

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ひ申し上げます。

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のですが、誤りが無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、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と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、ディスクリット半導体、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

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すは、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。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を負いません。
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。

H18-3X

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

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。

1. 本資料に記載された製品及び製品の仕様は、予告なく変更されることがあります。
2. 本資料に記載された内容は、正確かつ信頼し得るものであります。ただし、これら掲載された情報、製品または回路の使用に起因する損害または特許権その他権利の侵害に関しては、(株)日立製作所は一切その責任を負いません。
3. 本資料によって第三者または(株)日立製作所の特許権その他権利の実施権を許諾するものではありません。
4. 本資料の一部または全部を当社に無断で転載または複製することを堅くお断りいたします。
5. 日立半導体は、人命にかかわる装置用として特別に開発したものは用意しておりません。ライフサポート関連の医療機器用として日立半導体の採用をお考えのお客様は、当社営業窓口へお客様にてシステム設計上の対策をして頂けるかを是非ご連絡頂きますようお願い致します。

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 μ ITRON^{*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、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。

- MS-DOS版^{*2} H8/300シリーズクロスアセンブラ ユーザーズマニュアル
- MS-DOS版^{*2} H8/300シリーズCコンパイラ ユーザーズマニュアル
- MS-DOS版^{*2} Hシリーズ ライブラリアン ユーザーズマニュアル
- MS-DOS版^{*2} 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 >

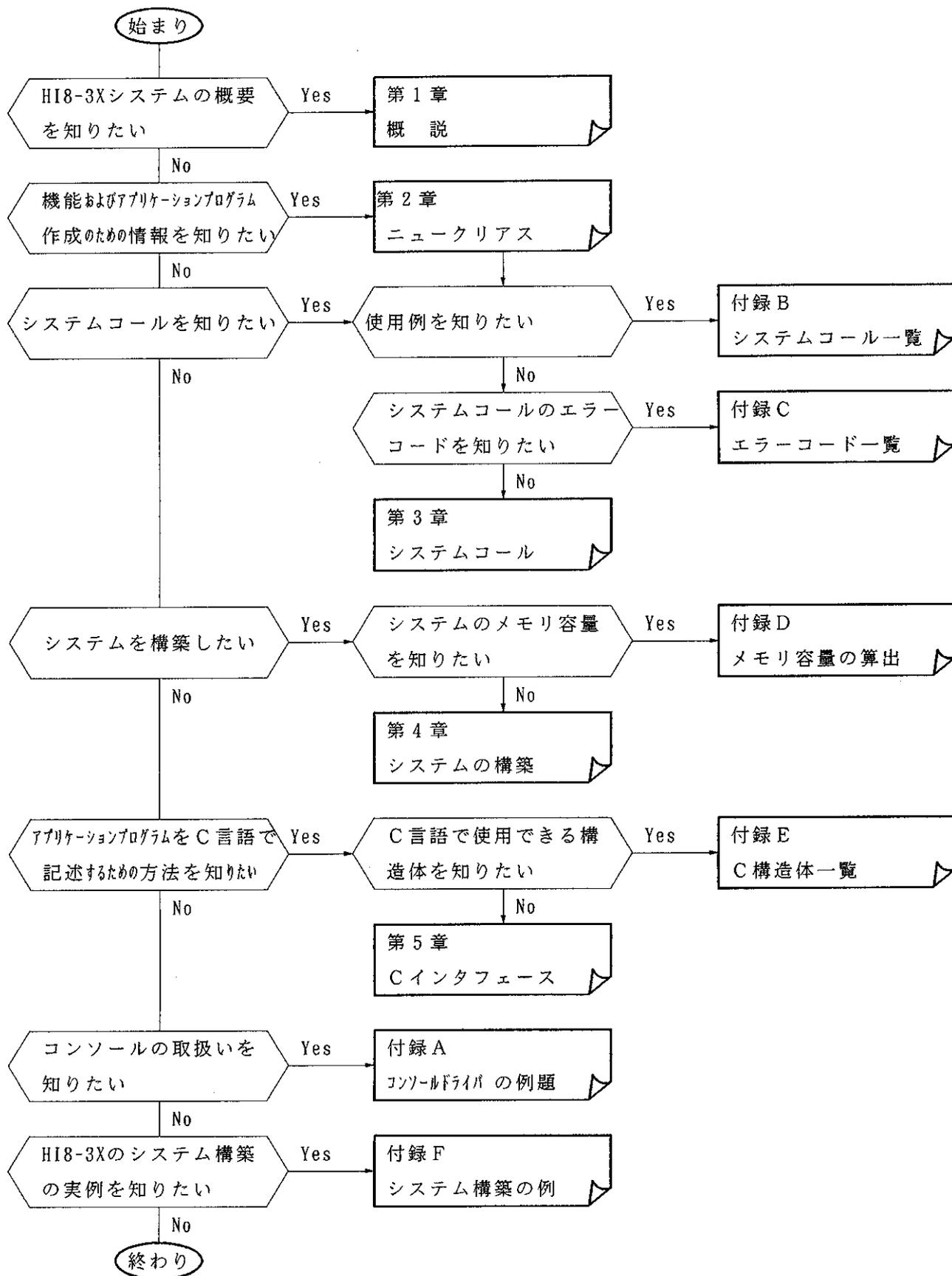
[]	省略できることを意味します。
{ }	いずれかひとつを選択することを意味します。
(RET)	RETURNキーの入力を意味します。
△	1つ以上の空白またはタブコードを意味します。
_____	アンダーラインの部分はユーザがキー入力するコマンドラインです。
< >	この記号で囲まれた内容を指定することを意味します。
..	直前の項目を1回以上指定することを示します。
H'	整数定数の16進数には、先頭に“H”を付けています。デフォルトは、10進数です。

*1 μ ITRON は、“Micro Industrial TRON”の略称です。

TRONは、“The Real time Operating system Nucleus”の略称です。

*2 MS-DOSは、米国マイクロソフト社の登録商標です。

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1章 概説>

1.1	概要	1-1
1.2	特長	1-1
1.3	構成	1-2
1.4	システムの構築手順	1-3

<第2章 ニュークリアス>

2.1	概要	2-1
2.2	タスク管理、タスク付属同期管理	2-3
2.2.1	タスク管理、タスク付属同期管理の概要	2-3
2.2.2	タスクの並列処理	2-4
2.2.3	タスクの状態	2-5
2.2.4	タスクの登録	2-6
2.2.5	タスクの起動	2-7
2.2.6	共有タスクスタック機能によるタスクの起動	2-8
2.2.7	タスクの中断・再開	2-10
2.2.8	タスクの終了	2-11
2.2.9	タスク実行中の割込みのマスク	2-11
2.2.10	タスク作成上の注意	2-12
2.3	同期・通信管理	2-13
2.3.1	同期・通信管理の概要	2-13
2.3.2	イベントフラグ	2-13
2.3.3	セマフォ	2-16
2.3.4	メールボックス	2-19
2.4	時間管理	2-22
2.4.1	時間管理の概要	2-22
2.4.2	ハードウェアタイマとソフトウェアタイマ	2-22
2.4.3	時間の管理	2-23
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)	2-24
2.4.5	タイマ処理ハンドラの登録	2-27

2. 5	割込み管理	2-28
2. 5. 1	割込み管理の概要	2-28
2. 5. 2	割込み処理ハンドラ	2-31
2. 5. 3	未定義割込み	2-32
2. 5. 4	割込み処理ハンドラの注意事項	2-33
2. 6	システム起動処理	2-34
2. 6. 1	システム起動処理の処理概要	2-34
2. 6. 2	ユーザリセットルーチン	2-36
2. 7	システムの異常終了処理	2-36

< 第3章 システムコール >

3. 1	概要	3-1
3. 2	システムコールインタフェース	3-3
3. 2. 1	パラメータ名称の統一的な省略形	3-3
3. 2. 2	エラーコード	3-3
3. 2. 3	アセンブラインタフェース	3-4
3. 2. 4	Cインタフェース	3-5
3. 3	タスク管理システムコール	3-7
3. 3. 1	sta_tsk (Start Task)タスクを起動する〔タスク 部用〕	3-7
	ista_tsk (Interrupt Start Task) タスクを起動する〔非タスク 部用〕	3-7
3. 3. 2	ext_tsk (Exit Task) 自タスクを正常終了する〔タスク 部用〕	3-9
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〕	3-10
3. 3. 4	get_tid (Get Task Identifier) 自タスクのIDを得る〔タスク 部用〕	3-11
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〕	3-12
3. 4	タスク付属・同期システムコール	3-14
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〕	3-14
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〕	3-15
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〕	3-16
	iwup_tsk (Interrupt Wakeup Task) 待ち状態のタスクを起床する 〔非タスク 部用〕	3-16
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〕	3-17
3. 5	同期・通信管理システムコール	3-18
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〕	3-18
	iset_flg (Interrupt Set Event Flag)イベントフラグをセットする 〔非タスク 部用〕	3-18
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〕	3-19
3. 5. 3	wai_flg (Wait Event Flag) イベントフラグを待つ〔タスク 部用〕	3-20
	pol_flg (Poll Event Flag) イベントフラグを得る〔タスク 部用〕	3-20

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〕	3-22
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〕	3-23
	isig_sem (Interrupt Signal Semaphore)セマフォに対する信号操作(V 命令) 〔非タスク 部用〕	3-23
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〕	3-24
	preq_sem (Poll and Request Semaphore)セマフォ資源を得る〔タスク 部用〕	3-24
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〕	3-26
3. 5. 8	snd_msg (Send Message to Mailbox) メッセージを送信する〔タスク 部用〕	3-27
	isnd_msg (Interrupt Send Message to Mailbox) メッセージを送信する 〔非タスク 部用〕	3-27
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〕	3-29
	prcv_msg (Poll and Receive Message)メッセージを受信する〔タスク 部用〕	3-29
3. 5. 10	mbx_sts (Get Mailbox Status)メールボックス状態を参照する 〔タスク/非タスク 部両用〕	3-31
3. 6	割込み管理システムコール	3-32
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〕	3-32
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〕	3-33
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〕	3-34
3. 7	時間管理システムコール	3-35
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〕	3-35
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〕	3-36
3. 8	システム管理システムコール	3-37
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〕	3-37

<第4章 システムの構築>

4. 1	概要	4-1
4. 2	ユーザ・アプリケーションプログラムの作成	4-2
4. 2. 1	タスクの作成	4-2
4. 2. 2	割込み処理ハンドラの作成	4-3
4. 2. 3	ユーザリセットルーチンの作成	4-3

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	4-3
4. 3	システム・アプリケーションプログラムの作成	4-4
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	4-4
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	4-6
4. 4	H I 8 - 3 Xのセットアップファイルの作成	4-7
4. 4. 1	システム定義テーブルの作成	4-8
4. 4. 2	タスク定義テーブルの作成	4-10
4. 4. 3	イベントフラグ定義テーブルの作成	4-11
4. 4. 4	セマフォ定義テーブルの作成	4-12
4. 4. 5	メールボックス定義テーブルの作成	4-13
4. 4. 6	H I 8 - 3 XのRAM領域の定義およびシステム管理テーブルの作成	4-13
4. 4. 7	セットアップファイルのアセンブル	4-14
4. 5	割込みベクタテーブルの作成	4-14
4. 5. 1	割込みベクタテーブルの作成	4-15
4. 5. 2	割込みベクタテーブルのアセンブル	4-18
4. 6	システムの結合	4-19
4. 6. 1	H I 8 - 3 Xシステム結合方法	4-20
4. 6. 2	リンケージの実行	4-22
4. 7	システムの起動	4-23
4. 7. 1	H 8 / 3 0 0 シリーズ A S E によるダウンロード	4-23
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)	4-24
4. 8	システムの異常終了	4-24

< 第5章 Cインタフェース >

5. 1	概要	5-1
5. 2	タスク	5-2
5. 3	ハンドラ	5-3
5. 3. 1	割込みハンドラのコーディング	5-3
5. 3. 2	ユーザリセットルーチンのコーディング	5-5
5. 4	実行形式プログラムの作成	5-6
5. 4. 1	オブジェクトモジュールの作成	5-6
5. 4. 2	オブジェクトモジュールの結合	5-6
5. 5	C言語インタフェースライブラリの作成	5-8

《付録》

<付録A. コンソールドライバの例題>

A. 1	コンソールドライバの概要	A-1
A. 2	ユーザタスクによるコンソール入出力処理	A-1
A. 3	コンソールドライバの機能	A-3
A. 4	メッセージフォーマット	A-3
A. 5	コンソールドライバのequate定義	A-4
A. 6	コンソールドライバの登録	A-4
A. 7	コンソールドライバのアセンブル	A-6
A. 8	コンソールドライバの構成	A-6

<付録B. システムコール一覧>

B. 1	HI8-3Xシステムコール・アセンブラインタフェース一覧	B-1
B. 2	HI8-3Xシステムコール実例集(アセンブラフォーマット)	B-2
B. 3	HI8-3Xシステムコール・Cインタフェース一覧	B-5
B. 4	HI8-3Xシステムコール・実例集(Cフォーマット)	B-6

<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	C-1
C. 2	HI8-3Xシステムコール エラーコード一覧	C-1

<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	D-1
D. 2	各タスク用スタック領域の算出表	D-1
D. 3	割込みハンドラ用スタック領域の算出表	D-2
D. 4	作業領域の算出表	D-3

<付録E. C構造体一覧>

E. 1	T_MSG メールボックスメッセージ構造体	E-1
E. 2	T_TIM 時間構造体	E-1
E. 3	T_VER バージョン構造体	E-1

<付録F. システム構築の例>

F. 1	概要	F-1
F. 2	システムの構成	F-1
F. 2. 1	ハードウェア構成	F-1
F. 2. 2	ソフトウェア構成一覧	F-2
F. 3	メモリマップ	F-6

F. 4	セットアップテーブル	F-7
F. 4. 1	システム定義テーブル	F-7
F. 4. 2	タスク定義テーブル	F-7
F. 4. 3	イベントフラグ定義テーブル	F-7
F. 4. 4	セマフォ定義テーブル	F-7
F. 4. 5	メールボックス定義テーブル	F-7
F. 4. 6	H I 8 - 3 X の R A M 領域の定義およびシステム管理テーブル	F-7
F. 5	割込みベクタテーブル	F-11
F. 6	H I 8 - 3 X システムの構築手順	F-13
F. 7	H I 8 - 3 X のロードモジュール生成	F-13
F. 7. 1	割込みベクタテーブルのアセンブル	F-13
F. 7. 2	セットアップテーブルのアセンブル	F-14
F. 7. 3	標準提供コンソールドライバのアセンブル	F-14
F. 7. 4	システムの結合	F-15
F. 7. 5	実行形式ロードモジュールのロード	F-17
F. 7. 6	実行形式ロードモジュールの実装	F-17
F. 7. 7	H I 8 - 3 X の起動	F-17

<付録G. A S C I Iコード表>

G. 1	A S C I Iコード表	G-1
------	---------------	-----

<付録H. 索引>

H. 1	五十音順・索引	H-1
H. 2	アルファベット順・索引	H-4

目 次

< 第 1 章 概説 >

図 1 - 1	システムの構築手順	1-4
---------	-----------	-----

< 第 2 章 ニュークリアス >

図 2 - 1	HI 8 - 3 X のシステム構成	2-2
図 2 - 2	タスクとニュークリアスの関係	2-3
図 2 - 3	タスクの並列処理の概要	2-4
図 2 - 4	タスクの状態遷移	2-6
図 2 - 5	共有タスクスタック機能の動作	2-8
図 2 - 6	共有タスクスタック機能のタスク間の状態推移	2-9
図 2 - 7	イベントフラグの操作	2-14
図 2 - 8	セマフォによる資源排他制御例	2-17
図 2 - 9	メールボックスの状態	2-19
図 2 - 10	メッセージフォーマット	2-21
図 2 - 11	ハードウェアタイマとソフトウェアタイマ	2-23
図 2 - 12	時間の設定および参照	2-24
図 2 - 13	タイマ処理ハンドラの処理	2-25
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	2-29
図 2 - 15	割込み処理ハンドラの制御移行	2-30
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	2-31
図 2 - 17	システム起動時の処理概要	2-35

< 第 3 章 システムコール >

図 3 - 1	タスクステータスフラグ (tskstat) の構成	3-13
図 3 - 2	送信メッセージの形式	3-28
図 3 - 3	受信メッセージの形式	3-30
図 3 - 4	コンディションコードレジスタ (CCR) の構成	3-33

< 第 4 章 システムの構築 >

図 4 - 1	システムの構築手順の概要	4-2
図 4 - 2	システム定義テーブル	4-9
図 4 - 3	タイマ初期設定ルーチンの登録	4-9
図 4 - 4	ユーザリセットルーチンの登録	4-9
図 4 - 5	タスク定義テーブルのフォーマット	4-11
図 4 - 6	イベントフラグ定義テーブルのフォーマット	4-12
図 4 - 7	セマフォ定義テーブルのフォーマット	4-12
図 4 - 8	メールボックス定義テーブルのフォーマット	4-13
図 4 - 9	割込み処理ハンドラの登録	4-15
図 4 - 10	タイマ割込みリセット処理の登録	4-16
図 4 - 11	システム結合の概要	4-19

< 第 5 章 C インタフェース >

図 5 - 1	C 言語でのタスクの記述例	5-2
図 5 - 2	C 言語での割込みハンドラの記述例	5-4
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	5-7

< 付録 A. コンソールドライバの例題 >

図 A - 1	コンソールドライバ入出力処理の概要	A-1
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	A-3
図 A - 3	コンソールドライバの equate 定義	A-4

< 付録 F. システム構築の例 >

図 F - 1	ハードウェア構成	F-1
図 F - 2	H I 8 - 3 X メモリマップ	F-6
図 F - 3	システム結合の概要	F-15

表 目 次

< 第 1 章 概説 >

表 1 - 1	プログラム容量	1-2
---------	---------	-----

< 第 2 章 ニュークリアス >

表 2 - 1	中断の要因と期間	2-10
表 2 - 2	タイマ初期設定ルーチンの処理条件	2-26
表 2 - 3	割込み処理ハンドラの処理条件	2-32
表 2 - 4	ユーザリセットルーチンの処理条件	2-36
表 2 - 5	システムの異常終了時のレジスタ状態	2-37

< 第 3 章 システムコール >

表 3 - 1	システムコール・発行可能条件一覧	3-2
表 3 - 2	待ちモード(wfmode)のコードと意味	3-21

< 第 4 章 システムの構築 >

表 4 - 1	ライブラリの機能	4-20
---------	----------	------

< 第 5 章 C インタフェース >

表 5 - 1	C 言語インタフェースライブラリファイル一覧	5-1
---------	------------------------	-----

< 付録 A. コンソールドライバの例題 >

表 A - 1	H 8 / 3 2 5 の S C I 割込みの種類	A-5
表 A - 2	サンプルのコンソールドライバの構成	A-6

< 付録 F. システム構築の例 >

表 F - 1	ソフトウェア構成一覧	F-2
---------	------------	-------	-----

リスト目次

< 第 3 章 システムコール >

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	3-5
-----------	----------------------	-----

< 第 4 章 システムの構築 >

リスト 4 - 1	割込みベクタテーブルのコーディング例	4-17
リスト 4 - 2	サブコマンドファイル『H3XLNKCS.SUB』	4-22

< 付録 F システム構築の例 >

リスト F - 1	セットアップテーブル『H3XSETUP.SRC』リスト	F-8
リスト F - 2	割込みベクタテーブル『H3XVEC25.SRC』リスト	F-11
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-16

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ました。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。

