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-MR308/4 V.4.00

リファレンスマニュアル

M16C/70,80,M32C/80 シリーズ用リアルタイムOS

ルネサス エレクトロニクス
www.renesas.com

Rev.2.00　2005.11

- Microsoft、MS-DOS、WindowsおよびWindows NTは、米国Microsoft Corporationの米国およびその他の国における登録商標です。
- Sun 、Java およびすべてのJava関連の商標およびロゴは、米国およびその他の国における米国Sun Microsystems, Inc.の商標または登録商標です。
- UNIXは、X/Open Company Limitedが独占的にライセンスしている米国ならびに他の国における登録商標です。
- IBMおよびATは、米国International Business Machines Corporationの登録商標です。
- Intel, Pentiumは、米国Intel Corporationの登録商標です。
- AdobeおよびAcrobatは、Adobe Systems Incorporated（アドビシステムズ社）の登録商標です。
- NetscapeおよびNetscape Navigatorは、米国およびその他の諸国のNetscape Communications Corporation社の登録商標です。
- TRONは、"The Real-Time Operating System Nucleus" の略称です。
- ITRONは、"Industrial TRON" の略称です。
- μITRONは、"Micro Industrial TRON" の略称です。μITRON仕様の著作権は(社)トロン協会に属しています。
- TRON、ITRON、およびμITRONは、コンピュータの仕様に対する名称であり、特定の商品ないし商品群を指すものではありません。
- その他すべてのブランド名および製品名は個々の所有者の登録商標もしくは商標です。

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
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-MR308/4(以下 MR308 と略す)は M16C/70,80,M32C/80 シリーズ用のリアルタイム・オペレーティングシステム[1]です。MR308 はμITRON 4.0 仕様[2]に準拠しています。

本マニュアルは MR308 のサービスコールの API について説明します。

MR308 を使うために必要なこと

MR308 を使用したプログラムを作成するには弊社下記製品を別途御購入して頂く必要があります。

- M16C/70,80,M32C/80 シリーズ用コンパイラパッケージ M3T-NC308WA(以下 NC308 と略す)

ドキュメント一覧

MR308 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)
  MR308 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)
  MR308 のサービス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。MR308 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] ・μITRON4.0 仕様は、(社)トロン協会が策定したオープンなリアルタイムカーネル仕様です。
・μITRON4.0 仕様の仕様書は、(社)トロン協会ホームページ（http://www.assoc.tron.org/）から入手が可能です。
・μITRON 仕様の著作権は(社)トロン協会に属しています。

目次

第1章 サービスコールインタフェース

1.1. タスク管理機能

表 1 タスク管理機能の仕様にタスク管理機能の仕様を示します。項番4タスク属性の記述言語は、GUI コンフィギュレータでの指定内容です。コンフィギュレーションファイルには出力されず、カーネルも関知しません。タスクスタックは、コンフィギュレーション時に、各タスク毎にセクション名を指定することが出来ます。

項番	項目	内容
1	タスク ID	1-255
2	タスク優先度	1-255
3	タスク起動要求キューイング数の最大値	255 回
4	タスク属性	TA_HLNG : 高級言語記述 TA_ASM : アセンブリ言語記述 TA_ACT:起動属性
5	タスクスタック	セクション指定可能

表 1 タスク管理機能の仕様

act_tsk　　タスクの起動
iact_tsk　　タスクの起動(ハンドラ専用)

■ C 言語 API

```
ER ercd = act_tsk( ID tskid );
ER ercd = iact_tsk( ID tskid );
```

● パラメータ

ID	tskid	対象タスク ID 番号

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ アセンブリ言語 API

```
.include mr308.inc
act_tsk  TSKID
iact_tsk TSKID
```

● パラメータ

TSKID	対象タスク ID 番号

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象タスク ID 番号

■ エラーコード

E_QOVR	キューイングオーバーフロー

■ 機能説明

tskid で示されたタスクを起動します。起動したタスクは休止(DORMANT)状態から実行可能(READY)状態もしくは実行(RUNNING)状態へ移行します。
タスク起動時に行われる処理は、以下の通りです。

1 タスクの現在優先度を初期化する。
2 起床要求キューイング数をクリアする。
3 強制待ち要求ネスト数をクリアする。

tskid=TSK_SELF(0)の指定により、自タスクの指定になります。タスクには、タスク生成時に指定したタスクの拡張情報がパラメータとして渡ります。非タスクコンテキストにおいて、tskid に TSK_SELF は指定した場合の動作は保証されません。
対象タスクが休止状態でない場合には、本サービスコールによるタスクの起動要求は、キューイングされます。すなわち、起動要求カウントに1加算されます。起動要求カウントの最大値は、255 です。起動要求カウントの

最大値を越える場合は、エラーコード E_QOVR を返します。
tskid に TSK_SELF が指定された場合は、自タスクを対象タスクとします。
本サービスコールは、タスクコンテキストからは、act_tsk、非タスクコンテキストからは、iact_tsk を使用してください。

■ 記述例

《 c 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task1( VP_INT stacd )
{
   ER ercd;
        :
    ercd = act_tsk( ID_task2 );
        :
}
void task2( VP_INT stacd )
{
        :
    ext_tsk();
}
```

《 アセンブリ言語の使用例 》

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
                :
    pushm               A0
    act_tsk             #ID_TASK3
                :
```

can_act　　起動要求カウントのキャンセル
ican_act　　起動要求カウントのキャンセル（ハンドラ専用）

■ C 言語 API

```
ER_UINT actcnt = can_act( ID tskid );
ER_UINT actcnt = ican_act( ID tskid );
```

● パラメータ

ID	tskid	対象タスク ID 番号

● リターンパラメータ

ER_UINT	actcnt > 0	キャンセルされた起動要求カウント
	actcnt <0	エラーコード

■ アセンブリ言語 API

```
.include mr308.inc
can_act  TSKID
ican_act TSKID
```

● パラメータ

TSKID	対象タスク ID 番号

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	キャンセルされた起動要求カウントまたは、エラーコード
A0	対象タスク ID 番号

■ エラーコード

なし

■ 機能説明

tskid で示されたタスクにキューイングされていた起動要求回数を求め、その結果をリターンパラメータとして返し、同時にその起動要求を全て無効にします。
tskid=TSK_SELF(0)の指定により、自タスクの指定になります。非タスクコンテキストにおいて、tskid に TSK_SELF は指定した場合の動作は保証されません。
休止状態のタスクを対象として呼び出すこともできます。その場合のリターンパラメータは 0 となります。
本サービスコールは、タスクコンテキストからは、can_act、非タスクコンテキストからは、ican_act を使用してください。

■ 記述例

《 C 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task1()
 {
    ER_UINT actcnt;
        :
    actcnt = can_act( ID_task2 );
        :
 }
 void task2()
 {
        :
    ext_tsk();
 }
```

《 アセンブリ言語の使用例 》

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
                :
    PUSHM       A0
    can_act     #ID_TASK2
                :
```

sta_tsk　　タスクの起動(起動コード指定)
ista_tsk　　タスクの起動（起動コード指定、ハンドラ専用）

■ C 言語 API

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

● パラメータ

ID	tskid	対象タスク ID 番号
VP_INT	stacd	タスク起動コード

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ アセンブリ言語 API

```
.include mr308.inc
sta_tsk  TSKID,STACD
ista_tsk TSKID,STACD
```

● パラメータ

TSKID	対象タスク ID 番号
STATCD	タスク起動コード

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	タスク起動コード(下位 16bit)
R3	タスク起動コード(上位 16bit)
A0	対象タスク ID 番号

■ エラーコード

E_OBJ	オブジェクト状態が不正(tskid のタスクが休止状態ではない)

■ 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(RUNNING)状態へ移行します。本サービスコールは、起動要求をキューイングしません。したがって、対象タスクが休止(DORMANT)状態にない場合に発せられた要求に対しては、サービスコール発行タスクにエラーE_OBJ を返します。本サービスコールは、指定したタスクが休止(DORMANT)状態であるときのみ有効です。起動コード stacd は、32 ビットです。stacd は起動タスクにパラメータとして渡されます。

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 。

1 タスクの現在優先度を初期化する。
2 起床要求キューイング数をクリアする。
3 強制待ち要求ネスト数をクリアする。

本サービスコールは、タスクコンテキストからは、sta_tsk、非タスクコンテキストからは、ista_tsk を使用してください。

■ 記述例

《 c 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
    ER ercd;
    VP_INT stacd = 0;
    ercd = sta_tsk( ID_task2, stacd );
           :
 }
 void task2(VP_INT msg)
 {
    if(msg == 0)
            :
 }
```

《 アセンブリ言語の使用例 》

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
                :
    PUSHM       A0,R1,R3
    sta_tsk     #ID_TASK4,#012345678H
                :
```

ext_tsk　　自タスクの終了

■ C 言語 API

```
ER ercd = ext_tsk();
```

● パラメータ

なし

● リターンパラメータ

本サービスコールからリターンしない

■ アセンブリ言語 API

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

● パラメータ

なし

● サービスコール発行後のレジスタ内容

本サービスコールからリターンしない

■エラーコード

本サービスコールからリターンしない

■ 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(RUNNING)状態から休止(DORMANT)状態へ移行します。ただし、自タスクに対する起動要求カウントが１の場合は、起動要求カウントを１減じ、再度 act_tsk,iact_tsk の処理と同様の処理を行い、タスクは、休止(DORMANT)状態から実行可能状態(READY)にします。タスクを起動するときにパラメータとして、タスク拡張情報を渡します。

C 言語で記述した場合、本サービスコールは、タスクからのリターンで自動的に発行されるようになっています。

本サービス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。

本サービスコールはタスクコンテキストでのみ使用可能です。また、本サービスコールは、CPU ロック状態、ディスパッチ禁止状態であっても使用可能です。この場合、CPU ロック状態、ディスパッチ禁止状態は解除されます。しかし、非タスクコンテキストでは使用できません。

■ 記述例

《 C 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task(void)
 {
      :
   ext_tsk();
 }
```

《 アセンブリ言語の使用例 》

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

ter_tsk　　タスクの強制終了

■ C 言語 API

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

● パラメータ

ID	tskid	対象タスク ID 番号

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ アセンブリ言語 API

```
.include mr308.inc
ter_tsk  TSKID
```

● パラメータ

TSKID	対象タスク ID 番号

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象タスク ID 番号

■エラーコード

E_OBJ	オブジェクト状態が不正(tskid のタスクが休止状態)
E_ILUSE	サービスコール不正使用(tskid に自タスクを指定)

■ 機能説明

tskid で示されたタスクを、強制的に終了させます。対象タスクの起動要求カウントが1以上の場合、起動要求カウントを1減じ、再度 act_tsk,iact_tsk の処理と同様の処理を行い、タスクは、休止(DORMANT)状態から実行可能状態(READY)にします。タスクを起動するときにパラメータとして、タスク拡張情報を渡します。
このサービスコールで自タスクを指定した場合(TSK_SELF を指定した場合も)、自タスクは終了することはなく、エラーコード E_ILUSE を返します。自タスクを終了する場合は ext_tsk サービスコールを使用してください。
自タスクを指定したタスクが待ち状態に入り、何らかの待ち行列 につながれていた場合には、このサービス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、サービスコールの戻り値としてエラーE_OBJ を返します。
本サービスコールはタスクコンテキストでのみ使用可能です。非タスクコンテキストでは使用できません。

■ 記述例

《 C 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
         :
     ter_tsk( ID_main );
         :
 }
```

《 アセンブリ言語の使用例 》

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
                :
    PUSHM       A0
    ter_tsk     #ID_TASK3
                :
```

chg_pri　　タスク優先度の変更
ichg_pri　　タスク優先度の変更（ハンドラ専用）

■ C 言語 API

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

● パラメータ

ID	tskid	対象タスク ID 番号
PRI	tskpri	対象タスク優先度

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ アセンブリ言語 API

```
.include mr308.inc
chg_pri  TSKID,TSKPRI
ichg_pri  TSKID,TSKPRI
```

● パラメータ

TSKID	対象タスク ID 番号
TSKPRI	対象タスク優先度

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R3	タスク優先度
A0	対象タスク ID 番号

■エラーコード

E_OBJ	オブジェクト状態が不正(tskid のタスクが休止状態)

■ 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、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、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サービス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 。

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ギュ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、コンフィギュ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。

TSK_SELF が指定された場合は自タスクの優先度を変更します。非タスクコンテキストにおいて、tskid に TSK_SELF は指定した場合の動作は保証されません。TPRI_INI が指定された場合、タスク生成時に指定したタスクの起動時優先度に変更します。変更したタスク優先度は、タスクの終了もしくは、本サービスコールが再度実行されるまで有効です。

tskid で示されたタスクが休止(DORMANT)状態にある場合は、サービスコールの戻り値としてエラーE_OBJ を返します。MR308 では、ミューテックス機能をサポートしないため、エラーコード E_ILUSE を返すことはありません。

本サービスコールは、タスクコンテキストからは、chg_pri、非タスクコンテキストからは、ichg_pri を使用してください。

■ 記述例

《 c 言語の使用例 》

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

《 アセンブリ言語の使用例 》

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
                :
    pushm       A0,R3
    chg_pri     #ID_TASK3,#1
                :
```

get_pri　　タスク優先度の参照
iget_pri　　タスク優先度の参照（ハンドラ専用）

■ C 言語 API

```
ER ercd = get_pri( ID tskid, PRI *p_tskpri );
ER ercd = iget_pri( ID tskid, PRI *p_tskpri );
```

● パラメータ

ID	tskid	対象タスク ID 番号
PRI	*p_tskpri	タスク優先度を返す領域へのポインタ

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ アセンブリ言語 API

```
.include mr308.inc
get_pri  TSKID
iget_pri  TSKID
```

● パラメータ

TSKID	対象タスク ID 番号

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	獲得したタスク優先度

■エラーコード

E_OBJ	オブジェクト状態が不正(tskid のタスクが休止状態)

■ 機能説明

tskid で示されたタスクの優先度を、p_tskpri で示される領域に返します。TSK_SELF が指定された場合は自タスクの優先度を参照します。非タスクコンテキストにおいて、tskid に TSK_SELF は指定した場合の動作は保証されません。

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

本サービスコールは、タスクコンテキストからは、get_pri、非タスクコンテキストからは、iget_pri を使用してください。

■ 使用例

《 C言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
   PRI p_tskpri;
   ER ercd;
        :
   ercd = get_pri( ID_task2, &p_tskpri );
        :
 }
```

《 アセンブリ言語の使用例 》

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
                :
    PUSHM       A0
    get_pri     #ID_TASK2
                :
```

ref_tsk　　タスクの状態参照
iref_tsk　　タスクの状態参照（ハンドラ専用）

■ C 言語 API

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

● パラメータ

ID	tskid	対象タスク ID 番号
T_RTSK	*pk_rtsk	タスク状態を返すパケットへのポインタ

● リターンパラメータ

ER	ercd	正常終了(E_OK)

pk_rtsk の内容

```
typedef   struct t_rtsk{
    STAT    tskstat     +0   2    タスク状態
    PRI     tskpri      +2   2    タスクの現在優先度
    PRI     tskbpri     +4   2    タスクのベース優先度
    STAT    tskWAITING  +6   2    待ち要因
    ID      wobjid      +8   2    待ちオブジェクト ID
    TMO     lefttmo     +10  4    タイムアウトするまでの時間
    UINT    actcnt      +14  2    起動要求キューイング数
    UINT    wupcnt      +16  2    起床要求キューイング数
    UINT    suscnt      +18  2    強制待ち要求ネスト数
} T_RTSK;
```

■ アセンブリ言語 API

```
.include mr308.inc
ref_tsk  TSKID, PK_RTSK
iref_tsk  TSKID, PK_RTSK
```

● パラメータ

TSKID	対象タスク ID 番号
PK_RTSK	タスク状態を返すパケットへのポインタ

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象タスク ID 番号
A1	タスク状態を返すパケットへのポインタ

■エラーコード

なし

■ 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 pk_rtsk の指す領域にリターンパラメータとして返します。tskid として TSK_SELF が指定された場合は、自タスクの状態を参照します。非タスクコンテキストにおいて、tskid に TSK_SELF は指定した場合の動作は保証されません。

◆ tskstat(タスク状態)

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_RUNNING(0x0001)　実行(RUNNING)状態
- TTS_RDY(0x0002) 実行可能(READY)状態
- TTS_WAI(0x0004) 待ち(WAITING)状態
- TTS_SUS(0x0008) 強制待ち(SUSPENDED)状態

- TTS_WAS(0x000C) 二重待ち(WAITING-SUSPENDED)状態
- TTS_DMT(0x0010) 休止(DORMANT)状態

◆ **tskpri(タスクの現在優先度)**

tskpriには、指定したタスクの現在優先度を返します。タスクが休止状態の場合は、tskpriは、不定となります。

◆ **tskbpri(タスクのベース優先度)**

tskpri には、指定したタスクのベース優先度を返します。MR308 は、ミューテックス機能をサポートしていないため、tskpriとtskbpri は、同じ値となります。また、タスクが休止状態の場合は、tskpri は、不定となります。

◆ **tskWAITING(タスクの待ち要因)**

対象タスクが待ち状態であるならば次の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。タスク状態が、待ち状態(TTS_WAI、または TTS_WAS)以外の場合は、tskWAITING は不定となります。

- TTW_SLP (0x0001) slp_tsk、tslp_tsk による待ち
- TTW_DLY (0x0002) dly_tsk による待ち
- TTW_SEM (0x0004) wai_sem、twai_sem による待ち
- TTW_FLG (0x0008) wai_flg、twai_flg による待ち
- TTW_SDTQ(0x0010) snd_dtq、tsnd_dtq による待ち
- TTW_RDTQ(0x0020) rcv_dtq,trcv_dtq による待ち
- TTW_MBX (0x0040) rcv_mbx、trcv_mbx による待ち
- TTW_MPF (0x2000) get_mpf,tget_mpf による待ち
- TTW_VSDTQ (0x4000) vsnd_dtq,vtsnd_dtq による待ち[3]
- TTW_VRDTQ(0x8000) vrcv_dtq,vtrcv_dtq による待ち

◆ **wobjid(待ちオブジェクト ID)**

対象タスクが待ち状態(TTS_WAI または、TTS_WAS)であるならば、待ち対象オブジェクト ID を返します。それ以外の場合は、wobjid は、不定となります。

◆ **lefttmo(タイムアウトまでの時間)**

対象タスクが dly_tsk 以外による待ち状態(TTS_WAI,または TTS_WAS)の場合、タイムアウトするまでの時間を返します。永久待ちの場合は、TMO_FEVR を返します。それ以外の状態の場合は、lefttmo は不定となります。

◆ **actcnt(起動要求カウント)**

現在の起動要求キューイング数を返します。

◆ **wupcnt(起床要求カウント)**

現在の起床要求キューイング数を返します。タスクが休止状態の場合は、wupcnt は、不定となります。

◆ **suscnt(強制待ち要求カウント)**

現在の強制待ち要求ネスト数を返します。タスクが休止状態の場合は、suscnt は、不定となります。

本サービスコールは、タスクコンテキストからは、ref_tsk、非タスクコンテキストからは、iref_tsk を使用してください。

■ 使用例

[3] TTW_VSDTQ,TTW_VRDTQ は μITRON 仕様 V.4.0 の仕様外の待ち要因です。

《 C 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
      T_RTSK rtsk;
      ER ercd;
           :
      ercd = ref_tsk( ID_main, &rtsk );
           :
 }
```

《 アセンブリ言語の使用例 》

```
_refdata:       .blkb   20
    .include mr308.inc
    .GLB        task
task:
           :
    PUSHM       A0,A1
    ref_tsk     #TSK_SELF,#_refdata
           :
```

ref_tst　　タスクの状態参照(簡易版)
iref_tst　　タスクの状態参照(簡易版、ハンドラ専用)

■ C 言語 API

```
ER ercd = ref_tst( ID tskid, T_RTST *pk_rtst );
ER ercd = iref_tst( ID tskid, T_RTST *pk_rtst );
```

● パラメータ

ID	tskid	対象タスク ID 番号
T_RTST	*pk_rtst	タスク状態を返すパケットへのポインタ

● リターンパラメータ

ER	ercd	正常終了(E_OK)

pk_rtsk の内容

```
typedef   struct t_rtst{
    STAT    tskstat     +0    2     タスク状態
    STAT    tskWAITING  +2    2     待ち要因
} T_RTST;
```

■ アセンブリ言語 API

```
.include mr308.inc
ref_tst  TSKID, PK_RTST
iref_tst  TSKID, PK_RTST
```

● パラメータ

TSKID	対象タスク ID 番号
PK_RTST	タスク状態を返すパケットへのポインタ

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象タスク ID 番号
A1	タスク状態を返すパケットへのポインタ

■エラーコード

なし

■ 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 pk_rtst が指す領域にリターン値として返します。tskid として TSK_SELF が指定された場合は、自タスクの状態を参照します。非タスクコンテキストにおいて、tskid に TSK_SELF は指定した場合の動作は保証されません。

◆ tskstat(タスク状態)

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_RUNNING(0x0001)　　実行(RUNNING)状態
- TTS_RDY(0x0002) 実行可能(READY)状態
- TTS_WAI(0x0004) 待ち(WAITING)状態
- TTS_SUS(0x0008) 強制待ち(SUSPENDED)状態
- TTS_WAS(0x000C) 二重待ち(WAITING-SUSPENDED)状態
- TTS_DMT(0x0010) 休止(DORMANT)状態

◆ tskWAITING(タスクの待ち要因)

対象タスクが待ち状態であるならば次の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。タスク状態が、待ち状態(TTS_WAI、または TTS_WAS)以外の場合は、tskWAITING は不定となります。

- TTW_SLP (0x0001)　　slp_tsk、tslp_tsk による待ち
- TTW_DLY (0x0002)　　dly_tsk による待ち
- TTW_SEM (0x0004)　　wai_sem、twai_sem による待ち
- TTW_FLG (0x0008)　　wai_flg、twai_flg による待ち
- TTW_SDTQ(0x0010)　　snd_dtq、tsnd_dtq による待ち
- TTW_RDTQ(0x0020)　　rcv_dtq,trcv_dtq による待ち
- TTW_MBX (0x0040)　　rcv_mbx、trcv_mbx による待ち
- TTW_MPF (0x2000)　　get_mpf,tget_mpf による待ち
- TTW_VSDTQ (0x4000)　　vsnd_dtq,vtsnd_dtq による待ち[4]
- TTW_VRDTQ(0x8000)　　vrcv_dtq,vtrcv_dtq による待ち

本サービスコールは、タスクコンテキストからは、ref_tst、非タスクコンテキストからは、iref_tst を使用してください。

[4] TTW_VSDTQ,TTW_VRDTQ は μITRON 仕様 V.4.0 の仕様外の待ち要因です。

■ 使用例

《 C 言語の使用例 》

