お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

ユーザーズ・マニュアル

保守／廃止

RX830（ITRON1）

リアルタイム・オペレーティング・システム（Ver.1.2）

基礎編

対象デバイス

V830 ファミリ ™

資料番号　U11730JJ3V1UM00（第 3 版）
発行年月　April 1999　CP(K)

保守／廃止

［メ　モ］

保守／廃止

V800 シリーズおよび V830 ファミリは，日本電気株式会社の商標です。

TRON は，The Real-time Operating system Nucleus の略称です。

ITRON は，Industrial TRON の略称です。

MS-DOS および Windows は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。

UNIX は X/Open カンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標です。

SPARCstation は米国 SPARC International, Inc.の商標です。

Green Hills Software, MULTI は米国 Green Hills Software, Inc.の商標です。

保守／廃止

本製品が外国為替および外国貿易管理法の規定による規制貨物等（または役務）に該当するか否かは，ユーザ（仕様を決定した者）が判定してください。

- 本資料の内容は予告なく変更することがありますので，最新のものであることをご確認の上ご使用ください。
- 文書による当社の承諾なしに本資料の転載複製を禁じます。
- 本資料に記載された製品の使用もしくは本資料に記載の情報の使用に際して，当社は当社もしくは第三者の知的財産権その他の権利に対する保証または実施権の許諾を行うものではありません。上記使用に起因する第三者所有の権利にかかわる問題が発生した場合，当社はその責を負うものではありませんのでご了承ください。
- 本資料に記載された回路，ソフトウエア，及びこれらに付随する情報は，半導体製品の動作例，応用例を説明するためのものです。従って，これら回路・ソフトウエア・情報をお客様の機器に使用される場合には，お客様の責任において機器設計をしてください。これらの使用に起因するお客様もしくは第三者の損害に対して，当社は一切その責を負いません。

M7A 98.8

巻末にアンケート・コーナを設けております。このドキュメントに対するご意見をお気軽にお寄せください。

はじめに

マイクロプロセッサは半導体技術の進歩に伴って急速に普及し，今日ではあらゆる分野で利用されるようになってきました。しかし，マイクロプロセッサを取り巻く処理プログラム量は増大し，各種ハードウエアに合わせた固有のプログラムをその都度作成することが困難となりました。

そこで，高性能，多機能化へと進むマイクロプロセッサの能力を完全に引き出すために，オペレーティング・システム (Operating System： OS) の重要性が高まってきました。

厳密な分け方ではありませんが， OS にはプログラム開発用と制御用の 2 種類があります。プログラム開発用の OS は，開発に使用するハードウエア構成をある程度固定 (パーソナル・コンピュータなど) することができるため，標準的な OS(MS-DOS™， Windows™， UNIX™ など) が流通しやすい環境にあります。

これに対し，制御用の OS は，制御機器に組み込まれて使用されます。つまり，各々のシステムによってハードウエア構成が異なり，しかも，用途に応じた効率の良い動作が要求されるため，標準的な OS が流通しにくい環境にあります。

日本電気 (株) では，このような市場状況を考慮し， V800 シリーズ ™V830 ファミリ ™ を開発，発売し，より強力なマイクロプロセッサを提供する一方で，高機能なマイクロプロセッサが持つ機能を十分に引き出すため，また，将来にわたっての体系的なソフトウエアの構築を支援するために， RX830 を開発，発売しました。

RX830 は高性能，高機能なマイクロプロセッサの応用範囲を拡大し，一層の汎用性を持たせるために開発されたリアルタイム，マルチタスク処理を実現する制御用 OS です。

保守／廃止

目次

表目次

図目次

第 1 章　概説

1.1　概要

RX830 は，効率の良いリアルタイム，マルチタスク処理環境を提供するとともに，対象プロセッサの制御機器分野における応用範囲を拡大することを目的として開発された，組み込み型制御用リアルタイム・マルチタスク OS です。

また，ターゲット・システムに組み込んで使用することを前提として開発されているため，ROM 化を意識し，高速かつコンパクトな OS となっています。

1.2　リアルタイム OS

制御機器分野におけるシステムでは，内外の事象変化に対する即応性が要求されます。しかし，従来のシステムでは，このような要求を単純な割り込み処理で対処してきたため，制御機器が高性能化，多機能化するにつれ，単純な割り込み処理だけでの対処が難しくなってきています。

つまり，システムの複雑化，処理プログラム量の増大により，内外の事象変化に対する処理を，どのような順序で実行させるかを管理することが困難になってきたということです。

そこで，このような問題に対処するために考えられたのがリアルタイム OS です。

リアルタイム OS は，内外の事象変化に対して即応し，最適な処理プログラムを，最適な順序で実行させることをおもな目的としています。

1.3　マルチタスク OS

OS の管理下で実行する処理プログラムの最小単位を「**タスク**」と呼びます。また，1 つのプロセッサ上で複数のタスクを時間的に同時実行させることを「**マルチタスキング**」と呼びます。

実際には，プロセッサ自体は，1 度に 1 つの処理プログラムしか実行することができません。しかし，複数のタスクの実行を何らかの基準 (きっかけ) を利用して切り替えることにより，あたかも複数のタスクが同時実行しているかのようになります。

このように，システム内で定めた何らかの基準を利用してタスクを切り替え，タスクの並列処理を可能なものとした OS がマルチタスク OS です。

マルチタスク OS は，タスクを並列に実行させることにより，システム全体の処理能力を向上させることをおもな目的としています。

1.4 特徴

RX830の特徴を，以下に示します。

(1) ITRON1仕様に準拠

RX830は，組み込み型制御用OSのアーキテクチャとして代表的なITRON1仕様に準拠しています。

なお，ITRON1仕様とは，組み込み型制御用リアルタイム・システムのオペレーティング・システム仕様です。

(2) 高い汎用性

RX830は，ITRON1仕様で規定されているシステム・コールを提供し，アプリケーション・システムの汎用性を高めています。

なお，RX830では，アプリケーション・システムが使用する機能（システム・コール）のみを選択することができるため，コンパクトでありながら，ユーザのニーズに最適なリアルタイム・マルチタスクOSを構築することができます。

(3) リアルタイム，マルチタスク処理の実現

RX830は，完全なリアルタイム，マルチタスク処理を実現するために豊富な機能を提供しています。

- スケジューラ
- タスク管理機能
- 同期通信管理機能
- 割り込み処理管理機能
- 例外処理管理機能
- メモリ管理機能
- 時間管理機能

(4) ROM化の実現

RX830は，ターゲット・システムに組み込んで使用することを想定したリアルタイム・マルチタスクOSであるため，ROM化を意識し，コンパクトな設計が行われています。

(5) 内蔵メモリの有効活用

RX830は，V830ファミリ™マイクロプロセッサが持つ内蔵メモリを有効活用することにより，高速な命令実行速度と，高速なデータ・アクセスを実現しています。

(6) システム構築に有益なユーティリティの提供

RX830は，システムを構築するうえで有益な2つのユーティリティを提供しています。

- コンフィギュレータCF830
- インタフェース・ライブラリ

(7) クロス・ツール

RX830 は，以下に示す V830 ファミリ ™ 用クロス・ツールに対応しています。

- CA830　日本電気 (株) 製
- CCV830　Green Hills Software Inc. 製

(8) RX732 との互換性

RX830 は，RX732 との互換性を保ったリアルタイム OS です。

1.5 構成

RX830 は，ニュークリアス，インタフェース・ライブラリ，コンフィギュレータといった，3 つのサブシステムから構成されています。
以下に，これらの概要を示します。

(1) ニュークリアス

ニュークリアスとは，RX830 の中心 (核) となる部分です。ターゲット・システムに処理プログラムとともに組み込まれ，リアルタイム，マルチタスク制御を実際に行います。また，処理プログラムから発行されたシステム・コールに対応し，様々な処理を行います。

(2) インタフェース・ライブラリ

C 言語で記述した処理プログラムからシステム・コールを発行する場合，システム・コールは外部関数形式で記述します。しかし，この形式とニュークリアスが理解できるシステム・コールの入力形式には相違があります。

そこで，外部関数形式で記述されたシステム・コールをニュークリアスが理解できる入力形式に変換し，処理プログラムとニュークリアスの仲介役を行うのがインタフェース・ライブラリです。

なお，RX830 では，日本電気 (株) 製 V830 ファミリ ™ 用 C コンパイラ CA830 に対応したインタフェース・ライブラリと，Green Hills Software Inc. 製 C クロス V830 ファミリ ™ コンパイラ CCV830 に対応したインタフェース・ライブラリの 2 種類を用意しています。

(3) コンフィギュレータ

コンフィギュレータは，ニュークリアスに提供する各種データを保持した情報ファイル (システム情報テーブル，ブランチ・テーブル、C 言語用ヘッダ・ファイル) の作成作業を支援するために用意されたユーティリティ・ツールです。

コンフィギュレータは独自の記述形式で作成された CF 定義ファイルを入力ファイルとして読み込んだのち、システム情報テーブル，ブランチ・テーブル、C 言語用ヘッダ・ファイルといった情報ファイルを出力します。

1.6 適用分野

RX830は，以下のような機器に適しています。

① 複写機，ワードプロセッサ，プリンタなどのOA機器

② 電話，ファクシミリなどの通信機器

③ エアコン，ビデオ，炊飯器などの家電機器

④ 工業用ロボット，自動車などの制御機器

⑤ 医療機器，宇宙観測機器などのデータ収集／計算機器

1.7 実行環境

RX830は，組み込み型制御用のOSとして開発されているため，以下のハードウエアを備えたターゲット・システム上で動作します。

- プロセッサ

  V830ファミリ™

- 周辺コントローラ

  RX830では，様々な実行環境に対応するために，ニュークリアス内のハードウエア依存部を切り出し，サンプル・ソース・ファイルで提供しています。このため，サンプル・ソース・ファイルを各ターゲット・システム用に書き替えることで，特定の周辺コントローラを要求しません。

- メモリ容量

  RX830が処理を実行するうえで必要となるメモリ容量は，以下のようになります。

  ニュークリアス・テキスト部 ： 約 11 Kbyte
  ニュークリアス・データ部 ： 約 2 Kbyte

  なお，これらの値は，CF定義ファイルにおいてタスクの優先度範囲を最大に設し，システム・コールをすべて使用した場合です。優先度の指定範囲や機能に制限を設けることにより，必要となるメモリ容量を小さくすることも可能です。

1.8 開発環境

システムを開発するうえで必要となるハードウエア環境，および，ソフトウエア環境を，以下に示します。

- ハードウエア環境
  - ホスト・マシン
    - * PC-9800 シリーズ　MS-Windows Ver.3.1
    - * IBM-PC/AT 互換機　MS-Windows Ver.3.1
    - * SPARC station™　SunOS™ 4.1.x
    - * SPARC station™　Solaris™ 2.x
  - インサーキット・エミューレータ
    - * IE-705100-MC
  - ネットワーク・モジュール
    - * IE-70000-MC-SV2　通信モジュール

- ソフトウエア環境
  - クロス・ツール
    - * CA830　日本電気 (株) 製
    - * CCV830　Green Hills Software Inc. 製
  - ディバッガ
    - * ID830　日本電気 (株) 製
    - * MULTI　Green Hills Software Inc. 製
  - システム・パフォーマンス・アナライザ
    - * AZ830　日本電気 (株) 製
  - タスク・ディバッガ
    - * RD830　日本電気 (株) 製

1.9 システム構築

システムの構築とは，RX830の提供媒体（CGMT，または，FD）からユーザの開発環境（ホスト・マシン）上に転送したファイル群を用いて，ロード・モジュールを作成したのち，ターゲット・システムへ組み込むことです。

以下に，システムの構築手順を示します。

(1) 情報テーブル（システム情報テーブル，ブランチ・テーブル）の作成

(2) ユーザ・オウン・コーディング部の作成

(3) 処理プログラム（タスク，割り込みハンドラなど）の作成

(4) ブート処理の作成

(5) 初期化データの退避領域の作成

(6) リンク・ディレクティブ・ファイルの作成

(7) ロード・モジュールの作成

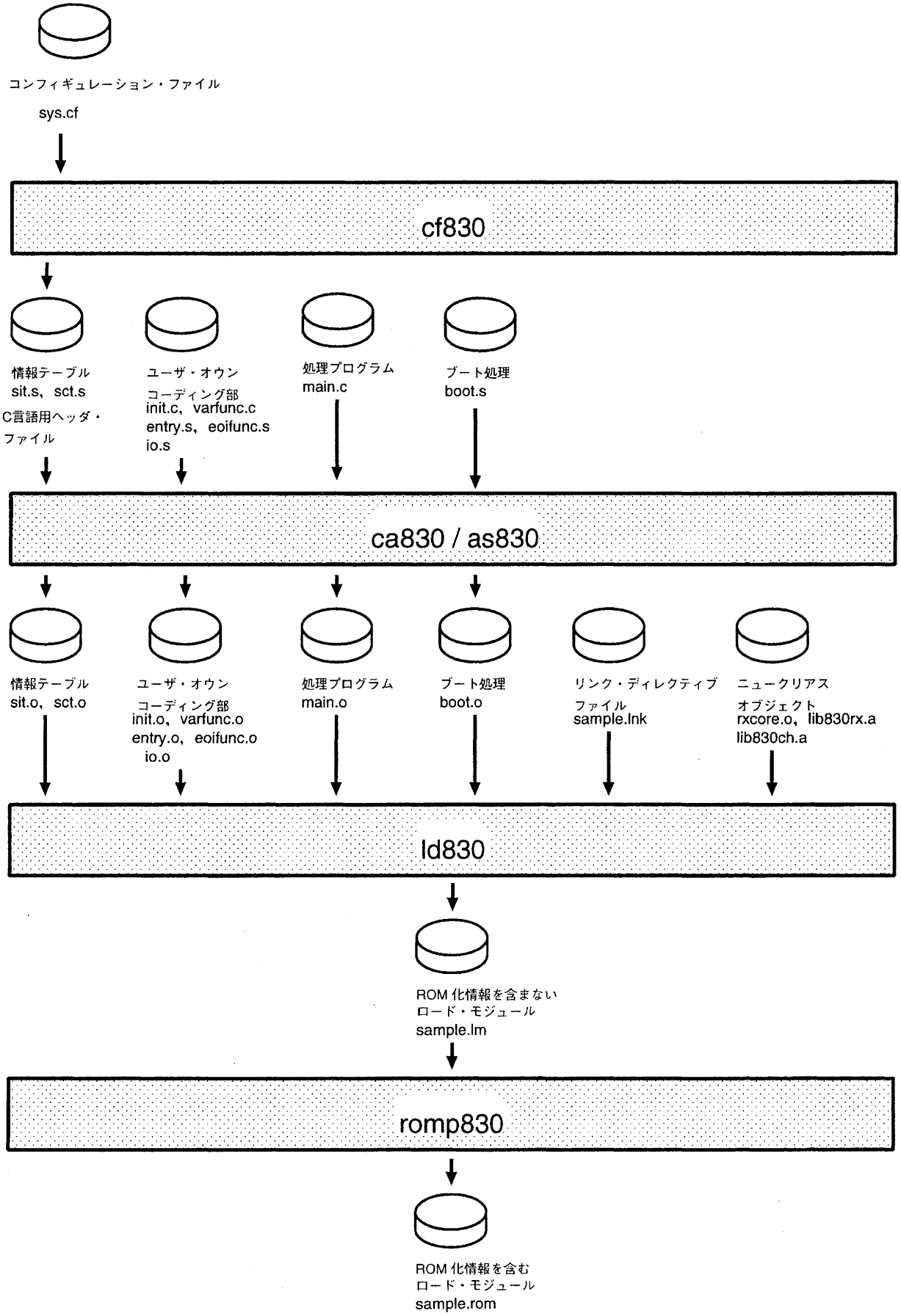
(8) システムへの組み込み

注意 Green Hills Software Inc. 製 C クロス V830 ファミリ ™ コンパイラ CCV830 を使用した場合，初期化データの退避領域を作成する必要はありません。

図 1-1～図 1-2に，システム構築手順を示します。

ただし，図 1-1は，日本電気（株）製 V830 ファミリ ™ 用 C コンパイラ CA830 を使用した場合のシステム構築手順であり，図 1-2は，Green Hills Software Inc. 製 C クロス V830 ファミリ ™ コンパイラ CCV830 を使用した場合のシステム構築手順です。

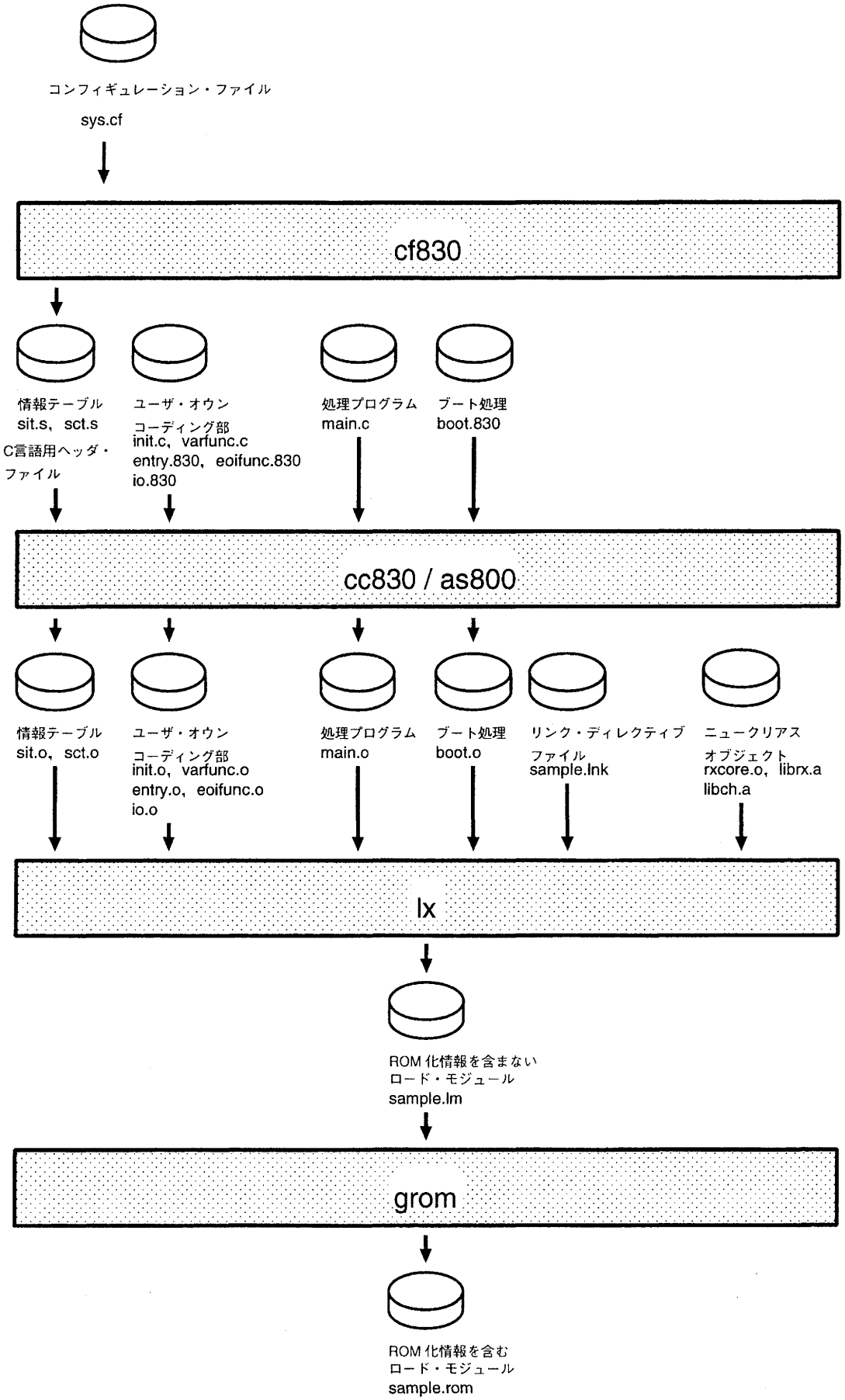
図 1-1　システム構築手順 CA830



注意　図1-1に示した各種ファイルの意味を，以下に示します。

- 情報テーブル
  - `sys.cf` ： コンフィギュレーション・ファイル
  - `sit.s` ： システム情報テーブル
  - `sct.s` ： ブランチ・テーブル
- ユーザ・オウン・コーディング部
  - `init.c` ： ハードウエア初期化部
  - `varfunc.c` ： ソフトウエア初期化部
  - `entry.s` ： ハードウエア依存部（割り込み／例外エントリ）
  - `eoifunc.s` ： ハードウエア依存部（クロック割り込みの後処理）
  - `io.s` ： ハードウエア依存部（ポート操作）
- 処理プログラム
  - `main.c` ： タスク，割り込みハンドラなど
- ブート処理
  - `boot.s` ： ブート処理
- リンク・ディレクティブ・ファイル
  - `sample.lnk` ： リンク・ディレクティブ・ファイル
- ニュークリアス・オブジェクト
  - `rxcore.o` ： ニュークリアス共通部
  - `lib830rx.a` ： ニュークリアス・ライブラリ
  - `lib830ch.a` ： システム・コール用インタフェース・ライブラリ
- ロード・モジュール
  - `sample.lm` ： ロード・モジュール（ROM 化情報を含まない）
  - `sample.rom` ： ロード・モジュール（ROM 化情報を含む）

図 1-2　システム構築手順 CCV830



注意　図 1-2に示した各種ファイルの意味を，以下に示します。

- 情報テーブル
  - `sys.cf` ： コンフィギュレーション・ファイル
  - `sit.s` ： システム情報テーブル
  - `sct.s` ： ブランチ・テーブル
- ユーザ・オウン・コーディング部
  - `init.c` ： ハードウエア初期化部
  - `varfunc.c` ： ソフトウエア初期化部
  - `entry.830` ： ハードウエア依存部（割り込み／例外エントリ）
  - `eoifunc.830` ： ハードウエア依存部（クロック割り込みの後処理）
  - `io.830` ： ハードウエア依存部（ポート操作）
- 処理プログラム
  - `main.c` ： タスク，割り込みハンドラなど
- ブート処理
  - `boot.830` ： ブート処理
- リンク・ディレクティブ・ファイル
  - `sample.lnk` ： リンク・ディレクティブ・ファイル
- ニュークリアス・オブジェクト
  - `rxcore.o` ： ニュークリアス共通部
  - `librx.a` ： ニュークリアス・ライブラリ
  - `libch.a` ： システム・コール用インタフェース・ライブラリ
- ロード・モジュール
  - `sample.lm` ： ロード・モジュール（ROM 化情報を含まない）
  - `sample.rom` ： ロード・モジュール（ROM 化情報を含む）

第 2 章　ニュークリアス

この章では，RX830 の中心 (核) であるニュークリアスについて説明しています。

2.1　概要

ニュークリアスとは，リアルタイム，マルチタスク処理を行う RX830 の核となる部分であり，以下に示す機能を提供しています。

- 管理オブジェクトの初期化処理
- 処理プログラム (タスク，割り込みハンドラなど) から発行されたシステム・コールに対応した処理
- ターゲット・システムの内外で発生した事象に対応し，次に実行すべき処理プログラム (タスク，割り込みハンドラなど) の選択処理

なお，管理モジュールの初期化処理，および，システム・コールに対応した処理は各種管理モジュールで，処理プログラムの選択処理はスケジューラで行われます。

2.2　機能

ニュークリアスは，各種管理モジュールとスケジューラから構成されています。
以下に，各種管理モジュールとスケジューラの機能概要を示します。
なお，これらの機能に関する詳細は，「第 3章　スケジューラ」～「第 10章　初期化処理」を参照してください。

(1) スケジューラ

タスクの実行順序を管理，決定し，タスクにプロセッサの使用権を与えています。

なお，RX830 では，タスクの実行順序を決定する方法として，優先度方式，および，FCFS 方式を採用しています。このため，スケジューラは，駆動された時点で，各タスクに付けられている優先度を参照し，実行可能な状態 (run 状態，および，ready 状態) にあるタスクの中から最適なタスクを選び出し，プロセッサの使用権を与えています。

(2) タスク管理

RX830 の処理単位である，「タスクの生成，起動，終了，削除」などといったタスクの状態操作，および，タスクの状態管理を行っています。

(3) 同期通信管理

「待ち合わせ，排他制御，タスク間通信」といったタスク間の同期通信機能として，以下に挙げる3つの機能を提供しています。

待ち合わせ機能　：イベント・フラグ
排他制御機能　　：セマフォ
タスク間通信機能：メールボックス

(4) 割り込み処理管理

「割り込みハンドラの登録，起動，復帰」などといった割り込みハンドラに関連した処理を行っています。

(5) 例外処理管理

「例外ハンドラの登録，起動」などといった例外ハンドラに関連した処理を行っています。

(6) メモリ管理

システム情報テーブルに定義されたメモリ領域を，以下に挙げる2つの領域に分けて管理しています。

- ニュークリアス・メモリ・プール
- ユーザ・メモリ・プール

なお，RX830では，ダイナミックなメモリ管理も行っており，メモリが必要になった際に獲得し，不要になった際には解放できるといった機能が用意されています。このため，ユーザは，このようなダイナミックなメモリ操作を行うことにより，限りあるメモリ領域を効率良く使用することが可能となります。

(7) 時間管理

ハードウエアにより一定周期で発生するクロック割り込みを利用したタイマ・オペレーション機能を提供しています。

(8) システム・コール拡張機能

汎用性のある豊富な機能を提供する一方で，ユーザ独自の機能（ユーザが記述した関数）をRX830の機能として登録する機能が提供されています。

(9) 初期化処理

RX830が動作するうえで必要となるハードウエアの初期化，および，ニュークリアス資源の生成といった処理を行っています。

2.3 ニュークリアス資源

ニュークリアス資源（タスク，イベント・フラグ，セマフォ，メールボックス，ユーザ・メモリ・プール）は，資源生成時にID番号が与えられ，このID番号によって管理されます。

なお，ID番号の実態は16ビット長の整数で，0x0 ～ 0xffffの範囲で指定することができます。しかし，一部の値はニュークリアスや拡張OSで予約されているため，ユーザがこの予約されている値を指定することは禁止されています。

表2-1に，指定可能なID番号の範囲を示します。

表 2-1 ID番号の範囲

ID番号の種類	範　囲
タスクID番号	0x1 ～ 0xefff
イベント・フラグID番号	0x1 ～ 0xefff
セマフォID番号	0x1 ～ 0xefff
メールボックスID番号	0x1 ～ 0xefff
メモリ・プールID番号	0x0 ～ 0xefff（注1）

注1　0x0は初期化処理（ニュークリアス初期化部）において生成されるユーザ・メモリ・プールに割り当てられます。このため，cre_mplシステム・コールで指定可能なメモリ・プールID番号は0x1 ～ 0xefffとなります。

保守／廃止

第 3 章　スケジューラ

この章では，スケジューリング処理について説明しています。

3.1　概要

RX830 のスケジューラでは，ダイナミックに変化していくタスクの状態を観察することにより，タスクの実行順序を管理／決定し，最適なタスクにプロセッサの使用権を与えています。

3.2　駆動方式

RX830 のスケジューラは，何らかの事象が発生した際に駆動される，事象駆動方式を採用しています。

なお，RX830 における「何らかの事象」とは，タスクの状態遷移を引き起こす可能性のあるシステム・コールの発行，ハンドラからの復帰命令の発行，および，クロック割り込みの発生を意味します。したがって，ユーザの処理プログラム内からタスクの状態遷移を引き起こす可能性のあるシステム・コールが発行された際，ハンドラからの復帰命令が発行された際，および，クロック割り込みが発生した際には，スケジューラが駆動し，タスクのスケジューリング処理を行います。

スケジューラを駆動するシステム・コールの一覧を，以下に示します。

- タスク管理システム・コール

sta_tsk	ext_tsk	exd_tsk	abo_tsk	ter_tsk	chg_pri
rot_rdq	rsm_tsk	slp_tsk	wai_tsk	wup_tsk	cyc_wup

- 同期通信管理システム・コール

set_flg	wai_flg	sig_sem	wai_sem	snd_msg	rcv_msg

- 割り込み処理管理システム・コール

  ret_int

- メモリ管理システム・コール

  get_blk　　rel_blk

- 複合システム・コール

  iret_wup

3.3 スケジューリング方式

RX830 のスケジューリング方式は，優先度方式，および，FCFS 方式を採用しています。

このため，スケジューラは，駆動された時点で，実行可能な状態（run 状態，および，ready 状態）にあるタスクの優先度を参照し，その中から最適なタスクを選び出し，プロセッサの使用権を与えています。

3.3.1 優先度方式

各タスクには，処理を実行するうえでの優先順位を決定する優先度が付けられています。

したがって，スケジューラは，実行可能な状態（run 状態，および，ready 状態）にあるタスクの優先度を参照し，その中から最も高い優先度（最高優先度）を持つタスクを選び出し，プロセッサの使用権を与えています。

なお，タスクの優先度として指定可能な値の範囲は，システム情報テーブルで設定します。

3.3.2 FCFS(First Come First Service) 方式

RX830 では，同一の優先度を複数のタスクに付けることが可能です。

このため，優先度方式におけるタスクを選び出す基準である，「最も高い優先度（最高優先度）を持つタスク」が複数存在する場合があります。

そこで，スケジューラは，最高優先度を持つタスクが複数存在する場合には，先に実行可能な状態となったタスク（ready 状態になってから最も時間が経過しているタスク）を選び出し，プロセッサの使用権を与えています。

3.4 ラウンドロビン方式の実現

RX830では，優先度方式，および，FCFS方式のスケジューリングを行っていますが，この方式では，タスクが複数存在した場合，最初にプロセッサの使用権を与えられたタスクが他の状態へ遷移，または，プロセッサの使用権を放棄しない限り，他タスクが「同一の優先度でありながらも処理を実行することができない」といった事態が生じてきます。

そこで，RX830では，このような事態を回避するスケジューリング方式（ラウンドロビン方式）を実現するために，wai_tsk，rot_rdq などといったシステム・コールを提供しています。

以下に，ラウンドロビン方式の実現例を示します。

（前提条件）

- タスクの優先度

  タスク X > タスク A = タスク B = タスク C

- タスクの状態

  タスク X　　：wait 状態（時限待ち状態）
  タスク A　　：run 状態
  タスク B，タスク C：ready 状態

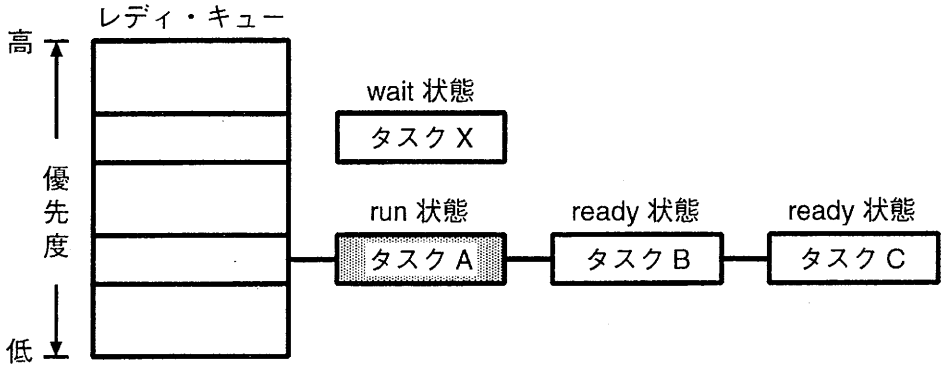
- タスク X は wai_tsk システム・コールと for 文の組み合わせにより，一定周期で起床（周期起床）し，レディ・キューの回転（rot_rdq システム・コールの発行）を行います。

(1) 現在，最高優先度を持つタスク X が wait 状態（時限待ち状態）であるため，タスク A が処理を実行しています。

なお，他タスク（タスク B，タスク C）は，タスク A と同一の優先度を持ちますが，タスク A が他の状態へ遷移，または，プロセッサの使用権を放棄しない限り，処理を実行することができません。

