お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

User's Manual

RENESAS

M3T-MR32R V.3.50

リファレンスマニュアル

M32R ファミリ用リアルタイムOS

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2003.06

安全設計に関するお願い

- 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

- 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報について株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
- 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは責任を負いません。
- 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは、予告なしに、本資料に記載した製品又は仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前に株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又は特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ（http://www.renesas.com）などを通じて公開される情報に常にご注意ください。
- 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズはその責任を負いません。
- 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは、適用可否に対する責任は負いません。
- 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又は特約店へご照会ください。
- 本資料の転載、複製については、文書による株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズの事前の承諾が必要です。
- 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたら株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又は特約店までご照会ください。

製品の内容及び本書についてのお問い合わせ先

インストーラが生成する以下のテキストファイルに必要事項を記入の上、ツール技術サポート窓口 support_tool@renesas.com まで送信ください。

¥SUPPORT¥製品名¥SUPPORT.TXT

株式会社ルネサス ソリューションズ マイコンツール部
ツール技術サポート窓口　support_tool@renesas.com
ユーザ登録窓口　regist_tool@renesas.com
ホームページ　http://www.renesas.com/jp/tools

はじめに

M3T-MR32R(**以下** MR32R **と略す**)は M32R **ファミリ**用のリアルタイム・オペレーティングシステム[1]です。MR32R は**μ**ITRON 仕様[2]に準拠しています。

本マニュアルは MR32R を使用したプログラムの作成手順および作成上の注意事項について説明します。各システムコールの詳細な使用方法については「MR32R リファレンスマニュアル」を参照してください。

MR32R を使うために必要なこと

MR32R を使用したプログラムを作成するには弊社下記製品または、サードパーティ製品を別途御購入して頂く必要があります。

- M32R ファミリクロスツール M3T-CC32R(以下 CC32R と略す)
- M32R **ファミリ** GNU クロスツール M3T-TW32R(以下 TW32R と略す)
- M32R ファミリ Wind River Systems,Inc 製コンパイラ D-CC/M32R

これらの製品をあわせて御使用頂ければ、より効率の良いプログラム開発がおこなえます。

ドキュメント一覧

MR32R に添付されているドキュメントは以下の３種類あります。

- リリースノート
  ソフトウェアの概要やユーザーズマニュアル、リファレンスマニュアルの訂正などを記載したドキュメントです。
- ユーザーズマニュアル (PDF ファイル)
  MR32R を使用したプログラムの作成手順や作成上の注意事項を記載したドキュメントです。
- リファレンスマニュアル (PDF ファイル)
  MR32R のシステムコールの使用方法や使用例を記述したドキュメントです。
  本マニュアルを読む前に必ずリリースノートをお読みください。

ソフトウェアの使用権

ソフトウェアの使用権はソフトウェア使用権許諾契約書に基づきます。MR32R はお客様の製品開発の目的でのみ使用できます。その他の目的での使用はできませんのでご注意ください。

また、本マニュアルによってソフトウェアの使用権の実施に対する保証及び使用権の実施の許諾を行うものではありません。

---

[1] 以降リアルタイム OS と略します。

2 μITRON 仕様は、東京大学理学部坂村健博士とその研究室により考案されたものです。したがって、μITRON 仕様の著作権は同氏に属しています。**MR32R** は同氏に承認を得て、μITRON 仕様に基づき製作されたものです。

目次

第1章　システムコールリファレンスの見方

1.1. システムコールリファレンスの見方

システムコールリファレンスは、以下の形式で記述しています。

【システムコール名】

システムコールの名称 → システムコールの機能

【アセンブリ言語による呼び出し方法】

.include “mr32r.inc”
アセンブリ言語から **MR32R** 機能呼び出し方法

《引数》

システムコールのパラメータの説明
パラメータはマクロの引数として記述します。
引数名　サイズ　説明
サイズは、以下の記号で示しています。
[-*]　1 バイトデータ
[**]　2 バイトデータ
[****]　4 バイトデータ

《レジスタ設定》

システムコールマクロ発行後のレジスタに設定される値を示します。

レジスタ名	システムコール発行後の内容
※1	※2

※1 レジスタ名。この欄には、R0、R1、R2、R3、R4、R5、R6 を記述しています。
※2 各レジスタに何が設定されるかを示しています。‘—‘が記述してあれば、「不定」を表わします。

PSW の SM、IE、C は、システムコール発行前の値が保存されてますが、BSM、BIC、BC は不定となります。

各(リターン)パラメータが使用するレジスタは、おおよそ以下のように決めています。

R0 レジスタ(32 ビット)	機能コード、エラーコード
R1 レジスタ(32 ビット)	オブジェクト ID 番号
R2 レジスタ(32 ビット)	パケットアドレス、その他のパラメータ
R3 レジスタ(32 ビット)	上記におさまらないもの
R4 レジスタ(32 ビット)	タイムアウト値
R5 レジスタ(32 ビット)	メッセージを格納したアドレス
R6 レジスタ(32 ビット)	ランデブのビットパターン

【 C 言語による呼び出し方法 】

C 言語からの **MR32R** 機能呼び出し方法

《引数》

引数の型の宣言

《戻り値》

呼び出しの結果の戻り値の説明
なおシステムコールリファレンス文中で使用されているデータ型はインクルードファイル”mr32r.h”に定義されています。付録にその定義を示しています。

【エラーコード】

エラーコード名　エラーコード値　エラーコードの意味

エラーコードの文字列、例えば E_OK などは”mr32r.h”に”#define”を使って、”mr32r.inc” に “.EQU” を使って定義されている。エラー判定をプログラム中で記述する場合は、この定義された文字列を用いなければなりません。

【機能説明】

機能の詳細説明

【使用例】

使用例

1.2. 各システムコールのスタック使用量

表 1.1は、タスクから発行可能なシステムコールのスタック使用量(システムスタック)を示しています。

()内は、D-CC/M32R 使用時のスタック消費サイズです。

*印は、ユーザスタックを消費します。

表 1.1　タスクから発行するシステムコールのスタック使用量一覧(単位:バイト)

システムコール	システムコール処理		C 言語インタフェース	
	CC32R	TW32R D-CC/M32R	CC32R	TW32R D-CC/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	12
twai_flg	0	0	8	12
cre_sem	0	0	4	4
del_sem	0	0	4	4
sig_sem	0	0	4	4
wai_sem	0	0	4	8
twai_sem	0	0	4	8
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

システムコール	システムコール処理		C 言語インタフェース	
	CC32R	TW32R D-CC/M32R	CC32R	TW32R D-CC/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
vret_exc	0	0	4	4
vrst_blf	0	0	4	4
vrst_blk	40	20(32)	4	4
vrst_mbf	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

表 1.2は、ハンドラから発行可能なシステムコールのスタック使用量(システムスタック)を示しています。

表 1.2 ハンドラから発行するシステムコールのスタック使用量一覧(単位:バイト)

システムコール	システムコール処理		C 言語インタフェース	
	CC32R	TW32R D-CC/M32R	CC32R	TW32R D-CC/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	48	48	4	4

表 1.3は、タスクあるいはハンドラの両方から発行可能なシステムコールのスタック使用量を示しています。ここで示すスタックの使用量は、タスクからシステムコールを発行した場合は、ユーザスタックを使用し、ハンドラから発行した場合は、システムスタックを使用します。

表 1.3　タスク、ハンドラの両方から発行可能なシステムコールのスタック使用量一覧

システムコール	システムコール処理		C 言語インタフェース	
	CC32R	TW32R D-CC/M32R	CC32R	TW32R D-CC/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. スタックサイズの算出方法

MR32R のスタックには、システムスタックとユーザスタックの 2 種類があります。スタックサイズの計算方法は、ユーザースタックとシステムスタックで異なります。

●ユーザスタック

タスクに存在するスタックです。従って、MR32R を使ってアプリケーションプログラムを記述する場合には、各タスクごとにスタック領域を確保する必要があります。

●システムスタック

MR32R 内部もしくはハンドラ実行中に使用するスタックサイズです。MR32R では、システムコールをタスクが発行するとユーザースタックからシステムスタックに切り替えます。システムスタックは、マイコンの割り込みスタックを使用します。

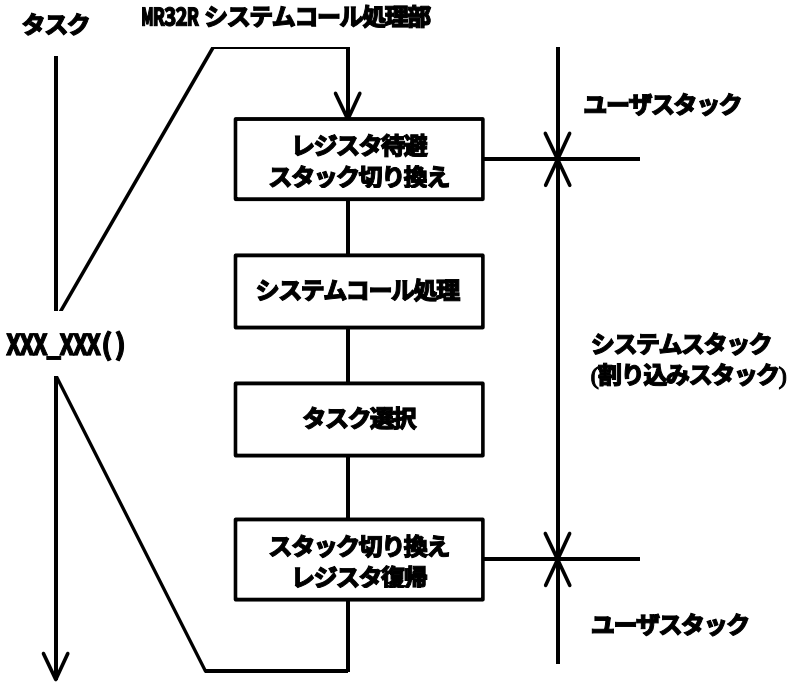

図 1.1 システムスタックとユーザスタック

システムスタックとユーザスタックの各セクションの配置は以下のようになります。

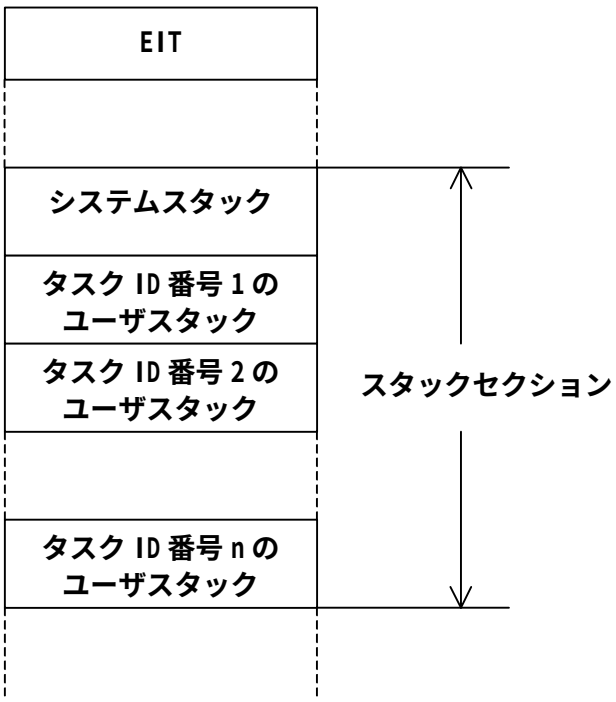

図 1.2　スタックの配置

1.3.1. ユーザスタックの算出方法

ユーザスタックは、各タスクごとに算出する必要があります。以下にアプリケーションを C 言語で記述した場合とアセンブリ言語で記述した場合のスタックの算出方法を以下に示します。

- C 言語でアプリケーションを記述した場合

C 言語でアプリケーションを記述した場合、下記に示す方法でユーザスタックサイズを求めることができます。

1. タスクで使用しているスタックサイズを算出します。
2. C 言語のインタフェースルーチンで使用しているスタックサイズを求めます。
3. **MR32R** が消費するスタックサイズ
   タスクからシステムコールを発行することによりそのシステムコールが消費するスタックサイズは、80 バイトに 2.の C 言語インタフェースルーチンで使用するスタックサイズを加えたものになります。また、タスクまたはハンドラの両方から発行できるシステムコールを発行している場合は、表 1.3に記載されたスタックサイズに 2.の言語インタフェースルーチンで使用するスタックサイズを加えたものになります。
   タスクから発行しているシステムコールの中で、消費するスタックサイズが最も大きいサイズが **MR32R** が消費するスタックサイズとなります。

上記 1、3 の合計サイズが、ユーザースタックサイズとなります。タスクで使用しているシステムコールがすべてタスクまたはハンドラの両方から発行できるシステムコールであってもタスクコンテキスト分(80 バイト)は消費します。

- アセンブリ言語でアプリケーションを記述した場合

1. ユーザプログラムで使用する部分
   タスクでレジスタをスタックに保存する場合に使用するサイズです。
2. システムコール発行により消費するスタックサイズ
   タスクからシステムコールを発行することによりそのシステムコールが消費するスタックサイズは、タスクからのみ発行可能なシステムコールの場合、80 バイトになります。また、タスクまたはハンドラの両方から発行できるシステムコールを発行している場合は、表 1.3に記載されたスタックサイズになります。
   タスクから発行しているシステムコールの中で、消費するスタックサイズが最も大きいサイズが **MR32R** が消費するスタックサイズとなります。

上記 1、2 の合計サイズが、ユーザースタックサイズとなります。
図 1.3にユーザスタックの算出例を示します。

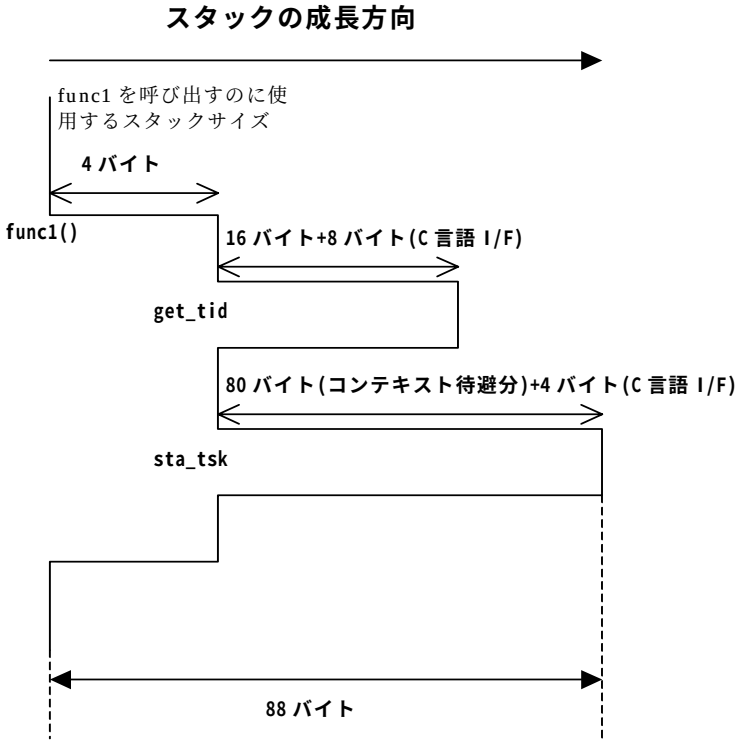

図 1.3 ユーザースタックサイズの算出例

1.3.2. システムスタックの算出方法

システムスタックを最も多く消費するのはシステムコール処理中[3]に割り込みが発生し、その上に多重割り込みが発生した場合です。すなわち、システムスタックの必要量（最大サイズ)は以下の計算式で算出することができます。[4]

$$\text{システムスタックの必要量} = \alpha + \sum \beta i (+\gamma)$$

- α

  使用するシステムコールの中で最大のシステムスタックサイズ[5]。

  例えば、sta_tsk、ext_tsk、cre_tsk を使用する場合、表 1.1で、それぞれのシステムスタックサイズを調べると、

システムコール名	システムスタックサイズ
cre_tsk	60 バイト
sta_tsk	0 バイト
ext_tsk	0 バイト
slp_tsk	0 バイト

  となるのでこの場合、使用するシステムコールの中で最大のシステムスタックサイズは cre_tsk の場合で 60 バイトです。

- βi

  割り込みハンドラの使用するスタックサイズ。詳細は後述します。

- γ

  システムクロック割り込みハンドラの使用するスタックサイズ。詳細は後述します。

[3] ユーザスタックからシステムスタックに切り替えた後

[4] TW32R・D-CC/M32R、CC32R ともに同様の方法で求めることができます。

[5] それぞれのシステムコールに使用するスタックサイズは、表 1.1から表 1.3を参照してください。

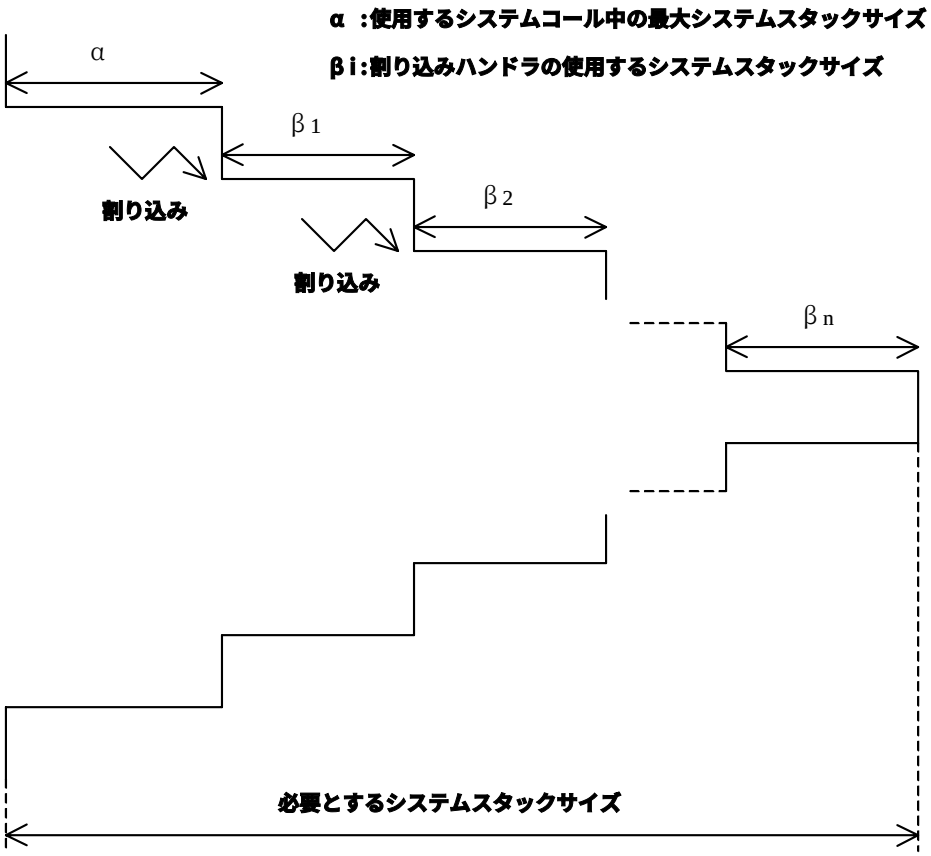

図 1.4　システムスタックサイズの算出方法

【 割り込みハンドラの使用するスタックサイズβi 】

システムコール中に発生した割り込みハンドラの使用するスタックサイズは以下の 3 つのサイズの合計となります。

- コンテキスト待避領域
- 割り込み処理から呼び出されるサブルーチンのスタック使用量の最大値
- ユーザスタックの記述する部分で使用するスタック使用量[6]

C 言語で記述したときも、アセンブリ言語で記述したときもスタック使用量は上記の方法で算出することができます。

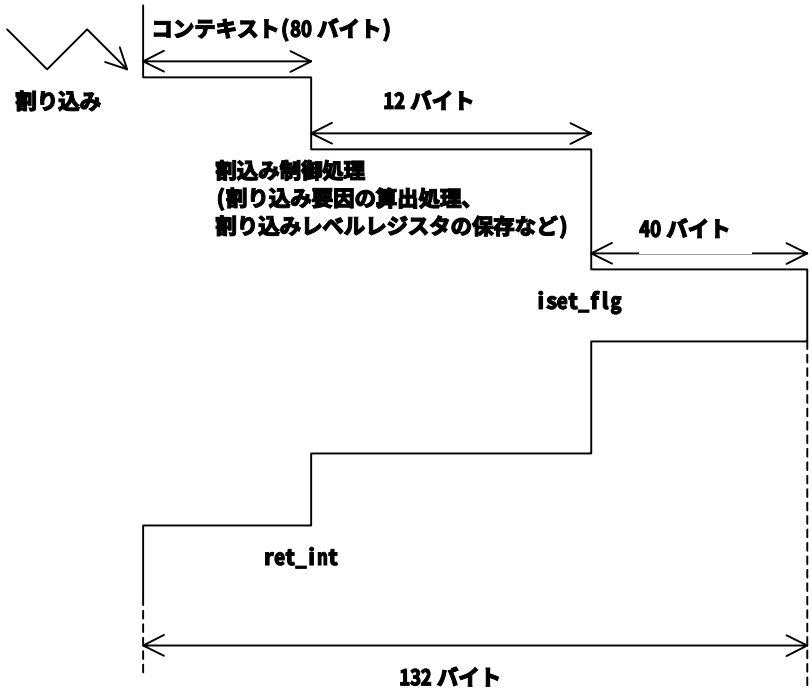

図 1.5 割り込みハンドラの使用するスタック量

[6] __RESTORE_IPL_from_STACK、__SAVE_IPL_to_STACK でレジスタを退避するのに使用するスタックサイズです。これらの割り込み制御プログラムについての詳細は MR32R ユーザーズマニュアル 割り込み制御プログラムの作成方法をご覧ください。

【 システムクロック割り込みハンドラが使用するシステムスタックサイズ γ 】

システムクロックを使用しないときは、システムクロック割り込みハンドラが使用するシステムスタックを加算する必要はありません。

システムクロック割り込みハンドラが使用するシステムスタック量 γ は以下に示すサイズです。

92+周期起動ハンドラ、アラームハンドラでスタック使用量の最も大きいサイズ

周期起動、アラームハンドラのどちらも使用しない場合は、

γ = 84 バイト

になります。(上記数値には、コンテキスト待避分を含みます。)

割り込みハンドラとシステムクロック割り込みハンドラを併用して使用する場合は、双方の使用するスタックサイズを加算してください。

第2章　システムコールリファレンス

2.1. タスク管理機能

2.1.1. cre_tsk (Create Task)

【システムコール名】

cre_tsk　→　タスクを生成します

【アセンブリ言語による呼び出し方法】

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

《引数》

tskid	[**]	生成するタスクの ID 番号
pk_ctsk	[****]	タスク生成情報を格納した先頭アドレス (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	生成するタスクの ID 番号
R2	--
R3	--

pk_ctsk の指す領域には、以下の情報を設定してください。

オフセット	サイズ		
+0	4	exinf	拡張情報
+4	4	tskatr	タスク属性
+8	4	task	タスク起動アドレス
+12	2	itskpri	タスク起動時優先度
+16	4	stksz	スタックサイズ
+20	4	stk	ユーザーが確保したスタックの先頭アドレス

【 C 言語による呼び出し方法】

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

《引数》

```
ID       tskid;          生成するタスク ID 番号
T_CTSK   *pk_ctsk;       タスク生成情報
```

pk_ctsk の指す構造体に以下に示す情報を設定してください。

```
typedef struct t_ctsk {
    VP      exinf;   /* 拡張情報 */
    ATR     tskatr;  /* タスク属性 */
    FP      task;    /* タスク起動アドレス */
    PRI     itskpri; /* タスク起動時優先度 */
    INT     stksz;   /* スタックサイズ */
    VP      stk;     /* ユーザーが確保したスタックの先頭アドレス */
} T_CTSK;
```

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000)	: 正常終了
E_NOMEM	0FFFFFFF6H(-H'0000000a)	: メモリ不足
E_OBJ	0FFFFFFC1H(-H'0000003f)	: オブジェクトの状態が不正

【機能説明】

tskid で示されたタスクを生成します。すなわち、指定したタスクを未登録(NON-EXISTENT)状態から休止(DORMANT)状態へ移行します。

生成するタスクの情報を設定した後、本システムコールを発行し、タスクを生成します。生成するタスクの情報 pk_ctsk について以下に示します。

●exinf(拡張情報)

生成するタスクに関する情報を格納するためにユーザが自由に利用できる領域です。exinf の内容に関しては、**MR32R** は関知しません。

●tskatr(タスク属性)

生成するタスクが使用するスタック領域を内蔵 RAM にするか外部 RAM にするかユーザーが確保するかを指定します。__MR_USER を指定した場合、stk にスタック領域の先頭アドレスを指定してください。

★ 内蔵 RAM を使用する場合
__MR_INT(0)を指定してください。

★ 外部 RAM を使用する場合
__MR_EXT(0x10000)を指定してください。

★ ユーザーが確保した領域をスタックとして使用する場合
__MR_USER(0x20000)を指定してください。

●task(タスク起動アドレス)

生成するタスクの開始アドレスを指定する領域です。
必ず指定してください。
C 言語でプログラムを記述する場合は生成するタスク(関数)をプロトタイプ宣言しなければなりません。

●itskpri(タスク起動時優先度)

生成するタスクが起動された時の優先度を指定する領域です。
必ず指定してください。

●stksz(スタックサイズ)

生成するタスクが使用するスタックサイズを指定する領域です。
必ず指定してください。

●stk(スタックの先頭アドレス)

ユーザーが確保したスタック領域の先頭アドレスを指定します。tskatr に __MR_INT,__MR_EXT が指定されていた場合は、OS カーネルがスタック領域を確保し、この値は無視します。

本システムコールは、指定したタスクが未登録(NON-EXISTENT)状態の時のみ有効で、その他の状態にあるタスクに対して、本システムコールを発行した場合は、エラー E_OBJ を返します。

生成するタスクの ID 番号は、ユーザ管理となります。本システムコールの指定可能な ID 番号の範囲は、1 からコンフィグレーションファイルの最大項目数定義で設定したユーザシステムの最大タスク数までです。

OS カーネルがスタック領域を割り当てる際、pk_ctsk の stksz で指定したサイズ分のメモリがない場合は、本システムコールを発行したタスクに対してエラーE_NOMEM を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
#define ID_task2   2
#define ID_task3   3
void task2(void);     /* プロトタイプ宣言 */
void task3(void);     /* プロトタイプ宣言 */
void task1(void)
{
    T_CTSK ctsk2;
    T_CTSK ctsk3 = {0,__MR_INT,task3,2,200};
    ctsk2.tskatr = __MR_EXT;    /* スタックに外部 RAM を指定 */
    ctsk2.task = task2;         /* タスク起動アドレス設定 */
    ctsk2.itskpri = 2;          /* タスク起動時の優先度設定 */
    ctsk2.stksz = 100;          /* タスクが使用するスタックサイズ設定 */
    cre_tsk( ID_task2, &ctsk2 );
        :
}
void task2(void)
{
      :
    ext_tsk();
}
void task3(void)
{
      :
    ext_tsk();
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
ID_task2:       .equ    2
ID_task3:       .equ    3
ctsk2:
    .DATA.W   0         ; 拡張情報
    .DATA.W   __MR_INT ; タスク属性
    .DATA.W   task2     ; タスク起動アドレス
    .DATA.H      2      ; タスク起動時優先度
    .RES.B    2
    .WORD    100        ; スタックサイズ

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .equ ID_task2,2
    .equ ID_task3,3
ctsk2:
    .LONG    0         ; 拡張情報
    .LONG    __MR_INT ; タスク属性
    .LONG    task2     ; タスク起動アドレス
    .SHORT   2         ; タスク起動時優先度
    .space   2
    .LONG    100       ; スタックサイズ

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

2.1.2. del_tsk(Delete Task)

【システムコール名】

del_tsk　→　タスクを削除します

【アセンブリ言語による呼び出し方法】

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

《引数》

tskid　　[**]　　削除するタスクの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	削除するタスクの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　　tskid;　　削除するタスク ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK        00000000H(-H'00000000) :  正常終了
E_OBJ       0FFFFFFC1H(-H'0000003f):  オブジェクトの状態が不正
E_NOEXS     0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
```

【機能説明】

tskid で示されるタスクを削除します。

このシステムコールは自タスクを指定することはできません。

本システムコールを発行することにより対象となるタスクの状態は、休止(DORMANT)状態から未登録(NON-EXISTENT)状態へ移行します。本システムコールにより、対象タスクが使用していたスタック領域は解放され、再利用可能な状態となります。

本システムコールは、対象となるタスクが休止状態の時のみ有効です。したがって、その他の状態[7]にあるタスクに対して本システムコールを発行した場合は、エラーE_OBJ を返します。

未登録状態のタスクに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は正常に動作しません。

[7] 未登録状態(NON-EXISTENT)を除く

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .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)

【システムコール名】

sta_tsk　→　タスクを起動します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
sta_tsk  tskid, stacd
```

《引数》

tskid	[**]	起動するタスクの ID 番号
stacd	[****]	タスク起動コード

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	起動するタスクの ID 番号
R2	タスク起動コード
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER sta_tsk (tskid, stacd);
```

《引数》

ID	tskid;	起動するタスク ID 番号
INT	stacd;	タスク起動コード

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない
E_OBJ	0FFFFFFC1H(-H'0000003f):	オブジェクトの状態が不正

【機能説明】

tskid で示されたタスクを起動します。なわち指定したタスクを休止(DORMANT)状態から実行可能(READY)状態もしくは、実行(RUN)状態へ移行します。

起動コード stacd は、32 ビットです。C 言語の場合、stacd は起動タスクに引数として渡されます。

本システムコールは、指定したタスクが休止(DORMANT)状態であるときのみ有効です。したがって、対象タスクが休止(DORMANT)状態にない[8]場合に発せられた要求に対しては、システムコール発行タスクにエラーE_OBJ を返します。

未登録状態のタスクに対して本システムコールを発行した場合は、エラーE_NOEXS を返します。

ter_tsk、ext_tsk などで終了したタスクを再起動した場合、タスクは以下の状態でスタートします[9]。

[8] 未登録状態を除く。
[9] すなわち、タスクは完全にリセット状態からスタートします。

●コンフィグレーションファイルあるいは、cre_tsk システムコールで設定した開始アドレスからスタートする。

●起床要求カウントは 0(ゼロ)クリアする。

●優先度はコンフィグレーションファイルあるいは、cre_tsk システムコールで設定したで初期優先度となる。

●PC、PSW と下記に示す以外のレジスタの初期値は不定です。

＊ M32R ファミリクロスツール **CC32R** をお使いの場合
R2、R4 レジスタに起動コードが設定されます。
R11、R12、R13 レジスタにベースアドレスが設定されます。
＊ M32R ファミリ GNU クロスツール **TW32R・D-CC/M32R** をお使いの場合
R0、R2 レジスタに起動コードが設定されます。

なお、タスクが再起動された場合でも、以前に定義した例外ハンドラは解除されません。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラからは発行する場合は、ista_tsk システムコールを使用してください。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.1.4. ista_tsk(Start Task)

【システムコール名】

ista_tsk　→　タスクを起動します(ハンドラ専用)

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
ista_tsk tskid, stacd
```

《引数》

tskid	[**]	起動するタスクの ID 番号
stacd	[****]	タスク起動コード

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	起動するタスクの ID 番号
R2	タスク起動コード
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER ista_tsk (tskid, stacd);
```

《引数》

ID	tskid;	起動するタスク ID 番号
INT	stacd;	タスク起動コード

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない
E_OBJ	0FFFFFFC1H(-H'0000003f):	オブジェクトの状態が不正

【機能説明】

sta_tsk システムコールと同じ機能を割込みハンドラ、周期起動ハンドラ、アラームハンドラから利用する場合に、このシステムコールを使用してください。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.1.5. ext_tsk(Exit Task)

【システムコール名】

ext_tsk　→　自タスクを終了します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
ext_tsk
```

《引数》

なし

《レジスタ設定》

本システムコールを発行したタスクには戻りません。

【 C 言語による呼び出し方法】

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

《引数》

なし

《戻り値》

本システムコールを発行したタスクには戻りません。

【エラーコード】

本システムコールを発行したタスクには戻りません。

【機能説明】

自タスクを終了します。すなわち、自タスクを実行(RUN)状態から休止(DORMANT)状態へ移行します。一度タスクが終了すると再度 sta_tsk、ista_tsk システムコールにより起動するまで動作することはありません。再度 sta_tsk、ista_tsk システムコールにより起動したタスクは、リセットされたように動作します。

本システムコールの発行では自タスクが以前に獲得していた資源 (セマフォなど)は解放しません。

なお、強制例外ハンドラから本システムコールを発行した場合は、強制例外ハンドラに対応したタスクも同時に正常終了します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラからは発行できません。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.1.6. exd_tsk(Exit and Delete Task)

【システムコール名】

exd_tsk　→　タスクを正常終了後、削除します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
exd_tsk
```

《引数》

なし

《レジスタ設定》

本システムコールを発行したタスクには戻りません。

【 C 言語による呼び出し方法】

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

《引数》

なし

《戻り値》

本システムコールを発行したタスクには戻りません。

【エラーコード】

本システムコールを発行したタスクには戻りません。

【機能説明】

自タスクを正常終了し、削除します。すなわち、自タスクを実行(RUN)状態から未登録(NON-EXISTENT)状態へ移行します。一度、タスクが削除されると、再び、cre_tsk、sta_tsk または ista_tsk システムコールを発行するまで動作しません。

本システムコール発行では、自タスクが以前獲得していた資源(セマフォ、メモリプールなど)は解放されません。ただし、自タスクが使用していたスタック領域は解放され、再利用可能となります。

本システムコールは、タスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は正常に動作しません。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .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)

【システムコール名】

ter_tsk　→　他タスクを強制的に終了します

【アセンブリ言語による呼び出し方法】

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

《引数》

tskid　[**]　強制終了するタスクの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	強制終了するタスクの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　tskid;　強制終了するタスク ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK        00000000H(-H’00000000) :  正常終了
E_NOEXS     0FFFFFFCCH(-H’00000034):  オブジェクトが存在していない
E_OBJ       0FFFFFFC1H(-H’0000003f):  オブジェクトの状態が不正
```

【機能説明】

tskid で示されたタスクを、強制的に終了させます。

このシステムコールで自タスクを指定することはできません。したがって、自タスクを終了する場合は ext_tsk システムコールを使用してください。指定したタスクが待ち状態に入り、何らかの待ち行列[10]につながれていた場合には、このシステムコールの実行によってその待ち行列から削除されます。しかし、指定したタスクがそれ以前に獲得したセマフォなどは解放されません。

tskid で示されたタスクが休止(DORMANT)状態にある場合は、システムコールの戻り値としてエラーE_OBJ を返します。

tskid で示されたタスクが未登録(NON-EXISTENT)状態にある場合は、システムコールの戻り値としてエラーE_NOEXS を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよび、アラームハンドラからは発行できません。

[10] タイムアウト待ち行列、イベントフラグ待ち行列など

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

    .global  task2
task2:
          :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

    .global  task2
task2:
          :
```

2.1.8. dis_dsp(Disable Dispatch)

【システムコール名】

dis_dsp　→　タスクのディスパッチを禁止します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
dis_dsp
```

《引数》

なし

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	--
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

なし

《戻り値》

関数の戻り値としてエラーコードを返す。

【エラーコード】

E_OK　　00000000H(-H’00000000): 正常終了

【機能説明】

タスクのディスパッチを禁止します。

本システムコール実行後、ena_dsp システムコールが実行されるまでの間はタスクのディスパッチは禁止状態となります。したがって、割り込みハンドラあるいは dis_dsp を実行したタスクから発行されたシステムコールによって、dis_dsp を実行したタスクより高い優先度を持つタスクが実行可能(READY)状態となっても、そのタスクにはディスパッチされません。すなわち、優先度の高いタスクへのディスパッチは、ディスパッチ禁止状態が終了するまで遅延されます。

ただし、外部割り込みは禁止しないので、ディスパッチ禁止状態であっても割り込みハンドラは起動されます。既に、ディスパッチ禁止状態にあるタスクが dis_dsp を発行した場合は、ディスパッチ禁止状態がそのまま継続するだけで、エラーにはなりません。ただし、dis_dsp を複数回、発行しても、その後、ena_dsp を 1 回発行するだけでディスパッチ禁止状態が解除されます。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラからは発行できません。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.1.9. ena_dsp(Enable Dispatch)

【システムコール名】

ena_dsp　→　タスクのディスパッチを許可します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
ena_dsp
```

《引数》

なし

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	--
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

なし

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK　　00000000H(-H’00000000): 正常終了

【機能説明】

タスクのディスパッチを許可します。

すなわち、dis_dsp によって設定されていたディスパッチ禁止状態を解除し、スケジューラを動作させます。ディスパッチ禁止状態でないタスクが ena_dsp を発行した場合は、ディスパッチを許可した状態がそのまま継続するだけで、エラーとはなりません。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラからは発行できません。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.1.10. chg_pri(Change Task Priority)

【システムコール名】

chg_pri　→　タスクの優先度を変更します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
chg_pri  tskid, tskpri
```

