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

RX78K0

リアルタイム・オペレーティング・システム

基礎編

対象デバイス
78K0シリーズ

資料番号　U11537JJ3V1UM00（第3版）
発行年月　May 2005 N　CP(K)

〔メ　モ〕

目次要約

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

ITRONは，“ Industrial TRON ”の略称です。

μITRONは，“ Micro Industrial TRON ”の略称です。

TRON, ITRON，およびμITRONは，コンピュータの仕様に対する名称であり，特定の商品ないし商品群を指すものではありません。

μITRON仕様の著作権は（社）トロン協会に属しています。

その他，記載の会社名／製品名は，各社の商標あるいは登録商標です。

- 本資料に記載されている内容は2005年5月現在のもので，今後，予告なく変更することがあります。量産設計の際には最新の個別データ・シート等をご参照ください。
- 文書による当社の事前の承諾なしに本資料の転載複製を禁じます。当社は，本資料の誤りに関し，一切その責を負いません。
- 当社は，本資料に記載された当社製品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，一切その責を負いません。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。
- 本資料に記載された回路，ソフトウエアおよびこれらに関する情報は，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，ソフトウエアおよびこれらに関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，一切その責を負いません。
- 当社は，当社製品の品質，信頼性の向上に努めておりますが，当社製品の不具合が完全に発生しないことを保証するものではありません。当社製品の不具合により生じた生命，身体および財産に対する損害の危険を最小限度にするために，冗長設計，延焼対策設計，誤動作防止設計等安全設計を行ってください。
- 当社は，当社製品の品質水準を「標準水準」，「特別水準」およびお客様に品質保証プログラムを指定していただく「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。

  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

  特別水準：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

  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，生命維持のための装置またはシステム等

  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。意図されていない用途で当社製品の使用をお客様が希望する場合には，事前に当社販売窓口までお問い合わせください。

（注）

（1）本事項において使用されている「当社」とは，NECエレクトロニクス株式会社およびNEC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う。

（2）本事項において使用されている「当社製品」とは，（1）において定義された当社の開発，製造製品をいう。

M8E　02.11

はじめに

このたびは，NECエレクトロニクス 78K0シリーズの組み込み用ソフトウエアである「78K0シリーズ用リアルタイムOS RX78K0」をお買上げいただきまして誠にありがとうございます。

本マニュアルは，78K0シリーズ用リアルタイムOS　RX78K0の機能を，正しく理解していただくことを目的として書かれています。

対 象 者　本マニュアルは，デバイスのユーザーズ・マニュアル一読程度の知識があり，ソフトウエア・プログラミングの経験がある方を対象として書かれています。

ただし，本パッケージはリアルタイムOS単体であるため，実際に本リアルタイムOSをご使用になる場合には，“ CC78K0　Cコンパイラ ”と“ RA78K0　アセンブラ・パッケージ ”が必要になります。

構　　成　本マニュアルの構成を以下に示します。

第1章　RX78K0の概要

本リアルタイムOSの概要，特徴および構成について説明します。

第2章　ニュークリアス

ニュークリアスの機能，構成および初期化について説明します。

第3章　タスク管理

タスク管理の機能と状態管理について説明します。

第4章　同期通信管理

イベントフラグ，セマフォおよびメイルボックスの機能について説明します。

第5章　メモリ管理

メモリプールとメモリブロックについて説明します。

第6章　時間管理

指定時間経過または周期的にタスクを起床する機能について説明します。

第7章　割り込み処理

RX78K0が管理する割り込みについて説明します。

第8章　スケジューラ

スケジューラの機能について説明します。

第9章　システムコール

システムコールの機能と発行方法について説明します。

第10章　インタフェース・ライブラリ

インタフェース・ライブラリの機能と位置づけについて説明します。

付録A　マクロの利用方法

システムコールをアセンブラ定義マクロを利用して記述する方法を説明します。

付録B　C言語記述例

タスクと割り込みハンドラをC言語で記述した場合の例を示します。

付録C　周期ハンドラの記述方法

周期ハンドラの記述方法について説明します。

凡　例　**本マニュアル中で共通に使用される記号などの意味を示します。**

…　：同一の形式を繰り返す
[ ]　：[ ]内は省略可能
「 」　：「 」で囲まれた文字そのもの
“ ”　：“ ”で囲まれた文字そのもの
‘ ‘　：‘ ‘で囲まれた文字そのもの
( )　：( )で囲まれた文字そのもの
太文字　：文字そのもの
＿　：重要箇所，使用例での下線は入力文字
△　：1文字以上の空白
：　：プログラム記述の省略形
／　：区切り記号
＼　：バックスラッシュ

関連資料

開発ツール（ソフトウエア）の資料（ユーザーズ・マニュアル）

資　料　名		資料番号
RX78K0　リアルタイム・オペレーティング・システム	基礎編	このマニュアル
	インストレーション編	U11536J
	タスク・デバッガ編	U17569J

注意　上記関連資料は予告なしに内容を変更することがあります。設計などには，必ず最新の資料をご使用ください。

目　次

第1章　RX78K0の概要

制御機器分野のシステムでは，外部や内部の事象の変化に対して，すぐに反応できることが要求されます。従来のシステムは，このような要求に対して，単純な割り込みプログラムを使って対処してきました。しかし，応用機器がますます高性能・高機能になるにつれて，単純な割り込み処理だけでは対処しにくい問題が生じてきました。

つまり，システムが複雑化することにより，処理するプログラムが増大し，それらをどのような順序で実行させるかを管理することが大変になってきたということです。

この問題に対処するために考えられたのが，リアルタイム・オペレーティング・システム（以降，リアルタイムOS）です。リアルタイムOS は，事象の変化に即時に対応し，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主な仕事としています。

1. 1　マルチタスクOSとは

リアルタイムOSの管理下で実行される最小単位をタスクと呼び，1つのCPU上で複数のタスクを同時実行させることをマルチタスキングといいます。

CPU自体は，実際には一度に1つのプログラムしか実行させることはできません。しかし，複数のタスクの実行を，例えば一定時間ごとに切り替えるという処理を行うと，人間の目にはあたかも同時に複数のプログラムが実行されているかのように見えます。

このように，システム内で定めた何かの基準（この場合は，一定時間）を「きっかけ」とし，この「きっかけ」の発生を利用して実行タスクを切り替え，タスクの並列処理をすることができる機能を持つOSをマルチタスクOSといいます。マルチタスクOSは，タスクを並列に処理させることにより，システム全体の処理機能を向上させることを目的としています。

1. 2　RX78K0とは

RX78K0とは，完全なリアルタイム／マルチタスク機能を持ち，効率よいリアルタイム／マルチタスク処理環境の提供と，対象CPU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OSです。

RX78K0は，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いるため，ROM化を意識し，高速かつコンパクトなOSになっています。

1. 2. 1　RX78K0の特徴

次に，RX78K0の特徴を示します。

(1) μITRON Ver2.01 仕様に準拠

RX78K0のシステムコールは，組み込み制御用リアルタイムOSの仕様であるμITRON仕様に準拠しています。

(2) リアルタイム／マルチタスク処理実現機能の提供

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。

- 事象駆動・優先度方式によるスケジューリング
- 3種類の同期・通信機能
- メモリ管理機能
- 割り込み管理機能
- 時間管理機能

(3) ROM化を意識した設計

組み込み制御用のオペレーティング・システムであるため，コンパクトな設計になっています。また，使用しないシステムコールを取り除いたシステムを構築できます。

(4) アプリケーションをC言語で記述可能

C言語インタフェース・ライブラリを提供しているため，タスク，割り込みハンドラをC言語で記述できます。

(5) システム構築が容易

システムを構築する際に，ユーザの初期情報の設定を支援するコンフィギュレータを提供しています。

1.2.2　RX78K0の構成

RX78K0は，以下の3つのサブシステムから構成されています。

(1) ニュークリアス（核）

RX78K0の中心となる部分です。ターゲット・システムにアプリケーション・プログラムとともに組み込まれ，実際にリアルタイム／マルチタスク制御を行い，また，処理タスク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) C言語インタフェース・ライブラリ

C言語でユーザ・プログラムを記述する場合，システムコールの呼び出しはC言語の関数呼び出し形式をとります。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とには違いがあります。

C言語で記述されたシステムコール呼び出しを，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変換し，C言語で記述された処理タスクとニュークリアスを結合するためのプログラムが，C言語インタフェース・ライブラリです。

C言語インタフェース・ライブラリは，NECエレクトロニクスの78K0シリーズ用のCコンパイラ“CC78K0”が使用できるように記述されています。

(3) コンフィギュレータ

ニュークリアスが必要とする，ユーザ・システムの各情報をのせたテーブルを作成するためのツールです。

コンフィギュレータは，開発マシン上で動作し，OSが必要とするユーザ・システムの情報を対話形式で入力および修正することができます。

第2章　ニュークリアス

本章では，RX78K0の核部分であるニュークリアスについて説明します。

2. 1　ニュークリアスの機能概要

ニュークリアスとは，RX78K0の核部分をいい，実際にターゲット・システムにアプリケーションと共に組み込まれて使用されるOSの本体です。

主な仕事は，以下の2点です。

- ターゲット・システムの内外で起こった事象（イベント）に対応し，次に実行すべき最適な処理タスクを選び出すこと。
- アプリケーションから発行されたシステムコールに対応した各機能をサービスすること。

これらの仕事を行うのはスケジューラであり，後者の仕事を行うのが各管理モジュールであり，詳細については，第3章～第9章をご覧ください。

なお，RX78K0およびユーザタスクは，CPUが持っている4個のレジスタバンクの内，バンク0を共有して使用します。

2. 2　ニュークリアスの構成

図2－1に示すように，RX78K0のニュークリアスは，スケジューラおよび各管理モジュールから構成されています。

これらの機能は，システムコールを発行することにより利用することができます。

以下に，スケジューラおよび各管理モジュールの持つ機能について概要を示します。

(1) スケジューラ

タスクの実行順序を決定する部分で，タスクにCPUの使用権を与えます。タスクの実行順序を決定する方法は，優先度方式を採用しています。

(2) タスク管理

リアルタイムOSの処理単位であるタスクの起動・終了・実行を管理します。

(3) 同期通信管理

タスク間での同期・排他制御，あるいはメッセージの交換などの通信を行うために，イベントフラグ，セマフォ，メイルボックスの3種類の機構があります。

(4) 割り込み管理

RX78K0では，割り込みに対する応答性をあげるために，割り込みハンドラという処理手段を定義し，割り込みハンドラ専用のシステムコールを提供しています。

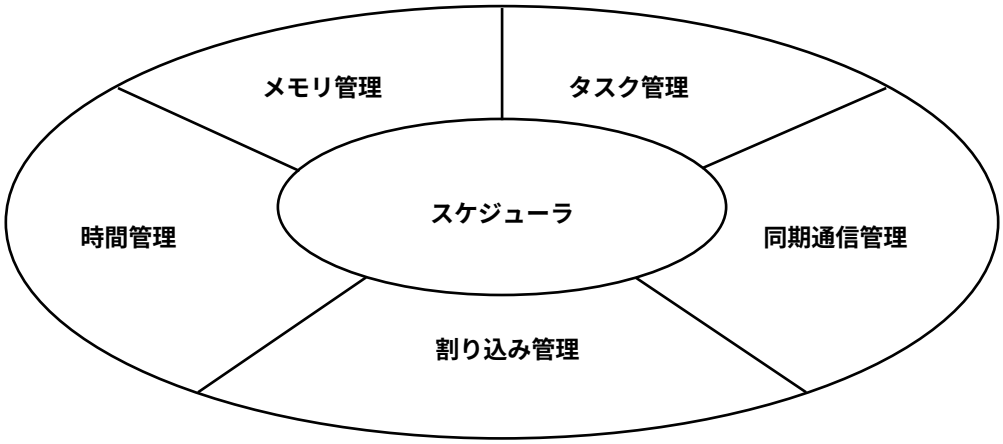
(5) 時間管理

一定の単位時間毎，または指定時間後に特定の処理を動作させるために，RX78K0では，タイマ・コントローラからの周期的な割り込み信号に同期した，タスクの遅延起床，割り込みハンドラの周期起床機能があります。

(6) メモリ管理

メモリブロックの集まりをメモリプールといいます。メモリが必要になったときに，メモリプールからメモリブロックを切り出して使用できるようなメモリ管理方法を有しています。

図2－1　ニュークリアスの構成



2.3　初期化

初期化は，ニュークリアスが動作を開始する上で必要な環境を作り上げる処理です。初期化処理を行うルーチンは，大きく分けて2つあります。1つは，ハードウエアなどの初期化を行うルーチンで，もう1つは，ニュークリアスの資源である管理テーブルなどの初期化を行うルーチンです。前者は，ユーザオウン・コーディングとなっており，後者は，ニュークリアス内に記述されているシステム初期化用ルーチンです。

システムが起動するとまず最初に，ハードウエアなどの初期化を行い，次にRX78K0のシステム初期化用ルーチンに分岐します。システム初期化用ルーチンでは初期化情報テーブルに従い，以下の資源について初期化します。

- タスク管理ブロックなどの各管理ブロックの生成・初期化
- ニュークリアスが管理する領域の初期化
- 初期ユーザ・タスクの生成

初期化終了後，スケジューラに制御が移り，初期タスクが動作を開始します。

初期化処理に関する詳細については，RX78K0　**リアルタイム・オペレーティング・システム　ユーザーズ・マニュアル　インストレーション編**（U11536J）を参照してください。

第3章　タスク管理

アプリケーション・プログラムの中では，1つのまとまった仕事をタスクとして記述します。したがって，タスクはアプリケーション・プログラムの構成要素です。

タスク管理とは，これらリアルタイムOSの処理単位であるタスクの起動，終了，実行を管理することをいいます。

3.1　タスク管理

タスクの管理は，大きく2つに分けて考えることができます。1つは，タスク自体の管理です。もう1つは，タスクが取り得るさまざまな状態の管理です。

以下に，RX78K0におけるタスクの管理方法について示します。

3.1.1　概　　要

タスクは実行実体であり，サイズなどが一様ではなく，直接管理することは困難です。ニュークリアスは，タスクを管理するにあたり，タスクと1対1に対応した論理実体を導入し，管理を行っています。この論理実体である管理オブジェクトを，タスク管理ブロック（TCB：Task Control Block）といいます。TCBは，ニュークリアス内に存在するシステム初期化用ルーチンにより，初期化情報テーブルに従い，リセット時に静的に生成されます。

TCBには，タスクを管理するための種々の情報が保持されています。以下に，TCBが保持している情報の例を示します。

- タスクの優先度
- タスクの状態
- 保持スタック・ポインタ
- 待ち時間
- 起床要求カウント

3.1.2　タスクの生成と起動

(1) 生成

タスクは，初期化情報テーブルに従い，システム初期化処理により静的に生成されます。

このようにして生成されたタスクは，dormant状態となります。

(2) 起動

起動は，sta_tskシステムコールによって行われます。このシステムコールの発行により，タスクはready状態となり，ニュークリアスのスケジューリングの対象となります。

3. 2　状態管理

タスクは，その動作を行う過程においてさまざまな状態に変化します。以下に，タスクが取り得る状態およびその管理方法について示します。

3. 2. 1　タスク状態

タスクはリアルタイムOSのもとで，タスクの実行に必要な資源の獲得の状況，事象の有り無しなどにより，状態を遷移します。したがって，リアルタイムOSは，各々のタスクが今どのような状態にあるかを管理する必要があります。