図 3-1に，レディ・キューの状態を示します。

図 3-1　レディ・キューの状態

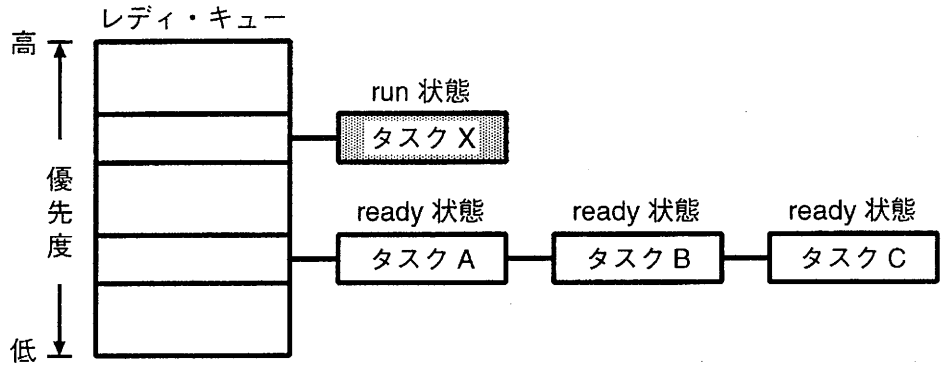


(2) 時限の経過により，タスク X が起床します。

これにより，最高優先度を持つタスク X が run 状態へと遷移します。

図 3-2，レディ・キューの状態を示します。

図 3-2　レディ・キューの状態

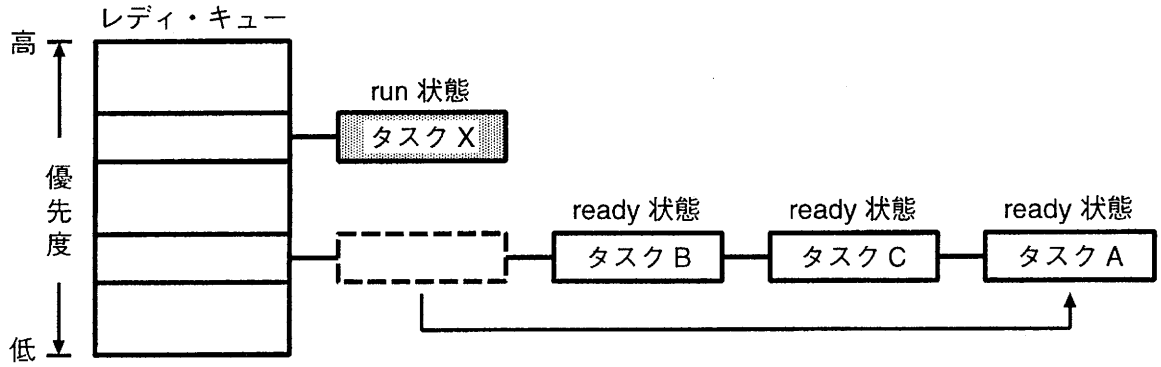


(3) タスク X が rot_rdq システム・コールを発行します。

これにより，タスク A は優先度に応じたレディ・キューの最後尾につなげられます。

図 3-3に，レディ・キューの状態を示します。

図 3-3　レディ・キューの状態

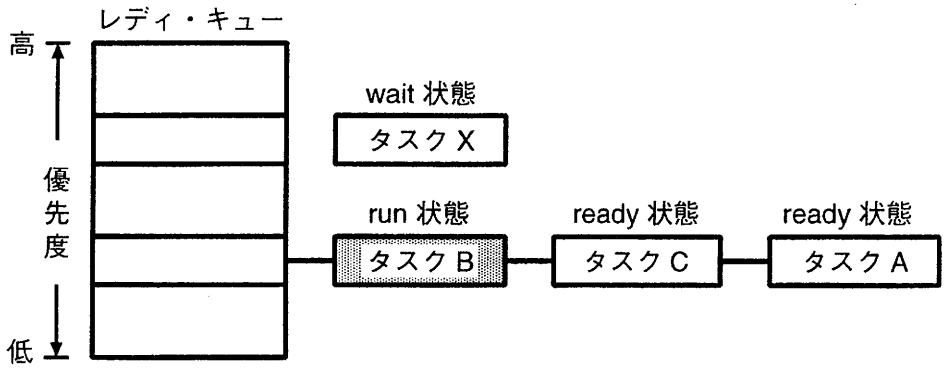


(4) タスクXがwai_tskシステム・コールを発行します。

これにより，タスクXはrun状態からwait状態(時限待ち状態)へと遷移し，タスクBがready状態からrun状態へと遷移します。

図3-4に，レディ・キューの状態を示します。

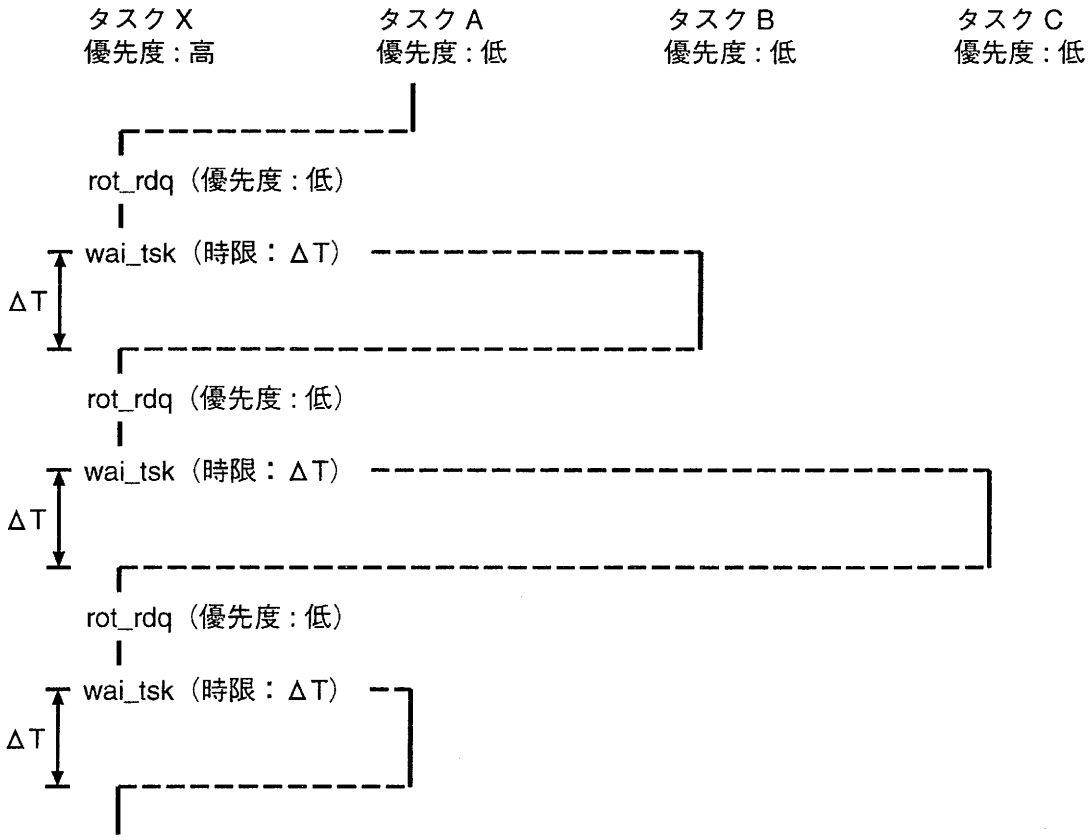
図 3-4　レディ・キューの状態



(5) このようにして，一定周期でrot_rdqシステム・コールを発行することにより，ラウンドロビン方式を実現することができます。

図3-5に，(1)～(5)のラウンドロビン方式による制御の流れを示します。

図 3-5　ラウンドロビン方式によるスケジューリング



第 4 章　タスク管理

この章では，タスク管理について説明しています。

4.1　概要

タスクは実行実体であり，サイズなどが一様でないため，直接管理することが困難です。そこで，RX830 では，タスクと 1 対 1 に対応した管理オブジェクトを用いることにより，タスクの状態管理，および，タスク自体の管理を行っています。

なお，RX830 では，タスクが処理を実行するうえで必要となる実行環境情報（プログラム・カウンタ，作業用レジスタなど）を，「**タスク・コンテキスト**」と呼んでいます。

4.2　タスクの状態

タスクは，処理を実行するうえで必要となる資源の獲得状況，および，事象発生の有無などにより，様々な状態へと遷移していきます。したがって，RX830 では，各タスクが現在どのような状態にあるかを認識し，管理する必要があります。

なお，RX830 では，タスクが遷移する状態を，以下に挙げる 7 つの状態に分けて管理しています。

(1) 未登録 (`non_existent`) 状態

タスクとして生成されていない状態，または，削除された際に遷移する状態です。なお，`non_existent` 状態のタスクは，その実行実体がメモリ上に配置されていながらも，RX830 の管理下にない状態です。

(2) 休止 (`dormant`) 状態

タスクとして生成された際の状態，または，タスクとしての処理を終了した際に遷移する状態です。なお，`dormant` 状態のタスクは，RX830 のスケジューリング対象からは除外されています。

また，`wait` 状態との相違点は，

- すべての資源を解放している
- タスク・コンテキストが処理再開時に初期化される
- 状態操作を伴うシステム・コール (`ter_tsk`，`chg_pri`，`wup_tsk` など) がエラーとなる

などといった点が挙げられます。

(3) 実行可能 (ready) 状態

タスクとしての処理を実行するうえで必要となる準備は整っているが，より高い優先度 (同じ優先度の場合もある) を持つ他タスクが処理を実行中のため，プロセッサの実行権が割り当てられるのを待っている状態です。なお，ready 状態のタスクは，RX830 のスケジューリング対象となります。

(4) 実行 (run) 状態

プロセッサの実行権が割り当てられ，タスクとしての処理を実行中の状態です。なお，run 状態のタスクは，システム全体をとおして 1 タスクしか存在しません。

(5) 待ち (wait) 状態

処理を実行するうえで必要となる条件が整わないため，処理の実行が中断した状態です。なお，wait 状態からの処理再開は，処理の実行が中断した箇所からの再開となります。したがって，処理を再開するうえで必要となるタスク・コンテキストは，中断直前の値が復元されます。

また，RX830 では，wait 状態へと遷移する要因となった条件の種類により，以下に示す 6 つの状態に分けて管理しています。

- 時限待ち状態

  wai_tsk システム・コールを発行した際，自タスクの起床要求カウンタが 0x0 の場合に遷移する状態です。

  この状態の解除は，wup_tsk，cyc_wup，iret_wup システム・コールにより起床要求が発行された際，または，パラメータで指定された時間が経過した際に行われます。

- 起床待ち状態

  slp_tsk システム・コールを発行した際，自タスクの起床要求カウンタが 0x0 の場合に遷移する状態です。

  この状態の解除は，wup_tsk，cyc_wup，iret_wup システム・コールにより起床要求が発行された際に行われます。

- イベント・フラグ待ち状態

  wai_flg システム・コールを発行した際，対象イベント・フラグがパラメータで指定された待ち条件を満足していなかった場合に遷移する状態です。

  この状態の解除は，set_flg システム・コールにより待ち条件を満足するようなビット・パターンが設定された際，または，パラメータで指定された時間が経過した際に行われます。

- 資源待ち状態

  wai_sem システム・コールを発行した際，対象セマフォからパラメータで指定された数の資源を獲得することができなかった場合に遷移する状態です。

この状態の解除は，sig_sem システム・コールにより要求する資源数が解放された際，または，パラメータで指定された時間が経過した際に行われます。

- メッセージ待ち状態

  rcv_msg システム・コールを発行した際，対象メールボックスからメッセージを受信することができなかった場合に遷移する状態です。

  この状態の解除は，snd_msg システム・コールによりメッセージが送信された際，または，パラメータで指定された時間が経過した際に行われます。

- メモリ・ブロック待ち状態

  get_blk システム・コールを発行した際，対象ユーザ・メモリ・プールからパラメータで指定された数のメモリ・ブロックを獲得することができなかった場合に遷移する状態です。

  この状態の解除は，rel_blk システムによりメモリ・ブロックが解放された際，または，パラメータで指定された時間が経過した際に行われます。

(6) 強制待ち (suspend) 状態

他タスクから強制的に処理の実行を中断させられた状態です。なお，suspend 状態からの処理再開は，処理の実行が中断した箇所からの再開となります。したがって，処理を再開するうえで必要となるタスク・コンテキストは，中断直前の値が復元されます。

注意　RX830 では，suspend 状態をネストさせることも可能です。

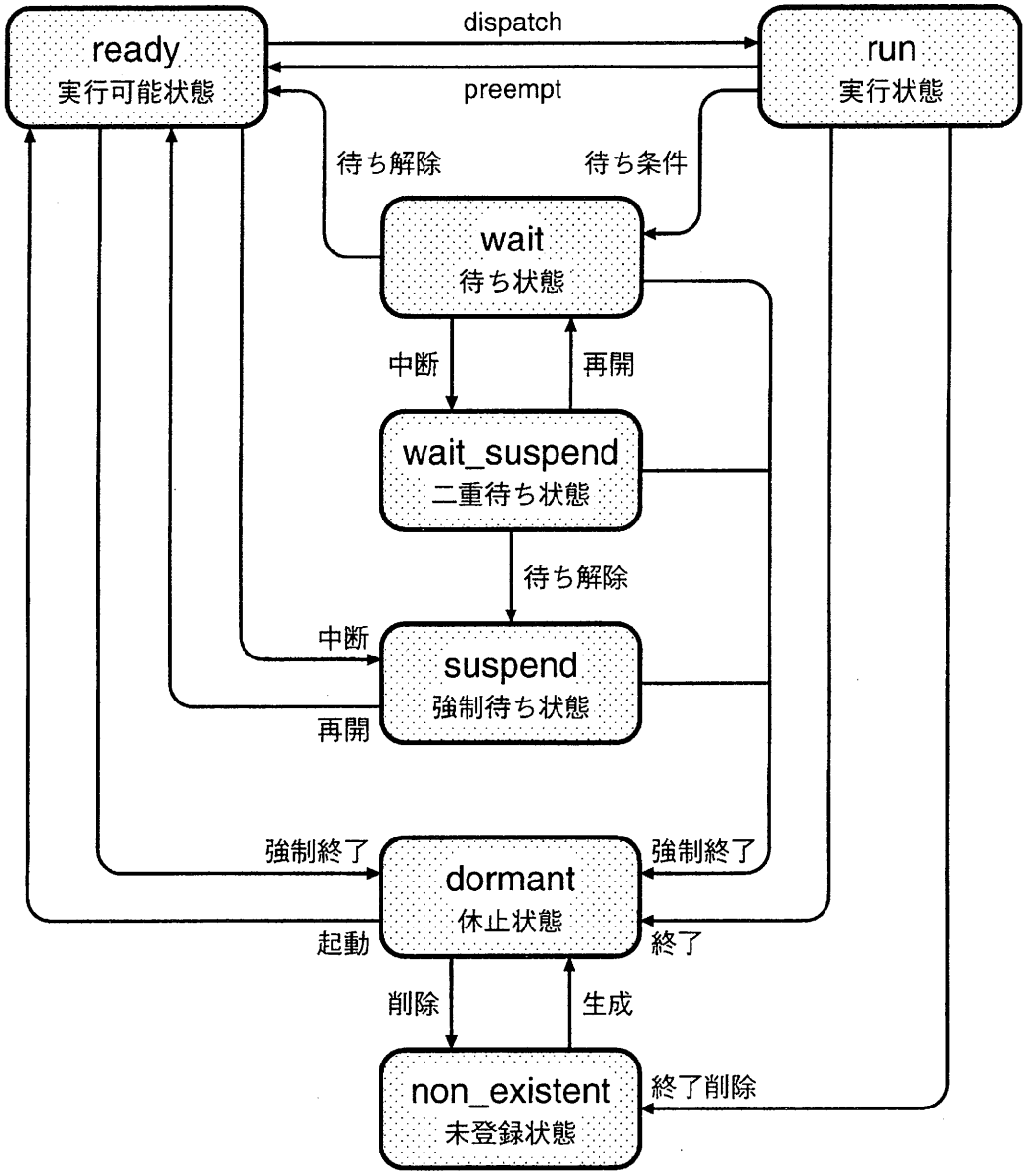
(7) 二重待ち (wait_suspend) 状態

wait 状態と suspend 状態が複合した状態です。なお，wait 状態が解除された際には suspend 状態へ，suspend 状態が解除された際には wait 状態へと遷移します。

図 4-1に，タスクが取り得る各種状態の関係を示します。

保守／廃止

図 4-1　タスクの状態遷移



4.3 タスクの生成／起動

RX830 におけるタスクの生成は，タスクを管理するための領域 (タスク管理ブロック)，および，タスク用のスタック領域を確保／初期化したのち，non_existent 状態のタスクを dormant 状態へと遷移させ，RX830 の管理対象とすることです。

また，RX830 におけるタスクの起動は，dormant 状態のタスクを ready 状態へと遷移させ，RX830 のスケジューリング対象とすることです。

注意　RX830 では，タスクを管理するための領域 (タスク管理ブロック)，および，タスク用のスタック領域をニュークリアス・メモリ・プールから確保しています。

以下に，タスクを生成／起動する際の実現方法を示します。

- システム情報テーブル

  タスクをスタティックに生成／起動する場合，システム情報テーブルに定義します。

  これにより，システム情報テーブルに定義されたタスクは，初期化処理 (ニュークリアス初期化部) において，non_existent 状態から ready 状態へと遷移し，RX830 のスケジューリング対象となります。

- cre_tsk システム・コール

  タスクをダイナミックに生成する場合，cre_tsk システム・コールを発行します。

  これにより，対象タスクは，non_existent 状態から dormant 状態へと遷移し，RX830 の管理対象となります。

- sta_tsk システム・コール

  タスクを起動する場合，sta_tsk システム・コールを発行します。

  これにより，対象タスクは，dormant 状態から ready 状態へと遷移し，RX830 のスケジューリング対象となります。

4.4 タスクの終了／削除

RX830 におけるタスクの終了は，各種状態 (ready 状態，run 状態，wait 状態，suspend 状態，wait_suspend 状態) のタスクを dormant 状態へと遷移させ，RX830 のスケジューリング対象から除外することです。

また，RX830 におけるタスクの削除は，dormant 状態のタスクを non_existent 状態へと遷移させたのち，生成時に確保した領域 (タスク管理ブロック，タスク用のスタック領域) を解放し，RX830 の管理対象から除外することです。

以下に，タスクを終了／削除する際の実現方法を示します。

- ext_tsk システム・コール

  自タスクを終了する場合，ext_tsk システム・コールを発行します。

  これにより，本システム・コールを発行したタスクは，run 状態から dormant 状態へと遷移し，RX830 のスケジューリング対象から除外されます。

  なお，本システム・コールによるタスクの終了形態は，正常終了となります。このため，自タスクに終了時処理ルーチンが登録されていても経由されません。

- exd_tsk システム・コール

  自タスクを終了／削除する場合，exd_tsk システム・コールを発行します。

  これにより，本システム・コールを発行したタスクは，run 状態から non_existent 状態へと遷移し，RX830 の管理対象から除外されます。

  なお，本システム・コールによるタスクの終了形態は，正常終了となります。このため，自タスクに終了時処理ルーチンが登録されていても経由されません。

- abo_tsk システム・コール

  自タスクを強制的に終了する場合，abo_tsk システム・コールを発行します。

  これにより，本システム・コールを発行したタスクは，run 状態から dormant 状態へと遷移し，RX830 のスケジューリング対象から除外されます。

  なお，本システム・コールによるタスクの終了形態は，異常終了となります。このため，自タスクに終了時処理ルーチンが登録されていた場合，ただちに終了時処理ルーチンに制御が移ります。

- ter_tsk システム・コール

  他タスクを強制的に終了する場合，ter_tsk システム・コールを発行します。

  これにより，RX830 では，対象タスクを各種状態（ready 状態，wait 状態，suspend 状態，wait_suspend 状態）から dormant 状態へと遷移させ，RX830 のスケジューリング対象から除外します。

  本システム・コールによるタスクの終了形態は，異常終了となります。このため，対象タスクに終了時処理ルーチンが登録されていた場合，ただちに終了時処理ルーチンに制御が移ります。

- del_tsk システム・コール

  他タスクを削除する場合，del_tsk システム・コールを発行します。

  これにより，対象タスクは，dormant 状態から non_existent 状態へと遷移し，RX830 の管理対象から除外されます。

4.5 終了時処理ルーチン

終了時処理ルーチンは，abo_tsk，または，ter_tsk システム・コールが発行された際，ただちに起動されるタスクの後処理専用ルーチンです。

なお，RX830 では，abo_tsk，または，ter_tsk システム・コールが発行された際，対象タスクに終了時処理ルーチンが登録されているか否かをチェックしています。

このとき，終了時処理ルーチンが登録されていた場合は，対象タスクの優先度を強制的に最高優先度より高い優先度へと変更したのち，対象タスクの再開アドレスを終了時処理ルーチンの先頭アドレスに変更します。このため，対象タスクが run 状態へと遷移した際には，終了時処理ルーチンが実行されることになります。

一方，終了時処理ルーチンが登録されていない場合は，ext_tsk システム・コールを発行した際に行われる終了操作と同様に，対象タスクはただちに dormant 状態へと遷移します。

4.5.1 終了時処理ルーチンの登録／登録解除

終了時処理ルーチンの登録／登録解除は，def_ext システム・コールを発行することにより行われます。

- def_ext システム・コール

  終了時処理ルーチンを登録／登録解除する場合，def_ext システム・コールを発行します。

  これにより，本システム・コールを発行したタスクに対する終了処理ルーチンの登録／登録解除が行われます。

  以下に，def_ext システム・コールを発行する際に必要となる情報（パラメータ）を示します。

  – 終了時処理ルーチンの先頭アドレス
  – 終了時処理ルーチンで使用する固有データ領域のアドレス（gp レジスタ値）

  注意　終了時処理ルーチンの先頭アドレスに 0x0 を指定した場合は，すでに登録されている終了時処理ルーチンの登録解除が行われます。

4.5.2 終了時処理ルーチンの終了

終了時処理ルーチンの終了は，ext_tsk，または，exd_tsk システム・コールを発行することにより行われます。

4.6 システム・タスク

4.6.1 アイドル・タスク

アイドル・タスクは，すべてのタスク (ユーザの定義したタスク) が run 状態，および，ready 状態でなくなった際，すなわち，RX830 のスケジューリング対象となるタスクがシステム内に 1 つも存在しなくなったときに実行されます。

(1) アイドル・タスクの生成／起動

アイドル・タスクは，初期化処理 (ニュークリアス初期化部) で生成，起動され，ready 状態となります。

なお，アイドル・タスクは，RX830 が定義するシステム・タスクであり，ユーザの処理プログラム (タスク，割り込みハンドラなど) からアイドル・タスクに対して操作 (生成，起動，終了，削除など) を行うことは禁止されています。

(2) アイドル・タスクの処理

アイドル・タスクの役割は，プロセッサを HALT 状態にすることです。そこで，アイドル・タスクでは，その処理として，HALT 命令の発行を行っています。

なお，プロセッサの HALT 状態を解除する要因には，以下の 2 つが挙げられます。

- 外部割り込み (マスカブル割り込み，ノンマスカブル割り込み) の発生
  割り込みが発生した場合，該当する割り込みハンドラが起動され，HALT 状態は解除されます。
  ただし，ノンマスカブル割り込みに対応した割り込みハンドラは，システム・コールの発行が禁止されているため，処理終了時には，再び HALT 状態となります。
- ハードウエア・リセット
  ハードウエア・リセットが発生した場合，初期化処理 (ブート処理) からの処理再開となり，HALT 状態は解除されます。

第 5 章　同期通信管理

この章では，同期通信管理について説明しています。

5.1　概要

複数のタスクが並行に実行可能な環境（マルチタスク処理）では，ある１つのタスクの実行結果により，次に起動されるタスクが選択されるとか，他タスクの処理内容に違いが出るなど，タスク間で処理の実行条件を制限しあったり，処理内容が相互に関係しているという場合があります。

このため，ある１つのタスクが実行結果を出すまで他タスクが実行を中断したり，処理を継続するうえで必要な条件が整うまで待つといった，タスク間の連絡機能が必要となります。

RX830 では，このような機能を「**同期機能**」と呼びます。なお，同期機能には，**待ち合わせ機能**と**排他制御機能**があり，RX830 では，待ち合わせ機能としてイベント・フラグを，排他制御機能としてセマフォを提供しています。

また，マルチタスク処理では，他タスクから実行結果を通知してもらうといった，タスク間の通信機能も必要となります。

RX830 では，このような機能を「**タスク間通信機能**」と呼びます。なお，RX830 では，タスク間通信機能としてメールボックスを提供しています。

なお，これらオブジェクト（イベント・フラグ，セマフォ，メールボックス）に対する操作は，ユーザ・タスクからダイナミックに行うことができます。

5.2 イベント・フラグ

マルチタスク処理では，あるタスクの処理結果が出るまで，他タスクが処理の実行を待つといった，タスク間の待ち合わせ機能が必要となります。このような場合，「処理結果が出た」という事象（イベント）が発生したか否かを，他タスクで判断できるような機能があればよく，RX830では，このような機能を実現するために，イベント・フラグを提供しています。

なお，RX830におけるイベント・フラグは，事象の有無を表す1ビットのフラグから構成されるデータの集合体であり，32ビットを一塊の情報として扱っています。

RX830では，イベント・フラグに対するダイナミックな操作を実現するために，以下に示すシステム・コールを提供しています。

cre_flg ： イベント・フラグを生成する
del_flg ： イベント・フラグを削除する
set_flg ： ビット・パターンを設定する
wai_flg ： ビット・パターンをチェックする
flg_adr ： イベント・フラグのアクセス・アドレスを獲得する

注意 RX830が提供するイベント・フラグは，1タスク専有の機構となっています。したがって，1つのイベント・フラグに対して複数のタスクがビット・パターンを設定することは可能ですが，待ち行列にキューイングされるタスクの数は1タスクに限られます。

5.2.1 イベント・フラグの生成／削除

RX830におけるイベント・フラグの生成は，イベント・フラグを管理するための領域（イベント・フラグ管理ブロック）を確保／初期化し，RX830の管理対象とすることです。

また，RX830におけるイベント・フラグの削除は，生成時に確保した領域（イベント・フラグ管理ブロック）を解放し，RX830の管理対象から除外することです。

注意 RX830では，イベント・フラグを管理するための領域（イベント・フラグ管理ブロック）をニュークリアス・メモリ・プールから確保しています。

以下に，イベント・フラグを生成／削除する際の実現方法を示します。

- cre_flg システム・コール

  イベント・フラグを生成する場合，cre_flg システム・コールを発行します。

  これにより，イベント・フラグは生成され，RX830の管理対象となります。

- del_flg システム・コール

  イベント・フラグを削除する場合，del_flg システム・コールを発行します。

  これにより，イベント・フラグは削除され，RX830の管理対象から除外されます。

5.2.2 ビット・パターンの設定

ビット・パターンの設定は，set_flg システム・コールを発行することにより行われます。

- set_flg システム・コール

  対象イベント・フラグにビット・パターンを設定する場合，set_flg システム・コールを発行します。

  ただし，本システム・コールを発行した際，対象イベント・フラグの待ち行列にタスクがキューイングされており，かつ，待ち条件を満足させた場合には，該当タスクを待ち行列から外したのち，wait 状態（イベント・フラグ待ち状態）から ready 状態へと遷移させています。

なお，RX830 では，set_flg システム・コールを発行する際のパラメータ（option）で，ビット・パターンの設定条件を指定することができます。

(1) 論理和（T_OR）

対象イベント・フラグのビット・パターンとパラメータで指定されたビット・パターンの論理和をとり，その結果を対象イベント・フラグに設定します。

(2) 論理積（T_AND）

対象イベント・フラグのビット・パターンとパラメータで指定されたビット・パターンの論理積をとり，その結果を対象イベント・フラグに設定します。

(3) 置換（T_REP）

対象イベント・フラグのビット・パターンとパラメータで指定されたビット・パターンを置換します。

(4) 排他的論理和（T_EXOR）

対象イベント・フラグのビット・パターンとパラメータで指定されたビット・パターンの排他的論理和をとり，その結果を対象イベント・フラグに設定します。

5.2.3 ビット・パターンのチェック

ビット・パターンのチェックは，wai_flg システム・コールを発行することにより行われます。

- wai_flg システム・コール

  対象イベント・フラグのビット・パターンをチェックする場合，wai_flg システム・コールを発行します。

  ただし，本システム・コールを発行した際，対象イベント・フラグのビット・パターンが待ち条件を満足していなかった場合には，自タスクを対象イベント・フラグの待ち行列にキューイングしたのち，run 状態から wait 状態（イベント・フラグ待ち状態）へと遷移させています。

なお，RX830では，wai_flgシステム・コールを発行する際のパラメータ(option)で，各種条件を指定することができます。

(1) タイムアウト

本システム・コールを発行した際，対象イベント・フラグのビット・パターンが待ち条件を満足していなかった場合に遷移するwait状態(イベント・フラグ待ち状態)の時限を指定します。

- タイムアウト指定なし(T_NOOPT)

  待ち条件を満足するまで，イベント・フラグ待ち状態に遷移したままとなります。

- タイムアウト指定あり(T_TMOUT)

  パラメータで指定された時間が経過した際には，イベント・フラグ待ち状態からready状態へと遷移します。

(2) 待ち条件

対象イベント・フラグに対する待ち合わせ条件を指定します。

- AND待ち(T_ANDW)

  要求ビット・パターンで1の設定されている全ビットが対象イベント・フラグに設定されるまでwait状態(イベント・フラグ待ち状態)へと遷移します。

- OR待ち(T_ORW)

  要求ビット・パターンで1の設定されている何れかのビットが対象イベント・フラグに設定されるまでwait状態(イベント・フラグ待ち状態)へと遷移します。

(3) リセット

待ち条件が成立した際，対象イベント・フラグのビット・パターンを0クリアするか否かを指定します。

- リセット指定なし(T_NOOPT)

  0クリアは行われません。

- リセット指定あり(T_RESET)

  0クリアが行われます。

5.2.4 イベント・フラグによる待ち合わせ

イベント・フラグを使用してタスク間の待ち合わせを行った場合の動作例を，以下に示します。

（前提条件）

- タスクの優先度

  タスク A > タスク B

- タスクの状態

  タスク A ： run 状態

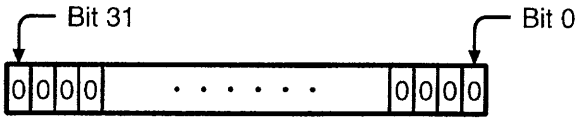
  タスク B ： ready 状態

(1) タスク A が cre_flg システム・コールを発行します。

これにより，タスク A とタスク B がタスク間の待ち合わせを行う際に使用するイベント・フラグが生成されます。

図 5-1に，イベント・フラグが生成された際の状態を示します。

図 5-1 イベント・フラグの初期状態

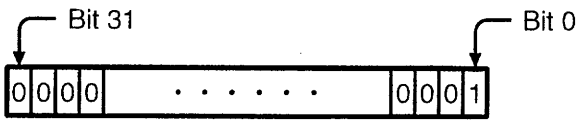


(2) タスク A が wai_flg システム・コールを発行します。なお，要求ビット・パターンは 0x1 であり，待ち条件は T_NOOPT | T_ANDW | T_NOOPT です。

このとき，RX830 が管理している対象イベント・フラグの現ビット・パターン 0x0 とタスク A の要求ビット・パターン 0x1 は異なります。そこで，RX830 では，タスク A を run 状態から wait 状態（イベント・フラグ待ち状態）へと遷移させたのち，対象イベント・フラグの待ち行列にキューイングします。

図 5-2に，wai_flg システム・コール発行時の要求ビット・パターンを示します。

図 5-2 イベント・フラグに対する要求ビット・パターン



(3) タスク A がイベント・フラグ待ち状態へと遷移したことにともない，タスク B が ready 状態から run 状態へと遷移します。

(4) タスクBがflg_adrシステム・コールを発行します。

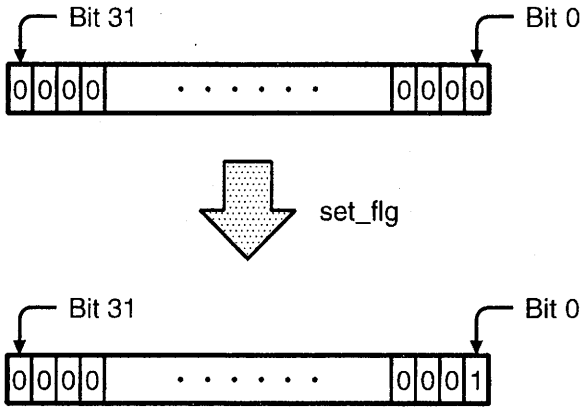
これにより，タスクBは，対象イベント・フラグのアクセス・アドレスを獲得し，対象イベント・フラグに対するダイナミックな操作が可能となります。

(5) タスクBがset_flgシステム・コールを発行します。なお，設定ビット・パターンは0x1であり，設定条件はT_REPです。

これにより，対象イベント・フラグのビット・パターンは0x1に設定され，待ち行列にキューイングされているタスクAの待ち条件を満足したものとなります。そこで，RX830では，タスクAを対象イベント・フラグの待ち行列から外したのち，タスクAをイベント・フラグ待ち状態からready状態へと遷移させます。