《引数》

tskid	[**]	優先度を変更するタスクの ID 番号
tskpri	[-*]	変更する優先度

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	優先度を変更するタスクの ID 番号
R2	変更する優先度
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER chg_pri (tskid,tskpri);
```

《引数》

ID	tskid;	優先度を変更するタスクの ID 番号
PRI	tskpri;	変更する優先度

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない
E_OBJ	0FFFFFFC1H(-H'0000003f):	オブジェクトの状態が不正

【機能説明】

tskid で示されたタスクの優先度を、tskpri で示される値に変更します。また、その変更結果に基づいて再スケジューリングを行います。したがって、対象タスクがレディキューや何らかの待ち行列につながれていた場合には、本システムコールによって行列の順番が変わることがあります。タスクの優先度は、数の小さい方が高く 1 が最高優先度です。優先度として指定できる数値は最小値が 1 です。また、優先度の最大値はコンフィグレーションファイルで指定した優先度の最大値であり、指定可能範囲は 1～255 です。例えば、コンフィグレーションファイルで

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

の場合は指定できる優先度の範囲は 1 から 13 までです。[11]

[11] 優先度の低いタスクへの切り替えは、処理時間、割り込み禁止時間が多く必要になります。したがって、優先度の範囲を狭くすることを推奨します。

tskid=TSK_SELF=0 を指定すると自タスクの指定になります。ハンドラから TSK_SELF は指定できません。

本システムコールで休止(DORMANT)状態または未登録(NON-EXISTENT)状態にあるタスクの優先度を変更することはできません。したがって、tskid で示された対象タスクが休止(DORMANT)状態にある場合は、システムコールの戻り値としてエラーE_OBJ を返し、未登録状態にある場合は、エラーE_NOEXS を返します。レディキューにつながれているタスク（実行状態のタスクを含む)、または優先度順の待ち行列の中のタスクに対して本システムコールが実行された場合、対象タスクはキューの該当優先度の部分の最後尾に移動します。以前と同じ優先度を指定した場合も、同様に、そのキューの最後尾に移動します[12]。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行する場合は ichg_pri システムコールを使用してください。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
          :
    chg_pri( ID_task2, 2 );
          :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

[12] したがって、自タスクを対象として現在と同じ優先度で本システムコールを発行することにより、実行権の放棄を行なうことができます。

2.1.11. ichg_pri(Change Task Priority)

【システムコール名】

ichg_pri　→　タスクの優先度を変更します（ハンドラ専用）

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
ichg_pri tskid, tskpri
```

《引数》

tskid	[**]	優先度を変更するタスクの ID 番号
tskpri	[-*]	変更する優先度

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	優先度を変更するタスクの ID 番号
R2	変更する優先度
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER ichg_pri (tskid,tskpri);
```

《引数》

ID	tskid;	優先度を変更するタスクの ID 番号
PRI	tskpri;	変更する優先度

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない
E_OBJ	0FFFFFFC1H(-H'0000003f):	オブジェクトの状態が不正

【機能説明】

chg_pri システムコールと同じ機能を割込みハンドラ、周期起動ハンドラ、アラームハンドラから利用する場合に、このシステムコールを使用します。本システムコールでは、tskid=TSK_SELF=0 による自タスク指定はできません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void inthand()
{
            :
    ichg_pri( ID_main, 2 );
            :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.1.12. rot_rdq(Rotate Ready Queue)

【システムコール名】

rot_rdq　→　タスクのレディキューを回転します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
rot_rdq  tskpri
```

《引数》

tskpri　[-*]　回転するレディキューの優先度

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	--
R2	変更する優先度
R3	--

【 C 言語による呼び出し方法】

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

《引数》

PRI　tskpri;　回転するレディキューの優先度

《戻り値》

関数の戻り値は常に E_OK を返します。

【エラーコード】

E_OK　00000000H(-H’00000000): 正常終了

【機能説明】

tskpri で示された優先度のレディキューを回転します。すなわち、その優先度のレディキューの先頭につながれているタスクをレディキューの最後尾につなぎかえ、同一優先度のタスクの実行を切り替えます。この様子を図 2.1に示します。

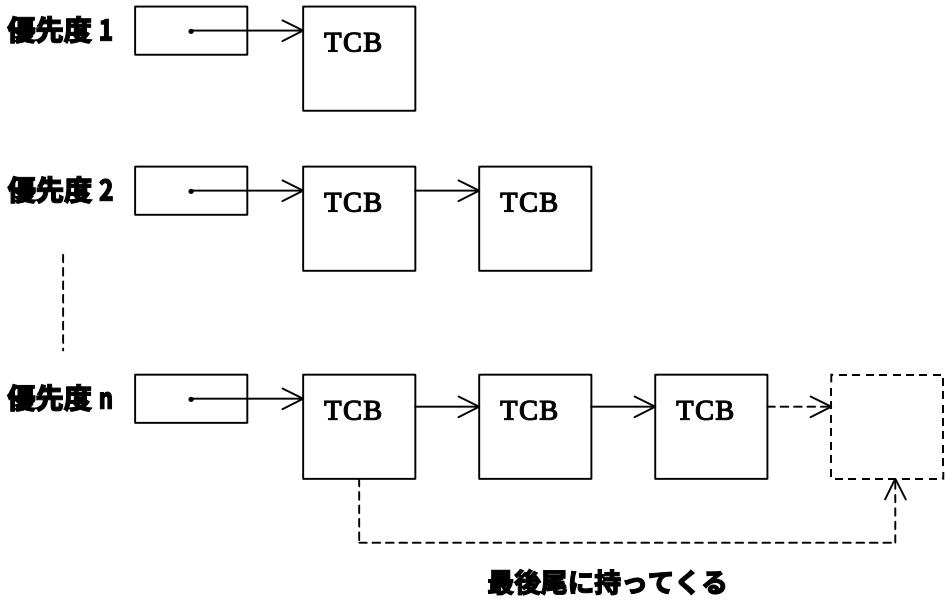

図 2.1　rot_rdq システムコールによるレディキューの操作

このシステムコールを一定時間間隔で発行することにより、ラウンドロビンスケジューリングをおこなうことができます。tskpri=TPRI_RUN=0 の指定により、自タスクの持つ優先度のレディキューを回転させます。

また、本システムコールで自タスクの優先度を指定した場合には、自タスクがそのレディキューの最後尾にまわることになります。なお、指定した優先度のレディキューにタスクがない場合は何も行いません。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行する場合は irot_rdq システムコールを使用してください。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
          :
    rot_rdq( 2 );
          :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.1.13. irot_rdq(Rotate Ready Queue)

【システムコール名】

irot_rdq　→　タスクのレディキューを回転します (ハンドラ専用)

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
irot_rdq tskpri
```

《引数》

tskpri　[-*]　回転するレディキューの優先度

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	--
R2	変更する優先度
R3	--

【 C 言語による呼び出し方法】

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

《引数》

PRI　tskpri;　優先度

《戻り値》

関数の戻り値は常に E_OK を返します。

【エラーコード】

E_OK　00000000H(-H’00000000): 正常終了

【機能説明】

rot_rdq システムコールと同じ機能を割り込みハンドラ、周期起動ハンドラ、アラームハンドラから利用する場合に、このシステムコールを使用します。irot_rdq(tskpri=TPRI_RUN)を発行した場合、割り込みハンドラが発生した時に実行していたタスクが持つ優先度のレディキューが回転します。

周期起動ハンドラから本システムコールを発行することによりラウンドロビンスケジューリングができます。

【使用例】

この例では周期起動ハンドラにより周期的に優先度 2 のレディキューを回転することによりラウンドロビンスケジューリングを実現しています。

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.1.14. rel_wai(Release Task Wait)

【システムコール名】

rel_wai　→　タスクの待ち状態を強制解除します

【アセンブリ言語による呼び出し方法】

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

《引数》

tskid　[**]　待ち状態を強制解除するタスクの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	待ち状態を強制解除するタスクの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　tskid;　待ち状態を強制解除するタスクの ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK      00000000H(-H'00000000) :  正常終了
E_NOEXS   0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
E_OBJ     0FFFFFFC1H(-H'0000003f):  オブジェクトの状態が不正
```

【機能説明】

tskid で示されたタスクの待ち状態(強制待ち(SUSPEND)状態を除く)を、強制的に解除し、実行可能(READY)状態あるいは実行(RUN)状態に移行します。強制的に解除されたタスクにはエラーコード E_RLWAI を返します。対象タスクが何らかの待ち行列[13]につながれていた場合には、本システムコールの実行によってその待ち行列から削除されます。

対象タスクが待ち状態にない場合は、システムコール発行タスクにエラーE_OBJ を返します。また、対象タスクが未登録状態(NON-EXISTENT)の場合に、本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールは、自タスクを指定できません。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行する場合は、irel_wai システムコールを使用してください。

[13] タイムアウト待ち行列、イベントフラグ待ち行列など。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
            :
    if( rel_wai( ID_main ) != E_OK )
        error(“Can’t rel_wai main()¥n”);
          :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.1.15. irel_wai(Release Task Wait)

【システムコール名】

irel_wai　→　タスクの待ち状態を強制解除します(ハンドラ専用)

【アセンブリ言語による呼び出し方法】

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

《引数》

tskid　[**]　待ち状態を強制解除するタスクの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	待ち状態を強制解除するタスクの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　tskid;　タスク ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK        00000000H(-H'00000000) :  正常終了
E_NOEXS     0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
E_OBJ       0FFFFFFC1H(-H'0000003f):  オブジェクトの状態が不正
```

【機能説明】

rel_wai システムコールの機能を割込みハンドラ、周期起動ハンドラ、アラームハンドラから利用する場合に、このシステムコールを使用します。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void inthand()
{
          :
    if( irel_wai( ID_main ) != E_OK )
        error(“Can’t irel_wai task(2)¥n”);
          :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.1.16. get_tid(Get Self Task ID)

【システムコール名】

get_tid　→　自タスクの ID を得ます

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
get_tid
```

《引数》

なし

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	自タスクの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　　*p_tskid;　　タスク ID 番号を格納する領域の先頭アドレス

《戻り値》

関数の戻り値は常に E_OK を返します。
p_tskid の示す領域に、自タスクの ID 番号が設定されます。

【エラーコード】

E_OK　　00000000H(-H'00000000):　正常終了

【機能説明】

タスクから本システムコールを発行した場合、自タスクの ID 番号を獲得します。また、割り込みハンドラ、周期起動ハンドラ、アラームハンドラから本システムコールを発行した場合は、FALSE=0(ゼロ)を返します。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
    ID tskid;
       :
    get_tid(&tskid);
       :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.1.17. ref_tsk(Refer Task Status)

【システムコール名】

ref_tsk　→　タスクの状態を参照します

【アセンブリ言語による呼び出し方法】

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

《引数》

taskid	[**]	状態を参照するタスクの ID 番号
pk_rtsk	[****]	タスクの状態を返すパケットアドレス (R2 レジスタに設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	状態を参照するタスクの ID 番号
R2	タスクの状態を返すパケットの先頭アドレス
R3	--

pk_rtsk の指す領域には、以下の情報が返されます。

オフセット	サイズ		
+0	4	exinf	拡張情報
+4	2	tskpri	現在の優先度
+8	4(U)	tskstat	タスクの状態
+12	4(U)	tskwait	タスクの待ち要因
+16	2	wid	タスクの待ちオブジェクト ID
+20	4	wupcnt	起床要求キューイング数
+24	4	tskatr	タスク属性
+28	4	task	タスク起動アドレス
+32	2	itskpri	タスク起動時優先度
+36	4	stksz	スタックサイズ
+40	4(U)	epndptn	例外ペンディングパターン

U は、unsigned データです。

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER ref_tsk (pk_rtsk,tskid);
```

《引数》

ID	tskid;	状態を参照するタスクの ID 番号
T_RTSK	*pk_rtsk;	タスク状態を返す構造体の先頭アドレス

《戻り値》

関数の戻り値にエラーコードを返します 。

pk_rtsk の指す構造体には、以下の情報が返されます。

VP	exinf	拡張情報
PR	tskpri	現在の優先度
UH	tskstat	タスク状態
UINT	tskwait	タスクの待ち要因

ID	wid	タスクの待ちオブジェクト ID
INT	wupcnt	起床要求キューイング数
ATR	tskatr	タスク属性
FP	task	タスク起動アドレス
PRI	itskpri	タスク起動時優先度
INT	stksz	スタックサイズ
UW	epndptn	例外ペンディングパターン

【エラーコード】

E_OK	00000000H(-H'00000000) : 正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034): オブジェクトが存在していない

【機能説明】

tskid で示されたタスクの状態を参照し、そのタスクの現在の情報をリターン値として返します。

●exinf

exinf には、指定したタスクの拡張情報を返します。

●tskpri

tskpri には、指定したタスクの優先度値を返します。

●tskstat

tskstat には指定したタスクの状態によって次の値が返されます。

TTS_RUN	(00000001H)	実行(RUN)状態
TTS_RDY	(00000002H)	実行可能(READY)状態
TTS_WAI	(00000004H)	待ち(WAIT)状態
TTS_SUS	(00000008H)	強制待ち(SUSPEND)状態
TTS_WAS	(0000000CH)	二重待ち(WAIT-SUSPEND)状態
TTS_DMT	(00000010H)	休止(DORMANT)状態

●tskwait

tskwait には、対象タスクが待ち状態であるならば待ち要因が返されます。各待ち要因の値を以下に示します。

TTW_SLP	(00000001H)	slp_tsk、tslp_tsk による待ち
TTW_DLY	(00000002H)	dly_tsk による待ち
TTW_FLG	(00000010H)	wai_flg、twai_flg による待ち
TTW_SEM	(00000020H)	wai_sem、twai_sem による待ち
TTW_MBX	(00000040H)	rcv_msg、trcv_msg による待ち
TTW_SMBF	(00000080H)	snd_mbf、tsnd_mbf による待ち
TTW_MBF	(00000100H)	rcv_mbf、trcv_mbf による待ち
TTW_CAL	(00000200H)	cal_pol、tcal_pol による待ち
TTW_ACP	(00000400H)	acp_pol、tacp_pol による待ち
TTW_RDV	(00000800H)	ランデブ待ち
TTW_MPL	(00001000H)	get_blk、tget_blk による待ち
TTW_MPF	(00002000H)	get_blf、tget_blf による待ち
TTW_VMBX	(00004000H)	vrcv_mbx、vtrcv_mbx による待ち

●wid

wid には、対象タスクが待ち状態となった場合、そのオブジェクトの ID 番号を返します。

●wupcnt

対象タスクの起床要求キューイング数を返します。

●tskatr

対象タスクの属性を返す。対象タスクが使用するスタックの RAM が内蔵 RAM(__MR_INT=0)であるか、外部 RAM(__MR_EXT=0x10000)であるか、あるいはユーザーがスタック領域を確保するか(__MR_USER=0x20000)を表わします。

●task

対象タスクの起動アドレスを返します。

●itskpri

対象タスクの起動時優先度を返します。

●stksz

対象タスクのスタックサイズを返します。

●epndptn

例外ペンディングパターン epndptn は、例外のマスク値と例外ペンディング情報を示し、以下の値を返します。[14]

epndptn		意味
EXM_SET	00000001H	例外マスクがセットされている
EXP_TER	00000002H	強制終了要求ペンディング中
EXP_FEX	00000004H	強制例外要求ペンディング中

タスクから tskid=TSK_SELF=0 によって、自タスクの指定はできますが、ハンドラから tskid=TSK_SELF の指定はできません。

ハンドラから、割り込まれたタスクを対象とした ref_tsk を発行した場合は、tskstat には、RUN 状態(TTS_RUN)が返されます。未登録状態のタスクに対して本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスク、ハンドラのどちらからでも発行できます。

[14]なお、コンフィグレーションファイルのシステム定義において例外ハンドラの設定を行っていない場合は、不定値が返されます。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
    T_RTSK rtsk;
          :
    ref_tsk( &rtsk, ID_main );
          :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.2. タスク付属同期機能

2.2.1. sus_tsk(Suspend Task)

【システムコール名】

sus_tsk　→　タスクを強制待ち状態へ移行します

【アセンブリ言語による呼び出し方法】

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

《引数》

tskid　[**]　強制待ちにするタスクの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	強制待ちにするタスクの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　tskid;　強制待ちするタスクの ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_QOVR	0FFFFFFB7H(-H'00000049):	キューイングまたはネストのオーバーフロー
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない
E_OBJ	0FFFFFFC1H(-H'0000003f):	オブジェクトの状態が不正

【機能説明】

tskid で示されたタスクの実行を中断させ、強制待ち(SUSPEND)状態へ移行します。強制待ち状態は、rsm_tsk システムコールの発行によって解除されます。tskid で示された対象タスクが休止(DORMANT)状態にある場合は、システムコールの戻り値としてエラーE_OBJ を返し、未登録状態(NON-EXISTENT)にある場合は、エラーE_NOEXS を返します。

本システムコールによる強制待ち要求のネストは行いません。従って、tskid で示された対象タスクが強制待ち(SUSPEND)状態にある場合は、システムコールの戻り値としてエラーE_QOVR を返します。

本システムコールで自タスクを指定することはできません。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行する場合は isus_tsk システムコールを使用してください。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
                :
    if( sus_tsk( ID_main ) != E_OK )
        printf(“Can’t suspend task main()¥n”);
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.2.2. isus_tsk(Suspend Task)

【システムコール名】

isus_tsk　→　タスクを強制待ち状態へ移行します（ハンドラ専用）

【アセンブリ言語による呼び出し方法】

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

《引数》

tskid　　[**]　　強制待ちにするタスクの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	強制待ちにするタスクの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　　tskid;　　強制待ちするタスクの ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK        00000000H(-H'00000000) :  正常終了
E_QOVR      0FFFFFFB7H(-H'00000049):  キューイングまたはネストのオーバーフロー
E_NOEXS     0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
E_OBJ       0FFFFFFC1H(-H'0000003f):  オブジェクトの状態が不正
```

【機能説明】

sus_tsk システムコールと同じ機能を割り込みハンドラ、周期起動ハンドラ、アラームハンドラから発行する場合にこのシステムコールを使用します。なお、本システムコールはハンドラからシステムコールを発行するので全てのタスク ID を指定することができます。
すなわち、被割り込みタスクもサスペンドすることができます。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void inthand()
{
                :
    if( isus_tsk( ID_main ) != E_OK )
        printf(“Can’t suspend main()¥n”);
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.2.3. rsm_tsk(Resume Task)

【システムコール名】

rsm_tsk　→　強制待ち状態のタスクを再開します

【アセンブリ言語による呼び出し方法】

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

《引数》

tskid　[**]　強制待ちを解除するタスクの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	強制待ちを解除するタスクの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　tskid;　タスク ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK        00000000H(-H'00000000) :  正常終了
E_NOEXS     0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
E_OBJ       0FFFFFFC1H(-H'0000003f):  オブジェクトの状態が不正
```

【機能説明】

tskid で示されたタスクが sus_tsk システムコールによって中断されている場合、対象タスクの強制待ち状態を解除し、実行を再開します。このとき、対象タスクはレディキューの最後尾につながれます。対象タスクが強制待ち(SUSPEND)状態にない場合(休止(DORMANT)状態を含む)に発せられた要求に対してはシステムコール発行タスクにエラーE_OBJ を返します。

なお、対象タスクが未登録状態の場合に、本システムコールを発行した場合は、エラー E_NOEXS を返します。

本システムコールは強制待ち(SUSPEND)状態または二重待ち(WAIT-SUSPEND)状態のタスクを対象としているので、自タスクを指定することはできません。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行する場合は irsm_tsk システムコールを使用してください。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
                :
    if( rsm_tsk( ID_main ) != E_OK )
        printf(“Can’t resume main()¥n”);
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.2.4. irsm_tsk(Resume Task)

【システムコール名】

irsm_tsk　→　強制待ち状態のタスクを再開します (ハンドラ専用)

【アセンブリ言語による呼び出し方法】

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

《引数》

tskid　[**]　強制待ちを解除するタスクの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	強制待ちを解除するタスクの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　tskid;　タスク ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK        00000000H(-H’00000000) :  正常終了
E_NOEXS     0FFFFFFCCH(-H’00000034):  オブジェクトが存在していない
E_OBJ       0FFFFFFC1H(-H’0000003f):  オブジェクトの状態が不正
```

【機能説明】

rsm_tsk システムコールと同じ機能を割込みハンドラ、周期起動ハンドラ、アラームハンドラから利用する場合にこのシステムコールを使用してください。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void inthand()
{
        :
    irsm_tsk( ID_main );
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.2.5. slp_tsk(Sleep Task)

【システムコール名】

slp_tsk　→　タスクを待ち状態へ移行します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
slp_tsk
```

《引数》

なし

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	--
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

なし

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H’00000000) :	正常終了
E_RLWAI	0FFFFFFAAH(-H’00000056):	待ち状態強制解除

【機能説明】

自タスクを実行(RUN)状態から起床待ち状態へ移行します。本システムコールによる待ち状態は、このタスクを対象として発行されたタスク起床のシステムコール[15]または待ち状態の強制解除のシステムコール[16]により解除されます。前者の場合はエラーコードとして E_OK が、後者の場合はエラーコードとして E_RLWAI が返されます。

本システムコールにより待ち(WAIT)状態となっているときに他のタスクから sus_tsk されるとそのタスクの状態は二重待ち(WAIT-SUSPEND)状態になります。この場合はタスク起床のシステムコールにより待ち状態が解除されても、まだ強制待ち状態であり、rsm_tsk の発行まで、タスクの実行は再開されません。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は正常に動作しません。

[15] wup_tsk、iwup_tsk システムコール
[16] rel_wai、irel_wai システムコール

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.2.6. tslp_tsk(Sleep Task with Timeout)

【システムコール名】

tslp_tsk　→　タスクを一定時間待ち状態へ移行します

【アセンブリ言語による呼び出し方法】

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

《引数》

tmout　　[****]　　タイムアウト値

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	--
R2	--
R3	--
R4	タイムアウト値

【 C 言語による呼び出し方法】

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

《引数》

TMO　　tmout;　　タイムアウト値

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK        00000000H(-H'00000000) :  正常終了
E_TMOUT     0FFFFFFABH(-H'00000055):  ポーリング失敗またはタイムアウト
E_RLWAI     0FFFFFFAAH(-H'00000056):  待ち状態強制解除
```

【機能説明】

自タスクを、指定した一定時間だけ実行(RUN)状態から待ち(WAIT)状態へ移行します。
本システムコール実行による待ち状態は、以下に示す場合に解除されます。

- 他タスクおよび割り込みからタスク起床のシステムコール[17]を発行した場合

  この時のエラーコードは、E_OK が返ります。

- 他タスクおよび割り込みから待ち状態強制解除のシステムコール[18]を発行した場合

  この時のエラーコードは、E_RLWAI が返ります。

- tmout で指定した時間が経過した場合

  この時のエラーコードは、E_TMOUT が返ります。

tmout で指定する時間の単位は、コンフィグレーションファイルで指定したシステムクロ

[17] wup_tsk、iwup_tsk システムコール
[18] rel_wai、irel_wai システムコール

ックの単位時間になります。コンフィグレーションファイルで、システムクロックの単位時間を 10ms に設定し、

tslp_tsk(10);

と記述すれば、100ms 間、自タスクが実行(RUN)状態から待ち(WAIT)状態へ移行します。

tmout には、-1 から 7FFFFFFFH まで指定できます。なお、tmout=TMO_FEVR(-1)にした場合は、永久待ちの指定で、slp_tsk システムコールと同じ動作になります。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラからは発行した場合は正常に動作しません。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.2.7. wup_tsk(Wakeup Task)

【システムコール名】

wup_tsk　→　待ち状態のタスクを起床します

【アセンブリ言語による呼び出し方法】

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

《引数》

tskid　　[**]　　起床するタスクの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	起床するタスクの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　　tskid;　　タスク ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK      00000000H(-H'00000000) :  正常終了
E_QOVR    0FFFFFFB7H(-H'00000049):  キューイングまたはネストのオーバーフロー
E_NOEXS   0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
E_OBJ     0FFFFFFC1H(-H'0000003f):  オブジェクトの状態が不正
```

【機能説明】

tskid で指定したタスクが slp_tsk あるいは tslp_tsk の実行による待ち(WAIT)状態であれば、待ちを解除して実行可能(READY)状態もしくは実行(RUN)状態に移行します。また、tskid で指定したタスクが二重待ち(WAIT-SUSPEND)状態である時は、待ちのみを解除して強制待ち(SUSPEND)状態に移行します。

対象タスクが休止(DORMANT)状態にある場合に発せられた要求に対しては、システムコール発行タスクにエラーE_OBJ を返し、対象タスクが未登録状態の場合は、エラーE_NOEXS を返します。

本システムコールは自タスクを指定することはできません。

slp_tsk あるいは tslp_tsk システムコール実行による待ち(WAIT)状態もしくは二重待ち(WAIT-SUSPEND)状態にないタスクに対して本システムコールを行なった場合は、起床要求が蓄積されます。すなわち、起床要求対象タスクの TCB[19]内にある起床要求カウントを１つ増やすことにより起床要求を蓄積します。[20]

---

[19] タスク制御ブロック

[20] この起床要求カウントには、wup_tsk、iwup_tsk システムコールにより起床しようとした時に対象タスクが待ち

起床要求カウントの最大値は 0x7FFFFFFF です。起床要求カウントが 0x7FFFFFFF の時に、これを越えて起床要求を発生させると起床要求カウントは 0x7FFFFFFF のままで本システムコールの発行タスクには、エラーコード E_QOVR を返します。

本システムコールはタスクからのみ発行することができます。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行する場合は iwup_tsk システムコールを使用してください。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
                :
    if( wup_tsk( ID_main ) != E_OK )
        printf(“Can’t wakeup main()¥n”);
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

(WAIT)状態または二重待ち(WAIT-SUSPEND)状態でないために起床要求を実現できなかった回数が記憶されます。起床要求カウントが 1 以上である時に、そのタスクが slp_tsk あるいは tslp_tsk システムコールにより待ち状態に入ろうとした場合はその起床要求カウントを 1 つ減らします。この時、そのタスクは待ち状態にはなりません。slp_tsk あるいは tslp_tsk システムコールにより待ち (WAIT) 状態に入るのは起床要求カウントが 0 の時のみです。

2.2.8. iwup_tsk(Wakeup Task)

【システムコール名】

iwup_tsk　→　待ち状態のタスクを起床します（ハンドラ専用）

【アセンブリ言語による呼び出し方法】

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

《引数》

tskid　　[**]　　起床するタスクの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	起床するタスクの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　　tskid;　　タスク ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK        00000000H(-H'00000000):  正常終了
E_QOVR      0FFFFFFB7H(-H'00000049):  キューイングまたはネストのオーバーフロー
E_NOEXS     0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
E_OBJ       0FFFFFFC1H(-H'0000003f):  オブジェクトの状態が不正
```

【機能説明】

wup_tsk システムコールと同じ機能を割込みハンドラ、周期起動ハンドラ、アラームハンドラから利用する場合にこのシステムコールを使用します。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void inthand()
{
    if( iwup_tsk( ID_main ) != E_OK )
        printf(“Can’t wakeup main()¥n”);
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.2.9. can_wup(Cancel Wakeup Task)

【システムコール名】

can_wup　→　タスクの起床要求を無効にします

【アセンブリ言語による呼び出し方法】

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

《引数》

tskid　[**]　起床要求を無効にするタスクの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	起床要求を無効にするタスクの ID 番号
R2	無効になった起床要求回数
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER can_wup (p_wupcnt,tskid);
```

《引数》

```
INT   *p_wupcnt;   無効になった起床要求回数を格納する領域の先頭アドレス
ID    tskid;       タスク ID 番号
```

《戻り値》

関数の戻り値としてエラーコードを返します。
変数 wupcnt に無効になった起床要求回数が設定されます。

【エラーコード】

```
E_OK          00000000H(-H'00000000) :  正常終了
E_NOEXS       0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
E_OBJ         0FFFFFFC1H(-H'0000003f):  オブジェクトの状態が不正
```

【機能説明】

tskid で示された対象タスクの起床要求カウントを 0(ゼロ)クリアします。すなわち、本システムコール発行以前に wup_tsk、iwup_tsk システムコールにより起床しようとした時に対象タスクが待ち(WAIT)状態もしくは二重待ち(WAIT-SUSPEND)状態でないために起床要求のみが蓄積されていたのをすべて無効にします。

また、本システムコールの戻り値として 0(ゼロ)クリアする前の起床要求カウント、すなわち無効になった起床要求回数(wupcnt)が返されます。対象タスクが休止(DORMANT)状態に発せられた要求に対しては、システムコール発行タスクにエラーE_OBJ を返し、未登録(NON-EXISTENT)状態の場合は、エラーE_NOEXS を返します。本システムコールは、タスクから発行する場合に限り、tskid=TSK_SELF=0 で自タスクの指定ができます。

本システムコールはタスク、ハンドラのどちらからでも発行できます。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
    INT wupcnt;
                :
    if( can_wup(&wupcnt,ID_main) != E_OK )
        printf(“Can’t cancle wakeup main() ¥n”);
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global   task
task:
        :
    can_wup    ID_task2
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .include “mr32r.inc”
    .global   task
task:
        :
    can_wup    ID_task2
        :
```

2.3. 同期・通信機能(1 ワードイベントフラグ)

2.3.1. cre_flg(Create EventFlag)

【システムコール名】

cre_flg　→　イベントフラグを生成します

【アセンブリ言語による呼び出し方法】

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

《引数》

flgid	[**]	生成するイベントフラグの ID 番号
pk_cflg	[****]	イベントフラグ生成情報を格納した先頭アドレス (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	生成するイベントフラグの ID 番号
R2	イベントフラグ生成情報を格納した先頭アドレス
R3	--

pk_cflg の指す領域には、以下に示す情報を設定してください。

オフセット	サイズ		
+0	4	exinf	拡張情報
+4	4	flgatr	イベントフラグ属性
+8	4(U)	iflgptn	イベントフラグの初期値

U は、unsigned のデータを表わします。

【 C 言語による呼び出し方法】

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

《引数》

ID	flgid;	生成するイベントフラグ ID 番号
T_CFLG	*pk_cflg;	イベントフラグ生成情報

pk_cflg の指す構造体に以下に示す情報を設定してください。

```
typedef struct t_cflg {
    VP      exinf;          /* 拡張情報 */
    ATR     flgatr;         /* イベントフラグ属性 */
    UINT    iflgptn;        /* イベントフラグの初期値 */
} T_CFLG;
```

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) : 正常終了
E_OBJ	0FFFFFFC1H(-H'0000003f): オブジェクトの状態が不正

【機能説明】

flgid で示されたイベントフラグを生成します。

生成されたイベントフラグは、32 ビットで構成され、iflgptn の値で初期化されます。生成するイベントフラグの情報 pk_cflg について以下に示します。

- exinf(拡張情報)

  生成するイベントフラグに関する情報を格納するためのユーザが自由に利用できる領域です。exinf の内容に関しては、**MR32R** は関知しません。

- flgatr(イベントフラグ属性)

  ここで設定した値は、**MR32R** は関知しません。

- iflgptn

  イベントフラグ生成時の対象イベントフラグのビットパターンをこの領域にセットします。

すでに存在するイベントフラグに対して、本システムコールを発行した場合は、エラー E_OBJ を返します。

本システムコールの指定可能な ID 番号の範囲は、1 から最大項目数定義で設定したユーザシステムの最大イベントフラグ数までです。

本システムコールはタスクからのみ発行することができます。割り込みハンドラ、周期起動ハンドラおよび、アラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#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;    /* イベントフラグの初期ビットパターン設定 */
    cre_flg( ID_flg1, &cflg1 );
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
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
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
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)

【システムコール名】

del_flg　→　イベントフラグを削除します

【アセンブリ言語による呼び出し方法】

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

《引数》

flgid　　[**]　　削除するイベントフラグの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	削除するイベントフラグの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　　flgid;　　削除するイベントフラグ ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK          00000000H(-H’00000000) :  正常終了
E_NOEXS       0FFFFFFCCH(-H’00000034):  オブジェクトが存在していない
```

【機能説明】

flgid で示されたイベントフラグを削除します。

削除されたイベントフラグは、再度、同じ ID 番号で新しいイベントフラグを生成できます。削除するイベントフラグに対して、条件成立を待っているタスクが存在しても、本システムコールは正常終了します。その時、待ち状態にあったタスクは、イベントフラグ待ち状態が解除され、エラーE_DLT を返し、実行可能(READY)状態へ移行します。

存在しないイベントフラグに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。なお、本システムコールの発行は、cre_flg システムコールによって生成したイベントフラグに対して行ってください。コンフィグレーションファイルで定義したイベントフラグに対して本システムコールを発行した場合は、正常に動作しません。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.3.3. set_flg(Set EventFlag)

【システムコール名】

set_flg　→　イベントフラグをセットします

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
set_flg  flgid, setptn
```

《引数》

flgid	[**]	セットするイベントフラグの ID 番号
setptn	[****]	セットするビットパターン

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	セットするイベントフラグの ID 番号
R2	セットするビットパターン
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER set_flg (flgid, setptn);
```

《引数》

ID	flgid;	イベントフラグ ID 番号
UINT	setptn;	セットするビットパターン

《戻り値》

関数の戻り値として常に E_OK を返します。

【エラーコード】

E_OK	00000000H(-H’00000000) : 正常終了
E_NOEXS	0FFFFFFCCH(-H’00000034): オブジェクトが存在していない

【機能説明】

flgid で示される 32 ビットのイベントフラグのうち、setptn で示されているビットをセットします。つまり、flgid で示されるイベントフラグの値に対して setptn の論理和(OR)をとります。イベントフラグ値の変更の結果、wai_flg、twai_flg システムコールによってそのイベントフラグを待っていたタスクの待ち解除の条件を満たすようになれば、そのタスクの待ちを解除して実行可能(READY)状態もしくは実行(RUN)状態へ移行します[21]。

イベントフラグは、同一フラグに対する複数タスクの待ちが可能ですので、一回の set_flg システムコール発行で、その複数タスクが同時に待ち解除となります。ただし、待ち行列のなかのタスクがクリア指定でイベントフラグがセットされるのを待っていた場合はそのタスクまでが待ち解除になります。

setptn の全ビットを 0 とした場合は、対象イベントフラグに対して何の操作も行いませんが、エラーとはなりません。なお、存在しないイベントフラグに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハン

[21] 実行可能 (READY) 状態に移すか実行(RUN)状態に移すかはそのときのレディキューの状態による。

ドラおよびアラームハンドラから発行する場合は iset_flg システムコールを使用してください。

【使用例】

システムコール発行前のイベントフラグのパターンが 0xff であった場合、
システムコール発行後のパターンは 0xffff となります。

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global  task
task:
        :
    set_flg   ID_flg,H’0ff00
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.3.4. iset_flg(Set EventFlag)

【システムコール名】

iset_flg　→　イベントフラグをセットします (ハンドラ専用)

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
iset_flg flgid, setptn
```

《引数》

flgid	[**]	セットするイベントフラグの ID 番号
setptn	[****]	セットするビットパターン

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	セットするイベントフラグの ID 番号
R2	セットするビットパターン
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER iset_flg (flgid, setptn);
```

《引数》

ID	flgid;	イベントフラグ ID 番号
UINT	setptn;	セットするビットパターン

《戻り値》

関数の戻り値として常に E_OK を返します。

【エラーコード】

E_OK	00000000H(-H'00000000) : 正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034): オブジェクトが存在していない

【機能説明】

set_flg システムコールと同じ機能を割り込みハンドラ、周期起動ハンドラ、アラームハンドラから利用する場合にこのシステムコールを使用してください。

【使用例】

システムコール発行前のイベントフラグのパターンが 0xff であった場合、
システムコール発行後のパターンは 0xffff となります。

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global  intr
intr:
        :
    iset_flg   ID_flg,H’0ff00
        :
    ret_int
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.3.5. clr_flg(Clear EventFlag)

【システムコール名】

clr_flg　→　イベントフラグをクリアします

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
clr_flg  flgid, clrptn
```

《引数》

flgid	[**]	クリアするイベントフラグの ID 番号
clrptn	[****]	クリアするビットパターン

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	クリアするイベントフラグの ID 番号
R2	クリアするビットパターン
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER clr_flg (flgid, clrptn);
```

《引数》

ID	flgid;	イベントフラグ ID 番号
UINT	clrptn;	クリアするビットパターン

《戻り値》

関数の戻り値として常に E_OK を返します。

【エラーコード】

E_OK	00000000H(-H’00000000) : 正常終了
E_NOEXS	0FFFFFFCCH(-H’00000034): オブジェクトが存在していない

【機能説明】

flgid で示される 32 ビットイベントフラグのうち、対応する clrptn の 0 になっているビットをクリアします。つまり、flgid で示されるイベントフラグ値に対して、clrptn の値で論理積(AND)をとります。clrptn の全ビットを 1 とした場合、イベントフラグに対して何の操作も行なわないことになりますが、エラーにはなりません。

存在しないイベントフラグに対して、本システムコールを発行した場合は、エラー E_NOEXS を返します。

本システムコールはタスク、ハンドラのどちらからでも発行できます。

【使用例】

システムコール発行前のイベントフラグのパターンが 0xffff であったとすると、システムコール発行後のパターンは 0xff00 となります。

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.3.6. wai_flg(Wait EventFlag)

【システムコール名】

wai_flg　→　イベントフラグを待ちます

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
wai_flg  flgid, waiptn, wfmode
```

《引数》

flgid	[**]	イベントフラグの ID 番号
waiptn	[****]	待ちビットパターン
wfmode	[****]	待ちモード

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	待つイベントフラグの ID 番号
R2	待ち解除時のビットパターン
R3	待ちモード

