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

HI2000/3

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

HS0200ITZE1SJ

ルネサス エレクトロニクス
www.renesas.com

Rev.2.00　2001.01

μITRON は、Micro Industrial TRON の略称です。TRON は The Realtime Operating System Nucleus の略称です。

Microsoft® Windows®95 operating system,Microsoft® Windows®98 operating system,Microsoft® Windows NT® operating system は、米国 Microsoft Corp.の米国およびその他の国における登録商標です。

その他、本書で登場するシステム名、製品名は各社の登録商標または商標です。

本マニュアルは Windows®95,Windows®98,WindowsNT®に対応するように書かれています。

はじめに

本マニュアルでは、μITRON 3.0 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)である HI2000/3 の使用方法について説明します。

HI2000/3 をご使用になる前に本マニュアルを良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。

〈マニュアルの構成〉

本マニュアルは、以下に示す 8 つの章と付録から構成されています。

- 第 1 章では、HI2000/3 の概説を記載しています。
- 第 2 章では、HI2000/3 カーネルの機能概略を説明しています。ユーザシステムの機能設計時に利用してください。
- 第 3 章では、HI2000/3 カーネルの詳細機能、および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時、コーディング時に利用してください。
- 第 4 章では、HI2000/3 デバッギングエクステンション(DX)の機能概略、および操作方法を説明しています。
- 第 5 章では、HI2000/3 システム構築に必要な各種ハンドラ、ルーチンの作成方法、および登録方法について説明しています。
- 第 6 章では、HI2000/3 システム構築に必須であるセットアップテーブルの作成方法について説明しています。
- 第 7 章では、HI2000/3 システム構築に必須である割込みベクタテーブルの作成方法について説明しています。
- 第 8 章では、ロードモジュール生成方法について説明しています。
- 付録では、ユーザおよびカーネル環境で使用する作業領域の算出方法、コンパイラ、アセンブラオプションの説明、タイマドライバの例題、エラーコード一覧表、およびシステムコール機能コード一覧表を記載しています。

〈関連マニュアル〉

本カーネルに関連する以下の製品のマニュアルも、あわせて参照してください。

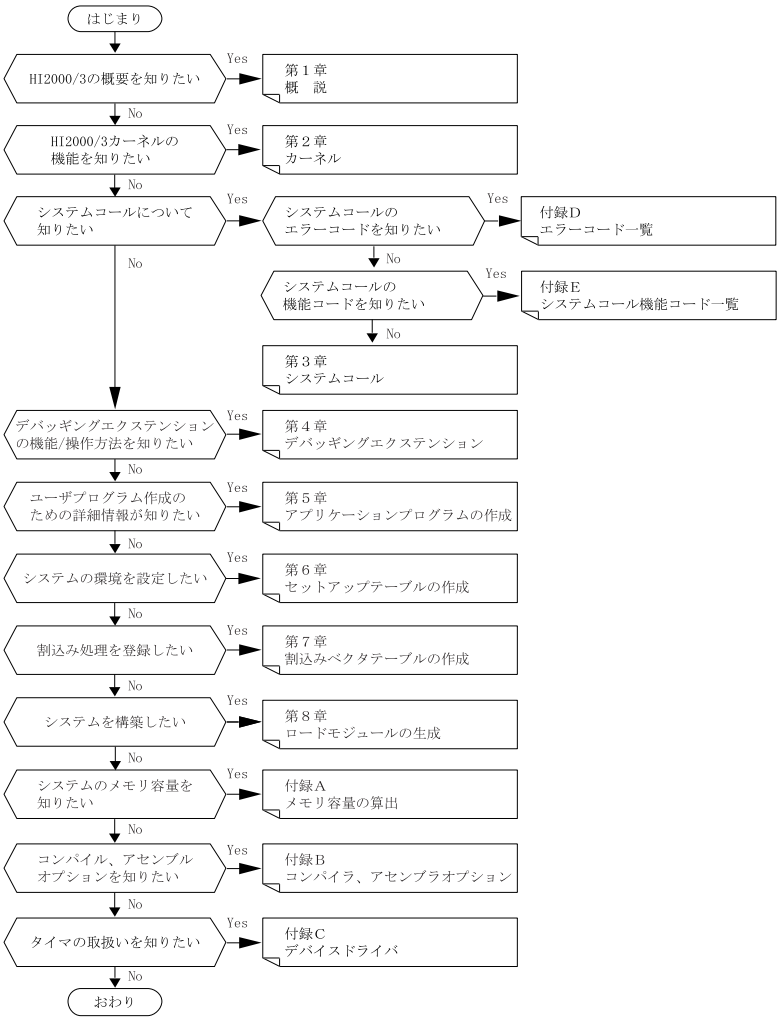
- 製品添付のリリースノート
- H8S，H8/300 シリーズ C/C++ コンパイラ ユーザーズマニュアル
- H8S，H8/300 シリーズ クロスアセンブラ ユーザーズマニュアル
- H シリーズ リンケージエディタ、ライブラリアン、オブジェクトコンバータ ユーザーズマニュアル
- 日立デバッギングインタフェース ユーザーズマニュアル
- 日立インテグレーションマネージャ ユーザーズマニュアル
- 使用する H8S シリーズハードウェアマニュアル、プログラミングマニュアル

〈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。
（‖）	括弧内の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。
nnnn	太字斜体のnnnnはデバイスに対応しています。 例：H8S/2655の場合はnnnn=2655となります。
z	太字斜体のzは動作モードを示します。 aはアドバンストモードです。 nはノーマルモードです。

なお、使用環境によっては、ディレクトリ区切り記号「¥」が「＼」と表示される場合があります。

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

図目次

図目次

5．アプリケーションプログラムの作成

6．セットアップテーブルの作成

7．割込みベクタテーブルの作成

8．ロードモジュールの生成

付録 C．デバイスドライバ

表目次

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。

HI2000/3 は H8S シリーズ CPU で動作するリアルタイム・マルチタスク OS で、μITRON 3.0 仕様に準拠しています。

また、HI2000/3 用に作成したアプリケーションプログラムのデバッグを行うためのソフトウェアとして、HI2000/3 デバッギングエクステンション(DX)も備えています。

HI2000/3 デバッギングエクステンション(DX)は、日立デバッギングインタフェースおよび HI2000/3 システムに組み込んで使用します。

1.2 特長

HI2000/3 の特長を以下に示します。

(1) 高速なカーネル

H8S シリーズ CPU の高速動作に適した命令セットを利用し、高速性能が発揮できるようにカーネルを最適化し、高速な処理を実現しています。

カーネルは、H8S シリーズの割込み制御モード(4 種類)すべてに対応可能で、CPU 動作モードに応じてアドバンストモード用とノーマルモード用の 2 種類があります。

アドバンストモード用は最大 16M バイト(データ専用の領域を含めて合計 4G 空間)のアドレス空間、ノーマルモード用は最大 64K バイトのアドレス空間で動作します。また、カーネル内でのパラメータチェックを行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。

(2) 機能選択可能でコンパクトなカーネル

ユーザシステムに必要となる ROM/RAM を最小にできるように、カーネルのプログラムサイズおよび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カーネル機能を選択することで、ユーザシステム用に最適化されたカーネルを構築することができます。

(3) 高級言語対応

日立製コンパイラの使用により、タスク、割込みハンドラなどすべてのプログラムを C 言語で記述することができます。

(4) デバッギングエクステンション

HI2000/3 のシステムコールの履歴表示、タスク等のオブジェクト状態の参照と変更が、ウィンドウやダイアログボックスを通して行え、日立デバッギングインタフェースの環境でマルチタスクアプリケーションをデバッグできます。また、Windows®標準のコンテキスト依存型ヘルプシステムも備えています。

(5) サンプルプログラム

サンプルプログラムとして、以下のソースプログラムを提供しています。必要に応じてユーザシステム用に修正するだけで容易にシステムを構築できます。

- システム構築用ファイル
- 各種ハンドラ、およびルーチン
- H8S シリーズ内蔵 TPU(Timer Pulse Unit), FRT(Free Running Timer)用タイマドライバ
- タスク例(HI2000/3 デバッギングエクステンション(DX)のチュートリアル)

2. カーネル

2.1 概要

リアルタイム・マルチタスク処理を行う核となる部分を「カーネル」と呼びます。
以下にユーザが利用するカーネルの 3 つの役割を示します。

(1) **各事象への対応**

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仕事(タスク)を直ちに実行します。

(2) **タスクのスケジューリング**

優先度に応じて、仕事(タスク)の実行順序を決定します。

(3) **システムコールの実行**

仕事(タスク)からの各種処理要求(システムコール)を受け付け、その処理を行います。

2.2 カーネルの機能

カーネル機能のほとんどは、アプリケーションプログラムからシステムコール発行という形で利用することができます。

(1) **タスク管理機能**

タスクは実行時に CPU が割り付けられます。カーネルは、割付け順序を制御したり、タスクの起動、終了などタスクの状態を管理します。また、共有スタック機能により複数のタスクでスタックを共有することができます。

(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、メイルボックスを用いて行います。

(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われます。

(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をメモリプールとして管理します。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。メモリプールには固定長メモリプールと可変長メモリプールがあります。

(6) 時間管理機能

時間の管理を行います。また、タスク実行制御のため、時間監視を行います。

(7) システム管理機能

カーネルのバージョン情報の読出しを行います。

(8) システムコールトレース機能

システム実行中のシステムコール発行の履歴を保存します。

2.3 システム状態

カーネルには、図 2.1 に示すシステム状態があります。ユーザのシステム構成で、システム状態を考慮する必要があります。表 2.1 にシステム状態を説明します。

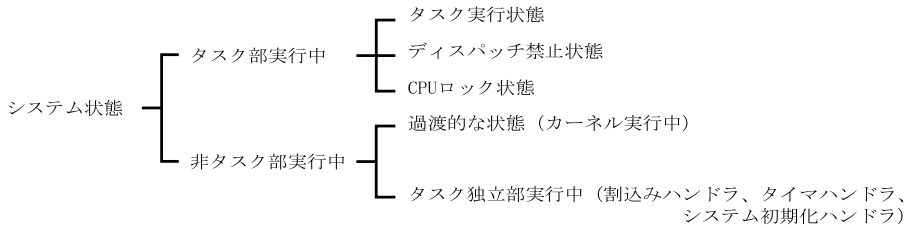


図 2.1 システム状態

表 2.1　システム状態

システム状態	**タスク部実行中** タスクが実行している状態です。	**タスク実行状態** タスク部実行中で、タスク切替え、割込み共に許可した状態です。タスクはこの状態で実行します。この状態以外では、タスクのディスパッチ(スケジューリング)はタスク実行状態に戻るまで遅延されます。 この状態では、タスク実行状態から発行可能なシステムコールが使用できます。
		ディスパッチ禁止状態 タスク部実行中で、タスクのディスパッチ(スケジューリング)を禁止した状態です。この状態では、他のタスクへの切替えは起こりません。 タスク実行状態から dis_dsp システムコールの発行により、ディスパッチ禁止状態に遷移します。タスク実行状態に戻るには、ena_dsp システムコールを発行します。この状態では、待ち状態に遷移するシステムコールは使用できません。ディスパッチ禁止状態から発行可能なシステムコールが使用できます。
		CPU ロック状態 タスク部実行中で、ディスパッチ禁止状態に加え、割込みも禁止した状態です。 タスク実行状態から loc_cpu システムコールの発行により、CPU ロック禁止状態に遷移します。タスク実行状態に戻るには、unl_cpu システムコールを発行します。この状態では、待ち状態に遷移するシステムコールは使用できません。CPU ロック状態から発行可能なシステムコールが使用できます。
	非タスク部実行中 タスク部以外を実行している状態です。	**過渡的な状態(カーネル実行中)** カーネル実行中(システムコール処理中)の状態です。
		タスク独立部実行中 タスク独立部は、タスクとは完全に独立して動作する部分であり、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発行できません。また、タスクの切替えも起こりません。タスク切替えはタスク実行状態に戻るまで遅延されます。 タスク独立部として動作するプログラムには、割込みハンドラ、タイマ割込みハンドラ、システム初期化ハンドラがあります。タスク独立部実行中では、タスク独立部から発行可能なシステムコールを使用できます。また、タスク部実行中に割込みをマスク(chg_ims システムコール)すると、その時点からタスク独立部に遷移します。その後、割込みマスクを 0 に戻した時点でタスク部実行中に戻ります。

2.4 タスク

2.4.1 タスクの概要

リアルタイム・マルチタスクシステムでは、アプリケーション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をタスクと呼びます。

タスクは、タスク間で必要な連絡をシステムコールを使用して行います。カーネルはシステムコールを介して、外部装置や計算機内部から非同期に発生した事象(イベント)を処理することができます。

表 2.2、表 2.3 にタスクを操作するシステムコールを示します。

表 2.2 タスクを操作するシステムコール(タスク管理)

項番	システムコール名	操作内容
1	sta_tsk ista_tsk	タスク起動
2	ext_tsk	自タスク終了
3	ter_tsk	他タスク強制終了
4	chg_pri	タスク優先度変更
5	rot_rdq irot_rdq	タスクのレディキュー回転
6	rel_wai	他タスクの待ち状態解除
7	get_tid	自タスクのタスク ID 参照
8	ref_tsk	タスク状態参照
9	dis_dsp	ディスパッチ禁止
10	ena_dsp	ディスパッチ許可

表 2.3 タスクを操作するシステムコール(タスク付属同期管理)

項番	システムコール名	操作内容
1	sus_tsk	他タスクを強制待ち状態へ移行
2	rsm_tsk	強制待ち状態のタスクを再開
3	slp_tsk	自タスクを起床待ち状態へ移行
4	tslp_tsk	自タスクを起床待ち状態へ移行(タイムアウト有)
5	wup_tsk iwup_tsk	他タスクの起床
6	can_wup	タスクの起床要求を無効化

2.4.2 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 6 つの状態を遷移します。

(1) **休止**(DORMANT)**状態**

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。

(2) **実行可能**(READY)**状態**

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はできない状態です。

(3) **実行**(RUN)**状態**

現在そのタスクを実行中であるという状態です。

(4) **待ち**(WAIT)**状態**

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。

(5) **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せられた状態です。

(6) **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。

図 2.2 にタスク状態遷移を示します。

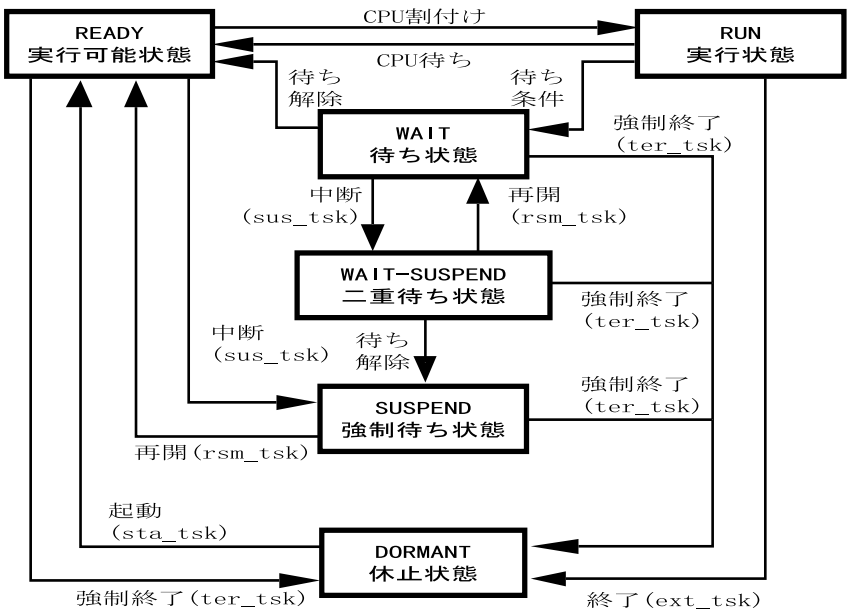


図 2.2 タスクの状態遷移

2.4.3 タスクの起動

休止状態のタスクを実行可能状態にすることを「タスクを起動する」といいます。タスクを起動するには、以下の方法があります。

- 目的のタスクに対して sta_tsk、ista_tsk システムコールを発行する。
- セットアップファイルにて、タスクの初期状態を実行可能状態に定義する。

2.4.4 タスクのスケジューリング

タスクのスケジューリングとは、実行可能状態のタスクを CPU に割り付けて実行させる順序を決定することです。実行可能状態のタスクの中から、1 つのタスクが選ばれ、実行状態になります。実行可能状態のタスクがない場合、カーネルはアイドリング状態となり、割込みからタスクが起床されることを待ち続けます。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 割り付け待ち行列によりタスクの実行順序が決められます。

レディキューは最大タスク優先度の数だけ存在し、FCFS(First Come First Service)で管理されます。タスク優先度は、値の小さい方が高い優先順位になります。

また、標準のスケジューリングの他に、レディキューを操作する rot_rdq、irot_rdq システムコールを使用することによって、ラウンドロビンスケジューリングを行うことができます。

ラウンドロビンスケジューリングとは、一定時間ごとにレディキューを回転させ同一優先度を持つタスクの CPU 割り付け時間を平均化するスケジューリングです。

ラウンドロビンスケジューリングは、一定周期で起動されるタイマ割込みハンドラから irot_rdq システムコールを発行することによってレディキューを回転させることで実現できます。

2.4.5 タスクの中断と再開

タスクの実行は、割込みや資源獲得待ちなどの要因で中断します。中断要因が解除されると以前の状態に復帰します。ただし、タスクは中断が解除されても直ちに実行を再開するわけではありません。タスクの実行順序は、前述したようにスケジューリングによって決まります。

表 2.4 にタスク実行中断と解除の要因を示します。

表 2.4 タスク実行中断と解除の要因

項番	中断の要因		中断解除の要因
1	自ら中断となる場合	slp_tsk，tslp_tsk システムコール	(1)wup_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《tslp_tsk》 (3)rel_wai システムコールが発行されるまで
		wai_flg，twai_flg システムコール	(1)イベントフラグの待ち条件が満たされるまで (2)指定したタイムアウト時間(tmout)が経過するまで《twai_flg》 (3)rel_wai システムコールが発行されるまで
		wai_sem，twai_sem システムコール	(1)セマフォで管理された資源が獲得できるまで (2)指定したタイムアウト時間(tmout)が経過するまで《twai_sem》 (3)rel_wai システムコールが発行されるまで
		rcv_msg，trcv_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)指定したタイムアウト時間(tmout)が経過するまで《trcv_msg》 (3)rel_wai システムコールが発行されるまで
		get_blf，tget_blf システムコール	(1)固定長メモリブロックが獲得できるまで (2)指定したタイムアウト時間(tmout)が経過するまで《tget_blf》 (3)rel_wai システムコールが発行されるまで
		get_blk，tget_blk システムコール	(1)可変長メモリ領域が獲得できるまで (2)指定したタイムアウト時間(tmout)が経過するまで《tget_blk》 (3)rel_wai システムコールが発行されるまで
2	他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 システムコールが発行されるまで
3	割込みにより中断させられる場合		割込みハンドラが終了するまで
4	共有スタックが占有されている場合	sta_tsk システムコール	共有スタックが開放されるまで

2.4.6 タスクの終了

タスクの終了とは、起動されたタスクの処理を終了し、休止状態になることをいいます。

- 自タスクで ext_tsk システムコールを発行する
- 目的のタスクに対して ter_tsk システムコールを発行する

タスクは一度終了すると、次に起動がかけられたときは初期状態から実行されます。
タスクは終了前に、獲得していた資源を解放しなければなりません。

2.4.7 共有スタック機能

共有スタック機能は、複数のタスクでスタックを共有する機能で、スタック全体の使用容量を削減します。
共有スタックの定義は、セットアップテーブルで行います。スタックを共有するタスク群では、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。

共有スタックは占有しているタスクが休止状態となった場合に解放され、共有スタック待ちが存在すれば、待ちの先頭タスクが占有し実行可能状態となります。
共有スタック待ちのタスクは、優先度に関係なく FIFO(First-In First-Out)で管理され、起動要求が行われた順番につながれます。
同一スタックを使用するように定義されているタスクが、同時に複数起動された場合は先に起動されたタスクがスタックを占有し、他のタスクは共有スタック待ちとなります。

図 2.3 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。

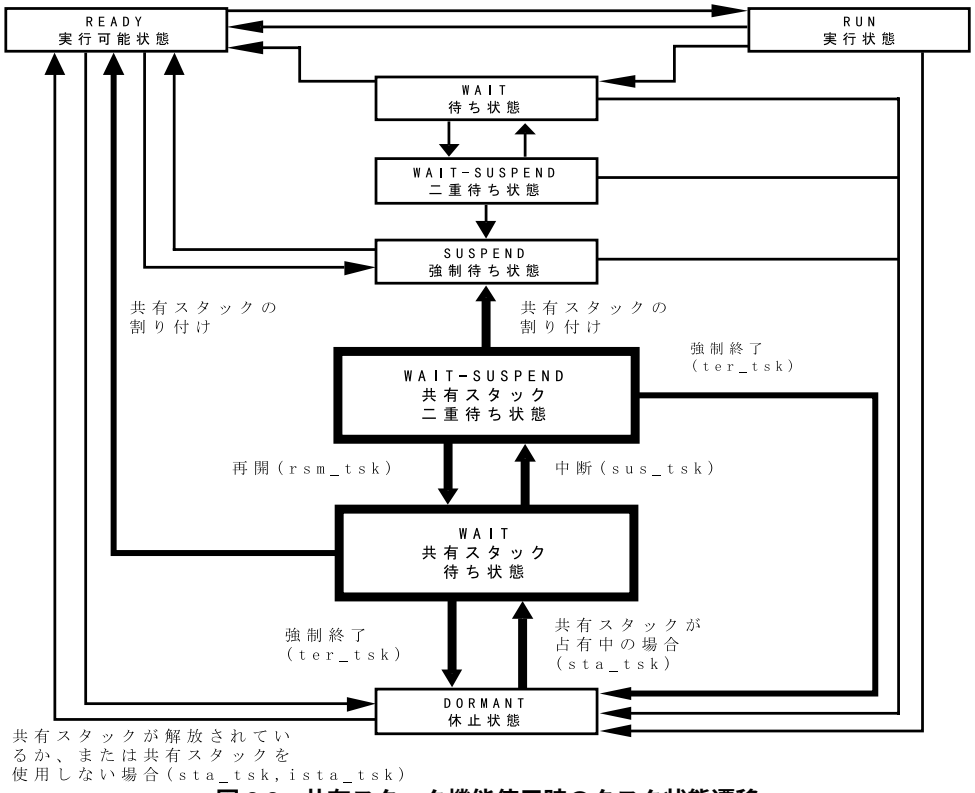


図 2.3 共有スタック機能使用時のタスク状態遷移

2.5 同期・通信

本カーネルでは、同期・通信のための機能として、以下の機能を持っています。これらは、タスクとは独立したオブジェクトです。

- イベントフラグ：複数事象の待ち合わせ、タスクの一斉動作
- セマフォ：資源の排他制御
- メイルボックス：データ通信(データへのポインタの受け渡し)

表 2.5 イベントフラグを操作するシステムコールを示します。

表 2.5 イベントフラグを操作するシステムコール

項番	システムコール名	操作内容
1	set_flg iset_flg	**イベントフラグのセット**
2	clr_flg	**イベントフラグのクリア**
3	wai_flg	**イベントフラグの待ち**
4	pol_flg	**イベントフラグの待ち(ポーリング)**
5	twai_flg	**イベントフラグの待ち(タイムアウト有)**
6	ref_flg	**イベントフラグ状態参照**

表 2.6 セマフォを操作するシステムコールを示します。

表 2.6 セマフォを操作するシステムコール

項番	システムコール名	操作内容
1	sig_sem isig_sem	**セマフォ資源返却**
2	wai_sem	**セマフォ資源獲得**
3	preq_sem	**セマフォ資源獲得(ポーリング)**
4	twai_sem	**セマフォ資源獲得(タイムアウト有)**
5	ref_sem	**セマフォ状態参照**

表 2.7 にメイルボックスを操作するシステムコールを示します。

表 2.7 メイルボックスを操作するシステムコール

項番	システムコール名	操作内容
1	snd_msg isnd_msg	**メイルボックスへ送信**
2	rcv_msg	**メイルボックスから受信**
3	prcv_msg	**メイルボックスから受信(ポーリング)**
4	trcv_msg	**メイルボックスから受信(タイムアウト有)**
5	ref_mbx	**メイルボックス状態参照**

2.5.1 イベントフラグ

イベントフラグを使用することにより、複数の事象発生の待ち合わせによるタスク間の同期処理を行うことができます。

イベントフラグは、事象に対応したビットの集合です。1 つのイベントフラグで複数個の事象を管理できます。イベントフラグは、1 つの事象発生が 1 ビットで表わされ、フラグのオン(1)／オフ(0)のみで行われます。

タスクは、イベントフラグの指定したビットのセット(事象の発生)を待つことができます。

1 つのイベントフラグには、複数のタスクが事象の発生を待つことができます。

図 2.4 にイベントフラグの使用例を示します。

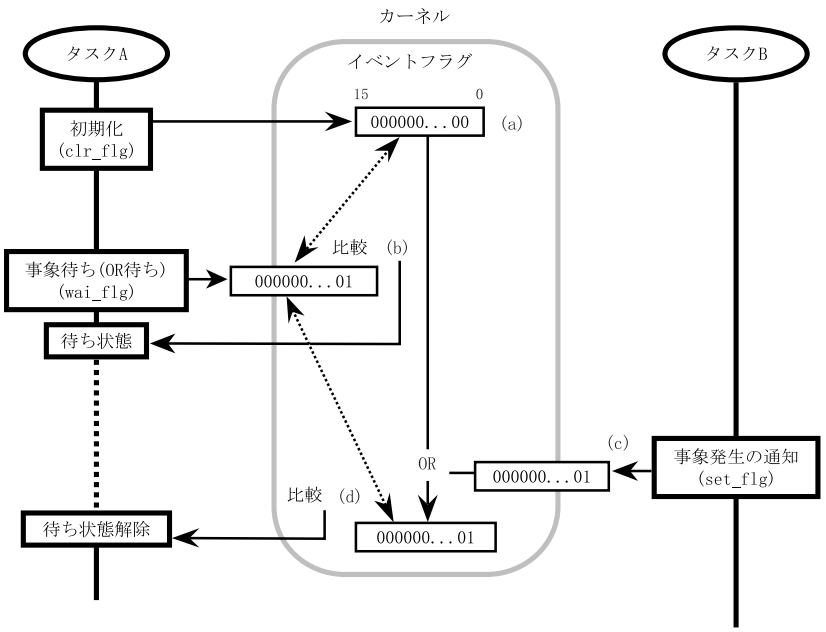


図 2.4 イベントフラグの使用例

(解説)

(a) タスク A がイベントフラグの全ビットをクリアします。

(b) 事象が発生していないので、タスク A は OR 待ちのモードで、指定した事象が発生するまで待ち状態なります(OR 待ち：指定した事象が少なくとも一つ発生することを待つ)。

(c) タスク B が事象発生の通知を行い、イベントフラグのビットがセットされます。

(d) 事象の発生を待っていたタスク A の OR 待ちの条件を満足したため、タスク A は待ち状態から解除されます。

2.5.2 セマフォ

タスク実行のために必要となる各種要素を資源と呼びます。資源には、各種 I/O や共有するメモリが考えられます。

セマフォを使用することにより、この資源の排他制御を行うことができます。

セマフォは、非負のカウンタ(セマフォカウンタ)を持っており、このカウンタ値を使用可能な資源の数に対応させて使用します。

図 2.5 にセマフォの使用例を示します。

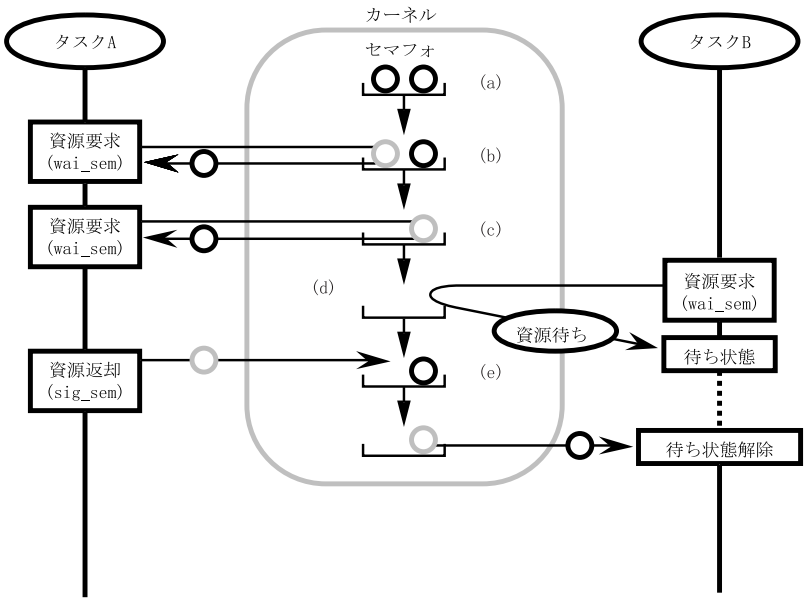


図 2.5 セマフォの使用例

(解説)

(a) 最初、資源が 2 つあります(セマフォカウンタ=2)。

(b) タスク A が 1 個の資源を要求したので、資源が 1 つになります(セマフォカウンタ=1)。

(c) タスク A がさらに 1 個の資源を要求したので、資源が 0 になります(セマフォカウンタ=0)。

(d) タスク B が 1 個の資源を要求したが、資源が 0 のため、タスク B は待ち状態となります。

(e) タスク A が 1 個の資源を返却したため、1 個の資源を要求したタスク B に資源が割り付けられ、タスク B は待ち状態から解除されます。

2.5.3 メイルボックス

メイルボックスを使用することにより、タスク間でメッセージと呼ばれるデータの受け渡しを行うことができます。

メイルボックスでは、メッセージの先頭アドレスの送受信を行います。

メイルボックスを用いた通信は、メッセージの先頭アドレスの受け渡しによって実現されているため、メッセージサイズに依存せずに高速に行われます。

図 2.6 にメイルボックスの使用例を示します。

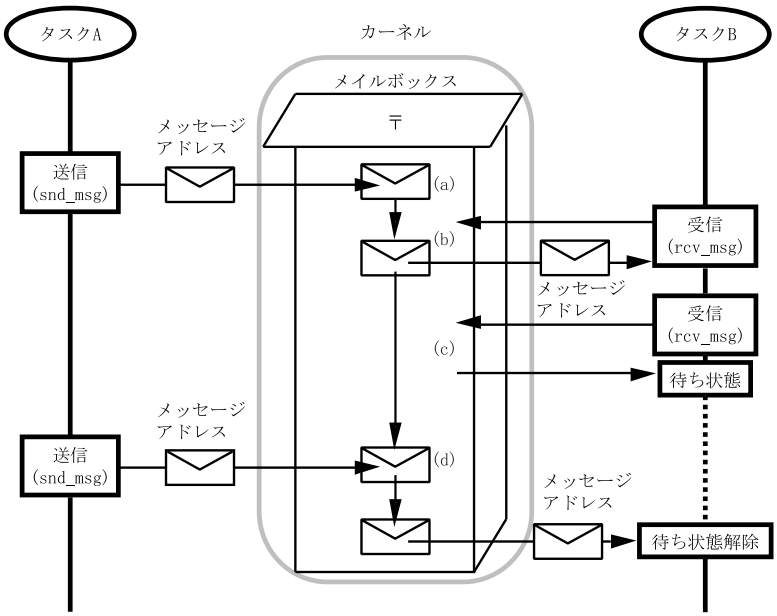


図 2.6 メイルボックスの使用例

(解説)

(a) タスク A がメッセージを送信し、メイルボックスに 1 通のメッセージが蓄えられます。

(b) タスク B がメッセージの受信要求を行うと、1 通のメッセージがタスク B に渡されます。

(c) タスク B がさらにメッセージの受信要求を行うと、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となります。

(d) タスク A がメッセージを送信したので、メッセージの受信要求をしていたタスク 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ます。

2.6 割込み

2.6.1 割込みの概要

外部ハードウェアや、周辺モジュールなどからの割込みが発生した場合、カーネルの介入を受けずに割込みハンドラが起動されます。

表 2.8 に割込み管理のシステムコールを示します。

表 2.8 割込み管理のシステムコール

項番	システムコール名	操作内容
1	ret_int	**割込みハンドラから復帰**
2	chg_ims	**割込みマスク変更**
3	ref_ims	**割込みマスク参照**
4	loc_cpu	**割込みとディスパッチの禁止**
5	unl_cpu	**割込みとディスパッチの許可**

2.6.2 割込みハンドラ

割込みが発生すると、その時実行していたタスクの実行は、割込みハンドラの処理が終了するまで中断されます。

割込みハンドラ実行中に発行したシステムコールにより、優先度の高いタスクが実行可能状態となっても、その時点ではスケジューリングは行われず、割込みハンドラの処理が終了するまで遅延されます。割込みがネストしている場合は、すべての割込みハンドラの処理が終了した時点でスケジューリングが行われます。

2.6.3 未定義割込み

未定義の割込みが発生した場合は、システム異常終了となります。このとき、システム異常終了ルーチンに未定義割込み発生時の情報が渡されます。

2.6.4 タスクの CPU 占有と割込みマスク

タスクが CPU を占有して実行するには、スケジューリングの抑止と割込みのマスクを行う必要があります。このためには、loc_cpu システムコールを用いてシステム状態を CPU ロック状態とします。CPU ロック状態からは、unl_cpu システムコールを発行することにより元のタスク実行状態に戻ります。また、タスクが CPU を占有して実行するには、CPU ロック状態の他に chg_ims システムコールで割込みをマスクすることによってシステム状態をタスク独立部にする方法があります。タスク独立部においても CPU ロック状態と同様に発行システムコールの制限、およびスケジューリングは遅延されます。

2.7 メモリプール

メモリプールを使用することにより、空きメモリを効率的に使用することができます。
メモリプールには、固定長メモリプールと可変長メモリプールがあります。

表 2.9 に固定長メモリプールを操作するシステムコールを示します。

表 2.9 固定長メモリプールを操作するシステムコール

項番	システムコール名	操作内容
1	get_blf	固定長メモリブロック獲得
2	pget_blf	固定長メモリブロック獲得(ポーリング)
3	tget_blf	固定長メモリブロック獲得(タイムアウト有)
4	rel_blf	固定長メモリブロック返却
5	ref_mpf	固定長メモリプール状態参照

表 2.10 に可変長メモリプールを操作するシステムコールを示します。

表 2.10 可変長メモリプールを操作するシステムコール

項番	システムコール名	操作内容
1	get_blk	可変長メモリブロック獲得
2	pget_blk	可変長メモリブロック獲得(ポーリング)
3	tget_blk	可変長メモリブロック獲得(タイムアウト有)
4	rel_blk	可変長メモリブロック返却
5	ref_mpl	可変長メモリプール状態参照

2.7.1 固定長メモリプール

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

図 2.7 に固定長メモリプールの使用例を示します。

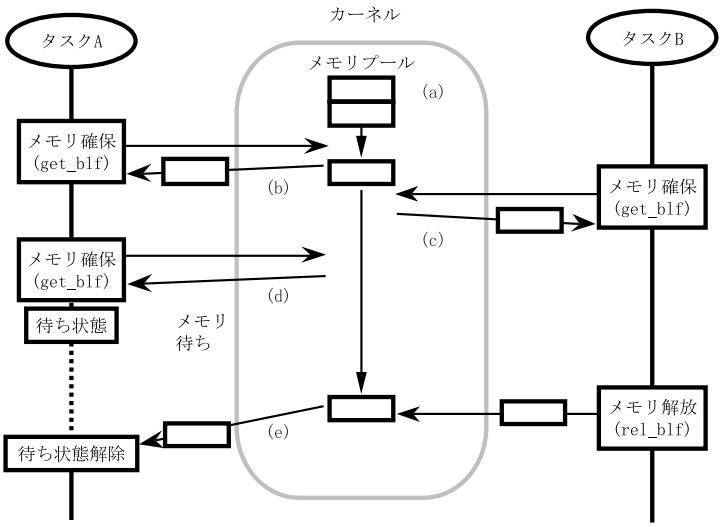


図 2.7 固定長メモリプールの使用例

(解説)

(a) メモリプールに 2 個分のメモリブロックがあります。
(b) タスク A がメモリブロックを獲得したので、メモリブロックが 1 個になります。
(c) タスク B もメモリブロックを獲得したので、メモリブロックが 0 個になります。
(d) タスク A がさらにメモリブロックを獲得しようとしたが、空きメモリブロックがないために、タスク A は待ち状態となります。
(e) タスク B がメモリブロックを返却したため、メモリブロックを獲得しようとしたタスク A にメモリブロックが割り付けられ、タスク A は待ち状態から解除されます。

2.7.2 可変長メモリプール

可変長メモリプールは、メモリプールから可変サイズ(バイト単位)のメモリを 1 つのメモリブロックとして獲得し、使用することができます。

図 2.8 に可変長メモリプールの使用例を示します。

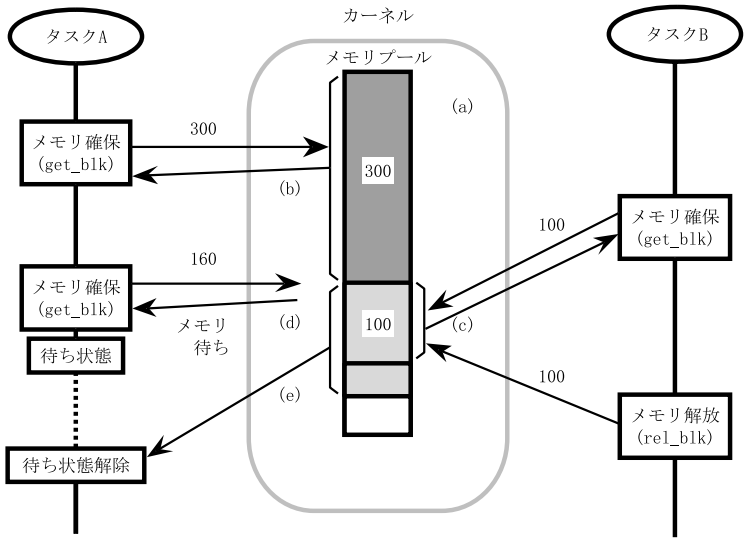


図 2.8 可変長メモリプールの使用例

(解説)

(a) メモリプールに連続した空き領域のメモリブロックが 500 バイトあります。

(b) タスク A がメモリブロックサイズ 300 バイトを獲得した(ブロック獲得管理用に 16 バイト使用されます)ので、空きメモリブロックサイズが 184(=500-300-16)バイトになります。

(c) タスク B もメモリブロックサイズ 100 バイトを獲得した(ブロック獲得管理用に 16 バイト使用されます)ので、空きメモリブロックが 68(=184-100-16)バイトになります。

(d) タスク A がメモリブロックサイズ 160 バイトを獲得しようとしたが、空きメモリブロックサイズが 68 バイトのために、タスク A は待ち状態となります。

(e) タスク B がメモリブロックサイズ 100 バイトを返却したため、タスク A に 160 バイトのメモリブロックが割り付けられ、タスク A は待ち状態から解除されます。空きメモリブロックサイズは、8 バイト(=68+100+16-160-16)になります。

2.8 時 間

2.8.1 時間の概要

ハードウェアタイマで作られる一定周期のクロックを使用することにより、時間の管理を行うことができます。以下に、その概要を示します。

(1) **時間の参照・設定**

システムで決められたある時点からハードウェアクロックをカウントすることで、時間を管理します。

(2) **時間によるハンドラの実行制御**

周期起動ハンドラの周期の経過を監視し、実行制御を行います。

(3) **時間によるタスクの実行制御**

時間を利用したタスクの実行制御を行います。

これらの機能を使用する場合には、タイマハンドラをユーザが作成する必要があります。
タイマハンドラの作成に関しては、「付録 C　デバイスドライバ」を参照してください。
表 2.11 にシステムクロックを操作するシステムコールを示します。

表 2.11　**システムクロックを操作するシステムコール**

項番	システムコール名	操作内容
1	set_tim	**システムクロック設定**
2	get_tim	**システムクロック参照**

2.8.2 ハードウェアタイマとシステムクロック

時間管理を使用するためには、一定周期で割込みを発生するハードウェアタイマが必要です。
ハードウェアタイマから起きる一定周期の割込みをカウントすることにより、時間を管理しています。カーネル内での時間(システムクロックの値)は、このハードウェアタイマの周期時間(tc)を単位とします。カーネル内での時間(システムクロックの値)と、現実の時間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)

例えば、ハードウェアタイマの周期が 1msec の場合、カーネル内(システムクロック値)の時間 100 は、100msec を表わします。

2.8.3 システムクロックの設定と参照

ハードウェアタイマの割込みごとに、符号付き 48 ビットのシステムクロックカウンタが更新(+1)されます。このシステムクロックカウンタの値を参照することにより時間を求めることができます(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 1msec であれば、約 4000 年に相当します)。

2.8.4 周期起動ハンドラ

周期起動ハンドラを用いることにより、周期的な処理を行うことができます。
周期起動ハンドラは、指定した周期起動時間間隔ごとに起動されるタスク独立部のハンドラです。

表 2.12 に周期起動ハンドラを操作するシステムコールを示します。

表 2.12 周期起動ハンドラを操作するシステムコール

項番	システムコール名	操作内容
1	act_cyc	**周期起動ハンドラ活性制御**
2	ref_cyc	**周期起動ハンドラ状態参照**

図 2.9 に周期起動ハンドラの処理概要を示します。

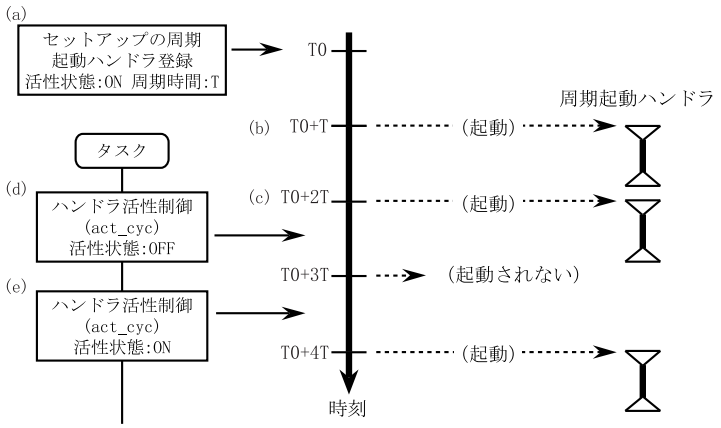


図 2.9 周期起動ハンドラの処理概要

(解説)

(a) セットアップテーブルに登録した周期起動ハンドラ(活性状態を ON、周期時間を T)が、システム起動からの経過時間で起動します。
(b) 周期時間 T が経過したとき、周期起動ハンドラが起動されます。
(c) 周期時間 T が経過し時刻 T0+2T となったとき、周期起動ハンドラが起動されます。
(d) act_cyc システムコールで活性状態を OFF にすると、周期時間 T が経過(時刻 T0+3T)しても周期起動ハンドラは起動されません。
(e) act_cyc システムコールで活性状態を ON にすると、周期時間 T が経過(時刻 T0+4T)したとき、再び周期起動ハンドラが起動されます。

2.9 システム管理

本カーネルでは、システム管理のための機能としてカーネルのバージョンを得る機能を持っています。

表 2.13 にカーネルのバージョンを得るシステムコールを示します。

表 2.13 **カーネルのバージョンを得るシステムコール**

項番	システムコール名	操作内容
1	get_ver	**バージョン参照**

2.10 システムコールトレース機能

システムコールトレース機能は、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。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には、システム構築時にセットアップテーブルでトレース機能の登録およびトレースバッファ領域の確保を行います。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替えられます。

