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。

お客様各位

資料中の「日立製作所」、「日立XX」等名称の株式会社ルネサス テクノロジへの変更について

2003年4月1日を以って三菱電機株式会社及び株式会社日立製作所のマイコン、ロジック、アナログ、ディスクリート半導体、及びDRAMを除くメモリ（フラッシュメモリ・SRAM等）を含む半導体事業は株式会社ルネサス テクノロジに承継されました。従いまして、本資料中には「日立製作所」、「株式会社日立製作所」、「日立半導体」、「日立XX」といった表記が残っておりますが、これらの表記は全て「株式会社ルネサス テクノロジ」に変更されておりますのでご理解の程お願い致します。尚、会社商標・ロゴ・コーポレートステートメント以外の内容については一切変更しておりませんので資料としての内容更新ではありません。

ルネサステクノロジ ホームページ（http://www.renesas.com）

2003年4月1日
株式会社ルネサス テクノロジ
カスタマサポート部

RENESAS

ご注意

安全設計に関するお願い

1. 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

1. 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ルネサス テクノロジ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。
2. 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、ルネサス テクノロジは責任を負いません。
3. 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、ルネサス テクノロジは、予告なしに、本資料に記載した製品また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、事前にルネサス テクノロジ、ルネサス販売また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。
4. 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、ルネサス テクノロジはその責任を負いません。
5. 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。ルネサス テクノロジは、適用可否に対する責任は負いません。
6. 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、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。
7. 本資料の転載、複製については、文書によるルネサス テクノロジの事前の承諾が必要です。
8. 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ルネサス テクノロジ、ルネサス販売または特約店までご照会ください。

User's Manual

RENESAS

HI-SH77

ユーザーズマニュアル

ルネサスマイクロコンピュータ開発環境システム
Industrial Realtime Operating System SH7000 Series

HS0770ITCN1S-1

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　1995.09

はじめに

本マニュアルでは、**μITRON**仕様に準拠した機器組込み用リアルタイム・マルチタスク**OS(Operating System)**、**HI-SH77(Hitachi Industrial Realtime Operating System SH3)**の使用方法について説明します。

HI-SH77の構築方法については、『**HI-SH77**構築マニュアル』を参照してください。

＜マニュアルの構成＞

本マニュアルは、以下に示す4つの章と付録から構成されています。

第**1**章では、**HI-SH77**の概説を記載しています。

第**2**章では、**HI-SH77**の機能を説明しています。システムの機能設計時に利用してください。

第**3**章では、**HI-SH77**のシステムコールの仕様を説明しています。ユーザプログラムの詳細設計、コーディング時に利用してください。

第**4**章では、**C** 言語を用いたユーザプログラムの作成方法について説明しています。

付録では、コンソールドライバとタイマドライバの例題、システムコール一覧表、SH3に関する注意事項、エラーコード一覧表、ヘッダファイル、**ASCII**コード表を記載しています。

HI-SH77をご使用になる前に本マニュアルをよく読んで理解してください。また、下記の関連マニュアルもお読みの上、理解してください。

＜関連マニュアル＞

関連マニュアルは以下のとおりです。

・**HI-SH77**構築マニュアル

・使用する**SH3**マイコンのハードウェアマニュアル

＜本マニュアルで使用する記号などの意味＞

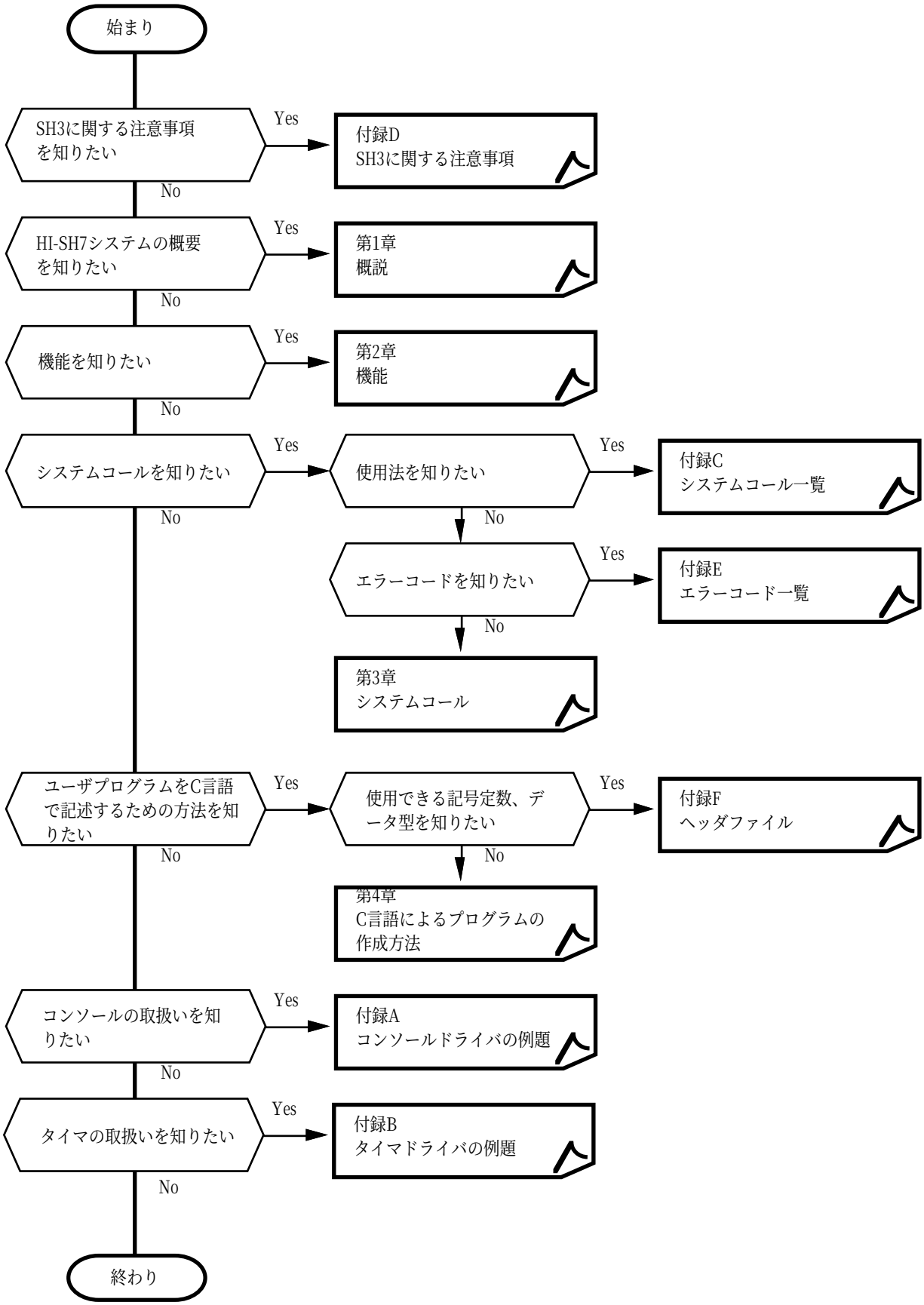
[ ]	省略できることを示します。
{ }	いずれかひとつを選択することを示します。
(RET)	リターンキーの入力を示します。
△	1つ以上の空白またはタブコードを示します。
_	アンダーラインの部分はユーザがキー入力するコマンドラインです。
< >	この記号で囲まれた内容を指定することを示します。
...	直前の項目を1回以上指定することを示します。
H'	整数定数の先頭に"**H'**"が付いているのは16進数です。それ以外は10進数です。

＜注意＞

μITRONは、"**Micro Industrial TRON**"の略称です。

TRONは、"**The Real time Operating system Nucleus**"の略称です。

本マニュアルを読まれる前に、知りたい事項を下記フローからピックアップされることをおすすめします。



目　次

（第１章　概　説）
１．１　概　要 …… 1-1
１．２　特　長 …… 1-1
１．３　構　成 …… 1-2
１．４　システムの構築手順 …… 1-3

（第２章　機　能）
２．１　概　要 …… 2-1
２．２　システム状態 …… 2-3
２．３　タスク …… 2-5
　２．３．１　タスクとカーネルの関係 …… 2-5
　２．３．２　タスクの状態と遷移 …… 2-6
　２．３．３　タスクの生成 …… 2-8
　２．３．４　タスクの起動と実行開始時の初期化 …… 2-8
　２．３．５　共有スタック機能 …… 2-9
　２．３．６　タスクのスケジューリング …… 2-12
　２．３．７　タスクの中断と再開 …… 2-14
　２．３．８　タスクの終了 …… 2-15
　２．３．９　タスク実行中の割込みマスク …… 2-16
２．４　イベントフラグ …… 2-17
　２．４．１　イベントフラグの概要 …… 2-17
　２．４．２　事象発生の通知 …… 2-18
　２．４．３　イベントフラグのクリア …… 2-19
　２．４．４　事象発生の待機、取得 …… 2-19
　２．４．５　イベントフラグ状態の参照 …… 2-19
２．５　セマフォ …… 2-20
　２．５．１　セマフォの概要 …… 2-20
　２．５．２　資源の占有要求 …… 2-21
　２．５．３　資源の解放 …… 2-22
　２．５．４　セマフォ状態の参照 …… 2-22
２．６　メイルボックス …… 2-23
　２．６．１　メイルボックスの概要 …… 2-23
　２．６．２　メッセージの送信 …… 2-24
　２．６．３　メッセージの受信 …… 2-25
　２．６．４　メイルボックス状態の参照 …… 2-25
　２．６．５　メッセージフォーマット …… 2-26
　２．６．６　メッセージ送受信の注意事項 …… 2-26
２．７　割込み …… 2-27
　２．７．１　割込みの概要 …… 2-27
　２．７．２　割込みハンドラの作成 …… 2-28
　２．７．３　割込みハンドラの登録 …… 2-29
　２．７．４　割込みマスクレベルの変更 …… 2-30
　２．７．５　割込みマスクレベルの参照 …… 2-30
　２．７．６　未定義割込み …… 2-30

２．７．７　割込みの注意事項 …… 2-31
２．８　例　外 …… 2-32
２．９　メモリプール …… 2-33
２．９．１　メモリプールの概要 …… 2-33
２．９．２　メモリブロックの獲得 …… 2-34
２．９．３　メモリブロックの返却 …… 2-35
２．９．４　メモリプール状態の参照 …… 2-35
２．１０　時　間 …… 2-36
２．１０．１　時間の概要 …… 2-36
２．１０．２　ハードウェアタイマとシステムクロック …… 2-36
２．１０．３　タイマドライバの作成 …… 2-37
２．１０．４　タイマ割込みハンドラの登録 …… 2-38
２．１０．５　システムクロックの設定と参照 …… 2-39
２．１１　拡張ＳＶＣ …… 2-40
２．１１．１　拡張ＳＶＣの概要 …… 2-40
２．１１．２　拡張ＳＶＣハンドラの作成と登録 …… 2-41
２．１２　ＭＣＵの初期化 …… 2-42
２．１２．１　ＭＣＵ初期化の概要 …… 2-42
２．１２．２　ＭＣＵ初期化ルーチンの作成と登録 …… 2-42
２．１３　システム起動処理 …… 2-44
２．１３．１　システム起動処理の概要 …… 2-44
２．１３．２　システム初期化ハンドラ …… 2-46
２．１４　システムの異常終了処理 …… 2-47
２．１５　カーネルのアイドリング …… 2-48
２．１６　トレース機能 …… 2-49
２．１６．１　トレース取得の概要 …… 2-49
２．１６．２　トレースバッファの構造 …… 2-50
２．１６．３　トレース取得データの解析例 …… 2-54
２．１６．４　トレース機能の使用上の注意 …… 2-57

（第３章　システムコール）
３．１　概　要 …… 3-1
３．２　システムコールインタフェース …… 3-4
３．２．１　Ｃ言語インタフェース …… 3-4
３．２．２　アセンブラインタフェース …… 3-7
３．２．３　エラーコード …… 3-10
３．３　システムコール仕様 …… 3-11
３．３．１　can_wup（Cancel Wakeup Task）タスクの起床要求を無効にする［タスク部用］ …… 3-11
３．３．２　chg_ims（Change Interrupt Mask Level）割込みマスクを変更する［タスク部用］ …… 3-13
３．３．３　chg_pri（Change task Priority）タスク優先度を変更する［タスク部用］ …… 3-15
３．３．４　clr_flg（Clear Event Flag）イベントフラグをクリアする［タスク部用］ …… 3-17
３．３．５　cre_tsk（Create Task）タスクを生成する ［タスク部用］ …… 3-19
３．３．６　del_tsk（Delete Task）タスクを削除する ［タスク部用］ …… 3-21
３．３．７　exd_tsk（Exit and Delete Task）自タスクを正常終了後、削除する［タスク部用］ …… 3-23
３．３．８　ext_tsk（Exit Task）自タスクを正常終了する［タスク部用］ …… 3-24

３．３．９　flg_sts（Get Event Flag Status）イベントフラグの状態を参照する
［タスク部用］ ･･･････････････ 3-25
３．３．１０　get_blk（Get Memory Block）固定長メモリブロックの獲得待ちを行なう
［タスク部用］ ･･･････････････ 3-27
３．３．１１　get_tid（Get Task Identifier）自タスクIDを得る［タスク部用］ ･･･････ 3-29
３．３．１２　get_tim（Get Time）システムクロックの値を読み出す［タスク部用］ ･･･････ 3-30
３．３．１３　get_ver（Get Version No）カーネルのバージョン番号を得る
［タスク部用］ ･･･････････････ 3-32
３．３．１４　ican_wup（Interrupt Cancel Wakeup Task）タスクの起床要求を無効にする
［非タスク部用］ ･･･････････････ 3-36
３．３．１５　ichg_ims（Interrupt Change Interrupt Mask Level）割込みマスクを変更する
［非タスク部用］ ･･･････････････ 3-38
３．３．１６　ichg_pri（Interrupt Change task Priority）タスク優先度を変更する
［非タスク部用］ ･･･････････････ 3-40
３．３．１７　iclr_flg（Clear Event Flag）イベントフラグをクリアする［非タスク部用］ ･･･ 3-42
３．３．１８　iflg_sts（Interrupt Get Event Flag Status）イベントフラグの状態を参照する
［非タスク部用］ ･･･････････････ 3-44
３．３．１９　iget_tid（Interrupt Get Task Identifier）自タスクIDを得る［非タスク部用］ ･･･ 3-46
３．３．２０　iget_tim（Interrupt Get Time）システムクロックの値を読み出す
［非タスク部用］ ･･･････････････ 3-47
３．３．２１　iget_ver（Interrupt Get Version No）カーネルのバージョン番号を得る
［非タスク部用］ ･･･････････････ 3-49
３．３．２２　iims_sts（Interrupt Get Interrupt Mask Level Status）割込みマスクを参照する
［非タスク部用］ ･･･････････････ 3-53
３．３．２３　imbx_sts（Interrupt Get Mailbox Status）メイルボックスの状態を参照する
［非タスク部用］ ･･･････････････ 3-54
３．３．２４　impl_sts（Interrupt Get Memory Pool Status）メモリプールの状態を参照する
［非タスク部用］ ･･･････････････ 3-56
３．３．２５　ims_sts（Get Interrupt Mask Level Status）割込みマスクを参照する
［タスク部用］ ･･･････････････ 3-58
３．３．２６　ipget_blk（Interrupt Poll and Get Memory Block）固定長メモリブロックを獲得する
［非タスク部用］ ･･･････････････ 3-59
３．３．２７　ipol_flg（Interrupt Poll Event Flag）イベントフラグを得る［非タスク部用］ ･ 3-61
３．３．２８　iprcv_msg（Interrupt Poll and Receive Message from Mailbox）
メイルボックスから受信する［非タスク部用］ ･･･････････････ 3-63
３．３．２９　ipreq_sem（Interrupt Poll and Request Semaphore）セマフォ資源を
［非タスク部用］ ･･･････････････ 3-65
３．３．３０　irel_blk（Interrupt Release Memory Block）固定長メモリブロックを返却する
［非タスク部用］ ･･･････････････ 3-67
３．３．３１　irel_wai（Interrupt Release Wait）タスクの待ち状態を強制解除する
［非タスク部用］ ･･･････････････ 3-69
３．３．３２　irot_rdq（Interrupt Rotate Ready Queue）タスクのレディーキューを回転する
［非タスク部用］ ･･･････････････ 3-71
３．３．３３　irsm_tsk（Interrupt Resume Task）強制待ち状態のタスクを再開する
［非タスク部用］ ･･･････････････ 3-73
３．３．３４　isem_sts（Interrupt Get Semaphore Status）セマフォの状態を参照する
［非タスク部用］ ･･･････････････ 3-75

３．３．３５ iset_flg（Interrupt Set Event Flag）イベントフラグをセットする
［非タスク部用］ ･･･････････････ 3-77
３．３．３６ iset_tim（Interrupt Set Time）システムクロックを設定する
［非タスク部用］ ･･･････････････ 3-79
３．３．３７ isig_sem（Interrupt Signal Semaphore）セマフォに対する信号操作（Ｖ命令）
［非タスク部用］ ･･･････････････ 3-81
３．３．３８ isnd_msg（Interrupt Send Message to Mailbox）メイルボックスへ送信する
［非タスク部用］ ･･･････････････ 3-83
３．３．３９ ista_tsk（Interrupt Start Task）タスクを起動する［非タスク部用］ ･･･････ 3-85
３．３．４０ isus_tsk（Interrupt Suspend Task）タスクを強制待ち状態へ移行する
［非タスク部用］ ･･･････････････ 3-87
３．３．４１ itsk_sts（Interrupt Get Task Status）タスク状態を見る［非タスク部用］ ･････ 3-89
３．３．４２ iwup_tsk（Interrupt Wakeup Task）待ち状態のタスクを起床する
［非タスク部用］ ･･･････････････ 3-91
３．３．４３ mbx_sts（Get Mailbox Status）メイルボックスの状態を参照する
［タスク部用］ ･･･････････････ 3-93
３．３．４４ mpl_sts（Get Memory Pool Status）メモリプールの状態を参照する
［タスク部用］ ･･･････････････ 3-95
３．３．４５ pget_blk（Poll and Get Memory Block）固定長メモリブロックを獲得する
［タスク部用］ ･･･････････････ 3-97
３．３．４６ pol_flg（Poll Event Flag）イベントフラグを得る［タスク部用］ ･･･････ 3-99
３．３．４７ prcv_msg（Poll and Receive Message from Mailbox）メイルボックスから受信する
［タスク部用］ ･･･････････････ 3-101
３．３．４８ preq_sem（Poll and Request Semaphore）セマフォ資源を得る
［タスク部用］ ･･･････････････ 3-103
３．３．４９ rcv_msg（Receive Message from Mailbox）メイルボックスからの受信を待つ
［タスク部用］ ･･･････････････ 3-105
３．３．５０ rel_blk（Release Memory Block）固定長メモリブロックを返却する
［タスク部用］ ･･･････････････ 3-107
３．３．５１ rel_wai（Release Wait）タスクの待ち状態を強制解除する［タスク部用］ ･･･ 3-109
３．３．５２ ret_exc（Return from Exception）例外処理から復帰する
［タスク部／非タスク部用］ ･･･････････････ 3-111
３．３．５３ ret_int（Return from Interrupt Handler）割込みハンドラから復帰する
［非タスク部用］ ･･･････････････ 3-112
３．３．５４ rot_rdq（Rotate Ready Queue）タスクのレディーキューを回転する
［タスク部用］ ･･･････････････ 3-114
３．３．５５ rsm_tsk（Resume Task）強制待ち状態のタスクを再開する［タスク部用］ ･･ 3-116
３．３．５６ sem_sts（Get Semaphore Status）セマフォの状態を参照する
［タスク部用］ ･･･････････････ 3-118
３．３．５７ set_flg（Set Event Flag）イベントフラグをセットする［タスク部用］ ･･･････ 3-120
３．３．５８ set_tim（Set Time）システムクロックを設定する［タスク部用］ ･･･････ 3-122
３．３．５９ sig_sem（Signal Semaphore）セマフォに対する信号操作（Ｖ命令）
［タスク部用］ ･･･････････････ 3-124
３．３．６０ slp_tsk（Sleep Task）自タスクを待ち状態へ移行する［タスク部用］ ･･･････ 3-126
３．３．６１ snd_msg（Send Message to Mailbox）メイルボックスへ送信する
［タスク部用］ ･･･････････････ 3-127
３．３．６２ sta_tsk（Start Task）タスクを起動する ［タスク部用］ ･･･････････････ 3-129

３．３．６３　sus_tsk（Suspend Task）タスクを強制待ち状態へ移行する［タスク部用］・ 3-131
３．３．６４　sys_cal（System Call）拡張SVCの発行［タスク部用］ ····· 3-133
３．３．６５　sys_clk（Request Kernel Timer Processing）
カーネルの時間管理処理を要求する［非タスク部用］····· 3-135
３．３．６６　ter_tsk（Terminate Task）他タスクを強制的に異常終了させる
［タスク部用］ ····· 3-136
３．３．６７　tsk_sts（Get Task Status）タスク状態を見る［タスク部用］ ····· 3-138
３．３．６８　wai_flg（Wait Event Flag）イベントフラグを待つ［タスク部用］ ····· 3-140
３．３．６９　wai_sem（Wait on Semaphore）セマフォに対する待ち操作（Ｐ命令）
［タスク部用］ ····· 3-143
３．３．７０　wai_tsk（Wait Task）自タスクを一定時間待ち状態へ移行する［タスク部用］・ 3-145
３．３．７１　wup_tsk（Wakeup Task）待ち状態のタスクを起床する［タスク部用］····· 3-147

（第４章　C言語によるプログラムの作成方法）
４．１　C言語インタフェース ····· 4-1
４．２　スタックの使用量について ····· 4-1
４．３　C言語によるタスクの記述方法 ····· 4-2
４．４　C言語による割込みハンドラの記述方法 ····· 4-3
４．４．１　割込みハンドラの基本形 ····· 4-3
４．４．２　タイマ割込みハンドラの記述方法 ····· 4-4
４．４．３　カーネル割込みマスクレベルより高い割込みハンドラの記述方法 ····· 4-5
４．５　C言語による拡張SVCハンドラの記述方法 ····· 4-6
４．６　C言語によるMCU初期化ルーチンの記述方法 ····· 4-8
４．７　C言語によるシステム初期化ハンドラの記述方法 ····· 4-9
４．８　C言語によるシステム異常終了処理ルーチンの記述方法 ····· 4-10
４．９　C言語による例外処理ルーチンの記述方法 ····· 4-11

（付録Ａ　コンソールドライバの例題）
Ａ．１　コンソールドライバの概要 ····· A-1
Ａ．２　コンソールドライバの機能 ····· A-2
Ａ．３　メッセージフォーマット ····· A-2
Ａ．４　コンソールドライバのファイル構成 ····· A-3
Ａ．５　コンソールドライバの定義情報 ····· A-4
Ａ．５．１　コンソールドライバの動作環境データ ····· A-4
Ａ．５．２　ハードウェア初期化情報 ····· A-5
Ａ．５．３　ハードウェアI/Oアドレス ····· A-6
Ａ．６　コンソールドライバのプログラム内容 ····· A-7
Ａ．６．１　モジュール構成 ····· A-7
Ａ．６．２　共通メモリ領域 ····· A-7
Ａ．６．３　コンソールドライバタスクのメイン処理(hi_cnsdrv) ····· A-8
Ａ．６．４　コンソールの初期化処理(hi_cnsini) ····· A-9
Ａ．６．５　コンソールからの入力処理(hi_cnsinp) ····· A-10
Ａ．６．６　コンソールへの出力処理(hi_cnsout) ····· A-11
Ａ．６．７　SCI受信割込みハンドラ(hi_cnshdrrx) ····· A-12
Ａ．６．８　SCI送信割込みハンドラ(hi_cnshdrtx) ····· A-13
Ａ．６．９　SCIエラー割込みハンドラ(hi_cnshdrer) ····· A-14

（付録Ｂ　タイマドライバの例題）
Ｂ．１　タイマドライバの処理概要 …… B-1
Ｂ．２　タイマドライバのファイル構成 …… B-2
Ｂ．３　タイマドライバの定義情報 …… B-2
　Ｂ．３．１　タイマドライバの動作環境データ …… B-2
　Ｂ．３．２　ハードウェア初期化情報 …… B-3
　Ｂ．３．３　ハードウェアI/Oアドレス …… B-5
Ｂ．４　タイマドライバのプログラム内容 …… B-6
　Ｂ．４．１　タイマ割込みハンドラ用スタック領域 …… B-6
　Ｂ．４．２　タイマ初期化ルーチン(hi_tmrini) …… B-7
　Ｂ．４．３　タイマ割込みハンドラ(hi_tmrhdr) …… B-8

（付録Ｃ　システムコール一覧）
Ｃ．１　システムコールＣ言語インタフェース一覧 …… C-1
Ｃ．２　システムコール・アセンブラインタフェース一覧 …… C-3

（付録Ｄ　ＳＨ３に関する注意事項）
Ｄ．１　エンディアン …… D-1
Ｄ．２　ＳＲレジスタ状態 …… D-1
Ｄ．３　ＶＢＲレジスタ …… D-2
Ｄ．４　ＳＰＣ，ＳＳＲレジスタ …… D-2
Ｄ．５　キャッシュ …… D-2
Ｄ．６　ＭＭＵ …… D-3
Ｄ．７　プログラム配置 …… D-3

（付録Ｅ　エラーコード一覧）
Ｅ．１　システムコールエラーコード一覧 …… E-1
Ｅ．２　システム異常終了エラーコード一覧 …… E-2
Ｅ．３　セットアップテーブルエラーコード一覧 …… E-3

（付録Ｆ　ヘッダファイル）
Ｆ．１　Ｃ言語用ヘッダファイル …… F-1
Ｆ．２　アセンブリ言語用ヘッダファイル …… F-10

（付録Ｇ　ＡＳＣＩＩコード表）
Ｇ．１　ＡＳＣＩＩコード表 …… G-1

（付録Ｈ　索　引）
Ｈ．１　五十音順・索引 …… H-1
Ｈ．２　アルファベット順・索引 …… H-4

図目次

（第１章　概　説）
図１－１　システムの構築手順 ····· 1-4

（第２章　機　能）
図２－１　HI-SH77のシステム構成 ····· 2-2
図２－２　システム状態 ····· 2-3
図２－３　タスクとカーネルの関係 ····· 2-5
図２－４　タスクの状態遷移 ····· 2-7
図２－５　共有スタック機能の動作 ····· 2-10
図２－６　共有スタック機能使用時のタスク状態遷移 ····· 2-11
図２－７　レディーキュー操作によるスケジューリング ····· 2-12
図２－８　irot_rdqシステムコールによるラウンドロビンスケジューリング ····· 2-13
図２－９　タスク実行中の割込みマスク ····· 2-16
図２－１０　イベントフラグ機構の概要 ····· 2-18
図２－１１　セマフォ機構の概要 ····· 2-21
図２－１２　メイルボックス機構の概要 ····· 2-24
図２－１３　メッセージフォーマット ····· 2-26
図２－１４　割込みハンドラへの制御移行 ····· 2-27
図２－１５　ベクタテーブルと割込みハンドラとの関連 ····· 2-29
図２－１６　ret_excシステムコール発行時のスタック ····· 2-32
図２－１７　メモリプール機構の概要 ····· 2-34
図２－１８　ハードウェアタイマとシステムクロックの関係 ····· 2-36
図２－１９　タイマドライバの処理 ····· 2-37
図２－２０　ベクタテーブルとタイマ割込みハンドラとの関連 ····· 2-38
図２－２１　時刻の設定および参照 ····· 2-39
図２－２２　拡張SVCの使用例 ····· 2-40
図２－２３　MCU初期化ルーチンの処理概要 ····· 2-42
図２－２４　システム起動処理の概要 ····· 2-45
図２－２５　トレース取得時の動作概要 ····· 2-49
図２－２６　トレースバッファの構造 ····· 2-50
図２－２７　トレースバッファ管理テーブルの構造 ····· 2-50
図２－２８　トレースバッファ管理の様子 ····· 2-51
図２－２９　トレースエントリの構造 ····· 2-52
図２－３０　トレース解析結果の図示例 ····· 2-56

（第４章　Ｃ言語によるプログラムの作成方法）
図４－１　C言語によるタスクの記述例 ･･･ 4-2
図４－２　C言語による割込みハンドラの記述例 ････････････････････････････････････ 4-3
図４－３　C言語によるタイマ割込みハンドラの記述例 ･････････････････････････････ 4-5
図４－４　C言語によるカーネル割込みレベルより高い割込みハンドラの記述例 ･････････ 4-5
図４－５　C言語による拡張SVCハンドラの記述例 ･････････････････････････････････ 4-6
図４－６　C言語による拡張SVCの発行例 ･･･ 4-7
図４－７　C言語によるMCU初期化ルーチンの記述例 ･･･････････････････････････････ 4-8
図４－８　C言語によるシステム初期化ハンドラの記述例 ･･････････････････････････ 4-9
図４－９　C言語によるシステム異常終了処理ルーチンの記述例 ････････････････････ 4-10
図４－１０　C言語による例外処理ルーチンの記述例 ･････････････････････････････ 4-11

（付録Ａ　コンソールドライバの例題）
図Ａ－１　コンソールドライバの処理概要 ･･････････････････････････････････････ A-1
図Ａ－２　コンソールドライバ・ユーザタスク間のメッセージフォーマット ･･････････ A-2
図Ａ－３　T_CIO構造体の内容 ･･ A-3
図Ａ－４　コンソールドライバの動作環境データ ･･･････････････････････････････ A-4
図Ａ－５　ハードウェア初期化情報 ･･ A-5
図Ａ－６　ハードウェアI/Oアドレス ･･･ A-6
図Ａ－７　共通メモリ領域 ･･･ A-7
図Ａ－８　コンソールドライバタスクのメイン処理(hi_cnsdrv) ･･････････････････････ A-8
図Ａ－９　コンソールの初期化処理(hi_cnsini) ･･････････････････････････････････ A-9
図Ａ－１０　コンソールからの入力処理(hi_cnsinp) ･･････････････････････････････ A-10
図Ａ－１１　コンソールへの出力処理(hi_cnsout) ･･･････････････････････････････ A-11
図Ａ－１２　SCI受信割込みハンドラ(hi_cnshdrrx) ･･････････････････････････････ A-12
図Ａ－１３　SCI送信割込みハンドラ(hi_cnshdrtx) ･･････････････････････････････ A-13
図Ａ－１４　SCIエラー割込みハンドラ(hi_cnshdrer) ･････････････････････････････ A-14

（付録Ｂ　タイマドライバの例題）
図Ｂ－１　タイマドライバの処理概要 ･･ B-1
図Ｂ－２　タイマドライバの動作環境データ ････････････････････････････････････ B-2
図Ｂ－３　ハードウェアの初期化情報 ･･ B-3
図Ｂ－４　ハードウェアI/Oアドレス ･･･ B-5
図Ｂ－５　タイマ割込みハンドラが使用するスタック領域 ････････････････････････ B-6
図Ｂ－６　タイマ初期化ルーチン(hi_tmdini) ････････････････････････････････････ B-7
図Ｂ－７　タイマ割込みハンドラ(hi_tmrhdr) ･･･････････････････････････････････ B-8

表目次

（第２章　機　能）
表２－１　　システム状態と発行できるシステムコールの種類の関係 ・・・・・・・・・・・・・・・・・・・・・・・・ 2-4
表２－２　　CPUレジスタの初期化 ・・ 2-9
表２－３　　タスク実行中断の要因と期間 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ 2-14
表２－４　　タスクの終了要因 ・・ 2-15
表２－５　　割込みハンドラの処理条件 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ 2-29
表２－６　　拡張SVCハンドラの処理条件 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ 2-41
表２－７　　MCU初期化ルーチンの処理条件 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ 2-43
表２－８　　システム初期化ハンドラの処理条件 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ 2-46
表２－９　　システム異常終了処理の入力パラメーター一覧 ・・・・・・・・・・・・・・・・・・・・・・・・・・・ 2-47
表２－１０　トレースエントリ内容の意味 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ 2-53
表２－１１　トレース取得データの例 ・・ 2-54

（第３章　システムコール）
表３－１　　システムコールの分類 ・・ 3-1
表３－２　　システムコールの発行可能なシステム状態と仕様記載ページ ・・・・・・・・・・・・・・・・・・・・ 3-2

（第４章　Ｃ言語によるプログラムの作成方法）
表４－１　　C言語インタフェースの構成ファイル ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ 4-1
表４－２　　割込みハンドラの記述方法と割込み仕様の指定内容一覧 ・・・・・・・・・・・・・・・・・・・・・・ 4-4

（付録Ａ　　コンソールドライバの例題）
表Ａ－１　　コンソールドライバのモジュール構成 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ A-7

（付録Ｄ　ＳＨ３に関する注意事項）
表Ｄ－１　　各種プログラムでのＳＲレジスタ状態 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ D-1

（付録Ｅ　エラーコード一覧）
表Ｅ－１　　システムコールエラーコード一覧 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ E-1
表Ｅ－２　　システム異常終了エラーコード一覧 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ E-2
表Ｅ－３　　セットアップテーブルエラーコード一覧 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ E-3

１．概　説

１．１　概　要

マイクロコンピュータ応用分野の広がりに伴い、**OS(Operating System)**の役割と重要性が高まってきています。このなかで、産業用システムに用いられる**OS**としてリアルタイム**OS**があります。

HI-SH77は、日立**SH3 CPU**上で動作する、**μITRON**仕様に準拠した産業機器組込み用リアルタイム・マルチタスク**OS**です。

１．２　特　長

HI-SH77の特長を以下に示します。

(1)μITRON仕様に準拠

シングルチップマイコン用に設計され、リアルタイム**OS**の仕様である**μITRON**仕様のVer2.02に準拠しています。

(2)高速な**OS**

RISC(Reduced Instruction Set Computer)タイプの命令セットを持つ**SH3**の高速性能が発揮できるように**OS**を最適化し、高速な処理を実現しています。

また、**OS**内でのパラメータチェックを行なわないようにできるなど、リアルタイム性を向上させる工夫をしています。

(3)機能選択可能なコンパクトな**OS**

少量のメモリでユーザシステムが稼働できるように、**OS**サイズおよび**OS**作業領域の小型化を図っています。

また、ユーザプログラムで使用する**OS**の機能モジュールをセットアップテーブルで選択することにより、必要最小限のモジュールで**OS**を容易に構築できます。

(4)高級言語対応

C言語インタフェースヘッダと日立**SH**シリーズ**C**コンパイラの使用により、タスク、割込みハンドラなど全てのプログラムを**C**言語で記述することができます。

(5)豊富なリアルタイム・マルチタスク処理機能

・優先度に基づくタスクスケジューリング
・タスクの生成、削除、起動、終了などのタスク管理機能
・タスクの実行中断、再開などのタスク付属同期機能

・イベントフラグ、セマフォ、メイルボックスによるタスク間の同期／通信機能
・割込み管理機能
・メモリブロックの獲得、開放などのメモリプール管理機能
・システムクロックの設定、参照、タスクの実行遅延などの時間管理機能
・拡張**SVC**機能

(6)デバイスドライバを容易に作成可能

コンソールドライバとタイマドライバのソースプログラムをサンプルとして提供していますので、各種の**I/O**ドライバを容易に作成できます。

１．３　構　成

HI-SH77を構成する主要なプログラムを以下に示します。

(1)カーネル

タスク管理、タスク間の同期／通信、時間管理など、リアルタイム・マルチタスク制御の中枢となるプログラムです。パラメータチェック機能なし、およびパラメータチェック機能付きの2種類を選択できます。ユーザシステム構築の初期段階には、パラメータチェック機能付きを使用することにより、デバッグが容易になります。デバッグ終了後は、パラメータチェック機能なしを用いることにより、システムのリアルタイム性を向上できます。

(2)セットアップテーブル

使用する**OS**機能や、タスク、イベントフラグなど各種資源の情報を定義するためのテーブルです。ユーザシステムに必要な**OS**の機能モジュールの選択は、セットアップテーブルファイルに定義するだけで自動的に行なわれます。

(3)C言語インタフェースヘッダ

C言語で記述したユーザプログラムから、システムコールを呼び出すためのヘッダです。

(4)コンソールドライバ

SH7702,SH7708 MCU内蔵のシリアルコミュニケーションインタフェース**(SCI:Serial Communication Interface)**を使用した、コンソールを制御するサンプルプログラムです。コンソールを使用しない場合は、ユーザシステムに組み込む必要はありません。

(5)タイマドライバ

SH7702,SH7708 MCU内蔵のタイマユニット**(TMU:Timer Unit)**を使用した、システムクロックを制御するサンプルプログラムです。**wai_tsk, set_tim,iset_tim, get_tim,iget_tim, sys_clk**システムコールを使用しない場合は、ユーザシステムに組み込む必要はありません。

１．４　システムの構築手順

ユーザシステムの構築手順を**図１－１**に示します。なお、システム構築の詳細については『**HI-SH77構築マニュアル**』を参照してください。

(1)ユーザプログラムの作成

C言語またはアセンブリ言語でユーザプログラムを記述します。必要な**I/O**ドライバは、提供されるデバイスドライバを参考にして作成します。提供されるヘッダファイルと共にコンパイル、またはアセンブルを行なってオブジェクトモジュールファイルを生成します。

(2)ユーザシステム用セットアップテーブルの作成

提供されるセットアップテーブルに、ユーザシステムで必要な**OS**機能の選択、および**OS**を稼働させるために必要なユーザシステムの環境を定義します。そして、このファイルをコンパイルしてオブジェクトモジュールファイルを生成します。

(3)ロードモジュールファイルの作成

ユーザプログラム、セットアップテーブルのオブジェクトモジュールファイル、カーネル機能モジュールのライブラリファイルをリンケージエディタで結合し、実行可能なロードモジュールファイルを作成します。

(4)プログラムのロード

作成したロードモジュールファイルをユーザ実機にロードするには、次の方法があります。

①**SH3**用インサーキットエミュレータを使用する方法

エミュレータのロード機能を使用して、ユーザ実機にプログラムをロードします。

②**ROM**を使用する方法

ROMにプログラムを書き込み、ユーザ実機に搭載します。

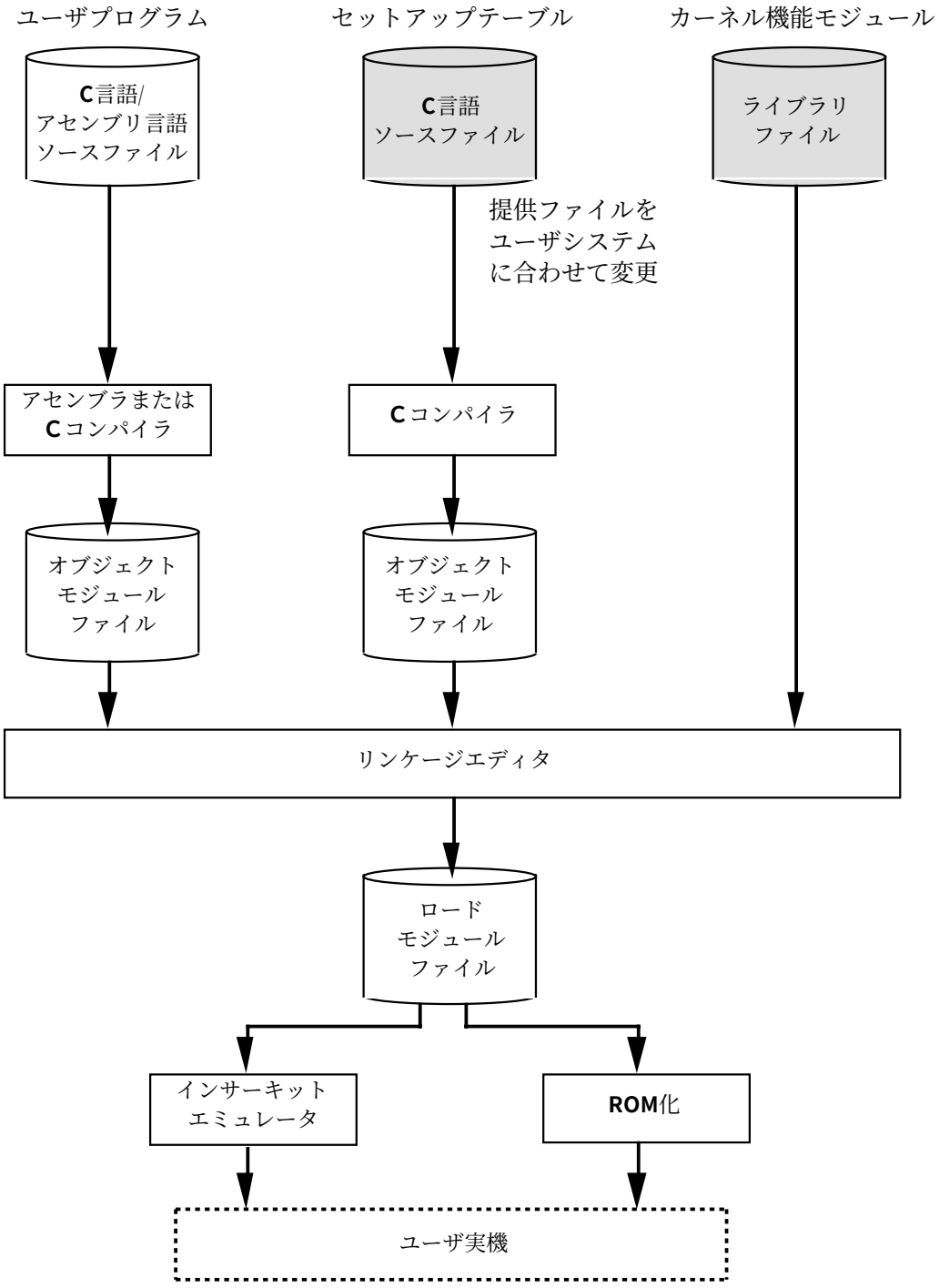


図１－１　システムの構築手順

：HI-SH77の構成要素

２．機　能

２．１　概　要

リアルタイム・マルチタスクシステムでは、ユーザは次の2種類のプログラムを作成します。

・タスク（独立かつ並列に処理可能な単位に分割されたプログラム）

・割込みなどに用いる各種ハンドラ（いずれのタスクにも属さない処理を行なうプログラム）

OSのリアルタイム・マルチタスク処理を行なう核となる部分を「カーネル」と呼びますが、ユーザは上記２種類のプログラムを、カーネルの以下の3つの役割りを利用して作成し、リアルタイム・マルチタスクシステムを実現します。

・タスクをその優先度に応じて、実行順序を決定する

・プログラムからの各種処理要求（システムコール）を受け付け、その処理を行なう

・非同期に発生する事象（イベント）を認識し、その事象を処理するプログラムを起動する

本章では、ユーザがリアルタイム・マルチタスクシステムを実現するために利用できるカーネルの働きを、**HI-SH77**の機能に沿って説明します。

図２－１に**HI-SH77**のシステム構成を示します。

(1)タスク管理（スケジューラを含む）機能

タスクには、実行時に**CPU(Central Processing Unit)**が割り付けられます。この割り付け順序をスケジューラ（タスクの実行順序決定機構）と共に制御したり、タスクの起動、終了などのタスクの状態を管理します。

(2)タスク付属同期機能

タスクの実行中断、再開など、タスクの基本的な同期処理を行ないます。

(3)同期／通信機能

タスク間の同期／通信処理をイベントフラグ、セマフォ、およびメイルボックスを用いて行ないます。

(4)割込み管理機能

外部割込みが発生すると、割込みハンドラが起動され、割込み処理やタスクへの割込み発生の連絡が行なわれます。割込みハンドラはユーザが作成します。

(5)メモリプール管理機能

ユーザシステム内の未使用メモリはメモリプールとして管理できます。タスクは必要に応じてメモリプールからメモリブロックを動的に獲得、返却します。

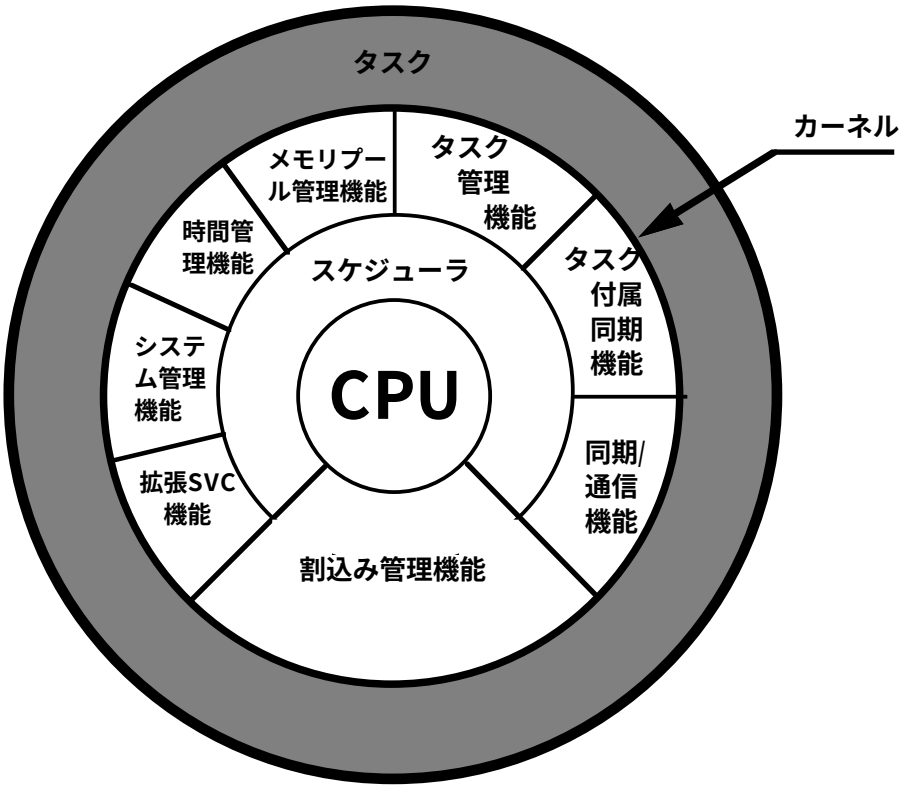


図２－１　HI-SH77のシステム構成

(6)時間管理機能

時刻の管理を行ないます。また、タスクの実行制御のための時間監視を行ないます。

(7)システム管理機能

カーネルのバージョン番号の読出しなどのシステム管理を行ないます。

(8)拡張**SVC**機能

ユーザが作成したプログラム（拡張**SVC**ハンドラ）を、カーネルの**1**つの機能かのように、通常のシステムコールと同様の手続きで呼び出せるようにします。

２．２　システム状態

カーネルの機能を利用するために発行するシステムコールの種類は、システム状態により異なります。システム状態は、**図２－２**のように分類されます。

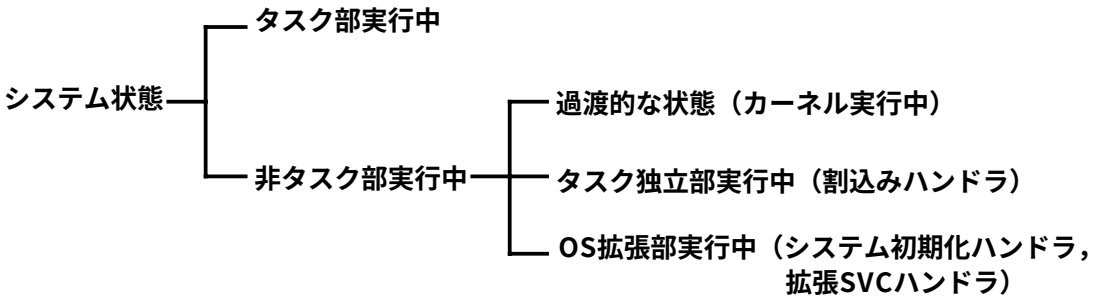


図２－２　システム状態

(1)タスク部実行中

タスクを実行している状態を、タスク部実行中といいます。

タスク内では、タスク部用のシステムコールを用います。

(2)非タスク部実行中

タスク以外の部分を実行している間のシステム状態を、非タスク部実行中といいます。非タスク部実行中は、さらに過渡的な状態、タスク独立部実行中、および**OS**拡張部実行中に分類されます。

(a)過渡的な状態

カーネル実行中（システムコール処理中）の状態です。

(b)タスク独立部実行中

タスク独立部は、通常は割込みハンドラの部分です。

タスク独立部はタスクとは完全に独立しており、自タスクの概念はありません。したがって、自分自身を待ち状態にするなど、自タスクを指定するシステムコールは、タスク独立部から発行できません。

タスク独立部では、非タスク部用のシステムコールを用います。

また、タスク独立部は現在実行中のタスクを特定できないので、タスクの切り換えは起こりません。タスク切り換えは、タスク独立部の終了を要求する**ret_int**システムコールを発行するまで遅延されます。

タスクは、割込みマスクを変更するシステムコール**(chg_ims)**を使用することにより、タスク独立部に遷移することができます。

割込みマスク値を**0**以外に変更したタスクは、タスク独立部として動作し、非タスク部用のシステムコールを使用することになります。タスクの切り換えは、割込みマスクを変更するシステムコールによりマスク値が**0**に戻されるまで遅延されます。

タスク実行中の割込みマスクについては、**「２．３．９　タスク実行中の割込みマスク」**を参照してください。

(c)OS拡張部実行中

OS拡張部はユーザが**OS**の機能を拡張した部分で、システム初期化ハンドラと拡張**SVC**ハンドラです。

システム初期化ハンドラは、カーネルのシステム起動処理を行なった後、カーネルから呼び出され、初期登録タスクの起動やハードウェアの初期化などを行なうものです。拡張**SVC**ハンドラは、ユーザが作成したプログラムをカーネルに登録し、通常のシステムコールと同様の手続きで呼び出せるようにするものです。

システム初期化ハンドラおよび拡張**SVC**ハンドラを使用するには、セットアップテーブルに定義する必要があります。

OS拡張部では非タスク部用のシステムコールを使用することができます。

表２－１にシステム状態と発行できるシステムコールの種類の関係をまとめます。

表２－１　システム状態と発行できるシステムコールの種類の関係

項番	システム状態	プログラムの種類	発行できるシステムコールの種類
1	タスク部	タスク	タスク部用
2	過渡的な状態	カーネル	—
3	タスク独立部	割込みハンドラ	非タスク部用
4		割込みマスク値を0以外に変更したタスク	非タスク部用
5	OS拡張部	システム初期化ハンドラ	非タスク部用
6		拡張**SVC**ハンドラ	非タスク部用

２．３　タスク

２．３．１　タスクとカーネルの関係

リアルタイム・マルチタスクシステムでは、ユーザはプログラムを独立して並列に処理可能な単位に分割して作成します。この分割したプログラム単位を「タスク」と呼びます。ユーザは最大**1023**個のタスクをカーネルに登録でき、**1**から**1023**までのタスク**ID**と呼ぶ番号で識別します。システムで使用するタスクの数はセットアップテーブルに定義します。

カーネルに登録したこれらのタスクは、タスク間での必要な連絡を、カーネル機能の処理要求であるシステムコールを使用して行ないます。外部装置や計算機内部から非同期に発生する事象（イベント）は、カーネルを通して、その事象を処理するタスクの実行を要求します。

図２－３にタスクとカーネルの関係を示します。

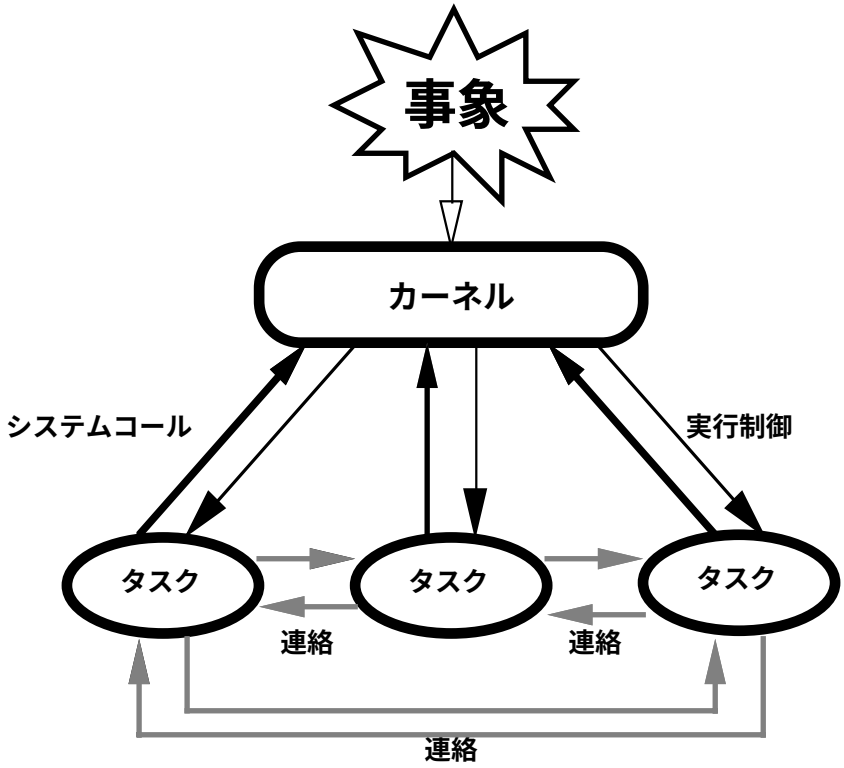


図２－３　タスクとカーネルの関係

２．３．２　タスクの状態と遷移

タスクは、外部や内部の事象により次の7つの状態を遷移します。

⑴未登録(**NON-EXISTENT)**状態

タスクがカーネルに登録されていない仮想的な状態です。

この状態のタスクが**cre_tsk**システムコールにより登録されると、休止状態になります。また、休止状態のタスクが**del_tsk**システムコールにより削除されたり、実行中のタスクが**exd_tsk**システムコールにより終了削除されると、未登録状態になります。

(2) 休止(**DORMANT)**状態

タスクがカーネルに登録された後、まだ起動されていない状態、または終了後の状態です。

この状態のタスクが**sta_tsk**(または**ista_tsk**)システムコールにより起動されると、実行可能状態になります。また、実行中のタスクが**ext_tsk**システムコールで終了すると、休止状態になります。

(3) 実行可能(**READY)**状態

タスクを実行するための準備がすべて整った状態ですが、他の高い優先度のタスクが実行中のため実行を待っている状態です。

休止状態のタスクが**sta_tsk**(または**ista_tsk**)システムコールにより起動されたとき、待ち状態のタスクの待ちが解除されたとき、または強制待ち状態のタスクに**rsm_tsk**(または**irsm_tsk**)システムコールが発行されたとき、実行可能状態になります。

(4) 実行(**RUN)**状態

CPUが割り付けられ、現在実行中の状態です。

実行可能状態のタスクの中で最も高い優先度のタスクが、実行状態になり実行されます。

(5) 待ち(**WAIT)**状態

タスクが何らかの事象の発生を待っている状態です。

実行状態のタスクが待ちを伴うシステムコールを発行し、条件が満足されないとき待ち状態になります。待ち状態のタスクの待ちが解除されると実行可能状態になります。

(6)強制待ち(**SUSPEND)**状態

タスクの実行が他のタスクからの要求により強制的に中断された状態です。

実行可能状態のタスクに**sus_tsk**(または**isus_tsk**)システムコールが発行されると、強制待ち状態になります。強制待ち状態は、**rsm_tsk**(または**irsm_tsk**)システムコールにより解除されます。

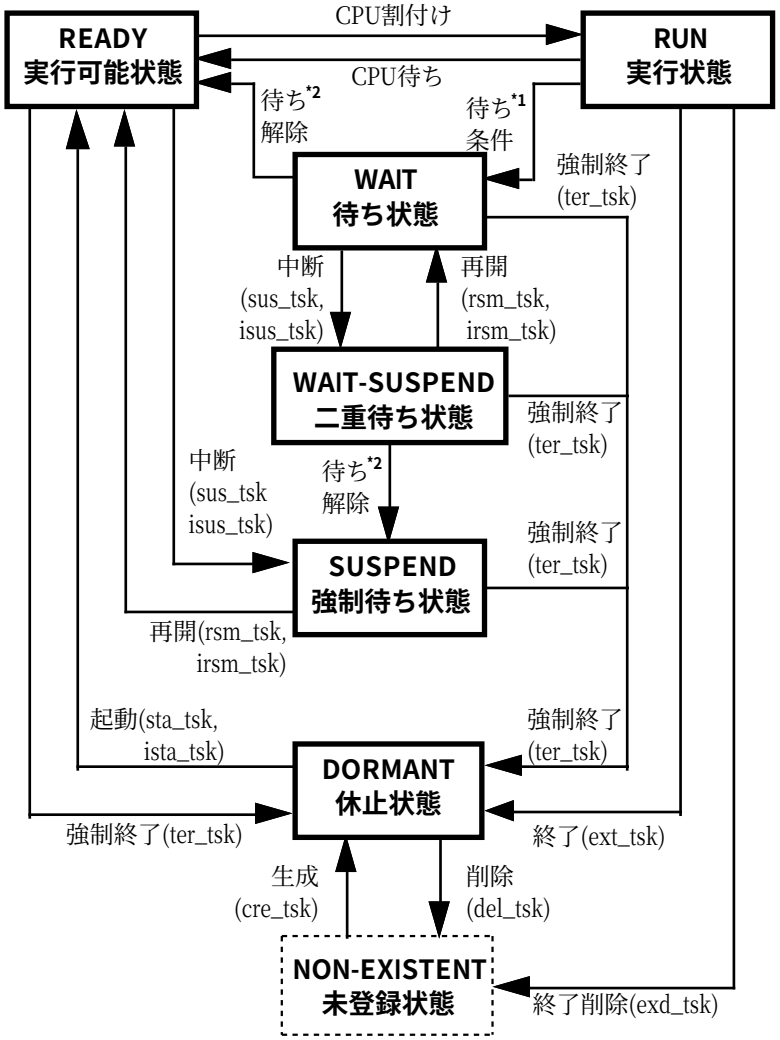
(7)二重待ち(**WAIT-SUSPEND)**状態

待ち状態と強制待ち状態が重なった状態です。

待ち状態のタスクは、**sus_tsk**(または**isus_tsk**)システムコールにより二重待ち状態になります。二重待ち状態は、待ち状態が解除され、かつ**rsm_tsk**(または**irsm_tsk**)システムコールにより強制待ち状態が解除

されたとき、実行可能状態になります。

図２－４にタスクの状態遷移を示します。



【注】*1　待ち条件となるシステムコール

get_blk, rcv_msg, slp_tsk, wai_flg, wai_sem, wai_tsk

*2　待ち解除となるシステムコール

irel_blk, irel_wai, iset_flg, isig_sem, isnd_msg, iwup_tsk,

rel_blk, rel_wai, set_flg, sig_sem, snd_msg, wup_tsk, 時間経過

図２－４　タスクの状態遷移

２．３．３　タスクの生成

タスクを未登録(NON-EXISTENT)状態から休止(DOMANT)状態にすることを「タスクを生成する」といいます。セットアップテーブルで定義した最大タスクIDまでの個数のタスクを生成できます。

タスクの生成は、システム構築時のセットアップテーブルへの定義、またはcre_tskシステムコールにより行ないます。このとき、タスクの制御、実行に必要な次の3つのパラメータを設定します。

(1)タスクID(tskid)

タスクを識別するための番号で、複数のタスクに同じ番号を付けることはできません。

なお、タスクのスタック領域は、各タスクID毎にセットアップテーブルに定義します。

(2)タスク開始アドレス(stadr)

タスクが実行を開始するときに制御が渡されるアドレスです。

(3)初期タスク優先度(itskpri)

タスクの実行順序を決めるための値であり、タスク生成時に付けるものを「初期タスク優先度」といいます。タスク優先度は、1～255の範囲を使用します。タスク優先度の値が小さいほど、優先順位は高くなります。タスク優先度は、複数のタスクに同じ優先度を設定することができます。

なお、タスク優先度はchg_pri,ichg_priシステムコールを用いることにより、タスク生成後でも変更が可能です。

２．３．４　タスクの起動と実行開始時の初期化

休止(DOMANT)状態のタスクを実行可能(READY)状態にすることを、「タスクを起動する」といいます。起動されて実行可能状態になるのは次の場合です。

- 他タスクからsta_tskシステムコールが発行された
- システム初期化ハンドラや割込みハンドラなどの非タスク部から、ista_tskシステムコールが発行された

タスクにCPUが割り付けられ、実行(RUN)状態になるとき、**表２－２**に示すCPUレジスタの初期化が行なわれます。

タスク起動時は特権モードで動作します。

表２－２に示したレジスタが、各タスクのコンテキストとなります。特に、グローバルベースレジスタ(GBR)もタスクコンテキストに含まれるので、システムとして共通の値を保持するわけでないことに注意してください。GBRを使用する場合は、各タスク毎に値を設定してください。

なお、ベクタベースレジスタ(VBR)は、HI-SH77応用システムでは常に固定と仮定しています。VBRの初期化はカーネルのリセットサービスルーチンで行ないますので、その後は変更しないでください。変更した場合、システムの正常な動作は保証されません。

表２－２　CPUレジスタの初期化

項番	レジスタ名	初期化の内容	
1	プログラムカウンタ(**PC**)	タスク生成で指定したタスク開始アドレス(**stadr**)	
2	ステータスレジスタ(**SR**)	モード**(MD)**	特権モード**(1)**
		レジスタバンク**(RB)**	バンク**0(0)**
		例外マスク**(BL)**	例外マスク解除**(0)**
		割込みマスク**(I3〜I0)**	割込みマスク解除**(0)**
3	スタックポインタ(**R15**)	セットアップテーブルで定義した スタック領域最終アドレス - 4	
4	汎用レジスタ(**R0** 〜 **R14**) 積和レジスタ(**MACH,MACL**) プロシージャレジスタ(**PR**) グローバルベースレジスタ(**GBR**)	不定	

【注】**HI-SH77**応用システムでは、ベクタベースレジスタ(**VBR**)を動的に変更することを仮定してません。**VBR**の値はカーネルのリセットサービスルーチンで設定しますので、その後はいっさい変更しないでください。