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER wai_flg (p_flgptn, flgid, waiptn, wfmode);
```

《引数》

UINT	*p_flgptn;	待ち解除時のビットパターンを返す領域の先頭アドレス
ID	flgid;	イベントフラグ ID 番号
UINT	waiptn;	待ちビットパターン
UINT	wfmode;	待ちモード

《戻り値》

関数の戻り値としてエラーコードを返します。
p_flgptn の指す領域に、待ち解除時のビットパターンが設定されます。

【エラーコード】

E_OK	00000000H(-H’00000000)	:	正常終了
E_RLWAI	0FFFFFFAAH(-H’00000056)	:	待ち状態強制解除
E_DLT	0FFFFFFAFH(-H’00000051)	:	待ちオブジェクトが削除された
E_NOEXS	0FFFFFFCCH(-H’00000034)	:	オブジェクトが存在していない

【機能説明】

flgid で示されるイベントフラグにおいて、waiptn で指定したビットが wfmode で示される待ち解除条件にしたがってセットされるのを待ちます。
wfmode では、次のような指定を行います。

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

TWF_ANDW	AND 待ち
TWF_ORW	OR 待ち
TWF_CLR	クリア指定

すなわち、以下のような意味を持ちます。

wfmode(待ちモード)	意味
TWF_ANDW	Waiptn で指定したビットが全てセットされるのを待つ(AND 待ち)。
TWF_ANDW+TWF_CLR	Waiptn で指定したビットの AND 待ち条件解除が満たされてタスクが待ち解除となった場合、イベントフラグの値のすべてのビットをクリアする。
TWF_ORW	Waiptn で指定したビットのいずれかがセットされるの待つ(OR 待ち)。
TWF_ORW+TWF_CLR	waiptn で指定したビットの OR 待ち解除条件が満たされてタスクが待ち解除となった場合、イベントフラグの値のすべてのビットをクリアする。

flgptn には、本システムコールにより待ち状態が解除される時のイベントフラグの値（クリア指定の場合は、イベントフラグがクリアされる前の値)が格納されます。flgptn に返る値は、待ち解除条件を満たす値になっています。同一イベントフラグに対する複数タスクの待ちも可能です。この場合、一回の set_flg システムコール発行で複数のタスクが待ち解除となります。ただし、待ち行列中で待ち解除条件が満たされたタスクがクリア指定を行なっていた場合、そのタスクまでが待ち解除になります。

イベントフラグは以下に示す動作をするタスクの待ち行列を形成します。

- 待ち行列の順番は、FIFO(ファーストインファーストアウト)です。
- 待ち行列中にクリア指定のタスクがあれば、そのタスクが待ち解除時にフラグをクリアします。
- クリア指定をおこなっていたタスクよりも後ろの待ち行列にあったタスクは、既にクリアされた後のイベントフラグをもとに待ちを解除するか否かを決定するため、待ち解除とはなりません。

rel_wai、irel_wai システムコールによって待ち状態が強制解除された場合には、エラーコード E_RLWAI が返されます。また、他のタスクの発行した del_flg システムコールによって条件成立を待っているイベントフラグが削除された場合は、待ち状態にあったタスクは、イベントラグ待ち状態が解除され、そのタスクにエラーE_DLT が返されます。

存在しないイベントフラグに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラからは発行した場合は、正常に動作しません。

【使用例】

この例ではフラグ名が flg2 のイベントフラグの指定したビットがセットされるのを待ちます。指定したビットがセットされたタスクは待ち状態が解除されます。ここでは待ちモードとしてクリア指定をしているので、flg2 のイベントフラグはタスクの待ち状態が解除されると同時に 0 にクリアされます。

《 C 言語の使用例 》

```
#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”);
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global  task
task:
        :
    wai_flg   ID_flg2,H’0ff0,(TWF_ANDW+TWF_CLR)
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.3.7. twai_flg(Wait EventFlag with Timeout)

【システムコール名】

twai_flg　→　イベントフラグを待ちます（タイムアウトあり）

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
twai_flg flgid, waiptn, wfmode, tmout
```

《引数》

flgid	[**]	イベントフラグの ID 番号
waiptn	[****]	待ちビットパターン
wfmode	[****]	待ちモード
tmout	[****]	タイムアウト値

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	待つイベントフラグの ID 番号
R2	待ち解除時のビットパターン
R3	待ちモード
R4	タイムアウト値

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER twai_flg (p_flgptn, flgid, waiptn, wfmode, tmout);
```

《引数》

UINT	*p_flptn;	待ち解除時のビットパターンを返す領域の先頭アドレス
ID	flgid;	イベントフラグ ID 番号
UINT	waiptn;	待ちビットパターン
UINT	wfmode;	待ちモード
TMO	tmout;	タイムアウト値

《戻り値》

関数の戻り値としてエラーコードを返します。
p_flgptn の指す領域に、待ち解除時のビットパターンが設定されます。

【エラーコード】

E_OK	00000000H(-H'00000000) : 正常終了
E_RLWAI	0FFFFFFAAH(-H'00000056): 待ち状態強制解除
E_TMOUT	0FFFFFFABH(-H'00000055): ポーリング失敗またはタイムアウト
E_DLT	0FFFFFFAFH(-H'00000051): 待ちオブジェクトが削除された
E_NOEXS	0FFFFFFCCH(-H'00000034): オブジェクトが存在していない

【機能説明】

flgid で示されるイベントフラグにおいて、waiptn で指定したビットが wfmode で示される待ち解除条件にしたがってセットされるのを待ちます。本システムコールを発行したタスクは、イベントフラグ待ち行列とタイムアウト待ち行列の 2 つの待ち行列にタスクがつながれます。

本システムコール実行による待ち状態は、以下に示す場合に解除されます。なお、待ち状

態が解除されると本システムコールを発行したタスクはイベントフラグ待ち行列とタイムアウト待ち行列の 2 つの待ち行列からはずされ、レディキューに接続されます。

- tmout の時間が経過する前に、待ち解除条件が成立した場合

  この時のエラーコードは、E_OK を返します。

- 待ち解除条件が満たされないまま、tmout の時間が経過した場合

  この時のエラーコードは、E_TMOUT を返します。

- 他のタスクおよびハンドラから発行した rel_wai、irel_wai システムコールによって待ち状態が強制解除された場合

  この時のエラーコードは、E_RLWAI を返します。

- 他のタスクから発行した del_flg システムコールによって待ち状態の対象となっているイベントフラグが削除された場合

  この時のエラーコードは、E_DLT を返します。

tmout には、-1 から 7FFFFFFFH まで指定できます。なお、tmout に TMO_POL(=0)を指定した場合は、タイムアウト値として 0 を指定したことを示し、pol_flg と同じ動作をします。また、tmout=TMO_FEVR(-1)にした場合は、永久待ちの指定で、wai_flg システムコールと同じ動作になります。

wfmode の指定方法および各モードの意味については、wai_flg システムコールを参照してください。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラからは発行した場合、正常に動作しません。

【使用例】

この例ではフラグ名が flg2 のイベントフラグの指定したビットがセットされるかあるいは、待ち時間 tmout が経過するのを待ちます。指定したビットがセットされるかあるいは待ち時間を経過した場合、待ち状態が解除されます。

《 C 言語の使用例 》

```
#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”);
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global task
task:
        :
    twai_flg    ID_flg2,H’0ff0,(TWF_ANDW+TWF_CLR),5
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.3.8. pol_flg(Poll EventFlag)

【システムコール名】

pol_flg　→　イベントフラグを得ます（待ち無し）

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
pol_flg  flgid, waiptn, wfmode
```

《引数》

flgid	[**]	イベントフラグの ID 番号
waiptn	[****]	待ちビットパターン
wfmode	[****]	待ちモード

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	待つイベントフラグの ID 番号
R2	待ち解除時のビットパターン
R3	待ちモード

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER pol_flg (p_flgptn, flgid, waiptn, wfmode);
```

《引数》

UINT	*p_flgptn;	待ち解除時のビットパターンを返す領域の先頭アドレス
ID	flgid;	イベントフラグ ID 番号
UINT	waiptn;	待ちビットパターン
UINT	wfmode;	待ちモード

《戻り値》

p_flgptn の指す領域に、待ち解除時のビットパターンが設定されます。
戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_TMOUT	0FFFFFFABH(-H'00000055):	ポーリング失敗またはタイムアウト
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない

【機能説明】

flgid で示されるイベントフラグにおいて、waiptn で示される待ち解除ビットパターンが wfmode にしたがってセットされているかどうかを調べます。対象イベントフラグが既に wfmode で示される待ち解除の条件を満たしている場合には、wai_flg と同様の処理をして(クリア指定がある場合はイベントフラグをクリアする)、正常終了します。対象イベントフラグが wfmode で示される待ち解除の条件を満たしていない場合には、エラー E_TMOUT を返します。この場合、タスクは待ち状態にはなりません。また、この場合は、クリア指定があってもイベントフラグはクリアされません。

存在しないイベントフラグに対して、本システムコールを発行した場合は、エラー E_NOEXS を返します。

本システムコールはタスク、ハンドラのどちらからでも発行できます。

【使用例】

この例ではフラグ名が flg2 のイベントフラグの指定したビットがセットされているかどうかを調べます。クリア指定が行なわれているので、イベントフラグが条件を満たしていれば、0 クリアされます。

《 C 言語の使用例 》

```
#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”);
              :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.3.9. ref_flg(Refer EventFlag Status)

【システムコール名】

ref_flg　→　イベントフラグ状態を参照します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
ref_flg  flgid ,pk_rflg
```

《引数》

flgid	[**]	状態を参照するイベントフラグ ID 番号
pk_rflg	[****]	イベントフラグ状態を返すパケットアドレス

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	参照するイベントフラグの ID 番号
R2	イベントフラグ状態を返すパケットの先頭アドレス
R3	--

pk_rflg が指す領域には、以下の情報が返されます。

オフセット	サイズ		
+0	4	exinf	拡張情報
+4	2	wtsk	待ちタスクの有無
+8	4(U)	flgptn	イベントフラグのビットパターン

U は、unsigned のデータを表わします。

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER ref_flg (pk_rflg, flgid);
```

《引数》

T_RFLG	*pk_rflg;	イベントフラグ状態を返す構造体の先頭アドレス
ID	flgid;	イベントフラグ ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

pk_rflg が指す構造体には、以下のようなイベントフラグの状態が設定されます。

```
typedef struct t_rflg {
    VP      exinf;          /* 拡張情報 */
    BOOL_ID wtsk;           /* 待ちタスクの有無 */
    UINT    flgptn;         /* イベントフラグのビットパターン */
} T_RFLG;
```

【エラーコード】

E_OK	00000000H(-H'00000000) : 正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034): オブジェクトが存在していない

【機能説明】

flgid で示されたイベントフラグの以下の各種状態を返します。

●exinf

exinf には、拡張情報を返します。

●wtskid

wtskid には、待ち行列の先頭タスク(最も早く待ちに入ったタスク)の ID 番号を返します。待ちタスクの無い場合は FALSE(0)を返します。

●flgptn

flgptn は現在のイベントフラグの値を返します。

本システムコールで、対象イベントフラグが存在しない場合には、エラーE_NOEXS を返します。

本システムコールはタスク、ハンドラのどちらからでも発行できます。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.4. 同期・通信機能(セマフォ)

2.4.1. cre_sem(Create Semaphore)

【システムコール名】

cre_sem　→　セマフォを生成します

【アセンブリ言語による呼び出し方法】

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

《引数》

semid	[**]	生成するセマフォの ID 番号
pk_csem	[****]	セマフォ生成情報を格納した先頭アドレス (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	生成するセマフォの ID 番号
R2	セマフォ生成情報を格納したパケットの先頭アドレス
R3	--

pk_csem の指す領域には、以下に示す情報を設定してください。

オフセット	サイズ		
+0	4	exinf	拡張情報
+4	4	sematr	セマフォ属性
+8	4	isemcnt	セマフォの初期値
+12	4	maxsem	セマフォの最大値

【 C 言語による呼び出し方法】

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

《引数》

```
ID       semid;          生成するセマフォ ID 番号
T_CSEM   *pk_csem;       セマフォ生成情報
```

pk_csem の指す構造体に以下に示す情報を設定してください。

```
typedef struct t_csem {
    VP              exinf;          /* 拡張情報 */
    ATR             sematr;         /* セマフォ属性 */
    INT             isemcnt;        /* セマフォの初期値 */
    INT             maxsem;         /* セマフォの最大値 */
} T_CSEM;
```

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK          00000000H(-H'00000000) :  正常終了
E_OBJ         0FFFFFFC1H(-H'0000003f):  オブジェクトの状態が不正
```

【機能説明】

semid で示されたセマフォを生成します。
生成するセマフォの情報 pk_csem について以下に示します。

●exinf(拡張情報)

生成するセマフォに関する情報を格納するためにユーザが自由に利用できる領域です。exinf の内容に関しては、**MR32R** は関知しません。

●sematr(セマフォ属性)

ここで設定した値は、**MR32R** は関知しません。

●isemcnt

セマフォ生成時の対象セマフォカウンタの初期値をこの領域にセットします。この領域には、0 から 7FFFFFFFH までの値が設定できます。

●maxsem

対象セマフォカウンタの最大値をこの領域にセットします。この領域には、0 から 7FFFFFFFH までの値が設定できます。

すでに存在するセマフォに対して、本システムコールを発行した場合は、エラーE_OBJ を返します。本システムコールの指定可能な ID 番号の範囲は、1 から最大項目数定義で設定したユーザシステムの最大セマフォ数までです。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
#define ID_sem1 1
void task1()
{
    T_CSEM csem;
    csem.isemcnt = 0xff;     /* セマフォカウンタの初期値 */
    csem.maxsem = 0x7fffff;  /* セマフォカウンタの最大値 */
    cre_sem( ID_sem1, &setsem );
               :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
csem: .RES.B 16
ID_sem1:        .equ    1
    .include “mr32r.inc”
    .global   task1
task1:
        :
    ld24    R2,#setsem
    ld24    R1,#0xFF
    st      R1,@(8,R2)     /* セマフォカウンタの初期値設定 */
    seth    R1,#H’7F
    or3     R1,R1,#H’FFFF   /* セマフォカウンタの最大値設定 */
    st      R1,@(12,R2)
    cre_sem  ID_sem1
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
csem: .space 16
    .equ ID_sem1,1
    .include “mr32r.inc”
    .global   task1
task1:
        :
    ld24    R2,#setsem
    ld24    R1,#0xFF
    st      R1,@(8,R2)     /* セマフォカウンタの初期値設定 */
    seth    R1,#0x7F
    or3     R1,R1,#0xFFFF   /* セマフォカウンタの最大値設定 */
    st      R1,@(12,R2)
    cre_sem  ID_sem1
        :
    ext_tsk
```

2.4.2. del_sem(Delete Semaphore)

【システムコール名】

del_sem　→　セマフォを削除します

【アセンブリ言語による呼び出し方法】

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

《引数》

semid　　[**]　　削除するセマフォの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	削除するセマフォの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　　semid;　　削除するセマフォの ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK          00000000H(-H'00000000) :  正常終了
E_NOEXS       0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
```

【機能説明】

semid で示されたセマフォを削除します。

削除されたセマフォは、再度、同じ ID 番号で新しいセマフォを生成できます。削除するセマフォに対して、条件成立を待っているタスクが存在しても、本システムコールは正常終了します。その時、待ち状態にあったタスクは、セマフォ待ち状態が解除され、エラーE_DLT を返し、実行可能(READY)状態に移行します。

また、存在しないセマフォに対して、本システムコールを発行した場合は、エラー E_NOEXS を返します。なお、本システムコールの発行は、cre_sem システムコールによって生成したセマフォに対して行ってください。コンフィグレーションファイルで定義したセマフォに対して本システムコールを発行した場合は、正常に動作しません。

本システムコールはタスクからのみ発行することができます。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.4.3. sig_sem(Signal Semaphore)

【システムコール名】

sig_sem　→　セマフォ資源を返却します

【アセンブリ言語による呼び出し方法】

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

《引数》

semid　　[**]　　返却をおこなうセマフォの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	セマフォの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　　semid;　　セマフォ ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK      00000000H(-H'00000000):  正常終了
E_QOVR    0FFFFFFB7H(-H'00000049):  キューイングまたはネストのオーバーフロー
E_NOEXS   0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
```

【機能説明】

semid で示されたセマフォに対して、資源を 1 つ返却します。

対象セマフォの待ち行列にタスクがつながれている場合には、行列の先頭タスクを実行可能(READY)状態へ移行します。一方、待ち行列にタスクがつながれていない場合には、そのセマフォの計数値を 1 だけ増やします[22]。

セマフォの計数値が cre_sem あるいはコンフィグレーションファイルで指定した最大値(maxsem)を越えて資源の返却(sig_sem、isig_sem システムコール)をおこなうとセマフォの計数値はそのままで、システムコール発行タスクにエラーE_QOVR を返します。

存在しないセマフォに対して、本システムコールを発行した場合は、エラー E_NOEXS を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行する場合は isig_sem システムコールを使用しなければなりません。

[22] このシステムコールの発行によって、計数値がコンフィグレーションファイルで定義したセマフォの初期値を越えてもエラーにはなりません。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.4.4. isig_sem(Signal Semaphore)

【システムコール名】

isig_sem　→　セマフォ資源を返却します（ハンドラ専用）

【アセンブリ言語による呼び出し方法】

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

《引数》

semid　　[**]　　返却をおこなうセマフォの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	セマフォの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　　semid;　　セマフォ ID 番号

《戻り値》

関数の戻り値としてエラーコードを返す。

【エラーコード】

```
E_OK      00000000H(-H'00000000):  正常終了
E_QOVR    0FFFFFFB7H(-H'00000049):  キューイングまたはネストのオーバーフロー
E_NOEXS   0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
```

【機能説明】

sig_sem システムコールと同じ機能を割り込みハンドラ、周期起動ハンドラ、アラームハンドラから利用する場合に、このシステムコールを使用してください。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.4.5. wai_sem(Wait on Semaphore)

【システムコール名】

wai_sem　→　セマフォ資源を獲得します

【アセンブリ言語による呼び出し方法】

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

《引数》

semid　　[**]　　獲得をおこなうセマフォの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	セマフォの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　　semid;　　セマフォ ID 番号

《戻り値》

関数の戻り値としてエラーコードを返す。

【エラーコード】

```
E_OK        00000000H(-H'00000000) :  正常終了
E_RLWAI     0FFFFFFAAH(-H'00000056):  待ち状態強制解除
E_DLT       0FFFFFFAFH(-H'00000051):  待ちオブジェクトが削除された
E_NOEXS     0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
```

【機能説明】

semid で示されたセマフォから、資源を一つ獲得する操作をおこないます。そのセマフォの計数値が 1 以上の場合には、計数値を 1 だけ減じて、システムコール発行タスクは実行を継続します。一方、セマフォの計数値が 0 の場合には、計数値は変更せず、システムコール発行タスクをそのセマフォの待ち行列に FIFO[23]順でつながれます。

rel_wai、irel_wai システムコールによって待ち状態が強制解除された場合には、エラーE_RLWAI が返されます。また、他のタスクの発行した del_sem システムコールによって条件成立を待っているセマフォが削除された場合は、待ち状態にあったタスクは、セマフォ待ち状態が解除され、実行可能(READY)状態に移行します。この時、そのタスクには、エラーE_DLT が返されます。

存在しないセマフォに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

[23] ファーストインファーストアウト。すなわち wai_sem システムコールで待ちに入った順で sig_sem、isig_sem システムコールで待ちが解除されます。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合、正常に動作しません。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.4.6. twai_sem(Wait on Semaphore with Timeout)

【システムコール名】

twai_sem　→　セマフォ資源を獲得します（タイムアウトあり）

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
twai_sem semid, tmout
```

《引数》

semid	[**]	獲得をおこなうセマフォ ID 番号
tmout	[****]	タイムアウト値

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	セマフォの ID 番号
R2	--
R3	--
R4	タイムアウト値

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER twai_sem (semid,tmout);
```

《引数》

ID	semid;	セマフォ ID 番号
TMO	tmout;	タイムアウト値

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_TMOUT	0FFFFFFABH(-H'00000055):	ポーリング失敗またはタイムアウト
E_RLWAI	0FFFFFFAAH(-H'00000056):	待ち状態強制解除
E_DLT	0FFFFFFAFH(-H'00000051):	待ちオブジェクトが削除された
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない

【機能説明】

semid で示されたセマフォから、資源を一つ獲得する操作を行います。

そのセマフォの計数値が 1 以上の場合には、計数値を 1 だけ減じてシステムコール発行タスクは実行を継続します。一方、セマフォの計数値が 0 の場合には、計数値は変更せず、システムコール発行タスクをそのセマフォ待ち行列とタイムアウト待ち行列につなぎます。

本システムコール実行による待ち状態は、以下に示す場合に解除されます。なお、待ち状態が解除されると本システムコールを発行したタスクはセマフォ待ち行列とタイムアウト待ち行列の 2 つの待ち行列からはずされ、レディキューに接続されます。

- tmout の時間が経過する前に、sig_sem、isig_sem システムコールが発行され、待ち解除条件が満足された場合

  この場合、エラーコードは、E_OK を返します。

●待ち解除条件が満足されないまま、tmout の時間が経過した場合

この場合、エラーコードは、E_TMOUT を返します。

●他のタスクおよびハンドラから発行した rel_wai、irel_wai システムコールによって待ち状態が強制解除された場合

この場合、エラーコードは、E_RLWAI を返します。

●他のタスクから発行した del_sem システムコールによって待ち状態の対象となっているセマフォが削除された場合

この場合、エラーコードは、E_DLT を返します。

存在しないセマフォに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

tmout には、-1 から 7FFFFFFFH まで指定できます。なお、tmout に TMO_POL=0 を指定した場合は、タイムアウト値として 0 を指定したことを示し、preq_sem と同じ動作をします。また、tmout=TMO_FEVR(-1)にした場合は、永久待ちの指定で、wai_sem システムコールと同じ動作をします。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合、正常に動作しません。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.4.7. preq_sem(Poll and Request Semaphore)

【システムコール名】

preq_sem　→　セマフォ資源を獲得します（待ち無し）

【アセンブリ言語による呼び出し方法】

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

《引数》

semid　　[**]　　獲得をおこなうセマフォの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	セマフォの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　　semid;　　セマフォ ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK        00000000H(-H'00000000) :  正常終了
E_TMOUT     0FFFFFFABH(-H'00000055):  ポーリング失敗またはタイムアウト
E_NOEXS     0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
```

【機能説明】

semid で示されたセマフォから資源を一つ獲得(待ち無し)します。対象セマフォの計数値が1以上の場合には計数値を1だけ減じて、システムコール発行タスクは実行を継続します。一方、セマフォの計数値が 0 の場合には計数値は変更せず、システムコール発行タスクにエラーE_TMOUT を返し、本システムコールを終了します。

存在しないセマフォに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスク、ハンドラのどちらからでも発行できます。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.4.8. ref_sem(Refer Semaphore Status)

【システムコール名】

ref_sem　→　セマフォ状態を参照します

【アセンブリ言語による呼び出し方法】

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

《引数》

semid	[-*]	セマフォ ID 番号
pk_rsem	[****]	状態を返すパケットの先頭アドレス (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	参照するセマフォの ID 番号
R2	セマフォの状態を返すパケットの先頭アドレス
R3	--

pk_rsem が指す領域には、以下の情報が返されます。

オフセット	サイズ		
+0	4	exinf	拡張情報
+4	2	wtsk	待ちタスクの有無
+8	4	semcnt	現在のセマフォのカウント値

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER ref_sem(pk_rsem,semid);
```

《引数》

T_RSEM	*pk_rsem ;	セマフォ状態を返す構造体の先頭アドレス
ID	semid;	セマフォ ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。
pk_rsem が指す構造体には、以下の情報が返されます。

```
typedef struct t_rsem{
    VP      exinf;          /* 拡張情報 */
    BOOL_ID wtsk;           /* 待ちタスクの有無 */
    INT     semcnt;         /* 現在のセマフォカウント値 */
} T_RSEM;
```

【エラーコード】

E_OK	00000000H(-H'00000000) : 正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034): オブジェクトが存在していない

【機能説明】

semid で示されたセマフォの各種の状態を返します。

●exinf

拡張情報を返します。

●wtsk

wtsk には待ち行列の先頭タスク(最も早く待ちに入ったタスク)の ID 番号を返します。待ちタスクの無い場合は 0(FALSE)を返します。

●semcnt

semcnt には、現在のセマフォカウント値を返します。

存在しないセマフォに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスク、ハンドラのどちらからでも発行できます。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.5. 同期・通信機能(メイルボックス)

2.5.1. cre_mbx(Create Mailbox)

【システムコール名】

cre_mbx　→　メイルボックスを生成します

【アセンブリ言語による呼び出し方法】

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

《引数》

mbxid	[**]	生成するメイルボックスの ID 番号
pk_cmbx	[****]	メイルボックス生成情報を格納した先頭アドレス (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	生成するメイルボックスの ID 番号
R2	生成情報を返すパケットの先頭アドレス
R3	--

pk_cmbx の指す領域には、以下に示す情報を設定してください。

オフセット	サイズ		
+0	4	exinf	拡張情報
+4	4	mbxatr	メイルボックス属性
+8	4	bufcnt	リングバッファの大きさ
+12	4	mbx	ユーザーが確保したメイルボックス領域の先頭アドレス

【 C 言語による呼び出し方法】

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

《引数》

```
ID       mbxid;          生成するメイルボックス ID 番号
T_CMBX   *pk_cmbx;       メイルボックス生成情報
```

pk_cmbx の指す構造体に以下に示す情報を設定してください。

```
typedef struct t_cmbx{
    VP              exinf;          /* 拡張情報 */
    ATR             mbxatr;         /* メイルボックス属性 */
    INT             bufcnt;         /* リングバッファの大きさ */
    VP              mbx;            /* ユーザーが確保したメイルボック
                                       ス領域の先頭アドレス */
} T_CMBX;
```

《戻り値》

関数の戻り値としてエラーコードを返す。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_OBJ	0FFFFFFC1H(-H'0000003f):	オブジェクトの状態が不正
E_NOMEM	0FFFFFFF6H(-H'0000000a):	メモリ不足

【機能説明】

mbxid で示されたメイルボックスを生成します。
生成するメイルボックスの情報 pk_cmbx について以下に示します。

●exinf(拡張情報)

生成するメイルボックスに関する情報を格納するためにユーザが自由に利用できる領域です。exinf の内容に関しては、**MR32R** は関知しません。

●mbxatr(メイルボックス属性)

生成するメイルボックスが使用するメッセージを格納する領域を内蔵 RAM にするか外部 RAM にするかユーザーが確保するかを指定します。__MR_USER を指定した場合、mbx にメイルボックス領域の先頭アドレスを指定してください。

★ 内蔵 RAM を使用する場合
__MR_INT(0)を指定してください。

★ 外部 RAM を使用する場合
__MR_EXT(0x10000)を指定してください。

★ ユーザーが確保した領域をメイルボックスとして使用する場合
__MR_USER(0x20000)を指定してください。

●bufcnt

対象メイルボックスのメッセージを格納するバッファサイズをここで指定します。この単位は、バイト数ではなく、メッセージ数を指定します。

すでに、存在するメイルボックスに対して、本システムコールを発行した場合は、エラー E_OBJ を返します。
本システムコールの指定可能な ID 番号の範囲は、1 から最大項目数定義で設定したユーザシステムの最大メイルボックス数までです。
OS カーネルがメイルボックス領域を割り当てる際、pk_cmbx の bufcnt で指定したメッセージ分のメモリがない場合は、本システムコールを発行したタスクに対してエラーE_NOMEM を返します。
本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

```
    .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
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .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)

【システムコール名】

del_mbx　→　メイルボックスを削除します

【アセンブリ言語による呼び出し方法】

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

《引数》

mbxid　　[**]　　削除するメイルボックスの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	削除するメイルボックスの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　　mbxid;　　削除するメイルボックスの ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK          00000000H(-H’00000000) :  正常終了
E_NOEXS       0FFFFFFCCH(-H’00000034):  オブジェクトが存在していない
```

【機能説明】

mbxid で示されたメイルボックスを削除します。

削除されたメイルボックスは、再度、同じ ID 番号で新しいメイルボックスを生成できます。削除するメイルボックスに対して、条件成立を待っているタスクが存在しても、本システムコールは正常終了します。その時、待ち状態にあったタスクは、メッセージ待ち状態が解除され、エラーE_DLT を返し、実行可能(READY)状態に移行します。削除するメイルボックスの中にメッセージがあった場合、メッセージは消滅します。本システムコールにより、対象メイルボックス領域は解放され、再利用可能な状態となります。

存在しないメイルボックスに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .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)

【システムコール名】

snd_msg　→　メッセージを送信します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
snd_msg  mbxid, pk_msg
```

《引数》

mbxid	[**]	送信を行うメイルボックスの ID 番号
pk_msg	[****]	メッセージパケットの先頭アドレス

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	送信を行うメイルボックスの ID 番号
R2	メッセージパケットの先頭アドレス
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER snd_msg (mbxid, pk_msg);
```

《引数》

ID	mbxid;	メイルボックス ID 番号
T_MSG	*pk_msg;	メッセージパケットの先頭アドレス

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK      00000000H(-H'00000000):正常終了
E_QOVR    0FFFFFFB7H(-H'00000049):キューイングまたはネストのオーバーフロー
E_NOEXS   0FFFFFFCCH(-H'00000034):オブジェクトが存在していない
```

【機能説明】

mbxid で示されたメイルボックスにメッセージを送信します。

メッセージを待つタスクがないときには、メッセージはメッセージキューに FIFO[24]順で格納します。すなわち、本システムコール発行によりメイルボックスに送信された順にメッセージが取り出されます。メッセージを待つタスクがある場合は、メッセージをそのタスクに渡し、そのタスクの待ち状態を解除します。メッセージキューのサイズは、cre_mbx システムコールで指定するか、あるいは、コンフィグレーションファイルにより定義します。メッセージキューが一杯になった状態のメイルボックスに対して本システムコールを発行した場合、システムコール発行タスクにエラーE_QOVR を返します。

存在しないメイルボックスに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

メッセージは 32 ビット幅データです。μITRON 仕様では、このデータをメッセージパケット(メッセージを含む構造体)の先頭アドレスとして扱うことを標準としています(アドレス

[24] ファーストインファーストアウト

渡し)が、**MR32R**では、メッセージを以下の 2 通りの方法でデータを通信することができます。

1. メッセージを、メッセージパケットの先頭アドレス(32 ビット)とする場合
   MR32Rでは、メッセージパケットの型(T_MSG)は、"mr32r.h"で以下のように定義されています。
   typedef UW T_MSG;
   メッセージパケットの型がこれと異なる場合、型変換してシステムコールを呼び出してください。[25]。
2. メッセージを単なる 32 ビットデータとする場合
   この場合、snd_msg、isnd_msg システムコールの第二引数(送信するメッセージデータ pk_msg)を(PT_MSG)で、rcv_msg、prcv_msg の第一引数(メッセージデータを格納する領域のアドレス ppk_msg)を(PT_MSG *)で型変換しなければなりません。
   例えば、int 型の変数 i を通信する場合、

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

   このように記述すれば、直接 32 ビットデータを通信できます。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行する場合は isnd_msg システムコールを使用してください。

【使用例】

《 C 言語の使用例 》

この例では、メッセージパケットの先頭アドレスを送る場合を示しています。

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

[25] 本マニュアルの**【 C 言語による呼び出し方法】**ではメッセージパケットの先頭アドレスを送信することを標準として記述しています。

2.5.4. isnd_msg(Send Message to Mailbox)

【システムコール名】

isnd_msg　→　メッセージを送信します（ハンドラ専用）

【アセンブリ言語による呼び出し方法】

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

《引数》

mbxid	[**]	送信をおこなうメイルボックス ID 番号
pk_msg	[****]	メッセージパケットの先頭アドレス

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	送信を行うメイルボックスの ID 番号
R2	メッセージパケットの先頭アドレス
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER isnd_msg (mbxid, pk_msg);
```

《引数》

ID	mbxid;	メイルボックス ID 番号
T_MSG	*pk_msg;	メッセージパケットの先頭アドレス

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK      00000000H(-H'00000000):  正常終了
E_QOVR    0FFFFFFB7H(-H'00000049):  キューイングまたはネストのオーバーフロー
E_NOEXS   0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
```

【機能説明】

snd_msg システムコールの機能をハンドラから利用する場合に、このシステムコールを使用します。

【使用例】

《 C 言語の使用例 》

```
#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”);
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.5.5. rcv_msg(Receive Message from Mailbox)

【システムコール名】

rcv_msg　→　メッセージを受信します

【アセンブリ言語による呼び出し方法】

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

《引数》

mbxid　　[**]　　受信するメイルボックスの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	受信を行うメイルボックスの ID 番号
R2	メッセージパケットの先頭アドレス
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER rcv_msg (ppk_msg, mbxid);
```

《引数》

```
ID      mbxid;      メイルボックス ID 番号
T_MSG   **ppk_msg;  メッセージパケットの先頭アドレスを格納する領域のアドレス
```

《戻り値》

関数の戻り値としてエラーコードを返します。
ppk_msg が指す領域に、受信したメッセージパケットの先頭アドレスが設定されます。

【エラーコード】

```
E_OK        00000000H(-H'00000000) :  正常終了
E_RLWAI     0FFFFFFAAH(-H'00000056):  待ち状態強制解除
E_DLT       0FFFFFFAFH(-H'00000051):  待ちオブジェクトが削除された
E_NOEXS     0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
```

【機能説明】

mbxid で示されたメイルボックスからメッセージを受信します。

対象メイルボックスにメッセージが到着している場合には、メッセージキューの先頭にあるメッセージを 1 つ取り出して、それをリターンパラメータ pk_msg として返します。一方、そのメイルボックスにまだメッセージが送信されていない場合、本システムコールを発行したタスクは待ち状態となり、待ち行列に FIFO 順でつながれます。

rel_wai、irel_wai システムコールによって待ち状態が解除された場合には、エラー E_RLWAI が返されます。また、他のタスクの発行した del_mbx システムコールによって条件成立を待っているメイルボックスが削除された場合、メッセージ待ち状態にあったタスクには、エラーE_DLT を返し、メッセージ待ち状態が解除され、実行可能(READY)状態に移行します。

存在しないメイルボックスに対して、本システムコールを発行した場合は、エラー

E_NOEXS を返します。
本システムコールはタスクからのみ発行できます。
メッセージを受信する場合、以下の点の注意が必要です。

1. メッセージをメッセージパケットの先頭アドレスとする場合
   メッセージ幅が 32 ビットですので、メッセージパケットの先頭アドレスを格納する領域へのポインタ変数(ppk_msg)を以下のように宣言しなければなりません。

```
T_MSG **ppk_msg;
```

2. メッセージを、単なる 32 ビットのデータとする場合
   rcv_msg 、prcv_msg の第一引数(メッセージデータを格納する領域のアドレス(ppk_msg)を(PT_MSG *)で型変換してください。

例えば、int 型の変数 i を通信する場合、

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

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

この例では、メッセージをデータの先頭アドレスとして送る場合を示している。

```
#include <mr32r.h>
#include “id.h”
void task()
{
    T_MSG *msg;
              :
    if( rcv_msg( (T_MSG **)&msg, ID_mbx ) != E_OK )
        error(“forced wakeup¥n”);
              :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.5.6. trcv_msg(Receive Message with Timeout)

【システムコール名】

trcv_msg　→　メッセージを受信します（タイムアウトあり）

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
trcv_msg mbxid,tmout
```

《引数》

mbxid	[**]	受信するメイルボックスの ID 番号
tmout	[****]	タイムアウト値

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	受信を行うメイルボックスの ID 番号
R2	メッセージパケットの先頭アドレス
R3	--
R4	タイムアウト値

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER trcv_msg (ppk_msg, mbxid, tmout);
```

《引数》

ID	mbxid;	メイルボックス ID 番号
T_MSG	**ppk_msg;	メッセージパケットの先頭アドレスを格納する領域のアドレス
TMO	tmout	タイムアウト値

《戻り値》

関数の戻り値としてエラーコードを返します。
ppk_msg が指す領域に、受信したメッセージパケットの先頭アドレスが設定されます。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_TMOUT	0FFFFFFABH(-H'00000055):	ポーリング失敗またはタイムアウト
E_RLWAI	0FFFFFFAAH(-H'00000056):	待ち状態強制解除
E_DLT	0FFFFFFAFH(-H'00000051):	待ちオブジェクトが削除された
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない

【機能説明】

mbxid で示されたメイルボックスにメッセージがあれば受信します。そのメイルボックスにメッセージが入っている場合には、メッセージキューの先頭にあるメッセージを 1 つ取り出して、それをリターンパラメータ ppk_msg として得ます。

一方、そのメイルボックスに、まだメッセージが送信されていない場合には、本システムコールを発行したタスクは、待ち状態となり、メッセージ待ち行列とタイムアウト待ち行列の 2 つの待ち行列につながれます。

本システムコール実行による待ち状態は、以下に示す場合に解除されます。なお、待ち状

態が解除されると本システムコールを発行したタスクはメッセージ待ち行列とタイムアウト待ち行列の 2 つの待ち行列からはずされ、レディキューに接続されます。

- tmout の時間が経過する前にメッセージが到着した場合

  この時のエラーコードは、E_OK を返します。

- メッセージが到着しないまま、tmout の時間が経過した場合

  この時のエラーコードは、E_TMOUT を返します。

- 他のタスクおよびハンドラから発行した rel_wai、irel_wai システムコールによって待ち状態が強制解除された場合

  この時のエラーコードは、E_RLWAI を返します。

- 他のタスクから発行した del_mbx システムコールによって待ち状態の対象となっているメイルボックスが削除された場合

  この時のエラーコードは、E_DLT を返します。

tmout には、-1 から 7FFFFFFFH まで指定できます。なお、tmout に TMO_POL=0 を指定した場合は、タイムアウト値として 0 を指定したことを示し、prcv_msg と同じ動作をします。また、tmout=TMO_FEVR(-1)にした場合は、永久待ちの指定で、rcv_msg システムコールと同じ動作になります。

メッセージ受信時の注意事項は、rcv_msg を参照してください。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
typedef char T_MSG;
void task()
{
    T_MSG   *msg;
                :
    if( trcv_msg( (T_MSG **)&msg, ID_mbx, 10 ) != E_OK ){
        error(“Can’t Get Message¥n”);
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global task
task:
        :
    trcv_msg ID_mbx,10
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .include “mr32r.inc”
    .global task
task:
        :
    trcv_msg ID_mbx,10
        :
```

2.5.7. prcv_msg(Poll and Receive Message)

【システムコール名】

prcv_msg　→　メッセージを受信します（待ちなし）

【アセンブリ言語による呼び出し方法】

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

《引数》

mbxid　[**]　受信するメイルボックスの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	受信を行うメイルボックスの ID 番号
R2	メッセージパケットの先頭アドレス
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER prcv_msg (ppk_msg, mbxid);
```