2.11 トレースバッファの構造

図 2.10 に、トレースバッファの構造を示します。

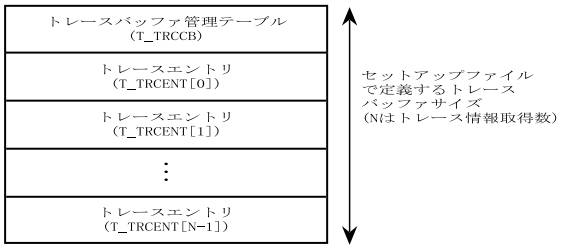


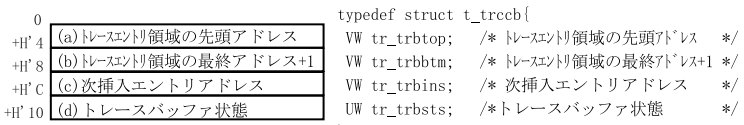
図 2.10 トレースバッファの構造

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 2.11 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{
 VW tr_trbtop;    /* ﾄﾚｰｽｴﾝﾄﾘ領域の先頭ｱﾄﾞﾚｽ    */
 VW tr_trbbtm;    /* ﾄﾚｰｽｴﾝﾄﾘ領域の最終ｱﾄﾞﾚｽ+1 */
 VW tr_trbins;    /* 次挿入エントリアドレス     */
 UW tr_trbsts;    /*トレースバッファ状態       */
} T_TRCCB;
```

図 2.11 トレースバッファ管理テーブルの構造

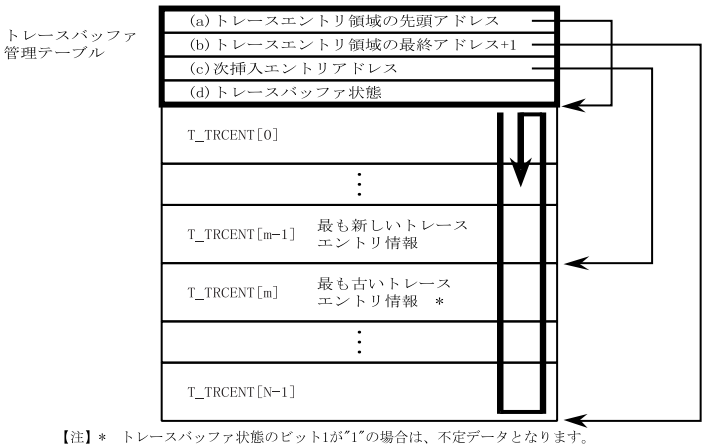
トレース情報の格納可能な領域は、(a)および(b)で保持しています。これらは、システム起動時にセットアップテーブルの指定内容にしたがって初期化されます。

(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 2.12 に、トレースバッファ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 2.12 **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 2.13 に、トレースエントリの構造を示します。

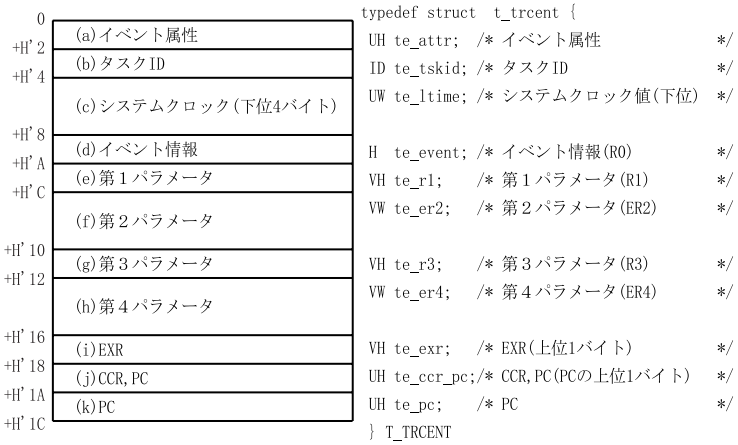


図 2.13 トレースエントリの構造

(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アプリケーション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、あるいはタスク、ハンドラ起動時の情報となります。

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。

(i) 割込みハンドラがRTE命令を実行することで割込み発生前に実行していたタスクに戻る場合
(ii) 割込みハンドラがタスク切り替えの必要のあるシステムコール(システムのタイマ割込み含む)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
(iii) 割込みハンドラでタスク切り替えの必要のあるシステムコール(システムのタイマ割込み含む)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(iii)の場合に取得されます。(i)、(ii)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 2.14 に、トレースエントリ内容の意味を示します。

表 2.14 トレースエントリ内容の意味

te_atr	TATR_SVC (H'0001)	TATR_RTN (H'0002)	TATR_CNT (H'0003)	TATR_IDLE (H'0004)
te_tskid	システムコールを発行したタスク ID タスク独立部の場合は 0	これから制御を渡すタスク ID タスク独立部に戻る場合は 0	割込み発生時点から実行再開するタスク ID タスク独立部(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ド*1 (ret_int システムコールは取得されません)	システムコールのエラーコードただし、H'8000 の場合はタスク起動を示します(R0)	不定	不定
te_r1, te_er2, te_r3, te_er4	システムコールのパラメータ(システムコール発行時の R1,ER2,R3,ER4)	システムコールのリターンパラメータ。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 R1,ER2,R3,ER4)	不定	不定
te_exr	システムコール発行時の EXR	ユーザプログラムに戻ったときの EXR	不定	不定
te_ccr_pc	システムコール発行時の CCR	ユーザプログラムに戻ったときの CCR	不定	不定
te_pc	システムコール発行アドレス	ユーザプログラムへの戻り先アドレス。タスク起動時は、タスクスタートアドレスとなり、タスク起動時以外は以前のシステムコール発行アドレスとなります。	不定	不定

【注】 *1 システムコールの機能コードについては、「付録 E．システムコール機能コード一覧」を参照してください。

2.12 トレース取得データの解析例

表 2.15 に、トレース取得データの例を示します。

表 2.15 トレース取得データの例

No.		te_atr	te_tskid	te_ltime	te_event	te_r1	te_pc	te_ccr	te_exr
-12	古	H'0001	H'0005	H'00001234	H'ffe9	H'0003	H'003018	H'00	H'00
	↑	SVC	tskid=5		sta_tsk	ID=3 を起動			
-11		H'0002	H'0003	H'00001234	H'8000	H'xxxx	H'003800	H'00	H'00
		RTN	tskid=3		ﾀｽｸ起動				
-10		H'0001	H'0003	H'00001234	H'ffda	H'xxxx	H'003810	H'00	H'00
		SVC	tskid=3		slp_tsk				
-9		H'0002	H'0005	H'00001234	H'0000	H'0003	H'003018	H'00	H'00
		RTN	tskid=5		E_OK				
-8		H'0001	H'0000	H'00001234	H'ff87	H'0003	H'007340	H'00	H'05
		SVC	非タスク		iwup_tsk	3 番のﾀｽｸを起床		ｺﾝﾄﾛｰﾙ ﾚﾍﾞﾙ=0	割込み ﾚﾍﾞﾙ=5
-7		H'0002	H'0000	H'00001234	H'0000	H'0003	H'007340	H'00	H'05
		RTN	非タスク		E_OK			ｺﾝﾄﾛｰﾙ ﾚﾍﾞﾙ=0	割込み ﾚﾍﾞﾙ=5
-6		H'0002	H'0003	H'00001234	H'0000	H'xxxx	H'003810	H'00	H'00
		RTN	tskid=3		E_OK				
-5		H'0001	H'0003	H'00001234	H'ffe1	H'0005	H'003840	H'00	H'00
		SVC	tskid=3		rel_wai	5 番を強制待ち解除			
-4		H'0002	H'0003	H'00001234	H'ffc1	H'0005	H'003840	H'00	H'00
		RTN	tskid=3		E_OBJ				
-3		H'0001	H'0003	H'00001234	H'ffeb	(不定データが入っている)			
		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'ffeb	(不定データが入っている)			
		SVC	tskid=5		ext_tsk				
-0	↓	H'0004	(不定)	H'00001234	(不定データが入っている)				
	新	IDLE							

【注】 ・説明を簡単にするため、te_er2**、**te_r3**、**te_er4 **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 2.15 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'ffe9 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_r1=H'0003 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'003018 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'8000 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'003800 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'ffda より 3 番のタスクが slp_tsk システムコール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'ff87、te_r1=3 より、タスク独立部から iwup_tsk システムコール(tskid=3)が発行されたことがわかります。タスク独立部には、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_ccr=H'00、te_exr=H'05 より優先度レベル 0 の割込みレベル 5 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、タスク独立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。直前のタスク独立部からのシステムコール発行は、イベント No.-8 の iwup_tsk システムコールなので、te_event はこの iwup_tsk システムコール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 のイベントはタスク独立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’ffe1、te_r1=H’5 より、3 番のタスクが rel_wai システムコール(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’ffc1 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_OBJ)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’ffeb 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。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’ffeb 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、図 2.14 にトレース解析結果の図示例を示します。理解を深めるための参考にしてください。

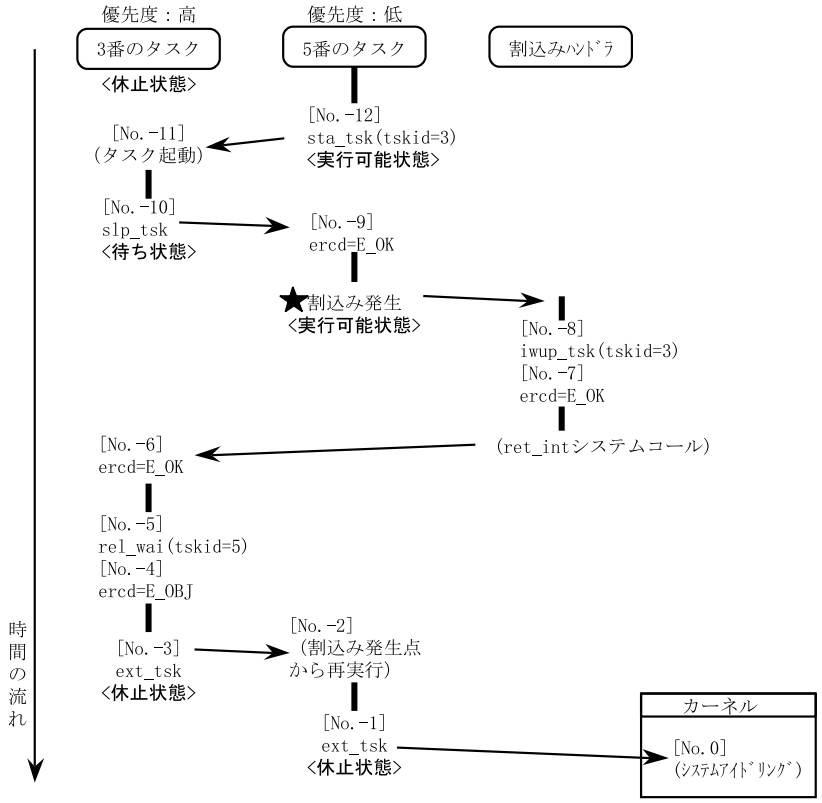


図 2.14　トレース解析結果の図示例

2.13 トレース機能の登録方法

トレース機能の登録については、「6.2.6　トレース機能の登録」を参照してください。

2.14 トレース機能の注意事項

(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。また、これらの性能悪化は、トレースバッファを割り当てるメモリにも依存します。

(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わないでください。トレースバッファ管理テーブルのデータを書き替えた場合、システムの正常な動作は保証されません。

(3) RTN 属性のトレース情報

ノーマルモードにおける RTN 属性で、トレース情報の E2 レジスタ値は不正になります。

(4) 割込み制御モード

割込み制御モード 0、1 の場合、トレース情報の EXR レジスタの値は不定になります。
また、制御モード 2 の場合は、トレース情報の CCR レジスタの値は不定になります。

3. システムコール

3.1 概 要

各システムコールの機能は、表 3.1 のように分類されます。

表 3.1 システムコールの機能分類

項番	分類	機能
1	タスク管理機能	タスクの起動、終了など
2	タスク付属同期機能	タスクの中断、再開、タスク付属イベントフラグなど
3	同期・通信機能	セマフォ、イベントフラグ、メイルボックス
4	割込み管理機能	割込みハンドラからの復帰、割込みマスクの変更と参照など
5	メモリプール管理機能	動的なメモリの割り付け
6	時間管理機能	システムクロックの設定と参照、周期起動ハンドラなど
7	システム管理機能	カーネルのバージョン参照

システムコール名は、システムコールを発行するプログラムのシステム状態で異なる場合があります。以下にその例を示します。

- **タスク部用とタスク独立部用が同一名を持つシステムコール**

get_tid、ref_tsk、can_wup など

- **タスク部用とタスク独立部用が異なる名前を持つシステムコール**

sta_tsk と ista_tsk、rot_rdq と irot_rdq、wup_tsk と iwup_tsk など

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方法について説明します。次節に、システムコール別に発行方法と機能について詳細に説明します。

3.2.1 C 言語インタフェース

カーネルは、C 言語で記述されたタスクやハンドラからシステムコールを使用できるよう、C 言語インタフェースライブラリを提供しています。

C 言語インタフェースライブラリはライブラリファイルと C 言語ヘッダファイルで構成されます。
ライブラリファイルは、2600CPU のアドバンストモード用とノーマルモード用、2000CPU のアドバンストモード用とノーマルモード用を各々提供してあります。
C 言語で記述されたプログラムからシステムコールを発行する場合、C 言語ヘッダファイルをソースプログラム中にインクルードし、システム構築時に C 言語インタフェースライブラリを結合してください。

(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 3 つの形式になります。

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

ercd	:システムコールのリターン値でエラーコード (符号付き 16 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またはパラメータの無いシステムコール

(2) パラメータ名称

パラメータ名称は統一的な省略名を使用しており、次の接頭語または接尾語を付けています。

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_〜	パケットへのポインタを示す
ppk_〜	パケットポインタへのポインタを示す
〜id	ID を示す
〜cd	コードを示す
i〜	初期値を示す
〜sz	サイズを示す
〜cnt	個数を示す

(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。これらは、カーネルの C 言語ヘッダファイルで定義されています。

```
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)();    /* プログラムのスタートアドレス一般         */
typedef H                 INT;        /* 符号付き 16 ビット整数                   */
typedef UH                UINT;       /* 符号なし 16 ビット整数                   */
typedef INT               BOOL;       /* ブール値 FALSE(0:偽)または TRUE(1:真)    */
typedef int               FN;         /* 機能コード                               */
typedef UH                ID;         /* オブジェクトの ID 番号(???id)            */
typedef ID                BOOL_ID;    /* ブール値または ID 番号                   */
typedef H                 HNO;        /* ハンドラ番号                             */
typedef H                 ER;         /* エラーコード                             */
typedef H                 PRI;        /* タスク優先度                             */
typedef W                 TMO;        /* タイムアウト                             */
typedef TMO               CYCTIME;    /* 周期起動時間間隔                         */
```

(4) システムコールの発行例

C 言語プログラムからのシステムコール発行例を、以下に示します。

```
#include "hi2000.h"
void        task(INT stacd)
{
ER          ercd;
ID          tskid;
INT         stacd;
            /* ...                              */
            ercd = sta_tsk(tskid,stacd);
            /* ...                              */
}
```

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

アセンブリ言語プログラムからシステムコールを発行する場合は、各レジスタにパラメータを設定後、JSR 命令を実行します。以下に sta_tsk システムコールを例に示します。

```
MOV.W       #TSKID, R1        ····································· (a)
MOV.W       #STACD, R2        ····································· (b)
JSR         @sta_tsk          ····································· (c)
```

(a)システムコールのパラメータ(タスク ID)を定められたレジスタに設定します。
(b)システムコールのパラメータ(起動コード)を定められたレジスタに設定します。
(c)システムコールごとに定められた形式で JSR 命令を行います(システムコールによっては、JMP 命令を用いるものもあります)。

システムコール終了時に、エラーコードを R0 に返します。エラーコードが設定される R0 およびリターンパラメータのレジスタ以外のレジスタ内容は、システムコール発行前の値を保持します。

(2) パラメータに用いる定数

カーネルは、アセンブリ言語ヘッダファイルを提供しています。アセンブリ言語ヘッダファイルでは、各種の定数があらかじめ定義されています。

3.2.3 エラーコード

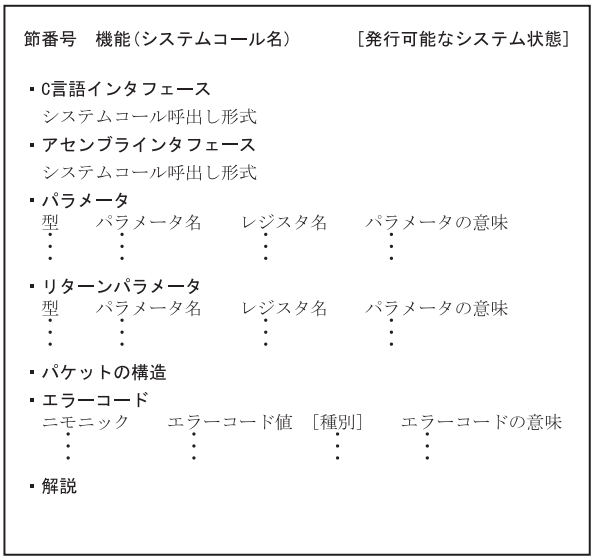
一部のシステムコールを除き、システムコール結果がエラーコードとして返されます。エラーコードは、ER 型(符号付き 16 ビット整数)として R0 に設定されます。なお、システムコールの結果は CCR レジスタの各フラグには反映されません。

各システムコールで発生する可能性のあるエラーコードについては次節で説明しています。
カーネルではパラメータチェック機能付き」と「パラメータチェック機能なし」の 2 種類のカーネルライブラリを提供しています。後者のカーネルライブラリを使用すると、カーネルはシステムコールパラメータの静的なエラー検出を省略するため、システムコール処理時間は短縮しますが、システムコールパラメータに誤りがあった場合、システムの正常な動作は保証されません。

通常は、デバッグ時にはパラメータチェック機能付きのカーネルライブラリを用い、デバッグ終了後にパラメータチェック機能なしのカーネルライブラリに切り替えます。

3.3 システムコール仕様

本節では、システムコールについての詳細を図 3.1 の形式で説明します。



・発行可能なシステム状態：
発行可能なシステム状態を以下のニモニックで示します。
T：タスク実行状態
D：ディスパッチ禁止状態
L：CPUロック状態
I：タスク独立部

・レジスタ名(パラメータ／リターンパラメータ)：
ERx／Rx：アドバンスト／ノーマルモードによってレジスタのサイズが異なります。
Rx ：アドバンスト／ノーマルモード共通のレジスタのサイズです。

・システムコール名
パラメータ、リターンパラメータおよびエラーコードに"(システムコール)"が記載されている場合、対象となるシステムコールを意味します。

・パケットの構造
パケットを扱うシステムコールでは、パケットを以下のように表記します。

```
typedef struct t_rsem {
        VP exinf ;          0/ 0    4/2    拡張情報
        BOOL_ID wtsk ;     +4/+2    2/2    待ちタスクID
        UINT semcnt ;      +6/+4    2/2    現在のセマフォカウント値
} T_RSEM ;
```

C言語での構造体表記　　メンバの説明
パケット先頭からのオフセット　メンバのサイズ

パケット先頭からのオフセット、およびメンバのサイズ
x/x　：アドバンスト／ノーマルモードによってパケット先頭からのオフセット、およびメンバのサイズが異なります。
x　　：アドバンスト／ノーマルモード共通のパケット先頭からのオフセット、およびメンバのサイズです。

・エラーコード種別
[p]　：パラメータチェック機能付きカーネルの場合のみ検出されるエラーです。
[k]　：パラメータチェック機能の有無に関係なく、必ず検出されるエラーです。

図 3.1　システムコールの説明形式

3.4 タスク管理機能

(1) タスク管理システムコール

表 3.2 にタスク管理でサポートしているシステムコール一覧を示します。

表 3.2 タスク管理システムコール

項番	システムコール	機 能	発行可能なシステム状態			
			T	D	L	I
1	sta_tsk	タスク起動	○	○	○	
2	ista_tsk	タスク起動(タスク独立部用)		○		○
3	ext_tsk	自タスク終了	○	○	○	
4	ter_tsk	他タスク強制終了	○	○	○	
5	chg_pri	タスク優先度変更	○	○	○	
6	rot_rdq	タスクのレディキュー回転	○	○	○	
7	irot_rdq	タスクのレディキュー回転(タスク独立部用)		○		○
8	rel_wai	他タスクの待ち状態解除	○	○	○	
9	get_tid	自タスク ID 参照	○	○	○	
10	ref_tsk	タスク状態参照	○	○	○	○
11	dis_dsp	ディスパッチ禁止	○	○		
12	ena_dsp	ディスパッチ許可	○	○		

(2) タスク管理の仕様

表 3.3 にタスク管理の仕様を示します。

表 3.3 タスク管理の仕様

項番	項目	内容
1	タスク定義最大数	255 個
2	タスク ID	1～255
3	タスク優先度	1～31
4	タスクスタック	共有スタック機能有り
5	レディキュー	FCFS(First Come First Service)
6	共有スタック待ち行列(共有スタック機能使用時)	FIFO(First-In First-Out)

(3) タスクの実行中断と解除

タスクの実行中断と解除の要因を表 3.4 に示します

表 3.4 タスク実行中断と解除の要因

項番	実行中断の要因		中断解除の要因
1	割込みにより中断させられる場合		割込みハンドラが終了するまで
2	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	共有スタックが解放されるまで

3.4.1 タスク起動(sta_tsk) [T/D/L]
(ista_tsk) [D/I]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @sta_tsk
JSR @ista_tsk
```

- **パラメータ**

ID	tskid	R1	タスク ID
INT	stacd	R2	タスク起動コード

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(tskid≦0,tskid>タスク定義数)
E_NOEXS	H'ffcc(-H'34)	[p]	未登録(tskid のタスクが存在しない)
E_OBJ	H'ffc1(-H'3f)	[k]	オブジェクトの状態が不正(tskid のタスクが休止状態でない)
E_CTX	H'ffbb(-H'45)	[P]	コンテキストエラー(タスク実行状態(ista_tsk)またはタスク独立部(sta_tsk)から発行できない)
		[k]	(CPU ロック状態(ista_tsk)から発行できない)

- **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状態から実行可能状態へ移行します。
起動したタスクには、stacd で示されたタスク起動コードが渡されます。stacd は、タスクがアセンブラ記述の場合は R0 レジスタ、C 言語記述の場合は第一引数に渡されます。
本システムコールによる起動要求のキューイングは行いません。

起動するタスクが共有スタックを使用する場合は、共有スタックが解放されていれば共有スタックを占有し、実行可能状態へ移行します。また、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、スタック領域を確保できないため、タスクは待ち状態になります(共有スタックの待ち行列につながれます)。なお、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_OBJ を返します。

3.4.2 自タスク終了(ext_tsk) [T/D/L]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JMP @ext_tsk
```

- **パラメータ**

なし

- **リターンパラメータ**

なし

- **エラーコード**

正常終了時：[k]	本システムコールの発行元には戻りません
異常終了時：[p]	タスク独立部から発行した場合、システム異常処理に制御を移します

- **解説**

自タスクを正常終了します。本システムコールを発行したタスクは、実行状態から休止状態へ移行します。
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。占有していた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ってください。

他のタスクとスタックを共有し、スタック待ち行列に他のタスクがつながれている場合、待ち行列の先頭タスクをスタック待ち行列から外し、スタックを割り付けスタック待ち状態を解除します。

本システムコールは、ディスパッチ禁止状態またはCPUロック状態から発行できます。この場合、システム状態はタスク実行状態に移行します。

3.4.3 他タスク強制終了(ter_tsk) [T/D/L]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @ter_tsk
```

- **パラメータ**

ID	tskid	R1	タスク ID

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(tskid≦0,tskid>タスク定義数)
E_NOEXS	H'ffcc(-H'34)	[p]	未登録(tskid のタスクが存在しない)
E_OBJ	H'ffc1(-H'3f)	[p]	オブジェクトの状態が不正(自タスク指定)
		[k]	(タスクが休止状態である)
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない)

- **解説**

tskid で示された他タスクを強制的に終了させます。終了させた他タスクは、休止状態へ移行します。
tskid には、強制的に終了させる他タスクの ID を指定します。

本システムコールは、tskid で示される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。tskid で示されるタスクが占有していた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 tskid で示されたタスクが行う必要があります。

強制終了によって、tskid で示されるタスクが他のタスクとスタックを共有し、スタック待ち行列に他のタスクがつながれている場合、待ち行列の先頭タスクをスタック待ち行列から外し、スタックを割り付けスタック待ち状態を解除します。

3.4.4 タスク優先度変更(chg_pri) [T/D/L]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @chg_pri
```

- **パラメータ**

ID	tskid	R1	タスク ID
PRI	tskpri	R2	タスク優先度(0～優先度定義数)

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_PAR	H'ffdf(-H'21)	[p]	パラメータエラー(tskpri<0,tskpri>優先度定義数)
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(tskid<0,tskid>タスク定義数)
E_NOEXS	H'ffcc(-H'34)	[p]	未登録(tskid のタスクが生成されていない)
E_OBJ	H'ffc1(-H'3f)	[k]	オブジェクトの状態が不正(タスクが休止状態である)
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない)

- **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(0)の指定により、自タスクの指定となります。

tskpri には、タスク優先度として 0～優先度定義数の値を指定します。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(0)の指定により、タスク登録時に定義したタスクの初期優先度に戻します。

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タスク登録時に定義した初期優先度になります。

3.4.5 タスクのレディキュー回転 (rot_rdq) [T/D/L]
(irot_rdq) [D/I]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @rot_rdq
JSR @irot_rdq
```

- **パラメータ**

PRI	tskpri	R2	タスク優先度

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_PAR	H'ffdf(-H'21)	[p]	パラメータエラー(tskpri<0,tskpri>優先度定義数)
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク実行状態(irot_rdq)または タスク独立部(rot_rdq)から発行できない)
		[k]	(CPU ロック状態(irot_rdq)から発行できない)

- **解説**

tskpri で示されたタスク優先度のレディキューにつながれている先頭のタスクを、レディキューの最後尾につなぎかえ(レディキューを回転)、次につながれているタスクに実行を切り替えます。

tskpri には、タスク優先度として 0～優先度定義数の値を指定します。
tskpri=TPRI_RUN(0)の指定により、最高優先度のレディキュー(実行状態にあるタスクを含むレディキュー)を回転します。ただし、システム状態がディスパッチ禁止状態である場合には、実行状態のタスクが最高優先度でないことがあります。
tskpri=TPRI_RUN(0)、または自タスクのタスク優先度を指定すると、自タスクをレディキューの最後尾にまわすため、自ら実行権を放棄することができます。

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キューにタスクがない場合は何も行いませんが、正常終了となります。

3.4.6 他タスクの待ち状態解除(rel_wai) [T/D/L]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @rel_wai
```

- **パラメータ**

ID	tskid	R1	タスク ID

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(tskid≦0,tskid>タスク定義数)
E_NOEXS	H'ffcc(-H'34)	[p]	未登録(tskid のタスクが存在しない)
E_OBJ	H'ffc1(-H'3f)	[k]	オブジェクトの状態が不正(tskid に自タスクを指定、または待ち状態でないタスクを指定)
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない)

- **解説**

tskid で示されるタスクが待ち状態(強制待ち状態および共有スタック解放待ち状態は含まれません)の場合、その待ち状態を強制的に解除します。

tskid には、待ち状態を強制解除する他タスクの ID を指定します。

本システムコールにより待ち状態が解除されたタスクには、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いません。強制待ち状態を解除する場合は、rsm_tsk システムコールを発行してください。

二重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、対象タスクは強制待ち状態へ移行します。その後 rsm_tsk システムコールを発行し、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。

3.4.7 自タスクのタスク ID 参照(get_tid) [T/D/L]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @get_tid
```

- **パラメータ**

ID	*p_tskid	---	タスク ID を返す領域の先頭アドレス(C 言語ｲﾝﾀﾌｪｰｽ)

- **リターンパラメータ**

ID	*p_tskid	---	タスク ID を格納した領域の先頭アドレス(C 言語ｲﾝﾀﾌｪｰｽ)
---	tskid	R1	タスク ID(ｱｾﾝﾌﾞﾗｲﾝﾀﾌｪｰｽ)
ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない)

- **解説**

自タスクの ID を得ます。

3.4.8 タスク状態参照(ref_tsk) [T/D/L/I]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @ref_tsk
```

- **パラメータ**

ID	tskid	R1	タスク ID
T_RTSK	*pk_rtsk	ER2/R2	タスク状態を返すパケットの先頭アドレス

- **リターンパラメータ**

T_RTSK	*pk_rtsk	ER2/R2	タスク状態を格納したパケットの先頭アドレス
ER	ercd	R0	エラーコード

- **パケットの構造**

```
typedef     struct          t_rtsk {
                            VP    exinf;        0/  0   4/2    拡張情報
                            PRI   tskpri;      +4/ +2   2/2    現在のタスク優先度
                            UINT  tskstat;     +6/ +4   2/2    タスク状態
                            UINT  tskwait;     +8/ +6   2/2    待ち要因
                            ID    wid;        +10/ +8   2/2    待ちオブジェクト ID
                            H     wupcnt;     +12/+10   2/2    起床要求カウント
                            FP    task;       +14/+12   4/2    タスク先頭アドレス
                            PRI   itskpri;    +18/+14   2/2    タスク初期優先度
                            } T_RTSK;
```

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_PAR	H'ffdf(-H'21)	[p]	不正アドレス(pk_rtsk が 0 または奇数)
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(tskid<0,tskid>タスク定義数) (tskid=0：タスク独立部の場合のみ)
E_NOEXS	H'ffcc(-H'34)	[p]	未登録(tskid のタスクが存在しない)

- **解説**

tskid で示されたタスクの状態を参照し、pk_rtsk が指す領域に返します。
pk_rtsk の指す領域として 20 バイト(アドバンストモード)、または 16 バイト(ノーマルモード)の RAM 領域が必要です。
pk_rtsk の指す領域には、以下の値を返します。
なお、*が付いているデータは、タスクが休止状態の場合、無効となります。

(exinf)：タスク登録時に定義した拡張情報を返します。

(tskpri)：現在のタスク優先度を返します。

(tskstat)：現在のタスクの状態で、次の値を返します。

TTS_RUN (H'0001)	実行(RUN)状態
TTS_RDY (H'0002)	実行可能(READY)状態
TTS_WAI (H'0004)	待ち(WAIT)状態
TTS_SUS (H'0008)	強制待ち(SUSPEND)状態
TTS_WAS (H'000c)	二重待ち(WAIT-SUSPEND)状態
TTS_DMT (H'0010)	休止(DORMANT)状態
TTS_STK (H'4000)	共有スタック解放待ち状態
TTS_STS (H'4008)	二重待ち(共有スタック解放待ち-SUSPEND)状態

(tskwait)*：tskstat が TTS_WAI,TTS_WAS のとき、次の値を返します。

TTW_SLP (H'0001)	slp_tsk,tslp_tsk システムコールによる待ち
TTW_FLG (H'0010)	wai_flg,twai_flg システムコールによる待ち
TTW_SEM (H'0020)	wai_sem,twai_sem システムコールによる待ち
TTW_MBX (H'0040)	rcv_msg,trcv_msg システムコールによる待ち
TTW_MPL (H'1000)	get_blk,tget_blk システムコールによる待ち
TTW_MPF (H'2000)	get_blf,tget_blf システムコールによる待ち

(wid)*：tskstat が TTS_WAI,TTS_WAS のとき、待ち対象のオブジェクト ID を返します。

(wupcnt)*：現在の起床要求カウントを返します。

(task)：タスク先頭アドレスを返します。

(itskpri)：タスク登録時に定義した優先度(初期優先度)を返します。

tskid=TSK_SELF(0)の指定により自タスクの指定になりますが、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。

3.4.9 ディスパッチ禁止(dis_dsp) [T/D]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @dis_dsp
```

- **パラメータ**

なし

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない)
		[k]	(CPU ロック状態から発行できない)

- **解説**

タスクのディスパッチを禁止(システム状態をディスパッチ禁止状態)します。
ディスパッチ禁止状態から、ena_dsp システムコールを発行するとタスク実行状態に戻ります。

ディスパッチ禁止状態の特長を、以下に示します。

(1)タスクのスケジューリングが行われなくなるため、自タスク以外のタスクが実行状態に移行することはなくなります。タスク実行状態に戻った時点で、タスクのディスパッチ(スケジューリング)が行われます。
(2)割込みは受け付けられます。
(3)待ち状態になるシステムコールは発行できません。発行した場合にはエラーリターンします。

ディスパッチ禁止状態の間に ext_tsk システムコールなどによってタスクが終了した場合、システム状態はタスク実行状態に戻ります。
また、ディスパッチ禁止状態から unl_cpu システムコールを発行した場合もタスク実行状態に戻ります。

すでにディスパッチ禁止状態のときに再度本システムコールを発行しても正常終了しますが、キューイングは行いません。

3.4.10 **ディスパッチ許可**(ena_dsp) [T/D]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @ena_dsp
```

- **パラメータ**

なし

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない)
		[k]	(CPU ロック状態から発行できない)

- **解説**

タスクのディスパッチを許可します。

本システムコールの発行により、ディスパッチ禁止状態を解除し、システム状態をタスク実行状態にします。そして、タスクのディスパッチ(スケジューリング)が行われます。

タスク実行状態から本システムコールを発行しても正常終了しますが、キューイングは行いません。

3.5 タスク付属同期機能

(1) タスク付属同期システムコール

表 3.5 にタスク付属同期でサポートしているシステムコール一覧を示します。

表 3.5 タスク付属同期システムコール

項番	システムコール	機 能	発行可能なシステム状態			
			T	D	L	I
1	sus_tsk	他タスクを強制待ち状態へ移行	○	○	○	
2	rsm_tsk	強制待ち状態のタスクを再開	○	○	○	
3	slp_tsk	自タスクを起床待ち状態へ移行	○			
4	tslp_tsk	自タスクを起床待ち状態へ移行(タイムアウト有)	○			
5	wup_tsk	タスクの起床	○	○	○	
6	iwup_tsk	タスクの起床(タスク独立部用)		○		○
7	can_wup	タスクの起床要求を無効化	○	○	○	

(2) タスク付属同期の仕様

表 3.6 にタスク付属同期の仕様を示します。

表 3.6 タスク付属同期の仕様

項番	項目	内容
1	タスク起床要求の最大値	255 回
2	タスク強制待ち要求	キューイング無し

(3) タスクの実行中断と解除

タスクの実行中断と解除の要因を表 3.7 に示します

表 3.7 タスク実行中断と解除の要因

項番	実行中断の要因		中断解除の要因
1	自ら中断となる場合	slp_tsk,tslp_tsk システムコール	(1)wup_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 (tslp_tsk) (3)rel_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 システムコールが発行されるまで

3.5.1 他タスクを強制待ち状態へ移行（sus_tsk） [T/D/L]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @sus_tsk
```

- **パラメータ**

ID	tskid	R1	タスク ID

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(tskid≦0,tskid>タスク定義数)
E_NOEXS	H'ffcc(-H'34)	[p]	未登録(tskid のタスクが存在しない)
E_OBJ	H'ffc1(-H'3f)	[p]	オブジェクトの状態が不正(自タスク指定)
		[k]	(タスクが休止状態である)
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない)
E_QOVR	H'ffb7(-H'49)	[k]	キューイングのオーバフロー(既に強制待ち状態である)

- **解説**

tskid で示されたタスクの実行を中断させ、強制待ち状態へ移行します。
強制待ち状態は、rsm_tsk システムコールの発行により解除されます。
tskid で示されたタスクが待ち状態にある場合は、二重待ち状態へ移行します。

強制待ち状態は、他タスクによる中断状態を意味するため、tskid に自タスクを指定することはできません。
強制待ちの要求はキューイングしません。

3.5.2 強制待ち状態のタスクを再開(rsm_tsk) [T/D/L]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @rsm_tsk
```

- **パラメータ**

ID	tskid	R1	タスク ID

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(tskid≦0,tskid>タスク定義数)
E_NOEXS	H'ffcc(-H'34)	[p]	未登録(tskid のタスクが存在しない)
E_OBJ	H'ffc1(-H'3f)	[k]	オブジェクトの状態が不正(tskid のタスクは強制待ち状態でない)
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない)

- **解説**

tskid で示されたタスクが強制待ちの場合、強制待ち状態を解除し実行可能状態へ移行します。また、対象タスクが二重待ち状態の場合は、待ち状態へ移行します。

強制待ち要求はキューイングされませんので、必ず強制待ち状態を解除します。
tskid に自タスクを指定することはできません。

3.5.3 自タスクを起床待ち状態へ移行 (slp_tsk) [T]
(tslp_tsk) [T]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @slp_tsk
JSR @tslp_tsk
```

- **パラメータ**

TMO	tmout	ER2	タイムアウト指定<tslp_tsk>

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_RSFN	H'ffec(-H'14)	[p]	未サポート機能(タイマドライバ未使用(tslp_tsk))
E_PAR	H'ffdf(-H'21)	[p]	不正時間指定(tmout≦-2(tslp_tsk))
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない)
		[k]	(ディスパッチ禁止状態、CPU ロック状態から発行できない。
			tslp_tsk は tmout が TMO_POL(0)以外の場合)
E_TMOUT	H'ffab(-H'55)	[k]	タイムアウト(tslp_tsk)
E_RLWAI	H'ffaa(-H'56)	[k]	待ち状態強制解除
			(待ちの間に rel_wai システムコールが発行された)

- **解説**

自タスクを実行状態から待ち状態へ移行します。

この待ち状態は、wup_tsk システムコールの発行により解除され正常終了します。ただし、wup_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起床要求カウントを 1 減らしてそのまま実行を続けます。

tslp_tsk システムコールを使用する場合は、システムにタイマドライバを組込む必要があります。タイマドライバの組み込み方法は、「6.2.1 定数定義部の登録」を参照してください。

tslp_tsk システムコールの tmout には、待ち状態になる場合の待ち時間を指定します。

tmout に正の値を指定した場合は、待ち状態のまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。

tmout=TMO_POL(0)を指定した場合は、起床要求カウントが正ならの起床要求カウントを 1 減らして実行を継続し、0 ならエラーコードとして E_TMOUT を返します。

tmout=TMO_FEVR(-1)を指定した場合はタイムアウト監視を行いません。slp_tsk と同じ処理を行います。

本システムコールにより待ち状態となっている間に、sus_tsk システムコールが発行された場合、wup_tsk システムコールにより待ち状態が解除されても、まだ強制待ち状態であり、rsm_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。

3.5.4 **他タスクの起床**(wup_tsk) [T/D/L]
(iwup_tsk) [D/I]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @wup_tsk
JSR @iwup_tsk
```

- **パラメータ**

ID	tskid	R1	タスク ID

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(tskid≦0,tskid>タスク定義数)
E_NOEXS	H'ffcc(-H'34)	[p]	未登録(tskid のタスクが存在していない)
E_OBJ	H'ffc1(-H'3f)	[p]	オブジェクトの状態が不正(自タスク指定)
		[k]	(タスクが休止状態である)
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク実行状態(iwup_tsk)または
			タスク独立部(wup_tsk)から発行できない)
		[k]	(CPU ロック状態(iwup_tsk)から発行できない)
E_QOVR	H'ffb7(-H'49)	[k]	キューイングのオーバフロー(wupcnt>H'ff)

- **解説**

slp_tsk システムコールまたは tslp_tsk システムコールの発行により待ち状態になっているタスクの待ち状態を解除します。
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。

対象タスクが待ち状態でない(slp_tsk または tslp_tsk システムコールを発行していない)場合は、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 または tslp_tsk システムコールを発行したときに有効になります。

3.5.5 タスクの起床要求を無効化(can_wup) [T/D/L]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @can_wup
```

- **パラメータ**

ID	tskid	R1	タスク ID
INT	*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(C言語ｲﾝﾀﾌｪｰｽ)

- **リターンパラメータ**

INT	*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(C言語ｲﾝﾀﾌｪｰｽ)
---	wupcnt	R2	キューイングされていた起床要求回数(ｱｾﾝﾌﾞﾗｲﾝﾀﾌｪｰｽ)
ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_ID	H'ffdd(-H'23)	[p]	ID 範囲外(tskid<0,tskid>タスク定義数)
E_NOEXS	H'ffcc(-H'34)	[p]	未登録(tskid のタスクが生成されていない)
E_OBJ	H'ffc1(-H'3f)	[k]	タスクが休止状態である
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない)

- **解説**

tskid で示されたタスクにキューイングされていた起床要求を全て解除し、その起床要求回数をリターンパラメータとして返します。キューイングされていた起床要求が 0 の場合は 0 を返します。
tskid=TSK_SELF(0)の指定により、自タスクの指定になります。

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。
起床要求回数が 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ことになるため、それに対して何らかの処置をすることができます。

3.6 同期･通信(イベントフラグ)機能

(1) イベントフラグシステムコール

表 3.8 にイベントフラグでサポートしているシステムコール一覧を示します。

表 3.8 同期･通信(イベントフラグ)システムコール

項番	システムコール	機 能	発行可能なシステム状態			
			T	D	L	I
1	set_flg	イベントフラグのセット	○	○	○	
2	iset_flg	イベントフラグのセット(タスク独立部用)		○		○
3	clr_flg	イベントフラグのクリア	○	○	○	○
4	wai_flg	イベントフラグの待ち	○			
5	pol_flg	イベントフラグの待ち(ポーリング)	○	○	○	○
6	twai_flg	イベントフラグ待ち(タイムアウト有)	○			
7	ref_flg	イベントフラグ状態参照	○	○	○	○

(2) イベントフラグの仕様

表 3.9 にイベントフラグの仕様を示します。

表 3.9 イベントフラグの仕様

項番	項目	内容
1	イベントフラグパターンサイズ	16 ビットサイズ
2	イベントフラグ定義最大数	255 個
3	イベントフラグ ID	1～255
4	イベントフラグ初期値	0(固定)
5	イベントフラグ待ち行列	FIFO(First-In First-Out)の複数待ち

(3) タスクの実行中断と解除

タスクの実行中断と解除の要因を表 3.10 に示します

表 3.10 タスク実行中断と解除の要因

項番	実行中断の要因		中断解除の要因
1	自ら中断となる場合	wai_flg、twai_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)指定したタイムアウト時間(tmout)が経過するまで (twai_flg) (3)rel_wai システムコールが発行されるまで

3.6.1 イベントフラグのセット(set_flg) [T/D/L]
(iset_flg) [D/I]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @set_flg
JSR @iset_flg
```

- **パラメータ**

ID	flgid	R1	イベントフラグ ID
UINT	setptn	R2	セットするビットパターン

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(flgid≦0,flgid>イベントフラグ定義数)
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク実行状態(iset_flg)または
			タスク独立部(set_flg)から発行できない)
		[k]	(CPU ロック状態(iset_flg)から発行できない)

- **解説**

flgid で示されたイベントフラグに、setptn で示されたビットを論理和でセットします。
本システムコールにおいて、イベントフラグ値更新の結果、そのイベントフラグを待っているタスクの待ち解除条件を満たす場合、その待ち条件を満たすタスクすべての待ち状態を解除します。ただし、待ちモードに TWF_CLR(クリア)指定のあるタスクの待ちを解除した後は、イベントフラグのビットパターンがクリアされるため、それ以降の待ちタスクの待ち状態は解除されません。

setptn の全ビットを 0 とした場合、flgid で示されたイベントフラグに対して何も操作せず正常終了します。

3.6.2 イベントフラグのクリア(clr_flg) [T/D/L/I]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @clr_flg
```

- **パラメータ**

ID	flgid	R1	イベントフラグ ID
UINT	clrptn	R2	クリアするビットパターン

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(flgid≦0,flgid>イベントフラグ定義数)

