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

RX78K4

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
78K4シリーズ

資料番号　U10603JJ3V1UM00（第3版）
発行年月　July 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 78K4シリーズの組み込み用ソフトウエアである，「78K4シリーズ用リアルタイムOS RX78K4」をお買上いただきまして，誠にありがとうございます。

本マニュアルは，78K4シリーズ用リアルタイムOS RX78K4の機能を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，“ CC78K4　Cコンパイラ ”と“ RA78K4　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　概　　要
本リアルタイムOSの概要や実行環境を説明します。

第2章　RX78K4**の基本機能**
本リアルタイムOSの基本機能について説明します。

第3章　RX78K4**の基本動作と制御ブロック**
本リアルタイムOSの動作と制御ブロックについて説明します。

第4章　RX78K4**のシステムコール一覧表**
本リアルタイムOSが提供するシステムコールについて説明します。

第5章　RX78K4**のリセット動作**
本リアルタイムOSを初期化するための処理について説明します。

付録A　**エラーコード一覧表とタスクの状態一覧表**
システムコールのリターンパラメータとタスクの状態を説明します。

付録B　C**言語での記述に関して**
Cコンパイラ“ CC78K4 ”がサポートしている機能について説明します。

付録C　**システムコール一覧表**
本リアルタイムOSがサポートするシステムコールの一覧表です。

付録D　**周期ハンドラの記述例**
周期ハンドラの記述例を示します。

付録E, F　メモリ容量の見積もり方法

本リアルタイムOSが使用するメモリ容量の見積もり方法を例を挙げて説明します。

付録G　スタンバイ機能について

78K4が提供しているスタンバイ機能を本リアルタイムOSを使用して実現するための方法を示します 。

付録H　システムコール別最大使用スタック・サイズ

システムコールを発行する際に使用するスタック量を示します。

付録I　シンボル名の制限事項について

本リアルタイムOSが使用しているシンボル名について示します。

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
RX78K4　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	U10604J
	タスク・デバッガ編	U15757J

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　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，ただちに反応できることを要求されます。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，システムが複雑化していき，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時間的に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，たとえば複数のタスクの実行を一定時間ごとに切り替えるという処理を行うとしますと，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。

上述の例では，タスクの実行を切り替えるきっかけを「一定時間」としましたが，このように，システム内で定めた何かの基準を「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　RX78K4とは

RX78K4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。

RX78K4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オペレーティングシステムになっています。

1. 3　RX78K4の概要と特徴

(1) μITRON Ver.2.01仕様に準拠

RX78K4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てシステムを構築できます。

(4) システム構築に有益なユーティリティの提供

システムを構築する上で有益な，二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
- 高級言語インタフェース

(5) RX78K4は，以下に示す78K4シリーズ用クロスツールに対応しています。

- RA78K4
- CC78K4

(6) 2種類のメモリモデルへの対応

CC78K4がサポートしているメモリモデルについて，以下のように対応しています。

- ラージモデル（1 Mバイト空間）－location 0fhに対応
- スモールモデル（64 Kバイト空間）－location 00hに対応

1. 4　RX78K4の構成とサイズ

RX78K4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）

RX78K4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。

ニュークリアスのサイズは，リンクするシステムコールによって異なりますが，ラージモデルの場合で最大約7 Kバイト，スモールモデルの場合で最大約5 Kバイトです。

(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78K4シリーズ用のCコンパイラ“CC78K4”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会話形式で入力および修正することができます。

1.5　RX78K4のメモリ構成

RX78K4を用いたシステムのメモリは，以下に示すもので構成されています。

（1）ニュークリアス（核）
（2）システム初期化用リセット・ルーチン
（3）システムコール・エントリ・アドレス格納テーブル
（4）OS管理領域[注]
（5）初期化情報テーブル
（6）オブジェクト管理用制御ブロック[注]
（7）ユーザ・プログラム

【ラージモデル】

0H～0FFFFH内の内部ROMまたは外部メモリ空間に，上記（1），（2），（3）が格納されます。

（1），（2），（3）は，リロケータブルなオブジェクトのため，上記範囲内の任意のメモリに配置することができます。

（4），（5），（6）は，（2）のリセット・ルーチンがシステムを初期化する時に参照する情報テーブルです（コンフィギュレータにより生成されます）。

（4），（5），（6）は，ユーザが任意のエリアに配置してください。

注　（4）と（6）を配置可能な領域は，最上位バイト固定の64 Kバイト以内です。
ユーザが指定したアドレスから，領域を確保します。

【スモールモデル】

0H～0FFFFH内の内部ROMまたは外部メモリ空間に，上記（1），（2），（3）が格納されます。

（1），（2），（3）は，リロケータブルなオブジェクトのため，上記範囲内の任意のメモリに配置することができます。

（4），（5），（6）は，（2）のリセット・ルーチンがシステムを初期化する時に参照する情報テーブルです（コンフィギュレータにより生成されます）。

（4），（6）は，内部RAM領域内の“0F700H～0F7FFH”の256バイト内に配置してください。

メモリ・マップについては，各デバイスのユーザーズ・マニュアルを参照してください。下記に，μPD784026にRX78K4に組み込む場合のメモリ構成例を示します。

図1－1　メモリ構成例（ラージモデル）

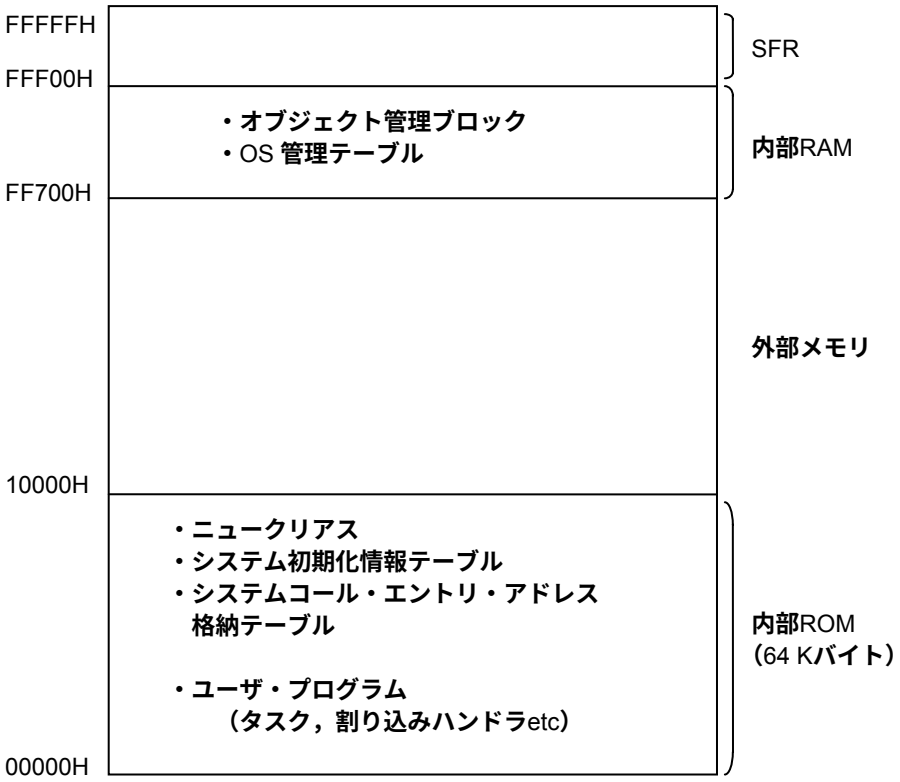
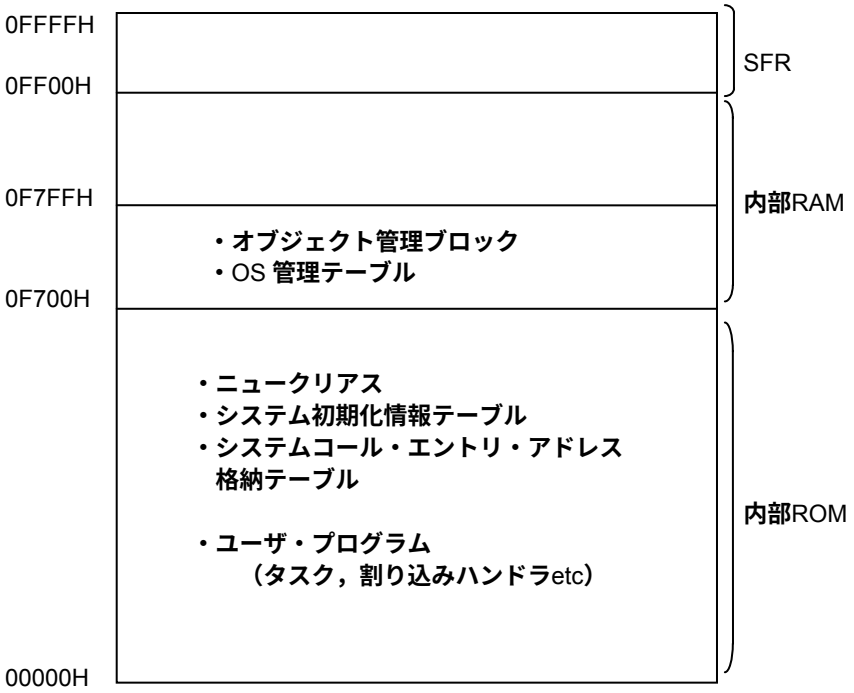


図1－2　メモリ構成例（スモールモデル）



第2章　RX78K4の基本機能

RX78K4は，マルチタスクOSであるので複数タスクを対象に処理を行います。

78K4シリーズには，8個のレジスタ・バンクが存在しますが，RX78K4では，このうちのバンク0を使用します。また，タスクをレジスタに割り付けて高速なタスク切り替えを実現します（詳しい説明は，3. 1　RX78K4の構成で示します）。

タスクの優先度は最大16レベル（0～15）あり，番号が小さいほど優先度は高くなります。ただし，優先度0と優先度1は，OSの予約優先度になりますので，ユーザ・タスクを割り当てることができる優先度は，2～15の14レベルです。

本章では，以下のような基本機能の項目について説明します。

（1）タスク状態
（2）タスク操作
（3）同期・通信
（4）割り込み処理
（5）メモリ管理
（6）時間管理

2. 1　タスク状態

タスクの取り得る状態は，次に挙げる4状態に分類することができます。

(1) 実行（RUN）状態

CPU利用権が与えられ現在実行中の状態です。システム内で，ある時点にただ1つだけ存在します。

(2) 実行可能（READY）状態

実行可能状態です。RUN状態になるための要素は全て揃っているが，そのタスクよりも優先順位が高いタスクが実行中であるため，実行できずにいる状態です。

(3) 待ち（WAIT）状態

資源要求などのシステムコールを発行したことにより待ちになった状態のことです。イベントフラグでのイベント待ち，セマフォでのリソース（資源）待ち，メイルボックスによるメッセージ待ち，時間待ちなどの場合があります。

(4) 休止（DORMANT）状態

RX78K4のリセット時，イニシャル・タスク以外のタスクの状態は休止状態になります。また，タスクが終了すると，休止状態になります。

2. 1. 1　タスク状態遷移

図2－1は，タスク状態遷移図です。各々の状態遷移は以下のような意味を持ち，そのためのシステムコールが用意されています。

(1) 起　　動

DORMANT状態のタスクをREADY状態にすることです。この状態遷移はシステムコールにより行われます。

(2) 終　　了

RUN状態のタスクをDORMANT状態にすることです。この状態遷移はシステムコールにより行われます。

(3) ディスパッチ（DISPATCH**）**

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優先度を持ちます。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。

タスクの優先度は最大14レベル（2～15）あり，番号が小さいほど優先度は高くなります。このほかに，優先度0と1がありますが，OSで使用するため，タスク優先度としては指定できません。

(4) プリエンプト（PREEMPT**）**

現在実行中のタスクよりも優先順位の高いタスクがRUN状態に遷移すると，現在実行中のタスクは一度中断されREADY状態に戻ります。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

何らかの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，一定時間待ちの要求を行ったがまだ時間が経過していない時などがこの場合にあたります。

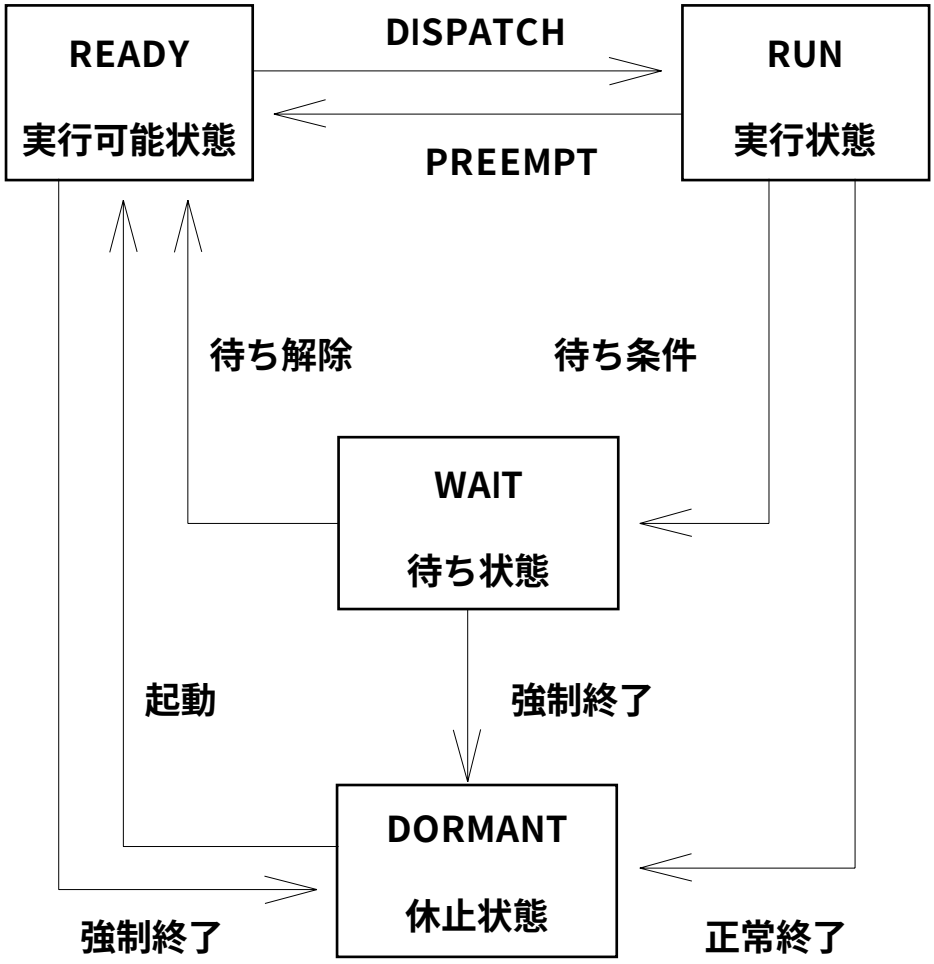
(6) 待ち解除

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状態にすることです。この状態遷移はシステムコールにより行います。

