HITACHI

ソフトウェアマニュアル

CPMS概説 &マクロ仕様

S10V
Programmable Controller

SVJ-3-201(A)

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。
なお、不明な場合は、弊社担当営業にお問合わせください。

２００３年　４月　（第１版）　ＳＶＪ－３－２０１（Ａ）

● このマニュアルの一部、または全部を無断で転写したり複写することは、固くお断りいたします。
● このマニュアルの内容を、改良のため予告なしに変更することがあります。

All Rights Reserved, Copyright © 2003, Hitachi, Ltd.

IC-NS-IK (FL-MW20, AI8.0)

⚠ 安全上のご注意

- システムの構築やプログラムの作成などは、このマニュアルの記載内容をよく読み、書かれている指示や注意を十分理解してから行ってください。誤操作により、システムが故障することがあります。
- このマニュアルは、必要なときすぐに参照できるよう、手近なところに保管してください。
- このマニュアルの記載内容について疑問点または不明点がございましたら、最寄りの当社営業またはSEまでお知らせください。
- お客様の誤操作に起因する事故発生や損害については、当社は責任を負いかねますのでご了承ください。
- 当社提供ソフトウェアを改変して使用した場合に発生した事故や損害については、当社は責任を負いかねますのでご了承ください。
- 当社提供以外のソフトウェアを使用した場合の信頼性については、当社は責任を負いかねますのでご了承ください。
- ファイルのバックアップ作業を日常業務に組み入れてください。ファイル装置の障害、ファイルアクセス中の停電、誤操作、その他何らかの原因によりファイルの内容を消失することがあります。このような事態に備え、計画的にファイルのバックアップを取っておいてください。
- 当社製品が故障や誤動作したりプログラムに欠陥があった場合でも、使用されるシステムの安全が十分に確保されるよう、保護・安全回路は外部に設け、人身事故や重大な災害に対する安全対策が十分確保できるようなシステム設計としてください。
- 非常停止回路、インタロック回路などはPLCの外部で構成してください。PLCの故障により、機械の破損や事故の恐れがあります。
- 運転中のプログラム変更、強制出力、RUN、STOPなどは十分安全を確認してから行ってください。誤操作により、機械の破損や事故の恐れがあります。

はじめに

このマニュアルは、S10V CMUのリアルタイム制御用オペレーティングシステムであるCPMS（Compact Process Monitor System）について、その機能とマクロコールのリンケージ仕様を中心に解説しています。このシステムにおけるリアルタイム制御プログラムを設計・開発されるときには、このマニュアルを読んでください。なお、このマニュアルは、一般のオペレーティングシステムについての基礎知識を持っている読者を対象に記述しています。

＜マニュアル構成＞

第１編　概　　説

第１章　概　　要
CPMSの構成と基本的な機能仕様について説明しています。

第２章　タスク管理
タスクの構成やスケジューリングなど、リアルタイム制御プログラムを製作する上で必要となるタスクに関する機能について説明しています。

第３章　メモリ管理
主メモリの割り当てやプロテクションなどのメモリ管理機能について説明しています。

第４章　タイマ管理
時刻と時間の管理方法について説明しています。

第５章　共用資源管理
タスク間で共用する資源の排他制御について説明しています。

第６章　入出力デバイス管理
入出力デバイスの識別方法などについて説明しています。

第７章　システム管理
このシステムの立ち上げについて説明しています。

第８章　タスクの異常処理
タスク異常が発生したときに実行される組み込みサブルーチンなどについて説明しています。

第９章　システムサービス

システムやタスクの稼働情報を取り出す機能などについて説明しています。

第２編　マクロ仕様

CPMSが提供するマクロコールの機能とリンケージ仕様について説明しています。

第３編　ライブラリ

算術演算などのライブラリの機能とリンケージ仕様について説明しています。

＜関連マニュアル＞

ソフトウェアマニュアルオペレーション　RPDP/S10V for Windows®（マニュアル番号 SVJ-3-133）

＜記憶容量の計算値についての注意＞

- 2^n計算値の場合（メモリ容量・所要量、ファイル容量・所要量など）

  1KB（キロバイト）＝1,024バイトの計算値です。

  1MB（メガバイト）＝1,048,576バイトの計算値です。

  1GB（ギガバイト）＝1,073,741,824バイトの計算値です。
- 10^n計算値の場合（ディスク容量など）

  1KB（キロバイト）＝1,000バイトの計算値です。

  1MB（メガバイト）＝$1,000^2$バイトの計算値です。

  1GB（ギガバイト）＝$1,000^3$バイトの計算値です。

目　次

図目次

表目次

第１編　概　　説

第１章　概　　要

１．１　CPMSの機能

CPMS（Compact Process Monitor System）は、リアルタイムオペレーティングシステムの核です。
CPMSは以下の機能を持っています。

- タスク管理機能
  最大255本までのマルチタスク実行を制御します。
- メモリ管理機能
  メモリのアドレス変換およびプロテクションを制御します。
- タイマ管理機能
  システムの持つ時刻・時間を制御します。
- 共用資源管理機能
  タスク間の共用資源の排他制御をします。
- 入出力デバイス管理機能
  各種入出力デバイスを管理し、I/Oドライバを組み込みます。
- システム管理機能
  システムの初期処理、状態や構成を制御します。
- システムサービス
  システムが持つ情報やサービスを提供します。

１．２　CPMSの仕様

表１－１にCPMSの仕様（システムパラメータ）を示します。

表１－１　CPMSの仕様

項　　目	値	備　　考
タスクの数	最大255タスク	タスク番号は、 1～224がユーザタスク 225～229がシステムタスク 230～255がOSタスク
タスクの優先度	32レベル	ユーザは4～27 システムは0～31
タイマの数	320	TIMERマクロ、DELAYマクロ、WAKEマクロで使用される
リソース管理の数	同時確保最大16個	RSERVマクロ、PRSRVマクロで使用される
DHPバッファ	128KB	12～36バイト／1ケース
エラーログバッファ	32KB	1KB／1ケース
組み込みサブルーチン	10ポイント	各ポイントに4エントリ

１．３　CPMSの構造

CPMSは、図１－１に示すようにエクセプション処理プログラム、ディスパッチャ、システムタスクで構成しています。

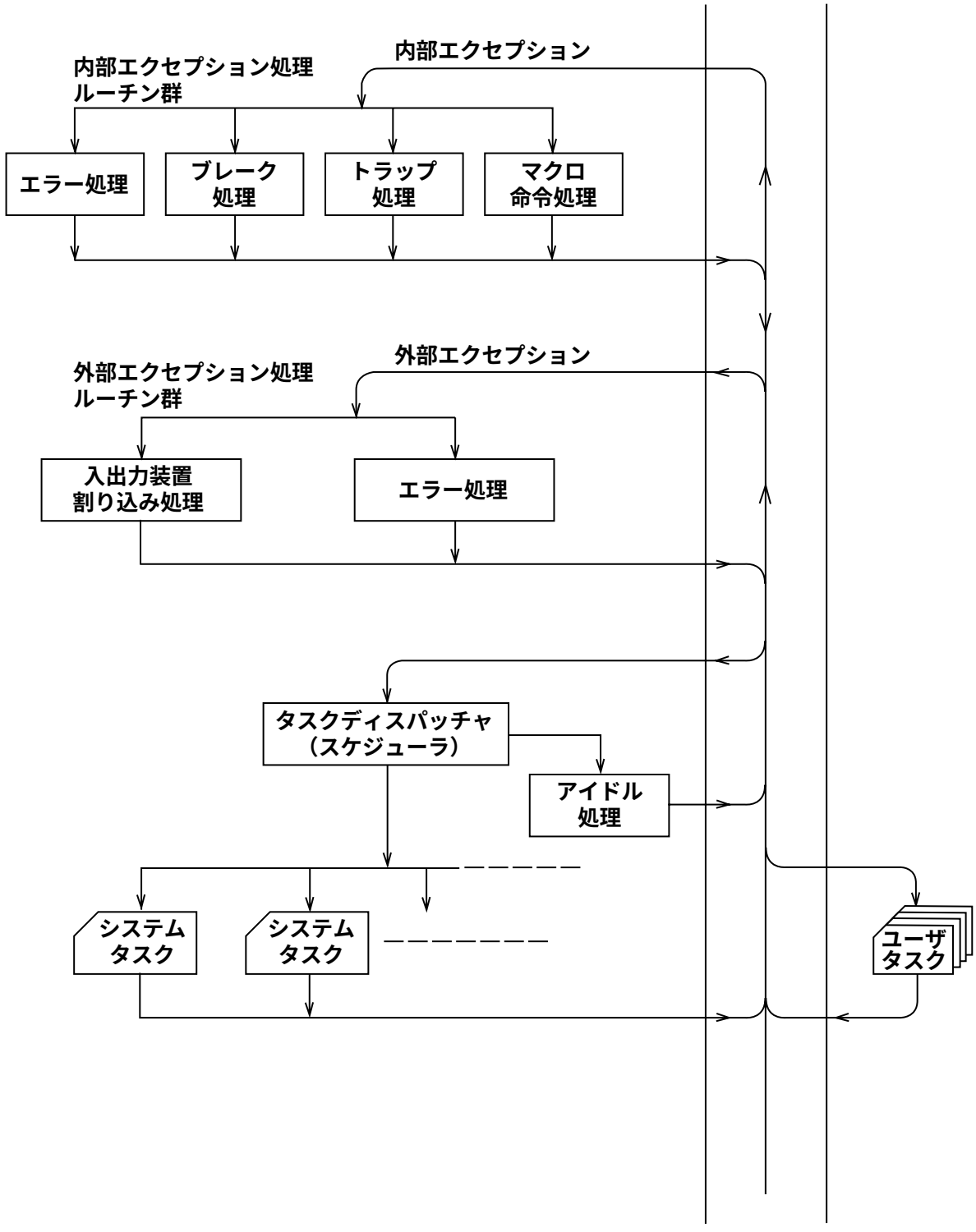


図１－１　CPMSの構造

１．４　CPMSとハードウェア

S10V CMU構成とCPMSとの関係を図１－２に示します。

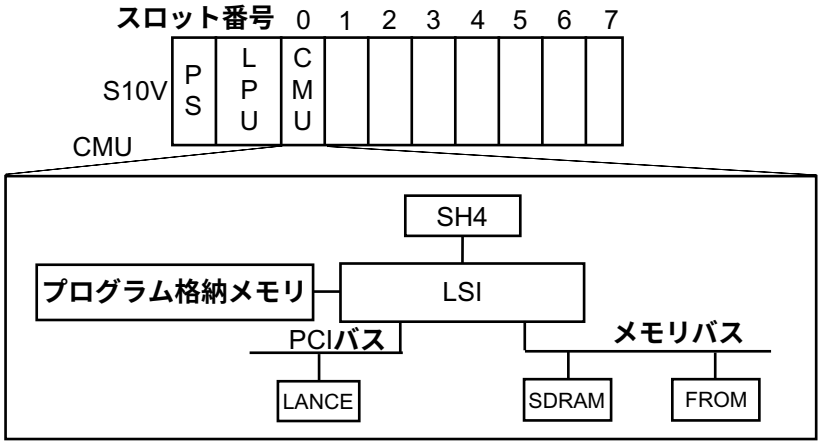


図１－２　ハードウェア構成とCPMSとの関係

・プロセッサ（SH4）

制御用プログラムタスクが動作します。

・LSI

プロセッサからのメモリアクセスやバスアクセスを制御します。

・メモリバスとメモリ

メモリバスに主メモリ（SDRAM）, FROMが存在します。

SDRAM　　：CMUの主メモリです。OSやプログラムが動作します。
　　電源断、リセットにより内容は消去します。

FROM　　：OS等プログラムが存在します。

プログラム格納メモリ：RPDP実行環境、タスク、HI-FLOWプログラムを格納するフラッシュメモリです。立ち上げ時にプログラム格納メモリからSDRAMへデータをコピーします。

１．５　CPMSとユーザのインタフェース

CPMSとユーザのインタフェースは、RPDP（Realtime Program Development Package）からの操作、ユーザタスクからのマクロ命令発行、組み込みサブルーチンへのリンクがあります。

RPDPは、タスク、組み込みサブルーチンの作成環境を提供します。

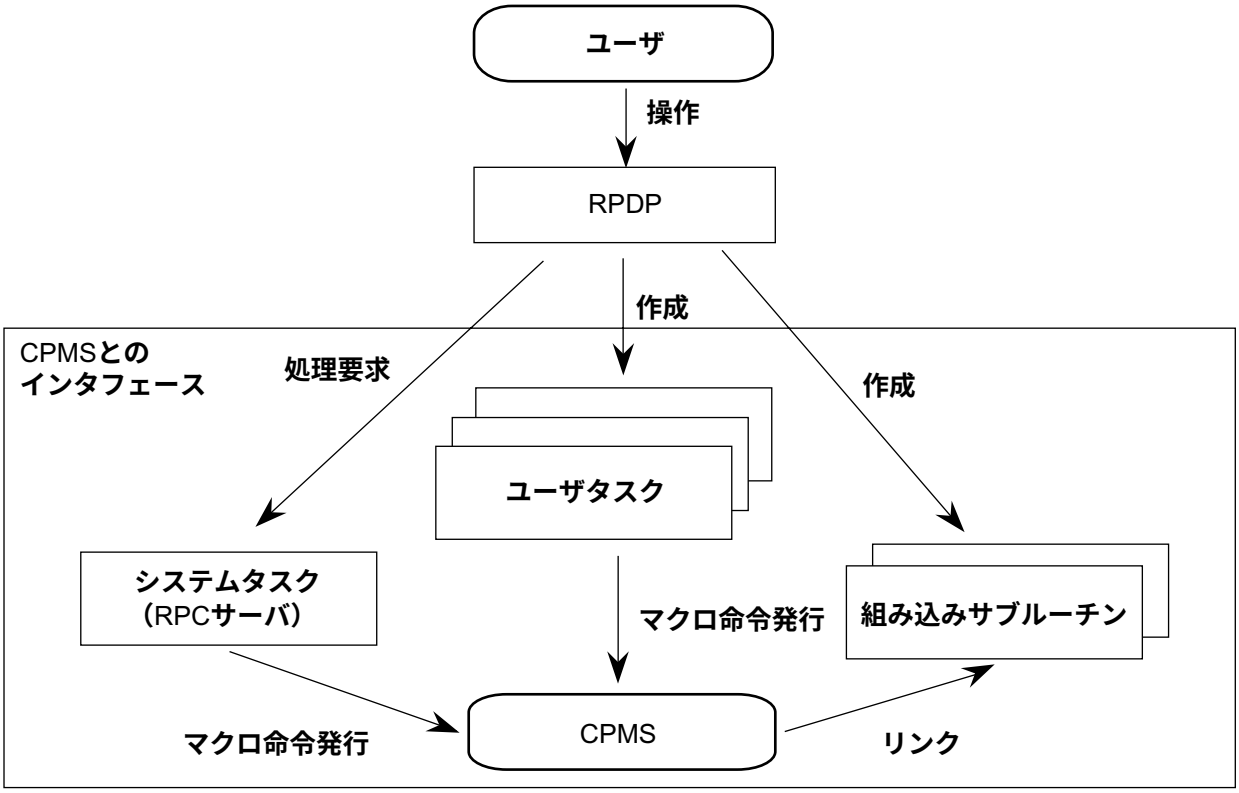


図１－３　CPMSとユーザのインタフェース

第２章　タスク管理

２．１　タスク

タスクとはプログラムの実行単位です。CPMSは、プログラムの実行管理や資源の割り当てをタスク単位に管理します。

(１) タスク番号

タスク番号（TN: Task Number）は、タスクを識別するための番号です。CPMSは最大255本のタスクを管理できます。

ユーザは、タスク番号1～224に、ユーザタスクを割り当てることができます。タスク番号225～229は、システムタスクです。また、タスク番号230～255は、OSで予約しています。

タスク番号1タスクは、イニシャルスタートタスクとして、立ち上げ時にCPMSによって起動されます。

(２) タスクの構造

タスクは、TEXT, DATA, BSS, STACK, OSワークからなります。TEXTはプログラムの実行部分です。DATAは初期値ありデータ部分です。BSSは初期値なしデータ部分です。STACKはプログラム実行に使用される作業データ部分で、アドレスの高い方から低い方へ使用されます。TEXT, DATAは書き込みから保護されます。OSワークは、CPMSがマクロ実行時に使用する作業データ部分です。

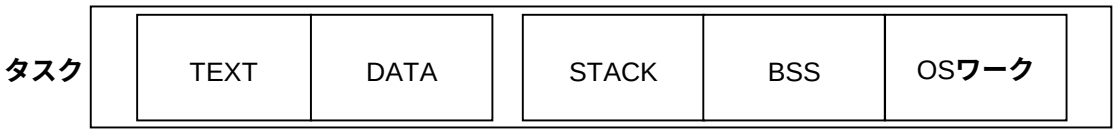


図１－４　タスクの構造

TEXT, DATA, BSSを共有するマルチタスクを作成できます。このマルチタスクの場合、STACKはタスクごとに持ちますが、BSSは共有されますので注意してください。

(３) タスクの種類

タスクは2種類あります。ユーザが作成するユーザタスクとシステムが提供するシステムタスクです。タスク番号の225から255はシステムタスク、OSのために予約されています。タスク番号の1から224にユーザタスクが割り当てられます。

(４) イニシャルスタートタスク

タスク番号1はユーザイニシャルスタートタスク（UIST）です。ユーザタスクは、ユーザイニシャルタスクから起動するようにユーザが作成してください。

(５) タスクの優先レベル

複数のタスクが、システム内の共用資源（CPU, メモリ）に同時に使用要求を出している場合に、どのタスクにその資源の使用権を与えるかは各タスクに付けられた処理優先度により決定されます。この処理優先度のことを優先レベルあるいは単にレベルといいます。レベルは0～31の数値であり、値が小さいほど優先度が高いことを示します。ユーザはレベル4から27を使用できます。タスクを登録する際に、そのタスクのレベルを指定します。このレベルを、そのタスクのオリジナルレベルといいます。通常はタスクが起動されると、このオリジナルレベルがタスクの動作中のレベル（カレントレベル）となります。このカレントレベルに従って各資源の使用割り当て順位が決定されます。

優先レベルはタスクを登録するときに指定します。システムタスク，ユーザタスクに割り付けできるレベルの関係を図１－５に示します。

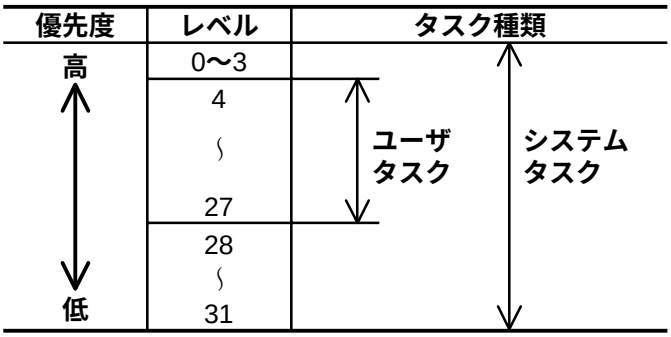
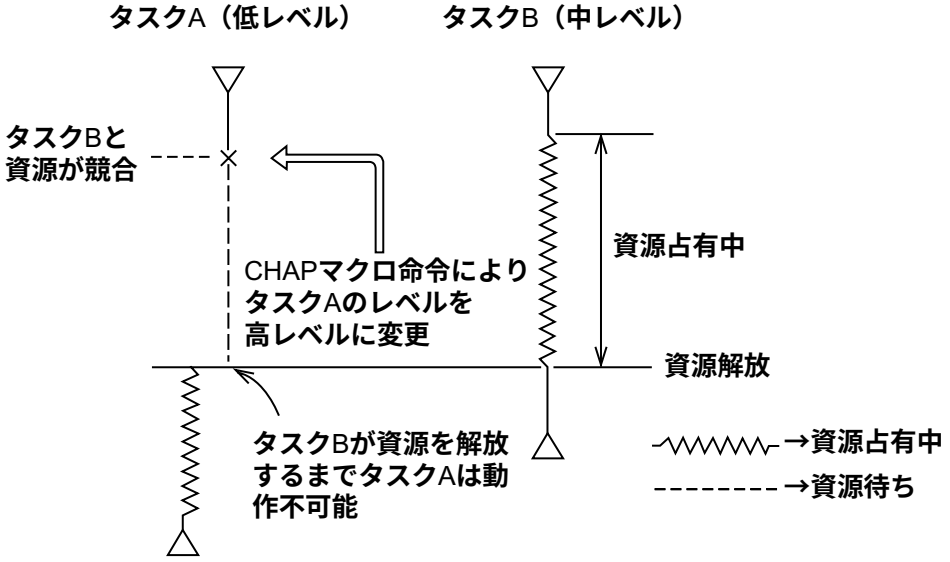


図１－５　タスクのレベルと種類の関係

(６) 優先レベルの変更

タスクが実行中に、与えられたレベルを変更することはCHAPマクロ命令により行えます。CHAPマクロ命令の効果は、レベルを変更されたタスクが動作をはじめてから終了するまで有効です。終了すると元のオリジナルレベルがそのタスクのレベルとなります。タスクが動作をはじめる前にCHAPマクロ命令によりレベルを変更しておくと、動作をはじめたときには変更された新しいレベルが動作中の優先レベルとなります。ただし、変更されてから動作開始までの間にABORTされると、CHAPマクロ命令の効果はなくなります。

CHAPマクロ命令は、諸資源の割り当て基準となる優先レベルを変更するものです。あるタスクにすでに割り当てられている資源を高い優先レベルに変更されたタスクへ強制的に与えることはしません。これを図１－６に示します。



（注）タスクAがタスクBと資源を競合して待ち状態となっているとき、CHAPマクロ命令によりタスクAの優先レベルを上げても、タスクBが占有している資源はタスクAには与えられません。

図１－６　優先レベルの変更と資源

２．２　タスクのスケジューリング

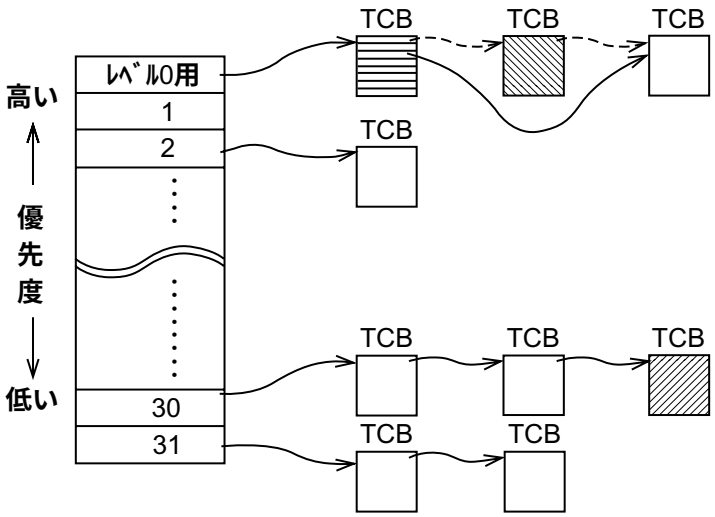
(1) スケジューリングのアルゴリズム

システムが稼働しているとき、そのシステム内の複数のタスクに起動要求が発行されるとCPUの使用権を要求しているタスクは複数個存在します。

しかし、CPUはシステム内に1台です。複数のタスクのうちCPUのサービスを受けられるのは常に1つのタスクです。このように、複数のタスクの中からCPUの使用権を与えるタスクを選び出すことを、「ディスパッチする」といいます。どのようにディスパッチするかをタスクのスケジューリングと呼びます。

スケジューリングのアルゴリズムはいろいろありますが、CPMSでは優先レベル順の固定プライオリティスケジューリング方式を採用しています。また、同一レベル内では、FCFS（First Come First Served）アルゴリズムを用いています。

FCFSでは、起動要求の先着順にCPU待ち行列にタスクがつながれます。実際には、図１－７に示されるように、タスクを管理するテーブルであるTCB（Task Control Block）がCPU待ち行列につながれます。1つのタスクには、1個のTCBが割り当てられています。



：新規登録の例です。
：解除の例です（点線のポインタが実線に変わります）。
：この図において最初にディスパッチされるタスクです。

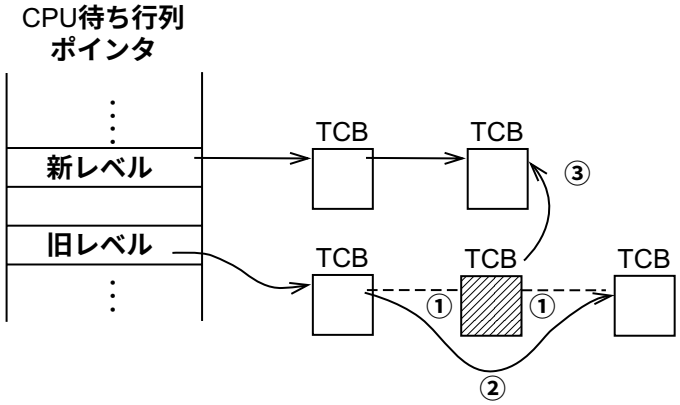
図１－７　CPU待ち行列

タスクがCPU待ち行列から解除されるのは、次の3つの場合です。

- 自タスクがEXITマクロ命令を発行したとき
- 自タスクが、ABORTマクロ命令（実行禁止）を発行した他タスクの対象となったとき
- タスクが異常を起こしたとき（例えば、不当なデータをアクセスしようとしてプロテクションエラーとなったとき）。タスクが異常を起こしたときは、CPMSによりタスクはABORTされます。

(2) レベルの変更

図１－８は、CHAPマクロ命令を発行したとき、CHAPマクロ命令の対象となったタスクのTCBがCPU待ち行列の中でどのように扱われるかを示したものです。



＜手順＞

① 旧レベルの待ち行列から解除します。

② 旧レベルの待ち行列をつなぎ変えます。

③ 削除した対象TCBを指定レベルの最後尾につなぎます。

（注）CHAPマクロ命令によりレベルを変更すると、FCFSアルゴリズムではレベルが変更されたタスクのTCBを新レベルの最後尾につなぎます。

図１－８　レベルの変更

(３) マルチタスキング

タスク管理では、CPUを無駄なく使用するための処理を行っています。

例えば、現在進行中のタスクが何らかの原因で先に進めなくなったときは、ただちにCPU待ち行列で現在進行中のタスクの次に接続されているタスクをディスパッチします。ディスパッチされたタスクは動作を開始します。このタスクの実行中に、先に中断されたタスクの中断要因がなくなったらディスパッチャは中断されていたタスクを再びディスパッチします。この例を、図１－９に示します。ある瞬時には1つのタスクしか動作していませんが、マクロにみるとあたかもタスクA, B, C…が同時に動作しているようにみえます。一般には複数のタスクがこのように処理されます。これをタスクの並行処理といいます。この並行処理によってCPUの効率を高めることができます。

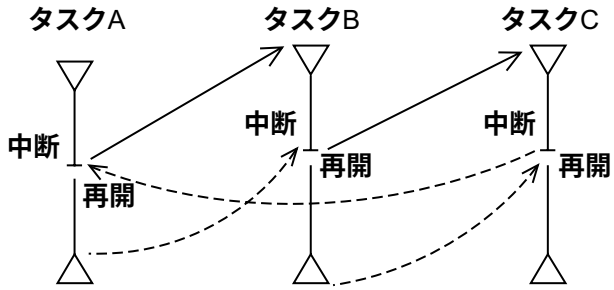


図１－９　タスクの並行処理（マルチタスキング）

２．３　タスクの動作

一般にタスクは1つのライフサイクルを持っています。すなわちタスクは生成され，起動され，実行され，中断され，再開され，終了し，消滅します。ただし、リアルタイムタスクでは、起動要求が出されてから実行されるまでのオーバヘッドは応答の速さを決める重要な要因になりますので、生成・消滅を必要最小限にとどめることが必要です。そのため起動要求が出されてからタスクを生成することは行わず、あらかじめタスクを作りこんでおきます。すなわち、リアルタイムタスクを起動するときは対象タスクをあらためて生成する必要はなく、単に起動要求を発する（QUEUEマクロ命令を発行する）だけで済みます。また、動作終了後に消滅させることもしません。