- **解説**

flgid で示されたイベントフラグを clrptn の値で論理積をとり、clrptn の 0 になっているビットをクリアします。

本システムコールにおいて、イベントフラグ値更新の結果、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。

3.6.3 イベントフラグ待ち (wai_flg) [T]
(pol_flg) [T/D/L/I]
(twai_flg) [T]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @wai_flg
JSR @pol_flg
JSR @twai_flg
```

- **パラメータ**

UINT	*p_flgptn	---	待ち解除時のビットパターンを返す領域の先頭アドレス(C言語ｲﾝﾀﾌｪｰｽ)
ID	flgid	R1	イベントフラグ ID
UINT	waiptn	R2	待ちビットパターン
UINT	wfmode	R3	待ちモード
TMO	tmout	ER4	タイムアウト指定<twai_flg>

- **リターンパラメータ**

UINT	*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(C言語ｲﾝﾀﾌｪｰｽ)
---	flgptn	R2	待ち解除時のビットパターン(ｱｾﾝﾌﾞﾗｲﾝﾀﾌｪｰｽ)
ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_RSFN	H'ffec(-H'14)	[p]	未サポート機能(タイマドライバ未使用、またはタイムアウト機能が NOTUSE(twai_flg))
E_PAR	H'ffdf(-H'21)	[p]	パラメータエラー(waiptn=0、wfmode が不正)
		[p]	不正時間指定(tmout≦-2(twai_flg))
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(flgid≦0,flgid>イベントフラグ定義数)
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない (wai_flg,twai_flg))
		[k]	(ディスパッチ禁止状態、CPU ロック状態から発行できない (wai_flg,twai_flg)twai_flg は tmout が TMO_POL(0)以外の場合)

E_RLWAI	H'ffaa(-H'56)	[k]	待ち状態強制解除(待ちの間に rel_wai システムコールが発行された)
E_TMOUT	H'ffab(-H'55)	[k]	ポーリング失敗(pol_flg) タイムアウト(twai_flg)

- **解説**

flgid で示されたイベントフラグがセットされるまで、waiptn、wfmode で示された待ち条件に従って待ちます。

wfmode には、次のような指定を行います。

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

TWF_ANDW(H'0000)AND待ち
TWF_ORW(H'0002)OR待ち
TWF_CLR(H'0001)クリア指定

TWF_ANDW 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全てがセットされるのを待ちます。

TWF_ORW 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 指定がある場合、待ち条件が満た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)を 0 にクリアします。ただし、システムコールがエラーリターンした場合はイベントフラグの値はクリアされません。TWF_CLR 指定が無い場合は、条件が満たされてタスクが待ち解除になってもイベントフラグはクリアされません。

wai_flg、twai_flg システムコール発行時に、前述した条件が既に満たされている場合は正常終了し、満たされない場合はイベントフラグの待ち行列につながれます。なお、イベントフラグ待ち行列は、複数のタスクが事象を待つ事ができます。

pol_flg システムコールは、flgid で示されたイベントフラグがセットされていれば正常終了し、セットされていなければエラーコードとして E_TMOUT を返します。

条件が満たされると、その時のイベントフラグ値(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 に返します。

twai_flg システムコールで指定する tmout には、この待ち時間を指定します。

tmout に正の値を指定した場合は、待ち解除の条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。

tmout=TMO_POL(0)を指定した場合は、待ち状態にはならずに、flgid で示されたイベントフラグがセットされていれば正常終了し、セットされていなければエラーコードとして E_TMOUT を返します。pol_flg システムコールと同じ処理を行います。

tmout=TMO_FEVR(-1)を指定した場合は、タイムアウト監視を行いません。wai_flg システムコールと同じ処理を行います。

twai_flg システムコールを使用する場合、システムにタイマドライバを組み込み、さらにセットアップテーブルのタイムアウト機能を(USE)に設定してください。タイマドライバの組み込み方法、およびセットアップテーブルのタイムアウト機能の設定方法は、「6.2.1 定数定義部の登録」を参照してください。

3.6.4 イベントフラグ状態参照(ref_flg) [T/D/L/I]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @ref_flg
```

- **パラメータ**

ID	flgid	R1	イベントフラグ ID
T_RFLG	*pk_rflg	ER2/R2	イベントフラグ状態を返すパケットの先頭アドレス

- **リターンパラメータ**

T_RFLG	*pk_rflg	ER2/R2	イベントフラグ状態を格納したパケットの先頭アドレス
ER	ercd	R0	エラーコード

- **パケットの構造**

```
typedef     struct          t_rflg {
                            VP       exinf;    0/ 0  4/2   拡張情報
                            BOOL_ID  wtsk;    +4/+2  2/2   待ちタスク ID
                            UINT     flgptn;  +6/+4  2/2   イベントフラグの
                                                           ビットパターン
                            } T_RFLG;
```

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_PAR	H'ffdf(-H'21)	[p]	不正アドレス(pk_rflg が 0 または奇数)
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(flgid≦0,flgid>イベントフラグ定義数)

- **解説**

flgid で示されたイベントフラグの状態を参照し、pk_rflg の指す領域に拡張情報(exinf)、待ちタスク ID(wtsk)、および現在のイベントフラグのビットパターン(flgptn)を格納し返します。

pk_rflg の指す領域として 8 バイト(アドバンストモード)または 6 バイト(ノーマルモード)の RAM 領域が必要です。

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(0)を返します。

また、対象イベントフラグに複数の待ちタスクが存在する場合は、待ちタスク ID として、待ち行列の先頭タスク ID を返します。

3.7 同期･通信(セマフォ)機能

(1) セマフォシステムコール

表 3.11 にセマフォでサポートしているシステムコール一覧を示します。

表 3.11 同期･通信(セマフォ)システムコール

項番	システムコール	機 能	発行可能なシステム状態			
			T	D	L	I
1	sig_sem	セマフォ資源返却	○	○	○	
2	isig_sem	セマフォ資源返却(タスク独立部用)		○		○
3	wai_sem	セマフォ資源獲得	○			
4	preq_sem	セマフォ資源獲得(ポーリング)	○	○	○	○
5	twai_sem	セマフォ資源獲得(タイムアウト有)	○			
6	ref_sem	セマフォ状態参照	○	○	○	○

(2) セマフォの仕様

表 3.12 にセマフォの仕様を示します。

表 3.12 セマフォの仕様

項番	項目	内容
1	セマフォカウンタ最大値	65535
2	セマフォ定義最大数	255 個
3	セマフォ ID	1～255
4	セマフォカウンタ初期値	1(固定)
5	セマフォ待ち行列	FIFO(First-In First-Out)の複数待ち

(3) タスクの実行中断と解除

タスクの実行中断と解除の要因を表 3.13 に示します

表 3.13 タスク実行中断と解除の要因

項番	実行中断の要因		中断解除の要因
1	自ら中断となる場合	wai_sem,twai_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)指定したタイムアウト時間(tmout)が経過するまで (twai_sem) (3)rel_wai システムコールが発行されるまで

3.7.1 セマフォ資源返却 (sig_sem) [T/D/L]
(isig_sem) [D/I]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @sig_sem
JSR @isig_sem
```

- **パラメータ**

ID	semid	R1	セマフォ ID

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(semid≦0,semid>セマフォ定義数)
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク実行状態(isig_sem)または
			タスク独立部(sig_sem)から発行できない)
		[k]	(CPU ロック状態(isig_sem)から発行できない)
E_QOVR	H'ffb7(-H'49)	[k]	キューイングのオーバフロー
			(セマフォカウント値>H'ffff)

- **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先頭タスクの待ち状態を解除します。

セマフォ待ちタスクが存在しない場合、そのセマフォのカウント値を 1 増やします。

3.7.2 セマフォ資源獲得 (wai_sem) [T]
(preq_sem) [T/D/L/I]
(twai_sem) [T]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @wai_sem
JSR @preq_sem
JSR @twai_sem
```

- **パラメータ**

ID	semid	R1	セマフォ ID
TMO	tmout	ER4	タイムアウト指定<twai_sem>

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_RSFN	H'ffec(-H'14)	[p]	未サポート機能(タイマドライバ未使用、またはタイムアウト機能未使用(twai_sem))
E_PAR	H'ffdf(-H'21)	[p]	不正時間指定(tmout≦-2(twai_sem))
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(semid≦0,semid>セマフォ定義数)
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない (wai_sem,twai_sem))
		[k]	(ディスパッチ禁止状態、CPU ロック状態から発行できない (wai_sem,twai_sem)twai_sem は tmout が TMO_POL(0)以外の場合)
E_RLWAI	H'ffaa(-H'56)	[k]	待ち状態強制解除 (待ちの間に rel_wai システムコールが発行された)
E_TMOUT	H'ffab(-H'55)	[k]	ポーリング失敗(preq_sem)
		[k]	タイムアウト(twai_sem)

- **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、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。

wai_sem、twai_sem システムコールは、semid で示されたセマフォのカウント値が 1 以上の場合、カウント値を 1 減らして正常終了し、カウント値が 0 の場合、カウント値を変更せずにセマフォに対する待ち行列につながれ、待ち状態へ移行します。

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

twai_sem システムコールで指定する tmout には、その待ち時間を指定します。
tmout に正の値を指定した場合は、待ち解除の条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。

tmout=TMO_POL(0)を指定した場合は、待ち状態にはならずに、semid で示されたセマフォのカウント値が 1 以上の場合、カウント値を 1 減らして正常終了し、カウント値が 0 の場合、カウント値を変更せずエラーコードとして E_TMOUT を返します。preq_sem システムコールと同じ処理を行います。
tmout=TMO_FEVR(-1)を指定した場合は、タイムアウト監視を行いません。wai_sem システムコールと同じ処理を行います。

twai_sem システムコールを使用する場合、システムにタイマドライバを組み込み、さらにセットアップテーブルのタイムアウト機能を(USE)に設定してください。タイマドライバの組み込み方法、およびセットアップテーブルのタイムアウト機能の設定方法は、「6.2.1 定数定義部の登録」を参照してください。

3.7.3 セマフォ状態参照(ref_sem) [T/D/L/I]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @ref_sem
```

- **パラメータ**

ID	semid	R1	セマフォ ID
T_RSEM	*pk_rsem	ER2/R2	セマフォ状態を返すパケットの先頭アドレス

- **リターンパラメータ**

T_RSEM	*pk_rsem	ER2/R2	セマフォ状態を格納したパケットの先頭アドレス
ER	ercd	R0	エラーコード

- **パケットの構造**

```
typedef     struct          t_rsem {
                            VP       exinf;    0/ 0  4/2   拡張情報
                            BOOL_ID  wtsk;    +4/+2  2/2   待ちタスク ID
                            UINT     semcnt;  +6/+4  2/2   現在のセマフォカウント値
                            } T_RSEM;
```

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_PAR	H'ffdf(-H'21)	[p]	不正アドレス(pk_rsem が 0 または奇数)
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(semid≦0,semid>セマフォ定義数)

- **解説**

semid で示されたセマフォの状態を参照し、pk_rsem が指す領域に拡張情報(exinf)、待ちタスク ID(wtsk)、および現在のセマフォカウント値(semcnt)を格納し返します。

pk_rsem の指す領域として 8 バイト(アドバンストモード)または 6 バイト(ノーマルモード)の RAM 領域が必要です。

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(0)を返します。

また、対象セマフォに複数の待ちタスクが存在する場合は、待ちタスク ID として、待ち行列の先頭タスク ID を返します。

3.8 同期･通信(メイルボックス)機能

(1) メイルボックスシステムコール

表 3.14 にメイルボックスでサポートしているシステムコール一覧を示します。

表 3.14 同期･通信(メイルボックス)システムコール

項番	システムコール	機 能	発行可能なシステム状態			
			T	D	L	I
1	snd_msg	メイルボックスへ送信	○	○	○	
2	isnd_msg	メイルボックスへ送信(タスク独立部用)		○		○
3	rcv_msg	メイルボックスから受信	○			
4	prcv_msg	メイルボックスから受信(ポーリング)	○	○	○	○
5	trcv_msg	メイルボックスから受信(タイムアウト有)	○			
6	ref_mbx	メイルボックス状態参照	○	○	○	○

(2) メイルボックスの仕様

表 3.15 にメイルボックスの仕様を示します。

表 3.15 メイルボックスの仕様

項番	項目	内容
1	メイルボックス定義最大数	255 個
2	メイルボックス ID	1～255
3	メッセージ待ち行列	FIFO(First-In First-Out)の複数待ち
4	メッセージ	メッセージ先頭から 4 バイトはカーネルで使用します メッセージ送信時は必ず 0 を設定してください メッセージは RAM エリアに作成しなければなりません

(3) タスクの実行中断と解除

タスクの実行中断と解除の要因を表 3.16 に示します

表 3.16 タスク実行中断と解除の要因

項番	実行中断の要因		中断解除の要因
1	自ら中断となる場合	rcv_msg,trcv_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)指定したタイムアウト時間(tmout)が経過するまで (trcv_msg) (3)rel_wai システムコールが発行されるまで

3.8.1 メイルボックスへ送信 (snd_msg) [T/D/L]
(isnd_msg) [D/I]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @snd_msg
JSR @isnd_msg
```

- **パラメータ**

ID	mbxid	R1	メイルボックス ID
T_MSG	*pk_msg	ER2/R2	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	R0	エラーコード

- **パケットの構造**

【注】T_MSG **構造体の構造はユーザ依存であるため標準ヘッダファイルでは定義していません。必要に応じてユーザが定義してください。**

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_PAR	H'ffdf(-H'21)	[p]	不正アドレス(メッセージ先頭アドレスが 0 または奇数)
		[K]	不正メッセージ形式(メッセージ先頭 4 バイトが 0 以外)
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(mbxid≦0,mbxid>メイルボックス定義数)
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク実行状態(isnd_msg)または タスク独立部(snd_msg)から発行できない)
		[k]	(CPU ロック状態(isnd_msg)から発行できない)

- **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、メッセージ受信時、メッセージの先頭アドレス(pk_msg の値)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替えるとrcv_msg, prcv_msg または trcv_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りますので注意してください。

対象メイルボックスに受信待ちタスクがある場合、受信待ち行列の先頭タスクにメッセージを渡し、そのタスクの待ち状態を解除します。受信待ちタスクが無い場合、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。

メッセージは、必ず RAM 領域に確保してください。メッセージの長さはカーネル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 バイトはカーネルで使用するため、メッセージ送信時は、メッセージの先頭 4 バイトに初期値として必ず 0 を設定し、送信後は書き替えないでください。

図 3.2 にメッセージ形式を示します。

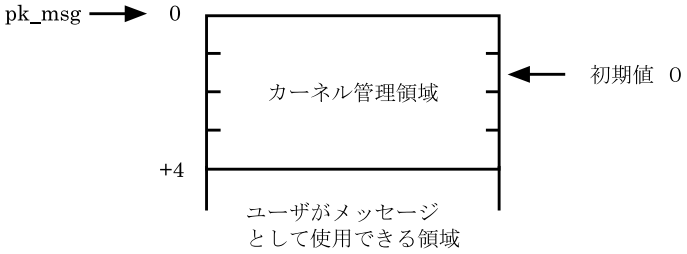


図 3.2 メッセージの形式

3.8.2 メイルボックスから受信 (rcv_msg) [T]
(prcv_msg) [T/D/L/I]
(trcv_msg) [T]

- **C言語インタフェース**

```
ER ercd = rcv_msg(T_MSG **ppk_msg, ID mbxid);
ER ercd = prcv_msg(T_MSG **ppk_msg, ID mbxid);
ER ercd = trcv_msg(T_MSG **ppk_msg, ID mbxid, TMO tmout);
```

- **アセンブラインタフェース**

```
JSR @rcv_msg
JSR @prcv_msg
JSR @trcv_msg
```

- **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(C言語ｲﾝﾀﾌｪｰｽ)
ID	mbxid	R1	メイルボックス ID
TMO	tmout	ER4	タイムアウト指定<trcv_msg>

- **リターンパラメータ**

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(C言語ｲﾝﾀﾌｪｰｽ)
---	*pk_msg	ER2/R2	受信メッセージの先頭アドレス(ｱｾﾝﾌﾞﾗｲﾝﾀﾌｪｰｽ)
ER	ercd	R0	エラーコード

- **パケットの構造**

【注】 T_MSG **構造体の構造はユーザ依存であるため標準ヘッダファイルでは定義していません。必要に応じてユーザが定義してください。**

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_RSFN	H'ffec(-H'14)	[p]	未サポート機能(タイマドライバ未使用、またはタイムアウト機能未使用)(trcv_msg)
E_PAR	H'ffdf(-H'21)	[p]	不正時間指定(tmout≦-2)(trcv_msg)
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(mbxid≦0,mbxid>メイルボックス定義数)
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない(rcv_msg,trcv_msg))
		[k]	(ディスパッチ禁止状態、CPU ロック状態から発行できない(rcv_msg,trcv_msg)trcv_msg は tmout が TMO_POL(0)以外の場合)

E_RLWAI	H'ffaa(-H'56)	[k]	待ち状態強制解除(待ちの間に rel_wai システムコールが発行された)
E_TMOUT	H'ffab(-H'55)	[k]	ポーリング失敗(prsv_msg) タイムアウト(trcv_msg)

- **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 の指す領域に格納し、発行タスクの実行を継続します。

rcv_msg、trcv_msg システムコールは、mbxid で示されたメイルボックスにメッセージがある場合、メッセージの先頭アドレスを ppk_msg の指す領域に格納して正常終了し、メッセージがない場合、メッセージの到着を待つ待ち行列(受信待ち行列)につながれます。待ち行列のつながれ方は、FIFO(First-InFirst-Out)になります。

prcv_msg システムコールは、mbxid で示されたメイルボックスにメッセージがある場合、メッセージの先頭アドレスを ppk_msg の指す領域に格納して正常終了し、メッセージがない場合、エラーコードとして E_TMOUT を返します。

trcv_msg システムコールで指定する tmout には、その待ち時間を指定します。
tmout に正の値を指定した場合は、待ち解除の条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。

tmout=TMO_POL(0)を指定した場合は、待ち状態にはならずに、mbxid で示されたメイルボックスにメッセージがある場合、メッセージの先頭アドレスを ppk_msg の指す領域に格納して正常終了し、メッセージがない場合、エラーコードとして E_TMOUT を返します。prcv_msg システムコールと同じ処理を行います。
tmout=TMO_FEVR(-1)を指定した場合は、タイムアウト監視を行いません。rcv_msg システムコールと同じ処理を行います。

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。

【注】 メッセージの先頭 4 バイトはカーネルで使用するため、この領域は書き替えないでください。メッセージ送信後、メッセージが受信される前にこの領域を書き替えた場合、システムの正常な動作は保証されません。

trcv_msg システムコールを使用する場合、システムにタイマドライバを組み込み、さらにセットアップテーブルのタイムアウト機能を(USE)に設定してください。タイマドライバの組み込み方法、およびセットアップテーブルのタイムアウト機能の設定方法は、「6.2.1 定数定義部の登録」を参照してください。

3.8.3 メイルボックス状態参照(ref_mbx) [T/D/L/I]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @ref_mbx
```

- **パラメータ**

ID	mbxid	R1	メイルボックス ID
T_RMBX	*pk_rmbx	ER2/R2	メイルボックス状態を返すパケットの先頭アドレス

- **リターンパラメータ**

T_RMBX	*pk_rmbx	ER2/R2	メイルボックス状態を格納したパケットの先頭アドレス
ER	ercd	R0	エラーコード

- **パケットの構造**

```
typedef     struct          t_rmbx {
                            VP       exinf;    0/ 0  4/2  拡張情報
                            BOOL_ID  wtsk;    +4/+2  2/2  待ちタスク ID
                            T_MSG    *pk_msg; +6/+4  4/2  次に受信されるメッセージ
                                                          の先頭アドレス
                            } T_RMBX;
```

【注】T_MSG 構造体の構造はユーザ依存であるため標準ヘッダファイルでは定義していません。必要に応じてユーザが定義してください。

- **エラーコード**

E_OK	H'00000	[k]	正常終了
E_PAR	H'ffdf(-H'21)	[p]	不正アドレス(pk_rmbx が 0 または奇数)
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(mbxid≦0,mbxid>メイルボックス定義数)

- **解説**

mbxid で示されたメイルボックスの状態を参照し、pk_rmbx が示す領域に拡張情報(exinf)、待ちタスク ID(wtsk)、次に受信されるメッセージの先頭アドレス(pk_msg)を格納し返します。

pk_rmbx の指す領域として 10 バイト(アドバンストモード)または 6 バイト(ノーマルモード)の RAM 領域が必要です。

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(0)を返します。

また、対象メイルボックスに複数の待ちタスクが存在する場合は、待ちタスク ID として、待ち行列の先頭タスク ID を返します。

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(-1)を返します。

3.9 割込み管理機能

(1) 割込み管理システムコール

表 3.17 に割込み管理でサポートしているシステムコール一覧を示します。

表 3.17 割込み管理システムコール

項番	システムコール	機 能	発行可能なシステム状態			
			T	D	L	I
1	ret_int	割込みハンドラから復帰				○
2	chg_ims	割込みマスク変更	○			○
3	ref_ims	割込みマスク参照	○	○	○	○
4	loc_cpu	割込みとディスパッチの禁止	○	○	○	
5	unl_cpu	割込みとディスパッチの許可	○	○	○	

(2) 割込み制御モードと割込みマスクレベル値

カーネルは、H8S シリーズマイコンの持つ 4 種類の割込み制御モードで使用することが可能です。

表 3.18、3.19、3.20、3.21、に各割込み制御モードの割込みマスクレベルを示します。割込み制御モード、および CCR 値、EXR 値、受け付ける割込みに関する詳細は、当該ハードウェアマニュアルを参照してください。

表 3.18 割込み制御モード 0 の割込みマスクレベル

割込みﾏｽｸﾚﾍﾞﾙ値 (imask)	CCR 値		EXR 値			受け付ける割込み
	I	UI	I2	I1	I0	
1	1	－	－	－	－	NMI のみ
0	0	－	－	－	－	すべて

表 3.19 割込み制御モード 1 の割込みマスクレベル

割込みﾏｽｸﾚﾍﾞﾙ値 (imask)	CCR 値		EXR 値			受け付ける割込み
	I	UI	I2	I1	I0	
3	1	1	－	－	－	NMI のみ
2	1	0	－	－	－	コントロールレベル 1
1	0	1	－	－	－	すべて
0	0	0	－	－	－	すべて

表 3.20　割込み制御モード 2 の割込みマスクレベル

割込みﾏｽｸﾚﾍﾞﾙ値 (imask)	CCR 値		EXR 値			受け付ける割込み
	I	UI	I2	I1	I0	
7	－	－	1	1	1	NMI のみ
6	－	－	1	1	0	プライオリティレベル 7
5	－	－	1	0	1	プライオリティレベル 6～7
4	－	－	1	0	0	プライオリティレベル 5～7
3	－	－	0	1	1	プライオリティレベル 4～7
2	－	－	0	1	0	プライオリティレベル 3～7
1	－	－	0	0	1	プライオリティレベル 2～7
0	－	－	0	0	0	すべて

表 3.21　割込み制御モード 3 の割込みマスクレベル

割込みﾏｽｸﾚﾍﾞﾙ値 (imask)	CCR 値		EXR 値			受け付ける割込み
	I	UI	I2	I1	I0	
8	1	1	1	1	1	NMI のみ
7	1	0	－	－	－	コントロールレベル 1
6	0	0	1	1	0	プライオリティレベル 7／ コントロールレベル 0、1
5	0	0	1	0	1	プライオリティレベル 6～7／ コントロールレベル 0、1
4	0	0	1	0	0	プライオリティレベル 5～7／ コントロールレベル 0、1
3	0	0	0	1	1	プライオリティレベル 4～7／ コントロールレベル 0、1
2	0	0	0	1	0	プライオリティレベル 3～7／ コントロールレベル 0、1
1	0	0	0	0	1	プライオリティレベル 2～7／ コントロールレベル 0、1
0	0	0	0	0	0	すべて

【注】　割込み制御モード 3 で、カーネル割込みマスクレベルとして割込みマスクレベル 7 を使用する場合、コントロールレベル 1 の割込みはカーネル割込みマスクレベルより高い割込みとなるため、システムコールの発行はできません。

3.9.1 割込みハンドラから復帰(ret_int) [I]

- **C言語インタフェース**

なし(C コンパイラの拡張機能#pragma interrupt を用いて ret_int を発行します)

- **アセンブラインタフェース**

```
JMP @ret_int
```

- **パラメータ**

なし

- **リターンパラメータ**

なし

- **エラーコード**

正常終了時： [k] 本システムコールの発行元には戻りません。

異常終了時： [p] タスク実行状態から発行した場合、システム異常終了処理に制御を移します。

[k] CPU ロック状態から発行した場合、システム異常終了処理に制御を移します。

- **解説**

割込みハンドラから復帰する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替えが必要となっても、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。

【注】本システムコールを発行する場合は、スタックポインタ、レジスタの内容は割込みハンドラが起動された時の状態と同じでなければなりません。割込みハンドラで使用するレジスタの退避や復帰をユーザ側で必ず行ってください。

3.9.2 割込みマスクレベル変更(chg_ims) [T/I]

- **C 言語インタフェース**

```
ER ercd = chg_ims(UINT imask);
```

- **アセンブラインタフェース**

```
JSR @chg_ims
```

- **パラメータ**

UINT	imask	R1	割込みマスク値 割込み制御モード 0：CR_IMS0～CR_IMS1(H'0~H'1) 割込み制御モード 1：CR_IMS0～CR_IMS3(H'0~H'3) 割込み制御モード 2：CR_IMS0～CR_IMS7(H'0~H'7) 割込み制御モード 3：CR_IMS0～CR_IMS8(H'0~H'8)

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_PAR	H'ffdf(-H'21)	[p]	パラメータエラー(imask に範囲外の値を指定した)
E_CTX	H'ffbb(-H'45)	[k]	コンテキストエラー (ディスパッチ禁止状態、CPU ロック状態から発行できない)

- **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。imask には、割込み制御モードに応じて、CR_IMSn(n：0～8)を指定します。

割込みマスクは、CCR、EXR に直接値を設定して割込みの禁止・解除を行います。

割込みマスク値（imask）と CCR、EXR 値の対応は、表 3.18、3.19、3.20、3.21 を参照してください。

本システムコールにより割込みをマスクした場合は、システム状態はタスク独立部となります。したがって、待ち状態へ移行するシステムコール、タスク部専用システムコールは発行できません。

本システムコールによってタスク実行状態からタスク独立部に遷移した場合、タスク実行状態に戻るには、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。タスク独立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替えが必要になっても、本システムコールによって割込みマスクを CR_IMS0(H'0)に戻す(タスク実行状態に戻る)まで遅延されます。

【注】 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いレベルに変更している間は、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。

3.9.3 割込みマスクレベル参照(ref_ims) [T/D/L/I]

- **C言語インタフェース**

```
ER ercd = ref_ims(UINT *p_imask);
```

- **アセンブラインタフェース**

```
JSR @ref_ims
```

- **パラメータ**

UINT	*p_imask	---	割込みマスクレベルを返す領域の先頭アドレス (C言語ｲﾝﾀﾌｪｰｽ)

- **リターンパラメータ**

UINT	*p_imask	---	割込みマスクレベルを格納した領域の先頭アドレス (C言語ｲﾝﾀﾌｪｰｽ)
---	imask	R1	割込みマスクレベル(ｱｾﾝﾌﾞﾗｲﾝﾀﾌｪｰｽ)
ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了

- **解説**

現在の割込みマスクレベルを返します。
割込み制御モードによって、割込みマスクレベルとして使用できる値の範囲と内容が異なります。

3.9.4 割込みとディスパッチの禁止(loc_cpu) [T/D/L]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @loc_cpu
```

- **パラメータ**

なし

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない)

- **解説**

システム状態を CPU ロック状態とし、割込みとタスクのディスパッチを禁止します。
CPU ロック状態からは、unl_cpu システムコールによってタスク実行状態に戻ります。
CPU ロック状態の特長を以下に示します。

(1)タスクのスケジューリングが行われなくなるため、自タスク以外のタスクが実行状態に移行することはなくなります。タスク実行状態に戻った時点で、タスクのスケジューリングが行われます。
(2)セットアップテーブルに定義するカーネル割込みマスクレベルで、割込みをマスクします。このレベル以下の割込みは受け付けられなくなります。
(3)待ち状態になるシステムコールは発行できません。

dis_dsp システムコールによるディスパッチ禁止状態から、本システムコールによって CPU ロック状態に移行することができますが､CPU ロック状態からディスパッチ禁止状態に戻ることはできません。CPU ロック状態から ena_dsp システムコールでディスパッチ禁止状態に戻ろうとすると、エラーコード E_CTX が返されます。

CPU ロック状態の間に ext_tsk システムコールによってタスクが終了した場合は、システム状態はタスク実行状態に戻ります。

すでに CPU ロック状態のときに、再度本システムコールを発行してもエラーにはなりませんが、キューイングは行いません。

表 3.22 に、システム状態の遷移を示します。この表において、矢印の次の数字は遷移する状態番号を示します。例えば、状態番号 1 で dis_dsp システムコールを発行すると状態番号 2 の状態に遷移します。

表 3.22　dis_dsp,loc_cpu **システムコールによる状態遷移**

状態番号	システム状態	現在の状態		発行するシステムコール			
		割込み	ディスパッチ	dis_dsp	ena_dsp	loc_cpu	unl_cpu
1	**タスク実行状態**(TSS_TSK)	**許可**	**許可**	→2	→1	→3	→1
2	**ディスパッチ禁止状態** (TSS_DDSP)	**許可**	**禁止**	→2	→1	→3	→1
3	CPU **ロック状態**(TSS_LOC)	**禁止**	**禁止**	E_CTX	E_CTX	→3	→1

【注】 (1) CPU **ロック状態における割込みマスクレベルは、セットアップファイルに定義する「カーネル割込みマスクレベル」となります。したがって、カーネル割込みマスクレベルより高いレベルの割込みは、**CPU **ロック状態においても受け付けられます。**

(2) **システム状態がディスパッチ禁止状態、または** CPU **ロック状態の時、直接レジスタを操作して、割込みマスクを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。**

3.9.5 割込みとディスパッチの許可(unl_cpu) [T/D/L]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @unl_cpu
```

- **パラメータ**

なし

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない)

- **解説**

割込みとタスクのディスパッチを許可します。

loc_cpu システムコールによって設定されていた CPU ロック状態を解除し、システム状態をタスク実行状態にします。そして、タスクのディスパッチ(スケジューリング)が行われます。

本システムコールは、CPU ロック状態を解除してタスク実行状態に戻すために用いますが、ディスパッチ禁止状態から本システムコールを発行しても同様にタスク実行状態に戻ります。

タスク実行状態から本システムコールを発行してもエラーにはなりませんが、キューイングは行いません。

3.10 メモリプール管理(固定長メモリプール)機能

(1) 固定長メモリプールシステムコール

表 3.23 に固定長メモリプールでサポートしているシステムコール一覧を示します。

表 3.23 固定長メモリプールシステムコール

項番	システムコール	機 能	発行可能なシステム状態			
			T	D	L	I
1	get_blf	固定長メモリブロック獲得	○			
2	pget_blf	固定長メモリブロック獲得(ポーリング)	○	○	○	○
3	tget_blf	固定長メモリブロック獲得(タイムアウト有)	○			
4	rel_blf	固定長メモリブロック返却	○	○	○	
5	ref_mpf	固定長メモリプール状態参照	○	○	○	○

(2) 固定長メモリプールの仕様

表 3.24 に固定長メモリプールの仕様を示します。

表 3.24 固定長メモリプールの仕様

項番	項目	内容
1	固定長メモリプール定義最大数	255 個
2	固定長メモリプール ID	1～255(未登録有り)
3	メモリブロック数	65535 個
4	メモリブロックサイズ	2～65530 バイト
5	メモリブロック待ち行列	FIFO(First-In First-Out)の複数待ち

(3) タスクの実行中断と解除

タスクの実行中断と解除の要因を表 3.25 に示します

表 3.25 タスク実行中断と解除の要因

項番	実行中断の要因		中断解除の要因
1	自ら中断となる場合	get_blf,tget_blf システムコール	(1)メモリブロックが獲得できるまで (2)指定したタイムアウト時間(tmout)が経過するまで (tget_blf) (3)rel_wai システムコールが発行されるまで

3.10.1 固定長メモリブロック獲得 (get_blf) [T]
(pget_blf) [T/D/L/I]
(tget_blf) [T]

- **C言語インタフェース**

```
ER ercd = get_blf(VP *p_blf, ID mpfid);
ER ercd = pget_blf(VP *p_blf, ID mpfid);
ER ercd = tget_blf(VP *p_blf, ID mpfid, TMO tmout);
```

- **アセンブラインタフェース**

```
JSR @get_blf
JSR @pget_blf
JSR @tget_blf
```

- **パラメータ**

VP	*p_blf	---	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(C言語ｲﾝﾀﾌｪｰｽ)
ID	mpfid	R1	固定長メモリプール ID
TMO	tmout	ER4	タイムアウト指定(tget_blf)

- **リターンパラメータ**

VP	*p_blf	---	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(C言語ｲﾝﾀﾌｪｰｽ)
---	blf	ER2/R2	メモリブロックの先頭アドレス(ｱｾﾝﾌﾞﾗｲﾝﾀﾌｪｰｽ)
ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_RSFN	H'ffec(-H'14)	[p]	未サポート機能(タイマドライバ未使用、またはタイムアウト機能未使用(tget_blf))
E_PAR	H'ffdf(-H'21)	[p]	パラメータエラー(tmout≦-2(tget_blf))
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(mpfid≦0,mpfid>メモリプール定義数)
E_NOEXS	H'ffcc(-H'34)	[p]	未登録(mpfid の固定長メモリプールが存在しない)
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない (get_blf,tget_blf))
		[k]	(ディスパッチ禁止状態、CPU ロック状態から発行できない (get_blf,tget_blf)tget_blf は tmout が TMO_POL(0)以外の場合)
E_RLWAI	H'ffaa(-H'56)	[k]	待ち状態強制解除 (待ちの間に rel_wai システムコールが発行された)

E_TMOUT	H'ffab(-H'55)	[k]	ポーリング失敗(pget_blf) タイムアウト(tget_blf)

- **解説**

mpfid で示される固定長メモリプールからひとつの固定長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_blf に格納後、発行タスクの実行を継続します。

get_blf、tget_blf システムコールは、mpfid で示される固定長メモリプールに空きメモリブロックがある場合、ひとつのメモリブロックの先頭アドレスを p_blf に格納して正常終了し、固定長メモリプールに空きメモリブロック獲得待ちタスクが存在する場合、または待ちタスクは存在しないが空きメモリブロックもない場合は、メモリ獲得待ち行列につながれます。待ち行列のつながり方は、FIFO(First-InFirst-Out)になります。

pget_blf システムコールは、mpfid で示される固定長メモリプールに空きメモリブロックがある場合、ひとつのメモリブロックの先頭アドレスを p_blf に格納して正常終了し、固定長メモリプールに空きメモリブロック獲得待ちタスクが存在する場合、または待ちタスクは存在しないが空きメモリブロックもない場合は、エラーコードとして E_TMOUT を返します。

tget_blf システムコールの tmout には、この待ち時間を指定します。
tmout に正の値を指定した場合は、待ち解除の条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。

tmout=TMO_POL(0)を指定した場合は、待ち状態にはならずに、mpfid で示される固定長メモリプールに空きメモリブロックがある場合、ひとつのメモリブロックの先頭アドレスを p_blf に格納して正常終了し、固定長メモリプールに空きメモリブロック獲得待ちタスクが存在する場合、または待ちタスクは存在しないが空きメモリブロックもない場合は、エラーコードとして E_TMOUT を返します。pget_blf システムコールと同じ処理を行います。
tmout=TMO_FEVR(-1)を指定した場合は、タイムアウト監視を行いません。get_blf システムコールと同じ処理を行います。

メモリブロックの獲得により、固定長メモリプールの空きは以下の式で算出されるサイズだけ減少します。
(減少サイズ)＝(登録時に指定したブロックサイズ)+4

tget_blf システムコールを使用する場合、システムにタイマドライバを組み込み、さらにセットアップテーブルのタイムアウト機能を(USE)に設定してください。タイマドライバの組み込み方法、およびセットアップテーブルのタイムアウト機能の設定方法は、「6.2.1 定数定義部の登録」を参照してください。

3.10.2 固定長メモリブロック返却(rel_blf) [T/D/L]

- **C言語インタフェース**

```
ER ercd = rel_blf(ID mpfid, VP blf);
```

- **アセンブラインタフェース**

```
JSR @rel_blf
```

- **パラメータ**

ID	mpfid	R1	固定長メモリプール ID
VP	blf	ER2/R2	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	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_PAR	H'ffdf(-H'21)	[p]	不正アドレス(blf が 0 または奇数)
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(mpfid≦0,mpfid>メモリプール定義数)
E_NOEXS	H'ffcc(-H'34)	[p]	未登録(mpfid の固定長メモリプールが存在していない)
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない)
EV_ILBLK	H'ff1e(-H'e2)	[k]	不正メモリブロックの返却、操作(blf がメモリプールの領域以外、または既に返却した blf を指定)

- **解説**

mpfid で示された固定長メモリプールへ blf で示されたメモリブロックを返却します。
blf には、get_blf,pget_blf または tget_blf 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を指定してください。

mpfid で示された固定長メモリプールに対して、メモリブロック獲得待ちタスクがある場合、返却アドレスを待ち行列の先頭タスクに渡し、待ちタスクの待ち状態を解除します。

3.10.3 固定長メモリプール状態参照(ref_mpf) [T/D/L/I]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @ref_mpf
```

- **パラメータ**

ID	mpfid	R1	固定長メモリプール ID
T_RMPF	*pk_rmpf	ER2/R2	固定長メモリプール状態を返すパケットの先頭アドレス

- **リターンパラメータ**

T_RMPF	*pk_rmpf	ER2/R2	固定長メモリプール状態を格納したパケットの先頭アドレス
ER	ercd	R0	エラーコード

- **パケットの構造**

```
typedef     struct          t_rmpf {
                            VP      exinf;    0/ 0  4/2  拡張情報
                            BOOL_ID wtsk;    +4/+2  2/2  待ちタスク ID
                            INT     frbcnt;  +6/+4  2/2  空き領域のブロック数
                            INT     mpfcnt;  +8/+6  2/2  メモリプール全体のブロック数
                            INT     blfsz;  +10/+8  2/2  固定長メモリブロックサイズ
                            } T_RMPF;
```

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_PAR	H'ffdf(-H'21)	[p]	不正アドレス(pk_rmpf が 0 または奇数)
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(mpfid≦0,mpfid>メモリプール定義数)
E_NOEXS	H'ffcc(-H'34)	[p]	未登録(mpfid の固定長メモリプールが存在していない)

- **解説**

mpfid で示された固定長メモリプールの状態を参照し、pk_rmpf の指す領域に拡張情報(exinf)、待ちタスク ID(wtsk)、空き領域のブロック数(frbcnt)、メモリプール全体のブロック数(mpfcnt)、および固定長メモリブロックサイズ(blfsz)を格納し返します。

pk_rmpf の指す領域として 12 バイト(アドバンストモード)または 10 バイト(ノーマルモード)の RAM 領域が必要です。

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(0)を返します。

また、対象メモリプールに複数の待ちタスクが存在する場合は、待ちタスク ID として、待ち行列の先頭タスク ID を返します。

3.11 メモリプール管理(可変長メモリプール)機能

(1) 可変長メモリプールシステムコール

表 3.26 に可変長メモリプールでサポートしているシステムコール一覧を示します。

表 3.26 可変長メモリプールシステムコール

項番	システムコール	機 能	発行可能なシステム状態			
			T	D	L	I
1	get_blk	可変長メモリブロック獲得	○			
2	pget_blk	可変長メモリブロック獲得(ポーリング)	○	○	○	○
3	tget_blk	可変長メモリブロック獲得(タイムアウト有)	○			
4	rel_blk	可変長メモリブロック返却	○	○	○	
5	ref_mpl	可変長メモリプール状態参照	○	○	○	○

(2) 可変長メモリプールの仕様

表 3.27 に可変長メモリプールの仕様を示します。

表 3.27 可変長メモリプールの仕様

項番	項目	内容
1	可変長メモリプール定義最大数	255 個
2	可変長メモリプール ID	1～255
3	メモリブロック待ち行列	FIFO(First-In First-Out)の複数待ち

(3) タスクの実行中断と解除

タスクの実行中断と解除の要因を表 3.28 に示します

表 3.28 タスク実行中断と解除の要因

項番	実行中断の要因		中断解除の要因
1	自ら中断となる場合	get_blk,tget_blk システムコール	(1)メモリブロックが獲得できるまで (2)指定したタイムアウト時間(tmout)が経過するまで (tget_blk) (3)rel_wai システムコールが発行されるまで

(4) 可変長メモリプールのフラグメンテーション

ひとつの可変長メモリプールからメモリブロックの獲得と返却を繰り返していると、メモリプール内の空き領域のフラグメンテーションが発生し、最大の空き領域サイズが減少します。しかし、本カーネルには、フラグメンテーションを整理する機能はありません。

また、絶対に返却されないメモリブロックが存在している場合、そのメモリブロックが「壁」となって最大の空き領域サイズが「あるサイズ」以上に増えない状況に陥る可能性があります。このような事態をある程度避けるためには、返却される予定のないメモリブロックは、可変長メモリプールの最初(返却する可能性のあるメモリブロックの獲得前)に、獲得するようにしてください。

3.11.1 可変長メモリブロック獲得 (get_blk) [T]
(pget_blk) [T/D/L/I]
(tget_blk) [T]

- **C言語インタフェース**

```
ER ercd = get_blk(VP *p_blk, ID mplid, UW blksz);
ER ercd = pget_blk(VP *p_blk, ID mplid, UW blksz);
ER ercd = tget_blk(VP *p_blk, ID mplid, UW blksz, TMO tmout);
```

- **アセンブラインタフェース**

```
JSR @get_blk
JSR @pget_blk
JSR @tget_blk
```

- **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(C言語ｲﾝﾀﾌｪｰｽ)
ID	mplid	R1	可変長メモリプール ID
UW	blksz	ER2	メモリブロックサイズ(バイト数)
TMO	tmout	ER4	タイムアウト指定(tget_blk)

- **リターンパラメータ**

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(C言語ｲﾝﾀﾌｪｰｽ)
---	blk	ER2/R2	メモリブロックの先頭アドレス (ｱｾﾝﾌﾞﾗｲﾝﾀﾌｪｰｽ)
ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_RSFN	H'ffec(-H'14)	[p]	未サポート機能(タイマドライバ未使用、またはタイムアウト機能未使用(tget_blk))
E_PAR	H'ffdf(-H'21)	[p]	パラメータエラー(tmout≦-2(tget_blk)) (blksz が 0 または奇数、mplsz<blksz)
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(mplid≦0,mplid>メモリプール定義数)
E_NOEXS	H'ffcc(-H'34)	[p]	未登録(mplid の可変長メモリプールが存在しない)
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない (get_blk,tget_blk))
		[k]	(ディスパッチ禁止状態、CPU ロック状態から発行できない (get_blk,tget_blk)tget_blk は tmout が TMO_POL(0)以外の場合))

E_RLWAI	H'ffaa(-H'56)	[k]	待ち状態強制解除 (待ちの間に rel_wai システムコールが発行された)
E_TMOUT	H'ffab(-H'55)	[k]	ポーリング失敗(pget_blk) タイムアウト(tget_blk)

