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.

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

M3T-MR32R V.3.50

Reference Manual

Real-time OS for M32R Family

Renesas Electronics
www.renesas.com

Rev.1.00 2003.06

• Microsoft, MS-DOS, Windows, and Windows NT are registered trademarks of Microsoft Corporation in the U.S. and other countries.

• 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

Contents

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 "mr32r.inc"
```

Calling by the assembly language

<< Argument >>

Explanation of system call parameters
Parameters are written as macro arguments.

Argument name Size Explanation

The size is indicated by the following symbols:

[−*] 1−byte data
[**] 2−byte data
[****] 4−byte data

<< Register setting >>

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

Register name	Contents after system call issuance
*1	*2

*1 Register name. Written in this column are R0,R1,R2,R3.R4.R5.R6
*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.

PSW is such that the values of SM, IE, and C before a system call are saved; BSM, BIC and BC are indeterminate.

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

R0 register (32 bits)	Function code and Error code
R1 register (32 bits)	ID number of object
R2 register (32 bits)	Packet address, other parameters
R3 register (32 bits)	None of the above (wfmode, blksz, etc.)
R4 register (32 bits)	Time out value
R5 register (32 bits)	The start address of message
R6 register (32 bits)	Rendezvous bits pattern

[(Calling by the C language)]

Calling an MR32R 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 "mr32r.h"The definitions are as Appendix.

[(Error codes)]

Error code name	Error code value	The meaning of Error code

Error code character strings such as E_OK are defined in "mr32r.h" by using "#define" and in "mr32r.inc" by using ".EQU" To determine errors, use these defined character strings.[1]

[(Function description)]

```
Detail functional description
```

[(Usage example)]

```
Usage example
```

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

1.2. Necessary Stack Size

Table 1.1 lists the stack sizes (system stack) used by system calls that can be issued from tasks. If the system call issued from task, system uses user stack. If the system call issued from handler, system uses system stack.

() means the stack size when using DCC/M32R. * means it uses user stack.

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

System call	System call processing		C language I/F	
	CC32R	TW32R DCC/M32R	CC32R	TW32R DCC/M32R
cre_tsk	60	76(64)	4	4
del_tsk	28	44(44)	4	4
sta_tsk	0	0	4	4
ext_tsk	0	0	0	0
exd_tsk	28	44(44)	0	0
ter_tsk	0	0	4	4
dis_dsp	0	0	4	4
ena_dsp	0	0	4	4
chg_pri	0	0	4	4
rot_rdq	0	0	4	4
rel_wai	0	0	4	4
sus_tsk	0	0	4	4
rsm_tsk	0	0	4	4
slp_tsk	0	0	4	4
tslp_tsk	0	0	4	4
wup_tsk	0	0	4	4
cre_flg	0	0	4	4
del_flg	0	0	4	4
set_flg	0	0	4	4
wai_flg	0	0	8	8
twai_flg	0	0	8	8
cre_sem	0	0	4	4
del_sem	0	0	4	4
sig_sem	0	0	4	4
wai_sem	0	0	4	4
twai_sem	0	0	4	4
cre_mbx	60	76(64)	4	4
del_mbx	28	44(44)	4	4
snd_msg	0	0	4	4
rcv_msg	0	0	4	4
trcv_msg	0	0	4	4
cre_mbf	60	76(64)	4	4
del_mbf	28	44(44)	4	4
snd_mbf	0	0	4	4
tsnd_mbf	0	0	4	4
psnd_mbf	0	0	4	4
rcv_mbf	0	0	4	4
trcv_mbf	0	0	4	4
prcv_mbf	0	0	4	4

System call	System call processing		C language I/F	
	CC32R	TW32R DCC/M32R	CC32R	TW32R DCC/M32R
cre_por	0	0	4	4
del_por	0	0	4	4
cal_por	0	0	4	4
tcal_por	0	0	4	4
pcal_por	0	0	4	4
acp_por	0	0	8	8
tacp_por	0	0	8	8
pacp_por	0	0	8	8
fwd_por	0	0	4	4
rpl_rdv	0	0	4	4
def_int	0	0	4	4
loc_cpu	0	0	4	4
unl_cpu	0	0	4	4
cre_mpf	60	76(64)	4	4
del_mpf	28	44(44)	4	4
get_blf	0	0	4	4
tget_blf	0	0	4	4
rel_blf	0	0	4	4
cre_mpl	68	84(72)	4	4
del_mpl	28	44(44)	4	4
get_blk	68	88(72)	4	4
tget_blk	68	88(72)	4	4
pget_blk	68	88(72)	4	4
rel_blk	20	20(32)	4	4
dly_tsk	0	0	4	4
def_cyc	0	0	4	4
def_exc	60	76(64)	4	4
vclr_ems	0	0	4	4
vset_ems	0	0	4	4
vras_fex	0	0	4	4
vrst_blf	0	0	4	4
vrst_blk	40	20(32)	4	4
vrst_mbf	0	0	4	4
vrst_msg	0	0	4	4
vcre_mbx	0	0	4	4
vdel_mbx	0	0	4	4
vsnd_mbx	0	0	4	4
vrcv_mbx	0	0	4	4
vtrcv_mbx	0	0	4	4
vrst_mbx	*16	*16	4	4
vret_exc	0	0	4	4

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	System call processing		C language I/F	
	CC32R	TW32R DCC/M32R	CC32R	TW32R DCC/M32R
ista_tsk	24	24	4	4
ichg_pri	28	28	4	4
irot_rdq	32	32	4	4
irel_wai	32	32	4	4
isus_tsk	28	28	4	4
irsm_tsk	24	24	4	4
iwup_tsk	32	32	4	4
iset_flg	44	44	4	4
isig_sem	36	36	4	4
isnd_msg	36	36	4	4
ret_int	0	0	0	0
irel_blf	32	32	4	4
visnd_mbx	0	0	4	4

Table 1.3 lists the stack sizes (system stack) used by system calls that can be issued from both tasks and handlers.

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

System call	System call processing		C language I/F	
	CC32R	TW32R DCC/M32R	CC32R	TW32R DCC/M32R
get_tid	16	16	4	4
ref_tsk	16	16	4	4
can_wup	20	20	4	4
clr_flg	16	16	4	4
pol_flg	20	20	4	4
ref_flg	16	16	4	4
preq_sem	16	16	4	4
ref_sem	20	20	4	4
prcv_msg	28	28	4	4
ref_mbx	20	20	4	4
ref_mbf	20	20	4	4
ref_por	16	16	4	4
pget_blf	24	24	4	4
ref_mpf	28	28	4	4
ref_mpl	16	16	4	4
set_tim	16	16	4	4
get_tim	16	16	4	4
act_cyc	20	20	4	4
ref_cyc	20	20	4	4
ref_alm	28	28	4	4
get_ver	28	28	4	4
ref_sys	16	16	4	4
vrst_msg	16	16	4	4
vprcv_mbx	24	24	4	4
vref_mbx	28	28	4	4

1.3. Stack Size Calculation Method

The MR32R 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 MR32R requires to allocate the stack area for each stack.

- System stack

  This stack is used inside the MR32R or during the execution of the handler.

  When a task issues an system call, the MR32R switches the user stack to the system stack. The system stack uses interrupt stack.

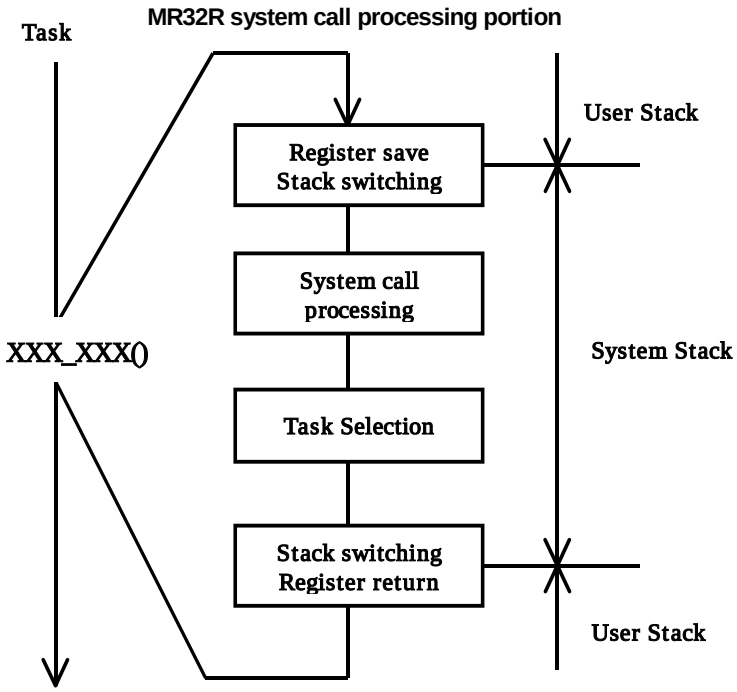

Figure 1.1 System Stack and User Stack

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

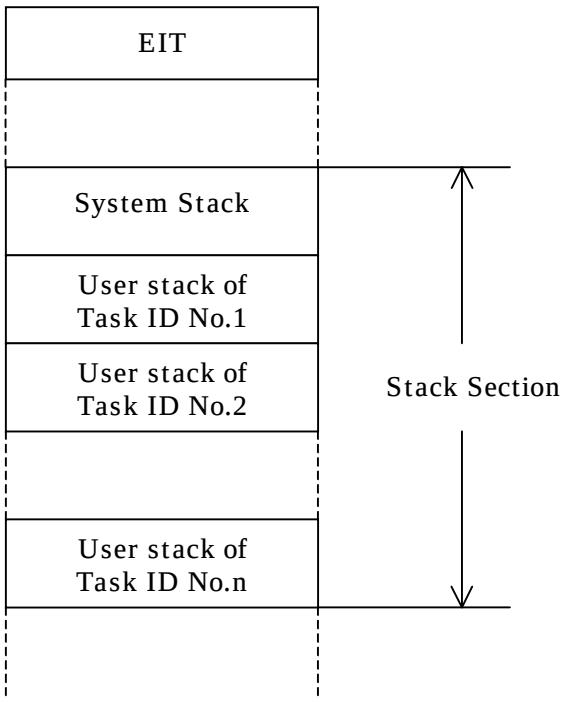

Figure 1.2 Layout of Stacks

1.3.1. 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

  For an application written in C, you can obtain user stack size in line with the way given below.

  1. The stack size that tasks use
  2. The stack size that the interface routines of C use
  3. The stack size consumed by issuing system calls

  In using MR32R, secure 80 bytes if you issue the only system calls that can be issued by tasks.

  If you issue system calls that can be issued by both tasks and handlers, secure a stack size by reference to the stack sizes shown in Table 1.3.

  With two or more system calls issued, calculate that the maximum of the stack sizes consumed by these system calls amounts to the size the MR32R uses.

  The sum of the three sizes - 1, 2, and 3 above - becomes the user stack size.

- When an application is written in the assembly language

  1. The stack size that the user program uses
     Obtain a size used to save registers in the stack
  2. The stack size consumed by issuing system calls

  In using MR32R, secure 80 bytes if you issue the only system calls that can be issued by tasks.

  If you issue system calls that can be issued by both tasks and handlers, secure a stack size by reference to the stack sizes shown in Table 1.3 .

  With two or more system calls issued, calculate that the maximum of the stack sizes consumed by these system calls amounts to the size the MR32R uses.

The sum of the two sizes - 1 and 2 above - becomes the user stack size.

Figure 1.3 shows an example of calculating a user stack.

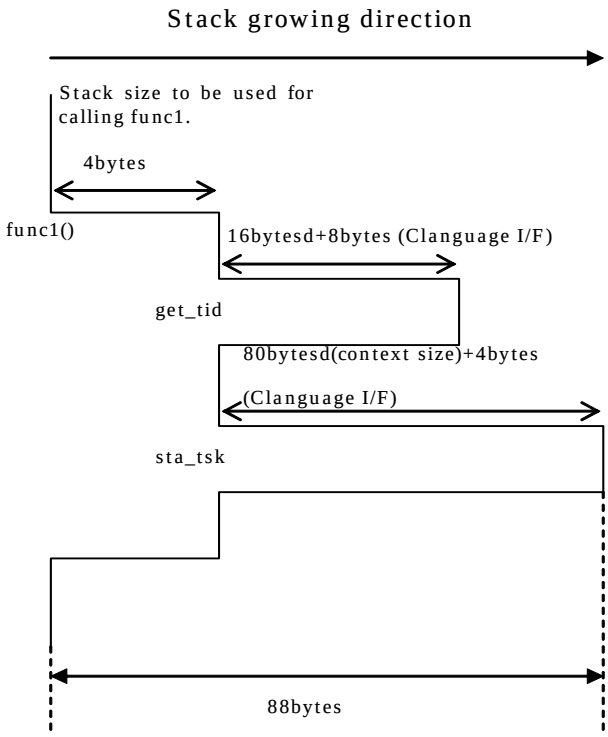

Figure 1.3 Example of User Stack Size Calculation

1.3.2. 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.[2] The necessary size (the maximum size) of the system stack can be obtained from the following relation:

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

- α

  The maximum system stack size among the system calls to be used.[3]

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

System call	System Stack Size
cre_tsk	60 bytes
sta_tsk	0 bytes
ext_tsk	0 bytes
slp_tsk	0 bytes

  Therefore,the maximum system stack size among the system calls to be used is the 60 bytes of cre_tsk.

- βi

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

- γ

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

[2] After switchover from user stack to system stack
[3] Refer Section 1.2 for the system stack size used for each individual system call.

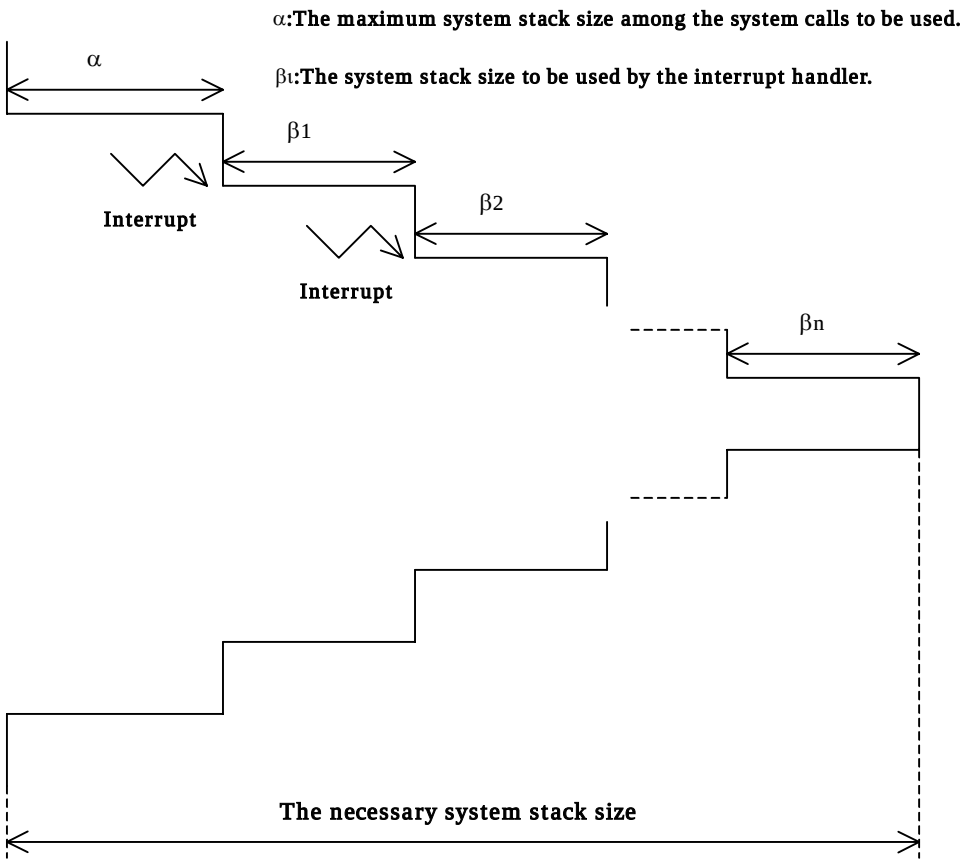

Figure 1.4 System Stack Calculation Method

[(Stack size βi used by interrupt handlers)]

The stack size used by an interrupt handler that is invoked during a system call is the sum of the following 3 sizes.

- The context save area
- The maximum stack extent the subroutines use when called by the interrupthandling routine[4].
- the extent user programs use.

You can calculate the extent of used stack in line with the manner given above regardless of whether you write your programs in C or in assembly language.

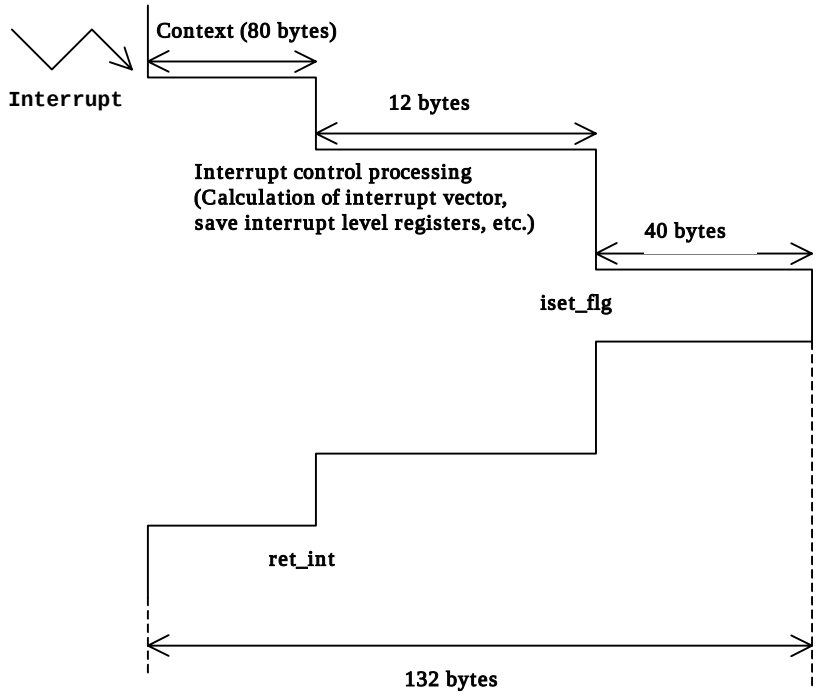

Table 1.5 Stack size to be used by Interrupt Handler

[4] The stack size used to save registers' contents by use of __RESTORE_IPL_from_STACK, or __SAVE_IPL_to_STACK. For details of these interrupt control programs, see How to Prepare Interrupt Control Programs given in User's Manual.

[(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:

92+ either the stack size the cyclic hander uses or the stack size the alarm handler uses, whichever is greater

If neither cyclic handler nor alarm handler is used, then

γ **= 84 bytes**

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

Chapter 2 **System Call Reference**

2.1. Task Management Functions

2.1.1. cre_tsk (Create Task)

[(System call name)]

cre_tsk → Create Task

[(Calling by the assembly language)]

```
.include “mr32r.inc”
cre_tsk     tskid
```

<< Argument >>

tskid	[**]	The ID No. of a task to be created
pk_ctsk	[****]	The start address in which the task generation information is stored (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a task to be created
R2	The start address in which the task generation information is stored
R3	--

Specify the following information in the structure indicased by pk_ctsk.

Offset	Size		
+0	4	exinf	Extended information
+4	4	tskatr	Task attribute
+8	4	task	Task startup address
+12	2	itskpri	Priority in task startup
+16	4	stksz	Stack size

[(Calling by the C language)]

```
#include <mr32r.h>
ER cre_tsk (tskid, pk_ctsk);
```

<< Argument >>

ID	tskid;	The ID No. of a task to be created
T_CTSK	*pk_ctsk;	The start address in which the task generation information is stored

Specify the following information in the structure indicased by pk_ctsk.

```
typedef struct t_ctsk {
    VP      exinf;          /* Extended information */
    ATR     tskatr;         /* Task attribute */
    FP      task;           /* Task startup address */
    PRI     itskpri;        /* Priority in task startup */
    INT     stksz;          /* Stack size */
} T_CTSK;
```

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
E_NOMEM         0FFFFFFF6H(-H’0000000a): Not enough of memory
E_OBJ           0FFFFFFC1H(-H’0000003f): Invalid object state
```

[(Function description)]

Creates a task tskid indicates.

That is, cre_tsk moves a task from the NON-EXISTENT state to the DORMANT state.

After having set the information as to the task to be generated, issues this system call to generate a task.

Here follows explanation of the information as to a task to be generated pk_ctsk.

- exinf (extended information)

  Exinf is an area you can freely use to store information as to a task to be generated. MR32R has nothing to do with the exinf's contents.

- tskatr (task attribute)

  Specify the location of the task stack area to be created. Specifically this means specifying whether you want the stack to be located in the internal RAM or in external RAM.

  - **To locate the stack area in internal RAM**
    Specify __MR_INT(0).
  - **To locate the stack area in external RAM**
    Specify __MR_EXT(0x10000).
  - **To locate the stack area user specified**
    Specify __MR_USER(0x20000).

- task (task start address)

  Task is an area to specify the start address of a task to be generated, so you have to invariably specify this.

  In writing a program in C, you have to make a prototype declaration on a task (function) to be generated.

- itskpri (priority in task start)

  itskpri is an area to specify a priority when a task to be generated is started up, so you have to invariably specify this.

- stksz (stack size)

  stksz is an area to specify a stack size a task to be generated uses, so you have to invariably specify this.

The system call cre_tsk is effective only when the specified task is in the NON-EXISTENT state. Issuing this system call toward a task in a different state causes MR32R to return the error code E_OBJ.

ID numbers to be generated are brought under your management. The numbers to be specified by this system call can range from 1 up to the maximum number of tasks used in the user system laid down in the system definition.

If the extent of memory as specified by stksz under pk_ctsk is not available, MR32R returns the error code E_NOMEM to the task that issued this system call.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
#define ID_task2   2
#define ID_task3   3
void task2(void);
void task3(void);
void task1(void)
{
    T_CTSK ctsk2;
    T_CTSK ctsk3 = {0,__MR_INT,task3,2,200};
    ctsk2.tskatr = __MR_EXT;    /* To locate the stack area in external RAM */
    ctsk2.task = task2;         /* Task startup address */
    ctsk2.itskpri = 2;          /* Priority in task startup */
    ctsk2.stksz = 100;          /* Stack size */
    cre_tsk( ID_task2, &ctsk2 );
         :
}
void task2(void)
{
       :
    ext_tsk();
}
void task3(void)
{
       :
    ext_tsk();
}
<< Usage example of the assembly language(CC32R) >>

ID_task2:       .equ    2
ID_task3:       .equ    3
ctsk2:
    .DATA.W                     ; Extended information
    .DATA.W    __MR_INT         ; Task attribute
    .DATA.W    task2            ; Task startup address
    .DATA.H       2             ; Priority in task startup
    .RES.B     2
    .WORD     100               ; Stack size

    .include “mr32r.inc”
    .global   task1,task2
task1:
    cre_tsk ID_task2, ctsk2
          :
    ext_tsk
task2:
       :
    ext_tsk
```

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

```
    .equ ID_task2,2
    .equ ID_task3,3
ctsk2:
    .LONG    0
    .LONG    __MR_INT
    .LONG    task2
    .SHORT   2
    .space   2
    .LONG    100

    .include “mr32r.inc”
    .global   task1,task2
task1:
    cre_tsk ID_task2, ctsk2
          :
    ext_tsk
task2:
       :
    ext_tsk
```

2.1.2. del_tsk(Delete Task)

[(System call name)]

del_tsk → Delete Task

[(Calling by the assembly language)]

```
.include “mr32r.inc”
del_tsk     tskid
```

<< Argument >>

tskid [**] The ID No. of a task to be deleted

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a task to be deleted
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER del_tsk (tskid);
```

<< Argument >>

ID tskid; The ID No. of a task to be deleted

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_OBJ         0FFFFFFC1H(-H’0000003f): Invalid object state
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

del_tsk deletes the task tskid indicates.

This system call cannot specify the task itself.

Issuing this system call causes the state of the task under consideration to switch from the DORMANT state to the NON-EXISTENT state.

The system call del_tsk is effective only when the specified task is in the DORMANT state. Issuing this system call toward a task in a different state causes MR32R to return the error code E_OBJ.

Error E_NOEXS is returned if this system call is issued for a NON-EXISTENT state task.

Make sure this system call is issued for only the task that has been created by the cre_tsk system call. If this system call is issued for the task that has been defined by the configuration file, it does not function normally.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
#define ID_task2   2
#define ID_task3   3
void task1()
{
        :
    del_tsk( ID_task2 );
        :
}
void task2()
{
        :
    ext_tsk();
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task1,task2
ID_task2:       .equ    2
ID_task3:       .equ  3
task1:
          :
    del_tsk ID_task2
          :
    ext_tsk
task2:
          :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task1,task2
    .equ   ID_task2,2
    .equ   ID_task3,3
task1:
          :
    del_tsk ID_task2
          :
    ext_tsk
task2:
          :
    ext_tsk
```

2.1.3. sta_tsk(Start Task)

[(System call name)]

sta_tsk → Starts the Task

[(Calling by the assembly language)]

```
.include “mr32r.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	The ID No. of the task to be started
R2	Task start code
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
E_OBJ         0FFFFFFC1H(-H’0000003f): 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 32 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 R2 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)[5], an error E_OBJ is returned to the system call issued task.

Error E_NOEXS is returned if this system call is issued for a NON-EXISTENT state task.

If a task is reactivated after being terminated by ter_tsk or ext_tsk, it starts under the following conditions:[6]

- The task starts from the start address set in the configuration file or when cre_tsk system

[5] except NON-EXISTENT state.
[6] Namely,the task totally starts from the reset state.

call is issued.

- The wakeup request count is cleared to 0.
- The priority is the initial priority specified in the configuration file or when cre_tsk system call is issued.
- The initial register values except PC, PSW and following registers are indeterminate.
  - □ For M32R Family Cross Tool **CC32R**
    A start code is stored R2 and R4 register.
  - □ For M32R Family GNU Cross Tool **TW32R:DCC/M32R**
    A start code is stored R0 and R2 register.

If the task restarts,its exception handler defined before is not reset.

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.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    sta_tsk( ID_task2, stacd );
          :
}
void task2(int msg)
{
    if(msg == 0)
           :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task,task2
task:
    sta_tsk ID_task2, msg
          :
task2:
    cmpi      R2,#0
          :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task,task2
task:
    sta_tsk ID_task2, msg
          :
task2:
    cmpi      R2,#0
          :
```

2.1.4. ista_tsk(Start Task)

[(System call name)]

ista_tsk → Starts the Task. (for the handler only)

[(Calling by the assembly language)]

```
.include “mr32r.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	The ID No. of the task to be started
R2	Task start code
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK           00000000H(-H’00000000): Normal End
E_NOEXS        0FFFFFFCCH(-H’00000034): Object does not exist
E_OBJ          0FFFFFFC1H(-H’0000003f): 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 <mr32r.h>
#include “id.h”
void inthand()
{
    ista_tsk( ID_task2, stacd );
           :
}
void task2(int msg)
{
    if(msg == 0)
           :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  intr
intr:
    ista_tsk ID_task2, msg
           :
    ret_int

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  intr
intr:
    ista_tsk ID_task2, msg
           :
    ret_int
```

2.1.5. ext_tsk(Exit Task)

[(System call name)]

ext_tsk → Ends the own task.

[(Calling by the assembly language)]

```
.include “mr32r.inc”
ext_tsk
```

<< Argument >>

None

<< Register setting >>

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

[(Calling by the C language)]

```
#include <mr32r.h>
void ext_tsk ();
```

<< Argument >>

None

<< Return value >>

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.

If this system call issued from exception handler,the task for it is normaly ended.

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 <mr32r.h>
void task(void)
{
       :
    ext_tsk();
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global task
task:
       :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global task
task:
       :
    ext_tsk
```

2.1.6. exd_tsk(Exit and Delete Task)

[(System call name)]

exd_tsk → Exit and delete Task.

[(Calling by the assembly language)]

```
.include “mr32r.inc”
exd_tsk
```

<< Argument >>

None

<< Register setting >>

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

[(Calling by the C language)]

```
#include <mr32r.h>
void exd_tsk ();
```

<< Argument >>

None

<< Return value >>

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 and deletes it; that is, it puts the own task from the RUN state to the NON-EXISTENT state. Once a task has been deleted, it does not operate until activated again by the cre_tsk, sta_tsk or ista_tsk system call.

When this system call is issued, the semaphore, memorypool etc. obtained by the own task is not freed, but the the stack area of the own task is freed.

This system call can only be issued from the task created by cre_tsk system call.

If exd_tsk is issued from the task defined in the configuration file,it does not work well..

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 <mr32r.h>
#include “id.h”
void task1()
{
          :
    cre_tsk( ID_task2, &ctsk2 );
          :
    sta_tsk( ID_task2, 0 );
          :
}
void task2()
{
          :
    exd_tsk();
}

<< Usage example of the assembly language(CC32R) >>

    .include  “mr32r.inc”
    .global   task1,task2
task1:
          :
    cre_tsk 2,settask2
          :
    sta_tsk 2,0
    ext_tsk
task2:
          :
    exd_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include  “mr32r.inc”
    .global   task1,task2
task1:
          :
    cre_tsk 2,settask2
          :
    sta_tsk 2,0
    ext_tsk
task2:
          :
    exd_tsk
```

2.1.7. ter_tsk(Terminate Task)

[(System call name)]

ter_tsk → Terminates a task forcibly.

[(Calling by the assembly language)]

```
.include "mr32r.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	The ID No. of the task to be forcibly terminated
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK           00000000H(-H'00000000): Normal End
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
E_OBJ         0FFFFFFC1H(-H'0000003f): 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[7] 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 NON-EXISTENT state, the system returns an error E_OBJ for the system call.

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

[7] 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 <mr32r.h>
#include “id.h”
void task()
{
          :
    ter_tsk( ID_main );
          :
    ext_tsk();
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task
task:
          :
    ter_tsk ID_task2
          :

    .global  task2
task2:
          :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task
task:
          :
    ter_tsk ID_task2
          :

    .global  task2
task2:
          :
```

2.1.8. dis_dsp(Disable Dispatch)

[(System call name)]

dis_dsp → Disable dispatch of the task.

[(Calling by the assembly language)]

```
.include “mr32r.inc”
dis_dsp
```

<< Argument >>

None

<< Register setting >>

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

[(Calling by the C language)]

```
#include <mr32r.h>
ER dis_dsp ();
```

<< Argument >>

None

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H’00000000): 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.

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 <mr32r.h>
#include “id.h”
void task()
{
          :
    dis_dsp();
          :
}

<< Usage example of the assembly language(CC32R) >>

.include “mr32r.inc”
    .global task
task:
       :
    dis_dsp
       :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global task
task:
       :
    dis_dsp
       :
```

2.1.9. ena_dsp(Enable Dispatch)

[(System call name)]

ena_dsp → Permits dispatch of the task.

[(Calling by the assembly language)]

```
.include “mr32r.inc”
ena_dsp
```

<< Argument >>

None

<< Register setting >>

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

[(Calling by the C language)]

```
#include <mr32r.h>
ER ena_dsp();
```

<< Argument >>

None

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H’00000000): 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 <mr32r.h>
#include “id.h”
void task()
{
        :
    ena_dsp();
        :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task
task:
        :
    ena_dsp
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task
task:
        :
    ena_dsp
        :
```

2.1.10. chg_pri(Change Task Priority)

[(System call name)]

chg_pri → Changes the priority of a task.

[(Calling by the assembly language)]

```
.include “mr32r.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	The ID No. of the task whose priority is changed
R2	The priority to be changed
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK           00000000H(-H’00000000): Normal End
E_NOEXS        0FFFFFFCCH(-H’00000034): Object does not exist
E_OBJ          0FFFFFFC1H(-H’0000003f): 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. 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 [8]

```
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. If it is in NON-EXISYENT state, the system returns an error E_NOEXS for the system call.

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.[9]

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 <mr32r.h>
#include “id.h”
void task()
{
          :
    chg_pri( ID_task2, 2 );
          :
}
<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task
task:
          :
    chg_pri   ID_task2,2
          :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task
task:
          :
    chg_pri   ID_task2,2
          :
```

[8] 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.

[9] 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.