２．３．５　共有スタック機能

共有スタック機能により、複数のタスクで1つのタスクスタック領域を共有することができます。この機能により、システム全体のタスクスタック領域の容量を減らすことが可能です。

共有スタックは、セットアップテーブルに定義して実現します。セットアップテーブルで初期タスクスタックポインタに同じ値を登録したタスク**ID**同士は、それらのタスク間でひとつのスタックを共有する(共有スタック)ことになります。スタックを共有するタスク群では、同時には1タスクのみがスタックを占有して実行する権利が与えられます。以下にその詳しい説明を示します。

図２－５は共有スタック機能の処理概要です。

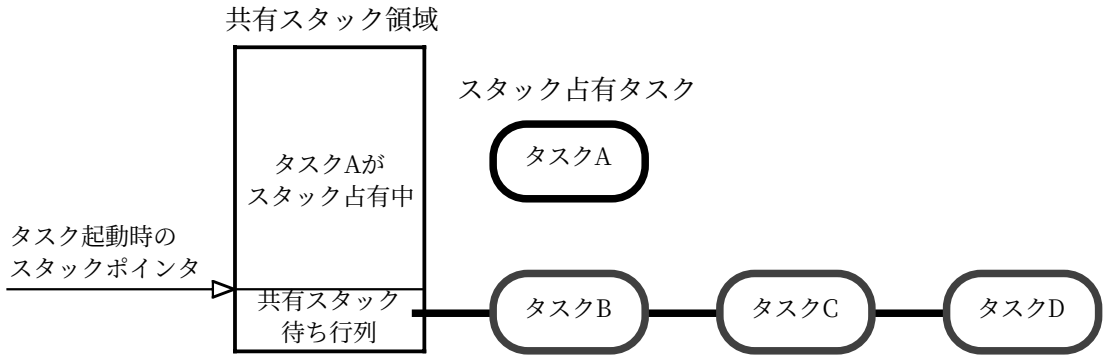
タスク**A**、**B**、**C**、**D**は、同一のスタック領域を共有し、初期状態はいずれも休止**(DOMANT)**状態にあるものとします。

まず、タスク**A**に起動要求(**sta_tsk, ista_tsk**システムコール発行)が行なわれると、タスク**A**が共有スタック領域を使用(占有)します。タスク**A**が共有スタック領域を占有中に、この共有スタック領域を使用(占有)しようとするタスク**B**、**C**、**D**に順番に起動要求が行なわれた場合、タスク**B**、**C**、**D**は共有スタック領域を使用(占有)できません。そのため、タスク**A**以外は実行可能**(READY)**状態になることができず、起動要求を保留された共有スタック待ち状態となります**(**起動要求の保留**)**。このとき、たとえタスク**B**がタスク**A**より高いタスク優先度であっても、タスク**B**は待ち状態になります。

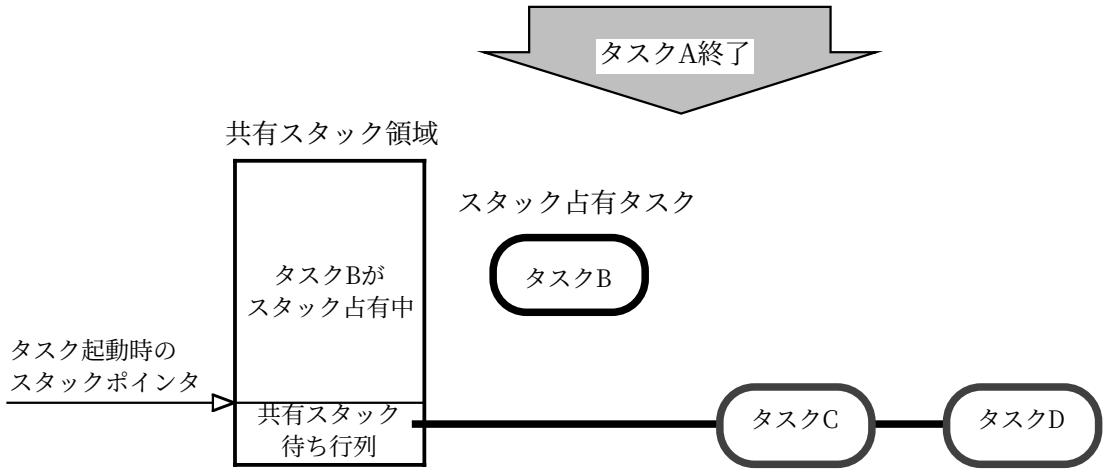
共有スタック待ち状態となったタスク**B**、**C**、**D**は、この共有スタック領域の待ち行列につながれます。共有スタック待ち行列は、優先度に関係なく**FIFO(First-In First-Out)**で管理され、起動要求が行なわれた順(タスク**B**、**C**、**D**)につながれます。

共有スタック領域は、共有スタック領域を使用(占有)していたタスクが終了(**ext_tsk, exd_tsk, ter_tsk**システムコール発行)することにより解放されます。そして、そのときに共有スタック待ち行列のタスクがあれば、共有スタック待ち行列の先頭タスクが共有スタック領域を使用(占有)し、実行可能状態に遷移します(起動要求の保留解除)。

図２－５の場合、タスク**A**が終了すると、次にタスク**B**が共有スタック領域を使用(占有)して、実行可能状態となります。



(a) 共有スタックをタスクAが占有中のスタック待ち行列の状態



(b)タスクAのext_tskシステムコール発行によるタスクA終了後
のスタック待ち行列の状態

図２－５　共有スタック機能の動作

なお、共有スタック領域を使用(占有)しているタスクが**wai_tsk**や**wai_flg**システムコールなどを発行して待ち状態となっても、共有スタック領域は解放されません。

タスクスタック領域の共有を指定せずに登録されたタスクは、共有スタック待ち状態となることはありえません。

図２－６に共有スタック機能を使用するタスクの状態遷移を示します。

共有スタック待ち状態のタスクに対して**sus_tsk(**または**isus_tsk)**システムコールを発行すると、共有スタック二重待ち状態に遷移します。この状態で共有スタックが解放されると、共有スタックが割り付けられて強制待ち状態に遷移します。

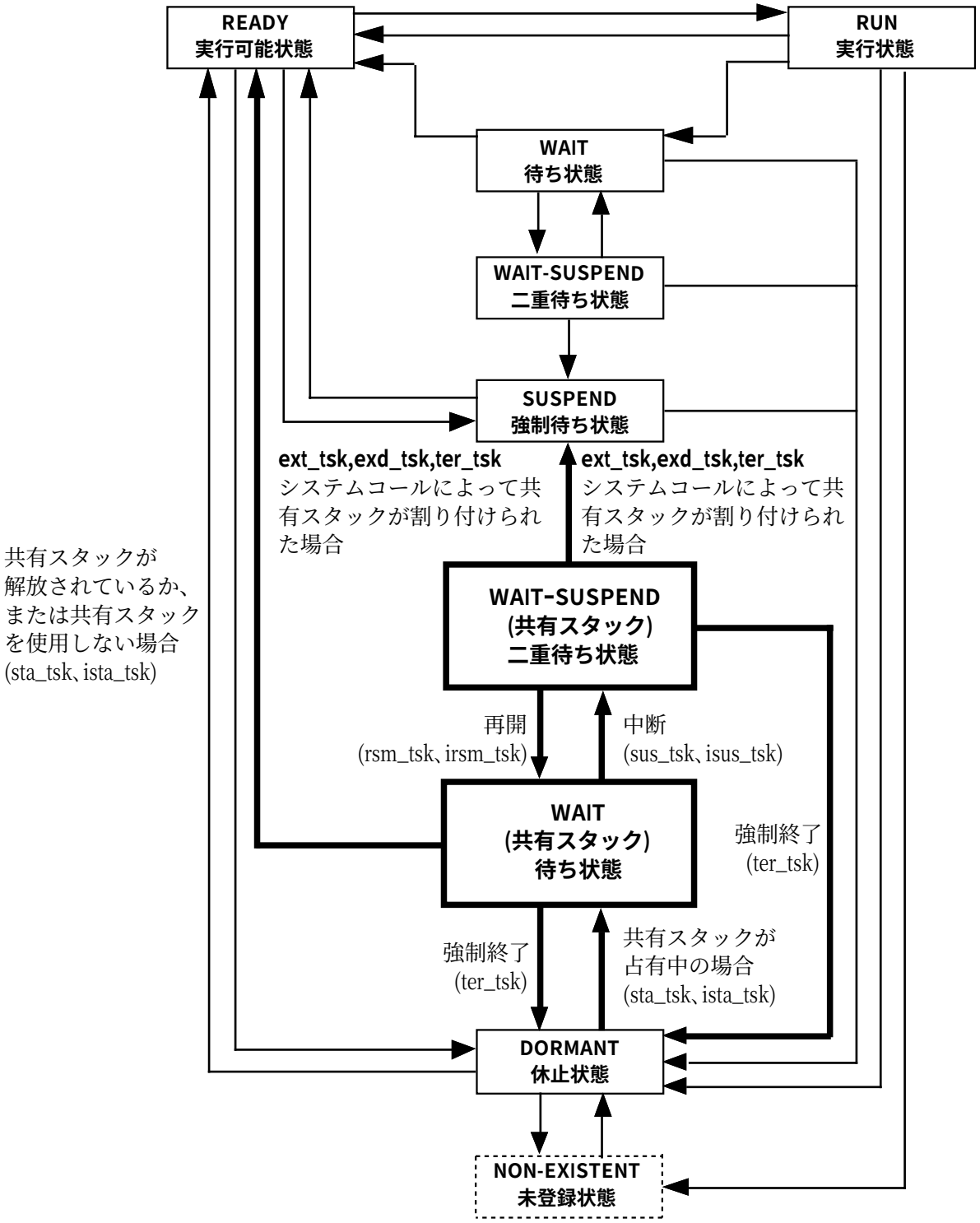


図２－６　共有スタック機能使用時のタスク状態遷移

２．３．６　タスクのスケジューリング

実行可能(**READY**)状態のタスクの内から1つのタスクが選ばれ、実行(**RUN**)状態になります。実行可能状態のタスクがない場合にはカーネルはアイドリング状態になり、割込みによりタスクが起動されるのを待ち続けます。実行可能状態のタスクが2つ以上ある場合、レディキューと呼ばれる**CPU**の割り付け待ち行列を用いてタスクの実行順序が決められます。この操作をスケジューリングといいます。

1からセットアップテーブルで定義する最大タスク優先度(**1** ～ **255**)のタスク優先度を使用できます。タスク優先度は、値が小さい方が高い優先度になります。レディキューはタスクの最大優先度の数だけ存在し、**FCFS(First Come First Service)**で管理されます。

図２－７にレディキューの概要を示します。**図２－７**では**4**種類のタスク優先度があり、それぞれに**3**つのタスクが属しています。

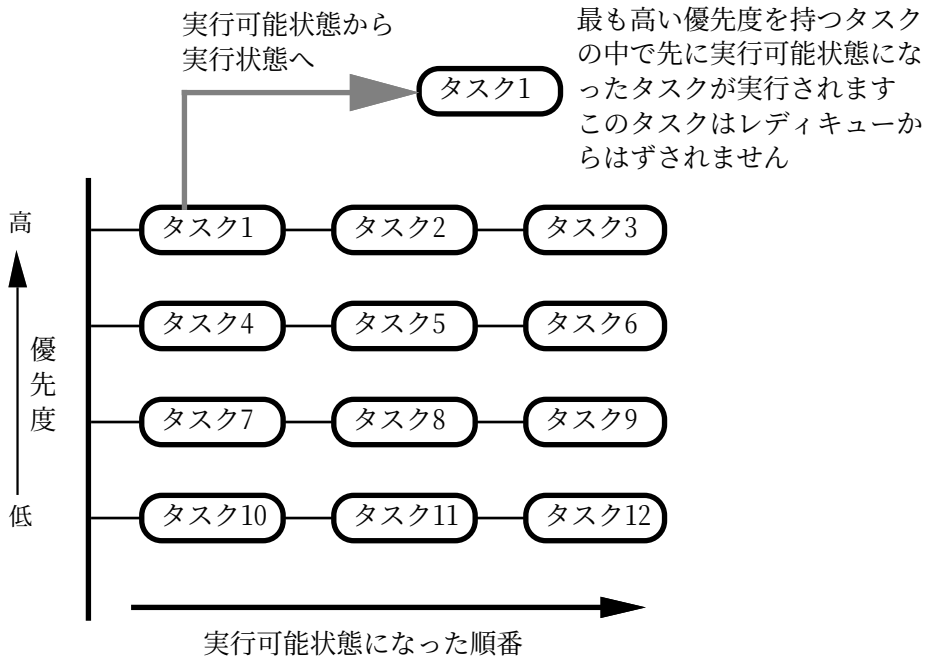


図２－７　レディキュー操作によるスケジューリング

スケジューリングには、大きく分けて次の2つがあります。

(1)標準のスケジューリング

実行可能状態で最も高い優先度のタスクに**CPU**が割り付けられます。

同一優先度のタスクが複数存在する場合には、その中で最も早く実行状態になったタスクに**CPU**が割り付けられます。**図２－７**ではタスク１が実行状態になり、**CPU**が割り付けられます。

タスク実行中に、より優先度の高い別のタスクが実行可能状態になった場合、元のタスクは実行可能状態に遷移し、制御はより優先度の高いタスクに渡されます。

実行状態のタスクはレディキューからはずされません。タスクは、実行状態または実行可能状態以外の状態になったときに、レディキューからはずされます。

(2)ラウンドロビンスケジューリング

同じ優先度を持つタスクへのサービス、すなわち**CPU**の割り付け時間を平均化するために、レディキューを回転させることがあります。このスケジューリング方法をラウンドロビンスケジューリングといいます。

ラウンドロビンスケジューリングは、時間管理機能と**irot_rdq**システムコールを組み合わせることで実現できます。

図２－８に**irot_rdq**システムコールによるレディキューの操作を示します。タイマ割込みハンドラで**irot_rdq**システムコールを発行するようにしておくと、一定時間毎にレディキューを回転させ、実行状態のタスクを次々に換えていくことができます。タイマ割込みハンドラについては、「**２．１０　時間**」を参照してください。

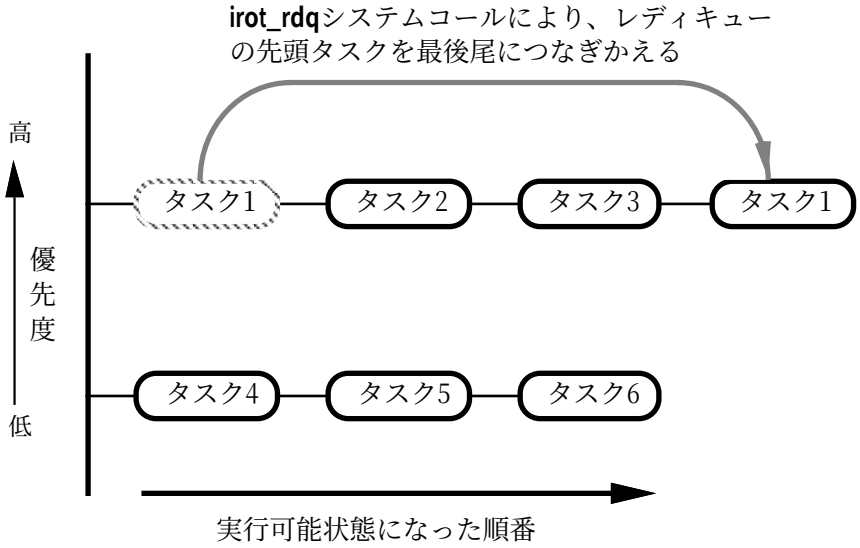


図２－８　irot_rdqシステムコールによるラウンドロビンスケジューリング

２．３．７　タスクの中断と再開

タスクの実行は、外部割込みや資源獲得待ちの要因で中断されます。中断要因が解除されると以前の状態に復帰します。

表２－３にタスク実行中断の要因と期間を示します。

表２－３　タスク実行中断の要因と期間

項番	中断の要因		中断の期間
1	自ら中断となる場合	**slp_tsk** **ｼｽﾃﾑｺｰﾙ**	(1)**wup_tsk, iwup_tsk**システムコールが発行されるまで (2)**rel_wai, irel_wai**システムコールが発行されるまで
		wai_tsk **ｼｽﾃﾑｺｰﾙ**	(1)**wup_tsk, iwup_tsk**システムコールが発行されるまで (2)指定したタイムアウト時間**(tmout)**が経過するまで (3)**rel_wai, irel_wai**システムコールが発行されるまで
		wai_flg **ｼｽﾃﾑｺｰﾙ**	(1)イベントフラグの待ち条件が成立するまで (2)**rel_wai, irel_wai**システムコールが発行されるまで
		wai_sem **ｼｽﾃﾑｺｰﾙ**	(1)セマフォで管理される資源を獲得できるまで (2)**rel_wai, irel_wai**システムコールが発行されるまで
		rcv_msg **ｼｽﾃﾑｺｰﾙ**	(1)メイルボックスにメッセージが送られるまで (2)**rel_wai, irel_wai**システムコールが発行されるまで
		get_blk **ｼｽﾃﾑｺｰﾙ**	(1)メモリプールからメモリブロックを獲得できるまで (2)**rel_wai, irel_wai**システムコールが発行されるまで
2	他のタスクから強制的に中断させられる場合	**sus_tsk** **ｼｽﾃﾑｺｰﾙ**	**rsm_tsk,irsm_tsk**システムコールが発行されるまで
3	非タスク部から強制的に中断させられる場合	**isus_tsk** **ｼｽﾃﾑｺｰﾙ**	**rsm_tsk,irsm_tsk**システムコールが発行されるまで
4	割込みにより中断させられる場合		割込みが発生し、**ret_int**システムコールが発行されるまで
5	共有スタックが占有されている場合	**sta_tsk** **ista_tsk** **ｼｽﾃﾑｺｰﾙ**	ext_tsk,exd_tsk,ter_tskシステムコールによって共有スタックが割り付けられるまで

２．３．８　タスクの終了

タスクの終了とは、タスクの処理を終了し休止**(DORMANT)**状態または未登録**(NON-EXISTENT)**状態になることをいいます。そのタスクを以降使用しないとき、またはそのタスクの**ID**に別のタスクを割り付けたいときには、未登録状態にしてください。

表２－４にタスクの終了要因を示します。

表２－４　タスクの終了要因

項番	分類	要因	終了後の状態
1	正常終了	**ext_tsk**システムコールを発行した	休止状態 **(DORMANT)**
2		**exd_tsk**システムコールを発行した	未登録状態 **(NON-EXISTENT)**
3	他からの強制終了	**ter_tsk**システムコールが発行された	休止状態 **(DORMANT)**

タスクの終了時、セマフォにより獲得した資源やメモリブロックは自動的に解放されるわけではありません。したがって、タスクは終了する前に、セマフォにより獲得・占有中の資源やメモリブロックを返却しなければなりません。また、他タスクを強制終了する場合、そのタスクがセマフォにより獲得・占有中の資源やメモリブロックを返却しなければなりません。

２．３．９　タスク実行中の割込みマスク

タスクは、**chg_ims**システムコールを用いてタスク実行中に割込みをマスクすることができます。割込みをマスクすると、タスクは非タスク部として動作し、タスクの切り換えを遅延させて**CPU**を占有することができます。

タスク実行中に割込みをマスクする場合、次の点に注意してください。

(1)割込みマスクの変更

割込みマスク値を0から0以外に変更（割込みをマスク）すると、システム状態はタスク部から非タスク部に移行します。逆に0以外から0に戻すと、非タスク部からタスク部に戻ります。タスク部に戻ったとき、割込みマスク中に自タスクより優先度の高いタスクが実行可能状態に遷移していると、タスク切り換えが発生し、優先度の高いタスクが実行状態となります。**図２－９**にこの様子を示します。

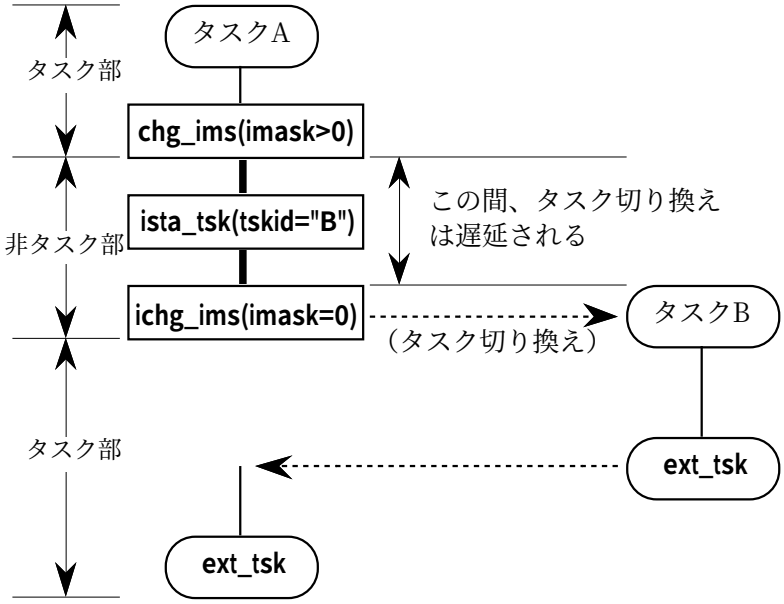


図２－９　タスク実行中の割込みマスク

(2)システムコールの制限

割込みマスク中のタスクはタスク独立部として動作するので、非タスク部用のシステムコールを使用します。タスク部用のシステムコールを使用するとコンテキストエラー**(E_CTX)**となります。

ただし、割込みハンドラではないので、**ret_int**システムコールは発行しないでください。

なお、割込みマスク値を、セットアップテーブルで定義する「カーネル割込みマスクレベル」を越える値に変更している間は、割込みマスク値を「カーネル割込みマスクレベル」より下げるための**ichg_ims**システムコールを除き、システムコールは発行しないでください。発行した場合、システムの正常な動作は保証されません。

２．４　　イベントフラグ

２．４．１　イベントフラグの概要

イベントフラグを用いることにより、複数の事象を組み合わせたタスク間の高速な同期処理ができます。

イベントフラグは事象の発生に応じた32ビットの集合です。ビット値"**1**"が事象発生あり、"**0**"が事象発生なしを示します。すなわち、1つのイベントフラグで32個の事象を管理できます。

タスクはイベントフラグに対し、指定したビットがセットされるのを待つ、すなわち指定した事象が発生するのを待つことができます。

イベントフラグを使用した同期処理は、フラグのセット／クリアのみで行なわれるので、高速に処理されます。

イベントフラグは最大1023個まで使用でき、1から1023のイベントフラグIDと呼ばれる番号で識別します。システムで使用するイベントフラグの数はセットアップテーブルに定義します。

イベントフラグの初期値は、H'00000000です。

1つのイベントフラグに対し、事象の発生を待つことができるタスクは1つだけです。

イベントフラグは、次のシステムコールにより操作します。

- **set_flg, iset_flg**システムコール(イベントフラグをセットする)
- **clr_flg,iclr_flg**システムコール(イベントフラグをクリアする)
- **wai_flg**システムコール(イベントフラグを待つ)
- **pol_flg,ipol_flg**システムコール(イベントフラグを得る)
- **flg_sts,iflg_sts**システムコール(イベントフラグ状態を参照する)

図２－１０にイベントフラグ機構の概要を示します。

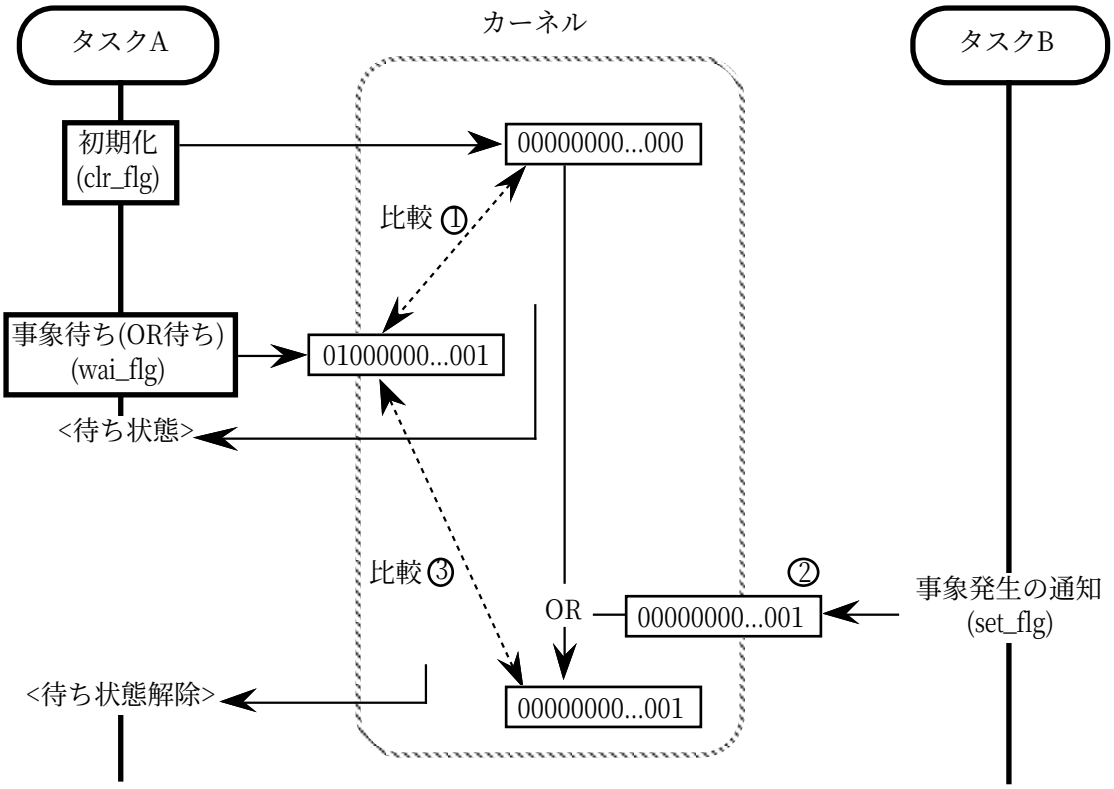


図２－１０　イベントフラグ機構の概要

(解説)

①タスク**A**は**wai_flg**システムコールを発行しますが、指定した事象が発生していないため、タスク**A**は事象発生の待ち状態になります。

②タスク**B**が**set_flg**システムコールを発行して事象発生の通知を行ない、イベントフラグの内容が変わります。

③事象発生を待っていたタスク**A**の待ち状態解除条件が成立したため、タスク**A**の待ち状態は解除されます。

２．４．２　事象発生の通知

事象発生の通知は、**set_flg, iset_flg**システムコールを発行して、発生した事象に対応したビットを**"1"**にした32ビットのビットパターンをイベントフラグに設定することにより行ないます。

set_flgシステムコールは、タスク部専用のシステムコール、**iset_flg**システムコールは、非タスク部専用のシステムコールです。

set_flg, iset_flgシステムコールを発行すると、イベントフラグの内容はパラメータで指定するビットパターンと論理和**(OR)**されます。そして、更新されたイベントフラグの内容から、**wai_flg**システムコールを発行して待たされているタスクの待ち状態解除条件が成立するかが調べられます。

待ちタスクの待ち状態解除条件が成立すると、待ち状態が解除されます。待ち状態を解除されたタスクには、**wai_flg**システムコールのリターンパラメータとして、待ち状態解除時のイベントフラグの内容が返されます。

２．４．３　イベントフラグのクリア

イベントフラグのビットのクリアには、**clr_flg**または**iclr_flg**システムコールを発行します。

イベントフラグのビットのクリアは、イベントフラグの内容をパラメータで指定するビットパターンと論理積**(AND)**することで行なわれます。

２．４．４　事象発生の待機、取得

wai_flgシステムコールを用いて、指定した事象が発生するまで実行を中断(待ち状態)することができます。すなわち、**wai_flg**システムコール発行時、指定した事象が発生していなければ、事象が発生するまで待ち状態となります。指定した事象がすでに発生していれば、待ち状態にはならずに処理を続行します。

pol_flgまたは**ipol_flg**システムコールを用いて、指定した事象が発生しているかを知ることができます。指定した事象がすでに発生していれば正常終了し、指定した事象が発生してなければエラーコードとしてポーリング失敗エラー**(E_PLFAIL)**が返されます。

wai_flg,**pol_flg,ipol_flg**システムコールで指定する待ちモードにより、"**AND**待ち"または"**OR**待ち"を選択することができます。

"**AND**待ち"とは、待ちビットパターンの"**1**"で指定したビットに対するすべての事象が発生したときに、待ち状態が解除されるモードです。"**OR**待ち"とは、待ちビットパターンの"**1**"で指定したビットに対する事象が少なくとも1つ発生したときに、待ち状態が解除されるモードです。

"**AND**待ち"と"**OR**待ち"の条件をまとめると次のようになります。

・"**AND**待ち"の解除条件

(イベントフラグ)　∩　(待ちビットパターン)　＝　(待ちビットパターン)

・"**OR**待ち"の解除条件

(イベントフラグ)　∩　(待ちビットパターン)　≠　0

待ち状態が解除されると、そのときのイベントフラグの内容が返されます。

また、"**AND**待ち"または"**OR**待ち"と共に、待ちモードに"クリア指定"を組み合わせて指定できます。"クリア指定"を行なうと、待ち条件が成立したときにイベントフラグが0クリアされ、クリアされる直前のイベントフラグの内容が返されます。

２．４．５　イベントフラグ状態の参照

現在のイベントフラグの状態を参照するには、**flg_sts**または**iflg_sts**システムコールを発行します。これらのシステムコールの発行により、リターンパラメータとして以下の情報が返されます。

・イベントフラグのビットパターン

・イベントフラグの待ちタスク**ID**

２．５　セマフォ

２．５．１　セマフォの概要

タスク実行のために必要な各種の要素を「資源」と呼びます。「資源」としては、**I/O**などのハードウェアやタスク間で共有するメモリなどが考えられます。

セマフォを用いることにより、このような資源の排他制御を行なうことができます。

セマフォは、符号なしの16ビットカウンタ(セマフォカウンタ)を持っており、これを実際の資源の数に対応させて使用します。

実際にセマフォを用いて資源の排他制御を行なうには、

(1)ユーザシステムの初期処理において、資源の数を使用するセマフォのセマフォカウンタと一致させます**(sig_sem, isig_sem**システムコールを使用**)**。

(2)wai_semまたは**preq_sem,ipreq_sem**システムコールを用いてセマフォカウントを獲得したタスクが、資源の使用権を得ます。すなわち、セマフォカウントの獲得とは、資源の獲得と同じ意味になります。

(3)資源の使用を終えたタスクは、その資源を他のタスクが使用できるようにするため、**sig_sem, isig_sem**システムコールを用いてセマフォカウント**(**資源と同じ意味**)**をカーネルに返却します。

という論理で、システムを設計します。

このように設計することで、資源の排他的な制御を簡単に行なうことができます。

セマフォは最大**1023**個まで使用でき、**1**から**1023**のセマフォ**ID**と呼ばれる番号で識別します。すなわち、最大**1023**種類の資源を扱うことができます。システムで使用するセマフォの数は、セットアップテーブルに定義します。

セマフォカウンタの初期値は**1**です。セマフォカウンタと資源の数を一致させるには、ユーザシステムの初期処理において、必要回数**sig_sem, isig_sem**システムコールを発行してセマフォカウンタを増やす必要があります。

セマフォに対するタスクの待ち行列は、**FIFO(First-In First-Out)**で管理されます。

セマフォカウントを返却せずにタスクが終了した場合は、そのセマフォと実際の資源との間に矛盾が生じてしまうので注意してください。

セマフォは、次のシステムコールを用いて操作します。

・**sig_sem, isig_sem**システムコール(セマフォに対する信号操作：**V**命令)

・**wai_sem**システムコール(セマフォに対する待ち操作：**P**命令)

・**preq_sem,ipreq_sem**システムコール(セマフォ資源を得る)

・**sem_sts,isem_sts**システムコール(セマフォ状態を参照する)

図２－１１にセマフォ機構の概要を示します。

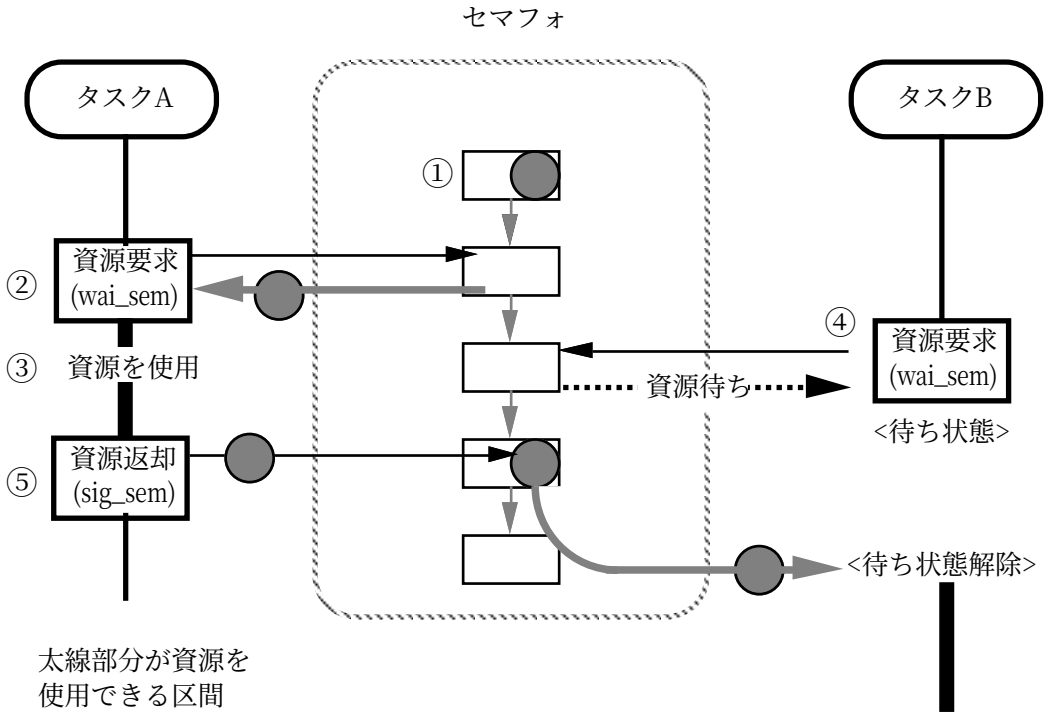


図２－１１　セマフォ機構の概要

(解説)

①はじめに資源が1つあります(セマフォカウンタが1)。

②タスクAが**wai_sem**システムコールを発行して資源を要求しました。セマフォには資源があるので、タスクAは資源を獲得します。この結果、セマフォの資源が**0**になります(セマフォカウンタが**0**に)。

③タスクAは、実際に資源を占有使用します。

④タスクBが**wai_sem**システムコールを発行して資源を要求しました。しかし、このときの資源は0のため、タスクBは資源を獲得できずにセマフォに対するタスクの待ち行列につながれます。

⑤タスクAが**sig_sem**システムコールを発行して資源を解放しました。この結果、解放された資源がタスクBに割り付けられ、タスクBの待ち状態は解除されます。以降、タスクBは実際の資源を使用することができます。

２．５．２　資源の占有要求

資源の占有要求には、**wai_sem**システムコール**(P**命令**)**を使用します。**wai_sem**システムコールは、指定した資源を占有できるまで実行を中断(待ち状態)することができます。すなわち、**wai_sem**システムコール発行時、セマフォカウンタが**0**であれば**(**資源が残っていない**)**、資源が他のタスクから解放されるまでタスクはセマフォに対するタスクの待ち行列につながれます。セマフォカウンタが**0**でなければ、**1**減算され、タスクは資源を割り付けられ、そのまま処理を続けます。

資源占有要求の待ちを行なわない(ポーリング)場合は、**preq_sem**または**ipreq_sem**システムコールを用います。**preq_sem,ipreq_sem**システムコールでは、指定した資源の占有が可能であれば正常終了し、不可能であればエラーコードとしてポーリング失敗エラー**(E_PLFAIL)**が返されます。

wai_sem,**preq_sem,ipreq_sem**システムコールは**1**回の発行で**1**つの資源を占有します。

２．５．３　資源の解放

wai_sem, **preq_sem,ipreq_sem**システムコールで占有した資源の解放は、**sig_sem**, **isig_sem**システムコール**(V**命令**)**を用います。

sig_semシステムコールは、タスク部専用のシステムコール、**isig_sem**システムコールは、非タスク部専用のシステムコールです。

sig_sem, **isig_sem**システムコールの発行時にセマフォに対するタスクの待ち行列に待ちタスクが存在する場合は、待ち行列の先頭のタスクは**P**命令同様に資源が割り付けられ、待ち行列からはずされます。

待ちタスクが存在しない場合は、セマフォカウンタが最大値**(65535)**を越えなければ、**1**だけ加算されます。最大値を越えると、エラーコードとしてオーバフローエラー**(E_QOVR)**が返されます。

２．５．４　セマフォ状態の参照

現在のセマフォの状態を参照するには、**sem_sts**または**isem_sts**システムコールを発行します。これらのリターンパラメータとして以下の情報が返されます。

・セマフォカウンタの値

・セマフォに対するタスクの待ち行列の先頭タスクのタスクID

２．６　メイルボックス

２．６．１　メイルボックスの概要

メイルボックスを用いることにより、タスク間でのメッセージと呼ばれるデータの受け渡しができます。

メッセージを送信するタスクはメイルボックスへメッセージを送り、メッセージを受信するタスクはメイルボックスからメッセージを受け取ります。

メイルボックスに送られたメッセージは、メッセージ到着を待つタスクが存在すれば、その待ち行列の先頭タスクに渡され、メッセージ到着を待つタスクが存在しなければ、メッセージの行列につながれます。

メッセージを受信しようとするタスクは、メイルボックスにメッセージが存在すれば、メッセージの行列の先頭メッセージを受け取り、メッセージが存在しなければ、メッセージ到着を待つタスクの待ち行列につながれます。

メイルボックスは最大**1023**個まで使用でき、**1**から**1023**のメイルボックス**ID**と呼ばれる番号で識別します。システムで使用するメイルボックスの数はセットアップテーブルに定義します。

メッセージの行列およびメッセージ到着を待つタスクの待ち行列は、共に**FIFO(First-In First-Out)**で管理されます。

メイルボックスは、以下のシステムコールにより操作します。

- **snd_msg、isnd_msg**システムコール(メイルボックスへ送信する)
- **rcv_msg**システムコール(メイルボックスからの受信を待つ)
- **prcv_msg,iprcv_msg**システムコール(メイルボックスから受信する)
- **mbx_sts,imbx_sts**システムコール(メイルボックス状態を参照する)

図２－１２にメイルボックスの概要を示します。

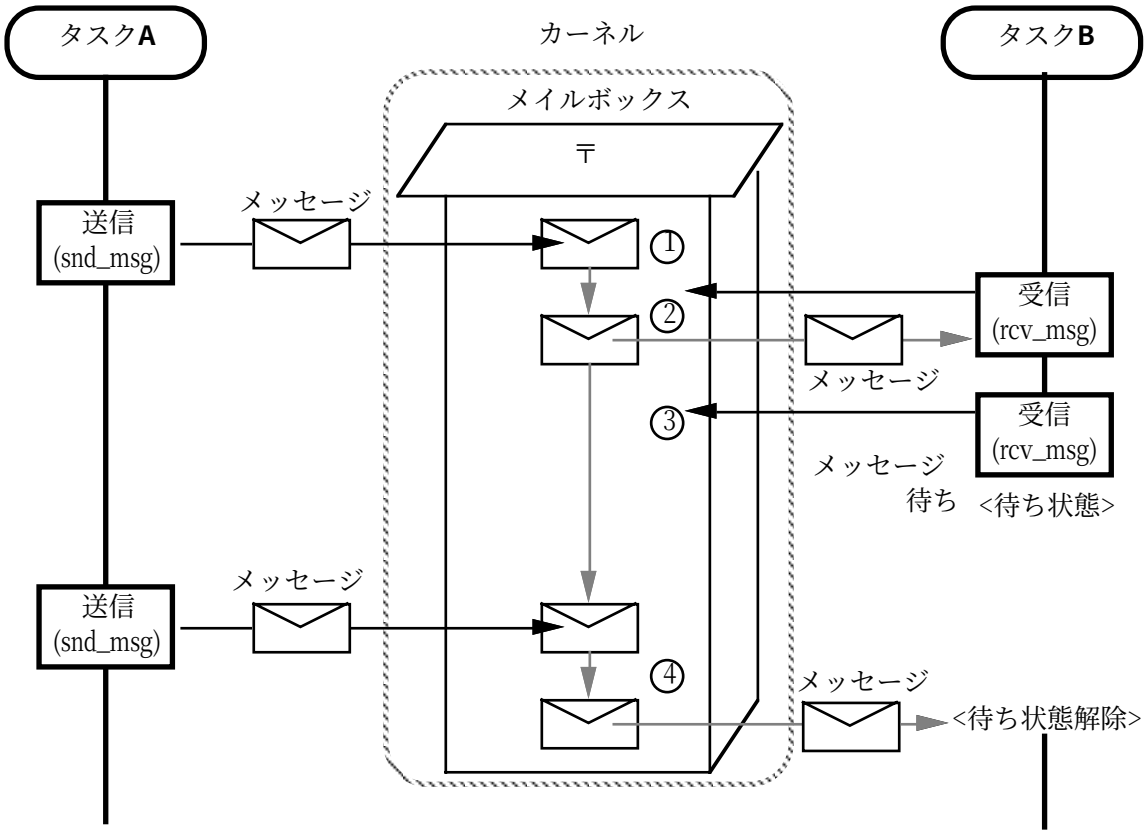


図２－１２　メイルボックス機構の概要

(解説)

①タスクAが**snd_msg**システムコールを発行してメッセージの送信を行ないます。この結果、メイルボックスに1通のメッセージが蓄えられます。

②タスクBが**rcv_msg**システムコールを発行して受信要求を行なうと、蓄えられたメッセージがタスクBに渡されます。

③タスクBがさらに**rcv_msg**システムコールを発行して受信要求を行なうと、メイルボックスにメッセージが存在しないため、タスクBはメッセージの待ち状態になります。

④タスクAが**snd_msg**システムコールを発行してメッセージを送信したので、メッセージの受信要求を行なっていたタスクBにメッセージが渡され、タスクBはメッセージの待ち状態から解除されて実行可能状態になります。

２．６．２　メッセージの送信

メイルボックスへのメッセージの送信は、**snd_msg, isnd_msg**システムコールを用います。

snd_msgシステムコールは、タスク部専用のシステムコール、 **isnd_msg**システムコールは、非タスク部専用のシステムコールです。

snd_msg, isnd_msgシステムコールの発行時にメイルボックスにメッセージ到着を待つタスクが存在する場合は、待ち行列の先頭タスクにメッセージが渡され、タスクは待ち行列からはずされます。

メッセージ到着を待つタスクが存在しない場合は、送られたメッセージは、メッセージの行列につながれます。

２．６．３　メッセージの受信

メイルボックスからメッセージを受信するには、**rcv_msg**システムコールを発行します。**rcv_msg**システムコールは、メッセージを受信できるまで実行を中断(待ち状態)することができます。すなわち、**rcv_msg**システムコール発行時、メイルボックスにメッセージが存在しなければ、タスクはメッセージ到着待ちの待ち行列につながれます。メッセージが存在する場合は、メッセージの行列の先頭メッセージがタスクに返され、そのまま処理を続けます。

メッセージ受信の待ちを行なわない(ポーリング)場合は、**prcv_msg**または**iprcv_msg**システムコールを用います。これらのシステムコールでは、メイルボックスにメッセージが存在するときは正常終了し、メッセージが存在しないときはエラーコードとしてポーリング失敗エラー**(E_PLFAIL)**が返されます。

２．６．４　メイルボックス状態の参照

現在のメイルボックスの状態を参照するには、**mbx_sts**または**imbx_sts**システムコールを発行します。これらのシステムコールの発行により、リターンパラメータとして以下の情報が返されます。

・メッセージの行列の先頭メッセージのアドレス

・メッセージ到着を待つタスクの待ち行列の先頭タスクのタスク**ID**

２．６．５　メッセージフォーマット

図２－１３にメッセージフォーマットを示します。メッセージは、**RAM**上に作成しなければなりません。

メッセージの先頭4バイトは、**OS**によりメッセージが管理されるための領域です。ユーザがメッセージ領域として自由に使用できる部分は、メッセージの先頭アドレス+4バイト目からです。メッセージ領域の大きさはユーザが定義します。

OS管理領域はメッセージ送信時には必ず0を設定してください。**OS**管理領域が0でないメッセージを送信しようとすると、エラーコードとして不正メッセージ形式エラー**(E_ILMSG)**を返されます。

メッセージ送信時、**OS**管理領域にはメイルボックスのメッセージの行列につながれていることを示すデータが書き込まれます。メッセージが受信されると、**OS**管理領域が0クリアされ、受信タスクにメッセージの先頭アドレスが渡されます。

したがって、同一メッセージ領域を繰り返して使用する場合は、最初の送信時に**OS**管理領域を0クリアするだけで充分です。

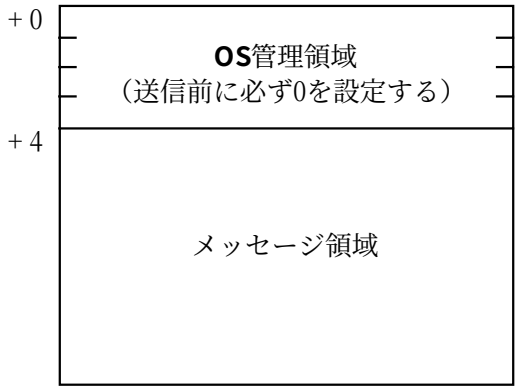


図２－１３　メッセージフォーマット

２．６．６　メッセージ送受信の注意事項

メッセージの送受信は、データそのものが転送されるわけではなく、そのメッセージの先頭アドレスが渡されるだけです。このためメッセージを送信後、受信する前にメッセージ内容を破壊すると、受信したタスクは破壊されたデータを読むことになります。

また、メッセージの先頭4バイトは**OS**が管理する領域です。メッセージ送信前に、この領域に必ず0を設定してください。

メッセージを送信後、受信する前にこの領域を破壊すると、システムの正常な動作は保証されません。

メッセージを使用した具体例としては、**「付録Ａ．コンソールドライバの例題」**を参照してください。

２．７　割込み

２．７．１　割込みの概要

割込みを使用するには、割込みハンドラの作成とベクタテーブルへの登録が必要です。
割込みが発生すると、カーネルの例外サービスルーチンを介し、割込みハンドラへ制御が渡されます。
なお、アドレスエラーやソフトウェア割込みなどの例外は割込みではありません。
割込み管理として次のシステムコールが用意されています。

・**ret_int**システムコール**(**割込みハンドラからの復帰**)**
・**chg_ims,ichg_ims**システムコール**(**割込みマスクの変更**)**
・**ims_sts,iims_sts**システムコール**(**割込みマスクの参照**)**

図２－１４に割込みハンドラへの制御移行を示します。

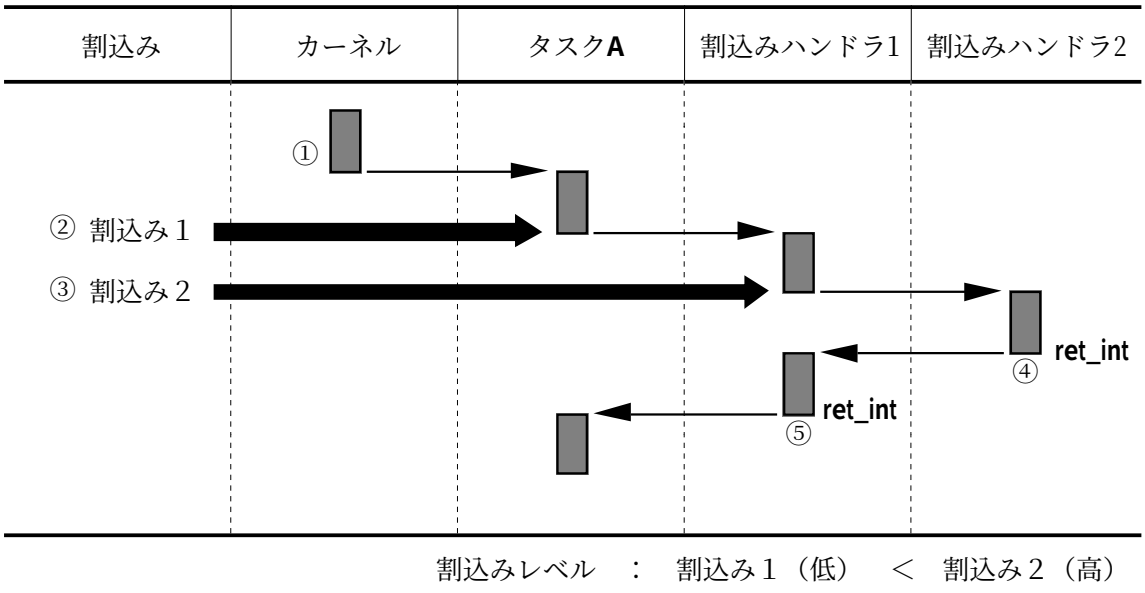


図２－１４　割込みハンドラへの制御移行

（解説）

①タスクAの起動要求にしたがい、タスクAが起動されます。
②タスクAの処理中に割込み１が発生すると、タスクAの実行が中断されて制御が割込みハンドラ１に移行します。
③割込みハンドラ１の処理中に割込みレベルが高い割込み２が発生すると、割込みハンドラ１の実行が中断されて制御が割込みハンドラ２に移行します。
④**ret_int**システムコールを発行して割込みハンドラ２を終了すると、中断されていた割込みハンドラ１の処理が再開されます。
⑤**ret_int**システムコールを発行して割込みハンドラ１を終了すると、中断されていたタスクAの処理が再開されます。

タスク実行中に発生した割込みハンドラが終了したとき、その時点で実行可能な最も優先度の高いタスクに制御が渡されますので、割込みが発生したタスクへは復帰しない場合があります。

２．７．２　割込みハンドラの作成

割込みが発生すると、カーネルの例外サービスルーチンを介して、ユーザの割込みハンドラに制御が渡されます。

割込みは非同期に発生するため、割込みハンドラでは割込み発生前のレジスタリソースを保証する必要があります。

割込みハンドラがスタックをそのまま使用すると、割込み発生前のプログラムとして確保したスタックがオーバフローする危険があるため、割込みハンドラは専用のスタックを用いる必要があります。

また、割込みハンドラ起動時は、**SR**の割込みマスクは割込み発生前のままで、かつ例外がマスクされた状態になっています。したがって、より高位の割込みを受け付け可能とするためには、割込みマスクを自割込みレベルに設定し例外マスクを解除するよう、**SR**を設定する必要があります。この処理は**chg_ims,ichg_ims**システムコールを使用せず、**SR**レジスタを直接操作して行なってください。なお、Ｃ言語で割込みハンドラを記述する場合は、Ｃコンパイラの組み込み関数**set_cr()**を利用してください。

割込みハンドラは通常以下の手順で作成してください。

(1)割込みハンドラで使用するレジスタの内容退避

・スタックポインタの保存

・割込みハンドラ専用スタック領域にスタックポインタを変更

（割込みハンドラでいっさいスタックを使用しない場合は不要）

・レジスタの内容保存

(2)割込みマスクレベル設定、例外マスク解除（ここでは**chg_ims,ichg_ims**システムコールは使用しないでください）

(3)割込み処理

(4)割込みハンドラで使用したレジスタの内容回復

・レジスタの内容復帰

・スタックポインタの復帰

（割込みハンドラでいっさいスタックを使用しない場合は不要）

(5)ret_intシステムコールの発行

（**RTE**命令による復帰はできません）

なお、割込みハンドラ内でシステムコールを発行する場合、例外マスクを解除してから行なってください。

例外マスク状態でシステムコールを発行した場合、システムの動作は保証されません。

Ｃ言語で割込みハンドラを記述する場合、Ｃコンパイラが自動的にレジスタの退避と回復を行なう機械語を生成します。詳細は**「４．４　C言語による割込みハンドラの記述方法」**を参照してください。

割込みハンドラ実行中はタスクの切り換えが遅延されます。**ret_int**システムコールにより制御が渡されるとき、遅延されていたタスク切り換えが再開されます。

割込みハンドラは、**表２－５**の処理条件を参考に作成してください。

表２－５　割込みハンドラの処理条件

項番	項　　目		内　　容
1	システム状態		タスク独立部で実行します
2	SR	モード(MD)	割込みハンドラ起動時、特権モード(1)に設定されています
3		レジスタバンク(RB)	・割込みハンドラ起動時、バンク0(0)が設定されています ・バンク1(1)のレジスタを使用する場合は例外マスク状態で行ってください
4		例外マスク(BL)	割込みハンドラ起動時、例外はマスク(1)されています システムのスループット向上のためには、できるだけはやくマスクを解除してください
5		割込みマスク(I3～I0)	割込みハンドラ起動時、割込み発生前の割込みマスクレベルに設定されていますので、発生割込みの割込みマスクレベルに設定してください
6	使用できるレジスタ		・使用する汎用レジスタ、プロシジャレジスタ、積和レジスタ、グローバルレジスタを割込みハンドラ開始時にスタック領域へ退避し、終了時に復帰してください* ・VBRは変更しないでください
7	SP(R15)		割込み発生元へ復帰する際は、割込みハンドラ起動時と同じ値にしてください*
8	使用できるスタック領域		割込みレベルごとに割込みハンドラ用のスタック領域を確保し、割込みハンドラ起動時にスタックポインタを切り替えてください
9	使用できるシステムコール		・例外マスク解除状態で非タスク部用システムコールが使用できます ・例外マスク状態では使用できません
10	復帰方法		ret_intシステムコールにより復帰してください （RTE命令による復帰はできません）

【注】* C言語記述の場合、考慮する必要はありません。詳細は**「４．４　C言語による割込みハンドラの記述方法」**を参照してください。

２．７．３　割込みハンドラの登録

作成した割込みハンドラの先頭アドレスを、ベクタテーブルの該当番号位置に登録することにより割込みが可能になります。**図２－１５**にベクタテーブルと割込みハンドラとの関連を示します。

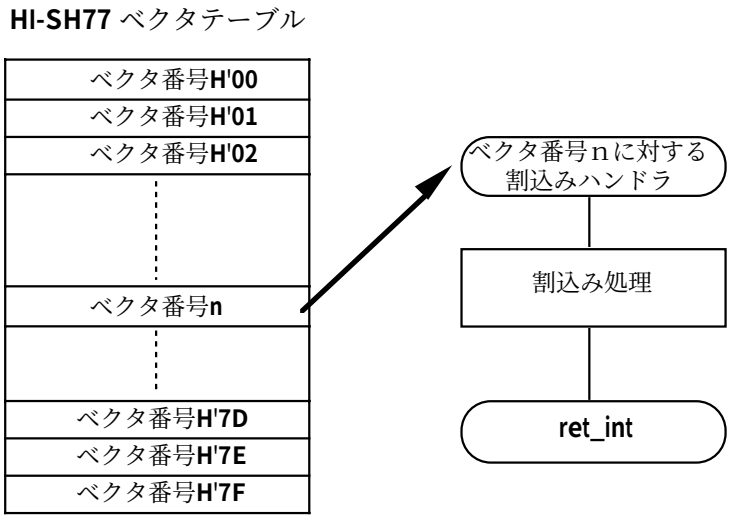


図２－１５　ベクタテーブルと割込みハンドラとの関連

２．７．４　割込みマスクレベルの変更

例外マスク解除状態の場合、**chg_ims**または**ichg_ims**システムコールを使用して割込みのマスクレベルを変更することができます。

割込みハンドラ内で**ichg_ims**システムコールを使用するときは、割込みマスクレベルを割込みハンドラの先頭で設定したレベルより下げないでください。

割込みマスクレベルを下げた場合、システムの正常な動作は保証されません。

なお、タスク実行中に割込みマスクを変更すると、タスクの切り換えを遅延させて**CPU**を占有することができます。詳しくは、**「２．３．９　タスク実行中の割込みマスク」**を参照してください。

【参考】**SH**シリーズ**C**コンパイラの組込み関数**set_imask**との関係

割込みマスクを変更する場合は、**chg_ims,ichg_ims**システムコールを用いてください。**SH**シリーズ**C**コンパイラでは、割込みマスクを変更する組込み関数**(set_imask**関数**)**を持っていますが、基本的には、これを使用しないでください。これは、主に**ITRON**仕様としての教育的配慮からです。

ただし、**chg_ims,ichg_ims**システムコールの処理時間が問題となる場合は、タスク部実行中に割込みマスクを**0**以外に変更する場合、および割込みマスクを**0**に戻す場合を除き、**set_imask**関数を用いても、機能的には何ら問題はありません。**set_imask**関数では、直接ステータスレジスタ**(SR)**を操作するので、**chg_ims,ichg_ims**システムコールと比べ、極めて高速に割込みマスクを変更することができます。

なお、**set_imask**関数は特権モードでのみ使用できます。ユーザモードで割込みマスクを変更する場合は、**chg_ims,ichg_ims**システムコールを用いてください。

２．７．５　割込みマスクレベルの参照

現在の割込みマスクレベルを参照するには、**ims_sts**または**iims_sts**システムコールを使用します。

これらのシステムコールの発行により、リターンパラメータとして現在の割込みマスクレベルが返されます。

２．７．６　未定義割込み

ユーザシステムで使用しない割込みは、未定義割込み用の割込みハンドラをユーザ自身で作成し、ベクタテーブルに登録してください。

サンプルプログラムとして提供する**hi_intdwn** 関数は、割込みマスクレベルを最高(15)に設定しています。そして、未定義割込みのベクタ番号はこの関数を呼び出すようにしています。

このように、未定義割込み用の割込みハンドラでは、割込みマスクレベルを最高(15)に設定してシステムの異常終了処理を行なってください。

２．７．７　割込みの注意事項

(1)ユーザは、割込みハンドラを自由に作成することができますが、割込みハンドラの実行時間が長すぎるとシステム全体のスループットを低下させることになります。システムの応答性に大きな影響を与えますので、注意して作成してください。

(2)割込みハンドラでは、必ず以下の処理を行なってください。

(a)割込み要求マスクレベルを割込みマスクビットに設定する。

(b)例外マスクを解除する。

(a)は必ず例外マスクの状態で行なってください。

また、割込みハンドラ実行中は割込みマスクレベルを割込みハンドラの先頭で設定したレベルより下げないでください。

割込みマスクレベルを割込みハンドラの先頭で設定したレベルより下げた場合、システムの正常な動作は保証されません。

また、例外マスク状態ではシステムコールを発行しないでください。

例外マスク状態でシステムコールを発行した場合、システムの正常な動作は保証されません。

(3)セットアップテーブルへの定義によりカーネルの割込みマスクレベルを決めることができます。このカーネル割込みマスクレベルより高いレベルの割込みハンドラでは、**ret_int**システムコールを除いて、システムコールを発行できません。また、**NMI(Non Maskable Interrupt)**の割込みハンドラでも、発行できません。

カーネル割込みマスクレベルより高いレベルの割込みハンドラで発行した場合、システムの正常な動作は保証されません。

また、カーネル割込みマスクレベル以下の割込みハンドラで割込みマスクレベルをカーネル割込みマスクレベルより高く変更している間は、**ichg_ims**システムコールを用いてカーネル割り込みマスクレベル以下に下げる場合と**ret_int**システムコールを除いて、システムコールを発行できません。

割込みマスクレベルをカーネル割込みマスクレベルより高く変更している間で発行した場合、システムの正常な動作は保証されません。

(4)割込みハンドラから復帰する場合は、**ret_int**システムコールを使用してください。**RTE**命令による復帰はできません。

RTE命令による復帰を行なった場合、システムの正常な動作は保証されません。

２．８　例　外

アドレスエラーやソフトウェア割込みなどの例外が発生した場合、カーネルの例外サービスルーチンを介し、ベクタテーブルまたはトラップベクタテーブルに登録されている例外処理プログラムの先頭アドレスへ制御が渡されます。

例外処理は、例外発生前に実行していたタスクや割込みハンドラの一部として動作します。

例外処理起動時は特権モード、レジスタバンク０に設定されています。バンク１のレジスタを使用する場合は例外マスク状態で行なってください。また、例外はマスクされています。システムのスループットを向上させるためには、すみやかに例外マスクを解除する必要があります。割込みマスクは例外発生前の割込みマスクレベルに設定されています。

例外発生の可能性があるときは、例外の発生するタスクや割込みハンドラのスタック領域に例外処理プログラムが使用するスタックサイズを加算して確保してください。スタック領域の計算方法については、『**HI-SH77構築マニュアル**』を参照してください。なお、例外処理プログラムは、再入される可能性があるので、例外処理プログラム専用にスタックを持つことはできません。

例外処理プログラム内でシステムコールを発行する場合、例外マスクを解除してから行なってください。

例外マスク状態でシステムコールを発行した場合、システムの正常な動作は保証されません。

例外処理プログラムから復帰する場合は、**ret_exc**システムコールを使用してください。**ret_exc**システムコール発行時のスタックポインタは、例外発生時と同じである必要があります。この時のスタックの状態は、**図２－１６**に示すようになっています。必要に応じて、正しい戻り先アドレスをスタックに格納した上で**ret_exc**システムコールを発行してください。なお、**RTE**命令による復帰はできません。