RX78K0では，タスクが遷移し得る状態を4通りに分けて管理しています。

(1) run（実行状態）

現在実行中の状態です。システム全体を通して，run状態のタスクは1つだけです。

(2) ready（実行可能状態）

スケジューリングの対象となる状態です。タスクでは実行する準備は整っていますが，このタスクよりも優先度の高い（同じ優先度の場合もありますが）タスクが実行されているために，実行権が割り当てられるのを待っている状態です。つまり，CPUが使用可能になればいつでも実行できる状態です。

(3) wait（待ち状態）

実行するために必要な条件が整わないため，実行できない状態です。すなわち，何らかの実行をするのに必要な条件が整うのを待っている状態です。

待ち状態からの実行再開は，待ち状態に入るために実行が中断した場所から行われます。その時，レジスタなどのプログラムを実行する状態を表現する各情報はそのまま復元されます。したがって，獲得していた資源もそのままとなります。

スリープ：	slp_tskシステムコールを発行したときに遷移する状態です。他タスクがwup_tsk, ret_wupシステムコールを発行しない限り，ready状態には遷移しません。
タイムアウト待ち：	wai_tskシステムコールを発行したときに遷移する状態です。他タスクからwup_tsk, ret_wupシステムコールを発行するか，または指定した時間の経過により，ready状態に遷移します。
イベントフラグ待ち：	wai_flg, cwai_flgシステムコールを発行して条件を満たされなかった場合に遷移する状態です。
セマフォ待ち：	wai_semシステムコールを発行して条件を満たされなかった場合に遷移する状態です。
メッセージ待ち：	rcv_msgシステムコールを発行して条件を満たされなかった場合に遷移する状態です。

(4) dormant（休止状態）

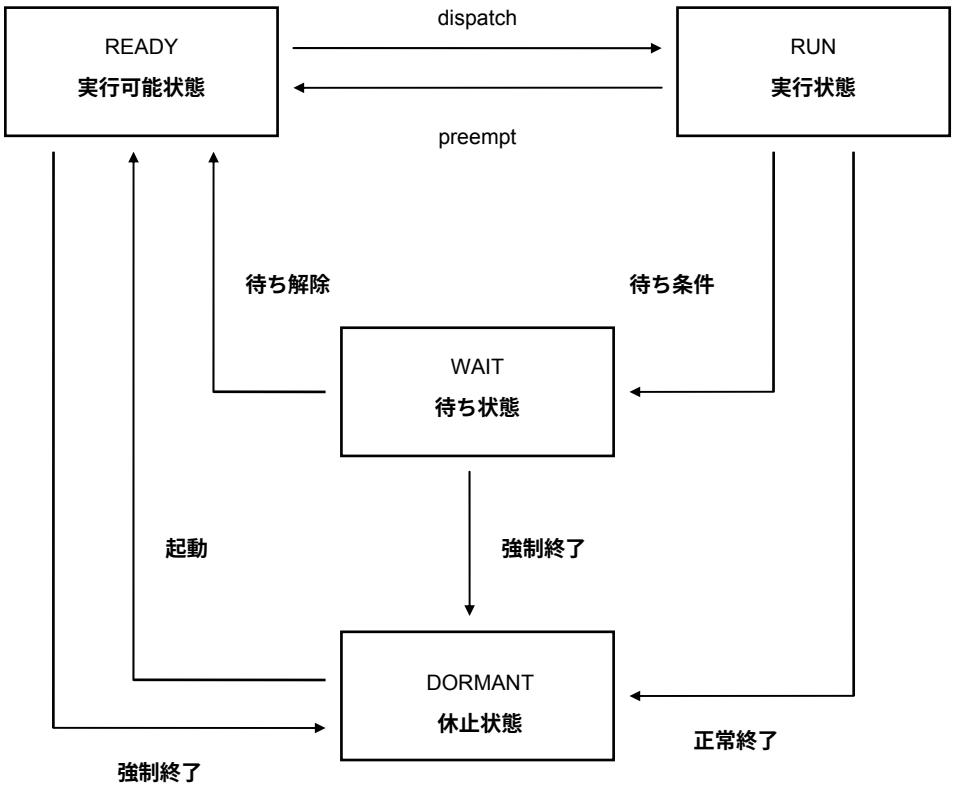
タスクが生成されたときの状態，または強制終了したときの状態です。dormant状態のタスクはスケジューリング対象から除外されます。

dormant状態のタスクは，sta_tskシステムコールを発行することによってready状態に遷移し，タスクの先頭から実行されます。

3. 2. 2　タスク状態とシステムコール

各タスクは，発行されたシステムコールにより4つの状態を遷移します。図3－1に状態遷移を示します。

図3－1　タスク状態遷移図



(1) 起動

dormant状態のタスクをready状態にすることをいいます。

(2) 終了

run状態のタスクをdormant状態にすることをいいます。

(3) dispatch（ディスパッチ）

ready状態のタスクの中で，次にrun状態に移行させるべきタスクを選び出し，run状態にすることをいいます。run状態のタスクは，ready状態のタスクの中で最高のプライオリティを持ちます。通常，この処理をディスパッチまたはスケジューリングと呼び，この処理部をディスパッチャまたはスケジューラと呼びます。

(4) preempt（プリエンプト）

現在実行中のタスクよりも優先度の高いタスクをrun状態に遷移させるために，現在実行中のタスクを一度中断し，ready状態に戻すことをいいます。一度ready状態に移ったタスクは，再度ディスパッチャにより選び出されるまでrun状態に移ることはできません。

(5) 待ち条件

イベント発生待ちの必要が生じると，タスクはrun状態からwait状態に移行します。イベントフラグのセットを待っている時，資源要求を行ったが要求分の資源が無い時，メッセージ受信要求を行ったがメッセージがまだ来ていない時などがこの場合に該当します。

(6) 待ち解除

wait状態のタスクをready状態にすることをいいます。この場合のイベントは，システムにより待ちが引き起こされた要因を解除する事象の発生です。

(7) 強制終了

run状態以外のタスクをdormant状態にすることをいいます。

3.3　タスクの終了

起動されていたタスクが，再びdormant状態に遷移することをタスクの終了とよびます。タスクの終了形態には，自タスクの正常終了（ext_tsk発行）と他タスクの異常終了（ter_tsk発行）の2種類があります。

3.3.1　正常終了

タスクを正常に終了したいときに，この終了形態をとります。つまりこのタスクは，全ての処理を完了し，スケジューリングの対象となる必要がなくなったときに，自分自身で終了する形態をいいます。

タスクの処理が終了したら，ext_tskシステムコールを発行します。これにより，タスクはdormant状態へ移行します。

3.3.2　異常終了

何らかの条件で他タスクを終了させたい場合，ter_tskシステムコールを発行し，異常終了という終了形態をとります。

これにより，対象タスクはdormant状態へ移行します。

3.4　タスク・コンテキスト

タスク・コンテキストとは，タスクがその処理を行うために必要な命令実行環境です。タスク・コンテキストは，CPUの汎用レジスタなどによって構成されています。

タスク・コンテキストは，タスク・スイッチが行われるときに，現在run状態のタスクのコンテキストが退避され，ディスパッチされるタスクのコンテキストがセットされます。

3. 5　アイドル状態

生成されたタスクがすべてready状態でなくなったとき，すなわち，スケジューリングされるべきタスクが1つも存在しなくなったときに，スケジューラはHALT命令を発行してアイドル状態になります。

RX78K0において，HALT状態から抜けるには，システム・クロックなどの割り込みにより，wait状態のタスクをready状態に遷移させなければなりません。

なお，HALT状態に移行する際のスタック・エリアは，直前までrun状態だったタスクのスタック・エリアを使用します。

第4章　同期通信管理

複数のタスクが並行に実行されている環境（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って，次に起動されるべきタスクに違いがでたり，タスク内での処理内容に違いがでるなど，タスク同士が実行の条件を制限しあったり，相互に関係しているという場合が多くあります。このため，タスクが他タスクの実行結果がでるまで実行を中断したり，処理を継続するうえで必要な条件が整うまで待つというように，他タスクとの連絡機能が必要となります。これを同期機能といい，一般に待ち合わせ機能と排他制御機能があります。RX78K0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，他タスクから処理結果を知らせてもらうというように，タスクとタスクの間での情報をやり取りすることが必要な場合もあります。これをタスク間通信機能とよびます。RX78K0では，タスク間通信機能としてメイルボックスを提供しています。

これらのオブジェクトは，システム初期化時に静的に生成されます。

4.1　イベントフラグ

マルチタスク処理では，1つのタスクの作業終了を待って，他タスクが動き始めるといったように，タスク間で実行を待ち合わせる機能が必要になります。このような場合，タスクの処理終了という事象（イベント）が発生したか否かが，他タスクでわかるような機能が必要であり，RX78K0ではこのような同期をとるために，イベントフラグを提供しています。

イベントフラグは1ビットのフラグであり，そのビットがセットされるのを，複数のタスクが待つことが可能です。また，一度セットされたイベントフラグは，システムコールによってクリアされるまでセットされたままになります。

イベントフラグに対する操作は，システムコールによってタスクと割り込みハンドラから行われます。以下に，イベントフラグ関連システムコールを示します。

set_flg	イベントフラグをセットする
iset_flg	〃　（割り込みハンドラで使用する）
clr_flg	イベントフラグをクリアする
wai_flg	イベントフラグの条件が満足するのを待つ（クリア無し）
cwai_flg	〃　（クリア有り）
pol_flg	イベントフラグを得る（クリア無し）
cpol_flg	〃　（クリア有り）

4.1.1　事象の発生待ち

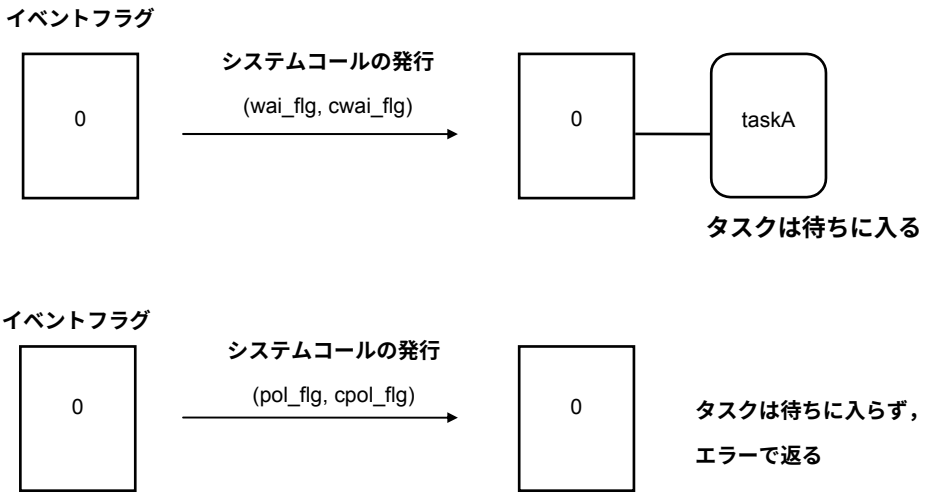
RX78K0では，タスクはイベントフラグがセットされるまで待つことができます。

wai_flg, cwai_flgシステムコールを発行したタスクは，イベントフラグがセットされていれば正常終了し，セットされていない場合はwait状態に入りセットされるのを待ちます。ただし，cwai_flgシステムコールを発行したタスクが待ち状態から解除されるときには，イベントフラグは0にクリアされます。

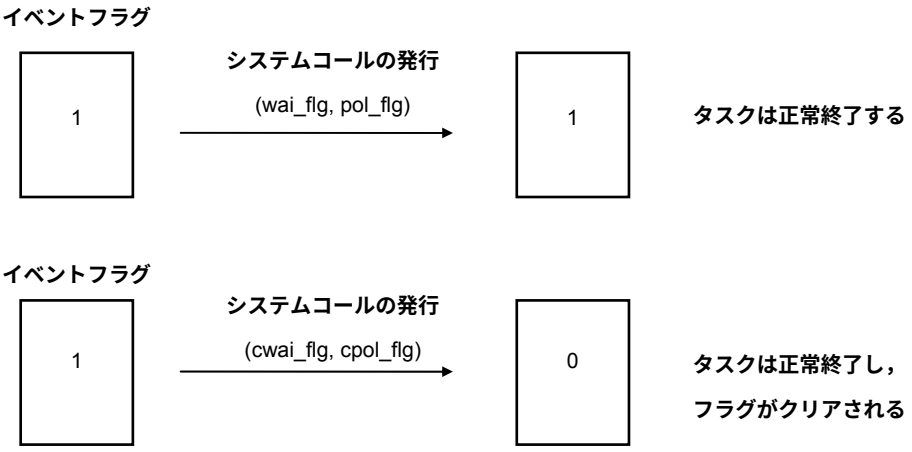
一方，pol_flg, cpol_flgシステムコールを発行したタスクは，イベントフラグがセットされていれば正常終了し，セットされていない場合はwait状態に入らずエラー・コードを持ってタスクに戻ります。ただし，cpol_flgシステムコールが発行された時は，イベントフラグがセットされている場合は0にクリアされます。

1つのイベントフラグの条件成立を待つタスクの数に制限はありません。

（1）イベントフラグがセットされていない場合



（2）イベントフラグがセットされている場合

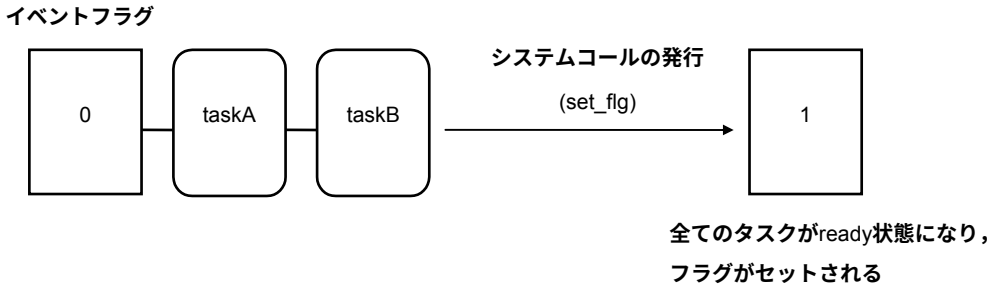


4.1.2　イベントフラグの操作

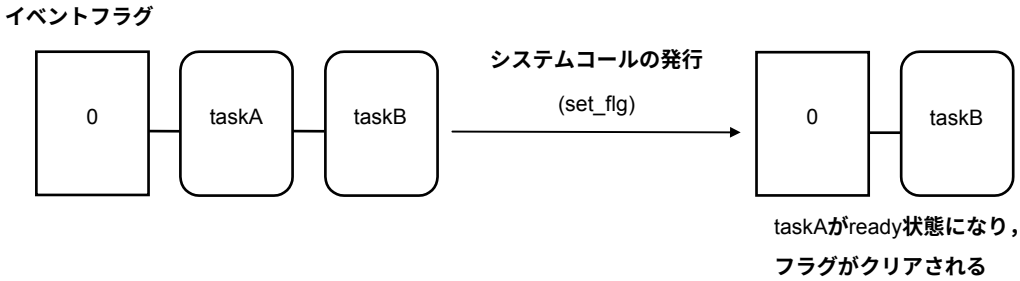
イベントフラグの操作には，イベントフラグのセット（set_flg, iset_flg）とクリア（clr_flg）の2種類があります。

（1）待ちタスクが存在しない場合

（2）wai_flgによる待ちタスクが存在する場合



（3）cwai_flgによる待ちタスク（taskA）が存在する場合



（4）イベントフラグのクリア

4.2　セマフォ

タスクの実行のために必要な各種の要素を資源とよびます。つまり資源とは，メモリ，入出力装置などのハードウエアや，ファイル，プログラムなどのソフトウエアなどすべてを指します。