```
    #include <itron.h>
    #include <kernel.h>
    #include "kernel_id.h"
    void task()
    {
        T_RTST rtst;
        ER ercd;
                :
        ercd = ref_tst( ID_main, &rtst );
                :
    }
```

《 アセンブリ言語の使用例 》

```
_refdata:       .blkb   4
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0,A1
    ref_tst     #ID_TASK2,#_refdata
          :
```

1.2. タスク付属同期機能

表 2 タスク付属同期機能の仕様にタスク付属同期機能の仕様を示します。

項番	項目	内容
1	タスク起床要求カウントの最大値	255 回
2	タスク強制待ち要求ネスト数の最大値	1 回

表 2 タスク付属同期機能の仕様

slp_tsk 起床待ち
tslp_tsk 起床待ち(タイムアウト)

■ C 言語 API

```
ER ercd = slp_tsk();
ER ercd = tslp_tsk( TMO tmout );
```

● パラメータ

- slp_tsk の場合
  なし
- tslp_tsk の場合

TMO	tmout	タイムアウト値

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ アセンブリ言語 API

```
.include mr308.inc
slp_tsk
tslp_tsk TMO
```

● パラメータ

TMO	タイムアウト値

● サービスコール発行後のレジスタ内容

tslp_tsk の場合

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	タイムアウト値(下位 16bit)
R3	タイムアウト値(上位 16bit)

slp_tsk の場合

レジスタ名	サービスコール発行後の内容
R0	エラーコード

■エラーコード

E_TMOUT	タイムアウト
E_RLWAI	強制待ち解除

■ 機能説明

自タスクを実行(RUNNING)状態から起床待ち状態へ移行します。本サービス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サービスコール 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サービスコール 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 経過後、最初のタイムティックが発生した場合(tslp_tsk の場合)**

この時のエラーコードは、E_TMOUT が返ります。

本サービスコールにより待ち(WAITING)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(WAITING-SUSPENDED)状態になります。この場合はタスク起床のサービス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。

tslp_tsk では、引数に tmout を指定し一定時間自タスクを起床待ち状態に移行させることが出来ます。tmout の単位は、ｍｓです。すなわち、tslp_tsk(10);と記述すれば、10ms 間、自タスクが実行(RUNNING)状態から待ち(WAITING)状態へ移行します。tmout=TMO_FEVR(-1)にした場合は、永久待ちの指定となり、slp_tsk サービスコールと同じ動作を行います。

tmout に指定可能な値は、(0x7FFFFFFF-タイムティック)以内でなければいけません。これより大きな値を指定した場合の動作は、保証されません。

本サービスコールはタスクコンテキストからのみ発行してください。非タスクコンテキストから発行することはできません。

■ 使用例

《 C 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
                :
     if( slp_tsk() != E_OK )
         error("Forced wakeup¥n");
                :
     if( tslp_tsk( 10 ) == E_TMOUT )
         error("time out¥n");
                :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    slp_tsk
          :
    PUSHM       R1,R3
    tslp_tsk    #TMO_FEVR
          :
    PUSHM       R1,R3
    tslp_tsk    #100
          :
```

wup_tsk　　タスクの起床
iwup_tsk　　タスクの起床（ハンドラ専用）

■ C 言語 API

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

● パラメータ

ID	tskid	対象タスク ID 番号

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ アセンブリ言語 API

```
.include mr308.inc
wup_tsk  TSKID
iwup_tsk  TSKID
```

● パラメータ

TSKID	対象タスク ID 番号

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象タスク ID 番号

■エラーコード

E_OBJ	オブジェクト状態が不正(tskid のタスクが休止状態)
E_QOVR	キューイングのオーバーフロー

■ 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(WAITING)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(RUNNING)状態に移行します。また、tskid で指定したタスクが二重待ち(WAITING-SUSPENDED)状態である時は、待ちのみを解除して強制待ち(SUSPENDED)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、サービスコール発行タスクにエラーE_OBJを返します。tskidにTSK_SELFが指定された場合は、自タスク指定となります。非タスクコンテキストにおいて、tskid に TSK_SELF は指定した場合の動作は保証されません。

slp_tsk あるいは tslp_tsk サービスコール実行による待ち(WAITING)状態もしくは二重待ち(WAITING-SUSPENDED)状態にないタスクに対して本サービス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。起床要求カウントの最大値は 255 です。起床要求カウントが 255 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 255 のままで本サービスコールの発行タスクには、エラーコード E_QOVR を返します。

本サービスコールは、タスクコンテキストからは、wup_tsk、非タスクコンテキストからは、iwup_tsk を使用してください。

■ 使用例

《 C 言語の使用例 》

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

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    wup_tsk     #ID_TASK1
          :
```

can_wup　　起床要求のキャンセル
ican_wup　　起床要求のキャンセル（ハンドラ専用）

■ C 言語 API

```
ER_UINT wupcnt = can_wup( ID tskid );
ER_UINT wupcnt = icam_wup( ID tskid );
```

● パラメータ

ID	tskid	対象タスク ID 番号

● リターンパラメータ

ER_UINT	wupcnt > 0	キャンセルされた起床要求カウント
	wupcnt <0	エラーコード

■ アセンブリ言語 API

```
.include mr308.inc
can_wup  TSKID
ican_wup  TSKID
```

● パラメータ

TSKID	対象タスク ID 番号

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード、キャンセルされた起床要求カウント
A0	対象タスク ID 番号

■エラーコード

E_OBJ	オブジェクト状態が不正(tskid のタスクが休止状態)

■ 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、本サービスコール発行以前に wup_tsk、iwup_tsk サービスコールにより起床しようとした時に対象タスクが待ち(WAITING)状態もしくは二重待ち(WAITING-SUSPENDED)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、本サービス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、サービスコール発行タスクにエラーE_OBJ を返します。tskid に TSK_SELF が指定された場合は、自タスク指定となります。

本サービスコールは、タスクコンテキストからは、can_wup、非タスクコンテキストからは、ican_wup を使用してください。

■ 使用例

《 C言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
     ER_UINT wupcnt;
             :
     wupcnt = can_wup(ID_main);
     if( wup_cnt  > 0 )
         printf("wupcnt = %d¥n",wupcnt);
             :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    can_wup     #ID_TASK3
          :
```

rel_wai　　待ち状態の強制解除
irel_wai　　待ち状態の強制解除（ハンドラ専用）

■ C 言語 API

```
ER ercd = rel_wai( ID tskid );
ER ercd = irel_wai( ID tskid );
```

● パラメータ

ID	tskid	対象タスク ID 番号

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ アセンブリ言語 API

```
.include mr308.inc
rel_WAITINGSKID
irel_WAITINGSKID
```

● パラメータ

TSKID	対象タスク ID 番号

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象タスク ID 番号

■エラーコード

E_OBJ	オブジェクト状態が不正(tskid のタスクが待ち状態でない)

■ 機能説明

tskid で示されたタスクの待ち状態(強制待ち(SUSPENDED)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(RUNNING)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 につながれていた場合には、本サービスコールの実行によってその待ち行列から削除されます。

二重待ち状態(WAITING-SUSPENDED)のタスクに対して、本サービスコールを発行した場合、対象タスクの待ち状態は解除され、強制待ち状態(SUSPENDED)に移行します。[5]

対象タスクが待ち状態にない場合は、サービスコール発行タスクにエラーE_OBJを返します。また、本サービスコールは、自タスクを指定できません。

本サービスコールは、タスクコンテキストからは、rel_wai、非タスクコンテキストからは、irel_wai を使用してください。

■ 使用例

《 c言語の使用例 》

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

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    rel_wai     #ID_TASK2
          :
```

[5] 強制待ち状態は、本サービスコールにより待ちは解除されません。強制待ち状態は、rsm_tsk,irsm_tsk,frsm_tsk,ifrsm_tsk サービスコールによって待ちが解除されます。

sus_tsk　　強制待ち状態への移行
isus_tsk　　強制待ち状態への移行（ハンドラ専用）

■ C 言語 API

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

● パラメータ

ID	tskid	対象タスク ID 番号

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ アセンブリ言語 API

```
.include mr308.inc
sus_tsk  TSKID
isus_tsk  TSKID
```

● パラメータ

TSKID	対象タスク ID 番号

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象タスク ID 番号

■エラーコード

E_OBJ	オブジェクト状態が不正(tskid のタスクが休止状態)
E_QOVR	キューイングのオーバーフロー

■ 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(SUSPENDED)状態へ移行します。強制待ち状態は、rsm_tsk,irsm_tsk,frsm_tsk,ifrsm_tsk サービス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、サービスコールの戻り値としてエラーE_OBJ を返します。

本サービスコールによる強制待ち要求のネスト数の最大値は1です。強制待ち状態のタスクに対して本サービスコールを発行した場合は、エラーE_QOVR を返します。

本サービスコールで自タスクを指定することはできません。

本サービスコールは、タスクコンテキストからは、sus_tsk、非タスクコンテキストからは、isus_tsk を使用してください。

■ 使用例

《 c言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
                :
     if( sus_tsk( ID_main ) != E_OK )
         printf("Can't SUSPENDED task main()¥n");
                :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    sus_tsk     #ID_TASK2
          :
```

rsm_tsk	**強制待ち状態の解除**
irsm_tsk	**強制待ち状態の解除（ハンドラ専用）**
frsm_tsk	**強制待ち状態の強制解除**
ifrsm_tsk	**強制待ち状態の強制解除（ハンドラ専用）**

■ C 言語 API

```
ER ercd = rsm_tsk( ID tskid );
ER ercd = irsm_tsk( ID tskid );
ER ercd = frsm_tsk( ID tskid );
ER ercd = ifrsm_tsk( ID tskid );
```

● パラメータ

ID	tskid	対象タスク ID 番号

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ アセンブリ言語 API

```
.include mr308.inc
rsm_tsk  TSKID
irsm_tsk  TSKID
frsm_tsk TSKID
ifrsm_tsk TSKID
```

● パラメータ

TSKID	対象タスク ID 番号

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象タスク ID 番号

■エラーコード

E_OBJ	オブジェクト状態が不正(tskid のタスクが強制待ち状態でない)

■ 機能説明

tskid で示されたタスクが sus_tsk サービス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。対象タスクの強制待ち状態の場合は、強制待ち状態を解除します。

対象タスクが強制待ち(SUSPENDED)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サービスコール発行タスクにエラーE_OBJ を返します。

rsm_tsk,irsm_tsk,frsm_tsk,ifrsm_tsk サービスコールいずれにおいても、強制待ち要求の最大ネスト数が1であるため、同じ動作をします。

本サービスコールは、タスクコンテキストからは、rsm_tsk・frsm_tsk、非タスクコンテキストからは、irsm_tsk・ifrsm_tsk を使用してください。

■ 使用例

《 C 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task1()
 {
                :
     if( rsm_tsk( ID_main ) != E_OK )
         printf("Can't resume main()¥n");
                :
                :
     if(frsm_tsk( ID_task2 ) != E_OK )
         printf("Can't forced resume task2()¥n");
                :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    rsm_tsk     #ID_TASK2
          :
    PUSHM       A0
    frsm_tsk    #ID_TASK1
          :
```

dly_tsk　　タスクの遅延

■ C 言語 API

```
ER ercd = dly_tsk(RELTIM dlytim);
```

● パラメータ

RELTIM	dlytim	遅延時間

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ アセンブリ言語 API

```
.include mr308.inc
dly_tsk  RELTIM
```

● パラメータ

RELTIM	遅延時間

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	遅延時間(下位 16bit)
R3	遅延時間(上位 16bit)

■エラーコード

E_RLWAI	強制待ち解除

■ 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(RUNNING)状態から待ち状態へ移行します。具体的には、dlytim で指定した時間経過後の最初のタイムティックで待ち状態が解除されます。そのため、dlytimに0が指定された場合は、いったん待ち状態に移行し、最初のタイムティックで待ち状態が解除されます。

本サービス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サービス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 経過後、最初のタイムティックが発生した場合場合

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 サービス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 サービスコールが発行されても、待ち解除とはなりません。

dlytim の単位は、ｍｓです。すなわち、dly_tsk(50);と記述すれば、50ms 間、自タスクが実行(RUNNING)状態から遅延待ち状態へ移行します。

dlytim に指定可能な値は、(0x7FFFFFFF-タイムティック)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。

本サービスコールはタスクコンテキストからのみ発行してください。非タスクコンテキスト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 <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
              :
     if( dly_tsk() != E_OK )
         error("Forced wakeup¥n");
              :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       R1,R3
    dly_tsk     #500
          :
```

1.3. 同期・通信機能(セマフォ)

表 3 セマフォ機能の仕様にセマフォ機能の仕様を示します。

項番	項目	内容
1	セマフォ ID	1-255
2	最大資源数	1-65535
3	セマフォ属性	TA_FIFO :タスクキューFIFO 順 TA_TPRI :タスクキュー優先度順

表 3 セマフォ機能の仕様

sig_sem　　セマフォ資源の返却
isig_sem　　セマフォ資源の返却（ハンドラ専用）

■ C 言語 API

```
ER ercd = sig_sem( ID semid );
ER ercd = isig_sem( ID semid );
```

● パラメータ

ID	semid	対象セマフォ ID 番号

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ アセンブリ言語 API

```
.include mr308.inc
sig_sem SEMID
isig_sem SEMID
```

● パラメータ

SEMID	対象セマフォ ID 番号

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象セマフォ ID 番号

■ エラーコード

E_QOVR	キューイングオーバーフロー

■ 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 。セマフォの計数値がコンフィギュ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 サービスコール)をおこなうとセマフォの計数値はそのままで、サービスコール発行タスクにエラーE_QOVR を返します。
本サービスコールは、タスクコンテキストからは、sig_sem、非タスクコンテキストからは、isig_sem を使用してください。

■ 使用例

《 c 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
                :
     if( sig_sem( ID_sem ) == E_QOVR )
       error("Overflow¥n");
                :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    sig_sem     #ID_SEM2
          :
```

wai_sem	**セマフォ資源の獲得**
pol_sem	**セマフォ資源の獲得（ポーリング）**
ipol_sem	**セマフォ資源の獲得（ポーリング、ハンドラ専用）**
twai_sem	**セマフォ資源の獲得（タイムアウト）**

■ C 言語 API

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

● パラメータ

ID	semid	対象セマフォ ID 番号
TMO	tmout	タイムアウト値(twai_sem の場合)

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ アセンブリ言語 API

```
.include mr308.inc
wai_sem SEMID
pol_sem SEMID
ipol_sem SEMID
twai_sem SEMID,TMO
```

● パラメータ

SEMID	セマフォ ID 番号
TMO	タイムアウト値(twai_sem の場合)

● サービスコール発行後のレジスタ内容

wai_sem,pol_sem,ipol_sem の場合

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象セマフォ ID 番号

twai_sem の場合

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	タイムアウト値(下位 16bit)
R3	タイムアウト値(上位 16bit)
A0	対象セマフォ ID 番号

■ エラーコード

E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト

■ 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 だけ減じてサービス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、wai_sem,twai_sem サービスコール呼び出しタスクは、そのセマフォ待ち行列につなぎます。semid のセマフォ属性が TA_TFIFO の場合は、待ち行列に FIFO 順でタスクをつなぎます。TA_TPRI の場合は、待ち行列に優先度順でタスクをつなぎます。pol_sem,ipol_sem サービスコールでは、直ちにリターンし、エラーE_TMOUT を返します。
twai_sem サービスコールの場合は、tmout には、待ち時間を ms 単位で指定します。tmout に指定可能な値は、(0x7FFFFFFF-タイムティック)以内でなければいけません。これより大きな値を指定した場合の動作は、保証されません。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_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 サービスコールと同じ動作をします。
wai_sem,twai_sem サービス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。

- **tmout の時間が経過する前に、sig_sem、isig_sem サービスコールが発行され、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 経過し、最初のタイムティックが発生した場合**
  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 サービス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。

タスクコンテキストにおいては、wai_sem,twai_sem,pol_sem サービスコール、非タスクコンテキストにおいては、ipol_sem を使用してください。

■ 使用例

《 C言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
                :
   if( wai_sem( ID_sem ) != E_OK )
       printf("Forced wakeup¥n");
                :
   if( pol_sem( ID_sem ) != E_OK )
       printf("Timeout¥n");
                :
     if( twai_sem( ID_sem, 10 ) != E_OK )
         printf("Forced wakeup or Timeout"n");
                :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    pol_sem     #ID_SEM1
          :
    PUSHM       A0
    wai_sem     #ID_SEM2
          :
    PUSHM       A0,R1,R3
    twai_sem    #ID_SEM3,300
          :
```

ref_sem　　セマフォの状態参照
iref_sem　　セマフォの状態参照（ハンドラ専用）

■ C 言語 API

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

● パラメータ

ID	semid	対象セマフォ ID 番号
T_RSEM	*pk_rsem	セマフォ状態を返すパケットへのポインタ

● リターンパラメータ

ER	ercd	正常終了(E_OK)
T_RSEM	*pk_rsem	セマフォ状態を返すパケットへのポインタ

pk_rsem の内容

```
typedef   struct t_rsem{
    ID      wtskid    +0    2     待ちタスク ID
    UINT    semcnt    +2    2     現在のセマフォカウンタ値
} T_RSEM;
```

■ アセンブリ言語 API

```
.include mr308.inc
ref_sem SEMID, PK_RSEM
iref_sem SEMID, PK_RSEM
```

● パラメータ

SEMID	対象セマフォ ID 番号
PK_RSEM	セマフォ状態を返すパケットへのポインタ

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象セマフォ ID 番号
A1	セマフォ状態を返すパケットへのポインタ

■エラーコード

なし

■ 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。

- **wtskid**
  wtskid には待ち行列の先頭タスク(次に待ち行列から削除されるタスク)の ID 番号を返します。待ちタスクの無い場合は TSK_NONE 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。

本サービスコールは、タスクコンテキストからは、ref_sem、非タスクコンテキストからは、iref_sem を使用してください。

■ 使用例

《 c 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
     T_RSEM rsem;
     ER ercd;
         :
     ercd = ref_sem( ID_sem1, &rsem );
         :
 }
```

《 アセンブリ言語の使用例 》

```
_ refsem:       .blkb   4
    .include mr308.inc
    .GLB        task
task:
         :
       PUSHM   A0,A1
       ref_sem #ID_SEM1,#_refsem
         :
```

1.4. 同期・通信機能(イベントフラグ)

表 4 イベントフラグ機能の仕様にイベントフラグ機能の仕様を示します。

項番	項目	
1	イベントフラグ ID	1-255
2	イベントフラグのビット数	16 ビット
3	イベントフラグ属性	TA_TFIFO :待ちタスクのキューイングは FIFO 順 TA_TPRI :待ちタスクのキューイングは優先度順 TA_WSGL : 複数タスクの待ちを許さない TA_WMUL : 複数タスクの待ちを許す TA_CLR : 待ちタスクを解除する時にビットパターンをクリアする

表 4 イベントフラグ機能の仕様

set_flg　　イベントフラグのセット
iset_flg　　イベントフラグのセット（ハンドラ専用）

■ C 言語 API

```
ER ercd = set_flg( ID flgid, FLGPTN setptn );
ER ercd = iset_flg( ID flgid, FLGPTN setptn );
```

● パラメータ

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

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

```
.include mr308.inc
set_flg FLGID,SETPTN
iset_flg FLGID,SETPTN
```

● パラメータ

FLGID	対象イベントフラグ ID 番号
SETPTN	セットするビットパターン

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R3	セットするビットパターン
A0	対象イベントフラグ ID 番号

■ エラーコード

なし

■ 機能説明

flgid で示される16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 サービスコールによって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(RUNNING)状態へ移行します 。

待ち解除条件は、待ち行列の先頭から順に評価されます。イベントフラグ属性として、TA_WMUL が指定されている場合、イベントフラグは、一回の set_flg,iset_flg サービスコール発行によって、複数タスクの待ち状態を同時に解除することができます。また、対象のイベントフラグ属性に TA_CLR 属性が指定されている場合は、イベントフラグのすべてのビットパターンをクリアし、サービスコールの処理を終了します。

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。

本サービスコールは、タスクコンテキストからは、set_flg、非タスクコンテキストからは、iset_flg を使用してください。

■ 使用例

《 c 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task(void)
 {
                :
   set_flg( ID_flg,(FLGPTN)0xff00 );
                :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0, R3
    set_flg     #ID_FLG3,#0ff00H
          :
```

clr_flg　　イベントフラグのクリア
iclr_flg　　イベントフラグのクリア（ハンドラ専用）

■ C 言語 API

```
ER ercd = clr_flg( ID flgid, FLGPTN clrptn );
ER ercd = iclr_flg( ID flgid, FLGPTN clrptn );
```

● パラメータ

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

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

```
.include mr308.inc
clr_flg FLGID,CLRPTN
iclr_flg FLGID,CLRPTN
```

● パラメータ

FLGID	対象イベントフラグ ID 番号
CLRPTN	クリアするビットパターン

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象イベントフラグ ID 番号
R3	クリアするビットパターン

■ エラーコード

なし

■ 機能説明

flgid で示される 16 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ビットパターンを、clrptn 値との論理積(AND)に更新します。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。

本サービスコールは、タスクコンテキストからは、clr_flg、非タスクコンテキストからは、iclr_flg を使用してください。

■ 使用例

《 C 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task(void)
 {
                :
   clr_flg( ID_flg,(FLGPTN) 0xf0f0);
                :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0, R3
    clr_flg     #ID_FLG1,#0f0f0H
          :
```

wai_flg	**イベントフラグ待ち**
pol_flg	**イベントフラグ待ち（ポーリング）**
ipol_flg	**イベントフラグ待ち（ポーリング、ハンドラ専用）**
twai_flg	**イベントフラグ待ち（タイムアウト）**

■ C 言語 API

```
ER ercd = wai_flg( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn );
ER ercd = pol_flg( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn );
ER ercd = ipol_flg( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn );
ER ercd = twai_flg( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn,
                    TMO tmout );
```

● パラメータ

ID	flgid	対象イベントフラグ ID 番号
FLGPTN	waiptn	待ちビットパターン
MODE	wfmode	待ちモード
FLGPTN	*p_flgptn	待ち解除時のビットパターンを返す領域へのポインタ
TMO	tmout	タイムアウト値(twai_flg の場合)

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
FLGPTN	*p_flgptn	待ち解除時のビットパターンを返す領域へのポインタ

■ アセンブリ言語 API