RTE命令による復帰を行なった場合、システムの正常な動作は保証されません。

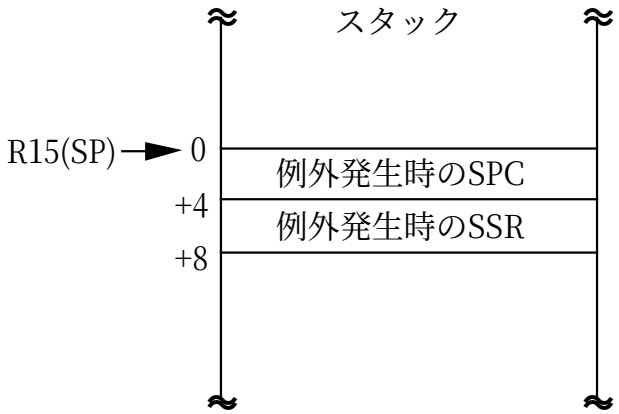


図２－１６　ret_excシステムコール発行時のスタック

２．９　メモリプール

２．９．１　メモリプールの概要

メモリプールを用いることにより、ユーザの未使用メモリ空間を動的に管理することができます。

メモリプールは、メモリブロックと呼ぶ複数の固定長メモリ領域から構成されています。タスクは、このメモリブロックをメモリプールから獲得して使用することができます。

メモリブロックの獲得を要求したタスクは、メモリプールに空きメモリブロックが存在すれば受け取り、空きメモリブロックが存在しなければ、メモリブロックの獲得を待つタスクの待ち行列につながれます。

メモリプールは最大1023個まで使用でき、1から1023のメモリプール**ID**と呼ばれる番号で識別します。システムで使用するメモリプールの数はセットアップテーブルに定義します。

メモリブロックの獲得を待つタスクの待ち行列は、**FIFO(First-In First-Out)**で管理されます。

獲得したメモリブロックを返却せずにタスクが終了した場合は、そのメモリブロックを以降使用できなくなりますので注意してください。

メモリプールは、次のシステムコールにより操作します。

- **get_blk**(メモリブロックを獲得待ちを行なう)
- **rel_blk,irel_blk**(メモリブロックを返却する)
- **pget_blk,ipget_blk**(メモリブロックを獲得する)
- **mpl_sts,impl_sts**(メモリプール状態を参照する)

図２－１７にメモリプール機構の概要を示します。

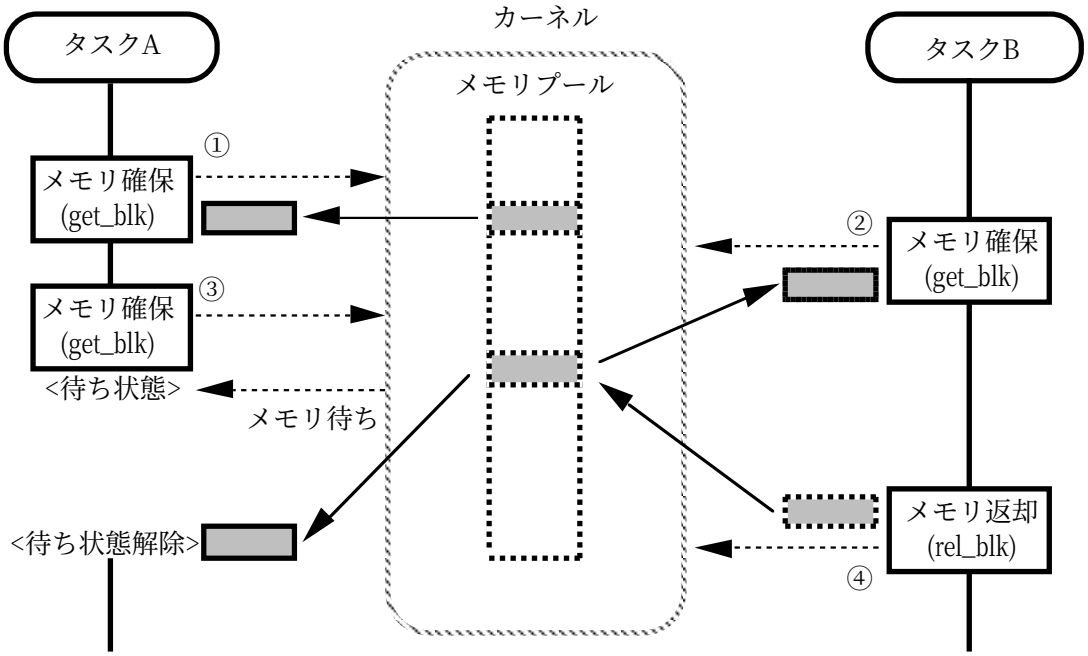


図２－１７　メモリプール機構の概要

（解説）

①タスクAが**get_blk**システムコールを発行して、メモリプールからメモリブロックを獲得します。

②タスクBが**get_blk**システムコールを発行して、メモリプールからメモリブロックを獲得します。

③ふたたび、タスクAが**get_blk**システムコールを発行して、メモリプールからメモリブロックを獲得しようとしますが、空きメモリブロックが存在しないため待ち状態になります。

④タスクBが獲得しているメモリブロックを**rel_blk**システムコールを発行して返却すると、タスクAに割り付けられ、待ち状態が解除されます。

２．９．２　メモリブロックの獲得

メモリブロックの獲得は、**get_blk**システムコールを発行します。**get_blk**システムコールは、メモリブロックを獲得できるまで実行を中断(待ち状態)することができます。すなわち、**get_blk**システムコール発行時、空きメモリブロックが存在しなければ、メモリブロックが返却されるまでタスクはメモリブロックの獲得を待つタスクの待ち行列につながれます。空きメモリブロックが存在すれば、タスクに割り付けられ、そのまま処理を続けます。

メモリブロックの獲得の待ちを行なわない**(**ポーリング**)**場合は、**pget_blk**または**ipget_blk**システムコールを用います。これらのシステムコールでは、メモリブロック獲得が可能であれば正常終了し、不可能であればエラーコードとしてポーリング失敗エラー**(E_PLFAIL)**が返されます。

get_blk, pget_blk,ipget_blkシステムコールは、**1**回の発行で**1**つのメモリブロックを獲得します。

２．９．３　メモリブロックの返却

get_blk, pget_blk,ipget_blkシステムコールで獲得したメモリブロックの返却は、**rel_blk**システムコールを用います。

rel_blk,irel_blkシステムコール発行時、メモリブロックの獲得を待つタスクの待ち行列にタスクが存在する場合は、待ち行列の先頭タスクにメモリブロックが割り付けられ、タスクは待ち行列からはずされます。待ちタスクが存在しない場合は、返却したメモリブロックは空きメモリブロックとして管理されます。

２．９．４　メモリプール状態の参照

現在のメモリプール状態の参照は、**mpl_sts**または**impl_sts**システムコールを発行します。これらのシステムコールの発行により、リターンパラメータとして以下の情報が返されます。

・空きメモリブロックのブロック数

・メモリブロックの獲得を待つタスクの待ち行列の先頭タスクのタスク**ID**

２．１０　時　間

２．１０．１　時間の概要

ハードウェアタイマを利用した時計であるシステムクロックを用いることにより、時間管理ができます。時間管理を行なうには、タイマ初期化ルーチンの作成、タイマ割込みハンドラの作成と登録が必要です。時間管理のシステムコールを次に示します。

- **sys_clk**システムコール(カーネルの時間管理処理を要求する)
- **set_tim,iset_tim**システムコール(時刻を設定する)
- **get_tim,iget_tim**システムコール(時刻を参照する)
- **wai_tsk**システムコール(自タスクを一定時間待ち状態へ以降する)

２．１０．２　ハードウェアタイマとシステムクロック

システムクロックは、一定周期で発生するハードウェアタイマからの割込みを計測する時計です。システムクロックは、符号付き**48**ビットのカウンタを持っています。ハードウェアタイマの割込みハンドラから**sys_clk**システムコールを発行し、システムクロックを更新**(+1)**します。

システムクロックは、ハードウェアタイマの周期時間**(tc)**を単位とします。システムクロックと現実の時刻には次の関係があります。

現実の時刻　＝　**システムクロックの値**　×　**ハードウェアタイマの周期時間(tc)**

たとえばハードウェアタイマの周期が**10msec**の場合、システムクロックのカウント値**10**は、**100msec**を表わします。

図２－１８にハードウェアタイマとシステムクロックの関係を示します。

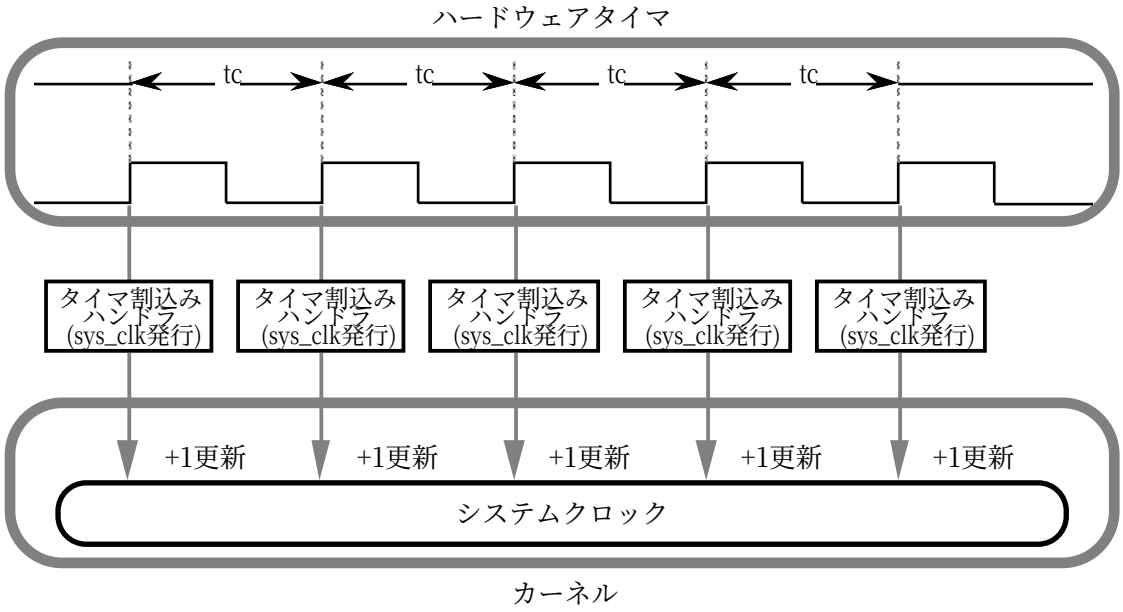


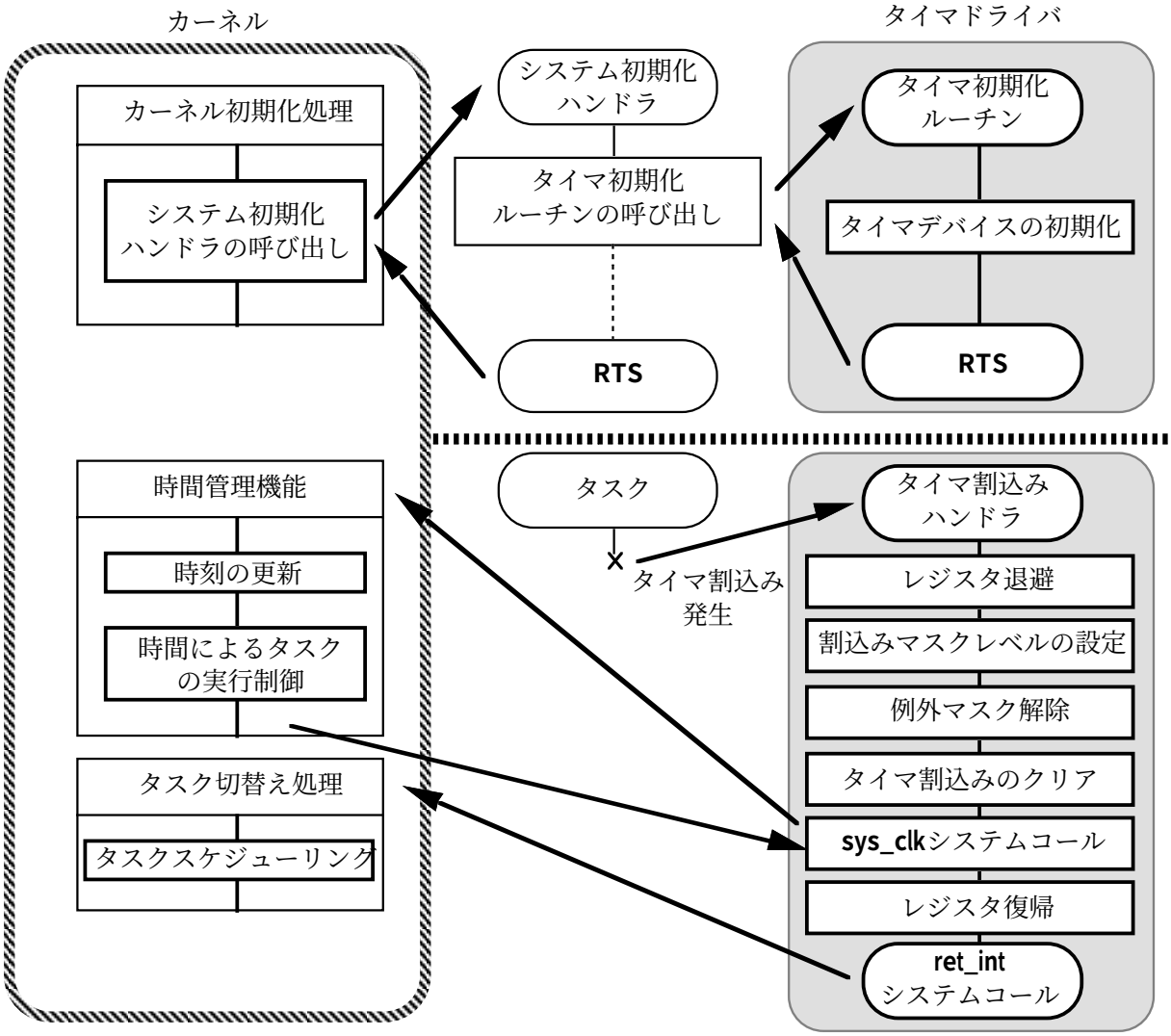
図２－１８　ハードウェアタイマとシステムクロックの関係

２．１０．３　タイマドライバの作成

タイマドライバは、タイマ初期化ルーチンとタイマ割込みハンドラから構成されます。

なお、**HI-SH77**では**SH7702,SH7708 MCU**内蔵のタイマユニット**(TMU:Timer Unit)**用のサンプルプログラムを提供します。**「付録Ｂ　タイマドライバの例題」**を参照してください。

図２－１９にタイマドライバの処理内容を示します。



【注】タイマ割込みはタスク実行中だけでなく、タイマ割込みより
割込みのレベルが低い割込みハンドラ実行中にも発生します。

図２－１９　タイマドライバの処理

(1)タイマ初期化ルーチンの作成

ハードウェアタイマに使用するタイマデバイスの初期化を行ないます。

システムクロックを正確に更新できるように、タイマデバイスの初期化内で、タイマの割込みレベルを他の割込みよりも高く、しかもカーネル割込みマスクレベル以下に設定してください。

タイマ初期化ルーチンは、セットアップテーブルに定義するシステム初期化ハンドラ内から呼び出すようにします。

(2)タイマ割込みハンドラの作成

タイマ割込みハンドラは、通常の割込みハンドラと同様に作成します。

タイマ割込みハンドラも、カーネルからみるとひとつの割込みハンドラです。ハードウェアタイマからの割込みが発生すると起動するようにします。

タイマ割込みハンドラでは、割込みマスクレベルの設定、例外マスク解除、タイマ割込みのクリアを行ない、**sys_clk**システムコールによりカーネルの時間管理処理を要求します。そして、**ret_int**システムコールにより終了します。

２．１０．４　タイマ割込みハンドラの登録

作成したタイマ割込みハンドラの先頭アドレスを、ベクタテーブルの該当番号位置に登録することによりタイマ割込みができるようになります。**図２－２０**にベクタテーブルとタイマ割込みハンドラとの関連を示します。

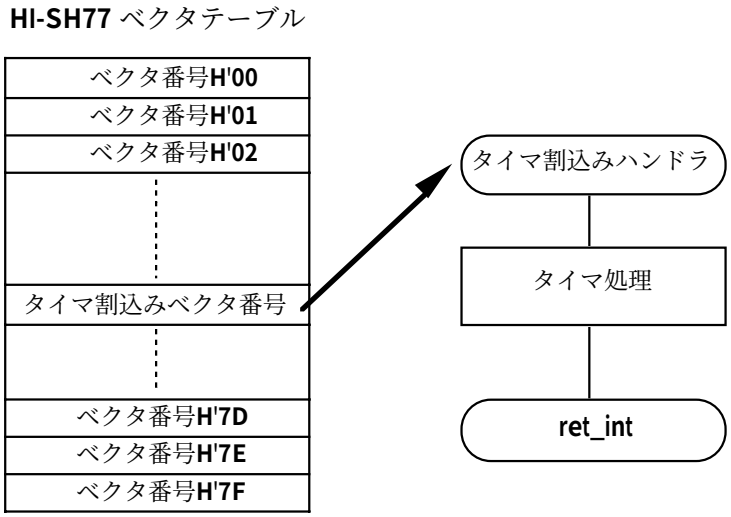


図２－２０　ベクタテーブルとタイマ割込みハンドラとの関連

２．１０．５　システムクロックの設定と参照

時刻はシステムクロックの値を参照することにより求めます。符号付き**48**ビットで表現される最大値は約**1.4**×**10**^{14}であり、ハードウェアタイマの割込み周期時間が**10msec**であれば、約**40000**年に相当します。時刻の基準点は**1985**年**1**月**1**日**0**時とすることを推奨します。

時刻の設定は**set_tim,iset_tim**システムコールを使用し、時刻の参照は**get_tim,iget_tim**システムコールを使用します。

図２－２１に時刻の設定および参照を示します。

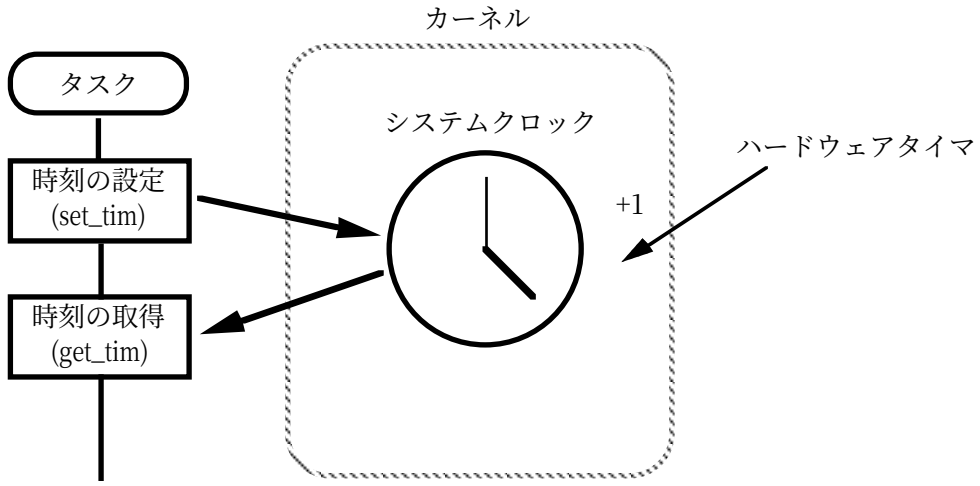


図２－２１　時刻の設定および参照

２．１１　拡張SVC

２．１１．１　拡張SVCの概要

ユーザが作成したプログラムをカーネルに組込み、このプログラムを拡張されたシステムコールとして通常のシステムコールと同様の手続きで発行することができます。この機能を拡張**SVC(SuperVisor Call)**、プログラムを拡張**SVC**ハンドラと呼びます。

拡張**SVC**は、各タスクにおける共通の処理や、タスクを切り換えることなく連続して実行したい処理などに利用できます。また、拡張**SVC**ハンドラ内は非タスク部専用システムコールが発行できますので、システムコールと組み合わせた処理が行なえます。

拡張**SVC**を使用するには、拡張**SVC**ハンドラの作成と登録が必要です。

拡張**SVC**はタスク部からのみ発行できます。拡張**SVC**の発行により、機能コードに対応する拡張**SVC**ハンドラが呼び出され、実行されます。

拡張**SVC**は最大**255**個まで使用でき、**1**から**255**の機能コードで識別します。システムで使用する拡張**SVC**ハンドラはセットアップテーブルに定義します。

図２－２２に拡張**SVC**の使用例を示します。

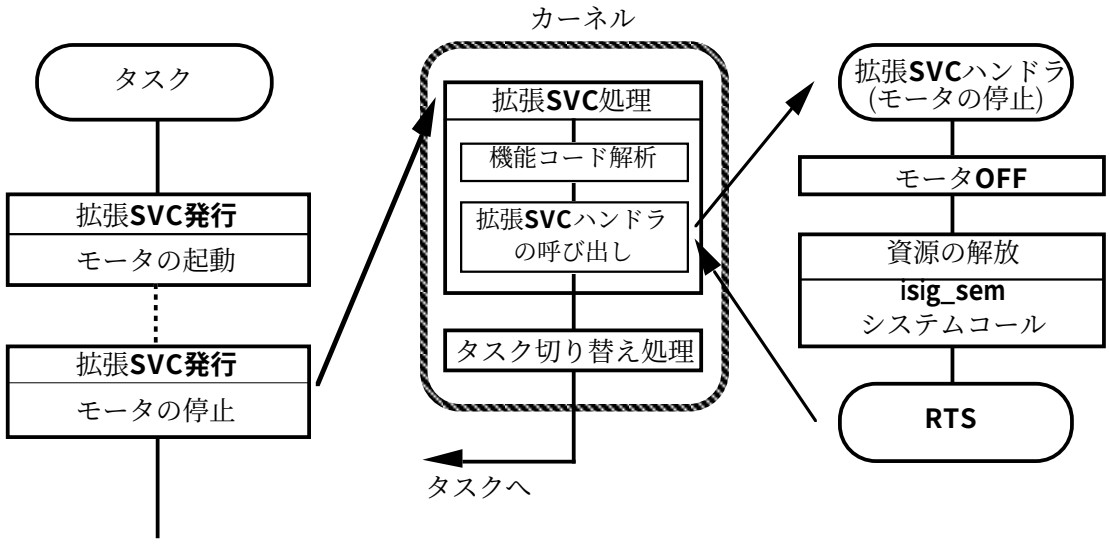


図２－２２　拡張SVCの使用例

２．１１．２　拡張SVCハンドラの作成と登録

拡張**SVC**ハンドラは、**表２－６**の処理条件を参考に作成してください。

表２－６　拡張SVCハンドラの処理条件

項番	項　　目		内　　容
1	システム状態		**OS**拡張部で実行します
2	**SR**	モード**(MD)**	特権モード**(1)**で動作します
3		レジスタバンク**(RB)**	・拡張**SVC**開始時、バンク**0(0)**に設定されています ・バンク**1(1)**のレジスタを使用する場合は例外マスク状態で行ってください ・終了する際は、バンク**0(0)**にしてください
4		例外マスク**(BL)**	拡張**SVC**開始時、例外マスクは解除**(0)**されています
5		割込みマスク**(I3～I0)**	・割込みマスクはレベル**0(0)**に初期設定されています ・終了する際は、開始時と同じ値にしてください
6	入力パラメータ		拡張**SVC**発行時のレジスタ**(R0, R4～R7)**が渡されます *1 **R0 :** 機能コード **R4～R7 :** パラメータ
7	出力パラメータ		拡張**SVC**ハンドラ終了時の**R0**レジスタが返されます *1 **R0** : エラーコード
8	使用できる レジスタ		・汎用レジスタ(**SP**を除く)、積和レジスタ、**GBR**に制限はありません ・プロシジャレジスタを使用する場合は、ハンドラ開始時にスタック領域に退避し、終了時に復帰してください *1 ・**VBR**は変更しないでください
9	**SP(R15)**		終了する際は、開始時と同じ値にしてください *1
10	使用できる スタック領域		・カーネルのスタック領域を使用します ・拡張**SVC**ハンドラのスタック使用量をセットアップテーブルに定義してください
11	使用できる システムコール		・例外マスク解除状態で非タスク部用システムコールが使用できます ・例外マスク状態では使用できません
12	終　了		**RTS**命令により終了します *1

【注】* 1　**C**言語記述の場合、考慮する必要はありません。詳細は**「４．５　C言語による拡張SVCハンドラの記述方法」**を参照してください。

拡張**SVC**ハンドラのカーネルへの登録は、先頭アドレスをセットアップテーブルに定義することで行ないます。拡張**SVC**の機能コードはこの定義順に**1**から決まります。

拡張**SVC**ハンドラの登録方法については、**『HI-SH77構築マニュアル』**を参照してください。

２．１２　ＭＣＵの初期化

２．１２．１　MCU初期化の概要

SH3のリセット時、カーネル起動前に、次に示す**MCU**の機能など各種ハードウェアの初期化が必要になる場合があります。

・キャッシュ制御レジスタ**(CCR**)
・MMU制御レジスタ**(MMUCR**)
・バスコントロールレジスタ(**BCR**)
・ウエイトステートコントロールレジスタ(**WCR**)
・個別メモリコントロールレジスタ(**MCR**)
・システム起動処理に関係するユーザハードウェア

リセットで起動し、これらの処理を行なうプログラムを、**MCU**初期化ルーチンと呼びます。

MCU初期化ルーチンは、リセット割込みで実行されるように、開始アドレスをリセットベクタに登録します。また、リセット時の初期スタックポインタはシステム構築用リンケージサブコマンドファイル**(himake.sub)**に登録します。

初期スタックポインタの登録方法については、『**HI-SH77構築マニュアル**』を参照してください。

２．１２．２　MCU初期化ルーチンの作成と登録

図２－２３に**MCU**初期化ルーチンの処理概要を示します。

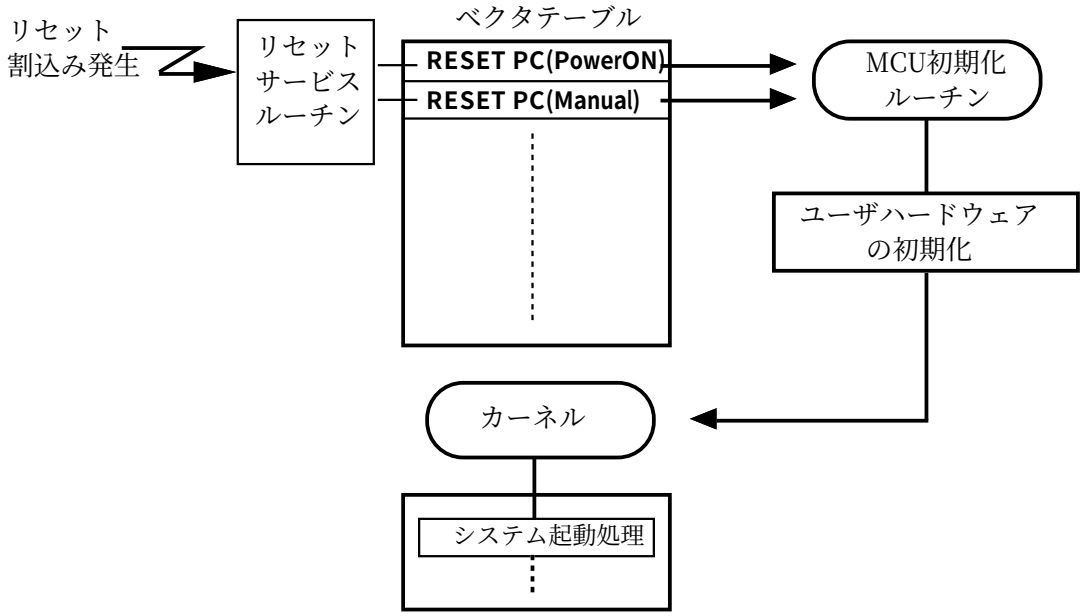


図２－２３　MCU初期化ルーチンの処理概要

SH3のリセットには、パワーオンリセットとマニュアルリセットの2種類があります。各リセットに対する**MCU**初期化ルーチンは、必要に応じて作成してください。

MCU初期化ルーチンは、**表２－７**の処理条件を参考に作成してください。

表２－７　MCU初期化ルーチンの処理条件

項番	項　　目		内　　容
1	**SR**	モード**(MD)**	特権モード**(1)**で動作します
2		レジスタバンク**(RB)**	・**MCU**初期化ルーチン起動時、バンク**0(0)**に設定されています ・バンク**1(1)**のレジスタを使用する場合は例外マスク状態で行ってください ・終了する際は、バンク**0(0)**にしてください
3		例外マスク**(BL)**	**MCU**初期化ルーチン起動時、例外はマスク**(1)**されています
4		割込みマスク**(I3〜I0)**	・割込みマスクはレベル**15(H'F)**に設定されています ・**MMU**、キャッシュ、バスコントローラなど**MCU**の初期化が終了するまでは、割込みマスクレベルを15に維持するようにしてください
5	使用できるレジスタ		・**VBR**は変更しないでください
6	使用できるスタック領域		・スタックポインタは、リセット時にシステム構築用リンケージサブコマンドファイルに設定したスタックポインタ値が設定されます ・カーネル作業領域やタスクスタック領域など、カーネル起動後に使用する領域と重なっていても問題はありません
7	使用できるシステムコール		システムコールは使用できません
8	終　了		カーネルのシステム起動処理の開始アドレスに分岐します

システム起動処理の名称は、Ｃ言語プログラムの場合は"**_0Hrs__knl**"、アセンブリ言語プログラムの場合は"**__0Hrs__knl**"と決められています。

なお、ベクタベースレジスタ(**VBR**)はリセット時カーネルのリセットサービスルーチンが設定します。ベクタベースレジスタ(**VBR**)は変更しないようにしてください。**HI-SH77**応用システムでは常に固定と仮定しています。変更した場合、システムの正常な動作は保証されません。

２．１３　システム起動処理

２．１３．１　システム起動処理の概要

システムを起動すると、次の処理が実行されます

(1)スタックポインタの設定
スタックポインタがカーネル用スタック領域に設定されます。

(2)セットアップテーブルのチェック
セットアップテーブルのチェック機能を組み込んだシステムでは、セットアップテーブルのチェックが行なわれます。チェックされるセットアップテーブルの項目や内容についての詳細は、
「付録Ｅ．３　セットアップテーブルエラーコード一覧」を参照してください。
エラーが発生すれば、システム異常終了処理へ移行します。

(3)カーネル作業領域の初期化

(4)システム初期化ハンドラの呼び出し
システム初期化ハンドラがセットアップテーブルに定義されていれば、呼び出されます。

(5)タスク起動
セットアップテーブルで初期登録タスクが定義され、システム初期化ハンドラで**ista_tsk**システムコールが発行されていれば、実行可能な最も優先度の高いタスクが起動します。
実行可能なタスクが存在しない場合は、カーネルアイドリング状態になります。

図２－２４にシステム起動処理の処理概要を示します。

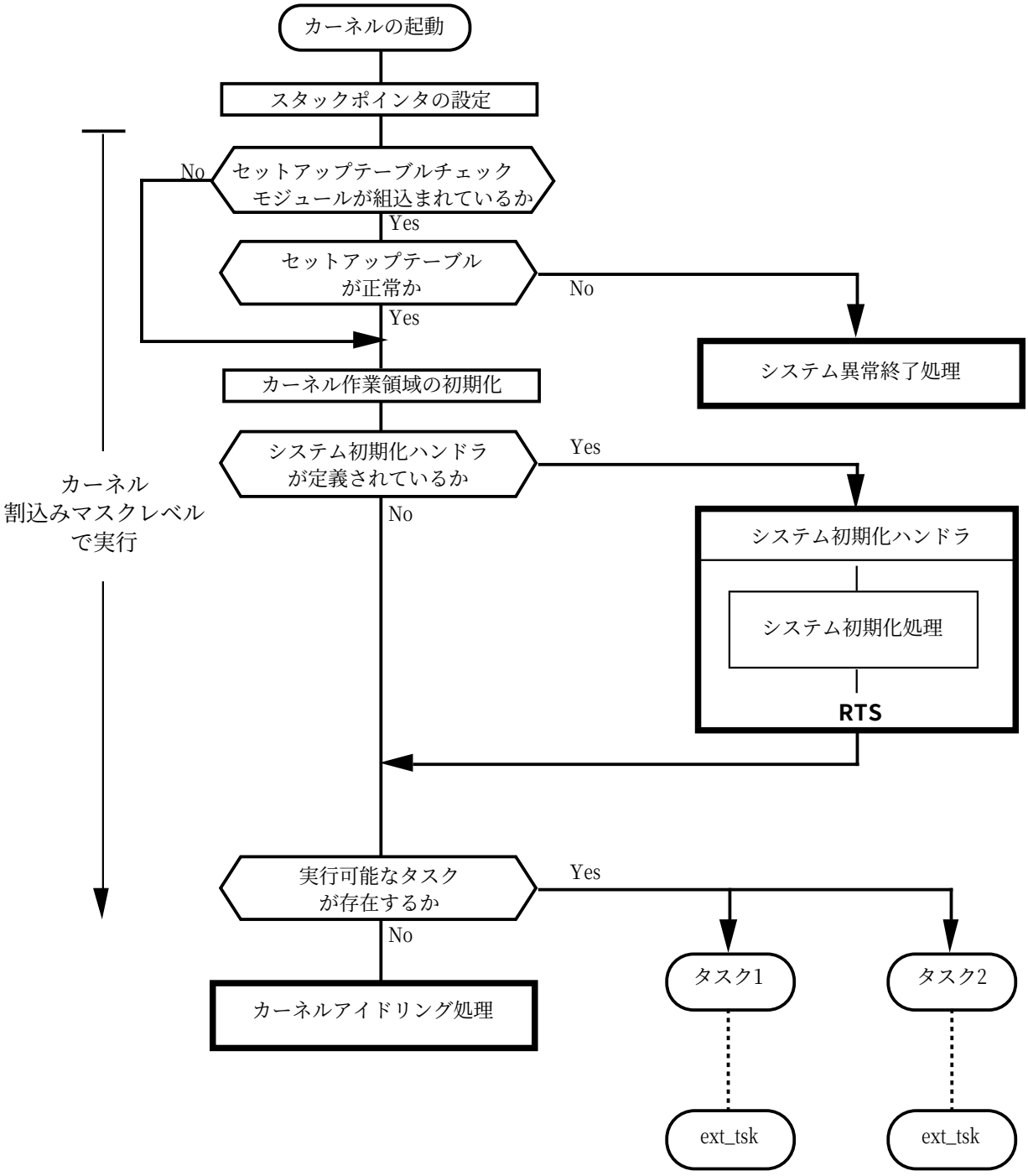


図２－２４　システム起動処理の概要

２．１３．２　システム初期化ハンドラ

システム初期化ハンドラとは、カーネルのシステム起動処理を行なった後、**OS**のユーザ拡張部としてカーネルから呼び出されるプログラムです。

システム初期化ハンドラはタスクを起動する前に実行されますので、初期登録タスクの起動、資源の初期化、およびハードウェアの初期化などを行なうことができます。

システム初期化ハンドラは、セットアップテーブルに定義することでカーネルに登録します。システム初期化ハンドラの定義方法の詳細は、『**HI-SH77構築マニュアル**』を参照してください。

システム初期化ハンドラは、**表２－８**の処理条件を参考に作成してください。

表２－８　システム初期化ハンドラの処理条件

項番	項　　目		内　　容
1	システム状態		**OS**拡張部で実行します
2	**SR**	モード**(MD)**	特権モード**(1)**に設定されています
3		レジスタバンク**(RB)**	システム初期化ハンドラ開始時、バンク**0(0)**に設定されています バンク**1(1)**のレジスタを使用する場合は、例外マスク状態で行ってください
4		例外マスク**(BL)**	システム初期化ハンドラ開始時、例外マスクは解除**(0)**されています
5		割込みマスク**(I3～I0)**	・カーネルの割込みマスクレベルでマスクされています ・割込みマスクを変更しないでください
6	使用できるレジスタ		・汎用レジスタ(**SP**を除く)、積和レジスタ、**GBR**に制限はありません ・プロシジャレジスタを使用する場合は、ハンドラ開始時にスタック領域に退避し、終了時に復帰してください * ・**VBR**は変更しないでください
7	**SP(R15)**		終了する際は、開始時と同じ値にしてください*
8	使用できるスタック領域		・カーネルのスタック領域を使用します ・システム初期化ハンドラのスタック使用量をセットアップテーブルに定義してください
9	使用できるシステムコール		・例外マスク解除状態で**ret_int,ret_exc**システムコール以外の非タスク部用システムコールが使用できます ・例外マスク状態では使用できません
10	終　了		**RTS**命令により終了します*

【注】* C言語記述の場合、考慮する必要はありません。詳細は**「４．７　C言語によるシステム初期化ハンドラの記述方法」**を参照してください。

２．１４　システムの異常終了処理

システムに異常が発生した場合、割込みマスクがカーネル割込みマスクレベルに設定され、システム異常終了処理に制御が渡されます。ユーザプログラムからシステム異常終了処理に制御を渡す場合は、特権モードから行ってください。

システム異常終了処理の名称は、C言語プログラムの場合は"**hi_sysdwn**"、アセンブリ言語プログラムの場合は"**_hi_sysdwn**"と決められています。

システム異常終了処理には、次のパラメータが渡されます。

R4 : エラー種別**(type)**

0以下の場合はカーネルのシステム異常を示し、**1**以上の場合はユーザのシステム異常を示します。

R5 : エラーコード**(ercd)**

カーネルシステム異常時のエラーコードを示します

R6 : システムダウン情報**(inf)**

システム異常の詳細情報です

システム異常終了処理には、異常内容に応じたプログラムを記述することができます。ただし、システムコールなどカーネルの機能は使用できません。

なお、システム異常終了処理**(hi_sysdwn)**はカーネルから取り外すことはできません。

表２－９にシステム異常終了処理の入力パラメーター一覧を示します。

表２－９　システム異常終了処理の入力パラメーター一覧

項番	エラー種別 **(R4)**	エラーコード **(R5)**	システムダウン情報**(R6)**	内　　容
1	**H'7FFFFFFF** ～ **H'00000001**	ユーザ仕様による		ユーザプログラム内で発生したシステム異常を**hi_sysdwn**で処理する場合に、正のエラー種別が使用できます
2	**H'00000000**	セットアップテーブルチェックエラーコード	**H'00000000**	セットアップテーブルのエラー（セットアップテーブルチェック機能が定義されている場合のみ）
3	**H'FFFFFFFF**	**H'FFFFF5BB (E_CTX)**	**ext_tsk**を発行した(アドレス+2)番地	非タスク部での**ext_tsk**システムコールの発行によるコンテキストエラー
4	**H'FFFFFFFE**	**H'FFFFF5BB (E_CTX)**	**exd_tsk**を発行した(アドレス+2)番地	非タスク部での**exd_tsk**システムコールの発行によるコンテキストエラー
5	**H'FFFFFFFD**	**H'FFFFF5BB (E_CTX)**	**ret_int**を発行した(アドレス+2)番地	タスク部での**ret_int**システムコールの発行によるコンテキストエラー
6	**H'FFFFFFFC**	**H'FFFFF5BB (E_CTX)**	**sys_clk**を発行した(アドレス+2)番地	タスク部での**sys_clk**システムコールの発行によるコンテキストエラー
7	**H'FFFFFFFB** ～ **H'80000000**	システム予約		システム予約

２．１５　カーネルのアイドリング

実行可能状態のタスクが存在しない場合、カーネルアイドリング状態になります。カーネルアイドリング状態では、例外マスクは解除(**SR**の**BL**ビット=0)割込みマスクはオープン(レベル=0)に設定され、無限ループとなります。

ユーザシステムで実行すべきタスクが存在しないときに、**MCU**の低消費電力機能を利用したい場合は、ユーザのタスク（システムで最も優先度を低く設定します）の中で低消費電力状態に移行してください。

２．１６　トレース機能

トレース機能は、システム実行中のシステムコール発行履歴をトレースバッファに保存する機能です。本機能により、システムコールの発行順序やマルチタスク環境での各タスクの動きを知ることができます。

２．１６．１　トレース取得の概要

図２－２５に、トレース取得時の動作概要を示します。

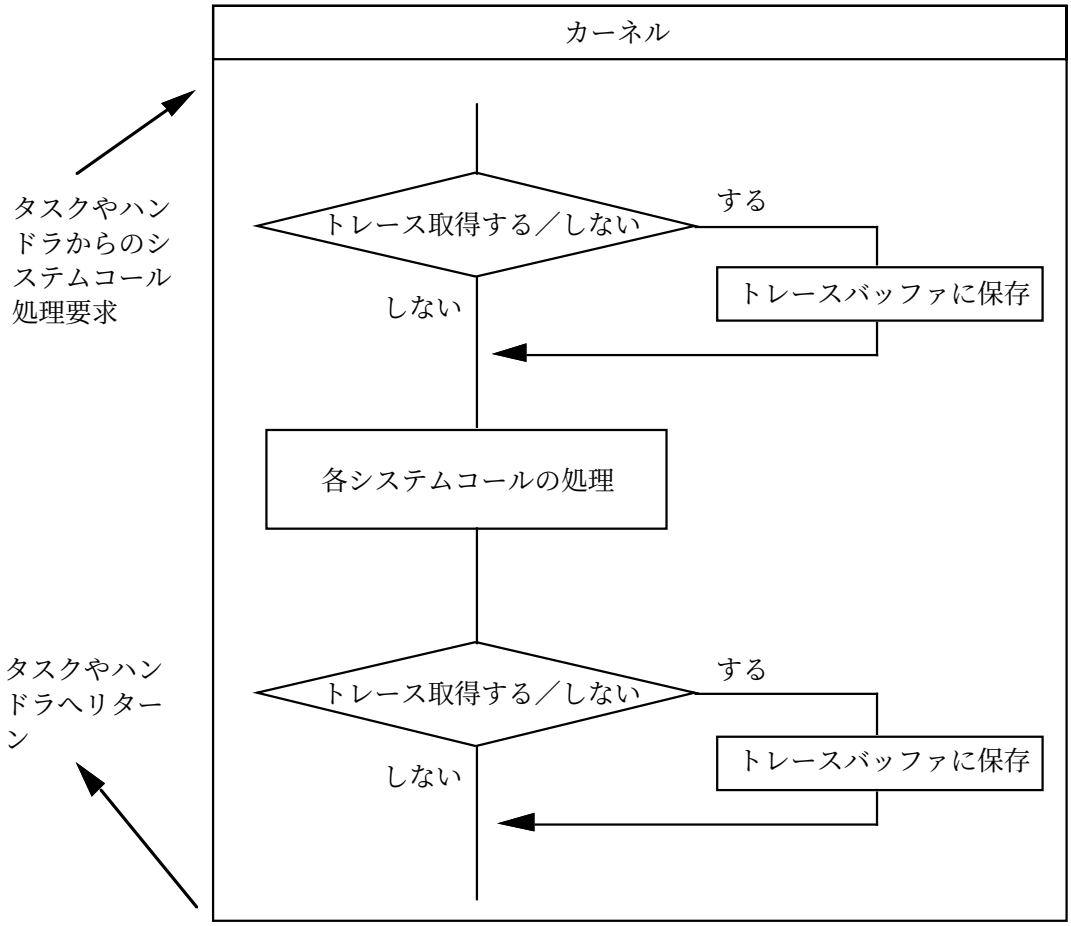


図２－２５　トレース取得時の動作概要

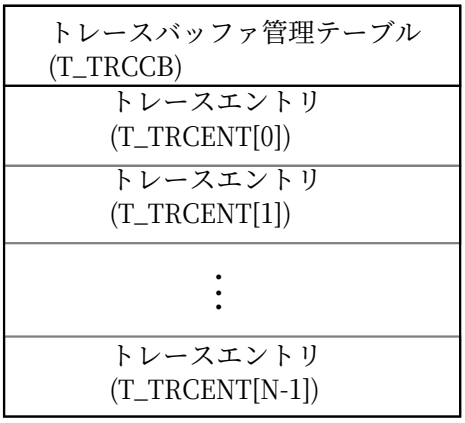
図２－２４に示すように、トレースの取得は、システムコールの発行により、ユーザプログラムからカーネルに入った時点と、カーネルからユーザプログラムに戻るときに行なわれます。すなわち、基本的には1回のシステムコール発行により、発行時とリターン時の2つの情報が取得されます。この情報を「イベント」と呼びます。

トレース機能を使用する場合、システム構築時に、セットアップテーブルで使用の有無を選択し、トレースバッファ領域定義プログラムで、トレースバッファ領域を確保します。トレース機能の使用を選択すると、システム起動後、システム初期化ハンドラ以降のすべてのイベントが取得されます。トレースバッファはリングバッファ構造となっており、古い情報は順次新しい情報に書き換えられます。

システムへのトレース機能の組み込みについては、『**HI-SH77構築マニュアル**』を参照してください。

２．１６．２　トレースバッファの構造

図２－２６に、トレースバッファの構造を示します。



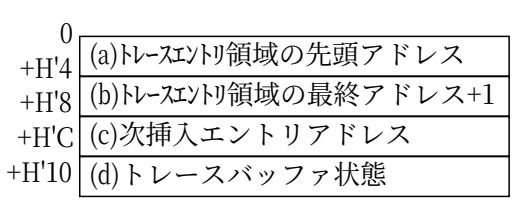
【注】 Nはトレース情報取得数

図２－２６　トレースバッファの構造

トレースエントリ領域は、実際に取得される情報が格納される領域で、リングバッファ構造になっています。1イベントにつき、1つのトレースエントリが使用されます。

(1)トレースバッファ管理テーブル(T_TRCCB)

トレースバッファの管理を行なうテーブルです。トレース取得時に、カーネルがこの領域を使用します。**図２－２７**に、トレースバッファ管理テーブルの構造を示します。




```
typedef   struct   t_trccb {
 VP   tr_trbtop;   /* ﾄﾚｰｽｴﾝﾄﾘ領域の先頭ｱﾄﾞﾚｽ     */
 VP   tr_trbbtm;   /* ﾄﾚｰｽｴﾝﾄﾘ領域の最終ｱﾄﾞﾚｽ+1   */
 VP   tr_trbins;   /* 次挿入エントリアドレス       */
 UW   trbsts;      /* トレースバッファ状態         */
 } T_TRCCB;
```

図２－２７　トレースバッファ管理テーブルの構造

トレース情報の格納可能な領域は、(a)～(b)で保持しています。これらは、システム起動時にセットアップテーブルの設定内容にしたがって初期化されます。

(c)の次挿入エントリアドレスは、次にイベントが発生したときにその情報が格納されるエントリのアドレスです。

(d)のトレースバッファ状態は、以下の意味を持ちます。

ビット0　　：リングバッファフラグ

0：トレースバッファへの書込みは一周していない

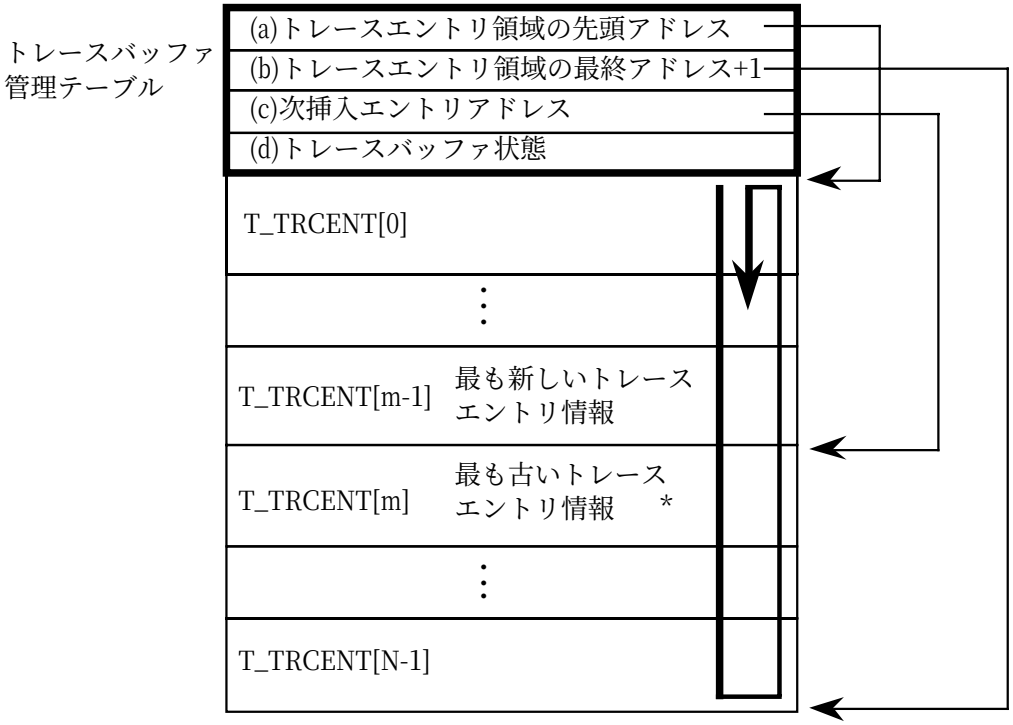
1：トレースバッファへの書込みは一周した

ビット1　　：トレース取得中フラグ

カーネルがトレース取得処理中の時に、"1"となります。この場合、カーネルは(c)次挿入エントリアドレスにトレース情報を格納している最中なので、このエントリの情報は不定となります。

その他のビットは不定です。

図２－２８に、トレースバッファの管理の様子を示します。



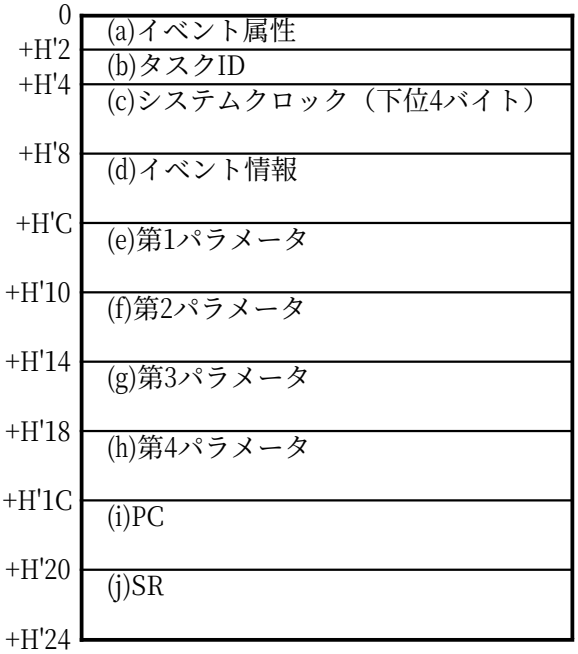
【注】* トレースバッファ状態のビット1が"1"の場合は、不定データとなります。

図２－２８　トレースバッファ管理の様子

(2)トレースエントリ(T_TRCENT)

1イベントにつき、1つのトレースエントリにトレース情報が格納されます。

図２－２９に、トレースエントリの構造を示します。




```
typedef  struct  t_trcent {
  UH  te_attr;   /* イベント属性              *
  ID  te_tskid;  /* タスクID                  *
  UW  te_ltime;  /* システムクロック値(下位) *

  W   te_event; /* イベント情報(R0)          */

  VW  te_par1;   /* 第1パラメータ(R4)         *

  VW  te_par2;   /* 第2パラメータ(R5)         *

  VW  te_par3;   /* 第3パラメータ(R6)         *

  VW  te_par4;   /* 第4パラメータ(R7)         *

  UW  te_pc;     /* PC                        *

  UW  te_sr;     /* SR                        *

  } T_TRCENT;
```

図２－２９　トレースエントリの構造

(a)のイベント属性は、そのトレースエントリの内容の意味を識別するための情報です。イベント属性には、以下の4つがあります。

・SVC属性(TATR_SVC:H'0001)

・RTN属性(TATR_RTN:H'0002)

・CONT属性(TATR_CONT:H'0003)

・IDLE属性(TATR_IDLE:H'0004)

SVC属性はシステムコール発行のイベントを意味し、トレースエントリの内容はシステムコール発行時の情報となります。

RTN属性はカーネルからタスクに戻るイベントを意味し、トレースエントリの内容はシステムコールリターン時の情報、またはタスク起動時の情報となります。

なお、システムダウンとなるコンテキストエラー(E_CTX)を発生させたシステムコールについては、RTN属性は取得されません。対象となるシステムコールには、タスク部からの**ret_int,sys_clk**システムコール、非タスク部からの**ext_tsk,exd_tsk**システムコールがあります。

CONT属性はタスクが割込み発生点から実行を再開する場合に取得されます。

割込み発生点から実行を再開するケースには、以下の場合があります。なお、例外はタスクの一部であるため、このケースにはあてはまりません。

(i)割込みハンドラがタスク切り換えの必要のあるシステムコール（**sys_clk**システムコールを含む）を発行しないまま**ret_int**システムコールを発行し、割込み発生前に実行していたタスクに戻る場合

(ii)割込みハンドラでタスク切り換えの必要のあるシステムコール（**sys_clk**システムコールを含む）を発行した後に**ret_int**システムコールを発行し、その結果割込み発生前に実行していたタスク以外のタスクにスケジューリングされ、その後割込み発生前に実行していたタスクに再度スケジューリングされた場合

CONT属性は(ii)の場合に取得されます。(i)の場合は、何のイベントも取得されません。

IDLE属性は、システムアイドリングに入るときに取得されます。

トレースエントリのイベント属性以外のデータは、イベント属性によってその意味が異なります。**表２－１０**に、トレースエントリ内容の意味を示します。

表２－１０　トレースエントリ内容の意味

te_attr	TATR_SVC (H'0001)	TATR_RTN (H'0002)	TATR_CONT (H'0003)	TATR_IDLE (H'0004)
te_tskid	ｼｽﾃﾑｺｰﾙを発行したﾀｽｸID。非ﾀｽｸ部の場合は0。	ｶｰﾈﾙからの戻り先となるﾀｽｸID。非ﾀｽｸ部の場合は0。	割込み発生点から実行再開するﾀｽｸID。非ﾀｽｸ部(0)の場合はありえません。	不定
te_ltime	イベント取得時のシステムクロックの下位4バイト			
te_event	発行したｼｽﾃﾑｺｰﾙの機能ｺｰﾄﾞ * (ｼｽﾃﾑｺｰﾙ発行時のR0)	ｼｽﾃﾑｺｰﾙのｴﾗｰｺｰﾄﾞ。ただし、H'80000000の場合は、ﾀｽｸ起動のイベントを示します。(R0)	不定	不定
te_par1～4	ｼｽﾃﾑｺｰﾙのﾊﾟﾗﾒｰﾀ (ｼｽﾃﾑｺｰﾙ発行時のR4～R7)	ｼｽﾃﾑｺｰﾙのﾘﾀｰﾝﾊﾟﾗﾒｰﾀ。ただし、ﾀｽｸ起動時は不定。(ｼｽﾃﾑｺｰﾙﾘﾀｰﾝ時のR4～R7)	不定	不定
te_pc	ｼｽﾃﾑｺｰﾙ発行ｱﾄﾞﾚｽ+2	ﾕｰｻﾞﾌﾟﾛｸﾞﾗﾑへの戻り先ｱﾄﾞﾚｽ。ﾀｽｸ起動時は、ﾀｽｸｽﾀｰﾄｱﾄﾞﾚｽとなり、ﾀｽｸ起動時以外は以前のｼｽﾃﾑｺｰﾙ発行ｱﾄﾞﾚｽ＋2となります。	不定	不定
te_sr	ｼｽﾃﾑｺｰﾙ発行時のSR	ﾕｰｻﾞﾌﾟﾛｸﾞﾗﾑに戻ったときのSR	不定	不定

【注】* **sys_clk**、**ret_int**および**ret_exc**システムコールは取得されません。

２．１６．３　トレース取得データの解析例

取得されたトレースデータが、**表２－１１**であったとします。

表２－１１　トレース取得データの例

No.		te_attr	te_tskid	te_ltime	te_event	te_par1	te_pc	te_sr
-12		H'0001	H'0005	H'00001234	H'fffffefe	H'00000003	H'00003018	H'00000000
	古	SVC	tskid=5		sta_tsk	ID=3を起動		
-11		H'0002	H'0003	H'00001234	H'80000000	H'xxxxxxxx	H'00003800	H'00000000
		RTN	tskid=3		タスク起動			
-10		H'0001	H'0003	H'00001234	H'fffffef2	H'xxxxxxxx	H'00003810	H'00000000
		SVC	tskid=3		slp_tsk			
-9		H'0002	H'0005	H'00001234	H'00000000	H'00000003	H'00003018	H'00000000
		RTN	tskid=5		E_OK			
-8		H'0001	H'0000	H'00001234	H'fffffff7	H'00000003	H'00007340	H'00000060
		SVC	非ﾀｽｸ		iwup_tsk	3番のﾀｽｸを起床		割込みﾏｽｸ=6
-7		H'0002	H'0000	H'00001234	H'00000000	H'00000003	H'00007340	H'00000060
		RTN	非ﾀｽｸ		E_OK			割込みﾏｽｸ=6
-6		H'0002	H'0003	H'00001234	H'00000000	H'xxxxxxxx	H'00003810	H'00000000
		RTN	tskid=3		E_OK			
-5		H'0001	H'0003	H'00001234	H'fffffef8	H'00000005	H'00003840	H'00000000
		SVC	tskid=3		rel_wai	5番を強制待ち解除		
-4		H'0002	H'0003	H'00001234	H'fffff7c2	H'00000005	H'00003840	H'00000000
		RTN	tskid=3		E_NOWAI			
-3		H'0001	H'0003	H'00001234	H'fffffefc	H'xxxxxxxx	H'0000385c	H'00000000
		SVC	tskid=3		ext_tsk			
-2		H'0003	H'0005	H'00001234	（不定データが入っている）			
		CONT	tskid=5					
-1		H'0001	H'0005	H'00001234	H'fffffefc	H'xxxxxxxx	H'00003054	H'00000000
		SVC	tskid=5		ext_tsk			
0	新	H'0004	（不定）	H'00001234	（不定データが入っている）			
		IDLE						

【注】・説明を簡単にするため、te_par2～te_par4のデータは省略しています。

・上段がトレース取得されたデータ、下段がその簡単な説明になっています。

・Noは、説明のために便宜上付けたものであり、トレースデータとして取得されるわけではありません。

以下に、**表２－１１**の説明を示します。

(1)イベントNo.-12

te_attr=H'0001より、SVC属性であることがわかります。te_tskid=5より、5番のタスクが何らかのシステムコールを発行したことがわかります。そのシステムコールは、システムコールの機能コードを示すte_eventがH'fffffefeなので、**sta_tsk**システムコールであることがわかります。**sta_tsk**システムコールのパラメータtskidは、te_par1=3より3番のタスクを起動したことがわかります。なお、**sta_tsk**システムコールを発行した命令のアドレスはte_pc=H'00003018より、H'3016番地(=3018-2)であることもわかります。

(2)イベントNo.-11

te_attr=H'0002(RTN属性),te_tskid=3より、3番のタスクに制御が渡ったことがわかります。te_event=H'80000000より、この時点で実際にtskid=3が起動されたことがわかります。タスクスタートアドレスは、te_pc=H'00003800です。

イベントNo.-12との関係から、5番のタスクが発行した**sta_tsk**システムコールにより、5番のタスクから3番のタスクにスイッチしたことになります。

(3)イベントNo.-10

te_attr=H'0001(SVC属性),te_tskid=3,te_event=H'fffffef2より3番のタスクが**slp_tsk**を発行したことがわかります。

(4)イベントNo.-9

te_attr=H'0002(RTN属性),te_tskid=5より、5番のタスクに制御が渡ったことがわかります。イベントNo.-10との関係から、3番のタスクが発行したslp_tskシステムコールにより、3番のタスクから5番のタスクにスイッチしたことがわかります。5番のタスクは、イベントNo.-12のところで**sta_tsk**システムコールの発行して以来、この時点まで実行していない（イベントが取得されていない）ので、te_eventは、イベントNo.-12の**sta_tsk**システムコールに対するエラーコードとなります。

(5)イベントNo.-8

te_attr=H'0001(SVC属性),te_tskid=0,te_event=H'ffffffff7,te_par1=3より、非タスク部から**wup_tsk**(tskid=3)が発行されたことがわかります。非タスク部には、割込みハンドラ、拡張SVCハンドラ、システム初期化ハンドラなどがありますが、te_sr=H'00000060より割込みレベル6の割込みハンドラであると推測できます。

割込みハンドラであるとすると、No.-9のイベントからこのイベントまでの間に、その割込みが発生したことになります。

(6)イベントNo.-7

te_attr=H'0002(RTN属性),te_tskid=0より、非タスク部のシステムコールリターンの情報であることがわかります。直前の非タスク部からのシステムコール発行は、イベントNo.-8のiwup_tskなので、te_eventはこのiwup_tskに対するエラーコードとなります。

(7)イベントNo.-6

te_attr=H'0002(RTN属性),te_tskid=3より、3番のタスクに制御が渡ったことがわかります。直前のNo.-7のイベントは非タスク部のイベントだったので、NO.-7とNO.-6のイベントの間で、割込みハンドラから**ret_int**システムコールが発行され、その結果3番のタスクにスケジューリングされ、本イベントが取得されたことになります。この結果より、タスク優先度は5番より3番のタスクのほうが高いことがわかります。3番のタスクはイベントNo.-10の**slp_tsk**システムコールの発行以来実行されていないので、te_eventはこの**slp_tsk**システムコールに対するエラーコードとなります。

(8)イベントNo.-5

te_attr=H'0001(SVC属性),te_tskid=3,te_event=H'fffffef8,te_par1=H'5より、3番のタスクが**rel_wai**(tskid=5)を発行したことがわかります。

(9)イベントNo.-4

te_attr=H'0002(RTN属性),te_tskid=3,te_event=H'fffff7c2,より、3番のタスクが発行したイベントNo.-5の**rel_wai**システムコールが、エラー終了（エラーコードE_NOWAI）したことがわかります。

(10)イベントNo.-3

te_attr=H'0001(SVC属性),te_tskid=3,te_event=H'fffffefcより、3番のタスクが**ext_tsk**システムコールを発行したことがわかります。

(11)イベントNo.-2

te_attr=H'0003(CONT属性),te_tskid=5より、5番のタスクが割込み発生点から実行を再開することがわ

かります。それまで実行していた3番のタスクが**ext_tsk**システムコールを発行した（イベントNo.-3）ことで、5番にスケジューリングされたことになります。トレースデータをさかのぼると、5番のタスクに関するデータは、No.-9のイベントからこの時点までありません。すなわち、No.-9とNo.-8のイベントの間で割込みが発生し、5番のタスクの実行が中断されていたことになります。