タスク起動のきっかけとなるイベント（要因）を、表１－２に示します。

タスクの実行条件（イニシャル起動）を、表１－３に示します。タスクは起動された後、表１－３に示す条件のすべてが成立したときに実行されます。

こうして実行されたタスクは、処理が続行できなくなったとき、あるいは割り込みが発生し、自タスクより優先レベルの高いタスクを動かさなければならなくなるときまで動作を続けます。必要な処理がすべて済んでしまえば（プログラムの実行が終了すれば）タスクの動作は終了します。これをタスクの中断および終了（打ち切り）と呼びます。

表１－４にタスクの中断要因を示します。中断されたタスクは、その中断要因が取り除かれ、自分より高レベルのタスクあるいは自分と同レベルで先に起動がかかっていたタスクが動作できないときに動作が再開されます。これをタスクの再開と呼びます。

表１－５にタスクの再開条件を示します。

表１－６にはタスクの終了条件を示します。

表１－２　タスクの起動要因

	イベント	説　　明
内部要因	QUEUEマクロ命令の発行	1つのタスクからQUEUEマクロ命令が発行されると、そのパラメータで指定されたタスクが起動されます。
	一定時間の経過または 一定時刻になった	TIMERマクロ命令が発行されていると、そのパラメータで指定されたタスクが、指定時間／時刻に起動されます。
外部要因	入出力装置からの アテンション割り込み	入出力装置からのアテンション割り込みにより、組み込みサブルーチンからタスクが起動されます。

表１－３　タスクの実行条件（イニシャル起動）

条　　件	説　　明
自分よりも高レベルあるいは同レベルで先に起動されているタスクのすべてが動作できないこと	自分よりも高レベルのタスクが動作できるときはそのタスクが実行されます。
主記憶装置にメインプログラムがローディングされていること	プログラムは主記憶装置にローディングされていなければ動作できません。
自分自身の実行が抑止されていないこと	SUSP, ASUSPマクロ命令により実行が抑止されている場合は実行されません。

表１－３のすべての条件が成立しているとタスクは実行されます。

表１－４　タスクの中断条件

条　　件	説　　明
より優先レベルの高いタスクに起動がかかったとき	割り込み（プロセス割り込み，タイマ）により、優先レベルの高いタスクが起動され、かつそのタスクが動作可能なときはそのタスクに制御が移ります。
実行を抑止されていた優先レベルの高いタスクの実行抑止が解除されたとき	より優先レベルの高いタスク（それまで実行抑止されていたタスク）が動作可能となるとそのタスクに制御が移ります。
自ら実行を中断したとき	同期をとるときなど自ら実行を中断すると他のタスクに制御が移ります。

表１－４の条件のいずれか1つが成立するとタスクは中断されます。

表１－５　タスクの再開条件

条　　件	説　　明
他のタスクによる実行抑止状態が解除された	SUSP, ASUSPマクロ命令などによる実行抑止状態が解除されました。
事象の発生待ちになっていたときその事象が発生した	自らの中断（DELAY, WAIT）要因が除かれる事象が発生しました。
自分より高レベルあるいは同レベルで先に起動がかかっていたタスクが終了あるいは中断したとき	自分より高レベルあるいは同レベルで先に起動がかかっていたタスクが動作可能である限り自分にはCPUのサービスの順番がまわってきません。

表１－５の条件のいずれか1つが成立したときタスクは動作を再開します。

表１－６　タスクの終了条件

条　　件	説　　明
EXITマクロ命令を発行したとき	通常タスクは、EXITマクロ命令で処理を終了します。
ABORTマクロ命令の対象となったとき	ABORTマクロ命令により処理を打ち切られる場合です。
プログラム異常などで処理継続不可能な事象が発生	CPMSが自動的に異常を生じたタスクのABORT処理を行います。

表１－６の条件のいずれか1つが成立したときタスクは動作を終了します。

２．４　タスクの状態遷移

CPMSシステムでは、複数のリアルタイムタスクが相互に結合され、動作し、システム全体としての機能を果たします。このため、各タスクが起動・中断・再開・終了（２．３節参照）を相互に繰り返しながら連携をとって動作を続けます。

タスク間相互のデータの受け渡しは、タスク間の共通データエリアであるGLB（Global Data Area: グローバルデータエリア）を用いて行われます。タスク間相互の制御の受け渡しはタスク管理が用意するマクロ命令を用いて行われます。

タスク管理マクロ命令は、タスクの状態を遷移（変化）させることによってタスクの動作を制御します。システムを効率よく、しかも確実に正しく動作させるためには、タスクの状態遷移がどのように行われ、どのようなマクロ命令により状態遷移が生じるのか正しい認識を持ったうえで、システム設計、プログラム設計を行ってください。

表１－７にタスクの状態を示します。また、図１－10には、タスクの実行・状態を制御する各マクロ命令とタスクの状態の関係を示します。

なお、図１－10の状態遷移のRUNNING状態は中断中であることも含んでいます。また、マクロ命令の対象となったタスクの状態遷移は1つの例を示したものであって、すべてのケースを表したものではありません。

表１－７　タスクの状態

状　　態	略　　称	説　　明
実行中状態	RUNNING	CPUを占有して、タスクを実行している状態です。
実行待ち状態	RUNNABLE	CPUが空くのを待っている状態です。
実行抑止状態	SUSPENDED	実行が抑止されている状態です。
イベント待ち状態	WAIT	イベントの発生を待ち合わせている状態です。
起動待ち状態	IDLE	起動されるのを待っている状態です。
起動抑止状態	DORMANT	起動が抑止されている状態です。
未登録状態	NON-EXISTENT	CPMSに登録されていません。

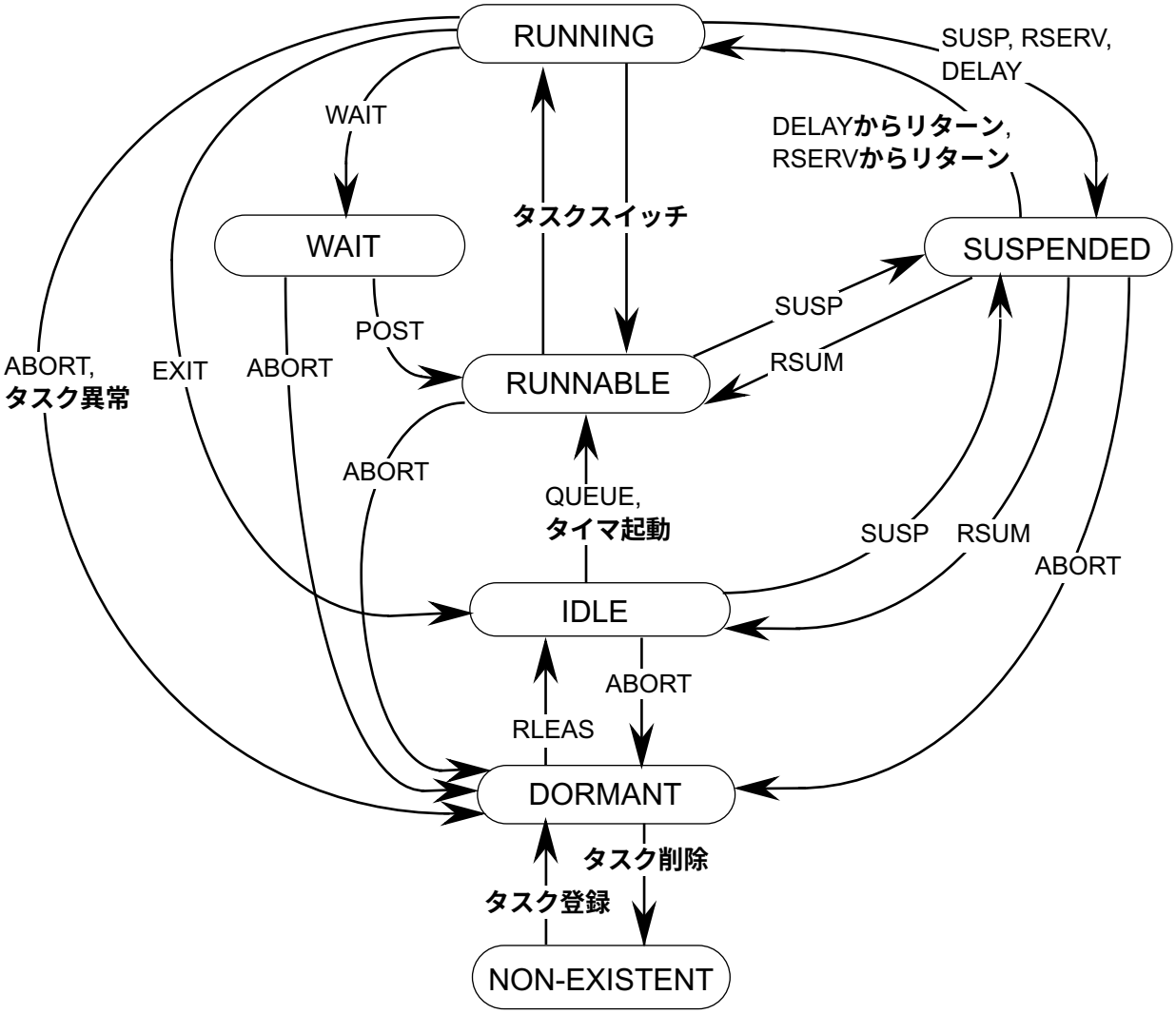


図１－10　タスクの状態遷移

（補足）queueされていないIDLE状態のタスクにsusp/rsumが発行されると、それぞれ実行抑止／実行抑止の解除が行われますが、そのタスクの状態はIDLE状態のままで、実行抑止されている情報をタスク管理テーブルTCBのtc_flagフィールドに格納します。

２．５　タスクの制御

タスクを制御する方法について、以下に例を示しながら説明します。

２．５．１　初期の状態

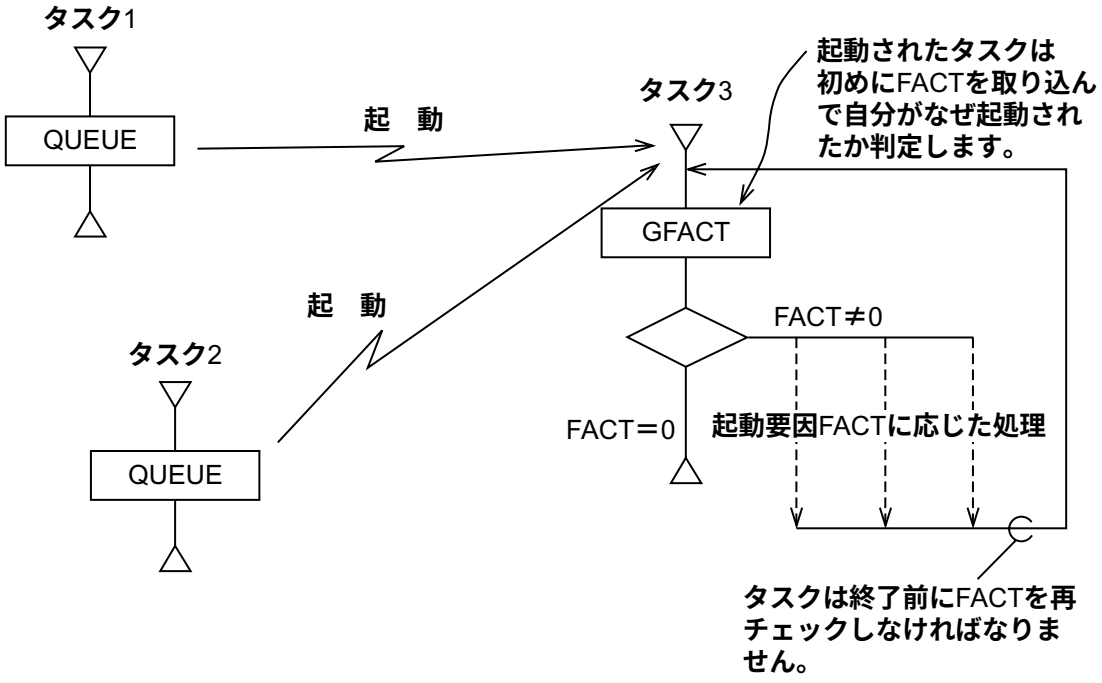
システムが立ち上がったとき（電源が投入され、処理装置が動作をはじめたとき）には、ユーザのタスクはイニシャルスタートタスクを除いて、すべてDORMANT状態にあります。

イニシャルスタートタスクは、システムが立ち上がったときにCPMSにより自動的に起動されます。イニシャルスタートタスクは、業務の実行に必要なタスクをRLEASマクロ命令によりIDLE状態とします（これを、タスクをreleaseするといいます）。これは起動受け付け可能な状態です。

２．５．２　タスクの起動

● QUEUEマクロ命令

タスクは、QUEUEマクロ命令により起動されます。起動されたタスクは起動要因（FACT）をGFACTマクロ命令により取り込み、何の要因で自分が起動されたのかを判定します。これを図１－11に示します。



タスク3はGFACTマクロ命令により、自分が誰から起動されたのか（タスク1あるいはタスク2）を判定します。すなわち、タスク1がタスク3を起動するときのFACTと、タスク2がタスク3を起動するときのFACTを変えておけば、タスク3がFACTを判定して、タスク1, 2いずれから起動されたかを知ることができます。

図１－11　タスクの起動

図１－11で、GFACTマクロ命令は起動要因を1つずつ取り込んでいきます。例えば、起動要因（1～32の整数）が1, 5, 10, 11と4つ設定されていたとすると、GFACTマクロ命令はこれを番号の若い方から順に取り込んでいきます。1回目のGFACTマクロ命令の発行でFACT＝1が取り込まれ、次に再びGFACTマクロ命令を発行するとFACT＝5が取り込まれます。一度取り込まれたFACTはGFACTマクロ命令によって0クリアされます。したがって、FACT＝1が取り込まれた後再びGFACTマクロ命令を発行しても、FACT＝1は取り込まれません。

こうしたFACTは、SFACTマクロ命令により設定することもできます。この例を図１－12に示します。

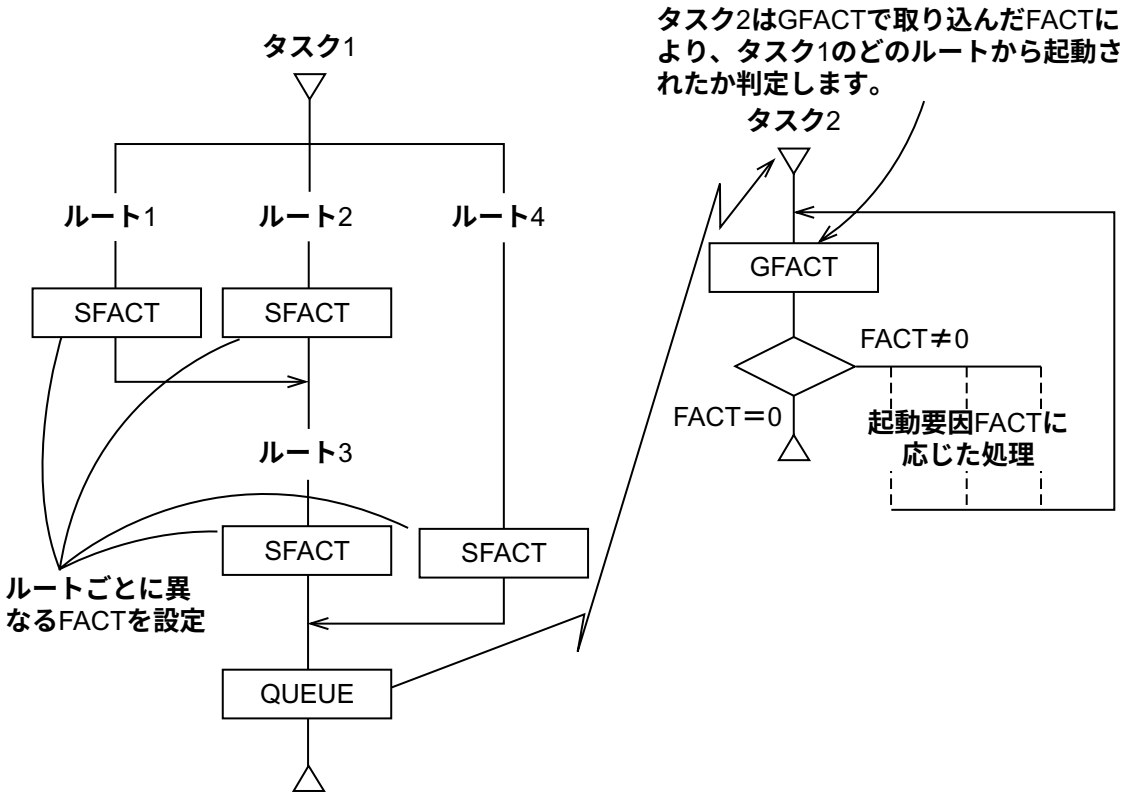
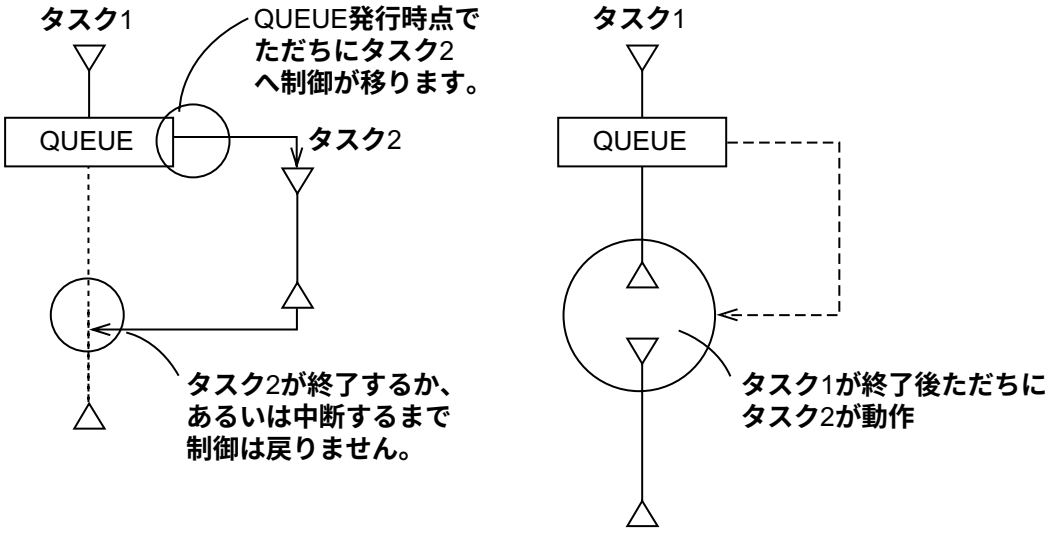


図１－12　SFACTマクロ命令

CPMSでは、リアルタイム制御を効率よく行うため、先に述べたように同一レベル内先着優先順，レベル順制御を行っていますので、タスク起動の相互のレベル関係などによりタスク実行順の流れが図１－13に示すように変わります。これは、CPMSでのタスクスケジューリングを理解する上でも重要な点です。



タスク2の優先レベルがタスク1の優先レベルより高い場合

タスク1の優先レベルがタスク2の優先レベルより高いかあるいは同レベルの場合

QUEUEマクロ命令によりタスク1がタスク2を起動したとき、タスク間の相互レベル関係よりプログラム実行制御の流れが異なります。

図１－13　QUEUEマクロ命令とタスクの実行順序

● TIMERマクロ命令

図１－13からわかるように、QUEUEマクロ命令によるタスクの起動は通常ただちに行われます。しかし、場合によってはある一定時間後あるいはある時刻にタスクを起動することがあります。このときは、TIMERマクロ命令を使用してください。このマクロ命令を用いると、パラメータで指定した時刻あるいは時間経過時に指定タスクを起動できます。なお、このときも起動要因FACTは、QUEUEマクロ命令によるFACTとまったく同様に起動されたタスクへ渡されます。

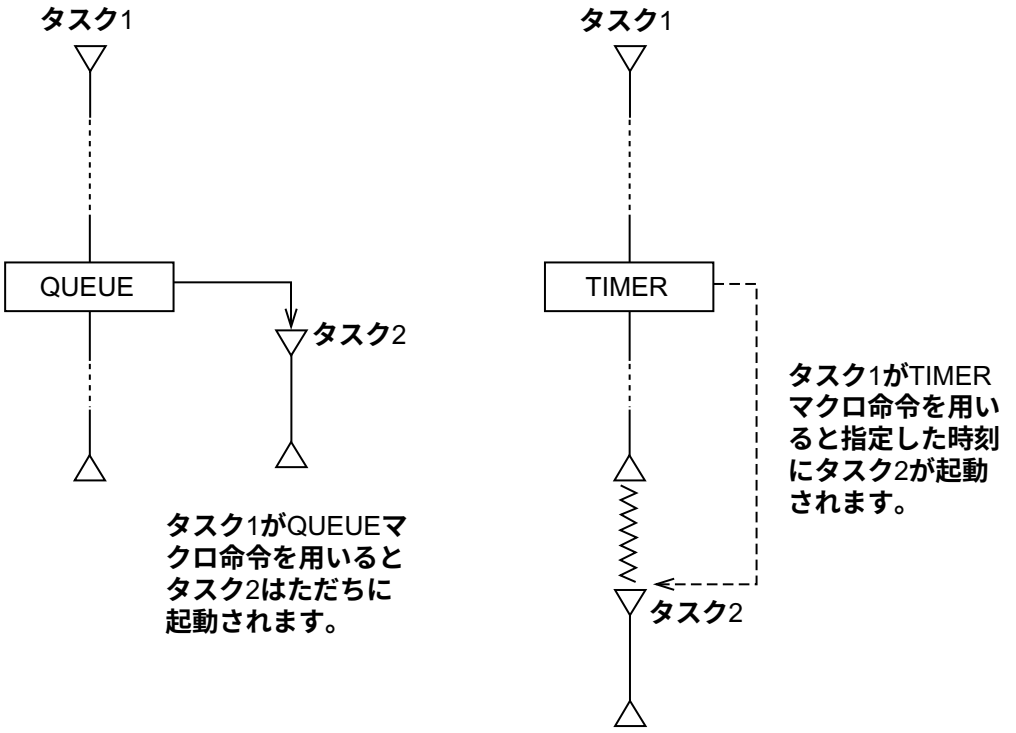


図１－14　QUEUEマクロ命令とTIMERマクロ命令によるタスク起動の違い

２．５．３　タスクの終了

タスクは、自ら発行するEXITマクロ命令によって終了します。CPMSでは、タスクのメインルーチンのリターンでもEXITマクロ命令が発行できます。

２．５．４　タスクの実行抑止

- DELAYマクロ命令

  TIMERマクロ命令は、主に他のタスクを一定時間後に起動するためのマクロ命令です。TIMERマクロ命令を自分自身に対して発行し、一定時間後に動作させることもできます。DELAYマクロ命令を用いればパラメータで指定した一定時間の経過後、再び自分自身に制御が戻ったときのためにDELAYマクロ命令発行時の環境（BSS, STACKの値など）が保存されています。TIMERマクロ命令を用いたときはタスクの先頭から動作をはじめます。環境は保存されていません。したがって、一定時間中断後、再び動作したいときはDELAYマクロ命令を用います。

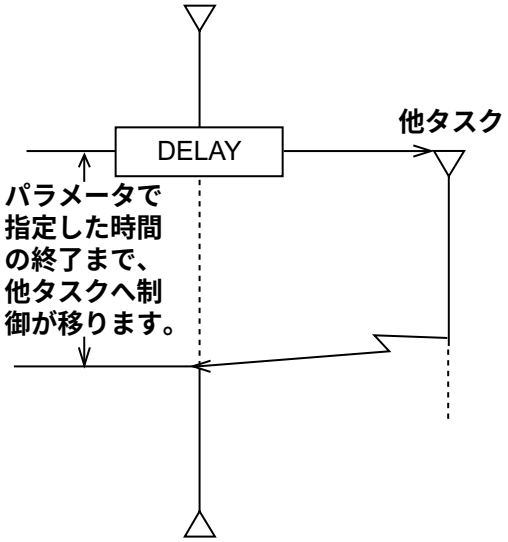


図１－15　DELAYマクロ命令

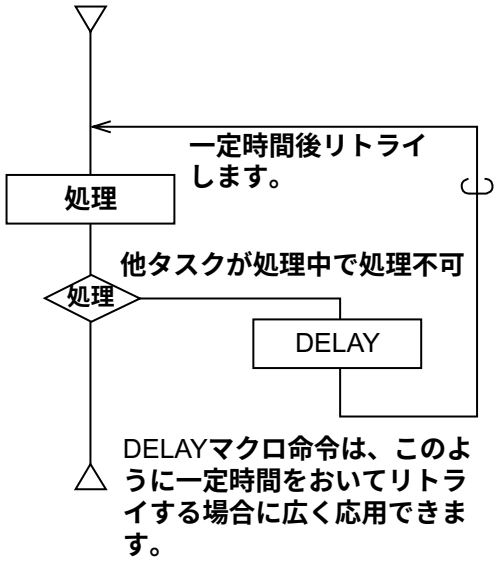


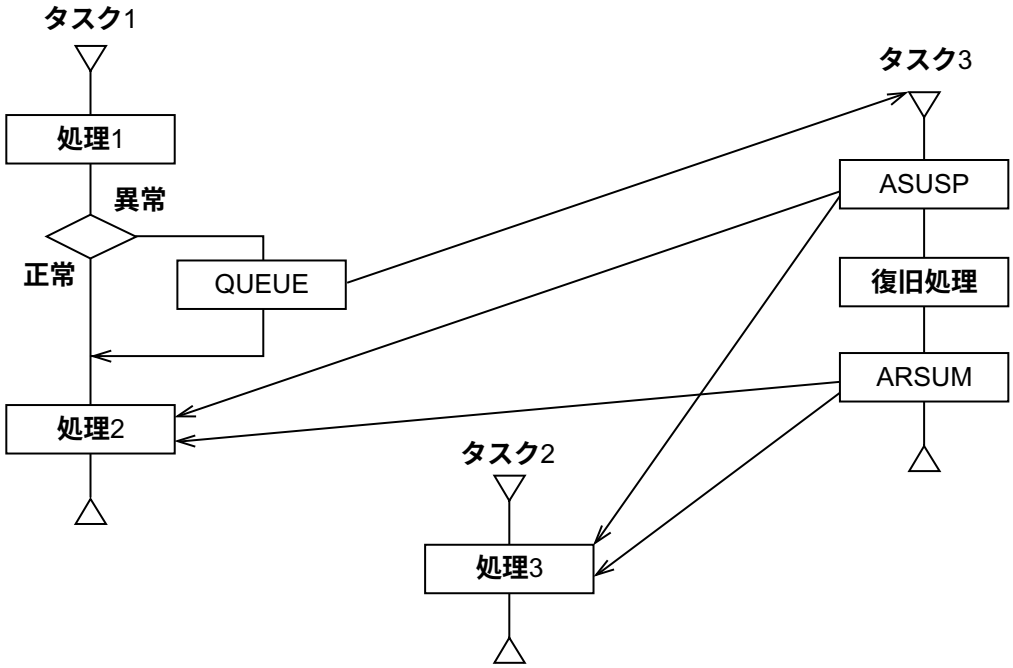
図１－16　DELAYマクロ命令の適用

- ASUSPマクロ命令

  優先レベルの高いタスクも含めて他のすべてのタスクの実行を抑止したい場合は、ASUSPマクロ命令を用います。

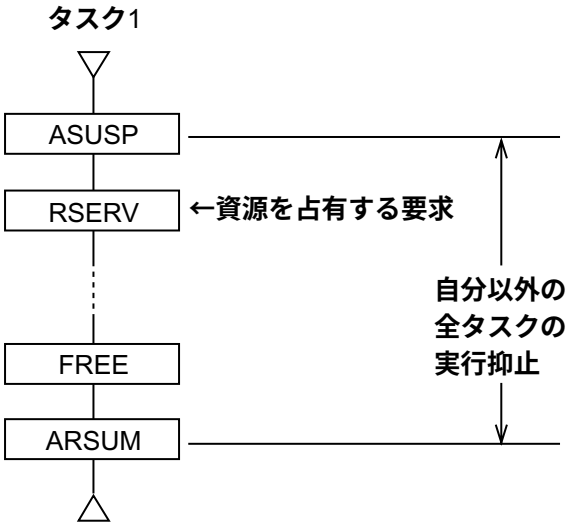
  ASUSPマクロ命令で実行を抑止されたタスクは、ARSUMマクロ命令で抑止を解除されます。ただし、これらの命令は他のタスクの実行を抑止するものですので使用を制限しないとデッドロックなどを引き起こす可能性があります。