図5-3に，set_flgシステム・コール発行時の対象イベント・フラグのビット・パターンを示します。

図5-3　イベント・フラグのビット・パターン



(6) 現在run状態であるタスクBよりも優先度の高いタスクAがready状態へと遷移したことにともない，タスクAがready状態からrun状態へと遷移します。

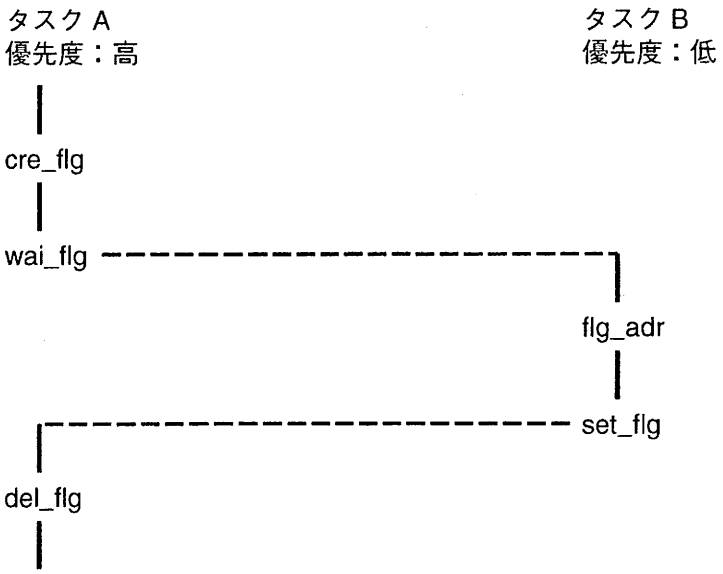
なお，タスクBは，run状態からready状態へと遷移します。

(7) タスクAがdel_flgシステム・コールを発行します。

これにより，タスクAとタスクBがタスク間の待ち合わせに使用したイベント・フラグが削除されます。

図5-4に，(1) ～ (7) の処理の流れを示します。

図 5-4　イベント・フラグによる待ち合わせ



5.3 セマフォ

マルチタスク処理では，並行に実行する複数のタスクが限られた数の資源 (A/D コンバータ，コプロセッサ，ファイル，プログラムなど) を同時に使用するといった競合を防ぐ機能 (排他制御機能)が必要となります。そこで，RX830 では，このような資源の競合を防ぐ手段として，非負数型のセマフォを提供しています。

なお，RX830 におけるセマフォは，資源の数を管理するカウンタであり，8 ビット幅のカウンタでタスク間の排他制御を行っています。

RX830 では，セマフォに対するダイナミックな操作を実現するために，以下に示すシステム・コールを提供しています。

`cre_sem` ： セマフォを生成する
`del_sem` ： セマフォを削除する
`sig_sem` ： 資源を返却する
`wai_sem` ： 資源を獲得する
`sem_adr` ： セマフォのアクセス・アドレスを獲得する

注意 RX830 では，タスクの実行に必要となる各種要素を「資源」と呼んでいます。したがって，RX830 における資源は，A/D コンバータ，コプロセッサなどといったハードウエアや，ファイル，プログラムなどといったソフトウエアのすべての指しています。

5.3.1 セマフォの生成／削除

RX830 におけるセマフォの生成は，セマフォを管理するための領域 (セマフォ管理ブロック) を確保／初期化し，RX830 の管理対象とすることです。

また，RX830 におけるセマフォの削除は，生成時に確保した領域 (セマフォ管理ブロック) を解放し，RX830 の管理対象から除外することです。

注意 RX830 では，セマフォを管理するための領域 (セマフォ管理ブロック) をニュークリアス・メモリ・プールから確保しています。

以下に，セマフォを生成／削除する際の実現方法を示します。

- `cre_sem` システム・コール

  セマフォを生成する場合，`cre_sem` システム・コールを発行します。

  これにより，セマフォは生成され，RX830 の管理対象となります。

- `del_sem` システム・コール

  セマフォを削除する場合，`del_sem` システム・コールを発行します。

  これにより，セマフォは削除され，RX830 の管理対象から除外されます。

5.3.2 資源の返却

資源の返却は，sig_sem システム・コールを発行することにより行われます。

- sig_sem システム・コール

  対象セマフォに資源を返却する場合，sig_sem システム・コールを発行します。

  ただし，本システム・コールを発行した際，対象セマフォの待ち行列にタスクがキューイングされており，かつ，先頭タスクの待ち条件を満足させた場合には，該当タスクを待ち行列から外したのち，wait 状態（資源待ち状態）から ready 状態へと遷移させています。

5.3.3 資源の獲得

資源の獲得は，wai_sem システム・コールを発行することにより行われます。

- wai_sem システム・コール

  対象セマフォから資源を獲得する場合，wai_sem システム・コールを発行します。

  ただし，本システム・コールを発行した際，対象セマフォから要求する数の資源を獲得することができなかった場合には，自タスクを対象セマフォの待ち行列にキューイングしたのち，run 状態から wait 状態（資源待ち状態）へと遷移させています。

なお，RX830 では，wai_sem システム・コールを発行する際のパラメータ（option）で，タイムアウトを指定することができます。

(1) タイムアウト指定なし（T_NOOPT）

待ち条件を満足するまで，wait 状態（資源待ち状態）へと遷移したままとなります。

(2) タイムアウト指定あり（T_TMOUT）

パラメータで指定された時間が経過した際には，wait 状態（資源待ち状態）から ready 状態へと遷移します。

5.3.4 セマフォによる排他制御

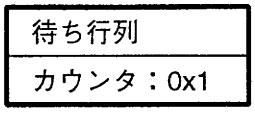
セマフォを使用してタスク間の排他制御を行った場合の動作例を，以下に示します。

（前提条件）

- タスクの優先度
  タスク A > タスク B
- タスクの状態
  タスク A ： run 状態
  タスク B ： ready 状態
- セマフォの属性
  初期カウント値 ： 0x1

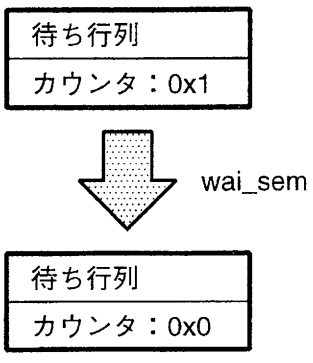
(1) タスク A が cre_sem システム・コールを発行します。なお，初期カウンタ値は 0x1 です。
これにより，タスク A とタスク B がタスク間の排他制御を行う際に使用するセマフォが生成されます。
図 5-5に，セマフォが生成された際の状態を示します。

図 5-5 セマフォの初期状態



(2) タスク A が wai_sem システム・コールを発行します。なお，要求する資源数は 0x1 です。
このとき， RX830 が管理している対象セマフォの資源数は 0x1 です。そこで， RX830 では，対象セマフォのカウンタから 0x1 を減算します。
なお，タスク A は， wai_sem システム・コールを発行した際，即時に資源を獲得することができたため wait 状態（資源待ち状態）へと遷移することはなく， run 状態のままとなります。
図 5-6に， wai_sem システム・コール発行時の対象セマフォのカウンタを示します。

図 5-6 セマフォのカウンタ

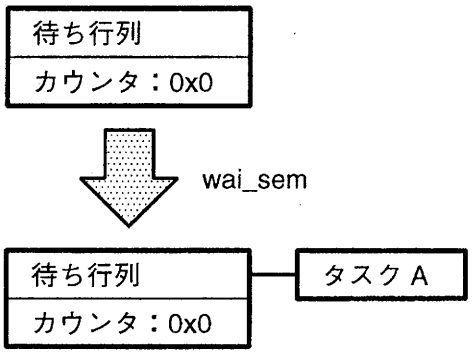


(3) タスク A が wai_sem システム・コールを発行します。なお，要求する資源数は 0x1 です。

このとき，RX830 が管理している資源数は 0x0 です。そこで，RX830 では，対象セマフォの減算処理は行わず，タスク A を run 状態から wait 状態 (資源待ち状態) へと遷移させたのち，対象セマフォの待ち行列にキューイングします。

図 5-7に，wai_sem システム・コール発行時の対象セマフォの待ち行列を示します。

図 5-7　セマフォの待ち行列



(4) タスク A が資源待ち状態へと遷移したことにともない，タスク B が ready 状態から run 状態へと遷移します。

(5) タスク B が sem_adr システム・コールを発行します。

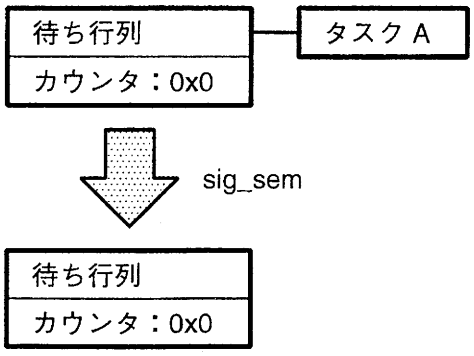
これにより，タスク B は，対象セマフォのアクセス・アドレスを獲得し，対象セマフォに対するダイナミックな操作が可能となります。

(6) タスク B が sig_sem システム・コールを発行します。なお，返却する資源の数は 0x1 です。

これにより，対象セマフォのカウンタは 0x1 に設定され，待ち行列の先頭にキューイングされているタスク A の待ち条件を満足したものとなります。そこで，RX830 では，タスク A を対象セマフォの待ち行列から外したのち，タスク A を資源待ち状態から ready 状態へと遷移させます。

図 5-8に，sig_sem システム・コール発行時の対象セマフォの待ち行列を示します。

図 5-8　セマフォの待ち行列



(7) 現在 run 状態であるタスク B よりも優先度の高いタスク A が ready 状態へと遷移したことにともない，タスク A が ready 状態から run 状態へと遷移します。

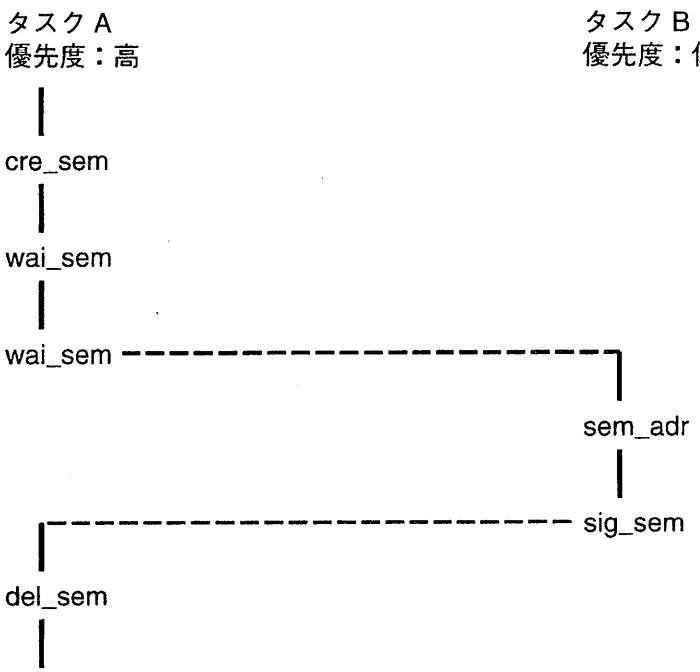
なお，タスク B は，run 状態から ready 状態へと遷移します。

(8) タスク A が del_sem システム・コールを発行します。

これにより，タスク A とタスク B がタスク間の排他制御に使用したセマフォが削除されます。

図 5-9に，(1) ～ (8) の処理の流れを示します。

図 5-9　セマフォによる排他制御



5.4 メールボックス

マルチタスク処理では，並行に実行する複数のタスク間で処理の実行結果を通知してもらうといった機能(タスク間通信機能)が必要となります。そこで，RX830では，このような機能を実現する手段として，メールボックスを提供しています。

なお，RX830におけるメールボックスは，タスク専用待ち行列とメッセージ専用待ち行列から構成されており，タスク間の待ち合わせ機能として利用することもできます。

RX830では，メールボックスに対するダイナミックな操作を実現するために，以下に示すシステム・コールを提供しています。

`cre_mbx` ： メールボックスを生成する
`del_mbx` ： メールボックスを削除する
`snd_msg` ： メッセージを送信する
`rcv_msg` ： メッセージを受信する
`mbx_adr` ： メールボックスのアクセス・アドレスを獲得する

注意　RX830が提供するタスク間通信機能は，メールボックスを介して行われるメッセージの受け渡しにおいて，メッセージのアドレスのみを受け渡しており，メッセージの内容を他の領域にコピーするなどといった処理は行っていません。

5.4.1 メールボックスの生成／削除

RX830におけるメールボックスの生成は，メールボックスを管理するための領域(メールボックス管理ブロック)を確保／初期化し，RX830の管理対象とすることです。

また，RX830におけるメールボックスの削除は，生成時に確保した領域(メールボックス管理ブロック)を解放し，RX830の管理対象から除外することです。

注意　RX830では，メールボックスを管理するための領域(メールボックス管理ブロック)をニュークリアス・メモリ・プールから確保しています。

以下に，メールボックスを生成／削除する際の実現方法を示します。

- `cre_mbx` システム・コール

  メールボックスを生成する場合，`cre_mbx` システム・コールを発行します。

  これにより，メールボックスは生成され，RX830の管理対象となります。

- `del_mbx` システム・コール

  メールボックスを削除する場合，`del_mbx` システム・コールを発行します。

  これにより，メールボックスは削除され，RX830の管理対象から除外されます。

5.4.2 メッセージの送信

メッセージの送信は，snd_msg システム・コールを発行することにより行われます。

- snd_msg システム・コール

  対象メールボックスにメッセージを送信する場合，snd_msg システム・コールを発行します。

  ただし，本システム・コールを発行した際，対象メールボックスのタスク専用待ち行列にタスクがキューイングされていた場合には，該当タスクをタスク専用待ち行列から外したのち，wait 状態（メッセージ待ち状態）から ready 状態へと遷移させています。

5.4.3 メッセージの受信

メッセージの受信は，rcv_msg システム・コールを発行することにより行われます。

- rcv_msg システム・コール

  対象メールボックスからメッセージを受信する場合，rcv_msg システム・コールを発行します。

  ただし，本システム・コールを発行した際，対象メールボックスからメッセージを受信することができなかった場合には，自タスクを対象メールボックスのタスク専用待ち行列にキューイングしたのち，run 状態から wait 状態（メッセージ待ち状態）へと遷移させています。

なお，RX830 では，rcv_msg システム・コールを発行する際のパラメータ（option）で，タイムアウトを指定することができます。

(1) タイムアウト指定なし（T_NOOPT）

メッセージを受信するまで，wait 状態（メッセージ待ち状態）へと遷移したままとなります。

(2) タイムアウト指定あり（T_TMOUT）

パラメータで指定された時間が経過した際には，wait 状態（メッセージ待ち状態）から ready 状態へと遷移します。

5.4.4 メッセージ

RX830では，メッセージを管理するうえで特殊なヘッダを必要とします。このため，ユーザがメッセージを作成する場合，RX830が管理する領域(ユーザ・メモリ・プール)から獲得したメモリ・ブロックを使用します。

また，RX830では，メッセージ専用待ち行列にメッセージをキューイングする際の方法として，メッセージの優先度順／FIFO順のいずれか一方を指定することができますが，メッセージの優先度順を指定した場合には，メッセージの先頭2 byteにメッセージの優先度を格納しなければなりません。

なお，メッセージの優先度は2 byteの符号付き数値(`-0x8000` ～ `0x7fff`)で指定することができ，値が小さいほど優先度は高くなります。

5.4.5 メールボックスによるタスク間通信

メールボックスを使用してタスク間通信を行った場合の動作例を，以下に示します。

(前提条件)

- タスクの優先度

  タスクA > タスクB
- タスクの状態

  タスクA ： `run`状態

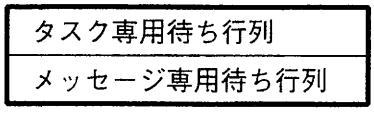
  タスクB ： `ready`状態

(1) タスクAが`cre_mbx`システム・コールを発行します。

これにより，タスクAとタスクBがタスク間の通信を行う際に使用するメールボックスが生成されます。

図5-10に，メールボックスが生成された際の状態を示します。

図5-10 メールボックスの初期状態

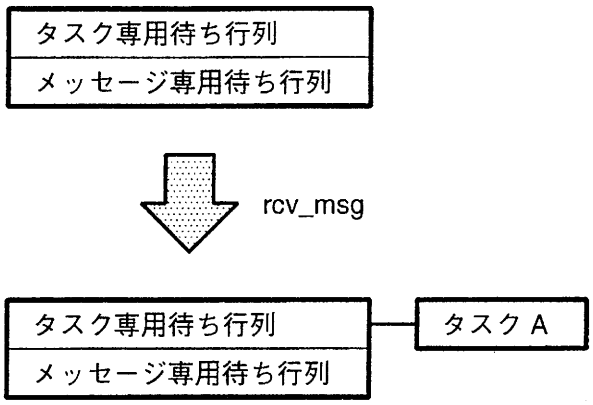


(2) タスクAが`rcv_msg`システム・コールを発行します。

このとき，RX830が管理しているメールボックスにはメッセージが送信されていません。そこで，RX830では，タスクAを`run`状態から`wait`状態(メッセージ待ち状態)へと遷移させたのち，対象メールボックスのタスク専用待ち行列にキューイングします。

図5-11に，`rcv_msg`システム・コール発行時のタスク専用待ち行列を示します。

図 5-11　メールボックスのタスク専用待ち行列



(3) タスク A がメッセージ待ち状態へと遷移したことにともない，タスク B が ready 状態から run 状態へと遷移します。

(4) タスク B が cre_mpl システム・コールを発行します。

これにより，メッセージを作成する際に必要となる領域を確保するためのユーザ・メモリ・プールが生成されます。

(5) タスク B が get_blk システム・コールを発行します。

これにより，メッセージを作成する際に必要となる領域 (メモリ・ブロック) が確保されます。

(6) タスク B がメッセージを作成します。

(7) タスク B が mbx_adr システム・コールを発行します。

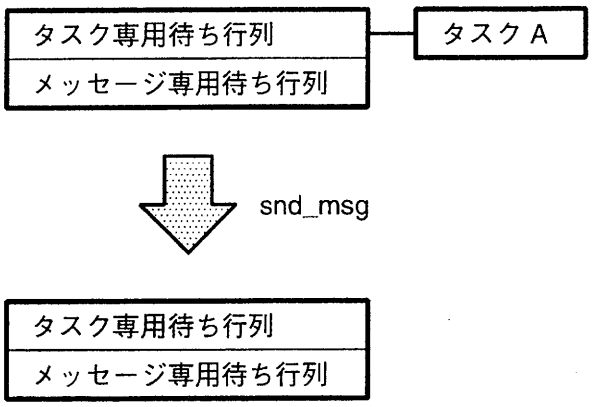
これにより，タスク B は，対象メールボックスのアクセス・アドレスを獲得し，対象メールボックスに対するダイナミックな操作が可能となります。

(8) タスク B が snd_msg システム・コールを発行します。

これにより，対象メールボックスのメッセージ専用待ち行列にメッセージがキューイングされ，タスク専用待ち行列の先頭にキューイングされているタスクの条件を満足したものとなります。そこで，RX830 では，タスク A をタスク専用待ち行列から外したのち，タスク A をメッセージ待ち状態から ready 状態へと遷移させます。

図 5-12に，snd_msg システム・コール発行時のタスク専用待ち行列を示します。

図 5-12　メールボックスのタスク専用待ち行列



(9) 現在 run 状態であるタスク B よりも優先度の高いタスク A が ready 状態へと遷移したことにともない，タスク A が ready 状態から run 状態へと遷移します。

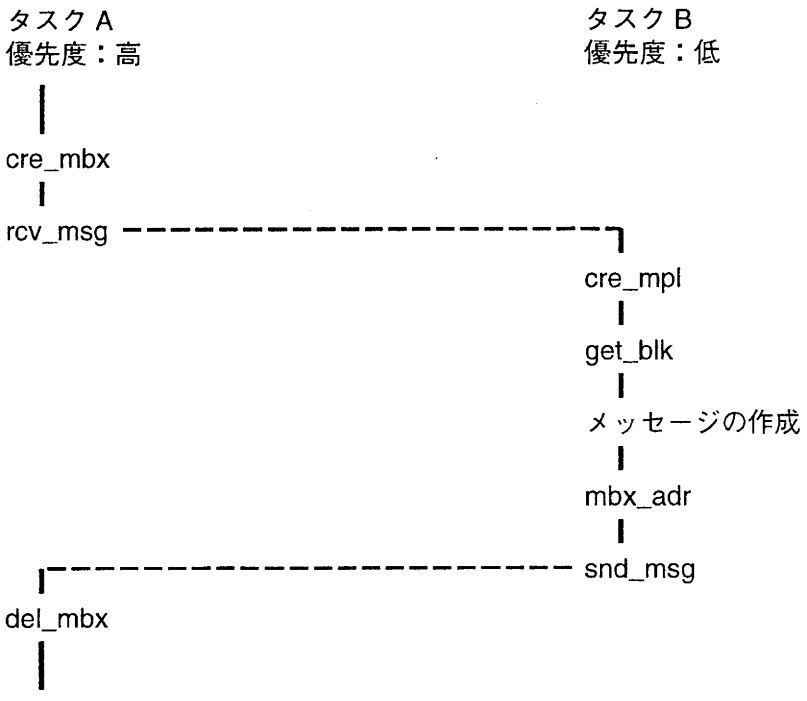
なお，タスク B は，run 状態から ready 状態へと遷移します。

(10) タスク A が del_mbx システム・コールを発行します。

これにより，タスク A とタスク B がタスク間の通信に使用したメールボックスが削除されます。

図 5-13に，(1) ～ (10) の処理の流れを示します。

図 5-13　メールボックスによるタスク間通信



第 6 章　割り込み処理管理

この章では，割り込み処理管理について説明しています。

6.1　概要

RX830 における割り込み処理管理では，

- 割り込みハンドラの登録
- 割り込みハンドラの前処理／後処理
- 割り込みハンドラからの復帰
- 割り込み許可レベルの設定

などといった処理が行われます。

6.2　割り込みハンドラ

割り込みハンドラは，割り込みが発生した際，ただちに起動される割り込み処理専用ルーチンであり，タスクとは独立したものとして扱われます。したがって，システム内で最高優先度を持つタスクが実行中であっても，その処理は中断され，割り込みハンドラに制御が移ります。

なお，RX830 では，割り込みが発生してから割り込みハンドラに制御が移るまでの応答性を考慮し，2 種類の割り込みハンドラ用インタフェースを提供しています。

- 直接起動割り込みハンドラ

  直接起動割り込みハンドラは，割り込みが発生した際，RX830 を介在させることなく起動される割り込み処理専用ルーチンです。

- 間接起動割り込みハンドラ

  間接起動割り込みハンドラは，割り込みが発生した際，RX830 による割り込み前処理 (レジスタの退避処理，スタックの切り替え処理など) を行わせたのちに起動される割り込み処理専用ルーチンです。

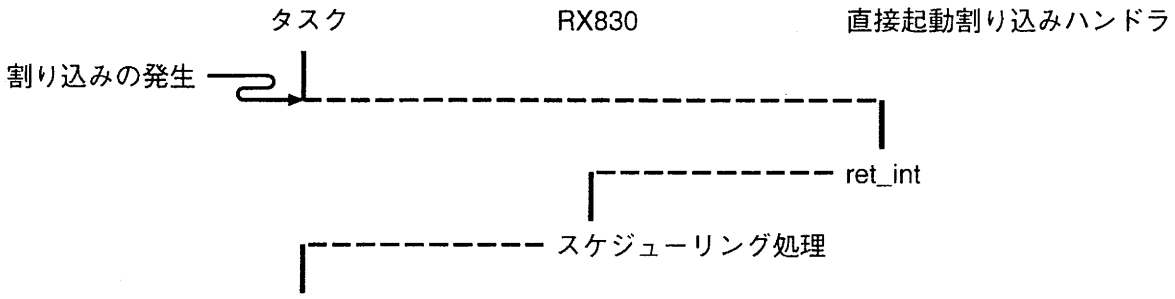
6.3 直接起動割り込みハンドラ

直接起動割り込みハンドラは，割り込みが発生した際，RX830 を介在させることなく起動される割り込み処理専用ルーチンです。

このため，ハードウエアの限界に近い高速な応答性が期待されます。

図 6-1に，直接起動割り込みハンドラの動作の流れを示します。

図 6-1　直接起動割り込みハンドラの動作の流れ



6.3.1 直接起動割り込みハンドラの登録

直接起動割り込みハンドラの登録は，割り込みが発生した際にプロセッサが制御を移すハンドラ・アドレスに直接起動割り込みハンドラを割り付ける，または，直接起動割り込みハンドラへの分岐命令を設定することにより行われます。

ただし，日本電気（株）製 V830 ファミリ™用 C コンパイラ CA830 を使用し，`#pragma rtos_interrupt` 指令を用いて直接起動割り込みハンドラを登録した場合は，ハンドラ・アドレスへの割り付け，または，分岐命令の設定といった処理を記述する必要がありません。

6.3.2 直接起動割り込みハンドラ内での処理

RX830 では，直接起動割り込みハンドラの起動に際し，いっさい関与していません。

このため，直接起動割り込みハンドラの処理を記述する際には，以下に示す注意事項があります。

(1) レジスタの退避／復帰

直接起動割り込みハンドラに制御が移った際のレジスタの内容は，割り込み発生時のものとなります。したがって，直接起動割り込みハンドラ内でレジスタを使用する場合は，直接起動割り込みハンドラの開始部分でレジスタの退避処理を，終了部分でレジスタの復帰処理を記述する必要があります。

ただし，日本電気（株）製 V830 ファミリ™用 C コンパイラ CA830 を使用し，`#pragma rtos_interrupt` 指令を用いて直接起動割り込みハンドラを登録した場合は，レジスタの退避／復帰に関する処理を記述する必要がありません。

(2) スタックの切り替え

直接起動割り込みハンドラに制御が移った際のスタックは，割り込み発生時のものとなります。したがって，割り込みハンドラ用スタックを使用する場合は，直接起動割り込みハンドラの開始部分で割り込みハンドラ用スタックへの切り替え処理を，終了部分で割り込み発生時のスタックへの切り替え処理を記述する必要があります。

(3) システム・コールの発行

直接起動割り込みハンドラ内で発行可能なシステム・コールの一覧を，以下に示します。

- 無条件で発行可能なシステム・コール

sta_tsk	rot_rdq	rsm_tsk	wup_tsk	set_flg	flg_adr
sig_sem	sem_adr	snd_msg	mbx_adr	def_int	ret_int
set_int	rel_blk	mpl_adr	set_tim	get_tim	get_ver
iret_wup	def_svc				

- 条件付きで発行可能なシステム・コール

以下のシステム・コールを発行する場合，タスク・アクセス・アドレスに「TSK_SELF」を指定することはできません。

chg_pri	tcb_adr	tsk_sts	can_wup	cyc_wup	can_cyc

以下のシステム・コールを発行する場合，命令オプションに「T_TEXC」を指定することはできません。

def_exc

(4) 直接起動割り込みハンドラからの復帰

直接起動割り込みハンドラからの復帰は，ret_int，または，iret_wup システム・コールを発行することにより行われます。

- ret_int システム・コール

直接起動割り込みハンドラから復帰する。

- iret_wup システム・コール

パラメータで指定されたタスクに対する起床要求の発行，および，直接起動割り込みハンドラからの復帰を行う。

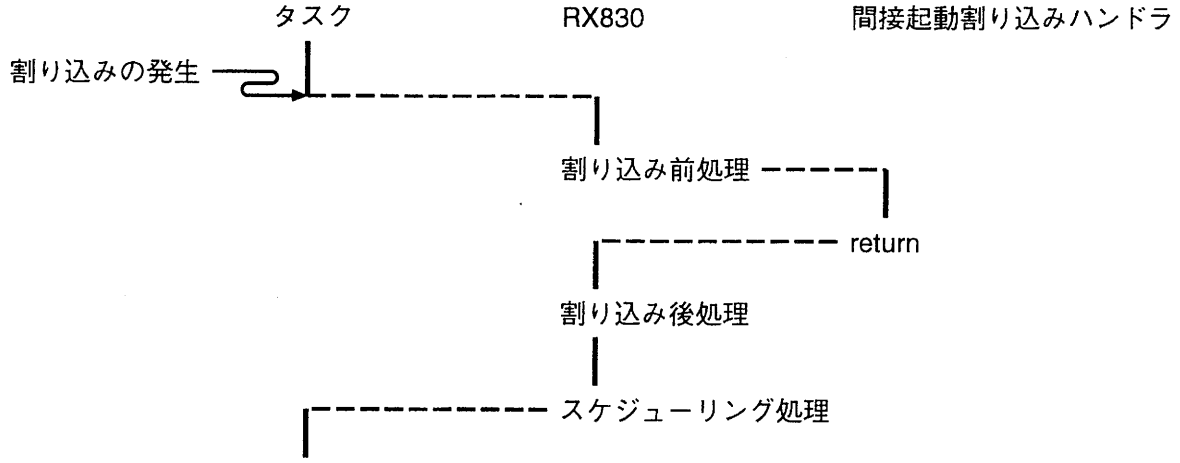
6.4 間接起動割り込みハンドラ

間接起動割り込みハンドラは，割り込みが発生した際，RX830 による割り込み前処理 (レジスタの退避処理，スタックの切り替え処理など) を行わせたのちに起動される割り込み処理専用ルーチンです。

このため，直接起動割り込みハンドラに比べて応答性の面では劣りますが，RX830 による割り込み前処理が行われるため，ハンドラ内での処理が簡素化されるという利点を持ちます。

図 6-2に，間接起動割り込みハンドラの動作の流れを示します。

図 6-2 間接起動割り込みハンドラの動作の流れ



6.4.1 間接起動割り込みハンドラの登録

間接起動割り込みハンドラの登録は，def_int システム・コールを発行することにより行われます。

- def_int システム・コール

  間接起動割り込みハンドラを登録する場合，def_int システム・コールを発行します。

  これにより，該当する割り込みレベルに対する間接起動割り込みハンドラの登録が行われます。

  以下に，def_int システム・コールを発行する際に必要となる情報 (パラメータ) を示します。

  - 割り込みレベル
  - 間接起動割り込みハンドラの先頭アドレス
  - 間接起動割り込みハンドラで使用する固有データ領域のアドレス (gp レジスタ値)

6.4.2 間接起動割り込みハンドラ内での処理

RX830では，割り込みが発生してから間接起動割り込みハンドラに制御を移す際，独自の割り込み前処理を行っています。また，間接起動割り込みハンドラから制御を戻す際にも，独自の割り込み後処理を行っています。

このため，間接起動割り込みハンドラの処理を記述する際には，以下に示す注意事項があります。

(1) レジスタの退避／復帰

RX830では，間接起動割り込みハンドラに制御を移す際，コンパイラの関数呼び出し規約に従ったレジスタの退避処理を行っています。また，間接起動割り込みハンドラから制御を戻す際にも，レジスタの復帰処理を行っています。

したがって，レジスタの退避／復帰に関する処理を記述する必要がありません。

(2) スタックの切り替え

RX830では，間接起動割り込みハンドラに制御を移す際，割り込みハンドラ用スタックへの切り替え処理を行っています。また，間接起動割り込みハンドラから制御を戻す際にも，割り込み発生時のスタックへの切り替え処理を行っています。

したがって，スタックの切り替えに関する処理を記述する必要がありません。

(3) システム・コールの発行

間接起動割り込みハンドラ内で発行可能なシステム・コールの一覧を，以下に示します。

- 無条件で発行可能なシステム・コール

sta_tsk	rot_rdq	rsm_tsk	wup_tsk	set_flg	flg_adr
sig_sem	sem_adr	snd_msg	mbx_adr	def_int	set_int
rel_blk	mpl_adr	set_tim	get_tim	get_ver	def_svc

- 条件付きで発行可能なシステム・コール

以下のシステム・コールを発行する場合，タスク・アクセス・アドレスに「TSK_SELF」を指定することはできません。

chg_pri	tcb_adr	tsk_sts	can_wup	cyc_wup	can_cyc

以下のシステム・コールを発行する場合，命令オプションに「T_TEXC」を指定することはできません。

def_exc

(4) 間接起動割り込みハンドラからの復帰

間接起動割り込みハンドラからの復帰は，**return** 命令を発行することにより行われます。

- `return(0x0);`

  間接起動割り込みハンドラから復帰する。

- `return(タスクのアクセス・アドレス);`

  パラメータで指定されたタスクに対する起床要求の発行，および，間接起動割り込みハンドラからの復帰を行う。

6.5 割り込みとスケジューリング

RX830 では，割り込みハンドラ内でタスクのスケジューリング処理が必要なシステム・コール (**chg_pri**，**wup_tsk**，**set_flg**，**sig_sem** システム・コールなど) が発行された場合，待ち行列へのキュー操作などといった処理を行うだけであり，実際のスケジューリング処理は，割り込みハンドラからの復帰処理 (**ret_int** システム・コール，**return** 命令の発行など) まで遅延され，一括して行うようにしています。