(12)イベントNo.-1

te_attr=H'0001(SVC属性),te_tskid=5,te_event=H'fffffefcより、5番のタスクが**ext_tsk**システムコールを発行したことがわかります。

(13)イベントNo.0

te_attr=H'0004(IDLE属性)により、システムアイドリングに移行したことがわかります。

このような解析を元に、**表２－１１**のトレースデータから実際のプログラムの動きを図示すると、**図２－３０**のようになります。理解を深めるための参考としてください。

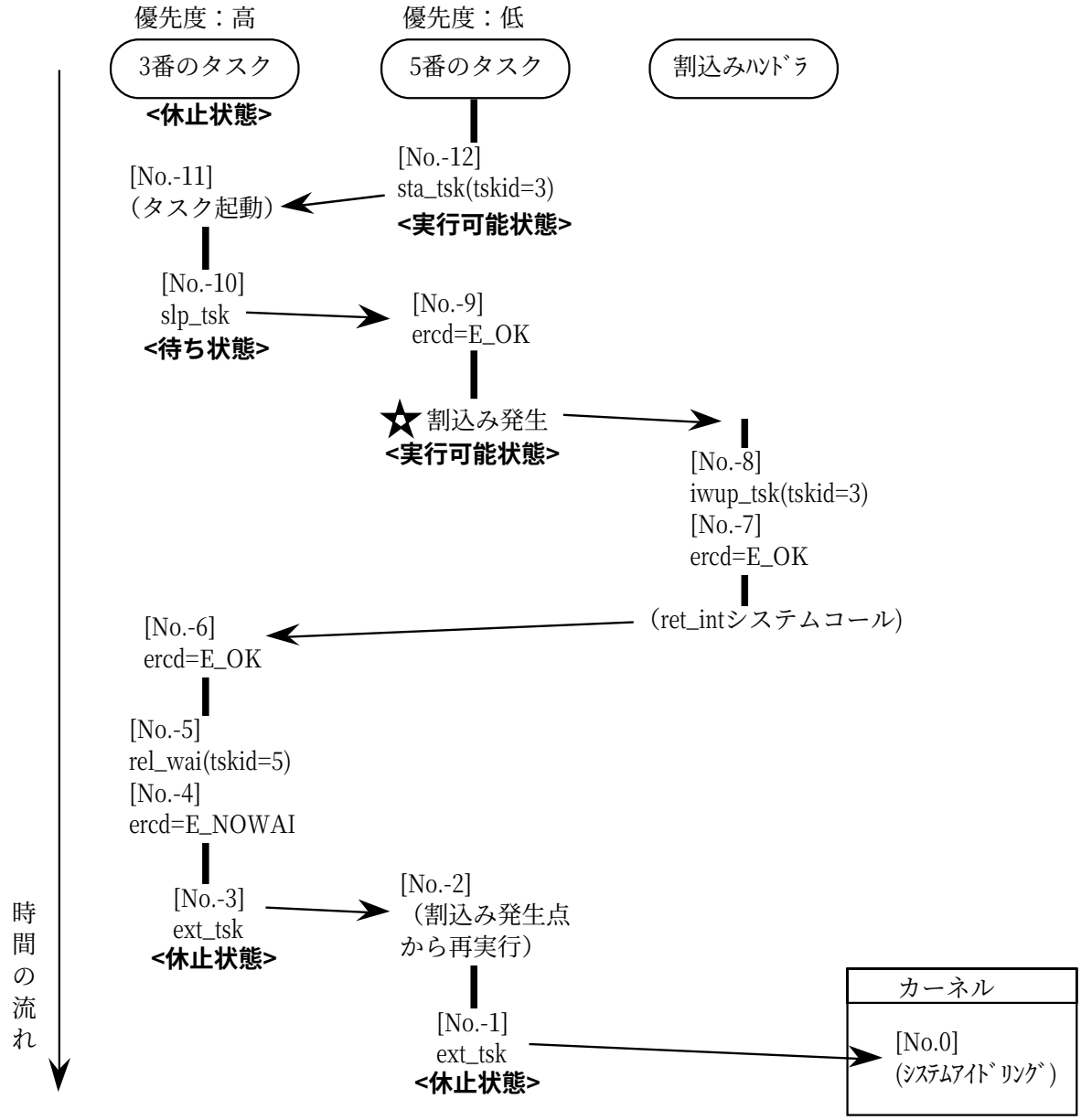


図２－３０　トレース解析結果の図示例

２．１６．４　トレース機能の使用上の注意

(1)カーネル性能の悪化

トレース機能を使用する場合、システムコール処理としてトレース取得の処理が加わるため、トレース機能を使用しない場合に比べ、システムコール処理時間が増加します。また、カーネルの割込み禁止時間も増加します。システムによっては、これらが原因でタイミング的な問題が発生することが考えられます。

これら性能悪化は、トレースバッファ領域を外部メモリに配置した場合に顕著となります。当社のE7000エミュレータを使用する場合は、トレースバッファ領域をエミュレーションメモリに割り付けることを推奨します。

(2)トレースバッファの書込み

ユーザプログラムから、トレースバッファへの書込みは、行なわないでください。特に、トレースバッファ管理テーブルのデータを書き換えた場合、システムの正常な動作は保証されません。

３.システムコール

３.１　概　要

HI-SH77では、全部で**71**個のシステムコールを使用できます。システムコールは**表３－１**のように分類されます。

表３－１　システムコールの分類

項番	分　類	機　能
1	タスク管理機能	タスクの起動、終了を行ないます
2	タスク付属同期機能	タスク実行の中断、再開を行ないます
3	同期／通信機能	イベントフラグ、セマフォ、メイルボックスの管理を行ないます
4	割込み管理機能	割込みハンドラからの復帰 および割込みマスクの変更、参照を行ないます
5	例外管理機能	例外処理からの復帰を行ないます
6	メモリプール管理機能	固定長メモリブロックで構成されるメモリプールの管理を行ないます
7	時間管理機能	カーネルの時間管理処理の要求 およびシステムクロックの設定、参照を行ないます
8	システム管理機能	カーネルのバージョン番号を求めます
9	拡張**SVC**機能	拡張**SVC**ハンドラの呼び出しを行ないます

表３－２に全システムコールの発行可能なシステム状態（タスク部または非タスク部）、および仕様を記載しているページを示します。

表３－２　システムコールの発行可能なシステム状態と仕様記載ページ

	システムコール	機　能	タスク部(T)／非タスク部(N)	仕様記載ページ
	タスク管理機能			
1.	cre_tsk*	タスクを生成する	T	3-19
2.	sta_tsk	タスクを起動する	T	3-129
3.	ista_tsk	同上（非タスク部用）	N	3-85
4.	del_tsk*	タスクを削除する	T	3-21
5.	ext_tsk	自タスクを正常終了する	T	3-24
6.	exd_tsk*	自タスクを正常終了後、削除する	T	3-23
7.	ter_tsk	他タスクを強制的に異常終了させる	T	3-136
8.	chg_pri	タスク優先度を変更する	T	3-15
9.	ichg_pri	同上（非タスク部用）	N	3-40
10.	rot_rdq	タスクのレディーキューを回転する	T	3-114
11.	irot_rdq	同上（非タスク部用）	N	3-71
12.	rel_wai	タスクの待ち状態を強制解除する	T	3-109
13.	irel_wai	同上（非タスク部用）	N	3-69
14.	get_tid	自タスクIDを得る	T	3-29
15.	iget_tid	同上（非タスク部用）	N	3-46
16.	tsk_sts	タスク状態を見る	T	3-138
17.	itsk_sts	同上（非タスク部用）	N	3-89
	タスク付属同期機能			
18.	sus_tsk	タスクを強制待ち状態へ移行する	T	3-131
19.	isus_tsk	同上（非タスク部用）	N	3-87
20.	rsm_tsk	強制待ち状態のタスクを再開する	T	3-116
21.	irsm_tsk	同上（非タスク部用）	N	3-73
22.	slp_tsk	自タスクを待ち状態へ移行する	T	3-126
23.	wai_tsk	自タスクを一定時間待ち状態へ移行する	T	3-145
24.	wup_tsk	待ち状態のタスクを起床する	T	3-147
25.	iwup_tsk	同上（非タスク部用）	N	3-91
26.	can_wup	タスクの起床要求を無効にする	T	3-11
27.	ican_wup	同上（非タスク部用）	N	3-36
	同期／通信機能			
28.	set_flg	イベントフラグをセットする	T	3-120
29.	iset_flg	同上（非タスク部用）	N	3-77
30.	clr_flg	イベントフラグをクリアする	T	3-17
31.	iclr_flg	同上（非タスク部用）	N	3-42
32.	wai_flg	イベントフラグを待つ	T	3-140
33.	pol_flg	イベントフラグを得る	T	3-99
34.	ipol_flg	同上（非タスク部用）	N	3-61
35.	flg_sts	イベントフラグの状態を参照する	T	3-25
36.	iflg_sts	同上（非タスク部用）	N	3-44
37.	sig_sem	セマフォに対する信号操作（V命令）	T	3-124
38.	isig_sem	同上（非タスク部用）	N	3-81
39.	wai_sem	セマフォに対する待ち操作（P命令）	T	3-143

表３－２　システムコールの発行可能なシステム状態と仕様記載ページ（つづき）

	システムコール	機　能	タスク部(T)／非タスク部(N)	仕様記載ページ
40.	preq_sem	セマフォ資源を得る	T	3-103
41.	ipreq_sem	同上（非タスク部用）	N	3-65
42.	sem_sts	セマフォの状態を参照する	T	3-118
43.	isem_sts	同上（非タスク部用）	N	3-75
44.	snd_msg	メイルボックスへ送信する	T	3-127
45.	isnd_msg	同上（非タスク部用）	N	3-83
46.	rcv_msg	メイルボックスからの受信を待つ	T	3-105
47.	prcv_msg	メイルボックスから受信する	T	3-101
48.	iprcv_msg	同上（非タスク部用）	N	3-63
49.	mbx_sts	メイルボックスの状態を参照する	T	3-93
50.	imbx_sts	同上（非タスク部用）	N	3-54
割込み管理機能				
51.	ret_int	割込みハンドラから復帰する	N	3-112
52.	chg_ims	割込みマスクを変更する	T	3-13
53.	ichg_ims	同上（非タスク部用）	N	3-38
54.	ims_sts	割込みマスクを参照する	T	3-58
55.	iims_sts	同上（非タスク部用）	N	3-53
例外管理機能				
56.	ret_exc*	例外処理から復帰する	T/N	3-111
メモリプール管理機能				
57.	get_blk	固定長メモリブロックの獲得待ちを行なう	T	3-27
58.	pget_blk	固定長メモリブロックを獲得する	T	3-97
59.	ipget_blk	同上（非タスク部用）	N	3-59
60.	rel_blk	固定長メモリブロックを返却する	T	3-107
61.	irel_blk	同上（非タスク部用）	N	3-67
62.	mpl_sts	メモリプールの状態を参照する	T	3-95
63.	impl_sts	同上（非タスク部用）	N	3-56
時間管理機能				
64.	set_tim	システムクロックを設定する	T	3-122
65.	iset_tim	同上（非タスク部用）	N	3-79
66.	get_tim	システムクロックの値を読み出す	T	3-30
67.	iget_tim	同上（非タスク部用）	N	3-47
68.	sys_clk*	カーネルの時間管理処理を要求する	N	3-135
システム管理機能				
69.	get_ver	カーネルのバージョン番号を得る	T	3-32
70.	iget_ver	同上（非タスク部用）	N	3-49
拡張SVC機能				
71.	sys_cal*	拡張SVCの発行	T	3-133

【注】*HI-SH77の独自仕様の機能です。その他の機能はμITRON仕様に準拠しています。

３．２　システムコールインタフェース

システムコールは、**C**言語およびアセンブリ言語プログラムから発行できます。

本節では、システムコール発行法の概略について説明します。次節に、システムコール別に発行法と機能を詳細に記述します。

３．２．１　C言語インタフェース

(1)システムコール発行形式

C言語プログラムからシステムコールを発行する場合、原則として次の形式になります。

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

ercd	: 関数のリターン値でエラーコード（符号付き**32**ビット整数）
<name>	: システムコール名
<return parameter address>	: リターンパラメータ用のアドレス（ポインタ）
<parameter>	: パラメータ
void	: リターン値が返らない関数

(2)パラメータ名称

パラメータ名称は、本書では統一的な省略名を使用しており、次の接頭語を付けています。

t_	構造体を示す
E_	エラーコードを示す
p_	ポインタを示す
pk_	パケットアドレスを示す
ppk_	パケットアドレスのポインタを示す
～cd	コードを示す
i～	初期値を示す
～sz	サイズを示す

(3)パラメータのデータ型とサイズ

パラメータのデータ型とサイズは、以下のものを使用します。

```
typedef   char            B;          /*  符号付き８ビット整数                    */
typedef   short           H;          /*  符号付き１６ビット整数                  */
typedef   long            W;          /*  符号付き３２ビット整数                  */
typedef   unsigned char   UB;         /*  符号なし８ビット整数                    */
typedef   unsigned short  UH;         /*  符号なし１６ビット整数                  */
typedef   unsigned long   UW;         /*  符号なし３２ビット整数                  */
typedef   char            VB;         /*  データ型が一定しない（８ビットサイズ）  */
typedef   short           VH;         /*  データ型が一定しない（１６ビットサイズ）*/
typedef   long            VW;         /*  データ型が一定しない（３２ビットサイズ）*/
typedef   void            *VP;        /*  データ型が一定しないものへのポインタ    */
typedef   void            (*FP)();    /*  プログラム先頭アドレス                  */
typedef   H               ID;         /*  オブジェクトID                          */
typedef   H               HNO;        /*  ハンドラ番号                            */
typedef   H               TPRI;       /*  タスク優先度                            */
typedef   W               ER;         /*  エラーコード                            */
typedef   int             FN;         /*  機能コード                              */
typedef   W               TMO;        /*  タイムアウト                            */
typedef   void            TASK;       /*  タスク                                  */
typedef   void            INTHDR;     /*  割込みハンドラ                          */
typedef   void            TMRHDR;     /*  タイマ割込みハンドラ                    */
typedef   void            INIHDR;     /*  初期化ハンドラ                          */
typedef   ER              SVCHDR;     /*  拡張SVCハンドラ                         */
typedef   TASK            (*TASKP)(); /*  タスク先頭アドレス                      */
typedef   INTHDR          (*INTHDRP)(); /*  割込みハンドラ先頭アドレス            */
typedef   TMRHDR          (*TMRHDR)(); /*  タイマ割込みハンドラ先頭アドレス       */
typedef   INIHDR          (*INIHDRP)(); /*  初期化ハンドラ先頭アドレス            */
typedef   SVCHDR          (*SVCHDRP)(); /*  拡張SVCハンドラ先頭アドレス           */
typedef   UW              SR;         /*  割込みマスク値                          */
```

大きな情報を受渡しするときは、情報をパケットにし、そのパケットの先頭アドレス(パケットアドレス)を設定します。

(4)システムコールの発行

タスク部用システムコールの**sta_tsk**システムコールを例にして発行法を示します。

Ｃ言語インターフェースは次のようになっています。

Ｃ言語インタフェース

ER ercd = sta_tsk(ID tskid); ・・・①

【パラメータ】

ID　　　　tskid　　　　タスクID ・・・②

【リターンパラメータ】

ER　　　　ercd　　　　エラーコード ・・・③

①sta_tsk システムコールを発行するステートメントで、引数は、 タスクID tskidです。
　関数のリターン値は、エラーコード ercd です。

②入力パラメータtskidのデータ型は、ID(符号付き16ビット整数)でタスクIDを設定します。

③リターンパラメータercdのデータ型は、ER(符号付き32ビット整数)でエラーコードを返します。

このため、システムコールの発行は以下のようになります。

```
#include <umachine.h>
#include "itron.h"

TASK    task()
{
ER      ercd;
ID      tskid;

        /* ...                  */
        ercd = sta_tsk(tskid);
        /* ...                  */
}
```

３．２．２　アセンブラインタフェース

(1)システムコール発行形式

アセンブリ言語のプログラムからのシステムコール発行は、原則として次の形式になります。

```
                MOV.L       lfncd,R0
                MOV.L       lpara1,R4
                MOV.L       lpara2,R5
                MOV.L       lpara3,R6
                MOV.L       lpara4,R7
                TRAPA       #n
        ;
                .ALIGN      4
        lfncd:  .DATA.L     functioncode
        lpara1: .DATA.L     paramater1
        lpara2: .DATA.L     paramater2
        lpara3: .DATA.L     paramater3
        lpara4: .DATA.L     paramater4
```

パラメータをレジスタに設定し、**TRAPA**命令を実行することにより、システムコールを発行します。システムコールからの戻りでは、エラーコードを返すレジスタ**(R0_BANK0),**バンク１のレジスタ**(R0_BANK1**～**R7_BANK1),SPC,SSR**以外のレジスタ(**R1_BANK0**～**R7_BANK0,R8**～**R15,SR,GBR,MACH,MACL, PR,VBR,PC**)の内容は保障されています**(**システムコール発行前と同じです**)**。

(2)パラメータ名称

C言語インタフェースで使用するものと同じ名称を用いています。

(3)パラメータの設定

システムコールのパラメータ受渡しはレジスタを用います。パラメータはすべて**32**ビットサイズです。

通常、次の目的で各レジスタを使用します。

- **R0**レジスタ　　機能コード、エラーコード
- **R4**レジスタ　　**ID**、その他のパラメータ
- **R5**レジスタ　　その他のパラメータ
- **R6**レジスタ　　その他のパラメータ
- **R7**レジスタ　　その他のパラメータ

大きな情報を受渡しするときは、情報をパケットにし、そのパケットの先頭アドレス**(**パケットアドレス**)**をレジスタに設定します。現在の年月日時刻データを返すパケットを例にしてパケットの構造の記述法を次に示します。

【パケットの構造】

[**--]	utime;	現在の年月日時刻データ（上位）
[****]	ltime;	現在の年月日時刻データ（下位）

[**--]は**4**バイト中上位**2**バイトが有効であることを示しています。[****]は**4**バイトすべてが有効であることを示しています。

(4)トラップ番号

システムコールを発行する**TRAPA**命令のトラップ番号は、プログラムのシステム状態により異なります。またシステムコールによっては、専用のトラップ番号が割り当てられている場合があります。

トラップ番号とシステム状態およびシステムコールの関係は次のようになっています。

- **TRAPA #57**　　**ret_exc**システムコール（例外処理のみから発行可能）
- **TRAPA #58**　　システム予約
- **TRAPA #59**　　システム予約
- **TRAPA #60**　　**sys_clk**システムコール（タイマハンドラのみから発行可能）
- **TRAPA #61**　　**ret_int**システムコール（割込みハンドラのみから発行可能）
- **TRAPA #62**　　非タスク部用システムコール
- **TRAPA #63**　　タスク部用システムコール

(5)アセンブリ言語インタフェース例

タスク部用システムコールの**sta_tsk**システムコールを例にして発行法を示します。

アセンブラインターフェースは次のようになっています。

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ（sta_tsk）・・・・・・・・・・・・・・・・・・・・・・・・①

【パラメータ】

R0	fncd	機能コード（H'fffffefe）・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・②
R4	tskid	タスクID・・・③

【リターンパラメータ】

R0	ercd	エラーコード・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・④

①TRAPA#63命令でシステムコールを発行します。

②機能コードとしてH'fffffefeをR0レジスタに設定します。

③タスクIDをR4レジスタに設定します。

④システムコールのエラーコードがR0レジスタに返されます。

R0 ＝ 0 : 正常終了した。

R0 ＜ 0 : エラーが発生した。

このため、システムコールの発行は以下のようになります。

```
        MOV.L       #H'fffffefe,R0
        MOV.L       #TSKI D,R4
        TRAPA       #63
;
```

３．２．３　エラーコード

エラーコードに設定される値はシステムコールにより異なります。**sta_tsk**システムコールを例に、エラーコードを以下に示します。

エラーコード			
E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[p]	予約ID（tskid = 0）
		[k]	未登録（タスクが生成されていない）
E_NODMT	H'fffff7ca(-H'836)	[k]	自タスク指定（tskid が自タスクである）
		[k]	タスクが休止（DORMANT）状態でない
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxtskid　：　セットアップテーブルで定義した最大タスクID

エラーチェック種別の"[p]"は、パラメータチェック機能付きカーネルの場合にのみチェックされて発行されるエラーコードであることを示しています。"[k]"は、パラメータチェック機能の有無に関係なく、カーネルでチェックされて発行されるエラーコードであることを示しています。

なお、エラーコードは、ステータスレジスタ(SR)には反映されません。

３．３　システムコール仕様

３．３．１　can_wup（Cancel Wakeup Task）タスクの起床要求を無効にする

［タスク部用］

C言語インタフェース

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

【パラメータ】

W	*p_wupcnt	キューイングされていた起床要求回数を返す領域の先頭アドレス
ID	tskid	タスクID(1～hi_maxtskid *1)

【リターンパラメータ】

ER	ercd	エラーコード
W	*p_wupcnt	キューイングされていた起床要求回数を格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffeef）
R4	p_wupcnt	キューイングされていた起床要求回数を返す領域の先頭アドレス
R5	tskid	タスクID(1～hi_maxtskid *1)

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_wupcnt	キューイングされていた起床要求回数を格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_wupcnt が4の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[k]	未登録（タスクが生成されていない）
E_DMT	H'fffff7cb(-H'835)	[k]	タスクが休止（DORMANT）状態である
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxtskid：　セットアップテーブルで定義した最大タスクID

解　　説

tskid で示されたタスクのキューイングされていた起床要求回数を求め、その結果を p_wupcnt の指す領域に返し、同時にその起床要求をすべて解除します。p_wupcnt の指す領域として4バイトのRAM領域が必要です。

tskid = TSK_SELF(H'0)の指定により、自タスクの指定が行なえます。

本システムコールは、時間内に処理が終わっているかどうか（前の起床要求に対するslp_tskシステムコールが実行される前に、次の起床要求が発生していないか）を判定する場合に利用できます。

リターンパラメータ p_wupcnt の指す領域が 0 でなければ、前の起床要求に対する処理が時間内に終了しなかったということがわかるので、それに対して何らかの処置をすることができます。

関連システムコール

ican_wup,iwup_tsk,slp_tsk,wai_tsk,wup_tsk

３．３．２　chg_ims（Change Interrupt Mask Level）割込みマスクを変更する

［タスク部用］

C言語インタフェース

ER ercd = chg_ims(SR imask);

【パラメータ】

SR	imask	割込みマスク

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffe1）
R4	imask	割込みマスク

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_IMS	H'fffff8d5(-H'72b)	[p]	不正IMASK(imask にSR_IMSn(n:00～15)以外の値を指定した）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

解　説

現在の割込みマスクを imask で指定した値に変更します。

本システムコールにより、割込みマスク値を 0～15 に変更できますが、 imask の指定はSRのデータタイプを使用して行ないますので、指定できる値は、SR_IMSn(n:00～15)となります。n が割込みマスク値に対応します。

SR_IMSnの具体的な値を以下に示します。

SR_IMS00	H'00000000
SR_IMS01	H'00000001
SR_IMS02	H'00000002
:	:
SR_IMS15	H'0000000f

imask としてSR_IMS00を指定した場合は、割込みマスクを解除します。

本システムコールにより割込みをマスクした場合は、非タスク部として実行されます。したがって、待ち（WAIT）状態へ移行するシステムコールは発行できません（エラーコードとしてE_CTXを返します）。

本システムコールによってタスク部から非タスク部に遷移した場合、タスク部に戻るには**ichg_ims**システムコールを用いて割込みマスクを解除する必要があります。非タスク部として実行中に、システムコールを発行してタスク切り換えが必要になっても、それは**ichg_ims**システムコールによって割込みマスクを0に戻すまで（タスク部に戻るまで）遅延されます。

割込みマスクレベルを、セットアップテーブルに定義したカーネル割込みマスクレベルより高く変更している間は、**ichg_ims**システムコールを用いてカーネル割込みマスクレベル以下に下げる場合と**ret_int, ret_exc**システムコールを除き、システムコールは発行できません。発行した場合、システムの正常な動作は保証されません。

関連システムコール

ichg_ims,iims_sts,ims_sts,ret_int

３．３．３　chg_pri（Change task Priority）タスク優先度を変更する［タスク部用］

C言語インタフェース

ER ercd = chg_pri(ID tskid, TPRI tskpri);

【パラメータ】

ID	tskid	タスクID（1～hi_maxtskid[*1]）
TPRI	tskpri	タスク優先度（0～hi_maxtskpri[*2]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffef9）
R4	tskid	タスクID（1～hi_maxtskid[*1]）
R5	tskpri	タスク優先度（0～hi_maxtskpri[*2]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_TPRI	H'fffff8da(-H'726)	[p]	不正タスク優先度 （tskpri < 0, tskpri > hi_maxtskpri[*2]）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[k]	未登録（タスクが生成されていない）
E_DMT	H'fffff7cb(-H'835)	[k]	タスクが休止（DORMANT）状態である
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxtskid　：　セットアップテーブルで定義した最大タスクID

*2　hi_maxtskpri　：　セットアップテーブルで定義した最大タスク優先度

解　説

tskid で示されたタスクの現在のタスク優先度を、tskpri で示された値に変更します。

tskid には、タスク優先度を変更するタスクのIDを指定します。tskid = TSK_SELF(H'0) の指定により、自タスクの指定が行なえます。

tskpri は、タスク優先度として 1～hi_maxtskpri（最大タスク優先度 ）の値を指定できます。タスク優先度は、値が小さいほどタスクの優先度が高くなります。tskpri = TPRI_INI(H'0) の指定により、cre_tskシステムコール発行時（またはセットアップテーブルにおける初期登録タスクの定義）に指定した、タスクの初期タスク優先度に戻します。

変更したタスク優先度は、タスクが終了するまで、または再び本システムコールを発行するまで有効です。タスクが休止（DORMANT）状態になると終了前のタスク優先度は無効になり、次に起動されたときにはcre_tskシステムコール発行時（またはセットアップテーブルにおける初期登録タスクの定義）に指定されたタスクの初期タスク優先度が有効になります。

関連システムコール

ichg_pri

３．３．４　clr_flg（Clear Event Flag）イベントフラグをクリアする

［タスク部用］

C言語インタフェース

ER ercd = clr_flg(ID flgid, UW clrptn);

【パラメータ】

ID	flgid	イベントフラグID（1～hi_maxflgid[*1]）
UW	clrptn	クリアするビットパターン

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffeed）
R4	flgid	イベントフラグID（1～hi_maxflgid[*1]）
R5	clrptn	クリアするビットパターン

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（flgid < 0, flgid > hi_maxflgid[*1]）
		[p]	予約ID（flgid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxflgid　：　セットアップテーブルで定義した最大イベントフラグID

解　説

flgid で示されたイベントフラグ（３２ビット）のうち、clrptn が 0 になっているビットをクリアします。すなわち、flgid で示されたイベントフラグに対して clrptn の値で論理積をとります。

本システムコールにおいて、イベントフラグ値の変更の結果、そのイベントフラグを待っているタスクが待ち解除となることはありません。

関連システムコール

flg_sts,iclr_flg,iflg_sts,ipol_flg,iset_flg,pol_flg,set_flg,wai_flg

３．３．５　cre_tsk（Create Task）タスクを生成する［タスク部用］

C言語インタフェース

ER ercd = cre_tsk(ID tskid, TASKP stadr, TPRI itskpri);

【パラメータ】

ID	tskid	タスクID（1～hi_maxtskid[*1]）
TASKP	stadr	タスク開始アドレス
TPRI	itskpri	初期タスク優先度（1～hi_maxtskpri[*2]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffeff）
R4	tskid	タスクID（1～hi_maxtskid[*1]）
R5	stadr	タスク開始アドレス
R6	itskpri	初期タスク優先度（1～hi_maxtskpri[*2]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_RSID	H'fffff9e9(-H'617)	[p]	予約ID（tskid = 0）
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（stadr が奇数または0）
E_IDOVR	H'fffff8dd(-H'723)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
E_TPRI	H'fffff8da(-H'726)	[p]	不正タスク優先度 （itskpri < 1, itskpri > hi_maxtskpri[*2]）
E_EXS	Hfffff7cd(-H'833)	[k]	自タスク指定（tskid が自タスクである）
		[k]	オブジェクトが存在している（tskid のタスクが存在）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxtskid　：　セットアップテーブルで定義した最大タスクID

*2　hi_maxtskpri　：　セットアップテーブルで定義した最大タスク優先度

解　説

tskid で示されたIDを持つタスクを生成します。すなわち、タスクプログラムをカーネルに登録します。生成したタスクは、未登録（NON-EXISTENT）状態から休止（DORMANT）状態へ移行します。

生成したタスクのスタックは、セットアップテーブルでタスクID毎に定義した静的な領域を使用します。

tskid には、セットアップテーブルで定義した 1～hi_maxtskid（最大タスクID）の値を指定できます。

stadr には、タスクの開始アドレスを指定します。

itskpri には、タスク生成時の初期タスク優先度として 1～hi_maxtskpri（最大タスク優先度 ）の値を指定できます。タスク優先度は、値が小さいほどタスクの優先度が高くなります。

本システムコールは、HI-SH77の独自仕様です。

関連システムコール

del_tsk

３．３．６　del_tsk（Delete Task）タスクを削除する［タスク部用］

C言語インタフェース

ER ercd = del_tsk(ID tskid);

【パラメータ】

ID	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffefd）
R4	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[p]	予約ID（tskid = 0）
		[k]	未登録（タスクが生成されていない）
E_NODMT	H'fffff7ca(-H'836)	[k]	自タスク指定（tskid が自タスクである）
		[k]	タスクが休止（DORMANT）状態でない
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxtskid　：　セットアップテーブルで定義した最大タスクID

解　説

tskid で示されたタスクを削除します。

削除したタスクは、休止（DORMANT）状態から未登録（NON-EXISTENT）状態へ移行します。タスクを削除することで、そのIDで新たなタスクを生成することができます。

本システムコールで削除することができるタスクは、休止状態のタスクです。

休止状態以外のタスクを指定すると、発行タスクにエラーコードとしてE_NODMTを返します。

tskidには、削除する他タスクのIDを指定します。tskid に自タスクのIDを指定した場合は、エラーコードとしてE_NODMTを返します。自タスクを削除する場合は、exd_tskシステムコールを発行してください。

本システムコールは、HI-SH77の独自仕様です。

関連システムコール

cre_tsk,exd_tsk

３．３．７　exd_tsk（Exit and Delete Task）自タスクを正常終了後、削除する［タスク部用］

C言語インタフェース

void exd_tsk(void);

【パラメータ】

なし

【リターンパラメータ】

本システムコールの発行元には戻らない

アセンブラインタフェース

TRAPA　　#63　　システムコール用トラップ

【パラメータ】

R0　　fncd　　機能コード（H'fffffefb）

【リターンパラメータ】

本システムコールの発行元には戻らない

エラーコード

E_CTX　　H'fffff5bb(-H'a45)　　[k]　　コンテキストエラー*1（非タスク部から発行できない）

*1　エラー検出時は、R4に-2（異常終了種別）、R5にエラーコード、R6にexd_tskシステムコール発行アドレス+2を設定し、システム異常終了処理（hi_sysdwn関数）に制御を渡します。

解　説

自タスクを正常終了し、さらに自タスクを削除します。

正常終了、削除した自タスクは、実行（RUN）状態から未登録（NON-EXISTENT）状態へ移行します。

本システムコールは、タスクが占有していた資源（セマフォにより獲得したもの）とメモリブロックを自動的に解放する機能はありません。資源とメモリブロックの解放は、本システムコール発行前に行なってください。

本システムコールの発行により、スタックを解放します。このスタックの待ち行列に他のタスクがつながれている場合（共有スタック）、先頭のタスクをスタックの待ち行列から外し、実行可能（READY）状態へ移行します。

本システムコールは、HI-SH77の独自仕様です。

関連システムコール

del_tsk,ext_tsk

３．３．８　ext_tsk（Exit Task）自タスクを正常終了する［タスク部用］

C言語インタフェース

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

【パラメータ】

なし

【リターンパラメータ】

本システムコールの発行元には戻らない

アセンブラインタフェース

```
TRAPA        #63          システムコール用トラップ
```

【パラメータ】

```
R0           fncd         機能コード（H'fffffefc）
```

【リターンパラメータ】

本システムコールの発行元には戻らない

エラーコード

```
E_CTX        H'fffff5bb(-H'a45)      [k]    コンテキストエラー*1（非タスク部から発行できない）
```

*1　エラー検出時は、R4に-1（異常終了種別）、R5にエラーコード、R6にext_tskシステムコール発行アドレス+2を設定し、システム異常終了処理（hi_sysdwn関数）に制御を渡します。

解　　説

自タスクを正常終了します。

正常終了した自タスクは、実行（RUN）状態から休止（DORMANT）状態へ移行します。

本システムコールは、タスクが占有していた資源（セマフォにより獲得したもの）とメモリブロックを自動的に解放する機能はありません。資源とメモリブロックの解放は、本システムコール発行前に行なってください。

本システムコールの発行により、スタックを解放します。このスタックの待ち行列に他のタスクがつながれている場合（共有スタック）、先頭のタスクをスタックの待ち行列から外し、実行可能（READY）状態へ移行します。

関連システムコール

exd_tsk,ter_tsk

３．３．９　flg_sts（Get Event Flag Status）イベントフラグの状態を参照する

［タスク部用］

C言語インタフェース

ER ercd = flg_sts(ID *p_wtskid, UW *p_flgptn, ID flgid);

【パラメータ】

ID	*p_wtskid	待ちタスクIDを返す領域の先頭アドレス
UW	*p_flgptn	イベントフラグのビットパターンを返す領域の先頭アドレス
ID	flgid	イベントフラグID（1～hi_maxflgid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード
ID	*p_wtskid	待ちタスクIDを格納した領域の先頭アドレス
UW	*p_flgptn	イベントフラグのビットパターンを格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffea）
R4	p_wtskid	待ちタスクIDを返す領域の先頭アドレス
R5	p_flgptn	イベントフラグのビットパターンを返す領域の先頭アドレス
R6	flgid	イベントフラグID（1～hi_maxflgid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_wtskid	待ちタスクIDを格納した領域の先頭アドレス
R5	p_flgptn	イベントフラグのビットパターンを格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_wtskid が奇数または0、 p_flgptn が 4 の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（flgid < 0, flgid > hi_maxflgid[*1]）
		[p]	予約ID（flgid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxflgid　：　セットアップテーブルで定義した最大イベントフラグID

解　説

flgid で示されたイベントフラグ（３２ビット）の各種状態を参照し、対象イベントフラグの現在のイベントフラグ値と待ちタスクIDを求め、その結果をそれぞれ p_flgptn, p_wtskid の指す領域に返します。p_wtskid の指す領域として 2 バイト、p_flgptn の指す領域として 4 バイトのRAM領域が必要です。

なお、対象イベントフラグの待ちタスクが無い場合は、待ちタスクIDとしてFALSE(H'0)を返します。

関連システムコール

clr_flg,iclr_flg,iflg_sts,ipol_flg,iset_flg,pol_flg,set_flg,wai_flg

３．３．１０　get_blk（Get Memory Block）固定長メモリブロックの獲得待ちを行なう

［タスク部用］

Ｃ言語インタフェース

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

【パラメータ】

VP	*p_blk	メモリブロックの先頭アドレスを返す領域の先頭アドレス
ID	mplid	メモリプールID（1～hi_maxmplid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード
VP	*p_blk	メモリブロックの先頭アドレスを格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffedf）
R4	p_blk	メモリブロックの先頭アドレスを返す領域の先頭アドレス
R5	mplid	メモリプールID（1～hi_maxmplid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_blk	メモリブロックの先頭アドレスを格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_blk が 4 の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（mplid < 0, mplid > hi_maxmplid[*1]）
		[p]	予約ID（mplid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）
E_RLWAI	H'fffff2aa(-H'd56)	[k]	待ち（WAIT）状態強制解除（待ちの間にrel_waiまたはirel_waiシステムコールが発行された）

*1　hi_maxmplid　：　セットアップテーブルで定義した最大メモリプールID

解　説

mplid で示される固定長メモリプールから 1 つの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 p_blk の指す領域に返します。指定したメモリプールからメモリブロックを獲得できない場合には、発行タスクはそのメモリプールのメモリ獲得の待ち行列につながれます。待ち行列のつながれ方は FIFO(First-In First-Out)です。

p_blk の指す領域として 4 バイトのRAM領域が必要です。

関連システムコール

iget_blk,impl_sts,ipget_blk,irel_blk,irel_wai,mpl_sts,pget_blk,rel_blk,rel_wai

３．３．１１　get_tid（Get Task Identifier）自タスク IDを得る［タスク部用］

C言語インタフェース

ER ercd = get_tid(ID *p_tskid);

【パラメータ】

ID	*p_tskid	タスクIDを返す領域の先頭アドレス

【リターンパラメータ】

ER	ercd	エラーコード
ID	*p_tskid	タスクIDを格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffef6）
R4	p_tskid	タスクIDを返す領域の先頭アドレス

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_tskid	タスクIDを格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_tskid が奇数または0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

解　　説

自タスクのIDを得ることができます。

発行タスク（自タスク）のタスクIDを求め、その結果を p_tskid の指す領域に返します。p_tskid の指す領域として、2 バイトのRAM領域が必要です。

関連システムコール

iget_tid

３．３．１２　get_tim（Get Time）システムクロックの値を読み出す

［タスク部用］

Ｃ言語インタフェース

```
ER  ercd = get_tim(T_TIM *pk_time);
```

【パラメータ】

T_TIM	*pk_time	現在の年月日時刻データを返すパケットの先頭アドレス

【リターンパラメータ】

ER	ercd	エラーコード
T_TIM	*pk_time	現在の年月日時刻データを格納したパケットの先頭アドレス

【パケットの構造】

```
typedef     struct      t_tim {
            H           utime;      現在の年月日時刻データ（上位）
            UW          ltime;      現在の年月日時刻データ（下位）
            } T_TIM;
```

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffeda）
R4	pk_time	現在の年月日時刻データを返すパケットの先頭アドレス

【リターンパラメータ】

R0	ercd	エラーコード
R4	pk_time	現在の年月日時刻データを格納したパケットの先頭アドレス

【パケットの構造】

[**--]	utime	現在の年月日時刻データ（上位）
[****]	ltime	現在の年月日時刻データ（下位）

エラーコード

E_OK	H'00000000	[k]	正常終了
E_TNOSPT	H'fffff9ed(-H'613)	[p]	時間管理機能が組み込まれていない
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（pk_time が4の倍数以外または0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

解　　説

システムが保持している現在のシステムクロックの値を読み出し、その結果を pk_time の指す領域に返します。pk_time の指す領域として 8バイトのRAM領域が必要です。

システムクロックは、48ビット（utime:16bit + ltime:32bit）で表現されます。

関連システムコール

iget_tim,iset_tim,set_tim,sys_clk

３．３．１３　get_ver（Get Version No）カーネルのバージョン番号を得る

［タスク部用］

Ｃ言語インタフェース

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

【パラメータ】

T_VER	*pk_ver	バージョン管理情報を返すパケットの先頭アドレス

【リターンパラメータ】

ER	ercd	エラーコード
T_VER	*pk_ver	バージョン管理情報を格納したパケットの先頭アドレス

【パケットの構造】

```
typedef     struct      t_ver {
            UH          maker;          メーカー
            UH          id;             形式番号
            UH          spver;          仕様書バージョン
            UH          prver;          製品バージョン
            UH          prno[4];        製品管理情報
            UH          cpu;            CPU情報
            UH          var;            バリエーション記述子
            } T_VER;
```

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffed9）
R4	pk_ver	バージョン管理情報を返すパケットの先頭アドレス

【リターンパラメータ】

R0	ercd	エラーコード
R4	pk_ver	バージョン管理情報を格納したパケットの先頭アドレス

【パケットの構造】

[**]	maker	メーカー
[**]	id	形式番号
[**]	spver	仕様書バージョン
[**]	prver	製品バージョン
[**]	prno[0]	製品管理情報
[**]	prno[1]	製品管理情報
[**]	prno[2]	製品管理情報
[**]	prno[3]	製品管理情報
[**]	cpu	CPU情報
[**]	var	バリエーション記述子

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（pk_ver が 奇数または0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

解　説

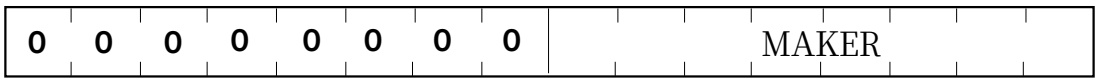
現在実行中のカーネルのバージョンに関する諸情報を読み出し、その結果を pk_ver の指す領域に返します。pk_ver の指す領域として、20バイトのRAM領域が必要です。

pk_ver の指すパケットには、次の情報を返します。

(maker)

カーネルの maker の値は H'000a です。

以下に maker のフォーマットを示します。

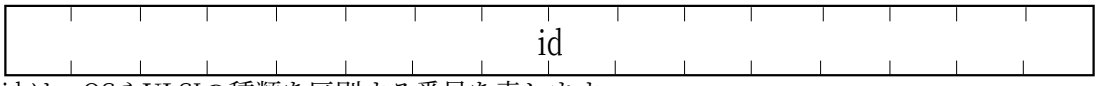


MAKER は、この製品を作ったメーカを表します。（H'0a : HITACHI）

(id)

カーネルの id の値は H'0004 です。

以下に id のフォーマットを示します。

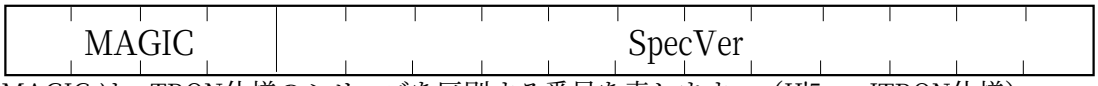


id は、OSやVLSIの種類を区別する番号を表します。

(spver)

カーネルの spver の値は H'5202 です。

以下に spver のフォーマットを示します。



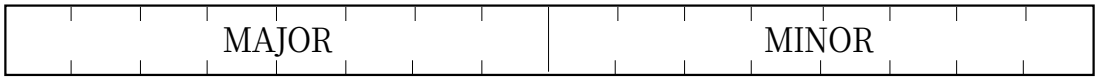
MAGIC は、TRON仕様のシリーズを区別する番号を表します。（H'5 : μITRON仕様）

SpecVer は、この製品のもとになった東京大学のTRON仕様書のバージョン番号を3桁のBCDコードで表します。（H'202 : Ver2.02）

(prver)

カーネルの prver の値は H'0100 です。

以下に prver のフォーマットを示します。



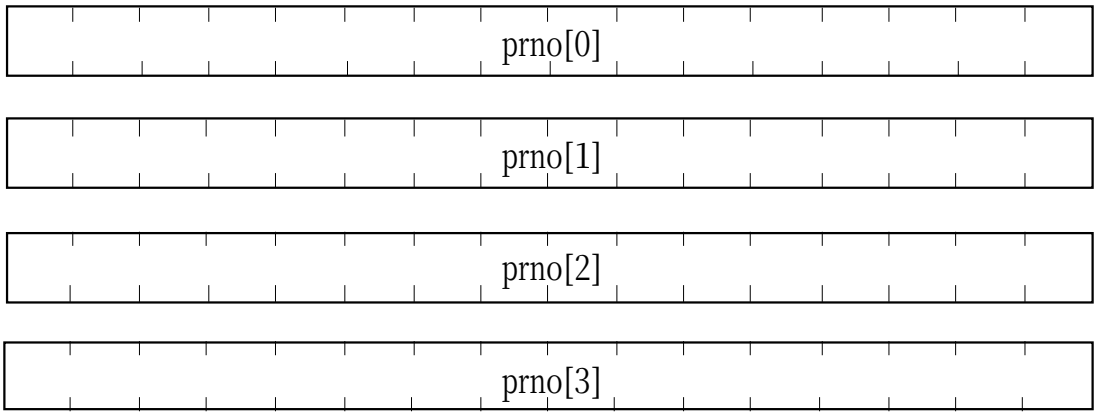
MAJOR は、バージョン番号の大きな区別を表します。

MINOR は、バージョン番号の小さな区別を表します。

(prno)

カーネルの prno[0] から prno[3] の値は H'0000 です。

以下に prno のフォーマットを示します。

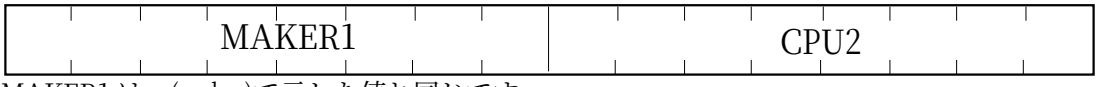


製品管理情報や製品番号などを表します。

(cpu)

カーネルの cpu の値は H'0a77 です。

以下に cpu のフォーマットを示します。



MAKER1 は、(maker)で示した値と同じです

CPU2 は、この製品を実行するプロセッサを表します。（H'77 : SH3）

(var)

カーネルの var の値は H'0000 です。

以下に var のフォーマットを示します。

－	LEV		－	M	V	P	－	－	FIL	IO	－	－

ー　：　未使用（B'0）

LEV　：　カーネル仕様のレベル分け（B'000 : μITRON仕様）

M　：　シングルプロセッサ用（B'0）

V　：　仮想記憶未サポート（B'0）

P　：　non-MMU 対応版（B'0）

FIL　：　ファイル仕様のレベル分け（B'00 : サポートなし）

IO　：　入出力仕様（B'00 : サポートなし）

関連システムコール

iget_ver

３．３．１４　ican_wup（Interrupt Cancel Wakeup Task）タスクの起床要求を無効にする

［非タスク部用］

C言語インタフェース

```
ER  ercd = ican_wup(W *p_wupcnt, ID tskid);
```

【パラメータ】

W	*p_wupcnt	キューイングされていた起床要求回数を返す領域の先頭アドレス
ID	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード
W	*p_wupcnt	キューイングされていた起床要求回数を格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffff6）
R4	p_wupcnt	キューイングされていた起床要求回数を返す領域の先頭アドレス
R5	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_wupcnt	キューイングされていた起床要求回数を格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_wupcnt が4の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[p]	予約ID（tskid = 0）
		[k]	未登録（タスクが生成されていない）
E_DMT	H'fffff7cb(-H'835)	[k]	タスクが休止（DORMANT）状態である
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

*1　hi_maxtskid：　セットアップテーブルで定義した最大タスクID

解　　説

tskid で示されたタスクのキューイングされていた起床要求回数を求め、その結果を p_wupcnt の指す領域に返し、同時にその起床要求をすべて解除します。p_wupcnt の指す領域として4バイトのRAM領域が必要です。

本システムコールは、時間内に処理が終わっているかどうか（前の起床要求に対するslp_tskシステムコールが実行される前に、次の起床要求が発生していないか）を判定する場合に利用できます。

リターンパラメータ p_wupcnt の指す領域が 0 でなければ、前の起床要求に対する処理が時間内に終了しなかったということがわかるので、それに対して何らかの処置をすることができます。

関連システムコール

can_wup,iwup_tsk,slp_tsk,wai_tsk,wup_tsk

３．３．１５ ichg_ims（Interrupt Change Interrupt Mask Level）割込みマスクを変更する

［非タスク部用］

Ｃ言語インタフェース

```
ER  ercd = ichg_ims(SR imask);
```

【パラメータ】

SR	imask	割込みマスク

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffeb）
R4	imask	割込みマスク

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_IMS	H'fffff8d5(-H'72b)	[p]	不正IMASK(imask にSR_IMSn(n:00～15)以外の値を指定した）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

解　説

現在の割込みマスクを imask で指定した値に変更します。

本システムコールにより、割込みマスク値を 0～15 に変更できますが、 imask の指定はSRのデータタイプを使用して行ないますので、指定できる値は、SR_IMSn(n:00～15)となります。n が割込みマスク値に対応します。

SR_IMSnの具体的な値を以下に示します。

SR_IMS00	H'00000000
SR_IMS01	H'00000001
SR_IMS02	H'00000002
:	:
SR_IMS15	H'0000000f

imask としてSR_IMS00を指定した場合は、割込みマスクを解除します。

chg_imsシステムコールにより割込みをマスクした場合は、非タスク部として実行されます。したがって、待ち（WAIT）状態へ移行するシステムコールは発行できません（エラーコードとしてE_CTXを返します）。

chg_imsシステムコールによってタスク部から非タスク部に遷移した場合、タスク部に戻るには本システムコールを用いて割込みマスクを解除する必要があります。非タスク部として実行中に、システムコールを発行してタスク切り換えが必要になっても、それは本システムコールによって割込みマスクを0に戻すまで（タスク部に戻るまで）遅延されます。

割込みマスクレベルを、セットアップテーブルに定義したカーネル割込みマスクレベルより高く変更している間は、本システムコールを用いてカーネル割込みマスクレベル以下に下げる場合と**ret_int,ret_exc**システムコールを除き、システムコールは発行できません。発行した場合、システムの正常な動作は保証されません。

関連システムコール

chg_ims,iims_sts,ims_sts,ret_int

３．３．１６ ichg_pri（Interrupt Change task Priority）タスク優先度を変更する

［非タスク部用］

C言語インタフェース

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

【パラメータ】

ID	tskid	タスクID（1～hi_maxtskid[*1]）
TPRI	tskpri	タスク優先度（0～hi_maxtskpri[*2]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffffe）
R4	tskid	タスクID（1～hi_maxtskid[*1]）
R5	tskpri	タスク優先度（0～hi_maxtskpri[*2]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_TPRI	H'fffff8da(-H'726)	[p]	不正タスク優先度 （tskpri < 0, tskpri > hi_maxtskpri[*2]）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[p]	予約ID（tskid = 0）
		[k]	未登録（タスクが生成されていない）
E_DMT	H'fffff7cb(-H'835)	[k]	タスクが休止（DORMANT）状態である
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

*1 hi_maxtskid ： セットアップテーブルで定義した最大タスクID

*2 hi_maxtskpri ： セットアップテーブルで定義した最大タスク優先度

解　説

tskid で示されたタスクの現在のタスク優先度を、tskpri で示された値に変更します。

tskpri は、タスク優先度として 1～hi_maxtskpri（最大タスク優先度 ）の値を指定できます。タスク優先度は、値が小さいほどタスクの優先度が高くなります。tskpri = TPRI_INI(H'0) の指定により、cre_tskシステムコール発行時（またはセットアップテーブルにおける初期登録タスクの定義）に指定した、タスクの初期タスク優先度に戻します。

変更したタスク優先度は、タスクが終了するまで、または再び本システムコールを発行するまで有効です。タスクが休止（DORMANT）状態になると終了前のタスク優先度は無効になり、次に起動されたときにはcre_tskシステムコール発行時（またはセットアップテーブルにおける初期登録タスクの定義）に指定されたタスクの初期タスク優先度が有効になります。

関連システムコール

chg_pri

３．３．１７ iclr_flg（Interrupt Clear Event Flag）イベントフラグをクリアする

［非タスク部用］

C言語インタフェース

ER ercd = iclr_flg(ID flgid, UW clrptn);

【パラメータ】

ID	flgid	イベントフラグID（1～hi_maxflgid[*1]）
UW	clrptn	クリアするビットパターン

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffff4）
R4	flgid	イベントフラグID（1～hi_maxflgid[*1]）
R5	clrptn	クリアするビットパターン

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（flgid < 0, flgid > hi_maxflgid[*1]）
		[p]	予約ID（flgid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

*1 hi_maxflgid ： セットアップテーブルで定義した最大イベントフラグID

解　説

flgid で示されたイベントフラグ（３２ビット）のうち、clrptn が 0 になっているビットをクリアします。すなわち、flgid で示されたイベントフラグに対して clrptn の値で論理積をとります。

本システムコールにおいて、イベントフラグ値の変更の結果、そのイベントフラグを待っているタスクが待ち解除となることはありません。

関連システムコール

clr_flg,flg_sts,iflg_sts,ipol_flg,iset_flg,pol_flg,set_flg,wai_flg

３．３．１８　iflg_sts（Interrupt　Get Event　Flag Status）イベントフラグの状態を参照する

［非タスク部用］

C言語インタフェース

ER　ercd = iflg_sts(ID *p_wtskid, UW *p_flgptn, ID flgid);

【パラメータ】

ID	*p_wtskid	待ちタスクIDを返す領域の先頭アドレス
UW	*p_flgptn	イベントフラグのビットパターンを返す領域の先頭アドレス
ID	flgid	イベントフラグID（1～hi_maxflgid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード
ID	*p_wtskid	待ちタスクIDを格納した領域の先頭アドレス
UW	*p_flgptn	イベントフラグのビットパターンを格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffff2）
R4	p_wtskid	待ちタスクIDを返す領域の先頭アドレス
R5	p_flgptn	イベントフラグのビットパターンを返す領域の先頭アドレス
R6	flgid	イベントフラグID（1～hi_maxflgid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_wtskid	待ちタスクIDを格納した領域の先頭アドレス
R5	p_flgptn	イベントフラグのビットパターンを格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_wtskid が奇数または0、 p_flgptn が 4 の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（flgid < 0, flgid > hi_maxflgid[*1]）
		[p]	予約ID（flgid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

*1　hi_maxflgid　：　セットアップテーブルで定義した最大イベントフラグID

解　説

flgid で示されたイベントフラグ（３２ビット）の各種状態を参照し、対象イベントフラグの現在のイベントフラグ値と待ちタスクIDを求め、その結果をそれぞれ p_flgptn, p_wtskid の指す領域に返します。p_wtskid の指す領域として 2 バイト、p_flgptn の指す領域として 4 バイトのRAM領域が必要です。

なお、対象イベントフラグの待ちタスクが無い場合は、待ちタスクIDとしてFALSE(H'0)を返します。

関連システムコール

clr_flg,flg_sts,iclr_flg,ipol_flg,iset_flg,pol_flg,set_flg,wai_flg

３．３．１９ iget_tid（Interrupt Get Task Identifier）自タスク IDを得る［非タスク部用］

C言語インタフェース

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

【パラメータ】

ID	*p_tskid	タスクIDを返す領域の先頭アドレス

【リターンパラメータ】

ER	ercd	エラーコード
ID	*p_tskid	タスクIDを格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffffb）
R4	p_tskid	タスクIDを返す領域の先頭アドレス

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_tskid	タスクIDを格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_tskid が奇数または0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

解　説

本システムコールを発行した場合は、タスクIDとして FALSE(H'0) を返します。

関連システムコール

get_tid

３．３．２０　iget_tim（Interrupt　Get Time）システムクロックの値を読み出す

［非タスク部用］

Ｃ言語インタフェース

```
ER  ercd = iget_tim(T_TIM *pk_time);
```

【パラメータ】

T_TIM	*pk_time	現在の年月日時刻データを返すパケットの先頭アドレス

【リターンパラメータ】

ER	ercd	エラーコード
T_TIM	*pk_time	現在の年月日時刻データを格納したパケットの先頭アドレス

【パケットの構造】

```
typedef     struct      t_tim {
            H           utime;          現在の年月日時刻データ（上位）
            UW          ltime;          現在の年月日時刻データ（下位）
            } T_TIM;
```

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffe5）
R4	pk_time	現在の年月日時刻データを返すパケットの先頭アドレス

【リターンパラメータ】

R0	ercd	エラーコード
R4	pk_time	現在の年月日時刻データを格納したパケットの先頭アドレス

【パケットの構造】

[**--]	utime	現在の年月日時刻データ（上位）
[****]	ltime	現在の年月日時刻データ（下位）

エラーコード

E_OK	H'00000000	[k]	正常終了
E_TNOSPT	H'fffff9ed(-H'613)	[p]	時間管理機能が組み込まれていない
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（pk_time が4の倍数以外または0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

解　　説

システムが保持している現在のシステムクロックの値を読み出し、その結果を pk_time の指す領域に返します。pk_time の指す領域として 8バイトのRAM領域が必要です。

システムクロックは、48ビット（utime:16bit + ltime:32bit）で表現されます。

関連システムコール

get_tim,iset_tim,set_tim,sys_clk

３．３．２１　iget_ver（Interrupt　Get Version　No）カーネルのバージョン番号を得る

［非タスク部用］

Ｃ言語インタフェース

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

【パラメータ】

T_VER	*pk_ver	バージョン管理情報を返すパケットの先頭アドレス

【リターンパラメータ】

ER	ercd	エラーコード
T_VER	*pk_ver	バージョン管理情報を格納したパケットの先頭アドレス

【パケットの構造】

```
typedef     struct      t_ver {
            UH          maker;          メーカー
            UH          id;             形式番号
            UH          spver;          仕様書バージョン
            UH          prver;          製品バージョン
            UH          prno[4];        製品管理情報
            UH          cpu;            CPU情報
            UH          var;            バリエーション記述子
            } T_VER;
```

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffe4）
R4	pk_ver	バージョン管理情報を返すパケットの先頭アドレス

【リターンパラメータ】

R0	ercd	エラーコード
R4	pk_ver	バージョン管理情報を格納したパケットの先頭アドレス

【パケットの構造】

[**]	maker	メーカー
[**]	id	形式番号
[**]	spver	仕様書バージョン
[**]	prver	製品バージョン
[**]	prno[0]	製品管理情報
[**]	prno[1]	製品管理情報
[**]	prno[2]	製品管理情報
[**]	prno[3]	製品管理情報
[**]	cpu	CPU情報
[**]	var	バリエーション記述子

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（pk_ver が 奇数または0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

解　説

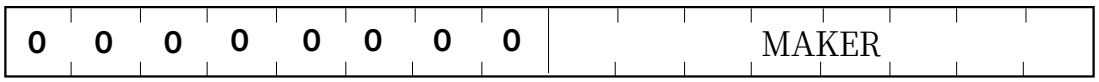
現在実行中のカーネルのバージョンに関する諸情報を読み出し、その結果を pk_ver の指す領域に返します。pk_ver の指す領域として、20バイトのRAM領域が必要です。

pk_ver の指すパケットには、次の情報を返します。

(maker)

カーネルの maker の値は H'000a です。

以下に maker のフォーマットを示します。

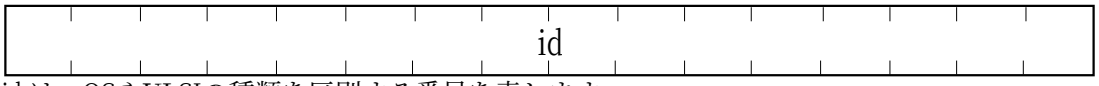


MAKER は、この製品を作ったメーカを表します。（H'0a : HITACHI）

(id)

カーネルの id の値は H'0004 です。

以下に id のフォーマットを示します。

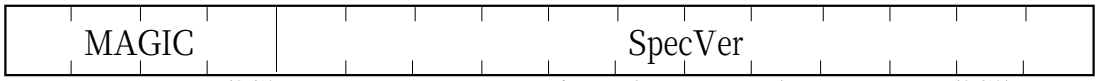


id は、OSやVLSIの種類を区別する番号を表します。

(spver)

カーネルの spver の値は H'5202 です。

以下に spver のフォーマットを示します。



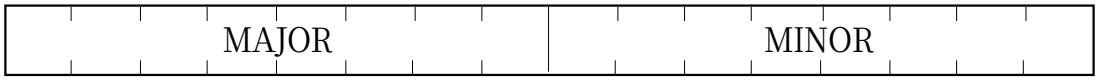
MAGIC は、TRON仕様のシリーズを区別する番号を表します。（H'5 : μITRON仕様）

SpecVer は、この製品のもとになった東京大学のTRON仕様書のバージョン番号を3桁のBCDコードで表します。（H'202 : Ver2.02）

(prver)

カーネルの prver の値は H'0100 です。

以下に prver のフォーマットを示します。



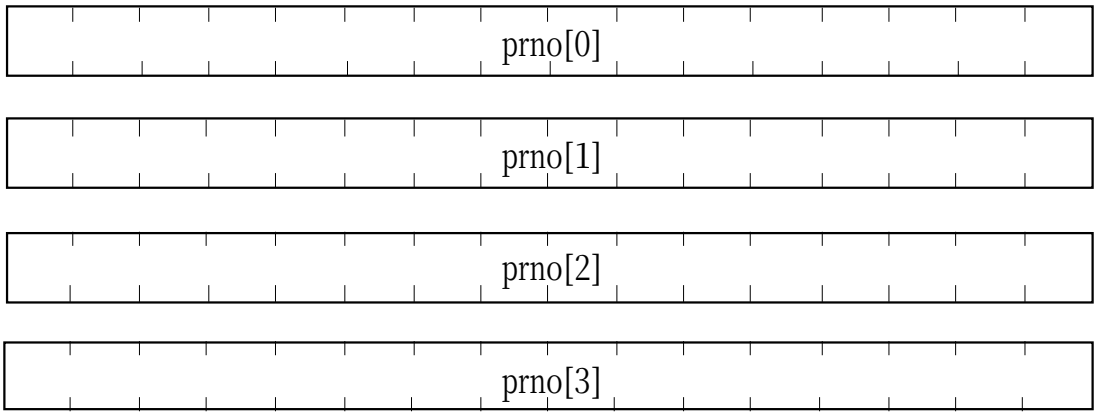
MAJOR は、バージョン番号の大きな区別を表します。

MINOR は、バージョン番号の小さな区別を表します。

(prno)

カーネルの prno[0] から prno[3] の値は H'0000 です。

以下に prno のフォーマットを示します。



製品管理情報や製品番号などを表します。

(cpu)

カーネルの cpu の値は H'0a77 です。

以下に cpu のフォーマットを示します。

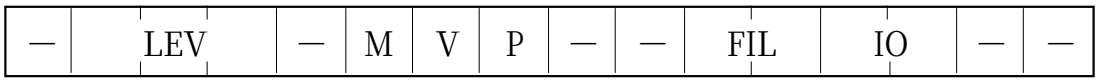
MAKER1 は、(maker)で示した値と同じです

CPU2 は、この製品を実行するプロセッサを表します。（H'77 : SH3）

(var)

