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

UNIX HI8-3X

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 H8/300

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　1994.03

本製品は、東京大学理学部情報科学科　坂村　健博士のご指導のもとに開発されたもので、TRON 仕様にもとづいています。

はじめに

本マニュアルは、機器組込み用リアルタイム・マルチタスクＯＳ(Operating System)であるμITRON*1仕様に準拠した、ＨＩ８－３Ｘ(Hitachi Industrial Realtime Operating System H８／３００）の使用方法について説明します。

＜マニュアルの構成＞

本マニュアルは、次に示す５つの章と付録から構成されています。

第１章では、ＨＩ８－３Ｘの概説が記述してあります。本システムの全体を理解していただけると思います。

第２章では、ニュークリアス（チップ核）の持つ機能の概要を記述しています。

第３章では、ニュークリアスのシステムコールの仕様を記述しています。

第４章では、セットアップテーブルを用いたＨＩ８－３Ｘのシステム構築方法を記述しています。

第５章では、Ｃ言語を用いたアプリケーションプログラムの記述方法について記述しています。

付録では、コンソールドライバの例題、各種一覧表、メモリ容量の算出表、ＵＮＩＸ*2ワークステーション（SPARC*3 Sun OS*4)上でのシステム構築の例を記述しています。

＜関連マニュアル＞

関連マニュアルは次のとおりです。

・Ｈ８／３００シリーズクロスアセンブラ　ユーザーズマニュアル
・Ｈ８／３００シリーズＣコンパイラ　ユーザーズマニュアル
・Ｈシリーズ　ライブラリアン　ユーザーズマニュアル
・Ｈシリーズ　リンケージエディタ　ユーザーズマニュアル
・Ｈ８／３２５シリーズ　ハードウェアマニュアル

＜本マニュアルで使用する記号等の意味＞

[　]	省略できることを意味します。
{　}	いずれかひとつを選択することを意味します。
(RET)	RETURNキーの入力を意味します。
△	１つ以上の空白またはタブコードを意味します。
<u>　　</u>	アンダーラインの部分はユーザがキー入力するコマンドラインです。
<　>	この記号で囲まれた内容を指定することを意味します。
..	直前の項目を１回以上指定することを示します。
H'	整数定数の16進数には、先頭に“H'”を付けています。デフォルトは、10進数です。

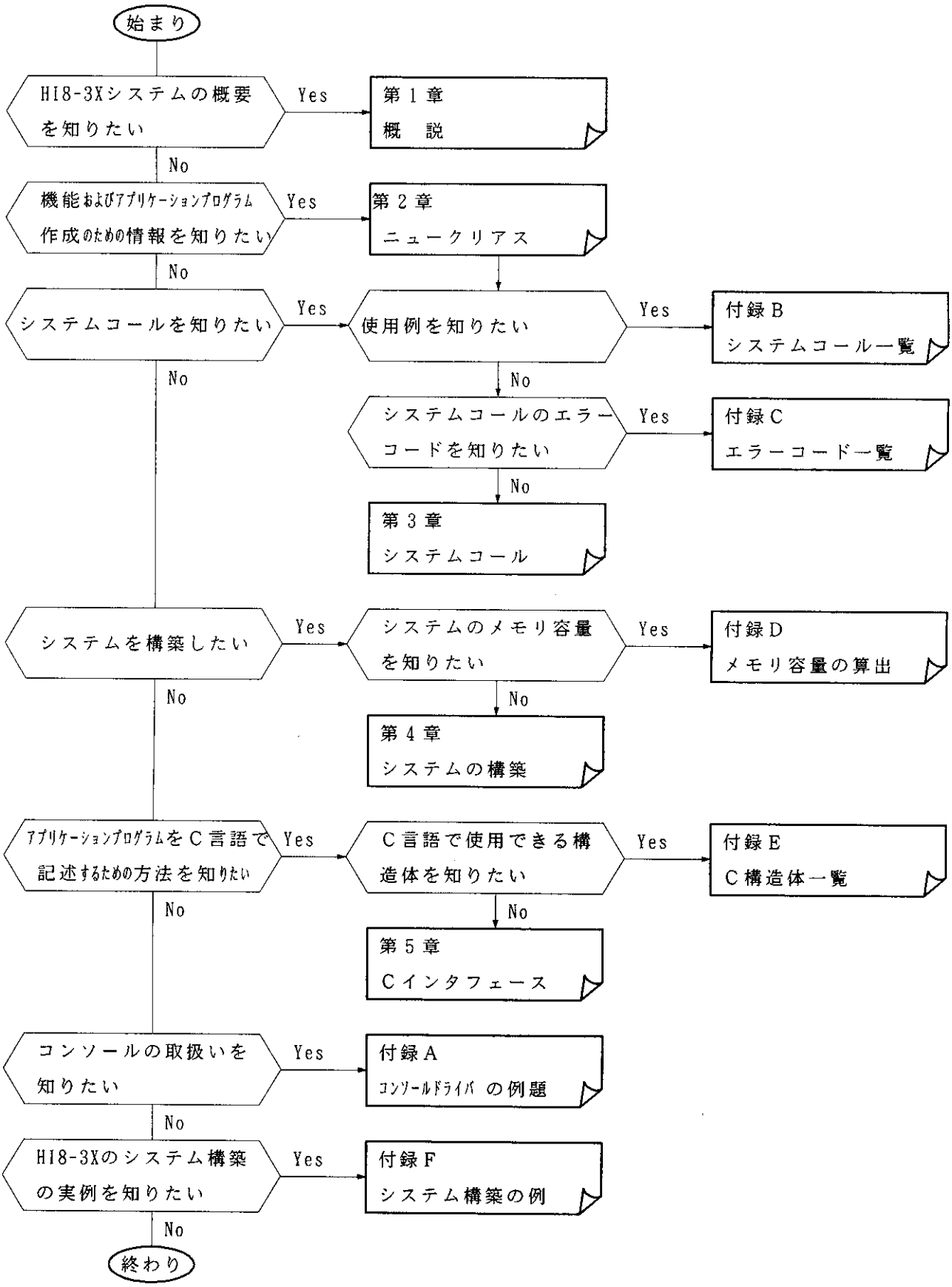
*1　μITRON は、"Micro Industrial TRON" の略称です。
TRONは、"The Real time Operating system Nucleus"の略称です。

*2　UNIXは米国UNIX System Laboratories, Inc.により管理されている、オペレーティングシステムの名称です。

*3　SPARC は、米国SPARC International, Inc. により管理されている。ＣＰＵおよびワークステーションの名称です。

*4　Sun OSは、米国Sun Micro Systems, Inc. により管理されているソフトウェアの名称です。

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

＜第３章　システムコール＞

＜第４章 システムの構築＞

《付録》

<付録Ａ．コンソールドライバの例題>

<付録Ｂ．システムコール一覧>

<付録Ｃ．エラーコード一覧>

<付録Ｄ．メモリ容量の算出>

<付録Ｅ．Ｃ構造体一覧>

<付録Ｆ．システム構築の例>

図　目　次

＜第４章　システムの構築＞

＜第５章　Ｃインタフェース＞

＜付録Ａ．コンソールドライバの例題＞

＜付録Ｆ．システム構築の例＞

表　目　次

リスト目次

＜第３章　システムコール＞

＜第４章　システムの構築＞

＜付録Ｆ　システム構築の例＞

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。

Ｈ８－３Ｘは、日立Ｈシリーズマイコンの一つである、Ｈ８／３００シリーズ用の機器組込み用リアルタイム・マルチタスクＯＳです。

Ｈ８－３Ｘの仕様は、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

Ｈ８－３Ｘには、次のような特長があります。

(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

１０ＭＨｚ動作時、タスク起床時間２７μｓ（wup_tsk システムコール使用時の例）、割込み禁止時間１３μｓ以下と、非常に高速なリアルタイムＯＳです。

(3) コンパクトなＯＳ

Ｈ８／３００シリーズの内蔵ＲＯＭ・内蔵ＲＡＭでシステムが構築できるように、ＯＳのプログラムサイズおよび作業領域の最小化を図ってあります。

また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、常に必要最低限の機能モジュールでシステムを構築できます。

(4) スタック領域の共有

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。

(5) 保障レジスタ数の変更

タスクで保障するレジスタの数を選択することができます。

これにより、スタック領域の縮小や、システム全体を通してのレジスタの共用が可能になります。

(6) 容易なシステム構築

アプリケーションプログラムとＯＳの機能モジュールライブラリをリンケージエディタで結合するだけで、自動的にＯＳの機能選択を行なうことができます。

(7) 豊富なリアルタイム・マルチタスク処理機能

Ｈ８－３Ｘは、つぎの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。

- 優先度に基づくタスクスケジューリング
- イベントフラグ・セマフォ・メイルボックスによるタスク間の同期・通信機能
- 割込み管理機能

・時間・タスクの実行遅延等の時間管理機能

(8) コンソールドライバの提供

各種のＩ／Ｏドライバを容易に作成できるよう、コンソール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。

(9) 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。

１．３　構　成

ＨＩ８－３Ｘで提供される主要なプログラムやテーブルには、次のものがあります。

(1) ニュークリアス

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。

(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。

(3) Ｃ言語インタフェースライブラリ

ＨＩ８－３Ｘのシステムコールを、Ｃ言語で記述したアプリケーション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

Ｈ８／３００シリーズの内蔵シリアルI/O を使用したコンソールを制御するプログラムです。コンソールを使用しなければ、ユーザシステムに組込む必要はありません。

表１－１　にＨＩ８－３Ｘの各プログラムの容量を示します。

表１－１　プログラム容量

項番	プログラム名		容　量
1	ニュークリアス	最小構成	約０．４ｋバイト
		最大構成	約４．０ｋバイト
2	セットアップテーブル		１２バイト以上 *
3	Ｃ言語インタフェースライブラリ		４バイト以上
4	コンソールドライバ		約０．４ｋバイト

【注】　* 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。

１．４　システムの構築手順

システムの構築手順を以下に示します。

(1) アプリケーションプログラムの作成

ＵＮＩＸワークステーション上で、アセンブリ言語またはＣ言語でアプリケーションプログラムを記述し、アセンブル／コンパイルを行なってオブジェクトファイルを作成してください。

(2) セットアップテーブルの作成

Ｈ８－３Ｘを動作させるために必要なシステムの環境を定義するセットアップテーブルを、アセンブリ言語で記述し、アセンブルを行なってオブジェクトファイルを作成してください。

(3) アブソリュートプログラムの作成

アプリケーションプログラム、セットアップテーブルの各オブジェクトファイル、Ｈ８－３Ｘのライブラリファイルをリンケージエディタで結合し、アブソリュートプログラムを作成してください。

Ｈ８－３Ｘはリンケージエディタで結合するだけで、自動的にＯＳの機能モジュールの選択が行なわれます。

(4) プログラムのロード

作成したアブソリュートプログラムをユーザ実機（ターゲットシステム）にロードするためには、次の方法があります。

- Ｈ８／３００シリーズ用 Adaptive System Evaluator（以下ＡＳＥと略す）を使用する方法
  ＡＳＥを使用して、ＵＮＩＸワークステーションからユーザ実機（ターゲットシステム）にプログラムをロードします（ただし、ＵＮＩＸワークステーションは、ＳＰＡＲＣのみ）。
- ＲＯＭを使用する方法
  Ｈ８／３００シリーズの内蔵ＲＯＭまたは外部ＲＯＭにプログラムを書き込み、ユーザ実機に搭載します。

図１－１　にシステムの構築手順を示します。

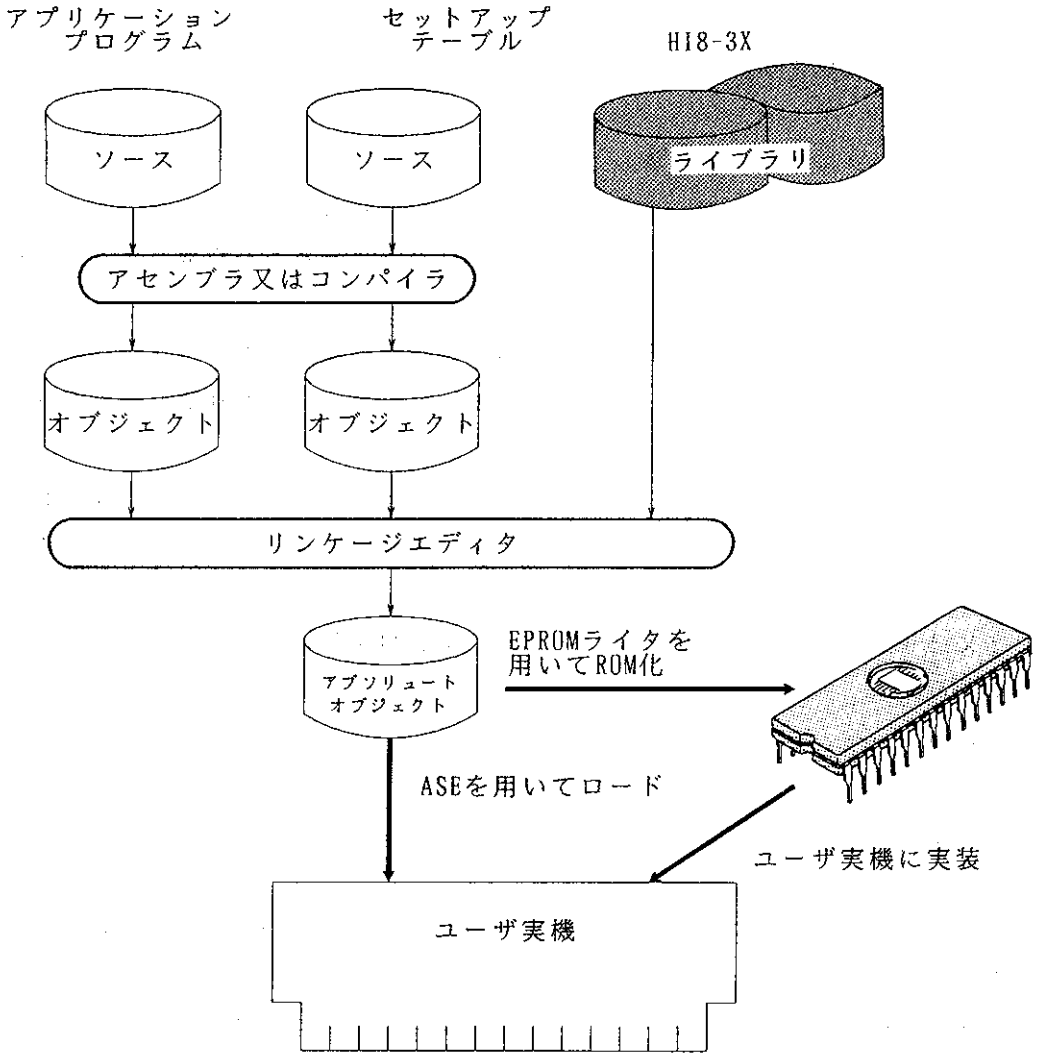


図１－１　システムの構築手順

2．ニュークリアス

2．1　概　要

オペレーティングシステムのリアルタイム・マルチタスク処理を行なう核となる部分を、ニュークリアスと呼びます。

ニュークリアスには大きく分けて次の３つの役割があります。

・　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。

・　タスクのスケジューリング

仕事（タスク）をその優先度に応じて、実行のスケジューリングを行ないます。

・　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。

ＨＩ８－３Ｘのニュークリアスは、μＩＴＲＯＮ仕様のレベル２のシステムコールを包含し、６つのモジュール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、優先順位の高いものから順にＣＰＵに割り付けられます。

(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を行ないます。

(3) 同期・通信管理

イベントフラグ、セマフォ、メイルボックスの三つの機能があり、タスク間の同期・通信処理を行ないます。

(4) 時間管理

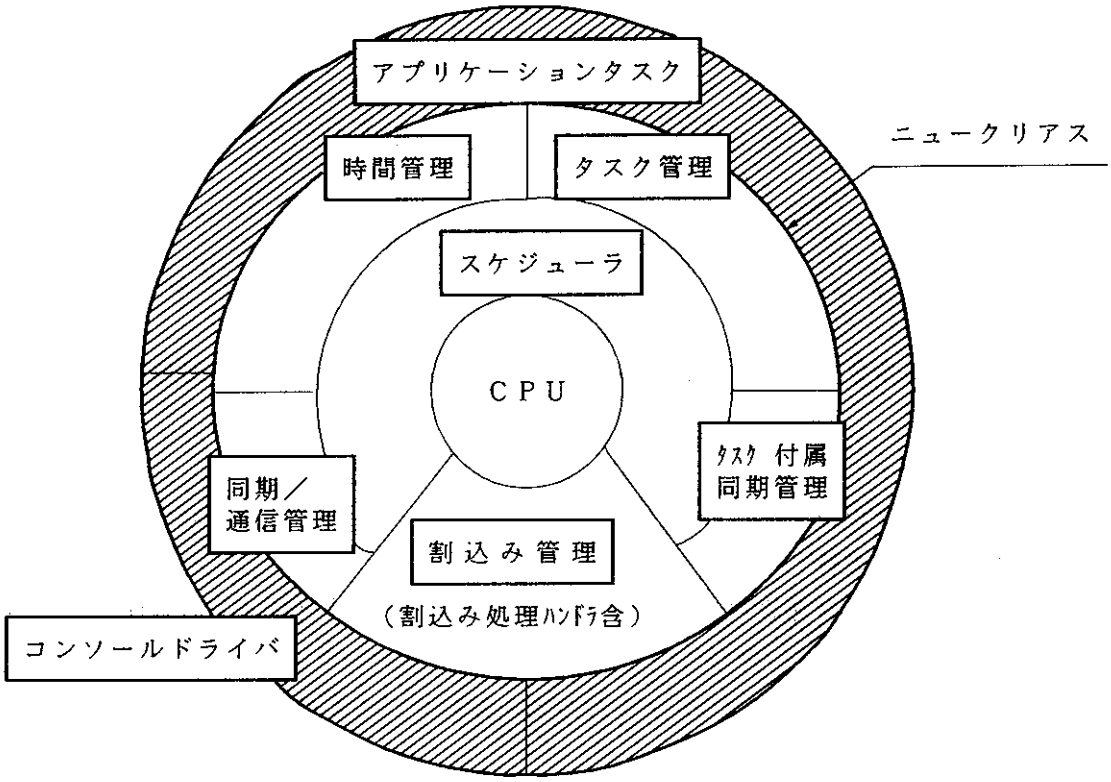
時間の管理を行ないます。また、タスクの実行制御のための時間監視を行ないます。

(5) 割込み管理

割込み発生時に割込み処理ハンドラ* を起動し、割込み処理やタスクへの割込み発生の連絡を行ないます。

【注】　* 割込み処理ハンドラは、必要な場合作成しなければなりません。

図２－１　にＨＩ８－３Ｘのシステム構成を示します。



【注】アプリケーションタスクと割込み管理の割込み処理ハンドラは、ユーザが作成します。

図２－１　Ｈ８－３Ｘのシステム構成

２．２　タスク管理、タスク付属同期管理

２．２．１　タスク管理、タスク付属同期管理の概要

ＨＩ８－３Ｘにおいて、アプリケーションプログラムはタスクと呼ばれる単位でニュークリアスによって制御されます。ニュークリアスは、READY(実行可能）状態のタスクにＣＰＵを割り付けて実行させます。

複数のタスクがREADY(実行可能）状態のときは、タスクの優先度に従ってＣＰＵを割り付けます。

システムを構築する場合は、独立して並列に実行可能な単位に処理を分割し、分割した処理をタスクとして作成します。

タスクは１から３１のタスクIDと呼ぶ番号で識別し、最大３１個のタスクが登録できます。

ニュークリアスは、システムに登録されたこれらのタスクをシステムの外部装置や計算機内部の事象に応じて制御し、タスクはニュークリアスのシステムコール機能を使用してタスク間の必要な連絡を行ないます。

図２－２　にタスクとニュークリアスの関係を示します。

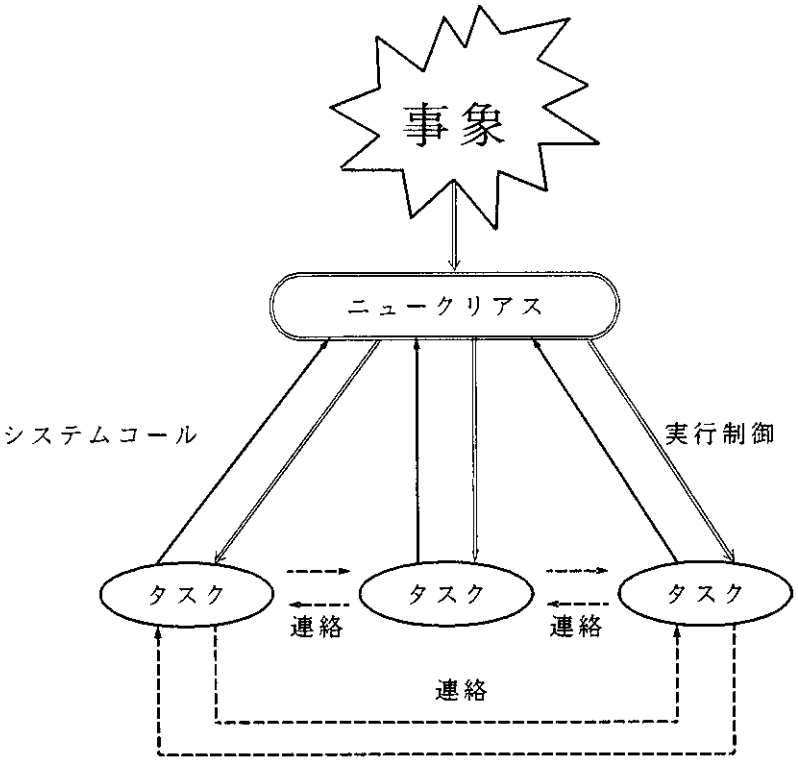


図２－２　タスクとニュークリアスの関係

２．２．２　タスクの並列処理

ニュークリアスは、タスクを互いに独立した並列処理可能なプログラムとして管理し、タスクの実行要求に従って、システム内でのタスクの状態とタスクに付けられた優先度に基づいて複数のタスクを並列に実行します。

優先度は１から３１の値で表わされ、値の小さい方が高い優先度になります。

ＨＩ８－３Ｘにおいて、通常タスクの優先度はタスクIDと同じ値ですが、タスク実行中にchg_priシステムコールで－１に変更することができます。

優先度が－１の間、そのタスクはシステム内で最高の優先度となり、タスクの切替えは行なわれなくなります。

図２－３　にタスクの並列処理の概要を示します。

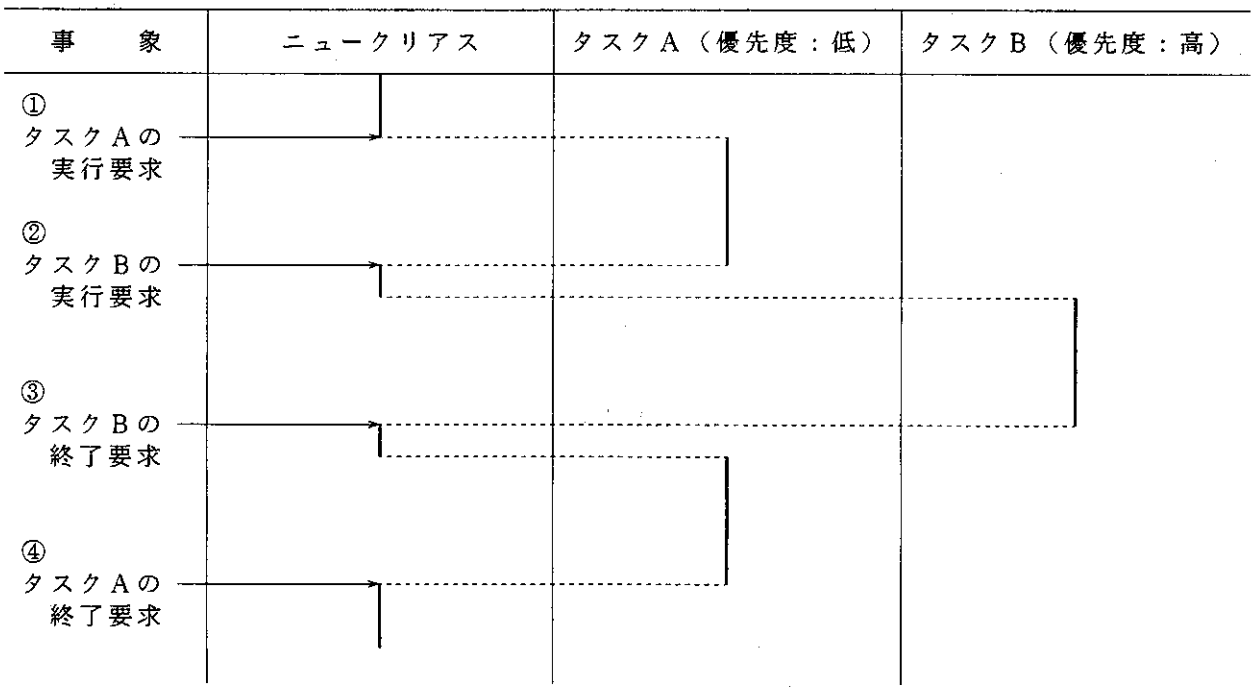


図２－３　タスクの並列処理の概要

（解説）

①タスクＡの実行要求に従いタスクＡを実行します。

②タスクＢの実行要求が発生しました。ニュークリアスはタスクＡの実行を一時中断し、優先度の高いタスクＢを実行します。

③タスクＢの実行終了要求に従い、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の実行終了要求に従いタスクＡの実行を終了します。

このようにニュークリアスは、複数のタスクをタスクに与えられた優先度にもとづいて実行を制御することで、タスクの並列処理を実現します。

２．２．３　タスクの状態

ＨＩ８－３Ｘにおいて、タスクはシステム内において、４つの状態を遷移します。

ニュークリアスは、タスク実行のスケジューリング方式にもとづき、READY(実行可能）状態のタスクの中から１つのタスクを選び、実行します。

READY(実行可能）状態のタスクが存在しない場合、ニュークリアスは割込みが発生するまで待ちます。

(1) ＤＯＲＭＡＮＴ(休止）状態

タスクが登録され、まだ起動されていない状態、または終了後の状態です。

この状態のタスクがsta_tsk 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。

タスクの処理が、ext_tsk システムコールで終了するとDORMANT(休止）状態になります。

(2) ＲＥＡＤＹ(実行可能）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。

DORMANT(休止）状態のタスクが起動されたとき、またはWAIT(待ち）状態のタスクの待ちが解除されたとき、READY(実行可能）状態になります。

(3) ＲＵＮ(実行）状態

ＣＰＵが割り付けられ、現在実行中の状態です。

READY(実行可能）状態のタスクの中から、スケジューリングにより最も高い優先度が与えられたタスクが、RUN(実行）状態になります。

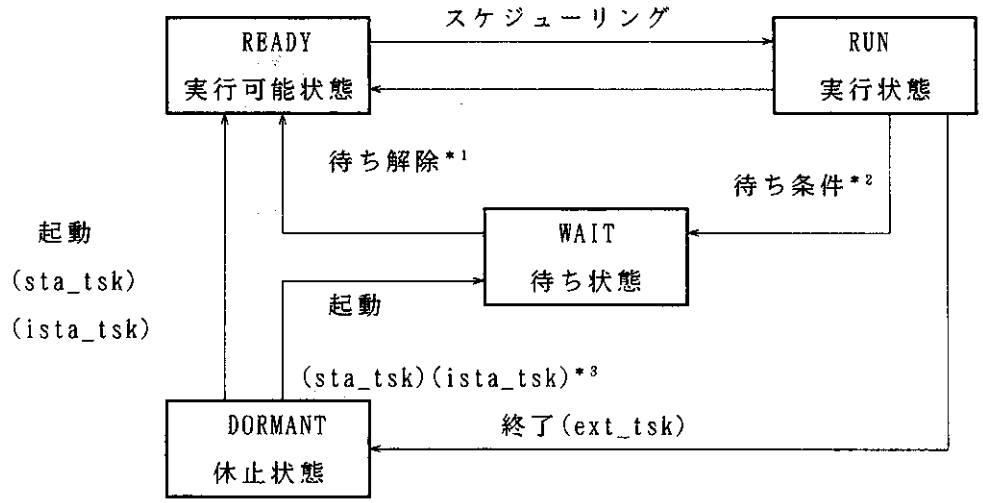
(4) ＷＡＩＴ(待ち）状態

タスクが何らかの事象の発生を待っている状態です。

RUN(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WAIT（待ち）状態になります。

待ちが解除されるとREADY(実行可能）状態になります。

図２－４　にタスクの状態遷移を示します。



【注】*1 待ち解除となる事象、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, ext_tsk(共有タスクスタック機能を使用した場合)

*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

*3 共有タスクスタック機能を使用した場合、待ち状態に移行する。

図２－４　タスクの状態遷移

２．２．４　タスクの登録

ＨＩ８－３Ｘにおいて、タスクをニュークリアスの管理下で制御するためには、タスクをニュークリアスに登録しなければなりません。

タスクのニュークリアスへの登録はシステム構築時に行ない、次のパラメータをセットアップテーブルに設定することによって登録されます。

・タスク先頭アドレス
・タスク初期状態フラグ
・タスク用スタックポインタ

以下に各項目の説明を行ないます。

(1) タスク先頭アドレス

タスクが起動されたときの制御が渡される絶対番地のアドレスです。
タスクの優先度は、登録する順番で決まり、タスクＩＤが小さいほど優先度が高くなります。

(2) タスク初期状態フラグ

タスクが起動されたときのタスクの状態を指定するフラグです。

タスクは、DORMANT(休止）またはREADY(実行可能）状態で登録されます。

(3) タスク用スタックポインタ

タスクが起動されたときのタスクが使用するスタック領域の終端アドレスです。

ＨＩ８－３Ｘはスタックのオーバフローに関して、まったくチェックを行なっていませんので、

アプリケーション側で監視しなければなりません。
タスクのスタックサイズの算出方法については、「付録Ｄ　メモリ容量の算出」を参照してください。

また、Ｈ８－３Ｘの共有タスクスタックライブラリを用いると、複数のタスクでスタック領域を共有することができ、このためスタック領域全体を少なくすることができます。

詳しいタスクの登録方法は、「４．４．２　タスク定義テーブルの作成」を参照してください。

２．２．５　タスクの起動

DORMANT(休止）状態のタスクが起動され、READY(実行可能）状態になる要因には、次のものがあります。

・システム構築時、タスクがREADY(実行可能）状態で登録されている場合
・他のタスクからsta_tsk システムコールが発行された場合
・割込み処理ハンドラなど、非タスク部からista_tskシステムコールが発行された場合

タスクがDORMANT(休止）状態のときは、既に初期化が行なわれていますので、タスクが起動されるとREADY(実行可能）状態になります。
DORMANT(休止）状態時に行なわれる初期化処理を以下に示します。

(1) スタックポインタ(R7)の初期化
スタックポインタ(R7)にタスクが使用するスタックポインタをセットします。
(2) コンディションコードレジスタ(CCR) の初期化
コンディションコードレジスタ(CCR) を０に設定し、割込みマスク解除の状態に移行します。
割込みマスクの変更は、chg_ims システムコールで行ないます。
(3) プログラムカウンタ(PC)の初期化
プログラムカウンタ(PC)を、タスク定義テーブルで指定されたタスク先頭アドレスに設定します。

２．２．６　共有タスクスタック機能によるタスクの起動

ＨＩ８－３Ｘでは、複数のタスクで１つのスタック領域を共有する機能を持っています。このため、全体のタスクが使用するスタック領域の容量を減らすことができます。

共有タスクスタック機能の処理概要を以下に示します。

図２－５　にタスクＡ・Ｂ・Ｃ・Ｄが、同一のタスクスタック（以下、共有スタックと呼ぶ）を使用しています。この状態で、順番にタスクＡ・Ｂ・Ｃ・Ｄを起動し、タスクＡが終了する場合の動作を示します。

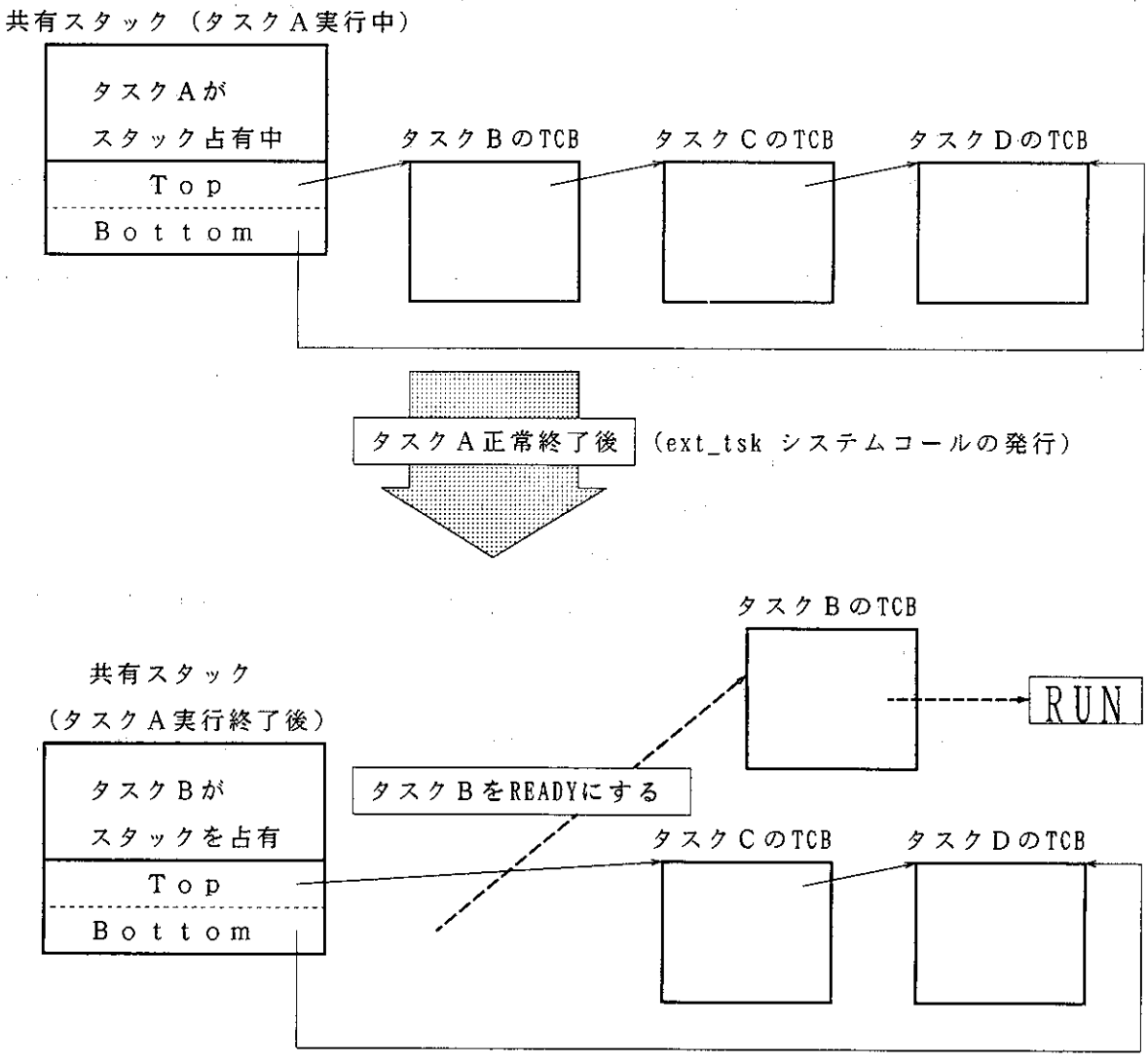


図２－５　共有タスクスタック機能の動作

タスクＡが共有スタックを占有し、この共有スタックを使用（占有）しようとするタスクＢ・Ｃ・Ｄに、順番に起動要求（sta_tsk)を行なった場合、その処理はそれぞれ正常でE_OKを返します。しかし、タスクＢ・Ｃ・Ｄは共有スタックを使用(占有）できないので、タスクＢ・Ｃ・Ｄは共有スタック待ち行列につながれます。

すなわち、実行中のタスクより高い優先度のタスクに起動要求がかかっても、タスクスタックが使用（占有）されている場合は、起動要求されたタスクはREADY(実行可能）状態にならず、WAIT（待ち）状態として扱われます。