```
.include mr308.inc
wai_flg   FLGID, WAIPTN, WFMODE
pol_flg   FLGID, WAIPTN, WFMODE
ipol_flg  FLGID, WAIPTN, WFMODE
twai_flg  FLGID, WAIPTN, WFMODE, TMO
```

● パラメータ

FLGID	対象イベントフラグ ID 番号
WAIPTN	待ちビットパターン
WFMODE	待ちモード
TMO	タイムアウト値(twai_flg の場合)

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	待ちモード
R2	待ち解除時のビットパターン
R3	待ちビットパターン
A0	対象イベントフラグ ID 番号

■ エラーコード

E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト
E_ILUSE	サービスコール不正使用 (TA_WSGL 属性のイベントフラグに待ちタスクが存在)

■ 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。p_flgptn の指す領域には、待ち解除されるときのイベントフラグのビットパターンを返します。
対象イベントフラグが TA_WSGL 属性の場合、既に他のタスクが待っている場合は、エラーE_ILUSE を返します。
本サービスコール呼び出し時にすでに待ち解除条件を満たしている場合は、すぐにリターンし、E_OK を返します。待ち解除条件を満たしていない時、wai_flg,twai_flg サービスコールの場合は、イベントフラグ待ち行列につながれます。その際、flgid のイベントフラグ属性が TA_TFIFO の場合は、FIFO 順で待ち行列にタスクをつなぎ、TA_TPRI の場合は、優先度順でタスクをつなぎます。pol_flg,ipol_flg サービスコールの場合は、直ちにリターンし、エラーE_TMOUT を返します。
twai_flg サービスコールの場合は、tmout に待ち時間を ms 単位で指定します。tmout に指定可能な値は、(0x7FFFFFFF-タイムティック)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。
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 サービスコールと同じ動作になります。

wai_flg,twai_flg サービス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。

- **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 経過し、最初のタイムティックが発生した場合**
  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 サービス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。

wfmode の指定方法および各モードの意味は以下のとおりです。

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_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)。

タスクコンテキストにおいては、wai_flg,twai_flg,pol_flg サービスコール、非タスクコンテキストにおいては、ipol_flg を使用してください。

■ 使用例

《 C 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
   UINT flgptn;
              :
     if(wai_flg(ID_flg2, (FLGPTN)0x0ff0, TWF_ANDW, &flgptn)!=E_OK)
       error("WAITING Released¥n");
              :
              :
     if(pol_flg(ID_flg2, (FLGPTN)0x0ff0, TWF_ORW, &flgptn)!=E_OK)
       printf("Not set EventFlag¥n");
              :
              :
     if( twai_flg(ID_flg2, (FLGPTN)0x0ff0, TWF_ANDW, &flgptn, 5) != E_OK )
       error("WAITING Released¥n");
              :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0,R1,R3
    wai_flg     #ID_FLG1,#0003H,#TWF_ANDW
          :
    PUSHM       A0,R1,R3
    pol_flg     #ID_FLG2,#0008H,#TWF_ORW
          :
    PUSHM       A0,R1,R3
    wai_flg     #ID_FLG3,#0003H,#TWF_ANDW,20
          :
```

ref_flg　　イベントフラグの状態参照
iref_flg　　イベントフラグの状態参照（ハンドラ専用）

■ C 言語 API

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

● パラメータ

ID	flgid	対象イベントフラグ ID 番号
T_RFLG	*pk_rflg	イベントフラグ状態を返すパケットへのポインタ

● リターンパラメータ

ER	ercd	正常終了(E_OK)
T_RFLG	*pk_rflg	イベントフラグ状態を返すパケットへのポインタ

pk_rflg の内容

```
typedef   struct t_rflg{
    ID      wtskid   +0   2     待ちタスク ID
    FLGPTN  flgptn   +2   2     現在のイベントフラグビットパターン
} T_RFLG;
```

■ アセンブリ言語 API

```
.include mr308.inc
ref_flg   FLGID, PK_RFLG
iref_flg  FLGID, PK_RFLG
```

● パラメータ

FLGID	対象イベントフラグ ID 番号
PK_RFLG	イベントフラグ状態を返すパケットへのポインタ

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象イベントフラグ ID 番号
A1	イベントフラグ状態を返すパケットへのポインタ

■エラーコード

なし

■ 機能説明

flgid で示されたイベントフラグの各種の状態を返します。

- **wtskid**
  wtskid には待ち行列の先頭タスク(次に待ち行列から削除されるタスク)の ID 番号を返します。待ちタスクの無い場合は TSK_NONE を返します。
- **flgptn**
  flgptn には、現在のイベントフラグビットパターンを返します。

本サービスコールは、タスクコンテキストからは、ref_flg、非タスクコンテキストからは、iref_flg を使用してください。

■ 使用例

《 c 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
     T_RFLG rflg;
     ER ercd;
         :
     ercd = ref_flg( ID_FLG1, &rflg );
         :
 }
```

《 アセンブリ言語の使用例 》

```
_ refflg:       .blkb   4
    .include mr308.inc
    .GLB        task
task:
         :
       PUSHM   A0,A1
       ref_flg #ID_FLG1,#_refflg
         :
```

1.5. 同期・通信機能(データキュー)

表 5 データキュー機能の仕様にデータキュー機能の仕様を示します。

項番	項目	内容
1	データキューID	1～255
2	データキュー領域の容量(データの個数)	0～65535
3	データサイズ	32 ビット
4	データキュー属性	TA_TFIFO : 待ちタスクのキューイングは FIFO 順 TA_TPRI :待ちタスクのキューイングは優先度順

表 5 データキュー機能の仕様

snd_dtq	**データキューへのデータ送信**
psnd_dtq	**データキューへのデータ送信(ポーリング)**
ipsnd_dtq	**データキューへのデータ送信(ポーリング、ハンドラ専用)**
tsnd_dtq	**データキューへのデータ送信(タイムアウト)**
fsnd_dtq	**データキューへのデータ強制送信**
ifsnd_dtq	**データキューへのデータ強制送信(ハンドラ専用)**

■ C 言語 API

```
ER ercd = snd_dtq( ID dtqid, VP_INT data );
ER ercd = psnd_dtq( ID dtqid, VP_INT data );
ER ercd = ipsnd_dtq( ID dtqid, VP_INT data );
ER ercd = tsnd_dtq( ID dtqid, VP_INT data, TMO tmout );
ER ercd = fsnd_dtq( ID dtqid, VP_INT data );
ER ercd = ifsnd_dtq( ID dtqid, VP_INT data );
```

● ■ パラメータ

ID	dtqid	対象データキューID 番号
TMO	tmout	タイムアウト値(tsnd_dtq の場合)
VP_INT	data	送信するデータ

● ■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ アセンブリ言語 API

```
.include mr308.inc
snd_dtq   DTQID, DTQDATA
isnd_dtq  DTQID, DTQDATA  ipsnd_dtq DTQID, DTQDATA
psnd_dtq  DTQID, DTQDATA
ipsnd_dtq  DTQID, DTQDATA
tsnd_dtq  DTQID, DTQDATA,TMO
fsnd_dtq  DTQID, DTQDATA
ifsnd_dtq  DTQID, DTQDATA
```

● パラメータ

DTQID	対象データキューID 番号
DTQDATA	送信するデータ
TMO	タイムアウト値(tsnd_dtq の場合)

● サービスコール発行後のレジスタ内容

snd_dtq,psnd_dtq,ipsnd_dtq,fsnd_dtq,ifsnd_dtq の場合

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	データ(下位 16bit)
R3	データ(上位 16bit)
A0	対象データキューID 番号

tsnd_dtq の場合

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	データ(下位 16bit)
R2	タイムアウト値(上位 16bit)
R3	データ(上位 16bit)
A0	対象データキューID 番号

■ エラーコード

E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト
E_ILUSE	サービスコール不正使用 (dtqcnt が 0 のデータキューに対して fsnd_dtq, ifsnd_dtq を発行)
EV_RST	データキュー領域クリアによって待ち状態が解除された

■ 機能説明

dtqid で示されたデータキューに対して、data で示された 32bit のデータを送信します。対象データキューに受信待ちタスクが存在する場合は、データキューにデータを格納せず、受信待ち行列の先頭タスクにデータを送信し、そのタスクの受信待ち状態を解除します。
一方、既にデータで一杯になったデータキューに対して、snd_dtq,tsnd_dtq を発行した場合、これらのサービスコールを発行したタスクは、実行状態からデータ送信待ち状態に移行し、データキューの空きを待つ送信待ち行列につながれます。その際、dtqid のデータキュー属性が TA_TFIFO の場合は、FIFO 順で待ち行列にタスクをつなぎ、TA_TPRI の場合は、優先度順でタスクをつなぎます。psnd_dtq,ipsnd_dtq の場合は、直ちにリターンし、エラーE_TMOUT を返します。
tsnd_dtq サービスコールの場合は、tmout には、待ち時間を ms 単位で指定します。tmout に指定可能な値は、(0x7FFFFFFF-タイムティック)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。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_dtq 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、snd_dtq サービスコールと同じ動作をします。
受信待ちタスクがなく、データキュー領域も一杯でない場合、送信したデータはデータキューに格納されます。
snd_dtq,tsnd_dtq サービス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。

- **tmout の時間が経過する前に、rcv_dtq,trcv_dtq,prcv_dtq,iprcv_dtq サービスコールが発行され、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 経過し、最初のタイムティックが発生した場合**
  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 サービス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 vrst_dtq サービスコールによって待ち状態の対象となっているデータキューが削除された場合**
  この場合、エラーコードは、EV_RST を返します。

fsnd_dtq,ifsnd_dtq の場合は、データキューの先頭(最古)のデータを削除し、送信データをデータキュー末尾に格納します。データキュー領域がデータで一杯になっていない場合は、fsnd_dtq,ifsnd_dtq は、snd_dtq と同じ動作を行います。
タスクコンテキストにおいては、snd_dtq,tsnd_dtq,psnd_dtq,fsnd_dtq サービスコール、非タスクコンテキストにおいては、ipsnd_dtq,ifsnd_dtq を使用してください。

■ 使用例

《 C言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 VP_INT data[10];
 void task(void)
 {
              :
     if( snd_dtq( ID_dtq, data[0]) == E_RLWAI ){
       error("Forced released¥n");
   }
              :
     if( psnd_dtq( ID_dtq, data[1])== E_TMOUT ){
       error("Timeout¥n");
   }
              :
     if( tsnd_dtq( ID_dtq, data[2], 10 ) != E_ TMOUT ){
       error("Timeout ¥n");
   }
              :
     if( fsnd_dtq( ID_dtq, data[3]) != E_OK ){
       error("error¥n");
   }
              :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
_g_dtq: .LWORD  12345678H
task:
          :
    PUSHM       R1,R2,R3,A0
    tsnd_dtq    #ID_DTQ1,_g_dtq,#100
          :
    PUSHM       R1,R3,A0
    psnd_dtq    #ID_DTQ2,#0FFFFH
          :
    PUSHM       R1,R3,A0
    fsnd_dtq    #ID_DTQ3,#0ABCDH
          :
```

rcv_dtq	**データキューからのデータ受信**
prcv_dtq	**データキューからのデータ受信（ポーリング）**
iprcv_dtq	**データキューからのデータ受信（ポーリング、ハンドラ専用）**
trcv_dtq	**データキューからのデータ受信（タイムアウト）**

■ C 言語 API

```
ER ercd = rcv_dtq( ID dtqid, VP_INT *p_data );
ER ercd = prcv_dtq( ID dtqid, VP_INT *p_data );
ER ercd = iprcv_dtq( ID dtqid, VP_INT *p_data );
ER ercd = trcv_dtq( ID dtqid, VP_INT *p_data, TMO tmout );
```

● パラメータ

ID	dtqid	対象データキューID 番号
TMO	tmout	タイムアウト値(trcv_dtq の場合)
VP_INT	*p_data	受信データを格納する領域先頭へのポインタ

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
VP_INT	*p_data	受信データを格納する領域先頭へのポインタ

■ アセンブリ言語 API

```
.include mr308.inc
rcv_dtq   DTQID
prcv_dtq  DTQID
iprcv_dtq  DTQID
trcv_dtq  DTQID,TMO
```

● パラメータ

DTQID	対象データキューID 番号
TMO	タイムアウト値(trcv_dtq の場合)

● サービスコール発行後のレジスタ内容

rcv_dtq,prcv_dtq,iprcv_dtq の場合

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	受信データ(下位 16bit)
R3	受信データ(上位 16bit)
A0	対象データキューID 番号

trcv_dtq の場合

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	受信データ(下位 16bit)
R2	タイムアウト値(上位 16bit)
R3	受信データ(上位 16bit)
A0	対象データキューID 番号

■ エラーコード

E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト

■ 機能説明

dtqid で示されたデータキューから、データを受信し、p_data の指す領域に格納します。対象データキューにデータが存在する場合は、その先頭の(最古の)データを受信します。この結果、データキュー領域に空きが発生するため、送信待ち行列につながれているタスクは、その送信待ち状態が解除され、データキュー領域へのデータを送信します。

データキューにデータが存在せず、データ送信待ちタスクが存在する場合(データキュー領域の容量が 0 の場合)、データ送信待ち行列先頭タスクのデータを受信します。この結果、そのデータ送信待ちタスクの待ち状態は解除されます。

一方、データキュー領域にデータが格納されていないデータキューに対して、rcv_dtq,trcv_dtq を発行した場合、これらのサービスコールを発行したタスクは、実行状態からデータ受信待ち状態に移行し、データ受信待ち行列につながれます。このとき、受信待ち行列へは、FIFO 順でつながれます。prcv_dtq,iprcv_dtq の場合は、直ちにリターンし、エラーE_TMOUT を返します。

trcv_dtq サービスコールの場合は、tmout には、待ち時間を ms 単位で指定します。tmout に指定可能な値は、(0x7FFFFFFF-タイムティック)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。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_dtq 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_dtq サービスコールと同じ動作をします。

rcv_dtq,trcv_dtq サービス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。

- **tmout の時間が経過する前に、snd_dtq,tsnd_dtq,psnd_dtq,ipsnd_dtq サービスコールが発行され、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 経過し、最初のタイムティックが発生した場合**
  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 サービス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。

タスクコンテキストにおいては、rcv_dtq,trcv_dtq,prcv_dtq サービスコール、非タスクコンテキストにおいては、iprcv_dtq を使用してください。

■ 使用例

《 C言語の使用例 》

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
    VP_INT data;
                :
      if( rcv_dtq( ID_dtq, &data ) != E_RLWAI )
          error("forced wakeup¥n");
                :
      if( prcv_dtq( ID_dtq, &data ) != E_TMOUT )
          error("Timeout¥n");
                :
      if( trcv_dtq( ID_dtq, &data, 10 ) != E_TMOUT )
          error("Timeout ¥n");
                :
  }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    trcv_dtq    #ID_DTQ1,#TMO_POL
                  :
    PUSHM       A0
    prcv_dtq    #ID_DTQ2
          :
    PUSHM       A0
    rcv_dtq     #ID_DTQ2
          :
```

ref_dtq　　データキューの状態参照
iref_dtq　　データキューの状態参照（ハンドラ専用）

■ C 言語 API

```
ER ercd = ref_dtq( ID dtqid, T_RDTQ *pk_rdtq );
ER ercd = iref_dtq( ID dtqid, T_RDTQ *pk_rdtq );
```

● パラメータ

ID	dtqid	対象データキューID 番号
T_RDTQ	*pk_rdtq	データキュー状態を返すパケットへのポインタ

● リターンパラメータ

ER	ercd	正常終了(E_OK)
T_RDTQ	*pk_rdtq	データキュー状態を返すパケットへのポインタ

pk_rdtq の内容

```
typedef   struct t_rdtq{
    ID      stskid   +0   2    送信待ちタスク ID
    ID      wtskid   +2   2    受信待ちタスク ID
    UINT    sdtqcnt  +4   2    データキューに入っているデータ数
} T_RDTQ;
```

■ アセンブリ言語 API

```
.include mr308.inc
ref_dtq   DTQID, PK_RDTQ
iref_dtq  DTQID, PK_RDTQ
```

● パラメータ

DTQID	対象データキューID 番号
PK_RDTQ	データキュー状態を返すパケットへのポインタ

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象データキューID 番号
A1	データキュー状態を返すパケットへのポインタ

■エラーコード

なし

■ 機能説明

dtqid で示されたデータキューの各種の状態を返します。

- **stskid**
  stskid には送信待ち行列の先頭タスク(次に待ち行列から削除されるタスク)の ID 番号を返します。待ちタスクの無い場合は TSK_NONE を返します。
- **wtskid**
  wtskid には受信待ち行列の先頭タスク(次に待ち行列から削除されるタスク)の ID 番号を返します。待ちタスクの無い場合は TSK_NONE を返します。
- **sdtqcnt**
  sdtqcnt には、データキュー領域に格納されているデータ個数を返します。

本サービスコールは、タスクコンテキストからは、ref_dtq、非タスクコンテキストからは、iref_dtq を使用してください。

■ 使用例

《 c 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
     T_RDTQ rdtq;
     ER ercd;
        :
     ercd = ref_dtq( ID_DTQ1, &rdtq );
        :
 }
```

《 アセンブリ言語の使用例 》

```
_ refdtq:       .blkb   6
    .include mr308.inc
    .GLB        task
task:
        :
       PUSHM   A0,A1
       ref_dtq #ID_DTQ1,#_refdtq
        :
```

1.6. 同期・通信機能(メールボックス)

表 6 メールボックス機能の仕様にメールボックス機能の仕様を示します。

項番	項目	内容
1	メールボックス ID	1～255
2	メッセージ優先度	1～255
3	メールボックス属性	TA_TFIFO : 待ちタスクのキューイングは FIFO TA_TPRI : 待ちタスクのキューイングは優先度順 TA_MFIFO : メッセージのキューイングは FIFO TA_MPRI : メッセージのキューイングは優先度順

表 6 メールボックス機能の仕様

snd_mbx　　メールボックスへのメッセージ送信
isnd_mbx　　メールボックスへのメッセージ送信（ハンドラ専用）

■ C 言語 API

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

● パラメータ

ID	mbxid	対象メールボックス ID 番号
T_MSG	*pk_msg	送信するメッセージ

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

```
.include mr308.inc
snd_mbx   MBXID,PK_MBX
isnd_mbx  MBXID,PK_MBX
```

● パラメータ

MBXID	対象メールボックス ID 番号
PK_MBX	送信するメッセージ(アドレス)

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象メールボックス ID 番号
A1	送信するメッセージ(アドレス)

■ メッセージパケットの構造

<<メールボックスのメッセージヘッダ>>

```
typedef   struct t_msg{
    VP      msghead   +0    4     カーネル管理領域
} T_MSG;
```

<<メールボックスの優先度付きメッセージヘッダ>>

```
typedef   struct t_msg{
    T_MSG   msgque    +0    4     メッセージヘッダ
    PRI     msgpri    +4    2     メッセージ優先度
} T_MSG;
```

■ エラーコード

なし

■ 機能説明

mbxid で示されたメールボックスに pk_msg で示されたメッセージを送信します。T_MSG*は、far ポインタとして扱います。 すでに対象メールボックスにメッセージの受信を待つタスクが存在していれば、待ち行列先頭のタスクに送信したメッセージが渡され、そのタスクの待ち状態が解除されます。

TA_MFIFO 属性のメールボックスにメッセージを送る場合は、以下の例に示すように先頭に T_MSG 構造体を付加した形式で、メッセージを作成してください。

TA_MPRI 属性のメールボックスにメッセージを送る場合は、以下の例に示すように先頭に T_MSG_PRI 構造体を付加した形式で、メッセージを作成してください。

TA_MFIFO, TA_MPRI いずれの属性の場合もメッセージは RAM 領域に作成してください。

T_MSG の領域はカーネルが使用するため、送信後は書き換えてはなりません。メッセージ送信後、メッセージが受信される前にこの領域を書き換えた場合の動作は保証されません。

タスクコンテキストにおいては、snd_mbx サービスコール、非タスクコンテキストにおいては、isnd_mbx を使用してください。

<<メッセージの形式例>>

```
typedef struct user_msg{
        T_MSG  t_msg;         /* T_MSG 構造体 */
        B      data[16];      /* ユーザーメッセージデータ */
} USER_MSG;
```

<<優先度付きメッセージの形式例>>

```
typedef struct user_msg{
        T_MSG_PRI     t_msg;         /* T_MSG_PRI 構造体 */
        B             data[16];      /* ユーザーメッセージデータ */
} USER_MSG;
```

■ 使用例

《 C言語の使用例 》

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  typedef struct pri_message
  {
         T_MSG_PRI      msgheader;
         char    body[12];
  } PRI_MSG;

  void task(void)
  {
     PRI_MSG     msg;
                 :
     msg.msgpri = 5;
     snd_mbx( ID_msg,(T_MSG)&msg);
         :
  }
```

《 アセンブリ言語の使用例 》

```
     .include mr308.inc
     .GLB        task
_g_userMsg:      .blkb   4       ; Header
                 .blkb   12      ; Body
task:
          :
     PUSHM       A0,A1
     snd_mbx     #ID_MBX1,#_g_userMsg              :
           :
```

rcv_mbx　メールボックスからのメッセージ受信
prcv_mbx　メールボックスからのメッセージ受信（ポーリング）
iprcv_mbx　メールボックスからのメッセージ受信（ポーリング、ハンドラ専用）
trcv_mbx　メールボックスからのメッセージ受信（タイムアウト）

■ C 言語 API

```
ER ercd = rcv_mbx( ID mbxid, T_MSG **ppk_msg );
ER ercd = prcv_mbx( ID mbxid, T_MSG **ppk_msg );
ER ercd = iprcv_mbx( ID mbxid, T_MSG **ppk_msg );
ER ercd = trcv_mbx( ID mbxid, T_MSG **ppk_msg, TMO tmout );
```

● パラメータ

ID	mbxid	対象メールボックス ID 番号
TMO	tmout	タイムアウト値(trcv_mbx の場合)
T_MSG	**ppk_msg	受信メッセージを格納する領域先頭へのポインタ

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_MSG	**ppk_msg	受信メッセージを格納する領域先頭へのポインタ

■ アセンブリ言語 API