図2－1　RX78K4のタスク状態遷移図



2. 2　タスク操作

RX78K4は，タスク操作として次のような機能を持ちます。

(1) タスク起動

システム起動時，生成されたすべてのタスクはDORMANT状態となります。

DORMANT状態のタスクは，起動されるまでディスパッチの対象とはなりません。従って，“ sta_tsk” システムコールにより，READY状態へ移行させる必要があります。

(2) タスク終了

タスクを終了するためのシステムコールには，“ ext_tsk ”（正常終了）と“ ter_tsk ”（強制終了）があります。

※　一番最初の起動時及びタスク終了した後の起動時は，割り込み許可状態となっていますので，割り込み禁止についてはタスク毎に指示してください。

2.3　同期・通信

タスクは1つの独立したプログラム単位として独自の環境の下で実行されます。各々独立したタスクが並行に動き全体に1つにまとまったシステムを構成するためには，複数のタスクの実行の流れを制御したり，複数のタスクが共有している資源を管理したり，タスク間で情報の送受信を行ったりする必要があります。同期と通信はこれらの機能を実現するためのものです。

同期には，一般に待ち合わせと相互排除があります。

待ち合わせとは，タスク間で仕事を分担している場合に，一方のタスクの作業の終了まで他方タスクの作業を控えることです。この場合，その2つのタスクは協力関係にある，といいます。また，相互排除とは，タスク間で共通の資源を利用する場合に，あるタスクがその資源を利用している間は他のタスクが利用を見合わせることをいいます。この場合，その2つのタスクは競合関係にある，といいます。

タスク間の通信とは，メッセージのやりとりをタスク間で行うことをいいます。メッセージを受け取るタスクはメッセージが送られてくるまで処理を先へ進めることができないため，タスク間の通信には同期の機能も含まれています。

RX78K4では，同期・通信を実現する手段として，同期用にイベントフラグとセマフォを，通信用にメイルボックスを用意しています。

2.3.1　イベントフラグ

タスクの待ち合わせに用います。1ビットのイベントフラグとなっているため，OR待ち，AND待ちといった指定はありません。

タスクは，イベント発生を待つ場合に，システムコール“ wai_flg ”，“ cwai_flg ” ，“ pol_flg ” ，“ cpol_flg ”を発行します。イベントが発生していない場合は，発行タスクがWAIT状態に遷移する（wai_flg, cwai_flg発行の場合）か，あるいはWAIT状態には遷移せずにエラー・コードを返します（pol_flg, cpol_flg発行の場合）。

イベント発生を知らせる場合には，システムコール“ set_flg ”を発行します。また，イベントフラグをクリアするシステムコールは，“ clr_flg ”です。

イベントフラグでは，同一イベントフラグに対して複数タスクが待つことも可能です。従って，イベントフラグでもタスクが待ち行列を作ることになります。

この場合，1回の“ set_flg ”で，複数のタスクが待ち解除となります。待ち条件を満たすタスクのうち少なくとも1つのタスクがクリアを指定（cwai_flg）していれば，そのタスクの待ちを解除した後にフラグがクリアされるため，それ以降のタスクは待ち解除の対象とはなりません。

2.3.2　セマフォ

タスクの相互排除に用います。現在利用可能な資源の有無，または，資源数を管理できる計数型セマフォです。

セマフォに対する待ち操作（P命令）と信号操作（V命令）の要求資源数は，1に固定します。

タスクは資源が必要な時に，資源獲得操作（P命令）をシステムコール“ wai_sem ”，“ preq_sem ”を発行することにより行い，資源数が1以上の時には，セマフォの持つ資源数を1だけ減じます。資源数が0の場合は要求したタスクがWAIT状態に遷移する（wai_sem）か，あるいは，WAIT状態には遷移せずにエラー・コードを返します（preq_sem）。逆に，獲得した資源が不要になった時に資源返還操作（V命令）をシステムコール“ sig_sem ”を発行することにより行い，セマフォの持つ資源数を1だけ増やします。この時，資源待ちのタスクがある場合は最も早く待ちになったタスクを資源待ち状態から解放します。セマフォでのタスクの待ち方はFIFOのみです。

2. 3. 3　メイルボックス

メイルボックスは，タスク間メッセージの授受を行うための機能で，メッセージ受信待ちのタスク・キュー，送信待ちのメッセージ・キューを持ちます。

タスクは，システムコール“rcv_msg”，“prcv_msg”によりメイルボックスからの受信要求を行うことができます。“rcv_msg”発行時，メッセージがまだ届いていない時は，タスクはWAIT状態に移行しメイルボックスにキューイングされます。“prcv_msg”は，WAIT状態には遷移せずにエラー・コードが返されます。逆に，システムコール“snd_msg”によりメイルボックスへメッセージの送信を行います。待ちタスクが存在した場合は，WAIT状態を解除した後そのタスクにメッセージを送信し，待ちタスクが存在しない場合には，メッセージがキューイングされます。メッセージのキューイング方法及びタスクがメッセージを待つ方法としては，FIFOのみです。

2.4　割り込み管理

割り込み管理は，割り込みを事象として駆動される処理を管理することです。割り込みによりベクタする先の処理ルーチンを割り込みハンドラと呼び，タスクとは独立した扱いとなります。割り込みハンドラからの復帰とタスクを起床するの機能があります。

割り込みハンドラを終了し，タスクに制御を渡すためのシステムコールには“ret_int”，“ret_wup”があります。もし，レジスタ・バンクを占有している割り込みハンドラから“ret_int”を発行する際には，割り込む前に実行中だったタスクのレジスタ・バンクに切り換えてから“ret_int”を発行してください。

RX78K4は，各割り込みソースのイニシャライズを行わないので，各ソースはユーザ・タスクによりイニシャライズする必要があります。

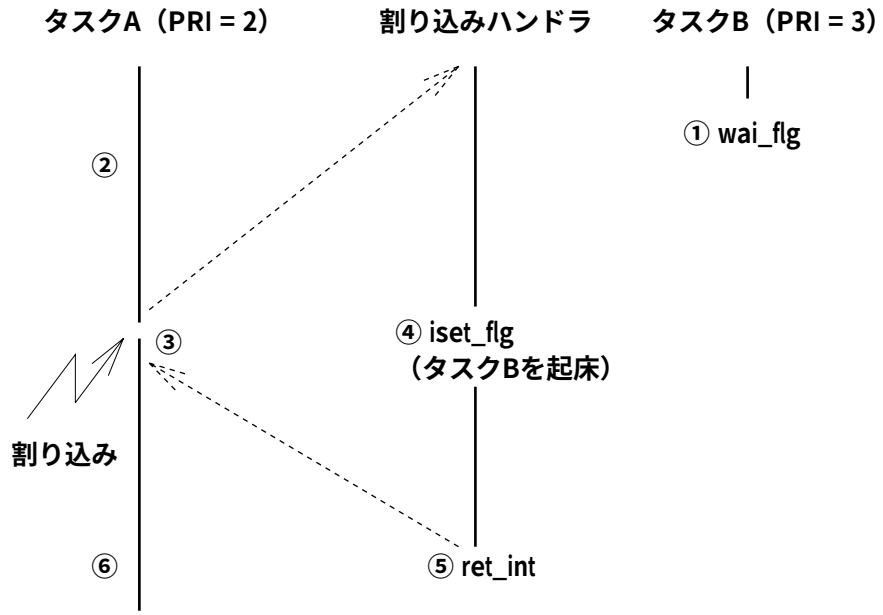
図2－2と図2－3に，割り込みハンドラ内でイベントフラグによるWAIT状態のタスクを起床させた場合の制御の流れを示します。システムコール“ret_int”，“ret_wup”では，割り込みハンドラ内でWAIT状態のタスクを起床させた場合，割り込みが発生した実行中のタスクより起床タスクのプライオリティが低い時は，制御を割り込みが発生した実行タスクの割り込まれた時点に戻します。

逆に，起床タスクが実行中のタスクよりプライオリティが高い場合は，起床タスクに制御を渡します。図2－2は，タスクAがタスクBよりプライオリティが高い場合の制御の流れを示しています。

注意　・NMIの割り込み処理では，システムコールを発行できません。
・割り込みレベル3（最下位レベル）に設定された割り込み同士の多重割り込みは，RX78K4ではサポートしていません。十分に注意してください。また，タイマ処理についても同様です。

① タスクBはシステムコール“wai_flg”を発行してWAIT状態である。
② タスクAが実行中（READY状態のタスクの中で最もプライオリティが高い）である。
③ タスクAを実行中に割り込みが入り，割り込みハンドラへ処理が移る。
④ 割り込みハンドラ先でシステムコール“iset_flg”を発行。タスクBが起床する。
⑤ 割り込みハンドラ先でシステムコール“ret_int”を発行。
⑥ タスクAの方がタスクBよりもプライオリティが高いので，処理はタスクAに戻る。

図2－2　“ret_int”の例



次の図2－3は，タスクAがタスクBよりもプライオリティが低い場合の例です。

① タスクBはシステムコール“wai_flg”を発行してWAIT状態である。

② タスクAが実行中（READY状態のタスクの中で最もプライオリティが高い）である。

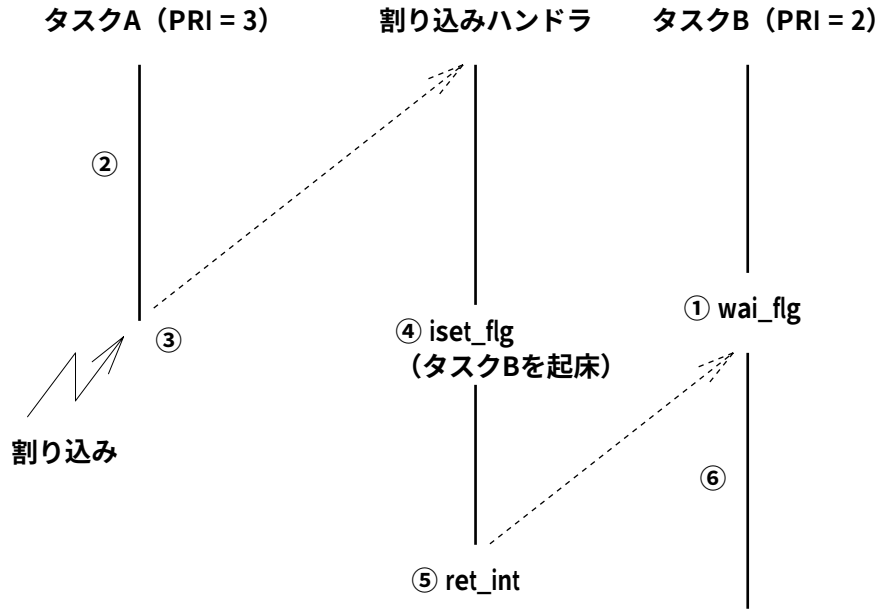
③ タスクAを実行中に割り込みが入り，割り込みハンドラへ処理が移る。

④ 割り込みハンドラ先でシステムコール“iset_flg”を発行。タスクBが起床する。

⑤ 割り込みハンドラ先でシステムコール“ret_int”を発行。

⑥ タスクBがタスクAよりもプライオリティが高いので，処理はタスクBに移る。

図2－3　“ret_int”の例



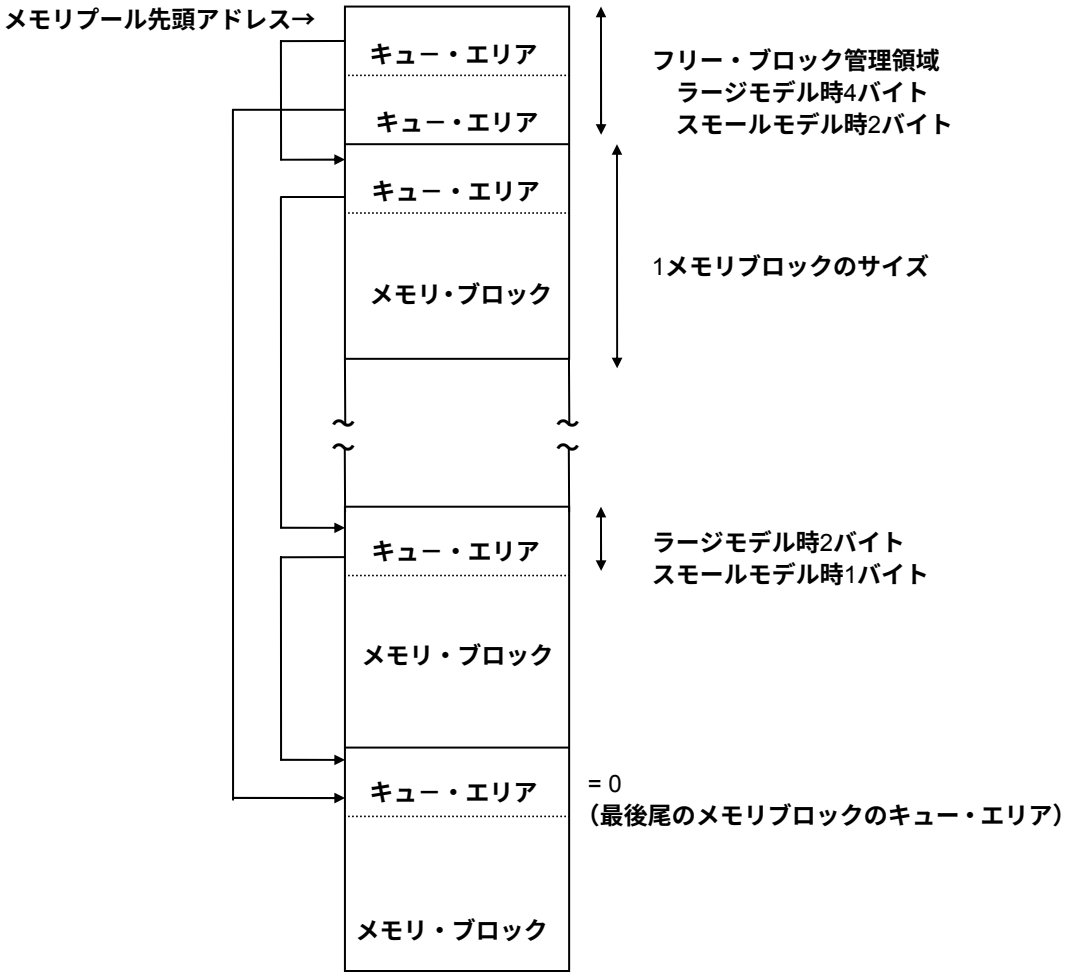
2.5　メモリ管理

RX78K4は，固定サイズのメモリブロックに基づいて動的なメモリ管理を行います。つまり，タスクが必要な時にメモリを獲得し，不必要になったら解放することができます。この事を利用すれば，複数のタスクに対して予めメモリを用意せずにすみ，効率よくメモリを利用できます。