共有スタック待ち行列は、FIFO(First-In First-Out)で管理されます。

共有スタックを使用しているタスクが処理を終了(ext_tsk)し、スタックを解放します。その後、ニュークリアスは共有スタック待ち行列の先頭につながれているタスクをREADY(実行可能）状態に移行させます。

共有スタックを使用しているタスクが、wai_tsk やwai_flg などの自タスクが待ちになるようなシステムコールを発行しても、共有スタックは解放されず、この共有スタック待ちのタスクは共有スタック待ち行列につながれたままで、起動することはありません。

また、共有スタックを使用するタスクが１つの場合は、共有スタック待ちは起りません。

図２－６　に共有タスクスタック機能を付加した場合のタスク間の状態推移を示します。

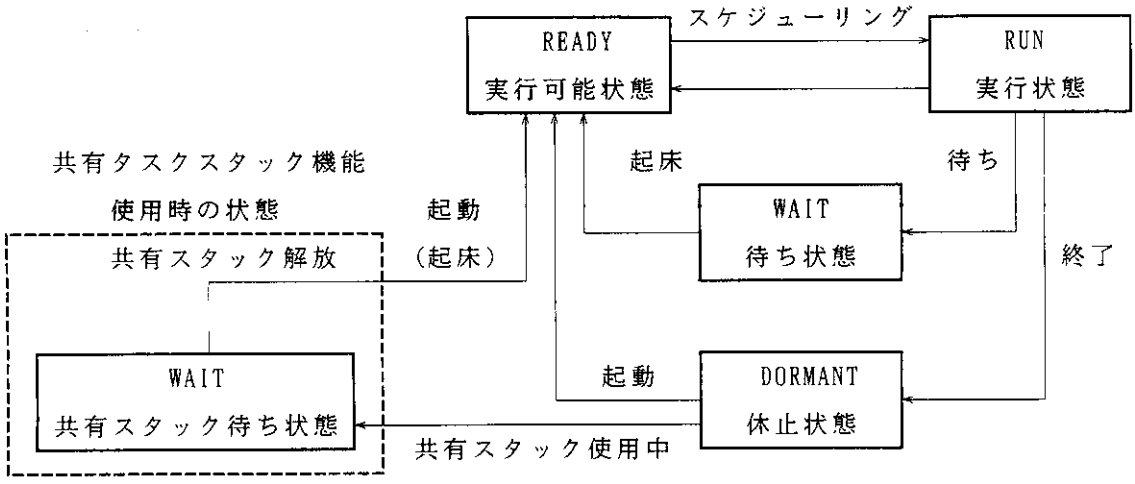


図２－６　共有タスクスタック機能のタスク間の状態推移

２．２．７　タスクの中断・再開

タスクの実行中にいろいろな割込みが入る毎に、タスクは中断され、中断要因が解除されるとまた以前の状態に復帰します。

表２－１　にタスク実行中断の要因と期間を示します。

表２－１　中断の要因と期間

項番	中断の要因		中断の期間
1	自ら中断となる場合	slp_tsk システムコール	wup_tsk・iwup_tsk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
		wai_flg システムコール	イベントフラグの待ち解除条件が成立するまで
		wai_sem システムコール	セマフォで管理されている資源が獲得できるまで
		rcv_msg システムコール	メイルボックスにメッセージが送られるまで
2	割込みにより中断させられる場合	タイマ割込み	タイマからの割込みが発生し、タイマ処理実行後、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、その割込み処理ハンドラ実行後、もとのタスクに戻ってくるまで

２．２．８　タスクの終了

タスクの終了とは、ext_tsk システムコールを使用して処理終了を宣言しDORMANT(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。

タスクは、終了する前に自分が占有していた資源（セマフォ）を自動的に解放することはありません。

実行中に確保した資源（セマフォ）は、タスク終了前に解放させる必要があります。

２．２．９　タスク実行中の割込みのマスク

タスク実行中にchg_imsシステムコールを使用して、割込みをマスク(禁止）することができます。

タスク内で割込みをマスクすると、ニュークリアスは割込みがマスクされている期間、そのタスクを割込み処理ハンドラ（非タスク部）と同一の扱いをします。

割込みをマスクしてタスクが実行しているときは、通常にタスクが実行しているときと比べ以下の点が異なります。

(1) 割込みマスク時発行不可能なシステムコール

割込みをマスクしている期間は、ista_tsk, iwup_tsk, iset_flg, isig_sem, isnd_msg システムコールを発行しないでください。これらのシステムコールを発行した場合システムの動作は保障されませんので注意してください。

タスク部専用のシステムコールを発行すると、コンテキストエラー(E_CTX)になります。

(2) ret_int システムコール

タスク内で割込みをマスクしている期間中は、ret_int システムコールを発行しないでください。

発行した場合、システムの動作は保障されませんので注意してください。

２．２．１０　タスク作成上の注意

タスクを作成する場合は、以下に示す点に留意してください。

(1) コンディションコードレジスタ(CCR) のユーザビット変更の注意

Ｈ８／３００のコンディションコードレジスタ(CCR) はユーザビットとして、2^6 ビットと2^4ビットを使用できますが、このユーザビットはＨＩ８－３Ｘの制御に使用されるため、絶対に変更しないでください。

ユーザビットを変更した場合、動作は保障されません。

(2) 割込みをマスクしたときの注意

ＨＩ８－３Ｘは、タスク部と非タスク部の違いを現在の割込みレベルで判断しますので、タスク部内でも割込みをマスクするとその部分は非タスク部と判断されます。

そのため、割込みマスク状態でret_intシステムコールを発行してもエラーとならず、システムの動作は保障されません。

(3) レジスタ選択の注意

標準で提供されるＨＩ８－３Ｘのライブラリは、タスクでＲ０～Ｒ３までのレジスタを保障したものです。

ＨＩ８－３Ｘは、タスクで保障しているレジスタを変更することができますが、レジスタＲ０～Ｒ３までを保障した場合、レジスタＲ４～Ｒ６はシステム全体（タスク、割込み処理ハンドラ）で共通のレジスタとして使用することができます。したがって、ユーザはレジスタの使用方法を考慮して、応用システムを作成してください。

また、Ｃ言語で作成する場合は、レジスタ選択をＲ０～Ｒ６すべてを保障したライブラリを作成し、使用してください。

２．３　同期・通信管理

２．３．１　同期・通信管理の概要

Ｈ８－３Ｘでは、タスク間での同期・通信処理を行なうための機能として、イベントフラグ・セマフォ・メイルボックスの３種類があります。

イベントフラグはタスク間の高速な同期処理に、セマフォは資源の排他制御に、メイルボックスはデータの受け渡しが必要な処理に使用します。

２．３．２　イベントフラグ

イベントフラグは、事象の発生の有無を１ビットのフラグで管理することによりタスク間の同期処理を行なう機能です。

Ｈ８－３Ｘでは、各事象に対応した８ビットのフラグをひとまとめにしたものを、１つのイベントフラグとして管理します。

イベントフラグは１から３１のイベントフラグIDと呼ばれる番号で識別され、最大３１個まで管理できます。

タスクはイベントフラグに対して事象の発生を組み合わせて持つことができ、１つのイベントフラグに１つのタスクだけが事象の発生を待つことが許されます。

事象の発生は、イベントフラグの各事象の発生に対応したフラグをセットすることで待ちタスクに通知します。

イベントフラグを使用した同期処理はフラグのオン・オフのみで行なわれるため、高速に処理することができます。

イベントフラグの初期値は、H'00です。

イベントフラグは、以下のシステムコールによって操作されます。

- set_flg ・iset_flg システムコール（事象の発生を通知する）
- clr_flg システムコール（イベントフラグをクリアする）
- wai_flg システムコール（事象の発生を待つ）
- pol_flg システムコール（ポーリング：事象の発生を得る）
- flg_sts システムコール（イベントフラグ状態の参照）

図２－７　にイベントフラグの操作を示します。

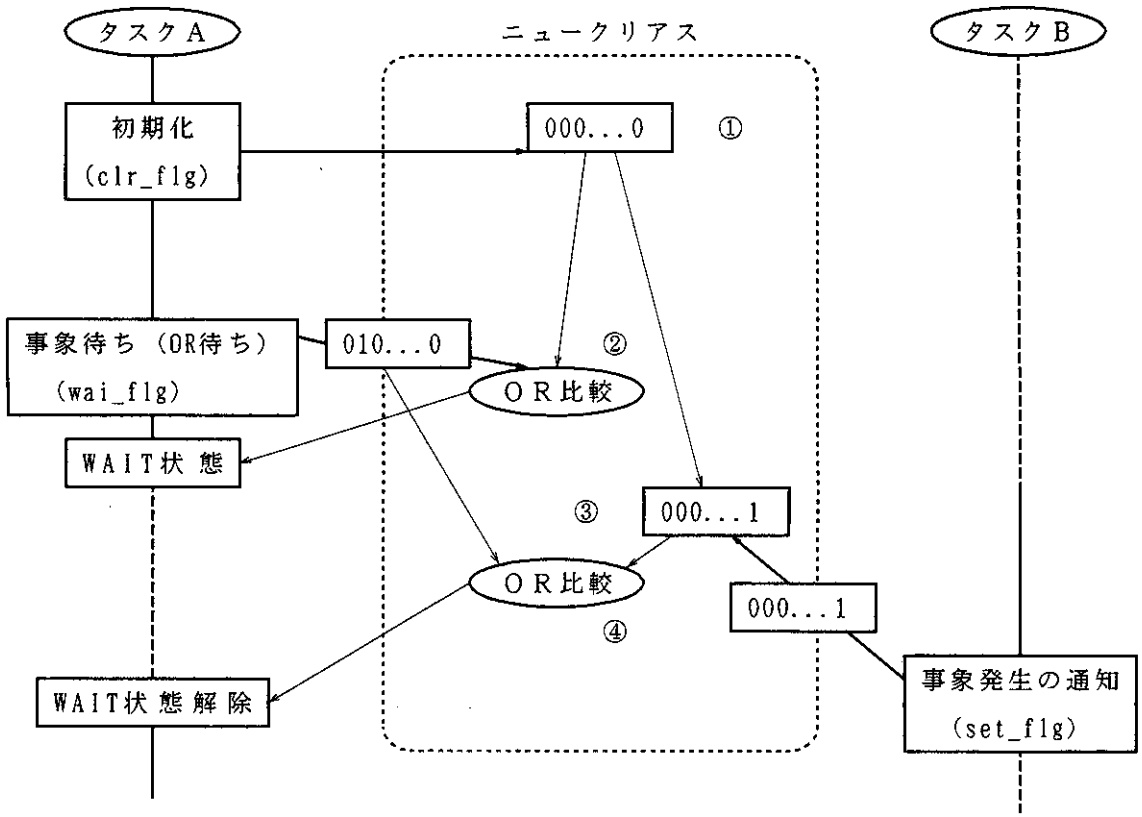


図２－７　イベントフラグの操作

（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事象がまったく発生していないので、タスクＡはＯＲ待ちのモードで、事象の発生を待っています。

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WAIT(待ち) 状態から解除されます。

(1) 事象の発生と待ちの解除

事象の発生の通知にはset_flg・iset_flgシステムコールを使用して、発生した事象に対応したビットをＯＮにした８ビットのビットパターンをイベントフラグに設定することにより行ないます。

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。

ニュークリアスはset_flg システムコールが発行されると、イベントフラグの内容と設定するビットパターンとの論理和(OR)を発生事象ビットパターンとしてセットし、待ちタスクの待ち解除条件が成立するかどうかを調べます。

待ちタスクの待ち解除条件が成立すると、待ちタスクをREADY(実行可能) 状態に移行して実行を再開させます。

実行が再開されたタスクには、wai_flg のリターンパラメータとしてイベントフラグの内容が

返されます。

イベントフラグの内容は、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あればクリアされ、リターンパラメータとしてクリア直前のイベントフラグの内容が返されます。

(2) イベントフラグのクリア

イベントフラグのクリアには、clr_flgシステムコールを使用します。

イベントフラグのクリアは、イベントフラグの内容とクリアするビットパターンの論理積(AND)により行ないます。

(3) 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。

ニュークリアスはwai_flgシステムコールが発行されると、事象が発生しているかを調べます。

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、発生していない場合、wai_flgシステムコールを発行したタスクを事象が発生するまでWAIT(待ち) 状態に移行します。

事象の発生待ちを行なわない(ポーリング) 場合はpol_flgシステムコールを使用します。

pol_flgシステムコールでは、事象が発生していれば正常終了し、発生していなければ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）。

wai_flg・pol_flgシステムコールでは、８ビットの待ち事象のビットパターンと事象の待ち条件を指定します。

１つのイベントフラグには１つのタスクだけが事象の発生を待つことができ、事象の待ち条件には”ＡＮＤ待ち”と”ＯＲ待ち”があります。

”ＡＮＤ待ち”は、ビットパターンの１で指定したビットに対応するすべての事象の発生を待ち、”ＯＲ待ち”は、ビットパターンの１で指定したビットに対応する事象が少なくとも１つ発生するのを待ちます。

”ＡＮＤ待ち”と”ＯＲ待ち”の条件をまとめると以下のようになります。

・ＡＮＤ待ちの解除条件

(イベントフラグ) ∩(待ち事象ビットパターン) ＝(待ち事象ビットパターン)

・ＯＲ待ちの解除条件

(イベントフラグ) ∩(待ち事象ビットパターン) ≠ 0

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あれば、イベントフラグの内容はクリアされ、リターンパラメータとしてクリア直前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以下の情報を返します。

・現在のイベントフラグのビットパターン

・イベントフラグの待ちタスクID

２．３．３　セマフォ

セマフォは、資源の排他制御を行なう機能です。

セマフォには、使用可能な資源数を表わす非負のカウンタ(セマフォカウンタ：１～255) があり、このカウンタの値により、資源の排他制御を行ないます。

セマフォは１から３１のセマフォIDと呼ばれる番号で識別され、最大３１個まで管理できます。

タスクはセマフォに対して資源の占有をＰ命令で要求します。

Ｐ命令処理ではセマフォカウンタの値を調べ、０でなければセマフォカウンタを１だけ減らしてタスクに資源を割り付けます。(ここで資源を割付るとは、実際に何らかの資源をタスクに与えるのではなく、資源要求を行なったタスクの実行を中断せず、そのまま続行することを意味します。)

０であれば、資源が他のタスクからＶ命令で解放されるまでＰ命令を発行したタスクはセマフォの待ち行列につながれます。

セマフォカウンタの初期値は１です。

セマフォの待ち行列はFIFO(First-In First-Out)で管理されます。

セマフォは、以下のシステムコールによって操作されます。

- wai_sem システムコール(Ｐ命令：資源を要求する)
- preq_semシステムコール(ポーリング：資源を得る)
- sig_sem ・isig_semシステムコール(Ｖ命令：資源を解放する)
- sem_sts システムコール(セマフォ状態の参照)

図２－８　にセマフォによる資源の排他制御の例を示します。

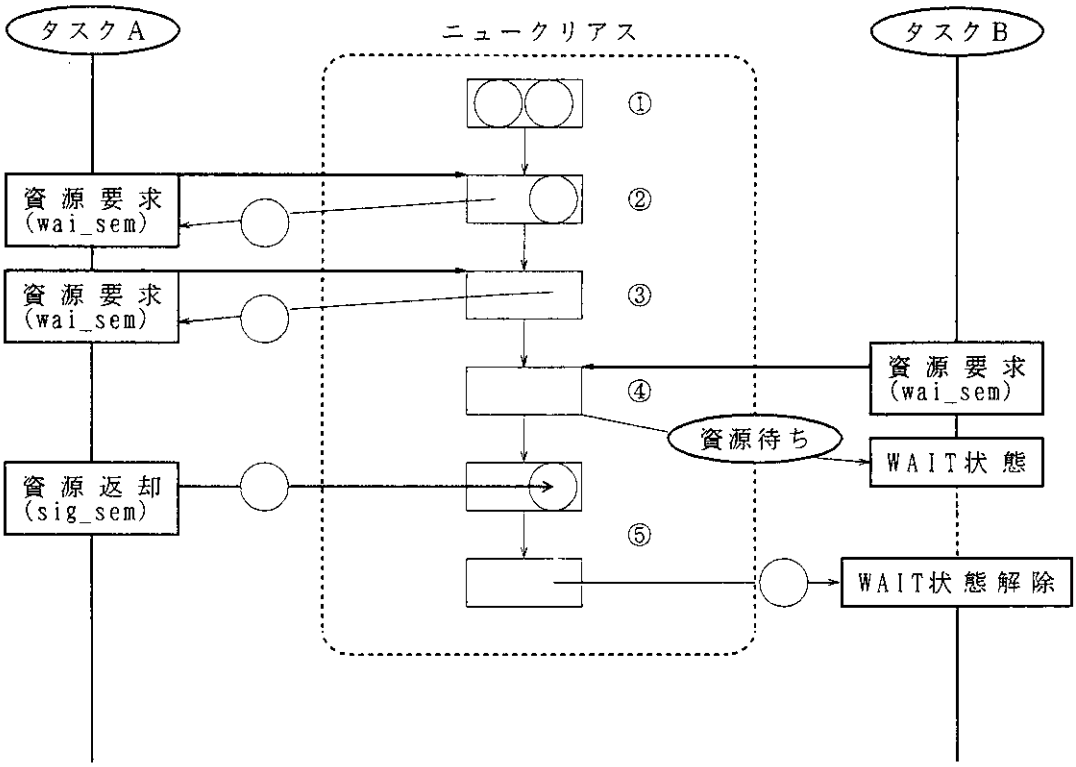


図２－８　セマフォによる資源排他制御例

（解説）

①最初、資源（セマフォ）が２つあります。

②タスクＡが１個の資源（セマフォ）を要求したので、資源（セマフォ）が１つになります。

③タスクＡがさらに１個の資源（セマフォ）を要求したので、資源（セマフォ）が０になります。

④タスクＢが１個の資源（セマフォ）を要求したが、資源（セマフォ）がないために、タスクＢはWAIT(待ち）状態となります。

⑤タスクＡが１個の資源（セマフォ）を解放したため，１個の資源を要求したタスクＢに資源（セマフォ）が割り付けられ、タスクＢはWAIT（待ち）状態から解除されます。

(1) 資源の占有要求(P命令)

資源の占有要求は、wai_sem(P命令) システムコールを使用します。

wai_sem システムコールは、1回の発行で1つの資源を占有します。

ニュークリアスはwai_semシステムコールが発行されると、使用可能資源数（セマフォカウント値）が0かどうかを調べます。

セマフォカウンタが0でなければ、セマフォカウンタを1だけ減らしてwai_sem システムコールを発行したタスクに資源を割り付け、0であれば、wai_semシステムコールを発行したタスクをそのセマフォの待ち行列につなぎ、WAIT（待ち）状態に移行します。

セマフォの待ち行列は、FIFO(First-In First-Out)で管理されます。

資源占有要求の待ちを行なわない(ポーリング) 場合は、preq_semシステムコールを使用します。

preq_semシステムコールでは、資源の占有が可能であれば正常終了し、不可能であればエラーリターンします（エラーコードとしてE_PLFAILを返します）。

(2) 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。

ニュークリアスはsig_sem・isig_semシステムコールが発行されると、セマフォに待ちタスクが存在するかどうかを調べます。

セマフォに待ちタスクが存在する場合は、待ち行列の先頭のタスクにP命令処理同様に資源を割り付け、待ち行列からはずし実行を再開します。

待ちタスクが存在しない場合は、セマフォカウント値が最大値(255) を超えなければ、セマフォカウンタを1だけ増し、最大値を超える場合は、エラーリターンします(エラーコードとしてE_QOVRを返します）。

(3) 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以下の情報を返します。

- 現在のセマフォカウント値
- セマフォの待ち行列先頭タスクのタスクID

２．３．４　メイルボックス

メイルボックスは、タスク間でメッセージと呼ばれるデータの受け渡しを行なうための機能です。

メイルボックスは、１から３１のメイルボックスIDと呼ばれる番号で識別され、最大３１個まで管理できます。

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、受信するタスクはメイルボックスからメッセージを受け取ります。

メイルボックスに送られたメッセージは、受信するタスクが存在すればメッセージ受信待ち行列の先頭タスクに渡され、タスクが存在しなければメッセージ待ち行列につながれます。

メッセージを受信するタスクは、メイルボックスにメッセージが存在すればメッセージ待ち行列の先頭メッセージを受け取り、メッセージが存在しなければメッセージ受信待ち行列につながれます。

メッセージ待ち行列、メッセージ受信待ち行列は、FIFO(First-in First-Out)で管理されます。

メイルボックスは、以下のシステムコールによって操作されます。

- snd_msg ・isnd_msgシステムコール(メッセージを送信する)
- rcv_msg システムコール(メッセージの受信を待つ)
- prcv_msgシステムコール(ポーリング：メッセージを受信する)
- mbx_sts システムコール(メイルボックス状態の参照)

図２－９　にメイルボックスの状態を示します。

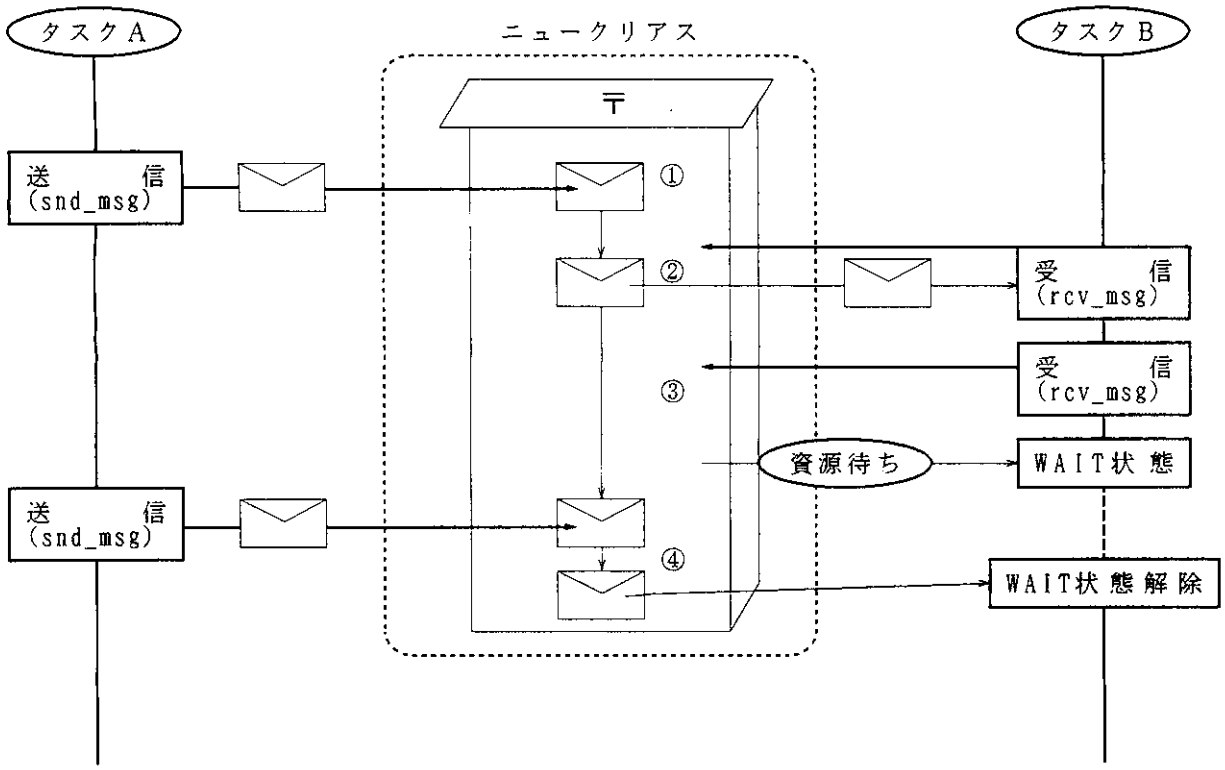


図２－９　メイルボックスの状態

①タスクＡがメッセージを送信したので、メイルボックスに１通のメッセージがあります。
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ないため、タスクＢがWAIT（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たタスクＢにメッセージが渡され、タスクＢはWAIT（待ち）状態から解除されます。

(1) 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。

snd_msg・isnd_msgシステムコールでは、送信するメイルボックスのIDと、送信するメッセージの先頭アドレスを指定します。

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。

ニュークリアスはsnd_msg・isnd_msgシステムコールが発行されると、そのメイルボックスにメッセージの受信待ちタスクが存在するかどうかを調べます。

メッセージの受信を待つタスクが存在する場合、メッセージ受信待ち行列の先頭タスクにメッセージを渡して待ち行列からはずし、実行を再開します。

メッセージの受信を待つタスクが存在しない場合、送られたメッセージをメイルボックスのメッセージ待ち行列につなぎます。

メイルボックスのメッセージ待ち行列は、FIFO(First-In First-Out)で管理されます。

(2) 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。

ニュークリアスはrcv_msgシステムコールが発行されると、その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、そのメッセージをメッセージ待ち行列からはずし、メッセージのアドレスをリターンパラメータとして、rcv_msgシステムコールの発行タスクに返します。

メッセージが存在しない場合は、タスクはメイルボックスのメッセージ受信待ち行列につながれます。

メイルボックスのメッセージ受信待ち行列は、FIFO(First-In First-Out)で管理されます。

メッセージ受信の待ちを行なわない(ポーリング）場合は、prcv_msgシステムコールを使用します。

prcv_msgシステムコールでは、メイルボックスにメッセージが存在するときは正常終了し、メッセージが存在しないときは、エラーリターンします(エラーコードとしてE_PLFAILを返します）。

(3) 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以下の情報を返します。

・メッセージ待ち行列の先頭メッセージのアドレス

・メッセージ受信待ち行列の先頭タスクのタスクID

図２－１０　にメッセージのフォーマットを示します。

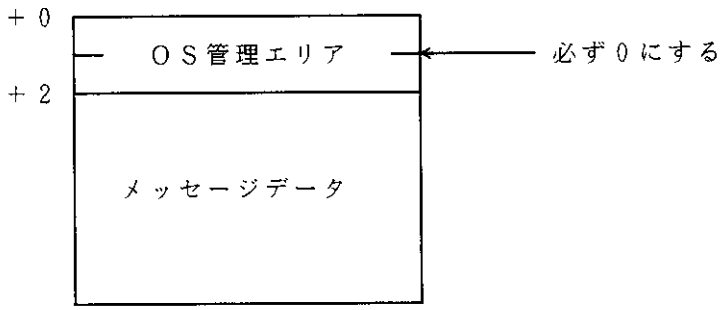


図２－１０　メッセージフォーマット

メッセージの先頭２バイトは、ニュークリアスがメッセージを管理するための領域です。

この領域が０の場合、メッセージがメイルボックスのメッセージ行列につながれていないことを表わし、０でない場合は、メッセージ行列につながれていることを表わします。

よって、メッセージ送信時、この領域に必ず０を設定してください。メッセージの先頭２バイトに０を設定し送信することにより、同一メッセージの二重送信を防ぐことができます。

ニュークリアスはメッセージが送信されると、メッセージの先頭２バイト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待ち行列につながれます。

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待ち行列から外し、先頭２バイトに０を設定してタスクに渡します。

よって、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先頭２バイトに０クリアするだけで送受信できます。

【注】　メッセージ送受信の注意事項

メッセージの送受信は、データそのものが転送されるわけではなく、そのメッセージの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。

また、メッセージの先頭２バイトはニュークリアスが管理する領域です。

メッセージ送信時、この領域に必ず０を設定してください。

メッセージを送信後、受信される前にこの領域を破壊すると、メッセージの送受信が正常に行なわれません。

２．４　時間管理

２．４．１　時間管理の概要

Ｈ８－３Ｘは、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して時間の管理を行ない、次の機能を提供します。

・時間の参照・設定
・時間によるタスクの実行制御

(1) 時間

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(タイマ値）を管理します。

(2) 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うため、時間管理を利用します。

２．４．２　ハードウェアタイマとソフトウェアタイマ

Ｈ８－３Ｘのwai_tsk・set_tim・get_timシステムコールを使用するためには、必ず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。

ニュークリアスは、一定周期のハードウェアタイマから割込みをカウントし、時間を管理します。

システム内での時間は、このハードウェアタイマの周期時間(tC)を単位とします。システム内での時間と、現実の時間には次の関係があります。

現実の時間	＝	システム内の時間	×	tC(ハードウェアタイマ周期時間)

【注】　ハードウェアタイマの周期が10msecの場合の、システム内の時間10は、100msec を表わします。

図２－１１　に、ハードウェアタイマとソフトウェアタイマの関係を示します。

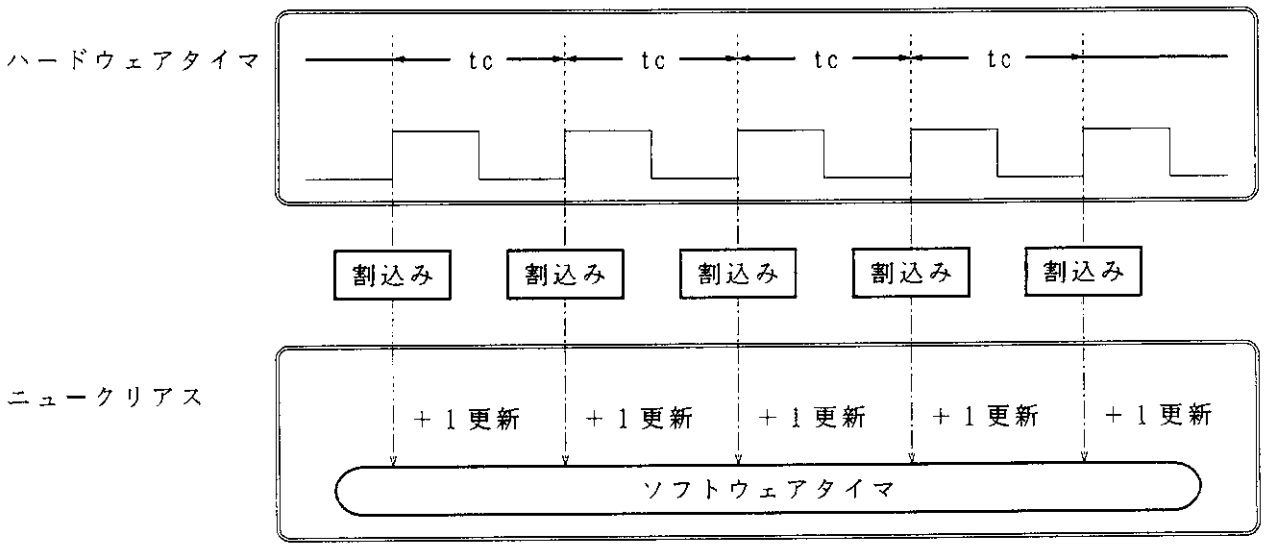


図２－１１　ハードウェアタイマとソフトウェアタイマ

２．４．３　時間の管理

ニュークリアスは、32ビットのタイマカウンタを持ち、ハードウェアタイマの割込みごとにカウンタを更新（＋１）します。

タイマカウンタは符号付き32ビット* H'00000000～H'7FFFFFFFの値の範囲です。このタイマ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。

【注】* タイマカウンタの最上位ビットは符号のビットです。

(1) 時間の設定・参照

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。

パラメータは、３２ビットのタイマカウンタの値です。

図２－１２　に時間の設定および参照を示します。

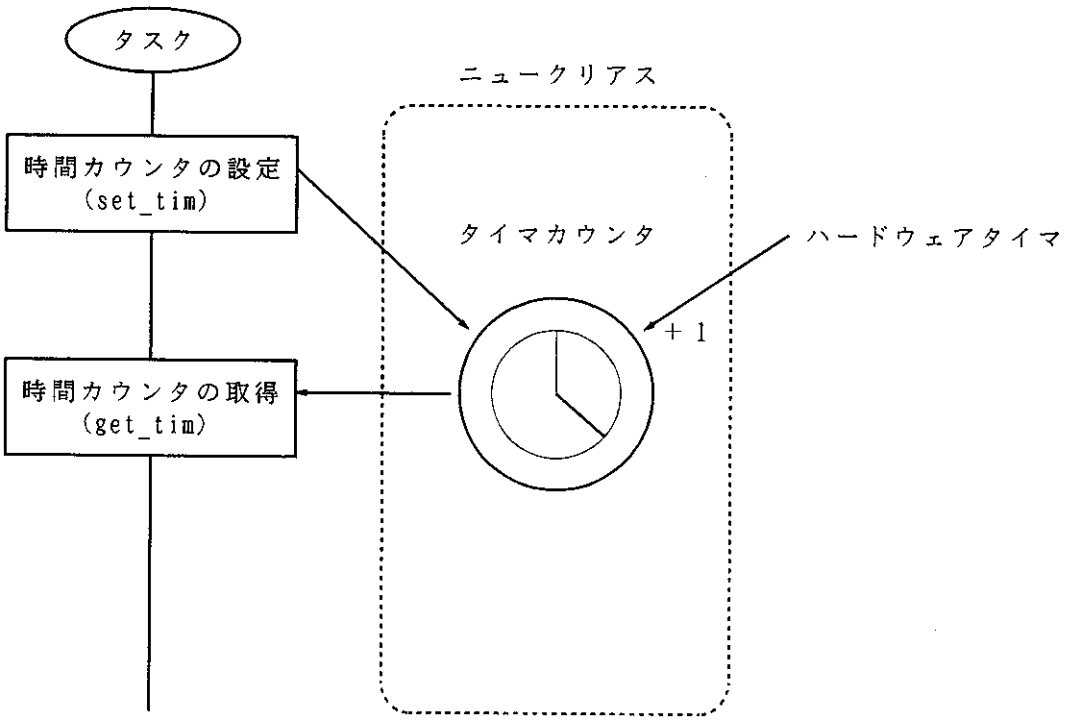


図２－１２　時間の設定および参照

タイマカウンタは符号付き３２ビットであり、時間の最大値は、ハードウェアタイマの周期時間が１０msecであれば約２５０日に相当します。

２．４．４　タイマ処理ハンドラ（タイマ初期設定ルーチン、タイマ割込み処理ハンドラ）

Ｈ８－３Ｘでは、標準のハードウェアタイマとしてＨ８／３００内蔵のフリーランニングタイマ（以下、ＦＲＴと略します）を使用します。ＦＲＴ以外のハードウェアタイマを使用する場合、標準提供されているタイマ処理ハンドラ『h3xtim25.mar』を参考に、使用するタイマ用のタイマ処理ハンドラを作成して登録してください。

詳しい作成方法は、「４．３．１　タイマ初期設定ルーチン・タイマ割込みリセット処理の作成」を参照してください。

タイマ処理ハンドラは、タイマ初期設定ルーチンとタイマ割込み処理ハンドラにより構成されます。

図２－１３　にタイマ処理ハンドラの処理を示します。

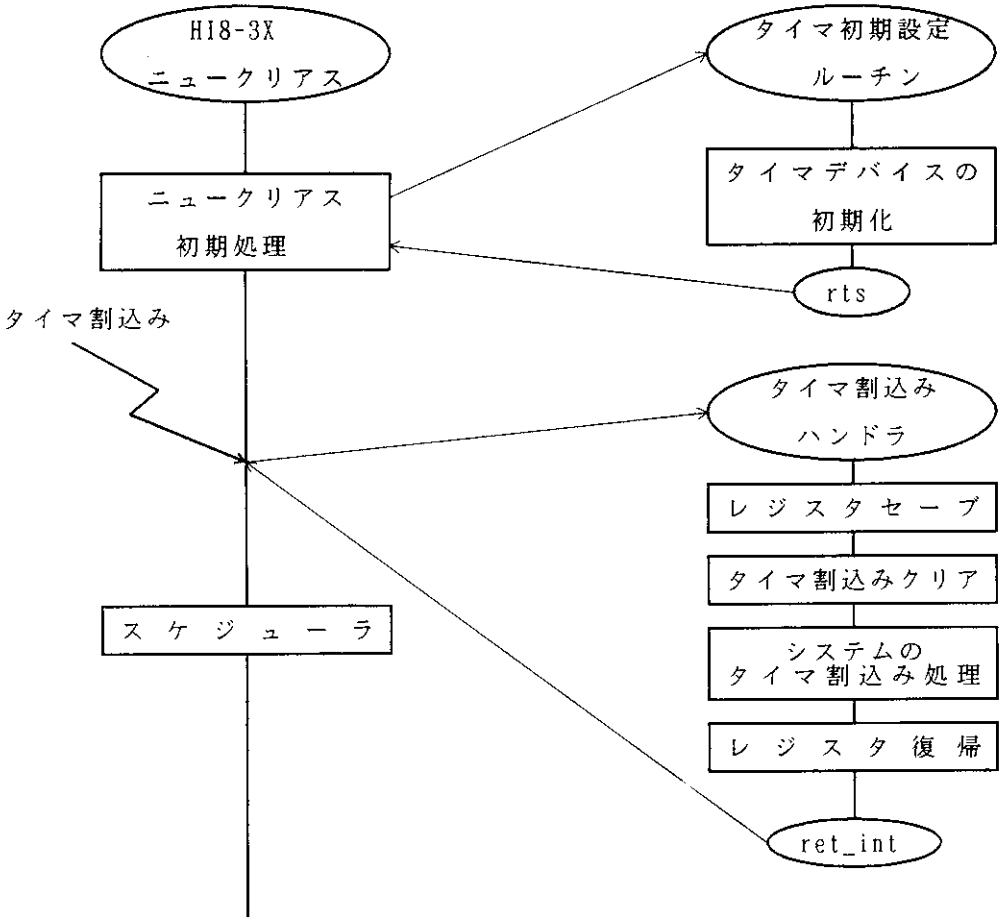


図２－１３　タイマ処理ハンドラの処理

(1) タイマ初期設定ルーチン

ハードウェアタイマに使用するタイマの初期設定を行ないます。

(2) タイマ割込み処理ハンドラ(タイマ割込みリセット部・システムのタイマ割込み部)