《引数》

ID	mbxid;	メイルボックス ID 番号
T_MSG	**ppk_msg;	メッセージパケットの先頭アドレスを格納する領域のアドレス

《戻り値》

関数の戻り値としてエラーコードを返します。
ppk_msg が指す領域に、受信したメッセージパケットの先頭アドレスが設定されます。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_TMOUT	0FFFFFFABH(-H'00000055):	ポーリング失敗またはタイムアウト
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない

【機能説明】

mbxid で示されたメイルボックスからメッセージがあれば受信します(待ちなし)。そのメイルボックスにメッセージが入っている場合には、メッセージキューの先頭にあるメッセージを１つ取り出して、それをリターンパラメータ ppk_msg として得ます。

一方、そのメイルボックスに、まだメッセージが送信されていない場合には、システムコール発行タスクにエラーE_TMOUT を返し、本システムコールを終了します。rcv_msg、trcv_msg とは、異なり本システムコールを発行したタスクは、待ち状態には移行しません。

また、存在しないメイルボックスに対して、本システムコールを発行した場合は、エラー E_NOEXS を返します。

メッセージ受信時の注意事項は、rcv_msg を参照してください。

本システムコールはタスク、タスク独立部(割り込みハンドラ、周期起動ハンドラおよびアラームハンドラ)のどちらからでも発行できます。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
typedef char T_MSG;
void task()
{
    T_MSG * msg;
                :
    if( prcv_msg( (T_MSG **)&msg, ID_mbx ) != E_OK ){
        error(“Can’t Get Message¥n”);
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global  task
task:
    prcv_msg ID_mbx1
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .include “mr32r.inc”
    .global  task
task:
    prcv_msg ID_mbx1
        :
```

2.5.8. ref_mbx(Refer Mailbox Status)

【システムコール名】

ref_mbx　→　メイルボックス状態を参照します

【アセンブリ言語による呼び出し方法】

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

《引数》

mbxid	[**]	状態を参照するメイルボックス ID 番号
pk_rmbx	[****]	メイルボックス状態を返すパケットの先頭アドレス (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	参照するメイルボックスの ID 番号
R2	メイルボックスの状態を返すパケットの先頭アドレス
R3	--

pk_rmbx の指す領域には、以下の情報が返されます。

オフセット	サイズ		
+0	4	exinf	拡張情報
+4	2	wtsk	待ちタスクの有無
+8	4(U)	pk_msg	次に受信されるメッセージパケットの先頭アドレス
+12	4	msgcnt	メールボックスに蓄積されているメッセージの数

U は、unsigned のデータを表わします。

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER ref_mbx (pk_rmbx, mbxid);
```

《引数》

T_RMBX	*rmbx;	メイルボックス状態を返す構造体の先頭アドレス
ID	mbxid;	メイルボックス ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します 。

pk_rmbx の指す構造体には、以下の情報が返されます。

```
typedef struct t_rmbx {
    VP      exinf;   /*拡張情報*/
    BOOL_ID wtsk;    /*待ちタスクの有無*/
    T_MSG   *pk_msg; /*次に受信されるメッセージパケットの先頭アドレス*/
    INT     msgcnt;  /*メールボックスに蓄積されているメッセージの数*/
} T_RMBX;
```

【エラーコード】

E_OK	00000000H(-H'00000000)	:正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034)	:オブジェクトが存在していない

【機能説明】

本システムコールでは、mbxid で指定したメイルボックスの以下の状態を返す。

●exinf

拡張情報を返します。

●wtsk

wtsk には、指定したメイルボックスでメッセージを待っている先頭タスク ID 番号を返します。メッセージを待っているタスクがない場合は FALSE(0)を返します。

●pk_msg

pk_msg には、次に rcv_msg を実行した場合に受信されるメッセージ(キューイングされている先頭メッセージ)を返します。メッセージがない場合は、NADR(-1) = 0xFFFFFFFF を返します。

存在しないメイルボックスに対して、本システムコールを発行した場合は、エラー E_NOEXS を返します。

本システムコールはタスク、タスク独立部(割り込みハンドラ、周期起動ハンドラ、アラームハンドラ)のどちらからでも発行できます。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.6. 拡張同期・通信機能(メッセージバッファ)

2.6.1. cre_mbf(Create MessageBuffer)

【システムコール名】

cre_mbf　→　メッセージバッファを生成します

【アセンブリ言語による呼び出し方法】

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

《引数》

mbfid	[**]	生成するメッセージバッファ ID 番号
pk_cmbf	[****]	メッセージバッファ生成情報を格納した先頭アドレス (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	生成するメッセージバッファの ID 番号
R2	生成情報を返すパケットの先頭アドレス
R3	--

pk_cmbf の指す領域には、以下に示す情報を設定してください。

オフセット	サイズ		
+0	4	exinf	拡張情報
+4	4	mbfatr	メッセージバッファ属性
+8	4	bufsz	メッセージバッファのサイズ
+12	4	maxmsz	メッセージの最大長
+16	4	mbf	ユーザーが確保したメッセージバッファ領域の先頭アドレス

【 C 言語による呼び出し方法】

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

《引数》

```
ID       mbfid;          生成するメッセージバッファ ID 番号
T_CMBF   *pk_cmbf;       メッセージバッファ生成情報
```

pk_cmbx の指す構造体に以下に示す情報を設定してください。

```
typedef struct t_cmbf{
    VP      exinf;          /* 拡張情報 */
    ATR     mbfatr;         /* メッセージバッファ属性 */
    INT     bufsz;          /* メッセージバッファのサイズ*/
    INT     maxmsz;         /* メッセージの最大長 */
    VP      mbf;            /* ユーザーが確保したメッセージバッファ
                               領域の先頭アドレス */
} T_CMBF;
```

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_NOMEM	0FFFFFFF6H(-H'0000000a):	メモリ不足
E_OBJ	0FFFFFFC1H(-H'0000003f):	オブジェクトの状態が不正

【機能説明】

mbfid で示されたメッセージバッファを生成します。
生成されたメッセージバッファは、bufsz で指定されたサイズのリングバッファで構成されます。
生成するメッセージバッファの情報 pk_cmbf について以下に示します。

●exinf(拡張情報)

生成するメッセージバッファに関する情報を格納するためにユーザが自由に利用できる領域です。exinf の内容に関しては、**MR32R** は関知しません。

●mbfatr(メッセージバッファ属性)

生成するメッセージバッファが使用するバッファ領域を内蔵 RAM にするか外部 RAM にするかユーザーが確保するかを指定します。__MR_USER を指定した場合、mbf にメッセージバッファ領域の先頭アドレスを指定してください。

★ 内蔵 RAM を使用する場合
__MR_INT(0)を指定してください。
★ 外部 RAM を使用する場合
__MR_EXT(0x10000)を指定してください。
★ ユーザーが確保した領域をメッセージバッファとして使用する場合
__MR_USER(0x20000)を指定してください。

●bufsz(4 の倍数を指定する)

生成するメッセージバッファのサイズを指定してください。ここで指定するサイズは、4 の倍数になるように指定してください。なお、bufsz=0 を指定することも可能です。この場合、メッセージバッファの送受信側が完全に同期した通信を行うことになります。

●maxmsz

生成するメッセージバッファで扱うことのできるメッセージの最大長を指定してください。なお、maxmsz は、**MR32R** は参照しませんので、この指定は必要ありません。他のリアルタイム OS との互換性を保つ必要がある場合は、この値を設定してください。

本システムコールの指定可能な ID 番号の範囲は、1 から最大項目数定義で設定した最大メッセージバッファ数までです。
OS カーネルがメッセージバッファ領域を割り当てる際、pk_cmbf の bufsz で指定したメッセージ分のメモリがない場合は、本システムコールを発行したタスクに対してエラーE_NOMEM を返します。
存在するメッセージバッファに対して、本システムコールを発行した場合は、エラーE_OBJ を返します。
本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#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 );
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
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
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
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)

【システムコール名】

del_mbf　→　メッセージバッファを削除します

【アセンブリ言語による呼び出し方法】

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

《引数》

mbfid　[**]　削除するメッセージバッファの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	削除するメッセージバッファの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　mbfid;　削除するメッセージバッファの ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK        00000000H(-H’00000000) :  正常終了
E_NOEXS     0FFFFFFCCH(-H’00000034):  オブジェクトが存在していない
```

【機能説明】

mbfid で示されたメッセージバッファを削除します。

削除されたメッセージバッファは、再度、同じ ID 番号で新しいメッセージバッファを生成できます。削除するメッセージバッファに対して、条件成立を待っているタスクが存在しても、本システムコールは正常終了します。その時、待ち状態にあったタスクは、メッセージバッファ待ち状態が解除され、そのタスクにエラーE_DLT を返します。メッセージバッファ領域が解放されるため、メッセージバッファの中にあったメッセージは消滅します。本システムコールにより、対象メッセージバッファ領域は解放され、再利用可能な状態となります。

存在しないメッセージバッファに対して、本システムコールを発行した場合は、エラー E_NOEXS を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .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)

【システムコール名】

snd_mbf　→　メッセージを送信します

【アセンブリ言語による呼び出し方法】

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

《引数》

mbfid	[**]	送信をおこなうメッセージバッファの ID 番号
msgsz	[****]	送信メッセージのサイズ
msg	[****]	メッセージパケットの先頭アドレス (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	送信するメッセージバッファの ID 番号
R2	メッセージパケットの先頭アドレス
R3	送信メッセージのサイズ

【 C 言語による呼び出し方法】

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

《引数》

ID	mbfid;	メッセージバッファ ID 番号
INT	msgsz;	送信メッセージのサイズ
VP	msg;	送信メッセージの先頭アドレス

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H’00000000) :	正常終了
E_RLWAI	0FFFFFFAAH(-H’00000056):	待ち状態強制解除
E_DLT	0FFFFFFAFH(-H’00000051):	待ちオブジェクトが削除された
EV_RST	0FFFFFF02H(-H’000000FE):	リセットにより待ちが解除された
E_NOEXS	0FFFFFFCCH(-H’00000034):	オブジェクトが存在していない

【機能説明】

mbfid で示されたメッセージバッファに対して msg のアドレスに入っているメッセージを送信します。メッセージのサイズは、msgsz に指定してください。msg 以下の msgsz バイトが、mbfid で指定されたメッセージバッファのメッセージキューにコピーされます。メッセージキューは、リングバッファによって実現されます。

msgsz が、cre_mbf で指定した maxmsz よりも大きくなっても、エラーにはなりません[26]。

バッファの空き領域が少なく、msg のメッセージがメッセージキューに入らない場合、本システムコールを発行したタスクは送信待ち状態となり、送信待ち行列につながれます。待

[26] msgsz が、cre_mbf で指定した maxmsz よりも小さいかどうかはユーザ側でチェックしてください。

ち行列の順序は FIFO です。

rel_wai、irel_wai システムコールによって待ち状態が強制解除された場合には、エラー E_RLWAI が返されます。

他のタスクが発行した del_mbf システムコールによって条件成立を待っているメッセージバッファが削除された場合、待ち状態にあったタスクは、送信メッセージバッファ待ち状態が解除され、そのタスクにエラーE_DLT が返されます。また、他のタスクが発行した vrst_mbf システムコールによって条件成立を待っているメッセージバッファが初期化された場合、待ち状態にあったタスクは、送信メッセージバッファ待ち状態が解除され、そのタスクにエラーEV_RST が返されます。

存在しないメッセージバッファに対して、本システムコールを発行した場合は、エラー E_NOEXS を返します。

本システムコールはタスクからのみ発行できます。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

この例では、メッセージパケットの先頭アドレスを送る場合を示しています。

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
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)

【システムコール名】

tsnd_mbf　→　メッセージを送信します(タイムアウトあり)

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
tsnd_mbf mbfid, msgsz,tmout
```

《引数》

mbfid	[**]	送信をおこなうメッセージバッファの ID 番号
msgsz	[****]	送信メッセージのサイズ
msg	[****]	メッセージパケットの先頭アドレス (R2 レジスタに先頭アドレスを設定してください)
tmout	[****]	タイムアウト値

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	送信するメッセージバッファの ID 番号
R2	メッセージパケットの先頭アドレス
R3	送信メッセージのサイズ
R4	タイムアウト値

【 C 言語による呼び出し方法】

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

《引数》

ID	mbfid;	メッセージバッファ ID 番号
INT	msgsz;	送信メッセージのサイズ
VP	msg;	送信メッセージの先頭アドレス
TMO	tmout;	タイムアウト値

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_RLWAI	0FFFFFFAAH(-H'00000056):	待ち状態強制解除
E_DLT	0FFFFFFAFH(-H'00000051):	待ちオブジェクトが削除された
EV_RST	0FFFFFF02H(-H'000000FE):	リセットにより待ちが解除された
E_TMOUT	0FFFFFFABH(-H'00000055):	ポーリング失敗またはタイムアウト
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない

【機能説明】

mbfid で示されたメッセージバッファに対して msg のアドレスに入っているメッセージを送信します。メッセージのサイズは、msgsz に指定してください。msg 以下の msgsz バイトが、mbfid で指定されたメッセージバッファのメッセージキューにコピーされます。メッセージキューは、リングバッファによって実現されます。msgsz が、cre_mbf で指定した maxmsz よ

りも大きくなっても、エラーにはなりません[27]。
　バッファの空き領域が少なく、msg のメッセージがメッセージキューに入らない場合、本システムコールを発行したタスクは送信待ち状態となり、送信待ち行列とタイムアウト待ち行列の 2 つの待ち行列につながれます。待ち行列の順序は FIFO です。
　本システムコール発行による待ち状態は、以下に示す場合に解除されます。なお、待ち状態が解除されると本システムコールを発行したタスクは、送信待ち行列とタイムアウト待ち行列の 2 つの待ち行列からはずされ、レディキューに接続されます。

- tmout の時間が経過する前に、待ち解除条件が成立した場合

  この時のエラーコードは、E_OK を返します。

- 待ち解除条件が成立する前に、tmout の時間が経過した場合

  この時のエラーコードは、E_TMOUT を返します。

- 待ち解除条件が成立する前に rel_wai、irel_wai システムコールを発行した場合

  この時のエラーコードは、E_RLWAI を返します。

- 他のタスクから発行した del_mbf システムコールによって待ち状態の対象となっているメッセージバッファが削除された場合

  この時のエラーコードは、E_DLT を返します。

- 他のタスクから発行した vrst_mbf システムコールによって待ち状態の対象となっているメッセージバッファが初期化された場合

  この時のエラーコードは、EV_RST を返します。

　tmout には、-1 から 7FFFFFFFH まで指定できます。なお、tmout に TMO_POL=0 を指定した場合は、タイムアウト値として 0 を指定したことを示し、psnd_mbf と同じ動作をします。また、tmout=TMO_FEVR(-1)を指定した場合は、永久待ちの指定で、snd_mbf と同じ動作をします。
　存在しないメッセージバッファに対して、本システムコールを発行した場合、エラー E_NOEXS を返します。
　本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

[27] msgsz が、cre_mbf で指定した maxmsz よりも小さいかどうかはユーザ側でチェックしてください。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
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 Message Buffer)

【システムコール名】

psnd_mbf　→　メッセージを送信します(待ちなし)

【アセンブリ言語による呼び出し方法】

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

《引数》

mbfid	[**]	送信をおこなうメッセージバッファ ID 番号
msg	[****]	送信メッセージの先頭アドレス (R2 レジスタに先頭アドレスを設定してください)
msgsz	[****]	送信メッセージのサイズ

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	送信するメッセージバッファの ID 番号
R2	メッセージパケットの先頭アドレス
R3	送信メッセージのサイズ

【 C 言語による呼び出し方法】

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

《引数》

ID	mbfid;	メッセージバッファ ID 番号
VP	msg;	送信メッセージの先頭アドレス
INT	msgsz;	送信メッセージのサイズ

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000): 正常終了
E_TMOUT	0FFFFFFABH(-H'00000055): ポーリング失敗またはタイムアウト
E_NOEXS	0FFFFFFCCH(-H'00000034): オブジェクトが存在していない

【機能説明】

mbfid で示されたメッセージバッファに対して msg のアドレスに入っているメッセージを送信します。メッセージのサイズは、msgsz に指定してください。msg 以下の msgsz バイトが、mpfid で指定されたメッセージバッファのメッセージキューにコピーされます。メッセージキューは、リングバッファによって実現されます。

msgsz が、cre_mbf で指定した maxmsz よりも大きくなっても、エラーにはなりません[28]。

なお、バッファに空き領域がない場合は、システムコール発行タスクにエラーE_TMOUT を返して、本システムコールを終了します。snd_mbf、tsnd_mbf とは、異なり本システムコールを発行したタスクは、待ち状態には移行しません。

存在しないメッセージバッファに対して、本システムコールを発行した場合は、エラー

[28] msgsz が、cre_mbf で指定した maxmsz よりも小さいかどうかはユーザ側でチェックしてください。

E_NOEXS を返します。
　本システムコールはタスクからのみ発行しません。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.6.6. rcv_mbf(Receive MessageBuffer)

【システムコール名】

rcv_mbf　→　メッセージを受信します

【アセンブリ言語による呼び出し方法】

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

《引数》

mbfid	[**]	メッセージバッファ ID 番号
msg	[****]	受信メッセージを格納するアドレス (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	受信するメッセージバッファの ID 番号
R2	受信メッセージを格納するアドレス
R3	受信メッセージのサイズ

【 C 言語による呼び出し方法】

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

《引数》

VP	msg	受信メッセージを格納するアドレス
INT	*p_msgsz;	受信メッセージのサイズ
ID	mbfid;	メッセージバッファ ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。
p_msgsz が指す領域に、受信メッセージのサイズが設定されます。

【エラーコード】

E_OK	00000000H(-H'00000000) : 正常終了
E_RLWAI	0FFFFFFAAH(-H'00000056): 待ち状態強制解除
E_DLT	0FFFFFFAFH(-H'00000051): 待ちオブジェクトが削除された
E_NOEXS	0FFFFFFCCH(-H'00000034): オブジェクトが存在していない

【機能説明】

mbfid で示されたメッセージバッファからメッセージを受信し、msg で指定した領域にメッセージを格納します。対象となるメッセージバッファに対して、送信待ちタスクが存在する場合、受信したメッセージサイズと送信待ち行列の先頭にあるタスクの送信メッセージサイズとを比較します。

1. 受信メッセージサイズが送信メッセージサイズよりも大きい場合
   メッセージバッファにメッセージを送信し、送信待ちタスクを送信待ち状態から実行可能(READY)状態に移行します。
2. 受信メッセージサイズが送信メッセージサイズよりも小さい場合
   メッセージバッファにメッセージを送信せず、送信待ち状態のままで本システムコー

ルの処理を終了します。

1の処理が終了した後、まだ送信待ちタスクがある場合は、上記と同様の比較処理を行っていきます。また、mbfid で示されたメッセージバッファに、まだメッセージが送信されていない場合には、本システムコールを発行したタスクは受信待ち状態となり、FIFO 順で受信待ち行列につながれます。

rel_wai、irel_wai システムコールによって待ち状態が強制解除された場合には、エラーE_RLWAI が返されます。

他のタスクが発行した del_mbf システムコールによって条件成立を待っているメッセージバッファが削除された場合、待ち状態にあったタスクは、受信待ち状態が解除され、そのタスクにエラーE_DLT が返されます。

存在しないメッセージバッファに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.6.7. trcv_mbf(Receive Message Buffer with Timeout)

【システムコール名】

trcv_mbf　→　メッセージを受信します（タイムアウトあり）

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
trcv_mbf mbfid, tmout
```

《引数》

mbfid	[**]	メッセージバッファ ID 番号
tmout	[****]	タイムアウト値
msg	[****]	受信メッセージを格納するアドレス (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	受信するメッセージバッファの ID 番号
R2	受信メッセージを格納するアドレス
R3	受信メッセージのサイズ
R4	タイムアウト値

【 C 言語による呼び出し方法】

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

《引数》

VP	msg	受信メッセージを格納するアドレス
INT	*p_msgsz;	受信メッセージのサイズ
ID	mbfid;	メッセージバッファ ID 番号
TMO	tmout	タイムアウト値

《戻り値》

関数の戻り値としてエラーコードを返します。
p_msgsz が指す領域に、受信メッセージのサイズが設定されます。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_RLWAI	0FFFFFFAAH(-H'00000056):	待ち状態強制解除
E_DLT	0FFFFFFAFH(-H'00000051):	待ちオブジェクトが削除された
E_TMOUT	0FFFFFFABH(-H'00000055):	ポーリング失敗またはタイムアウト
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない

【機能説明】

mbfid で示されたメッセージバッファからメッセージを受信し、msg で指定した領域にメッセージを格納します。対象となるメッセージバッファに対して、送信メッセージバッファ待ちタスクが存在する場合、受信したメッセージサイズと送信待ち行列の先頭にあるタスクの送信メッセージサイズとを比較します。

1. 受信メッセージサイズが送信メッセージサイズよりも大きい場合
   メッセージバッファにメッセージを送信し、送信待ち状態から実行可能(READY)状態

に移行します。

2. 受信メッセージサイズが送信メッセージサイズよりも小さい場合

メッセージバッファにメッセージを送信せず、送信待ち状態のままで本システムコールの処理を終了します。

1の処理が終了した後、まだ送信待ちタスクがある場合は、上記と同様の比較処理を行っていきます。また、mbfid で示されたメッセージバッファにまだメッセージが送信されていない場合には、本システムコールを発行したタスクは受信待ち状態となり、メッセージ待ち行列とタイムアウト待ち行列の 2 つの待ち行列につながれます。

本システムコール実行による待ち状態は、以下に示す場合に解除されます。なお、待ち状態が解除されると本システムコールを発行したタスクは受信待ち行列とタイムアウト待ち行列の 2 つの待ち行列からはずされ、レディキューに接続されます。

- tmout の時間が経過する前にメッセージが到着した場合

  この時のエラーコードは、E_OK を返します。

- メッセージが到着しないまま、tmout の時間が経過した場合

  この時のエラーコードは、E_TMOUT を返します。

- 他のタスクおよびハンドラから発行した rel_wai、irel_wai システムコールによって待ち状態が強制解除された場合

  この時のエラーコードは、E_RLWAI を返します。

- 他のタスクから発行した del_mbf システムコールによって待ち状態の対象となっているメッセージバッファが削除された場合

  この時のエラーコードは、E_DLT を返します。

tmout には、-1 から 7FFFFFFFH まで指定できます。tmout に TMO_POL=0 を指定した場合は、タイムアウト値として 0 を指定したことを示し、prcv_mbf と同じ動作をします。また、tmout=TMO_FEVR(-1)にした場合は、永久待ちの指定で、rcv_mbf システムコールと同じ動作になります。

存在しないメッセージバッファに対して、本システムコールを発行した場合は、エラー E_NOEXS を返します。本システムコールはタスクからのみ発行することができます。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#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”);
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    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 Message Buffer)

【システムコール名】

prcv_mbf　→　メッセージを受信します（待ちなし）

【アセンブリ言語による呼び出し方法】

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

《引数》

mbfid	[**]	メッセージバッファの ID 番号
msg	[****]	受信メッセージを格納するアドレス (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	受信するメッセージバッファの ID 番号
R2	受信メッセージを格納するアドレス
R3	受信メッセージのサイズ
R4	タイムアウト値

【 C 言語による呼び出し方法】

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

《引数》

VP	msg	受信メッセージを格納するアドレス
INT	*p_msgsz;	受信メッセージのサイズ
ID	mbfid;	メッセージバッファ ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。
p_msgsz が指す領域に、受信メッセージのサイズが設定されます。

【エラーコード】

E_OK	00000000H(-H'00000000) : 正常終了
E_TMOUT	0FFFFFFABH(-H'00000055): ポーリング失敗またはタイムアウト
E_NOEXS	0FFFFFFCCH(-H'00000034): オブジェクトが存在していない

【機能説明】

mbfid で示されたメッセージバッファからメッセージを受信し、msg で指定した領域にメッセージを格納します。

対象となるメッセージバッファに対して、送信メッセージバッファ待ちタスクが存在する場合、受信したメッセージサイズと送信メッセージ待ち行列の先頭にあるタスクの送信メッセージサイズとを比較します。

1. 受信メッセージサイズが送信メッセージサイズよりも大きい場合
   メッセージバッファにメッセージを送信し、送信待ち状態から実行可能(READY)状態に移行します。
2. 受信メッセージサイズが送信メッセージサイズよりも小さい場合
   メッセージバッファにメッセージを送信せず、送信待ち状態のままで本システムコー

ルの処理を終了します。

1の処理が終了した後、まだ送信待ちタスクがある場合は、上記と同様の比較処理を行っていきます。また、mbfid で示されたメッセージバッファにまだメッセージが送信されていない場合には、エラーE_TMOUT を返します。rcv_mbf、trcv_mbf とは、異なり本システムコールを発行したタスクは、待ち状態には移行しません。

存在しないメッセージバッファに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスクからのみ発行することができます。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#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 )
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.6.9. ref_mbf(Refer MessageBuffer Status)

【システムコール名】

ref_mbf　→　メッセージバッファ状態を参照します

【アセンブリ言語による呼び出し方法】

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

《引数》

mbfid	[**]	状態を参照するメッセージバッファ ID 番号
pk_rmbf	[****]	メッセージバッファ状態を返すパケットの先頭アドレス (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	参照するメッセージバッファの ID 番号
R2	メッセージバッファの状態を返すパケットの先頭アドレス
R3	--

オフセット	サイズ		
+0	4	exinf	拡張情報
+4	2	wtsk	受信待ちタスクの有無
+6	2	stsk	送信待ちタスクの有無
+8	4	msgsz	次に受信されるメッセージパケットの 先頭アドレス
+12	4	frbufsz	空きバッファのサイズ

【 C 言語による呼び出し方法】

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

《引数》

T_RMBF	*pk_rmbf;	メッセージバッファ状態を返す構造体の先頭アドレス
ID	mbfid;	メッセージバッファ ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。
pk_rmbf の指す構造体には、以下の情報が返されます。

```
typedef struct t_rmbf {
    VP              exinf;   /*拡張情報*/
    BOOL_ID         wtsk;    /*受信待ちタスクの有無*/
    BOOL_ID         stsk;    /*送信待ちタスクの有無*/
    INT             msgsz;   /*次に受信されるメッセージのサイズ(バイト
数)*/
    INT             frbufsz; /*空きバッファのサイズ(バイト数)*/
} T_RMBF;
```

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない

【機能説明】

本システムコールでは mbfid で指定したメッセージバッファの以下の状態を返します。

●exinf

拡張情報を返します。

●wtsk

wtsk には、指定したメッセージバッファで受信メッセージを待っている先頭タスクの ID 番号を返します。受信メッセージを待っているタスクがない場合は、FALSE(0)を返します。

●stsk

stsk には、定したメッセージバッファで送信メッセージを待っている先頭タスクの ID 番号を返します。送信メッセージを待っているタスクがない場合は、FALSE(0)を返します。

●msgsz

指定したメッセージバッファに格納されている先頭メッセージのサイズが返ります。メッセージバッファにメッセージがない場合は、FALSE(0)を返します。

●frbufsz

指定したメッセージバッファの空き領域のサイズを返します。

存在しないメッセージバッファに対して、本システムコールを発行した場合は、エラー E_NOEXS を返します。

本システムコールはタスク、ハンドラのどちらからでも発行できます。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
    T_RMBF rmbf;
        :
    ref_mbf(&mbf, ID_mbf);
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
rmbf:    .RES.B 16
    .include mr32r.inc
    .global task
task:
        :
    ld24      R2,#rmbf
    ref_mbf    ID_mbf
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
rmbf:    .space 16
    .include mr32r.inc
    .global task
task:
        :
    ld24      R2,#rmbf
    ref_mbf    ID_mbf
        :
```

2.7. 拡張同期・通信機能(ランデブ)

2.7.1. cre_por(Create Port for Rendezvous)

【システムコール名】

cre_por　→　ランデブ用ポートを生成します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
cre_por  porid
```

《引数》

porid	[**]	生成するランデブ用ポート ID 番号
pk_cpor	[****]	ランデブ用ポートの生成情報 (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	生成するランデブ用ポートの ID 番号
R2	生成情報を返すパケットの先頭アドレス
R3	--

pk_cpor の指す領域には、以下に示す情報を設定してください。

オフセット	サイズ		
+0	4	exinf	拡張情報
+4	4	poratr	ポート属性
+8	4	maxcmsz	呼出時のメッセージの最大数
+12	4	maxrmsz	返答時のメッセージの最大数

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER cre_por (porid, pk_cpor);
```

《引数》

```
ID       porid;          生成するランデブ用ポート ID 番号
T_CPOR   *pk_cpor;       ランデブ用ポート生成情報を格納した先頭アドレス
```

pk_cpor の指す構造体に以下に示す情報を設定してください。

```
typedef struct t_cpor {
    VP      exinf;          /* 拡張情報 */
    ATR     poratr;         /* ポート属性 */
    INT     maxcmsz;        /* 呼出時のメッセージの最大数 */
    INT     maxrmsz;        /* 返答時のメッセージの最大数 */
} T_CPOR;
```

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK          00000000H(-H'00000000) :  正常終了
E_OBJ         0FFFFFFC1H(-H'0000003f):  オブジェクトの状態が不正
```

【機能説明】

porid で指定された ID 番号を持つランデブ用ポートを生成します。生成するランデブ用ポートの情報 pk_cpor について以下に示します。

●exinf(拡張情報)

生成するランデブ用ポートに関する情報を格納するためにユーザが自由に利用できる領域です。exinf の内容に関しては、関知しません。

●poratr(ポート属性)

ここで設定した値は、**MR32R** は関知しません。

●maxcmsz

呼出時に使用するメッセージの最大長を指定します。
なお、maxcmsz は、**MR32R** は参照しませんので、この指定は必要ありません。他のリアルタイム OS との互換性を保つ必要がある場合は、この値を設定してください。

●maxrmsz

返答時に使用するメッセージの最大長を指定します。
なお、maxrmsz は、**MR32R** は参照しませんので、この指定は必要ありません。他のリアルタイム OS との互換性を保つ必要がある場合は、この値を設定してください。

すでに存在するランデブ用ポートに対して、本システムコールを発行した場合は、エラー E_OBJ を返します。

本システムコールの指定可能な ID 番号の範囲は、1 から最大項目数定義で設定したユーザシステムの最大ランデブ用ポート数までです。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task1()
{
    T_CPOR setpor;
        :
    setmbf.maxcmsz = 300;
    setmbf.maxrmsz = 200;
    cre_por( 1, &setpor );
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .equ        ID_por1,1
setpor: .space 16
    .include “mr32r.inc”
    .global   task1
task1:
        :
    ld24    R2,#setmbf
    ld24    R1,#300
    st      R1,@(8,R2)
    ld24    R1,#200
    st      R1,@(12,R2)
    cre_por ID_por1
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .equ        ID_por1,1
setpor: .space 16
    .include “mr32r.inc”
    .global   task1
task1:
        :
    ld24    R2,#setmbf
    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)

【システムコール名】

del_por　→　ランデブ用ポートを削除します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
del_por  porid
```

《引数》

porid　　[**]　　削除するランデブ用ポートの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	削除するランデブ用ポートの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　　porid;　　削除するランデブ用ポート ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK          00000000H(-H’00000000) :  正常終了
E_NOEXS       0FFFFFFCCH(-H’00000034):  オブジェクトが存在していない
```

【機能説明】

porid で示されたランデブ用ポートを削除します。

削除されたランデブ用ポートは、再度、同じ ID 番号で新しいランデブ用ポートを生成できます。

削除するランデブ用ポートに対して、呼出待ちや受付待ちタスクがある場合でも、本システムコールは正常終了します。その時、待ち状態にあったタスクは、待ち状態から解除され、そのタスクにエラーE_DLT を返します。

存在しないランデブ用ポートに対して、本システムコールを発行した場合は、エラー E_NOEXS を返します。なお、本システムコールの発行は、cre_por システムコールによって生成したポートに対して行ってください。コンフィグレーションファイルで定義したポートに対して本システムコールを発行した場合は、正常に動作しません。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
#define ID_por2 2
void task1()
{
        :
    del_por( ID_por2 );
        :
    ext_tsk();
}
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .include “mr32r.inc”
    .global  task1
task1:
        :
    del_por ID_por2
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .include “mr32r.inc”
    .global  task1
task1:
        :
    del_por ID_por2
        :
    ext_tsk
```

2.7.3. cal_por(Call Port for Rendezvous)

【システムコール名】

cal_por　→　ランデブ呼出を行います

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
cal_por  porid, calptn, cmsgsz
```

《引数》

porid	[**]	ランデブ用ポートの ID 番号
msg	[****]	メッセージを格納したアドレス (R5 レジスタに先頭アドレスを設定してください)
calptn	[****]	呼出側選択条件を表わすビットパターン
cmsgsz	[****]	呼出メッセージのサイズ

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	ランデブ用ポートの ID 番号
R2	返答メッセージサイズ
R3	呼び出しメッセージのサイズ
R4	--
R5	メッセージを格納した領域の先頭アドレス
R6	呼出側選択条件を表わすビットパターン

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER cal_por (msg, p_rmsgsz, porid, calptn, cmsgsz);
```

《引数》

VP	msg;	呼出メッセージを格納したアドレス
INT	*p_rmsgsz;	返答メッセージのサイズを格納する領域のアドレス
ID	porid;	ランデブ用ポート ID 番号
UINT	calptn;	呼出側選択条件を表わすビットパターン
INT	cmsgsz	呼出メッセージのサイズ

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H’00000000) :	正常終了
E_RLWAI	0FFFFFFAAH(-H’00000056):	待ち状態強制解除
E_NOEXS	0FFFFFFCCH(-H’00000034):	オブジェクトが存在していない
E_DLT	0FFFFFFAFH(-H’00000051):	待ちオブジェクトが削除された

【機能説明】

ポートに対してランデブ呼出を行います。

porid で指定したポートにおいてランデブ受付待ち状態(acp_por あるいは tacp_por による待ち)のタスクがあり、その受付待ち状態のタスクと本システムコールを発行するタスクと

の間でランデブ成立条件が満たされた場合は、ランデブ成立となります。ランデブ成立とは、呼出側タスクの calptn と受付側タスクの acpptn の論理積が 0 かどうかによって決まります。0 以外の場合はランデブ成立となります。ランデブ成立時、ランデブ受付待ちタスクの状態は、待ち状態から実行可能(READY)状態へ移行し、本システムコールを発行したタスクは、ランデブ終了待ち状態となります。ランデブ終了待ち状態となったタスクは、ランデブ受付タスクが rpl_rdv を実行することにより待ち状態が解除されます。この時点で cal_por システムコールが終了します。

ランデブが成立した時、呼出側タスクは受付側タスクにメッセージを送信します。msg のアドレスに格納されているメッセージを cmsgsz バイトだけ受付側タスクの acp_por で指定したメッセージ格納領域にコピーします。また、ランデブ終了時には、受付側タスクから呼出側タスクに返答メッセージを送信することができます。受付側タスクが、rpl_rdv で指定した返答メッセージの内容が呼出側タスクの cal_por で指定した msg が示すメッセージ格納領域にコピーされます。

ランデブ呼出とランデブ受付を行う 2 つのタスク間でのメッセージの交換は、双方のタスクが持っているメッセージを互いにコピーしあうことになります。そのため、以前に持っていたメッセージは破壊されます。porid で指定したポートに受付待ちタスクがなかった場合や、受付待ちタスクがあってもランデブ成立条件が満たされなかった場合(論理積の結果が 0 の場合)、本システムコールを発行したタスクは、このポートの呼出待ち状態となり、呼出待ち行列につながれます。

本システムコール発行による待ち状態は、以下に示す場合に解除されます。なお、待ち状態が解除されると本システムコールを発行したタスクは、呼出待ち待ち行列からはずされ、レディキューに接続されます。

- 待ち解除条件が成立する前に rel_wai、irel_wai システムコールを発行した場合

  この時のエラーコードは、E_RLWAI を返します。

- 他のタスクから発行した del_por システムコールによって待ち状態の対象となっているポートが削除された場合

  この時のエラーコードは、E_DLT を返します。

本システムコールの引数 calptn には、0 を指定することはできませんが、0 を指定してもエラーにはなりません。

存在しないポートに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task(void)
{
    char msg[128];
    INT   rmsgsz;
        :
    cal_por((VP)msg, &rmsgsz, ID_por1, 0x3, 10);
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global   task
msg:   .RES.B   128
task:
    ld24        R5,#msg
    cal_por ID_por1,H’3,10
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .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)

【システムコール名】

tcal_por　→　ランデブ呼出を行います

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
tcal_por porid, calptn, cmsgsz, tmout
```

《引数》

porid	[**]	ランデブ用ポート ID 番号
msg	[****]	メッセージを格納したアドレス (R5 レジスタに先頭アドレスを設定してください)
calptn	[****]	呼出側選択条件を表わすビットパターン
cmsgsz	[****]	呼出メッセージのサイズ
tmout	[****]	タイムアウト値

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	ランデブ用ポートの ID 番号
R2	返答メッセージサイズ
R3	呼び出しメッセージのサイズ
R4	タイムアウト値
R5	メッセージを格納した領域の先頭アドレス
R6	呼出側選択条件を表わすビットパターン

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER tcal_por (msg, p_rmsgsz, porid, calptn, cmsgsz, tmout );
```

《引数》

VP	msg;	呼出メッセージを格納したアドレス
INT	*p_rmsgsz;	返答メッセージのサイズを格納する領域のアドレス
ID	porid;	ランデブ用ポート ID 番号
UINT	calptn;	呼出側選択条件を表わすビットパターン
INT	cmsgsz;	呼出メッセージのサイズ
TMO	tmout;	タイムアウト値

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_RLWAI	0FFFFFFAAH(-H'00000056):	待ち状態強制解除
E_TMOUT	0FFFFFFABH(-H'00000055):	ポーリング失敗またはタイムアウト
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない
E_DLT	0FFFFFFAFH(-H'00000051):	待ちオブジェクトが削除された