RX78K4のメモリブロックは，システム実行中のシステムコールにより動的に獲得され，使用後，不必要になった時には返却されます。1メモリブロックを獲得できなくても，タスクの状態遷移は起こらずエラー・コードだけが返されます。また，1メモリブロックのサイズは，システム生成時に指定された値で固定となります。

通常，獲得したメモリブロックはメッセージ・エリアとして利用します。その場合，ラージモデルは最初の2バイト，スモールモデルは最初の1バイトはキューのリンク・エリアとして利用されるため，メッセージとして利用できる空間は，リンク・エリアを除いたエリアとなります。

図2－4　メモリプールの構成



注意　メモリブロックをメッセージ領域として使用する場合は，1メモリブロックのサイズを2の倍数にしてください。2の倍数以外の場合は，動作の保証はできません。

2. 6　時間管理

時間管理は，タスクの遅延起床および周期ハンドラの起動を管理することです。

タスクの遅延起床とは，指定された時間が経過した後にタスクに対して起床処理を行うことで，“ wai_tsk ”システムコールにより実現します。また，周期ハンドラの起動とは，指定した時間間隔で割り込みハンドラの起動を管理することで，“ act_cyc（iact_cyc） ”システムコールにより実現します。

RX78K4では，CPUが持っている16ビット・タイマ／カウンタ・ユニット内の1つを，インターバル・タイマとして使用します。OSが提供しているタイマ・ディスパッチ・ルーチン（?tmdsp）を，使用するタイマのベクタ・テーブルに割り付けることにより，一定時間毎に起動します。タイマ・ディスパッチ・ルーチンでは，“ wai_tsk ”システムコール発行タスクの待ち時間と，周期ハンドラの待ち時間のカウントを行い，指定時間に達したときに該当タスクを起床，または該当ハンドラを起動します。

具体的な状態遷移を示すと，“ wai_tsk ”を発行したタスクはWAIT状態からREADY状態へ遷移し，周期ハンドラはタイマ・ディスパッチ・ルーチンから直接呼び出されます。

第3章　RX78K4の基本動作と制御ブロック

RX78K4におけるシステムの基本動作と，RX78K4が管理する制御ブロックの概要について説明します。

3. 1　RX78K4の基本動作

3. 1. 1　全体の処理の流れ

RX78K4は，78K4シリーズの8個のレジスタ・バンクのうち，レジスタ・バンク0を使用します。ユーザ・タスクは，レジスタ・バンク1に複数タスクを割り当て，レジスタ・バンク2～7には1タスクずつ割り当てることができます。

レジスタ・バンク2～7に割り当てられたタスクは，タスクが実行途中で切り替わった時にレジスタ等を退避する必要がなく，高速なタスク切り替えが実現できます。

以下に，レジスタ・バンクの割り当て方法の例を示します。

例1.　タスクが6個の場合

レジスタ・バンク1	・・・	なし
レジスタ・バンク2～7	・・・	1タスクずつ

例2.　タスクが6個の場合

レジスタ・バンク1	・・・	3タスク
レジスタ・バンク2～4	・・・	1タスクずつ
レジスタ・バンク5～7	・・・	割り込み用

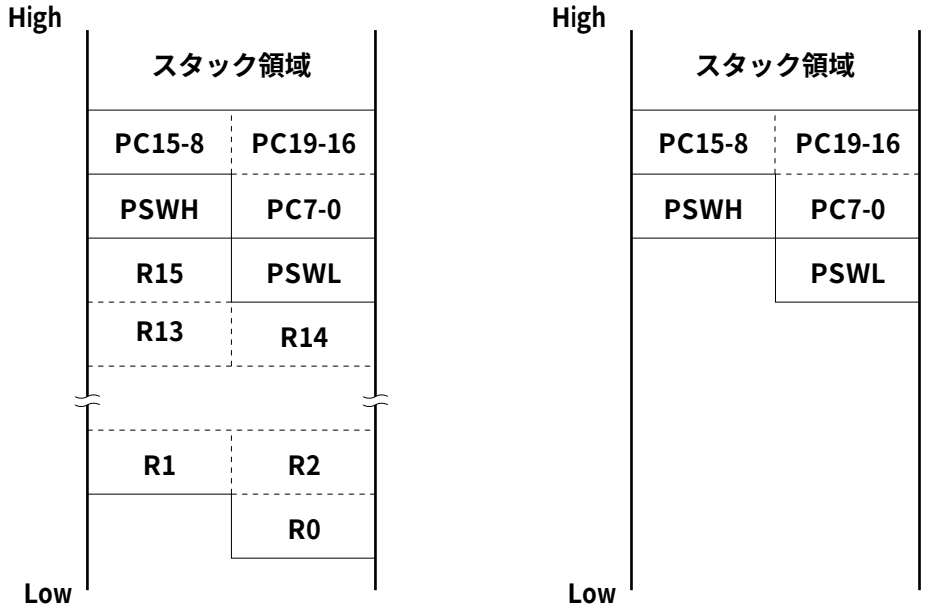
例3.　タスクが10個の場合

レジスタ・バンク1	・・・	4タスク
レジスタ・バンク2～7	・・・	1タスクずつ

また，スタック領域はタスク毎に必要で，タスク実行時には実行タスクのスタック領域を使用します。割り込みハンドラでは，割り込みが発生したときに実行していたタスクのスタック領域を使用します。次に，ディスパッチ時のスタックの状態を示します。

図3－1　ディスパッチ時のスタックの状態

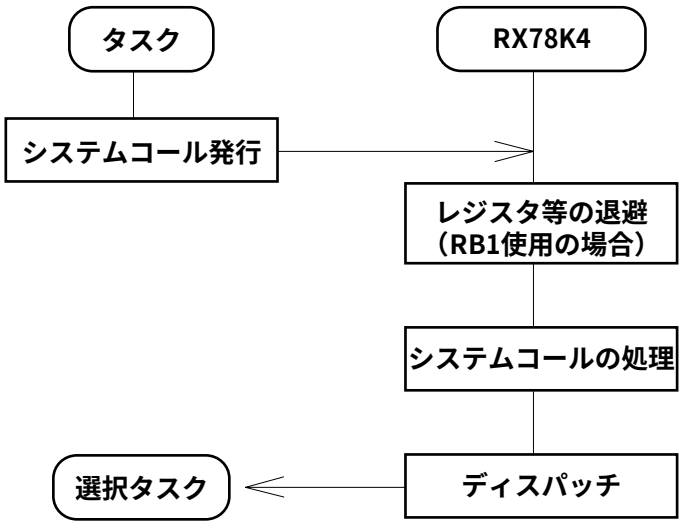
タスクをレジスタ・バンク1に割り付けた場合　　タスクをレジスタ・バンク2～7に割り付けた場合



レジスタ・バンク1は複数のタスクが共用するバンクのため，ディスパッチとプリエンプト時にはレジスタの退避と復帰が生じます。このため，レジスタ・バンク2～7に割り当てられたタスクよりも，多少オーバ・ヘッドが大きくなります。

図3－2に，システムコールを発行してからタスクに制御が戻るまでの処理の流れを示します。

図3－2　全体の処理の流れ



3. 1. 2　レディ・キューの構成とディスパッチ方法

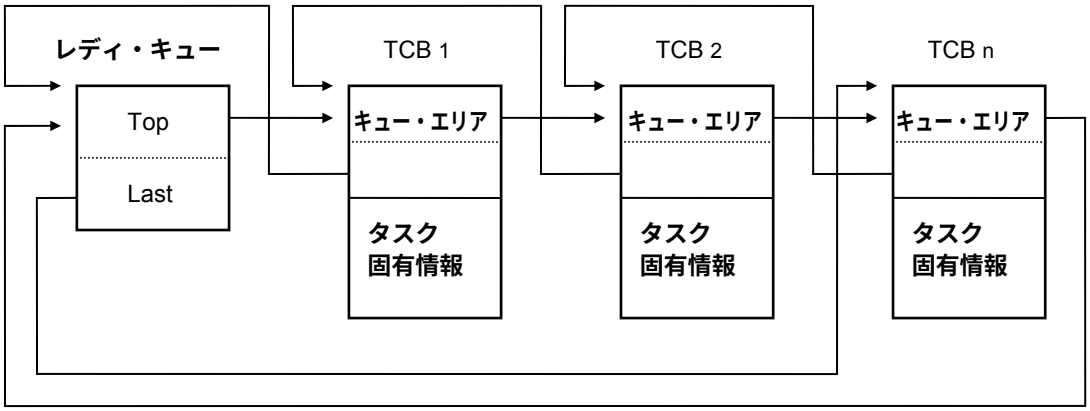
RX78K4は，CPU実行権をタスクに渡す際，READY状態のタスクの中で最もプライオリティの高いタスクを探し出し実行権を渡しますが，この時に参照するのがレディ・キューです。

RX78K4は，16レベルの優先度（0～15）毎にレディ・キューを持っています。各レディ・キューの先頭にチェインされているタスクが同優先度内のタスクの中で最高のプライオリティを持ち，逆に，最後尾にチェインされているタスクが最下位のプライオリティを持ちます。ただし，16レベルの優先度のうち，タスクに指定可能な優先度は2～15です。

RX78K4のディスパッチャでは，上位優先度のレディ・キューからタスクを探し，最初に探し出したタスクをRUN状態に移します。ただし，この時実行可能なユーザ・タスクが無い場合は，CPUをHALT状態にしますので，ユーザ・タスクの実行再開は割り込みによらなくてはなりません。この時に使用するスタック・エリアは，最後に実行していたタスクのスタック・エリアです。

図3－3に，レディ・キューの構成を示します。

図3－3　レディ・キューの構成



3. 2　RX78K4の制御ブロック

RX78K4は，各オブジェクトを管理するための制御ブロックを持っています。制御ブロックの作成はユーザがコンフィギュレート時に行い，RX78K4はそれをシステムコールに利用，または管理します。このため，ユーザ・タスクからはいつでも制御ブロックの状態を把握することができます。RX78K4の制御ブロックには以下のものがあります。

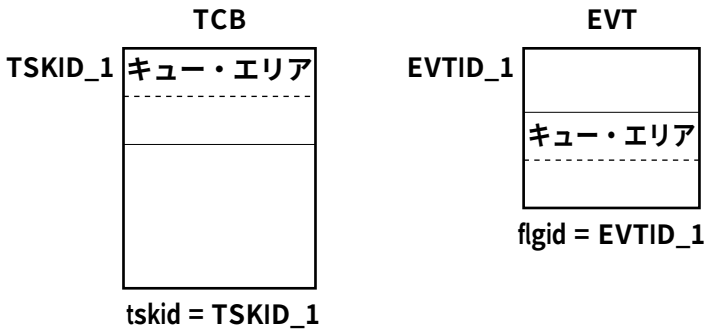
① TCB（Task Control Block）
② EVT（Eventflag）
③ SEM（Semaphore）
④ MBX（Mail Box）
⑤ MPL（Memory Pool）
⑥ HCB（Cyclic Handler Control Block）

3. 2. 1　制御ブロックのメモリ・エリア

前述したように，RX78K4では制御ブロックの作成はユーザがコンフィギュレート時に行います。このため，システムコール（制御ブロックに関するもの）に対するパラメータの受け渡しは，全てシンボル名によるアドレス渡しとなります。

図3－4に，パラメータの例を示します。

図3－4　パラメータの例



xxxid = 対象オブジェクトの先頭アドレス

3. 2. 2　各制御ブロックの説明

① タスク制御ブロック（TCB）

RX78K4がタスクを管理するために使用する制御ブロックで，各タスクに一つずつ割り当てられます。

TCBのサイズは，ラージモデルが10バイト，スモールモデルは8バイトです。

② イベントフラグ（EVT）

イベントフラグは，タスク間の同期や待ち合わせに用いられるイベント管理用の制御ブロックです。

イベントフラグ・サイズは，ラージモデルが4バイト，スモールモデルは2バイトです。

※ イベントフラグのセットが行われる時に，複数のタスクが待ち状態の場合，全ての待ち状態タスクがREADY状態へと移行します。ただし，“ cwai_flg ”を発行して待ち状態となったタスクが存在する場合は，そのタスクを待ち解除とした後にフラグがクリアされますので，それ以後のタスクの状態遷移は起こりません。

③ セマフォ（SEM）

セマフォは，タスク間の同期や相互排除に用いられる資源管理用の制御ブロックです。

セマフォ・サイズは，ラージモデルが4バイト，スモールモデルは2バイトです。

④ メイルボックス（MBOX）

メイルボックスは，タスク間のメッセージ通信を管理するための制御ブロックです。

メイルボックス・サイズは，ラージモデルが4バイト，スモールモデルは2バイトです。

⑤ メモリプール（MPL）

メモリプールは，フリー・メモリブロックを管理します。メモリブロックは，通常，メッセージ領域として使用します。メッセージ領域をして使用する際は，必ず1メモリブロックのサイズを2の倍数にしてください。また，メッセージ以外の目的でも，メモリブロックを必要な場合だけ確保することができます。

メモリプールのサイズは，フリー・ブロック管理領域とメモリブロックの合計になります。

フリー・ブロック管理領域は，ラージモデルが4バイト，スモールモデルは2バイトです。

⑥ 周期ハンドラ制御ブロック（HCB）

RX78K4が周期ハンドラを管理するために使用する制御ブロックで，各周期ハンドラに1つずつ割り当てられます。

周期ハンドラ制御ブロックのサイズは，ラージモデルが10バイト，スモールモデルは8バイトです。

第4章　RX78K4のシステムコール一覧表

本章では，RX78K4の各々のシステムコールを，次の形式にしたがって説明します。

機　　能

各システムコール機能の概要説明。

解　　説

システムコール機能の解説。

システムコールID番号

アセンブラからシステムコールを呼び出す際の，システムコールの固有番号。

パラメータ

各パラメータについて，名称，データ・サイズ，説明の順番で記述します。

リターン・パラメータ

システムコールの結果として発生する返却値の一覧。

返却値は，C言語の時はシステムコールの関数値として，また，アセンブラではCレジスタに返されます。

全てのシステムコールに対して異常終了時の処理がありませんので，リターン・パラメータの値を参照して異常終了時の処理を追加してください。

アセンブラ・フォーマット

アセンブラでシステムコールを発行する時の，入力パラメータの受け渡し方法および，出力パラメータがどのレジスタを使用するかを記述します。

割り込みハンドラから呼び出されるシステムコールについては，どのレジスタが入力パラメータになるかを記述します。フォーマットに記述しているシンボルは，次のアドレスとなります。