タイマ割込み処理ハンドラは、タイマ割込みリセット部とシステムのタイマ割込み部により構成されます。タイマからの割込みが発生すると、タイマ割込みのクリア（タイマ割込みリセット部)、タイマカウントアップ、時間によるタスクの実行制御（システムのタイマ割込み部）を行ないます。

Ｈ８／３００内蔵のＦＲＴ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部を作成しなければなりません。

タイマ割込み処理ハンドラは、基本的には割込み処理ハンドラと同じ処理ですが、ニュークリアスに対して特別な処理を行なっていますので、タイマ割込み処理ハンドラを作成する場合、ニュークリアスに対しての処理は変更しないでください。

表２－２　にタイマ初期設定ルーチンの処理条件を示します。

表２－２　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 割込みマスクを解除しないでください	
2	使用できる レジスタ	・R0～R6	終了前に起動時の値に戻してください
3	SP(R7)	・ニュークリアス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	スタック領域のサイズは、「付録Ｄ．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	

２．４．５　タイマ処理ハンドラの登録

タイマ処理ハンドラの構築には、下記プログラムを作成後、次の登録作業が必要です。

・タイマ初期設定ルーチン(標準提供ファイル名『h3xtim25.mar』)

・タイマ割込み処理ハンドラ(標準提供ファイル名『h3xtim25.mar』)

(1) タイマ初期設定ルーチンの登録

タイマを初期化するため、タイマ初期設定ルーチンの先頭アドレスをセットアップテーブルに登録します。

Ｈ８／３２５内蔵ＦＲＴを使用する場合は、標準提供のタイマ初期設定ルーチンを使用してください。

システム定義テーブルにタイマ初期設定ルーチンの先頭アドレスとして『H_3X_TIM_INI』と登録します。

それ以外のタイマを使用する場合は、標準提供のタイマ初期設定ルーチンにしたがって、使用するタイマ用のタイマ初期設定ルーチンを作成してください。

(2) タイマ割込み処理ハンドラの登録

タイマ割込みによりＨＩ８－３Ｘの時間管理等を行なうため、タイマ割込み処理ハンドラの先頭アドレスを割込みベクタテーブルに登録します。

Ｈ８／３２５内蔵ＦＲＴを使用する場合は、標準提供のタイマ割込み処理ハンドラを使用してください。

それ以外のタイマを使用する場合は、標準提供のタイマ割込み処理ハンドラにしたがって、使用するタイマ用のタイマ割込み処理ハンドラを作成してください。

２．５　割込み管理

２．５．１　割込み管理の概要

ＨＩ８－３Ｘでは、システム構築時に必要に応じて割込み処理ハンドラを作成し、該当する割込みベクタテーブルにハンドラの先頭アドレスを登録します。

割込み処理ハンドラを定義すると、割込み発生時にＨＩ８－３Ｘの介入を受けず直接割込み処理ハンドラに制御が移ります。

(1) 割込み

割込みとは、Ｈ８／３００の割込み(Interrupt) です。

リアルタイム制御システムにおいて割込みは、事象発生を通知する重要な手段です。

(2)　割込み処理ハンドラ

割込み処理ハンドラは、割込みが発生するとニュークリアスの介入なしにただちに起動され、割込みの処理を行なうプログラムで、必要に応じてＨ８／３００の割込みベクタごとに作成し登録できます。

図２－１４　に割込みベクタテーブルと割込み処理ハンドラとの関連を示します。

【注】 *H8/325シリーズの割込みベクタテーブルです。他のＨ８／３００シリーズを使用する場合は、当該ＣＰＵのハードウェアマニュアルを参照してください。

図２－１４　割込みベクタテーブルと割込み処理ハンドラとの関連

図２－１５　に割込み処理ハンドラの制御移行を示します。

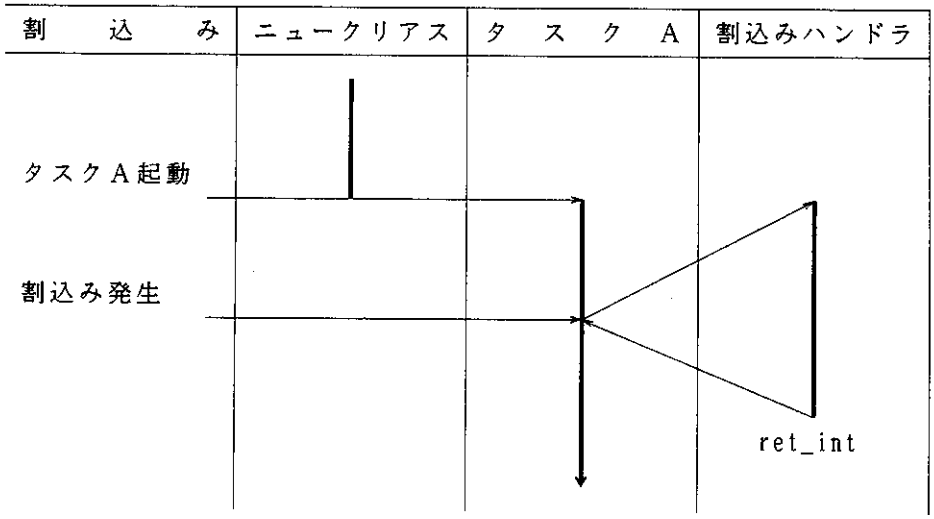


図２－１５　割込み処理ハンドラの制御移行

タスクＡが処理中、割込みIRQ0が発生すると制御はIRQ0割込み処理ハンドラに移ります。

非タスク部で起動された割込み処理ハンドラの場合、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。

タスク部で起動された割込み処理ハンドラの場合、処理が終了すると、現在READY(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。

２．５．２　割込み処理ハンドラ

割込み要因が発生すると、ニュークリアスの介入なしに制御が割込み処理ハンドラに渡されるので、割込み処理ハンドラは割込み発生時のレジスタ内容を保障しなければなりません。

そのため、割込み処理ハンドラ内で使用するレジスタを、割込み処理ハンドラ用スタック領域に退避し、処理完了後、ret_int システムコール発行前にレジスタを割込み処理ハンドラ用スタック領域から回復する必要があります。

図２－１６　に割込み処理ハンドラのレジスタ退避および回復の概要を示します。

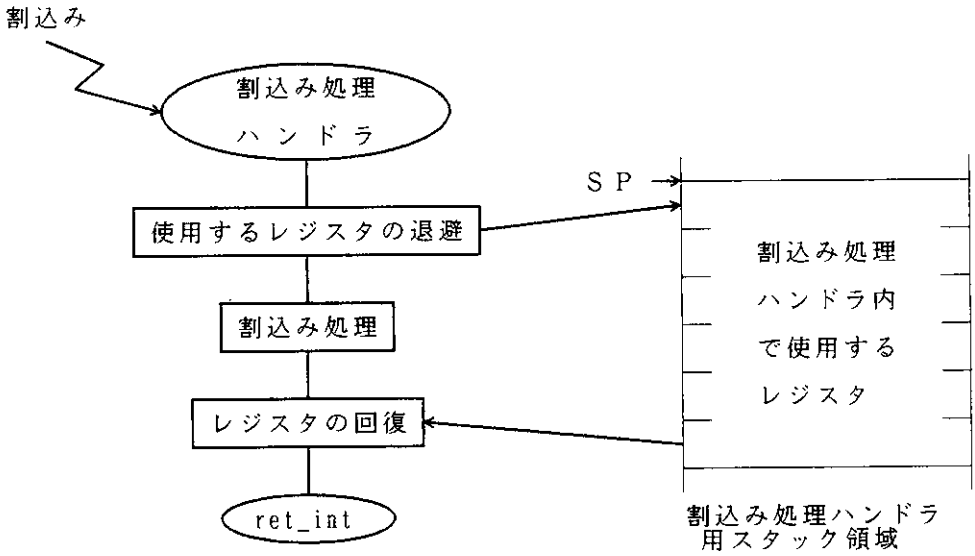


図２－１６　割込み処理ハンドラのレジスタ退避および回復の概要

割込み処理ハンドラは自由に作成することができますが、割込み処理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。

なお、ＮＭＩ（ノンマスカブルインタラプト）の割込み処理ハンドラでは、システムコールを発行することができません。

ＮＭＩ割込み処理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、割込み処理ハンドラに制御が渡されたときと同じ状態にレジスタを戻し、ＲＴＥ命令を実行してください。

表２－３　に割込み処理ハンドラの処理条件を示します。

表２－３　割込み処理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 割込み処理ハンドラ処理中は、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	・R0～R6	終了前に起動時の値に戻してください
3	SP(R7)	・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割込み処理ハンドラからは、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	・スタック領域は、システム構築時に確保し、割込み処理ハンドラ起動時にスタックを切り替えてください	スタック領域のサイズは「付録Ｄ　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、ＲＴＥ命令（ＮＭＩ割込み処理ハンドラのみ）により処理を終了します	

２．５．３　未定義割込み

ＨＩ８－３Ｘは、割込み処理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、レジスタR0に発生したエラーの種類（未定義割込み：H'2222）を設定し、割込みマスク状態でシステムの異常終了処理に制御を移します。

標準提供のシステム異常終了処理では、無限ループに入ります。

システムの異常終了処理については、「２．７　システムの異常終了処理」を参照してください。

２．５．４　割込み処理ハンドラの注意事項

(1) 割込み処理ハンドラはタスクとは独立した処理であり、独自のスタック領域をもちます。

このため割込み処理ハンドラは、起動時に使用するレジスタを割込み処理ハンドラ用スタック領域に退避し、復帰（ret_int発行）前にレジスタを回復する必要があります。

(2) 割込み処理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READY(実行可能）状態となっても、必ず割込み処理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。

割込み処理ハンドラを終了する場合は、必ずret_int システムコールを使用してください。

(3) Ｈ８／３００のコンディションコードレジスタ(CCR) にはユーザビットとして、2^6ビットと2^4ビットがあります。このユーザビットはＨ１８－３Ｘの制御に使用されるため、絶対に変更しないでください。

ユーザビットを変更した場合、動作は保障されません。

(4) 割込み処理ハンドラ実行中には、割込みマスク状態を解除しないでください。

割込みマスク状態を解除した場合、システムの動作は保障されませんので注意してください。

(5) 割込み処理ハンドラから次の条件を含むシステムコールを発行しないでください。

- 待ちを含むシステムコール（wai_tsk, slp_tsk, wai_flg, wai_sem, rcv_msg)
- 自分自身指定のシステムコール（ext_tsk, chg_pri, get_tid, can_wup, tsk_sts)

２．６　システム起動処理

システムの起動処理は、システムの構築も関係しますので、「４．　システムの構築」も参照してください。

２．６．１　システム起動処理の処理概要

パラメータチェックモジュールライブラリを組込んだシステムを起動すると、以下のシステム起動処理を行ないます。

(1) 定義テーブルのチェック

チェックするテーブルを以下に示します。

・システム定義テーブル
・タスク定義テーブル
・各定義数テーブル(タスク・イベントフラグ・セマフォ・メイルボックス)
・ＲＡＭ定義テーブル

各テーブルが正常であれば、システム環境を初期化し、正常でなければ、システム異常終了します。

(2) タイマ初期設定ルーチン

タイマ初期設定ルーチンがセットアップファイル『h3xsetup.src』のテーブル『H$TIM_INI』に登録されていれば、タイマ初期設定ルーチンを呼び出し、処理を行ないます。

(3) ユーザリセットルーチン

ユーザリセットルーチンがセットアップファイル『h3xsetup.src』のテーブル『H$USR_RST』に登録されていれば、ユーザリセットルーチンを呼び出し、処理を行ないます。

(4) タスク起動

システム起動時にタスクが起動するように登録されていれば、そのタスクを起動します。

起動するタスクがない場合、システムアイドリングをして割込みを待ち続けます。

図２－１７　にシステム起動時の処理概要を示します。

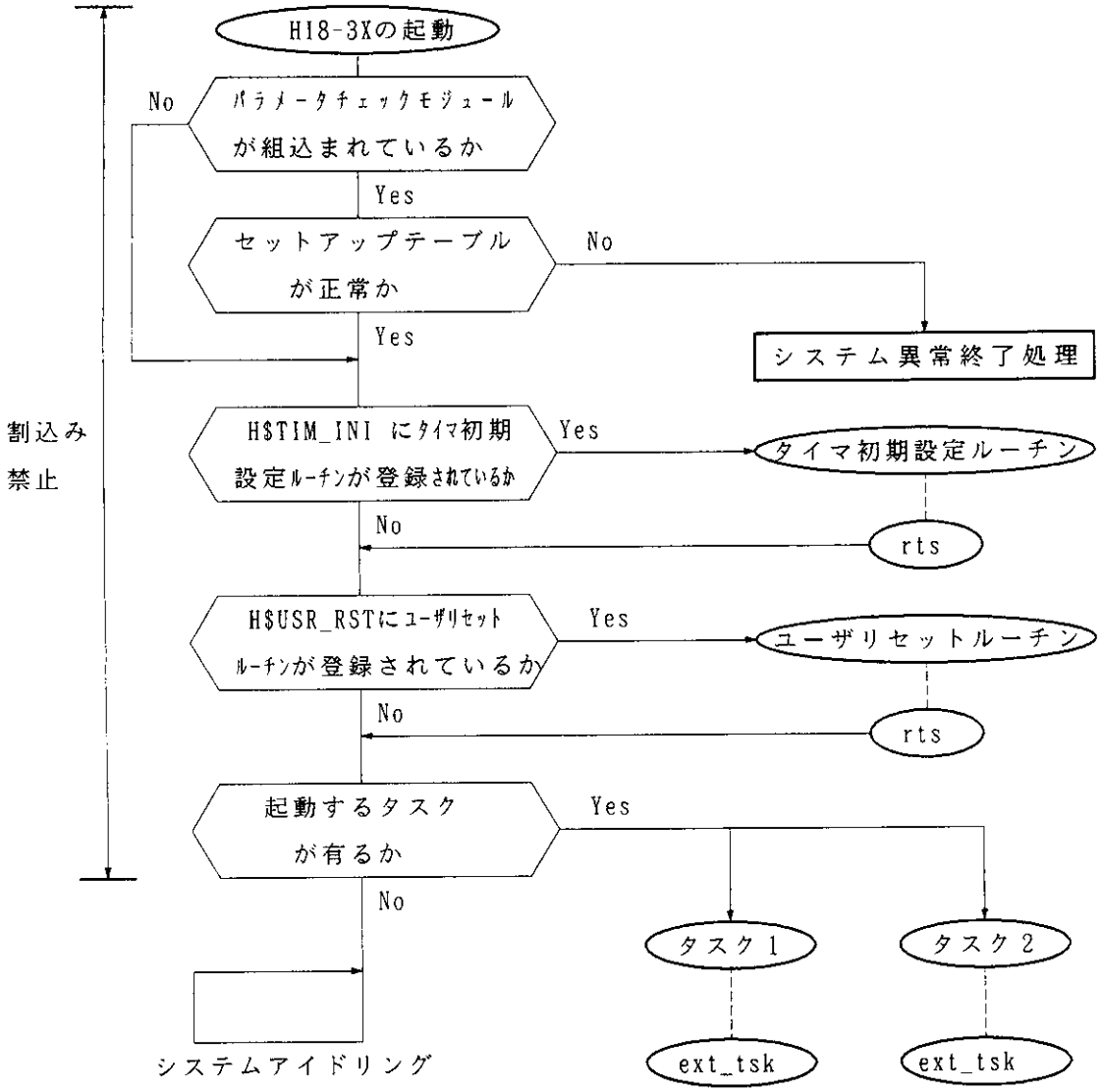


図２－１７　システム起動時の処理概要

２．６．２　ユーザリセットルーチン

ユーザリセットルーチンとは、Ｈ８－３Ｘ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、必要に応じて作成します。

ユーザリセットルーチンを登録すると、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。

登録する場合、セットアップファイルのテーブル『H$USR_RST』にユーザリセットルーチンの先頭アドレスを登録してください。

表２－４　にユーザリセットルーチンの処理条件を示します。

表２－４　ユーザリセット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 ユーザリセットルーチン実行中は 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	・R0～R6	終了前に起動時の値に戻してください
3	SP(R7)	・ニュークリアス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	・非タスク部で使用できるシステム コール(ret_intは除く)	
5	使用できる スタック領域	・スタックを使用する場合、ユーザリセット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、ユーザリセットルーチン起動時にスタックを切り替えてください	スタック領域のサイズは、「付録Ｄ　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	

２．７　システムの異常終了処理

Ｈ８－３Ｘはシステムに異常が起こった場合、割込みをマスクして対応したアドレスに制御を移します。

また、異常終了処理に制御が移るときは、レジスタR0にエラーの種類が設定され、レジスタR1にエラーの情報が設定されます。

標準提供の異常終了処理は、セットアップファイルに組込まれていて、無限ループに入ります。

異常終了時のジャンプ先アドレスのラベルに、異常終了に応じたプログラムを記述することもできます。ただし、Ｈ８－３Ｘの動作は保障できませんので注意してください。

表２－５　にシステムの異常終了時のレジスタ状態を示します。

表２－５　システムの異常終了時のレジスタ状態

R0	R1	原　　因	異常終了時のｼﾞｬﾝﾌﾟ先アドレスのﾗﾍﾞﾙ
H'1111	不正アドレス	セットアップテーブルの不正	H_setup_error *
H'2222	ベクタNo.	未定義割込み	H_unint_error *
H'5555	H'FFEB	非タスク部からのext_tsk システムコール発行	H_ctx_error *
H'5555	H'FFBB	タスク部からのret_int システムコール発行	H_ctx_error *

【注】　* これらのラベルは標準提供のセットアップファイル『h3xsetup.src』にあります。

3．システムコール

3．1　概　要

ＨＩ８－３Ｘはμ ＩＴＲＯＮ仕様に準拠した、３３個のシステムコールを提供します。
ＨＩ８－３Ｘでサポートしているシステムコールは、次の６つに分類されます。

(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
割込み処理ハンドラの終了・割込みマスクの変更および参照を行ないます。
(5) 時間管理システムコール
システムクロックの設定・読込みを行ないます。
(6) システム管理システムコール
ＨＩ８－３Ｘのバージョン識別子を求めます。

表３－１　にシステムコールと、発行可能条件の一覧を示します。

表３－１　システムコール・発行可能条件一覧

	システムコール		タスク/非タスク (T)/(N)	
タスク管理				
1	sta_tsk	タスクを起動する	T	
2	ista_tsk	非タスクからタスクを起動する		N
3	ext_tsk	自タスクを正常終了する	T	
4	chg_pri	自タスク優先度を変更する	T	
5	get_tid	自タスクのIDを得る	T	
6	tsk_sts	タスクの状態を見る	T	N
タスク付属同期管理				
7	slp_tsk	自タスクを待ち状態へ移行する	T	
8	wai_tsk	自タスクを一定時間待ち状態に移行する	T	
9	wup_tsk	待ち状態のタスクを起床する	T	
10	iwup_tsk	同上(非タスク部専用)		N
11	can_wup	タスクの起床要求を無効にする	T	
同期・通信管理				
12	set_flg	イベントフラグをセットする	T	
13	iset_flg	同上(非タスク部専用)		N
14	clr_flg	イベントフラグをクリアする	T	
15	wai_flg	イベントフラグを待つ	T	
16	pol_flg	イベントフラグを得る	T	
17	flg_sts	イベントフラグ状態を参照する	T	N
18	sig_sem	セマフォに対する信号操作(V命令)	T	
19	isig_sem	同上(非タスク部専用)		N
20	wai_sem	セマフォに対する待ち操作(P命令)	T	
21	preq_sem	セマフォ資源を得る	T	
22	sem_sts	セマフォ状態を参照する	T	N
23	snd_msg	メッセージを送信する	T	
24	isnd_msg	同上(非タスク部専用)		N
25	rcv_msg	メッセージの受信を待つ	T	
26	prcv_msg	メッセージを受信する	T	
27	mbx_sts	メイルボックス状態を参照する	T	N
割込み管理				
28	ret_int	割込みハンドラから復帰する		N
29	chg_ims	割込みマスクを変更する	T	
30	ims_sts	割込みマスクを参照する	T	N
時間管理				
31	set_tim	システムクロックを設定する	T	N
32	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				
33	get_ver	ニュークリアス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以下に説明します。
なお、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。
本章は、以下の形式でシステムコールを説明しています。

- システムコール名
- アセンブラフォーマット
- Ｃ言語フォーマット
- エラーコード
- 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、次のプリフィクスを付けています。

p_	ポインタを表わす
pk_	パラメータパケットアドレスを表わす
t_	構造体を表わす
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

ＨＩ８－３Ｘのシステムコールは、実行結果として８ビットの整数をエラーコードとして返します。
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。

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、E_CTX エラーとなりますが、システムコールを発行したプログラムには戻らずに、システム異常終了します。
システムの異常終了処理については、「２．７　システムの異常終了処理」を参照してください。

wai_tskシステムコールを例にエラーコードの説明をします。
エラー名称は　『E_』のプリフィクスを付けて表わされます。

エラーコード(ROH/ercd)：

E_OK	H'00(H'00)	正常終了
E_TNOSPT	H'ED(-H'13)	タイマがサポートされていない
E_ILTIME	H'27(-H'D9)	不正時間指定(tmout≦-2)
E_TMOUT	H'55(-H'AB)	タイムアウト

３．２．３　アセンブラインタフェース

(1) パラメータの表記法

パラメータは、以下のような表現・意味で記述しています。

R0H : レジスタR0H(R0の上位 8ビット）が有効

R0L : レジスタR0L(R0の下位 8ビット）が有効

R1 : レジスタR1（R1の全16ビット）が有効

(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です。R2～R6はシステムコールでは使用しません。

通常、次の目的でレジスタを使用します。

R0H : エラーコード

R0L : ID番号

R1 : その他のパラメータ

また、Ｈ８－３ＸではＨ８／３００のコンディションコードレジスタ(CCR) の２つのユーザビットを使用していますので、プログラムでは、ユーザビットを絶対に変更しないでください。ユーザビットの値を変更した場合、動作はまったく保障されません。

(3) エラーコード

エラーコードは、レジスタR0H にセットされます。

また、コンディションコードレジスタ(CCR) のゼロフラグには反映されません。

(4) システムコールの発行

システムコールはパラメータをレジスタに設定した後、JSR 命令や、JMP 命令(ext_tsk・ret_int のみ）を実行することにより発行されます。

システムコールから戻ってきた場合、R0H 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値は保障（システムコール発行前と同じ）されます。

(5) パラメータの説明

can_wup 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     @can_wup  ····································· ①
```

パラメータ：

```
        R0L     タスクID(tskid:0, 1~31) ····················· ②
```

リターンパラメータ：

```
        R0H     エラーコード(ercd)  ·························· ③
        R1H     キューイングされていた起床要求回数(wupcnt:0, 1)·· ④
```

（解説）

① JSR 命令でシステムコールを発行します。

② タスクIDをレジスタR0L に１バイトで設定します。

③ システムコールのエラーコードがレジスタR0H に１バイトで返ります。

④ キューイングされていた起床要求回数がレジスタR1H に返されます。

３．２．４　Ｃインタフェース

ＨＩ８－３Ｘ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８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用のアドレス(ポインタ)

<parameter> ：パラメータ

(1) データタイプとサイズ

リスト３－１　に使用するパラメータのデータタイプとそのサイズを示します。

```
typedef char            B;      /*  符号付き　８ﾋﾞｯﾄ 整数                  */
typedef short           H;      /*  符号付き１６ﾋﾞｯﾄ 整数                  */
typedef long            W;      /*  符号付き３２ﾋﾞｯﾄ 整数                  */
typedef char            VB;     /*  ﾃﾞｰﾀﾀｲﾌﾟが一定しないもの( ８ ﾋﾞｯﾄｻｲｽﾞ)  */
typedef unsigned char   UB;     /*  符号無し　８ﾋﾞｯﾄ 整数                  */
typedef unsigned short  UH;     /*  符号無し１６ﾋﾞｯﾄ 整数                  */
typedef unsigned long   UW;     /*  符号無し３２ﾋﾞｯﾄ 整数                  */

typedef void            *VP;    /*  ﾃﾞｰﾀﾀｲﾌﾟが一定しないものへのﾎﾟｲﾝﾀ       */

typedef UB              ID;     /*  ｵﾌﾞｼﾞｪｸﾄのID (xxxid)                   */
typedef B               ER;     /*  ｴﾗｰｺｰﾄﾞ - 一般には符号付きである       */
typedef H               TMO;    /*  ﾀｲﾑｱｳﾄ時間                             */
typedef B               TPRI;   /*  ﾀｽｸ優先度                             */
typedef UB              CCR;    /*  割込みﾏｽｸ                             */

typedef B               INT;    /*  符号付き８ﾋﾞｯﾄ 整数( ﾌﾟﾛｾｯｻ のﾋﾞｯﾄ 幅)*/
typedef UB              UINT;   /*  符号無し８ﾋﾞｯﾄ 整数( ﾌﾟﾛｾｯｻ のﾋﾞｯﾄ 幅)*/
```

リスト３－１　使用するパラメータのデータタイプとサイズ

(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の説明をします。

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:1~31） ····· ②

リターンパラメータ：

ID *p_wtskid ; 待ちタスクID（wtskid:1~31） ····· ③

UINT *p_flgptn ; イベントフラグのビットパターン（flgptn:H'00~FF)···· 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。

関数のリターン値はエラーコードercdです。

② flg_stsの引数flgid は、ID(unsigned char〔符号なし8ビット整数〕）で、タスクIDを設定します。

③ flg_stsの引数p_wtskidは、ID(unsigned char〔符号なし8ビット整数〕）へのポインタを設定します。

④ flg_stsの引数p_flgptnは、UINT(unsigned char〔符号なし8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：

```
R0L     タスクID(tskid:1～31)
```

リターンパラメータ：

```
R0H     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:1～31)
```

リターンパラメータ：

なし

エラーコード(R0H/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_NOEXS	H'CC(-H'34)	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31）、タスクが未登録)
	E_NODMT	H'CA(-H'36)	タスクがDORMANT でない
*	E_CTX	H'BB(-H'45)	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：

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。

sta_tskシステムコールとista_tskシステムコールは、tskidで示されたタスクをDORMANT（休止）状態からREADY （実行可能）状態へ移行します。

sta_tsk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DORMANT（休止）状態にない場合に発行された起動要求に対しては、エラーコードとしてE_NODMTが返ります。

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

Ｈ８－３Ｘは、複数のタスクでスタック領域を共有する機能（共有タスクスタック機能）をもっています。

共有タスクスタック機能を組込んでいる場合、共有スタックを使用するタスクは以下のように制御されます。

共有スタックが解放されているときは、スタック領域を占有し、タスクをREADY(実行可能）状態へ移行します。

共有スタックが占有されているときは、スタック領域を使用できない為、タスクはWAIT（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。

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

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

共有タスクスタック機能については、「２．２．６　共有タスクスタック機能によるタスクの起動」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	R1にエラー情報(H'FFEB)をセットしてシステム異常終了します。

Ｃフォーマット：

```
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	R1にエラー情報(H'FFEB)をセットしてシステム異常終了します。

解　説：

自タスクを正常終了させ、DORMANT(休止）状態へ移行します。

タスクは、終了する前に自分が占有していた資源を自動的に解放することはありませんので、実行中に確保した資源は、タスク終了前に解放させる必要があります。

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、異常終了処理に制御を移行します。異常終了処理については、「２．７　システムの異常終了処理」を参照してください。

Ｈ８－３Ｘは、複数のタスクでスタック領域を共有する機能（共有タスクスタック機能）をもっています。

共有タスクスタック機能を組込んでいる場合、本システムコールを発行したタスクの共有スタック領域を解放します。

その共有スタックの待ち行列にタスクがつながれているならば、そのタスクを共有スタックの待ち行列から外し、READY(実行可能）状態へ移行します。

共有タスクスタック機能については、「２．２．６　共有タスクスタック機能によるタスクの起動」を参照してください。

３．３．３　chg_pri (Change Task Priority)　自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：

```
R1H     優先度(tskpri:-1,0, 初期優先度)
```

リターンパラメータ：

```
R0H     エラーコード(ercd)
```

Ｃフォーマット：

```
ER ercd=chg_pri(TPRI tskpri);
```

パラメータ：

```
TPRI    tskpri; 優先度 (tskpri:-1,0,初期優先度)
```

リターンパラメータ：

なし

エラーコード(R0H/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_TPRI	H'DA(-H'26)	不正タスク優先度（-1,0および初期優先度以外）
*	E_CTX	H'BB(-H'45)	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：

自タスクをシステム最高の優先度（－１）に変更する場合か、または初期優先度に戻す場合に使用します。

システム最高の優先度にする場合は、優先度(tskpri)を－１に、初期優先度に戻す場合には、０または初期優先度の値を指定してください。

優先度が－１の間はタスクの切替えが行なわれないため、割込み処理に影響を与えずにCPU を占有することができます。

タスク優先度は、１～３１のあいだで値の小さい方が高い優先度になります。

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。また、タスクが終了すると、初期優先度に戻ります。

本システムコールで最高優先度に変更した期間は、待ち状態に移行するシステムコール(slp_tsk,wai_tsk,wai_flg,wai_sem,rcv_msg)を発行できません。

3．3．4　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：

```
R0H     エラーコード(ercd)
R0L     タスクID(tskid:1～31)
```

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：1～31)
```

エラーコード(R0H/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_CTX	H'BB(-H'45)	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。

3．3．5　tsk_sts（Get Task Status）タスクの状態を見る〔タスク/ 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：

```
R0L     タスクID(tskid:0,1～31)
```

リターンパラメータ：

```
R0H     エラーコード(ercd)
R1H     現在のタスク優先度(tskpri:-1,1～31)
R1L     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:0,1～31)
```

リターンパラメータ：

```
UINT    *p_tskstat;     タスクステータスパケットの先頭アドレス
TPRI    *p_tskpri;      タスク優先度パケットの先頭アドレス
```

エラーコード(R0H/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_NOEXS	H'CC(-H'34)	そのタスクは生成されていない （tskid> タスク定義数（最大31）、タスクが未登録、 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。

なお、tskid=0 で発行した場合は、自タスク指定になります。

ただし、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。

また、対象タスクがシステム最高の優先度の場合、タスクの現在の優先度(tskpri)として、－1が返ります。

図３－１　に示すように、タスクステータス(tskstat) は８ビット長のフラグで構成されています。

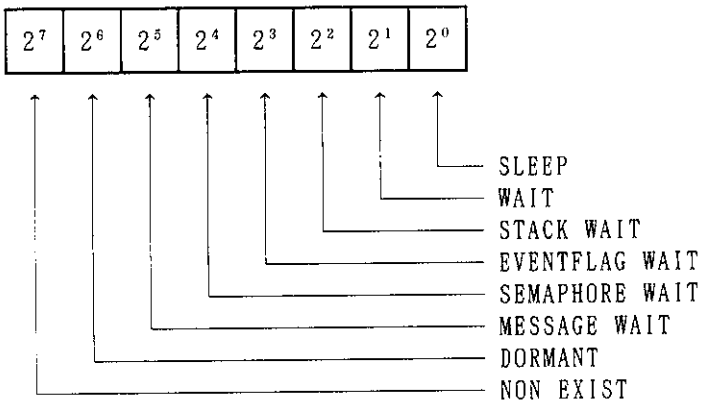


図３－１　タスクステータスフラグ(tskstat) の構成

以下にtskstat のステータスフラグを示します。

ビット0 = ON SLEEP	(slp_tsk によるWAIT（待ち）状態)
ビット1 = ON WAIT	(WAIT(待ち）状態)
ビット2 = ON STACK WAIT	(共有スタックによるWAIT(待ち）状態)
ビット3 = ON EVENT FLAG WAIT	(wai_flg によるWAIT（待ち）状態)
ビット4 = ON SEMAPHORE WAIT	(wai_sem によるWAIT（待ち）状態)
ビット5 = ON MESSAGE WAIT	(rcv_msg によるWAIT（待ち）状態)
ビット6 = ON DORMANT	(DORMANT（休止）状態)
ビット7 = ON NON EXIST	(タスクが登録されていない状態)

READY(実行可能）状態または、RUN(実行）状態の場合は、tskstatは０です。

wai_tsk(tmout=-1) によるWAIT（待ち）状態の場合はslp_tskと同じ動作を行ないますので、ビット０のみＯＮです。

３．４　タスク付属・同期システムコール

３．４．１　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：

ROH　　エラーコード(ercd)

Ｃフォーマット：

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

パラメータ：

なし

リターンパラメータ：

なし

エラーコード(ROH/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_OBJ	H'C1(-H'3F)	オブジェクトエラー （タスク優先度がシステム最高(tskpri=-1）のとき発行できない）
*	E_CTX	H'BB(-H'45)	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：

自タスクをRUN(実行）状態からWAIT(待ち）状態に移行します。

このWAIT(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既に出されていれば、WAIT(待ち）状態とならず、そのまま実行を続けます。

3．4．2　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：

R1　　タイムアウト指定(tmout:H'FFFF,H'1～H'7FFF)

リターンパラメータ：

ROH　　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:H'FFFF,H'1～H'7FFF)

リターンパラメータ：

なし

エラーコード(ROH/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_TNOSPT	H'ED(-H'13)	タイマがサポートされていない
*	E_ILTIME	H'D9(-H'27)	不正時間指定(tmout≦-2)
*	E_OBJ	H'C1(-H'3F)	オブジェクトエラー（タスク優先度がシステム最高(tskpri=-1) のとき発行できない）
*	E_CTX	H'BB(-H'45)	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'AB(-H'55)	タイムアウト

【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RUN(実行) 状態からWAIT(待ち) 状態に移行します。

このWAIT(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ます。

wup_tsk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=-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。

tmout のビット数は16ビットです。

システムのタイマ割込み周期が10msecの場合、約327 秒の制御を行なうことができます。

３．４．３　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：

```
ROL     タスクID(tskid:1～31)
```

リターンパラメータ：

```
ROH     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:1～31)
```

リターンパラメータ：

なし

エラーコード(ROH/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_SELF	H'CF(-H'31)	自タスク指定（tskid=自タスクID）《wup_tskのみ》
*	E_NOEXS	H'CC(-H'34)	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31) 、タスクが未登録)
	E_DMT	H'CB(-H'35)	タスクがDORMANT である。
*	E_CTX	H'BB(-H'45)	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'B7(-H'49)	キューイングオーバフロー(wupcnt>1)

【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：

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。

wup_tskシステムコールとiwup_tskシステムコールは、slp_tskまたはwai_tskシステムコールの発行によりWAIT(待ち) 状態になっているタスクを、READY(実行可能) 状態に移行させます。

対象タスクはtskid で示され、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。

対象タスクがWAIT(待ち) 状態でない場合、この起床要求は１回だけキューイングされます。後に対象タスクがslp_tskか、またはwai_tskシステムコールを発行したときに有効になります。

キューイングの最大値は１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。

Ｈ８－３Ｘは、複数のタスクでタスクスタック領域を共有する機能をもっています。
共有スタック待ち状態のタスクに対して、本システムコールを発行すると、エラーコードとしてE_OKまたはE_QOVRを返します。

3．4．4　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：

```
R0L     タスクID (tskid:0,1～31)
```

リターンパラメータ：

```
R0H     エラーコード(ercd)
R1H     キューイングされていた起床要求回数(wupcnt:0,1)
```

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:0,1～31)
```

