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

HI8-3H

ユーザーズマニュアル

ルネサスマイクロコンピュータサポートソフトウェア
Industrial Realtime Operating System H8/300H

ルネサス エレクトロニクス
www.renesas.com

Rev.2.00　1994.03

ご注意

1. 本書に記載の製品及び技術のうち「外国為替及び外国貿易法」に基づき安全保障貿易管理関連貨物・技術に該当するものを輸出する場合，または国外に持ち出す場合は日本国政府の許可が必要です。
2. 本書に記載された情報の使用に際して，弊社もしくは第三者の特許権，著作権，商標権，その他の知的所有権等の権利に対する保証または実施権の許諾を行うものではありません。また本書に記載された情報を使用した事により第三者の知的所有権等の権利に関わる問題が生じた場合，弊社はその責を負いませんので予めご了承ください。
3. 製品及び製品仕様は予告無く変更する場合がありますので，最終的な設計，ご購入，ご使用に際しましては，事前に最新の製品規格または仕様書をお求めになりご確認ください。
4. 弊社は品質・信頼性の向上に努めておりますが，宇宙，航空，原子力，燃焼制御，運輸，交通，各種安全装置，ライフサポート関連の医療機器等のように，特別な品質・信頼性が要求され，その故障や誤動作が直接人命を脅かしたり，人体に危害を及ぼす恐れのある用途にご使用をお考えのお客様は，事前に弊社営業担当迄ご相談をお願い致します。
5. 設計に際しては，特に最大定格，動作電源電圧範囲，放熱特性，実装条件及びその他諸条件につきましては，弊社保証範囲内でご使用いただきますようお願い致します。
   保証値を越えてご使用された場合の故障及び事故につきましては，弊社はその責を負いません。
   また保証値内のご使用であっても半導体製品について通常予測される故障発生率，故障モードをご考慮の上，弊社製品の動作が原因でご使用機器が人身事故，火災事故，その他の拡大損害を生じないようにフェールセーフ等のシステム上の対策を講じて頂きますようお願い致します。
6. 本製品は耐放射線設計をしておりません。
7. 本書の一部または全部を弊社の文書による承認なしに転載または複製することを堅くお断り致します。
8. 本書をはじめ弊社半導体についてのお問い合わせ，ご相談は弊社営業担当迄お願い致します。

はじめに

本マニュアルは、μITRON*1仕様に準拠した産業機器組み込み用リアルタイム・マルチタスクOS(Operating System)であるHI8-3H(Hitachi Industrial Realtime Operating System H8/300H)のシステムの機能について説明したものです。

HI8-3Hの構築方法については、UNIX HI8-3H構築マニュアルまたはMS-DOS HI8-3H構築マニュアルを参照してください。

＜マニュアルの構成＞

本マニュアルは以下に示す４つの章と付録から構成されております。

第１章では，HI8-3Hの概説を記述しています。本システムの全体を理解するために使用してください。

第２章では，HI8-3Hの機能を説明しています。ユーザプログラムの作成時に使用してください。

第３章では，HI8-3Hのシステムコールの仕様を説明しています。ユーザプログラムの作成時に使用してください。

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

付録では，コンソールドライバ、各種一覧表を記述しています。

＜本マニュアルで使用する記号などの意味＞

H'　　整数定数の16進数には、H'を付けています。それ以外は10進数です。

＜関連マニュアル＞

・UNIX HI8-3H構築マニュアルまたはMS-DOS HI8-3H構築マニュアル
・H8/300Hシリーズハードウェアマニュアル

＜注意＞

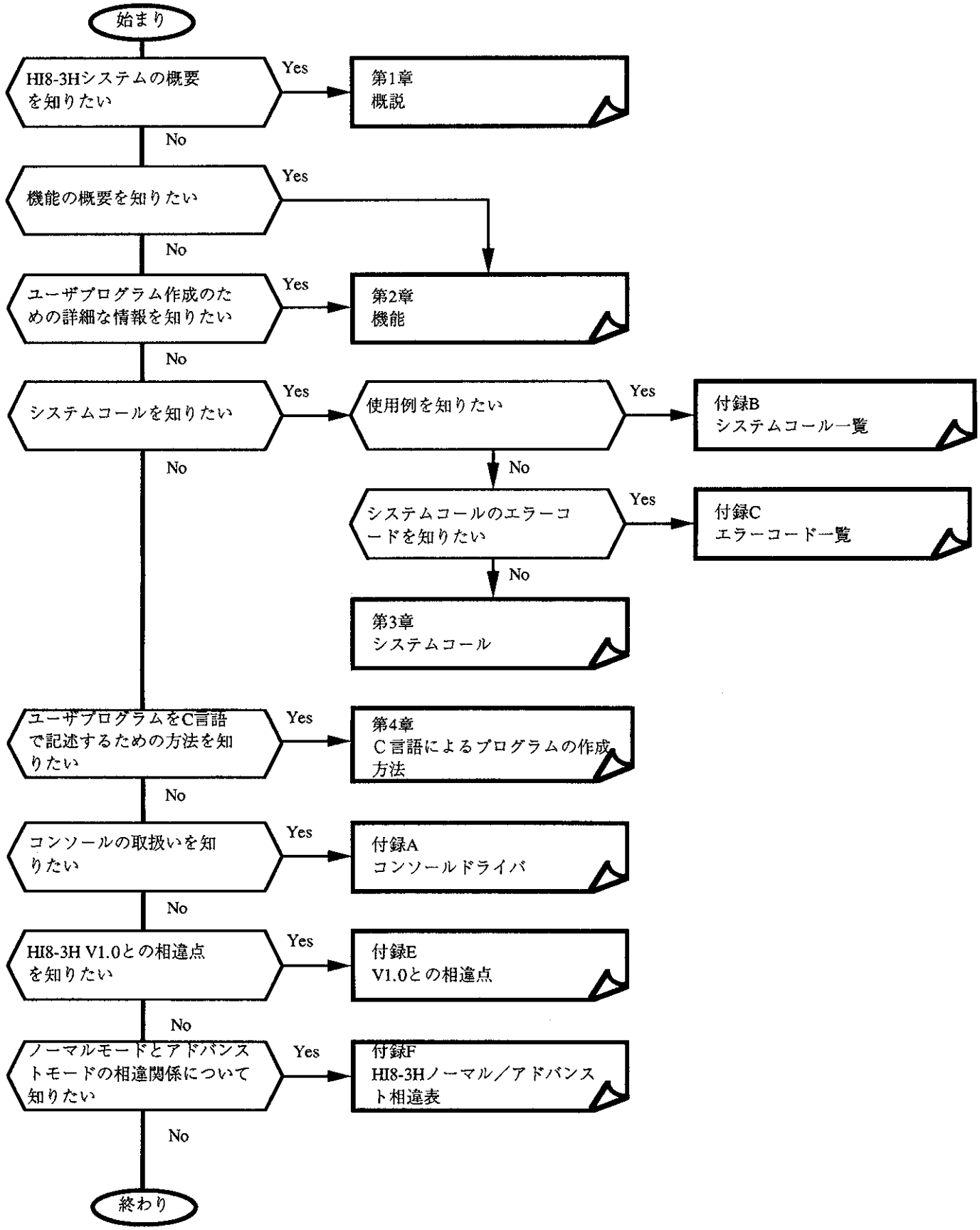
HI8-3H Ver.2.0は、HI8-3H Ver.1.0に対し、以下の機能向上を行なった製品です。

旧バージョン(HI8-3H Ver.1.0)との相違点については、「付録Ｅ．Ｖ１．０との相違点」を参照してください。

(1)H8/300Hノーマルモードをサポートしました。
(2)システムの作業領域を抑止する共有スタック機能をサポートしました。
(3)オンラインデバッガ用の機能としてトレース機能をサポートしました。

*1　μITRONは，"Micro Industrial TRONの略称です。
TRONは，"The Real time Operating System Nucleus"の略称です。

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

〈第２章　機能〉

〈第３章　システムコール〉

〈第４章　Ｃ言語によるプログラムの作成方法〉

〈付録Ａ．コンソールドライバ〉

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の役割と重要性が高まってきています。

このなかで、計測制御システムなどの産業用に用いられるOSとしてリアルタイムOSがあります。

H18-3Hは、日立Hシリーズマイコンの一つである、H8/300Hシリーズで動作する、産業機器組み込み用リアルタイム・マルチタスクOSです。

H18-3Hは、リアルタイムOSの標準仕様である、μITRON 仕様に準拠しています。

1．2　特　長

以下にH18-3Hの特長を示します。

(1) μITRON 仕様に準拠

リアルタイムOSの標準仕様である、μITRON 仕様に準拠しています。

(2) 高速なOS

16MHz 動作時、タスク起床時間36μsec （wup_tsk システムコール使用時の例）、割込みマスク時間12μsec 以下で動作する高速なリアルタイムＯＳです。

(3) コンパクトなOS

ＯＳのプログラムサイズ（ノーマルモード用：最小構成 約０．７Ｋバイト）および作業領域（タスク１個当たり１８バイト）とコンパクトなＯＳです。また、複数のタスクでスタック領域を共有することができるので、少ないＲＡＭ容量のシステムでも多くのタスクを実行することができます。

(4) 容易なシステム構築

ＯＳの機能モジュールはライブラリ形式で提供されますので、リンケージエディタで結合するだけで、自動的に必要最小限の機能モジュールでシステムを構築することができます。

(5) 豊富なリアルタイム・マルチタスク処理機能

H18-3Hは、表１－１に示すようなリアルタイム・マルチタスク処理機能をもっています。

表１－１　H18-3Hのリアルタイム・マルチタスク機能

項番	機能
1	優先度に基づくタスクスケジューリング
2	イベントフラグ・セマフォ・メイルボックスによるタスク間の同期・通信機能
3	割込み管理機能
4	メモリ管理機能
5	時間・タスクの実行遅延等の時間管理機能

(6) トレース機能

システムコールの発行順序やマルチタスク環境での個々のタスクの動きを知ることができる機能を提供しています。

(7) 高級言語インタフェース

Ｃ言語でプログラム開発が行なえるように、Ｃ言語インタフェースライブラリを用意しています。

(8) デバイスドライバを容易に作成可能

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

１．３　構　成

HI8-3Hを構成する主要なプログラムには次のものがあります。

(1) カーネル

タスク管理，タスク間の同期通信，時間管理など、リアルタイム・マルチタスク制御の中枢となるプログラムです。

H8/300HのCPU動作モードに応じて、ノーマルモード用とアドバンストモード用の２種類のＯＳがあります。

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するＯＳです。

アドバンストモード用は、最大１６Ｍバイトのアドレス空間で動作するＯＳです。

(2) セットアップテーブル

タスク・イベントフラグなど各種資源の情報を登録するためのテーブルです。

セットアップテーブルの登録方法については、UNIX HI8-3H構築マニュアルまたはMS-DOS HI8-3H構築マニュアルを参照してください。

(3) Ｃ言語インタフェースライブラリ

HI8-3Hの各システムコールを、Ｃ言語で記述したユーザプログラムから呼び出すためのライブラリです。

(4) コンソールドライバ

H8/300Hシリーズ内蔵のシリアルコミュニケーションインタフェース(SCI:Serial Communication Interface)を使用したコンソールを制御するサンプルプログラムです。

コンソールを使用しない場合は、ユーザシステムに組み込む必要はありません。

(5) タイマドライバ

H8/300Hシリーズ内蔵のインテグレーテッドタイマパルスユニット(ITU:Integrated Timer pulse Unit)を使用した、システムクロックを制御するサンプルプログラムです。wai_tsk, set_tim, get_timシステムコールを使用しない場合は、ユーザシステムに組み込む必要はありません。

表１－２にH18-3Hの各プログラム容量を示します。

表１　２　プログラム容量

項番	プログラム名		容量
1	カーネル（H8/300Hﾉｰﾏﾙﾓｰﾄﾞ用）	最小構成	約０．７Ｋバイト
		最大構成*1	約１２．６Ｋバイト
	カーネル（H8/300H ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用）	最小構成	約０．８Ｋバイト
		最大構成*1	約１２．７Ｋバイト
2	セットアップテーブル*2		９７バイト以上
3	Ｃ言語インタフェースライブラリ	ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用	約１．６Ｋバイト
		ﾉｰﾏﾙﾓｰﾄﾞ用	約１．３Ｋバイト
4	コンソールドライバ	ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用	約０．７Ｋバイト
		ﾉｰﾏﾙﾓｰﾄﾞ用	約０．６Ｋバイト

【注】 *1 タイマドライバサンプルを含みます。

*2 使用するタスク数など、構築するシステムの規模によって異なります。

2．機　能

2．1　概　要

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

・タスク（独立かつ並列に処理可能な単位に分割されたプログラム）

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

ユーザは、リアルタイム・マルチタスクOS：HI8-3Hを利用して、リアルタイム・マルチタスクシステムを実現します。

HI8-3Hのリアルタイム・マルチタスク処理を行なう核となる部分を、カーネルと呼びます。

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（イベント）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）の実行順序を決定する。

(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、その処理を行ないます。

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

2．1．1　HI8-3Hの機能構成

HI8-3Hは、次の６つの各機能により構成されています。

以下に、その概要を示します。

(1) タスク管理（スケジューラを含む）

CPU(Central Processing Unit)をタスクに割り付ける順序やタスクの起動・終了など、タスクの状態を管理します。

タスクは、優先度の高いものから順にCPU に割り付けられます。詳細は、「２．２　タスク」を参照してください。

(2) タスク付属同期管理

タスクの実行中断・再開など、タスクの基本的な同期処理が行なわれます。詳細は、「２．２　タスク」を参照してください。

(3) 同期・通信管理

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ないます。詳細は、「２．３　イベントフラグ」，「２．４　セマフォ」，「２．５　メイルボックス」を参照してください。

(4) 割込み管理

割込み発生時に割込みハンドラ* が起動され、割込みやタスクへの割込み発生の連絡が行なわれます。詳細は、「２．６　割込み」を参照してください。

【注】 *　割込みハンドラは、必要な場合作成しなければなりません。

(5) メモリ管理

固定長メモリ領域の獲得、不要になったメモリ領域を返却する処理を行ないます。

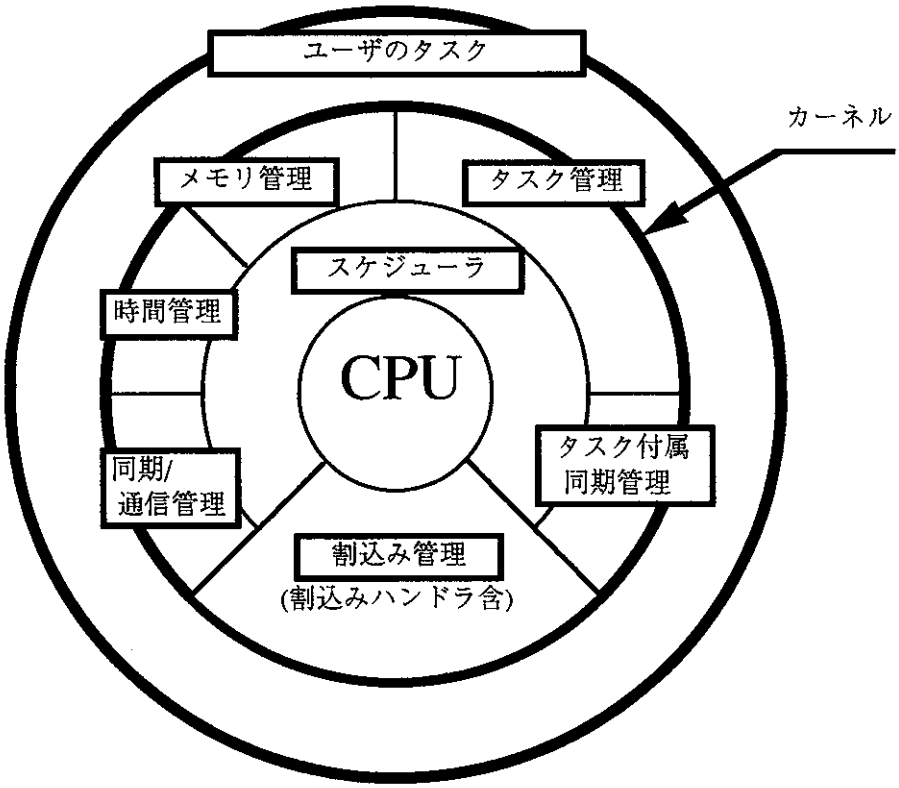
詳細は、「２．７　メモリプール」を参照してください。

(6) 時間管理

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

詳細は、「２．８　時間」を参照してください。

図２－１にHI8-3Hの機能構成を示します。



【注】タスクと割込み管理の割込みハンドラは、ユーザが作成します。

図２－１　HI8-3Hの機能構成

２．２　タスク

２．２．１　タスクの概要

リアルタイム・マルチタスクシステムでは、ユーザはプログラムを独立して並列に処理可能な単位に分割して作成します。この分割したプログラムをタスクと呼びます。

ユーザは最大255個のタスクを使用でき、タスクは１から255までのタスクIDと呼ぶ番号で識別します。

タスクは、タスク間で必要な連絡にカーネルの機能であるシステムコールを使用して行ないます。ユーザは、カーネルの機能（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。

図２－２にタスクとカーネルの関係を示します。

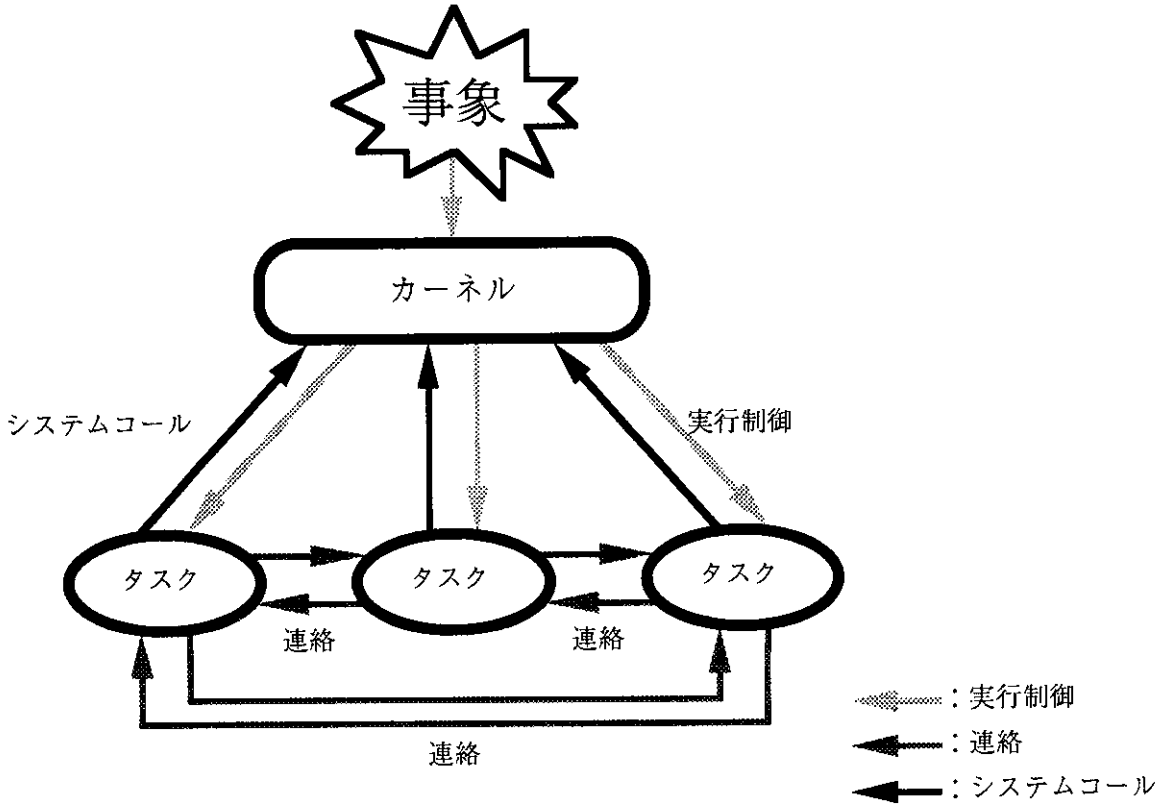


図２－２　タスクとカーネルの関係

２．２．２　タスクの並列処理

タスクは、発生した事象（タスクの実行要求）にしたがって、HI8-3Hシステム内でタスクの状態とタスクに付けられた優先度に基づいて並列に実行されます。

タスクの優先度は１から31までの値で表わされ、値の小さい方が高い優先度になります。

図２－３にタスクの並列処理の概要を示します。

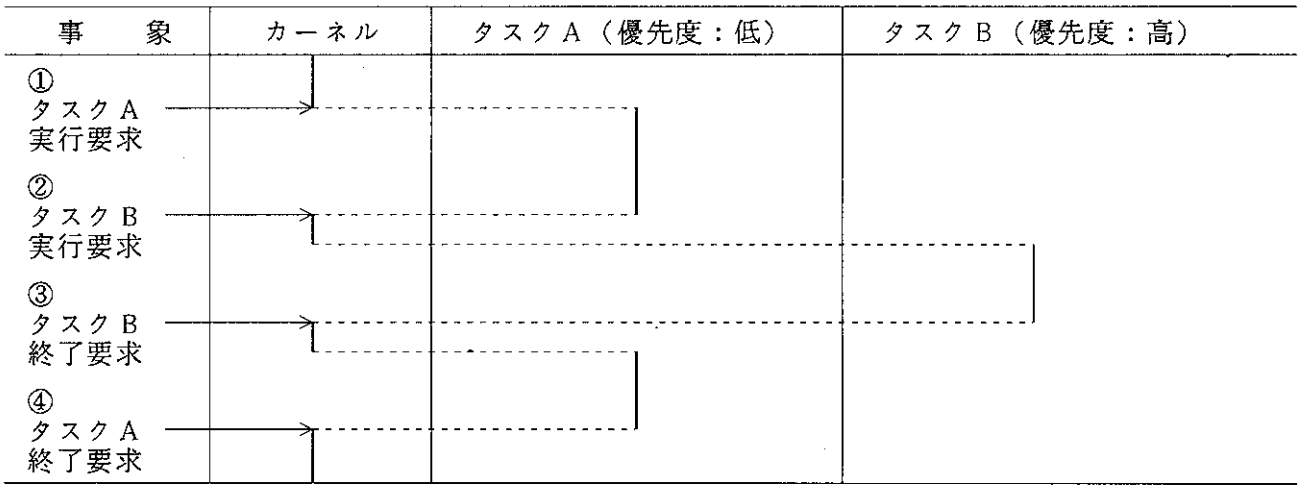


図２－３　タスクの並列処理の概要

（解説）

①タスクＡの実行要求にしたがい、タスクＡが実行されます。

②タスクＢの実行要求が発生しました。タスクＡの実行が一時中断され、優先度の高いタスクＢを実行します。

③タスクＢの終了要求にしたがい、タスクＢの実行を終了し、実行を中断されていたタスクＡの実行を再開します。

④タスクＡの終了要求にしたがいタスクＡの実行を終了します。

このようにタスクに付けられた優先度に基づいて複数のタスクが実行され、並列処理を実現します。

２．２．３　タスクの状態

タスクは、外部や内部からの処理要求（システムコール）、事象発生により６つの状態を遷移します。

(1) 休止(DORMANT) 状態

タスクが登録されてまだ起動されていない状態、または終了後の状態です。

休止状態のタスクが他のタスクからsta_tsk システムコールにより起動されると、タスク登録時に指定されたタスク実行開始アドレスから実行されます。

タスク自身がext_tsk システムコールで終了するか、他のタスクからter_tsk システムコールで強制的に終了させられると、休止状態に移行します。

(2) 実行可能(READY) 状態

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

休止状態のタスクが起動されたとき、または待ち状態のタスクの待ちが解除されたとき、実行可能状態に移行します。

(3) 実行(RUN) 状態

CPU が割り付けられ、現在実行中の状態です。

実行可能状態のタスクの中で最も高い優先度のタスクが、実行状態に移行します。

(4) 待ち(WAIT)状態

タスクが何らかの事象の発生を待っている状態です。

実行状態のタスクが待ちを伴うシステムコールを発行し、条件が満足されないとき、待ち状態に移行します。

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により強制的に中断された状態です。

実行可能状態のタスクに対して他のタスクからsus_tsk システムコールを発行されると、強制待ち状態に移行します。

強制待ち状態のタスクに対して他のタスクからrsm_tsk システムコールが発行されると、強制待ち状態は解除されます。

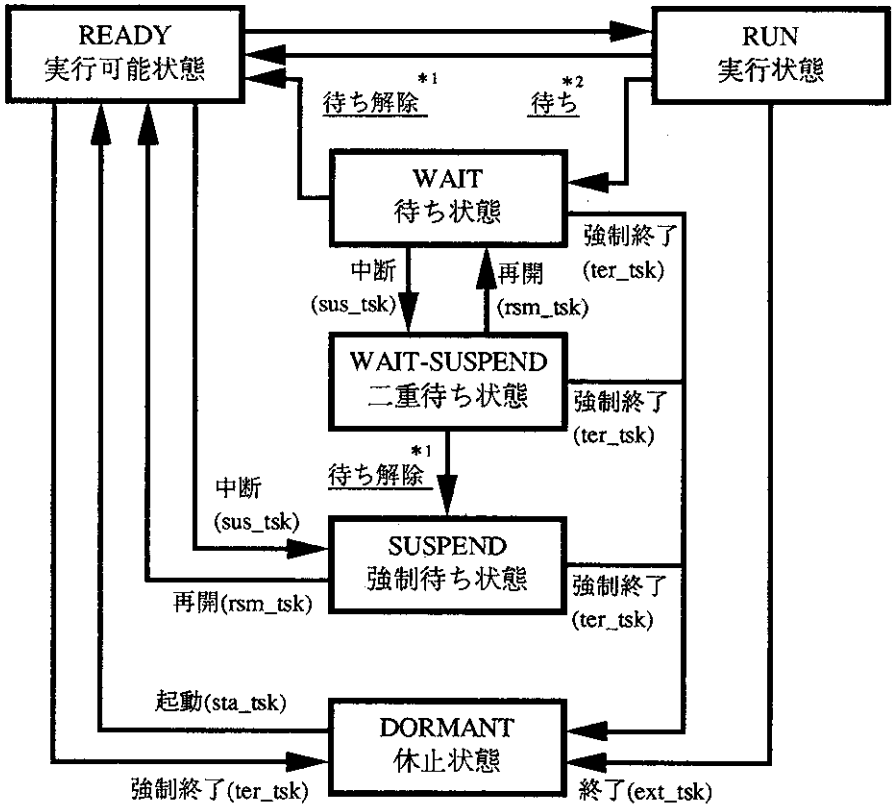
(6) 二重待ち(WAIT-SUSPEND)状態

待ち状態と強制待ち状態が重なった状態です。

待ち状態のタスクが、他のタスクからsus_tsk システムコールを発行されると、二重待ち状態に移行します。

二重待ち状態は、待ち状態の待ち条件が解除され、かつrsm_tsk システムコールにより強制待ち状態が解除されなければ、実行可能状態に移行できません。

図２－４にタスクの状態遷移を示します。



【注】*1　待ち解除となる事象，システムコール
時間経過，wup_tsk，iwup_tsk，set_flg，iset_flg，sig_sem，isig_sem，snd_msg，isnd_msg，rel_blk，rel_wai
*2　待ち条件となるシステムコール
slp_tsk，wai_tsk，wai_flg，wai_sem，rcv_msg，get_blk

図２－４　タスクの状態遷移

２．２．４　タスクの起動

タスクの起動とは、タスクが休止状態から実行可能状態になることをいいます。

以下にタスクを起動する要因，タスク起動時のレジスタと優先度の初期化状態を示します。

(1)タスクの起動要因

表２－１にタスクの起動要因を示します。

表２－１　タスクの起動要因

項番	分類	起動要因
1	初期設定	タスクが実行可能状態で登録されている場合
2	他のタスクから起動	sta_tsk システムコールが発行された場合
3	割込みハンドラから起動	ista_tskシステムコールが発行された場合

(2)レジスタの初期化

表２－２にタスク起動時のレジスタ初期化を示します。

表２－２　レジスタの初期化

項番	レジスタ名	初期化の内容
1	プログラムカウンタ(PC)	タスク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)	割込みマスク解除状態
3	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
4	汎用レジスタ(ER0～ER6)	不定

(3)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が設定されます。

２．２．５　共有スタック機能

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

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

図２－５は共有スタック機能の処理概要です。

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

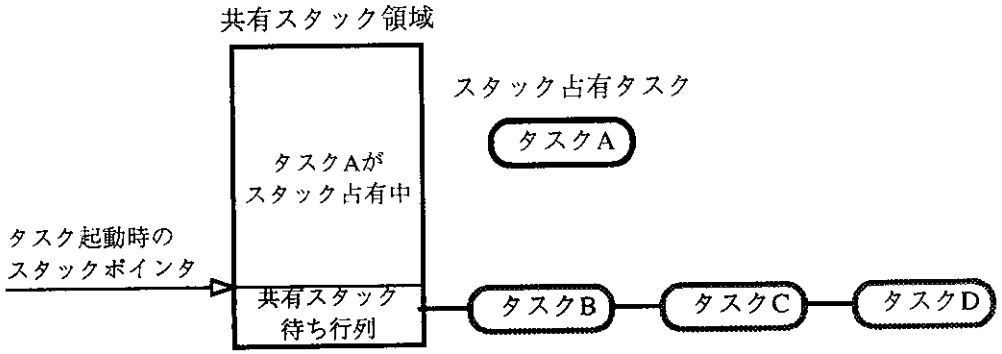
まず、タスクAに起動要求(sta_tskシステムコール発行)が行なわれると、タスクAが共有スタック領域を使用(占有)します。タスクAが共有スタック領域を占有中に、この共有スタック領域を使用(占有)するタスクB、C、Dの順番に起動要求が行なわれた場合、タスクB、C、Dは共有スタック領域を使用(占有)できません。そのため、タスクA以外は実行可能状態(READY状態)になることができず、起動要求を保留され、共有スタック待ち状態となります(起動要求の保留)。

このとき、たとえタスクBがタスクＡより高いタスク優先度であっても、タスクBは待ち状態になります。

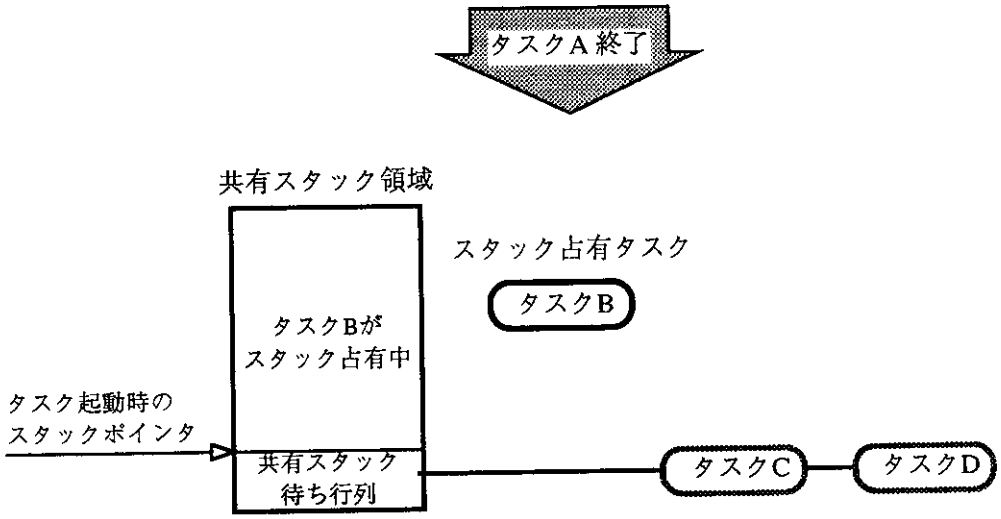
共有スタック待ち状態となったタスクB、C、Dは、この共有スタック領域の待ち行列につながれます。共有スタック待ち行列は、優先度に関係なくFIFO(First-In First-Out)で管理され、起動要求が行なわれた順番(タスクB、C、D)につながれます。
このときの共有スタックの状態は、図２－５の(a)のようになります。

共有スタック領域は、共有スタック領域を使用(占有)していたタスクが終了することにより解放されます。そして、そのときに共有スタック待ち行列のタスクがあれば、共有スタック待ち行列の先頭タスクが共有スタック領域を使用(占有)し、実行可能状態に遷移します(起動要求の保留解除)。
このときの共有スタックの状態は、図２－５の(b)のようになります。

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



(a)共有スタックをタスクAが占有中のスタック待ち行列の状態



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

図２－５　共有スタック機能の動作

なお、共有スタック領域を使用(占有)しているタスクがwai_tskシステムコールやwai_flgシステムコールなどを発行して待ち状態となっても、共有スタック領域は解放されません。タスクスタック領域の共有を指定せずに登録されたタスクは、共有スタック待ち状態となることはありません。

図２－６に共有スタック機能を使用するタスクの状態遷移を示します。

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

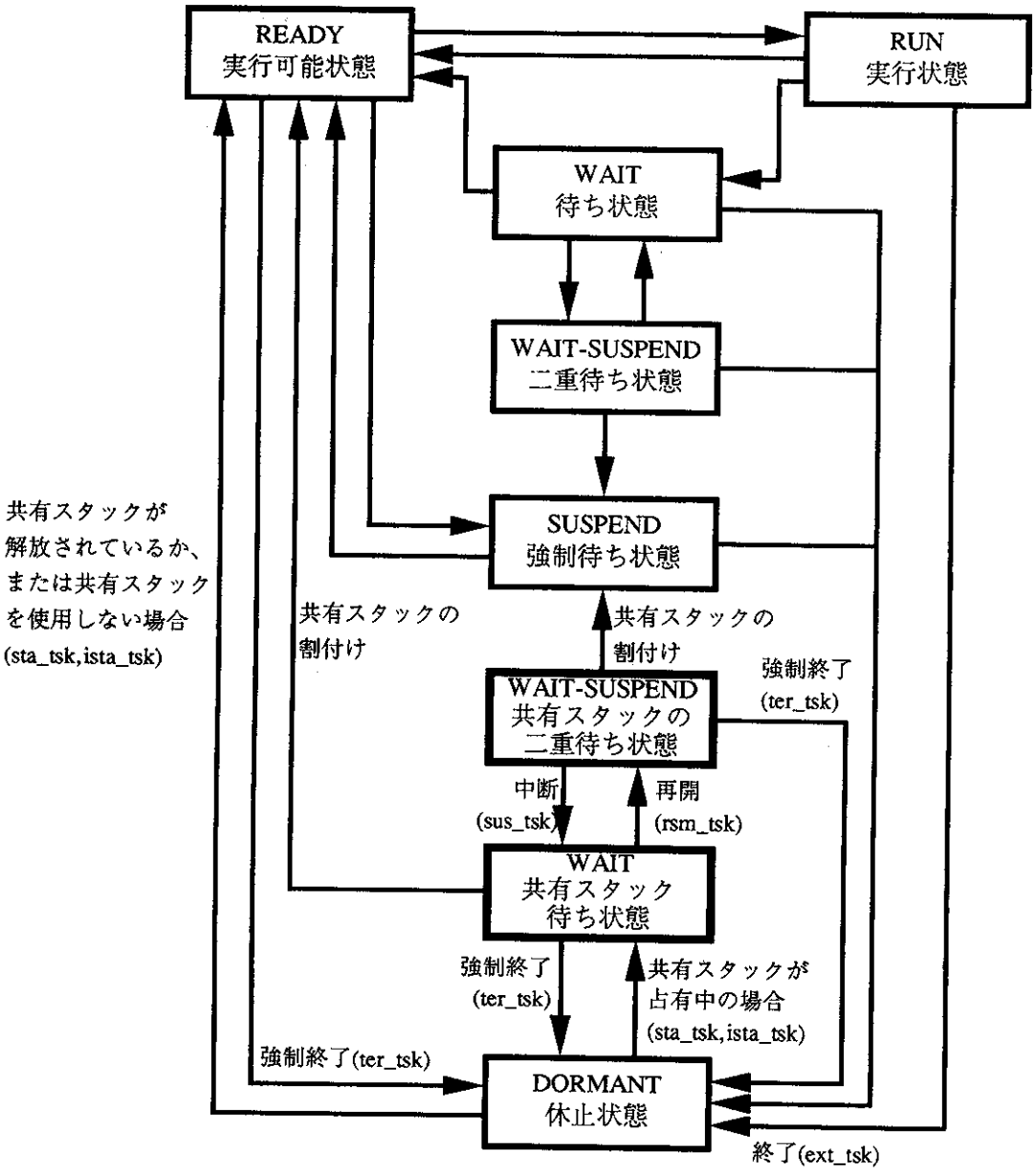


図２－６　共有スタック機能使用時のタスク状態遷移

２．２．６　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、カーネルはシステムアイドリング状態になり、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、タスクの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、1から31までを使用することができます。タスクの優先度は、値が小さい方が高い優先度になります。