- **解説**

mplid で示される可変長メモリプールから、blksz で示されるサイズ(バイト数)の空きメモリ領域(blksz+16 バイト以上)がある場合は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 の指す領域に格納後、発行タスクの実行を継続します。

get_blk,tget_blk システムコールは、mplid で示される可変長メモリプールに blksz で示されるサイズ(バイト数)の空きメモリ領域(blksz+16 バイト以上)がある場合は、メモリブロックの先頭アドレスを p_blk の指す領域に格納して正常終了し、可変長メモリプールに空きメモリ獲得待ちタスクが存在する場合、または待ちタスクは存在しないが blksz で示されるサイズ(バイト数)の空きメモリ領域(blksz+16 バイト以上)もない場合は、空きメモリ獲得待ち行列につながれます。待ち行列のつながり方は、FIFO(First-InFirst-Out)になります。

pget_blk システムコールは、mplid で示される可変長メモリプールに blksz で示されるサイズ(バイト数)の空きメモリ領域(blksz+16 バイト以上)がある場合は、メモリブロックの先頭アドレスを p_blk の指す領域に格納して正常終了し、可変長メモリプールに空きメモリ獲得待ちタスクが存在する場合、または待ちタスクは存在しないが blksz で示されるサイズ(バイト数)の空きメモリ領域(blksz+16 バイト以上)もない場合は、エラーコードとして E_TMOUT を返します。

tget_blk システムコールの tmout には、この待ち時間を指定します。
tmout に正の値を指定した場合は、待ち解除の条件が満たされないまま tmout 時間が経過すると、エラーコードとして E_TMOUT を返します。

tmout=TMO_POL(0)を指定した場合は、待ち状態にはならずに、mplid で示される可変長メモリプールに blksz で示されるサイズ(バイト数)の空きメモリ領域(blksz+16 バイト以上)がある場合は、メモリブロックの先頭アドレスを p_blk の指す領域に格納して正常終了し、可変長メモリプールに空きメモリ獲得待ちタスクが存在する場合、または待ちタスクは存在しないが blksz で示されるサイズ(バイト数)の空きメモリ領域(blksz+16 バイト以上)もない場合は、エラーコードとして E_TMOUT を返します。pget_blk システムコールと同じ処理を行います。
tmout=TMO_FEVR(-1)を指定した場合はタイムアウト監視を行いません。get_blk システムコールと同じ処理を行います。

メモリブロックの獲得により、可変長メモリプールの空きは以下の式で算出されるサイズだけ減少します。
(減少サイズ)＝blksz+16

tget_blk システムコールを使用する場合、システムにタイマドライバを組み込み、さらにセットアップテーブルのタイムアウト機能を(USE)に設定してください。タイマドライバの組み込み方法、およびセットアップテーブルのタイムアウト機能の設定方法は、「6.2.1 定数定義部の登録」を参照してください。

3.11.2 可変長メモリブロック返却(rel_blk) [T/D/L]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @rel_blk
```

- **パラメータ**

ID	mplid	R1	可変長メモリプール ID
VP	blk	ER2/R2	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	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_PAR	H'ffdf(-H'21)	[p]	不正アドレス(blk が 0 または奇数)
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(mplid≦0, mplid>メモリプール定義数)
E_NOEXS	H'ffcc(-H'34)	[p]	未登録(mplid の可変長メモリプールが存在していない)
E_CTX	H'ffbb(-H'45)	[p]	コンテキストエラー(タスク独立部から発行できない)
EV_ILBLK	H'ff1e(-H'e2)	[k]	不正メモリブロックの返却、操作(blk がメモリプールの領域以外、または既に返却した blk を指定)

- **解説**

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

mplid で示された可変長メモリプールに対して空きメモリ獲得待ちタスクが存在し、メモリブロックの返却により待ち行列の先頭タスクが獲得可能となった場合、メモリブロックの先頭アドレスを待ち行列の先頭タスクに渡し、待ちタスクの待ち状態を解除します。

mplid で示された可変長メモリプールに対して空きメモリ獲得待ちタスクが複数存在し、メモリブロックの返却により待ち行列の複数タスクが獲得可能となった場合、待ち行列の先頭タスクから順に、要求サイズのメモリブロック先頭アドレスを待ち行列のタスクに渡し、待ち行列タスクが要求したサイズのメモリが獲得できなくなるまで、複数の待ちタスクの待ち状態を解除します。

blk には、get_blk,pget_blk または tget_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を指定します。異なるアドレスを指定すると、エラーコードとして EV_ILBLK を返します。また、すでに返却したアドレスを指定した場合にも、エラーコードとして EV_ILBLK を返します。

3.11.3 可変長メモリプール状態参照(ref_mpl) [T/D/L/I]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @ref_mpl
```

- **パラメータ**

ID	mplid	R1	可変長メモリプール ID
T_RMPL	*pk_rmpl	ER2/R2	可変長メモリプール状態を返すパケットの先頭アドレス

- **リターンパラメータ**

T_RMPL	*pk_rmpl	ER2/R2	可変長メモリプール状態を格納したパケットの先頭アドレス
ER	ercd	R0	エラーコード

- **パケットの構造**

```
typedef     struct          t_rmpl {
                            VP       exinf;   0/  0  4/2  拡張情報
                            BOOL_ID  wtsk;   +4/ +2  2/2  待ちタスク ID
                            UW       frsz;   +6/ +4  4/4  空き領域の合計サイズ
                            UW       maxsz;  +10/ +8  4/4  最大の空き領域のサイズ
                            UW       mplsz;  +14/+12  4/4  メモリプール全体のサイズ
                            } T_RMPL;
```

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_PAR	H'ffdf(-H'21)	[p]	不正アドレス(pk_rmpl が 0 または奇数)
E_ID	H'ffdd(-H'23)	[p]	不正 ID 番号(mplid≦0, mplid>メモリプール定義数)
E_NOEXS	H'ffcc(-H'34)	[p]	未登録(mplid の可変長メモリプールが存在していない)

- **解説**

mplid で示された可変長メモリプールの状態を参照し、pk_rmpl が指す領域に拡張情報(exinf)、待ちタスク ID(wtsk)、現在の空き領域の合計サイズ(frsz)、最大の空き領域のサイズ(maxsz)、およびメモリプール全体のサイズ(mplsz)を格納し返します。

pk_rmpl の指す領域として 18 バイト(アドバンストモード)または 16 バイト(ノーマルモード)の RAM 領域が必要です。

現在の空き領域の合計サイズとは、空き領域が複数箇所に存在(フラグメント)する場合を含んだ、空き領域の合計サイズです。

最大の空き領域のサイズとは、空き領域が複数箇所に存在(フラグメント)する場合、その中で最大の連続した領域のサイズです。なお、最大の空き領域サイズには 1 回の get_blk, pget_blk または tget_blk システムコールで必要なカーネル用管理領域(16 バイト)が考慮されています。つまり、get_blk, pget_blk または tget_blk システムコール発行で即座に獲得できる最大の blksz となります。

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(0)を返します。
また、対象メモリプールに複数の待ちタスクが存在する場合は、待ちタスク ID として、待ち行列の先頭タスク ID を返します。

対象メモリプールから現在獲得できるメモリブロックが無い場合は、最大の空き領域サイズとして FALSE(0)を返します。

3.12 時間管理機能

(1) 時間管理システムコール

時間管理機能では、システムクロックおよび周期起動ハンドラの機能をサポートしています

表 3.29 に時間管理、表 3.30 に周期起動ハンドラでサポートしているシステムコール一覧を示します。

表 3.29 システムクロック関連システムコール

項番	システムコール	機　能	発行可能なシステム状態			
			T	D	L	I
1	set_tim	**システムクロック設定**	○	○	○	○
2	get_tim	**システムクロック参照**	○	○	○	○

表 3.30 周期起動ハンドラシステムコール

項番	システムコール	機　能	発行可能なシステム状態			
			T	D	L	I
1	act_cyc	**周期起動ハンドラ活性制御**	○	○	○	○
2	ref_cyc	**周期起動ハンドラ状態参照**	○	○	○	○

(2) 時間管理の仕様

表 3.31 に時間管理、表 3.32 に周期起動ハンドラの仕様を示します。

表 3.31 時間管理の仕様

項番	項目	内容
1	**クロック値**	**符号付き 48 ビット**
2	**クロック初期値(初期起動時)**	H'000000000000

表 3.32 周期起動ハンドラの仕様

項番	項目	内容
1	**周期起動ハンドラ定義最大数**	255 **個**
2	**周期起動ハンドラ指定番号**	1～255
3	**周期起動ハンドラ初期活性化状態**	TCY_ON **または** TCY_OFF(**セットアップテーブル設定値**)
4	**周期起動時間間隔**	H'1～H'7fffffff

(3) 時間管理の注意事項

(a) 時間管理の実現

時間管理を使用する場合は、ハードウェアタイマの割込みごとにタイマドライバを作成し、システムに組み込む必要があります。

(b) 多用による弊害

カーネルのタイマ割込み処理では、以下の処理を行います。

- システムクロックの更新(＋1)
- 周期時間に達したすべての周期起動ハンドラの起動と実行
- タイムアウト付きシステムコールによるタスクのタイムアウト処理

これらの処理はすべて、タイマ割込みレベルをマスクした状態で行われます。

周期時間に達した全ての周期起動ハンドラの起動と実行と、タイムアウト付きシステムコールによるタスクのタイムアウト処理は、複数のタスクやハンドラに対する処理が重複する可能性があるため、このような場合の処理時間が極端に長くなります。これは、以下のような弊害をもたらします。

- 割込みに対するレスポンスの悪化
- システムクロックの遅れ

これを避けるために、以下を遵守してください。

- タイマ割込みの周期を極端に短くしないでください。
- タイマハンドラの処理は、可能な限り短くしてください。
- タイマハンドラの周期、タイムアウト付きシステムコールで指定するタイムアウト値は、なるべく大きな値にしてください。極端な例としては、ある周期起動ハンドラの周期時間が 1 で、そのハンドラ処理時間がタイマ周期時間以上かかるような場合、永久にその周期起動ハンドラだけが実行されることになり、事実上ハングアップします。

(c) 時間の管理方法

タイムアウト付きシステムコール、周期起動ハンドラ、システムクロックによる時間の管理は、すべて要求時の時刻からの相対時間に変換して管理しています。このため、set_tim システムコールでシステムクロックを変更しても、以前に行った時間管理要求は影響されません。

3.12.1 システムクロック設定(set_tim) [T/D/L/I]

- **C言語インタフェース**

```
ER ercd = set_tim(SYSTIME *pk_tim);
```

- **アセンブラインタフェース**

```
JSR @set_tim
```

- **パラメータ**

SYSTIME	*pk_tim	ER2/R2	システムの現在時刻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	R0	エラーコード

- **パケットの構造**

```
typedef     struct          systime {
                            H       utime;     0   2    システムの現在時刻(上位)
                            UH      mtime;    +2   2    システムの現在時刻(中位)
                            UH      ltime;    +4   2    システムの現在時刻(下位)
                            }SYSTIME;
```

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_RSFN	H'ffec(-H'14)	[p]	未サポート機能(タイマドライバ未使用)
E_PAR	H'ffdf(-H'21)	[p]	不正アドレス(pk_tim が 0 または奇数) 不正時間指定(pk_tim で示される値が負)

- **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_tim 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+mtime:16bit+ltime:16bit)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、およびタイマハンドラ(周期起動ハンドラ)のタイムアウト時刻は更新しません。

本システムコールを発行してからタイムアウトまでの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。

3.12.2 システムクロック参照(get_tim) [T/D/L/I]

- **C言語インタフェース**

```
ER ercd = get_tim(SYSTIME *pk_tim);
```

- **アセンブラインタフェース**

```
JSR @get_tim
```

- **パラメータ**

SYSTIME	*pk_tim	ER2/R2	システムの現在時刻を返すパケットの先頭アドレス

- **リターンパラメータ**

SYSTIME	*pk_tim	ER2/R2	システムの現在時刻を格納したパケットの先頭アドレス
ER	ercd	R0	エラーコード

- **パケットの構造**

```
typedef     struct          systime {
                            H       utime;      0   2   システムの現在時刻(上位)
                            UH      mtime;     +2   2   システムの現在時刻(中位)
                            UH      ltime;     +4   2   システムの現在時刻(下位)
                            }SYSTIME;
```

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_RSFN	H'ffec(-H'14)	[p]	未サポート機能(タイマドライバ未使用)
E_PAR	H'ffdf(-H'21)	[p]	不正アドレス(pk_tim が 0 または奇数)

- **解説**

システムクロックの現在値を読み出し、その結果を pk_tim の指す領域に返します。pk_tim の指す領域として 6 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+mtime:16bit+ltime:16bit)で表現されます。

3.12.3 周期起動ハンドラ活性制御(act_cyc) [T/D/L/I]

- **C言語インタフェース**

```
ER ercd = act_cyc(HNO cycno, UINT cycact);
```

- **アセンブラインタフェース**

```
JSR @act_cyc
```

- **パラメータ**

HNO	cycno	R1	周期起動ハンドラ指定番号
UINT	cycact	R2	周期起動ハンドラ活性状態

- **リターンパラメータ**

ER	ercd	R0	エラーコード

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_RSFN	H'ffec(-H'14)	[p]	未サポート機能(タイマドライバ未使用)
E_PAR	H'ffdf(-H'21)	[p]	パラメータエラー(cycact が不正) (cycno 範囲外 : cycno ≦ 0,cycno > 周期起動ハンドラ定義数)
E_NOEXS	H'ffcc(-H'34)	[p]	未登録(cycno の周期起動ハンドラが定義されていない)

- **解説**

cycno で示された周期起動ハンドラの活性状態を、cycact で示された状態に変更します。
cycact には、ハンドラ活性状態の指定を行います。

cycact:=(TCY_OFF ‖ TCY_ON) [| TCY_INI]
TCY_OFF(H'0000)周期起動ハンドラは起動されない
TCY_ON(H'0001)周期起動ハンドラは起動される
TCY_INI(H'0002)周期起動ハンドラのカウントが初期化される

cycact＝TCY_OFF では、周期起動ハンドラの活性状態を OFF にします。したがって周期時間が経過しても周期起動ハンドラは起動されません。ただし、活性状態が OFF の場合でも周期時間のカウントは行います。

cycact＝TCY_ON では、周期起動ハンドラの活性状態を ON にします。活性状態が OFF のときも周期時間のカウントは行うため、活性状態を ON にした後の周期起動ハンドラ起動までの時間は不定ですので注意してください。ただし、cycact=(TCY_ON | TCY_INI)の指定を行った場合には本システムコール発行からちょうど周期時間経過後に周期起動ハンドラが起動されます。

3.12.4 周期起動ハンドラ状態参照(ref_cyc) [T/D/L/I]

- **C言語インタフェース**

```
ER  ercd = ref_cyc(T_RCYC *pk_rcyc, HNO cycno);
```

- **アセンブラインタフェース**

```
JSR @ref_cyc
```

- **パラメータ**

HNO	cycno	R1	周期起動ハンドラ指定番号
T_RCYC	*pk_rcyc	ER2/R2	周期起動ハンドラの状態を返すパケットの先頭アドレス

- **リターンパラメータ**

T_RCYC	*pk_rcyc	ER2/R2	周期起動ハンドラの状態を格納したパケットの 先頭アドレス
ER	ercd	R0	エラーコード

- **パケットの構造**

```
typedef     struct          t_rcyc {
                            VP      exinf;    0/  0   4/2    拡張情報
                            CYCTIME lfttim;  +4/ +2   4/4    周期起動ハンドラ起動
                                                             までの残り時間
                            UINT    cycact;  +8/ +6   2/2    周期起動ハンドラ活性状態
                            FP      cychdr;  +10/ +8  4/2    周期起動ハンドラアドレス
                            CYCTIME cyctim;  +14/+10  4/4    周期起動時間間隔
                            } T_RCYC;
```

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_RSFN	H'ffec(-H'14)	[p]	未サポート機能(タイマドライバ未使用)
E_PAR	H'ffdf(-H'21)	[p]	不正アドレス(pk_rcyc が 0 または奇数) パラメータエラー (cycno≦0,cycno>周期起動ハンドラ定義数)
E_NOEXS	H'ffcc(-H'34)	[p]	未登録(cycno の周期起動ハンドラが定義されていない)

- **解説**

cycno で示された周期起動ハンドラの状態を参照し、pk_rcyc が指す領域に拡張情報(exinf)、周期起動ハンドラ起動までの残り時間(lfttim)、周期起動ハンドラ活性状態(cycact)、周期起動ハンドラアドレス(cychdr)、および周期起動時間間隔(cyctim)を格納し返します。

pk_rcyc の指す領域として 18 バイト(アドバンストモード)または 14 バイト(ノーマルモード)の RAM 領域が必要です。

周期起動ハンドラ活性状態(cycact)では、TCY_ON(H'0001)と TCY_OFF(H'0000)の情報のみが返され、TCY_INI(H'0002)の情報は返されません。

3.13 システム管理機能

3.13.1 バージョン参照(get_ver) [T/D/L/I]

- **C言語インタフェース**

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

- **アセンブラインタフェース**

```
JSR @get_ver
```

- **パラメータ**

T_VER	*pk_ver	ER2/R2	バージョン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**

T_VER	*pk_ver	ER2/R2	バージョン情報を格納したパケットの先頭アドレス
ER	ercd	R0	エラーコード

- **パケットの構造**

```
typedef     struct          t_ver {
                            UH    maker;      0   2   メーカー
                            UH    id;         +2  2   形式番号
                            UH    spver;      +4  2   仕様書バージョン
                            UH    prver;      +6  2   製品バージョン
                            UH    prno[4];    +8  8   製品管理情報
                            UH    cpu;        +16 2   CPU 情報
                            UH    var;        +18 2   バリエーション記述子
                            } T_VER;
```

- **エラーコード**

E_OK	H'0000	[k]	正常終了
E_PAR	H'ffdf(-H'21)	[p]	不正アドレス(pk_ver が 0 または奇数)

- **解説**

現在実行中のカーネルのバージョンに関する情報を読み出し、その結果を pk_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。
maker は、この製品を作ったメーカを表します。(H’000a:(株)日立製作所)

(id)：カーネルの id の値は H’000a です。
id は、OS や VLSI の種類を区別する番号を表します。(H’000a:HI2000/3)

(spver)：カーネルの spver の値は H’5302 です。
spver は、TRON 仕様のシリーズを区別する番号(H’5: μITRON 仕様)と、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(H’302:Ver3.02)表します。

(prver)：カーネルの prver の値は H’0100 です。
prver は、バージョン番号を表します。(H’0100:Ver1.0)

(prno)：カーネルの prno[0]から prno[3]の値は H’0000 です。
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’0a26、または H’0a20 です。
cpu は、(maker)で示した値(H’0a:(株)日立製作所)と、この μITRON を実行するプロセッサ(H’26:H8S/2600、H’20:H8S/2000)を表します。

(var)：カーネルの var の値は H’4000 です。
var は、以下の内容を表します。

- カーネル仕様のレベル分け(B'0100: μITRON レベル R)
- 未使用(B'0)
- シングルプロセッサ用(B'0)
- 仮想記憶未サポート(B'0)
- MMU 未対応版(B'0)
- 未使用(B'0)
- ファイル仕様のレベル分け(B'000:サポートなし)
- 未使用(B'0000)

4. デバッギングエクステンション

4.1 概要

デバッギングエクステンション(DX)は、日立デバッギングインタフェース(以下 HDI と略す)および HI2000/3 システムに組み込んで使用します。

4.1.1 オブジェクトの参照と操作

デバッギングエクステンション(DX)のオブジェクト参照と操作は、HDI の[View]メニューで開いたウィンドウから選択して行います。表 4.1 にデバッギングエクステンション(DX)で HDI の[View]メニューに追加される内容を示します。

表 4.1 HDI の [View]メニューに追加される内容

項番	View Menu	Status Bar
1	Task List	Open Task List
2	Trace System	Open System Trace
3	Event Flags	Open Event Flag
4	Variable Memory Pool	Open Variable Memory Pool
5	Fixed Memory Pool	Open Fixed Memory Pool
6	Semaphores	Open Semaphore
7	Mailboxes	Open Mailbox
8	Cyclic Handler	Open Cyclic Handler

HDI の[View]メニューで開いたウィンドウを選択すると、オブジェクトの状態を表示します。ウィンドウタイプは最も一般的なリストタイプウィンドウと階層構造タイプウィンドウの２種類があります。図 4.1 と図 4.2 に表示例を示します。

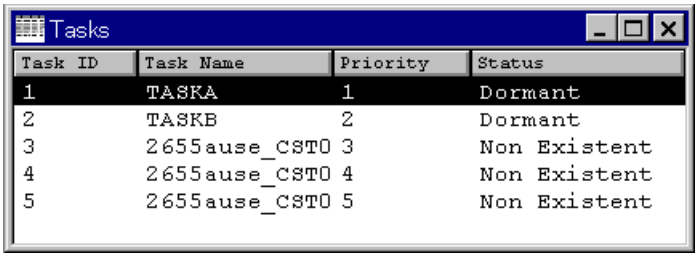


図 4.1 オブジェクト状態の表示例(リストタイプウィンドウ)

図 4.1 は、Tasks ウィンドウで TaskID(タスク ID),TaskName(タスク名称),Priority(現在のタスク優先度),Status(現在のタスク状態)を表示します。

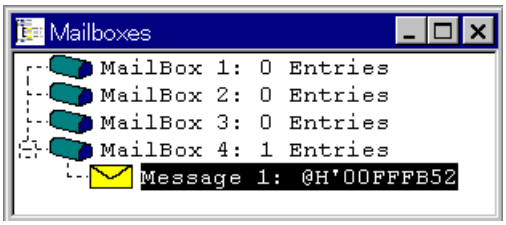


図 4.2　オブジェクト状態の表示例(階層構造タイプウィンドウ)

図 4.2 は、Mailboxes ウィンドウで全てのメイルボックスの状態(先頭待ちタスクの ID,またはメッセージ数)を表示します。また、階層構造でメッセージのキュー状態、メッセージアドレスを表示できます。

各オブジェクトのウィンドウからオブジェクト操作を要求します。オブジェクト操作の要求は、各オブジェクトウィンドウのポップアップメニューから開いたダイアログボックスで行います。デバッギングエクステンション(DX)のオブジェクト操作は、デバッグデーモンを通じてカーネルに要求するだけで、実際のオブジェクト操作はカーネルが行います。
図 4.3 に表示例を示します。

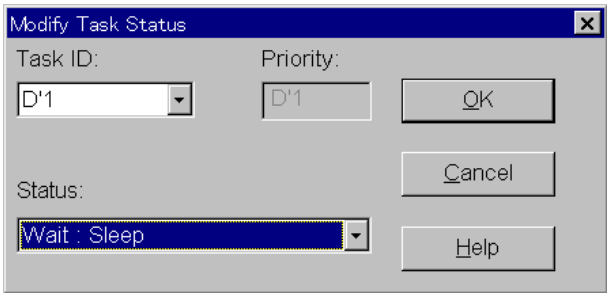


図 4.3　オブジェクト操作要求の表示例

図 4.3 は、Modify Task Status ダイアログでタスク状態の変更を行います。Task ID,Status コンボボックスのドロップダウンリストで変更します。

4.1.2　オブジェクト操作に対する結果表示

オブジェクト操作に対する結果は、各ウィンドウにそれぞれの状態として表示されます。以下のタイミングで表示を更新します。

- 各ウィンドウの右ボタンクリックによるポップアップメニューから[Update]を選択したとき
- ターゲットシステムが停止したとき(ブレークポイントやその他の要因)

4.1.3 レジスタ値の表示

タスクリストポップアップメニューの[View Context]オプションを選択すると、選択タスクのレジスタ値を表示します。
図 4.4 に表示例を示します。

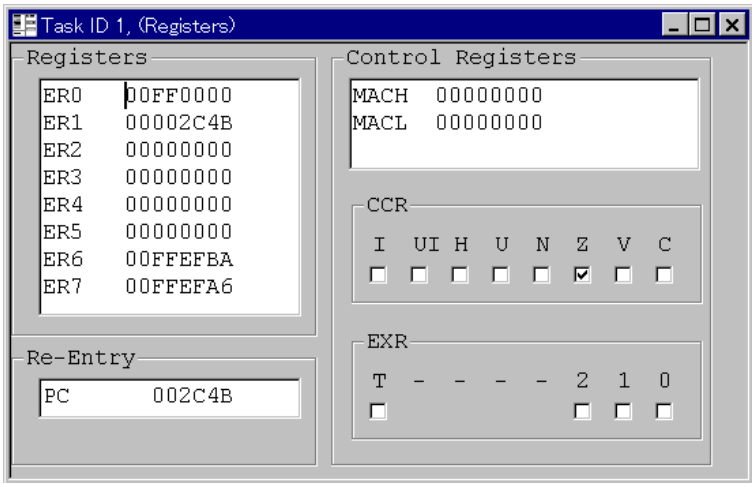


図 4.4 [Task Context Registers]**ウィンドウ**

図 4.4 は、Task Context Registers ウィンドウで、プログラム停止中にタスクのレジスタ値を編集できます。本例は、タスク ID１のレジスタ表示です。

4.1.4 システムコール履歴の表示

カーネルのトレースバッファから情報を取得して、システムコール全トレース情報を一覧表示できます。最新情報は、cycle：-0 の行に表示されます。
図 4.5 に表示例を示します。

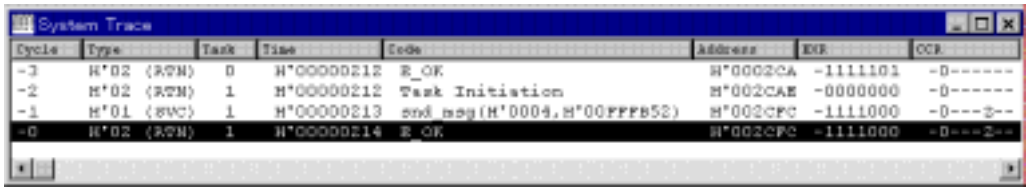


図 4.5 [System Trace]**ウィンドウ**

図 4.5 は、System Trace ウィンドウで、タスクからのシステムコール発行、リターンをトレース情報として、表示します。

4.1.5 オンラインヘルプ

Windows®標準のコンテキスト依存型ヘルプシステムを備えています。

デバッギングエクステンション(DX)の操作方法およびウィンドウ、ダイアログボックスの詳細についてはオンラインヘルプを参照してください。

オンラインヘルプは、デバッギングエクステンション(DX)のウィンドウがアクティブのときに[F1]キーを押すか、またはダイアログボックス内の[Help]ボタンをクリックすることにより開くことができます。

4.2 機能一覧

4.2.1 メニュー

デバッギングエクステンション(DX)のメニューの一覧を表 4.2 に示します。

表 4.2 **メニュー一覧**

項番	メニューバー	プルダウンメニュー	機能
1	View	Task List	[Tasks]**ウィンドウのオープン**
2		Trace System	[System Trace]**ウィンドウのオープン**
3		Event Flags	[Event Flags]**ウィンドウのオープン**
4		Variable Memory Pool	[Variable Length Memory Pool]**ウィンドウのオープン**
5		Fixed Memory Pool	[Fixed Length Memory Pool]**ウィンドウのオープン**
6		Semaphores	[Semaphore]**ウィンドウのオープン**
7		Mailboxes	[Mailboxes]**ウィンドウのオープン**
8		Cyclic Handler	[Cyclic Handler]**ウィンドウのオープン**

4.2.2 ウィンドウとダイアログボックス

ウィンドウとダイアログボックスの一覧を表 4.3 に示します。ウィンドウとダイアログボックスの詳細についてはオンラインヘルプを参照してください。オンラインヘルプは、デバッギングエクステンション(DX)のウィンドウがアクティブのときに[F1]キーを押すか、またはダイアログボックス内の[Help]ボタンをクリックすることにより開くことができます。

表 4.3 ウィンドウ一覧

項番	分類	ウィンドウ名称	機　能
1	タスク	[Tasks]ウィンドウ	全タスクの状態表示
		[Task Modification]ダイアログボックス	タスク状態の変更
2	イベントフラグ	[Event Flag]ウィンドウ	全イベントフラグの状態表示
		[Event Flag Modification]ダイアログボックス	イベントフラグ状態の変更
3	セマフォ	[Semaphore]ウィンドウ	全セマフォの状態表示と変更
4	メイルボックス	[Mailboxes]ウィンドウ	全メイルボックスの状態表示
		[Mailbox Post message]ダイアログボックス	メイルボックスへのメッセージ送信
5	固定長メモリプール	[Fixed Length Memory Pool]ウィンドウ	全固定長メモリプールの状態表示
6	可変長メモリプール	[Variable Length Memory Pool]ウィンドウ	全可変長メモリプールの状態表示
7	タイマ	[Timer]ウィンドウ	システムクロック値の表示
		[Timer Modification]ダイアログボックス	システムクロック値の変更
8	トレース	[System Trace]ウィンドウ	全トレース情報の表示
9	タスクコンテキスト	[Task Context Local Variables]ウィンドウ	タスクコンテキストローカル変数の表示
		[Task Context Registers]ウィンドウ	タスクコンテキストレジスタ値の表示
		[Edit Value]ダイアログボックス	タスクコンテキストローカル変数の変更
		[Registers]ダイアログボックス	タスクコンテキストレジスタ値の変更
10	周期起動ハンドラ	[Cyclic Handler]ウィンドウ	周期起動ハンドラの状態表示
		[Activate Cyclic]ダイアログボックス	周期起動ハンドラの活性状態の変更

4.3 留意事項

デバッギングエクステンション(DX)を使用する上での留意事項を表 4.4 に示します。

表 4.4 留意事項

項目	内容
E6000 エミュレータのセットアップ	プログラム実行中にウィンドウの更新を行うには、E6000 エミュレータを以下のようにセットアップしておく必要があります。 [Setup]メニューから[Configure Platform]を選択し、コンフィグレーションダイアログボックスを表示します。 コンフィグレーションダイアログボックスの[Enable read and write on the fly]チェックボックスをチェックし、プログラム実行中のメモリアクセスを可能にしてください。
ウィンドウ表示	HDI 起動後、最初にデバッギングエクステンション(DX)のウィンドウを表示するときは、必要な情報をターゲットから取得するため、表示まで暫く時間がかかります。
ターゲットシステムのリアルタイム性	デバッギングエクステンション(DX)は、ターゲットシステムのメモリ内容を参照あるいは更新することで、その機能を実現しています。ターゲットシステム実行中に、以下の操作をするとメモリアクセスが発生します。 ・デバッギングエクステンション(DX)のウィンドウを開いたとき、更新したとき ・ダイアログボックスで[OK]ボタンをクリックしたとき また、ターゲットシステムでは、デバッグデーモンが周期的に動作するため、デバッギングエクステンション(DX)の機能を使用していないときも、上記に比べるとわずかですが、ターゲットシステムのスループットが低下します。
表示内容の整合性	デバッギングエクステンション(DX)は、ターゲットシステムのメモリ内容を直接読み出して参照機能を実現しています。このため、以下に示すシステム状態では、正しい情報を表示しない場合があります。 ・プログラム実行中の表示内容 カーネル実行中にメモリ内容の読み出しが行われた場合、正しい情報を表示しない場合があります。 ・カーネル起動完了前の表示内容 カーネルの初期化が完了するまで(システム初期化ハンドラの起動前)は、正しい情報を表示しない場合があります。
トレース機能	デバッギングエクステンション(DX)の[System Trace]ウィンドウを表示するには、カーネルのシステムコールトレース機能の登録が必要です。トレース機能を登録については、「6.2.6 トレース機能の登録」を参照してください。
ターゲットシステムのメモリ	デバッギングエクステンション(DX)出荷時の設定でのメモリ使用量を以下に示します。 ターゲットシステムのメモリ使用量 メモリ種別 — メモリ使用量 ROM 領域(デバッグデーモンのみ) — 最大 500 バイト RAM 領域 — 最大 200 バイト デバッグデーモンが結合するシステムコールによって、カーネルのメモリ使用量(ROM 領域)が最大 2.9k バイト増加する場合があります。
セッション	デバッギングエクステンション(DX)は、HDI のセッションに対応していません。セッションを保存してもデバッギングエクステンション(DX)の設定は保存されません。
ロードモジュールのロード	デバッギングエクステンション(DX)動作後、ロードモジュールを再度ロードする場合は、デバッギングエクステンション(DX)のウィンドウをオープンした状態で行ってください。

4.4 デバッグデーモン

4.4.1 デバッグデーモンの組込み

デバッギングエクステンション(DX)を使用する場合、デバッグデーモンをシステムに組み込む必要があります。
デバッグデーモンは周期起動ハンドラとして動作します。

デバッグデーモンを組み込む場合は、CPU 初期化ルーチン(***nnnnz***cpu.src)、およびセットアップテーブル(***nnnnz***sup.src)のアセンブル時、アセンブルオプションとして-define=DX=“ Action ”を指定してください。

(1) セットアップテーブルの変更

デバッグデーモンの周期起動時間間隔は次式で定まります。周期起動時間間隔はユーザのシステムに応じて変更してください。

デバッグデーモンの周期=周期起動時間間隔×ハードウェアタイマの周期

デバッグデーモンの周期起動時間間隔は、50msec 程度となるように設定してください。
提供しているタイマドライバでは、ハードウェアタイマの周期を 10msec としており、セットアップテーブルのデバッグデーモンの周期起動時間間隔は 5 に設定しています。

セットアップテーブルの変更に関する詳細は、「6.2.5 周期起動ハンドラの登録」を参照してください。

(2) CPU 初期化ルーチンの変更

提供している CPU 初期化ルーチンを使用せず、ユーザが新たに CPU 初期化ルーチンを作成した場合、デバッグデーモン初期化処理を追加してください。

- _HI_DEAMON_INI を import 宣言します
- カーネルの初期化処理(jmp @_H_2S_INIT)にジャンプする前に_HI_DEAMON_INI のサブルーチンコール命令(jsr @_HI_DEAMON_INI)を追加します

標準提供している CPU 初期化ルーチン(***nnnnz***cpu.src)、およびセットアップテーブル(***nnnnz***sup.src)は、既に上記変更内容が組み込まれています。

4.5 チュートリアル

デバッギングエクステンション(DX)を使用した具体的な操作例を説明します。

以下の操作例を行う前に、ロードモジュールを生成する必要があります。標準提供ファイルは、前項「4.4.1 デバッグデーモンの組込み」に示す変更が、既に組み込まれていますので、HEW ワークスペースプロジェクトの define オプションに“ DX=Action ”を追加し、システムを構築することで例題に示すロードモジュールが生成できます。「8.ロードモジュールの生成」を参照し、ロードモジュールを生成してください。

例題では、H8S/2655 マイコンのアドバンストモードを使用した例で操作説明をしています。ユーザの使用環境に置き換えて操作してください。

例題プログラムには、TASKA と TASKB の 2 つのタスクがあります。TASKA は、メイルボックス 4 にメッセージを送信(snd_msg システムコールを発行)し、メイルボックス 3 からメッセージを受信(rcv_msg システムコールを発行)し、起床待ち状態(slp_tsk システムコールを発行)に移行する処理を行うタスクです。TASKB は、メイルボックス 4 からメッセージを受信(rcv_msg システムコールを発行)し、メイルボックス 3 にメッセージを送信(snd_msg システムコールを発行)し、起床待ち状態(slp_tsk システムコールを発行)に移行する処理を行うタスクです。
図 4.6 に例題プログラムの処理概要を示します。

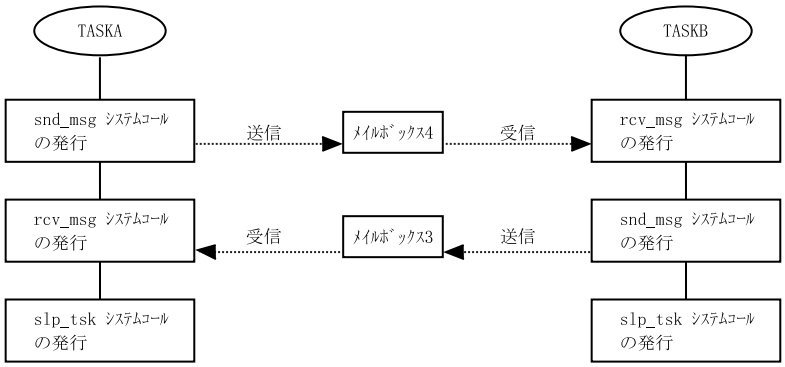


図 4.6 例題プログラムの処理

本例では HDI を使用しています。HDI の操作方法の詳細については、日立デバッギングインタフェースユーザーズマニュアル、および使用する H8S シリーズマイコン用の E6000 エミュレータユーザーズマニュアルを参照してください。

4.5.1 ＨＤＩスタートアップとプログラムのロード

(1) HDI スタートアップ

[HDI for E6000 H8S]アイコンを選択して HDI を起動します。
図 4.7 のような HDI 起動画面が表示されます。

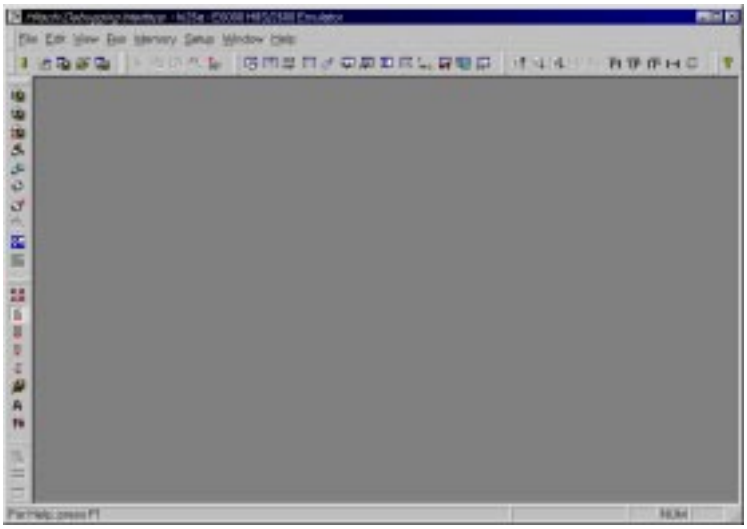


図 4.7 HDI **起動画面**

(2) プログラムのロード

[File]メニューから[Load Program...]を選択して[Open]ダイアログボックスを開き、hi26a ディレクトリ下のアプリケーションロードモジュール「hi26a.abs」をロードします(エミュレータの Read/Write 領域にロードしてください)。この段階では、カーネルが起動されていないため、プログラムはまだ動作していません。

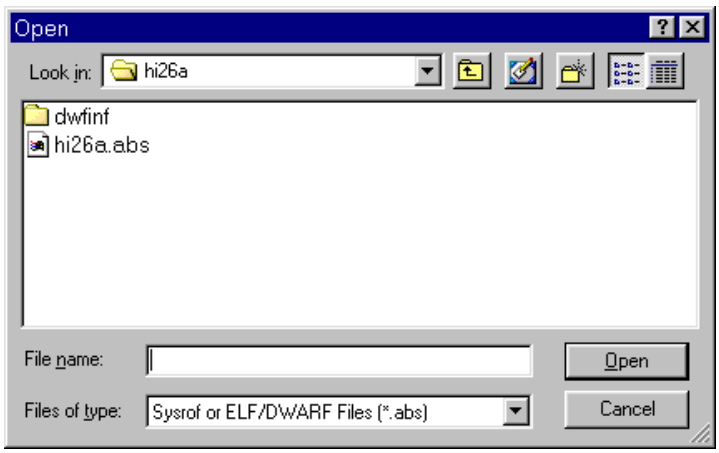


図 4.8 [Open]**ダイアログボックス**

(3) システムの初期化

[Run]メニューから[Go Reset]を選択してプログラムを実行します。数秒後 STOP ボタンを押してください。システムが初期化された後、HDI ソースウィンドウが開きます。

(4) タスク状態の確認

[View]メニューから[Task List]を選択して、[Tasks]ウィンドウを開き、タスクの状態を確認します。タスクは「2655asup.src」のタスク定義テーブルにしたがって実行されます。最初の 2 タスクは休止状態、その他のタスクは未登録状態です。

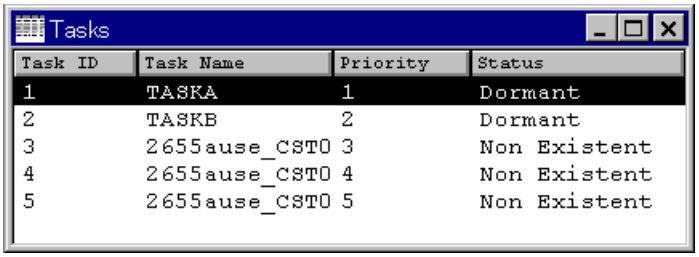


図 4.9 [Tasks]**ウインドウ**

4.5.2 タスクの起動

(1) ブレークポイントの設定

プログラムを実行する前にブレークポイントを設定します。[View]メニューから[Source...]を選択して[Open]ダイアログボックスを開き、「task.c」ファイルを開きます。2 つの休止状態のタスク TASKA と TASKB のソースコードウィンドウが表示されます。TASKA の snd_msg ラインにブレークポイントを設定します。

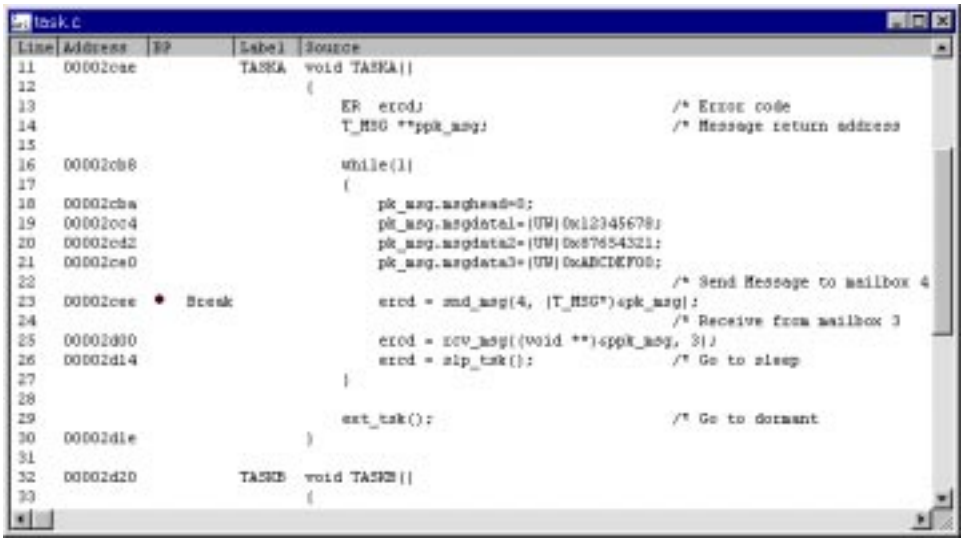


図 4.10 **ソースコード表示**

(2) タスクの起動

[Modify Task Status]編集ダイアログボックスを開き、TASKA(タスク ID1)を起動します(図 4.11)。プログラムが動作するまでは何も変化しません。[Run]メニューから[Go]を選択してプログラムを動作させます。プログラムがブレークポイントに到達し、停止します。

図 4.11 **タスクの起動**

(3) システムトレースの確認

(2)項の動作内容を確認します。[View]メニューから[Trace System]を選択して[System Trace]ウィンドウを開きます(図 4.12)。停止直前 4 エントリ分のシステムトレース情報が表示されます(cycle:-0 がトレース最新情報です)。表 4.5 でその内容を説明します。