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:0,1)
```

エラーコード(R0H/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_NOEXS	H'CC(-H'34)	そのタスクは生成されていない （tskid> タスク定義数（最大31）、タスクが未登録）
	E_DMT	H'CB(-H'35)	タスクがDORMANT である
*	E_CTX	H'BB(-H'45)	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(wupcnt:1)をリターンパラメータとして返し、その起床要求を解除します。

本システムコールが発行された時点で対象タスクのキューイングされている起床要求回数がすでに0である場合は、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。

ＨＩ８－３Ｘは、複数のタスクでタスクスタック領域を共有する機能をもっています。共有スタック待ち状態のタスクに対して、本システムコールを発行すると、エラーコードとしてE_OKを返します。

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：

```
R0L     イベントフラグID(flgid:1～31)
R1H     セットするビットパターン(setptn:H'00～H'FF)
```

リターンパラメータ：

```
R0H     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:1～31)
UINT    setptn; セットするビットパターン(setptn:H'00～H'FF)
```

リターンパラメータ：

なし

エラーコード(R0H/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_NOEXS	H'CC(-H'34)	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31))
*	E_CTX	H'BB(-H'45)	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：

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。

flgidで示されるイベントフラグ(８ビット）のうち、対応するsetptnのＯＮ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READY(実行可能）状態へ移行します。

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〔タスク部用〕

アセンブラフォーマット：

```
JSR     @clr_flg
```

パラメータ：

```
R0L     イベントフラグID(flgid:1～31)
R1H     クリアするビットパターン(clrptn:H'00～H'FF)
```

リターンパラメータ：

```
R0H     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:1～31)
UINT    clrptn; クリアするビットパターン(clrptn:H'00～H'FF)
```

リターンパラメータ：

なし

エラーコード(R0H/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_NOEXS	H'CC(-H'34)	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31))
*	E_CTX	H'BB(-H'45)	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：

flgid で示されるイベントフラグ（8ビット）のうち、対応するclrptnの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。

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。

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。

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：

```
R0L     イベントフラグID(flgid:1～31)
R1H     待ちビットパターン(waiptn:H'01～H'FF)
R1L     待ちモード(wfmode:0～3)
```

リターンパラメータ：

```
R0H     エラーコード(ercd)
R1H     セットされた後のイベントフラグの内容(flgptn:H'01～H'FF)
```

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:1～31)
UINT    waiptn ;  待ちビットパターン(waiptn:H'01～H'FF)
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'01～H'FF)
```

エラーコード(R0H/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_PAR	H'DF(-H'21)	パラメータエラー(waiptn=0 または、wfmode>3)
*	E_NOEXS	H'CC(-H'34)	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31))
*	E_OBJ	H'C1(-H'3F)	オブジェクトエラー　《wai_flgのみ》 (タスク優先度がシステム最高(tskpri=-1) のとき発行できない)
*	E_CTX	H'BB(-H'45)	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'B7(-H'49)	キューイングのオーバーフロー (既に待ちタスクが存在している) 《wai_flgのみ》
	E_PLFAIL	H'A7(-H'59)	ポーリング失敗　《pol_flg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。

一つのイベントフラグに、事象の発生を待つことができるタスクは一つしか許されません。

pol_flg システムコールは、flgid で示される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）がセットされていれば、本システムコールの発行タスクは正常終了しますが、セットされていなければ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、待ち条件を満足したときのイベントフラグの内容(flgptn)が返されます。

待ちモードにTWF_CLRを指定した場合は、クリア前の待ち条件を満足したときのイベントフラグの内容(flgptn)が返されます。またエラーリターンした場合、イベントフラグはクリアされません。

表３－２　に待ちモードの詳細を示します。

表３－２　待ちモード(wfmode)のコードと意味

項番	待ちモード	コード	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（８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（８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／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：

```
        R0L     イベントフラグID(flgid:1～31)
```

リターンパラメータ：

```
        R0H     エラーコード(ercd)
        R1H     イベントフラグのビットパターン(flgptn:H'00～H'FF)
        R0L     待ちタスクID(wtskid:0～31)
```

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:1～31)
```

リターンパラメータ：

```
        ID      *p_wtskid;      待ちタスクID(wtskid:0～31)
        UINT    *p_flgptn;      イベントフラグのビットパターン(flgptn:H'00～H'FF)
```

エラーコード(R0H/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_NOEXS	H'CC(-H'34)	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31))

【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各種の状態を参照し、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'00) 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：

```
ROL     セマフォID(semid:1～31)
```

リターンパラメータ：

```
ROH     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:1～31)
```

リターンパラメータ：

なし

エラーコード(ROH/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_NOEXS	H'CC(-H'34)	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31))
*	E_CTX	H'BB(-H'45)	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'B7(-H'49)	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：

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。

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増やします。そのセマフォの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、READY(実行可能）状態へ移行します。

なお、セマフォのカウント値の最大値は255 です。本システムコールを実行したときにこの値を越える場合はE_QOVRでエラーリターンします。

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：

ROL　セマフォID(semid:1～31)

リターンパラメータ：

ROH　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:1～31)

リターンパラメータ：

なし

エラーコード(ROH/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_NOEXS	H'CC(-H'34)	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31))
*	E_OBJ	H'C1(-H'3F)	オブジェクトエラー　《wai_semのみ》 (タスク優先度がシステム最高(tskpri=-1) のとき発行できない)
*	E_CTX	H'BB(-H'45)	コンテキストエラー (非タスク部から発行できない)
	E_PLFAIL	H'A7(-H'59)	ポーリング失敗　《preq_sem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が１以上であれば、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は実行を継続し、セマフォのカウント値が０の場合は、セマフォのカウント値を変更せず、本システムコールを発行したタスクはWAIT（待ち）状態となり、そのセマフォの待ち行列につながれます。

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

preq_semシステムコールは、semidで示されたセマフォのカウント値が１以上であれば、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は正常終了し、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そのままにして、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／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：

```
        R0L     セマフォID(semid:1～31)
```

リターンパラメータ：

```
        R0H     エラーコード(ercd)
        R1H     セマフォカウント値(semcnt:0～255)
        R0L     待ちタスクID(wtskid:0～31)
```

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:1～31)
```

リターンパラメータ：

```
        ID      *p_wtskid;      待ちタスクID(wtskid:0～31)
        UINT    *p_semcnt;      セマフォカウント値(semcnt:0～255)
```

エラーコード(R0H/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_NOEXS	H'CC(-H'34)	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31))

【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各種の状態を参照し、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'00) 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：

```
ROL     メイルボックスID(mbxid:1～31)
R1      メッセージの先頭アドレス(pk_msg)
```

リターンパラメータ：

```
ROH     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:1～31)
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(ROH/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_ILADR	H'DE(-H'22)	不正アドレス(メッセージアドレスが０または奇数)
	E_ILMSG	H'D7(-H'29)	不正メッセージ形式 (メッセージ先頭２バイトが０でない)
*	E_NOEXS	H'CC(-H'34)	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31))
*	E_CTX	H'BB(-H'45)	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：

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。

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。

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側で管理する必要があります。

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キューと呼ぶ待ち行列につながれます。

つながれ方はFIFO(First-In First-Out)です。

メッセージを待つタスクがないときにsnd_msgが発行されても、発行タスクはWAIT(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その先の実行を続けます。

ユーザはメッセージ領域を、必ずRAM 上に確保してください。

ユーザがメッセージとして利用できる領域は、メッセージの先頭アドレス＋２バイト目からです。

メッセージの先頭２バイトはＯＳで使用するため、初期値として必ず０を設定し、送信後も書き替えないでください。書き替えた場合は、動作は保障されません。

なお、０が設定されていない場合は、エラーコードにE_ILMSG を返します。

図３－２　に送信メッセージの形式を示します。

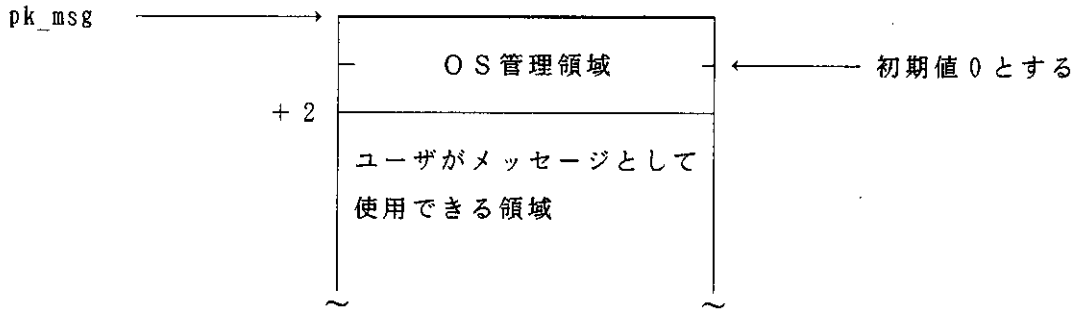


図３－２　送信メッセージの形式

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：

```
R0L     メイルボックスID(mbxid：1～31)
```

リターンパラメータ：

```
R0H     エラーコード(ercd)
R1      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:1～31)
```

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(R0H/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_NOEXS	H'CC(-H'34)	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31))
*	E_OBJ	H'C1(-H'3F)	オブジェクトエラー　《rcv_msg のみ》 (タスク優先度がシステム最高(tskpri=-1) のとき発行できない)
*	E_CTX	H'BB(-H'45)	コンテキストエラー(非タスク部から発行できない)
	E_PLFAIL	H'A7(-H'59)	ポーリング失敗　《prcv_msg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：

rcv_msgシステムコールは、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。

まだそのメイルボックスにメッセージが到着していない場合には、本システムコール発行タスクはメッセージ到着を待つ、待ち行列につながれます。

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

prcv_msgシステムコールは、mbxidで示されたメイルボックスにメッセージが入っていれば、そのメッセージを受信し、受信したメッセージの先頭アドレスをリターンパラメータとして返し、正常終了します。

また、そのメイルボックスにメッセージが到着していない場合には、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)

ユーザがメッセージとして利用できる領域は、メッセージの先頭アドレス＋２バイト目からです。メッセージの先頭２バイトはＯＳで使用するため、この部分は書き替えないでください。書き替えた場合、再度メッセージを送信しようとしてもその処理は保障されません。

図３－３　に受信メッセージの形式を示します。

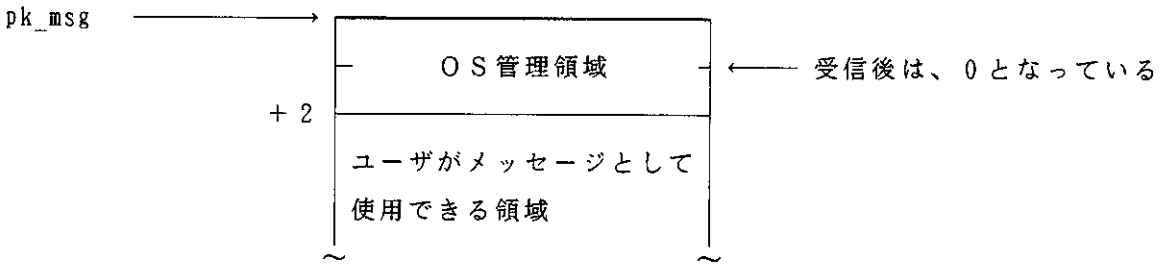


図３－３　受信メッセージの形式

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／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：

```
        R0L     メイルボックスID(mbxid:1～31)
```

リターンパラメータ：

```
        R0H     エラーコード (ercd)
        R1      次のメッセージの先頭アドレス(pk_msg)
        R0L     待ちタスクID(wtskid:0～31)
```

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:1～31)
```

リターンパラメータ：

```
        ID      *p_wtskid;          待ちタスクID(wtskid:0～31)
        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;          次のメッセージの先頭アドレスへのポインタ
```

エラーコード(R0H/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_NOEXS	H'CC(-H'34)	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31))

【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各種の状態を参照し、対象メイルボックスの次に受信されるメッセージの先頭アドレス(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。

次のメッセージの先頭アドレス(pk_msg)には、次に受信されるメッセージの先頭アドレスが返されます。次に受信されるメッセージがない場合は、NADR(H'FFFF)が返されます。

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が返されます。待ちタスクがない場合は、NOTSK(H'00) が返されます。

３．６　割込み管理システムコール

３．６．１　ret_int（Return From Interrupt Handler）割込み処理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	R1にエラー情報(H'FFBB)をセットしてシステム異常終了します。

Ｃフォーマット：

なし

解　説：

割込み処理ハンドラから復帰するときに発行するシステムコールです。

割込み処理ハンドラの中でシステムコールを発行してもディスパッチは起こらず、このシステムコールが発行されて割込み処理ハンドラを抜けるまで、タスクのディスパッチが遅延されます。

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、割込み処理ハンドラが起動されたときの状態と同じでなければなりません。

タスク部（割込みマスクを解除した状態）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。異常終了処理については、「２．７　システムの異常終了処理」を参照してください。

3．6．2　chg_ims（Change Interrupt Mask Level）割込みマスクを変更する〔タスク部用〕

アセンブラフォーマット：

```
JSR     @chg_ims
```

パラメータ：

```
R1H     割込みマスク(imask)
```

リターンパラメータ：

```
R0H     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
```

リターンパラメータ：

```
なし
```

エラーコード(R0H/ercd)：

```
E_OK  H'00 (H'00)    正常終了
```

解　説：

本システムコールは、ＣＰＵのコンディションコードレジスタ(CCR) の割込みマスクビット(I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。

割込みを禁止したい場合はR1H に０を、解除したい場合は０以外の値を設定してください。

図３－４　にコンディションコードレジスタ(CCR) の構成を示します。

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
CCR	I	U	H	U	N	Z	V	C

I　　:割込みマスクビット
U　　:ユーザビット(2^6 ・2^4 bit) ＯＳが使用します
H　　:ハーフキャリビット
N　　:ネガティブフラグ
Z　　:ゼロフラグ
V　　:オーバフローフラグ
C　　:キャリフラグ

図３－４　コンディションコードレジスタ(CCR) の構成

【注】　割込みのマスクについては、「２．２．９　タスク実行中の割込みのマスク」を参照してください。

３．６．３　ims_sts (Get Interrupt Mask Level Status) 割込みマスクを参照する

〔タスク／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：

```
R0H     エラーコード(ercd)
R1H     割込みマスク(imask:0,1)
```

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:0,1)
```

エラーコード(R0H/ercd)：

E_OK	H'00(H'00)	正常終了

解　説：

ＣＰＵのコンディションコードレジスタ(CCR) の割込みマスクビット(Iビット) を参照し、割込み禁止状態をレジスタR1H に反映させます。

割込みマスク(imask) は、禁止されているときは１、解除されているときは０になります。

コンディションコードレジスタ(CCR) の構成については、「図３－４　コンディションコードレジスタ(CCR) の構成」を参照してください。

3．7　時間管理システムコール

3．7．1　set_tim (Set Time)システムクロックを設定する〔タスク／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：

```
R1      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)  }
        現在の時間データ〈下位〉(ltime)  } 32 ビット(H'0～H'7FFFFFFF)
```

リターンパラメータ：

```
R0H     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  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(R0H/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_TNOSPT	H'ED(-H'13)	タイマがサポートされていない
*	E_ILADR	H'DE(-H'22)	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'D9(-H'27)	不正時間指定(utime,ltime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。

pk_timeで示される値が0以下である場合、エラーコードとしてE_ILTIMEを返します。

システムクロックのビット数は32ビットです。

3．7．2　get_tim (Get Time)　システムクロックの値を読み出す〔タスク／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：

```
        R1      データ読み出し領域(4バイト) へのポインタ(pk_time)
```

リターンパラメータ：

```
        R0H     エラーコード(ercd)
        R1      データ読み出し領域(4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)  }
                現在の時間データ <下位> (ltime)  }  32 ビット(H'0～H'FFFFFFFF)
```

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  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(R0H/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_TNOSPT	H'ED(-H'13)	タイマがサポートされていない
*	E_ILADR	H'DE(-H'22)	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32ビットです。

3．8　システム管理システムコール

3．8．1　get_ver (Get Version No)ニュークリアスのバージョン識別子を得る

〔タスク／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：

```
R1      バージョン管理ブロック読み出し領域（２０バイト）へのポインタ(pk_ver)
```

リターンパラメータ：

```
ROH     エラーコード
R1      バージョン管理ブロック読み出し領域（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])
        製品管理情報            (prno[1])
        製品管理情報            (prno[2])
        製品管理情報            (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(ver)
} T_VER *pk_ver;        バージョン管理ブロック読出し領域へのポインタ
```

エラーコード(ROH/ercd)：

	E_OK	H'00(H'00)	正常終了
*	E_ILADR	H'DE(-H'22)	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。

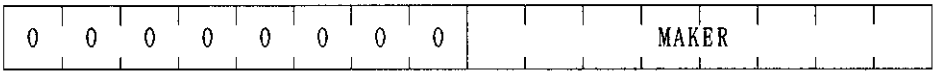
解　説：

本システムコールは、現在使用中のＨＩ８－３Ｘ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。

なお、本システムコールのみ、最大割込み禁止時間は24μｓとなりますので、注意してください。

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)

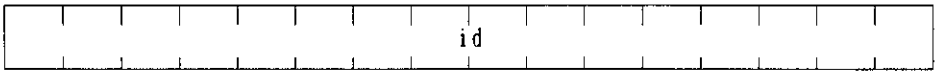


MAKER ： この製品を作ったメーカー

ＨＩＴＡＣＨＩ ： H'000A

ＨＩ８－３ＸのMAKER の値はH'000Aです。

(id)



id ： ＯＳやＶＬＳＩの種類を区別する番号

ＨＩ８－３Ｘのidの値はH'0000です。

(spver)

MAGIC ： ＴＲＯＮ仕様のシリーズを区別する番号

μＩＴＲＯＮ仕様 ： H'5

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。

Ｖｅｒ　２．０２ ： H'202

ＨＩ８－３Ｘのspver の値はH'5202です。

(prver)

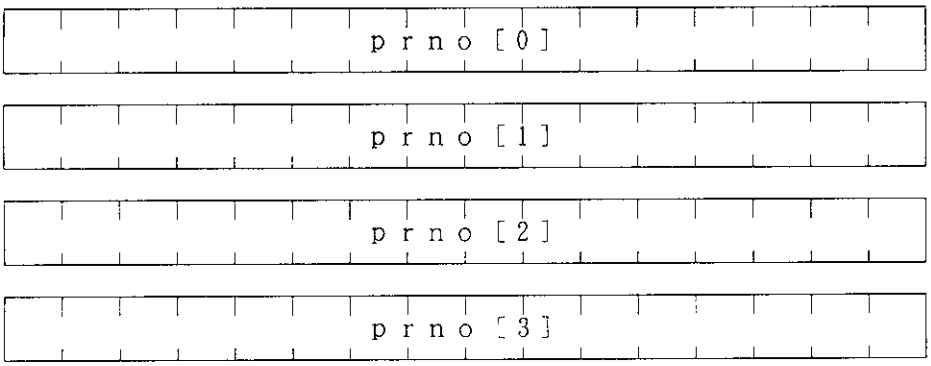
内部インプリメント上のバージョン番号を表わします。

MAJOR : バージョン番号の大きな区別を表わします

MINOR : バージョン番号の小さな区別を表わします

ＨＩ８－３Ｘのprver の値はH'0110です。

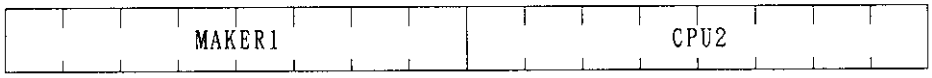
(prno)



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

ＨＩ８－３Ｘ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。

ＨＩＴＡＣＨＩ : H'0A

CPU2 : このμＩＴＲＯＮを実行するプロセッサを表わします。

ＨＩ８－３ＸのCPU2の値はH'83とします。

ＨＩ８－３Ｘのcpu の値はH'0A83です。

(var)

－	LEV		－	M	V	P	－	－	FIL		IO		－	－

このμＩＴＲＯＮで利用できる機能の概要を表わします。

－ ：reserved(0が返る)

LEV：ニュークリアス仕様のレベル分け

　　μＩＴＲＯＮ：000

M：プロセッサ

　　シングルプロセッサ用：0

V：仮想記憶

　　仮想記憶未対応：0

P：ＭＭＵ

　　ＭＭＵ未対応：0

FIL：ファイル仕様のレベル分け

　　サポートなし：00

IO：入出力仕様

　　サポートなし：00

Ｈ８－３Ｘのver の値はH'0000です。

4．システムの構築

4．1　概　要

ＨＩ８－３Ｘを使用したシステムの構築には、次の作業が必要になります。

(1) ユーザ・アプリケーションプログラムの作成

タスク・割込み処理ハンドラ・ユーザリセットルーチンなどを作成し、オブジェクトファイルを生成します。

(2) システム・アプリケーションプログラムの作成

システムに必要なタイマ初期化ルーチン・タイマ割込み処理ハンドラを作成し、オブジェクトファイルを生成します。

(3) ＨＩ８－３Ｘのセットアップファイルの作成

ＨＩ８－３Ｘが必要とする作業領域、およびシステムで必要な資源を定義するセットアップテーブルを作成し、オブジェクトファイルを生成します。

(4) 割込みベクタテーブルの作成

割込み処理ハンドラの先頭アドレスを登録した割込みベクタテーブルを作成し、オブジェクトファイルを生成します。

(5) システムの結合

(1) ～(4) まで、生成されたオブジェクトファイルを結合し、アブソリュートプログラムを作成します。

一度に結合できない場合、中間ファイル：リロケータブルファイルを通して、システムを結合します。

図４－１　にシステムの構築手順の概要を示します。

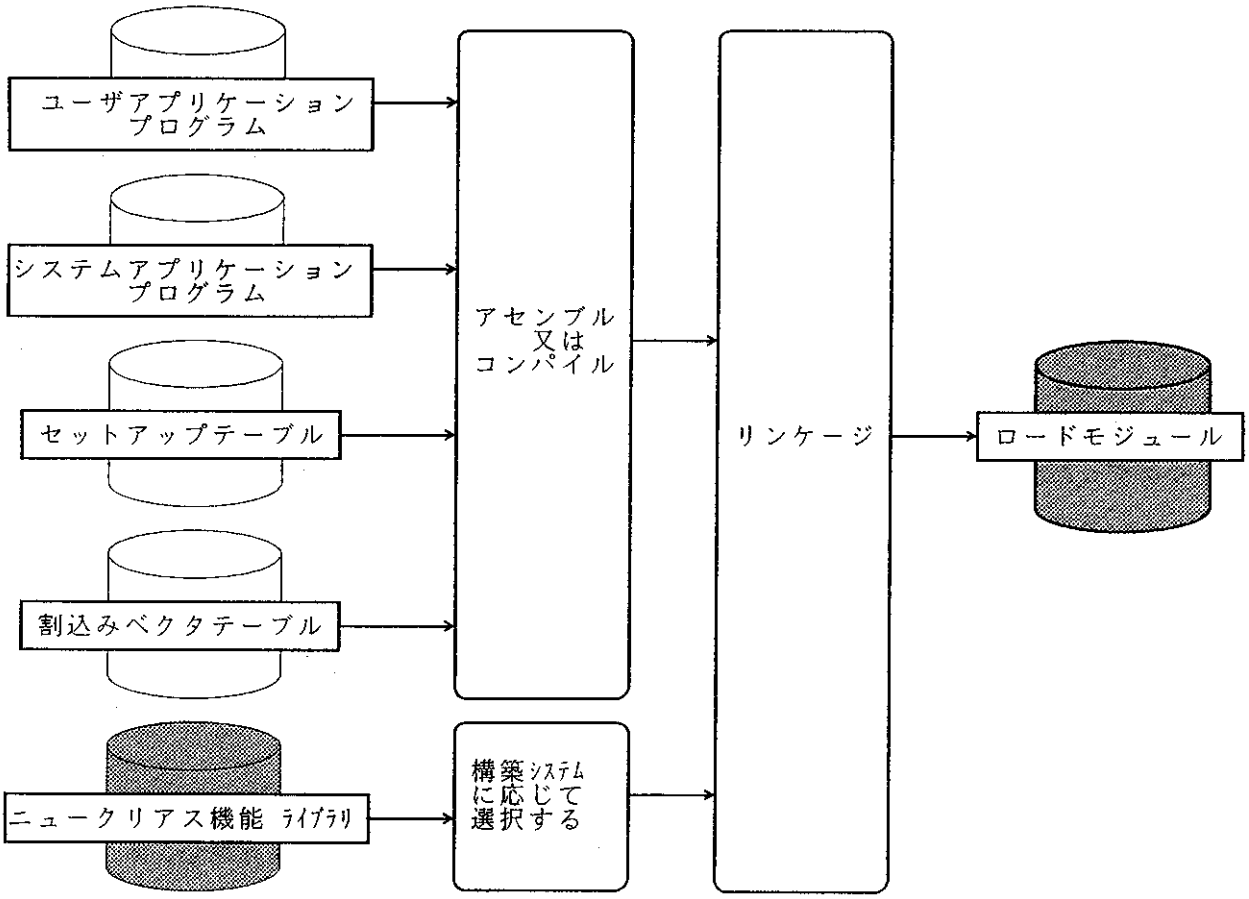


図４－１　システムの構築手順の概要

４．２　ユーザ・アプリケーションプログラムの作成

ユーザのシステムに必要となるアプリケーションプログラムを、アセンブリ言語またはＣ言語により作成します。

作成するプログラムには以下のものがあります。

・タスク
・割込み処理ハンドラ
・ユーザリセットルーチン

４．２．１　タスクの作成

構築するシステムに応じて、タスクを作成します。

Ｈ８－３Ｘは、タスクの登録数が３１までに限られ、タスク優先度はタスクIDと同一ですので、これらを考慮して作成してください。

詳しい内容は、「２．２　タスク管理、タスク付属同期管理」を参照してください。

作成したタスクを登録するには、Ｈ１８－３Ｘのセットアップファイルのタスク定義テーブル『H$TSK_ADR』にタスクの先頭アドレスを定義します。

詳しいタスクの登録方法は、「４．４．２　タスク定義テーブルの作成」を参照してください。

４．２．２　割込み処理ハンドラの作成

割込み処理ハンドラとは、ニュークリアスの介入なしに制御が移される処理プログラムです。各種割込みに対応した割込み処理ハンドラを、構築するシステムに応じて作成します。

詳しい内容は、「２．５　割込み管理」を参照してください。

作成した割込み処理ハンドラを登録するには、割込みベクタテーブルの該当ベクタ番号に割込み処理ハンドラの先頭アドレスを定義します。

詳しい割込み処理ハンドラの登録方法は、「４．５．１　割込みベクタテーブルの作成」を参照してください。

４．２．３　ユーザリセットルーチンの作成

ユーザリセットルーチンとは、Ｈ１８－３Ｘ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。ユーザリセットルーチンは、必要に応じて作成します。

詳しい内容は、「２．６　システム起動処理」を参照してください。

作成したユーザリセットルーチンを登録するには、ユーザリセットルーチンアドレス『H$USR_RST』にユーザリセットルーチンの先頭アドレスを定義します。

詳しいユーザリセットルーチンの登録方法は、「４．４．１　システム定義テーブルの作成」を参照してください。

４．２．４　ユーザ・アプリケーションプログラムのアセンブル

ユーザのシステムに必要となるアプリケーションプログラムのソースファイルをアセンブルまたはコンパイルすることによりオブジェクトファイルを生成します。

このオブジェクトファイルは、システムの結合時に他のオブジェクトファイルやライブラリファイルと結合するときの入力ファイルになります。

ユーザ・アプリケーションプログラムをアセンブルするには次のコマンドを入力します。

% asm38 △＜ユーザ・アプリケーションプログラムファイル名＞△-cpu＝300(RET)

ユーザ・アプリケーションプログラムをコンパイルするには次のコマンドを入力します。

% ch38△-cpu＝300reg△＜ユーザ・アプリケーションプログラムファイル名＞(RET)

この操作で、拡張子『.obj』のシステム・アプリケーションプログラムのオブジェクトファイルが生成されます。

Ｈ８／３００のアセンブラの詳しい説明は、「Ｈ８／３００シリーズ　クロスアセンブラ　ユーザーズマニュアル」を参照してください。

Ｈ８／３００のＣコンパイラの詳しい説明は、「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。

４．３　システム・アプリケーションプログラムの作成

システム・アプリケーションプログラムとは、システムを構築するときにＨＩ８－３Ｘとして必要なプログラムをいいます。

４．３．１　タイマ初期設定ルーチン・タイマ割込みリセット処理の作成

ＨＩ８－３Ｘは、ハードウェアからの一定周期の割込みを利用して、時間管理(wai_tsk・set_tim ・get_tim)を行なっています。

(1) タイマ初期設定ルーチン

タイマ初期設定ルーチンは、使用するハードウェアのタイマの初期化を行ないます。

(2) タイマ割込みリセット処理

タイマ割込みリセット処理は、タイマの割込み発生後、タイマ割込みをクリアするための処理で、タイマの割込みをクリアする必要がある場合に作成するプログラムです。

詳しい内容は、「２．４．４　タイマ処理ハンドラ（タイマ初期設定ルーチン・タイマ割込み処理ハンドラ）」を参照してください。

(3) タイマ周期の変更

標準提供のタイマ初期設定ルーチン・タイマ割込みリセット処理は、ハードウェアタイマとしてＨ８／３２５のフリーランニングタイマ(以下、ＦＲＴと略す）を利用しています。

また、標準提供のタイマの仕様として、ＦＲＴのアウトプットコンペアレジスタＡ(ＯＣＲＡ)

を利用して、タイマ周期を１０m(sec) に指定しています。

このタイマ周期を容易に変更できるように、インクルードファイル『h3xtim25.i』の中で、ｅｑｕａｔｅ定義しています。以下に、その部分を説明します。

```
CLKDEVNO        .equ    0               ;: clock devide no. ····①
OCRADATA        .equ    h'c34f          ;: ocra set data····· ②
```

①クロック分周比コード(０～２)
３種類の内部クロック(φ／２＝０, φ／８＝１, φ／32＝２) を指定します。
②ＯＣＲＡ設定データ(０～h' ｆｆｆｆ)
アウトプットコンペアレジスタＡの設定データを指定します。

タイマ周期は、CPU クロックとクロック分周比とＯＣＲＡの値で決定します。
タイマ周期は、以下の式で算出します。

・タイマ周期　：x(sec)
・クロック分周比(CPUクロック／クロックタイマ) ：n
・ＯＣＲＡ設定データ＝x(CPU クロック／n)－１

(例：標準提供の値)

CPU クロック　＝10M (Hz)
タイマ周期　＝10m(sec)
クロック分周比＝２
OCRA設定データ＝0.01(10,000,000/2)-1
＝49,999
＝H'c34f

CPUクロックが10MHz の場合, タイマ周期は以下の範囲まで設定できます。

クロック分周比：２＝ 0.2μ(sec) ～ 13.1072 m(sec)
クロック分周比：８＝ 0.8μ(sec) ～ 52.4288 m(sec)
クロック分周比：32＝ 3.2μ(sec) ～209.7152 m(sec)

フリーランニングタイマの詳しい説明は、「Ｈ８／３２５シリーズ　ハードウェアマニュアル」を参照してください。

(4) 他のＨ８／３００シリーズを使用する場合の注意

Ｈ８／３２５のフリーランニングタイマ以外のタイマを利用する場合、タイマ初期設定ルーチン・タイマ割込みリセット処理を修正・登録する必要があります。

標準提供のタイマ初期設定ルーチン・タイマ割込みリセット処理（『h3xtim25.mar』，『h3xtim25.i』）の、ハードウェアのタイマ周りのみを注意して変更してください。タイマ割込みハンドラモジュールファイル『h3xtim.inc』は、システムのタイマ処理ですので変更しないでください。

なお、Ｈ８／３３０のフリーランニングタイマのタイマ初期設定ルーチン・タイマ割込みリセット処理（『h3xtim30.mar』，『h3xtim30.i』）も提供しています。

４．３．２　システム・アプリケーションプログラムのアセンブル

構築するシステムに必要なシステム・アプリケーションプログラムのソースファイルをアセンブルします。

システム・アプリケーションプログラムのソースファイルをアセンブルすることにより、システムと結合可能なオブジェクトファイルが生成されます。

標準提供のタイマ初期設定ルーチン、タイマ割込みリセット処理は、ニュークリアスのライブラリに組込まれています。よって、タイマ周期を変更しない場合は、タイマ初期設定ルーチン・タイマ割込みリセット処理をアセンブルする必要はありません。

タイマの周期を変更した場合は、タイマ初期設定ルーチン・タイマ割込みリセット処理をアセンブルしてください。

システム・アプリケーションプログラムをアセンブルするには次のコマンドを入力します。

% asm38 △＜システム・アプリケーションプログラムファイル名＞△-cpu＝300(RET)

この操作で、拡張子『.obj』のシステム・アプリケーションプログラムのオブジェクトファイルが生成されます。

Ｈ８／３００のアセンブラの詳しい説明は、「Ｈ８／３００シリーズ　クロスアセンブラ　ユーザーズマニュアル」を参照してください。

４．４　Ｈ８－３Ｘのセットアップファイルの作成

システムを構築する場合に必要な情報を定義するため、セットアップファイルを作成します。

Ｈ８－３Ｘのセットアップファイル（ファイル名『h3xsetup.src』）には、以下に示す各機能の定義テーブルがあります。

(1) システム定義テーブル

システムが使用するルーチンの先頭アドレスを登録します。

登録する情報を以下に示します。

- H$TIM_INI(タイマ初期設定ルーチンアドレス)
- H$USR_RST(ユーザリセットルーチンアドレス)

(2) タスク定義テーブル

タスクを使用するのに必要な情報を登録します。

登録する情報を以下に示します。

- H$TSK_ADR(タスク先頭アドレス定義テーブル)
- H$TSK_STK(タスクスタックポインタ定義テーブル)
- t_stk　(タスク用スタック領域)
- H$TSK_CNT(タスク定義数)
- H$TCB　(Task Control Block：タスク管理用の制御領域)
- H$TSKSPTBL(タスクスタックポインタセーブ領域)

(3) イベントフラグ定義テーブル

イベントフラグを使用するのに必要な情報を登録します。

登録する情報を以下に示します。

- H$FLG_CNT(イベントフラグ定義数)
- H$FLGCB　(eventFLaG Control Block ：イベントフラグ管理用の制御領域)

(4) セマフォ定義テーブル

セマフォを使用するのに必要な情報を登録します。

登録する情報を以下に示します。

- H$SEM_CNT(セマフォ定義数)
- H$SEMCB　(SEMaphore Control Block ：セマフォ管理用の制御領域)

(5) メイルボックス定義テーブル

メイルボックスを使用するのに必要な情報を登録します。

登録する情報を以下に示します。

- H$MBX_CNT(メイルボックス定義数)
- H$MBXCB　(MailBoX Control Block ：メイルボックス管理用の制御領域)

(6) ＨＩ８－３ＸのＲＡＭ領域の定義およびシステム管理テーブル

ＨＩ８－３Ｘの作業領域として、以下のものがあります。

・H$OS_SP （ＯＳ用スタックポインタ・領域）

・H$TM_SP （タイマ用スタックポインタ・領域、スタックポインタセーブ領域）

・システム管理テーブル（HCUR_SP, HTOP_PRI_ID, H$RDYTBL,
HSYS_CLK, HTIM_CNT, H$TIMCB, H$TIMCB_BTM）

４．４．１　システム定義テーブルの作成

wai_tsk 、set_tim 、get_tim システムコールを使用するには、タイマ割込み処理ハンドラを登録しなければなりません。この場合、タイマ初期設定ルーチンをセットアップファイルのシステム定義テーブルに登録してください。

また、ユーザプログラムの初期処理を行ないたい場合は、ユーザリセットルーチンをセットアップファイルのシステム定義テーブルに登録してください。

(1) H$TIM_INI(タイマ初期設定ルーチンアドレス)

タイマ初期設定ルーチンを登録するには、タイマ初期設定ルーチンアドレス『H$TIM_INI*1』に、タイマ初期設定ルーチンの先頭アドレスを設定してください。登録をしない場合は、０を設定してください。

タイマ初期設定ルーチンの詳しい内容は、「２．４．５　タイマ処理ハンドラの登録」を参照してください。

(2) H$USR_RST(ユーザリセットルーチンアドレス)