  デッドロック防止のためASUSPマクロ命令を発行してからARSUMマクロ命令を発行する間に、システム内の資源を必要とする処理を行わないでください。



処理1, 2, 3を管理するタスク3は、処理1が異常であったときASUSPマクロ命令によりタスク1, 2の実行を抑止します。抑止後、処理2, 3が正常に行えるよう復旧処理をした後、ARSUMマクロ命令によりタスク1, 2の実行抑止を解除します。タスク3は復旧処理が終わるまで、処理2, 3などをASUSPマクロ命令により待たせます。

図１－17　ASUSPマクロ命令による実行抑止



他のタスクの実行を抑止しておいて資源を占有する要求を出すと、実行を抑止されたタスクがその資源を占有中の場合、デッドロックとなります。

図１－18　ASUSPマクロ命令によるデッドロックの例

２．５．５　タスクの打ち切り

- ABORTマクロ命令

  タスクの実行を打ち切り、以後そのタスクを実行禁止状態にするには、ABORTマクロ命令を用います。

  ABORTマクロ命令は、実行中（あるいは実行待ち）のタスクを打ち切り、そのタスクが占有している資源を強制的に解放し、タスクをDORMANT状態にします。

２．５．６　タスク間の同期

複数のタスク間で同期をとる（あるタスクの処理が終わってから他のタスクでの処理を行う）ために、WAITマクロ命令およびPOSTマクロ命令が用意されています。こうした同期は、イベントと呼ばれる概念を用いて制御されます。すなわち、同期をとるため待ちになるタスクは、ECB（Event Control Block）と呼ばれるエリアにイベントの発生を待っていることを表示し、待ち状態となります。このECBはイベントごとに定義されています。イベントの発生を知らせるタスクはECBを参照し、誰がイベントの発生を待っているかを調べ、待っているタスクに対してイベントの発生を知らせ、その待ち状態を解除します。こうした処理はそれぞれWAITマクロ命令およびPOSTマクロ命令により行われます。この様子を図１－19に示します。

ECBは、1つのイベントに対して1つ割り当てます。複数のイベントで同一のECBを共用しないでください。また、複数のタスクで同一のECBを共用しないでください。ECBを通じて、タスク間でイベントの詳細情報を受け渡すことができます。これをPOSTコードと呼びます。

WAIT/POSTマクロ命令には発行上の順序関係に制限はありません。これを、図１－20に示します。

デッドロック防止のため、ASUSPマクロ命令発行後WAITマクロ命令を発行すると、ASUSPマクロ命令の効力が無効とされます。

図１－21にECBの状態遷移を示します。

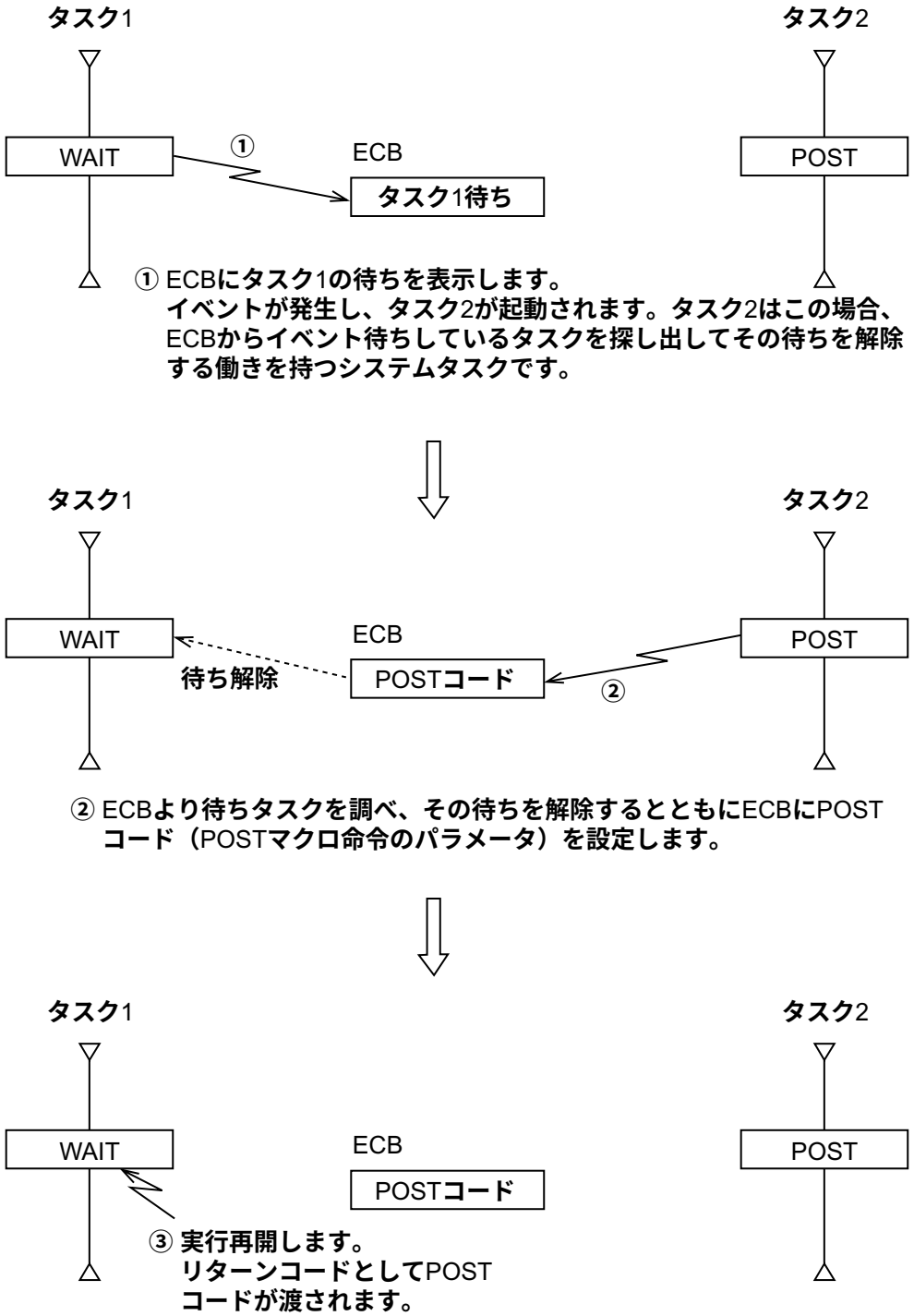
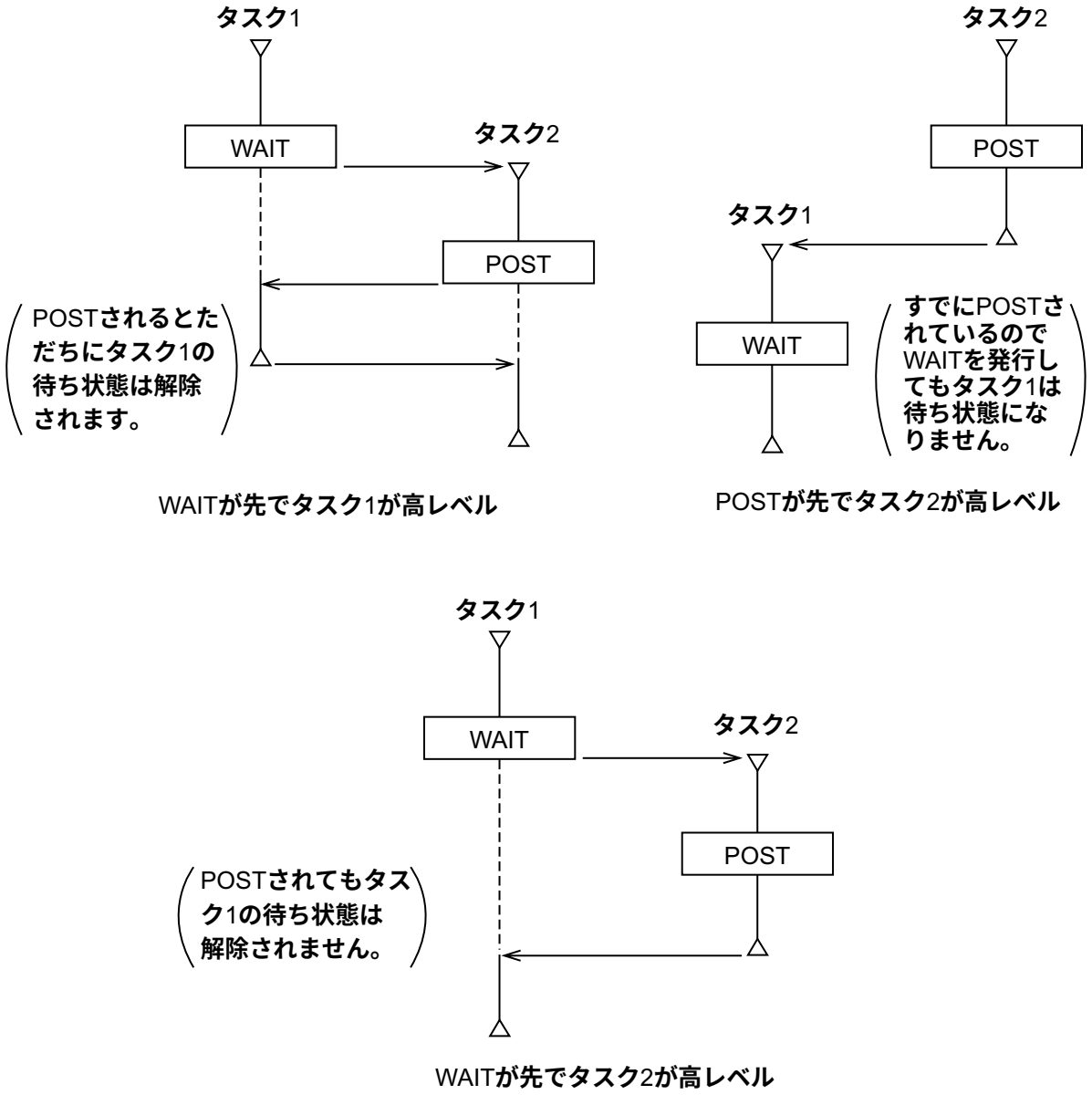
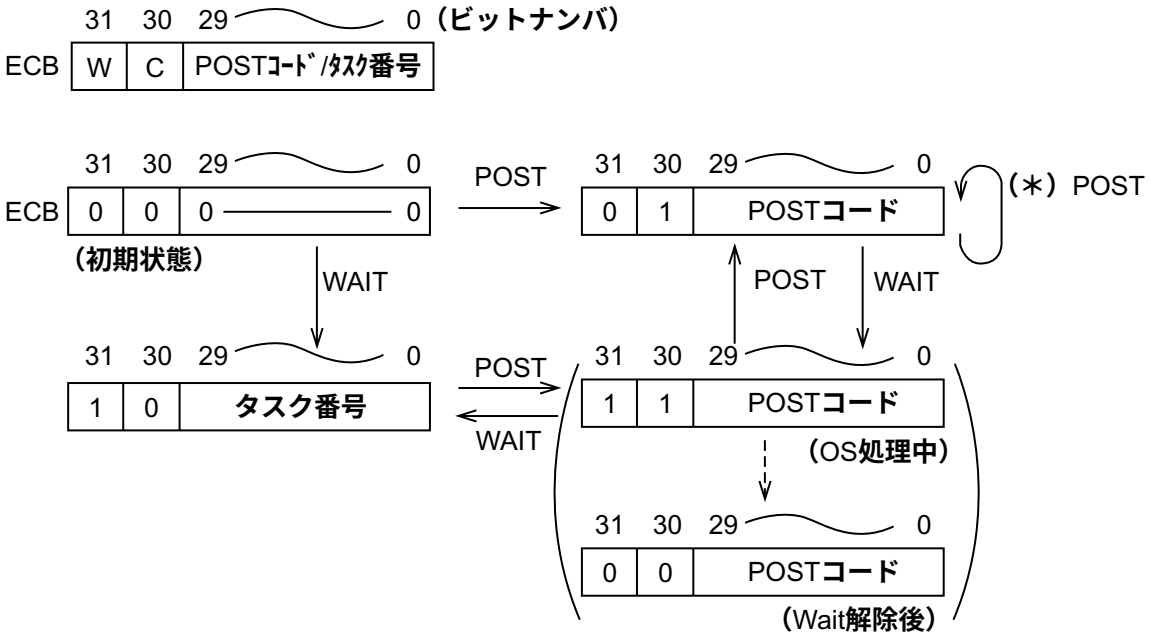


図１－19　WAIT/POSTによるタスク間の同期



タスク相互のレベルの違い、WAIT/POSTのどちらが先に出ているかでタスク間の制御の流れが異なります。
---- 部はその間タスクの実行が抑止（待ち状態）されていることを示します。

図１－20　WAIT/POSTでの制御の流れ



ECBのビットナンバ31, 30は、それぞれW（Wait）ビット，C（Complete）ビットと呼びます。

（*）POSTコードは上塗りされます。

図１－21　ECBの状態遷移

第３章　メモリ管理

３．１　論理空間

CPMSは、すべてのタスクを1つの論理空間で動作させます。CPMSは、論理アドレスと物理アドレスの変換を管理します。

リザーブ：CPMSで予約されています。
S10空間：LPUのメモリが割り当てられます。
NX用ユーザエリア：NXが使用するバッファエリアが割り当てられます。
HFLOW空間：HIFLOWプログラムが割り当てられます。
システムバス空間：高速バスのI/Oやメモリが割り当てられます。
PCIバスメモリ空間：PCIバスのI/Oやメモリが割り当てられます。内蔵LANCEが使用します。
MAP空間：CPMSが使用するタスクやIRSUB、組み込みサブルーチンの管理テーブルが配置されます。
CPMS空間：CPMS専用の空間です。
タスク空間：タスクのTEXT, DATA, BSS, STACK, OSワークが割り当てられます。
GLBR：PU内タスク間共用メモリ（読み取りのみ）が割り当てられます。
GLBW：PU内タスク間共用メモリ（読み取り、書き込み可能）が割り当てられます。
IRSUB：タスク間共有の間接リンクサブプログラムが割り当てられます。
ユーザアクセス禁止領域：0x80000000～は、タスクがアクセスすることができません。アクセス時にはプログラムエラーとなります。

図１－22　論理アドレスマップ

３．２　メモリプロテクション

CPMSはメモリを4KBページ単位に書き込み、保護の管理をします。

表１－８にメモリアクセス権を示します。

タスクが書き込みできるメモリを以下に示します。その他はタスクの書き込みから保護されます。

- 自タスクのBSS, STACK（マルチタスクでは、BSSが共有されることに注意）
- GLBW, CMで論理空間と物理メモリがマッピングされている領域
- システムバスメモリ空間でのPI/O, サイクリック転写メモリ

CPMSは、ユーザのプログラミング系タスクからプログラムや保護されているデータを書き換えるためにwrtmemマクロを用意しています。このマクロを使用して書き込み保護されている主メモリに書き込みできます。

表１－８　メモリアクセス権

空間種別	アクセス者 / アクセスモード	CPMS システム	タスク ユーザ	備　考
タスク空間（ユーザ空間内）				
	自タスクのテキスト	R-X	R-X	
	自タスクのデータ	R-X	R-X	
	自タスクのスタック	RWX	RWX	
	自タスクのBSS	RWX	RWX	
	他タスクのテキスト	R-X	R-X	
	他タスクのデータ	R-X	R-X	
	他タスクのスタック	R-X	R-X	
	他タスクのBSS	R-X	R-X	マルチタスクの場合、RWX
ユーザ空間（タスク空間以外）				
	NX用ユーザエリア	RWX	RWX	
	HIFLOW空間	RWX	RWX	
	GLBW	RWX	RWX	
	GLBR	R-X	R-X	
	IRSUB	R-X	R-X	
	MAP	R-X	R-X	
	ｼｽﾃﾑﾊﾞｽ空間（ﾕｰｻﾞ用）	RWX	RWX	
	ｼｽﾃﾑﾊﾞｽ空間（ｼｽﾃﾑ用）	R-X	R-X	OSサブシステム（ドライバ）用
	CPMS空間	R-X	R-X	
	PCI空間（ﾕｰｻﾞ用）	RWX	RWX	
	PCI空間（ｼｽﾃﾑ用）	R-X	R-X	
	LPU空間	RWX	RWX	
カーネル空間				
	主メモリのV=R空間	RWX	---	CPMSのテキスト、データを含む
	I/Oレジスタ空間	RWX	---	カーネル、ドライバだけがアクセスできます。
	KROM空間	R-X	R-X	

R ：読み出し可　W：書き込み可　X：実行可

---：不許可（タスクがこのアクセスを実行すると、アボートされます。）

３．３　メモリアクセス時の異常処理

- メモリエラー

  ECC付メモリのマルチビットエラーが発生した場合は、システムが停止します。

- メモリシングルビットエラー

  ECC付メモリのシングルビットエラーは訂正され、リードデータは正しいのでエラーとしません。メモリをパトロールして、シングルビットエラーがあった場合には、再書き込みして訂正します。それでも訂正されずに再度シングルビットエラーとなる場合は、ソリッドな故障として、アラーム報告をエラーログします。

- システムバスアクセスエラー

  システムバス接続のI/Oを実装しないと、システムバスメモリ空間にマッピングされません。マッピングされていないアドレスをアクセスすると、プログラムエラーとなります。マッピングされているにもかかわらず、ハードウェアの不良によりシステムバス上でバスエラーとなった場合は、プログラムエラーとはならず、以下のようにターゲットアボートとなります。

  ・リードアクセスでは、全ビット1のデータが読み出されます。

  ・ライトアクセスでは、書き込みをしたかのようにプログラムは動作を継続します。

  ・ターゲットアボート発生によりCMUには割り込みが入りモジュールエラーとなります。

- ライトプロテクト（書き込み保護）エラー

  ソフトウェア不良により、ライトプロテクトされているアドレスに書き込むと、プログラムエラーとなり、タスクをアボートします。

３．４　システムバスアクセス手順

システムバス接続I/Oのサイクリック転写メモリは、バスメモリとしてユーザプログラムから直接アクセスされます。この場合、バスメモリのアクセスの障害を検出するために、以下のような手順が必要です。

バスメモリをアクセスするユーザプログラムは、該当スロットのバスメモリのアクセス可否をCHKBMEMマクロでチェックしてください。CHKBMEMマクロは、指定スロットのバスメモリの未実装、システムバスアクセスエラーによるターゲットアボートの有無を返します。CHKBMEMマクロによって異常が検出されたスロットのバスメモリには、アクセスしないようにしてください。

バスメモリをアクセスした後は、ターゲットアボート発生の有無をCHKTAERマクロでチェックしてください。これは、ターゲットアボートが発生してもタスクはアボートされないため、正常に実行されたかのようにタスクの処理が続行されるためです。

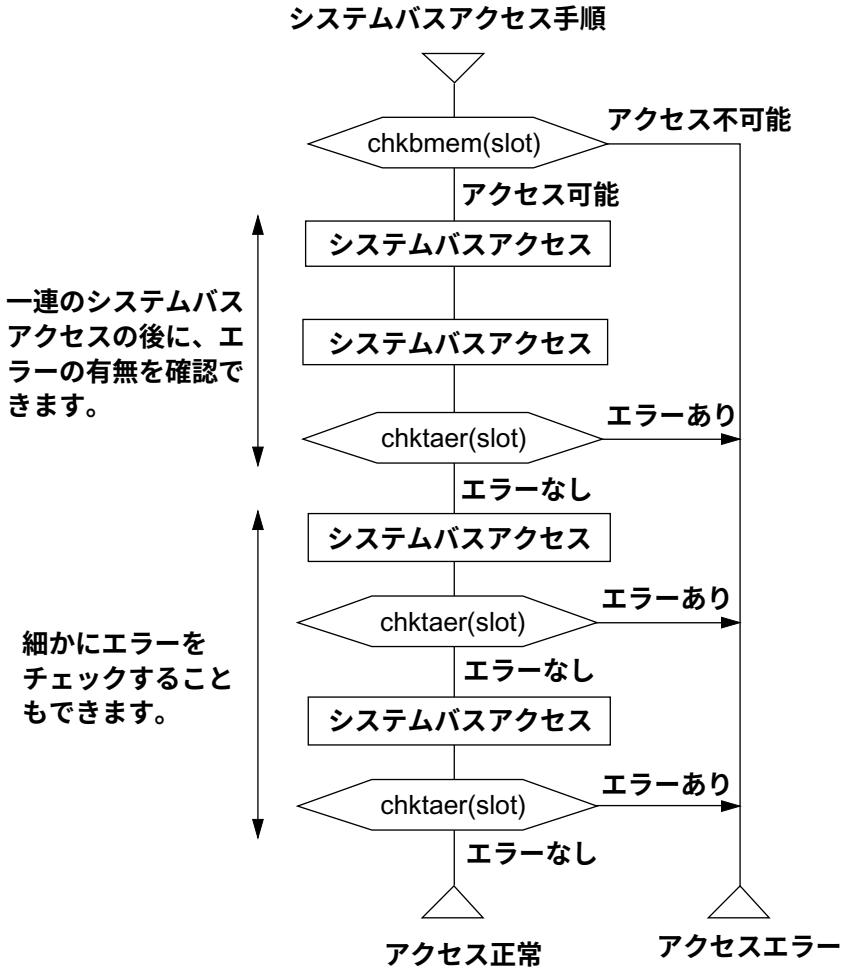


図１－23　システムバスアクセス手順

第４章　タイマ管理

４．１　時間と時刻

CPMSは時間と時刻を管理します。時刻は、西暦年、月、日、時、分、秒で表します。西暦年は1970年から2069年が対象です。時間はミリ秒で表します。

タスクはGTIMEマクロによりCPMSの管理する時刻を取り出せます。また、STIMEマクロによりCPMSの管理する時刻を設定できます。

CMUには、停電時にもバッテリで動作する時計（RTC: Real Time Clock）がありません。ただし、LPUにRTCがあります。CPMSは起動時にLPUのRTCから年、月、日、時、分、秒を読み取り、それを時刻の起点とします。CPMS動作中は、プロセッサに供給されるクロックによる内部タイマで時間と時刻を管理します。RTCと内部タイマは別々のクロックで動作しているため、長時間経過すると誤差が生じる可能性があります。CPMSは、一日に一度内部タイマによる時刻をLPUのRTCに設定することで誤差を修正します。

４．２　時間・時刻によるタスク制御

タスクは、DELAYマクロで指定の時間、自タスクの実行を抑止できます。また、TIMERマクロで、指定の時刻または時間経過後にタスクを起動し、さらに周期的にタスクを起動するタイマを作成できます。このタイマはCTIMEマクロで削除できます。TIMERマクロで作成するタイマに設定できる時刻はTIMERマクロ発行時点から24時間以内です。

４．３　時刻の変更

STIMEマクロにより時刻が変更された場合、TIMERマクロで時刻起動によりタスク起動を設定されたタイマの動作に影響します。時刻が進められて予定時刻を飛び越された場合は、最初の起動予定時刻が過ぎてしまい起動タイミングが失われたものは、時刻が変更されたときに起動されます。時刻周期指定の場合には、最初の起動予定時刻に周期時刻を加えていった時刻が、変更後の時刻以降となる時刻に起動予定時刻を移します。

予定時刻にタスクを起動したタイマは時刻が戻されても予定時刻に起動する再登録は行いません。

時間指定のタイマは時刻が変更されても起動時間は変更されません。

４．４　CMU、LPU間の時刻一致化

LPUはRTCを実装していますが、CMUはRTCを実装していないので、次のタイミングで、CMUの時刻がLPUの現在時刻に設定されます。

・CMUでSTIMEマクロ発行

・CMUの時刻一致化時（00時00分30秒）

基本システムで現在時刻を設定するとき、CMUの時刻が不連続になりますので注意してください。

第５章　共用資源管理

５．１　共用資源

タスク間で共用される資源には、主記憶装置，CPU，I/O，データエリア（GLB）などがあります。このうち、主記憶装置，CPU，I/Oについてはシステム側で排他制御をしていますが、GLBなどについてはユーザ側で排他制御してください。

図１－24は排他制御の必要性を示します。図１－25は、排他制御することにより資源競合による不具合を防止した場合を示します。

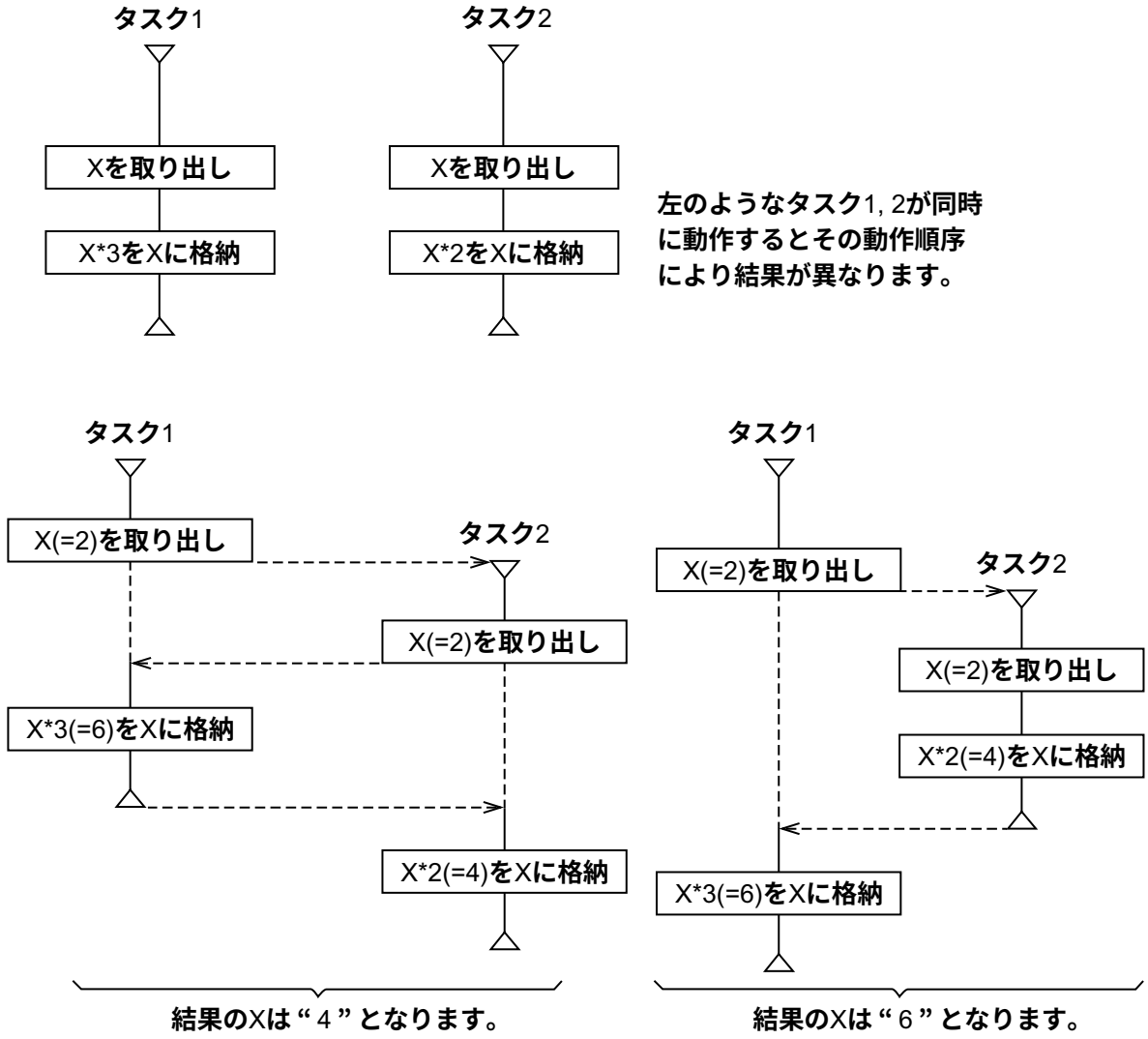
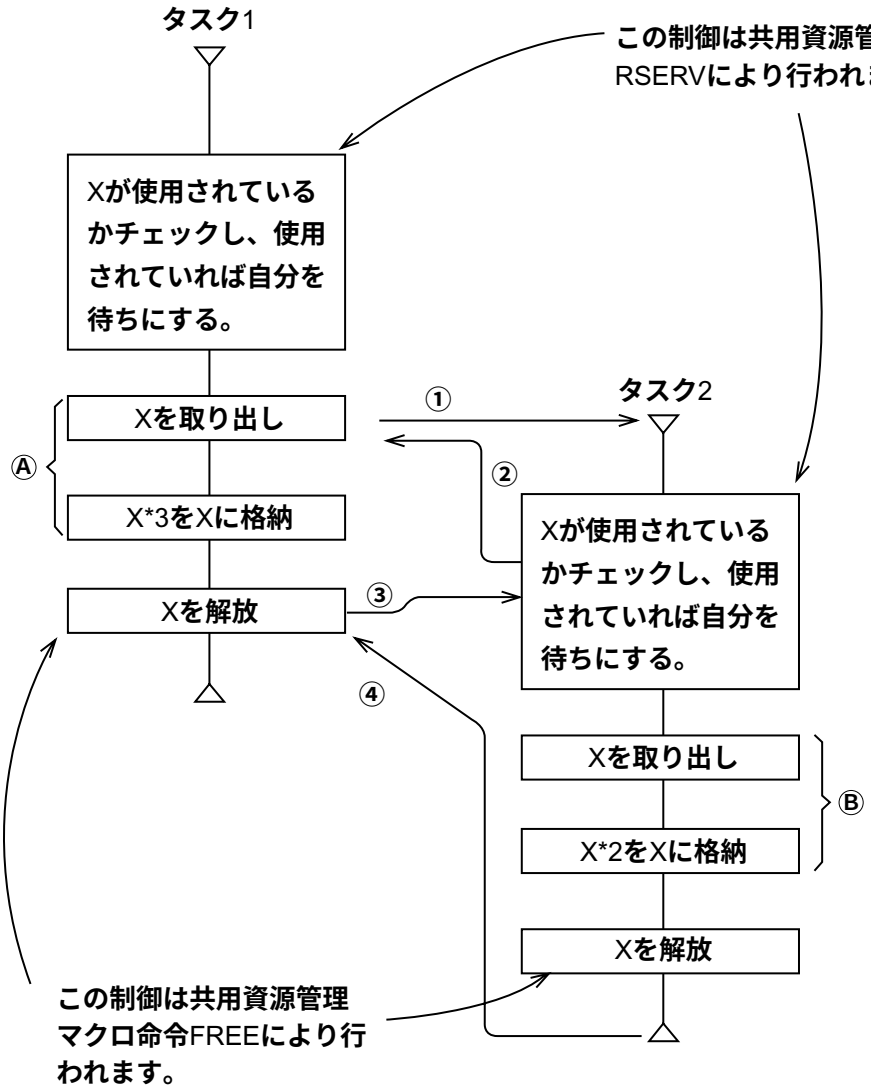