HI8-3X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。

HI8-3Xの仕様は、リアルタイムOSの標準仕様である、 μ ITRON仕様に準拠しています。

1. 2 特 長

HI8-3Xには、次のような特長があります。

(1) μ ITRON仕様に準拠

リアルタイムOSの標準仕様である、 μ ITRON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 $27\mu s$ (wup_tsk システムコール使用時の例)、割込み禁止時間 $13\mu s$ 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います。

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カージェ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

HI8-3X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・メー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

H I 8 - 3 X 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核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

H I 8 - 3 X 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

H 8 / 3 0 0 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 H I 8 - 3 X 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 0.4 k バイト
		最大構成	約 4.0 k 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 0.4 k 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

PC-9801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

HI8-3X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、HI8-3X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。

HI8-3X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、PC-9801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。
- 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。

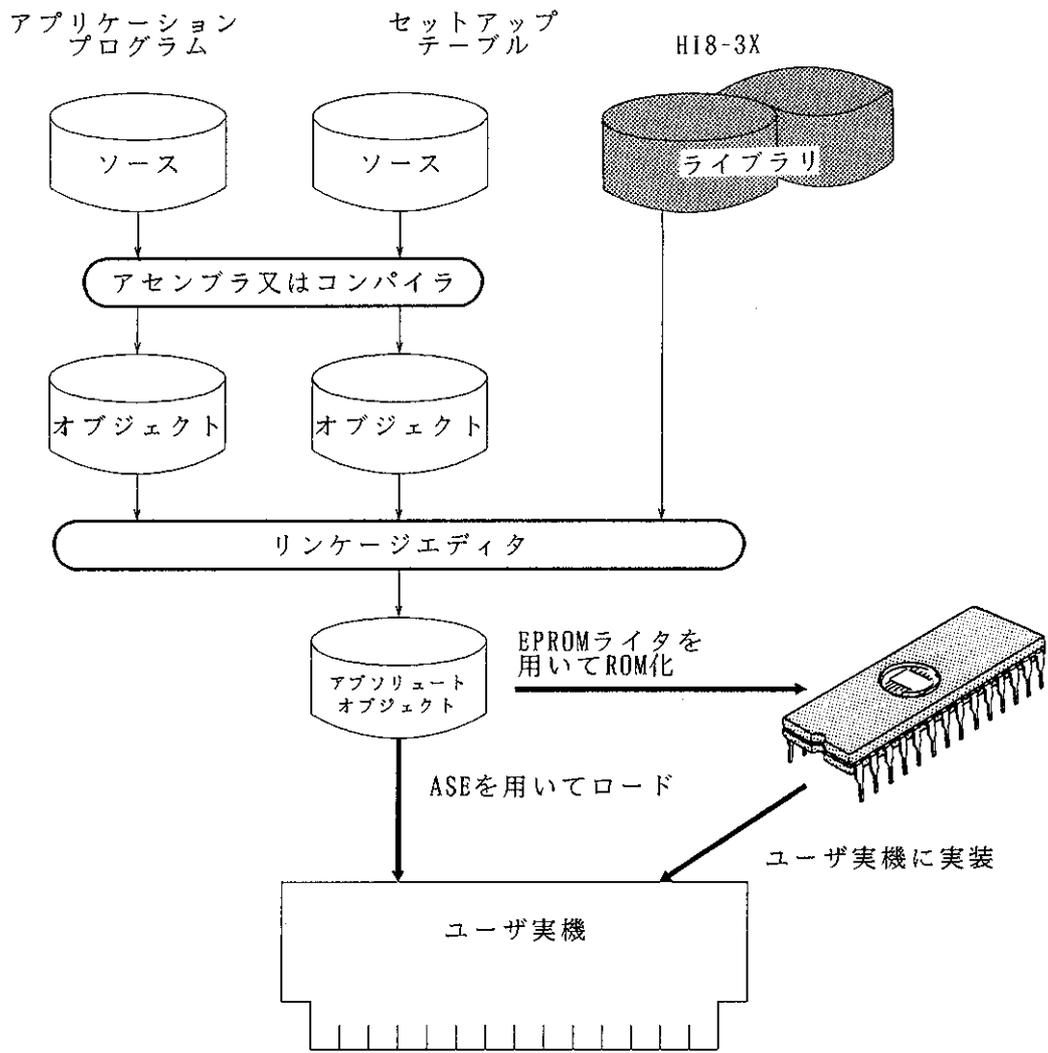


図 1 - 1 システムの構築手順

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。

H I 8 - 3 Xのニュークリアスは、 μ I T R O N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）

C P U (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C P U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、メー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

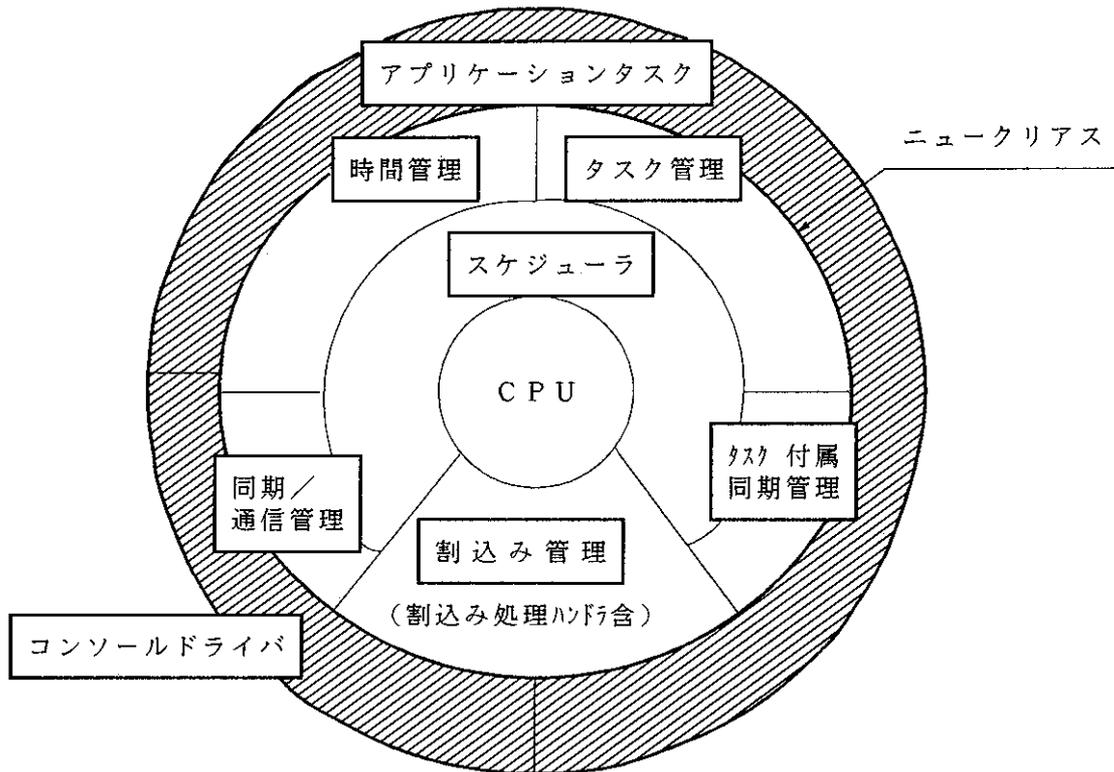
時間の管理を行ないます。また、タスクの実行制御のための時間監視を行ないます。

(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H I 8 - 3 X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 HI8 - 3 X 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

H I 8 - 3 X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。

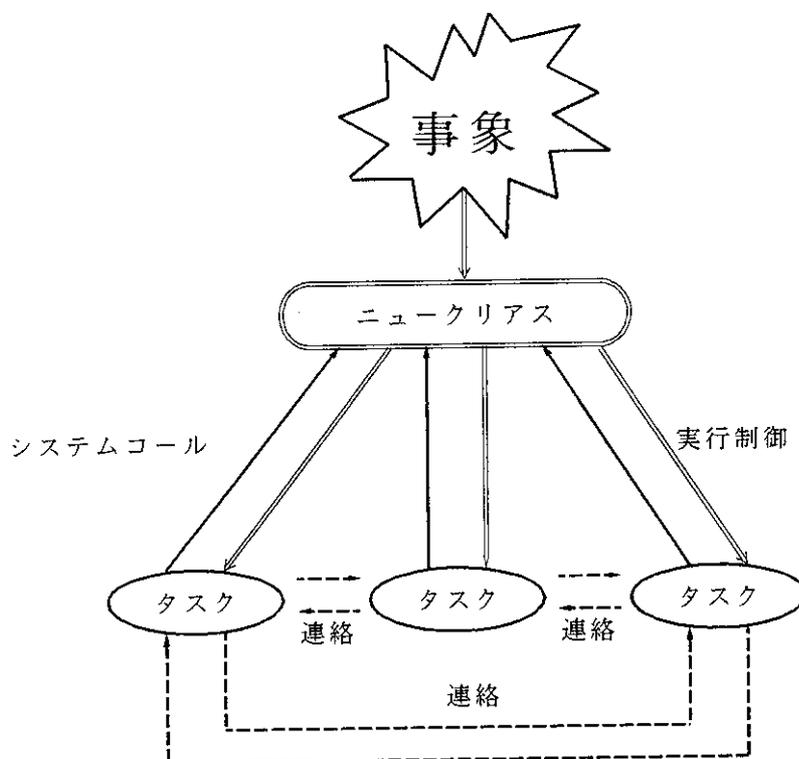


図2-2 タスクとニュークリアスの関係

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。

H18-3X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。

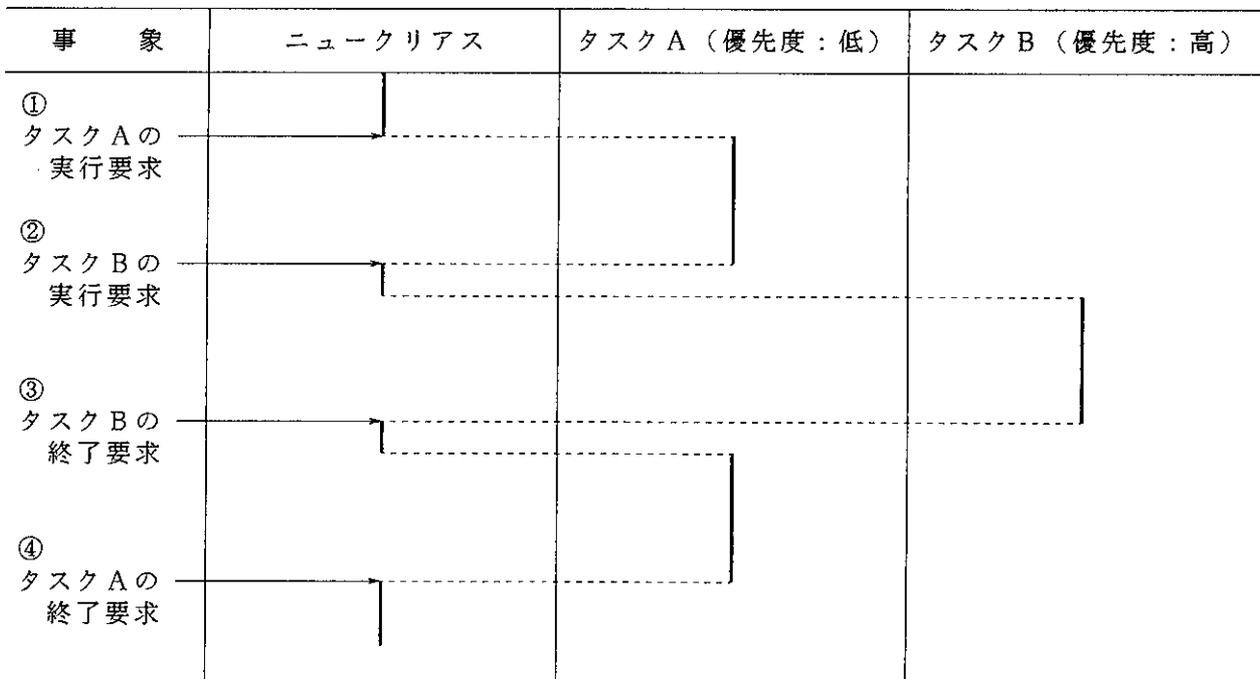


図2-3 タスクの並列処理の概要

(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

H I 8 - 3 X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) D O R M A N T (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) R E A D Y (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) R U N (実行) 状態

C P U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。

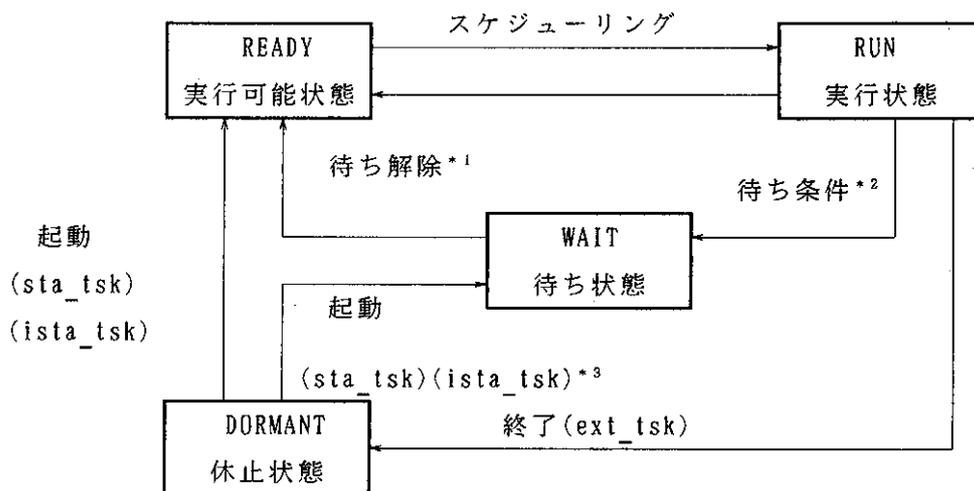
(4) W A I T (待ち) 状態

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

H I 8-3 X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。

H I 8-3 X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、HI 8-3X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
- ・割り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、割り込みマスク解除の状態に移行します。

割り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

HI 8-3 X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。

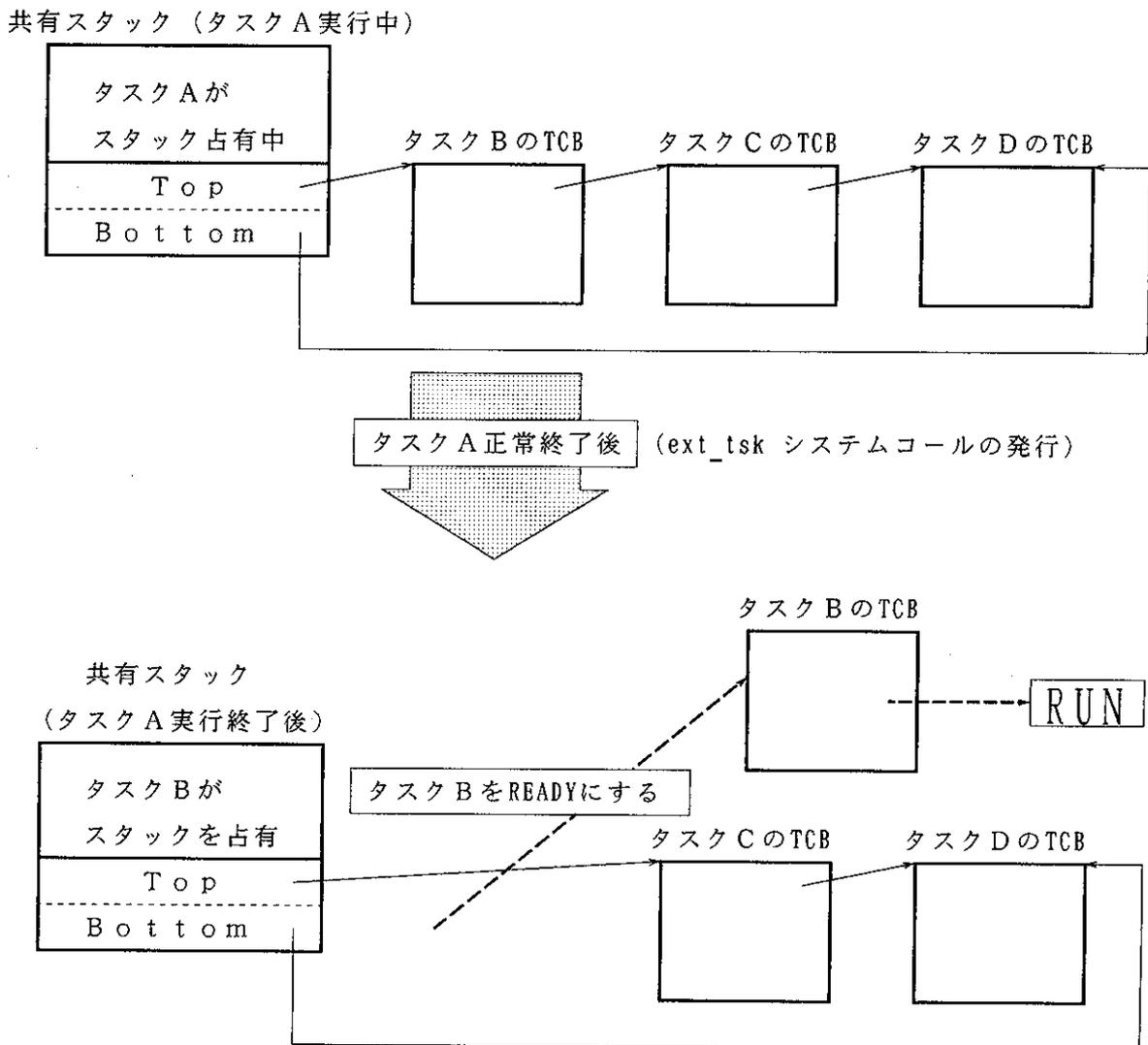


図2-5 共有タスクスタック機能の動作

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。

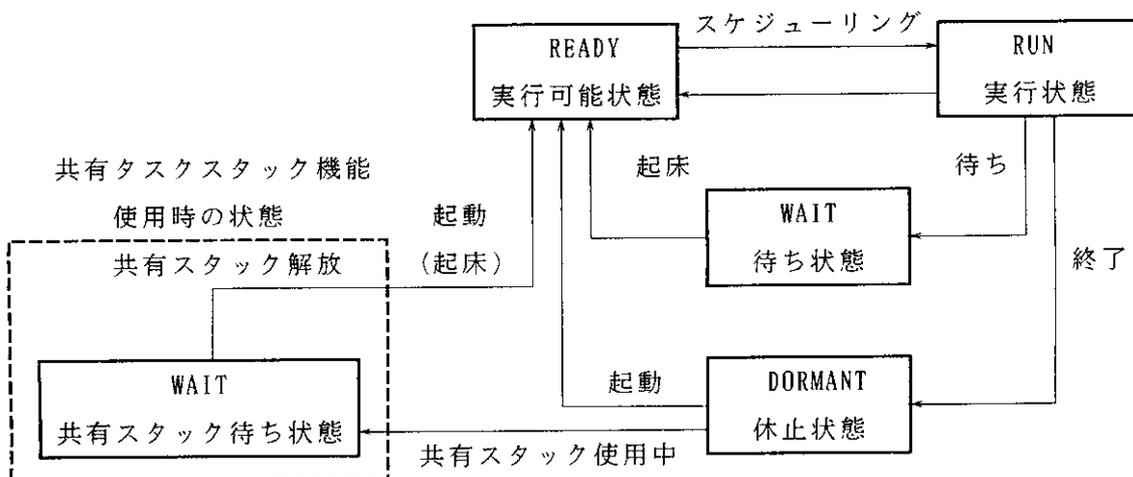


図 2 - 6 共有タスクスタック機能のタスク間の状態推移

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	メー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 H 8 / 3 0 0 のコンディションコードレジスタ(CCR)を変更し、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) 発行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