図２－７にレディキューの概要を示します。

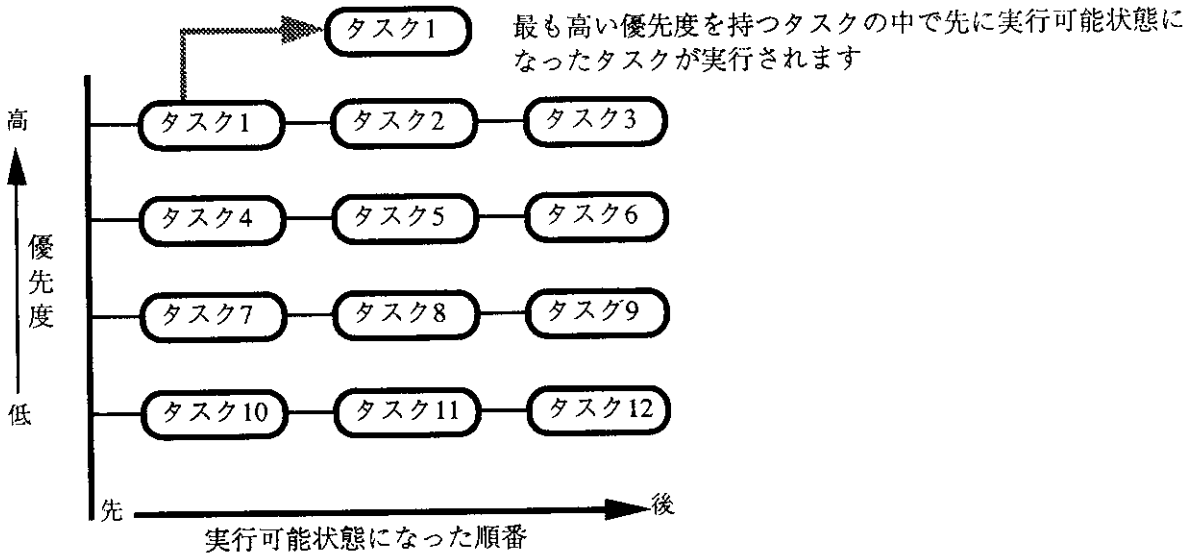


図２－７　レディキューの概要

スケジューリングには、大きく分けて次の２つがあります。

(1) HI8-3H標準のスケジューリング

実行可能状態で最も高い優先度のタスクに、CPUが割り付けられます。

同一優先度のタスクが複数存在する場合には、その中で最も早く実行可能状態になったタスクにCPU が割り付けられます。

タスク実行中により高い優先度の別のタスクが実行可能状態になった場合、元の実行状態のタスクは、実行可能状態に遷移し、より高い優先度のタスクが実行されます。

タスクは、CPU が割り付けられ実行状態に移行してもレディキューから外されません。

タスクは、待ち状態または休止状態になった場合レディキューから外されます。

(2) ラウンドロビンスケジューリング

HI8-3Hでは、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なうことができます。

ラウンドロビンスケジューリングとは、一定時間毎にレディキューを回転させ、同じ優先度を待つタスクにCPU の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、タイマ機能とirot_rdq システムコールを組み合わせることで実現できます。

図２－８にタイマ割込みとirot_rdq システムコールによるラウンドロビンスケジューリングを示します。

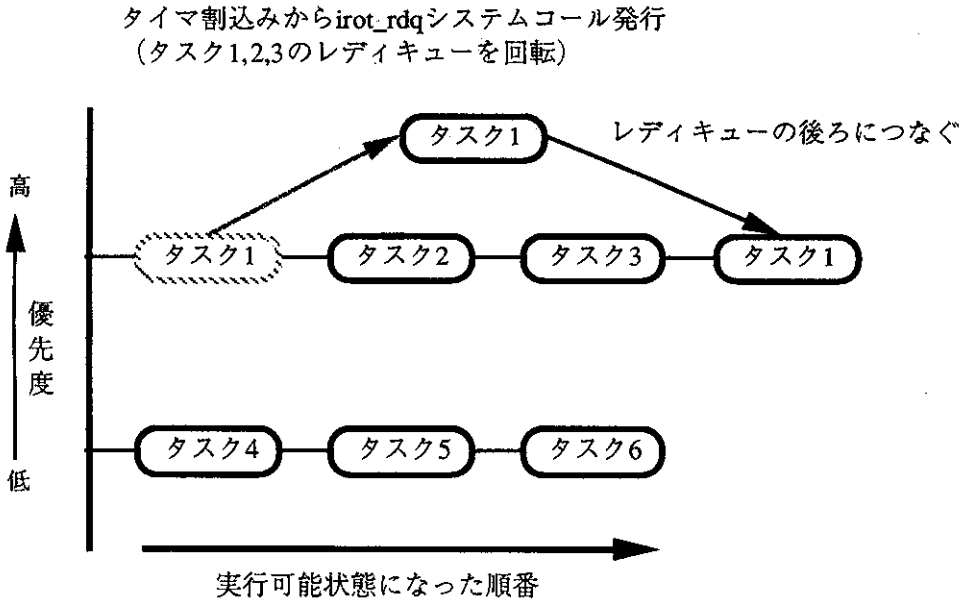


図２－８　タイマ割込みとirot_rdq システムコールによるラウンドロビンスケジューリング

２．２．７　タスクの中断・再開

タスクの実行は、割込みや資源獲得待ち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。

表２－３にタスク実行中断の要因と期間を示します。

表２－３　中断の要因と期間

項番	中断要因		中断期間
1	自ら中断となる場合	slp_tsk システムコール	(1)wup_tsk, iwup_tskシステムコールが発行されるまで (2)rel_wai システムコールが発行されるまで
		wai_tsk システムコール	(1)wup_tsk, iwup_tskシステムコールが発行されるまで (2)指定したタイムアウト時間(tmout) が経過するまで (3)rel_wai システムコールが発行されるまで
		wai_flg システムコール	(1)set_flg, iset_flgシステムコールによって、待ち解除条件が成立するまで (2)rel_wai システムコールが発行されるまで
		wai_sem システムコール	(1)sig_sem, isig_semシステムコールによって、管理されている資源が獲得できるまで (2)rel_wai システムコールが発行されるまで
		rcv_msg システムコール	(1)snd_msg, isnd_msgシステムコールによって、メッセージが送られるまで (2)rel_wai システムコールが発行されるまで
		get_blk システムコール	(1)メモリプールから固定長メモリブロックが獲得できるまで (2)rel_wai システムコールが発行されるまで
2	他タスクから中断させられる場合	sus_tsk システムコール	rsm_tsk システムコールの発行により強制待ち状態が解除されるまで
3	割込みにより中断させられる場合	タイマ割込み	タイマ割込みが発生し、タイマ割込み実行後、もとのタスクに戻ってくるまで
		その他の割込み	割込みが発生し、その割込みハンドラ実行後、もとのタスクに戻ってくるまで
4	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

２．２．８　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。
タスクは一度終了すると、次に起動がかけられたとき再び初期状態から実行されます。

(1) タスクの終了要因

表２－４にタスクの終了要因を示します。

表２－４　タスクの終了要因

項番	分類	要因	終了後の状態
1	正常終了	ext_tsk システムコールを発行した	休止状態
2	他からの強制終了	ter_tsk システムコールが発行された	

(2) 終了前の処理

タスクは終了前に、システムコールにより獲得していた資源を開放しなければなりません。
表２－５に資源の一覧を示します。

表２－５　資源の一覧

項番	資源	内容	獲得のｼｽﾃﾑｺｰﾙ	解放のｼｽﾃﾑｺｰﾙ
1	メモリブロック	メモリプールから獲得した メモリ領域	get_blk pget_blk	rel_blk
2	セマフォカウント	P命令で獲得した資源数	wai_sem preq_sem	sig_sem isig_sem

２．２．９　タスク実行中の割込みマスク

タスクは、chg_ims システムコールを用いてタスク実行中に割込みをマスクすることができます。

タスクが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。

タスクが割込みをマスクして実行しているときは、通常に実行しているときとくらべ以下の点が異なります。

(1) タスク部専用システムコールの制限

タスク部専用のシステムコールを発行すると、カーネルはコンテキストエラー(E_CTX)を返し、処理を行ないません。

(2) 非タスク部専用システムコールの制限

非タスク部専用のシステムコール(ista_tsk, irot_rdq, iwup_tsk, iset_flg, isig_sem, isnd_msg, ret_intの各システムコール)を発行すると、カーネルはコンテキストエラー(E_CTX) を返しません。

この場合、H18-3Hシステムの動作は保証されません。注意してください。

(3) タスクの切り換えの制限

割込みをマスクしている期間は、そのタスクよりも優先度の高いタスクが実行可能状態となっても、タスクの切り換えは行なわれず割込みをマスクしているタスクの実行が続行されます。タスクの切り換えは、割込みマスク解除後の最初のシステムコール発行時、またはタイマ割込み発生時に行なわれます。

２．３　イベントフラグ

２．３．１　イベントフラグの概要

イベントフラグを使用することにより、複数の事象発生の組合せによるタスク間の同期処理を実現することができます。

イベントフラグは、事象に対応した16ビットのフラグの集合です。１つのイベントフラグで16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)のみで行なわれるため高速に処理可能です。

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、１つのタスクが事象の発生を待つことができます。

イベントフラグは、最大255個まで使用することができ、1から255までのイベントフラグIDと呼ぶ番号で識別します。

イベントフラグの登録は、ユーザシステムに必要なイベントフラグの総数をセットアップテーブルに設定することにより行ないます。

イベントフラグの初期値は、H'0000 固定です。

表２－６に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２－６　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	事象の発生を通知する
2	clr_flg	イベントフラグをクリアする
3	wai_flg	事象の発生を待つ
4	pol_flg	事象の発生を得る（ポーリング）
5	flg_sts	イベントフラグ状態を参照する

図２－９にイベントフラグの操作を示します。

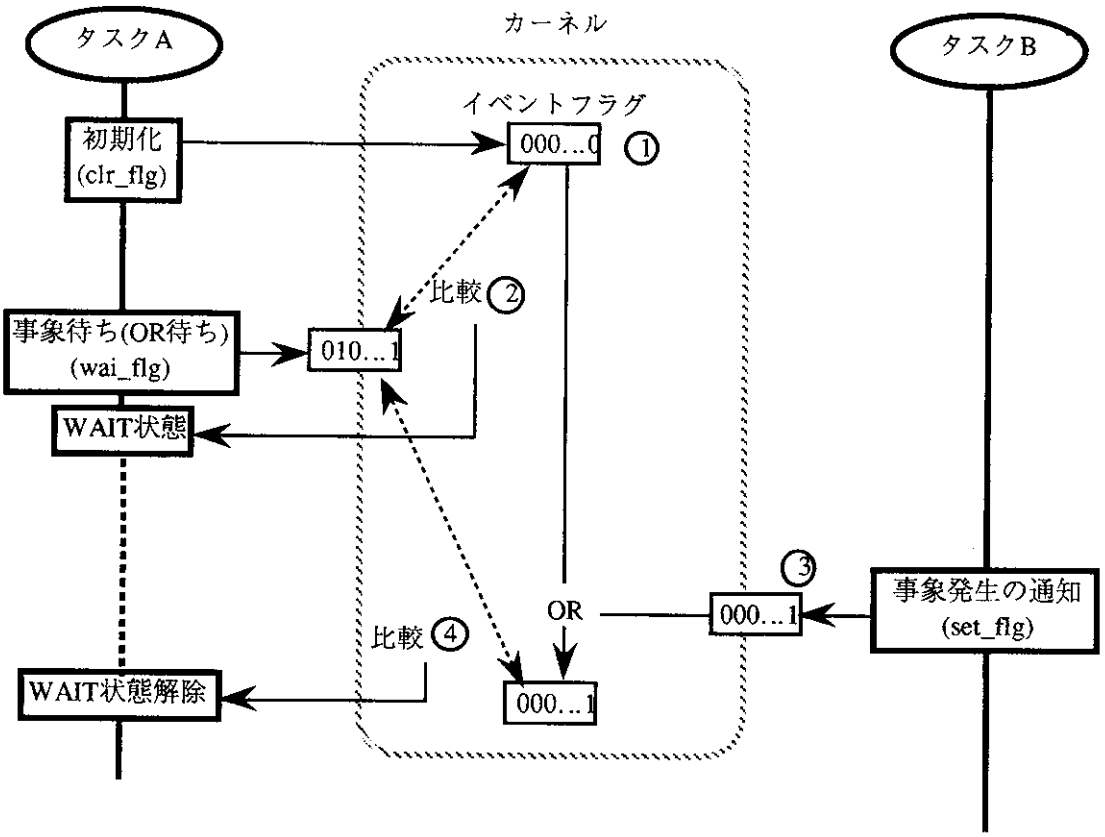


図２－９　イベントフラグの操作

（解説）

①タスクAがイベントフラグを全ビットクリアします。

②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になります。(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)

③タスクＢが事象発生の通知を行ない、イベントフラグの内容が変ります。

④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。

２．３．２　事象の発生と待ちの解除

事象の発生の通知と待ちの解除は、set_flg,iset_flgシステムコールを使用します。（iset_flgシステムコールは、非タスク部専用のset_flg システムコールです。）

事象の発生の通知と待ちの解除は、以下のように行ないます。

(1) 事象の発生の通知

発生した事象に対応したビットをオン(1)にした16ビットのビットパターンをパラメータとして、set_flg システムコールを発行します。

(2) 待ち解除成立の調査

イベントフラグの内容とパラメータのビットパターンで論理和(OR)をとり、イベントフラグの内容を更新します。更新したイベントフラグの内容から、待ちタスクの待ち状態解除条件が成立するかどうかを調べます。

(3) 待ちの解除

待ち状態解除条件が成立すると、待ちタスクを実行可能状態に移行します。

実行が再開されたタスクには、wai_flg システムコールのリターンパラメータとして、待ち状態解除条件成立時のイベントフラグの内容が返されます。

(4) イベントフラグのクリア

wai_flg システムコール発行時にイベントフラグのクリア指定が行なわれていれば、待ち状態解除条件成立後、イベントフラグの内容がクリアされます。

２．３．３　イベントフラグのクリア

クリアするビットをオフ(0) にしたビットパターンをパラメータとして、clr_flg システムコールを発行します。

イベントフラグの内容とクリアするビットパターンとの論理積(AND) によりイベントフラグの内容がクリアされます。

２．３．４　事象の発生を待つ・事象の発生を得る

事象の発生を待つには、wai_flg システムコールを使用します。
事象発生の待ちを行なわない場合（ポーリング）は、pol_flg システムコールを使用します。
事象の発生待ちと取得は、以下のように行ないます。

(1) 事象の発生の待ち

事象に対応したビットをオン(1) にした16ビットのビットパターンと、事象の待ちモード（AND待ち，OR待ち）をパラメータとしてwai_flg,pol_flgシステムコールを発行します。

(2) 事象の発生の調査

事象が発生しているかを調べ、事象が発生している場合、リターンパラメータとしてイベントフラグの内容を返します。

(3) 事象が発生していない場合の処理

wai_flg システムコールを発行した場合、タスクは事象が発生するまで待ち状態に移行します。
pol_flg システムコールを発行した場合、エラーコードとしてポーリング失敗エラー(E_PLFAIL)が返されます。

(4) 事象の待ちモードの指定

wai_flg,pol_flg システムコールを発行するとき、パラメータとして16ビットのビットパターンの他に、事象の待ちモードを指定することができます。
待ちモードを指定することによって、事象発生時の待ち状態を解除する条件を指定できます。
また、待ちモードにクリア指定を行なうと、事象の待ち解除処理後にイベントフラグの内容をクリアすることができます。
表２－７に待ちモードの解除条件を示します。

表２－７　待ちモードの解除条件

項番	待ちモード	解除条件	条件式
1	AND待ち	ビットパターンのオン(1)で指定したビットに対応するすべての事象の発生	(ｲﾍﾞﾝﾄﾌﾗｸﾞ)∩(ﾋﾞｯﾄﾊﾟﾀｰﾝ)=(ﾋﾞｯﾄﾊﾟﾀｰﾝ)
2	OR 待ち	ビットパターンのオン(1)で指定したビットに対応する事象が少なくとも１つ発生	(ｲﾍﾞﾝﾄﾌﾗｸﾞ)∩(ﾋﾞｯﾄﾊﾟﾀｰﾝ)≠0

２．３．５　イベントフラグ状態の参照

現在のイベントフラグの状態を参照するには、flg_sts システムコールを使用します。
flg_sts システムコールは、リターンパラメータとして現在のイベントフラグのビットパターンと、イベントフラグの待ちタスクIDが返されます。

２．４　セマフォ

２．４．１　セマフォの概要

タスク実行のために必要になる各種の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なうことができます。

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(0～65535)を使用可能な資源の数に対応させて使用します。

タスクは、セマフォに対しカウンタの獲得（占有）を行ない、カウンタの獲得に応じて資源を使用できます。このセマフォカウンタの獲得は、資源の獲得と同じ意味になります。

セマフォは、最大255個まで使用することができ、1から255までのセマフォIDと呼ばれる番号で識別します。

セマフォの登録は、ユーザシステムに必要なセマフォの総数をセットアップテーブルに設定することにより行ないます。

セマフォカウンタの初期値は、１固定です。セマフォカウンタ値を資源と同じにするには、システム初期化ハンドラから必要に応じてisig_semを発行し資源数を設定します。

表２－８にセマフォを操作するシステムコールを示します。

表２－８　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	資源を解放する（Ｖ命令）
2	wai_sem	資源を占有要求する（Ｐ命令）
3	preq_sem	資源を得る（ポーリング）
4	sem_sts	セマフォ状態を参照する

図２－１０にセマフォによる資源の排他制御の例を示します。

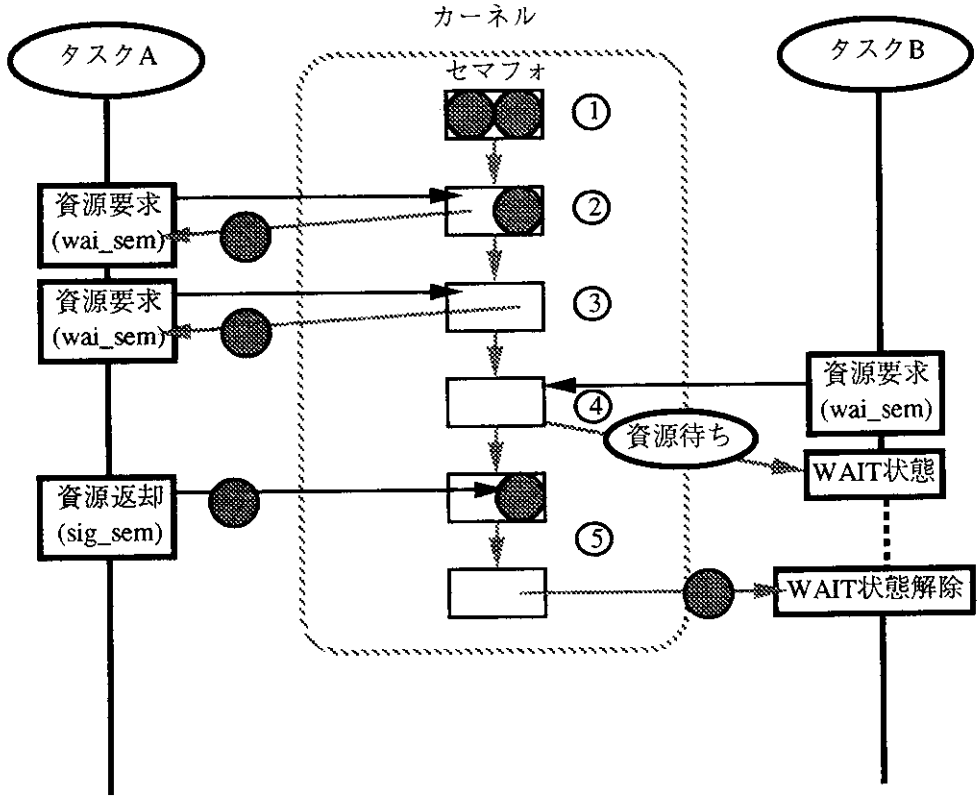


図２－１０　セマフォによる資源排他制御例

（解説）

①最初、資源が２つあります。（セマフォカウンタ=2）

②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）

③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）

④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。

⑤タスクＡが１個の資源を解放したため、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。

２．４．２　資源の占有要求（Ｐ命令）

資源の占有要求（Ｐ命令）は、wai_sem システムコールを使用します。
資源占有要求の待ちを行なわない場合（ポーリング）は、preq_semシステムコールを使用します。
資源の占有要求は、以下のように行ないます。

(1) 資源の占有要求
資源の占有要求を行なうため、wai_sem,preq_semシステムコールを発行します。
(2) 資源数の調査
使用可能な資源数（セマフォカウンタ）が０かどうかを調べ、０でなければセマフォカウンタから１を減算し、タスクに資源を割り付けます。
(3) 資源数が０の場合の処理
wai_sem システムコールを発行した場合、タスクをセマフォの待ち行列につなぎ、待ち状態に移行します。
preq_semシステムコールを発行した場合、エラーコードとしてポーリング失敗エラー(E_PLFAIL)が返されます。
セマフォの待ち行列は、FIFO(First-In First-Out)で管理されます。

２．４．３　資源の解放（Ｖ命令）

wai_sem システムコールで占有した資源の解放は、sig_sem,isig_semシステムコールを使用します。（isig_semは、非タスク部専用のsig_semシステムコールです。）
資源の解放は、以下のように行ないます。

(1) 資源の解放
資源を解放するため、sig_sem,isig_semシステムコールを発行します。
(2) 待ちの解除
セマフォに待ちタスクが存在するかどうかを調べます。
待ちタスクが存在する場合は、待ち行列の先頭のタスクにＰ命令処理同様に資源が割り付けられ、待ち行列からはずされます。
待ちタスクが存在しない場合、セマフォカウンタを１だけ加算します。
この時、最大値(65535) を超える場合は、エラーコードとしてE_QOVRが返されます。

２．４．４　セマフォ状態の参照

現在のセマフォの状態を参照するには、sem_sts システムコールを使用します。
sem_sts システムコールは、リターンパラメータとして現在のセマフォカウント値と、セマフォの待ち行列の先頭タスクIDが返されます。

２．５　メイルボックス

２．５．１　メイルボックスの概要

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ができます。

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

メイルボックスは、最大255個まで使用することができ、1から255までのメイルボックスIDと呼ばれる番号で識別します。

メイルボックスの登録は、ユーザシステムに必要なメイルボックスの総数をセットアップテーブルに設定することにより行ないます。

表２－９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２－９　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	メッセージを送信する
2	rcv_msg	メッセージの受信を待つ
3	prcv_msg	メッセージを受信する（ポーリング）
4	mbx_sts	メイルボックス状態を参照する

図２－１１にメイルボックスを利用した例を示します。

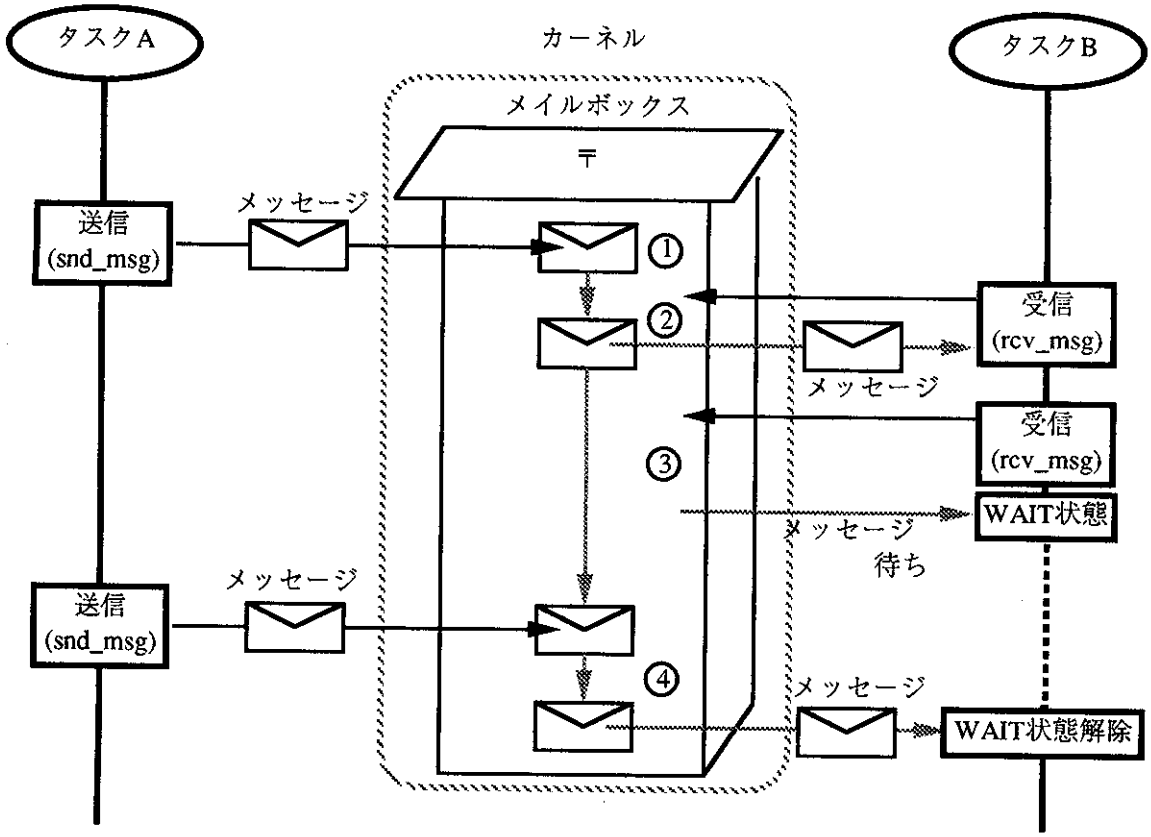


図２－１１　メイルボックスの使用例

（解説）

①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。

②タスクＢがメッセージの受信要求を行なうと、１通のメッセージがタスクＢに渡されます。

③タスクＢがさらにメッセージの受信要求を行なうと、メイルボックスにメッセージが存在しないため、タスクＢはメッセージの待ち状態となります。

④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。

２．５．２　メッセージ

図２－１２にメッセージのフォーマットを示します。メッセージは、RAM上に作成しなければなりません。

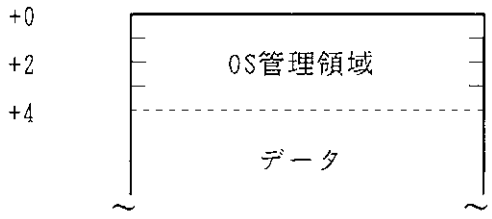


図２－１２　メッセージフォーマット

メッセージの先頭４バイトは、ＯＳがメッセージを管理するための領域（ＯＳ管理領域）です。ユーザは、メッセージの先頭４バイト以降をメッセージ領域として自由に使用することができます。

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

メッセージが送信されると、ＯＳ管理領域にメイルボックスにつながれていることを示すデータを設定しメッセージの待ち行列につなぎます。

タスクがメッセージを受信すると、メッセージをメッセージの待ち行列から外し、ＯＳ管理領域に０を設定してタスクに渡します。

同一メッセージを繰り返して送受信する場合は、最初のメッセージ送信時にＯＳ管理領域を０クリアするだけで送受信できます。

メッセージを送信後、受信される前にメッセージのＯＳ管理領域を破壊すると、メッセージの送受信が正常に行なわれません。注意してください。

メッセージの送受信は、データそのものが転送されるわけではなく、そのメッセージの先頭アドレスが渡されます。

このためメッセージを送信後、受信される前にメッセージを破壊すると、受信したタスクは破壊されたデータを読み込むことになります。

２．５．３　メッセージの送信

メッセージのメイルボックスへの送信は、snd_msg,isnd_msgシステムコールを使用します。（isnd_msgシステムコールは、非タスク部専用のsnd_msgシステムコールです。）

メッセージの送信は、以下のように行ないます。

(1) メッセージの送信

メッセージの先頭アドレスをパラメータとして、snd_msg システムコールを発行します。

(2) 待ちの解除

メッセージ到着を待つタスクが存在するかどうかを調べます。

メッセージ到着を待つタスクが存在する場合、メッセージ到着を待つタスクの待ち行列の先頭タスクにメッセージが渡され待ち行列からはずされます。

メッセージ到着を待つタスクが存在しない場合、送信したメッセージはメイルボックスのメッセージの待ち行列につながれます。

メッセージの待ち行列は、FIFO(First-In First-Out)で管理されます。

２．５．４　メッセージの受信

メイルボックスからメッセージを受信するには、rcv_msg システムコールを使用します。

メッセージの受信待ちを行なわない場合（ポーリング）は、prcv_msgシステムコールを使用します。

メッセージの受信は、以下のように行ないます。

(1) メッセージの受信要求

メッセージを受信するため、rcv_msg,prcv_msgシステムコールを発行します。

(2) メッセージの調査

rcv_msgシステムコールが発行されると、対象となるメイルボックスにメッセージが存在するか調べます。メッセージが存在する場合は、メッセージをメッセージの待ち行列からはずし、メッセージの先頭アドレスをリターンパラメータとして返します。

(3) メッセージがない場合の処理

rcv_msg システムコールを発行した場合、タスクをメイルボックスのメッセージ到着を待つタスクの待ち行列につなぎ、待ち状態に移行します。

prcv_msgシステムコールを発行した場合、エラーコードとしてポーリング失敗エラー(E_PLFAIL)が返されます。

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

２．５．５　メイルボックス状態の参照

現在のメイルボックスの状態を参照するには、mbx_sts システムコールを使用します。

mbx_sts システムコールは、リターンパラメータとしてメッセージの待ち行列の先頭メッセージアドレスと、メッセージ到着を待つタスクの待ち行列の先頭タスクIDが返されます。

２．６　割込み

２．６．１　割込みの概要

割込みを使用するには、割込みハンドラの作成とベクタテーブルへの登録が必要です。

割込みは、カーネルが介入せずに処理されるため、直接割込みハンドラへ制御が渡されることになり、迅速に処理されます。

また、タスク実行中に割込みレベルを変更することにより、タスクのCPU 占有時間を制御することもできます。

表２－１０に割込み管理のシステムコールを示します。

表２－１０　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する
2	chg_ims	割込みマスクレベルを変更する
3	ims_sts	割込みマスクレベルを参照する

ＨＩ８－３Ｈでは、該当するベクタテーブルに割込みハンドラの先頭アドレスを登録します。

割込みハンドラを定義すると、割込み発生時に直接割込みハンドラに制御が移ります。したがって、割込み要因に対しＨＩ８－３Ｈの介入を受けずに、応答可能です。

図２－１３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。

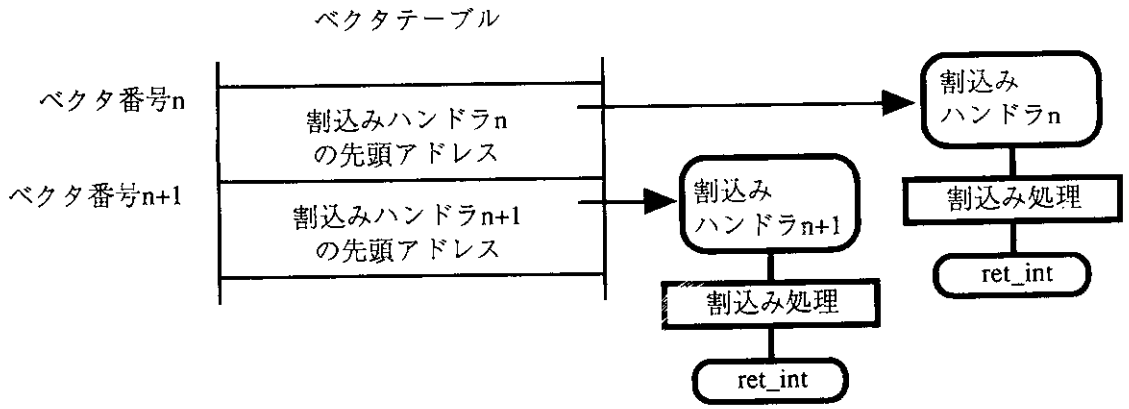


図２－１３　割込みベクタテーブルと割込みハンドラとの関連

図２－１４に割込みハンドラの制御移行を示します。

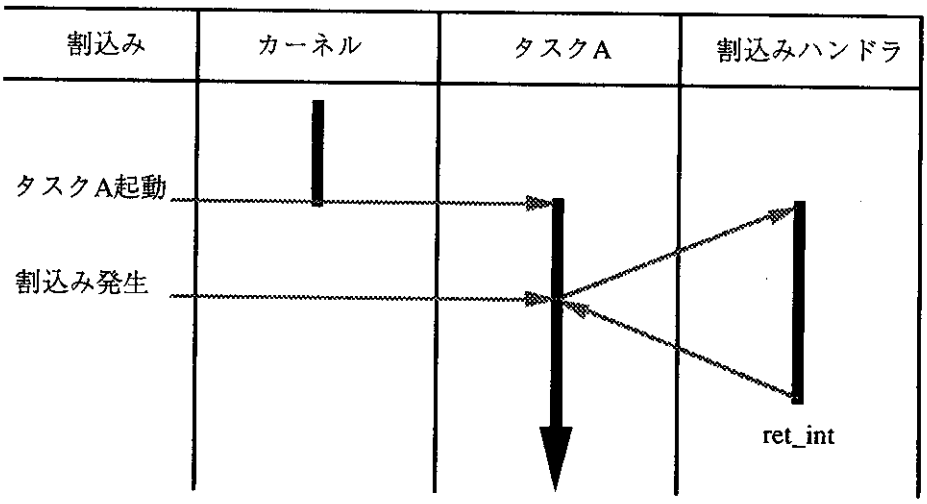


図２－１４　割込みハンドラの制御移行

①タスクＡの処理中、割込みが発生すると制御は割込みハンドラに移ります。
②非タスク部で起動された割込みハンドラの場合、処理が終了すると中断された処理の直後に制御が移ります。
③タスク部で起動された割込みハンドラの場合、処理が終了すると、現在実行可能状態で一番優先度の高いタスクに制御が移ります。