カーネルの var の値は H'0000 です。

以下に var のフォーマットを示します。



ー　：　未使用（B'0）

LEV　：　カーネル仕様のレベル分け（B'000 : μITRON仕様）

M　：　シングルプロセッサ用（B'0）

V　：　仮想記憶未サポート（B'0）

P　：　non-MMU 対応版（B'0）

FIL　：　ファイル仕様のレベル分け（B'00 : サポートなし）

IO　：　入出力仕様（B'00 : サポートなし）

関連システムコール

get_ver

３．３．２２　iims_sts (Interrupt Get Interrupt Mask Level Status)

割込みマスクを参照する［非タスク部用］

C言語インタフェース

```
ER  ercd = iims_sts(SR *p_imask);
```

【パラメータ】

SR	*p_imask	割込みマスクを返す領域の先頭アドレス

【リターンパラメータ】

ER	ercd	エラーコード
SR	*p_imask	割込みマスクを格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffea）
R4	p_imask	割込みマスクを返す領域の先頭アドレス

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_imask	割込みマスクを格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_imask が 4 の倍数以外または0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

解　説

CPUのステータス・レジスタ（SR）の割込みマスクビット（Iビット）を参照し、割込みマスク値をp_imask の指す領域に返します。p_imask の指す領域として 4 バイトのRAM領域が必要です。

関連システムコール

chg_ims,ichg_ims,ims_sts,ret_int

３．３．２３　imbx_sts（Interrupt Get Mailbox Status）メイルボックスの状態を参照する

［非タスク部用］

C言語インタフェース

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

【パラメータ】

ID	*p_wtskid	待ちタスクIDを返す領域の先頭アドレス
T_MSG	**ppk_msg	次のメッセージの先頭アドレスを返す領域の先頭アドレス
ID	mbxid	メイルボックスID（1～hi_maxmbxid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード
ID	*p_wtskid	待ちタスクIDを格納した領域の先頭アドレス
T_MSG	**ppk_msg	次のメッセージの先頭アドレスを格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffec）
R4	p_wtskid	待ちタスクIDを返す領域の先頭アドレス
R5	ppk_msg	次のメッセージの先頭アドレスを返す領域の先頭アドレス
R6	mbxid	メイルボックスID（1～hi_maxmbxid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_wtskid	待ちタスクIDを格納した領域の先頭アドレス
R5	ppk_msg	次のメッセージの先頭アドレスを格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_wtskid が奇数または0, ppk_msg が 4 の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（mbxid < 0, mbxid > hi_maxmbxid[*1]）
		[p]	予約ID（mbxid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

*1　hi_maxmbxid　：　セットアップテーブルで定義した最大メイルボックスID

解　　説

mbxid で示されたメイルボックスの各種状態を参照し、対象メイルボックスの次に受信されるメッセージの先頭アドレスと待ち行列の先頭のタスクIDを求め、その結果をそれぞれ ppk_msg, p_wtskid の指す領域に返します。ppk_msg の指す領域として 4 バイト、 p_wtskidの指す領域として 2 バイトのRAM領域が必要です。

なお、次に受信されるメッセージが無い場合は、メッセージの先頭アドレスとしてNADR(H'ffffffff)を返します。

また、対象メイルボックスの待ちタスクが無い場合は、待ちタスクIDとしてFALSE(H'0)を返します。

関連システムコール

isnd_msg,iprcv_msg,mbx_sts,prcv_msg,rcv_msg,snd_msg

３．３．２４　impl_sts（Interrupt Get Memory Pool Status）メモリプールの状態を参照する

［非タスク部用］

C言語インタフェース

```
ER  ercd = impl_sts(ID *p_wtskid, W *p_frbcnt, ID mplid);
```

【パラメータ】

ID	*p_wtskid	待ちタスクIDを返す領域の先頭アドレス
W	*p_frbcnt	空き領域のブロック数を返す領域の先頭アドレス
ID	mplid	メモリプールID（1～hi_maxmplid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード
ID	*p_wtskid	待ちタスクIDを格納した領域の先頭アドレス
W	*p_frbcnt	空き領域のブロック数を格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffffe7）
R4	p_wtskid	待ちタスクIDを返す領域の先頭アドレス
R5	p_frbcnt	空き領域のブロック数を返す領域の先頭アドレス
R6	mplid	メモリプールID（1～hi_maxmplid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_wtskid	待ちタスクIDを格納した領域の先頭アドレス
R5	p_frbcnt	空き領域のブロック数を格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_wtskidが奇数または0, p_frbcnt が4の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（mplid < 0, mplid > hi_maxmplid[*1]）
		[p]	予約ID（mplid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

*1　hi_maxmplid　：　セットアップテーブルで定義した最大メモリプールID

解　　説

mplid で示されたメモリプールの各種状態を参照し、対象メモリプールの現在の空きブロック数と待ち行列の先頭のタスクIDを求め、その結果をそれぞれ p_frbcnt, p_wtskid の指す領域に返します。p_frbcntの指す領域として 4バイト, p_wtskid の指す領域として 2 バイトのRAM領域が必要です。

なお、対象メモリプールの待ちタスクが無い場合は、待ちタスクIDとしてFALSE(H'0)を返します。

関連システムコール

get_blk,ipget_blk,irel_blk,mpl_sts,pget_blk,rel_blk

３．３．２５　ims_sts（Get Interrupt Mask Level Status）割込みマスクを参照する

［タスク部用］

Ｃ言語インタフェース

```
ER  ercd = ims_sts(SR *p_imask);
```

【パラメータ】

SR	*p_imask	割込みマスクを返す領域の先頭アドレス

【リターンパラメータ】

ER	ercd	エラーコード
SR	*p_imask	割込みマスクを格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffe0）
R4	p_imask	割込みマスクを返す領域の先頭アドレス

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_imask	割込みマスクを格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_imask が 4 の倍数以外または0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

解　　説

CPUのステータス・レジスタ（SR）の割込みマスクビット（Iビット）を参照し、割込みマスク値をp_imask の指す領域に返します。p_imask の指す領域として 4 バイトのRAM領域が必要です。

関連システムコール

chg_ims,ichg_ims,iims_sts,ret_int

３．３．２６ ipget_blk (Interrupt Poll and Get Memory Block)

固定長メモリブロックを獲得する［非タスク部用］

C言語インタフェース

```
ER  ercd = ipget_blk(VP *p_blk, ID mplid);
```

【パラメータ】

VP	*p_blk	メモリブロックの先頭アドレスを返す領域の先頭アドレス
ID	mplid	メモリプールID（1～hi_maxmplid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード
VP	*p_blk	メモリブロックの先頭アドレスを格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffe9）
R4	p_blk	メモリブロックの先頭アドレスを返す領域の先頭アドレス
R5	mplid	メモリプールID（1～hi_maxmplid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_blk	メモリブロックの先頭アドレスを格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_blk が 4 の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（mplid < 0, mplid > hi_maxmplid[*1]）
		[p]	予約ID（mplid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）
E_PLFAIL	H'fffff1a7(-H'e59)	[k]	ポーリング失敗

*1　hi_maxmplid　：　セットアップテーブルで定義した最大メモリプールID

解　説

mplidで示される固定長メモリプールから１つのメモリブロックを獲得できる場合には、その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p_blkの指す領域に返し正常終了します。指定したメモリプールからメモリブロックを獲得できない場合にはエラーコードとしてE_PLFAILを返します。

p_blk の指す領域として４バイトのRAM領域が必要です。

関連システムコール

get_blk,iget_blk,impl_sts,irel_blk,mpl_sts,pget_blk,rel_blk

３．３．２７ ipol_flg（Interrupt Poll Event Flag）イベントフラグを得る［非タスク部用］

C言語インタフェース

ER ercd = ipol_flg(UW *p_flgptn, ID flgid, UW waiptn, UW wfmode);

【パラメータ】

UW	*p_flgptn	待ち解除時のビットパターンを返す領域の先頭アドレス
ID	flgid	イベントフラグID（1～hi_maxflgid[*1]）
UW	waiptn	待ちビットパターン
UW	wfmode	待ちモード

【リターンパラメータ】

ER	ercd	エラーコード
UW	*p_flgptn	待ち解除時のビットパターンを格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffff3）
R4	p_flgptn	待ち解除時のビットパターンを返す領域の先頭アドレス
R5	flgid	イベントフラグID（1～hi_maxflgid[*1]）
R6	waiptn	待ちビットパターン
R7	wfmode	待ちモード

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_flgptn	待ち解除時のビットパターンを格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_PAR	H'fffff8df(-H'721)	[p]	パラメータエラー（waiptn = 0, wfmode が不正）
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_flgptn が 4 の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（flgid < 0, flgid > hi_maxflgid[*1]）
		[p]	予約ID（flgid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）
E_PLFAIL	H'fffff1a7(-H'e59)	[k]	ポーリング失敗

*1　hi_maxflgid　　：　セットアップテーブルで定義した最大イベントフラグID

解　　説

flgid で示されたイベントフラグ（３２ビット）が 、wfmode で示された条件にセットされていれば正常終了します。セットされていなければエラーコードとしてE_PLFAILを返します。

wfmode では、次のような指定を行なうことができます。

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

TWF_ANDW	H'00000000	AND待ち
TWF_ORW	H'00000002	OR待ち
TWF_CLR	H'00000001	クリア指定

TWF_ANDWでは、flgid で示されたイベントフラグのうち waiptn で指定したビットのすべてがセットされていれば正常終了します。TWF_ORWでは、flgid で示されたイベントフラグのうち waiptn で指定したビットのいずれかがセットされていれば正常終了します。

TWF_CLRの指定が無い場合には、条件が満足されてタスクが正常終了してもイベントフラグの値は変化しません。一方、TWF_CLRの指定がある場合には、条件が満足されてタスクが正常終了すると、イベントフラグの値（全部のビット）が0 にクリアされます。

正常終了するときのイベントフラグの値（TWF_CLR指定の場合は、イベントフラグがクリアされる前の値）を p_flgptn の指す領域に返します。p_flgptn 指す領域として、4 バイトのRAM領域が必要です。

関連システムコール

clr_flg,flg_sts,iclr_flg,iflg_sts,iset_flg,pol_flg,set_flg,wai_flg

３．３．２８ iprcv_msg (Interrupt Poll and Receive Message from Mailbox)

メイルボックスから受信する［非タスク部用］

C言語インタフェース

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

【パラメータ】

T_MSG	**ppk_msg	受信メッセージの先頭アドレスを返す領域の先頭アドレス
ID	mbxid	メイルボックスID（1～hi_maxmbxid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード
T_MSG	**ppk_msg	受信メッセージの先頭アドレスを格納した領域の先頭アドレス

【パケットの構造】

typedef	struct	t_msg {	
	UW	msghead;	ＯＳ管理領域
	VB	msgcont[n];	メッセージの内容
	} T_MSG;		

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffed）
R4	ppk_msg	受信メッセージの先頭アドレスを返す領域の先頭アドレス
R5	mbxid	メイルボックスID（1～hi_maxmbxid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード
R4	ppk_msg	受信メッセージの先頭アドレスを格納した領域の先頭アドレス

【パケットの構造】

	[****]	msghead	ＯＳ管理領域
	[*]	msgcont[n]	メッセージの内容

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（ppk_msg が 4 の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（mbxid < 0, mbxid > hi_maxmbxid[*1]）
		[p]	予約ID（mbxid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）
E_PLFAIL	H'fffff1a7(-H'e59)	[k]	ポーリング失敗

*1　hi_maxmbxid　：　セットアップテーブルで定義した最大メイルボックスID

解　　説

mbxid で示されたメイルボックスにメッセージがある場合には、そのメッセージを受信し、受信したメッセージの先頭アドレスを ppk_msg の指す領域に返し正常終了します。メイルボックスにメッセージがない場合にはエラーコードとしてE_PLFAILを返します。

ppk_msg の指す領域として 4 バイトのRAM領域が必要です。

ユーザがメッセージとして利用できる領域は、メッセージの先頭アドレス＋4バイト目からです。メッセージの先頭 4バイトはＯＳで使用するため、この部分は書き換えないでください。送信後、受信される前にこの領域を破壊すると、システムの正常な動作は保証されません。

関連システムコール

isnd_msg,imbx_sts,mbx_sts,prcv_msg,rcv_msg,snd_msg

３．３．２９ ipreq_sem（Interrupt Poll and Request Semaphore）セマフォ資源を得る

［非タスク部用］

C言語インタフェース

ER ercd = ipreq_sem(ID semid);

【パラメータ】

ID	semid	セマフォID（1～hi_maxsemid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffff0）
R4	semid	セマフォID（1～hi_maxsemid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（semid < 0, semid > hi_maxsemid[*1]）
		[p]	予約ID（semid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）
E_PLFAIL	H'fffff1a7(-H'e59)	[k]	ポーリング失敗

*1 hi_maxsemid ： セットアップテーブルで定義した最大セマフォID

解　　説

semid で示されたセマフォのカウント値が 1 以上の場合、セマフォのカウント値を 1 減らして正常終了します。セマフォのカウント値が 0 の場合は、セマフォのカウント値を変更せず、エラーコードとしてE_PLFAILを返します。

関連システムコール

isem_sts,isig_sem,preq_sem,sem_sts,sig_sem,wai_sem

３．３．３０ irel_blk（Interrupt Release Memory Block）固定長メモリブロックを返却する

［非タスク部用］

C言語インタフェース

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

【パラメータ】

ID	mplid	メモリプールID（1～hi_maxmplid[*1]）
VP	blk	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	エラーコード

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffe8）
R4	mplid	メモリプールID（1～hi_maxmplid[*1]）
R5	blk	メモリブロックの先頭アドレス

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（blk が 4 の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（mplid < 0, mplid > hi_maxmplid[*1]）
		[p]	予約ID（mplid = 0）
E_ILBLK	H'fffff7c5(-H'83b)	[p]	不正メモリブロックの返却、操作（blk がメモリブロックの先頭アドレス以外）
		[k]	不正メモリブロックの返却、操作（すでに返却した blk を指定）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

*1　hi_maxmplid　：　セットアップテーブルで定義した最大メモリプールID

解　説

mplid で示されたメモリプールへ blk で示されたメモリブロックを返却します。

blk には、get_blkまたはpget_blkシステムコールで獲得したメモリブロックの先頭アドレスと同じ値を指定してください。異なる blk を指定すると、エラーコードとしてE_ILBLKを返します。また、すでに返却した blk を指定した場合にも、エラーコードとしてE_ILBLKを返します。

関連システムコール

get_blk,impl_sts,ipget_blk,mpl_sts,pget_blk,rel_blk

３．３．３１　irel_wai（Interrupt　Release　Wait）タスクの待ち状態を強制解除する

［非タスク部用］

C言語インタフェース

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

【パラメータ】

ID	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffffc）
R4	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[p]	予約ID（tskid = 0）
		[k]	未登録（タスクが生成されていない）
E_NOWAI	H'fffff7c2(-H'83e)	[k]	タスクが待ち状態でない
		[k]	共有スタック待ちである
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

*1　hi_maxtskid　：　セットアップテーブルで定義した最大タスクID

解　説

tskid で示されるタスクが何らかの待ち（WAIT）状態（強制待ち（SUSPEND）状態は含まれません）の場合、それを強制的に解除します。

tskid で示されるタスクが待ち状態でない場合は、エラーコードとしてE_NOWAI を返します。

本システムコールにより待ち状態を解除したタスクに対しては、エラーコードとして E_RLWAI を返します。

本システムコールでは、強制待ち状態の解除は行ないません。強制待ち状態を解除する場合は、rsm_tsk（またはirsm_tsk）システムコールを発行してください。

二重待ち（WAIT-SUSPEND）状態のタスクに対して本システムコールを発行すると、対象タスクは強制待ち状態へ移行します。その後rsm_tsk（またはirsm_tsk）システムコールが発行され、強制待ち状態が解除されると、対象タスクにはエラーコードとして E_RLWAI が返されます。

なお、本システムコールで共有スタック待ち状態を解除することはできません。対象タスクが共有スタック待ち状態の場合は、エラーコードとしてE_NOWAI が返されます。

関連システムコール

get_blk,rcv_msg,rel_wai,slp_tsk,wai_flg,wai_sem,wai_tsk

３．３．３２ irot_rdq（Interrupt Rotate Ready Queue）タスクのレディーキューを回転する

［非タスク部用］

C言語インタフェース

ER ercd = irot_rdq(TPRI tskpri);

【パラメータ】

TPRI	tskpri	タスク優先度（1～hi_maxtskpri[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffffd）
R4	tskpri	タスク優先度（1～hi_maxtskpri[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_TPRI	H'fffff8da(-H'726)	[p]	不正タスク優先度 （tskpri < 0, tskpri > hi_maxtskpri[*1]）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

*1 hi_maxtskpri ： セットアップテーブルで定義した最大タスク優先度

解　説

tskpri で示されたタスク優先度のレディーキューを回転します。すなわち、そのタスク優先度のレディーキューの先頭につながれているタスクをレディーキューの最後尾につなぎかえ、同一タスク優先度のタスクに実行を切り換えます。

tskpriには、タスク優先度として 1～hi_maxtskpri（最大タスク優先度 ）の値を指定できます。

tskpri =TPRI_RUN(H'0)の指定により、そのとき実行（RUN）状態にあるタスクを含むレディーキュー（最高優先度のレディーキュー）を回転します。

現在、実行状態のタスクがない場合、または指定したタスク優先度のレディーキューにタスクがない場合は何も行ないませんが、エラーとはなりません。エラーコードとして E_OK を返します。

本システムコールを一定時間間隔で発行することにより、ラウンドロビンスケジューリングが行なえます。

関連システムコール

rot_rdq

３．３．３３　irsm_tsk（Interrupt　Resume　Task）強制待ち状態のタスクを再開する

［非タスク部用］

C言語インタフェース

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

【パラメータ】

ID	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffff8）
R4	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[p]	予約ID（tskid = 0）
		[k]	未登録（タスクが生成されていない）
E_NOSUS	H'fffff7c8(-H'838)	[k]	タスクが強制待ち（SUSPEND）状態でない
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

*1　hi_maxtskid　：　セットアップテーブルで定義した最大タスクID

解　説

tskid で示されたタスクがsus_tsk（またはisus_tsk）システムコールの発行によって中断されている場合、対象タスクの強制待ち（SUSPEND）状態を解除し、実行可能（READY）状態へ移行します。

対象タスクが二重待ち（WAIT-SUSPEND）状態の場合は、待ち（WAIT）状態へ移行します。

sus_tsk（またはisus_tsk）システムコールの要求のキューイングは行なわれません。したがって、本システムコールの発行により、必ず強制待ち状態が解除されます。

関連システムコール

isus_tsk,rsm_tsk,sus_tsk

３．３．３４　isem_sts（Interrupt　Get Semaphore　Status）セマフォの状態を参照する

［非タスク部用］

C言語インタフェース

ER　ercd = isem_sts(ID *p_wtskid, W *p_semcnt, ID semid);

【パラメータ】

ID	*p_wtskid	待ちタスクIDを返す領域の先頭アドレス
W	*p_semcnt	セマフォカウント値を返す領域の先頭アドレス
ID	semid	セマフォID（1～hi_maxsemid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード
ID	*p_wtskid	待ちタスクIDを格納した領域の先頭アドレス
W	*p_semcnt	セマフォカウント値を格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffef）
R4	p_wtskid	待ちタスクIDを返す領域の先頭アドレス
R5	p_semcnt	セマフォカウント値を返す領域の先頭アドレス
R6	semid	セマフォID（1～hi_maxsemid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_wtskid	待ちタスクIDを格納した領域の先頭アドレス
R5	p_semcnt	セマフォカウント値を格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_wtskidが奇数または0, p_semcnt が4の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（semid < 0, semid > hi_maxsemid[*1]）
		[p]	予約ID（semid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

*1　hi_maxsemid　：　セットアップテーブルで定義した最大セマフォID

解　　説

semid で示されたセマフォの各種状態を参照し、対象セマフォの現在のセマフォカウント値と待ち行列の先頭のタスクIDを求め、その結果をそれぞれ p_semcnt, p_wtskid の指す領域に返します。p_semcntの指す領域として4バイト , p_wtskidの指す領域として 2 バイトのRAM領域が必要です。

なお、対象セマフォの待ちタスクが無い場合は、待ちタスクIDとしてFALSE(H'0)を返します。

関連システムコール

ipreq_sem,isig_sem,preq_sem,sem_sts,sig_sem,wai_sem

３．３．３５ iset_flg（Interrupt Set Event Flag）イベントフラグをセットする

［非タスク部用］

C言語インタフェース

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

【パラメータ】

ID	flgid	イベントフラグID（1～hi_maxflgid[*1]）
UW	setptn	セットするビットパターン

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffff5）
R4	flgid	イベントフラグID（1～hi_maxflgid[*1]）
R5	setptn	セットするビットパターン

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（flgid < 0, flgid > hi_maxflgid[*1]）
		[p]	予約ID（flgid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

*1 hi_maxflgid ： セットアップテーブルで定義した最大イベントフラグID

解　説

flgid で示されたイベントフラグ（３２ビット）のうち、setptn で示されたビットをセットします。すなわち、flgid で示されたイベントフラグに対して setptn の値で論理和をとります。

本システムコールにおいて、イベントフラグ値の変更の結果、そのイベントフラグを待っていたタスクの待ち解除条件を満たすようになれば、そのタスクが実行可能（READY）状態へと移行します。

関連システムコール

clr_flg,flg_sts,iclr_flg,iflg_sts,ipol_flg,pol_flg,set_flg,wai_flg

３．３．３６　iset_tim（Interrupt　Set Time）システムクロックを設定する［非タスク部用］

C言語インタフェース

```
ER  ercd = iset_tim(T_TIM *pk_time);
```

【パラメータ】

T_TIM	*pk_time	現在の年月日時刻データを示すパケットの先頭アドレス

【リターンパラメータ】

ER	ercd	エラーコード

【パケットの構造】

```
typedef     struct      t_tim {
            H           utime;          現在の年月日時刻データ（上位）
            UW          ltime;          現在の年月日時刻データ（下位）
            } T_TIM;
```

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffe6）
R4	pk_time	現在の年月日時刻データを示すパケットの先頭アドレス

【リターンパラメータ】

R0	ercd	エラーコード

【パケットの構造】

```
            [**--]      utime           現在の年月日時刻データ（上位）
            [****]      ltime           現在の年月日時刻データ（下位）
```

エラーコード

E_OK	H'00000000	[k]	正常終了
E_TNOSPT	H'fffff9ed(-H'613)	[p]	時間管理機能が組み込まれていない
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（pk_time が４の倍数以外または0）
E_ILTIME	H'fffff8d9(-H'727)	[p]	不正時間指定（pk_time で示される値が負）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

解　　説

システムが保持しているシステムクロックの現在の値を、pk_time で示される値に設定します。

システムクロックは、48ビット（utime:16bit + ltime:32bit）で表現されます。

システムの動作中に本システムコールでシステムクロックを更新しても、タイムアウト監視中のタスクのタイムアウト時刻は更新しません。すなわち、タイムアウト監視中のタスクはwai_tskシステムコール発行後、指定した相対的な時間が経過するまで、タイムアウトにはなりません。

関連システムコール

get_tim,iget_tim,set_tim,sys_clk

３．３．３７ isig_sem（Interrupt Signal Semaphore）セマフォに対する信号操作（V命令）

［非タスク部用］

C言語インタフェース

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

【パラメータ】

ID	semid	セマフォID（1～hi_maxsemid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffff1）
R4	semid	セマフォID（1～hi_maxsemid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（semid < 0, semid > hi_maxsemid[*1]）
		[p]	予約ID（semid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）
E_QOVR	H'fffff4b7(-H'b49)	[k]	キューイングのオーバフロー（semcnt > H'ffff）

*1　hi_maxsemid ： セットアップテーブルで定義した最大セマフォID

解　　説

semid で示されたセマフォに対して待っているタスクがあれば、セマフォの待ち行列の先頭のタスクを実行可能（READY）状態へ移行します。セマフォに対して待っているタスクが無ければ、そのセマフォのカウント値（semcnt）を 1 増やします。

セマフォのカウント値の最大値は H'ffff(65,535) です。この値を越えた場合にはエラーコードとして E_QOVRを返します。

関連システムコール

ipreq_sem,isem_sts,preq_sem,sem_sts,sig_sem,wai_sem

３．３．３８ isnd_msg（Interrupt Send Message to Mailbox）メイルボックスへ送信する

［非タスク部用］

C言語インタフェース

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

【パラメータ】

ID	mbxid	メイルボックスID（1～hi_maxmbxid[*1]）
T_MSG	*pk_msg	送信メッセージの先頭アドレス

【リターンパラメータ】

ER	ercd	エラーコード

【パケットの構造】

typedef	struct	t_msg {	
	UW	msghead;	ＯＳ管理領域
	VB	msgcont[n];	メッセージの内容
	} T_MSG;		

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffee）
R4	mbxid	メイルボックスID（1～hi_maxmbxid[*1]）
R5	pk_msg	送信メッセージの先頭アドレス

【リターンパラメータ】

R0	ercd	エラーコード

【パケットの構造】

[****]	msghead	ＯＳ管理領域
[*]	msgcont[n]	メッセージの内容

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（pk_msg が 4 の倍数以外または0）
E_ILMSG	H'fffff8d7(-H'729)	[k]	不正メッセージ形式（メッセージ先頭 4 バイトが 0 でない）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（mbxid < 0, mbxid > hi_maxmbxid[*1]）
		[p]	予約ID（mbxid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

*1　hi_maxmbxid　：　セットアップテーブルで定義した最大メイルボックスID

解　　説

mbxid で示されたメイルボックスに pk_msg で示されたメッセージを送信します。メッセージの内容はコピーされず、アドレス（pk_msg の値）のみが渡されます。このため、本システムコールでメッセージ送信後、メッセージ内容を書き換えると、rcv_msgまたはprcv_msgシステムコールで正しい内容のメッセージを受け取ることができなくなるので注意してください。

対象メイルボックスに待ちタスクがある場合、待ち行列の先頭タスクにメッセージを渡し、そのタスクを実行可能（READY）状態へ移行します。待ちタスクがない場合、メッセージをメイルボックスに入れ、メッセージの行列につなぎます。メッセージの行列へのつなぎ方はFIFO(First-In First-Out)です。

メッセージは、必ずRAM領域に確保してください。メッセージの長さはＯＳで管理していませんので、ユーザ側で管理する必要があります。ユーザがメッセージとして利用できる領域は、メッセージの先頭アドレス＋4バイト目からです。メッセージの先頭 4バイトはＯＳで使用するため、初期値として必ず 0 を設定し、送信後も書き換えないでください。0 が設定されていない場合は、エラーコードとしてE_ILMSGを返します。

関連システムコール

imbx_sts,iprcv_msg,mbx_sts,prcv_msg,rcv_msg,snd_msg

３．３．３９　ista_tsk（Interrupt　Start Task）タスクを起動する［非タスク部用］

C言語インタフェース

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

【パラメータ】

ID	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffff）
R4	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[p]	予約ID（tskid = 0）
		[k]	未登録（タスクが生成されていない）
E_NODMT	H'fffff7ca(-H'836)	[k]	タスクが休止（DORMANT）状態でない
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

*1　hi_maxtskid　：　セットアップテーブルで定義した最大タスクID

解　説

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）状態へ移行します。

起動時のタスク優先度は、cre_tskシステムコール発行時（またはセットアップテーブルにおける初期登録タスクの定義）に指定した初期タスク優先度となります。

本システムコールによる起動要求のキューイングは行ないません。対象タスクが休止状態にない場合に発行された起動要求に対しては、エラーコードとしてE_NODMTを返します。

HI-SH77では、複数のタスクでスタック領域を共有する機能（共有スタック機能）を持っています。セットアップテーブルで初期タスクスタックポインタに同じ値を登録したタスクID同士は、それらのタスク間でひとつのスタックを共有する（共有スタック）ことになります。

本システムコール発行時に、tskidで示されたタスクのスタック（共有スタック）が開放（どのタスクも使用していない）されている場合は、tskidで示されたタスクが共有スタックを占有し、実行可能状態に移行します。

共有スタックが占有されている（他のタスクが使用中）場合は、スタック領域を使用できないため、タスクは待ち（WAIT）状態になり、共有スタックの待ち行列につながれます。待ち行列のつながれ方は、FIFO(First-In First-Out)です。共有スタック待ち状態のタスクに対して本システムコールを発行すると、エラーコードとしてE_OKを返します。

本システムコールは、HI-SH77の独自仕様です。

関連システムコール

cre_tsk,sta_tsk,ter_tsk

３．３．４０ isus_tsk（Interrupt Suspend Task）タスクを強制待ち状態へ移行する

［非タスク部用］

C言語インタフェース

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

【パラメータ】

ID	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffff9）
R4	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[p]	予約ID（tskid = 0）
		[k]	未登録（タスクが生成されていない）
E_DMT	H'fffff7cb(-H'835)	[k]	タスクが休止（DORMANT）状態である
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）
E_QOVR	H'fffff4b7(-H'b49)	[k]	キューイングのオーバフロー（既にタスクが強制待ち（SUSPEND）状態である）

*1　hi_maxtskid　：　セットアップテーブルで定義した最大タスクID

解　説

tskid で示されたタスクの実行を中断させ、強制待ち（SUSPEND）状態へ移行します。

強制待ち状態は、rsm_tsk（またはirsm_tsk）システムコールの発行により解除されます。

tskid で示されたタスクが待ち（WAIT）状態にある場合は、二重待ち（WAIT-SUSPEND）状態へ移行します。

本システムコールの要求のキューイングは行ないません。強制待ち状態のときに本システムコールを発行した場合、エラーコードとしてE_QOVRを返します。

関連システムコール

irsm_tsk,rsm_tsk,sus_tsk

３．３．４１　itsk_sts（Interrupt　Get Task Status）タスク状態を見る［非タスク部用］

C言語インタフェース

ER　ercd = itsk_sts(UH *p_tskstat, TPRI *p_tskpri, ID tskid);

【パラメータ】

UH	*p_tskstat	タスク状態を返す領域の先頭アドレス
TPRI	*p_tskpri	現在タスク優先度を返す領域の先頭アドレス
ID	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード
UH	*p_tskstat	タスク状態を格納した領域の先頭アドレス
TPRI	*p_tskpri	現在タスク優先度を格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffffa）
R4	p_tskstat	タスク状態を返す領域の先頭アドレス
R5	p_tskpri	現在タスク優先度を返す領域の先頭アドレス
R6	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_tskstat	タスク状態を格納した領域の先頭アドレス
R5	p_tskpri	現在タスク優先度を格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_tskstatが奇数または0，p_tskpriが奇数または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[p]	予約ID（tskid = 0）
		[k]	未登録（タスクが生成されていない）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）

*1　hi_maxtskid：　セットアップテーブルで定義した最大タスクID

解　　説

tskid で示されたタスクの状態を読み出し、現在のタスク状態、タスク優先度をそれぞれp_tskstat, p_tskpriの指す領域に返します。

p_tskstat, p_tskpri の指す領域として、それぞれ2バイトのRAM領域が必要です。

p_tskstatの指す領域には、次の値を返します。

p_tskstat:

wupcnt	H'0000 - H'000F	wup_tsk,iwup_tskシステムコールのキューイング数
TTS_RDY	H'0010	実行状態／実行可能状態
TTS_WAI	H'0020	待ち状態
TTS_SUS	H'0040	強制待ち状態
TTS_WAS	H'0060	二重待ち状態
TTS_DMT	H'0080	休止状態
TTS_STK	H'4000	共有スタック解放待ち状態
TTW_SLP	H'0100	slp_tskによる待ち
TTW_WAI	H'0200	wai_tskによる待ち
TTW_FLG	H'0400	wai_flgによる待ち
TTW_SEM	H'0800	wai_semによる待ち
TTW_MBX	H'1000	rcv_msgによる待ち
TTW_MPL	H'2000	get_blkによる待ち

p_tskpri の指す領域には、現在のタスク優先度を返します。

関連システムコール

tsk_sts

３．３．４２　iwup_tsk（Interrupt Wakeup Task）待ち状態のタスクを起床する［非タスク部用］

C言語インタフェース

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

【パラメータ】

ID	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#62	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffff7）
R4	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[p]	予約ID（tskid = 0）
		[k]	未登録（タスクが生成されていない）
E_DMT	H'fffff7cb(-H'835)	[k]	タスクが休止（DORMANT）状態である
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（タスク部から発行できない）
E_QOVR	H'fffff4b7(-H'b49)	[k]	キューイングのオーバフロー（wupcnt > H'0f）

*1　hi_maxtskid：　セットアップテーブルで定義した最大タスクID

解　説

slp_tskシステムコールまたはwai_tskシステムコールの発行により待ち（WAIT）状態になっているタスクを、実行可能（READY）状態へ移行します。

対象タスクがslp_tskシステムコール、またはwai_tskシステムコールを発行しておらず、待ち状態でない場合には、本システムコールの起床要求はキューイングされ、後に対象タスクがslp_tskシステムコール、またはwai_tskシステムコールを発行したときに有効になります。

起床要求（wupcnt）のキューイング数の最大値は H'f(15) です。この値を越えた場合にはエラーコードとしてE_QOVRを返します。

関連システムコール

can_wup,ican_wup,slp_tsk,wai_tsk,wup_tsk

３．３．４３　mbx_sts（Get Mailbox Status）メイルボックスの状態を参照する

［タスク部用］

C言語インタフェース

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

【パラメータ】

ID	*p_wtskid	待ちタスクIDを返す領域の先頭アドレス
T_MSG	**ppk_msg	次のメッセージの先頭アドレスを返す領域の先頭アドレス
ID	mbxid	メイルボックスID（1～hi_maxmbxid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード
ID	*p_wtskid	待ちタスクIDを格納した領域の先頭アドレス
T_MSG	**ppk_msg	次のメッセージの先頭アドレスを格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffee2）
R4	p_wtskid	待ちタスクIDを返す領域の先頭アドレス
R5	ppk_msg	次のメッセージの先頭アドレスを返す領域の先頭アドレス
R6	mbxid	メイルボックスID（1～hi_maxmbxid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_wtskid	待ちタスクIDを格納した領域の先頭アドレス
R5	ppk_msg	次のメッセージの先頭アドレスを格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_wtskid が奇数または0, ppk_msg が 4 の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（mbxid < 0, mbxid > hi_maxmbxid[*1]）
		[p]	予約ID（mbxid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxmbxid　：　セットアップテーブルで定義した最大メイルボックスID

解　　説

mbxid で示されたメイルボックスの各種状態を参照し、対象メイルボックスの次に受信されるメッセージの先頭アドレスと待ち行列の先頭のタスクIDを求め、その結果をそれぞれ ppk_msg, p_wtskid の指す領域に返します。ppk_msg の指す領域として 4 バイト、 p_wtskidの指す領域として 2 バイトのRAM領域が必要です。

なお、次に受信されるメッセージが無い場合は、メッセージの先頭アドレスとしてNADR(H'ffffffff)を返します。

また、対象メイルボックスの待ちタスクが無い場合は、待ちタスクIDとしてFALSE(H'0)を返します。

関連システムコール

isnd_msg,imbx_sts,iprcv_msg,prcv_msg,rcv_msg,snd_msg

３．３．４４　mpl_sts（Get Memory Pool Status）メモリプールの状態を参照する

［タスク部用］

C言語インタフェース

```
ER  ercd = mpl_sts(ID *p_wtskid, W *p_frbcnt, ID mplid);
```

【パラメータ】

ID	*p_wtskid	待ちタスクIDを返す領域の先頭アドレス
W	*p_frbcnt	空き領域のブロック数を返す領域の先頭アドレス
ID	mplid	メモリプールID（1～hi_maxmplid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード
ID	*p_wtskid	待ちタスクIDを格納した領域の先頭アドレス
W	*p_frbcnt	空き領域のブロック数を格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffedc）
R4	p_wtskid	待ちタスクIDを返す領域の先頭アドレス
R5	p_frbcnt	空き領域のブロック数を返す領域の先頭アドレス
R6	mplid	メモリプールID（1～hi_maxmplid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_wtskid	待ちタスクIDを格納した領域の先頭アドレス
R5	p_frbcnt	空き領域のブロック数を格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_wtskidが奇数または0, p_frbcnt が4の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（mplid < 0, mplid > hi_maxmplid[*1]）
		[p]	予約ID（mplid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxmplid　：　セットアップテーブルで定義した最大メモリプールID

解　　説

mplid で示されたメモリプールの各種状態を参照し、対象メモリプールの現在の空きブロック数と待ち行列の先頭のタスクIDを求め、その結果をそれぞれ p_frbcnt, p_wtskid の指す領域に返します。p_frbcntの指す領域として 4バイト, p_wtskid の指す領域として 2 バイトのRAM領域が必要です。

なお、対象メモリプールの待ちタスクが無い場合は、待ちタスクIDとしてFALSE(H'0)を返します。

関連システムコール

get_blk,impl_sts,ipget_blk,irel_blk,pget_blk,rel_blk

３．３．４５　pget_blk（Poll and Get Memory Block）固定長メモリブロックを獲得する

［タスク部用］

C言語インタフェース

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

【パラメータ】

VP	*p_blk	メモリブロックの先頭アドレスを返す領域の先頭アドレス
ID	mplid	メモリプールID（1～hi_maxmplid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード
VP	*p_blk	メモリブロックの先頭アドレスを格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffede）
R4	p_blk	メモリブロックの先頭アドレスを返す領域の先頭アドレス
R5	mplid	メモリプールID（1～hi_maxmplid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_blk	メモリブロックの先頭アドレスを格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_blk が 4 の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（mplid < 0, mplid > hi_maxmplid[*1]）
		[p]	予約ID（mplid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）
E_PLFAIL	H'fffff1a7(-H'e59)	[k]	ポーリング失敗

*1　hi_maxmplid　：　セットアップテーブルで定義した最大メモリプールID

解　説

mplidで示される固定長メモリプールから１つのメモリブロックを獲得できる場合には、その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p_blkの指す領域に返し正常終了します。指定したメモリプールからメモリブロックを獲得できない場合にはエラーコードとしてE_PLFAILを返します。

p_blk の指す領域として４バイトのRAM領域が必要です。

関連システムコール

get_blk,impl_sts,ipget_blk,irel_blk,mpl_sts,rel_blk

３．３．４６　pol_flg（Poll Event Flag）イベントフラグを得る［タスク部用］

C言語インタフェース

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

【パラメータ】

UW	*p_flgptn	待ち解除時のビットパターンを返す領域の先頭アドレス
ID	flgid	イベントフラグID（1～hi_maxflgid[*1]）
UW	waiptn	待ちビットパターン
UW	wfmode	待ちモード

【リターンパラメータ】

ER	ercd	エラーコード
UW	*p_flgptn	待ち解除時のビットパターンを格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffeb）
R4	p_flgptn	待ち解除時のビットパターンを返す領域の先頭アドレス
R5	flgid	イベントフラグID（1～hi_maxflgid[*1]）
R6	waiptn	待ちビットパターン
R7	wfmode	待ちモード

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_flgptn	待ち解除時のビットパターンを格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_PAR	H'fffff8df(-H'721)	[p]	パラメータエラー（waiptn = 0, wfmode が不正）
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_flgptn が 4 の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（flgid < 0, flgid > hi_maxflgid[*1]）
		[p]	予約ID（flgid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）
E_PLFAIL	H'fffff1a7(-H'e59)	[k]	ポーリング失敗

*1　hi_maxflgid　：　セットアップテーブルで定義した最大イベントフラグID

解　説

flgid で示されたイベントフラグ（３２ビット）が 、wfmode で示された条件にセットされていれば正常終了します。セットされていなければエラーコードとしてE_PLFAILを返します。

wfmode では、次のような指定を行なうことができます。

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

TWF_ANDW	H'00000000	AND待ち
TWF_ORW	H'00000002	OR待ち
TWF_CLR	H'00000001	クリア指定

TWF_ANDWでは、flgid で示されたイベントフラグのうち waiptn で指定したビットのすべてがセットされていれば正常終了します。TWF_ORWでは、flgid で示されたイベントフラグのうち waiptn で指定したビットのいずれかがセットされていれば正常終了します。

TWF_CLRの指定が無い場合には、条件が満足されてタスクが正常終了してもイベントフラグの値は変化しません。一方、TWF_CLRの指定がある場合には、条件が満足されてタスクが正常終了すると、イベントフラグの値（全部のビット）が0 にクリアされます。

正常終了するときのイベントフラグの値（TWF_CLR指定の場合は、イベントフラグがクリアされる前の値）を p_flgptn の指す領域に返します。p_flgptn 指す領域として、4 バイトのRAM領域が必要です。

関連システムコール

clr_flg,flg_sts,iclr_flg,iflg_sts,ipol_flg,iset_flg,set_flg,wai_flg

３．３．４７　prcv_msg（Poll and Receive Message from Mailbox）
メイルボックスから受信する［タスク部用］

Ｃ言語インタフェース

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

【パラメータ】

T_MSG	**ppk_msg	受信メッセージの先頭アドレスを返す領域の先頭アドレス
ID	mbxid	メイルボックスID（1～hi_maxmbxid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード
T_MSG	**ppk_msg	受信メッセージの先頭アドレスを格納した領域の先頭アドレス

【パケットの構造】

```
typedef     struct      t_msg {
            UW          msghead;        ＯＳ管理領域
            VB          msgcont[n];     メッセージの内容
            } T_MSG;
```

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffe3）
R4	ppk_msg	受信メッセージの先頭アドレスを返す領域の先頭アドレス
R5	mbxid	メイルボックスID（1～hi_maxmbxid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード
R4	ppk_msg	受信メッセージの先頭アドレスを格納した領域の先頭アドレス

【パケットの構造】

```
            [****]      msghead         ＯＳ管理領域
            [*]         msgcont[n]      メッセージの内容
```

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（ppk_msg が 4 の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（mbxid < 0, mbxid > hi_maxmbxid[*1]）
		[p]	予約ID（mbxid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）
E_PLFAIL	H'fffff1a7(-H'e59)	[k]	ポーリング失敗

*1　hi_maxmbxid　：　セットアップテーブルで定義した最大メイルボックスID

解　　説

mbxid で示されたメイルボックスにメッセージがある場合には、そのメッセージを受信し、受信したメッセージの先頭アドレスを ppk_msg の指す領域に返し正常終了します。メイルボックスにメッセージがない場合にはエラーコードとしてE_PLFAILを返します。

ppk_msg の指す領域として 4 バイトのRAM領域が必要です。

ユーザがメッセージとして利用できる領域は、メッセージの先頭アドレス＋4バイト目からです。メッセージの先頭 4バイトはＯＳで使用するため、この部分は書き換えないでください。送信後、受信される前にこの領域を破壊すると、システムの正常な動作は保証されません。

関連システムコール

isnd_msg,imbx_sts,iprcv_msg,mbx_sts,rcv_msg,snd_msg

３．３．４８　preq_sem（Poll and Request Semaphore）セマフォ資源を得る

［タスク部用］

C言語インタフェース

ER ercd = preq_sem(ID semid);

【パラメータ】

ID	semid	セマフォID（1～hi_maxsemid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffe7）
R4	semid	セマフォID（1～hi_maxsemid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（semid < 0, semid > hi_maxsemid[*1]）
		[p]	予約ID（semid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）
E_PLFAIL	H'fffff1a7(-H'e59)	[k]	ポーリング失敗

*1　hi_maxsemid　：　セットアップテーブルで定義した最大セマフォID

解　　説

semid で示されたセマフォのカウント値が 1 以上の場合、セマフォのカウント値を 1 減らして正常終了します。セマフォのカウント値が 0 の場合は、セマフォのカウント値を変更せず、エラーコードとして E_PLFAILを返します。

関連システムコール

ipreq_sem,isem_sts,isig_sem,sem_sts,sig_sem,wai_sem

３．３．４９ rcv_msg（Receive Message from Mailbox）

メイルボックスからの受信を待つ［タスク部用］

Ｃ言語インタフェース

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

【パラメータ】

T_MSG	**ppk_msg	受信メッセージの先頭アドレスを返す領域の先頭アドレス
ID	mbxid	メイルボックスID（1～hi_maxmbxid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード
T_MSG	**ppk_msg	受信メッセージの先頭アドレスを格納した領域の先頭アドレス

【パケットの構造】

typedef	struct	t_msg {	
	UW	msghead;	ＯＳ管理領域
	VB	msgcont[n];	メッセージの内容
	} T_MSG;		

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffe4）
R4	ppk_msg	受信メッセージの先頭アドレスを返す領域の先頭アドレス
R5	mbxid	メイルボックスID（1～hi_maxmbxid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード
R4	ppk_msg	受信メッセージの先頭アドレスを格納した領域の先頭アドレス

【パケットの構造】

	[****]	msghead	ＯＳ管理領域
	[*]	msgcont[n]	メッセージの内容

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（ppk_msg が 4 の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（mbxid < 0, mbxid > hi_maxmbxid[*1]）
		[p]	予約ID（mbxid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）
E_RLWAI	H'fffff2aa(-H'd56)	[k]	待ち（WAIT）状態強制解除（待ちの間にrel_waiまたはirel_waiシステムコールが発行された）

*1　hi_maxmbxid　：　セットアップテーブルで定義した最大メイルボックスID

解　　説

mbxid で示されたメイルボックスからメッセージを受信し、受信したメッセージの先頭アドレスを ppk_msg の指す領域に返します。まだそのメイルボックスにメッセージが到着していない場合には、発行タスクはメッセージ到着を待つ待ち行列につながれます。待ち行列のつながれ方はFIFO(First-In First-Out)です。

ppk_msg の指す領域として 4 バイトのRAM領域が必要です。

ユーザがメッセージとして利用できる領域は、メッセージの先頭アドレス＋4バイト目からです。メッセージの先頭 4バイトはＯＳで使用するため、この部分は書き換えないでください。送信後、受信される前にこの領域を破壊すると、システムの正常な動作は保証されません。

関連システムコール

imbx_sts,iprcv_msg,irel_wai,isnd_msg,mbx_sts,prcv_msg,rel_wai,snd_msg

３．３．５０ rel_blk（Release Memory Block）固定長メモリブロックを返却する

［タスク部用］

C言語インタフェース

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

【パラメータ】

ID	mplid	メモリプールID（1～hi_maxmplid[*1]）
VP	blk	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	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffedd）
R4	mplid	メモリプールID（1～hi_maxmplid[*1]）
R5	blk	メモリブロックの先頭アドレス

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（blk が 4 の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（mplid < 0, mplid > hi_maxmplid[*1]）
		[p]	予約ID（mplid = 0）
E_ILBLK	H'fffff7c5(-H'83b)	[p]	不正メモリブロックの返却、操作（blk がメモリブロックの先頭アドレス以外）
		[k]	不正メモリブロックの返却、操作（すでに返却した blk を指定）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxmplid　：　セットアップテーブルで定義した最大メモリプールID

解　説

mplid で示されたメモリプールへ blk で示されたメモリブロックを返却します。

blk には、get_blkまたはpget_blkシステムコールで獲得したメモリブロックの先頭アドレスと同じ値を指定してください。異なる blk を指定すると、エラーコードとしてE_ILBLKを返します。また、すでに返却した blk を指定した場合にも、エラーコードとしてE_ILBLKを返します。

関連システムコール

get_blk,impl_sts,ipget_blk,irel_blk,mpl_sts,pget_blk

３．３．５１　rel_wai（Release Wait）タスクの待ち状態を強制解除する［タスク部用］

C言語インタフェース

ER ercd = rel_wai(ID tskid);

【パラメータ】

ID	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffef7）
R4	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[p]	予約ID（tskid = 0）
		[k]	未登録（タスクが生成されていない）
E_NOWAI	H'fffff7c2(-H'83e)	[k]	自タスク指定（tskid が自タスクである）
		[k]	タスクが待ち状態でない
		[k]	共有スタック待ちである
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxtskid　：　セットアップテーブルで定義した最大タスクID

解　説

tskid で示されるタスクが何らかの待ち（WAIT）状態（強制待ち（SUSPEND）状態は含まれません）の場合、それを強制的に解除します。

tskid は、待ち状態を強制解除する他タスクのIDを指定します。tskid に自タスクのIDを指定した場合、または tskid で示されるタスクが待ち状態でない場合は、発行タスクにエラーコードとしてE_NOWAI を返します。

本システムコールにより待ち状態を解除したタスクに対しては、エラーコードとして E_RLWAI を返します。

本システムコールでは、強制待ち状態の解除は行ないません。強制待ち状態を解除する場合は、rsm_tsk（またはirsm_tsk）システムコールを発行してください。

二重待ち（WAIT-SUSPEND）状態のタスクに対して本システムコールを発行すると、対象タスクは強制待ち状態へ移行します。その後rsm_tsk（またはirsm_tsk）システムコールが発行され、強制待ち状態が解除されると、対象タスクにはエラーコードとして E_RLWAI が返されます。

なお、本システムコールで共有スタック待ち状態を解除することはできません。対象タスクが共有スタック待ち状態の場合は、エラーコードとしてE_NOWAI が返されます。

関連システムコール

get_blk,irel_wai,rcv_msg,slp_tsk,wai_flg,wai_sem,wai_tsk

３．３．５２　ret_exc（Return fromException）例外処理から復帰する

［タスク部／非タスク部用］

Ｃ言語インタフェース

なし（"#pragma interrupt"文の定義により、例外処理からの復帰で発行される）

アセンブラインタフェース

TRAPA　　#57　　システムコール用トラップ

【パラメータ】

なし

【リターンパラメータ】

本システムコールの発行元には戻らない

エラーコード

なし

解　　説

例外処理ルーチンからの復帰の際に発行するシステムコールです。

PCとSRをスタックから復帰し、復帰したPCの示すアドレスから処理を続行します。

例外処理ルーチンをC言語で記述する場合は、例外処理ルーチンとなる関数を以下のようにSHシリーズCコンパイラの"#pragma interrupt"文（プリプロセッサ制御文）を用いて宣言してください。この宣言により、例外処理ルーチン終了時のret_excシステムコール発行処理の機械語が自動的に生成されます。

#pragma interrupt(関数名(tn=57))

tn=TRAPA命令リターン指定　　ret_excシステムコールにより、例外発行元に復帰します。

３．３．５３　ret_int（Return from Interrupt Handler）割込みハンドラから復帰する

［非タスク部用］

Ｃ言語インタフェース

なし（"#pragma interrupt"文の定義により、割込みハンドラからの復帰で発行される）

アセンブラインタフェース

TRAPA　　#61　　システムコール用トラップ

【パラメータ】

なし

【リターンパラメータ】

本システムコールの発行元には戻らない

エラーコード

E_CTX　　H'fffff5bb(-H'a45)　　[k]　　コンテキストエラー*1（タスク部から発行できない）

*1　エラー検出時は、R4に-3（異常終了種別）、R5にエラーコード、R6にret_intシステムコール発行アドレス+2を設定し、システム異常終了処理（hi_sysdwn関数）に制御を渡します。

解　説

割込みハンドラからの復帰の際に発行するシステムコールです。

割込みハンドラの中でタスク切り換えが必要となっても、それは本システムコールを発行し割込みハンドラから復帰するまで遅延されます。

割込みハンドラには、割込みが発生するとカーネルの例外サービスルーチンを介して制御が渡されます。

割込みハンドラは割込み発生時のレジスタ内容を保存し、保障しなければなりません。

割込みハンドラをアセンブリ言語で記述する場合は、使用するレジスタの退避や復帰をユーザ側で必ず行なってください。

割込みハンドラをＣ言語で記述する場合は、割込みハンドラとなる関数を以下のようにSHシリーズCコンパイラの"#pragma interrupt"文（プリプロセッサ制御文）を用いて宣言してください。この宣言により、レジスタの退避や復帰と、割込みハンドラ終了時のret_intシステムコール発行処理の機械語が自動的に生成されます。

#pragma interrupt (関数名(sp=&stack, tn = 61))

sp=	スタック切り換え指定	割込みハンドラが使用するスタック領域の初期スタックポインタ
tn=	TRAPA命令リターン指定	ret_intシステムコールにより割込み発行元に復帰します。

関連システムコール

chg_ims,ichg_ims,iims_sts,ims_sts

３．３．５４　rot_rdq（Rotate Ready Queue）タスクのレディーキューを回転する

［タスク部用］

C言語インタフェース

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

【パラメータ】

TPRI	tskpri	タスク優先度（1～hi_maxtskpri[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffef8）
R4	tskpri	タスク優先度（1～hi_maxtskpri[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_TPRI	H'fffff8da(-H'726)	[p]	不正タスク優先度 （tskpri < 0, tskpri > hi_maxtskpri[*1]）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxtskpri　：　セットアップテーブルで定義した最大タスク優先度

解　説

tskpri で示されたタスク優先度のレディーキューを回転します。すなわち、そのタスク優先度のレディーキューの先頭につながれているタスクをレディーキューの最後尾につなぎかえ、同一タスク優先度のタスクに実行を切り換えます。

tskpriには、タスク優先度として 1～hi_maxtskpri（最大タスク優先度 ）の値を指定できます。

tskpri =TPRI_RUN(H'0)の指定により、そのとき実行（RUN）状態にあるタスクを含むレディーキュー（最高優先度のレディーキュー）を回転します。

tskpri = TPRI_RUN(H'0)の指定、または自タスクのタスク優先度を指定した場合には、自タスクをそのレディーキューの最後尾にまわします。つまり、自ら実行権を放棄するために、本システムコールを発行することができます。

現在、実行状態のタスクがない場合、または指定したタスク優先度のレディーキューにタスクがない場合は何も行ないませんが、エラーとはなりません。エラーコードとして E_OK を返します。

関連システムコール

irot_rdq

３．３．５５　rsm_tsk（Resume Task）強制待ち状態のタスクを再開する［タスク部用］

C言語インタフェース

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

【パラメータ】

ID	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffef3）
R4	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[p]	予約ID（tskid = 0）
		[k]	未登録（タスクが生成されていない）
E_NOSUS	H'fffff7c8(-H'838)	[k]	自タスク指定（tskid が自タスクである）
		[k]	タスクが強制待ち（SUSPEND）状態でない
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxtskid　：　セットアップテーブルで定義した最大タスクID

解　説

tskid で示されたタスクがsus_tsk（またはisus_tsk）システムコールの発行によって中断されている場合、対象タスクの強制待ち（SUSPEND）状態を解除し、実行可能（READY）状態へ移行します。

対象タスクが二重待ち（WAIT-SUSPEND）状態の場合は、待ち（WAIT）状態へ移行します。

本システムコールでは、自タスクを指定することはできません。

sus_tsk（またはisus_tsk）システムコールの要求のキューイングは行なわれません。したがって、本システムコールの発行により、必ず強制待ち状態が解除されます。

関連システムコール

irsm_tsk,isus_tsk,sus_tsk

３．３．５６　sem_sts（Get Semaphore　Status）セマフォの状態を参照する

［タスク部用］

C言語インタフェース

ER　ercd = sem_sts(ID *p_wtskid, W *p_semcnt, ID semid);

【パラメータ】

ID	*p_wtskid	待ちタスクIDを返す領域の先頭アドレス
W	*p_semcnt	セマフォカウント値を返す領域の先頭アドレス
ID	semid	セマフォID（1～hi_maxsemid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード
ID	*p_wtskid	待ちタスクIDを格納した領域の先頭アドレス
W	*p_semcnt	セマフォカウント値を格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffee6）
R4	p_wtskid	待ちタスクIDを返す領域の先頭アドレス
R5	p_semcnt	セマフォカウント値を返す領域の先頭アドレス
R6	semid	セマフォID（1～hi_maxsemid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_wtskid	待ちタスクIDを格納した領域の先頭アドレス
R5	p_semcnt	セマフォカウント値を格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_wtskidが奇数または0, p_semcnt が4の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（semid < 0, semid > hi_maxsemid[*1]）
		[p]	予約ID（semid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxsemid　：　セットアップテーブルで定義した最大セマフォID

解　　説

semid で示されたセマフォの各種状態を参照し、対象セマフォの現在のセマフォカウント値と待ち行列の先頭のタスクIDを求め、その結果をそれぞれ p_semcnt, p_wtskid の指す領域に返します。p_semcntの指す領域として4バイト , p_wtskidの指す領域として 2 バイトのRAM領域が必要です。

なお、対象セマフォの待ちタスクが無い場合は、待ちタスクIDとしてFALSE(H'0)を返します。

関連システムコール

ipreq_sem,isem_sts,isig_sem,preq_sem,sig_sem,wai_sem

３．３．５７　set_flg（Set Event Flag）イベントフラグをセットする［タスク部用］

C言語インタフェース

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

【パラメータ】

ID	flgid	イベントフラグID（1～hi_maxflgid[*1]）
UW	setptn	セットするビットパターン

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffeee）
R4	flgid	イベントフラグID（1～hi_maxflgid[*1]）
R5	setptn	セットするビットパターン

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（flgid < 0, flgid > hi_maxflgid[*1]）
		[p]	予約ID（flgid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxflgid　：　セットアップテーブルで定義した最大イベントフラグID

解　　説

flgid で示されたイベントフラグ（３２ビット）のうち、setptn で示されたビットをセットします。すなわち、flgid で示されたイベントフラグに対して setptn の値で論理和をとります。

本システムコールにおいて、イベントフラグ値の変更の結果、そのイベントフラグを待っていたタスクの待ち解除条件を満たすようになれば、そのタスクが実行可能（READY）状態へと移行します。

関連システムコール

clr_flg,flg_sts,iclr_flg,iflg_sts,ipol_flg,iset_flg,pol_flg,wai_flg

３．３．５８　set_tim（Set Time）システムクロックを設定する［タスク部用］

C言語インタフェース

```
ER  ercd = set_tim(T_TIM *pk_time);
```

【パラメータ】

T_TIM	*pk_time	現在の年月日時刻データを示すパケットの先頭アドレス

【リターンパラメータ】

ER	ercd	エラーコード

【パケットの構造】

```
typedef     struct      t_tim {
            H           utime;          現在の年月日時刻データ（上位）
            UW          ltime;          現在の年月日時刻データ（下位）
            } T_TIM;
```

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffedb）
R4	pk_time	現在の年月日時刻データを示すパケットの先頭アドレス

【リターンパラメータ】

R0	ercd	エラーコード

【パケットの構造】

```
            [**--]      utime           現在の年月日時刻データ（上位）
            [****]      ltime           現在の年月日時刻データ（下位）
```

エラーコード

E_OK	H'00000000	[k]	正常終了
E_TNOSPT	H'fffff9ed(-H'613)	[p]	時間管理機能が組み込まれていない
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（pk_time が４の倍数以外または0）
E_ILTIME	H'fffff8d9(-H'727)	[p]	不正時間指定（pk_time で示される値が負）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

解　　説

システムが保持しているシステムクロックの現在の値を、pk_time で示される値に設定します。

システムクロックは、48ビット（utime:16bit + ltime:32bit）で表現されます。

システムの動作中に本システムコールでシステムクロックを更新しても、タイムアウト監視中のタスクのタイムアウト時刻は更新しません。すなわち、タイムアウト監視中のタスクはwai_tskシステムコール発行後、指定した相対的な時間が経過するまで、タイムアウトにはなりません。

関連システムコール

get_tim,iget_tim,iset_tim,sys_clk

３．３．５９　sig_sem（Signal Semaphore）セマフォに対する信号操作（V命令）

［タスク部用］

C言語インタフェース

ER ercd = sig_sem(ID semid);

【パラメータ】

ID	semid	セマフォID（1～hi_maxsemid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffe9）
R4	semid	セマフォID（1～hi_maxsemid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（semid < 0, semid > hi_maxsemid[*1]）
		[p]	予約ID（semid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）
E_QOVR	H'fffff4b7(-H'b49)	[k]	キューイングのオーバフロー（semcnt > H'ffff）

*1　hi_maxsemid ：　セットアップテーブルで定義した最大セマフォID

解　説

semid で示されたセマフォに対して待っているタスクがあれば、セマフォの待ち行列の先頭のタスクを実行可能（READY）状態へ移行します。セマフォに対して待っているタスクが無ければ、そのセマフォのカウント値（semcnt）を 1 増やします。

セマフォのカウント値の最大値は H'ffff(65,535) です。この値を越えた場合にはエラーコードとして E_QOVRを返します。

関連システムコール

ipreq_sem,isem_sts,isig_sem,preq_sem,sem_sts,wai_sem

３．３．６０slp_tsk（Sleep　Task）自タスクを待ち状態へ移行する［タスク部用］

C言語インタフェース

ER ercd = slp_tsk(void);

【パラメータ】

なし

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffef2）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）
E_RLWAI	H'fffff2aa(-H'd56)	[k]	待ち（WAIT）状態強制解除（待ちの間にrel_waiまたはirel_waiシステムコールが発行された）

解　　説

自タスクを実行（RUN）状態から待ち（WAIT）状態へ移行します。

この待ち状態は、本タスクを対象としてwup_tsk（またはiwup_tsk）システムコールの発行により解除されます。ただし、wup_tsk（またはiwup_tsk）システムコールによる起床要求が既に発行されていた場合、待ち状態へ移行せずにそのまま実行を続けます。

本システムコールにより待ち状態となっているときに、sus_tsk（またはisus_tsk）システムコールが発行された場合、wup_tsk（またはiwup_tsk）システムコールにより待ち状態が解除されても、まだ強制待ち（SUSPEND）状態であり、rsm_tsk（またはirsm_tsk）システムコールの発行までタスクの実行は再開されません。

関連システムコール

can_wup,ican_wup,irel_wai,iwup_tsk,rel_wai,wai_tsk,wup_tsk

３．３．６１　snd_msg（Send Message to Mailbox）メイルボックスへ送信する

［タスク部用］

Ｃ言語インタフェース

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

【パラメータ】

ID	mbxid	メイルボックスID（1～hi_maxmbxid[*1]）
T_MSG	*pk_msg	送信メッセージの先頭アドレス

【リターンパラメータ】

ER	ercd	エラーコード

【パケットの構造】

typedef	struct	t_msg {	
	UW	msghead;	ＯＳ管理領域
	VB	msgcont[n];	メッセージの内容
	} T_MSG;		