【機能説明】

ポートに対してランデブ呼出を行います。

porid で指定したポートにおいてランデブ受付待ち状態(acp_por あるいは tacp_por による待ち)のタスクがあり、その受付待ち状態のタスクと本システムコールを発行するタスクとの間でランデブ成立条件が満たされた場合は、ランデブ成立となります。ランデブ成立とは、呼出側タスクの calptn と受付側タスクの acpptn の論理積が 0 かどうかによって決まります。0 以外の場合はランデブ成立となります。

ランデブ成立時、ランデブ受付待ちタスクの状態は、待ち状態から実行可能(READY)状態へ移行し、本システムコールを発行したタスクは、ランデブ終了待ち状態となります。ランデブ終了待ち状態となったタスクは、ランデブ受付タスクが rpl_rdv を実行することにより待ち状態が解除されます。この時点で tcal_por システムコールが終了します。ランデブが成立した時、呼出側タスクは受付側タスクにメッセージを送信します。msg のアドレスに格納されているメッセージを cmsgsz バイトだけ受付側タスクが acp_por で指定したメッセージ格納領域にコピーします。また、ランデブ終了時には、受付側タスクから呼出側タスクに返答メッセージを送信することができます。受付側タスクが、rpl_rdv で指定した返答メッセージの内容が呼出側タスクの cal_por で指定した msg が示すメッセージ格納領域にコピーされます。

ランデブ呼出とランデブ受付を行う 2 つのタスク間でのメッセージの交換は、双方のタスクが持っているメッセージを互いにコピーしあうことになります。そのため、以前に持っていたメッセージは破壊されます。

porid で指定したポートに受付待ちタスクがなかった場合や、受付待ちタスクがあってもランデブ成立条件が満たされなかった場合(論理積の結果が 0 の場合)、本システムコールを発行したタスクは、このポートの呼出側待ち状態となり、呼出待ち行列とタイムアウト待ち行列の 2 つの待ち行列につながれます。待ち行列の順序は、FIFO です。

本システムコール発行による待ち状態は、以下に示す場合に解除されます。なお、待ち状態が解除されると本システムコールを発行したタスクは、呼出待ち行列とタイムアウト待ち行列の 2 つの待ち行列がはずされ、レディキューに接続されます。

- tmout の時間が経過する前に、待ち解除条件が成立した場合

  この時のエラーコードは、E_OK を返します。

- 待ち解除条件が成立する前に、tmout の時間が経過した場合

  この時のエラーコードは、E_TMOUT を返します。

- 待ち解除条件が成立する前に rel_wai、irel_wai システムコールを発行した場合

  この時のエラーコードは、E_RLWAI を返します。

- 他のタスクから発行した del_por システムコールによって待ち状態の対象となっているメッセージバッファが削除された場合

  この時のエラーコードは、E_DLT を返します。

tmout には、-1 から 7FFFFFFFH まで指定できます。なお、tmout に TMO_POL=0 を指定した場合は、タイムアウト値として 0 を指定したことを示し、pcal_por と同じ動作をします。また、tmout=TMO_FEVR(-1)を指定した場合は、永久待ちの指定で、cal_por と同じ動作をします。

存在しないポートに対して、本システムコールを発行した場合、エラーE_NOEXS を返します。本システムコールの引数 calptn には 0 を指定することはできませんが、0 を指定してもエラーにはなりません。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#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);
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
msg:  .RES.B  128
    .equ    ID_por1,1
    .include “mr32r.inc”
    .global  task
task:
        :
    ld24       R5,#msg
    tcal_por ID_por1,0x3,10,300
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
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)

【システムコール名】

pcal_por　→　ランデブ呼出を行います

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
pcal_por porid, calptn, cmsgsz
```

《引数》

porid	[**]	ランデブ用ポートの ID 番号
msg	[****]	メッセージを格納したアドレス (R5 レジスタに先頭アドレスを設定してください)
calptn	[****]	呼出側選択条件を表わすビットパターン
cmsgsz	[****]	呼出メッセージのサイズ

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	ランデブ用ポートの ID 番号
R2	返答メッセージサイズ
R3	呼び出しメッセージのサイズ
R4	--
R5	メッセージを格納した領域の先頭アドレス
R6	呼出側選択条件を表わすビットパターン

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER pcal_por (msg, p_rmsgsz, porid, calptn, cmsgsz);
```

《引数》

VP	msg;	呼出メッセージを格納したアドレス
INT	*p_rmsgsz;	返答メッセージのサイズを格納する領域のアドレス
ID	porid;	ランデブ用ポート ID 番号
UINT	calptn;	呼出側選択条件を表わすビットパターン
INT	cmsgsz;	呼出メッセージのサイズ

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_TMOUT	0FFFFFFABH(-H'00000055):	ポーリング失敗またはタイムアウト
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない

【機能説明】

ポートに対してランデブ呼出を行います。

porid で指定したポートにおいてランデブ受付待ち状態のタスクがあり、その受付待ち状態のタスクと本システムコールを発行するタスクとの間でランデブ成立条件が満たされた場合は、ランデブ成立となります。ランデブ成立とは、呼出側タスクの calptn と受付側タスクの acpptn の論理積が 0 かどうかによって決まります。0 以外の場合はランデブ成立となります。

ランデブ成立時、ランデブ受付待ちタスクの状態は、待ち状態から実行可能(READY)状態へ移行し、本システムコールを発行したタスクは、ランデブ終了待ち状態となります。ランデブ終了待ち状態となったタスクは、ランデブ受付タスクが rpl_rdv を実行することにより待ち状態が解除されます。この時点で pcal_por システムコールが終了します。

ランデブが成立した時、呼出側タスクは受付側タスクにメッセージを送信します。msg のアドレスに格納されているメッセージを cmsgsz バイトだけ受付側タスクが acp_por で指定したメッセージ格納領域にコピーします。また、ランデブ終了時には、受付側タスクから呼出側タスクに返答メッセージを送信することができます。受付側タスクが、rpl_rdv で指定した返答メッセージの内容が呼出側タスクが cal_por で指定した msg が示すメッセージ格納領域にコピーされます。

ランデブ呼出とランデブ受付を行う 2 つのタスク間でのメッセージの交換は、双方のタスクが持っているメッセージを互いにコピーしあうことになります。そのため、以前に持っていたメッセージは破壊されます。

porid で指定したポートに受付待ちタスクがなかった場合や、受付待ちタスクがあってもランデブ成立条件が満たされなかった場合(論理積の結果が 0 の場合)、本システムコールを発行したタスクには、エラーE_TMOUT を返し、本システムコールを終了します。cal_por、tcal_por とは、異なり本システムコールを発行したタスクは、待ち状態には移行しません。

存在しないポートに対して、本システムコールを発行した場合、エラーE_NOEXS を返します。

本システムコールの引数 calptn には 0 を指定することはできませんが、0 を指定してもエラーにはなりません。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
#define ID_por1 1
void task()
{
    char msg[128];
    INT  rmsgsz;
        :
    pcal_por((VP)msg, &rmsgsz, ID_por1, H’3, 10);
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
msg:  .RES.B  128
    .equ    ID_por1,1
    .include “mr32r.inc”
    .global  task
task:
        :
    ld24       R5,#msg
    pcal_por ID_por1,H’3,10
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
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)

【システムコール名】

acp_por　→　ランデブ受付を行います

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
acp_por  porid, acpptn
```

《引数》

porid	[**]	ランデブ用ポートの ID 番号
acpptn	[****]	受付側選択条件を表わすビットパターン
msg	[****]	呼出メッセージを格納する領域の先頭アドレス (R5 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	ランデブ用ポートの ID 番号
R2	呼出メッセージサイズ
R3	ランデブ番号
R4	--
R5	呼出メッセージを格納する領域の先頭アドレス
R6	受付側選択条件を表わすビットパターン

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER acp_por (p_rdvno, msg, p_cmsgsz, porid, acpptn );
```

《引数》

RNO	*p_rdvno;	ランデブ番号
VP	msg;	呼出メッセージを格納する領域の先頭アドレス
INT	*p_cmsgsz;	呼出メッセージのサイズ
ID	porid;	ランデブ用ポート ID 番号
UINT	acpptn;	受付側選択条件を表わすビットパターン

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_RLWAI	0FFFFFFAAH(-H'00000056):	待ち状態強制解除
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない
E_DLT	0FFFFFFAFH(-H'00000051):	待ちオブジェクトが削除された

【機能説明】

ポートに対してランデブ受付を行います。

porid で指定したポートにおいてランデブ呼出待ち状態(cal_por あるいは tcal_por による待ち)のタスクがあり、その呼出待ち行列の中のタスクと本システムコールを発行するタスクとの間でランデブ成立条件が満たされた場合は、ランデブ成立となります。ランデブ成立

とは、呼出側タスクの calptn と受付側タスクの acpptn の論理積が 0 かどうかによって決まります。0 以外の場合はランデブ成立となります。ランデブ成立時、ランデブ呼出待ち行列中のタスクは、待ち行列から外れ、ランデブ呼出待ちの状態からランデブ終了待ち状態に変わります。

porid で指定したポートに呼出待ちタスクがなかった場合や、呼出待ちタスクがあってもランデブ成立条件が満たされなかった場合、本システムコールを発行したタスクは、このポートの受付待ち状態となり受付待ち行列につながれます。

本システムコール発行による待ち状態は、以下に示す場合に解除されます。なお、待ち状態が解除されると本システムコールを発行したタスクは、受付待ち行列からはずされ、レディキューに接続されます。

- 待ち解除条件が成立する前に rel_wai、irel_wai システムコールを発行した場合

  この時のエラーコードは、E_RLWAI を返します。

- 他のタスクから発行した del_por システムコールによって待ち状態の対象となっているポートが削除された場合

  この時のエラーコードは、E_DLT を返します。

ランデブ受付側タスクは、ランデブ返答(成立したランデブに関する rpl_rdv を発行)を行う前に、再度、本システムコールを発行することができます。こうすることで、同時に複数のランデブを行うことができます。このときのポートは、前と同じであっても異なっていても構いません。

ランデブが成立した時、呼出側タスクは受付側タスクに呼出メッセージを送信します。cal_por で指定した msg のアドレスに格納されているメッセージを cmsgsz バイトだけ受付側タスクが acp_por で指定したメッセージ格納領域にコピーします。この時、呼出メッセージのサイズ cmsgsz は、acp_por の戻り値となります。また、ランデブ終了時には、受付側タスクから呼出側タスクに返答メッセージを送信することができます。受付側タスクが、rpl_rdv で指定した返答メッセージの内容を呼出側タスクの cal_por で指定した msg が示すメッセージ格納領域にコピーします。

rdvno は、同時に成立しているランデブを区別するための情報です。ランデブ終了時に、rpl_rdv のパラメータやランデブ回送時の fwd_por のパラメータに使用されます。この情報には、ランデブ成立相手の呼出側タスクを指定するための情報が含まれています。

本システムコールの引数 acpptn には 0 を指定することはできませんが。acpptn に、0 を指定してもエラーにはなりません。

存在しないポートに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
char msg[128];
RNO rdvno;
INT cmsgsz;
        :
acp_por(&rdvno, msg, &cmsgsz, ID_por1, 0x3);
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
msg:   .RES.B   128
     .include “mr32r.inc”
     .global   task
task:
        :
     ld24        R5,#msg
     acp_por ID_por1, H’3
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
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)

【システムコール名】

tacp_por　→　ランデブの受付を行います（タイムアウトあり）

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
tacp_por porid, acpptn, tmout
```

《引数》

porid	[**]	ランデブ用ポートの ID 番号
acpptn	[****]	受付側選択条件を表わすビットパターン
msg	[****]	呼出メッセージを格納したアドレス (R5 レジスタに先頭アドレスを設定してください)
tmout	[****]	タイムアウト値

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	ランデブ用ポートの ID 番号
R2	呼出メッセージサイズ
R3	ランデブ番号
R4	タイムアウト値
R5	呼出メッセージを格納する領域の先頭アドレス
R6	受付側選択条件を表わすビットパターン

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER tacp_por (p_rdvno, msg, p_cmsgsz, porid, acpptn, tmout);
```

《引数》

RNO	*p_rdvno;	ランデブ番号
VP	msg;	呼出メッセージを格納する領域のアドレス
INT	*p_cmsgsz;	呼出メッセージのサイズ
ID	porid;	ランデブ用ポート ID 番号
UINT	acpptn;	受付側選択条件を表わすビットパターン
TMO	tmout;	タイムアウト値

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_RLWAI	0FFFFFFAAH(-H'00000056):	待ち状態強制解除
E_TMOUT	0FFFFFFABH(-H'00000055):	ポーリング失敗またはタイムアウト
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない
E_DLT	0FFFFFFAFH(-H'00000051):	待ちオブジェクトが削除された

【機能説明】

ポートに対してランデブ受付を行います。

porid で指定したポートにおいてランデブ呼出待ち状態(cal_por あるいは tcal_por による待ち)のタスクがあり、その呼出待ち行列の中のタスクと本システムコールを発行するタスクとの間でランデブ成立条件が満たされた場合は、ランデブ成立となります。

porid で指定したポートに呼出待ちタスクがなかった場合や、呼出待ちタスクがあってもランデブ成立条件が満たされなかった場合、本システムコールを発行したタスクは、このポートの受付待ち状態となり受付待ち行列とタイムアウト待ち行列の 2 つの待ち行列につながれます。

ランデブ成立とは、呼出側タスクの calptn と受付側タスクの acpptn の論理積が 0 かどうかによって決まります。0 以外の場合はランデブ成立となります。ランデブ成立時、ランデブ呼出待ち行列中のタスクは、待ち行列から外れ、ランデブ呼出待ちの状態からランデブ終了待ち状態に変わります。

本システムコール発行による待ち状態は、以下に示す場合に解除されます。なお、待ち状態が解除されると本システムコールを発行したタスクは、受付待ち行列とタイムアウト待ち行列の 2 つの待ち行列からからはずされ、レディキューに接続されます。

- tmout の時間が経過する前に、待ち解除条件が成立した場合

  この時のエラーコードは、E_OK を返します。

- 待ち解除条件が成立する前に、tmout の時間が経過した場合

  この時のエラーコードは、E_TMOUT を返します。

- 待ち解除条件が成立する前に rel_wai、irel_wai システムコールを発行した場合

  この時のエラーコードは、E_RLWAI を返します。

- 他のタスクから発行した del_por システムコールによって待ち状態の対象となっているポートが削除された場合

  この時のエラーコードは、E_DLT を返します。

ランデブ受付側タスクは、ランデブ返答(成立したランデブに関する rpl_rdv を発行)を行う前に、再度、本システムコールを発行することができます。こうすることで、同時に複数のランデブを行うことができます。このときのポートは、前と同じであっても異なっていても構いません。

ランデブが成立した時、呼出側タスクは受付側タスクに呼出メッセージを送信します。cal_por で指定した msg のアドレスに格納されているメッセージを cmsgsz バイトだけ受付側タスクが acp_por で指定したメッセージ格納領域にコピーします。この時、呼出メッセージのサイズ cmsgsz は、acp_por の戻り値となります。また、ランデブ終了時には、受付側タスクから呼出側タスクに返答メッセージを送信します。受付側タスクが、rpl_rdv で指定した返答メッセージの内容が呼出側タスクが cal_por で指定した msg が示すメッセージ格納領域にコピーされます。

rdvno は、同時に成立しているランデブを区別するための情報です。ランデブ終了時に、rpl_rdv のパラメータやランデブ回送時の fwd_por のパラメータに使用されます。この情報には、ランデブ成立相手の呼出側タスクを指定するための情報が含まれています。

tmout には、-1 から 7FFFFFFFH まで指定できます。なお、tmout に TMO_POL=0 を指定した場合は、タイムアウト値として 0 を指定したことを示し、pacp_por と同じ動作をします。また、tmout=TMO_FEVR(-1)を指定した場合は、永久待ちの指定で、acp_por と同じ動作をします。

本システムコールの引数 acpptn には、0 を指定することはできませんが、acpptn に 0 を指定してもエラーにはなりません。

存在しないポートに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#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);
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
msg:  .RES.B  128
    .include “mr32r.inc”
    .global  task
task:
        :
    ld24       R5,#msg
    tacp_por ID_por1, H’3, 300
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
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)

【システムコール名】

pacp_por　→　ランデブの受付を行います

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
pacp_por porid, acpptn
```

《引数》

porid	[**]	ランデブ用ポートの ID 番号
acpptn	[****]	受付側選択条件を表わすビットパターン
msg	[****]	呼出メッセージを格納したアドレス (R5 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	ランデブ用ポートの ID 番号
R2	呼出メッセージサイズ
R3	ランデブ番号
R4	--
R5	呼出メッセージを格納する領域の先頭アドレス
R6	受付側選択条件を表わすビットパターン

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER pacp_por (p_rdvno, msg, p_cmsgsz, porid, acpptn);
```

《引数》

RNO	*p_rdvno;	ランデブ番号
VP	msg;	呼出メッセージを格納する領域のアドレス
INT	*p_cmsgsz;	呼出メッセージのサイズ
ID	porid;	ランデブ用ポート ID 番号
UINT	acpptn;	受付側選択条件を表わすビットパターン

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_TMOUT	0FFFFFFABH(-H'00000055):	ポーリング失敗またはタイムアウト
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない

【機能説明】

ポートに対してランデブ受付を行います。

porid で指定したポートにおいてランデブ呼出待ち状態のタスクがあり、その呼出待ち行列の中のタスクと本システムコールを発行するタスクとの間でランデブ成立条件が満たされた場合は、ランデブ成立となります。

ランデブ成立とは、呼出側タスクの calptn と受付側タスクの acpptn の論理積が 0 かどう

かによって決まります。0 以外の場合はランデブ成立となります。ランデブ成立時、ランデブ呼出待ち行列中のタスクは、待ち行列から外れ、ランデブ呼出待ちの状態からランデブ終了待ち状態に変わります。

porid で指定したポートに呼出待ちタスクがなかった場合や、呼出待ちタスクがあってもランデブ成立条件が満たされなかった場合、本システムコールを発行したタスクには、エラーE_TMOUT を返し、本システムコールを終了します。acp_por、tacp_por とは、異なり本システムコールを発行したタスクは、待ち状態には移行しません。

ランデブ受付側タスクは、ランデブ返答(成立したランデブに関する rpl_rdv を発行)を行う前に、再度、本システムコールを発行することができます。こうすることで、同時に複数のランデブを行うことができます。このときのポートは、前と同じであっても異なっていても構いません。ランデブが成立した時、呼出側タスクは受付側タスクに呼出メッセージを送信することができます。cal_por で指定した msg のアドレスに格納されているメッセージを cmsgsz バイトだけ受付側タスクが acp_por で指定したメッセージ格納領域にコピーします。この時、呼出メッセージのサイズ cmsgsz は、acp_por の戻り値となります。また、ランデブ終了時には、受付側タスクから呼出側タスクに返答メッセージを送信することができます。受付側タスクが、rpl_rdv で指定した返答メッセージの内容が呼出側タスクが cal_por で指定した msg が示すメッセージ格納領域にコピーされます。

rdvno は、同時に成立しているランデブを区別するための情報です。ランデブ終了時に、rpl_rdv のパラメータやランデブ回送時の fwd_por のパラメータに使用されます。この情報には、ランデブ成立相手の呼出側タスクを指定するための情報が含まれています。

本システムコールの引数 acpptn には、0 を指定することはできませんが、acpptn に、0 を指定してもエラーにはなりません。

存在しないポートに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
    char msg[128];
    RNO rdvno;
    INT cmsgsz;
        :
    pacp_por(&rdvno, msg, &cmsgsz, ID_por1, 0x3);
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
msg:  .RES.B  128
    .include “mr32r.inc”
    .global  task
task:
        :
    ld24       R5,#msg
    pacp_por ID_por1, 0x3
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
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)

【システムコール名】

fwd_por　→　ランデブを回送します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
fwd_por  porid, calptn, rdvno, cmsgsz
```

《引数》

porid	[**]	回送先ポートの ID 番号
calptn	[****]	呼出側選択条件を表わすビットパターン
rdvno	[****]	ランデブ番号 (R2 レジスタにランデブ番号を設定してください)
msg	[****]	呼出メッセージを格納する領域のアドレス (R5 レジスタに先頭アドレスを設定してください)
cmsgsz	[****]	呼出メッセージのサイズ

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	回送先ポートの ID 番号
R2	ランデブ番号
R3	呼出メッセージのサイズ
R4	--
R5	呼出メッセージを格納する領域の先頭アドレス
R6	呼出側選択条件を表わすビットパターン

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER fwd_por (porid, calptn, rdvno, msg, cmsgsz);
```

《引数》

ID	porid;	回送先ポートの ID 番号
UINT	calptn;	呼出側選択条件を表わすビットパターン
RNO	rdvno	ランデブ番号
VP	msg	呼出メッセージを格納するアドレス
INT	cmsgsz	呼出メッセージのサイズ

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_RLWAI	0FFFFFFAAH(-H'00000056):	待ち状態強制解除
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない
E_OBJ	0FFFFFFC1H(-H'0000003f):	オブジェクトの状態が不正

【機能説明】

ランデブ番号 rdvno で、受け付けたランデブを porid で指定した別のポートへ回送します。本システムコールは、現在、ランデブ中の状態(acp_por を実行した後)にあるタスクからのみ発行することができます。ランデブ番号 rdvno で指定した呼出タスクのランデブ成立を解除し、porid のポートに対して、呼出タスクをランデブ呼出待ち状態に戻します。つまり、呼出タスクの状態は、ランデブ終了待ち状態からランデブ呼出状態に戻ることになります。受付待ちタスクがある場合は、本システムコールで指定した calptn と受付側タスクの acpptn との論理積がとられ 0 でなければランデブが成立となります。ランデブが成立したタスクは、本システムコールで指定した msg のアドレスに格納されているメッセージを cmsgsz バイト分、受付側タスクにコピーし、ランデブ終了待ち状態になります。

受付待ちタスクがなかったり、回送先のポートでランデブが成立しなかった場合は、呼出側タスクは、呼出待ち状態になります。

本システムコールを発行したタスクは、回送したポートでの呼出タスクのランデブ状態に関係なく待ち状態にはならずそのまま実行を継続します。

本システムコールの引数に以下に示す場合を指定しても、エラーにはなりません。以下に示すチェックは、ユーザ側で行ってください。

- cmsgsz が、回送後のポートの送信最大メッセージサイズ maxcmsz を超えた場合
- cmsgsz が、回送前のポートの受信最大メッセージサイズ maxrmsz を超えた場合
- cmsgsz に、0 を指定した場合
- calptn に、0 を指定した場合

rdvno が不正な場合は、エラーE_OBJ を返します。

存在しないポートに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスクからのみ発行することができます。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
    char *msg=”abcdef”;
    RNO rdvno;
    INT cmsgsz;
                :
    fwd_por(ID_porid, 0x3, rdvno, (VP)msg, 6);
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
msg: .SDATA “abcdef”
rdvno: .RES.B 4
    .include “mr32r.inc”
    .global  task
task:
        :
        ld24    R5,#rdvno
        ld      R2,@R5
        ld24    R5,#msg
fwd_por ID_porid, H’3, rdvno, 6
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
msg: .byte “abcdef”
rdvno: .space 4
    .include “mr32r.inc”
    .global  task
task:
        :
    ld24        R5,#rdvno
    ld          R2,@R5
    ld24        R5,#msg
    fwd_por ID_porid, 0x3, rdvno, 6
        :
```

2.7.10. rpl_rdv(Reply Rendezvous)

【システムコール名】

rpl_rdv　→　ランデブ返答を行います

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
rpl_rdv  rmsgsz
```

《引数》

rdvno	[****]	ランデブ番号 (R2 レジスタにランデブ番号を設定してください)
msg	[****]	返答メッセージを格納する領域のアドレス (R2 レジスタに先頭アドレスを設定してください)
rmsgsz	[****]	返答メッセージのサイズ

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	ランデブ番号
R2	返答メッセージを格納する領域のアドレス
R3	返答メッセージのサイズ

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER rpl_rdv (rdvno, msg, rmsgsz);
```

《引数》

RNO	rdvno	ランデブ番号
VP	msg	返答メッセージを格納するアドレス
INT	rmsgsz	返答メッセージのサイズ

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) : 正常終了
E_OBJ	0FFFFFFC1H(-H'0000003f): オブジェクトの状態が不正

【機能説明】

呼出側タスクに対して返答を返し、ランデブを終了します。

本システムコールは、現在、ランデブ中の状態(acp_por を実行した後)にあるタスクからのみ発行することができます。呼出側タスクは、ランデブ終了待ち状態が解除され、msg のアドレスに格納されているメッセージを rmsgsz バイトだけ呼出側タスクの cal_por で指定した msg アドレス以下にコピーされます。呼出側タスクの状態は、ランデブ終了待ち状態から実行可能(READY)状態に移行します。本システムコールの引数に以下に示す場合を指定しても、エラーにはなりません。以下に示すチェックは、ユーザ側で行ってください。

- rmsgsz に 0 以下の値を指定した場合
- rmsgsz が、返答最大メッセージサイズ maxrmsz を超えた場合

ランデブ成立後に呼出側タスクが何らかの理由で、ランデブ終了前(rpl_rdv 実行前)に異常終了したような場合、ランデブ受付側のタスクは直接それを知ることができません。この場合、ランデブ受付側のタスクには、本システムコールからエラーE_OBJ が返されます。

本システムコールは、タスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
    char *msg=”abcdef”;
    RNO rdvno;
    INT rmsgsz;
        :
    rpl_rdv(rdvno, (VP)msg, 6);
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
msg: .SDATA “abcdef”
    .include “mr32r.inc”
    .global  task
task:
        :
    ld24        R5,#rdvno
    ld          R2,@R5
    ld24        R5,#msg
    rpl_rdv rdvno, 6
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
msg: .byte “abcdef”
    .include “mr32r.inc”
    .global  task
task:
        :
    ld24        R5,#rdvno
    ld          R2,@R5
    ld24        R5,#msg
    rpl_rdv rdvno, 6
        :
```

2.7.11. ref_por(Refer Port Status)

【システムコール名】

ref_por　→　ランデブ用ポート状態を参照します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
ref_por  porid
```

《引数》

porid	[**]	状態を参照するポート ID 番号
pk_rpor	[****]	ポート状態を返すパケットの先頭アドレス (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	参照するランデブ用ポートの ID 番号
R2	ランデブ用ポートの状態を返すパケットの先頭アドレス
R3	--

pk_rpor の指す領域には、以下の情報が返されます。

オフセット	サイズ		
+0	4	exinf	拡張情報
+4	2	wtsk	呼出待ちタスクの有無
+6	2	atsk	受付待ちタスクの有無

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER ref_por (pk_rpor, porid);
```

《引数》

T_RMBF	*pk_rpor;	ポートの状態を返す構造体の先頭アドレス
ID	porid;	ランデブ用ポート ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します 。

pk_rmbf の指す構造体には、以下の情報が返されます。

```
typedef struct t_rpor{
    VP              exinf;   /*拡張情報*/
    BOOL_ID         wtsk;    /*受付待ちタスクの有無*/
    BOOL_ID         atsk;    /*呼出待ちタスクの有無*/
} T_RPOR
```

【エラーコード】

E_OK	00000000H(-H'00000000) : 正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034): オブジェクトが存在していない

【機能説明】

本システムコールでは、porid で指定したポートの以下の状態を返します。

●exinf

拡張情報を返します。

●wtsk

wtsk には、対象ポートの呼出待ち行列の先頭タスク ID 番号を返します。呼出待ち行列にタスクがない場合は、FALSE(0)を返します。

●atsk

atsk には、対象ポートの受付待ち行列の先頭タスク ID 番号を返します。受付待ち行列にタスクがない場合は、FALSE(0)を返します。

存在しないポートに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスク、ハンドラのどちらからでも発行できます。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task(void)
{
    T_RPOR rpor;
        :
    ref_por(&rpor,ID_por);
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
rpor:    .RES.B 8
    .include mr32r.inc
    .global task
task:
        :
    ld24      R2,#rpor
    ref_por    ID_por
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
rpor:    .space 8
    .include mr32r.inc
    .global task
task:
        :
    ld24      R2,#rpor
    ref_por    ID_por
        :
```

2.8. 割り込み管理機能

2.8.1. def_int(Define Interrupt Handler)

【システムコール名】

def_int　→　割り込みハンドラを定義します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
def_int  dintno
```

《引数》

dintno	[****]	割り込みベクタ番号
pk_dint	[****]	割り込みハンドラ定義情報を格納した先頭アドレス (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	定義する割り込みベクタ番号
R2	割り込みハンドラ定義情報を格納した領域の先頭アドレス
R3	--

pk_dint の指す領域には、以下に示す情報を設定してください。

オフセット	サイズ		
+0	4	intatr	割り込みハンドラ属性
+4	4	inthdr	割り込みハンドラアドレス

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER def_int (dintno, pk_dint);
```

《引数》

```
UINT     dintno;          定義する割り込みベクタ番号
T_DINT   *pk_dint;        割り込み定義情報
```

pk_dint の指す構造体に以下に示す情報を設定してください。

```
typedef struct t_dint {
    ATR      intatr;          /* 割り込みハンドラ属性 */
    FP       inthdr;          /* 割り込みハンドラアドレス */
} T_DINT;
```

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK　　00000000H(-H'00000000):　正常終了

【機能説明】

dintno で示される割り込み番号に割り込みハンドラを定義し、割り込みハンドラを使用可能にします。すなわち、dintno で示される割り込みベクタの領域に割り込みエントリアドレスを設定します。

- intatr(割り込みハンドラ属性)

  ここで設定した値は、**MR32R** は関知しません。

- inthdr

  定義する割り込みハンドラのエントリアドレスをこの領域に設定してください。
  pk_dint.inthdr=NADR=(-1)とした場合には、前に定義した割り込みハンドラの定義解除を行います。

すでに定義済みの割り込み番号に対して割り込みハンドラを再定義することも可能です。再定義の場合、あらかじめ定義の解除を行う必要はありません。既に定義されている割り込み番号に対して新しいハンドラを再定義してもエラーにはなりません。

本システムコールは、割り込みベクタテーブルを RAM 上に割り当てないと使用できません。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void intr(void);
void task1()
{
    T_DINT setint;
                :
    setint.inthdr = intr; /* 割り込みハンドラのエントリアドレス設定 */
    def_int( 15, &setint );
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
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
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
setint:
    .space 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)

【システムコール名】

ret_int　→　割り込みハンドラから復帰します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
ret_int
```

《引数》

なし

《レジスタ設定》

本システムコールを発行した割り込みハンドラには戻りません。

【 C 言語による呼び出し方法】

本システムコールは、C 言語では記述できません。

【エラーコード】

本システムコールを発行した割り込みハンドラには戻りません。

【機能説明】

ret_int システムコールは、他のμITRON 仕様 OS や今後のバージョンアップの互換性をとるために、jmp R14 というマクロ定義にしています。従って、アセンブリ言語で割り込みハンドラを記述した場合は、割り込みハンドラの最後に ret_int を記述してください。割り込みハンドラの処理フローについては、ユーザーズマニュアルを参照してください。

2.8.3. loc_cpu(Lock CPU)

【システムコール名】

loc_cpu　→　割り込みとディスパッチを禁止します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
loc_cpu
```

《引数》

なし

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	--
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

なし

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK　　00000000H(-H’00000000): 正常終了

【機能説明】

外部割り込みとタスクのディスパッチを禁止します。

本システムコール実行後、unl_cpu が実行されるまでの間は、割り込みおよびディスパッチ禁止状態となり、loc_cpu を実行したタスクが割り込みハンドラや他のタスクによってプリエンプト(CPU の実行権の横取り)される可能性はなくなります。つまり、loc_cpu 実行後の割り込み要求や、loc_cpu を実行したタスクが発行したシステムコールにより発生したディスパッチは、unl_cpu により割り込みおよびディスパッチ禁止状態が解除されるまで遅延されます。

既に割り込みおよびディスパッチ禁止状態にあるタスクが loc_cpu を発行した場合は、同じ状態を継続するだけでエラーとはなりません。ただし、loc_cpu を複数回発行しても、その後、unl_cpu を 1 回発行すると割り込みおよびディスパッチ禁止状態が解除されます。

本システムコールは、タスクからのみ発行することができます。

割り込みハンドラ、周期起動ハンドラ、アラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.8.4. unl_cpu(Unlock CPU)

【システムコール名】

unl_cpu　→　割り込みとディスパッチを許可します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
unl_cpu
```

《引数》

なし

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	--
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

なし

《戻り値》

関数の戻り値として常に E_OK を返します。

【エラーコード】

E_OK　　00000000H(-H’00000000): 正常終了

【機能説明】

外部割り込みとタスクのディスパッチを許可します。

すなわち、loc_cpu によって設定されていた割り込みおよびディスパッチ禁止状態を解除します。割り込みおよびディスパッチが禁止されていない状態で unl_cpu を発行した場合は、同じ状態が継続するだけでエラーとはなりません。

本システムコールは、タスクからのみ発行することができます。

割り込みハンドラ、周期起動ハンドラ、アラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.9. メモリプール管理機能

2.9.1. cre_mpf(Create Fixed-size Memory Pool)

【システムコール名】

cre_mpf　→　固定長メモリプールを生成します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
cre_mpf  mpfid
```

《引数》

mpfid	[**]	生成するメモリプール ID 番号
pk_cmpf	[****]	生成情報を格納するパケットの先頭アドレス (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	生成する固定長メモリプールの ID 番号
R2	生成情報を格納したパケットの先頭アドレス
R3	--

pk_cmpf の指す領域には、以下に示す情報を設定してください。

オフセット	サイズ		
+0	4	exinf	拡張情報
+4	4	mpfatr	メモリプール属性
+8	4	mpfcnt	メモリプール全体のブロック数
+12	4	blfsz	固定長メモリブロックのサイズ
+16	4	mpf	ユーザーが確保したメモリプール領域の先頭アドレス

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER cre_mpf (mpfid, pk_cmpf);
```

《引数》

```
ID       mpfid;       固定長メモリプール ID 番号
T_CMPF   *pk_cmpf;    固定長メモリプール生成情報を格納した領域のアドレス
```

pk_cmpf の指す構造体に以下に示す情報を設定してください。

```
typedef struct t_cmpf {
    VP      exinf;   /* 拡張情報 */
    ATR     mpfatr;  /* メモリプール属性 */
    INT     mpfcnt;  /* メモリプール全体のブロック数 */
    INT     blfsz;   /* 固定長メモリブロックサイズ */
    VP      mpf;     /* ユーザーが確保したメモリプール領域の先頭
                        アドレス */
} T_CMPF;
```

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_NOMEM	0FFFFFFF6H(-H'0000000a):	メモリ不足
E_OBJ	0FFFFFFC1H(-H'0000003f):	オブジェクトの状態が不正

【機能説明】

mpfid で指定した ID 番号を持つ固定長メモリプールを生成します。メモリプールとして利用するメモリ領域を確保し、生成したメモリプールの管理ブロックデータの初期化を行います。

生成するメモリプールの情報 pk_cmpf について以下に示します。

●exinf(拡張情報)

生成するメモリプールに関する情報を格納するためにユーザが自由に利用できる領域です。exinf の内容に関しては、**MR32R** は関知しません。

●mpfatr(メモリプール属性)

生成するメモリプールが使用するバッファ領域を内蔵 RAM にするか外部 RAM にするかユーザーが確保するかを指定します。__MR_USER を指定した場合、mpf にメモリプール領域の先頭アドレスを指定してください。

★ 内蔵 RAM を使用する場合

__MR_INT(0)を指定してください。

★ 外部 RAM を使用する場合

__MR_EXT(0x10000)を指定してください。

★ ユーザーが確保した領域をメモリプールとして使用する場合

__MR_USER(0x20000)を指定してください。

●mpfcnt

生成するメモリプールのブロック数を指定してください。ここでは、1 から 32 までの値が指定できます。

●blfsz

生成するメモリプールの 1 ブロックのブロックサイズを指定してください。

本システムコールの指定可能な ID 番号の範囲は、1 から最大項目数定義で設定したユーザシステムの最大固定長メモリプール数までです。

OS カーネルメモリプール領域を確保する際、mpfcnt×blfsz 分のメモリが確保できない場合は、エラーE_NOMEM を返します。

すでに存在するメモリプールに対して、本システムコールを発行した場合は、エラーE_OBJ を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#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);
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
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
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
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 Memory Pool)

【システムコール名】

del_mpf　→　固定長メモリプールを削除します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
del_mpf  mpfid
```

《引数》

mpfid　[**]　削除するメモリプールの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	削除する固定長メモリプールの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　mpfid;　メモリプール ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK        00000000H(-H’00000000) :  正常終了
E_NOEXS     0FFFFFFCCH(-H’00000034):  オブジェクトが存在していない
```

【機能説明】

mpfid で指定した ID 番号を持つ固定長メモリプールを削除します。

メモリプールを削除した後は、同じ ID 番号で新しいメモリプールを生成することができます。削除するメモリプールのメモリブロックを待っているタスクがある場合でも、本システムコールは、正常終了します。その時、メモリブロック待ち状態のタスクは、メモリブロック待ち状態が解除され、エラーE_DLT を返し、実行(RUN)状態あるいは実行可能(READY)状態へ移行します。

また、本システムコールで指定したメモリプールのメモリブロックを獲得しているタスクが存在しても、本システムコールは、正常終了します。そのとき、メモリブロックを獲得しているタスクには何の通知もしません。本システムコールにより、対象メモリプール領域は解放され、再利用可能な状態となります。

なお、存在しないメモリプールに対して本システムコールを発行した場合は、エラー E_NOEXS を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよび、アラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task(void)
{
        :
    del_mpf(ID_mpf1);
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global task
task:
        :
    del_mpf ID_mpf1
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .include “mr32r.inc”
    .global task
task:
        :
    del_mpf ID_mpf1
        :
    ext_tsk
```

2.9.3. get_blf(Get Fixed-size Memory Block)

【システムコール名】

get_blf　→　固定長メモリブロックを獲得します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
get_blf  mpfid
```