ユーザリセットルーチンを登録するには、ユーザリセットルーチンアドレス『H$USR_RST*1』に、ユーザリセットルーチンの先頭アドレスを設定してください。登録をしない場合は、０を設定してください。

ユーザリセットルーチンの詳しい内容は、「２．６．２　ユーザリセットルーチン」を参照してください。

【注】 *1 このラベルの領域を奇数アドレスから確保した場合、システム起動時にシステム異常終了します。

図４－２　にシステム定義テーブルのフォーマットを示します。

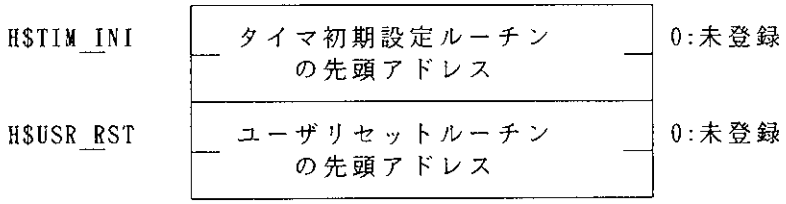


図４－２　システム定義テーブル

図４－３　タイマ初期設定ルーチンの登録を以下に示します。

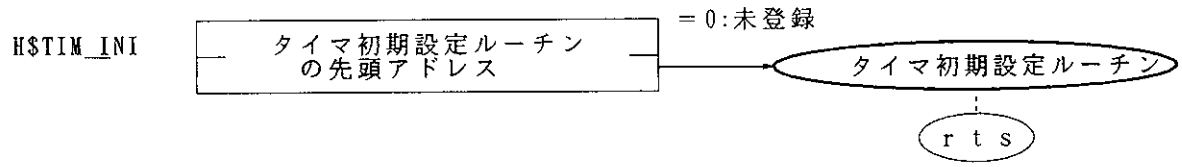


図４－３　タイマ初期設定ルーチンの登録

図４－４　ユーザリセットルーチンの登録を以下に示します。

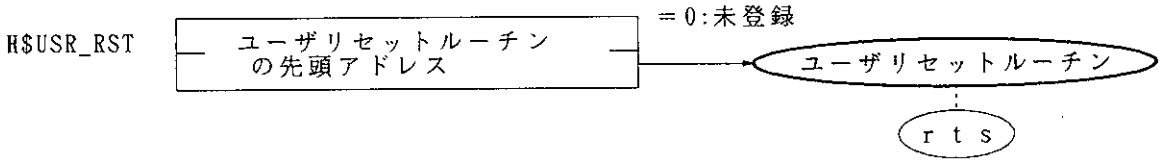


図４－４　ユーザリセットルーチンの登録

４．４．２　タスク定義テーブルの作成

タスクを利用するためには、タスク情報をセットアップファイルのタスク定義テーブルに登録しなければなりません。

タスク情報には、以下のものがあります。

(1) H$TSK_ADR+2(tsk_adr:タスク先頭アドレス定義テーブル)

タスク先頭アドレス定義テーブルには、タスクID＝１から順にタスクの先頭アドレスを設定します。

該当タスクIDのタスクを登録しない場合は、０を設定してください。ただし、ＴＣＢ領域（タスク管理用の制御領域）は確保されます。

また、タスク初期状態フラグとして、それぞれのタスクの初期状態をDORMANT(休止）状態にするかまたはREADY(実行可能）状態にするかを設定できます。

該当タスクIDのタスク先頭アドレス定義テーブルを、タスク先頭アドレスのまま(2^0 ﾋﾞｯﾄ ＝０）にすれば、タスクはREADY(実行可能）状態で登録されます。タスク先頭アドレス＋１(2^0 ﾋﾞｯﾄ ＝１）にすれば、タスクはDORMANT(休止）状態で登録されます。

(2) H$TSK_STK+2(tsk_stk:タスクスタックポインタ定義テーブル)

タスクスタックポインタ定義テーブルには、タスクID＝１から順にタスクの初期起動時のタスクスタックポインタを設定します。該当タスクIDのタスクを登録しない場合は、０を設定してください。

共有タスクスタック機能を用いる場合は、スタックを共有する複数のタスクに同一の初期起動時のタスクスタックポインタを指定してください。このとき、初期起動時のタスクスタックポインタから２バイトが共有タスクスタック管理に必要ですので、確保するようにしてください。

ＨＩ８－３Ｘは、初期起動時のタスクスタックポインタが他の領域と重なっていないかどうかのチェックする機能を持っていません。個々のタスクスタック領域は、充分注意して確保するようにしてください。

スタック領域の算出方法は、「付録Ｄ　メモリ容量の算出」を参照してください。

(3) H$TSK_CNT(タスク定義数)

タスク定義数には、構築するシステムに必要なタスクの総数を設定します。未登録に設定したタスクもタスク定義数に含まれます。

ＨＩ８－３Ｘは、登録されたタスク毎にＴＣＢ(タスク管理用の制御領域）を２バイトおよび、タスクスタックポインタセーブ領域を２バイト生成します。

よって、ＴＣＢ領域(タスク管理用の制御領域）として、(タスク定義数）×２バイトの領域を確保し、この領域の先頭アドレス－２には『H$TCB*1』というラベルを付けてEXPORT宣言をしてください。また、タスクスタックポインタセーブ領域として、(タスク定義数）×２バイトの領域を

確保し、この領域の先頭アドレス－２には『H$TSKSPTBL*1』というラベルを付けてEXPORT宣言をしてください。

これらのラベル『H$TCB*1』と『H$TSKSPTBL*1』は、タスク定義数が０の場合でも必要です。ただし、このときは領域を確保する必要はありません。

【注】*1　このラベルの領域を奇数アドレスから確保した場合、システム起動時にシステム異常終了します。

図４－５　にタスク定義テーブルのフォーマットを示します。

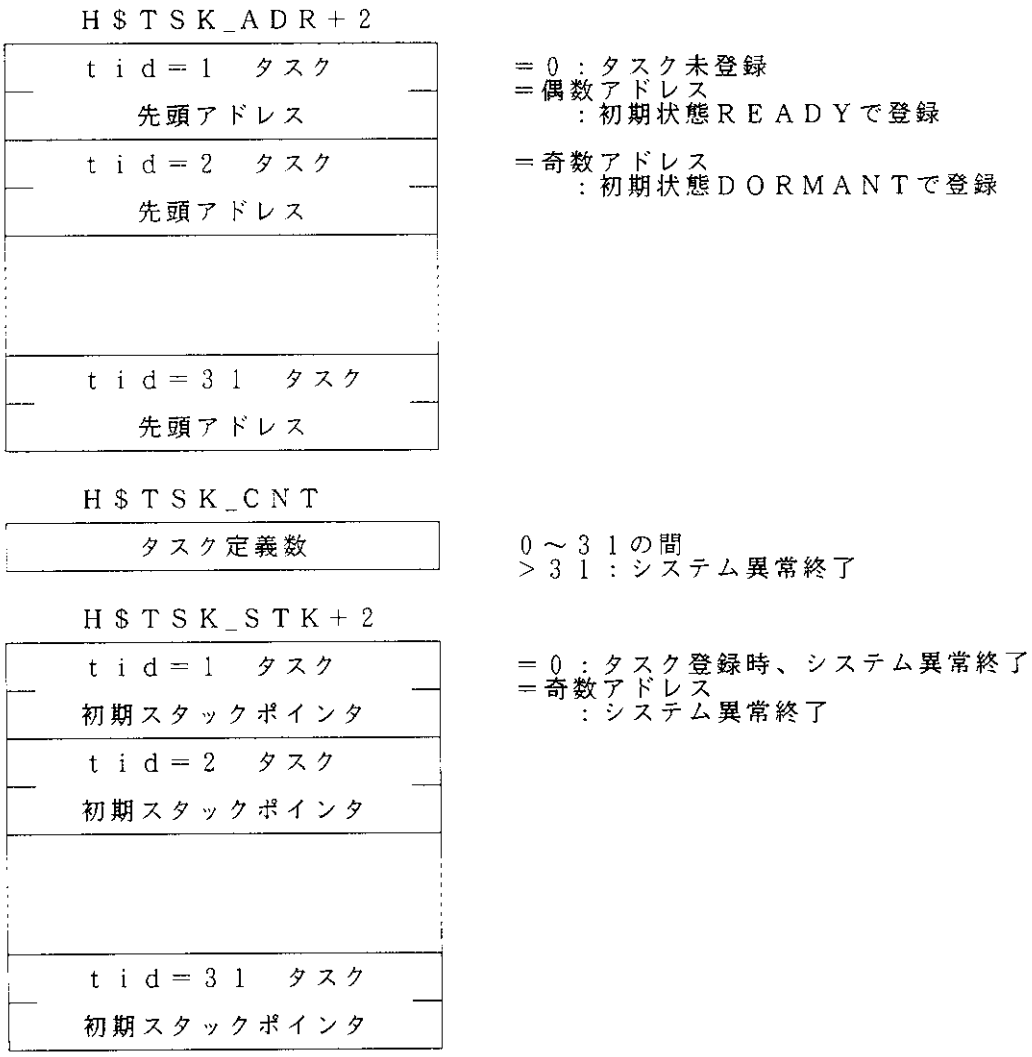


図４－５　タスク定義テーブルのフォーマット

４．４．３　イベントフラグ定義テーブルの作成

イベントフラグを利用するためには、イベントフラグ情報をセットアップファイルのイベントフラグ定義テーブルに登録しなければなりません。

イベントフラグ情報には、以下のものがあります。

(1) H$FLG_CNT(イベントフラグ定義数)

イベントフラグ定義数には、構築するシステムに必要なイベントフラグの総数を設定します。

ＨＩ８－３Ｘは、登録されたイベントフラグ毎にＦＬＧＣＢ（イベントフラグ管理用の制御領域）を２バイト生成します。

よって、ＦＬＧＣＢ用領域（イベントフラグ管理用の制御領域）として、（イベントフラグ定義数）×２バイトの領域を確保し、この領域の先頭アドレス－２には『H$FLGCB 』というラベルを付けてEXPORT宣言をしてください。

このラベル『H$FLGCB』は、イベントフラグ定義数が０の場合でも必要です。ただし、このときは領域を確保する必要はありません。

図４－６　にイベントフラグ定義テーブルのフォーマットを示します。

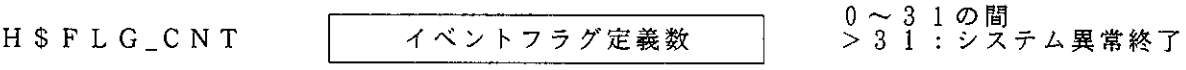


図４－６　イベントフラグ定義テーブルのフォーマット

４．４．４　セマフォ定義テーブルの作成

セマフォを利用するためには、セマフォ情報をセットアップファイルのセマフォ定義テーブルに登録しなければなりません。

セマフォ情報には、以下のものがあります。

(1) H$SEM_CNT(セマフォ定義数)

セマフォ定義数には、構築するシステムに必要なセマフォの総数を設定します。

ＨＩ８－３Ｘは、登録されたセマフォ毎にＳＥＭＣＢ（セマフォ管理用の制御領域）を３バイト生成します。

よって、ＳＥＭＣＢ用領域（セマフォ管理用の制御領域）として、（セマフォ定義数）×３バイトの領域を確保し、この領域の先頭アドレス－３には『H$SEMCB』というラベルを付けてEXPORT宣言をしてください。

このラベル『H$SEMCB』は、セマフォ定義数が０の場合でも必要です。ただし、このときは領域を確保する必要はありません。

図４－７　にセマフォ定義テーブルのフォーマットを示します。

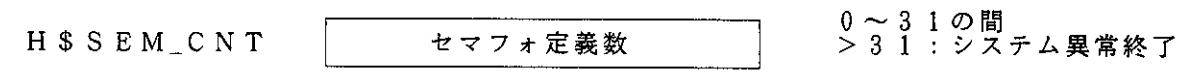


図４－７　セマフォ定義テーブルのフォーマット

４．４．５　メイルボックス定義テーブルの作成

メイルボックスを利用するためには、メイルボックス情報をセットアップファイルのメイルボックス定義テーブルに登録しなければなりません。

メイルボックス情報には、以下のものがあります。

(1) H$MBX_CNT(メイルボックス定義数)

メイルボックス定義数には、構築するシステムに必要なメイルボックスの総数を設定します。

Ｈ８－３Ｘは、登録されたメイルボックス毎にＭＢＸＣＢ(メイルボックス管理用の制御領域）を４バイト生成します。

よって、ＭＢＸＣＢ用領域(メイルボックス管理用の制御領域）として、(メイルボックス定義数）×４バイトの領域を確保し、この領域の先頭アドレス－４には『H$MBXCB*1』というラベルを付けてEXPORT宣言をしてください。

このラベル『H$MBXCB*1』は、メイルボックス定義数が０の場合でも必要です。ただし、このときは領域を確保する必要はありません。

【注】 *1 このラベルの領域を奇数アドレスから確保した場合、システム起動時にシステム異常終了します。

図４－８　にメイルボックス定義テーブルのフォーマットを示します。

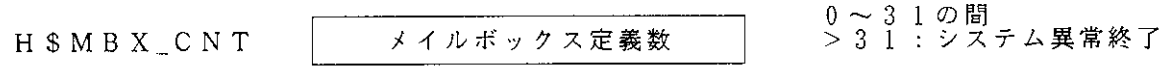


図４－８　メイルボックス定義テーブルのフォーマット

４．４．６　Ｈ８－３ＸのＲＡＭ領域の定義およびシステム管理テーブルの作成

Ｈ８－３Ｘの作業領域として、以下のものがあります。

(1)H$OS_SP*1　（ＯＳ用スタックポインタ・領域）

必ず、ＯＳ用スタックポインタを定義してください。

(2)H$TM_SP*1　（タイマ用スタックポインタ・領域、スタックポインタセーブ領域）

タイマ用スタックポインタを定義してください。wai_tsk 、set_tim 、get_tim システムコールを使用する場合は、タイマ割込み処理ハンドラを登録しなければなりません。この場合、必ずタイマ用スタック領域、およびタイマ用スタックポインタから２バイトをスタックポインタセーブ領域として使用しますので、確保するようにしてください。

【注】ＯＳ用、タイマ用のスタック領域の算出方法は、「付録Ｄ　メモリ容量の算出」を参照してください。

(3) システム管理テーブル(HCUR_SP, HTOP_PRI_ID, H$RDYTBL, H$SYS_CLK, HTIM_CNT, HTIMCB, H$TIMCB_BTM)

Ｈ Ｉ ８ － ３ Ｘのシステム管理テーブルとして、『H$CUR_SP*1』は偶数アドレスから２バイト、『H$TOP_PRI_ID』は１バイト、『H$RDYTBL』は４バイトを必ず、確保してください。

wai_tsk 、set_tim 、get_tim システムコールを使用する場合は、タイマ割込み処理ハンドラを登録しなければなりません。この場合、必ず、『H$SYS_CLK*1』は偶数アドレスから４バイト、『H$TIM_CNT』は１バイト、『H$TIMCB*1』は、偶数アドレスから(wai_tskシステムコールを使用するタスク数）×４バイト、『H$TIMCB』の領域のボトムアドレスにラベル『H$TIMCB_BTM*1*2』を定義してください。

これらのラベルは、使用するしないにかかわらず、必ず定義し、EXPORT宣言をしてください。

【注】 *1 このラベルの領域を奇数アドレスから確保した場合、システム起動時にシステム異常終了します。

*2 このラベルを『H$TIMCB』より低いアドレスに定義した場合、システム起動時にシステム異常終了します。

(4) セットアップテーブルのチェック処理(H_chk_init)

Ｈ Ｉ ８ － ３ Ｘの起動時にセットアップテーブルのチェック処理を行ないたい場合、必ず『H_chk_init』をimport宣言してください。なお、システム結合時にパラメータチェック機能ありのライブラリを使用してください。

セットアップテーブルのチェック処理を行なわない場合、『H_chk_init』をimport宣言しないでください。

４．４．７　セットアップファイルのアセンブル

このように、これらのセットアップ用テーブルをソースファイルとして作成し、アセンブルすることによりシステムと結合可能なオブジェクトファイルが生成されます。

セットアップファイルをアセンブルするには次のコマンドを入力します。

```
% asm38 △< セットアップファイル名 >△-cpu＝300(RET)
```

この操作で、拡張子『.obj』のセットアップファイルのオブジェクトファイルが生成されます。

Ｈ８／３００のアセンブラの詳しい説明は、「Ｈ８／３００シリーズ　クロスアセンブラ　ユーザーズマニュアル」を参照してください。

４．５　割込みベクタテーブルの作成

標準提供の割込みベクタテーブルのファイルは、『h3xvec25.src』です。これを参考にして作成してください。

４．５．１　割込みベクタテーブルの作成

割込みベクタテーブル（ファイル名『h3xvec25.src』）は、システム動作中に割込みが発生した場合にその割込みの種類に対応した処理を行なうように各割込み処理ハンドラの先頭アドレスを登録しておくテーブルです。

ＨＩ８－３Ｘを動作させるには、ベクタ番号０に『H_3X_INIT 』を必ず登録しなければなりません。

また、使用しないベクタ番号には『H_3XINT ××』（××はベクタ番号）を登録してください。

このように登録すると、未定義割込みが発生した場合、システム異常終了となります。

詳しい内容は、「２．５　割込み管理」を参照してください。

割込みベクタテーブルは、システムとの結合時にメモリ領域の０番地から配置してください。

他のＨ８／３００シリーズを使用する場合、割込み要因の数が異なる為、割込みベクタテーブル『h3xvec25.src』および未定義割込み処理『h3xuni25.src』の修正が必要です。割込み要因の数に応じて、それぞれのファイルにシンボル『H_3XINT××』を追加してください。割込み要因の詳細は、当該CPUのハードウェアマニュアルを参照してください。

なお、Ｈ８／３３０用の未定義割込み処理『h3xuni30.src』および割込みベクタテーブル『h3xvec30.src』も提供しています。

図４－９　に割込み処理ハンドラの登録を示します。

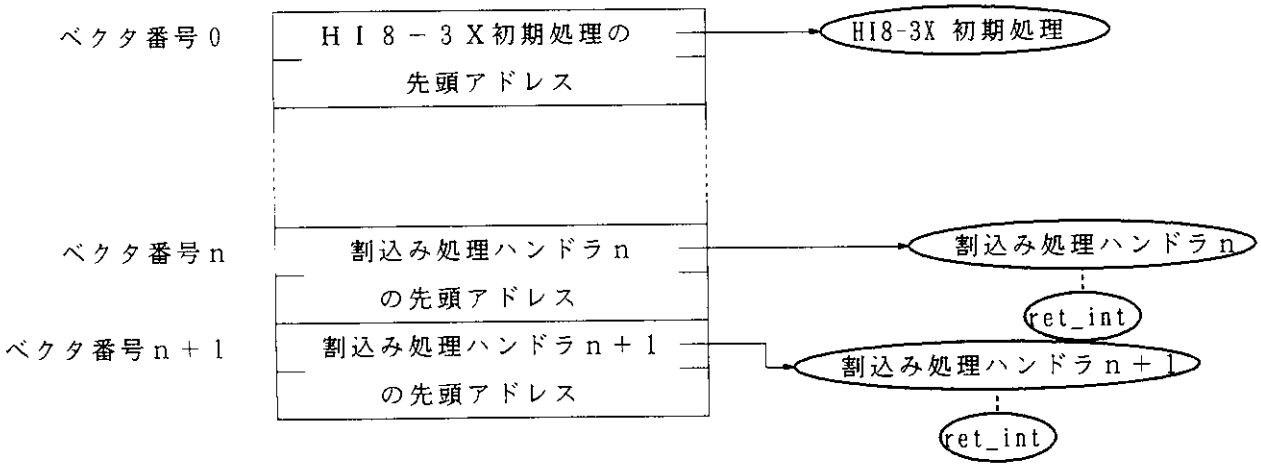
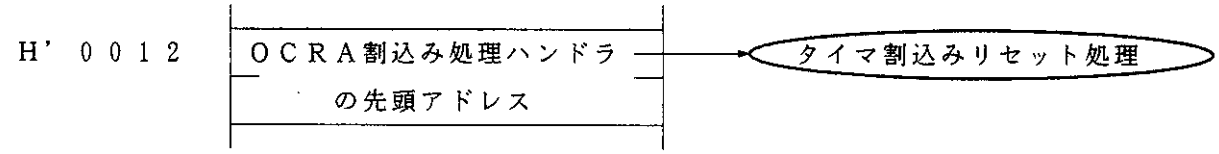


図４－９　割込み処理ハンドラの登録

wai_tsk 、set_tim 、get_tim システムコールを使用する場合は、タイマ割込み処理ハンドラを登録しなければなりません。Ｈ８／３２５において、標準提供のタイマ割込み処理ハンドラ（タイマ割込みリセット処理）を登録する場合は、先頭アドレス（『H_3X_TIM』）を割込みベクタ９のテーブルに設定してください。

図４－１０　にタイマ割込みリセット処理の登録を示します。



【注】 H8/325 シリーズのタイマ割込み処理ハンドラの登録です。他のH8/300シリーズを使用する場合は、当該ＣＰＵのハードウェアマニュアルを参照してください。

図４－１０　タイマ割込みリセット処理の登録

リスト４－１　に割込みベクタテーブルのコーディング例を示します。

```
;3333333333333333333333333333333333333333333333333333333333333333333333333333
;XXX                                                                        XXX
;XXX    H18-3X Version (uITRON V2.01/02)                                    XXX
;XXX    H18-3X/nucleus H8/325 vector table    Ver1.00                       XXX
;XXX                                  in Tokyo Japan date:Feb.15.'91 XXX
;XXX                                                                        XXX
;3333333333333333333333333333333333333333333333333333333333333333333333333333
;       .sdata          "H18-3X/nucleus H8/325 vector V1.00"
;       .sdata          "Copyright (c)  Hitachi, Ltd. 1991."
;       .sdata          "Licensed Material of Hitachi, Ltd."
;|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        .program        h3xvectr
        .heading        "### h3xvec25.src : for H8/325 ###"
        .section        hi8_3x_v,code,locate=0
;
        .import H_3X_INIT,H_3XINT01,H_3XINT02,H_3XINT03
        .import H_3XINT04,H_3XINT05,H_3XINT06,H_3XINT07
        .import H_3XINT08,H_3XINT09,H_3XINT10,H_3XINT11
        .import H_3XINT12,H_3XINT13,H_3XINT14,H_3XINT15
        .import H_3XINT16,H_3XINT17,H_3XINT18,H_3XINT19
        .import H_3XINT20,H_3XINT21,H_3XINT22,H_3XINT23
;
        .import H_3X_TIM                :: in 'h3xtim'
        .import H_CNSHDL0ER
        .import H_CNSHDL0RX
        .import H_CNSHDL0TX             ;: in 'h3xcns_0'
;
;*************************************************************************
;*specifications ;                                                       *
;*name      = h3xvectr : h8/325 vector table for HI8-3X            ;   *
;*function  = 1. h8 interrupt handler address define for hi8       ;   *
;*          = 2. h8 exception handler address define for hi8       ;   *
;*          = 3. hi8 system standard support module                ;   *
;*          =    (1) reset    : "H_3XINIT"   for power on          ;   *
;*          = 4. hi8 system standard support module                ;   *
;*          =    (1) FRT  ocia: "H_3X_TIM"   for system timer      ;   *
;*          = 5. hi8 system standard support i/o interrupt handler ;   *
;*          =    (1) for console i/o driver [sci use]              ;   *
;*          =             (a) : "H_CNSHDL0ER" for receive error    ;   *
;*          =             (b) : "H_CNSHDL0RX" for receive  (input) ;   *
;*          =             (c) : "H_CNSHDL0TX" for transmit (output);   *
;*date      = 02/01/91                                                ;  *
;*author    = Hitachi, Ltd.                                           ;  *
;*history   = V1.00                                                   ;  *
;*attribute = public                                                  ;  *
;*class     = unit                                                    ;  *
;*linkage   = h8 vector table top address = h'0000 for HI8-3X         ;  *
;*input     = none                                                    ;  *
;*output    = none                                                    ;  *
;*parameter = @(0,r7) : interrupt before "CCR"                        ;  *
;*          = @(2,r7) : interrupt before "PC"(next mnemonic)          ;  *
;*end of specifications ;                                                *
;*************************************************************************
```

リスト４－１　割込みベクタテーブルのコーディング例（１／２）

```
;--------.data.w < address >            ;: h8/325 vector no. contents
        .data.w H_3X_INIT               ;: vector no.00 <reset>
        .data.w H_3XINT01               ;: vector no.01 [reserve]
        .data.w H_3XINT02               ;: vector no.02 [reserve]
        .data.w H_3XINT03               ;: vector no.03 <NMI         >
        .data.w H_3XINT04               ;: vector no.04 <irq0        >
        .data.w H_3XINT05               ;: vector no.05 <irq1        >
        .data.w H_3XINT06               ;: vector no.06 <irq2        >
        .data.w H_3XINT07               ;: vector no.07 <port        >
        .data.w H_3XINT08               ;: vector no.08 <FRT    ici  >
        .data.w H_3X_TIM     ;H_3XINT09 ;: vector no.09 <FRT    ocia >
        .data.w H_3XINT10               ;: vector no.10 <FRT    ocib >
        .data.w H_3XINT11               ;: vector no.11 <FRT    ovi  >
        .data.w H_3XINT12               ;: vector no.12 <8bitT0 cmia >
        .data.w H_3XINT13               ;: vector no.13 <8bitT0 cmib >
        .data.w H_3XINT14               ;: vector no.14 <8bitT0 ovi  >
        .data.w H_3XINT15               ;: vector no.15 <8bitT1 cmia >
        .data.w H_3XINT16               ;: vector no.16 <8bitT1 cmib >
        .data.w H_3XINT17               ;: vector no.17 <8bitT1 ovi  >
        .data.w H_CNSHDL0ER  ;H_3XINT18 ;: vector no.18 <SCI0   eri  >
        .data.w H_CNSHDL0RX  ;H_3XINT19 ;: vector no.19 <SCI0   rxi  >
        .data.w H_CNSHDL0TX  ;H_3XINT20 ;: vector no.20 <SCI0   txi  >
        .data.w H_3XINT21               ;: vector no.21 <SCI1   eri  >
        .data.w H_3XINT22               ;: vector no.22 <SCI1   rxi  >
        .data.w H_3XINT23               ;: vector no.23 <SCI1   txi  >

        .end
```

リスト４－１　割込みベクタテーブルのコーディング例（２／２）

リスト４－１　の例では、ベクタ番号０に、Ｈ８－３Ｘの起動処理プログラム、ベクタ番号９にタイマ割込み処理ハンドラ、ベクタ番号１８～２０に、コンソールドライバの割込み処理ハンドラを登録しています。それ以外は、システム異常終了するための未定義割込みのハンドラを登録しています。

４．５．２　割込みベクタテーブルのアセンブル

このように、割込みベクタテーブルをソースファイルとして作成し、アセンブルすることにより、システムと結合可能なオブジェクトファイルが生成されます。

割込みベクタテーブルファイルをアセンブルするには次のコマンドを入力します。

% asm38 △＜割込みベクタテーブルファイル名＞△-cpu＝300(RET)

この操作で、拡張子『.obj』の割込みベクタテーブルのオブジェクトファイルが生成されます。

Ｈ８／３００のアセンブラの詳しい説明は、「Ｈ８／３００シリーズ　クロスアセンブラ　ユーザーズマニュアル」を参照してください。

他のＨ８／３００シリーズを使用する場合、修正した未定義割込み処理および割込みベクタテーブルをアセンブルし、オブジェクトファイルを作成してください。

４．６　システムの結合

これまでに個別に作成したオブジェクトファイルをもとに、システムとしての最終的な実行形式のロードモジュール(アブソリュートプログラム）ファイルを生成します。

図４－１１　にシステム結合の概要を示します。

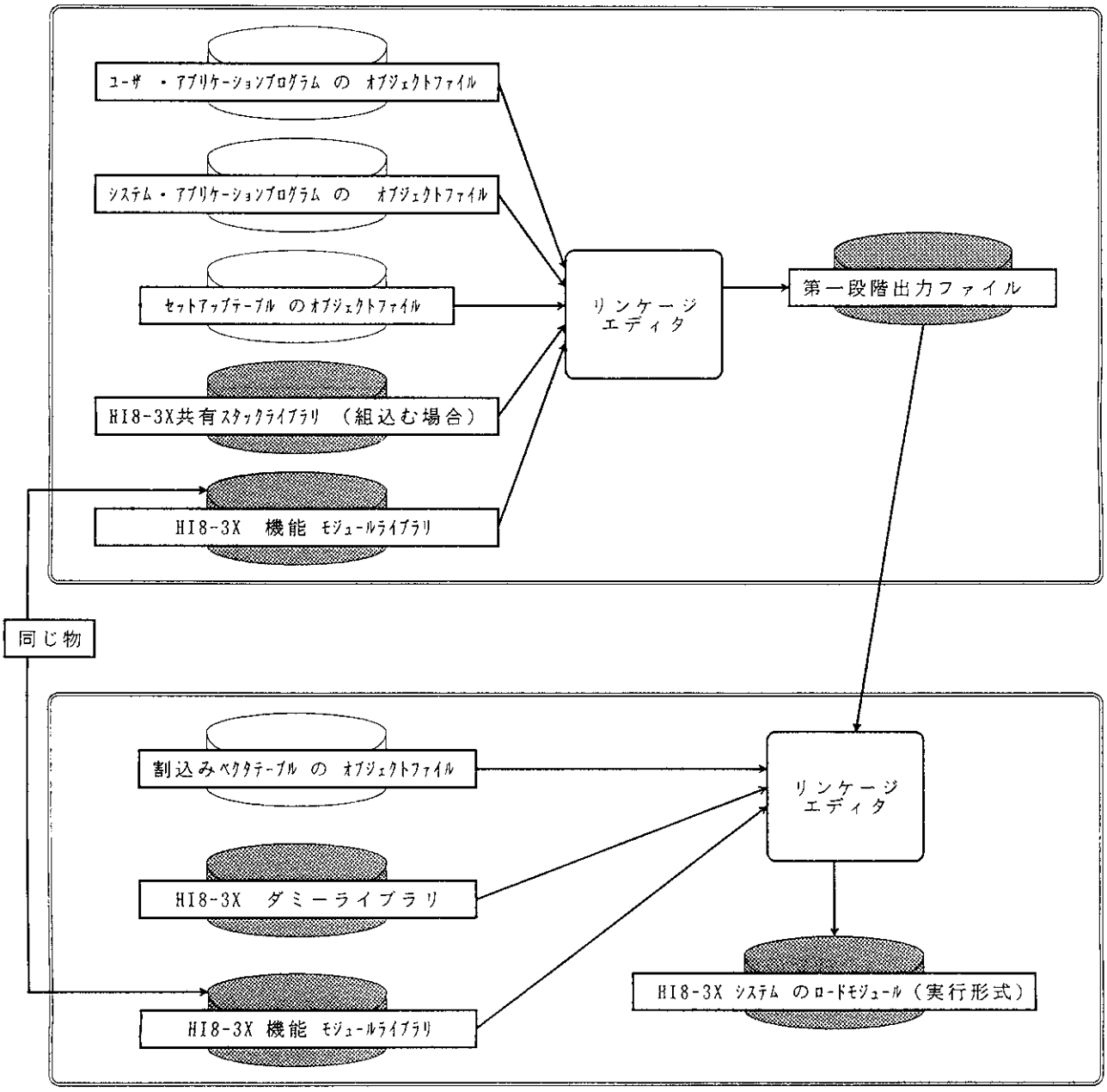


図４－１１　システム結合の概要

４．６．１　Ｈ８－３Ｘシステム結合方法

図４－１１　に示したように、構築するシステムに必要なオブジェクトファイルやライブラリファイルをリンケージエディタにより結合し、Ｈ８－３Ｘシステムのロードモジュールを作成します。

Ｈ８－３Ｘの使用する機能によって、結合するライブラリを選択して、最適なロードモジュールを生成します。

Ｈ８－３Ｘのライブラリは、Ｒ０～Ｒ３までを保障するライブラリと、Ｒ０～Ｒ６までを保障するライブラリがあり、それぞれ５つに分かれています。

表４－１　にライブラリの機能を以下に示します。

表４－１　ライブラリの機能

項番	機能	ファイル名	備考
1	機能モジュールライブラリ	h3xnumdl.lib h3xncmdl.lib	パラメータチェック機能なし パラメータチェック機能あり
2	共有タスクスタックライブラリ	h3xnustk.lib h3xncstk.lib	パラメータチェック機能なし パラメータチェック機能あり
3	ダミーライブラリ	h3xnudmy.lib	

Ｈ８－３Ｘのシステム結合において、以下に示す２種類の機能モジュールを選択します。

① 共有タスクスタック機能の使用／未使用の選択

② パラメータチェックモジュールを組込まれたシステムコールの使用／未使用の選択

(1) 共有タスクスタック機能の使用／未使用の選択

『h3xnustk.lib』または『h3xncstk.lib』（パラメータチェック機能あり）を結合すると、共有タスクスタック機能を使用することができます。これらのライブラリを結合しないと、共有タスクスタック機能を使用することができません。

(2) パラメータチェックモジュールを組込まれたシステムコールの使用／未使用の選択

『h3xncmdl.lib』、『h3xncstk.lib』（共有タスクスタック機能のライブラリ）を結合すると、パラメータチェックモジュールが組込まれたシステムコールを使用することができます。

『h3xnumdl.lib』、『h3xnustk.lib』（共有タスクスタック機能のライブラリ）を結合すると、パラメータチェックモジュールが組込まれていないシステムコールを使用することができます。

(3) システム結合の順序

Ｈ８－３Ｘのシステム結合は、必ず２つの段階で行なってください。

第１段階では、割込みベクタテーブルおよびダミーライブラリを除くオブジェクトファイル・ライブラリを結合します。第２段階では、第１段階で結合したモジュールと割込みベクタテーブル、ダミーライブラリ、機能モジュールライブラリを結合します。

以下に、Ｈ８－３Ｘのシステム結合時の入力ファイルの段階別の順序を示します。

(a) 第１段階の入力ファイル(使用しているシステムコールモジュールとのリンケージ)

①ユーザ・アプリケーションプログラム

②システム・アプリケーションプログラム

③セットアップファイルオブジェクト

④Ｈ８－３Ｘ共有タスクスタックライブラリ(共有タスクスタック機能を使用する場合のみ)

(必ず、⑤のライブラリより前に入力)

⑤Ｈ８－３Ｘ機能モジュールライブラリ

(b) 第２段階の入力ファイル(ダミーモジュールおよび初期化モジュールとのリンケージ)

①割込みベクタテーブルオブジェクト

②第１段階の出力ファイル

③Ｈ８－３Ｘダミーライブラリ(必ず、④のライブラリより前に入力)

④Ｈ８－３Ｘ機能モジュールライブラリ

(4) システム結合の注意事項

(a) 標準で提供されるＨ８－３Ｘのライブラリは、タスクでＲ０～Ｒ３までのレジスタを保障したライブラリと、Ｒ０～Ｒ６までのレジスタを保障したライブラリに分かれています。

Ｃインタフェースを使用する場合はＲ０～Ｒ６までのレジスタを保障したライブラリを使用してください。

(b) Ｈ８／３２５以外のタイマを使用する場合、システム結合時にタイマ処理ハンドラのオブジェクトファイルを第１段階のユーザ・アプリケーションプログラムに指定してください。

(c) Ｈ８－３Ｘの起動時にセットアップテーブルのチェック処理を行ないたい場合、セットアップファイルで必ず『H_chk_init』をimport宣言し、システム結合時にパラメータチェック機能ありのライブラリを使用してください。

セットアップテーブルのチェック処理を行なわない場合、『H_chk_init』をimport宣言しないでください。

(d) 共有タスクスタック機能を使用する場合、第１段階にＨ８－３Ｘ共有タスクスタックライブラリを入力ファイルとし追加してください。このとき、Ｈ８－３Ｘ共有タスクスタックライブラリは、必ずＨ８－３Ｘ機能モジュールライブラリの前に入力されるようにしてください。後に入力されるようにした場合、共有タスクスタック機能は組込まれません。

４．６．２　リンケージの実行

標準提供の『h3xlnk』は、Ｈ８－３Ｘのシステムをリンケージするためのシェルスクリプトファイルです。

以下に、このシェルスクリプトファイルの起動方法を示します。

```
% h3xlnk△[option] (RET)
```

optionには、次のものがあります。

		＜実行するサブコマンドファイル＞
(ﾃﾞｨﾌｫﾙﾄ) ：	パラメータチェックなし・共有タスクスタック機能なし	h3xlnk.sub
c ：	パラメータチェックあり・共有タスクスタック機能なし	h3xlnkc.sub
s ：	パラメータチェックなし・共有タスクスタック機能あり	h3xlnks.sub
cs ：	パラメータチェックあり・共有タスクスタック機能あり	h3xlnkcs.sub