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffe5）
R4	mbxid	メイルボックスID（1～hi_maxmbxid[*1]）
R5	pk_msg	送信メッセージの先頭アドレス

【リターンパラメータ】

R0	ercd	エラーコード

【パケットの構造】

[****]	msghead	ＯＳ管理領域
[*]	msgcont[n]	メッセージの内容

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（pk_msg が 4 の倍数以外または0）
E_ILMSG	H'fffff8d7(-H'729)	[k]	不正メッセージ形式（メッセージ先頭 4 バイトが 0 でない）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（mbxid < 0, mbxid > hi_maxmbxid[*1]）
		[p]	予約ID（mbxid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxmbxid　：　セットアップテーブルで定義した最大メイルボックスID

解　　説

mbxid で示されたメイルボックスに pk_msg で示されたメッセージを送信します。メッセージの内容はコピーされず、アドレス（pk_msg の値）のみが渡されます。このため、本システムコールでメッセージ送信後、メッセージ内容を書き換えると、rcv_msgまたはprcv_msgシステムコールで正しい内容のメッセージを受け取ることができなくなるので注意してください。

対象メイルボックスに待ちタスクがある場合、待ち行列の先頭タスクにメッセージを渡し、そのタスクを実行可能（READY）状態へ移行します。待ちタスクがない場合、メッセージをメイルボックスに入れ、メッセージの行列につなぎます。メッセージの行列へのつなぎ方はFIFO(First-In First-Out)です。

メッセージは、必ずRAM領域に確保してください。メッセージの長さはＯＳで管理していませんので、ユーザ側で管理する必要があります。ユーザがメッセージとして利用できる領域は、メッセージの先頭アドレス＋4バイト目からです。メッセージの先頭 4バイトはＯＳで使用するため、初期値として必ず 0 を設定し、送信後も書き換えないでください。0 が設定されていない場合は、エラーコードとしてE_ILMSGを返します。

関連システムコール

imbx_sts,iprcv_msg,isnd_msg,mbx_sts,prcv_msg,rcv_msg

３．３．６２　sta_tsk（Start Task）タスクを起動する［タスク部用］

C言語インタフェース

ER ercd = sta_tsk(ID tskid);

【パラメータ】

ID	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffefe）
R4	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[p]	予約ID（tskid = 0）
		[k]	未登録（タスクが生成されていない）
E_NODMT	H'fffff7ca(-H'836)	[k]	自タスク指定（tskid が自タスクである）
		[k]	タスクが休止（DORMANT）状態でない
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxtskid　：　セットアップテーブルで定義した最大タスクID

解　説

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）状態へ移行します。

起動時のタスク優先度は、cre_tskシステムコール発行時（またはセットアップテーブルにおける初期登録タスクの定義）に指定した初期タスク優先度となります。

本システムコールによる起動要求のキューイングは行ないません。対象タスクが休止状態にない場合に発行された起動要求に対しては、発行タスクにエラーコードとしてE_NODMTを返します。

tskid は、起動する他タスクのIDを指定します。tskid に自タスクのIDを指定した場合は、エラーコードとしてE_NODMTを返します。

HI-SH77では、複数のタスクでスタック領域を共有する機能（共有スタック機能）を持っています。セットアップテーブルで初期タスクスタックポインタに同じ値を登録したタスクID同士は、それらのタスク間でひとつのスタックを共有する（共有スタック）ことになります。

本システムコール発行時に、tskidで示されたタスクのスタック（共有スタック）が開放（どのタスクも使用していない）されている場合は、tskidで示されたタスクが共有スタックを占有し、実行可能状態に移行します。

共有スタックが占有されている（他のタスクが使用中）場合は、スタック領域を使用できないため、タスクは待ち（WAIT）状態になり、共有スタックの待ち行列につながれます。待ち行列のつながれ方は、FIFO(First-In First-Out)です。共有スタック待ち状態のタスクに対して本システムコールを発行すると、発行タスクにエラーコードとしてE_OKを返します。

関連システムコール

cre_tsk,ista_tsk,ter_tsk

３．３．６３　sus_tsk（Suspend Task）タスクを強制待ち状態へ移行する［タスク部用］

C言語インタフェース

ER ercd = sus_tsk(ID tskid);

【パラメータ】

ID	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffef4）
R4	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_SELF	H'fffff7cf(-H'831)	[k]	自タスク指定（tskid が自タスクである）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[p]	予約ID（tskid = 0）
		[k]	未登録（タスクが生成されていない）
E_DMT	H'fffff7cb(-H'835)	[k]	タスクが休止（DORMANT）状態である
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）
E_QOVR	H'fffff4b7(-H'b49)	[k]	キューイングのオーバフロー（既にタスクが強制待ち（SUSPEND）状態である）

*1　hi_maxtskid　：　セットアップテーブルで定義した最大タスクID

解　説

tskid で示されたタスクの実行を中断させ、強制待ち（SUSPEND）状態へ移行します。

強制待ち状態は、rsm_tsk（またはirsm_tsk）システムコールの発行により解除されます。

tskid で示されたタスクが待ち（WAIT）状態にある場合は、二重待ち（WAIT-SUSPEND）状態へ移行します。

強制待ち状態は、他タスクからのシステムコールによる中断状態を意味するものなので、本システムコールで自タスクを指定することはできません。

本システムコールの要求のキューイングは行ないません。強制待ち状態のときに本システムコールを発行した場合、エラーコードとしてE_QOVRを返します。

関連システムコール

irsm_tsk,isus_tsk,rsm_tsk

３．３．６４　sys_cal（System Call）拡張ＳＶＣの発行［タスク部用］

C言語インタフェース

ER ercd = sys_cal(FN fncd, VW para1, VW para2, VW para3, VW para4);

【パラメータ】

FN	fncd	拡張SVCの機能コード（1 ≦ fncd ≦ hi_maxsvccd[*1]）
VW	para1	パラメータ1
VW	para2	パラメータ2
VW	para3	パラメータ3
VW	para4	パラメータ4

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	拡張SVC用トラップ

【パラメータ】

R0	fncd	機能コード（1 ≦ fncd ≦ hi_maxsvccd[*1]）
R4	para1	パラメータ1
R5	para2	パラメータ2
R6	para3	パラメータ3
R7	para4	パラメータ4

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）
E_RSFN	H'fffff9ec(-H'614)	[k]	予約機能コード番号（未定義の機能コード指定）

*1 hi_maxsvccd　：　セットアップテーブルで定義した最大拡張SVC機能コード

E_CTX以外のエラーコードは、拡張SVCハンドラの関数値(R0)の内容を返します。**「付録Ｅ．エラーコード一覧」**を参考に、正常終了はE_OK、エラー発生は負の値となるように拡張SVCハンドラを作成してください。

解　説

カーネルは、fncdで指定された機能コードに対応する拡張SVCハンドラを調べ、呼び出して実行します。

拡張SVCハンドラに引き渡すことができるパラメータは、機能コード(R0)と最大4つまでの入力パラメータ(R4～R7)です。C言語で記述する場合、入力パラメータは必ず４つ指定してください。詳細は**「４．５ C言語による拡張SVCハンドラの記述方法」**を参照してください。また、リターンパラメータ ercdは、拡張SVCハンドラの関数値(R0)です。

拡張SVCハンドラは、セットアップテーブルに定義します。

拡張SVCハンドラの定義方法は、**『HI-SH77構築マニュアル』**を参照してください。

３．３．６５　sys_clk（Count system clock）カーネルの時間管理処理を要求する

［非タスク部用］

C言語インタフェース

```
void    sys_clk(void);
```

アセンブラインタフェース

```
TRAPA   #60    システムコール用トラップ
```

【パラメータ】

なし

【リターンパラメータ】

なし

エラーコード

E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー*1（タスク部から発行できない）

*1　エラー検出時は、R4に- 4（異常終了種別）、R5にエラーコード、R6にsys_clkシステムコール発行アドレス+2を設定し、システム異常終了処理（hi_sysdwn関数）に制御を渡します。

解　　説

本システムコールは、システムクロックを更新するシステムコールで、本システムコールの発行により、時間管理を実現することができます。

具体的には、本システムコールの発行により次のことが行なわれます。

・システムクロックの更新(+1)

・wai_tskシステム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

本システムコールは、タイマ割込みハンドラから発行してください。

なお、タイマ割込みハンドラは、ret_intシステムコールにより割込み復帰する必要があります。

本システムコールは、HI-SH77の独自仕様です。

関連システムコール

get_tim,iget_tim,iset_tim,set_tim,wai_tsk

３．３．６６　ter_tsk（Terminate Task）他タスクを強制的に異常終了させる［タスク部用］

C言語インタフェース

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

【パラメータ】

ID	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffefa）
R4	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_SELF	H'fffff7cf(-H'831)	[k]	自タスク指定（tskid が自タスクである）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[p]	予約ID（tskid = 0）
		[k]	未登録（タスクが生成されていない）
E_DMT	H'fffff7cb(-H'835)	[k]	タスクが休止（DORMANT）状態である
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxtskid　：　セットアップテーブルで定義した最大タスクID

解　説

tskid で示された他タスクを強制的に異常終了させます。

異常終了させた他タスクは、休止（DORMANT）状態へ移行します。

tskid には、強制的に異常終了させる他タスクのIDを指定します。tskid に自タスクのIDを指定した場合は、エラーコードとしてE_SELFを返します。

本システムコールは、タスクが占有していた資源（セマフォにより獲得したもの）とメモリブロックを自動的に解放する機能はありません。資源とメモリブロックの解放は、本システムコール発行前に行なってください。

本システムコールの発行により、tskid で示された他タスクが占有していたスタックを解放します。このスタックの待ち行列に別のタスクがつながれている場合（共有スタック）、先頭のタスクをスタックの待ち行列から外し、実行可能（READY）状態へ移行します。

関連システムコール

ext_tsk,ista_tsk,sta_tsk

３．３．６７　tsk_sts（Get Task Status）タスク状態を見る［タスク部用］

C言語インタフェース

```
ER  ercd = tsk_sts(UH *p_tskstat, TPRI *p_tskpri, ID tskid);
```

【パラメータ】

UH	*p_tskstat	タスク状態を返す領域の先頭アドレス
TPRI	*p_tskpri	現在タスク優先度を返す領域の先頭アドレス
ID	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード
UH	*p_tskstat	タスク状態を格納した領域の先頭アドレス
TPRI	*p_tskpri	現在タスク優先度を格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffef5）
R4	p_tskstat	タスク状態を返す領域の先頭アドレス
R5	p_tskpri	現在タスク優先度を返す領域の先頭アドレス
R6	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_tskstat	タスク状態を格納した領域の先頭アドレス
R5	p_tskpri	現在タスク優先度を格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_tskstatが奇数または0，p_tskpriが奇数または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[k]	未登録（タスクが生成されていない）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）

*1　hi_maxtskid：　セットアップテーブルで定義した最大タスクID

解　　説

tskid で示されたタスクの状態を読み出し、現在のタスク状態、タスク優先度をそれぞれp_tskstat, p_tskpriの指す領域に返します。

p_tskstat, p_tskpri の指す領域として、それぞれ2バイトのRAM領域が必要です。

p_tskstatの指す領域には、次の値を返します。

p_tskstat:

wupcnt	H'0000 - H'000F	wup_tsk,iwup_tskシステムコールのキューイング数
TTS_RDY	H'0010	実行状態／実行可能状態
TTS_WAI	H'0020	待ち状態
TTS_SUS	H'0040	強制待ち状態
TTS_WAS	H'0060	二重待ち状態
TTS_DMT	H'0080	休止状態
TTS_STK	H'4000	共有スタック解放待ち状態
TTW_SLP	H'0100	slp_tskによる待ち
TTW_WAI	H'0200	wai_tskによる待ち
TTW_FLG	H'0400	wai_flgによる待ち
TTW_SEM	H'0800	wai_semによる待ち
TTW_MBX	H'1000	rcv_msgによる待ち
TTW_MPL	H'2000	get_blkによる待ち

p_tskpri の指す領域には、現在のタスク優先度を返します。

tskid = TSK_SELF(H'0) の指定により自タスクの指定になりますが、本システムコールでは自タスクのタスクIDは返しません。自タスクのタスクIDを得る場合は、get_tidシステムコールを発行してください。

関連システムコール

itsk_sts

３．３．６８　wai_flg（Wait Event Flag）イベントフラグを待つ［タスク部用］

C言語インタフェース

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

【パラメータ】

UW	*p_flgptn	待ち解除時のビットパターンを返す領域の先頭アドレス
ID	flgid	イベントフラグID（1～hi_maxflgid[*1]）
UW	waiptn	待ちビットパターン
UW	wfmode	待ちモード

【リターンパラメータ】

ER	ercd	エラーコード
UW	*p_flgptn	待ち解除時のビットパターンを格納した領域の先頭アドレス

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffeec）
R4	p_flgptn	待ち解除時のビットパターンを返す領域の先頭アドレス
R5	flgid	イベントフラグID（1～hi_maxflgid[*1]）
R6	waiptn	待ちビットパターン
R7	wfmode	待ちモード

【リターンパラメータ】

R0	ercd	エラーコード
R4	p_flgptn	待ち解除時のビットパターンを格納した領域の先頭アドレス

エラーコード

E_OK	H'00000000	[k]	正常終了
E_PAR	H'fffff8df(-H'721)	[p]	パラメータエラー（waiptn = 0, wfmode が不正）
E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス（p_flgptn が 4 の倍数以外または0）
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（flgid < 0, flgid > hi_maxflgid[*1]）
		[p]	予約ID（flgid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）
E_QOVR	H'fffff4b7(-H'b49)	[k]	キューイングのオーバフロー（既に待ちタスクが存在する）
E_RLWAI	H'fffff2aa(-H'd56)	[k]	待ち（WAIT）状態強制解除（待ちの間にrel_waiまたはirel_waiシステムコールが発行された）

*1　hi_maxflgid　　：　セットアップテーブルで定義した最大イベントフラグID

解　　説

flgid で示されたイベントフラグ（３２ビット）がセットされるのを、wfmode で示された待ち条件にしたがって待ちます。

wfmode では、次のような指定を行なうことができます。

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

TWF_ANDW	H'00000000	AND待ち
TWF_ORW	H'00000002	OR待ち
TWF_CLR	H'00000001	クリア指定

TWF_ANDWでは、flgid で示されたイベントフラグのうち waiptn で指定したビットのすべてがセットされるのを待ちます。TWF_ORWでは、flgid で示されたイベントフラグのうち waiptn で指定したビットのいずれかがセットされるのを待ちます。

TWF_CLRの指定が無い場合には、条件が満足されてタスクが待ち解除になってもイベントフラグの値は変化しません。一方、TWF_CLRの指定がある場合には、条件が満足されてタスクが待ち解除になると、イベントフラグの値（全部のビット）が0にクリアされます。

待ち状態が解除されるときのイベントフラグの値（TWF_CLR指定の場合は、イベントフラグがクリアされる前の値）を p_flgptn の指す領域に返します。p_flgptnの 指す領域として、4 バイトのRAM領域が必要です。

１つのイベントフラグに対して複数のタスクが待ち（WAIT）状態へ移行することはできません。したがって、すでにタスクが待っているイベントフラグに対して、wai_flgシステムコールを発行すると、エラーコードとしてE_QOVRを返します。

関連システムコール

clr_flg,flg_sts,iclr_flg,iflg_sts,ipol_flg,irel_wai,iset_flg,pol_flg,rel_wai,set_flg

３．３．６９　wai_sem（Wait on Semaphore） セマフォに対する待ち操作（P命令）

［タスク部用］

C言語インタフェース

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

【パラメータ】

ID	semid	セマフォID（1～hi_maxsemid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffe8）
R4	semid	セマフォID（1～hi_maxsemid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（semid < 0, semid > hi_maxsemid[*1]）
		[p]	予約ID（semid = 0）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）
E_RLWAI	H'fffff2aa(-H'd56)	[k]	待ち（WAIT）状態強制解除（待ちの間にrel_waiまたはirel_waiシステムコールが発行された）

*1　hi_maxsemid　：　セットアップテーブルで定義した最大セマフォID

解　　説

semid で示されたセマフォのカウント値が 1 以上の場合、セマフォのカウント値を 1 減らして、発行タスクは実行を継続します。セマフォのカウント値が 0 の場合は、セマフォのカウント値を変更せず、発行タスクはそのセマフォに対する待ち行列につながれ、待ち（WAIT）状態へ移行します。セマフォに対する待ち行列へのつながれ方はFIFO（First-In First-Out）です。

関連システムコール

ipreq_sem,irel_wai,isem_sts,isig_sem,preq_sem,rel_wai,sem_sts,sig_sem

３．３．７０　wai_tsk（Wait Task）自タスクを一定時間待ち状態へ移行する［タスク部用］

C言語インタフェース

```
ER  ercd = wai_tsk(TMO tmout);
```

【パラメータ】

TMO	tmout	タイムアウト指定（H'00000001～H'7fffffff）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'ffffffef1）
R4	tmout	タイムアウト指定（H'00000001～H'7fffffff）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_TNOSPT	H'fffff9ed(-H'613)	[p]	時間管理機能が組み込まれていない
E_ILTIME	H'fffff8d9(-H'727)	[p]	不正時間指定（tmout < -1）
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）
E_TMOUT	H'fffff2ab(-H'd55)	[k]	タイムアウト
		[k]	起床要求なし（tmout = 0 指定でwupcnt = 0の場合）
E_RLWAI	H'fffff2aa(-H'd56)	[k]	待ち（WAIT）状態強制解除（待ちの間にrel_waiまたはirel_waiシステムコールが発行された）

解　説

自タスクを一定時間だけ実行（RUN）状態から待ち（WAIT）状態へ移行します。

この待ち状態は、本タスクを対象としたwup_tsk（またはiwup_tsk）システムコールの発行、または tmout で指定した時間の経過により解除されます。

wup_tsk（またはiwup_tsk）システムコールが発行された場合は正常終了、時間経過の場合はタイムアウトエラー（E_TMOUT）となります。

tmout には、H'00000001～H'7fffffff（３２ビット正整数）の値を指定できます。

tmout = TMO_FEVR(-1) の指定は、タイムアウト指定を行なわないことを示します。この場合、wup_tsk（またはiwup_tsk）システムコールが発行されるまで永久に待ち状態になるため、slp_tskシステムコールと同じ動作になります。

tmout = 0 の指定をした場合、起床要求のキューイング数（wupcnt）が 0であれば待ち状態にならずにタイムアウトエラー（E_TMOUT）を返します。

なお、タイムアウト処理はタイマ割込みハンドラから発行されるsys_clkシステムコールにより行なわれ、tmout は要求タイマ割り込み回数です。すなわち、指定時間経過は、ハードウェアタイマ周期時間の tmout 倍となります。

関連システムコール

can_wup,ican_wup,irel_wai,iwup_tsk,rel_wai,slp_tsk,sys_clk,wup_tsk

３．３．７１　wup_tsk（Wakeup Task）待ち状態のタスクを起床する［タスク部用］

C言語インタフェース

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

【パラメータ】

ID	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

ER	ercd	エラーコード

アセンブラインタフェース

TRAPA	#63	システムコール用トラップ

【パラメータ】

R0	fncd	機能コード（H'fffffef0）
R4	tskid	タスクID（1～hi_maxtskid[*1]）

【リターンパラメータ】

R0	ercd	エラーコード

エラーコード

E_OK	H'00000000	[k]	正常終了
E_SELF	H'fffff7cf(-H'831)	[k]	自タスク指定
E_NOEXS	H'fffff7cc(-H'834)	[p]	ID範囲外（tskid < 0, tskid > hi_maxtskid[*1]）
		[p]	予約ID（tskid = 0）
		[k]	未登録（タスクが生成されていない）
E_DMT	H'fffff7cb(-H'835)	[k]	タスクが休止（DORMANT）状態である
E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー（非タスク部から発行できない）
E_QOVR	H'fffff4b7(-H'b49)	[k]	キューイングのオーバフロー（wupcnt > H'0f）

*1　hi_maxtskid：　セットアップテーブルで定義した最大タスクID

解　説

slp_tskシステムコールまたはwai_tskシステムコールの発行により待ち（WAIT）状態になっているタスクを、実行可能（READY）状態へ移行します。

本システムコールでは、自タスクを指定することはできません。

対象タスクがslp_tskシステムコール、またはwai_tskシステムコールを発行しておらず、待ち状態でない場合には、本システムコールの起床要求はキューイングされ、後に対象タスクがslp_tskシステムコール、またはwai_tskシステムコールを発行したときに有効になります。

起床要求（wupcnt）のキューイング数の最大値は H'f(15) です。この値を越えた場合にはエラーコードとしてE_QOVRを返します。

関連システムコール

can_wup,ican_wup,iwup_tsk,slp_tsk,wai_tsk

４．Ｃ言語によるプログラムの作成方法

タスクや割込みハンドラなど、ユーザが作成するプログラムはすべてC言語で記述することができます。
本章では、SHシリーズCコンパイラを用いてC言語プログラムを作成する方法を説明します。
SHシリーズCコンパイラについての詳細は、『**SHシリーズCコンパイラユーザーズマニュアル**』を参照してください。

４．１　Ｃ言語インタフェース

HI-SH77では、C言語インタフェースをサポートするヘッダファイルとして**表４－１**に示すプログラムを提供しています。

表４－１　C言語インタフェースの構成ファイル

項番	ファイル名	内　　容
1	**itron.h**	C言語インタフェースヘッダ （C言語プログラムで使用できる記号定数、データ型およびシステムコールのプロトタイプの宣言）

sys_clkシステムコールは、SHシリーズCコンパイラの組込み関数**trapa()** を利用して発行します。
また、**ret_int**システムコールおよび**ret_exc**システムコールは、SHシリーズCコンパイラの割込み関数 **#pragma interrupt**を利用して発行します。詳細は**「４．４　C言語による割込みハンドラの記述方法」**および**「４．９　C言語による例外処理ルーチンの記述方法」**を参照してください。
その他のシステムコールは、SHシリーズCコンパイラの組込み関数**trapa_svc()** を利用して発行します。

必ず、以下のように必要なファイルをインクルードしてください。

```
#include <umachine.h>  ･･･････････  (a)
#include "itron.h"     ･･･････････  (b)
```

(a)SHシリーズCコンパイラの提供するユーザモード用ヘッダファイル
(b)C言語インタフェースヘッダファイル

４．２　スタックの使用量について

C言語で記述されたプログラムは、通常、関数の呼び出しのたびに、引数の引き渡しと関数内の作業領域としてスタック領域が割り付けられます。
使用するスタックの使用量は、SHシリーズCコンパイラの出力するオブジェクトリストから算出することができます。この値を元に各プログラムのスタック使用量を算出し、必要なスタック領域を確保してください。

ソースプログラムをコンパイルした場合は、必ずスタック使用サイズを算出してスタックを確保してください。算出方法は、SHシリーズCコンパイラユーザーズマニュアルのシステム組み込み編「**２．２　動的領域の割り付け**」を参照してください。また、このサイズを「ユーザ（タスク、割り込みハンドラ）独自」に設定してください。
標準提供のサンプルプログラムをそのまま使用する場合も、使用するコンパイラバージョンによってはスタック使用サイズが変わる場合があるので、必ずスタック使用サイズを確認の上、スタック定義ファイルを設定してください。

４．３　C言語によるタスクの記述方法

C言語でタスクを記述する場合の例を**図４－１**に示します。

タスクとして実行する関数は、セットアップテーブルに初期登録するか、または**cre_tsk**システムコールにより生成する必要があります。

タスクのデータ型は、**TASK**を使用してください。

タスクは、**return**文を用いてリターン値を返すことはできません。

プログラムの最後で**ext_tsk**または**exd_tsk**システムコールを発行し、終了させます。これを行なわないと、タスク終了時の制御の渡し先が不定になり、システムの正常な動作は保証されません。

```
#include <umachine.h>
#include "itron.h"

TASK        taskxx()
{
ID          tskid;                  /* task id                  */
ER          ercd;                   /* error code               */

/*
 *         ........
 *         ........
 */
            ercd = wup_tsk(tskid);  /* wake up task             */
/*
 *         ........
 *         ........
 */
            exd_tsk();              /* exit and delete task     */
}
```

図４－１　C言語によるタスクの記述例

４．４　C言語による割込みハンドラの記述方法

４．４．１　割込みハンドラの基本形

SHシリーズCコンパイラのプリプロセッサ制御文(#pragma)を用いることにより、割込みハンドラをC言語で記述できます。

図４－２に割込みハンドラの記述例を示します。

#pragma interrupt を用いて割込みハンドラとなる関数と割込み仕様を宣言します。図４－２では、inthdr関数を割込みハンドラとして宣言しています。

割込み仕様として、「スタック切り換え指定」と「トラップ命令リターン指定」を行ないます。

「スタック切り換え指定」は、割込みハンドラ開始時に切り換えるスタック領域の指定であり、初期スタックポインタを"sp=<アドレス>"で指定します。スタック領域は、割込みレベル固有の領域を指定してください。

「トラップ命令リターン指定」は、割込みハンドラ終了時の復帰方法の指定であり、実行するTRAPA命令の番号を"tn=<トラップ番号>"で指定します。割込みハンドラは終了時にret_intシステムコールを発行する必要がありますので、"tn=61"と指定してください。これにより、割込みハンドラ終了時、TRAPA#61命令が実行されます。

```
#include  <umachine.h>
#include  <smachine.h>
#include  "itron.h"
#include  "hiintstk.h"

extern VW hi_intstkxx[];
static const VP  p_intstkxx  =  (VP)&hi_intstkxx[(hi_intstkszxx) / sizeof(VW)];

#pragma   interrupt   (inthdr(sp = p_intstkxx, tn = 61))
INTHDR    inthdr(void)
{
ID        tskid;                        /*  task id                 */
ER        ercd;                         /* error code               */

/*
 *       割込みマスクレベルの設定、例外マスク解除などの処理
 *       ........
 */
          ercd = iwup_tsk(tskid);       /* wake up task             */
/*
 *       ........
 *       ........
 */

}
```

図４－２　C言語による割込みハンドラの記述例

表４－２に割込みハンドラの記述方法と割込み仕様の指定内容一覧を示します。

割込みハンドラのデータ型は、**INTHDR**を使用してください。

なお、割込みハンドラは、**return**文を用いてリターン値を返すことはできません。

割込みハンドラ起動時は、例外がマスクされており、また割込みマスクは割込み発生前のレベルに設定されています。したがって、割込みハンドラの先頭で割込みマスクレベルの設定を行い、必要に応じて例外マスクを解除してください。割込みマスクレベルの設定は、**SH**シリーズ**C**コンパイラが提供する組込み関数"**void set_imask(int mask)**"を利用して行うことができます。また、組込み関数"**void set_cr(int cr)**"を利用すると割込みマスクレベルの設定と例外マスクの解除を同時に行うことができます。組込み関数"**void set_imask(int mask)**"、"**void set_cr(int cr)**"を使用する場合は、"**#include <smachine.h>**"を宣言してください。

割込みハンドラ内はタスク独立部（処理がどのタスクに関係するものかを特定できない）として実行されます。システムコールを発行するときは、例外マスク解除状態で非タスク部用のシステムコールを使用してください。

表４－２　割込みハンドラの記述方法と割込み仕様の指定内容一覧

#pragma interrupt (関数名 [(割込み仕様)] [, 関数名 [(割込み仕様)]...])			
項番	割込み仕様の項目	形式	指定内容
1	スタック 切り換え指定	**sp=**	割込みハンドラが使用するスタック領域の初期スタックポインタを指定します。スタック領域は、発生する割込みのレベル毎に確保してください。標準提供では、割込みハンドラ用スタック領域定義プログラム**(hiintstk.c)**で確保しています。
2	トラップ命令 リターン指定	**tn=**	割込みハンドラ終了時に発行するシステムコールのトラップ番号を指定します。**ret_int**システムコールは**TRAPA#61**で実行するので"**tn=61**"と指定してください。

４．４．２　タイマ割込みハンドラの記述方法

図４－３にタイマ割込みハンドラの記述例を示します。

通常の割込みハンドラの宣言と同様に、**#pragma interrupt**を用いてタイマ割込みハンドラとなる関数と割込み仕様を宣言します。"**sp=<**アドレス**>**"を用いてスタック切り換え指定を行ない、"**tn=61**"により**ret_int**システムコールによる終了を指定します。

タイマ割込みハンドラでは、**sys_clk**システムコールを発行します。**sys_clk**システムコールは、**SH**シリーズ**C**コンパイラが提供する組込み関数"**int trapa(int trap_no)**"を利用して実現しています。したがって、必ず"**#include <umachine.h>**"を宣言してください。

タイマ割込みハンドラのデータ型は、**TMRHDR**を使用してください。

```
#include  <umachine.h>
#include  <smachine.h>
#include  "itron.h"
#include  "hiintstk.h"

extern VW hi_intstktmr[];
static const VP   p_intstktmr  =  (VP)&hi_intstktmr[(hi_intstksztmr) / sizeof(VW)];

#pragma   interrupt   (tmrhdr(sp = p_intstktmr, tn = 61))
TMRHDR    tmrhdr(void)
{
ER        ercd;                     /* error code             */
/*
 *       割込みマスクレベルの設定、例外マスク解除などの処理
 *       タイマ割込みの解除など、タイマハードウェア処理
 */
          ercd=sys_clk();           /* count system clock     */

}
```

図４－３　C言語によるタイマ割込みハンドラの記述例

４．４．３　カーネル割込みマスクレベルより高い割込みハンドラの記述方法

カーネル割込みマスクレベルより高い割込みの割込みハンドラでは、**ret_int**以外のシステムコールは発行できません。システムコールを発行した場合、システムの正常な動作は保証されません。

「スタック切り換え指定」で指定するスタック領域は、割込みレベル固有の領域を指定してください。

割込みハンドラ終了時には**ret_int**システムコールを発行する必要がありますので、「トラップ命令リターン指定」として**"tn=61"**と指定してください。

```
#include  <smachine.h>
#include  "itron.h"
#include  "hiintstk.h"

extern VW hi_intstkxx[];
static const VP   p_intstkxx  =  (VP)&hi_intstkxx[(hi_intstkszxx) / sizeof(VW)];

#pragma   interrupt   (minthdr(sp = p_intstkxx, tn = 61))
INTHDR    minthdr(void)
{
ID        tskid;                    /* task id                */
ER        ercd;                     /* error code             */
/*
 *       割込みマスクレベルの設定、例外マスク解除などの処理
 *       ・・・・・・・・
 */
}
```

図４－４　C言語によるカーネル割込みマスクレベルより高い割込みハンドラの記述例

４．５　C言語による拡張SVCハンドラの記述方法

C言語による拡張SVCハンドラの記述例を図４－５に示します。

拡張SVCハンドラは、通常の関数と同様に記述することができます。拡張SVCハンドラは、その先頭アドレスをセットアップテーブルに定義してカーネルに登録する必要があります。

拡張SVCハンドラには入力パラメータとして、拡張SVCを発行する際に指定した最大4つのパラメータが渡されます。また、出力パラメータとして、拡張SVCハンドラのリターン値を返すことができます。

拡張SVCハンドラのデータ型は、SVCHDRを使用してください。

拡張SVCハンドラ内はOS拡張部（ユーザがOSを拡張した部分）として実行されます。システムコールを発行するときは、非タスク部用のシステムコールを使用してください。

```
#include  <umachine.h>
#include  "itron.h"

SVCHDR    stp_mtr(mtrid)            /* stop motor(extended svc)  */
ID        mtrid;                    /* motor id                  */
{
ER        ercd;                     /* error code                */

/*
 *       拡張SVCハンドラ処理
 */

          ercd = isig_sem(mtrid);   /* signal semaphore          */
          if (ercd < E_OK)
                goto exit;
exit:
          return(ercd);
}
```

図４－５　C言語による拡張SVCハンドラの記述例

図４－６に拡張**SVC**の発行例を示します。

拡張**SVC**の発行は、**sys_cal**システムコールを使用する方法と**trapa_svc()**を使用する方法があります。

sys_calシステムコールは拡張**SVC**の汎用C言語インタフェースであり、機能コード１つと入力パラメータ４つ全て指定する必要があります。

trapa_svc()を使用した場合は、入力パラメータを省略することができます。この際、**trapa_svc()**の第１引数は６３、第２引数は機能コード、第３から第６引数には入力パラメータを指定してください。

図４－６に示すように**#define**文を使用してマクロ定義し、ユーザ仕様の名称で拡張**SVC**を発行すれば、プログラムがわかりやすくなります。

```
#include  <umachine.h>
#include  "itron.h"

#define   MOTOR1  1
#define   MOTOR2  2
#define   LEFT              0x01
#define   RIGHT             0x02

#define   sta_mtr(mtrid, dir)  sys_cal((FN)1,  (ID)mtrid,  (UB)dir, 0, 0)
#define   stp_mtr(mtrid)       trapa_svc((int)63, (FN)2, (ID)mtrid)

TASK      taskxx(void)
{
ER        ercd;                           /* error code                    */

          ercd = sta_mtr(MOTOR1, LEFT);   /* start motor                   */
          if (ercd < E_OK)
              goto exit;

          ercd = slp_tsk();               /* sleep task                    */

          ercd = stp_mtr(MOTOR1);         /* stop motor                    */
          if (ercd < E_OK)
              goto exit;

/*
 *       ........
 *       ........
 */
exit:
          ext_tsk();                      /* exit task                     */
}
```

図４－６　C言語による拡張SVCの発行例

４．６　C言語によるMCU初期化ルーチンの記述方法

C言語によるMCU初期化ルーチンの記述例を**図４－７**に示します。

MCU初期化ルーチンの先頭アドレスは、ベクタテーブルのリセットPCに登録してください。また、MCU初期化ルーチンが使用するスタック領域の初期スタックポインタを、システム構築用リンケージサブコマンドファイル(himake.sub)に登録してください。リセットSPに登録するスタック領域は、リセット直後に利用可能なRAM領域の最終アドレス＋１を指定してください。

初期スタックポインタの登録方法については、『**HI-SH77構築マニュアル**』を参照してください。

MCU初期化ルーチンのデータ型は、voidを使用してください。

MCU初期化ルーチンは、return文を用いてリターン値を返すことはできません。

MCU初期化ルーチンでは、キャッシュ、MMU、バスコントローラなどのMCU初期化を行ないます。処理が終了したら、最後にカーネルのシステム起動処理(_OHrs__knl関数)を呼び出してカーネルを起動します。カーネルの起動関数名は、"_OHrs__knl"と名称が決められています。

MCU初期化ルーチン内はまだカーネルが起動していないため、システムコールは発行できません。

```
#include    <umachine.h>
#include    <smachine.h>
#include    "itron.h"

#define IOBASE    0xfffffe80            /* I/O base address    = 0xfffffe80 */
#define MMUCR     (0xffffffe0 - IOBASE) /* CCN MMUCR         address offset */
#define CCR       (0xffffffec - IOBASE) /* CCN CCR           address offset */

#define MMU_OFF    (UW)0x00000000       /* MMU   disable data               */
#define CACHE_ON   (UW)0x00000001       /* CACHE enable  data               */
#define CACHE_OFF  (UW)0x00000000       /* CACHE disable data               */

#define RAMSTA    (VW *)0x0c000000      /* RAM start address   = 0x0c000000 */
#define RAMEND    (VW *)0x0c07fffc      /* RAM end address     = 0x0c07fffc */

extern  void    _OHrs__knl(void);       /* kernel reset routine             */
/* extern void    _INITSCT(void);          section initialization           */
                                        /*                    routine for C */

void    hi_mcuini(void)
{
    register    VW      *p;             /* pointer to memory                */

    set_cr(MD_SET | (SR_IMS15 << 4));   /* clear BL, set imask              */

    set_gbr((VP)IOBASE);                /* set I/O base address to GBR      */
    gbr_write_long(CCR, CACHE_OFF);     /* CACHE disable                    */
    gbr_write_long(MMUCR, MMU_OFF);     /* MMU disable                      */

    for(p = RAMSTA; p <= RAMEND; p++)   /* RAM clear                        */
        *p = 0x00000000;

/*   _INITSCT();                           call section initialization      */
                                        /*                    routine for C */

    _OHrs__knl();                       /* go to kernel reset routine       */
}
```

図４－７　　C言語によるMCU初期化ルーチンの記述例

４．７　C言語によるシステム初期化ハンドラの記述方法

C言語によるシステム初期化ハンドラの記述例を**図４－８**に示します。

システム初期化ハンドラとして使用する関数は、その先頭アドレスをセットアップテーブルに定義してカーネルに登録する必要があります。

システム初期化ハンドラのデータ型は、**INIHDR**を使用してください。

システム初期化ハンドラは、**return**文を用いてリターン値を返すことはできません。

システム初期化ハンドラ内は**OS**拡張部（ユーザが**OS**を拡張した部分）として実行されます。システムコールを発行するときは、例外マスク解除状態で非タスク部用のシステムコールを使用してください。

なお、システム初期化ハンドラ内でエラーが発生したときは、システム異常終了処理プログラム**(hi_sysdwn)**を呼び出してください。

```
#include    <umachine.h>
#include    "itron.h"

extern void hi_sysdwn(W, ER, UW);

INIHDR  hi_sysini(void)
{
    ER      ercd;                       /* error code                   */

/*
 *        資源の初期化
 */

    ercd = ista_tsk(CTSKID);            /* start task                   */
    if(ercd < E_OK)                     /* if ista_tsk fail             */
        hi_sysdwn((W)1, ercd, (UW)0);   /* goto hi_sysdwn()             */

}
```

図４－８　　C言語によるシステム初期化ハンドラの記述例

４．８　C言語によるシステム異常終了処理ルーチンの記述方法

システムに異常が発生した場合、カーネル割込みマスクレベルにマスクしてシステム異常終了処理ルーチンに制御が渡されます。ユーザプログラムからシステム異常終了処理ルーチンに制御を渡す場合は、特権モードから行ってください。

C言語プログラムにおけるシステム異常終了処理ルーチンは、**"hi_sysdwn"**と名称が決められています。**図４－９**にシステム異常終了処理ルーチンの記述例を示します。

hi_sysdwn()関数には入力パラメータとして、エラー種別、エラーコード、およびシステムダウン情報が渡されます。

システム異常終了処理ルーチンでは、異常内容に応じた処理を行なうことができます。ただし、システムコールなどカーネルの機能は使用できません。

```
#include    <smachine.h>
#include    "itron.h"

void        hi_sysdwn(type, ercd, inf)
W           type;           /* type of system down                          */
                            /*      type >=  1 : system down of user program */
                            /*      type ==  0 : setup table error           */
                            /*      type == -1 : context error of ext_tsk    */
                            /*      type == -2 : context error of exd_tsk    */
                            /*      type == -3 : context error of ret_int    */
                            /*      type == -4 : context error of sys_clk    */
                            /*      type <= -5 : system reserve              */
ER          ercd;           /* error code of system down                    */
                            /*      type ==  0 : error code of setup table   */
                            /*      type == -1 : error code of ext_tsk       */
                            /*      type == -2 : error code of exd_tsk       */
                            /*      type == -3 : error code of ret_int       */
                            /*      type == -4 : error code of sys_clk       */
UW          inf;            /* information of system down                   */
                            /*      type == -1 : address of ext_tsk call     */
                            /*      type == -2 : address of exd_tsk call     */
                            /*      type == -3 : address of ret_int call     */
                            /*      type == -4 : address of sys_clk call     */
{
      set_cr(MD_SET | (SR_IMS15 << 4));      /* clear BL, set imask          */

      while(TRUE);          /* endless loop                                 */
}
```

図４－９　　C言語によるシステム異常終了処理ルーチンの記述例

４．９　C言語による例外処理ルーチンの記述方法

C言語による例外処理ルーチンの記述例を**図４－１０**に示します。

割込みハンドラと同様に、**#pragma interrupt**を用いて例外処理ルーチンの関数を宣言します。

割込み仕様として「トラップ命令リターン指定」を行ないます。「スタック切り換え指定」は行なわないでください。

「トラップ命令リターン指定」は、例外処理ルーチン終了時の復帰方法の指定であり、実行する**TRAPA**命令の番号を**"tn=<**トラップ番号**>"**で指定します。例外処理ルーチンは終了時に**ret_exc**システムコールを発行する必要がありますので、**"tn=57"**と指定してください。これにより、例外処理ルーチン終了時、**TRAPA#57**命令が実行されます。

例外処理ルーチン起動時は、例外がマスクされています。したがって、必要に応じて例外マスクを解除してください。例外マスクの解除は、**SH**シリーズ**C**コンパイラが提供する組込み関数**"void set_cr(int cr)"**を利用して行うことができます。組込み関数**"void set_cr(int cr)"**を使用する場合は、必ず**"#include <smachine.h>"**を宣言してください。

例外処理ルーチンは、例外発生前に実行していたタスクや割込みハンドラの一部として動作します。

```
#include    <smachine.h>
#include    "itron.h"

#pragma     interrupt    (exchdr(tn = 57))
INTHDR      exchdr(void)
{

/*
 *         例外マスク解除処理
 *         ........
 */

}
```

図４－１０　C言語による例外処理ルーチンの記述例

付録Ａ．コンソールドライバの例題

本章では、サンプルプログラムとして提供するSH7702,SH7708 MCU内蔵のシリアルコミュニケーションインタフェース(SCI:Serial Communication Interface、以下SCIと略す)を使用したコンソールドライバについて説明します。

他のSCIを使用する場合は、各SCIのハードウェア仕様を参照してください。

Ａ．１　コンソールドライバの概要

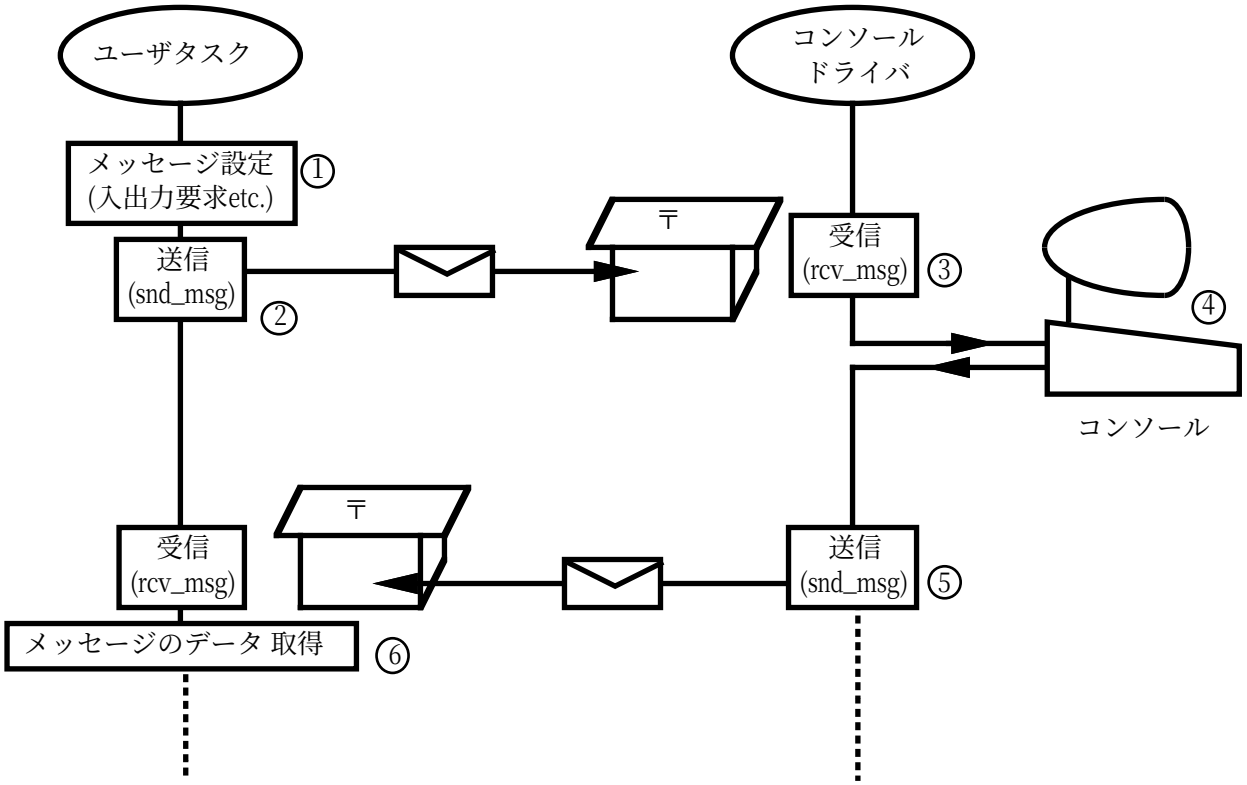
コンソールドライバは、ユーザプログラムからの入出力要求にしたがって、コンソールへの入出力処理を行なうプログラムです。**図Ａ－１**に本コンソールドライバの処理概要を示します。

ユーザタスクとコンソールドライバは、メッセージを用いて通信します。

ユーザタスクは、コンソールへの入出力要求メッセージをコンソールドライバ用のメイルボックスへ送信します（①，②）。入出力要求メッセージの送信後は、コンソールドライバからの処理結果メッセージの受信待ちになります。

コンソールドライバはメッセージを受信し、要求に対する入出力処理を行ないます（③，④）。入出力処理が完了したら、処理結果メッセージをユーザタスク用のメイルボックスへ送信します（⑤）。

ユーザタスクは処理結果メッセージを受信し、タスクの処理を続行します（⑥）。



図Ａ－１　コンソールドライバの処理概要

Ａ．２　コンソールドライバの機能

本コンソールドライバは、次の機能を持っています。

(1)キー入力データの**1**文字入力

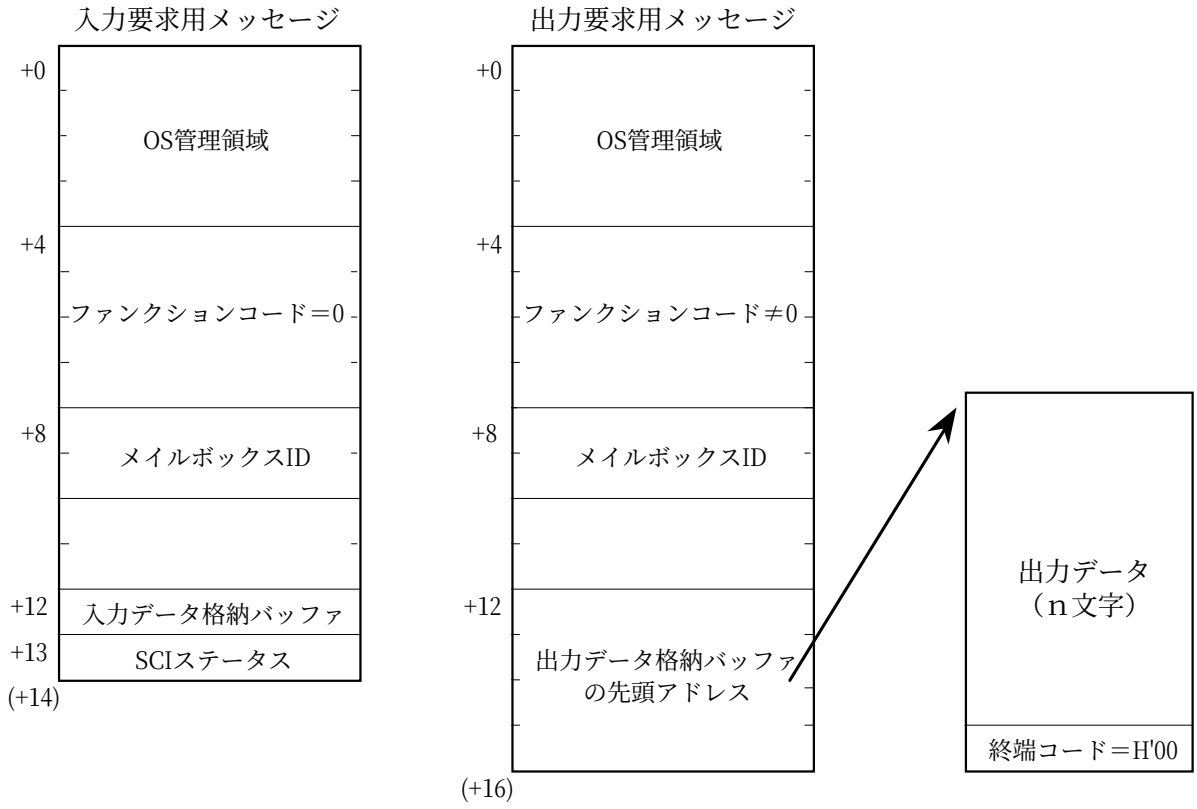
キー入力された文字を読み込みます。エコーバックは行ないません。

(2)コンソール画面への**n**文字出力

終端が**NULL**コード**(H'00)**の文字列をコンソール画面に出力します。

Ａ．３　メッセージフォーマット

コンソールドライバへの入出力要求とユーザタスクへの処理結果報告は、メッセージを用いて行ないます。**図Ａ－２**にメッセージのフォーマットを示します。



図Ａ－２　コンソールドライバ・ユーザタスク間のメッセージフォーマット

コンソールドライバのメッセージフォーマットは、**T_CIO**構造体として**C**言語用ヘッダファイル**(itron.h)**に定義されています。

T_CIO構造体の内容を**図Ａ－３**に示します。

```
/*-------------------- t_cio (for sample console driver) ---------------------------------*/
typedef struct  t_cio   {
        UW       msghead;               /* message head (OS management area)*/
        FN       fncd;                  /* function code (0:input, 1:output)     */
        ID       mbxid;                 /* mailbox id (for end information)      */
         union
        {
            struct
          {
             B  inbuf;                  /* input buffer                          */
             UB siossr;                 /* serial I/O status (SCI SSR)           */
           } ci;
            struct
          {
             B  *outbufp;               /* pointer to output buffer              */
           } co;
         } cio;
      } T_CIO;
```

図Ａ－３　T_CIO構造体の内容

Ａ．４　コンソールドライバのファイル構成

コンソールドライバの構成ファイルを以下に示します。本コンソールドライバは、C言語で記述されています。

(a)コンソールドライバ・**C**言語ヘッダファイル**(hicnsdrv.h)**

・コンソールドライバの動作環境データ

・**SCI**の初期化情報

・**I/O**アドレスの定義

(b)コンソールドライバ・**C**言語プログラムファイル**(hicnsdrv.c)**

・コンソールドライバを構成するタスク

・割込みハンドラ

Ａ．５　コンソールドライバの定義情報

Ａ．５．１　コンソールドライバの動作環境データ

図Ａ－４にコンソールドライバの動作環境データ**(hicnsdrv.h)**を示します。

割込みレベル**=11**、コンソールドライバタスクのタスク**ID=1**、コンソールドライバタスクの受信メールボックス**ID=1**を定義しています。

```
/*****************************************************************************/
/*                                                                           */
/*            console driver environment data                                */
/*                                                                           */
/*****************************************************************************/
#define CINTLVL    (UB)11          /* console interrupt level   = 11      */  ①
#define CTSKID     (ID)1           /* console driver task id    = 1       */  ②
#define CMBXID     (ID)1           /* console receive mailbox id = 1      */  ③
#define CCMDIN     (FN)0           /* console input command     = 0       */  ④
#define CCMDOUT    (FN)1           /* console output command    = 1       */  ⑤
```

図Ａ－４　コンソールドライバの動作環境データ

（説明）

①**CINTLVL**（コンソールからの割込みレベル）

②**CTSKID**（コンソールドライバタスクのタスク**ID**）

③**CMBXID**（コンソールドライバタスクの受信用メイルボックス**ID**）

④**CCMDIN**（入力要求の機能コード）

⑤**CCMDOUT**（出力要求の機能コード）

Ａ．５．２　ハードウェア初期化情報

図Ａ－５にハードウェアの初期化情報**(hicnsdrv.h)**を示します。

コンソールドライバは、**SCI**と、割込みコントローラの割込み優先レベル設定レジスタ**(IPR)**を使用します。なお、**BITRATE**設定データは**SH7708**用の値です。

```
/************************************************************************/
/*                                                                      */
/*            SCI,IPR environment data                                  */
/*                                                                      */
/************************************************************************/
#define MODE      (UB)0x00      /* SCR serial mode data              */  ①
                                /*    trans clock    : internal      */
                                /*    receive clock  : internal      */
#define FORMAT    (UB)0x00      /* SMR serial format data            */  ②
                                /*    communication  : async.        */
                                /*    bit        : 8                 */
                                /*    parity     : none              */
                                /*    stop       : 1                 */
                                /*    multi-process  : none          */
                                /*    clock      : 1 / 1             */
#define BITRATE   (UB)0x30      /* BRR serial bit rate data          */  ③
                                /*    clock = 60MHz / 4  : 9600bps    */

#define RIEENA    (UB)0x40      /* SCR RIE   enable  data            */  ④
#define RTXENA    (UB)0x30      /* SCR RE,TE  enable  data           */  ⑤
#define RXIENA    (UB)0x50      /* SCR RIE,RE enable  data           */  ④
#define RXIDIS    (UB)0xaf      /* SCR RIE,RE disable data           */
#define TXIENA    (UB)0x80      /* SCR TIE   enable  data            */  ⑥
#define TXIDIS    (UB)0x7f      /* SCR TIE   disable data            */

#define TDRETST   (UB)0x80      /* SSR TDRE  test  data              */  ⑦
#define TDRECLR   (UB)0x7f      /* SSR TDRE  clear data              */
#define RDRFCLR   (UB)0xbf      /* SSR RDRF  clear data              */
#define ERRCHK    (UB)0x38      /* SSR error check data              */  ⑧
#define ERRCLR    (UB)0xc7      /* SSR error clear data              */

#define IPRCLR    (UH)0xff0f    /* IPR bit 4-7 clear data            */  ⑨
```

図Ａ－５　ハードウェア初期化情報

（説明）

①**MODE**（シリアルコントロールレジスタ**(SCR)**に設定するモードデータ）

②**FORMAT**（シリアルモードレジスタ**(SMR)**に設定するビット長，パリティ，ストップビットなどのフォーマットデータ）

③**BITRATE**（ビットレートレジスタ**(BRR)**に設定する転送速度データ）

④**RXIENA,RIEENA,RXIDIS**（**SCR**に設定する受信の有効／無効フラグデータ）

⑤**RTXENA**（**SCR**に設定する送受信の有効フラグデータ）

⑥**TXIENA,TXIDIS**（**SCR**に設定する送信の有効／無効フラグデータ）

⑦**TDRETST,TDRECLR,RDRFCLR**（シリアルステータスレジスタ**(SSR)**に設定する、送信用テストフラグデータと送受信用クリアフラグデータ）

⑧**ERRCHK,ERRCLR**（**SSR**に設定するエラーチェックおよびエラークリア用データ）

⑨**IPRCLR**（割込み優先レベル設定レジスタ**(IPR)**に設定する**SCI**用割込みレベルのクリア用データ）

Ａ．５．３　ハードウェアI/Oアドレス

図Ａ－６に、コンソールドライバが使用する**I/O**レジスタのアドレス定義**(hicnsdrv.h)**を示します。

```
/*****************************************************************************/
/*                                                                           */
/*            SCI,IPR I/O address                                            */
/*                                                                           */
/*****************************************************************************/
#define IOBASE  0xfffffe80              /* I/O base address = 0xfffffe80    */  ①
#define SMR  (0xfffffe80 - IOBASE)      /* SCI SMR    address offset        */  ②
#define BRR  (0xfffffe82 - IOBASE)      /* SCI BRR    address offset        */  ③
#define SCR  (0xfffffe84 - IOBASE)      /* SCI SCR    address offset        */  ④
#define TDR  (0xfffffe86 - IOBASE)      /* SCI TDR    address offset        */  ⑤
#define SSR  (0xfffffe88 - IOBASE)      /* SCI SSR    address offset        */  ⑥
#define RDR  (0xfffffe8a - IOBASE)      /* SCI RDR    address offset        */  ⑦
#define IPR  (0xffffffee4 - IOBASE)     /* IPR(IPRB:SCI) address offset     */  ⑧
```

図Ａ－６　ハードウェアI/Oアドレス

（説明）

①**IOBASE**（**GBR**に設定する**I/O**のベースアドレス）

②**SMR**（**SCI**のシリアルモードレジスタ**(SMR)**の**I/O**のベースアドレスからのオフセット）

③**BRR**（**SCI**のビットレートレジスタ**(BRR)**の**I/O**のベースアドレスからのオフセット）

④**SCR**（**SCI**のシリアルコントロールレジスタ**(SCR)**の**I/O**のベースアドレスからのオフセット）

⑤**TDR**（**SCI**のトランスミットデータレジスタ**(TDR)**の**I/O**のベースアドレスからのオフセット）

⑥**SSR**（**SCI**のシリアルステータスレジスタ**(SSR)**の**I/O**のベースアドレスからのオフセット）

⑦**RDR**（**SCI**のレシーブデータレジスタ**(RDR)**の**I/O**のベースアドレスからのオフセット）

⑧**IPR**（割込み優先レベル設定レジスタ**(IPR)**の**I/O**のベースアドレスからのオフセット）

Ａ．６　コンソールドライバのプログラム内容

Ａ．６．１　モジュール構成

本コンソールドライバは、**表Ａ－１**に示すモジュールで構成されます。

表Ａ－１　コンソールドライバのモジュール構成

項番	モジュール名	モジュールの処理内容	プログラム種別
1	**hi_cnsdrv**	コンソールドライバメイン処理 　ユーザタスクからの入出力要求メッセージを受信して入出力を行ない、処理結果メッセージを返す	タスク
2	**hi_cnsini**	コンソールドライバ初期化処理 　割込みの設定や**SCI**の初期化を行なう	
3	**hi_cnsinp**	コンソールドライバ入力処理 　ユーザタスクからの入力要求を処理する	
4	**hi_cnsout**	コンソールドライバ出力処理 　ユーザタスクからの出力要求を処理する	
5	**hi_cnshdrrx**	**SCI**受信割込みハンドラ 　**SCI**からの受信可能割込みによって起動される割込みハンドラ	割込みハンドラ
6	**hi_cnshdrtx**	**SCI**送信割込みハンドラ 　**SCI**からの送信完了割込みによって起動される割込みハンドラ	
7	**hi_cnshdrer**	**SCI**エラー割込みハンドラ 　**SCI**からのエラー発生割込みによって起動される割込みハンドラ	

Ａ．６．２　共通メモリ領域

図Ａ－７に共通メモリ領域のプログラム内容**(hicnsdrv.c)**を示します。
割込みハンドラ用スタックエリア**(hi_intstk11, p_intstk11)**、コンソールドライバタスクと割込みハンドラ間の共通エリア**(ssrsave,rdrsave)**を確保します。

```
/************************************************************************/
/*                                                                      */
/*              stack area of interrupt handler                         */
/*                                                                      */
/************************************************************************/
extern  VW    hi_intstk11[ ];           /* stack area of interrupt handler      */   ①
                                        /* end address of stack area            */
                                        /*       (interrupt level = 11)         */
static const VP p_intstk11 = (VP)&hi_intstk11[(hi_intstksz11) / sizeof(VW)];

/************************************************************************/
/*                                                                      */
/*                save area                                             */
/*                                                                      */
/************************************************************************/
static  UB    ssrsave;                  /* save area of SSR data                */
static  UB    rdrsave;                  /* save area of RDR data                */
```

図Ａ－７　共通メモリ領域

（説明）
①この配列は、割込みハンドラ用スタック領域定義プログラム**(hiintstk.c)**で確保されています。

Ａ．６．３　コンソールドライバタスクのメイン処理(hi_cnsdrv)

図Ａ－８にコンソールドライバタスクのメイン処理のプログラム内容**(hicnsdrv.c)**を示します。

メイン処理はユーザタスクからの入出力要求メッセージを受信し、必要な処理を呼び出すプログラムです。処理が終了したら、処理結果メッセージを要求タスクに送信します。

```
/*****************************************************************************/
/*SPECIFICATIONS ;                                                           */
/*  NAME     = hi_cnsdrv ;                                                   */
/*  DATE     = 95/02/20 ;                                                    */
/*  AUTHOR   = Hitachi, Ltd. ;                                               */
/*  FUNCTION = HI-SH77 console driver task ;                                 */
/*  ATTRIBUTE = PUBLIC ;                                                     */
/*  HISTORY  = V1.0 ;                                                        */
/*END OF SPECIFICATIONS ;                                                    */
/*****************************************************************************/
TASK       hi_cnsdrv(void)
{
  void   hi_cnsini(void);              /* console initialize routine       */
  ER     hi_cnsinp(T_CIO *);           /* console input routine            */
  ER     hi_cnsout(T_CIO *);           /* console output routine           */

  ER     ercd;                         /* error code                       */
  T_CIO  *pk_cio;                      /* pointer to message packet        */

  set_gbr((VP)IOBASE);                 /* set I/O base address to GBR      */   ①

  hi_cnsini();                         /* call console initialize routine  */   ②

  while(TRUE)                          /* endless loop                     */
  {
                                       /* receive message from user task   */
    ercd = rcv_msg((void **)&pk_cio, CMBXID);                                   ③
    if(ercd < E_OK)                    /* if rcv_msg fail                  */
      break;                           /* break endless loop               */

    if(pk_cio->fncd == CCMDIN)         /* if function code is input        */
      ercd = hi_cnsinp(pk_cio);        /* call console input routine       */   ④
    else                               /* if function code is output       */
      ercd = hi_cnsout(pk_cio);        /* call console output routine      */   ⑤
    if(ercd < E_OK)                    /* if input/output routine fail     */
      break;                           /* break endless loop               */

                                       /* send message to user task        */
    ercd = snd_msg(pk_cio->mbxid, (void *)pk_cio);                              ⑥
    if(ercd < E_OK)                    /* if snd_msg fail                  */
      break;                           /* break endless loop               */
  }

  ext_tsk();                           /* exit console driver task         */   ⑦
}
```

図Ａ－８　コンソールドライバタスクのメイン処理(hi_cnsdrv)