```
.include mr308.inc
rcv_mbx   MBXID
prcv_mbx  MBXID
iprcv_mbx MBXID
trcv_mbx  MBXID, TMO
```

● パラメータ

MBXID	対象メールボックス ID 番号
TMO	タイムアウト値(trcv_mbx の場合)

● サービスコール発行後のレジスタ内容

rcv_mbx,prcv_mbx,iprcv_mbx の場合

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	受信したメッセージ（上位アドレス ）
R2	受信したメッセージ（下位アドレス ）
A0	対象メールボックス ID 番号

trcv_mbx の場合

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	受信したメッセージ(上位アドレス)
R2	受信したメッセージ(下位アドレス)
R3	タイムアウト値(上位 16bit)
A0	対象メールボックス ID 番号

■ エラーコード

E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト

■ 機能説明

mbxid で示されたメールボックスから、メッセージを受信し、ppk_msg の指す領域に受信したメッセージの先頭アドレスを格納します。T_MSG**は、far ポインタとして扱います。 対象メールボックスにデータが存在する場合は、その先頭のデータを受信します。

一方、メールボックスにメッセージがないメールボックスに対して、rcv_mbx,trcv_mbx を発行した場合、これらのサービスコールを発行したタスクは、実行状態からメッセージ受信待ち状態に移行し、メッセージ受信待ち行列につながれます。その際、mbxid のメールボックス属性が TA_TFIFO の場合は、FIFO 順で待ち行列にタスクをつなぎ、TA_TPRI の場合は、優先度順でタスクをつなぎます。prcv_mbx,iprcv_mbx の場合は、直ちにリターンし、エラーE_TMOUT を返します。

trcv_mbx サービスコールの場合は、tmout には、待ち時間を ms 単位で指定します。tmout に指定可能な値は、(0x7FFFFFFF-タイムティック)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。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_mbx 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_mbx サービスコールと同じ動作をします。

rcv_mbx,trcv_mbx サービス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。

- **tmout の時間が経過する前に、rcv_mbx,trcv_mbx,prcv_mbx,iprcv_mbx サービスコールが発行され、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 経過し、最初のタイムティックが発生した場合**
  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 サービス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。

タスクコンテキストにおいては、rcv_mbx,trcv_mbx,prcv_mbx サービスコール、非タスクコンテキストにおいては、iprcv_mbx を使用してください。

■ 使用例

《 C言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"

 typedef struct fifo_message
 {
        T_MSG   head;
        char    body[12];
 } FIFO_MSG;
 void task()
 {
   FIFO_MSG *msg;
                :
     if( rcv_mbx((T_MSG *)&msg, ID_mbx) == E_RLWAI )
         error("forced wakeup¥n");
                :
                :
     if( prcv_mbx((T_MSG *)&msg, ID_mbx) != E_TMOUT )
         error("Timeout¥n");
                :
                :
     if( trcv_mbx((T_MSG *)&msg, ID_mbx,10) != E_TMOUT )
         error("Timeout¥n");
                :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       R3,A0
    trcv_mbx    #ID_MBX1,#100
          :
    PUSHM       R3,A0
    rcv_mbx     #ID_MBX1
          :
    PUSHM       R3,A0
    prcv_mbx    #ID_MBX1
          :
```

ref_mbx　　メールボックスの状態参照
iref_mbx　　メールボックスの状態参照（ハンドラ専用）

■ C 言語 API

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

● パラメータ

ID	mbxid	対象メールボックス ID 番号
T_RMBX	*pk_rmbx	メールボックス状態を返すパケットへのポインタ

● リターンパラメータ

ER	ercd	正常終了(E_OK)
T_RMBX	*pk_rmbx	メールボックス状態を返すパケットへのポインタ

pk_rmbx の内容

```
typedef   struct t_rmbx{
    ID      wtskid   +0   2     受信待ちタスク ID
    T_MSG   *pk_msg  +4   4     次に受信されるメッセージパケット
} T_RMBX;
```

■ アセンブリ言語 API

```
.include mr308.inc
ref_mbx   MBXID, PK_RMBX
iref_mbx  MBXID, PK_RMBX
```

● パラメータ

MBXID	対象メールボックス ID 番号
PK_RMBX	メールボックス状態を返すパケットへのポインタ

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象メールボックス ID 番号
A1	メールボックス状態を返すパケットへのポインタ

■エラーコード

なし

■ 機能説明

mbxid で示されたメールボックスの各種の状態を返します。

- **wtskid**
  wtskid には受信待ち行列の先頭タスク(次に待ち行列から削除されるタスク)の ID 番号を返します。待ちタスクの無い場合は TSK_NONE を返します。
- ***pk_msg**
  次に受信されるメッセージの先頭アドレスを返します。次に受信されるメッセージがない場合は、NULL を返します。

本サービスコールは、タスクコンテキストからは、ref_mbx、非タスクコンテキストからは、iref_mbx を使用してください。

■ 使用例

《 c 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
      T_RMBX rmbx;
      ER ercd;
           :
      ercd = ref_mbx( ID_MBX1, &rmbx );
           :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
_ refmbx:       .blkb   6
task:
           :
        PUSHM   A0,A1
        ref_mbx #ID_MBX1,#_refmbx
           :
```

1.7. メモリプール管理機能(固定長メモリプール)

表 7 固定長メモリプール機能の仕様に固定長メモリプール機能の仕様を示します。
メモリプール領域は、コンフィギュレーション時に、各メモリプール毎にセクション名を指定することが出来ます。

項番	項目	内容
1	固定長メモリプール ID	1～255
2	固定長メモリプール個数	1～65535
3	固定長メモリプールサイズ	4～65535
4	サポート属性	TA_TFIFO : 待ちタスクのキューイングは FIFO TA_TPRI : 待ちタスクのキューイングは優先度順
5	メモリプール領域の指定	メモリプール領域をセクション指定可能

表 7 固定長メモリプール機能の仕様

get_mpf　　固定長メモリブロックの獲得
pget_mpf　　固定長メモリブロックの獲得(ポーリング)
ipget_mpf　　固定長メモリブロックの獲得(ポーリング、ハンドラ専用)
tget_mpf　　固定長メモリブロックの獲得(タイムアウト)

■ C 言語 API

```
ER ercd = get_mpf( ID mpfid, VP *p_blk );
ER ercd = pget_mpf( ID mpfid, VP *p_blk );
ER ercd = ipget_mpf( ID mpfid, VP *p_blk );
ER ercd = tget_mpf( ID mpfid, VP *p_blk,TMO tmout );
```

● パラメータ

ID	mpfid	対象固定長メモリプール ID 番号
VP	*p_blk	獲得したメモリブロック先頭アドレスへのポインタ
TMO	tmout	タイムアウト値(tget_mpf の場合)

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
VP	*p_blk	獲得したメモリブロック先頭アドレスへのポインタ

■ アセンブリ言語 API

```
.include mr308.inc
get_mpf   MPFID
pget_mpf  MPFID
ipget_mpf MPFID
tget_mpf  MPFID,TMO
```

● パラメータ

MPFID	対象固定長メモリプール ID 番号
TMO	タイムアウト値(tget_mpf の場合)

● サービスコール発行後のレジスタ内容

get_mpf,pget_mpf,ipget_mpf の場合

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	獲得したブロックの先頭アドレス(下位 16bit)

R3	獲得したブロックの先頭アドレス(上位 16bit)
A0	対象固定長メモリプール ID 番号

tget_mpf の場合

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	獲得したブロックの先頭アドレス(下位 16bit)
R2	タイムアウト値(上位 16bit)
R3	獲得したブロックの先頭アドレス(上位 16bit)
A0	対象固定長メモリプール ID 番号

■ エラーコード

E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト
EV_RST	メモリプール領域クリアによって待ち状態が解除された

■ 機能説明

mpfid で示される固定長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。獲得したメモリブロックの内容は、不定です。
指定した固定長メモリプールにメモリブロックがない場合は、tget_mpf,get_mpf サービスコール使用時には、本サービス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つながれます。その際、mpfid の固定長メモリプール属性が TA_TFIFO の場合は、FIFO 順で待ち行列にタスクをつなぎ、TA_TPRI の場合は、優先度順でタスクをつなぎます。pget_mpf,ipget_mpf サービスコール使用時は、直ちにリターンし、エラーE_TMOUT を返します。
tget_mpf サービスコールの場合は、tmout には、待ち時間を ms 単位で指定します。tmout に指定可能な値は、(0x7FFFFFFF-タイムティック)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。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_mpf 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、get_mpf サービスコールと同じ動作をします。
get_mpf,tget_mpf サービス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。

- **tmout の時間が経過する前に、rel_mpf,irel_mpf サービスコールが発行され、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 経過し、最初のタイムティックが発生した場合**
  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 サービス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 vrst_mpf サービス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**
  この場合、エラーコードは、EV_RST を返します。

本サービスコールによって獲得されたメモリブロックの値は、初期化しないため不定となります。
本サービスコールは、タスクコンテキストからは、get_mpf,pget_mpf,tget_mpf、非タスクコンテキストからは、ipget_mpf を使用してください。

■ 使用例

《 C 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 VP      p_blk;
 void task()
 {
   if( get_mpf(ID_mpf ,&p_blk) != E_OK ){
         error("Not enough memory¥n");
   }
               :
   if( pget_mpf(ID_mpf ,&p_blk) != E_OK ){
         error("Not enough memory¥n");
   }
               :
   if( tget_mpf(ID_mpf ,&p_blk, 10) != E_OK ){
         error("Not enough memory¥n");
   }

 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    get_mpf     #ID_MPF1
          :
    PUSHM       A0
    pget_mpf    #ID_MPF1
          :
    PUSHM       A0
    tget_mpf    #ID_MPF1,#200
          :
```

rel_mpf　　固定長メモリプールブロックの解放
irel_mpf　　固定長メモリプールブロックの解放（ハンドラ専用）

■ C 言語 API

```
ER ercd = rel_mpf( ID mpfid, VP blk );
ER ercd = irel_mpf( ID mpfid, VP blk);
```

● パラメータ

ID	mpfid	対象固定長メモリプール ID 番号
VP	blk	返却するメモリブロックの先頭アドレス

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

```
.include mr308.inc
rel_mpf   MPFID,BLK
irel_mpf  MPFID,BLK
```

● パラメータ

MPFID	対象固定長メモリプール ID 番号
BLK	返却するメモリブロックの先頭アドレス

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	返却するメモリブロックの先頭アドレス(下位 16bit)
R3	返却するメモリブロックの先頭アドレス(上位 16bit)
A0	対象固定長メモリプール ID 番号

■ エラーコード

なし

■ 機能説明

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。返却するメモリブロックの先頭アドレスは必ず、get_mpf,tget_mpf,pget_mpf,ipget_mpf 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(RUNNING)状態あるいは実行可能(READY)状態へ移行します。本サービスコールは、blk の内容をチェックしません。従って、blk に正しいアドレスが格納されていない場合は正しく動作しません。
タスクコンテキストにおいては、rel_mpf サービスコール、非タスクコンテキストにおいては、irel_mpf を使用してください。

■ 使用例

《 C 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
     VP   p_blf;
     if( get_mpf(ID_mpf1,&p_blf) != E_OK )
         error("Not enough memory ¥n");
               :
     rel_mpf(ID_mpf1,p_blf);
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
_g_blk: .blkb 4
task:
          :
    PUSHM       A0
    get_mpf     #ID_MPF1
          :
    MOV.L       R3R1,_g_blk
    PUSHM       A0
    rel_mpf     #ID_MPF1,_g_blk
          :
```

ref_mpf　　固定長メモリプールの状態参照
iref_mpf　　固定長メモリプールの状態参照（ハンドラ専用）

■ C 言語 API

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

● パラメータ

ID	mpfid	対象固定長メモリプール ID 番号
T_RMPF	*pk_rmpf	固定長メモリプール状態を返すパケットへのポインタ

● リターンパラメータ

ER	ercd	正常終了(E_OK)
T_RMPF	*pk_rmpf	固定長メモリプール状態を返すパケットへのポインタ

pk_rmpf の内容

```
typedef   struct t_rmpf{
    ID      wtskid   +0   2     メモリブロック獲得待ちタスク ID
    UINT    fblkcnt  +2   2     空きメモリブロック数(個数)
} T_RMPF;
```

■ アセンブリ言語 API

```
.include mr308.inc
ref_mpf   MPFID,PK_RMPF
iref_mpf  MPFID,PK_RMPF
```

● パラメータ

MPFID	対象固定長メモリプール ID 番号
PK_RMPF	固定長メモリプール状態を返すパケットへのポインタ

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象固定長メモリプール ID 番号
A1	固定長メモリプール状態を返すパケットへのポインタ

■エラーコード

なし

■ 機能説明

mpfid で示されたメッセージバッファの各種の状態を返します。

- **wtskid**
  wtskid にはメモリブロック獲得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 TSK_NONE を返します。
- **fblkcnt**
  指定したメモリプールの空きブロック数を返します。

本サービスコールは、タスクコンテキストからは、rel_mpf、非タスクコンテキストからは、irel_mpf を使用してください。

■ 使用例

《 C 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
      T_RMPF rmpf;
      ER ercd;
           :
      ercd = ref_mpf( ID_MPF1, &rmpf );
           :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
_ refmpf:       .blkb   4
task:
           :
        PUSHM   A0,A1
        ref_mpf #ID_MPF1,#_refmpf
           :
```

1.8. メモリプール管理機能(可変長メモリプール)

表 8 可変長メモリプール機能の仕様に可変長メモリプール機能の仕様を示します。
メモリプール領域は、コンフィギュレーション時に、各メモリプール毎にセクション名を指定することが出来ます。

項番	項目	内容
1	可変長メモリプール ID	1～255
2	可変長メモリプールサイズ	16-524288
3	確保するメモリブロックの最大値	1-65520
4	サポート属性	メモリ不足時のタスク待ちの API は未サポート
5	メモリ領域の指定	可変長メモリプール領域をセクション指定可能

表 8 可変長メモリプール機能の仕様

pget_mpl　　可変長メモリブロックの獲得

■ C 言語 API

```
ER ercd = pget_mpl( ID mplid, UINT blksz, VP *p_blk );
```

● パラメータ

ID	mplid	対象可変長メモリプール ID 番号
UINT	blksz	獲得するメモリのサイズ(バイト数)
VP	*p_blk	獲得したメモリの先頭アドレスへのポインタ

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
VP	*p_blk	獲得したメモリブロック先頭アドレスへのポインタ

■ アセンブリ言語 API

```
.include mr308.inc
pget_mpl  MPLID,BLKSZ
```

● パラメータ

MPLID	対象可変長メモリプール ID 番号
BLKSZ	獲得するメモリのサイズ(バイト数)

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	獲得したメモリの先頭アドレス(下位 16bit)
R3	獲得したメモリの先頭アドレス(上位 16bit)
A0	対象可変長メモリプール ID 番号

■ エラーコード

E_TMOUT	メモリブロックなし

■ 機能説明

mplid で示される可変長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。獲得したメモリブロックの内容は、不定です。
指定した可変長メモリプールにメモリブロックがない場合は、直ちにリターンし、エラーE_TMOUT を返します。
本サービスコールによって獲得されたメモリブロックの値は、初期化しないため不定となります。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用した場

合は正常に動作しません。

■ 使用例

《 c 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 VP      p_blk;
 void task()
 {
   if( pget_mpl(ID_mpl , 200, &p_blk) != E_OK ){
        error("Not enough memory¥n");
   }
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    pget_mpl    #ID_MPL1,#200
          :
```

rel_mpl　　可変長メモリプールブロックの解放

■ C 言語 API

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

● パラメータ

ID	mplid	対象可変長メモリプール ID 番号
VP	blk	返却するメモリブロックの先頭アドレス

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ アセンブリ言語 API

```
.include mr308.inc
rel_mpl   MPLID,BLK
```

● パラメータ

MPLID	対象可変長メモリプール ID 番号
BLK	返却するメモリブロックの先頭アドレス

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	返却するメモリブロックの先頭アドレス(下位 16bit)
R3	返却するメモリブロックの先頭アドレス(上位 16bit)
A0	対象可変長メモリプール ID 番号

■ エラーコード

なし

■ 機能説明

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。返却するメモリブロックの先頭アドレスは必ず、pget_mpl で獲得した先頭アドレスを指定してください。
本サービスコールは、blk の内容をチェックしません。従って、blk に正しいアドレスが格納されていない場合は正しく動作しません。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用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 <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
     VP   p_blk;
     if( get_mpl(ID_mpl1, 200, &p_blk) != E_OK )
         error("Not enough memory ¥n");
               :
     rel_mpl(ID_mp1,p_blk);
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
_g_blk: .blkb 4
task:
          :
    PUSHM       A0
    get_mpl     #ID_MPL1,#200
          :
    MOV.L       R3R1,_g_blk
    PUSHM       A0
    rel_mpf     #ID_MPL1,_g_blk
          :
```

ref_mpl　　可変長メモリプールの状態参照
iref_mpl　　可変長メモリプールの状態参照(ハンドラ専用)

■ C 言語 API

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

● パラメータ

ID	mplid	対象可変長メモリプール ID 番号
T_RMPL	*pk_rmpl	可変長メモリプール状態を返すパケットへのポインタ

● リターンパラメータ

ER	ercd	正常終了(E_OK)
T_RMPL	*pk_rmpl	可変長メモリプール状態を返すパケットへのポインタ

pk_rmpl の内容

```
typedef   struct t_rmpl{
    ID      wtskid   +0   2    メモリ獲得待ちタスク ID(未使用)
    SIZE    fmplsz   +4   4    空きメモリサイズ(バイト数)
    UINT    fblksz   +8   2    すぐに獲得可能なメモリの最大サイズ(バイト数)
} T_RMPL;
```

■ アセンブリ言語 API

```
.include mr308.inc
ref_mpl   MPLID,PK_RMPL
iref_mpl  MPLID,PK_RMPL
```

● パラメータ

MPLID	対象可変長メモリプール ID 番号
PK_RMPL	可変長メモリプール状態を返すパケットへのポインタ

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象可変長メモリプール ID 番号
A1	可変長メモリプール状態を返すパケットへのポインタ

■エラーコード

なし

■ 機能説明

mplid で示されたメッセージバッファの各種の状態を返します。

- **wtskid**
  未使用。
- **fmplsz**
  空きメモリサイズを返します。
- **fblksz**
  すぐに獲得できるメモリの最大サイズを返します。

本サービスコールは、タスクコンテキストからは、ref_mpl、非タスクコンテキストからは、iref_mpl を使用してください。

■ 使用例

《 c 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
      T_RMPL rmpl;
      ER ercd;
           :
      ercd = ref_mpl( ID_MPL1, &rmpl );
           :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
_ refmpl:       .blkb   8
task:
           :
        PUSHM   A0,A1
        ref_mpl #ID_MPL1,_refmpl
           :
```

1.9. 時間管理機能

表 9 システム時間管理の仕様に時間管理機能の仕様を示します。

項番	項目	内容
1	システム時刻値	符号なし 48 ビット
2	システム時刻の単位	1[ms]
3	システム時刻の更新周期	ユーザ指定のタイムティック更新時間[ms]
4	システム時刻初期値(初期起動時)	000000000000H

表 9　システム時間管理の仕様

set_tim　　システム時刻の設定
iset_tim　　システム時刻の設定（ハンドラ専用）

■ C 言語 API

```
ER ercd = set_tim( SYSTIM *p_systim );
ER ercd = iset_tim( SYSTIM *p_systim );
```

● パラメータ

SYSTIM	*p_systim	設定するシステム時刻を示すパケットへのポインタ

p_systim の内容

```
typedef   struct t_systim {
    UH       utime     0     2     上位 16bit
    UW       ltime     +4    4     下位 32bit
} SYSTIM;
```

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

```
.include mr308.inc
set_tim   PK_TIM
iset_tim  PK_TIM
```

● パラメータ

PK_TIM	設定するシステム時刻を示すパケットへのポインタ

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	設定するシステム時刻を示すパケットへのポインタ

■ エラーコード

なし

■ 機能説明

システム時刻の現在値を p_systim で示される値に更新します。p_systim に指定する時刻の単位は、タイムティック数ではなく"ms"となります。

p_systtim に指定可能な値は、0x7FFF:FFFFFFFF 以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。

本サービスコールは、タスクコンテキストからは、set_tim、非タスクコンテキストからは、iset_tim を使用してください。

■ 使用例

《 C 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
   SYSTIME 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 */
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
_g_systim:
        .WORD   1111H
        .LWORD  22223333H
task:
          :
        PUSHM   A0
        set_tim #_g_systim
          :
```

get_tim　　システム時刻の参照
iget_tim　　システム時刻の参照（ハンドラ専用）

■ C 言語 API

```
ER ercd = get_tim( SYSTIM *p_systim );
ER ercd = iget_tim( SYSTIM *p_systim );
```

● パラメータ

SYSTIM	*p_systim	現在のシステム時刻を返すパケットへのポインタ

● リターンパラメータ

ER	ercd	正常終了(E_OK)
SYSTIM	*p_systim	現在のシステム時刻を返すパケットへのポインタ

p_systim の内容

```
typedef   struct t_systim {
    UH       utime     0     2     上位 16bit
    UW       ltime     +4    4     下位 32bit
} SYSTIM;
```

■ アセンブリ言語 API

```
.include mr308.inc
get_tim   PK_TIM
iget_tim  PK_TIM
```

● パラメータ

PK_TIM	現在のシステム時刻を返すパケットへのポインタ

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	現在のシステム時刻を返すパケットへのポインタ

■ エラーコード

なし

■ 機能説明

システム時刻の現在値を p_systim に格納します。
本サービスコールは、タスクコンテキストからは、get_tim、非タスクコンテキストからは、iget_tim を使用してください。

■ 使用例

《 C言語の使用例 》

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
     SYSTIME time;          /* 時刻データ格納変数 */
     get_tim( &time );      /* システム時刻の変更 */
  }
```

《 アセンブリ言語の使用例 》

```
     .include mr308.inc
     .GLB        task
_g_systim:       .blkb   6
task:
           :
         PUSHM   A0
         get_tim #_g_systim
           :
```

isig_tim　　タイムティックの供給

■ 機能説明

システム時刻を更新します。
コンフィギュレーションファイルにてシステムクロックを定義すると、tick_nume(ms)の間隔で isig_tim が自動的に起動されるようになっています。本機能は、サービスコールとして実装されていませんのでアプリケーションから呼び出すことは出来ません。
タイムティック供給時には、カーネルは、以下の処理を行います。

(1) システム時刻の更新
(2) アラームハンドラの起動
(3) 周期ハンドラの起動
(4) tslp_tsk などタイムアウト付きサービスコールで待ち状態になっているタスクのタイムアウト処理

1.10. 時間管理機能(周期ハンドラ)

表 10 周期ハンドラ機能の仕様に時間管理機能の仕様を示します。項番4周期ハンドラ属性の記述言語は、GUI コンフィギュレータでの指定内容です。コンフィギュレーションファイルには出力されず、カーネルも関知しません。

項番	項目	内容
1	周期ハンドラ ID	1～255
2	起動周期	0～7fffffff[ms]
3	起動位相	0～7fffffff[ms]
4	拡張情報	32bit
4	周期ハンドラ属性	TA_HLNG : 高級言語記述 TA_ASM : アセンブリ言語記述 TA_STA :周期ハンドラの動作開始 TA_PHS :起動位相の保存

表 10 周期ハンドラ機能の仕様

sta_cyc 周期ハンドラの動作開始
ista_cyc 周期ハンドラの動作開始(ハンドラ専用)

■ C 言語 API