2.1.11. ichg_pri(Change Task Priority)

[(System call name)]

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

[(Calling by the assembly language)]

```
.include “mr32r.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	The ID No. of the task whose priority is changed
R2	The priority to be changed
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
E_OBJ         0FFFFFFC1H(-H’0000003f): 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 <mr32r.h>
#include “id.h”
void inthand()
{
            :
    ichg_pri( ID_main, 2 );
            :
}

<< Usage example of the assembly language(CC32R) >>

.include “mr32r.inc”
.global  intr
intr:
            :
    ichg_pri   ID_task2, 2
            :
    ret_int

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

.include “mr32r.inc”
.global  intr
intr:
            :
    ichg_pri   ID_task2, 2
            :
    ret_int
```

2.1.12. rot_rdq(Rotate Ready Queue)

[(System call name)]

rot_rdq → Rotates the ready queue of a task.

[(Calling by the assembly language)]

```
.include “mr32r.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
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK            00000000H(-H’00000000): 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.

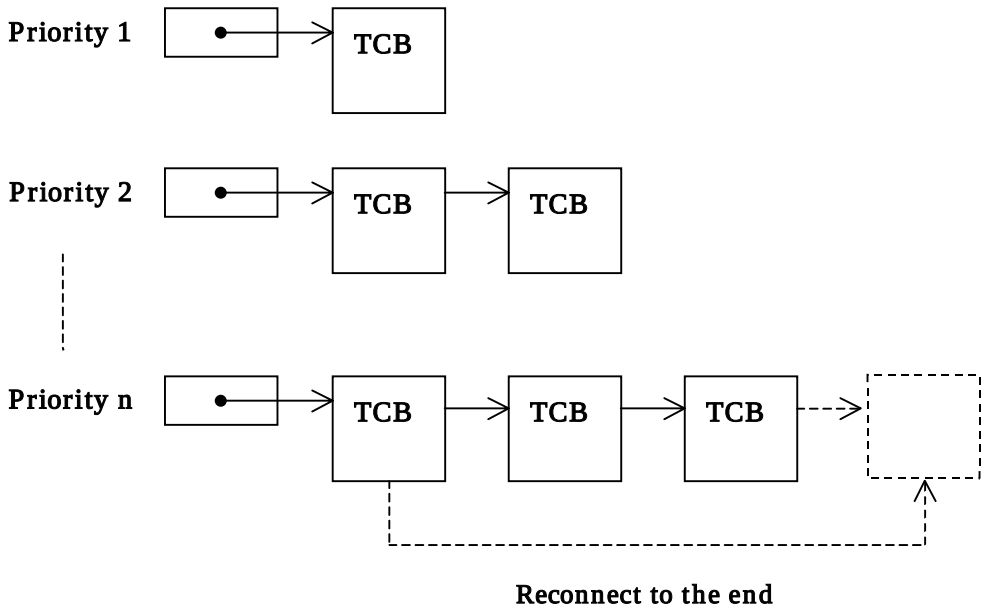

Figure 2.1 Ready Queue Operation by rot_rdq System Call

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. If there is no task on the queue specified by this system call, the system do nothing.

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 <mr32r.h>
void task()
{
          :
    rot_rdq( 2 );
          :
}

<< Usage example of the assembly language(CC32R) >>

.include “mr32r.inc”
    .global   task
task:
          :
    rot_rdq    2
          :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global   task
task:
          :
    rot_rdq    2
          :
```

2.1.13. irot_rdq(Rotate Ready Queue)

[(System call name)]

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

[(Calling by the assembly language)]

```
.include “mr32r.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
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK            00000000H(-H’00000000): 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 <mr32r.h>
#include “id.h”
void cyc()
{
        :
    irot_rdq( 2 );
        :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  cyc
cyc:
        :
    irot_rdq  2
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

.include “mr32r.inc”
    .global  cyc
cyc:
        :
    irot_rdq  2
        :
```

2.1.14. rel_wai(Release Task Wait)

[(System call name)]

rel_wai → Releases the task WAIT state forcibly.

[(Calling by the assembly language)]

```
.include "mr32r.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	The ID No. of the task to be forcibly released from the WAIT state
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK           00000000H(-H'00000000): Normal End
E_NOEXS        0FFFFFFCCH(-H'00000034): Object does not exist
E_OBJ          0FFFFFFC1H(-H'0000003f): 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 [10] 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. If the task is in NON-EXISTENT state, the system returns an error E_NOEXS 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.

[10] 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 <mr32r.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(CC32R) >>

    .include “mr32r.inc”
    .global   task
task:
        :
    rel_wai    ID_main
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global   task
task:
        :
    rel_wai    ID_main
        :
```

2.1.15. irel_wai(Release Task Wait)

[(System call name)]

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

[(Calling by the assembly language)]

```
.include “mr32r.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	The ID No. of the task to be forcibly released from the WAIT state
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK           00000000H(-H’00000000): Normal End
E_NOEXS        0FFFFFFCCH(-H’00000034): Object does not exist
E_OBJ          0FFFFFFC1H(-H’0000003f): 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 <mr32r.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(CC32R) >>

.include “mr32r.inc”
    .global   intr
intr:
        :
    irel_wai ID_main
        :
    ret_int

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global   intr
intr:
        :
    irel_wai ID_main
        :
    ret_int
```

2.1.16. get_tid(Get Self Task ID)

[(System call name)]

get_tid → Gets the ID of the self task

[(Calling by the assembly language)]

```
.include “mr32r.inc”
get_tid
```

<< Argument >>

None

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID of the self task
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK            00000000H(-H’00000000): 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 <mr32r.h>
void task()
{
    ID tskid;
       :
    get_tid(&tskid);
       :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task
task:
       :
    get_tid
       :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task
task:
       :
    get_tid
       :
```

2.1.17. ref_tsk(Refer Task Status)

[(System call name)]

ref_tsk → Reference Task Status

[(Calling by the assembly language)]

```
.include "mr32r.inc"
ref_tsk   tskid
```

<< Argument >>

taskid	[**]	The ID No. of the task to Reference Task
pk_rtsk	[****]	Packet address to Reference Task (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the task to Reference Task
R2	Packet address to Reference Task
R3	--

The area indicated by pk_rtsk returns the following information.

Offset	Size		
+0	4	exinf	Extended information
+4	2	tskpri	Current task priority level
+8	4(U)	tskstat	Task status
+12	4(U)	tskwait	Reason for wait
+16	2	wid	Wait object ID
+20	4	wupcnt	Number of queued wakeup requests
+24	4	tskatr	Task attributes
+28	4	task	Task starting address
+32	2	tskpri	Initial task priority
+36	4	stksz	Stack size
+40	4(U)	epndptn	Pending exception class pattern

U: unsigned data.

[(Calling by the C language)]

```
#include <mr32r.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 */
    PR      tskpri;         /* Current task priority level */
    UH      tskstat;        /* Task status */
    UINT    tskwait;        /* Reason for wait */
    ID      wid;            /* Wait object ID */
    INT     wupcnt;         /* Number of queued wakeup requests */
    ATR     tskatr;         /* Task attributes */
    FP      task;           /* Task starting address */
    PRI     itskpri;        /* Initial task priority */
```

```
    INT     stksz;          /* Stack size */
    UW      epndptn;        /* Pending exception class pattern */
} T_RTSK;
```

[(Error codes)]

```
E_OK            00000000H(-H'00000000): Normal End
E_NOEXS         0FFFFFFCCH(-H'00000034): Object does not exist
```

[(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

- tskpri

  Returns the task priority level in tskpri

- tskstat

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

TTS_RUN	(00000001H)	RUN state
TTS_RDY	(00000002H)	READY state
TTS_WAI	(00000004H)	WAIT state
TTS_SUS	(00000008H)	SUSPEND state
TTS_WAS	(0000000CH)	WAIT-SUSPEND state
TTS_DMT	(00000010H)	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	(00000001H)	Waiting with slp_tsk or tslp_tsk
TTW_DLY	(00000002H)	Waiting with dly_tsk
TTW_FLG	(00000010H)	Waiting with wai_flg or twai_flg
TTW_SEM	(00000020H)	Waiting with wai_sem or twai_sem
TTW_MBX	(00000040H)	Waiting with rcv_msg or trcv_msg
TTW_SMBF	(00000080H)	Waiting with snd_mbf or tsnd_mbf
TTW_MBF	(00000100H)	Waiting with rcv_mbf or trcv_mbf
TTW_CAL	(00000200H)	Waiting with cal_pol or tcal_pol
TTW_ACP	(00000400H)	Waiting with acp_pol or tacp_pol
TTW_RDV	(00000800H)	Waiting with Rendezvous
TTW_MPL	(00001000H)	Waiting with get_blk or tget_blk
TTW_MPF	(00002000H)	Waiting with get_blf or tget_blf
TTW_VMBX	(00004000H)	Waiting with vrcv_mbx or vtrcv_mbx

- wid

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

- wupcnt

- tskatr

  Returns the attribute of the task.It means whther the stack area of the task is internal RAM(__MR_INT=0) or external RAM(MR_EXT=0x10000).

- task

  Returns the entry address of the task.

- itskpri

  Returns the priority of the task.

- stksz

  Returns the stack size of the task.

- epndptn

  Returns the pending pattern. It means the information of exception mask and the information of exception pending.[11]

epndptn		mean
EXM_SET	00000001H	exception mask is set
EXP_TER	00000002H	forced end request is pending
EXP_FEX	00000004H	forced exception request is pending

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. If the task is in NON-EXISTENT state, the system returns an error E_NOEXS 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 <mr32r.h>
#include “id.h”
void task()
{
    T_RTSK rtsk;
          :
    ref_tsk( &rtsk, ID_main );
          :
}

<< Usage example of the assembly language(CC32R) >>

    .include  “mr32r.inc”
    .global task
task:
    ld24      R2,#rtsk
    ref_tsk   ID_task2
         :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include  “mr32r.inc”
    .global task
task:
    ld24      R2,#rtsk
    ref_tsk   ID_task2
         :
```

[11] If you specify “YES” as exc_handler in configuration file, indeterminate value is returned.

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 assembly language)]

```
.include “mr32r.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	The ID No. of the task to be put in the SUSPEND state
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_QOVR        0FFFFFFB7H(-H’00000049): Queuing or nest overflow
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
E_OBJ         0FFFFFFC1H(-H’0000003f): 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. If the task is in NON-EXISTENT state, the system returns an error E_NOEXS to the system call issued 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 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 isus_tsk.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.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(CC32R) >>

    .include “mr32r.inc”
    .global   task
task:
        :
    sus_tsk    ID_task2
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global   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 assembly language)]

```
.include "mr32r.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	The ID No. of the task to be put in the SUSPEND state
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_QOVR        0FFFFFFB7H(-H'00000049): Queuing or nest overflow
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
E_OBJ         0FFFFFFC1H(-H'0000003f): 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 <mr32r.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(CC32R) >>

    .include “mr32r.inc”
    .global  intr
intr:
        :
    isus_tsk   ID_main
        :
    ret_int

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  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 assembly language)]

```
.include “mr32r.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	The ID No. of the task to be taken from the SUSPEND state
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK           00000000H(-H’00000000): Normal End
E_NOEXS        0FFFFFFCCH(-H’00000034): Object does not exist
E_OBJ          0FFFFFFC1H(-H’0000003f): 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. If the task is in NON-EXISTENT state, the system returns an error E_NOEXS to the system call issued task.

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 <mr32r.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(CC32R) >>

    .include “mr32r.inc”
    .global  task
task:
        :
    rsm_tsk   ID_task2
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  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 assembly language)]

```
.include "mr32r.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	The ID No. of the task to be taken from the SUSPEND state
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK           00000000H(-H'00000000): Normal End
E_NOEXS        0FFFFFFCCH(-H'00000034): Object does not exist
E_OBJ          0FFFFFFC1H(-H'0000003f): 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 <mr32r.h>
#include “id.h”
void inthand()
{
        :
    irsm_tsk( ID_main );
        :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  intr
intr:
              :
    irsm_tsk   ID_main
              :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  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 assembly language)]

```
.include “mr32r.inc”
slp_tsk
```

<< Argument >>

None

<< Register setting >>

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

[(Calling by the C language)]

```
#include <mr32r.h>
ER slp_tsk ();
```

<< Argument >>

None

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_RLWAI      0FFFFFFAAH(-H’00000056): 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[12] or the system call which forcibly clears the WAIT state. [13] 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 only be issued from tasks, and cannot be issued from the interrupt handler, cyclic handler, or alarm handler.

[12] wup_tsk,iwup_tsk System call
[13] rel_wai,irel_wai System call

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
                :
    if( slp_tsk() != E_OK )
        error(“Forced wakeup\n”);
                :
}

<< Usage example of the assembly language(CC32R) >>

    .INCLUDE “mr32r.inc”
    .GLOBAL   task
task:
        :
    slp_tsk
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .INCLUDE “mr32r.inc”
    .GLOBAL   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 assembly language)]

```
.include “mr32r.inc”
tslp_tsk   tmout
```

<< Argument >>

tmout [****] Timeout value

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
R3	--
R4	Timeout value

[(Calling by the C language)]

```
#include <mr32r.h>
ER tslp_tsk (tmout);
```

<< Argument >>

TMO tmout; Timeout value

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_TMOUT       0FFFFFFABH(-H’00000055): Polling failed or timeout
E_RLWAI       0FFFFFFAAH(-H’00000056): 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[14] to start a task is invoked from another task or interrupt.

  Error code E_OK is returned.

- When a system call[15] to forcibly cancel the wait state is invoked from another task or interrupt.

  Error code E_RLWAI is returned.

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

  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 0x7FFFFFF. 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 only be issued from tasks, and cannot be issued from the interrupt handler, cyclic handler, or alarm handler.

[14] wup_tsk, iwup_tsk System call
[15] rel_wai, irel_wai System call

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
                :
    if( tslp_tsk( 10 ) != E_TMOUT )
        printf(“Forced wakeup\n”);
                :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global task
task:
        :
    tslp_tsk   200
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global task
task:
        :
    tslp_tsk   200
        :
```

2.2.7. wup_tsk(Wakeup Task)

[(System call name)]

wup_tsk → Wakes up the task in the wait state.

[(Calling by the assembly language)]

```
.include “mr32r.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	The ID No. of the task to be waked up
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_QOVR        0FFFFFFB7H(-H’00000049): Queuing or nest overflow
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
E_OBJ         0FFFFFFC1H(-H’0000003f): 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. If the task is in NON-EXISTENT state, the system returns an error E_NOEXS 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[16] of the task is incremented by 1[17]

The maximum value of the wakeup request count is 0x7FFFFFFF. If a wakeup request is issued beyond 0x7FFFFFFF, the count remains 0x7FFFFFFF 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 cannot 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 <mr32r.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(CC32R) >>

    .include “mr32r.inc”
    .global  task
task:
        :
    wup_tsk   ID_task2
        :


<< Usage example of the assembly language(TW32R:DCC/M32R) >>
    .include “mr32r.inc”
    .global  task
task:
        :
    wup_tsk   ID_task2
        :
```

[16] Task Control Block.

[17] 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 assembly language)]

```
.include “mr32r.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	The ID No. of the task to be waked up
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_QOVR        0FFFFFFB7H(-H’00000049): Queuing or nest overflow
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
E_OBJ         0FFFFFFC1H(-H’0000003f): 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 <mr32r.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(CC32R) >>

    .include “mr32r.inc”
    .global  intr
intr:
        :
    iwup_tsk   ID_main
        :
    ret_int

    << Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  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 assembly language)]

```
.include “mr32r.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	The ID No. of the task whose wakeup request is to be canceled
R2	The variable to store the count of canceled wakeup
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK           00000000H(-H’00000000): Normal End
E_NOEXS        0FFFFFFCCH(-H’00000034): Object does not exist
E_OBJ          0FFFFFFC1H(-H’0000003f): 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 request count, is returned.

For the request issued when the task whose wakeup request is to be canceled is in the DORMANT state, error code E_OBJ is returned to the task which issued this system call. If the task is in NON-EXISTENT state, the system returns an error E_NOEXS to the system call issued task.

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

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

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.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(CC32R) >>

    .include "mr32r.inc"
    .global  task
task:
        :
    can_wup   ID_task2
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include "mr32r.inc"
    .global  task
task:
        :
    can_wup   ID_task2
        :
```

2.3. Eventflags

2.3.1. cre_flg(Create EventFlag)

[(System call name)]

cre_tsk → Create Eventflag

[(Calling by the assembly language)]

```
.include “mr32r.inc”
cre_flg     flgid
```

<< Argument >>

flgid	[**]	The ID No. of an eventflag to be created
pk_cflg	[****]	The start address in which the eventflag generation information is stored (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of an eventflag to be created
R2	The start address in which the eventflag generation information is stored
R3	--

Specify the following information in the structure indicased by pk_cflg.

Offset	Size		
+0	4	exinf	Extended information
+4	4	flgatr	Eventflag attribute
+8	4(U)	iflgptn	Initial eventflag pattern

U: unsigned data.

[(Calling by the C language)]

```
#include <mr32r.h>
ER cre_flg (flgid, pk_cflg);
```

<< Argument >>

ID	flgid;	The ID No. of an eventflag to be created
T_CFLG	*pk_cflg;	The start address in which the eventflag generation information is stored

Specify the following information in the structure indicased by pk_cflg.

```
typedef struct t_cflg {
    VP      exinf;  /* Extended information */
    ATR     flgatr; /* Task attribute */
    UINT    iflgptn;        /* Initial eventflag pattern */
} T_CFLG;
```

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
E_OBJ           0FFFFFFC1H(-H’0000003f): Invalid object state
```

[(Function description)]

Creates an eventflag flgid indicates.

The created eventflag consists of 32bits bit-pattern and is initialized as the value of iflgptn.

Here follows explanation of the information as to an eventflag to be generated pk_cflg.

- exinf (extended information)

  Exinf is an area you can freely use to store information as to an eventflag to be generated. MR32R has nothing to do with the exinf's contents.

- flgatr (eventflag attribute)

  MR32R has nothing to do with this contents.

- iflgptn

  Set the initial bit-pattern of in this area when eventflag is created.

Error E_OBJ is returned if this system call is issued for a created eventflag.

The range of the specifiable ID number is 1 to the maximum value specified in the configuration file.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
#define ID_flg1 1
#define ID_flg2 2
void task1()
{
    T_CFLG cflg1;
    T_CFLG cflg2=-0,0,0xfff”;
                :
    cflg1.iflgptn = 0xff;
    cre_flg( ID_flg1, &cflg1 );
                :
}

<< Usage example of the assembly language(CC32R) >>

cflg1: .RES.B 12
cflg2: .RES.B 12
ID_flg1:        .equ    1
ID_flg2:        .equ    2
    .include “mr32r.inc”
    .global  task1
task1:
        :
    ld24    R2,#cflg1
    ld24    R1,#H’FF
    st      R1,@(8,R2)
    cre_flg ID_flg1
        :
    ld24    R2,#cflg2
    ld24    R1,#H’FFF
    st      R1,@(8,R2)
    cre_flg ID_flg2
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

cflg1: .space 12
cflg2: .space 12
    .equ   ID_flg1,1
    .equ   ID_flg2,2
    .include “mr32r.inc”
    .global  task1
task1:
        :
    ld24    R2,#cflg1
    ld24    R1,#0xFF
    st      R1,@(8,R2)
    cre_flg ID_flg1
        :
    ld24    R2,#cflg2
    ld24    R1,#0xFFF
    st      R1,@(8,R2)
    cre_flg ID_flg2
        :
    ext_tsk
```

2.3.2. del_flg(Delete EventFlag)

[(System call name)]

del_flg → Delete Eventflag

[(Calling by the assembly language)]

```
.include “mr32r.inc”
del_flg     flgid
```

<< Argument >>

flgid [**] The ID No. of an eventflag to be deleted

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of an eventflag to be deleted
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER del_flg ( flgid );
```

<< Argument >>

ID flgid; The ID No. of an eventflag to be deleted

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
E_NOEXS         0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

del_flg deletes the eventflag flgid indicates.

You can create the eventflag deleted as the same ID again.If the task is linked to the eventflag wait queue and del_flg is issued for the eventflag,this system call normally end.In this case,del_flg moves the task WAIT state to READY state.And an error E_DLT is returned.

An error E_NOEXS is returned if this system call is issued for a nonexistent eventflag.

Make sure this system call is issued for only the eventflag that has been created by the cre_flg system call. If this system call is issued for the eventflag that has defined by the configuration file, it does not function normally.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
#define ID_flg2 2
void task1()
{
        :
    del_flg( ID_flg2 );
        :
    ext_tsk();
}

<< Usage example of the assembly language(CC32R) >>

ID_flg2:        .equ    2
    .include “mr32r.inc”
    .global   task1
task1:
        :
    del_flg   ID_flg2
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .equ ID_flg2,2
    .include “mr32r.inc”
    .global   task1
task1:
        :
    del_flg   ID_flg2
        :
    ext_tsk
```

2.3.3. set_flg(Set EventFlag)

[(System call name)]

set_flg → Sets an eventflag.

[(Calling by the assembly language)]

```
.include "mr32r.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	The ID No. of the eventflag to be set
R2	The bit pattern to be set
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
```

[(Function description)]

Among the 32-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.After the eventflag value is changed,set_flg system call moves the task WAIT state to READY or RUN state if it's wait condition is matched.[18]

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.

An error E_NOEXS is returned if this system call is issued for a nonexistent eventflag.

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.

[18] Whether the task is moved to READY state or RUN state depends on the state of the ready queue.

[(Usage example)]

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

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task(void)
{
                :
    set_flg( ID_flg,(UINT)0xff00 );
                :
    ext_tsk();
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task
task:
        :
    set_flg   ID_flg,0x0ff00
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task
task:
        :
    set_flg   ID_flg,0x0ff00
        :
    ext_tsk
```

2.3.4. iset_flg(Set EventFlag)

[(System call name)]

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

[(Calling by the assembly language)]

```
.include “mr32r.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	The ID No. of the eventflag to be set
R2	The bit pattern to be set
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_NOEXS      0FFFFFFCCH(-H’00000034): Object does not exist
```

[(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 0xff, the pattern after this system call becomes 0xffff.

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void inthand(void)
{
        :
    iset_flg( ID_flg, (UINT)0xff00   );
        :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  intr
intr:
        :
    iset_flg   ID_flg,H’ff00
        :
    ret_int

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  intr
intr:
        :
    iset_flg   ID_flg,0x0ff00
        :
    ret_int
```

2.3.5. clr_flg(Clear EventFlag)

[(System call name)]

clr_flg → Clears an eventflag.

[(Calling by the assembly language)]

```
.include “mr32r.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	The ID No. of the eventflag to be cleared
R2	The bit pattern to be cleared
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
E_NOEXS        0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

Among the 32-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.

An error E_NOEXS is returned if this system call is issued for a nonexistent eventflag.

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

[(Usage example)]

If the eventflag pattern issuing this system call was 0xffff, the pattern after this system call becomes 0xff00.

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task(void)
{
        :
    clr_flg( ID_flg, (UINT)0xff00 );
        :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task
task:
        :
    clr_flg ID_flg,H’ff00
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task
task:
        :
    clr_flg ID_flg,0xff00
        :
```

2.3.6. wai_flg(Wait EventFlag)

[(System call name)]

wai_flg → Waits for an eventflag.

[(Calling by the assembly language)]

```
.include "mr32r.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	The ID No. of the eventflag to waited for
R2	The bit pattern when wait state is cleared
R3	Wait mode

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_RLWAI       0FFFFFFAAH(-H'00000056): Wait state forcibly
                                       cleared
E_DLT         0FFFFFFAFH(-H'00000051): The object being waited for
                                       was deleted
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
```

[(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, an 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 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 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.

If the task is linked to the eventflag wait queue and del_flg is issued for the eventflag, del_flg system call moves the task WAIT state to READY state.And error E_DLT is returned.

An error E_NOEXS is returned if this system call is issued for a nonexistent eventflag.

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)]

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 <mr32r.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(CC32R) >>

    .include “mr32r.inc”
    .global  task
task:
        :
    wai_flg   ID_flg2,H’ff0,(TWF_ANDW+TWF_CLR)
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task
task:
        :
    wai_flg   ID_flg2,0x0ff0,(TWF_ANDW+TWF_CLR)
        :
```

2.3.7. twai_flg(Wait EventFlag with Timeout)

[(System call name)]

twai_flg → Waits for an eventflag. (With Timeout)

[(Calling by the assembly language)]

```
.include "mr32r.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	The ID No. of the eventflag to be waited for
R2	The bit pattern when wait state is cleared
R3	Wait mode
R4	Timeout value

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_RLWAI       0FFFFFFAAH(-H'00000056): Wait state forcibly
                                       cleared
E_TMOUT       0FFFFFFABH(-H'00000055): Polling failed or timeout
E_DLT         0FFFFFFAFH(-H'00000051): The object being waited for
                                       was deleted
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
```

[(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.

- When the eventflag for which a task has been kept waiting is deleted by the del_flg system call issued by another task

  Error code E_DLT is returned.

You can specify a timeout (tmout) of -1 to 0x7FFFFFFF. 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.

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 <mr32r.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(CC32R) >>

    .include “mr32r.inc”
    .global task
task:
        :
    twai_flg   ID_flg2,H’ff0,(TWF_ANDW+TWF_CLR),5
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global task
task:
        :
    twai_flg   ID_flg2,0x0ff0,(TWF_ANDW+TWF_CLR),5
        :
```

2.3.8. pol_flg(Poll EventFlag)

[(System call name)]

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

[(Calling by the assembly language)]

```
.include "mr32r.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	The ID No. of the eventflag to check
R2	The bit pattern when wait state is cleared
R3	Wait mode

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_TMOUT      0FFFFFFABH(-H'00000055): Polling failed or timeout
E_NOEXS      0FFFFFFCCH(-H'00000034): Object does not exist
```

[(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.

An error E_NOEXS is returned if this system call is issued for a nonexistent eventflag.

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 <mr32r.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(CC32R) >>

    .include “mr32r.inc”
    .global  task
task:
    pol_flg   ID_flg2,H’ff0 (TWF_ORW+TWF_CLR)
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task
task:
    pol_flg   ID_flg2,0xff0 (TWF_ORW+TWF_CLR)
        :
```

2.3.9. ref_flg(Refer EventFlag Status)

[(System call name)]

ref_flg → Reference Eventflag Status.

[(Calling by the assembly language)]

```
.include “mr32r.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	The ID No. of the eventflag to Reference Eventflag
R2	Packet address to Reference Eventflag
R3	--

The area indicated by pk_rflg returns the following information.

Offset	Size		
+0	4	exinf	Extended information
+4	2	wtsk	Waiting task information
+8	4(U)	flgptn	Bit pattern of Eventflag

U: unsigned data.

[(Calling by the C language)]

```
#include <mr32r.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 */
}
```

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
E_NOEXS         0FFFFFFCCH(-H’00000034): Object does not exist
```

[(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

- 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.

An error E_NOEXS is returned if this system call is issued for a nonexistent eventflag.

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

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    T_RFLG rflg;
    ref_flg(&rflg, ID_flg );
                :
}

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global task
task:
    ld24     R2,#pk_rflg
    ref_flg   ID_flg
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global task
task:
    ld24     R2,#pk_rflg
    ref_flg   ID_flg
        :
```

2.4. Semaphore

2.4.1. cre_sem(Create Semaphore)

[(System call name)]

cre_sem → Create Semaphore

[(Calling by the assembly language)]

```
.include "mr32r.inc"
cre_sem     semid
```

<< Argument >>

semid	[**]	The ID No. of a semaphore to be created
pk_csem	[****]	The start address in which the semaphore generation information is stored (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a semaphore to be created
R2	The start address in which the semaphore generation information is stored
R3	--

Specify the following information in the structure indicased by pk_csem.

Offset	Size		
+0	4	exinf	Extended information
+4	4	sematr	Semaphore attribute
+8	4	isemcnt	Initial semaphore count
+12	4	maxsem	Maximun semaphore count

[(Calling by the C language)]

```
#include <mr32r.h>
ER cre_sem (semid, pk_csem);
```

<< Argument >>

ID	semid;	The ID No. of a semaphore to be created
T_CSEM	*pk_csem;	The start address in which the semaphore generation information is stored

Specify the following information in the structure indicased by pk_csem.

```
typedef struct t_csem {
    VP      exinf;  /* Extended information */
    ATR     sematr; /* Semaphore attribute */
    INT     isemcnt;        /* Initial semaphore count */
    INT     maxsem; /* Maximun semaphore count */
} T_CSEM;
```

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H'00000000): Normal End
E_OBJ           0FFFFFFC1H(-H'0000003f): Invalid object state
```

[(Function description)]

Creates a semaphore semid indicates.

Here follows explanation of the information as to a semaphore to be generated pk_csem.

- exinf (extended information)

  Exinf is an area you can freely use to store information as to a semaphore to be generated. MR32R has nothing to do with the exinf's contents.

- sematr (semaphore attribute)

  MR32R has nothing to do with this contents.

- isemcnt

  Set the initial semaphore counter value in this area when a semaphore created. The range of the specifiable value is 0 to 7FFFFFFFH.

- maxsem

  Set the maximam semaphore counter value in this area. The range of the specifiable value is 0 to 7FFFFFFFH.

An error E_OBJ is returned if cre_sem system call is issued for the semaphore which is existent. The range of the specifiable ID number is 1 to the maximum value specified in the configuration file.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
#define ID_sem1 1
void task1()
{
    T_CSEM csem;
    csem.isemcnt = 0xff;     /* Initial semaphore count */
    csem.maxsem = 0x7fffff;  /* Maximun semaphore count */
    cre_sem( ID_sem1, &setsem );
                :
}

<< Usage example of the assembly language(CC32R) >>

csem: .RES.B 16
ID_sem1:        .equ    1
    .include “mr32r.inc”
    .global   task1
task1:
        :
    ld24    R2,#setsem
    ld24    R1,#H’FF
    st      R1,@(8,R2)     /* Initial semaphore count */
    seth    R1,#H’7F
    or3     R1,R1,#H’FFFF   /* Maximun semaphore count */
    st      R1,@(12,R2)
    cre_sem  ID_sem1
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

csem: .space 16
    .equ ID_sem1,1
    .include “mr32r.inc”
    .global   task1
task1:
        :
    ld24    R2,#setsem
    ld24    R1,#0xFF
    st      R1,@(8,R2)     ; /* Initial semaphore count */
    seth    R1,#0x7F
    or3     R1,R1,#0xFFFF   ; /* Maximun semaphore count */
    st      R1,@(12,R2)
    cre_sem  ID_sem1
        :
    ext_tsk
```

2.4.2. del_sem(Delete Semaphore)

[(System call name)]

del_sem → Delete Semaphore

[(Calling by the assembly language)]

```
.include “mr32r.inc”
del_sem     semid
```

<< Argument >>

semid [**] The ID No. of a semaphore to be deleted

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a semaphore to be deleted
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER del_sem ( semid );
```

<< Argument >>

ID semid; The ID No. of a semaphore to be deleted

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
E_NOEXS         0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

del_sem deletes the semaphore semid indicates.

You can create the semaphore deleted as the same ID again.If the task is linked to the semaphore wait queue and del_sem is issued for the semaphore,this system call normally end.In this case,del_sem moves the task WAIT state to READY state.And an error E_DLT is returned.

An error E_NOEXS is returned if this system call is issued for a nonexistent semaphore.

Make sure this system call is issued for only the semaphore that has been created by the cre_sem system call. If this system call is issued for the semaphore that has been defined by the configuration file, it does not function normally.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
#define ID_sem2 2
void task1()
{
        :
    del_sem( ID_sem2 );
        :
}

<< Usage example of the assembly language(CC32R) >>

    .equ ID_sem2,2
    .include “mr32r.inc”
    .global  task1
task1:
        :
    del_sem ID_sem2
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .equ ID_sem2,2
    .include “mr32r.inc”
    .global  task1
task1:
        :
    del_sem ID_sem2
        :
    ext_tsk
```

2.4.3. sig_sem(Signal Semaphore)

[(System call name)]

sig_sem → Returns resource to the semaphore

[(Calling by the assembly language)]

```
.include "mr32r.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	The ID No. of the semaphore
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_QOVR       0FFFFFFB7H(-H'00000049): Queuing or nest overflow
E_NOEXS      0FFFFFFCCH(-H'00000034): Object does not exist
```

[(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.[19]

If it returns resource (sig_sem or isig_sem system call) is executed beyond the semaphore count value specified by cre_sem system call or the maximum value setting(maxsem) in the configuration file , error code E_QOVR is returned to the task which issued the system call with the semaphore count value left unchanged.

An error E_NOEXS is returned if this system call is issued for a nonexistent semaphore.

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.