割込みハンドラ処理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(READY)状態となっても、割込みハンドラの処理が完了するまで該当タスクの実行は遅延されます。

２．６．２　割込みハンドラ

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るので、割込みハンドラは割込み発生時のレジスタ内容を保証しなければなりません。割込みハンドラは以下の手順で作成してください。

①割込みハンドラで使用するレジスタの内容退避
- スタックポインタの保存
- 割込みハンドラ専用スタック領域にスタックポインタを変更
  (割込みハンドラでいっさいレジスタを使用しない場合は不要)
- レジスタの内容保存

②割込み処理

③割込みハンドラで使用したレジスタの内容回復
- レジスタの内容復帰
- スタックポインタの変更
  (割込みハンドラでいっさいレジスタを使用しない場合は不要)

④ret_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割込みハンドラの場合)
または、RTE命令(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)

なお、Ｃ言語で割込みハンドラを記述する場合、「４．３．１　割込みハンドラの記述方法」を参照してください。割込みハンドラは、表２－１１の処理条件を参考に作成してください。

割込みハンドラを作成する場合、以下の注意事項を守ってください。

(1) 性能低下

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。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、システムのリアルタイム性に大きく影響を与えますので、注意して作成してください。

(2) ＮＭＩ（ノンマスカブルインタラプト）割込みハンドラ

ＮＭＩ割込みハンドラでは、システムコールを使用できません。

ＮＭＩ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せずにレジスタ（スタックポインタを含む）を割込み発生時に戻し、rte 命令を実行してください。

システムコールを発行した場合、H8-3Hシステムの動作は保証されません。注意してください。

(3) 割込みマスクレベルの保証

H8/300Hでは、割込みハンドラが起動されると、ＣＣＲのＩ，ＵＩビットともに１にセットされています。

優先度レベル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、優先度レベル１の割込みを受け付けられるように設定してください。ただし、ＣＣＲのＩビットをクリア（全ての割込みを受け付け可）しないでください。優先度レベル０の割込みハンドラでＣＣＲのＩビットをクリアした場合、HI8-3Hシステムの動作は保証されません。優先度レベル１の割込みハンドラでは、割込みマスクレベルを変更することができません。

(4)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

セットアップテーブルに設定する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は、システムコールを発行することができません。また、ＮＭＩ割込みハンドラと同様に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せずにレジスタ（スタックポインタを含む）割込み発生時に戻し、rte 命令を実行してください。

(5)パラメータチェック機能モジュール

パラメータチェック機能モジュールを組込んでいないHI8-3Hシステムで、タスク部専用システムコールを発行した場合、HI8-3Hシステムの動作は保証されません。

表２－１１に割込みハンドラの処理条件を示します。

表２－１１　割込みハンドラの処理条件

項番	項目	内容	備考
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	終了前に起動時の値に戻してください
3	SP(ER7)	発生元へ制御を戻すときは、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	非タスク部から発行できるシステムコール	NMI 割込みハンドラ，カーネル割込みマスクレベルより高い割込みハンドラでは、システムコールを発行することができません
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	スタックのサイズは、UNIX HI8-3H構築マニュアルまたはMS-DOS HI8-3H構築マニュアル「付録Ａ．メモリ容量の算出」を参照してください
6	終了	ret_int システムコールにより処理を終了します	NMI 割込みハンドラ，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は、rte命令で終了してください

【注】＊　同一割込みレベルの割込みハンドラは、スタックを共有することができます。

２．６．３　未定義割込み

割込みハンドラが定義されていない割込み（未定義割込み）が発生すると、カーネルはエラー情報をスタックに積み、H18-3Hのシステム異常終了ルーチンに移行します。

H18-3Hのシステム異常終了ルーチンについての詳しい説明は、「２．１０　H18-3Hのシステムの異常終了処理」を参照してください。

２．６．４　割込みマスクレベルの変更

chg_ims システムコールを用いて、タスク実行中に割込みをマスクすることができます。
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。
詳しい説明は、「２．２．８　タスク実行中の割込みマスク」を参照してください。

２．６．５　割込みマスクレベルの参照

現在の割込みマスクレベルを参照するには、ims_sts システムコールを使用します。
ims_sts システムコールは、リターンパラメータとして現在の割込みマスクレベルを返します。

２．７　メモリプール

２．７．１　メモリプールの概要

メモリプールを使用することにより、ユーザのメモリ空間を動的に管理することができます。メモリプールは、メモリブロックと呼ぶ複数の固定長メモリ領域から構成されています。
タスクは、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。

メモリプールは、最大255個まで使用することができ、1から255までのメモリプールIDと呼ぶ番号で識別します。
メモリプールの登録は、ユーザシステムに必要なメモリプールの数，メモリプール領域および各メモリプール毎にメモリブロック長，メモリブロック数をセットアップテーブルに設定することにより行ないます。

表２－１２にメモリプールを操作するシステムコールを示します。

表２－１２　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	メモリブロックを確保する
2	pget_blk	メモリブロックを得る（ポーリング）
3	rel_blk	メモリブロックを解放する
4	mpl_sts	メモリプール状態を参照する

図２－１５にメモリプールの操作を示します。

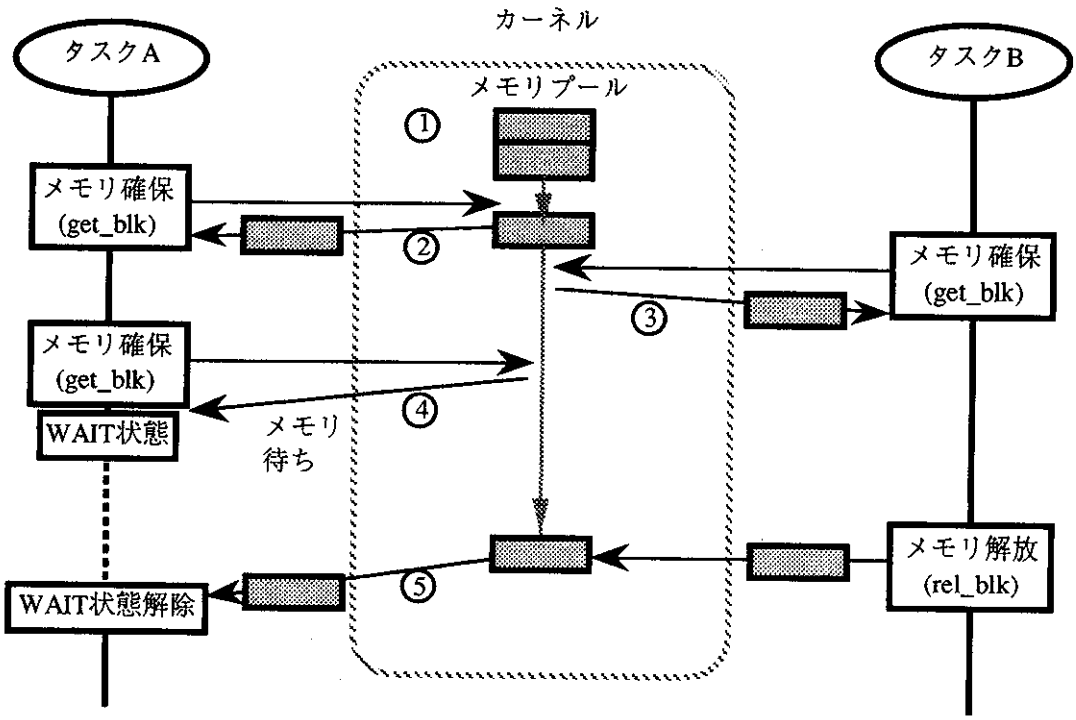


図２－１５　メモリプールの操作

（解説）

①メモリプールにメモリブロックが２個あります。

②タスクＡがメモリブロックを確保したので、メモリブロックが１個になります。

③タスクＢもメモリブロックを確保したので、メモリブロックが０個になります。

④タスクＡがさらにメモリブロックを確保しようとしたが、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。

⑤タスクＢがメモリブロックを解放したため、メモリブロックを確保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　メモリブロックの獲得

メモリブロックの獲得はget_blk システムコールを使用します。

メモリブロック獲得の待ちを行なわない場合（ポーリング）は、pget_blkシステムコールを使用します。

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

メモリブロックの獲得を以下のように行ないます。

(1) メモリブロックの獲得要求

メモリブロックを獲得するため、get_blkシステムコールを発行します。

(2) 空きメモリブロックの調査

get_blkシステムコールが発行されると、対象となるメモリプールに空きメモリブロックが存在するかどうか調べ、存在すればタスクにそのメモリブロックを割り付けます。

(3) 空きメモリブロックが存在しない場合の処理

get_blkシステムコールを発行した場合、タスクをメモリブロックの待ち行列につなぎ、待ち状態に移行します。

pget_blkシステムコールを発行した場合、エラーコードとしてポーリング失敗エラー(E_PLFAIL)が返されます。

メモリブロックの待ち行列は、FIFO(First-in First-out)で管理されます。

２．７．３　メモリブロックの解放

獲得したメモリブロックの解放は、rel_blk システムコールを使用します。

メモリブロックの解放は、以下のように行ないます。

(1) メモリブロックの解放

メモリブロックを解放するため、rel_blkシステムコールを発行します。

(2) 待ちの解除

メモリプールにメモリブロックの割付けを待つタスクの待ち行列が存在するかどうかを調べます。

待ちタスクが存在する場合、解放したメモリブロックを待ち行列の先頭タスクに割り付け、待ち行列からはずして実行可能状態に移行させます。

待ちタスクが存在しない場合、解放したメモリブロックを空きメモリブロックとします。

２．７．４　メモリプール状態の参照

現在のメモリプール状態を参照するには、mpl_sts システムコールを使用します。

mpl_sts システムコールは、リターンパラメータとして空きメモリブロックのブロック数とメモリブロックの待ち行列の先頭タスクIDが返されます。

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、時間の管理が行なえます。

以下に、その概要を示します。

(1) 時間の参照・設定

H18-3Hの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ないます。(wai_tskシステムコール)

表２－１３に時間管理のシステムコールを示します。

表２－１３　時間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	set_tim	時刻を設定する
2	get_tim	時刻を参照する

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、時間が管理されています。

ＯＳ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。ＯＳ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。

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

ハードウェアタイマの周期が10msecの場合の、ＯＳ内の時間（システムクロックの値）の10は、100msec を表わします。

図２－１６に、ハードウェアタイマとシステムクロックの関係を示します。

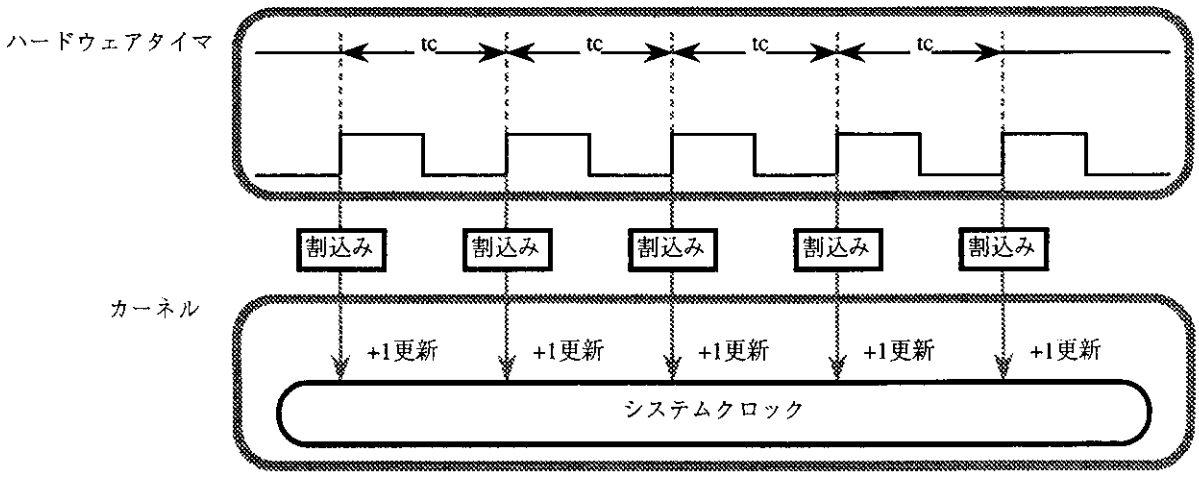


図２－１６　ハードウェアタイマとシステムクロック

２．８．３　時間の管理

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 $\times 10^{14}$であり、ハードウェアタイマの周期時間が1msecであれば約4000年に相当します。）

時間の設定はset_tim システムコール、時間の参照はget_tim システムコールを使用します。

システムコールのパラメータは、48ビットのシステムクロックのカウンタ値です。

図２－１７に時間の設定および参照を示します。

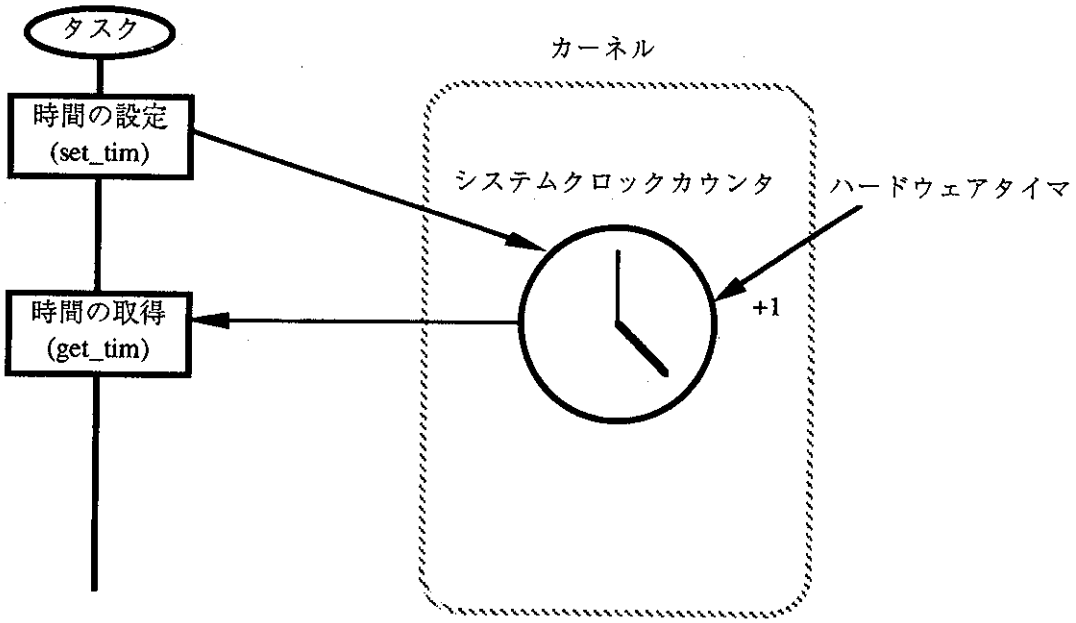


図２－１７　時間の設定および参照

２．８．４　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）

タイマドライバは、タイマ初期設定ルーチン、タイマ割込みハンドラ（タイマ割込みリセット処理、ＯＳのタイマ割込み処理）のモジュールにより構成されます。標準提供のタイマドライバのファイル名を次に示します。

・『h3huser.mar』：H8/300Hアドバンストモード用(sampleaﾃﾞｨﾚｸﾄﾘ)
・『h3husern.mar』：H8/300Hノーマルモード用(samplenﾃﾞｨﾚｸﾄﾘ)

表２－１４にタイマドライバの内容を示します。

表２－１４　タイマドライバ

項番	名称	内容	備考
1	タイマ初期設定ルーチン	ハードウェアタイマの初期設定を行ないます	
2	タイマ割込みハンドラ	・タイマ割込みリセット処理 タイマ割込みのクリア ・ＯＳのタイマ割込み処理 タイマカウントアップと時間によるタスクの実行制御	ベクタテーブルに登録します

図２－１８にタイマドライバの処理を示します。

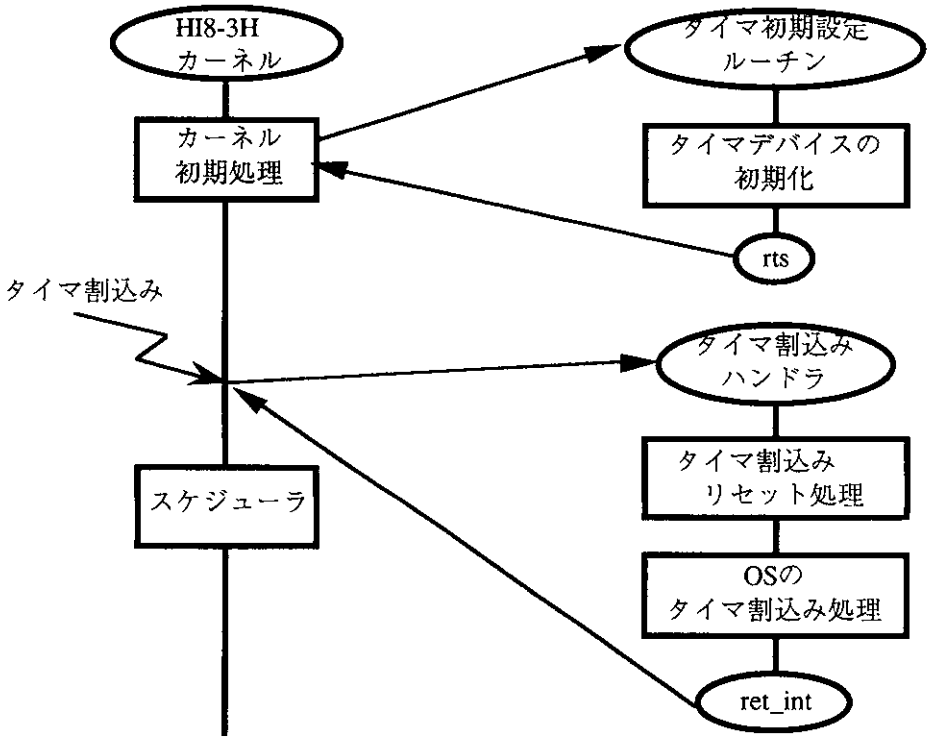


図２－１８　タイマドライバの処理

(1) タイマ初期設定ルーチン

タイマ初期設定ルーチンは、H18-3Hのシステム起動時に、ハードウェアタイマの初期設定を行ないます。

表２－１５にタイマ初期設定ルーチンの処理条件を示します。

表２－１５　タイマ初期設定ルーチンの処理条件

項番	項目	内容	備考
1	割込みマスク	割込みマスク状態で起動されます	タイマ初期設定ルーチン実行中は割込みマスク状態を解除しないでください
2	使用できるレジスタ	ER0～ER6が使用できます	
3	SP(ER7)	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	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	スタックのサイズは、UNIX H18-3H構築マニュアルまたはMS-DOS H18-3H構築マニュアル「付録Ａ．メモリ容量の算出」を参照してください
6	終了	RTS 命令により処理を終了します	

(2) タイマ割込みハンドラ（タイマ割込みリセット処理、ＯＳのタイマ割込み処理）

タイマ割込みハンドラは、ハードウェアタイマの割込みが発生すると、タイマ割込みリセット処理を行ない、ＯＳのタイマ割込み処理へジャンプします。

タイマ割込みリセット処理は、ハードウェアタイマの割込みをクリアします。

ＯＳのタイマ割込み処理は、タイマ割込みリセット処理終了後に実行され、システムクロックのカウントアップや時間によるタスクの実行制御を行ないます。

タイマ割込みリセット処理から非タスク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２－１６にタイマ割込みリセット処理の処理条件を示します。

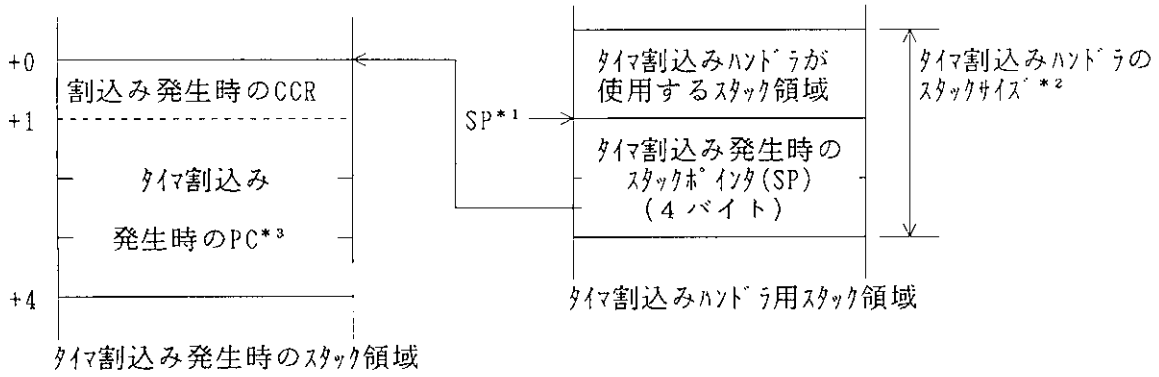
表２－１６　タイマ割込みリセット処理の処理条件

項番	項目	内容	備考
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	SP(ER7)	割込み発生元へ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	非タスク部から発行できるシステムコール	
5	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	スタックのサイズは、UNIX HI8-3H構築マニュアルまたはMS-DOS HI8-3H構築マニュアル「付録Ａ．メモリ容量の算出」を参照してください
6	終了	jmp　@H_timsys	ＯＳのタイマ割込み処理へジャンプしてください

図２－１９に、タイマ割込みハンドラ終了時（ＯＳのタイマ割込み処理へジャンプ時）のSPの状態を示します。

タイマ割込みハンドラ終了時には、タイマ割込み発生時のSP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SPとしてＯＳのタイマ割込み処理へジャンプしてください。

タイマ割込みハンドラの登録方法については、UNIX HI8-3H構築マニュアルまたはMS-DOS HI8-3H構築マニュアル「２．７　タイマドライバの登録」を参照してください。



【注】*1　ＯＳのタイマ割込み処理へジャンプする時のスタックポインタ(SP)です。
*2　スタックのサイズは、UNIX HI8-3H構築マニュアルまたはMS-DOS HI8-3H構築マニュアル「付録Ａ．メモリ容量の算出」を参照してください
*3　H8/300H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２－１９　タイマハンドラ終了時のSPの状態

(3) H8/3003またはH8/3042内蔵のITU以外のタイマを使用する場合

HI8-3Hでは、標準のハードウェアタイマとしてH8/300H 内蔵の16ビットインテグレーテッドタイマパルスユニット（以下、ITU と略す）を使用します。

ITU 以外のタイマを使用する場合は、タイマ初期設定ルーチンとタイマ割込みリセット処理を作成してください。

２．９　HI8-3Hのシステム起動処理

２．９．１　HI8-3Hのシステム起動処理の概要

HI8-3Hのシステムが起動されると、以下のHI8-3Hのシステム起動処理を行ないます。

図２－２０にHI8-3Hのシステム起動時の処理概要を示します。

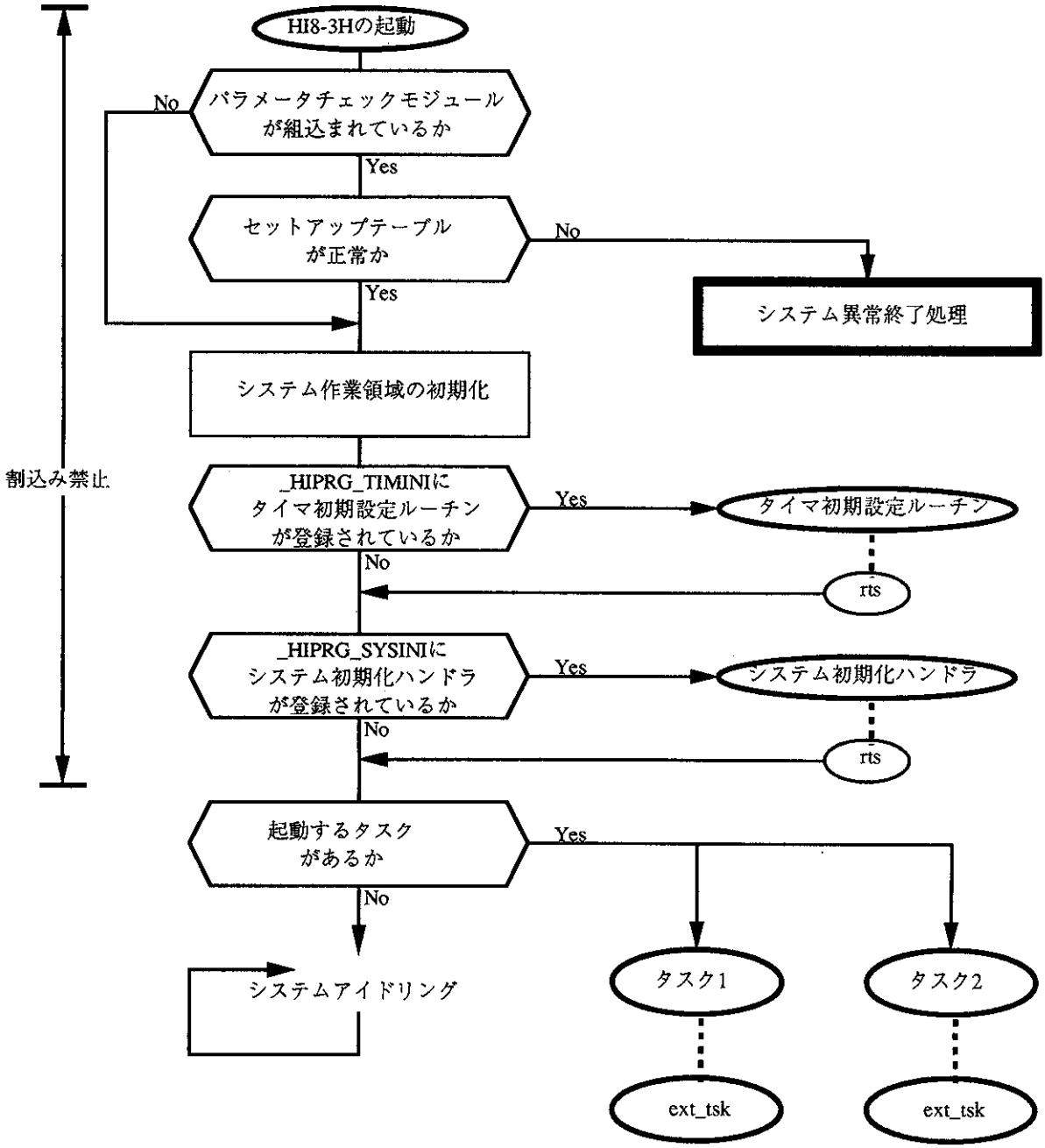


図２－２０　HI8-3Hのシステム起動時の処理概要

以下にHI8-3Hのシステム起動時の各処理について説明します。

HI8-3Hはシステム起動後、タスク起動まで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で割込みを禁止しています。

(1) セットアップ情報のチェック

パラメータチェックモジュールが組み込まれている場合、セットアップ情報のチェックを行ないます。

チェック結果が正常であればHI8-3Hのシステム環境を初期化し、正常でなければHI8-3Hのシステム異常終了ルーチンに移行します。

(2) HI8-3Hのシステム作業領域の初期化

HI8-3Hのシステムが使用する領域の初期化を行ないます。

(3) タイマ初期設定ルーチンの実行

タイマ初期化ルーチンのラベル『_HIPRG_TIMINI 』に設定されている値を、タイマ初期設定ルーチンの先頭アドレスとして実行します。

『_HIPRG_TIMINI 』に0が設定されていた場合、処理は行なわれません。

(4) システム初期化ハンドラの実行

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。

『_HIPRG_SYSINI 』に0が設定されていた場合、処理は行なわれません。

(5) タスク起動

実行可能状態で登録されているタスクがある場合、スケジューリングを行ないます。

実行可能状態で登録されているタスクがない場合、アイドル状態に入り割込みが発生するのを待ちます。

２．９．２　セットアップ情報のチェック

パラメータチェックモジュールが組込まれている場合、セットアップ情報のチェックを行ない、結果が正常でなければシステム異常終了ルーチンに移行します。

詳しくは「２．１１　HI8-3Hのシステムの異常終了処理」を参照してください。

２．９．３　システム初期化ハンドラ

システム初期化ハンドラとは、HI8-3Hの初期処理を行なった後、タスクを起動する前に実行されるプログラムです。標準提供のシステム初期化ハンドラのファイル名を次に示します。

・『h3huser.mar』：H8/300Hアドバンストモード用(sampleaﾃﾞｨﾚｸﾄﾘ)
・『h3husern.mar』：H8/300Hノーマルモード用(samplenﾃﾞｨﾚｸﾄﾘ)

システム初期化ハンドラを使用すると、タスク起動前に任意の初期処理を行なうことができます。

表２－１７にシステム初期化ハンドラの処理条件を示します。

表２－１７　システム初期化ハンドラの処理条件

項番	項目	内容	備考
1	割込みマスク	割込みマスク状態で起動されます	システム初期化ハンドラ実行中は割込みマスクを解除しないでください
2	使用できるレジスタ	ER0～ER6が使用できます	終了前に起動時の値に戻してください
3	SP(ER7)	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	非タスク部から発行できるシステムコール（ret_int システムコールは除く）	
5	使用できるスタック領域	システム構築時に確保し、システム初期化ハンドラ起動時にスタックを切り替えてください	スタックのサイズは、UNIX HI8-3H構築マニュアルまたはMS-DOS HI8-3H構築マニュアル「付録Ａ．メモリ容量の算出」を参照してください
6	終了	RTS 命令により処理を終了します	

２．１０　ＣＰＵの初期化

２．１０．１　ＣＰＵ初期化の概要

H8/300Hでは、構築するシステムによってCPUを初期化しなければなりません。

HI8-3Hのシステム初期化ハンドラでCPUを初期化してもかまいませんが、HI8-3Hが動作する前に行なわければならないCPUやハードウェアの初期化は、その処理を行なうプログラムを作成しなければなりません。このプログラムをCPU初期化ルーチンといいます。

CPU初期化ルーチンは、リセット割込みでHI8-3Hが動作する前に起動されます。

CPU初期化ルーチンは、ユーザシステムに必要なCPUやハードウェアの初期化を行ない、HI8-3Hのシステム起動処理に制御を渡します。

図２－２１にCPU初期化ルーチンの処理を示します。

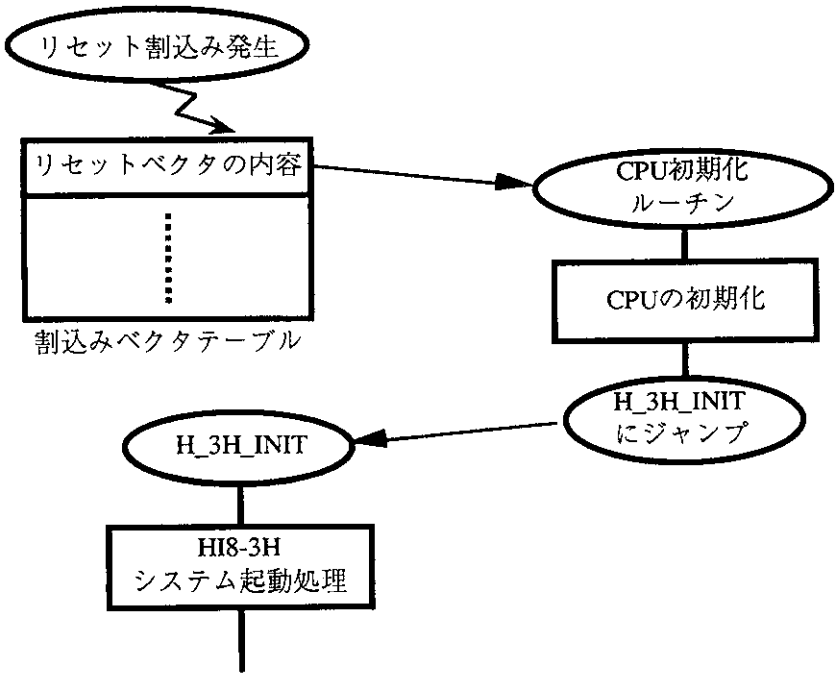


図２－２１　CPU初期化ルーチンの処理

２．１０．２　ＣＰＵ初期化ルーチンの作成

CPU初期化ルーチンは、使用するCPUによって行なう処理が違うので構築するシステムに合せて初期化処理を作成してください。標準提供のCPU初期化ルーチンのファイル名を次に示します。

・『cpuini.mar』：H8/300Hアドバンストモード用(sampleaﾃﾞｨﾚｸﾄﾘ)
・『cpuinin.mar』：H8/300Hノーマルモード用(samplenﾃﾞｨﾚｸﾄﾘ)