1つの資源は，多くの場合，複数のタスクで同時に使用することはできません。したがって，並行に動作する複数のタスクが，限られた個数の資源を同時に使用してしまうような競合が起きるのを，何らかの方法で制御する必要があります。

このような資源の競合を防ぐ手段として，RX78K0ではセマフォを提供しています。セマフォでは，資源の個数を管理するカウンタを持ち，排他制御を行います。1回のシステムコールで操作できるカウント数は1です。

セマフォに対する操作は，システムコールによってタスクと割り込みハンドラから行われます。以下に，セマフォ関連システムコールを示します。

sig_sem	資源返却操作
isig_sem	〃　（割り込みハンドラで使用する）
wai_sem	資源獲得操作
preq_sem	セマフォ資源を得る

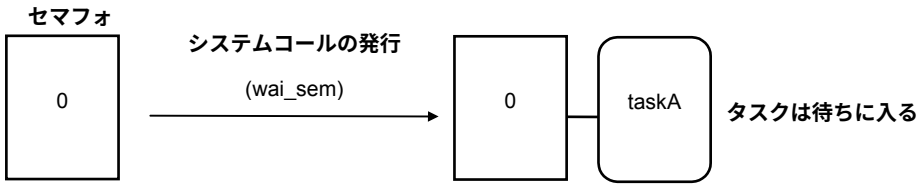
4.2.1　資源の獲得

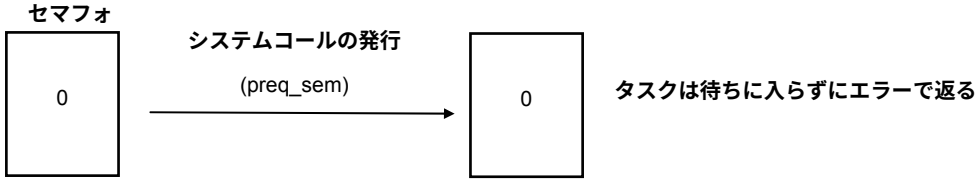
セマフォが管理している資源を使用したいタスクは，使用したいセマフォに対してwai_sem, preq_semシステムコールを発行します。

wai_semシステムコール発行時に，要求した資源が獲得できる場合にはタスクはwait状態にはなりませんが，要求した資源が獲得できない場合にはタスクはwait状態となり，対象セマフォの待ち行列につながります。

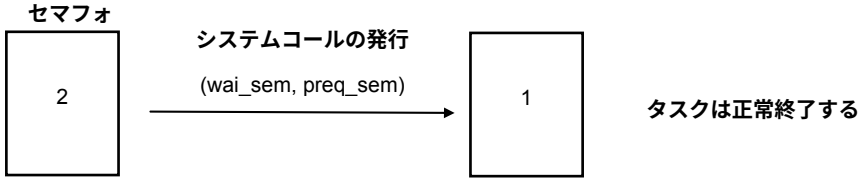
一方，preq_semシステムコールを発行したタスクは，要求した資源が獲得できなくてもwait状態にはならず，エラー・コードを持ってタスクに戻ります。

（1）資源が獲得できない場合





（2）資源が獲得できる場合

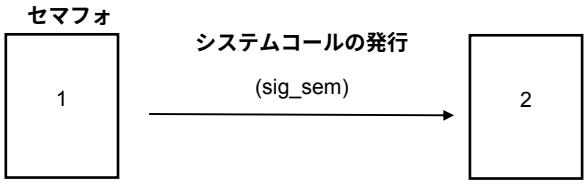


4.2.2　資源返却操作

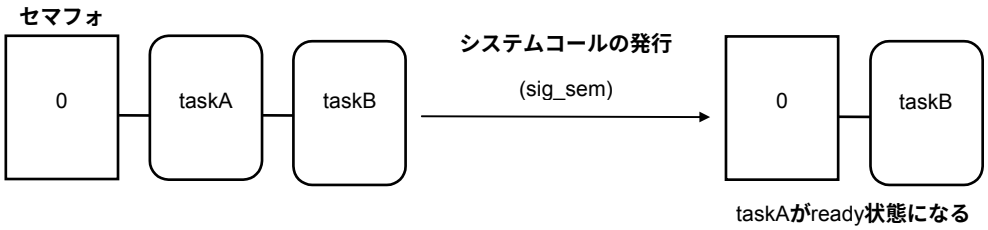
タスクで所有していた資源をセマフォに返却したい場合は，返却したいセマフォに対してsig_sem, isig_semシステムコールを発行します。

sig_sem, isig_semシステムコール発行時に待ちタスクが存在した場合は，待ち行列の先頭につながれているタスクをready状態にします。また，待ちタスクが存在しない場合は，資源数を1だけ増やします。

(1) 待ちタスクが存在しない場合



(2) 待ちタスクが存在する場合



4.3　メイルボックス

メイルボックスは，タスク間でデータの通信を行うための機構です。したがって，この機構においては，メッセージを投函する発信人となるタスク，メッセージを受け取る受信人となるタスク，およびメイルボックス，メッセージが必要となり，発信人と受信人とその間で受け渡されるメッセージとの同期をとるものです。このようにして，メッセージの受け渡しによってタスク間で通信を行うことができます。

メイルボックスに対する操作は，システムコールによってタスクと割り込みハンドラから行われます。メイルボックスは，おもに通信に用いられますが，ダミーのメッセージを用いて，単純な同期処理も行うことができます。以下に，メイルボックス関連システムコールを示します。

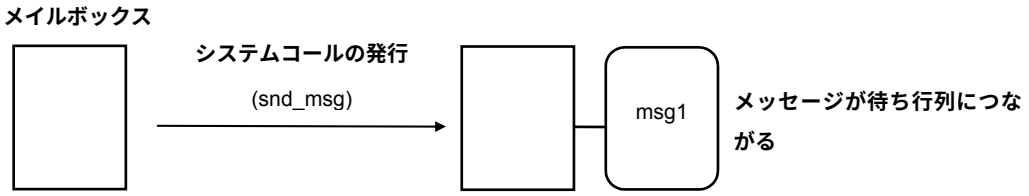
snd_msg	メイルボックスにメッセージを送信する
isnd_msg	〃　（割り込みハンドラで使用する）
rcv_msg	メイルボックスからのメッセージの受信を待つ
prcv_msg	メイルボックスのメッセージを受信する

4. 3. 1　メッセージの送信

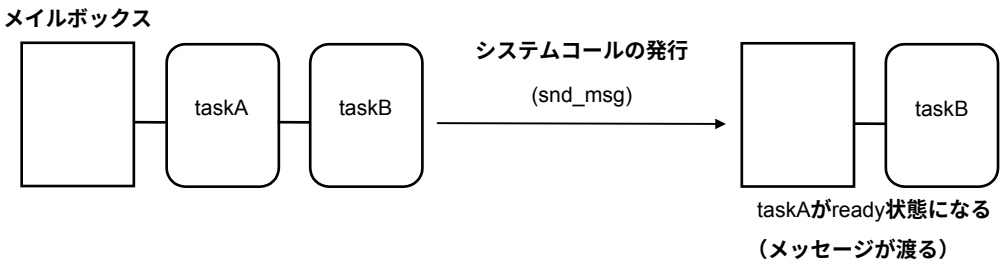
メッセージを送るタスクは，送信したいメッセージを作り，対象メイルボックスに対してsnd_msg, isnd_msg システムコールを発行し，メッセージを送信します。

メッセージが送信されたメイルボックスにおいて，メッセージを受信するためにタスクが待っていた場合，そのタスクはready状態になり，メッセージはそのままタスクに渡されます。しかし，メッセージを受信するタスクがない場合には，送信されたメッセージは対象メイルボックスにおける待ち行列につながり受信されるのを待ちます。

（1）待ちタスクが存在しない場合



（2）待ちタスクが存在する場合



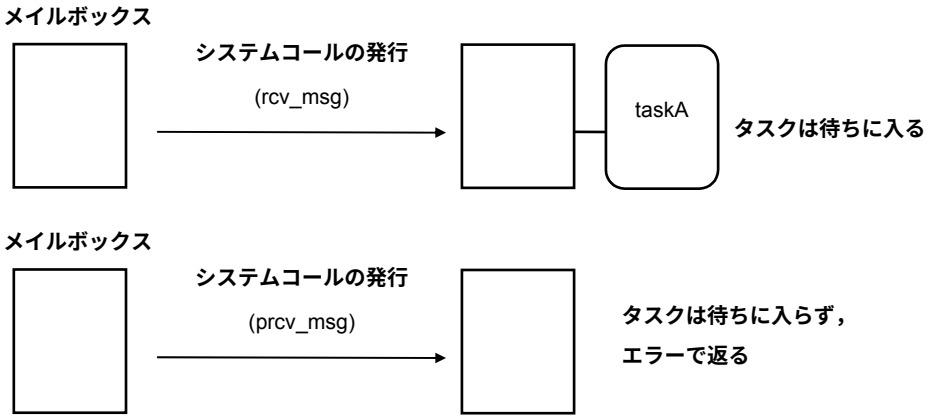
4. 3. 2　メッセージの受信

メッセージを受け取るタスクは，rcv_msg, prcv_msgシステムコールを発行して，対象メイルボックスにメッセージを受け取りにいきます。対象メイルボックスにおいて，メッセージを受信することができた場合には，タスクは実行を続けることができます。

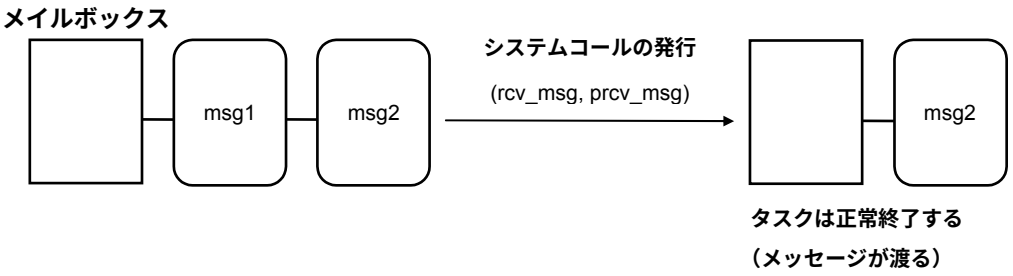
rcv_msgシステムコール発行時に，対象メイルボックスにメッセージが送信されていないため，メッセージを受信することができなかった場合，タスクは実行を続けることができなくなります。したがって，このタスクは対象メイルボックスにおける待ち行列につながれ，wait状態に入ります。

一方，prcv_msgシステムコールを発行したタスクは，対象メイルボックスからメッセージを受信できなくてもwait状態には入らずに，エラー・コードを持ってタスクに戻ります。

(1) メッセージが受信できない場合



(2) メッセージが受信できる場合



4.3.3　メッセージ

メイルボックスを介してタスク間でやり取りされるデータのことを，メッセージとよびます。

メッセージは，メイルボックスを介して任意のタスクに送ることができます。

ただし，メッセージの引き渡しは，メッセージのアドレスが渡されるだけであり，メッセージの内容がどこかにコピーされるわけではありません。

(1) メッセージ領域の作り方

メッセージとして送信する領域は，pget_blkシステムコールを発行して，任意のメモリプールからメモリブロックを切り出して使用します。

なお，メッセージの先頭1バイトは，メイルボックスへつながれる際のリンク・エリアとして使用するため，メッセージの書き込みは2バイト目以降に行ってください。

(2) メッセージの内容

RX78K0では，送信するメッセージの内容や長さについての規定はしていません。したがって，タスク間での取り決めにより，自由にメッセージを作ることができます。通信し合うタスク間で，メッセージの内容や長さを取り決めておいてください。

第5章　メモリ管理

RX78K0では，ダイナミックなメモリ管理を行っています。すなわち，メモリを必要時に確保でき，不要になったメモリ領域はその時点で返却することができるという機能が用意されています。この機能により，メモリを効率よく使用することができます。

以下に，RX78K0におけるメモリ管理について示します。

5.1　概　　要

RX78K0におけるメモリ管理とは，コンフィギュレータによって指定されたメモリプール領域を，システム初期化時に静的に生成し，管理，利用することをいいます。

5.2　メモリプール

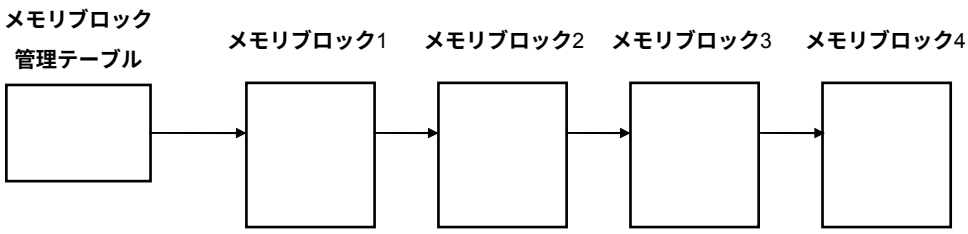
メモリプールは，RAM領域に存在し，タスクの要求に応えてメモリブロックを切り出し，解放するためのものです。メモリプールは，コンフィギュレータによって指定されたメモリブロックのサイズと数によって，システム初期化時に静的に生成されます。したがって，同一のメモリプールから切り出されるメモリブロックのサイズは，コンフィギュレータによって指定された固定サイズになります。

メモリプールは複数生成することができ，各メモリプールは独立しています。したがって，異なるメモリプールを複数使用することによって，メモリ領域の競合を避けることが可能です。

5. 3　メモリブロック

各タスクで使用するワーク領域としては，メモリプールから切り出すメモリブロックとよばれる領域を使用します。このメモリブロックは，必要時に獲得／解放を繰り返すことができるようになっています。

以下に，メモリプール領域について示します。



1つのメモリプールから切り出せるメモリブロックのサイズは，メモリプール生成時に指定された値で固定となっています。しかし，複数のメモリプールを，異なったブロックサイズで生成することにより，メモリプールの数と同じ数の，異なったサイズのメモリブロックを使用することが可能です。

5. 3. 1　メモリブロックの獲得

メモリブロックの獲得には，pget_blkシステムコールを発行します。どのメモリプールから切り出すのかは，パラメータによって指定します。

メモリブロックを要求した時に，対象メモリプールにメモリブロックが存在する場合は，先頭につながれているメモリブロックがはずされ，要求タスクに渡されます。要求した時にメモリブロックが存在しなかった場合は，要求タスクにエラー・コードが渡されます。

5. 3. 2　メモリブロックの解放

メモリブロックの解放には，rel_blkシステムコールを発行します。どのメモリプールに返すかは，ユーザがパラメータによって指定します。ただし，この時指定するメモリプールは，返却しようとするメモリブロックを獲得したメモリプールにしてください。