H I 8 / 3 0 0 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H I 8 - 3 X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

H I 8 - 3 X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H I 8 - 3 Xのライブラリは、タスクでR 0 ~ R 3までのレジスタを保障したものです。

H I 8 - 3 X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R 0 ~ R 3までを保障した場合、レジスタR 4 ~ R 6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R 0 ~ R 6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

H I 8 - 3 X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・メールボックスの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、メー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。

H I 8 - 3 X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。

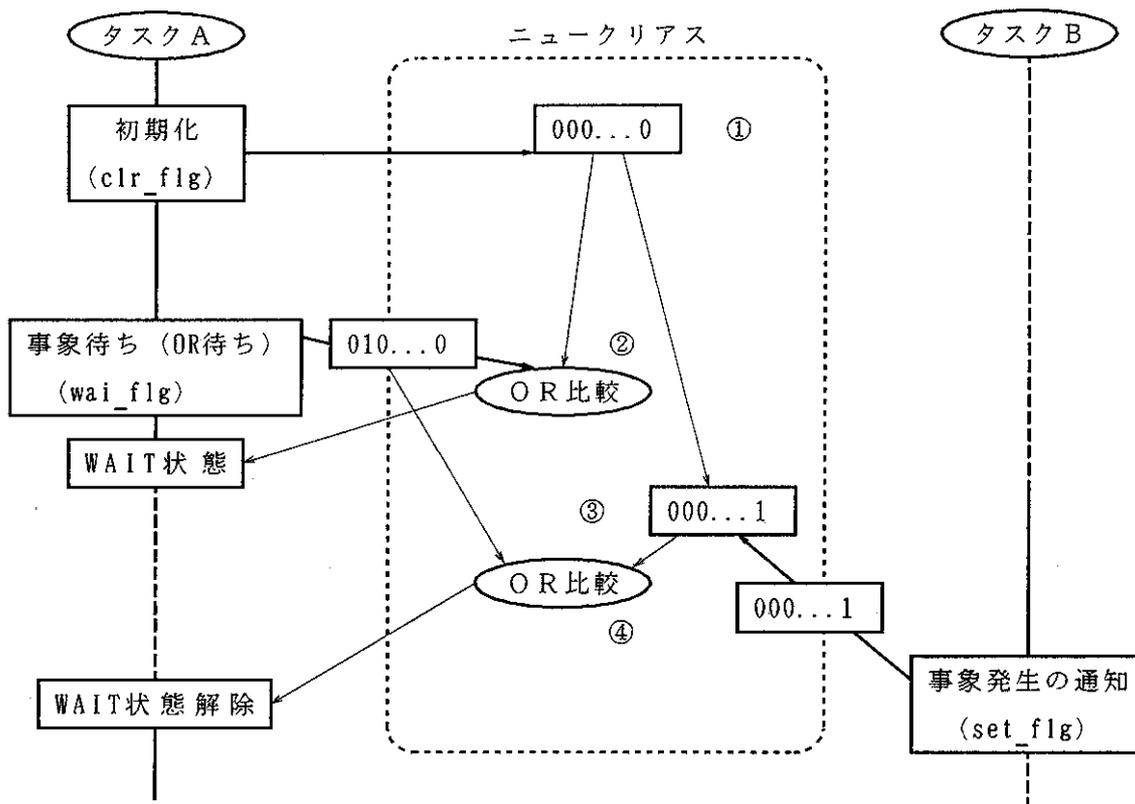


図 2-7 イベントフラグの操作

(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 が事象発生のお知らせを行ない、イベントフラグの内容が変わります。
- 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

事象の発生のお知らせには `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ようになります。

- AND 待ちの解除条件

$(\text{イベントフラグ}) \cap (\text{待ち事象ビットパターン}) = (\text{待ち事象ビットパターン})$

- OR 待ちの解除条件

$(\text{イベントフラグ}) \cap (\text{待ち事象ビットパターン}) \neq 0$

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。

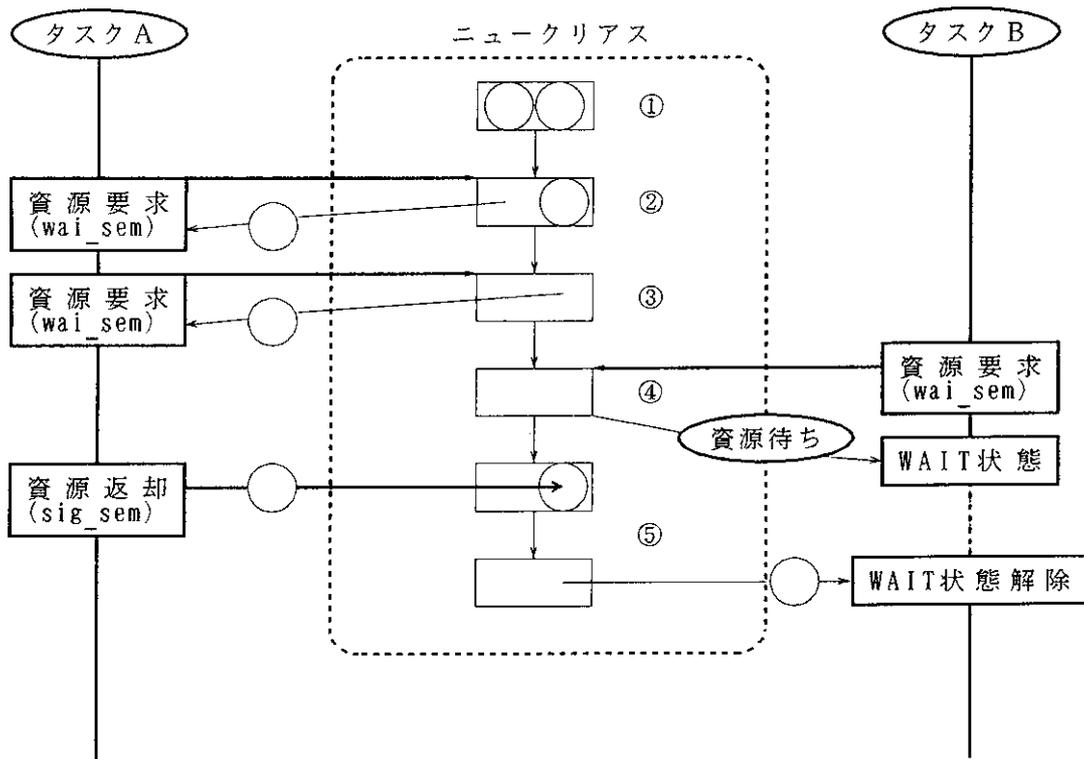


図 2 - 8 セマフォによる資源排他制御例

(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
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、セマフォカウンタ値が最大値 (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タ値
- セマフォの待ち行列先頭タスクのタスク ID

2. 3. 4 メールボックス

メールボックスは、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。メールボックスは、1から31のメールボックス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ールボックスにメッセージを送り、受信するタスクはメールボックスからメッセージを受け取ります。

メー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、メー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せん。

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。

メールボックスは、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(メールボックス状態の参照)

図2-9 にメールボックスの状態を示します。

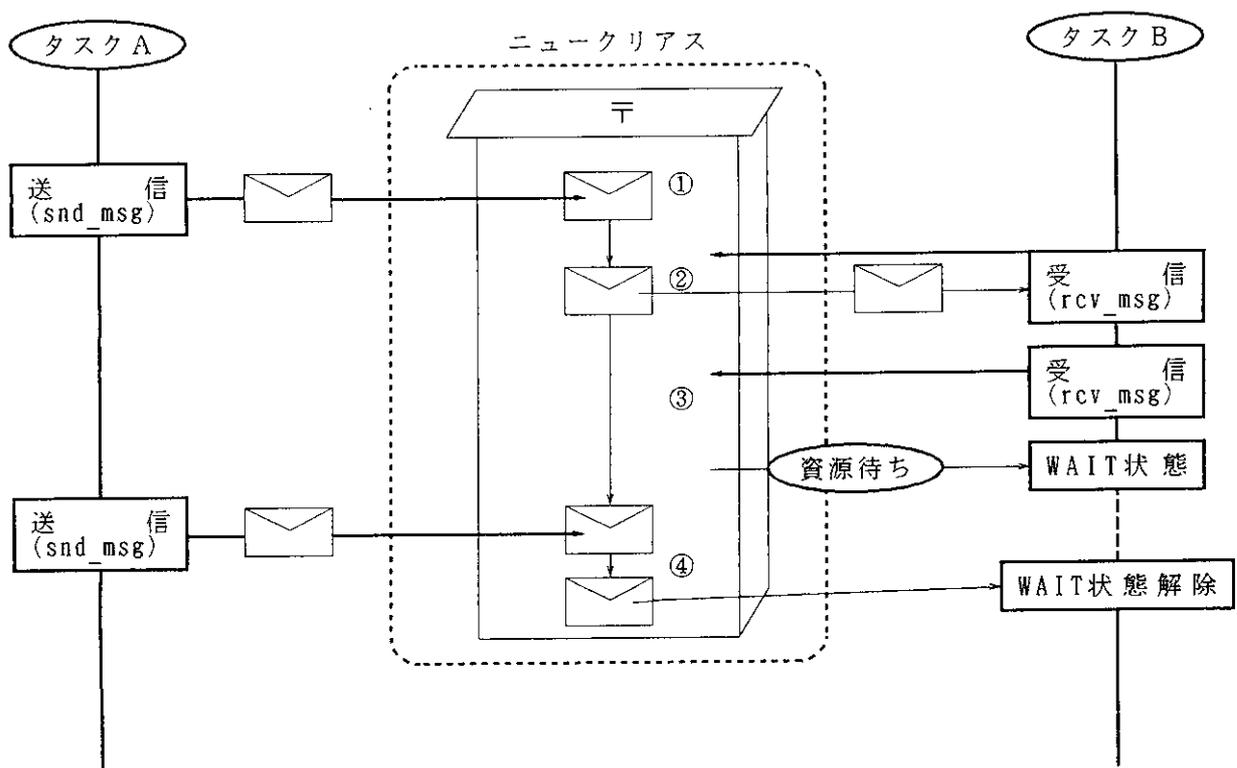


図2-9 メールボックスの状態

- ①タスク A がメッセージを送信したので、メールボックスに 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、メー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

メッセージのメー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、送信するメールボックスの 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、そのメー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ールボックスのメッセージ待ち行列につなぎます。

メー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

メー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、そのメー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、タスクはメールボックスのメッセージ受信待ち行列につながれます。

メー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、メー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) メールボックス状態の参照

現在のメー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 - 1 0 にメッセージのフォーマットを示します。

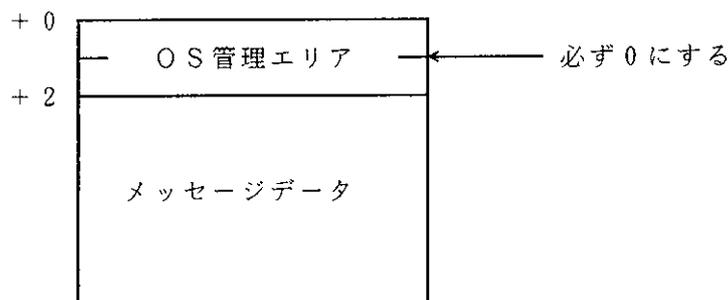


図 2 - 1 0 メッセージフォーマット

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、メッセージがメー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 バイトにメー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となります。

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

HI 8-3 X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

HI 8-3 Xのwai_tsk・set_tim・get_timシステムコールを使用するためには、必ず一定周期で割り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り込みをカウントし、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。

$$\boxed{\text{現実の時間}} = \boxed{\text{システム内の時間}} \times \boxed{tC(\text{ハードウェアタイマ周期時間})}$$

【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。

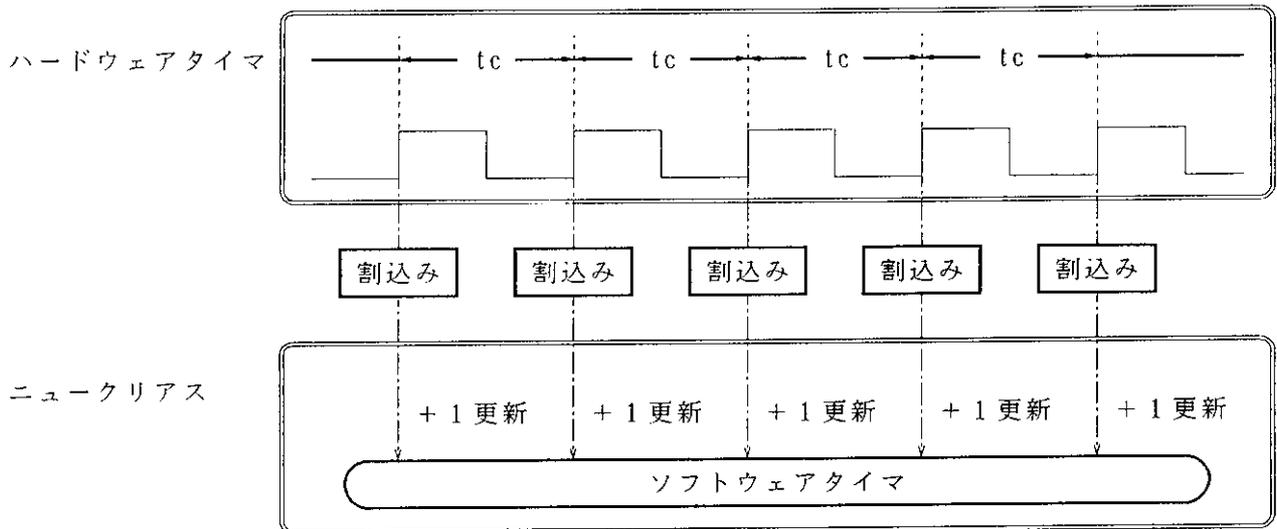


図 2 - 1 1 ハードウェアタイマとソフトウェアタイマ

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り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 - 1 2 に時間の設定および参照を示します。

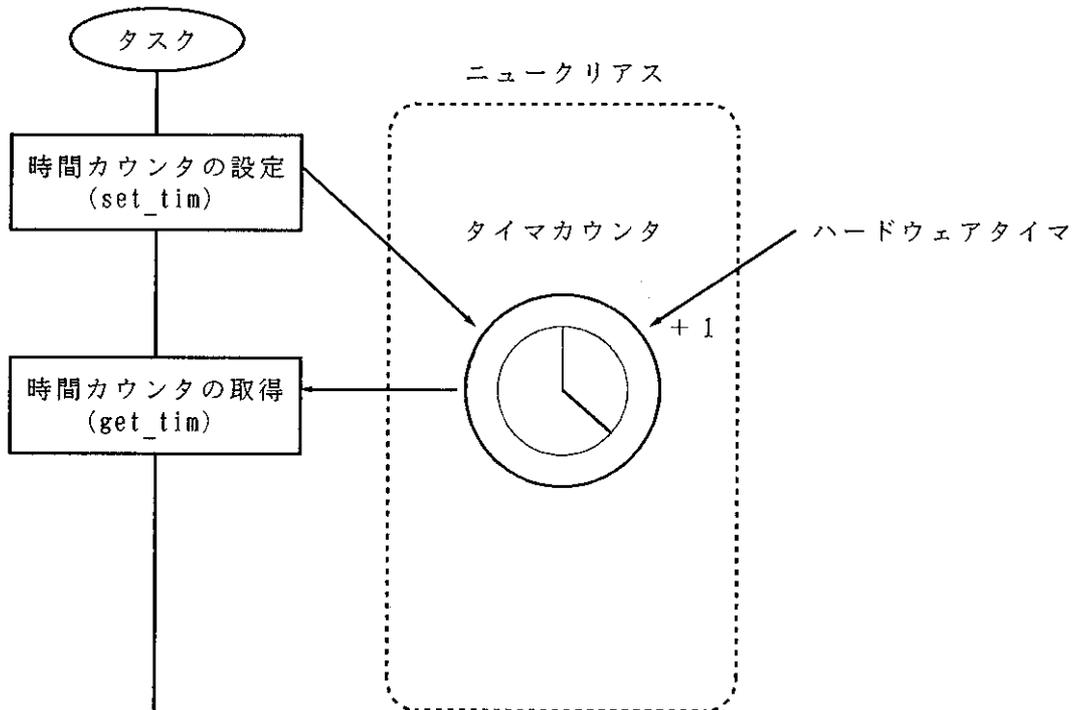


図 2 - 1 2 時間の設定および参照

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 10 msec 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)

H I 8 - 3 X では、標準のハードウェアタイマとして H 8 / 3 0 0 内蔵のフリーランニングタイマ (以下、F R T と略します) を使用します。F R T 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 - 1 3 にタイマ処理ハンドラの処理を示します。

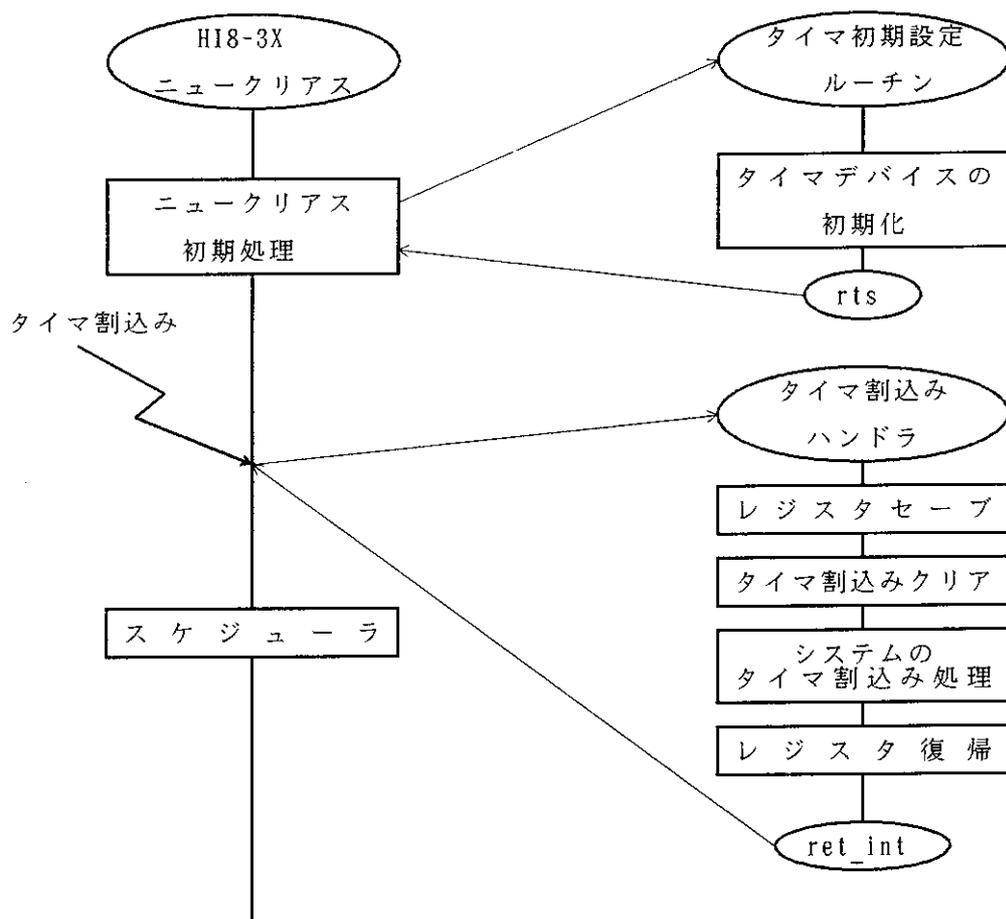


図 2 - 1 3 タイマ処理ハンドラの処理

(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。

H 8 / 3 2 5 内蔵 F R T 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 H I 8 - 3 X 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。