初期処理の内容についての詳細は、該当するCPUのハードウェアマニュアルを参照してください。

CPU初期化ルーチンを作成するときの留意点を以下に示します。

(1) 割込みマスク

リセット割込み後、CCRのIビットがセットされ割込みマスクされています。

(2) 初期化処理

HI8-3Hでは、CCRのUIビットをユーザ・ビットとして使用できません。

必ず、CCRのUIビット（ビット6）を割込みマスクビットとして使用できるようにSYSCR（システム・コントロール・レジスタ）を初期化してください。

(3) 終了処理

CPU初期化の終了後HI8-3Hに制御を渡すため、HI8-3Hのシステム起動処理の実行開始アドレス『H_3H_INIT』にジャンプしてください。

(4) HI8-3Hの機能の使用

CPU初期化ルーチンが処理を行なっているとき、HI8-3Hは起動していないためHI8-3Hの機能は使用できません。

(5) スタックポインタの初期化

必要に応じてスタックポインタを初期化してください。

標準提供のCPU初期化ルーチンでは、スタックポインタが初期化されていません。

スタックポインタの初期化前にNMI割込みが発生すると動作が保証できませんのでスタックポインタの初期化または、NMI割込みが発生しないようにしてください。

図２－２２にCPU初期化ルーチンのコーディング例(『cpuini.mar』:H8/300Hｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用)を示します。

```
;3333333333333333333333333333333333333333333333333333333333333333333333333333333333
;HHH                                                                             HHH
;HHH     H18-3H Version (uITRON V2.01/02)                                        HHH
;HHH     H18-3H user/system application file Ver2.00                             HHH
;HHH                                                                             HHH
;HHH     Copyright (c)  Hitachi, Ltd. 1993.                                      HHH
;HHH     Licensed Material of Hitachi, Ltd.                                      HHH
;HHH                                                                             HHH
;3333333333333333333333333333333333333333333333333333333333333333333333333333333333
        .program        cpuini
        .heading        "### cpuini.mar : H8/3003 initialize module ###"
        .section        h3huser,code,align=2
;
        .export         H_3H_CPUINI
        .import         H_3H_INIT
;
;*************************************************************************
;*specifications ;                                                       *
;*name      = H_3H_CPUINI : H8/3003 initialize module                    *
;*function  =                                                            *
;*notes     =                                                            *
;*date      = 93/07/06                                                   *
;*author    = Hitachi, Ltd.                                              *
;*history   = V1.00/V2.00                                                *
;*attribute = public                                                     *
;*class     = system                                                     *
;*linkage   =                                                            *
;*input     = none                                                       *
;*output    = none                                                       *
;*end of specifications ;                                                *
;*************************************************************************
;
        .radix  d                       ;:xxxxx -> d'xxxxx
;
;###### interrupt register address #####;:
;
SYSCR:          .equ    h'fffff2        ;:system control register
;
;###### system control register ########;:SYSCR
;
;       +-+-+"SYSCR"+-+-+
;       |7|6|5|4|3|2|1|0|
;       +-+-+-+-+-+-+-+-+
;        ^ ^ ^ ^ ^ ^ ^ ^
;        | | | | | |   +--- RAME
;        | | | | | + ------ NMIEG
;        | | | | +--------- UE
;        | | | +----------- STS0
;        | | +------------- STS1
;        | +--------------- STS2
;        +----------------- SSBY
;
RANE:           .equ    b'00000001      ;:RAM enable
NMIEG:          .equ    b'00000100      ;:NMI edge select
UE:             .equ    b'00001000      ;:User Enable
STS0:           .equ    b'00010000      ;:Standby Timer Select 0
STS1:           .equ    b'00100000      ;:Standby Timer Select 1
STS2:           .equ    b'01000000      ;:Standby Timer Select 2
SSBY:           .equ    b'10000000      ;:Software Standby
;
H_3H_CPUINI:
        mov.b   @SYSCR:8,r0L            ;:set SYSCR
        and.b   #low ~(UE):8,r0L        ;:
        mov.b   r0L,@SYSCR:8            ;:
;
        jmp     @H_3H_INIT              ;:goto H18-3H initialize module
;
;****************************************;
        .end; of CPUINI.MAR
```

図２－２２　CPU初期化ルーチンのコーディング例(『cpuini.mar』:H8/300Hｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用)

２．１１　HI8-3Hのシステムの異常終了処理

HI8-3Hは、システムに異常が発生した場合、システム異常終了ルーチンを起動します。

HI8-3Hのシステム異常終了ルーチンには、異常終了原因に応じたプログラムを記述します。

（標準提供のHI8-3Hのシステム異常終了ルーチンは、無限ループに入ります。）

標準提供のシステム異常終了ルーチンのファイル名を次に示します。

・『h3huser.mar』：H8/300Hアドバンストモード用(sampleaﾃﾞｨﾚｸﾄﾘ)
・『h3husern.mar』：H8/300Hノーマルモード用(samplenﾃﾞｨﾚｸﾄﾘ)

この処理中、HI8-3Hの動作は保証できませんので注意してください。

HI8-3Hのシステム起動時と通常動作時のシステム異常終了の原因を示します。

(1)システム起動時のシステム異常終了

HI8-3Hのパラメータチェック有りの機能モジュールライブラリを組み込んだシステムは、次の内容をチェックします。不正が発生した場合、スタックにエラー情報を積んでHI8-3Hのシステム異常終了ルーチンに移行します。

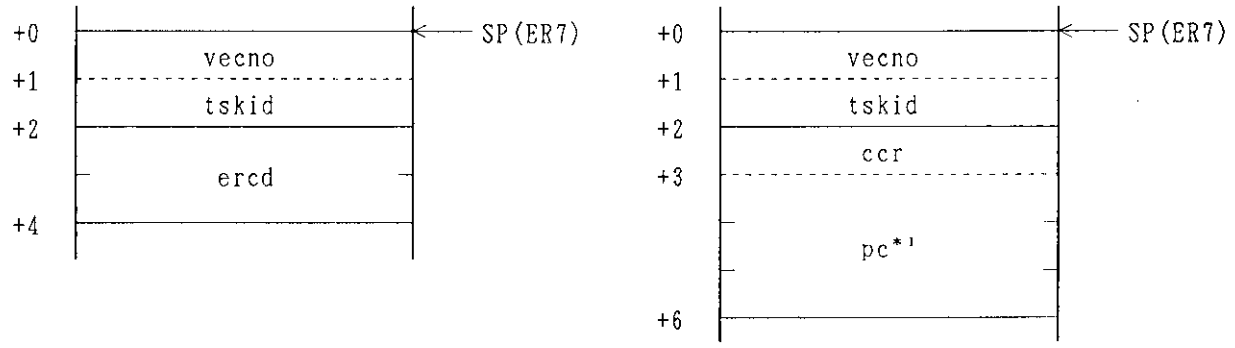
・セットアップ情報のエラー
・タイマ未サポート

(2)通常動作時のシステム異常終了

通常動作中に次の異常が起った場合、スタックにエラー情報を積んでHI8-3Hのシステム異常終了ルーチンに移行します。

・非タスク部からext_tskシステムコール発行
・タスク部からret_intシステムコール発行
・未定義割込みの発生

図２－２３にシステム異常終了時のスタック内のエラー情報を示します。



【注】*1　H8/300Hノーマルモードでは、下位１６ビットが有効になります。

図２－２３　スタック内のエラー情報

図２－２３のスタック状態Ａは、セットアップ情報のエラー，タイマ未サポート，非タスク部からext_tskシステムコール発行，タスク部からret_intシステムコール発行の異常が発生したときのスタック状態です。スタック状態Ｂは、未定義割込み発生の異常が発生したときのスタック状態です。

表２－１８に異常終了となる原因とスタックに積まれる値を示します。

表２－１８　異常終了となる原因

項番	原因	vecno	tskid	ercd/ccr,pc
1	セットアップ情報のエラー	H'00	H'00	H'0000～H'0FFF
2	タイマ未サポート	H'00	H'00	H'F9ED
3	非タスク部からのext_tsk システムコール発行	H'00	H'00	H'FFEB
4	タスク部からのret_int システムコール発行	H'00	tskid	H'FFBB
5	未定義割込みの発生	割込み ベクタ番号	tskid*	発生時のCCR,PC

【注】*　タスク部で発生した場合はtskid が設定され、非タスク部で発生した場合は０が設定されます。

表２－１９にセットアップ情報不正内容とercdを示します。

表２－１９　セットアップ情報不正内容一覧

項番	不正項目		ercd
1	アドレス不正 （０か奇数）	OS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
		システム作業領域の先頭アドレス (セクション名『hi8_3h_ram』)が０または奇数	H'0103
		TIMCB 領域(_HI_TIMCB)が０または奇数	H'0104
		TCB 領域(_HI_TCB)が０または奇数	H'0105
		FLGCB 領域(_HI_FLGCB)が０または奇数	H'0106
		SEMCB 領域(_HI_SEMCB)が０または奇数	H'0107
		MBXCB 領域(_HI_MBXCB)が０または奇数	H'0108
		MPLCB 領域(_HI_MPLCB)が０または奇数	H'0109
		トレーススタックポインタ(_HI_TRC_SP)が０または奇数	H'010A
		トレース管理領域(TBACB)が０または奇数	H'010B
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）	カーネル割込みマスクレベル(IMASK)が４以上	H'0301
		優先度定義数(MAXPRI)が32以上	H'0302
		タスク定義数(TSKCNT)が256 以上	H'0303
		イベントフラグ定義数(FLGCNT)が256 以上	H'0304
		セマフォ定義数(SEMCNT)が256 以上	H'0305
		メイルボックス定義数(MBXCNT)が256 以上	H'0306
		メモリプール定義数(MPLCNT)が256 以上	H'0307
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)が０または奇数	H'0401
		メモリプール定義テーブル(_HI_MPLDT)が０または奇数	H'0402
		未定義割込みハンドラ(_HI_ILT)が０または奇数	H'0403
		トレースバッファ情報テーブル(INITRC)が０または奇数	H'0404
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(BLKLEN)が０または奇数 または65532バイト以上	H'0504
		メモリプールアドレス(MPLADR)が０または奇数	H'0505
		トレースバッファアドレス(TRACE BUFFER ADDRESS)が０ または奇数	H'0506

２．１２　トレース機能

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

２．１２．１　トレース取得の概要

図２－２４に、トレース取得時の動作概要を示します。

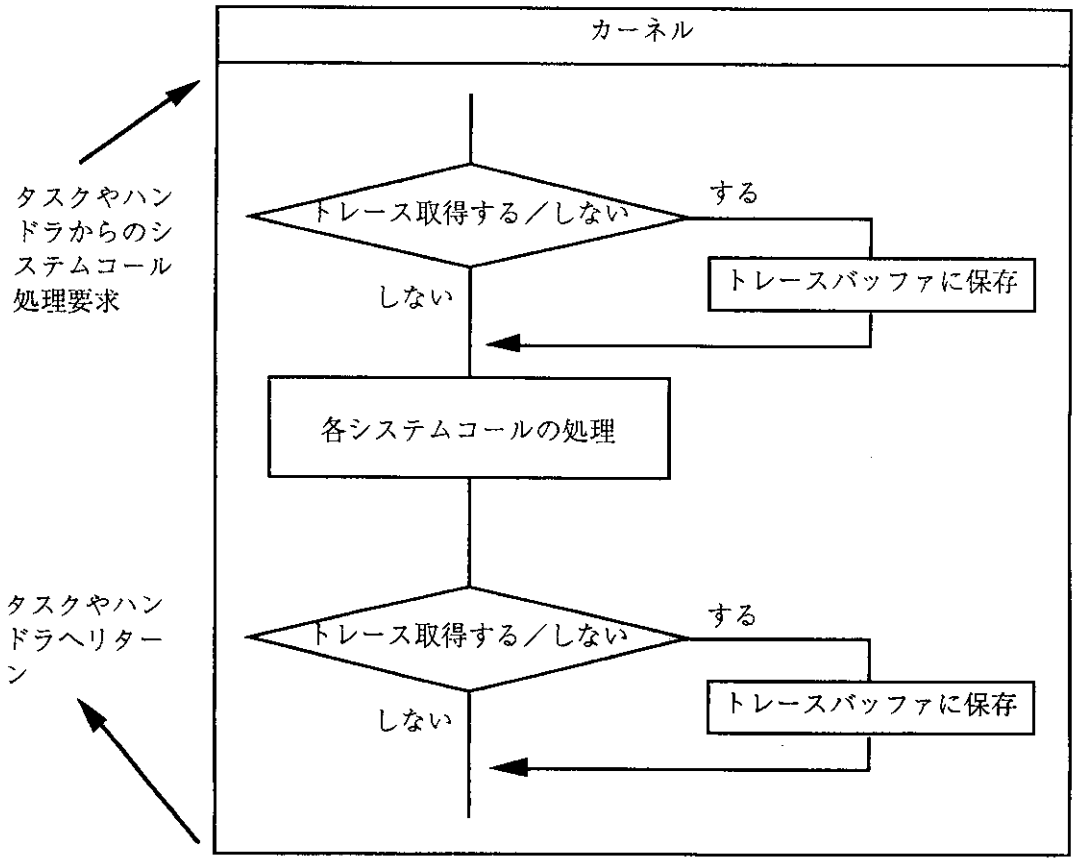


図２－２４　トレース取得時の動作概要

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

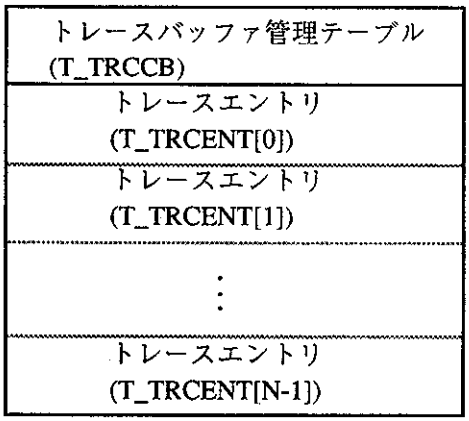
このトレース取得される情報を、「トレース情報」と呼びます。

トレース機能を使用する場合、システム構築時にセットアップテーブルで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ないます。トレース機能を登録すると、システム起動後、システム初期化ハンドラから、トレース情報の取得が開始されます。トレースバッファはリングバッファ構造になっており、古い情報は順次新しい情報に書き換えられます。

システムへのトレース機能の登録については、『UNIX HI8-3H構築マニュアル』または『MS-DOS HI8-3H構築マニュアル』を参照してください。

２．１２．２　トレースバッファの構造

図２－２５に、トレースバッファの構造を示します。



【注】 Nはトレース情報取得数

図２－２５　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つのトレース情報につき、1つのトレースエントリ領域が使用されます。

(1)トレースバッファ管理テーブル(T_TRCCB)

トレースバッファの管理を行なうテーブルです。トレース取得時に、カーネルがこの領域を使用します。

図２－２６に、トレースバッファ管理テーブルの構造を示します。

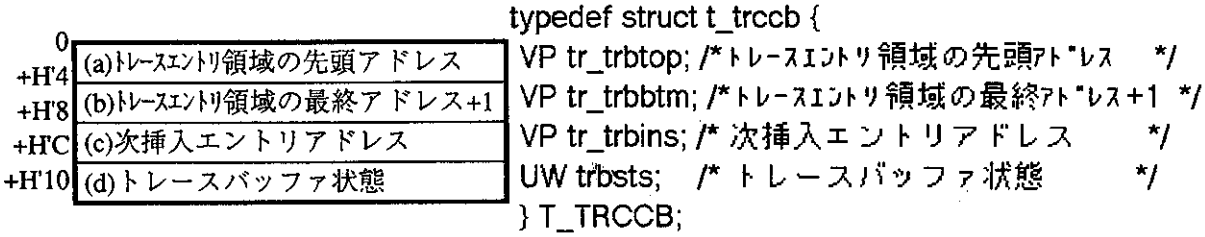


図２－２６　トレースバッファ管理テーブルの構造

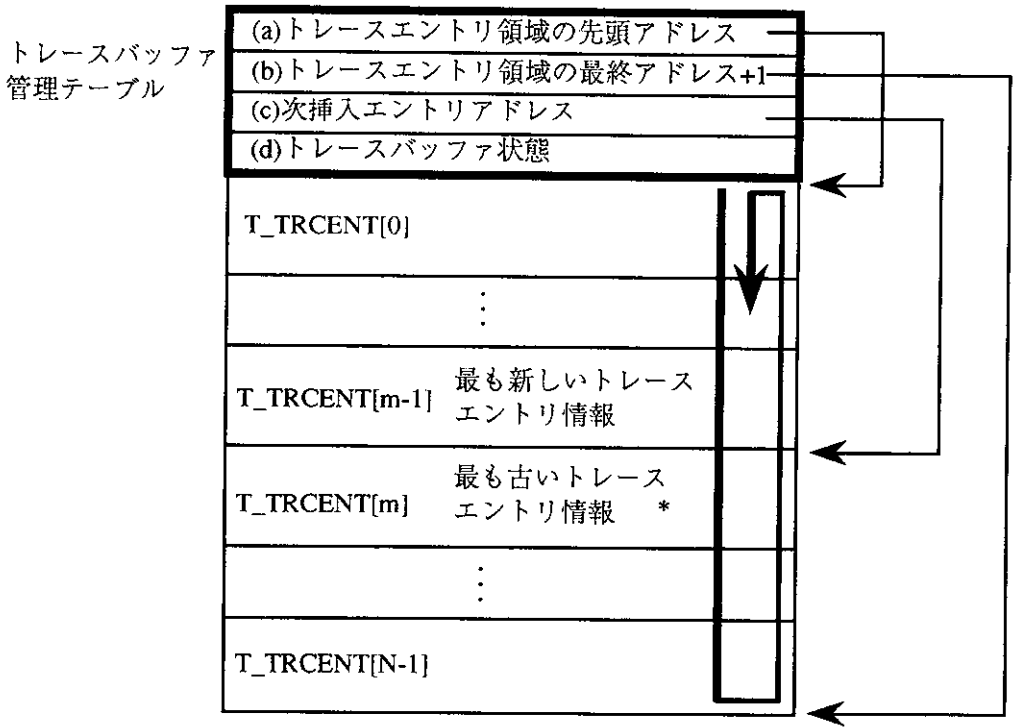
トレース情報の格納可能な領域は、(a)～(b)で保持しています。これらは、システム起動時にセットアップテーブルの設定内容にしたがって初期化されます。(c)の次挿入エントリアドレスは、次に取得したトレース情報が格納されるエントリのアドレスです。

(d)のトレースバッファ状態は、以下の意味を持ちます。また、0,1以外のビットは不定です。

ビット0 ：リングバッファフラグ
0：ﾄﾚｰｽﾊﾞｯﾌｧへの書込みは1周していない／1：ﾄﾚｰｽﾊﾞｯﾌｧへの書込みは1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)次挿入ｴﾝﾄﾘｱﾄﾞ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。

図２－２７に、トレースバッファの管理の様子を示します。



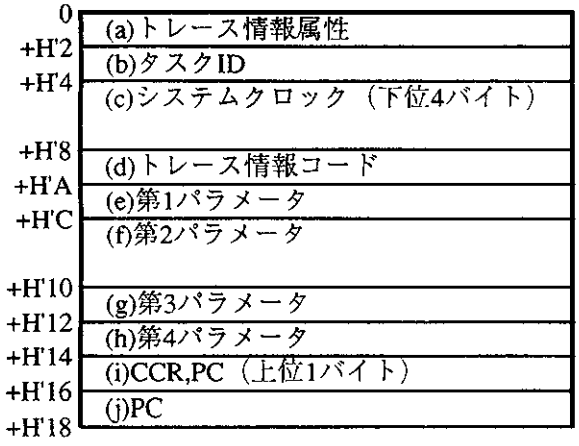
【注】＊ トレースバッファ状態のビット1が"1"の場合は、不定データとなります。

図２－２７　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つのトレース情報につき、1つのトレースエントリが使用されます。

図２－２８に、トレースエントリの構造を示します。




```
typedef struct t_trcent {
 UH te_attr;  /* トレース情報属性          */
 ID te_tskid; /* タスクID                  */
 UW te_ltime; /* システムクロック値(下位) */

 H  te_event; /* トレース情報コード(r0)   */
 VH te_r1;    /* 第1パラメータ(r1)        */
 VW te_er2;   /* 第2パラメータ(er2)       */

 VH te_r3;    /* 第3パラメータ(r3)        */
 VH te_r4;    /* 第4パラメータ(r4)        */
 UH te_ccr_pc;    /* CCR,PC(上位)         */
 UH te_pc;    /* PC                       */
} T_TRCENT;
```

図２－２８　トレースエントリの構造

(a)のトレース情報属性は、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。

以下に示す4つの属性があります。

- 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を意味し、トレースエントリの内容はシステムコール発行時の情報となります。

RTN属性は、カーネルからタスクに戻ることを意味し、トレースエントリの内容はシステムコール終了時の情報またはタスク起動時の情報となります。なお、コンテキストエラー(E_CTX)を発生させたシステムコールについては、SVC属性、RTN属性ともに取得されません。

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。

①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のタスクに戻る場合
②割込みハンドラがタスク切換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を含む）を発行しないままret_intシステムコールを発行し、割込み発生前のタスクに戻る場合
③割込みハンドラでタスク切換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を含む）を発行した後にret_intシステムコールを発行し、その結果割込み発生前の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のタスクに再度スケジューリングされた場合

CONT属性は、③の場合に取得されます。①,②の場合は、何の情報も取得されません。

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トレース情報属性以外のデータは、属性によってその意味が異なります。表２－２０に、トレースエントリ内容の意味を示します。

表２－２０　トレースエントリ内容の意味

te_attr	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)
te_tskid	ｼｽﾃﾑｺｰﾙを発行したﾀｽｸID。非ﾀｽｸ部の場合は0。	ｶｰﾈﾙからの戻り先となるﾀｽｸID。非ﾀｽｸ部の場合は0。	割込み発生点から実行再開するﾀｽｸID。非ﾀｽｸ部(0)の場合はありえません。	不定
te_ltime	トレース取得時のシステムクロックの下位4バイト			
te_event	発行したｼｽﾃﾑｺｰﾙの機能ｺｰﾄﾞ* (ｼｽﾃﾑｺｰﾙ発行時のR0)	ｼｽﾃﾑｺｰﾙのｴﾗｰｺｰﾄﾞ。ただし、H'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_r4	ｼｽﾃﾑｺｰﾙのﾊﾟﾗﾒｰﾀ (ｼｽﾃﾑｺｰﾙ発行時のR1,ER2,R3,R4)	ｼｽﾃﾑｺｰﾙのﾘﾀｰﾝﾊﾟﾗﾒｰﾀ。ただし、ﾀｽｸ起動時は不定。 (ｼｽﾃﾑｺｰﾙﾘﾀｰﾝ時のR1,ER2,R3,R4)	不定	不定
te_pc	ｼｽﾃﾑｺｰﾙ発行ｱﾄﾞﾚｽ	ﾕｰｻﾞﾌﾟﾛｸﾞﾗﾑへの戻り先ｱﾄﾞﾚｽ。ﾀｽｸ起動時は、ﾀｽｸｽﾀｰﾄｱﾄﾞﾚｽとなり、ﾀｽｸ起動時以外は以前のｼｽﾃﾑｺｰﾙ発行ｱﾄﾞﾚｽとなります。	不定	不定
te_ccr	ｼｽﾃﾑｺｰﾙ発行時のCCR	ﾕｰｻﾞﾌﾟﾛｸﾞﾗﾑに戻ったときのCCR	不定	不定

【注】*　ret_intｼｽﾃﾑｺｰﾙは取得されません

２．１２．３　トレース情報の解析例

取得されたトレース情報が、表２－２１であったとします。

表２－２１　トレース情報の例

No.		te_attr	te_tskid	te_ltime	te_event	te_r1	te_pc	te_ccr
-12		H'0001	H'0005	H'00001234	H'ffe9	H'0003	H'003018	H'00
	古	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
		RTN	tskid=3		ﾀｽｸ起動			
-10		H'0001	H'0003	H'00001234	H'ffda	H'xxxx	H'003810	H'00
		SVC	tskid=3		slp_tsk			
-9		H'0002	H'0005	H'00001234	H'0000	H'0003	H'003018	H'00
		RTN	tskid=5		E_OK			
-8		H'0001	H'0000	H'00001234	H'ff27	H'0003	H'007340	H'c0
		SVC	非ﾀｽｸ		iwup_tsk	3番のﾀｽｸを起床		ﾌﾟﾗｲｵﾘﾃｨﾚﾍﾞﾙ=1
-7		H'0002	H'0000	H'00001234	H'0000	H'0003	H'007340	H'c0
		RTN	非ﾀｽｸ		E_OK			ﾌﾟﾗｲｵﾘﾃｨﾚﾍﾞﾙ=1
-6		H'0002	H'0003	H'00001234	H'0000	H'xxxx	H'003810	H'00
		RTN	tskid=3		E_OK			
-5		H'0001	H'0003	H'00001234	H'ffe1	H'0005	H'003840	H'00
		SVC	tskid=3		rel_wai	5番を強制待ち解除		
-4		H'0002	H'0003	H'00001234	H'f7c2	H'0005	H'003840	H'00
		RTN	tskid=3		E_NOWAI			
-3		H'0001	H'0003	H'00001234	H'ffeb	H'xxxx	H'00385c	H'00
		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	H'xxxx	H'003054	H'00
		SVC	tskid=5		ext_tsk			
0	新	H'0004	（不定）	H'00001234	（不定データが入っている）			
		IDLE						

【注】・説明を簡単にするため、te_er2, te_r3, te_r4のデータは省略しています。
・上段がトレース情報のデータ、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、トレース情報として取得されるわけではありません。

以下に、表２－２１の説明を示します。

(1)トレース情報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、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アドレスはte_pc=H'003018より、H'3018番地であることがわかります。

(2)トレース情報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。

トレース情報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)トレース情報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)トレース情報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。トレース情報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、トレース情報No.-12のところでsta_tskシステムコールの発行後、この時点まで実行していない（トレース情報が取得されていない）ので、te_eventはトレース情報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)トレース情報No.-8

te_attr=H'0001(SVC属性),te_tskid=0,te_event=H'ff27,te_r1=3より、非タスク部からwup_tskシステムコール(tskid=3)が発行されたことがわかります。非タスク部には、割込みハンドラ、システム初期化ハンドラなどがありますが、te_ccr=H'c0よりﾌﾟﾗｲｵﾘﾃｨﾚﾍﾞﾙ1の割込みハンドラであると推測できます。割込みハンドラの場合、トレース情報No.-9からこのトレース情報までの間にその割込みが発生したことになります。

(6)トレース情報No.-7

te_attr=H'0002(RTN属性),te_tskid=0より、非タスク部のシステムコールリターンの情報であることがわかります。直前の非タスク部からのシステムコール発行は、トレース情報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)トレース情報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。直前のトレース情報No.-7は非タスク部の情報だったので、NO.-7とNO.-6のトレース情報の間で、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、本トレース情報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番のタスクはトレース情報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)トレース情報No.-5

te_attr=H'0001(SVC属性),te_tskid=3,te_event=H'ffe1,te_r1=H'0005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)トレース情報No.-4

te_attr=H'0002(RTN属性),te_tskid=3,te_event=H'f7c2,より、3番のタスクが発行したトレース情報No.-5のrel_waiシステムコールが、エラー終了（エラーコードE_NOWAI）したことがわかります。

(10)トレース情報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)トレース情報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（トレース情報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。

トレース情報をさかのぼると、5番のタスクに関する情報は、No.-9のトレース情報からこの時点までありません。すなわち、No.-9とNo.-8のトレース情報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)トレース情報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)トレース情報No.0

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

このような解析を元に、表２－２１のトレース情報から実際のプログラムの動きを図示すると、図２－２９のようになります。理解を深めるための参考にしてください。

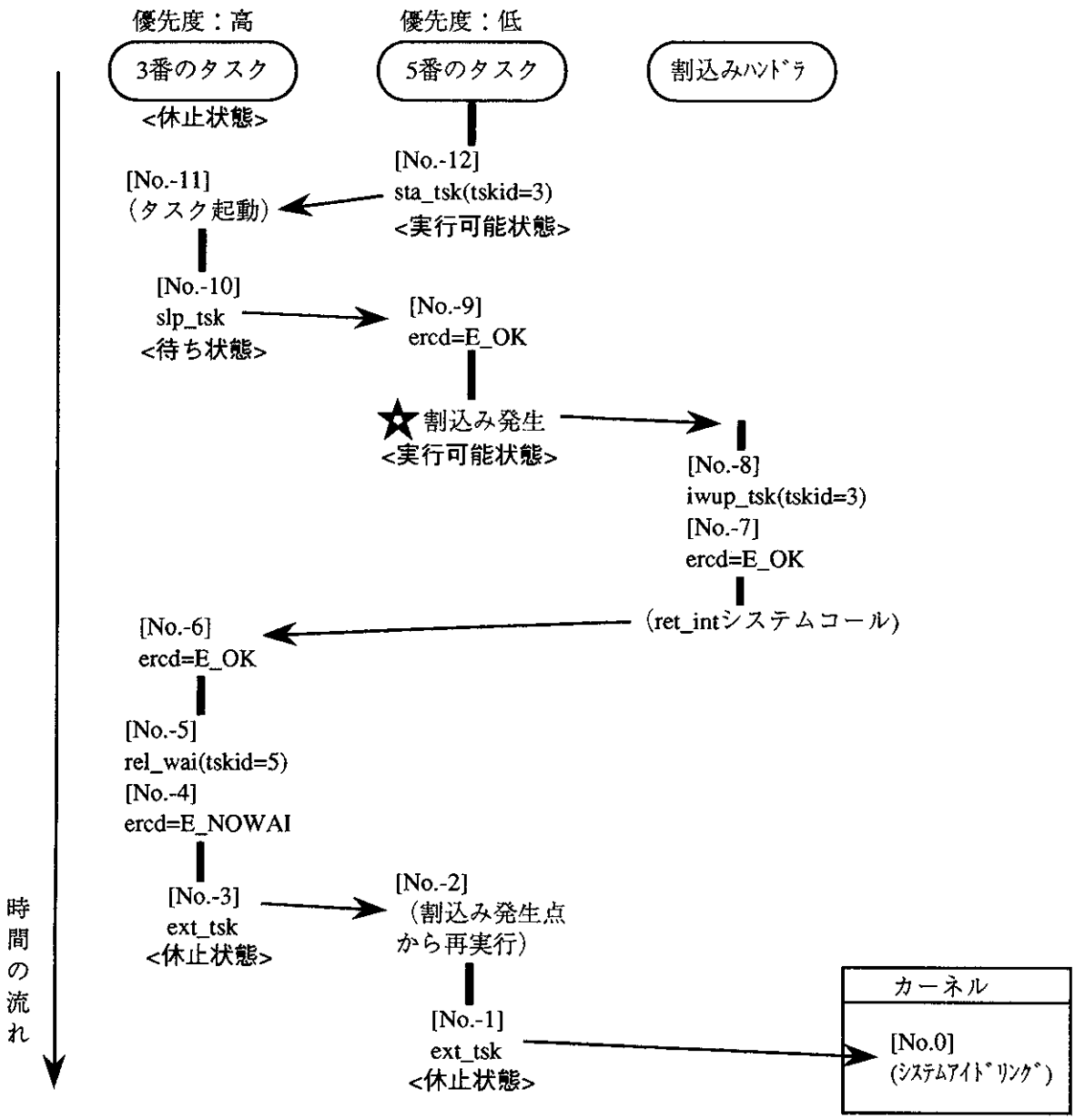


図２－２９　トレース解析結果の図示例

２．１２．４　トレース機能の使用上の注意

(1)カーネル性能の悪化

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

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

(2)トレースバッファの書込み

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

(3)コンテキストエラーの扱い

コンテキストエラー(E_CTX)が発生したとき、トレース情報は取得されません。

(4)ＲＴＮ属性のトレース情報

H8/300Hノーマルモードのとき、ＲＴＮ属性のトレース情報のE2レジスタの値は不定値になります。

3．システムコール

3．1　概　要

H18-3Hはμ ITRON 仕様のレベル３に準拠し、全部で43個のシステムコールをサポートしています。
H18-3Hのシステムコールは、表３－１のように分類されます。

表３－１　システムコールの分類

項番	分類	機能
1	タスク管理システムコール	タスクの実行・終了を行ないます
2	タスク付属同期管理システムコール	タスクの実行の中断・再開を行ないます
3	同期／通信管理システムコール	イベントフラグ・セマフォ・メイルボックスの管理を行ないます
4	割込み管理システムコール	割込みハンドラの終了・割込みマスクの変更および参照を行ないます
5	メモリ管理システムコール	固定長メモリブロックを持つメモリプールの管理を行ないます
6	時間管理システムコール	システムクロックの設定・参照を行ないます
7	システム管理システムコール	H18-3Hのバージョン識別子を求めます

表３－２　にシステムコールと、発行可能条件の一覧を示します。

表３－２　システムコール・発行可能条件一覧（１／２）

	システムコール		タスク(T)／非タスク(N)	
タスク管理				
1	sta_tsk	タスクを起動する	T	
2	ista_tsk	同上（非タスク部専用）		N
3	ext_tsk	自タスクを正常終了する	T	
4	ter_tsk	他タスクを強制的に異常終了させる	T	
5	chg_pri	自タスクの優先度を変更する	T	
6	rot_rdq	タスクのレディキューを回転する	T	
7	irot_rdq	同上（非タスク部専用）		N
8	rel_wai	タスクの待ち状態を強制解除する	T	
9	get_tid	自タスクのIDを得る	T	
10	tsk_sts	タスクの状態を見る	T	N
タスク付属同期管理				
11	sus_tsk	他タスクを強制待ち状態へ移行する	T	
12	rsm_tsk	強制待ち状態のタスクを再開する	T	
13	slp_tsk	自タスクを待ち状態へ移行する	T	
14	wai_tsk	自タスクを一定時間待ち状態に移行する	T	
15	wup_tsk	待ち状態のタスクを起床する	T	
16	iwup_tsk	同上（非タスク部専用）		N
17	can_wup	タスクの起床要求を無効にする	T	
同期・通信管理				
18	set_flg	イベントフラグをセットする	T	
19	iset_flg	同上（非タスク部専用）		N
20	clr_flg	イベントフラグをクリアする	T	N
21	wai_flg	イベントフラグを待つ	T	
22	pol_flg	イベントフラグを得る	T	
23	flg_sts	イベントフラグ状態を参照する	T	N
24	sig_sem	セマフォに対する信号操作（V命令）	T	
25	isig_sem	同上（非タスク部専用）		N
26	wai_sem	セマフォに対する待ち操作（P命令）	T	
27	preq_sem	セマフォ資源を得る	T	
28	sem_sts	セマフォ状態を参照する	T	N

表３－２　システムコール・発行可能条件一覧（２／２）

	システムコール		タスク(T)／非タスク(N)	
２９	snd_msg	メッセージを送信する	T	
３０	isnd_msg	同上（非タスク部専用）		N
３１	rcv_msg	メッセージの受信を待つ	T	
３２	prcv_msg	メッセージを受信する	T	
３３	mbx_sts	メイルボックス状態を参照する	T	N
	割込み管理			
３４	ret_int	割込みハンドラから復帰する		N
３５	chg_ims	割込みマスクを変更する	T	N
３６	ims_sts	割込みマスクを参照する	T	N
	メモリ管理			
３７	get_blk	固定長メモリブロックの獲得待ちを行なう	T	
３８	pget_blk	固定長メモリブロックを獲得する	T	
３９	rel_blk	固定長メモリブロックを返却する	T	
４０	mpl_sts	メモリプール状態を参照する	T	N
	時間管理			
４１	set_tim	システムクロックを設定する	T	N
４２	get_tim	システムクロックの値を読み出す	T	N
	システム管理			
４３	get_ver	カーネルのバージョン識別子を得る	T	N

３．２　システムコールインタフェース

プログラムでシステムコールを記述するための形式を説明します。
システムコールはアセンブリ言語およびＣ言語で記述できます。
本章で記述してある各システムコールの詳細説明は表３－３の順序で記載されています。

表３－３　説明順序

項番	項目
1	システムコール名
2	アセンブラフォーマット
3	Ｃフォーマット
4	エラーコード
5	解　説

３．２．１　パラメータ名称の統一的な省略形

パラメータの名称は統一的な省略形を使用しており、表３－４のプリフィクスを付けています。

表３－４　パラメータ名称の統一的な省略形

項番	省略形	意味
1	p_	ポインタを表わす
2	pk_	パラメータパケットアドレスを表わす
3	t_	構造体を表わす
4	ppk_	パラメータパケットアドレスへのポインタを表わす