（説明）

①グローバルベースレジスタ(GBR)にI/Oベースアドレス(IOBASE)を設定します(set_gbr)。

②コンソールの初期化処理を呼び出します(hi_cnsini)。

③ユーザタスクからの入出力要求メッセージを受信します(rcv_msg)。

④入力要求の場合、入力処理を呼び出します(hi_cnsinp)。

⑤出力要求の場合、出力処理を呼び出します(hi_cnsout)。

⑥処理結果メッセージを入出力要求タスクへ送信します(snd_msg)。

⑦エラーが発生した場合、コンソールドライバタスクを終了します(ext_tsk)。

Ａ．６．４　コンソールの初期化処理(hi_cnsini)

図Ａ－９にコンソールの初期化処理のプログラム内容(hicnsdrv.c)を示します。

初期化処理はメイン処理(hi_cnvdrv)から呼ばれ、SCIの各I/Oレジスタ(SCR,SMR,BRR)の初期化、割込み優先レベルレジスタ(IPR)の設定を行ないます。IPRの設定中に割込みが入らないようにするため、割込みマスクレベルを15に設定します。IPRの設定後、割込みマスクレベルを0に戻します。

```
/*****************************************************************************/
/*SPECIFICATIONS ;                                                           */
/*  NAME     = hi_cnsini ;                                                   */
/*  DATE     = 95/02/20 ;                                                    */
/*  AUTHOR   = Hitachi, Ltd. ;                                               */
/*  FUNCTION  = HI-SH77 console initialize routine ;                         */
/*  ATTRIBUTE = PUBLIC ;                                                     */
/*  HISTORY   = V1.0 ;                                                       */
/*END OF SPECIFICATIONS ;                                                    */
/*****************************************************************************/
void       hi_cnsini(void)
{
   UH    ipr;                          /* variable of IPR                  */

   gbr_write_byte(SCR, MODE);          /* set serial mode                  */   ①
   gbr_write_byte(SMR, FORMAT);        /* set serial format                */
   gbr_write_byte(BRR, BITRATE);       /* set serial bit rate              */
   gbr_or_byte(SCR, RIEENA);           /* enable RIE                       */   ②

   chg_ims(SR_IMS15);                  /* disable interrupt                */   ③
   ipr = gbr_read_word(IPR) & IPRCLR;  /* clear IPR (SCI) to ipr           */   ④
                                       /* set CINTLVL to IPR (SCI)         */
   gbr_write_word(IPR, ipr | (CINTLVL << 4));
   ichg_ims(SR_IMS00);                 /* enable interrupt                 */   ⑤

   gbr_or_byte(SCR, RTXENA);           /* enable RX,TX                     */   ⑥
}
```

図Ａ－９　コンソールの初期化処理(hi_cnsini)

（説明）

①シリアルコントロールレジスタ(SCR)、シリアルモードレジスタ(SMR)、ビットレートレジスタ(BRR)に初期値(MODE, FORMAT, BITRATE)を設定します。

②SCRの受信割込みフラグを有効(RIEENA)に設定します。

③割込み優先レベル設定レジスタ(IPR)を変更するため、割込みレベルを15に設定します(chg_ims)。

④IPR内のSCI用の割込み優先レベルのみ初期値(CINTLVL)に設定します。

⑤割込みマスクを0に戻します(ichg_ims)。

⑥送受信を可能にします。

Ａ．６．５　コンソールからの入力処理(hi_cnsinp)

図Ａ－１０にコンソールからの入力処理のプログラム内容**(hicnsdrv.c)**を示します。

入力処理はメイン処理**(hi_cnvdrv)**から呼ばれ、**1**文字入力を行ないます。入力処理は**slp_tsk**システムコールを用いて、キー入力を待ちます。キー入力が行なわれると**SCI**受信割込みハンドラ**(hi_cnshdrrx)**が起動し、受信データを共通メモリ領域に書き込んで入力処理を起床します。入力処理は共通メモリ領域のデータを読み込んで、処理結果メッセージに格納し、メイン処理に制御を渡します。

```
/************************************************************************/
/*SPECIFICATIONS ;                                                      */
/*  NAME     = hi_cnsinp ;                                             */
/*  DATE     = 95/02/20 ;                                              */
/*  AUTHOR   = Hitachi, Ltd. ;                                         */
/*  FUNCTION  = HI-SH77 console input routine ;                        */
/*  ATTRIBUTE = PUBLIC ;                                               */
/*  HISTORY   = V1.0 ;                                                 */
/*END OF SPECIFICATIONS ;                                             */
/************************************************************************/
ER        hi_cnsinp(pk_cio)
T_CIO     *pk_cio;                  /* pointer to message packet      */
{
  ER     ercd;                      /* error code                     */

  ercd = slp_tsk();                  /* wait receive process          */   ①
  if(ercd < E_OK)                   /* if slp_tsk fail                */
    goto exit;                      /* go to exit                     */

  if(!(ssrsave & ERRCHK))           /* if receive process  success    */   ②
    pk_cio->cio.ci.inbuf = rdrsave; /* set receive data to input buffer */
  else                              /* if receive process fail        */
    pk_cio->cio.ci.inbuf = (UB)'\0'; /* set '\0' to input buffer      */
  pk_cio->cio.ci.siossr = ssrsave;  /* set serial I/O status to siossr */   ③

exit:
  return(ercd);                     /* return ercd                    */
}
```

図Ａ－１０　コンソールからの入力処理(hi_cnsinp)

（説明）

①**slp_tsk**システムコールを発行し、**SCI**受信割込みハンドラ**(hi_cnshdrrx)**による起床を待ちます。

②キー入力データ受信時のシリアルステータスレジスタ**(SSR)**にエラーが発生しているかを調べます。エラーが発生していなければ受信データを、エラーが発生していれば**H'00**を処理結果メッセージに格納します。

③キー入力データ受信時の**SSR**の内容を処理結果メッセージに格納します。

Ａ．６．６　コンソールへの出力処理(hi_cnsout)

図Ａ－１１にコンソールへの出力処理のプログラム内容**(hicnsdrv.c)**を示します。

出力処理はメイン処理**(hi_cnvdrv)**から呼ばれ、**n**文字出力を行ないます。出力処理の間、受信割込みを無効にします。指定された文字列の終端**(H'00**コード**)**まで**SCI**が送信可能であれば、**1**文字づつ出力を行ないます。トランスミットデータレジスタ**(TDR)**に**1**文字設定したら**slp_tsk**システムコールを発行し、コンソールの出力完了を待ちます。出力が完了すると、**SCI**送信割込みハンドラ**(hi_cnshdrtx)**が起動して出力処理を起床し、次の文字の出力を行ないます。**n**文字の出力が終了したら受信割込みを有効にし、メイン処理に制御を渡します。

```
/**************************************************************************/
/*SPECIFICATIONS ;                                                        */
/*  NAME     = hi_cnsout ;                                                */
/*  DATE     = 95/02/20 ;                                                 */
/*  AUTHOR   = Hitachi, Ltd. ;                                            */
/*  FUNCTION  = HI-SH77 console output routine ;                          */
/*  ATTRIBUTE = PUBLIC ;                                                  */
/*  HISTORY   = V1.0 ;                                                    */
/*END OF SPECIFICATIONS ;                                                 */
/**************************************************************************/
ER      hi_cnsout(pk_cio)
T_CIO   *pk_cio;                    /* pointer to message packet        */
{
    ER      ercd;                   /* error code                       */
    B       *p;                     /* pointer to charcter data         */

    ercd = E_OK;                    /* set E_OK to ercd                 */

    p = pk_cio->cio.co.outbufp;     /* set output buffer address        */

    gbr_and_byte(SCR, RXIDIS);      /* disable RXI                      */   ①

    while(TRUE)                     /* endless loop                     */
    {
      if(gbr_read_byte(SSR) & TDRETST)/* if SSR TDRE is set             */   ②
      {
        if(*p == '\0')              /* if charcter data is terminater   */   ③
          break;                    /* break endless loop               */
        gbr_write_byte(TDR, (UB)*p); /* write charcter data to TDR      */   ④
        gbr_and_byte(SSR, TDRECLR); /* clear SSR TDRE                   */   ⑤
        p++;                        /* next charcter                    */
      }

      gbr_or_byte(SCR, TXIENA);     /* enable TXI                       */   ⑥

      ercd = slp_tsk();             /* wait trans process               */   ⑦
      if(ercd < E_OK)               /* if slp_tsk fail                  */
        break;                      /* break endless loop               */
    }

    gbr_or_byte(SCR, RXIENA);       /* enable RXI                       */   ⑧

    return(ercd);                   /* return ercd                      */
}
```

図Ａ－１１　　コンソールへの出力処理(hi_cnsout)

（説明）

①コンソールへの出力処理中は、シリアルコントロールレジスタ(SCR)の受信割込みを無効に設定します。

②シリアルステータスレジスタ(SSR)を読み込んで、送信可能かを調べます。

③出力文字列の終端(H'00)であれば、出力処理を終了します。

④出力データをトランスミットデータレジスタ(TDR)に設定します。

⑤SSRの送信可能フラグをクリアします。

⑥SCRの送信完了割込みを有効に設定します。

⑦slp_tskシステムコールを発行し、SCI送信完了割込みハンドラ(hi_cnshdrtx)からの起床を待ちます。

⑧コンソールへの出力処理が終了したら、SCRの受信割込みを有効に戻します。

Ａ．６．７　SCI受信割込みハンドラ(hi_cnshdrrx)

図Ａ－１２にSCI受信割込みハンドラのプログラム内容(hicnsdrv.c)を示します。

受信割込みハンドラは、SCIが受信可能になったとき起動される割込みハンドラです。割込みハンドラ先頭で割込みマスクレベルを設定し、例外マスクを解除します。割込み発生時のシリアルステータスレジスタ(SSR)とレシーブデータレジスタ(RDR)を共通メモリ領域に格納し、iwup_tskシステムコールを発行してコンソールドライバ入力処理(hi_cnsinp)を起床します。割込みハンドラ終了時は #pragma interrupt 文の宣言により、ret_intシステムコール(TRAPA#61)を発行します。

```
/************************************************************************/
/*SPECIFICATIONS ;                                                      */
/*  NAME     = hi_cnshdrrx ;                                           */
/*  DATE     = 95/02/20 ;                                              */
/*  AUTHOR   = Hitachi, Ltd. ;                                         */
/*  FUNCTION = HI-SH77 console interrupt handler (receive process) ;   */
/*  ATTRIBUTE = PUBLIC ;                                               */
/*  HISTORY  = V1.0 ;                                                  */
/*END OF SPECIFICATIONS ;                                             */
/************************************************************************/
#pragma interrupt   (hi_cnshdrrx(sp = p_intstk11, tn = 61))                    ①

INTHDR  hi_cnshdrrx(void)
{
  VP    gbrsave;                    /* save variable of GBR             */

  set_cr(MD_SET | (CINTLVL << 4));  /* clear BL, set imask              */   ②

  gbrsave = get_gbr();              /* save GBR to gbrsave              */   ③
  set_gbr((VP)IOBASE);              /* set I/O base address to GBR      */   ④

  ssrsave = gbr_read_byte(SSR);     /* set SSR data to ssrsave          */   ⑤
  rdrsave = gbr_read_byte(RDR);     /* set RDR data to rdrsave          */   ⑥
  gbr_and_byte(SSR, RDRFCLR);       /* clear SSR RDRF                   */   ⑦
  iwup_tsk(CTSKID);                 /* wake up console driver task      */   ⑧

  set_gbr(gbrsave);                 /* load GBR from gbrsave            */   ⑨
}
```

図Ａ－１２　　SCI受信割込みハンドラ(hi_cnshdrrx)

（説明）

①**#pragma interrupt**文により、割込みハンドラ用スタックと、割込みハンドラ終了時の**ret_int**システムコール発行**(TRAPA#61)**を宣言します。

②割込みマスクレベルを設定し、例外マスクを解除します**(set_cr)**。

③グローバルベースレジスタ**(GBR)**の値を保存します**(get_gbr)**。

④**GBR**に**I/O**ベースアドレス**(IOBASE)**を設定します**(set_gbr)**。

⑤シリアルステータスレジスタ**(SSR)**の内容を共通メモリ領域に格納します。

⑥レシーブデータレジスタ**(RDR)**の内容を共通メモリ領域に格納します。

⑦**SSR**の受信データフラグをクリアします。

⑧**iwup_tsk**システムコールを発行し、コンソールドライバ入力処理を起床します。

⑨**GBR**の値を元に戻します**(set_gbr)**。

Ａ．６．８　SCI送信割込みハンドラ(hi_cnshdrtx)

図Ａ－１３に**SCI**送信割込みハンドラのプログラム内容**(hicnsdrv.c)**を示します。

送信割込みハンドラは、**SCI**が送信完了したとき起動される割込みハンドラです。割込みハンドラ先頭で割込みマスクレベルを設定し、例外マスクを解除します。**iwup_tsk**システムコールを発行して送信完了待ちのコンソールドライバ出力処理(**hi_cnsout**)を起床します。割込みハンドラ終了時は **#pragma interrupt**文の宣言により、**ret_int**システムコール**(TRAPA#61)**を発行します。

```
/************************************************************************/
/*SPECIFICATIONS ;                                                      */
/*  NAME    = hi_cnshdrtx ;                                             */
/*  DATE    = 95/02/20 ;                                                */
/*  AUTHOR  = Hitachi, Ltd. ;                                           */
/*  FUNCTION = HI-SH77 console interrupt handler (trans process) ;      */
/*  ATTRIBUTE = PUBLIC ;                                                */
/*  HISTORY  = V1.0 ;                                                   */
/*END OF SPECIFICATIONS ;                                               */
/************************************************************************/
#pragma interrupt  (hi_cnshdrtx(sp = p_intstk11, tn = 61))                    ①

INTHDR  hi_cnshdrtx(void)
{
  VP    gbrsave;                  /* save variable of GBR            */

  set_cr(MD_SET | (CINTLVL << 4)); /* clear BL, set imask             */      ②

  gbrsave = get_gbr();            /* save GBR to gbrsave             */      ③
  set_gbr((VP)IOBASE);            /* set I/O base address to GBR     */      ④

  gbr_and_byte(SCR, TXIDIS);      /* disable TXI                     */      ⑤
  iwup_tsk(CTSKID);               /* wake up console driver task     */      ⑥

  set_gbr(gbrsave);               /* load GBR from gbrsave           */      ⑦
}
```

図Ａ－１３　　SCI送信割込みハンドラ(hi_cnshdrtx)

（説明）

①**#pragma interrupt**文により、割込みハンドラ用スタックと、割込みハンドラ終了時の**ret_int**システムコール発行**(TRAPA#61)**を宣言します。

②割込みマスクレベルを設定し、例外マスクを解除します**(set_cr)**。

③グローバルベースレジスタ**(GBR)**の値を保存します**(get_gbr)**。

④**GBR**に**I/O**ベースアドレス**(IOBASE)**を設定します**(set_gbr)**。

⑤シリアルコントロールレジスタ**(SCR)**の送信割込みを無効に設定します。

⑥**iwup_tsk**システムコールを発行し、コンソールドライバ出力処理を起床します。

⑦**GBR**の値を元に戻します**(set_gbr)**。

Ａ．６．９　SCIエラー割込みハンドラ(hi_cnshdrer)

図Ａ－１４に**SCI**エラー割込みハンドラのプログラム内容**(hicnsdrv.c)**を示します。

エラー割込みハンドラは、キー入力データ受信のエラー発生時に起動される割込みハンドラです。割込みハンドラ先頭で割込みマスクレベルを設定し、例外マスクを解除します。割込み発生時のシリアルステータスレジスタ(**SSR**)を共通メモリ領域に格納し、iwup_tskシステムコールを発行してコンソールドライバ入力処理(**hi_cnsinp**)を起床します。割込みハンドラ終了時は #pragma interrupt文の宣言により、ret_intシステムコール(TRAPA#61)を発行します。

```
/************************************************************************/
/*SPECIFICATIONS ;                                                      */
/*  NAME    = hi_cnshdrer ;                                             */
/*  DATE    = 93/02/20 ;                                                */
/*  AUTHOR  = Hitachi, Ltd. ;                                           */
/*  FUNCTION = HI-SH77 console interrupt handler (receive process fail) ; */
/*  ATTRIBUTE = PUBLIC ;                                                */
/*  HISTORY  = V1.0 ;                                                   */
/*END OF SPECIFICATIONS ;                                               */
/************************************************************************/
#pragma interrupt  (hi_cnshdrer(sp = p_intstk11, tn = 61))                    ①

INTHDR  hi_cnshdrer(void)
{
  VP    gbrsave;                  /* save variable of GBR              */

  set_cr(MD_SET | (CINTLVL << 4)); /* clear BL, set imask               */   ②

  gbrsave = get_gbr();            /* save GBR to gbrsave               */   ③
  set_gbr((VP)IOBASE);            /* set I/O base address to GBR       */   ④

  ssrsave = gbr_read_byte(SSR);   /* set SSR data to ssrsave           */   ⑤
  gbr_and_byte(SSR, ERRCLR);      /* clear SSR error bit               */   ⑥
  iwup_tsk(CTSKID);               /* wake up console driver task       */   ⑦

  set_gbr(gbrsave);               /* load GBR from gbrsave             */   ⑧
}
```

図Ａ－１４　　SCIエラー割込みハンドラ(hi_cnshdrer)

（説明）

①**#pragma interrupt**文により、割込みハンドラ用スタックと、割込みハンドラ終了時の**ret_int**システムコール発行**(TRAPA#61)**を宣言します。

②割込みマスクレベルを設定し、例外マスクを解除します**(set_cr)**。

③グローバルベースレジスタ**(GBR)**の値を保存します**(get_gbr)**。

④**GBR**に**I/O**ベースアドレス**(IOBASE)**を設定します**(set_gbr)**。

⑤シリアルステータスレジスタ**(SSR)**の内容を共通メモリ領域に格納します。

⑥**SSR**のエラーフラグをクリアします。

⑦**iwup_tsk**システムコールを発行し、コンソールドライバ入力処理を起床します。

⑧**GBR**の値を元に戻します**(set_gbr)**。

付録B．タイマドライバの例題

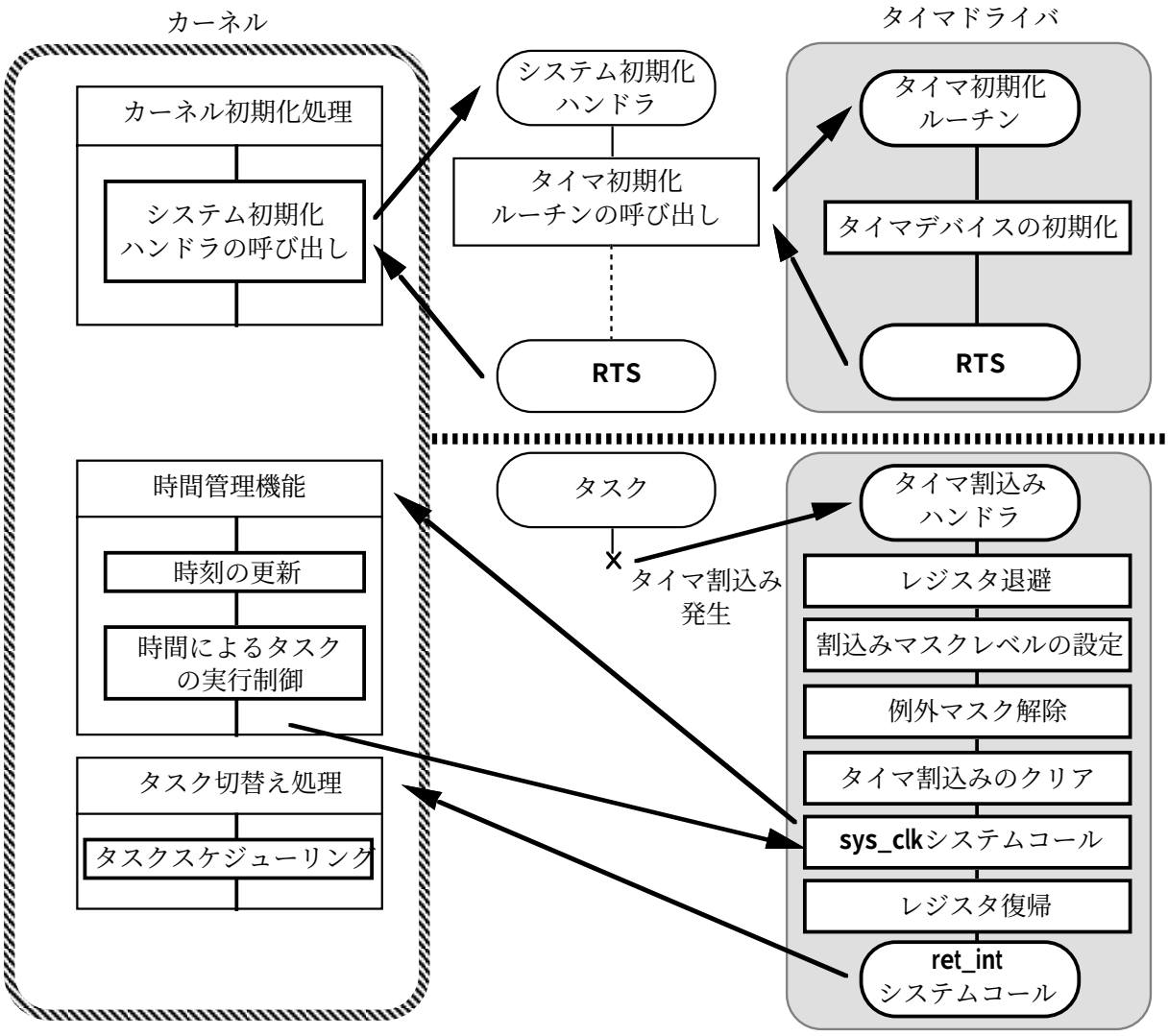
本章では、サンプルプログラムとして提供するSH7702,SH7708 MCU内蔵の32ビットタイマユニット（TMU:Timer Unit、以下TMUと略す）を使用したタイマドライバについて説明します。本タイマドライバはTMUのチャネル0を使用しています。

他のハードウェアタイマを使用する場合は、各タイマのハードウェア仕様を参照してください。

B．1　タイマドライバの処理概要

タイマドライバは、タイマ初期化ルーチンとタイマ割込みハンドラから構成されます。

図B－1にタイマドライバの処理概要を示します。



図B－1　タイマドライバの処理概要

(1)タイマ初期化ルーチン

ハードウェアタイマに使用するタイマデバイスの初期化を行ないます。タイマ初期化ルーチンはシステム初期化ハンドラから呼び出すようにします。

(2)タイマ割込みハンドラ

タイマ割込みハンドラは、ハードウェアタイマからの割込み発生により起動するようにします。タイマ割込みハンドラでは、割込みマスクレベルの設定、例外マスクの解除、タイマ割込みのクリアを行ない、**sys_clk**システムコールによりカーネルに対して時間管理処理要求を行ないます。そして、**ret_int**システムコールにより終了します。

Ｂ．２　タイマドライバのファイル構成

タイマドライバのファイル構成を以下に示します。本タイマドライバは、**C**言語で記述されています。

(a)タイマドライバ・ヘッダファイル**(hitmrdrv.h)**

・タイマドライバの動作環境データ

・**TMU**の初期化情報

・割込み優先レベル設定レジスタ**(IPR)**の初期化情報

・**I/O**アドレスの定義

(b)タイマドライバ・ソースファイル**(hitmrdrv.c)**

・タイマ初期化ルーチン

・タイマ割込みハンドラ

Ｂ．３　タイマドライバの定義情報

Ｂ．３．１　タイマドライバの動作環境データ

図Ｂ－２にタイマドライバの動作環境データ**(hitmrdrv.h)**を示します。
インタバル時間**=10msec**、割込みレベル**=13**を定義しています。

```
/*****************************************************************************/
/*                                                                           */
/*            timer driver environment data                                  */
/*                                                                           */
/*****************************************************************************/
#define TINTVAL (UB)10              /* timer interval time = 10ms            */   ①
#define TINTLVL (UB)13              /* timer interrupt level = 13            */   ②
```

図Ｂ－２　タイマドライバの動作環境データ

（説明）

①**TINTVAL**（タイマ割込みのインタバル時間；単位**msec**）

②**TINTLVL**（ハードウェアタイマからの割込みレベル）

Ｂ．３．２　ハードウェア初期化情報

図Ｂ－３にハードウェアの初期化情報**(hitmrdrv.h)**を示します。

タイマドライバは、**TMU**と、割込みコントローラの割込み優先レベル設定レジスタ**(IPR)**を使用します。なお、本タイマドライバは、**Pφ**（周辺モジュールクロック）**=15MHz**で初期化しています。

```
/*************************************************************************/
/*                                                                       */
/*              TMU,IPR environment data                                 */
/*                                                                       */
/*************************************************************************/
#define TCNTSTR   (UB)0x01        /* TSTR TCNT start data (ch0)        */   ①
#define TCNTSTP   (UB)0xfe        /* TSTR TCNT stop  data (ch0)        */   ②
#define TCRDAT    (UH)0x0020      /* TCR  initialize data              */   ③
                                  /*    TCNT reload  : TCNT = 0        */
                                  /*    clock       : internal         */
                                  /*    pre-scaler  : 1 /4 (267ns)     */
#define TCORDAT   (UW)37499       /* TCORDAT = TINTVAL*15MHz/4 - 1     */   ④

#define IPRCLR    (UH)0x0fff      /* IPR  bit 12-15 clear data         */   ⑤
```

図Ｂ－３　ハードウェアの初期化情報

（説明）

①**TCNTSTR**（タイマスタートレジスタ**(TSTR)**に設定するタイマ動作許可値）

②**TCNTSTP**（**TSTR**に設定するタイマ動作停止値）

③**TCRDAT**（タイマコントロールレジスタ**(TCR)**に設定するカウンタクロック，プリスケーラ値）

④**TCORDAT**（タイマコントロールレジスタ**(TCOR)**に設定するリロード値）

⑤**IPRCLR**（割込み優先レベルレジスタ**(IPR)**に設定するクリア値）

【参考】ハードウェアタイマ周期時間の設定

ハードウェアタイマ周期時間(Ｔ)は、カウンタクロック周期時間(ｔ)とカウンタ値(ｎ)で決定します。

Ｔ ＝ ｔ × （ｎ ＋ １）

ｔは、タイマコントロールレジスタ(TCR)でカウンタクロックの選択（Pφ／４，Pφ／１６，Pφ／６４，Pφ／２５６）を行なうことで決定します。

Pφ（周辺モジュールクロック）が15MHzの場合、ｔは以下の数値になります。

カウンタクロック＝Pφ／４　：　ｔ＝66.7nsec

カウンタクロック＝Pφ／１６　：　ｔ＝267nsec

カウンタクロック＝Pφ／６４　：　ｔ＝1.07μsec

カウンタクロック＝Pφ／２５６　：　ｔ＝4.27μsec

ｎは、タイマコンスタントレジスタ(TCOR)に0x00000000～0xffffffffの値を設定することで決定します。

したがって、Pφが15MHzの場合、Ｔは以下の範囲になります。

カウンタクロック＝Pφ／４　：　Ｔ＝66.7nsec　～　286sec

カウンタクロック＝Pφ／１６　：　Ｔ＝267nsec　～　1145sec

カウンタクロック＝Pφ／６４　：　Ｔ＝1.07μsec　～　4581sec

カウンタクロック＝Pφ／２５６　：　Ｔ＝4.27μsec　～　18325sec

本サンプルプログラムでは次の設定を行ない、ハードウェアタイマ周期時間(T)を10msecにしています。

```
#define TCRDAT      (UH)0x0020      /* TCR initialize data                    */   ①
                                    /*    TCNT reload : TCNT = 0              */
                                    /*    clock       : internal              */
                                    /*    pre-scaler  : 1 / 4 (267ns)         */
#define TCORDAT     (UW)37499       /* TCORDAT = TINTVAL * 15MHz / 4 - 1      */   ②
#define TINTVAL     (UB)10          /* timer interval time = 10ms             */   ③
```

（説明）

①カウンタクロックにPφ／４を選択します。

②TCORには37499が入ります。

③タイマ周期時間(T)を10msecにしています。

（導き方）

TCOR = タイマ周期時間(s) × n - 1

上式より、

タイマ周期時間を10msecとするので、タイマ周期時間＝10×10^{-3}

クロックセレクトはPφ／４を選択したので、$n = 15 \times 10^{6} / 4$

$\therefore TCOR = 10 \times 10^{-3} \times 15 \times 10^{6} / 4 - 1$

$= 37499$

Ｂ．３．３　ハードウェアI/Oアドレス

図Ｂ－４にタイマドライバが使用する**I/O**レジスタのアドレス定義**(hitmrdrv.h)**を示します。

```
/************************************************************************/
/*                                                                      */
/*              TMU,IPR I/O address                                     */
/*                                                                      */
/************************************************************************/
#define IOBASE  0xfffffe80              /* I/O base address = 0xfffffe80        */  ①
#define TSTR  (0xfffffe92 - IOBASE) /* TMU TSTR          address offset      */  ②
#define TCOR  (0xfffffe94 - IOBASE) /* TMU TCOR(ch0)  address offset      */  ③
#define TCNT  (0xfffffe98 - IOBASE) /* TMU TCNT(ch0)   address offset      */  ④
#define TCR   (0xfffffe9c - IOBASE) /* TMU TCR(ch0)    address offset      */  ⑤
#define IPR   (0xfffffee2 - IOBASE) /* IPR(IPRA:TMU-ch0) address offset  */  ⑥
```

図Ｂ－４　ハードウェアI/Oアドレス

（説明）

①**IOBASE**（**GBR**に設定する**I/O**ベースアドレス）

②**TSTR**（タイマスタートレジスタ**(TSTR)**の**I/O**ベースアドレスからのオフセット）

③**TCOR**（タイマコンスタントレジスタ**(TCOR)**の**I/O**ベースアドレスからのオフセット）

④**TCNT**（タイマカウンタ**(TCNT)**の**I/O**ベースアドレスからのオフセット）

⑤**TCR**（タイマコントロールレジスタ**(TCR)**の**I/O**ベースアドレスからのオフセット）

⑥**IPR**（割込み優先レベルレジスタ**(IPR)**の**I/O**ベースアドレスからのオフセット）

Ｂ．４　タイマドライバのプログラム内容

Ｂ．４．１　タイマ割込みハンドラ用スタック領域

図Ｂ－５にタイマ割込みハンドラが使用するスタック領域**(hitmrdrv.h)**の定義内容を示します。

```
/*************************************************************************/
/*                                                                       */
/*              stack area of interrupt handler                          */
/*                                                                       */
/*************************************************************************/
extern  VW    hi_intstk13[ ];           /* stack area of interrupt handler      */   ①
                                        /* end address of stack area            */
                                        /*       (interrupt level = 13)         */
static const VP p_intstk13 = (VP)&hi_intstk13[(hi_intstksz13) / sizeof(VW)];
```

図Ｂ－５　タイマ割込みハンドラが使用するスタック領域

（説明）

①この配列は、割込みハンドラ用スタック領域定義プログラム**(hiintstk.c)**で確保されています。

Ｂ．４．２　タイマ初期化ルーチン(hi_tmrini)

図Ｂ－６にタイマ初期化ルーチンのプログラム内容**(hitmrdrv.c)**を示します。

タイマ初期化ルーチンはシステム初期化ハンドラ**(hi_sysini)**から呼ばれ、**TMU**の各レジスタ**(TSTR,TCR,TCOR,TCNT)**と割込み優先レベル設定レジスタ**(IPR)**の設定を行ないます。

```
/************************************************************************/
/*SPECIFICATIONS ;                                                      */
/*  NAME     = hi_tmrini ;                                             */
/*  DATE     = 95/02/20 ;                                              */
/*  AUTHOR   = Hitachi, Ltd. ;                                         */
/*  FUNCTION  = HI-SH77 timer initialize routine ;                     */
/*  ATTRIBUTE = PUBLIC ;                                               */
/*  HISTORY   = V1.0 ;                                                 */
/*END OF SPECIFICATIONS ;                                             */
/************************************************************************/
void    hi_tmrini(void)
{
  VP    gbrsave;                    /* save variable of GBR          */
  UH    ipr;                        /* valiable  of  IPR             */

  gbrsave = get_gbr();              /* save GBR to gbrsave           */   ①
  set_gbr((VP)IOBASE);              /* set I/O base address to GBR   */   ②

  gbr_and_byte(TSTR, TCNTSTP);      /* stop TCNT (ch0)               */   ③
  ipr = gbr_read_word(IPR) & IPRCLR;/* clear IPR (TMU-ch0) to ipr    */   ④
                                    /* set TINTLVL to IPR (TMU-ch0)  */
   gbr_write_word(IPR, ipr ¦ (TINTLVL << 12));
  gbr_write_word(TCR, TCRDAT);      /* initialize TCR                */   ⑤
  gbr_write_long(TCOR, TCORDAT);    /* set count to TCOR             */   ⑥
  gbr_write_long(TCNT, TCNTCLR);    /* set count to TCNT             */   ⑦
  gbr_or_byte(TSTR, TCNTSTR);       /* start TCNT (ch0)              */   ⑧

  set_gbr(gbrsave);                 /* load GBR from gbrsave         */   ⑨
}
```

図Ｂ－６　タイマ初期化ルーチン(hi_tmrini)

（説明）

①グローバルベースレジスタ**(GBR)**の値を保存します**(get_gbr)**。

②**GBR**に**I/O**ベースアドレス**(IOBASE)**を設定します**(set_gbr)**。

③タイマスタートレジスタ**(TSTR)**にタイマ動作の停止を設定します。

④**IPR**にタイマドライバの動作環境データ(**TINTLVL**)を設定します。

⑤タイマコントロールレジスタ**(TCR)**にアンダフローによる割込み許可、立ち上がりエッジ、
　カウンタクロックPφ／４を設定します。

⑥タイマカウンタ**(TCNT)**をクリアします。

⑦タイマコンスタントレジスタ**(TCOR)**にリロード値を設定します。

⑧**TSTR**にタイマ動作の開始を設定します。

⑨**GBR**の値を元に戻します**(set_gbr)**。

Ｂ．４．３　タイマ割込みハンドラ(hi_tmrhdr)

図Ｂ－７にタイマ割込みハンドラ（**hi_tmrhdr**）のプログラム内容**(hitmrdrv.c)**を示します。

タイマ割込みハンドラは、割込みマスクレベルの設定、例外マスクの解除、タイマ割込みのクリアを行ない、**sys_clk**システムコールによりカーネルに対して時間管理処理要求を行ないます。割込みハンドラ終了時は **#pragma interrupt**文の宣言により、**ret_int**システムコール**(TRAPA#61)**を発行します。

```
/***************************************************************************/
/*SPECIFICATIONS :                                                         */
/*  NAME     = hi_tmrhdr ;                                                 */
/*  DATE     = 95/02/20 ;                                                  */
/*  AUTHOR   = Hitachi, Ltd. ;                                             */
/*  FUNCTION = HI-SH77 hardware timer interrupt handler ;                  */
/*  ATTRIBUTE = PUBLIC ;                                                   */
/*  HISTORY  = V1.0 ;                                                      */
/*END OF SPECIFICATIONS :                                                  */
/***************************************************************************/
#pragma interrupt   (hi_tmrhdr(sp = p_intstk13, tn = 61))                          ①

TMRHDR  hi_tmrhdr(void)
{
  VP    gbrsave;                    /* save variable of GBR              */

  set_cr(MD_SET | (TINTVAL << 4));  /* clear BL, set imask               */      ②

  gbrsave = get_gbr();              /* save GBR to gbrsave               */      ③
  set_gbr((VP)IOBASE);              /* set I/O base address to GBR       */      ④

  gbr_write_word(TCR, TCRDAT);      /* clear  UNF                        */      ⑤

  sys_clk();                        /* request kernel timer processing   */      ⑥

  set_gbr(gbrsave);                 /* load GBR from gbrsave             */      ⑦
}
```

図Ｂ－７　タイマ割込みハンドラ(hi_tmrhdr)

（説明）

①**#pragma interrupt**文により、割込みハンドラ用スタックと、割込みハンドラ終了時の**ret_int**システムコール発行**(TRAPA#61)**を宣言します。

②割込みマスクレベルの設定、例外マスクの解除をします**(set_cr)**。

③グローバルベースレジスタ**(GBR)**の値を保存します**(get_gbr)**。

④**GBR**に**I/O**ベースアドレス**(IOBASE)**を設定します**(set_gbr)**。

⑤**TCR**のアンダフローフラグをクリアします。

⑥**sys_clk**システムコールにより、カーネルに対して時間管理処理を要求します。

⑦**GBR**の値を元に戻します**(set_gbr)**。

付録Ｃ．システムコール一覧

Ｃ．１　システムコールC言語インタフェース一覧

タスク管理機能

1	ER ercd=	cre_tsk(ID tskid, TASKP stadr, TPRI itskpri);	Create Task
2	ER ercd=	sta_tsk(ID tskid);	Start Task
3	ER ercd=	ista_tsk(ID tskid);	Interrupt Start Task
4	ER ercd=	del_tsk(ID tskid);	Delete Task
5	void	ext_tsk(void);	Exit Task
6	void	exd_tsk(void);	Exit and Delete Task
7	ER ercd=	ter_tsk(ID tskid);	Terminate Task
8	ER ercd=	chg_pri(ID tskid,TPRI tskpri);	Change Task Priority
9	ER ercd=	ichg_pri(ID tskid,TPRI tskpri);	Interrupt Change Task Priority
10	ER ercd=	rot_rdq(TPRI tskpri);	Rotate Ready Queue
11	ER ercd=	irot_rdq(TPRI tskpri);	Interrupt Rotate Ready Queue
12	ER ercd=	rel_wai(ID taskid);	Release Wait
13	ER ercd=	irel_wai(ID taskid);	Interrupt Release Wait
14	ER ercd=	get_tid(ID *p_tskid);	Get Task Identifier
15	ER ercd=	iget_tid(ID *p_tskid);	Interrupt Get Task Identifier
16	ER ercd=	tsk_sts(UH *p_tskstat, TPRI *p_tskpri, ID tskid);	Get Task Status
17	ER ercd=	itsk_sts(UH *p_tskstat, TPRI *p_tskpri, ID tskid);	Interrupt Get Task Status

タスク付属同期機能

18	ER ercd=	sus_tsk(ID tskid);	Suspend Task
19	ER ercd=	isus_tsk(ID tskid);	Interrupt Suspend Task
20	ER ercd=	rsm_tsk(ID tskid);	Resume Task
21	ER ercd=	irsm_tsk(ID tskid);	Interrupt Resume Task
22	ER ercd=	slp_tsk(void);	Sleep Task
23	ER ercd=	wai_tsk(TMO tmout);	Wait Task
24	ER ercd=	wup_tsk(ID tskid);	Wakeup Task
25	ER ercd=	iwup_tsk(ID tskid);	Interrupt Wakeup Task
26	ER ercd=	can_wup(W *p_wupcnt, ID tskid);	Cancel Wakeup Task
27	ER ercd=	ican_wup(W *p_wupcnt, ID tskid);	Interrupt Cancel Wakeup Task

同期／通信機能

28	ER ercd=	set_flg(ID flgid, UW setptn);	Set EventFlag
29	ER ercd=	iset_flg(ID flgid, UW setptn);	Interrupt Set EventFlag
30	ER ercd=	clr_flg(ID flgid, UW clrptn);	Clear EventFlag
31	ER ercd=	iclr_flg(ID flgid, UW clrptn);	Interrupt Clear EventFlag
32	ER ercd=	wai_flg(UW *p_flgptn, ID flgid, UW waiptn, UW wfmode);	Wait EventFlag
33	ER ercd=	pol_flg(UW *p_flgptn, ID flgid, UW waiptn, UW wfmode);	Poll EventFlag
34	ER ercd=	ipol_flg(UW *p_flgptn, ID flgid, UW waiptn, UW wfmode);	Interrupt Poll EventFlag
35	ER ercd=	flg_sts(ID *p_wtskid, UW *p_flgptn, ID flgid);	Get EventFlag Status
36	ER ercd=	iflg_sts(ID *p_wtskid, UW *p_flgptn, ID flgid);	Interrupt Get EventFlag Status
37	ER ercd=	sig_sem(ID semid);	Signal Semaphore
38	ER ercd=	isig_sem(ID semid);	Interrupt Signal Semaphore
39	ER ercd=	wai_sem(ID semid);	Wait on Semaphore
40	ER ercd=	preq_sem(ID semid);	Poll and Request Semaphore
41	ER ercd=	ipreq_sem(ID semid);	Interrupt Poll and Request Semaphore
42	ER ercd=	sem_sts(ID *p_wtskid, W *p_semcnt, ID semid);	Get Samaphore Status

```
43  ER ercd=  isem_sts(ID *p_wtskid, W *p_semcnt, ID semid);        Interrupt Get Samaphore Status
44  ER ercd=  snd_msg(ID mbxid, T_MSG *pk_msg);                          Send Message to Mailbox
45  ER ercd=  isnd_msg(ID mbxid, T_MSG *pk_msg);                Interrupt Send Message to Mailbox
46  ER ercd=  rcv_msg(T_MSG **ppk_msg, ID mbxid);                     Receive Message from Mailbox
47  ER ercd=  prcv_msg(T_MSG **ppk_msg, ID mbxid);       Poll and Receive Message from Mailbox
48  ER ercd=  iprcv_msg(T_MSG **ppk_msg, ID mbxid);
                                          Interrupt Poll and Receive Message from Mailbox
49  ER ercd=  mbx_sts(ID *p_wtskid, T_MSG **ppk_msg, ID mbxid);          Get Mailbox Status
50  ER ercd=  imbx_sts(ID *p_wtskid, T_MSG **ppk_msg, ID mbxid);  Interrupt Get Mailbox Status
```

割込み管理機能

```
51              ret_int();*1                                          Return from Interrupt Handler
52  ER ercd=  chg_ims(SR imask);                                      Change Interrupt Mask Level
53  ER ercd=  ichg_ims(SR imask);                            Interrupt Change Interrupt Mask Level
54  ER ercd=  ims_sts(SR *p_imask);                                Get Interrupt Mask Level Status
55  ER ercd=  iims_sts(SR *p_imask);                      Interrupt Get Interrupt Mask Level Status
```

例外管理機能

```
56              ret_exc(); *2                                         Return from Exception
```

メモリプール管理機能

```
57  ER ercd=  get_blk(VP *p_blk, ID mplid);                                Get Memory Block
58  ER ercd=  pget_blk(VP *p_blk, ID mplid);                      Poll and Get Memory Block
59  ER ercd=  ipget_blk(VP *p_blk, ID mplid);            Interrupt Poll and Get Memory Block
60  ER ercd=  rel_blk(ID mplid, VP blk);                             Release Memory Block
61  ER ercd=  irel_blk(ID mplid, VP blk);                   Interrupt Release Memory Block
62  ER ercd=  mpl_sts(ID *p_wtskid, W *p_frbcnt, ID mplid);          Get Memory Pool Status
63  ER ercd=  impl_sts(ID *p_wtskid, W *p_frbcnt, ID mplid);  Interrupt Get Memory Pool Status
```

時間管理機能

```
64  ER ercd=  set_tim(T_TIM *pk_time);                                            Set Time
65  ER ercd=  iset_tim(T_TIM *pk_time);                                 Interrupt Set Time
66  ER ercd=  get_tim(T_TIM *pk_time);                                            Get Time
67  ER ercd=  iget_tim(T_TIM *pk_time);                                 Interrupt Get Time
68  void      sys_clk(void);                                          Count System Clock
```

システム管理機能

```
69  ER ercd=  get_ver(T_VER *pk_ver);                                      Get Version No
70  ER ercd=  iget_ver(T_VER *pk_ver);                           Interrupt Get Version No
```

拡張SVC機能

```
71  ER ercd=  sys_cal(FN fncd, VW para1, VW para2, VW para3, VW para4);        System Call
```

【注】*1 C言語プログラムにおいて、 "#pragma interrupt"文を宣言し、TRAPA #61(ret_int)を発行します。
*2 C言語プログラムにおいて、 "#pragma interrupt"文を宣言し、TRAPA #57(ret_exc)を発行します。

C．2　システムコール・アセンブラインタフェース一覧

以下に、システムコールのアセンブラインタフェース一覧を示します。
表中の意味を次に示します。

タスク／非タスク**(T/N)**　　発行できるシステム状態を示します
　　T:タスク部から発行できるシステムコール
　　N:非タスク部から発行できるシステムコール
＝　　リターンパラメータに使用（リターン値が返る）
トラップ番号**(#n)**　　システムコール発行時に使用する**TRAPA**命令の番号**(n)**

システムコール			タスク/非タスク (T/N)	パラメータ/リターンパラメータ(=) R0	R4	R5	R6	R7	トラップ番号
タスク管理機能									
1	cre_tsk	(Create Task)	T	H'fffffeff ercd=	tskid	stadr	itskpri		#63
2	sta_tsk	(Start Task)	T	H'ffffffef ercd=	tskid				#63
3	ista_tsk	(Interrupt Start Task)	N	H'ffffffff ercd=	tskid				#62
4	del_tsk	(Delete Task)	T	H'ffffffed ercd=	tskid				#63
5	ext_tsk	(Exit Task)	T	H'ffffffec					#63
6	exd_tsk	(Exit and Delete Task)	T	H'ffffffeb					#63
7	ter_tsk	(Terminate Task)	T	H'ffffffea ercd=	tskid				#63
8	chg_pri	(Change Task Priority)	T	H'ffffffe9 ercd=	tskid	tskpri			#63
9	ichg_pri	(Interrupt Change Task Priority)	N	H'ffffffe ercd=	tskid	tskpri			#62
10	rot_rdq	(Rotate Ready Queue)	T	H'ffffffe8 ercd=	tskpri				#63
11	irot_rdq	(Interrupt Rotate Ready Queue)	N	H'fffffffd ercd=	tskpri				#62
12	rel_wai	(Release Wait)	T	H'ffffffe7 ercd=	tskid				#63
13	irel_wai	(Interrupt Release Wait)	N	H'fffffffc ercd=	tskid				#62
14	get_tid	(Get Task Identifier)	T	H'ffffffe6 ercd=	p_tskid p_tskid=				#63
15	iget_tid	(Interrupt Get Task Identifier)	N	H'fffffffb ercd=	p_tskid p_tskid=				#62
16	tsk_sts	(Get Task Status)	T	H'ffffffe5 ercd=	p_tskstat p_tskstat=	p_tskpri p_tskpri=	tskid		#63
17	itsk_sts	(Interrupt Get Task Status)	N	H'fffffffa ercd=	p_tskstat p_tskstat=	p_tskpri p_tskpri=	tskid		#62

	システムコール		タスク/非タスク (T/N)	パラメータ/リターンパラメータ(=) R0	R4	R5	R6	R7	トラップ番号
タスク付属同期機能									
18	sus_tsk	(Suspend Task)	T	H'fffffef4 ercd=	tskid				#63
19	isus_tsk	(Interrupt Suspend Task)	N	H'ffffffff9 ercd=	tskid				#62
20	rsm_tsk	(Resume Task)	T	H'fffffef3 ercd=	tskid				#63
21	irsm_tsk	(Interrupt Resume Task)	N	H'ffffffff8 ercd=	tskid				#62
22	slp_tsk	(Sleep Task)	T	H'fffffef2 ercd=					#63
23	wai_tsk	(Wait Task)	T	H'fffffef1 ercd=	tmout				#63
24	wup_tsk	(Wakeup Task)	T	H'fffffef0 ercd=	tskid				#63
25	iwup_tsk	(Interrupt Wakeup Task)	N	H'ffffffff7 ercd=	tskid				#62
26	can_wup	(Cancel Wakeup Task)	T	H'fffffeef ercd=	p_wupcnt p_wupcnt=	tskid			#63
27	ican_wup	(Interrupt Cancel Wakeup Task)	N	H'ffffffff6 ercd=	p_wupcnt p_wupcnt=	tskid			#62
同期／通信機能									
28	set_flg	(Set EventFlag)	T	H'fffffeee ercd=	flgid	setptn			#63
29	iset_flg	(Interrupt Set EventFlag)	N	H'ffffffff5 ercd=	flgid	setptn			#62
30	clr_flg	(Clear EventFlag)	T	H'fffffeed ercd=	flgid	clrptn			#63
31	iclr_flg	(Interrupt Clear EventFlag)	N	H'ffffffff4 ercd=	flgid	clrptn			#62
32	wai_flg	(Wait EventFlag)	T	H'fffffeec ercd=	p_flgptn p_flgptn=	flgid	waiptn	wfmode	#63
33	pol_flg	(Poll EventFlag)	T	H'fffffeeb ercd=	p_flgptn p_flgptn=	flgid	waiptn	wfmode	#63
34	ipol_flg	(Interrupt Poll EventFlag)	N	H'ffffffff3 ercd=	p_flgptn p_flgptn=	flgid	waiptn	wfmode	#62
35	flg_sts	(Get EventFlag Status)	T	H'fffffeea ercd=	p_wtskid p_wtskid=	p_flgptn p_flgptn=	flgid		#63
36	iflg_sts	(Interrupt Get EventFlag Status)	N	H'ffffffff2 ercd=	p_wtskid p_wtskid=	p_flgptn p_flgptn=	flgid		#62
37	sig_sem	(Signal Semaphore)	T	H'fffffee9 ercd=	semid				#63
38	isig_sem	(Interrupt Signal Semaphore)	N	H'ffffffff1 ercd=	semid				#62
39	wai_sem	(Wait on Semaphore)	T	H'fffffee8 ercd=	semid				#63
40	preq_sem	(Poll and Request Semaphore)	T	H'fffffee7 ercd=	semid				#63
41	ipreq_sem	(Interrupt Poll and Request Semaphore)	N	H'ffffffff0 ercd=	semid				#62

	システムコール	タスク/非タスク (T/N)	パラメータ/リターンパラメータ(=) R0	R4	R5	R6	R7	トラップ番号
42	sem_sts (Get Smaphore Status)	T	H'ffffffe6 ercd=	p_wtskid p_wtskid=	p_semcnt p_semcnt=	semid		#63
43	isem_sts (Interrupt Get Smaphore Status)	N	H'fffffffef ercd=	p_wtskid p_wtskid=	p_semcnt p_semcnt=	semid		#62
44	snd_msg (Send Message to Mailbox)	T	H'ffffffe5 ercd=	mbxid	pk_msg			#63
45	isnd_msg (Interrupt Send Message to Mailbox)	N	H'ffffffee ercd=	mbxid	pk_msg			#62
46	rcv_msg (Receive Message from Mailbox)	T	H'ffffffe4 ercd=	ppk_msg ppk_msg=	mbxid			#63
47	prcv_msg (Poll and Receive Message from Mailbox)	T	H'ffffffe3 ercd=	ppk_msg ppk_msg=	mbxid			#63
48	iprcv_msg (Interrupt Poll and Receive Message from Mailbox)	N	H'fffffffed ercd=	ppk_msg ppk_msg=	mbxid			#62
49	mbx_sts (Get Mailbox Status)	T	H'ffffffe2 ercd=	p_wtskid p_wtskid=	ppk_msg ppk_msg=	mbxid		#63
50	imbx_sts (Interrupt Get Mailbox Status)	N	H'fffffffec ercd=	p_wtskid p_wtskid=	ppk_msg ppk_msg=	mbxid		#62
	割込み管理機能							
51	ret_int (Return from Interrupt Handler)	N						#61
52	chg_ims (Change Interrupt Mask Level)	T	H'ffffffe1 ercd=	imask				#63
53	ichg_ims (Interrupt Change Interrupt Mask Level)	N	H'fffffffeb ercd=	imask				#62
54	ims_sts (Get Interrupt Mask Level Status)	T	H'ffffffe0 ercd=	p_imask p_imask=				#63
55	iims_sts (Interrupt Get Interrupt Mask Level Status)	N	H'fffffffea ercd=	p_imask p_imask=				#62
	例外管理機能							
56	ret_exc (Return from Exception)	T,N						#57
	メモリプール管理機能							
57	get_blk (Get Memory Block)	T	H'ffffffedf ercd=	p_blk p_blk=	mplid			#63
58	pget_blk (Poll and Get Memory Block)	T	H'ffffffede ercd=	p_blk p_blk=	mplid			#63
59	ipget_blk (Interrupt Poll and Get Memory Block)	N	H'ffffffe9 ercd=	p_blk p_blk=	mplid			#62
60	rel_blk (Release Memory Block)	T	H'ffffffedd ercd=	mplid	blk			#63
61	irel_blk (Interrupt Release Memory Block)	N	H'ffffffe8 ercd=	mplid	blk			#62
62	mpl_sts (Get Memory Pool Status)	T	H'ffffffedc ercd=	p_wtskid p_wtskid=	p_frbcnt p_frbcnt=	mplid		#63
63	impl_sts (Interrupt Get Memory Pool Status)	N	H'ffffffe7 ercd=	p_wtskid p_wtskid=	p_frbcnt p_frbcnt=	mplid		#62

	システムコール	タスク/非タスク (T/N)	パラメータ/リターンパラメータ(=) R0	R4	R5	R6	R7	トラップ番号
時間管理機能								
64	set_tim (Set Time)	T	H'fffffedb ercd=	pk_time				#63
65	iset_tim (Interrupt Set Time)	N	H'ffffffe6 ercd=	pk_time				#62
66	get_tim (Get Time)	T	H'fffffeda ercd=	pk_time pk_time=				#63
67	iget_tim (Interrupt Get Time)	N	H'ffffffe5 ercd=	pk_time pk_time=				#62
68	sys_clk(Count System Clock)	N						#60
システム管理機能								
69	get_ver (Get Version No)	T	H'fffffed9 ercd=	pk_ver pk_ver=				#63
70	iget_ver (Interrupt Get Version No)	N	H'ffffffe4 ercd=	pk_ver pk_ver=				#62
拡張SVC機能								
71	sys_cal (System Call)	T	H'00000001<= ercd=	para1	para2	para3	para4	#63

付録Ｄ．SH3に関する注意事項

本章では、**SH3**を使用する上で特に注意すべきことを記述しています。

Ｄ．１　エンディアン

SH3は、ビッグエンディアンとリトルエンディアンの両方をサポートしていますが、**HI-SH77**はビッグエンディアンのみで動作します。

Ｄ．２　SRレジスタ状態

各種ユーザプログラム実行中の**SR**レジスタの状態を、表**Ｄ－１**に示します。

表Ｄ－１　各種プログラムでのSRレジスタ状態

本文の項番	ﾌﾟﾛｸﾞﾗﾑ種別	タスク	割込みハンドラ	拡張SVCハンドラ	例外処理ルーチン
(1)	モード(MDﾋﾞｯﾄ)	・起動時は特権ﾓｰﾄﾞ ・変更可能	・起動時は特権ﾓｰﾄﾞ ・変更可能	・起動時は特権ﾓｰﾄﾞ ・変更不可	・起動時は特権ﾓｰﾄﾞ ・変更可能
(2)	例外ﾏｽｸ(BLﾋﾞｯﾄ)	起動時は例外ﾏｽｸ解除	起動時は例外ﾏｽｸ	起動時は例外ﾏｽｸ解除	起動時は例外ﾏｽｸ
(2)	割込みﾏｽｸ(I3～I0ﾋﾞｯﾄ)	起動時は0	・起動時は割込み前の値 ・ﾊﾝﾄﾞﾗでﾏｽｸ処理が必要	起動時は0	起動時は例外前の値
(3)	ﾚｼﾞｽﾀﾊﾞﾝｸ(RBﾋﾞｯﾄ)	起動時はバンク0			

本文の項番	ﾌﾟﾛｸﾞﾗﾑ種別	MCU初期化ルーチン	ｼｽﾃﾑ初期化ハンドラ	未定義割込みﾙｰﾁﾝ, 未定義ﾄﾗｯﾌﾟﾙｰﾁﾝ	ｼｽﾃﾑ異常終了処理ﾙｰﾁﾝ
(1)	モード(MDﾋﾞｯﾄ)	・起動時は特権ﾓｰﾄﾞ ・変更不可	・起動時は特権ﾓｰﾄﾞ ・変更不可	・起動時は特権ﾓｰﾄﾞ ・変更可能	・起動時は不定 ・変更可能 ・特権ﾓｰﾄﾞで動作できるよう作成してください
(2)	例外ﾏｽｸ(BLﾋﾞｯﾄ)	起動時は例外ﾏｽｸ	起動時は例外ﾏｽｸ解除	起動時は例外ﾏｽｸ	起動時は不定
(2)	割込みﾏｽｸ(I3～I0ﾋﾞｯﾄ)	起動時は15	起動時はｶｰﾈﾙ割込みﾏｽｸﾚﾍﾞﾙ	起動時は事象発生前の値	起動時は事象発生前の値
(3)	ﾚｼﾞｽﾀﾊﾞﾝｸ(RBﾋﾞｯﾄ)	起動時はバンク0			

（1）モード

SH3には、特権モードとユーザモードの２つのモードがありますが、**HI-SH77**においてはすべてのプログラムが特権モードで起動されます。ユーザプログラム内で、ユーザモードに移行しても構いません。

なお、カーネル本体は特権モードで動作します。

（2）例外マスクと割込みマスク

SH3は、割込みや例外発生時にソフトウェアで要因を解析して要因毎のルーチンに分岐させる、というアーキテクチャです。この処理をソフトウェアで行なうためにはその間の割込みや例外の再入を禁止する必要があり、これを実現するために**SH3**には例外マスク機能があります。例外マスクは**SR**レジスタの**BL**ビットで制御されます。

HI-SH77では、この処理をカーネルの例外サービスルーチンとして提供しています。

ユーザの割込みハンドラや例外処理ルーチン起動時は、**SR**レジスタは以下の状態になっています。

(1) 例外マスク状態

(2) 割込みマスクは割込み／例外発生前の値

例外マスク状態は、システムのスループットを向上させるためには速やかに解除する必要があります。

また、割込みハンドラの場合は割込みの再入を防ぐために発生した割込みレベルを設定する必要があります。この処理は起動後例外マスクを解除するまでの間に行なう必要があります。直接**SR**レジスタを変更してください。**C**言語で記述する場合は、**C**コンパイラの組み込み関数の**set_imask()**または**set_cr()**を利用してください。

例外マスク状態ではシステムコールは発行できません。発行した場合、システムの正常な動作は保証されません。

（3）レジスタバンク

SH3は、バンク０とバンク１の２つのレジスタバンクを持っています。レジスタバンクは、**SR**レジスタの**RB**ビットで制御されます。

通常はカーネルも含めてバンク０のレジスタ（**R0_BANK0 ～ R7_BANK0**）を使用します。

バンク１のレジスタ（**R0_BANK1 ～ R7_BANK1**）は、カーネルの例外サービスルーチンで使用しています。したがって、ユーザがバンク１のレジスタを使用するには、例外マスク状態で使用する必要があります。

なお、バンク１のレジスタを使用しているときは必然的に例外マスク状態であるため、システムコールは発行できません。

Ｄ．３　VBRレジスタ

VBRは、システム稼働中は一切変更しないでください。変更した場合、システムの正常な動作は保証されません。なお、**VBR**の初期設定はカーネルのリセットサービスルーチンによって行なわれます。

Ｄ．４　SPC,SSRレジスタ

SPC,SSRを使用する場合は、例外マスク状態で使用してください。

Ｄ．５　キャッシュ

カーネルは、キャッシュの制御は行ないません。必要に応じて**MCU**初期化ルーチンでキャッシュの使用／未使用を設定してください。なお、出荷時の**MCU**初期化ルーチンのサンプルではキャッシュを使用しない初期化を行なっています。

D．6　MMU

カーネルは、**MMU**をサポートしていません。なお、出荷時の**MCU**初期化ルーチンのサンプルでは**MMU**を使用しない初期化を行なっています。

D．7　プログラム配置

システム結合時、リセットサービスルーチンは必ず**H'A0000000**番地に配置してください。例外サービスルーチンは**P1**または**P2**領域の先頭から**H'100**番地以降に配置してください。**MCU**初期化ルーチンは**P2**領域に配置してください。また、ベクタテーブルはエリア**0**に配置してください。