[19] 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 <mr32r.h>
#include “id.h”
void task()
{
                :
    if( sig_sem( ID_sem ) != E_OK )
        error(“Overflow\n”);
                :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task
task:
    sig_sem   ID_sem
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task
task:
    sig_sem   ID_sem
        :
```

2.4.4. isig_sem(Signal Semaphore)

[(System call name)]

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

[(Calling by the assembly language)]

```
.include “mr32r.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	The ID No. of the semaphore
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK           00000000H(-H’00000000): Normal End
E_QOVR        0FFFFFFB7H(-H’00000049): Queuing or nest overflow
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
```

[(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 <mr32r.h>
#include “id.h”
void inthand()
{
                :
    if( isig_sem( ID_sem ) != E_OK )
        error(“Overflow\n”);
                :
}
```

```
<< Usage example of the assembly language(CC32R) >>

    .include mr32r.inc
    .global  intr
intr:
    isig_sem  ID_sem
        :
    ret_int

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include mr32r.inc
    .global  intr
intr:
    isig_sem  ID_sem
        :
    ret_int
```

2.4.5. wai_sem(Wait on Semaphore)

[(System call name)]

wai_sem → Obtains one resource from the semaphore.

[(Calling by the assembly language)]

```
.include "mr32r.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	The ID No. of the semaphore from which the resource is obtained
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_RLWAI       0FFFFFFAAH(-H'00000056): Wait state forcibly
                                       cleared
E_DLT         0FFFFFFAFH(-H'00000051): The object being waited for
                                       was deleted
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
```

[(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.[20]

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

If the task waits for semaphore and del_sem is issued for it, del_sem system call moves the task WAIT state to READY state.And error E_DLT is returned.

Error E_NOEXS is returned if this system call is issued for a nonexistent semaphore.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
                :
    if( wai_sem( ID_sem ) != E_OK )
        printf(“Forced wakeup\n”);
                :
}

<< Usage example of the assembly language(CC32R) >>
    .include “mr32r.inc”
    .global  task
task:
    wai_sem   ID_sem
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>
    .include “mr32r.inc”
    .global  task
task:
    wai_sem   ID_sem
        :
```

[20] 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.

2.4.6. twai_sem(Wait on Semaphore with Timeout)

[(System call name)]

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

[(Calling by the assembly language)]

```
.include "mr32r.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	The ID No. of the semaphore from which the resource
R2	--
R3	--
R4	Timeout value

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_TMOUT       0FFFFFFABH(-H'00000055): Polling failed or timeout
E_RLWAI       0FFFFFFAAH(-H'00000056): Wait state forcibly
                                       cleared
E_DLT         0FFFFFFAFH(-H'00000051): The object being waited for
                                       was deleted
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
```

[(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 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 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.

- When the semaphore for which a task has been kept waiting is deleted by the del_sem system call issued by another task

  Error code E_DLT is returned.

Error E_NOEXS is returned if this system call is issued for a nonexistent semaphore.

You can specify a timeout (tmout) of -1 to 0x7FFFFFFF. 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.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
                :
    if( twai_sem( ID_sem, 10 ) != E_OK )
        printf(“Forced wakeup\n”);
                :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .GLOBAL  task
task:
        :
    twai_sem   ID_sem,10
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .GLOBAL  task
task:
        :
    twai_sem   ID_sem,10
        :
```

2.4.7. preq_sem(Poll and Request Semaphore)

[(System call name)]

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

[(Calling by the assembly language)]

```
.include "mr32r.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	The ID No. of the semaphore from which the resource is obtained
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_TMOUT       0FFFFFFABH(-H'00000055): Polling failed or timeout
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
```

[(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.

Error E_NOEXS is returned if this system call is issued for a nonexistent semaphore.

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

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
                :
    if( preq_sem( ID_sem ) != E_OK )
        printf(“No more resource\n”);
                :
}

<< Usage example of the assembly language(CC32R) >>

    .include mr32r.inc
    .global  task
task:
        :
    preq_sem  ID_sem
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>
    .include mr32r.inc
    .global  task
task:
        :
    preq_sem  ID_sem
        :
```

2.4.8. ref_sem(Refer Semaphore Status)

[(System call name)]

ref_sem → Reference Semaphore Status

[(Calling by the assembly language)]

```
.include "mr32r.inc"
ref_sem   semid
```

<< 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	The ID No. of the semaphore to Reference Semaphore
R2	Packet address to Reference Semaphore
R3	--

The area indicated by pk_rsem returns the following information.

Offset	Size		
+0	4	exinf	Extended information
+4	2	wtsk	Waiting task information
+8	4	semcnt	Current semaphore count

[(Calling by the C language)]

```
#include <mr32r.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 */
}
```

[(Error codes)]

```
E_OK            00000000H(-H'00000000): Normal End
E_NOEXS         0FFFFFFCCH(-H'00000034): Object does not exist
```

[(Function description)]

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

- exinf

  Returns extended task information in exinf.

- 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.

An error E_NOEXS is returned if this system call is issued for a nonexistent semaphore.

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

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    T_RSEM   rsem;
        :
    ref_sem( &rsem, ID_sem );
        :
}

<< Usage example of the assembly language(CC32R) >>

rsem:   .RES.B   12
    .include “mr32r.inc”
    .global task
task:
        :
    ld24      R2,#rsem
    ref_sem    ID_sem1
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

rsem:   .space   12
    .include “mr32r.inc”
    .global task
task:
        :
    ld24      R2,#rsem
    ref_sem    ID_sem1
        :
```

2.5. Mailbox

2.5.1. cre_mbx(Create Mailbox)

[(System call name)]

cre_mbx → Create Mailbox

[(Calling by the assembly language)]

```
.include “mr32r.inc”
cre_mbx     mbxid
```

<< Argument >>

mbxid	[**]	The ID No. of a mailbox to be created
pk_cmbx	[****]	The start address in which the mailbox generation information is stored (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a mailbox to be created
R2	The start address in which the mailbox generation information is stored
R3	--

Specify the following information in the structure indicased by pk_cmbx.

Offset	Size		
+0	4	exinf	Extended information
+4	4	mbxatr	Mailbox attribute
+8	4	bufcnt	Ringbuffer size

[(Calling by the C language)]

```
#include <mr32r.h>
ER cre_mbx (mbxid, pk_cmbx);
```

<< Argument >>

ID	mbxid;	The ID No. of a mailbox to be created
T_CMBX	*pk_cmbx;	The start address in which the mailbox generation information is stored

Specify the following information in the structure indicased by pk_cmbx.

```
typedef struct t_cmbx {
    VP      exinf;  /* Extended information */
    ATR     mbxatr; /* Mailbox attribute */
    INT     bufcnt; /* Ringbuffer size */
} T_CMBX;
```

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
E_OBJ           0FFFFFFC1H(-H’0000003f): Invalid object state
E_NOMEM         0FFFFFFF6H(-H’0000000a): Not enough of memory
```

[(Function description)]

Creates a mailbox mbxid indicates.

Here follows explanation of the information as to a mailbox to be generated pk_cmbx.

- exinf (extended information)

  Exinf is an area you can freely use to store information as to a mailbox to be generated. MR32R has nothing to do with the exinf's contents.

- mbxatr (mailbox attribute)

  Specify the location of the mailbox area to be created. Specifically this means specifying whether you want the mailbox to be located in the internal RAM or in external RAM.

  - **To locate the mailbox area in internal RAM**
    Specify __MR_INT(0).
  - **To locate the mailbox area in external RAM**
    Specify __MR_EXT(0x10000).
  - **To locate the mailbox area user specified**
    Specify __MR_USER(0x20000).

- bufcnt

  Specify the buffer size stored with messages of the mailbox.The unit is not bytes number,but message number.

An error E_OBJ is returned if cre_mbx system call is issued for the mailbox which is existent.

The range of the specifiable ID number is 1 to the maximum value specified in the configuration file.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
#define ID_mbx1 1
void task1()
{
    T_CMBX setmbx;
                :
    setmbx.mbxatr = __MR_EXT;
    setmbx.bufcnt = 10;   /* Ringbuffer size */
    cre_mbx( ID_mbx1, &setmbx );
                :
    ext_tsk();
}

<< Usage example of the assembly language(CC32R) >>

    .equ ID_mbx1,1
    setmbx: .RES.B 12
    .include “mr32r.inc”
    .global   task1
task1:
        :
    ld24    R2,#setmbx
    ld24    R1,#__MR_EXT
    st      R1,@(4,R2)
    ld24    R1,#10
    st      R1,@(8,R2)
    cre_mbx ID_mbx1
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .equ ID_mbx1,1
    setmbx: .space 12
    .include “mr32r.inc”
    .global   task1
task1:
        :
    ld24    R2,#setmbx
    ld24    R1,#__MR_EXT
    st      R1,@(4,R2)
    ld24    R1,#10
    st      R1,@(8,R2)
    cre_mbx ID_mbx1
        :
    ext_tsk
```

2.5.2. del_mbx(Delete Mailbox)

[(System call name)]

del_mbx → Delete Mailbox

[(Calling by the assembly language)]

```
.include “mr32r.inc”
del_mbx     mbxid
```

<< Argument >>

mbxid [**] The ID No. of a mailbox to be deleted

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a mailbox to be deleted
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER del_mbx ( mbxid );
```

<< Argument >>

ID mbxid; The ID No. of a mailbox to be deleted

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
E_NOEXS         0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

del_mbx deletes the mailbox mbxid indicates.

You can create the mailbox deleted as the same ID again.If the task is linked to the message wait queue and del_mbx is issued for the mailbox,this system call normally end.In this case,del_mbx moves the task WAIT state to READY state.And error E_DLT is returned.If some messages are in the mailbox,these are deleted.

An error E_NOEXS is returned if this system call is issued for a nonexistent mailbox.

Make sure this system call is issued for only the mailbox that has been created by the cre_mbx system call. If this system call is issued for the mailbox that has been defined by the configuration file, it does not function normally.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
#define ID_mbx2 2
void task1(void)
{
        :
    del_mbx( ID_mbx2 );
        :
    ext_tsk();
}

<< Usage example of the assembly language(CC32R) >>

ID_mbx2:        .equ    2
    .include “mr32r.inc”
    .global   task1
task1:
        :
    del_mbx ID_mbx2
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .equ ID_mbx2,2
    .include “mr32r.inc”
    .global   task1
task1:
        :
    del_mbx ID_mbx2
        :
    ext_tsk
```

2.5.3. snd_msg(Send Message to Mailbox)

[(System call name)]

snd_msg → Sends a message.

[(Calling by the assembly language)]

```
.include "mr32r.inc"
snd_msg     mbxid
```

<< Argument >>

mbxid	[**]	The ID No. of the mailbox to which a message is sent
pk_msg	[****]	The start address of message packet (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the mailbox to which a message is sent
R2	The start address of message packet
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_QOVR        0FFFFFFB7H(-H'00000049): Queuing or nest overflow
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
```

[(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.[21] 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 or when cre_mbx system call is issued.

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.

An error E_NOEXS is returned if this system call is issued for a nonexistent mailbox.

A message is 32bits wide data.[22]In standard μITRON specifications, this data is interpreted as indicating the start address of a message packet (a structure including the message), i.e., address transfer. In MR32R, however, messages can be handled in two ways to perform data communication as described below.

1. Using a message as the start address (32 bits) of a message packet
   Since no specific types of message packets (T_MSG) are stipulated in MR32R, any desired message type can be defined by the user. It can be an array, for example. [23]

   Example:

   ```
   typedef char * T_MSG;
   ```

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

   ```
   T_MSG    * pk_msg;
   ```

2. Using a message simply as 32-bits 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 32-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.

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

[(Usage example)]

```
<< Usage example of the C language >>
```

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

```
#include <mr32r.h>
#include “id.h”
typedef char T_MSG;
T_MSG msg[10];
void task(void)
{
                :
    if( snd_msg( ID_msg, msg) != E_OK ){
        error(“overflow\n”);
    }
        :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task
msg:.SDATA “message”
        .DATA.B 0
task:
    snd_msg ID_msg, msg
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task
msg:.byte “message”
        .byte 0
task:
    snd_msg ID_msg, msg
        :
```

2.5.4. isnd_msg(Send Message to Mailbox)

[(System call name)]

isnd_msg → Sends a message. (for the handler only).

[(Calling by the assembly language)]

```
.include   “mr32r.inc”
isnd_msg   mbxid
```

<< Argument >>

mbxid	[**]	The ID No. of the mailbox to which a message is sent
pk_msg	[****]	The start address of message packet (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the mailbox to which a message is sent
R2	The start address of message packet
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK           00000000H(-H’00000000): Normal End
E_QOVR        0FFFFFFB7H(-H’00000049): Queuing or nest overflow
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
```

[(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 <mr32r.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(CC32R) >>

    .include “mr32r.inc”
    .global  intr
intr:
        :
    isnd_msg ID_msg, H’1234
        :
    ret_int

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  intr
intr:
        :
    isnd_msg ID_msg, 0x1234
        :
    ret_int
```

2.5.5. rcv_msg(Receive Message from Mailbox)

[(System call name)]

rcv_msg → Waits for receiving a message.

[(Calling by the assembly language)]

```
.include "mr32r.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 ID No. of the mailbox to which a message is received
R2	The start address of message packet
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_RLWAI       0FFFFFFAAH(-H'00000056): Wait state forcibly
                                       cleared
E_DLT         0FFFFFFAFH(-H'00000051): The object being waited for
                                       was deleted
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
```

[(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 mailbox, 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.

Also, if the mailbox for a task waiting for conditions to be met is deleted by the del_mbx system call issued by another task, the waiting task is released from the transmit mailbox wait state and error E_DLT is returned to that task and changes to executable (READY) state.

An error E_NOEXS is returned if this system call is issued for a nonexistent mailbox.

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
   The message width comprises 32 bits, so you have to declare the pointer variable (ppk_msg) toward the area in which the foremost address of the message packet is stored as given below.

   ```
   T_MSG **ppk_msg;
   ```

2. When using a message simply as data
   You have to declare the pointer variable (ppk_msg) toward the area in which 32-bit data is stored as given below.

   ```
   T_MSG * ppk_msg;
   ```

   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 l of int type, for example:

   ```
   int i, j;

   snd_msg( ID_mbx, (PT_MSG)i );

   rcv_msg( (PT_MSG *)&j, ID_mbx );
   ```

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>
```

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

```
#include <mr32r.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(CC32R) >>

    .include “mr32r.inc”
    .global  task
task:
        :
    rcv_msg ID_mbx
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task
task:
        :
    rcv_msg ID_mbx
        :
```

2.5.6. trcv_msg(Receive Message with Timeout)

[(System call name)]

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

[(Calling by the assembly language)]

```
.include "mr32r.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 ID No. of the mailbox from which a message is received
R2	The start address of message packet
R3	--
R4	Timeout value

[(Calling by the C language)]

```
#include <mr32r.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 >>

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

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_TMOUT       0FFFFFFABH(-H'00000055): Polling failed or timeout
E_RLWAI       0FFFFFFAAH(-H'00000056): Wait state forcibly
                                       cleared
E_DLT         0FFFFFFAFH(-H'00000051): The object being waited for
                                       was deleted
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
```

[(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.

- When the mailbox for which a task has been kept waiting is deleted by the del_mbx system call issued by another task

  Error code E_DLT is returned.

You can specify a timeout (tmout) of -1 to 0x7FFFFFFF. 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 page for precautions should observed when receiving a message.

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 <mr32r.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(CC32R) >>
    .include “mr32r.inc”
    .global task
task:
        :
    trcv_msg ID_mbx,10
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>
    .include “mr32r.inc”
    .global task
task:
        :
    trcv_msg ID_mbx,10
        :
```

2.5.7. prcv_msg(Poll and Receive Message)

[(System call name)]

prcv_msg → Receiving a message. (no wait)

[(Calling by the assembly language)]

```
.include “mr32r.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 ID No. of the mailbox from which a message is received
R2	The start address of message packet
R3	--

[(Calling by the C language)]

```
#include <mr32r.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 >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_TMOUT      0FFFFFFABH(-H’00000055): Polling failed or timeout
E_NOEXS      0FFFFFFCCH(-H’00000034): Object does not exist
```

[(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 mailbox, an error E_TMOUT is returned to the system call issued task.

An error E_NOEXS is returned if this system call is issued for a nonexistent mailbox.

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

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

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.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(CC32R) >>
    .include “mr32r.inc”
    .global  task
task:
    prcv_msg ID_mbx1
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>
    .include “mr32r.inc”
    .global  task
task:
    prcv_msg ID_mbx1
        :
```

2.5.8. ref_mbx(Refer Mailbox Status)

[(System call name)]

ref_mbx → Reference Mailbox Status

[(Calling by the assembly language)]

```
.include "mr32r.inc"
ref_mbx mbxid
```

<< Argument >>

mbxid	[**]	The ID No. of the mailbox to Reference Mailbox
pk_rmbx	[****]	Packet address to Reference Mailbox (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the mailbox to Reference Mailbox
R2	Packet address to Reference Mailbox
R3	--

The structure indicated by pk_rmbx returns the following data.

Offset	Size		
+0	4	exinf	Extended information
+4	2	wtsk	Waiting task information
+8	4(U)	pk_msg	Starting address of next received message packet
12	4	msgcnt	The number of messages

U: unsigned data.

[(Calling by the C language)]

```
#include <mr32r.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;
```

[(Error codes)]

```
E_OK            00000000H(-H'00000000):Normal End
E_NOEXS         0FFFFFFCCH(-H'00000034): Object does not exist
```

[(Function description)]

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

- exinf

  Returns extended task information in exinf.

- 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=FFFFFFFFH=(-1). if there is no message.

- msgcnt

  Returns the number of messages currently in the target mailbox.

An error E_NOEXS is returned if this system call is issued for a nonexistent mailbox.

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

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    T_RMBX rmbx;
        :
    ref_mbx(ID_mbx, &rmbx);
        :
}

<< Usage example of the assembly language(CC32R) >>

rmbx:    .RES.B 12
    .include “mr32r.inc”
    .global task
task:
        :
    ld24      R2,#rmbx
    ref_mbx    ID_mbx
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

rmbx:    .space 12
    .include “mr32r.inc”
    .global task
task:
        :
    ld24      R2,#rmbx
    ref_mbx    ID_mbx
        :
```

2.6. Messagebuffer

2.6.1. cre_mbf(Create Messagebuffer)

[(System call name)]

cre_mbf → Create Messagebuffer

[(Calling by the assembly language)]

```
.include "mr32r.inc"
cre_mbf     mbfid
```

<< Argument >>

mbfid	[**]	The ID No. of a messagebuffer to be created
pk_cmbf	[****]	The start address in which the messagebuffer generation information is stored (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a messagebuffer to be created
R2	The start address in which the messagebuffer generation information is stored
R3	--

Specify the following information in the structure indicased by pk_ctsk.

Offset	Size		
+0	4	exinf	Extended information
+4	4	mbfatr	Messagebuffer attribute
+8	4	bufsz	Messagebuffer size
+12	4	maxmsz	Maximum size of messages

[(Calling by the C language)]

```
#include <mr32r.h>
ER cre_mbf (mbfid, pk_cmbf);
```

<< Argument >>

ID	mbfid;	The ID No. of a messagebuffer to be created
T_CTSK	*pk_ctsk;	The start address in which the messagebuffer generation information is stored

Specify the following information in the structure indicased by pk_cmbf.

```
typedef struct t_cmbf {
    VP      exinf;  /* Extended information */
    ATR     mbfatr; /* Messagebuffer attribute */
    INT     bufsz;  /* Messagebuffer size */
    INT     maxmsz; /* Maximum size of message */
} T_CMBF;
```

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H'00000000): Normal End
E_NOMEM         0FFFFFFF6H(-H'0000000a): Not enough of memory
E_OBJ           0FFFFFFC1H(-H'0000003f): Invalid object state
```

[(Function description)]

Creates a messagebuffer mbfid indicates.

The message buffer consists of the ring buffer whose size is specified as bufsz. Here follows explanation of the information as to a messagebuffer to be generated pk_cmbf.

- exinf (extended information)

  Exinf is an area you can freely use to store information as to a messagebuffer to be generated. MR32R has nothing to do with the exinf's contents.

- mbfatr (messagebuffer attribute)

  Specify the location of the messagebuffer area to be created. Specifically this means specifying whether you want the messagebuffer to be located in the internal RAM or in external RAM.

  - **To locate the messagebuffer area in internal RAM**

    Specify __MR_INT(0).

  - **To locate the messagebuffer area in external RAM**
    Specify __MR_EXT(0x10000).

  - **To locate the messagebuffer area user specified**
    Specify __MR_USER(0x30000).

- bufsz (Specify multiple of four)

  Specify the size of message bufferto be created.It must be multiple of four.

  You can specify the bufsz=0.In this case,the message buffer communication is completely synchronized.

- maxmsz

  Specify the maximum length of message in the message buffer to be created.MR32R does not refer maxmsz,so you need not set this item.If you want to have a compatibility between MR32R and other Realtime OS,set this item.

An error E_OBJ is returned if cre_mbf system call is issued for the message buffer which is existent.

The range of the specifiable ID number is 1 to the maximum value specified in the configuration file.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
#define ID_mpl1 1
void task1(void)
{
    T_CMBX setmbf;
    setmbf.mbfatr = __MR_INT;
    setmbf.bufsz = 200;
    setmbf.maxmsz = 30;
    cre_mbf( ID_mpl1, &setmbf );
                :
}

<< Usage example of the assembly language(CC32R) >>

setmbf: .RES.B 16
ID_mbf1:        .equ    1
    .include “mr32r.inc”
    .global  task1
task1:
        :
    ld24    R2,#setmbf
    ld24    R1,#__MR_INT
    st      R1,@(4,R2)
    ld24    R1,#200
    st      R1,@(8,R2)
    ld24    R1,#30
    st      R1,@(12,R2)
    cre_mbf ID_mbf1
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

setmbf: .space 16
    .equ        ID_mbf1,1
    .include “mr32r.inc”
    .global  task1
task1:
        :
    ld24    R2,#setmbf
    ld24    R1,#__MR_INT
    st      R1,@(4,R2)
    ld24    R1,#200
    st      R1,@(8,R2)
    ld24    R1,#30
    st      R1,@(12,R2)
    cre_mbf ID_mbf1
        :
    ext_tsk
```

2.6.2. del_mbf(Delete Massagebuffer)

[(System call name)]

del_tsk → Delete Messagebuffer

[(Calling by the assembly language)]

```
.include “mr32r.inc”
del_mbf     mbfid
```

<< Argument >>

mbfid [**] The ID No. of a messagebuffer to be deleted

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a messagebuffer to be deleted
R2	--
R3	--

[(Calling by the C language)]

```
# include <mr32r.h>
ER del_mbf ( mbfid );
```

<< Argument >>

ID tskid; The ID No. of a task to be deleted

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
E_NOEXS         0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

del_mbf deletes the messagebuffer mbfid indicates.

Once this messagebuffer is deleted, you can create a new messagebuffer with the same ID number. Even when there is any task waiting for a messagebuffer to be deleted, this system call is terminated normally. In this case, the said task is freed from the messagebuffer wait state and returns error E_DLT.The messages in the message buffer are deleted by del_mbf system call because the message buffer area is released.

An error E_NOEXS is returned if this system call is issued for a nonexistent messagebuffer.

Make sure this system call is issued for only the messagebuffer that has been created by the cre_mbf system call. If this system call is issued for the messagebuffer that has been defined by the configuration file, it does not function normally.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
#define ID_mbf2 2
void task1()
{
        :
    del_mbf( ID_mbf2 );
        :
    ext_tsk();
}

<< Usage example of the assembly language(CC32R) >>

ID_mbf2:        .equ    2
    .include “mr32r.inc”
    .global   task1
task1:
        :
    del_mbf ID_mbf2
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .equ ID_mbf2,2
    .include “mr32r.inc”
    .global   task1
task1:
        :
    del_mbf ID_mbf2
        :
    ext_tsk
```

2.6.3. snd_mbf(Send Message to Messagbuffer)

[(System call name)]

snd_msf → Sends a message.

[(Calling by the assembly language)]

```
.include “mr32r.inc”
snd_mbf     mbfid, msgsz
```

<< Argument >>

mbfid	[**]	The ID No. of the messagebuffer to which a message is sent
msg	[****]	The start address of a message packet (Set the address in the R2 register.)
msgsz	[****]	The size of message

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the messagebuffer to which a message is sent
R2	The start address of a message packet
R3	The size of a message

[(Calling by the C language)]

```
#include <mr32r.h>
ER snd_mbf (mbfid, msg, msgsz);
```

<< Argument >>

ID	mbxid;	The ID No. of the messagebuffer to which a message is sent
VP	msg;	The start address of a message packet
INT	msgsz;	The size of a message

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_RLWAI       0FFFFFFAAH(-H’00000056): Wait state forcibly
                                       cleared
E_DLT         0FFFFFFAFH(-H’00000051): The object being waited for
                                       was deleted
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

snd_mbf system call sends the message in the address of msg to the message buffer specified with mbfid.Specify message size in msgsz.snd_mbf copys the msgsz bytes letters after msg to the queue of the message buffer specified as mbfid. The message buffer consists of the ring buffer.

If msgsz is larger than the value specified by cre_mbf,no error is returned.[24]

If the buffer's available space is so small that the msg message cannot fit into the message queue, the task that issued this system call is placed in send wait state. Accordingly, the task is queued up in two queues: the send wait queue. The sequence of wait queues is FIFO.

If the task is forcibly released from the wait state by the rel_wai or irel_wai system call, error E_RLWAI is returned.

Also, if the messagebuffer for a task waiting for conditions to be met is deleted by the del_mbf system call issued by another task, the waiting task is released from the transmit send messagebuffer wait state and error E_DLT is returned to that task.

If the message buffer for a task waiting for conditions to be met is reset by the vrst_mbf system call issed by another task,the waiting task is released from send messagebuffer wait state and error EV_RST is returned to that task.

An error E_NOEXS is returned if this system call is issued for a nonexistent messagebuffer.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    char *msg=”abcdef”;
                :
    tsnd_mbf( ID_mbf, msg, 6);
                :
}
```

[24] Please check yourself whether msgsz is smaller than maxmsz specified by cre_mbf or not.

```
<< Usage example of the assembly language(CC32R) >>

mbf:.SDATA “abcdef”
        .DATA.B 0
    .include “mr32r.inc”
    .GLOBAL   task
task:
        :
    ld24        R2,#mbf
    snd_mbf ID_mbf,6
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

mbf:.byte “abcdef”
    .byte 0
    .include “mr32r.inc”
    .GLOBAL   task
task:
        :
    ld24        R2,#mbf
    snd_mbf ID_mbf,6
        :
```

2.6.4. tsnd_mbf(Send Message to Messagbuffer with Timeout)

[(System call name)]

tsnd_msg → Sends a message (With Timeout).

[(Calling by the assembly language)]

```
.include "mr32r.inc"
tsnd_mbf     mbfid, msgsz,tmout
```

<< Argument >>

mbfid	[**]	The ID No. of the messagebuffer to which a message is sent
msg	[****]	The start address of a message packet (Set the address in the R2 register.)
msgsz	[****]	The size of a message
tmout	[****]	Timeout value

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the messagebuffer to which a message is sent
R2	The start address of a message packet
R3	The size of a message

[(Calling by the C language)]

```
#include <mr32r.h>
ER tsnd_mbf (mbfid, msg, msgsz, tmout);
```

<< Argument >>

ID	mbxid;	The ID No. of the messagebuffer to which a message is sent
VP	msg;	The start address of a message packet
INT	msgsz;	The size of a message
TMO	tmout;	Timeout value

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_RLWAI       0FFFFFFAAH(-H'00000056): Wait state forcibly
                                       cleared
E_DLT         0FFFFFFAFH(-H'00000051): The object being waited for
                                       was deleted
E_TMOUT       0FFFFFFABH(-H'00000055): Polling failed or timeout
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
```

[(Function description)]

snd_mbf system call sends the message in the address of msg to the message buffer specified with mbfid.Specify message size in msgsz.snd_mbf copys the msgsz bytes letters after msg to the queue of the message buffer specified as mbfid. The message buffer consists of the ring buffer.

If msgsz is larger than the value specified by cre_mbf,no error is returned[25].

If the buffer's available space is so small that the msg message cannot fit into the message queue, the task that issued this system call is placed in send wait state. Accordingly, the task is queued up in two queues: the send wait queue and the timeout wait queue. The sequence of wait queues is FIFO.

The wait state committed by issuing this system call is released in the cases described below. Note that when released from the wait state, the task that issued this system call is removed from both of the send wait and timeout wait queues and is connected to the ready queue.

- When the release-from-wait condition is met before the tmout time expires

  Error code E_OK is returned.

- When the tmout time expires before the release-from-wait condition is met

  Error code E_TMOUT is returned.

- When the rel_wai or irel_wai system call is issued before the send messagebuffer wait condition is met

  Error code E_RLWAI is returned.

- When the messagebuffer for which a task has been kept waiting is deleted by the del_mbf system call issued by another task

  Error code E_DLT is returned.

- When the messagebuffer for which a task has been kept waiting is reset by the vrst_mbf system call issued by another task

  Error code EV_RST is returned.

Any value from -1 to 7FFFFFFFH can be specified for tmout. If you specify TMO_POL = 0 for tmout, the effect is the same as 0 is specified for the timeout value, in which case tmout functions the same way as psnd_mbf. Also, if you specify tmout = TMO_FEVR(-1), the effect is the same as endless wait is specified, in which case tmout functions the same way as snd_mbf.

An error E_NOEXS is returned if this system call is issued for a nonexistent messagebuffer.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[25] Please check yourself whether msgsz is smaller than maxmsz specified by cre_mbf or not.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    char *msg=”abcdef”;
        :
    tsnd_mbf( ID_mbf, msg, 6, 100);
        :
}

<< Usage example of the assembly language(CC32R) >>

mbf:.SDATA “abcdef”
        .DATA.B 0
    .include “mr32r.inc”
    .global  task
task:
        :
    ld24       R2,#mbf
    tsnd_mbf ID_mbf,6,100
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

mbf:.byte “abcdef”
    .byte 0
    .include “mr32r.inc”
    .global  task
task:
        :
    ld24       R2,#mbf
    tsnd_mbf ID_mbf,6,100
        :
```

2.6.5. psnd_mbf(Poll and Send Messagebuffer)

[(System call name)]

psnd_msg → Sends a message (no wait).

[(Calling by the assembly language)]

```
.include "mr32r.inc"
psnd_mbf     mbfid, msgsz
```

<< Argument >>

mbfid	[**]	The ID No. of the messagebuffer to which a message is sent
msg	[****]	The start address of a message packet (Set the address in the R2 register.)
msgsz	[****]	The size of a message

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the messagebuffer to which a message is sent
R2	The start address of message packet
R3	The size of message

[(Calling by the C language)]

```
#include <mr32r.h>
ER psnd_mbf (mbfid, msg, msgsz);
```

<< Argument >>

ID	mbxid;	The ID No. of the messagebuffer to which a message is sent
VP	msg;	The start address of a message packet
INT	msgsz;	The size of a message

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_TMOUT      0FFFFFFABH(-H'00000055): Polling failed or timeout
E_NOEXS      0FFFFFFCCH(-H'00000034): Object does not exist
```

[(Function description)]

psnd_mbf system call sends the message in the address of msg to the message buffer specified with mbfid.Specify message size in msgsz.psnd_mbf copys the msgsz bytes letters after msg to the queue of the message buffer specified as mbfid. The message buffer consists of the ring buffer.

If msgsz is larger than the value specified by cre_mbf,no error is returned[26].

If there is no space in message buffer area,error E_TMOUT is returned.

The task is not moved to WAIT state by psnd_mbf.

An error E_NOEXS is returned if this system call is issued for a nonexistent messagebuffer.