３．２．２　エラーコード

H18-3Hのシステムコールは、実行結果として16ビットの整数をエラーコードとしてレジスタR0に返します。なお、システムコールの結果はCCR の各フラグには反映されません。
エラーコードに設定される値はシステムコールによって異なります。
エラーコードについては、「付録Ｃ　エラーコード一覧」を参照してください。
非タスク部でext_tsk システムコールを発行したり、タスク部でret_int システムコールを発行すると、システム異常終了します。

３．２．３　アセンブラインタフェース

H18-3Hでは、アセンブリ言語からのシステムコールの呼出しは、原則として次のような形になります。

```
JSR     @<システムコール名>
```

システムコールから戻ってきた場合、R0およびリターンパラメータとなるレジスタ以外のレジスタは保証（システムコール発行前と同じ）されます。

(1) パラメータの表記法

パラメータは、表３－５のような表現・意味で記述しています。

表３－５　パラメータの表現・意味

表現	意味
Rn	レジスタRn （全16ビット）が有効
ERn	レジスタERn（全32ビット）が有効

【注】　nはレジスタ番号

(2) システムコール発行時のパラメータ

システムコールのパラメータの受け渡しはレジスタを使用します。パラメータが多いときは、パラメータパケットにパラメータを設定し、パラメータパケットの先頭アドレス（パケットアドレス）をレジスタに設定します。

システムコールのパラメータで使用するレジスタはR0, R1, R2, R3, R4, ER2です。

その他のレジスタは使用しません。

通常レジスタを使用する目的を表３－６に示します。

表３－６　レジスタの使用目的

項番	レジスタ	目的
1	R0	エラーコード
2	R1	ID番号／割込みマスク値
3	R2	パラメータパケットアドレス／その他のパラメータ
4	R3	その他のパラメータ
5	R4	その他のパラメータ
6	ER2	パラメータパケットアドレス／その他のパラメータ

(3) エラーコード

エラーコードは、レジスタR0にセットされます。

CCR のＺフラグには反映されません。

(4) パラメータの説明

snd_msg, isnd_msgシステムコールを例にパラメータを説明します。

アセンブラフォーマット：

```
      JSR     @snd_msg  ---┐
      JSR     @isnd_msg ---+-------------------------------------------- ①
```

パラメータ：

```
      (1)H8/300Hアドバンストモード ------------------------------------ ②
        R1  メイルボックスID(mbxid:1～255) ------------------------- ③
        ER2 メッセージの先頭アドレス(pk_msg) ------------------------ ④
      (2)H8/300Hノーマルモード ---------------------------------------- ⑤
        R1  メイルボックスID(mbxid:1～255) ------------------------- ⑥
        R2  メッセージの先頭アドレス(pk_msg) ------------------------ ⑦
```

リターンパラメータ：

```
      R0  エラーコード(ercd) ------------------------------------------ ⑧
```

（解説）

① JSR 命令でシステムコールを発行します。

② H8/300Hアドバンストモード使用時のパラメータを説明します。

③ メイルボックスIDをR1 に設定します。

④ メッセージの先頭アドレス(24ﾋﾞｯﾄ)をER2に設定します。

⑤ H8/300Hノーマルモード使用時のパラメータを説明します。

⑥ メイルボックスIDをR1 に設定します。

⑦ メッセージの先頭アドレス(16ﾋﾞｯﾄ)を R2に設定します。

⑧ システムコールのエラーコードがR0 に返されます。

３．２．４　Ｃインタフェース

HI8-3Hでは、Ｃ言語からのシステムコールの呼出しは、原則として次のような形になります。

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

ercd：関数のリターン値でエラーコード(符号付き１６ビット整数)
void：リターン値が返らない関数
name：システムコール名
return parameter address：リターンパラメータアドレス(ポインタ)
parameter：パラメータ

(1) データタイプとサイズ

HI8-3Hが使用するパラメータのデータタイプとそのサイズを表３－７に示します。

表３－７　パラメータのデータタイプ・サイズ

項番	ﾃﾞｰﾀﾀｲﾌﾟ	サイズ	説明
1	B	char	符号付き８ビット整数
2	H	short	符号付き16ビット整数
3	W	long	符号付き32ビット整数
4	UB	unsigned char	符号なし８ビット整数
5	UH	unsigned short	符号なし16ビット整数
6	UW	unsigned long	符号なし32ビット整数
7	VB	char	データタイプが一定しないもの（８ビットサイズ）
8	VP	void *	データタイプが一定でないものへのポインタ
9	ID	unsigned short	オブジェクトのID(xxxid)
１０	ER	short	エラーコード
１１	TMO	long	タイムアウト時間
１２	TPRI	unsigned short	タスク優先度
１３	CCR	unsigned char	割込みマスク
１４	INT	short	符号付き整数（１６ビットサイズ）
１５	UINT	unsigned short	符号なし整数（１６ビットサイズ）

(2)パラメータの説明

flg_sts システムコールを例にパラメータを説明します。

Cフォーマット：

```
ER ercd=flg_sts(ID *p_wtskid, UINT *p_flgptn, ID flgid); ----------①
```

パラメータ：

```
ID      flgid;      イベントフラグID(flgid:1～255) ----------②
```

リターンパラメータ：

```
ID      *p_wtskid; 待ちタスクID(wtskid:0～255) ----------③
UINT    *p_flgptn; イベントフラグのビットパターン(flgptn:H'0000～H'FFFF) ------④
```

(解説)

① flg_stsシステムコールを発行するステートメントで、引数はp_wtskid,p_flgptnとflgidです。

② flg_stsシステムコールの引数flgidは、ID(unsigned short〔符号なし16ビット整数〕)でイベントフラグIDを設定します。

③ flg_stsシステムコールの引数p_wtskidは、ID(unsigned short〔符号なし16ビット整数〕)へのポインタを設定します。

④ flg_stsシステムコールの引数p_flgptnは、UINT(unsigned short〔符号なし16ビット整数〕)へのポインタを設定します。

３．３　タスク管理システムコール

３．３．１　sta_tsk (Start Task) タスクを起動する　［タスク部用］
ista_tsk (Interrupt Start Task) タスクを起動する　［非タスク部用］

アセンブラフォーマット：

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

パラメータ：

R1　タスクID(tskid:1～255)

リターンパラメータ：

R0　エラーコード(ercd)

Cフォーマット：

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

パラメータ：

ID　tskid;　タスクID(tskid:1～255)

リターンパラメータ：

なし

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_NOEXS	H'F7CC(-H'834)	そのタスクは生成されていない (tskid=0,tskid>タスク定義数(最大255),タスク未登録)
E_NODMT	H'F7CA(-H'836)	タスクが休止状態でない
*E_CTX	H'F5BB(-H'A45)	コンテキストエラー (タスク部または非タスク部から発行できない)

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

tskid で示されたタスクを休止状態から実行可能状態へ移行します。

ista_tskシステムコールは、非タスク部専用のsta_tsk システムコールです。

これらのシステムコールによる起動要求のキューイングは行なわれず、対象タスクが休止状態でない場合に発行された起動要求に対しては、エラーコードとしてE_NODMT を返します。

tskid が自タスクのIDでsta_tsk システムコールを発行した場合は、E_NODMT のエラーを返します。

tskid=0でsta_tsk システムコールを発行した場合は、E_NOEXS のエラーを返します。

H18-3Hは、複数のタスクでスタック領域を共有する機能（共有スタック機能）を持っています。セットアップテーブルでタスクスタックポインタに同じ値を登録したタスクID同士は、それらのタスク間でひとつのスタックを共有する（共有スタック）ことになります。
本システムコール発行時に、tskidで示されたタスクのスタック（共有スタック）が開放（どのタスクも使用していない）されている場合は、tskidで示されたタスクが共有スタックを占有し、実行可能状態に移行します。

共有スタックが占有されている（他のタスクが使用中）場合は、スタック領域を使用できないため、タスクは待ち状態になり、共有スタックの待ち行列につながれます。

待ち行列のつながれ方は、FIFO(First-In First-Out)です。

共有スタック待ち状態のタスクに対して本システムコールを発行すると、発行タスクにエラーコードとしてE_OKを返します。

共有スタック機能については、「２．２．５　共有スタック機能」を参照してください。

３．３．２　ext_tsk (Exit Task) 自タスクを正常終了する　［タスク部用］

アセンブラフォーマット：

```
JMP     @ext_tsk
```

パラメータ：

なし

リターンパラメータ：

正常終了時　システムコールを発行したコンテキストには戻らない

異常終了時　システム異常終了処理に制御を移す

Cフォーマット：

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

パラメータ：

なし

リターンパラメータ：

正常終了時　システムコールを発行したコンテキストには戻らない

異常終了時　システム異常終了処理に制御を移す

エラーコード(R0/ercd)：

返さない

解　説：

自タスクを正常終了させ、休止状態へ移行します。

非タスク部（割込みをマスクした状態）から本システムコールを発行すると、システム異常終了処理に制御を移行します。

ext_tsk システムコールでは、タスクがそれ以前に獲得した資源（メモリブロック，セマフォなど）を自動的に解放することはありません。したがって、ext_tsk システムコールを発行する前に資源を解放させる必要があります。

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

共有スタック機能については、「２．２．５　共有スタック機能」を参照してください。

３．３．３　ter_tsk (Terminate Task) 他タスクを強制的に異常終了させる　　[タスク部用]

アセンブラフォーマット：

```
JSR     @ter_tsk
```

パラメータ：

R1　タスクID(tskid:1～255)

リターンパラメータ：

R0　エラーコード(ercd)

Cフォーマット：

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

パラメータ：

ID　　tskid;　タスクID(tskid:1～255)

リターンパラメータ：

なし

エラーコード(R0/ercd)：

	E_OK	H'0000(H'000)	正常終了
*	E_SELF	H'F7CF(-H'831)	自タスク指定 (tskid=自タスクID)
*	E_NOEXS	H'F7CC(-H'834)	そのタスクは生成されていない (tskid=0,tskid>タスク定義数(最大255),タスク未登録)
	E_DMT	H'F7CB(-H'835)	タスクが休止状態である
*	E_CTX	H'F5BB(-H'A45)	コンテキストエラー (非タスク部から発行できない)

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

対象タスクを強制的に異常終了させます。

tskid=0でter_tsk システムコールを発行した場合は、E_NOEXS のエラーを返します。

ter_tsk システムコールでは、対象タスクがそれ以前に獲得した資源（メモリブロック，セマフォなど）を自動的に解放することはありません。したがって、ter_tsk システムコールを発行する前に資源を解放させる必要があります。

共有スタック機能を組み込んでいる場合、本システムコールの発行により、tskid で示された他タスクが占有していたスタックを解放します。このスタックの待ち行列にタスクがつながれている場合（共有スタック）、先頭のタスクをスタックの待ち行列から外し、実行可能状態へ移行します。

共有スタック機能については、「２．２．５　共有スタック機能」を参照してください。

３．３．４　chg_pri (Change Task Priority) タスク優先度を変更する　　[タスク部用]

アセンブラフォーマット：

```
JSR     @chg_pri
```

パラメータ：

R1　タスクID(tskid:0,1～255)

R2　優先度(tskpri:0,1～31)

リターンパラメータ：

R0　エラーコード(ercd)

Cフォーマット：

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

パラメータ：

ID　　tskid;　タスクID(tskid:0,1～255)

TPRI　tskpri;　優先度(tskpri:0,1～31)

リターンパラメータ：

なし

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_TPRI	H'F8DA(-H'726)	不正タスク優先度 (tskpri>優先度定義数(最大31))
*E_NOEXS	H'F7CC(-H'834)	そのタスクは生成されていない (tskid>タスク定義数 (最大255),タスク未登録)
E_DMT	H'F7CB(-H'835)	タスクが休止状態である
*E_CTX	H'F5BB(-H'A45)	コンテキストエラー (非タスク部から発行できない)

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

tskid で示されたタスクの優先度をtskpriで示された値に変更します。

tskid=0 とした場合は、自タスク指定となります。

タスク優先度は、0 ～優先度定義数の値で、値の小さい方が高い優先度となります。tskpri=0を指定した場合、システム構築時に定義したタスク初期優先度(itskpri) が設定されます。

このシステムコールで変更した優先度は、タスクが終了するまで有効です。タスクが終了すると、システム構築時に定義したタスク初期優先度に戻ります。

３．３．５　rot_rdq (Rotate Ready Queue) タスクのレディキューを回転する　［タスク部用］
irot_rdq (Interrupt Rotate Ready Queue) タスクのレディキューを回転する［非タスク部用］

アセンブラフォーマット：

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

パラメータ：

R2　優先度(tskpri:0,1～31)

リターンパラメータ：

R0　エラーコード(ercd)

Cフォーマット：

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

パラメータ：

TPRI　tskpri;　優先度(tskpri:0,1～31)

リターンパラメータ：

なし

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_TPRI	H'F8DA(-H'726)	不正タスク優先度 (tskpri>優先度定義数(最大31))
*E_CTX	H'F5BB(-H'A45)	コンテキストエラー (タスク部または非タスク部から発行できない)

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

tskpriで示される優先度のレディキューの先頭につながれているタスクをレディキューの最後尾につなぎかえ、同一優先度のタスクの実行を切り換えます。このシステムコールを一定時間間隔で発行することにより、ラウンドロビン・スケジューリングを行なうことが可能となります。

irot_rdqシステムコールは、非タスク部専用のrot_rdq システムコールです。

irot_rdqシステムコールを、tskpri=0で発行した場合には、そのとき実行状態にあるタスクを含むレディキュー（最高優先度のレディキュー）を回転させます。

rot_rdq システムコールを、tskpri=0で発行した場合には、自タスクがそのレディキューの最後尾にまわることになり、自ら実行権を放棄することができます。

対象となる優先度のレディキューにタスクが存在しない場合は何も行いませんが、エラーとはなりません。エラーコードとしてE_OKが返されます。

３．３．６　rel_wai (Release Wait) タスクの待ち状態を強制解除する　　[タスク部用]

アセンブラフォーマット：

```
JSR     @rel_wai
```

パラメータ：

R1　タスクID(tskid:1～255)

リターンパラメータ：

R0　エラーコード(ercd)

Cフォーマット：

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

パラメータ：

ID　tskid;　タスクID(tskid:1～255)

リターンパラメータ：

なし

エラーコード(R0/ercd)：

	E_OK	H'0000(H'000)	正常終了
*	E_NOEXS	H'F7CC(-H'834)	そのタスクは生成されていない (tskid=0,tskid>タスク定義数(最大255),タスク未登録)
	E_NOWAI	H'F7C2(-H'83E)	タスクが待ち状態でない
*	E_CTX	H'F5BB(-H'A45)	コンテキストエラー（非タスク部から発行できない）

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

tskidで示されるタスクが待ち状態（強制待ち状態を除く）の場合、それを強制的に解除し、実行可能状態に移行します。

表３－８に、タスクの待ち状態を示します。

表３－８　タスクの待ち状態

項番	待ち状態	発行したシステムコール
1	スリープ	slp_tskシステムコール
2	時間待ち	wai_tskシステムコール
3	イベントフラグ待ち	wai_flgシステムコール
4	セマフォ待ち	wai_semシステムコール
5	メッセージ待ち	rcv_msgシステムコール
6	メモリ取得待ち	get_blkシステムコール

強制的に待ち状態が解除されたタスクに対しては、エラーコードとしてE_RLWAI （待ち状態強制解除）を返します。

tskid で示される対象タスクが待ち状態であればその待ち状態を解除し、rel_wai システムコール発行元に正常終了としてE_OKを返しますが、待ち状態でなければ、エラーコードとしてE_NOWAIを返します。対象タスクに自タスクを指定した場合は、エラーコードとしてE_NOWAI を返します。

tskid=0でrel_wai システムコールを発行した場合は、E_NOEXS のエラーを返します。

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

【注】　本システムコールでは、強制待ち状態の解除は行なえません。
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。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。

なお、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し、その後rsm_tsk システムコールが発行され、強制待ち状態が解除されたとき、待ちタスクにはエラーコードとしてE_RLWAI が返されます。

３．３．７　get_tid (Get Task Identifier) 自タスクのIDを得る　　［タスク部用］

アセンブラフォーマット：

```
JSR     @get_tid
```

パラメータ：

なし

リターンパラメータ：

R0　エラーコード(ercd)

R1　タスクID(tskid:1～255)

Cフォーマット：

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

パラメータ：

なし

リターンパラメータ：

```
ID    *p_tskid;  タスクID(tskid:1～255)
```

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_CTX	H'F5BB(-H'A45)	コンテキストエラー（非タスク部から発行できない）

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

自タスクのタスクIDを取得します。

３．３．８　tsk_sts (Get Task Status) タスクの状態を見る　　[タスク部／非タスク部両用]

アセンブラフォーマット：

```
JSR     @tsk_sts
```

パラメータ：

R1　タスクID(tskid:0,1～255)

リターンパラメータ：

R0　エラーコード(ercd)
R2　現在のタスク優先度(tskpri:1～31)
R3　タスクステータス(tskstat)

Cフォーマット：

```
ER ercd=tsk_sts(UINT *p_tskstat, TPRI *p_tskpri, ID tskid);
```

パラメータ：

```
ID      tskid;      タスクID(tskid:0,1～255)
```

リターンパラメータ：

```
TPRI    *p_tskpri;  現在のタスク優先度(tskpri:1～31)
UINT    *p_tskstat; タスクステータス
```

エラーコード(R0/ercd)：

	E_OK	H'0000(H'000)	正常終了
*	E_NOEXS	H'F7CC(-H'834)	そのタスクは生成されていない (tskid>タスク定義数(最大255),タスク未登録, 非タスク部からtskid=0 を指定)

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

tskid で示されるタスクの状態を読み出し、そのタスクの現在の優先度(tskpri)、タスクステータス(tskstat) をリターンパラメータとして返します。

【注】

(1)　tskid=0 で発行した場合は、自タスク指定となりますが、このシステムコールでは、自タスクのIDは分りません。自タスクのIDを知りたい場合は、get_tid システムコールを利用してください。非タスク部からtskid=0 を指定した場合、エラーコードとしてE_NOEXS を返します。

(2) 非タスク部からtsk_stsシステムコールを発行したときには、タスクステータス(tskstat)が正しく返されない場合があります。以下に正しく返されない場合を示します。

(a)カーネルがタスクの状態を移行しているときには、タスクステータス(tskstat)として0が返されることがあります。

(b)wai_tskシステムコールを発行し、カーネルがタスクの状態を移行しているときには、タスクステータス(tskstat)としてslp_tskシステムコールによる待ち状態が返されることがあります。

図3－1に示すように、タスクステータス(tskstat)は16ビット長のフラグで構成されています。

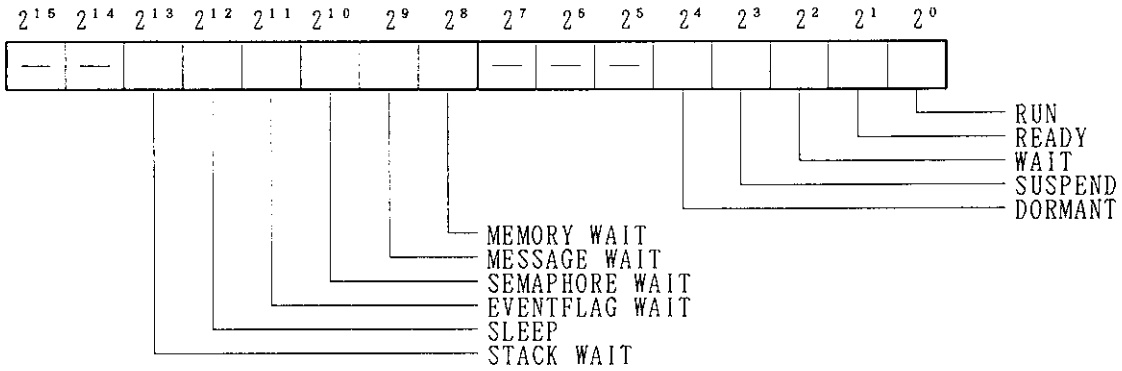


図3－1　タスクステータス

以下にtskstatのステータスフラグを示します。

tskstat の下位バイトはタスクの状態を示します。

ビット 0=1	RUN	(実行状態)
ビット 1=1	READY	(実行可能状態)
ビット 2=1	WAIT	(待ち状態)
ビット 3=1	SUSPEND	(強制待ち状態)
ビット 2,3=1	WAIT-SUSPEND	(二重待ち状態)
ビット 4=1	DORMANT	(休止状態)
ビット 5,6=0	予約ビットで特に意味はありません	
ビット 7	カーネルが使用する制御ビットです	

tskstat の上位バイトはタスクが待ち(WAIT)状態に移行した要因を示します。

ビット 8=1	MEMORY WAIT	(get_blkシステムコールによる待ち状態)
ビット 9=1	MESSAGE WAIT	(rcv_msgシステムコールによる待ち状態)
ビット10=1	SEMAPHORE WAIT	(wai_semシステムコールによる待ち状態)
ビット11=1	EVENTFLAG WAIT	(wai_flgシステムコールによる待ち状態)
ビット12=1	SLEEP	(slp_tskシステムコールによる待ち状態)
ビット13=1	STACK WAIT	(共有スタックの待ち状態)
ビット14=0	予約ビットで特に意味はありません	
ビット15	カーネルが使用する制御ビットです	

実行状態のときは、ビット0の他にビット1も1です。

休止状態のときは、ビット4以外はすべて0です。

wai_tsk(tmout≠-1)システムコールの場合は、ビット8～12、15は0ですが、wai_tsk(tmout=-1)システムコールによる待ち状態の場合は、slp_tsk システムコールと同じ動作を行ないますので、ビット12は1です。

３．４　タスク付属同期管理システムコール

３．４．１　sus_tsk (Suspend Task) 他タスクを強制待ち状態へ移行する　　[タスク部用]

アセンブラフォーマット：

```
    JSR     @sus_tsk
```

パラメータ：

R1　タスクID(tskid:1～255)

リターンパラメータ：

R0　エラーコード(ercd)

Cフォーマット：

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

パラメータ：

ID　　tskid;　タスクID(tskid:1～255)

リターンパラメータ：

なし

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_SELF	H'F7CF(-H'831)	自タスク指定 (tskid=自タスクID)
*E_NOEXS	H'F7CC(-H'834)	そのタスクは生成されていない (tskid=0,tskid>タスク定義数(最大255),タスク未登録)
E_DMT	H'F7CB(-H'835)	タスクが休止状態である
*E_CTX	H'F5BB(-H'A45)	コンテキストエラー (非タスク部から発行できない)
E_QOVR	H'F4B7(-H'B49)	キューイングのオーバフロー (既に強制待ち状態になっている)

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

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って解除されます。

tskid で示されたタスクが、すでに待ち状態にある場合は、二重待ち状態になります。

強制待ち要求のキューイングはないので、強制待ち状態のときに本システムコールが発行された場合、エラーコードとしてE_QOVRを返します。

強制待ち状態は、他タスクからのシステムコールによる中断状態を意味するものなので、本システムコールで自タスクを指定することはできません。この場合、エラーコードとしてE_SELFを返します。

tskid=0でsus_tsk システムコールを発行した場合は、E_NOEXS のエラーを返します。

３．４．２　rsm_tsk (Resume Task) 強制待ち状態のタスクを再開する　　[タスク部用]

アセンブラフォーマット：

```
    JSR     @rsm_tsk
```

パラメータ：

R1　タスクID(tskid:1～255)

リターンパラメータ：

R0　エラーコード(ercd)

Cフォーマット：

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

パラメータ：

ID　tskid　タスクID(tskid:1～255)

リターンパラメータ：

なし

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_NOEXS	H'F7CC(-H'834)	そのタスクは生成されていない (tskid=0,tskid>タスク定義数(最大255),タスク未登録)
E_NOSUS	H'F7C8(-H'838)	タスクが強制待ち状態でない
*E_CTX	H'F5BB(-H'A45)	コンテキストエラー（非タスク部から発行できない）

【注】*システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

tskid で示されたタスクがsus_tsk システムコールによって中断されている場合、対象タスクの強制待ち状態を解除し、実行可能状態へ移行させます。対象タスクが二重待ち状態であれば、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、本システムコールで必ず強制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。この場合、エラーコードとしてE_NOSUS を返します。

対象タスクが二重待ち状態のとき、強制待ち状態が解除されてもまだ待ち状態であり、待ち解除条件が満たされるまでタスクは実行可能状態へ移行されません。

tskid=0でrsm_tsk システムコールを発行した場合は、E_NOEXS のエラーを返します。

３．４．３　slp_tsk (Sleep Task) 自タスクを待ち状態へ移行する　　[タスク部用]

アセンブラフォーマット：

```
JSR     @slp_tsk
```

パラメータ：

なし

リターンパラメータ：

R0　エラーコード(ercd)

Cフォーマット：

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

パラメータ：

なし

リターンパラメータ：

なし

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_CTX	H'F5BB(-H'A45)	コンテキストエラー (非タスク部から発行できない)
E_RLWAI	H'F2AA(-H'D56)	待ち状態強制解除

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

自タスクを実行状態から待ち状態に移行します。

この待ち状態は、本タスクを対象として発行されたwup_tsk システムコールにより解除されます。ただし、wup_tsk システムコールによる起床要求がキューイングされていれば、待ち状態とならず、そのまま実行を続けます。

本システムコールにより待ち状態となっているときに、他のタスクからsus_tsk システムコールが実行された場合、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は実行可能状態へ移行されません。

３．４．４　wai_tsk (Wait For Wakeup Task) 自タスクを一定時間待ち状態に移行する　[タスク部用]

アセンブラフォーマット：

```
JSR     @wai_tsk
```

パラメータ：

ER2 タイムアウト指定(tmout:-1,H'1～H'7FFFFFFF)

リターンパラメータ：

R0　エラーコード(ercd)

Cフォーマット：

```
ER ercd=wai_tsk(TMO tmout);
```

パラメータ：

TMO　tmout;　タイムアウト指定(tmout:-1,H'1～H'7FFFFFFF)

リターンパラメータ：

なし

エラーコード(R0/ercd)：

	E_OK	H'0000(H'000)	正常終了
*	E_ILTIME	H'F8D9(-H'727)	不正時間指定(tmout≦-2)
*	E_CTX	H'F5BB(-H'A45)	コンテキストエラー (非タスク部から発行できない)
	E_TMOUT	H'F2AB(-H'D55)	タイムアウト
	E_RLWAI	H'F2AA(-H'D56)	待ち状態強制解除

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

自タスクを一定時間だけ実行状態から待ち状態に移行します。

この待ち状態は、本タスクを対象としたwup_tsk システムコールの発行、またはtmout で指定した時間の経過により解除され、wup_tsk システムコールが発行された場合は正常終了、時間経過の場合はタイムアウトエラー(E_TMOUT) のエラーコードを返します。

tmout のビット数は32ビットで、１クロック１msの場合、約24日の制御を行なうことができます。

tmout=-1は永久待ち指定となり、slp_tsk システムコールと同じ動作を行ないます。

また、tmout=0 と指定した場合は、エラーコードとしてE_TMOUT を返します。

本システムコールにより待ち状態となっているときに、他のタスクからsus_tsk システムコールが実行された場合、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は実行可能状態へ移行されません。

３．４．５　wup_tsk (Wakeup Task) 待ち状態のタスクを起床する　［タスク部用］
iwup_tsk (Interrupt Wakeup Task) 待ち状態のタスクを起床する　［非タスク部用］

アセンブラフォーマット：

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

パラメータ：

R1 タスクID(tskid:1～255)

リターンパラメータ：

R0　エラーコード(ercd)

Cフォーマット：

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

パラメータ：

ID　tskid;　タスクID(tskid:1～255)

リターンパラメータ：

なし

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_SELF	H'F7CF(-H'831)	自タスク指定 (tskid=自タスクID)　《wup_tsk システムコールのみ》
*E_NOEXS	H'F7CC(-H'834)	そのタスクは生成されていない (tskid=0,tskid>タスク定義数(最大255),タスク未登録)
E_DMT	H'F7CB(-H'835)	タスクが休止状態である
*E_CTX	H'F5BB(-H'A45)	コンテキストエラー (タスク部または非タスク部から発行できない)
E_QOVR	H'F4B7(-H'B49)	キューイングオーバフロー

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

slp_tsk システムコールまたはwai_tsk システムコールの発行により待ち状態になっているタスクを、実行可能状態に移行させます。

iwup_tskシステムコールは非タスク部専用のwup_tskシステムコールです。

tskidで示されたタスクが待ち状態でない場合、この起床要求はキューイングされ、後に対象タスクがslp_tsk システムコールまたはwai_tsk システムコールを発行したとき有効になります。

キューイングの最大値は255 で、これを越えた場合にはエラーコードとしてE_QOVRを返します。

wup_tskシステムコールは、自タスクを指定することはできません。この場合、エラーコードとしてE_SELFを返します。

tskid=0でwup_tsk システムコールまたはiwup_tskシステムコールを発行した場合は、E_NOEXSのエラーを返します。

３．４．６　can_wup (Cancel Wakeup Task) タスクの起床要求を無効にする　　[タスク部用]

アセンブラフォーマット

```
    JSR     @can_wup
```

パラメータ：

R1　タスクID(tskid:0,1～255)

リターンパラメータ：

R0　エラーコード(ercd)

R2　キューイングされていた起床要求回数(wupcnt:0～255)

Cフォーマット：

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

パラメータ：

```
    ID      tskid;      タスクID(tskid:0,1～255)
```

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(wupcnt:0～255)
```

エラーコード(R0/ercd)：

	E_OK	H'0000(H'000)	正常終了
*	E_NOEXS	H'F7CC(-H'834)	そのタスクは生成されていない (tskid>タスク定義数(最大255),タスク未登録)
	E_DMT	H'F7CB(-H'835)	タスクが休止状態である
*	E_CTX	H'F5BB(-H'A45)	コンテキストエラー (非タスク部から発行できない)

【注】*システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

tskid で示されたタスクに対してキューイングされていた起床要求回数をリターンパラメータとして返し、その起床要求をすべて解除します。

本システムコールが発行された時点で対象タスクのwup_tsk システムコールによるキューカウンタがすでに０である場合は、起床要求回数(wupcnt)として０を返します。

tskid=0 を指定した場合は、自タスク指定になります。

３．５　同期／通信管理システムコール

３．５．１　set_flg (Set Event Flag) イベントフラグをセットする　［タスク部用］
iset_flg (Interrupt Set Event Flag) イベントフラグをセットする［非タスク部用］

アセンブラフォーマット：

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

パラメータ：

R1　イベントフラグID(flgid:1～255)
R2　セットするビットパターン(setptn:H'0000～H'FFFF)

リターンパラメータ：

R0　エラーコード(ercd)

Cフォーマット：

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

パラメータ：

```
ID      flgid;   イベントフラグID(flgid:1～255)
UINT    setptn;  セットするビットパターン(setptn:H'0000～H'FFFF)
```

リターンパラメータ：

なし

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_NOEXS	H'F7CC(-H'834)	そのイベントフラグは生成されていない (flgid=0,flgid>イベントフラグ定義数(最大255))
*E_CTX	H'F5BB(-H'A45)	コンテキストエラー (タスク部または非タスク部から発行できない)

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

flgid で示されるイベントフラグ（16ビット）のうち、対応するsetptnのオン(1) になっているビットがセットされます。すなわち、flgid で示されるイベントフラグ値に対してsetptnの値で論理和(OR)をとってセットします。

イベントフラグ値の変更の結果、そのイベントフラグを待っていたタスクの待ち解除条件を満たすようになれば、そのタスクを実行可能状態へ移行します。

iset_flgシステムコールは非タスク部用のset_flg システムコールです。

setptnの全ビットを0 とした場合、対象イベントフラグに対して何の操作も行なわないことになりますが、エラーとはなりません。

３．５．２　clr_flg (Clear Event Flag) イベントフラグをクリアする

［タスク部／非タスク部両用］

アセンブラフォーマット：

```
JSR     @clr_flg
```

パラメータ：

R1　イベントフラグID(flgid:1～255)

R2　クリアするビットパターン(clrptn:H'0000～H'FFFF)

リターンパラメータ：

R0　エラーコード(ercd)

Cフォーマット：

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

パラメータ：

```
ID      flgid;   イベントフラグID(flgid:1～255)
UINT    clrptn;  クリアするビットパターン(clrptn:H'0000～H'FFFF)
```

リターンパラメータ：

なし

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_NOEXS	H'F7CC(-H'834)	そのイベントフラグは生成されていない (flgid=0,flgid>イベントフラグ定義数(最大255))

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

flgid で示されるイベントフラグ（16ビット）のうち、対応するclrptnのオフ(0) になっているビットがクリアされます。すなわち、flgid で示されるイベントフラグ値に対してclrptnの値で論理積(AND)をとってセットします。

本システムコールで、flgid で示される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。

clrptnの全ビットを1 とした場合、対象イベントフラグに対して何の操作も行なわないことになりますが、エラーとはなりません。

３．５．３　wai_flg (Wait Event Flag) イベントフラグを待つ　[タスク部用]
pol_flg (Poll Event Flag) イベントフラグを得る　[タスク部用]

アセンブラフォーマット：

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

パラメータ：

R1　イベントフラグID(flgid:1～255)
R2　待ちビットパターン(waiptn:H'0001～H'FFFF)
R3　待ちモード(wfmode:0～3)

リターンパラメータ：

R0　エラーコード(ercd)
R2　セットされた後のイベントフラグの内容(flgptn:H'0001～H'FFFF)

Cフォーマット：

```
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);
```

パラメータ：

```
ID      flgid;      イベントフラグID(flgid:1～255)
UINT    waiptn;     待ちビットパターン(waiptn:H'0001～H'FFFF)
UINT    wfmode;     待ちモード(wfmode:0～3)
```

リターンパラメータ：