H 8 / 3 2 5 内蔵 F R T 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

H I 8 - 3 X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手元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H I 8 - 3 X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、H 8 / 3 0 0の割込み(Interrupt)です。

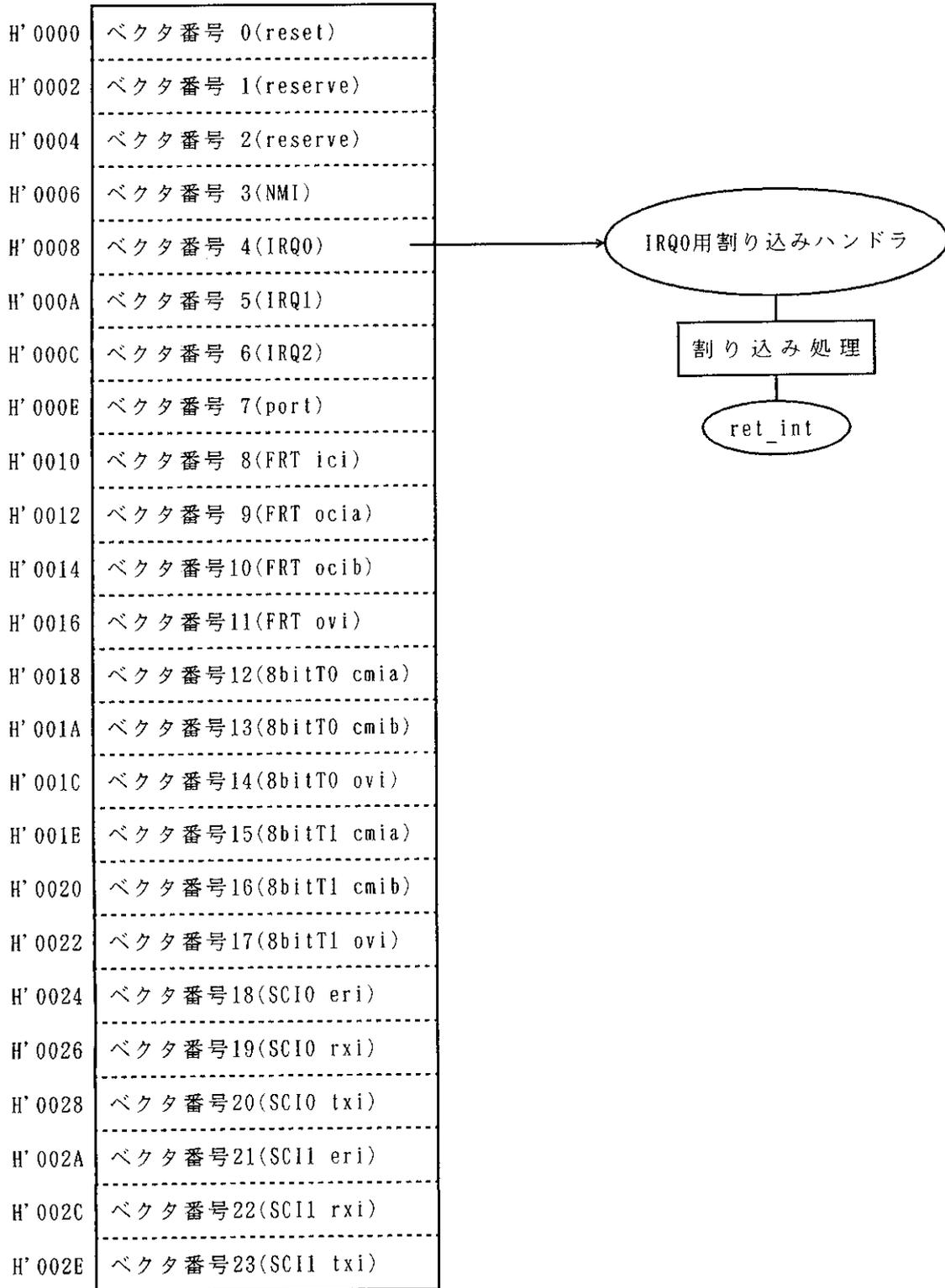
リアルタイム制御システムにおいて割込みは、事象発生を通知する重要な手段です。

(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H 8 / 3 0 0の割込みベクタごとに作成し登録できます。

図 2 - 1 4 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。

割込みベクタテーブル*



(H' 0030)

【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 - 1 4 割込みベクタテーブルと割り込み処理ハンドラとの関連

図 2 - 1 5 に割り込み処理ハンドラの制御移行を示します。

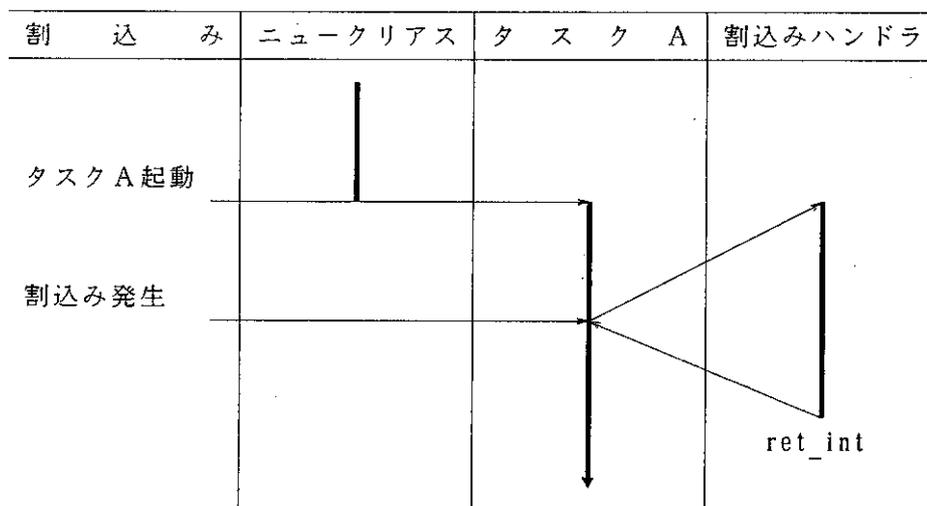


図 2 - 1 5 割り込み処理ハンドラの制御移行

タスク A が処理中、割り込み IRQ0 が発生すると制御は IRQ0 割り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り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り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。

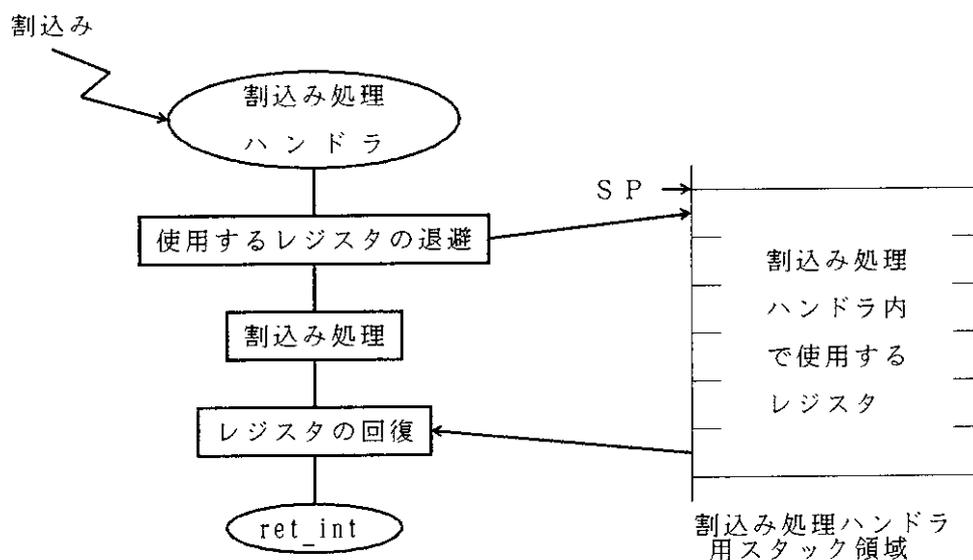


図 2-16 割込み処理ハンドラのレジスタ退避および回復の概要

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

H I 8 - 3 X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、2⁶ビットと2⁴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・メールボックス)
- ・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、システムアイドル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。

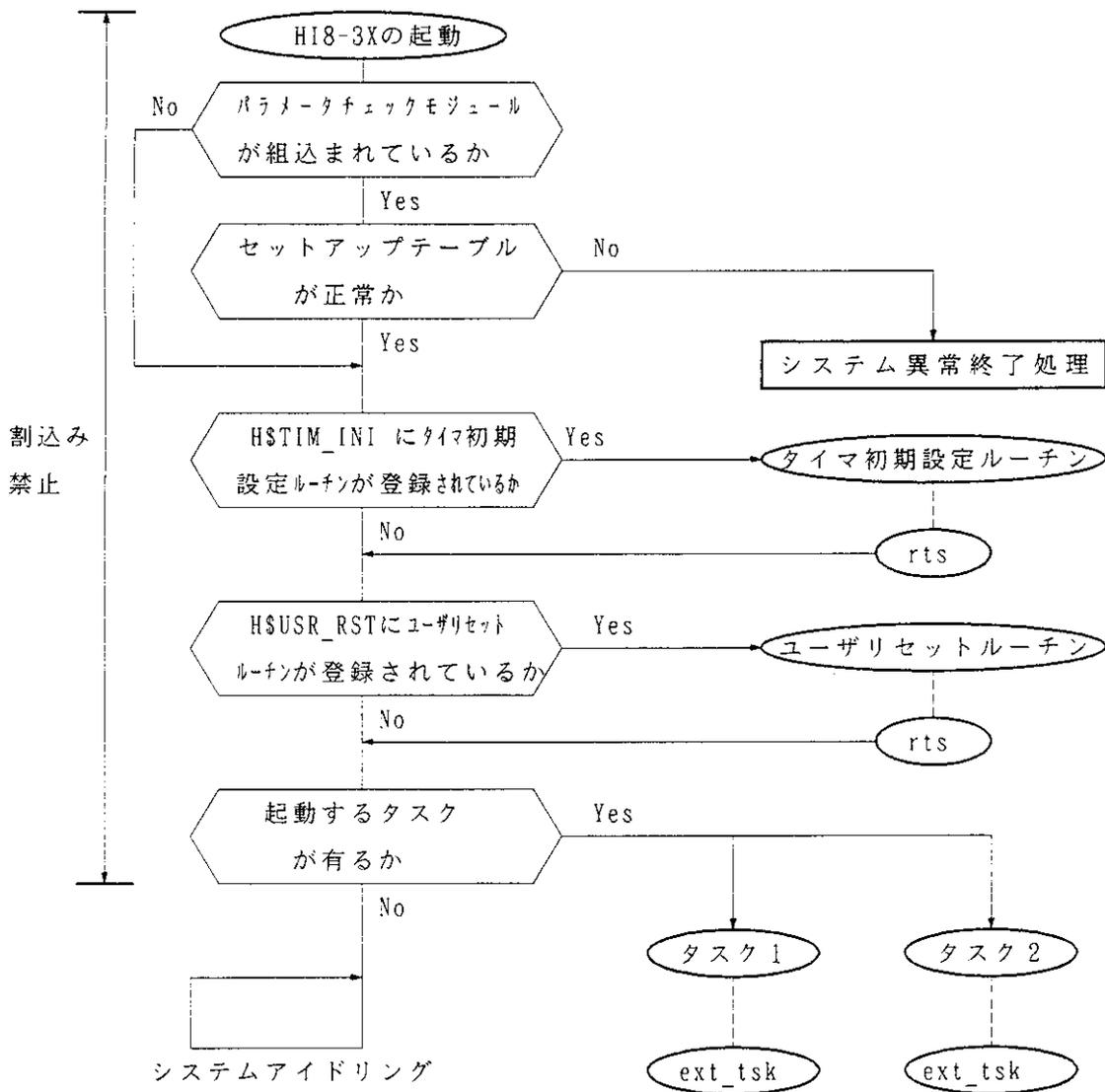


図 2 - 1 7 システム起動時の処理概要

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、H I 8 - 3 X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	・R T S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

H I 8 - 3 X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、H I 8 - 3 X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	異常終了時のジャンプ先アドレスのラベル
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

H I 8 - 3 X は μ I T R O N 仕様に準拠した、33個のシステムコールを提供します。
H I 8 - 3 X 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・メー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・読み込みを行ないます。
- (6) システム管理システムコール
H I 8 - 3 X 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	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	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	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	メールボックス状態を参照する	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	N
時間管理			
31	set_tim	システムクロックを設定する	N
32	get_tim	システムクロックの値を読み出す	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	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

H I 8 - 3 X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。

ROH : レジスタROH(R0の上位 8ビット) が有効

ROL : レジスタROL(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。

ROH : エラーコード

ROL : ID番号

R1 : その他のパラメータ

また、HI8-3X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ROH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、ROH 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 :

ROL タスクID(tskid:0, 1~31) ②

リターンパラメータ :

ROH 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

H I 8 - 3 X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ビット 整数 */  
typedef short    H;      /* 符号付き 16ビット 整数 */  
typedef long     W;      /* 符号付き 32ビット 整数 */  
typedef char     VB;     /* データタイプが一定しないもの(8ビットサイズ) */  
typedef unsigned char UB; /* 符号無し 8ビット 整数 */  
typedef unsigned short UH; /* 符号無し 16ビット 整数 */  
typedef unsigned long UW; /* 符号無し 32ビット 整数 */  
  
typedef void     *VP;    /* データタイプが一定しないものへのポインタ */  
  
typedef UB      ID;     /* オブジェクトのID (xxxid) */  
typedef B       ER;     /* エラーコード - 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ビット 整数(プロセッサのビット幅) */  
typedef UB      UINT;   /* 符号無し 8ビット 整数(プロセッサのビット幅) */
```

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:

```
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=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(ROH/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。

HI8-3X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。

H I 8 - 3 X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:

```
ROH 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 (ROH/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 :

```
ROL    タスクID (tskid:0, 1~31)
```

リターンパラメータ :

```
ROH    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 (ROH/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。

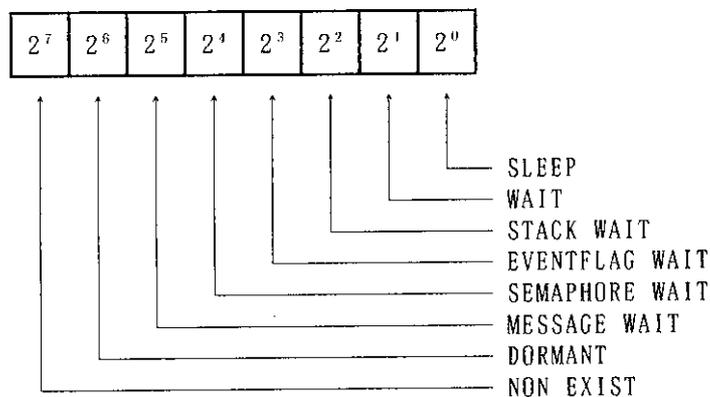


図3-1 タスクステータスフラグ(`tskstat`)の構成

以下に`tskstat`のステータスフラグを示します。

- | | |
|---------------------------|---------------------------------------|
| ビット0 = ON SLEEP | (<code>slp_tsk</code> 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 | (<code>wai_flg</code> によるWAIT(待ち)状態) |
| ビット4 = ON SEMAPHORE WAIT | (<code>wai_sem</code> によるWAIT(待ち)状態) |
| ビット5 = ON MESSAGE WAIT | (<code>rcv_msg</code> 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`と同じ動作を行なわずに、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いません。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ーフロー(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。

HI 8-3 X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:

```
ROL    タスクID (tskid:0, 1~31)
```

リターンパラメータ:

```
ROH    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(ROH/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。

HI 8-3X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 :

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

```
ROH    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(ROH/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となりますが、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 :

```
ROL    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 :

```
ROH    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(ROH/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となりますが、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 (waitpn:H'01~H'FF)
R1L    待ちモード (wfmde: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 waitpn, UINT wfmde);
ER ercd=pol_flg(UINT *p_flgptn, ID flgid, UINT waitpn, UINT wfmde);
```

パラメータ :

```
ID    flgid ; イベントフラグID (flgid:1~31)
UINT  waitpn ; 待ちビットパターン (waitpn:H'01~H'FF)
UINT  wfmde ; 待ちモード (wfmde: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 (waitpn=0 または、wfmde>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(flagptn)が返されません。

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(flagptn)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:

```
ROL    イベントフラグID(flgid:1~31)
```

リターンパラメータ:

```
ROH    エラーコード(ercd)
```

```
R1H    イベントフラグのビットパターン(flgpfn:H'00~H'FF)
```

```
ROL    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(flgpfn:H'00~H'FF)
```

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(flgpfn)・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タ値が1以上であれば、セマフォのカウンタ値を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、セマフォのカウンタ値が0の場合は、セマフォのカウンタ値を変更せず、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タ値が1以上であれば、セマフォのカウンタ値を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、セマフォのカウンタ値が0の場合、セマフォのカウンタ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 :

```
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)
```

```
R1H    セマフォカウント値(semcnt:0~255)
```

```
ROL    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(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))

【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 メールボックス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; メールボックス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)	そのメールボックスは生成されていない (mbxid=0, mbxid>メールボックス定義数(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で示されたメールボックスに、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、対象メー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、メッセージをメー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。

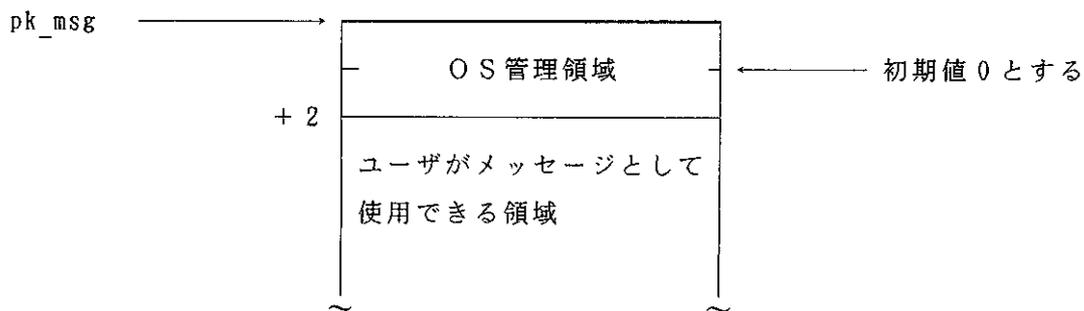


図3-2 送信メッセージの形式

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：

ROL メールボックスID(mbxid:1~31)

リターンパラメータ：

ROH 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; メールボックス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(ROH/ercd)：

E_OK	H'00(-H'00)	正常終了
* E_NOEXS	H'CC(-H'34)	そのメールボックスは生成されていない (mbxid=0,mbxid>メールボックス定義数(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で示されたメー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。

まだそのメー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。

recv_msgシステムコールは、mbxidで示されたメー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、そのメー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。

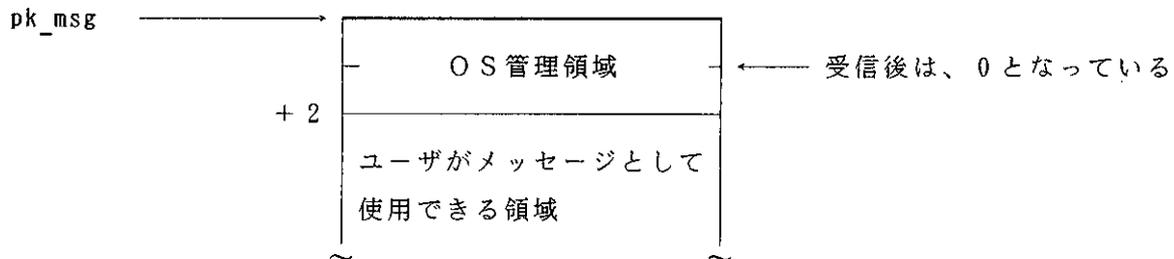


図3-3 受信メッセージの形式

3. 5. 10 mbx_sts (Get Mailbox Status) メー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:

```
ROL    メールボックスID(mbxid:1~31)
```

リターンパラメータ:

```
ROH    エラーコード (ercd)
```

```
RI     次のメッセージの先頭アドレス(pk_msg)
```

```
ROL    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;   メールボックス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(ROH/ercd):

```
E_OK          H'00( H'00)    正常終了
```

```
* E_NOEXS     H'CC(-H'34)    そのメールボックスは生成されていない  
                                     (mbxid=0,mbxid>メールボックス定義数(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 で示されるメールボックスの各種の状態を参照し、対象メー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、対象メー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) :

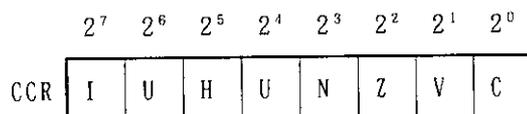
```
E_OK  H'00 (H'00)  正常終了
```

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。



I : 割込みマスクビット

U : ユーザビット(2⁶・2⁴ 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：

```
ROH    エラーコード(ercd)
```

```
RIH    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(ROH/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RIH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) } 32 ビット(H'0~H'7FFFFFFF)
        現在の時間データ <下位> (ltime)
```

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=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(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_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 :

```
ROH    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'FFFFFFF)

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 (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_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(var)
```

```
} T_VER *pk_ver;   バージョン管理ブロック読み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)

(var)

-	LEV	-	M	V	P	-	-	FIL	IO	-	-
---	-----	---	---	---	---	---	---	-----	----	---	---

この μ ITRON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

μ ITRON : 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

H I 8 - 3 Xのver の値はH'0000です。

4. システムの構築

4. 1 概要

H I 8 - 3 X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) H I 8 - 3 Xのセットアップファイルの作成

H I 8 - 3 Xが必要とする作業領域、およびシステムに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。

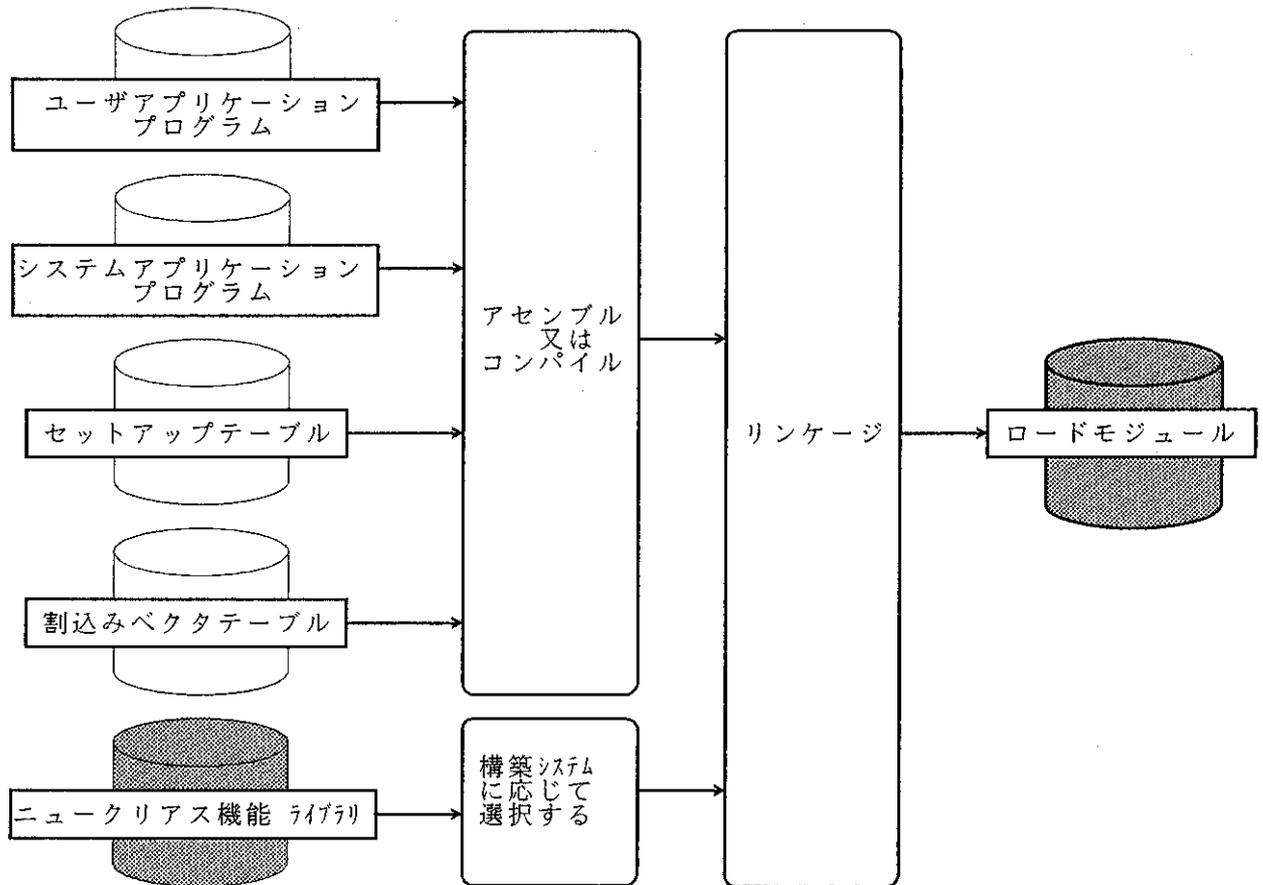


図 4 - 1 システムの構築手順の概要

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。

H I 8 - 3 X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、HI8-3X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手元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、HI8-3X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手元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SEMB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SEMB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SEMBルするには次のコマンドを入力します。

A>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。

A>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。

H 8 / 3 0 0 のアSEMBラの詳しい説明は、「H 8 / 3 0 0 シリーズ クロスアSEMBラ ユーザズマニュアル」を参照してください。

H 8 / 3 0 0 のCコンパイラの詳しい説明は、「H 8 / 3 0 0 シリーズ Cコンパイラ ユーザズマニュアル」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H I 8 - 3 X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

H I 8 - 3 X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H 8 / 3 2 5のフリーランニングタイマ(以下、F R T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、F R TのアウトプットコンペアレジスタA(O C R A)

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、`equ`定義しています。以下に、その部分を説明します。

```
CLKDEVNO      .equ    0                ;; clock divide 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($\phi/2=0$, $\phi/8=1$, $\phi/32=2$)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) : 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 = $x(\text{CPUクロック}/n) - 1$

(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 H 8 / 3 0 0 シリーズを使用する場合の注意

H 8 / 3 2 5 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、H 8 / 3 3 0 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SEMB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SEMB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SEMB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SEMB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SEMB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SEMBルするには次のコマンドを入力します。

```
A>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。

H 8 / 3 0 0 のアSEMBラの詳しい説明は、「H 8 / 3 0 0 シリーズ クロスアSEMBラ ユーザーズマニュアル」を参照してください。

4.4 HI8-3X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。

HI8-3X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\$SEM CB (SEMaphore Control Block : セマフォ管理用の制御領域)

(5) メールボックス定義テーブル

メー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(メールボックス定義数)
- ・H\$MBXCB (MailBoX Control Block : メールボックス管理用の制御領域)

(6) H I 8 - 3 X の R A M 領域の定義およびシステム管理テーブル

H I 8 - 3 X 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 (H\$CUR_SP, H\$TOP_PRI_ID, H\$RDYTB, H\$SYS_CLK, H\$TIM_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。

H\$TIM_INI	タイマ初期設定ルーチンの先頭アドレス	0:未登録
H\$USR_RST	ユーザリセットルーチンの先頭アドレス	0:未登録

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。

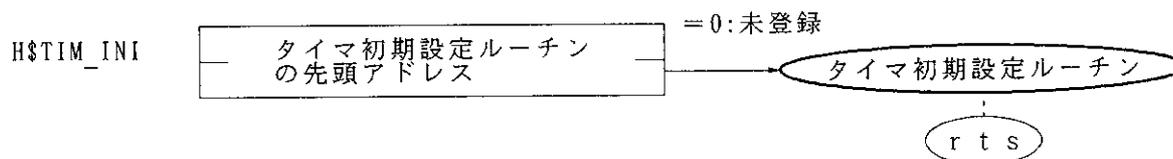


図 4 - 3 タイマ初期設定ルーチンの登録

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。

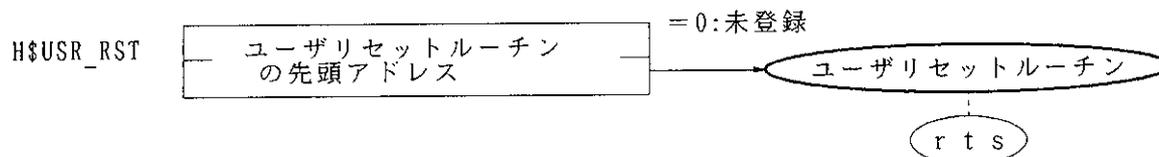


図 4 - 4 ユーザリセットルーチンの登録

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(2⁰ビット=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(2⁰ビット=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。

H I 8 - 3 X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。

H I 8 - 3 X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。

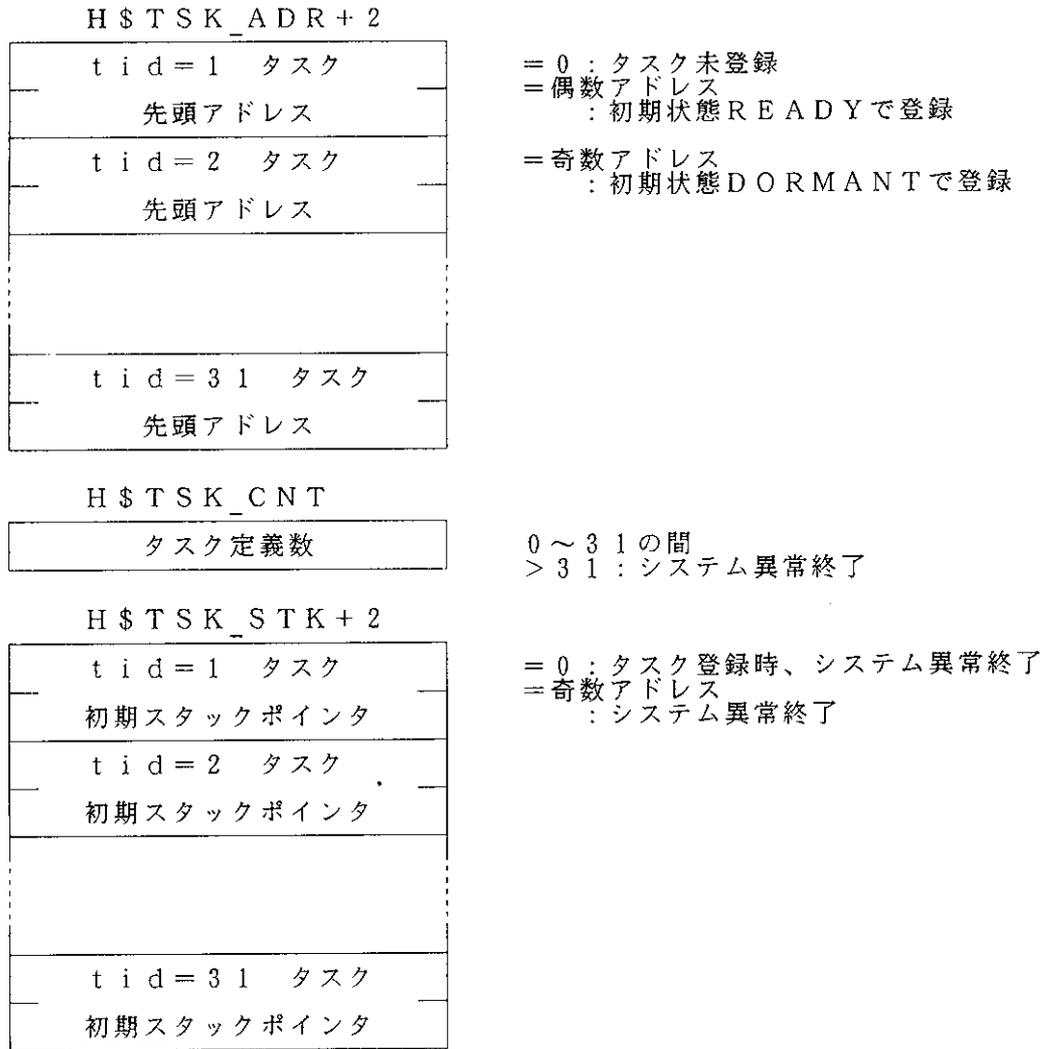


図4-5 タスク定義テーブルのフォーマット

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。

H I 8 - 3 X は、登録されたイベントフラグ毎に F L G C B (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、F L G C B 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。

H \$ F L G _ C N T	イベントフラグ定義数	0 ~ 3 1 の間 > 3 1 : システム異常終了
--------------------	------------	--------------------------------

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。

H I 8 - 3 X は、登録されたセマフォ毎に S E M C B (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、S E M C B 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。

H \$ S E M _ C N T	セマフォ定義数	0 ~ 3 1 の間 > 3 1 : システム異常終了
--------------------	---------	--------------------------------

図 4 - 7 セマフォ定義テーブルのフォーマット

4. 4. 5 メールボックス定義テーブルの作成

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

メールボックス情報には、以下のものがあります。

(1) H\$MBX_CNT(メールボックス定義数)

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

H I 8 - 3 Xは、登録されたメールボックス毎にMBXC B(メー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、MBXC B用領域(メールボックス管理用の制御領域)として、(メールボックス定義数) × 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\$MBXC B*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\$MBXC B*1』は、メー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 にメールボックス定義テーブルのフォーマットを示します。

H\$MBX_CNT	メールボックス定義数	0 ~ 31の間 > 31 : システム異常終了
------------	------------	-----------------------------

図4-8 メールボックス定義テーブルのフォーマット

4. 4. 6 H I 8 - 3 XのRAM領域の定義およびシステム管理テーブルの作成

H I 8 - 3 X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(H\$CUR_SP, H\$TOP_PRI_ID, H\$RDYTB, H\$SYS_CLK, H\$TIM_CNT, H\$TIMCB, H\$TIMCB_BT)

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\$RDYTB』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_BT*^{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のアセンブルするには次のコマンドを入力します。

```
A>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。

H 8 / 3 0 0 のアセンブラの詳しい説明は、「H 8 / 3 0 0 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。

H I 8 - 3 X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H 8 / 3 0 0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、H 8 / 3 3 0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。

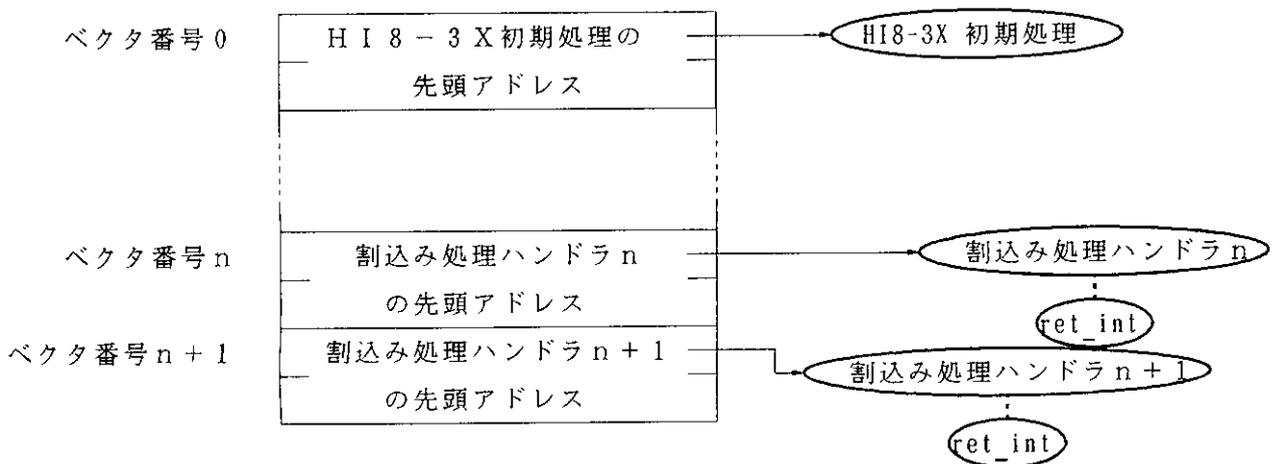
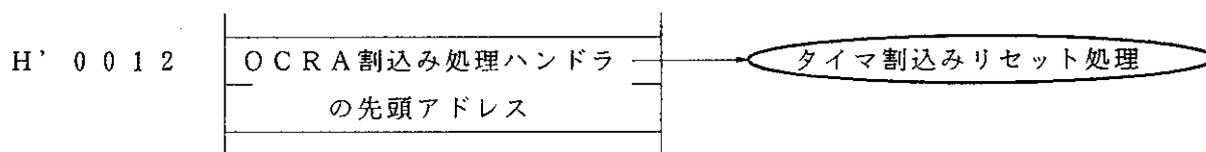


図4-9 割込み処理ハンドラの登録

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。H 8 / 3 2 5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