図１－24　排他制御が行われないときの不具合



制御は、①、②、③、④の流れで行われ、ⒶとⒷが同時に動くことは防止されます。

図１－25　共用資源管理マクロ命令による排他制御

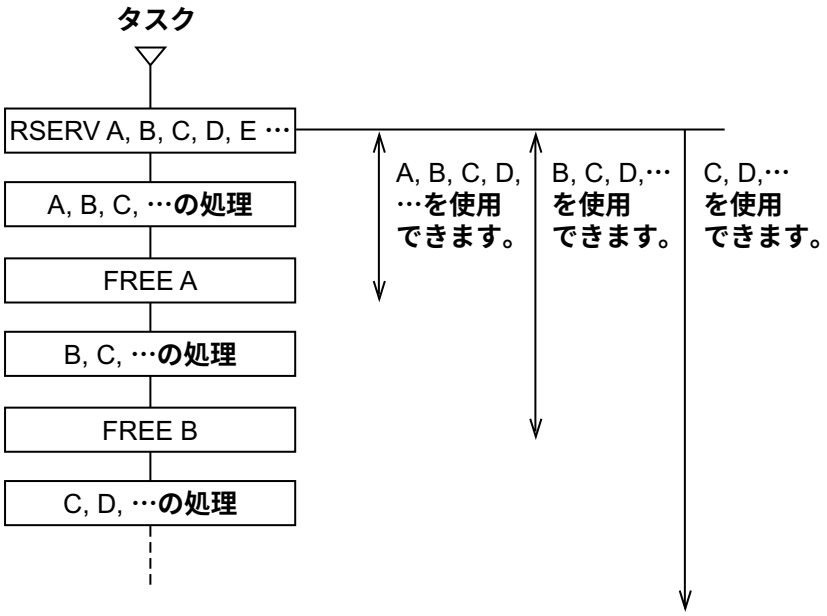
５．２　共用資源管理方法

タスク間共用資源であるGLBについては、物理的資源そのものが占有できます。すなわち、共用資源を管理するシステムテーブルにGLBのアドレス，大きさが登録され、RSERVマクロ命令によって占有要求が出されるごとにこのシステムテーブルが参照され、目的のGLBがすでに占有されているかどうかがチェックされます。もしすでに占有されていれば、その資源が解放されるまで要求タスクはRSERVマクロ命令で待ち状態となります。このタスクの待ち状態は、その資源が解放され使用可能となったときに解除されます。

複数のタスクが資源の解放待ちとなっているとき、その資源が解放されると待ちとなっていたタスクのうちで最もレベルの高いものに資源が割り当てられます。ただし、そのタスクが別の要因で動作できないときはこれにあてはまりません。

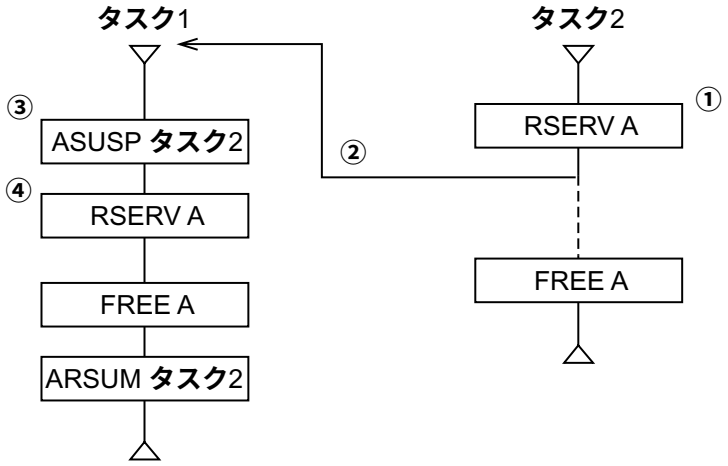
共用資源は、デッドロックを防止するためにそのタスクが必要とする資源は一度にすべて占有することを原則とします。このため、RSERVマクロ命令は多重発行（すでにRSERVマクロ命令で資源を占有しているタスクがRSERVマクロ命令を発行）を許しません。図１－26のように、必要な資源をすべて占有し、使い終わったらただちにFREEマクロ命令によって使用終了した資源を解放するようにします。

図１－27に、デッドロックの例を示します。この例のように、SUSPマクロ命令など他タスクの実行を抑止するマクロ命令を発行してから、RSERVマクロ命令を発行しないでください。



・タスクで使用する資源はすべて一度に占有し、使い終わったものから順次FREEマクロ命令により解放していきます。
・1つのFREEマクロ命令で複数の資源を一度に解放できます。

図１－26　RSERV/FREEの使い方



① タスク2が資源Aを占有します。

② Aを解放する前にタスク1に制御が移ります。

③ タスク1はタスク2をSUSPします（これによってタスク2は動作不可となりAを解放できなくなります）。

④ タスク1は資源Aを占有しようとしますが、すでにタスク2に占有されています。タスク1は待ち状態となり、タスク2をRSUMできません。

⑤ これによりタスク1, 2の両方とも実行不可の状態となります。

図１－27　デッドロックの例

５．３　PRSRV/PFREEマクロによる共用資源排他制御

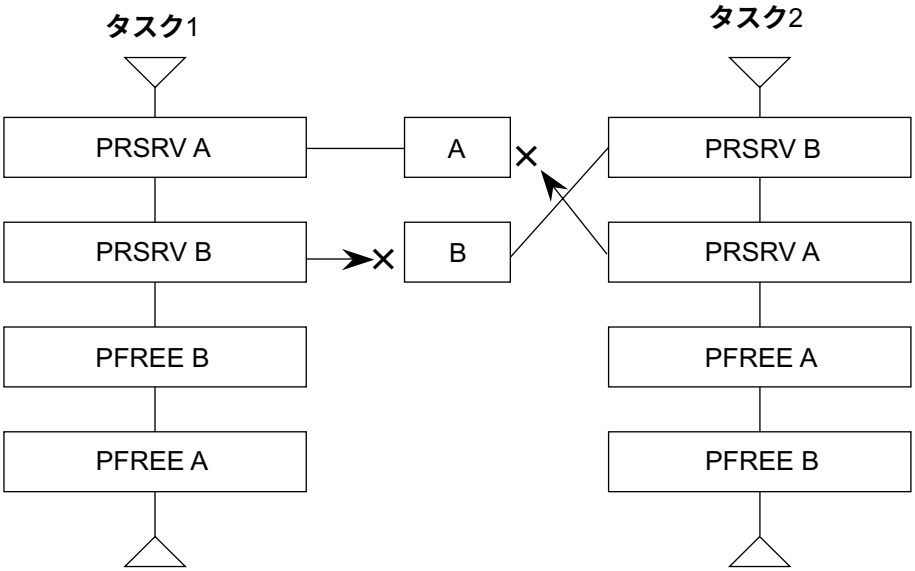
PRSRV/PFREEマクロを用いて、タスク間での共用資源の排他制御を、RSERV/FREEマクロよりもきめ細かく行うことができます。

占有開始：GLBのSAREAと占有範囲を指定して、PRSRVを発行します。

占有終了：GLBのSAREAと占有範囲を指定して、PFREEを発行します。

指定したGLBエリアを占有できないときは、共用資源占有が解除されるまで、PRSRVを発行したタスクに制御が戻りません。

タスクは何度もPRSRVを発行できます。複数の共用資源について、複数回に分けて徐々に占有できますので、占有待ちが発生する機会を減らすことができます。ただし、共用資源の占有順序を明確にして、デッドロックが起こらないように注意してください。



タスク1が資源Aを占有し、タスク2が資源Bを占有している状態で、
タスク1が資源Bの解放を待ち、タスク2が資源Aの解放を待ちます。
互いに資源の解放を待っているので実行不可の状態となります。
この不具合は、同じ資源を占有する場合、同じ順序で占有することで、避けることができます。

図１－28　PRSRVによるデッドロックの例

第６章　入出力デバイス管理

６．１　入出力デバイス管理機能の構造

CPMSは、入出力デバイスを制御するサブシステム（I/Oドライバ）に対し、入出力デバイス管理の基本機能を提供します。ユーザは、各サブシステムが提供するインタフェースを使用して、入出力を行ってください。

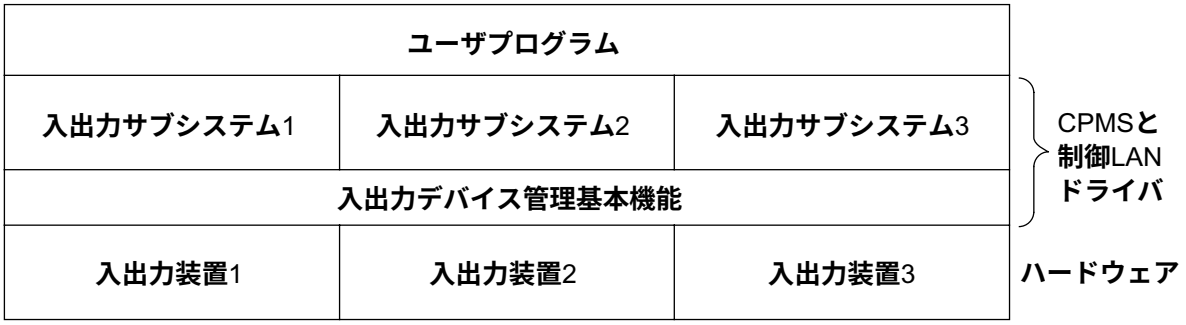


図１－29　入出力デバイス管理機能の構造

６．２　入出力ユニット番号

CPMSでは、システムバス接続I/Oをユニット番号（UNOと略します）で入出力の対象（デバイス）を識別します。ユニット番号は、接続したスロット番号に4を足した番号が割り当たります。

６．３　デバイス番号

デバイス番号は、論理デバイスとそれを制御するドライバを識別するために使用されます。

論理デバイスは、あるデバイスに対する用途を定義するもので、1つのデバイスに対して複数個の論理デバイスが定義されることがあります。

デバイス番号は、メジャー番号とマイナー番号で構成されます。メジャー番号は、デバイスを制御するサブシステムを識別するものです。マイナー番号は、デバイスの接続位置と用途を特定するものです。デバイス依存部はサブシステムごとに定義されます。

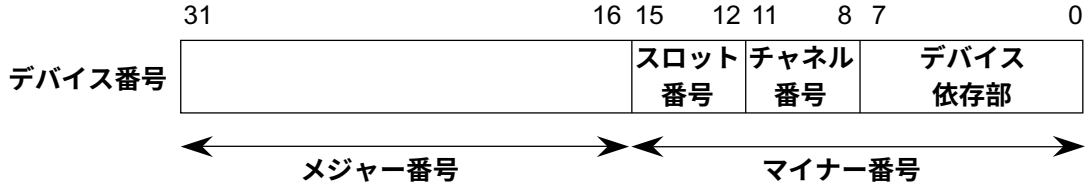


図１－30　デバイス番号

第７章　システム管理

７．１　CPMSの立ち上げ・停止の状態遷移

７．１．１　立ち上げ・停止の状態遷移

図１－31にCPMS立ち上げ・停止の状態遷移を示します。表１－９に状態の説明、表１－10にイベントの説明を示します。

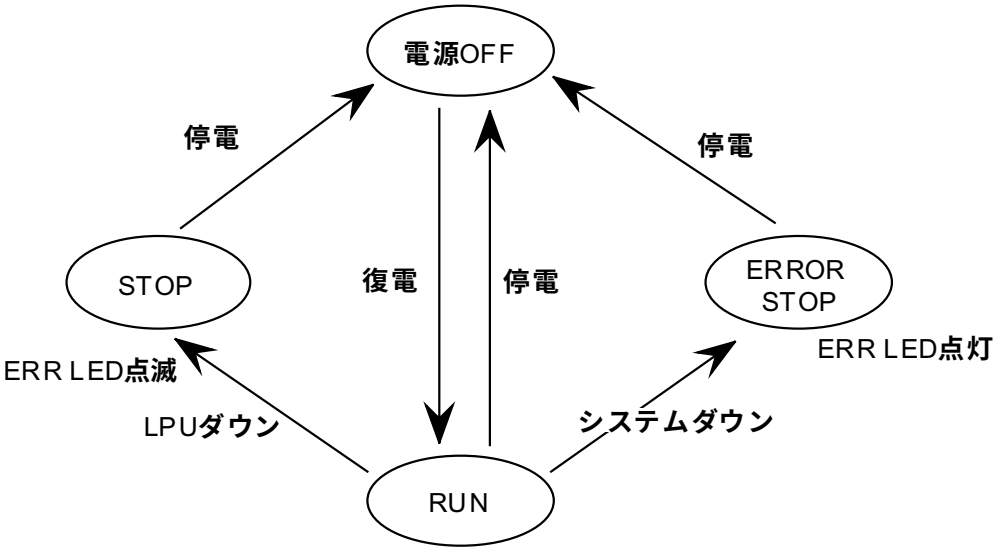


図１－31　CPMS立ち上げ・停止の状態遷移

表１－９　立ち上げ・停止の状態

状　態	説　　明
電源OFF	電源がOFFの状態です。
STOP	LPUがダウンした状態です。
ERROR STOP	システムプログラムエラーで停止している状態です。
RUN	システムプログラムが実行している状態です。

表１－10　立ち上げ・停止のイベント

イベント	説　　明
復電	電源をOFFからONにします。
停電	電源をONからOFFにします。
システムダウン	エラーによりシステムプログラムが停止することです。

７．１．２　立ち上げ操作

ハードウェアの最初の状態は、電源OFF（メモリ消去）です。この状態から復電すると、CPMSが起動され、RUN状態となります。ROMカードにデータを一度ダウンロードしておけば、このタイミングでプログラム格納メモリ上のプログラム、データもメモリにコピーされます。

７．１．３　停止操作

電源OFFでシステムの停止を行います。

７．２　組み込みサブルーチンINSとイニシャルスタートタスク

CPMSは、OSスタート処理の最後に以下の順番で処理します。

①　組み込みサブルーチンINSにリンクします。

②　システムイニシャルスタートタスク（SIST：タスク番号255）を起動します。

③　ユーザイニシャルスタートタスク（UIST：タスク番号1）を起動します。

CPMSは、組み込みサブルーチンINSのパラメータおよびイニシャルスタートタスクの起動要因として表１－11に示す立ち上げ要因の番号を渡します。

表１－11　立ち上げ要因

番号	立ち上げ要因	説　　明
1	IPLスタート	この要因で常に立ち上がります。

７．３　ウォッチドッグタイマ（WDT）

７．３．１　WDTの機能

CPMSでは、タスクが無限ループになることを監視するためにウォッチドッグタイマ（WDT）を使用します。WDTはタスクの実行時間が異常に長くかかり、プラント制御に間に合わなくなったことを検出できます。WDTがタイムアウトすると、組み込みサブルーチンWDTESへリンクします。ユーザはWDTESにエラー処理プログラムを登録できます。また、WDTESのリターン値でCPU停止を指示できます。

WDTESに登録されていない場合は、CPMSはWDTがタイムアウトしてもタスクアボートやCPU停止を行いません。

７．３．２　WDTの使い方

WDTを使用するときは、実行時間を監視する1つのタスクから、WDTの設定時間より短い周期でWDT制御マクロ（WDTSET）を発行するようにします。このタスクおよびそれより優先レベルの高いタスクの実行時間が延びてしまったとき、このタスクによりWDTが新たに設定されないので、WDTタイムアウトになります。

イニシャルスタートタスク起動時には、WDTはまだスタートしていません。ユーザプログラムから最初にWDTSETマクロを発行したときからWDTがスタートします。また、WDTSETマクロで、設定時間を0とすると、WDTはタイムアウトせずに停止します。

CPMSが用いるWDTは1つだけなので、WDTにより実行時間を監視できるタスクは1つだけです。したがって、プラント制御にかかわるタスクを監視するタスクを1つ設け、この監視タスクをWDTで監視してください。

第８章　タスクの異常処理

タスク実行中の異常処理について、基本的な考え方を以下に示します。

・タスクの異常を検出した場合は、そのタスクの実行を打ち切ります。ただし、タスクの回復ポイントに戻ることで、実行継続できる方法もあります（「８．５　プログラムエラー回復処理」参照）。

・タスクの実行を妨げないハードウェアエラーでは、タスクの実行は継続します。タスクはハードウェアエラー情報を得て、エラー処理ができます。

・タスクの異常処理は、組み込みサブルーチンによって行います。異常となったタスクのタスク番号は、組み込みサブルーチンの入力パラメータで渡されます。

８．１　組み込みサブルーチンのレパートリ

CPMSには、システム処理の一部をユーザが作成できるように組み込みサブルーチンの仕組みがあります。

各組み込みサブルーチンポイントのエントリ数は、ミドルウェア・OS用2、ユーザ用2の計4つです。

エントリ番号は、1と2＝ミドルウェア・OS用、3と4＝ユーザ用で、1→2→3→4の順序でリンクします。

表１－12　組み込みサブルーチンレパートリ

組み込みｻﾌﾞﾙｰﾁﾝ名	リンクタイミング	入力情報	出力情報	マクロ発行の可否	エントリ数
CPES	プログラムエラー	PRGEB	あり	可	4
IES	I/Oエラー	IOERB	あり	可	4
EAS	エラーログ	ADB	あり	可	4
INS	ISTスタート前	立ち上げ要因	なし	不可	4
EXS	タスクEXIT	タスク番号	なし	可	4
ABS	タスクABORT	タスク番号	なし	可	4
PCKS	マクロパラメータエラー	SVCEB	あり	可	4
MODES	モジュールエラー	HARDEB	あり	可	4
WDTES	WDTタイムアウト	なし	あり	可	4
ADTS	ADT例外発生	ブレーク情報	なし	可	4

８．２　組み込みサブルーチンの実行環境

CPMSは、組み込みサブルーチンをシステムモード・割り込み禁止で実行します。また、実行優先レベルはすべてのタスクよりも高く設定されます。

組み込みサブルーチンの実行環境には、以下の制約条件があります。

・組み込みサブルーチンが使用するスタックエリアとして、ユーザは、1KB以内を目安に使用してください。組み込みサブルーチンのスタックエリアがオーバフローした場合は、CPU停止します。

・組み込みサブルーチンは、イベントのログ、GLBのアクセス、他タスクの起動・停止という処理を行うものとし、実行中の組み込みサブルーチンを待ちにしたり停止したりするような処理を行わないでください。

・割り込み禁止時間を制限するために、組み込みサブルーチンの実行時間は1ミリ秒以内としてください。

・組み込みサブルーチンから呼び出してもよいマクロは、rleas, queue, abortです。

・組み込みサブルーチン内で浮動小数点演算を使用できません。浮動小数点演算を使用すると、CPU停止になります。

・組み込みサブルーチン内でプログラムエラーが発生すると、CPU停止になります。

８．３　組み込みサブルーチンのリンク処理

組み込みサブルーチンINS, ABS, EXS, CPES, PCKS, WDTESとEASのリンク処理を図１－32に示します。

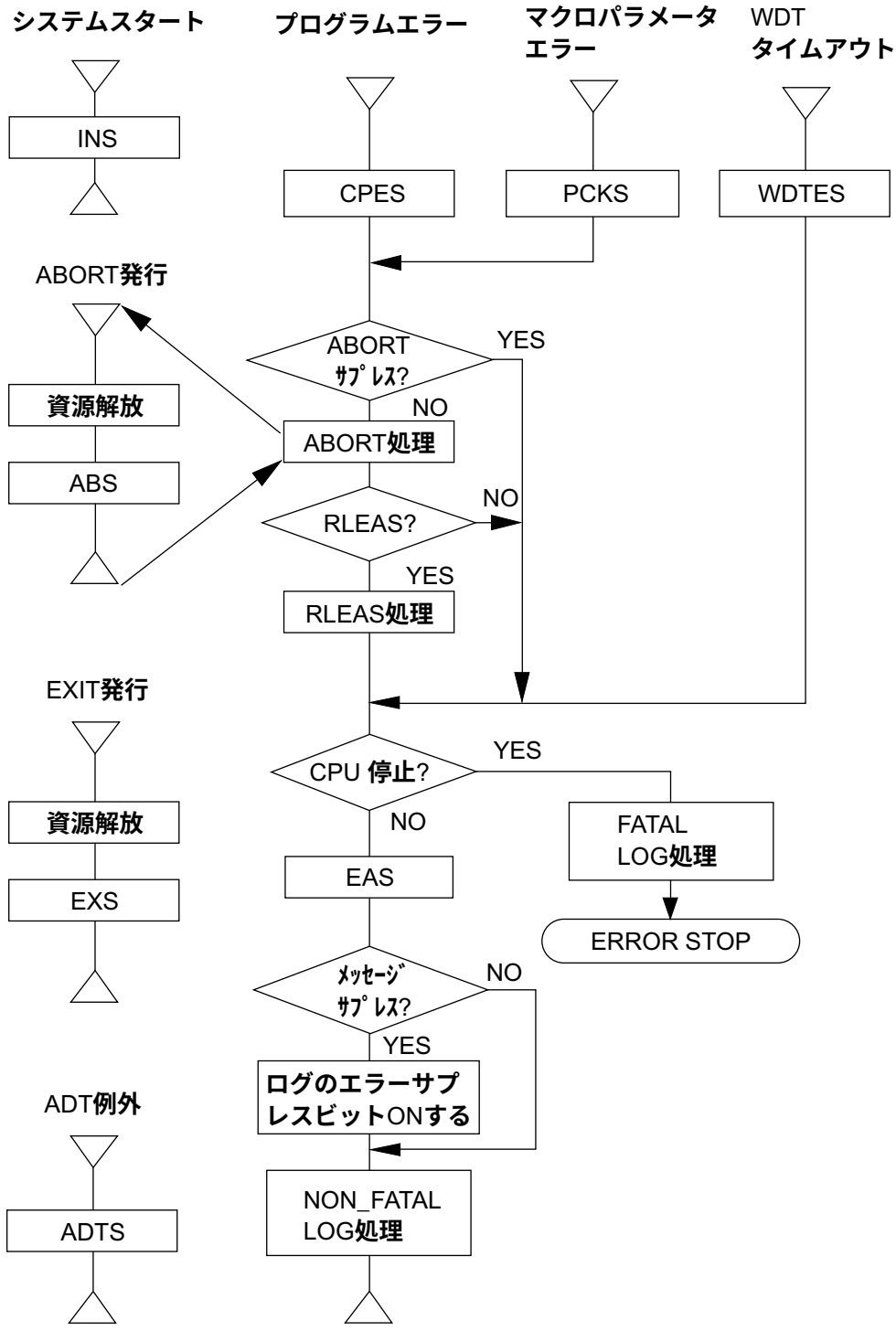


図１－32　組み込みサブルーチンリンク処理（1）

組み込みサブルーチンIES, PIOS, MODESとEASのリンク処理を図１－33に示します。

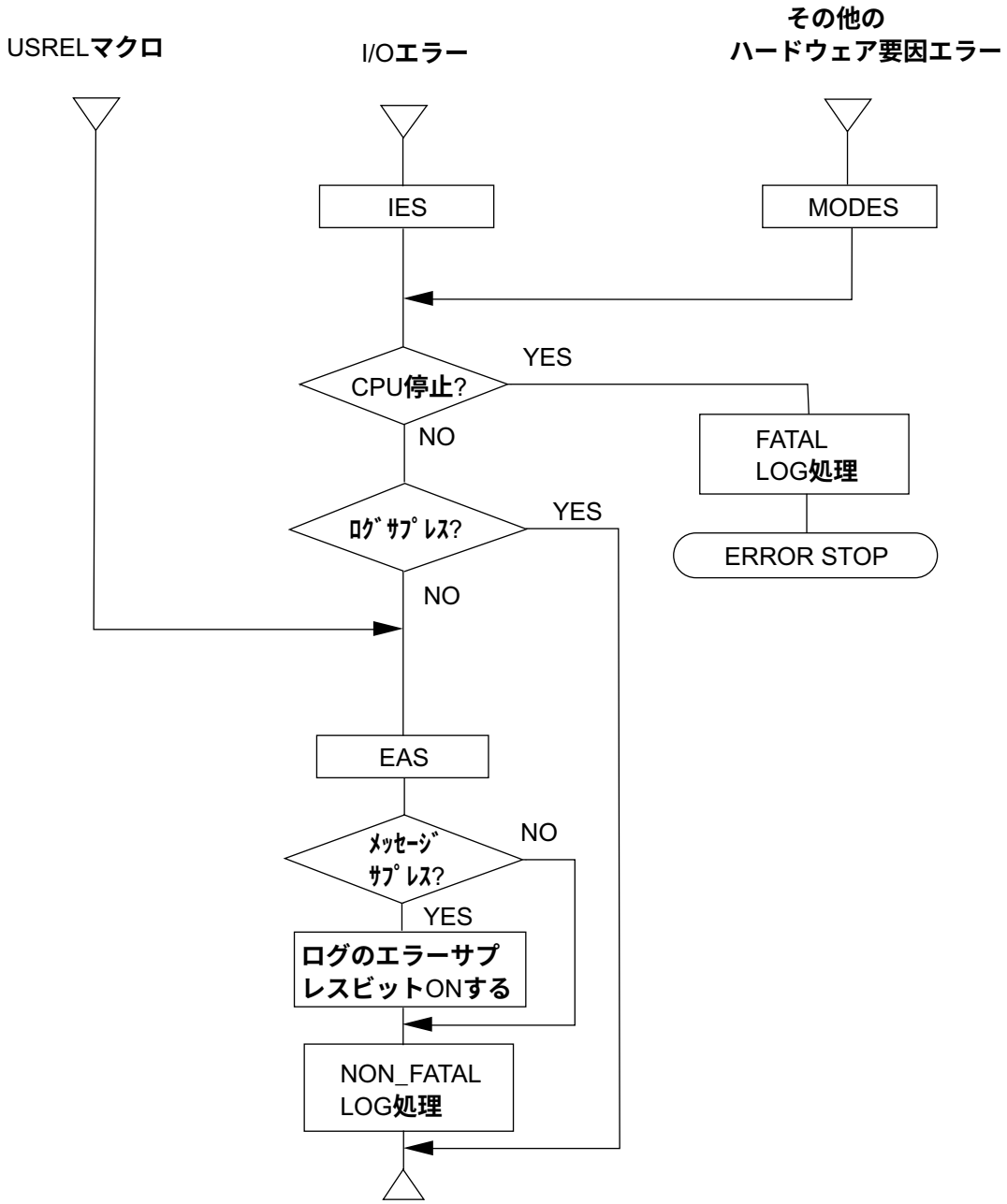


図１－33　組み込みサブルーチンリンク処理（2）

８．４　組み込みサブルーチンのリンケージ

#include <cpms_ulsub.h>

- CPES - CPU Error Subroutine

  int cpes(prgeb)

  struct PRGEB *prgeb;　　/* Program Error Block */

- IES - I/O Error Subroutine

  int ies(ioerb)

  struct IOERB *ioerb;　　/* I/O Error Block */

- EAS - Error Alert Subroutine

  int eas(adb)

  struct ADB *adb; /* Alert Data Block */

- INS - Initial Start Subroutine

  int ins(reset)

  long reset; /* システムスタート要因 */「表１－11　立ち上げ要因」を参照してください。

- EXS - Exit Subroutine

  int exs(tn)

  long tn; /* Task Number */

- ABS - Abort Subroutine

  int abs(tn)

  long tn; /* Task Number */

- PCKS - Parameter Check Subroutine

  int pcks(svceb)

  struct SVCEB *svceb; /* SVC Error Block */

  long piono, isw, idp;