《引数》

mpfid　　[**]　　獲得するメモリプール ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	獲得する固定長メモリプールの ID 番号
R2	獲得したメモリブロックの先頭アドレス
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER get_blf (p_blf,mpfid);
```

《引数》

ID	mpfid;	メモリプール ID 番号
VP	*p_blf;	獲得したメモリブロックの先頭アドレスを格納する領域のアドレス

《戻り値》

変数 p_blf に獲得したメモリブロックの先頭アドレスが設定されます。関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) : 正常終了
E_RLWAI	0FFFFFFAAH(-H'00000056): 待ち状態強制解除
E_DLT	0FFFFFFAFH(-H'00000051): 待ちオブジェクトが削除された
EV_RST	0FFFFFF02H(-H'000000FE): リセットにより待ちが解除された
E_NOEXS	0FFFFFFCCH(-H'00000034): オブジェクトが存在していない

【機能説明】

mpfid で示されるメモリプールからメモリブロックを獲得し、獲得したメモリブロックの先頭アドレスを変数 p_blf に格納します。獲得したメモリブロックの内容は、不定です。

指定したメモリプールにメモリブロックがない場合は、本システムコールを発行したタスクは、メモリブロック待ち状態に移行し、メモリブロック待ち行列に FIFO 順でつながれます。

他のタスクの発行した rel_wai、irel_wai システムコールによって待ち状態が強制解除された場合は、エラーE_RLWAI を返します。

他のタスクが del_mpf システムコールによってメモリプールを削除された場合、削除されたメモリプールを待っているタスクは、メモリブロック待ち状態を解除し、エラーE_DLT を返し、実行可能(READY)状態に移行します。

また、他のタスクが vrst_blf システムコールによってメモリプールを初期化した場合、初期化されたメモリプールを待っているタスクは、メモリブロック待ち状態を解除し、エラ

ーEV_RST を返し、実行可能(READY)状態に移行します。
　存在しないメモリプールに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。
　メモリブロックは、固定長です。1 つのメモリブロックのサイズおよびメモリブロック数は、コンフィグレーションファイルあるいは cre_mpf システムコールで設定します。
　本システムコールは、タスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラ、アラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#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”);
    }
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
p_blf: .RES.B 4
    .include “mr32.inc”
    .global task
task:
        :
    get_blf    ID_mpf
    ld24      R5,#p_blf
    st         R2,@R5
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
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)

【システムコール名】

tget_blf　→　固定長メモリブロックを獲得します（タイムアウトあり）

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
tget_blf mpfid,tmout
```

《引数》

mpfid	[**]	獲得するメモリプールの ID 番号
tmout	[****]	タイムアウト値

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	獲得する固定長メモリプールの ID 番号
R2	獲得したメモリブロックの先頭アドレス
R3	--
R4	タイムアウト値

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER tget_blf (p_blf,mpfid,tmout);
```

《引数》

ID	mpfid;	メモリプール ID 番号
VP	*p_blf;	獲得したメモリブロックの先頭アドレスを格納する領域のアドレス
TMO	tmout	タイムアウト値

《戻り値》

変数 p_blf に獲得したメモリブロックの先頭アドレスが設定されます。関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H’00000000) :	正常終了
E_RLWAI	0FFFFFFAAH(-H’00000056):	待ち状態強制解除
E_DLT	0FFFFFFAFH(-H’00000051):	待ちオブジェクトが削除された
E_TMOUT	0FFFFFFABH(-H’00000055):	ポーリング失敗またはタイムアウト
EV_RST	0FFFFFF02H(-H’000000FE):	リセットにより待ちが解除された
E_NOEXS	0FFFFFFCCH(-H’00000034):	オブジェクトが存在していない

【機能説明】

mpfid で示されるメモリプールからメモリブロックを獲得し、獲得したメモリブロックの先頭アドレスを変数 p_blf に格納します。獲得したメモリブロックの内容は、不定です。

指定したメモリプールにメモリブロックがない場合は、本システムコールを発行したタスクは、メモリブロック待ち状態に移行し、メモリブロック待ち行列とタイムアウト待ち行列の 2 つの待ち行列につながれます。

本システムコール発行による待ち状態は、以下に示す場合に解除されます。なお、待ち状態が解除されると本システムコールを発行したタスクは、メモリブロック待ち行列とタイム

アウト待ち行列の 2 つの待ち行列からはずされ、レディキューに接続されます。

- tmout の時間が経過する前に、待ち解除条件が成立した場合

  この時のエラーコードは、E_OK を返します。

- 待ち解除条件が成立する前に、tmout の時間が経過した場合

  この時のエラーコードは、E_TMOUT を返します。

- 待ち解除条件が成立する前に rel_wai、irel_wai システムコールを発行した場合

  この時のエラーコードは、E_RLWAI を返します。

- 他のタスクから発行した del_mpf システムコールによって待ち状態の対象となっているメモリプールが削除された場合

  この時のエラーコードは、E_DLT を返します。

- 他のタスクから発行した vrst_blf システムコールによって待ち状態の対象となっているメモリプールが削除された場合

  この時のエラーコードは、EV_RST を返します。

tmout には、-1 から 7FFFFFFFH まで指定できます。なお、tmout に TMO_POL=0 を指定した場合は、タイムアウト値として 0 を指定したことを示し、pget_blf と同じ動作をします。また、tmout=TMO_FEVR(-1)を指定した場合は、永久待ちの指定で、get_blf と同じ動作をします。メモリブロックは、固定長です。1 つのメモリブロックのサイズおよびメモリブロック数はコンフィグレーションファイルあるいは cre_mpf システムコールで設定します。

存在しないメモリプールに対して、本システムコールを発行した場合、エラーコード E_NOEXS を返します。

本システムコールは、タスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラ、アラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32.h>
#include “id.h”
VP      p_blf;
void task()
{
    if( tget_blf(&p_blf,ID_mpl,50) != E_OK ){
        error(“Not enough memory¥n”);
    }
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
p_blf: .RES.B 4
    .include mr32.inc
    .global task
task:
        :
    tget_blf   ID_mpl,50
    ld24      R5,#p_blf
    st        R2,@R5
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
p_blf: .space 4
    .include mr32.inc
    .global task
task:
        :
    tget_blf   ID_mpl,50
    ld24      R5,#p_blf
    st        R2,@R5
        :
    ext_tsk
```

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

【システムコール名】

pget_blf　→　固定長メモリブロックを獲得します（待ちなし）

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
pget_blf mpfid
```

《引数》

mpfid	[**]	獲得するメモリプールの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	獲得する固定長メモリプールの ID 番号
R2	獲得したメモリブロックの先頭アドレス
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER pget_blf (p_blf,mpfid);
```

《引数》

ID	mpfid;	メモリプール ID 番号
VP	*p_blf;	獲得したメモリブロックの先頭アドレスを格納する領域のアドレス

《戻り値》

変数 p_blf に獲得したメモリブロックの先頭アドレスが設定されます。関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_TMOUT	0FFFFFFABH(-H'00000055):	ポーリング失敗またはタイムアウト
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない

【機能説明】

mpfid で示されるメモリプールからメモリブロックを獲得し、獲得したメモリブロックの先頭アドレスを変数 p_blf に格納します。

指定したメモリプールにメモリブロックがない場合は、エラーE_TMOUT を返します。get_blf、tget_blf とは、異なり本システムコールを発行したタスクは、待ち状態には移行しません。

存在しないメモリプールに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。メモリブロックは、固定長です。1 つのメモリブロックのサイズおよびメモリブロック数はコンフィグレーションファイルあるいは、cre_mpf システムコールで設定します。

本システムコールは、タスク、ハンドラのどちらからでも発行できます。

【使用例】

《 C 言語の使用例 》

```
#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”);
    }
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
p_blf: .RES.B 4
    .include mr32.inc
    .global task
task:
        :
    pget_blf   ID_mpf
    ld24      R5,#p_blf
    st        R2,@R5
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
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)

【システムコール名】

rel_blf　→　固定長メモリブロックを解放します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
rel_blf  mpfid
```

《引数》

mpfid	[**]	メモリプール ID 番号
p_blf	[****]	解放するメモリブロックのアドレス (R2 レジスタに解放するメモリブロックのアドレス設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	解放するメモリの固定長メモリプール ID 番号
R2	解放するメモリブロックの先頭アドレス
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER rel_blf (mpfid, p_blf);
```

《引数》

ID	mpfid;	メモリプール ID 番号
VP	p_blf;	解放するメモリブロックの先頭アドレス

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) : 正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034): オブジェクトが存在していない

【機能説明】

p_blf に示される先頭アドレスをもつメモリブロックを解放します。

解放するメモリブロックの先頭アドレスは必ず、get_blf、tget_blf あるいは pget_blf で獲得した先頭アドレスを指定してください。

また、対象メモリプールの待ち行列にタスクがつながれている場合は、待ち行列の先頭タスクを待ち行列からはずし、レディキューにつなぎかえこのタスクにメモリブロックを割り当てます。この時のタスクの状態は、メモリブロック待ち状態から実行(RUN)状態あるいは実行可能(READY)状態へ移行します。

本システムコールは、p_blf の内容をチェックしません。

本システムコールは、タスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラ、アラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#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);
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
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  ; 解放するメモリブロックの先頭アドレス
        :
    rel_blf    ID_mpf1
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
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  ; 解放するメモリブロックの先頭アドレス
        :
    rel_blf    ID_mpf1
        :
    ext_tsk
```

2.9.7. irel_blf(Release Fixed-size Memory Block)

【システムコール名】

irel_blf　→　固定長メモリブロックを解放します(ハンドラ専用)

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
irel_blf mpfid
```

《引数》

mpfid	[**]	メモリプール ID 番号
p_blf	[****]	解放するメモリブロックのアドレス (R2 レジスタに解放するメモリブロックのアドレス設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	解放するメモリの固定長メモリプール ID 番号
R2	解放するメモリブロックの先頭アドレス
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER irel_blf (mpfid, p_blf);
```

《引数》

ID	mpfid;	メモリプール ID 番号
VP	p_blf;	解放するメモリブロックの先頭アドレス

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) : 正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034): オブジェクトが存在していない

【機能説明】

rel_blf システムコールと同じ機能を割り込みハンドラ、周期起動ハンドラ、アラームハンドラから利用する場合に、このシステムコールを使用してください。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
#define ID_mpf1 1
void int_hand()
{
    VP   p_blf;
    if( pget_blf(&p_blf, ID_mpf1) != E_OK )
        error(“Not enough memory ¥n”);
                :
    irel_blf(ID_mpf1,p_blf);
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
p_blf:   .RES.B 4
ID_mpf1:        .equ    1
    .include “mr32r.inc”
    .global int_hand
int_hand:
        :
    pget_blf   ID_mpf1
    ld24        R5,#p_blf
    st          R2,@R5
        :
    ld24        R5,#p_blf
    ld          R2,@R5  ; 解放するメモリブロックの先頭アドレス
        :
    irel_blf    ID_mpf1
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
p_blf:   .space 4
    .equ ID_mpf1,1
    .include “mr32r.inc”
    .global int_hand
int_hand:
        :
    pget_blf   ID_mpf1
    ld24        R5,#p_blf
    st  R2,@R5
        :
    ld24        R5,#p_blf
    ld          R2,@R5  ; 解放するメモリブロックの先頭アドレス
        :
    irel_blf    ID_mpf1
        :
```

2.9.8. ref_mpf(Refer Fixed-size MemoryPool Status)

【システムコール名】

ref_mpf　→　固定長メモリプール状態を参照します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
ref_mpf  mpfid
```

《引数》

```
mplid     [ ** ]    メモリプール ID 番号
pk_rmpf   [****]    状態を返すパケットの先頭アドレス
                      (R2 レジスタに設定してください)
```

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	固定長メモリプール ID 番号
R2	メモリプールの状態を返すパケットの先頭アドレス
R3	--

pk_rmpf の指す領域には、以下の情報が返されます。

```
オフセット   サイズ
    +0         4     exinf    拡張情報
    +4         4     wtsk     待ちタスクの有無
    +8         4     frbcnt   空き領域のブロック数
    +12        4     blksz    ブロックサイズ
```

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER ref_mpf (pk_rmpf, mpfid);
```

《引数》

```
T_RMPF   *pk_rmpf;      メモリプール状態を返す構造体の先頭アドレス
ID       mpfid;         メモリプール ID 番号
```

《戻り値》

関数の戻り値としてエラーコードを返します。
pk_rmpf の指す構造体には、以下の情報が返されます。

```
typedef  struct t_rmpf {
    VP       exinf;   /* 拡張情報 */
    BOOL_ID  wtsk;    /* 待ちタスクの有無 */
    INT      frbcnt;  /* 空き領域のブロック数 */
    INT      blksz;   /* ブロックサイズ */
} T_RMPF;
```

【エラーコード】

```
E_OK          00000000H(-H'00000000) :  正常終了
E_NOEXS       0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
```

【機能説明】

本システムコールでは、mpfid で指定したメモリプールの以下の状態を返します。

●exinf

拡張情報を返します。

●wtsk

wtsk には、指定されたメモリプールを待っている先頭タスクの ID 番号を返します。対象となるメモリプールを待つタスクがない場合は、FALSE(0)を返します。

●frbcnt

指定したメモリプールの空きブロック数を返します。

●blksz

対象となるメモリプールのブロックサイズを返します。

存在しないメモリプールに対して本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスク、割り込みハンドラ、周期起動ハンドラ、アラームハンドラから発行することができます。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
    T_RMPF rmpf;
        :
    ref_mpf(&mpf,ID_mpf);
        :
    ext_tsk();
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
rmpf:   .RES.B   16
    .include “mr32r.inc”
    .global task
task:
        :
    ld24      R2,#rmpf
    ref_mpf    ID_mpf
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
rmpf:   .space   16
    .include “mr32r.inc”
    .global task
task:
        :
    ld24      R2,#rmpf
    ref_mpf    ID_mpf
        :
```

2.9.9. cre_mpl(Create Variable-size Memory Pool)

【システムコール名】

cre_mpl　→　可変長メモリプールを生成します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
cre_mpl  mplid
```

《引数》

mplid	[**]	生成するメモリプールの ID 番号
pk_cmpl	[****]	可変長メモリプール生成情報 (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	生成する可変長メモリプールの ID 番号
R2	生成情報を格納したパケットの先頭アドレス
R3	--

pk_cmpl の指す領域には、以下に示す情報を設定してください。

オフセット	サイズ		
+0	4	exinf	拡張情報
+4	4	mplatr	メモリプール属性
+8	4	mplsz	メモリプール全体のサイズ
+12	4	maxblksz	獲得するメモリブロックの最大サイズ
+16	4	mpl	ユーザーが確保したメモリプール領域の先頭アドレス

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER cre_mpl (mplid, pk_cmpl);
```

《引数》

```
ID       mplid;          メモリプール ID 番号
T_CMPL   *pk_cmpl;       可変長メモリプール生成情報を格納した領域の
                         アドレス
```

pk_cmpf の指す構造体に以下に示す情報を設定してください。

```
typedef struct t_cmpl {
    VP              exinf;          /* 拡張情報 */
    ATR             mplatr;         /* メモリプール属性 */
    INT             mplsz;          /* メモリプール全体のサイズ */
    INT             maxblksz;       /* 獲得するメモリブロックの最大サ
                                       イズ */
    VP              mpf;            /* ユーザーが確保したメモリプール
                                       領域の先頭アドレス */
} T_CMPL;
```

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_NOMEM	0FFFFFFF6H(-H'0000000a):	メモリ不足
E_OBJ	0FFFFFFC1H(-H'0000003f):	オブジェクトの状態が不正

【機能説明】

mplid で指定した ID 番号を持つ可変長メモリプールを生成します。
メモリプールとして利用するメモリ領域を確保し、生成したメモリプールの管理ブロックデータの初期化を行います。
生成するメモリプールの情報 pk_cmpl について以下に示します。

●exinf(拡張情報)

生成するメモリプールに関する情報を格納するためのユーザが自由に利用できる領域です。exinf の内容に関しては、**MR32R** は関知しません。

●mplatr(メモリプール属性)

生成するメモリプールが使用するバッファ領域を内蔵 RAM にするか外部 RAM にするかユーザーが確保するかを指定します。__MR_USER を指定した場合、mpf にメモリプール領域の先頭アドレスを指定してください。

★ 内蔵 RAM を使用する場合
__MR_INT(0)を指定してください。

★ 外部 RAM を使用する場合
__MR_EXT(0x10000)を指定してください。

★ ユーザーが確保した領域をメモリプールとして使用する場合
__MR_USER(0x20000)を指定してください。

●mplsz

ここで設定した値のメモリ領域を確保し、メモリプールとして利用します。ここで指定したメモリがない場合は、エラーE_NOMEM を返します。

●maxblksz

MR32R の可変長メモリプールは、4 つの固定長メモリブロックのサイズに分けて、ユーザが指定したサイズに最も最適なメモリブロックを 4 つのサイズの中から選択し、メモリの割り当てを行います。各固定長メモリブロックのサイズは、maxblksz をユーザが指定することで決定します。

本システムコールの指定可能な ID 番号の範囲は、1 から最大項目数定義で設定したユーザシステムの最大可変長メモリプール数までです。
すでに存在するメモリプールに対して、本システムコールを発行した場合は、エラーE_OBJ を返します。OS カーネルメモリプール領域を確保する際、mplsize 分のメモリが確保できない場合は、エラーE_NOMEM を返します。
本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#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);
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
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
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
setmpl: .space   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,R)
    cre_mpl    ID_mpl1
        :
    ext_tsk
```

2.9.10. del_mpl(Delete Variable-size Memory Pool)

【システムコール名】

del_mpl　→　可変長メモリプールを削除します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
del_mpl  mplid
```

《引数》

mplid　[**]　削除するメモリプールの ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	削除する可変長メモリプールの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　mplid;　可変長メモリプール ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK        00000000H(-H'00000000) :  正常終了
E_NOEXS     0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
```

【機能説明】

mplid で指定した ID 番号を持つ可変長メモリプールを削除します。

メモリプールを削除した後は、同じ ID 番号で新しいメモリプールを生成することができます。削除するメモリプールのメモリブロックを待っているタスクがある場合でも、本システムコールは、正常終了します。その時、メモリブロック待ち状態のタスクは、待ち状態が解除され、エラーE_DLT を返し、実行(RUN)状態あるいは実行可能(READY)状態へ移行します。

また、本システムコールで指定したメモリプールのメモリブロックを獲得しているタスクが存在しても、本システムコールは、正常終了します。そのとき、メモリブロックを獲得しているタスクには何の通知もしません。本システムコールにより、対象メモリプール領域は解放され、再利用可能な状態となります。

存在しないメモリプールに対して本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task(void)
{
        :
    del_mpl(ID_mpl1);
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global task
task:
        :
    del_mpl ID_mpl1
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .include “mr32r.inc”
    .global task
task:
        :
    del_mpl ID_mpl1
        :
    ext_tsk
```

2.9.11. get_blk(Get Variable-size Memory Block)

【システムコール名】

get_blk　→　可変長メモリブロックを獲得します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
get_blk  mplid, blksz
```

《引数》

mplid	[**]	可変長メモリプール ID 番号
blksz	[****]	獲得するメモリブロックのサイズ

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	獲得する可変長メモリプールの ID 番号
R2	獲得したメモリブロックの先頭アドレス
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER get_blk (p_blk,mplid,blksz);
```

《引数》

ID	mplid;	メモリプール ID 番号
VP	*p_blk;	獲得したメモリブロックの先頭アドレスを格納する領域のアドレス
INT	blksz	獲得するメモリのサイズ

《戻り値》

変数 p_blk に獲得したメモリブロックの先頭アドレスが設定されます。関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_RLWAI	0FFFFFFAAH(-H'00000056):	待ち状態強制解除
E_DLT	0FFFFFFAFH(-H'00000051):	待ちオブジェクトが削除された
EV_RST	0FFFFFF02H(-H'000000FE):	リセットにより待ちが解除された
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない

【機能説明】

本システムコールは、可変長のメモリブロックを獲得します。

変数 blksz に指定された大きさのメモリを指定したメモリプールから獲得し、獲得したメモリブロックの先頭アドレスを変数 p_blk に格納します。獲得したメモリブロックの内容は不定です。

メモリプールにメモリブロックがない場合は、本システムコールを発行したタスクは、メモリブロック待ち状態に移行し、メモリブロック待ち行列に FIFO 順でつながれます。

rel_wai、irel_wai システムコールによって待ち状態が強制解除された場合は、エラー E_RLWAI を返します。

他のタスクの発行した del_mpl システムコールによって条件成立を待っているメモリプー

ルが削除された場合は、待ち状態にあったタスクはメモリブロック待ち状態が解除され、エラーE_DLT を返し、実行(RUN)状態あるいは実行可能(READY)状態に移行します。

また、他のタスクの発行した vrst_blk システムコールによって条件成立を待っているメモリプールが初期化された場合は、待ち状態にあったタスクはメモリブロック待ち状態が解除され、エラーEV_RST を返し、実行(RUN)状態あるいは実行可能(READY)状態に移行します。

存在しないメモリプールに対して本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
#define ID_mpl1 1
VP      p_blk;
void task(void)
{
    /* 70 バイトのメモリブロックを獲得 */
    if( get_blk(&p_blk,ID_mpl1,70) != E_OK ){
        error(“Not enough memory¥n”);
    }
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
p_blk: .RES.B 4
    .include “mr32r.inc”
    .global task
task:
        :
    get_blk     ID_mpl1,50       ; 50 バイトのメモリを獲得
    ld24        R5,#p_blk
    st  R2,@R5          ; 獲得したメモリのアドレスを転送
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
p_blk: .space 4
    .include “mr32r.inc”
    .global task
task:
        :
    get_blk     ID_mpl1,50       ; 50 バイトのメモリを獲得
    ld24        R5,#p_blk
    st  R2,@R5          ; 獲得したメモリのアドレスを転送
        :
    ext_tsk
```

2.9.12. tget_blk(Get Variable-size Memory Block with Timeout)

【システムコール名】

tget_blk　→　可変長メモリブロックを獲得します（タイムアウト付き）

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
tget_blk mplid,blksz,tmout
```

《引数》

mplid	[**]	獲得するメモリプールの ID 番号
blksz	[****]	獲得するメモリサイズ
tmout	[****]	タイムアウト値

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	獲得する可変長メモリプールの ID 番号
R2	獲得したメモリブロックの先頭アドレス
R3	--
R4	タイムアウト値

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER tget_blk (p_blk,mplid,blksz,tmout);
```

《引数》

VP	*p_blk;	獲得したメモリブロックの先頭アドレスを格納する領域のアドレス
ID	mplid;	メモリプール ID 番号
INT	blksz;	獲得するメモリブロックのサイズ
TMO	tmout	タイムアウト値

《戻り値》

変数 p_blk に獲得したメモリブロックの先頭アドレスが設定されます。関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) : 正常終了
E_RLWAI	0FFFFFFAAH(-H'00000056): 待ち状態強制解除
E_DLT	0FFFFFFAFH(-H'00000051): 待ちオブジェクトが削除された
E_TMOUT	0FFFFFFABH(-H'00000055): ポーリング失敗またはタイムアウト
EV_RST	0FFFFFF02H(-H'000000FE): リセットにより待ちが解除された
E_NOEXS	0FFFFFFCCH(-H'00000034): オブジェクトが存在していない

【機能説明】

mplid で示されるメモリプールからメモリブロックを獲得し、獲得したメモリブロックの先頭アドレスを変数 p_blk に格納します。指定したサイズのメモリブロックがメモリプールにない場合、本システムコールを発行したタスクは、メモリブロック待ち状態に移行し、メモリブロック待ち行列とタイムアウト待ち行列の 2 つの待ち行列につながれます。

本システムコール発行による待ち状態は、以下に示す場合に解除されます。なお、待ち状

態が解除されると本システムコールを発行したタスクは、メモリブロック待ち行列とタイムアウト待ち行列の 2 つの待ち行列がはずされ、レディキューに接続されます。

- tmout の時間が経過する前に、待ち解除条件が成立した場合

  この時のエラーコードは、E_OK を返します。

- 待ち解除条件が成立する前に、tmout の時間が経過した場合

  この時のエラーコードは、E_TMOUT を返します。

- 待ち解除条件が成立する前に rel_wai、irel_wai システムコールを発行した場合

  この時のエラーコードは、E_RLWAI を返します。

- 他のタスクから発行した del_mpl システムコールによって待ち状態の対象となっているメモリプールが削除された場合

  この時のエラーコードは、E_DLT を返します。

- 他のタスクから発行した vrst_blk システムコールによって待ち状態の対象となっているメモリプールが削除された場合

  この時のエラーコードは、EV_RST を返します。

tmout には、-1 から 7FFFFFFFH まで指定できます。なお、tmout に TMO_POL=0 を指定した場合は、タイムアウト値として 0 を指定したことを示し、pget_blk と同じ動作をします。また、tmout=TMO_FEVR(-1)を指定した場合は、永久待ちの指定で、get_blk と同じ動作をします。

存在しないメモリプールに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールは、タスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラ、アラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#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”);
    }
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
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
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
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)

【システムコール名】

pget_blk　→　可変長メモリブロックを獲得します (待ちなし)

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
pget_blk mplid, blksz
```

《引数》

mplid	[**]	メモリプール ID 番号
blksz	[****]	獲得するメモリブロックのサイズ

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	獲得する可変長メモリプールの ID 番号
R2	獲得したメモリブロックの先頭アドレス
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER pget_blk (p_blk,mplid,blksz);
```

《引数》

VP	*p_blk;	獲得したメモリブロックの先頭アドレスを格納する領域のアドレス
ID	mplid;	メモリプール ID 番号
INT	blksz	獲得するメモリのサイズ

《戻り値》

変数 p_blk に獲得したメモリブロックの先頭アドレスが設定されます。関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) : 正常終了
E_TMOUT	0FFFFFFABH(-H'00000055): ポーリング失敗またはタイムアウト
E_NOEXS	0FFFFFFCCH(-H'00000034): オブジェクトが存在していない

【機能説明】

可変長のメモリブロックを獲得します。

変数 blksz に指定された大きさのメモリをメモリプールから獲得し、獲得したメモリブロックの先頭アドレスを変数 p_blk に格納します。獲得したメモリブロックの内容は不定です。

指定したサイズのメモリブロックが、メモリプールにない場合は、エラーE_TMOUT を返します。get_blk、tget_blk とは、異なり本システムコールを発行したタスクは、待ち状態には移行しません。

存在しないメモリプールに対して本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよび、アラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
#define ID_mpl1 1
VP      p_blk;
void task()
{
    /* 70 バイトのメモリブロックを獲得 */
    if( pget_blk(&p_blk,ID_mpl1,70) != E_OK )
        error(“Not enough memory¥n”);
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
p_blk: .RES.B 4
    .equ ID_mpl1,1
    .include “mr32r.inc”
    .global task
task:
        :
    pget_blk  ID_mpl1,50      ; 50 バイトのメモリを獲得
    ld24 R5,#p_blk
    st    R2,@R5          ; 獲得したメモリのアドレスを転送
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
p_blk: .space  4
    .equ ID_mpl1,1
    .include “mr32r.inc”
    .global task
task:
        :
    pget_blk  ID_mpl1,50      ; 50 バイトのメモリを獲得
    ld24 R5,#p_blk
    st    R2,@R5          ; 獲得したメモリのアドレスを転送
        :
    ext_tsk
```

2.9.14. rel_blk(Release Variable-size Memory Block)

【システムコール名】

rel_blk　→　可変長メモリブロックを解放します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
rel_blk  mplid
```

《引数》

mplid	[**]	可変長メモリプール ID 番号
p_blf	[****]	解放するメモリブロックのアドレス (R2 レジスタに解放するメモリブロックのアドレス設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	解放するメモリの可変長メモリプール ID 番号
R2	解放するメモリブロックの先頭アドレス
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER rel_blk (mplid,p_blk);
```

《引数》

ID	mplid;	メモリプール ID 番号
VP	p_blk;	解放メモリブロックの先頭アドレス

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) : 正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034): オブジェクトが存在していない

【機能説明】

p_blk に示される先頭アドレスをもつメモリブロックを解放します。

解放するメモリブロックの先頭アドレスは必ず get_blk、tget_blk あるいは pget_blk で獲得した先頭アドレスを指定してください。本システムコールは、p_blk の内容をチェックしていません。

タスクがメモリブロックの獲得を待っている場合は、メモリブロック待ち行列の先頭タスクから要求サイズを調べ、条件を満足すればメモリ待ち状態から実行可能(READY)状態へ移行します。なお、メモリの割り当てにおいて、メモリ待ち行列の先頭タスクから要求サイズのチェックを行い、要求サイズを満足すれば、次に接続されているタスクの要求サイズのチェックを順に行っていく。もし、要求サイズを満足しないタスクがあれば、その時点で、メモリブロックの割り当てを終了します。

存在しないメモリプールに対して本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハン

ドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
#define ID_mpl1 1
void task()
{
    VP   p_blk;
    /* 60 バイトのメモリブロックを獲得 */
    if( pget_blk(&p_blk,ID_mpl1,60) != E_OK )
        error(“Not enough memory ¥n”);
              :
    rel_blk(ID_mpl1,p_blk);    /* メモリブロックを解放 */
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
p_blk:   .RES.B 4
ID_mpl1:        .equ    1
    .include “mr32r.inc”
    .global _task
_task:
        :
    pget_blk ID_mpl1,60    ; 60 バイトのメモリブロックを獲得
    ld24    R5,#p_blk
    st      R2,@R5
        :
    ; 解放メモリブロックのアドレスはユーザ設定とします。
    ld24    R5,#p_blk
    ld      R2,@R5        ; 解放するメモリブロックの先頭アドレス (上位)
    rel_blk               ; メモリブロックを解放
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
p_blk:   .space 4
    .equ ID_mpl1,1
    .include “mr32r.inc”
    .global _task
_task:
        :
    pget_blk ID_mpl1,60    ; 60 バイトのメモリブロックを獲得
    ld24    R5,#p_blk
    st      R2,@R5
        :
    ; 解放メモリブロックのアドレスはユーザ設定とします。
    ld24    R5,#p_blk
    ld      R2,@R5        ; 解放するメモリブロックの先頭アドレス (上位)
    rel_blk               ; メモリブロックを解放
```

2.9.15. ref_mpl(Refer Variable-size MemoryPool Status)

【システムコール名】

ref_mpl　→　可変長メモリプール状態を参照します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
ref_mpl  pk_rmpl,mplid,pk_rmpl
```

《引数》

```
mplid     [ ** ]    メモリプール ID 番号
pk_rmpl   [****]    状態を返すパケットアドレス(pk_rmpl)
                    (R2 レジスタに設定してください)
```

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	可変長メモリプール ID 番号
R2	メモリプールの状態を返すパケットの先頭アドレス
R3	--

pk_rmpl の指す領域には、以下の情報が返されます。

```
オフセット    サイズ
    +0          4     exinf     拡張情報
    +4          4     wtsk      待ちタスクの有無
    +8          4     frsz      空き領域の合計サイズ
    +12         4     maxsz     空き領域の最大サイズ
```

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER ref_mpl (pk_rmpl, mplid);
```

《引数》

```
T_RMPL   * pk_rmpl;      メモリプール状態を返すパケットの先頭アドレス
ID       mplid;          メモリプール ID 番号
```

《戻り値》

関数の戻り値としてエラーコードを返します。
pk_rmpl の指す構造体には、以下の情報が返されます。

```
typedef struct t_rmpl {
    VP      exinf;  /* 拡張情報 */
    BOOL_ID wtsk;   /* 待ちタスクの有無 */
    INT     frsz;   /* 空き領域の合計サイズ */
    INT     maxsz;  /* 空き領域の最大サイズ */
} T_RMPL;
```

【エラーコード】

```
E_OK          00000000H(-H'00000000) :  正常終了
E_NOEXS       0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
```

【機能説明】

mplid で指定した可変長メモリプールの状態を返します。

●exinf

拡張情報を返します。

●wtsk

wtsk には、指定されたメモリプールを待っている先頭タスク ID 番号を返します。対象となるメモリプールを待つタスクがない場合は、FALSE(0)を返します。

●frsz

空き領域の合計サイズを返します。

●maxsz

すぐに獲得できる最大の空き領域サイズを返します。

存在しないメモリプールに対して本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスク、ハンドラのどちらからでも発行できます。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
    T_RMPL rmpl;
    ref_mpl(&rmpl,ID_mpl1);
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
rmpl:   .space    10
    .equ ID”mpl1,1
    .include mr32r.inc
    .global task
task:
        :
    ld24      R2,#rmpl
    ref_mpl    ID_mpl1
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
rmpl:   .space    10
    .equ ID”mpl1,1
    .include mr32r.inc
    .global task
task:
        :
    ld24      R2,#rmpl
    ref_mpl    ID_mpl1
```

2.10. 時間管理機能

2.10.1. set_tim(Set Time)

【システムコール名】

set_tim　→　システムクロックを設定します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
set_tim  pk_tim
```

《引数》

pk_tim　[****]　設定するシステム時刻を示すパケットの先頭アドレスを示すラベル

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	--
R2	設定するシステム時刻パケットの先頭アドレス
R3	--

【 C 言語による呼び出し方法】

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

《引数》

SYSTIME　*pk_tim;　設定するシステム時刻データの先頭アドレス

《戻り値》

関数の戻り値として常に E_OK を返します。

【エラーコード】

E_OK　00000000H(-H’00000000): 正常終了

【機能説明】

システムクロックの値を pk_tim で示される値に設定します[29]。
48 ビットのシステムクロックは ltime(32 ビット)、utime(16 ビット)に分けて扱います。
システムクロックの単位は、**MR32R** のシステムクロックの時間を 1 クロックとして扱います。
本システムコールはタスク、ハンドラのどちらからでも発行できます。

[29] システム時刻は、リセット時を 0 とし、システムクロックの割り込みの発生回数を 48 ビットのデータで表します。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
    SYSTITIME time;       /* 時刻データ格納変数 */
    time.utime = 0;       /* 上位時刻データの設定 */
    time.ltime = 0;       /* 下位時刻データの設定 */
    set_tim( &time );     /* システム時刻の変更 */
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
time:   .RES.B    6
    .INCLUDE “mr32r.inc”
    .GLOBAL   task
task:
    set_tim   time
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
time:   .space    6
    .INCLUDE “mr32r.inc”
    .GLOBAL   task
task:
    set_tim   time
        :
```

2.10.2. get_tim(Get Time)

【システムコール名】

get_tim　→　システム時刻の値を読み出します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
get_tim pk_tim
```

《引数》

pk_tim　[****]　読みだしたシステム時刻を格納するパケットの先頭アドレスを示すラベル

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	--
R2	読み出したシステム時刻を格納するパケットの先頭アドレス
R3	--

【 C 言語による呼び出し方法】

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

《引数》

SYSTIME　*pk_tim;　　現在の時刻データを返す構造体の先頭アドレス

《戻り値》

関数の戻り値として常に E_OK を返します。
pk_tim の指す構造体には、現在の時刻データが返されます。

【エラーコード】

E_OK　　00000000H(-H'00000000):　正常終了

【機能説明】

システムクロックの現在の値を読みだし、リターンパラメータ pk_tim に返します[30]。
48 ビットのシステム時刻は、ltime(32 ビット)、utime(16 ビット)に分けて扱います。
本システムコールはタスク、ハンドラのどちらからでも発行できます。

[30] システム時刻は、リセット時を 0 とし、システムクロックの割り込みの発生回数を 48 ビットのデータで表します。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
    SYSTIME    time;           /* 時刻データ格納変数 */
    get_tim( &time );   /* 時刻データの読み出し */
    printf(“system_clock.utime = %X¥n”,time.utime);
    printf(“system_clock.ltime = %X¥n”,time.ltime);
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
time: .RES.B    6
    .include “mr32r.inc”
    .global   task
task:
    get_tim    time
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
time: .space    6
    .include “mr32r.inc”
    .global   task
task:
    get_tim    time
        :
```

2.10.3. dly_tsk(Delay Task)

【システムコール名】

dly_tsk　→　タスクの実行を遅延します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
dly_tsk  dlytim
```

《引数》

dlytim　[****]　遅延時間

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	--
R2	遅延時間
R3	--

【 C 言語による呼び出し方法】

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

《引数》

DLYTIME dlytim;　遅延時間

《戻り値》

関数の戻り値として常にエラーコードを返します。

【エラーコード】

```
E_OK          00000000H(-H’00000000) :  正常終了
E_RLWAI       0FFFFFFAAH(-H’00000056):  待ち状態強制解除
```

【機能説明】

自タスクの実行を、dlytim で指定した時間だけ一時的に停止し、実行(RUN)状態から待ち状態へ移行します。本システムコール発行による待ち状態は、以下に示す場合に解除されます。なお、待ち状態が解除されると本システムコールを発行したタスクは、タイムアウト待ち行列からはずされ、レディキューに接続されます。

●dlytim の時間が経過した場合

この時のエラーコードは、E_OK を返します。

●dlytim の時間が経過する前に前に rel_wai、irel_wai システムコールを発行した場合

この時のエラーコードは、E_RLWAI を返します。

なお、遅延時間中に wup_tsk、iwup_tsk システムコールが発行されても、待ち解除とはなりません。

dlytim で指定する時間の単位はコンフィグレーションファイルで指定したシステムクロックの単位時間です[31]。

dlytim の最大値は 0x7FFFFFFF です。システムクロックの単位時間が 10ms の場合プログラ

[31] システムクロックの設定方法については、ユーザーズマニュアルを参照してください。

ムで、

```
dly_tsk(5);
```

と記述すれば 50ms 間、自タスクが実行(RUN)状態からタイムアウト待ち状態へ移行します。

本システムコールは、タスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラ、アラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
    if( dly_tsk( 10 ) != E_OK )
        printf(“Forced wakeup¥n”);
              :
}
```

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

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

2.10.4. def_cyc (Define Cyclic Handler)