また，メッセージとしてメイルボックスにつながれているメモリブロックを，返却しようとする場合の動作については保証しません。

第6章　時間管理

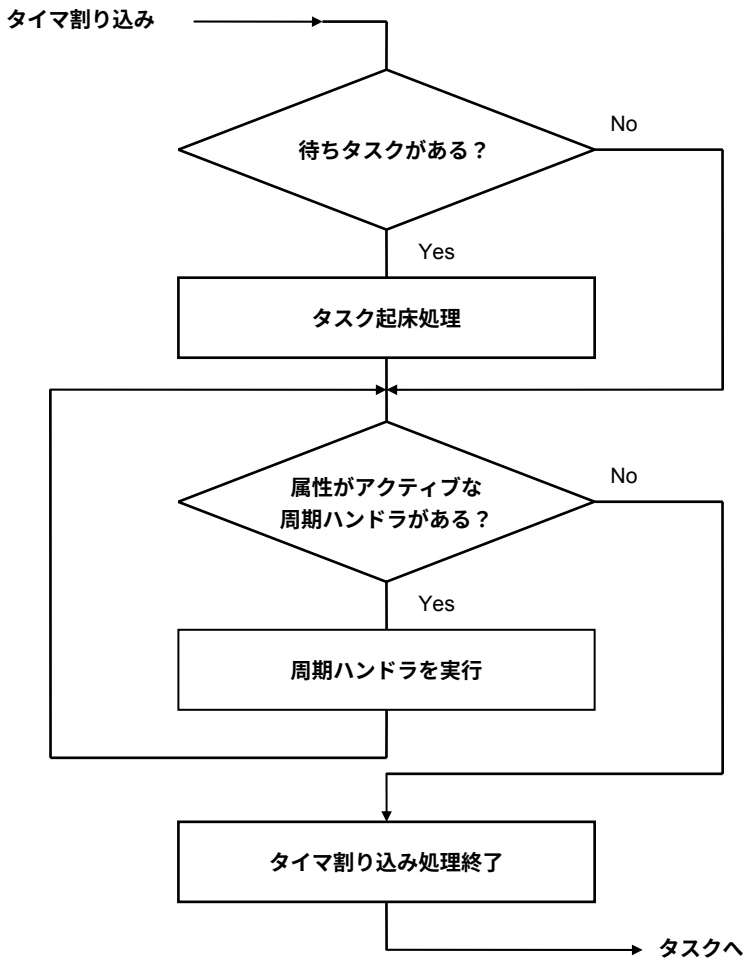
リアルタイム・システムにおいて，時間はタスクを操作する上で基本となる要素です。ニュークリアスにおける遅延起床，周期起床，タイムアウトなどの操作は，この時間を基準に行われます。基準時間は，ハードウエアによって外部で生成され，タイマ割り込みとして実現されています。

6.1　タイマ割り込み

タイマ割り込みは，ニュークリアスが時間管理を行う上で基本タイミングとなる割り込み信号です。タイマ割り込みは，CPUのタイマ／イベント・カウンタをインターバル・タイマとして使用することで，一定時間ごとに発生します。

タイマ割り込みが発生すると，ニュークリアス内の時間管理用の割り込みハンドラが起動します。図6－1に，このハンドラの処理を示します。

図6－1　タイマ割り込み処理フロー

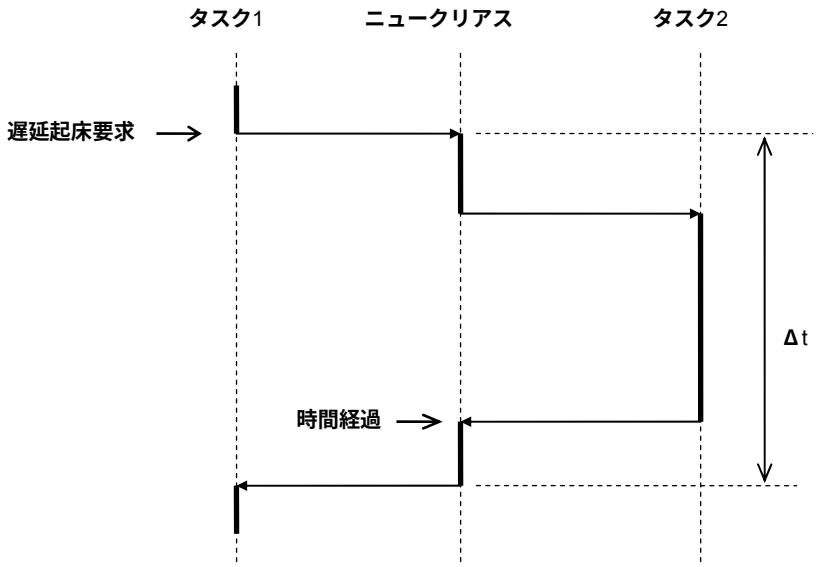


6. 2　遅延起床処理

遅延起床は，指定された時間が経過した後にタスクを起床する処理です。この機能は，wai_tskシステムコールを発行することで実現します。

wai_tskシステムコールをタスクが発行すると，その時点でタスクはwait状態に入ります。

wait状態に遷移した時点から指定時間（Δt）が経過すると，そのタスクは起床されready状態に遷移します。この時，起床したタスクには，時間経過による起床というリターン・パラメータが返されます。



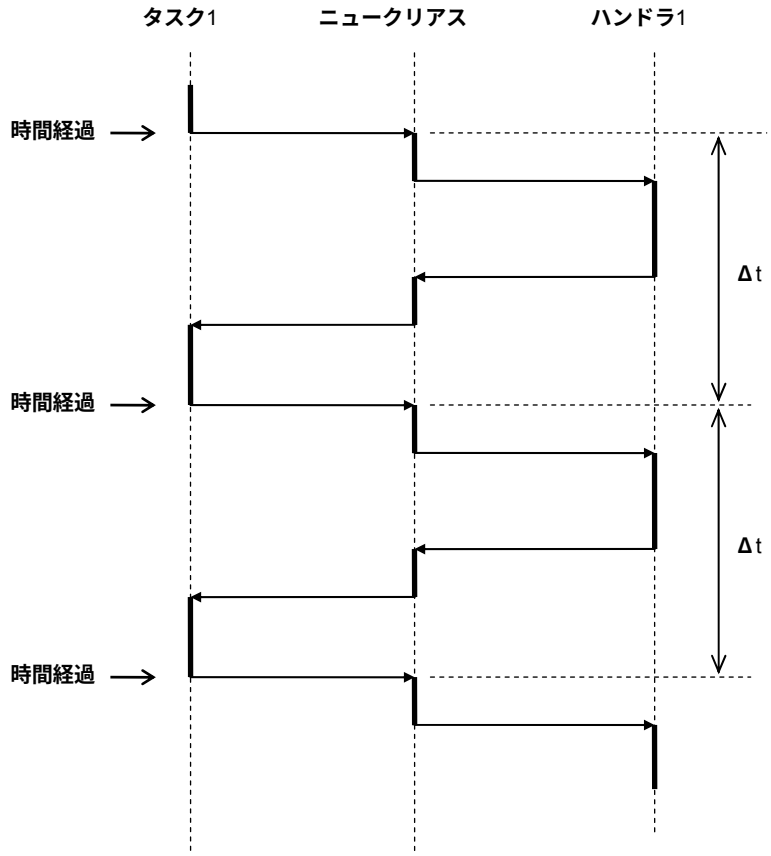
タスク1：　遅延起床タスク，ready状態
タスク2：　無関係タスク，ready状態
優先順位：　タスク1＞タスク2

タスクが遅延起床要求を行いwait状態の時に，他タスクから起床要求のwup_tskシステムコールを受けた場合には，タスクはその時点でready状態に遷移し，正常終了というリターン・パラメータが返されます。

6.3　周期起床

周期起床とは，割り込みハンドラを一定の時間間隔で起床させることをいい，この割り込みハンドラのことを，周期ハンドラとよびます。

周期ハンドラは，コンフィギュレーション時に複数定義することが可能で，この情報をもとにシステム初期化時に生成されます。そして，指定時間が経過すると，ニュークリアス内の時間管理用の割り込みハンドラによって起床されます。



タスク1　：　無関係タスク，ready状態
ハンドラ1:　周期ハンドラ

周期ハンドラは，その状態をアクティブ（起床される）／非アクティブ（起床されない）に切り替えることが可能です。この状態は，act_cycシステムコールの発行によって変更できます。

第7章　割り込み処理

割り込み処理は，外部に対してリアルタイムな反応をするための最も重要な手段です。一般に割り込み処理は，割り込みが発生すると，発生した割り込みに対応した割り込みハンドラに制御が移ります。この処理ルーチンに制御を移すのはハードウエアが行うため，ここにおいてRX78K0はまったく介入しません。

RX78K0が提供する割り込み関係システムコールは，次のとおりです。

ret_int	割り込みハンドラからの復帰
ret_wup	割り込みハンドラからの復帰とタスクの起床

7. 1　割り込み処理

割り込みが発生し，割り込みが受け付けられると，現在の割り込み処理が終了するか，または割り込み処理内で割り込み許可状態にしない限り，割り込みは受け付けられません。RX78K0管理下のシステムにおいては，システムコールを発行する割り込みハンドラ同士の多重割り込みをサポートしていません。したがって，割り込み処理が長ければ長いほど，システムの割り込みに対する応答性が悪くなりますので，割り込み処理はできるだけ短くしてください。

RX78K0では，割り込みに対する応答性を高めるために，割り込み処理を2種類の処理に切り分けて実行することを推奨しています。

(1) 割り込みハンドラ

簡単で短時間に処理が終わり，かつ高速応答が必要とされる部分には，割り込みハンドラとよばれる処理ルーチンを使用するようにします。

(2) 割り込みタスク

複雑で長時間かかり比較的応答速度の制限が緩やかな部分には，割り込みハンドラから起動される割り込みタスクを使用するようにします。

7.2　割り込みハンドラ

割り込みが発生した際に，ただちに起動される割り込み処理ルーチンを割り込みハンドラとよびます。割り込みハンドラは，タスクとは独立したものとして扱われます。

7.2.1　割り込みハンドラ内での処理

割り込みハンドラは，ニュークリアスを経由せずに，割り込み発生時に直接起動されます。したがって，その処理内容には次のような制限，手続きが存在します。

(1) レジスタの退避と復帰

レジスタの退避と復帰は，ユーザが割り込みハンドラ内で行わなければなりません。これは，割り込みハンドラがニュークリアスを経由せずに直接起動されるためです。

(2) システムコール発行の制限

割り込みハンドラ内で使用できるシステムコールは，次のとおりです。

iwup_tsk, iset_flg, isig_sem, isnd_msg, ret_int, ret_wup, iact_cyc

ただし，システムコールが発行可能な割り込み要求タイプは，マスカブル割り込みのみであり，他のソフトウエア割り込み，オペランド・エラー割り込み，およびノンマスカブル割り込みからはシステムコールを発行することはできません。また，割り込み処理モードも，ベクタ割り込み処理からのみシステムコールの発行が可能であり，マクロサービスおよび，コンテキスト・スイッチングからのシステムコール発行はできません。

(3) 割り込みハンドラの終了

割り込みハンドラから復帰するには，割り込みハンドラからの復帰を示すret_intシステムコール，または，復帰と共にタスクを起床させるret_wupシステムコールのいずれかを発行します。

ただし，割り込みハンドラ内でシステムコールを発行しない場合は，reti命令で終了してもかまいません。

7.3　多重割り込み

多重割り込みとは，割り込みハンドラ中でさらに割り込みが発生することです。RX78K0では，この多重割り込みに関してサポートしていませんので注意してください。また，時間管理用の割り込みハンドラに，通常の割り込みハンドラを割り込ませることもできません。

7.4　割り込みとスケジューリング

RX78K0では，割り込みハンドラ中にシステムコールが発行された場合，そのシステムコールでは待ち行列の操作などが行われるだけであり，実際の実行タスクの切り替えは，割り込みハンドラから戻るシステムコールで一括して行っています。

したがって，割り込みハンドラの中で発行されたシステムコールにより，優先度の高いタスクがready状態となっても，必ず割り込みハンドラの後処理が優先されます。

第8章　スケジューラ

ここでは，スケジューラについての詳細な説明を行います。

8.1　スケジューラとは

リアルタイム・システムでは，非同期に発生する事象に応答して動作を行うため，多くの事象に対応した処理が並行に実行されなければなりません。つまり，複数のタスクが，関係しながら順次に実行されるというマルチタスク処理ができることが基本となります。

スケジューラとは，タスクの実行順序を管理決定し，タスクにCPUの使用権を与えることを仕事としている部分です。

8.2　優先度方式のスケジューリング

制御用のリアルタイムOSでは，事象の重要度に応じて処理順序の決定を行う「優先度方式」によるスケジューリングの機能が要求され，RX78K0のスケジューラも優先度方式を採用しています。

つまり，RX78K0のスケジューラは，スケジューラが呼び出されたときに各タスクにあらかじめ付けられている優先度を参照し，その時点でready状態のタスクのうちから一番高い優先度を持つタスクを選び出し，そのタスクにCPUの使用権を与えます。

RX78K0では，タスクを選択する基準は，次の2つです。

(1) 優先度

各タスクは生成時に優先度が与えられています。優先度は，そのタスクが実行する順位を指定するものです。したがって，あらかじめ付けられているプライオリティを参照し，ready状態のタスクのうち最も優先度の高いタスクにCPUの使用権を与えます。

タスクごとに指定できる優先度は1から4までの4レベルの優先度で，優先度1が最高優先度になります。

(2) ready状態になった順序

RX78K0では，複数のタスクが同一の優先度を持つことが可能であるため，最も高い優先度のタスクが複数存在する場合があります。この場合，先にready状態になったタスクが選択されます（FIFO）。

8.3　スケジューラの駆動方式

RX78K0のスケジューラは，何らかの事象（イベント）が発生したときに駆動される事象駆動（イベント・ドリブン）方式を採用しています。そして，RX78K0においての「何らかの事象」とは，システムコールの発行を意味します。

したがって，RX78K0のスケジューラは，システムコールが発行されるごとに呼び出されます。ただし，割り込み処理中はこの限りではありません。

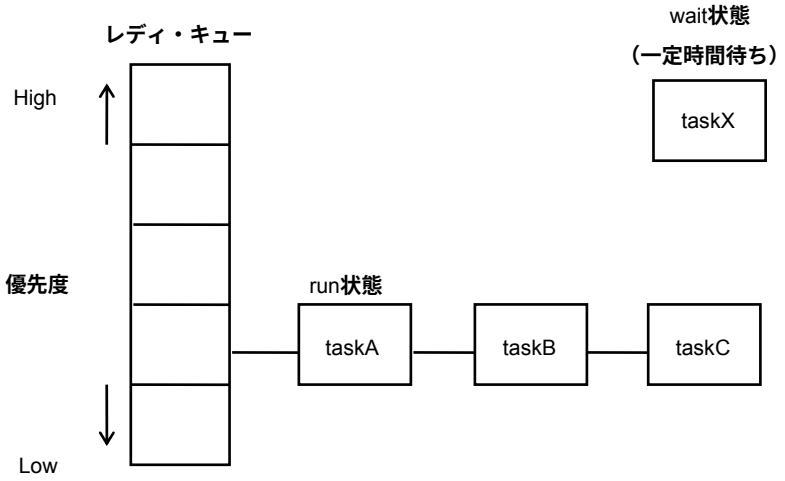
8.4　ラウンドロビンの実現

RX78K0では，優先度方式のスケジューリングを行っていますが，この方式のもとでは各タスクに付けられた優先度順に実行されるため，高い優先度を持つタスクが優先的に実行されます。したがって，実行されているタスクよりも低い優先度を持つタスクは，一度も実行されない場合があります。

これは，実行タスクと同一優先度を持つタスクにも同じことがいえます。つまり，システム中，最高優先度を持つタスクであっても，同一優先度のタスクが他にもあり，そのタスクが先に実行権を握っている（run状態にある）限り，実行タスクと同一優先度を持つタスクは実行されることがありません。

このような状態を避けるために，ある一定の周期をもってタスクを切り替え，どのタスクも同様に動作させるというスケジューリング方式があります。このスケジューリング方式をラウンドロビン方式と呼び，RX78K0ではrot_rdqシステムコールで実現できます。