6.6 割り込み許可レベルの設定

割り込み許可レベルの設定は，**set_int** システム・コールを発行することにより行われます。

- **set_int** システム・コール

  割り込み許可レベルを設定する場合，**set_int** システム・コールを発行します。

  これにより，V830 ファミリ ™ が持つ 16 種類の割り込み要因をレベル指定で許可，禁止することができます。

6.7 ノンマスカブル割り込み

ノンマスカブル割り込みは，割り込み優先順位の対象外となっているため，すべての割り込みに優先して受け付けられます。また，プロセッサを割り込み禁止状態 (PSW の ID フラグをセット) にしても受け付けられる割り込みです。

このため，RX830 の処理中，および，割り込みハンドラの処理中であっても，ノンマスカブル割り込みは受け付けられることになります。

そこで，RX830 では，ノンマスカブル割り込みハンドラ内でシステム・コールを発行した場合，その動作を保証していません。

6.8　クロック割り込み

RX830では，ハードウエア・タイマにより一定周期で発生するクロック割り込みを利用して，時間管理を行っています。

つまり，クロック割り込みが発生した際には，時間管理用割り込みハンドラ(クロック・ハンドラ)が起動され，システム・クロックの更新，タスクの遅延起床，指定時刻起床，周期起床，タイムアウトなどといった時間に関連した処理が行われます。

なお，時間管理についての詳細は，「第9章　時間管理」を参照してください。

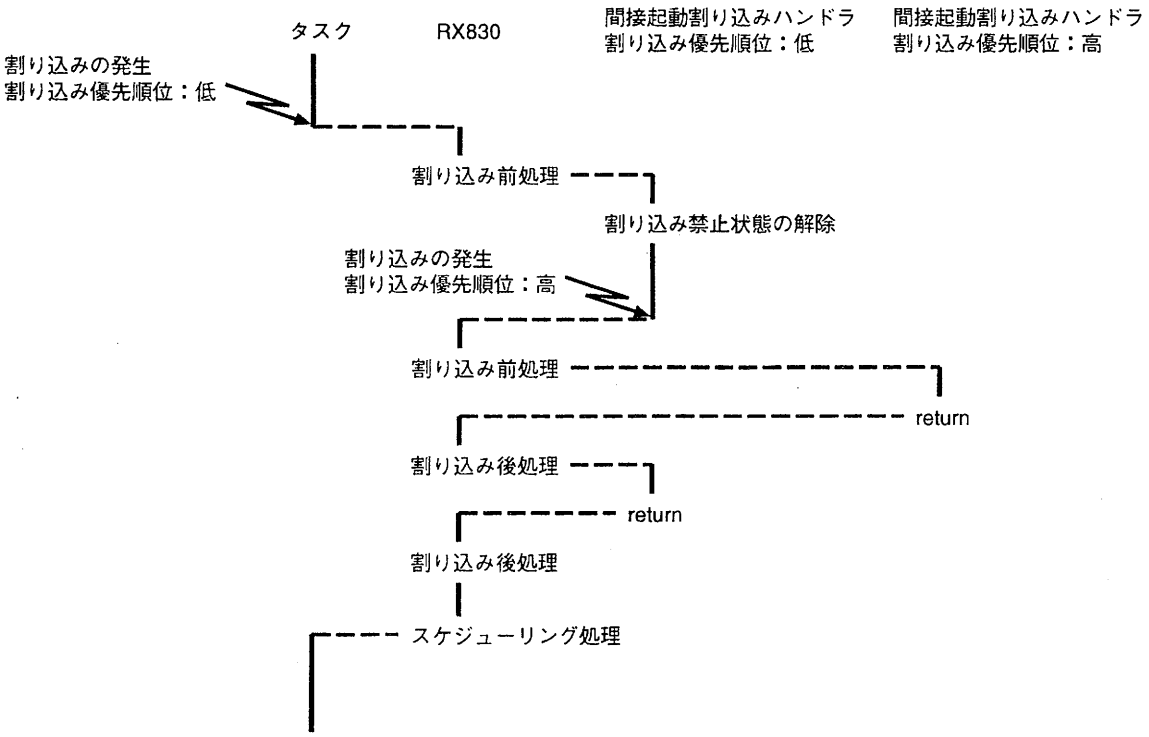
6.9　多重割り込み

RX830では，割り込みハンドラ内で再び割り込みが発生することを「**多重割り込み**」と呼びます。

ただし，割り込みハンドラは，割り込み禁止状態(PSWのIDフラグをセット)でその処理が開始されるため，多重割り込みを受け付けるためには，割り込みハンドラ内で割り込み禁止状態を解除する処理を記述する必要があります。

図6-3に，多重割り込み発生時の動作の流れを示します。

図 6-3　多重割り込み発生時の動作の流れ



第 7 章　例外処理管理

この章では，例外処理管理について説明しています。

7.1　概要

RX830 における例外処理管理では，

- 例外ハンドラの登録／登録解除
- 例外ハンドラの起動
- ディフォールト処理の登録

などといった処理が行われます。

7.2　例外の種類

RX830 では，例外を CPU 例外とシステム・コール例外の 2 種類に分けて管理しています。

(1) CPU 例外

CPU 例外は，処理プログラム内の命令実行に関係して発生したエラー (ゼロ除算，不正命令コード，トラップ命令など) を対象としています。

表 7-1に，CPU 例外の詳細を示します。

表 7-1　CPU 例外

例外コード	例外発生要因
0xff80	ゼロ除算
0xff90	不正命令コード
0xffan	トラップ命令 (パラメータが 0x0n)
0xffbn	トラップ命令 (パラメータが 0x1n)
注 1	二重例外
−	致命的例外

注 1　二重例外の要因となった例外コードとなります。

(2) システム・コール例外

システム・コール例外は，処理プログラム内で発行されたシステム・コールに関係して発生したエラー (不正パラメータ，不正オブジェクト，異常終了など) を対象としています。

表 7-2に，システム・コール例外の詳細を示します。

表 7-2　システム・コール例外

例外コード		例外発生要因
マクロ	数値	
TE_MEM	0x01	必要なメモリ領域が確保できない
TE_UDF	0x02	未定義のシステム・コールを発行した
TE_CTX	0x03	システム・コールを発行する状態が不正である
TE_OBJ	0x04	指定したアクセス・アドレスが正しいオブジェクトを示していない
TE_AA	0x05	指定したアクセス・アドレスが不正である
TE_PA	0x06	指定したパケット・アドレスが不正である
TE_MA	0x07	指定したメッセージ・アドレス，メモリ・ブロック・アドレスが不正である
TE_SA	0x08	定義したタスク，割り込みハンドラの開始アドレスが不正である
TE_EXS	0x09	同じ ID 番号のオブジェクトがすでに存在する
TE_NOEXS	0x0a	指定したオブジェクトは存在しない
TE_NODMT	0x0b	指定したタスクが dormant 状態ではない
TE_NOSUS	0x0c	指定したタスクが suspend 状態ではない
TE_DMT	0x0d	指定したタスクが dormant 状態である
TE_IDZR	0x0e	指定した ID 番号が 0 である
TE_IDOVR	0x0f	指定した ID 番号が範囲外である
TE_TMOUT	0x10	指定した時間が経過した
TE_QOVR	0x11	カウンタが許容範囲を越えた
TE_SELF	0x12	自タスクを指定した
TE_DLT	0x13	待ち行列を管理しているオブジェクトが削除された
TE_OPT	0x14	定義されていないシステム・コール・オプションを指定した
TE_WEVF	0x15	指定したイベント・フラグはすでに使用されている
TE_NOTMR	0x16	タイマ処理用の領域が確保できなかった
TE_TPRI	0x17	指定した優先度が大きすぎる
TE_IPRI	0x18	指定した割り込みレベルが不正である
TE_NOCYC	0x19	指定したタスクには周期起床が指定されていない
TE_MPLSZ	0x1a	指定したユーザ・メモリ・プールのサイズが大きすぎる
TE_SYS	0x1e	ニュークリアス・メモリ・プール，または，スタティックに生成したユーザ・メモリ・プールを削除しようとした
TE_PAR	0x1f	指定したパラメータが不正である
TE_SVC	0x40	指定した機能コード番号が不正である
TE_STSK	0x81	システム・タスクをアクセスした

7.3 例外ハンドラ

例外ハンドラは，例外が発生した際に起動される例外処理専用ルーチンであり，例外が発生したタスク／非タスクの延長として位置付けられています。

なお，RX830では，発生した例外の種類により，2種類の例外ハンドラ用インタフェースを提供しています。

- CPU例外ハンドラ

  CPU例外ハンドラは，CPU例外が発生した際に起動される例外処理専用ルーチンです。

  なお，RX830では，CPU例外が発生した際，図7-1に示したCPU例外情報をCPU例外ハンドラへの引き数として渡しています。

図7-1 CPU例外情報

```
struct  t_exccpuinfo {
    int             cpuaa;    /* タスク・アクセス・アドレス     (注1) */
    char            *cpupc;   /* 例外ハンドラからの戻りアドレス (注2) */
    int             cpupsw;   /* 例外発生時のPSW                      */
    unsigned short  cpucode;  /* 例外コード                           */
};
```

注1 非タスク内でCPU例外が発生した場合，cpuaaには0x0が設定されます。

注2 cpupcに設定される値がCPU例外発生時のCurrentがNextかは，CPU例外の種類により異なります。

- システム・コール例外ハンドラ

  システム・コール例外ハンドラは，システム・コール例外が発生した際に起動される例外処理専用ルーチンです。

  なお，RX830では，システム・コール例外が発生した際，図7-2に示したシステム・コール例外情報をシステム・コール例外ハンドラへの引き数として渡しています。

図7-2 システム・コール例外情報

```
struct  t_excsysinfo {
    int             sysaa;    /* タスク・アクセス・アドレス     (注1) */
    char            *syspc;   /* 例外ハンドラからの戻りアドレス (注2) */
    unsigned short  syserr;   /* 例外コード                           */
    unsigned short  sysno;    /* 機能コード                           */
};
```

注1 非タスク内でシステム・コール例外が発生した場合，sysaaには0x0が設定されます。

注2 syspcに設定される値は，システム・コール例外発生時のNextとなります。

7.4 例外ハンドラの登録／登録解除

例外ハンドラの登録／登録解除は，def_exc システム・コールを発行することにより行われます。

- def_exc システム・コール

  例外ハンドラを登録／登録解除する場合，def_exc システム・コールを発行します。

  これにより，該当する起動条件に対応した例外ハンドラの登録／登録解除が行われます。

  以下に，def_exc システム・コールを発行する際に必要となる情報（パラメータ）を示します。

  - 例外ハンドラの起動条件
  - 例外ハンドラの先頭アドレス
  - 例外ハンドラで使用する固有データ領域のアドレス（gp レジスタ値）

  注意 例外ハンドラの先頭アドレスに 0x0 を指定した場合は，すでに登録されている例外ハンドラの登録解除が行われます。

なお，RX830 では，def_exc システム・コールを発行する際のパラメータ（option）で，例外ハンドラの起動条件を指定することができます。

(1) 例外種別

例外種別を指定します。

- CPU 例外（T_CPU）

  CPU 例外が発生した際に起動されます。

- システム・コール例外（T_OS）

  システム・コール例外が発生した際に起動されます。

(2) 起動対象

例外ハンドラの起動対象を指定します。

- タスク固有（T_TEXC）

  本システム・コールを発行したタスク内で例外が発生した際に起動されます。

- 非タスク（T_NEXC）

  非タスク内で例外が発生した際に起動されます。

- 全タスク共通（T_CEXC）

  タスク内で例外が発生した際に起動されます。

  ただし，タスク固有の例外ハンドラが登録されていた場合は，タスクの固有の例外ハンドラが優先され，起動されます。

7.5 例外ハンドラからの復帰

例外ハンドラからの復帰は，return 命令を発行することにより行われます。

- `return(0x0);`

  例外ハンドラからの復帰を行う場合，return 命令を発行します。

  これにより，例外情報 (CPU 例外情報，システム・コール例外情報) 内の戻りアドレスで指定されたアドレスがプログラム・カウンタに設定されます。

7.6 ディフォールト処理

RX830 では，例外が発生した際，処理プログラム (タスク／非タスク) にユーザ定義の例外ハンドラが登録されているか否かをチェックしています。

このとき，ユーザ定義の例外ハンドラが登録されていた場合は，ユーザ定義の例外ハンドラを起動します。しかし，ユーザ定義の例外ハンドラが登録されていない場合には，RX830 が用意しているディフォールト処理が行われます。

なお，ディフォールト処理は，システム情報テーブル内で定義され，初期化処理 (ニュークリアス初期化部) において登録されます。

- タスク内 CPU 例外ディフォールト処理

  タスク内 CPU 例外ディフォールト処理は，タスク内で CPU 例外が発生した際，該当するユーザ定義の例外ハンドラが登録されていない場合に起動される例外ハンドラです。

  なお，タスク内 CPU 例外ディフォールト処理をシステム情報テーブルに定義する際，以下の中から処理を選択します。

`abort task`	：例外を発生させたタスクを強制的に終了させ，dormant 状態へと遷移させます。 なお，終了時処理ルーチンが登録されていた場合には，終了時処理ルーチンに制御が移ります。
`no operation`	：例外処理は行わず，エラー・トラップから復帰させます。
`stop system`	：システムを停止します。

- タスク内システム・コール例外ディフォールト処理

  タスク内システム・コール例外ディフォールト処理は，タスク内でシステム・コール例外が発生した際，該当するユーザ定義の例外ハンドラが登録されていない場合に起動される例外ハンドラです。

  なお，タスク内システム・コール例外ディフォールト処理をシステム情報テーブルに定義する際，以下の中から処理を選択します。

`abort task` ： 例外を発生させたタスクを強制的に終了させ，dormant 状態へと遷移させます。
なお，終了時処理ルーチンが登録されていた場合には，終了時処理ルーチンに制御が移ります。

`return error code` ： 例外処理は行わず，システム・コールの戻り値に例外コード (エラー・コード) を返します。

- 非タスク内 CPU 例外ディフォールト処理

非タスク内 CPU 例外ディフォールト処理は，非タスク内で CPU 例外が発生した際，該当するユーザ定義の例外ハンドラが登録されていない場合に起動される例外ハンドラです。

なお，非タスク内 CPU 例外ディフォールト処理をシステム情報テーブルに定義する際，以下の中から処理を選択します。

`no operation` ： 例外処理は行わず，エラー・トラップから復帰させます。
`stop system` ： システムを停止します。

- 非タスク内システム・コール例外ディフォールト処理

非タスク内システム・コール例外ディフォールト処理は，非タスク内でシステム・コール例外が発生した際，該当するユーザ定義の例外ハンドラが登録されていない場合に起動される例外ハンドラです。

なお，非タスク内システム・コール例外ディフォールト処理では，以下の処理が行われます。

`return error code` ： 例外処理は行わず，システム・コールの戻り値に例外コード (エラー・コード) を返します。

第 8 章　メモリ管理

この章では，メモリ管理について説明しています。

8.1　概要

RX830 では，システム情報テーブルに定義されたメモリ領域を，以下に挙げる 2 つの領域に分けて管理しています。

- ニュークリアス・メモリ・プール

  ニュークリアス・メモリ・プールは，ニュークリアス資源 (タスク管理ブロック，イベント・フラグ管理ブロック，セマフォ管理ブロックなど)，および，スタック領域 (タスク用のスタック領域，割り込みハンドラ用のスタック領域) を確保／解放する際に使用される領域です。

  なお，RX830 では，ニュークリアス・メモリ・プールとして，ニュークリアス・メモリ・プール #0，ニュークリアス・メモリ・プール #1 といった 2 種類の領域を生成することができます。

  – ニュークリアス・メモリ・プール #0

    ニュークリアス・メモリ・プール #0 は，ニュークリアス資源，および，スタック領域を確保／解放する際に使用される領域です。

    ただし，ニュークリアス・メモリ・プール #1 が存在する場合は，ニュークリアス資源を確保／解放する際に使用される領域となります。

  – ニュークリアス・メモリ・プール #1

    ニュークリアス・メモリ・プール #1 は，スタック領域を確保／解放する際に使用される領域です。

- ユーザ・メモリ・プール

  ユーザ・メモリ・プールは，ユーザの処理プログラム (タスク，割り込みハンドラなど) からダイナミックにメモリ領域の獲得／解放といった操作が可能な領域です。

8.2　ニュークリアス・メモリ・プールの生成

RX830 におけるニュークリアス・メモリ・プールの生成は，ニュークリアス・メモリ・プールを管理するための領域 (メモリ・プール管理ブロック)，および，ニュークリアス・メモリ・プールとして使用するための領域を確保／初期化し，RX830 の管理対象とすることです。

注意　RX830では，ニュークリアス・メモリ・プールを管理するための領域（メモリ・プール管理ブロック）をニュークリアス・メモリ・プールから確保しています。

以下に，ニュークリアス・メモリ・プールを生成する際の実現方法を示します。

- システム情報テーブル

  ニュークリアス・メモリ・プールは，システム情報テーブルに定義することにより，初期化処理（ニュークリアス初期化部）において，システム情報テーブルに定義したアドレスからニュークリアス・メモリ・プールとして使用する領域が確保され，RX830の管理対象となります。

8.3　ユーザ・メモリ・プールの生成／削除

RX830におけるユーザ・メモリ・プールの生成は，ユーザ・メモリ・プールを管理するための領域（メモリ・プール管理ブロック），および，ユーザ・メモリ・プールとして使用するための領域を確保／初期化し，RX830の管理対象とすることです。

また，RX830におけるユーザ・メモリ・プールの削除は，生成時に確保した領域（メモリ・プール管理ブロック，および，ユーザ・メモリ・プールとして使用する領域）を解放し，RX830の管理対象から除外することです。

注意　RX830では，ユーザ・メモリ・プールを管理するための領域（メモリ・プール管理ブロック）をニュークリアス・メモリ・プールから確保しています。

以下に，ユーザ・メモリ・プールを生成／削除する際の実現方法を示します。

- システム情報テーブル

  ユーザ・メモリ・プールをスタティックに生成する場合，システム情報テーブルに定義します。

  これにより，初期化処理（ニュークリアス初期化部）において，システム情報テーブルで定義したアドレスからユーザ・メモリ・プールとして使用するための領域が確保され，RX830の管理対象となります。

- `cre_mpl` システム・コール

  ユーザ・メモリ・プールをダイナミックに生成する場合，`cre_mpl` システム・コールを発行します。

  これにより，ユーザ・メモリ・プール0番からユーザ・メモリ・プールとして使用するための領域が確保され，RX830の管理対象となります。

- `del_mpl` システム・コール

  ユーザ・メモリ・プールを削除する場合，`del_mpl` システム・コールを発行します。

これにより，ユーザ・メモリ・プールは削除され，RX830 の管理対象から除外されます。

注意　RX830 では，スタティックに生成したユーザ・メモリ・プールを削除することは禁止されています。

8.4　メモリ・ブロックの獲得

メモリ・ブロックの獲得は，get_blk システム・コールを発行することにより行われます。

- get_blk システム・コール

  対象ユーザ・メモリ・プールからメモリ・ブロックを獲得する場合，get_blk システム・コールを発行します。

  ただし，本システム・コールを発行した際，対象ユーザ・メモリ・プールから要求する数のメモリ・ブロックを獲得することができなかった場合には，自タスクを対象ユーザ・メモリ・プールの待ち行列にキューイングしたのち，run 状態から ready 状態へと遷移させています。

なお，RX830 では，get_blk システム・コールを発行する際のパラメータ (option) でタイムアウトを指定することができます。

(1) タイムアウト指定なし (T_NOOPT)

待ち条件を満足するまで，wait 状態 (メモリ・ブロック待ち状態) へと遷移したままとなります。

(2) タイムアウト指定あり (T_TMOUT)

パラメータで指定された時間が経過した際には，wait 状態 (メモリ・ブロック待ち状態) から ready 状態へと遷移します。

8.5　メモリ・ブロックの解放

メモリ・ブロックの解放は，rel_blk システム・コールを発行することにより行われます。

- rel_blk システム・コール

  対象ユーザ・メモリ・プールにメモリ・ブロックを解放する場合，rel_blk システム・コールを発行します。

  ただし，本システム・コールを発行した際，対象ユーザ・メモリ・プールの待ち行列にタスクがキューイングされており，かつ，先頭タスクの待ち条件を満足させた場合には，該当タスクを待ち行列から外したのち，wait 状態 (メモリ・ブロック待ち状態) から ready 状態へと遷移させています。

第 9 章　時間管理

この章では，時間管理について説明しています。

9.1　概要

RX830 では，ハードウエア・タイマにより一定周期で発生するクロック割り込みを利用して，時間管理を行っています。

つまり，クロック割り込みが発生した際には，時間管理用割り込みハンドラ(クロック・ハンドラ)が起動され，システム・クロックの更新，タスクの遅延起床，指定時刻起床，周期起床，タイムアウトなどといった時間に関連した処理が行われます。

9.2　システム・クロック

システム・クロックは，RX830 が時間管理を行う際に使用する時刻 (48 ビット幅，単位： msec) を保持したソフトウエア・タイマです。

なお，システム・クロックは，初期化処理で 0x0 に設定されたのち，時間管理用割り込みハンドラで基本クロック周期(システム情報テーブルで定義) を単位として更新されます。

以下に，システム・クロックの設定／読み出しを行う際の実現方法を示します。

- set_tim システム・コール

  システム・クロックに時刻を設定する場合，set_tim システム・コールを発行します。

  これにより，システム・クロックは，パラメータで指定された時刻となります。

- get_tim システム・コール

  システム・クロックの時刻を読み出す場合，get_tim システム・コールを発行します。

  これにより，システム・クロックの現時刻がパラメータで指定された領域に格納されます。

9.3　タイマ・オペレーション機能

リアルタイム処理では，あるタスクの処理を一定時間だけ中断したり，一定の周期で実行させたりといった時間と同期した機能が必要となります。

RX830 では，このような機能を「**タイマ・オペレーション機能**」と呼びます。なお，RX830 では，タイマ・オペレーション機能としてタスクの遅延起床，指定時刻起床，周期起床，タイムアウトを提供しています。

9.4 遅延起床

RX830 における遅延起床は，指定された時間が経過するまでの間，タスクを wait 状態 (時限待ち状態) へと遷移させ，時間が経過した際には，タスクを時限待ち状態から ready 状態へと遷移させるものです。

以下に，タスクの遅延起床を行う際の実現方法を示します。

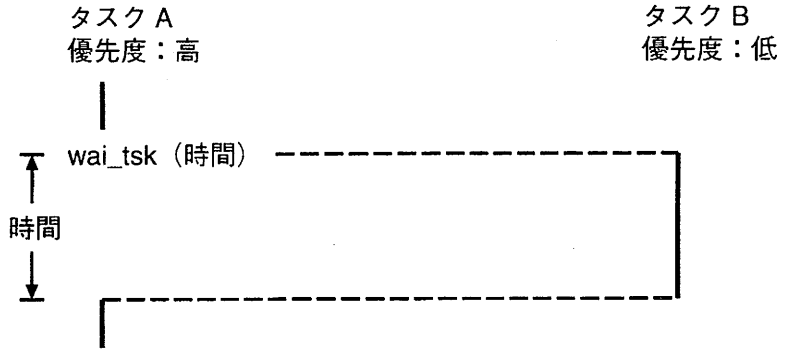
- wai_tsk システム・コール

  タスクの遅延起床を行う場合， wai_tsk システム・コールを発行します。

  これにより，本システム・コールを発行したタスクは，パラメータで指定された時間が経過するまでの間， run 状態から wait 状態 (時限待ち状態) へと遷移します。なお，パラメータで指定された時間が経過した際には，時限待ち状態から ready 状態へと遷移します。

  図 9-1に， wai_tsk システム・コールを発行した際の処理の流れを示します。

図 9-1 タスクの遅延起床



9.5 指定時刻起床

RX830 における指定時刻起床は，指定された時刻に達した際，起床要求を発行し，タスクを wait 状態 (起床待ち状態) から ready 状態へと遷移させるものです。

以下に，タスクの指定時刻起床を行う際の実現方法を示します。

- cyc_wup システム・コール

  タスクの指定時刻起床を行う場合， cyc_wup システム・コールを発行します。

  これにより，対象タスクは，パラメータで指定された時刻に達した際には， wait 状態 (起床待ち状態) から ready 状態へと遷移します。

  注意 cyc_wup システム・コールを発行する際，命令オプションには T_ABS を，起床回数には 0x1 を指定します。

9.6 周期起床

RX830 における周期起床は，指定された周期で起床要求を発行し，タスクを周期的に wait 状態 (起床待ち状態) から ready 状態へと遷移させるものです。

以下に，タスクの周期起床を行う際の実現方法を示します。

- cyc_wup システム・コール

  タスクの周期起床を行う場合， cyc_wup システム・コールを発行します。

  これにより，対象タスクは，パラメータで指定された周期に達した際には， wait 状態から ready 状態へと遷移します。

9.7 タイムアウト

RX830 におけるタイムアウトは，要求する条件が即時に成立しなかった場合，指定された時間が経過するまでの間，タスクを wait 状態へと遷移させ，指定された時間が経過した際には，タスクを wait 状態から ready 状態へと遷移させるものです。

以下に，タイムアウト指定が可能なシステム・コールを示します。

- wai_flg システム・コール
- wai_sem システム・コール
- rcv_msg システム・コール
- get_blk システム・コール

第 10 章　初期化処理

この章では，初期化処理について説明しています。

なお，初期化処理についての詳細は，「RX830 ユーザーズ・マニュアル インストレーション編」を参照してください。

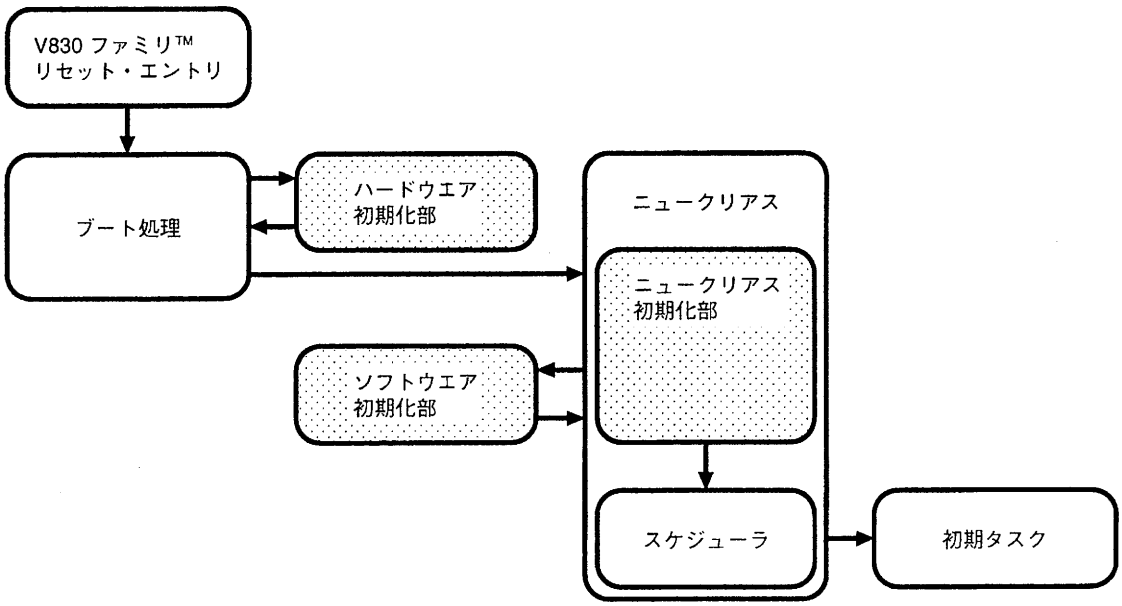
10.1　概要

初期化処理は，RX830 が動作するうえで必要となるハードウエアの初期化処理，ニュークリアスの初期化処理，および，ソフトウエアの初期化処理から構成されています。

なお，RX830 では，システムが起動した際，まず最初に行われる処理が，初期化処理となります。

図 10-1に，初期化処理の流れを示します。

図 10-1　初期化処理の流れ

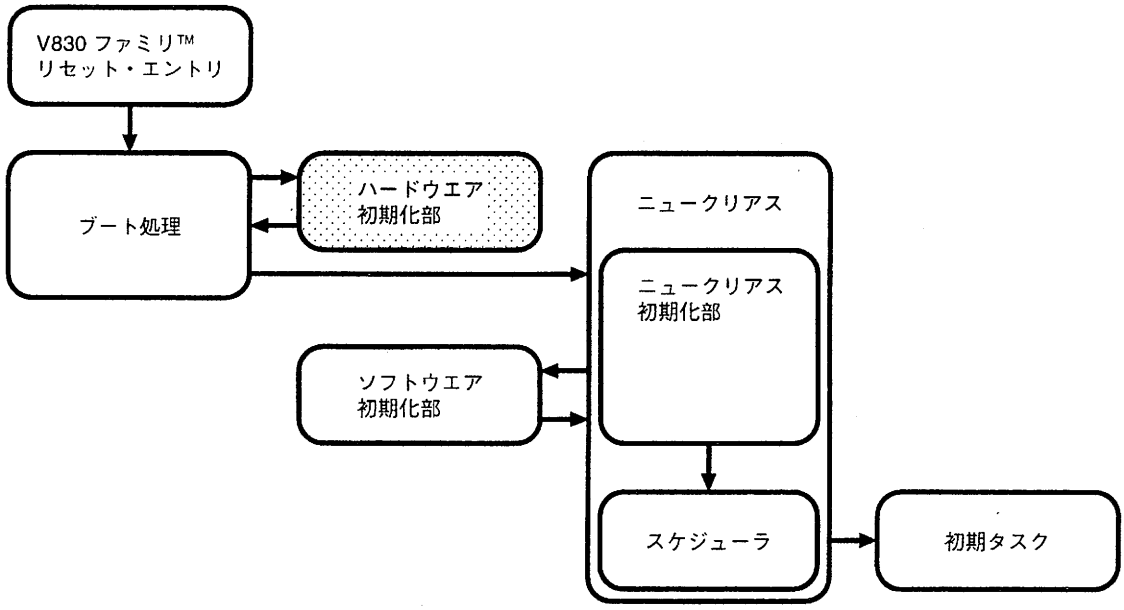


10.2 ハードウエア初期化部

ハードウエア初期化部は，実行環境上の周辺コントローラを初期化するために用意された関数です。

図 10-2に，ハードウエア初期化部の位置付けを示します。

図 10-2 ハードウエア初期化部の位置付け



なお，標準添付のハードウエア初期化部では，次のような処理を行っています。

- 割り込みコントローラの初期化
- クロック・コントローラの初期化

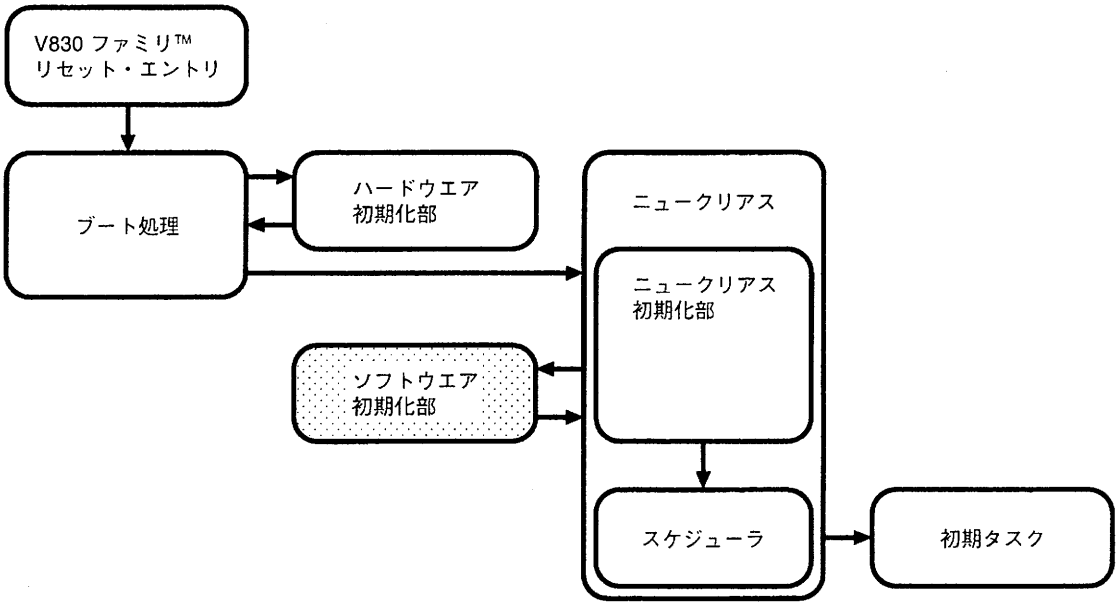
注意 標準添付のハードウエア初期化部は，ユーザの実行環境として，（株）電産製 V830 ファミリ ™ ボード [DVE-V830/20] を想定した処理が記述してあります。
このため，ユーザの実行環境が標準構成でない場合や，周辺コントローラのモード設定を変更する場合は，ハードウエア初期化部を書き替える必要があります。