```
UINT    *p_flgptn;  セットされた後のイベントフラグの内容(flgptn:H'0001～H'FFFF)
```

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_PAR	H'F8DF(-H'721)	パラメータエラー(waiptn=0または, wfmode>3)
*E_NOEXS	H'F7CC(-H'834)	そのイベントフラグは生成されていない (flgid=0,flgid>イベントフラグ定義数(最大255))
*E_CTX	H'F5BB(-H'A45)	コンテキストエラー (非タスク部から発行できない)
E_QOVR	H'F4B7(-H'B49)	キューイングのオーバーフロー《wai_flgシステムコールのみ》 (既に待ちタスクが存在している)
E_RLWAI	H'F2AA(-H'D56)	待ち状態強制解除
E_PLFAIL	H'F1A7(-H'E59)	ポーリング失敗　《pol_flgシステムコールのみ》

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

wai_flg システムコールは、flgid で示されたイベントフラグのうち、待ちビットパターン(waiptn)で指定したビットのすべて（あるいは、いずれか）がセットされるまで待ちます。

pol_flg システムコールは、待ちビットパターン(waiptn)で指定したビットのすべて（あるいは、いずれか）がセットされていない場合には、エラーリターンします。（エラーコードとしてE_PLFAILを返します）

リターンパラメータには、セットされた後のイベントフラグの内容(flgptn)が返されますが、待ちモードにTWF_CLR を指定した場合、クリア前の内容が返されます。この時エラーリターンするとイベントフラグはクリアされません。

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

表３－９に待ちモードの詳細を示します。

表３－９　待ちモード(wfmode)のコードと意味

項番	待ちモード	ｺｰﾄﾞ	意　　味
1	TWF_ANDW	H'00	waiptnで指定したビットが、すべてセットされるまで待つ（AND待ち）
2	TWF_ANDW|TWF_CLR	H'01	AND待ち条件が満たされてタスクが待ち解除となった場合、イベントフラグの値（16ビットすべて）を０にクリアする
3	TWF_ORW	H'02	waiptnで指定したビットのいずれかがセットされるまで待つ（OR待ち）
4	TWF_ORW|TWF_CLR	H'03	OR待ち条件が満たされて、タスクが待ち解除となった場合、イベントフラグの値（16ビットすべて）を０にクリアする

３．５．４　flg_sts (Get Event Flag Status) イベントフラグ状態を参照する

［タスク部／非タスク部両用］

アセンブラフォーマット：

```
JSR     @flg_sts
```

パラメータ：

R1　イベントフラグID(flgid:1～255)

リターンパラメータ：

R0　エラーコード(ercd)

R2　イベントフラグのビットパターン(flgptn:H'0000～H'FFFF)

R3　待ちタスクID(wtskid:0～255)

Cフォーマット：

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

パラメータ：

```
ID      flgid;      イベントフラグID(flgid:1～255)
```

リターンパラメータ：

```
ID      *p_wtskid;  待ちタスクID(wtskid:0～255)
UINT    *p_flgptn;  イベントフラグのビットパターン(flgptn:H'0000～H'FFFF)
```

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_NOEXS	H'F7CC(-H'834)	そのタスクは生成されていない (flgid=0,flgid>イベントフラグ定義数(最大255))

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

flgid で示されるイベントフラグの状態を参照し、対象イベントフラグの現在のイベントフラグ値(flgptn)、待ちタスクID(wtskid)をリターンパラメータとして返します。

待ちタスクID(wtskid)には、対象イベントフラグの待ちタスクIDが返されます。待ちタスクがない場合は、NOTSK(H'0000)が返されます。

３．５．５　sig_sem (Signal Semaphore) セマフォに対する信号操作(Ｖ命令)　［タスク部用］
isig_sem (Interrupt Signal Semaphore) セマフォに対する信号操作(Ｖ命令)［非タスク部用］

アセンブラフォーマット：

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

パラメータ：

R1　セマフォID(semid:1～255)

リターンパラメータ：

R0　エラーコード(ercd)

Cフォーマット：

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

パラメータ：

ID　semid;　セマフォID(semid:1～255)

リターンパラメータ：

なし

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_NOEXS	H'F7CC(-H'834)	そのセマフォは生成されていない (semid=0,semid>セマフォ定義数(最大255))
*E_CTX	H'F5BB(-H'A45)	コンテキストエラー (タスク部または非タスク部から発行できない)
E_QOVR	H'F4B7(-H'B49)	キューイングのオーバーフロー (セマフォのカウント値のオーバーフロー)

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

semid で示されるセマフォに待ち行列がなければ、セマフォのカウント値を１増やし、待ち行列にタスクがつながれているならば、その先頭のタスクをセマフォの待ち行列から外し、実行可能状態へ移行します。

isig_semシステムコールは非タスク部用のsig_sem システムコールです。

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

３．５．６　wai_sem (Wait on Semaphore) セマフォに対する待ち操作(P命令)　[タスク部用]
preq_sem (Poll and Request Semaphore) セマフォ資源を得る　[タスク部用]

アセンブラフォーマット：

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

パラメータ：

R1　セマフォID(semid:1～255)

リターンパラメータ：

R0　エラーコード(ercd)

Cフォーマット：

```
ER ercd=wai_sem(ID semid);
ER ercd=preq_sem(ID semid);
```

パラメータ：

ID　semid;　セマフォID(semid:1～255)

リターンパラメータ：

なし

エラーコード(R0/ercd)：

	E_OK	H'0000(H'000)	正常終了
*	E_NOEXS	H'F7CC(-H'834)	そのセマフォは生成されていない (semid=0,semid>セマフォ定義数(最大255))
*	E_CTX	H'F5BB(-H'A45)	コンテキストエラー (非タスク部から発行できない)
	E_RLWAI	H'F2AA(-H'D56)	待ち状態強制解除
	E_PLFAIL	H'F1A7(-H'E59)	ポーリング失敗　《preq_semシステムコールのみ》

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

wai_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(semcnt)が１以上の場合、セマフォのカウント値を１減らした後、本システムコールの発行タスクの実行を継続します。セマフォのカウント値が０の場合、セマフォのカウント値は変更せず、本システムコールの発行タスクを待ち状態とし、そのセマフォの待ち行列につなぎます。つなぎ方はFIFO(First-In First-Out)です。

preq_semシステムコールは、セマフォのカウント値が０の場合、本システムコールを発行したタスクにエラーリターンします。（エラーコードとしてE_PLFAILを返します）

３．５．７　sem_sts (Get Semaphore Status) セマフォ状態を参照する
[タスク部／非タスク部両用]

アセンブラフォーマット：

```
    JSR     @sem_sts
```

パラメータ：

R1　セマフォID(semid:1～255)

リターンパラメータ：

R0　エラーコード(ercd)
R2　セマフォカウント値(semcnt:0～65,535)
R3　待ちタスクID(wtskid:0～255)

Cフォーマット：

```
    ER ercd=sem_sts(ID *p_wtskid, UINT *p_semcnt, ID semid);
```

パラメータ：

```
    ID      semid;      セマフォID(semid:1～255)
```

リターンパラメータ：

```
    ID      *p_wtskid;  待ちタスクID(wtskid:0～255)
    UINT    *p_semcnt;  セマフォカウント値(semcnt:0～65,535)
```

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_NOEXS	H'F7CC(-H'834)	そのセマフォは生成されていない (semid=0,semid>セマフォ定義数(最大255))

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

semid で示されるセマフォの状態を参照し、対象セマフォの現在のセマフォカウント値(semcnt)、待ちタスクID(wtskid)をリターンパラメータとして返します。

待ちタスクID(wtskid)には、対象セマフォのタスクの待ち行列の先頭のタスクIDが返されます。

待ちタスクがない場合は、NOTSK(H'0000)が返されます。

３．５．８　snd_msg (Send Message to Mailbox) メッセージを送信する　[タスク部用]
isnd_msg (Interrupt Send Message to Mailbox) メッセージを送信する　[非タスク部用]

アセンブラフォーマット：

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

パラメータ：

(1)H8/300Hアドバンストモード

R1　メイルボックスID(mbxid:1～255)

ER2 メッセージの先頭アドレス(pk_msg)

(2)H8/300Hノーマルモードの場合

R1　メイルボックスID(mbxid:1～255)

R2　メッセージの先頭アドレス(pk_msg)

リターンパラメータ：

R0　エラーコード(ercd)

Cフォーマット：

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

パラメータ：

```
ID    mbxid;        メイルボックスID(mbxid:1～255)
typedef struct t_msg {
    UW msghead;     OS管理領域
    VB msgcont[];   メッセージの内容
} T_MSG *pk_msg;    メッセージの先頭アドレス;
```

リターンパラメータ：

なし

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_ILADR	H'F8DE(-H'722)	不正アドレス(メッセージの先頭アドレスが０または奇数)
E_ILMSG	H'F8D7(-H'729)	不正メッセージ形式（メッセージ先頭４バイトが０でない)
*E_NOEXS	H'F7CC(-H'834)	そのメイルボックスは生成されていない (mbxid=0,mbxid>メイルボックス定義数(最大255))
*E_CTX	H'F5BB(-H'A45)	コンテキストエラー (タスク部または非タスク部から発行できない)

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

mbxid で示されたメイルボックスに、pk_msgのアドレスで示されるメッセージを送信します。

メッセージの内容はコピーされずアドレスのみが渡されます。

isnd_msgシステムコールは非タスク部用のsnd_msg システムコールです。

メッセージは、対象メイルボックスに待ちタスクがない場合、メッセージの待ち行列につなぎます。つなぎ方はFIFO(First-In First-Out)です。この時、発行タスクは待ち状態とはならず、メッセージをメイルボックスに入れるだけでタスクは実行を継続します。

メッセージとして利用できる領域は、メッセージの先頭アドレス＋４バイト目からです。メッセージの先頭４バイトはOSで使用するため、メッセージ領域は必ずRAM 上に確保してください。また、送信する際には、先頭４バイトに初期値として必ず０を設定し、送信後も書き換えないでください。送信する際に０が設定されていない場合は、エラーコードとしてE_ILMSG を返します。

図３－２にメッセージの形式を示します。

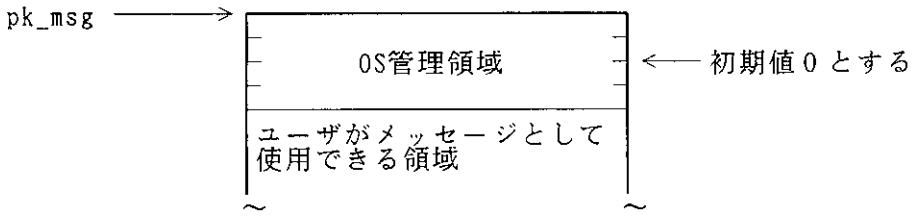


図３－２　メッセージの形式

【注】　メッセージの先頭４バイトを送信後に書き換えた場合、動作は保証されません。

メッセージの長さについてはOSで管理していませんのでユーザプログラムで管理する必要があります。

３．５．９　rcv_msg (Receive Message from Mailbox) メッセージの受信を待つ　[タスク部用]
prcv_msg (Poll and Receive Message) メッセージを受信する　[タスク部用]

アセンブラフォーマット：

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

パラメータ：

R1　メイルボックスID(mbxid:1～255)

リターンパラメータ：

(1)H8/300Hアドバンストモード

R0　エラーコード(ercd)

ER2 メッセージの先頭アドレス(pk_msg)

(2)H8/300Hノーマルモード

R0　エラーコード(ercd)

R2　メッセージの先頭アドレス(pk_msg)

Cフォーマット：

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

パラメータ：

```
ID    mbxid;        メイルボックスID(mbxid:1～255)
```

リターンパラメータ：

```
typedef struct t_msg {
    UW msghead;     OS管理領域
    VB msgcont[];   メッセージの内容
} T_MSG **ppk_msg;  メッセージの先頭アドレスへのポインタ
```

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_NOEXS	H'F7CC(-H'834)	そのメイルボックスは生成されていない (mbxid=0,mbxid>メイルボックス定義数(最大255))
*E_CTX	H'F5BB(-H'A45)	コンテキストエラー (非タスク部から発行できない)
E_RLWAI	H'F2AA(-H'D56)	待ち状態強制解除
E_PLFAIL	H'F1A7(-H'E59)	ポーリング失敗　《prcv_msgシステムコールのみ》

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

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リターンパラメータとして返します。

対象メイルボックスにメッセージが到着していない場合、本システムコール発行タスクをメッセージ到着を待つタスクの待ち行列につなぎます。つなぎ方は、FIFO(First-In First-Out)です。

prcv_msgシステムコールは、対象メイルボックスにメッセージが到着していない場合、エラーリターンします。（エラーコードとしてE_PLFAILを返します）

図３－３に受信メッセージの形式を示します。

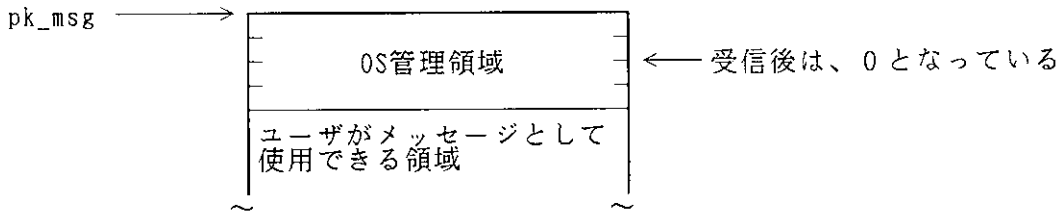


図３－３　受信メッセージの形式

【注】　メッセージの先頭４バイトはOSで使用するため、この部分は書き換えないでください。
書き換えた場合、再度メッセージを送信しようとしても動作は保証されません。

３．５．１０　mbx_sts (Get Mailbox Status) メイルボックス状態を参照する

［タスク部／非タスク部両用］

アセンブラフォーマット：

```
JSR     @mbx_sts
```

パラメータ：

R1　メイルボックスID(mbxid:1～255)

リターンパラメータ：

(1)H8/300Hアドバンストモード

R0　エラーコード(ercd)

ER2 メッセージの先頭アドレス(pk_msg)

R3　待ちタスクID(wtskid:0～255)

(2)H8/300Hノーマルモード

R0　エラーコード(ercd)

R2　メッセージの先頭アドレス(pk_msg)

R3　待ちタスクID(wtskid:0～255)

Cフォーマット：

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

パラメータ：

```
ID    mbxid;        メイルボックスID(mbxid:1～255)
```

リターンパラメータ：

```
ID    *p_wtskid;    待ちタスクID(wtskid:0～255)
typedef struct t_msg {
    UW msghead;     OS管理領域
    VB msgcont[];   メッセージの内容
} T_MSG **ppk_msg;  メッセージの先頭アドレスへのポインタ
```

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_NOEXS	H'F7CC(-H'834)	そのメイルボックスは生成されていない (mbxid=0,mbxid>メイルボックス定義数(最大255))

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

mbxid で示されるメイルボックスの状態を参照し、次に受信されるメッセージの先頭アドレス(pk_msg)、待ちタスクID(wtskid)をリターンパラメータとして返します。

次に受信されるメッセージの先頭アドレス(pk_msg)には、対象メイルボックスのメッセージの待ち行列の先頭につながれているメッセージの先頭アドレスを返します。メッセージが到着していない場合は、pk_msgにリターンパラメータとしてNADR(-1)を返します。

待ちタスクID(wtskid)には、対象メイルボックスのタスクの待ち行列の先頭のタスクIDを返します。待ちタスクがない場合は、NOTSK(H'0000)を返します。

３．６　割込み管理システムコール

３．６．１　ret_int (Return From Interrupt Handler) 割込みハンドラから復帰する　[非タスク部用]

アセンブラフォーマット：

```
JMP     @ret_int
```

パラメータ：

なし

リターンパラメータ：

正常終了時　システムコールを発行したコンテキストには戻らない

異常終了時　システム異常終了処理に制御を移します。

Cフォーマット：

なし

エラーコード(R0/ercd)：

返さない

解　説：

割込みハンドラからの復帰の際に発行するシステムコールです。

割込みハンドラの中でシステムコールを発行してもタスク切換えは起こらず、ret_int システムコールが発行されて割込みハンドラを抜けるまで、タスクの切換えが遅延されます。

なお、本システムコールにより復帰する場合、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。

【注】　タスク部から本システムコールを発行すると、異常終了処理に制御を移行します。

３．６．２　chg_ims (Change Interrupt Mask Level) 割込みマスクを変更する
［タスク部／非タスク部両用］

アセンブラフォーマット：

```
JSR     @chg_ims
```

パラメータ：

R1　割込みマスク(imask:0～3)

リターンパラメータ：

R0　エラーコード(ercd)

Cフォーマット：

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

パラメータ：

CCR　imask;　割込みマスク(imask:0～3)

リターンパラメータ：

なし

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_IMS	H'F8D5(-H'72B)	不正IMASK (割込みマスク(imask)0～3以外)

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

CPU のCCR （コンディションコードレジスタ）の割込みマスクビット（Ｉ、ＵＩビット）に直接値を設定して、割込みの禁止・解除を行ないます。

割込みマスク(imask)は０～３の整数値です。表３－１０にimaskの値と割込みマスク状態の関係を示します。

表３－１０　imask と割込みマスク状態の関係

項番	割込みﾏｽｸ (imask)	CCR設定値 (I,UIﾋﾞｯﾄ)	プライオリティレベル１の割込み（優先）	プライオリティレベル０の割込み（非優先）
1	3	I=1,UI=1	×（禁止）	×（禁止）
2	2	I=1,UI=0	○（許可）	×（禁止）
3	1	I=0,UI=1	○（許可）	○（許可）
4	0	I=0,UI=0	○（許可）	○（許可）

【注】　割込みマスク(imask)をカーネルの割込みマスクレベルよりも高いレベルに変更している間は、システムコールを発行しないでください。発行した場合は、システムの動作は保証されません。ただし、本システムコールを使用しての割込みマスク変更は、カーネルの割込みマスクレベルより高いレベルからでも可能です。

３．６．３　ims_sts (Get Interrupt Mask Level Status) 割込みマスクを参照する
［タスク部／非タスク部両用］

アセンブラフォーマット：

```
JSR     @ims_sts
```

パラメータ：

なし

リターンパラメータ：

R0　エラーコード(ercd)

R1　割込みマスク(imask:0～3)

Cフォーマット：

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

パラメータ：

なし

リターンパラメータ：

```
CCR     *p_imask;  割込みマスク(imask:0～3)
```

エラーコード(R0/ercd)：

```
E_OK        H'0000( H'000)  正常終了
```

解　説：

CPU のCCR （コンディションコードレジスタ）の割込みマスクビット（Ｉ、ＵＩビット）を参照し、割込みマスク(imask)をリターンパラメータとして返します。

表３－１１にimaskの値と割込みマスク状態の関係を示します。

表３－１１　imask と割込みマスク状態の関係

項番	割込みﾏｽｸ (imask)	CCR設定値 (I,UIﾋﾞｯﾄ)	プライオリティレベル１の割込み（優先）	プライオリティレベル０の割込み（非優先）
1	3	I=1,UI=1	×（禁止）	×（禁止）
2	2	I=1,UI=0	○（許可）	×（禁止）
3	1	I=0,UI=1	○（許可）	○（許可）
4	0	I=0,UI=0	○（許可）	○（許可）

３．７　メモリ管理システムコール

３．７．１　get_blk (Get Memory Block) 固定長メモリブロックの獲得待ちを行なう [タスク部用]

pget_blk (Poll and Get Memory Block) 固定長メモリブロックを獲得する [タスク部用]

アセンブラフォーマット：

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

パラメータ：

R1　固定長メモリプールID(mplid:1～255)

リターンパラメータ：

(1)H8/300Hアドバンストモード

R0　エラーコード(ercd)

ER2 メモリブロックの先頭アドレス(blk)

(2)H8/300Hノーマルモード

R0　エラーコード(ercd)

R2　メモリブロックの先頭アドレス(blk)

Cフォーマット：

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

パラメータ：

```
ID    mplid;   固定長メモリプールID(mplid:1～255)
```

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
```

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_NOEXS	H'F7CC(-H'834)	そのメモリプールは生成されていない (mplid=0,mplid>メモリプール定義数(最大255), メモリプール未登録)
*E_CTX	H'F5BB(-H'A45)	コンテキストエラー（非タスク部から発行できない）
E_RLWAI	H'F2AA(-H'D56)	待ち状態強制解除
E_PLFAIL	H'F1A7(-H'E59)	ポーリング失敗　《pget_blkシステムコールのみ》

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

mplid で示される固定長メモリプールから1つのメモリブロックを獲得します。

獲得したメモリブロックの先頭アドレスは、リターンパラメータとしてblk に返します。

指定したメモリプールからメモリブロックを獲得できない場合、本システムコールの発行タスクを待ち状態とし、そのメモリプールのメモリブロック獲得を待つタスクの待ち行列につなぎます。つなぎ方は、FIFO(First-In First-Out)です。

pget_blkシステムコールは、指定したメモリプールからメモリブロックを獲得できない場合、エラーリターンします。（エラーコードとしてE_PLFAILを返します）

3．7．2　rel_blk (Release Shared Memory Block) 固定長メモリブロックを返却する [タスク部用]

アセンブラフォーマット：

```
JSR     @rel_blk
```

パラメータ：

(1)H8/300Hアドバンストモード

R1　固定長メモリプールID(mplid:1～255)

ER2 メモリブロックの先頭アドレス(blk)

(2)H8/300Hノーマルモード

R1　固定長メモリプールID(mplid:1～255)

R2　メモリブロックの先頭アドレス(blk)

リターンパラメータ：

R0　エラーコード(ercd)

Cフォーマット：

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

パラメータ：

```
ID    mplid;  固定長メモリプールID(mplid:1～255)
VP    blk;    メモリブロックの先頭アドレス
```

リターンパラメータ：

なし

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_ILADR	H'F8DE(-H'722)	不正アドレス（メモリブロックアドレスが0または奇数）
*E_NOEXS	H'F7CC(-H'834)	そのメモリプールは生成されていない (mplid=0, mplid>メモリプール定義数(最大255), メモリプール未登録)
E_ILBLK	H'F7C5(-H'83B)	不正メモリブロック返却,操作
*E_CTX	H'F5BB(-H'A45)	コンテキストエラー（非タスク部から発行できない）

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

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

返却するメモリブロックが対象メモリプールに属さない場合には、エラーコードとしてE_ILBLKを返します。また、メモリブロックの先頭アドレスが0または奇数の場合、エラーとしてE_ILADRを返します。

３．７．３　mpl_sts (Get Memory Pool Status) メモリプール状態を参照する
［タスク部／非タスク部両用］

アセンブラフォーマット：

```
JSR     @mpl_sts
```

パラメータ：

R1　固定長メモリプールID(mplid:1～255)

リターンパラメータ：

R0　エラーコード(ercd)

R3　待ちタスクID(wtskid:0～255)

R4　空き領域のメモリブロック数(frbcnt)

Cフォーマット：

```
ER ercd=mpl_sts(ID *p_wtskid, UINT *p_frbcnt, ID mplid);
```

パラメータ：

ID	mplid;	固定長メモリプールID(mplid:1～255)

リターンパラメータ：

ID	*p_wtskid;	待ちタスクID(wtskid:0～255)
UINT	*p_frbcnt;	空き領域のメモリブロック数

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_NOEXS	H'F7CC(-H'834)	そのメモリプールは生成されていない (mplid=0,mplid>メモリプール定義数(最大255), メモリプール未登録)

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

mplid で示されるメモリプールの状態を参照し、対象メモリプールの現在の空き領域のメモリブロック数(frbcnt)、待ちタスクID(wtskid)をリターンパラメータとして返します。

待ちタスクID(wtskid)には、対象メモリプールのタスクの待ち行列の先頭のタスクIDを返します。待ちタスクがない場合は、NOTSK (H'0000)を返します。

３．８　時間管理システムコール

３．８．１　set_tim (Set Time)システムクロックを設定する　　[タスク部／非タスク部両用]

アセンブラフォーマット：

```
    JSR     @set_tim
```

パラメータ：

(1)H8/300Hアドバンストモード

ER2 現在の時間データへのポインタ(pk_time)

pk_timeで示されるパケットの内容

現在の時間データ上位(utime) 2バイト

現在の時間データ中位(mtime) 2バイト

現在の時間データ下位(ltime) 2バイト

(2)H8/300Hノーマルモード

R2　現在の時間データへのポインタ(pk_time)

pk_timeで示されるパケットの内容

現在の時間データ上位(utime) 2バイト

現在の時間データ中位(mtime) 2バイト

現在の時間データ下位(ltime) 2バイト

リターンパラメータ：

R0　エラーコード(ercd)

Cフォーマット：

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

パラメータ：

```
typedef struct t_tim {
    H  utime;       現在の時間データ〈上位〉(utime)
    UH mtime;       現在の時間データ〈中位〉(mtime)
    UH ltime;       現在の時間データ〈下位〉(ltime)
} T_TIM *pk_time;  現在の時間データへのポインタ
```

リターンパラメータ：

なし

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_ILADR	H'F8DE(-H'722)	不正アドレス（パケットの先頭アドレスが0または奇数）
*E_ILTIME	H'F8D9(-H'727)	不正時間指定（pk_timeで示される値が負）

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

システムが保持しているシステムクロックの現在の値を、pk_time で示されるパケットアドレス内の値に設定します。システムクロックのビット数は48 ビットです。

パケットの先頭アドレスが0または奇数のとき、エラーとしてE_ILADR を返します。また、pk_time で示される値が負である場合、エラーコードとしてE_ILTIME を返します。

３．８．２　get_tim (Get Time) システムクロックの値を読み出す　[タスク部／非タスク部両用]

アセンブラフォーマット：

```
    JSR     @get_tim
```

パラメータ：

(1)H8/300Hアドバンストモード

ER2 データ読み出し領域（６バイト）へのポインタ(pk_time)

(2)H8/300Hノーマルモード

R2　データ読み出し領域（６バイト）へのポインタ(pk_time)

リターンパラメータ：

(1)H8/300Hアドバンストモード

R0　エラーコード(ercd)

ER2 データ読み出し領域（６バイト）へのポインタ(pk_time)

pk_timeで示されるパケットの内容

現在の時間データ上位(utime) 2バイト

現在の時間データ中位(mtime) 2バイト

現在の時間データ下位(ltime) 2バイト

(2)H8/300Hノーマルモード

R0　エラーコード(ercd)

R2　データ読み出し領域（６バイト）へのポインタ(pk_time)

pk_timeで示されるパケットの内容

現在の時間データ上位(utime) 2バイト

現在の時間データ中位(mtime) 2バイト

現在の時間データ下位(ltime) 2バイト

Cフォーマット：

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

パラメータ：

なし

リターンパラメータ：

```
    typedef struct t_tim {
        H  utime;       現在の時間データ〈上位〉(utime)
        UH mtime;       現在の時間データ〈中位〉(mtime)
        UH ltime;       現在の時間データ〈下位〉(ltime)
    } T_TIM *pk_time;  現在の時間データへのポインタ
```

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_ILADR	H'F8DE(-H'722)	不正アドレス（パケットの先頭アドレスが0または奇数）

【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

システムが保持しているシステムクロックの現在の値を読み出し、pk_time で示されるパケットアドレス内にシステムクロック値を返します。システムクロックのビット数は48ビットです。

パケットの先頭アドレスが0または奇数のとき、エラーとしてE_ILADR を返します。

３．９　システム管理システムコール

３．９．１　get_ver (Get Version No) カーネルのバージョン識別子を得る
［タスク部／非タスク部両用］

アセンブラフォーマット：

```
JSR     @get_ver
```

パラメータ：

(1)H8/300Hアドバンストモード

ER2 バージョンデータ読み出し領域（２０バイト）へのポインタ(pk_ver)

(2)H8/300Hノーマルモード

R2　バージョンデータ読み出し領域（２０バイト）へのポインタ(pk_ver)

リターンパラメータ：

(1)H8/300Hアドバンストモード

R0　エラーコード(ercd)

ER2 バージョンデータ読み出し領域（２０バイト）へのポインタ(pk_ver)

pk_verで示されるパケットの内容

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

(2)H8/300Hノーマルモード

R0　エラーコード(ercd)

R2　バージョンデータ読み出し領域（２０バイト）へのポインタ(pk_ver)

pk_verで示されるパケットの内容

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

Cフォーマット：

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

パラメータ：

なし

リターンパラメータ：

```
typedef struct t_ver {
    UH maker;      メーカー(maker)
    UH id;         形式番号(id)
    UH spver;      仕様書バージョン(spver)
    UH prver;      製品バージョン(prver)
    UH prno[4];    製品管理情報(prno)
    UH cpu;        CPU情報(cpu)
    UH var;        バリエーション記述子(var)
} T_VER *pk_ver;   バージョンデータ読出し領域へのポインタ
```

エラーコード(R0/ercd)：

E_OK	H'0000(H'000)	正常終了
*E_ILADR	H'F8DE(-H'722)	不正アドレス(パケットの先頭アドレスが0または奇数)

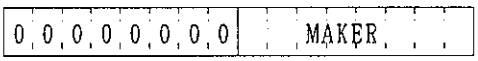
【注】 *システム構築時にパラメータチェック機能モジュールを組み込んでいない場合、このエラーは検出されませんので注意してください。

解　説：

現在使用中のH18-3Hのバージョンに関する諸情報をプログラムから読み出します。

get_verシステムコールで、得られる情報を以下に示します。

(maker)

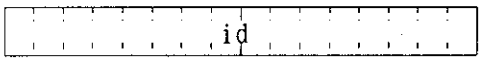


MAKER ： この製品を作ったメーカー

MAKER　HITACHI(H'000A)

H18-3HのMAKERの値はH'000Aです。

(id)



id ： ＯＳやＶＬＳＩの種類を区別する番号

１つのメーカーの中で、MMU 対応版とnon-MMU 対応版のような複数の製品を作った場合に、それらを区別するために使用する番号です。

H18-3H V2.0のidの値はH'0003です。

(spver)

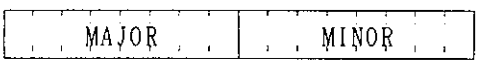
MAGIC : TRON仕様のシリーズを区別する番号
SpecVer : この製品のもとになった東京大学のTRON仕様書のバージョン番号。3桁のBCDコードで入れます。

MAGIC　μITRON 仕様(H'5)

SpecVer Ver 2.01(H'201)

HI8-3Hのspver の値はH'5201です。

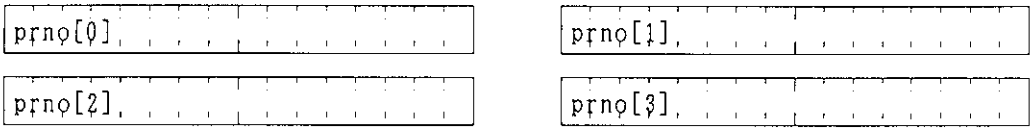
(prver)　内部インプリメント上のバージョン番号を表わします。



MAJOR : バージョン番号の大きな区別を表します
MINOR : バージョン番号の小さな区別を表します

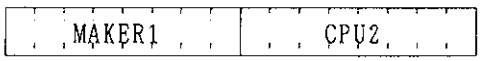
HI8-3Hのprverの値はH'0200です。

(prno)　prno[0]～prno[3] : 製品管理情報や製品番号などが入っています。



HI8-3Hのprno[0]からprno[3]の値はH'0000です。

(cpu)



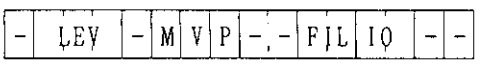
MAKER1 : (maker)で示した値と同じです。
CPU2 : このμITRONを実行するプロセッサを表します

MAKER1　HITACHI(H'0A)

CPU2　HI8-3HのCPU2の値はH'84とします。

HI8-3Hのcpuの値はH'0A84です。

(var)　このμITRONで利用できる機能の概要を表示します。



- : reserved(0が返る)
LEV : カーネル仕様のレベル分け
M : プロセッサ
V : 仮想記憶
P : MMU
FIL : ファイル仕様のレベル分け
IO : 入出力仕様

なお、製品の種類の区別はあくまでもidフィールドで行ないますので、var・cpuが異なり、かつidの等しいものが存在することはありません。

LEV　μITRON : 000
M　シングルプロセッサ用 : 0
V　仮想記憶未対応 : 0
P　MMU 未対応 : 0
FIL　サポートなし : 00
IO　サポートなし : 00

HI8-3Hのverの値はH'0000です。

４．Ｃ言語によるプログラムの作成方法

タスクや割込みハンドラなど、ユーザが作成するプログラムはすべてＣ言語で記述することができます。本章では、Ｈ８／３００シリーズ　Ｃコンパイラを用いてＣ言語プログラムを作成する方法を説明します。Ｈ８／３００シリーズ　Ｃコンパイラについての詳細は、「Ｈ８／３００シリーズ　Ｃコンパイラ　ユーザーズマニュアル」を参照してください。

４．１　Ｃ言語インタフェースライブラリ

HI8-3H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『h3hcif.lib』とＣ言語ヘッダファイル『hi8_3h.h』から構成されています。ライブラリファイル『h3hcif.lib』は、H8/300Hノーマルモード用とH8/300Hアドバンストモード用の２種類があります。

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、『hi8_3h.h』をソースプログラム中にインクルードし、HI8-3Hのシステム結合時に『h3hcif.lib』を結合してください。

表４－１にＣ言語インタフェースライブラリファイルの一覧を示します。

表４－１　Ｃ言語インタフェースライブラリファイル一覧

項番	ファイル名	内　　容
1	h3hcif.lib	システムコールのＣ言語インタフェースライブラリ (ret_int システムコール*を除いた42個のライブラリ) ｃｉｆａディレクトリのh3hcif.lib：H8/300Hアドバンストモード用 ｃｉｆｎディレクトリのh3hcif.lib：H8/300Hノーマルモード用
2	hi8_3h.h	Ｃ言語プログラム用ヘッダファイル(sampleaﾃﾞｨﾚｸﾄﾘ/samplenﾃﾞｨﾚｸﾄﾘ) (Ｃ言語プログラムで用いる型・定数・原型宣言)

【注】* ret_int システムコールは、Ｃコンパイラのアセンブル埋め込み機能 #pragma を使用して発行しますので、Ｃ言語インタフェースライブラリは提供されません。
詳細については、「４．３．１　割込みハンドラの記述方法」を参照してください。

Ｃ言語で記述するプログラムは、次の２種類に分けられます。

Ｃ言語で記述したプログラムは、コンパイル時のオブジェクト指定により、H8/300Hノーマルモード、H8/300Hアドバンストモードのどちらにも使用できます。

(1) タスク

Ｃ言語のみで記述します。

(2) ハンドラ

Ｃ言語とアセンブル埋め込み機能（アセンブリ言語）で記述します。

４．２　Ｃ言語によるタスクの記述方法

Ｃ言語でタスクを記述する場合は、Ｃ言語のみで記述することができます。

Ｃ言語でのタスクの記述例を以下に示します。

タスクの最後でext_tskシステムコールを発行し、終了を行ないます。終了を行なわないと、タスク終了後の制御が不定となり、システムの正常な動作は、保証されません。

タスクは、return文を使用してリターン値を返すことはできません。

```
#include "hi8_3h.h"

taskXX()
{
ID subtid;
ER ercd;

/*
 *              ...
 *              ...
 *              ...
 */

   ercd = wup_tsk(subtid);

 /*
  *             ...
  *             ...
  *             ...
  */

   ext_tsk();
 }
```

図４－１　Ｃ言語でのタスクの記述例

４．３　Ｃ言語によるハンドラの記述方法

Ｃ言語でハンドラを記述する場合、スタックポインタの切替えやレジスタの保証が必要となるため、アセンブル埋め込み機能(#pragma)を使用しなければなりません。

アセンブル埋め込み機能を使用して記述するハンドラを以下に示します。

(1) 割込みハンドラ
(2) システム初期化ハンドラ　（ラベル名 『_HIPRG_SYSINI』）
(3) HI8-3Hのシステム異常終了ルーチン　（ラベル名 『_HIPRG_ABNOML』）