【ラージモデル】

```
bnk0_b = 0FFEF3H    bnk0_vp = 0FFEF8H   bnk0_up = 0FFEFAH
bnk0_e = 0FFEFCH    bnk0_d = 0FFEFDH
```

【スモールモデル】

```
bnk0_b = 0FEF3H     bnk0_vp = 0FEF8H    bnk0_up = 0FEFAH
bnk0_e = 0FEFCH
```

割り込みハンドラから発行可能なシステムコールでは，その時のレジスタ・バンクを使用します。アセンブラ・フォーマットでは，次のように記述します。

【ラージモデル】

例）　E,　UUP,　VVP

【スモールモデル】

例）　E,　UP,　VP

Cフォーマット

C言語でシステムコールを発行する場合の，記述形式とパラメータの型を記述します。

◎　C言語とのインタフェース

アプリケーション・プログラムを開発するためにサポートされている高級言語は，C言語です。C言語でシステムコールを発行する場合，OSとのインタフェースをとるためのアセンブラ・ルーチンを必要としますので，タスク毎に，使用しているシステムコールに対応するインタフェース・ルーチンをリンクしなければなりません。

C言語でのシステムコールの形式は，次のとおりです。

```
ret = <name> ([<parameter>], ...) ;
```

ret	：関数返却値（char型）
<name>	：システムコール名
<parameter>	：入力パラメータ

各パラメータのデータ・タイプとそのサイズは，以下のとおりです。

char	：8ビット整数
unsigned short	：16ビット整数
char *	：ラージモデル時24ビット・ポインタ
	：スモールモデル時16ビット・ポインタ

◎　アセンブラとのインタフェース

アセンブラでシステムコールを発行する場合は，下記のようにパラメータをレジスタ・バンク0の該当レジスタに設定してから呼び出してください。

sta_tsk発行例（ラージモデル）

```
MOV     BNK0_REG1,#システムコールID番号
MOVW    BNK0_REG2,#パラメータ1（下位16ビット）
MOV     BNK0_REG3,#パラメータ1（上位8ビット）
CALLT   [40H]
```

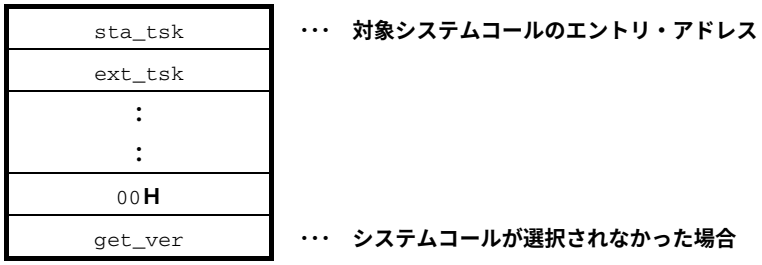
sta_tsk発行例（スモールモデル）

```
MOV     BNK0_REG1,#システムコールID番号
MOVW    BNK0_REG2,#パラメータ1
CALLT   [40H]
```

40Hには，システムコールの各処理に分岐するための分岐処理モジュールのエントリ・アドレスが設定されています。分岐処理モジュールでは，コンフィギュレータで生成されるテーブルより，各システムコールのエントリ・アドレスを取り出し分岐します。

システムコールのエントリ・アドレス格納テーブルの構成は，次のようになっています。

図4－1　システムコール・エントリ・アドレス格納テーブル



割り込みハンドラから発行する下記のixxx_xxxシステムコールは，分岐処理モジュールを介さずに直接，呼び出してください。（　）内は，CALLTテーブルのアドレスを示します。ただし，未使用のシステムコールに対応するアドレスは，ユーザが使用しても構いません。

ichg_pri（42H），irot_rdq（44H），iwup_tsk（46H），iset_flg（48H），
isig_sem（4AH），isnd_msg（4CH）

また，ret_int, ret_wupシステムコールは，BR命令で直接各システムコールの処理に分岐します。

◎　“xxx_xxx”システムコールと“ixxx_xxx”システムコールの違い

“ixxx_xxx”システムコールは，割り込みハンドラから起動されるシステムコールのため，自システムコールではディスパッチを起こしません。ディスパッチが起きるのは，割り込みハンドラからの復帰“ret_int”または，割り込みハンドラからの復帰とタスク起床“ret_wup”が発行された時です。また，割り込みハンドラからの復帰時にRETI命令で復帰した場合は，次のシステムコールでディスパッチが起こります。

4.1　タスク関連

タスク関連には，以下に挙げるシステムコールがあります。

(1) sta_tsk　タスクの起動
(2) ext_tsk　自タスクの正常終了
(3) ter_tsk　他タスクの強制終了
(4) chg_pri　タスク優先度の変更
　　ichg_pri　〃　（割り込みハンドラ内で使用する場合）
(5) rot_rdq　指定優先度レディ・キューの回転
　　irot_rdq　〃　（割り込みハンドラ内で使用する場合）
(6) tsk_sts　タスクの状態を見る
(7) slp_tsk　タスクの待ち状態への移行
(8) wai_tsk　タスクの一定時間待ち状態への移行
(9) wup_tsk　タスクの起床
　　iwup_tsk　〃　（割り込みハンドラ内で使用する場合）
(10) can_wup　タスクの起床要求の無効

Start Task

sta_tsk

機　能

タスクを起動し，DORMANT状態からREADY状態へ遷移させる。

解　説

tskidで示されたタスクを起動します。タスクの状態は，DORMANT状態からREADY状態へ移行します。DORMANT状態でないタスクに対して本システムコールを発行することはできません。この場合，エラーコードとして“E_NODMT”が返されます。

初めてタスクを起動する場合および，一度終了した後に再起動する場合は，タスクのスタート・アドレスから処理が行われ，スタック・ポインタの値も初期設定時の値に設定されます。その他のレジスタの値は不定です。

“sta_tsk”システムコールの対象タスクは，割り込み許可状態になっていますので，割り込み禁止についてはタスク毎に指定してください。

システムコールID番号

```
sta_tsk = 0
```

パラメータ

tskid（Task Identifier）	タスクID（TCB先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了
E_NODMT	対象タスクがDORMANT状態でない

アセンブラ・フォーマット

【ラージモデル】

```
MOV     bnk0_b,#0
MOVW    bnk0_up,#tskid
MOV     bnk0_d,#tskid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#0
MOVW    bnk0_up,#tskid
CALLT   [40H]
:
```

Cレジスタ = リターン・パラメータ

Cフォーマット

```
ret = sta_tsk（tskid）；
char *tskid；
```

Exit Task

ext_tsk

機　能

自タスクを正常終了し，RUN状態からDORMANT状態へ移行する。

解　説

自タスクを正常終了します。タスクの状態はRUNからDORMANTへ移行します。

本システムコールの発行によりDORMANT状態へ移行したタスクが“ sta_tsk ”システムコールにより再起動する時は，スタート・アドレスとスタック・ポインタは初期値に戻ります。また，起床要求カウンタは0にクリアされます。

“ ext_tsk ”システムコールの発行により，タスクがそれより以前に獲得していた資源を自動的に解放することはありません。したがって，資源の解放は本システムコール発行前に行ってください。

システムコールID番号

```
ext_tsk = 1
```

パラメータ

なし

リターン・パラメータ

なし（本システムコールを発行したタスクには戻りません。）

アセンブラ・フォーマット

【ラージモデル／スモールモデル】

```
MOV         bnk0_b,#1
CALLT       [40H]
```

Cフォーマット

```
ext_tsk () ;
```

Terminate Task

ter_tsk

機　能

他タスクを強制的に異常終了させる。

解　説

tskidで示されたタスクを強制的に異常終了させます。

対象タスクが待ち状態に入り，何らかの待ち行列につながれていた場合には，“ter_tsk”の発行によりその待ち行列から外されます。つまり，対象タスクは待ち状態が解除になってから終了します。

“ter_tsk”では，対象タスクがそれ以前に獲得していた資源を自動的に解放しません。タスクを終了する前に資源を解放しておいてください。

本システムコールでは，自タスクの指定はできません。

システムコールID番号

```
ter_tsk = 2
```

パラメータ

tskid（Task Identifier）	タスクID（TCB先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了
E_DMT	対象タスクがDORMANT状態である

アセンブラ・フォーマット

【ラージモデル】

```
MOV     bnk0_b,#2
MOVW    bnk0_up,#tskid
MOV     bnk0_d,#tskid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#2
MOVW    bnk0_up,#tskid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

```
ret = ter_tsk（tskid）；
char *tskid；
```

chg_pri
ichg_pri

Change Task Priority

Change Task Priority for interrupt

機　能

タスク優先度を変更する。

解　説

tskidで示されたタスクの現在の優先度を，tskpriで示される値に変更します。

tskid = 0 で自タスクの指定になります。ただし，ichg_priでは自タスクの指定はできません。本システムコールで変更された優先度は，再度変更されるまで，または，タスクの実行が終了するまで有効です。タスクが再起動された時の優先度は，初期優先度になります。

システムコールID番号

chg_pri = 3

パラメータ

tskid （Task Identifier）	タスクID（TCB先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit
tskpri （Task Priority）	タスクの優先度　8bit

リターン・パラメータ

E_OK	正常終了
E_DMT	対象タスクがDORMANT状態である

アセンブラ・フォーマット

・chg_priの場合

【ラージモデル】

```
MOV     bnk0_b,#3
MOV     bnk0_e,#tskpri
MOVW    bnk0_up,#tskid
MOV     bnk0_d,#tskid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#3
MOV     bnk0_e,#tskpri
MOVW    bnk0_up,#tskid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

・ichg_priの場合

【ラージモデル】

```
MOV       E,#tskpri
MOVG      UUP,#tskid
CALLT     [42H]
          :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV       E,#tskpri
MOVW      UP,#tskid
CALLT     [42H]
          :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

・chg_priの場合

```
ret = chg_pri (tskid,tskpri) ;
char *tskid;
char tskpri;
```

・ichg_priの場合

```
ret = ichg_pri (tskid,tskpri) ;
char *tskid;
char tskpri;
```

rot_rdq
irot_rdq

Rotate Ready Queue

Rotate Ready Queue for Interrupt

機　能

指定優先度のレディ・キューを回転する。

解　説

tskpriで示される優先度のレディ・キューを回転します。すなわち，その優先度のレディ・キューの先頭につながれているタスクをレディ・キューの最後尾につなぎかえ，同一優先度のタスクの実行を切り換えます。

tskpri = 0 で，自タスクの持つ優先度のレディ・キューを回転します。つまり，自ら実行権を放棄するために本システムコールを発行することができます。その時，自タスクはそのレディ・キューの最後尾につながれます。ただし，irot_rdqでは，tskpri = 0 の指定はできません。

指定された優先度のレディ・キューにタスクが無い場合は何もしませんが，エラーとはなりません。

システムコールID番号

rot_rdq = 4

パラメータ

tskpri（Task Priority）	タスクの優先度　8bit

リターン・パラメータ

E_OK	正常終了

アセンブラ・フォーマット

・rot_rdqの場合

【ラージモデル／スモールモデル】

```
MOV         bnk0_b,#4
MOV         bnk0_e,#tskpri
CALLT       [40H]
            :
```

Cレジスタ = リターン・パラメータ

・irot_rdqの場合

【ラージモデル／スモールモデル】

```
MOV         E,#tskpri
CALLT       [44H]
            :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

・rot_rdqの場合

```
ret = rot_rdq (tskpri) ;
char  tskpri;
```

・irot_rdqの場合

```
ret = irot_rdq (tskpri) ;
char  tskpri;
```

Get Task Status

tsk_sts

機　能

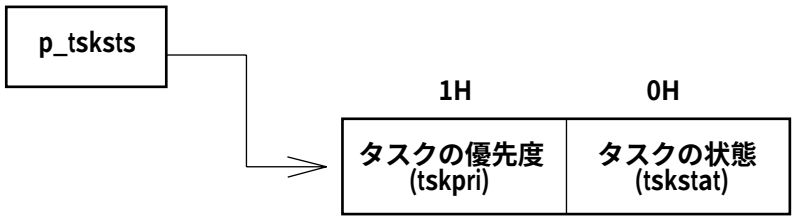
タスクの状態を見る。

解　説

tskidで示された対象タスクの各種の状態を参照し，対象タスクの現在の優先度“ tskpri ”，タスク状態“ tskstat ”を，p_tskstsで指定されたアドレス内に設定します（下図参照）。

tskid = 0 で自タスクの指定になります。p_tskstsで領域はユーザが確保してください。

タスクの状態については，付録A　エラー・コード一覧表とタスクの状態一覧表を参照してください。



システムコールID番号

tsk_sts = 5

パラメータ

tskid （Task Identifier）	タスクID（TCB先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit
p_tsksts （Pointer of Task Status）	タスク状態を格納するエリアのアドレス ラージモデル時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了

アセンブラ・フォーマット

【ラージモデル】

```
MOV     bnk0_b,#5
MOVW    bnk0_up,#tskid
MOV     bnk0_d,#tskid
MOVW    bnk0_vp,#p_tsksts
MOV     bnk0_e,#p_tsksts
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#5
MOVW    bnk0_up,#tskid
MOVW    bnk0_vp,#p_tsksts
CALLT   [40H]
:
```

Cレジスタ = リターン・パラメータ

Cフォーマット

```
ret = tsk_sts (p_tsksts,tskid) ;
short   *p_tsksts;
char    *tskid;
```

Sleep Task

slp_tsk

機　能

タスクを待ち状態へ移行する。

解　説

自タスクをRUN状態からWAIT状態へ移行します。このWAIT状態は，本タスクを対象として発行された“wup_tsk（iwup_tsk）”または“ret_wup”システムコールにより解除されます。

システムコールID番号

```
slp_tsk = 6
```

パラメータ

なし

リターン・パラメータ

E_OK	正常終了

アセンブラ・フォーマット

【ラージモデル／スモールモデル】

```
MOV         bnk0_b,#6
CALLT       [40H]
            :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

```
ret = slp_tsk () ;
```

Wait for Wakeup Task

wai_tsk

機　能

タスクを一定時間待ち状態へ移行する。

解　説

自タスクを一定時間だけRUN状態からWAIT状態へ移行します。このWAIT状態は，本タスクを対象とした“wup_tsk（iwup_tsk）”または“ret_wup”システムコールの発行または，tmoutで指定した時間の経過により解除されます。tmoutの値が0の時は即時リターンとなり，リターン・パラメータはE_TMOUTが戻ります。また，tmoutで0を指定し，起床要求カウントが1以上の場合にはE_OKが戻ります（起床要求カウントの更新を行います）。

本システムコールを使用する場合は，使用タイマのベクタ・アドレスの設定が必要です。使用タイマは，5. 1 利用ハードウエアを参考にしてください。

待ち時間 = 設定割り込み時間×TMOUTの値

システムコールID番号

wai_tsk = 7

パラメータ

tmout （Timeout）	タイムアウト指定　16bit

リターン・パラメータ

E_OK	正常終了
E_TMOUT	時間経過

アセンブラ・フォーマット

【ラージモデル／スモールモデル】