- MODES - Module Error Subroutine

  int modes(hardeb)

  struct HARDEB *hardeb;

- WDTES - WDT Error Subroutine

  int wdtes( )

- ADTS - ADT Subroutine

  int adts(adtdb)

  struct ADTDB *adtdb;

（注1）CPES, PCKSが発生した場合には、デフォルト処理としてタスクをアボートします。タスクアボートを抑止したい場合には、出力情報のULSUB_OUT_ABORTSUPRESビットをONしてください。

（注2）WDTESはシステムウォッチドッグタイマタイムアウトエラー時に発生します。
タスク単位の終了監視ではありません。

（注3）システムタスクの場合、EXSまたはABSにはリンクしません。

入力情報は、「付録Ｃ　組み込みサブルーチンの入力データ」を参照してください。

＜出力情報（リターン値）＞

組み込みサブルーチンの出力情報（リターン情報）はすべて共通フォーマットとし、ビット判定します。また、エントリポイントが複数あるので、各サブルーチンの出力情報のORを取ります。

```
#define ULSUB_OUT_LOGSUPRES      0x00000010    /* エラーログをサプレスする          */
#define ULSUB_OUT_MSGSUPRES      0x00000020    /* エラーメッセージをサプレスする*/
#define ULSUB_OUT_RLEAS          0x00000040    /* タスクをリリースする              */
#define ULSUB_OUT_ABORTSUPRES    0x00000080    /* タスクアボートをサプレスする    */
#define ULSUB_OUT_CPUDOWN        0x00000100    /* CPUダウンする                     */
```

上記ビットの有効／無効は下表に示すように組み込みサブルーチン種別ごとに異なります。

出力情報がありの場合には、どのビットが有効かを○で示します。

表１－13　組み込みサブルーチンの出力情報一覧

	CPES	IES	EAS	INS	EXS	ABS	PCKS	MODES	WDTES	ADTS
出力情報の有無	有	有	有	無	無	無	有	有	有	無
ULSUB_OUT_ABORTSUPRES	○	×	×	×	×	×	○	×	×	×
ULSUB_OUT_RLEAS	○	×	×	×	×	×	○	×	×	×
ULSUB_OUT_LOGSUPRES	×	○	×	×	×	×	×	○	×	×
ULSUB_OUT_MSGSUPRES	×	×	○	×	×	×	×	×	×	×
ULSUB_OUT_CPUDOWN	○	○	×	×	×	×	○	○	○	×

(1) ULSUB_OUT_ABORTSUPRES

引数で指定されたタスクのアボートをサプレスします。組み込みサブルーチンCPESとPCKSで有効です。

ULSUB_OUT_CPUDOWNがONで、ULSUB_OUT_ABORTSUPRESがOFFの場合には、先にタスクをアボートした後CPU停止します。

(2) ULSUB_OUT_RLEAS

引数で指定されたタスクをリリースします。組み込みサブルーチンCPESとPCKSで有効です。

タスクをアボート／リリースしたい場合には、ULSUB_OUT_RLEASビットをONに、ULSUB_OUT_ABORTSUPRESビットはOFFとしてください。

(3) ULSUB_OUT_LOGSUPRES

組み込みサブルーチンMODESとIESで有効です。モジュールエラーあるいはI/Oアクセスで正常処理された場合には、このビットを立ててください。このビットはCPMSあるいはI/Oドライバにおいて判定され、ビットが1の場合にはエラーログの処理をスキップします。

（４） ULSUB_OUT_MSGSUPRES

組み込みサブルーチンEASでのみ有効です。このビットが1の場合には、エラー情報の中のメッセージサプレスフラグを1にします。実際のメッセージのサプレスは表示プログラムの処理なので、表示プログラムはエラー情報のメッセージサプレスフラグを見て処理してください。エラーログは行います。

（５） ULSUB_OUT_CPUDOWN

CPU停止（ERROR STOP）します。

８．５　プログラムエラー回復処理

タスクのプログラムエラーに対し、あらかじめ設定した回復ポイントに戻ることでタスクの実行継続を可能とします。ただし、回復ポイントを含むルーチン、または回復ポイントを含むルーチンから呼び出したサブルーチンでのプログラムエラーを対象とします。

・save_envを呼び出して、回復ポイントの実行環境データをGLBに退避します。

・組み込みサブルーチンCPESでresume_envを呼び出して、CPES実行後に回復ポイントへ制御を戻します。

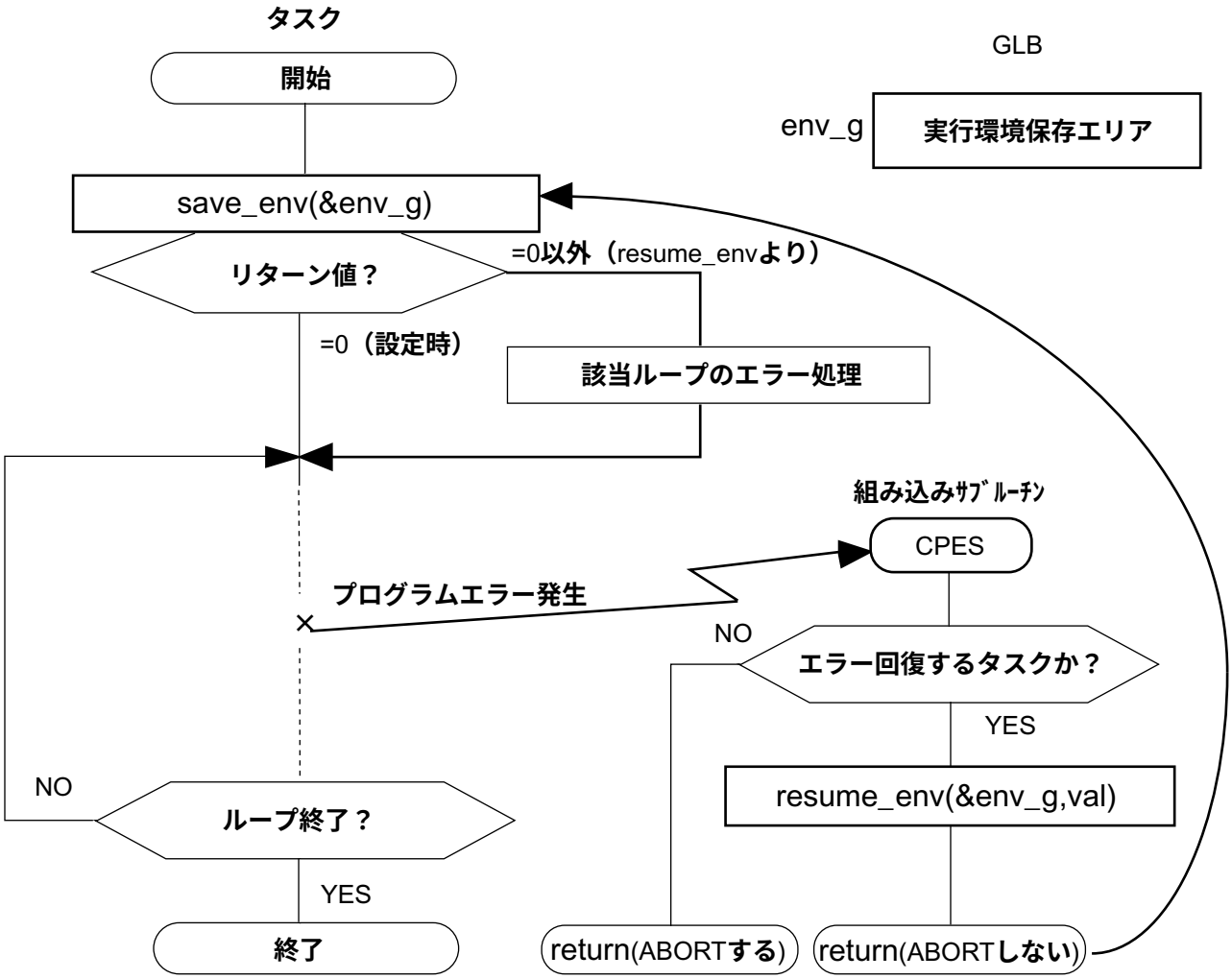


図１－34　プログラムエラー回復処理

(１) 使い方

- save_env(&env_g)を呼び出して、エラー回復ポイントの実行環境をGLBに割り当てたenv_gに保存します。このときsave_envは0を返します。
- タスクでプログラムエラーが発生すると、組み込みサブルーチンCPESにリンクします。
- 組み込みサブルーチンCPESに登録したユーザ組み込みサブルーチンで、エラー発生タスクがエラー回復すべきタスクの場合は、resume_env(&env_g, val)により、タスクをエラー回復ポイントから再開するように設定し、タスクをアボートしないようにします。valは0以外としてください。
- タスクをアボートしないで組み込みサブルーチンから戻ると、タスクは回復ポイントから再開します。このときsave_envはvalを返します。
- タスクはsave_envのリターンコードにより、CPESからリターンしたことを判定し、エラーに対する後処理を実行できます。

(２) 注意事項

- エラー発生時には、回復ポイント設定時点でのスタック内容は破壊されていないことを前提とします。スタック破壊やプログラム破壊によるプログラムエラーは回復できません。
- resume_envにより回復したときには、外部変数や静的変数や自動変数は回復していません。例えば左記のループ回数はそのままなので、エラー発生した該当ループが判ります。逆に、この変数が破壊されるようなエラーは、エラー回復ポイントに戻っても処理を正しく継続できません。
- 回復できないプログラムエラーに対して、CPESにてこの回復処理を行うと、プログラムエラーと回復処理を繰り返す無限ループになる場合があります。CPESにて、エラー回復するタスクかどうかを判定し、回復すべきプログラムエラーの条件を限定してください。
- resume_envでは必ずエラーを起こしたタスクがsave_envした実行環境保存エリア（env_g）を指定してください。そうしなければ、正しく回復ポイントへ戻れません。

第９章　システムサービス

９．１　DHP

CPMSは、ある決められた処理ポイントにおいて、そのポイントを通過したことを主メモリ上のバッファに逐次記録します。この記録をDHP（Debugging Helper）と呼びます。DHPバッファはカーネルワーク内にあり、CPMSがDHP処理をサポートします。

(１)　記録ポイント

下記のポイントにてDHPを記録します。

・原則として、全CPMSマクロ発行時、パラメータを取り込んだ後で記録します。

マクロ処理終了時に処理結果（例えば、GFACTでユーザに返す起動要因など）を、DHPとして記録する場合もあります。

・タスクのスイッチ処理前後
・タスクの起動／終了処理
・I/Oの起動処理と終了割り込み処理
・タスクの異常処理
・OS／ハードウェアの異常処理
・USRDHPマクロにより、ユーザの情報を記録できます。

(２)　記録内容

DHPの各ポイントで、下記のデータを記録します。

・DHPポイントを表すコード（4バイト）
・DHP記録時刻（4バイト）
・タスク番号とタスクの優先度（それぞれ2バイト）
・解析に必要なデータ （可変長：0バイトから最大20バイト）

(３)　記録モード

デフォルトでは、常に記録状態です。RPDPのsvdhpコマンドによって、DHP停止・再開を制御できます。

(４)　DHPバッファ

DHPバッファは、主メモリを128KB使用します。

(５)　記録内容の出力

・RPDPのsvdhpコマンドで、現在のDHPデータを取り込むことができます。
・エラーログを記録するときに、最新のDHPデータを合わせてログします。

９．２　CPU負荷率

CPUの負荷率およびタスクごとのCPU実行時間の測定をサポートします。

（１）CPUの負荷率

getsysinfoマクロのSYS_IDLE機能で、CPUのIDLE時間の累積を得ることができます。IDLE時間は常に累積するので、前回SYS_IDLEを発行した時点のIDLE時間の累積と現在のIDLE時間の累積の差でIDLE時間を求めてください。IDLE時間の累積は32ビットでオーバフローするので、差を求めるにはオーバフローを考慮してください。すなわち、前回値より今回値が大きければ今回値より前回値を引き、前回値より今回値が小さければ今回値に前回値の2の補数（反転＋1）を加えてください。

測定時間は、24時間以内としてください。

CPU負荷率は以下のように算出してください。

CPU負荷率＝（測定時間－IDLE時間の累積の差）／（測定時間）

第２編　マクロ仕様

第1章　総　　説

1.1　マクロ命令

マクロ命令は、ユーザプログラム（タスク）からCPMSに処理を依頼するための命令です。ユーザプログラムではサブルーチンの呼び出しとしてマクロ命令を記述します。このサブルーチンは、CPMSマクロリンケージライブラリにより自動的にCPMS呼び出し命令であるtrap命令に展開されます。プログラムがマクロ命令を発行するとそのマクロ命令は、このtrap命令によりCPMSへリンクし、CPMS処理が行われます。

1.2　CPMSマクロリンケージライブラリ

CPMSマクロリンケージライブラリは、CPMSのマクロ命令を使用するときにユーザプログラム内に記述されたマクロ命令をtrap命令に展開するためのサブルーチンです。CPMSマクロリンケージライブラリは、呼び出されるとパラメータ（引数）を各マクロ命令ごとに定められた順番にユーザスタックエリアに格納し、その後trap命令を発行します。これを図２－１に示します。

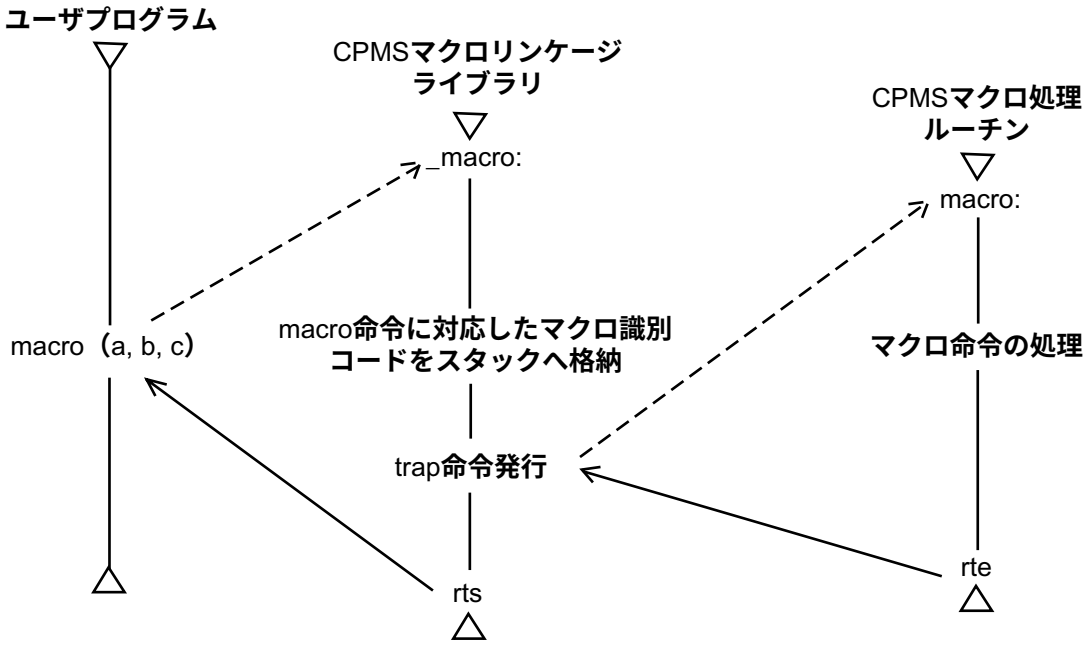


図２－１　CPMSマクロリンケージライブラリの働き

1.3　マクロ命令の一般規則

(1) パラメータの渡し方

CPMSマクロリンケージライブラリを使用する場合、パラメータはアドレス渡しまたは内容渡しとなります。例えば、C言語でユーザプログラムを作成する場合は、次のように訂正してください。

```
long    tn ;

tn ＝ 100 ;

abort(&tn) ;
```

ABORTマクロ命令の場合、tn（＝100）が格納されたアドレスを引数に記述します（&tnはtnへのポインタ、tnが格納されているアドレスを示します）。これをabort（tn）とは記述しないでください。

C言語の場合はこの他にもいろいろな記述方法がありますので、コーディングしやすい方法でプログラムを作成してください。コーディング例を以下に示します。

- パラメータが配列全体であるとき

```
long      x[n] ;
macro     (x) ;
```

- パラメータが配列の要素の1つであるとき（下の3つは同等です）

```
long      x[n] ;        long     x[n] ;        long     *x[i] ;
x[i]＝100 ;             x[i]＝100 ;            *x[i]＝100 ;
macro(&x[i]) ;          macro(x＋i) ;          macro(x[i]) ;
```

- パラメータが単純変数のとき（下の2つは同等です）

```
long      x ;           long     *x ;
x＝100 ;                *x＝100 ;
macro(&x) ;             macro(x) ;
```

(2) リターンコード

マクロ命令の実行結果は、リターンコードとしてCPMSのマクロ処理モジュールから戻されます。マクロ命令を関数として使用する場合、マクロ命令の処理の結果は以下のようにリターンコードにより判定できます。

```
long macro(    ) , rtn ;
long *x ;

rtn = macro(x) ;
if(rtn)
   {

   }
```

（注）マクロ命令の処理が正常に行われると、リターンコードは通常0を返します。しかし、マクロ命令によっては正常に処理が行われてもリターンコードが0以外の値を返すものもあります。

１．４　マクロ命令のパラメータチェック

マクロ命令はユーザプログラムとCPMSの直接のデータのやりとりです。パラメータを誤るとシステムの誤動作，システム停止となる場合があります。

CPMSのマクロ命令は、そのパラメータの主要なものをソフトウェア上で合理性をチェックします。チェックにより不合理とみなされると、マクロパラメータエラーとして報告され、マクロ命令発行タスクはABORTされます。

表２－１は、パラメータチェックが行われるときのタスク番号（TN）の関係について示しています。

各マクロでのパラメータチェックで、ユーザ最大タスク番号は224です。メモリプロテクトのチェックは、CPMSがタスクごとに設定した内容（アクセス可否）により行われます。

表２－１　パラメータチェックにおけるTNの関係

対象TN / 発行元TN	ユーザタスク 1〜224	システムタスク、ROMタスク 225〜255	異常なTN 256〜
1〜224	○	△	×
225〜255	○	○	×

発行元TNを付けたタスクが、対象TNを付けたタスクを対象としてパラメータチェックを行うマクロ命令を発行したとき、○、△、×は以下のことを示します。

○：それぞれ正常に処理されます。

△：処理は行われますが、発行しないでください。

×：パラメータエラーとして検出されます。

対象TN＝0の場合は、CPMSはパラメータエラーとはしませんが、処理は行わずリターンコードは1を返します。

1.5 CPMSマクロ一覧

(1) タスク管理

rleas
queue
exit
abort
wait
post
susp
rsum
asusp
arsum
chap
sfact
gfact

(2) メモリ管理

wrtmem
chkbmem
chktaer
mvmem
uspchk

(3) タイマ管理

timer
ctime
delay
stime
gtime
wake
cwake

(4) 共用リソース管理

rserv
prsrv
free
pfree

(5) システム管理

wdtset

(6) システムサービス

getsysinfo
gettaskinfo
gtkmem
usrdhp
usrel
save_env
resume_env
gettimebase
TimebaseToSecs
atmswap
atmand
atmor
atmxor
atmadd
atmtas
atmcas

＜コーディング上の注意＞

CPMSは、以下のインクルードファイルを提供します。

・cpms_types.h　：マクロで使われる変数タイプを定義しています。

・cpms_macro.h：マクロの関数を定義しています。

・cpms_errno.h　：マクロのリターンコードを定義しています。

・cpms_table.h　：CPMS内のテーブルの構造体を定義しています。

・cpms_dhp.h　　：DHPで使われるコードを定義しています。

・cpms_elog.h　：エラーログで使われるコードや構造体を定義しています。

・cpms_ulsub.h　：組み込みサブルーチンで使われるコードや構造体を定義しています。

システム管理のマクロを使用する場合にはロード時（svload）に-lsysctlを付加してください。

名　称

rleas　－　タスクを起動待ち状態にする

C形式

```
int rleas(&tn)
long tn;
```

機能説明

rleasは、パラメータtnで指定された対象タスクがDORMANT状態かどうかをチェックし、DORMANT状態ならばIDLE状態にします。対象タスクがDORMANT状態でないときは何もしません。

パラメータtnには、対象タスクのタスク番号を指定してください。

対象タスクが入出力処理を実行中にアボートされた場合、タスクはDORMANT状態になりますが、アボート処理は入出力処理が終了してから行われます。DORMANT状態で入出力処理を実行中のタスクにrleasを発行すると、rleasはリターンコード＝0でただちに正常終了しますが、対象タスクは入出力処理が終了し、アボート処理が終了した後でIDLE状態になります。

診　断

処理が正常終了すると、リターンコード0が返されます。そうでない場合は、以下のリターンコードが返されます。

1：tn＝0です。

3：tnで指定されたタスクは、DORMANT状態ではありません。

4：tnで指定されたタスクは、未登録です。

パラメータチェック

以下のパラメータチェックを行い、異常ならばパラメータチェックエラーとして報告します。

・tnで指定されたタスク番号が、0≦タスク番号≦最大タスク番号であること。

名　称

queue　―　タスクを起動する

C形式

```
int queue(&tn, &fact)
long tn, fact;
```

機能説明

queueは、パラメータtnで指定された対象タスクがIDLE状態のとき、対象タスクをCPU実行待ち状態にします。タスクをCPU実行待ち状態にすることを、タスクを起動するといいます。

CPU実行待ち状態のタスクは、優先レベル順にディスパッチされます。したがって、対象タスクの優先レベルが、rleasを発行したタスクより高いと、対象タスクにディスパッチされます。一方、対象タスクの優先レベルが、rleasを発行したタスクより低いと、rleasを発行したタスクが継続して実行されます。

パラメータtnには、対象タスクのタスク番号を指定してください。

対象タスクをCPU実行待ち状態にするとき、パラメータfactで指定された内容が起動要因として設定されます。起動要因としては、32個の要因を設定できます。ただし、同じ要因は多重に記憶されません。gfactマクロにより起動要因を取り込まれると、取り込まれた起動要因は消滅します。

指定された起動要因が、1≦起動要因≦32の範囲にないときは、起動要因なしとして処理されます。

対象タスクがCPU実行待ち状態のときにqueueが発行されると、対象タスクが終了した後で、もう一度CPU実行待ち状態となります。このような多重起動は、2回まで記憶されます。すなわち、CPU実行待ち状態の対象タスクは、あと1回起動されるよう記憶され、CPU実行待ち状態ではない対象タスクは、あと2回CPU実行待ち状態になるよう記憶されます。ただし、対象タスクの実行が、アボート処理によって打ち切られると、記憶されていた2回目の起動は行われません。

診　断

処理が正常終了すると、リターンコード0が返されます。そうでない場合は、以下のリターンコードが返されます。

1：tn＝0です。

2：tnで指定されたタスクは、DORMANT状態でした。

4：tnで指定されたタスクは、未登録です。

パラメータチェック

以下のパラメータチェックを行い、異常ならばパラメータチェックエラーとして報告します。

・tnで指定されたタスク番号が、0≦タスク番号≦最大タスク番号であること。

名　称

exit　―　タスクを終了する

C形式

exit()

機能説明

exitは、exitを発行したタスクの実行を終了させます。すなわち、タスクをCPU実行待ち状態でなく、IDLE状態にします。

タスク実行中に、chapマクロによって変更されていた優先レベルは、タスク登録時の値に戻ります。

rservマクロによって占有していた共用リソースは、解放されます。

タスクの終了監視は、解除されます。

組み込みサブルーチンEXSが登録されている場合は、EXSにリンクします。

起動要求が記憶されている場合は、タスクの終了処理を実行した後で、再びCPU実行待ち状態になります。

診　断

exitを発行すると、タスクには制御が戻りません。したがって、リターンコードもありません。

名　称

abort　―　タスクを強制終了する

C形式

int abort(&tn)

long tn;

機能説明

abortは、パラメータtnで指定された対象タスクを強制終了させ、DORMANT状態にします。対象タスクが、実行抑止状態またはイベント待ち状態ならば、それぞれの状態を解除して、DORMANT状態にします。

タスク実行中に、chapマクロによって変更されていた優先レベルは、タスク登録時の値に戻ります。

rservマクロによって占有していた共用リソースは、解放されます。

タスクに設定されている起動要因は、クリアされます。

タスクの終了監視は、解除されます。

組み込みサブルーチンABSが登録されている場合は、ABSにリンクします。

タスクに対して設定されている起動要因は、クリアされます。

対象タスクが、timerマクロによって登録したタイマイベントはキャンセルされません。また、対象タスクに対するタイマイベント起動はキャンセルされませんが、対象タスクがDORMANT状態ならばタイマイベント起動は正常に処理されません。

診　断

処理が正常終了すると、値0が返されます。そうでない場合は、以下のリターンコードが返されます。

1：tn＝0です。

2：tnで指定されたタスクは、DORMANT状態です。

4：tnで指定されたタスクは、未登録です。

パラメータチェック

以下のパラメータチェックを行い、異常ならばパラメータチェックエラーとして報告します。

・tnで指定されたタスク番号が、0≦tn≦最大タスク番号であること。

名　称

wait　－　イベントが発生するまでタスクの実行を抑止する

C形式

int wait(&ecb_g)

long ecb_g;

機能説明

waitは、waitを発行したタスクを、postマクロによるイベントが発生するまでイベント待ち状態にします。待ち合わせるイベントは、パラメータecb_gで指定します。パラメータecb_gには、GLBに割り当てられたイベント制御ブロック（ECB）へのアドレスを設定します。

postによって指定されたパラメータecb_gの値が、waitで指定されたパラメータecb_gの値と同じならば、waitを発行したタスクは、イベント待ち状態が解除され、CPU実行待ち状態になります。

もしも、待ち合わせるイベントについて、すでにpostによってイベントが発生していたならば、waitを発行したタスクはイベント待ち状態になりません。

postによってイベントを発生させるとき、postのパラメータpcodeで指定されたPOSTコードが、ECBに設定されます。POSTコードは、ECBのビット29から0（0がLSB）に設定されます。

POSTコードは、waitのリターンコードとして報告されます。例えば、POSTコードとして、イベントが発生した要因をpostを発行したタスクが設定しておくと、waitを発行したタスクは、イベント待ち状態が解除された後で、その要因を知ることができます。