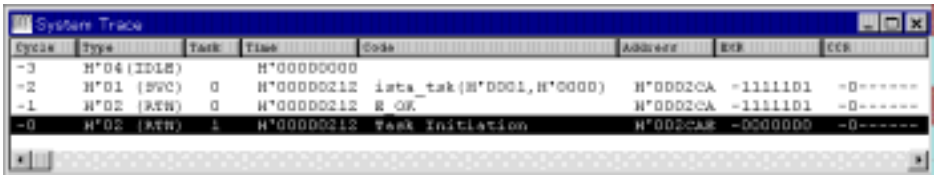


図 4.12 [System Trace]**ウィンドウ**

表 4.5 Trace **内容の説明**

項番	Cycle	説　明
1	-3	**システムはアイドル状態です。これは** TASKA(**タスク** ID1)**がスタートする前のシステムの状態を表します。**
2	-2	**タスク独立部から** ista_tsk **システムコールコールが発行されます。**
3	-1	ista_tsk **システムコールのエラーコードです。**
4	-0	TASKA(**タスク** ID1)**用の** RTN **属性を表します。これはタスクが起動したことを示します。**

4.5.3 メイルボックスとメッセージ

(1) メイルボックス状態の確認

[View]メニューから[Mailboxes]を選択して、[Mailboxes]ウィンドウを開き、メイルボックスの状態を確認します(図 4.13)。

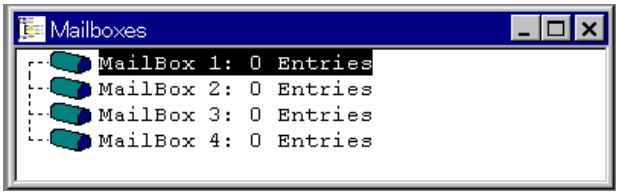


図 4.13 [Mailboxes]**ウィンドウ**

(2) メッセージの送信

メイルボックス 4 にメッセージを送信してみます。[View]メニューから[Source...]を選択して[Open]ダイアログボックスを開き、「task.c」ファイルを開きます。2 つの休止タスク TASKA と TASKB のソースコードウィンドウが表示されます。[Run]メニューから[Step Over]を選択して TASKA の、ercd=snd_msg(4,(T_MSG*)&pk_msg)ラインをステップオーバー実行します(図 4.14)。

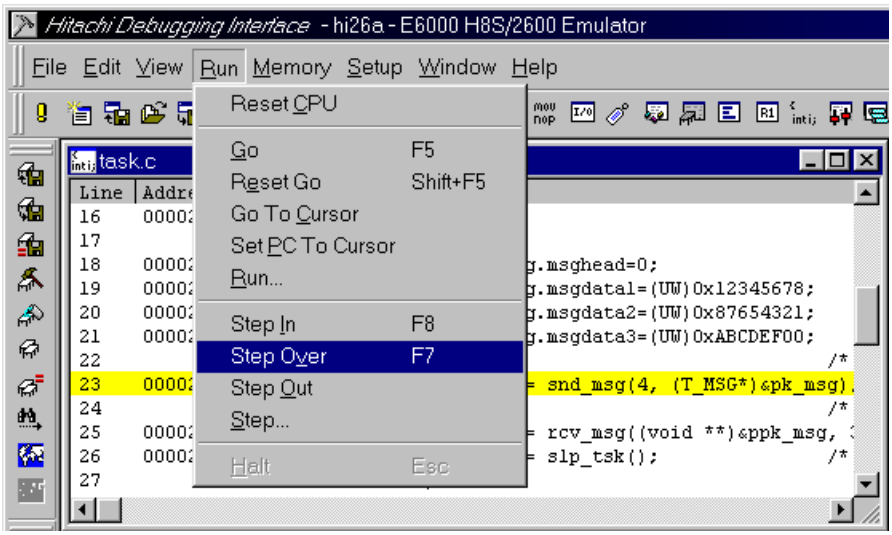


図 4.14 **プログラムのステップオーバー実行**

(3) 結果の確認

メイルボックス 4 に 1 つのエントリ項目が表示されます。

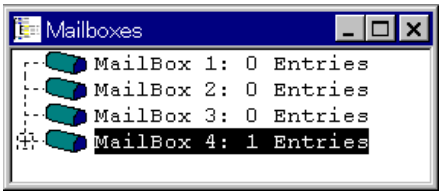


図 4.15 [Mailboxes]**ウィンドウ(結果の確認)**

このメイルボックスを拡張表示して下さい([+]ボックスをマウスの右ボタンでクリック)。メッセージの先頭アドレス H'00FFFB52 が表示されます。

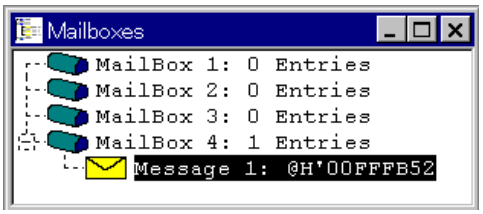


図 4.16 [Mailboxes]**ウィンドウ(拡張表示)**

(4) システムトレースの確認

プログラムの動作内容を確認します。[View]メニューから[Trace System]を選択して[System Trace]ウィンドウを開きます(図 4.17)。プログラムの動作内容がトレース情報として表示されます。cycle-1 でメッセージを送信(snd_msg システムコールの発行)し、cycle-0 でシステムコールの応答結果(エラーコード)を受け取っていることが示されています。

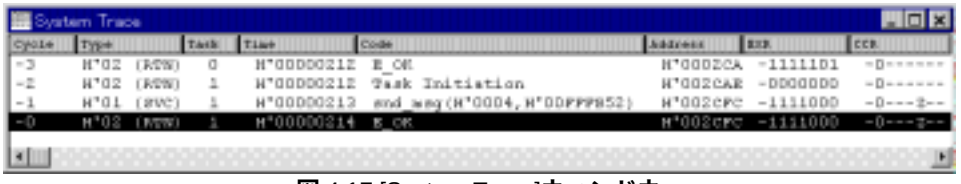


図 4.17 [System Trace]**ウィンドウ**

4.5.4 システム動作中の操作例

4.5.3 項までは、システム停止中の使用方法について説明しました。ここでは、システム動作中の使用方法について説明します。

(1) [Tasks]ウィンドウ、[Mailboxes]ウィンドウの表示

[View]メニューから[Task List]を選択して、[Tasks]ウィンドウを開きます。同様に、[View]メニューから[Mailboxes]を選択して、[Mailboxes]ウィンドウを開きます。

(2) システムのスタート

[Run]メニューから[Go]を選択してプログラムを実行します。

(3) タスク状態の確認

[Task List]ウィンドウのポップアップメニューから[Update]オプションを選択してください。TASKA(タスク ID1)は、メッセージボックス 3 のメッセージ待ち状態であることが表示されます(図 4.18)。

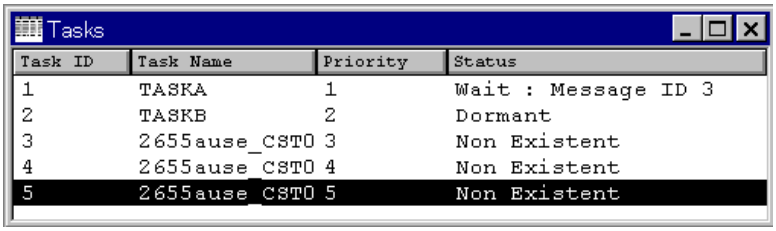


図 4.18 [Tasks]**ウインドウ**([Update]**オプション選択後**)

(4) メイルボックス状態の確認

[Mailboxes]ウィンドウのポップアップメニューから[Update]オプションを選択してください。メイルボックス 3 に待ちタスクとして、TASKA(タスク ID1)が存在することが表示されます(図 4.19)。

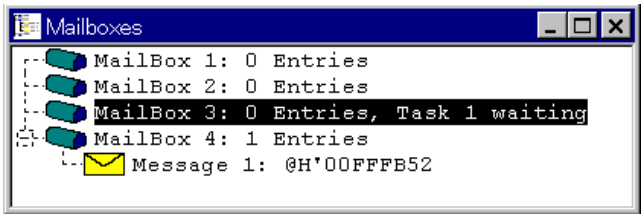


図 4.19 [Mailboxes]**ウインドウ**([Update]**オプション選択後**)

(5) システムトレースの確認

[View]メニューから[Trace System]を選択して[System Trace]ウィンドウを開きます。トレース内容は、TASKA(タスク ID1)が rcv_msg をコールし、システムがアイドル状態に変化することが表示されます。

図 4.20 [System Trace]**ウインドウ**

(6) タスクの起動

TASKB(タスク ID2)を起動して、メイルボックス 3 にメイルを送信してみます。

[Task List]ウィンドウで TASKB(タスク ID2)を選択して、ポップアップメニューから[Edit Properties]を選択します。[Modify Task Status]ダイアログボックスが開きます(図 4.21)。[Status]ボックスから Start を選択し、タスクを起動します。TASKB(タスク ID2)は、メイルボックス 3 にメッセージを送信します。

例題プログラムでは、TASKB(タスク ID2)からメイルボックス 3 にメイルを送信すると、TASKA(タスク ID1)がそれを受信し、それぞれ待ち状態に移行します。

図 4.21 [Modify Task Status]**ダイアログ**

(7) 結果の確認

[Task List]ウィンドウのポップアップメニューから[Update]オプションを選択して、各ウィンドウ内容を更新してください。メッセージの送受信が完了し、TASKA(タスク ID1)、TASKB(タスク ID2)ともに待ち状態に移行しています(図 4.22)。

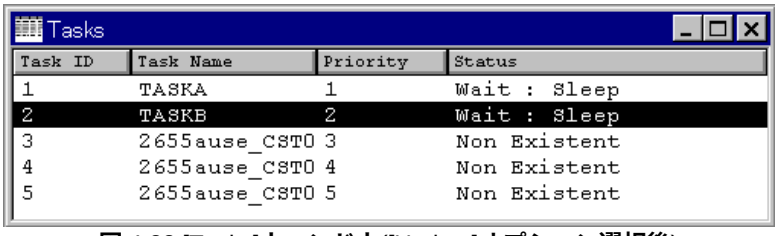


図 4.22 [Tasks]**ウィンドウ**([Update]**オプション選択後**)

また、[System Trace]ウィンドウには、これらのタスクへの slp_tsk システムコールから、アイドリング状態に変化するまでが表示されます(図 4.23)。

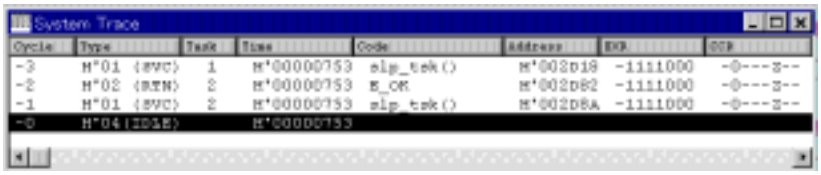


図 4.23 [System Trace]**ウィンドウ**

5. アプリケーションプログラムの作成

5.1 ユーザプログラムの作成

ユーザのシステムに必要となるプログラムを、C 言語、アセンブリ言語により作成します。
作成するプログラムを以下に示します。

- 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ハンドラおよび各種ルーチン
  (CPU初期化ルーチン、システム異常終了ルーチン、タイマ初期化ルーチン、システム初期化ハンドラ、システムアイドリングルーチン)

作成するプログラムは、ユーザのシステムに合わせてプログラムを作成してください。
図 5.1 にカーネルの起動処理概要を示します。

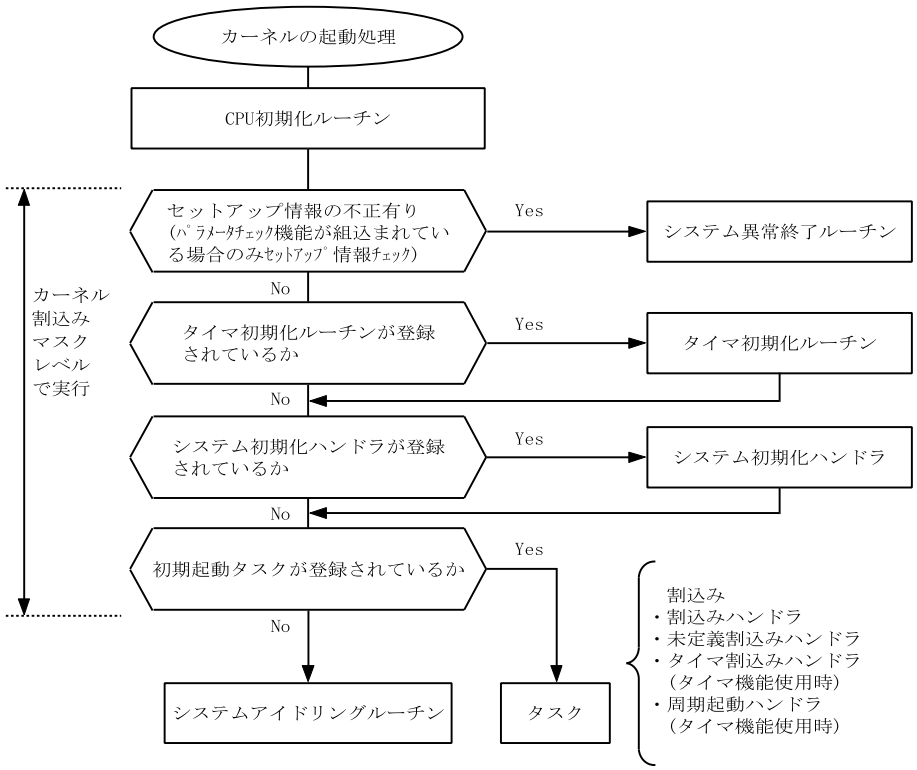


図 5.1 カーネルの起動処理概要

5.2 タスク

5.2.1 タスクの作成

タスクは、図 5.2 に示すように通常の関数として記述します。
タスクを終了する場合は、必ず ext_tsk システムコールを用いて終了してください。タスクの開始関数から呼び出し元にリターンした場合、システムの正常な動作は保証されません。
C 言語で記述したプログラムはコンパイル時の CPU オプション／環境変数指定により、ノーマルモード、アドバンストモードのどちらにも使用できます。

```
#include “ hi2000.h ”

void        task(INT stacd)
INT         stacd;
{
ID          tskid;                          /* ﾀｽｸ id                        */
ER          ercd;                           /* ｴﾗｰｺｰﾄﾞ                      */
            .
            .
            ercd = wup_tsk(tskid);          /* ﾀｽｸ部から発行できるｼｽﾃﾑｺｰﾙ    */
            .
            .
            ext_tsk();                      /* ﾀｽｸ部から発行できるｼｽﾃﾑｺｰﾙ    */
}
```

図 5.2　C 言語によるタスクの記述例

タスクは、発生した事象(タスクの実行要求)にしたがって、タスクの状態とタスクに付けられた優先度に基づいて実行されます。
タスクの優先度は値の小さい方が高い優先度になります。

(1) タスク起動時の初期化処理内容

タスク起動時のタスクに関する初期化内容を表 5.1 に示します。
対象となるシステムコールは sta_tsk、ista_tsk です。

表 5.1 タスクの初期化内容

項番	項目	初期化の内容
1	プログラムカウンタ(PC)	タスク登録時に指定したタスク先頭アドレス
2	コンディションコードレジスタ(CCR)	割込みマスク解放(0)
3	エクステンドレジスタ(EXR)	割込みマスク解放(0)
4	スタックポインタ(ER7)	タスク登録時に指定したタスクスタックポインタ
5	R0(ｱｾﾝﾌﾞﾗ記述)／第 1 引数(C 記述)	sta_tsk システムコールで指定された任意の起動コード値(stacd)
6	汎用レジスタ(E0,ER1～ER6) 積和演算レジスタ(MACH,MACL)	不定(積和演算レジスタ(MACH,MACL)は 2600CPU のみ)
7	タスク優先度	タスク登録時に指定したタスク初期優先度
8	タスク起床要求(wupcnt)	0

(2) タスク終了前の処理内容

タスクが休止状態となる場合に解放しなければならない資源(セマフォとメモリブロック)を、表 5.2 に示します。
対象となるシステムコールは ext_tsk、ter_tsk です。

表 5.2 メモリブロックと資源の一覧

項番	資源	内容	獲得のｼｽﾃﾑｺｰﾙ	開放のｼｽﾃﾑｺｰﾙ
1	セマフォカウント	P 命令で獲得した資源数	wai_sem,preq_sem, twai_sem	sig_sem
2	メモリブロック	固定長メモリプールから獲得したメモリブロック	get_blf,pget_blf, tget_blf	rel_blf
3		可変長メモリプールから獲得したメモリブロック	get_blk,pget_blk, tget_blk	rel_blk

(3) タスクの CPU 占有と割込みマスク

(a) loc_cpu システムコールによる CPU 占有

タスクが CPU を占有して実行するためには、loc_cpu システムコールを用いてシステム状態を CPU ロック状態とします。CPU ロック状態からは、unl_cpu システムコールを発行することにより元のタスク実行状態に戻ります。CPU ロック状態は、タスク実行状態と比べ以下の点が異なります。

- タスク部用のシステムコール
  待ち状態となるシステムコールは発行できません。
- タスク独立部用のシステムコール
  タスク独立部用のシステムコールは発行できません。
- タスク切り替えの遅延
  CPUロック状態の間にタスク切り替えが必要となっても、タスク実行状態に戻る(unl_cpuシステムコール発行)まで遅延されます。
- 割込みのマスク
  セットアップファイルに定義したカーネル割込みマスクレベル以下のレベルの割込みは受け付けられません。

(b) chg_ims システムコールによる CPU 占有

タスク実行中に割込みをマスクすることによって、システム状態をタスク独立部にすることができます。割込みマスクは、chg_ims システムコールを使用してください。タスク実行中に割込みをマスクすると、割込みがマスクされている間、タスクの切換えは起こらず CPU を占有することができます。割込みマスク中はタスク独立部として動作するため、タスク実行状態と以下の内容が異なります。

- タスク部用のシステムコール
  タスク部用のシステムコールは発行できません。パラメータチェック機能を組み込んでいる場合、E_CTXとなります。パラメータチェック機能を組み込んでいない場合、システムの正常な動作は保証されません。
- タスク独立部用のシステムコール
  タスク独立部用の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。
- タスク切り替えの遅延
  割込みマスク中にタスク切り替えが必要となっても、タスク実行状態に戻る(割込みマスクを0でchg_imsシステムコール発行)まで遅延されます。
- 割込みマスク値
  セットアップファイルに定義したカーネル割込みマスクレベルより高い割込みマスクレベルからは、chg_ims以外のシステムコールを発行しないでください。

5.2.2 タスクの登録

タスクの登録は、セットアップテーブルにタスク先頭アドレス、初期優先度、タスクスタックサイズ、タスク初期状態および拡張情報を設定します。

タスクの登録の詳細については、「6.2.2 タスクの登録」を参照してください。

5.3 割込みハンドラ

5.3.1 割込みハンドラの作成

割込みハンドラは、割込みが発生すると実行するプログラムです。
割込みが発生すると、制御はカーネルの介入なしに割込みハンドラに渡されます。割込みハンドラでは、割込み発生時レジスタを保証しなければなりません。
割込みハンドラは通常以下の手順で作成してください。

(a) **割込みハンドラで使用するレジスタの退避**

- 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
    (割込みハンドラでスタックを使用しない場合は不要)
  - レジスタの内容保存

(b) **割込み処理**

(c) **割込みハンドラで使用したレジスタの復帰**

- レジスタの内容復帰
- スタックポインタの変更(割込みハンドラでスタックを使用しない場合は不要)

(d) ret_int **システムコールの発行(カーネル割込みマスクレベル以下の場合)、または** RTE **命令実行(カーネル割込みマスクレベルより高い場合)**

H8S、H8/300 シリーズ C コンパイラの割込み関数の作成機能(#pragma interrupt)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。

#pragma interrupt を用いて割込みハンドラとなる関数を宣言します。図 5.3 では、inthdrxx 関数を割込みハンドラとして宣言しています。

割込みの仕様として「スタック切り換え指定」と「割込み関数終了指定」を行います。

「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=＜ｱﾄﾞﾚｽ＞”で指定します。スタック領域は、割込みレベル毎に固有の領域を指定してください。

「割込み関数終了指定」は、割込みハンドラ終了時の復帰方法の指定であり、割込みハンドラ終了時に ret_int システムコールの発行、または RTE 命令を実行する必要があります。

カーネル割込みマスクレベル以下の割込みハンドラの場合、”sy=$ret_int”と指定します。これにより、割込みハンドラ終了時、ret_int システムコールを呼び出す“ jmp @ret_int”の命令が実行されます。

また、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場合、何も記述する必要はありません。

詳細については、『H8S，H8/300 シリーズ C/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。

C 言語で記述したプログラムはコンパイル時の CPU オプション／環境変数指定により、ノーマルモード、アドバンストモードのどちらにも使用できます。

図 5.3 に割込みハンドラの記述例を示します。

```
#include    “hi2000.h”
extern      VH hi_intstkxx[];
const VP P_intstkxx =(VP)&hi_intstkxx[60];


#pragma interrupt(inthdrxx(sp=P_intstkxx,sy=$ret_int))
void inthdrxx(void)                             /* 割込みﾊﾝﾄﾞﾗ関数のﾃﾞｰﾀ型は void */
{
ID          tskid;                              /* ﾀｽｸ id           */
ER          ercd;                               /* ｴﾗｰｺｰﾄﾞ          */
UINT        imask;                              /* 割込みﾏｽｸ値      */
              .
              .
            ercd = chg_ims(imask);              /* ﾀｽｸ独立部から発行できるｼｽﾃﾑｺｰﾙ */
              .
              .
            ercd = iwup_tsk(tskid);             /* ﾀｽｸ独立部から発行できるｼｽﾃﾑｺｰﾙ */
              .
              .
}
```

図 5.3　C 言語による割込みハンドラの記述例

表 5.3 に割込みハンドラの処理条件を示します。

表 5.3 割込みハンドラの処理条件

項番	項目	内容
1	割込みマスク	割込みマスク状態で起動されます。
2	使用できるレジスタ	ER0～ER6、MACH、MACL(MACH,MACL は 2600CPU のみ)が使用できます。 割込みハンドラ処理終了前に起動時の値に戻してください。
3	スタックポインタ	割込み発生元に制御も戻すときは起動時と同じ値にしてください
4	使用できるシステムコール	タスク独立部から発行可能なシステムコール。 ただし、カーネル割込みマスクレベルより高い割込みハンドラおよび NMI 割込みハンドラからシステムコールを発行することはできません。
5	使用できるスタック領域	システム構築時に確保し、起動時にスタックを切り替えてください。 同一割込みレベルの割込みハンドラでは、スタックを共有することができます。
6	終了方法	ret_int システムコールにより処理を終了します。 終了時は、スタックを起動時と同じ状態にしてください。 カーネル割込みマスクレベルより高い割込み、および NMI 割込みハンドラは RTE 命令で終了してください。

割込みハンドラを作成する場合、以下の項目に注意してください。

(1) 割込みマスクレベルの保証

カーネルでは、4 種類の割込み制御モードを使用することができます。
割込みのマスクレベルは CCR レジスタ、EXR レジスタの割込み制御ビットで表わされます。
割込みハンドラでは、各割込み制御モードによって割込みマスクレベルに対する処理が異なります。

(a) 割込み制御モード 0

CCR レジスタの I ビットのみで制御するモードです。割込みハンドラが起動されると、CCR レジスタの I ビットがセットされます。割込みハンドラでは、CCR の I ビットをクリアしないでください。I ビットをクリアした場合、システムの動作は保証されません。

(b) 割込み制御モード 1

CCR レジスタの I ビット、UI ビットで制御するモードです。割込みハンドラが起動されると、CCR レジスタの I ビット、UI ビットがセットされます。

コントロールレベル 0 の割込みハンドラでは、CCR の UI ビットをクリアして割込みマスクレベルを変更し、コントロールレベル 1 の割込みを受け付けるように設定してください。コントロールレベル 0 の割込みハンドラで CCR レジスタの I ビットをクリアした場合、システムの動作は保証されません。また、コントロールレベル 1 の割込みハンドラでは、割込みマスクレベルを変更しないでください。

(c) 割込み制御モード 2

EXR レジスタの I0～I2 ビットで制御するモードで、CCR レジスタの I ビット、UI ビットは無視されます。割込みハンドラが起動されると、EXR レジスタの I0～I2 ビットが発生した割込みのレベルでマスクレベルがセットされます。

(d) 割込み制御モード 3

CCR レジスタの I ビット、UI ビットと EXR レジスタの I0～I2 ビットで制御するモードです。割込みハンドラが起動されると、CCR レジスタの I ビット、UI ビット、EXR レジスタの I0～I2 ビットがセットされます。

コントロールレベル 0 の割込みハンドラでは、CCR の I ビット、UI ビットをクリアして割込みマスクレベルを変更し、優先度レベルの高い割込みを受け付けるようにしてください。割込みハンドラでは、EXR レジスタを変更しないでください。EXR レジスタを変更した場合、システムの動作は保証されません。

(2) カーネル割込みマスクレベル

カーネルには、内部に持つ情報に矛盾が生じるのを防ぐため、割込みをマスクして実行する部分(クリティカルセクション)があります。カーネルのクリティカルセクション実行中に発生した割込みは、通常はクリティカルセクションが終わるまで受け付けが遅延されます。しかし、カーネル割込みマスクレベルより高い割込みについては、クリティカルセクション実行中にも即座に受け付けられます。

【注】カーネル割込みマスクレベルよりも高いレベルの割込みハンドラからは、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。
また、カーネル割込みマスクレベルよりも高いレベルの割込みハンドラからの復帰は RTE 命令を実行してください。

【注】割込み制御モードを 3 で、カーネル割込みマスクレベルとしてレベル 7 を使用した場合、コントロールレベル 1 の割込みハンドラからはシステムコールを発行できません。

(3) 割込みの注意事項

- 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。
- 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、システムコールを発行することはできません。また、NMI(Non Maskable Interrupt)の割込みハンドラでも、システムコールは発行できません。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から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。
- カーネル割込みマスクレベル以下の割込みハンドラからの復帰は、ret_int システムコールを使用してください。ret_int システムコール以外を使用した場合、システムの正常な動作は保証されません。

5.3.2 割込みハンドラの登録

割込みハンドラの登録は、該当する割込みベクタテーブルに割込みハンドラの先頭アドレスを設定します。割込みハンドラは、割込み発生時にカーネルの介入を受けずに直接制御が移ります。
割込み発生要因に関する詳細は当該ハードウェアマニュアルを参照してください。

図 5.4 に割込みベクタテーブルと割込みハンドラとの関連を示します。

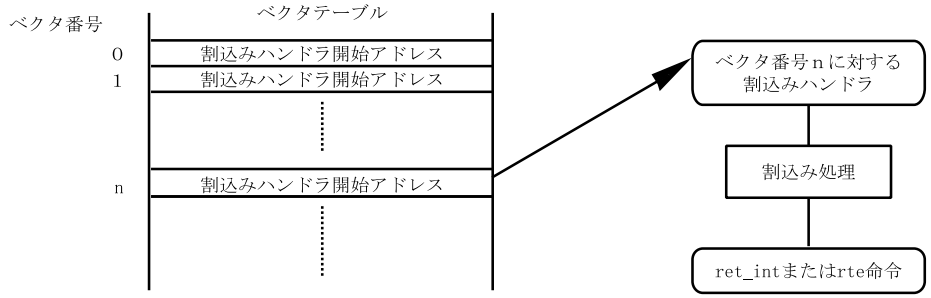


図 5.4 割込みベクタテーブルと割込みハンドラとの関連

割込みハンドラ登録の詳細については、「7. 割込みベクタテーブルの作成」を参照してください。

5.4 未定義割込みハンドラ

5.4.1 未定義割込みハンドラの作成

未定義割込みハンドラは、システムで予期せぬ割込みが発生すると実行するプログラムです。
サンプル提供の未定義割込みハンドラのプログラムは、カーネルの未定義割込み処理(jsr @_H_ilint)をサブルーチンコールしシステム異常終了となります。

システム異常終了時の未定義割込みの情報については、「5.9 システム異常終了ルーチン」および当該ハードウェアマニュアルを参照してください。

未定義割込みハンドラは、割込みハンドラと同様に作成することができます。

5.4.2 未定義割込みハンドラの登録

未定義割込みハンドラの登録の詳細については、「7. 割込みベクタテーブルの作成」を参照してください。

サンプルとして提供している未定義割込みハンドラのファイルは「sample¥***nnnnz***smp¥***nnnnz***ili.src」です。

5.5 周期起動ハンドラ

5.5.1 周期起動ハンドラの作成

周期起動ハンドラは、指定した周期時間ごとに起動されるタスク独立部のハンドラです。
指定した周期時間が経過すると、タイマ割込みハンドラから周期起動ハンドラが起動されます。周期起動ハンドラでは、ハンドラ起動時のレジスタを保証しなければなりません。
周期起動ハンドラは通常以下の手順で作成してください。

- 周期起動ハンドラで使用するレジスタの退避
  - レジスタの内容保存
  - (スタックはタイマ割込みハンドラのスタックを使用します)
- 周期処理
- 周期起動ハンドラで使用したレジスタの復帰
  - レジスタの内容復帰
- RTS 命令の実行

H8S、H8/300 シリーズ C コンパイラの拡張機能(#pragma asm)を用いることにより、周期起動ハンドラを C 言語で記述できます。C 言語で記述したプログラムはコンパイル時の CPU オプション／環境変数指定により、ノーマルモード、アドバンストモードのどちらにも使用できます。
コンパイル時に code=asmcode オプションを用いてアセンブリプログラムの出力を指定してください。アセンブリプログラムの出力を行うため、周期起動ハンドラの C ソースプログラムは、他の C ソースプログラムと別のファイルとして作成することを推奨します。

図 5.5 に周期起動ハンドラの記述例を示します。

```
#include    “hi2000.h”
void        cyc_hdr(void)
#pragma asm
            stm.l   (er0-er1),@-sp          ;: er0,er1 をｽﾀｯｸに退避
            bsr     cychdr_main:8           ;: 関数の呼び出し
;
            ldm.l   @sp+,(er0-er1)          ;: er0,er1 を回復
            rts                             ;: rts 命令を実行
cychdr_main:
#pragma endasm
{
            /* 周期起動ハンドラ処理 */
}
```

図 5.5　C 言語による周期起動ハンドラの記述例

表 5.4 に周期起動ハンドラの処理条件を示します。

表 5.4 周期起動ハンドラの処理条件

項番	項目	内容
1	割込みマスク	タイマ割込みマスクレベルで起動されます。
2	使用できるレジスタ	ER0～ER6、MACH、MACL(MACH,MACL は 2600CPU のみ)が使用できます。 周期起動ハンドラ処理終了前に、起動時の値に戻してください。 C 言語で記述する場合は、図 5.5 に示すように ER0、ER1 を退避･復帰してください。
3	スタックポインタ	タイマ割込みハンドラのスタックを使用します。 周期起動ハンドラ終了前に、起動時の値に戻してください。
4	使用できるシステムコール	タスク独立部から発行可能なシステムコール。
5	使用できるスタック領域	システム構築時にタイマ割込みハンドラスタックに、周期起動ハンドラで使用するサイズを加算して確保してください。
6	終了方法	RTS 命令により処理を終了します。 終了時は、スタックを起動時と同じ状態にしてください。

周期起動ハンドラを作成する場合、以下の項目に注意してください。

(1) 周期起動ハンドラの多用による弊害

周期起動ハンドラは、タイマ割込みマスクレベルでマスクした状態で実行されます。周期時間に達したハンドラが複数ある場合は、タイマ割込みマスクレベルでハンドラの処理が行われます。これにより、以下のような弊害をもたらす可能性があります。

- システムクロックの遅れ
- タイマ割込みマスクレベル以下の割込みに対するレスポンスの悪化

これを避けるために、以下を遵守してください。

- タイマ割込みの周期を極端に短くしないでください。
- 周期起動ハンドラの処理は、可能な限り短くしてください。
- 周期起動ハンドラの周期は、なるべく大きな値にしてください。

極端な例としては、ある周期起動ハンドラの周期時間が 1 で、そのハンドラ処理時間がタイマ周期時間以上かかるような場合、永久にその周期起動ハンドラだけが実行されることになり、事実上ハングアップします。

5.5.2 周期起動ハンドラの登録

カーネルはセットアップテーブルに登録された内容に従って、周期時間のカウントと周期起動ハンドラの実行制御を行います。

周期起動ハンドラの登録は、セットアップテーブルに定義することで行います。
セットアップテーブルの周期起動ハンドラ定義部の例を参考に、登録を行ってください。

周期起動ハンドラの登録の詳細については、「6.2.5 周期起動ハンドラの登録」を参照してください。

5.6 CPU 初期化ルーチン

5.6.1 CPU 初期化ルーチンの作成

CPU 初期化ルーチンは、カーネル起動前に CPU を初期化するためのプログラムです。
CPU 初期化ルーチンは通常以下の手順で作成してください。

- スタックポインタの設定
- CPU 初期化処理
- デーモン初期化処理の呼び出し(デバッギングエクステンション(DX)使用時)
- カーネルの初期化処理(_H_2S_INIT)にジャンプ(jmp @_H_2S_INIT)

サンプル提供の CPU 初期化ルーチンは、アセンブリ言語で記述しています。
H8S、H8/300 シリーズ C コンパイラに添付されているサンプルプログラム(インクルードファイル：ディレクトリ名「2600」)を用いて、CPU 初期化ルーチンを C 言語で記述することができます。C 言語で記述したプログラムはコンパイル時の CPU オプション／環境変数指定により、ノーマルモード、アドバンストモードのどちらにも使用できます。

図 5.6 に CPU 初期化ルーチンの記述例を示します。

```
#include "2655s.h"                              /* H8S/2655 用ｲﾝｸﾙｰﾄﾞﾌｧｲﾙ指定 */
void H_2S_INIT(void);                           /* ｶｰﾈﾙの初期化処理の宣言 */
#ifdef DX
  void HI_DEAMON_INI(void);                     /* ﾃﾞｰﾓﾝ初期化処理の宣言  */
#endif
#pragma stacksize 0x012                         /* ｽﾀｯｸｾｸｼｮﾝの宣言        */
#pragma entry H_2S_CPUINI                       /* ｴﾝﾄﾘ関数の宣言         */
void H_2S_CPUINI(void)
{
            SYSCR.BIT.INTM = 3;                 /* 割込み制御ﾓｰﾄﾞの設定   */
            MSTPCR.BIT.B13 = 1;                 /* ﾓｼﾞｭｰﾙｽﾄｯﾌﾟﾋﾞｯﾄをｸﾘｱ   */
 #ifdef DX
   HI_DEAMON_INI();                             /* ﾃﾞｰﾓﾝ初期化処理の呼び出し       */
 #endif
            H_2S_INIT();                        /* ｶｰﾈﾙの初期化処理にｼﾞｬﾝﾌﾟ        */
}
```

図 5.6　C 言語による CPU 初期化ルーチンの記述例

表 5.5 に CPU 初期化ルーチンの処理条件を示します。

表 5.5 CPU 初期化ルーチンの処理条件

項番	項目	内容
1	割込みマスク	リセット直後は NMI を含めた全ての割込みがマスクされます。
2	使用できるレジスタ	全レジスタ
3	スタックポインタ	スタックポインタの設定は本処理の先頭命令で設定してください。 例 ＞ mov.l #xx：32,sp スタックポインタ初期化前に NMI 割込みが発生すると動作の保証はできません。
4	使用できるシステムコール	カーネルが起動していないため、システムコールは使用できません。
5	使用できるスタック領域	必要に応じてシステム構築時に確保し、起動時にスタックを設定してください。
6	終了方法	カーネルの初期化処理にジャンプして処理を終了します。 jmp @_H_2S_INIT

5.6.2 CPU 初期化ルーチンの登録

CPU 初期化ルーチンを登録する場合、CPU 初期化ルーチンのプログラムの先頭にラベル名を付け、export 宣言してください。

CPU 初期化ルーチンは、リセットベクタ*1(ベクタ番号 0，1)に登録します。

- ベクタ番号 0：パワーオンリセット
- ベクタ番号 1：マニュアルリセット

【注】 使用する H8S シリーズマイコンによっては、マニュアルリセットのないものがあります。その場合、パワーオンリセットに登録するだけで構いません。リセットに関する詳細は、当該ハードウェアマニュアルを参照してください。

CPU 初期化ルーチンの登録の詳細については、「7. 割込みベクタテーブルの作成」を参照してください。

サンプル提供している CPU 初期化ルーチンのファイルは「sample¥***nnnnz***smp¥***nnnnz***cpu.src」です。

5.7 タイマ初期化ルーチン

タイマ初期化ルーチンは、時間管理機能を使用する場合にタイマ割込みハンドラとともに必要となります。

タイマ初期化ルーチン、およびタイマ割込みハンドラの詳細については、「付録 C デバイスドライバ」を参照してください。

5.8 システム初期化ハンドラ

5.8.1 システム初期化ハンドラの作成

システム初期化ハンドラは、カーネルの初期化処理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、初期起動タスクが起動する前に、資源の初期化、およびハードウェアの初期化などを行います。

システム初期化ハンドラは通常以下の手順で作成してください。

- システム初期化ハンドラで使用するレジスタの退避
  - レジスタの内容保存（C 言語プログラム（関数）のレジスタ保証規則に合わせて保証してください）
  - (スタックは OS のスタックを使用します)
- システム初期化処理
  - セマフォにて管理する資源数の初期化等
- システム初期化ハンドラで使用したレジスタの復帰
  - レジスタの内容復帰
- RTS 命令の実行

図 5.7 にシステム初期化ハンドラの記述例を示します。

```
#include    “hi2000.h”
void        HIPRG_SYSINI(void)          /* ラベル名「 HIPRG_SYSINI 」*/
{
            /* システム初期化ハンドラ処理 */
}
```

図 5.7　C 言語によるシステム初期化ハンドラの記述例

表 5.6 にシステム初期化ハンドラの処理条件を示します。

表 5.6 システム初期化ハンドラの処理条件

項番	項目	内容
1	割込みマスク	割込みマスク状態(カーネル割込みマスクレベル)で起動されます。 システム初期化ハンドラ実行中は割込みマスクを変更しないでください。
2	使用できるレジスタ	レジスタの保証は C 言語プログラム（関数）のレジスタ規則に合わせてください。
3	スタックポインタ	カーネルに制御を戻すときは起動時と同じ値にしてください。
4	使用できるシステムコール	タスク独立部から発行可能なシステムコール(ret_int システムコールは除く)
5	使用できるスタック領域	OS 用スタック領域を使用します。OS 用スタックサイズにシステム初期化ハンドラが使用するスタックサイズを加算してください。 システム初期化ハンドラのスタックサイズの算出は、割込みハンドラ用スタック領域サイズの算出表を使用してください。
6	終了方法	RTS 命令により処理を終了します。 終了時は、スタックを起動時と同じ状態にしてください。

5.8.2 システム初期化ハンドラの登録

カーネルは、システム初期化ハンドラのラベル『_HIPRG_SYSINI』に設定されている値を、システム初期化ハンドラの先頭アドレスとして実行します。

システム初期化ハンドラを登録する場合、システム初期化ハンドラのプログラムの先頭にラベル名『_HIPRG_SYSINI』を付け、export 宣言してください。

システム初期化ハンドラを未登録にする場合、ラベル名『_HIPRG_SYSINI』に 0 を equate 定義し、export 宣言してください。

サンプル提供しているシステム初期化ハンドラは未登録で、ラベル名『_HIPRG_SYSINI』に 0 を equate 定義し export 宣言しています。
提供のシステム初期化ハンドラのファイルは「sample¥***nnnnz***smp¥***nnnnz***use.src」です。

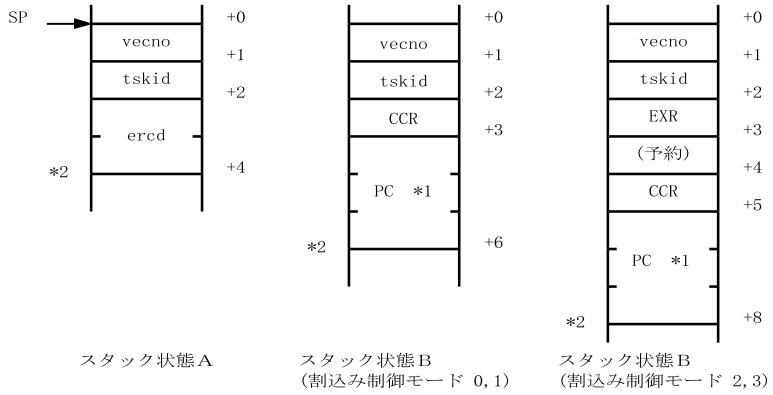
5.9 システム異常終了ルーチン

5.9.1 システム異常終了ルーチンの作成

システム異常終了ルーチンは、システム実行中に致命的なエラーが発生した場合に起動されるプログラムです。提供のシステム異常終了ルーチンは、割込みマスク(カーネル割込みマスクレベル)の状態で無限ループに入ります。

システム異常終了ルーチンが起動される場合、スタックにはエラー情報が設定されています。異常終了に応じたプログラムを作成する場合、スタックのエラー情報を参照してください。システム異常終了時のスタック状態は、異常終了の原因によって 2 種類に分かれます。

図 5.8 にシステム異常終了時のスタック状態を示します。



*1：ノーマルモードでは、下位16ビットが有効になります。
*2：スタックが奇数値の場合は最下位ビットが無視されて情報が積まれます。

図 5.8 システム異常終了時のスタック状態

表 5.7 にシステム異常終了の原因とスタックに積まれる情報を示します。

表 5.7 システム異常終了となる原因

ｽﾀｯｸ状態	原因	vecno	tskid	ercd／ｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ
A	セットアップ情報エラー	H'00	H'00	H'0000〜H'0FFF
	タイマ未サポート	H'00	H'00	H'F9ED
	タスク独立部からの ext_tsk システムコール発行	H'00	H'00	H'FFEB
	タスク実行状態、または CPU ロック状態からの ret_int システムコール発行	H'00	tskid (H'00〜H'FF)	H'FFBB
B	未定義割込みの発生	割込みﾍﾞｸﾀ番号	tskid*1	発生時の CCR、EXR*2、PC

【注】 *1 タスク部の場合は tskid、タスク独立部の場合は 0 が設定されます。
*2 割込み制御モード 0、1 では、EXR レジスタはスタックされません。

表 5.8 にセットアップ情報エラーの不正内容および ercd を示します。

表 5.8 セットアップ情報不正内容一覧(1/2)