このシェルスクリプトファイルは、オプションに従って機能別のリンケージエディタのサブコマンドファイルを実行し、各オブジェクト・ライブラリファイルを結合します。

ユーザは、これらのサブコマンドファイルを参考に、ユーザシステムのプログラムファイル、セクションのメモリ配置などを修正・追加して、システムの結合を行なってください。

リスト４－２　に　“パラメータチェックあり・共有タスクスタック機能あり”のサブコマンドファイル『h3xlnkcs.sub』を示します。

```
input   h3xcns_0,h3xsetup ·····①
library h3xncstk,h3xncmdl ·····②
output  h3xprg.rel  ····· ③
form    r
debug
end

input   h3xvec25,h3xprg.rel ·····④
library h3xnudmy,h3xncmdl ····· ⑤
output  h3xprgcs.abs  ····· ⑥
entry   H_3X_INIT ····· ⑦
start   hi8_3x(80),h3xsetup,h3xcns(2000),&
        h3x_ram(0fd80) ····· ⑧
debug
print   h3xprgcs.map  ····· ⑨
exit
```

リスト４－２　サブコマンドファイル『h3xlnkcs.sub』

（解説）

①h3xcns_0.obj, h3xsetup.obj　を入力します。

②h3xncstk.lib, h3xncmdl.lib　を入力します。

③リロケータブルファイル名をh3xprg.relとします。
④h3xvec25.obj, h3xprg.rel　を入力します。
⑤h3xnudmy.lib, h3xncmdl.lib　を入力します。
⑥出力ファイル名をh3xprgcs.absとします。
⑦実行開始アドレスをH_3X_INIT とします。
⑧ＨＩ８－３ＸをH'0080番地から配置します。
　セットアップテーブル、コンソールドライバをH'2000番地から配置します。
　ＨＩ８－３Ｘの作業領域をＨ８／３２５内蔵ＲＡＭのH'FD80番地から配置します。
⑨このシステムのリンケージリストをリストファイルh3xprgcs.mapへ出力します。

リンケージエディタの詳しい説明は「Ｈシリーズ　リンケージエディタ　ユーザーズマニュアル」を参照してください。

４．７　システムの起動

システムの結合で作成したロードモジュールをユーザ実機(ターゲットシステム）にダウンロードするには、実機デバッグ装置Ｈ８／３００シリーズＡＳＥを使用する方法と、Ｈ８／３００内蔵ＰＲＯＭまたはＥＰＲＯＭに書き込み、ハードウェア環境に搭載する方法があります。

４．７．１　Ｈ８／３００シリーズＡＳＥによるダウンロード

(1) ＡＳＥを使用してのロードモジュールのダウンロード

以下にＡＳＥを使用してユーザ実機(ターゲットシステム）にロードモジュールをダウンロードする方法を説明します。

①ホストシステムにてＨシリーズインタフェースソフトを起動します。

②ＡＳＥシステムを起動します。

③ＡＳＥのＬＯＡＤコマンドを使用して、ホストシステムからロードモジュールをロードします。

：LOAD△:<ホストシステムのファイル名>（RET）

(2) システムの起動方法

ロードモジュールのスタートアドレスからプログラムを実行すると、ＨＩ８－３Ｘシステムが起動します。

スタートアドレスには、ＨＩ８－３Ｘニュークリアスの先頭アドレス　『H_3X_INIT 』（ベクタ番号０の内容のアドレス）を指定します。

ＡＳＥのコマンドを使用した起動方法を次に示します。

: RESET (RET)
: GO (RET)

４．７．２　ハードウェア環境への搭載によるダウンロード（Ｐｏｗｅｒ　ＯＮによる起動）

電源をＯＮした状態からシステムを起動するには、システムのロードモジュールをＲＯＭに書き込み、ハードウェア環境に実装することにより、実現します。

ロードモジュールをフォーマット変換し、モトローラＳタイプ形式ロードモジュールを作成します。

ホストシステムで次のコマンドを実行します。

% cnvs△＜ロードモジュール＞（RET)

この操作で作成したモトローラＳタイプ形式ロードモジュールをＨ８／３００内蔵ＰＲＯＭまたは、ＥＰＲＯＭに書込み、ハードウェア環境に実装します。

４．８　システムの異常終了

ＨＩ８－３Ｘのパラメータチェックモジュールライブラリを組込んだシステムを起動すると、最初にセットアップテーブルのチェックを行ないます。

システム起動時にセットアップテーブルの不正が発生すると、処理が続行できないためシステムを異常終了します。

このシステム異常終了時には、レジスタR0・R1に次の情報を設定し、割込み禁止状態で、システム異常終了します。

・レジスタR0　　発生したエラーの種類(セットアップテーブルの不正：H'1111)
・レジスタR1　　不正情報を設定している定義テーブルのアドレス

また、ＨＩ８－３Ｘは通常動作中に異常が発生すると、発生したエラーの種類をレジスタR0に設定し、割込み禁止状態で、システム異常終了します。

ＨＩ８－３Ｘにおいて、通常動作中のシステム異常を認識できる項目は、以下のとおりです。

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

システム異常終了時は、ＡＳＥのブレークキー等によりＨＩ８－３Ｘの実行を中断し、レジスタR0・R1を手がかりに異常箇所を見つけてください。

システムの異常終了処理については、「２．７　システムの異常終了処理」を参照してください。

5．Cインタフェース

5．1　概　要

ＨＩ８－３Ｘ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『hi8_3x_c.lib』とＣ言語ヘッダファイル『hi8_3x.h』から構成されています。

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_3x.h』をソースプログラム中にインクルードし、システム結合時に『hi8_3x_c.lib』を一緒に結合してください。

表５－１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５－１　Ｃ言語インタフェースライブラリファイル一覧

項目	ファイル名	内　　容
1	hi8_3x_c.lib	HI8-3Xの33個のシステムコールからret_int システムコール*を除いた32個のシステムコールのライブラリファイル
2	hi8_3x.h	コーディングで用いる型・定数・原型宣言を定義しているヘッダファイル（インクルードファイル）

【注】 *ret_int システムコールは、アセンブリ言語で記述します。

Ｃ言語で記述されたプログラムは、構造の違いから２つの種類に分けられます。

(1) タスク

Ｃ言語のみ記述します。

(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。

５．２　タスク

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。

```
#include "hi8_3x.h"

void taskXX(void)
{
ID tskid;
ER ercd;

        .
        .
        .
    ercd = wup_tsk(tskid);      /* systemcall */
    if (ercd !=E_OK)  {
        .
        .
        .
    ext_tsk();                  /* end of taskXX */
}
```

図５－１　Ｃ言語でのタスクの記述例

５．３　ハンドラ

Ｃ言語で割込みハンドラやユーザリセットルーチンを記述する場合、スタックポインタの切替えやレジスタの保障が必要となるため、アセンブル埋め込み機能を使用して記述しなければなりません。

アセンブル埋め込み機能でのコーディングを必要とするプログラムを以下に示します。

(1) 割込みハンドラ
(2) ユーザリセットルーチン

タイマ割込みリセット処理も一部をアセンブル埋め込み機能で記述することによりＣ言語で作成することが可能です。

５．３．１　割込みハンドラのコーディング

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) レジスタの保障

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_3x.h"


void h3xintXX(void)
#pragma asm
        .import STKM,TRAPSP       ;: sp save area/stack atea              ←― ①
        .import ret_int
;
        mov.w   sp,@STKM          ;: sp save to STKM                      ←― ②
        mov.w   #TRAPSP,sp        ;: change sp                            ←― ③
        push    r0                ;: variable and temporaly register save ←― ④
        push    r1
        bsr     _h3xintXX_main                                            ←― ⑤
        pop     r1                ;: register recover                     ←― ⑥
        pop     r0
        mov.w   @STKM,sp          ;: sp recover                           ←― ⑦
        jmp     @ret_int          ;: return interrupt                     ←― ⑧
;
_h3xintXX_main:
#pragma endasm
{
ID tskid;
ER ercd;
        .
        .
        .
        ercd = iwup_tsk(tskid);     /* systemcall (for interrupt handler) */
        if (ercd != E_OK)  {
        .
        .
        .
}
```

図５－２　C言語での割込みハンドラの記述例

（解説）

①割込みハンドラ専用スタックおよび割込み発生時のスタックポインタ退避領域を外部参照シンボルとして宣言

②割込み発生時のスタックポインタ退避

③割込みハンドラ専用スタックへの切替え

④レジスタの保障

⑤割込みハンドラの呼出し

⑥レジスタの回復

⑦スタックポインタの回復

⑧ret_int システムコールの発行

５．３．２　ユーザリセットルーチンのコーディング

ユーザリセットルーチンの起動時には、以下の処理が必要になります。

(1) ユーザリセットルーチン専用スタックへの切替え
(2) レジスタの保障

ユーザリセットルーチンの終了時には、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

ユーザリセットルーチンは割込みハンドラと同様なコーディングを行ないます。

ただし、ユーザリセットルーチンの終了は、ret_int システムコールでなく、RTS 命令を実行します。

５．４　実行形式プログラムの作成

Ｃ言語で記述したプログラムをコンパイル、結合する場合の注意を以下に示します。

５．４．１　オブジェクトモジュールの作成

Ｃ言語で記述したタスクや割込みハンドラなどのユーザプログラムを、他のオブジェクトモジュールやライブラリファイルと結合可能にするために、コンパイルしてオブジェクトモジュールを生成します。

コンパイル時は、コンパイラオプションとしてオブジェクト種類にＨ８／３００用レジスタモード（-cpu=300reg)を指定してください。

また、割込みハンドラ等のユーザプログラムでアセンブラ埋め込み機能を使用している場合、コンパイラオプションとしてオブジェクト形式にアセンブリソースプログラム出力(-code=asmcode)を指定し、出力されたアセンブリソースプログラムをアセンブルしてください。

Ｈ８／３００シリーズ　Ｃコンパイラの詳しい使用方法は、「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。

５．４．２　オブジェクトモジュールの結合

コンパイラによって作成されたオブジェクトモジュールを他のオブジェクトモジュールやライブラリファイルと結合することによりＨＩ８－３Ｘシステムを作成します。

Ｃ言語でＨＩ８－３Ｘのシステムコールを使用する場合は、入力ファイルにＣ言語インタフェースライブラリ『hi8_3x_c.lib』を追加してください。

また、Ｃ言語インタフェースライブラリのセクション『h3xc』（セクションの属性は、code）を追加してください。

リンケージエディタの詳しい使用方法は、「Ｈシリーズ　リンケージエディタ　ユーザーズマニュアル」を参照してください。

図５－３に、Ｃ言語で記述したプログラムを結合する場合のリンケージエディタのサブコマンドファイルの例を示します。

```
input    h3xcns_0,h3xsetup, <ユーザのオブジェクトモジュール>          ←――― ①
library  h3xnumdl,hi8_3x_c,c38reg                                  ←――― ②
output   h3xprg.rel
form     r
debug
end

input    h3xvec25,h3xprg.rel
library  h3xnudmy,h3xnumdl
output   h3xprg.abs
entry    H_3X_INIT
start    hi8_3x(80),h3xsetup,h3xcns,h3xc,P,C(2000),&               ←――― ③
         h3x_ram,D,B(0fd80)
debug
print    h3xprg.map
exit
```

図５－３　Ｃ言語プログラムを結合する場合のサブコマンドファイルの例

（解説）

①ユーザプログラムのオブジェクトモジュールを入力します。

②Ｃインタフェースライブラリ(hi8_3x_c.lib)と標準ライブラリ(c38reg.lib)を入力します。

③Ｃインタフェースライブラリ(h3xc)とユーザプログラムのセクション名(P,C,D,B) を追加します。

５．５　Ｃ言語インタフェースライブラリの作成

Ｃ言語インタフェースライブラリのソースを変更しライブラリを作成する場合、次のように行なってください。

標準提供の『hi8_3x_c』はＣ言語インタフェースライブラリのライブラリファイルを作成するためのシェルスクリプトファイルです。

このシェルスクリプトファイルは、Ｃ言語インタフェースライブラリのソースプログラムファイルからオブジェクトモジュールの作成およびライブラリファイルの作成を行ないます。

以下に、このシェルスクリプトファイルの起動方法を示します。

```
% hi8_3x_c△[option] (RET)
```

optionには、次のものがあります。

(ﾃﾞｨﾌｫﾙﾄ)　：Ｃ言語インタフェースライブラリのソースをアセンブルし、ライブラリを作成します。

1　　：Ｃ言語インタフェースライブラリのアセンブルリストを出力します。

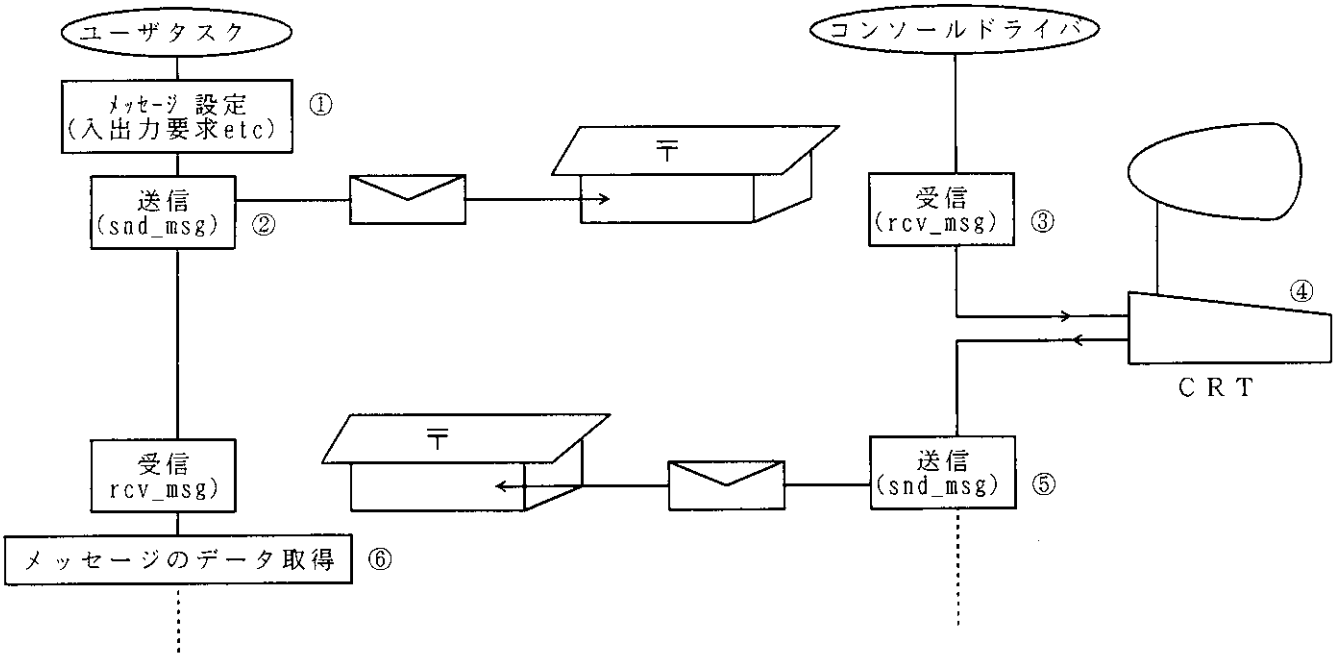
この操作で、Ｃ言語インタフェースライブラリのライブラリファイル『hi8_3x_c.lib 』が作成されます。

付録Ａ．コンソールドライバの例題

Ａ．１　コンソールドライバの概要

サンプルとして、Ｈ８／３２５内蔵のＳＣＩ(Serial Communication Interface)を用いたコンソールドライバを提供しています。他のＨ８／３００シリーズを使用する場合は、当該ＣＰＵのハードウェアマニュアルのＳＣＩの仕様を参照してください。以下にこのコンソールドライバの機能を説明します。

コンソールドライバは、ユーザタスク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。



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タスク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、ユーザタスクはタスクごとにメイルボックスを１つ持つようにします。

Ａ．２　ユーザタスクによるコンソール入出力処理

ユーザ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)　ユーザタスク・メッセージ設定　　（図Ａ－１の①）

ユーザタスクが、メッセージに入出力要求のデータを設定します。

（詳しいメッセージフォーマットは、「Ａ．４　メッセージのフォーマット」を参照してください。）

(2)　ユーザタスク・入出力要求送信　　(図Ａ－１の②)

ユーザタスクは、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、ユーザタスクからの入出力要求(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による入出力処理 (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、入出力処理を行ないます。

(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、ユーザタスク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)　ユーザタスク・処理結果受信　　(図Ａ－１の⑥)

ユーザタスクは、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．３　コンソールドライバの機能

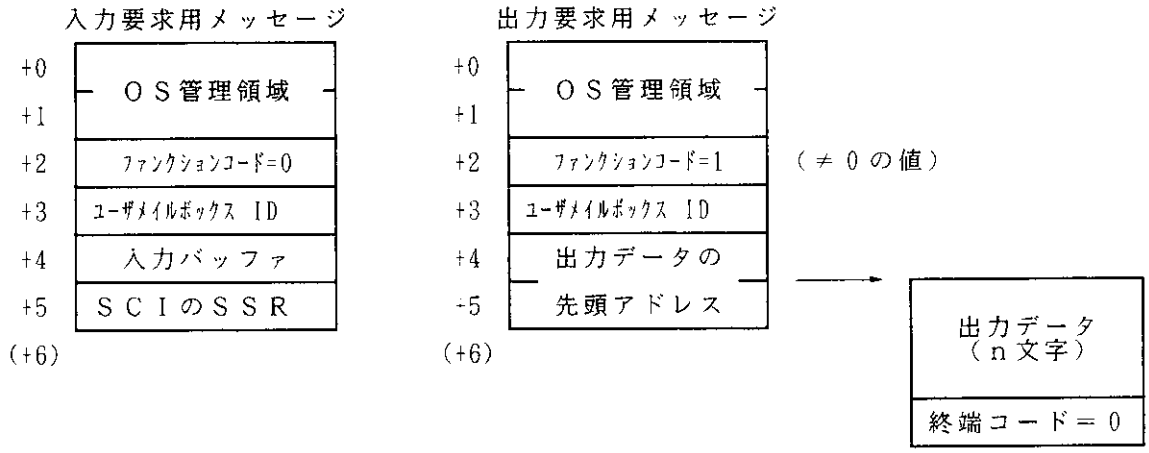
コンソールドライバは、以下の機能をサポートしています。

(1) キー入力データの格納(１文字入力)
(2) ＣＲＴへの文字表示　(ｎ文字出力)

Ａ．４　メッセージフォーマット

コンソールドライバとユーザタスク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われます。

図Ａ－２　にメッセージのフォーマットを示します。



図Ａ－２　コンソールドライバとユーザタスク間のメッセージフォーマット

ユーザタスクは、ファンクションコードに機能コード(リード要求またはライト要求）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。ライト要求の場合は、さらに出力データの先頭アドレスを設定します。これらのデータ設定の後、コンソールドライバに入出力要求(snd_msg　システムコール）を行ないます。

リード要求では、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、ユーザメイルボックスに対しメッセージを送信して、処理の終了報告(snd_msg　システムコール）を行ないます。

ライト要求では、出力データに設定された文字をＣＲＴへ出力します。出力データの終端にNULLコード(H'00)を設定してください。出力処理中にNULLコード(H'00)が検出された場合、コンソールドライバは出力処理を終了し、ユーザメイルボックスに対してメッセージを送信して、処理の終了報告(snd_msg　システムコール）を行ないます。

Ａ．５　コンソールドライバのequate定義

コンソールドライバのソースファイル『h3xcns_0.mar』には、Ｈ８／３２５内蔵のＳＣＩを制御するためのequate定義があります。

Ｈ８／３２５内蔵のＳＣＩの詳しい説明は、「Ｈ８／３２５シリーズ　ハードウェアマニュアル」を参照してください。

図Ａ－３　にコンソールドライバのequate定義を示します。

```
SMR     .equ    h'ffd8                  ;: SCI0 SMR address -> h'ffd8……①
BRR     .equ    h'ffd9                  ;: SCI0 BRR address -> h'ffd9……②
SCR     .equ    h'ffda                  ;: SCI0 SCR address -> h'ffda……③
TDR     .equ    h'ffdb                  ;: SCI0 TDR address -> h'ffdb……④
SSR     .equ    h'ffdc                  ;: SCI0 SSR address -> h'ffdc……⑤
RDR     .equ    h'ffdd                  ;: SCI0 RDR address -> h'ffdd……⑥

SMRDT   .equ    h'04                    ;: SCI0 SMR output data …………⑦
BRRDT   .equ    h'1f                    ;: SCI0 BRR output data …………⑧

CMBXID  .equ    1                       ;: console driver mailbox id ……⑨
CTSKID  .equ    1                       ;: console driver task id ……⑩
```

図Ａ－３　コンソールドライバのequate定義

（説明）
① ＳＣＩのＳＭＲ（シリアルモードレジスタ）のアドレス
② ＳＣＩのＢＲＲ（ビットレートレジスタ）のアドレス
③ ＳＣＩのＳＣＲ（シリアルコントロールレジスタ）のアドレス
④ ＳＣＩのＴＤＲ（トランスミットデータレジスタ）のアドレス
⑤ ＳＣＩのＳＳＲ（シリアルステータスレジスタ）のアドレス
⑥ ＳＣＩのＲＤＲ（レシーブデータレジスタ）のアドレス
⑦ ＳＣＩのＳＭＲに設定するデータ（転送フォーマットの設定：８ビットデータ、パリティなし、１ストップビット）
⑧ ＳＣＩのＢＲＲに設定するデータ（データ転送速度の設定：10MHz時9600bps）
⑨ コンソールドライバが入出力要求メッセージを受信するためのメイルボックスＩＤ
⑩ コンソールドライバタスクのタスクＩＤ

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『h3xcns_0.mar』equate定義は、ＳＣＩ０を使用、データ転送速度が９６００bps、コンソールドライバのタスクＩＤおよびメイルボックスＩＤが１の値が定義されています。

Ａ．６　コンソールドライバの登録

コンソールドライバの登録は、下記の(1),(2) の登録作業を行なってください。

ＳＣＩ０のコンソールドライバの登録を説明します。

(1) セットアップテーブルへの登録

① タスク先頭アドレス定義テーブルへの登録

タスク先頭アドレス定義テーブル（タスクID=1）に、コンソールドライバのタスクの先頭アドレス『H_CNSDRV_0』を登録します。（標準提供のファイル『h3xsetup.src』）

タスク先頭アドレス定義テーブルは、優先度順に配置されています。構築するユーザシステムに

含まれるアプリケーションプログラムの優先度より高い優先度のテーブルに、コンソールドライバの先頭アドレスを登録してください。

② タスクスタックポインタ定義テーブルへの登録

タスクスタックポインタ定義テーブルに（タスクID=1）に、タスク初期起動時のタスクスタックポインタ『tO1_sp』を登録します。（標準提供のファイル『h3xsetup.src』）

コンソールドライバのタスク独自で使用するスタックサイズは２バイトです。

(2) 割込みベクタテーブルへの登録

コンソールドライバ用の割込み処理ハンドラを使用しますので、割込み処理ハンドラの先頭アドレスを割込みベクタテーブル（ 標準提供のファイル『h3xvec25.src』）へ登録します。

・ERi 割込みには、割込み処理ハンドラ名『H_CNSHDLOER』を登録します。

・Rxi 割込みには、割込み処理ハンドラ名『H_CNSHDLORX』を登録します。

・Txi 割込みには、割込み処理ハンドラ名『H_CNSHDLOTX』を登録します。

表Ａ－１　に、Ｈ８／３２５のＳＣＩ割込みの種類を示します。

表Ａ－１　Ｈ８／３２５のＳＣＩ割込みの種類

項番	割込み要因	ＳＣＩ０		ＳＣＩ１	
		ﾍﾞｸﾀ 番号	ﾍﾞｸﾀｱﾄﾞﾚｽ	ﾍﾞｸﾀ 番号	ﾍﾞｸﾀ ｱﾄﾞﾚｽ
1	ERi 割込み（受信ｴﾗｰ 割込み）	18	H'24～H'25	21	H'2A～H'2B
2	Rxi 割込み（入力割込み）	19	H'26～H'27	22	H'2C～H'2D
3	Txi 割込み（出力割込み）	20	H'28～H'29	23	H'2E～H'2F

(3) コンソールドライバ用の割込み処理ハンドラのスタックサイズの設定

ＳＣＩ割込み処理ハンドラのスタック領域は、「付録Ｄ　メモリ容量の算出」を参照してください。

ＳＣＩ割込み処理ハンドラのスタック領域は、ファイル名『h3xcns.inc』のラベル名『CSTACK』に設定されています。ユーザシステムの環境に合わせて設定してください。標準値は、18バイトです。

A．7　コンソールドライバのアセンブル

標準提供の『h3xcns』はコンソールドライバをアセンブルするシェルスクリプトファイルです。サンプルのコンソールドライバのソースファイル『h3xcns_0.mar』，『h3xcns_1.mar』をアセンブルするときは、以下のように行なってください。

```
% h3xcns△[option] (RET)
```

optionには次のものがあります。

(ﾃｨﾌｫﾙﾄ) ：SCI0,SCI1 のコンソールドライバのオブジェクトを出力します。

1　　：SCI0,SCI1 のオブジェクトを出力せずにアセンブルリストを出力します。

これにより、システムと結合可能なオブジェクトを生成することができます。

A．8　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、割込み処理ハンドラ名、セクション名、資源などの詳細情報を示します。

表Ａ－２　に、コンソールドライバの構成を示します。

表Ａ－２　サンプルのコンソールドライバの構成

項番	名称	ｺﾝｿｰﾙﾄﾞﾗｲﾊﾞ 1	ｺﾝｿｰﾙﾄﾞﾗｲﾊﾞ 2
1	ファイル名	h3xcns_0.mar	h3xcns_1.mar
2	使用するＳＣＩ	H8/325 のＳＣＩ０	H8/325 のＳＣＩ１
3	タスク名	H_CNSDRV_0	H_CNSDRV_1
4	ＥＲＩの割込み処理ハンドラ名	H_CNSHDL0ER	H_CNSHDL1ER
5	ＲＸＩの割込み処理ハンドラ名	H_CNSHDL0RX	H_CNSHDL1RX
6	ＴＸＩの割込み処理ハンドラ名	H_CNSHDL0TX	H_CNSHDL1TX
7	コードのセクション名	h3xcns	h3xcns
8	データのセクション名	h3xcns_ram	h3xcns_ram
9	CPU=10MHz 時のボーレート	９６００bps	９６００bps
１０	タスクＩＤ	1	2
１１	メイルボックスＩＤ	1	2

付録Ｂ．システムコール一覧

Ｂ．１　Ｈ８－３Ｘシステムコール・アセンブラインタフェース一覧

	システムコール	パラメータ／リターンパラメータ(=) R0L	R1H	R1L	タスク/非タスク (T)/(N)	
タスク管理						
1	sta_tsk (Start Task)	tskid			T	
2	ista_tsk (Interrupt Start Task)	tskid				N
3	ext_tsk (Exit Task)				T	
4	chg_pri (Change Task Priority)		tskpri		T	
5	get_tid (Get Task Identifier)	tskid=			T	
6	tsk_sts (Get Task Status)	tskid	tskpri=	tskstat=	T	N
タスク付属同期管理						
7	slp_tsk (Sleep Task)				T	
8	wai_tsk (Wait For Wakeup Task)		(R1	tmout)	T	
9	wup_tsk (Wakeup Task)	tskid			T	
10	iwup_tsk (Interrupt Wakeup Task)	tskid				N
11	can_wup (Cancel Wakeup Task)	tskid	wupcnt=		T	
同期／通信管理						
12	set_flg (Set Event Flag)	flgid	setptn		T	
13	iset_flg (Interrupt Set Event Flag)	flgid	setptn			N
14	clr_flg (Clear Event Flag)	flgid	clrptn		T	
15	wai_flg (Wait Event Flag)	flgid	waiptn flgptn=	wfmode	T	
16	pol_flg (Poll Event Flag)	flgid	waiptn flgptn=	wfmode	T	
17	flg_sts (Get Event Flag Status)	flgid wtskid=	flgptn=		T	N
18	sig_sem (Signal Semaphore)	semid			T	
19	isig_sem (Interrupt Signal Semaphore)	semid				N
20	wai_sem (Wait on Semaphore)	semid			T	
21	preq_sem (Poll and Request Semaphore)	semid			T	
22	sem_sts (Get Semaphore Status)	semid wtskid=	semcnt=		T	N
23	snd_msg (Send Message to Mailbox)	mbxid	(R1	pk_msg)	T	
24	isnd_msg (Interrupt Send Message to Mailbox)	mbxid	(R1	pk_msg)		N
25	rcv_msg (Receive Message from Mailbox)	mbxid	(R1	pk_msg=)	T	
26	prcv_msg (Poll and Receive Message)	mbxid	(R1	pk_msg=)	T	
27	mbx_sts (Get Mailbox Status)	mbxid wtskid=	(R1	pk_msg=)	T	N
割込み管理						
28	ret_int (Return from Interrupt Handler)					N
29	chg_ims (Change Interrupt Mask Level)		imask		T	
30	ims_sts (Get Interrupt Mask Level Status)		imask=		T	N
時間管理						
31	set_tim (Set Time)		(R1	pk_time)	T	N
32	get_tim (Get Time)		(R1	pk_time)	T	N
システム管理						
33	get_ver (Get Version No)		(R1	pk_ver)	T	N

【注】(1)(R1 ..... XXXXX)は、レジスタR1に16ビットのパラメータ／リターンパラメータが入ることを示します。

(2) システムコールのエラーコードは、レジスタR0Hに返されます。ただし、ext_tsk, ret_int はエラーコードを返さず、レジスタR1にエラー情報をセットしてシステム異常終了します。

B．2　Ｈ８－３Ｘ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		備　　考
タスク管理				
1 sta_tsk	タスクを起動する	MOV.B JSR	#tskid, R0L @sta_tsk	tskid:タスクID
2 ista_tsk	タスクを起動する （非タスク部専用）	MOV.B JSR	#tskid, R0L @ista_tsk	tskid:タスクID
3 ext_tsk	自タスクを正常終了する	JMP	@ext_tsk	
4 chg_pri	自タスク優先度を変更する	MOV.B JSR	#tskpri, R1H @chg_pri	tskpri: タスク優先度
5 get_tid	自タスクのIDを得る	JSR	@get_tid	
6 tsk_sts	タスクの状態を見る	MOV.B JSR	#tskid, R0L @tsk_sts	tskid:タスクID
タスク付属同期管理				
7 slp_tsk	自タスクを待ち状態へ移行する	JSR	@slp_tsk	
8 wai_tsk	自タスクを一定時間待ち状態に移行する	MOV.W JSR	#tmout, R1 @wai_tsk	tmout:タイムアウト指定
9 wup_tsk	待ち状態のタスクを起床する	MOV.B JSR	#tskid, R0L @wup_tsk	tskid:タスクID
10 iwup_tsk	待ち状態のタスクを起床する（非タスク部専用）	MOV.B JSR	#tskid, R0L @iwup_tsk	tskid:タスクID
11 can_wup	タスクの起床要求を無効にする	MOV.B JSR	#tskid, R0L @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		備　　考
同期／通信管理				
12 set_flg	イベントフラグをセットする	MOV.B MOV.B JSR	#flgid, R0L #setptn, R1H @set_flg	flgid:イベントフラグID setptn: セットするビットパターン
13 iset_flg	イベントフラグをセットする（非タスク部専用）	MOV.B MOV.B JSR	#flgid, R0L #setptn, R1H @iset_flg	flgid:イベントフラグID setptn: セットするビットパターン
14 clr_flg	イベントフラグをクリアする	MOV.B MOV.B JSR	#flgid, R0L #clrptn, R1H @clr_flg	flgid:イベントフラグID clrptn: クリアするビットパターン
15 wai_flg	イベントフラグを待つ	MOV.B MOV.B MOV.B JSR	#flgid, R0L #waiptn, R1H #wfmode, R1L @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
16 pol_flg	イベントフラグを得る	MOV.B MOV.B MOV.B JSR	#flgid, R0L #waiptn, R1H #wfmode, R1L @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
17 flg_sts	イベントフラグ状態を参照する	MOV.B JSR	#flgid, R0L @flg _sts	flgid:イベントフラグID
18 sig_sem	セマフォに対するＶ操作	MOV.B JSR	#semid, R0L @sig_sem	semid:セマフォID
19 isig_sem	セマフォに対するＶ操作（非タスク部専用）	MOV.B JSR	#semid, R0L @isig _sem	semid:セマフォID
20 wai_sem	セマフォに対するＰ操作	MOV.B JSR	#semid, R0L @wai_sem	semid:セマフォID
21 preq_sem	セマフォ資源を得る	MOV.B JSR	#semid, R0L @preqs_em	semid:セマフォID
22 sem_sts	セマフォ状態を参照する	MOV.B JSR	#semid, R0L @sem_sts	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		備　　考
23 snd_msg	メッセージを送信する	MOV.B MOV.W JSR	#mbxid, R0L #pk_msg, R1 @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
24 isnd_msg	メッセージを送信する（非タスク部専用）	MOV.B MOV.W JSR	#mbxid, R0L #pk_msg, R1 @isndmsg	mbxid:メイルボックスID pk_msg:メッセージの先頭アドレス
25 rcv_msg	メッセージの受信を待つ	MOV.B JSR	#mbxid, R0L @rcv_msg	mbxid:メイルボックスID
26 prcv_msg	メッセージを受信する	MOV.B JSR	#mbxid, R0L @prcv_msg	mbxid:メイルボックスID
27 mbx_sts	メイルボックス状態を参照する	MOV.B JSR	#mbxid, R0L @mbx_sts	mbxid:メイルボックスID
割込み管理				
28 ret_int	割り込みハンドラから復帰する	JMP	@ret_int	
29 chg_ims	割り込みマスクを変更する	MOV.B JSR	#imask, R1H @chg_ims	imask:割り込みマスク
30 ims_sts	割り込みマスクを参照する	JSR	@ims_sts	
時間管理				
31 set_tim	システムクロックを設定する	MOV.W JSR	#pk_time, R1 @set_tim	pktime:設定する時間データへのポインタ
32 get_tim	システムクロックの値を読みだす	MOV.W JSR	#pk_time, R1 @get_tim	pktime:データ読み出しエリアへのポインタ
システム管理				
33 get_ver	ニュークリアスのバージョン識別子を得る	MOV.W JSR	#pkver, R1 @get_ver	pk_ver:バージョン管理ブロック呼出しエリアへのポインタ

タスク管理機能

```
 1 ER ercd=  sta_tsk(ID tskid);                          Start Task
 2 ER ercd= ista_tsk(ID tskid);                          Interrupt Start Task
 3          ext_tsk();                                   Exit Task
 4 ER ercd=  chg_pri(TPRI tskpri);                       Change Task Priority
 5 ER ercd=  get_tid(ID *p_tskid);                       Get Task Identifier
 6 ER ercd=  tsk_sts(UINT *p_tskstat,TPRI *p_tskpri,ID tskid); Get Task Status
```

タスク付属同期管理機能

```
 7 ER ercd=  slp_tsk();                                  Sleep Task
 8 ER ercd=  wai_tsk(TMO tmout);                         Wait For Wakeup Task
 9 ER ercd=  wup_tsk(ID tskid);                          Wakeup Task
10 ER ercd= iwup_tsk(ID tskid);                          Interrupt Wakeup Task
11 ER ercd=  can_wup(INT *p_wupcnt,ID tskid);            Cancel Wakeup Task
```

同期／通信機能