【システムコール名】

def_cyc　→　周期起動ハンドラを定義します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
def_cyc  cycno, cycact
```

《引数》

cycno　[**]　周期起動ハンドラ ID 番号
pk_dcyc　[****]　周期起動ハンドラの定義情報
(先頭アドレスを R2 レジスタに設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	周期起動ハンドラ ID 番号
R2	周期起動ハンドラの定義情報を格納したパケットの先頭アドレス
R3	--

pk_dcyc の指す領域には、以下に示す情報を設定してください。

オフセット	サイズ		
+0	4	exinf	拡張情報
+4	4	cycatr	周期起動ハンドラ属性
+8	4	cychdr	周期起動ハンドラアドレス
+12	4	cycact	周期起動ハンドラ活性状態
+16	4	cyctim	周期起動間隔

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER def_cyc (cycno, pk_dcyc);
```

《引数》

```
HNO      cycno;          周期起動ハンドラ ID 番号
T_DCYC   *pk_dcyc;       割り込み定義情報
```

pk_dint の指す構造体に以下に示す情報を設定してください。

```
typedef struct t_dcyc {
    VP      exinf;          /* 拡張情報 */
    ATR     cycatr;         /* 周期起動ハンドラ属性 */
    FP      cychdr;         /* 周期起動ハンドラアドレス */
    UINT    cycact;         /* 周期起動ハンドラ活性状態 */
    CYCTIME cyctim;         /* 周期起動間隔 */
} T_DCYC;
```

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000)	: 正常終了

【機能説明】

cycno で指定された番号に対応する周期起動ハンドラを定義します、周期起動ハンドラを使用可能にします。pk_dcyc=NADR(=-1)を指定した場合は、既に定義済みの周期起動ハンドラの定義を解除します。

●exinf(拡張属性)

ここで設定した値は、**MR32R** は関知しません。

●cycatr(周期起動ハンドラ属性)

ここで設定した値は、**MR32R** は関知しません。

●cychdr(周期起動ハンドラアドレス)

定義する周期起動ハンドラのエントリアドレスをこの領域に設定してください。

●cycact(周期起動ハンドラ活性状態)

定義する周期起動ハンドラの活性状態をこの領域に設定してください。
指定可能な値は、以下の通りです。

★TCY_OFF(=0)
周期起動ハンドラは起動されません

★TCY_ON(=1)
周期起動ハンドラは起動されます

●cyctim(周期起動間隔)

定義する周期起動ハンドラの起動間隔をこの領域に設定してください。

すでに定義済みの周期起動ハンドラに対して周期起動ハンドラを再定義することも可能です。再定義の場合、あらかじめ定義の解除を行う必要はありません。既に定義されている周期起動ハンドラに対して新しいハンドラを再定義してもエラーにはなりません。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#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)
{
    :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .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:
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .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)

【システムコール名】

act_cyc　→　周期起動ハンドラの活性制御を行います

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
act_cyc  cycno, cycact
```

《引数》

cycno	[**]	周期起動ハンドラ ID 番号
cycact	[****]	周期起動ハンドラ活性状態

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	周期起動ハンドラ ID 番号
R2	周期起動ハンドラ活性状態
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER act_cyc (cycno, cycact);
```

《引数》

HNO	cycno;	周期起動ハンドラ指定番号
UINT	cycact;	周期起動ハンドラ活性状態

《戻り値》

関数の戻り値として常に E_OK を返します。

【エラーコード】

E_OK	00000000H(-H’00000000)	: 正常終了
E_NOEXS	0FFFFFFCCH(-H’00000034):	オブジェクトが存在していない

【機能説明】

cycno で示された周期起動ハンドラの活性状態を変更します。すなわち周期起動ハンドラを有効にしたり無効にしたりします。

cycno で指定した周期起動ハンドラが未定義の場合、エラーE_NOEXS を返します。

cyhact で指定できるのは以下の 3 通りです。

表 2.1　周期起動ハンドラの活性状態指定

C 言語	アセンブリ言語	意味
TCY_OFF	TCY_OFF	周期起動ハンドラを無効にします。
TCY_ON	TCY_ON	周期起動ハンドラを有効にします。
TCY_ON|TCY_INI	TCY_INI_ON	周期起動ハンドラを有効にすると同時に周期カウンタをクリアします。

周期起動ハンドラは、システムクロック割り込みハンドラの一部として実行されます[32]。

本システムコールはタスク、ハンドラのどちらからでも発行できます。

[32] すなわち、システムクロック割り込みハンドラからサブルーチンコールにより呼び出されます。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
        :
    act_cyc ( ID_cyc, TCY_ON );
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global   task
task:
    act_cyc   ID_cyc, TCY_INI_ON
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .include “mr32r.inc”
    .global   task
task:
    act_cyc   ID_cyc, TCY_INI_ON
        :
```

2.10.6. ref_cyc(Refer Cyclic Handler Status)

【システムコール名】

ref_cyc　→　周期起動ハンドラ状態を参照します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
ref_cyc  cycno
```

《引数》

cycno	[**]	周期起動ハンドラ指定番号
pk_rcyc	[****]	周期起動ハンドラの状態を返すパケットの先頭アドレス (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	周期起動ハンドラの ID 番号
R2	周期起動ハンドラの状態を返すパケットの先頭アドレス
R3	--

pk_rcyc の指す領域には、以下の情報が返されます。

オフセット	サイズ		
+0	4	exinf	拡張情報
+4	4	lftim	次のハンドラ起動までの残り時間
+8	4	cycact	周期起動ハンドラ活性状態

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER ref_cyc (pk_rcyc, cycno);
```

《引数》

T_RCYC	*pk_rcyc;	周期起動ハンドラの状態を返す構造体の先頭アドレス
HNO	cycno;	周期起動ハンドラ指定番号

《戻り値》

関数の戻り値としてエラーコードを返します 。

pk_rcyc の指す構造体には､以下のような周期起動ハンドラの状態が設定されます。

```
typedef struct t_rcyc {
    VP       exinf;   /* 拡張情報 */
    CYCTIME  lfttim;  /* 次にハンドラが起動されるまでの残り時間 */
    UINT     cycact;  /* 周期起動ハンドラ活性状態 */
} T_RCYC;
```

【エラーコード】

E_OK	00000000H(-H'00000000): 正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034): オブジェクトが存在していない

【機能説明】

cycno で指定した周期起動ハンドラの状態を参照し、その結果をリターンパラメータに返します。

●exinf

拡張情報を返します。

●cycact

cycact には、周期起動ハンドラ活性状態すなわち周期起動ハンドラが有効か(TCY_ON(=1))無効か(TCY_OFF(=0))を返します。

●lfttime

lftime には、次の周期起動ハンドラ起動までの残り時間を返します。周期起動ハンドラ起動までの残り時間はシステムクロックのカウント数で表します。

cycno で指定した周期起動ハンドラが未定義の場合、エラーE_NOEXS を返します。

本システムコールはタスク、ハンドラのどちらからでも発行できます。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
    T_RCYC rcyc;
    ref_cyc( &rcyc, ID_cyc );
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global task
task:
        :
    ld24     R2,#pk_rcyc
    ref_cyc   ID_cyc
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .include “mr32r.inc”
    .global task
task:
        :
    ld24     R2,#pk_rcyc
    ref_cyc   ID_cyc
        :
```

2.10.7. ref_alm(Refer Alarm Handler Status)

【システムコール名】

ref_alm　→　アラームハンドラ状態を参照します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
ref_alm  almno
```

《引数》

almno	[**]	アラームハンドラ ID 番号
pk_ralm	[****]	アラームハンドラ状態を返すパケットの先頭アドレス (R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	アラームハンドラの ID 番号
R2	アラームハンドラの状態を返すパケットの先頭アドレス
R3	--

pk_ralm の示す領域に、以下の情報が返されます。

オフセット	サイズ		
+0	4	exinf	拡張情報
+4	4	lftim	ハンドラ起動までの残り時間

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER ref_alm (pk_ralm, almno);
```

《引数》

HNO	almno;	アラームハンドラ指定番号
T_RALM	*pk_ralm ;	アラームハンドラ状態を返す構造体の先頭アドレス

《戻り値》

関数の戻り値としてエラーコードを返します。
構造体 pk_ralm に以下のようなアラームハンドラの状態が設定されます。

```
typedef struct t_ralm  {
    VP       exinf;  /*拡張情報*/
    ALMTIME  lfttim; /*ハンドラ起動までの残り時間*/
}T_RALM;
```

【エラーコード】

E_OK	00000000H(-H'00000000):	正常終了

【機能説明】

almno で指定したアラームハンドラの状態を参照し、その結果をリターンパラメータに返します。

●exinf

拡張情報を返します。

●lfttim

lfttim には対象となるアラームハンドラが起動するまでの残り時間を返します。アラームハンドラ起動までの残り時間は残りのシステムクロックの割り込みの発生回数を 48 ビットのデータで表します。

48 ビットのシステム時刻は、ltime、utime に分けて扱います。

本システムコールはタスク、ハンドラのどちらからでも発行できます。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void func()
{
    T_RALM   ralm;
    ref_alm( &ralm, ID_alarm );
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global task
task:
    ld24      R2,#pk_ralm
    ref_alm    ID_alm
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .include “mr32r.inc”
    .global     task
task:
    ld24      R2,#pk_ralm
    ref_alm    ID_alm
        :
```

2.11. システム管理機能

2.11.1. get_ver(Get Version Information)

【システムコール名】

get_ver　→　**MR32R** のバージョン番号を得ます

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
get_ver  pk_ver
```

《引数》

pk_ver　[****]　バージョン情報を返すパケットの先頭アドレスを示すラベル

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	--
R2	バージョン情報を返すパケットの先頭アドレス
R3	--

【 C 言語による呼び出し方法】

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

《引数》

T_VER　*pk_ver;　バージョン管理情報を返す構造体の先頭アドレス

《戻り値》

関数の戻り値として常に E_OK を返します。
pk_rtsk の指す構造体には、バージョン情報が設定されます。

【エラーコード】

E_OK　00000000H(-H'00000000)　:　正常終了

【機能説明】

MR32R のバージョン番号等の情報を得ます。バージョン番号は TRON 仕様で標準化された形式で得ます。したがって、異なるマイクロコンピュータ間でも、あるいは他の TRON 仕様のオペレーティングシステム間でも共通の形式で得ることができます。

得られるバージョン情報を以下に示します。

```
UH    maker      /* メーカー */
UH    id         /* 形式番号 */
UH    spver      /* 仕様書バージョン */
UH    prver      /* 製品バージョン */
UH    prno[4]    /* 製品管理情報 */
UH    cpu        /* CPU 情報 */
UH    var        /* バリエーション記述子*/
```

バージョン番号のフォーマットは次のようになります。

1. メーカー
   株式会社ルネサステクノロジを示す H'115 が返されます。
2. 形式番号
   MR32R の内部識別 ID H'221 が返されます。
3. 仕様書バージョン
   μITRON 仕様書 Ver3.02 に準拠していることを示す H'5302 が返されます。
4. 製品バージョン
   MR32R のバージョンを示す H'350 が返されます。
5. 製品管理情報

   ●prno[0]

   製品のリリース番号が得られます。
   prno[0] ← '01'

   ●prno[1]

   製品のリリース年の下 2 桁(西暦)と月が得られます。
   prno[1] ← 0x0306

   ●prno[2]

   拡張のための予約
   prno[2] ← 0x????

   ●prno[3]

   拡張のための予約
   prno[3] ← 0x????
6. CPU 情報
   M32R マイクロコンピュータを示す H'C31 が返されます。
7. バリエーション記述子
   MR32R のバリエーションを示す H'8000 が返されます。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
    T_VER   pk_ver;
    get_ver( &pk_ver );
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
ver:   .RES.H     10
    .include “mr32r.inc”
    .global   task
task:
         :
    get_ver ver
         :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
ver:   .space     20
    .include “mr32r.inc”
    .global   task
task:
         :
    get_ver ver
         :
```

2.11.2. ref_sys(Refer System Status)

【システムコール名】

ref_sys　→　CPU や OS の状態を参照します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
ref_sys
```

《引数》

pk_rsys　[****]　システム状態を返すパケットアドレス
(R2 レジスタに先頭アドレスを設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	--
R2	システム状態を返すパケットの先頭アドレス
R3	--

pk_rsys の指す領域には、以下の情報が返されます。

オフセット	サイズ		
+0	4	sysstat	システム状態
+4	2	runtskid	実行(RUN 状態)タスクの ID 番号
+6	2	runtskpri	実行(RUN 状態)タスクの優先度
+8	2	psw	PSW レジスタ

【 C 言語による呼び出し方法】

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

《引数》

T_RSYS　*pk_rsys;　システム状態を返す構造体の先頭アドレス

《戻り値》

関数の戻り値としてエラーコードを返す。
pk_rsys の指す構造体には、以下のようなシステム状態を設定します。

```
typdef strcut t_rsys {
    INT     sysstat;        /* システム状態 */
    ID      runtskid;       /* 実行(RUN 状態)タスクの ID 番号 */
    PRI     runtskpri;      /* 実行(RUN 状態)タスクの優先度 */
    UINT    psw;            /* プロセッサステータスワード */
} T_RSYS;
```

【エラーコード】

E_OK　00000000H(-H'00000000)　:　正常終了

【機能説明】

CPU や OS の実行状態を参照し、結果を pk_rsys の領域に返します。

●sysstat

システムの状態を示す。以下の示す値を返します。

sysstat:=(TSS_TSK||TSS_DDSP||TSS_LOC||TSS_INDP)

sysstat	値	システム状態	タスクディスパッチ	割り込み
TSS_TSK	0	タスク部	許可	許可
TSS_DDSP	1	タスク部	禁止	許可
TSS_LOC	2	タスク部	禁止	禁止
TSS_INDP	4	タスク独立部	禁止	禁止

●runtskid

現在実行中(RUN 状態)のタスク ID 番号を返します。

●runtskpri

現在実行中(RUN 状態)のタスクの優先度を返します。

●psw

現在実行中(RUN 状態)のタスクあるいはタスク独立部の PSW レジスタの値を返します。

本システムコールはタスク、ハンドラのどちらからでも発行できます。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task()
{
    T_RSYS rsys;
    ref_sys( &rsys );
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
pk_rsys: .RES.B 12
    .include mr32r.inc
    .global task
task:
        :
    ld24    R2,#pk_rsys
    ref_sys
        :
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
pk_rsys: .space 12
    .include mr32r.inc
    .global task
task:
        :
    ld24    R2,#pk_rsys
    ref_sys
        :
```

2.11.3. def_exc(Define Exception Handler)

【システムコール名】

def_exc　→　例外ハンドラを定義します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
def_exc  exckind
```

《引数》

exckind	[****]	例外の種類
pk_dexc	[****]	例外ハンドラの定義情報 (先頭アドレスを R2 レジスタに設定してください)

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	例外の種類
R2	例外ハンドラ情報を格納したパケットの先頭アドレス
R3	--

pk_dexc の指す領域には、以下に示す情報を設定してください。

オフセット	サイズ		
+0	4	excatr	例外ハンドラ属性
+4	4	exchdr	例外ハンドラアドレス
+8	2	tskid	タスク ID 番号
+12	4	excstksz	スタックサイズ

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER def_exc(exckind, pk_dexc);
```

《引数》

```
INT      exckind;        例外の種類
T_DEXC   *pk_dexc;       例外ハンドラの定義情報
```

pk_dexc の指す構造体に以下に示す情報を設定してください。

```
typedef struct t_dexc {
    ATR      excatr;          /* 例外ハンドラ属性 */
    FP       exchdr;          /* 例外ハンドラアドレス */
    ID       tskid;           /* タスク ID 番号 */
    W        excstksz         /* スタックサイズ */
} T_DEXC;
```

《戻り値》

関数の戻り値としてエラーコードを返します。
pk_dexc の内容

【エラーコード】

```
E_OK          00000000H(-H'00000000) :  正常終了
E_NOMEM       0FFFFFFF6H(-H'0000000a):  メモリ不足
E_NOEXS       0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
E_OBJ         0FFFFFFC1H(-H'0000003f):  オブジェクトの状態が不正
```

【機能説明】

exckind の例外に対する例外ハンドラを定義します。

exckind には、例外ハンドラの種類を定義します。**MR32R** では、強制例外(EXK_FEX=2)のみを指定してください。他の例外ハンドラ(CPU 例外、強制終了)を定義してもエラーにはなりません。

DORMANT 状態にあるタスクに対して本システムコールを発行した場合は、エラーE_OBJ を返します。

定義する例外ハンドラの情報 pk_dexc について以下に示します。

●excatr

例外ハンドラに使用するスタック領域を内蔵 RAM にするか外部 RAM にするかを指定します。

★ 内蔵 RAM を使用する場合

__MR_INT(0)を指定してください。

★ 外部 RAM を使用する場合

__MR_EXT(0x10000)を指定してください。

●exchdr

定義する例外ハンドラの開始アドレスを指定してください。

pk_dexc.exchdr=NADR=(-1)とした場合には、定義した例外ハンドラは解除されます。解除するとあらかじめ定義されているデフォルト例外ハンドラに変わります。また、定義を解除する前に再定義することもできます。

●tskid

ここで指定したタスクに対して例外ハンドラを定義します。例外ハンドラを定義するタスク ID 番号を指定してください。tskid=TSK_SELF=0 で自タスクの指定になります。なお、例外ハンドラ内で本システムコールを発行する場合、tskid=TSK_SELF の指定はできません。

●excstksz

定義する例外ハンドラのスタックサイズを指定してください。本システムコールで、例外ハンドラが使用するスタック領域のメモリを確保します。exchdr=NADR を指定すれば、スタック領域のメモリは解放されます。

指定したサイズ分のメモリが足りない場合は、エラーE_NOMEM を返します。

例外ハンドラのスタックは、タスク生成時に使用するスタック用メモリ領域からスタック領域を確保しますので、本システムコールを使用する場合は、必ず、コンフィグレーションファイルで int_memstk 定義または ext_memstk 定義を行ってください。

vras_fex によって強制例外ハンドラの起動要求があった例外ハンドラに対して、例外ハンドラの再定義を行った場合、再度 vras_fex システムコールを発行しなければいけません。発行せずに例外マスクがクリアされた場合、正常に動作しません。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよび、アラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#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)
{
        :
    /* 例外ハンドラ処理 */
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
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:
        :
    ; 例外ハンドラの処理
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
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:
        :
    ; 例外ハンドラの処理
        :
    ext_tsk
```

2.12. 拡張機能

2.12.1. vclr_ems(Clear Exception Mask)

【システムコール名】

vclr_ems　→　例外マスクをクリアします

【アセンブリ言語による呼び出し方法】

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

《引数》

tskid　　[**]　　例外マスクをクリアするタスク ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	例外マスクをクリアするタスク ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　　tskid;　　例外マスクをクリアするタスク ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない
E_OBJ	0FFFFFFC1H(-H'0000003f):	オブジェクトの状態が不正

【機能説明】

tskid で指定したタスクの例外マスクをクリアします。強制例外がペンディングされているタスクは、本システムコール発行により、例外マスクがクリアされ、それぞれの例外ハンドラが起動されます。

例外マスクをセットしている間に、複数回の強制例外の起動要求があっても例外ハンドラが起動されるのは、1 回のみです。

引数 tskid には、自タスクを指定することができます。tskid=TSK_SELF=0 で自タスクの指定になります。

RUN 状態あるいは DORMANT 状態以外の状態にあるタスクに対して本システムコールを発行した場合は、エラーE_OBJ となります。

また、未登録(NON-EXISTENT)状態のタスクに対して本システムコールを発行した場合は、エラーE_NOEXS となります。

本システムコールは、タスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task1(void)
{
        :
    vclr_ems( ID_task2 );
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global  task1
task1:
        :
    vclr_ems ID_task2
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .include “mr32r.inc”
    .global  task1
task1:
        :
    vclr_ems ID_task2
        :
    ext_tsk
```

2.12.2. vset_ems(Set Exception Mask)

【システムコール名】

vset_ems　→　例外マスクをセットします

【アセンブリ言語による呼び出し方法】

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

《引数》

tskid　[**]　例外マスクをセットするタスク ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	例外マスクをセットするタスク ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　tskid;　例外マスクをセットするタスク ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK        00000000H(-H'00000000) :  正常終了
E_NOEXS     0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
E_OBJ       0FFFFFFC1H(-H'0000003f):  オブジェクトの状態が不正
```

【機能説明】

tskid で指定したタスクの例外マスクをセットします。

例外マスクをセットされたタスクは、強制例外がペンディングされ、例外マスクがクリアされるまで、例外ハンドラの起動が遅延されます。

引数 tskid には、自タスクを指定することができます。tskid=TSK_SELF=0 で自タスクの指定になります。

RUN 状態あるいは DORMANT 状態以外の状態にあるタスクに対して本システムコールを発行した場合は、エラーE_OBJ となります。

また、未登録(NON-EXISTENT)状態のタスクに対して本システムコールを発行した場合は、エラーE_NOEXS となります。

なお、本システムコールを強制例外ハンドラから発行した場合は、対応したタスクに戻ります。

本システムコールは、タスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task1(void)
{
        :
    vset_ems( ID_task2 );
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global  task1
task1:
        :
    vset_ems ID_task2
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .include “mr32r.inc”
    .global  task1
task1:
        :
    vset_ems ID_task2
        :
    ext_tsk
```

2.12.3. vras_fex(Raise Forcibly Exception)

【システムコール名】

vras_fex　→　強制例外を起動します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
vras_fex tskid,exccd
```

《引数》

tskid	[**]	タスクの ID 番号
exccd	[****]	強制例外コード

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	タスク ID 番号
R2	強制例外コード
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER vras_fex (tskid,exccd);
```

《引数》

ID	tskid;	タスクの ID 番号
UW	exccd;	強制例外コード

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない
E_OBJ	0FFFFFFC1H(-H'0000003f):	オブジェクトの状態が不正

【機能説明】

tskid で指定したタスクに強制例外を起動します。

tskid で指定したタスクが未登録状態の場合は、エラーE_NOEXS を返します。

引数 tskid には、自タスクを指定できません。指定した場合は、エラーE_OBJ を返します。TSK_SELF も指定することはできません。指定した場合は、正常に動作しません。本システムコールを複数回発行しても、キューイングされません。例外ハンドラが起動されるまでに 2 回以上の強制例外起動要求があっても例外ハンドラの起動は、1 回のみです。

例外ハンドラには、例外コード exccd が、例外パラメータ pk_exc として渡されます。複数の強制例外の起動要求があった場合には、exccd の論理和がとられます。強制例外は、タスクの待ち状態や SUSPEND 状態を解除しません。待ち状態や SUSPEND 状態のタスクに本システムコールを発行しても実行状態になるまで例外ハンドラの起動は遅延されます。

本システムコールは、タスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task1()
{
        :
    vras_fex(ID_task2,0x3)
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
setpor: .space 16
    .include “mr32r.inc”
    .global   task1
task1:
        :
    vras_fex ID_task2,0x3
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
setpor: .space 16
    .include “mr32r.inc”
    .global   task1
task1:
        :
    vras_fex ID_task2,0x3
        :
    ext_tsk
```

2.12.4. vret_exc(Return Exception)

【システムコール名】

vret_exc　→　強制例外ハンドラか復帰します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
vret_exc
```

《引数》

なし

《レジスタ設定》

本システムコールを発行した例外ハンドラには戻りません。

【 C 言語による呼び出し方法】

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

《引数》

なし

《戻り値》

本システムコールを発行した例外ハンドラには戻りません。

【エラーコード】

なし

【機能説明】

強制例外ハンドラから例外発生したタスクに復帰します。

強制例外ハンドラから、戻る際に、例外発生時のタスクのコンテキストの状態に復帰します。

例外ハンドラを再度、起動するには、vras_fex システムコールを発行することで、例外ハンドラが再起動します。

本システムコールは、例外ハンドラからのみ発行してください。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task1()
{
        :
        :
}
void exc_hdr(void)
{
        :
        :
    vret_exc();
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global   exc_hdr
exc_hdr:
        :
        :
ret_exc
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .include “mr32r.inc”
    .global   exc_hdr
exc_hdr:
        :
        :
ret_exc
```

2.12.5. vrst_msg(Reset Message)

【システムコール名】

vrst_msg　→　メールボックスにあるメッセージをクリアします

【アセンブリ言語による呼び出し方法】

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

《引数》

mbxid　　[**]　　クリアするメールボックス ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	メールボックス ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　　mbxid;　　例外マスクをセットするタスク ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない

【機能説明】

mbxid で示されたメールボックスに格納されているメッセージをクリアします。
メッセージがないときは、何も行いません。対象メールボックスが存在しない場合は、E_NOEXS を返します。
本システムコールはタスク、ハンドラのどちらからでも発行できます。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task1(void)
{
        :
    vrst_msg( ID_mbx1 );
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global  task1
task1:
        :
    vrst_msg ID_mbx1
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .include “mr32r.inc”
    .global  task1
task1:
        :
    vrst_msg ID_mbx1
        :
    ext_tsk
```

2.12.6. vrst_blf (Reset Fixed-Memory Block)

【システムコール名】

vrst_blf　→　指定された固定長メモリブロックをすべて解放します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
vrst_blf blfid
```

《引数》

blfid　[**]　解放するメモリプール ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	解放するメモリプール ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　blfid;　解放するメモリプール ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK        00000000H(-H'00000000) :  正常終了
E_NOEXS     0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
```

【機能説明】

指定された固定長メモリプール ID 番号のメモリブロックをすべて解放します。

指定されたメモリプールが存在しない場合は、E_NOEXS を返します。また、指定されたメモリプールにたいして tget_blf,get_blf システムコールの発行により待ち状態になっているタスクが存在する場合、待ち状態になっているタスクの待ちを解除し、それらのタスクに対して EV_RST を返します。vrst_blf システムコールによって解放されたメモリプールは、待ち状態になっているタスクには割り当てられませんので注意してください。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよび、アラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task1()
{
        :
    vrst_blf(ID_mpf1);
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
setpor: .RES.B 16
    .include “mr32r.inc”
    .global   task1
task1:
        :
    vrst_blf ID_mpf1
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
setpor: .space 16
    .include “mr32r.inc”
    .global   task1
task1:
        :
    vrst_blf ID_mpf1
        :
    ext_tsk
```

2.12.7. vrst_blk(Reset Variable-Memory Block)

【システムコール名】

vrst_blk　→　可変長メモリブロックをすべて解放します。

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
vrst_blk blkid
```

《引数》

blkid　　[**]　　解放するメモリプール ID

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	解放するメモリプール ID
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　　blkid;　　解放するメモリプール ID

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない

【機能説明】

指定された ID 番号の可変長メモリプールのメモリブロックを全て解放します。

指定されたメモリプールが存在しない場合は、E_NOEXS を返します。また、指定されたメモリプールにたいして tget_blk,get_blk システムコールの発行により待ち状態になっているタスクが存在する場合、待ち状態になっているタスクの待ちを解除し、それらのタスクに対して EV_RST を返します。vrst_blk システムコールによって解放されたメモリプールは、待ち状態になっているタスクには割り当てられませんので注意してください。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよび、アラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task1()
{
        :
    vrst_blk(ID_mpl1);
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
setpor: .RES.B 16
    .include “mr32r.inc”
    .global   task1
task1:
        :
    vrst_blk ID_mpl1
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
setpor: .space 16
    .include “mr32r.inc”
    .global   task1
task1:
        :
    vrst_blk ID_mpl1
        :
    ext_tsk
```

2.12.8. vrst_mbf (Reset Message Buffer)

【システムコール名】

vrst_mbf　→　メッセージバッファをクリアします

【アセンブリ言語による呼び出し方法】

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

《引数》

mbfid　[**]　クリアするメッセージバッファ ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	メッセージバッファ ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　mbfid;　クリアするメッセージバッファ ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

```
E_OK        00000000H(-H'00000000) :  正常終了
E_NOEXS     0FFFFFFCCH(-H'00000034):  オブジェクトが存在していない
```

【機能説明】

mbfid で指定されたメッセージバッファをクリアします。指定されたメッセージバッファが存在しない場合は、E_NOEXS を返します。また、指定されたメモリプールに対して tsnd_mbf,snd_mbf システムコールの発行により待ち状態になっているタスクが存在する場合、待ち状態になっているタスクの待ちを解除し、それらのタスクに対して EV_RST を返します。vrst_mbf システムコールによってクリアされたメッセージバッファに対して、送信待ち状態のタスクは送信せずに、待ちが解除されますので注意してください。

本システムコールは、タスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task1()
{
        :
    vrst_mbf(ID_mbf1);
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
setpor: .space 16
    .include “mr32r.inc”
    .global   task1
task1:
        :
    vrst_mbf ID_mbf1
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
setpor: .space 16
    .include “mr32r.inc”
    .global   task1
task1:
        :
    vrst_mbf ID_mbf1
        :
    ext_tsk
```

2.13. 拡張機能(優先度付きメイルボックス)

2.13.1. vcre_mbx(Create Mailbox)

【システムコール名】

vcre_mbx → 優先度付きメイルボックスを生成します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
vcre_mbx      vmbxid
```

《引数》

vmbxid	[**]	生成するメイルボックスの ID 番号
pk_cvmbx	[****]	メイルボックス生成情報を格納した先頭アドレス (R2 レジスタに先頭アドレスを設定してください)

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	生成するメイルボックスの ID 番号
R2	生成情報を返すパケットの先頭アドレス
R3	--

pk_cmbx の指す領域には、以下に示す情報を設定します。

オフセット	サイズ		
+0	4	mbxatr	メイルボックス属性
+4	4	maxmpri	メッセージ優先度の最大値
+8	4	mprihd	優先度別のメッセージキューヘッダ領域の先頭番地

【C 言語による呼び出し方法】

```
#include <mr32r.h>
ER vcre_mbx (vmbxid, pk_cvmbx);
```

《引数》

ID	vmbxid;	生成するメイルボックス ID 番号
T_CVMBX	*pk_cvmbx;	メイルボックス生成情報

pk_cvmbx の指す構造体に以下に示す情報を設定します。

```
typedef struct t_cvmbx{
ATR         mbxatr;         /* メイルボックス属性 */
PRI         maxpri;         /* メッセージ優先度の最大値 */
VP          mprihd;         /* 優先度別のメッセージキューヘッ
                               ダ領域の先頭番地 */
} T_CVMBX;
```

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H’00000000) : 正常終了
E_OBJ	0FFFFFFC1H(-H’0000003f): オブジェクトの状態が不正

【機能説明】

vmbxid で示されたメイルボックスを生成します。
生成するメイルボックスの情報 pk_cvmbx は以下の通りです。

●mbxatr(メイルボックス属性)
メイルボックスの属性を指定する。以下の値が定義可能です。

メッセージの待ち方を指定する
TA_TFIFO、TA_TPRI のいずれかを指定することができます。
TA_TFIFO(=0x00)　　FIFO でつなぐ
TA_TPRI(=0x01)　　優先度順でつなぐ

メッセージの管理を指定する
TA_MFIFO、TA_MPRI のいずれかを指定することができます。
TA_MFIFO(=0x00)　FIFO で管理
TA_MPRI(=0x02)　　優先度で管理

●maxpri(メッセージの最大優先度)
メッセージ優先度の最大値を指定します。
●mprihd(メッセージ優先度のキューヘッダ先頭番地)
mprihd には NULL(=0)を指定してください。

すでに、存在するメイルボックスに対して、本システムコールを発行した場合は、エラー E_OBJ を返します。本システムコールの指定可能な ID 番号の範囲は、1 から最大項目数定義で設定したユーザシステムの最大メイルボックス数までです。
本システムコールはタスクからのみ発行できます。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
#define ID_vmbx1 1
T_CVMBX setmbx;
void task1()
{
    setmbx.mbxatr = 0x02;
    setmbx.maxpri = 10;
    setmbx.mprihd = NULL;
    vcre_mbx( ID_vmbx1, &setmbx );
    ext_tsk();
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
ID_vmbx1:  .equ,1
setmbx:    .RES.B 12
    .include “mr32r.inc”
task1:
        :
    ld24        R2,#setmbx
    ldi R1,#H’02
    st          R1,@R2
    ldi R1,#10
    st          R1,@(4,R2)
    ldi R1,#0
    st          R1,@(8,R2)
    vcre_mbx ID_vmbx1
        :
```

《 アセンブリ言語(TW32R)の使用例 》

```
    .equ ID_mbx1,1
    setmbx: .space 12
    .include “mr32r.inc”
    .global  task1
task1:
        :
    ld24   R2,#setmbx
    ld24        R2,#setmbx
    ldi R1,#H’02
    st          R1,@R2
    ldi R1,#10
    st          R1,@(4,R2)
    ldi R1,#0
    st          R1,@(8,R2)
    vcre_mbx ID_mbx1
        :
```

2.13.2. vdel_mbx(Delete Mailbox)

【システムコール名】

vdel_mbx　→　メイルボックスを削除します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
vdel_mbx vmbxid
```

《引数》

vmbxid　　[**]　削除するメイルボックスの ID 番号

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	削除するメイルボックスの ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID	vmbxid;	削除するメイルボックスの ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H’00000000) : 正常終了
E_NOEXS	0FFFFFFCCH(-H’00000034): オブジェクトが存在していない

【機能説明】

vmbxid で示されたメイルボックスを削除します。

削除されたメイルボックスは、再度、同じ ID 番号で新しいメイルボックスを生成できます。削除するメイルボックスに対して、条件成立を待っているタスクが存在しても、本システムコールは正常終了する。その時、待ち状態にあったタスクは、メッセージ待ち状態が解除され、エラーE_DLT を返し、実行可能(READY)状態に移行します。削除するメイルボックスの中にメッセージがあった場合、メッセージは消滅します。

存在しないメイルボックスに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

なお、本システムコールの発行は、vcre_mbx システムコールによって生成したメイルボックスに対して行わなければいけません。コンフィグレーションファイルで定義したメイルボックスに対して本システムコールを発行した場合は、正常に動作しません。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
#define ID_vmbx2 2
void task1(void)
{
        :
    vdel_mbx( ID_vmbx2 );
        :
    ext_tsk();
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
ID_vmbx2:       .equ    2
    .include “mr32r.inc”
    .global   task1
task1:
        :
    vdel_mbx ID_vmbx2
        :
    ext_tsk
```

《 アセンブリ言語(TW32R)の使用例 》

```
    .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)

【システムコール名】

vsnd_mbx　→　メッセージを送信します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
vsnd_mbx vmbxid
```

《引数》

vmbxid	[**]	送信を行うメイルボックスの ID 番号 (R2 レジスタにメッセージの先頭アドレスを設定してください)

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	送信を行うメイルボックスの ID 番号
R2	メッセージパケットの先頭アドレス
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER vsnd_mbx (vmbxid, pk_msg);
```

《引数》

ID	vmbxid;	メイルボックス ID 番号
T_MSG	pk_msg;	メッセージパケットの先頭アドレス[33]

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H’00000000): 正常終了
E_NOEXS	0FFFFFFCCH(-H’00000034): オブジェクトが存在していない

【機能説明】

mbxid で示されたメイルボックスにメッセージを送信します。

メッセージを待つタスクがないときには、メッセージはメッセージキューに格納します。対象メイルボックスで、受信を待っているタスクがある場合は、待ち行列の先頭タスクにメッセージパケットの先頭番地を渡し、そのタスクの待ち状態を解除します。このとき、待ち状態が解除されたタスクに対して、E_OK を返し、メイルボックスから受信したメッセージパケットの先頭番地として、pk_msg の値を返します。

受信を待っているタスクがない場合は、pk_msg を先頭番地とするメッセージパケットをメッセージキューに入れます。ここでメイルボックス属性に TA_MPRI(=0x02)が指定されている場合は、メッセージパケットを優先度順にメッセージキューに入れます。同じ優先度のメッセージの場合は、一番新しいメッセージを最後に入れます。また、メイルボックス属性に

[33]メッセージパケットの先頭アドレスを指定しますが、T_MSG 型に型変換してシステムコールを呼び出してください。

TA_MPRI(=0x02)が指定された場合は、pk_msg を先頭番地とするメッセージパケットの先頭に、T_MSG_PRI 型のメッセージヘッダがあるものと仮定し、そのメッセージの優先度を msgpri フィールドから取り出します。

メイルボックス属性に TA_MFIFO(=0x00)が指定された場合は、FIFO でメッセージキューに格納されます。すなわち、本システムコール発行によりメイルボックスに送信された順にメッセージが取り出されることになります。

存在しないメイルボックスに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスクからのみ発行できる。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行する場合は visnd_msg システムコールを使用しなければいけません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
typedef struct 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”);
    }
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .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
        :
```

《 アセンブリ言語(TW32R)の使用例 》

```
    .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)

【システムコール名】

visnd_mbx　→　メッセージを送信する（ハンドラ専用）

【アセンブリ言語による呼び出し方法】

```
.include        “mr32r.inc”
visnd_mbxvmbxid
```

《引数》

vmbxid　[**]　送信をおこなうメイルボックス ID 番号
(R2 レジスタにメッセージの先頭アドレスを設定してください。)

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	送信を行うメイルボックスの ID 番号
R2	メッセージパケットの先頭アドレス
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER visnd_mbx (vmbxid, pk_msg);
```

《引数》

ID	vmbxid;	メイルボックス ID 番号
T_MSG	pk_msg;	メッセージパケットの先頭アドレス[34]

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H’00000000): 正常終了
E_QOVR	0FFFFFFB7H(-H’00000049): キューイングまたはネストのオーバーフロー
E_NOEXS	0FFFFFFCCH(-H’00000034): オブジェクトが存在していない

【機能説明】

vsnd_mbx システムコールの機能をハンドラから利用する場合に、このシステムコールを使用します。

[34]メッセージパケットの先頭アドレスを指定しますが、T_MSG 型に型変換してシステムコールを呼び出してください。

【使用例】

《 C 言語の使用例 》

```
#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”);
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .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
```

《 アセンブリ言語(TW32R)の使用例 》

```
    .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)