[26] Please check yourself whether msgsz is smaller than maxmsz specified by cre_mbf or not.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void inthand()
{
    char *msg=”abcdef”;
                :
    if( psnd_mbf( ID_mbf, msg, 6) != E_OK ){
        error(“overflow\n”);
                :
}

<< Usage example of the assembly language(CC32R) >>

mbf:.SDATA “abcdef”
        .DATA.B 0
    .include “mr32r.inc”
    .global  intr
intr:
        :
    ld24        R2,#mbf
    psnd_mbf ID_mbf, 6
        :
    ret_int

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

mbf:.byte “abcdef”
    .byte 0
    .include “mr32r.inc”
    .global  intr
intr:
        :
    ld24        R2,#mbf
    psnd_mbf ID_mbf, 6
        :
    ret_int
```

2.6.6. rcv_mbf(Receive Messagebuffer)

[(System call name)]

rcv_mbf → Waits for receiving a message from Messagebuffer.

[(Calling by the assembly language)]

```
.include "mr32r.inc"
rcv_mbf     mbfid, msg
```

<< Argument >>

mbfid	[**]	The ID No. of the messagebuffer from which a message pakect is received
msg	[****]	The start address in which a receive message packet is stored. (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the messagebuffer from which a message packet is received
R2	The start address in which a receive message packet is stored.
R3	The size of a receive message packet

[(Calling by the C language)]

```
#include <mr32r.h>
ER rcv_mbf (msg, p_msgsz, mbfid);
```

<< Argument >>

ID	mbfid;	The ID No. of the messagebuffer from which a message packet is received
INT	*p_msgsz;	The start address in which a receive message packet is stored.
VP	msg;	The size of a receive message packet

<< Return value >>

An error code is returned as the return value of a function.
The size of a receive message packet is stored in p_msgsz.

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_RLWAI       0FFFFFFAAH(-H'00000056): Wait state forcibly
                                       cleared
E_DLT         0FFFFFFAFH(-H'00000051): The object being waited for
                                       was deleted
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
```

[(Function description)]

rcv_mbf recieves the message in the message buffer specified as mbfid,and store to the area specified as msg.If there is a send WAIT state task,rcv_mbf compare the size of message to be sent with the size of received message.

1. Recieve message size is larger than send message size.

The message is sent to messagebuffer and the task is moved WAIT state to READY state.

2. Receive message size is smaller than send message size.
   rcv_mbf does not send the message to message buffer,leave the task WAIT state, and end.

After the process of 1,rcv_mbf does as the sam if there is the send WAIT state task.If the message is not sent to the message buffer speified as mbfid,the task is moved to WAIT state FIFO ordered.

If the task is forcibly released from the wait state by the rel_wai or irel_wai system call, error E_RLWAI is returned.

If the messagebuffer for a task waiting for conditions to be met is deleted by the del_mbf system call issued by another task, the waiting task is released from the receive wait state and error E_DLT is returned to that task.

If the messagebuffer for a task waiting for conditions to be met is reset by the vrst_mbf system call issued by another task, the waiting task is released from the receive wait state and error EV_RST is returned to that task.

An error E_NOEXS is returned if this system call is issued for a nonexistent messagebuffer.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    char   msg[128];
    INT    msgsz;
    if( rcv_mbf( (VP)msg, &msgsz, ID_mbf) != E_OK )
                :
}

<< Usage example of the assembly language(CC32R) >>

msg: .RES.B 32
    .include “mr32r.inc”
    .global  task
task:
        :
    ld24       R2,#msg
    rcv_mbf ID_mbf
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

msg: .space 32
    .include “mr32r.inc”
    .global  task
task:
        :
    ld24       R2,#msg
    rcv_mbf ID_mbf
        :
```

2.6.7. trcv_mbf(Receive Messagebuffer with Timeout)

[(System call name)]

trcv_mbf	→	Waits for receiving a message from Messagebuffer. (With Timeout)

[(Calling by the assembly language)]

```
.include "mr32r.inc"
trcv_mbf     mbfid, tmout
```

<< Argument >>

mbfid	[**]	The ID No. of the messagebuffer from which a message pakect is received
msg	[****]	The start address in which a receive message packet is stored. (Set the address in the R2 register.)
tmout	[****]	Timeout value

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the messagebuffer from which a message packet is received
R2	The start address in which a receive message packet is stored.
R3	The size of a receive message packet
R4	Timeout value

[(Calling by the C language)]

```
#include <mr32r.h>
ER trcv_mbf (msg, p_msgsz, mbfid, tmout);
```

<< Argument >>

ID	mbfid;	The ID No. of the messagebuffer from which a message packet is received
INT	*p_msgsz;	The start address in which a receive message packet is stored.
VP	msg;	The size of a receive message packet
TMO\	tmout;	Timeout value

<< Return value >>

An error code is returned as the return value of a function.
The size of a receive message packet is stored in p_msgsz.

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_RLWAI       0FFFFFFAAH(-H'00000056): Wait state forcibly
                                       cleared
E_DLT         0FFFFFFAFH(-H'00000051): The object being waited for
                                       was deleted
E_TMOUT       0FFFFFFABH(-H'00000055): Polling failed or timeout
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
```

[(Function description)]

trcv_mbf recieves the message in the message buffer specified as mbfid,and store to the area specified as msg.If there is a send WAIT state task,trcv_mbf compare the size of message to be sent with the size of received message.

1. Recieve message size is larger than send message size.
   The message is sent to messagebuffer and the task is moved WAIT state to READY state.
2. Receive message size is smaller than send message size.
   rcv_mbf does not send the message to message buffer,leave the task WAIT state, and end.

After the process of 1,trcv_mbf does as the sam if there is the send WAIT state task.If the message is not sent to the message buffer speified as mbfid,the task is moved to WAIT state and linked to the timeout queue and the receive wait queue.

The wait state committed by issuing this system call is released in the cases described below. Note that when released from the wait state, the task that issued this system call is removed from both of the recieve wait and timeout wait queues 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 rel_wai or irel_wai system call is issued before the recieve messagebuffer wait condition is met

  Error code E_RLWAI is returned.

- When the message buffer for which a task has been kept waiting is deleted by the del_mbf system call issued by another task

  Error code E_DLT is returned.

- When the message buffer for which a task has been kept waiting is reset by the vrst_mbf system call issued by another task

  Error code EV_RST is returned.

Any value from -1 to 7FFFFFFFH can be specified for tmout. If you specify TMO_POL = 0 for tmout, the effect is the same as 0 is specified for the timeout value, in which case tmout functions the same way as prcv_mbf. Also, if you specify tmout = TMO_FEVR(-1), the effect is the same as endless wait is specified, in which case tmout functions the same way as rcv_mbf.

An error E_NOEXS is returned if this system call is issued for a nonexistent messagebuffer.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    char   msg[128];
    INT    msgsz;
    if( trcv_mbf( (VP)msg, &msgsz, ID_mbf, 200 ) != E_OK )
        error(“forced wakeup\n”);
              :
}

}

<< Usage example of the assembly language(CC32R) >>

    msg: .RES.B 30
    .include “mr32r.inc”
    .global   task
task:
        :
    ld24        R2,#msg
    trcv_mbf ID_mbf, 200
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    msg: .space 30
    .include “mr32r.inc”
    .global   task
task:
        :
    ld24        R2,#msg
    trcv_mbf ID_mbf, 200
        :
```

2.6.8. prcv_mbf(Poll and Receive Messagebuffer)

[(System call name)]

prcv_mbf	→	Waits for receiving a message from Messagebuffer. (no wait)

[(Calling by the assembly language)]

```
.include "mr32r.inc"
prcv_mbf     mbfid
```

<< Argument >>

mbfid	[**]	The ID No. of the messagebuffer from which a message pakect is received
msg	[****]	The start address in which a receive message packet is stored. (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the messagebuffer from which a message packet is received
R2	The start address in which a receive message packet is stored.
R3	The size of a receive message packet
R4	TImeout value

[(Calling by the C language)]

```
#include <mr32r.h>
ER prcv_mbf (msg, p_msgsz, mbfid);
```

<< Argument >>

ID	mbfid;	The ID No. of the messagebuffer from which a message packet is received
INT	*p_msgsz;	The start address in which a receive message packet is stored.
VP	msg;	The size of a receive message packet

<< Return value >>

An error code is returned as the return value of a function.
The size of a receive message packet is stored in p_msgsz.

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_TMOUT      0FFFFFFABH(-H'00000055): Polling failed or timeout
E_NOEXS      0FFFFFFCCH(-H'00000034): Object does not exist
```

[(Function description)]

prcv_mbf recieves the message in the message buffer specified as mbfid,and store to the area specified as msg.If there is a send WAIT state task,prcv_mbf compare the size of message to be sent with the size of received message.

1. Recieve message size is larger than send message size.

The message is sent to messagebuffer and the task is moved WAIT state to READY state.

2. Receive message size is smaller than send message size.
   rcv_mbf does not send the message to message buffer,leave the task WAIT state, and end.

After the process of 1,prcv_mbf does as the sam if there is the send WAIT state task.If the message is not sent to the message buffer speified as mbfid,error E_TMOUT is returned..

The task is not moved to WAIT state by prcv_mbf.

An error E_NOEXS is returned if this system call is issued for a nonexistent messagebuffer.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task(void)
{
    char   msg[128];
    INT    msgsz;
    if( prcv_mbf( (VP)msg, &msgsz, ID_mbf, 200 ) != E_OK )
                :
}

<< Usage example of the assembly language(CC32R) >>

msg: .space 32
    .include “mr32r.inc”
    .global   task
task:
    ld24        R2,#msg
    prcv_mbf ID_mbf
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

msg: .space 32
    .include “mr32r.inc”
    .global   task
task:
    ld24        R2,#msg
    prcv_mbf ID_mbf
        :
```

2.6.9. ref_mbf(Refer Messagebuffer Status)

[(System call name)]

ref_mbf → Reference Messagebuffer Status

[(Calling by the assembly language)]

```
.include "mr32r.inc"
ref_mbf mbfid
```

<< Argument >>

mbfid	[**]	The ID No. of the messagebuffer to Reference Messagebuffer
pk_rmbf	[****]	Packet address to Reference Messagebuffer (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the messagebuffer to Reference Messagebuffer
R2	Packet address to Reference Messagebuffer
R3	--

The structure indicated by pk_rmbx returns the following data.

Offset	Size		
+0	4	exinf	Extended information
+4	2	wtsk	Waiting Task Information
+6	2	stsk	Sending Task Information
+8	4	pk_msg	Message Size (in bytes)
12	4	frbufsz	Free Buffer Size (in bytes)

[(Calling by the C language)]

```
#include <mr32r.h>
ER ref_mbf (pk_rmbf, mbfid);
```

<< Argument >>

T_RMBF	* pk_rmbf;	Packet address to Reference Messagebuffer
ID	mbfid;	The ID No. of the messagebuffer to Reference Messagebuffer

<< Return value >>

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

```
typedef struct t_rmbx {
    VP      exinf;  /* Extended informatio */
    BOOL_ID wtsk;   /* Waiting Task Information */
    BOOL_ID stsk;   /* Sending Task Information */
    INT     msgsz;  /* Message Size (in bytes) */
    INT     frbufsz;        /* Free Buffer Size (in bytes) */
} T_RMBF;
```

[(Error codes)]

```
E_OK            00000000H(-H'00000000): Normal End
E_NOEXS         0FFFFFFCCH(-H'00000034): Object does not exist
```

[(Function description)]

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

- exinf

  Returns extended task information in exinf.

- wtsk

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

- stsk

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

- msgsz

  The top message size in the message buffer specified as mbfid is stored.If there is no message in the message buffer, FALSE(0) is returned.

- frbufsz

  The free buffer size specified is returned.

An error E_NOEXS is returned if this system call is issued for a nonexistent messagebuffer.

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

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    T_RMBF rmbf;
        :
    ref_mbf(ID_mbf, &rmbf);
        :
}

<< Usage example of the assembly language(CC32R) >>

rmbf:    .RES.B 16
    .include mr32r.inc
    .global task
task:
        :
    ld24      R2,#rmbf
    ref_mbf    ID_mbf
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

rmbf:    .space 16
    .include mr32r.inc
    .global task
task:
        :
    ld24      R2,#rmbf
    ref_mbf    ID_mbf
        :
```

2.7. Rendezvous

2.7.1. cre_por(Create Port for Rendezvous)

[(System call name)]

cre_por → Create Port for Rendezvous

[(Calling by the assembly language)]

```
.include "mr32r.inc"
cre_por     porid
```

<< Argument >>

porid	[**]	The ID No. of a port for redenzvous to be created
pk_cpor	[****]	The start address in which the port for redenzvous generation information is stored (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a port for redenzvous to be created
R2	The start address in which the port for redenzvous generation information is stored
R3	--

Specify the following information in the structure indicased by pk_cpor.

Offset	Size		
+0	4	exinf	Extended information
+4	4	poratr	Port for rendezvous attribute
+8	4	maxcmsz	Maximum call message size
+12	2	maxrmsz	Maximum reply message size

[(Calling by the C language)]

```
#include <mr32r.h>
ER cre_por (porid, pk_cpor);
```

<< Argument >>

ID	porid;	The ID No. of a port for redenzvous to be created
T_CPOR	*pk_cpor;	The start address in which the port for redenzvous generation information is stored

Specify the following information in the structure indicased by pk_ctsk.

```
typedef struct t_cpor {
    VP      exinf;  /* Extended information */
    ATR     poratr; /* Port for redenzvous attribute */
    INT     maxcmsz;        /* Maximum call message size */
    INT     maxrmsz;        /* Maximum reply message size */
} T_CPOR;
```

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H'00000000): Normal End
E_OBJ           0FFFFFFC1H(-H'0000003f): Invalid object state
```

[(Function description)]

Creates a port for rendezvous porid indicates.

Here follows explanation of the information as to a port for rendezvous to be generated pk_cpor.

- exinf (extended information)

  Exinf is an area you can freely use to store information as to a port for rendezvous to be generated. MR32R has nothing to do with the exinf's contents.

- poratr (port attribute)

  MR32R has nothing to do with this contents.

- maxcmsz

  Specify the maximum length of message in calling.

  MR32R does not refer maxcmsz,so you need not set this item.If you want to have a compatibility between MR32R and other Realtime OS,set this item.

- maxrmsz

  Specify the maximum length of message in replying.

  MR32R does not refer maxrmsz,so you need not set this item.If you want to have a compatibility between MR32R and other Realtime OS,set this item.

An error E_OBJ is returned if cre_por system call is issued for the port which is existent.

The range of the specifiable ID number is 1 to the maximum value specified in the configuration file.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task1()
{
    T_CPOR setpor;
        :
    setmbf.maxcmsz = 300;
    setmbf.maxrmsz = 200;
    cre_por( 1, &setpor );
        :
}

<< Usage example of the assembly language(CC32R) >>

    .equ       ID_por1,1
setpor: .space 16
    .include “mr32r.inc”
    .global  task1
task1:
        :
    ld24   R2,#setpor
    ld24   R1,#300
    st     R1,@(8,R2)
    ld24   R1,#200
    st     R1,@(12,R2)
    cre_por ID_por1
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .equ       ID_por1,1
setpor: .space 16
    .include “mr32r.inc”
    .global  task1
task1:
        :
    ld24   R2,#setpor
    ld24   R1,#300
    st     R1,@(8,R2)
    ld24   R1,#200
    st     R1,@(12,R2)
    cre_por ID_por1
        :
    ext_tsk
```

2.7.2. del_por(Delete Port for Rendezvous)

[(System call name)]

del_por → Delete Port for Rendezous

[(Calling by the assembly language)]

```
.include “mr32r.inc”
del_por     porid
```

<< Argument >>

porid	[**]	The ID No. of a port for rendezvous to be deleted

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a port for rendezvous to be deleted
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER del_por ( porid );
```

<< Argument >>

ID	porid;	The ID No. of a port for rendezvous to be deleted

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
E_NOEXS         0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

del_por deletes the port for rendezvous porid indicates.

You can create the port deleted as the same ID again.If the task is linked to the port wait queue and del_por is issued for the port,this system call normally end.In this case,del_por moves the task WAIT state to READY state.And error E_DLT is returned.

An error E_NOEXS is returned if this system call is issued for a nonexistent port for rendezvous.

Make sure this system call is issued for only the port for rendezvous that has been created by the cre_por system call. If this system call is issued for the port for rendezvous that has been defined by the configuration file, it does not function normally.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
#define ID_por2 2
void task1()
{
        :
    del_por( ID_por2 );
        :
    ext_tsk();
}

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task1
task1:
        :
    del_por ID_por2
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task1
task1:
        :
    del_por ID_por2
        :
    ext_tsk
```

2.7.3. cal_por(Call Port for Rendezvous)

[(System call name)]

cal_por → Call Port for Rendezous

[(Calling by the assembly language)]

```
.include "mr32r.inc"
cal_por     porid, calptn, cmsgsz
```

<< Argument >>

porid	[**]	The ID No. of a port for rendezvous
msg	[****]	The start address of a call message packet (Set the address in the R5 register.)
calptn	[****]	Call bit pattern representing Rendezvous condition
cmsgsz	[****]	The size of a call message packet

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a port for rendezvous
R2	The size of a message to reply
R3	The size of a message to call
R4	--
R5	The start address of a call message packet
R6	Call bit pattern representing Rendezvous condition

[(Calling by the C language)]

```
#include <mr32r.h>
ER cal_por (msg, p_rmsgsz, porid, calptn, cmsgsz);
```

<< Argument >>

VP	msg;	The start address of a call message packet
INT	*p_rmsgsz;	The start address in which the size of a message to reply is stored
ID	porid;	The ID No. of a port for redenzvous
UINT	calptn;	Call bit pattern representing Rendezvous condition
INT	cmsgsz;	The size of a call message packet

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_RLWAI       0FFFFFFAAH(-H'00000056): Wait state forcibly
                                       cleared
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
E_DLT         0FFFFFFAFH(-H'00000051): The object being waited for
                                       was deleted
```

[(Function description)]

This system call executes the port rendezvous call.

In the port specified with porid,there is a task in the rendezvous receive wait state.When conditions for establishing a rendezvous between the task in the receive wait state and the task issuing this system call are satisfied, then the rendezvous is established.Whether a rendezvous is established or not is determined by the logical AND of the call task calptn and the receive task acpptn. The rendezvous is established when the logical AND is anything other than 0.

When the rendezvous is established, the task in the rendezvous receive wait state changes from the WAIT state to the READY state, while the task issued with this system call changes to the rendezvous end wait state. The wait state of this former task is canceled when the rendezvous receive task executes rpl_rdv. At the same time, the cal_pol system call ends.

When the rendezvous is established, the call task can send a message to the receive task. The receive task copies the message stored in the msg address into the message storage area specified with acp_por (only the bytes specified in cmsgsz are copied). Also, a reply message can be sent from the receive task to the call task when the rendezvous ends. The content of the reply message specified by the receive task with rpl_rdv is copied into the message storage area indicated by the msg which the call task specifies with cal_por.

The rendezvous call and rendezvous receive tasks copy each other's message. For this reason, previous messages are lost.When a receive wait task does not exist in the port specified with porid or when the rendezvous establish conditions are not established even though the port has a receive wait task, the task issued with this system call sets the port to the call wait state and connects it to the call wait queue.

In the following cases, the task is freed from a wait state.

- A rel_wai system call issued by some other task
  Error E_RLWAI is returned.
- A del_por system call issued by some other task for this port
  Error E_DLT is returned.

0 cannot be specified for the calptn index of this system call.However, an error is not generated when 0 is specified.

If the port does not exist, an error E_NOEXS is returned.

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.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task(void)
{
    char msg[128];
    INT  rmsgsz;
        :
    cal_por((VP)msg, &rmsgsz, ID_por1, 0x3, 10);
        :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task
msg:  .RES.B  128
task:
    ld24       R5,#msg
    cal_por ID_por1,0x3,10
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task
msg:  .space  128
task:
    ld24       R5,#msg
    cal_por ID_por1,0x3,10
        :
```

2.7.4. tcal_por(Call Port for Rendezvous with Timeout)

[(System call name)]

tcal_por → Call Port for Rendezous (With Timeout)

[(Calling by the assembly language)]

```
.include “mr32r.inc”
tcal_por     porid, msg, calptn, cmsgsz, tmout
```

<< Argument >>

porid	[**]	The ID No. of a port for rendezvous
msg	[****]	The start address of a call message packet (Set the address in the R5 register.)
calptn	[****]	Call bit pattern representing Rendezvous condition
cmsgsz	[****]	The size of a call message packet
tmout	[****]	Timeout value

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a port for rendezvous
R2	The size of a message to reply
R3	The size of a call message packet
R4	Timeout value
R5	The start address of a call message packet
R6	Call bit pattern representing Rendezvous condition

[(Calling by the C language)]

```
#include <mr32r.h>
ER tcal_por (msg, p_rmsgsz, porid, calptn, cmsgsz, tmout);
```

<< Argument >>

VP	msg;	The start address of a call message packet
INT	*p_rmsgsz;	The start address in which the size of a message to reply is stored
ID	porid;	The ID No. of a port for redenzvous
UINT	calptn;	Call bit pattern representing Rendezvous condition
INT	cmsgsz;	The size of a call message packet
TMO	timeout	Timeout value

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_RLWAI       0FFFFFFAAH(-H’00000056): Wait state forcibly
                                       cleared
E_TMOUT       0FFFFFFABH(-H’00000055): Polling failed or timeout
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
E_DLT         0FFFFFFAFH(-H’00000051): The object being waited for
                                       was deleted
```

[(Function description)]

This system call executes the port rendezvous call.

In the port specified with porid,there is a task in the rendezvous receive wait state.When conditions for establishing a rendezvous between the task in the receive wait state and the task issuing this system call are satisfied, then the rendezvous is established.Whether a rendezvous is established or not is determined by the logical AND of the call task calptn and the receive task acpptn. The rendezvous is established when the logical AND is anything other than 0.

When the rendezvous is established, the task in the rendezvous receive wait state changes from the WAIT state to the READY state, while the task issued with this system call changes to the rendezvous end wait state. The wait state of this former task is canceled when the rendezvous receive task executes rpl_rdv. At the same time, the tcal_pol system call ends.

When the rendezvous is established, the call task can send a message to the receive task. The receive task copies the message stored in the msg address into the message storage area specified with acp_por (only the bytes specified in cmsgsz are copied). Also, a reply message can be sent from the receive task to the call task when the rendezvous ends. The content of the reply message specified by the receive task with rpl_rdv is copied into the message storage area indicated by the msg which the call task specifies with tcal_por.

The rendezvous call and rendezvous receive tasks copy each other's message. For this reason, previous messages are lost.

If there is no task waiting for acceptance at the port specified by porid, or although there is a task waiting for acceptance the rendezvous establishment condition is not met (i.e., AND result = 0), the task that issued this system call is kept waiting on the calling side of this port, so it is queued up in two queues: the call wait queue and the timeout wait queue. The sequence of wait queues is FIFO.

In the following cases, the WAIT state by tcal_por system call issue is canceled.The task canceled WAIT state exits from the two wait queues (rendezvous call 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 E_OK is returned.

- When tmout time has elapsed without any message being received

  Error 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 E_RLWAI is returned.

- When the port for which a task has been kept waiting is deleted by the del_por system call issued by another task

  Error E_DLT is returned.

Any value from -1 to 7FFFFFFFH can be specified for tmout. If you specify TMO_POL = 0 for tmout, the effect is the same as 0 is specified for the timeout value, in which case tmout functions the same way as pcal_por. Also, if you specify tmout = TMO_FEVR(-1), the effect is the same as endless wait is specified, in which case tmout functions the same way as cal_por.

An error E_NOEXS is returned if this system call is issued for a nonexistent port for rendezvous.

0 cannot be specified for the calptn index of this system call.However, an error is not generated when 0 is specified.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
#define ID_por1 1
void task(void)
{
    char msg[128];
    INT  rmsgsz;
    tcal_por((VP)msg, &rmsgsz, ID_por1, 0x3, 10, 300);
        :
}

<< Usage example of the assembly language(CC32R) >>

msg:  .RES.B  128
    .equ    ID_por1,1
    .include “mr32r.inc”
    .global  task
task:
        :
    ld24       R5,#msg
    tcal_por ID_por1,0x3,10,300
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

msg:  .space  128
    .equ    ID_por1,1
    .include “mr32r.inc”
    .global  task
task:
        :
    ld24       R5,#msg
    tcal_por ID_por1,0x3,10,300
        :
```

2.7.5. pcal_por(Poll and Call Port for Rendezvous)

[(System call name)]

pcal_por	→	Call Port for Rendezous (no wait)

[(Calling by the assembly language)]

```
.include “mr32r.inc”
pcal_por     porid, calptn, cmsgsz, tmout
```

<< Argument >>

porid	[**]	The ID No. of a port for rendezvous
msg	[****]	The start address of a call message packet (Set the address in the R5 register.)
calptn	[****]	Call bit pattern representing Rendezvous condition
cmsgsz	[****]	The size of a call message packet

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a port for rendezvous
R2	The size of a message to reply
R3	The size of a call message packet
R4	--
R5	The start address of a call message packet
R6	Call bit pattern representing Rendezvous condition

[(Calling by the C language)]

```
#include <mr32r.h>
ER pcal_por (msg, p_rmsgsz, porid, calptn, cmsgsz);
```

<< Argument >>

VP	msg;	The start address of a call message packet
INT	*p_rmsgsz;	The start address in which the size of a message to reply is stored
ID	porid;	The ID No. of a port for redenzvous
UINT	calptn;	Call bit pattern representing Rendezvous condition
INT	cmsgsz;	The size of a call message packet

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_TMOUT       0FFFFFFABH(-H’00000055): Polling failed or timeout
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

This system call executes the port rendezvous call.

In the port specified with porid,there is a task in the rendezvous receive wait state.When conditions for establishing a rendezvous between the task in the receive wait state and the task issuing this system call are satisfied, then the rendezvous is established.Whether a rendezvous is established or not is determined by the logical AND of the call task calptn and the receive task acpptn. The rendezvous is established when the logical AND is anything other than 0.

When the rendezvous is established, the task in the rendezvous receive wait state changes from the WAIT state to the READY state, while the task issued with this system call changes to the rendezvous end wait state. The wait state of this former task is canceled when the rendezvous receive task executes rpl_rdv. At the same time, the pcal_por system call ends.

When the rendezvous is established, the call task can send a message to the receive task. The receive task copies the message stored in the msg address into the message storage area specified with acp_por (only the bytes specified in cmsgsz are copied). Also, a reply message can be sent from the receive task to the call task when the rendezvous ends. The content of the reply message specified by the receive task with rpl_rdv is copied into the message storage area indicated by the msg which the call task specifies with pcal_por.

The rendezvous call and rendezvous receive tasks copy each other's message. For this reason, previous messages are lost.

When there is no receive wait task linked to the port specified by porid or rendezvous condition does not match, error E_TMOUT is returned to the task that issued this system call and the system call.

The task which issued pcal_por system call is not moved to WAIT state unlike cal_por and tcal_po.

Error E_NOEXS is returned if this system call is issued for a nonexistent port for rendezvous.

0 cannot be specified for the calptn index of this system call.However, an error is not generated when 0 is specified.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
#define ID_por1 1
void task()
{
    char msg[128];
    INT  rmsgsz;
        :
    pcal_por((VP)msg, &rmsgsz, ID_por1, 0x3, 10);
        :
}
```

```
<< Usage example of the assembly language(CC32R) >>

msg:   .RES.B   128
    .equ    ID_por1,1
    .include “mr32r.inc”
    .global   task
task:
        :
    ld24        R5,#msg
    pcal_por ID_por1,0x3,10
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

msg:   .space   128
    .equ    ID_por1,1
    .include “mr32r.inc”
    .global   task
task:
        :
    ld24        R5,#msg
    pcal_por ID_por1,0x3,10
        :
```

2.7.6. acp_por(Accept Port for Rendezvous)

[(System call name)]

acp_por → Accept Port for Rendezous

[(Calling by the assembly language)]

```
.include “mr32r.inc”
acp_por     porid, acpptn
```

<< Argument >>

porid	[**]	The ID No. of a port for rendezvous
acpptn	[****]	Accept bit pattern representing Rendezvous condition
msg	[****]	The start address in which a call message packet is stored. (Set the address in the R5 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a port for rendezvous
R2	The size of a call message packet
R3	Rendezvous number
R4	--
R5	The start address in which a call message packet is stored.
R6	Accept bit pattern representing Rendezvous condition

[(Calling by the C language)]

```
#include <mr32r.h>
ER acp_por (p_rdvno, msg, p_cmsgsz, porid, acpptn );
```

<< Argument >>

RNO	*p_rdvno;	Rendezvous number
VP	msg;	The start address in which a call message packet is stored.
INT	*p_cmsgsz;	The start address in which the size of a call message packet is stored
ID	porid;	The ID No. of a port for redenzvous
UINT	acpptn;	Accept bit pattern representing Rendezvous condition

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_RLWAI       0FFFFFFAAH(-H’00000056): Wait state forcibly
                                       cleared
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
E_DLT         0FFFFFFAFH(-H’00000051): The object being waited for
                                       was deleted
```

[(Function description)]

This system call executes the port rendezvous reception.

In the port specified with porid, there is a task in the rendezvous call wait state. When conditions for establishing a rendezvous between the task in that call wait string and the task issuing this system call are satisfied, then the rendezvous is established.

Whether a rendezvous is established or not is determined by the logical AND of the call task calptn and the receive task acpptn. The rendezvous is established when the logical AND is anything other than 0. When the rendezvous is established, the task in the rendezvous call wait string is detached from the wait string and is changed from the rendezvous call wait state to the rendezvous end wait state.

When a call wait task does not exist in the port specified with porid or when the rendezvous establish conditions are not established even though the port has a call wait task, the task issued with this system call is set the port to the recieve wait state and connects it to the recieve wait queue.

In the following cases, the task is freed from a wait state.

- A rel_wai system call issued by some other task
  Error E_RLWAI is returned.
- A del_por system call issued by some other task for this port
  Error E_DLT is returned.

This system call can be issued again before the rendezvous reply (before the rpl_rdv that indicates that the rendezvous has been established is issued).As such, multiple rendezvous can be executed at the same time. It does not matter in this case whether the port is the same port or a different port from before.

When the rendezvous is established, the call task can send a message to the receive task. The receive task copies the message stored in the msg address into the message storage area specified with acp_por (only the bytes specified in cmsgsz are copied). acp_por is returned as the cmsgsz message size.

Also, a reply message can be sent from the receive task to the call task when the rendezvous ends. The content of the reply message specified by the receive task with rpl_rdv is copied into the message storage area indicated by the msg which the call task specifies with cal_por.

rdvno is information for discriminating between rendezvous established at the same time. It is used for the rpl_rdv parameter or fwd_rdv parameter (for forwarding the rendezvous) at rendezvous end.

0 cannot be specified for the calptn index of this system call.However, an error is not generated when 0 is specified. If the port does not exist, an error E_NOEXS is returned.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
char msg[128];
RNO rdvno;
INT cmsgsz;
        :
acp_por(&rdvno, msg, &cmsgsz, ID_por1, 0x3);
        :
}

<< Usage example of the assembly language(CC32R) >>

msg:   .RES.B   128
    .include “mr32r.inc”
    .global   task
task:
        :
    ld24        R5,#msg
    acp_por ID_por1, 0x3
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

msg:   .space   128
    .include “mr32r.inc”
    .global   task
task:
        :
    ld24        R5,#msg
    acp_por ID_por1, 0x3
        :
```

2.7.7. tacp_por(Accept Port for Rendezvous with Timeout)

[(System call name)]

tacp_por → Accept Port for Rendezous (With Timeout)

[(Calling by the assembly language)]

```
.include "mr32r.inc"
tacp_por     porid, acpptn, tmout
```

<< Argument >>

porid	[**]	The ID No. of a port for rendezvous
acpptn	[****]	Accept bit pattern representing Rendezvous condition
msg	[****]	The start address in which a call message packet is stored. (Set the address in the R5 register.)
tmout	[****]	Timeout value

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a port for rendezvous
R2	The size of a call message packet
R3	Rendezvous number
R4	Timeout value
R5	The start address in which a call message packet is stored.
R6	Accept bit pattern representing Rendezvous condition

[(Calling by the C language)]

```
#include <mr32r.h>
ER tacp_por (p_rdvno, msg, p_cmsgsz, porid, acpptn, tmout);
```

<< Argument >>

RNO	*p_rdvno;	Rendezvous number
VP	msg;	The start address in which a call message packet is stored.
INT	*p_cmsgsz;	The start address in which the size of a call message packet is stored
ID	porid;	The ID No. of a port for redenzvous
UINT	acpptn;	Accept bit pattern representing Rendezvous condition
TMO	timeout;	Timeout value

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_RLWAI       0FFFFFFAAH(-H'00000056): Wait state forcibly
                                       cleared
E_TMOUT       0FFFFFFABH(-H'00000055): Polling failed or timeout
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
E_DLT         0FFFFFFAFH(-H'00000051): The object being waited for
                                       was deleted
```

[(Function description)]

This system call executes the port rendezvous reception.

In the port specified with porid, there is a task in the rendezvous call wait state. When conditions for establishing a rendezvous between the task in that call wait queue and the task issuing this system call are satisfied, then the rendezvous is established.