```
MOV         bnk0_b,#7
MOVW        bnk0_vp,#tmout
CALLT       [40H]
            :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

```
ret = wai_tsk（tmout）；
unsigned short  tmout；
```

wup_tsk　Wakeup Task
iwup_tsk　Wakeup Task for Interrupt

機　能

タスクを起床し，WAIT状態からREADY状態へと移行する。

解　説

“ slp_tsk ”または“ wai_tsk ”システムコールの実行により待ち状態になっていたタスクを，WAIT状態からREADY状態へと移行します。対象タスクはtskidで示され，自タスクを指定することはできません。また，DORMANT状態のタスクに対しても本システムコールを発行することはできません。その場合，エラーコードとして“ E_DMT ”を返します。

対象タスクが“ slp_tsk ”または“ wai_tsk ”を実行しておらず，待ち状態でない場合には，この“ wup_tsk ”要求はキューイングされます。その場合，この“ wup_tsk ”要求は，後に，対象タスクが“ slp_tsk ”または“ wai_tsk ”を実行した時に有効となります。

なお，起床要求のキューイング数が上限値（0FFH）を越えようとした時点で，エラー（E_QOVR）を返します。

システムコールID番号

```
wup_tsk = 8
```

パラメータ

tskid （Task Identifier）	タスクID（TCB先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了
E_DMT	対象タスクがDORMANT状態である
E_QOVR	キューイングのオーバフロー

アセンブラ・フォーマット

・wup_tskの場合

【ラージモデル】

```
MOV     bnk0_b,#8
MOVW    bnk0_up,#tskid
MOV     bnk0_d,#tskid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#8
MOVW    bnk0_up,#tskid
CALLT   [40H]
:
```

Cレジスタ = リターン・パラメータ

・iwup_tskの場合

【ラージモデル】

```
MOVG      UUP,#tskid
CALLT     [46H]
          :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOVW      UP,#tskid
CALLT     [46H]
          :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

・wup_tskの場合

```
ret = wup_tsk (tskid) ;
char *tskid;
```

・iwup_tskの場合

```
ret = iwup_tsk (tskid) ;
char *tskid;
```

Cancel Wakeup Task

can_wup

機　能

タスクの起床要求を無効にする。

解　説

tskidで示された対象タスクにキューイングされていた起床要求回数を，指定されたポインタ領域p_wupcntに格納し，同時にその起床要求を全て解除します。p_wupcntの領域はユーザが確保してください。

tskid = 0 で，自タスクの指定になります。

システムコールID番号

```
can_wup = 9
```

パラメータ

tskid （Task Identifier）	タスクID（TCB先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit
p_wupcnt （Pointer of Wakeup Count）	キューイングされていた起床要求 カウントを格納するポインタ ラージモデル時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了
E_DMT	対象タスクがDORMANT状態である

アセンブラ・フォーマット

【ラージモデル】

```
MOV     bnk0_b,#9
MOVW    bnk0_up,#tskid
MOV     bnk0_d,#tskid
MOVW    bnk0_vp,#p_wupcnt
MOV     bnk0_e,#p_wupcnt
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#9
MOVW    bnk0_up,#tskid
MOVW    bnk0_vp,#p_wupcnt
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

```
ret = can_wup（p_wupcnt,tskid）；
char  *p_wupcnt；
char  *tskid；
```

4. 2　同期・通信

同期・通信関連には，以下に挙げるシステムコールがあります。

（1）set_flg　イベントフラグのセット
（2）iset_flg　〃　（割り込みハンドラ内で使用する場合）
（3）clr_flg　イベントフラグのクリア
（4）wai_flg　イベントフラグを待つ　（クリア無し）
（5）cwai_flg　〃　（クリア有り）
（6）pol_flg　イベントフラグを得る　（クリア無し）
（7）cpol_flg　〃　（クリア有り）
（8）sig_sem　セマフォに対する信号操作　（V命令）
（9）isig_sem　〃　（割り込みハンドラ内で使用する場合）
（10）wai_sem　セマフォに対する待ち操作　（P命令）
（11）preq_sem　セマフォ資源を得る
（12）snd_msg　メッセージの送信
（13）isnd_msg　〃　（割り込みハンドラ内で使用する場合）
（14）rcv_msg　メッセージの受信を待つ
（15）prcv_msg　メッセージの受信

set_flg
iset_flg

Set Eventflag

Set Eventflag for Interrupt

機　能

イベントフラグのセットを行う。

解　説

flgidで示されたイベントフラグが1にセットされます。“ wai_flg ”，“ cwai_flg ”によってそのイベントフラグを待っているタスクがあれば，そのタスクの待ちを解除してREADY状態へと移行します。

また，同一イベントフラグに対する複数タスクの待ちも可能です。複数タスクがイベントフラグを待っていた場合，一度の“ set_flg ”で複数タスクの待ちを解除できます。

システムコールID番号

```
set_flg = 10
```

パラメータ

flgid（Eventflag Identifier）	イベントフラグID（イベントフラグ先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了

アセンブラ・フォーマット

・set_flgの場合

【ラージモデル】

```
MOV     bnk0_b,#10
MOVW    bnk0_up,#flgid
MOV     bnk0_d,#flgid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#10
MOVW    bnk0_up,#flgid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

・iset_flgの場合

【ラージモデル】

```
MOVG      UUP,#flgid
CALLT     [48H]
          :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOVW      UP,#flgid
CALLT     [48H]
          :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

・set_flgの場合

```
ret = set_flg (flgid) ;
char  *flgid;
```

・iset_flgの場合

```
ret = iset_flg (flgid) ;
char  *flgid;
```

Clear Eventflag

clr_flg

機　能

イベントフラグのクリアを行う。

解　説

flgidで示されたイベントフラグをクリアします。“wai_flg”，“cwai_flg”によってそのイベントフラグを待っているタスクがあっても，そのタスクの待ちが解除になることはありません。

システムコールID番号

```
clr_flg = 11
```

パラメータ

flgid（Eventflag Identifier）	イベントフラグID（イベントフラグ先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了

アセンブラ・フォーマット

【ラージモデル】

```
MOV       bnk0_b,#11
MOVW      bnk0_up,#flgid
MOV       bnk0_d,#flgid
CALLT     [40H]
          :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV       bnk0_b,#11
MOVW      bnk0_up,#flgid
CALLT     [40H]
          :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

```
ret = clr_flg（flgid）；
char  *flgid；
```

Wait Eventflag

wai_flg

機　能

イベントフラグを待つ。（クリア無し）

解　説

flgidで示されるイベントフラグが1にセットされるのを待ちます。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でも，フラグの値はセットされたままです。

同一のイベントフラグを複数タスクが待つことも可能です。

システムコールID番号

```
wai_flg = 12
```

パラメータ

flgid（Eventflag Identifier）	イベントフラグID（イベントフラグ先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了

アセンブラ・フォーマット

【ラージモデル】

```
MOV       bnk0_b,#12
MOVW      bnk0_up,#flgid
MOV       bnk0_d,#flgid
CALLT     [40H]
          :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV       bnk0_b,#12
MOVW      bnk0_up,#flgid
CALLT     [40H]
          :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

```
ret = wai_flg（flgid）；
char  *flgid；
```

Wait and Clear Eventflag

cwai_flg

機　能

イベントフラグを待つ。（クリア有り）

解　説

flgidで示されるイベントフラグが1にセットされるのを待ちます。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。

複数のタスクがイベントフラグを待っている状態の時に，“ set_flg ”が発行されると，キューの先頭にキューインされているタスクから順に起床されます。

ただし，“ cwai_flg ”によりWAIT状態に遷移しているタスクより後のタスクについては，イベントフラグがクリアされてしまうために，起床することができません。

システムコールID番号

```
cwai_flg = 13
```

パラメータ

flgid（Eventflag Identifier）	イベントフラグID（イベントフラグ先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了

アセンブラ・フォーマット

【ラージモデル】

```
MOV     bnk0_b,#13
MOVW    bnk0_up,#flgid
MOV     bnk0_d,#flgid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#13
MOVW    bnk0_up,#flgid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

```
ret = cwai_flg（flgid）；
char  *flgid；
```

Poll Eventflag

pol_flg

機　能

イベントフラグを得る。（クリア無し）

解　説

flgidで示されたイベントフラグがセットされているかどうか調べます。

フラグがセットされていれば正常終了し，フラグがセットされていなければエラー・リターン（E_PLFAIL）します。

いずれの場合にも，イベントフラグの値は変化しません。

システムコールID番号

```
pol_flg = 14
```

パラメータ

flgid（Eventflag Identifier）	イベントフラグID（イベントフラグ先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了
E_PLFAIL	ポーリング失敗

アセンブラ・フォーマット

【ラージモデル】

```
MOV     bnk0_b,#14
MOVW    bnk0_up,#flgid
MOV     bnk0_d,#flgid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#14
MOVW    bnk0_up,#flgid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

```
ret = pol_flg（flgid）；
char  *flgid；
```

Poll and Clear Eventflag

cpol_flg

機　能

イベントフラグを得る。（クリア有り）

解　説

flgidで示されたイベントフラグがセットされているかどうか調べます。

フラグがセットされている場合はそのフラグをクリアして正常終了し，フラグがセットされていない場合はフラグの値はそのままにしてエラー・リターン（E_PLFAIL）します。

システムコールID番号

```
cpol_flg = 15
```

パラメータ

flgid（Eventflag Identifier）	イベントフラグID（イベントフラグ先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit	

リターン・パラメータ

E_OK	正常終了	- 8BIT
E_PLFAIL	ポーリング失敗	- 8BIT

アセンブラ・フォーマット

【ラージモデル】

```
MOV     bnk0_b,#15
MOVW    bnk0_up,#flgid
MOV     bnk0_d,#flgid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#15
MOVW    bnk0_up,#flgid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

```
ret = cpol_flg（flgid）；
char  *flgid；
```

sig_sem
isig_sem

Signal Semaphore

Signal Semaphore for Interrupt

機　能

セマフォに対する信号操作（V命令）を行う。

解　説

semidで示されたセマフォに対して待っているタスクがあれば，待ち行列の先頭のタスクをREADY状態にします。セマフォに対して待っているタスクがなければ，そのセマフォのカウント値を1だけ増やします。

なお，セマフォのカウント値が上限値（0FFH）を越えようとした時点で，エラーとして “E_QOVR”を返します。

システムコールID番号

```
sig_sem = 16
```

パラメータ

semid（Semaphore Identifier）	セマフォID（セマフォ先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了
E_QOVR	キューイングのオーバフロー

アセンブラ・フォーマット

・sig_semの場合

【ラージモデル】

```
MOV     bnk0_b,#16
MOVW    bnk0_up,#semid
MOV     bnk0_d,#semid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#16
MOVW    bnk0_up,#semid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

・isig_semの場合

【ラージモデル】

```
MOVG    UUP,#semid
CALLT   [4AH]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOVW    UP,#semid
CALLT   [4AH]
        :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

・sig_semの場合

```
ret = sig_sem (semid) ;
char  *semid;
```

・isig_semの場合

```
ret = isig_sem (semid) ;
char  *semid;
```

Wait on Semaphore

wai_sem

機　能

セマフォに対する待ち操作（P命令）を行う。

解　説

semidで示されたセマフォのカウント値が1以上であれば，セマフォのカウント値を1だけ減じて本システムコールの発行タスクに制御を戻します。セマフォのカウント値が0の場合はセマフォのカウント値は変更せず，本システムコールを発行したタスクをWAIT状態に移行します。

システムコールID番号

```
wai_sem = 17
```

パラメータ

semid（Semaphore Identifier）	セマフォID（セマフォ先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了

アセンブラ・フォーマット

【ラージモデル】

```
MOV     bnk0_b,#17
MOVW    bnk0_up,#semid
MOV     bnk0_d,#semid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#17
MOVW    bnk0_up,#semid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

```
ret = wai_sem（semid）；
char  *semid；
```

Poll and Request Semaphore

preq_sem

機　能

セマフォ資源を得る。

解　説

semidで示されたセマフォのカウント値が1以上であれば,セマフォのカウント値を1だけ減じて正常終了します。
セマフォのカウント値が0であれば，セマフォ値は変更せずにエラー・リターン（E_PLFAIL）します。

システムコールID番号

```
preq_sem = 18
```

パラメータ

semid（Semaphore Identifier）	セマフォID（セマフォ先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了
E_PLFAIL	ポーリング失敗

アセンブラ・フォーマット

【ラージモデル】

```
MOV     bnk0_b,#18
MOVW    bnk0_up,#semid
MOV     bnk0_d,#semid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#18
MOVW    bnk0_up,#semid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

```
ret = preq_sem（semid）；
char  *semid；
```

snd_msg
isnd_msg

Send Message to Mailbox

Send Message to Mailbox for Interrupt

機　能

メッセージを送信する。

解　説

mbxidで示されたメイルボックスに，pk_msgのアドレスに入っているメッセージを送信します。待ちタスクが存在した場合は，送信メッセージの先頭アドレスを返し，READY状態へ移行します。この場合，メイルボックスへのメッセージの送信は行いません。

メッセージを待つタスクが無い時に，“ snd_msg ”が発行されても，“ snd_msg ”発行タスクは待ち状態とはなりません。このシステムコールではメッセージをメイルボックスに入れるだけで，タスクはその先の実行を続けます。つまり，非同期のメッセージ送信を行います。

なお，送信メッセージ領域の先頭2バイトは，キューのリンク・エリアとして使用しますので，メッセージは3バイト目以降に格納してください。

システムコールID番号

```
snd_msg = 19
```

パラメータ

mbxid（Mailbox Identifier）	メイルボックスID（メイルボックス先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit
pk_msg（Message Packet）	送信メッセージの先頭アドレス ラージモデル時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了

アセンブラ・フォーマット

・snd_msgの場合

【ラージモデル】

```
MOV     bnk0_b,#19
MOVW    bnk0_up,#mbxid
MOV     bnk0_d,#mbxid
MOVW    bnk0_vp,#pk_msg
MOV     bnk0_e,#pk_msg
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#19
MOVW    bnk0_up,#mbxid
MOVW    bnk0_vp,#pk_msg
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

・isnd_msgの場合

【ラージモデル】

```
MOVG        UUP,#mbxid
MOVG        VVP,#pk_msg
CALLT       [4CH]
            :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOVW      UP,#mbxid
MOVW      VP,#pk_msg
CALLT     [4CH]
          :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

・snd_msgの場合