```

;----- .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 <PRT      ici >
.data.w H_3X_TIM      ;H_3XINT09    :: vector no.09 <PRT      ocia >
.data.w H_3XINT10                    :: vector no.10 <PRT      ocib >
.data.w H_3XINT11                    :: vector no.11 <PRT      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_CNSHDLOER ;H_3XINT18    :: vector no.18 <SCIO   eri >
.data.w H_CNSHDLORX ;H_3XINT19    :: vector no.19 <SCIO   rxi >
.data.w H_CNSHDLOTX ;H_3XINT20    :: vector no.20 <SCIO   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、H I 8-3 X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。

```
A>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。

H 8 / 3 0 0のアセンブラの詳しい説明は、「H 8 / 3 0 0 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H 8 / 3 0 0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。

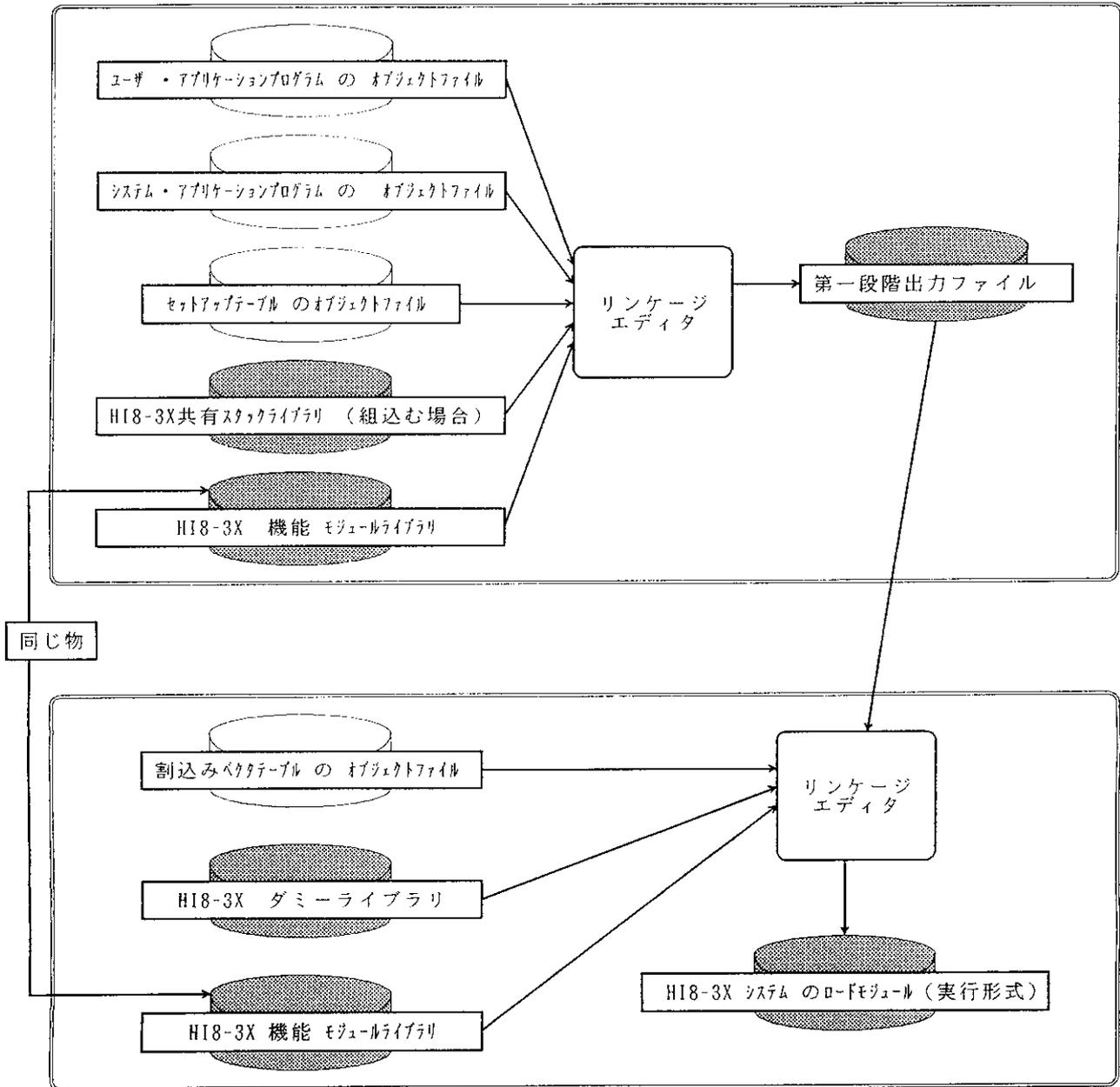


図4-11 システム結合の概要

4. 6. 1 HI8-3X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カージェディタにより結合し、HI8-3Xシステムのロードモジュールを作成します。

HI8-3X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。

HI8-3X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	H3XNUMY.LIB	

HI8-3X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

H I 8 - 3 X 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、H I 8 - 3 X 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
- ④ H I 8 - 3 X 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)

- ⑤ H I 8 - 3 X 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
- ③ H I 8 - 3 X ダミーライブラリ(必ず、④のライブラリより前に入力)
- ④ H I 8 - 3 X 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 H I 8 - 3 X のライブラリは、タスクで R 0 ~ R 3 までのレジスタを保障したライブラリと、R 0 ~ R 6 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 R 0 ~ R 6 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) H 8 / 3 2 5 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) 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、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 H I 8 - 3 X 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、H I 8 - 3 X 共有タスクスタックライブラリは、必ず H I 8 - 3 X 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.BAT』は、HI8-3Xのシステムをリンケージするためのバッチファイルです。

以下に、このバッチファイルの起動方法を示します。

A>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〉

(デフォルト) :	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

このバッチファイルは、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。
- ⑧H I 8 - 3 X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。
 H I 8 - 3 Xの作業領域をH 8 / 3 2 5 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

システムの設定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H 8 / 3 0 0 シリーズASEを使用する方法と、H 8 / 3 0 0 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 H 8 / 3 0 0 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、H I 8 - 3 Xシステムが起動します。

スタートアドレスには、H I 8 - 3 X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。

A S E 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(P o w e r O N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。

A>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H 8 / 3 0.0内蔵PROMまたは、EPROMに書き込み、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

H I 8 - 3 Xのパラメータチェックモジュールライブラリを組み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、H I 8 - 3 X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。

H I 8 - 3 X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、A S Eのブレークキー等によりH I 8 - 3 X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

H I 8 - 3 X 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『H18_3X_C.LIB』と C 言語ヘッダファイル『H18_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、『H18_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『H18_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	H18_3X_C.LIB	H18-3X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	H18_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);      /* syscall */
  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言語での割り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 area          ← ①
    .import ret_int
;
    mov.w    sp, @STKM       ;; sp save to STKM                  ← ②
    mov.w    #TRAPSP, sp    ;; change sp                          ← ③
    push    r0               ;; variable and temporary register save ← ④
    push    r1
    bsr     h3xintXX_main   ;; register recover                  ← 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;
    BR ercd;
    .
    .
    .
    ercd = iwup tsk(tskid); /* syscall (for interrupt handler) */
    if (ercd != E_OK) {
        .
        .
    }
}
```

図 5 - 2 C言語での割り込みハンドラの記述例

(解説)

- ① 割り込みハンドラ専用スタックおよび割り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 割り込み発生時のスタックポインタ退避
- ③ 割り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 割り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
- ⑧ ret_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いません。

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カージェ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。
- ② C 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。
- ③ C 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、次のように行ってください。

標準提供の『HI8_3X_C.BAT』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バッチファイルです。

このバッチファイルは、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いません。

以下に、このバッチファイルの起動方法を示します。

```
A>HI8_3X_C[option] (RET)
```

optionには、次のものがあります。

- (デフォルト) : 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。
- L : 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S C I (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S C I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。

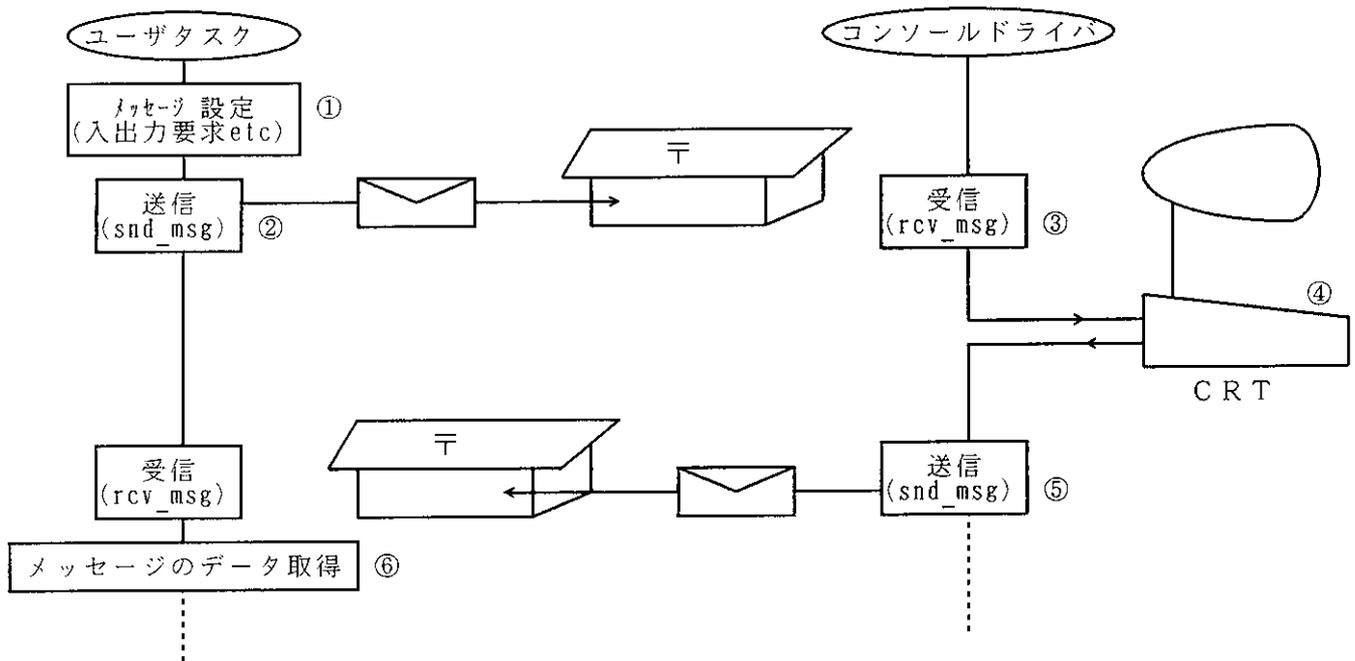


図 A - 1 コンソールドライバ入出力処理の概要

コンソールドライバとユーザタスクはメー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ールボックスを1つ持ち、ユーザタスクはタスクごとにメールボックスを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ールボックスに対し、`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、ユーザタスクのメールボックスに対し、`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

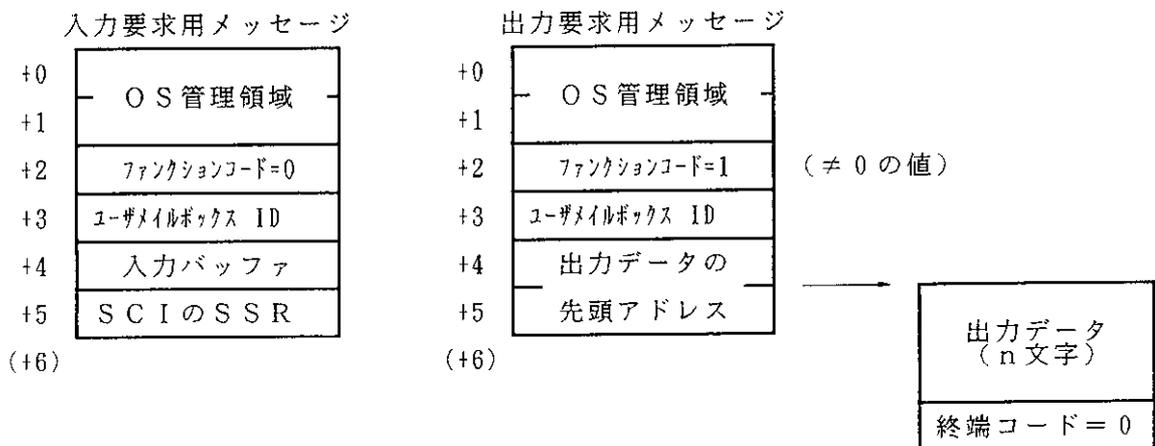
コンソールドライバは、以下の機能をサポートしています。

- (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ールボックス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、ユーザメー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、ユーザメー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SCIOを制御するためのequate定義があります。

H8/325内蔵のSCIO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          ;; SCIO SMR address -> h'ffd8.....①
BRR      .equ      h'ffd9          ;; SCIO BRR address -> h'ffd9.....②
SCR      .equ      h'ffda          ;; SCIO SCR address -> h'ffda.....③
TDR      .equ      h'ffdb          ;; SCIO TDR address -> h'ffdb.....④
SSR      .equ      h'ffdc          ;; SCIO SSR address -> h'ffdc.....⑤
RDR      .equ      h'ffdd          ;; SCIO RDR address -> h'ffdd.....⑥

SMRDT    .equ      h'04            ;; SCIO SMR output data .....⑦
BRRDT    .equ      h'1f            ;; SCIO 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 SCIOのSMR (シリアルモードレジスタ) のアドレス
- ② SCIOのBRR (ビットレートレジスタ) のアドレス
- ③ SCIOのSCR (シリアルコントロールレジスタ) のアドレス
- ④ SCIOのTDR (トランスミットデータレジスタ) のアドレス
- ⑤ SCIOのSSR (シリアルステータスレジスタ) のアドレス
- ⑥ SCIOのRDR (レシーブデータレジスタ) のアドレス
- ⑦ SCIO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 SCIO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ールアドレス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、SCIO0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およびメールアドレス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。

SCIO0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『t01_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) 割り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り込み処理ハンドラを使用しますので、割り込み処理ハンドラ先頭アドレスを割り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 割り込みには、割り込み処理ハンドラ名『H_CNSHDL0ER』を登録します。
- ・Rxi 割り込みには、割り込み処理ハンドラ名『H_CNSHDL0RX』を登録します。
- ・Txi 割り込みには、割り込み処理ハンドラ名『H_CNSHDL0TX』を登録します。

表A-1 に、H8/325のSCI割り込みの種類を示します。

表A-1 H8/325のSCI割り込みの種類

項番	割り込み要因	SCI 0		SCI 1	
		ベクタ番号	ベクタアドレス	ベクタ番号	ベクタアドレス
1	Eri 割り込み（受信エラー 割り込み）	18	H' 24～H' 25	21	H' 2A～H' 2B
2	Rxi 割り込み（入力割り込み）	19	H' 26～H' 27	22	H' 2C～H' 2D
3	Txi 割り込み（出力割り込み）	20	H' 28～H' 29	23	H' 2E～H' 2F

(3) コンソールドライバ用の割り込み処理ハンドラのスタックサイズの設定

SCI割り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割り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。ユーザシステム的环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.BAT』はコンソールドライバをアセンブルするバッチファイルです。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、以下のように行ってください。

```
A>H3XCNS△[option] (RET)
```

optionには次のものがあります。

(デフォルト) : SCIO, SC11 のコンソールドライバのオブジェクトを出力します。

L : SCIO, SC11 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	コンソールドライバ 1	コンソールドライバ 2
1	ファイル名	H3XCNS_0.MAR	H3XCNS_1.MAR
2	使用する S C I	H8/325 の S C I 0	H8/325 の S C I 1
3	タスク名	H_CNSDRV_0	H_CNSDRV_1
4	E R I の割込み処理ハンドラ名	H_CNSHDLOER	H_CNSHDL1ER
5	R X I の割込み処理ハンドラ名	H_CNSHDLORX	H_CNSHDL1RX
6	T X I の割込み処理ハンドラ名	H_CNSHDLOTX	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	9 6 0 0 bps	9 6 0 0 bps
1 0	タスク I D	1	2
1 1	メールボックス I D	1	2

付録 B. システムコール一覧

B. 1 HI8-3X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(=)			タスク/非タスク (T)/(N)
	ROL	R1H	R1L	
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 tcout)		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	wfmode	T
		flgptn=		
16 pol_flg (Poll Event Flag)	flgid	waiptn	wfmode	T
		flgptn=		
17 flg_sts (Get Event Flag Status)	flgid	flgptn=		T N
	wtskid=			
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	semcnt=		T N
	wtskid=			
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	(R1 pk_msg=)		T N
	wtskid=			
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 H18-3X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, ROL @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, ROL @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, ROL @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, ROL @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, ROL @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, ROL @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 #flgid, ROL MOV. B #setptn, R1H JSR @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 #flgid, ROL MOV. B #setptn, R1H JSR @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 #flgid, ROL MOV. B #clrptn, R1H JSR @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 #flgid, ROL MOV. B #waiptn, R1H MOV. B #wfmode, R1L JSR @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 #flgid, ROL MOV. B #waiptn, R1H MOV. B #wfmode, R1L JSR @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 #flgid, ROL JSR @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 #semid, ROL JSR @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 #semid, ROL JSR @isig_sem	semid: セマフォID
20 wai_sem	セマフォに対するP操作	MOV. B #semid, ROL JSR @wai_sem	semid: セマフォID
21 preq_sem	セマフォ資源を得る	MOV. B #semid, ROL JSR @preqs_em	semid: セマフォID
22 sem_sts	セマフォ状態を参照する	MOV. B #semid, ROL JSR @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 #mbxid, R0L MOV.W #pk_msg, R1 JSR @snd_msg	mbxid:メールボックス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 #mbxid, R0L MOV.W #pk_msg, R1 JSR @isndmsg	mbxid:メールボックス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 #mbxid, R0L JSR @rcv_msg	mbxid:メールボックスID
26 prcv_msg	メッセージを受信する	MOV.B #mbxid, R0L JSR @prcv_msg	mbxid:メールボックスID
27 mbx_sts	メールボックス状態を参照する	MOV.B #mbxid, R0L JSR @mbx_sts	mbxid:メールボックスID
割り込み管理			
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 #imask, R1H JSR @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 #pk_time, R1 JSR @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出す	MOV.W #pk_time, R1 JSR @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 #pkver, R1 JSR @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 Semaphore 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.

B. 4 HI8-3Xシステムコール・実例集 (Cフォーマット)

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 "hi8_3x.h"
/*
 * 7. SLP_TSK
 */
ER slp_tsk(void);

void tsk_XX07()
{
ER ercd;

    /* ... */

    ercd = slp_tsk();

    /* ... */
}
```

8. wai_tsk 自タスクを一定時間待ち状態に移行する

```
#include "hi8_3x.h"
/*
 * 8. WAI_TSK
 */
ER wai_tsk(TMO tmout);

void tsk_XX08()
{
ER ercd;
TMO tmout;

    /* ... */

    ercd = wai_tsk(tmout);

    /* ... */
}
```

9. wup_tsk 待ち状態のタスクを起床する

```
#include "hi8_3x.h"
/*
 * 9. WUP_TSK
 */
ER wup_tsk(ID tskid);

void tsk_XX09()
{
ER ercd;
ID tskid;

    /* ... */

    ercd = wup_tsk(tskid);

    /* ... */
}
```

10. iwup_tsk 待ち状態のタスクを起床する (非タスク部専用)

```
#include "hi8_3x.h"
/*
 * 10. IWUP_TSK :
 */
ER iwup_tsk(ID tskid);

void int_XX10()
{
ER ercd;
ID tskid;

    /* ... */

    ercd = iwup_tsk(tskid);

    /* ... */
}
```

11. can_wup タスクの起床要求を無効にする

```
#include "hi8_3x.h"
/*
 * 11. CAN_WUP
 */
ER can_wup(INT *p_wupcnt, ID tskid);

void tsk_XX11()
{
ER ercd;
INT *p_wupcnt;
ID tskid;

    /* ... */

    ercd = can_wup(p_wupcnt, tskid);

    /* ... */
}
```

12. set_flg イベントフラグをセットする

```
#include "hi8_3x.h"
/*
 * 12. SET_FLG
 */
ER set_flg(ID flgid, UINT setptn);

void tsk_XX12()
{
ER ercd;
ID flgid;
UINT setptn;

    /* ... */

    ercd = set_flg(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 "hi8_3x.h"
/*
 * 13. ISET_FLG
 */
ER iset_flg(ID flgid, UINT setptn);

void int_XX13()
{
  ER ercd;
  ID flgid;
  UINT setptn;

  /* ... */

  ercd = iset_flg(flgid, setptn);

  /* ... */
}
```

14. clr_flg イベントフラグをクリアする

```
#include "hi8_3x.h"
/*
 * 14. CLR_FLG
 */
ER clr_flg(ID flgid, UINT clrptn);

void tsk_XX14()
{
  ER ercd;
  ID flgid;
  UINT clrptn;

  /* ... */

  ercd = clr_flg(flgid, clrptn);

  /* ... */
}
```

15. wai_flg イベントフラグを待つ

```
#include "hi8_3x.h"
/*
 * 15. WAI_FLG
 */
ER wai_flg(UINT *p_flgptn, ID flgid,
           UINT waiptn, UINT wfmode);

void tsk_XX15()
{
  ER ercd;
  UINT *p_flgptn;
  ID flgid;
  UINT waiptn;
  UINT wfmode;