Whether a rendezvous is established or not is determined by the logical AND of the call task calptn and the receive task acpptn. The rendezvous is established when the logical AND is anything other than 0. When the rendezvous is established, the task in the rendezvous call wait queue is detached from the wait string and is changed from the rendezvous call wait state to the rendezvous end wait state.

When a call wait task does not exist in the port specified with porid or when the rendezvous establish conditions are not established even though the port has a call wait task, the task issued with this system call is set the port to the recieve wait state and connects it to the recieve wait queue and to the time out queue.

In the following cases, the task is freed from a wait state. The task canceled WAIT state exits from the two wait queues (rendezvous call 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 E_OK is returned.

- When tmout time has elapsed without any message being received

  Error 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 E_RLWAI is returned.

- When the port for which a task has been kept waiting is deleted by the del_por system call issued by another task

  Error E_DLT is returned.

This system call can be issued again before the rendezvous reply (before the rpl_rdv that indicates that the rendezvous has been established is issued).As such, multiple rendezvous can be executed at the same time. It does not matter in this case whether the port is the same port or a different port from before.

When the rendezvous is established, the call task can send a message to the receive task. The receive task copies the message stored in the msg address into the message storage area specified with tacp_por (only the bytes specified in cmsgsz are copied). tacp_por is returned as the cmsgsz message size.

Also, a reply message can be sent from the receive task to the call task when the rendezvous ends. The content of the reply message specified by the receive task with rpl_rdv is copied into the message storage area indicated by the msg which the call task specifies with cal_por.

rdvno is information for discriminating between rendezvous established at the same time. It is used for the rpl_rdv parameter or fwd_por parameter (for forwarding the rendezvous) at rendezvous end.

Any value from -1 to 7FFFFFFFH can be specified for tmout. If you specify TMO_POL = 0 for tmout, the effect is the same as 0 is specified for the timeout value, in which case tmout functions the same way as pacp_por. Also, if you specify tmout = TMO_FEVR(-1), the effect is the same as endless wait is specified, in which case tmout functions the same way as

acp_por.

0 cannot be specified for the calptn index of this system call.However, an error is not generated when 0 is specified. If the port does not exist, an error E_NOEXS is returned.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task(void)
{
    char msg[128];
    RNO rdvno;
    INT cmsgsz;
                :
    tacp_por(&rdvno, msg, &cmsgsz, ID_por1, 0x3, 300);
                :
}

<< Usage example of the assembly language(CC32R) >>

msg:  .RES.B  128
    .include “mr32r.inc”
    .global  task
task:
        :
    ld24       R5,#msg
    tacp_por ID_por1, 0x3, 300
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

msg:  .space  128
    .include “mr32r.inc”
    .global  task
task:
        :
    ld24       R5,#msg
    tacp_por ID_por1, 0x3, 300
        :
```

2.7.8. pacp_por(Poll and Accept Port for Rendezvous)

[(System call name)]

pacp_por → Accept Port for Rendezous (no wait)

[(Calling by the assembly language)]

```
.include “mr32r.inc”
pacp_por     porid, acpptn
```

<< Argument >>

porid	[**]	The ID No. of a port for rendezvous
acpptn	[****]	Accept bit pattern representing Rendezvous condition
msg	[****]	The start address in which a call message packet is stored. (Set the address in the R5 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a port for rendezvous
R2	The size of a call message packet
R3	Rendezvous number
R4	--
R5	The start address in which a call message packet is stored.
R6	Accept bit pattern representing Rendezvous condition

[(Calling by the C language)]

```
#include <mr32r.h>
ER pacp_por (p_rdvno, msg, p_cmsgsz, porid, acpptn);
```

<< Argument >>

RNO	*p_rdvno;	Rendezvous number
VP	msg;	The start address in which a call message packet is stored.
INT	*p_cmsgsz;	The start address in which the size of a call message packet is stored
ID	porid;	The ID No. of a port for redenzvous
UINT	acpptn;	Accept bit pattern representing Rendezvous condition

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_TMOUT       0FFFFFFABH(-H’00000055): Polling failed or timeout
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

This system call executes the port rendezvous reception.

In the port specified with porid, there is a task in the rendezvous call wait state. When conditions for establishing a rendezvous between the task in that call wait string and the task issuing this system call are satisfied, then the rendezvous is established.

Whether a rendezvous is established or not is determined by the logical AND of the call task calptn and the receive task acpptn. The rendezvous is established when the logical AND is anything other than 0. When the rendezvous is established, the task in the rendezvous call wait string is detached from the wait string and is changed from the rendezvous call wait state to the rendezvous end wait state.

When a call wait task does not exist in the port specified with porid or when the rendezvous establish conditions are not established even though the port has a call wait task,error E_TMOUT is returned. The task which issues pcal_por system call is not moved to WAIT state unlike acp_por and tacp_por.

This system call can be issued again before the rendezvous reply (before the rpl_rdv that indicates that the rendezvous has been established is issued).As such, multiple rendezvous can be executed at the same time. It does not matter in this case whether the port is the same port or a different port from before.

When the rendezvous is established, the call task can send a message to the receive task. The receive task copies the message stored in the msg address into the message storage area specified with pacp_por (only the bytes specified in cmsgsz are copied). pacp_por is returned as the cmsgsz message size.

Also, a reply message can be sent from the receive task to the call task when the rendezvous ends. The content of the reply message specified by the receive task with rpl_rdv is copied into the message storage area indicated by the msg which the call task specifies with cal_por.

rdvno is information for discriminating between rendezvous established at the same time. It is used for the rpl_rdv parameter or fwd_rdv parameter (for forwarding the rendezvous) at rendezvous end.

0 cannot be specified for the calptn index of this system call.However, an error is not generated when 0 is specified. If the port does not exist, an error E_NOEXS is returned.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    char msg[128];
    RNO rdvno;
    INT cmsgsz;
        :
    pacp_por(&rdvno, msg, &cmsgsz, ID_por1, 0x3);
        :
}
```

```
<< Usage example of the assembly language(CC32R) >>

msg:   .RES.B   128
    .include “mr32r.inc”
    .global   task
task:
        :
    ld24        R5,#msg
    pacp_por ID_por1, 0x3
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

msg:   .space   128
    .include “mr32r.inc”
    .global   task
task:
        :
    ld24        R5,#msg
    pacp_por ID_por1, 0x3
        :
```

2.7.9. fwd_por(Forward Rendezvous to Other Port)

[(System call name)]

fwd_por → Forward Rendezous to Other Port.

[(Calling by the assembly language)]

```
.include "mr32r.inc"
fwd_por     porid, calptn, cmsgsz
```

<< Argument >>

porid	[**]	The ID No. of a port for rendezvous to be forward to
calptn	[****]	Call bit pattern representing Rendezvous condition
rdvno	[****]	Rendezvous Number (Set the rdvno in the R2 register.)
msg	[****]	The start address in which a call message packet is stored (Set the address in the R5 register.)
cmsgsz	[****]	The size of a call message packet

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a port for rendezvous to be forward to
R2	Rendezvous Number
R3	The size of a call message packet
R4	--
R5	The start address in which a call message packet is stored
R6	Call bit pattern representing Rendezvous condition

[(Calling by the C language)]

```
#include <mr32r.h>
ER fwd_por (porid, calptn, rdvno, msg, cmsgsz);
```

<< Argument >>

ID	porid;	The ID No. of a port for rendezvous to be forward to
UINT	calptn;	Call bit pattern representing Rendezvous condition
RNO	rdvno;	Rendezvous Number
VP	msg;	The start address of a call message packet
INT	cmsgsz;	The size of a call message packet

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_RLWAI       0FFFFFFAAH(-H'00000056): Wait state forcibly
                                       cleared
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
E_OBJ         0FFFFFFC1H(-H'0000003f): Invalid object state
```

[(Function description)]

This system call forwards the rendezvous assigned with the rdvno rendezvous No. to the port specified with porid.fwd_por system call can be issued from the rendezvous state task only.

The rendezvous established state of the call task specified with rdvno is canceled and the call task of the porid port is returned to the rendezvous wait state. In other words, the call task changes from the rendezvous end wait state to the rendezvous call state.

With a receive wait task, the rendezvous is established as long as the logical AND of calptn specified with this system call and acpptn of the receive task is not 0. When the rendezvous is established, the message stored in the msg address specified in this system call is copied into the receive task (only cmsgsz portion is copied) and the task assumes the rendezvous end state.

If there is not a receive wait task or if the rendezvous is not established with the destination port, the call task assumes the call wait state.

The task that issues this system call is continuously executed without changing to the wait state, regardless of the call task rendezvous state of the forwarding port.

An error is not generated even when the index of this system call is as follows.

- When cmsgsz exceeds the maxcmsz maximum send size of the port after the message is forwarded
- When cmsgsz exceeds the maxrmsz maximum receive size of the port before the message is forwarded
- When 0 is specified for cmsgsz
- When 0 is specified for calptn

An error E_OBJ is returned when maxrmsz of the port after the message is forwarded is larger than the port's maxrmsz before the message is forwarded.

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.

If the port does not exist, an error E_NOEXS is returned.

This system call can be issued only from a task. It does not function properly when issued from the interrupt handler,cyclic handler or alarm handler.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    char *msg=”abcdef”;
    RNO rdvno;
    INT cmsgsz;
              :
    fwd_por(ID_porid, 0x3, rdvno, (VP)msg, 6);
              :
}

<< Usage example of the assembly language(CC32R) >>

msg: .SDATA “abcdef”
rdvno: .RES.B 4
    .include “mr32r.inc”
    .global  task
```

```
task:
        :
    ld24        R4,#rdvno
    ld          R2,@R4
    ld24        R5,#msg
    fwd_por ID_porid, 0x3, 6
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

msg: .byte “abcdef”
rdvno: .space 4
    .include “mr32r.inc”
    .global   task
task:
        :
    ld24        R4,#rdvno
    ld          R2,@R4
    ld24        R5,#msg
    fwd_por ID_porid, 0x3, 6
        :
```

2.7.10. rpl_rdv(Reply Rendezvous)

[(System call name)]

rpl_rdv → Reply Rendezous

[(Calling by the assembly language)]

```
.include “mr32r.inc”
rpl_rdv     rdvno, msg, rmsgsz
```

<< Argument >>

porid	[****]	Rendevzous Number (Set the address in the R1 register.)
msg	[****]	The start address in which a reply message packet is stored (Set the address in the R5 register.)
rmsgsz	[****]	The size of a reply message packet

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	Rendevzous Number
R2	The start address in which a reply message packet is stored
R3	The size of a reply message packet

[(Calling by the C language)]

```
#include <mr32r.h>
ER rpl_rdv (rdvno, msg, rmsgsz);
```

<< Argument >>

RNO	rdvno;	Rendevzous Number
VP	msg;	The start address of a reply message packet
INT	rmsgsz;	The size of a reply message packet

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000):  Normal End
E_OBJ         0FFFFFFC1H(-H’0000003f): Invalid object state
```

[(Function description)]

This system call returns a reply to the call task and ends the rendezvous. rpl_rdv system call can be issued from the rendezvous state task only.

The call task wait state is canceled and the message (only rmsgsz portion is copied) stored in the msg address is copied into the msg address specified by cal_por of the call task. The call task changes from the end wait state to the ready state.

An error is not returned even when the index of this system call is as follows. Checks must be made from the user's side.

- When a value other than 0 is specified for rmsgsz
- When rmsgsz exceeds the relay maximum message size maxrmsz

If an established rendezvous fails for some reason before ending (before rpl_rdv is executed), it is not possible to know directly the rendezvous receive task. In such case, the rendezvous receive task results as an error E_OBJ when this system call is executed.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    char *msg=”abcdef”;
    RNO rdvno;
    INT rmsgsz;
        :
    rpl_rdv(rdvno, (VP)msg, 6);
        :
}

<< Usage example of the assembly language(CC32R) >>

msg: .SDATA “abcdef”
rdvno: .RES.B 4
    .include “mr32r.inc”
    .global  task
task:
        :
    ld24       R5,#rdvno
    ld         R1,@R5
    ld24       R5,#msg
    rpl_rdv 6
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

msg: .byte “abcdef”
rdvno: .space 4
    .include “mr32r.inc”
    .global  task
task:
        :
    ld24       R5,#rdvno
    ld         R1,@R5
    ld24       R5,#msg
    rpl_rdv 6
        :
```

2.7.11. ref_por(Refer Port Status)

[(System call name)]

ref_por → Reference Port Status

[(Calling by the assembly language)]

```
.include “mr32r.inc”
ref_por porid
```

<< Argument >>

porid	[**]	The ID No. of the messagebuffer to Reference Port
pk_rpor	[****]	Packet address to Reference Port (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the port to Reference Port for Rendevzous
R2	Packet address to Reference Port
R3	--

The structure indicated by pk_rmbx returns the following data.

Offset	Size		
+0	4	exinf	Extended information
+4	2	wtsk	Waiting Task Information
+6	2	atsk	Accepting Task Information

[(Calling by the C language)]

```
#include <mr32r.h>
ER ref_por (pk_rpor, porid);
```

<< Argument >>

T_RPOR	* pk_rpor;	Packet address to Reference Port
ID	porid;	The ID No. of the port to Reference Port for Rendevzous

<< Return value >>

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

```
typedef struct t_rpor {
    VP      exinf;  /* Extended informatio */
    BOOL_ID wtsk;   /* Waiting Task Information */
    BOOL_ID atsk;   /* Accepting Task Information */
} T_RPOR;
```

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
E_NOEXS         0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

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

- exinf

  Returns extended task information in exinf.

- wtsk

  wtsk returns the ID No. of the first task connected to the rendezvous call wait queue. wtsk returns FALSE (0) if there are no tasks connected to the rendezvous call wait queue.

- atsk

  atsk returns the ID No. of the first task connected to the rendezvous recieve wait queue. atsk returns FALSE (0) if there are no tasks connected to the rendezvous recieve wait queue.

An error E_NOEXS is returned if this system call is issued for a nonexistent port for rendezvous.

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

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task(void)
{
    T_RPOR rpor;
        :
    ref_por(&rpor , ID_por);
        :
}

<< Usage example of the assembly language(CC32R) >>

rpor:    .RES.B 8
    .include mr32r.inc
    .global task
task:
        :
    ld24      R2,#rpor
    ref_por    ID_por
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

rpor:    .space 8
    .include mr32r.inc
    .global task
task:
        :
    ld24      R2,#rpor
    ref_por    ID_por
        :
```

2.8. Interrupt Management Function

2.8.1. def_int(Define Interrupt Handler)

[(System call name)]

def_int → Define Interrupt Handler.

[(Calling by the assembly language)]

```
.include “mr32r.inc”
def_int     dintno
```

<< Argument >>

dintno	[****]	The vectore No. of an interrupt handler
pk_dint	[****]	The start address in which the interrupt hundler defined information is stored (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The vectore No. of an interrupt handler
R2	The start address in which the interrupt hundler defined information is stored
R3	--

Specify the following information in the structure indicased by pk_cmpf.

Offset	Size		
+0	4	intatr	Interrupt handler attribute
+4	4	inthdr	Interrupt handler startup address

[(Calling by the C language)]

```
#include <mr32r.h>
ER def_int (dintno, pk_dint);
```

<< Argument >>

ID	dintno;	The vectore No. of an interrupt handler
T_DINT	*pk_dint;	The start address in which the interrupt hundler defined information is stored

Specify the following information in the structure indicased by pk_dint.

```
typedef struct t_dint {
    ATR     intatr; /* Interrupt handler attribute */
    FP      inthdr; /* Interrupt handler startup address */
} T_DINT;
```

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
```

[(Function description)]

This system call defines the interrupt handler with the interrupt No. specified with dintno and enables the interrupt handler.In other words, it sets the interrupt entry area in the interrupt vector area specified with dintno.

- intatr (interrupt handler attribute)
  The MR32R has nothing to do with this setting.
- inthdr
  The entry address of the interrupt handler to be defined is set in this area. pk_dint.inthdr = NADR (--1) cancels the previously defined interrupt handler.

An interrupt handler can be specified for an interrupt No. that has already been defined; before redefining an interrupt handler, it is not necessary to cancel the definition. An error is not generated when a new handler is defined for a defined interrupt handler No.

This system call cannot be issued if the interrupt vector table is allocated in RAM.

This system call can be issued only from a task. It does not function properly when issued from an interrupt handler, cyclic handler or alarm handler.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void intr(void);
void task1()
{
    T_DINT setint;
                :
    setint.inthdr = _intr; /* Interrupt handler startup address */
    def_int( 15, &setint );
                :
}

<< Usage example of the assembly language(CC32R) >>

setint: .RES.B 12
    .include “mr32r.inc”
    .global   task1
task1:
        :
    ld24    R2,#setint
    ld24    R1,#_intr
    st      R1,@(4,R2)
    def_int  15
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

setint: .RES.B 12
    .include “mr32r.inc”
    .global   task1
task1:
        :
    ld24    R2,#setint
    ld24    R1,#_intr
    st      R1,@(4,R2)
    def_int  15
        :
    ext_tsk
```

2.8.2. ret_int(Return from Interrupt Handler)

[(System call name)]

ret_int → Returns from the interrupt handler.

[(Calling by the assembly language)]

```
.include “mr32r.inc”
ret_int
```

<< Argument >>

None

<< Register setting >>

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

[(Calling by the C language)]

Cannot describe this system call in C language.

[(Error codes)]

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

[(Function description)]

The system call ret_int leads to the definition of a macro named "jmp r14" so as to ensure compatibility with an OS that uses μITRON or with future updates. If you include the interrupt handler in a program written in assembly language, put ret_int at the end of the interrupt handler.

2.8.3. loc_cpu(Lock CPU)

[(System call name)]

loc_cpu → Disables interrupts and task dispatch.

[(Calling by the assembly language)]

```
.include “mr32r.inc”
loc_cpu
```

<< Argument >>

None

<< Register setting >>

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

[(Calling by the C language)]

```
#include <mr32r.h>
ER loc_cpu ();
```

<< Argument >>

None

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
```

[(Function description)]

loc_cpu disables external interrupts and task dispatches.

After this system call is executed, interrupts and dispatches are disabled until unl_cpu is executed. And there can be no chances that the task that executed loc_cpu is preempted (the CPU's execution right is intercepted by something higher in priority) by the interrupt handler or by another task. That is, interrupt requests after the execution of loc_cpu or dispatches generated by a system call issued by a task that executed loc_cpu are made to wait until unl_cpu releases the interrupts and dispatches from the disabled condition.

If a task already in the state that has disabled interrupts and dispatches issues loc_cpu, the same state is kept and no error occurs. Issuing unl_cpu once after having issued loc_cpu twice or more releases the condition in which interrupts and dispatches are disabled.

You cannot issue this system call from a section independent of tasks (the interrupt handler, the cyclic handler, the alarm handler).

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
       :
     loc_cpu();
       :
}

<< Usage example of the assembly language(CC32R) >>

.include “mr32r.inc”
.global  task
task:
       :
     loc_cpu
       :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

.include “mr32r.inc”
.global  task
task:
       :
     loc_cpu
       :
```

2.8.4. unl_cpu(Unlock CPU)

[(System call name)]

unl_cpu → Enables interrupts and task dispatch.

[(Calling by the assembly language)]

```
.include “mr32r.inc”
unl_cpu
```

<< Argument >>

None

<< Register setting >>

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

[(Calling by the C language)]

```
#include <mr32r.h>
ER unl_cpu ();
```

<< Argument >>

None

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H’00000000): 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. 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 (e.g., interrupt handler, cyclic handler, or alarm handler).

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
       :
    unl_cpu();
       :
}
```

```
<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task
task:
       :
    unl_cpu
       :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task
task:
       :
    unl_cpu
       :
```

2.9. Memorypool Management Function

2.9.1. cre_mpf(Create Fixed-size Memorypool)

[(System call name)]

cre_mpf → Create Fixed-size Memorypool

[(Calling by the assembly language)]

```
.include "mr32r.inc"
cre_mpf     mpfid
```

<< Argument >>

mpfid	[**]	The ID No. of a fixed-size memorypool to be created
pk_cmpf	[****]	The start address in which the fixed-size memorypool generation information is stored (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a fixed-size memorypool to be created
R2	The start address in which the fixed-size memorypool generation information is stored
R3	--

Specify the following information in the structure indicased by pk_cmpf.

Offset	Size		
+0	4	exinf	Extended information
+4	4	mpfatr	Fixed-size memorypool attribute
+8	4	mpfcnt	Memory block count
+12	4	blfsz	Fixed-size memorypool size

[(Calling by the C language)]

```
#include <mr32r.h>
ER cre_mpf (mpfid, pk_cmpf);
```

<< Argument >>

ID	mpfid;	The ID No. of a fixed-size memorypool to be created
T_CMPF	*pk_cmpf;	The start address in which the fixed-size memorypool generation information is stored

Specify the following information in the structure indicased by pk_cmpf.

```
typedef struct t_cmpf {
    VP      exinf;  /* Extended information */
    ATR     mpfatr; /* Fixed-size memorypool attribute */
    INT     mpfcnt; /* Memory block count */
    INT     maxblksz;       /* Fixed-size memorypool size */
} T_CMPF;
```

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H'00000000): Normal End
E_NOMEM         0FFFFFFF6H(-H'0000000a): Not enough of memory
E_OBJ           0FFFFFFC1H(-H'0000003f): Invalid object state
```

[(Function description)]

This system call creates a fixed-length memory pool that bears the ID number specified by mpfid. It allocates a memory area to be used as the memory pool and initializes the created memory pool's management block data.

Here follows explanation of the information as to a fixed-size memorypool to be generated pk_cmpf.

- exinf (extended information)

  Exinf is an area you can freely use to store information as to a fixed-size memorypool to be generated. MR32R has nothing to do with the exinf's contents.

- mpfatr (fixed-size memorypool attribute)

  Specify the location of the fixed-size memorypool area to be created. Specifically this means specifying whether you want fixed-size memorypool to be located in the internal RAM or in external RAM.

  - **To locate the fixed-size memorypool area in internal RAM**
    Specify __MR_INT(0).
  - **To locate the fixed-size memorypool area in external RAM**
    Specify __MR_EXT(0x10000).
  - **To locate the fixed-size memorypool area user specified**
    Specify __MR_USER(0x30000).

- mpfcnt

  Specify the number of blocks in the memory pool to be created. Any value from 1 to 32 can be specified here.

- blfsz

  Specify the block size of one block in the memory pool to be created.

The ID numbers that can be specified in this system call range from 1 to the maximum number of fixed-length memory pools in the user system that you set when defining the maximum number of items.

An error E_NOMEM is returned if the memory pool does not have a sufficient memory space to accommodate mpfcnt x blfsz.

An error E_OBJ is returned if cre_mpf system call is issued for the fixed-size memorypool which is existent.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
#define ID_mpf1 1
void task(void)
{
    T_CMPF   setmpf;
        :
    setmpf.mpfatr = __MR_INT;
    setmpf.mpfcnt = 20;
    setmpf.blfsz = 400;
    cre_mpf(ID_mpf1,&setmpf);
        :
}

<< Usage example of the assembly language(CC32R) >>

setmpf: .RES.B   16
ID_mpf1:        .equ    1
    .include “mr32r.inc”
    .global task
task:
    ld24      R2,#setmpf
    ld24      R1,#__MR_INT
    st        R1,@(4,R2)
    ld24      R1,#20
    st        R1,@(8,R2)
    ld24      R1,#400
    st        R1,@(12,R2)
    cre_mpf   ID_mpf1
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

setmpf: .space  16
    .equ ID_mpf1,1
    .include “mr32r.inc”
    .global task
task:
    ld24      R2,#setmpf
    ld24      R1,#__MR_INT
    st        R1,@(4,R2)
    ld24      R1,#20
    st        R1,@(8,R2)
    ld24      R1,#400
    st        R1,@(12,R2)
    cre_mpf   ID_mpf1
        :
    ext_tsk
```

2.9.2. del_mpf(Delete Fixed-size Memorypool)

[(System call name)]

del_mpf → Delete Fixed-size Memorypool

[(Calling by the assembly language)]

```
.include “mr32r.inc”
del_mpf     mpfid
```

<< Argument >>

mpfid	[**]	The ID No. of a fixed-size memorypool to be deleted

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a fixed-size memorypool to be deleted
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER del_mpf (mpfid);
```

<< Argument >>

ID	mpfid;	The ID No. of a fixed-size memorypool to be deleted

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
E_NOEXS        0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

This system call deletes the fixed-length memory pool that bears the ID number specified by mpfid. Once this memorypool is deleted, you can create a new memorypool with the same ID number. Even when there is any task waiting for a memory block in the memorypool to be deleted, this system call is terminated normally. In this case, the said task is freed from the memory block wait state and returns error E_DLT before entering an RUN or READY state.

An error E_NOEXS is returned if this system call is issued for a nonexistent fixed-size memorypool.

Make sure this system call is issued for only the fixed-size memorypool that has been created by the cre_mpf system call. If this system call is issued for the fixed-size memorypool that has been defined by the configuration file, it does not function normally.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task(void)
{
        :
    del_mpf(ID_mpf1);
        :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global task
task:
        :
    del_mpf ID_mpf1
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global task
task:
        :
    del_mpf ID_mpf1
        :
    ext_tsk
```

2.9.3. get_blf(Get Fixed-size Memory Block)

[(System call name)]

get_blf → Gets a fixed-size memory block

[(Calling by the assembly language)]

```
.include “mr32r.inc”
get_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 ID No. of the memorypool to be obtained
R2	The start address of memory block to be obtained
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER get_blf (p_blf,mpfid);
```

<< Argument >>

ID	mplid;	The ID No. of the memorypool to be obtained
VP	*p_blf;	The start address of memory block to be obtained

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_RLWAI       0FFFFFFAAH(-H’00000056): Wait state forcibly
                                       cleared
E_DLT         0FFFFFFAFH(-H’00000051): The object being waited for
                                       was deleted
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

This system call acquires a memory block from the memorypool indicated by mpfid and stores the start address of the acquired memory block in the variable p_blf. The content of the acquired memory block is indeterminate.

If no memory block exists in the specified memorypool, the task that issued this system call goes to a memory block wait state,it is connected to the memory block wait queue in FIFO order.

When the rel_wai or irel_wai system call is issued before the send fixed-size memorypool wait condition is met, error code E_RLWAI is returned.

When the fixed-size memorypool for which a task has been kept waiting is deleted by the del_mpf system call issued by another task, error code E_DLT is returned and it is moved to READY state.

When the fixed-size memorypool for which a task has been kept waiting is reset by the vrst_mpf system call issued by another task, error code EV_RST is returned and it is moved to READY state.

Memory blocks are fixed in length. Use the configuration file or cre_mpf system call to set the size of each memory block and the number of memory blocks.

An error E_NOEXS is returned if this system call is issued for a nonexistent fixed-size memorypool.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32.h>
#include “id.h”
VP      p_blf;
void task()
{
    if( get_blf(&p_blf,ID_mpf) != E_OK ){
        error(“Not enough memory\n”);
    }
}

<< Usage example of the assembly language(CC32R) >>

p_blf: .RES.B 4
    .include “mr32.inc”
    .global task
task:
        :
    get_blf    ID_mpf
    ld24      R5,#p_blf
    st         R2,@R5
        :
    ext_tsk
```

```
<< Usage example of the assembly language(TW32R:DCC/M32R) >>

p_blf: .space 4
    .include “mr32.inc”
    .global task
task:
        :
    get_blf    ID_mpf
    ld24      R5,#p_blf
    st         R2,@R5
        :
    ext_tsk
```

2.9.4. tget_blf(Get Fixed-size Memory Block with Timeout)

[(System call name)]

tget_blf → Gets a fixed-size memory block(With Timeout)

[(Calling by the assembly language)]

```
.include “mr32r.inc”
tget_blf     mpfid,tmout
```

<< Argument >>

mpfid	[**]	The ID No. of the memory pool to be obtained
tmout	[****]	Timeout value

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the memorypool to be obtained
R2	The start address of memory block to be obtained
R3	--
R4	Timeout value

[(Calling by the C language)]

```
#include <mr32r.h>
ER tget_blf (p_blf,mpfid,tmout);
```

<< Argument >>

ID	mplid;	The ID No. of the memorypool to be obtained
VP	*p_blf;	The start address of memory block to be obtained
TMO	tmout;	Timeout value

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_RLWAI       0FFFFFFAAH(-H’00000056): Wait state forcibly
                                       cleared
E_DLT         0FFFFFFAFH(-H’00000051): The object being waited for
                                       was deleted
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

This system call acquires a memory block from the memorypool indicated by mpfid and stores the start address of the acquired memory block in the variable p_blf. The content of the acquired memory block is indeterminate.

If no memory block exists in the specified memorypool, the task that issued this system call goes to a memory block wait state, so it is queued up in two queues: the memory block wait queue and the timeout wait queue.

The wait state committed by issuing this system call is released in the cases described below. Note that when released from the wait state, the task that issued this system call is removed from both of the transmit wait and timeout wait queues and is connected to the ready queue.

- When the release-from-wait condition is met before the tmout time expires

  Error code E_OK is returned.

- When the tmout time expires before the release-from-wait condition is met

  Error code E_TMOUT is returned.

- When the rel_wai or irel_wai system call is issued before the send fixed-size memorypool wait condition is met

  Error code E_RLWAI is returned.

- When the fixed-size memorypool for which a task has been kept waiting is deleted by the del_mpf system call issued by another task

  Error code E_DLT is returned.

- When the fixed-size memorypool for which a task has been kept waiting is reset by the vrst_mpf system call issued by another task

  Error code EV_RST is returned.

Any value from -1 to 7FFFFFFFH can be specified for tmout. If you specify TMO_POL = 0 for tmout, the effect is the same as 0 is specified for the timeout value, in which case tmout functions the same way as pget_blf. Also, if you specify tmout = TMO_FEVR(-1), the effect is the same as endless wait is specified, in which case tmout functions the same way as get_blf.

Memory blocks are fixed in length. Use the configuration file or cre_mpf system call to set the size of each memory block and the number of memory blocks.

An error E_NOEXS is returned if this system call is issued for a nonexistent fixed-size memorypool.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32.h>
#include “id.h”
VP      p_blf;
void task()
{
    if( tget_blf(&p_blf,ID_mpf,50) != E_OK ){
        error(“Not enough memory\n”);
    }
}

<< Usage example of the assembly language(CC32R) >>

p_blf: .RES.B 4
    .include mr32.inc
    .global task
task:
        :
    tget_blf   ID_mpf,50
    ld24      R5,#p_blf
    st        R2,@R5
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

p_blf: .space 4
    .include mr32.inc
    .global task
task:
        :
    tget_blf   ID_mpf,50
    ld24      R5,#p_blf
    st        R2,@R5
        :
    ext_tsk
```

2.9.5. 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 assembly language)]

```
.include “mr32r.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 ID No. of the memorypool to be obtained
R2	The start address of memory block to be obtained
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER pget_blf (p_blf,mplid);
```

<< Argument >>

ID	mplid;	The ID No. of the memorypool to be obtained
VP	*p_blf;	The start address of memory block to be obtained

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_TMOUT       0FFFFFFABH(-H’00000055): Polling failed or timeout
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
```

[(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.

The task is not moved to WAIT state by pget_blf.

An error E_NOEXS is returned if this system call is issued for a nonexistent fixed-size memorypool.

Each memory block is fixed in size. The size of each memory block is defined in the configuration file or when cre_mpf system call is issued.

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

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32.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(CC32R) >>

p_blf: .RES.B 4
    .include mr32.inc
    .global task
task:
        :
    pget_blf   ID_mpf
    ld24      R5,#p_blf
    st        R2,@R5
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

p_blf: .space 4
    .include mr32.inc
    .global task
task:
        :
    pget_blf   ID_mpf
    ld24      R5,#p_blf
    st        R2,@R5
        :
    ext_tsk
```

2.9.6. rel_blf(Release Fixed-size Memory Block)

[(System call name)]

rel_blf → Release a fixed-size memory block

[(Calling by the assembly language)]

```
.include "mr32r.inc"
rel_blf     mpfid
```

<< Argument >>

mpfid [**] The ID No. of the memorypool to be released
You are to set the address of the memory block to be released.Set the address in the R2 reg For the details,see Usage example of the assembly language on the next page.

<< Register setting >>

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

[(Calling by the C language)]

```
#include <mr32r.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 released

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_NOEXS      0FFFFFFCCH(-H'00000034): Object does not exist
```

[(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 get_blf, tget_blf or pget_blf.