項番	不正項目		ercd
1	アドレス不正	カーネル用スタックポインタ(_HI_OS_SP)が 0 または奇数	H'0101
		タイマ割込み用スタックポインタ(_HI_TIM_SP)が 0 または奇数	H'0102
		カーネル作業領域の先頭アドレス(セクション名「hi8_2s_ram」)が 0 または奇数	H'0103
		TIMCB 領域(_HI_TIMCB)の先頭アドレスが 0 または奇数	H'0104
		TIMCB2 領域(_HI_TIMCB2)の先頭アドレスが 0 または奇数	H'0105
		TCB 領域(_HI_TCB)の先頭アドレスが 0 または奇数	H'0106
		TCB2 領域(_HI_TCB2)の先頭アドレスが 0 または奇数	H'0107
		FLGCB 領域(_HI_FLGCB)の先頭アドレスが 0 または奇数	H'0108
		SEMCB 領域(_HI_SEMCB)の先頭アドレスが 0 または奇数	H'0109
		MBXCB 領域(_HI_MBXCB)の先頭アドレスが 0 または奇数	H'010A
		MPFCB 領域(_HI_MPFCB)の先頭アドレスが 0 または奇数	H'010B
		MPLCB 領域(_HI_MPLCB)の先頭アドレスが 0 または奇数	H'010C
		トレーススタックポインタ(_HI_TRC_SP)が 0 または奇数	H'010D
		トレース管理領域(TBACB)の先頭アドレスが 0 または奇数	H'010E
		TIMCB3 領域(_HI_TIMCB3)の先頭アドレスが 0 または奇数	H'010F
		CYHCB 領域(_HI_CYHCB)の先頭アドレスが 0 または奇数	H'0110
2	ルーチンアドレス不正	システム初期化ハンドラの先頭アドレス(_HIPRG_SYSINI)が奇数	H'0201
		タイマ初期化設定ルーチンの先頭アドレス(_HIPRG_TIMINI)が奇数	H'0202
3	設定値不正(範囲外)	CPU 割込み制御モード(CPUINTM)が 4 以上	H'0301
		カーネル割込みマスクレベル(IMASK)が 9 以上	H'0302
		優先度定義数(MAXPRI)が 32 以上	H'0303
		タスク定義数(TSKCNT)が 256 以上	H'0304
		イベントフラグ定義数(FLGCNT)が 256 以上	H'0305
		セマフォ定義数(SEMCNT)が 256 以上	H'0306
		メイルボックス定義数(MBXCNT)が 256 以上	H'0307
		固定長メモリプール定義数(MPFCNT)が 256 以上	H'0308
		可変長メモリプール定義数(MPLCNT)が 256 以上	H'0309
		周期起動ハンドラ定義数(CYHCNT)が 256 以上	H'030A
4	セットアップテーブルアドレス不正(0 か奇数)	タスク定義テーブル(_HI_TDT)の先頭アドレスが 0 または奇数	H'0401
		固定長メモリプール定義テーブル(_HI_MPFDT)の先頭アドレスが 0 または奇数	H'0402
		可変長メモリプール定義テーブル(_HI_MPLDT)の先頭アドレスが 0 または奇数	H'0403
		未定義割込みハンドラ(_HI_ILT)の先頭アドレスが 0 または奇数	H'0404
		トレースバッファ情報テーブル(INITRC)の先頭アドレスが 0 または奇数	H'0405
		周期起動ハンドラ定義テーブル(_HI_CYCDT)の先頭アドレスが 0 または奇数	H'0406

表 5.8 セットアップ情報不正内容一覧(2/2)

項番	不正項目		ercd
5	セットアップテーブル項目不正	タスク初期優先度(ITSKPRI)が 0 または優先度定義数より大きい値	H'0501
		タスク先頭アドレス(TSKADR)が 0 または奇数	H'0502
		タスクスタックポインタ(ITSKSP)が 0 または奇数	H'0503
		メモリブロック長(BLFLEN)が 0、奇数または 65530 バイト以上	H'0504
		固定長メモリプールアドレス(MPF?_TOP)が 0 または奇数	H'0505
		可変長メモリプールサイズが 0、奇数または 16 バイト以下	H'0506
		可変長メモリプールアドレス(MPL?_TOP)が 0 または奇数	H'0507
		トレースバッファアドレス(TRACE BUFFER ADDRESS)が 0 または奇数	H'0508
		周期起動ハンドラの先頭アドレスが 0 または奇数	H'0509
		周期起動ハンドラの周期起動時間間隔が 0 または H'80000000 以上	H'050A
		周期起動ハンドラの活性情報が不正(0 または 1 以外)	H'050B
6	セットアップテーブル項目不正(拡張情報)	タスク定義数(TSKCNT)とタスク拡張情報定義数(TSKECNT)が不一致	H'0601
		イベントフラグ定義数(FLGCNT)とイベントフラグ拡張情報定義数(FLGECNT)が不一致	H'0602
		セマフォ定義数(SEMCNT)とセマフォ拡張情報定義数(SEMECNT)が不一致	H'0603
		メイルボックス定義数(MBXCNT)とメイルボックス拡張情報定義数(MBXECNT)が不一致	H'0604
		固定長メモリプール定義数(MPFCNT)と固定長メモリプール拡張情報定義数(MPFECNT)が不一致	H'0605
		可変長メモリプール定義数(MPLCNT)と可変長メモリプール拡張情報定義数(MPLECNT)が不一致	H'0606
		周期起動ハンドラ定義数(CYHCNT)と周期起動ハンドラ拡張情報定義数(CYHECNT)が不一致	H'0607

システム異常処理ルーチンは C 言語で記述できます。処理はユーザシステム依存で作り方によって変わります。

スタックポインタやレジスタを回復する場合は、他のハンドラを参考にして回復処理を記述してください。

5.9.2 システム異常終了ルーチンの登録

システム異常終了ルーチンを登録する場合、プログラムの先頭にラベル名『_HIPRG_ABNOML』を付け、export 宣言してください。

システム異常終了ルーチンは、必ず登録してください。

【注】『_HIPRG_ABNOML』に 0 または奇数アドレスを equate 定義した場合、システムの正常な動作は保証されません。

サンプル提供しているシステム異常終了ルーチンのファイルは「sample¥***nnnnz***smp¥***nnnnz***use.src」です。

5.10 システムアイドリングルーチン

5.10.1 システムアイドリングルーチンの作成

システムアイドリングルーチンは、実行可能状態のタスクが存在しない場合、割込みからタスクが起床されるのを待つために実行するプログラムです。

システムアイドリング状態では、割込みマスクは解放(0)に設定されます。サンプルとして提供しているシステムアイドリングルーチンでは、BRA 命令と SLEEP 命令をのいずれかを選択することができます。システムアイドリング状態で、CPU の低消費電力機能を利用したい場合は SLEEP 命令を使用してください。

5.10.2 システムアイドリングルーチンの登録

システムアイドリングルーチンのプログラムの先頭には、ラベル名『_H_SYSTEM_IDLE』を付け export 宣言してください。システムアイドリングルーチンは、必ず作成してください。

システムアイドリングルーチンが無い場合、システムの正常な動作は保証されません。

サンプル提供しているシステムアイドリングルーチンのファイルは
「sample¥***nnnnz***smp¥***nnnnz***use.src」です。

6. セットアップテーブルの作成

6.1 概要

システムを構築する場合に必要な情報をセットアップテーブルに定義します。
セットアップテーブルは、ユーザ定義部とカーネルのシステム定義部から構成されています。
ユーザ定義部は、タスク定義数などユーザ環境の定義を行うテーブルです。ユーザシステムの環境に合せて修正してください。
システム定義部は、カーネルが使用する外部定義シンボルおよび作業領域などを定義しています。システム定義部はユーザ定義部を基に自動的に更新(設定)されますので、修正はしないでください。修正した場合、システムの正常な動作は保証されません。
セットアップテーブルのファイルは「 sample¥***nnnnz***smp¥***nnnnz***sup.src 」です。

6.2 ユーザ定義部

以下にユーザ定義部の設定項目を、H8S/2655 マイコンのアドバンストモードを例に示します。

- 定数定義部
  定数定義部には、本カーネルの機能(同期通信機能、時間管理機能等)を使用するために必要な情報を登録します。
- タスク登録部
  タスク登録部には、タスクを使用するために必要な情報を登録します。
- 固定長メモリプール登録部
  固定長メモリプール登録部には、固定長メモリプールを使用するために必要な情報を登録します。
- 可変長メモリプール登録部
  可変長メモリプール登録部には、可変長メモリプールを使用するために必要な情報を登録します。
- 周期起動ハンドラ登録部
  周期起動ハンドラ登録部には、周期起動ハンドラを使用するために必要な情報を登録します。
- システムコールトレース機能登録部
  システムコールトレース機能登録部には、システムコールトレース機能を使用するために必要な情報を登録します。
- 拡張情報登録部
  拡張情報登録部には、タスク、イベントフラグ、セマフォ、メイルボックス、固定長/可変長メモリプール、周期起動ハンドラの拡張情報を登録します。

登録／未登録に関わらず、すべての項目を定義してください。
定義されていない場合、結合時に未定義エラーとなります。

6.2.1 定数定義部の登録

ここでは、カーネルの各種機能(同期通信機能、時間管理機能など)を使用するために必要な情報を登録します。なお、サンプルとして以下の内容で登録することを前提に定義しています。

表 6.1 に定数定義部の登録情報を示します。

表 6.1 定数定義部の登録情報一覧

内 容	ラベル名	定義情報	備考
割込み制御モード	CPUINTM	**モード**＝3	
カーネル割込みマスクレベル	IMASK	**レベル**＝6	
優先度定義数	MAXPRI	**タスクの最も低い優先度**＝31	
イベントフラグ定義数	FLGCNT	**最大イベントフラグ** ID＝4	
セマフォ定義数	SEMCNT	**最大セマフォ** ID＝4	
メイルボックス定義数	MBXCNT	**最大メイルボックス** ID＝4	
OS **スタックサイズ**	OSSTKSIZ	OS **使用スタックサイズ**＝52	18+20+6+8
タイマスタックサイズ	TIMSTKSIZ	**タイマドライバ使用スタックサイズ**＝64	40+10+6+8
トレーススタックサイズ	TRCSTKSIZ	**トレース機能使用スタックサイズ**＝40	26+6+8
タイムアウト機能定義	TTMOUT	USE(**使用**)	

【注】 ラベル名は、本カーネルが参照しているため、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。

図 6.1 に OS スタックサイズの算出方法を示します。算出方法については、「付録 A. メモリ容量の算出」を参照してください。 タイマスタックサイズ、およびトレーススタックサイズも同様に算出表から算出しています。

表A.2 OS用スタック領域サイズの算出表

内 訳	計算式	容量(ﾊﾞｲﾄ)	備 考
OSが使用するスタックサイズ	18(アドバンストモード)	18	常に必要です
割込み用スタックサイズ	10 × LOWINTNST *1 ＋ 6 × UPPINTNST *2	10 × 2 ＋ 6 × 1	割込み制御ﾓｰﾄﾞ3使用, IMASK以下割込みのネスト=2, IMASK以上割込みのネスト=1
未定義割込み用スタックサイズ *3	8	8	未定義割込み発生有り
合 計		52	18＋20＋6＋8

【注】 *1 カーネル割込みマスクレベル以下の割込みのネスト数
*2 カーネル割込みマスクレベルより高い割込み(NMIを含む)のネスト数
*3 未定義割込みが発生する場合必要

図 6.1 「OS 用スタック領域算出表」からの算出

図 6.2 に定数定義部を示します。□ **部以外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。**

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    VALUE define section                                                  %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage --------------------------------------------------------------
;LABEL                  VALUE   ;:[ RANGE ]     ;: COMMENT
;---------------------------------------------------------------------------
CPUINTM:        .assign    3
```

CPU の割込み制御モードを定義します。定義できる値は０～３です。詳細は、使用するCPUのハードウェアマニュアルを参照してください。

```
IMASK:          .assign    6
```

カーネル割込みマスクレベルを定義します。定義できる値は CPU の割込み制御モード(CPUINTM)によって変わります。

CPUINTM	設定出来るIMASK値
0	０～１
1	０～３
2	０～７
3	０～８

カーネル割込みマスクレベルより高いレベルの割込みは、カーネル実行中に発生しても遅延されずに受け付けられます。

```
MAXPRI:         .assign    31
```

構築するシステムでタスクの最も低い優先度を定義します。定義できる値は１～３１までです。値の小さい方が高い優先度です。

```
FLGCNT:         .assign    4
```

最大イベントフラグIDを定義します。定義できる値は０～２５５です。０を設定した場合、イベントフラグは未登録となります。

```
SEMCNT:         .assign    4
```

最大セマフォIDを定義します。定義できる値は０～２５５です。０を設定した場合、セマフォは未登録となります。

```
MBXCNT:         .assign    4
```

最大メイルボックスIDを定義します。定義できる値は０～２５５です。０を設定した場合、メイルボックスは未登録となります。

```
OSSTKSIZ:       .equ       18+(10*2)+(6*1)+8
```

OSが使用するスタック領域のサイズを定義します。
詳細は「付録Ａ．メモリ容量の算出」を参照してください。

```
TIMSTKSIZ:      .equ       40+(10*1)+(6*1)+8
```

タイマドライバが使用するスタック領域のサイズを定義します。詳細は「付録Ａ．メモリ容量の算出」を参照してください。
なお、タイマドライバを使用しない場合は、０を指定してください。

```
TRCSTKSIZ:      .equ       26+(6*1)+8
;
```

トレース機能が使用するスタック領域のサイズを定義します。
詳細は「付録Ａ．メモリ容量の算出」を参照してください。
なお、トレース機能を使用しない場合は、０を指定してください。

```
TTMOUT:         .assign    USE
;
```

タイムアウト機能の使用/未使用を定義します。
USE　　：使用
NOTUSE ：未使用
TIMSTKSIZに０を定義した場合は無効となります。

図 6.2　定数定義部

6.2.2 タスクの登録

ここではタスクを登録するための各種定義を行います。なお、サンプルとして以下のタスクを登録することを前提に説明しています。

- タスク数：5(タスク ID1～5 を使用)
  - タスク 1 とタスク 2 はデバッギングエキステンション(DX)用チュートリアルタスクを登録
  - タスク 3～5 は未登録
- タスクスタック
  - タスク最小スタックサイズ：86 バイト
  - タスク 4 とタスク 5 は共有スタック使用

表 6.2 にタスクの定義内容を示します。

表 6.2　タスクの定義内容

タスク ID	先頭アドレス	初期状態	初期優先度	スタックポインタ	独自スタック使用サイズ
1	TASKA	休止状態	1	TSK1_SP	36 バイト
2	TASKB	休止状態	2	TSK2_SP	36 バイト
3	無し(0)	未登録	3	TSK3_SP	32 バイト
4	無し(0)	未登録	4	TSK4_SP(共有)	32 バイト
5	無し(0)	未登録	5	TSK4_SP(共有)	32 バイト

図 6.3 にタスク登録部を示します。□ **部以外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。**

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%     TASK define section                                              %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage ------------------------------------------------------------------
;               TASK_TOP_LABEL                    ;: COMMENT
;-------------------------------------------------------------------------------
        .import _TASKA      ← 使用するタスクの先頭アドレスを外部参照シンボルとして宣言します。
        .import _TASKB
;
;------ Usage ------------------------------------------------------------------
;               .res.b      SIZE +TSKSTKSIZ
;TSK?_SP_LABEL: .equ        $
;-------------------------------------------------------------------------------
TSKSTKSIZ:      .equ        50+(10*2)+(6*1)+6+8
```

タスク最小スタックサイズを定義します。タスク最小スタックサイズとはタスクが独自に使用するスタックを除いたサイズです。
詳細は「付録Ａ．メモリ容量の算出」を参照してください。

図 6.3　タスク登録部(1/2)

```
        .section        h2sstack,stack,align=2
                .res.b  (36) +TSKSTKSIZ
TSK1_SP:        .equ    $
                .res.b  8

                .res.b  (36) +TSKSTKSIZ
TSK2_SP:        .equ    $
                .res.b  8

                .res.b  (32) +TSKSTKSIZ
TSK3_SP:        .equ    $
                .res.b  8

                .res.b  (32) +TSKSTKSIZ
TSK4_SP:        .equ    $
                .res.b  8
;
        .section        h2ssetup,code,align=2
_HI_H8S:        .res.b  10                      ;: System Area
;------ Usage ---------------------------------------------------------
;LABEL          .data.b IMOD, ITSKPRI           ;: COMMENT
;               .data.l ITSKADR, ITSKSP         ;: COMMENT
;----------------------------------------------------------------------
NOEXS:          .assign 0
RDY:            .assign 1
DMT:            .assign (-1)
TDTLEN:         .assign 10;<- Not Change !
        .section        h2ssetup,code,align=2
_HI_TDT:        .equ    $-TDTLEN
TDT_TOP:        .equ    $
tdt_id1:        .data.b DMT, 1
                .data.l _TASKA, TSK1_SP

tdt_id2:        .data.b DMT, 2
                .data.l _TASKB, TSK2_SP

tdt_id3:        .data.b NOEXS, 3
                .data.l 0, TSK3_SP

tdt_id4:        .data.b NOEXS, 4
                .data.l 0, TSK4_SP

tdt_id5:        .data.b NOEXS, 5
                .data.l 0, TSK4_SP
TDT_BTM:
TSKCNT:         .equ    (TDT_BTM-TDT_TOP)/TDTLEN
```

【各スタックポインタについて定義要】

タスクスタック領域を確保します。
1行目：使用スタックサイズを定義
　スタックサイズ＝独自に使用するサイズ
　　＋タスク最小スタックサイズ
2行目：ラベル定義
（タスクスタックボトムとなります。）
3行目：共有スタック管理領域を定義
（共有スタック機能使用する場合、８バイトの管理領域を定義する必要があります。共有スタック機能を使用しない場合は定義する必要はありません。）

【各タスクについて定義要】

タスク情報を定義します。
【書式】

```
LABEL   .data.b  IMOD, ITSKPRI
        .data.l  ITSKADR, ITSKSP
```

●LABEL：自由に定義可能（無くても問題なし）
●IMOD（登録／起動要求）
　該当タスクIDのタスクの登録およびシステム起動時のタスク初期状態を定義します。
　　NOEXS(=0)　　：未登録
　　RDY(=1)　　：起動時に実行可能状態
　　DMT(0,1以外)：起動時に休止状態
●ITSKPRI（初期優先度）
　タスクの起動時の優先度を定義します。定義できる値は、1～『MAXPRI』（優先度定義数）の範囲内です。
●ITSKADR（タスク先頭アドレス）
　タスクの先頭アドレスを定義します。
●ITSKSP（タスクスタックポインタ）
　タスクが使用するスタック領域の最終アドレスを定義します。

図 6.3　タスク登録部(2/2)

6.2.3 固定長メモリプールの登録

ここでは固定長メモリプールを登録するための各種定義を行います。なお、サンプルとして以下の固定長メモリプールを登録することを前提に説明しています。

- 固定長メモリプール数：4(固定長メモリプール ID 1～4 を使用)

表 6.3 に固定長メモリプールの定義内容を示します。

表 6.3 固定長メモリプールの定義内容

メモリプール ID	メモリブロック数	メモリブロックサイズ	ラベル名
1	14	12 バイト	MPF1_TOP
2	14	12 バイト	MPF2_TOP
3	14	12 バイト	MPF3_TOP
4	14	12 バイト	MPF4_TOP

図 6.4 に固定長メモリプールの登録部を示します。□ **部以外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。**

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    FIXED-SIZE MEMORYPOOL define section                            %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage ----------------------------------------------------------------
;MB?_CNT_LABEL: .assign VALUE   ;:[  RANGE  ]   ;: COMMENT
;MB?_LEN_LABEL: .assign VALUE   ;:[  RANGE  ]   ;: COMMENT
;----------------------------------------------------------------------------
MB1_CNT:        .assign 14
MB1_LEN:        .assign 12
;
MB2_CNT:        .assign 14
MB2_LEN:        .assign 12
;
MB3_CNT:        .assign 14
MB3_LEN:        .assign 12
;
MB4_CNT:        .assign 14
MB4_LEN:        .assign 12
;
;------ Usage ----------------------------------------------------------------
;MPF?_TOP_LABEL:.res.b  MEMORYPOOL_SIZE         ;: COMMENT
;----------------------------------------------------------------------------
        .section        h2smpf,data,align=2
MPF1_TOP:       .res.b  MB1_CNT * (MB1_LEN + 4)

MPF2_TOP:       .res.b  MB2_CNT * (MB2_LEN + 4)

MPF3_TOP:       .res.b  MB3_CNT * (MB3_LEN + 4)

MPF4_TOP:       .res.b  MB4_CNT * (MB4_LEN + 4)
;
;------ Usage ----------------------------------------------------------------
;LABEL          .data.w BLFCNT, BLFLEN          ;: COMMENT
;               .data.l MPF_TOP_ADDRESS         ;: COMMENT
;----------------------------------------------------------------------------
MPFDTLEN:       .assign 8;<- Not Change !       ;: MPFDT Length
        .section        h2ssetup,code,align=2
_HI_MPFDT:      .equ    $-MPFDTLEN              ;: Fixed-size MemoryPool define table
MPFDT_TOP:      .equ    $
mpfdt_id1:      .data.w MB1_CNT, MB1_LEN
                .data.l MPF1_TOP

mpfdt_id2:      .data.w MB2_CNT, MB2_LEN
                .data.l MPF2_TOP

mpfdt_id3:      .data.w MB3_CNT, MB3_LEN
                .data.l MPF3_TOP

mpfdt_id4:      .data.w MB4_CNT, MB4_LEN
                .data.l MPF4_TOP
MPFDT_BTM:
MPFCNT:         .equ    (MPFDT_BTM-MPFDT_TOP)/MPFDTLEN
```

【各メモリブロックについて定義要】
MB?_CNTにはメモリブロック数を、MB?_LENにはメモリブロックサイズを定義します。なおMB1_CNTやMB1_LENなどのラベルは領域確保と定義テーブルの設定に使用されます。

【各メモリブロックについて定義要】
固定長メモリプール領域を確保します。確保したメモリプールの先頭アドレスにはラベル（例としてMPF?_TOP）を設定してください。
また、例では計算式を使って領域を確保しています。
固定長メモリプール領域のサイズ＝MB?_CNT×（MB?_LEN＋4）
固定長メモリプールを使用しない場合は、□枠すべての行を削除してください。

【各メモリブロックについて定義要】
固定長メモリプール情報を定義します。
【書式】

```
LABEL   .data.w  BLFCNT, BLFLEN
        .data.l  MPF_TOP_ADDRESS
```

●LABEL：自由に定義可能（無くても問題なし）
●BLFCNT：メモリブロック数
●BLFLEN：メモリブロックサイズ
●MPF_TOP_ADDRESS：固定長メモリプール先頭アドレス
メモリプールIDを未登録にする場合、BLFCNTに0を設定してください。
固定長メモリプールを使用しない場合は、□枠すべての行を削除してください。

図 6.4 固定長メモリプール登録部

6.2.4 可変長メモリプールの登録

ここでは可変長メモリプールを登録するための各種定義を行います。なお、サンプルとして以下の可変長メモリプールを登録することを前提に説明しています。

- 可変長メモリプール数：4(可変長メモリプール ID 1～4 を使用)

表 6.4 に可変長メモリプールの定義内容を示します。

表 6.4 可変長メモリプールの定義内容

メモリプール ID	メモリブロックサイズ	ラベル名
1	380 バイト	MPL1_TOP
2	380 バイト	MPL2_TOP
3	380 バイト	MPL3_TOP
4	380 バイト	MPL4_TOP

図 6.5 に可変長メモリプールの登録部を示します。□ **部以外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。**

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    VARIABLE-SIZE MEMORYPOOL define section                            %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage --------------------------------------------------------------
;MPL?_SIZ_LABEL:.assign VALUE    ;:[  RANGE  ]   ;: COMMENT
;--------------------------------------------------------------------------
MPL1_SIZ:       .assign 380
MPL2_SIZ:       .assign 380
MPL3_SIZ:       .assign 380
MPL4_SIZ:       .assign 380
;
;------ Usage --------------------------------------------------------------
;MPL?_TOP_LABEL:.res.b VARIABLE_MEMORYPOOL_SIZE ;: COMMENT
;--------------------------------------------------------------------------
        .section        h2smpl,data,align=2
MPL1_TOP:      .res.b  MPL1_SIZ
MPL2_TOP:      .res.b  MPL2_SIZ
MPL3_TOP:      .res.b  MPL3_SIZ
MPL4_TOP:      .res.b  MPL4_SIZ
;
;------ Usage --------------------------------------------------------------
;LABEL          .data.l BLKSIZ                  ;: COMMENT
;               .data.l VARIABLE_MEMORYPOOL_TOP ;: COMMENT
;--------------------------------------------------------------------------
MPLDTLEN:      .assign 8;<- Not Change !        ;: MPLDT Length
        .section        h2ssetup,code,align=2
_HI_MPLDT:      .equ    $-MPLDTLEN
MPLDT_TOP:      .equ    $
mpldt_id1:      .data.l MPL1_SIZ
                .data.l MPL1_TOP
mpldt_id2:      .data.l MPL2_SIZ
                .data.l MPL2_TOP
mpldt_id3:      .data.l MPL3_SIZ
                .data.l MPL3_TOP
mpldt_id4:      .data.l MPL4_SIZ
                .data.l MPL4_TOP
MPLDT_BTM:
MPLCNT:         .equ    (MPLDT_BTM-MPLDT_TOP)/MPLDTLEN
```

【各メモリブロックについて定義要】
可変長メモリブロックサイズを定義します。
ラベルは領域確保と定義テーブルの設定に使用されます。
可変長メモリプールサイズには16バイトのカーネル管理領域を含んだサイズを指定してください。
指定サイズ=使用するﾒﾓﾘﾌﾟｰﾙｻｲｽﾞ+(16×最大ﾌﾞﾛｯｸ獲得数)

【各メモリブロックについて定義要】
可変長メモリプール領域を確保します。確保したメモリプールの先頭アドレスにはラベル（例としてMPL?_TOP）を設定してください。
可変長メモリプールを使用しない場合は、□枠すべての行を削除してください。

【各メモリブロックについて定義要】
可変長メモリプール情報を定義します。
【書式】

```
LABEL   .data.l  BLKSIZ
        .data.l  VARIABLE_MEMORYPOOL_TOP
```

●LABEL：自由に定義可能（無くても問題なし）
●BLKSIZ：メモリブロック数
●MPL_TOP_ADDRESS：可変長メモリプール先頭アドレス
メモリプールIDを未登録にする場合、BLKSIZに0を設定してください。
可変長メモリプールを使用しない場合、□枠すべての行を削除してください。

図 6.5 可変長メモリプール登録部

6.2.5 周期起動ハンドラの登録

ここでは周期起動ハンドラを登録するための各種定義を行います。なお、サンプルでは周期起動ハンドラをすべて未登録にすることを前提に説明しています。

(a) デバッギングエクステンションを使用しない場合(サンプル提供ファイルの定義)

- 周期起動ハンドラ数：4(周期起動ハンドラ指定番号 1～4 をすべて未登録)

(b) デバッギングエクステンションを使用する場合(サンプル提供ファイルの定義)

周期起動ハンドラ指定番号 5 は、デバッギングエクステンションを使用する場合に登録されます。

- 周期起動ハンドラ数：5(周期起動ハンドラ指定番号 1～4 をすべて未登録)
  (周期起動ハンドラ指定番号5にデバッグデーモンを登録)

表 6.5 にサンプル提供ファイルの周期起動ハンドラの定義内容を示します。

表 6.5 周期起動ハンドラの定義内容

周期起動ハンドラ指定番号	活性状態	起動間隔	ラベル名
1	OFF	0	**無し**(NADR)
2	OFF	0	**無し**(NADR)
3	OFF	0	**無し**(NADR)
4	OFF	0	**無し**(NADR)
5	ON	5	HI_DEAMON_MAIN

(c) 周期起動ハンドラの登録例(図 6.6 周期起動ハンドラ登録部の登録例)

サンプル提供ファイルに追加した周期起動ハンドラの定義内容を以下に示します。

- 周期起動ハンドラアドレスのシンボル(_CYCHDR)を外部参照シンボルとして宣言(import)します。
- 周期起動ハンドラ情報を定義します。
  周期起動ハンドラ指定番号：6
  周期起動ハンドラ活性状態：CYCON(活性)
  周期起動時間間隔　　：10
  周期起動ハンドラアドレス：_CYCHDR

なお、周期起動ハンドラを追加する場合、拡張情報の追加も必要になります。

図 6.6 に周期起動ハンドラ登録部の登録例を示します。本例は、サンプル提供に対して周期起動ハンドラ(周期起動ハンドラ指定番号 6)を追加登録してあります。

□ **部以外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。**

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    cyclic handler define section                                   %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage --------------------------------------------------------------
;       .import CYCHDR_TOP_LABEL               ;: COMMENT
;--------------------------------------------------------------------------
        .import _CYCHDR
;
;------ Usage --------------------------------------------------------------
;LABEL:         .data.w CYC_ACTIVATE           ;: COMMENT
;               .data.l CYC_TIME, CYCHDR_TOP   ;: COMMENT
;--------------------------------------------------------------------------
CYHOFF          .assign 0                      ;:initial cycact data = OFF
CYHON           .assign 1                      ;:initial cycact data = ON
CYHDTLEN        .assign 10;<-Dont't Change!    ;:CYHDT length
;
_HI_CYHDT:      .equ    $-CYHDTLEN
CYHDT_TOP:      .equ    $
cyhdt_no1:      .data.w CYHOFF
                .data.l 0, NADR

cyhdt_no2:      .data.w CYHOFF
                .data.l 0, NADR

cyhdt_no3:      .data.w CYHOFF
                .data.l 0, NADR

cyhdt_no4:      .data.w CYHOFF
                .data.l 0, NADR

                .aifdef DX
cyhdt_no5:      .data.w CYHON
                .data.l 5, HI_DEAMON_MAIN
                .aendi

cyhdt_no6:      .data.w CYHON
                .data.l 10, _CYCHDR

CYHDT_BTM:
CYHCNT:         .equ    (CYHDT_BTM-CYHDT_TOP)/CYHDTLEN
```

（.import _CYCHDR の枠への注記）
周期起動ハンドラの先頭アドレスを外部参照シンボルとして宣言(import)しています(周期起動ハンドラの登録例)。

【各周期起動ハンドラについて定義要】

周期起動ハンドラ情報を定義します。

【書式】

```
LABEL  .data.w  CYC_ACTIVATE
       .data.l  CYC_TIME, CYCHDR_TOP
```

●LABEL：自由に定義可能(無くても問題なし)

●CYC_ACTIVATE(周期起動ハンドラ活性状態の指定)
システム起動時の周期起動ハンドラ活性状態を定義します。
CYHOFF(=0)：起動されない(非活性)
CYHON (=1)：起動される(活性)

●CYC_TIME(周期起動時間間隔)
周期起動ハンドラを起動する周期時間を指定します。

●CYCHDR_TOP(周期起動ハンドラアドレス)
定義するハンドラの開始アドレスを指定します。
NADR(-1)を指定すると、周期起動ハンドラIDは未登録になります。
周期起動ハンドラを使用しない場合、□枠すべての行を削除してください。
デバッギングエクステンションを使用する場合、デバッグデーモンの周期起動ハンドラが登録されます。

（cyhdt_no6 の枠への注記）
周期起動ハンドラ指定番号6に、周期起動ハンドラ活性状態を活性(CYHON)、周期起動時間間隔を10、周期起動ハンドラアドレスをシンボル(_CYCHDR)として、登録しています(周期起動ハンドラの登録例)。

図 6.6　周期起動ハンドラ登録部の登録例

6.2.6 トレース機能の登録

ここではトレース機能を登録するための各種定義を行います。なお、サンプルとして以下のトレース機能を登録することを前提に説明しています。

- 最大トレース情報取得数： 4

図 6.7 にトレース機能登録部を示します。□ **部以外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。**

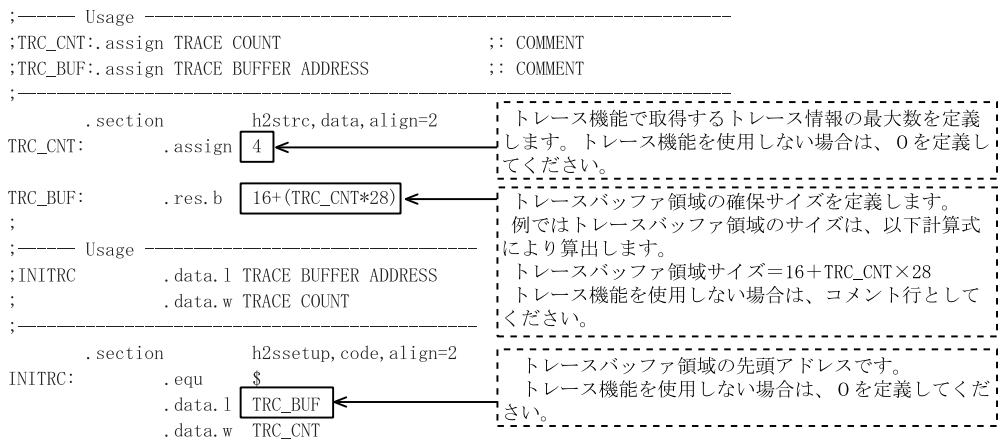


図 6.7 トレース機能登録部

6.2.7　　拡張情報の登録

ここでは、タスク、イベントフラグ、セマフォ、メイルボックス、固定長/可変長メモリプール、周期起動ハンドラの拡張情報を設定します。拡張情報は各資源の ID 毎に設定が可能です。拡張情報には、対象のオブジェクトに関する情報を入れるなどの目的で確保したメモリの領域をパケットとして、その先頭アドレスを設定します。なお、サンプルでは拡張情報の先頭アドレスとして H'0 を登録しています(サンプルの拡張情報の先頭アドレス（H'0）には、拡張情報を設定していません)。

図 6.8 に拡張情報登録部を示します。□ **部以外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。**

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Task Extended Information define section                      %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage -------------------------------------------------------------
;LABEL          .data.l TSK?_EXINF              ;: COMMENT
;-------------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_TSKEXINF:   .equ    $-EXLEN
TSKE_TOP:       .equ    $
tsk1_exinf:     .data.l 00000000

tsk2_exinf:     .data.l 00000000

tsk3_exinf:     .data.l 00000000

tsk4_exinf:     .data.l 00000000

tsk5_exinf:     .data.l 00000000

TSKE_BTM:
TSKECNT:        .equ    (TSKE_BTM-TSKE_TOP)/EXLEN
                                ;:[0...255]     ;: tsk exinf count
;
```

タスク拡張情報を定義します。
【書式】 LABEL　.data.l　TSK?_EXINF
●LABEL：自由に定義可能(無くても問題なし)
●TSK?_EXINF：拡張情報
　アドレスを定義します。
【注意】
タスク拡張情報を定義する場合は「6.2.2 タスクの登録」で定義したタスク数と同じ数の拡張情報を登録してください。
タスク数と一致しない場合はシステム異常終了となります。
拡張情報が不要な場合は、□枠すべての行を削除してください。

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Event Flag Extended Information define section                %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage -------------------------------------------------------------
;LABEL          .data.l FLG?_EXINF              ;: COMMENT
;-------------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_FLGEXINF:   .equ    $-EXLEN
FLGE_TOP:       .equ    $
flg1_exinf:     .data.l 00000000

flg2_exinf:     .data.l 00000000

flg3_exinf:     .data.l 00000000

flg4_exinf:     .data.l 00000000

FLGE_BTM:
FLGECNT:        .equ    (FLGE_BTM-FLGE_TOP)/EXLEN
```

イベントフラグ拡張情報を定義します。
【書式】 LABEL　.data.l　FLG?_EXINF
●LABEL：自由に定義可能(無くても問題なし)
●FLG?_EXINF：拡張情報
　アドレスを定義します。
【注意】
イベントフラグ拡張情報を定義する場合は「6.2.1 定数定義部の登録」で定義した最大イベントフラグ定義数(FLGCNT)と同じ数の拡張情報を登録してください。最大イベントフラグ定義数と一致しない場合はシステム異常終了となります。
拡張情報が不要な場合は、□枠すべての行を削除してください。

図 6.8　拡張情報登録部(1/4)

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%     Semaphore Extended Information define section                   %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage ---------------------------------------------------------------
;LABEL          .data.l SEM?_EXINF              ;: COMMENT
;---------------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_SEMEXINF:   .equ    $-EXLEN
SEME_TOP:       .equ    $
sem1_exinf:     .data.l 00000000

sem2_exinf:     .data.l 00000000

sem3_exinf:     .data.l 00000000

sem4_exinf:     .data.l 00000000

SEME_BTM:
SEMECNT:        .equ    (SEME_BTM-SEME_TOP)/EXLEN
                                ;:[0...255]     ;: sem exinf count
;
```

セマフォ拡張情報を定義します。
【書式】 LABEL .data.l SEM?_EXINF
●LABEL：自由に定義可能(無くても問題なし)
●SEM?_EXINF：拡張情報
アドレスを定義します。
【注意】
セマフォ拡張情報を定義する場合は「6.2.1 定数定義部の登録」で定義した最大セマフォ定義数(SEMCNT)と同じ数の拡張情報を登録してください。
最大セマフォ定義数と一致しない場合はシステム異常終了となります。
拡張情報が不要な場合は、□枠すべての行を削除してください。

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%     Mailbox Extended Information define section                     %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage ---------------------------------------------------------------
;LABEL          .data.l MBX?_EXINF              ;: COMMENT
;---------------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_MBXEXINF:   .equ    $-EXLEN
MBXE_TOP:       .equ    $
mbx1_exinf:     .data.l 00000000

mbx2_exinf:     .data.l 00000000

mbx3_exinf:     .data.l 00000000

mbx4_exinf:     .data.l 00000000

MBXE_BTM:
MBXECNT:        .equ    (MBXE_BTM-MBXE_TOP)/EXLEN
```

メイルボックス拡張情報を定義します。
【書式】 LABEL .data.l MBX?_EXINF
●LABEL：自由に定義可能(無くても問題なし)
●MBX?_EXINF：拡張情報
アドレスを定義します。
【注意】
メイルボックス拡張情報を定義する場合は「6.2.1 定数定義部の登録」で定義した最大メイルボックス定義数(MBXCNT)と同じ数の拡張情報を登録してください。最大メイルボックス定義数と一致しない場合はシステム異常終了となります。
拡張情報が不要な場合は、□枠すべての行を削除してください。

図 6.8 拡張情報登録部(2/4)

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Fixed-size MemoryPool Extended Information define section      %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage ----------------------------------------------------------------
;LABEL          .data.l MPF?_EXINF              ;: COMMENT
;----------------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_MPFEXINF:   .equ    $-EXLEN
MPFE_TOP:       .equ    $
mpf1_exinf:     .data.l 00000000

mpf2_exinf:     .data.l 00000000

mpf3_exinf:     .data.l 00000000

mpf4_exinf:     .data.l 00000000

MPFE_BTM:
MPFECNT:        .equ    (MPFE_BTM-MPFE_TOP)/EXLEN
                                ;:[0...255]      ;: mpf exinf count
;
```

固定長メモリプール拡張情報を定義します。
【書式】 LABEL .data.l MPF?_EXINF
●LABEL：自由に定義可能(無くても問題なし)
●MPF?_EXINF：拡張情報
アドレスを定義します。
【注意】
固定長メモリプール拡張情報を定義する場合は「6.2.3 固定長メモリプールの登録」で定義した固定長メモリプール数と同じ数の拡張情報を登録してください。固定長メモリプール数と一致しない場合はシステム異常終了となります。
拡張情報が不要な場合は、□枠すべての行を削除してください。

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Variable-size MemoryPool Extended Information define section    %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage ----------------------------------------------------------------
;LABEL          .data.l MPL?_EXINF              ;: COMMENT
;----------------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_MPLEXINF:   .equ    $-EXLEN
MPLE_TOP:       .equ    $
mpl1_exinf:     .data.l 00000000

mpl2_exinf:     .data.l 00000000

mpl3_exinf:     .data.l 00000000

mpl4_exinf:     .data.l 00000000

MPLE_BTM:
MPLECNT:        .equ    (MPLE_BTM-MPLE_TOP)/EXLEN
```

可変長メモリプール拡張情報を定義します。
【書式】 LABEL .data.l MPL?_EXINF
●LABEL：自由に定義可能(無くても問題なし)
●MPL?_EXINF：拡張情報
アドレスを定義します。
【注意】
可変長メモリプール拡張情報を定義する場合は「6.2.4 可変長メモリプールの登録」で定義した可変長メモリプール数と同じ数の拡張情報を登録してください。可変長メモリプール数と一致しない場合はシステム異常終了となります。
拡張情報が不要な場合は、□枠すべての行を削除してください。

図 6.8 拡張情報登録部(3/4)

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Cyclic Handler Extended Information define section              %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage -----------------------------------------------------------------
;LABEL          .data.l CYH?_EXINF              ;: COMMENT
;------------------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_CYCEXINF:   .equ    $-EXLEN
CYHE_TOP:       .equ    $
cyh1_exinf:     .data.l 00000000
cyh2_exinf:     .data.l 00000000
cyh3_exinf:     .data.l 00000000
cyh4_exinf:     .data.l 00000000
                .aifdef DX
cyh5_exinf:     .data.l 00000000
                .aendi
CYHE_BTM:
CYHECNT:        .equ    (CYHE_BTM-CYHE_TOP)/EXLE
```

周期起動ハンドラ拡張情報を定義します。
【書式】 LABEL .data.l CYH?_EXINF
●LABEL：自由に定義可能(無くても問題なし)
●CYH?_EXINF：拡張情報
アドレスを定義します。
【注意】
周期起動ハンドラ拡張情報を定義する場合は「6.2.5 周期起動ハンドラの登録」で定義した周期起動ハンドラ数と同じ数の拡張情報を登録してください。周期起動ハンドラ数と一致しない場合はシステム異常終了となります。
拡張情報が不要な場合は、□枠すべての行を削除してください。
デバッギングエクステンションを使用する場合、デバッグデーモンの周期起動ハンドラ拡張情報が登録されます。

図 6.8 **拡張情報登録部**(4/4)

6.3 システム定義部

システム定義部は、本カーネルが使用する外部定義シンボル、定数の定義およびカーネルのシステム作業領域を確保しています。

図 6.9 にシステム定義部を示します。システム定義部ではユーザ定義部に定義した値から、外部定義シンボル、定数の定義およびカーネルのシステム作業領域を自動的に確保します。したがって、**システム定義部の修正はしないでください。修正した場合、システムの正常な動作は保証されません。**

```
        .include        "setup.inc"
;
;***********************************************;
        .end; of 2655asup.src
```

システム定義ファイルをインクルードしています。

図 6.9 **システム定義部**

システム定義部で確保されるシステムの作業領域の詳細については、「付録 A. メモリ容量の算出」を参照してください。

7. 割込みベクタテーブルの作成

7.1 概要

割込みベクタテーブルは、システム動作中に割込みが発生した場合、その割込みの種類に対応した処理に制御が移るよう、各割込みハンドラの先頭アドレスを登録しておくテーブルです。また、割込みハンドラを登録しない空きベクタには未定義割込みハンドラを登録する必要があります。

サンプルとして提供している割込みベクタテーブルのファイルは「sample¥***nnnnz***smp¥***nnnnz***vec.src」です。

提供ファイルの内容を参考にして、ユーザシステムにあわせて作成してください。

7.2 割込みハンドラの登録

ここでは、作成した割込みハンドラを使用するため、割込みベクタテーブルの該当ベクタ番号に、割込みハンドラの先頭アドレスを登録します。

なお、サンプルとして以下の割込みハンドラを登録することを前提にしています。

- 使用 CPU：H8S/2655
- CPU モード：アドバンストモード

表 7.1 に割込みハンドラ登録情報を示します。

表 7.1 割込みハンドラの登録情報

項番	割込みハンドラ	ラベル名	該当ベクタ番号	備考
1	CPU **初期化ルーチン**	_H_2S_CPUINI	0	**パワーオンリセット**
2	CPU **初期化ルーチン**	_H_2S_CPUINI	1	**マニュアルリセット**
3	**タイマ割込みハンドラ**	_H_2S_TIM	32	TPU ch0
4	**未定義割込みハンドラ**	_H_2SINT??	??	??**：項番** 1**～**3 **以外のベクタ番号**