以下に，ラウンドロビン方式の例を示します。



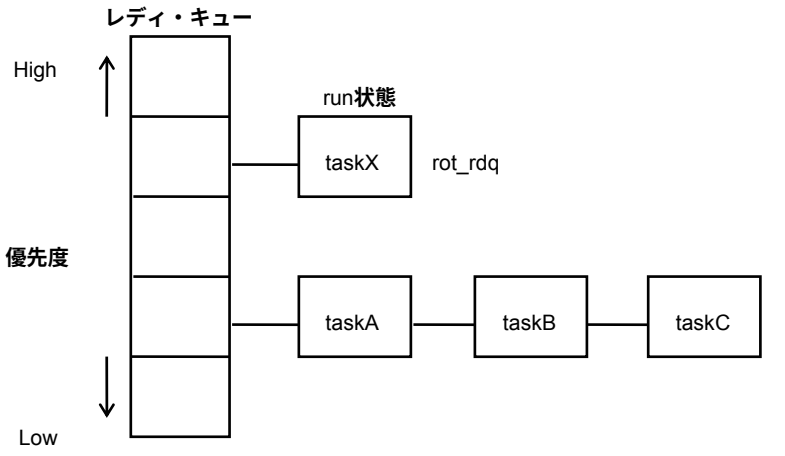
タスクの優先度は

taskX＞taskA = taskB = taskC

で，taskAはrun状態，taskBとtaskCはready状態，taskXはwait状態です。

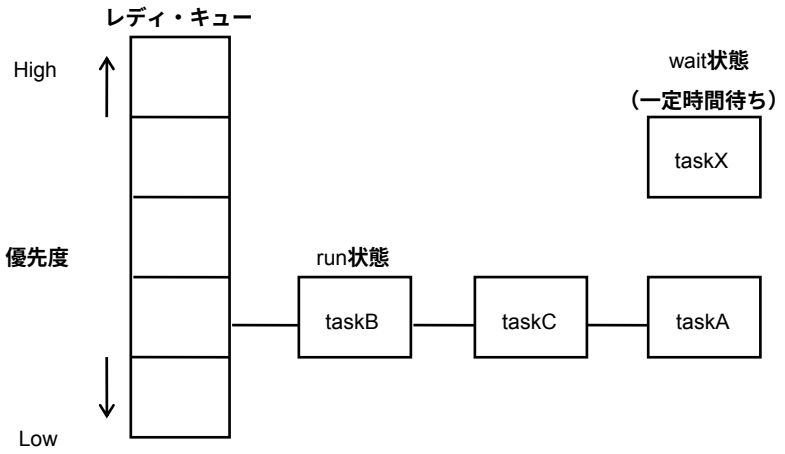
taskXは一定時間が経過すると起床し，タスクの切り替えを行った後，再びwait状態に移行します。

現在，最高優先度を持つtaskXがwait状態であるため，taskAが実行されています。他のタスクはtaskAと同一優先度のため，taskAが実行を中断しない限りrun状態にはなれません。



taskXは，指定時間経過後に起床しrun状態になると，その処理の中でrot_rdqシステムコールを発行して ，taskAが持つ優先度のレディキューを回転させます。

その結果，次のようにレディキューの状態が変化します。これにより，一定時間ごとに実行タスクを切り替えることができます。



第9章　システムコール

ユーザ・タスクがニュークリアスにサービスを要求する場合には，システムコールを発行します。RX78K0では，タスク制御関係，同期通信関係などのシステムコールが用意されています。

9.1　概　　要

システムコールとは，タスクがニュークリアスに対してサービスを要求する手続き，またはその機能のことです。システムコールによりタスクは，通信，割り込み制御などのニュークリアス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になります。

以下に，システムコールの機能概要について示します。

9.1.1　機能概要

システムコールは全部で33個用意されており，その機能によって次の6つのグループに分類されます。

(1) タスク制御システムコール（11個）

タスク制御システムコールは，タスクの起動，起床（遅延）などの操作を行うシステムコールのグループです。このグループに属するシステムコールは次のとおりです。

```
sta_tsk,   ext_tsk,  ter_tsk,   chg_pri,  rot_rdq,   tsk_sts,
slp_tsk,   wai_tsk,  wup_tsk,   iwup_tsk, can_wup
```

(2) 同期通信システムコール（15個）

同期通信システムコールは，タスク間の同期，排他，通信制御を行うために，イベントフラグ，セマフォ，メイルボックスの制御を行うシステムコールのグループです。このグループに属するシステムコールは次のとおりです。

```
set_flg,   iset_flg, clr_flg,   wai_flg,  cwai_flg,  pol_flg,
cpol_flg,  sig_sem,  isig_sem,  wai_sem,  preq_sem,  snd_msg,
isnd_msg,  rcv_msg,  prcv_msg
```

(3) 割り込み制御システムコール（2個）

割り込み制御システムコールは，割り込みからの復帰を行うシステムコールのグループです。このグループに属するシステムコールは次のとおりです。

```
ret_int,   ret_wup
```

(4) メモリ管理システムコール（2個）

メモリ管理システムコールは，メモリブロックの獲得と返却を行うシステムコールのグループです。このグループに属するシステムコールは次のとおりです。

```
pget_blk,       rel_blk
```

(5) バージョン管理システムコール（1個）

バージョン管理システムコールは，RX78K0のバージョン情報を得るシステムコールです。このグループに属するシステムコールは次のとおりです。

```
get_ver
```

(6) 時間管理システムコール（2個）

時間管理システムコールは，周期ハンドラの状態変更を行うシステムコールのグループです。このグループに属するシステムコールは次のとおりです。

```
act_cyc,        iact_cyc
```

9.1.2　システムコールのエントリ

RX78K0では，システムコールの呼び出しにcallt命令を使用しています。callt命令とは，CPUが提供しているサブルーチンコール命令の1つで，calltテーブルを参照して目的のルーチンに分岐する1バイト命令です。

calltテーブル領域は，40H番地から7FH番地まで利用でき，32個のサブルーチン・エントリを設定できます。RX78K0では，本領域を以下のように利用することを前提としています。

```
40H  ------  共通分岐処理モジュール
42H  ------  iwup_tsk
44H  ------  iset_flg
46H  ------  isig_sem
48H  ------  isnd_msg
4AH  ------  iact_cyc
```

共通分岐処理モジュールとは，タスクから発行されるシステムコールに分岐するためのモジュールです。すなわち，タスクがシステムコールを発行する際は，必ず本モジュールを介してから各システムコールに分岐します。そのため，各システムコールにはID番号が割り当てられ，ユーザが各システムコールを発行する際に，該当するID番号を分岐処理モジュールに渡します。分岐処理モジュールは，ID番号とシステムコール・エントリ・テーブルを参照して，指定システムコールへ分岐します。システムコール・エントリ・テーブルは，コンフィギュレータによって生成されます。また，ID番号および各システムコールごとのパラメータについては，ユーザ・タスクのスタックを使用して，分岐処理モジュールおよび各システムコールに渡されます。ID番号および1バイトのパラメータをスタックに積む際は，メモリのLOW側に配置されるように積んでください。

割り込みハンドラから発行するシステムコールは，calltテーブルを介して直接システムコールに分岐します。各システムコールのパラメータは，システムコール呼び出し時のレジスタを使用します。

9. 2　システムコール・エラー・コード一覧

以下に，システムコールのエラー・コードと対応するエラーについて示します。

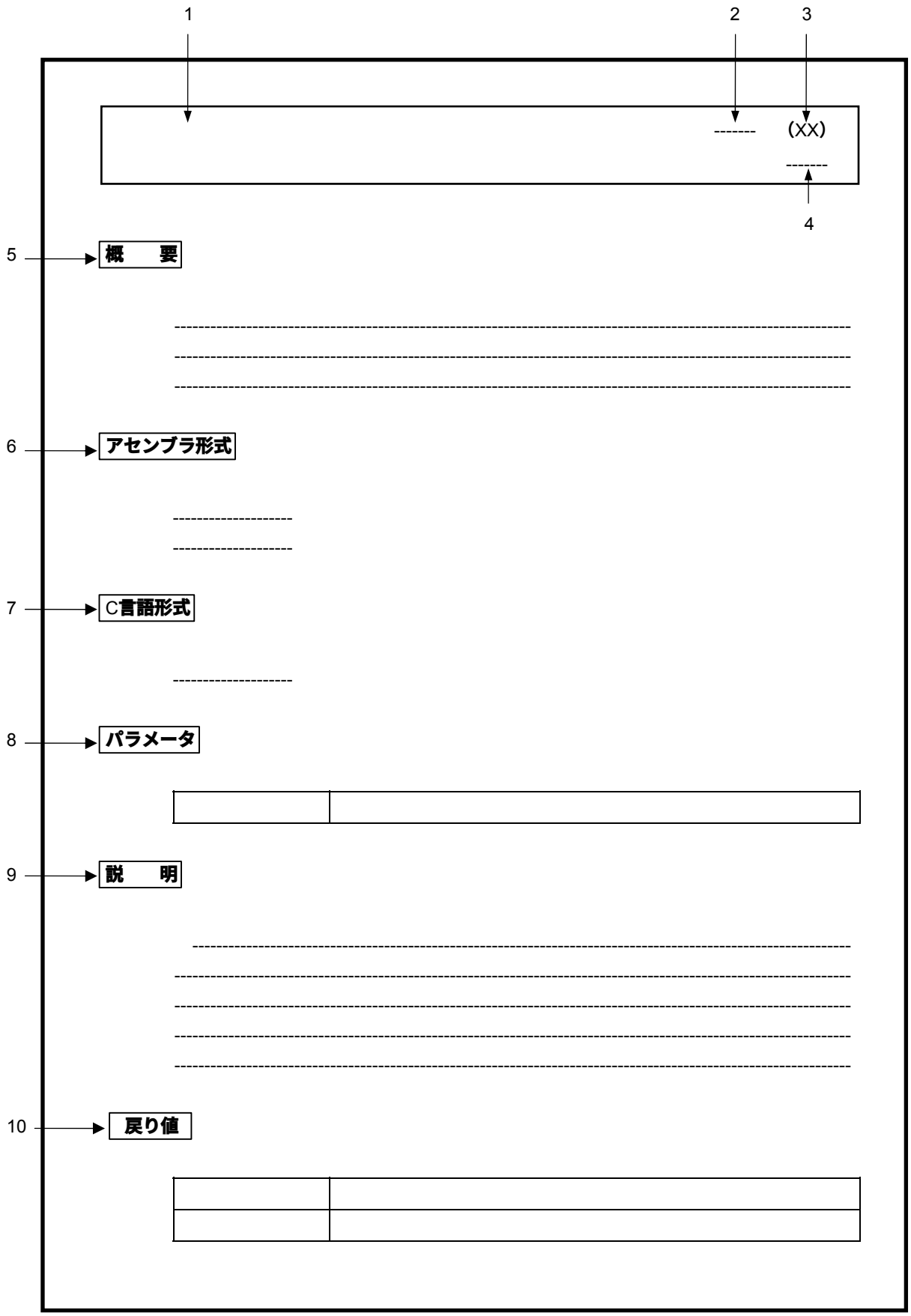
エラー・コード		エラー発生要因
	値	
E_OK	0x0	正常終了
E_NODMT	0x1	指定されたタスクがdormant状態ではない
E_DMT	0x2	指定されたタスクがdormant状態である
E_QOVR	0x3	カウントがオーバ・フローした
E_TMOUT	0x4	指定された時間が経過した
E_PLFAIL	0x5	ポーリング失敗

9. 3　システムコール仕様

以下に，各システムコールのパラメータ，機能などの仕様の詳細について示します。

9. 3. 1　仕様概要

次の項から，システムコールの仕様の詳細について示しています。これらの仕様は，次のようなフォーマットで記述してあります。


1
2
3

(XX)

4
5
概　　要
6
アセンブラ形式
7
C言語形式
8
パラメータ
9
説　　明
10
戻り値

1　システムコールの名称

2　システムコールの正式名称

　　システムコールの名称として使用されている文字は大文字に，それ以外の文字は小文字で示しています。

3　ID番号

　　ニュークリアスがシステムコールを識別する際の番号を示しています。

4　システムコールを発行できる状態を示しています。

　　タスクのみ　　：　タスクからのみ発行できるシステムコールです。
　　割り込みハンドラのみ：　割り込みハンドラからのみ発行できるシステムコールです。

5　概　　要

　　システムコールの機能概要を述べています。

6　アセンブラ形式

　　アセンブラでシステムコール発行を記述する際の形式を示しています。

7　C言語形式

　　C言語でシステムコール発行を記述する際の形式を示しています。

8　パラメータ

　　パラメータは，次のような形式の表で示されています，

パラメータの型	パラメータの説明

9　説　　明

　　システムコールの処理内容について説明してあります。

10　戻り値

　　システムコールの戻り値を示しています。アセンブラからシステムコールを発行した場合は，Cレジスタに戻り値が返されます。

戻り値	戻り値の説明

9. 3. 2　タスク制御システムコール

この項では，次のシステムコールの仕様について示します。

ID番号	システムコール名	機　　能
0	sta_tsk	タスクを起動する
1	ext_tsk	自タスクを正常終了する
2	ter_tsk	他タスクを強制的に異常終了させる
3	chg_pri	タスクの優先度を変更する
4	rot_rdq	タスクのレディ・キューを回転する
5	tsk_sts	タスクの状態を獲得する
6	slp_tsk	自タスクをwait状態へ移行する
7	wai_tsk	自タスクを時限付きでwait状態へ移行する
8	wup_tsk	wait状態のタスクを起床させる
−	iwup_tsk	wait状態のタスクを起床させる（割り込みハンドラ用）
9	can_wup	起床要求を無効にする

STArt TaSK (0)

sta_tsk

タスクのみ

概　　要

タスクを起動する。

アセンブラ形式

```
movw      ax, #tskid
push      ax
mov       x, #0
push      ax
callt     [40h]
pop       ax
pop       ax
```

C言語形式

```
char      sta_tsk(tskid);
```

パラメータ

char　*tskid	**起動するタスクのアクセス・アドレス**

説　　明

tskidで示されたタスクを起動します。つまり，指定されたタスクをdormant状態からready状態へ移行させます。対象タスクがdormant状態でない場合，起動要求は無視され，発行タスクにはエラー・コード“E_NODMT”が返されます。

タスクが初めて起動する場合は，割り込みが許可状態となっていますので，タスクごとに適切な箇所で割り込み禁止にしてください。

戻　り　値

E_OK	**正常終了**
E_NODMT	**対象タスクがdormant状態でない**

EXiT TaSK (1)

ext_tsk

タスクのみ

概　　要

自タスクを正常終了する。

アセンブラ形式

```
mov       x, #1
push      ax
callt     [40h]
```

C言語形式

```
void      ext_tsk();
```

パラメータ

なし

説　　明

自タスクを正常終了し，dormant状態へ移行させます。該当するTCBおよびコンテキストは，生成されたときの状態に初期化します。

注意　ext_tskシステムコールではタスクがそれ以前に獲得した資源（セマフォ，メモリブロックなど）を，自動的には解放しませんので，タスク終了前に獲得資源を解放してください。

戻 り 値

なし

TERminate TaSK (2)

ter_tsk

タスクのみ

概　　要

他タスクを強制的に異常終了させる。

アセンブラ形式

```
movw      ax, #tskid
push      ax
mov       x, #2
push      ax
callt     [40h]
pop       ax
pop       ax
```