10.3　ソフトウエア初期化部

ソフトウエア初期化部は，ユーザのソフトウエア環境を快適なものとするために用意された関数です。

図 10-3に，ソフトウエア初期化部の位置付けを示します。

図 10-3　ソフトウエア初期化部の位置付け



なお，標準添付のソフトウエア初期化部では，次のような処理を行っています。

- 初期化データのコピー
- クロック割り込みの許可

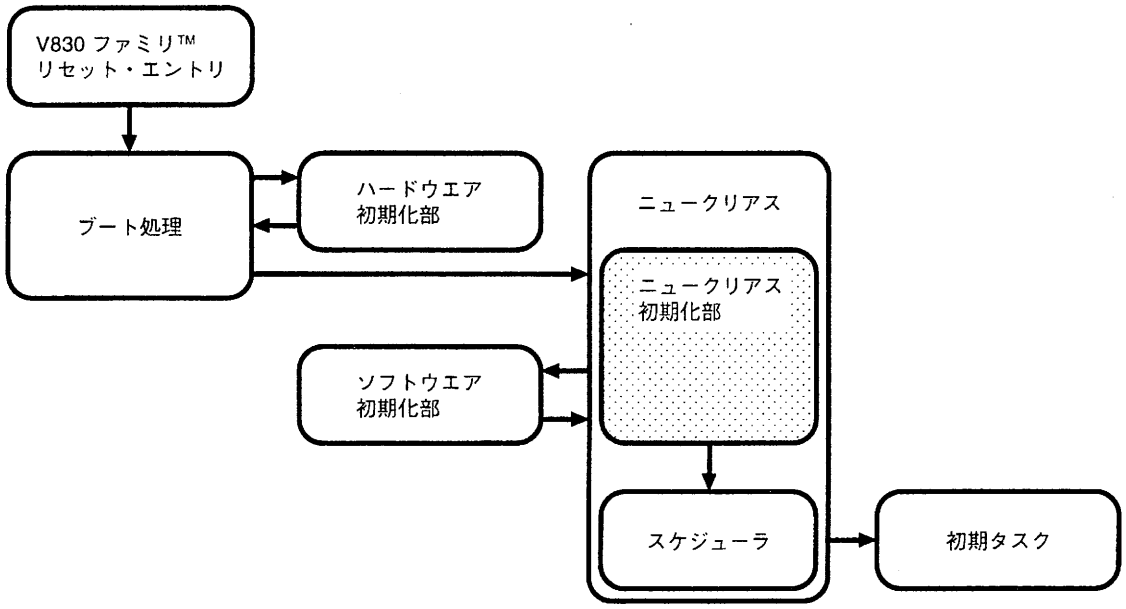
注意　標準添付のソフトウエア初期化部は，その処理内容をユーザのニーズにあわせて書き替えることが可能です。

10.4 ニュークリアス初期化部

ニュークリアス初期化部は，ユーザがコンフィグレーション時に作成したシステム情報テーブル内のデータをもとに，ニュークリアスの初期化を行います。

図 10-4に，ニュークリアス初期化部の位置付けを示します。

図 10-4 ニュークリアス初期化部の位置付け



なお，ニュークリアス初期化部では，次のような処理を行っています。

- メモリ・プールの生成
- 管理オブジェクトの生成
- ハンドラの登録
- 初期タスクの生成／起動

10.4.1 メモリ・プールの生成

システム情報テーブル内のニュークリアス・メモリ情報で指定されたデータをもとに，ニュークリアス・メモリ・プールの生成を行います。

なお，メモリ情報で指定されたデータをもとに，ユーザ・メモリ・プールの生成も，このときに行われます。

10.4.2 管理オブジェクトの生成

(1) オペレーティング・システム管理テーブルの生成

システム情報テーブル内のシステム情報で指定されたデータをもとに，オペレーティング・システム管理テーブルの生成／初期化を行います。

なお，オペレーティング・システム管理テーブルは，ニュークリアス・メモリ・プールの先頭から割り付けられます。

(2) 資源の生成

システム情報テーブル内のシステム情報で指定されたデータをもとに，資源 (周期起床用時間管理ブロック，割り込みハンドラ用スタックなど) の生成／初期化を行います。

なお，各種管理ブロックは，ニュークリアス・メモリ・プール内のオペレーティング・システム管理テーブルに続けて割り付けられ，割り込みハンドラ用スタックは，ニュークリアス・メモリ・プールの末尾から割り付けられます。

10.4.3 ハンドラの登録

システム情報テーブル内の例外処理情報で指定されたデータをもとに，ディフォールトの例外ハンドラの登録を行います。

10.4.4 初期タスクの生成／起動

システム情報テーブル内のタスク情報で指定されたデータをもとに，初期タスクの生成／起動を行います。

なお，システム・タスクの生成／起動も，このときに行われます。

第 11 章　インタフェース・ライブラリ

この章では，インタフェース・ライブラリについて説明しています。

なお，インタフェース・ライブラリについての詳細は，「RX830 ユーザーズ・マニュアル インストレーション編」を参照してください。

11.1　概要

アプリケーション・プログラム（タスク，割り込みハンドラなど）をＣ言語で記述した場合，システム・コールの発行形式，および，拡張SVCハンドラの呼び出し形式は，外部関数形式となります。しかし，ニュークリアスが理解可能な発行形式（ニュークリアス発行形式）と外部関数形式には，相違があります。

そこで，外部関数形式で発行されたシステム・コール，または，外部関数形式で呼び出された拡張SVCハンドラをニュークリアス発行形式に変換し，アプリケーション・プログラムとニュークリアスの仲介役を行うインタフェース・ライブラリが必要となります。

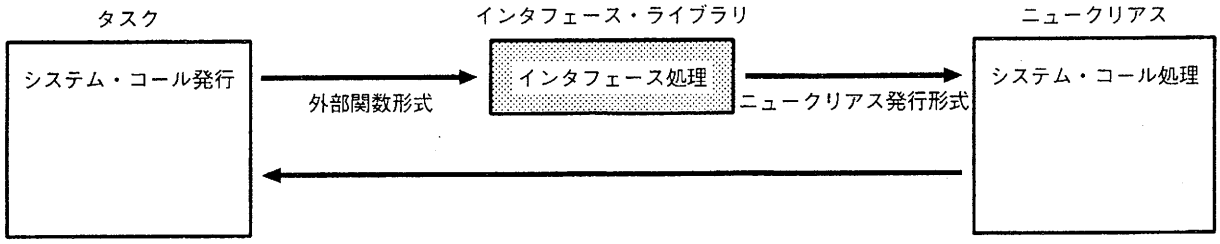
つまり，インタフェース・ライブラリは，アプリケーション・プログラムとニュークリアスの中間に位置し，ニュークリアスが対応した処理を行ううえで必要となる各種情報を設定したのち，ニュークリアスに制御を移す機能を持ちます。

なお，RX830が提供するインタフェース・ライブラリは，クロス・ツールとして日本電気（株）製V830ファミリ™用Ｃコンパイラ CA830，または，Green Hills Software Inc.製Ｃクロス V830ファミリ™コンパイラ CCV830の使用を推奨し，同クロス・ツール用に設計されています。したがって，上記以外のクロス・ツールを使用する場合，インタフェース・ライブラリの変更が必要となります。

また，RX830が提供するシステム・コール用インタフェース・ライブラリには，システム・コールのパラメータをチェックするインタフェース・ライブラリ（パラメータ・チェック機能あり）と，システム・コールのパラメータをチェックしないインタフェース・ライブラリ（パラメータ・チェック機能なし）の２種類があり，用途に応じて使い分けることができます。

図 11-1に，インタフェース・ライブラリの位置付けを示します。

図 11-1　インタフェース・ライブラリの位置付け



保守／廃止

第 12 章　システム・コール

この章では，RX830 が提供するシステム・コールについて説明しています。

12.1　概要

ユーザはシステム・コール (RX830 が提供する各種サービス・ルーチン) を利用することにより，タスク管理，同期通信管理，割り込み処理管理などで RX830 が直接管理している資源を間接的に操作することが可能となります。

なお，RX830 では，49 種類のシステム・コールを提供していますが，その機能により，以下に示す 9 グループに分類することができます。

(1) タスク管理システム・コール　20 個

このグループには，タスクの状態を操作する機能，タスクに従属した同期操作を行う機能などが含まれます。

cre_tsk	sta_tsk	del_tsk	def_ext	ext_tsk
exd_tsk	abo_tsk	ter_tsk	chg_pri	rot_rdq
tcb_adr	tsk_sts	sus_tsk	rsm_tsk	slp_tsk
wai_tsk	wup_tsk	can_wup	cyc_wup	can_cyc

(2) 同期通信管理システム・コール　15 個

このグループには，待ち合わせ機能，排他制御機能，タスク間通信機能などが含まれます。

cre_flg	del_flg	set_flg	wai_flg	flg_adr
cre_sem	del_sem	sig_sem	wai_sem	sem_adr
cre_mbx	del_mbx	snd_msg	rcv_msg	mbx_adr

(3) 割り込み処理管理システム・コール　3 個

このグループには，間接起動割り込みハンドラを登録する機能，直接起動割り込みハンドラから復帰する機能，マスカブル割り込みの割り込み許可レベルを設定する機能などが含まれます。

def_int	ret_int	set_int

(4) 例外処理管理システム・コール　2 個

このグループには，例外ハンドラを登録／登録解除する機能などが含まれます。

def_exc

保守／廃止

(5) メモリ管理システム・コール　5個

このグループには，ユーザ・メモリ・プールを生成／削除する機能，メモリ・ブロックを獲得／解放する機能などが含まれます。

cre_mpl　del_mpl　get_blk　rel_blk　mpl_adr

(6) 時間管理システム・コール　2個

このグループには，システム・クロックに時刻を設定する機能，システム・クロックの時刻を読み出す機能などが含まれます。

set_tim　get_tim

(7) バージョン管理システム・コール　1個

このグループには，RX830のバージョン情報を獲得する機能などが含まれます。

get_ver

(8) 複合システム・コール　1個

このグループには，直接起動割り込みハンドラからの復帰と起床要求の発行を複合した機能などが含まれます。

iret_wup

(9) 外核システム・コール　1個

このグループには，拡張SVCハンドラを登録／登録解除する機能などが含まれます。

def_svc

保守／廃止

12.2 機能コード

RX830では，ITRON1仕様に準拠した機能コードの割り当てが行われています。

表12-1に，機能コードの割り当てを示します。

なお，0xe0 ～ 0xffの機能コードについては，ユーザが記述した拡張SVCハンドラに機能コードを割り当てる際に使用します。

表12-1 機能コードの割り当て

機能コード	分　　類
0x00 ～ 0x32	一般的なシステム・コール
0x33 ～ 0x37	システム予約
0x38	複合システム・コール
0x39 ～ 0x3f	システム予約
0x40	拡張システム・コール
0x41 ～ 0xdf	システム予約
0xe0 ～ 0xff	ユーザ用

12.3 エラー・コード

RX830では，ITRON1仕様に準拠したエラー・コードの割り当てが行われています。

表12-2に，エラー・コードの割り当てを示します。

なお，0x100以上のエラー・コードについては，ユーザ独自のエラー・チェックをインタフェース・ライブラリに追加した際などに使用します。

表12-2 エラー・コードの割り当て

エラー・コード	用　　途
0x00	正常終了
0x01 ～ 0x1f	一般的なエラー
0x20 ～ 0x3f	システム予約
0x40	拡張システム・コール用のエラー
0x41 ～ 0x7f	システム予約
0x80	2重例外発生のエラー
0x81	アイドル・タスク・アクセスのエラー
0x82 ～ 0xff	システム予約
0x100以上	ユーザ用

12.4 パラメータ

12.4.1 パラメータ値の範囲

RX830が提供するシステム・コールのパラメータには，値に範囲のあるもの，ある値をシステムで予約しているものなどがあります。

表12-3に，パラメータ値の範囲を示します。

表 12-3 パラメータ値の範囲

パラメータの種類	範　　囲	システム・コール
タスクID番号	0x1 ～ 0xefff	cre_tsk, tcb_adr, tsk_sts
イベント・フラグID番号	0x1 ～ 0xefff	cre_flg, flg_adr
セマフォID番号	0x1 ～ 0xefff	cre_sem, sem_adr
メールボックスID番号	0x1 ～ 0xefff	cre_mbx, mbx_adr
メモリ・プールID番号	0x0 ～ 0xefff　(注1)	cre_mpl, mpl_adr
優先度	0x1 ～ 0xf8　(注2)	cre_tsk, chg_pri, rot_rdq
ユーザ・スタック・サイズ	0x0 ～ 0xffffffff	cre_tsk
時刻	0x0 ～ 0xffffffffffff	cyc_wup
周期起床時間間隔	0x1 ～ 0xffffffff	cyc_wup
周期起床回数	0x0 ～ 0xffff　(注3)	cyc_wup
タイムアウト値	0x0 ～ 0xffffffff	wai_tsk, wai_flg, wai_sem, rcv_msg, get_blk
セマフォ・カウント値	0x0 ～ 0xffffffff	cre_sem, sig_sem, wai_sem
サスペンド要求回数	0x1 ～ 0xff	sus_tsk
起床要求回数	0x1 ～ 0xff	wup_tsk
ブロック数	0x1 ～ 0xffff	get_blk
拡張システム・コールの機能コード	0xe0 ～ 0xff	def_svc
割り込み許可レベル	0x0 ～ 0x10	set_int

注1　0x0は初期化処理(ニュークリアス初期化部)において生成されるユーザ・メモリ・プールに割り当てられます。このため，cre_mplシステム・コールで指定可能なメモリ・プールID番号は0x1 ～ 0xefffとなります。

注2　システム情報テーブルにおいて優先度の指定可能範囲を制限することが可能です。

注3　0x0は対象タスクがdormant状態に遷移するまで起床要求を発行し続けます。

12.4.2 命令オプション

RX830が提供しているシステム・コールの命令オプションには，その値をマクロ定義しているものがあります。

表12-4に，命令オプションの一覧を示します。

表 12-4 命令オプションの一覧

命令オプション		意　　味
マクロ	数値	
T_POOL0	0x0	ニュークリアス・メモリ・プール #0 からスタック領域を確保する
T_POOL1	0x2	ニュークリアス・メモリ・プール #1 からスタック領域を確保する
T_NOOPT	0x0	オプションを使用しない
T_FRCRSM	0x1	サスペンド要求をすべて解除する
T_ABS	0x0	絶対時間で指定する
T_REL	0x1	相対時間で指定する
T_OR	0x0	論理和をとる
T_AND	0x1	論理積をとる
T_REP	0x2	置き換える
T_EXOR	0x3	排他的論理和をとる
T_TMOUT	0x1	タイムアウトを指定する
T_ANDW	0x0	AND 待ち
T_ORW	0x2	OR 待ち
T_RESET	0x4	リセットを指定する
T_TPRI	0x0	待ちタスクのキューイングは優先度順
T_TFIFO	0x1	待ちタスクのキューイングは FIFO
T_MPRI	0x0	メッセージのキューイングは優先度順
T_MFIFO	0x2	メッセージのキューイングは FIFO
T_CPU	0x0	CPU 例外
T_OS	0x1	システム・コール例外
T_TEXC	0x0	各タスク固有の例外処理
T_NEXC	0x2	非タスク部分の例外処理
T_CEXC	0x4	全タスク共通の例外処理

12.5 システム・コールの拡張

RX830では，システム・コールを拡張(ユーザが記述した関数を拡張システム・コールとしてニュークリアスに登録)することができます。

なお，拡張システム・コールとして登録する関数に制限はなく，標準システム・コール(RX830が提供するシステム・コール)を含ませることも可能です。ただし，タスク状態からのみ発行可能な標準システム・コールを含ませた場合には，拡張システム・コールの発行状態が「タスクからのみ発行可」に制限されます。

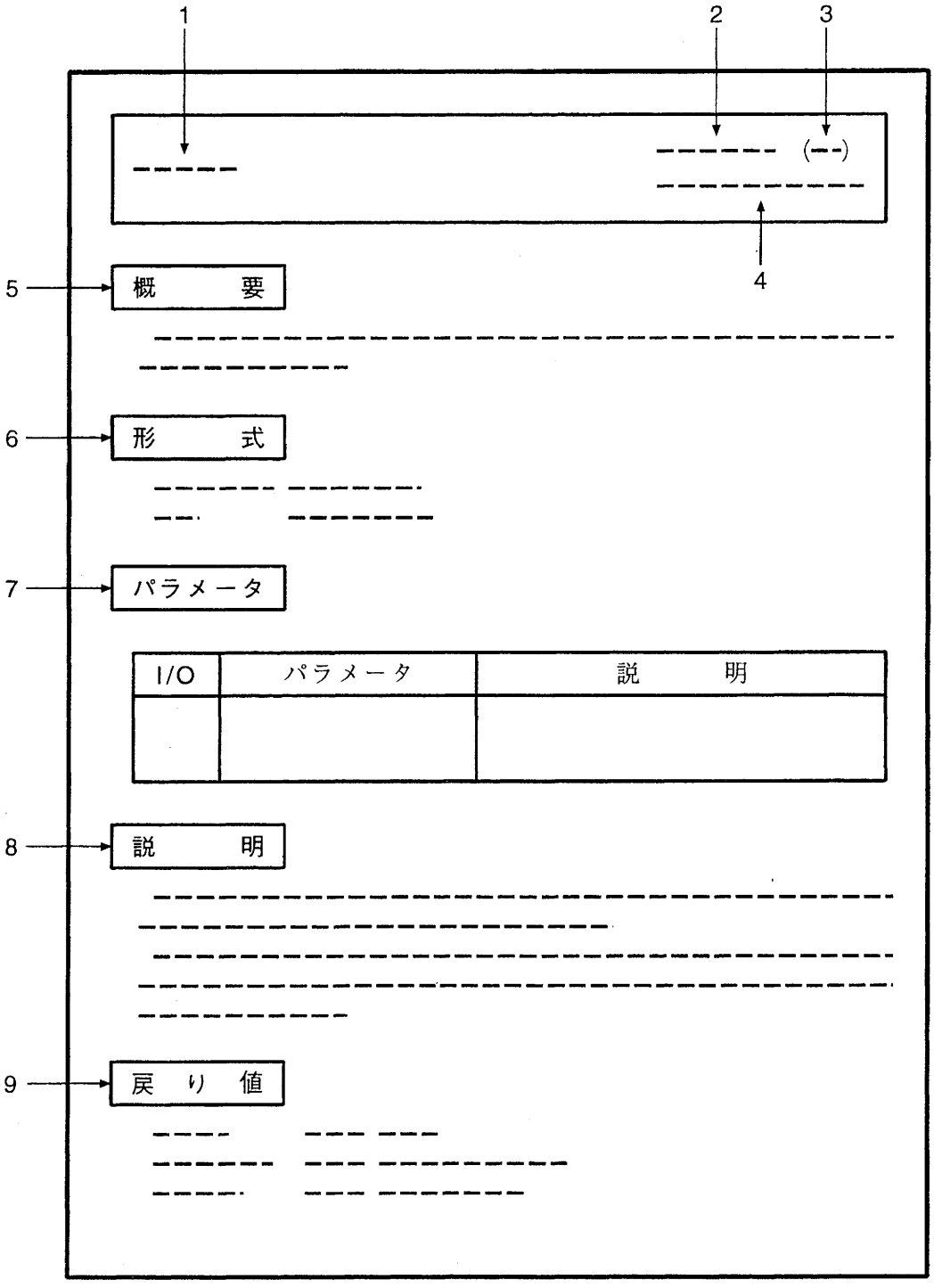
また，拡張システム・コールは，ユーザ定義のシステム・コールとして位置付けられる一方で，タスクに準じた部分を有します。つまり，標準システム・コールと同様に，処理が終了した際にはスケジューラが起動され，最適なタスクの選択処理が行われます。

ただし，拡張システム・コール内に標準システム・コールを含ませた場合には，標準システム・コールの処理が終了した際にもスケジューラが起動されるため，拡張システム・コールの処理途中で，他タスクに制御が移ることがあります。

12.6　システム・コールの解説

次項から RX830 が提供するシステム・コールについて，以下の形式に従って解説します。

図 12-1　システム・コールの記述フォーマット



保守／廃止

1 名称

システム・コールの名称を示しています。

2 名称の由来

システム・コールの名称の由来を示しています。

3 機能コード

RX830 がシステム・コールを識別する際の機能コードを数値で示しています。

4 発行可能な場所

システム・コールを発行する際の状態を示しています。

タスク／非タスク	：タスク，非タスクのどちらからも発行可能
タスク	：タスクからのみ発行可能
非タスク	：ハンドラからのみ発行可能
直接起動割り込みハンドラ	：直接起動割り込みハンドラからのみ発行可能
例外ハンドラ	：例外ハンドラからのみ発行可能

5 概　　要

システム・コールの機能概要を示しています。

6 形　　式

システム・コールを C 言語で発行する際の記述形式を示しています。

7 パラメータ

システム・コールのパラメータを，以下の形式で示しています。

なお，命令オプションのマクロ定義は，option.hファイルで行われています。

I/O	パラメータ	説　　明
A	B	C

A ：パラメータの種類

I ･･･ RX830 への入力パラメータ

O ･･･ RX830 からの出力パラメータ

B ：パラメータの型

C ：パラメータの説明

8 説　　明

システム・コールの機能について示しています。

9 戻　り　値

システム・コールからの戻り値をマクロと値で示しています。

なお，戻り値のマクロ定義は，errcode.hファイルで行われています。

＊付きの戻り値 ･･･ パラメータ・チェック機能あり／なしの両方の RX830 で返す値

＊無しの戻り値 ･･･ パラメータ・チェック機能ありの RX830 でのみ返す値

保守／廃止

12.6.1 タスク管理システム・コール

本項では，表12-5に示すタスク管理システム・コールについて説明しています。

表 12-5 タスク管理システム・コール

番号	システム・コール	機　　能
0	cre_tsk	他タスクを dormant 状態に移行する
1	sta_tsk	他タスクを ready 状態に移行する
2	del_tsk	他タスクを non_existent 状態に移行する
3	def_ext	終了時処理ルーチンを登録／登録解除する
4	ext_tsk	自タスクを dormant 状態に移行する
5	exd_tsk	自タスクを non_existent 状態に移行する
6	abo_tsk	自タスクを dormant 状態に移行する
7	ter_tsk	他タスクを dormant 状態に移行する
8	chg_pri	タスクの優先度を変更する
9	rot_rdq	タスクのレディ・キューを回転する
10	tcb_adr	タスクのアクセス・アドレスを獲得する
11	tsk_sts	タスクの情報を獲得する
12	sus_tsk	他タスクを suspend 状態に移行する
13	rsm_tsk	サスペンド要求を解除する
14	slp_tsk	自タスクを wait 状態（起床待ち状態）に移行する
15	wai_tsk	自タスクを wait 状態（時限待ち状態）に移行する
16	wup_tsk	起床要求を発行する
17	can_wup	すべての起床要求を解除する
18	cyc_wup	周期起床要求，指定時刻起床要求，または，遅延起床要求を発行する
19	can_cyc	周期起床要求を解除する

保守／廃止

Create Task (0)

CRE_TSK

タスク

概　要

他タスクを dormant 状態に移行する。

形　式

```
#include "stdrx.h"
int       cre_tsk(pa_tsk, tskid, pk_crtsk);
```

パラメータ

I/O	パラメータ	説　　明
O	int　　*pa_tsk;	生成したタスクのアクセス・アドレスを格納する領域のアドレス
I	unsigned short tskid;	タスク ID 番号 (0x1 ～ 0xefff)
I	struct t_crtsk *pk_crtsk;	パラメータ・パケットのアドレス

t_crtsk の構造

```
struct  t_crtsk {
        void            (*sta)();   /* タスクの先頭アドレス                */
        unsigned int    stksz;      /* スタック・サイズ (単位 : byte)      */
        char            *p_data;    /* タスクの gp レジスタ値              */
        short           tskpri;     /* 初期優先度                          */
        short           tskopt;     /* スタック領域の確保                  */
                                    /*   T_POOL0 (0x0) :                   */
                                    /*     ニュークリアス・メモリ・プール  */
                                    /*     #0 からスタック領域を確保する   */
                                    /*   T_POOL1 (0x2) :                   */
                                    /*     ニュークリアス・メモリ・プール  */
                                    /*     #1 からスタック領域を確保する   */
};
```

説　明

tskid で指定されたタスクを non_existent 状態から dormant 状態へと遷移させたのち，pa_tsk で指定された領域にタスクのアクセス・アドレスを格納します。

なお，staにはタスクの先頭アドレスを，stkszにはタスクが使用するスタック領域のサイズを，p_dataにはタスクのgpレジスタ値を，tskpriにはタスクの初期優先度を，tskoptにはスタック領域を確保するニュークリアス・メモリ・プールの種別を指定します。

ただし，p_dataに0x0を指定した場合は，本システム・コール発行時のgpレジスタ値がタスクのgpレジスタ値として設定されます。

戻り値

*TE_OK	0x00	正常終了
*TE_MEM	0x01	管理ブロック用の領域，または，スタック用の領域が確保できない
TE_PA	0x06	生成したタスクのアクセス・アドレスを格納する領域のアドレス，または，パラメータ・パケットのアドレスが0x0である
TE_SA	0x08	タスクの先頭アドレスが0x0である
*TE_EXS	0x09	同じタスクID番号のタスクがすでに存在している
TE_IDZR	0x0e	タスクID番号が0x0である
TE_IDOVR	0x0f	タスクID番号が0x1 ～ 0xefffの範囲外である
TE_OPT	0x14	オプション指定が不適当である
TE_TPRI	0x17	タスクの優先度が不正である

保守／廃止

Start Task (1)

STA_TSK

タスク／非タスク

概　要

他タスクを ready 状態に移行する。

形　式

```
#include "stdrx.h"
int       sta_tsk(a_tsk, initcode);
```

パラメータ

I/O	パラメータ	説　明
I	int　a_tsk;	起動するタスクのアクセス・アドレス
I	int　initcode;	起動するタスクに引き渡す初期情報 (起動コード)

説　明

a_tsk で指定されたタスクを dormant 状態から ready 状態へと遷移させます。
なお，initcode には，対象タスクに引き渡す初期情報 (起動コード) を指定します。

注意　対象タスクは，起動コードを関数パラメータと同様に取り扱うことにより，参照可能となります。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	起動するタスクのアクセス・アドレスが不正である
TE_AA	0x05	起動するタスクのアクセス・アドレスが 0x0 である
*TE_NODMT	0x0b	対象タスクが dormant 状態でない

保守／廃止

Delete Task (2)

DEL_TSK

タスク

概　要

他タスクを non_existent 状態に移行する。

形　式

```
#include "stdrx.h"
int       del_tsk(a_tsk);
```

パラメータ

I/O	パラメータ	説　　明
I	int　　a_tsk;	削除するタスクのアクセス・アドレス

説　明

a_tsk で指定されたタスクを dormant 状態から non_existent 状態へと遷移させます。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	削除するタスクのアクセス・アドレスが不正である
TE_AA	0x05	削除するタスクのアクセス・アドレスが 0x0 である
*TE_NODMT	0x0b	対象タスクが dormant 状態でない

保守／廃止

Define Exit Routine (3)

DEF_EXT

タスク

概　要

終了時処理ルーチンを登録／登録解除する。

形　式

```
#include "stdrx.h"
int       def_ext(extrtn, p_extdat);
```

パラメータ

I/O	パラメータ	説　　明
I	void　(*extrtn)();	終了時処理ルーチンの先頭アドレス
I	char　*p_extdat;	終了時処理ルーチンの gp レジスタ値

説　明

タスクが異常終了した際に起動される終了時処理ルーチンの登録／登録解除を行います。

なお，extrtn には終了時処理ルーチンの先頭アドレスを，p_extdat には終了時処理ルーチンの gp レジスタ値を指定します。

ただし，extrtn に 0x0 を指定した場合は，すでに登録されている終了時処理ルーチンの登録解除が行われます。

また，p_extdat に 0x0 を指定した場合は，本システム・コール発行時の gp レジスタ値が終了時処理ルーチンの gp レジスタ値として設定されます。

戻り値

*TE_OK	0x00	正常終了
TE_CTX	0x03	終了時処理ルーチン内で def_ext システム・コールを発行した

保守／廃止

Exit Task (4)

EXT_TSK

タスク

概　要

自タスクを dormant 状態に移行する。

形　式

```
#include "stdrx.h"
void      ext_tsk();
```

パラメータ

なし

説　明

自タスクを run 状態から dormant 状態へと遷移させます。

注意
- 本システム・コールによるタスクの終了形態は，正常終了となります。このため，自タスクに終了時処理ルーチンが登録されていても経由されません。
- 本システム・コールでは，タスクが終了する以前に獲得した資源 (イベント・フラグ，メモリ・ブロックなど) の解放処理を行っていません。このため，本システム・コールを発行する前に，これら資源の解放処理を行う必要があります。
- タスクをアセンブリ言語で記述した場合，ext_tsk システム・コールの発行は，以下のように記述します。

```
jr      _ext_tsk
```

戻り値

なし

保守／廃止

Exit and Delete Task (5)

EXD_TSK

タスク

概　要

自タスクを non_existent 状態に移行する。

形　式

```
#include "stdrx.h"
void      exd_tsk();
```

パラメータ

なし

説　明

自タスクを run 状態から non_existent 状態へと遷移させます。

注意
- 本システム・コールによるタスクの終了形態は，正常終了となります。このため，自タスクに終了時処理ルーチンが登録されていても経由されません。
- 本システム・コールでは，タスクが終了する以前に獲得した資源 (イベント・フラグ，メモリ・ブロックなど) の解放処理を行っていません。このため，本システム・コールを発行する前に，これら資源の解放処理を行う必要があります。
- タスクをアセンブリ言語で記述した場合，exd_tsk システム・コールの発行は，以下のように記述します。

```
jr        _exd_tsk
```

戻り値

なし

保守／廃止

Abort Task (6)

ABO_TSK

タスク

概　要

自タスクを dormant 状態に移行する。

形　式

```
#include "stdrx.h"
void      abo_tsk(abocode);
```

パラメータ

I/O	パラメータ	説　　明
I	int　　abocode;	終了時処理ルーチンに引き渡す情報（異常終了コード）

説　明

自タスクを run 状態から dormant 状態へと遷移させます。
なお，abocode には，終了時処理ルーチンに引き渡す情報（異常終了コード）を指定します。

注意
- 本システム・コールによるタスクの終了形態は，異常終了となります。このため，自タスクに終了時処理ルーチンが登録されていた場合，ただちに終了時処理ルーチンに制御が移ります。
- 終了時処理ルーチンは，異常終了コードを関数パラメータと同様に取り扱うことにより，参照可能となります。
- 本システム・コールでは，タスクが終了する以前に獲得した資源（イベント・フラグ，メモリ・ブロックなど）の解放処理を行っていません。このため，本システム・コールを発行する前に，または，終了時処理ルーチン内で，これら資源の解放処理を行う必要があります。

戻り値

なし

保守／廃止

Terminate Task (7)

TER_TSK

タスク

概　要

他タスクを dormant 状態に移行する。

形　式

```
#include "stdrx.h"
int       ter_tsk(a_tsk, abocode);
```

パラメータ

I/O	パラメータ	説　明
I	int　a_tsk;	異常終了させるタスクのアクセス・アドレス
I	int　abocode;	終了時処理ルーチンに引き渡す情報（異常終了コード）

説　明

a_tsk で指定されたタスクを dormant 状態へと遷移させます。
なお，abocode には，終了時処理ルーチンに引き渡す情報（異常終了コード）を指定します。

注意
- 本システム・コールによるタスクの終了形態は，異常終了となります。このため，対象タスクに終了時処理ルーチンが登録されていた場合，ただちに終了時処理ルーチンに制御が移ります。
- 終了時処理ルーチンは，異常終了コードを関数パラメータと同様に取り扱うことにより，参照可能となります。
- 本システム・コールでは，タスクが終了する以前に獲得した資源（イベント・フラグ，メモリ・ブロックなど）の解放処理を行っていません。このため，本システム・コールを発行する前に，または，終了時処理ルーチン内で，これら資源の解放処理を行う必要があります。

戻り値

*TE_OK	0x00	正常終了
*TE_CTX	0x03	終了時処理ルーチン処理中のタスクに対して ter_tsk システム・コールを発行した
*TE_OBJ	0x04	異常終了させるタスクのアクセス・アドレスが不正である
TE_AA	0x05	異常終了させるタスクのアクセス・アドレスが 0x0 である
*TE_DMT	0x0d	対象タスクが dormant 状態である

TE_SELF　0x12　自タスクを指定した
TE_STSK　0x81　システム・タスクを指定した

保守／廃止

Change Task Priority (8)