ECBは、イベントごとに定義してください。

診　断

処理が正常終了すると、POSTコードが、リターンコードとして返されます。

パラメータチェック

以下のパラメータチェックを行い、異常ならばパラメータチェックエラーとして報告します。

・ecb_gで指定されたECBが、すでに他のタスクのwaitによって使用されています。

注意事項

ECBは、GLBに確保した後、初期値を0にしてから使用してください。

名　称

post　―　イベントを発生させ、タスクの実行を再開する

C形式

```
int post(&ecb_g, &pcode)
long ecb_g;
long pcode;
```

機能説明

postは、waitマクロによってイベントの発生を待っているタスクのイベント待ち状態を解除して、パラメータpcodeで指定されたPOSTコードを受け渡します。発生させるイベントは、パラメータecb_gで指定します。パラメータecb_gには、GLBに割り当てられたイベント制御ブロック（ECB）へのアドレスを設定してください。また、POSTコードとしては、ECBのビット29から0（0がLSB）に設定してください。

postによってイベント待ち状態が解除されたタスクが、postを発行したタスクより優先レベルが高いと、イベント待ち状態が解除されたタスクへ制御が移ります。

postによって指定されたパラメータecb_gの値が、waitで指定されたパラメータecb_gの値と同じならば、waitを発行したタスクは、イベント待ち状態が解除され、CPU実行待ち状態になります。

発生させるイベントについて、waitを発行して待ち合わせているタスクがない場合は、pcodeで指定されたPOSTコードを、ECBのビット29から0（0がLSB）に設定して、すでにイベントが発生したことを登録します。ここで設定されたPOSTコードは、設定waitが発行されたときに渡されます。

診　断

ecb_gによって指定されたイベントについて、待ち合わせているタスクがない場合は、処理は正常終了してリターンコード3が返されます。そのほかの条件で、処理が正常終了すると、リターンコード0が返されます。そうでない場合は、以下のリターンコードが返されます。

2：指定されたイベントを待ち合わせているタスクは、DORMANT状態です。

パラメータチェック

以下のパラメータチェックを行い、異常ならばパラメータチェックエラーとして報告します。

・pcodeで指定されたPOSTコードが、ECBのビット29から0（0がLSB）を使用していること。

名　称

susp　－　タスクの実行を抑止する

C形式

int susp(&tn)

long tn;

機能説明

suspは、パラメータtnで指定された対象タスクを実行抑止状態にします。対象タスクは、CPU実行待ち状態またはIDLE状態でなければなりません。

実行抑止状態のタスクは、rsumマクロまたはarsumマクロが発行されるか、abortマクロによって強制終了されるまで、実行抑止状態が解除されません。

同じ対象タスクに重複してsuspが発行されると、1回分だけが有効となるので、1回のrsumマクロによって実行抑止状態は解除されます。

診　断

処理が正常終了すると、リターンコード0が返されます。そうでない場合は、以下のリターンコードが返されます。

1：tn＝0です。

2：tnで指定されたタスクは、DORMANT状態です。

3：tnで指定されたタスクは、すでに実行抑止状態です。

4：tnで指定されたタスクは、未登録です。

パラメータチェック

以下のパラメータチェックを行い、異常ならばパラメータチェックエラーとして報告します。

・tnで指定されたタスク番号が、0≦tn≦最大タスク番号であること。

名　称

rsum　－　タスクの実行抑止を解除する

C形式

int rsum(&tn)

long tn;

機能説明

rsumは、パラメータtnで指定された対象タスクが、suspマクロによって実行抑止状態であれば、その実行抑止状態を解除します。

rsumを発行したタスクよりも優先レベルの高いタスクの実行抑止が解除されれば、そのタスクへ制御が移ります。

asuspマクロによって実行抑止状態であるタスクに、rsumを発行してもasuspによる実行抑止状態は解除できません。

診　断

処理が正常終了すると、リターンコード0が返されます。そうでない場合は、以下のリターンコードが返されます。

1：tn＝0です。

2：tnで指定されたタスクは、DORMANT状態です。

3：tnで指定されたタスクは、suspによる実行抑止状態ではありません。

4：tnで指定されたタスクは、未登録です。

パラメータチェック

以下のパラメータチェックを行い、異常ならばパラメータチェックエラーとして報告します。

・tnで指定されたタスク番号が、0≦tn≦最大タスク番号であること。

名　称

asusp　－　複数のタスクの実行を抑止する

C形式

int asusp()

機能説明

asuspは、asuspを発行したタスク以外のすべてのタスクの実行を抑止します。CPMSは実行抑止カウンタをもち、asuspが発行された回数を記憶します。すなわち、asuspが発行されると実行抑止カウンタが1増加し、arsumが発行されると実行抑止カウンタが1減算されます。実行抑止カウンタが0ならば、減算されません。実行抑止カウンタが0より大きいと、asuspを発行したタスク以外のすべてのタスクを実行抑止状態にします。asuspは、同時に1つのタスクだけが発行できます。

asuspを発行したタスクが、waitマクロまたはexitマクロを発行すると、実行抑止カウンタが0になります。また、asuspを発行したタスクが、アボートされたときも、実行抑止カウンタが0になります。

asuspによる実行抑止状態を、rsumマクロによって解除できません。一方、asuspによる実行抑止状態のタスクに対して、suspマクロを発行しても、実行抑止カウンタは変化しませんが、suspによって実行抑止されたことが記憶されます。このようなタスクの実行抑止状態を解除するには、arsumによって実行抑止カウンタを0クリアすることと、rsumによってsuspの実行抑止状態を解除する必要があります。

診　断

asuspが処理された後の実行抑止カウンタの値が、リターンコードとして返ります。

注意事項

asuspは、実行しているタスク数によりオーバヘッドが大きく異なります。

asuspは、CPUを占有するためのマクロですが、他のリソースの占有まで保証するものではありません。

すなわち、rservマクロによって占有するリソースなどについて、他のタスクと競合が発生すると、デッドロックとなります。したがってasusp発行後は、リソースの競合が発生する処理やマクロの発行を行わないでください。

asuspが有効である時間をできるだけ短くしないと、システムの運転に悪影響がでます。asuspの発行からarsumを発行するまでの間に、マクロを発行しないでください。

名　称

arsum　―　複数のタスクの実行抑止を解除する

C形式

int arsum()

機能説明

arsumは、asuspマクロによる実行抑止状態を解除します。CPMSは実行抑止カウンタをもち、asuspが発行された回数を記憶します。すなわち、asuspが発行されると実行抑止カウンタが1増加し、arsumが発行されると実行抑止カウンタが1減算されます。実行抑止カウンタが0ならば、減算されません。実行抑止カウンタが0より大きいと、asuspを発行したタスク以外のすべてのタスクを実行抑止状態にします。

asuspを発行したタスクが、waitマクロまたはexitマクロを発行すると、実行抑止カウンタが0になります。また、asuspを発行したタスクが、アボートされたときも、実行抑止カウンタが0になります。

asuspによる実行抑止状態を、rsumマクロによって解除できません。一方、suspマクロによって実行抑止状態であるタスクに、arsumを発行しても、suspによる実行抑止状態は解除できません。

arsumを発行したタスクよりも優先レベルの高いタスクの実行抑止が解除されれば、そのタスクへ制御が移ります。

診　断

arsumが処理された後の、実行抑止カウンタの値が、リターンコードとして返されます。

0：asuspによる実行抑止状態は解除されています。

n：さらにn回のarsumを発行しなければ、asuspによる実行抑止状態は解除できません。

名　称

chap　－　タスクの優先レベルを一時的に変更する

C形式

```
int chap(&tn, &chgp)
long tn, chgp;
```

機能説明

chapは、パラメータtnで指定された対象タスクの優先レベルを、パラメータchgpで指定された優先レベルに変更します。chapが処理された後、対象タスクの優先レベルが、chapを発行したタスクより高いレベルに変更されると、対象タスクがCPU実行待ち状態であれば、対象タスクに制御が移ります。対象タスクがchapを発行したタスクの場合、優先レベルを下げると、より優先レベルの高いタスクに制御が移ります。

chapによって一時的に変更された優先レベルは、対象タスクが終了するかアボートするまで有効です。

リソース待ち状態の対象タスクについて、chapによって優先レベルを高くしても、強制的にそのリソースが割り当てられるわけではありません。

診　断

処理が正常終了すると、リターンコード0が返されます。そうでない場合は、以下のリターンコードが返されます。

1：tn＝0です。

2：tnで指定されたタスクは、DORMANT状態です。

4：tnで指定されたタスクは、未登録です。

パラメータチェック

以下のパラメータチェックを行い、異常ならばパラメータチェックエラーとして報告します。

・tnで指定されたタスク番号が、0≦tn≦最大タスク番号であること。

・対象タスクがシステムタスクならば、chgpで指定された優先レベルが0≦chgp≦31であること。

・対象タスクがユーザタスクならば、chgpで指定された優先レベルが4≦chgp≦27であること。

名　称

sfact　－　タスクの起動要因を設定する

C形式

int sfact(&tn, &fact)

long tn, fact;

機能説明

sfactは、パラメータtnで指定された対象タスクに、パラメータfactで指定された起動要因を設定します。起動要因が1から32ではないときは、起動要因なしとみなされます。

設定された起動要因は、gfactマクロによって読み込まれ、その起動要因はクリアされます。同じタスクに対して、同じ起動要因を設定しても、多重に記憶されませんので、1回のgfactによって、その起動要因はクリアされます。

対象タスクが、未登録またはDORMANT状態の場合は、起動要因は設定されません。対象タスクがアボートされると、起動要因はすべてクリアされます。

診　断

処理が正常終了すると、リターンコード0が返されます。そうでない場合は、以下のリターンコードが返されます。

1：tn＝0です。

2：tnで指定されたタスクは、DORMANT状態です。

4：tnで指定されたタスクは、未登録です。

パラメータチェック

以下のパラメータチェックを行い、異常ならばパラメータチェックエラーとして報告します。

・tnで指定されたタスク番号が、0≦tn≦最大タスク番号であること。

名　称

gfact　―　タスクの起動要因を取り込む

C形式

```
int gfact(fact)
long *fact;
```

機能説明

gfactは、パラメータfactで指定されたアドレスに、gfactを発行した対象タスクに設定されている起動要因を取り込みます。起動要因は、数値の小さい順に1つ取り込みます。

取り込まれた起動要因は、クリアされます。取り込まれずに残っている起動要因は、再度gfactを発行することにより取り込まれます。起動要因がすべて取り込まれると、gfactはfactに0を返します。

タスクが起動されたときは、gfactによってすべての起動要因を取り込むようにしてください。

診　断

処理が正常終了すると、リターンコード0が返されます。

名　称

wrtmem　－　プロテクトメモリへの書き込み

C形式

```
int     wrtmem(vaddr, dst, size)
long   *vaddr;
long   *dst;
int     size;
```

機能説明

書き込みプロテクトされているメモリに書き込みを行います。これは、プログラミング系のタスクがプログラムやデータを書き込むためのものです。

パラメータ

vaddr：転送元先頭アドレス（4バイト境界を指定のこと）
dst　：転送先先頭アドレス（4バイト境界を指定のこと）
size　：データ数（バイト単位、4の倍数でかつ8192以下であること）

診　断

処理が正常終了するとリターンコードに0を返します。
そうでなければ、以下のリターンコードを返します。
1：パラメータエラー（vaddr，data，sizeのいずれかが条件外）

注意事項

このマクロは指定したアドレスによりプログラムを破壊します。CPMSはこのマクロでプログラムが破壊されるのを防ぐことはできません。

名　称

chkbmem　―　バスメモリのアクセスチェック

C形式

```
int  chkbmem(slot)
long slot;
```

機能説明

指定スロットのバスメモリのアクセス可否を返します。

パラメータ

slot：バスメモリのスロット番号（0～7）

診　断

バスメモリが正常（アクセス可能）ならばリターンコードに0を返します。
そうでなければ、以下のリターンコードを返します。
0x8000：パラメータエラー
0x8001：未実装
0x8004：ターゲットアボート検出あり（故障）

名　称

chktaer　―　ターゲットアボートエラーチェック

C形式

int chktaer(slot)

long slot;

機能説明

指定スロットのバスメモリのターゲットアボート発生の有無を返します。

パラメータ

slot：バスメモリのスロット番号（0～7）

診　断

リターンコード

0：ターゲットアボート発生なし

1：ターゲットアボート発生あり

名　称

mvmem　－　指定エリア間のデータ転送

C形式

```
int mvmem(wno, daddr, saddr)
long *wno;
long *daddr;
long *saddr;
```

機能説明

saddrで指定されたアドレスから、daddrで指定されたアドレスにデータを指定バイト数転送します。

パラメータ

wno　：転送ワード数
daddr：データの転送先先頭アドレス
saddr：データの転送元先頭アドレス

診　断

リターンコードは常に0です。

名　称

uspchk　－　スタック使用バイト数のチェック

C形式

```
int uspchk(usebyt, addr)
long *usebyt;
long *addr;
```

機能説明

このマクロ命令を発行したタスクが、パラメータで指定したバイト数を超えてスタックを使用しているかチェックします。このマクロをコールした際の使用しているスタックサイズとチェックします。

パラメータ

usebyt：チェックするスタックエリアのバイト数
addr　：変更するステータスレジスタのマスク

診　断

正常終了するとリターンコードに0が返り、addrに指定スタックエリアのバイト数までの空き容量が返ります。

異常終了時、リターンコードに1が返り、addrに指定スタックエリアのバイト数を超えた容量が返ります。

備　考

・プログラムのネスティングの最も深い所でこのマクロ命令を実行することが最も効果的です。
・デバック終了後は、このマクロ命令をプログラムから削除することをお勧めします。
・ユーザは、リターンコード判定により、異常処理を記述する必要があります。

名　称

timer　－　タイマイベント起動するタスクを登録する

C形式

```
int timer(&id, &tn, &fact, &t, &cyt)
long id, tn, fact, t, cyt;
```

機能説明

timerは、パラメータtnで指定された対象タスクをタイマイベント起動されるように登録します。

タイマイベントのタイプは、パラメータidで指定します。タイマイベントのタイプは、時間指定、時刻指定、時間周期指定、時刻周期指定の4つあり、それぞれの説明を次ページの表に記述します。

時刻を指定してタイマイベントを登録するときに、すでにその時刻が過ぎているときには、翌日の同じ時刻として登録されます。stimeマクロで時刻を進めたときに、時刻指定のタイマイベントがスキップされると、翌日の同じ時刻として登録されます。

タイマイベントで起動されるタスクに、パラメータfactで指定された起動要因を受け渡します。

指定された起動要因が、1≦起動要因≦32の範囲にないときは、起動要因なしとして処理されます。

タイマイベントを登録するときに、対象タスクの状態はチェックされません。タイマイベントが発生したときに、対象タスクがDORMANT状態のときは、起動は行われません。

タイマイベントを取り消すときは、ctimeマクロを使用してください。タイマイベント起動が登録されているタスクをアボートしたりタスクを削除したりしても、タイマイベントは取り消されません。

パラメータの意味は以下のとおりです。

id　：タイマイベントのタイプ（1～4のどれかを指定してください）
tn　：タイマイベント起動を登録したいタスクのタスク番号
fact：起動されるタスクに渡す起動要因
t　　：初回のタイマイベントの時刻または現在からの相対時間（どちらもミリ秒単位）
cyt ：周期的にイベントを発生させる場合の周期時間（ミリ秒単位）
idが1または2のとき0、idが3または4のとき0＜cyt≦86400000としてください。

timerマクロのパラメータid, t, cytの説明

タイマイベント	id	t	cyt	説　明
時間指定	1	現時刻から起動までの相対時間	0を指定してください	パラメータtで指定された時間経過後、パラメータtnで指定されたタスクを起動します。
時刻指定	2	午前0時を起点にした起動時刻	0を指定してください	パラメータtで指定された時刻に、パラメータtnで指定されたタスクを起動します。
時間周期指定	3	現時刻から起動までの相対時間（初回の起動までの相対時間）	初回の起動後、周期的に起動する周期時間	パラメータtで指定された時間経過後、パラメータtnで指定されたタスクを起動します。そのあと、パラメータcytで指定された周期で、パラメータtnで指定されたタスクを起動します。
時刻周期指定	4	午前0時を起点にした起動時刻（初回の起動時刻）	初回の起動後、周期的に起動する周期時間	パラメータtで指定された時刻に、パラメータtnで指定されたタスクを起動します。そのあと、パラメータcytで指定された周期で、パラメータtnで指定されたタスクを起動します。

診　断

処理が正常終了すると、リターンコード0が返されます。そうでない場合は、以下のリターンコードが返されます。

1：tn＝0です。

4：システムテーブル不足により、タイマイベントの登録ができません。

パラメータチェック

以下のパラメータチェックを行い、異常ならばパラメータチェックエラーとして報告します。

・tnで指定されたタスク番号が、0≦tn≦最大タスク番号であること。

・idで指定されたタイプが、1≦id≦4であること。

・idが1または3のとき、tで指定された時間が、0＜t≦86400000であること。

・idが2または4のとき、tで指定された時刻が、0≦t＜86400000であること。

・idが1または2のとき、cyt＝0であること。

・idが3または4のとき、cytで指定された周期が、0＜cyt≦86400000であること。

名　称

ctime　－　タスクのタイマイベント起動を取り消す

C形式

```
int ctime(&tn, &fact)
long tn, fact;
```

機能説明

ctimeは、timerマクロで登録されたタイマイベントを取り消します。

パラメータtnで指定されたタスク番号とパラメータfactで指定された起動要因が一致するタイマイベントを検索して、これらの両方が一致したタイマイベントをすべて取り消します。起動要因が1から32ではないときは、起動要因なしとみなされます。

すでに起動されたタスクの実行を、ctimeによって打ち切ることはできません。ただし、すでに起動されたタスクについて、現在以降に登録されている周期イベントがあれば、そのタイマイベントは取り消されます。

診　断

処理が正常終了すると、リターンコード0が返されます。そうでない場合は、以下のリターンコードが返されます。

1：指定されたタスク番号と起動要因に一致するタイマイベントが、登録されていません。

パラメータチェック

以下のパラメータチェックを行い、異常ならばパラメータチェックエラーとして報告します。

・tnで指定されたタスク番号が、0≦tn≦最大タスク番号であること。

名　称

delay　－　タスクの実行を指定された時間だけ抑止する

C形式

int delay(&t)

long t;

機能説明

delayは、delayを発行したタスクを、パラメータtで指定された時間だけ実行抑止状態にします。

パラメータtには実行抑止したい時間を、ミリ秒単位で指定します。実行抑止されている間は、ほかのタスクに制御が移ります。指定された時間だけ実行抑止後、ほかに動作可能なタスク（delay発行タスクより高い優先レベル、または同じ優先レベルでも先に起動されているもの）がなければdelay発行タスクに制御が戻ります。

診　断

処理が正常終了すると、リターンコード0を返します。そうでない場合は、タスクの実行抑止は行われず、以下のリターンコードが返されます。

4：システムテーブル不足により、タスクの実行を抑止できません。

パラメータチェック

以下のパラメータチェックを行い、異常ならばパラメータチェックエラーとして報告します。

・tで指定された実行抑止時間が、0＜t≦86400000（24時間）であること。

注意事項

共用リソースを占有したままでdelayを発行することは、システムの運転上好ましくありませんので避けてください。

名　称

stime　－　時刻を設定する

C形式

```
int stime(&time)
struct{
    short year;
    short month;
    short day;
    short week;
    long  msec;
}time;
```

機能説明

stimeは、パラメータtimeで指定された時刻を、CPMSが管理している時刻とTODに設定します。

timeは、以下のように指定します。

year　：西暦年を設定します。
　　1970≦year≦2069を指定してください。

month：月を設定します。

day　：日を設定します。

week　：使用しませんので0を設定してください。

msec　：午前0時を起点としたミリ秒単位の時刻を設定します。
　　0≦msec≦86399000（=23時59分59秒）を指定してください。

診　断

処理が正常終了すると、リターンコード0が返されます。そうでない場合は、以下のリターンコードが返されます。

1：timeで指定された時刻が、正しくありません。

パラメータチェック

以下のパラメータチェックを行い、異常ならばパラメータチェックエラーとして報告します。

・1970≦year≦2069、1≦month≦12、1≦day≦31、0≦msec≦86399000であること。

注意事項

timerマクロによって登録されているタイマイベントは、下表のようにイベント発生時刻が変更されます。

タイマイベントのタイプ	時刻を遅らせる場合	時刻を進める場合	備　考
時間指定と 時間周期指定	タイマイベント時刻は、影響を受けません。	タイマイベント時刻は、影響を受けません。	timer によって指定された時間経過後に、タイマイベントが発生します。
時刻指定と 時刻周期指定	時刻を遅らせることにより、タイマイベント時刻が24時間以上遠のいたものは、変更後の日付の同じ時刻に設定されます。	最初の起動予定時刻に周期時刻を加えていった時刻が、変更後の時刻以降となる時刻に起動予定時刻を移します。最初の起動予定時刻が過ぎてしまい起動タイミングが失われたものは、時刻が変更されたときに起動されます。	

wakeマクロによって登録されているタイマイベントは、下表のようにイベント発生時刻が変更されます。

タイマイベントのタイプ	時刻を遅らせる場合	時刻を進める場合	備　考
時刻指定	（Don’t care指定ありの場合）時刻を遅らせることにより、起動時刻が24時間以上遠のいたものは、変更後の日付の同じ時刻に設定されます。 （絶対時刻指定の場合）時刻の変更に影響を受けません。	時刻を進めることにより、起動時刻がスキップされたものは、翌日の同じ時刻として登録されます。	
時刻指定と 周期指定	同上	時刻を進めることにより、起動時刻がスキップされたものは、最初の起動時刻に周期時間を加えていったものが新しく設定した時刻以降となる時刻に登録されます。	

名　称

gtime　－　現在時刻の取り込み

C形式

```
int gtime(time)
struct{
    short year;
    short month;
    short day;
    short week;
    long  msec;
}*time;
```

機能説明

gtimeは、現在時刻をパラメータtimeで指定されたアドレスに取り込みます。

timeには以下の内容が返されます。

year　：西暦年が格納されます（1970～2069）。

month：月が格納されます。

day　：日が格納されます。

week　：曜日が格納されます（1～7が日～土に対応します）。

msec　：午前0時を起点としたミリ秒単位の時刻が格納されます。

診　断

処理が正常終了すると、リターンコード0が返されます。

名　称

wake　―　指定時刻のタスク起動

C形式

```
int         wake(&id, &tn, &fact, &time, &cycle)
long id;
long tn;
long fact;
TIME time;
Long cycle;
```

機能説明

パラメータに指定された起動時刻にタスクを起動します。周期起動を指定した場合は、起動時刻にタスクを起動後、指定周期時間ごとにタスク起動します。タスクの起動要因としてパラメータに指定されたfactがタスクに渡ります。

パラメータ

id　：起動モード（0：時刻起動，1：周期起動）です。
tn　：起動したいタスクのタスク番号
fact　：起動するタスクへ渡す起動要因
time　：起動する時刻を設定するTIME構造体へのポインタ
　　short sec　：午前0時を0として秒単位で与えます（0≦sec≦86399）。
　　short day　：日を与えます。
　　short month：月を与えます。
　　short year　：西暦年を与えます（1970≦year≦2069）。
　　long week　：使用しませんので0を設定してください。
cycle：周期時間を指定（1≦msec≦863400）

・idとtimeおよびcycleの関係は次のようになります。

id	time	cycle	内　容
0	起動時刻	0	timeで指定した時刻に1回だけ指定されたタスクを起動します。
1	最初の起動時刻	最初の起動時刻以降、周期的に起動する時間	timeで指定した時刻に指定されたタスクを起動します。その後、cycleで指定された周期ごとに被要求タスクを起動します。

・起動時刻はDon’t careコード（＝-1）を使用することにより次のような設定が可能です。

No.	year	month	day	sec	内　容
1	1990	1	10	36610	1990年1月10日10時10分10秒に起動（絶対時刻指定）
2	-1 Don’t care	1	10	36610	今年または翌年の1月10日10時10分10秒に起動 (*2)
3	(*1)	-1 Don’t care	10	36610	今月または翌月の10日10時10分10秒に起動 (*2)
4	(*1)	(*1)	-1 Don’t care	36610	今日または翌日の10時10分10秒に起動 (*2)

(*1)　Don’t careコードより位の大きいデータは無視されます。

(*2)　現在時刻より前の場合、翌年、翌月、または翌日に起動されます。現在時刻より後の場合は、今年、今月、または今日に起動されます。

診　断

正常終了するとリターンコードに0が返ります。

1：tn＝0です。

4：システムテーブル不足により、登録ができません。

名　称

cwake　－　wakeで登録されたタスク起動要求を取り消す

C形式

```
int         cwake(&tn, &fact)
long tn;
long fact;
```

機能説明

wakeで登録された起動要求を取り消します。

指定tnとfactが共に一致するタスクを起動要求から取り消します。

パラメータ

tn　：起動したいタスクのタスク番号

fact：起動するタスクへ渡す起動要因

診　断

正常終了するとリターンコードに0が返ります。

1：指定したものと一致するものが、システムテーブルにありません。

名　称

rserv　－　共用リソースを一括占有する

C形式

```
#include <cpms/cpms_rsrv.h>

int rserv(&n, &para1, &para2,...)
long n;
cpms_rserv_t para1, para2,...;
```

機能説明

rservは、パラメータpara1, para2,...で指定された複数の共用リソースの占有処理を行います。rservを発行した対象タスクが、すでにrservによって共用リソースを占有中ならば、rservはエラーリターンします。つまり、共有リソースを一括占有することによって、デッドロックを防止しています。

rservは、prsrvマクロとの間での、共用リソースの排他制御は行いません。

対象タスクが、まだrservによって共用リソースを占有していなければ、パラメータpara1, para2,...で指定された共用リソースが、ほかのタスクによって占有されていないかを調べます。もしも、指定された共用リソースのすべてが占有されていなければ、これらの共用リソースを対象タスクが占有します。もしも、指定された共用リソースのどれかが、ほかのタスクによって占有されていれば、rservからリターンせず、対象タスクは実行抑止状態になります。

共用リソースを占有できないため実行抑止状態になったタスクは、ほかのタスクがfreeマクロによって、対象の共用リソースがすべて解放されると、共用リソースを占有してrservからリターンします。

占有されている共用リソースは、freeを発行するか、占有中のタスクが終了またはアボートすると、解放されます。

rservで占有した共用リソースを、pfreeマクロで解放できません。

rservを発行したタスクは、共用リソースがほかのタスクに占有されていなくても、システムテーブルが不足すると、実行抑止状態となります。この場合は、ほかのタスクがfreeによって共用リソースを解放することによって、システムテーブルの空きが発生しますので、共用リソースを占有してrservからリターンします。共用リソースを管理するシステムテーブルの数は、CPMSによって定義されていますので、その数を超えないように共用リソースを設定してください。

ほかのタスクの実行を抑止するsuspやasuspマクロを発行した後で、rservを発行しないでください。実行抑止されたタスクが共用リソースを占有中ならば、デッドロックとなります。

共用リソースは、GLBのSAREA内の任意の領域として表現されます。

パラメータnとcpms_rserv_t構造体には、以下の内容を指定してください。

n　　：占有したい共用リソースの数（1～5）

```
typedef struct cpms_rserv{
        long type;
        long addr;
        long top;
        long last;
}cpms_rserv_t;
```

type ：このパラメータは、意味を持ちません。0を指定してください。

addr ：占有したい共用リソースを含むSAREAのアドレス

top　：占有したい共用リソースの先頭アドレス。SAREA先頭からの相対アドレスです。

last　：占有したい共用リソースの最終アドレス。SAREA先頭からの相対アドレスです。

診　断

すべての共用リソースが占有できると、リターンコード0が返されます。そうでない場合は、以下のリターンコードが返されます。

2：すでにrservまたはprsrvによって共用リソースを占有中です。

パラメータチェック

以下のパラメータチェックを行い、異常ならばパラメータチェックエラーとして報告します。

・addrの値が正しいこと。

・topまたはlastの値が正しいこと。

・nで指定された共用リソースの数が、1≦n≦5であること。

注意事項

prsrvにより資源を占有しているときには、rservで資源を占有できません。

名　　称

free　－　rservで占有した共用リソースの占有を解除する

C形式

```
#include <cpms/cpms_rsrv.h>

int free(&n, &para1, &para2,...)
long n;
cpms_rserv_t para1, para2,...;
```

機能説明

freeは、rservマクロによって占有されている共用リソースを解放します。freeは、パラメータpara1, para2,...で指定された複数の共用リソースについて、占有されている共用リソースがあれば、すべて解放します。