【システムコール名】

vrcv_mbx　→　メッセージを受信します

【アセンブリ言語による呼び出し方法】

```
.include        “mr32r.inc”
vrcv_mbx        vmbxid
```

《引数》

vmbxid　[**]　受信するメイルボックスの ID 番号

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	受信を行うメイルボックスの ID 番号
R2	メッセージパケットの先頭アドレス
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER vrcv_mbx ( vmbxid ,ppk_msg );
```

《引数》

ID	vmbxid;	メイルボックス ID 番号
T_MSG	*ppk_msg;	メッセージパケットの先頭アドレスを格納する領域のアドレス[35]

《戻り値》

関数の戻り値としてエラーコードを返します。
ppk_msg が指す領域に、受信したメッセージパケットの先頭アドレスが設定されます。

【エラーコード】

E_OK	00000000H(-H’00000000) ：正常終了
E_RLWAI	0FFFFFFAAH(-H’00000056)：待ち状態強制解除
E_DLT	0FFFFFFAFH(-H’00000051)：待ちオブジェクトが削除された
E_NOEXS	0FFFFFFCCH(-H’00000034)：オブジェクトが存在していない

【機能説明】

vmbxid で示されたメイルボックスからメッセージを受信します。

対象メイルボックスにメッセージが到着している場合には、メッセージキューの先頭にあるメッセージを１つ取り出して、それをリターンパラメータ pk_msg として返します。

一方、そのメイルボックスにまだメッセージが送信されていない場合、本システムコールを発行したタスクは待ち状態となり、待ち行列につながれます。メイルボックス属性に TA_TFIFO(=0x00)が指定された場合は、自タスクを待ち行列の末尾につなげます。TA_TPRI (=0x01)が指定されている場合は、自タスクを優先度順で待ち行列につなぎます。同じ優先度

[35]メッセージパケットの先頭アドレス格納する領域のアドレスを指定しますが、T_MSG *型に型変換してシステムコールを呼び出してください。

のタスクの中では自タスクを待ち行列の最後につなぎます。

rel_wai、irel_wai システムコールによって待ち状態が解除された場合には、エラーE_RLWAI が返されます。また、他のタスクの発行した vdel_mbx システムコールによって条件成立を待っているメイルボックスが削除された場合、メッセージ待ち状態にあったタスクには、エラーE_DLT を返し、メッセージ待ち状態が解除され、実行可能(READY)状態に移行します。

存在しないメイルボックスに対して、本システムコールを発行した場合は、エラーE_NOEXS を返します。

本システムコールはタスクからのみ発行できます。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#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((T_MSG *)&msg, ID_vmbx) != E_OK )
        error(“forced wakeup¥n”);
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global  task
task:
        :
    vrcv_mbx ID_vmbx
        :
```

《 アセンブリ言語(TW32R)の使用例 》

```
    .include “mr32r.inc”
    .global  task
task:
        :
    vrcv_mbx ID_vmbx
        :
```

2.13.6. vtrcv_mbx(Receive Message with Timeout)

【システムコール名】

vtrcv_mbx　→　メッセージを受信します（タイムアウトあり）

【アセンブリ言語による呼び出し方法】

```
.include “ mr32r.inc ”
vtrcv_mbxvmbxid,tmout
```

《引数》

vmbxid	[**]	受信するメイルボックスの ID 番号
tmout	[****]	タイムアウト値

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	受信を行うメイルボックスの ID 番号
R2	メッセージパケットの先頭アドレス
R3	--
R4	タイムアウト値

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER vtrcv_mbx ( mbxid , ppk_msg, tmout);
```

《引数》

ID	mbxid;	メイルボックス ID 番号
T_MSG	*ppk_msg;	メッセージパケットの先頭アドレスを格納する領域のアドレス[36]
TMO	tmout	タイムアウト値

《戻り値》

関数の戻り値としてエラーコードを返します。
ppk_msg が指す領域に、受信したメッセージパケットの先頭アドレスが設定されます。

【エラーコード】

E_OK	00000000H(-H ’ 00000000) ：正常終了
E_TMOUT	0FFFFFFABH(-H ’ 00000055)：ポーリング失敗またはタイムアウト
E_RLWAI	0FFFFFFAAH(-H ’ 00000056)：待ち状態強制解除
E_DLT	0FFFFFFAFH(-H ’ 00000051)：待ちオブジェクトが削除された
E_NOEXS	0FFFFFFCCH(-H ’ 00000034)：オブジェクトが存在していない

【機能説明】

mbxid で示されたメイルボックスにメッセージがあれば受信します。そのメイルボックスにメッセージが入っている場合には、メッセージキューの先頭にあるメッセージを１つ取り出して、それをリターンパラメータ ppk_msg として返されます。

[36]メッセージパケットの先頭アドレス格納する領域のアドレスを指定しますが、T_MSG *型に型変換してシステムコールを呼び出してください。

一方、そのメイルボックスに、まだメッセージが送信されていない場合には、本システムコールを発行したタスクは、待ち状態となり、メッセージ待ち行列とタイムアウト待ち行列の 2 つの待ち行列につながれます。メイルボックス属性に TA_TFIFO(=0x00)が指定された場合は、自タスクを待ち行列の末尾につながれます。TA_TPRI (=0x01)が指定されている場合は、自タスクは優先度順で待ち行列につながれます。同じ優先度のタスクの中では自タスクは待ち行列の最後につながれます。

本システムコール実行による待ち状態は、以下に示す場合に解除されます。なお、待ち状態が解除されると本システムコールを発行したタスクはメッセージ待ち行列とタイムアウト待ち行列の 2 つの待ち行列からはずされ、レディキューに接続されます。

●tmout の時間が経過する前にメッセージが到着した場合

この時のエラーコードは、E_OK を返します。

●メッセージが到着しないまま、tmout の時間が経過した場合

この時のエラーコードは、E_TMOUT を返します。

●他のタスクおよびハンドラから発行した rel_wai、irel_wai システムコールによって待ち状態が強制解除された場合

この時のエラーコードは、E_RLWAI を返します。

●他のタスクから発行した del_mbx システムコールによって待ち状態の対象となっているメイルボックスが削除された場合

この時のエラーコードは、E_DLT を返します。

tmout には、-1 から 7FFFFFFFH まで指定できます。なお、tmout に TMO_POL=0 を指定した場合は、タイムアウト値として 0 を指定したことを示し、vprcv_mbx と同じ動作を行います。また、tmout=TMO_FEVR(-1)にした場合は、永久待ちの指定で、vrcv_mbx システムコールと同じ動作を行います。

本システムコールはタスクからのみ発行できます。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#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((T_MSG *)&msg, ID_mbx, 10 ) != E_OK ){
        error(“Can’t Get Message¥n”);
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global task
task:
        :
    vtrcv_mbx  ID_mbx, 10
        :
```

《 アセンブリ言語(TW32R)の使用例 》

```
    .include “mr32r.inc”
    .global task
task:
        :
    vtrcv_mbx  ID_mbx, 10
        :
```

2.13.7. vprcv_mbx(Poll and Receive Message)

【システムコール名】

vprcv_mbx　→　メッセージを受信します（待ちなし）

【アセンブリ言語による呼び出し方法】

```
.include “ mr32r.inc ”
vprcv_mbxvmbxid
```

《引数》

vmbxid　　[**]　受信するメイルボックスの ID 番号

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	受信を行うメイルボックスの ID 番号
R2	メッセージパケットの先頭アドレス
R3	--

【 C 言語による呼び出し方法】

```
#include <mr32r.h>
ER vprcv_mbx ( vmbxid , ppk_msg );
```

《引数》

ID	vmbxid;	メイルボックス ID 番号
T_MSG	*ppk_msg;	メッセージパケットの先頭アドレスを格納する領域のアドレス[37]

《戻り値》

関数の戻り値としてエラーコードを返します。
ppk_msg が指す領域に、受信したメッセージパケットの先頭アドレスが設定されます。

【エラーコード】

E_OK	00000000H(-H ’ 00000000) ：正常終了
E_TMOUT	0FFFFFFABH(-H ’ 00000055)：ポーリング失敗またはタイムアウト
E_NOEXS	0FFFFFFCCH(-H ’ 00000034)：オブジェクトが存在していない

【機能説明】

vmbxid で示されたメイルボックスからメッセージがあれば受信します(待ちなし)。そのメイルボックスにメッセージが入っている場合には、メッセージキューの先頭にあるメッセージを１つ取り出して、それをリターンパラメータ ppk_msg として返します。

一方、そのメイルボックスに、まだメッセージが送信されていない場合には、システムコール発行タスクにエラーE_TMOUT を返し、本システムコールを終了します。vrcv_mbx、

[37]メッセージパケットの先頭アドレス格納する領域のアドレスを指定しますが、T_MSG *型に型変換してシステムコールを呼び出してください。

vtrcv_mbx とは、異なり本システムコールを発行したタスクは、待ち状態には移行しません。
また、存在しないメイルボックスに対して、本システムコールを発行した場合は、エラー E_NOEXS を返します。
メッセージ受信時の注意事項は、vrcv_mbx を参照してください。
本システムコールはタスク、タスク独立部(割り込みハンドラ、周期起動ハンドラおよびアラームハンドラ)のどちらからでも発行できます。

【使用例】

《 C 言語の使用例 》

```
#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((T_MSG *)&msg, ID_mbx) != E_OK ){
        error(“Can’t Get Message¥n”);
                :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global  task
task:
    vprcv_mbx ID_mbx1
        :
```

《 アセンブリ言語(TW32R)の使用例 》

```
    .include “mr32r.inc”
    .global  task
task:
    vprcv_mbx ID_mbx1
        :
```

2.13.8. vref_mbx(Refer Mailbox Status)

【システムコール名】

vref_mbx　→　メイルボックス状態を参照します

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
vref_mbx vmbxid
```

《引数》

vmbxid	[**]	状態を参照するメイルボックス ID 番号
pk_rmbx	[****]	メイルボックス状態を返すパケットの先頭アドレス (R2 レジスタに先頭アドレスを設定してください)

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	参照するメイルボックスの ID 番号
R2	メイルボックスの状態を返すパケットの先頭アドレス
R3	--

pk_rmbx の指す領域には、以下の情報が返されます。

オフセット	サイズ		
+0	2	wtsk	待ちタスクの有無
+4	4(U)	pk_msg	次に受信されるメッセージパケットの先頭アドレス

U は、unsigned のデータを表わします。

【C 言語による呼び出し方法】

```
#include <mr32r.h>
ER vref_mbx (pk_rmbx, vmbxid);
```

《引数》

T_RMBX	*rmbx;	メイルボックス状態を返す構造体の先頭アドレス
ID	vmbxid;	メイルボックス ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します 。
pk_rmbx の指す構造体には、以下の情報が返されます。

```
typedef struct t_rmbx {
BOOL_ID     wtsk;          /*待ちタスクの有無*/
T_MSG       *pk_msg; /*次に受信されるメッセージパケットの先頭アドレス*/
} T_RVMBX;
```

【エラーコード】

E_OK	00000000H(-H’00000000): 正常終了
E_NOEXS	0FFFFFFCCH(-H’00000034): オブジェクトが存在していない

【機能説明】

本システムコールでは、mbxid で指定したメイルボックスの以下の状態を返します。

- wtsk

  wtsk には、指定したメイルボックスでメッセージを待っている先頭タスク ID 番号を返します。メッセージを待っているタスクがない場合は TSK_NON(=0)を返します。

- pk_msg

  pk_msg には、次に rcv_msg を実行した場合に受信されるメッセージ(キューイングされている先頭メッセージ)を返す。メッセージがない場合は、NULL(=0)を返します。

存在しないメイルボックスに対して、本システムコールを発行した場合は、エラー E_NOEXS を返します。

本システムコールはタスク、タスク独立部(割り込みハンドラ、周期起動ハンドラ、アラームハンドラ)のどちらからでも発行できます。

【使用例】

《 C 言語の使用例 》

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

《 アセンブリ言語(CC32R)の使用例 》

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

《 アセンブリ言語(TW32R)の使用例 》

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

2.13.9. vrst_mbx (Reset Mailbox)

【システムコール名】

vrst_mbx　→　優先度付きメールボックスにあるメッセージをクリアします

【アセンブリ言語による呼び出し方法】

```
.include “mr32r.inc”
vrst_mbx vmbxid
```

《引数》

vmbxid　[**]　クリアするメールボックス ID 番号

《レジスタ設定》

レジスタ名	システムコール発行後の内容
R0	エラーコード
R1	メールボックス ID 番号
R2	--
R3	--

【 C 言語による呼び出し方法】

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

《引数》

ID　vmbxid;　クリアするメールボックス ID 番号

《戻り値》

関数の戻り値としてエラーコードを返します。

【エラーコード】

E_OK	00000000H(-H'00000000) :	正常終了
E_NOEXS	0FFFFFFCCH(-H'00000034):	オブジェクトが存在していない

【機能説明】

vmbxid で示されたメールボックスに格納されているメッセージをクリアします。

メッセージがないときは、何も行いません。対象メールボックスが存在しない場合は、E_NOEXS を返します。

本システムコールはタスクからのみ発行してください。割り込みハンドラ、周期起動ハンドラおよびアラームハンドラから発行した場合は、正常に動作しません。

【使用例】

《 C 言語の使用例 》

```
#include <mr32r.h>
#include “id.h”
void task1(void)
{
        :
    vrst_mbx( ID_vmbx1 );
        :
}
```

《 アセンブリ言語(CC32R)の使用例 》

```
    .include “mr32r.inc”
    .global  task1
task1:
        :
    vrst_mbx ID_vmxbx1
        :
    ext_tsk
```

《 アセンブリ言語(TW32R・D-CC/M32R)の使用例 》

```
    .include “mr32r.inc”
    .global  task1
task1:
        :
    vrst_mbx ID_vmbx1
        :
    ext_tsk
```

第3章　付録

3.1. システムコール一覧

タスク管理機能システムコール

システムコール名		機能	スケジューラ
cre_tsk	[E]	タスクを生成します	起動
del_tsk	[E]	タスクを削除します	起動
sta_tsk	[S]	タスクを起動します	起動
ista_tsk	[S]	タスクを起動します(ハンドラ専用)	--
ext_tsk	[S]	自タスクを正常終了します	起動
exd_tsk	[E]	自タスクを終了後削除します	起動
ter_tsk	[S]	他タスクを強制的に異常終了します	起動
chg_pri	[S]	タスクの優先度を変更します	起動
ichg_pri	[S]	タスクの優先度を変更します(ハンドラ専用)	--
dis_dsp	[S]	タスクのディスパッチを禁止します	--
ena_dsp	[S]	タスクのディスパッチを許可します	起動
rot_rdq	[S]	レディキューを回転します	起動
irot_rdq	[S]	レディキューを回転します(ハンドラ専用)	--
rel_wai	[S]	待ち状態を強制解除します	起動
irel_wai	[S]	待ち状態を強制解除します(ハンドラ専用)	--
get_tid	[S]	自タスクの ID を得ます	--
ref_tsk	[E]	タスクの状態を参照します	--

タスク付属同期機能システムコール

システムコール名		機能	スケジューラ
sus_tsk	[S]	タスクを強制待ち状態を移行します	起動
isus_tsk	[S]	タスクを強制待ち状態を移行します(ハンドラ専用)	--
rsm_tsk	[S]	強制待ち状態のタスクを再開する	起動
irsm_tsk	[S]	強制待ち状態のタスクを再開する(ハンドラ専用)	--
slp_tsk	[R]	タスクを待ち状態へ移行する	起動
tslp_tsk	[E]	タスクを一定時間待ち状態へ移行する	起動
wup_tsk	[R]	待ち状態のタスクを起床します	起動
iwup_tsk	[R]	待ち状態のタスクを起床します(ハンドラ専用)	--
can_wup	[S]	タスクの起床要求を無効にします	--

同期・通信機能システムコール

システムコール名		機能	スケジューラ
cre_flg	[E]	イベントフラグを生成します	起動
del_flg	[E]	イベントフラグを削除します	起動
set_flg	[S]	イベントフラグをセットします	起動
iset_flg	[S]	イベントフラグをセットします (ハンドラ専用)	--
clr_flg	[S]	イベントフラグをクリアします	--
wai_flg	[S]	イベントフラグを待ちます	起動
twai_flg	[E]	イベントフラグを待ちます(タイムアウトあり)	起動
pol_flg	[S]	イベントフラグを得ます(待ちなし)	--
ref_flg	[E]	イベントフラグの状態を参照します	--
cre_sem	[E]	セマフォを生成します	起動
del_sem	[E]	セマフォを削除します	起動
sig_sem	[R]	セマフォ資源を返却します	起動
isig_sem	[R]	セマフォ資源を返却します(ハンドラ専用)	--
wai_sem	[R]	セマフォ資源を獲得します	起動
twai_sem	[E]	セマフォ資源を獲得します(タイムアウトあり)	起動
preq_sem	[R]	セマフォ資源を獲得します(ポーリング)	--
ref_sem	[E]	セマフォの状態を参照します	--
cre_mbx	[E]	メイルボックスを生成します	起動
del_mbx	[E]	メイルボックスを削除します	起動
snd_msg	[S]	メッセージを送信します	起動
isnd_msg	[S]	メッセージを送信します(ハンドラ専用)	--
rcv_msg	[S]	メッセージを受信します	起動
trcv_msg	[E]	メッセージを受信します(タイムアウトあり)	起動
prcv_msg	[S]	メッセージを受信します(待ちなし)	--
ref_mbx	[E]	メイルボックスの状態を参照します	--

拡張同期・通信機能

システムコール名		機能	スケジューラ
cre_mbf	[E]	メッセージバッファを生成します	起動
del_mbf	[E]	メッセージバッファを削除します	起動
snd_mbf	[E]	メッセージバッファへ送信します	起動
tsnd_mbf	[E]	メッセージバッファへ送信します(タイムアウトあり)	起動
psnd_mbf	[E]	メッセージバッファへ送信します(待ちなし)	--
rcv_mbf	[E]	メッセージバッファから受信します	起動
trcv_mbf	[E]	メッセージバッファから受信します(タイムアウトあり)	起動
prcv_mbf	[E]	メッセージバッファから受信します(待ちなし)	起動
ref_mbf	[E]	メッセージバッファの状態を参照します	--
cre_por	[E]	ランデブ用ポートを生成します	起動
del_por	[E]	ランデブ用ポートを削除します	起動
cal_por	[E]	ポートに対するランデブ呼出を行います	起動
tcal_por	[E]	ポートに対するランデブ呼出を行います(タイムアウトあり)	起動
pcal_por	[E]	ポートに対するランデブ呼出を行います(待ちなし)	起動
acp_por	[E]	ポートに対するランデブ受付を行います	起動
tacp_por	[E]	ポートに対するランデブ受付を行います(タイムアウトあり)	起動
pacp_por	[E]	ポートに対するランデブ受付を行います(待ちなし)	起動
fwd_por	[E]	ランデブを別のポートに回送します	起動
rpl_rdv	[E]	ランデブ返答を行います	起動
ref_por	[E]	ランデブ用ポートを参照します	起動

割り込み管理機能

システムコール名		機能	スケジューラ
def_int	[C]	割り込みハンドラを定義します	起動
ret_int	[R]	割り込みハンドラから復帰します	起動
loc_cpu	[R]	割り込みとディスパッチを禁止します	--
unl_cpu	[R]	割り込みとディスパッチを許可します	起動

メモリプール管理機能

システムコール名		機能	スケジューラ
cre_mpf	[E]	固定長メモリプールを生成します	起動
del_mpf	[E]	固定長メモリプールを削除します	起動
get_blf	[E]	メモリブロックを獲得します	起動
tget_blf	[E]	メモリブロックを獲得します(タイムアウトあり)	起動
pget_blf	[E]	メモリブロックを獲得します(待ちなし)	--
rel_blf	[E]	メモリブロックを解放します	起動
irel_blf	[E]	メモリブロックを解放します	--
ref_mpf	[E]	固定長メモリプールの状態を参照します	--
cre_mpl	[E]	可変長メモリプールを生成します	起動
del_mpl	[E]	可変長メモリプールを削除します	起動
get_blk	[E]	メモリブロックを獲得します	起動
tget_blk	[E]	メモリブロックを獲得します(タイムアウトあり)	起動
pget_blk	[E]	メモリブロックを獲得します(待ちなし)	起動
rel_blk	[E]	メモリブロックを解放します	起動
ref_mpl	[E]	可変長メモリプールの状態を参照します	--

時間管理機能

システムコール名		機能	スケジューラ
set_tim	[S]	システムクロックを設定します	--
get_tim	[S]	システムクロックを参照します	--
dly_tsk	[S]	タスクの実行を一定時間遅延します	起動
def_cyc	[E]	周期起動ハンドラを定義します	--
act_cyc	[E]	周期起動ハンドラの活性制御を行います	--
ref_cyc	[E]	周期起動ハンドラの状態を参照します	--
ref_alm	[E]	アラームハンドラの状態を参照します	--

システム管理機能

システムコール名		機能	スケジューラ
get_ver	[R]	OS のバージョンを参照します	--
ref_sys	[E]	システムの状態を参照します	--
def_exc	[C]	例外ハンドラの定義を行います	起動

拡張機能

システムコール名		機能	スケジューラ
vclr_ems	[--]	例外マスクをクリアします	起動
vset_ems	[--]	例外マスクをセットします	起動
vras_fex	[--]	強制例外ハンドラを起動します	起動
vrst_msg	[--]	メールボックスをクリアします	--
vrst_mbf	[--]	メッセージバッファをクリアします	起動
vrst_blf	[--]	固定長メモリプールを解放します	起動
vrst_blk	[--]	可変長メモリプールを解放します	起動

拡張機能(優先度付きメールボックス機能)

システムコール名		機能	スケジューラ
vcre_mbx	[--]	優先度付きメールボックスを生成します	起動
vdel_mbx	[--]	優先度付きメールボックスを削除します	起動
vsnd_mbx	[--]	優先度付きメッセージを送信します	起動
visnd_mbx	[--]	優先度付きメッセージを送信します(ハンドラ専用)	--
vrcv_mbx	[--]	優先度付きメッセージを受信します(タイムアウトなし)	起動
vtrcv_mbx	[--]	優先度付きメッセージを受信します(タイムアウトあり)	起動
vprcv_mbx	[--]	優先度付きメッセージを受信します(まちなし)	--
vrst_mbx	[--]	優先度付きメールボックスをクリアします	--
vref_mbx	[--]	優先度付きメールボックスの状態を参照します	--

3.2. エラーコード一覧

エラーコード	値	説明
E_OK	00000000H(-H'00000000)	正常終了
E_OBJ	0FFFFFFC1H(-H'0000003F)	オブジェクトの状態が不正
E_QOVR	0FFFFFFB7H(-H'00000049)	キューイングまたはネストのオーバーフロー
E_TMOUT	0FFFFFFABH(-H'00000055)	ポーリング失敗またはタイムアウト
E_RLWAI	0FFFFFFAAH(-H'00000056)	待ち状態強制解除
E_NOEXS	0FFFFFFCCH(-H'00000034)	オブジェクトが存在していない
E_DLT	0FFFFFFAFH(-H'00000051)	待ちオブジェクトが削除された
E_NOMEM	0FFFFFFF6H(-H'0000000A)	メモリ不足
EV_RST	0FFFFFF02H(-H'000000FE)	リセットのため待ちが解除された

3.3. アセンブリ言語インタフェース

アセンブリ言語でシステムコールを発行する場合、システムコールの呼び出し用マクロを使用します。システムコールの呼び出し用マクロ内の処理は、各パラメータをレジスタに設定してから、ソフトウエア割り込みによりシステムコールのルーチンの実行を開始します。また、システムコールの呼び出し用マクロを使用せず直接システムコールを呼び出した場合、将来のバージョンにおいて互換性が保証できなくなります。以下にアセンブリ言語インタフェースの一覧表を記載します。機能コードについては、μITRON 仕様で規定された値は使用しておりません。

タスク管理機能

システムコール名	TRAP番号	引数			戻り値	
		R0 機能コード	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'6c	tskid		ercd	
get_tid	#8	H'70			ercd	tskid
ref_tsk	#8	H'd4	tskid	pk_rtsk	ercd	

タスク付属同期機能

システムコール名	TRAP番号	引数					戻り値	
		R0 機能コード	R1	R2	R3	R4	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				-1	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

同期・通信機能

システムコール名	TRAP番号	引数					戻り値		
		R0 機能コード	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	-1	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			-1	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			-1	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'e0	mbxid	pk_rmbx			ercd		

拡張同期・通信機能

システムコール名	TRAP番号	引数							戻り値		
		R0 機能コード	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	-1			ercd		
tsnd_mbf	#7	H'c8	mbfid	msg	msgsz	tmout			ercd		
psnd_mbf	#7	H'c8	mbfid	msg	msgsz	0			ercd		
rcv_mbf	#7	H'124	mbfid	msg		-1			ercd		msgsz
trcv_mbf	#7	H'124	mbfid	msg		tmout			ercd		msgsz
prcv_mbf	#7	H'124	mbfid	msg		0			ercd		msgsz
ref_mbf	#8	H'114	mbfid	pk_rmbf					ercd		
cre_por	#7	H'144	porid	pk_cpor					ercd		
del_por	#7	H'148	porid						ercd		
cal_por	#7	H'14c	porid		cmsgsz	-1	msg	calptn	ercd	rmsgsz	
tcal_por	#7	H'14c	porid		cmsgsz	tmout	msg	calptn	ercd	rmsgsz	
pcal_por	#7	H'14c	porid		cmsgsz	0	msg	calptn	ercd	rmsgsz	
acp_por	#7	H'150	porid			-1	msg	acpptn	ercd	cmsgsz	rdvno
tacp_por	#7	H'150	porid			tmout	msg	acpptn	ercd	cmsgsz	rdvno
pacp_por	#7	H'150	porid			0	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		

割り込み管理機能

システムコール名	TRAP番号	引数			戻り値	
		R0 機能コード	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

メモリプール管理機能

システムコール名	TRAP番号	引数					戻り値	
		R0 機能コード	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	
irel_blf	#7	H'17c	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	

時間管理機能

システムコール名	TRAP番号	引数			戻り値
		R0 機能コード	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

システム管理機能

システムコール名	TRAP番号	引数			戻り値
		R0 機能コード	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

拡張機能

システム コール名	TRAP 番号	引数			戻り値
		R0 機能コード	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		

拡張機能(優先度付きメールボックス機能)

システム コール名	TRAP 番号	引数					戻り値	
		R0 機能コード	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 言語インタフェース

タスク管理機能

```
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);
```

タスク付属同期機能

```
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)
```

同期・通信機能

```
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);
```

拡張同期・通信機能

```
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);
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);
```

割り込み管理機能

```
ER    ercd =  def_int    (UINT dintno, T_DINT *pk_dint);
      void    ret_int    ();
ER    ercd =  loc_cpu    ();
ER    ercd =  unl_cpu    ();
```

メモリプール管理機能

```
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 =  irel_blf   (ID mpfid);
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);
```

時間管理機能

```
ER    ercd =  set_tim    (SYSTIME *pk_tim);
ER    ercd =  get_tim    (SYSTIME *pk_tim);
ER    ercd =  dly_tsk    (DLYTIMe dlytim);
ER    ercd =  def_cyc    (HNO cycno, T_DCYC pk_dcyc);
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);
```

システム管理機能

```
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);
```

拡張機能

```
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);
ER    ercd =
```

拡張機能(優先度付きメールボックス)

```
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_mbx  (ID vmbxid, T_MSG pk_msg);
ER   ercd =  vrcv_mbx   (T_MSG *ppk_msg, ID vmbxid);
ER   ercd =  vtrcv_mbx  (T_MSG *ppk_msg, ID vmbxid, TMO tmout);
ER   ercd =  vprcv_mbx  (T_MSG *ppk_msg, ID vmbxid);
ER   ercd =  vref_mbx   (T_RVMBF *pk_rvmbf,ID vmbxid);
ER   ercd =  vrst_mbx   (ID vmbxid);
```

3.5. データタイプ

```
typedef    char             B;          /* 符号付き 8 ビット整数 */
typedef    short            H;          /* 符号付き 16 ビット整数 */
typedef    long             W;          /* 符号付き 32 ビット整数 */
typedef    unsigned char    UB;         /* 符号なし 8 ビット整数 */
typedef    unsigned short   UH;         /* 符号なし 16 ビット整数 */
typedef    unsigned long    UW;         /* 符号なし 32 ビット整数 */
typedef    char             VB          /* データタイプが一致しないもの符号付き
                                           (8 ビットサイズ) */
typedef    short            VH;         /* データタイプが一致しないもの符号付き
                                           (16 ビットサイズ) */
typedef    long             VW;         /* データタイプが一致しないもの符号付き
                                           (32 ビットサイズ) */
typedef    void             *VP;        /* データタイプが一致しないものへのポインタ */
typedef    void             (*FP)();    /* プログラムのスタートアドレス一般 */
typedef    W                INT         /* 符号付き 32 ビット整数 */
typedef    UW               UINT;       /* 符号なし 32 ビット整数 */
typedef    W                RNO         /* ランデブ番号 */
typedef    H                ID;         /* オブジェクト ID 番号 */
typedef    H                PRI;        /* タスク優先度 */
typedef    H                TMO;        /* タイムアウト */
typedef    H                HNO;        /* ハンドラ番号 */
typedef    INT              ER;         /* エラーコード(符号付き整数) */
typedef    INT              ATR;        /* オブジェクト属性(符号なし整数) */
typedef    INT              DLYTIME;    /* 遅延時間 */
typedef    INT              CYCTIME;    /* 周期起動ハンドラの起動間隔*/
typedef    H                BOOL_ID;    /* ブール値または ID 番号 */
typedef    UINT             PSW;        /* PSW 値 */
typedef    void             *PT_MSG;    /* メッセージデータ */
```

3.6. 共通定数と構造体のパケット形式

```
----共通----
NADR        -1          /* アドレスやポインタ値が無効 */
TRUE        1           /* 真 */
FALSE       0           /* 偽 */
----タスク管理関係----
typedef struct t_ctsk {
        VP      exinf;  /* 拡張情報 */
        ATR     tskatr; /* タスク属性 */
        FP      task;   /* タスク起動アドレス */
        PRI     itskpri; /* タスク起動時優先度 */
        INT     stksz;  /* スタックサイズ */
} T_CTSK;
TSK_SELF    0           /* 自タスク指定 */
TPRI_RUN    0           /* その時実行中の優先度を指定 */
typedef struct t_rtsk {
        VP      exinf;  /* 拡張情報 */
        PRI     tskpri; /* 現在の優先度 */
        UINT    tskstat; /* タスクの状態 */
        UINT    tskwait; /* タスクの待ち要因 */
        ID      wid;    /* タスクの待ちオブジェクト ID */
        INT     wupcnt; /* 起床要求キューイング数 */
        ATR     tskatr; /* タスク属性 */
        FP      task;   /* タスク起動アドレス */
        PRI     itskpri; /* タスク起動時アドレス */
        INT     stksz;  /* スタックサイズ */
        UW      epndptn; /* 例外ペンディングパターン */
} T_RTSK;
----セマフォ関係----
typedef struct t_csem {
        VP      exinf;  /* 拡張情報 */
        ATR     sematr; /* セマフォ属性 */
        INT     isemcnt; /* セマフォの初期値 */
        INT     maxsem; /* セマフォの最大値 */
} T_CSEM;
typedef struct t_rsem {
        VP      exinf;  /* 拡張情報 */
        BOOL_ID wtsk;   /* 待ちタスクの有無 */
        INT     semcnt; /* 現在のセマフォカウントの値 */
} T_RSEM;
```

```
----イベントフラグ関係----
typedef struct t_cflg {
        VP      exinf;   /* 拡張情報 */
        ATR     flgatr;  /* イベントフラグ属性 */
        UINT    iflgptn; /* イベントフラグの初期値 */
} T_CFLG;
wfmod:
        TWF_ANDW   H'0000    /* AND 待ち */
        TWF_ORW    H'0002    /* OR 待ち */
        TWF_CLR    H'0001    /* クリア指定 */
typedef struct t_rflg {
        VP      exinf;    /* 拡張情報 */
        BOOL_ID wtsk;     /* 待ちタスクの有無 */
        UINT    flgptn;   /* イベントフラグの初期値 */
} T_RFLG;
----メイルボックス関係----
typedef struct t_cmbx {
        VP      exinf;   /* 拡張情報 */
        ATR     mbxatr;  /* メイルボックス属性 */
        INT     bufcnt;  /* リングバッファのサイズ */
} T_CMBX;
typedef struct t_rmbx {
        VP      exinf;    /* 拡張情報 */
        BOOL_ID wtsk;     /* 待ちタスクの有無 */
        T_MSG   pk_msg;   /* 次に受信されるメッセージ */
        INT     msgcnt;   /* メイルボックスに格納されているメッセージ数 */
} T_RMBX;
----メッセージバッファ関係----
typedef struct t_cmbf {
        VP      exinf;   /* 拡張情報 */
        ATR     mbfatr;  /* メッセージバッファ属性 */
        INT     bufsz;   /* メッセージバッファのサイズ */
        INT     maxmsz;  /* メッセージの最大長 */
} T_CMBF;
typedef struct t_rmbf {
        VP      exinf;   /* 拡張情報 */
        BOOL_ID wtsk;     /* 受信待ちタスクの有無 */
        BOOL_ID stsk;     /* 送信待ちタスクの有無 */
        INT     msgsz     /* 次に受信されるメッセージのサイズ */
        INT     frbufsz; /* 空きバッファのサイズ */
} T_RMBF;
----ポート、ランデブ関係----
typedef struct t_cpor {
        VP      exinf;   /* 拡張情報 */
        ATR     poratr;  /* ポート属性 */
        INT     maxcmsz; /* 呼出時のメッセージの最大数 */
        INT     maxrmsz; /* 返答時のメッセージの最大数 */
} T_CPOR;
typedef struct t_rpor {
        VP      exinf;    /* 拡張情報 */
        BOOL_ID wtsk;     /* 呼出待ちタスクの有無 */
        BOOL_ID atsk;     /* 受付待ちタスクの有無 */
} T_RPOR;
```

```
----割り込み関係----
typedef struct t_dint {
        ATR     intatr;  /* 割り込みハンドラ属性 */
        FP      inthdr;  /* 割り込みアドレス */
} T_DINT;
----固定長メモリプール関係----
typedef struct t_cmpf {
        VP      exinf;   /* 拡張情報 */
        ATR     mpfatr;  /* メモリプール属性 */
        INT     mpfcnt;  /* メモリブロック数 */
        INT     blfsz;   /* メモリブロックサイズ */
} T_CMPF;
typedef struct t_rmpf {
        VP      exinf;   /* 拡張情報 */
        BOOL_ID wtsk;    /* 待ちタスクの有無 */
        INT     frbcnt;  /* メモリブロック数 */
        INT     blksz;   /* メモリブロックサイズ */
} T_RMPF;
----可変長メモリプール関係----
typedef struct t_cmpl {
        VP      exinf;   /* 拡張情報 */
        ATR     mplatr;  /* メモリプール属性 */
        INT     mplsz;   /* メモリプールサイズ */
        INT     maxblksz /* メモリブロック最大サイズ */
} T_CMPL;
typedef struct t_rmpl {
        VP      exinf;   /* 拡張情報 */
        BOOL_ID wtsk;    /* 待ちタスクの有無 */
        INT     frsz;    /* 空き領域の合計サイズ */
        INT     maxsz;   /* 空き領域の最大サイズ */
} T_RMPL;
/* 時間管理関係 */
typedef struct t_systime{
        H       utime;   /* 上位 16 ビット */
        UW      ltime;   /* 下位 32 ビット */
} SYSTIME, ALMTIME;
typedef struct t_dcyc {
    VP   exinf;          /* 拡張情報 */
    ATRcycatr;           /* 周期起動ハンドラ属性 */
    FP   cychdr;         /* 周期起動ハンドラアドレス */
    UINT   cycact;       /* 周期起動ハンドラ活性状態 */
    CYCTIME   cyctim;    /* 周期起動間隔 */
} T_DCYC;
cycact:
        TCY_OFF    H'0000    /* 周期起動ハンドラが起動されない */
        TCY_ON     H'0001    /* 周期起動ハンドラが起動される */
        TCY_INI    H'0002    /* 周期カウンタが初期化される */
```

```
/* システム管理関係 */
typedef struct t_ver {
        UH      maker;    /* メーカー */
        UH      id;       /* 形式番号 */
        UH      spver;    /* 仕様書バージョン */
        UH      prver;    /* 製品バージョン */
        UH      prno[4];  /* 製品管理情報 */
        UH      cpu;      /* CPU 情報 */
        UH      var;      /* バリエーション記述子 */
} T_VER;
typedef   struct t_rsys {
        INT     sysstat;  /* システム状態 */
        ID      runtskid; /* 実行タスクの ID 番号 */
        PRI     runtskpri;/* 実行タスクの優先度 */
        UINT    psw;      /* プロセッサステータスワード */
} T_RSYS;
typedef struct t_dexc {
        ATR     excatr;           /* 例外ハンドラ属性 */
        FP      exchdr;           /* 例外ハンドラアドレス */
        ID      tskid;            /* タスク ID */
        W       excstksz  ;       /* スタックサイズ */
} 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;
};
typedef   struct    t_reit {
        PSW     psw;
        FP      pc;
};
typedef   struct    t_exc {
        W       exckind;
        UW      exccd;
        ID      tskid;
        UW      exeenv;
};
```

```
/* 優先度付きメールボックス関係 */
typedef struct t_cvmbx {
        ATR     mbxatr;  /* 拡張メイルボックス属性 */
        PRI     maxpri;  /* メッセージの最大優先度 */
        VP      mprihd;  /* メッセージ優先度のキューヘッダ先頭番地 */
} T_CVMBX;
typedef struct t_vmbx {
        BOOL_ID  wtsk;     /* 待ちタスクの有無 */
        T_MSG    pk_msg;   /* 次に受信されるメッセージ */
} T_RVMBX;
```

索引

M3T-MR32R V.3.50 リファレンスマニュアル

Rev. 1.00
03.06.16
RJJ10J0117-0100Z

M3T-MR32R V.3.50
リファレンスマニュアル

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668

RJJ10J0117-0100Z