CHG_PRI

タスク／非タスク

概　要

タスクの優先度を変更する。

形　式

```
#include "stdrx.h"
int       chg_pri(a_tsk, tskpri);
```

パラメータ

I/O	パラメータ	説　　明
I	int　　a_tsk;	優先度を変更するタスクのアクセス・アドレス，または，TSK_SELF
I	short　tskpri;	変更後の優先度

説　明

a_tsk で指定されたタスクの優先度を tskpri で指定された優先度に変更します。

ただし，本システム・コールを発行した際，a_tsk で指定されたタスクが run 状態，または，ready 状態であった場合には，優先度の変更を行ったのち，変更後の優先度に応じたレディ・キューの最後尾につなげかえます。

なお，本システム・コールの発行により変更した優先度は，chg_pri システム・コールが再発行されるまで，または，対象タスクが dormant 状態へと遷移するまで有効です。

注意
- 本システム・コールを発行した際，対象タスクが何らかの待ち行列に優先度順でキューイングされていた場合，待ち行列のキューイング順序が変わることがあります。
- a_tsk に TSK_SELF を指定した場合，自タスクが指定されたものとして扱われます。このため，本システム・コールを非タスクから発行する場合は，a_tsk に TSK_SELF を指定することはできません。

戻り値

*TE_OK	0x00	正常終了
*TE_CTX	0x03	終了時処理ルーチン処理中のタスクに対して chg_pri システム・コールを発行した
*TE_OBJ	0x04	優先度を変更するタスクのアクセス・アドレスが不正である
*TE_DMT	0x0d	対象タスクが dormant 状態である
TE_TPRI	0x17	変更後の優先度が不正である

TE_STSK　0x81　　システム・タスクを指定した

保守／廃止

Rotate Ready Queue (9)

ROT_RDQ

タスク／非タスク

概　要

タスクのレディ・キューを回転する。

形　式

```
#include "stdrx.h"
int       rot_rdq(tskpri);
```

パラメータ

I/O	パラメータ	説　　明
I	short　tskpri;	回転させるレディ・キューの優先度

説　明

tskpri で指定された優先度に応じたレディ・キューの先頭にキューイングされているタスクを最後尾にキューイングし，タスクの実行順序を切り替えます。

なお，**tskpri** に自タスクと同じ優先度を指定した場合は，自タスクがレディ・キューの最後尾にキューイングされます。これにより，自タスクは **run** 状態から **ready** 状態へと遷移します。

注意　本システム・コールを発行した際，指定した優先度のレディ・キューにタスクが存在しない場合には，キュー操作などは行わず，エラーとしても扱いません。

戻り値

*TE_OK	0x00	正常終了
TE_TPRI	0x17	回転させるレディ・キューの優先度が不正である

保守／廃止

Get TCB Access Address (10)

TCB_ADR

タスク／非タスク

概　要

タスクのアクセス・アドレスを獲得する。

形　式

```
#include "stdrx.h"
int       tcb_adr(pa_tsk, tskid);
```

パラメータ

I/O	パラメータ	説　　明
O	int　　*pa_tsk;	指定したタスクのアクセス・アドレスを格納する領域のアドレス
I	unsigned short tskid;	タスク ID 番号 (0x0 ～ 0xefff)，または，TSK_SELF

説　明

tskid で指定されたタスクのアクセス・アドレスを pa_tsk で指定された領域に格納します。

注意　pa_tsk に TSK_SELF を指定した場合，自タスクが指定されたものとして扱われます。このため，本システム・コールを非タスクから発行する場合は，tskid に TSK_SELF を指定することはできません。

戻り値

*TE_OK	0x00	正常終了
TE_PA	0x06	タスクのアクセス・アドレスを格納する領域のアドレスが 0x0 である
*TE_NOEXS	0x0a	指定したタスク ID 番号のタスクは存在しない
TE_IDOVR	0x0f	タスク ID 番号が 0x0 ～ 0xefff の範囲外である

保守／廃止

Get Task Status (11)

TSK_STS

タスク／非タスク

概　要

タスクの情報を獲得する。

形　式

```
#include "stdrx.h"
int       tsk_sts(pk_tskst, a_tsk);
```

パラメータ

I/O	パラメータ	説　　明
O	struct t_tskst *pk_tskst;	獲得したタスクの情報を格納するパケットのアドレス
I	int a_tsk;	情報を獲得するタスクのアクセス・アドレス，または，TSK_SELF

t_tskst の構造

```
struct  t_tskst {
        unsigned short  tskid;   /* タスク ID 番号                          */
        short           tskpri;  /* タスクの優先度                          */
        short           tsksts;  /* タスクの状態，および，実行プロシージャ */
};
```

説　明

a_tsk で指定されたタスクの情報（タスク ID 番号，タスクの優先度，タスクの状態，実行プロシージャ）を pk_tskst で指定された領域に格納します。

以下に，タスクの情報の詳細を示します。

パラメータ	説　　明
tskid	タスク ID 番号
tskpri	タスクの優先度
tsksts	タスクの状態（ビット 0 ～ 8） 000000000 ： run 状態 000000001 ： ready 状態 000000010 ： wait 状態（起床待ち状態） 000000100 ： wait 状態（イベント・フラグ待ち状態）

パラメータ	説　　明
	000001000 ： wait 状態（資源待ち状態） 000010000 ： wait 状態（メッセージ待ち状態） 000100000 ： wait 状態（メモリ・ブロック待ち状態） 001000000 ： wait 状態（時限待ち状態） 010000000 ： suspend 状態 100000000 ： dormant 状態 実行プロシージャ（ビット 13 ～ 15） 000 ： タスク 001 ： 終了時処理ルーチン 010 ： CPU 例外ハンドラ 100 ： システム・コール例外ハンドラ

注意　a_tsk に TSK_SELF を指定した場合，自タスクが指定されたものとして扱われます。このため，本システム・コールを非タスクから発行する場合は，tskid に TSK_SELF を指定することはできません。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	情報を獲得するタスクのアクセス・アドレスが不正である
TE_PA	0x06	獲得したタスクの情報を格納するパケットのアドレスが 0x0 である

保守／廃止

Suspend Task (12)

SUS_TSK

タスク

概　要

他タスクを suspend 状態に移行する。

形　式

```
#include "stdrx.h"
int       sus_tsk(a_tsk);
```

パラメータ

I/O	パラメータ	説　　明
I	int　　a_tsk;	サスペンドさせるタスクのアクセス・アドレス

説　明

a_tsk で指定されたタスクを suspend 状態へと遷移させます。

ただし，本システム・コールを発行した際，対象タスクのサスペンド要求カウンタが 0x0 以外の値であった場合には，対象タスクの状態操作は行わず，サスペンド要求カウンタの加算処理 (サスペンド要求カウンタに 0x1 を加算) を行います。

注意　RX830 が管理するサスペンド要求カウンタは，8 ビット幅で構成されています。このため，本システム・コールでは，サスペンド要求数が 0xff を越えるような場合には，サスペンド要求カウンタへの加算処理は行わず，戻り値として TE_QOVR(0x11) を返しています。

戻り値

*TE_OK	0x00	正常終了
*TE_CTX	0x03	終了時処理ルーチン処理中のタスクに対して sus_tsk システム・コールを発行した
*TE_OBJ	0x04	サスペンドさせるタスクのアクセス・アドレスが不正である
TE_AA	0x05	サスペンドさせるタスクのアクセス・アドレスが 0x0 である
*TE_DMT	0x0d	対象タスクが dormant 状態である
*TE_QOVR	0x11	サスペンド要求が 0xff 回を越えた
TE_SELF	0x12	自タスクを指定した
TE_STSK	0x81	システム・タスクを指定した

保守／廃止

Resume Task (13)

RSM_TSK

タスク

概　要

サスペンド要求を解除する。

形　式

```
#include "stdrx.h"
int       rsm_tsk(option, a_tsk);
```

パラメータ

I/O	パラメータ	説　　明
I	short　option;	命令オプション サスペンド要求解除の指定方法 T_NOOPT　(0x0) ： 1つだけ解除 T_FRCRSM (0x1) ： すべて解除
I	int　　a_tsk;	サスペンド要求を解除するタスクのアクセス・アドレス

説　明

a_tsk で指定されたタスクに発行されているサスペンド要求を解除します。
なお， option には，サスペンド要求を解除する際の方法を指定します。
以下に， option の指定形式を示します。

- option = T_NOOPT

  対象タスクに発行されているサスペンド要求を 1 つだけ解除します。

- option = T_FRCRSM

  対象タスクに発行されているサスペンド要求をすべて解除します。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	サスペンド要求を解除するタスクのアクセス・アドレスが不正である
TE_AA	0x05	サスペンド要求を解除するタスクのアクセス・アドレスが 0x0 である
*TE_NOSUS	0x0c	対象タスクが suspend 状態でない
TE_OPT	0x14	オプション指定が不適当である

保守／廃止

Sleep Task (14)

SLP_TSK

タスク

概　要

自タスクを wait 状態 (起床待ち状態) に移行する。

形　式

```
#include "stdrx.h"
int       slp_tsk();
```

パラメータ

なし

説　明

自タスクを run 状態から wait 状態 (起床待ち状態) へと遷移させます。

ただし，本システム・コールを発行した際，自タスクの起床要求カウンタが 0x0 以外の値であった場合には，自タスクの状態操作は行わず，起床要求カウンタの減算処理 (起床要求カウンタから 0x1 を減算) を行います。

なお，起床待ち状態の解除は，起床要求を発行 (wup_tsk， cyc_wup， iret_wup システム・コールの発行) した際に行われ，起床待ち状態から ready 状態へと遷移します。

戻り値

*TE_OK	0x00	正常終了

保守／廃止

Wait for Wakeup Task (15)

WAI_TSK

タスク

概　要

自タスクを wait 状態 (時限待ち状態) に移行する。

形　式

```
#include "stdrx.h"
int       wai_tsk(p_tmout);
```

パラメータ

I/O	パラメータ	説　　明
I	unsigned long　*p_tmout;	タイムアウト値が格納されている領域のアドレス (タイムアウト値の単位： msec)

説　明

自タスクを run 状態から wait 状態 (時限待ち状態) へと遷移させます。

ただし，本システム・コールを発行した際，自タスクの起床要求カウンタが 0x0 以外の値であった場合には，自タスクの状態操作は行わず，起床要求カウンタの減算処理 (起床要求カウンタから 0x1 を減算) を行います。

なお，時限待ち状態の解除は，起床要求を発行 (wup_tsk， cyc_wup， iret_wup システム・コールの発行) した際，または， p_tmout で指定された時間が経過した際に行われ，時限待ち状態から ready 状態へと遷移します。

戻り値

*TE_OK	0x00	正常終了
TE_PA	0x06	タイムアウト値を格納する領域のアドレスが 0x0 である
*TE_TMOUT	0x10	指定された時間が経過した

保守／廃止

Wakeup Task (16)

WUP_TSK

タスク／非タスク

概　要

起床要求を発行する。

形　式

```
#include "stdrx.h"
int       wup_tsk(a_tsk);
```

パラメータ

I/O	パラメータ	説　　明
I	int　　a_tsk;	起床させるタスクのアクセス・アドレス

説　明

a_tsk で指定されたタスクに，起床要求を発行（起床カウンタに 0x1 を加算）します。

ただし，本システム・コールを発行した際，a_tsk で指定されたタスクが wait 状態（時限待ち状態，または，起床待ち状態）であった場合には，起床要求の発行（起床要求カウンタへの加算処理）は行わず，対象タスクを時限待ち状態，または，起床待ち状態から ready 状態へと遷移させます。

注意　RX830 が管理する起床要求カウンタは，8 ビット幅で構成されています。このため，本システム・コールでは，起床要求数が 0xff を越えるような場合には，起床要求カウンタへの加算処理は行わず，戻り値として TE_QOVR(0x11) を返しています。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	起床させるタスクのアクセス・アドレスが不正である
TE_AA	0x05	起床させるタスクのアクセス・アドレスが 0x0 である
*TE_DMT	0x0d	対象タスクが dormant 状態である
*TE_QOVR	0x11	起床要求が 0xff 回を越えた
TE_SELF	0x12	自タスクを指定した

保守／廃止

Cancel Wakeup Task (17)

CAN_WUP

タスク／非タスク

概　要

すべての起床要求を解除する。

形　式

```
#include "stdrx.h"
int       can_wup(p_wupcnt, a_tsk);
```

パラメータ

I/O	パラメータ	説　　明
O	short　*p_wupcnt;	解除した起床要求数を格納する領域のアドレス
I	int　　a_tsk;	起床要求を解除するタスクのアクセス・アドレス，または，TSK_SELF

説　明

a_tsk で指定されたタスクに発行されているすべての起床要求を解除 (起床要求カウンタに 0x0 を設定) したのち，p_wupcnt で指定された領域に解除した起床要求数を格納します。

注意　a_tsk に TSK_SELF を指定した場合，自タスクが指定されたものとして扱われます。このため，本システム・コールを非タスクから発行する場合は，a_tsk に TSK_SELF を指定することはできません。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	起床要求を解除するタスクのアクセス・アドレスが不正である
TE_PA	0x06	解除した起床要求数を格納する領域のアドレスが 0x0 である
*TE_DMT	0x0d	対象タスクが dormant 状態である

保守／廃止

Cyclic Wakeup Task (18)

CYC_WUP

タスク／非タスク

概　要

周期起床要求，指定時刻起床要求，または，遅延起床要求を発行する。

形　式

```
#include "stdrx.h"
int       cyc_wup(option, a_tsk, pk_cywup);
```

パラメータ

I/O	パラメータ	説　　明
I	short　option;	命令オプション 初期起床時刻の指定方法 T_ABS　(0x0)：絶対時間 T_REL　(0x1)：相対時間
I	int　　a_tsk;	起床させるタスクのアクセス・アドレス，または， TSK_SELF
I	struct　t_cywup *pk_cywup;	パラメータ・パケットのアドレス

t_cywup の構造

```
struct  t_cywup {
        struct  t_time {
                unsigned long   ltime;  /* 時刻（下位 32 ビット）          */
                unsigned short  utime;  /* 時刻（上位 16 ビット）          */
        } init;                         /* 初期起床時刻（単位：msec）      */
        unsigned long   interval;       /* 周期起床時間間隔（単位：msec）  */
        unsigned short  count;          /* 起床回数                        */
};
```

説　明

a_tsk で指定されたタスクに，init で指定された時間経過後から interval で指定された時間間隔で，count で指定された回数の起床要求を発行します。

なお，RX830 が管理するシステム・クロックは，48 ビット幅で構成されています。そこで，初期起床時刻 init では，下位 32 ビットを ltime に，上位 16 ビットを utime に設定します。

また，option には，init で指定された初期起床時刻がシステム・クロックを基準とした絶対時間であるのか，本システム・コール発行時からの相対時間であるのかを指定します。

以下に，option の指定形式を示します。

- option = T_ABS

  初期起床時刻 init は，システム・クロックを基準とした絶対時間となります。

- option = T_REL

  初期起床時刻 init は，本システム・コール発行時からの相対時間となります。

周期起床要求　：init に初期起床時刻，interval に起床間隔，count に起床回数を指定することにより，周期起床が実現されます。
なお，count に 0x0 を指定した場合，対象タスクが non_existent 状態に遷移するまで，起床要求を発行し続けます。

指定時刻起床要求：init に起床時刻，count に 0x1，option に T_ABS を指定することにより，指定時刻起床が実現されます。
このとき，interval は無視されます。

遅延起床要求　：init に起床時刻，count に 0x1，option に T_REL を指定することにより，遅延起床が実現されます。
このとき，interval は無視されます。

注意　a_tsk に TSK_SELF を指定した場合，自タスクが指定されたものとして扱われます。このため，本システム・コールを非タスクから発行する場合は，a_tsk に TSK_SELF を指定することはできません。

戻り値

*TE_OK	0x00	正常終了
*TE_CTX	0x03	終了時処理ルーチン処理中のタスクに対して cyc_wup システム・コールを発行した
*TE_OBJ	0x04	起床させるタスクのアクセス・アドレスが不正である
TE_PA	0x06	パラメータ・パケットのアドレスが 0x0 である
TE_OPT	0x14	オプション指定が不適当である
*TE_NOTMR	0x16	タイマ・ブロックの領域が確保できない
TE_PAR	0x1f	周期起床時間間隔が 0x0 である

保守／廃止

Cancel Cyclic Wakeup Task (19)

CAN_CYC

タスク／非タスク

概　要

周期起床要求を解除する。

形　式

```
#include "stdrx.h"
int       can_cyc(a_tsk);
```

パラメータ

I/O	パラメータ	説　　明
I	int　　a_tsk;	周期起床要求を解除するタスクのアクセス・アドレス，または，TSK_SELF

説　明

a_tsk で指定されたタスクに発行されている周期起床要求を解除します。

注意　a_tsk に TSK_SELF を指定した場合，自タスクが指定されたものとして扱われます。このため，本システム・コールを非タスクから発行する場合は，a_tsk に TSK_SELF を指定することはできません。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	周期起床要求を解除するタスクのアクセス・アドレスが不正である
*TE_NOCYC	0x19	対象タスクに周期起床要求は発行されていない

12.6.2 同期通信管理システム・コール

本項では，表12-6に示す同期通信管理システム・コールについて説明しています。

表 12-6 同期通信管理システム・コール

番号	システム・コール	機能
20	cre_flg	イベント・フラグを生成する
21	del_flg	イベント・フラグを削除する
22	set_flg	ビット・パターンを設定する
23	wai_flg	ビット・パターンをチェックする
24	flg_adr	イベント・フラグのアクセス・アドレスを獲得する
25	cre_sem	セマフォを生成する
26	del_sem	セマフォを削除する
27	sig_sem	資源を返却する
28	wai_sem	資源を獲得する
29	sem_adr	セマフォのアクセス・アドレスを獲得する
30	cre_mbx	メールボックスを生成する
31	del_mbx	メールボックスを削除する
32	snd_msg	メッセージを送信する
33	rcv_msg	メッセージを受信する
34	mbx_adr	メールボックスのアクセス・アドレスを獲得する

保守／廃止

Create Event Flag (20)

CRE_FLG

タスク

概　要

イベント・フラグを生成する。

形　式

```
#include "stdrx.h"
int       cre_flg(pa_flg, flgid);
```

パラメータ

I/O	パラメータ	説　　明
O	int　　*pa_flg;	生成したイベント・フラグのアクセス・アドレスを格納する領域のアドレス
I	unsigned short flgid;	イベント・フラグ ID 番号 (0x1 ～ 0xefff)

説　明

flgid で指定された ID 番号を持つイベント・フラグを生成したのち，pa_flg で指定された領域にイベント・フラグのアクセス・アドレスを格納します。

なお，生成されたイベント・フラグの初期ビット・パターンは，0x0 となります。

戻り値

*TE_OK	0x00	正常終了
*TE_MEM	0x01	管理ブロック用の領域が確保できない
TE_PA	0x06	生成したイベント・フラグのアクセス・アドレスを格納する領域のアドレスが 0x0 である
*TE_EXS	0x09	同じイベント・フラグ ID 番号のイベント・フラグがすでに存在している
TE_IDZR	0x0e	イベント・フラグ ID 番号が 0x0 である
TE_IDOVR	0x0f	イベント・フラグ ID 番号が 0x1 ～ 0xefff の範囲外である

保守／廃止

Delete Event Flag (21)

DEL_FLG

タスク

概　要

イベント・フラグを削除する。

形　式

```
#include "stdrx.h"
int       del_flg(a_flg);
```

パラメータ

I/O	パラメータ	説　　明
I	int　　a_flg;	削除するイベント・フラグのアクセス・アドレス

説　明

a_flg で指定されたイベント・フラグを削除します。

注意　本システム・コールを発行した際，対象イベント・フラグの待ち行列に条件の成立を待っているタスク (wai_flg システム・コールを発行したタスク) が存在した場合，待っているタスクを wait 状態 (イベント・フラグ待ち状態) から ready 状態へと遷移させたのち，wai_flg システム・コールの戻り値として TE_DLT(0x13) を返しています。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	削除するイベント・フラグのアクセス・アドレスが不正である
TE_AA	0x05	削除するイベント・フラグのアクセス・アドレスが 0x0 である

保守／廃止

Set Event Flag (22)

SET_FLG

タスク／非タスク

概　要

ビット・パターンを設定する。

形　式

```
#include "stdrx.h"
int       set_flg(option, p_prptn, a_flg, setptn);
```

パラメータ

I/O	パラメータ	説　明
I	short option;	命令オプション 設定条件の指定方法 T_OR (0x0) : 論理和 T_AND (0x1) : 論理積 T_REP (0x2) : 置換 T_EXOR (0x3) : 排他的論理和
O	int *p_prptn;	設定前のビット・パターンを格納する領域のアドレス
I	int a_flg;	ビット・パターンを設定するイベント・フラグのアクセス・アドレス
I	int setptn;	設定するビット・パターン

説　明

a_flg で指定されたイベント・フラグに option で指定された設定条件に従って setptn で指定されたビット・パターンを設定します。

なお，リターン・パラメータとして，p_prptn で指定された領域に本システム・コール発行前の対象イベント・フラグのビット・パターンが格納されます。

また，本システム・コールを発行した結果，対象イベント・フラグの待ち行列にキューイングされているタスク (イベント・フラグ待ち状態) の待ち条件を満足した場合には，該当タスクをイベント・フラグ待ち状態から ready 状態へと遷移させます。

以下に，option の指定形式を示します。

- option = T_OR

  対象イベント・フラグのビット・パターンと setptn で指定されたビット・パターンの論理和をとり，その結果を対象イベント・フラグに設定します。

- option = T_AND

  対象イベント・フラグのビット・パターンと setptn で指定されたビット・パターンの論理積をとり，その結果を対象イベント・フラグに設定します。

- option = T_REP

  対象イベント・フラグのビット・パターンと setptn で指定されたビット・パターンを置換します。

- option = T_EXOR

  対象イベント・フラグのビット・パターンと setptn で指定されたビット・パターンの排他的論理和をとり，その結果を対象イベント・フラグに設定します。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	ビット・パターンを設定するイベント・フラグのアクセス・アドレスが不正である
TE_AA	0x05	ビット・パターンを設定するイベント・フラグのアクセス・アドレスが 0x0 である
TE_PA	0x06	設定前のビット・パターンを格納する領域のアドレスが 0x0 である
TE_OPT	0x14	オプション指定が不適当である

保守／廃止

Wait Event Flag (23)

WAI_FLG

タスク

概　要

ビット・パターンをチェックする。

形　式

```
#include "stdrx.h"
int       wai_flg(option, p_prptn, a_flg, mskptn, p_tmout);
```

パラメータ

I/O	パラメータ	説　　明
I	short option;	命令オプション タイムアウトの指定方法 T_NOOPT (0x0) : タイムアウト指定なし T_TMOUT (0x1) : タイムアウト指定あり 待ち条件の指定 T_ANDW (0x0) : AND 待ち T_ORW (0x2) : OR 待ち 条件成立後の指定 T_NOOPT (0x0) : リセット指定なし T_RESET (0x4) : リセット指定あり
O	int *p_prptn;	条件成立時のイベント・フラグのビット・パターンを格納する領域のアドレス
I	int a_flg;	条件成立を待つイベント・フラグのアクセス・アドレス
I	int mskptn;	要求するビット・パターン
I	unsigned long *p_tmout;	タイムアウト値が格納されている領域のアドレス (タイムアウト値の単位：msec)

説　明

a_flg で指定されたイベント・フラグに mskptn で指定されたビット・パターンと option で指定された待ち条件を満足するビット・パターンが設定されているか否かをチェックします。

なお，リターン・パラメータとして，p_prptn で指定された領域に条件成立時の対象イベント・フラグのビット・パターンが格納されます。

ただし，本システム・コールを発行した際，対象イベント・フラグのビット・パターンが要求する待ち条件を満足していなかった場合には，自タスクを対象イベント・フラグの待ち行列にキューイングしたのち，run 状態から wait 状態 (イベント・フラグ待ち状態) へと遷移させます。

以下に，option の指定形式を示します。

- option = (T_NOOPT | T_ANDW | T_NOOPT)

  本システム・コールを発行したタスクは，mskptn で 1 に設定されている全ビットが対象イベント・フラグに設定されるまで wait 状態 (イベント・フラグ待ち状態) へと遷移します。

- option = (T_NOOPT | T_ANDW | T_RESET)

  本システム・コールを発行したタスクは，mskptn で 1 に設定されている全ビットが対象イベント・フラグに設定されるまで wait 状態 (イベント・フラグ待ち状態) へと遷移します。

  なお，待ち条件が成立した際には，対象イベント・フラグのビット・パターンを 0 クリアします。

- option = (T_NOOPT | T_ORW | T_NOOPT)

  本システム・コールを発行したタスクは，mskptn で 1 に設定されているいずれかのビットが対象イベント・フラグに設定されるまで wait 状態 (イベント・フラグ待ち状態) へと遷移します。

- option = (T_NOOPT | T_ORW | T_RESET)

  本システム・コールを発行したタスクは，mskptn で 1 に設定されているいずれかのビットが対象イベント・フラグに設定されるまで wait 状態 (イベント・フラグ待ち状態) へと遷移します。

  なお，待ち条件が成立した際には，対象イベント・フラグのビット・パターンを 0 クリアします。

- option = (T_TMOUT | T_ANDW | T_NOOPT)

  本システム・コールを発行したタスクは，mskptn で 1 に設定されている全ビットが対象イベント・フラグに設定されるまで wait 状態 (イベント・フラグ待ち状態) へと遷移します。

  ただし，本システム・コールの発行から p_tmout で指定された時間が経過した際には，イベント・フラグ待ち状態は強制的に解除され，wai_flg システム・コールの戻り値として TE_TMOUT(0x10) が返されます。

- option = (T_TMOUT | T_ANDW | T_RESET)

  本システム・コールを発行したタスクは，mskptn で 1 に設定されている全ビットが対象イベント・フラグに設定されるまで wait 状態 (イベント・フラグ待ち状態) へと遷移します。

ただし，本システム・コールの発行からp_tmoutで指定された時間が経過した際には，イベント・フラグ待ち状態は強制的に解除され，wai_flgシステム・コールの戻り値としてTE_TMOUT(0x10)が返されます。

なお，待ち条件が成立した際には，対象イベント・フラグのビット・パターンを0クリアします。

- option = (T_TMOUT | T_ORW | T_NOOPT)

  本システム・コールを発行したタスクは，mskptnで1に設定されているいずれかのビットが対象イベント・フラグに設定されるまでwait状態(イベント・フラグ待ち状態)へと遷移します。

  ただし，本システム・コールの発行からp_tmoutで指定された時間が経過した際には，イベント・フラグ待ち状態は強制的に解除され，wai_flgシステム・コールの戻り値としてTE_TMOUT(0x10)が返されます。

- option = (T_TMOUT | T_ORW | T_RESET)

  本システム・コールを発行したタスクは，mskptnで1に設定されているいずれかのビットが対象イベント・フラグに設定されるまでwait状態(イベント・フラグ待ち状態)へと遷移します。

  ただし，本システム・コールの発行からp_tmoutで指定された時間が経過した際には，イベント・フラグ待ち状態は強制的に解除され，wai_flgシステム・コールの戻り値としてTE_TMOUT(0x10)が返されます。

  なお，待ち条件が成立した際には，対象イベント・フラグのビット・パターンを0クリアします。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	条件成立を待つイベント・フラグのアクセス・アドレスが不正である
TE_AA	0x05	条件成立を待つイベント・フラグのアクセス・アドレスが0x0である
TE_PA	0x06	条件成立時のイベント・フラグのビット・パターンを格納する領域のアドレス，または，タイムアウト値を格納する領域のアドレスが0x0である
*TE_TMOUT	0x10	条件が成立しないまま，指定された時間が経過した
*TE_DLT	0x13	条件が成立するのを待っている間にイベント・フラグが削除された
TE_OPT	0x14	オプション指定が不適当である
*TE_WEVF	0x15	すでに他タスクが使用している
TE_PAR	0x1f	要求するビット・パターンが0x0である

保守／廃止

Get Event Flag Access Address (24)

FLG_ADR

タスク／非タスク

概　要

イベント・フラグのアクセス・アドレスを獲得する。

形　式

```
#include "stdrx.h"
int       flg_adr(pa_flg, flgid);
```

パラメータ

I/O	パラメータ	説　　明
O	int　　*pa_flg;	指定したイベント・フラグのアクセス・アドレスを格納する領域のアドレス
I	unsigned short flgid;	イベント・フラグ ID 番号 (0x1 ～ 0xefff)

説　明

flgid で指定されたイベント・フラグのアクセス・アドレスを pa_flg で指定された領域に格納します。

戻り値

*TE_OK	0x00	正常終了
TE_PA	0x06	イベント・フラグのアクセス・アドレスを格納する領域のアドレスが 0x0 である
*TE_NOEXS	0x0a	指定したイベント・フラグ ID 番号のイベント・フラグは存在しない
TE_IDZR	0x0e	イベント・フラグ ID 番号が 0x0 である
TE_IDOVR	0x0f	イベント・フラグ ID 番号が 0x1 ～ 0xefff の範囲外である

保守／廃止

Create Semaphore (25)

CRE_SEM

タスク

概　要

セマフォを生成する。

形　式

```
#include "stdrx.h"
int       cre_sem(option, pa_sem, semid, initcnt);
```

パラメータ

I/O	パラメータ	説　　明
I	short　option;	命令オプション タスクの待ち方の指定方法 T_TPRI　(0x0) ： タスクの優先度順 T_TFIFO　(0x1) ： FIFO 順
O	int　　*pa_sem;	生成したセマフォのアクセス・アドレスを格納する領域のアドレス
I	unsigned short　semid;	セマフォ ID 番号 (0x1 ～ 0xefff)
I	unsigned int　　initcnt;	セマフォの初期カウント値 (0x0 ～ 0xffffffff)

説　明

semid で指定された ID 番号を持つセマフォを生成したのち，pa_sem で指定された領域にセマフォのアクセス・アドレスを格納します。

なお，option には本システム・コールの発行により生成したセマフォから資源の獲得を行うタスクが待ち行列にキューイングされる際のキューイング方法を，initcnt にはセマフォの初期カウント値を指定します。

以下に，option の指定形式を示します。

- option = T_TPRI

  wai_sem システム・コールを発行した際，ただちに要求する数の資源を獲得することができなかった場合には，タスクの優先度順で待ち行列にキューイングされます。

- option = T_TFIFO

  wai_sem システム・コールを発行した際，ただちに要求する数の資源を獲得することができなかった場合には，FIFO 順 (資源の獲得要求を行った順) で待ち行列にキューイングされます。

保守／廃止

戻り値

*TE_OK	0x00	正常終了
*TE_MEM	0x01	管理ブロック用の領域が確保できない
TE_PA	0x06	生成したセマフォのアクセス・アドレスを格納する領域のアドレスが 0x0 である
*TE_EXS	0x09	同じセマフォ ID 番号のセマフォがすでに存在している
TE_IDZR	0x0e	セマフォ ID 番号が 0x0 である
TE_IDOVR	0x0f	セマフォ ID 番号が 0x1 ～ 0xefff の範囲外である
TE_OPT	0x14	オプション指定が不適当である

保守／廃止

Delete Semaphore (26)

DEL_SEM

タスク

概　要

セマフォを削除する。

形　式

```
#include "stdrx.h"
int       del_sem(a_sem);
```

パラメータ

I/O	パラメータ	説　　明
I	int　　a_sem;	削除するセマフォのアクセス・アドレス

説　明

a_sem で指定されたセマフォを削除します。

注意　本システム・コールを発行した際，対象セマフォの待ち行列に資源の獲得を待っているタスク (wai_sem システム・コールを発行したタスク) が存在した場合，待っているタスクを wait 状態 (資源待ち状態) から ready 状態へと遷移させたのち，wai_sem システム・コールの戻り値として TE_DLT(0x13) を返しています。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	削除するセマフォのアクセス・アドレスが不正である
TE_AA	0x05	削除するセマフォのアクセス・アドレスが 0x0 である

保守／廃止

Signal Semaphore (27)

SIG_SEM

タスク／非タスク

概　要

資源を返却する。

形　式

```
#include "stdrx.h"
int       sig_sem(a_sem, cnt);
```

パラメータ

I/O	パラメータ	説　　明
I	int　　a_sem;	資源を返却するセマフォのアクセス・アドレス
I	unsigned int　　cnt;	返却する資源数

説　明

a_sem で指定されたセマフォに cnt で指定された数の資源を返却 (セマフォ・カウンタに cnt を加算) します。

なお，対象セマフォに資源を返却した際，対象セマフォの待ち行列にキューイングされている先頭タスクの待ち条件を満足した場合には，先頭タスクが wait 状態 (資源待ち状態) から ready 状態へと遷移します。このとき， ready 状態に遷移したタスクの要求していた資源数が，セマフォ・カウンタから減算されます。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	資源を返却するセマフォのアクセス・アドレスが不正である
TE_AA	0x05	資源を返却するセマフォのアクセス・アドレスが 0x0 である
TE_PAR	0x1f	返却する資源数が 0x0 である