```
12 ER ercd=  set_flg(ID flgid, UINT setptn);             Set Event Flag
13 ER ercd= iset_flg(ID flgid, UINT setptn);             Interrupt Set Event Flag
14 ER ercd=  clr_flg(ID flgid, UINT clrptn);             Clear Event Flag
15 ER ercd=  wai_flg(UINT *p_flgptn,ID flgid,UINT waiptn,UINT wfmode); Wait Event Flag
16 ER ercd=  pol_flg(UINT *p_flgptn,ID flgid,UINT waiptn,UINT wfmode); Poll Event Flag
17 ER ercd=  flg_sts(ID *p_wtskid,UINT *p_flgptn,ID flgid); Get Event Flag Status
18 ER ercd=  sig_sem(ID semid);                          Signal Semaphore
19 ER ercd= isig_sem(ID semid);                          Interrupt Signal  Semaphore
20 ER ercd=  wai_sem(ID semid);                          Wait on Semaphore
21 ER ercd= preq_sem(ID semid);                          Poll and request  Semaphore
22 ER ercd=  sem_sts(ID *pwtskid,UINT *p_semcnt,ID semid); Get Samaphone Status
23 ER ercd=  snd_msg(ID mbxid,T_MSG *pk_msg);            Send Message to Mailbox
24 ER ercd= isnd_msg(ID mbxid,T_MSG *pk_msg);            Interrupt Send Message to Mailbox
25 ER ercd=  rcv_msg(T_MSG **ppk_msg,ID mbxid);          Receive Message from  Mailbox
26 ER ercd= prcv_msg(T_MSG **ppk_msg,ID mbxid);          Poll and Receive  Message
27 ER ercd=  mbx_sts(ID *p_wtskid,T_MSG **ppk_msg,ID mbxid); Get Mailbox Status
```

割込み管理機能

```
28 ER ercd=  ret_int();(アセンブリ言語からのみ発行可能) Return from Interrupt Handler
29 ER ercd=  chg_ims(CCR imask);                         Change Interrupt Mask Level
30 ER ercd=  ims_sts(CCR *p_imask);                      Get Interrupt Mask Level Status
```

時間管理機能

```
31 ER ercd=  set_tim(T_TIM *pk_time);                    Set Time
32 ER ercd=  get_tim(T_TIM *pk_time);                    Get Time
```

バージョン管理機能

```
33 ER ercd=  get_ver(T_VER *pk_ver);                     Get Version No.
```

1. sta_tsk　タスクを起動する

```
#include "hi8_3x.h"
/*
 *  1.STA_TSK :
 */
ER sta_tsk(ID tskid);

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_3x.h"
/*
 *  2.ISTA_TSK :
 */
ER ista_tsk(ID tskid);

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_3x.h"
/*
 *  3.EXT_TSK
 */
void ext_tsk(void);

void tsk_XX03()
 {
    /* ...          */

    ext_tsk();
}
```

4. chg_pri　自タスク優先度を変更する

```
#include "hi8_3x.h"
 /*
 *  4.CHG_PRI
  */
ER chg_pri(TPRI tskpri);

void tsk_XX04()
  {
 ER ercd;
 TPRI tskpri;

     /* ...          */

    ercd = chg_pri(tskpri);

     /* ...          */
 }
```

5. get_tid　自タスクのIDを得る

```
#include "hi8_3x.h"
 /*
 *  5.GET_TID
  */
ER get_tid(ID *p_tskid);

void tsk_XX05()
  {
 ER ercd;
 ID *p_tskid;

     /* ...          */

    ercd = get_tid(p_tskid);

     /* ...          */
   }
```

6. tsk_sts　タスクの状態を見る

```
#include "hi8_3x.h"
 /*
 *  6.TSK_STS
  */
ER tsk_sts(UINT *p_tskstat,TPRI *p_tskpri,
           ID tskid);

void tsk_XX06()
 {
ER ercd;
UINT *p_tskstat;
TPRI *p_tskpri;
ID tskid;

     /* ...          */

     ercd = tsk_sts(p_tskstat,p_tskpri,tskid);

     /* ...          */
 }
```

7. slp_tsk 自タスクを待ち状態へ移行する

```
#include "hi83x.h"
/*
 *  7.SLPTSK
 */
ER slptsk(void);

void tskXX07()
 {
ER ercd;

    /* ...          */

    ercd = slptsk();

    /* ...          */
  }
```

8. wai_tsk 自タスクを一定時間待ち状態に移行する

```
#include "hi83x.h"
/*
 *  8.WAITSK
 */
ER waitsk(TMO tmout);

void tskXX08()
 {
ER ercd;
TMO tmout;

    /* ...          */

    ercd = waitsk(tmout);

    /* ...          */
  }
```

9. wup_tsk 待ち状態のタスクを起床する

```
#include "hi83x.h"
/*
 *  9.WUPTSK
 */
ER wuptsk(ID tskid);

void tskXX09()
 {
ER ercd;
ID tskid;

    /* ...          */

    ercd = wuptsk(tskid);

    /* ...          */
  }
```

10. iwup_tsk 待ち状態のタスクを起床にする（非タスク部専用）

```
#include "hi83x.h"
/*
 *  10.IWUPTSK :
 */
ER iwuptsk(ID tskid);

void intXX10()
 {
ER ercd;
ID tskid;

    /* ...          */

    ercd = iwuptsk(tskid);

    /* ...          */
}
```

11. can_wup タスクの起床要求を無効にする

```
#include "hi83x.h"
 /*
 *  11.CANWUP
  */
ER canwup(INT *pwupcnt,ID tskid);

void tskXX11()
  {
 ER ercd;
 INT *pwupcnt;
 ID tskid;

     /* ...          */

     ercd = canwup(pwupcnt,tskid);

     /* ...          */
 }
```

12. set_flg イベントフラグをセットする

```
#include "hi83x.h"
 /*
 *
  */
ER setflg(ID flgid,UINT setptn);

void tskXX12()
  {
 ER ercd;
 ID flgid;
 UINT setptn;

     /* ...          */

     ercd = setflg(flgid,setptn);

     /* ...          */
 }
```

13. iset_flg　イベントフラグをセットする（非タスク部専用）

```
#include "hi83x.h"
/*
 *  13.ISETFLG
 */
ER isetflg(ID flgid,UINT setptn);

void intXX13()
 {
ER ercd;
ID flgid;
UINT setptn;

    /* ...          */

    ercd = isetflg(flgid,setptn);

    /* ...          */
}
```

14. clr_flg　ｲﾍﾞﾝﾄﾌﾗｸﾞをクリアする

```
#include "hi83x.h"
/*
 *  14.CLRFLG
 */
ER clrflg(ID flgid,UINT clrptn);

void tskXX14()
 {
ER ercd;
ID flgid;
UINT clrptn;

    /* ...          */

    ercd = clrflg(flgid, clrptn);

    /* ...          */
}
```

15. wai_flg　イベントフラグを待つ

```
#include "hi83x.h"
/*
 *  15.WAIFLG
 */
ER waiflg(UINT *pflgptn,ID flgid,
          UINT waiptn,UINT wfmode);

void tskXX15()
 {
ER ercd;
UINT *pflgptn;
ID flgid;
UINT waiptn;
UINT wfmode;

    /* ...          */

    ercd = waiflg(pflgptn,flgid,
                  waiptn,wfmode);

    /* ...          */
}
```

16. pol_flg　イベントフラグを得る

```
#include "hi83x.h"
/*
 *  16.POLFLG
 */
ER polflg(UINT *pflgptn,ID flgid,
          UINT waiptn,UINT wfmode);

void tskXX16()
 {
ER ercd;
UINT *pflgptn;
ID flgid;
UINT waiptn;
UINT wfmode;

    /* ...          */

    ercd = polflg(pflgptn,flgid,
                  waiptn,wfmode);

    /* ...          */
}
```

17. flg_sts　ｲﾍﾞﾝﾄﾌﾗｸﾞ状態を参照する

```
#include "hi83x.h"
/*
 *  17.FLGSTS
 */
ER flgsts(ID *pwtskid,UINT *pflgptn,
          ID flgid);

void tskXX17()
 {
ER ercd;
ID *pwtskid;
UINT *pflgptn;
ID flgid;

    /* ...          */

    ercd = flgsts(pwtskid,pflgptn,flgid);

    /* ...          */
}
```

18. sig_sem　セマフォに対するＶ操作

```
#include "hi83x.h"
/*
 *  18.SIGSEM
 */
ER sigsem(ID semid);

void tskXX18()
 {
ER ercd;
ID semid;

    /* ...          */

    ercd = sigsem(semid);

    /* ...          */
}
```

19. isig_sem　セマフォに対するＶ操作（非タスク部専用）

```
#include "hi83x.h"
/*
 *  19.ISIGSEM
 */
ER isigsem(ID semid);

void intXX19()
 {
ER ercd;
ID semid;

    /* ...          */

    ercd = isigsem(semid);

    /* ...          */
}
```

20. wai_sem　セマフォに対するＰ操作

```
#include "hi83x.h"
/*
 *  20.WAISEM
 */
ER waisem(ID semid);

void tskXX20()
 {
ER ercd;
ID semid;

    /* ...          */

    ercd = waisem(semid);

    /* ...          */
}
```

21. preq_sem　セマフォ資源を得る

```
#include "hi83x.h"
/*
 *  21.PREQSEM
 */
ER preqsem(ID semid);

void tskXX21()
 {
ER ercd;
ID semid;

    /* ...          */

    ercd = preqsem(semid);

    /* ...          */
}
```

22. sem_sts　セマフォ状態を参照する

```
#include "hi83x.h"
/*
 *  22.SEMSTS
 */
ER semsts(ID *pwtskid,UINT *psemcnt,
          ID semid);

void tskXX22()
 {
ER ercd;
ID *pwtskid;
UINT *psemcnt;
ID semid;

    /* ...          */

    ercd = semsts(pwtskid,psemcnt,semid);

    /* ...          */
}
```

23. snd_msg　メッセージを送信する

```
#include "hi83x.h"
/*
 *  23.SNDMSG
 */
ER sndmsg(ID mbxid,TMSG *pkmsg);

void tskXX23()
 {
ER ercd;
ID mbxid;
TMSG *pkmsg;

    /* ...          */

    ercd = sndmsg(mbxid,pkmsg);

    /* ...          */
}
```

24. isnt_msg　メッセージを送信する（非タスク部専用）

```
#include "hi83x.h"
/*
 *  24.ISNDMSG
 */
ER isndmsg(ID mbxid,TMSG *pkmsg);

void intXX24()
 {
ER ercd;
ID mbxid;
TMSG *pkmsg;

    /* ...          */

    ercd = isndmsg(mbxid,pkmsg);

    /* ...          */
}
```

25. rcv_msg　メッセージの受信を待つ

```
#include "hi83x.h"
/*
 *  25.RCVMSG
 */
ER rcvmsg(TMSG **ppkmsg,ID mbxid);

void tskXX25()
 {
ER ercd;
TMSG **ppkmsg;
ID mbxid;

    /* ...          */

    ercd = rcvmsg(ppkmsg,mbxid);

    /* ...          */
}
```

26. prcv_msg　メッセージを受信する

```
#include "hi83x.h"
/*
 *  26.PRCVMSG
 */
ER prcvmsg(TMSG **ppkmsg,ID mbxid);

void tskXX26()
 {
ER ercd;
TMSG **ppkmsg;
ID mbxid;

    /* ...          */

    ercd = prcvmsg(ppkmsg, mbxid);

    /* ...          */
}
```

27. mbx_sts　ﾒｲﾝﾎﾞｯｸｽ状態を参照する

```
#include "hi83x.h"
/*
 *  27.MBXSTS
 */
ER mbxsts(ID *pwtskid,TMSG **ppkmsg,
          ID mbxid);

void tskXX27()
 {
ER ercd;
ID *pwtskid;
TMSG **ppkmsg;
ID mbxid;

    /* ...          */

   ercd = mbxsts(pwtskid,ppkmsg,mbxid);  }

    /* ...          */
}
```

28. chg_ims　割込みマスクを変更する

```
#include "hi83x.h"
 /*
  *  28.CHGIMS
  */
ER chgims(CCR imask);

 void tskXX28()
  {
  ER ercd;
 CCR imask;

      /* ...          */

      ercd = chgims(imask);

      /* ...          */
  }
```

29. ims_sts　割込みマスクを参照する

```
#include "hi83x.h"
 /*
  *  29.IMSSTS
  */
ER imssts(CCR *pimask);

 void tskXX29()
  {
  ER ercd;
 CCR *pimask;

      /* ...          */

      ercd = imssts(pimask);

      /* ...          */
  }
```

30. set_tim　ｼｽﾃﾑｸﾛｯｸを設定する

```
#include "hi83x.h"
 /*
  *  30.SETTIM
  */
ER settim(TTIM *pktime);

 void tskXX30()
 {
 ER ercd;
 TTIM *pktime;

   /* ...          */

      ercd = settim(pktime);

      /* ...          */
```

31. get_tim　ｼｽﾃﾑｸﾛｯｸの値を読み出す

```
#include "hi83x.h"
/*
 *  31.GETTIM
 */
ER gettim(TTIM *pktime);

void tskXX31()
 {
ER ercd;
TTIM *pktime;

    /* ...          */

    ercd = gettim(pktime);

    /* ...          */
}
```

32. get_ver　ニュークリアスのバージョン識別子を得る

```
#include "hi83x.h"
/*
 *  32.GETVER
*/
ER getver(TVER *pkver);

void tskXX32()
 {
ER ercd;
TVER *pkver;

    /* ...          */

    ercd = getver(pkver);

    /* ...          */
}
```

付録Ｃ．エラーコード一覧

Ｃ．１　システム異常終了時のエラーコード一覧

エラーコード ニーモニック（ﾚｼﾞｽﾀR0）	エラー情報 （レジスタR1）	説　　明
*1 E_3XFILE　(H'1111)	不正定義テーブルアドレス	セットアップテーブルが不正
E_UNINT　(H'2222)	ベクタ番号	未定義割込み（割込み処理ハンドラが定義されていない）の発生
*2 E_CTXDWN　(H'5555)	H'FFEB(ext_tsk) H'FFBB(ret_int)	コンテキストエラー(ext_tskを非タスク部から発行したか、ret_int をタスク部から発行した)

【注】　*1　ニュークリアス機能モジュールの選択時にセットアップテーブルチェックのモジュールを組み込んでいる場合、チェックを行ないます。

*2　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。

Ｃ．２　Ｈ８－３Ｘシステムコール　エラーコード一覧

エラーコード (ﾆｰﾓﾆｯｸ)	エラーコード 符号付(ercd)	説　　明
E_OK	H'00 (H'00)	正常終了
* E_TNOSPT	H'ED (-H'13)	タイマがサポートされていない（システムタイマが未登録）
* E_PAR	H'DF (-H'21)	パラメータエラー(waiptn=0 、またはwfmode>3)
* E_ILADR	H'DE (-H'22)	不正アドレス(アドレスが０、または奇数)
* E_TPRI	H'DA (-H'26)	不正タスク優先度(-1，0，自タスク優先度以外を指定した)
* E_ILTIME	H'D9 (-H'27)	不正時間指定(tmout≦-2，pk_time で示される値が負)
E_ILMSG	H'D7 (-H'29)	不正メッセージ形式(メッセージ先頭２バイトが０でない)
* E_SELF	H'CF (-H'31)	自タスク指定(tskid= 自タスクID)
* E_NOEXS	H'CC (-H'34)	オブジェクトが存在していない(未登録のID指定)
E_DMT	H'CB (-H'35)	タスクがDORMANT である
E_NODMT	H'CA (-H'36)	タスクがDORMANT でない
* E_OBJ	H'C1 (-H'3F)	タスク優先度が－１のときにWAIT状態に移行しようとした
* E_CTX	H'BB (-H'45)	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'B7 (-H'49)	キューイングのオーバーフロー(セマフォカウンタのオーバーフロー、イベントフラグに待ちタスクが存在している)
E_TMOUT	H'AB (-H'55)	タイムアウト(wai_tskで指定した待ち時間が経過した)
E_PLFAIL	H'A7 (-H'59)	ポーリング失敗(同期通信，資源の獲得を失敗した)

【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。

付録Ｄ．メモリ容量の算出

Ｄ．１　セットアップテーブル容量の算出表

セットアップテーブル容量の算出表です。本算出表によって、テーブル（ＲＯＭ）の容量を求めてください。

内　　訳	計　算　式	容　　量	備　　考
ﾕｰｻﾞ ﾘｾｯﾄﾙｰﾁﾝ先頭ｱﾄﾞﾚｽ	2（固定）	2	
ﾀｲﾏ 初期設定ﾙｰﾁﾝ先頭ｱﾄﾞﾚｽ	2（固定）	2	
ﾀｽｸ 先頭ｱﾄﾞﾚｽ定義ﾃｰﾌﾞﾙ	2 ×ﾀｽｸ 定義数		
ﾀｽｸｽﾀｯｸﾎﾟｲﾝﾀ 定義ﾃｰﾌﾞﾙ	2 ×ﾀｽｸ 定義数		
ﾀｽｸ 定義ﾃｰﾌﾞﾙ	1（固定）	1	
ｲﾍﾞﾝﾄﾌﾗｸﾞ 定義ﾃｰﾌﾞﾙ	1（固定）	1	
ｾﾏﾌｫ定義ﾃｰﾌﾞﾙ	1（固定）	1	
ﾒｲﾙﾎﾞｯｸｽ 定義ﾃｰﾌﾞﾙ	1（固定）	1	
合計			

Ｄ．２　各タスク用スタック領域の算出表

タスクのスタック領域の算出表です。本算出表によって、個々のタスクのスタック容量を求めてください。なお、共有タスクスタック領域として使用する場合、共有するタスクの中で最大のスタックサイズを確保してください。

共有タスクスタック機能を使用する場合は、共有タスクスタック機能の制御領域を必ず確保してください。

内　　訳	計　算　式	容　　量	備　　考
ﾀｽｸ が独自に使用するｽﾀｯｸ領域	－		
ﾚｼﾞｽﾀ退避領域	8または１４		R0～R6を使用する場合、１４バイト必要です
OSが使用するｽﾀｯｸ領域	6（固定）	6	
NMI 用ｽﾀｯｸ領域	4（固定）	4	NMI を使用する場合、４バイト必要です
共有ﾀｽｸｽﾀｯｸ 機能の制御領域	2（固定）	2	共有ﾀｽｸｽﾀｯｸ 機能を用いるとき必要です
合計			

【注】　Ｃインタフェースを使用する場合、スタックを６バイト使用しますので、スタック領域の合計に６バイト追加してください。また、共有タスクスタック機能の制御領域は、タスクスタックポインタの位置から確保してください。

D．3　割込みハンドラ用スタック領域の算出表

割込み処理ハンドラのスタック領域の算出表です。本算出表によって、個々の割込み処理ハンドラのスタック容量を求めてください。

内　　訳	計　算　式	容　　量	備　　考
割込みﾊﾝﾄﾞﾗが独自に使用するｽﾀｯｸ領域	－		
OSが使用するｽﾀｯｸｻｲｽﾞ	１２（固定）	１２	
NMI 用ｽﾀｯｸｻｲｽﾞ	４（固定）	４	NMI を使用する場合、4ﾊﾞｲﾄ必要です
合計			

【注】　Ｃインタフェースを使用する場合、スタックを６バイト使用しますので、スタック領域の合計に６バイト追加してください。

D．4　作業領域の算出表

Ｈ８－３Ｘの作業領域の算出表です。本算出表によって、ＲＡＭの使用量を求めることができます。個々のタスクおよび割込み処理ハンドラのスタック領域の容量は、それぞれの算出表から求めてください。なお、本算出表の割込み処理ハンドラのスタック領域は、Ｈ８／３２５シリーズ用のベクタNo.を記述しています。

内　　訳	計　算　式	容　　量	備　　考
ｼｽﾃﾑ管理ﾃｰﾌﾞﾙ	固定	7	
ﾀｲﾏ 管理ﾌﾞﾛｯｸ	１＋(4×wai_tsk を発行するﾀｽｸ 数)		
ﾀｽｸ 管理ﾌﾞﾛｯｸ	２×ﾀｽｸ 定義数		
ﾀｽｸ SPｾｰﾌﾞ 領域	２×ﾀｽｸ 定義数		
ｲﾍﾞﾝﾄﾌﾗｸﾞ 管理ﾌﾞﾛｯｸ	２×ｲﾍﾞﾝﾄﾌﾗｸﾞ 定義数		
ｾﾏﾌｫ管理ﾌﾞﾛｯｸ	３×ｾﾏﾌｫ定義数		
ﾒｲﾙﾎﾞｯｸｽ 管理ﾌﾞﾛｯｸ	４×ﾒｲﾙﾎﾞｯｸｽ 定義数		
OS用ｽﾀｯｸ領域	１０または１４		NMI を使用する場合、１４ﾊﾞｲﾄ 必要
ﾀｲﾏ 割込み処理用ｽﾀｯｸ領域	１０または１４		NMI を使用する場合、１４ﾊﾞｲﾄ 必要
ﾀｽｸ ID 01 用ｽﾀｯｸ領域	－		
ﾀｽｸ ID 02 用ｽﾀｯｸ領域	－		
ﾀｽｸ ID 03 用ｽﾀｯｸ領域	－		
ﾀｽｸ ID 04 用ｽﾀｯｸ領域	－		
ﾀｽｸ ID 05 用ｽﾀｯｸ領域	－		
ﾀｽｸ ID 06 用ｽﾀｯｸ領域	－		
ﾀｽｸ ID 07 用ｽﾀｯｸ領域	－		
ﾀｽｸ ID 08 用ｽﾀｯｸ領域	－		
ﾀｽｸ ID 09 用ｽﾀｯｸ領域	－		
ﾀｽｸ ID 10 用ｽﾀｯｸ領域	－		
ﾀｽｸ ID 11 用ｽﾀｯｸ領域	－		
ﾀｽｸ ID 12 用ｽﾀｯｸ領域	－		
ﾀｽｸ ID 13 用ｽﾀｯｸ領域	－		
ﾀｽｸ ID 14 用ｽﾀｯｸ領域	－		

内　　訳	計　算　式	容　　量	備　　考
ﾀｽｸ ID 15 用ｽﾀｯｸ領域	−		
ﾀｽｸ ID 16 用ｽﾀｯｸ領域	−		
ﾀｽｸ ID 17 用ｽﾀｯｸ領域	−		
ﾀｽｸ ID 18 用ｽﾀｯｸ領域	−		
ﾀｽｸ ID 19 用ｽﾀｯｸ領域	−		
ﾀｽｸ ID 20 用ｽﾀｯｸ領域	−		
ﾀｽｸ ID 21 用ｽﾀｯｸ領域	−		
ﾀｽｸ ID 22 用ｽﾀｯｸ領域	−		
ﾀｽｸ ID 23 用ｽﾀｯｸ領域	−		
ﾀｽｸ ID 24 用ｽﾀｯｸ領域	−		
ﾀｽｸ ID 25 用ｽﾀｯｸ領域	−		
ﾀｽｸ ID 26 用ｽﾀｯｸ領域	−		
ﾀｽｸ ID 27 用ｽﾀｯｸ領域	−		
ﾀｽｸ ID 28 用ｽﾀｯｸ領域	−		
ﾀｽｸ ID 29 用ｽﾀｯｸ領域	−		
ﾀｽｸ ID 30 用ｽﾀｯｸ領域	−		
ﾀｽｸ ID 31 用ｽﾀｯｸ領域	−		
ﾍﾞｸﾀ No. 03　用ｽﾀｯｸ領域	−		NMI
ﾍﾞｸﾀ No. 04　用ｽﾀｯｸ領域	−		IRQ0
ﾍﾞｸﾀ No. 05　用ｽﾀｯｸ領域	−		IRQ1
ﾍﾞｸﾀ No. 06　用ｽﾀｯｸ領域	−		IRQ2
ﾍﾞｸﾀ No. 07　用ｽﾀｯｸ領域	−		ﾎﾟｰﾄ
ﾍﾞｸﾀ No. 08　用ｽﾀｯｸ領域	−		16ﾋﾞｯﾄﾌﾘｰﾗﾝﾆﾝｸﾞﾀｲﾏ ICI
ﾍﾞｸﾀ No. 09　用ｽﾀｯｸ領域	−		16ﾋﾞｯﾄﾌﾘｰﾗﾝﾆﾝｸﾞﾀｲﾏ OCIA
ﾍﾞｸﾀ No. 10　用ｽﾀｯｸ領域	−		16ﾋﾞｯﾄﾌﾘｰﾗﾝﾆﾝｸﾞﾀｲﾏ OCIB
ﾍﾞｸﾀ No. 11　用ｽﾀｯｸ領域	−		16ﾋﾞｯﾄﾌﾘｰﾗﾝﾆﾝｸﾞﾀｲﾏ FOVI
ﾍﾞｸﾀ No. 12　用ｽﾀｯｸ領域	−		8 ﾋﾞｯﾄﾀｲﾏ0 CMI0A
ﾍﾞｸﾀ No. 13　用ｽﾀｯｸ領域	−		8 ﾋﾞｯﾄﾀｲﾏ0 CMI0B

内　　訳	計　算　式	容　　量	備　　考
ﾍﾞｸﾀ No. 14　用ｽﾀｯｸ領域	−		8 ﾋﾞｯﾄﾀｲﾏ0 OVI0
ﾍﾞｸﾀ No. 15　用ｽﾀｯｸ領域	−		8 ﾋﾞｯﾄﾀｲﾏ1 CMI1A
ﾍﾞｸﾀ No. 16　用ｽﾀｯｸ領域	−		8 ﾋﾞｯﾄﾀｲﾏ1 CMI1B
ﾍﾞｸﾀ No. 17　用ｽﾀｯｸ領域	−		8 ﾋﾞｯﾄﾀｲﾏ1 OVI1
ﾍﾞｸﾀ No. 18　用ｽﾀｯｸ領域	−		ｼﾘｱﾙ･ｺﾐｭﾆｹｰｼｮﾝ･ｲﾝﾀﾌｪｰｽ 0 ERI0
ﾍﾞｸﾀ No. 19　用ｽﾀｯｸ領域	−		ｼﾘｱﾙ･ｺﾐｭﾆｹｰｼｮﾝ･ｲﾝﾀﾌｪｰｽ 0 RXI0
ﾍﾞｸﾀ No. 20　用ｽﾀｯｸ領域	−		ｼﾘｱﾙ･ｺﾐｭﾆｹｰｼｮﾝ･ｲﾝﾀﾌｪｰｽ 0 TXI0
ﾍﾞｸﾀ No. 21　用ｽﾀｯｸ領域	−		ｼﾘｱﾙ･ｺﾐｭﾆｹｰｼｮﾝ･ｲﾝﾀﾌｪｰｽ 1 ERI1
ﾍﾞｸﾀ No. 22　用ｽﾀｯｸ領域	−		ｼﾘｱﾙ･ｺﾐｭﾆｹｰｼｮﾝ･ｲﾝﾀﾌｪｰｽ 1 RXI1
ﾍﾞｸﾀ No. 23　用ｽﾀｯｸ領域	−		ｼﾘｱﾙ･ｺﾐｭﾆｹｰｼｮﾝ･ｲﾝﾀﾌｪｰｽ 1 TXI1
その他（　　）	−		
その他（　　）	−		
その他（　　）	−		
合計			

付録Ｅ．Ｃ構造体一覧

ＨＩ８－３Ｘで使用している構造体と、構造体を使用するシステムコールを以下に示します。

Ｅ．１　T_MSG メイルボックスメッセージ構造体

使用するシステムコール：

```
ercd  snd_msg(ID mbxid, T_MSG *pk_msg);
ercd isnd_msg(ID mbxid, T_MSG *pk_msg);
ercd  rcv_msg(T_MSG **ppk_msg, ID mbxid);
ercd prcv_msg(T_MSG **ppk_msg, ID mbxid);
```

```
typedef struct t_msg {
    UH msghead;         ＯＳ管理エリア
    VB msgcont[1];      メッセージの内容
} T_MSG;
```

Ｅ．２　T_TIM 時間構造体

使用するシステムコール：