  /* ... */

  ercd = wai_flg(p_flgptn, flgid,
                waiptn, wfmode);

  /* ... */
}
```

16. pol_flg イベントフラグを得る

```
#include "hi8_3x.h"
/*
 * 16. POL_FLG
 */
ER pol_flg(UINT *p_flgptn, ID flgid,
           UINT waiptn, UINT wfmode);

void tsk_XX16()
{
  ER ercd;
  UINT *p_flgptn;
  ID flgid;
  UINT waiptn;
  UINT wfmode;

  /* ... */

  ercd = pol_flg(p_flgptn, flgid,
                waiptn, wfmode);

  /* ... */
}
```

17. flg_sts イベントフラグ状態を参照する

```
#include "hi8_3x.h"
/*
 * 17. FLG_STS
 */
ER flg_sts(ID *p_wtskid, UINT *p_flgptn,
           ID flgid);

void tsk_XX17()
{
  ER ercd;
  ID *p_wtskid;
  UINT *p_flgptn;
  ID flgid;

  /* ... */

  ercd = flg_sts(p_wtskid, p_flgptn, flgid);

  /* ... */
}
```

18. sig_sem セマフォに対するV操作

```
#include "hi8_3x.h"
/*
 * 18. SIG_SEM
 */
ER sig_sem(ID semid);

void tsk_XX18()
{
  ER ercd;
  ID semid;

  /* ... */

  ercd = sig_sem(semid);

  /* ... */
}
```

19. isig_sem セマフォに対するV操作
(非タスク部専用)

```
#include "hi8_3x.h"
/*
 * 19. ISIG_SEM
 */
ER isig_sem(ID semid);

void int_XX19()
{
  ER ercd;
  ID semid;

  /* ... */

  ercd = isig_sem(semid);

  /* ... */
}
```

20. wai_sem セマフォに対するP操作

```
#include "hi8_3x.h"
/*
 * 20. WAI_SEM
 */
ER wai_sem(ID semid);

void tsk_XX20()
{
  ER ercd;
  ID semid;

  /* ... */

  ercd = wai_sem(semid);

  /* ... */
}
```

21. preq_sem セマフォ資源を得る

```
#include "hi8_3x.h"
/*
 * 21. PREQ_SEM
 */
ER preq_sem(ID semid);

void tsk_XX21()
{
  ER ercd;
  ID semid;

  /* ... */

  ercd = preq_sem(semid);

  /* ... */
}
```

22. sem_sts セマフォ状態を参照する

```
#include "hi8_3x.h"
/*
 * 22. SEM_STS
 */
ER sem_sts(ID *p_wtskid, UINT *p_semcnt,
            ID semid);

void tsk_XX22()
{
  ER ercd;
  ID *p_wtskid;
  UINT *p_semcnt;
  ID semid;

  /* ... */

  ercd = sem_sts(p_wtskid, p_semcnt, semid);

  /* ... */
}
```

23. snd_msg メッセージを送信する

```
#include "hi8_3x.h"
/*
 * 23. SND_MSG
 */
ER snd_msg(ID mbxid, T_MSG *pk_msg);

void tsk_XX23()
{
  ER ercd;
  ID mbxid;
  T_MSG *pk_msg;

  /* ... */

  ercd = snd_msg(mbxid, pk_msg);

  /* ... */
}
```

24. isnd_msg メッセージを送信する
(非タスク部専用)

```
#include "hi8_3x.h"
/*
 * 24. ISND_MSG
 */
ER isnd_msg(ID mbxid, T_MSG *pk_msg);

void int_XX24()
{
  ER ercd;
  ID mbxid;
  T_MSG *pk_msg;

  /* ... */

  ercd = isnd_msg(mbxid, pk_msg);

  /* ... */
}
```

25. rcv_msg メッセージの受信を待つ

```
#include "hi8_3x.h"
/*
 * 25.RCV_MSG
 */
ER rcv_msg(T_MSG **ppk_msg, ID mbxid);

void tsk_XX25()
{
ER ercd;
T_MSG **ppk_msg;
ID mbxid;

/* ... */

ercd = rcv_msg(ppk_msg, mbxid);

/* ... */
}
```

26. prcv_msg メッセージを受信する

```
#include "hi8_3x.h"
/*
 * 26.PRCV_MSG
 */
ER prcv_msg(T_MSG **ppk_msg, ID mbxid);

void tsk_XX26()
{
ER ercd;
T_MSG **ppk_msg;
ID mbxid;

/* ... */

ercd = prcv_msg(ppk_msg, mbxid);

/* ... */
}
```

27. mbx_sts メールボックス状態を参照する

```
#include "hi8_3x.h"
/*
 * 27.MBX_STS
 */
ER mbx_sts(ID *p_wtskid, T_MSG **ppk_msg,
           ID mbxid);

void tsk_XX27()
{
ER ercd;
ID *p_wtskid;
T_MSG **ppk_msg;
ID mbxid;

/* ... */

ercd = mbx_sts(p_wtskid, ppk_msg, mbxid);

/* ... */
}
```

28. chg_ims 割り込みマスクを変更する

```
#include "hi8_3x.h"
/*
 * 28.CHG_IMS
 */
ER chg_ims(CCR imask);

void tsk_XX28()
{
ER ercd;
CCR imask;

/* ... */

ercd = chg_ims(imask);

/* ... */
}
```

29. ims_sts 割り込みマスクを参照する

```
#include "hi8_3x.h"
/*
 * 29.IMS_STS
 */
ER ims_sts(CCR *p_imask);

void tsk_XX29()
{
ER ercd;
CCR *p_imask;

/* ... */

ercd = ims_sts(p_imask);

/* ... */
}
```

30. set_tim システムクロックを設定する

```
#include "hi8_3x.h"
/*
 * 30.SET_TIM
 */
ER set_tim(T_TIM *pk_time);

void tsk_XX30()
{
ER ercd;
T_TIM *pk_time;

/* ... */

ercd = set_tim(pk_time);

/* ... */
}
```

31. get_tim システムクロックの値を読み出す

```
#include "hi8_3x.h"
/*
 * 31.GET_TIM
 */
ER get_tim(T_TIM *pk_time);

void tsk_XX31()
{
  ER ercd;
  T_TIM *pk_time;

  /* ... */
  ercd = get_tim(pk_time);
  /* ... */
}
```

32. get_ver ニュークリアスのバージョン識別子を得る

```
#include "hi8_3x.h"
/*
 * 32.GET_VER
 */
ER get_ver(T_VER *pk_ver);

void tsk_XX32()
{
  ER ercd;
  T_VER *pk_ver;

  /* ... */
  ercd = get_ver(pk_ver);
  /* ... */
}
```


付録 C. エラーコード一覧

C. 1 システム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 (レジスタ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 H' FFEB(ext_tsk)	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' 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、チェックを行いません。

*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 H18-3X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)	タイムがサポートされていない (システムタイムが未登録)
* E_PAR	H' DF (-H' 21)	パラメータエラー (waitpn=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
ユーザリセットルーチン先頭アドレス	2（固定）	2	
タイム初期設定ルーチン先頭アドレス	2（固定）	2	
タスク先頭アドレス定義テーブル	2×タスク定義数		
タスクスタックポインタ定義テーブル	2×タスク定義数		
タスク定義テーブル	1（固定）	1	
イベントフラグ定義テーブル	1（固定）	1	
セマフォ定義テーブル	1（固定）	1	
メールボックス定義テーブル	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	—		
レジスタ退避領域	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
割込みハンド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ズ	1 2 (固定)	1 2	
NMI 用スタックサイズ	4 (固定)	4	NMI を使用する場合、4バイト必要です
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

H I 8 - 3 X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り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、本算出表の割り込み処理ハンドラのスタック領域は、H 8 / 3 2 5 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
システム管理テーブル	固定	7	
タイマ 管理ブロック	$1 + (4 \times \text{wai_tsk を発行するタスク数})$		
タスク 管理ブロック	$2 \times \text{タスク 定義数}$		
タスク SPセーブ 領域	$2 \times \text{タスク 定義数}$		
イベントフラグ 管理ブロック	$2 \times \text{イベントフラグ 定義数}$		
セマフォ管理ブロック	$3 \times \text{セマフォ定義数}$		
メールボックス 管理ブロック	$4 \times \text{メールボックス 定義数}$		
OS用スタック領域	1 0 または 1 4		NMI を使用する場合、 1 4 バイト 必要
タイマ 割り込み処理用スタック領域	1 0 または 1 4		NMI を使用する場合、 1 4 バイト 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	—		
ベクタ No. 03 用スタック領域	—		NMI
ベクタ No. 04 用スタック領域	—		IRQ0
ベクタ No. 05 用スタック領域	—		IRQ1
ベクタ No. 06 用スタック領域	—		IRQ2
ベクタ No. 07 用スタック領域	—		ポート
ベクタ No. 08 用スタック領域	—		16ビットフリーランニングタイム ICI
ベクタ No. 09 用スタック領域	—		16ビットフリーランニングタイム OCIA
ベクタ No. 10 用スタック領域	—		16ビットフリーランニングタイム OCIB
ベクタ No. 11 用スタック領域	—		16ビットフリーランニングタイム FOVI
ベクタ No. 12 用スタック領域	—		8 ビットタイム0 CMIOA
ベクタ No. 13 用スタック領域	—		8 ビットタイム0 CMIOB

内 訳	計 算 式	容 量	備 考
ベクタ No. 14 用スタック領域	—		8 ビットタイマ0 OV10
ベクタ No. 15 用スタック領域	—		8 ビットタイマ1 CMI1A
ベクタ No. 16 用スタック領域	—		8 ビットタイマ1 CMI1B
ベクタ No. 17 用スタック領域	—		8 ビットタイマ1 OV11
ベクタ 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 ER10
ベクタ 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 RX10
ベクタ 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 TX10
ベクタ 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 ER11
ベクタ 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 RX11
ベクタ 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 TX11
その他 ()	—		
その他 ()	—		
その他 ()	—		
合計			

付録 E. C 構造体一覧

H I 8 - 3 X 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 メー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

H I 8 - 3 X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SEMBル、リンケージ）は、P C - 9801上で行ないます。

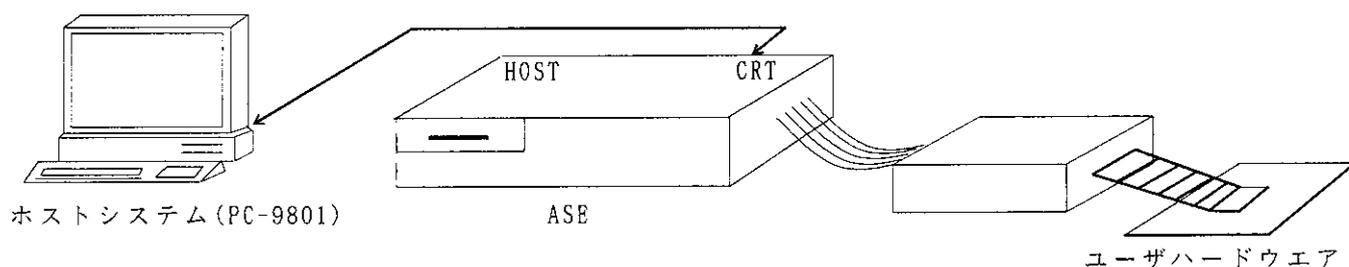
標準提供しているH 8 / 3 2 5用コンソールドライバを使用する場合は、本例を参考にしてシステムを開発してください。

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P C - 9801、実機デバック装置にH 8 / 3 2 5 A S E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（P C - 9 8 0 1）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：

MS - D O S Ver 3.1 以上

関連ユーティリティソフトウェア：

- ①MS - D O S H 8 / 3 0 0 シリーズ クロスアセンブラ
- ②MS - D O S H 8 / 3 0 0 シリーズ Cコンパイラ
- ③MS - D O S H 8 / 3 0 0 シリーズ シミュレータデバッガ
- ④MS - D O S Hシリーズライブラリアン
- ⑤MS - D O S Hシリーズリンケージエディタ
- ⑥MS - D O S 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¥RO_R3	R0～R3までのレジスタ保障ディレクトリ
¥NUC¥RO_R6	R0～R6までのレジスタ保障ディレクトリ

以下にHI8-8X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.BAT	システム結合用バッチファイル
2	H3XLNK.SUB	パ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	パ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	パ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	パ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	H18_3X_C	アセンブル用MAKEファイル
2	H18_3X_C.BAT	アセンブル用バッチファイル
3	H18_3X_C.I	Cインタフェースライブラリ用定義
4	H18_3X_C.L	リスト出力用MAKEファイル
5	H18_3X_C.LIB	Cインタフェースライブラリ
6	H3X_ISETF.MAR	iset_flgCインタフェースのソースファイル
7	H3X_ISIGS.MAR	isig_semCインタフェースのソースファイル
8	H3X_ISNDM.MAR	isnd_msgCインタフェースのソースファイル
9	H3X_ISTAT.MAR	ista_tskCインタフェースのソースファイル
10	H3X_IWUPT.MAR	iwup_tskCインタフェースのソースファイル
11	H3X_PRCVM.MAR	prcv_msgCインタフェースのソースファイル
12	H3X_PREQS.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_RCV.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	H3X_ISETF.OBJ	iset_flgCインタフェースのオブジェクトファイル
39	H3X_ISIGS.OBJ	isig_semCインタフェースのオブジェクトファイル
40	H3X_ISNDM.OBJ	isnd_msgCインタフェースのオブジェクトファイル
41	H3X_ISTAT.OBJ	ista_tskCインタフェースのオブジェクトファイル
42	H3X_IWUPT.OBJ	iwup_tskCインタフェースのオブジェクトファイル
43	H3X_PRCVM.OBJ	prcv_msgCインタフェースのオブジェクトファイル
44	H3X_PREQS.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_RCV.MAR	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	H18_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	H18_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.BAT	アセンブル用バッチファイル
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	SC10用コンソールドライバのソースファイル
6	H3XCNS_1.MAR	SC11用コンソールドライバのソースファイル
7	H3XCNS_0.OBJ	SC10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	SC11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.BAT	CPU別ファイルのアセンブル用バッチファイル
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	H18-3X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	H3XTIM25.MAR	H8/325用タイマ処理
11	H3XTIM30.MAR	H8/330用タイマ処理
12	H3XSETUP.SRC	セットアップテーブル
13	H3XUNI25.SRC	H8/325用未定義割込み処理
14	H3XUNI30.SRC	H8/330用未定義割込み処理
15	H3XVEC25.SRC	H8/325用ベクタテーブル
16	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	H18-3Xモジュールライブラリ(全機能のパ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パラメータチェック有り)
4	H3XNUMDY.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(全機能のパ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パ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 (全機能のパ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パラメータチェック有り)
4	H3XNUMDY. 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 (全機能のパ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パ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 メールボックス定義テーブル

『H\$MBX_CNT』には5を設定し、メールボックスを5つ定義しています。

F. 4. 6 HI8-3X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\$RDY_TBL』，『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/メー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。


```

:|||||||||||||||||||||||||||||||||||||
:|||| HI8-3X set up table ||| TDT (Task Definition Table) |||
:|||||||||||||||||||||||||||||||||||||
DMT .equ 1 ;: dormant status (initial task status)
;
tsk_adr:
H$TSK_ADR: .equ tsk_adr-2 ←タスク 先頭アドレス定義テーブル
tsk_adr01: .data.w H_CNSDRV_0 ;: task top address ←コンソールドライバ 先頭アドレス
tsk_adr02: .data.w 0
tsk_adr03: .data.w 0
tsk_adr04: .data.w 0
tsk_adr05: .data.w 0 } ←タスク 先頭アドレス未登録
tsk_adr_btm:
;
;:.....[ TDT ]:.....[finish!]
;
;:.....[count define]
;:.....[TCNT]
;:.....[FLGCNT]
;:.....[SEMCNT]
;:.....[MBXCNT]
;:.....[WAICNT]
;
tsk_stk:
H$TSK_STK: .equ tsk_stk-2 ←タスクスタックポインタ 定義テーブル
tsk_sp01: .data.w t01_sp ;: task initial sp ←初期スタックポインタ定義
tsk_sp02: .data.w t02_sp ←初期スタックポインタ定義
tsk_sp03: .data.w t03_sp ←初期スタックポインタ定義
tsk_sp04: .data.w t04_sp ←初期スタックポインタ定義
tsk_sp05: .data.w t05_sp ←初期スタックポインタ定義
;
H$TSK_CNT: .data.b TCNT ←タスク 定義テーブル
H$FLG_CNT: .data.b FLGCNT ←イベントフラグ 定義テーブル
H$SEM_CNT: .data.b SEMCNT ←セマフォ定義テーブル
H$MBX_CNT: .data.b MBXCNT ←メールボックス 定義テーブル
;
;:.....[Break point, when HI8-3X crashes]
;:.....[H_setup_error]
;:.....[H_unint_error]
;:.....[H_ctx_error]
;:.....[H_forever_loop]
;

```

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
HSOS_SP: .equ $ ←OS用スタックポインタ
tm_stk .res. b 12 ←タイマ用スタック領域
HSTM_SP: .res. b 02 ←タイマ用スタックポインタ、スタックポインタ
; ←セーブ領域
t_stk .res. b 24 ;; task 01 stack ←タスク用スタック領域 (ID=1)
t01_sp: .equ $ ←タスクスタックポインタ
. res. b 02 ←共用タスクスタック機能の制御領域
t02_sp: .equ $ ←タスク用スタック領域 (ID=2)
. res. b 24 ;; task 02 stack ←タスクスタックポインタ
. res. b 02 ←共用タスクスタック機能の制御領域
t03_sp: .equ $ ←タスク用スタック領域 (ID=3)
. res. b 24 ;; task 03 stack ←タスクスタックポインタ
. res. b 02 ←共用タスクスタック機能の制御領域
t04_sp: .equ $ ←タスク用スタック領域 (ID=4)
. res. b 02 ←タスクスタックポインタ
. res. b 24 ;; task 04 stack ←共用タスクスタック機能の制御領域
t05_sp: .equ $ ←タスク用スタック領域 (ID=5)
. res. b 02 ←タスクスタックポインタ
. 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)
HSCUR_SP: .res. b 02 ;; current task sp save table address
HSTOP_PRI_ID: .res. b 01 ;; top priority task id area
HSTIM_CNT: .res. b 01 ;; wai_tsk counter
HSTIMCB: .res. b 04*WAICNT ;; TIMCB (Timer Control Block)
HSTIMCB_BTM: .equ $
HSRDYTB: .res. b 04 ;; ready tcb counter table
HSSYS_CLK: .res. b 04 ;; system clock counter area
HSTCB: .equ $-2 ←タスク管理用の制御領域
tcb .res. b 02*TCNT ;; TCB (Task Control Block)
HSTKSPTBL: .equ $-2 ←タスクスタックポインタセーブ領域
tsksptbl .res. b 02*TCNT ;; task current sp save area
HSFLGCB: .equ $-2 ←イベントフラグ管理用の制御領域
flgcb .res. b 02*FLGCNT ;; FLGCB (eventFLAG Control Block)
H$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
HSMBXCB: .equ $-4 ←メールボ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)