```
ret = snd_msg (mbxid,pk_msg) ;
char  *mbxid;
char  *pk_msg;
```

・isnd_msgの場合

```
ret = isnd_msg (mbxid,pk_msg) ;
char  *semid;
char  *pk_msg;
```

Receive Message from Mailbox

rcv_msg

機　能

メイルボックスからの受信を待つ。

解　説

mbxidで示されたメイルボックスからメッセージを受信し，受信したメッセージの先頭アドレスを，パラメータで指定されたアドレス内に設定します。また，そのメイルボックスにメッセージが到着していない場合には，本システムコール発行タスクをWAIT状態へ移行します。ppk_msg 領域はユーザが確保してください。

受信メッセージは，実データが格納されているメッセージ領域の先頭アドレスとなります。

システムコールID番号

```
rcv_msg = 20
```

パラメータ

mbxid（Mailbox Identifier）	メイルボックスID（メイルボックス先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit
ppk_msg（Pointer of Message Packet）	受信メッセージの先頭アドレスを格納するエリアのアドレス ラージモデル時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了

アセンブラ・フォーマット

【ラージモデル】

```
MOV     bnk0_b,#20
MOVW    bnk0_up,#mbxid
MOV     bnk0_d,#mbxid
MOVG    VVP,#ppk_msg
PUSH    VVP
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#20
MOVW    bnk0_up,#mbxid
MOVW    AX,#ppk_msg
PUSH    AX
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

```
ret = rcv_msg（ppk_msg,mbxid）；
char  **ppk_msg；
char  *mbxid；
```

Poll and Receive Message from Mailbox

prcv_msg

機　能

メッセージを受信する。

解　説

mbxidで示されたメイルボックスにメッセージが入っていればそのメッセージを受信し，受信したメッセージの先頭アドレスを，パラメータで指定されたアドレス内に設定します。この場合，本システムコールは正常終了します。一方，そのメイルボックスにまだメッセージが到着していない場合には，本システムコールはエラーとして “E_PLFAIL” を返します。ppk_msg 領域はユーザが確保してください。

受信メッセージは，実データが格納されているメッセージ領域の先頭アドレスとなります。

システムコールID番号

```
prcv_msg = 21
```

パラメータ

mbxid（Mailbox Identifier）	メイルボックスID（メイルボックス先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit
ppk_msg（Pointer of Message Packet）	受信メッセージの先頭アドレスを格納するエリアのアドレス ラージモデル時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了
E_PLFAIL	ポーリング失敗

アセンブラ・フォーマット

【ラージモデル】

```
MOV     bnk0_b,#21
MOVW    bnk0_up,#mbxid
MOV     bnk0_d,#mbxid
MOVW    bnk0_vp,#ppk_msg
MOV     bnk0_e,#ppk_msg
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#21
MOVW    bnk0_up,#mbxid
MOVW    bnk0_vp,#ppk_msg
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

```
ret = prcv_msg (ppk_msg,mbxid) ;
char  **ppk_msg;
char   *mbxid;
```

4. 3　メモリ管理

メモリ管理に関連するシステムコールは，以下のとおりです。

(1) pget_blk　　固定長メモリブロックの獲得
(2) rel_blk　　固定長メモリブロックの返却

Poll and Get Fixed-Length Memory Block

pget_blk

機　能

固定長メモリブロックを獲得する。

解　説

mplidで示されたメモリプールから，固定長メモリブロックを獲得します。獲得したメモリブロックは，タスク間通信用メッセージ・エリアとして利用することができます。

メモリブロックのサイズは，システム生成時に決定され固定です。

メモリブロックが獲得できる時には，メモリブロックの先頭アドレスがパラメータで指定されたアドレス内に設定されます。逆に，獲得できない時はエラー・コードとして“E_PLFAIL”が返却されます。どちらの場合でも，WAIT状態に遷移することはありません。p_blk領域はユーザが確保してください。

システムコールID番号

pget_blk = 22

パラメータ

mplid（Memory Pool Identifier）	メモリプールID（メモリプール先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit
p_blk（Pointer of Block Start Address）	メモリブロックの先頭アドレスを格納するエリアのアドレス ラージモデル時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了
E_PLFAIL	ポーリング失敗

アセンブラ・フォーマット

【ラージモデル】

```
MOV     bnk0_b,#22
MOVW    bnk0_up,#mplid
MOV     bnk0_d,#mplid
MOVW    bnk0_vp,#p_blk
MOV     bnk0_e,#p_blk
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#22
MOVW    bnk0_up,#mplid
MOVW    bnk0_vp,#p_blk
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

```
ret = pget_blk (p_blk,mplid) ;
char  **p_blk;
char   *mplid;
```

Release Fixed-Length Memory Block

rel_blk

機　能

固定長メモリブロックを返却する。

解　説

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

システムコールID番号

```
rel_blk = 23
```

パラメータ

mplid（Memory Pool Identifier）	メモリプールID（メモリプール先頭アドレス） ラージモデル時　24bit　スモールモデル時　16bit
blk（Block Start Address）	メモリブロックの先頭アドレス ラージモデル時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了

アセンブラ・フォーマット

【ラージモデル】

```
MOV     bnk0_b,#23
MOVW    bnk0_up,#mplid
MOV     bnk0_d,#mplid
MOVW    bnk0_vp,#blk
MOV     bnk0_e,#blk
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#23
MOVW    bnk0_up,#mplid
MOVW    bnk0_vp,#blk
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

```
ret = rel_blk（mplid,blk）；
char  *mplid；
char  *blk；
```

4. 4　割り込み処理

割り込み処理に関連するシステムコールは，以下のとおりです。

（1）ret_int　　割り込みハンドラの終了
（2）ret_wup　　割り込み処理復帰とタスクの起床

Return from Interrupt Handler

ret_int

機　能

割り込みハンドラから復帰する。

解　説

割り込みハンドラからのリターンの際に発行するシステムコールです。

割り込みハンドラ内で，“ixxx_yyy”システムコールを発行してもディスパッチは起こらず，このシステムコールが発行されて割り込みハンドラを抜けるまで，ディスパッチは遅延させられます。

システムコールID番号

なし

パラメータ

なし

リターン・パラメータ

なし

システムコールを発行した割り込みハンドラには戻りません。

アセンブラ・フォーマット

【ラージモデル／スモールモデル】

```
BR      !ret_int
```

本システムコールは他のシステムコールと違い，ジャンプ命令により分岐します。

注意　割り込みハンドラから本システムコールを発行する際は，必ず割り込む前のレジスタ値をスタック内にセーブしてください。レジスタ値をスタック内にセーブする順番は図4－2に示す順番にしてください。
また，割り込みハンドラがレジスタ・バンクを占有している場合には，割り込む前に実行中だったタスクのレジスタ・バンクに切り換えてから本システムコールの呼び出しを行ってください。

Cフォーマット

```
ret_int () ;
```

※　本システムコールはCコンパイラ“CC78K4”のバージョン2.00以上に対応していますので，バージョン2.00以前のCコンパイラでは使用できません。バージョン2.00以上のCコンパイラにおいて，“#pragma△rtos_interrupt”を指定してください（**付録**B　C**言語での記述に関して**参照）。

Return and Wakeup Task

ret_wup

機　能

割り込み処理復帰とタスク起床を行う。

解　説

割り込みハンドラからリターンし，同時にtskidで示される割り込み処理タスク（“ slp_tsk ”，“ wai_tsk ”で待っているタスク）を起床させます。対象タスクが“ slp_tsk ”，“ wai_tsk ”を実行していない場合には，要求がキューイングされます。

割り込みハンドラ内で“ ixxx_yyy ”システムコールを発行してもディスパッチは起こらず，このシステムコールが発行されて割り込みハンドラを抜けるまで，ディスパッチは遅延させられます。

システムコールID番号

なし

パラメータ

tskid（Task Identifier）	タスクID（TCB先頭アドレス） ラージタイプ時　24bit　スモールモデル時　16bit

リターン・パラメータ

なし

システムコールを発行した割り込みハンドラには戻りません。

アセンブラ・フォーマット

【ラージモデル／スモールモデル】

```
MOVG        UUP,#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，ジャンプ命令により分岐します。

注意　割り込みハンドラから本システムコールを発行する際は，必ず割り込む前のレジスタ値をスタック内にセーブしてください。レジスタ値をスタック内にセーブする順番は図4－2に示す順番にしてください。

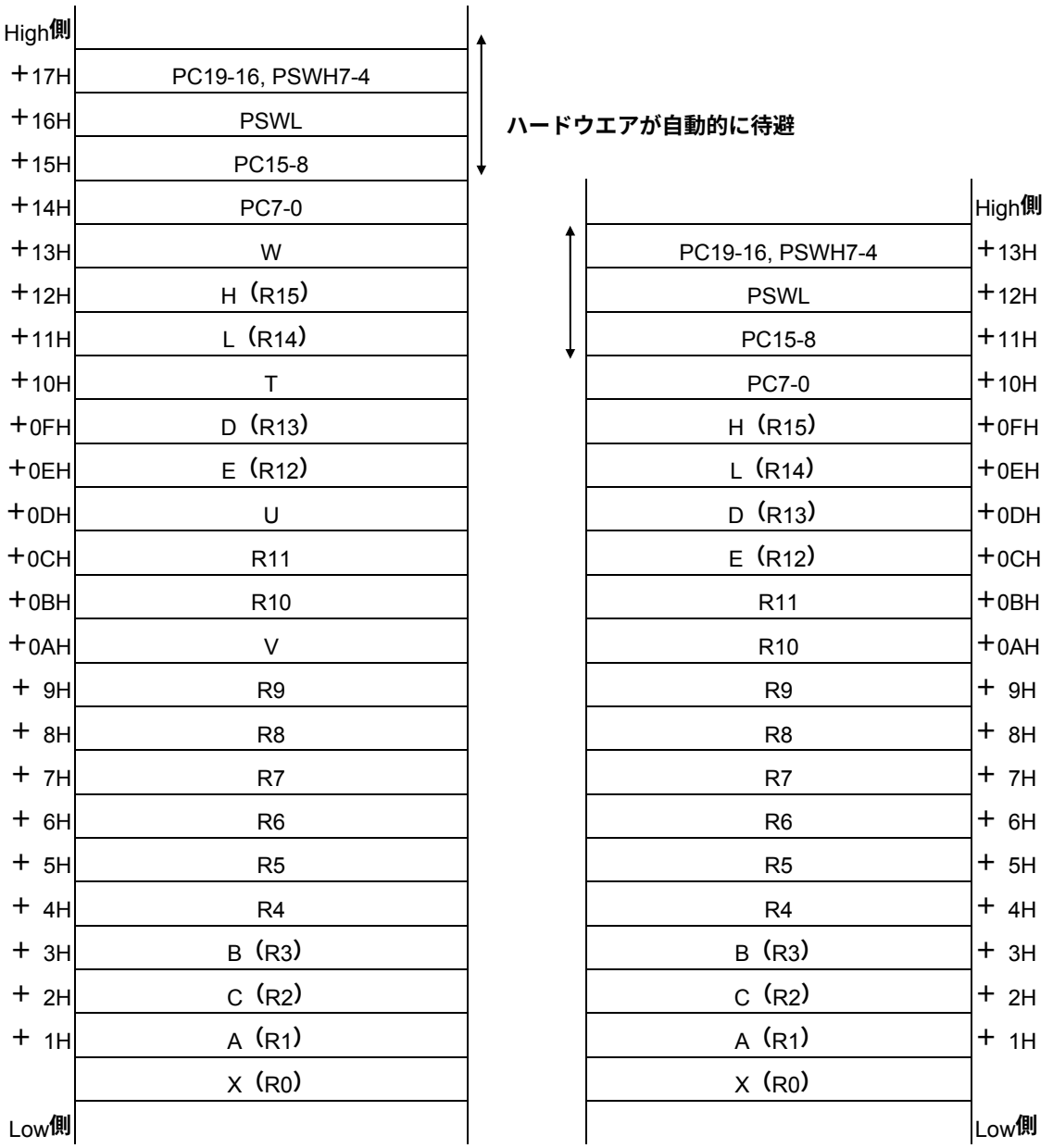
また，割り込みハンドラがレジスタ・バンクを占有している場合には，割り込む前に実行中だったタスクのレジスタ・バンクに切り換えてから本システムコールの呼び出しを行ってください。

Cフォーマット

```
ret_wup（tskid）；
char　*tskid；
```

※ **本システムコールはCコンパイラ“** CC78K4 **”のバージョン**2.00**以上に対応していますので，バージョン**2.00**以前のCコンパイラでは使用できません。バージョン**2.00**以上のCコンパイラにおいて，“** #pragma△rtos_interrupt **”を指定してください（付録**B　C**言語での記述に関して参照）。**

図4－2　ret_int,ret_wup**を呼び出す際のスタックの状態**



4.5　バージョン管理

バージョン管理に関連するシステムコールは，以下のとおりです。

（1）get_ver　　RX78K4のバージョン番号を得る。

Get Version No.

get_ver

機　能

RX78K4のバージョン番号を得る。

解　説

バージョン管理ブロックの先頭アドレスを，パラメータで指定されたアドレス内に設定します。なお，バージョン管理ブロックには，以下の情報が格納されています。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	メーカー（NEC Electronics）
id	2 byte	0x0	形式番号［ラージモデル］
spver	2 byte	0x5201	TRON仕様書バージョン（μITRON　Ver2.01）
prver	2 byte	0x0120	製品のバージョン（RX78K4　Ver1.20）
prno	2 byte×4	0x0	製品管理番号
cpu	2 byte	0x0d14	CPU情報（NEC Electronics　78K4シリーズ）
var	2 byte	0x0	バリエーション記述子

システムコールID番号

```
get_ver = 25
```

パラメータ

pk_ver（Packet of Version Numbers）	バージョン管理ブロックのポインタを格納するエリアのアドレス ラージタイプ時　24bit　スモールモデル時　16bit

リターン・パラメータ

E_OK	正常終了

アセンブラ・フォーマット

【ラージモデル】

```
MOV     bnk0_b,#25
MOVW    bnk0_vp,#pk_ver
MOV     bnk0_e,#pk_ver
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#25
MOVW    bnk0_vp,#pk_ver
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