```
ER ercd = sta_cyc( ID cycid );
ER ercd = ista_cyc( ID cycid );
```

● パラメータ

ID	cycid	対象周期ハンドラ ID 番号

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

```
.include mr308.inc
sta_cyc   CYCNO
ista_cyc  CYCNO
```

● パラメータ

CYCNO	対象周期ハンドラ ID 番号

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象周期ハンドラ ID 番号

■ エラーコード

なし

■ 機能説明

cycid で示された周期ハンドラを動作している状態に移行させます。周期ハンドラ属性に TA_PHS が指定されていない場合は、このサービスコールを呼び出した時刻を基準として、その時刻から起動周期が経過する毎に周期ハンドラが起動されます。

TA_PHS が指定されておらず、既に動作状態の周期ハンドラに対して本サービスコールを発行した場合、周期ハンドラが次に起動する時刻を再設定します。

TA_PHS が指定されており、既に動作状態の周期ハンドラに対して本サービスコールを発行した場合、本サービスコールは起動時刻の再設定はしません。

本サービスコールは、タスクコンテキストからは、sta_cyc、非タスクコンテキストからは、ista_cyc を使用してください。

■ 使用例

《 c 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
       :
   sta_cyc ( ID_cyc1 );
       :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
        PUSHM   A0
        sta_cyc #ID_CYC1
          :
```

stp_cyc　　周期ハンドラの動作停止
istp_cyc　　周期ハンドラの動作停止（ハンドラ専用）

■ C 言語 API

```
ER ercd = stp_cyc( ID cycid );
ER ercd = istp_cyc( ID cycid );
```

● パラメータ

ID	cycid	対象周期ハンドラ ID 番号

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

```
.include mr308.inc
stp_cyc   CYCNO
istp_cyc  CYCNO
```

● パラメータ

CYCNO	対象周期ハンドラ ID 番号

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象周期ハンドラ ID 番号

■ エラーコード

なし

■ 機能説明

cycid で示された周期ハンドラを動作していない状態に移行させます。
本サービスコールは、タスクコンテキストからは、stp_cyc、非タスクコンテキストからは、istp_cyc を使用してください。

■ 使用例

《 C言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
       :
   stp_cyc ( ID_cyc1 );
       :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
        PUSHM   A0
        stp_cyc #ID_CYC1
          :
```

ref_cyc　　周期ハンドラの状態参照
iref_cyc　　周期ハンドラの状態参照（ハンドラ専用）

■ C 言語 API

```
ER ercd = ref_cyc( ID cycid, T_RCYC *pk_rcyc );
ER ercd = iref_cyc( ID cycid, T_RCYC *pk_rcyc );
```

● パラメータ

ID	cycid	対象周期ハンドラ ID 番号
T_RCYC	*pk_rcyc	周期ハンドラ状態を返すパケットへのポインタ

● リターンパラメータ

ER	ercd	正常終了(E_OK)
T_RCYC	*pk_rcyc	周期ハンドラ状態を返すパケットへのポインタ

pk_rcyc の内容

```
typedef   struct t_rcyc{
    STAT    cycstat   +0    2     周期ハンドラの動作状態
    RELTIM  lefttim   +2    4     周期ハンドラ起動までの時間
} T_RCYC;
```

■ アセンブリ言語 API

```
.include mr308.inc
ref_cyc   ID,PK_RCYC
iref_cyc  ID,PK_RCYC
```

● パラメータ

CYCNO	対象周期ハンドラ ID 番号
PK_RCYC	周期ハンドラ状態を返すパケットへのポインタ

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象周期ハンドラ ID 番号
A1	周期ハンドラ状態を返すパケットへのポインタ

■エラーコード

なし

■ 機能説明

cycid で示された周期ハンドラの各種の状態を返します。

◆ cycstat

対象周期ハンドラの状態を返します。

・TCYC_STA　　周期ハンドラは動作している

・TCYC_STP　　周期ハンドラは動作していない

◆ **lefttim**

対象周期ハンドラの次に起動するまでの残り時間を返します。単位は、"ms"です。対象周期ハンドラが動作していない場合は、不定値となります。

本サービスコールは、タスクコンテキストからは、ref_cyc、非タスクコンテキストからは、iref_cyc を使用してください。

■ 使用例

《 c 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
     T_RCYC rcyc;
     ER ercd;
         :
     ercd = ref_cyc( ID_CYC1, &rcyc );
         :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
_ refcyc:       .blkb   6
task:
         :
       PUSHM   A0,A1
       ref_cyc #ID_CYC1,#_refcyc
         :
```

1.11. 時間管理機能(アラームハンドラ)

表 11 アラームハンドラ機能の仕様に時間管理機能の仕様を示します。項番4アラームハンドラ属性の記述言語は、GUI コンフィギュレータでの指定内容です。コンフィギュレーションファイルには出力されず、カーネルも関知しません。

項番	項目	内容
1	アラームハンドラ ID	1-255
2	起床時間	0～7fffffff [ms]
3	拡張情報	32bit
4	アラームハンドラ属性	TA_HLNG : 高級言語記述 TA_ASM : アセンブリ言語記述

表 11 アラームハンドラ機能の仕様

sta_alm　　アラームハンドラの動作開始
ista_alm　　アラームハンドラの動作開始(ハンドラ専用)

■ C 言語 API

```
ER ercd = sta_alm( ID almid, RELTIM almtim );
ER ercd = ista_alm( ID almid, RELTIM almtim );
```

● パラメータ

ID	almid	対象アラームハンドラ ID 番号
RELTIM	almtim	アラームハンドラの起動時刻(相対時間)

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

```
.include mr308.inc
sta_alm   ALMID,ALMTIM
ista_alm  ALMID,ALMTIM
```

● パラメータ

ALMID	対象アラームハンドラ ID 番号
ALMTIM	アラームハンドラの起動時刻(相対時間)

● サービスコール発行後のレジスタ内容

レジスタ名	サービス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起動時刻(相対時間)(下位 16bit)
R3	アラームハンドラの起動時刻(相対時間)(上位 16bit)
A0	対象アラームハンドラ ID 番号

■ エラーコード

なし

■ 機能説明

almid で示されたアラームハンドラの起動時刻を本サービスコールが呼び出された時刻から、almtim で指定された時間後の時刻と設定し、アラームハンドラを動作している状態に移行させます。
既に動作しているアラームハンドラが指定された場合は、以前の起動時刻の設定を解除し、新しい起動時刻に更新します。almtim に 0 が指定されて場合は、次のタイムティックでアラームハンドラが起動します。almtim に指定可能な値は、(0x7FFFFFFF-タイムティック)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。almtim に 0 を指定した場合は、次のタイムティックにてアラームハンドラを起動します。

本サービスコールは、タスクコンテキストからは、sta_alm、非タスクコンテキストからは、ista_alm を使用してください。

■ 使用例

《 C 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
       :
   sta_alm ( ID_alm1,100 );
       :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
        PUSHM   A0
        sta_alm #ID_ALM1,#100
          :
```

stp_alm　　アラームハンドラの動作停止
istp_alm　　アラームハンドラの動作停止（ハンドラ専用）

■ C 言語 API

```
ER ercd = stp_alm( ID almid );
ER ercd = istp_alm( ID almid );
```

● パラメータ

ID	almid	対象アラームハンドラ ID 番号

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

```
.include mr308.inc
stp_alm   ALMID
istp_alm  ALMID
```

● パラメータ

ALMID	対象アラームハンドラ ID 番号

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象アラームハンドラ ID 番号

■ エラーコード

なし

■ 機能説明

almid で示されたアラームハンドラを動作していない状態に移行させます。
本サービスコールは、タスクコンテキストからは、stp_alm、非タスクコンテキストからは、istp_alm を使用してください。

■ 使用例

《 C言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
       :
   stp_alm ( ID_alm1 );
       :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
        PUSHM   A0
        stp_alm #ID_ALM1
          :
```

ref_alm　　アラームハンドラの状態参照
iref_alm　　アラームハンドラの状態参照（ハンドラ専用）

■ C 言語 API

```
ER ercd = ref_alm( ID almid, T_RALM *pk_ralm );
ER ercd = iref_alm( ID almid, T_RALM *pk_ralm );
```

● パラメータ

ID	almid	対象アラームハンドラ ID 番号
T_RALM	*pk_ralm	アラームハンドラ状態を返すパケットへのポインタ

● リターンパラメータ

ER	ercd	正常終了(E_OK)
T_RALM	*pk_ralm	アラームハンドラ状態を返すパケットへのポインタ

pk_ralm の内容

```
typedef   struct t_ralm{
    STAT    almstat   +0    2     アラームハンドラの動作状態
    RELTIM  lefttim   +2    4     アラームハンドラ起動までの時間
} T_RALM;
```

■ アセンブリ言語 API

```
.include mr308.inc
ref_alm   ALMID,PK_RALM
iref_alm  ALMID,PK_RALM
```

● パラメータ

ALMID	対象アラームハンドラ ID 番号
PK_RALM	アラームハンドラ状態を返すパケットへのポインタ

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象アラームハンドラ ID 番号
A1	アラームハンドラ状態を返すパケットへのポインタ

■エラーコード

なし

■ 機能説明

almid で示されたアラームハンドラの各種の状態を返します。

◆ almstat

対象アラームハンドラの状態を返します。

・TALM_STA　　アラームハンドラは動作している
・TALM_STP　　アラームハンドラは動作していない

◆ **lefttim**

対象アラームハンドラの次に起動するまでの残り時間を返します。単位は、"ms"です。対象アラームハンドラが動作していない場合は、不定値となります。

本サービスコールは、タスクコンテキストからは、ref_alm、非タスクコンテキストからは、iref_alm を使用し

てください。

■ 使用例

《 C 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
     T_RALM ralm;
     ER ercd;
         :
     ercd = ref_alm( ID_ALM1, &ralm );
         :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
_ refalm:       .blkb   6
task:
         :
       PUSHM   A0,A1
       ref_alm #ID_ALM1,#_refalm
         :
```

1.12. システム状態管理機能

rot_rdq　タスク優先順位の回転
irot_rdq　タスク優先順位の回転（ハンドラ専用）

■ C 言語 API

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

● パラメータ

PRI	tskpri	回転するタスク優先度

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

```
.include mr308.inc
rot_rdq   TSKPRI
irot_rdq  TSKPRI
```

● パラメータ

TSKPRI	回転するタスク優先度

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R3	回転するタスク優先度

■ エラーコード

なし

■ 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 1-1に示します。

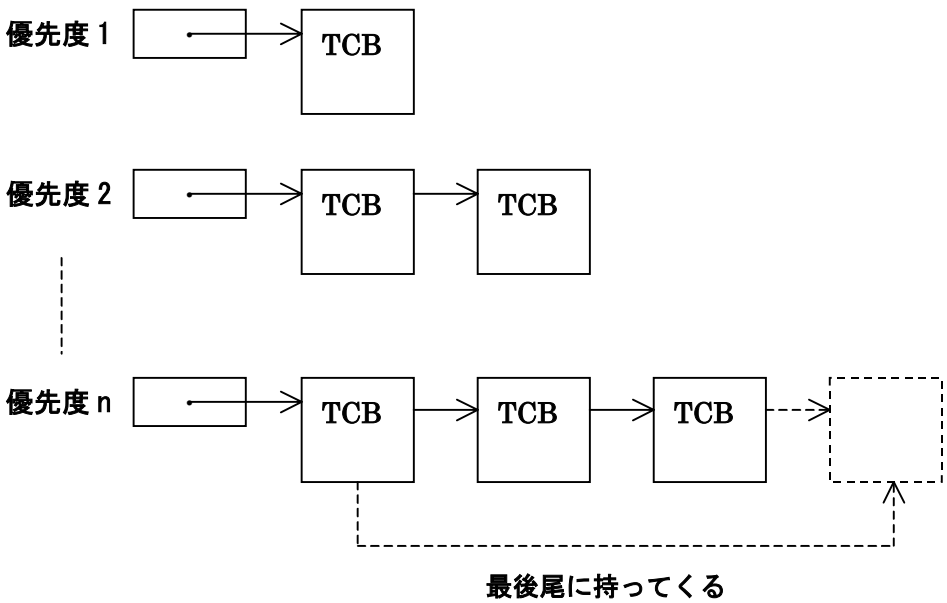


図 1-1　rot_rdq サービスコールによるレディキューの操作

このサービス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。rot_rdq サービスコール使用時は、tskpri=TPRI_SELF 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。irot_rdq サービスコールで TPRI_SELF を指定することは出来ません。指定してもエラーとなりません。
また、本サービス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。
本サービスコールは、タスクコンテキストからは、rot_rdq、非タスクコンテキストからは、irot_rdq を使用してください。

■ 使用例

《 c 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
          :
     rot_rdq( 2 );
          :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
        PUSHM   R3
        rot_rdq #2
          :
```

get_tid　　実行中タスク ID の参照
iget_tid　　実行中タスク ID の参照（ハンドラ専用）

■ C 言語 API

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

● パラメータ

ID	*p_tskid	タスク ID へのポインタ

● リターンパラメータ

ER	ercd	正常終了(E_OK)
ID	*p_tskid	タスク ID へのポインタ

■ アセンブリ言語 API

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

● パラメータ

なし

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	獲得したタスク ID

■ エラーコード

なし

■ 機能説明

実行状態のタスク ID を p_tskid の指す領域に返します。タスクから本サービスコールを発行した場合、自タスクの ID 番号を返します。また、非タスクコンテキストから本サービスコールを発行した場合は、そのとき実行していたタスク ID を返します。実行状態のタスクがない場合は、TSK_NONE を返します。
本サービスコールは、タスクコンテキストからは、get_tid、非タスクコンテキストからは、iget_tid を使用してください。

■ 使用例

《 C言語の使用例 》

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
     ID tskid;
         :
       get_tid(&tskid);
         :
  }
```

《 アセンブリ言語の使用例 》

```
     .include mr308.inc
     .GLB        task
task:
            :
         PUSHM   A0
         get_tid
            :
```

loc_cpu　　CPU ロック状態への移行
iloc_cpu　　CPU ロック状態への移行（ハンドラ専用）

■ C 言語 API

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

● パラメータ

なし

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

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

● パラメータ

なし

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード

■ エラーコード

なし

■ 機能説明

システム状態を CPU ロック状態とし、割込みとタスクのディスパッチを禁止します。CPU ロック状態の特長を以下に示します。

(1) CPU ロック状態の間は、タスクのスケジューリングは行われません。
(2) コンフィギュレータで定義したカーネル割込みマスクレベルより高いレベルの割込み以外の外部割り込みは、受け付けられません。
(3) CPU ロック状態から呼び出し可能なサービスコールは、以下のサービスコールのみです。その他のサービスコールが呼び出された場合の動作は保証されません。

- ext_tsk
- loc_cpu, iloc_cpu
- unl_cpu, iunl_cpu
- sns_ctx
- sns_loc
- sns_dsp
- sns_dpn

CPU ロック状態は、以下の操作で解除されます。

(a) unl_cpu, iunl_cpu サービスコールの呼び出し
(b) ext_tsk サービスコールの呼び出し

CPU ロック状態と CPU ロック解除状態の間の遷移は、loc_cpu, iloc_cpu, unl_cpu, iunl_cpu, ext_tsk サービスコールによってのみ発生します。割込みハンドラ、タイムイベントハンドラ終了時には、必ず CPU ロック解除状態でなければなりません。CPU ロック状態の場合、動作は保証されません。なお、これらのハンドラ開始時は、常に CPU ロック解除状態です。
すでに CPU ロック状態のときに、再度本サービスコールを呼び出してもエラーにはなりませんが、キューイングは行いません。
本サービスコールは、タスクコンテキストからは、loc_cpu、非タスクコンテキストからは、iloc_cpu を使用してください。

■ 使用例

《 C 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
       :
     loc_cpu();
       :
 }
```

《 アセンブリ言語の使用例 》

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

unl_cpu　　CPU ロック状態の解除
iunl_cpu　　CPU ロック状態の解除（ハンドラ専用）

■ C 言語 API

```
ER ercd = unl_cpu();
ER ercd = iunl_cpu();
```

● パラメータ

なし

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

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

● パラメータ

なし

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード

■ エラーコード

なし

■ 機能説明

loc_cpu, iloc_cpu サービスコールによって設定されていた CPU ロック状態を解除します。ディスパッチ許可状態から unl_cpu サービスコールを発行した場合、タスクのスケジューリングが行われます。割込みハンドラ内で iloc_cpu を呼び出し、CPU ロック状態に移行した場合は、割込みハンドラからリターンする前に必ず iunl_cpu を呼び出し、CPU ロック状態を解除してください。

CPU ロック状態とディスパッチ禁止状態は、独立して管理されます。そのため、unl_cpu, iunl_cpu サービスコールでは、ena_dsp サービスコールによるディスパッチ禁止状態は解除されません。

本サービスコールは、タスクコンテキストからは、unl_cpu、非タスクコンテキストからは、iunl_cpu を使用してください。

■ 使用例

《 C 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
       :
     unl_cpu();
       :
 }
```

《 アセンブリ言語の使用例 》

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

dis_dsp　　ディスパッチの禁止

■ C 言語 API

```
ER ercd = dis_dsp();
```

● パラメータ

なし

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

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

● パラメータ

なし

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード

■ エラーコード

なし

■ 機能説明

システム状態をディスパッチ禁止状態にします。ディスパッチ禁止状態の特長を、以下に示します。

(1) タスクのスケジューリングが行われなくなるため、自タスク以外のタスクが実行状態に移行することはなくなります。
(2) 割込みは受け付けられます。
(3) 待ち状態になるサービスコールを呼び出せません。

ディスパッチ禁止状態の間に以下の操作を行うと、システム状態はタスク実行状態に戻ります。

(a) ena_dsp サービスコールの呼び出し
(b) ext_tsk サービスコールの呼び出し

ディスパッチ禁止状態とディスパッチ許可状態の間の遷移は、dis_dsp, ena_dsp, ext_tsk サービスコールによってのみ発生します。
すでにディスパッチ禁止状態のときに再度本サービスコールを呼び出してもエラーにはなりませんが、キューイングは行いません。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用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 <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
       :
     dis_dsp();
       :
 }
```

《 アセンブリ言語の使用例 》

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

ena_dsp　　ディスパッチの許可

■ C 言語 API

```
ER ercd = ena_dsp();
```

● パラメータ

なし

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

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

● パラメータ

なし

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード

■ エラーコード

なし

■ 機能説明

dis_dsp サービスコール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ます。それにより、システムがタスク実行状態になった場合は、タスクのスケジューリングが行われます。
タスク実行状態から本サービスコールを呼び出してもエラーにはなりませんが、キューイングは行いません。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用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 <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
       :
     ena_dsp();
       :
 }
```

《 アセンブリ言語の使用例 》

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

sns_ctx　　コンテキストの参照

■ C 言語 API

```
BOOL state = sns_ctx();
```

● パラメータ

なし

● リターンパラメータ

BOOL	state	TRUE：非タスクコンテキスト FALSE：タスクコンテキスト

■ アセンブリ言語 API

```
.include mr308.inc
sns_ctx
```

● パラメータ

なし

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	TRUE：非タスクコンテキスト FALSE：タスクコンテキスト

■ エラーコード

なし

■ 機能説明

非タスクコンテキストから呼び出された場合に TRUE、タスクコンテキストから呼び出された場合に FALSE を返します。本サービスコールは、CPU ロック状態からも呼び出せます。

■ 使用例

《 C言語の使用例 》

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
     BOOL stat;
         :
       stat = sns_ctx();
         :
  }
```

《 アセンブリ言語の使用例 》

```
     .include mr308.inc
     .GLB        task
task:
            :
         sns_ctx
            :
```

sns_loc　　CPU ロック状態の参照

■ C 言語 API

```
BOOL state = sns_loc();
```

● パラメータ

なし

● リターンパラメータ

BOOL	state	TRUE：CPU ロック状態 FALSE：CPU ロック解除状態

■ アセンブリ言語 API

```
.include mr308.inc
sns_loc
```

● パラメータ

なし

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	TRUE：CPU ロック状態 FALSE：CPU ロック解除状態

■ エラーコード

なし

■ 機能説明

システムが CPU ロック状態の場合に TRUE、CPU ロック解除状態の場合に FALSE を返します。本サービスコールは、CPU ロック状態からも呼び出せます。

■ 使用例

《 C言語の使用例 》

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
     BOOL stat;
         :
       stat = sns_loc();
         :
  }
```

《 アセンブリ言語の使用例 》

```
     .include mr308.inc
     .GLB        task
task:
            :
         sns_loc
            :
```

sns_dsp　　ディスパッチ禁止状態の参照

■ C 言語 API

```
BOOL state = sns_dsp();
```

● パラメータ

なし

● リターンパラメータ

BOOL	state	TRUE：ディスパッチ禁止状態 FALSE：ディスパッチ許可状態

■ アセンブリ言語 API

```
.include mr308.inc
sns_dsp
```

● パラメータ

なし

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	TRUE：ディスパッチ禁止状態 FALSE：ディスパッチ許可状態

■ エラーコード

なし

■ 機能説明

システムがディスパッチ禁止状態の場合に TRUE、ディスパッチ許可状態の場合に FALSE を返します。
本サービスコールは、CPU ロック状態からも呼び出せます。

■ 使用例

《 C言語の使用例 》

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
     BOOL stat;
         :
       stat = sns_dsp();
         :
  }
```

《 アセンブリ言語の使用例 》

```
     .include mr308.inc
     .GLB        task
task:
           :
         sns_dsp
           :
```

sns_dpn　　ディスパッチ保留状態の参照

■ C 言語 API

```
BOOL state = sns_dpn();
```

● パラメータ

なし

● リターンパラメータ

BOOL	state	TRUE:ディスパッチ保留状態 FALSE:ディスパッチ保留状態ではない

■ アセンブリ言語 API

```
.include mr308.inc
sns_dpn
```

● パラメータ

なし

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	TRUE:ディスパッチ保留状態 FALSE:ディスパッチ保留状態ではない

■ エラーコード

なし

■ 機能説明

システムがディスパッチ保留状態の場合に TRUE、そうでない場合に FALSE を返します。具体的には、以下の全ての条件が満足される場合に FALSE を返し、その他の場合には TRUE を返します。