Also, if the wait queue of the target memory pool has tasks queued up in it, this system call removes a task from the wait queue that has been placed at the beginning of the wait queue, reconnects it to the ready queue, and assigns it a memory block. In this case, the task status changes from the memory block wait state to an execution (RUN) or executable (READY) state.

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 <mr32r.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(CC32R) >>

p_blf:   .RES.B 4
ID_mpf1:        .equ    1
    .include “mr32r.inc”
    .global _task
_task:
        :
    pget_blf   ID_mpf1
    ld24      R5,#p_blf
    st        R2,@R5
        :
    ld24      R5,#p_blf
    ld        R2,@R5      ; The start address of memory block to be released
        :
    rel_blf    ID_mpf1
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

p_blf:   .space 4
    .equ ID_mpf1,1
    .include “mr32r.inc”
    .global _task
_task:
        :
    pget_blf   ID_mpf1
    ld24      R5,#p_blf
    st        R2,@R5
        :
    ld24      R5,#p_blf
    ld        R2,@R5      ; The start address of memory block to be released
        :
    rel_blf    ID_mpf1
        :
    ext_tsk
```

2.9.7. irel_blf(Release Fixed-size Memory Block)

[(System call name)]

irel_blf → Release a fixed-size memory block.(for the handler only).

[(Calling by the assembly language)]

```
.include “mr32r.inc”
irel_blf     mpfid
```

<< Argument >>

mpfid [**] The ID No. of the memorypool to be released

You are to set the address of the memory block to be released.Set the address in the R2 reg For the details,see Usage example of the assembly language on the next page.

<< Register setting >>

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

[(Calling by the C language)]

```
#include <mr32r.h>
ER irel_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 released

<< Return value >>

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

[(Error codes)]

```
E_OK           00000000H(-H’00000000): Normal End
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
```

[(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 <mr32r.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”);
              :
    irel_blf(ID_mpf1,p_blf);
}
```

```
<< Usage example of the assembly language(CC32R) >>

p_blf:   .RES.B 4
ID_mpf1:        .equ    1
     .include “mr32r.inc”
     .global _task
_task:
         :
     pget_blf   ID_mpf1
     ld24       R5,#p_blf
     st         R2,@R5
         :
     ld24       R5,#p_blf
     ld         R2,@R5      ; The start address of memory block to be released
         :
     irel_blf   ID_mpf1
         :
     ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

p_blf:   .space 4
     .equ ID_mpf1,1
     .include “mr32r.inc”
     .global _task
_task:
         :
     pget_blf   ID_mpf1
     ld24       R5,#p_blf
     st         R2,@R5
         :
     ld24       R5,#p_blf
     ld         R2,@R5      ; The start address of memory block to be released
         :
     irel_blf   ID_mpf1
         :
     ext_tsk
```

2.9.8. ref_mpf(Refer Fixed-size Memorypool Status)

[(System call name)]

ref_mpf → Reference Fixed-size Memorypool Status

[(Calling by the assembly language)]

```
.include "mr32r.inc"
ref_mpf   mpfid
```

<< Argument >>

mpfid	[**]	The ID No. of the memorypool to Reference fixed-size memorypool
pk_rmpf	[****]	Packet address to Reference fixed-size memorypool (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the memorypool to Reference fixed-size memorypool
R2	Packet address to Reference fixed-size memorypool
R3	--

The structure indicated by pk_rmpf returns the following data

Offset	Size		
+0	4	exinf	Extended information
+4	4	wtsk	Waiting task information
+8	4	frbcnt	The number of free blocks
+12	4	blksz	The size of blocks

[(Calling by the C language)]

```
#include <mr32r.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;
```

[(Error codes)]

```
E_OK            00000000H(-H'00000000): Normal End
E_NOEXS         0FFFFFFCCH(-H'00000034): Object does not exist
```

[(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.

- wtsk

  wtsk returns the ID No. of the first task waiting for the specified memorypool. wtsk returns FALSE (0) if there are no tasks waiting to obtain memoy block.

- 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.

An error E_NOEXS is returned if this system call is issued for a nonexistent fixed-size memorypool.

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

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    T_RMPF rmpf;
        :
    ref_mpf(ID_mpf, &rmpf );
        :
    ext_tsk();
}

<< Usage example of the assembly language(CC32R) >>

rmpf:   .RES.B   16
    .include “mr32r.inc”
    .global task
task:
        :
    ld24      R2,#rmpf
    ref_mpf    ID_mpf
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

rmpf:   .space   16
    .include “mr32r.inc”
    .global task
task:
        :
    ld24      R2,#rmpf
    ref_mpf    ID_mpf
        :
```

2.9.9. cre_mpl(Create Variable-size Memorypool)

[(System call name)]

cre_mpl → Create Variable-size Memorypool

[(Calling by the assembly language)]

```
.include “mr32r.inc”
cre_mpl     mplid
```

<< Argument >>

mplid	[**]	The ID No. of a variable-size memorypool to be created
pk_cmpl	[****]	The start address in which the variable-size memorypool generation information is stored (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a variable-size memorypool to be created
R2	The start address in which the variable-size memorypool generation information is stored
R3	--

Specify the following information in the structure indicased by pk_cmpl.

Offset	Size		
+0	4	exinf	Extended information
+4	4	mplatr	Variable-size memorypool attribute
+8	4	mplsz	Variable-size memorypool size
+12	4	maxblksz	Maximum memory block size to be allocated

[(Calling by the C language)]

```
#include <mr32r.h>
ER cre_mpl (mplid, pk_cmpl);
```

<< Argument >>

ID	mplid;	The ID No. of a variable-size memorypool to be created
T_CMPL	*pk_cmpl;	The start address in which the variable-size memorypool generation information is stored

Specify the following information in the structure indicased by pk_cmpl.

```
typedef struct t_cmpl {
    VP      exinf;  /* Extended information */
    ATR     mplatr; /* Variable-size memorypool attribute */
    INT     mplsz;  /* Variable-size memorypool size */
    INT     maxblksz; /* Maximum memory block size to be allocated */
} T_CMPL;
```

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
E_NOMEM         0FFFFFFF6H(-H’0000000a): Not enough of memory
E_OBJ           0FFFFFFC1H(-H’0000003f): Invalid object state
```

[(Function description)]

This system call generates the variable size memorypool of the ID No. specified with mplid.

It secures the memory area used for the memorypool and initializes management block data of the generated memorypool.

The information pk_cmpl of the generated memorypool is as follows.Creates a variable-size memorypool mplid indicates.

Here follows explanation of the information as to a variable-size memorypool to be generated pk_cmpl.

- exinf (extended information)

  Exinf is an area you can freely use to store information as to a variable-size memorypool to be generated. MR32R has nothing to do with the exinf's contents.

- mplatr (variable-size memorypool attribute)

  Specify the location of the variable-size memorypool area to be created. Specifically this means specifying whether you want variable-size memorypool to be located in the internal RAM or in the external RAM.

  - ♦ **To locate the variable-size memorypool area in internal RAM**
    Specify __MR_INT(0).
  - ♦ **To locate the variable-size memorypool area in external RAM**
    Specify __MR_EXT(0x10000).
  - ♦ **To locate the variable-size memorypool area user specified**
    Specify __MR_USER(0x30000).

- mplsz

  Secures the memory area specified with this setting and utilizes it as the memory pool. The E_NOMEM error is returned if the specified memory does not exist.

- maxblksz

  The variable size memory pool of the **MR32R** is divided into 4 memory blocks of fixed sizes. The memory block whose size best matches the size specified by the user is selected from these 4 and assigned as the memory. The sizes are specified by the user with maxblksz. Here, an error is not generated even if the specified size is smaller than mplsz. Checks must be made from the user's side.

The range of ID Nos. which can be specified with this system call is from 1 to the maximum number of variable size memory pools set in the user system by the maximum item number definition.

If this system call is issued for an already existing variable-size memorypool, an error E_OBJ is returned.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task(void)
{
    T_CMPL   setmpl;
        :
    setmpl.mplatr = __MR_INT;
    setmpl.mplsz = 5000;
    setmpl.maxblksz = 400;
    cre_mpl(ID_mpl1,&setmpl);
        :
}

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

setmpl: .RES.B   16
    .include “mr32r.inc”
    .global task
task:
    ld24      R2,#setmpl
    ld24      R1,#__MR_INT
    st        R1,@(4,R2)
    ld24      R1,#5000
    st        R1,@(8,R2)
    ld24      R1,#400
    st        R1,@(12,R2)
    cre_mpl    ID_mpl1
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

setmpl: .space   16
    .include “mr32r.inc”
    .global task
task:
    ld24      R0,#setmpl
    ld24      R1,#__MR_INT
    st        R1,@(4,R2)
    ld24      R1,#5000
    st        R1,@(8,R2)
    ld24      R1,#400
    st        R1,@(12,R2)
    cre_mpl    ID_mpl1
        :
    ext_tsk
```

2.9.10. del_mpl(Delete Variable-size Memorypool)

[(System call name)]

del_mpl → Delete Variable-size Memorypool

[(Calling by the assembly language)]

```
.include "mr32r.inc"
del_mpl     mplid
```

<< Argument >>

mplid	[**]	The ID No. of a variable-size memorypool to be deleted

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a variable-size memorypool to be deleted
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER del_mpl (mplid);
```

<< Argument >>

ID	mplid;	The ID No. of a variable-size memorypool to be deleted

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
```

[(Function description)]

This system call deletes the memorypool indicated with mplid.The deleted memorypool can be generated as a new memorypool with the same ID No. If a task is in the WAIT state, the memorypool wait state is canceled ,and the state changes to READY state.And an error E_DLT is returned for that task.

This system call ends successfully even if there is a task which procures the memory blocks of the memory pool specified with this system call. In such case, no notification as to the task acquiring memory blocks is made.

Also, if this system call is issued for a non--existent memorypool, an error E_NOEXS is returned.

Make sure this system call is issued for only the variable-size memorypool that has been created by the cre_mpl system call. If this system call is issued for the variable-size memorypool that has been defined by the configuration file, it does not function normally.

This system call can be issued only from a task. It does not function properly when issued from the interrupt handler, cyclic handler or alarm handler.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task(void)
{
        :
    del_mpl(ID_mpl1);
        :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global task
task:
        :
    del_mpl ID_mpl1
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global task
task:
        :
    del_mpl ID_mpl1
        :
    ext_tsk
```

2.9.11. get_blk(Get Variable-size Memory Block)

[(System call name)]

get_blk → Gets a variable-size memory block

[(Calling by the assembly language)]

```
.include “mr32r.inc”
get_blk     mplid, blksz
```

<< Argument >>

mplid	[**]	The ID No. of the variable-size memorypool to be obtained
blksz	[****]	Memory block size to be obtained

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the variable-size memorypool to be obtained
R2	The start address of memory block to be obtained
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER get_blk (p_blk,mplid,blksz);
```

<< Argument >>

ID	mplid;	The ID No. of the variable-size memorypool to be obtained
VP	*p_blk;	The start address of memory block to be obtained
INT	blksz;	Memory block size to be obtained

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_RLWAI       0FFFFFFAAH(-H’00000056): Wait state forcibly
                                       cleared
E_DLT         0FFFFFFAFH(-H’00000051): The object being waited for
                                       was deleted
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

This system call gets a variable-size memory block from the memory pool specified by mplid and returns the start address of that memory block to p_blk.The content of the obtained memory block is not fixed.

If the memory block cannot be obtained, the task that has issued this system call is placed in a wait state and linked in a variable-size memory block wait 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.

Also, if the variable-size memorypool whose vliable-size memory block wait queue has a task is deleted by the del_mpl system call of another task, the variable-size memory block wait state of the task in the wait state is canceled and an error E_DLT is returned for that task.

When the variable-size memorypool for which a task has been kept waiting is reset by the vrst_mpl system call issued by another task, error code EV_RST is returned and it is moved to READY state.

If the memorypool does not exist, an error E_NOEXS is returned.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
#define ID_mpl1 1
VP      p_blk;
void task(void)
{
    /* Get 70 bytes memory block */
    if( get_blk(&p_blk,ID_mpl1,70) != E_OK ){
        error(“Not enough memory\n”);
    }
}

<< Usage example of the assembly language(CC32R) >>

p_blk: .RES.B 4
    .include “mr32r.inc”
    .global task
task:
        :
    get_blk     ID_mpl1,50       ; Get 50 bytes memory block
    ld24        R5,#p_blk
    st  R2,@R5          ; send the start address of memory block to be obtaine
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

p_blk: .space 4
    .include “mr32r.inc”
    .global task
task:
        :
    get_blk     ID_mpl1,50       ; Get 50 bytes memory block
    ld24        R5,#p_blk
    st  R2,@R5          ; send the start address of memory block to be obtained
        :
    ext_tsk
```

2.9.12. tget_blk(Get Variable-size Memory Block with Timeout)

[(System call name)]

tget_blk → Gets a variable-size memory block(With Timeout)

[(Calling by the assembly language)]

```
.include "mr32r.inc"
tget_blk     mplid,blksz,tmout
```

<< Argument >>

mplid	[**]	The ID No. of the variable-size memorypool to be obtained
blksz	[****]	Memory block size to be obtained
tmout	[****]	Timeout value

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the variable-size memorypool to be obtained
R2	The start address of memory block to be obtained
R3	--
R4	Timeout value

[(Calling by the C language)]

```
#include <mr32r.h>
ER tget_blk (p_blk,mplid,blksz,tmout);
```

<< Argument >>

ID	mplid;	The ID No. of the variable-size memorypool to be obtained
VP	*p_blk;	The start address of memory block to be obtained
INT	blksz;	Memory block size to be obtained
TMO	tmout;	Timeout value

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_RLWAI       0FFFFFFAAH(-H'00000056): Wait state forcibly
                                       cleared
E_DLT         0FFFFFFAFH(-H'00000051): The object being waited for
                                       was deleted
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
```

[(Function description)]

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

If no memory block exists in the specified memorypool, the task that issued this system call goes to a memory block wait state, so it is queued up in two queues: the memory block wait queue and the timeout wait queue.

The wait state committed by issuing this system call is released in the cases described below. Note that when released from the wait state, the task that issued this system call is removed from both of the transmit wait and timeout wait queues and is connected to the ready queue.

- When the release-from-wait condition is met before the tmout time expires

  Error code E_OK is returned.

- When the tmout time expires before the release-from-wait condition is met

  Error code E_TMOUT is returned.

- When the rel_wai or irel_wai system call is issued before the send variable-size memorypool wait condition is met

  Error code E_RLWAI is returned.

- When the variable-size memorypool for which a task has been kept waiting is deleted by the del_mpl system call issued by another task

  Error code E_DLT is returned.

- When the variable-size memorypool for which a task has been kept waiting is reset by the vrst_mpf system call issued by another task

  Error code EV_RST is returned.

Any value from -1 to 7FFFFFFFH can be specified for tmout. If you specify TMO_POL = 0 for tmout, the effect is the same as 0 is specified for the timeout value, in which case tmout functions the same way as pget_blk. Also, if you specify tmout = TMO_FEVR(-1), the effect is the same as endless wait is specified, in which case tmout functions the same way as get_blk.

An error E_NOEXS is returned if this system call is issued for a nonexistent variable-size memorypool.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32.h>
#include “id.h”
VP      p_blk;
void task()
{
    if( tget_blk(&p_blk,ID_mpl1,50,100) != E_OK ){
        error(“Not enough memory\n”);
    }
        :
}

<< Usage example of the assembly language(CC32R) >>

p_blf: .space 4
    .include “mr32.inc”
    .global task
task:
        :
    tget_blk   ID_mpl,50,100
    ld24      R5,#p_blf
    st        R2,@R5
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

p_blf: .space 4
    .include “mr32.inc”
    .global task
task:
        :
    tget_blk   ID_mpl,50,100
    ld24      R5,#p_blf
    st        R2,@R5
        :
    ext_tsk
```

2.9.13. 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 assembly language)]

```
.include “mr32r.inc”
pget_blk     mplid, blksz
```

<< Argument >>

mplid	[**]	The ID No. of the variable-size memorypool to be obtained
blksz	[****]	Memory block size to be obtained

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the variable-size memorypool to be obtained
R2	The start address of memory block to be obtained
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER pget_blk (p_blk,mplid,blksz);
```

<< Argument >>

ID	mplid;	The ID No. of the variable-size memorypool to be obtained
VP	*p_blk;	The start address of memory block to be obtained
INT	blksz;	Memory block size to be obtained

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_TMOUT       0FFFFFFABH(-H’00000055): Polling failed or timeout
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

This system call gets a variable-size memory block from the memory pool specified by mplid and returns the start address of that memory block to p_blk. The content of the acquired memory block is indeterminate.

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

The task is not moved to WAIT state by pget_blk.

The size of each memory block is defined in the configuration file or or when cre_mpf system call is issued.

An error E_NOEXS is returned if this system call is issued for a nonexistent variable-size memorypool.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.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(CC32R) >>

p_blk: .RES.B 4
    .equ ID_mpl1,1
    .include “mr32r.inc”
    .global task
task:
        :
    pget_blk  ID_mpl1,50      ; Get 50 bytes memory block
    ld24 R5,#p_blk
    st    R2,@R5         ; send the start address of memory block to be obtaine
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

p_blk: .space  4
    .equ ID_mpl1,1
    .include “mr32r.inc”
    .global task
task:
        :
    pget_blk  ID_mpl1,50      ; Get 50 bytes memory block
    ld24 R5,#p_blk
    st    R2,@R5         ; send the start address of memory block to be obtained
        :
    ext_tsk
```

2.9.14. rel_blk(Release Variable-size Memory Block)

[(System call name)]

rel_blk → Release a variable-size memory block

[(Calling by the assembly language)]

```
.include “mr32r.inc”
rel_blk    mplid
```

<< Argument >>

mplid	[**]	The ID No. of the variable-size memorypool to be released
p_blk	[****]	the address of the memory block to be released. Set the address in the R2 register.

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the variable-size memorypool to be released
R2	The start address of memory block to be released
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER rel_blk (mplid,p_blk);
```

<< Argument >>

ID	mplid;	The ID No. of the variable-size memorypool 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.

[(Error codes)]

```
E_OK           00000000H(-H’00000000): Normal End
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

This system call returns the memory block whose start address is specified by p_blk to the memory pool.

For the start address of the memory block to be freed (returned), always use the value obtained by get_blk, tget_blk or pget_blk.

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

If a task is waiting to release memory blocks, the request size is checked from the head task in the memory wait queue. If conditions are satisfied, the memory wait state is changed to the ready state.

In assigning memory, if conditions are satisfied, request size is checked with all subsequently connected tasks. However, the moment a task does not satisfy the request size, memory block assignment ends.

An error E_NOEXS is returned if this system call is issued for a nonexistent variable-size memorypool.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.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(CC32R) >>

p_blk:   .RES.B 4
ID_mpl1:        .equ    1
    .include “mr32r.inc”
    .global _task
_task:
        :
    pget_blk ID_mpl1,60    ; Get 60 bytes memory block
    ld24    R5,#p_blk
    st      R2,@R5
        :
    ; You must set the start address of the memory block to be released
    ld24    R5,#p_blk
    ld      R2,@R5        ; start address of the memory block
    rel_blk               ; Release memory block

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

p_blk:   .space 4
    .equ ID_mpl1,1
    .include “mr32r.inc”
    .global _task
_task:
        :
    pget_blk ID_mpl1,60    ; Get 60 bytes memory block
    ld24    R5,#p_blk
    st      R2,@R5
        :
    ; You must set the start address of the memory block to be released
    ld24    R5,#p_blk
    ld      R2,@R5        ; start address of the memory block
    rel_blk               ; Release memory block
```

2.9.15. ref_mpl(Refer Variable-size Memorypool Status)

[(System call name)]

ref_mpl → Reference Variable-size Memorypool Status

[(Calling by the assembly language)]

```
.include “mr32r.inc”
ref_mpl   mplid,pk_rmpl
```

<< Argument >>

mplid	[**]	The ID No. of the variable-size memorypool to be referenced
pk_rmpl	[****]	Packet address to Reference variable-size memorypool (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the memory block to Reference variable-size memorypool
R2	Packet address to Reference variable-size memorypool
R3	--

The structure indicated by pk_rmpl returns the following data

Offset	Size		
+0	4	exinf	Extended information
+4	4	wtsk	Waiting task information
+6	4	frsz	The total size of the free area
+10	4	maxsz	The size of the maximum free area

[(Calling by the C language)]

```
#include <mr32r.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

<< 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 */
    INT     frsz;   /* The total size of the free area */
    INT     maxsz;  /* The size of the maximum free area */
} T_RMPL;
```

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
E_NOEXS         0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

Refers to the state of the variable-size memory pool specified by mplid, and returns the following information as return values.

- exinf

  Returns extended task information in exinf

- wtsk

  Returns the ID No. of the first task waiting for the specified variable-size memory pool. In the MR32R, however, wtsk always returns FALSE (0), because tasks cannot enter the wait state for the memory pool.

- frsz

  Returns the total size of the free area.

- maxsz

  Returns the size of the maximum free area that can immediately be obtained.

An error E_NOEXS is returned if this system call is issued for a nonexistent variable-size memorypool.

This system call can be issued from both tasks and handlers (the interrupt handler, the cyclic handler, or the alarm handler).

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    T_RMPL rmpl;
    ref_mpl(&rmpl,ID_mpl1);
}

<< Usage example of the assembly language(CC32R) >>

rmpl:   .space   10
    .equ ID”mpl1,1
    .include mr32r.inc
    .global task
task:
        :
    ld24      R2,#rmpl
    ref_mpl    ID_mpl1

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

rmpl:   .space   10
    .equ ID”mpl1,1
    .include mr32r.inc
    .global task
task:
        :
    ld24      R2,#rmpl
    ref_mpl    ID_mpl1
```

2.10. Time Management Function

2.10.1. set_tim(Set Time)

[(System call name)]

set_tim → Sets the system clock.

[(Calling by the assembly language)]

```
.include "mr32r.inc"
set_tim
```

<< Argument >>

pk_tim	[****]	The start address of packet specifying the system clock to be set (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	The start address of packet specifying the system clock to be set
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK            00000000H(-H'00000000): Normal End
```

[(Function description)]

This system call sets the value of the system clock to a value indicated by pk_tim. [27]

The 48-bit system clock is handled separately in ltime(32-bits) and utime(16-bits).

The timer interrupt interval used in MR32R kernel is treated as a unit of system clock.

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

[27] 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 <mr32r.h>
#include “id.h”
void task()
{
    SYSTITIME time;       /* Time data storing variable */
    time.utime = 0;       /* Sets upper time data */
    time.ltime = 0;       /* Sets lower time data */
    set_tim( &time );     /* modify the system time */
}
<< Usage example of the assembly language(CC32R) >>

time:   .RES.B    6
    .INCLUDE “mr32r.inc”
    .GLOBAL   task
task:
    set_tim   time
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

time:   .space    6
    .INCLUDE “mr32r.inc”
    .GLOBAL   task
task:
    set_tim   time
        :
```

2.10.2. get_tim(Get Time)

[(System call name)]

get_tim → Reads the system clock value.

[(Calling by the assembly language)]

```
.include "mr32r.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	The start address of packet in which the read system clock is stored
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER get_tim (pk_tim);
```

<< Argument >>

SYSTIME	*pk_tim;	The start address of packet in which the read system clock is stored (Set the address in the R2 register.)

<< 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.

[(Error codes)]

E_OK	00000000H(-H'00000000): Normal End

[(Function description)]

This system call reads out the current value of the system clock and returns it to return parameter pk_tim. [28]

The 48-bit system clock time is handled separately in ltime(32-bits) and utime(16-bits).

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

[28] 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 <mr32r.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.ltime = %X\n”,time.ltime);
}
<< Usage example of the assembly language(CC32R) >>

time: .RES.B    6
    .include “mr32r.inc”
    .global   task
task:
    ld24       R2,#time
    get_tim
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

time: .space    6
    .include “mr32r.inc”
    .global   task
task:
    ld24       R2,#time
    get_tim
        :
```

2.10.3. dly_tsk(Delay Task)

[(System call name)]

dly_tsk → Delays task execution.

[(Calling by the assembly language)]

```
.include “mr32r.inc”
dly_tsk     dlytim
```

<< Argument >>

dlytim [****] Delay time

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	Delay time
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER dly_tsk (dlytim);
```

<< Argument >>

DLYTIME dlytim; Delay time

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
E_RLWAI        0FFFFFFAAH(-H’00000056): 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 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.[29]

The maximum value of dlytim is 0x7FFFFFFF.

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 <mr32r.h>
#include “id.h”
void task()
{
    if( dly_tsk( 10 ) != E_OK )
        printf(“Forced wakeup\n”);
              :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task
task:
    dly_tsk 200
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task
task:
    dly_tsk 200
        :
```

[29] Refer Users Manual how to specify the unit of time of the system clock in the configuration file.

2.10.4. def_cyc(Define Cyclic Handler)

[(System call name)]

def_cyc → Define Cyclic Handler

[(Calling by the assembly language)]

```
.include “mr32r.inc”
def_cyc     cycno,pk_dcyc
```

<< Argument >>

cycno	[****]	ID No. of cyclic handler
pk_dcyc	[****]	The start address in which the cyclic handler definition information is stored (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	ID No. of cyclic handler
R2	The start address in which the cyclic handler definition information is stored
R3	--

Specify the following information in the structure indicased by pk_dexc.

Offset	Size		
+0	4	exinf	Extended information
+4	4	cycatr	Cyclic handler attribute
+8	4	cychdr	Cyclic handler startup address
+12	4	cycact	The cyclic handler activation status
+16	4	cyctim	Interval count

[(Calling by the C language)]

```
#include <mr32r.h>
ER def_cyc(cycno, pk_dcyc);
```

<< Argument >>

HNO	cycno	ID No. of cyclic handler
T_DCYC	*pk_dcyc	The start address in which the cyclic handler definition information is stored

Specify the following information in the structure indicased by pk_dcyc.

```
typedef struct t_dcyc {
    VP      exinf;          /* Extended information */
    ATR     cycatr;         /* Cyclic handler attribute */
    FP      cychdr;         /* Cyclic handler startup address */
    UINT    cycact;         /* The cyclic handler activation status */
    CYCTIME cyctim;         /* Interval count */
} T_DCYC;
```

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H'00000000): Normal End
```

[(Function description)]

- exinf

  Exinf is an area you can freely use to store information as to a cyclic handler to be generated. MR32R has nothing to do with the exinf's contents.

- cycatr

  cycatr is an area you can freely use to store information as to a cyclic handler to be generated. MR32R has nothing to do with the cycatr's contents.

- cychdr

  Specifies the start address of the defined cyclic handler.

- cycact

  The following two specifications can be made by cycact:

Table 2.1Specifications of Cyclic Handler Activation Status

C language	Meaning
TCY_OFF	Disables the cyclic handler
TCY_ON	Enables the cyclic handler

- cyctim

  Specifies the interval count for cyclic handler.

It is also possible to re-define a cyclic handler to the cyclic handler already defined. In a re-definition, it is not necessary to cancel a definition beforehand. It does not become an error even if it re-defines a new a cyclic handler to the cyclic handler already defined.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    T_DCYC dcyc1;
        :
    dcyc1.cychdr = cyc1;
    dcyc1.cycact = TCY_ON;
    dcyc1.cyctim = 200;
    def_cyc ( ID_cyc, &dcyc );
        :
}
void cyc1(void)
{
    :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global   task
dcyc1:  .RES.B 20
task:
    ld24    R2,#dcyc1
    ld24    R1,#cyc1
    st      R1,@(8,R2)
    ldi     R1,#TCY_ON
    st      R1,@(12,R2)
    ldi     R1,#200
    st      R1,@(16,R2)
    def_cyc  ID_CYC
        :
    ext_tsk
cyc1:
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>
    .include “mr32r.inc”
    .global   task
dcyc1:
    .space  20
task:
    ld24    R2,#dcyc1
    ld24    R1,#cyc1
    st      R1,@(8,R2)
    ldi     R1,#TCY_ON
    st      R1,@(12,R2)
    ldi     R1,#200
    st      R1,@(16,R2)
    def_cyc  ID_CYC
        :
    ext_tsk
cyc1:
        :
```

2.10.5. act_cyc (Activate Cyclic Handler)

[(System call name)]

act_cyc → Controls the activation of the cyclic handler.

[(Calling by the assembly language)]

```
.include “mr32r.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	The cyclic handler specification number
R2	The cyclic handler activation status
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

```
E_OK            00000000H(-H’00000000): 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:

Table 2.2Specifications 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.[30]

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

[30] Namely, the cyclic handler is called from the system clock handler by a subroutine call.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
        :
    act_cyc ( ID_cyc, TCY_ON );
        :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global   task
task:
    act_cyc   ID_cyc, TCY_INI_ON
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global   task
task:
    act_cyc   ID_cyc, TCY_INI_ON
        :
```

2.10.6. ref_cyc(Refer Cyclic Handler Status)

[(System call name)]

ref_cyc → Reference Cyclic handler Status

[(Calling by the assembly language)]

```
.include “mr32r.inc”
ref_cyc   cycno
```

<< Argument >>

cycno	[**]	The cyclic handler specification number
pk_rcyc	[****]	Packet address to Reference cyclic handler (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The cyclic handler specification number
R2	Packet address to Reference cyclic handler
R3	--

The structure indicated by pk_rcyc returns the following data.

Offset	Size		
+0	4	exinf	Extended information
+4	4	lftim	The time remaining until the next cycle start handler starts
+8	4	cycact	The active state of the cycle start handler

[(Calling by the C language)]

```
#include <mr32r.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;
```

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
```

[(Function description)]

Refers to the state of the cyclic handler specified by almno, and returns the following information as return values.

- exinf

  Returns extended task information in exinf.

- cycact

  cycact returns the active state of the cyclic handler. That is, cycact returns TCY_ON (=1) when the cyclic handler is ON, and TCY_OFF (=0) when it is OFF.

- lfttime

  lftim returns the time remaining until the next cyclic handler starts. The time remaining until the next cyclic handler starts is expressed as the number of system clock counts.

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

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    T_RCYC rcyc;
    ref_cyc( &rcyc, ID_cyc );
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global task
task:
        :
    ld24     R2,#pk_rcyc
    ref_cyc   ID_cyc
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global task
task:
        :
    ld24     R2,#pk_rcyc
    ref_cyc   ID_cyc
        :
```

2.10.7. ref_alm(Refer Alarm Handler Status)

[(System call name)]

ref_alm → Reference Alarm handler Status

[(Calling by the assembly language)]

```
.include "mr32r.inc"
ref_alm   almno
```

<< Argument >>

almno	[**]	The alarm handler specification number
pk_ralm	[****]	Packet address to Reference alarm handler (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The alarm handler specification number
R2	Packet address to Reference alarm handler
R3	--

The structure indicated by pk_ralm returns the following data.

Offset	Size		
+0	4	exinf	Extended information
+4	4	lftim	The time remaining until the next alarm start handler starts

[(Calling by the C language)]

```
#include <mr32r.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;
```

[(Error codes)]

```
E_OK            00000000H(-H'00000000): 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.

- 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 and utime.

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

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void func()
{
    T_RALM   ralm;
    ref_alm( &ralm, ID_alarm );
                :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global task
task:
    ld24      R2,#pk_ralm
    ref_alm    ID_alm
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global     task
task:
    ld24      R2,#pk_ralm
    ref_alm    ID_alm
        :
```

2.11. System Management Function

2.11.1. get_ver(Get Version Information)

[(System call name)]

get_ver → Gets the version number of the MR32R.

[(Calling by the assembly language)]

```
.include “mr32r.inc”
get_ver
```

<< Argument >>

pk_ver	[****]	The start address of the structure in which version information is stored (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	The start address of the structure in which version information is stored
R3	--

[(Calling by the C language)]

```
#include <mr32r.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.

[(Error codes)]

E_OK	00000000H(-H’00000000): Normal End

[(Function description)]

This system call gets the version number and other information on the MR32R.

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'0C indicating Mitsubishi Electric Corporation is returned.
2. Format number
   Internal identification ID H'221of the MR32R is returned.
3. Specification version
   H'5302 indicating the μITRONspecifications Ver.3.02 is returned.
4. Product version
   H'320 indicating the version of the MR32R is returned.
5. Product control information
   - prno[0]

     The product release number is obtained

     prno[0] ← '01'

   - prno[1]

     A two digit of the product release year and month are obtained

     prno[1] ← 0x0007

   - prno[2]

     Reserved for Mitsubishi use.

     prno[2] ← 0x????