```
ret = get_ver（pk_ver）；
char  **pk_ver；
```

4. 6　時間管理

時間管理に関連するシステムコールは，以下のとおりです。

（1）act_cyc　　周期ハンドラの活性制御を行う。

　　iact_cyc　　〃　　（割り込みハンドラ内で使用する場合）

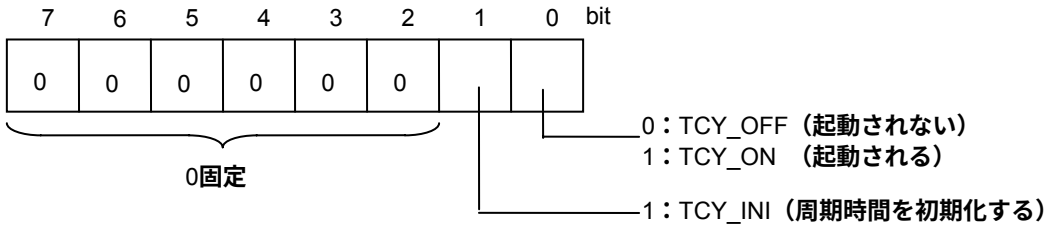
act_cyc iact_cyc	Activate Cyclic Handler Activate Cyclic Handler for Interrupt

機　能

周期ハンドラの活性制御を行う。

解　説

cyhidで示された周期ハンドラの活性状態をcyhactで指定された状態に変更します。cyhactの具体的な指定方法は次のようになります。



cyhact: =（TCY_OFF||TCY_ON）| [TCY_INI]

cyhact = TCY_OFFは周期ハンドラの一時的な中断を意味し，この間はハンドラが起動されません。

cyhact = TCY_ONは周期の経過とは独立に活性状態をONにします。活性状態がOFFの間も指定周期のカウントを行っているため，“ act_cyc（iact_cyc） ”システムコールを実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ではありません。

cyhact =（TCY_ON|TCY_INI）は活性状態をONにするのと同時に周期ハンドラのカウンタをクリアします。したがって，“ act_cyc（iact_cyc） ”システムコールを実行してからちょうど指定周期経過した後に，最初のハンドラが起動が起こります。

周期ハンドラは，動的に生成および削除することはできません。コンフィギュレータで周期ハンドラを登録します。

システムコールID番号

act_cyc = 26

パラメータ

cyhid（Cyclic Handler Identifier）	周期ハンドラのID（HCB先頭アドレス） ラージタイプ時　24bit　スモールモデル時　16bit
cyhact（Cyclic Handler Activation）	周期ハンドラ活性状態　8bit

リターン・パラメータ

E_OK	正常終了

アセンブラ・フォーマット

・act_cycの場合

【ラージモデル】

```
MOV     bnk0_b,#26
MOV     bnk0_e,#cyhact
MOVW    bnk0_up,#cyhid
MOV     bnk0_d,#tskpri
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     bnk0_b,#26
MOV     bnk0_e,#cyhact
MOVW    bnk0_up,#cyhid
CALLT   [40H]
        :
```

Cレジスタ = リターン・パラメータ

・iact_cycの場合

【ラージモデル】

```
MOV     E,#cyhact
MOVG    UUP,#cyhid
CALLT   [4EH]
        :
```

Cレジスタ = リターン・パラメータ

【スモールモデル】

```
MOV     E,#cyhact
MOVW    UP,#cyhid
CALLT   [4EH]
        :
```

Cレジスタ = リターン・パラメータ

Cフォーマット

・act_cycの場合

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

・iact_cycの場合

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

第5章　RX78K4のリセット動作

RX78K4上のシステムは，78K4シリーズの内部ROMまたは外部メモリに，RX78K4と共に格納されているシステム初期化用リセット・ルーチンを実行することにより初期化されます。

デバイスのリセット後，システムはアドレス0000Hに設定されているアドレスに分岐しますので，ユーザの初期設定ルーチンのアドレスを設定しておいてください。ユーザの初期設定ルーチンでは，周辺ハードウエアの初期化等，必要な処理を行ってください。その後，コンフィギュレータにより生成された初期化情報テーブルよりカーネルのロケーション・アドレスを取り出し，システム初期化用リセット・ルーチンに分岐してください。分岐する際は，初期化情報テーブルの先頭アドレスをTDEレジスタに設定してください。

図5－1　システム初期化用リセット・ルーチン分岐プログラムにシステム初期化用リセット・ルーチンに分岐する際のサンプル・プログラムを示します。

システム初期化用リセット・ルーチンは，初期化情報テーブルに従い，リザーブ・エリアやオブジェクトの初期化等を行います。初期化終了後に，RX78K4のディスパッチャへ制御が移行します。ディスパッチャは，システム初期化用リセット・ルーチンによって初期設定されたレディ・キューからタスクを選び出しRUN状態にします。この時，選択の対象になるタスクをイニシャル・タスクといいます。

図5－1　システム初期化用リセット・ルーチン分岐プログラム（ラージモデル）

```
            NAME        RES_PROG
;
            EXTRN       SYS_INF             ;初期化情報テーブル
;
VCTTBL0     CSEG  AT    0000H
            DW          RES_RTN             ;リセット時ベクタ・アドレス設定
VCTTBL1     CSEG  AT    0018H
            DW          ?TMDSP              ;タイマ処理用ベクタ・アドレス設定


RESRTN      CSEG
RES_RTN:
            LOCATION 0FH
            （RAMの初期化，周辺ハードウエアの初期化など）
            MOVG        TDE, #SYS_INF       ;初期化情報テーブルの先頭アドレスをTDEreg.に設定
            MOVW        AX, [TDE]           ;カーネル・ロケーション・アドレスをAXreg.に設定
            BR          AX                  ;RX78K4のシステム初期化用リセット・ルーチンに分岐
;
            END
```

5. 1　利用ハードウエア

RX78K4で利用するハードウエアには，“ wai_tsk ”，“ act_cyc（iact_cyc） ”に関連して使用するタイマ／カウンタ・ユニットがあります。使用するタイマは，ディフォールトで表5－1のようになっています。

RX78K4ではタイマ／カウンタ・ユニットの初期化は行いませんので，ユーザの初期設定ルーチンで初期化を行ってください。

また，“ wai_tsk ”，“ act_cyc（iact_cyc） ”を使用しない場合は，タイマ／カウンタ・ユニットは使用しません。

表5－1　各CPUの使用タイマ

CPU名称	使用タイマ	コンペア・レジスタ	割り込みベクタ
μPD784021, 784025 μD784026, 78P4026	TM3 （16ビット・タイマ3）	CR30	INTC30

TM3以外のタイマを使用する場合は，使用するタイマのベクタ・テーブル・アドレスにタイマ・ディスパッチャのスタート・アドレス“ ?tmdsp ”を設定してください。

また，タイマ処理の割り込みレベルを3に設定し，割り込みハンドラとの多重割り込みを許可する場合は，本OSの動作を保証できません。したがって，タイマ処理の割り込みレベルを3以外に設定するか，または，割り込みレベル3にする場合は多重割り込みを禁止してください。

付録A　エラー・コード一覧表とタスクの状態一覧表

・エラーコード一覧表

エラー・コード	値 （8ビット，char型）	内　容
E_OK	0	正常終了。
E_NODMT	1	システムコール“sta_tsk”発行時，対象タスクがDORMANT状態ではない。
E_DMT	2	システムコール“wup_tsk（iwup_tsk）”，“ter_tsk”，“chg_pri”発行時，対象タスクがDORMANT状態である。
E_QOVR	3	キューイングのオーバ・フロー。
E_TMOUT	4	システムコール“wai_tsk”発行時，時間経過により終了した場合。
E_PLFAIL	5	ポーリング失敗。

・タスクの状態一覧表

値	状態
0x0	休止（DORMANT）状態
0x1	実行（RUN）状態または実行可能（READY）状態
0x2	待ち状態（timeout）
0x3	待ち状態（sleep）
0x4	待ち状態（eventflag）クリアなし
0x5	待ち状態（eventflag）クリアあり
0x6	待ち状態（semaphore）
0x7	待ち状態（message）

付録B　C言語での記述に関して

Cコンパイラ“CC78K4”のバージョン2.00以上では，タスク，割り込みハンドラをサポートしています。以下にタスク，割り込みハンドラをC言語記述した例を示します。

・タスク

記述形式

```
#pragma rtos_task ［タスク関数名］
```

例）

```
#pragma rtos_task sample
extern void slp_tsk（void）；          /* slp_tskの外部参照宣言 */
void sample ()
{
        slp_tsk () ；
        ext_tsk () ；
}
```

※“ext_tsk”システムコールに関しては，外部参照宣言はいりません。

・割り込みハンドラ

記述形式

```
#pragma rtos_interrupt ［（割り込み要求名）（割り込みハンドラ名）{スタック切り替え指定］
※スタック切り替え指定：sp = 配列名［＋オフセット位置］
```

例）

```
#pragma rtos_interrupt INTP0 sample_h sp = int_stack+10
unsigned char int_stack[10]；          /* 割り込み用スタック領域確保 */
extern char iset_flg（char*）；        /* iset_flgの外部参照宣言 */
extern char flag_id1；                 /* イベントフラグIDの外部参照宣言 */
sample_h ()
{
        iset_flg（&flag_id1）；
        ret_int () ；
}
```

※“ret_int”，”ret_wup”システムコールに関しては，外部参照宣言はいりません。

詳しくは，CC78K4　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。

付録C　システム・コール一覧表

システム・コール名	機能概要	エントリNo.
sta_tsk	タスクを起動し、休止状態から実行可能状態へ遷移させます	0
ext_tsk	自タスクを正常終了し、実状態から休止状態へ遷移させます	1
ter_tsk	他タスクを強制的に異常終了させ、休止状態へ遷移させます	2
chg_pri	タスク優先度を変更します	3
ichg_pri	〃　［割り込みハンドラから発行する］	−
rot_rdq	指定優先度のレディ・キューを回転します	4
irot_rdq	〃　［割り込みハンドラから発行する］	−
tsk_sts	タスクの状態を見ます	5
slp_tsk	タスクを待ち状態へ遷移させます	6
wai_tsk	タスクを一定時間待ち状態へ遷移させます	7
wup_tsk	タスクを起床し、待ち状態から実行可能状態へ遷移させます	8
iwup_tsk	〃　［割り込みハンドラから発行する］	−
can_wup	タスクの起床要求を無効にします	9
set_flg	イベントフラグのセットを行います	10
iset_flg	〃　［割り込みハンドラから発行する］	−
clr_flg	イベントフラグのクリアを行います	11
wai_flg	イベントフラグのセットを待ちます（クリア無し）	12
cwai_flg	〃　（クリア有り）	13
pol_flg	イベントフラグを得ます　（クリア無し）	14
cpol_flg	〃　（クリア有り）	15
sig_sem	セマフォに対する信号操作（V命令）を行います	16
isig_sem	〃　［割り込みハンドラから発行する］	−
wai_sem	セマフォに対する待ち操作（P命令）を行います	17
preq_sem	セマフォ資源を得ます	18
snd_msg	メッセージを送信します	19
isnd_msg	〃　［割り込みハンドラから発行する］	−
rcv_msg	メイルボックスからの受信を待ちます	20
prcv_msg	メッセージを受信します	21
pget_blk	固定長メモリブロックを獲得します	22
rel_blk	固定長メモリブロックを返却します	23
ret_int	割り込みハンドラから復帰します	−
ret_wup	割り込みハンドラからの復帰とタスク起床を行います	−
get_ver	RX78K4のバージョン番号を得ます	25
act_cyc	周期ハンドラの活性制御を行います	26
iact_cyc	〃　［割り込みハンドラから発行する］	−

付録D　周期ハンドラの記述例

周期ハンドラを作成する場合は，次の点に注意してください。

①周期ハンドラは1 Mバイトのメモリ空間にマッピングできます（ラージモデルのみ）。

②周期ハンドラはタイマ・ディスパッチ・ルーチンから “ br　rg ” 命令で呼び出されます。
このため，割り込みハンドラと同様な扱いになりますので，発行できるシステムコールは，“ ixxx_xxx ”システムコールだけです。ただし，周期ハンドラから周期ハンドラを対象とした“ ixxx_xxx ”システムコールの発行はできません。

③周期ハンドラの処理終了後は，必ずタイマ・ディスパッチ・ルーチン “ ?tm_ret ” へ戻ってください。
周期ハンドラの処理は，次のように記述してください。

```
        public          [周期ハンドラ・アドレス]
        extrn         ?tm_ret
[周期ハンドラ･アドレス] :
              [周期ハンドラの処理]
        br            !?tm_ret
```

④周期ハンドラでレジスタ・バンクを切り換える場合は，元のレジスタ・バンクへ戻ってからタイマ・ディスパッチ・ルーチンへ戻ってください。

⑤タイマ・ディスパッチ・ルーチンでは，割り込んだ時に “ rg7, rg6, rg5, rp1, rp0 ” レジスタをスタックに格納します。そのため，周期ハンドラで “ rg4, rp3, rp2 ” レジスタを使用する場合は，タイマ・ディスパッチ・ルーチンに戻る際に元の値に戻してください。