共用リソースの解放を待っていたタスクは、その共用リソースが解放されることによって、実行抑止が解除されます。

もし、指定された複数の共用リソースのうち、占有されていない共用リソースがあれば、リターンコード1を返しますが、この場合でも、占有されている共用リソースは解放されます。

prsrvマクロで占有した共用リソースは、freeで解放できません。

共用リソースは、GLBのSAREA内の任意の領域として表現されます。

パラメータnとcpms_rserv_t構造体には、以下の内容を指定してください。

n　　：解放したい共用リソースの数（1～5）

```
typedef struct cpms_rserv{
        long type;
        long addr;
        long top;
        long last;
}cpms_rserv_t;
```

type ：このパラメータは、意味を持ちません。0を指定してください。

addr ：解放したい共用リソースを含むSAREAのアドレス

top　：解放したい共用リソースの先頭アドレス。SAREA先頭からの相対アドレスです。

last　：解放したい共用リソースの最終アドレス。SAREA先頭からの相対アドレスです。

診　断

共用リソースが解放されると、リターンコード0または1が返されます。そうでない場合は、以下のリターンコードが返されます。

2：指定された共用リソースは、すべて占有中ではありませんでした。

パラメータチェック

以下のパラメータチェックを行い、異常ならばパラメータチェックエラーとして報告します。

・addrの値が正しいこと。

・topまたはlastの値が正しいこと。

・nで指定された共用リソースの数が、1≦n≦5であること。

名　称

prsrv　－　共用リソースを占有する

C形式

```
#include <cpms/cpms_rsrv.h>

int prsrv(&n, &para1, &para2,...)
long n;
cpms_rserv_t para1, pata2,...;
```

機能説明

prsrvは、パラメータpara1, para2,...で指定された複数の共用リソースの占有処理を行います。prsrvを発行した対象タスクが、すでにprsrvによって共用リソースを占有中でも、別の共用リソースに対してprsrvを発行できます。

prsrvは、rservマクロとの間での共用リソースの排他制御は行いません。

prsrvは、パラメータpara1, para2,...で指定された共用リソースが、他のタスクによって占有されていないかを調べます。もし、指定された共用リソースのすべてが占有されていなければ、これらの共用リソースを対象タスクが占有します。もし、指定された共用リソースのどれかが、すでに他のタスクによって占有されていれば、prsrvからリターンせず、対象タスクは実行抑止状態になります。

指定された共用リソースが、自タスクが発行したprsrvによって占有されている場合は、その共用リソースが占有できたものとして処理します。このように、同じ共用リソースを、同じタスクがprsrvによって多重に占有した場合は、prsrvを発行した回数分、その共用リソースに対してpfreeを発行しなければ解放されません。

共用リソースを占有できないため実行抑止状態になったタスクは、ほかのタスクがpfreeマクロによって、対象の共用リソースがすべて解放されると、共用リソースを占有してprsrvからリターンします。

占有されている共用リソースは、pfreeを発行するか、占有中のタスクが終了またはアボートすると解放されます。

prsrvで占有した共用リソースを、freeマクロで解放できません。

prsrvを発行したタスクは、共用リソースがほかのタスクに占有されていなくても、システムテーブルが不足すると、実行抑止状態となります。この場合は、ほかのタスクがpfreeによって共用リソースを解放することによって、システムテーブルの空きが発生しますので、共用リソースを占有してprsrvからリターンします。共用リソースを管理するシステムテーブルの数は、CPMSによって定義されていますので、その数を超えないように共用リソースを設定してください。

共用リソースは、GLBのSAREA内の任意の領域として表現されます。

パラメータnとcpms_rserv_t構造体には、以下の内容を指定してください。

n　　：占有したい共用リソースの数（1～5）

```
typedef struct cpms_rserv{
        long type;
        long addr;
        long top;
        long last;
}cpms_rserv_t;
```

type ：このパラメータは、意味を持ちません。0を指定してください。

addr：占有したい共用リソースを含むSAREAのアドレス

top　：占有したい共用リソースの先頭アドレス。SAREA先頭からの相対アドレスです。

last　：占有したい共用リソースの最終アドレス。SAREA先頭からの相対アドレスです。

診　断

すべての共用リソースが占有できると、リターンコード0が返されます。そうでない場合は、以下のリターンコードが返されます。

2：同一タスクで占有できる共用リソースの数を超えています。

パラメータチェック

以下のパラメータチェックを行い、異常ならばパラメータチェックエラーとして報告します。

・addrの値が正しいこと。

・topまたはlastの値が正しいこと。

・nで指定された共用リソースの数が、1≦n≦5であること。

名　称

pfree　－　prsrvで占有した共用リソースの占有を解除する

C形式

```
#include <cpms/cpms_rsrv.h>

int pfree(&n, &para1, &para2,...)
long n;
cpms_rserv_t para1, para2,...;
```

機能説明

pfreeは、prsrvマクロによって占有されている共用リソースを解放します。pfreeは、パラメータpara1, para2,...で指定された複数の共用リソースについて、占有されている共用リソースがあれば、すべて解放します。

共用リソースの解放を待っていたタスクは、その共用リソースが解放されることによって、実行抑止が解除されます。

もし、指定された複数の共用リソースのうち、占有されていない共用リソースがあれば、リターンコード1を返しますが、この場合でも、占有されている共用リソースは解放されます。

rservマクロで占有した共用リソースを、pfreeで解放できません。

共用リソースは、GLBのSAREA内の任意の領域として表現されます。

パラメータnとcpms_rserv_t構造体には、以下の内容を指定してください。

n　：解放したい共用リソースの数（1～5）

```
typedef struct cpms_rserv{
        long type;
        long addr;
        long top;
        long last;
}cpms_rserv_t;
```

type ：このパラメータは、意味を持ちません。0を指定してください。

addr ：解放したい共用リソースを含むSAREAのアドレス

top　：解放したい共用リソースの先頭アドレス。SAREA先頭からの相対アドレスです。

last　：解放したい共用リソースの最終アドレス。SAREA先頭からの相対アドレスです。

診　断

共用リソースが解放されると、リターンコード0または1が返されます。そうでない場合は、以下のリターンコードが返されます。

2：指定された共用リソースは、すべて占有中ではありませんでした。

パラメータチェック

以下のパラメータチェックを行い、異常ならばパラメータチェックエラーとして報告します。

・addrの値が正しいこと。

・topまたはlastの値が正しいこと。

・nで指定された共用リソースの数が、1≦n≦5であること。

名　称

wdtset　－　WDTスタート／ストップの制御

C形式

```
int     wdtset(msec)
long   *msec;
```

機能説明

WDT（ウォッチドッグタイマ）のスタート／ストップの制御を行います。
WDTタイムアウト時には、組み込みサブルーチンWDTESにリンクします。
WDTESにてWDTタイムアウト処理を行ってください。

パラメータ

msec：WDT時間（0～65535）（単位：1msec）
　　msecに1～65535を設定時、WDTを開始します。
　　msecに0を設定時、WDTを停止します。

診　断

0：正常終了
1：パラメータ異常

名　称

getsysinfo　－　システムの状態を取り出す

C形式

```
int getsysinfo(type, addr)
int    type;
char  *addr;
```

機能説明

getsysinfoは、typeで指定されたシステム情報を、addrで指定されたアドレスへ返します。
typeには、以下のどれか1つを指定してください。

- SYS_IDLE

  IDLE時間の累積を返します。

```
struct sys_idle {
        unsigned int idle_sec;   /* 秒単位 */
        int idle_nsec; /* ナノ秒単位*/
};
```

  IDLE時間はCPMS起動時を0として累積するので、前回SYS_IDLEを発行した時点のIDLE時間と現在のIDLE時間の差でその間のIDLE時間を求めてください。
  測定時間は、24時間以内としてください。

- SYS_CPMS

  CPMSのバージョン番号を返します。

```
int  cpms_ver;
```

- SYS_PROC

  プロセッサ番号を返します。

```
int  proc_no;
```

診　断

処理が正常終了すると、リターンコードに情報のサイズ（バイト単位）を返します。そうでなければ、以下のリターンコードを返します。

0 ：typeで指定されたシステム情報は、処理対象ではありません。
-1：システム情報が正しく取り出せませんでした。

名　称

gettaskinfo　－　タスクの状態を取り出す

C形式

```
int gettaskinfo(type, tn, addr)
int  type, tn;
char  *addr;
```

機能説明

gettaskinfoは、tnで指定されたタスクについて、typeで指定された情報をaddrで指定されたアドレスへ返します。gettaskinfoを発行するタスクの情報を取り出すときは、tnに0を指定してください。

typeには、以下のどれか1つを指定してください。

- TASK_TN

  gettaskinfoを発行したタスクのタスク番号を返します。tnには、0を指定してください。

  int task_tn;

- TASK_PRI

  tnで指定されたタスクの優先レベルを返します。

  int task_pri;

- TASK_STAT

  tnで指定されたタスクの現在のタスク状態を返します。

  int task_stat;

  0：未登録、1：DORMANT、2：IDLE、3：READY、4：SUSPENDED、5：WAIT

診　断

処理が正常終了すると、リターンコードに情報のサイズ（バイト単位）を返します。そうでなければ、以下のリターンコードを返します。

0 ：typeで指定されたタスク情報は、処理対象ではありません。または、tnで指定されたタスクは、未登録です。

-1：タスク情報が正しく取り出せませんでした。

名　称

gtkmem　－　CPMS管理テーブルを読み出す

C形式

```
int gtkmem(tblno, caseno, offset, size, buf)
int  tblno, caseno, offset, size;
char  *buf;
```

機能説明

gtkmemは、CPMSが管理するテーブル内のデータを読み出します。

パラメータの意味は、以下のとおりです。

tblno　：対象テーブルを指定する番号

テーブル：OSCB＝1　　RSVB＝6
SYSCB＝2　　UCB＝7
TCB＝3　　TRB＝8
TMCB＝4
RSCB＝5

caseno：対象テーブル内の相対ケース番号
対象テーブルがOSCB, SYSCBの場合は、0を指定してください。

offset　：読み出すデータのケース内の相対アドレス

size　　：読み出すデータサイズ（バイト数）

buf　　：読み出すメモリのアドレス

診　断

処理が正常終了すると、リターンコードに0を返します。そうでなければ、以下のリターンコードを返します。

1：tblnoで指定されたテーブルは、処理対象ではありません。

2：テーブルのデータが、正しく取り出せませんでした。

名　称

usrdhp　－　DHP記録書き込み

C形式

```
#include <cpms_dhp.h>

int  usrdhp(code, data, ndata)
unsigned long code;
long *data;
long ndata;
```

機能説明

ユーザ定義イベントをカーネル動作トレース（DHP）へ記録します。

パラメータ

code ：トレースコード
　　DHP_USR0～DHP_USR7のどれかを指定してください。
data ：トレースデータを格納する配列を指定するポインタ
ndata：配列の要素（0～5、1ケースは4バイト）

診　断

0：正常終了
1：パラメータ異常

名　称

usrel　―　ユーザエラーログの書き込み

C形式

```
#include <cpms_elog.h>

int  usrel(type, class, retcode, errtype, erb)
long  type;
long  class;
long  retcode;
long  errtype;
long  *erb;
```

機能説明

組み込みサブルーチンEASにリンクした後、OS内のエラーログ用のバッファエリアに引数で指定されたエラー情報を書き込みます。

パラメータ

type：重要度タイプを指定します。タイプは以下のいずれかを指定してください。

LOG_TYPE_NONFATAL
システムダウンしないタイプのエラーですが、機能的に一部縮退する場合に指定します。
プログラムエラー、入出力エラーなどが該当します。

LOG_TYPE_WARNING
警告エラーです。修復可能なエラーの場合に指定します。一時的なメモリ不足などの資源不足エラーなどが該当します。

LOG_TYPE_NOTE
ユーザに情報を提供するためのメッセージです。

class：エラーメッセージ用のクラス（サブシステムの識別）を指定します。クラスの指定には以下のいずれかを指定してください。これらの意味付けはユーザが行ってください。

LOG_CLASS_MSOFT1～LOG_CLASS_MSOFT16　ミドルウェア用
LOG_CLASS_USER1～LOG_CLASS_USER16　　アプリケーション用

retcode：エラー検出直前で関数呼び出しをしたリターン値を設定します。該当する関数がない場合は0を設定してください。

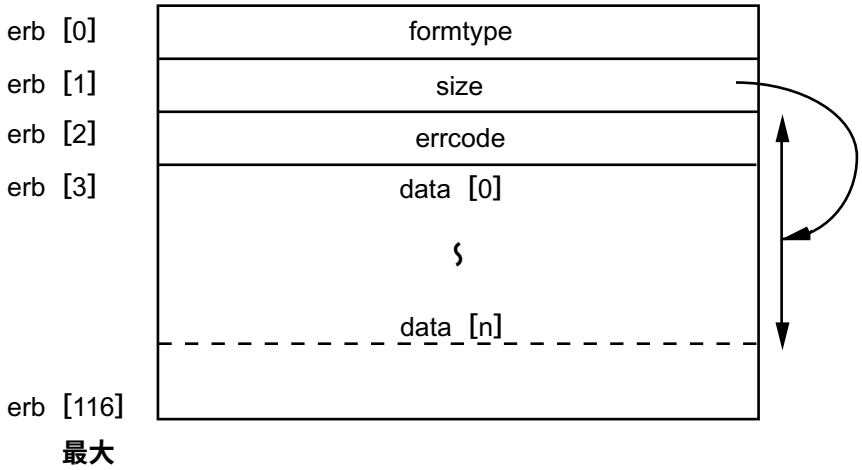
errtype：故障推定原因の種別を指定します。以下のいずれかを指定してください。

LOG_ERRTYPE_HARD　ハードウェア

LOG_ERRTYPE_SOFT　ソフトウェア

erb　　：エラーブロックへのポインタを指定します。エラーブロックのフォーマットを以下に示します。

● エラーブロックのフォーマット



formtype：エラーメッセージのフォーマットタイプを指定します。

フォーマットタイプの値は、以下とします。

LOG_FORM_MSOFT1～LOG_FORM_MSOFT16　ミドルウェア用

LOG_FORM_USER1～LOG_FORM_USER16　アプリケーション用

LOG_FORM_PIOERR　PI/Oエラー

LOG_FORM_MODULERR　モジュールエラー

ミドルウェア用、アプリケーション用はそれぞれのユーザが内容を決めてください。

PI/Oエラー、モジュールエラーは、OSが規定するフォーマットです。

size　　：errcode以降の有効データサイズをバイト数で指定します（4～460）。

errcode　：エラーコードを指定します。

0x08000000～0x08FFFFFF：ミドルウェア用

0x09000000～0x09FFFFFF：アプリケーション用

これらの意味付けはユーザが行ってください。ただし、上位16ビットをメジャー（エラー種別）、下位16ビットをマイナー（詳細要因）としてください。

その他、OSで規定するPI/Oエラー、モジュールエラーのエラーコードを使用できます。

data　　：エラー詳細データ。

内容は、formtypeで指定するフォーマットに合わせてください。

診　断

0：正常終了

1：異常終了

名　称

save_env　―　タスクの実行環境を保存する

C形式

```
#include <cpms_table.h>
int save_env(env)
struct task_env *env;
```

機能説明

save_envは、パラメータenvで指定されたアドレスに、save_envが発行された時点のタスクの実行環境データを保存します。保存されたタスクの実行環境データは、resume_envマクロによって使用されます。

タスクの実行環境データを保存するための構造体task_envのアドレスをパラメータenvに設定してください。

構造体task_envの構成は次のとおりで、424バイトのエリアが必要です。

```
struct task_env {
    struct basic_regs       env_basic_regs;
    struct float_regs       env_float_regs;
};
```

タスクの実行環境データを保存するエリアは、GLBに確保してください。

save_envを組み込みサブルーチン内で発行しても、何も処理しません。

診　断

save_envで、タスクの実行環境データが保存されると、リターンコード0を返します。

resume_envが発行されたことによって、save_envからリターンする場合は、resume_envのパラメータvalで指定された値が、リターンコードとして返されます。valに0が指定された場合は、リターンコード1が返されます。

注意事項

resume_envでsave_envを発行した時点に制御を戻すとき、ユーザスタックまたはタスクの制御に関連するBSSやGLBデータの内容がsave_envを発行した時点の内容と異なるときは、resume_envしても前回と同じ処理が行われない場合があります。

名　称

resume_env　－　タスクの実行環境を回復する

C形式

```
#include <cpms_table.h>
void resume_env(env, val)
struct task_env *env;
int  val;
```

機能説明

resume_envは、パラメータenvで指定されたタスクの実行環境を回復します。回復されるタスクの実行環境はレジスタだけですので、ユーザスタックとBSSの内容は、回復されません。

resume_envは、組み込みサブルーチンCPESから発行されなければなりません。組み込みサブルーチンCPESが実行され、その出力情報を判定した結果、タスクのアボートとCPUダウンが行われない場合に、resume_envは有効になります。resume_envが発行されても、すぐにタスクの実行環境が回復されるのではなく、組み込みサブルーチンCPESのすべてのエントリが実行された後で、タスクの実行環境が回復され、envに対応するsave_envのリターンアドレスに制御が移ります。

タスクの実行環境が正常に回復されると、指定されたタスクの実行環境に対応するsave_envマクロから、リターンします。このとき、save_envはリターンコードとしてパラメータvalを返します。valに0が指定されている場合は、save_envはリターンコードとして1を返します。

組み込みサブルーチンCPESのいくつかのエントリで、resume_envが複数回発行された場合は、最後に発行されたresume_envのパラメータが有効になります。

診　断

resume_envは、リターンコードを返しません。

注意事項

resume_envは、組み込みサブルーチンCPESから発行されなければなりません。

組み込みサブルーチンCPES以外から発行された場合は、resume_envは何も処理しません。

resume_envでsave_envを発行した時点に制御を戻すとき、ユーザスタックまたはタスクの制御に関連するBSSやGLBデータの内容がsave_envを発行した時点の内容と異なるときは、resume_envしても前回と同じ処理が行われない場合があります。

パラメータenvが正しくない場合は、CPUダウンとなる場合があります。

名　称

gettimebase　－　タイムベースの読み出し

C形式

void gettimebase(timebase)

unsigned long timebase[2];

機能説明

64ビットのタイムベースを読み出し返します。タイムベースは、4バスクロックごとにインクリメントします。CMUのバスクロックは39.996MHzなので、タイムベースは9.999MHzでインクリメントします。タイムベースを9999000で割ると1970年1月1日午前0時0分0秒からの秒数が求められます。

パラメータは以下を出力します。

timebase［0］：タイムベースレジスタの上位32ビット（TBU）

timebase［1］：タイムベースレジスタの下位32ビット（TBL）

注意事項

タイムベースは機種に依存するものです。将来は機種や動作周波数の違いにより、扱いが異なる可能性があります。

CMUの場合は、39996000（0x02624a60）となっています。

名　称

TimebaseToSecs　－　タイムベース値から秒、ナノ秒への変換

C形式

```
void      TimebaseToSecs(timebase, tval)
unsigned long timebase[2];
struct tval{
        unsigned int tv_sec;
        int          tv_nsec;
} tval;
```

機能説明

64ビットのタイムベース値を、1970年からの相対秒と、秒以下のナノ秒に変換します。

名　称

atmswap, 他　－　アトミックオペレーションライブラリ

C形式

```
long atmswap(addr, data)
long *addr, data;

long atmand(addr, data)
long *addr, data;

long atmor(addr, data)
long *addr, data;

long atmxor(addr, data)
long *addr, data;

long atmadd(addr, data)
long *addr, data;

long atmtas(addr, data)
long *addr, data;

long atmcas(addr, data1, data2)
long *addr, data1, data2;
```

機能説明

このライブラリは、メモリを読み出し・変更・書き込みする間に、他のタスクや割り込み処理にメモリを書き換えられなくすることで、排他的に読み出し・変更・書き込みすることを保証するものです。これにより、排他制御ができます。

いずれも扱うデータは、32ビット整数（long int）のみです。

リターン値olddataは、オペレーション前のメモリの値（addr）です。

“→addr”は、addrの示すメモリに格納することです。

```
olddata=atmswap(addr, data)        :data→addr
olddata=atmand(addr, data)         :(addr)AND data→addr
olddata=atmor(addr, data)          :(addr)OR data→addr
olddata=atmxor(addr, data)         :(addr)XOR data→addr
olddata=atmadd(addr, data)         :(addr) + data→addr
```

olddata=atmtas(addr, data)　　:Test And Swap if (addr)=0 then data→addr

olddata=atmcas(addr, data1, data2)　　:Compare And Swap if (addr)=data1 then data2→addr

注意事項

この排他性は自プロセッサ内の処理間で有効であり、他プロセッサやI/OのDMAとの間の排他制御には使用できません。

第３編　ライブラリ

第１章　総　　説

１．１　ライブラリの指定条件

ライブラリ内のサブルーチンを使用してプログラムを作成する場合、svloadコマンドに-lオプションを指定してライブラリをリンクします。ライブラリをリンクするときには、以下の点に注意してください。

- libcrs.a内のサブルーチンを使用している場合
  svloadコマンドに“ -lcrs ”を指定してください。

１．２　ライブラリの指定順序

svloadにてライブラリを指定する場合、以下の点に注意してください。

- 指定した複数のライブラリの中に同一名称がある場合、結合したいオプションファイルのあるライブラリを前に指定してください。

１．３　ライブラリ内で使用している名称

以下にライブラリ内で定義されている名称を示します。名称が重複しないようにプログラミングしてください。もし、重複した名称を使用する場合、ライブラリファイルの指定順序を結合したいオブジェクトファイルの後にすれば、ライブラリファイルからは結合されません。

- libcrs.a

fpcheck	fpchecko	fpsetmask	fpgetmask
fpsetround	fpgetround	fpsetsticky	fpgetsticky

IEEE浮動小数点処理環境制御サブルーチン

名　前

fpgetround, fpsetround, fpgetmask, fpsetmask, fpgetsticky, fpsetsticky　－　IEEE浮動小数点処理環境の制御

形　式

```
#include <ieeefp.h>

typedef enum {
        FP_RN =0,    /*   round nearest   */
        FP_RZ =1     /*   round zero (truncate)  */
    } fp_rnd;

#define fp_except  int
#define FP_X_INV        0x10    /* invalid operation exception  */
#define FP_X_OFL        0x04    /* overflow exception           */
#define FP_X_UFL        0x02    /* underflow exception          */
#define FP_X_DZ         0x08    /* devided by zeros exception   */
#define FP_X_IMP        0x01    /* imprecise(loss of precision) */

fp_rnd      fpgetround(void);
fp_rnd      fpsetround(fp_rnd rnd_dir);
fp_except   fpgetmask(void);
fp_except   fpsetmask(fp_except mask);
fp_except   fpgetsticky(void);
fp_except   fpsetsticky(fp_except sticky);
```

機能説明

浮動小数点の丸め、浮動小数点例外の発生を制御します。

(１)　丸め

丸めには2つのモードがありfpgetround(), fpswtround()で制御します。

FP_RN：近傍への丸め（round to nearest）

FP_RZ：0方向への丸め（round to zero）

丸めの初期値はFP_RNです。

(2) 浮動小数点例外

CMUで発生する浮動小数点例外は以下のとおりです。

・FPUエラー（E）　：FPSCR.DN=0、かつ非正規化数の入力時（*）
・無効演算（V）　：NaN入力のような無効な演算の場合
・0による除算（Z）：除数0による除算
・オーバフロー（O）：演算結果がオーバフローする場合
・アンダフロー（U）：演算結果がアンダフローする場合
・不正確例外（I）　：オーバフロー、アンダフロー、丸めが発生する場合

（*）CMUではFPSCR.DN=1に設定されているため、非正規化数は0として扱われFPUエラーは発生しません。

浮動小数点例外は、浮動小数点制御レジスタ（FPSCR）の例外に該当するイネーブルビットに1が設定されているときに発生します。

浮動小数点例外が発生すると、浮動小数点制御レジスタ（FPSCR）のFPU例外要因フィールドの該当するビットは1に設定されFPU例外フラグフィールドに該当するビットに1が累積されます。FPU例外が発生しない場合、FPU例外要因フィールドの該当するビットは0クリアされ、FPU例外フラグフィールドに該当するビットは変更されません。

浮動小数点例外のイネーブルビットの初期値は以下のとおりです。

・無効演算（V）　：有効
・0による除算（Z）：有効
・オーバフロー（O）：有効
・アンダフロー（U）：無効
・不正確例外（I）　：無効

浮動小数点例外の制御はfpgetmask(), fpsetmask(), fpgetsticky(), fpsetsticky()で行います。

・fpgetround()は現在の丸めモードを返します。
　FP_RN：近傍への丸め（round to nearest）
　FP_RZ：0方向への丸め（round to zero）
・fpsetround()は丸めモードを設定し、以前の丸めモードを返します。
・fpgetmask()は現在のFPSCRの例外イネーブルビットの値を返します。

例外マスクとFPSCRの例外イネーブルビットの対応を以下に示します。

例外マスク	FPSCRのイネーブルビット
FP_X_INV	無効演算（V）
FP_X_DZ	0による除算（Z）
FP_X_OFL	オーバフロー（O）
FP_X_UFL	アンダフロー（U）
FP_X_IMP	不正確例外（I）

・fpsetmask()は例外マスクの値に従ってFPSCRの例外イネーブルビットを設定し、以前の設定値を返します。
例外マスクとFPSCRの例外イネーブルビットの対応はfpgetmask()と同様です。

・fpgetsticky()はFPU例外フラグフィールドの値を返します。
stickyフラグとFPSCRのFPU例外フラグフィールドの対応を以下に示します。

stickyフラグ	FPSCRのフラグフィールド
FP_X_INV	無効演算（V）
FP_X_DZ	0による除算（Z）
FP_X_OFL	オーバフロー（O）
FP_X_UFL	アンダフロー（U）
FP_X_IMP	不正確例外（I）

・fpsetsticky()はstickyフラグの値に従ってFPU例外フラグフィールドの値を設定し、以前の設定値を返します。
stickyフラグとFPSCRのFPU例外フラグフィールドの対応はfpgetsticky()と同様です。

注意事項

fpsetsticky()はすべてのstickyフラグに対応するFPU例外フラグフィールドの値を変更します。
fpsetmask()はすべての例外マスクの値に対応する例外イネーブルビットを変更します。
fpgetround(), fpsetround()での丸めの制御で、以下のモードは使用できません。

・FP_RP ：負数は切り捨て正数は切り上げ（round to plus）
・FP_RM：正数は切り捨て負数は切り上げ（round to minus）

名　前

fpcheck, fpchecko　－　浮動小数点例外検出

形　式

```
#include <ieeefp.h>

typedef enum {
                FP_RN =0,    /*   round nearest   */
                FP_RZ =1     /*   round zero (truncate)  */
        } fp_rnd;

#define fp_except  int
#define FP_X_INV        0x10    /* invalid operation exception  */
#define FP_X_OFL        0x04    /* overflow exception           */
#define FP_X_UFL        0x02    /* underflow exception          */
#define FP_X_DZ         0x08    /* devided by zeros exception   */
#define FP_X_IMP        0x01    /* imprecise(loss of precision) */

void        fpcheck(fp_except flg);
void        fpchecko(void);
```