   - prno[3]

     Reserved for Mitsubishi use.

     prno[3] ← 0x????
6. CPU information
   H'C31 indicating the M32R Micro computeris returned.
7. Variation descriptor
   H'8000 indicating the variation of the MR32R is returned.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    T_VER   pk_ver;
    get_ver( &pk_ver );
}

<< Usage example of the assembly language(CC32R) >>

ver:   .RES.B     10
    .include “mr32r.inc”
    .global   task
task:
    ld24        R2,#ver
    get_ver
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

ver:   .space     10
    .include “mr32r.inc”
    .global   task
task:
    ld24        R2,#ver
    get_ver    ver
        :
```

2.11.2. ref_sys(Refer System Status)

[(System call name)]

ref_sys → Reference Status of CPU and OS.

[(Calling by the assembly language)]

```
.include “mr32r.inc”
ref_sys
```

<< Argument >>

pk_rsys	[****]	The start address of the structure in which system status information is stored (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	The start address of the structure in which system status information is stored
R3	--

The structure indicated by pk_sys returns the following data.

Offset	Size		
+0	4	exinf	Extended information
+4	2	runtskid	The ID No. of RUN state task
+6	2	runtskpri	The priority of RUN state task
+8	4	psw	PSW

[(Calling by the C language)]

```
#include <mr32r.h>
ER ref_sys (pk_rsys);
```

<< Argument >>

T_RSYS	*pk_rsys;	Packet address to Reference system status

<< Return value >>

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

```
typdef strcut t_rsys {
    INT     sysstat;        /* System status */
    ID      runtskid;       /* The ID No. of RUN state task */
    PRI     runtskpri;      /* The priority of RUN state task */
    UINT    psw;            /* PSW */
} T_RSYS;
```

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
```

[(Function description)]

This system call check execution state of the CPU and OS, and returns results to the pk_rsys area.

- sysstat

  Indicates system status. The following values are returned.

  sysstat:=(TSS_TSK||TSS_DDSP||TSS_LOC||TSS_INDP)

sysstat	Value	Status	Dispatch	Interrupt
TSS_TSK	0	task	enable	enable
TSS_DDSP	1	task	disable	enable
TSS_LOC	2	task	disable	disable
TSS_INDP	4	task independent	disable	disable

- runtskid

  Returns the ID No. of the task currently being run.

- runtskpri

  Returns the priority level of the task currently being run.

- psw

  Returns the value of the processor status word of the running task or task--independent portions.

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

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    T_RSYS rsys;
    ref_sys( &rsys );
}

<< Usage example of the assembly language(CC32R) >>

pk_rsys: .RES.B 12
    .include mr32r.inc
    .global task
task:
        :
    ld24     R2,#pk_rsys
    ref_sys
        :
```

```
<< Usage example of the assembly language(TW32R:DCC/M32R) >>

pk_rsys: .space 12
    .include mr32r.inc
    .global task
task:
        :
    ld24     R2,#pk_rsys
    ref_sys
        :
```

2.11.3. def_exc(Define Exception Handler)

[(System call name)]

def_exc → Define Exception Handler

[(Calling by the assembly language)]

```
.include "mr32r.inc"
def_exc     exckind
```

<< Argument >>

exckind	[****]	Kind of exception handler
pk_dexc	[****]	The start address in which the exception handler generation information is stored (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	Kind of exception handler
R2	The start address in which the exception handler generation information is stored
R3	--

Specify the following information in the structure indicased by pk_dexc.

Offset	Size		
+0	4	excatr	Exception handler attribute
+4	4	exchdr	Exception handler startup address
+8	2	tskid	The ID No. of task
+12	4	excstksz	Stack size

[(Calling by the C language)]

```
#include <mr32r.h>
ER def_exc(exckind, pk_dexc);
```

<< Argument >>

INT	execkind;	Kind of exception handler
T_DEXC	*pk_dexec;	The start address in which the exception handler generation information is stored

Specify the following information in the structure indicased by pk_dexc.

```
typedef struct t_dexc {
    ATR     excatr;         /* Exception handler attribute */
    FP      exchdr;         /* Exception handler startup address */
    ID      taskid;         /* The ID No. of task */
    W       excstksz;       /* Stack size */
} T_DEXC;
```

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H'00000000): Normal End
E_NOMEM         0FFFFFFF6H(-H'0000000a): Not enough of memory
E_NOEXS         0FFFFFFCCH(-H'00000034): Object does not exist
E_OBJ           0FFFFFFC1H(-H'0000003f): Invalid object state
```

[(Function description)]

This system call defines the exception handler corresponding to exckind exception.

exckind defines the kind of exception handler. With the MR32R, only the forced exception (EXK_FEX = 2) can be specified. However, an error is not returned when other exception handlers (CPU exception or forced end) are specified.

tskid = TSK_SELF (=0) specifies the self task. An error E_OBJ is returned if this system call is issued for a task in the dormant state.

The information pk_dexc of the generated exception handler is as follows.

- excatr

  Specify the location of the excption handler stack area to be created. Specifically this means specifying whether you want the stack to be located in the internal RAM or in external RAM.

  - **To locate the stack area in internal RAM**
    Specify __MR_INT(0).
  - **To locate the stack area in external RAM**
    Specify __MR_EXT(0x10000).
  - **To locate the stack area in user specified**
    Specify __MR_USER(0x30000).

- exchdr

  Specifies the start address of the defined exception handler.

  pk_decx.exchdr = NADR (= --1) cancels the defined exception handler. When canceled, the exception handler changes to the predefined default. Also, an exception handler can be redefined before it is canceled.

- tskid

  Defines an exception handler for the task specified here. tskid=TSK_SELF=0 means specifying own task.tskid=TSK_SELF can’t be specified when this system call is issued from the forced exception handler.

- excstksz

  Specifies the stack size of the defined exception handler. Memory for the exception handler stack is secured by the OS when the exception handler starts up. When exchdr=NADR is specified,the memory for it’s stack is released.If the memoy size for stack is not enough,an error E_NOMEM is returned.

The stack of forced exception handler is obtained from the stack area for task creating.So, int_memstk or ext_memstk must be specified in configuration file when this system call is issued.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void excr(void);
void task1()
{
    ER ercd;
    T_DEXC pk_dexc;
    void fexhdr(T_EXC *pk_exc, T_REGS *pk_regs, T_EIT *pk_eit);

    pk_dexc.exchdr    = (FP)fexhdr;
    pk_dexc.tskid     = TSK_SELF;
    pk_dexc.excstksz   = 100;
    ercd = def_exc( EXK_FEX, &pk_dexc );
        :
}
void fexhdr(T_EXC *pk_exc, T_REGS *pk_regs, T_EIT *pk_eit)
{
        :
    /* Exception handler processing */
        :
}

<< Usage example of the assembly language(CC32R) >>

pk_exc: .RES.B 14
    .include “mr32r.inc”
    .global  task1
task1:
        :
    ld24    R2,#pk_exc
    ld24    R1,#_fexhdr
    st      R1,@(4,R2)
    ld24    R1,#ID_tskid
    sth     R1,@(8,R2)
    ld24    R1,#100
    st      R1,@(12,R2)
    def_exc EXK_FEX
                :
    ext_tsk
_fexhdr:
        :
    ; Exception handler processing
        :
    ext_tsk
```

```
<< Usage example of the assembly language(TW32R:DCC/M32R) >>

pk_exc: .space 14
    .include “mr32r.inc”
    .global  task1
task1:
        :
    ld24    R2,#pk_exc
    ld24    R1,#_fexhdr
    st      R1,@(4,R2)
    ld24    R1,#ID_tskid
    sth     R1,@(8,R2)
    ld24    R1,#100
    st      R1,@(12,R2)
    def_exc EXK_FEX
                :
    ext_tsk
_fexhdr:
        :
    ; Exception handler processing
        :
    ext_tsk
```

2.12. Implementation-Dependent System Call

2.12.1. vclr_ems(Clear Exception Mask)

[(System call name)]

vclr_ems → Clear Exception Mask.

[(Calling by the assembly language)]

```
.include "mr32r.inc"
vclr_ems     tskid
```

<< Argument >>

tskid	[**]	The ID No. of a task to be cleared exception mask.

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a task to be cleared exception mask.
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER vclr_ems ( tskid );
```

<< Argument >>

ID	tskid;	The ID No. of a task to be cleared exception mask.

<< Return value >>

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

[(Error codes)]

```
E_OK           00000000H(-H'00000000): Normal End
E_NOEXS        0FFFFFFCCH(-H'00000034): Object does not exist
E_OBJ          0FFFFFFC1H(-H'0000003f): Invalid object state
```

[(Function description)]

This system call clears the exception mask of the task specified with tskid.

When this system call is issued, the exception mask of tasks for which a forced exception is pending is cleared and the respective exception handler is started up.

While an exception mask is set, an exception handler can be started up only 1 time even if the forced exception start request is sent multiple times.

The self task can be specified. tskid = TSK_SELF (=0) specifies the self task.

If the task is in DORMANT state,an error E_OBJ is returned for the system call. Also, if the task described with tskid is the NON--EXISTENT state,an error E_NOEXS is returned.

This system can be issued from only tasks.This system call, if issued either from the interrupt handler, the cyclic, or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task1(void)
{
        :
    vclr_ems( ID_task2 );
        :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task1
task1:
        :
    vclr_ems ID_task2
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task1
task1:
        :
    vclr_ems ID_task2
        :
    ext_tsk
```

2.12.2. vset_ems(Set Exception Mask)

[(System call name)]

vset_ems → Set Exception Mask.

[(Calling by the assembly language)]

```
.include “mr32r.inc”
vset_ems     tskid
```

<< Argument >>

tskid [**] The ID No. of a task to be set exception mask.

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a task to be set exception mask.
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER vset_ems ( tskid );
```

<< Argument >>

ID tskid; The ID No. of a task to be set exception mask.

<< Return value >>

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

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
E_NOEXS        0FFFFFFCCH(-H’00000034): Object does not exist
E_OBJ          0FFFFFFC1H(-H’00
00003f): Invalid object state
```

[(Function description)]

This system call sets the exception mask of the task specified with tskid.

This system call puts in the pending state the forced exception of a task with an exception mask and delays the start of the exception handler until the exception mask is cleared.

While an exception mask is set, an exception handler can be started up only 1 time even if the forced exception start request is sent multiple times.

If you specify tskid = TSK_SELF (=0), it specifies the task itself.If the task is in the DORMANT state, an error E_OBJ is returned.If the task does not exist, an error E_NOEXS is returned

This system can be issued from only tasks.

This system call, if issued either from the interrupt handler, the cyclic, or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task1(void)
{
        :
    vset_ems( ID_task2 );
        :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task1
task1:
        :
    vset_ems ID_task2
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task1
task1:
        :
    vset_ems ID_task2
        :
    ext_tsk
```

2.12.3. vras_fex(Raise Forcibly Exception)

[(System call name)]

vras_fex → Raise forcibly exception.

[(Calling by the assembly language)]

```
.include “mr32r.inc”
vras_fex     tskid,exccd
```

<< Argument >>

tskid	[**]	The ID No. of a task
exccd	[****]	Forcibly exception code

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a task
R2	Forcibly exception code
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER vras_fex (tskid,exccd);
```

<< Argument >>

ID	tskid;	The ID No. of a task
UW	exccd;	Forcibly exception code

<< Return value >>

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

[(Error codes)]

```
E_OK           00000000H(-H’00000000): Normal End
E_NOEXS        0FFFFFFCCH(-H’00000034): Object does not exist
E_OBJ          0FFFFFFC1H(-H’0000003f): Invalid object state
```

[(Function description)]

This system call starts the forcible exception of the task specified with tskid.If the task specified with tskid does not exist,the E_NOEXS error is returned.The self task cannot be specified.The E_OBJ error is returned if it is.Also, TSK_SELF cannot be specified.

Queuing is not possible even if this system call is issued multiple times.

An exception handler can be started up only 1 time even if the forced exception start request is sent more than 2 times up until the interrupt handler starts up.

The exception code 'exccd' is transferred to the exception handler as the pk_exc exception parameter. If multiple forcible exception start requests are sent, the exccd logical OR is taken.

The forcible exception does not cancel the task wait or suspend state.The forced exception hander startup is delayed until the task changes to the RUN state,even if this system call is issued.

This system can be issued from only tasks.This system call, if issued either from the interrupt handler, the cyclic, or the alarm handler, doesn't work properly

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task1()
{
        :
    vras_fex(ID_task2,0x3)
        :
}

<< Usage example of the assembly language(CC32R) >>

setpor: .space 16
    .include “mr32r.inc”
    .global  task1
task1:
        :
    vras_fex ID_task2,0x3
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

setpor: .space 16
    .include “mr32r.inc”
    .global  task1
task1:
        :
    vras_fex ID_task2,0x3
        :
    ext_tsk
```

2.12.4. vret_exc(Return Exception Handler)

[(System call name)]

vret_exc → Return Exception Handler

[(Calling by the assembly language)]

```
.include “mr32r.inc”
vret_exc
```

<< Argument >>

None

<< Register setting >>

Control is not returned to the exception handler which issued this system call.

[(Calling by the C language)]

```
#include <mr32r.h>
ER vret_exc();
```

<< Argument >>

None

<< Return value >>

Control is not returned to the exception handler which issued this system call.

[(Error codes)]

None

[(Function description)]

This system call returns control from a forced exception handler to the task in which the exception occurred. At this time, control returns to the task context in which state the exception had occurred.

To restart the exception handler, issue the vras_fex system call. It restarts the exception handler.

This system call can be issued only from exception handler.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task1()
{
        :
        :
}
void exc_hdr(void)
{
        :
        :
    vret_exc();
}
```

```
<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  exc_hdr
exc_hdr:
        :
        :
    ret_exc

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  exc_hdr
exc_hdr:
        :
        :
    ret_exc
```

2.12.5. vrst_msg(Reset Message)

[(System call name)]

```
vrst_msg  →  Clear all messages in the specified mailbox.
```

[(Calling by the assembly language)]

```
.include “mr32r.inc”
vrst_msg     mbxid
```

<< Argument >>

mbxid	[**]	The ID No. to be cleared

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a mailbox
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER vrst_msg ( mbxid );
```

<< Argument >>

ID	mbxid;	The ID No. of a mailbox

<< Register setting >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000) :  Normal End
E_NOEXS        0FFFFFFCCH(-H’00000034):  Object does not exist
```

[(Function description)]

Clear all messages in the specified mailbox.If there is no message in the mailbox, this system call does nothing.

An error E_NOEXS is returned if this system call is issued for a nonexistent mailbox.

This system call can be issued from both tasks and handlers (the interrupt handler, the cyclic handler, or the alarm handler).

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task1(void)
{
        :
    vrst_msg( ID_mbx1 );
        :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task1
task1:
        :
    vrst_msg ID_mbx1
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task1
task1:
        :
    vrst_msg ID_mbx1
        :
    ext_tsk
```

2.12.6. vrst_blf (Reset Fixed-Memory Block)

[(System call name)]

```
vrst_blf   →   All memory blocks specified as blfid are released.
```

[(Calling by the assembly language)]

```
.include “mr32r.inc”
vrst_blf     blfid
```

<< Argument >>

blfid [**] The ID No. of the memory pool to be released

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the memory pool to be released
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER vrst_blf (blfid);
```

<< Argument >>

ID blfid; The ID No. of the memory pool to be released

<< Register setting >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000) :  Normal End
E_NOEXS        0FFFFFFCCH(-H’00000034):  Object does not exist
```

[(Function description)]

All memory blocks specified as blfid are released.

An error E_NOEXS is returned if this system call is issued for a nonexistent fixed-size memorypool.

Even when there is any task waiting for a memory block in the memorypool to be reset, this system call is terminated normally. In this case, the said task is freed from the memory block wait state and returns error EV_RST before entering an execution (RUN) or executable READY) state.

Notice ,the memorypool released by vrst_blf is not allocated for the wait tasks.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task1()
{
        :
    vrst_blf(ID_mpf1)
        :
}

<< Usage example of the assembly language(CC32R) >>

setpor: .RES.B 16
    .include “mr32r.inc”
    .global  task1
task1:
        :
    vrst_blf ID_mpf1
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

setpor: .space 16
    .include “mr32r.inc”
    .global  task1
task1:
        :
    vrst_blf ID_mpf1
        :
    ext_tsk
```

2.12.7. vrst_blk(Reset Variable-Memory Block)

[(System call name)]

```
vrst_blk  →  All memory blocks specified as blkid are released.
```

[(Calling by the assembly language)]

```
.include “mr32r.inc”
vrst_blk     blkid
```

<< Argument >>

blkid [**] The ID No. of the memory pool to be released

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the memory pool to be released
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER vrst_blk ( blkid );
```

<< Argument >>

ID blkid; The ID No. of the memory pool to be released

<< Register setting >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000) :  Normal End
E_NOEXS        0FFFFFFCCH(-H’00000034):  Object does not exist
```

[(Function description)]

All variable-size memory blocks specified as blfid are released.

An error E_NOEXS is returned if this system call is issued for a nonexistent variable-size memorypool.

Even when there is any task waiting for a memory block in the memorypool to be reset, this system call is terminated normally. In this case, the said task is freed from the memory block wait state and returns error EV_RST before entering an execution (RUN) or executable READY) state.

Notice ,the memorypool released by vrst_blk is not allocated for the wait tasks.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task1()
{
        :
    vrst_blk(ID_mpl1)
        :
}

<< Usage example of the assembly language(CC32R) >>

setpor: .RES.B 16
    .include “mr32r.inc”
    .global  task1
task1:
        :
    vrst_blk ID_mpl1
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

setpor: .space 16
    .include “mr32r.inc”
    .global  task1
task1:
        :
    vrst_blk ID_mpl1
        :
    ext_tsk
```

2.12.8. vrst_mbf (Reset Message Buffer)

[(System call name)]

```
vrst_mbf   →   All message buffer specified as mbfid are cleared.
```

[(Calling by the assembly language)]

```
.include “mr32r.inc”
vrst_mbf     mbfid
```

<< Argument >>

mbfid [**] The ID No. of the message buffer to be cleared

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the message buffer to be cleared
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER vrst_mbf (mbfid);
```

<< Argument >>

ID mbfid; The ID No. of the message buffer to be cleared

<< Register setting >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000) :  Normal End
E_NOEXS        0FFFFFFCCH(-H’00000034):  Object does not exist
```

[(Function description)]

The message buffer specified as blfid are cleared.

An error E_NOEXS is returned if this system call is issued for a nonexistent message buffer.

Even when there is any task waiting for a message in the message buffer to be reset, this system call is terminated normally. In this case, the said task is freed from the send message wait state or the receive message wait and returns error EV_RST before entering an execution (RUN) or executable READY) state.

Notice ,the send message wait task is moved to READY state without sending message by vrst_mbf.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task1()
{
        :
    vrst_mbf(ID_mbf1)
        :
}

<< Usage example of the assembly language(CC32R) >>

setpor: .space 16
    .include “mr32r.inc”
    .global  task1
task1:
        :
    vrst_mbf ID_mbf1
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

setpor: .space 16
    .include “mr32r.inc”
    .global  task1
task1:
        :
    vrst_mbf ID_mbf1
        :
    ext_tsk
```

2.13. Implementation-Dependent System Call(Mailbox)

2.13.1. vcre_mbx(Create Mailbox)

[(System call name)]

vcre_mbx → Create Mailbox with priority value

[(Calling by the assembly language)]

```
.include "mr32r.inc"
vcre_mbx     vmbxid
```

<< Argument >>

vmbxid	[**]	The ID No. of a mailbox to be created
pk_cvmbx	[****]	The start address in which the mailbox generation information is stored (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a mailbox to be created
R2	The start address in which the mailbox generation information is stored
R3	--

Specify the following information in the structure indicased by pk_cmbx.

Offset	Size		
+0	4	mbxatr	Mailbox attribute
+4	4	maxpri	Max priority value of the message
+8	4	mprihd	The start address of the message queue header area

[(Calling by the C language)]

```
#include <mr32r.h>
ER vcre_mbx (vmbxid, pk_cvmbx);
```

<< Argument >>

ID	vmbxid;	The ID No. of a mailbox to be created
T_CVMBX	*pk_cvmbx;	The start address in which the mailbox generation information is stored

Specify the following information in the structure indicased by pk_cmbx.

```
typedef struct t_cmbx {
    ATR     mbxatr; /* Mailbox attribute */
    PRI     maxpri; /* Max priority value of the message */
    VP      mprihd; /* The start address of the message queue header area
*/
 } T_CVMBX;
```

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_OBJ         0FFFFFFC1H(-H'0000003f): Invalid object state
```

[(Function description)]

Creates a mailbox with priority value mbxid indicates.

Here follows explanation of the information as to a mailbox to be generated pk_cvmbx.

- **mbxatr** (mailbox attribute)

  Specify the mailbox attribute as below.

♦ **How to wait a message**

- **TA_TFIFO(=0x00)** **connect the task as FIFO order**
- **TA_TPRI(=0x01)** **connect the task as priority order**

♦ **How to send a message**

- **TA_MFIFO(=0x00)** **connect the message as FIFO order**
- **TA_MPRI(=0x01)** **connect the message as priority order**

- **maxpri**

  Specify the max priority value of the message.

- **mprihd**

  Specify NULL(=0) in this item.

An error E_OBJ is returned if vcre_mbx system call is issued for the mailbox which is existent.

The range of the specifiable ID number is 1 to the maximum value specified in the configuration file.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
#define ID_vmbx1 1
void task1()
{
    T_CVMBX setmbx;
                :
    setmbx.mbxatr = 0x02;
    setmbx.maxpri = 10;
    setmbx.mprihd = NULL;
    vcre_mbx( ID_mbx1, &setmbx );
                :
    ext_tsk();
}

<< Usage example of the assembly language(CC32R) >>

    .equ ID_mbx1,1
    setmbx: .RES.B 12
    .include “mr32r.inc”
    .global  task1
task1:
        :
    ld24   R2,#setmbx
    ld24   R1,#H’02
    st     R1,@(4,R2)
    ld24   R1,#10
    st     R1,@(8,R2)
    ldi    R1,#0
    st     R1,@(12,R2)
    vcre_mbx ID_mbx1
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .equ ID_mbx1,1
    setmbx: .space 12
    .include “mr32r.inc”
    .global  task1
task1:
        :
    ld24   R2,#setmbx
    ld24   R1,#0x02
    st     R1,@(4,R2)
    ld24   R1,#10
    st     R1,@(8,R2)
    ldi    R1,#0
    st     R1,@(12,R2)
    vcre_mbx ID_mbx1
        :
    ext_tsk
```

2.13.2. vdel_mbx(Delete Mailbox)

[(System call name)]

vdel_mbx → Delete Mailbox

[(Calling by the assembly language)]

```
.include “mr32r.inc”
vdel_mbx     vmbxid
```

<< Argument >>

vmbxid [**] The ID No. of a mailbox to be deleted

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a mailbox to be deleted
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER vdel_mbx ( vmbxid );
```

<< Argument >>

ID vmbxid; The ID No. of a mailbox to be deleted

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_NOEXS       0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

vdel_mbx deletes the mailbox vmbxid indicates.

You can create the mailbox deleted as the same ID again.If the task is linked to the message wait queue and vdel_mbx is issued for the mailbox,this system call normally end.In this case,vdel_mbx moves the task WAIT state to READY state.And error E_DLT is returned.If some messages are in the mailbox,these are deleted.

An error E_NOEXS is returned if this system call is issued for a nonexistent mailbox.

Make sure this system call is issued for only the mailbox that has been created by the vcre_mbx system call. If this system call is issued for the mailbox that has been defined by the configuration file, it does not function normally.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
#define ID_vmbx2 2
void task1(void)
{
        :
    vdel_mbx( ID_vmbx2 );
        :
    ext_tsk();
}

<< Usage example of the assembly language(CC32R) >>

ID_vmbx2:       .equ    2
    .include “mr32r.inc”
    .global   task1
task1:
        :
    vdel_mbx ID_vmbx2
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .equ ID_vmbx2,2
    .include “mr32r.inc”
    .global   task1
task1:
        :
    vdel_mbx ID_vmbx2
        :
    ext_tsk
```

2.13.3. vsnd_mbx(Send Message to Mailbox)

[(System call name)]

vsnd_mbx → Sends a message.

[(Calling by the assembly language)]

```
.include “mr32r.inc”
vsnd_mbx     vmbxid, pk_msg
```

<< Argument >>

mbxid	[**]	The ID No. of the mailbox to which a message is sent
pk_msg	[****]	The start address of message packet (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the mailbox to which a message is sent
R2	The start address of message packet
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER vsnd_mbx (vmbxid, pk_msg);
```

<< Argument >>

ID	vmbxid;	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.

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_NOEXS      0FFFFFFCCH(-H’00000034): Object does not exist
```

[(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. If there is any task waiting for a message, the message is passed to that task and the task has its wait state removed. In this case, the task removed its wait state recieves error code E_OK and pk_msg as the start address of the message packet.

If there is no task waiting fo amessage, the start address of the message packet is connected to message queue.If the attribute of the mailbox is specified **TA_MPRI(0x02)**, the massage is connected to the message queue in priority order. If the same value of the priority, the newer message is connected to the end of the message queue. The Operating system supposes that the head of the message packet has a T_MSG_PRI type message header, and get the priority of the message from its msgpri field.

If the attribute of the mailbox is specified **TA_MFIFO(0x00)**, the massage is connected to the message queue in FIFO order. Therefore, the newest message is connected to the end of the massage 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 visnd_mbx.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
typedef pri_message
{
    T_MSG_PRI  msgheader;
    char body[12];
} PRI_MSG;

void task(void)
{
    PRI_MSG    msg;
               :
    msg.msgpri = 5;
    if( vsnd_mbx( ID_msg,(T_MSG)&msg) != E_OK ){
        error(“error\n”);
    }
        :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task
msg:
    .res.w     3
    .SDATA “message”
    .DATA.B 0
task:
    ldi R1,#5
    ld24       R2,#msg
    st         R1,@(4,R2)
    vsnd_mbx ID_msg
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task
msg:
        .space 4*3
        .byte “message”
        .byte 0
task:
    vsnd_mbx ID_msg
        :
```

2.13.4. visnd_mbx(Send Message to Mailbox)

[(System call name)]

visnd_mbx → Sends a message. (for the handler only).

[(Calling by the assembly language)]

```
.include   “mr32r.inc”
visnd_mbx   mbxid, pk_msg
```

<< Argument >>

vmbxid	[**]	The ID No. of the mailbox to which a message is sent
pk_msg	[****]	The start address of message packet

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the mailbox to which a message is sent
R2	The start address of message packet
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER visnd_mbx (vmbxid, pk_msg);
```

<< Argument >>

ID	vmbxid;	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.

[(Error codes)]

```
E_OK            00000000H(-H’00000000): Normal End
E_NOEXS        0FFFFFFCCH(-H’00000034): Object does not exist
```

[(Function description)]

This system call is used when using the function of the vsnd_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 <mr32r.h>
#include “id.h”
typedef struct pri_message
{
       T_MSG_PRI      msgheader;
       char    body[12];
} PRI_MSG;

void inthand()
{
    PRI_MSG     msg;
                :
    if( visnd_mbx( ID_msg,(T_MSG)&msg) != E_OK )
        error(“overflow\n”);
                :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  intr
msg:
    .res.w      3
    .SDATA “message”
        .DATA.B 0
intr:
        :
    ld24        R1,#msg
    ld          R2,@R1
    visnd_mbx ID_msg
        :
    ret_int

<< Usage example of the assembly language(TW32R:DCC/M32R) >>
    .include “mr32r.inc”
    .global  intr
msg:
        .space 4*3
        .byte “message”
        .byte 0
intr:
        :
    ld24        R1,#msg
    ld          R2,@R1
    visnd_mbx ID_msg, msg
        :
    ret_int
```

2.13.5. vrcv_mbx(Receive Message from Mailbox)

[(System call name)]

vrcv_mbx → Waits for receiving a message.

[(Calling by the assembly language)]

```
.include "mr32r.inc"
vrcv_mbx     vmbxid
```

<< Argument >>

vmbxid	[**]	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 ID No. of the mailbox to which a message is received
R2	The start address of message packet
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER vrcv_mbx (ppk_msg, vmbxid);
```

<< Argument >>

ID	vmbxid;	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.

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_RLWAI       0FFFFFFAAH(-H'00000056): Wait state forcibly
                                       cleared
E_DLT         0FFFFFFAFH(-H'00000051): The object being waited for
                                       was deleted
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
```

[(Function description)]

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

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 mailbox, the task that has issued this system call is placed in a wait state and linked in a waiting queue. If the attribute of the mailbox specifies as **TA_TPRI(=0x01)**, the task is connected to the message wait queue in priority order. If in the same priority, the task is connected to the end of the message wait queue.

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.

Also, if the mailbox for a task waiting for conditions to be met is deleted by the vdel_mbx system call issued by another task, the waiting task is released from the transmit mailbox wait state and error E_DLT is returned to that task and changes to executable (READY) state.

An error E_NOEXS is returned if this system call is issued for a nonexistent mailbox.

You can issue this system call exclusively from a task. This system call, if issued either from the interrupt handler, the cyclic handler or the alarm handler, doesn't work properly.

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”

typedef struct fifo_message
{
       T_MSG   head;
       char    body[12];
} FIFO_MSG;
void task()
{
    FIFO_MSG *msg;
                :
    if( vrcv_mbx( ID_vmbx ,(T_MSG *)&msg ) != E_OK )
        error(“forced wakeup\n”);
                :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task
task:
        :
    vrcv_mbx ID_vmbx
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task
task:
        :
    vrcv_mbx ID_vmbx
        :
```

2.13.6. vtrcv_mbx(Receive Message with Timeout)

[(System call name)]

vtrcv_mbx → Waits for receiving a message. (With Timeout)

[(Calling by the assembly language)]

```
.include "mr32r.inc"
vtrcv_mbx     vmbxid,tmout
```

<< Argument >>

vmbxid	[**]	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 ID No. of the mailbox from which a message is received
R2	The start address of message packet
R3	--
R4	Timeout value

[(Calling by the C language)]

```
#include <mr32r.h>
ER vtrcv_mbx (ppk_msg, vmbxid, tmout);
```

<< Argument >>

ID	vmbxid;	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 >>

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

[(Error codes)]

```
E_OK          00000000H(-H'00000000): Normal End
E_TMOUT       0FFFFFFABH(-H'00000055): Polling failed or timeout
E_RLWAI       0FFFFFFAAH(-H'00000056): Wait state forcibly
                                       cleared
E_DLT         0FFFFFFAFH(-H'00000051): The object being waited for
                                       was deleted
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
```

[(Function description)]

This system call receives a message from the mailbox specified by vmbxid. 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. If the attribute of the mailbox specifies as **TA_TFIFO(=0x00)**, the task is connected to the message wait queue in FIFO order. If the attribute of the mailbox specifies as **TA_TPRI(=0x01)**, the task is connected to the message wait queue in priority order. If in the same priority, the task is connected to the end of the message 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.

- When the mailbox for which a task has been kept waiting is deleted by the del_mbx system call issued by another task

  Error code E_DLT is returned.

You can specify a timeout (tmout) of -1 to 0x7FFFFFFF. Specifying TMO_FEVR = -1 to vtrcv_mbx for tmout indicates that an infinite timeout value be used, resulting in exactly the same processing as vrcv_mbx. If you specify tmout as TMO_POL(=0), it works like vprcv_mbx.

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

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 <mr32r.h>
#include “id.h”
typedef struct fifo_message
{
       T_MSG   head;
       char    body[12];
} FIFO_MSG;
void task()
{
    FIFO_MSG   *msg;
                :
    if( vtrcv_mbx( ID_mbx,(T_MSG *)&msg , 10 ) != E_OK ){
        error(“Can’t Get Message\n”);
                :
}

<< Usage example of the assembly language(CC32R) >>
    .include “mr32r.inc”
    .global task
task:
        :
    vtrcv_mbx ID_mbx,10
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>
    .include “mr32r.inc”
    .global task
task:
        :
    vtrcv_mbx ID_mbx,10
        :
```

2.13.7. vprcv_mbx(Poll and Receive Message)

[(System call name)]

vprcv_mbx → Receiving a message. (no wait)

[(Calling by the assembly language)]

```
.include “mr32r.inc”
vprcv_mbx     vmbxid
```

<< Argument >>

vmbxid	[**]	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 ID No. of the mailbox from which a message is received
R2	The start address of message packet
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER vprcv_mbx (ppk_msg, vmbxid);
```

<< Argument >>

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

<< Return value >>

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

[(Error codes)]

```
E_OK          00000000H(-H’00000000): Normal End
E_TMOUT      0FFFFFFABH(-H’00000055): Polling failed or timeout
E_NOEXS      0FFFFFFCCH(-H’00000034): Object does not exist
```

[(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 mailbox, an error E_TMOUT is returned to the system call issued task and the task is not moved to WAIT state.

An error E_NOEXS is returned if this system call is issued for a nonexistent mailbox.

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

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

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
typedef struct fifo_message
{
       T_MSG   head;
       char    body[12];
} FIFO_MSG;
void task()
{
    FIFO_MSG * msg;
                :
    if( vprcv_mbx( ID_mbx ,(T_MSG *)&msg ) != E_OK ){
        error(“Can’t Get Message\n”);
                :
}

<< Usage example of the assembly language(CC32R) >>
    .include “mr32r.inc”
    .global  task
task:
    vprcv_mbx ID_mbx1
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>
    .include “mr32r.inc”
    .global  task
task:
    vprcv_mbx ID_mbx1
        :
```

2.13.8. vref_mbx(Refer Mailbox Status)

[(System call name)]

vref_mbx → Reference Mailbox Status

[(Calling by the assembly language)]

```
.include "mr32r.inc"
vref_mbx vmbxid
```

<< Argument >>

vmbxid	[**]	The ID No. of the mailbox to Reference Mailbox
pk_rmbx	[****]	Packet address to Reference Mailbox (Set the address in the R2 register.)