付録Ｅ．エラーコード一覧

Ｅ．1　システムコールエラーコード一覧

表Ｅ－１　システムコールエラーコード一覧

	ｴﾗｰｺｰﾄﾞ (ﾆｰﾓﾆｯｸ)	ｴﾗｰｺｰﾄﾞ (ercd)	ｴﾗｰﾁｪｯｸ 種別	エラー内容
1	E_OK	H'00000000	[k]	正常終了
2	E_TNOSPT	H'fffff9ed(-H'613)	[p]	タイマがサポートされていない(ｼｽﾃﾑﾀｲﾏが未登録)
3	E_RSFN	H'fffff9ec(-H'614)	[k]	予約機能コード番号(未定義の機能コード指定)
4	E_RSID	H'fffff9e9(-H'617)	[p]	予約ID(cre_tskｼｽﾃﾑｺｰﾙでtskid=0)
5	E_PAR	H'fffff8df(-H'721)	[p]	パラメータエラー
6	E_ILADR	H'fffff8de(-H'722)	[p]	不正アドレス(ｱﾄﾞﾚｽが奇数, ｱﾄﾞﾚｽが4ﾊﾞｲﾄ境界でない)
7	E_IDOVR	H'fffff8dd(-H'723)	[p]	IDが範囲外(tskid<0, tskid>max_tskid)
8	E_TPRI	H'fffff8da(-H'726)	[p]	不正タスク優先度(<0, 最大ﾀｽｸ優先度を越えて指定)
9	E_ILTIME	H'fffff8d9(-H'727)	[p]	不正時間指定(tmout≦-1, pk_timeで示される値が負)
10	E_ILMSG	H'fffff8d7(-H'729)	[k]	不正メッセージ形式(ﾒｯｾｰｼﾞ先頭4ﾊﾞｲﾄが0でない)
11	E_IMS	H'fffff8d5(-H'72b)	[p]	不正IMASK(割込みﾏｽｸ値が0～15以外)
12	E_SELF	H'fffff7cf(-H'831)	[k]	自タスク指定(tskid=自ﾀｽｸID)
13	E_EXS	H'fffff7cd(-H'833)	[k]	オブジェクトが存在している
14	E_NOEXS	H'fffff7cc(-H'834)	[p]	オブジェクトが存在していない(IDが範囲外, 予約ID)
			[k]	オブジェクトが未登録(タスクが生成されていない)
15	E_DMT	H'fffff7cb(-H'835)	[k]	タスクが休止状態である
16	E_NODMT	H'fffff7ca(-H'836)	[k]	タスクが休止状態でない
17	E_NOSUS	H'fffff7c8(-H'838)	[k]	タスクが強制待ち状態でない
18	E_ILBLK	H'fffff7c5(-H'83b)	[p]	不正メモリブロックの返却、操作 (ﾒﾓﾘﾌﾟｰﾙに含まれない、ﾒﾓﾘﾌﾞﾛｯｸの先頭でない)
			[k]	すでに返却したメモリブロックの返却、操作
19	E_NOWAI	H'fffff7c2(-H'83e)	[k]	タスクが待ち状態でない
20	E_CTX	H'fffff5bb(-H'a45)	[k]	コンテキストエラー(ﾀｽｸ部または非ﾀｽｸ部からの発行)
21	E_QOVR	H'fffff4b7(-H'b49)	[k]	キューイングのオーバーフロー (強制待ち状態へのsus_tskｼｽﾃﾑｺｰﾙ, 起床要求回数ｵｰﾊﾞｰ, 待ちﾀｽｸが存在しているｲﾍﾞﾝﾄﾌﾗｸﾞへのwai_flgｼｽﾃﾑｺｰﾙ, ｾﾏﾌｫｶｳﾝﾀのｵｰﾊﾞｰﾌﾛｰ)
22	E_TMOUT	H'fffff2ab(-H'd55)	[k]	タイムアウト（起床要求なし） (wai_tskｼｽﾃﾑｺｰﾙで指定した待ち時間が経過した)
24	E_RLWAI	H'fffff2aa(-H'd56)	[k]	待ち(WAIT)状態が強制解除された
25	E_PLFAIL	H'fffff1a7(-H'e59)	[k]	ポーリング失敗(同期通信，資源の獲得を失敗した)

エラーチェック種別：[p]は、パラメータチェック機能を組み込んだときにチェックされるエラー
[k]は、パラメータチェック機能なしでも常にチェックされるエラー

Ｅ．２　システム異常終了エラーコード一覧

表Ｅ－２　システム異常終了エラーコード一覧

項番	エラー種別 **(R4)**	エラーコード **(R5)**	システムダウン情報**(R6)**	内　　容
1	**H'7FFFFFFF** ～ **H'00000001**	ユーザ仕様による		ユーザプログラム内で発生したシステム異常を**hi_sysdwn**で処理する場合に、正のエラー種別が使用できます
2	**H'00000000**	セットアップテーブルチェックエラーコード	**H'00000000**	セットアップテーブルのエラー（セットアップテーブルチェック機能が定義されている場合のみ）
3	**H'FFFFFFFF**	**H'FFFFF5BB (E_CTX)**	**ext_tsk**を発行した(アドレス+2)番地	非タスク部での**ext_tsk**システムコールの発行によるコンテキストエラー
4	**H'FFFFFFFE**	**H'FFFFF5BB (E_CTX)**	**exd_tsk**を発行した(アドレス+2)番地	非タスク部での**exd_tsk**システムコールの発行によるコンテキストエラー
5	**H'FFFFFFFD**	**H'FFFFF5BB (E_CTX)**	**ret_int**を発行した(アドレス+2)番地	タスク部での**ret_int**システムコールの発行によるコンテキストエラー
6	**H'FFFFFFFC**	**H'FFFFF5BB (E_CTX)**	**sys_clk**を発行した(アドレス+2)番地	タスク部での**sys_clk**システムコールの発行によるコンテキストエラー
7	**H'FFFFFFFB** ～ **H'80000000**	システム予約		システム予約

E．３　セットアップテーブルエラーコード一覧

表Ｅ－３　セットアップテーブルエラーコード一覧

	ｴﾗｰｺｰﾄﾞ (ﾆｰﾓﾆｯｸ)	ｴﾗｰｺｰﾄﾞ (ercd)	エラー内容
1	ES_KMSKLVL	H'00000001	カーネル割込みマスクレベル(hi_knlmsklvl)が0または16以上である
2	ES_INSTNUM	H'00000002	カーネル割込みマスクレベルより高い割込みネスト数(hi_uppintnst)が16以上または低い割込みネスト数(hi_lowintnst)が16以上である
3	ES_MTSKID	H'00000003	最大タスクID(hi_maxtskid)が1024以上である
4	ES_MTSKPRI	H'00000004	最大タスク優先度(hi_maxtskpri)が256以上または0である
5	ES_ITSKNUM	H'00000005	初期登録タスク数(hi_initsknum)が1024以上である
6	ES_ITSKSP	H'00000006	タスクスタックポインタ(_0Hinitsp[])の値が4バイト境界でない
7	ES_ITSKID	H'00000007	初期登録タスク(_0Hinitsk)のタスクIDが0または最大タスクID(hi_maxtskid)を越えている 初期登録タスク(_0Hinitsk)間のタスクIDが重なっている
8	ES_ITSKPRI	H'00000008	初期登録タスク(_0Hinitsk)のタスク優先度が0または最大タスク優先度(hi_maxtskpri)を越えている
9	ES_ITSKADR	H'00000009	初期登録タスク(_0Hinitsk)のタスク開始アドレスが2バイト境界でない
10	ES_MFLGID	H'0000000A	最大イベントフラグID(hi_maxflgid)が1024以上である
11	ES_MSEMID	H'0000000B	最大セマフォID(hi_maxsemid)が1024以上である
12	ES_MMBXID	H'0000000C	最大メイルボックスID(hi_maxmbxid)が1024以上である
13	ES_MMPLID	H'0000000D	最大メモリプールID(hi_maxmplid)が1024以上である
14	ES_MPLADR	H'0000000E	メモリプール領域(_0Hinimpl)の先頭アドレスが4バイト境界でない、または0である
15	ES_BLKSIZ	H'0000000F	メモリブロックサイズ(_0Hinimpl)が4バイト境界でない、または0である
16	ES_BLKCNT	H'00000010	メモリブロック数(_0Hinimpl)が0である
17	ES_MPLARA	H'00000011	メモリプール領域がアドレスの上限(32bit)を越えている
18	ES_IHDRADR	H'00000012	システム初期化ハンドラ(_0Hiniihdr)の開始アドレスが2バイト境界でない
19	ES_MSVCFCD	H'00000013	最大拡張SVC機能コード(hi_maxsvccd)が256以上である
20	ES_SVCADR	H'00000014	拡張SVCハンドラ開始アドレス(_0Hinisvc)が2バイト境界でない
21	ES_TRCADR	H'00000015	トレースバッファ先頭アドレス(_0Hinitrc)が4バイト境界でない
22	ES_TRCCNT	H'00000016	トレースエントリ情報取得数(_0Hinitrc)が0である
23	ES_TRCARA	H'00000017	トレースバッファ領域がアドレスの上限(32bit)を越えている

付録Ｆ．ヘッダファイル

Ｆ．１　Ｃ言語用ヘッダファイル

```
/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/*              HI-SH77 header file (C language) ( Ver. 1.0 )                */
/*                                                                           */
/*                                                                           */
/*              Copyright (c)  Hitachi, Ltd. 1995.                           */
/*              Licensed Material of Hitachi, Ltd.                           */
/*                                                                           */
/*              HI-SH77(HS0770ITCN1SM) V1.0                                  */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
/*****************************************************************************/
/*SPECIFICATIONS ;                                                           */
/*  FILE      = itron.h ;                                                    */
/*  DATE      = 95/02/20 ;                                                   */
/*  AUTHOR    = Hitachi, Ltd. ;                                              */
/*  FUNCTION  = HI-SH77 header file (C language) ;                           */
/*  ATTRIBUTE = PUBLIC ;                                                     */
/*  HISTORY   = V1.0 ;                                                       */
/*END OF SPECIFICATIONS ;                                                    */
/*****************************************************************************/
/*****************************************************************************/
/*                                                                           */
/*                              type definition                              */
/*                                                                           */
/*****************************************************************************/
/*-------------------------------- common ---------------------------------*/
typedef char            B;              /* signed 8 bit integer             */
typedef short           H;              /* signed 16 bit integer            */
typedef long            W;              /* signed 32 bit integer            */
typedef unsigned char   UB;             /* unsigned 8 bit integer           */
typedef unsigned short  UH;             /* unsigned 16 bit integer          */
typedef unsigned long   UW;             /* unsigned 32 bit integer          */
typedef char            VB;             /* variable data type (8 bit)       */
typedef short           VH;             /* variable data type (16 bit)      */
typedef long            VW;             /* variable data type (32 bit)      */
typedef void            *VP;            /* pointer to variable data type    */
typedef void            (*FP)();        /* program start address            */
typedef H               ID;             /* object ID (xxxid)                */
typedef H               HNO;            /* handler number                   */
typedef H               TPRI;           /* task priority                    */
typedef W               ER;             /* error code                       */
typedef int             FN;             /* function code                    */
typedef W               TMO;            /* time out                         */

/*----------------- data type related to task & handler -------------------*/
typedef void            TASK;           /* task                             */
typedef void            INTHDR;         /* interrupt handler                */
typedef void            TMRHDR;         /* timer handler                    */
typedef void            INIHDR;         /* initialization handler           */
typedef ER              SVCHDR;         /* extended SVC handler             */
```

```
/*---------------------- pointer to task & handler ------------------------*/
typedef TASK            (*TASKP)();     /* pointer to task                  */
typedef INTHDR          (*INTHDRP)();   /* pointer to interrupt handler     */
typedef TMRHDR          (*TMRHDRP)();   /* pointer to timer handler         */
typedef INIHDR          (*INIHDRP)();   /* pointer to initialize handler    */
typedef SVCHDR          (*SVCHDRP)();   /* pointer to extended SVC handler  */

/*----------------- data type related to SH architecture -------------------*/
typedef UW              SR;             /* SR                               */

/*****************************************************************************/
/*                                                                           */
/*                   fixed number & packet form (common)                     */
/*                                                                           */
/*****************************************************************************/
/*---------------------------------- whole ----------------------------------*/
#define NADR            (-1)            /* invalid address                  */
#define TRUE            1               /* true                             */
#define FALSE           0               /* false                            */

/*---------------------------------- tmout ----------------------------------*/
#define TMO_FEVR        (TMO)(-1)       /* forever wait                     */

/*---------------------------------- tskid ----------------------------------*/
#define TSK_SELF        (ID)0           /* self task                        */

/*---------------------------------- tskpri ---------------------------------*/
#define TPRI_INI        (TPRI)0         /* initial priority of starting task*/
#define TPRI_RUN        (TPRI)0         /* highest priority in running task */

/*---------------------------------- imask ----------------------------------*/
#define SR_IMS00        (SR)0x00000000  /* imask0                           */
#define SR_IMS01        (SR)0x00000001  /* imask1                           */
#define SR_IMS02        (SR)0x00000002  /* imask2                           */
#define SR_IMS03        (SR)0x00000003  /* imask3                           */
#define SR_IMS04        (SR)0x00000004  /* imask4                           */
#define SR_IMS05        (SR)0x00000005  /* imask5                           */
#define SR_IMS06        (SR)0x00000006  /* imask6                           */
#define SR_IMS07        (SR)0x00000007  /* imask7                           */
#define SR_IMS08        (SR)0x00000008  /* imask8                           */
#define SR_IMS09        (SR)0x00000009  /* imask9                           */
#define SR_IMS10        (SR)0x0000000a  /* imask10                          */
#define SR_IMS11        (SR)0x0000000b  /* imask11                          */
#define SR_IMS12        (SR)0x0000000c  /* imask12                          */
#define SR_IMS13        (SR)0x0000000d  /* imask13                          */
#define SR_IMS14        (SR)0x0000000e  /* imask14                          */
#define SR_IMS15        (SR)0x0000000f  /* imask15                          */

/*---------------------------------- sr -------------------------------------*/
#define MD_SET          (SR)0x40000000  /* set MD                           */

/*---------------------------------- tskstat --------------------------------*/
#define TTS_RDY         0x0010          /* READY                            */
#define TTS_WAI         0x0020          /* WAIT                             */
#define TTS_SUS         0x0040          /* SUSPEND                          */
#define TTS_WAS         0x0060          /* WAIT-SUSPEND                     */
#define TTS_DMT         0x0080          /* DORMANT                          */
#define TTS_STK         0x4000          /* STACK WAIT                       */
#define TTW_SLP         0x0100          /* wait (slp_tsk)                   */
#define TTW_WAI         0x0200          /* wait (wai_tsk)                   */
```

```
#define TTW_FLG         0x0400          /* wait (wai_flg)                   */
#define TTW_SEM         0x0800          /* wait (wai_sem)                   */
#define TTW_MBX         0x1000          /* wait (rcv_msg)                   */
#define TTW_MPL         0x2000          /* wait (get_blk)                   */

/*---------------------------------- wfmode ----------------------------------*/
#define TWF_ANDW        (UW)0x00000000  /* AND wait                         */
#define TWF_ORW         (UW)0x00000002  /* OR wait                          */
#define TWF_CLR         (UW)0x00000001  /* clear                            */

/*---------------------------- trace attribute ----------------------------*/
#define TATR_SVC        (UH)0x0001      /* call SVC                         */
#define TATR_RTN        (UH)0x0002      /* return SVC                       */
#define TATR_CONT       (UH)0x0003      /* continue task                    */
#define TATR_IDLE       (UH)0x0004      /* change idle                      */

/*--------------------------------- t_msg ---------------------------------*/
typedef struct  t_msg   {
            UW          msghead;        /* message head (OS management area)*/
            VB          msgcont[1];     /* message contents                 */
        } T_MSG;

/*------------------- t_cio (for sample console driver) -------------------*/
typedef struct  t_cio   {
            UW          msghead;        /* message head (OS management area)*/
            FN          fncd;           /* function code (0:input, 1:output)*/
            ID          mbxid;          /* mailbox id (for end information) */
            union
            {
                struct
                {
                    B   inbuf;          /* input buffer                     */
                    UB  siossr;         /* serial I/O status (SCI SSR)      */
                } ci;
                struct
                {
                    B   *outbufp;       /* pointer to output buffer         */
                } co;
            } cio;
        } T_CIO;

/*--------------------------------- t_tim ---------------------------------*/
typedef struct  t_tim   {
            H           utime;          /* current date/time (upper)        */
            UW          ltime;          /* current date/time (lower)        */
        } T_TIM;

/*--------------------------------- t_ver ---------------------------------*/
typedef struct  t_ver   {
            UH          maker;          /* maker                            */
            UH          id;             /* id                               */
            UH          spver;          /* specification version            */
            UH          prver;          /* production version               */
            UH          prno[4];        /* production number                */
            UH          cpu;            /* CPU information                  */
            UH          var;            /* variation                        */
        } T_VER;

/*----------------- t_trccb (trace buffer control block) -----------------*/
typedef struct  t_trccb {
```

```
            VP          tr_trbtop;      /* top address of trace buffer      */
            VP          tr_trbbtm;      /* bottom address of trace buffer   */
            VP          tr_trbins;      /* insert address of trace entry    */
            UW          tr_trbsts;      /* trace buffer status              */
        } T_TRCCB;

/*------------------ t_trcent (trace buffer entry block) -------------------*/
typedef struct t_trcent {
            UH          te_attr;        /* attribute                        */
            ID          te_tskid;       /* task id of call svc              */
            UW          te_ltime;       /* system time                      */
            W           te_event;       /* event       (r0 register)        */
            VW          te_par1;        /* parameter 1 (r4 register)        */
            VW          te_par2;        /* parameter 2 (r5 register)        */
            VW          te_par3;        /* parameter 3 (r6 register)        */
            VW          te_par4;        /* parameter 4 (r7 register)        */
            UW          te_pc;          /* pc register                      */
            UW          te_sr;          /* sr register                      */
        } T_TRCENT;

/*****************************************************************************/
/*                                                                           */
/*                              error code                                   */
/*                                                                           */
/*****************************************************************************/
#define E_OK            0x0             /* normal end                       */
#define E_TNOSPT        (-0x613)        /* timer no support                 */
#define E_RSFN          (-0x614)        /* reserved function code number    */
#define E_RSID          (-0x617)        /* reserved id number               */
#define E_PAR           (-0x721)        /* parameter error                  */
#define E_ILADR         (-0x722)        /* illegal address                  */
#define E_IDOVR         (-0x723)        /* ID over                          */
#define E_TPRI          (-0x726)        /* illegal task priority            */
#define E_ILTIME        (-0x727)        /* illegal time                     */
#define E_ILMSG         (-0x729)        /* illegal message type             */
#define E_IMS           (-0x72b)        /* illegal imask                    */
#define E_SELF          (-0x831)        /* tskid is self                    */
#define E_EXS           (-0x833)        /* object exist                     */
#define E_NOEXS         (-0x834)        /* object non existent              */
#define E_DMT           (-0x835)        /* task is dormant status           */
#define E_NODMT         (-0x836)        /* task isn't dormant status        */
#define E_NOSUS         (-0x838)        /* task isn't suspend status        */
#define E_ILBLK         (-0x83b)        /* illegal momory block             */
#define E_NOWAI         (-0x83e)        /* task isn't wait status           */
#define E_OBJ           (-0x83f)        /* object status error              */
#define E_CTX           (-0xa45)        /* context error                    */
#define E_QOVR          (-0xb49)        /* queuing over flow                */
#define E_TMOUT         (-0xd55)        /* time out                         */
#define E_RLWAI         (-0xd56)        /* wait status forced release       */
#define E_PLFAIL        (-0xe59)        /* polling fail                     */

/*****************************************************************************/
/*                                                                           */
/*                    error code of setup table check                        */
/*                                                                           */
/*****************************************************************************/
#define ES_KMSKLVL      0x01            /* illegal kernel mask level of     */
                                        /*                          interrupt */
#define ES_INSTNUM      0x02            /* illegal nest number of interrupt */
#define ES_MTSKID       0x03            /* illegal max task id              */
```

```
#define ES_MTSKPRI      0x04            /* illegal max task priority        */
#define ES_ITSKNUM      0x05            /* illegal number of initial task   */
#define ES_ITSKSP       0x06            /* illegal initial task stack       */
                                        /*                          pointer */
#define ES_ITSKID       0x07            /* illegal initial task id          */
#define ES_ITSKPRI      0x08            /* illegal initial task priority    */
#define ES_ITSKADR      0x09            /* illegal initial task start       */
                                        /*                          address */
#define ES_MFLGID       0x0a            /* illegal max event flag id        */
#define ES_MSEMID       0x0b            /* illegal max semaphore id         */
#define ES_MMBXID       0x0c            /* illegal max mail box id          */
#define ES_MMPLID       0x0d            /* illegal max memory pool id       */
#define ES_MPLADR       0x0e            /* illegal memory pool top address  */
#define ES_BLKSIZ       0x0f            /* illegal memory block size        */
#define ES_BLKCNT       0x10            /* illegal memory block count       */
#define ES_MPLARA       0x11            /* illegal memory pool area         */
#define ES_IHDRADR      0x12            /* illegal initial handler address  */
#define ES_MSVCFCD      0x13            /* illegal max function code of     */
                                        /*                      extended SVC */
#define ES_SVCADR       0x14            /* illegal extended SVC handler     */
                                        /*                          address */
#define ES_TRCADR       0x15            /* illegal trace buffer top address */
#define ES_TRCCNT       0x16            /* illegal trace count              */
#define ES_TRCARA       0x17            /* illegal trace buffer area        */

/*****************************************************************************/
/*                                                                           */
/*                          SVC function code                                */
/*                                                                           */
/*****************************************************************************/
#define TFN_CRE_TSK     0xffffffef
#define TFN_STA_TSK     0xfffffefe
#define TFN_ISTA_TSK    0xffffffff
#define TFN_DEL_TSK     0xfffffefd
#define TFN_EXT_TSK     0xfffffefc
#define TFN_EXD_TSK     0xfffffefb
#define TFN_TER_TSK     0xfffffefa
#define TFN_CHG_PRI     0xfffffef9
#define TFN_ICHG_PRI    0xfffffffe
#define TFN_ROT_RDQ     0xfffffef8
#define TFN_IROT_RDQ    0xfffffffd
#define TFN_REL_WAI     0xfffffef7
#define TFN_IREL_WAI    0xfffffffc
#define TFN_GET_TID     0xfffffef6
#define TFN_IGET_TID    0xfffffffb
#define TFN_TSK_STS     0xfffffef5
#define TFN_ITSK_STS    0xfffffffa
#define TFN_SUS_TSK     0xfffffef4
#define TFN_ISUS_TSK    0xfffffff9
#define TFN_RSM_TSK     0xfffffef3
#define TFN_IRSM_TSK    0xfffffff8
#define TFN_SLP_TSK     0xfffffef2
#define TFN_WAI_TSK     0xfffffef1
#define TFN_WUP_TSK     0xfffffef0
#define TFN_IWUP_TSK    0xfffffff7
#define TFN_CAN_WUP     0xfffffeef
#define TFN_ICAN_WUP    0xfffffff6
#define TFN_SET_FLG     0xfffffeee
#define TFN_ISET_FLG    0xfffffff5
#define TFN_CLR_FLG     0xfffffeed
```

```
#define TFN_ICLR_FLG    0xffffffff4
#define TFN_WAI_FLG     0xffffffec
#define TFN_POL_FLG     0xffffffeb
#define TFN_IPOL_FLG    0xffffffff3
#define TFN_FLG_STS     0xffffffea
#define TFN_IFLG_STS    0xffffffff2
#define TFN_SIG_SEM     0xffffffe9
#define TFN_ISIG_SEM    0xffffffff1
#define TFN_WAI_SEM     0xffffffe8
#define TFN_PREQ_SEM    0xffffffe7
#define TFN_IPREQ_SEM   0xffffffff0
#define TFN_SEM_STS     0xffffffe6
#define TFN_ISEM_STS    0xffffffef
#define TFN_SND_MSG     0xffffffe5
#define TFN_ISND_MSG    0xffffffee
#define TFN_RCV_MSG     0xffffffe4
#define TFN_PRCV_MSG    0xffffffe3
#define TFN_IPRCV_MSG   0xffffffed
#define TFN_MBX_STS     0xffffffe2
#define TFN_IMBX_STS    0xffffffec
#define TFN_CHG_IMS     0xffffffe1
#define TFN_ICHG_IMS    0xffffffeb
#define TFN_IMS_STS     0xffffffe0
#define TFN_IIMS_STS    0xffffffea
#define TFN_GET_BLK     0xffffffedf
#define TFN_PGET_BLK    0xffffffede
#define TFN_IPGET_BLK   0xffffffe9
#define TFN_REL_BLK     0xffffffedd
#define TFN_IREL_BLK    0xffffffe8
#define TFN_MPL_STS     0xffffffedc
#define TFN_IMPL_STS    0xffffffe7
#define TFN_SET_TIM     0xffffffedb
#define TFN_ISET_TIM    0xffffffe6
#define TFN_GET_TIM     0xffffffeda
#define TFN_IGET_TIM    0xffffffe5
#define TFN_GET_VER     0xffffffed9
#define TFN_IGET_VER    0xffffffe4

/*************************************************************************/
/*                                                                       */
/*                          prototype declaration                        */
/*                                                                       */
/*************************************************************************/
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/*                          Task management                              */
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
#define cre_tsk(tskid, stadr, itskpri) (ER)trapa_svc((int)63, \
        (int)TFN_CRE_TSK, (ID)tskid, (TASKP)stadr, (TPRI)itskpri)
#define sta_tsk(tskid) (ER)trapa_svc((int)63, \
        (int)TFN_STA_TSK, (ID)tskid)
#define ista_tsk(tskid) (ER)trapa_svc((int)62, \
        (int)TFN_ISTA_TSK, (ID)tskid)
#define del_tsk(tskid) (ER)trapa_svc((int)63, \
        (int)TFN_DEL_TSK, (ID)tskid)
#define ext_tsk() (void)trapa_svc((int)63, \
        (int)TFN_EXT_TSK)
#define exd_tsk() (void)trapa_svc((int)63, \
        (int)TFN_EXD_TSK)
#define ter_tsk(tskid) (ER)trapa_svc((int)63, \
        (int)TFN_TER_TSK, (ID)tskid)
```

```
#define chg_pri(tskid, tskpri) (ER)trapa_svc((int)63, \
        (int)TFN_CHG_PRI, (ID)tskid, (TPRI)tskpri)
#define ichg_pri(tskid, tskpri) (ER)trapa_svc((int)62, \
        (int)TFN_ICHG_PRI, (ID)tskid, (TPRI)tskpri)
#define rot_rdq(tskpri) (ER)trapa_svc((int)63, \
        (int)TFN_ROT_RDQ, (TPRI)tskpri)
#define irot_rdq(tskpri) (ER)trapa_svc((int)62, \
        (int)TFN_IROT_RDQ, (TPRI)tskpri)
#define rel_wai(tskid) (ER)trapa_svc((int)63, \
        (int)TFN_REL_WAI, (ID)tskid)
#define irel_wai(tskid) (ER)trapa_svc((int)62, \
        (int)TFN_IREL_WAI, (ID)tskid)
#define get_tid(p_tskid) (ER)trapa_svc((int)63, \
        (int)TFN_GET_TID, (ID *)p_tskid)
#define iget_tid(p_tskid) (ER)trapa_svc((int)62, \
        (int)TFN_IGET_TID, (ID *)p_tskid)
#define tsk_sts(p_tskstat, p_tskpri, tskid) (ER)trapa_svc((int)63, \
        (int)TFN_TSK_STS, (UH *)p_tskstat, (TPRI *)p_tskpri, (ID)tskid)
#define itsk_sts(p_tskstat, p_tskpri, tskid) (ER)trapa_svc((int)62, \
        (int)TFN_ITSK_STS, (UH *)p_tskstat, (TPRI *)p_tskpri, (ID)tskid)

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/*                      Task-dependent synchronization                         */
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
#define sus_tsk(tskid) (ER)trapa_svc((int)63, \
        (int)TFN_SUS_TSK, (ID)tskid)
#define isus_tsk(tskid) (ER)trapa_svc((int)62, \
        (int)TFN_ISUS_TSK, (ID)tskid)
#define rsm_tsk(tskid) (ER)trapa_svc((int)63, \
        (int)TFN_RSM_TSK, (ID)tskid)
#define irsm_tsk(tskid) (ER)trapa_svc((int)62, \
        (int)TFN_IRSM_TSK, (ID)tskid)
#define slp_tsk() (ER)trapa_svc((int)63, \
        (int)TFN_SLP_TSK)
#define wai_tsk(tmout) (ER)trapa_svc((int)63, \
        (int)TFN_WAI_TSK, (TMO)tmout)
#define wup_tsk(tskid) (ER)trapa_svc((int)63, \
        (int)TFN_WUP_TSK, (ID)tskid)
#define iwup_tsk(tskid) (ER)trapa_svc((int)62, \
        (int)TFN_IWUP_TSK, (ID)tskid)
#define can_wup(p_wupcnt, tskid) (ER)trapa_svc((int)63, \
        (int)TFN_CAN_WUP, (W *)p_wupcnt, (ID)tskid)
#define ican_wup(p_wupcnt, tskid) (ER)trapa_svc((int)62, \
        (int)TFN_ICAN_WUP, (W *)p_wupcnt, (ID)tskid)

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/*                    Synchronization and communication                        */
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
#define set_flg(flgid, setptn) (ER)trapa_svc((int)63, \
        (int)TFN_SET_FLG, (ID)flgid, (UW)setptn)
#define iset_flg(flgid, setptn) (ER)trapa_svc((int)62, \
        (int)TFN_ISET_FLG, (ID)flgid, (UW)setptn)
#define clr_flg(flgid, clrptn) (ER)trapa_svc((int)63, \
        (int)TFN_CLR_FLG, (ID)flgid, (UW)clrptn)
#define iclr_flg(flgid, clrptn) (ER)trapa_svc((int)62, \
        (int)TFN_ICLR_FLG, (ID)flgid, (UW)clrptn)
#define wai_flg(p_flgptn, flgid, waiptn, wfmode) (ER)trapa_svc((int)63, \
        (int)TFN_WAI_FLG, (UW *)p_flgptn, (ID)flgid, (UW)waiptn, (UW)wfmode)
#define pol_flg(p_flgptn, flgid, waiptn, wfmode) (ER)trapa_svc((int)63, \
        (int)TFN_POL_FLG, (UW *)p_flgptn, (ID)flgid, (UW)waiptn, (UW)wfmode)
```

```
#define ipol_flg(p_flgptn, flgid, waiptn, wfmode) (ER)trapa_svc((int)62, \
        (int)TFN_IPOL_FLG, (UW *)p_flgptn, (ID)flgid, (UW)waiptn, (UW)wfmode)
#define flg_sts(p_wtskid, p_flgptn, flgid) (ER)trapa_svc((int)63, \
        (int)TFN_FLG_STS, (ID *)p_wtskid, (UW *)p_flgptn, (ID)flgid)
#define iflg_sts(p_wtskid, p_flgptn, flgid) (ER)trapa_svc((int)62, \
        (int)TFN_IFLG_STS, (ID *)p_wtskid, (UW *)p_flgptn, (ID)flgid)
#define sig_sem(semid) (ER)trapa_svc((int)63, \
        (int)TFN_SIG_SEM, (ID)semid)
#define isig_sem(semid) (ER)trapa_svc((int)62, \
        (int)TFN_ISIG_SEM, (ID)semid)
#define wai_sem(semid) (ER)trapa_svc((int)63, \
        (int)TFN_WAI_SEM, (ID)semid)
#define preq_sem(semid) (ER)trapa_svc((int)63, \
        (int)TFN_PREQ_SEM, (ID)semid)
#define ipreq_sem(semid) (ER)trapa_svc((int)62, \
        (int)TFN_IPREQ_SEM, (ID)semid)
#define sem_sts(p_wtskid, p_semcnt, semid) (ER)trapa_svc((int)63, \
        (int)TFN_SEM_STS, (ID *)p_wtskid, (W *)p_semcnt, (ID)semid)
#define isem_sts(p_wtskid, p_semcnt, semid) (ER)trapa_svc((int)62, \
        (int)TFN_ISEM_STS, (ID *)p_wtskid, (W *)p_semcnt, (ID)semid)
#define snd_msg(mbxid, pk_msg) (ER)trapa_svc((int)63, \
        (int)TFN_SND_MSG, (ID)mbxid, (void *)pk_msg)
#define isnd_msg(mbxid, pk_msg) (ER)trapa_svc((int)62, \
        (int)TFN_ISND_MSG, (ID)mbxid, (void *)pk_msg)
#define rcv_msg(ppk_msg, mbxid) (ER)trapa_svc((int)63, \
        (int)TFN_RCV_MSG, (void **)ppk_msg, (ID)mbxid)
#define prcv_msg(ppk_msg, mbxid) (ER)trapa_svc((int)63, \
        (int)TFN_PRCV_MSG, (void **)ppk_msg, (ID)mbxid)
#define iprcv_msg(ppk_msg, mbxid) (ER)trapa_svc((int)62, \
        (int)TFN_IPRCV_MSG, (void **)ppk_msg, (ID)mbxid)
#define mbx_sts(p_wtskid, ppk_msg, mbxid) (ER)trapa_svc((int)63, \
        (int)TFN_MBX_STS, (ID *)p_wtskid, (void **)ppk_msg, (ID)mbxid)
#define imbx_sts(p_wtskid, ppk_msg, mbxid) (ER)trapa_svc((int)62, \
        (int)TFN_IMBX_STS, (ID *)p_wtskid, (void **)ppk_msg, (ID)mbxid)

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/*                          Interrupt management                               */
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
#define chg_ims(imask) (ER)trapa_svc((int)63, \
        (int)TFN_CHG_IMS, (SR)imask)
#define ichg_ims(imask) (ER)trapa_svc((int)62, \
        (int)TFN_ICHG_IMS, (SR)imask)
#define ims_sts(p_imask) (ER)trapa_svc((int)63, \
        (int)TFN_IMS_STS, (SR *)p_imask)
#define iims_sts(p_imask) (ER)trapa_svc((int)62, \
        (int)TFN_IIMS_STS, (SR *)p_imask)

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/*                          Memorypool management                              */
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
#define get_blk(p_blk, mplid) (ER)trapa_svc((int)63, \
        (int)TFN_GET_BLK, (VP *)p_blk, (ID)mplid)
#define pget_blk(p_blk, mplid) (ER)trapa_svc((int)63, \
        (int)TFN_PGET_BLK, (VP *)p_blk, (ID)mplid)
#define ipget_blk(p_blk, mplid) (ER)trapa_svc((int)62, \
        (int)TFN_IPGET_BLK, (VP *)p_blk, (ID)mplid)
#define rel_blk(mplid, blk) (ER)trapa_svc((int)63, \
        (int)TFN_REL_BLK, (ID)mplid, (VP)blk)
#define irel_blk(mplid, blk) (ER)trapa_svc((int)62, \
        (int)TFN_IREL_BLK, (ID)mplid, (VP)blk)
```

```
#define mpl_sts(p_wtskid, p_frbcnt, mplid) (ER)trapa_svc((int)63, \
        (int)TFN_MPL_STS, (ID *)p_wtskid, (W *)p_frbcnt, (ID)mplid)
#define impl_sts(p_wtskid, p_frbcnt, mplid) (ER)trapa_svc((int)62, \
        (int)TFN_IMPL_STS, (ID *)p_wtskid, (W *)p_frbcnt, (ID)mplid)

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/*                              Time management                                */
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
#define set_tim(pk_tim) (ER)trapa_svc((int)63, \
        (int)TFN_SET_TIM, (T_TIM *)pk_tim)
#define iset_tim(pk_tim) (ER)trapa_svc((int)62, \
        (int)TFN_ISET_TIM, (T_TIM *)pk_tim)
#define get_tim(pk_tim) (ER)trapa_svc((int)63, \
        (int)TFN_GET_TIM, (T_TIM *)pk_tim)
#define iget_tim(pk_tim) (ER)trapa_svc((int)62, \
        (int)TFN_IGET_TIM, (T_TIM *)pk_tim)
#define sys_clk() trapa(60)

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/*                             System management                               */
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
#define get_ver(pk_ver) (ER)trapa_svc((int)63, \
        (int)TFN_GET_VER, (T_VER *)pk_ver)
#define iget_ver(pk_ver) (ER)trapa_svc((int)62, \
        (int)TFN_IGET_VER, (T_VER *)pk_ver)

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/*                                Extended SVC                                 */
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
#define sys_cal(fncd, para1, para2, para3, para4) (ER)trapa_svc((int)63, \
        (int)fncd, (VW)para1, (VW)para2, (VW)para3, (VW)para4)
```

F．２　アセンブリ言語用ヘッダファイル

```
;************************************************************************;
;*                                                                      *;
;*                                                                      *;
;*                                                                      *;
;*              HI-SH77 header file (ASM language) ( Ver. 1.0 )         *;
;*                                                                      *;
;*                                                                      *;
;*              Copyright (c)  Hitachi, Ltd. 1995.                      *;
;*              Licensed Material of Hitachi, Ltd.                      *;
;*                                                                      *;
;*              HI-SH77(HS0770ITCN1SM) V1.0                             *;
;*                                                                      *;
;*                                                                      *;
;************************************************************************;
;************************************************************************;
;*SPECIFICATIONS ;                                                      *;
;*  FILE      = itron.def ;                                             *;
;*  DATE      = 95/02/20 ;                                              *;
;*  AUTHOR    = Hitachi, Ltd. ;                                         *;
;*  FUNCTION  = HI-SH77 header file (ASM language) ;                    *;
;*  ATTRIBUTE = PUBLIC ;                                                *;
;*  HISTORY   = V1.0 ;                                                  *;
;*END OF SPECIFICATIONS ;                                               *;
;************************************************************************;
;************************************************************************;
;*                                                                      *;
;*                              function code                           *;
;*                                                                      *;
;************************************************************************;
;*---------------------- task management function --------------------------*;
CRE_TSK         .equ    -h'101          ;* create task                   *;
STA_TSK         .equ    -h'102          ;* start task                    *;
DEL_TSK         .equ    -h'103          ;* delete task                   *;
EXT_TSK         .equ    -h'104          ;* exit task                     *;
EXD_TSK         .equ    -h'105          ;* exit and delete task          *;
TER_TSK         .equ    -h'106          ;* terminate task                *;
CHG_PRI         .equ    -h'107          ;* change task priority          *;
ROT_RDQ         .equ    -h'108          ;* rotate ready queue            *;
REL_WAI         .equ    -h'109          ;* release wait                  *;
GET_TID         .equ    -h'10a          ;* get task identifier           *;
TSK_STS         .equ    -h'10b          ;* get task status               *;
;
ISTA_TSK        .equ    -h'1            ;* interrupt start task         *;
ICHG_PRI        .equ    -h'2            ;* interrupt change task priority *;
IROT_RDQ        .equ    -h'3            ;* interrupt rotate ready queue  *;
IREL_WAI        .equ    -h'4            ;* interrupt release wait        *;
IGET_TID        .equ    -h'5            ;* interrupt get task identifier *;
ITSK_STS        .equ    -h'6            ;* interrupt get task status     *;

;*-------------- task synchronization management function ----------------*;
SUS_TSK         .equ    -h'10c          ;* suspend task                  *;
RSM_TSK         .equ    -h'10d          ;* resume task                   *;
SLP_TSK         .equ    -h'10e          ;* sleep task                    *;
WAI_TSK         .equ    -h'10f          ;* wait task                     *;
WUP_TSK         .equ    -h'110          ;* wakeup task                   *;
CAN_WUP         .equ    -h'111          ;* cancel wakeup task            *;
;
ISUS_TSK        .equ    -h'7            ;* interrupt suspend task       *;
IRSM_TSK        .equ    -h'8            ;* interrupt resume task         *;
```

```
IWUP_TSK        .equ    -h'9            ;* interrupt wakeup task            *;
ICAN_WUP        .equ    -h'a            ;* interrupt cancel wakeup task      *;

;*-------------- synchronization and communication function ----------------*;
SET_FLG         .equ    -h'112          ;* set event flag                    *;
CLR_FLG         .equ    -h'113          ;* clear event flag                  *;
WAI_FLG         .equ    -h'114          ;* wait event flag                   *;
POL_FLG         .equ    -h'115          ;* poll event flag                   *;
FLG_STS         .equ    -h'116          ;* get event flag status             *;
SIG_SEM         .equ    -h'117          ;* signal semaphore                  *;
WAI_SEM         .equ    -h'118          ;* wait on semaphore                 *;
PREQ_SEM        .equ    -h'119          ;* poll and request semaphore        *;
SEM_STS         .equ    -h'11a          ;* get semaphore status              *;
SND_MSG         .equ    -h'11b          ;* send message to mailbox           *;
RCV_MSG         .equ    -h'11c          ;* receive message from mailbox      *;
PRCV_MSG        .equ    -h'11d          ;* poll and receive message from     *;
                                        ;*                                mailbox*;
MBX_STS         .equ    -h'11e          ;* get mailbox status                *;
;
ISET_FLG        .equ    -h'b            ;* interrupt set event flag          *;
ICLR_FLG        .equ    -h'c            ;* interrupt clear event flag        *;
IPOL_FLG        .equ    -h'd            ;* interrupt poll event flag         *;
IFLG_STS        .equ    -h'e            ;* interrupt get event flag status   *;
ISIG_SEM        .equ    -h'f            ;* interrupt signal semaphore        *;
IPREQ_SEM       .equ    -h'10           ;* interrupt poll and request        *;
                                        ;*                          semaphore   *;
ISEM_STS        .equ    -h'11           ;* interrupt get semaphore status    *;
ISND_MSG        .equ    -h'12           ;* interrupt send message to mailbox*;
IPRCV_MSG       .equ    -h'13           ;* interrupt poll and receive        *;
                                        ;*              message from mailbox*;
IMBX_STS        .equ    -h'14           ;* interrupt get mailbox status      *;

;*-------------------- interrupt management function ----------------------*;
CHG_IMS         .equ    -h'11f          ;* change interrupt mask level       *;
IMS_STS         .equ    -h'120          ;* get interrupt mask level status   *;
;
ICHG_IMS        .equ    -h'15           ;* interrupt change interrupt mask   *;
                                        ;*                              level   *;
IIMS_STS        .equ    -h'16           ;* interrupt get interrupt mask      *;
                                        ;*                          level status*;

;*------------------ memory pool management function ----------------------*;
GET_BLK         .equ    -h'121          ;* get memory block                  *;
PGET_BLK        .equ    -h'122          ;* poll and get memory block         *;
REL_BLK         .equ    -h'123          ;* release memory block              *;
MPL_STS         .equ    -h'124          ;* get memory pool status            *;
;
IPGET_BLK       .equ    -h'17           ;* interrupt poll and get memory     *;
                                        ;*                                block   *;
IREL_BLK        .equ    -h'18           ;* interrupt memory block            *;
IMPL_STS        .equ    -h'19           ;* interrupt get memory pool status *;

;*--------------------- timer management function --------------------------*;
SET_TIM         .equ    -h'125          ;* set time                          *;
GET_TIM         .equ    -h'126          ;* get time                          *;
;
ISET_TIM        .equ    -h'1a           ;* interrupt set time                *;
IGET_TIM        .equ    -h'1b           ;* interrupt get time                *;

;*------------------- system management function -------------------------*;
```

```
GET_VER         .equ    -h'127          ;* get version no.                  *;
;
IGET_VER        .equ    -h'1c           ;* interrupt get version no.          *;

;*************************************************************************;
;*                                                                       *;
;*                              common symbols                           *;
;*                                                                       *;
;*************************************************************************;
;*------------------------------ data size ------------------------------*;
D_BYTE          .equ    h'1             ;* size of byte                       *;
D_HALF          .equ    h'2             ;* size of half word                  *;
D_WORD          .equ    h'4             ;* size of word                       *;
D_DWORD         .equ    h'8             ;* size of double word                *;

;*-------------------------------- whole --------------------------------*;
NADR            .equ    -d'1            ;* invalid address                   *;
TRUE            .equ    d'1             ;* true                               *;
FALSE           .equ    d'0             ;* false                              *;

;*-------------------------------- tmout --------------------------------*;
TMO_FEVR        .equ    -d'1            ;* forever wait                       *;

;*-------------------------------- tskid --------------------------------*;
TSK_SELF        .equ    d'0             ;* self task                          *;

;*-------------------------------- tskpri -------------------------------*;
TPRI_INI        .equ    d'0             ;* initial priority of starting task*;
TPRI_RUN        .equ    d'0             ;* highest priority in running task *;

;*-------------------------------- imask --------------------------------*;
SR_IMS00        .equ    h'00000000      ;* imask0                             *;
SR_IMS01        .equ    h'00000001      ;* imask1                             *;
SR_IMS02        .equ    h'00000002      ;* imask2                             *;
SR_IMS03        .equ    h'00000003      ;* imask3                             *;
SR_IMS04        .equ    h'00000004      ;* imask4                             *;
SR_IMS05        .equ    h'00000005      ;* imask5                             *;
SR_IMS06        .equ    h'00000006      ;* imask6                             *;
SR_IMS07        .equ    h'00000007      ;* imask7                             *;
SR_IMS08        .equ    h'00000008      ;* imask8                             *;
SR_IMS09        .equ    h'00000009      ;* imask9                             *;
SR_IMS10        .equ    h'0000000a      ;* imask10                            *;
SR_IMS11        .equ    h'0000000b      ;* imask11                            *;
SR_IMS12        .equ    h'0000000c      ;* imask12                            *;
SR_IMS13        .equ    h'0000000d      ;* imask13                            *;
SR_IMS14        .equ    h'0000000e      ;* imask14                            *;
SR_IMS15        .equ    h'0000000f      ;* imask15                            *;

;*------------------------------- tskstat -------------------------------*;
TTS_RDY         .equ    h'0010          ;* READY                              *;
TTS_WAI         .equ    h'0020          ;* WAIT                               *;
TTS_SUS         .equ    h'0040          ;* SUSPEND                            *;
TTS_WAS         .equ    h'0060          ;* WAIT-SUSPEND                       *;
TTS_DMT         .equ    h'0080          ;* DORMANT                            *;
TTS_STK         .equ    h'4000          ;* STACK WAIT                         *;
TTW_SLP         .equ    h'0100          ;* wait (slp_tsk)                     *;
TTW_WAI         .equ    h'0200          ;* wait (wai_tsk)                     *;
TTW_FLG         .equ    h'0400          ;* wait (wai_flg)                     *;
TTW_SEM         .equ    h'0800          ;* wait (wai_sem)                     *;
TTW_MBX         .equ    h'1000          ;* wait (rcv_msg)                     *;
```

```
TTW_MPL         .equ    h'2000          ;* wait (get_blk)                   *;

;*---------------------------------- wfmode ----------------------------------*;
TWF_ANDW        .equ    h'00000000      ;* AND wait                           *;
TWF_ORW         .equ    h'00000002      ;* OR wait                            *;
TWF_CLR         .equ    h'00000001      ;* clear                              *;

;*---------------------------- trace attribute -----------------------------*;
TATR_SVC        .equ    h'0001          ;* call SVC                           *;
TATR_RTN        .equ    h'0002          ;* return SVC                         *;
TATR_CONT       .equ    h'0003          ;* continue task                      *;
TATR_IDLE       .equ    h'0004          ;* change idle                        *;

;*---------------------------------- t_msg -----------------------------------*;
MSGHEAD         .equ    h'0             ;* message head (OS management area)*;
MSGCONT         .equ    MSGHEAD + D_WORD;* message contents                  *;

;*---------------------------------- t_tim -----------------------------------*;
UTIME           .equ    h'0             ;* current date;time (upper)         *;
LTIME           .equ    UTIME + D_WORD  ;* current date;time (lower)         *;

;*---------------------------------- t_ver -----------------------------------*;
MAKER           .equ    h'0             ;* maker                             *;
ID              .equ    MAKER + D_HALF  ;* id                                *;
SPVER           .equ    ID + D_HALF     ;* specification version             *;
PRVER           .equ    SPVER + D_HALF  ;* production version                 *;
PRNO            .equ    PRVER + D_HALF  ;* production number                  *;
CPU             .equ    PRNO + D_DWORD  ;* CPU information                    *;
VAR             .equ    CPU + D_HALF    ;* variation                         *;

;*------------------ t_trccb (trace buffer control block) -------------------*;
tr_trbtop       .equ    h'00            ;* top address of trace buffer       *;
tr_trbbtm       .equ    h'04            ;* bottom address of trace buffer    *;
tr_trbins       .equ    h'08            ;* insert address of trace entry     *;
tr_trbsts       .equ    h'0c            ;* trace buffer status               *;
sz_trccb        .equ    h'10            ;* size of t_trccb                   *;

;*------------------ t_trcent (trace buffer entry block) -------------------*;
te_attr         .equ    h'00            ;* attribute                         *;
te_tskid        .equ    h'02            ;* task id of call svc               *;
te_ltime        .equ    h'04            ;* system time                       *;
te_event        .equ    h'08            ;* event       (r0 register)        *;
te_par1         .equ    h'0c            ;* parameter 1 (r4 register)         *;
te_par2         .equ    h'10            ;* parameter 2 (r5 register)         *;
te_par3         .equ    h'14            ;* parameter 3 (r6 register)         *;
te_par4         .equ    h'18            ;* parameter 4 (r7 register)         *;
te_pc           .equ    h'1c            ;* pc register                       *;
te_sr           .equ    h'20            ;* sr register                       *;
sz_trcent       .equ    h'24            ;* size of t_trcent                  *;

;*************************************************************************;
;*                                                                       *;
;*                              error code                               *;
;*                                                                       *;
;*************************************************************************;
E_OK            .equ    h'0             ;* normal end                        *;
E_TNOSPT        .equ    -h'613          ;* timer no support                  *;
E_RSFN          .equ    -h'614          ;* reserved function code number     *;
E_RSID          .equ    -h'617          ;* reserved id number                *;
E_PAR           .equ    -h'721          ;* parameter error                   *;
```

```
E_ILADR         .equ    -h'722          ;* illegal address                  *;
E_IDOVR         .equ    -h'723          ;* ID over                            *;
E_TPRI          .equ    -h'726          ;* illegal task priority              *;
E_ILTIME        .equ    -h'727          ;* illegal time                       *;
E_ILMSG         .equ    -h'729          ;* illegal message type               *;
E_IMS           .equ    -h'72b          ;* illegal imask                      *;
E_SELF          .equ    -h'831          ;* tskid is self                      *;
E_EXS           .equ    -h'833          ;* object exist                       *;
E_NOEXS         .equ    -h'834          ;* object non existent                *;
E_DMT           .equ    -h'835          ;* task is dormant status             *;
E_NODMT         .equ    -h'836          ;* task isn't dormant status          *;
E_NOSUS         .equ    -h'838          ;* task isn't suspend status          *;
E_ILBLK         .equ    -h'83b          ;* illegal momory block               *;
E_NOWAI         .equ    -h'83e          ;* task isn't wait status             *;
E_OBJ           .equ    -h'83f          ;* object status error               *;
E_CTX           .equ    -h'a45          ;* context error                      *;
E_QOVR          .equ    -h'b49          ;* queuing over flow                  *;
E_TMOUT         .equ    -h'd55          ;* time out                           *;
E_RLWAI         .equ    -h'd56          ;* wait status forced release         *;
E_PLFAIL        .equ    -h'e59          ;* polling fail                       *;

;************************************************************************;
;*                                                                      *;
;*                  error code of setup table check                     *;
;*                                                                      *;
;************************************************************************;
ES_KMSKLVL      .equ    h'01            ;* illegal kernel mask level of      *;
                                        ;*                          interrupt *;
ES_INSTNUM      .equ    h'02            ;* illegal nest number of interrupt   *;
ES_MTSKID       .equ    h'03            ;* illegal max task id                *;
ES_MTSKPRI      .equ    h'04            ;* illegal max task priority          *;
ES_ITSKNUM      .equ    h'05            ;* illegal number of initial task     *;
ES_ITSKSP       .equ    h'06            ;* illegal initial task stack         *;
                                        ;*                             pointer *;
ES_ITSKID       .equ    h'07            ;* illegal initial task id            *;
ES_ITSKPRI      .equ    h'08            ;* illegal initial task priority      *;
ES_ITSKADR      .equ    h'09            ;* illegal initial task start         *;
                                        ;*                             address *;
ES_MFLGID       .equ    h'0a            ;* illegal max event flag id          *;
ES_MSEMID       .equ    h'0b            ;* illegal max semaphore id           *;
ES_MMBXID       .equ    h'0c            ;* illegal max mail box id            *;
ES_MMPLID       .equ    h'0d            ;* illegal max memory pool id         *;
ES_MPLADR       .equ    h'0e            ;* illegal memory pool top address    *;
ES_BLKSIZ       .equ    h'0f            ;* illegal memory block size          *;
ES_BLKCNT       .equ    h'10            ;* illegal memory block count         *;
ES_MPLARA       .equ    h'11            ;* illegal memory pool area           *;
ES_IHDRADR      .equ    h'12            ;* illegal initial handler address    *;
ES_MSVCFCD      .equ    h'13            ;* illegal max function code of       *;
                                        ;*                          extended SVC *;
ES_SVCADR       .equ    h'14            ;* illegal extended SVC handler       *;
                                        ;*                             address *;
ES_TRCADR       .equ    h'15            ;* illegal trace buffer top address  *;
ES_TRCCNT       .equ    h'16            ;* illegal trace count                *;
ES_TRCARA       .equ    h'17            ;* illegal trace buffer area          *;
```

付録Ｇ．ＡＳＣＩＩコード表

Ｇ．１　ＡＳＣＩＩコード表

上位4ビット / 下位4ビット	0	1	2	3	4	5	6	7
0	NULL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	＼	l	¦
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

Ｈ．索　引

Ｈ．１　五十音順・索引

五十音順・索引

ア　行

アセンブラインタフェース ･････････････････････････････････ 3-7　C-3
イベント ･･･ 2-1　2-5　2-49
イベント属性 ･･･ 2-52
イベントフラグ ･･･ 2-17
イベントフラグID ･･･ 2-17
エラーコード ･･･ 3-10　E-1
エンディアン ･･･ D-1

カ　行

カーネル ･･･ 1-2　2-1
カーネルアイドリング状態 ･････････････････････････････････ 2-48
　　カーネル割込みマスクレベル ･･･････････････････････････ 2-16　2-31　4-5
拡張SVC ･･ 2-2　2-40　3-1　4-7
　　拡張SVCの機能コード ･････････････････････････････････ 2-41
　　拡張SVCハンドラ ･････････････････････････････････････ 2-40　4-6
過渡的な状態 ･･･ 2-3
起動要求の保留 ･･･ 2-9
機能コード（拡張SVCの） ･････････････････････････････････ 2-41
キャッシュ ･･･ D-2
休止状態 ･･･ 2-6
強制待ち状態 ･･･ 2-6
共有スタック ･･･ 2-9
　　共有スタック二重待ち状態 ･････････････････････････････ 2-10
　　共有スタック待ち状態 ･････････････････････････････････ 2-9
クリア指定 ･･･ 2-19
コンソールドライバ ･･･････････････････････････････････････ 1-2　A-1

サ　行

時間管理 ･･･ 2-2　3-1
資源 ･･･ 2-15　2-20
事象 ･･･ 2-1　2-5　2-17
システム異常終了処理 ･････････････････････････････････････ 2-47　4-10
　　システム異常終了処理エラーコード ･････････････････････ 2-47　E-2
システム管理 ･･･ 2-2　3-1
システム起動処理 ･･･ 2-42　2-44
システムクロック ･･･ 2-36
システムコール ･･･ 2-1　3-1
システム初期化ハンドラ ･･･････････････････････････････････ 2-46　4-9

実行可能状態 ・・ 2-6
実行状態 ・・・ 2-6
初期タスク優先度 ・・・ 2-8
シリアルコミュニケーションインタフェース ・・・・・・・・・・・・・・・・・・・・・・・・・・ 1-2　A-1
スタック切り換え指定 ・・ 4-3
セットアップテーブル ・・ 1-2
　　セットアップテーブルエラーコード ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ E-3
　　セットアップテーブルチェック ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ 2-44
セマフォ ・・ 2-20
　　セマフォカウンタ ・・・ 2-20
　　セマフォID ・・ 2-20

タ　行

タイマ
　　タイマ初期化ルーチン ・・・ 2-37　B-2
　　タイマドライバ ・・・ 1-2　2-37　B-1
　　タイマ割込みハンドラ ・・・ 2-38　4-4　B-2
タイマユニット ・・・ 1-2　B-1
タスク ・・ 2-1　2-5　4-2
　　タスク開始アドレス ・・・ 2-8
　　タスク管理 ・・ 2-1　3-1
　　タスク独立部実行中 ・・・ 2-3
　　タスク部実行中 ・・ 2-3
　　タスク付属同期管理 ・・・ 2-1　3-1
　　タスク部用システムコール ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ 2-3
　　タスク優先度 ・・ 2-8　2-12
　　タスクID ・・ 2-5　2-8
データ型 ・・ 3-5
同期／通信 ・・ 2-1　3-1
トラップ番号 ・・ 3-8　4-3
トラップ命令リターン指定 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ 4-3
トラップベクタテーブル ・・ 2-32
トレース ・・ 2-49
　　トレースエントリ ・・・ 2-52
　　トレースバッファ管理テーブル ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ 2-50

ナ　行

二重待ち状態 ・・ 2-6　2-10

ハ　行

ハードウェアタイマ ・・ 2-36　B-4
パケット ・・ 3-8
パラメータ
　　パラメータチェック ・・・ 1-2　3-10　E-1
　　パラメータのデータ型とサイズ ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ 3-5

パラメータ名称 ………………………………………………3-4
ハンドラ ………………………………………………2-1
非タスク部
非タスク部実行中 ………………………………………………2-3
非タスク部専用システムコール ………………………………………………2-4
非タスク部用システムコール ………………………………………………2-3
プリプロセッサ制御文 ………………………………………………4-3
プログラム配置 ………………………………………………D-3
ベクタ
ベクタテーブル ………………………………………………2-29　2-32　2-38
ベクタベースレジスタ ………………………………………………2-8　2-29　2-38　2-42
リセットベクタ ………………………………………………2-42

マ　行

待ち状態 ………………………………………………2-6　2-9
未定義割込み ………………………………………………2-30
未登録状態 ………………………………………………2-6
メイルボックス ………………………………………………2-23
メイルボックスID ………………………………………………2-23
メッセージ ………………………………………………2-23
メッセージフォーマット ………………………………………………2-26
メモリプール ………………………………………………2-33
メモリプール管理 ………………………………………………2-1　3-1
メモリプールID ………………………………………………2-33
メモリブロック ………………………………………………2-15　2-33
モード ………………………………………………D-1

ラ　行

ラウンドロビンスケジューリング ………………………………………………2-13
リセット ………………………………………………2-42
例外 ………………………………………………2-32　4-11
例外マスク ………………………………………………D-2
例外サービスルーチン ………………………………………………D-2
レジスタバンク ………………………………………………D-2
レディキュー ………………………………………………2-12

ワ　行

割込み ………………………………………………2-27
カーネル割込みマスクレベル ………………………………………………2-16　2-31　4-5
未定義割込み ………………………………………………2-30
割込み管理 ………………………………………………2-1　3-1
割込み仕様 ………………………………………………4-3
割込みハンドラ ………………………………………………2-28　4-3
割込みベクタ ………………………………………………2-29　2-38
割込みマスク ………………………………………………2-16　2-30　D-2

Ｈ．２　アルファベット順・索引

A

AND待ち ……… 2-19

C

C言語インタフェース ……… 3-4　C-1
can_wup ……… 3-11
chg_ims ……… 2-3　2-16　2-30　3-13
chg_pri ……… 3-15
clr_flg ……… 2-19　3-17
CONT属性 ……… 2-52
cre_tsk ……… 2-6　2-8　3-19

D

del_tsk ……… 2-6　3-21
DORMANT状態 ……… 2-6

E

exd_tsk ……… 2-6　3-23　4-2
ext_tsk ……… 2-6　3-24　4-2

F

flg_sts ……… 2-19　3-25

G

get_blk ……… 2-34　3-27
get_tid ……… 3-29
get_tim ……… 2-39　3-30
get_ver ……… 3-32

H

hi_intdwn ……… 2-30
hi_sysdwn ……… 2-47　4-10

I

ican_wup ……… 3-36
ichg_ims ……… 3-38
ichg_pri ……… 3-40
iclr_flg ……… 3-42
IDLE属性 ……… 2-52
iflg_sts ……… 3-44
iget_tid ……… 3-46

iget_tim ····· 3-47
iget_ver ····· 3-49
iims_sts ····· 3-53
imbx_sts ····· 3-54
impl_sts ····· 3-56
ims_sts ····· 2-30 3-58
INIHDR ····· 4-9
INTHDR ····· 4-4
ipol_flg ····· 3-61
iprcv_msg ····· 3-63
ipreq_sem ····· 3-65
irel_blk ····· 3-67
irel_wai ····· 3-69
irot_rdq ····· 2-13 3-71
irsm_tsk ····· 2-6 3-73
isem_sts ····· 3-75
iset_flg ····· 2-18 3-77
iset_tim ····· 3-79
isig_sem ····· 2-22 3-81
isnd_msg ····· 2-24 3-83
ista_tsk ····· 2-6 2-8 3-85
isus_tsk ····· 2-6 3-87
itron.def ····· F-7
itron.h ····· 4-1 F-1
itsk_sts ····· 3-89
iwup_tsk ····· 3-91

M

MCU初期化 ····· 2-42 4-8
mbx_sts ····· 2-25 3-93
MMU ····· D-3
mpl_sts ····· 2-35 3-95

N

NMI ····· 2-31
NON-EXISTENT状態 ····· 2-6

O

OR待ち ····· 2-19
OS
OS拡張部実行中 ····· 2-4
OS管理領域 ····· 2-26

P

P命令 ····· 2-21
pget_blk ····· 2-34 3-97
pol_flg ····· 2-19 3-99

pragma ····· 4-3
prcv_msg ····· 2-25 3-101
preq_sem ····· 2-21 3-103

R

rcv_msg ····· 2-25 3-105
READY状態 ····· 2-6
rel_blk ····· 2-35 3-107
rel_wai ····· 3-109
ret_exc ····· 2-32 3-111 4-1
ret_int ····· 2-3 2-28 3-112 4-1
rot_rdq ····· 3-114
rsm_tsk ····· 2-6 3-116
RTE ····· 2-31 2-32 4-5
RTN属性 ····· 2-52
RUN状態 ····· 2-6

S

SCI ····· 1-2 A-1
sem_sts ····· 2-22 3-118
set_flg ····· 2-18 3-120
set_imask ····· 2-30
set_tim ····· 2-39 3-122
sig_sem ····· 2-22 3-124
slp_tsk ····· 3-126
snd_msg ····· 2-24 3-127
SPC ····· D-2
SRレジスタ状態 ····· D-1
SSR ····· D-2
sta_tsk ····· 2-6 2-8 3-129
SUSPEND状態 ····· 2-6
sus_tsk ····· 2-6 3-131
SVC属性 ····· 2-52
SVCHDR ····· 4-6
sys_cal ····· 3-133 4-7
sys_clk ····· 2-38 3-135 4-1 4-4

T

TASK ····· 4-2
ter_tsk ····· 3-136
TMRHDR ····· 4-4
TMU ····· 1-2 B-1
TRAPA ····· 3-7 4-3
tsk_sts ····· 3-138

V

V命令 ····· 2-21
VBR ····· 2-8 2-29 2-38 2-42

W

wai_flg ····· 2-19 3-140
wai_sem ····· 2-21 3-143
WAIT状態 ····· 2-6
wai_tsk ····· 3-145
WAIT-SUSPEND状態 ····· 2-6
wup_tsk ····· 3-147

_

0Hrs _knl ····· 2-43 4-8
_hi_sysdwn ····· 2-47
_ _0Hrs_ _knl ····· 2-43

HI-SH77
ユーザーズマニュアル

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

ADJ-702-199