(1) ディスパッチ禁止状態でない
(2) CPU ロック状態でない
(3) タスクである

本サービスコールは、CPU ロック状態からも呼び出せます。システムがディスパッチ禁止状態の場合に TRUE、ディスパッチ許可状態の場合に FALSE を返します。

■ 使用例

《 C言語の使用例 》

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
     BOOL stat;
         :
       stat = sns_dpn();
         :
  }
```

《 アセンブリ言語の使用例 》

```
     .include mr308.inc
     .GLB        task
task:
            :
         sns_dpn
            :
```

1.13. 割込管理機能

ret_int　　割り込みハンドラからの復帰(アセンブリ言語記述時)

■ C 言語 API

本サービスコールは、C 言語では記述できません。[6]

■ アセンブリ言語による呼び出し方法

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

■ パラメータ

なし

■ エラーコード

本サービス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

割り込みハンドラからの復帰処理を行います。復帰処理に応じてスケジューラを動作させ、タスクの切り替えを行います。
割り込みハンドラの中でサービスコールを実行してもタスク切り替えは起こらず、割り込みハンドラを終了するまでタスク切り替えが遅延されます。
ただし、多重割り込み発生により起動された割り込みハンドラからの ret_int サービスコールの発行の場合はスケジューラを動作させません。タスクからの割り込みの場合のみスケジューラを動作させます。
なお、アセンブリ言語で記述する場合、本サービスコールは割り込みハンドラ入り口ルーチンから呼ばれたサブルーチンからは発行できません。必ず、割り込みハンドラの入り口ルーチンまたは入り口関数内で本サービスコールを実行してください。すなわち、以下のようなプログラムは正常に動作しません。

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

すなわち、以下のように記述してください。

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

本サービスコールは割り込みハンドラからのみ発行してください。周期起動ハンドラ、アラームハンドラ及びタスクから発行した場合は、正常に動作しません。

[6] 割り込みハンドラの開始関数を#pragma INTHANDLER で宣言すると、関数の出口で自動的に ret_int サービスコールを発行します。

1.14. システム構成理機能

ref_ver	**バージョン情報の参照**
iref_ver	**バージョン情報の参照（ハンドラ専用）**

■ C 言語 API

```
ER ercd = ref_ver( T_RVER *pk_rver );
ER ercd = iref_ver( T_RVER *pk_rver );
```

● パラメータ

T_RVER	*pk_rver	バージョン情報を返すパケットへのポインタ

pk_rver の内容

```
typedef   struct t_rver {
    UH      maker    0     2     カーネルのメーカーコード
    UH      prid     +2    2     カーネルの識別番号
    UH      spver    +4    2     ITRON 仕様のバージョン番号
    UH      prver    +6    2     カーネルのバージョン番号
    UH      prno[4]  +8    2     カーネル製品の管理情報
} T_RVER;
```

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

```
.include mr308.inc
ref_ver   PK_VER
iref_ver  PK_VER
```

● パラメータ

PK_VER	バージョン情報を返すパケットへのポインタ

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	バージョン情報を返すパケットへのポインタ

■ エラーコード

なし

■ 機能説明

現在実行中のカーネルのバージョンに関する情報を読み出し、その結果を pk_rver の指す領域に返します。

pk_rver の指すパケットには、次の情報を返します。

- maker
  株式会社ルネサステクノロジを示す H'115 が返されます。
- prid
  M3T-MR308 の内部識別 IDH'150 が返されます。
- spver
  μITRON 仕様書 Ver4.02.00 に準拠していることを示す H'5402 が返されます。
- prver
  M3T-MR308 カーネルのバージョンを示す H'0401 が返されます。
- prno
  - prno[0]
    拡張のための予約。
  - prno[1]
    製品のリリース年の下 2 桁(西暦)と月 H'0510 が得られます。
  - prno[2]
    拡張のための予約。
  - prno[3]
    拡張のための予約。

本サービスコールは、タスクコンテキストからは、ref_ver、非タスクコンテキストからは、iref_ver を使用してください。

■ 使用例

《 C 言語の使用例 》

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
void task()
{
    T_RVER   pk_rver;
    ref_ver( &pk_rver );
}
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
_ refver:       .blkb   6
task:
          :
        PUSHM   A0
        ref_ver #_refver
          :
```

1.15. 拡張機能(short データキュー)

表 12 short データキュー機能の仕様にデータキュー機能の仕様を示します。本機能は、データキューのデータを 16bit で扱います。本機能は、μITRON 4.0 仕様の仕様外の機能です。

項番	項目	内容
1	short データキューID	1～255
2	short データキュー領域の容量 （データの個数）	0～65535
3	short データサイズ	16 ビット
4	short データキュー属性	TA_TFIFO : 待ちタスクのキューイングは FIFO 順 TA_TPRI :待ちタスクのキューイングは優先度順

表 12　short データキュー機能の仕様

vsnd_dtq	**short データキューへのデータ送信**
vpsnd_dtq	**short データキューへのデータ送信（ポーリング）**
vipsnd_dtq	**short データキューへのデータ送信（ポーリング、ハンドラ専用）**
vtsnd_dtq	**short データキューへのデータ送信（タイムアウト）**
vfsnd_dtq	**short データキューへのデータ強制送信**
vifsnd_dtq	**short データキューへのデータ強制送信（ハンドラ専用）**

■ C 言語 API

```
ER ercd = vsnd_dtq( ID vdtqid, H data );
ER ercd = vpsnd_dtq( ID vdtqid, H data );
ER ercd = vipsnd_dtq( ID vdtqid, H data );
ER ercd = vtsnd_dtq( ID vdtqid, H data, TMO tmout );
ER ercd = vfsnd_dtq( ID vdtqid, H data );
ER ercd = vifsnd_dtq( ID vdtqid, H data );
```

● ■ パラメータ

ID	vdtqid	対象 short データキューID 番号
TMO	tmout	タイムアウト値(tsnd_dtq の場合)
H	data	送信するデータ

● ■ リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード

■ アセンブリ言語 API

```
.include mr308.inc
vsnd_dtq        VDTQID, DTQDATA
visnd_dtq       VDTQID, DTQDATA
vpsnd_dtq       VDTQID, DTQDATA
vipsnd_dtq      VDTQID, DTQDATA
vtsnd_dtq       VDTQID, DTQDATA,TMO
vfsnd_dtq       VDTQID, DTQDATA
vifsnd_dtq      VDTQID, DTQDATA
```

● パラメータ

VDTQID	対象 short データキューID 番号
DTQDATA	送信するデータ
TMO	タイムアウト値(tsnd_dtq の場合)

● サービスコール発行後のレジスタ内容

vsnd_dtq,vpsnd_dtq,vipsnd_dtq,vfsnd_dtq,vifsnd_dtq の場合

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	データ
A0	対象 short データキューID 番号

vtsnd_dtq の場合

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	データ
R2	タイムアウト値(上位 16bit)
A0	対象 short データキューID 番号

■ エラーコード

E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト
E_ILUSE	サービスコール不正使用(dtqcnt が 0 の short データキューに対して fsnd_dtq, ifsnd_dtq を発行)
EV_RST	short データキュー領域クリアによって待ち状態が解除された

■ 機能説明

vdtqid で示された short データキューに対して、data で示された符号付き 2 バイトのデータを送信します。
対象 short データキューに受信待ちタスクが存在する場合は、short データキューにデータを格納せず、受信待ち行列の先頭タスクにデータを送信し、そのタスクの受信待ち状態を解除します。
一方、既にデータで一杯になった short データキューに対して、vsnd_dtq,vtsnd_dtq を発行した場合、これらのサービスコールを発行したタスクは、実行状態からデータ送信待ち状態に移行し、short データキューの空きを待つ送信待ち行列につながれます。その際、vdtqid の short データキュー属性が TA_TFIFO の場合は、FIFO 順で待ち行列にタスクをつなぎ、TA_TPRI の場合は、優先度順でタスクをつなぎます。
vpsnd_dtq,vipsnd_dtq の場合は、直ちにリターンし、エラーE_TMOUT を返します。
vtsnd_dtq サービスコールの場合は、tmout には、待ち時間を ms 単位で指定します。tmout に指定可能な値は、(0x7FFFFFFF-タイムティック)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。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、vpsnd_dtq 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、vsnd_dtq サービスコールと同じ動作をします。
受信待ちタスクがなく、short データキュー領域も一杯でない場合、送信したデータは short データキューに格納されます。
vsnd_dtq,vtsnd_dtq サービス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。

◆ **tmout の時間が経過する前に、vrcv_dtq,vtrcv_dtq,vprcv_dtq,viprcv_dtq サービスコー**

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 経過し、最初のタイムティックが発生した場合**
  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 サービス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 vrst_vdtq サービスコールによって待ち状態の対象となっている short データキューが削除された場合**
  この場合、エラーコードは、EV_RST を返します。

vfsnd_dtq,vifsnd_dtq の場合は、short データキューの先頭(最古)のデータを削除し、送信データを short データキュー末尾に格納します。short データキュー領域がデータで一杯になっていない場合は、vfsnd_dtq,vifsnd_dtq は、vsnd_dtq と同じ動作を行います。
タスクコンテキストにおいては、vsnd_dtq,vtsnd_dtq,vpsnd_dtq,vfsnd_dtq サービスコール、非タスクコンテキストにおいては、vipsnd_dtq,vifsnd_dtq を使用してください。

■ 使用例

《 C言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 H data[10];
 void task(void)
 {
                :
     if( vsnd_dtq( ID_dtq, data[0]) == E_RLWAI ){
       error("Forced released¥n");
   }
                :
     if( vpsnd_dtq( ID_dtq, data[1])== E_TMOUT ){
       error("Timeout¥n");
   }
                :
     if( vtsnd_dtq( ID_dtq, data[2], 10 ) != E_ TMOUT ){
       error("Timeout ¥n");
   }
                :
     if( vfsnd_dtq( ID_dtq, data[3]) != E_OK ){
       error("error¥n");
   }
                :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
_g_dtq: .WORD   1234H
task:
          :
    PUSHM       R1,R2,A0
    vtsnd_dtq   #ID_DTQ1,_g_dtq,#100
          :
    PUSHM       R1,A0
    vpsnd_dtq   #ID_DTQ2,#0FFFFH
          :
    PUSHM       R1,A0
    vfsnd_dtq   #ID_DTQ3,#0ABCDH
          :
```

vrcv_dtq　short データキューからのデータ受信
vprcv_dtq　short データキューからのデータ受信（ポーリング）
viprcv_dtq　short データキューからのデータ受信（ポーリング、ハンドラ専用）
vtrcv_dtq　short データキューからのデータ受信（タイムアウト）

■ C 言語 API

```
ER ercd = vrcv_dtq( ID dtqid, H *p_data );
ER ercd = vprcv_dtq( ID dtqid, H *p_data );
ER ercd = viprcv_dtq( ID dtqid, H *p_data );
ER ercd = vtrcv_dtq( ID dtqid, H *p_data, TMO tmout );
```

● パラメータ

ID	vdtqid	対象 short データキューID 番号
TMO	tmout	タイムアウト値(vtrcv_dtq の場合)
H	*p_data	受信データを格納する領域先頭へのポインタ

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
H	*p_data	受信データを格納する領域先頭へのポインタ

■ アセンブリ言語 API

```
.include mr308.inc
vrcv_dtq        VDTQID
vprcv_dtq       VDTQID
viprcv_dtq      VDTQID
vtrcv_dtq       VDTQID,TMO
```

● パラメータ

VDTQID	対象 short データキューID 番号
TMO	タイムアウト値(trcv_dtq の場合)

● サービスコール発行後のレジスタ内容

vrcv_dtq,vprcv_dtq,viprcv_dtq の場合

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	受信データ
A0	対象 short データキューID 番号

vtrcv_dtq の場合

レジスタ名	サービスコール発行後の内容
R0	エラーコード
R1	受信データ
R2	タイムアウト値(上位 16bit)
A0	対象 short データキューID 番号

■ エラーコード

E_RLWAI	待ち状態強制解除
E_TMOUT	ポーリング失敗、またはタイムアウト

■ 機能説明

vdtqid で示された short データキューから、データを受信し、p_data の指す領域に格納します。対象 short データキューにデータが存在する場合は、その先頭の(最古の)データを受信します。この結果、short データキュー領域に空きが発生するため、送信待ち行列につながれているタスクは、その送信待ち状態が解除され、short データキュー領域へのデータを送信します。

short データキューにデータが存在せず、データ送信待ちタスクが存在する場合(short データキュー領域の容量が 0 の場合)、データ送信待ち行列先頭タスクのデータを受信します。この結果、そのデータ送信待ちタスクの待ち状態は解除されます。

一方、short データキュー領域にデータが格納されていない short データキューに対して、vrcv_dtq,vtrcv_dtq を発行した場合、これらのサービスコールを発行したタスクは、実行状態からデータ受信待ち状態に移行し、データ受信待ち行列につながれます。このとき、受信待ち行列へは、FIFO 順でつながれます。vprcv_dtq,viprcv_dtq の場合は、直ちにリターンし、エラーE_TMOUT を返します。

vtrcv_dtq サービスコールの場合は、tmout には、待ち時間を ms 単位で指定します。tmout に指定可能な値は、(0x7FFFFFFF-タイムティック)以内でなければいけません。これより大きな値を指定した場合は、正しく動作しません。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_dtq 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、vrcv_dtq サービスコールと同じ動作をします。

vrcv_dtq,vtrcv_dtq サービス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。

- **tmout の時間が経過する前に、vsnd_dtq,vtsnd_dtq,vpsnd_dtq,vipsnd_dtq サービスコールが発行され、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 経過し、最初のタイムティックが発生した場合**
  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 サービス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。

タスクコンテキストにおいては、vrcv_dtq,vtrcv_dtq,vprcv_dtq サービスコール、非タスクコンテキストにおいては、viprcv_dtq を使用してください。

■ 使用例

《 C言語の使用例 》

```
  #include <itron.h>
  #include <kernel.h>
  #include "kernel_id.h"
  void task()
  {
    H data;
                :
      if( vrcv_dtq( ID_dtq, &data ) != E_RLWAI )
          error("forced wakeup¥n");
                :
      if( vprcv_dtq( ID_dtq, &data ) != E_TMOUT )
          error("Timeout¥n");
                :
      if( vtrcv_dtq( ID_dtq, &data, 10 ) != E_TMOUT )
          error("Timeout¥n");
                :
  }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0,R3
    vtrcv_dtq   #ID_DTQ1,#TMO_POL
                  :
    PUSHM       A0
    vprcv_dtq   #ID_DTQ2
          :
    PUSHM       A0
    vrcv_dtq    #ID_DTQ2
          :
```

vref_dtq　　short データキューの状態参照
viref_dtq　　short データキューの状態参照（ハンドラ専用）

■ C 言語 API

```
ER ercd = vref_dtq( ID vdtqid, T_RDTQ *pk_rdtq );
ER ercd = viref_dtq( ID vdtqid, T_RDTQ *pk_rdtq );
```

● パラメータ

ID	vdtqid	対象 short データキューID 番号
T_RDTQ	*pk_rdtq	short データキュー状態を返すパケットへのポインタ

● リターンパラメータ

ER	ercd	正常終了(E_OK)またはエラーコード
T_RDTQ	*pk_rdtq	short データキュー状態を返すパケットへのポインタ

pk_rdtq の内容

```
typedef   struct t_rdtq{
    ID      stskid   +0   2    送信待ちタスク ID
    ID      wtskid   +2   2    受信待ちタスク ID
    UINT    sdtqcnt  +4   2    short データキューに入っているデータ数
} T_RDTQ;
```

■ アセンブリ言語 API

```
.include mr308.inc
vref_dtq  VDTQID, PK_RDTQ
viref_dtq VDTQID, PK_RDTQ
```

● パラメータ

VDTQID	対象 short データキューID 番号
PK_RDTQ	short データキュー状態を返すパケットへのポインタ

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象 short データキューID 番号
A1	short データキュー状態を返すパケットへのポインタ

■エラーコード

なし

■ 機能説明

vdtqid で示された short データキューの各種の状態を返します。

- **stskid**
  stskid には送信待ち行列の先頭タスク(次に待ち行列から削除されるタスク)の ID 番号を返します。待ちタスクの無い場合は TSK_NONE を返します。
- **wtskid**
  wtskid には受信待ち行列の先頭タスク(次に待ち行列から削除されるタスク)の ID 番号を返します。待ちタスクの無い場合は TSK_NONE を返します。
- **sdtqcnt**
  sdtqcnt には、short データキュー領域に格納されているデータ個数を返します。

本サービスコールは、タスクコンテキストからは、ref_dtq、非タスクコンテキストからは、iref_dtq を使用してください。

■ 使用例

《 c 言語の使用例 》

```
 #include <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task()
 {
     T_RDTQ rdtq;
     ER ercd;
        :
     ercd = vref_dtq( ID_DTQ1, &rdtq );
        :
 }
```

《 アセンブリ言語の使用例 》

```
_ refdtq:       .blkb   6
    .include mr308.inc
    .GLB        task
task:
        :
       PUSHM   A0,A1
       vref_dtq        #ID_DTQ1,#_refdtq
        :
```

1.16. 拡張機能(リセット機能)

本機能は、オブジェクトの内容を初期化する機能です。本機能は、μITRON 4.0 仕様の仕様外の機能です。

vrst_dtq　　データキュー領域のクリア

■ C 言語 API

```
ER ercd = vrst_dtq( ID dtqid );
```

● パラメータ

ID	dtqid	対象データキューID 番号

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

```
.include mr308.inc
vrst_dtq  DTQID
```

● パラメータ

DTQID	対象データキューID 番号

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象データキューID 番号

■ エラーコード

なし

■ 機能説明

dtqid で示されたデータキューに格納されているデータをクリアします。データキュー領域にさらに追加するエリアがなく、データ送信待ち行列にタスクがつながれている場合、データ送信待ち行列につながれているすべてのタスクの待ち状態を解除します。さらに解除されたタスクに対してエラーEV_RST が返されます。
また、データキューの定義個数がゼロ個の場合においても、データ送信待ち行列につながれているすべてのタスクの待ち状態を解除します。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用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 <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task1(void)
 {
       :
     vrst_dtq( ID_dtq1 );
       :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    vrst_dtq    #ID_DTQ1
          :
```

vrst_vdtq　　short データキュー領域のクリア

■ C 言語 API

```
ER ercd = vrst_vdtq( ID vdtqid );
```

● パラメータ

ID	vdtqid	対象 short データキューID

● リターンパラメータ

ER	Ercd	正常終了(E_OK)

■ アセンブリ言語 API

```
.include mr308.inc
vrst_vdtq VDTQID
```

● パラメータ

VDTQID	対象 short データキューID

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象 short データキューID

■ エラーコード

なし

■ 機能説明

vdtqid で示された short データキューに格納されているデータをクリアします。short データキュー領域にさらに追加するエリアがなく、データ送信待ち行列にタスクがつながれている場合、データ送信待ち行列につながれているすべてのタスクの待ち状態を解除します。さらに解除されたタスクに対してエラーEV_RST が返されます。また、short データキューの定義個数がゼロ個の場合においても、データ送信待ち行列につながれているすべてのタスクの待ち状態を解除します。

本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用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 <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task1(void)
 {
       :
     vrst_vdtq( ID_vdtq1 );
       :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    vrst_vdtq   #ID_VDTQ1
          :
```

vrst_mbx　　メールボックス領域のクリア

■ C 言語 API

```
ER ercd = vrst_mbx( ID mbxid );
```

● パラメータ

ID	mbxid	対象メールボックス ID 番号

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

```
.include mr308.inc
vrst_mbx  MBXID
```

● パラメータ

MBXID	対象メールボックス ID 番号

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象メールボックス ID 番号

■ エラーコード

なし

■ 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。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用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 <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task1(void)
 {
       :
     vrst_mbx( ID_mbx1 );
       :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    vrst_mbx    #ID_MBX1
          :
```

vrst_mpf　　固定長メモリプール領域のクリア

■ C 言語 API

```
ER ercd = vrst_mpf( ID mpfid );
```

● パラメータ

ID	mpfid	対象固定長メモリプール ID 番号

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

```
.include mr308.inc
vrst_mpf  MPFID
```

● パラメータ

MPFID	対象固定長メモリプール ID 番号

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象固定長メモリプール ID 番号

■ エラーコード

なし

■ 機能説明

mpfid で示された固定長メモリプール初期状態にします。メモリブロック獲得待ち行列にタスクがつながれている場合、メモリブロック獲得待ち行列につながれているすべてのタスクの待ち状態を解除します。さらに解除されたタスクに対してエラーEV_RST が返されます。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用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 <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task1(void)
 {
       :
     vrst_mpf( ID_mpf1 );
       :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    vrst_mpf    #ID_MPF1
          :
```

vrst_mpl　　可変長メモリプール領域のクリア

■ C 言語 API

```
ER ercd = vrst_mpl( ID mplid );
```

● パラメータ

ID	mplid	対象可変長メモリプール ID 番号

● リターンパラメータ

ER	ercd	正常終了(E_OK)

■ アセンブリ言語 API

```
.include mr308.inc
vrst_mpl  MPLID
```

● パラメータ

MPLID	対象可変長メモリプール ID 番号

● サービスコール発行後のレジスタ内容

レジスタ名	サービスコール発行後の内容
R0	エラーコード
A0	対象可変長メモリプール ID 番号

■ エラーコード

なし

■ 機能説明

mplid で示された可変長メモリプールを初期状態にします。
本サービスコールは、タスクコンテキストにおいてのみ使用可能です。非タスクコンテキストにおいて使用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 <itron.h>
 #include <kernel.h>
 #include "kernel_id.h"
 void task1(void)
 {
       :
     vrst_mpl( ID_mpl1 );
       :
 }
```

《 アセンブリ言語の使用例 》

```
    .include mr308.inc
    .GLB        task
task:
          :
    PUSHM       A0
    vrst_mpl    #ID_MPL1
          :
```

第2章 スタック算出方法

2.1. スタックサイズの算出方法

MR308 のスタックには、システムスタックとユーザースタックの 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。

●ユーザースタック

タスクに存在するスタックです。従って、MR308 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

MR308 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。MR308 では、サービス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。