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of the mailbox to Reference Mailbox
R2	Packet address to Reference Mailbox
R3	--

The structure indicated by pk_rmbx returns the following data.

Offset	Size		
+0	2	wtsk	Waiting task information
+4	4(U)	pk_msg	Starting address of next received message packet

U: unsigned data.

[(Calling by the C language)]

```
#include <mr32r.h>
ER vref_mbx (pk_rmbx, vmbxid);
```

<< Argument >>

T_RMBX	*rmbx;	Packet address to Reference Mailbox
ID	vmbxid;	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 {
BOOL_ID     wtsk;          /* Waiting task information */
T_MSG       *pk_msg;       /* Starting address of next received message
                              packet */
} T_RVMBX;
```

[(Error codes)]

```
E_OK          00000000H(-H'00000000):Normal End
E_NOEXS       0FFFFFFCCH(-H'00000034): Object does not exist
```

[(Function description)]

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

- wtsk

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

- pk_msg

  pk_msg returns the message received (the first message in the queue) when vrcv_mbx or vtrcv_mbx is executed next. pk_msg returns NULL(=0). if there is no message.

An error E_NOEXS is returned if this system call is issued for a nonexistent mailbox.

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

[(Usage example)]

```
<< Usage example of the C language >>

#include <mr32r.h>
#include “id.h”
void task()
{
    T_RMBX rmbx;
        :
    ref_mbx(&vmbx, ID_mbx);
        :
}

<< Usage example of the assembly language(CC32R) >>

rmbx:    .RES.B 12
    .include “mr32r.inc”
    .global task
task:
        :
    ld24      R2,#rmbx
    ref_mbx    ID_mbx
        :

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

rmbx:    .space 12
    .include “mr32r.inc”
    .global task
task:
        :
    ld24      R2,#rmbx
    ref_mbx    ID_mbx
        :
```

2.13.9. vrst_mbx(Reset Message)

[(System call name)]

```
vrst_mbx  →  Clear all messages in the specified mailbox.
```

[(Calling by the assembly language)]

```
.include "mr32r.inc"
vrst_mbx     vmbxid
```

<< Argument >>

vmbxid	[**]	The ID No. to be cleared

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The ID No. of a mailbox
R2	--
R3	--

[(Calling by the C language)]

```
#include <mr32r.h>
ER vrst_mbx ( vmbxid );
```

<< Argument >>

ID	vmbxid;	The ID No. of a mailbox

<< Register setting >>

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

[(Error codes)]

```
E_OK          00000000H(-H'00000000) :  Normal End
E_NOEXS        0FFFFFFCCH(-H'00000034):  Object does not exist
```

[(Function description)]

Clear all messages in the specified mailbox.If there is no message in the mailbox, this system call does nothing.

An error E_NOEXS is returned if this system call is issued for a nonexistent mailbox.

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 <mr32r.h>
#include “id.h”
void task1(void)
{
        :
    vrst_mbx( ID_vmbx1 );
        :
}

<< Usage example of the assembly language(CC32R) >>

    .include “mr32r.inc”
    .global  task1
task1:
        :
    vrst_mbx ID_vmxbx1
        :
    ext_tsk

<< Usage example of the assembly language(TW32R:DCC/M32R) >>

    .include “mr32r.inc”
    .global  task1
task1:
        :
    vrst_mbx ID_vmbx1
        :
    ext_tsk
```

Chapter 3 Appendix

3.1. List of System calls

Task Manegement Functions

System call		Function	Scheduler
cre_tsk	[E]	Create Task	call
del_tsk	[E]	Delete Task	call
sta_tsk	[S]	Starts a task.	call
ista_tsk	[S]	Starts a task.(handler only)	--
ext_tsk	[S]	Normally ends the self task.	call
exd_tsk	[E]	Exit and delete 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
cre_flg	[E]	Create Eventflag	call
del_flg	[E]	Delete Eventflag	call
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.	--
cre_sem	[E]	Create Semaphore	call
del_sem	[E]	Delete Semaphore	call
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.	--
cre_mbx	[E]	Create Mailbox	call
del_mbx	[E]	Delete Mailbox	call
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.	--

Rendezvous

System call		Function	Scheduler
cre_mbf	[E]	Create Messagebuffer	call
del_mbf	[E]	Delete Messagebuffer	call
snd_mbf	[E]	Sends a message	call
tsnd_mbf	[E]	Sends a message (With Timeout)	call
psnd_mbf	[E]	Sends a message (no wait)	--
rcv_mbf	[E]	Waits for receiving a message from Messagebuffer	call
trcv_mbf	[E]	Waits for receiving a message from Messagebuffer (With Timeout)	call
prcv_mbf	[E]	Waits for receiving a message from Messagebuffer (no wait)	call
ref_mbf	[E]	Reference Messagebuffer Status	--
cre_por	[E]	Create Port for Rendezvous	call
del_por	[E]	Delete Port for Rendezous	call
cal_por	[E]	Call Port for Rendezous	call
tcal_por	[E]	Call Port for Rendezous (With Timeout)	call
pcal_por	[E]	Call Port for Rendezous (no wait)	call
acp_por	[E]	Accept Port for Rendezous	call
tacp_por	[E]	Accept Port for Rendezous (With Timeout)	call
pacp_por	[E]	Accept Port for Rendezous (no wait)	call
fwd_por	[E]	Forward Rendezous to Other Port	call
rpl_rdv	[E]	Reply Rendezous	call
ref_por	[E]	Reference Port Status	call

Interrupt Management Functions

System call		Function	Scheduler
def_int	[C]	Define Interrupt Handler	call
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
cre_mpf	[E]	Create Fixed-size Memorypool	call
del_mpf	[E]	Delete Fixed-size Memorypool	call
get_blf	[E]	Gets a fixed-size memory block	call
tget_blf	[E]	Gets a fixed-size memory block (With Timeout)	call
pget_blf	[E]	Gets fixed-size memory block (no wait)	--
rel_blf	[E]	Release fixed-size memory block.	call
ref_mpf	[E]	Reference fixed-size Memorypool status.	--
cre_mpl	[E]	Create Variable-size Memorypool	call
del_mpl	[E]	Delete Variable-size Memorypool	call
get_blk	[E]	Gets a variable-size memory block	call
tget_blk	[E]	Gets a variable-size memory block (With Timeout)	call
pget_blk	[E]	Gets variable-size memory|block. (no wait)	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
def_cyc	[E]	Define cyclic handler.	call
act_cyc	[E]	Controls activation of the cyclic handler.	--
ref_cyc	[E]	Reference Cyclic handler Status.	--
ref_alm	[E]	Reference Alarm Handler Status.	--

System Management Function

System call		Function	Scheduler
get_ver	[R]	Gets the OS version number.	--
ref_sys	[E]	Reference Status of CPU and OS.	--
def_exc	[C]	Define Exception Handler	call

Implementation-Dependent System Call

System call		Function	Scheduler
vrst_msg	[--]	Cears messages in mailbox	--
vrst_blf	[--]	Releases all specified fixed-size memory blocks	call
vrst_blk	[--]	Releases all specified valiable-size memory blocks	call
vrst_mbf	[--]	Clears message in message buffer	call
vclr_ems	[--]	Clear Exception Mask	call
vset_ems	[--]	Set Exception Mask	call
vras_fex	[--]	Raise forcibly exception	call

Implementation-Dependent System Call (Mailbox)

System call		Function	Scheduler
vcre_mbx	[--]	Create mailbox with priority	call
vdel_mbx	[--]	Delete mailbox with prioity	call
vsnd_mbx	[--]	Sends a message with priority to the mailbox	call
visnd_mbx	[--]	Sends a message with priority to the mailbox (Handler only)	--
vrcv_mbx	[--]	Recieves a message with priority to the mailbox (without timeout)	call
vtrcv_mbx	[--]	Recieves a message with priority to the mailbox (with)	call
vprcv_mbx	[--]	Recieves a message with priortimeoutity to the mailbox (without waiting)	--
vrst_mbx	[--]	Resets the mailbox with priority	--
vref_mbx	[--]	Refers the status of the mailbox with priority	--

3.2. List of Error code

Error code	Value	Description
E_OK	00000000H(-H'00000000)	Normal End
E_OBJ	0FFFFFFC1H(-H'0000003F)	Invalid object state
E_QOVR	0FFFFFFB7H(-H'00000049)	Queuing or nest overflow
E_TMOUT	0FFFFFFABH(-H'00000055)	Polling failed or timeout
E_RLWAI	0FFFFFFAAH(-H'00000056)	Wait state forcibly cleared
E_NOEXS	0FFFFFFCCH(-H'00000034)	Object does not exist
E_DLT	0FFFFFFAFH(-H'00000051)	The object being waited for was deleted
E_NOMEM	0FFFFFFF6H(-H'0000000A)	Not enough of memory

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 MR32R. 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

Systemcall	INT No.	Parameter			Return Parameter	
		R0 (Function code)	R1	R2	R0	R1
cre_tsk	#7	H'00	tskid	pk_ctsk	ercd	
del_tsk	#7	H'04	tskid		ercd	
sta_tsk	#7	H'08	tskid	stacd	ercd	
ista_tsk	#8	H'60	tskid	stacd	ercd	
ext_tsk	#8	H'bc				
exd_tsk	#8	H'c0				
ter_tsk	#7	H'0c	tskid		ercd	
dis_dsp	#8	H'b4			ercd	
ena_dsp	#7	H'1c			ercd	
chg_pri	#7	H'10	tskid	tskpri	ercd	
ichg_pri	#8	H'64	tskid	tskpri	ercd	
rot_rdq	#7	H'14		tskpri	ercd	
irot_rdq	#8	H'68		tskpri	ercd	
rel_wai	#7	H'18	tskid		ercd	
irel_wai	#8	H'24	tskid		ercd	
get_tid	#8	H'70			ercd	tskid
ref_tsk	#8	H'd4	tskid	pk_rtsk	ercd	

Synchronization Functions Attached to Task

Systemcall	INT No.	Parameter			Return Parameter	
		R0 (Function code)	R1	R2	R0	R2
sus_tsk	#7	H'20	tskid		ercd	
isus_tsk	#8	H'74	tskid		ercd	
rsm_tsk	#7	H'24	tskid		ercd	
irsm_tsk	#8	H'78	tskid		ercd	
slp_tsk	#7	H'28			ercd	
tslp_tsk	#7	H'28		tmout	ercd	
wup_tsk	#7	H'2c	tskid		ercd	
iwup_tsk	#8	H'7c	tskid		ercd	
can_wup	#8	H'80	tskid		ercd	wupcnt

Synchronization and Communication Functions

Systemcall	INT No.	Parameter					Return Parameter		
		R0 (Function code)	R1	R2	R3	R4	R0	R2	R3
cre_flg	#7	H'f4	flgid	pk_cflg			ercd		
del_flg	#7	H'f8	flgid				ercd		
set_flg	#7	H'30	flgid	setptn			ercd		
iset_flg	#8	H'84	flgid	setptn			ercd		
clr_flg	#8	H'88	flgid	clrptn			ercd		
wai_flg	#7	H'34	flgid	waiptn	wfmode		ercd	flgptn	
twai_flg	#7	H'34	flgid	waiptn	wfmode	tmout	ercd	flgptn	
pol_flg	#8	H'8c	flgid	waiptn	wfmode		ercd	flgptn	
ref_flg	#8	H'd8	flgid	pk_rflg			ercd		
cre_sem	#7	H'10c	semid	pk_csem			ercd		
del_sem	#7	H'110	semid				ercd		
sig_sem	#7	H'38	semid				ercd		
isig_sem	#8	H'90	semid				ercd		
wai_sem	#7	H'3c	semid				ercd		
twai_sem	#7	H'3c	semid			tmout	ercd		
preq_sem	#8	H'94	semid				ercd		
ref_sem	#8	H'dc	semid	pk_rsem			ercd		
cre_mbx	#7	H'fc	mbxid	pk_cmbx			ercd		
del_mbx	#7	H'100	mbxid				ercd		
snd_msg	#7	H'40	mbxid	pk_msg			ercd		
isnd_msg	#8	H'98	mbxid	pk_msg			ercd		
rcv_msg	#7	H'44	mbxid				ercd	pk_msg	
trcv_msg	#7	H'44	mbxid			tmout	ercd	pk_msg	
prcv_msg	#8	H'9c	mbxid				ercd	pk_msg	
ref_mbx	#8	H'20	mbxid	pk_rmbx			ercd		

Rendezvous

System call	INT No.	Parameter							Return Parameter		
		R0 (Function code)	R1	R2	R3	R4	R5	R6	R0	R2	R3
cre_mbf	#7	H'118	mbfid	pk_cmbf					ercd		
del_mbf	#7	H'11c	mbfid						ercd		
snd_mbf	#7	H'c8	mbfid	msg	msgsz				ercd		
tsnd_mbf	#7	H'c8	mbfid	msg	msgsz	tmout			ercd		
psnd_mbf	#7	H'c8	mbfid	msg	msgsz				ercd		
rcv_mbf	#7	H'124	mbfid	msg					ercd		
trcv_mbf	#7	H'124	mbfid	msg		tmout			ercd		msgsz
prcv_mbf	#7	H'124	mbfid	msg					ercd		msgsz
ref_mbf	#8	H'114	mbfid	pk_rmbf					ercd		msgsz
cre_por	#7	H'144	porid	pk_cpor					ercd		
del_por	#7	H'148	porid						ercd		
cal_por	#7	H'14c	porid		cmsgsz		msg	calptn	ercd	rmsgsz	
tcal_por	#7	H'14c	porid		cmsgsz		msg	calptn	ercd	rmsgsz	
pcal_por	#7	H'14c	porid		cmsgsz		msg	calptn	ercd	rmsgsz	
acp_por	#7	H'150	porid			tmout	msg	acpptn	ercd	cmsgsz	rdvno
tacp_por	#7	H'150	porid				msg	acpptn	ercd	cmsgsz	rdvno
pacp_por	#7	H'150	porid				msg	acpptn	ercd	cmsgsz	rdvno
fwd_por	#7	H'158	porid	rdvno	cmsgsz		msg	calptn	ercd		
rpl_rdv	#7	H'154	rdvno	msg	rmsgsz				ercd		
ref_por	#8	H'd0	porid	pk_rpor					ercd		

Interrupt Management Functions

Systemcall	INT No.	Parameter			Return Parameter	
		R0 (Function code)	R1	R2	R0	R2
def_int	#7	H'128	dintno	pk_dint	ercd	
ret_int						
loc_cpu	#8	H'b8			ercd	blf
unl_cpu	#7	H'58			ercd	blf

Memorypool Management Functions

Systemcall	INT No.	Parameter					Return Parameter	
		R0 (Function code)	R1	R2	R3	R4	R0	R2
cre_mpf	#7	H'160	mpfid	pk_cmpf			ercd	
del_mpf	#7	H'164	mpfid				ercd	
get_blf	#7	H'c4	mpfid				ercd	blf
tget_blf	#7	H'c4	mpfid			tmout	ercd	blf
pget_blf	#8	H'48	mpfid				ercd	blf
rel_blf	#7	H'4c	mpfid	blf			ercd	
ref_mpf	#8	H'e8	mpfid	pk_rmpf			ercd	
cre_mpl	#7	H'104	mplid	pk_cmpl			ercd	
del_mpl	#7	H'108	mplid				ercd	
get_blk	#7	H'50	mplid		blksz		ercd	blk
tget_blk	#7	H'50	mplid		blksz	tmout	ercd	blk
pget_blk	#7	H'50	mplid		blksz		ercd	blk
rel_blk	#7	H'54	mplid	blk			ercd	
ref_mpl	#8	H'e4	mplid	pk_rmpl			ercd	

TIme Management Functions

Systemcall	INT No.	Parameter			Return Parameter
		R0 (Function code)	R1	R2	R0
set_tim	#8	H'a0		pk_tim	ercd
get_tim	#8	H'a4		pk_tim	ercd
dly_tsk	#7	H'5c		dlytim	ercd
def_cyc	#7	H'1a0		pk_dcyc	ercd
act_cyc	#8	H'a8	cycno	cycact	ercd
ref_cyc	#8	H'ec	cycno	pk_rcyc	ercd
ref_alm	#8	H'f0	almno	pk_ralm	ercd

System Management Function

Systemcall	INT No.	Parameter			Return Parameter
		R0 (Function code)	R1	R2	R0
get_ver	#8	H'ac		pk_ver	ercd
ref_sys	#8	H'15c		pk_rsys	ercd
def_exc	#7	H'12c	exckind	pk_dexc	ercd

Implementation-Dependent System Call

Systemcall	INT No.	Parameter			Return Parameter
		R0 (Function code)	R1	R2	R0
vclr_ems	#7	H'130	tskid		ercd
vset_ems	#7	H'134	tskid		ercd
vret_exc	#7	H'168			
vras_fex	#7	H'138	tskid	exccd	ercd
vrst_blf	#7	H'170	mpfid		ercd
vrst_blk	#7	H'16c	mplid		ercd
vrst_msg	#8	H'178	mbxid		ercd
vrst_mbf	#7	H'174	mbfid		

Implementation-Dependent System Call (Mailbox)

Systemcall	INT No.	Parameter					Return Parameter	
		R0 (Function code)	R1	R2	R3	R4	R0	R2
vcre_mbx	#7	H'180	vmbxid	pk_cvmbx			ercd	
vdel_mbx	#7	H'184	vmbxid				ercd	
vsnd_mbx	#7	H'188	vmbxid	pk_vmbx			ercd	
visnd_mbx	#8	H'18c	vmbxid	pk_vmbx			ercd	
vrcv_mbx	#7	H'190	vmbxid				ercd	
vtrcv_mbx	#8	H'190	vmbxid			tmout	ercd	pk_msg
vprcv_mbx	#8	H'194	vmbxid				ercd	pk_msg
vref_mbx	#7	H'198	vmbxid	pk_vrmbx			ercd	pk_msg
vrst_mbx	#8	H'19c	vmbxid				ercd	

3.4. C Language Interface

Task Manegement Functions

ER	ercd =	cre_tsk	(ID tskid,T_CTSK *pk_ctsk);
ER	ercd =	del_tsk	(ID tskid);
ER	ercd =	sta_tsk	(ID tskid, INT stacd);
ER	ercd =	ista_tsk	(ID tskid, INT stacd);
	void	ext_tsk	();
	void	exd_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 =	cre_flg	(ID flgid,T_CFLG *pk_cflg);
ER	ercd =	del_flg	(ID flgid);
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 =	cre_sem	(ID semid, T_CSEM *pk_csem);
ER	ercd =	del_sem	(ID semid);
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_sem	(ID semid);
ER	ercd =	ref_sem	(T_RSEM *pk_rsem, ID semid);
ER	ercd =	cre_mbx	(ID mbxid, T_CMBX *pk_cmbx);
ER	ercd =	del_mbx	(ID mbxid);
ER	ercd =	snd_msg	(ID mbxid, T_MSG *pk_msg);
ER	ercd =	isnd_msg	(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_msg	(T_MSG **ppk_msg, ID mbxid);
ER	ercd =	ref_mbx	(T_RMBX *pk_rmbx, ID mbxid);
ER	ercd =	cre_mbf	(ID mbfid, T_CMBF *pk_rmbf);
ER	ercd =	del_mbf	(ID mbfid);
ER	ercd =	snd_mbf	(ID mbfid, VP msg, INT msgsz);
ER	ercd =	tsnd_mbf	(ID mbfid, VP msg, INT msgsz, TMO tmout);
ER	ercd =	psnd_mbf	(ID mbfid, VP msg, INT msgsz);
ER	ercd =	rcv_mbf	(VP msg, INT *p_msgsz, ID mbfid);
ER	ercd =	trcv_mbf	(VP msg, INT *p_msgsz, ID mbfid, TMO tmout);
ER	ercd =	prcv_mbf	(VP msg, INT *p_msgsz, ID mbfid);
ER	ercd =	ref_mbf	(T_RMBF *pk_rmbf, ID mbfid);

Rendezvous

ER	ercd =	cre_por	(ID porid,T_CPOR *pk_cpor);
ER	ercd =	del_por	(ID porid);
ER	ercd =	cal_por	(VP msg, INT *p_rmsgsz, ID porid, UINT calptn, INT cmsgsz);
ER	ercd =	tcal_por	(VP msg, INT *p_rmsgsz, ID porid, UINT calptn, INT cmsgsz, TMO tmout);
ER	ercd =	pcal_por	(VP msg, INT *p_rmsgsz, ID porid, UINT calptn, INT cmsgsz);
ER	ercd =	acp_por	(RNO *p_rdvno, VP msg, INT *p_cmsgsz, ID porid, UINT acpptn);
ER	ercd =	tacp_por	(RNO *p_rdvno, VP msg, INT *p_cmsgsz, ID porid, UINT acpptn, TMO tmout);
ER	ercd =	pacp_por	(RNO *p_rdvno, VP msg, INT *p_cmsgsz, ID porid, UINT acpptn);
ER	ercd =	fwd_por	(ID porid, UINT calptn, RNO rdvno, VP msg, INT cmsgsz);
ER	ercd =	rpl_rdv	(RNO rdvno, VP msg, INT rmsgsz);
ER	ercd =	ref_por	(T_RPOR *pk_rpor, ID porid);

Interrupt Management Functions

ER	ercd =	def_int	(UINT dintno, T_DINT *pk_dint);
	void	ret_int	();
ER	ercd =	loc_cpu	();
ER	ercd =	unl_cpu	();

Memorypool Management Functions

```
ER    ercd =   cre_mpf     (ID mpfid, T_CMPF *pk_cmpf);
ER    ercd =   del_mpf     (ID mpfid);
ER    ercd =   get_blf     (VP *p_blf, ID mpfid);
ER    ercd =   tget_blf    (VP *p_blf, ID mpfid, TMO tmout);
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 =   cre_mpl     (ID mplid, T_CMPL *pk_cmpl);
ER    ercd =   del_mpl     (ID mplid);
ER    ercd =   get_blk     (VP *p_blk, ID mplid, INT blksz);
ER    ercd =   tget_blk    (VP *p_blk, ID mplid, INT blksz,TMO tmout);
ER    ercd =   pget_blk    (VP *p_blk, ID mplid, INT blksz);
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);
```

System Management Function

```
ER    ercd =   get_ver     (T_VER *pk_ver);
ER    ercd =   ref_sys     (T_RSYS *pk_rsys);
ER    ercd =   def_exc     (UINT exckind, T_DEXC *pk_dexc);
```

Implementation-Dependent System Call

```
ER    ercd =   vclr_ems    (ID tskid);
ER    ercd =   vset_ems    (ID tskid);
ER    ercd =   vras_fex    (ID tskid, UW exccd);
ER    ercd =   vrst_blf    (ID mpfid);

ER    ercd =   vrst_blk    (ID mplid);
ER    ercd =   vrst_msg    (ID mbxid);
ER    ercd =   vrst_mbf    (ID mbfid);
```

Implementation-Dependent System Call (Mailbox)

```
ER    ercd =   vcre_mbx    (ID vmbxid, T_CVMBX *pk_rmbf);
ER    ercd =   vdel_mbx    (ID vmbxid);
ER    ercd =   vsnd_mbx    (ID vmbxid, T_MSG *pk_msg);
ER    ercd =   visnd_mb    (ID vmbxid, T_MSG *pk_msg);
                  x
ER    ercd =   vrcv_mbx    (**ppk_msg, ID vmbxid);
ER    ercd =   vtrcv_mbx   (**ppk_msg, ID vmbxid, TMO tmout);
ER    ercd =   vprcv_mb    (**ppk_msg, ID vmbxid);
                  x
ER    ercd =   vref_mbx    (T_RVMBF *pk_rvmbf,ID vmbxid);
ER    ercd =   vrst_mbx    (ID vmbxid);
```

3.5. Data Type

```
typedef     char            B;          /* Signed 8-bit integer */
typedef     short           H;          /* Signed 16-bit integer */
typedef     long            W;          /* Signed 32-bit integer */
typedef     unsigned char   UB;         /* Unsigned 8-bit integer */
typedef     unsigned short  UH;         /* Unsigned 16-bit integer */
typedef     unsigned long   UW;         /* Unsigned 32-bit integer */
typedef     char            VB          /* Unpredicatable data, signed (8-bit size) */
typedef     short           VH;         /* Unpredicatable data, signed (16-bit size) */
typedef     long            VW;         /* Unpredicatable data, signed (32-bit size) */
typedef     void            *VP;        /* Pointer to Unpredicatable data */
typedef     void            (*FP)();    /* Start address of program general */
typedef     W               INT         /* Signed 32-bit integer */
typedef     UW              UINT;       /* Unsigned 32-bit integer */
typedef     W               RNO         /* Rendezvous number */
typedef     H               ID;         /* ID number of object */
typedef     H               PRI;        /* Task priority */
typedef     H               TMO;        /* Timeout */
typedef     H               HNO;        /* ID number of handler */
typedef     INT             ER;         /* Error code */
typedef     INT             ATR;        /* Object attribute(unsigned) */
typedef     INT             DLYTIME;    /* Delay time */
typedef     INT             CYCTIME;    /* Interval of cyclic handler starts*/
typedef     H               BOOL_ID;    /* Boolean value or ID number */
typedef     UINT            PSW;        /* PSW value */
typedef     void            *PT_MSG;    /* 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 ----
 typedef struct t_ctsk {
    VP          exinf;          /* Extended information */
    ATR         tskatr;         /* Task attribute */
    FP          task;           /* Task startup address */
    PRI         itskpri;        /* Priority in task startup */
    INT         stksz;          /* Stack size */
 } T_CTSK;

 TSK_SELF       0       /* Own task specification */
 TPRI_RUN       0       /* Specifies the highest priority then under execution */

 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 */
    INT         wupcnt;         /* Number of queued wakeup requests */
    ATR         tskatr;         /* Task attributes */
    FP          task;           /* Task starting address */
    PRI         itskpri;        /* Initial task priority */
    INT         stksz;          /* Stack size */
    UW          epndptn;        /* Pending exception class pattern */
 };

---- Related to Semaphore ----
 typedef struct t_csem {
    VP          exinf;          /* Extended information */
    ATR         sematr;         /* Semaphore attribute */
    INT         isemcnt;        /* Initial semaphore count */
    INT         maxsem;         /* Maximun semaphore count */
 } T_CSEM;

 typedef struct t_rsem {
    VP          exinf;          /* Extended information */
    BOOL_ID     wtsk;           /* Waiting task information */
    INT         semcnt;         /* Current semaphore count */
 } T_RSEM;
```

```
---- Related to Eventflag ----
 typedef struct t_cflg {
    VP          exinf;          /* Extended information */
    ATR         flgatr;         /* Task attribute */
    UINT        iflgptn;        /* Initial eventflag pattern */
 } T_CFLG;

 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 Mailbox ----
 typedef struct t_cmbx {
    VP          exinf;          /* Extended information */
    ATR         mbxatr;         /* Mailbox attribute */
    INT         bufcnt;         /* Ringbuffer size */
 } T_CMBX;

 typedef struct t_rmbx {
    VP          exinf;  /* Extended information */
    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;

---- Related to Messagebuffer ----
 typedef struct t_cmbf {
    VP          exinf;          /* Extended information */
    ATR         mbfatr;         /* Messagebuffer attribute */
    INT         bufsz;          /* Messagebuffer size */
    INT         maxmsz;         /* Maximum size of message */
 } T_CMBF;

 typedef struct t_rmbf {
    VP      exinf;              /* Extended information */
    BOOL_ID     wtsk;           /* Waiting Task Information */
    BOOL_ID     stsk;           /* Sending Task Information */
    INT         msgsz           /* Message Size (in bytes) */
    INT         frbufsz;        /* Free Buffer Size (in bytes) */
 } T_RMBF;

---- Related to Rendezvous ----
 typedef struct t_cpor {
    VP          exinf;          /* Extended information */
    ATR         poratr;         /* Port for redenzvous attribute */
    INT         maxcmsz;        /* Maximum call message size */
    INT         maxrmsz;        /* Maximum reply message size */
 } T_CPOR;

 typedef struct t_rpor {
    VP          exinf;          /* Extended information */
    BOOL_ID     wtsk;           /* Waiting Task Information */
    BOOL_ID     atsk;           /* Accepting Task Information */
 } T_RPOR;
```

```
---- Related to Interrupt ----
 typedef struct t_dint {
    ATR         intatr;         /* Interrupt handler attribute */
    FP          inthdr;         /* Interrupt handler startup address */
 } T_DINT;

---- Related to Fixed-size Memorypool ----
typedef struct t_cmpf {
    VP          exinf;          /* Extended information */
    ATR         mpfatr;         /* Fixed-size memorypool attribute */
    INT         mpfcnt;         /* Memory block count */
    INT         blfsz;          /* Fixed-size memorypool size */
} T_CMPF;

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;

---- Related to Variable-size Memorypool ----
 typedef struct t_cmpl {
    VP          exinf;          /* Extended information */
    ATR         mplatr;         /* Variable-size memorypool attribute */
    INT         mplsz;          /* Variable-size memorypool size */
    INT         maxblksz;       /* Maximum memory block size to be allocated */
 } T_CMPL;

 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 */
    UW          ltime;          /* 32 high-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;

 typedefstruct t_rsys {
    INT         sysstat;        /* System status */
    ID          runtskid;       /* The ID No. of RUN state task */
    PRI         runtskpri;      /* The priority of RUN state task */
    UINT        psw;            /* PSW */
 } T_RSYS;

 typedef struct t_dexc {
    ATR         excatr;         /* Exception handler attribute */
    FP          exchdr;         /* Exception handler startup address */
    ID          tskid;          /* The ID No. of task */
    W           excstksz;       /* Stack size */
  T_DEXC;

 typedef struct t_regs {
    VW          r0;
    VW          r1;
    VW          r2;
    VW          r3;
    VW          r4;
    VW          r5;
    VW          r6;
    VW          r7;
    VW          r8;
    VW          r9;
    VW          r10;
    VW          r11;
    VW          r12;
    VW          r13;
    VW          r14;
    VW          sp;
    VW          accl;
    VW          acch;
 };

 typedefstruct  t_reit {
    PSW         psw;
    FP          pc;
 };

 typedefstruct  t_exc {
    W           exckind;
    UW          exccd;
    ID          tskid;
    UW          exeenv;
 };
```

/* Related to Implementation-Dependent System Call (Mailbox) */

```
typedef struct t_cvmbx {
        ATR     mbxatr;  /* Mailbox attribute */
        PRI     maxpri;  /* Max priority value of the message */
        VP      mprihd;  /* The start address of the message queue header area
```

```
*/
} T_CVMBX;
typedef struct t_vmbx {
        BOOL_ID wtsk;   /* Waiting task information */
        T_MSG   pk_msg; /* Starting address of next received message packet
*/
} T_RVMBX;
```

Index

M3T-MR32R V.3.50 Reference Manual

Rev. 1.00
June 1, 2003
REJ10J0085-0100Z

M3T-MR32R V.3.50
Reference Manual

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ10J0085-0100Z