機能説明

発生を抑止している浮動小数点例外の発生状況を検出します。

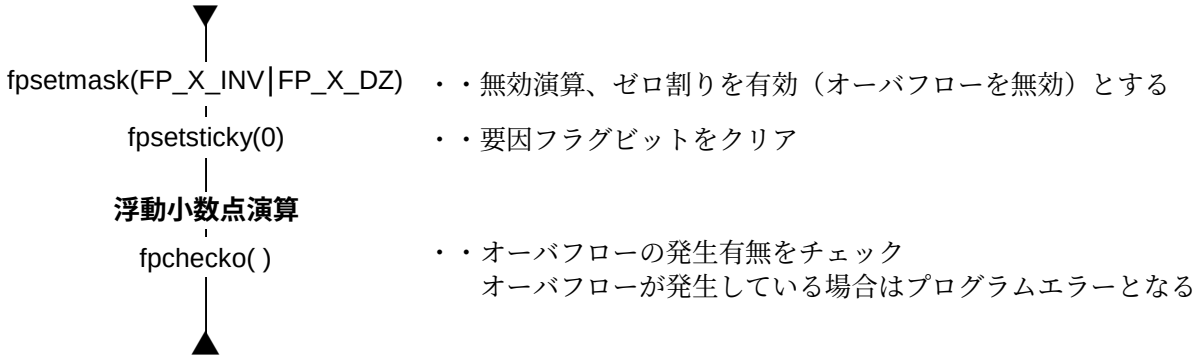
浮動小数点例外の発生を検出するとプログラムエラーとなり、タスクはアボートされます。

fpchecko()はオーバフローの発生有無を検出します。

fpcheck()はパラメータで指定した例外の発生有無を検出します。同時に複数の例外を検出する場合は例外要因の論理和を指定してください。

使用例

以下にオーバフローの発生を抑止し、演算後にオーバフローの発生を検出する場合の例を示します。



付　　録

付録A　マクロパラメータ一覧

マクロ名称	パラメータ1	パラメータ2	パラメータ3	パラメータ4	パラメータ5	パラメータ6
rleas	tn					
queue	tn	fact				
exit						
abort	tn					
wait	ecb					
post	ecb	pcode				
susp	tn					
rsum	tn					
asusp						
arsum						
chap	tn	chgp				
sfact	tn	fact				
gfact	fact					
wrtmem	vaddr	data	size			
chkbmem	slot					
chktaer	slot					
mvmem	wno	daddr	saddr			
uspchk	usebyt	addr				
timer	id	tn	fact	t	cyt	
ctime	tn	fact				
delay	t					
stime	time					
gtime	time					
wake	id	tn	fact	time	cycle	
cwake	tn	fact				
rserv	n	para1	para2	para3	para4	para5
prsrv	n	para1	para2	para3	para4	para5
free	n	para1	para2	para3	para4	para5
pfree	n	para1	para2	para3	para4	para5
wdtset	msec					
getsysinfo	type	addr				
gettaskinfo	type	tn	addr			
gtkmem	tblno	caseno	offset	size	buf	
usrdhp	code	data	ndata			
usrel	type	class	retcode	errtype	erb	
save_env	env					
resume_env	env	val				
gettimebase	timebase					
TimebaseToSecs	timebase	tval				
atmswap	addr	data				
atmand	addr	data				
atmor	addr	data				
atmxor	addr	data				
atmadd	addr	data				
atmtas	addr	data				
atmcas	addr	data1	data2			

付録B　CPMSのエラー処理一覧

No.	エラーコード	エラーメッセージ	内容	障害分類	障害部位	打ち切り	復旧処置
1	03620000	Program error (Invalid Data Access)	データアクセスエラー	ソフトウェア	TASK	TASK ABORT	プログラム修正
2	03660000	Program error (Data Access Protection)	データアクセスプロテクトエラー	ソフトウェア	TASK	TASK ABORT	プログラム修正
3	03600000	Program error (Data Page Fault)	データアクセスページフォールト	ソフトウェア	TASK	TASK ABORT	プログラム修正
4	03420000	Program error (Invalid Inst. Access)	命令アクセスエラー	ソフトウェア	TASK	TASK ABORT	プログラム修正
5	03460000	Program error (Inst. Access Protection)	命令アクセスプロテクトエラー	ソフトウェア	TASK	TASK ABORT	プログラム修正
6	03400000	Program error (Instruction Page Fault)	命令アクセスページフォールト	ソフトウェア	TASK	TASK ABORT	プログラム修正
7	03030000	Program error (Inst. Alignment Error)	命令アラインメントエラー	ソフトウェア	TASK	TASK ABORT	プログラム修正
8	03080000	Program error (Privileged Instruction)	特権命令エラー	ソフトウェア	TASK	TASK ABORT	プログラム修正
9	03040000	Program error (Illegal Instruction)	不当命令エラー	ソフトウェア	TASK	TASK ABORT	プログラム修正
10	03390000	Program error (FP Program Error)	浮動小数点演算エラー	ソフトウェア	TASK	TASK ABORT	プログラム修正
11	03470000	Program error (Data Alignment Error)	データアラインメントエラー	ソフトウェア	TASK	TASK ABORT	プログラム修正
12	05130000	Invalid macro	未定義マクロ発行	ソフトウェア	TASK	TASK ABORT	プログラム修正
13	05110000	Macro parameter error	マクロパラメータ異常	ソフトウェア	TASK	TASK ABORT	プログラム修正
14	05C70000	WDT timeout error	ウォッチドッグタイマタイムアウト	ソフトウェア	TASK	－	修正
15	03B70000	Module error (Bus Target Abort)	バスターゲットアボート	ハードウェア	I/O	－	ハードウェア交換またはプログラム修正
16	05000000	Module error (Invalid Interrupt)	無効割り込み	ハードウェア	CMU	－	ハードウェア交換
17	05000001	Module error (Undefined Invalid Interrupt)	未定義無効割り込み	ハードウェア	CMU	－	ハードウェア交換
18	05000002	Module error (INTEVT Invalid Interrupt)	INTEVT無効割り込み	ハードウェア	CMU	－	バッテリ交換
19	0500F001	Module error (HERST Invalid Interrupt)	重障害無効割り込み	ハードウェア	CMU	－	バッテリ交換
20	0500F002	Module error (HERST2 Invalid Interrupt)	重障害無効割り込み2	ハードウェア	CMU	－	ハードウェア交換
21	0500F003	Module error (BUERRSTAT Invalid Interrupt)	バスエラー重障害割り込みｽﾃｰﾀｽ無効	ハードウェア	CMU	－	ハードウェア交換
22	0500F006	Module error (MHPMCLG Invalid Interrupt)	メモリ重障害割り込みステータス無効	ハードウェア	CMU	－	ハードウェア交換
23	0500F007	Module error (ECC 2bit Master Invalid Interrupt)	メモリECC2ビットエラー重障害ステータス無効	ハードウェア	CMU	－	ハードウェア交換
24	0500F008	Module error (RERRMST Invalid Interrupt)	RERR割り込みステータス無効	ハードウェア	CMU	－	ハードウェア交換
25	0500C001	Module error (NINTR Invalid Interrupt)	NINTステータス無効	ハードウェア	CMU	－	ハードウェア交換
26	0500B001	Module error (PUINTR Invalid Interrupt)	PUINTステータス無効	ハードウェア	CMU	－	ハードウェア交換
27	05008001	Module error (PUINTC Invalid Interrupt)	PUINTCステータス無効	ハードウェア	CMU, LPU	－	ハードウェア交換
28	05005001	Module error (RINTR Invalid Interrupt)	RINTステータス無効	ハードウェア	CMU	－	ハードウェア交換
29	05003001	Module error (LV3 INTST Invalid Interrupt)	レベル3割り込みステータス無効	ハードウェア	CMU	－	ハードウェア交換
30	05003002	Module error (RQI6 INF Invalid Interrupt)	RQI6ステータス無効	ハードウェア	CMU	－	ハードウェア交換
31	05001001	Module error (RQI3 INT Invalid Interrupt)	RQI3ステータス無効	ハードウェア	CMU	－	ハードウェア交換
32	05001002	Module error (RQI3 Link Invalid Interrupt)	RQI3リンクステータス無効	ハードウェア	CMU	－	ハードウェア交換
33	05001003	Module error (RQI3 Module Invalid Interrupt)	RQI3モジュールステータス無効	ハードウェア	CMU	－	ハードウェア交換
34	0D010000	Module error (Memory Alarm)	メモリ1ビットエラー（ソリッド）	ハードウェア	CMU	－	ハードウェア交換
35	0D320000	Module error (Memory Error)	メモリエラー	ハードウェア	CMU, I/O	－	ハードウェア交換
36	0D330000	Module error (Hardware WDT timeout)	ハードウェアWDTタイムアウト	ハードウェア	CMU, I/O	－	交換
37	0D340000	Module error (Software WDT Timeout)	ソフトウェアWDTタイムアウト	ハードウェア	CMU, I/O	－	ハードウェア交換またはプログラム修正
38	0D350000	Module error (RAM Sum Check Error)	RAMチェックサムエラー	ハードウェア	CMU, I/O	－	ハードウェア交換またはプログラム修正
39	0D360000	Module error (ROM Sum Check Error)	ROMチェックサムエラー	ハードウェア	CMU, I/O	－	ハードウェア交換
40	0D370000	Module error (Clock Stop Error)	クロックストップエラー	ハードウェア	CMU, I/O	－	ハードウェア交換
41	0D380000	Module error (OS Clear Error)	OSクリアエラー	ハードウェア	CMU, I/O	－	プログラムロード
42	0D800000	Module error (TOD Error)	バックアップ時計エラー	ハードウェア	CMU, LPU	－	ハードウェア交換
43	05A00000	Kernel warning	カーネルワーニング	ソフトウェア	－	－	－
44	05A00001	Clock synchronization	カーネルワーニング	ソフトウェア	－	－	－
45	05D00000	Kernel information	カーネルインフォメーション	ソフトウェア	－	－	－
46	0D810000	System down (BPU Error)	BPUエラー	ハードウェア	CMU	CMU STOP	ハードウェア交換
47	03820000	System down (Memory Error)	メモリエラー	ハードウェア	CMU	CMU STOP	ハードウェア交換
48	038A0000	System down (Memory Access Error)	メモリアクセスエラー	ハードウェア	CMU	CMU STOP	ハードウェア交換
49	038B0000	System down (Internal Bus Parity)	内部バスパリティエラー	ハードウェア	CMU	CMU STOP	ハードウェア交換
50	038C0000	System down (System Bus Parity)	システムバスパリティエラー	ハードウェア	CMU	CMU STOP	ハードウェア交換
51	038F0000	System down (Undefined Machine Check)	未定義マシンチェックエラー	ハードウェア	CMU	CMU STOP	ハードウェア交換
52	03620000	System down (Invalid Data Access)	データアクセスエラー	ソフトウェア	CPMS	CMU STOP	プログラム修正
53	03660000	System down (Data Access Protection)	データアクセスプロテクトエラー	ソフトウェア	CPMS	CMU STOP	プログラム修正
54	03600000	System down (Data Page Fault)	データアクセスページフォールト	ソフトウェア	CPMS	CMU STOP	プログラム修正
55	03420000	System down (Invalid Inst. Access)	命令アクセスエラー	ソフトウェア	CPMS	CMU STOP	プログラム修正
56	03460000	System down (Inst. Access Protection)	命令アクセスプロテクトエラー	ソフトウェア	CPMS	CMU STOP	プログラム修正
57	03400000	System down (Instruction Page Fault)	命令アクセスページフォールト	ソフトウェア	CPMS	CMU STOP	プログラム修正
58	03030000	System down (Inst. Alignment Error)	命令アラインメントエラー	ソフトウェア	CPMS	CMU STOP	プログラム修正
59	03040000	System down (Illegal Instruction)	不当命令エラー	ソフトウェア	CPMS	CMU STOP	プログラム修正
60	03380000	System down (FP Unavailable)	浮動小数点使用不可例外	ソフトウェア	CPMS	CMU STOP	プログラム修正
61	03390000	System down (FP System down)	浮動小数点演算エラー	ソフトウェア	CPMS	CMU STOP	プログラム修正
62	03470000	System down (Data Alignment Error)	データアラインメントエラー	ソフトウェア	CPMS	CMU STOP	プログラム修正
63	030F0000	System down (Illegal Exception)	不当例外エラー	ソフトウェア	CPMS	CMU STOP	プログラム修正
64	05700000	System down (System Error)	システムダウン（システムエラー）	ソフトウェア	CPMS	CMU STOP	プログラム修正
65	05800000	System down (Kernel Trap)	システムダウン（カーネルトラップ）	ソフトウェア	CPMS	CMU STOP	プログラム修正
66	03620000	ULSUB down (Invalid Data Access)	データアクセスエラー	ソフトウェア	ULSUB	CMU STOP	プログラム修正
67	03660000	ULSUB down (Data Access Protection)	データアクセスプロテクトエラー	ソフトウェア	ULSUB	CMU STOP	プログラム修正
68	03600000	ULSUB down (Data Page Fault)	データアクセスページフォールト	ソフトウェア	ULSUB	CMU STOP	プログラム修正
69	03420000	ULSUB down (Invalid Inst. Access)	命令アクセスエラー	ソフトウェア	ULSUB	CMU STOP	プログラム修正
70	03460000	ULSUB down (Inst. Access Protection)	命令アクセスプロテクトエラー	ソフトウェア	ULSUB	CMU STOP	プログラム修正
71	03400000	ULSUB down (Instruction Page Fault)	命令アクセスページフォールト	ソフトウェア	ULSUB	CMU STOP	プログラム修正
72	03030000	ULSUB down (Inst. Alignment Error)	命令アラインメントエラー	ソフトウェア	ULSUB	CMU STOP	プログラム修正
73	03080000	ULSUB down (Privileged Instruction)	特権命令エラー	ソフトウェア	ULSUB	CMU STOP	プログラム修正
74	03040000	ULSUB down (Illegal Instruction)	不当命令エラー	ソフトウェア	ULSUB	CMU STOP	プログラム修正
75	03380000	ULSUB down (FP Unavailable)	浮動小数点使用不可例外	ソフトウェア	ULSUB	CMU STOP	プログラム修正
76	03390000	ULSUB down (FP System down)	浮動小数点演算エラー	ソフトウェア	ULSUB	CMU STOP	プログラム修正
77	03470000	ULSUB down (Data Alignment Error)	データアラインメントエラー	ソフトウェア	ULSUB	CMU STOP	プログラム修正
78	030F0000	ULSUB down (Illegal Exception)	不当例外エラー	ソフトウェア	ULSUB	CMU STOP	プログラム修正
79	05140000	System down (ULSUB Stop)	ｼｽﾃﾑﾀﾞｳﾝ（組み込みｻﾌﾞﾙｰﾁﾝｽﾄｯﾌﾟ）	ソフトウェア	ULSUB	CMU STOP	プログラム修正
80	05F00000	Program Error (ADT Error)	メモリアクセス検出	ソフトウェア	TASK	ログ	プログラム修正
81	00000201	Message frame error	メッセージフレームエラー	ソフトウェア	NXACP	－	－
82	00000401	Buffer status	バッファ状態報告	ソフトウェア	NXACP	－	－
83	00000501	Socket error	ソケットエラー	ソフトウェア	NXACP	－	－
84	00000601	Transfer memory address error	転写エリア重複エラー	ソフトウェア	TASK	－	プログラム修正

付録C　組み込みサブルーチンの入力データ

(1) CPESの入力データフォーマット（PRGEB）

	名称	説明
0	pge_form	LOG_FORM_PRGERR
4	pge_frsz	pge_ecd以降のﾃﾞｰﾀｻｲｽﾞ（ﾊﾞｲﾄ単位）
8	pge_ecd	エラーコード
12	pge_tn	タスク番号
16	pge_gr0_b0	汎用レジスタR0_BANK0
20	pge_gr1_b0	汎用レジスタR0_BANK0
24	pge_gr2_b0	汎用レジスタR0_BANK0
28	pge_gr3_b0	汎用レジスタR0_BANK0
32	pge_gr4_b0	汎用レジスタR0_BANK0
36	pge_gr5_b0	汎用レジスタR0_BANK0
40	pge_gr6_b0	汎用レジスタR0_BANK0
44	pge_gr7_b0	汎用レジスタR0_BANK0
48	pge_gr8	汎用レジスタR8
52	pge_gr9	汎用レジスタR9
56	pge_gr10	汎用レジスタR10
60	pge_gr11	汎用レジスタR11
64	pge_gr12	汎用レジスタR12
68	pge_gr13	汎用レジスタR13
72	pge_gr14	汎用レジスタR14
76	pge_gr15	汎用レジスタR15
80	pge_pc	プログラムカウンタ
84	pge_sr	ステータスレジスタ
88	pge_pr	プロシジャレジスタ
92	pge_gbr	グローバルベースレジスタ
96	pge_mach	積和上位レジスタ
100	pge_macl	積和下位レジスタ
104	pge_expevt	expevtレジスタ
108	pge_fadr	Fault Address
112	pge_fr0	浮動小数点ﾚｼﾞｽﾀFPR0_BANK0
116	pge_fr1	浮動小数点ﾚｼﾞｽﾀFPR1_BANK0
120	pge_fr2	浮動小数点ﾚｼﾞｽﾀFPR2_BANK0
124	pge_fr3	浮動小数点ﾚｼﾞｽﾀFPR3_BANK0
128	pge_fr4	浮動小数点ﾚｼﾞｽﾀFPR4_BANK0
132	pge_fr5	浮動小数点ﾚｼﾞｽﾀFPR5_BANK0
136	pge_fr6	浮動小数点ﾚｼﾞｽﾀFPR6_BANK0
140	pge_fr7	浮動小数点ﾚｼﾞｽﾀFPR7_BANK0
144	pge_fr8	浮動小数点ﾚｼﾞｽﾀFPR8_BANK0
148	pge_fr9	浮動小数点ﾚｼﾞｽﾀFPR9_BANK0
152	pge_fr10	浮動小数点ﾚｼﾞｽﾀFPR10_BANK0
156	pge_fr11	浮動小数点ﾚｼﾞｽﾀFPR11_BANK0
160	pge_fr12	浮動小数点ﾚｼﾞｽﾀFPR12_BANK0
164	pge_fr13	浮動小数点ﾚｼﾞｽﾀFPR13_BANK0
168	pge_fr14	浮動小数点ﾚｼﾞｽﾀFPR14_BANK0
172	pge_fr15	浮動小数点ﾚｼﾞｽﾀFPR15_BANK0
176	pge_fr16	浮動小数点ﾚｼﾞｽﾀFPR0_BANK1
180	pge_fr17	浮動小数点ﾚｼﾞｽﾀFPR1_BANK1
184	pge_fr18	浮動小数点ﾚｼﾞｽﾀFPR2_BANK1
188	pge_fr19	浮動小数点ﾚｼﾞｽﾀFPR3_BANK1
192	pge_fr20	浮動小数点ﾚｼﾞｽﾀFPR4_BANK1
196	pge_fr21	浮動小数点ﾚｼﾞｽﾀFPR5_BANK1
200	pge_fr22	浮動小数点ﾚｼﾞｽﾀFPR6_BANK1
204	pge_fr23	浮動小数点ﾚｼﾞｽﾀFPR7_BANK1
208	pge_fr24	浮動小数点ﾚｼﾞｽﾀFPR8_BANK1
212	pge_fr25	浮動小数点ﾚｼﾞｽﾀFPR9_BANK1
216	pge_fr26	浮動小数点ﾚｼﾞｽﾀFPR10_BANK1
220	pge_fr27	浮動小数点ﾚｼﾞｽﾀFPR11_BANK1
224	pge_fr28	浮動小数点ﾚｼﾞｽﾀFPR12_BANK1
228	pge_fr29	浮動小数点ﾚｼﾞｽﾀFPR13_BANK1
232	pge_fr30	浮動小数点ﾚｼﾞｽﾀFPR14_BANK1
236	pge_fr31	浮動小数点ﾚｼﾞｽﾀFPR15_BANK1
240	pge_fpscr	浮動小数点ｽﾃｰﾀｽ、ｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ
244	pge_fpul	浮動小数点通信レジスタ
248	pge_iarvn9	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ-36の内容
252	pge_iarvn8	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ-32の内容
256	pge_iarvn7	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ-28の内容
260	pge_iarvn6	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ-24の内容
264	pge_iarvn5	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ-20の内容
268	pge_iarvn4	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ-16の内容
272	pge_iarvn3	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ-12の内容
276	pge_iarvn2	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ-8の内容
280	pge_iarvn1	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ-4の内容
284	pge_iarv0	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽの内容
288	pge_iarv1	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ+4の内容

(2) IESの入力データフォーマット（IOERB）

	名称	説明
0	ioe_form	フォーマットタイプ（この場合はLOG_FORM_IOERR）
4	ioe_frsz	ioe_ecd以降のデータサイズ（バイト単位）
8	ioe_ecd	エラーコード
12	ioe_uno	ユニット番号
16	ioe_dev	デバイス番号
20	ioe_dva	デバイスアドレス
24	ioe_ioec	詳細エラーコード
28	ioe_tn	タスク番号（無効の場合は-1）
32	ioe_data[110]	I/Oエラーの詳細情報（I/Oごとに異なる）
472		

(3) EASの入力データフォーマット（ADB）

	名称	説明
0	adb_logno	エラーログ番号
4	adb_timestamp	時刻（ホストクロックの値）
8	adb_type	重要度タイプ
12	adb_class	障害検出コンポーネントクラス
16	adb_retcode	障害検出時の関数のリターンコード
18	adb_errtype	故障種別（ハードウェア／CPMS／その他）
20	adb_flag	エラーメッセージフラグ（表示抑止等）
24	adb_site[16]	サイト名称
40	erb[118]	エラーブロック（障害報告データ） エリアサイズは472バイト固定だが、実効あるデータのサイズはフォーマットタイプにより異なる。詳細はエラーログフォーマット参照。
512	adb_dhpbuf[128]	DHPデータ（512バイト）
1024		

(4) PCKSの入力データフォーマット（SVCEB）

	名称	説明
0	sve_form	フォーマットタイプ（この場合はLOG_FORM_PARAMERR）
4	sve_frsz	sve_ecd以降のデータサイズ（バイト単位）
8	sve_ecd	エラーコード
12	sve_tn	タスク番号
16	sve_svc	マクロID
20	sve_epn	エラーパラメータ番号
24	sve_p1	マクロ命令パラメータ1
28	sve_p2	マクロ命令パラメータ2
32	sve_p3	マクロ命令パラメータ3
36	sve_p4	マクロ命令パラメータ4
40	sve_p5	マクロ命令パラメータ5
44	sve_p6	マクロ命令パラメータ6
48	sve_p7	マクロ命令パラメータ7

(5) MODESの入力データフォーマット（HARDEB）

	名称	説明
0	mde_form	フォーマットタイプ（この場合はLOG_FORM_MODULERR）
4	mde_frsz	mde_ecd以降のデータサイズ（バイト単位）
8	mde_ecd	エラーコード
12	mde_slot	スロット番号
16	mde_msw0	モジュールステータスワード0（無効の場合は-1）
20	mde_msw1	モジュールステータスワード1（無効の場合は-1）
24	mde_data[112]	モジュールエラー詳細フォーマット
472		

(6) ADTSの入力データフォーマット（ADTDB）

	名称	説明
0	adt_form	LOG_FORM_ADTERR
4	adt_frsz	adt_ecd以降のﾃﾞｰﾀｻｲｽﾞ（ﾊﾞｲﾄ単位）
8	adt_ecd	エラーコード
12	adt_tn	タスク番号
16	adt_gr0	汎用レジスタR0_BANK0
20	adt_gr1	汎用レジスタR1_BANK0
24	adt_gr2	汎用レジスタR2_BANK0
28	adt_gr3	汎用レジスタR3_BANK0
32	adt_gr4	汎用レジスタR4_BANK0
36	adt_gr5	汎用レジスタR5_BANK0
40	adt_gr6	汎用レジスタR6_BANK0
44	adt_gr7	汎用レジスタR7_BANK0
48	adt_gr8	汎用レジスタR8
52	adt_gr9	汎用レジスタR9
56	adt_gr10	汎用レジスタR10
60	adt_gr11	汎用レジスタR11
64	adt_gr12	汎用レジスタR12
68	adt_gr13	汎用レジスタR13
72	adt_gr14	汎用レジスタR14
76	adt_gr15	汎用レジスタR15
80	adt_pc	プログラムカウンタ
84	adt_sr	ステータスレジスタ
88	adt_pr	プロシジャレジスタ
92	adt_gbr	グローバルベースレジスタ
96	adt_mach	積和上位レジスタ
100	adt_macl	積和下位レジスタ
104	adt_expevt	expevtレジスタ
108	adt_fadr1	Fault Address1
112	adt_fadr2	Fault Address2
116	adt_bara	ブレークアドレスレジスタA
120	adt_bamra	ブレークアドレスマスクレジスタA
124	adt_bbra	ブレークバスサイクルレジスタA
128	adt_basra	ブレークA SIDレジスタA
132	adt_bamrb	ブレークアドレスレジスタB
136	adt_bamrb	ブレークアドレスマスクレジスタB
140	adt_bbrb	ブレークバスサイクルレジスタB
144	adt_basrb	ブレークA SIDレジスタB
148	adt_brcr	ブレークコントロールレジスタ
152	adt_fr0	浮動小数点ﾚｼﾞｽﾀFPR0_BANK0
156	adt_fr1	浮動小数点ﾚｼﾞｽﾀFPR1_BANK0
160	adt_fr2	浮動小数点ﾚｼﾞｽﾀFPR2_BANK0
164	adt_fr3	浮動小数点ﾚｼﾞｽﾀFPR3_BANK0
168	adt_fr4	浮動小数点ﾚｼﾞｽﾀFPR4_BANK0
172	adt_fr5	浮動小数点ﾚｼﾞｽﾀFPR5_BANK0
176	adt_fr6	浮動小数点ﾚｼﾞｽﾀFPR6_BANK0
180	adt_fr7	浮動小数点ﾚｼﾞｽﾀFPR7_BANK0
184	adt_fr8	浮動小数点ﾚｼﾞｽﾀFPR8_BANK0
188	adt_fr9	浮動小数点ﾚｼﾞｽﾀFPR9_BANK0
192	adt_fr10	浮動小数点ﾚｼﾞｽﾀFPR10_BANK0
196	adt_fr11	浮動小数点ﾚｼﾞｽﾀFPR11_BANK0
200	adt_fr12	浮動小数点ﾚｼﾞｽﾀFPR12_BANK0
204	adt_fr13	浮動小数点ﾚｼﾞｽﾀFPR13_BANK0
208	adt_fr14	浮動小数点ﾚｼﾞｽﾀFPR14_BANK0
212	adt_fr15	浮動小数点ﾚｼﾞｽﾀFPR15_BANK0
216	adt_fr16	浮動小数点ﾚｼﾞｽﾀFPR0_BANK1
220	adt_fr17	浮動小数点ﾚｼﾞｽﾀFPR1_BANK1
224	adt_fr18	浮動小数点ﾚｼﾞｽﾀFPR2_BANK1
228	adt_fr19	浮動小数点ﾚｼﾞｽﾀFPR3_BANK1
232	adt_fr20	浮動小数点ﾚｼﾞｽﾀFPR4_BANK1
236	adt_fr21	浮動小数点ﾚｼﾞｽﾀFPR5_BANK1
240	adt_fr22	浮動小数点ﾚｼﾞｽﾀFPR6_BANK1
244	adt_fr23	浮動小数点ﾚｼﾞｽﾀFPR7_BANK1
248	adt_fr24	浮動小数点ﾚｼﾞｽﾀFPR8_BANK1
252	adt_fr25	浮動小数点ﾚｼﾞｽﾀFPR9_BANK1
256	adt_fr26	浮動小数点ﾚｼﾞｽﾀFPR10_BANK1
260	adt_fr27	浮動小数点ﾚｼﾞｽﾀFPR11_BANK1
264	adt_fr28	浮動小数点ﾚｼﾞｽﾀFPR12_BANK1
268	adt_fr29	浮動小数点ﾚｼﾞｽﾀFPR13_BANK1
272	adt_fr30	浮動小数点ﾚｼﾞｽﾀFPR14_BANK1
276	adt_fr31	浮動小数点ﾚｼﾞｽﾀFPR15_BANK1
280	adt_fpscr	浮動小数点ｽﾃｰﾀｽ、ｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ
284	adt_fpul	浮動小数点通信ﾚｼﾞｽﾀ
288	adt_iarvn9	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ-36の内容
292	adt_iarvn8	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ-32の内容
296	adt_iarvn7	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ-28の内容
300	adt_iarvn6	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ-24の内容
304	adt_iarvn5	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ-20の内容
308	adt_iarvn4	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ-16の内容
312	adt_iarvn3	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ-12の内容
316	adt_iarvn2	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ-8の内容
320	adt_iarvn1	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ-4の内容
324	adt_iarv0	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽの内容
328	adt_iarv1	ﾌﾟﾛｸﾞﾗﾑｶｳﾝﾀの指すｱﾄﾞﾚｽ+4の内容

ご利用者各位

〒１０１－８０１０
東京都千代田区神田駿河台４丁目６番地
株 式 会 社　日　立　製　作　所

お　願　い

各位にはますますご清栄のことと存じます。
さて、この資料をより良くするために、お気付きの点はどんなことでも結構ですので、下欄にご記入の上、当社営業担当または当社所員に、お渡しくださいますようお願い申しあげます。なお、製品開発、サービス、その他についてもご意見を併記して頂ければ幸甚に存じます。

ご 住 所　〒	
貴会社名 (団体名)	
芳　　名	
製 品 名	
ご意見欄	