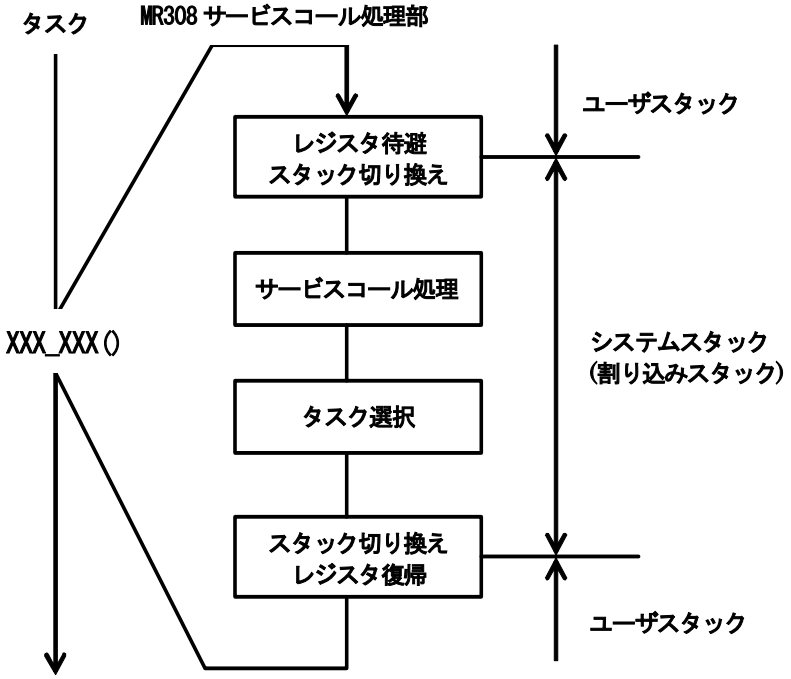


図 2.1:システムスタックとユーザースタック

システムスタックとユーザー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。ただし、以下の図は、コンフィギュレーション時にすべてのタスクのスタック領域を stack セクションに配置した場合です。

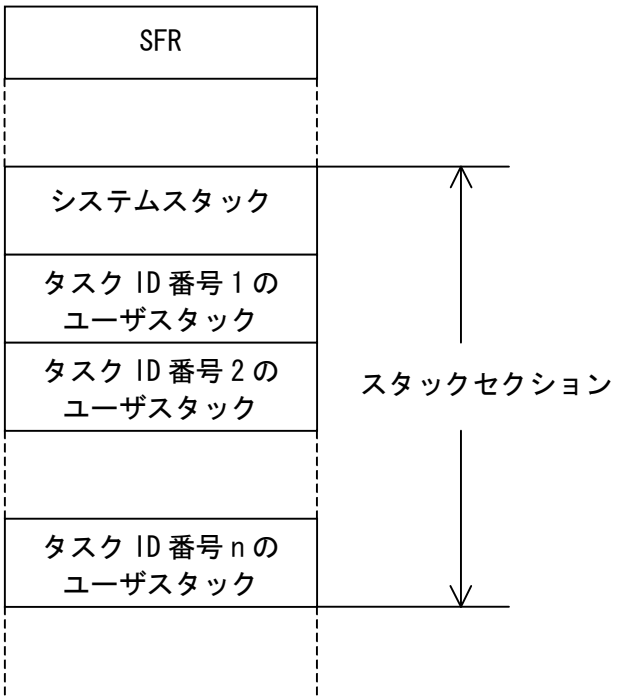


図 2.2:スタックの配置

2.1.1. ユーザースタックの算出方法

ユーザースタックは、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

STK ビューワ[7]をご使用下さい。
STK ビューワは各タスクが使用するスタックサイズを表示します。その表示された各タスクのスタックサイズとコンテキスト格納領域 30 バイト[8]の合計が、タスクのスタックサイズとなります。
STK ビューワの詳細な使用方法については、STK ビューワのマニュアルをご覧ください。

●アセンブリ言語でアプリケーションを記述した場合

- **ユーザープログラムで使用する部分**
  そのタスクがサブルーチン呼び出しで使用するスタック量、および、その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量などの合計。
- **MR308 で使用する部分**
  サービスコールを発行することで消費するスタックサイズです。
  MR308 では、タスクから発行可能なサービスコールのみを発行した場合は、PC+FLG レジスタを格納する領域 6 バイトを確保してください。また、タスクまたはハンドラの両方から発行できるサービスコールを発行した場合は、表 2.3に記載されたスタックサイズを参考に確保して下さい。
  複数のサービスコールを発行している場合は、それらのサービスコールが消費するスタックサイズの最大値を確保して下さい。

よって、
ユーザースタックサイズ =
ユーザープログラムで使用する部分 + 使用するレジスタ分 + MR308 で使用する部分
になります。(使用するレジスタ分は、R0,R1,R2,R3 の場合は各 2byte、A0,A1,SB,FB の場合は各 4byte で使用分を加算する)
図 2.3にユーザースタックの算出例を示します。以下の例では、対象とするタスクが、R0,R1,A0 レジスタを使用している場合です。

[7] STK ビューワは、NC308WA に付属されているスタックを計算するためのツールです。
[8] C 言語で記述した場合、このサイズは固定となります。

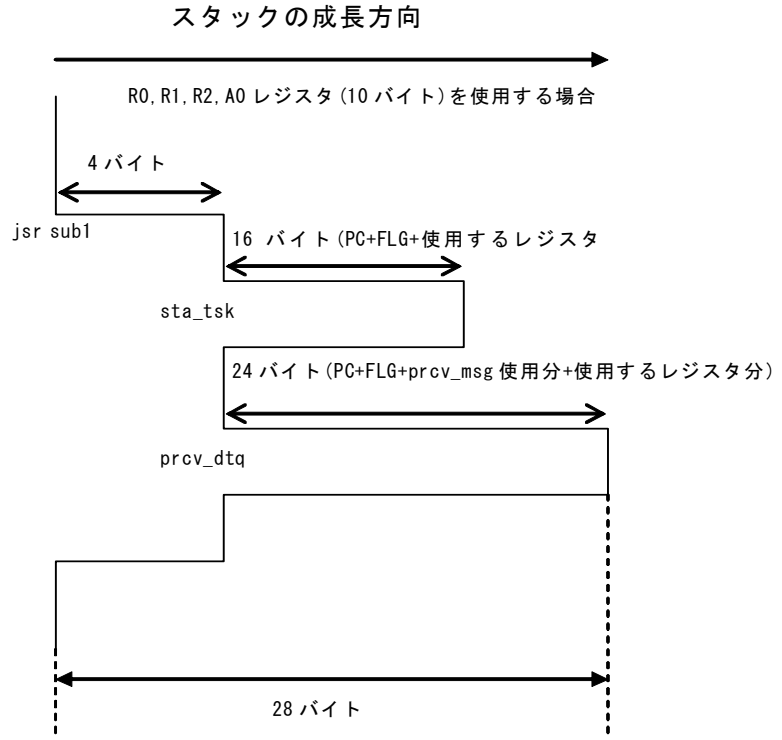


図 2.3:ユーザースタックサイズの算出例

2.1.2. 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サービスコール処理中[9]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。

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

- α

  使用するサービスコールの中で最大のシステムスタックサイズ[10]。

  例えば、sta_tsk、ext_tsk、slp_tsk、dly_tsk を使用する場合、表 2.1で、それぞれのシステムスタックサイズを調べると、

サービスコール名	システムスタックサイズ
sta_tsk	4 バイト
ext_tsk	0 バイト
slp_tsk	4 バイト
dly_tsk	8 バイト

  となるのでこの場合、使用するサービスコールの中で最大のシステムスタックサイズは dly_tsk の場合で 8 バイトです。

- βi

  割り込みハンドラ[11]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。

[9] ユーザースタックからシステムスタックに切り替えた後

[10] それぞれのサービスコールに使用するスタックサイズは、表 2.1から表 2.3を参照してください。

[11] システムクロック割り込みハンドラを含まないカーネル管理割り込みハンドラ(OS 依存割り込みハンドラ)

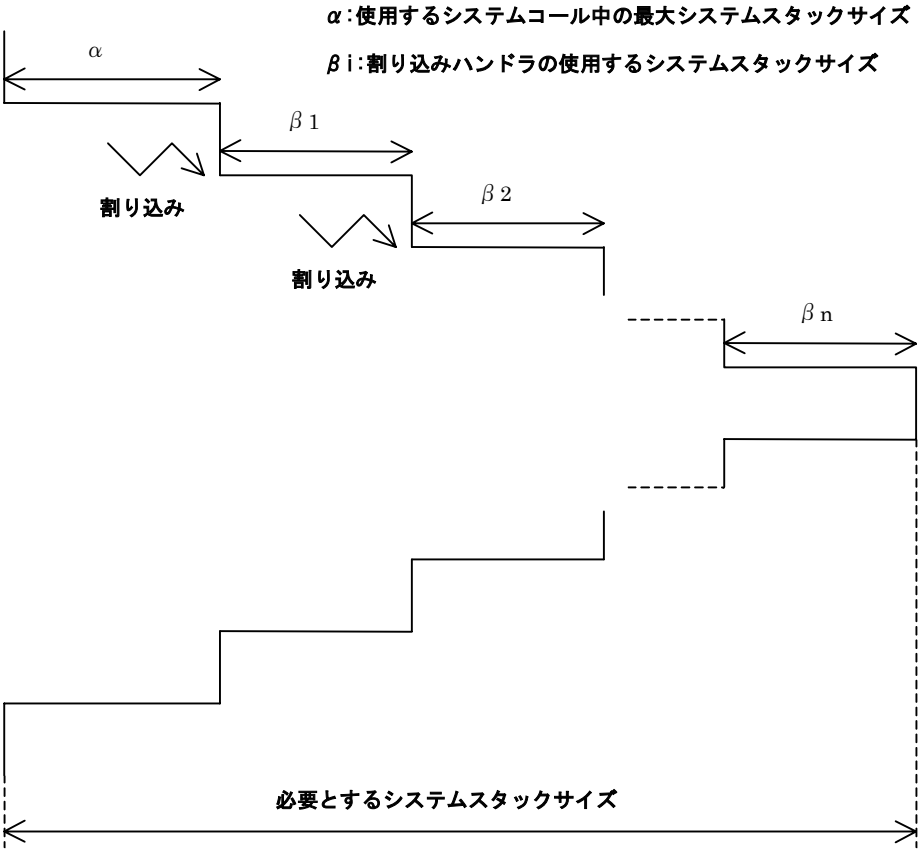


図 2.4:システムスタックサイズの算出方法

【割り込みハンドラの使用するスタックサイズβi】

サービス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計算式で算出できます。
割り込みハンドラの使用するスタックサイズβｉを、以下に示します。

C 言語

STK ビューワ[12]をご使用下さい。

STK ビューワは各割り込みハンドラが使用するスタックサイズを表示します。その表示された値が各割り込みハンドラの使用するスタックサイズになります。

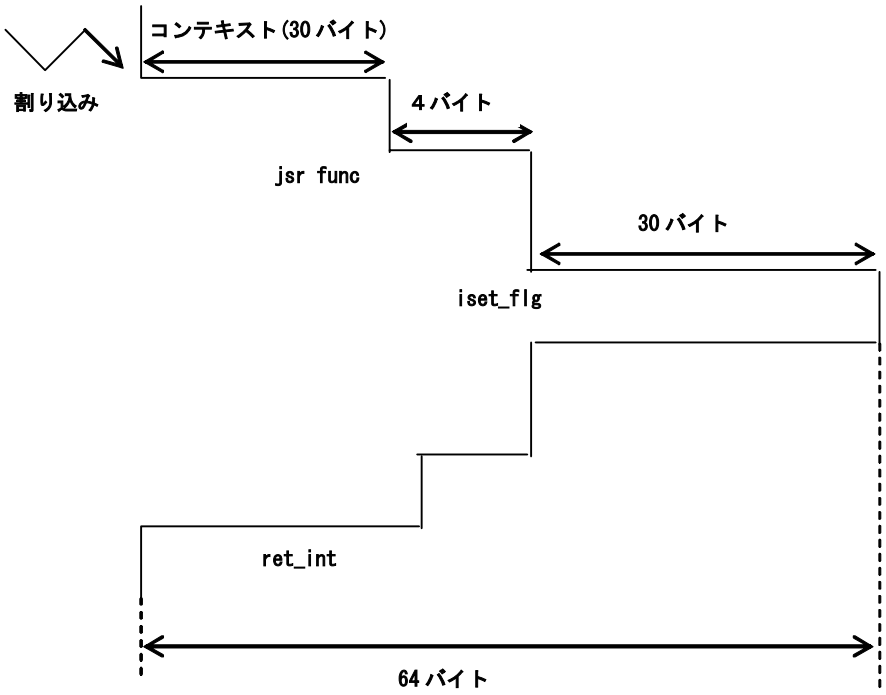
STK ビューワの詳細な使用方法については、STK ビューワのマニュアルをご覧ください。

アセンブリ言語

OS 依存割り込みハンドラの使用するスタックサイズ =
使用するレジスタ分 + ユーザー使用量 + サービスコールの使用量

OS 独立割り込みハンドラの使用するスタックサイズ =
使用するレジスタ分 + ユーザー使用量

ユーザー使用量は、ユーザーの記述する部分で使用するスタック使用量です。



コンテキスト: C 言語で記述した場合は 30 バイト
アセンブリ言語で記述した場合は、使用レジスタ分+6 (PC+FLG)バイト

図 2.5:割り込みハンドラの使用するスタック量

[12] STK ビューワは、三菱製 C コンパイラ NC308WA に付属されているスタックを計算するためのツールです。

【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】

システムタイマ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 2 つの場合のうちの大きいサイズです。

42 + 周期起動ハンドラのスタック使用量の最も大きいサイズ
42+ アラームハンドラのスタック使用量の最も大きいサイズ
ック使用量の最も大きいサイズ

周期起動ハンドラおよびアラームハンドラが使用するスタックサイズの算出方法を以下に示します。

C 言語

STK ビューワ[13]をご使用下さい。

STK ビューワは各ハンドラが使用するスタックサイズを表示します。その表示された値が各ハンドラの使用するスタックサイズになります。

STK ビューワの詳細な使用方法については、STK ビューワのマニュアルをご覧ください。

アセンブリ言語

周期起動ハンドラあるいはアラームハンドラの使用するスタックサイズ =
使用するレジスタ分 + ユーザー使用量 + サービスコールの使用量

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 = 30 バイト

になります。

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。

[13] STK ビューワは、NC308WA に付属されているスタックを計算するためのツールです。

2.2. 各サービスコールのスタック使用量

表 2.1は、タスクコンテキストから発行可能なサービスコールのスタック使用量(ユーザースタック及び、システムスタック)を示しています。

表 2.1　タスクコンテキストから発行するサービスコールのスタック使用量一覧(単位:バイト)

サービスコール	スタックサイズ		サービスコール	スタックサイズ	
	ユーザスタック	システムスタック		ユーザスタック	システムスタック
act_tsk	0	4	rcv_mbx	0(8)	34
can_act	14	0	prcv_mbx	26(8)	0
sta_tsk	0	4	trcv_mbx	0(8)	38
ext_tsk	0	4	ref_mbx	14	0
ter_tsk	0	8	get_mpf	0(8)	34
chg_pri	0	36	pget_mpf	26(8)	0
get_pri	14(8)	0	tget_mpf	0(8)	38
ref_tsk	30	0	rel_mpf	0	8
ref_tst	14	0	ref_mpf	14	0
slp_tsk	0	4	pget_mpl	0(8)	40
tslp_tsk	0	8	rel_mpl	0	76
wup_tsk	0	4	ref_mpl	18	0
can_wup	14	0	set_tim	14	0
rel_wai	0	8	get_tim	14	0
sus_tsk	0	4	sta_cyc	14	0
rsm_tsk	0	4	stp_cyc	14	0
frsm_tsk	0	4	ref_cyc	14	0
dly_tsk	0	8	sta_alm	14	0
sig_sem	0	4	stp_alm	14	0
wai_sem	0	34	ref_alm	14	0
pol_sem	14	0	rot_rdq	0	0
twai_sem	0	36	get_tid	14(8)	0
ref_sem	14	0	loc_cpu	6	0
set_flg	0	8	unl_cpu	0	0
clr_flg	14	0	ref_ver	18	0
wai_flg	0(6)	34	vsnd_dtq	0	34
pol_flg	14(6)	0	vpsnd_dtq	0	8
twai_flg	0(10)	34	vtsnd_dtq	0(8)	38
ref_flg	14	0	vfsnd_dtq	0	8
snd_dtq	0	34	vrcv_dtq	0(6)	8
psnd_dtq	0	8	vprcv_dtq	0(6)	8
tsnd_dtq	0(8)	38	vtrcv_dtq	0(6)	8
fsnd_dtq	0	8	vref_dtq	14	0
rcv_dtq	0(8)	8	vrst_dtq	0	30
prcv_dtq	0(8)	8	vrst_vdtq	0	30
trcv_dtq	0(8)	8	vrst_mbx	14	0
ref_dtq	14	0	vrst_mpf	0	30
snd_mbx	0	28	vrst_mpl	58	0
dis_dsp	0	0	ena_dsp	0	0

()内: C 言語で使用時に必要となるスタック使用量。

表 2.2は、非タスクコンテキストから発行可能なサービス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 2.2　非タスクコンテキストから発行するサービスコールのスタック使用量一覧(単位:バイト)

サービスコール	スタックサイズ	サービスコール	スタックサイズ
iact_tsk	20	iprcv_mbx	26(34)
ican_act	14	iref_mbx	14
ista_tsk	20	ipget_mpf	26(34)
ichg_pri	50	irel_mpf	28
iget_pri	14(22)	iref_mpf	14
iref_tsk	30	iset_tim	14
iref_tst	14	iget_tim	14
iwup_tsk	22	ista_cyc	14
ican_wup	14	istp_cyc	14
irel_wai	22	iref_cyc	14
isus_tsk	18	ista_alm	14
irsm_tsk	18	istp_alm	14
ifrsm_tsk	18	iref_alm	14
isig_sem	26	irot_rdq	18
ipol_sem	14	iget_tid	14(22)
iref_sem	14	iloc_cpu	6
iset_flg	30	iunl_cpu	14
iclr_flg	14	ret_int	0
ipol_flg	14(20)	iref_ver	18
iref_flg	14	vipsnd_dtq	28
ipsnd_dtq	28	vifsnd_dtq	28
ifsnd_dtq	28	viprcv_dtq	28(34)
iprcv_dtq	30(34)	viref_dtq	14
iref_dtq	14	isnd_mbx	48
iref_mpl	18		

()内: C 言語で使用時に必要となるスタック使用量。

表 2.3は、タスクコンテキストあるいは非タスクコンテキストの両方から発行可能なサービス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、タスクからサービスコールを発行した場合は、ユーザースタックを使用し、非タスクコンテキストから発行した場合は、システムスタックを使用します。

表 2.3　両方から発行可能なサービスコールのスタック使用量一覧

サービスコール	スタックサイズ	サービスコール	スタックサイズ
sns_ctx	14	sns_loc	14
sns_dsp	14	sns_dpn	14

*: C 言語で使用時に必要となるスタック使用量。

第3章 付録

3.1. サービスコール一覧

タスク管理機能サービスコール

サービスコール名		機能
act_tsk	[S]	タスクを起動します
iact_tsk	[S]	タスクを起動します
can_act	[S]	タスク起動要求をクリアします
ican_act	[S]	タスク起動要求をクリアします(ハンドラ専用)
sta_tsk		タスクを起動します(起動コード指定)
ista_tsk		タスクを起動します(起動コード指定,ハンドラ専用)
ext_tsk	[S]	自タスクを正常終了します
ter_tsk	[S]	他タスクを強制的に異常終了します
chg_pri	[S]	タスクの優先度を変更します
ichg_pri		タスクの優先度を変更します(ハンドラ専用)
get_pri	[S]	優先度を取得します
iget_pri		優先度を取得します(ハンドラ専用)
ref_tsk		タスクの状態を参照します
iref_tsk		タスクの状態を参照します(ハンドラ専用)
ref_tst		タスク状態を参照します(簡易版)
iref_tst		タスク状態を参照します(簡易版,ハンドラ専用)

タスク付属同期機能サービスコール

サービスコール名		機能
slp_tsk	[S]	タスクを待ち状態へ移行する
tslp_tsk	[S]	タスクを一定時間待ち状態へ移行する
wup_tsk	[S]	待ち状態のタスクを起床します
iwup_tsk	[S]	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
ican_wup		タスクの起床要求を無効にします(ハンドラ専用)
rel_wai	[S]	待ち状態を強制解除します
irel_wai	[S]	待ち状態を強制解除します(ハンドラ専用)
sus_tsk	[S]	タスクを強制待ち状態を移行します
isus_tsk		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します
irsm_tsk		強制待ち状態のタスクを再開します(ハンドラ専用)
frsm_tsk	[S]	強制待ち状態のタスクを強制再開します
ifrsm_tsk		強制待ち状態のタスクを強制再開します(ハンドラ専用)
dly_tsk	[S]	タスクの実行を一定時間遅延します。

セマフォ

サービスコール名		機能
sig_sem	[S]	セマフォ資源を返却します
isig_sem	[S]	セマフォ資源を返却します(ハンドラ専用)
wai_sem	[S]	セマフォ資源を獲得します
twai_sem	[S]	セマフォ資源を獲得します(タイムアウトあり)
pol_sem	[S]	セマフォ資源を獲得します(ポーリング)
ipol_sem		セマフォ資源を獲得します(ポーリング,ハンドラ専用)
ref_sem		セマフォの状態を参照します
iref_sem		セマフォの状態を参照します(ハンドラ専用)

イベントフラグ

サービスコール名		機能
set_flg	[S]	イベントフラグをセットします
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
iclr_flg		イベントフラグをクリアします(ハンドラ専用)
wai_flg	[S]	イベントフラグを待ちます
twai_flg	[S]	イベントフラグを待ちます(タイムアウトあり)
pol_flg	[S]	イベントフラグを得ます(ポーリング)
ipol_flg		イベントフラグを得ます(ポーリング, ハンドラ専用)
ref_flg		イベントフラグの状態を参照します
iref_flg		イベントフラグの状態を参照します(ハンドラ専用)

データキュー

サービスコール名		機能
snd_dtq	[S]	データキューにデータを送信します
psnd_dtq	[S]	データキューにデータを送信します(ポーリング)
ipsnd_dtq	[S]	データキューにデータを送信します(ポーリング, ハンドラ専用)
tsnd_dtq	[S]	データキューにデータを送信します(タイムアウトあり)
fsnd_dtq	[S]	データキューにデータを強制送信します
ifsnd_dtq	[S]	データキューにデータを強制送信します(ハンドラ専用)
rcv_dtq	[S]	データキューからデータを受信します
prcv_dtq	[S]	データキューからデータを受信します(ポーリング)
iprcv_dtq		データキューからデータを受信します(ポーリング, ハンドラ専用)
trcv_dtq	[S]	データキューからデータを受信します(タイムアウトあり)
ref_dtq		データキューの状態を参照します
iref_dtq		データキューの状態を参照します(ハンドラ専用)