割込み要因の詳細については、当該ハードウェアマニュアルを参照してください。

図 7.1 にサンプルとして提供している H8S/2655 シリーズ用の割込みベクタテーブル『2655avec.src』（アドバンストモード用）の例を示します。

```
;***************************************************************************
;***                                                                     ***
;***    HI2000/3 Version (uITRON V3.0)                                   ***
;***    HI2000/3 vector table                                            ***
;***                                                                     ***
;***    Copyright (c)  Hitachi, Ltd. 1998.                               ***
;***    Licensed Material of Hitachi, Ltd.                               ***
;***                                                                     ***
;***************************************************************************
        .program        _2655avec
        .heading        "### 2655avec.src : for H8S/2655 ###"
        .section        h2svectr,code,locate=0
;
        .import         _H_2SINT00,_H_2SINT01,_H_2SINT02,_H_2SINT03,_H_2SINT04
        .import         _H_2SINT05,_H_2SINT06,_H_2SINT07,_H_2SINT08,_H_2SINT09
        .import         _H_2SINT10,_H_2SINT11,_H_2SINT12,_H_2SINT13,_H_2SINT14
        .import         _H_2SINT15,_H_2SINT16,_H_2SINT17,_H_2SINT18,_H_2SINT19
        .import         _H_2SINT20,_H_2SINT21,_H_2SINT22,_H_2SINT23,_H_2SINT24
        .import         _H_2SINT25,_H_2SINT26,_H_2SINT27,_H_2SINT28,_H_2SINT29
        .import         _H_2SINT30,_H_2SINT31,_H_2SINT32,_H_2SINT33,_H_2SINT34
        .import         _H_2SINT35,_H_2SINT36,_H_2SINT37,_H_2SINT38,_H_2SINT39
        .import         _H_2SINT40,_H_2SINT41,_H_2SINT42,_H_2SINT43,_H_2SINT44
        .import         _H_2SINT45,_H_2SINT46,_H_2SINT47,_H_2SINT48,_H_2SINT49
        .import         _H_2SINT50,_H_2SINT51,_H_2SINT52,_H_2SINT53,_H_2SINT54
        .import         _H_2SINT55,_H_2SINT56,_H_2SINT57,_H_2SINT58,_H_2SINT59
        .import         _H_2SINT60,_H_2SINT61,_H_2SINT62,_H_2SINT63,_H_2SINT64
        .import         _H_2SINT65,_H_2SINT66,_H_2SINT67,_H_2SINT68,_H_2SINT69
        .import         _H_2SINT70,_H_2SINT71,_H_2SINT72,_H_2SINT73,_H_2SINT74
        .import         _H_2SINT75,_H_2SINT76,_H_2SINT77,_H_2SINT78,_H_2SINT79
        .import         _H_2SINT80,_H_2SINT81,_H_2SINT82,_H_2SINT83,_H_2SINT84
        .import         _H_2SINT85,_H_2SINT86,_H_2SINT87,_H_2SINT88,_H_2SINT89
        .import         _H_2SINT90,_H_2SINT91
;
        .import         _H_2S_CPUINI    ;: in 'cpuini'
        .import         _H_2S_TIM       ;: in 'h2suser'
;
```

未定義割込みハンドラの先頭ルーチンを外部参照シンボルとして宣言します。（表7.1の項番4）

定義する割込みハンドラの先頭ルーチンを外部参照シンボルとして宣言します。（表7.1の項番1～3）

図 7.1　割込みベクタテーブル『2655avec.src』の例(1/4)

```
;***********************************************************************
;*specifications ;                                                     *
;*name      = h2svectr : H8S/2655 vector table for HI2000/3          ;   *
;*function  = 1. h8 interrupt handler address define for hi8         ;   *
;*          = 2. h8 exception handler address define for hi8         ;   *
;*          = 3. hi8 system standard support module                  ;   *
;*          =    (1) reset     : "_H_2S_CPUINI"  for power on        ;   *
;*          = 4. hi8 system standard support module                  ;   *
;*          =    (1) TPU0 tgi0a: "_H_2S_TIM"   for system timer      ;   *
;*date      = 99/02/22                                               ;   *
;*author    = Hitachi, Ltd.                                          ;   *
;*attribute = public                                                 ;   *
;*class     = unit                                                   ;   *
;*linkage   = h8 vector table top address = h'0000 for HI2000/3      ;   *
;*input     = none                                                   ;   *
;*output    = none                                                   ;   *
;*parameter = er7 : stack pointer                                    ;   *
;***********  CPU interrupt mode = 3  ******************************;   *
;*                              |           |                        ;   *
;*    er7(stack pointer) --> +0 +-----------+                        ;   *
;*                              |    EXR    |                        ;   *
;*                           +1 +-----------+                        ;   *
;*                              | reserved  |                        ;   *
;*                           +2 +-----------+                        ;   *
;*                              |    CCR    |                        ;   *
;*                           +3 +-----------+                        ;   *
;*                              |           |                        ;   *
;*                              +           +                        ;   *
;*                              |     PC    |                        ;   *
;*                              +           +                        ;   *
;*                              |           |                        ;   *
;*                           +6 +-----------+                        ;   *
;*end of specifications ;                                            *
;***********************************************************************
;
        .radix  d                       ;:xxxxx -> d'xxxxx
;
;------.data.l < address >              ;: H8S/2655 vector no.  contents
        .data.l _H_2S_CPUINI            ;_H_2SINT00   ;: vector no.00 <reset>
        .data.l _H_2S_CPUINI            ;_H_2SINT01   ;: vector no.01 <reset>
        .data.l _H_2SINT02              ;: vector no.02 [reserve]
        .data.l _H_2SINT03              ;: vector no.03 [reserve]
        .data.l _H_2SINT04              ;: vector no.04 [reserve]
        .data.l _H_2SINT05              ;: vector no.05 [reserve]
```

CPU初期化ルーチン登録。
必ず登録してください。
(表7.1の項番1,2)

図 7.1　割込みベクタテーブル『2655avec.src』の例(2/4)

```
.data.l _H_2SINT06              ;: vector no.06 [reserve]
.data.l _H_2SINT07              ;: vector no.07 <NMI          >
.data.l _H_2SINT08              ;: vector no.08 <TRAPA  #0    >
.data.l _H_2SINT09              ;: vector no.09 <TRAPA  #1    >
.data.l _H_2SINT10              ;: vector no.10 <TRAPA  #2    >
.data.l _H_2SINT11              ;: vector no.11 <TRAPA  #3    >
.data.l _H_2SINT12              ;: vector no.12 [reserve]
.data.l _H_2SINT13              ;: vector no.13 [reserve]
.data.l _H_2SINT14              ;: vector no.14 [reserve]
.data.l _H_2SINT15              ;: vector no.15 [reserve]
.data.l _H_2SINT16              ;: vector no.16 <IRQ0         >
.data.l _H_2SINT17              ;: vector no.17 <IRQ1         >
.data.l _H_2SINT18              ;: vector no.18 <IRQ2         >
.data.l _H_2SINT19              ;: vector no.19 <IRQ3         >
.data.l _H_2SINT20              ;: vector no.20 <IRQ4         >
.data.l _H_2SINT21              ;: vector no.21 <IRQ5         >
.data.l _H_2SINT22              ;: vector no.22 <IRQ6         >
.data.l _H_2SINT23              ;: vector no.23 <IRQ7         >
.data.l _H_2SINT24              ;: vector no.24 <SWDTEND      >
.data.l _H_2SINT25              ;: vector no.25 <WOVI         >
.data.l _H_2SINT26              ;: vector no.26 <CMI          >
.data.l _H_2SINT27              ;: vector no.27 [reserve]
.data.l _H_2SINT28              ;: vector no.28 <ADI          >
.data.l _H_2SINT29              ;: vector no.29 [reserve]
.data.l _H_2SINT30              ;: vector no.30 [reserve]
.data.l _H_2SINT31              ;: vector no.31 [reserve]
.data.l _H_2S_TIM ;_H_2SINT32   ;: vector no.32 <TGI0A  tpu0 >
.data.l _H_2SINT33              ;: vector no.33 <TGI0B  tpu0 >
.data.l _H_2SINT34              ;: vector no.34 <TGI0C  tpu0 >
.data.l _H_2SINT35              ;: vector no.35 <TGI0D  tpu0 >
.data.l _H_2SINT36              ;: vector no.36 <TCI0V  tpu0 >
.data.l _H_2SINT37              ;: vector no.37 [reserve]
.data.l _H_2SINT38              ;: vector no.38 [reserve]
.data.l _H_2SINT39              ;: vector no.39 [reserve]
.data.l _H_2SINT40              ;: vector no.40 <TGI1A  tpu1 >
.data.l _H_2SINT41              ;: vector no.41 <TGI1B  tpu1 >
.data.l _H_2SINT42              ;: vector no.42 <TCI1V  tpu1 >
.data.l _H_2SINT43              ;: vector no.43 <TCI1U  tpu1 >
.data.l _H_2SINT44              ;: vector no.44 <TGI2A  tpu2 >
.data.l _H_2SINT45              ;: vector no.45 <TGI2B  tpu2 >
.data.l _H_2SINT46              ;: vector no.46 <TCI2V  tpu2 >
.data.l _H_2SINT47              ;: vector no.47 <TCI2U  tpu2 >
.data.l _H_2SINT48              ;: vector no.48 <TGI3A  tpu3 >
.data.l _H_2SINT49              ;: vector no.49 <TGI3B  tpu3 >
.data.l _H_2SINT50              ;: vector no.50 <TGI3C  tpu3 >
.data.l _H_2SINT51              ;: vector no.51 <TGI3D  tpu3 >
.data.l _H_2SINT52              ;: vector no.52 <TCI3V  tpu3 >
.data.l _H_2SINT53              ;: vector no.53 [reserve]
.data.l _H_2SINT54              ;: vector no.54 [reserve]
.data.l _H_2SINT55              ;: vector no.55 [reserve]
.data.l _H_2SINT56              ;: vector no.56 <TGI4A  tpu4 >
.data.l _H_2SINT57              ;: vector no.57 <TGI4B  tpu4 >
```

タイマ割込みハンドラの登録。
wai_flg, t付きSVC(例：twai_sem), set_tim, get_timシステムコール使用時、必ず登録してください。
(表7.1の項番3)

図 7.1　割込みベクタテーブル『2655avec.src』の例(3/4)

```
        .data.l _H_2SINT58              ;: vector no.58 <TCI4V  tpu4 >
        .data.l _H_2SINT59              ;: vector no.59 <TCI4U  tpu4 >
        .data.l _H_2SINT60              ;: vector no.60 <TGI5A  tpu5 >
        .data.l _H_2SINT61              ;: vector no.61 <TGI5B  tpu5 >
        .data.l _H_2SINT62              ;: vector no.62 <TCI5V  tpu5 >
        .data.l _H_2SINT63              ;: vector no.63 <TCI5U  tpu5 >
        .data.l _H_2SINT64              ;: vector no.64 <CMIA0       >
        .data.l _H_2SINT65              ;: vector no.65 <CMIB0       >
        .data.l _H_2SINT66              ;: vector no.66 <OVI0        >
        .data.l _H_2SINT67              ;: vector no.67 [reserve]
        .data.l _H_2SINT68              ;: vector no.68 <CMIA1       >
        .data.l _H_2SINT69              ;: vector no.69 <CMIB1       >
        .data.l _H_2SINT70              ;: vector no.70 <OVI1        >
        .data.l _H_2SINT71              ;: vector no.71 [reserve]
        .data.l _H_2SINT72              ;: vector no.72 <DEND0A dmac >
        .data.l _H_2SINT73              ;: vector no.73 <DEND0B dmac >
        .data.l _H_2SINT74              ;: vector no.74 <DEND1A dmac >
        .data.l _H_2SINT75              ;: vector no.75 <DEND1B dmac >
        .data.l _H_2SINT76              ;: vector no.76 [reserve]
        .data.l _H_2SINT77              ;: vector no.77 [reserve]
        .data.l _H_2SINT78              ;: vector no.78 [reserve]
        .data.l _H_2SINT79              ;: vector no.79 [reserve]
        .data.l _H_2SINT80              ;: vector no.80 <ERI0   sci0 >
        .data.l _H_2SINT81              ;: vector no.81 <RXI0   sci0 >
        .data.l _H_2SINT82              ;: vector no.82 <TXI0   sci0 >
        .data.l _H_2SINT83              ;: vector no.83 <TEI0   sci0 >
        .data.l _H_2SINT84              ;: vector no.84 <ERI1   sci1 >
        .data.l _H_2SINT85              ;: vector no.85 <RXI1   sci1 >
        .data.l _H_2SINT86              ;: vector no.86 <TXI1   sci1 >
        .data.l _H_2SINT87              ;: vector no.87 <TEI1   sci1 >
        .data.l _H_2SINT88              ;: vector no.88 <ERI2   sci2 >
        .data.l _H_2SINT89              ;: vector no.89 <RXI2   sci2 >
        .data.l _H_2SINT90              ;: vector no.90 <TXI2   sci2 >
        .data.l _H_2SINT91              ;: vector no.91 <TEI2   sci2 >
;
;**************************************;
        .end; of 2655avec.src
```

図 7.1　割込みベクタテーブル『2655avec.src』の例(4/4)

8. ロードモジュールの生成

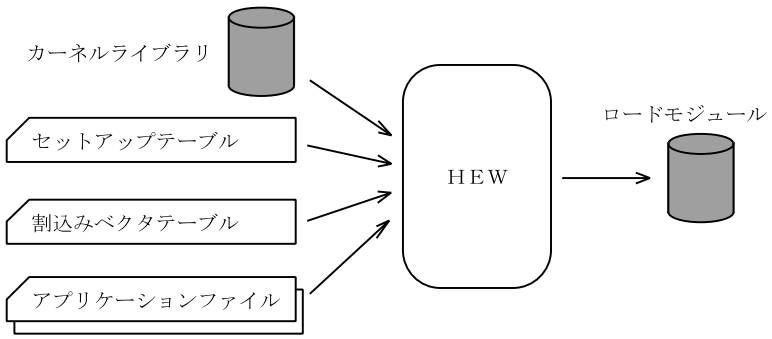
8.1 ロードモジュール生成方法

ロードモジュールの生成は、HEW を用いて行います。HEW については、HEW のマニュアルやオンラインヘルプなどを用いて、あらかじめ操作方法を習得しておいてください。

本カーネルシステムを構成する以下の 4 つのプログラムを結合して実行可能なロードモジュールを生成します。

- カーネルライブラリ
- アプリケーション
- セットアップテーブル
- 割込みベクタテーブル

ロードモジュールの生成を図 8.1 に示します。



【注】 セットアップテーブル、割込みベクタテーブルが外部参照しているアプリケーションは必須です。

図 8.1 ロードモジュールの生成

本カーネルは、CPU に応じて H8S/2600CPU、H8S/2000CPU の 2 種類のカーネルを提供しています。さらに、カーネルはアドバンストモード、ノーマルモードの 2 種類の MCU 動作モードに分かれています。

カーネルおよび提供しているアプリケーションについては、使用する環境に応じて選択してください。

8.2 HEW ワークスペースとプロジェクト

ロードモジュールの生成は HEW のビルド機能を用いて行います。HEW のビルド機能では、以下の手順でロードモジュールを生成します。

- ロードモジュールの生成に必要なファイルをプロジェクトに登録する。
- Cコンパイラ、アセンブラ、モジュール間最適化ツールのオプションを設定する。
- ビルドを実行する。

本製品では、サンプルワークスペースファイル product.hws を提供しています。product.hws をダブルクリックすると、HEW が起動してワークスペース product が開きます。

ワークスペース product には、各種デバイスに対応したサンプルのプロジェクトがあらかじめ登録されています。表 8.1 に示すように、CPU と動作モードに対応した 4 種類のサンプルプロジェクトがあります。ユーザの使用する環境(CPU,動作モード)に合ったプロジェクトを選択し、以降の説明を参考に設定を変更してください。

サンプルプロジェクトを選択するには、図 8.2 のように HEW のワークスペースウィンドウでプロジェクトを選び、ポップアップメニューで [Set as Current Project] を選択してください。

なお、使用しない環境用のプロジェクトは削除しても構いません。

H8S/2655,H8S/2245 以外のデバイスを使用する場合、プロジェクトの選択後、既に登録されているシステム構築用ファイルを使用する CPU 用のファイルに変更する必要があります。

表 8.1 提供サンプルプロジェクト

項番	プロジェクト名	ｺﾝﾌｨｸﾞﾚｰｼｮﾝ*1	内容
1	hi26a	hi26a	H8S/2600CPU･**アドバンスト用ロードモジュール** (H8S/2655 **用に登録済み**)
2	hi26n	hi26n	H8S/2600CPU･**ノーマル用ロードモジュール** (H8S/2655 **用に登録済み**)
3	hi20a	hi20a	H8S/2000CPU･**アドバンスト用ロードモジュール** (H8S/2245 **用に登録済み**)
4	hi20n	hi20n	H8S/2000CPU･**ノーマル用ロードモジュール** (H8S/2245 **用に登録済み**)

【注】*1 **コンフィグレーション内にロードモジュールが生成されるように設定してあります。**

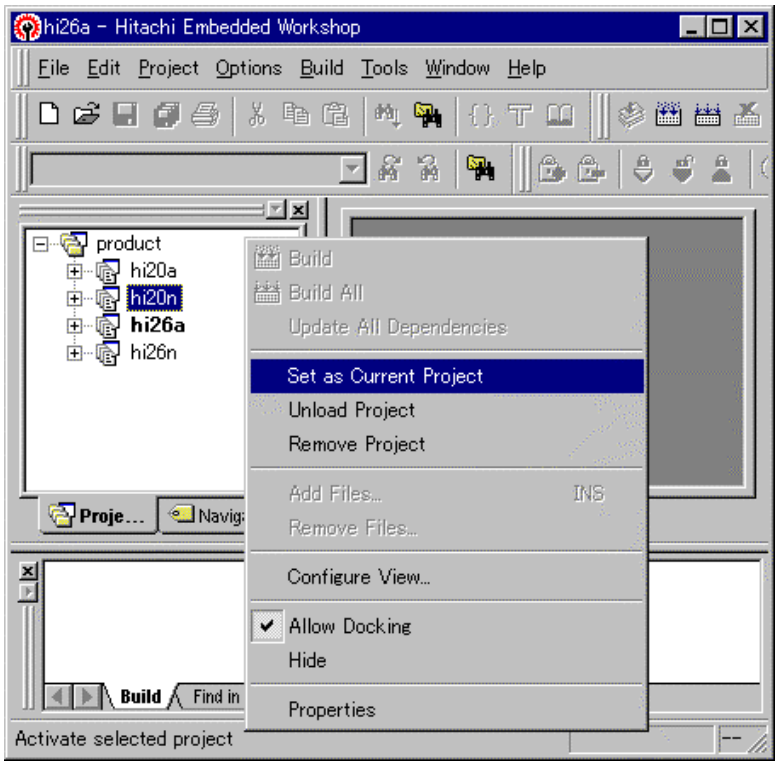


図 8.2　プロジェクトの選択

提供のサンプルプロジェクトを選択して、ビルドを実行すると、プロジェクトに登録してあるファイルのコンパイル、アセンブル、モジュール間最適化ツールを一連の流れで実行し、ロードモジュールを生成します。

8.3 ロードモジュールの生成

8.3.1 プロジェクトへの登録

プロジェクトに登録するファイルを表 8.2 に示します。出荷時のサンプルプロジェクトには、H8S/2655 または H8S/2245 用のシステム構築用ファイルが登録されています。

H8S/2655,H8S/2245 以外の場合は、システム構築用ファイルを登録し直してください。

表 8.2 プロジェクトに登録するファイル

項番	ファイル名*1	内容	備考
1	**アプリケーションファイル**	**タスク、割込みハンドラ**	
2	sample¥***nnnnz***smp¥***nnnnz***sup.src	**セットアップテーブル**	**必須**
3	sample¥***nnnnz***smp¥***nnnnz***use.src	**タイマドライバ**	
		システム異常終了ルーチン	**必須**
		システムアイドリングルーチン	**必須**
		システム初期化ハンドラ	
4	sample¥***nnnnz***smp¥***nnnnz***vec.src	**割込みベクタテーブル**	**必須**
5	sample¥***nnnnz***smp¥***nnnnz***ili.src	**未定義割込みハンドラ**	
6	sample¥***nnnnz***smp¥***nnnnz***cpu.src	CPU **初期化ルーチン**	**必須**
7	sample¥task.c	DX **チュートリアルタスク**	

【注】*1 ***nnnn*** **斜体の *nnnn* はデバイスに対応しています。**

z **斜体の *z* は動作モードを示します**(a:ｱﾄﾞﾊﾞﾝｽﾄ、n:ﾉｰﾏﾙ)。

以下の手順でプロジェクトへの登録を行います。

- HEW を起動し、サンプルワークスペースを開きます。
- 使用する環境のプロジェクトを選択します。
- ファイルメニューの[Add Files]でアプリケーションファイルを登録します。
- H8S/2655,H8S/2245 以外の場合は、システム構築用ファイルを登録します。
- 既に登録されているシステム構築用ファイルのオプションを参照し、追加したファイルのオプションを設定してください。オプションの設定後、使用しないシステム構築用ファイルを削除してください。

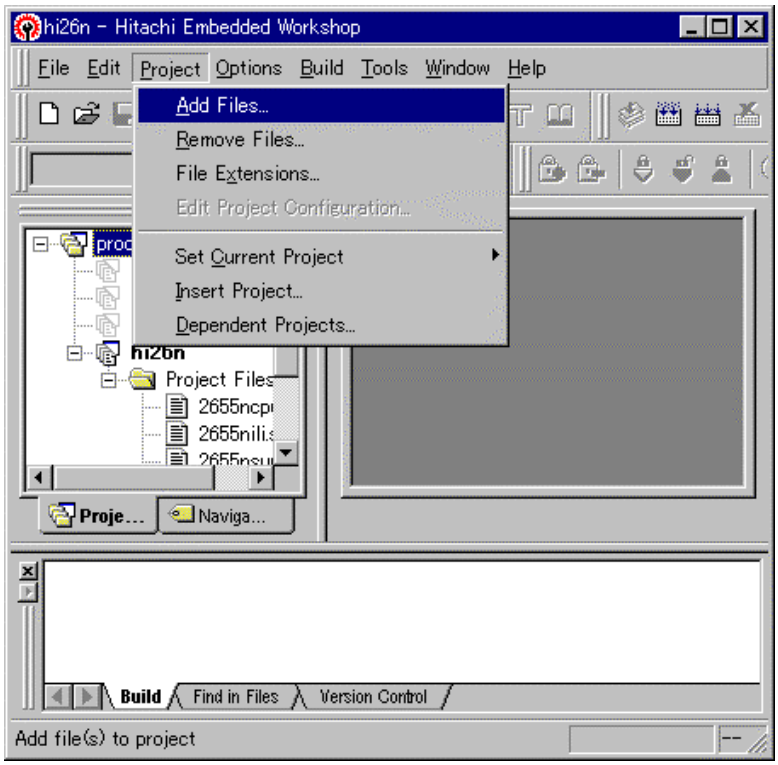


図 8.3　プロジェクトへの登録

8.3.2 コンパイラ、アセンブラオプション

コンパイラ、アセンブラオプションに関しては、「付録 B. コンパイラ、アセンブラオプション」および関連マニュアルを参照してください。

システム構築用ファイルのコンパイラ、アセンブラオプションは、表 8.3、図 8.4～図 8.9 を参考に設定してください。

表 8.3 コンパイラ、アセンブラオプション

項番	ファイル名*1	オプション内容
1	**システム構築用ファイル共通**	CPU **タブ** **使用する** CPU **に合わせて指定してください** Object **タブ** Output file directory: $(CONFIGDIR) Debug information: **デバッグ情報出力の指定** List **タブ** **リスト出力なしの指定**
2	sample¥*nnnnz*smp¥*nnnnz*sup.src	Source **タブ** Include file directories: $(PROJDIR)¥sample **を指定** Defines: DX=Action(Debugging Extension **を組込む場合**)
3	sample¥*nnnnz*smp¥*nnnnz*use.src	―
4	sample¥*nnnnz*smp¥*nnnnz*vec.src	―
5	sample¥*nnnnz*smp¥*nnnnz*ili.src	―
6	sample¥*nnnnz*smp¥*nnnnz*cpu.src	Source **タブ** Defines: DX=Action (Debugging Extension **を組込む場合**)
7	sample¥task.c	Source **タブ** Include file directories: $(PROJDIR)¥sample **を指定** Object **タブ** Section: P=Ptask, B=Btask **を指定**

【注】*1 ***nnnn*** **斜体の *nnnn* はデバイスに対応しています。**
z **斜体の *z* は動作モードを示します**(a:ｱﾄﾞﾊﾞﾝｽﾄ、n:ﾉｰﾏﾙ)。

共通に指定する CPU タブの例を図 8.4 に示します

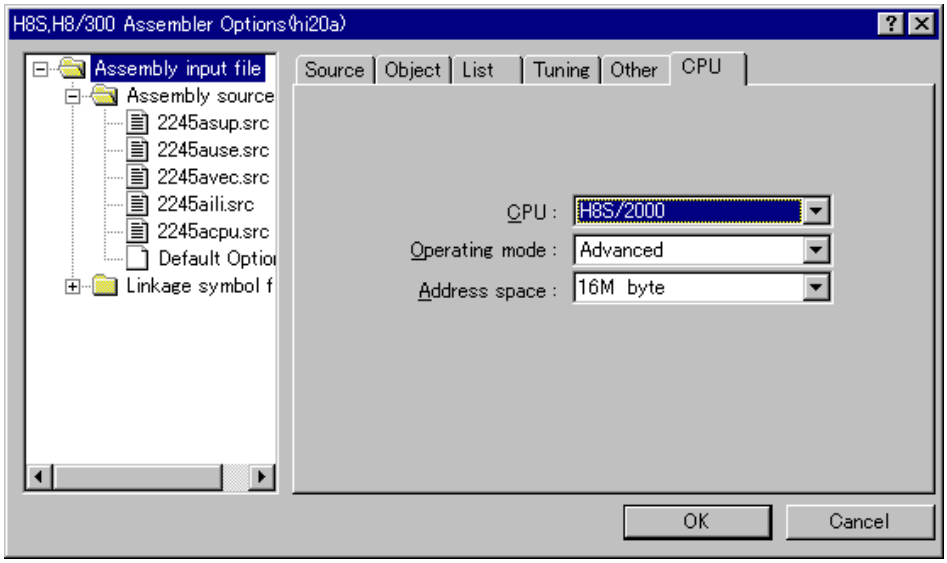


図 8.4　H8S,H8/300 Assembler Options **の** CPU **タブウィンドウ**

共通に指定する Object タブの例を図 8.5 に示します。

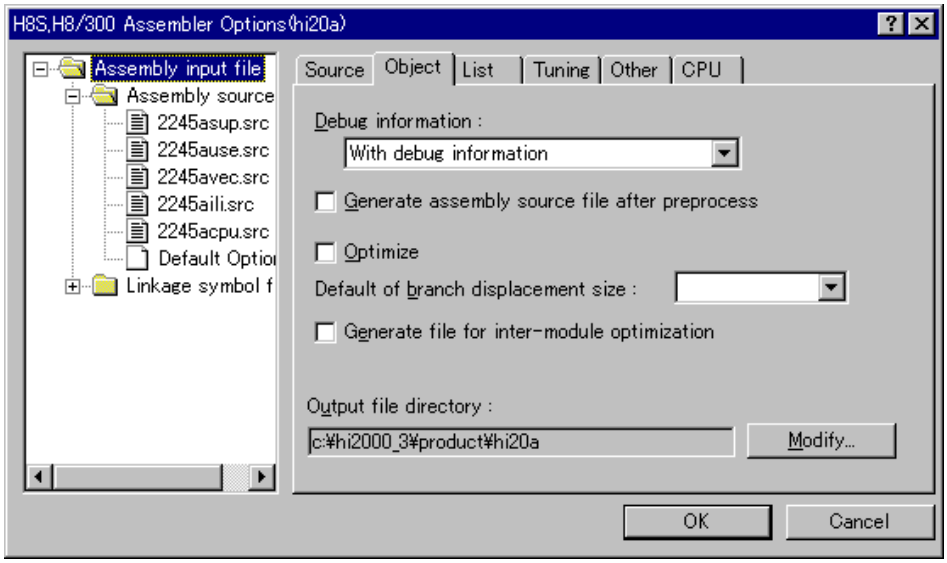


図 8.5　H8S,H8/300 Assembler Options **の** Object **タブウィンドウ**

共通に指定する List タブの例を図 8.6 に示します。

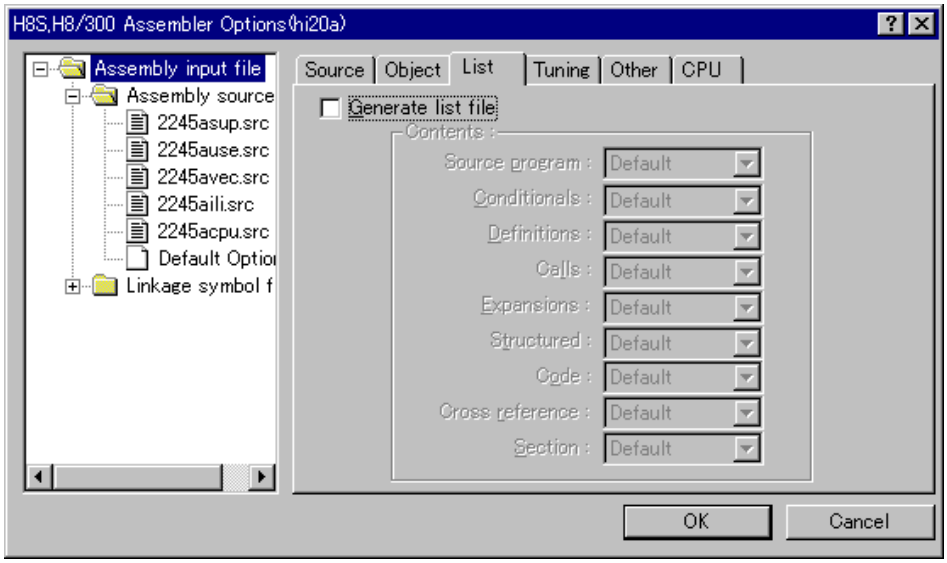


図 8.6　H8S,H8/300 Assembler Options **の** List **タブウィンドウ**

インクルードファイルの指定を行う Source タブの例を図 8.7 に示します。
“Show entries for:”ドロップダウンリストから“Include file directories”を選択してください。

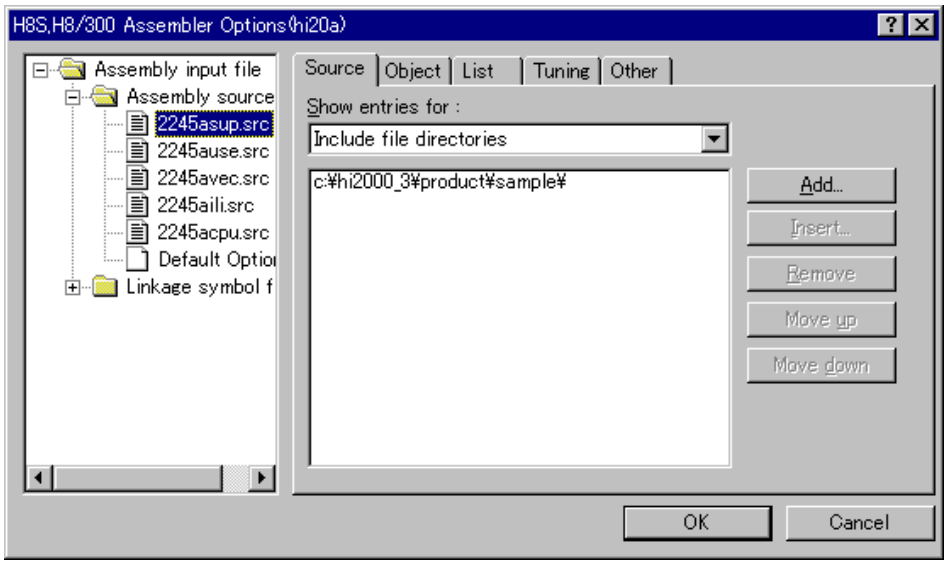


図 8.7　H8S,H8/300 Assembler Options **の** Source **タブウィンドウ**(Include file directories)

デバッギングエクステンション(DX)の指定を行う Source タブの例を図 8.8 に示します。
“Show entries for:”ドロップダウンリストから“Defines”を選択してください。

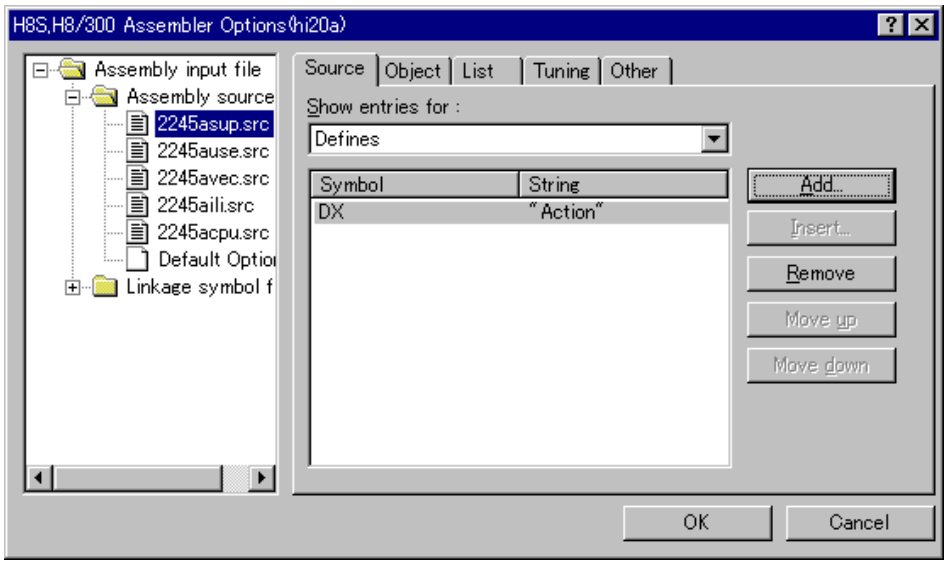


図 8.8　H8S,H8/300 Assembler Options **の** Source **タブウィンドウ**(Defines)

セクションの指定を行う Object タブの例を図 8.9 に示します。
“Section”ドロップダウンリストからセクションを選択してください。

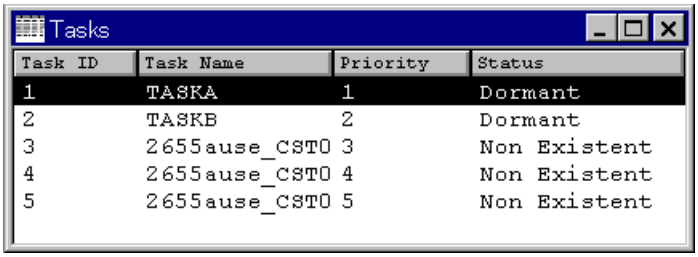


図 8.9　H8S,H8/300 C Compiler Options **の** Object **タブウィンドウ**

8.3.3 モジュール間最適化ツールの設定

(1) Inter-module Optimizer Options の Input タブ

提供のプロジェクトファイルは、パラメータチェック機能あり、共有スタック機能ありのカーネルライブラリ、C 言語インタフェースライブラリが指定されています。

ユーザの使用する環境(CPU,動作モード)に応じて、ライブラリファイルを指定しなおしてください。

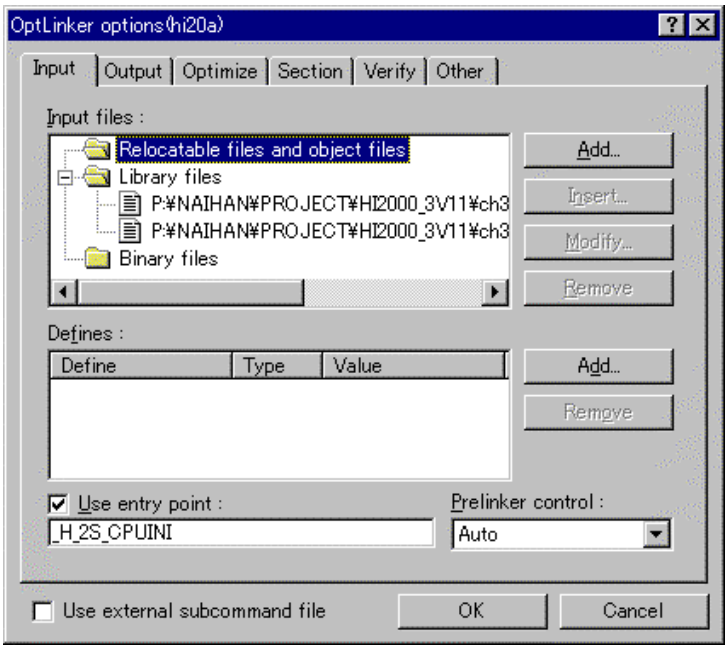


図 8.10 Inter-module Optimizer Options **の** Input **タブウィンドウ**

ライブラリファイルは、表 8.4 に示すカーネルライブラリ、C 言語インタフェースライブラリから選択します。

また、アプリケーションのライブラリや、H8S シリーズ C コンパイラが提供する標準ライブラリなどを使用する場合、この Input タブで指定してください。

表 8.4 提供ライブラリファイル一覧

項番	ライブラリ名			ファイル名	内容
1	ｶｰﾈﾙ ﾗｲﾌﾞﾗﾘ	H8S/2600CPU	ｱﾄﾞﾊﾞﾝｽﾄ	hilib¥26aknlps.lib	ﾊﾟﾗﾒｰﾀﾁｪｯｸ機能あり､共有ｽﾀｯｸ機能あり
				hilib¥26aknlpn.lib	ﾊﾟﾗﾒｰﾀﾁｪｯｸ機能あり､共有ｽﾀｯｸ機能なし
				hilib¥26aknlns.lib	ﾊﾟﾗﾒｰﾀﾁｪｯｸ機能なし､共有ｽﾀｯｸ機能あり
				hilib¥26aknlnn.lib	ﾊﾟﾗﾒｰﾀﾁｪｯｸ機能なし､共有ｽﾀｯｸ機能なし
			ﾉｰﾏﾙ	hilib¥26nknlps.lib	ﾊﾟﾗﾒｰﾀﾁｪｯｸ機能あり､共有ｽﾀｯｸ機能あり
				hilib¥26nknlpn.lib	ﾊﾟﾗﾒｰﾀﾁｪｯｸ機能あり､共有ｽﾀｯｸ機能なし
				hilib¥26nknlns.lib	ﾊﾟﾗﾒｰﾀﾁｪｯｸ機能なし､共有ｽﾀｯｸ機能あり
				hilib¥26nknlnn.lib	ﾊﾟﾗﾒｰﾀﾁｪｯｸ機能なし､共有ｽﾀｯｸ機能なし
		H8S/2000CPU	ｱﾄﾞﾊﾞﾝｽﾄ	hilib¥20aknlps.lib	ﾊﾟﾗﾒｰﾀﾁｪｯｸ機能あり､共有ｽﾀｯｸ機能あり
				hilib¥20aknlpn.lib	ﾊﾟﾗﾒｰﾀﾁｪｯｸ機能あり､共有ｽﾀｯｸ機能なし
				hilib¥20aknlns.lib	ﾊﾟﾗﾒｰﾀﾁｪｯｸ機能なし､共有ｽﾀｯｸ機能あり
				hilib¥20aknlnn.lib	ﾊﾟﾗﾒｰﾀﾁｪｯｸ機能なし､共有ｽﾀｯｸ機能なし
			ﾉｰﾏﾙ	hilib¥20nknlps.lib	ﾊﾟﾗﾒｰﾀﾁｪｯｸ機能あり､共有ｽﾀｯｸ機能あり
				hilib¥20nknlpn.lib	ﾊﾟﾗﾒｰﾀﾁｪｯｸ機能あり､共有ｽﾀｯｸ機能なし
				hilib¥20nknlns.lib	ﾊﾟﾗﾒｰﾀﾁｪｯｸ機能なし､共有ｽﾀｯｸ機能あり
				hilib¥20nknlnn.lib	ﾊﾟﾗﾒｰﾀﾁｪｯｸ機能なし､共有ｽﾀｯｸ機能なし
2	ｼｽﾃﾑｺｰﾙ C 言語 ｲﾝﾀﾌｪｰｽ ﾗｲﾌﾞﾗﾘ	H8S/2600CPU	ｱﾄﾞﾊﾞﾝｽﾄ	hilib¥26acif.lib	
			ﾉｰﾏﾙ	hilib¥26ncif.lib	
		H8S/2000CPU	ｱﾄﾞﾊﾞﾝｽﾄ	hilib¥20acif.lib	
			ﾉｰﾏﾙ	hilib¥20ncif.lib	

(2) Inter-module Optimizer Options **の** Output **タブ**

出力するロードモジュールのフォーマット、デバッグ情報出力、出力ディレクトリを設定します。

提供プロジェクトは、ロードモジュールがコンフィグレーションの中に出力されるように設定してあります。

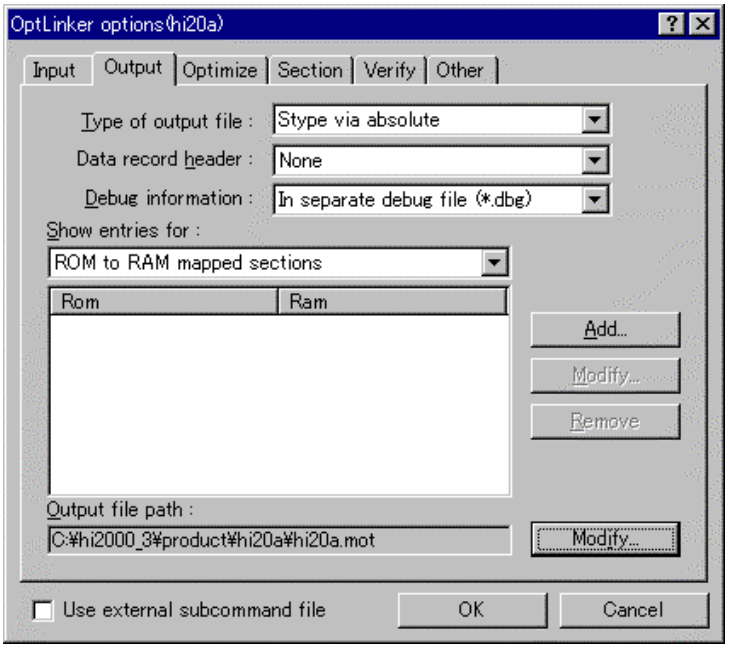


図 8.11 Inter-module Optimizer Options **の** Output **タブウィンドウ**

(3) Inter-module Optimizer Options の Section タブ

各セクションの配置アドレスを指定します。提供プロジェクトには、H8S/2655 または H8S/2245 用にセクションの配置アドレスが指定されています。

ターゲットハードウェアにあわせて、入力ファイルに含まれる各セクションを指定し、配置アドレスを指定しなおしてください。

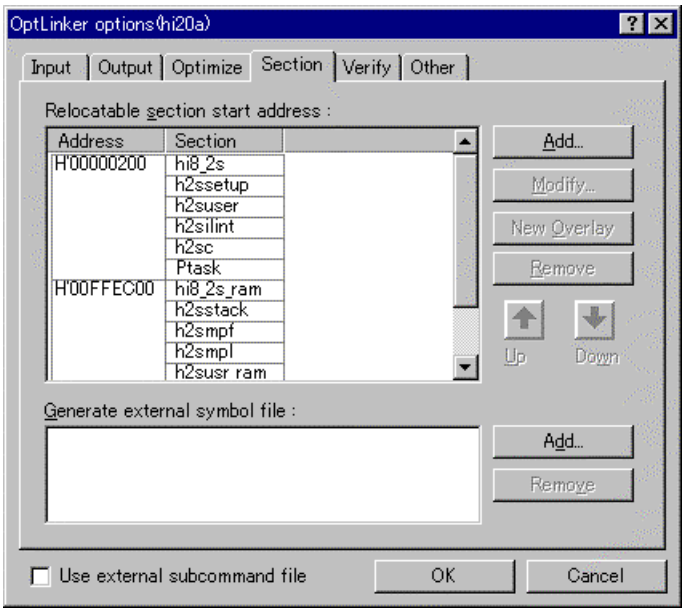


図 8.12 Inter-module Optimizer Options **の** Section **タブウィンドウ**

提供プロジェクトに含まれるセクションについて、表 8.5 に示します。

表 8.5　セクション一覧

項番	ﾒﾓﾘ種別	ｾｸｼｮﾝ名	内容
1	ROM	h2svectr	割込みベクタテーブル
2		hi8_2s	カーネル
3		h2ssetup	セットアップテーブル
4		h2suser	システム初期化ハンドラ、タイマ初期化ルーチン、 タイマ割込みハンドラ、システム異常終了ルーチン、 CPU 初期化ルーチン、システムアイドリングルーチン
5		h2silint	未定義割込みハンドラ
6		h2sc	Ｃ言語インタフェースライブラリ
7		Ptask	デバッギングエクステンション(DX)用チュートリアルタスク
8	RAM	hi8_2s_ram	カーネルのシステム作業領域
9		h2sstack	タスクスタック領域
10		h2smpf	固定長メモリプール領域
11		h2smpl	可変長メモリプール領域
12		h2susr_ram	CPU 初期化ルーチンのスタック領域
13		h2strc	トレースバッファ領域
14		Btask	デバッギングエクステンション(DX)用チュートリアルタスクのメッセージ領域