保守／廃止

Wait on Semaphore (28)

WAI_SEM

タスク

概　要

資源を獲得する。

形　式

```
#include "stdrx.h"
int       wai_sem(option, a_sem, cnt, p_tmout);
```

パラメータ

I/O	パラメータ	説　　明
I	short　option;	命令オプション タイムアウトの指定方法 T_NOOPT (0x0) ： タイムアウト指定なし T_TMOUT (0x1) ： タイムアウト指定あり
I	int　　a_sem;	資源を要求するセマフォのアクセス・アドレス
I	unsigned int　cnt;	要求する資源数
I	unsigned long　*p_tmout;	タイムアウト値が格納されている領域のアドレス (タイムアウト値の単位： msec)

説　明

a_sem で指定されたセマフォから cnt で指定された数の資源を獲得 (セマフォ・カウンタから cnt を減算) します。

ただし，本システム・コールを発行した際，対象セマフォから要求した数の資源を獲得することができなかった場合には，自タスクを対象セマフォの待ち行列にキューイングしたのち，run 状態から wait 状態 (資源待ち状態) へと遷移させます。

以下に，option の指定形式を示します。

- option = T_NOOPT

  本システム・コールを発行したタスクは，cnt で指定された数の資源が獲得できるまで wait 状態 (資源待ち状態) へと遷移します。

- option = T_TMOUT

  本システム・コールを発行したタスクは，cnt で指定された数の資源が獲得できるまで wait 状態 (資源待ち状態) へと遷移します。

  ただし，本システム・コールの発行から p_tmout で指定された時間が経過した際には，資源待ち状態は強制的に解除され，wai_sem システム・コールの戻り値

として TE_TMOUT(0x10) が返されます。

注意　自タスクを対象セマフォの待ち行列にキューイングする際のキューイング方法は，セマフォ生成時に指定したキューイング方法（タスクの優先度順，または，FIFO 順）となります。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	資源を要求するセマフォのアクセス・アドレスが不正である
TE_AA	0x05	資源を要求するセマフォのアクセス・アドレスが 0x0 である
TE_PA	0x06	タイムアウト値を格納する領域のアドレスが 0x0 である
*TE_TMOUT	0x10	資源を獲得しないまま，指定された時間が経過した
*TE_DLT	0x13	資源の獲得を待っている間にセマフォが削除された
TE_OPT	0x14	オプション指定が不適当である
TE_PAR	0x1f	要求する資源数が 0x0 である

保守／廃止

Get Semaphore Access Address (29)

SEM_ADR

タスク／非タスク

概　要

セマフォのアクセス・アドレスを獲得する。

形　式

```
#include "stdrx.h"
int       sem_adr(pa_sem, semid);
```

パラメータ

I/O	パラメータ	説　　明
O	int　　*pa_sem;	指定したセマフォのアクセス・アドレスを格納する領域のアドレス
I	unsigned short semid;	セマフォ ID 番号 (0x1 ～ 0xefff)

説　明

semid で指定されたセマフォのアクセス・アドレスを pa_sem で指定された領域に格納します。

戻り値

*TE_OK	0x00	正常終了
TE_PA	0x06	セマフォのアクセス・アドレスを格納する領域のアドレスが 0x0 である
*TE_NOEXS	0x0a	指定したセマフォ ID 番号のセマフォは存在しない
TE_IDZR	0x0e	セマフォ ID 番号が 0x0 である
TE_IDOVR	0x0f	セマフォ ID 番号が 0x1 ～ 0xefff の範囲外である

保守／廃止

Create Mailbox (30)

CRE_MBX

タスク

概　要

メールボックスを生成する。

形　式

```
#include "stdrx.h"
int       cre_mbx(option, pa_mbx, mbxid);
```

パラメータ

I/O	パラメータ	説　　明
I	short　option;	命令オプション タスクの待ち方の指定方法 T_TPRI　(0x0) ： タスクの優先度順 T_TFIFO　(0x1) ： FIFO 順 メッセージの待ち方の指定方法 T_MPRI　(0x0) ： メッセージの優先度順 T_MFIFO　(0x2) ： FIFO 順
O	int　　*pa_mbx;	生成したメールボックスのアクセス・アドレスを格納する領域のアドレス
I	unsigned short mbxid;	メールボックス ID 番号 (0x1 ～ 0xefff)

説　明

mbxid で指定された ID 番号を持つメールボックスを生成したのち，pa_mbx で指定された領域にメールボックスのアクセス・アドレスを格納します。

なお，option には，本システム・コールの発行により生成したメールボックスからメッセージの受信を行うタスクが待ち行列 (タスク専用待ち行列) にキューイングされる際のキューイング方法，および，メッセージの送信を行うタスクのメッセージが待ち行列 (メッセージ専用待ち行列) にキューイングされる際のキューイング方法を指定します。

以下に，option の指定形式を示します。

- option = (T_TPRI | T_MPRI)

  rcv_msg システム・コールを発行した際，ただちにメッセージを受信することができなかった場合には，タスクの優先度順で待ち行列にキューイングされます。

  また，snd_msg システム・コールを発行した際，ただちにメッセージを受け取るタスクが存在しなかった場合には，メッセージの優先度順で待ち行列にキュー

イングされます。

- option = (T_TPRI | T_MFIFO)

  rcv_msg システム・コールを発行した際，ただちにメッセージを受信することができなかった場合には，タスクの優先度順で待ち行列にキューイングされます。

  また，snd_msg システム・コールを発行した際，ただちにメッセージを受け取るタスクが存在しなかった場合には，FIFO 順 (メッセージがメールボックスに到着した順) で待ち行列にキューイングされます。

- option = (T_TFIFO | T_MPRI)

  rcv_msg システム・コールを発行した際，ただちにメッセージを受信することができなかった場合には，FIFO 順 (メッセージの受信要求を行った順) で待ち行列にキューイングされます。

  また，snd_msg システム・コールを発行した際，ただちにメッセージを受け取るタスクが存在しなかった場合には，メッセージの優先度順で待ち行列にキューイングされます。

- option = (T_TFIFO | T_MFIFO)

  rcv_msg システム・コールを発行した際，ただちにメッセージを受信することができなかった場合には，FIFO 順 (メッセージの受信要求を行った順) で待ち行列にキューイングされます。

  また，snd_msg システム・コールを発行した際，ただちにメッセージを受け取るタスクが存在しなかった場合には，メッセージの優先度順で待ち行列にキューイングされます。

戻り値

*TE_OK	0x00	正常終了
*TE_MEM	0x01	管理ブロック用の領域が確保できない
TE_PA	0x06	生成したメールボックスのアクセス・アドレスを格納する領域のアドレスが 0x0 である
*TE_EXS	0x09	同じメールボックス ID 番号のメールボックスがすでに存在している
TE_IDZR	0x0e	メールボックス ID 番号が 0x0 である
TE_IDOVR	0x0f	メールボックス ID 番号が 0x1 ～ 0xefff の範囲外である
TE_OPT	0x14	オプション指定が不適当である

保守／廃止

Delete Mailbox (31)

DEL_MBX

タスク

概　要

メールボックスを削除する。

形　式

```
#include "stdrx.h"
int       del_mbx(a_mbx);
```

パラメータ

I/O	パラメータ	説　　明
I	int　　a_mbx;	削除するメールボックスのアクセス・アドレス

説　明

a_mbx で指定されたメールボックスを削除します。

注意　本システム・コールを発行した際，対象メールボックスのタスク用待ち行列にメッセージの受信を待っているタスク (rcv_msg システム・コールを発行したタスク) が存在した場合，待っているタスクを wait 状態 (メッセージ待ち状態) から ready 状態へと遷移させたのち，rcv_msg システム・コールの戻り値として TE_DLT(0x13) を返しています。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	削除するメールボックスのアクセス・アドレスが不正である
TE_AA	0x05	削除するメールボックスのアクセス・アドレスが 0x0 である

保守／廃止

Send Message to Mailbox (32)

SND_MSG

タスク／非タスク

概　要

メッセージを送信する。

形　式

```
#include "stdrx.h"
int       snd_msg(a_mbx, a_msg);
```

パラメータ

I/O	パラメータ	説　明
I	int　a_mbx;	メッセージを送信するメールボックスのアクセス・アドレス
I	char　*a_msg;	メールボックスに送信するメッセージのアドレス

説　明

a_mbx で指定されたメールボックスに a_msg で指定されたメッセージを送信します。

なお，対象メールボックスにメッセージを送信した際，対象メールボックスのタスク専用待ち行列にタスクがキューイングされていた場合には，先頭タスクにメッセージを渡したのち，先頭タスクを wait 状態（メッセージ待ち状態）から ready 状態へと遷移させます。

また，対象メールボックスのタスク専用待ち行列にタスクがキューイングされていなかった場合には，送信したメッセージを対象メールボックスのメッセージ専用待ち行列にキューイングします。

注意
- メッセージを対象メールボックスのメッセージ専用待ち行列にキューイングする際のキューイング方法は，メールボックス生成時に指定したキューイング方法（メッセージの優先度順，または，FIFO 順）となります。
- RX830 では，タスク間通信機能を実現するうえで，メッセージを管理するためのヘッダ領域を必要とします。このため，RX830 では，メッセージとして使用するメモリ領域は，get_blk システム・コールにより獲得したメモリ・ブロックを使用することを推奨しています。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	メッセージを送信するメールボックスのアクセス・アドレス，または，メールボックスに送信するメッセージのアドレスが不正である

TE_AA	0x05	メッセージを送信するメールボックスのアクセス・アドレスが0x0である
TE_MA	0x07	メールボックスに送信するメッセージのアドレスが0x0である

保守／廃止

Receive Message from Mailbox (33)

RCV_MSG

タスク

概　要

メッセージを受信する。

形　式

```
#include "stdrx.h"
int       rcv_msg(option, a_msg, a_mbx, p_tmout);
```

パラメータ

I/O	パラメータ	説　　明
I	short　option;	命令オプション タイムアウトの指定方法 T_NOOPT (0x0) ： タイムアウト指定なし T_TMOUT (0x1) ： タイムアウト指定あり
O	char　**a_msg;	受信したメッセージのアドレスを格納する領域のアドレス
I	int　a_mbx;	メッセージを受信するメールボックスのアクセス・アドレス
I	unsigned long　*p_tmout;	タイムアウト値が格納されている領域のアドレス (タイムアウト値の単位： msec)

説　明

a_mbx で指定したメールボックスからメッセージを受信したのち，a_msg で指定された領域に受信したメッセージのアドレスを格納します。

ただし，本システム・コールを発行した際，対象メールボックスからメッセージを受信することができなかった場合には，自タスクを対象メールボックスのタスク専用待ち行列にキューイングしたのち，run 状態から wait 状態 (メッセージ待ち状態) へと遷移させます。

以下に，option の指定形式を示します。

- option = T_NOOPT

  本システム・コールを発行したタスクは，メッセージが受信できるまで wait 状態 (メッセージ待ち状態) へと遷移します。

- option = T_TMOUT

  本システム・コールを発行したタスクは，メッセージが受信できるまで wait 状態 (メッセージ待ち状態) へと遷移します。

ただし，本システム・コールの発行からp_tmputで指定された時間が経過した際には，メッセージ待ち状態は強制的に解除され，rcv_msgシステム・コールの戻り値としてTE_TMOUT(0x10)が返されます。

注意
- 自タスクを対象メールボックスのタスク専用待ち行列にキューイングする際のキューイング方法は，メールボックス生成時に指定したキューイング方法（タスクの優先度順，または，FIFO順）となります。
- RX830では，タスク間通信機能を実現するうえで，メッセージを管理するためのヘッダ領域を必要とします。このため，RX830では，メッセージとして使用するメモリ領域は，get_blkシステム・コールにより獲得したメモリ・ブロックを使用することを推奨しています。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	メッセージを受信するメールボックスのアクセス・アドレスが不正である
TE_AA	0x05	メッセージを受信するメールボックスのアクセス・アドレスが0x0である
TE_PA	0x06	受信したメッセージのアドレスを格納する領域のアドレス，または，タイムアウト値を格納する領域のアドレスが0x0である
*TE_TMOUT	0x10	メッセージを受信しないまま，指定された時間が経過した
*TE_DLT	0x13	メッセージの到着を待っている間にメールボックスが削除された
TE_OPT	0x14	オプション指定が不適当である

保守／廃止

Get Mailbox Access Address (34)

MBX_ADR

タスク／非タスク

概　要

メールボックスのアクセス・アドレスを獲得する。

形　式

```
#include "stdrx.h"
int       mbx_adr(pa_mbx, mbxid);
```

パラメータ

I/O	パラメータ	説　　明
O	int　　*pa_mbx;	指定したメールボックスのアクセス・アドレスを格納する領域のアドレス
I	unsigned short mbxid;	メールボックスID番号 (0x1 ～ 0xefff)

説　明

mbxidで指定されたメールボックスのアクセス・アドレスをpa_mbxで指定された領域に格納します。

戻り値

*TE_OK	0x00	正常終了
TE_PA	0x06	メールボックスのアクセス・アドレスを格納する領域のアドレスが0x0である
*TE_NOEXS	0x0a	指定したメールボックスID番号のメールボックスは存在しない
TE_IDZR	0x0e	メールボックスID番号が0x0である
TE_IDOVR	0x0f	メールボックスID番号が0x1 ～ 0xefffの範囲外である

保守／廃止

12.6.3　割り込み処理管理システム・コール

本項では，表12-7に示す割り込み処理管理システム・コールについて説明しています。

表 12-7　割り込み処理管理システム・コール

番号	システム・コール	機　　能
35	def_int	間接起動割り込みハンドラを登録する
36	ret_int	直接起動割り込みハンドラから復帰する
37	set_int	割り込み許可レベルを設定する

保守／廃止

Define Interrupt Handler (35)

DEF_INT

タスク／非タスク

概　要

間接起動割り込みハンドラを登録する。

形　式

```
#include "stdrx.h"
int       def_int(level, pk_dfint);
```

パラメータ

I/O	パラメータ	説　　明
I	short　level;	割り込みレベル (0x0 ～ 0xf)
I	struct　t_dfint *pk_dfint;	パラメータ・パケットのアドレス

t_dfint の構造

```
struct  t_dfint {
        int     (*inthdr)();   /* 間接起動割り込みハンドラの先頭アドレス   */
        char    *p_intdata;    /* 間接起動割り込みハンドラの gp レジスタ値 */
};
```

説　明

level で指定されたマスカブル割り込みが発生した際に起動される間接起動割り込みハンドラを登録します。

なお，inthdr には間接起動割り込みハンドラの先頭アドレスを，p_intdata には間接起動割り込みハンドラの gp レジスタ値を指定します。

ただし，p_intdata に 0x0 を指定した場合は，本システム・コール発行時の gp レジスタ値が間接起動割り込みハンドラの gp レジスタ値として設定されます。

注意　本システム・コールを非タスクから発行する場合，p_intdata に 0x0 を指定することはできません。

戻り値

*TE_OK	0x00	正常終了
TE_PA	0x06	パラメータ・パケットのアドレスが 0x0 である
TE_SA	0x08	間接起動割り込みハンドラの先頭アドレスが 0x0 である
TE_IPRI	0x18	割り込みレベルが 0x0 ～ 0xf の範囲外である

保守／廃止

Return from Interrupt Handler (36)

RET_INT

直接起動割り込みハンドラ

概　要

直接起動割り込みハンドラから復帰する。

形　式

```
#include "stdrx.h"
void      ret_int();
```

パラメータ

なし

説　明

直接起動割り込みハンドラから復帰します。

注意
- 本システム・コールでは，割り込みコントローラに対するEOIコマンドの発行を行っていません。このため，本システム・コールを発行する前にEOIコマンドの発行を行う必要があります。
- 直接起動割り込みハンドラをアセンブリ言語で記述した場合，直接起動割り込みハンドラからの復帰は，以下のように記述します。

```
jr      _ret_int
```

戻り値

なし

保守／廃止

Set Interrupt Level (37)

SET_INT

タスク／非タスク

概　要

割り込み許可レベルを設定する。

形　式

```
#include "stdrx.h"
int       set_int(intlvl);
```

パラメータ

I/O	パラメータ	説　　明
I	short　intlvl;	割り込みを許可するレベル (0x0 ～ 0x10)

説　明

intlvl で指定されたレベルをプロセッサ (V830 ファミリ ™) の割り込み許可レベルとして設定します。

なお，V830 ファミリ ™ は，レベル 0x0 からレベル 0xf までの 16 種類の割り込み要因を持ち，PSW 内の割り込み許可フラグ (I3 ～ I0) を操作することにより，割り込み要求をレベル指定で許可／禁止することができます。そこで，本システム・コールでは，intlvl に 0x0 ～ 0xf の値 (0xn) が指定された場合は，レベル 0x0 からレベル 0xn-1 までの割り込み要求を禁止し，レベル 0xn 以上の割り込み要求は許可しています。

しかし，V830 ファミリ ™ の仕様上，割り込み許可フラグのみの操作では，レベル 0xf の割り込み要求を禁止することができません。そこで，本システム・コールでは，intlvl に 0x10 が指定された場合には，PSW 内の割り込み禁止フラグ (ID) を 1 に設定することにより，すべての割り込み要求を禁止しています。

戻り値

*TE_OK	0x00	正常終了
TE_IPRI	0x18	割り込みを許可するレベルの指定が不適当である

保守／廃止

12.6.4　例外処理管理システム・コール

本項では，表 12-8に示す例外処理管理システム・コールについて説明しています。

表 12-8　例外処理管理システム・コール

番号	システム・コール	機　　能
41	def_exc	例外ハンドラを登録／登録解除する

保守／廃止

Define Exception Handler (41)

DEF_EXC

タスク／非タスク

概　要

例外ハンドラを登録／登録解除する。

形　式

```
#include "stdrx.h"
int       def_exc(option, exchdr, p_excdata);
```

パラメータ

I/O	パラメータ	説　　明
I	short　option;	命令オプション 例外種別の指定 T_CPU (0x0) : CPU 例外 T_OS (0x1) : システム・コール例外 起動対象の指定 T_TEXC (0x0) : 各タスク固有の例外処理 T_NEXC (0x2) : 非タスク部分の例外処理 T_CEXC (0x4) : 全タスク共通の例外処理
I	int　(*exchdr)();	例外ハンドラの先頭アドレス
I	char　*p_excdata;	例外ハンドラの gp レジスタ値

説　明

option で指定された例外 (CPU 例外，システム・コール例外) が発生した際に起動される例外ハンドラの登録／登録解除を行います。

なお，option には例外種別，および，起動対象を，exchdr には例外ハンドラの先頭アドレスを，p_excdata には例外ハンドラの gp レジスタ値を指定します。

ただし，exchdr に 0x0 を指定した場合は，すでに登録されている例外ハンドラの登録解除が行われます。

また，p_excdata に 0x0 を指定した場合は，本システム・コール発行時の gp レジスタ値が例外ハンドラの gp レジスタ値として設定されます。

以下に，option の指定形式を示します。

- option = (T_CPU | T_NEXC)

  非タスク CPU 例外ハンドラの登録／登録解除となります。

なお，非タスクCPU例外ハンドラは，非タスク内でCPU例外が発生した際に起動される例外ハンドラであるため，すべての非タスクに共通したCPU例外発生時処理を記述します。

- option = (T_CPU | T_CEXC)

  全タスク共通CPU例外ハンドラの登録／登録解除となります。

  なお，全タスク共通CPU例外ハンドラは，タスク内でCPU例外が発生した際に起動される例外ハンドラであるため，すべてのタスクに共通したCPU例外発生時処理を記述します。

- option = (T_OS | T_TEXC)

  タスク固有システム・コール例外ハンドラの登録／登録解除となります。

  なお，タスク固有システム・コール例外ハンドラは，本システム・コールを発行したタスク内でシステム・コール例外が発生が発生した際に起動される例外ハンドラであるため，各タスクに依存したシステム・コール例外発生時処理を記述します。

- option = (T_OS | T_NEXC)

  非タスク・システム・コール例外ハンドラの登録／登録解除となります。

  なお，非タスク・システム・コール例外ハンドラは，非タスク内でシステム・コール例外が発生した際に起動される例外ハンドラであるため，すべての非タスクに共通したシステム・コール例外発生時処理を記述します。

- option = (T_OS | T_CEXC)

  全タスク共通システム・コール例外ハンドラの登録／登録解除となります。

  なお，全タスク共通システム・コール例外ハンドラは，タスク内でシステム・コール例外が発生した際に起動される例外ハンドラであるため，すべてのタスクに共通したシステム・コール例外発生時処理を記述します。

注意
- 本システム・コールを非タスクから発行する場合，p_excdataに0x0を指定することはできません。
- RX830では，タスク固有システム・コール例外ハンドラと全タスク共通システム・コール例外ハンドラの両方が登録されていた場合，タスク固有システム・コール例外ハンドラのみを起動します。
- RX830では，タスク固有CPU例外ハンドラを登録／登録解除することができません。このため，optionに(T_CPU | T_TEXC)を指定した場合には，戻り値としてTE_OPT(0x14)が返されます。

戻り値

*TE_OK	0x00	正常終了
TE_OPT	0x14	オプション指定が不適当である

保守／廃止

12.6.5 メモリ管理システム・コール

本項では，表12-9に示すメモリ管理システム・コールについて説明しています。

表 12-9 メモリ管理システム・コール

番号	システム・コール	機能
43	cre_mpl	ユーザ・メモリ・プールを生成する
44	del_mpl	ユーザ・メモリ・プールを削除する
45	get_blk	メモリ・ブロックを獲得する
46	rel_blk	メモリ・ブロックを解放する
47	mpl_adr	ユーザ・メモリ・プールのアクセス・アドレスを獲得する

保守／廃止

Create Memory Pool (43)

CRE_MPL

タスク

概　要

ユーザ・メモリ・プールを生成する。

形　式

```
#inclde   "packet.h"
int       cre_mpl(option, pa_mpl, mplid, pk_crmpl);
```

パラメータ

I/O	パラメータ	説　　明
I	short　option;	命令オプション タスクの待ち方の指定方法 T_TPRI (0x0) ： タスクの優先度順 T_TFIFO (0x1) ： FIFO 順
O	int　*pa_mpl;	生成したユーザ・メモリ・プールのアクセス・アドレスを格納する領域のアドレス
I	unsigned short mplid;	メモリ・プール ID 番号 (0x1 ～ 0xefff)
I	struct t_crmpl *pk_crmpl;	パラメータ・パケットのアドレス

t_crmpl の構造

```
struct  t_crmpl {
  unsigned long   mplsz;  /* ユーザ・メモリ・プール全体のサイズ (単位： byte) */
  unsigned short  blksz;  /* 基本ブロック・サイズ (単位： byte)               */
};
```

説　明

mplid で指定された ID 番号を持つユーザ・メモリ・プールを生成したのち，pa_mpl で指定された領域にユーザ・メモリ・プールのアクセス・アドレスを格納します。

なお，option には本システム・コールの発行により生成したユーザ・メモリ・プールからメモリ・ブロックの獲得を行うタスクが待ち行列にキューイングされる際のキューイング方法を，mplsz にはユーザ・メモリ・プールの全体サイズを，blksize には基本ブロック・サイズ (ユーザ・メモリ・プールからメモリ・ブロックを獲得する際の最小単位) を指定します。

以下に，option の指定形式を示します。

- option = T_TPRI

  get_blk システム・コールを発行した際，ただちに要求する数のメモリ・ブロックを獲得することができなかった場合には，タスクの優先度順で待ち行列にキューイングされます。

- option = T_TFIFO

  get_blk システム・コールを発行した際，ただちに要求する数のメモリ・ブロックを獲得することができなかった場合には，FIFO 順（メモリ・ブロックの獲得要求を行った順）で待ち行列にキューイングされます。

注意　RX830 では，ユーザ・メモリ・プールからメモリ・ブロックを獲得／解放する際，16 の整数倍を単位として行っています。このため，RX830 では，blksz に指定する値を 16 の整数倍にすることを推奨しています。

戻り値

*TE_OK	0x00	正常終了
*TE_MEM	0x01	管理ブロック用の領域，または，ユーザ・メモリ・プール用の領域が確保できない
TE_PA	0x06	生成したユーザ・メモリ・プールのアクセス・アドレスを格納する領域のアドレス，または，パラメータ・パケットのアドレスが 0x0 である
*TE_EXS	0x09	同じメモリ・プール ID 番号のユーザ・メモリ・プールがすでに存在している
TE_IDZR	0x0e	メモリ・プール ID 番号が 0x0 である
TE_IDOVR	0x0f	メモリ・プール ID 番号が 0x1 ～ 0xefff の範囲外である
TE_OPT	0x14	オプション指定が不適当である
TE_MPLSZ	0x1a	ユーザ・メモリ・プール全体のサイズが基本ブロック・サイズよりも小さい
TE_PAR	0x1f	ユーザ・メモリ・プール全体のサイズ，または，基本ブロック・サイズが 0x0 である

保守／廃止

Delete Memory Pool (44)

DEL_MPL

タスク

概　要

ユーザ・メモリ・プールを削除する。

形　式

```
#include "stdrx.h"
int       del_mpl(a_mpl);
```

パラメータ

I/O	パラメータ	説　　明
I	int　　a_mpl;	削除するユーザ・メモリ・プールのアクセス・アドレス

説　明

a_mpl で指定されたユーザ・メモリ・プールを削除します。

注意　本システム・コールを発行した際，対象ユーザ・メモリ・プールの待ち行列にメモリ・ブロックの獲得を待っているタスク (get_blk システム・コールを発行したタスク) が存在した場合，待っているタスクを wait 状態 (メモリ・ブロック待ち状態) から ready 状態へと遷移させたのち，get_blk システム・コールの戻り値として TE_DLT (0x13) を返しています。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	削除するユーザ・メモリ・プールのアクセス・アドレスが不正である
TE_AA	0x05	削除するユーザ・メモリ・プールのアクセス・アドレスが 0x0 である
TE_IDZR	0x0e	ユーザ・メモリ・プール 0 番を削除しようとした
TE_SYS	0x1e	ニュークリアス・メモリ・プール，または，スタティックに生成したユーザ・メモリ・プールを削除しようとした

保守／廃止

Get Memory Block (45)

GET_BLK

タスク

概　要

メモリ・ブロックを獲得する。

形　式

```
#include "stdrx.h"
int       get_blk(option, pa_blk, a_mpl, blkcnt, p_tmout);
```

パラメータ

I/O	パラメータ	説　　明
I	short option;	命令オプション タイムアウトの指定方法 T_NOOPT (0x0) ： タイムアウト指定なし T_TMOUT (0x1) ： タイムアウト指定あり
O	char **pa_blk;	獲得したメモリ・ブロックのメモリ・ブロック・アドレスを格納する領域のアドレス
I	int a_mpl;	メモリ・ブロックを獲得するユーザ・メモリ・プールのアクセス・アドレス
I	unsigned short blkcnt;	要求するメモリ・ブロック数
I	unsigned long *p_tmout;	タイムアウト値が格納されている領域のアドレス (タイムアウト値の単位： msec)

説　明

a_mpl で指定されたユーザ・メモリ・プールから blkcnt で指定された数のメモリ・ブロックを獲得します。

なお，リターン・パラメータとして，pa_blk で指定された領域に獲得したメモリ・ブロックのメモリ・ブロック・アドレスが格納されます。

ただし，本システム・コールを発行した際，対象ユーザ・メモリ・プールから要求した数のメモリ・ブロックを獲得するができなかった場合には，自タスクを対象ユーザ・メモリ・プールの待ち行列にキューイングしたのち，run 状態から wait 状態 (メモリ・ブロック待ち状態) へと遷移させます。

以下に，option の指定形式を示します。

- option = T_NOOPT

  本システム・コールを発行したタスクは，blkcnt で指定された数のメモリ・ブロックが獲得できるまで wait 状態 (メモリ・ブロック待ち状態) へと遷移します。

- option = T_TMOUT

  本システム・コールを発行したタスクは，blkcnt で指定された数のメモリ・ブロックが獲得できるまで wait 状態 (メモリ・ブロック待ち状態) へと遷移します。

  ただし，本システム・コールの発行から p_tmout で指定された時間が経過した際には，メモリ・ブロック待ち状態は強制的に解除され，get_blk システム・コールの戻り値として TE_TMOUT(0x10) が返されます。

注意　自タスクを対象ユーザ・メモリ・プールの待ち行列にキューイングする際のキューイング方法は，ユーザ・メモリ・プール生成時に指定したキューイング方法 (タスクの優先度順，または，FIFO 順) となります。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	メモリ・ブロックを獲得するユーザ・メモリ・プールのアクセス・アドレスが不正である
TE_AA	0x05	メモリ・ブロックを獲得するユーザ・メモリ・プールのアクセス・アドレスが 0x0 である
TE_PA	0x06	獲得したメモリ・ブロックのメモリ・ブロック・アドレスを格納する領域のアドレス，または，タイムアウト値を格納する領域のアドレスが 0x0 である
*TE_TMOUT	0x10	メモリ・ブロックを獲得しないまま，指定された時間が経過した
*TE_DLT	0x13	メモリ・ブロックの獲得を待っている間にユーザ・メモリ・プールが削除された
TE_OPT	0x14	オプション指定が不適当である
TE_PAR	0x1f	ユーザ・メモリ・プールから獲得するメモリ・ブロックの数が 0x0 である

保守／廃止

Release Memory Block (46)

REL_BLK

タスク／非タスク

概　要

メモリ・ブロックを解放する。

形　式

```
#include "stdrx.h"
int       rel_blk(a_blk);
```

パラメータ

I/O	パラメータ	説　　明
I	char　*a_blk;	解放するメモリ・ブロックのメモリ・ブロック・アドレス

説　明

a_blk で指定されたメモリ・ブロックを解放します。

なお，対象ユーザ・メモリ・プールにメモリ・ブロックを解放した際，対象ユーザ・メモリ・プールの待ち行列にキューイングされている先頭タスクの待ち条件を満足した場合には，先頭タスクにメモリ・ブロックを渡したのち，先頭タスクを wait 状態 (メモリ・ブロック待ち状態) から ready 状態へと遷移させます。

戻り値

*TE_OK	0x00	正常終了
*TE_OBJ	0x04	解放するメモリ・ブロックのメモリ・ブロック・アドレスが不正である
TE_MA	0x07	解放するメモリ・ブロックのメモリ・ブロック・アドレスが 0x0 である

保守／廃止

Get Memory Pool Access Address (47)

MPL_ADR

タスク／非タスク

概　要

ユーザ・メモリ・プールのアクセス・アドレスを獲得する。

形　式

```
#include "stdrx.h"
int       mpl_adr(pa_mpl, mplid);
```

パラメータ

I/O	パラメータ	説　　明
O	int　　*pa_mpl;	指定したユーザ・メモリ・プールのアクセス・アドレスを格納する領域のアドレス
I	unsigned short mplid;	メモリ・プール ID 番号 (0x0 ～ 0xefff)

説　明

mplid で指定されたユーザ・メモリ・プールのアクセス・アドレスを pa_mpl で指定された領域に格納します。

戻り値

*TE_OK	0x00	正常終了
TE_PA	0x06	ユーザ・メモリ・プールのアクセス・アドレスを格納する領域のアドレスが 0x0 である
*TE_NOEXS	0x0a	指定したメモリ・プール ID 番号のユーザ・メモリ・プールは存在しない
TE_IDOVR	0x0f	メモリ・プール ID 番号が 0x0 ～ 0xefff の範囲外である

保守／廃止

12.6.6 時間管理システム・コール

本項では，表12-10に示す時間管理システム・コールについて説明しています。

表 12-10 時間管理システム・コール

番号	システム・コール	機能
48	set_tim	システム・クロックに時刻を設定する
49	get_tim	システム・クロックの時刻を読み出す

保守／廃止

Set Time (48)

SET_TIM

タスク／非タスク

概　要

システム・クロックに時刻を設定する。

形　式

```
#include "stdrx.h"
int       set_tim(pk_time);
```

パラメータ

I/O	パラメータ	説　　明
I	struct t_time *pk_time;	設定する時刻が格納されているパケットのアドレス (設定する時刻の単位：msec)

t_time の構造

```
struct  t_time {
        unsigned long   ltime;  /* 時刻 (下位 32 ビット) */
        unsigned short  utime;  /* 時刻 (上位 16 ビット) */
};
```

説　明

pk_time で指定された時刻をシステム・クロックの現時刻として設定します。

なお，RX830 が管理するシステム・クロックは，48 ビット幅で構成されています。そこで，下位 32 ビットを ltime に，上位 16 ビットを utime に設定します。

戻り値

*TE_OK	0x00	正常終了
TE_PA	0x06	設定する時刻が格納されているパケットのアドレスが 0x0 である

保守／廃止

Get Time (49)