C言語形式

```
char      ter_tsk(tskid);
```

パラメータ

char　*tskid	**異常終了させるタスクのアクセス・アドレス**

説　　明

tskid **で示されたタスクを強制的に終了（異常終了）させ，**dormant**状態へと移行させます。該当する**TCB**およびコンテキストは，生成されたときの状態に初期化します。**

注意　待ち状態のタスクに対して，本システムコールが発行された場合，対象タスクはその待ち行列から削除されますが，タスクが獲得していた資源（セマフォ，メモリブロックなど）の解放は行いません。

戻　り　値

E_OK	**正常終了**
E_DMT	**対象タスクが**dormant**状態である**

CHanGe task PRIority (3)

chg_pri

タスクのみ

概　　要

タスクの優先度を変更する。

アセンブラ形式

```
movw      ax, #tskid
push      ax
mov       x, #tskpri
push      ax
mov       x, #3
push      ax
callt     [40h]
pop       ax
pop       ax
pop       ax
```

C言語形式

```
char      chg_pri(tskid, tskpri);
```

パラメータ

char	*tskid	**優先度を変更するタスクのアクセス・アドレス**
char	tskpri	**変更後の優先度**

説　　明

tskidで示されたタスクの優先度を，tskpriで示される値に変更します。なお，タスクの優先度は数の小さい方が高い優先度となっており，1から4までの値を指定できます。また，tskidに“TSK_SELF”（0x0）を指定した場合は自タスクの指定となります。

本システムコールで変更した優先度は，タスクが終了するまで有効となります。タスクがdormant状態になると，終了前のタスク優先度は捨てられ，次にタスクが起動したときの優先度は，システム起動時に指定された初期優先度となります。

戻　り　値

E_OK	**正常終了**
E_DMT	**対象タスクがdormant状態である**

ROTate ReaDy Queue (4)

rot_rdq

タスクのみ

概　　要

タスクのレディキューを回転する。

アセンブラ形式

```
mov       x, #tskpri
push      ax
mov       x, #4
push      ax
callt     [40h]
pop       ax
pop       ax
```

C言語形式

```
char      rot_rdq(tskpri);
```

パラメータ

char　tskpri	回転させるレディ・キューの優先度

説　　明

tskpriで指定した優先度のレディ・キューの先頭につながれているタスクを，レディ・キューの最後尾につなぎかえ，指定優先度内でのタスクの実行順序を切り替えます。

なお，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は何も行いませんが，エラーとしても扱われません。

また，tskpriに0x0を指定した場合，自タスクの持つ優先度を回転することになり，自タスクがレディ・キューの最後尾に付き実行タスクが切り替わります。つまり，自タスクはrun状態からready状態へと遷移します。

戻　り　値

E_OK	正常終了

get TaSK STatuS (5)

tsk_sts

タスクのみ

概　　要

タスクの情報を獲得する。

アセンブラ形式

```
movw      ax, #p_tsksts
push      ax
movw      ax, #tskid
push      ax
mov       x, #5
push      ax
callt     [40h]
pop       ax
pop       ax
pop       ax
```

C言語形式

```
char      tsk_sts(p_tsksts, tskid);
```

パラメータ

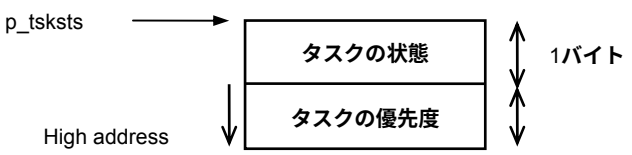
short	*p_tsksts	獲得したタスクの情報を格納するパケットのアドレス
char	*tskid	情報を獲得するタスクのアクセス・アドレス

説　　明

tskidで指定したタスクの情報をp_tskstsで指定される領域に格納します。獲得する情報の内容は，タスクの優先度とタスクの状態です。また，tskidに“TSK_SELF”（0x0）を指定した場合は，自タスクの指定となります。

なお，p_tskstsで領域は，ユーザが確保してください。

以下に，タスクの情報の詳細を示します。



また，タスクの状態は，以下のようになっています。

0x0 ： dormant状態
0x1 ： runまたはready状態
0x2 ： wai_tsk発行によるwait状態
0x3 ： slp_tsk発行によるwait状態
0x4 ： wai_flg発行によるwait状態
0x5 ： cwai_flg発行によるwait状態
0x6 ： wai_sem発行によるwait状態
0x7 ： rcv_msg発行によるwait状態

戻　り　値

E_OK	正常終了

SLeeP TaSK (6)

slp_tsk

タスクのみ

概　　要

自タスクをwait状態へ移行する。

アセンブラ形式

```
mov       x, #6
push      ax
callt     [40h]
pop       ax
```

C言語形式

```
char      slp_tsk();
```

パラメータ

なし

説　　明

自タスクをrun状態からwait状態へ遷移させます。

このwait状態は，起床要求（wup_tsk, iwup_tskまたはret_wupシステムコールの発行）により解除され，ready状態へ遷移します。

戻 り 値

E_OK	正常終了

WAIt for wakeup TaSK (7)

wai_tsk

タスクのみ

概　　要

自タスクを時限付きでwait状態へ移行する。

アセンブラ形式

```
movw      ax, #tmout
push      ax
mov       x, #7
push      ax
callt     [40h]
pop       ax
pop       ax
```

C言語形式

```
char      wai_tsk(tmout);
```

パラメータ

unsigned short tmout	タイムアウト値

説　　明

自タスクを指定した時間だけwait状態へ遷移させます。

このwait状態は，起床要求（wup_tsk, iwup_tskまたはret_wupシステムコールの発行），または，指定した時間の経過により解除され，ready状態へ遷移します。

なお，起床要求により解除された場合，正常終了となり，エラー・コード“E_OK”が返されます。また，指定した時間の経過により解除された場合，タイムアウト・エラーとなり，エラー・コード“E_TMOUT”が返されます。

tmoutに0x0を指定した場合，wait状態には遷移しないで即時リターンとなり，エラー・コード“E_TMOUT”が返されます。

戻　り　値

E_OK	正常終了
E_TMOUT	指定された時間が経過した

Wake UP TaSK (8)

wup_tsk

タスクのみ

概　　要

wait状態のタスクを起床させる。

アセンブラ形式

```
movw      ax, #tskid
push      ax
mov       x, #8
push      ax
callt     [40h]
pop       ax
pop       ax
```

C言語形式

```
char      wup_tsk(tskid);
```

パラメータ

char　*tskid	起床させるタスクのアクセス・アドレス

説　　明

tskidで指定したタスクを起床させます。つまり，slp_tskまたはwai_tskシステムコールの発行により，wait状態のタスクをready状態へ遷移させます。

各タスクは，発行された起床要求の数を保持するカウンタを持ちます。したがって，wait状態でないタスクに対して，本システムコールを発行した場合でも，その起床要求は有効となり，発行された回数分だけカウンタの値が上がります。なお，カウンタの値が0x0でないタスクが，slp_tskまたはwai_tskシステムコールを発行した場合，カウンタの値が0x1減算されるだけで，対象タスクはwait状態には遷移しません。

タスクに対する起床要求は，1タスクで最高0xff回まで保持されます。なお，起床要求が0xff回を越えた場合，エラー・コード“E_QOVR”が返されます。

戻　り　値

E_OK	正常終了
E_DMT	対象タスクがdormant状態である
E_QOVR	起床要求が0xff回を越えた

Wake UP TaSK for Interrupt

iwup_tsk

割り込みハンドラのみ

概　　要

wait状態のタスクを起床させる。

アセンブラ形式

```
movw      hl, #tskid
callt     [42h]
```

C言語形式

```
char      iwup_tsk(tskid);
```

パラメータ

char　*tskid	起床させるタスクのアクセス・アドレス

説　　明

tskidで指定したタスクを起床させます。つまり，slp_tskまたはwai_tskシステムコールの発行により，wait状態のタスクをready状態へ遷移させます。

各タスクは，発行された起床要求の数を保持するカウンタを持ちます。つまり，wait状態でないタスクに対して，本システムコールを発行した場合でも，その起床要求は有効となり，発行された回数分だけカウンタの値が上がります。

なお，カウンタの値が0x0でないタスクが，slp_tskまたはwai_tskシステムコールを発行した場合，カウンタの値が0x1減算されるだけで，wait状態には遷移しません。

タスクに対する起床要求は，1タスクで最高0xff回まで保持されます。なお，起床要求が0xff回を越えた場合，エラー・コード“E_QOVR”が返されます。

戻　り　値

E_OK	正常終了
E_DMT	対象タスクがdormant状態である
E_QOVR	起床要求が0xff回を越えた

CANcel WakeUP task （9）

can_wup

タスクのみ

概　　要

起床要求を無効にする。

アセンブラ形式

```
movw      ax, #p_wupcnt
push      ax
movw      ax, #tskid
push      ax
mov       x, #9
push      ax
callt     [40h]
pop       ax
pop       ax
pop       ax
```

C言語形式

```
char      can_wup(p_wupcnt, tskid);
```

パラメータ

char	*p_wupcnt	無効にした起床要求回数を格納する領域のアドレス
char	*tskid	起床要求を無効にするタスクのアクセス・アドレス

説　　明

tskidで指定したタスクに発行されている起床要求を無効にします。

タスクに対する起床要求は，1タスクで最高0xff回まで保持されます。しかし，本システムコールの発行により，対象タスクに発行されているすべての起床要求が無効となります。

また，tskidに“TSK_SELF”（0x0）を指定した場合，自タスクが対象となります。

なお，p_wupcnt領域はユーザが確保してください。

戻　り　値

E_OK	正常終了
E_DMT	対象タスクがdormant状態である

9. 3. 3　同期通信システムコール

この項では，次のシステムコールの仕様について示します。

ID番号	システムコール名	機　　能
10	set_flg	イベントフラグをセットする
－	iset_flg	イベントフラグをセットする（割り込みハンドラ用）
11	clr_flg	イベントフラグをクリアする
12	wai_flg	イベントフラグがセットされるのを待つ（クリア無し）
13	cwai_flg	イベントフラグがセットされるのを待つ（クリア有り）
14	pol_flg	イベントフラグを得る（クリア無し）
15	cpol_flg	イベントフラグを得る（クリア有り）
16	sig_sem	セマフォに対する信号操作を行う
－	isig_sem	セマフォに対する信号操作を行う（割り込みハンドラ用）
17	wai_sem	セマフォに対する待ち操作を行う
18	preq_sem	セマフォ資源を得る
19	snd_msg	メッセージを送信する
－	isnd_msg	メッセージを送信する（割り込みハンドラ用）
20	rcv_msg	メッセージの受信を待つ
21	prcv_msg	メッセージを受信する

SET event FLaG (10)

set_flg

タスクのみ

概　　要

イベントフラグをセットする。

アセンブラ形式

```
movw      ax, #flgid
push      ax
mov       x, #10
push      ax
callt     [40h]
pop       ax
pop       ax
```

C言語形式

```
char      set_flg(flgid);
```

パラメータ

char　*flgid	セットするイベントフラグのアクセス・アドレス

説　　明

flgidで指定したイベントフラグに1をセットします。

イベントフラグがセットされるのを待っているタスクがある場合，待ち行列の先頭タスクをはずし，wait状態（イベントフラグ待ち）からready状態へ移行します。この処理を，待ち行列にタスクが無くなるか，またはクリア指定のあるタスクをready状態へ移行するまで，繰り返し続けます。

戻　り　値

E_OK	正常終了

SET event FLaG for Interrupt

iset_flg

割り込みハンドラのみ

概　　要

イベントフラグをセットする。

アセンブラ形式

```
movw      hl, #flgid
callt     [44h]
```

C言語形式

```
char      iset_flg(flgid);
```

パラメータ

char　*flgid	**セットするイベントフラグのアクセス・アドレス**

説　　明

flgidで指定したイベントフラグに1をセットします。

イベントフラグがセットされるのを待っているタスクがある場合，待ち行列の先頭タスクをはずし，wait状態（イベントフラグ待ち）からready状態へと遷移します。この処理を，待ち行列にタスクが無くなるか，またはクリア指定のあるタスクをready状態へ遷移するまで，繰り返し続けます。

戻　り　値

E_OK	**正常終了**

CLeaR event FLaG (11)

clr_flg

タスクのみ

概　　要

イベントフラグをクリアする。

アセンブラ形式

```
movw      ax, #flgid
push      ax
mov       x,#11
push      ax
callt     [40h]
pop       ax
pop       ax
```

C言語形式

```
char      clr_flg(flgid);
```

パラメータ

char　*flgid	**クリアするイベントフラグのアクセス・アドレス**

説　　明

flgidで指定したイベントフラグをクリアします。

すでにクリアされていても，エラーとはなりません。

戻 り 値

E_OK	**正常終了**

WAIt for event FLaG to be set (12)

wai_flg

タスクのみ

概　　要

イベントフラグがセットされるのを待つ（クリア無し）。

アセンブラ形式

```
movw      ax, #flgid
push      ax
mov       x,#12
push      ax
callt     [40h]
pop       ax
pop       ax
```

C言語形式

```
char      wai_flg(flgid);
```

パラメータ

char　*flgid	セットされるのを待つイベントフラグのアクセス・アドレス

説　　明

flgidで指定したイベントフラグがセットされるのを待ちます。つまり，ready状態からwait状態（イベントフラグ待ち）へ遷移させ，指定イベントフラグの待ち行列につなげます。

すでに指定イベントフラグがセットされている場合，wait状態（イベントフラグ待ち）には遷移しません。

戻　り　値

E_OK	正常終了

Clear and WAIt for event FLaG to be set (13)

cwai_flg

タスクのみ

概　　要

イベントフラグがセットされるのを待つ（クリア有り）。

アセンブラ形式

```
movw      ax, #flgid
push      ax
mov       x,#13
push      ax
callt     [40h]
pop       ax
pop       ax
```

C言語形式

```
char      cwai_flg(flgid);
```

パラメータ

char　*flgid	セットされるのを待つイベントフラグのアクセス・アドレス

説　　明

flgidで指定したイベントフラグがセットされるのを待ちます。つまり，ready状態からwait状態（イベントフラグ待ち）へ遷移させ，指定イベントフラグの待ち行列につなげます。イベントフラグがセットされて，このタスクが待ち解除となった場合に，イベントフラグがクリアされます。

すでに指定イベントフラグがセットされている場合，イベントフラグのクリアを行い，wait状態（イベントフラグ待ち）には遷移しません。

戻　り　値

E_OK	正常終了

POLI event FLaG (14)

pol_flg

タスクのみ

概　　要

イベントフラグを得る（クリア無し）。

アセンブラ形式

```
movw      ax, #flgid
push      ax
mov       x,#14
push      ax
callt     [40h]
pop       ax
pop       ax
```

C言語形式

```
char      pol_flg(flgid);
```

パラメータ

char　*flgid	**イベントフラグのアクセス・アドレス**

説　　明

flgidで指定したイベントフラグがセットされているかどうかを調べます。イベントフラグがセットされている場合は正常終了となり，エラー・コード“E_OK**”が返されます。また，イベントフラグがセットされていない場合はポーリング・エラーとなり，エラー・コード “**E_PLFAIL**”が返されます。**

戻　り　値

E_OK	**正常終了**
E_PLFAIL	**イベントフラグがセットされていない**

Clear and POLI event FLaG (15)

cpol_flg

タスクのみ

概　　要

イベントフラグを得る（クリア有り）。

アセンブラ形式

```
movw      ax, #flgid
push      ax
mov       x, #15
push      ax
callt     [40h]
pop       ax
pop       ax
```