```
        public   cyc_ptr
        extrn    ?tm_ret

cyc_ptr:
        push     psw                ← ④
        sel      rb6

        push     rp2                ← ⑤

          周期ハンドラの処理

        pop      rp2                ← ⑤
        pop      psw                ← ④
        br       !?tm_ret           ← ③
```

付録E　メモリ容量の見積もり方法（ラージモデル）

RX78K4では，オブジェクト（TCB，イベントフラグ，セマフォ，メイルボックス，メモリプール，周期ハンドラ）は任意の領域に設定できます。ただし，ユーザが設定した領域の先頭アドレスから，20バイトの領域については，OSの管理領域となります。それに続いて，各オブジェクト管理領域を確保します。それ以降に，12～64バイトのレディ・キュー領域を確保します。レディ・キューの領域は，タスクの指定優先度数により可変となります。各オブジェクトの生成数は最大255です。

下記に，RX78K4を用いたシステムのオブジェクト・サイズ（データ部）の算出方法を示します。

例）　・生成するオブジェクトの数を次のようにします。

タスク　：10個　　イベントフラグ：5個

セマフォ　：4個　　メイルボックス：3個

周期ハンドラ　：2個

メモリプール　：2個（メモリブロック数：5個，1メモリブロック･サイズ：16バイト）

（メモリブロック数：10個，1メモリブロック･サイズ：32バイト）

・タスクの優先度は，10レベルまで使用します。

・OS管理領域の先頭アドレス（?objhead）を，0FF700Hとします。

使用RAMサイズ　OS管理テーブル　レディキュー　TCB　イベントフラグ　セマフォ　メイルボックス　周期ハンドラ

↓　↓　↓　↓　↓　↓　↓　↓

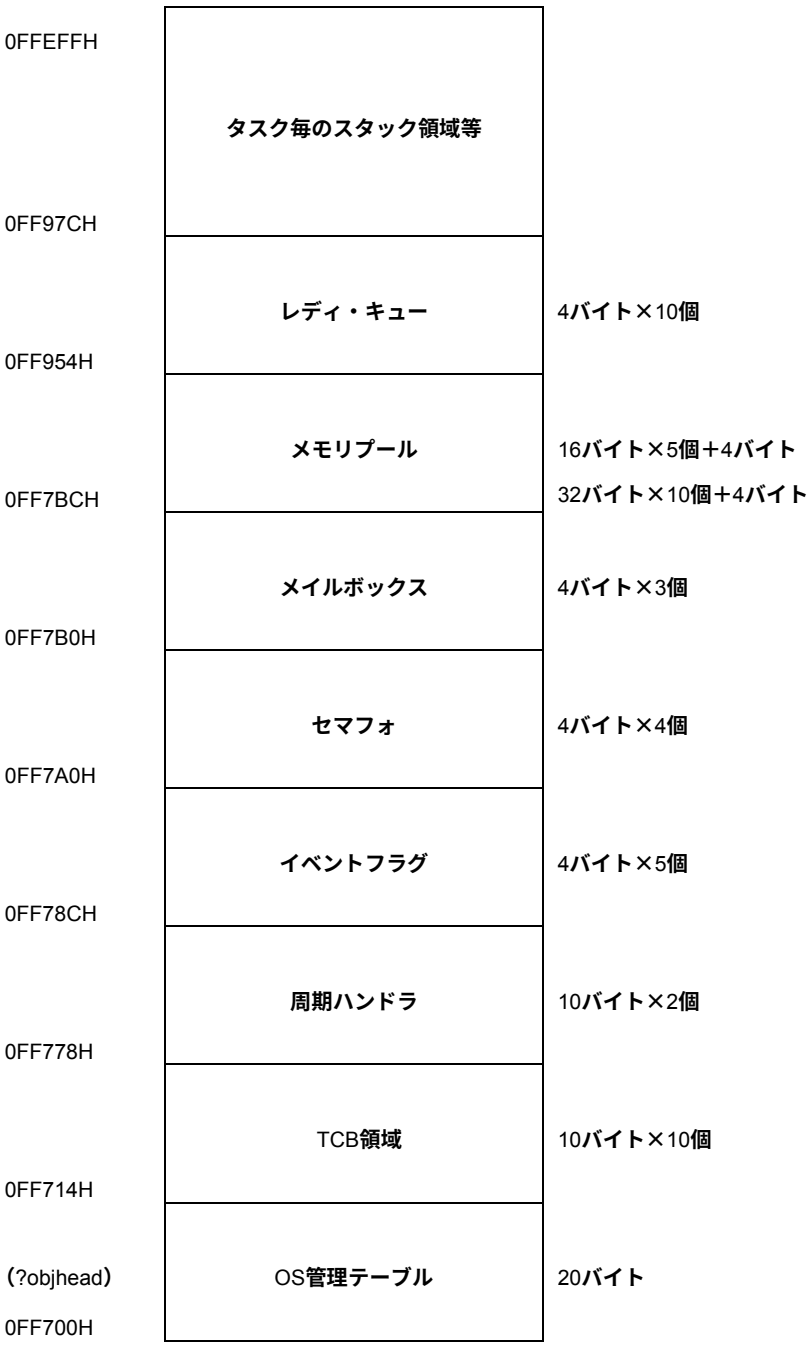
228（バイト）＝ 20 ＋ 4×10 ＋ 10×10 ＋ 4×5 ＋ 4×4 ＋ 4×3 ＋ 10×2

使用RAMサイズ　メモリプール

↓　↓

408（バイト）＝ 4＋16×5＋4＋32×10

合計　228 ＋ 408 ＝ 636（バイト）


0FFEFFH
タスク毎のスタック領域等
0FF97CH
レディ・キュー
4バイト×10個
0FF954H
メモリプール
16バイト×5個＋4バイト
32バイト×10個＋4バイト
0FF7BCH
メイルボックス
4バイト×3個
0FF7B0H
セマフォ
4バイト×4個
0FF7A0H
イベントフラグ
4バイト×5個
0FF78CH
周期ハンドラ
10バイト×2個
0FF778H
TCB領域
10バイト×10個
0FF714H
（?objhead）
OS管理テーブル
20バイト
0FF700H

付録F　メモリ容量の見積もり方法（スモールモデル）

RX78K4では，オブジェクト（TCB，イベントフラグ，セマフォ，メイルボックス，メモリプール，周期ハンドラ）は内部RAM領域内の“0F700H～0F7FFH”の256バイト内に生成できます。ただし，0F700Hから，14バイトの領域については，OSの管理領域となります。それに続いて，各オブジェクト管理領域を確保します。それ以降に，6～32バイトのレディ・キュー領域を確保します。レディ・キューの領域は，タスクの指定優先度数により可変となります。各オブジェクトの生成数は最大255です。

下記に，RX78K4を用いたシステムのオブジェクト・サイズ（データ部）の算出方法を示します。

例）　・生成するオブジェクトの数を次のようにします。

タスク　：10個　　イベントフラグ：5個

セマフォ　：4個　　メイルボックス：3個

周期ハンドラ　：2個

メモリプール　：2個（メモリブロック数：5個，1メモリブロック･サイズ：10バイト）

（メモリブロック数：10個，1メモリブロック･サイズ：6バイト）

・タスクの優先度は，10レベルまで使用します。

・OS管理領域の先頭アドレス（?objhead）は，0F700Hです。

使用RAMサイズ　OS管理テーブル　レディキュー　TCB　イベントフラグ　セマフォ　メイルボックス　周期ハンドラ

↓　↓　↓　↓　↓　↓　↓　↓

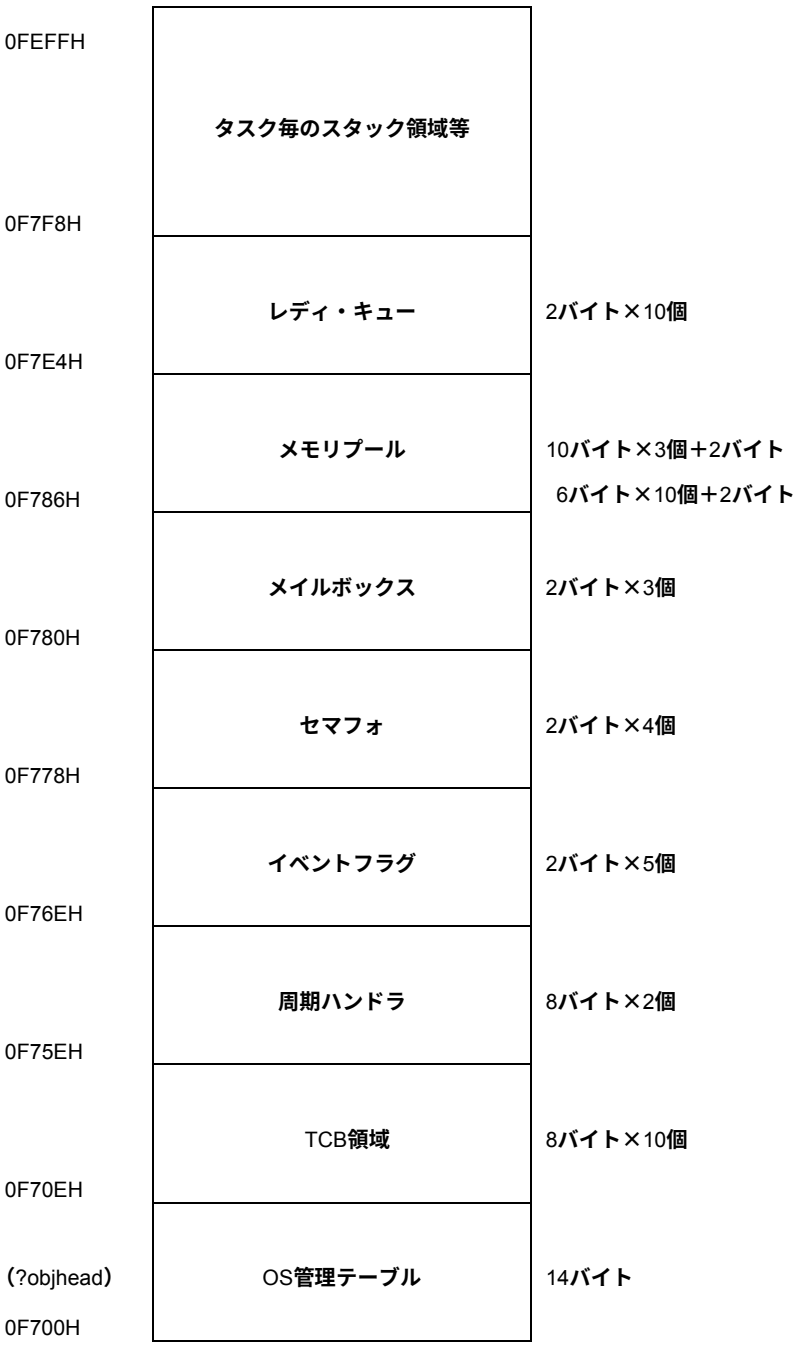
154（バイト）＝ 14 ＋ 2×10 ＋ 8×10 ＋ 2×5 ＋ 2×4 ＋ 2×3 ＋ 8×2

使用RAMサイズ　メモリプール

↓　↓

94（バイト）＝ 2＋10×3＋2＋6×10

合計　154 ＋ 94 ＝ 248（バイト）


0FEFFH
タスク毎のスタック領域等
0F7F8H
レディ・キュー
2バイト×10個
0F7E4H
メモリプール
10バイト×3個＋2バイト
6バイト×10個＋2バイト
0F786H
メイルボックス
2バイト×3個
0F780H
セマフォ
2バイト×4個
0F778H
イベントフラグ
2バイト×5個
0F76EH
周期ハンドラ
8バイト×2個
0F75EH
TCB領域
8バイト×10個
0F70EH
（?objhead）
OS管理テーブル
14バイト
0F700H

付録G　スタンバイ機能について

RX78K4は選択タスクがなくなるとCPUをディフォールトでHALT状態にします。しかし，HALTモードではなく，STOPモード，IDLEモードにしたいシステムもあります。このような要求に対応するために，RX78K4は，スタンバイモード処理をユーザオウンとすることにより，これらの要求に対応しています。以下にユーザオウン部の提供サンプル“ nuc_idle.asm ”を示します。

サンプル“ nuc_idle.asm ”

```
public   @nuc_idl
         cseg     base
@nuc_idl:
         mov      x,stbc
         and      x,#0f0h
         shr      x,1
         mov      a,#0
         movw     bc,#hlt_tbl
         addw     bc,ax
         br       bc
;
hlt_tbl:
         ei
         mov      stbc,#00000001b          ; set halt mode (fxx/2)
         ret
;
         nop
         nop
;
         ei
         mov      stbc,#00010001b          ; set halt mode (fxx/4)
         ret
;
         nop
         nop
;
         ei
         mov      stbc,#00100001b          ; set halt mode (fxx/8)
         ret
;
         nop
         nop
;
         ei
         mov      stbc,#00110001b          ; set halt mode (fxx/16)
         ret
;
         end
```

※　上記はサンプルで提供しているファイルです。このファイルは，ユーザシステムで使用する内部システム・クロックを変更してもRX78K4が対応できるようにしたルーチンです。

次に示すサンプルは，ユーザシステムで使用する内部システム・クロックが固定の場合の変更例を示します。

サンプル1

```
public   @nuc_idl
         cseg     base
@nuc_idl:
         ei
         mov      stbc,#00000001b            ; set halt mode (fxx/2)
         ret
;
         end
```

※　上記のサンプルは内部システム・クロックが固定の場合のサンプルです。ただし，シンボル名“ @nuc_idl ”は変更しないでください。

RX78K4のディフォールトのスタンバイモードはHALTです。モードを変更する際は,以下を参考にしてください。

・HALTモードをSTOPモードに変更する場合

スタンバイ・コントロール・レジスタ（STBC）を変更する

変更前		変更後
`mov stbc,#00000001b`	→	`mov stbc,#00000010b`

・HALTモードをIDLEモードに変更する場合

スタンバイ・コントロール・レジスタ（STBC）を変更する

変更前		変更後
`mov stbc,#00000001b`	→	`mov stbc,#00000011b`

スタンバイ・コントロール・レジスタ（STBC）については，デバイスのユーザーズ・マニュアルを参照してください。

付録H　システムコール別最大使用スタック・サイズ

システムコールの処理で使用するスタック領域は，システムコールを発行したタスクのスタック領域を使用します。したがって，各タスクのスタック・サイズは，ユーザ・タスクで使用するスタック・サイズとシステムコールの最大スタック・サイズを考えて決定してください。

割り込みハンドラとタイマ処理で使用するスタック領域は，任意のアドレスに設定できます。ただし，割り込みが入る際には，タスクのスタック領域を4バイト使用します。

アセンブリ言語で記述した場合の各システムコールの処理における最大スタック・サイズは，次の通りです。

システムコール名	最大使用スタック・サイズ（バイト）			
	ラージモデル		スモールモデル	
	使用レジスタバンク		使用レジスタバンク	
	1	2〜7	1	2〜7
sta_tsk	28	8	24	8
ext_tsk	28	8	24	8
ter_tsk	28	8	24	8
chg_pri	28	8	24	8
ichg_pri	22	22	19	19
rot_rdq	28	8	24	8
irot_rdq	19	19	14	14
tsk_sts	28	8	24	8
slp_tsk	28	8	24	8
wai_tsk	28	8	24	8
wup_tsk	28	8	24	8
iwup_tsk	18	18	16	16
can_wup	28	8	24	8
set_flg	28	8	24	8
iset_flg	23	23	17	17
clr_flg	28	8	24	8
wai_flg	28	8	24	8
cwai_flg	28	8	24	8
pol_flg	28	8	24	8
cpol_flg	28	8	24	8
sig_sem	28	8	24	8
isig_sem	21	21	16	16
wai_sem	28	8	24	8
preq_sem	28	8	24	8
snd_msg	28	8	24	8
isnd_msg	23	23	20	20
rcv_msg	31	11	26	10
prcv_msg	28	8	24	8
pget_blk	28	8	24	8
rel_blk	28	8	24	8
ret_int	24	24	20	20
ret_wup	24	24	20	20
get_ver	28	8	24	8
act_cyc	28	8	24	8
iact_cyc	10	10	9	9
タイマ処理	24	17	20	14

＊1　割り込み時に退避する4バイト（PC, PSW）を含みます。

付録I　シンボル名の制限事項について

RX78K4では，以下のシンボルをパブリック・シンボルとして使用しています。
したがって，タスクや割り込みハンドラで同名のシンボルを使用しないようにしてください。

?objhead
brproc
?sysrt
?qin
?qin1
?qout
?qout1
?cancl
?cnsav
?itdsp
?tmdsp
?tkdsp
?tm_ret
x_serial
bittbl
ent_tbl
sys_inf
rdyqtp
az_arg
wtcbh
wtcbt
runtcb
timnst
wqlnkp
retdata
pribit
cnftblp
tranos
bittbl
end_intsys
?db_tkds
end_itdsp
?db_itds
@nuc_idl
システムコール名

〔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。