メールボックス

サービスコール名		機能
snd_mbx	[S]	メッセージを送信します
isnd_mbx		メッセージを送信します(ハンドラ専用)
rcv_ mbx	[S]	メッセージを受信します
trcv_ mbx	[S]	メッセージを受信します(タイムアウトあり)
prcv_ mbx	[S]	メッセージを受信します(ポーリング)
iprcv_mbx		メッセージを受信します(ポーリング, ハンドラ専用)
ref_mbx		メールボックスの状態を参照します
iref_mbx		メールボックスの状態を参照します(ハンドラ専用)

固定長メモリプール

サービスコール名		機能
get_mpf	[S]	メモリブロックを獲得します
tget_mpf	[S]	メモリブロックを獲得します(タイムアウトあり)
pget_mpf	[S]	メモリブロックを獲得します(ポーリング)
ipget_mpf		メモリブロックを獲得します(ポーリング, ハンドラ専用)
rel_mpf	[S]	メモリブロックを解放します
irel_mpf		メモリブロックを解放します(ハンドラ専用)
ref_mpf		固定長メモリプールの状態を参照します
ief_mpf		固定長メモリプールの状態を参照します(ハンドラ専用)

可変長メモリプール

サービスコール名	機能
pget_npl	メモリブロックを獲得します(ポーリング)
rel_mpl	メモリブロックを解放します
ref_mpl	可変長メモリプールの状態を参照します
iref_mpl	可変長メモリプールの状態を参照します(ハンドラ専用)

時間管理機能

サービスコール名		機能
set_tim	[S]	システム時刻を設定します
iset_tim		システム時刻を設定します(ハンドラ専用)
get_tim	[S]	システム時刻を参照します
iget_tim		システム時刻を参照します(ハンドラ専用)
isig_tim	[S]	タイムティックを供給します(ハンドラ専用)
		(本機能は、カーネルに組み込まれています)

周期ハンドラ

サービスコール名		機能
sta_cyc	[S]	周期ハンドラの動作を開始します
ista_cyc		周期ハンドラの動作を開始します(ハンドラ専用)
stp_cyc	[S]	周期ハンドラの動作を停止します
stp_cyc		周期ハンドラの動作を停止します(ハンドラ専用)
ref_cyc		周期ハンドラの状態を参照します
iref_cyc		周期ハンドラの状態を参照します(ハンドラ専用)

アラームハンドラ

サービスコール名	機能
sta_ alm	アラームハンドラの動作を開始します
ista_ alm	アラームハンドラの動作を開始します(ハンドラ専用)
stp_ alm	アラームハンドラの動作を停止します
stp_ alm	アラームハンドラの動作を停止します(ハンドラ専用)
ref_ alm	アラームハンドラの状態を参照します
iref_ alm	アラームハンドラの状態を参照します(ハンドラ専用)

システム状態管理機能

サービスコール名		機能
rot_rdq	[S]	タスク優先順位を回転させます
irot_rdq	[S]	タスク優先順位を回転させます(ハンドラ専用)
get_tid	[S]	タスク ID 番号を参照します
iget_tid	[S]	タスク ID 番号を参照します(ハンドラ専用)
loc_cpu	[S]	CPU ロック状態に移行させます
iloc_cpu	[S]	CPU ロック状態に移行させます(ハンドラ専用)
unl_cpu	[S]	CPU ロック状態を解除します
iunl_cpu	[S]	CPU ロック状態を解除します(ハンドラ専用)
dis_dsp	[S]	ディスパッチ禁止状態に移行させます
ena_dsp	[S]	ディスパッチ禁止状態を解除します
sns_ctx	[S]	コンテキスト状態を参照します
sns_loc	[S]	CPU ロック状態を参照します
sns_dsp	[S]	ディスパッチ禁止状態を参照します
sns_dpn	[S]	ディスパッチ保留状態を参照します

割り込み管理機能

サービスコール名	機能
ret_int	割り込みハンドラから復帰します(アセンブリ言語専用)

システム構成管理機能

サービスコール名	機能
ref_ver	バージョン情報を参照します
iref_ver	バージョン情報を参照します(ハンドラ専用)

short データキュー

サービスコール名	機能
vsnd_dtq	short データキューにデータを送信します
vpsnd_dtq	short データキューにデータを送信します(ポーリング)
vipsnd_dtq	short データキューにデータを送信します(ポーリング，ハンドラ専用)
vtsnd_dtq	short データキューにデータを送信します(タイムアウトあり)
vfsnd_dtq	short データキューにデータを強制送信します
vifsnd_dtq	short データキューにデータを強制送信します(ハンドラ専用)
vrcv_dtq	short データキューからデータを受信します
vprcv_dtq	short データキューからデータを受信します(ポーリング)
viprcv_dtq	short データキューからデータを受信します(ポーリング，ハンドラ専用)
vtrcv_dtq	short データキューからデータを受信します(タイムアウトあり)
vref_dtq	short データキューの状態を参照します
viref_dtq	short データキューの状態を参照します(ハンドラ専用)

拡張機能

サービスコール名	機能
vrst_dtq	データキューを初期化します
vrst_vdtq	short データキューを初期化します
vrst_mbx	メールボックスを初期化します
vrst_mpf	固定長メモリプールを初期化します
vrst_mpl	可変長メモリプールを初期化します

3.2. エラーコード一覧

エラーコード	値	説明
E_OK	0	正常終了
E_ILUSE	-28	サービスコール不正使用
E_OBJ	-41	オブジェクトの状態が不正
E_QOVR	-43	キューイングまたはネストのオーバーフロー
E_TMOUT	-50	ポーリング失敗またはタイムアウト
E_RLWAI	-49	待ち状態強制解除
EV_RST	-254	リセットのため待ちが解除された

3.3. データタイプ

```
typedef    signed char       B;          /* 符号付き 8 ビット整数 */
typedef    signed short      H;          /* 符号付き 16 ビット整数 */
typedef    signed 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                 FN          /* 機能コード */
typedef    H                 ID;         /* オブジェクト ID 番号 */
typedef    H                 PRI;        /* タスク優先度 */
typedef    W                 TMO;        /* タイムアウト */
typedef    W                 ER;         /* エラーコード(符号付き整数) */
typedef    UW                ATR;        /* オブジェクト属性(符号なし整数) */
typedef    UW                STAT;       /* タスクの状態 */
typedef    UW                MODE;       /* サービスコールの動作モード */
typedef    UW                SIZE;       /* メモリ領域のサイズ */
typedef    UW                RELTIM      /* 相対時間 */
typedef    W                 VP_INT;     /* データタイプが定まらないものへのポインタまた
                                            はプロセッサに自然なサイズの符号付き整数 */
typedef    struct            systim{     /* システム時刻 */
           UH                utime;      /* 時刻の上位 16bit */
           UW                ltimer;     /* 時刻の下位 32bit */
} SYSTIM;
typedef    W                 ER_ID;      /* エラーコードまたは ID */
typedef    W                 ER_UINT;    /* エラーコードまたは符号無し整数 */
```

3.4. 共通定数と構造体のパケット形式

```
----共通----
TRUE         1             /* 真 */
FALSE        0             /* 偽 */
----タスク管理関係----
TSK_SELF     0            /* 自タスク指定 */
TPRI_RUNNING     0            /* その時実行中の優先度を指定 */
typedef struct t_rtsk {
        STAT     tskstat; /* タスクの状態 */
        PRI      tskpri;  /* タスクの現在優先度 */
        PRI      tskbpri; /* タスクのベース優先度 */
        STAT     tskWAITING;        /* タスクの待ち要因 */
        ID       wid;     /* タスクの待ちオブジェクト ID */
        TMO      tskatr;  /* タイムアウトするまでの時間 */
        UINT     actcnt;  /* 起動要求キューイング数 */
        UINT     wupcnt;  /* 起床要求キューイング数 */
        UINT     suscnt;  /* 強制待ち要求ネスト数 */
} T_RTSK;
typedef struct t_rtst {
        STAT     tskstat;           /* タスクの状態 */
        STAT     tskWAITING;                /* タスクの待ち要因 */
} T_RTST;
----セマフォ関係----
typedef struct t_rsem {
        ID       wtskid;  /* 待ち行列先頭タスクの ID 番号 */
        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----
wfmod:
        TWF_ANDW   H'0000    /* AND 待ち */
        TWF_ORW    H'0002    /* OR 待ち */
typedef struct t_rflg {
        ID       wtskid;  /* 待ち行列先頭タスクの ID 番号 */
        UINT     flgptn;  /* イベントフラグの現在のビットパターン */
} T_RFLG;
----データキュー、short データキュー関係----
typedef struct t_rdtq {
        ID       stskid;            /* 送信待ち行列先頭タスクの ID 番号 */                    ID
        rtskid;             /* 受信待ち行列先頭タスクの ID 番号 */
        UINT     sdtqcnt;           /* データキューに入っているデータの個数 */
} T_RDTQ;
----メールボックス関係----
typedef struct     t_msg   {
    VP   msghead;                   /* メッセージヘッダ                          */
} T_MSG;
typedef struct t_msg_pri {
    T_MSG                msgque; /* メッセージヘッダ                          */
    PRI          msgpri;            /* メッセージ優先度                          */
} T_MSG_PRI;



typedef struct t_mbx {
        ID       wtskid;            /* 待ち行列先頭タスクの ID 番号*/
        T_MSG    *pk_msg;           /* 次に受信されるメッセージ */
} T_RMBX;
----固定長メモリプール関係----
typedef struct t_rmpf {
```

```
        ID      wtskid;  /* メモリ獲得待ち行列先頭タスクの ID 番号*/
        UINT    frbcnt;  /* メモリブロック数 */
} T_RMPF;

----可変長メモリプール関係----
typedef struct t_rmpl {
        ID      wtskid;  /* メモリ獲得待ち行列先頭タスクの ID 番号*/
        SIZE    fmplsz;  /* 空き領域の合計サイズ */
        UINT    fblksz;  /* すぐに獲得可能な最大メモリブロックサイズ */
} T_RMPL;
----周期ハンドラ関係----
typedef struct  t_rcyc {
        STAT    cycstat; /* 周期ハンドラ動作状態 */
        RELTIM lefttim;  /* 周期ハンドラの起動までの残り時間 */
} T_RCYC;
----アラームハンドラ関係----
typedef struct t_ralm {
        STAT    almstat; /* アラームハンドラ動作状態 */
        RELTIM lefttim;  /* アラームハンドラの起動までの残り時間 */
} T_RALM;
/* システム管理関係 */
typedef struct t_rver {
        UH      maker;   /* メーカー */
        UH      prid;    /* 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 */
} T_RVER;
```

3.5. アセンブリ言語インタフェース

アセンブリ言語でサービスコールを発行する場合、サービスコールの呼び出し用マクロを使用します。

サービス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、サービスコールの呼び出し用マクロを使用せず直接サービス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

ServiceCall	INTNo.	Parameter					ReturnParameter	
		FuncCode R0	R1	R3	A0	A1 FuncCode	R0	A0
ista_tsk	62	10	stacd	stacd	tskid	−	ercd	−
sta_tsk	63	8	stacd	stacd	tskid	−	ercd	−
act_tsk	63	0	−	−	tskid	−	ercd	−
iact_tsk	62	2	−	−	tskid	−	ercd	−
ter_tsk	63	14	−	−	tskid	−	ercd	−
can_act	62	4	−	−	tskid	−	actcnt	−
ican_act	62	6	−	−	tskid	−	actcnt	−
chg_pri	63	16	−	tskpri	tskid	−	ercd	−
ichg_pri	62	18	−	tskpri	tskid	−	ercd	−
rel_wai	63	44	−	−	tskid	−	ercd	−
irel_wai	62	46	−	−	tskid	−	ercd	−
ref_tst	62	28	−	−	tskid	pk_rtst	ercd	−
iref_tst	62	30	−	−	tskid	pk_rtst	ercd	−
ref_tsk	62	24	−	−	tskid	pk_rtsk	ercd	−
iref_tsk	62	26	−	−	tskid	pk_rtsk	ercd	−
ext_tsk	58	106	−	−	−	−	−	−
get_pri	62	20	−	−	tskid	−	ercd	tskpri
iget_pri	62	22	−	−	tskid	−	ercd	tskpri

タスク付属同期

ServiceCall	INTNo.	Parameter					ReturnParameter
		FuncCode R0	R1	R3	A0	A1 FuncCode	R0
slp_tsk	63	32	–	–	–	–	ercd
wup_tsk	63	36	–	–	tskid	–	ercd
iwup_tsk	62	38	–	–	tskid	–	ercd
can_wup	62	40	–	–	tskid	–	wupcnt
ican_wup	62	42	–	–	tskid	–	wupcnt
tslp_tsk	63	34	tmout	tmout	–	–	ercd
sus_tsk	63	48	–	–	tskid	–	ercd
isus_tsk	62	50	–	–	tskid	–	ercd
rsm_tsk	63	52	–	–	tskid	–	ercd
irsm_tsk	62	54	–	–	tskid	–	ercd
frsm_tsk	63	56	–	–	tskid	–	ercd
ifrsm_tsk	62	58	–	–	tskid	–	ercd
dly_tsk	63	60	tmout	tmout	–	–	ercd

同期・通信機能

ServiceCall	INTNo.	Parameter						ReturnParameter			
		FuncCode R0	R1	R2	R3	A0	A1 FuncCode	R0	R1	R2	R3
wai_sem	63	66	–	–	–	semid	–	ercd	–	–	–
pol_sem	62	68	–	–	–	semid	–	ercd	–	–	–
ipol_sem	62	70	–	–	–	semid	–	ercd	–	–	–
sig_sem	63	62	–	–	–	semid	–	ercd	–	–	–
isig_sem	62	64	–	–	–	semid	–	ercd	–	–	–
twai_sem	63	72	tmout	–	tmout	semid	–	ercd	–	–	–
ref_sem	62	74	–	–	–	semid	pk_rsem	ercd	–	–	–
iref_sem	62	76	–	–	–	semid	pk_rsem	ercd	–	–	–
wai_flg	63	86	wfmode	–	waiptn	flgid	–	ercd	–	flgptn	–
twai_flg	55	tmout	wfmode	tmout	waiptn	flgid	92	ercd	–	flgptn	–
pol_flg	62	88	wfmode	–	waiptn	flgid	–	ercd	–	flgptn	–
ipol_flg	62	90	wfmode	–	waiptn	flgid	–	ercd	–	flgptn	–
set_flg	63	78	–	–	setptn	flgid	–	ercd	–	–	–
iset_flg	62	80	–	–	setptn	flgid	–	ercd	–	–	–
ref_flg	62	94	–	–	–	flgid	pk_rflg	ercd	–	–	–
iref_flg	62	96	–	–	–	flgid	pk_rflg	ercd	–	–	–
clr_flg	62	82	–	–	clrptn	flgid	–	ercd	–	–	–
iclr_flg	62	84	–	–	clrptn	flgid	–	ercd	–	–	–
snd_dtq	63	98	data	–	data	dtqid	–	ercd	–	–	–
psnd_dtq	63	100	data	–	data	dtqid	–	ercd	–	–	–
ipsnd_dtq	62	102	data	–	data	dtqid	–	ercd	–	–	–
fsnd_dtq	63	106	data	–	data	dtqid	–	ercd	–	–	–
ifsnd_dtq	62	108	data	–	data	dtqid	–	ercd	–	–	–
tsnd_dtq	55	tmout	data	tmout	data	dtqid	104	ercd	–	–	–

同期・通信機能（つづき）

ServiceCall	INTNo.	Parameter						ReturnParameter			
		FuncCode R0	R1	R2	R3	A0	A1 FuncCode	R0	R1	R2	R3
rcv_dtq	63	110	–	–	–	dtqid	–	ercd	data	–	data
prcv_dtq	63	112	–	–	–	dtqid	–	ercd	data	–	data
iprcv_dtq	62	114	–	–	–	dtqid	–	ercd	data	–	data
trcv_dtq	63	116	tmout	–	tmout	dtqid	–	ercd	data	–	data
ref_dtq	62	118	–	–	–	dtqid	pk_rdtq	ercd	–	–	–
iref_dtq	62	120	–	–	–	dtqid	pk_rdtq	ercd	–	–	–
snd_mbx	63	122	–	–	–	mbxid	pk_msg	ercd	–	–	–
isnd_mbx	62	124	–	–	–	mbxid	pk_msg	ercd	–	–	–
rcv_mbx	63	126	–	–	–	mbxid	–	ercd	pk_msg	pk_msg	–
prcv_mbx	62	128	–	–	–	mbxid	–	ercd	pk_msg	pk_msg	–
iprcv_mbx	62	130	–	–	–	mbxid	–	ercd	pk_msg	pk_msg	–
trcv_mbx	63	132	tmout	–	tmout	mbxid	–	ercd	pk_msg	pk_msg	–
ref_mbx	62	134	–	–	–	mbxid	pk_rmbx	ercd	–	–	–
iref_mbx	62	136	–	–	–	mbxid	pk_rmbx	ercd	–	–	–

システム状態管理機能

ServiceCall	INTNo.	Parameter		ReturnParameter	
		FuncCode R0	R3	R0	A0
rot_rdq	63	190	tskpri	ercd	–
irot_rdq	62	192	tskpri	ercd	–
get_tid	62	194	–	ercd	tskid
iget_tid	62	196	–	ercd	tskid
loc_cpu	59	198	–	ercd	–
iloc_cpu	59	200	–	ercd	–
dis_dsp	60	206	–	ercd	–
ena_dsp	63	208	–	ercd	–
unl_cpu	63	202	–	ercd	–
iunl_cpu	62	204	–	ercd	–
sns_ctx	62	210	–	ercd	–
sns_loc	62	212	–	ercd	–
sns_dsp	62	214	–	ercd	–
sns_dpn	62	216	–	ercd	–

割り込み管理機能

ServiceCall	INTNo.	Parameter	ReturnParameter
		FuncCode R0	R0
ret_int	61	–	–

メモリプール管理機能

ServiceCall	INTNo.	Parameter						ReturnParameter			
		FuncCode R0	R1	R2	R3	A0	A1 FuncCode	R0	R1	R2	R3
get_mpf	63	140	–	–	–	mpfid	–	ercd	p_blk	–	p_blk
pget_mpf	62	138	–	–	–	mpfid	–	ercd	p_blk	–	p_blk
ipget_mpf	62	246	–	–	–	mpfid	–	ercd	p_blk	–	p_blk
tget_mpf	63	142	tmout	–	tmout	mpfid	–	ercd	p_blk	–	p_blk
rel_mpf	63	144	blk	–	blk	mpfid	–	ercd	–	–	–
irel_mpf	62	146	blk	–	blk	mpfid	–	ercd	–	–	–
ref_mpf	62	148	–	–	–	mpfid	pk_rmpf	ercd	–	–	–
iref_mpf	62	150	–	–	–	mpfid	pk_rmpf	ercd	–	–	–
pget_mpl	63	152	blksz	–	–	mplid	–	ercd	p_blk	–	p_blk
rel_mpl	63	154	blk	–	blk	mplid	–	ercd	–	–	–
ref_mpl	62	156	–	–	–	mplid	pk_rmpl	ercd	–	–	–
iref_mpl	62	262	–	–	–	mplid	pk_rmpl	ercd	–	–	–

時間管理機能

ServiceCall	INTNo.	Parameter					ReturnParameter
		FuncCode R0	R1	R3	A0	A1 FuncCode	R0
set_tim	62	158	–	–	p_systim	–	ercd
iset_tim	62	160	–	–	p_systim	–	ercd
get_tim	62	162	–	–	p_systim	–	ercd
iget_tim	62	164	–	–	p_systim	–	ercd
sta_cyc	62	166	–	–	cycid	–	ercd
ista_cyc	62	168	–	–	cycid	–	ercd
stp_cyc	62	170	–	–	cycid	–	ercd
istp_cyc	62	172	–	–	cycid	–	ercd
ref_cyc	62	174	–	–	cycid	pk_rcyc	ercd
iref_cyc	62	176	–	–	cycid	pk_rcyc	ercd
sta_alm	62	178	almtim	almtim	almid	–	ercd
ista_alm	62	180	almtim	almtim	almid	–	ercd
stp_alm	62	182	–	–	almid	–	ercd
istp_alm	62	184	–	–	almid	–	ercd
ref_alm	62	186	–	–	almid	pk_ralm	ercd
iref_alm	62	188	–	–	almid	pk_ralm	ercd

システム構成管理機能

ServiceCall	INTNo.	Parameter		ReturnParameter
		FuncCode R0	A0	R0
ref_ver	62	218	pk_rver	ercd
iref_ver	62	220	pk_rver	ercd

拡張機能(リセット機能)

ServiceCall	INTNo.	Parameter		ReturnParameter
		FuncCode R0	A0	R0
vrst_vdtq	63	256	vdtqid	ercd
vrst_dtq	63	248	dtqid	ercd
vrst_mbx	62	250	mbxid	ercd
vrst_mpf	63	252	mpfid	ercd
vrst_mpl	62	254	mplid	ercd

拡張機能(Short データキュー機能)

ServiceCall	INTNo.	Parameter						ReturnParameter			
		FuncCode R0	R1	R2	R3	A0	A1 FuncCode	R0	R1	R2	R3
vsnd_dtq	63	222	data	-	-	vdtqid	-	ercd	-	-	-
vpsnd_dtq	63	224	data	-	-	vdtqid	-	ercd	-	-	-
vipsnd_dtq	62	226	data	-	-	vdtqid	-	ercd	-	-	-
vfsnd_dtq	63	230	data	-	-	vdtqid	-	ercd	-	-	-
vifsnd_dtq	62	232	data	-	-	vdtqid	-	ercd	-	-	-
vtsnd_dtq	55	tmout	data	tmout		vdtqid	228	ercd	-	-	-
vrcv_dtq	63	234	-	-	-	vdtqid	-	ercd	data	-	-
vprcv_dtq	63	236	-	-	-	vdtqid	-	ercd	data	-	-
viprcv_dtq	62	238	-	-	-	vdtqid	-	ercd	data	-	-
vtrcv_dtq	63	240	tmout	-	tmout	vdtqid	-	ercd	data	-	-
vref_dtq	62	242	-	-	-	vdtqid	pk_rdtq	ercd	-	-	-
viref_dtq	62	244	-	-	-	vdtqid	pk_rdtq	ercd	-	-	-

M16C/70,80,M32C/80 シリーズ用リアルタイム OS
リファレンスマニュアル
M3T-MR308/4

発行年月日　2005 年 11 月 1 日　Rev.2.00

発行　　株式会社 ルネサス テクノロジ　営業企画統括部
〒100-0004　東京都千代田区大手町 2-6-2

編集　　株式会社 ルネサス ソリューションズ　第一応用技術部

M3T-MR308/4 V.4.00
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

RJJ10J0837-0200Z