C言語形式

```
char      cpol_flg(flgid);
```

パラメータ

char　*flgid	**イベントフラグのアクセス・アドレス**

説　　明

flgidで指定したイベントフラグがセットされているかどうかを調べます。イベントフラグがセットされている場合は正常終了となり，イベントフラグがクリアされます。この時のエラー・コードは“E_OK**”が返されます。また，イベントフラグがセットされていない場合はポーリング・エラーとなり，エラー・コード“**E_PLFAIL**”が返されます。**

戻　り　値

E_OK	**正常終了**
E_PLFAIL	**イベントフラグがセットされていない**

SIGnal SEMaphore (16)

sig_sem

タスクのみ

概　　要

セマフォに対する信号操作を行う。

アセンブラ形式

```
movw      ax, #semid
push      ax
mov       x,#16
push      ax
callt     [40h]
pop       ax
pop       ax
```

C言語形式

```
char      sig_sem(semid);
```

パラメータ

char　*semid	資源を返却するセマフォのアクセス・アドレス

説　　明

semidで指定したセマフォに資源を返却します。

対象セマフォの待ち行列にタスクがつながれている場合，先頭のタスクをwait状態（セマフォ待ち）からready状態へと遷移します。また，対象セマフォの待ち行列にタスクがつながれていない場合はセマフォの管理する資源数を0x1だけ増やします。

なお，セマフォの資源数が0xff個を越えた場合は，エラー・コード“E_QOVR”が返されます。

戻　り　値

E_OK	正常終了
E_QOVR	資源数が0xff個を越えた

SIGnal SEMaphore for Interrupt

isig_sem

割り込みハンドラのみ

概　　要

セマフォに対する信号操作を行う。

アセンブラ形式

```
movw      hl, #semid
callt     [46h]
```

C言語形式

```
char      isig_sem(semid);
```

パラメータ

char　*semid	資源を返却するセマフォのアクセス・アドレス

説　　明

semidで指定したセマフォに資源を返却します。

対象セマフォの待ち行列にタスクがつながれている場合，先頭のタスクをwait状態（セマフォ待ち）からready状態へと遷移します。また，対象セマフォの待ち行列にタスクがつながれていない場合は，セマフォの管理する資源数を0x1だけ増やします。

なお，セマフォの資源数が0xff個を越えた場合は，エラー・コード“E_QOVR”が返されます。

戻　り　値

E_OK	正常終了
E_QOVR	資源数が0xff個を越えた

WAIt on SEMaphore (17)

wai_sem

タスクのみ

概　　要

セマフォに対する待ち操作を行う。

アセンブラ形式

```
movw      ax, #semid
push      ax
mov       x,#17
push      ax
callt     [40h]
pop       ax
pop       ax
```

C言語形式

```
char      wai_sem(semid);
```

パラメータ

char　*semid	資源を要求するセマフォのアクセス・アドレス

説　　明

semidで指定したセマフォに対して資源を要求します。

対象セマフォから資源が獲得できる場合，本システムコール発行タスクは，対象セマフォの資源数を0x1減じます。このとき，本システムコール発行タスクは，wait状態（セマフォ待ち）に遷移することはなく，run状態のままとなります。

また，対象セマフォから資源が獲得できない場合，本システムコール発行タスクは対象セマフォの待ち行列にキューイングされ，wait状態（セマフォ待ち）へ遷移します。

戻　り　値

E_OK	正常終了

Poll and REQuest SEMaphore (18)

preq_sem

タスクのみ

概　　要

セマフォ資源を得る。

アセンブラ形式

```
movw      ax, #semid
push      ax
mov       x, #18
push      ax
callt     [40h]
pop       ax
pop       ax
```

C言語形式

```
char      preq_sem(semid)
```

パラメータ

char　*semid	資源を要求するセマフォのアクセス・アドレス

説　　明

semidで指定したセマフォに対して資源を要求します。

対象セマフォから資源が獲得できる場合は正常終了となり，対象セマフォの資源数を0x1減じます。この時のエラー・コードは“E_OK”が返されます。また，対象セマフォから資源が獲得できない場合，ポーリング・エラーとなり，エラー・コード“E_PLFAIL”が返されます。

戻　り　値

E_OK	正常終了
E_PLFAIL	セマフォの資源数が0x0である

SeND MeSsaGe (19)

snd_msg

タスクのみ

概　　要

メッセージを送信する。

アセンブラ形式

```
movw      ax, #mbxid
push      ax
movw      ax, #pk_msg
push      ax
mov       x, #19
push      ax
callt     [40h]
pop       ax
pop       ax
pop       ax
```

C言語形式

```
char      snd_msg(mbxid, pk_msg);
```

パラメータ

char	*mbxid	メッセージを送信するメイルボックスのアクセス・アドレス
char	*pk_msg	メイルボックスに送信するメッセージのアドレス

説　　明

mbxidで指定したメイルボックスに，pk_msgで指定したメッセージを送信します。

メッセージの送信とは，メッセージの内容がコピーされることではなく，pk_msgで指定したメッセージの先頭アドレスが渡されることです。

対象メイルボックスにメッセージを送信した時に対象メイルボックスの待ち行列にタスクがつながれている場合，待ち行列の先頭タスクにメッセージが渡されます。これにより，メッセージを受信したタスクは，wait状態（メッセージ待ち）からready状態へ遷移します。

また，対象メイルボックスの待ち行列にタスクがつながれていない場合，送信されたメッセージは，対象メイルボックスの待ち行列につながれ，メッセージを送信したタスクはrun状態のままとなります。

メイルボックスを使用したメッセージ通信では，メッセージを管理するためのヘッダを必要とします。このため，メッセージ通信で使用するメッセージの領域は，メモリブロック（pget_blkシステムコールの発行により獲得）を使用します。

戻　り　値

E_OK	正常終了

SeND MeSsaGe for Interrupt

isnd_msg

割り込みハンドラのみ

概　　要

メッセージを送信する。

アセンブラ形式

```
movw      hl, #mbxid
movw      de, #pk_msg
callt     [48h]
```

C言語形式

```
char      isnd_msg(mbxid, pk_msg);
```

パラメータ

char　*mbxid	メッセージを送信するメイルボックスのアクセス・アドレス
char　*pk_msg	メイルボックスに送信するメッセージのアドレス

説　　明

mbxidで指定したメイルボックスに，pk_msgで指定したメッセージを送信します。

メッセージの送信とは，メッセージの内容がコピーされることではなく，pk_msgで指定したメッセージの先頭アドレスが渡されることです。

対象メイルボックスにメッセージを送信した結果，対象メイルボックスの待ち行列にタスクがつながれている場合，待ち行列の先頭タスクにメッセージが渡されます。このため，メッセージを受信したタスクは，wait状態（メッセージ待ち）からready状態へと遷移します。

また，対象メイルボックスの待ち行列にタスクがつながれていない場合，送信されたメッセージは，対象メイルボックスの待ち行列につながれます。

メイルボックスを使用したメッセージ通信では，メッセージを管理するためのヘッダを必要とします。このため，メッセージ通信で使用するメッセージの領域は，メモリブロック（pget_blkシステムコールの発行により獲得）を使用してください。

戻　り　値

E_OK	正常終了

ReCeiVe MeSsaGe (20)

rcv_msg

タスクのみ

概　　要

メッセージの受信を待つ。

アセンブラ形式

```
movw      ax, #ppk_msg
push      ax
movw      ax, #mbxid
push      ax
mov       x, #20
push      ax
callt     [40h]
pop       ax
pop       ax
pop       ax
```

C言語形式

```
char      rcv_msg(ppk_msg, mbxid);
```

パラメータ

char	**ppk_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**
char	*mbxid	**メッセージを受信するメイルボックスのアクセス・アドレス**

説　　明

mbxidで指定したメイルボックスからメッセージを受信します。

対象メイルボックスからメッセージが受信できる場合，本システムコール発行タスクは，対象メイルボックスからメッセージを受信します。このとき，受信するメッセージは，待ち行列の先頭につながれているメッセージです。

また，対象メイルボックスからメッセージが受信できない場合，本システムコール発行タスクは，対象メイルボックスの待ち行列につながれ，wait状態（メッセージ待ち）へと遷移します。

なお，ppk_msg領域はユーザが確保してください。

戻　り　値

E_OK	**正常終了**

Poll and ReCeiVe MeSsaGe (21)

prcv_msg

タスクのみ

概　　要

メッセージを受信する

アセンブラ形式

```
movw      ax, #ppk_msg
push      ax
movw      ax, #mbxid
push      ax
mov       x, #21
push      ax
callt     [40h]
pop       ax
pop       ax
pop       ax
```

C言語形式

```
char      prcv_msg(ppk_msg, mbxid);
```

パラメータ

char	**ppk_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
char	*mbxid	メッセージを受信するメイルボックスのアクセス・アドレス

説　　明

mbxidで指定したメイルボックスからメッセージを受信します。

対象メイルボックスにメッセージが到着している場合，正常終了となり，対象メイルボックスからメッセージを受信します。この時のエラー・コードは“E_OK”が返されます。また，対象メイルボックスにメッセージが到着していない場合，ポーリング・エラーとなり，エラー・コード“E_PLFAIL”が返されます。

なお，ppk_msg領域はユーザが確保してください。

戻　り　値

E_OK	正常終了
E_PLFAIL	メッセージが到着していない

9. 3. 4　割り込み制御システムコール

この項では，次のシステムコールの仕様について示します。

ID番号	システムコール名	機　　能
－	ret_int	割り込みハンドラから復帰する
－	ret_wup	割り込みハンドラからの復帰とタスクの起床を行う

RETurn from INTerrupt handler

ret_int

割り込みハンドラのみ

概　　要

割り込みハンドラから復帰する。

アセンブラ形式

```
push    rp0
push    rp1
push    rp2
push    rp3
call    !@cnt_sav  ; 他ハンドラやタスクをCで記述している場合
br      !ret_int
```

本システムコールは他のシステムコールと違い，br命令にて分岐します。

C言語形式

```
void    ret_int();
```

パラメータ

なし

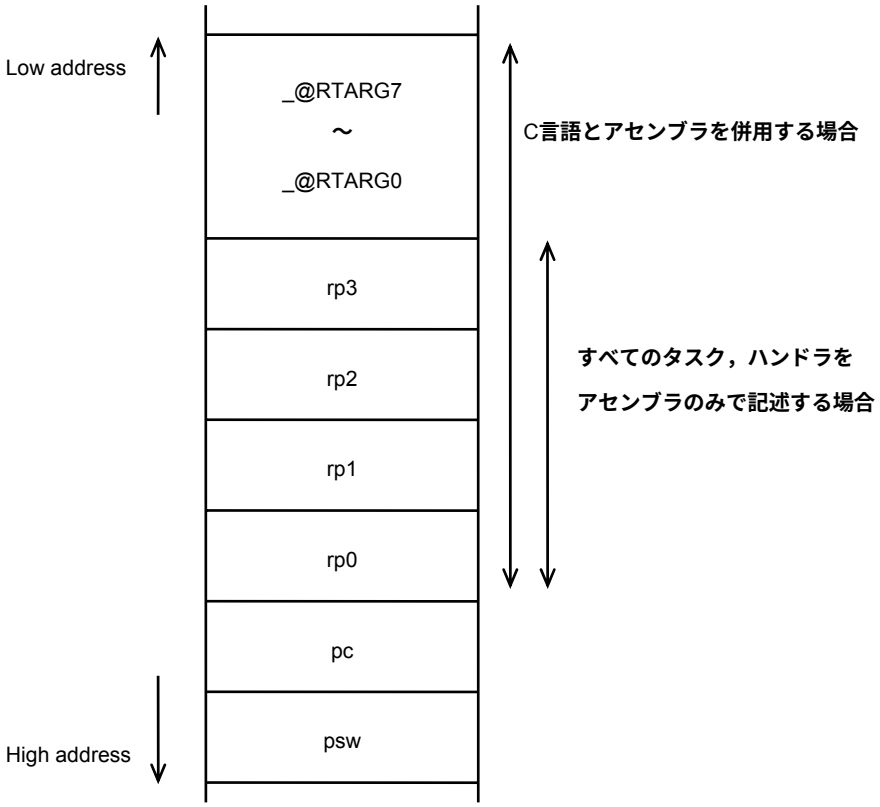
説　　明

割り込みハンドラからの復帰処理を行います。

システムコールを発行する割り込みハンドラは，必ず本システムコールにより割り込みから復帰してください。なお，割り込みハンドラ内で発行されたシステムコールにより，タスクを切り替える必要が生じても，本システムコールの処理が終了し，割り込みハンドラから抜けるまでタスクの切り替え処理は行われません。

本システムコールをアセンブラから発行する際には，割り込む前のレジスタ値を，スタック内にセーブしてからにしてください。また，C言語とアセンブラを併用してシステムを構築し，アセンブラで記述した割り込みハンドラから本システムコールを呼び出す場合には，“@cnt_sav”を呼び出してランタイム・ライブラリの引数をスタックにセーブしてください。

ret_int**システムコールを発行する際のスタック内容を示します。**



戻　り　値

なし

RETurn and WakeUP task

ret_wup

割り込みハンドラのみ

概　　要

割り込みハンドラからの復帰とタスクの起床を行う。

アセンブラ形式

```
push    rp0
push    rp1
push    rp2
push    rp3
call    !@cnt_sav  ; 他ハンドラやタスクをCで記述している場合
movw    hl, #tskid
br      !ret_wup
```

本システムコールは他のシステムコールと違い，br命令にて分岐します。

C言語形式

```
void    ret_wup(tskid);
```

パラメータ

char　tskid	起床させるタスクのアクセス・アドレス

説　　明

割り込み処理ハンドラからの復帰処理を行うとともに，tskidで指定され他タスクを起床させます。

ret_wupシステムコールは，iwup_tskとret_intの複合システムコールです。

本システムコールをアセンブラから発行する際には，割り込む前のレジスタ値を，スタック内にセーブしてからにしてください。また，C言語とアセンブラを併用してシステムを構築し，アセンブラで記述した割り込みハンドラから本システムコールを呼び出す場合には，“@cnt_sav”を呼び出してランタイム・ライブラリの引数をスタックにセーブしてください。

本システムコールを発行する際のスタック内容は，ret_intシステムコール呼び出し時と同じです。

戻 り 値

なし

9. 3. 5　メモリ管理システムコール

この項では，次のシステムコールの仕様について示します。

ID番号	システムコール名	機　　能
22	pget_blk	固定長メモリブロックを獲得する
23	rel_blk	固定長メモリブロックを解放する

Poll and GET fixed-length memory BLocK (22)

pget_blk

タスクのみ

概　　要

固定長メモリブロックを獲得する

アセンブラ形式

```
movw      ax, #p_blk
push      ax
movw      ax, #mplid
push      ax
mov       x, #22
push      ax
callt     [40h]
pop       ax
pop       ax
pop       ax
```

C言語形式

```
char      pget_blk(p_blk, mplid);
```

パラメータ

char	**p_blk	獲得したメモリブロックのアドレスを格納する領域のアドレス
char	*mplid	メモリブロックを獲得するメモリプールのアクセス・アドレス

説　　明

mplidで指定したメモリプールからメモリブロックを獲得します。

対象メモリプールに獲得できるメモリブロックが存在する場合，正常終了となり，対象メモリプールからメモリブロックを獲得します。この時のエラー・コードは“E_OK”が返されます。また，対象メモリプールに獲得できるメモリブロックが存在しない場合，ポーリング・エラーとなり，エラー・コード“E_PLFAIL”が返されます。