GET_TIM

タスク／非タスク

概　要

システム・クロックの時刻を読み出す。

形　式

```
#include "stdrx.h"
int       get_tim(pk_time);
```

パラメータ

I/O	パラメータ	説　　明
O	struct t_time *pk_time;	読み出した時刻を格納するパケットのアドレス (読み出した時刻の単位： msec)

t_time の構造

```
struct  t_time {
        unsigned long   ltime;  /* 時刻（下位 32 ビット）*/
        unsigned short  utime;  /* 時刻（上位 16 ビット）*/
};
```

説　明

システム・クロックの現時刻を pk_time で指定された領域に格納します。

なお，RX830 が管理するシステム・クロックは，48 ビット幅で構成されています。このため，下位 32 ビットが ltime に，上位 16 ビットが utime に格納されます。

戻り値

*TE_OK	0x00	正常終了
TE_PA	0x06	読み出した時刻を格納するパケットのアドレスが 0x0 である

保守／廃止

12.6.7　バージョン管理システム・コール

本項では，表12-11に示すバージョン管理システム・コールについて説明しています。

表 12-11　バージョン管理システム・コール

番号	システム・コール	機　　能
50	get_ver	RX830 のバージョン情報を獲得する

保守／廃止

Get Version Number (50)

GET_VER

タスク／非タスク

概　要

RX830 のバージョン情報を獲得する。

形　式

```
#include "stdrx.h"
int       get_ver(pk_ver);
```

パラメータ

I/O	パラメータ	説　　明
O	struct t_ver *pk_ver;	獲得したバージョン情報を格納するパケットのアドレス

t_ver の構造

```
struct  t_ver {
        short   maker;      /* メーカ                    */
        short   rfu;        /* システム予約              */
        short   spver;      /* TRON 仕様書のバージョン   */
        short   prver;      /* 製品のバージョン          */
        short   prno[4];    /* 製品管理番号              */
        short   cpu;        /* プロセッサ                */
        short   var;        /* バリエーション記述子      */
};
```

説　明

RX830 のバージョン情報を pk_ver で指定された領域に格納します。
以下に，バージョン情報の詳細を示します。

パラメータ	数値	説　　明
maker	0x000d	日本電気 (株)
spver	0x1111	ITRON1 Ver1.11
prver	0x0100	RX830 Ver1.00
prno	不定	製品の出荷管理をするためのシリアル番号 (64 ビット幅)
cpu	0x0d32	μPD705100
var	0x0003	シングル・プロセッサ用，ITRON 外核サポート，アスキー版

戻り値

*TE_OK	0x00	正常終了
TE_PA	0x06	獲得したバージョン情報を格納するパケットのアドレスが0x0である

保守／廃止

12.6.8 複合システム・コール

本項では，表12-12に示す複合システム・コールについて説明しています。

表 12-12 複合システム・コール

番号	システム・コール	機能
56	iret_wup	起床要求の発行，および，直接起動割り込みハンドラからの復帰を行う

保守／廃止

Return and Wakeup Task (56)

IRET_WUP

直接起動割り込みハンドラ

概　要

起床要求の発行，および，直接起動割り込みハンドラからの復帰を行う。

形　式

```
#include "stdrx.h"
#define  iret_wup(a_tsk)  return(a_tsk)
```

パラメータ

I/O	パラメータ	説　　明
I	int　　a_tsk;	起床させるタスクのアクセス・アドレス

説　明

a_tsk で指定されたタスクに起床要求を発行 (起床要求カウンタに 0x1 を加算) したのち，直接起動割り込みハンドラから復帰します。

ただし，本システム・コールを発行した際，a_tsk で指定されたタスクが wait 状態 (時限待ち状態，または，起床待ち状態) であった場合には，起床要求の発行 (起床要求カウンタへの加算処理) は行わず，対象タスクを時限待ち状態，または，起床待ち状態から ready 状態へと遷移させます。

注意
- RX830 が管理する起床要求カウンタは，8 ビット幅で構成されています。このため，本システム・コールでは，起床要求数が 0xff を越えるような場合には，起床要求カウンタへの加算処理は行わず，エラーとしても扱いません。
- 本システム・コールでは，割り込みコントローラに対する EOI コマンドの発行を行っていません。このため，本システム・コールを発行する前に EOI コマンドの発行を行う必要があります。
- 直接起動割り込みハンドラをアセンブリ言語で記述した場合，起床要求の発行，および，直接起動割り込みハンドラからの復帰は，以下のように記述します。

```
mov     a_tsk, r6
jr      _iret_wup
```

戻り値

なし

保守／廃止

12.6.9 外核システム・コール

本項では，表12-13に示す外核システム・コールについて説明しています。

表12-13 外核システム・コール

番号	システム・コール	機能
64	def_svc	拡張SVCハンドラを登録／登録解除する

保守／廃止

Define Supervisor Call Handler (64)

DEF_SVC

タスク／非タスク

概　要

拡張 SVC ハンドラを登録／登録解除する。

形　式

```
#include "stdrx.h"
int       def_svc(fncode, pk_dfsvc);
```

パラメータ

I/O	パラメータ	説　　明
I	short　fncode;	拡張 SVC ハンドラの機能コード (0xe0 ～ 0xff)
I	struct　t_dfsvc *pk_dfsvc;	パラメータ・パケットのアドレス

t_dfsvc の構造

```
struct  t_dfsvc {
        int     (*svchdr)(); /* 拡張 SVC ハンドラの先頭アドレス   */
        char    *p_svcdata;  /* 拡張 SVC ハンドラの gp レジスタ値 */
};
```

説　明

fncode で指定された機能コードを持つ拡張 SVC ハンドラの登録／登録解除を行います。

なお，svchdr には拡張 SVC ハンドラの先頭アドレスを，p_svcdata には拡張 SVC ハンドラの gp レジスタ値を指定します。

ただし，svchdr に 0x0 を指定した場合は，すでに登録されている拡張 SVC ハンドラの登録解除が行われます。

また，p_svcdata に 0x0 を指定した場合は，本システム・コール発行時の gp レジスタ値が拡張 SVC ハンドラの gp レジスタ値として設定されます。

注意
- 本システム・コールを非タスクから発行する場合，p_svcdata に 0x0 を指定することはできません。
- C 言語で記述された処理プログラム (タスク，割り込みハンドラなど) から拡張 SVC ハンドラを呼び出す場合，専用のインタフェース・プログラム (拡張 SVC ハンドラ用インタフェース・ライブラリ) が必要となります。

保守／廃止

戻り値

*TE_OK	0x00	正常終了
*TE_MEM	0x01	管理ブロック用の領域が確保できない
TE_PA	0x06	パラメータ・パケットのアドレスが 0x0 である
TE_SVC	0x40	機能コードが不適当である

付録 A　プログラミングのために

この章では，日本電気（株）製 V830 ファミリ™ 用 C コンパイラ CA830，または，Green Hills Software Inc. 製 C クロス V830 ファミリ™ コンパイラ CCV830 を使用した際の処理プログラムの記述方法について説明しています。

A.1　概要

RX830 では，処理プログラムを用途別に，

- タスク

  RX830 の管理下で実行可能な処理プログラムの最小単位です。

- 終了時処理ルーチン

  `abo_tsk`，または，`ter_tsk` システム・コール発行時，ただちに起動される「タスクの後処理専用ルーチン」です。

- 割り込みハンドラ

  割り込み発生時，ただちに起動される「割り込み処理専用ルーチン」です。

- 例外ハンドラ

  例外発生時，ただちに起動される「例外処理専用ルーチン」です。

- 拡張 SVC ハンドラ

  ユーザが拡張システム・コールとして登録した関数です。

  また，拡張 SVC ハンドラには，専用のインタフェース・ライブラリ（拡張 SVC ハンドラ用インタフェース・ライブラリ）が必要となります。

などと呼び，区別しています。

なお，これらの処理プログラムは，一般的，あるいは，RX830 を使用するうえでの約束ごとなどにより，それぞれに基本型があります。

A.2　タスク

A.2.1　CA830 対応版の場合

タスクを C 言語で記述する場合は，プラグマ指令による関数宣言を行ったのち，int 型の引き数を 1 つ持った void 型の関数として記述します。

なお，引き数 (*initcode*) には，sta_tsk システム・コール発行時に指定された初期情報 (起動コード) が設定されます。

図 A-1に，タスクの基本型 (C 言語) を示します。

図 A-1　タスクの基本型　CA830

```
#include            <stdrx.h>

#pragma rtos_task   func_task

void
func_task(int initcode)
{
        /* タスク func_task の処理 */
        ..........................
        ..........................
        ..........................

        /* タスク func_task の終了 */
        ext_tsk();
}
```

注意　プラグマ指令による関数宣言についての詳細は，「V800 シリーズ　C コンパイラ・パッケージ　ユーザーズ・マニュアル　C 言語編」を参照してください。

また，タスクをアセンブリ言語で記述する場合は，CA830のコール・コンベンションに従った関数として記述します。

なお，引き数(r6レジスタ)には，sta_tskシステム・コール発行時に指定された初期情報(起動コード)が設定されます。

図A-2に，タスクの基本型(アセンブリ言語)を示します。

図 A-2　タスクの基本型 CA830

```
.include        "stdrx.inc"

        .text
        .align  4
        .globl  _func_task
_func_task :
        # タスク func_task の処理
        ........................
        ........................
        ........................

        # タスク func_task の終了
        jr      _ext_tsk
```

注意　プロローグ，および，エピローグにおけるレジスタの退避／復帰，スタックの切り替えなどといった処理は記述する必要がありません。

また，タスク内でシステム・コールを発行する場合は，以下の形式で記述します。

```
jal     _システム・コール名
```

A.2.2 CCV830 対応版の場合

タスクをC言語で記述する場合は，int 型の引き数を1つ持った void 型の関数として記述します。

なお，引き数 (*initcode*) には，sta_tsk システム・コール発行時に指定された初期情報 (起動コード) が設定されます。

図 A-3に，タスクの基本型 (C 言語) を示します。

図 A-3　タスクの基本型　CCV830

```
#include            <stdrx.h>

void
func_task(int initcode)
{
        /* タスク func_task の処理 */
        ..........................
        ..........................
        ..........................

        /* タスク func_task の終了 */
        ext_tsk();
}
```

また，タスクをアセンブリ言語で記述する場合は，CCV830 のコール・コンベンションに従った関数として記述します。

なお，引き数 (r6 レジスタ) には，sta_tsk システム・コール発行時に指定された初期情報 (起動コード) が設定されます。

図 A-4に，タスクの基本型 (アセンブリ言語) を示します。

図 A-4　タスクの基本型 CCV830

```
.include        "stdrx.inc"

        .text
        .align  4
        .globl  _func_task
_func_task :
        # タスク func_task の処理
        ........................
        ........................
        ........................

        # タスク func_task の終了
        jr      _ext_tsk
```

注意　プロローグ，および，エピローグにおけるレジスタの退避／復帰，スタックの切り替えなどといった処理は記述する必要がありません。

また，タスク内でシステム・コールを発行する場合は，以下の形式で記述します。

```
jal     _システム・コール名
```

A.3　終了時処理ルーチン

A.3.1　CA830 対応版の場合

終了時処理ルーチンを C 言語で記述する場合は，プラグマ指令による関数宣言を行ったのち，int 型の引き数を 1 つ持った void 型の関数として記述します。

なお，引き数 (*abocode*) には，abo_tsk，または，ter_tsk システム・コール発行時に指定された情報 (異常終了コード) が設定されます。

図 A-5に，終了時処理ルーチンの基本型 (C 言語) を示します。

図 A-5　終了時処理ルーチンの基本型　CA830

```
#include            <stdrx.h>

#pragma rtos_task   func_extrtn

void
func_extrtn(int abocode)
{
        /* 終了時処理ルーチン func_extrtn の処理 */
        ..........................................
        ..........................................
        ..........................................

        /* 終了時処理ルーチン func_extrtn の終了 */
        ext_tsk();
}
```

注意　プラグマ指令による関数宣言についての詳細は，「V800 シリーズ　C コンパイラ・パッケージ　ユーザーズ・マニュアル　C 言語編」を参照してください。

また，終了時処理ルーチンをアセンブリ言語で記述する場合は，CA830 のコール・コンベンションに従った関数として記述します。

なお，引き数 (r6 レジスタ) には，abo_tsk，または，ter_tsk システム・コール発行時に指定された情報 (異常終了コード) が設定されます。

図 A-6に，終了時処理ルーチンの基本型 (アセンブリ言語) を示します。

図 A-6　終了時処理ルーチンの基本型 CA830

```
.include        "stdrx.inc"

        .text
        .align  4
        .globl  _func_extrtn
_func_extrtn :
        # 終了時処理ルーチン func_extrtn の処理
        .....................................
        .....................................
        .....................................

        # 終了時処理ルーチン func_extrtn の終了
        jr      _ext_tsk
```

注意　プロローグ，および，エピローグにおけるレジスタの退避／復帰，スタックの切り替えなどといった処理は記述する必要がありません。
また，終了時処理ルーチン内でシステム・コールを発行する場合は，以下の形式で記述します。

```
jal     _システム・コール名
```

A.3.2 CCV830 対応版の場合

終了時処理ルーチンを C 言語で記述する場合は，int 型の引き数を 1 つ持った void 型の関数として記述します。

なお，引き数 (*abocode*) には，abo_tsk，または，ter_tsk システム・コール発行時に指定された情報 (異常終了コード) が設定されます。

図 A-7に，終了時処理ルーチンの基本型 (C 言語) を示します。

図 A-7　終了時処理ルーチンの基本型 CCV830

```
#include                <stdrx.h>

void
func_extrtn(int  abocode)
{
        /* 終了時処理ルーチン func_extrtn の処理 */
        ..........................................
        ..........................................
        ..........................................

        /* 終了時処理ルーチン func_extrtn の終了 */
        ext_tsk();
}
```

また，終了時処理ルーチンをアセンブリ言語で記述する場合は，CCV830 のコール・コンベンションに従った関数として記述します。

なお，引き数 (r6 レジスタ) には，abo_tsk，または，ter_tsk システム・コール発行時に指定された情報 (異常終了コード) が設定されます。

図 A-8に，終了時処理ルーチンの基本型 (アセンブリ言語) を示します。

図 A-8　終了時処理ルーチンの基本型　CCV830

```
.include        "stdrx.inc"

        .text
        .align  4
        .globl  _func_extrtn
_func_extrtn :
        # 終了時処理ルーチン func_extrtn の処理
        .....................................
        .....................................
        .....................................

        # 終了時処理ルーチン func_extrtn の終了
        jr      _ext_tsk
```

注意　プロローグ，および，エピローグにおけるレジスタの退避／復帰，スタックの切り替えなどといった処理は記述する必要がありません。

また，終了時処理ルーチン内でシステム・コールを発行する場合は，以下の形式で記述します。

```
jal     _システム・コール名
```

A.4 直接起動割り込みハンドラ

A.4.1 CA830 対応版の場合

直接起動割り込みハンドラをC言語で記述する場合は，プラグマ指令による関数宣言を行ったのち，引き数を持たない void 型の関数として記述します。

図 A-9に，直接起動割り込みハンドラの基本型 (C 言語) を示します。

図 A-9 直接起動割り込みハンドラの基本型 CA830

```
#include                    <stdrx.h>

#pragma rtos_interrupt   int_name      func_inthdr    GP = __gp_DATA

__rtos_interrupt
void
func_inthdr(void)
{
        /* 直接起動割り込みハンドラの func_inthdr の処理 */
        ..................................................
        ..................................................
        ..................................................

        /* 直接起動割り込みハンドラ func_inthdr からの復帰 */
        ret_int();
}
```

注意 プラグマ指令による関数宣言についての詳細は，「V800 シリーズ Cコンパイラ・パッケージ ユーザーズ・マニュアル C 言語編」を参照してください。

また，直接起動割り込みハンドラをアセンブリ言語で記述する場合は，CA830 のコール・コンベンションに従った関数として記述します。

図 A-10に，直接起動割り込みハンドラの基本型（アセンブリ言語）を示します。

図 A-10　直接起動割り込みハンドラの基本型 CA830

```
.include        "stdrx.inc"

        .text
        .align  4
        .globl  _func_inthdr
_func_inthdr :
        # 直接起動割り込みハンドラ func_inthdr の前処理
        RTOS_IntEntry

        # グローバル・ポインタ gp の設定
        ..........................................
        ..........................................

        # 直接起動割り込みハンドラ func_inthdr の処理
        ..........................................
        ..........................................
        ..........................................

        # グローバル・ポインタ gp の復帰
        ..........................................
        ..........................................

        # 直接起動割り込みハンドラ func_inthdr からの復帰
        RTOS_IntReturn
```

注意　プロローグ，および，エピローグにおけるレジスタの退避／復帰，スタックの切り替えなどといった処理は記述する必要がありません。
また，直接起動割り込みハンドラ内でシステム・コールを発行する場合は，以下の形式で記述します。

```
jal     _システム・コール名
```

なお、直接起動割り込みハンドラ内でシステム・コールを使用しない場合は RTOS_Int_Entry、 RTOS_IntReturn は必要ありません。ただし、プロローグ，および，エピローグにおけるレジスタの退避／復帰，スタックの切り替えなどといった処理は記述する必要があります。またこの場合直接起動割り込みハンドラからの復帰には reti を使用します。

A.4.2 CCV830 対応版の場合

直接起動割り込みハンドラを C 言語で記述することはできません。したがって，クロス・ツールに CCV830 を使用する場合は，直接起動割り込みハンドラの記述はアセンブリ言語となります。

なお，直接起動割り込みハンドラをアセンブリ言語で記述する場合は，CCV830 のコール・コンベンションに従った関数として記述します。

図 A-11に，直接起動割り込みハンドラの基本型 (アセンブリ言語) を示します。

図 A-11　直接起動割り込みハンドラの基本型 CCV830

```
.include        "stdrx.inc"

        .text
        .align  4
        .globl  _func_inthdr
_func_inthdr :
        # 直接起動割り込みハンドラ func_inthdr の前処理
        RTOS_IntEntry

        # グローバル・ポインタ gp の設定
        ..........................................
        ..........................................

        # 直接起動割り込みハンドラ func_inthdr の処理
        ..........................................
        ..........................................
        ..........................................

        # グローバル・ポインタ gp の復帰
        ..........................................
        ..........................................

        # 直接起動割り込みハンドラ func_inthdr からの復帰
        RTOS_IntReturn
```

注意　プロローグ，および，エピローグにおけるレジスタの退避／復帰，スタックの切り替えなどといった処理は記述する必要がありません。
また，直接起動割り込みハンドラ内でシステム・コールを発行する場合は，以下の形式で記述します。

```
jal     _システム・コール名
```

なお、直接起動割り込みハンドラ内でシステム・コールを使用しない場合は_RTOS_Int_Entry、RTOS_IntReturn は必要ありません。ただし、プロローグ，および，エピローグにおけるレジスタの退避／復帰，スタックの切り替えなどといった処理は記述する必要があります。またこの場合直接起動割り込みハンドラからの復帰には reti を使用します。

A.5 間接起動割り込みハンドラ

A.5.1 CA830 対応版／CCV830 対応版の場合

間接起動割り込みハンドラをC言語で記述する場合は，引き数を持たないint型の関数として記述します。

図A-12に，間接起動割り込みハンドラの基本型（C言語）を示します。

図 A-12 間接起動割り込みハンドラの基本型 CA830／CCV830

```
#include                <stdrx.h>

int
func_inthdr(void)
{
        /* 間接起動割り込みハンドラ func_inthdr の処理 */
        ..........................................
        ..........................................
        ..........................................

        /* 間接起動割り込みハンドラ func_inthdr からの復帰 */
        return(0x0);
}
```

注意 間接起動割り込みハンドラ終了時に他タスクを起床する場合は間接割り込みハンドラ終了時にreturn()文のパラメータに起床するタスクのアクセス・アドレスを指定します。

また，間接起動割り込みハンドラをアセンブリ言語で記述する場合は，CA830／CCV830のコール・コンベンションに従った関数として記述します。

図A-13に，間接起動割り込みハンドラの基本型（アセンブリ言語）を示します。

図 A-13　間接起動割り込みハンドラの基本型　CA830／CCV830

```
.include        "stdrx.inc"

        .text
        .align  4
        .globl  _func_inthdr
_func_inthdr :
        # 間接起動割り込みハンドラ func_inthdr の処理
        ..........................................
        ..........................................
        ..........................................

        # 間接起動割り込みハンドラ func_inthdr からの復帰
        mov     r0, r10
        jmp     [lp]
```

注意　プロローグ，および，エピローグにおけるレジスタの退避／復帰，スタックの切り替えなどといった処理は記述する必要がありません。

また，間接起動割り込みハンドラ内でシステム・コールを発行する場合は，以下の形式で記述します。

```
jal     _システム・コール名
```

間接起動割り込みハンドラ終了時に他タスクを起床する場合は間接割り込みハンドラ終了時にr10にr0ではなく起床するタスクのアクセス・アドレスを格納します。

A.6 CPU 例外ハンドラ

A.6.1 CA830 対応版／CCV830 対応版の場合

CPU 例外ハンドラを C 言語で記述する場合は，CPU 例外情報パケット (**pk_exccpuinfo*) を引き数として持った int 型の関数として記述します。

なお，引き数 (**pk_exccpuinfo*) には，CPU 例外が発生した際に，CPU 例外情報パケットへのアクセス・アドレスが設定されます。

図 A-14に，CPU 例外ハンドラの基本型 (C 言語) を示します。

図 A-14 CPU 例外ハンドラの基本型 CA830 ／ CCV830

```
#include                <stdrx.h>

int
func_exchdr(struct t_exccpuinfo *pk_exccpuinfo)
{
            /* CPU 例外ハンドラ func_exchdr の処理 */
            ..........................................
            ..........................................
            ..........................................

            /* CPU 例外ハンドラ func_exchdr からの復帰 */
            return(0x0);
}
```

また，CPU 例外ハンドラをアセンブリ言語で記述する場合は，CA830 ／ CCV830 のコール・コンベンションに従った関数として記述します。

なお，引き数 (r6 レジスタ) には，CPU 例外が発生した際に，CPU 例外情報パケットへのアクセス・アドレスが設定されます。

図 A-15に，CPU 例外ハンドラの基本型 (アセンブリ言語) を示します。

図 A-15　CPU 例外ハンドラの基本型　CA830 ／ CCV830

```
.include        "stdrx.inc"

        .text
        .align  4
        .globl  _func_exchdr
_func_exchdr :
        # システム・コール例外ハンドラ func_exchdr の処理
        ..................................................
        ..................................................
        ..................................................

        # システム・コール例外ハンドラ func_exchdr の終了
        mov     r0, r10
        jmp     [lp]
```

注意　プロローグ，および，エピローグにおけるレジスタの退避／復帰，スタックの切り替えなどといった処理は記述する必要がありません。

また，CPU 例外ハンドラ内でシステム・コールを発行する場合は，以下の形式で記述します。

```
jal     _システム・コール名
```

A.7　システム・コール例外ハンドラ

A.7.1　CA830 対応版／CCV830 対応版の場合

システム・コール例外ハンドラを C 言語で記述する場合は，システム・コール例外情報パケットへのアクセス・アドレス (**pk_excsysinfo*) を引き数として持った int 型の関数として記述します。

なお，引き数 (**pk_excsysinfo*) には，システム・コール例外が発生した際に，システム・コール例外情報パケットへのアクセス・アドレスが設定されます。

図 A-16に，システム・コール例外ハンドラの基本型 (C 言語) を示します。

図 A-16　システム・コール例外ハンドラの基本型　CA830 ／ CCV830

```
#include            <stdrx.h>

int
func_exchdr(struct t_excsysinfo *pk_excsysinfo)
{
        /* システム・コール例外ハンドラ func_exchdr の処理 */
        ..........
        ..........
        ..........

        /* システム・コール例外ハンドラ func_exchdr からの復帰 */
        return(0x0);
}
```

また，システム・コール例外ハンドラをアセンブリ言語で記述する場合は，CA830 ／ CCV830 のコール・コンベンションに従った関数として記述します。

なお，引き数 (r6 レジスタ) には，システム・コール例外が発生した際に，システム・コール例外情報パケットへのアクセス・アドレスが設定されます。

図 A-17に，システム・コール例外ハンドラの基本型 (アセンブリ言語) を示します。

図 A-17　システム・コール例外ハンドラの基本型　CA830 ／ CCV830

```
.include        "stdrx.inc"

        .text
        .align  4
        .globl  _func_exchdr
_func_exchdr :
        # システム・コール例外ハンドラ func_exchdr の処理
        ..................................................
        ..................................................
        ..................................................

        # システム・コール例外ハンドラ func_exchdr の終了
        mov     r0, r10
        jmp     [lp]
```

注意　プロローグ，および，エピローグにおけるレジスタの退避／復帰，スタックの切り替えなどといった処理は記述する必要がありません。

また，システム・コール例外ハンドラ内でシステム・コールを発行する場合は，以下の形式で記述します。

```
jal     _システム・コール名
```

A.8　拡張 SVC ハンドラ

A.8.1　CA830 対応版／ CCV830 対応版の場合

拡張 SVC ハンドラを C 言語で記述する場合は，int 型の関数として記述します。
図 A-18に，拡張 SVC ハンドラ (C 言語) の基本型を示します。

図 A-18　拡張 SVC ハンドラの基本型　CA830 ／ CCV830

```
#include            <stdrx.h>

int
func_svchdr(int arg_A, int arg_B)
{
        int         ret;

        /* 拡張 SVC ハンドラ func_svchdr の処理 */
        ..........................................
        ..........................................
        ..........................................

        /* 拡張 SVC ハンドラ func_svchdr からの復帰 */
        return(ret);
}
```

また，拡張SVCハンドラをアセンブリ言語で記述する場合は，CA830／CCV830のコール・コンベンションに従った関数として記述します。

図A-19に，拡張SVCハンドラ（アセンブリ言語）の基本型を示します。

図 A-19　拡張SVCハンドラの基本型　CA830／CCV830

```
.include        "stdrx.inc"

        .text
        .align  4
        .globl  _func_svchdr
_func_svchdr :
        # 拡張SVCハンドラ func_svchdr の処理
        .....................................
        .....................................
        .....................................

        # 拡張SVCハンドラ func_svchdr の終了
        mov     ret, r10
        jmp     [lp]
```

注意　プロローグ，および，エピローグにおけるレジスタの退避／復帰，スタックの切り替えなどといった処理は記述する必要がありません。
また，拡張SVCハンドラ内でシステム・コールを発行する場合は，以下の形式で記述します。

```
jal     _システム・コール名
```

保守／廃止

A.9 拡張SVCハンドラ用インタフェース・ライブラリ

A.9.1 CA830対応版／CCV830対応版の場合

拡張SVCハンドラ用インタフェース・ライブラリでは，次のような処理を行います。
ただし，拡張SVCハンドラの引き数の渡し方は，使用するクロス・ツールに準じます。

- 拡張SVCハンドラの機能コードをr10レジスタに設定
- 拡張SVCハンドラからの戻り番地をlpレジスタに設定
- 拡張SVCハンドラのパラメータをチェック
- 拡張SVCハンドラの引き数領域のサイズをr11レジスタに設定
- 拡張SVCハンドラの入口のアドレス（hpレジスタ + 0x110番地）を獲得
- 拡張SVCハンドラの入口にジャンプ

また，拡張SVCハンドラのパラメータをチェックした際にエラーを検出した場合，次のような処理を行います。

- 拡張SVCハンドラの機能コードをr10レジスタの上位16ビットに設定
- 検出したエラーに対応したエラー・コードをr10レジスタの下位16ビットに設定
- 拡張SVCハンドラ例外処理の入口のアドレス（hpレジスタ + 0x128番地）を獲得
- 拡張SVCハンドラからの戻り番地をr11レジスタに設定
- 拡張SVCハンドラ例外処理の入口にジャンプ

図A-20に，CA830を使用した場合，および，CCV830を使用した場合の拡張SVCハンドラ用インタフェース・ライブラリの基本型を示します。
ただし，図A-20に示した拡張SVCハンドラ用インタフェース・ライブラリは，jal命令で呼び出されることを想定しているため，lpレジスタに拡張SVCハンドラからの戻り番地を設定する処理は行っていません。

図 A-20　拡張 SVC ハンドラ用インタフェース・ライブラリの基本型　CA830 ／ CCV830

```
#include        <stdrx.h>
#include        <user.h>

        .text
        .align  2
        .globl  _svc_call
-- 拡張 SVC ハンドラ用インタフェース・ライブラリ
_svc_call:
        mov     fncode, r10         -- 機能コードの設定

        -- パラメータ・チェック

        mov     prmsize, r11        -- 引き数領域のサイズの設定
        ld.w    0x110[hp], r12      -- 拡張 SVC ハンドラの入り口のアドレスの獲得
        jmp     [r12]               -- 拡張 SVC ハンドラの入り口にジャンプ


-- エラー処理
_svc_err:
        shl     0x10, r10           -- 機能コードの設定
        or      err_code, r10       -- エラー・コードの設定
        ld.w    0x128[hp], r12      -- 例外処理の入り口のアドレスの獲得
        mov     lp, r11             -- 戻り番地の設定
        jmp     [r12]               -- 例外処理の入り口にジャンプ
```

保守／廃止

索引

ろ

わ

保守／廃止

お問い合わせ先

【技術的なお問い合わせ先】

ＮＥＣ半導体テクニカルホットライン（インフォメーションセンター）
（電話：午前 9:00～12:00、午後 1:00～5:00）

電　話　：044-548-8899
FAX　：044-548-7900
E-mail　：s-info@saed.tmg.nec.co.jp

【営業関係お問い合わせ先】

半導体第一販売事業部 半導体第二販売事業部 半導体第三販売事業部	〒108-8001	東京都港区芝５－７－１　（日本電気本社ビル）	(03)3454-1111
中部支社 半導体第一販売部 半導体第二販売部	〒460-8525	愛知県名古屋市中区錦１－17－１　（日本電気中部ビル）	(052)222-2170 (052)222-2190
関西支社 半導体第一販売部 半導体第二販売部 半導体第三販売部	〒540-8551	大阪府大阪市中央区城見１－４－24　（日本電気関西ビル）	(06)6945-3178 (06)6945-3200 (06)6945-3208

北海道支社	札　幌	(011)251-5599	宇都宮支店	宇都宮	(028)621-2281	北陸支社	金　沢 (076)232-7303
東北支社	仙　台	(022)267-8740	小山支店	小　山	(0285)24-5011	京都支社	京　都 (075)344-7824
岩手支店	盛　岡	(019)651-4344	甲府支店	甲　府	(055)224-4141	神戸支社	神　戸 (078)333-3854
郡山支店	郡　山	(024)923-5511	長野支社	松　本	(0263)35-1662	中国支社	広　島 (082)242-5504
いわき支店	いわき	(0246)21-5511	静岡支社	静　岡	(054)254-4794	鳥取支店	鳥　取 (0857)27-5311
長岡支店	長　岡	(0258)36-2155	立川支社	立　川	(042)526-5981,6167	岡山支店	岡　山 (086)225-4455
水戸支店	水　戸	(029)226-1717	埼玉支社	大　宮	(048)649-1415	松山支店	松　山 (089)945-4149
土浦支店	土　浦	(0298)23-6161	千葉支社	千　葉	(043)238-8116	九州支社	福　岡 (092)261-2806
群馬支店	高　崎	(027)326-1255	神奈川支社	横　浜	(045)682-4524		
太田支店	太　田	(0276)46-4011	三重支店	津	(059)225-7341		

保守／廃止

アンケート記入のお願い

お手数ですが，このドキュメントに対するご意見をお寄せください。今後のドキュメント作成の参考にさせていただきます。

［ドキュメント名］RX830（ITRON1）ユーザーズ・マニュアル　基礎編
（U11730JJ3V1UM00（第3版））

［お名前など］（さしつかえのない範囲で）

御社名（学校名，その他）　（　　）
ご住所　（　　）
お電話番号　（　　）
お仕事の内容　（　　）
お名前　（　　）

1. ご評価（各欄に○をご記入ください）

項　　目	大変良い	良　い	普　通	悪　い	大変悪い
全体の構成					
説明内容					
用語解説					
調べやすさ					
デザイン，字の大きさなど					
その他（　　）					
（　　）					

2. わかりやすい所（第　章，第　章，第　章，第　章，その他　　）
理由［　　］

3. わかりにくい所（第　章，第　章，第　章，第　章，その他　　）
理由［　　］

4. ご意見，ご要望

5. このドキュメントをお届けしたのは
NEC販売員，特約店販売員，NEC半導体ソリューション技術本部員，
その他（　　）

ご協力ありがとうございました。
下記あてにFAXで送信いただくか，最寄りの販売員にコピーをお渡しください。

NEC半導体テクニカルホットライン
FAX：（044）548-7900

キリトリ