```
ercd set_tim(T_TIM *pk_time);
ercd get_tim(T_TIM *pk_time);
```

```
typedef struct t_tim {
    H  utime;           現在の時間データ〈上位〉
    UH ltime;           現在の時間データ〈下位〉
} T_TIM;
```

Ｅ．３　T_VER バージョン構造体

使用するシステムコール：

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

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

付録Ｆ．システム構築の例

Ｆ．１　概　要

ＨＩ８－３Ｘで標準提供しているコンソールドライバを使用して、実際にシステムを構築する例を示します。システムの構築（アセンブル、リンケージ）は、ＵＮＩＸワークステーション上で行ないます。

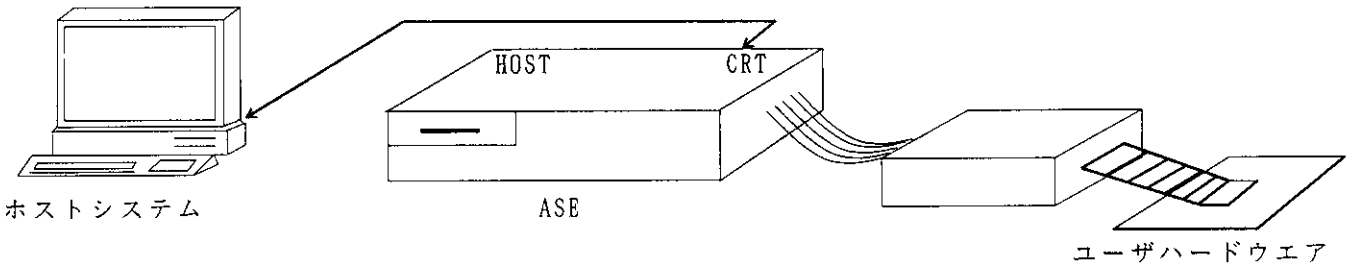
標準提供しているＨ８／３２５用コンソールドライバを使用する場合は、本例を参考にしてシステムを開発してください。

Ｆ．２　システムの構成

Ｆ．２．１　ハードウェア構成

本例では、ホストシステムにＵＮＩＸワークステーション、実機デバック装置にＨ８／３２５ＡＳＥを使用します。

図Ｆ－１にハードウェア構成を示します。



図Ｆ－１　ハードウェア構成

ホストシステム（ＵＮＩＸワークステーション）上に、以下のソフトウェアが必要です。

システムソフトウェア：
　　Ｓｕｎ　ＯＳ　Ver 4.0 以上

関連ユーティリティソフトウェア：
　　①Ｈ８／３００シリーズ　クロスアセンブラ
　　②Ｈ８／３００シリーズ　Ｃコンパイラ
　　③Ｈ８／３００シリーズ　シミュレータデバッガ
　　④Ｈシリーズライブラリアン
　　⑤Ｈシリーズリンケージエディタ
　　⑥オブジェクトコンバータ
　　⑦Ｈシリーズインタフェースソフト

②，③，④は、システム構築の例では、使用していません。

Ｆ．２．２　ソフトウェア構成一覧

ディレクトリ構成を、以下に示します。

ディレクトリ構成	内　　容
/	ルートディレクトリ
/cif	Ｃインタフェースディレクトリ
/cif/include	Ｃインタフェースヘッダファイルディレクトリ
/cns	コンソールディレクトリ
/nuc	ニュークリアスディレクトリ
/nuc/r0_r3	R0～R3までのレジスタ保障ディレクトリ
/nuc/r0_r6	R0～R6までのレジスタ保障ディレクトリ

以下にH8-3Xのソフトウェア構成一覧を示します。

表Ｆ－１　ソフトウェア構成一覧(1/3)

(1) ルートディレクトリ

No.	ファイル名	内　　容
1	h3xlnk	システム結合用バッチファイル
2	h3xlnk.sub	ﾊﾟﾗﾒｰﾀﾁｪｯｸ無し／共有ﾀｽｸｽﾀｯｸ機能無しのサブコマンドファイル
3	h3xlnkc.sub	ﾊﾟﾗﾒｰﾀﾁｪｯｸ有り／共有ﾀｽｸｽﾀｯｸ機能無しのサブコマンドファイル
4	h3xlnkcs.sub	ﾊﾟﾗﾒｰﾀﾁｪｯｸ有り／共有ﾀｽｸｽﾀｯｸ機能有りのサブコマンドファイル
5	h3xlnks.sub	ﾊﾟﾗﾒｰﾀﾁｪｯｸ無し／共有ﾀｽｸｽﾀｯｸ機能有りのサブコマンドファイル

表Ｆ－１　ソフトウェア構成一覧(2/3)

(2) Ｃインタフェースディレクトリ

No.	ファイル名	内　　容
1	hi8_3x_c	アセンブル用MAKEファイル
2	hi8_3x_c	アセンブル用シェルスクリプトファイル
3	hi8_3x_c.i	Ｃインタフェースライブラリ用定義
4	hi8_3x_c.l	リスト出力用MAKEファイル
5	hi8_3x_c.lib	Ｃインタフェースライブラリ
6	h3xisetf.mar	iset_flgＣインタフェースのソースファイル
7	h3xisigs.mar	isig_semＣインタフェースのソースファイル
8	h3xisndm.mar	isnd_msgＣインタフェースのソースファイル
9	h3xistat.mar	ista_tskＣインタフェースのソースファイル
10	h3xiwupt.mar	iwup_tskＣインタフェースのソースファイル
11	h3xprcvm.mar	prcv_msgＣインタフェースのソースファイル
12	h3xpreqs.mar	preq_semＣインタフェースのソースファイル
13	h3x_canw.mar	can_wupＣインタフェースのソースファイル
14	h3x_chgi.mar	chg_imsＣインタフェースのソースファイル
15	h3x_chgp.mar	chg_priＣインタフェースのソースファイル
16	h3x_clrf.mar	clr_flgＣインタフェースのソースファイル
17	h3x_extt.mar	ext_tskＣインタフェースのソースファイル
18	h3x_flgs.mar	flg_stsＣインタフェースのソースファイル
19	h3x_geti.mar	get_tidＣインタフェースのソースファイル
20	h3x_gett.mar	get_timＣインタフェースのソースファイル
21	h3x_getv.mar	get_verＣインタフェースのソースファイル
22	h3x_imss.mar	ims_stsＣインタフェースのソースファイル
23	h3x_mbxs.mar	mbx_stsＣインタフェースのソースファイル
24	h3x_polf.mar	pol_flgＣインタフェースのソースファイル
25	h3x_rcvm.mar	rcv_msgＣインタフェースのソースファイル
26	h3x_sems.mar	sem_stsＣインタフェースのソースファイル
27	h3x_setf.mar	set_flgＣインタフェースのソースファイル
28	h3x_sett.mar	set_timＣインタフェースのソースファイル
29	h3x_sigs.mar	sig_semＣインタフェースのソースファイル
30	h3x_slpt.mar	slp_tskＣインタフェースのソースファイル
31	h3x_sndm.mar	snd_msgＣインタフェースのソースファイル
32	h3x_stat.mar	sta_tskＣインタフェースのソースファイル
33	h3x_tsks.mar	tsk_stsＣインタフェースのソースファイル
34	h3x_waif.mar	wai_flgＣインタフェースのソースファイル
35	h3x_wais.mar	wai_semＣインタフェースのソースファイル
36	h3x_wait.mar	wai_tskＣインタフェースのソースファイル
37	h3x_wupt.mar	wup_tskＣインタフェースのソースファイル
38	h3xisetf.obj	iset_flgＣインタフェースのオブジェクトファイル
39	h3xisigs.obj	isig_semＣインタフェースのオブジェクトファイル
40	h3xisndm.obj	isnd_msgＣインタフェースのオブジェクトファイル
41	h3xistat.obj	ista_tskＣインタフェースのオブジェクトファイル
42	h3xiwupt.obj	iwup_tskＣインタフェースのオブジェクトファイル
43	h3xprcvm.obj	prcv_msgＣインタフェースのオブジェクトファイル
44	h3xpreqs.obj	preq_semＣインタフェースのオブジェクトファイル
45	h3x_canw.obj	can_wupＣインタフェースのオブジェクトファイル
46	h3x_chgi.obj	chg_imsＣインタフェースのオブジェクトファイル
47	h3x_chgp.obj	chg_priＣインタフェースのオブジェクトファイル
48	h3x_clrf.obj	clr_flgＣインタフェースのオブジェクトファイル
49	h3x_extt.obj	ext_tskＣインタフェースのオブジェクトファイル
50	h3x_flgs.obj	flg_stsＣインタフェースのオブジェクトファイル
51	h3x_geti.obj	get_tidＣインタフェースのオブジェクトファイル
52	h3x_gett.obj	get_timＣインタフェースのオブジェクトファイル
53	h3x_getv.obj	get_verＣインタフェースのオブジェクトファイル
54	h3x_imss.obj	ims_stsＣインタフェースのオブジェクトファイル
55	h3x_mbxs.obj	mbx_stsＣインタフェースのオブジェクトファイル
56	h3x_polf.obj	pol_flgＣインタフェースのオブジェクトファイル
57	h3x_rcvm.obj	rcv_msgＣインタフェースのオブジェクトファイル
58	h3x_sems.obj	sem_stsＣインタフェースのオブジェクトファイル
59	h3x_setf.obj	set_flgＣインタフェースのオブジェクトファイル
60	h3x_sett.obj	set_timＣインタフェースのオブジェクトファイル
61	h3x_sigs.obj	sig_semＣインタフェースのオブジェクトファイル
62	h3x_slpt.obj	slp_tskＣインタフェースのオブジェクトファイル
63	h3x_sndm.obj	snd_msgＣインタフェースのオブジェクトファイル
64	h3x_stat.obj	sta_tskＣインタフェースのオブジェクトファイル
65	h3x_tsks.obj	tsk_stsＣインタフェースのオブジェクトファイル
66	h3x_waif.obj	wai_flgＣインタフェースのオブジェクトファイル
67	h3x_wais.obj	wai_semＣインタフェースのオブジェクトファイル
68	h3x_wait.obj	wai_tskＣインタフェースのオブジェクトファイル
69	h3x_wupt.obj	wup_tskＣインタフェースのオブジェクトファイル
70	hi8_3x_c.sub	Ｃインタフェースライブラリ構築用サブコマンドファイル

表Ｆ－１　ソフトウェア構成一覧(3/3)

(3) Ｃインタフェースヘッダファイルディレクトリ

No.	ファイル名	内　　容
1	hi8_3x.h	Ｃヘッダファイル

(4) コンソールディレクトリ

No.	ファイル名	内　　容
1	h3xcns	アセンブル用MAKEファイル
2	h3xcns	アセンブル用シェルスクリプトファイル
3	h3xcns.inc	コンソールドライバ共通処理
4	h3xcns.l	リスト出力用MAKEファイル
5	h3xcns_0.mar	SCI0用コンソールドライバのソースファイル
6	h3xcns_1.mar	SCI1用コンソールドライバのソースファイル
7	h3xcns_0.obj	SCI0用コンソールドライバのオブジェクトファイル
8	h3xcns_1.obj	SCI1用コンソールドライバのオブジェクトファイル

(5) ニュークリアスディレクトリ

No.	ファイル名	内　　容
1	h3xcpu	CPU別ファイルのアセンブル用シェルスクリプトファイル
2	h3xequ.i	equate定義インクルードファイル
3	h3ximp.i	import宣言インクルードファイル
4	h3xmac.i	macro定義インクルードファイル
5	h3xtim25.i	H8/325用タイマ定義インクルードファイル
6	h3xtim30.i	H8/330用タイマ定義インクルードファイル
7	h3xtim.inc	HI8-3Xタイマモジュールファイル
8	h3xcpu25.mak	H8/325用ファイルのアセンブル用MAKEファイル
9	h3xcpu30.mak	H8/330用ファイルのアセンブル用MAKEファイル
10	h3xcp25.mak	H8/325用ファイルのアセンブルリスト出力用MAKEファイル
11	h3xcp30.mak	H8/330用ファイルのアセンブルリスト出力用MAKEファイル
12	h3xtim25.mar	H8/325用タイマ処理
13	h3xtim30.mar	H8/330用タイマ処理
14	h3xsetup.src	セットアップテーブル
15	h3xuni25.src	H8/325用未定義割込み処理
16	h3xuni30.src	H8/330用未定義割込み処理
17	h3xvec25.src	H8/325用ベクタテーブル
18	h3xvec30.src	H8/330用ベクタテーブル

(6) R0～R3までのレジスタ保障ディレクトリ

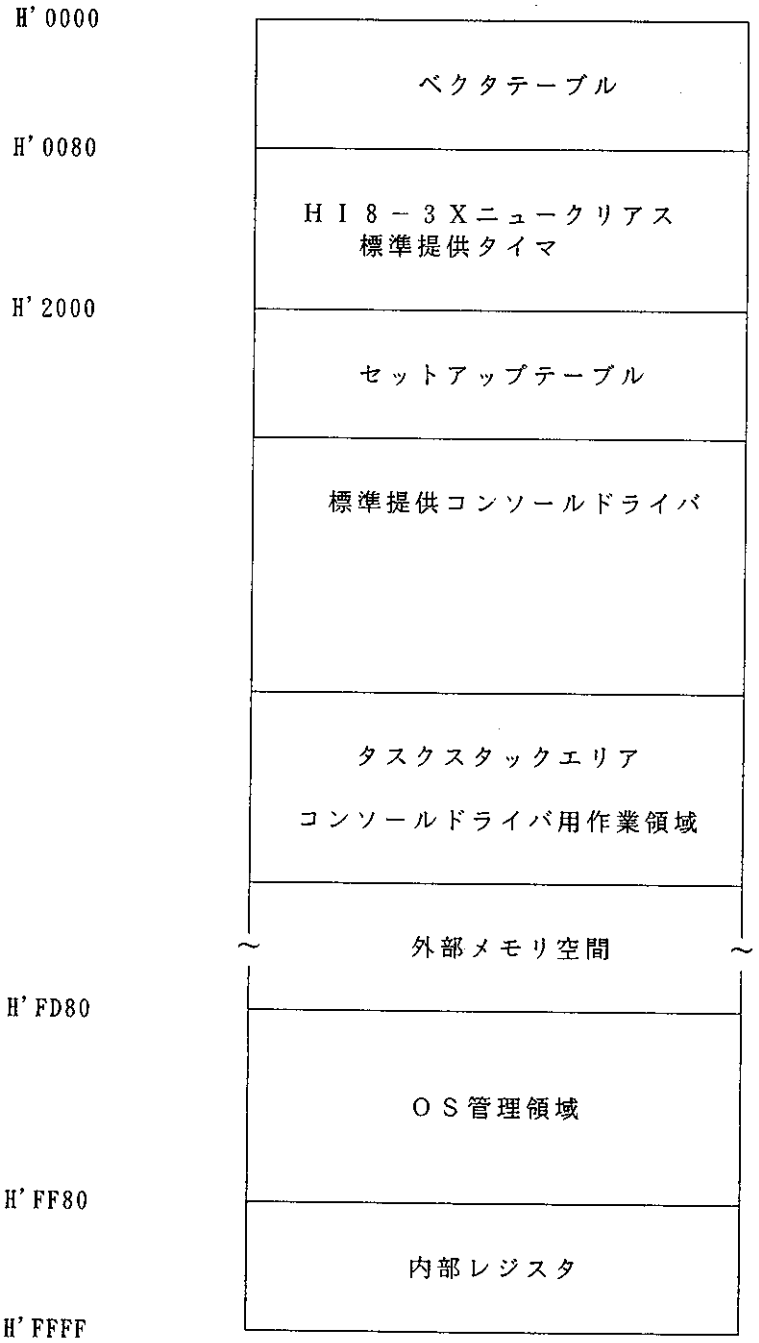
No.	ファイル名	内　　容
1	h3xreg.i	R0～R3までのレジスタ保障を宣言するインクルードファイル
2	h3xncmdl.lib	HI8-3Xモジュールライブラリ（全機能のﾊﾟﾗﾒｰﾀﾁｪｯｸ有り）
3	h3xncstk.lib	HI8-3Xモジュールライブラリ（共有ﾀｽｸｽﾀｯｸ機能のﾊﾟﾗﾒｰﾀﾁｪｯｸ有り）
4	h3xnudmy.lib	HI8-3Xダミーモジュールライブラリ
5	h3xnumdl.lib	HI8-3Xモジュールライブラリ（全機能のﾊﾟﾗﾒｰﾀﾁｪｯｸ無し）
6	h3xnustk.lib	HI8-3Xモジュールライブラリ（共有ﾀｽｸｽﾀｯｸ機能のﾊﾟﾗﾒｰﾀﾁｪｯｸ無し）

(7) R0～R6までのレジスタ保障ディレクトリ

No.	ファイル名	内　　容
1	h3xreg.i	R0～R6までの保障を宣言するインクルードファイル
2	h3xncmdl.lib	H18-3Xモジュールライブラリ（全機能のﾊﾟﾗﾒｰﾀﾁｪｯｸ有り）
3	h3xncstk.lib	H18-3Xモジュールライブラリ（共有ﾀｽｸｽﾀｯｸ機能のﾊﾟﾗﾒｰﾀﾁｪｯｸ有り）
4	h3xnudmy.lib	H18-3Xダミーモジュールライブラリ
5	h3xnumdl.lib	H18-3Xモジュールライブラリ（全機能のﾊﾟﾗﾒｰﾀﾁｪｯｸ無し）
6	h3xnustk.lib	H18-3Xモジュールライブラリ（共有ﾀｽｸｽﾀｯｸ機能のﾊﾟﾗﾒｰﾀﾁｪｯｸ無し）

Ｆ．３　メモリマップ

図Ｆ－２　に標準提供のリンケージエディタのサブコマンドファイル『h3xlnk.sub』を使用した場合のメモリマップを示します。



図Ｆ－２　ＨＩ８－３Ｘメモリマップ

F．4　セットアップテーブル

F．4．1　システム定義テーブル

『H$TIM_INI』に、タイマ初期設定ルーチンを定義しています。

『H$USR_RST』には0を設定し、ユーザリセットルーチンを未登録として定義しています。

F．4．2　タスク定義テーブル

『H$TSK_ADR』のＩＤ＝１に、コンソールドライバの先頭アドレスを定義しています。

ＩＤ＝２～５までは、タスクを未登録として定義しています。

『H$TSK_STK』のＩＤ＝１に、コンソールドライバの初期スタックポインタを定義しています。

ＩＤ＝２～５までは、タスクを未登録にしていますが、それぞれ初期スタックポインタを定義しています。

『H$TSK_CNT』には5を設定し、タスクを５つ定義しています。

F．4．3　イベントフラグ定義テーブル

『H$FLG_CNT』には5を設定し、イベントフラグを５つ定義しています。

F．4．4　セマフォ定義テーブル

『H$SEM_CNT』には5を設定し、セマフォを５つ定義しています。

F．4．5　メイルボックス定義テーブル

『H$MBX_CNT』には5を設定し、メイルボックスを５つ定義しています。

F．4．6　Ｈ１８－３ＸのＲＡＭ領域の定義およびシステム管理テーブル

『H$OS_SP』には、ＯＳ用スタックポインタ・領域を定義しています。

『H$TM_SP』には、タイマ用スタックポインタ・領域，スタックポインタセーブ領域を定義しています。

システム管理テーブル（『H$CUR_SP』，『H$TOP_PRI_ID』，『H$RDYTBL』，『H$SYS_CLK』，『H$TIM_CNT』，『H$TIMCB』，『H$TIMCB_BTM』）を定義しています。

また、タスク用スタック領域『t_stk 』，タスク／イベントフラグ／セマフォ／メイルボックス管理用の制御領域（『H$TCB』，『H$FLGCB』，『H$SEMCB』，『H$MBXCB』）を定義しています。

リストＦ－１　にセットアップテーブルのリストを示します。

```
;3333333333333333333333333333333333333333333333333333333333333333333333333333
;XXX                                                                    XXX
;XXX    H18-3X Version (uITRON V2.01/02)                                XXX
;XXX    H18-3X setup table                    Ver1.00                   XXX
;XXX                                      in Tokyo Japan date:Feb.15.'91 XXX
;XXX                                                                    XXX
;3333333333333333333333333333333333333333333333333333333333333333333333333333
;       .sdata          "H18-3X setup table          V1.00 "
;       .sdata          "Copyright (c)  Hitachi, Ltd. 1991."
;       .sdata          "Licensed Material of Hitachi, Ltd."
;|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        .program        h3xsetup
        .heading        "### h3xsetup.src : H18-3X setupfile V1.00 ###"
        .section        h3xsetup,code,align=2
;:%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;:%%%   export  define                                                   %%%
;:%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        .export H$OS_SP         ,H$TM_SP        ,H$CUR_SP
        .export H$TOP_PRI_ID    ,H$TIM_CNT      ,H$TIMCB        ,H$TIMCB_BTM
        .export H$RDYTBL        ,H$SYS_CLK
        .export H$TCB           ,H$FLGCB        ,H$SEMCB        ,H$MBXCB
        .export H$TSKSPTBL
        .export H$TIM_INI       ,H$USR_RST      ,H$TSK_ADR      ,H$TSK_STK
        .export H$TSK_CNT       ,H$FLG_CNT      ,H$SEM_CNT      ,H$MBX_CNT
;
        .export H_setup_error,H_unint_error,H_ctx_error
;:%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;:%%%   import  define                                                   %%%
;:%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        .import H_chk_init              ;: check setup table
        .import H_3X_TIM_INI,H_CNSDRV_0
;|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
;|||  H18-3X set up table  |||  System Definition Table                    |||
;|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
;
H$TIM_INI:      .data.w H_3X_TIM_INI                    ←―― タイマ初期設定ルーチンの登録
H$USR_RST:      .data.w 0                               ←―― ユーザリセットルーチン未登録
;
```

リストＦ－１　セットアップテーブル『h3xsetup.src』リスト（１／３）

```
;||||||||||||||||||||||||||||||________________________________________|||
;|||  HI8-3X set up table  |||  TDT  (Task         Definition Table)  |||
;||||||||||||||||||||||||||||||________________________________________|||
DMT             .equ    1              ;: dormant status (initial task status)
;
tsk_adr:
H$TSK_ADR:      .equ    tsk_adr-2                                  ←──ﾀｽｸ 先頭ｱﾄﾞﾚｽ定義ﾃｰﾌﾞﾙ
tsk_adr01:      .data.w H_CNSDRV_0          ;: task top address ←ｺﾝｿｰﾙﾄﾞﾗｲﾊﾞ 先頭ｱﾄﾞﾚｽ
tsk_adr02:      .data.w 0                                      ┐
tsk_adr03:      .data.w 0                                      ├──ﾀｽｸ 先頭ｱﾄﾞﾚｽ未登録
tsk_adr04:      .data.w 0                                      │
tsk_adr05:      .data.w 0                                      ┘
tsk_adr_btm:
;
;;;;;;;[ TDT ];;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;[finish!]
;
;:%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;:%%%  equate define                                                    %%%
;:%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
;label          .equ    value   ;[range];: comments
;-----[ count define                  ]-----
TCNT            .equ    (tsk_adr_btm-tsk_adr)/2 ;: task definition size  ←──ﾀｽｸ定義数=5
FLGCNT          .equ    05      ;[0..31];: eventflag  definition size    ←──ｲﾍﾞﾝﾄﾌﾗｸﾞ 定義数=5
SEMCNT          .equ    05      ;[0..31];: semaphore  definition size    ←──ｾﾏﾌｫ定義数=5
MBXCNT          .equ    05      ;[0..31];: mailbox    definition size    ←──ﾒｲﾙﾎﾞｯｸｽ 定義数=5
WAICNT          .equ    (tsk_adr_btm-tsk_adr)/2 ;: wai_tsk use task count ←──wai_tskを使用すﾀｽｸの数
;
tsk_stk:
H$TSK_STK:      .equ    tsk_stk-2                           ←──ﾀｽｸｽﾀｯｸﾎﾟｲﾝﾀ 定義ﾃｰﾌﾞﾙ
tsk_sp01:       .data.w t01_sp          ;: task initial sp   ←──初期ｽﾀｯｸﾎﾟｲﾝﾀ定義
tsk_sp02:       .data.w t02_sp                               ←──初期ｽﾀｯｸﾎﾟｲﾝﾀ定義
tsk_sp03:       .data.w t03_sp                               ←──初期ｽﾀｯｸﾎﾟｲﾝﾀ定義
tsk_sp04:       .data.w t04_sp                               ←──初期ｽﾀｯｸﾎﾟｲﾝﾀ定義
tsk_sp05:       .data.w t05_sp                               ←──初期ｽﾀｯｸﾎﾟｲﾝﾀ定義
;
H$TSK_CNT:      .data.b TCNT                                 ←──ﾀｽｸ 定義ﾃｰﾌﾞﾙ
H$FLG_CNT:      .data.b FLGCNT                               ←──ｲﾍﾞﾝﾄﾌﾗｸﾞ 定義ﾃｰﾌﾞﾙ
H$SEM_CNT:      .data.b SEMCNT                               ←──ｾﾏﾌｫ定義ﾃｰﾌﾞﾙ
H$MBX_CNT:      .data.b MBXCNT                               ←──ﾒｲﾙﾎﾞｯｸｽ 定義ﾃｰﾌﾞﾙ
;
;:<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
;:<<<<<<<<<<<<< Break point, when HI8-3X crashes          <<<<<<<<<<<<<<
;:<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
;
H_setup_error:                  ;: r0=h'1111/r1=illegal table address
H_unint_error:                  ;: r0=h'2222/r1=vector no.
H_ctx_error:                    ;: r0=h'5555/r1=h'ffeb(ext_tsk)|h'ffbb(ret_int)
H_forever_loop: sleep
        bra     H_forever_loop  ;: GOTO H_unint_error
;
```

リストＦ－１　セットアップテーブル『h3xsetup.src』リスト（２／３）

```
;:%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;:%%%   HI8-3X SYSTEM RAM define                                    %%%
;:%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        .section      h3x_ram,data,align=2
os_stk          .res.b  14                                ←OS用ｽﾀｯｸ領域
H$OS_SP:        .equ    $                                 ←OS用ｽﾀｯｸﾎﾟｲﾝﾀ
tm_stk          .res.b  12                                ←ﾀｲﾏ 用ｽﾀｯｸ領域
H$TM_SP:        .res.b  02                                ←ﾀｲﾏ 用ｽﾀｯｸﾎﾟｲﾝﾀ、ｽﾀｯｸﾎﾟｲﾝﾀ
;                                                         ←ｾｰﾌﾞ 領域
t_stk           .res.b  24          ;: task 01 stack      ←ﾀｽｸ 用ｽﾀｯｸ領域(ID=1)
t01_sp:         .equ    $                                 ←ﾀｽｸｽﾀｯｸﾎﾟｲﾝﾀ
                .res.b  02                                ←共用ﾀｽｸｽﾀｯｸ 機能の制御領域
                .res.b  24          ;: task 02 stack      ←ﾀｽｸ 用ｽﾀｯｸ領域(ID=2)
t02_sp:         .equ    $                                 ←ﾀｽｸｽﾀｯｸﾎﾟｲﾝﾀ
                .res.b  02                                ←共用ﾀｽｸｽﾀｯｸ 機能の制御領域
                .res.b  24          ;: task 03 stack      ←ﾀｽｸ 用ｽﾀｯｸ領域(ID=3)
t03_sp:         .equ    $                                 ←ﾀｽｸｽﾀｯｸﾎﾟｲﾝﾀ
                .res.b  02                                ←共用ﾀｽｸｽﾀｯｸ 機能の制御領域
                .res.b  24          ;: task 04 stack      ←ﾀｽｸ 用ｽﾀｯｸ領域(ID=4)
t04_sp:         .equ    $                                 ←ﾀｽｸｽﾀｯｸﾎﾟｲﾝﾀ
                .res.b  02                                ←共用ﾀｽｸｽﾀｯｸ 機能の制御領域
                .res.b  24          ;: task 05 stack      ←ﾀｽｸ 用ｽﾀｯｸ領域(ID=5)
t05_sp:         .equ    $                                 ←ﾀｽｸｽﾀｯｸﾎﾟｲﾝﾀ
                .res.b  02                                ←共用ﾀｽｸｽﾀｯｸ 機能の制御領域
;
sysmt:          .equ    $           ;: SYSMT(SYStem Management Table)      ┐
H$CUR_SP:       .res.b  02          ;: current task sp save table address  │
H$TOP_PRI_ID:   .res.b  01          ;: top priority task id area           │
H$TIM_CNT:      .res.b  01          ;: wai_tsk counter                     ├ｼｽﾃﾑ管理ﾃｰﾌﾞﾙ
H$TIMCB:        .res.b  04*WAICNT   ;: TIMCB(Timer     Control Block)      │
H$TIMCB_BTM:    .equ    $                                                  │
H$RDYTBL:       .res.b  04          ;: ready tcb counter table             │
H$SYS_CLK:      .res.b  04          ;: system clock counter area           ┘
H$TCB:          .equ    $-2                                    ←ﾀｽｸ管理用の制御領域
tcb             .res.b  02*TCNT     ;: TCB  (Task      Control Block)
H$TSKSPTBL:     .equ    $-2                                    ←ﾀｽｸｽﾀｯｸﾎﾟｲﾝﾀｾｰﾌﾞ領域
tsksptbl        .res.b  02*TCNT     ;: task current sp save area
H$FLGCB:        .equ    $-2                                    ←ｲﾍﾞﾝﾄﾌﾗｸﾞ管理用の制御領域
flgcb           .res.b  02*FLGCNT   ;: FLGCB(eventFLaG  Control Block)
H$SEMCB:        .equ    $-3                                    ←ｾﾏﾌｫ管理用の制御領域
semcb           .res.b  03*SEMCNT   ;: SEMCB(SEMaphore  Control Block)
                .align  2
H$MBXCB:        .equ    $-4                                    ←ﾒｲﾙﾎﾞｯｸｽ管理用の制御領域
mbxcb           .res.b  04*MBXCNT   ;: MBXCB(MailBoX    Control Block)
;
ramend          .equ    $           ;: end of OS ram area
;
;
;
                .end
```

リストＦ－１　セットアップテーブル『h3xsetup.src』リスト（３／３）

F．5　割込みベクタテーブル

構築するシステムで、割込みに対し独自の処理ハンドラに制御を移すための割込み処理ハンドラ先頭アドレス登録用テーブルです。

リストF－2　に割込みベクタテーブルのコーディング例を示します。

```
;3333333333333333333333333333333333333333333333333333333333333333333333333333333
;XXX                                                                        XXX
;XXX    H18-3X Version (uITRON V2.01/02)                                    XXX
;XXX    H18-3X/nucleus H8/325 vector table    Ver1.00                       XXX
;XXX                                    in Tokyo Japan date:Feb.15.'91      XXX
;XXX                                                                        XXX
;3333333333333333333333333333333333333333333333333333333333333333333333333333333
;       .sdata          "H18-3X/nucleus H8/325 vector V1.00"
;       .sdata          "Copyright (c)  Hitachi, Ltd. 1991."
;       .sdata          "Licensed Material of Hitachi, Ltd."
;|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        .program        h3xvectr
        .heading        "### h3xvec25.src : for H8/325 ###"
        .section        hi8_3x_v,code,locate=0
;
        .import H_3X_INIT,H_3XINT01,H_3XINT02,H_3XINT03
        .import H_3XINT04,H_3XINT05,H_3XINT06,H_3XINT07
        .import H_3XINT08,H_3XINT09,H_3XINT10,H_3XINT11
        .import H_3XINT12,H_3XINT13,H_3XINT14,H_3XINT15
        .import H_3XINT16,H_3XINT17,H_3XINT18,H_3XINT19
        .import H_3XINT20,H_3XINT21,H_3XINT22,H_3XINT23
;
        .import H_3X_TIM                ;; in 'h3xtim'
        .import H_CNSHDL0ER
        .import H_CNSHDL0RX
        .import H_CNSHDL0TX             ;; in 'h3xcns_0'
;
;*************************************************************************
;*specifications :                                                       *
;*name      = h3xvectr : h8/325 vector table for HI8-3X            ;    *
;*function  = 1. h8 interrupt handler address define for hi8       ;    *
;*          = 2. h8 exception handler address define for hi8       ;    *
;*          = 3. hi8 system standard support module                ;    *
;*          =    (1) reset    : "H_3XINIT"   for power on          ;    *
;*          = 4. hi8 system standard support module                ;    *
;*          =    (1) FRT  ocia: "H_3X_TIM"   for system timer      ;    *
;*          = 5. hi8 system standard support i/o interrupt handler ;    *
;*          =    (1) for console i/o driver [sci use]              ;    *
;*          =             (a) : "H_CNSHDL0ER" for receive error    ;    *
;*          =             (b) : "H_CNSHDL0RX" for receive  (input) ;    *
;*          =             (c) : "H_CNSHDL0TX" for transmit (output);    *
;*date      = 02/01/91                                             ;    *
;*author    = Hitachi, Ltd.                                        ;    *
;*history   = V1.00                                                ;    *
;*attribute = public                                               ;    *
;*class     = unit                                                 ;    *
;*linkage   = h8 vector table top address = h'0000 for HI8-3X      ;    *
;*input     = none                                                 ;    *
;*output    = none                                                 ;    *
;*parameter = @(0,r7) : interrupt before "CCR"                     ;    *
;*          = @(2,r7) : interrupt before "PC"(next mnemonic)       ;    *
;*end of specifications :                                                *
;*************************************************************************
```

リストF－2　割込みベクタテーブル『h3xvec25.src』リスト（1／2）

```
;-------.data.w < address >            ;: h8/325 vector no. contents
        .data.w H_3X_INIT              ;: vector no.00 <reset>
        .data.w H_3XINT01              ;: vector no.01 [reserve]
        .data.w H_3XINT02              ;: vector no.02 [reserve]
        .data.w H_3XINT03              ;: vector no.03 <NMI         >
        .data.w H_3XINT04              ;: vector no.04 <irq0        >
        .data.w H_3XINT05              ;: vector no.05 <irq1        >
        .data.w H_3XINT06              ;: vector no.06 <irq2        >
        .data.w H_3XINT07              ;: vector no.07 <port        >
        .data.w H_3XINT08              ;: vector no.08 <FRT    ici  >
        .data.w H_3X_TIM     ;H_3XINT09 ;: vector no.09 <FRT    ocia >
        .data.w H_3XINT10              ;: vector no.10 <FRT    ocib >
        .data.w H_3XINT11              ;: vector no.11 <FRT    ovi  >
        .data.w H_3XINT12              ;: vector no.12 <8bitT0 cmia >
        .data.w H_3XINT13              ;: vector no.13 <8bitT0 cmib >
        .data.w H_3XINT14              ;: vector no.14 <8bitT0 ovi  >
        .data.w H_3XINT15              ;: vector no.15 <8bitT1 cmia >
        .data.w H_3XINT16              ;: vector no.16 <8bitT1 cmib >
        .data.w H_3XINT17              ;: vector no.17 <8bitT1 ovi  >
        .data.w H_CNSHDL0ER  ;H_3XINT18 ;: vector no.18 <SCI0   eri  >
        .data.w H_CNSHDL0RX  ;H_3XINT19 ;: vector no.19 <SCI0   rxi  >
        .data.w H_CNSHDL0TX  ;H_3XINT20 ;: vector no.20 <SCI0   txi  >
        .data.w H_3XINT21              ;: vector no.21 <SCI1   eri  >
        .data.w H_3XINT22              ;: vector no.22 <SCI1   rxi  >
        .data.w H_3XINT23              ;: vector no.23 <SCI1   txi  >

        .end
```

リストＦ－２　割込みベクタテーブル『h3xvec25.src』リスト（２／２）

F．6　ＨＩ８－３Ｘシステムの構築手順

ＨＩ８－３Ｘシステム(以下ＨＩ８－３Ｘと略します）をシステムに搭載するには、以下の２とおりの方法があります。

(1) ＨＩ８－３Ｘの実行形式ロードモジュールを、Ｈ８／３２５シリーズＡＳＥのロードコマンド(LOAD)を用いて、ターゲットユーザシステム上へロードします。

(2) ＨＩ８－３Ｘの実行形式ロードモジュールを、ＲＯＭ化してシステムに実装します。

F．7　ＨＩ８－３Ｘのロードモジュール生成

提供されているＦＤの中の全ファイルをコピーし、ＨＩ８－３Ｘのロードモジュールを生成します。

ＨＩ８－３Ｘのロードモジュールの生成手順を以下に示します。

① 割込みベクタテーブルのアセンブル
② セットアップテーブルのアセンブル
③ 標準提供コンソールドライバのアセンブル
④ システムの結合（ニュークリアスの機能選択）

以上の作業で、ＨＩ８－３Ｘのロードモジュールが生成されます。
生成されるロードモジュール名は以下のとおりです。

・ＨＩ８－３Ｘの実行形式ロードモジュール：『h3xprg.abs』

F．7．1　割込みベクタテーブルのアセンブル

次のコマンドを実行します。

```
% asm38 △h3xvec25.src△-cpu＝300(RET)
```

この操作で割込みベクタテーブルのオブジェクトファイル『h3xvec25.obj』が生成されます。
割込みベクタテーブルを変更しない場合は、提供している『h3xvec25.obj』を使用してもかまいません。

Ｆ．７．２　セットアップテーブルのアセンブル

次のコマンドを実行します。

```
% asm38 △h3xsetup.src△-cpu＝300(RET)
```

この操作でセットアップテーブルのオブジェクトファイル『h3xsetup.obj』が生成されます。

セットアップテーブルを変更しない場合は、提供している『h3xsetup.obj』を使用してもかまいません。

Ｆ．７．３　標準提供コンソールドライバのアセンブル

標準提供のコンソールドライバのソースファイルをアセンブルすることにより、システムと結合可能なオブジェクトファイル『h3xcns_0.obj』が生成されます。

コンソールドライバを変更しない場合は、提供している『h3xcns_0.obj』を使用してもかまいません。

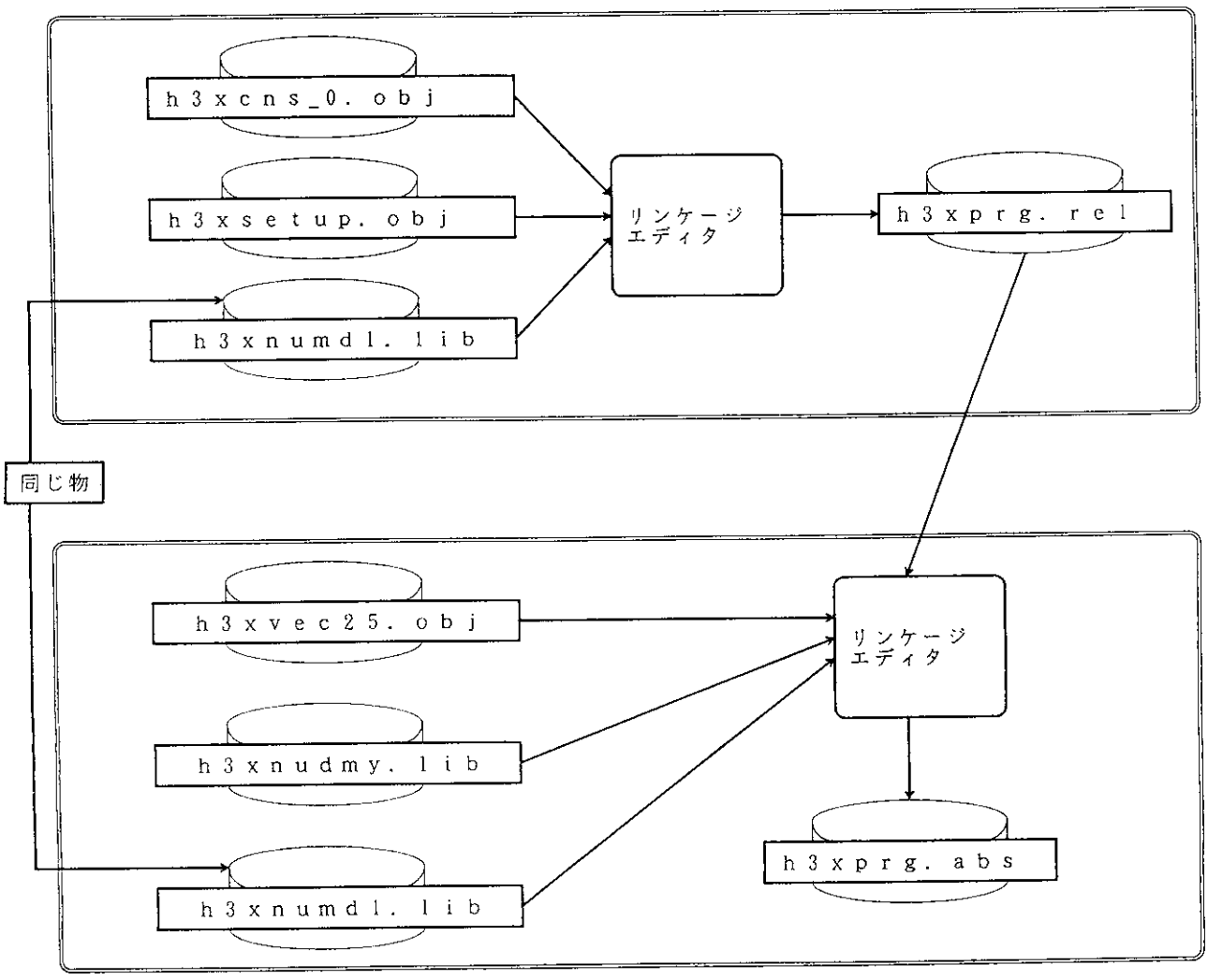
Ｆ．７．４　システムの結合

これまで、個別に作成したオブジェクトファイルまたはライブラリファイルをもとに、システムとしての最終的な実行形式ファイル『h3xprg.abs』を生成します。

必要に応じて、モトローラＳタイプ形式ロードモジュールに変換してください。

システムの結合時、保障するレジスタの数、共有タスクスタック機能およびパラメータチェック機能の有無を選択します。これらの選択は、結合するニュークリアスのライブラリで行ないます。

図Ｆ－３　にシステム結合の概要を示します。



図Ｆ－３　システム結合の概要

次に、Ｈ Ｉ ８ － ３ Ｘシステム結合方法を示します。
以下に示した、リンケージエディタの入力ファイルを用意します。

① 『h3xnumdl.lib』：ニュークリアスの機能モジュールライブラリファイル(R0~R6保障)
② 『h3xnudmy.lib』：ニュークリアスのダミーライブラリファイル(R0~R6保障)
③ 『h3xvec25.obj』：割込みベクタテーブルのオブジェクトファイル
④ 『h3xsetup.obj』：セットアップテーブルのオブジェクトファイル
⑤ 『h3xcns_0.obj』：コンソールドライバのオブジェクトファイル

上記ファイルの準備が完了した後、次のコマンドを実行します。

```
% h3xlnk (RET)
```

この操作によりシェルスクリプトファイル『h3xlnk』が実行され、リンケージエディタのサブコマンドファイル『h3xlnk.sub』の内容にしたがって各ファイルが結合され、パラメータチェックなし・共有スタック機能なしの実行形式モジュール『h3xprg.abs』が生成されます。

リストＦ－３　にパラメータチェックなし・共有スタック機能なしＨ Ｉ ８ － ３ Ｘのリンケージエディタ用のサブコマンドファイル『h3xlnk.sub』リストを示します。

```
input   h3xcns_0,h3xsetup
library h3xnumdl
output  h3xprg.rel
form    r
debug
end

input   h3xvec25,h3xprg.rel
library h3xnudmy,h3xnumdl
output  h3xprg.abs
entry   H_3X_INIT
start   hi8_3x(80),h3xsetup,h3xcns(2000),&
        h3x_ram(0fd80)
debug
print   h3xprg.map
exit
```

リストＦ－３　Ｈ Ｉ ８ － ３ Ｘのリンケージエディタ用のサブコマンドファイル『h3xlnk.sub』

Ｆ．７．５　実行形式ロードモジュールのロード

Ｈ８－３Ｘの実行形式ロードモジュール『h3xprg.abs』を、ＡＳＥのロードコマンド(LOAD)を用いてターゲットシステム上へロードします。

ＡＳＥのコマンドライン上で次のコマンドを実行します。

```
: LOAD△:H3XPRG.ABS (RET)
```

この操作でホストシステムから実行形式ロードモジュール『h3xprg.abs』をターゲットシステムにロードします。

また、実行形式ロードモジュール『h3xprg.abs』をフォーマット変換し、モトローラＳタイプ形式ロードモジュール『h3xprg.mot』をロードすることもできます。

Ｆ．７．６　実行形式ロードモジュールの実装

Ｈ８－３Ｘの実行形式ロードモジュール『h3xprg.abs』をフォーマット変換し、モトローラＳタイプ形式ロードモジュール『h3xprg.mot』を作成します。

次のコマンドを実行します。

```
% cnvs△h3xprg.abs (RET)
```

この操作で生成されたモトローラＳタイプ形式ロードモジュール『h3xprg.mot』をＲＯＭ化し、システムに実装します。

Ｆ．７．７　Ｈ８－３Ｘの起動

システムを起動するには、ＡＳＥのGOコマンドを使用します。

ＡＳＥのコマンドライン上で次のコマンドを実行します。

```
: RESET (RET)
: GO (RET)
```

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

ア　行

カ　行

サ　行

タ　行

ナ　行

ハ　行

マ　行

ヤ　行

ラ　行

ワ　行

アルファベット順・索引

A

C

D

E

F

G

H

I

M

N

O

P

R

S

T

V

W

UNIX HI8-3X
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-702-169