```

:----- .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 <PRT   ici >
.data.w H_3X_TIM      ;H_3XINT09    :: vector no.09 <PRT   ocia >
.data.w H_3XINT10                    :: vector no.10 <PRT   ocib >
.data.w H_3XINT11                    :: vector no.11 <PRT   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_CNSHDLOER ;H_3XINT18    :: vector no.18 <SC10  eri  >
.data.w H_CNSHDLORX ;H_3XINT19    :: vector no.19 <SC10  rxi  >
.data.w H_CNSHDLOTX ;H_3XINT20    :: vector no.20 <SC10  txi  >
.data.w H_3XINT21                    :: vector no.21 <SC11  eri  >
.data.w H_3XINT22                    :: vector no.22 <SC11  rxi  >
.data.w H_3XINT23                    :: vector no.23 <SC11  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 H I 8 - 3 Xシステムの構築手順

H I 8 - 3 Xシステム(以下H I 8 - 3 X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) H I 8 - 3 Xの実行形式ロードモジュールを、H 8 / 3 2 5 シリーズA S Eのロードコマンド(L O A D)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) H I 8 - 3 Xの実行形式ロードモジュールを、R O M化してシステムに実装します。

F. 7 H I 8 - 3 Xのロードモジュール生成

提供されているF Dの中の全ファイルをコピーし、H I 8 - 3 Xのロードモジュールを生成します。

H I 8 - 3 X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、H I 8 - 3 X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。

- ・ H I 8 - 3 X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。

```
A>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。

```
A>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。

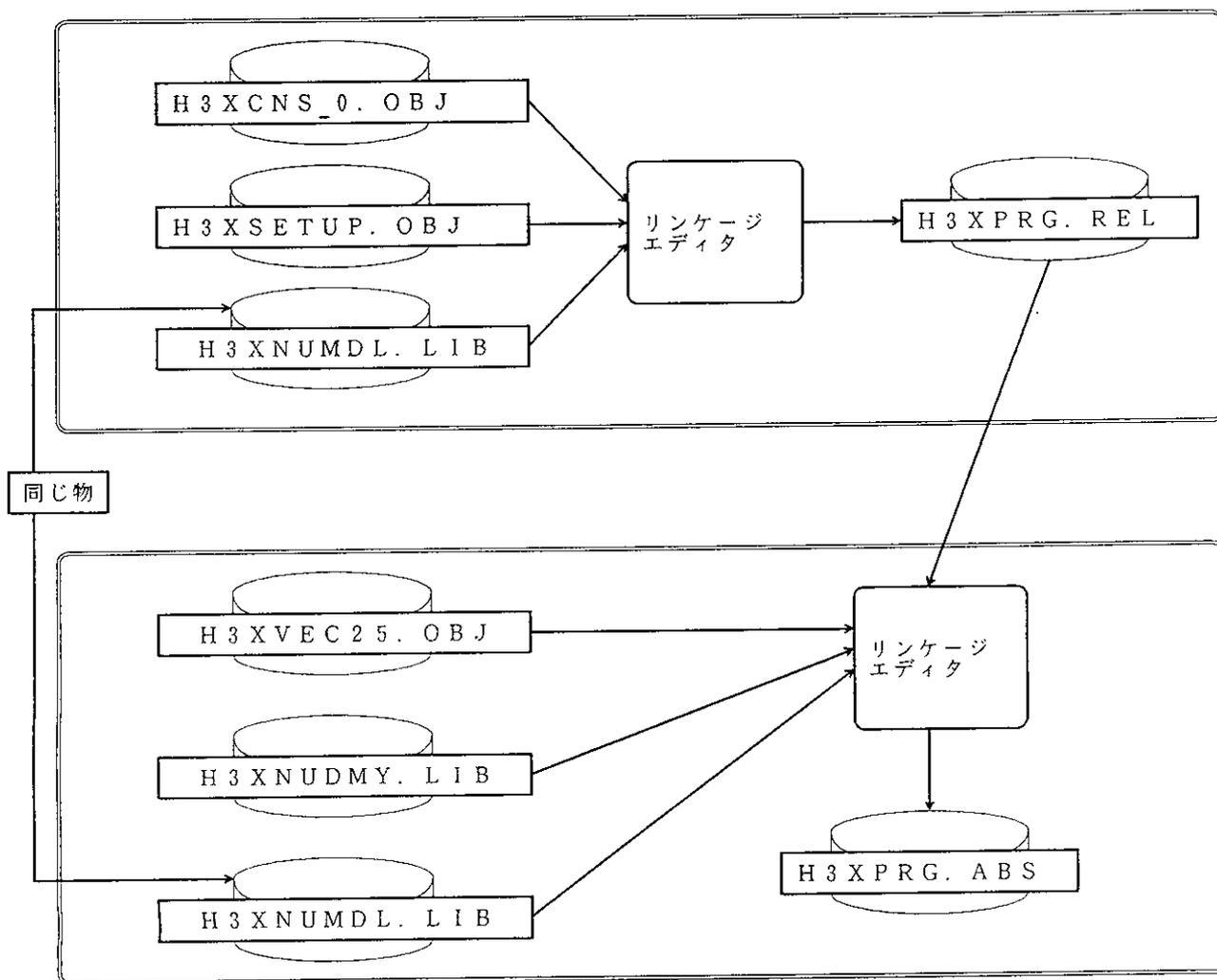
F. 7. 4 システムの結合

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。

A>H3XLNK (RET)

この操作によりバッチファイル『H3XLNK.BAT』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、実行形式モジュール『H3XPRG.ABS』が生成されます。

リストF-3 に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    h18_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

H I 8 - 3 X の実行形式ロードモジュール『H3XPRG. ABS』を、A S E 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。

A S E 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

H I 8 - 3 X 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。

```
A>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 R O M 化し、システムに実装します。

F. 7. 7 H I 8 - 3 X の起動

システムを起動するには、A S E の G O コマンドを使用します。

A S E 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

アセンブラインタフェース	3-4, B-1
アセンブル埋め込み機能	5-1, 5-3
アプリケーションプログラム	1-1, 1-3, 2-3, 4-1, 4-2, 4-3
異常終了	2-36, 2-37, 3-9, 3-32, 4-24, C-1
イベントフラグ	2-13, 2-14, 3-18, 3-19, 3-20, 3-22, 4-7, 4-11
イベントフラグ ID	2-13, 3-18, 3-19, 3-20, 3-22
イベントフラグ管理	4-7, D-3
イベントフラグ定義数	3-18, 3-19, 3-20, 3-22, 4-7, 4-11
イベントフラグ定義テーブル	4-7, 4-11, F-8, D-1
インタフェース	4-23
永久待ち指定	3-15
エラーコード	C-1

カ 行

起床要求回数	3-17
(休止) 状態	2-5, 2-6, 2-11, 3-7, 3-13
キューイング	3-13, 3-14,
共有スタック	2-8, 3-8, 3-9, 3-13, 3-16, 3-17
共有タスクスタック機能	2-6, 2-8, 2-9, 3-8, 3-9, 4-10, 4-20, 4-21, D-1, F-16
クリア指定	2-15
コンソールドライバ	1-1, 1-2, A-1, F-1, F-6, F-17
コンディションコードレジスタ	2-7, 2-11, 2-33, 3-4, 3-33, 3-34

サ 行

サブコマンドファイル	4-22, 5-5, F-17
時間管理	2-1, 2-22, 3-1, 3-35, B-1, B-4, B-5
事象	2-1, 2-5, 2-13, 2-14, 3-21
システム管理システムコール	3-1, 3-37
システム管理テーブル	4-8, 4-13, D-3, F-8
システム起動処理	2-34

システムクロック	3-35, 3-36
システムコールインタフェース	3-3
システムの起動	2-34, 4-23
システムの結合	4-1, 4-19, F-16
（実行可能）状態	2-3, 2-5, 2-6, 3-13
（実行）状態	2-5, 3-13
スケジューリング	2-1, 2-5
セットアップテーブル	1-2, 1-3, 2-37, 4-14, 4-19, A-4, D-1, F-3, F-15
セットアップテーブルチェック	C-1
セットアップファイル	2-34, 2-36, 4-1, 4-7, 4-14
セマフォ	2-10, 2-11, 2-13, 2-16, 3-23, 3-24, 3-26, 4-7, 4-12
セマフォ I D	2-11, 3-23, 3-24, 3-26
セマフォ管理	4-7, D-3
セマフォ定義数	3-23, 3-24, 3-26, 4-7, 4-12
セマフォ定義テーブル	4-7, 4-12, D-1, F-8
セマフォのカウント値	3-23, 3-24

タ 行

タイマカウンタ	2-23
タイマ初期設定ルーチン	2-24, 2-26, 2-34, 4-4, 4-7, 4-8, D-1
タイマ処理ハンドラ	2-24, 2-27
タイマ割込み処理ハンドラ	2-24, 2-26, 4-15
タスク管理	2-1, 2-3, 3-1, 3-7, 4-7, B-1, B-2, B-5, D-3
タスク初期状態	2-6, 4-10
タスクステータスフラグ	3-13
タスク先頭アドレス	2-6, 4-10, A-3, D-1
タスク定義数	3-7, 3-12, 3-16, 3-17, 4-7, 4-10
タスク定義テーブル	2-7, 4-7, 4-10, D-1, F-8
タスクの状態遷移	2-6
タスク付属同期管理	2-1, 2-3, 3-1, B-1, B-2, B-5

ナ 行

内蔵 R O M	1-3
ニュークリアス	1-2, 2-1, 2-3, 2-4, 2-5, 2-6, 2-11, 2-14, 2-18, 2-20, 3-37

ハ 行

バージョン識別子	3-37, B-4
フリーランニングタイム	2-24, 4-4
プログラム容量	1-2
ポーリング	2-13, 2-15, 2-16, 2-18, 2-19, 2-20

マ 行

待ち行列	2-9, 2-16, 2-18, 2-19, 2-20, 3-8, 3-9, 3-23, 3-24, 3-26, 3-28, 3-29, 3-30
(待ち)状態	2-4, 3-8, 3-10, 3-12, 3-13, 3-14, 3-15, 3-24, 3-28
待ちモード	3-21
マルチタスク	1-2, 2-1
未定義割込み	2-32, 2-37, C-1
メールボックス	2-19, 3-28, 3-29, 3-31, 4-7, 4-13, A-1, A-3
メールボックスID	2-19, 3-27, 3-29, 3-31, A-3, A-4, A-6
メールボックス管理	4-7, D-3
メールボックス定義数	3-27, 3-29, 3-31, 4-7, 4-13
メールボックス定義テーブル	4-7, 4-13, D-1, F-8
メッセージ	2-19, 2-20, 3-28, 3-29, 3-30, A-1, A-3,
メッセージフォーマット	2-21, A-3
メモリ容量の算出	D-1

ヤ 行

ユーザリセットルーチン	2-34, 2-36, 4-2, 4-7, 4-8, 5-3, 5-5, D-1
-------------	--

ラ 行

ライブラリ	1-2, 1-3, 4-20, 5-1, 5-6, 5-8, F-5
リアルタイム	1-1, 1-2, 2-1
リンケージ	1-3, 4-21, 4-22, F-1
論理積	2-15, 3-19
論理和	2-14, 3-18

ワ 行

割込み管理	2-1, 2-28, 3-1, 3-32, B-1, B-4, B-5
割込み管理システムコール	3-1, 3-32
割込み処理ハンドラ	2-1, 2-28, 2-31, 2-32, 3-32, 4-3, 4-15, 5-3, 5-4, A-5, A-6, D-2
割込みベクタテーブル	2-28, 4-1, 4-3, 4-15, 4-17, 4-15, A-3, F-12, F-14
割込みマスク	2-11, 2-33, 3-33, 3-34

アルファベット順・索引

A

AND 待ち2-15, 3-21
 ASE1-3, 4-23, F-1

C

can_wup 3-17, B-1, B-2, B-5, B-7
 chg_ims 3-33, B-1, B-4, B-5, B-10
 chg_pri 2-4, 3-10, B-1, B-2, B-5, B-6
 CLKDEVNO 4-5
 clr_flg 2-13, 2-15, 3-19, B-1, B-3, B-5, B-8
 clrptn 3-19
 Cインタフェース 3-5, B-5
 C言語インタフェースライブラリ 1-2, 5-1, 5-6

D

DORMANT(休止) 状態2-5, 2-6, 2-7, 3-13

E

ext_tsk 2-11, 3-9, B-1, B-2, B-5, B-6
 E_CTX 3-3, 3-7, 3-10, 3-11, 3-14, 3-15, 3-16, 3-17,
 3-18, 3-19, 3-20, 3-23, 3-24, 3-27, 3-29,
 C-1
 E_DMT3-16, 3-17, C-1
 E_ILADR3-27, 3-35, 3-36, 3-37, C-1
 E_ILTIME3-15, 3-35, C-1
 E_NODMT3-7, C-1
 E_NOEXS3-7, 3-12, 3-16, 3-17, 3-18, 3-19, 3-20,
 3-22, 3-23, 3-24, 3-26, 3-27, 3-29, 3-31, C-1
 E_PLFAIL 2-15, 2-18, 2-20, 3-20, 3-24, 3-29, C-1
 E_QOVR 2-18, 3-16, 3-20, 3-23, C-1
 E_OBJ3-14, 3-15, 3-20, 3-24, 3-29, C-1
 E_PAR3-20, C-1
 E_SELF 3-16, C-1
 E_TMOUT3-15
 E_TNOSPT 3-15, 3-35, 3-36, C-1

E_TPRI 3-10, C-1

F

FIFO(First-In First-Out) 2-9, 2-16, 2-18, 2-19, 2-20, 3-8, 3-24, 3-28,
3-30

FLGCB 4-7, 4-12

flgid 3-18, 3-19, 3-20, 3-22

flgptn 3-20, 3-22

flg_sts 2-13, 2-15, 3-22, B-1, B-3, B-5, B-8

FRT 2-24, 2-26

G

get_tid 3-11, B-1, B-2, B-5, B-6

get_tim 2-22, 2-23, 3-36, 4-4, 4-8, 4-14, 4-15, B-1,
B-4, B-5, B-11, E-1

get_ver 3-37, B-1, B-4, B-5, B-11, E-1

H

H8/325 2-27, 4-4, 4-15, 4-21, F-1

H\$CUR_SP 4-8, 4-14, F-8

H\$FLGCB 4-7, 4-12, F-8

H\$FLG_CNT 4-7, 4-11, F-8

H\$MBXCB 4-7, 4-13, F-8

H\$MBX_CNT 4-7, 4-13

H\$OS_SP 4-8, 4-13, F-8

H\$RDYTBLL 4-8, 4-14, F-8

H\$SEMxCB 4-7, 4-12, F-8

H\$SEM_CNT 4-7, 4-12, F-8

H\$SYS_CLK 4-8, 4-14, F-8

H\$TCB 4-7, 4-10, F-8

H\$TIM_CNT 4-8, 4-14, F-8

H\$TIM_INI 2-22, 4-7, 4-8, F-8

H\$TIMCB 4-8, 4-14, F-8

H\$TIMCB_BTM 4-8, 4-14, F-8

H\$TM_SP 4-8, 4-13, F-8

H\$TOP_PRI_ID 4-8, 4-14, F-8

H\$TSKSPTBL 4-7, 4-11

H\$TSK_ADR 4-7, 4-10, F-8

H\$TSK_CNT 4-7, 4-10, F-8

H\$TSK_STK 4-7, 4-10, F-8

H\$USR_RST2-34, 2-36, 4-7, 4-8, F-8
 H3XSETUP.SRC 2-34, 4-7, A-4, F-8
 H3XTIM25.MAR 2-24, 2-27, 4-6
 H3XTIM25.I 4-6
 H3XLNK 4-22
 H3XLNK.SUB 4-22
 H3XLNKC.SUB4-22
 H3XLNKCS.SUB 4-22
 H3XLNKS.SUB4-22
 H3XNCMDL.LIB 4-20
 H3XNCSTK.LIB 4-20
 H3XNUDMY.LIB 4-20
 H3XNUMDL.LIB 4-20
 H3XNUSTK.LIB 4-20
 H3XTIM.INC 4-6
 H3XVEC25.SRC 4-14, A-5
 H18_3X_C.LIB.....5-1, 5-6, 5-8
 H18_3X.H5-1
 H183X.BAT4-25
 H_CNSHDLOERA-5, A-6
 H_CNSHDLORXA-5, A-6
 H_CNSHDLOTXA-5, A-6
 H_3X_INIT4-22
 H_3X_TIM 4-15

I

imask3-33
 ims_sts3-33, B-1, B-4, B-5, B-10
 iset_flg 2-13, 2-14, 3-18, B-1, B-3, B-5, B-8
 isig_sem 2-16, 2-18, 3-23, B-1, B-3, B-5, B-9
 isnd_msg 2-19, 2-20, 3-27, B-1, B-4, B-5, B-9, B-1
 ista_tsk 3-7, B-1, B-2, B-5, B-6
 iwup_tsk 3-16, B-1, B-2, B-5, B-7

M

mbx_sts2-19, 2-21, 3-31, B-1, B-4, B-5, B-10
 mbxid3-27, 3-29, 3-31
 MBXCB4-13

N

NMI2-31, D-1, D-3

NON EXIST3-13
NOTSK3-22, 3-26, 3-31

O

OCRADATA 4-5
OR待ち 2-15, 3-21
OS 1-1, 3-28, 3-30, D-1, D-2, D-3
OS管理 3-28, 3-30, A-3

P

pk_msg 3-27, 3-29, 3-31, E-1
pk_time3-35, 3-36, E-1
pk_ver 3-37, E-1
pol_flg2-13, 2-15, 3-20, B-1, B-3, B-5, B-8
prcv_msg 2-19, 2-20, 3-29, B-1, B-4, B-5, B-10, E-1
preq_sem 2-16, 2-18, 3-24, B-1, B-3, B-5, B-9
p_wtskid 3-22, 3-26, 3-31
P 命令 2-16, 2-18, 3-24

R

rcv_msg2-19, 2-20, 3-29, B-1, B-4, B-5, B-10, E-1
READY(実行可能) 状態2-3, 2-5, 2-6, 3-13
ret_int2-11, 2-31, 2-33, 3-32, B-1, B-4, B-5
RUN(実行) 状態2-5, 3-13

S

SCIA-1
SEMCB4-12
semcnt 3-26
semid3-23, 3-24, 3-26
sem_sts2-16, 2-18, 3-26, B-1, B-3, B-5, B-9
setptn 3-18
set_flg2-13, 2-14, 3-18, B-1, B-3, B-5, B-7
set_tim2-22, 2-23, 3-35, 4-8, 4-13, 4-15, B-1,
B-4, B-5, B-10, E-1
sig_sem2-16, 2-18, 3-23, B-1, B-3, B-5, B-8
SLEEP2-5, 2-34
slp_tsk3-14, 3-15, B-1, B-2, B-5, B-7
snd_msg2-19, 2-20, 3-27, B-1, B-4, B-5, B-9, E-1
sta_tsk2-9, 3-7, B-1, B-2, B-5, B-6

T

TCB	4-10
tmout	3-15
tsk_sts	3-12, B-1, B-2, B-5, B-6
tskid	3-7, 3-10, 3-11, 3-12, 3-16, 3-17
tskpri	3-10, 3-12
tskstat	3-12
TWF_ANDW	3-21
TWF_CLR	3-21
TWF_ORW	3-21
t_stk	4-7

V

V命令	2-16, 2-18, 3-23
-----	-------	------------------

W

waitpn	3-20
WAIT(待ち)状態	2-5, 3-14
wai_flg	2-13, 2-15, 3-20, B-1, B-3, B-5, B-8
wai_sem	2-16, 2-18, 3-24, B-1, B-3, B-5, B-9
wai_tsk	2-10, 2-22, 3-15, 4-4, 4-8, 4-13, B-1, B-2, B-5, B-7, D-3
wfmode	3-20
wupcnt	3-16, 3-17
wup_tsk	3-16, B-1, B-2, B-5, B-7

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-702-107A