タイマドライバもアセンブル埋め込み機能を使用し、Ｃ言語で作成することが可能です。

４．３．１　割込みハンドラの記述方法

割込みハンドラの起動時には、以下の処理が必要になります。

(1) 割込みハンドラ専用スタックへの切替え
(2) レジスタの保証

割込みハンドラの終了時には、以下の処理が必要になります。

(1) レジスタの回復
(2) スタックポインタの回復
(3) ret_int システムコールの発行

C言語での割込みハンドラの記述例を以下に示します。

```
#include "hi8_3h.h"

void INTxx(void)
#pragma asm
LEVEL:  .equ    2                                ;:ﾌﾟﾗｲｵﾘﾃｨﾚﾍﾞﾙ0の割込みﾚﾍﾞﾙを
                                                 ;:定義します
        .IMPORT _sp_savxx                        ;:割込み発生時のSP退避領域の外部
                                                 ;:参照宣言
        .IMPORT _stk_intxx                       ;:割込み用ｽﾀｯｸ領域の外部参照宣言
        .IMPORT ret_int                          ;:ret_intｼｽﾃﾑｺｰﾙの外部参照宣言

        ldc     #LEVEL*h'40:8,ccr                ;:割込みﾏｽｸﾚﾍﾞﾙをccrに設定します
        mov.l   sp,@_sp_savxx                    ;:SPを@_sp_savxxに退避します
        mov.l   #_stk_intxx,sp                   ;:SPを割込みｽﾀｯｸに切り替えます
        push.l  er0                              ;:ER0をｽﾀｯｸに退避します
        push.l  er1                              ;:ER1をｽﾀｯｸに退避します
        bsr     intxx_main:8                     ;:関数INTxxの戻りｱﾄﾞﾚｽとして,
;:                                               ;:ｱﾄﾞﾚｽINTをｽﾀｯｸに設定します
INT:
        pop.l   er1                              ;:ER1を回復します
        pop.l   er0                              ;:ER0を回復します
        mov.l   @_sp_savxx,sp                    ;:@_sp_savxxの値をSPに回復します
        jmp     @ret_int                         ;:ret_intｼｽﾃﾑｺｰﾙを発行します
intxx_main:
#pragma endasm
{
        iwup_tsk(10);                            /*iwup_tskｼｽﾃﾑｺｰﾙを発行します*/
}
```

図４－２　Ｃ言語での割込みハンドラの記述例(H8/300H/-ﾏﾙ,ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ共通)

４．３．２　システム初期化ハンドラの記述方法

システム初期化ハンドラの起動時には、以下の処理が必要になります。

(1) システム初期化ハンドラ専用スタックへの切替え
(2) レジスタの保証

システム初期化ハンドラの終了時には、以下の処理が必要になります。

(1) レジスタの回復
(2) スタックポインタの回復
(3) RTS 命令の発行

Ｃ言語で記述されたシステム初期化ハンドラのアセンブラ埋め込み機能部分は、最後がret_intシステムコールの発行から、rts 命令の実行になっている以外は割込みハンドラと同じです。

４．３．３　HI8-3Hのシステム異常終了ルーチンの記述方法

HI8-3Hのシステム異常終了ルーチンの起動時には、以下の処理が必要になります。

(1) HI8-3Hのシステム異常終了ルーチン専用スタックへの切替え
(2) レジスタの保証

HI8-3Hのシステム異常終了ルーチンの作り方によって変ります。
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部分を記述してください。

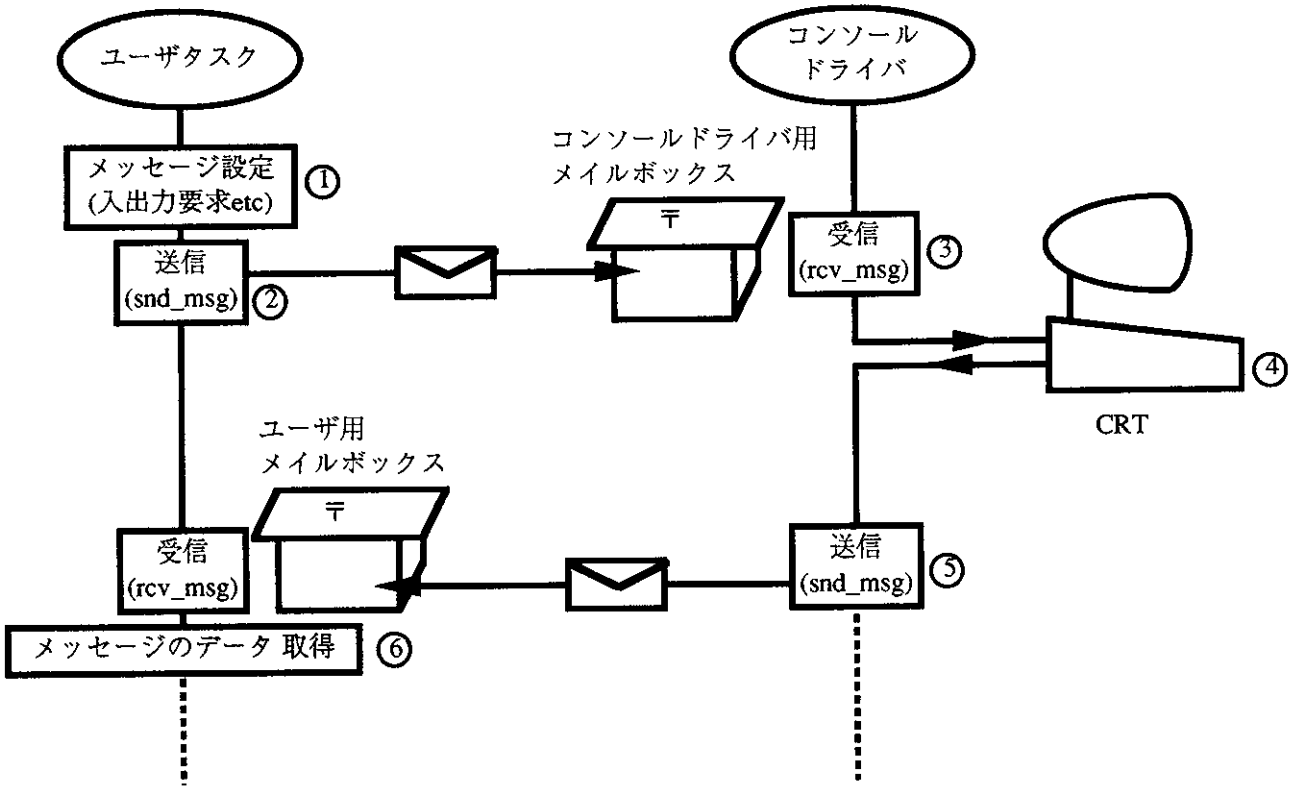
付録A．コンソールドライバ

H8-3Hは、H8/3003およびH8/3042シリーズ内蔵のシリアルコミュニケーションインタフェース(SCI:Serial Communication Interface 以下SCIと略す）を用いたコンソールドライバのサンプルを提供しています。

A．1　概　要

コンソールドライバは、ユーザプログラムからの入出力要求にしたがいコンソールへの入出力処理を行ないます。

以下にコンソールドライバの概要を説明します。



図A－1　コンソールドライバ入出力処理の概要

コンソールドライバとユーザプログラムは、メイルボックスを介して接続されます。

コンソールドライバは専用のメイルボックスを1つ持ち、ユーザプログラムはタスクごとにメイルボックスを1つ持たなければなりません。

Ａ．１．１　コンソールドライバの処理

ユーザプログラム（タスク）によるコンソール入出力処理は、以下の手順で行ないます。

(1) ユーザプログラム：メッセージ設定（図Ａ－１の①）

ユーザプログラムが、メッセージに入出力要求のデータを設定します。

ユーザプログラムは、ファンクションコードに機能コード（入力要求または出力要求）および終了報告のためのユーザメイルボックスIDを設定します。

出力要求の場合は、さらに終端にNULLコード(H'00)が設定された文字列の先頭アドレスを設定します。

(2) ユーザプログラム：入出力要求送信（図Ａ－１の②）

ユーザプログラムは、コンソールドライバのメイルボックスに対し、snd_msg システムコールを発行し、入出力要求を行ないます。

(3) コンソールドライバ：入出力要求受信（図Ａ－１の③）

コンソールドライバは、rcv_msg システムコールを発行し、ユーザプログラムからの入出力要求（メッセージ）を受信します。

メッセージがまだ到着していない場合、コンソールドライバはWAIT状態に移行し、メッセージの到着を待ちます。

(4) コンソールドライバ：入出力処理（図Ａ－１の④）

受信したメッセージの内容にしたがい、コンソールドライバはコンソールに対して入出力処理を行ないます。

・入力要求の場合

入力要求では、メッセージの先頭から４バイト目にキーボードからの入力データ（１文字分）、５バイト目にSCI のSSR （ステータスレジスタ）の値を設定します。

・出力要求の場合

出力要求では、出力データに設定された文字をNULLコード(H'00)が検出されるまでCRT へ出力します。

(5) コンソールドライバ：処理結果送信（図Ａ－１の⑤）

コンソールドライバは、コンソールに対する処理結果をメッセージに設定し、ユーザプログラムのメイルボックスに対し、snd_msg システムコールを発行します。

(6) ユーザプログラム：処理結果受信（図Ａ－１の⑥）

ユーザプログラムは、rcv_msg システムコールを発行し、コンソールドライバからのメッセージを受信した後、処理結果にしたがってタスクの処理を行ないます。

Ａ．２　コンソールドライバの機能

コンソールドライバは、以下の機能をサポートしています。

(1) キー入力データの格納（１文字入力）

キー入力された文字を読み込みます。（エコーバックは行いません）

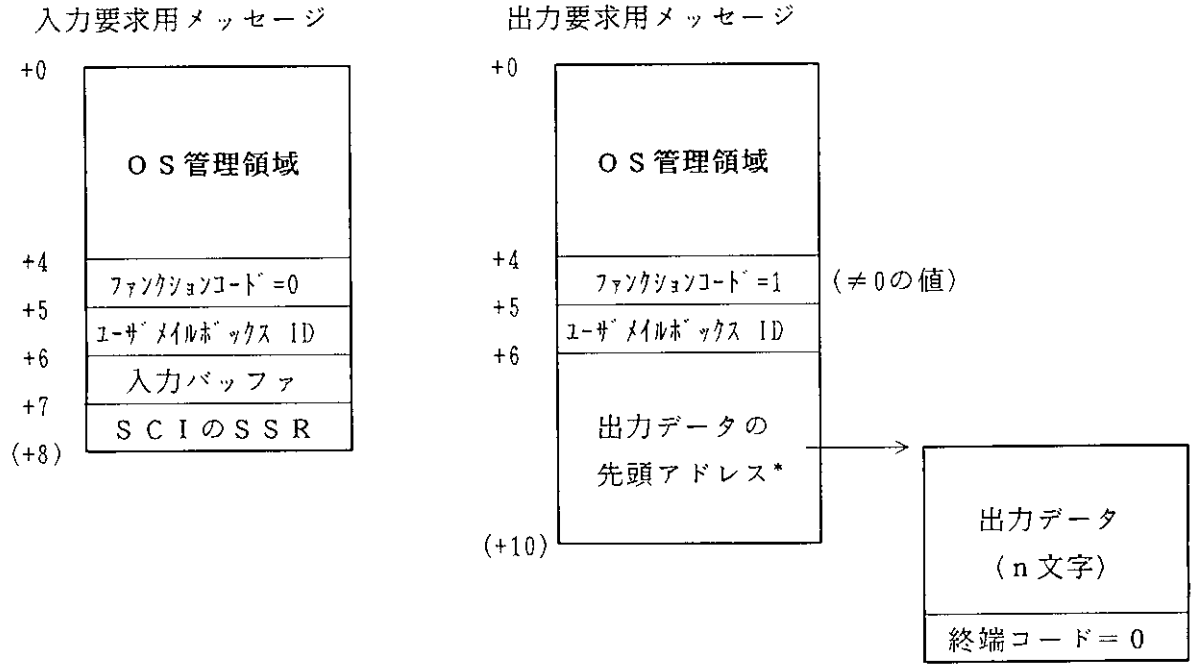
(2) ＣＲＴへの文字表示（ｎ文字出力）

指定された終端がNULLコード(H'00)の文字列を、CRT に表示します。

Ａ．２．１　メッセージフォーマット

コンソールドライバとユーザプログラムとの入出力要求と入出力完了は、メッセージを介して行ないます。

以下にメッセージのフォーマットを示します。



【注】＊　H8/300Hノーマルモードの場合、下位16ビットが出力データの先頭アドレスになります。

図Ａ－２　コンソールドライバとユーザプログラム間のメッセージフォーマット

A．2．2　コンソールドライバの構成

サンプルとして提供するコンソールドライバのソースファイルに対応した、タスク名、割込みハンドラ名、セクション名、資源などの構成情報を示します。

表A－1，表A－2に、コンソールドライバの構成を示します。

表A－1　コンソールドライバの構成(H8/300Hｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ：sampleaﾃﾞｨﾚｸﾄﾘ)

項番	名称	ｺﾝｿｰﾙﾄﾞﾗｲﾊﾞ 0	ｺﾝｿｰﾙﾄﾞﾗｲﾊﾞ 1
1	ファイル名	h3hcns_0.mar	h3hcns_1.mar
2	使用するSCI	H8/3003のSCI0	H8/3003のSCI1
3	タスク名	H_CNSDRV_0	H_CNSDRV_1
4	タスクID	構築時に決定	構築時に決定
5	メイルボックスID	タスクIDと同じ	タスクIDと同じ
6	ERI の割込みハンドラ名	H_CNSHDL0ER	H_CNSHDL1ER
7	RXI の割込みハンドラ名	H_CNSHDL0RX	H_CNSHDL1RX
8	TXI の割込みハンドラ名	H_CNSHDL0TX	H_CNSHDL1TX
9	コードのセクション名	h3hcns	h3hcns
10	データのセクション名	h3hcns_ram	h3hcns_ram
11	CPU=16MHz時のボーレート	9600bps	9600bps

表A－2　コンソールドライバの構成(H8/300Hﾉｰﾏﾙﾓｰﾄﾞ：samplenﾃﾞｨﾚｸﾄﾘ)

項番	名称	ｺﾝｿｰﾙﾄﾞﾗｲﾊﾞ 0	ｺﾝｿｰﾙﾄﾞﾗｲﾊﾞ 1
1	ファイル名	h3hcnsn0.mar	h3hcnsn1.mar
2	使用するSCI	H8/3042のSCI0	H8/3042のSCI1
3	タスク名	H_CNSDRV_0	H_CNSDRV_1
4	タスクID	構築時に決定	構築時に決定
5	メイルボックスID	タスクIDと同じ	タスクIDと同じ
6	ERI の割込みハンドラ名	H_CNSHDL0ER	H_CNSHDL1ER
7	RXI の割込みハンドラ名	H_CNSHDL0RX	H_CNSHDL1RX
8	TXI の割込みハンドラ名	H_CNSHDL0TX	H_CNSHDL1TX
9	コードのセクション名	h3hcns	h3hcns
10	データのセクション名	h3hcns_ram	h3hcns_ram
11	CPU=16MHz時のボーレート	9600bps	9600bps

付録Ｂ．システムコール一覧

Ｂ．１　ＨＩ８－３Ｈシステムコール・アセンブラインタフェース一覧

表中の意味を以下に示します。

ノーマル／アドバンスト(N/A)　：使用するCPU動作モードを示します
　　N:H8/300Hノーマルモード
　　A:H8/300Hアドバンストモード
＝　　：リターンパラメータに使用(リターン値が返される)
タスク／非タスク(T/N)　　：発行できるシステム状態を示します
　　T:タスク部から発行できるシステムコール
　　N:非タスク部から発行できるシステムコール

システムコール	ﾉｰﾏﾙ/ｱﾄﾞﾊﾞﾝｽﾄ (N/A)	パラメータ／リターンパラメータ(=) R0	R1	R2	R3	R4	ER2	ﾀｽｸ/非ﾀｽｸ (T/N)
タスク管理								
1 sta_tsk	N/A	ercd=	tskid					T
2 ista_tsk	N/A	ercd=	tskid					N
3 ext_tsk	N/A							T
4 ter_tsk	N/A	ercd=	tskid					T
5 chg_pri	N/A	ercd=	tskid	tskpri				T
6 rot_rdq	N/A	ercd=		tskpri				T
7 irot_rdq	N/A	ercd=		tskpri				N
8 rel_wai	N/A	ercd=	tskid					T
9 get_tid	N/A	ercd=	tskid=					T
10 tsk_sts	N/A	ercd=	tskid	tskpri=	tskstat=			T/N
タスク付属同期管理								
11 sus_tsk	N/A	ercd=	tskid					T
12 rsm_tsk	N/A	ercd=	tskid					T
13 slp_tsk	N/A	ercd=						T
14 wai_tsk	N/A	ercd=					tmout	T
15 wup_tsk	N/A	ercd=	tskid					T
16 iwup_tsk	N/A	ercd=	tskid					N
17 can_wup	N/A	ercd=	tskid	wupcnt=				T

システムコール	ﾉｰﾏﾙ/ｱﾄﾞﾊﾞﾝｽﾄ (N/A)	パラメータ／リターンパラメータ(=) R0	R1	R2	R3	R4	ER2	ﾀｽｸ/非ﾀｽｸ (T/N)
同期／通信管理								
18 set_flg	N/A	ercd=	flgid	setptn				T
19 iset_flg	N/A	ercd=	flgid	setptn				N
20 clr_flg	N/A	ercd=	flgid	clrptn				T/N
21 wai_flg	N/A	ercd=	flgid	waiptn	wfmode			T
			flgptn=					
22 pol_flg	N/A	ercd=	flgid	waiptn	wfmode			T
			flgptn=					
23 flg_sts	N/A	ercd=	flgid	flgptn=	wtskid=			T/N
24 sig_sem	N/A	ercd=	semid					T
25 isig_sem	N/A	ercd=	semid					N
26 wai_sem	N/A	ercd=	semid					T
27 preq_sem	N/A	ercd=	semid					T
28 sem_sts	N/A	ercd=	semid	semcnt=	wtskid=			T/N
29 snd_msg	N	ercd=	mbxid	pk_msg				T
snd_msg	A	ercd=	mbxid				pk_msg	T
30 isnd_msg	N	ercd=	mbxid	pk_msg				N
isnd_mag	A	ercd=	mbxid				pk_msg	N
31 rcv_msg	N	ercd=	mbxid	pk_msg=				T
rcv_msg	A	ercd=	mbxid				pk_msg=	T
32 prcv_msg	N	ercd=	mbxid	pk_msg=				T
prcv_msg	A	ercd=	mbxid				pk_msg=	T
33 mbx_sts	N	ercd=	mbxid	pk_msg=		wtskid=		T/N
mbx_sts	A	ercd=	mbxid			wtskid=	pk_msg=	T/N
割込み管理								
34 ret_int	N/A							N
35 chg_ims	N/A	ercd=	imask					T/N
36 ims_sts	N/A	ercd=	imask=					T/N

システムコール	ﾉｰﾏﾙ/ｱﾄﾞﾊﾞﾝｽﾄ (N/A)	パラメータ／リターンパラメータ(=) R0	R1	R2	R3	R4	ER2	ﾀｽｸ/非ﾀｽｸ (T/N)
メモリ管理								
37 get_blk	N	ercd=	mplid	blk=				T
get_blk	A	ercd=	mplid				blk=	T
38 pget_blk	N	ercd=	mplid	blk=				T
pget_blk	A	ercd=	mplid				blk=	T
39 rel_blk	N	ercd=	mplid	blk				T
rel_blk	A	ercd=	mplid				blk	T
40 mpl_sts	N/A	ercd=	mplid		wtskid=	frbcnt=		T/N
時間管理								
41 set_tim	N	ercd=		pk_time				T/N
set_tim	A	ercd=					pk_time	T/N
42 get_tim	N	ercd=		pk_time				T/N
get_tim	A	ercd=					pk_time	T/N
システム管理								
43 get_ver	N	ercd=		pk_ver				T/N
get_ver	A	ercd=					pk_ver	T/N

【注】　システムコールのエラーコードは、レジスタR0に返されます。ただし、ext_tsk システムコール,ret_intシステムコールは、エラーコードを返さず、システム異常終了処理に制御を移行します。

B．2　ＨＩ８－３Ｈシステムコール実例集(アセンブラフォーマット)

システムコール	機　　能	システムコール発行方法		パラメータ
タスク管理				
1 sta_tsk	タスクを起動する (ﾉｰﾏﾙ/ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用)	MOV.W JSR	#tskid, R1 @sta_tsk	tskid:タスクID
2 ista_tsk	タスクを起動する (ﾉｰﾏﾙ/ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用)	MOV.W JSR	#tskid, R1 @ista_tsk	tskid:タスクID
3 ext_tsk	自タスクを正常終了する (ﾉｰﾏﾙ/ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用)	JMP	@ext_tsk	
4 ter_tsk	他タスクを強制的に 異常終了させる (ﾉｰﾏﾙ/ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用)	MOV.W JSR	#tskid, R1 @ter_tsk	tskid:タスクID
5 chg_pri	タスク優先度を変更 する (ﾉｰﾏﾙ/ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用)	MOV.W MOV.W JSR	#tskid, R1 #tskpri, R2 @chg_pri	tskid:タスクID tskpri:タスク優先度
6 rot_rdq	タスクのレディキューを 回転する (ﾉｰﾏﾙ/ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用)	MOV.W JSR	#tskpri, R2 @rot_rdq	tskpri:タスク優先度
7 irot_rdq	タスクのレディキューを 回転する (ﾉｰﾏﾙ/ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用)	MOV.W JSR	#tskpri, R2 @irot_rdq	tskpri:タスク優先度
8 rel_wai	タスクの待ち状態を 強制解除する (ﾉｰﾏﾙ/ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用)	MOV.W JSR	#tskid, R1 @rel_wai	tskid:タスクID
9 get_tid	自タスクのIDを得る (ﾉｰﾏﾙ/ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用)	JSR	@get_tid	

システムコール	機　　能	システムコール発行方法		パラメータ
10 tsk_sts	タスクの状態を見る (ノーマル/アドバンストモード用)	MOV.W JSR	#tskid, R1 @tsk_sts	tskid:タスクID
タスク付属同期管理				
11 sus_tsk	他タスクを強制待ち状態へ移行する (ノーマル/アドバンストモード用)	MOV.W JSR	#tskid, R1 @sus_tsk	tskid:タスクID
12 rsm_tsk	強制待ち状態のタスクを再開する (ノーマル/アドバンストモード用)	MOV.W JSR	#tskid, R1 @rsm_tsk	tskid:タスクID
13 slp_tsk	自タスクを待ち状態へ移行する (ノーマル/アドバンストモード用)	JSR	@slp_tsk	
14 wai_tsk	自タスクを一定時間待ち状態に移行する (ノーマル/アドバンストモード用)	MOV.L JSR	#tmout, ER2 @wai_tsk	tmout:タイムアウト指定
15 wup_tsk	待ち状態のタスクを起床する (ノーマル/アドバンストモード用)	MOV.W JSR	#tskid, R1 @wup_tsk	tskid:タスクID
16 iwup_tsk	待ち状態のタスクを起床する (ノーマル/アドバンストモード用)	MOV.W JSR	#tskid, R1 @iwup_tsk	tskid:タスクID
17 can_wup	タスクの起床要求を無効にする (ノーマル/アドバンストモード用)	MOV.W JSR	#tskid, R1 @can_wup	tskid:タスクID

システムコール	機　　能	システムコール発行方法		パラメータ
同期／通信管理				
18 set_flg	イベントフラグを セットする (ノーマル/アドバンストモード用)	MOV.W MOV.W JSR	#flgid, R1 #setptn, R2 @set_flg	flgid:イベントフラグID setptn:セットするビット パターン
19 iset_flg	イベントフラグをセット する (ノーマル/アドバンストモード用)	MOV.W MOV.W JSR	#flgid, R1 #setptn, R2 @iset_flg	flgid:イベントフラグID setptn:セットするビット パターン
20 clr_flg	イベントフラグをクリア する (ノーマル/アドバンストモード用)	MOV.W MOV.W JSR	#flgid, R1 #clrptn, R2 @clr_flg	flgid:イベントフラグID clrptn:クリアするビット パターン
21 wai_flg	イベントフラグを待つ (ノーマル/アドバンストモード用)	MOV.W MOV.W MOV.W JSR	#flgid, R1 #waiptn, R2 #wfmode, R3 @wai_flg	flgid:イベントフラグID waiptn:待ちビットパター ン wfmode:待ちモード
22 pol_flg	イベントフラグを得る (ノーマル/アドバンストモード用)	MOV.W MOV.W MOV.W JSR	#flgid, R1 #waiptn, R2 #wfmode, R3 @pol_flg	flgid:イベントフラグID waiptn:待ちビットパター ン wfmode:待ちモード
23 flg_sts	イベントフラグ状態を 参照する (ノーマル/アドバンストモード用)	MOV.W JSR	#flgid, R1 @flg_sts	flgid:イベントフラグID
24 sig_sem	セマフォに対するV操作 (ノーマル/アドバンストモード用)	MOV.W JSR	#semid, R1 @sig_sem	semid:セマフォID

システムコール	機　　能	システムコール発行方法		パラメータ
25 isig_sem	セマフォに対するV操作 (ノーマル/アドバンストモード用)	MOV.W JSR	#semid, R1 @isig_sem	semid:セマフォID
26 wai_sem	セマフォに対するP操作 (ノーマル/アドバンストモード用)	MOV.W JSR	#semid, R1 @wai_sem	semid:セマフォID
27 preq_sem	セマフォ資源を得る (ノーマル/アドバンストモード用)	MOV.W JSR	#semid, R1 @preq_sem	semid:セマフォID
28 sem_sts	セマフォ状態を参照する (ノーマル/アドバンストモード用)	MOV.W JSR	#semid, R1 @sem_sts	semid:セマフォID
29 snd_msg	メッセージを送信する (ノーマルモード用)	MOV.W MOV.W JSR	#mbxid, R1 #pk_msg, R2 @snd_msg	mbxid:メイルボックスID pk_msg:メッセージの先頭アドレス
snd_msg	メッセージを送信する (アドバンストモード用)	MOV.W MOV.L JSR	#mbxid, R1 #pk_msg, ER2 @snd_msg	mbxid:メイルボックスID pk_msg:メッセージの先頭アドレス
30 isnd_msg	メッセージを送信する (ノーマルモード用)	MOV.W MOV.W JSR	#mbxid, R1 #pk_msg, R2 @isnd_msg	mbxid:メイルボックスID pk_msg:メッセージの先頭アドレス
isnd_msg	メッセージを送信する (アドバンストモード用)	MOV.W MOV.L JSR	#mbxid, R1 #pk_msg, ER2 @isnd_msg	mbxid:メイルボックスID pk_msg:メッセージの先頭アドレス
31 rcv_msg	メッセージの受信を待つ (ノーマルモード用)	MOV.W JSR	#mbxid, R1 @rcv_msg	mbxid:メイルボックスID
rcv_msg	メッセージの受信を待つ (アドバンストモード用)	MOV.W JSR	#mbxid, R1 @rcv_msg	mbxid:メイルボックスID

システムコール	機　　能	システムコール発行方法		パラメータ
32 prcv_msg	メッセージを受信する (ノーマルモード用)	MOV.W JSR	#mbxid, R1 @prcv_msg	mbxid:メイルボックスID
prcv_msg	メッセージを受信する (アドバンストモード用)	MOV.W JSR	#mbxid, R1 @prcv_msg	mbxid:メイルボックスID
33 mbx_sts	メイルボックス状態を参照する (ノーマル/アドバンストモード用)	MOV.W JSR	#mbxid, R1 @mbx_sts	mbxid:メイルボックスID
割込み管理				
34 ret_int	割込みハンドラから復帰する (ノーマル/アドバンストモード用)	JMP	@ret_int	
35 chg_ims	割込みマスクを変更する (ノーマル/アドバンストモード用)	MOV.W JSR	#imask, R1 @chg_ims	imask:割込みマスク
36 ims_sts	割込みマスクを参照する (ノーマル/アドバンストモード用)	JSR	@ims_sts	
メモリ管理				
37 get_blk	固定長メモリブロックの獲得待ちを行なう (ノーマルモード用)	MOV.W JSR	#mplid, R1 @get_blk	mplid:固定長メモリプールID
get_blk	固定長メモリブロックの獲得待ちを行なう (アドバンストモード用)	MOV.W JSR	#mplid, R1 @get_blk	mplid:固定長メモリプールID

システムコール	機　　能	システムコール発行方法		パラメータ
38 pget_blk	固定長メモリブロックを獲得する (ﾉｰﾏﾙﾓｰﾄﾞ用)	MOV.W JSR	#mplid, R1 @pget_blk	mplid:固定長メモリプールID
pget_blk	固定長メモリブロックを獲得する (ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用)	MOV.W JSR	#mplid, R1 @pget_blk	mplid:固定長メモリプールID
39 rel_blk	固定長メモリブロックを返却する (ﾉｰﾏﾙﾓｰﾄﾞ用)	MOV.W MOV.W JSR	#mplid, R1 #blk, R2 @rel_blk	mplid:固定長メモリプールID blk:メモリブロックの先頭アドレス
rel_blk	固定長メモリブロックを返却する (ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用)	MOV.W MOV.L JSR	#mplid, R1 #blk, ER2 @rel_blk	mplid:固定長メモリプールID blk:メモリブロックの先頭アドレス
40 mpl_sts	メモリプール状態を参照する (ﾉｰﾏﾙ/ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用)	MOV.W JSR	#mplid, R1 @mpl_sts	mplid:固定長メモリプールID
時間管理				
41 set_tim	システムクロックを設定する (ﾉｰﾏﾙﾓｰﾄﾞ用)	MOV.W JSR	#pk_time, R2 @set_tim	pk_time:設定する時間データへのポインタ
set_tim	システムクロックを設定する (ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用)	MOV.L JSR	#pk_time, ER2 @set_tim	pk_time:設定する時間データへのポインタ

システムコール		機　　能	システムコール発行方法		パラメータ
42	get_tim	システムクロックの値を読みだす (ﾉｰﾏﾙﾓｰﾄﾞ用)	MOV.W JSR	#pk_time, R2 @get_tim	pk_time:データ読み出し領域へのポインタ
	get_tim	システムクロックの値を読みだす (ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用)	MOV.L JSR	#pk_time, ER2 @get_tim	pk_time:データ読み出し領域へのポインタ
システム管理					
43	get_ver	カーネルのバージョン識別子を得る (ﾉｰﾏﾙﾓｰﾄﾞ用)	MOV.W JSR	#pk_ver, R2 @get_ver	pk_ver:バージョンデータ読み出し領域へのポインタ
	get_ver	カーネルのバージョン識別子を得る (ｱﾄﾞﾊﾞﾝｽﾄﾓｰﾄﾞ用)	MOV.L JSR	#pk_ver, ER2 @get_ver	pk_ver:バージョンデータ読み出し領域へのポインタ

タスク管理

```
 1 ER ercd=  sta_tsk(ID tskid);
 2 ER ercd= ista_tsk(ID tskid);
 3           ext_tsk();
 4 ER ercd=  ter_tsk(ID tskid);
 5 ER ercd=  chg_pri(ID tskid, TPRI tskpri);
 6 ER ercd=  rot_rdq(TPRI tskpri);
 7 ER ercd= irot_rdq(TPRI tskpri);
 8 ER ercd=  rel_wai(ID tskid);
 9 ER ercd=  get_tid(ID *p_tskid);
10 ER ercd=  tsk_sts(UINT *p_tskstat, TPRI *p_tskpri, ID tskid);
```

タスク付属同期管理

```
11 ER ercd=  sus_tsk(ID tskid);
12 ER ercd=  rsm_tsk(ID tskid);
13 ER ercd=  slp_tsk();
14 ER ercd=  wai_tsk(TMO tmout);
15 ER ercd=  wup_tsk(ID tskid);
16 ER ercd= iwup_tsk(ID tskid);
17 ER ercd=  can_wup(INT *p_wupcnt, ID tskid);
```

同期／通信

```
18 ER ercd=  set_flg(ID flgid, UINT setptn);
19 ER ercd= iset_flg(ID flgid, UINT setptn);
20 ER ercd=  clr_flg(ID flgid, UINT clrptn);
21 ER ercd=  wai_flg(UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode);
22 ER ercd=  pol_flg(UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode);
23 ER ercd=  flg_sts(ID *p_wtskid, UINT *p_flgptn, ID flgid);
24 ER ercd=  sig_sem(ID semid);
25 ER ercd= isig_sem(ID semid);
26 ER ercd=  wai_sem(ID semid);
27 ER ercd= preq_sem(ID semid);
28 ER ercd=  sem_sts(ID *p_wtskid, UINT *p_semcnt, ID semid);
```

```
29 ER ercd=  snd_msg(ID mbxid, T_MSG *pk_msg);
30 ER ercd= isnd_msg(ID mbxid, T_MSG *pk_msg);
31 ER ercd=  rcv_msg(T_MSG **ppk_msg, ID mbxid);
32 ER ercd= prcv_msg(T_MSG **ppk_msg, ID mbxid);
33 ER ercd=  mbx_sts(ID *p_wtskid, T_MSG **ppk_msg, ID mbxid);
```

割込み管理

```
34 ER ercd=  chg_ims(CCR imask);
35 ER ercd=  ims_sts(CCR *p_imask);
```

メモリ管理

```
36 ER ercd=  get_blk(VP *p_blk, ID mplid);
37 ER ercd= pget_blk(VP *p_blk, ID mplid);
38 ER ercd=  rel_blk(ID mplid, VP blk);
39 ER ercd=  mpl_sts(ID *p_wtskid, UINT *p_frbcnt, ID mplid);
```

時間管理

```
40 ER ercd=  set_tim(T_TIM *pk_time);
41 ER ercd=  get_tim(T_TIM *pk_time);
```

システム管理

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

1.sta_tsk　タスクを起動する

```
#include "hi8_3h.h"
/*
 *  1.STA_TSK :
 */
void tsk_XX01()
{
ER ercd;
ID tskid;
    /*...            */
    ercd = sta_tsk(tskid);
    /*...            */
}
```

2.ista_tsk　タスクを起動する（非タスク部専用）

```
#include "hi8_3h.h"
/*
 *  2.ISTA_TSK :
 */
void int_XX02()
{
ER ercd;
ID tskid;
    /*...            */
    ercd = ista_tsk(tskid);
    /*...            */
}
```

3.ext_tsk　自タスクを正常終了する

```
#include "hi8_3h.h"
/*
 *  3.EXT_TSK :
 */
void tsk_XX03()
{
    /*...            */
    ext_tsk();
}
```

4.ter_tsk 他タスクを強制的に異常終了させる

```
#include "hi8_3h.h"
/*
 *  4.TER_TSK :
 */
void tsk_XX04()
{
ER ercd;
ID tskid;
    /*...            */
    ercd = ter_tsk(tskid);
    /*...            */
}
```

5.chg_pri　タスク優先度を変更する

```
#include "hi8_3h.h"
/*
 *  5.CHG_PRI :
 */
void tsk_XX05()
{
ER ercd;
ID tskid;
TPRI tskpri;
    /*...            */
    ercd = chg_pri(tskid, tskpri);
    /*...            */
}
```