なお，獲得できるメモリブロックのサイズは，各メモリプールごとに，コンフィギュレータによって指定された値で固定となっています。

なお，p_blk領域はユーザが確保してください。

戻　り　値

E_OK	正常終了
E_PLFAIL	メモリブロックが獲得できない

RELease fixed-length memory BLocK (23)

rel_blk

タスクのみ

概　　要

固定長メモリブロックを解放する。

アセンブラ形式

```
movw      ax, #mplid
push      ax
movw      ax, #blk
push      ax
mov       x, #23
push      ax
callt     [40h]
pop       ax
pop       ax
pop       ax
```

C言語形式

```
char      rel_blk(mplid, blk);
```

パラメータ

char	*mplid	**メモリブロックを解放するメモリプールのアクセス・アドレス**
char	*blk	**解放するメモリブロックのアクセス・アドレス**

説　　明

mplidで指定したメモリプールに，blkで指定したメモリブロックを解放します。

戻 り 値

E_OK	**正常終了**

9.3.6　バージョン管理システムコール

この項では，次のシステムコールの仕様について示します。

ID番号	システムコール名	機　　能
25	get_ver	バージョン情報を獲得する

GET VERsion number (25)

get_ver

タスクのみ

概　　要

バージョン情報を獲得する。

アセンブラ形式

```
movw      ax, #pk_ver
push      ax
mov       x, #25
push      ax
callt     [40h]
pop       ax
pop       ax
```

C言語形式

```
char      get_ver(pk_ver)
```

パラメータ

char　**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**

説　　明

RX78K0**ニュークリアスのバージョン情報を，**pk_ver**で指定される領域に格納します。**

なお，pk_ver**領域はユーザが確保してください。**

以下に，バージョン情報の詳細を示します。

パラメータ	サイズ	値	意　　味
maker	2 byte	0x0117	**メーカ（**NEC Electronics**）**
id	2 byte	0x0	**形式番号**
spver	2 byte	0x5201	TRON**仕様書のバージョン** (μITRON Ver2.01)
prver	2 byte	0x0120	**製品のバージョン（**RX78K0 Ver1.20**）**
prno	2 byte*4	0x0	**製品管理番号**
cpu	2 byte	0x0d10	CPU**情報（**NEC Electronics RX78K0**）**
var	2 byte	0x0	**バリエーション記述子**

戻　り　値

E_OK	**正常終了**

9. 3. 7　時間管理システムコール

この項では，次のシステムコールの仕様について示します。

ID番号	システムコール名	機　　能
26	act_cyc	周期ハンドラの制御を行う
－	iact_cyc	周期ハンドラの制御を行う（割り込みハンドラから発行する）

ACTivate CYClic handler (26)

act_cyc

タスクのみ

概　　要

周期ハンドラの活性制御を行う。

アセンブラ形式

```
movw      ax, #cyhid
push      ax
mov       x, #cyhact
push      ax
mov       x, #26
push      ax
callt     [40h]
pop       ax
pop       ax
pop       ax
```

C言語形式

```
char      act_cyc(cyhid, cyhact);
```

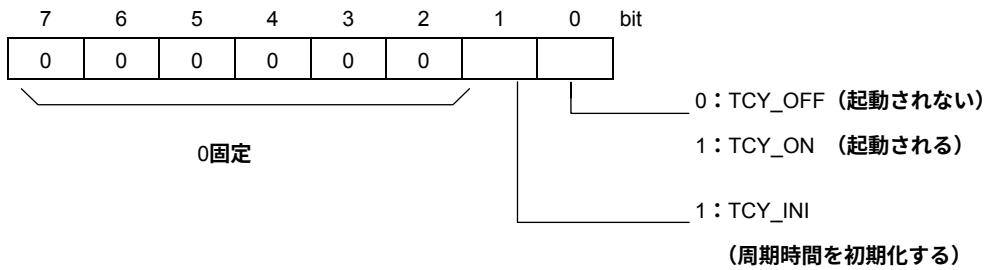
パラメータ

char	*cyhid	**活性制御を行う周期ハンドラのアクセス・アドレス**
char	cyhact	**周期ハンドラの活性状態**

説　　明

cyhid**で指定した周期ハンドラの活性状態を，**cyhact**で指定した状態に変更します。**

cyhact**の指定方法は，次のとおりです。**



cyhact := （TCY_OFF || TCY_ON） | [TCY_INI]

“TCY_OFF”は，周期ハンドラの一時的な中断を意味するため，この間は周期ハンドラが起動されません。

“TCY_ON”を指定すると，周期の経過とは独立に活性状態をONにします。ニュークリアスの内部動作としては，活性状態がOFFの間も指定周期時間のカウントダウンを行っているため，act_cyc, iact_cycシステムコールを発行してから最初に周期ハンドラが実行されるまでの時間は一定しません。

一方，“TCY_ON”と“TCY_INI”を同時に指定すると，活性状態をONにするのと同時に周期カウントをクリアしますので，本システムコールを発行してから指定周期時間経過後に，最初の周期ハンドラの起動が起こることになります。

なお，周期ハンドラを動的に生成および削除することはできません。

戻　り　値

E_OK	正常終了

ACTivate CYClic handler for Interrupt

iact_cyc

割り込みハンドラのみ

概　　要

周期ハンドラの活性制御を行う。

アセンブラ形式

```
mov       c, #cyhact
movw      hl, #cyhid
callt     [4ah]
```

C言語形式

```
char      iact_cyc(cyhid, cyhact);
```

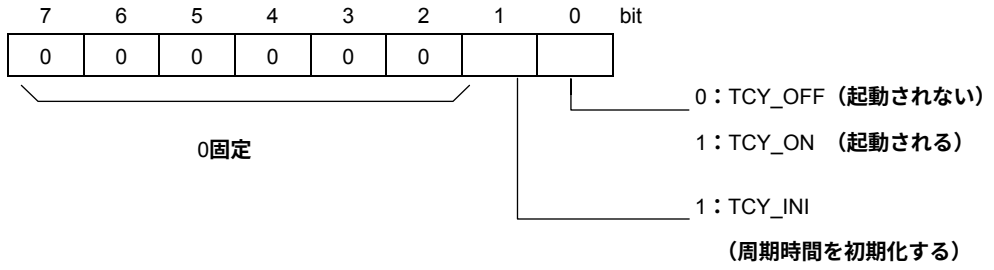
パラメータ

char　*cyhid	活性制御を行う周期ハンドラのアクセス・アドレス
char　cyhact	周期ハンドラの活性状態

説　　明

cyhidで指定した周期ハンドラの活性状態を，cyhactで指定した状態に変更します。

cyhactの指定方法は，次のとおりです。



cyhact :=　（TCY_OFF || TCY_ON）　| [TCY_INI]

“TCY_OFF”は，周期ハンドラの一時的な中断を意味するため，この間は周期ハンドラが起動されません。

“TCY_ON”を指定すると，周期の経過とは独立に活性状態をONにします。ニュークリアスの内部動作としては，活性状態がOFFの間も指定周期時間のカウントダウンを行っているため，act_cyc, iact_cycシステムコールを発行してから最初に周期ハンドラが実行されるまでの時間は一定しません。

一方，“TCY_ON”と“TCY_INI”を同時に指定すると，活性状態をONにするのと同時に周期カウントをクリアしますので，本システムコールを発行してから指定周期時間経過後に，最初の周期ハンドラの起動が起こることになります。

なお，周期ハンドラを動的に生成および削除することはできません。

戻　り　値

E_OK	**正常終了**

第10章　インタフェース・ライブラリ

本章では，インタフェース・ライブラリについて示します。

10.1　概　　要

RX78K0では，ユーザ・タスクの記述言語として，アセンブリ言語およびC言語の使用を推奨しています。

ユーザタスクをC言語で記述し，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の形式で記述します。この外部関数の形式で発行されたシステムコールを，ニュークリアスの発行形式に変換し，ユーザタスク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。

RX78K0が提供するインタフェース・ライブラリは，標準Cコンパイラとして当社のCC78K0の使用を推奨し，同コンパイラ用に設計されています。したがって，NECエレクトロニクスのCコンパイラ以外のコンパイラを使用する場合，インタフェース・ライブラリを変更しなければならない場合も生じます。

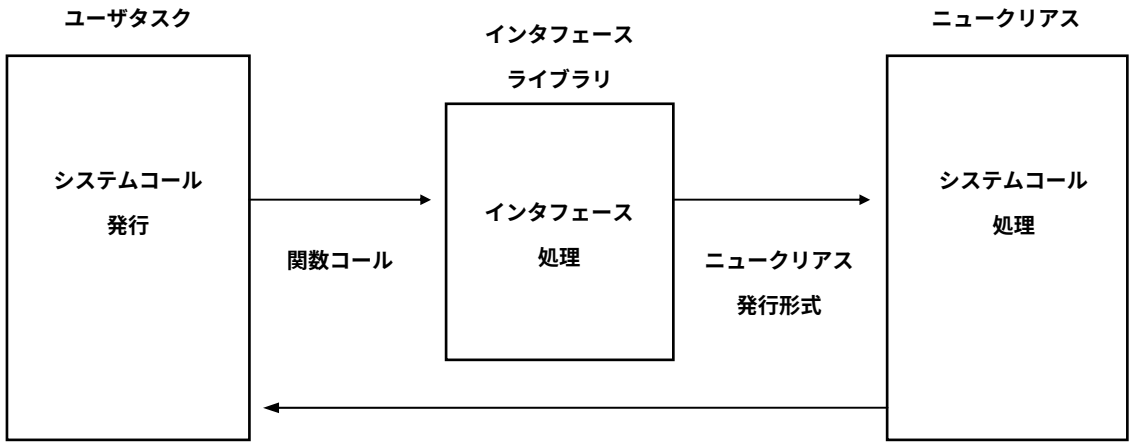
本章では，ユーザ独自のインタフェース・ライブラリの作成ができるように，提供のインタフェース・ライブラリの機能について示します。

10.2　インタフェース・ライブラリの機能と位置付け

インタフェース・ライブラリは，C言語で記述されたユーザタスクから外部関数の形式で発行されたシステムコールを，ニュークリアスの発行形式に変換するためのインタフェース・プログラムです。インタフェース・ライブラリは，ユーザタスクとニュークリアスの間に位置し，ニュークリアスが処理を行う上で必要な情報を設定し，ニュークリアスへ制御を移すという機能を持ちます。

図10－1に，本インタフェース・ライブラリの位置付けを示します。

図10－1　インタフェース・ライブラリの位置付け



付録A　マクロの利用方法

システムコールをアセンブラで記述するのが煩雑な場合は，マクロ記述も可能です。
利用方法は，アセンブルするファイル単位に“¥include¥asm_intf.mac**“をインクルードしてください。**
各システムコールの記述形式は，次のようになっています。

システムコール名	アセンブラ・マクロ記述形式
sta_tsk	sta_tsk　tskid
ext_tsk	ext_tsk
ter_tsk	ter_tsk　tskid
chg_pri	chg_pri　tskid, tskpri
rot_rdq	rot_rdq　tskpri
tsk_sts	tsk_sts　p_tsksts, tskid
slp_tsk	slp_tsk
wai_tsk	wai_tsk　tmout
wup_tsk	wup_tsk　tskid
iwup_tsk	iwup_tsk　tskid
can_wup	can_wup　p_wupcnt, tskid
set_flg	set_flg　flgid
iset_flg	iset_flg　flgid
clr_flg	clr_flg　flgid
wai_flg	wai_flg　flgid
cwai_flg	cwai_flg　flgid
pol_flg	pol_flg　flgid
cpol_flg	cpol_flg　flgid
sig_sem	sig_sem　semid
isig_sem	isig_sem　semid
wai_sem	wai_sem　semid
preq_sem	preq_sem　semid
snd_msg	snd_msg　mbxid, pk_msg
isnd_msg	isnd_msg　mbxid, pk_msg
rcv_msg	rcv_msg　ppk_msg, mbxid
prcv_msg	prcv_msg　ppk_msg, mbxid
ret_int	push　rp0 push　rp1 push　rp2 push　rp3 call　@cnt_sav　; C言語と併記している場合 retint
ret_wup	push　rp0 push　rp1 push　rp2 push　rp3 call　@cnt_sav　; C言語と併記している場合 retwup　tskid
pget_blk	pget_blk　p_blk, mplid
rel_blk	rel_blk　mplid, blk
get_ver	get_ver　pk_ver
act_cyc	act_cyc　cyhid, cyhact
iact_cyc	iact_cyc　cyhid, cyhact

付録B　C言語記述例

CC78K0を使用して，タスクおよび割り込みハンドラを記述した場合の例を示します。記述についての詳細は，CC78K0 Cコンパイラ　ユーザーズ・マニュアル　言語編を参照してください。

B. 1　タスクの記述例

```
#pragma sfr
#pragma NOP
#pragma RTOS_task task_1

extern char sta_tsk(unsigned short*);   /* sta_tsk()のextern宣言 */
extern char cwai_flg(unsigned short*);  /* cwai_flg()のextern宣言 */

extern unsigned short flg_id1;          /* イベントフラグ1のID(EVTアドレス) */
extern unsigned short tsk_id2;          /* タスク2のID(TCBアドレス) */



void task_1(void)
{
   int i;

   sta_tsk(&tsk_id2);                   /* タスク2を起動 (sta_tsk) */

   while(1)
   {
      cwai_flg(&flg_id1);               /* イベントフラグ1のセット待ち */
                                        /* (クリア機能あり) (cwai_flg) */
      NOP();
      for(i=0; i<50; i++)
      P1 =0x04;
   }
}
```

B. 2　割り込みハンドラの記述例

```
#pragma RTOS_interrupt INTP3 h1_func sp=int_sp

extern char iset_flg(unsigned short*);  /* iset_flg()のextern宣言 */

extern unsigned short flg_id1;          /* イベントフラグ1のID(EVTアドレス) */

void h1_func(void)
{
    iset_flg(&flg_id1);                 /* イベントフラグ1にフラグをセットする */
    ret_int();
}
```

付録C　周期ハンドラの記述方法

周期ハンドラは，br命令によってニュークリアス内の時間管理用の割り込みハンドラから起床されますので，同様に，br命令を用いて時間管理用の割り込みハンドラに戻ってください。

```
extrn   ?tm_ret

        |

（周期ハンドラの処理）

        |

br      !?tm_ret
```

以下に，周期ハンドラ記述の際の注意事項について示します。

- 周期ハンドラは割り込み許可状態で起床されますので，終了する際には割り込み許可状態で終了してください。
- 周期ハンドラで，b, hレジスタを使用する場合は，終了する際に元の値に戻してから終了してください。
- 周期ハンドラでレジスタバンクを切り替えて使用する場合は，終了する際に元のレジスタバンクに戻してから終了してください。
- 周期ハンドラから発行できるシステムコールは，割り込みハンドラから発行できるシステムコールのみです。

〔メ　モ〕

【発　行】
NECエレクトロニクス株式会社
〒211-8668　神奈川県川崎市中原区下沼部1753
電話（代表）：044(435)5111

お問い合わせ先

【ホームページ】

NECエレクトロニクスの情報がインターネットでご覧になれます。
URL(アドレス)　http://www.necel.co.jp/

【営業関係，技術関係お問い合わせ先】

半導体ホットライン
（電話：午前 9:00～12:00，午後 1:00～5:00）
電　話　：044−435−9494
E-mail　：info@necel.com

【資料請求先】

NECエレクトロニクスのホームページよりダウンロードいただくか，NECエレクトロニクスの販売特約店へお申し付けください。