入力ファイル中に含まれるすべてのセクションを漏れなく指定してください。指定されていないセクションが入力ファイル中に存在する場合、モジュール間最適化ツールは最後に指定したセクションに続いてこれらのセクションを自動的に配置するため、ユーザが意図しない配置となる可能性があり、誤動作の原因となります。なお、Other タブで[Check for Unlinked Sections]を指定すれば、このような場合ウォーニングを表示させることができます。

逆に、指定したセクションが入力ファイル中に存在しない場合、モジュール間最適化ツールはウォーニングを表示しますが、結合は正常に行われます。

メモリ配置について注意事項を以下に示します。

- 割込みベクタテーブル(h2svectr)は、必ず 0 番地に配置してください。サンプル提供の割込みベクタテーブルを使用する場合は、自動的に 0 番地に配置されるため、Section タブの指定は必要ありません。
- カーネル(hi8_2s)は偶数番地から配置してください。また、アドバンストモードの場合はこれらのセクションを H’xx0000～H’xxFFFF 番地の間に配置してください。配置する番地の上位アドレス“ xx ”は同一にしてください。
- 本カーネルのシステム作業領域(hi8_2s_ram)は偶数番地から配置してください。また、アドバンストモードの場合にこれらのセクションを H’xx0000～H’xxFFFF 番地の間に配置してください。配置する番地の上位アドレス“ xx ”は同一にしてください。
- セットアップテーブル(h2ssetup)は偶数番地から配置してください。また、アドバンストモードの場合にこれらのセクションを H’xx0000～H’xxFFFF 番地の間に配置してください。配置する番地の上位アドレス“ xx ”は同一にしてください。

8.3.4 ビルドの実行

プロジェクトへのアプリケーションファイルの登録、コンパイル、アセンブル、モジュール間最適化のオプション設定後、ビルドを実行することでロードモジュールを生成します。

ビルドを実行するには、図 8.13 のように HEW の Build メニューから[Build]または[Build All]を選択してください。

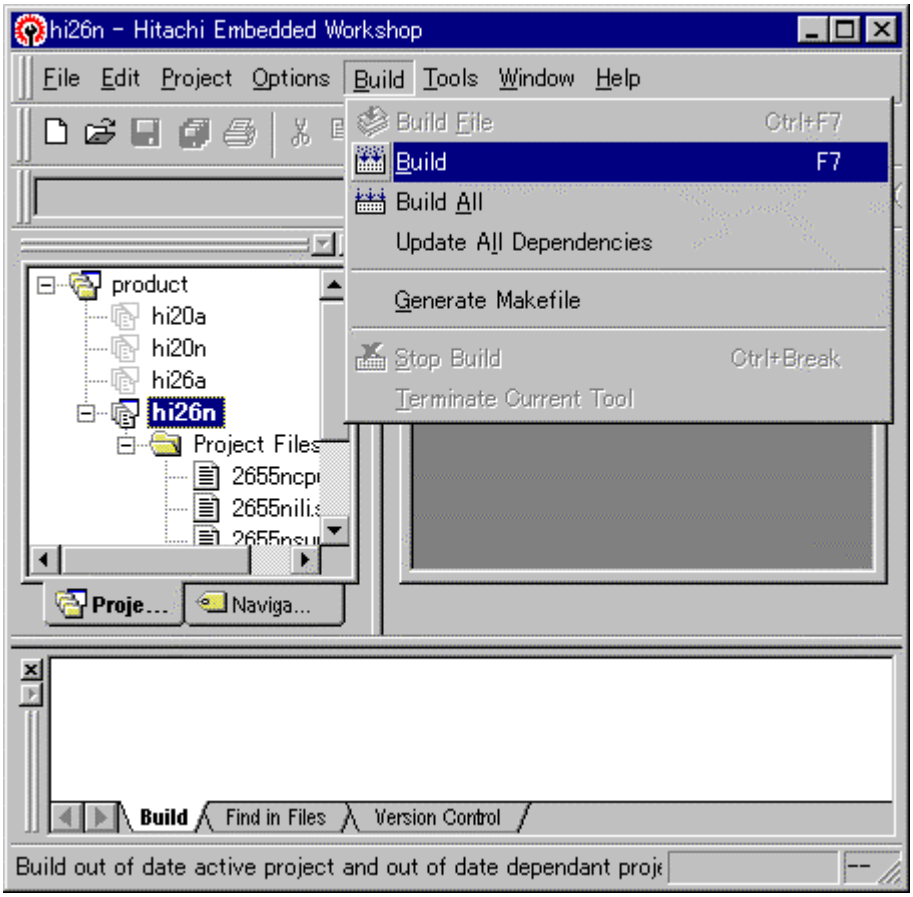


図 8.13 ビルドの実行

8.4 C 言語インタフェースライブラリのプロジェクト

C 言語インタフェースのソースファイルを変更し、C 言語インタフェースライブラリファイルを再生成する場合は、C 言語インタフェースワークスペースファイル cif.hws をダブルクリックして開き、ワークスペース cif に登録されているプロジェクトの中から表 8.6 を参考に使用する環境のプロジェクトを選択して、ビルドを実行してください。

表 8.6　C 言語インタフェースのプロジェクト

項番	ﾌﾟﾛｼﾞｪｸﾄ名	内容
1	26acif	H8S/2600CPU･ｱﾄﾞﾊﾞﾝｽﾄ用
2	26ncif	H8S/2600CPU･ﾉｰﾏﾙ用
3	20acif	H8S/2000CPU･ｱﾄﾞﾊﾞﾝｽﾄ用
4	20ncif	H8S/2000CPU･ﾉｰﾏﾙ用

付録 A．メモリ容量の算出

A.1　メモリ容量の算出

使用するメモリ(RAM)容量は以下の算出表から求めてください。

システム初期化ハンドラ、タイマ初期化ルーチンのスタックサイズを求める場合は、割込みハンドラ用スタックサイズの算出表を使用してください(カーネル割込みマスクレベルと同一の割込みレベルを持つ割込みハンドラとして算出してください)。

A.1.1 OS 作業領域の算出

表 A.1 に OS 作業領域の算出表を示します。
本算出表を使用して、OS の使用する作業領域のサイズを求めてください。

表 A.1 OS 作業領域の算出表

内 訳	計算式	容量 (バイト)	備 考
システム管理テーブル (_HI_SYSMT)	10 ＋ 4×優先度定義(MAXPRI)		必須です
タスク管理ブロック (_HI_TCB)	18×タスク定義数(TSKCNT)		必須です
タスク管理ブロック 2 (_HI_TCB2)	8×タスク定義数(TSKCNT)		タイムアウト機能付システムコールを使用する場合必要
イベントフラグ管理ブロック(_HI_FLGCB)	6×イベントフラグ定義数(FLGCNT)		イベントフラグを使用する場合必要
セマフォ管理ブロック (_HI_SEMCB)	6×セマフォ定義数(SEMCNT)		セマフォを使用する場合必要
メイルボックス管理ブロック(_HI_MBXCB)	8×メイルボックス定義数(MBXCNT)		メイルボックスを使用する場合必要
固定長メモリプール管理ブロック(_HI_MPFCB)	6×固定長メモリプール定義数(MPFCNT)		固定長メモリプールを使用する場合必要
可変長メモリプール管理ブロック(_HI_MPLCB)	20×可変長メモリプール定義数(MPLCNT)		可変長メモリプールを使用する場合必要
周期起動ハンドラ管理ブロック(_HI_CYHCB)	20×周期起動ハンドラ定義数(CYHCNT)		周期起動ハンドラを使用する場合必要
タイマ管理ブロック (_HI_TIMCB, _HI_TIMCB2, _HI_TIMCB3)	10 [*1] ＋ 4 [*2] ＋ 14 [*3]		[*1]:タイマドライバを使用する場合必要 [*2]:タイムアウト機能付システムコールを使用する場合必要 [*3]:周期起動ハンドラを使用する場合必要
トレースバッファ管理ブロック(TBACB)	8		システムコールトレース機能を使用する場合必要
合 計			

【注】
- セットアップテーブルのタイムアウト機能定義(ラベル名「TTMOUT」)を NOTUSE にした場合、タイムアウト機能が使用する領域(TCB2、TIMCB2 領域)は確保されません
- セットアップテーブルのタイマスタックサイズ(ラベル名「TIMSTKSIZ」)を 0 にした場合、タイマ管理ブロック(TIMCB、TIMCB2、TIMCB3 領域)およびタイマ関連管理ブロック(TCB2、CYHCB)は確保されません
- セットアップテーブルのシステムコールトレース用スタックサイズ(ラベル名「TRCSTKSIZ」)を 0 にした場合、トレースバッファ管理ブロックは確保されません

A.1.2　OS 用スタック領域サイズの算出

表 A.2 に OS 用スタック領域サイズ(OSSTKSIZ)の算出表を示します。
本算出表を使用して、OS のスタック領域のサイズを求めてください。
OS のスタック領域サイズは、セットアップテーブルに定義してください。

表 A.2　OS 用スタック領域サイズの算出表

内　訳	計算式	容量 (バイト)	備　考
OS が使用するスタックサイズ	18(アドバンストモード) 14(ノーマルモード)	18 or 14	常に必要です
割込み用スタックサイズ	10 × LOWINTNST *1 ＋ 6 × UPPINTNST *2		(割込み制御モード 2、3 使用時)
	8 × LOWINTNST *1 ＋ 4 × UPPINTNST *2		(割込み制御モード 0、1 使用時)
未定義割込み用スタックサイズ *3	8		(割込み制御モード 2、3 使用時)
	6		(割込み制御モード 0、1 使用時)
合　計			

【注】 *1　カーネル割込みマスクレベル以下の割込みのネスト数
*2　カーネル割込みマスクレベルより高い割込み(NMI を含む)のネスト数
*3　未定義割込みが発生する場合必要

A.1.3　タイマ割込み用スタック領域サイズの算出

表 A.3 にタイマ割込み用スタック領域サイズ(TIMSTKSIZ)の算出表を示します。
本算出表を使用して、タイマ割込み用スタック領域のサイズを求めてください。
タイマ割込み用スタック領域サイズは、セットアップテーブルに定義してください。

表 A.3　タイマ割込み用スタック領域サイズの算出表

内　訳	計算式	容量 (バイト)	備　考
タイマ割込みハンドラが独自に使用するスタックサイズ	40(アドバンストモード) 38(ノーマルモード)	40 or 38	常に必要です
割込み用スタックサイズ	10 × LOWINTNST [*1] ＋ 6 × UPPINTNST [*2]		(割込み制御モード 2、3 使用時)
	8 × LOWINTNST [*1] ＋ 4 × UPPINTNST [*2]		(割込み制御モード 0、1 使用時)
未定義割込み用スタックサイズ [*3]	8		(割込み制御モード 2、3 使用時)
	6		(割込み制御モード 0、1 使用時)
周期起動ハンドラが独自に使用するスタックサイズ [*4]	ユーザ使用サイズ		表 A.5 から算出したサイズを加算
合　計			

【注】 [*1] カーネル割込みマスクレベル以下でかつタイマ割込みレベルより高い割込みのネスト数

[*2] カーネル割込みマスクレベルより高い割込み(NMI を含む)のネスト数

[*3] 未定義割込みが発生する場合必要

[*4] 周期起動ハンドラを複数使用する場合は、ハンドラ単位にスタックサイズを算出し、その中から、最大のスタックサイズを加算してください。
周期起動ハンドラを C 言語で記述した場合、周期起動ハンドラが独自に使用するスタックサイズは、コンパイルリストに出力される関数のフレームサイズから求めてください。
また、周期起動ハンドラからシステムコールを発行する場合は、「表 A.5 割込みハンドラ用スタック領域サイズの算出」を使用して、サイズを算出してください。

A.1.4　タスクスタック領域サイズの算出

表 A.4 にタスクスタック領域サイズの算出表を示します。
本算出表を使用して、タスク ID 毎のスタック領域のサイズを求めてください。
タスクスタック領域は、セットアップテーブルに定義し、タスク ID 毎にスタック領域を確保してください。タスクをC言語で記述した場合、タスクが独自に使用するスタックサイズはコンパイルリストに出力される関数のフレームサイズから求めてください。
タスク用スタック領域全体のサイズは、タスク ID 毎に求めたスタックサイズの総和になります。なお、共有スタック機能を使用する場合は、同じスタック領域を使用するタスクの中で、最も大きなサイズを指定してください。

> 共有スタック機能を使用する場合は、すべてのスタック領域の最終アドレスから上位アドレス側に 8 バイト分の共有スタック機能用の領域を確保してください。

表 A.4　タスクスタック領域サイズの算出表

内　訳	計算式	容量 (バイト)	備　考
タスクが独自に使用するスタックサイズ	ユーザ使用サイズ		
OS が使用するスタックサイズ	50(H8S/2600CPU 使用時) 42(H8S/2000CPU 使用時)	50 or 42	常に必要です
割込み用スタックサイズ	10 × LOWINTNST *1 ＋ 6 × UPPINTNST *2		(割込み制御モード 2、3 使用時)
	8 × LOWINTNST *1 ＋ 4 × UPPINTNST *2		(割込み制御モード 0、1 使用時)
システムコールトレース用スタックサイズ	6		システムコールトレース機能を使用する場合必要
未定義割込み用スタックサイズ *3	8		(割込み制御モード 2、3 使用時)
	6		(割込み制御モード 0、1 使用時)
C 言語インタフェースが使用するスタックサイズ	22(アドバンストモード) 14(ノーマルモード)		C 言語でタスクを記述した場合必要
共有タスクスタック機能が使用するスタックサイズ	8		共有スタック機能使用する場合必要
合　計			

【注】 *1　カーネル割込みマスクレベル以下の割込みのネスト数
*2　カーネル割込みマスクレベルより高い割込み(NMI を含む)のネスト数
*3　未定義割込みが発生する場合必要

A.1.5　割込みハンドラ用スタック領域サイズの算出

表 A.5 に割込みハンドラ用スタック領域サイズの算出表を示します。

本算出表を使用して、個々の割込みハンドラのスタック領域のサイズを求めてください。割込みハンドラのスタック領域は、割込みレベル毎に共有できます。同じ割込みレベルの割込みハンドラで最大に使用するときのスタックサイズを確保してください。

割込みハンドラ用スタック領域は、ハンドラ毎にスタック領域を確保してください。

割込みハンドラをＣ言語で記述した場合、割込みハンドラが独自に使用するスタックサイズはコンパイルリストに出力される関数のフレームサイズから求めてください。

表 A.5　割込みハンドラ用スタック領域サイズの算出表

内　訳	計算式	容量 (バイト)	備　考
割込みハンドラが独自に使用するスタックサイズ	ユーザ使用サイズ		
OS が使用するスタックサイズ	42(アドバンストモード) 38(ノーマルモード)	42 or 38	常に必要です
割込み用スタックサイズ	10 × LOWINTNST *1 ＋ 6 × UPPINTNST *2		(割込み制御モード 2、3 使用時)
	8 × LOWINTNST *1 ＋ 4 × UPPINTNST *2		(割込み制御モード 0、1 使用時)
システムコールトレース用スタックサイズ	6		システムコールトレース機能を使用する場合必要
未定義割込み用スタックサイズ *3	8		(割込み制御モード 2、3 使用時)
	6		(割込み制御モード 0、1 使用時)
C 言語インタフェースが使用するスタックサイズ	22(アドバンストモード) 14(ノーマルモード)		Ｃ言語で割込みハンドラを記述している場合必要
合　計			

【注】 *1　カーネル割込みマスクレベル以下で、かつ自割込みレベルより高い割込みのネスト数

*2　カーネル割込みマスクレベルより高い割込み(NMI を含む)のネスト数

*3　未定義割込みが発生する場合必要

A.1.6　固定長メモリプール領域サイズの算出

表 A.6 に、固定長メモリプール領域サイズの算出表を示します。
本算出表を使用して、固定長メモリプール ID 毎のメモリプール領域サイズを求めてください。固定長メモリプール領域全体のサイズは、メモリプール ID 毎に求めた領域サイズの総和になります。
セットアップテーブルにメモリプール ID 毎の固定長メモリブロック数(MB?_CNT)と固定長メモリブロックサイズ(MB?_LEN)を定義し、メモリプール領域を確保してください。

表 A.6　固定長メモリプール領域サイズの算出表

内　訳	計算式	容量 (バイト)	備　考
固定長メモリプール領域サイズ	固定長メモリブロック数 [*1] × (固定長メモリブロックサイズ[*2] ＋ 4)		メモリブロック毎に管理領域(4 バイト)が必要
合　計			

【注】 [*1]　固定長メモリブロック数のラベル(MB?_CNT)
[*2]　固定長メモリブロックサイズのラベル(MB?_LEN)

A.1.7　可変長メモリプール領域サイズの算出

表 A.7 に、可変長メモリプール領域サイズの算出表を示します。
本算出表を使用して、可変長メモリプール ID 毎のメモリプール領域サイズ(MPL?_SIZ)を求めてください。可変長メモリプール領域全体のサイズは、メモリプール ID 毎に求めた領域サイズの総和になります。
セットアップテーブルにメモリプール ID 毎の可変長メモリプール領域サイズ(MPL?_SIZ)を定義し、メモリプール領域を確保してください。

表 A.7　可変長メモリプール領域サイズの算出表

内　訳	計算式	容量 (バイト)	備　考
可変長メモリプール領域サイズ(MPL?_SIZ)	可変長メモリプールサイズ＋ (16 × n [*1])		メモリブロック獲得毎に管理領域(16 バイト)が必要
合　計			

【注】 [*1]　n：可変長メモリブロックを獲得する最大数

A.1.8　トレース機能用スタック領域サイズの算出

表 A.8 に、トレース機能用スタック領域サイズ(TRCSTKSIZ)の算出表を示します。
本スタック領域は、トレース機能を使用する場合にのみ必要です。
本算出表を使用して、トレース機能が使用するスタック領域のサイズを求めてください。
トレース機能用スタック領域サイズは、セットアップテーブルに定義してください。

表 A.8　トレース機能用スタック領域サイズの算出表

内　訳	計算式	容量 (バイト)	備　考
OS が使用するスタックサイズ	26	26	常に必要です
割込み用スタックサイズ	6 × UPPINTNST *1		(割込み制御モード 2、3 使用時)
	4 × UPPINTNST *1		(割込み制御モード 0、1 使用時)
未定義割込み用スタックサイズ *2	8		(割込み制御モード 2、3 使用時)
	6		(割込み制御モード 0、1 使用時)
合　計			

【注】*1　カーネル割込みマスクレベルより高い割込み(NMI を含む)のネスト数
*2　未定義割込みが発生する場合必要

A.1.9　トレースバッファ領域サイズの算出

表 A.9 に、トレースバッファ領域サイズ(TRC_BUF)の算出表を示します。
本領域は、トレース機能を使用する場合にのみ必要です。
本算出表を使用して、トレースバッファ領域サイズを求めてください。
本算出表は、ノーマルモード、アドバンストモードの両方で使用できます。
トレースバッファ領域サイズは、セットアップテーブルに定義し、トレースバッファ領域を確保してください。

表 A.9　トレースバッファ領域サイズの算出表

内　訳	計算式	容量 (バイト)	備　考
トレースバッファ管理領域	16	16	
トレースエントリ情報領域	28 × トレース情報取得数(TRCCNT)		
合　計			

A.1.10　システム作業領域の算出

表 A.10 にシステム作業領域の算出表を示します。
本算出表を使用して、システムが使用する RAM の使用量を求めてください。

表 A.10　作業領域の算出表

内　訳	計算式	容量 (バイト)	備　考
OS 作業領域	−		表 A.1 を参照
OS 用スタック領域	−		表 A.2 を参照
タイマ割込み用スタック領域	−		表 A.3 を参照
タスクスタック領域(全体) *1	−		表 A.4 を参照
割込みハンドラ用スタック領域 *2	−		表 A.5 を参照
固定長メモリプール領域(全体)	−		表 A.6 を参照
可変長メモリプール領域(全体)	−		表 A.7 を参照
トレース機能用スタック領域	−		表 A.8 を参照
トレースバッファ領域	−		表 A.9 を参照
NMI 割込みハンドラ用スタック領域	−		
システム初期化ハンドラ用スタック領域	−		
CPU 初期化ルーチン用スタック領域 *3	−		
タイマ初期設定ルーチン用スタック領域	−		
その他 (　　)	−		
その他 (　　)	−		
その他 (　　)	−		
その他 (　　)	−		
合　計			

【注】 *1　共有スタック機能を使用する場合、全タスクスタック領域に共有スタック用管理領域が必要になります

*2　割込みハンドラのスタック領域は、割込みレベル毎に共有できます。同じ割込みレベルの割込みハンドラで最大に使用するスタックサイズを確保してください

*3　CPU 初期化ルーチンは、カーネル起動前に実行されるため、スタック領域として NMI 割込みハンドラ用スタック領域以外の任意の RAM 領域と共有することができます

付録 B．コンパイラ、アセンブラオプション

B.1　コンパイラオプション

システムの構築で使用するコンパイラオプションについて説明します。各コマンドオプションに関する詳細は、『H8S，H8/300 シリーズ C/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。

(1)　cpu コマンドオプション

CPU 種別の指定です。

使用する CPU に応じた指定をしてください。**正しく指定しないでコンパイルしたプログラム、および C　オプションを混在し作成したプログラムをカーネル上で実行させた場合、システムの正常な動作は保証されません。**

(2)　include コマンドオプション

インクルードファイルの指定です。

カーネルでは標準ヘッダ hi2000.h を提供しています。hi2000.h は sample ディレクトリにあるので必要に応じてインクルードしてください。

(3)　debug コマンドオプション

オブジェクトにデバッグ情報を付加する指定です。

当社製のデバッグ環境を使用してデバッグする場合に指定してください。

(4)　list コマンドオプション

コンパイルリストファイルを生成するための指定です。

リストファイルには、スタックフレームサイズやセクションサイズなど、重要な情報が出力されます。スタックサイズの算出時や結合時には、これらの情報が有効となります。

(5)　objectfile コマンドオプション

出力するオブジェクトファイルを指定します。

B.2　アセンブラオプション

システムの構築で使用するアセンブラオプションについて説明します。各コマンドオプションに関する詳細は、『H8S，H8/300 シリーズ クロスアセンブラ ユーザーズマニュアル』を参照してください。

(1)　cpu コマンドオプション

CPU 種別の指定です。

使用する CPU に応じた指定をしてください。**正しく指定しないでコンパイルしたプログラム、および C　オプションを混在し作成したプログラムをカーネル上で実行させた場合、システムの正常な動作は保証されません。**

(2)　include コマンドオプション

インクルードファイルの指定です。

カーネルでは標準ヘッダ hi2000.inc を提供しています。hi2000.inc は sample ディレクトリにあるので必要に応じてインクルードしてください。

(3)　debug コマンドオプション

オブジェクトにデバッグ情報を付加する指定です。

当社製のデバッグ環境を使用してデバッグする場合に指定してください。

(4)　list コマンドオプション

アセンブルリストファイルを生成するための指定です。

リストファイルには、セクションサイズなど、重要な情報が出力されます。結合時には、これらの情報が有効となります。

(5)　objectfile コマンドオプション

出力するオブジェクトファイルを指定します。

付録 C．デバイスドライバ

C.1　タイマドライバ

カーネルは、H8S シリーズ内蔵の TPU(Timer Pulse Unit)、FRT(Free Running Timer)を使用したタイマドライバをサンプル提供しています。以下にサンプル提供しているタイマドライバを例に説明します。

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。

カーネルの時間管理機能を使用するには、タイマドライバを作成してシステムに組み込まなければなりません。

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

図 C.1 にタイマドライバの処理概要を示します。

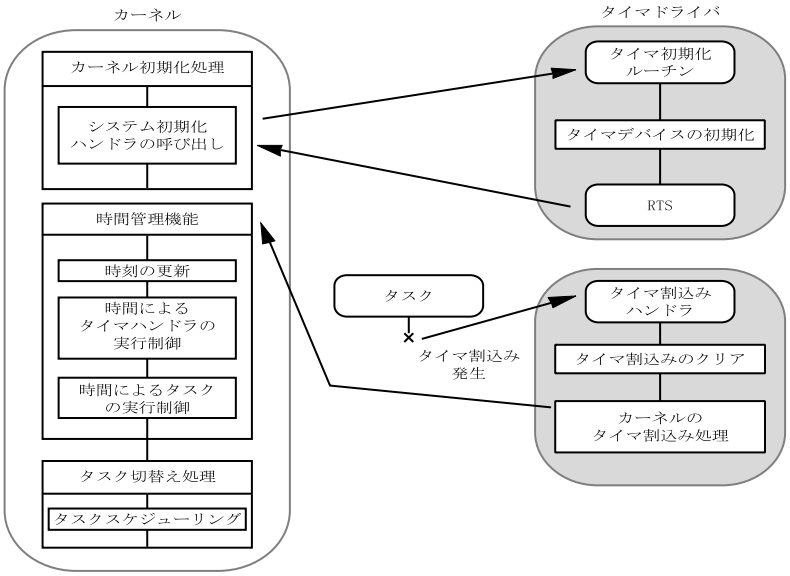


図 C.1 タイマドライバの処理概要

C.1.1　タイマ初期化ルーチン

タイマ初期化ルーチンは、使用するハードウェアタイマの初期化を行います。
表 C.1 にタイマ初期化ルーチンの処理条件を示します。

表 C.1 タイマ初期設定ルーチンの処理条件

項番	項目	内容
1	割込みマスク	割込みマスク状態で起動されます。
2	使用できるレジスタ	レジスタの保証は、C 言語プログラム(関数)のレジスタ保証規則に合わせてください。
3	スタックポインタ	カーネルに制御を戻すときは起動時と同じ値にしてください。
4	使用できるシステムコール	システムコールは発行できません。
5	使用できるスタック領域	スタックを使用する場合、タイマ初期設定ルーチンで使用するスタック領域をシステム構築時に確保し、タイマ初期設定ルーチン起動時にスタックを切り替えてください。
6	終了方法	RTS 命令により処理を終了します。 終了時は、スタックを起動時と同じ状態にしてください。

C.1.2　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って起動されます。

タイマ割込みハンドラでは、タイマ割込みリセット処理でタイマ割込みのクリアを行い、カーネルのタイマ割込み処理へジャンプすることにより、カーネルに対して時間管理処理要求を行います。

タイマ割込みリセット処理からタスク独立部用システムコールを発行し、タスクの実行を制御することも可能です。

表 C.2 にタイマ割込みリセット処理の処理条件を示します。

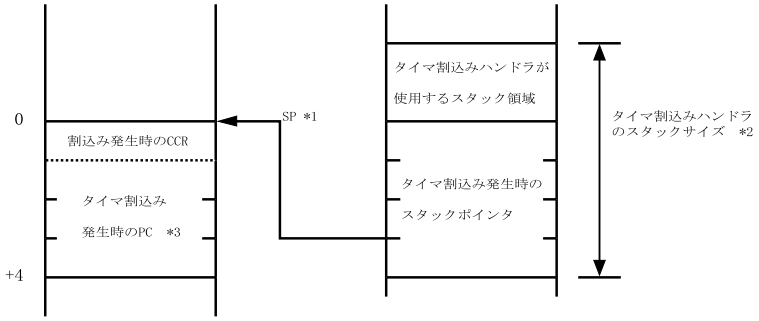
表 C.2 タイマ割込みリセット処理の処理条件

項番	項目	内容
1	割込みマスク	割込みマスク状態で起動されます。
2	使用できるレジスタ	ER0～ER6 が使用できます。
3	スタックポインタ	時間管理処理要求を行わない場合 割込み発生元に制御も戻すときは起動時と同じ値にしてください。 時間管理処理要求を行う場合 タイマ割込みハンドラ用のスタックポインタに切り換えた状態にしてください。 図 C.2 のタイマ割込みリセット処理終了時のスタックの状態を参照してください。
4	使用できるシステムコール	タスク独立部から発行可能なシステムコール
5	使用できるスタック領域	システム構築時に確保し、起動時にスタックを切り替えてください。
6	終了方法	カーネルのタイマ割込み処理にジャンプして処理を終了します。 jmp @_H_timsys (_H_timsys はカーネルのタイマ割込み処理の先頭シンボルです) 終了時は、スタックを起動時と同じ状態にしてください。

タイマ割込みリセット処理終了時は、タイマ割込み発生時のスタックを設定したタイマ割込みハンドラ用スタック領域のアドレスをスタックとしてカーネルのタイマ割込み処理へジャンプしてください。

図 C.2 にタイマ割込みリセット処理終了時のスタック状態を示します。

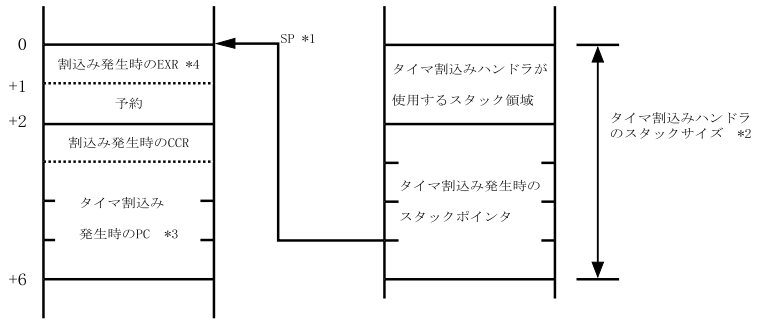
(1)　**割込み制御モード 0、1 の場合**



*1：カーネルのタイマ割込み処理へジャンプするときのスタックポインタ値
*2：タイマ割込み用のスタックサイズは「付録A　メモリ容量の算出」を参照してください
*3：ノーマルモードの場合、下位16ビットが有効になります

図 C.2 タイマ割込みリセット処理終了時のスタック状態(1/2)

(2)　**割込み制御モード 2、3 の場合**



*1：カーネルのタイマ割込み処理へジャンプするときのスタックポインタ値
*2：タイマ割込み用のスタックサイズは「付録A　メモリ容量の算出」を参照してください
*3：ノーマルモードの場合、下位16ビットが有効になります
*4：割込み制御モード0, 1の場合、exrレジスタはスタックに積まれません

図 C.2 タイマ割込みリセット処理終了時のスタック状態(2/2)

C.1.3　タイマドライバの定義情報

時間管理機能を使用する場合、タイマドライバが必要です。
カーネルは、ハードウェアからの一定周期の割込みを利用して、時間管理を行います。
時間管理は、ハードウェアタイマとして H8S シリーズ内蔵の 16 ビットタイマを使用しています。
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ています。
サンプルとして提供する H8S シリーズ用タイマドライバのファイルは
「sample¥***nnnnz***smp¥***nnnnz***use.src」です。

(1)　タイマ初期設定ルーチン(ラベル名『_HIPRG_TIMINI』)

システムクロックとして使用するハードウェアタイマの初期化を行うプログラムです。タイマ割込みの割込みレベルは、セットアップテーブルに設定したカーネル割込みマスクレベルより高いレベルにしないでください。

(2)　タイマ割込みハンドラ(ラベル名『_H_2S_TIM』)

ハードウェアタイマの割込み発生後、ハードウェアタイマの割込みをクリア(タイマ割込みリセット処理)して、OS のタイマ割込み処理にジャンプするプログラムです。
ハードウェアタイマが、割込みをクリアする必要があるときに作成します。

ハードウェアタイマの詳しい説明は、当該ハードウェアマニュアルを参照してください。

(3)　タイマ周期の変更

ここでは、H8S/2655 内蔵 TPU を対象として説明します。

提供のタイマドライバは、TPU のジェネラル・レジスタ 0A(TGR0A)をアウトプットコンペアレジスタとして使用し、タイマ周期を 10msec に設定しています。

このタイマ周期を容易に変更できるように、表 C.3 にタイマドライバの assign 定義しています。

表 C.3 タイマドライバの assign 定義

項番	ラベル名	内　容	設定値
1	TGRA_DATA	**タイマジェネラル・レジスタ** 0A(TGR0A)**設定データ** **タイマプリスケーラ** **タイマのカウンタクロックを指定します。** 4 **種類の内部クロック(φ、φ/4、φ/16、φ/64)を指定します。** **タイマ周期は、**TGR0A **設定データとタイマプリスケーラの値で決定します。** **タイマ周期は、以下の式で算出します。** **タイマ周期：** x (sec) **タイマプリスケーラ：** n TGR0A **設定データ＝** x × n − 1 (**例：提供の値**) CPU **クロック(φ)**＝20M (Hz) **タイマプリスケーラ＝φ/16** **タイマ周期**＝10m (sec) TGR0A **設定データ**＝0.01(20,000,000/16)-1 ＝12,500-1 ＝H' 30d3 CPU **クロック(φ)が** 20MHz **の場合、タイマ周期は以下の範囲まで設定できます。** (**タイマプリスケーラ：φ**)＝50.0μ(sec)〜3.27m(sec) (**タイマプリスケーラ：φ/4**)＝200.0μ(sec)〜13.1m(sec) (**タイマプリスケーラ：φ/16**)＝800.0μ(sec)〜52.4m(sec) (**タイマプリスケーラ：φ/64**)＝3200.0μ(sec)〜209.7m(sec)	H' 30D3
2	TCR_DATA	**タイマコントロールレジスタ** 0(TCR0)**の設定データ** **カウンタクリア** TGR0A **のコンペア・マッチでタイマカウンタ**(TCNT0)**をクリアするように指定します。** **タイマプリスケーラ** **タイマプリスケーラを内部クロック：φ/16 でカウントするように指定します。**	H' 22
3	IPRF_TPU0	TPU **チャネル** 0 **の割込みハンドラの割込みレベル**	H' 05

(4) タイマドライバの登録・未登録

タイマドライバを登録する場合、タイマドライバはタイマ初期設定ルーチンとタイマ割込みハンドラの登録が必要です。

タイマドライバを使用しない場合は、割込みベクタテーブル、セットアップテーブルから削除、および提供タイマドライバの削除が必要です。

(a) タイマ初期設定ルーチンの登録

タイマ初期設定ルーチンを登録する場合、タイマ初期設定ルーチンのプログラムの先頭にラベル名『_HIPRG_TIMINI』を付け、export 宣言してください。

タイマ初期設定ルーチンを未登録にする場合、ラベル名『_HIPRG_TIMINI』に 0 を equate 定義し、export 宣言してください。

(b) タイマ割込みハンドラの登録

タイマ割込みハンドラの先頭アドレスを割込みベクタテーブルに登録します。

(c) タイマドライバを未登録にする方法

タイマドライバを使用しない場合、以下の設定を行ってください。

- **割込みベクタテーブル**

H8S/2655 の場合、タイマ割込みハンドラの先頭アドレス(ラベル名『_H_2S_TIM』)の外部参照宣言(import)を削除し、ベクタ番号 32 に未定義割込みハンドラ(ラベル名『_H_2SINT32』)を登録します。

- **セットアップテーブル**

タイマスタックサイズ(ラベル名『TIMSTKSIZ』)を 0 にします。

- **タイマドライバ(タイマ初期設定ルーチン、タイマ割込みハンドラ)**

タイマ初期設定ルーチン(ラベル名『_HIPRG_TIMINI』)のプログラムを削除し、ラベル名『_HIPRG_TIMINI』に 0 を equate 定義します。

タイマ割込みハンドラ(ラベル名『_H_2S_TIM』)のプログラムを削除します。

OS のタイマ割込み処理(ラベル名『_H_timsys』)の外部参照シンボルの宣言(import)を削除します。

(5) 他の H8S シリーズを使用する場合の注意

H8S シリーズの TPU、FRT 以外のタイマを使用する場合、新たにタイマドライバを作成してください。

付録 D．エラーコード一覧

D.1　システムコールエラーコード一覧

表 D.1　システムコールエラーコード一覧

エラーコード (ニーモニック)		エラーコード(ercd)		エラーチェック種別	エラー内容
1	E_OK	H'0000	(H'0)	[k]	正常終了
2	E_RSFN	H'ffec	(-H'14)	[p]	未サポート機能（機能が未登録）
3	E_PAR	H'ffdf	(-H'21)	[p]/[k]	パラメータエラー
4	E_ID	H'ffdd	(-H'23)	[p]	不正 ID 番号
5	E_NOEXS	H'ffcc	(-H'34)	[p]	オブジェクトが存在していない オブジェクトが未登録
6	E_OBJ	H'ffc1	(-H'3f)	[k]	オブジェクトの状態不正
7	E_CTX	H'ffbb	(-H'45)	[p]/[k]	コンテキストエラー
8	E_QOVR	H'ffb7	(-H'49)	[k]	キューイングまたはネストのオーバフロー
9	E_TMOUT	H'ffab	(-H'55)	[k]	ポーリング失敗またはタイムアウト
10	E_RLWAI	H'ffaa	(-H'56)	[k]	待ち状態強制解除
11	EV_ILBLK	H'ff1e	(-H'e2)	[k]	不正メモリブロックの操作

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

D.2　デバッギングエクステンションエラー一覧

表 D.2　デバッギングエクステンションエラーメッセージ一覧(1/2)

エラーメッセージ	意味、対策
Cannot open HIOS window - no HIOS program loaded.	ロードモジュールがロードされていません。 ロードモジュールをロードしてください。
Cannot open memory display window @ H'xxxxxx Operation not implemented on this version of HDI.	Memory ウィンドウが表示できません。 HDI のバージョンによっては、本機能はサポートされません。
Cannot open program code window @ H'xxxxxx Operation not implemented on this version of HDI.	Program ウィンドウが表示できません。HDI のバージョンによっては、本機能はサポートされません。
ERROR : Command Already On Stack.	指定したコマンド要求は既にスタックされています。

表 D.2　デバッギングエクステンションエラーメッセージ一覧(2/2)

エラーメッセージ	意味、対策
ERROR : Demon Code Not Present. Command Cancelled.	デバッグデーモンが組み込まれていません。 「4.4.1 デバッグデーモンの組込み」を参照し、デバッグデーモンを組み込んでください。
ERROR : Demon Code Not Running. Command Cancelled.	デバッグデーモンの初期化が実行されていません。 カーネルを Go Reset 実行し、デバッグデーモンの初期化を行ってください。
Error : Number Out of Range	データが指定可能な範囲を超えています。 指定したデータを確認してください。
Error: Invalid input expression	指定したデータが不正です。 指定したデータを確認してください。
HIOS Error H'xxxx : <Error Message>	デバッグデーモンのシステムコールでエラーが発生した。 指定した ID の状態を確認してください。
Invalid Expression	指定可能なフラグ値ではありません。 指定したフラグ値を確認してください。
Invalid Format in Message Address	メッセージアドレスのフォーマットが不正です。 メッセージアドレスを確認してください。
Invalid Format in Message String!	指定したメッセージ文字列のフォーマットが不正です。 指定したメッセージを確認してください。
Timer Value invalid or wrong format!	タイマ値またはフォーマットが不正です。 指定したタイマ値を確認してください。
Unable to remove message from selected Mailbox.	選択したメイルボックスのメッセージを削除できません。 選択したメイルボックスを確認してください。
Unable to set breakpoint on HDI!	タスクが ROM 領域等にあるため、ブレークポイントを設定できません。 ブレークポイントを設定する領域を確認してください。
Value Too Large	フラグ値が指定可能な範囲を超えています。 指定したフラグ値を確認してください。

付録 E．システムコール機能コード一覧

E.1　システムコール機能コード一覧

以下に、システムコールトレース機能でのシステムコール名と機能コード対応表を示します。

表 E.1 システムコール名と機能コード対応表

項番	システムコール名	機能コード値	項番	システムコール名	機能コード値
1	ista_tsk	H'ff09 (-H'f7)	30	snd_msg	H'ffc1 (-H'3f)
2	trcv_msg	H'ff54 (-H'ac)	31	rcv_msg	H'ffc3 (-H'3d)
3	twai_sem	H'ff55 (-H'ab)	32	ref_mbx	H'ffc4 (-H'3c)
4	twai_flg	H'ff56 (-H'aa)	33	sig_sem	H'ffc9 (-H'37)
5	tget_blk	H'ff58 (-H'a8)	34	wai_sem	H'ffcb (-H'35)
6	tget_blf	H'ff59 (-H'a7)	35	ref_sem	H'ffcc (-H'34)
7	rel_blk	H'ff71 (-H'8f)	36	set_flg	H'ffd0 (-H'30)
8	get_blk	H'ff73 (-H'8d)	37	clr_flg	H'ffd1 (-H'2f)
9	ref_mpl	H'ff74 (-H'8c)	38	wai_flg	H'ffd2 (-H'2e)
10	isnd_msg	H'ff84 (-H'7c)	39	ref_flg	H'ffd4 (-H'2c)
11	isig_sem	H'ff85 (-H'7b)	40	can_wup	H'ffd8 (-H'28)
12	iset_flg	H'ff86 (-H'7a)	41	wup_tsk	H'ffd9 (-H'27)
13	iwup_tsk	H'ff87 (-H'79)	42	slp_tsk	H'ffda (-H'26)
14	irot_rdq	H'ff8a (-H'76)	43	tslp_ts	H'ffdb (-H'25)
15	prcv_msg	H'ff94 (-H'6c)	44	rsm_tsk	H'ffdd (-H'23)
16	preq_sem	H'ff95 (-H'6b)	45	sus_tsk	H'ffdf (-H'21)
17	pol_flg	H'ff96 (-H'6a)	46	rel_wai	H'ffe1 (-H'1f)
18	pget_blk	H'ff98 (-H'68)	47	dis_dsp	H'ffe2 (-H'1e)
19	pget_blf	H'ff99 (-H'67)	48	ena_dsp	H'ffe3 (-H'1d)
20	act_cyc	H'ffa2 (-H'5e)	49	rot_rdq	H'ffe4 (-H'1c)
21	ref_cyc	H'ffa4 (-H'5c)	50	chg_pri	H'ffe5 (-H'1b)
22	get_tim	H'ffac (-H'54)	51	ter_tsk	H'ffe7 (-H'19)
23	set_tim	H'ffad (-H'53)	52	get_tid	H'ffe8 (-H'18)
24	rel_blf	H'ffb1 (-H'4f)	53	sta_tsk	H'ffe9 (-H'17)
25	get_blf	H'ffb3 (-H'4d)	54	ext_tsk	H'ffeb (-H'15)
26	ref_mpf	H'ffb4 (-H'4c)	55	ref_tsk	H'ffec (-H'14)
27	ret_int	H'ffbb (-H'45)	56	get_ver	H'fff0 (-H'10)
28	ref_ims	H'ffbc (-H'44)	57	loc_cpu	H'fff8 (-H'8)
29	chg_ims	H'ffbd (-H'43)	58	unl_cpu	H'fff9 (-H'7)

付録 F. 索引

F.1　五十音索引

【サ行】

【タ行】

【ナ行】

【ハ行】

【マ行】

【ヤ行】

【ラ行】

【ワ行】

F.2　英数字索引

【E】

【F】

【G】

【H】

【I】

【L】

【M】

【N】

【O】

【T】

HI2000/3
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-702-276A