6.rot_rdq タスクのレディキューを回転する

```
#include "hi8_3h.h"
/*
 *  6.ROT_RDQ :
 */
void tsk_XX06()
{
ER ercd;
TPRI tskpri;
    /*...            */
    ercd = rot_rdq(tskpri);
    /*...            */
}
```

7.irot_rdq タスクのレディキューを回転する（非タスク部専用）

```
#include "hi8_3h.h"
/*
 *  7.IROT_RDQ :
 */
void int_XX07()
{
ER ercd;
TPRI tskpri;
    /*...              */
    ercd = irot_rdq(tskpri);
    /*...              */
}
```

8.rel_wai タスクの待ち状態を強制解除する

```
#include "hi8_3h.h"
/*
 *  8.REL_WAI :
 */
void tsk_XX08()
{
ER ercd;
ID tskid;
    /*...              */
    ercd = rel_wai(tskid);
    /*...              */
}
```

9.get_tid　自タスクのIDを得る

```
#include "hi8_3h.h"
/*
 *  9.GET_TID :
 */
void tsk_XX09()
{
ER ercd;
ID *p_tskid;
    /*...              */
    ercd = get_tid(p_tskid);
    /*...              */
}
```

10.tsk_sts　タスクの状態を見る

```
#include "hi8_3h.h"
/*
 *  10.TSK_STS :
 */
void tsk_XX10()
{
ER ercd;
UINT *p_tskstat;
TPRI *p_tskpri;
ID tskid;
    /*...              */
    ercd = tsk_sts(p_tskstat, p_tskpri,
                   tskid);
    /*...              */
}
```

11.sus_tsk　他タスクを強制待ち状態へ移行する

```
#include "hi8_3h.h"
/*
 *  11.SUS_TSK :
 */
void tsk_XX11()
{
ER ercd;
ID tskid;
    /*...              */
    ercd = sus_tsk(tskid);
    /*...              */
}
```

12.rsm_tsk　強制待ち状態のタスクを再開する

```
#include "hi8_3h.h"
/*
 *  12.RSM_TSK :
 */
void tsk_XX12()
{
ER ercd;
ID tskid;
    /*...              */
    ercd = rsm_tsk(tskid);
    /*...              */
}
```

13. slp_tsk　自タスクを待ち状態へ移行する

```
#include "hi8_3h.h"
/*
 *  13.SLP_TSK :
 */
void tsk_XX13()
{
ER ercd;
    /*...            */
    ercd = slp_tsk();
    /*...            */
}
```

14. wai_tsk　自タスクを一定時間待ち状態に移行する

```
#include "hi8_3h.h"
/*
 *  14.WAI_TSK :
 */
void tsk_XX14()
{
ER ercd;
TMO tmout;
    /*...            */
    ercd = wai_tsk(tmout);
    /*...            */
}
```

15. wup_tsk　待ち状態のタスクを起床する

```
#include "hi8_3h.h"
/*
 *  15.WUP_TSK :
 */
void tsk_XX15()
{
ER ercd;
ID tskid;
    /*...            */
    ercd = wup_tsk(tskid);
    /*...            */
}
```

16. iwup_tsk　待ち状態のタスクを起床する（非タスク部専用）

```
#include "hi8_3h.h"
/*
 *  16.IWUP_TSK :
 */
void int_XX16()
{
ER ercd;
ID tskid;
    /*...            */
    ercd = iwup_tsk(tskid);
    /*...            */
}
```

17. can_wup　タスクの起床要求を無効にする

```
#include "hi8_3h.h"
/*
 *  17.CAN_WUP :
 */
void tsk_XX17()
{
ER ercd;
INT *p_wupcnt;
ID tskid;
    /*...            */
    ercd = can_wup(p_wupcnt, tskid);
    /*...            */
}
```

18. set_flg　イベントフラグをセットする

```
#include "hi8_3h.h"
/*
 *  18.SET_FLG :
 */
void tsk_XX18()
{
ER ercd;
ID flgid;
UINT setptn;
    /*...            */
    ercd = set_flg(flgid, setptn);
    /*...            */
}
```

19.iset_flg　イベントフラグをセットする（非タスク部専用）

```
#include "hi8_3h.h"
/*
 *  19.ISET_FLG :
 */
void int_XX19()
{
ER ercd;
ID flgid;
UINT setptn;
    /*...            */
    ercd = iset_flg(flgid, setptn);
    /*...            */
}
```

20.clr_flg　イベントフラグをクリアする

```
#include "hi8_3h.h"
/*
 *  20.CLR_FLG :
 */
void tsk_XX20()
{
ER ercd;
ID flgid;
UINT clrptn;
    /*...            */
    ercd = clr_flg(flgid, clrptn);
    /*...            */
}
```

21.wai_flg　イベントフラグを待つ

```
#include "hi8_3h.h"
/*
 *  21.WAI_FLG :
 */
void tsk_XX21()
{
ER ercd;
UINT *p_flgptn;
ID flgid;
UINT waiptn;
UINT wfmode;
    /*...            */
    ercd = wai_flg(p_flgptn, flgid, waiptn,
                   wfmode);
    /*...            */
}
```

22.pol_flg　イベントフラグを得る

```
#include "hi8_3h.h"
/*
 *  22.POL_FLG :
 */
void tsk_XX22()
{
ER ercd;
UINT *p_flgptn;
ID flgid;
UINT waiptn;
UINT wfmode;
    /*...            */
    ercd = pol_flg(p_flgptn, flgid, waiptn,
                   wfmode);
    /*...            */
}
```

23.flg_sts　イベントフラグ状態を参照する

```
#include "hi8_3h.h"
/*
 *  23.FLG_STS :
 */
void tsk_XX23()
{
ER ercd;
ID *p_wtskid;
UINT *p_flgptn;
ID flgid;
    /*...            */
    ercd = flg_sts(p_wtskid, p_flgptn,
                   flgid);
    /*...            */
}
```

24.sig_sem　セマフォに対するV操作

```
#include "hi8_3h.h"
/*
 *  24.SIG_SEM :
 */
void tsk_XX24()
{
ER ercd;
ID semid;
    /*...            */
    ercd = sig_sem(semid);
    /*...            */
}
```

25.isig_sem　セマフォに対するＶ操作（非タスク部専用）

```
#include "hi8_3h.h"
/*
 *  25.ISIG_SEM :
 */
void int_XX25()
{
ER ercd;
ID semid;
    /*...            */
    ercd = isig_sem(semid);
    /*...            */
}
```

26.wai_sem　セマフォに対するＰ操作

```
#include "hi8_3h.h"
/*
 *  26.WAI_SEM :
 */
void tsk_XX26()
{
ER ercd;
ID semid;
    /*...            */
    ercd = wai_sem(semid);
    /*...            */
}
```

27.preq_sem　セマフォ資源を得る

```
#include "hi8_3h.h"
/*
 *  27.PREQ_SEM :
 */
void tsk_XX27()
{
ER ercd;
ID semid;
    /*...            */
    ercd = preq_sem(semid);
    /*...            */
}
```

28.sem_sts　セマフォ状態を参照する

```
#include "hi8_3h.h"
/*
 *  28.SEM_STS :
 */
void tsk_XX28()
{
ER ercd;
ID *p_wtskid;
UINT *p_semcnt;
ID semid;
    /*...            */
    ercd = sem_sts(p_wtskid, p_semcnt,
                   semid);
    /*...            */
}
```

29.snd_msg　メッセージを送信する

```
#include "hi8_3h.h"
/*
 *  29.SND_MSG :
 */
void tsk_XX29()
{
ER ercd;
ID mbxid;
T_MSG *pk_msg;
    /*...            */
    ercd = snd_msg(mbxid, pk_msg);
    /*...            */
}
```

30.isnd_msg　メッセージを送信する（非タスク部専用）

```
#include "hi8_3h.h"
/*
 * 30.ISND_MSG :
 */
void int_XX30()
{
ER ercd;
ID mbxid;
T_MSG *pk_msg;
    /*...            */
    ercd = isnd_msg(mbxid, pk_msg);
    /*...            */
}
```

31.rcv_msg　メッセージの受信を待つ

```
#include "hi8_3h.h"
/*
 *  31.RCV_MSG :
 */
void tsk_XX31()
{
ER ercd;
T_MSG **ppk_msg;
ID mbxid;
    /*...            */
    ercd = rcv_msg(ppk_msg, mbxid);
    /*...            */
}
```

32.prcv_msg　メッセージを受信する

```
#include "hi8_3h.h"
/*
 *  32.PRCV_MSG :
 */
void tsk_XX32()
{
ER ercd;
T_MSG **ppk_msg;
ID mbxid;
    /*...            */
    ercd = prcv_msg(ppk_msg, mbxid);
    /*...            */
}
```

33.mbx_sts　メイルボックス状態を参照する

```
#include "hi8_3h.h"
/*
 *  33.MBX_STS :
 */
void tsk_XX33()
{
ER ercd;
ID *p_wtskid;
T_MSG **ppk_msg;
ID mbxid;
    /*...            */
    ercd = mbx_sts(p_wtskid, ppk_msg,
                   mbxid);
    /*...            */
}
```

34.chg_ims　割込みマスクを変更する

```
#include "hi8_3h.h"
/*
 *  34.CHG_IMS :
 */
void tsk_XX34()
{
ER ercd;
CCR imask;
    /*...            */
    ercd = chg_ims(imask);
    /*...            */
}
```

35.ims_sts　割込みマスクを参照する

```
#include "hi8_3h.h"
/*
 * 35.IMS_STS :
 */
void tsk_XX35()
{
ER ercd;
CCR *p_imask;
    /*...            */
    ercd = ims_sts(p_imask);
    /*...            */
}
```

36.get_blk　固定長メモリブロックの獲得待ちを行なう

```
#include "hi8_3h.h"
/*
 *  36.GET_BLK :
 */
void tsk_XX36()
{
ER ercd;
VP *p_blk;
ID mplid;
    /*...            */
    ercd = get_blk(p_blk, mplid);
    /*...            */
}
```

37.pget_blk　固定長メモリブロックを獲得する

```
#include "hi8_3h.h"
/*
 *  37.PGET_BLK :
 */
void tsk_XX37()
{
ER ercd;
VP *p_blk;
ID mplid;
    /*...              */
    ercd = pget_blk(p_blk, mplid);
    /*...              */
}
```

38.rel_blk　固定長メモリブロックを返却する

```
#include "hi8_3h.h"
/*
 *  38.REL_BLK :
 */
void tsk_XX38()
{
ER ercd;
ID mplid;
VP blk;
    /*...              */
    ercd = rel_blk(mplid, blk);
    /*...              */
}
```

39.mpl_sts　メモリプール状態を参照する

```
#include "hi8_3h.h"
/*
 *  39.MPL_STS :
 */
void tsk_XX39()
{
ER ercd;
ID *p_wtskid;
UINT *p_frbcnt;
ID mplid;
    /*...              */
    ercd = mpl_sts(p_wtskid, p_frbcnt,
                   mplid);
    /*...              */
}
```

40.set_tim　システムクロックを設定する

```
#include "hi8_3h.h"
/*
 *  40.SET_TIM :
 */
void tsk_XX40()
{
ER ercd;
T_TIM *pk_time;
    /*...              */
    ercd = set_tim(pk_time);
    /*...              */
}
```

41.get_tim　システムコールの値を読み出す

```
#include "hi8_3h.h"
/*
 *  41.GET_TIM :
 */
void tsk_XX41()
{
ER ercd;
T_TIM *pk_time;
    /*...              */
    ercd = get_tim(pk_time);
    /*...              */
}
```

42.get_ver　カーネルのバージョン識別子を得る

```
#include "hi8_3h.h"
/*
 *  42.GET_VER :
 */
void tsk_XX42()
{
ER ercd;
T_VER *pk_ver;
    /*...              */
    ercd = get_ver(pk_ver);
    /*...              */
}
```

付録Ｃ．エラーコード一覧

Ｃ．１　システムコールエラーコード一覧

表Ｃ－１　システムコールエラーコード一覧

No.	ercd(r0)	値		エラー内容	備考
		符号付	符号無		
1	E_OK	H'000	H'0000	正常終了	
2	E_PAR	-H'721	H'F8DF	パラメータエラー	ﾊﾟﾗﾒｰﾀﾁｪｯｸが必要
3	E_ILADR	-H'722	H'F8DE	不正アドレス	ﾊﾟﾗﾒｰﾀﾁｪｯｸが必要
4	E_TPRI	-H'726	H'F8DA	不正タスク優先度	ﾊﾟﾗﾒｰﾀﾁｪｯｸが必要
5	E_ILTIME	-H'727	H'F8D9	不正時間指定	ﾊﾟﾗﾒｰﾀﾁｪｯｸが必要
6	E_ILMSG	-H'729	H'F8D7	不正メッセージ形式	
7	E_IMS	-H'72B	H'F8D5	不正imask	ﾊﾟﾗﾒｰﾀﾁｪｯｸが必要
8	E_SELF	-H'831	H'F7CF	自タスク指定	ﾊﾟﾗﾒｰﾀﾁｪｯｸが必要
9	E_NOEXS	-H'834	H'F7CC	オブジェクトが存在していない	
10	E_DMT	-H'835	H'F7CB	タスクが休止状態である	
11	E_NODMT	-H'836	H'F7CA	タスクが休止状態でない	
12	E_NOSUS	-H'838	H'F7C8	タスクが強制待ち状態でない	
13	E_ILBLK	-H'83B	H'F7C5	不正メモリブロックの返却、操作	ﾊﾟﾗﾒｰﾀﾁｪｯｸが必要
14	E_NOWAI	-H'83E	H'F7C2	タスクが待ち状態でない	
15	E_CTX	-H'A45	H'F5BB	コンテキストエラー	ﾊﾟﾗﾒｰﾀﾁｪｯｸが必要
16	E_QOVR	-H'B49	H'F4B7	キューイングのオーバーフロー	
17	E_TMOUT	-H'D55	H'F2AB	タイムアウト	
18	E_RLWAI	-H'D56	H'F2AA	待ち状態強制解除	
19	E_PLFAIL	-H'E59	H'F1A7	ポーリング失敗	

Ｃ．２　システム異常終了エラーコード一覧

システム異常終了時、スタックにエラー情報(割込みベクタ番号、タスクID、エラーコート)が積まれます。エラー情報の積まれ方については、「２．１１　H18-3Hのシステムの異常終了処理」を参照してください。

表Ｃ－２　システム異常終了エラーコード一覧

No.	割込みﾍﾞｸﾀ番号(vecno)	ﾀｽｸID (tskid)	ｴﾗｰｺｰﾄﾞ (ercd)	エラー内容
1	H'00	H'00	H'0000～H'0FFF	セットアップテーブルのエラー (ercdにセットアップ情報エラーコードを設定)
2	H'00	H'00	H'F9ED	タイマ未サポート
3	H'00	H'00	H'FFEB	非タスク部からext_tsk システムコール発行
4	H'00	tskid	H'FFBB	タスク部からret_int システムコール発行
5	ﾍﾞｸﾀ番号	tskid*	発生時のCCR,PC	未定義割込みの発生

【注】＊　タスク部で発生した場合はtskidが設定され、非タスク部で発生した場合は０が設定されます。

C．３　セットアップ情報エラーコード一覧

システム起動時、セットアップテーブルの情報にエラーがある場合、スタックにセットアップ情報エラーコードが積まれます。セットアップ情報エラーコードの積まれ方については、「２．１１　HI8-3Hのシステムの異常終了処理」を参照してください。

表Ｃ－３　セットアップ情報エラーコード一覧

No.	ｴﾗｰｺｰﾄﾞ (ercd)	不正項目
1	H'0101	OS用スタックポインタ(_HI_OS_SP)が０または奇数
2	H'0102	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
3	H'0103	システム作業領域の先頭アドレス(セクション名「hi8_3h_ram」)が０または奇数
4	H'0104	TIMCB 領域(_HI_TIMCB)が０または奇数
5	H'0105	TCB 領域(_HI_TCB)が０または奇数
6	H'0106	FLGCB 領域(_HI_FLGCB)が０または奇数
7	H'0107	SEMCB 領域(_HI_SEMCB)が０または奇数
8	H'0108	MBXCB 領域(_HI_MBXCB)が０または奇数
9	H'0109	MPLCB 領域(_HI_MPLCB)が０または奇数
10	H'010A	トレーススタックポインタ(_HI_TRC_SP)が０または奇数
11	H'010B	トレース管理領域(TBACB)が０または奇数
12	H'0201	システム初期化ハンドラの先頭アドレス(_HIPRG_SYSINI)が奇数
13	H'0202	タイマ初期設定ルーチンの先頭アドレス(_HIPRG_TIMINI)が奇数
14	H'0301	カーネル割込みマスクレベル(IMASK)が４以上
15	H'0302	優先度定義数(MAXPRI)が32以上
16	H'0303	タスク定義数(TSKCNT)が256 以上
17	H'0304	イベントフラグ定義数(FLGCNT)が256 以上
18	H'0305	セマフォ定義数(SEMCNT)が256 以上
19	H'0306	メイルボックス定義数(MBXCNT)が256 以上
20	H'0307	メモリプール定義数(MPLCNT)が256 以上
21	H'0401	タスク定義テーブル(_HI_TDT)が０または奇数
22	H'0402	メモリプール定義テーブル(_HI_MPLDT)が０または奇数
23	H'0403	未定義割込みハンドラ(_HI_ILT)が０または奇数
24	H'0404	トレースバッファ情報テーブル(INITRC)が０または奇数
25	H'0501	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
26	H'0502	タスク先頭アドレス(TSKADR)が０または奇数
27	H'0503	タスクスタックポインタ(ITSKSP)が０または奇数
28	H'0504	メモリブロック長(BLKLEN)が０または奇数または65532バイト以上
29	H'0505	メモリプールアドレス(MPLADR)が０または奇数
30	H'0506	トレースバッファアドレス(TRACE BUFFER ADDRESS)が０または奇数

付録D．ヘッダファイル

D．1　アセンブリ言語用ヘッダファイル

アセンブリ言語用ヘッダファイル『hi8_3h.equ』は、アセンブリ言語で用いる変数を宣言しているファイルです。ユーザプログラムをアセンブリ言語で作成する場合には、ユーザプログラムの先頭で.include 制御文により本ヘッダファイルを指定してください。

アセンブリ言語用ヘッダファイルの内容を以下に示します。

```
;3333333333333333333333333333333333333333333333333333333333333333333333333333333333333333
;HHH    H18-3H Assembler Language include Ver2.00                                    HHH
;HHH    Copyright (c)  Hitachi, Ltd. 1993.                                           HHH
;HHH    Licensed Material of Hitachi, Ltd.                                           HHH
;3333333333333333333333333333333333333333333333333333333333333333333333333333333333333333
;
        .radix  d                       ;:xxxxx -> d'xxxxx
;
;###### general equate ################;: for "mbx_sts" only
;
NADR:           .equ    (-1)            ;:
;
;###### tskid equate ##################;: tskid of self
;
TSK_SELF:       .equ    0               ;:
;
;###### tskpri equate #################;: for "chg_pri/(i)rot_rdq"
;
TPRI_INI:       .equ    0               ;: task initial priority  :chg_pri
TPRI_RUN:       .equ    0               ;: executing task priority:(i)rot_rdq
;
;###### wtskid equate #################;: for "xxx_sts"
;
NOTSK:          .equ    0               ;:
;
;###### task status equate #############;: for tsk_sts
;
TTS_RUN:        .equ    b'0000000000000001;: status of RUN
TTS_RDY:        .equ    b'0000000000000010;: status of READY
TTS_WAI:        .equ    b'0000000000000100;: status of WAIT
TTS_SUS:        .equ    b'0000000000001000;: status of SUSPEND
TTS_WAS:        .equ    b'0000000000001100;: status of WAIT-SUSPEND
TTS_DMT:        .equ    b'0000000000010000;: status of DORMANT
;
;###### wfmode equate #################;: for "wai_flg/pol_flg"
;
TWF_ANDW:       .equ    0               ;: AND wait mode
TWF_ORW:        .equ    2               ;: OR  wait mode
TWF_CLR:        .equ    1               ;: CLEAR mode
;
```

```
;###### ercd equate ####################;: for all systemcall
;
E_OK:           .equ    h'0000          ;: h'000
E_TNOSPT:       .equ    h'F9ED          ;: -h'613
E_PAR:          .equ    h'F8DF          ;: -h'721
E_ILADR:        .equ    h'F8DE          ;: -h'722
E_TPRI:         .equ    h'F8DA          ;: -h'726
E_ILTIME:       .equ    h'F8D9          ;: -h'727
E_ILMSG:        .equ    h'F8D7          ;: -h'729
E_IMS:          .equ    h'F8D5          ;: -h'72B
E_SELF:         .equ    h'F7CF          ;: -h'831
E_NOEXS:        .equ    h'F7CC          ;: -h'834
E_DMT:          .equ    h'F7CB          ;: -h'835
E_NODMT:        .equ    h'F7CA          ;: -h'836
E_NOSUS:        .equ    h'F7C8          ;: -h'838
E_ILBLK:        .equ    h'F7C5          ;: -h'83B
E_NOWAI:        .equ    h'F7C2          ;: -h'83E
E_CTX:          .equ    h'F5BB          ;: -h'A45
E_QOVR:         .equ    h'F4B7          ;: -h'B49
E_TMOUT:        .equ    h'F2AB          ;: -h'D55
E_RLWAI:        .equ    h'F2AA          ;: -h'D56
E_PLFAIL:       .equ    h'F1A7          ;: -h'E59
;
;****** end of HI8_3H.EQU **************;
```

D．2　C言語用ヘッダファイル

C言語用ヘッダファイル『hi8_3h.h』は、C言語で用いる変数を宣言しているファイルです。ユーザプログラムをC言語で作成する場合には、ユーザプログラムの先頭で#include 制御文により本ヘッダファイルを指定してください。

C言語用ヘッダファイルの内容を以下に示します。

```
/*3333333333333333333333333333333333333333333333333333333333333333333333333333*/
/*HH    H18-3H C Language Header V2.00                                      HH*/
/*HH    Copyright (c)  Hitachi, Ltd. 1993.                                  HH*/
/*HH    Licensed Material of Hitachi, Ltd.                                  HH*/
/*3333333333333333333333333333333333333333333333333333333333333333333333333333**/

/*----------------------------------------------------------------------*/
/*      type define                                                     */
/*----------------------------------------------------------------------*/
typedef char            B;      /*   signed  8 bits integer            */
typedef short           H;      /*   signed 16 bits integer            */
typedef long            W;      /*   signed 32 bits integer            */
typedef unsigned char   UB;     /* unsigned  8 bits integer            */
typedef unsigned short  UH;     /* unsigned 16 bits integer            */
typedef unsigned long   UW;     /* unsigned 32 bits integer            */

typedef long            VW;     /* void data type (32 bits size)       */
typedef short           VH;     /* void data type (16 bits size)       */
typedef char            VB;     /* void data type ( 8 bits size)       */
typedef void            *VP;    /* pointer of to void data type        */

typedef UH              ID;     /* type of object ID (xxxid)           */
typedef H               ER;     /* type of error code                  */
typedef W               TMO;    /* type of time out's time             */
typedef H               TPRI;   /* type of task priority               */
typedef UH              CCR;    /* type of interrupt mask              */
typedef H               INT;    /*   signed integer (H8 : 16 bits)     */
typedef UH              UINT;   /* unsigned integer (H8 : 16 bits)     */
/*----------------------------------------------------------------------*/
/*      structure type define                                           */
/*----------------------------------------------------------------------*/
typedef struct t_msg {          /* ----------[ xxx_msg ]----------- */
        UW      msghead;        /* message header (by OS management)*/
        VB      msgcont[1];     /* message contents                    */
} T_MSG;

typedef struct t_tim {          /* ----------[ xxx_tim ]----------- */
        H       utime;          /* current date/time (upper)           */
        UH      mtime;          /* current date/time (middle)          */
        UH      ltime;          /* current date/time (lower)           */
} T_TIM;

typedef struct t_ver {          /* ----------[ get_ver ]----------- */
        UH      maker;          /* maker                               */
        UH      id;                     /* id No.                              */
        UH      spver;          /* specification version No.           */
        UH      prver;          /* process version No.                 */
        UH      prno[4];        /* process management infomation       */
        UH      cpu;                    /* CPU infomation                      */
        UH      var;                    /* variation                           */
} T_VER;

typedef struct t_trccb {        /* ----------[ dact_trc ]----------- */
        VP      tr_trbtop;      /* trace entry top address */
        VP      tr_trbbtm;      /* trace entry btm address */
        VP      tr_trbins;      /* trace entry insert address */
        UW      trbsts;         /* trace buffer status */
} T_TRCCB;
```

```
typedef struct t_trcent {
        UH      te_attr;        /* event attribute */
        ID      te_tskid;       /* task id */
        UW      te_ltime;       /* system clock (low 4byte) */
        H       te_event;       /* event */
        VH      te_r1;          /* parameter r1 */
        VW      te_er2;         /* parameter er2 */
        VH      te_r3;          /* parameter r3 */
        VH      te_r4;          /* parameter r4 */
        UH      te_ccr_pc;      /* parameter ccr */
        UH      te_pc;          /* parameter pc */
}T_TRCENT;

/*-------------------------------------------------------------------*/
/*      general constant                                             */
/*-------------------------------------------------------------------*/
#define NADR        (VP)(-1)    /* null address/null pointer        */
#define TRUE            1       /* true                             */
#define FALSE           0       /* false                            */
/*-------------------------------------------------------------------*/
/*      tskid constant                                               */
/*-------------------------------------------------------------------*/
#define TSK_SELF        0       /* tskid of self                    */
/*-------------------------------------------------------------------*/
/*      tskpri constant                                              */
/*-------------------------------------------------------------------*/
#define TPRI_INI        0       /* task initial priority   (chg_pri)*/
#define TPRI_RUN        0       /* executing task priority (rot_rdq)*/
/*-------------------------------------------------------------------*/
/*      wtskid constant                                              */
/*-------------------------------------------------------------------*/
#define NOTSK           0       /* status of no waiting task        */
/*-------------------------------------------------------------------*/
/*      task status constant                                         */
/*-------------------------------------------------------------------*/
#define TTS_RUN         0x0001  /* status of RUN                    */
#define TTS_RDY         0x0002  /* status of READY                  */
#define TTS_WAI         0x0004  /* status of WAIT                   */
#define TTS_SUS         0x0008  /* status of SUSPEND                */
#define TTS_WAS         0x000c  /* status of WAIT-SUSPEND           */
#define TTS_DMT         0x0010  /* status of DORMANT                */
/*-------------------------------------------------------------------*/
/*      wfmode constant                                              */
/*-------------------------------------------------------------------*/
#define TWF_ANDW        0x0000  /* AND-wait condition               */
#define TWF_ORW         0x0002  /* OR-wait condition                */
#define TWF_CLR         0x0001  /* clear condition                  */
/*-------------------------------------------------------------------*/
/*      ercd constant                                                */
/*-------------------------------------------------------------------*/
#define E_OK            0x0000  /* normal end                       */
#define E_TNOSPT        0xf9ed  /* no support timer                 */
#define E_PAR           0xf8df  /* parameter error                  */
#define E_ILADR         0xf8de  /* illegal address                  */
#define E_TPRI          0xf8da  /* illegal task priority            */
#define E_ILTIME        0xf8d9  /* illegal time                     */
#define E_ILMSG         0xf8d7  /* illegal message (msghead<>0)     */
#define E_IMS           0xf8d5  /* illegal IMASK                    */
#define E_SELF          0xf7cf  /* self task ID error               */
#define E_NOEXS         0xf7cc  /* object not exists                */
#define E_DMT           0xf7cb  /* task status is DORMANT           */
#define E_NODMT         0xf7ca  /* task status is not DORMANT       */
#define E_NOSUS         0xf7c8  /* task status is not SUSPEND       */
#define E_ILBLK         0xf7c5  /* illegal block                    */
#define E_NOWAI         0xf7c2  /* task status is not WAIT          */
#define E_CTX           0xf5bb  /* context error                    */
#define E_QOVR          0xf4b7  /* queueing overflow                */
#define E_TMOUT         0xf2ab  /* time out error                   */
#define E_RLWAI         0xf2aa  /* release wait error               */
#define E_PLFAIL        0xf1a7  /* polling error                    */
```

```
/*-----------------------------------------------------------------------*/
/*      prototype declaration                                            */
/*-----------------------------------------------------------------------*/
ER    sta_tsk(ID tskid);
ER   ista_tsk(ID tskid);
void ext_tsk(void);
ER    ter_tsk(ID tskid);
ER    chg_pri(ID tskid, TPRI tskpri);
ER    rot_rdq(TPRI tskpri);
ER   irot_rdq(TPRI tskpri);
ER    rel_wai(ID tskid);
ER    get_tid(ID *p_tskid);
ER    tsk_sts(UINT *p_tskstat, TPRI *p_tskpri, ID tskid);
ER    sus_tsk(ID tskid);
ER    rsm_tsk(ID tskid);
ER    slp_tsk(void);
ER    wai_tsk(TMO tmout);
ER    wup_tsk(ID tskid);
ER   iwup_tsk(ID tskid);
ER    can_wup(INT *p_wupcnt, ID tskid);
ER    set_flg(ID flgid, UINT setptn);
ER   iset_flg(ID flgid, UINT setptn);
ER    clr_flg(ID flgid, UINT clrptn);
ER    wai_flg(UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode);
ER    pol_flg(UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode);
ER    flg_sts(ID *p_wtskid, UINT *p_flgptn, ID flgid);
ER    sig_sem(ID semid);
ER   isig_sem(ID semid);
ER    wai_sem(ID semid);
ER   preq_sem(ID semid);
ER    sem_sts(ID *p_wtskid, UINT *p_semcnt, ID semid);
ER    snd_msg(ID mbxid, T_MSG *pk_msg);
ER   isnd_msg(ID mbxid, T_MSG *pk_msg);
ER    rcv_msg(T_MSG **ppk_msg, ID mbxid);
ER   prcv_msg(T_MSG **ppk_msg, ID mbxid);
ER    mbx_sts(ID *p_wtskid, T_MSG **ppk_msg, ID mbxid);
ER    chg_ims(CCR imask);
ER    ims_sts(CCR *p_imask);
ER    get_blk(VP *p_blk, ID mplid);
ER   pget_blk(VP *p_blk, ID mplid);
ER    rel_blk(ID mplid, VP blk);
ER    mpl_sts(ID *p_wtskid, UINT *p_frbcnt, ID mplid);
ER    set_tim(T_TIM *pk_tim);
ER    get_tim(T_TIM *pk_tim);
ER    get_ver(T_VER *pk_ver);

/*** end of hi8_3h.h ***************************************************/
```

付録Ｅ．Ｖ１．０との相違点

Ｅ．１　機能追加項目

(1)H8/300Hノーマルモードへの対応

HI8-3H V2.0では、H8/300Hノーマルモード用とH8/300Hアドバンストモード用の２種類のカーネルを選択して使用することができます。

H8/300Hのノーマルモード用、アドバンストモード用のカーネルの相違については、「付録Ｆ．HI8-3Hﾉｰﾏﾙ/ｱﾄﾞﾊﾞﾝｽﾄ相違表」を参照してください。

(2)共有スタック機能のサポート

HI8-3H V2.0では、複数のタスクで１つのスタック領域を共有し、システムの使用する作業領域を削減することができる共有スタック機能を使用できます。

共有スタック機能の詳細については、「２．２．５　共有スタック機能」を参照してください。

共有スタック機能を使用するための構築方法については、UNIX HI8-3H構築マニュアルまたはMS-DOS HI8-3H構築マニュアルの「２．４　セットアップテーブルの作成」を参照してください。

(3)トレース機能のサポート

システムコールの実行履歴を保存するトレース機能を使用できます。トレース機能を使用することにより、デバッグが容易に行なえます。

H8/300Hマルチタスクデバッガのコマンドを使用することにより、システムコールの実行履歴を表示することができます。(HI8-3H V2.0では、E7000用H8/300Hマルチタスクデバッガ V2.0を使用してください。)

トレース機能の詳細については、「２．１２　トレース機能」を参照してください。

トレース機能を使用するための構築方法については、UNIX HI8-3H構築マニュアルまたはMS-DOS HI8-3H構築マニュアルの「２．４　セットアップテーブルの作成」を参照してください。

Ｅ．２　機能変更項目

(1)ヘッダファイル（ファイル名『hi8_3h.h』）

HI8-3H V2.0では、メッセージのtypedefのメンバmsgheadの型宣言がVP(データタイプが一定でないものへのポインタ)からUW(符号なし32ビット整数)に変更されています。

(2)割込みハンドラ

(a)NMI 割込みハンドラ

V1.0ではNMI 割込みハンドラからchg_ims,ims_stsシステムコールを発行できましたが、V2.0では発行できませんので直接CCRレジスタを変更／参照してください。

(b)カーネル割込みマスクレベルより高い割込みレベルの割込みハンドラ

V1.0では割込みハンドラからims_stsシステムコールを発行できましたが、V2.0では発行できませんので直接CCRレジスタを参照してください。

(3)E7000用HI8-3Hマルチタスクデバッガ

HI8-3H V1.0では、E7000用HI8-3Hマルチタスクデバッガ V1.0を使用できましたが、HI8-3H V2.0では使用できません。

HI8-3H V2.0には、E7000用HI8-3Hマルチタスクデバッガ V2.0を使用してください。

付録Ｆ．HI8-3Hﾉｰﾏﾙ/ｱﾄﾞﾊﾞﾝｽﾄ相違表

Ｆ．１　相違内容

表Ｆ－１　ノーマル／アドバンストモードの相違表

項番	相違内容	ノーマルモード	アドバンストモード
1	システムコール（アセンブラインタフェース） ｼｽﾃﾑｺｰﾙ　ﾊﾟﾗﾒｰﾀ snd_msg　pk_msg isnd_msg　pk_msg rcv_msg　pk_msg prcv_msg　pk_msg mbx_sts　pk_msg get_blk　blk pget_blk　blk rel_blk　blk set_tim　pk_time get_tim　pk_time get_ver　pk_ver	Ｒ２	ＥＲ２
2	割込みベクタテーブル ベクタアドレスのサイズ	２バイト	４バイト
3	メモリ容量の算出	メモリ容量の詳細は、UNIX HI8-3H構築マニュアルまたはMS-DOS HI8-3H構築マニュアル「付録Ａ．メモリ容量の算出」を参照してください	
4	コンソールドライバ，タイマドライバ， ＣＰＵ初期化ルーチン 内蔵レジスタアドレスのサイズ（equate定義）	２バイト	４バイト

付録G．ＡＳＣＩＩコード表

G．１　ＡＳＣＩＩコード表

上位４ビット / 下位４ビット	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

付録H. 索引

H. 1 五十音順・索引

五十音順・索引

タ　行

ナ　行

ハ　行

マ　行

ヤ　行

ラ　行

ワ　行

アルファベット順・索引

F

G

H

I

M

N

O

P

R

S

T

V

W

HI8-3H
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-702-148A