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。

User's Manual

RENESAS

M3T-MR32R V.3.50

ユーザーズマニュアル

M32R ファミリ用リアルタイムOS

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2003.06

安全設計に関するお願い

- 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

- 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報について株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
- 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは責任を負いません。
- 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは、予告なしに、本資料に記載した製品又は仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前に株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又は特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ（http://www.renesas.com）などを通じて公開される情報に常にご注意ください。
- 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズはその責任を負いません。
- 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは、適用可否に対する責任は負いません。
- 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又は特約店へご照会ください。
- 本資料の転載、複製については、文書による株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズの事前の承諾が必要です。
- 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたら株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又は特約店までご照会ください。

製品の内容及び本書についてのお問い合わせ先

インストーラが生成する以下のテキストファイルに必要事項を記入の上、ツール技術サポート窓口 support_tool@renesas.com まで送信ください。

¥SUPPORT¥製品名¥SUPPORT.TXT

株式会社ルネサス ソリューションズ マイコンツール部
ツール技術サポート窓口　support_tool@renesas.com
ユーザ登録窓口　regist_tool@renesas.com
ホームページ　http://www.renesas.com/jp/tools

はじめに

M3T-MR32R(**以下** MR32R **と略す**)は M32R **ファミリ**用のリアルタイム・オペレーティングシステム[1]です。MR32R は**μ**ITRON 仕様[2]に準拠しています。

本マニュアルは MR32R を使用したプログラムの作成手順および作成上の注意事項について説明します。各システムコールの詳細な使用方法については「MR32R リファレンスマニュアル」を参照してください。

MR32R を使うために必要なこと

MR32R を使用したプログラムを作成するには弊社下記製品または、サードパーティ製品を別途御購入して頂く必要があります。

- M32R **ファミリ**クロスツール M3T-CC32R(**以下** CC32R **と略す**)
- M32R **ファミリ** GNU クロスツール M3T-TW32R(以下 TW32R と略す)
- M32R **ファミリ** Wind River Systems,Inc **製コンパイラ** D-CC/M32R

これらの製品をあわせて御使用頂ければ、より効率の良いプログラム開発がおこなえます。

ドキュメント一覧

MR32R に添付されているドキュメントは以下の３種類あります。

- リリースノート
  ソフトウェアの概要やユーザーズマニュアル、リファレンスマニュアルの訂正などを記載したドキュメントです。
- ユーザーズマニュアル (PDF ファイル)
  MR32R を使用したプログラムの作成手順や作成上の注意事項を記載したドキュメントです。
- リファレンスマニュアル (PDF ファイル)
  MR32R のシステムコールの使用方法や使用例を記述したドキュメントです。
  本マニュアルを読む前に必ずリリースノートをお読みください。

ソフトウェアの使用権

ソフトウェアの使用権はソフトウェア使用権許諾契約書に基づきます。MR32R はお客様の製品開発の目的でのみ使用できます。その他の目的での使用はできませんのでご注意ください。

また、本マニュアルによってソフトウェアの使用権の実施に対する保証及び使用権の実施の許諾を行うものではありません。

---

[1] 以降リアルタイム OS と略します。

[2] μITRON 仕様は、東京大学理学部坂村健博士とその研究室により考案されたものです。したがって、μITRON 仕様の著作権は同氏に属しています。**MR32R** は同氏に承認を得て、μITRON 仕様に基づき製作されたものです。

目次

図目次

表目次

第 1 章　　ユーザーズマニュアルの構成

M3T-MR32R(**以下** MR32R **と略す**)ユーザーズマニュアルは、９つの章から構成されています。

- 第 2 章 概要
  MR32R の目的や、概略の機能、位置づけなどを説明します。
- 第 3 章 MR32R 入門
  MR32R を使用する上で必要となる考え方や用語などを説明します。
- 第 4 章 アプリケーション作成手順概要
  MR32R を使用してアプリケーションプログラムを作成する場合の開発手順の概要を説明します。
- 第 5 章 アプリケーション作成手順詳細
  MR32R を使用してアプリケーションプログラムを作成する場合の開発手順を詳細に説明します。
- 第 7 章 コンフィグレータの使用方法
  コンフィグレーションファイルの記述方法、および、コンフィグレータの使用方法を詳細に説明します。
- 第 8 章　各設定ファイルのカスタマイズ方法
  スタートアップファイル、割り込み制御プログラム、セクションファイルについて説明します。
- 第 9 章　アプリケーション作成の手引き
  MR32R を使用してアプリケーションプログラムを作成する際に知っておいたほうがよい事項や、注意事項について説明します。
- 第 10 章 付録
  製品にソースファイル形式で含まれている MR32R サンプルアプリケーションプログラムについて説明します。

第 2 章　　概要

2.1 MR32R のねらい

近年マイクロコンピュータの急激な進歩にともない、マイクロコンピュータ応用製品の機能が複雑化してきています。これにともない、マイクロコンピュータのプログラムサイズが大きくなってきています。また製品開発競争が激化しマイクロコンピュータ応用製品を短期間に開発しなければなりません。すなわち、マイクロコンピュータのソフトウェアを開発している技術者は今までより大きなプログラムを今までより短期間で開発することが要求されてきます。そこでこの困難な要求を解決するためには以下のことを考えていかなければなりません。

1. **ソフトウェアの再利用性を高めて、 開発すべきソフトウェアの量を削減する。**

   このためにはソフトウェアをできるだけ機能単位で独立したモジュールに分割して再利用できるようにする方法があります。すなわち、汎用サブルーチン集などを多く蓄積してそれをプログラム開発時に使用します。ただこの方法では、時間やタイミングに依存したプログラムは再利用するのは困難です。ところが実際の応用プログラムは時間やタイミングに依存したプログラムがかなりの部分を占めていてこのような手法で再利用できるプログラムはあまり多くありません。

2. **チームプログラミングを推進し、1 つのソフトウェアを何名かの技術者でおこなうようにする。**

   チームプログラミングをおこなうには色々な問題があります。1 つはデバッグ作業をおこなうにあたり、チームプログラミングをおこなっている技術者全員のソフトウェアがデバッグできる状態にないとデバッグに入れません。また、チーム内の意志統一を十分におこなう必要があります。

3. **ソフトウェアの生産効率を向上させ、技術者 1 名あたりの開発可能量を増加させる。**

   このためには 1 つは技術者の教育をおこない技術者のスキルアップをはかる方法があります。また、構造化記述アセンブラや C コンパイラなどを用いることによりより簡単にプログラムを作成できるようにする方法があります。また、ソフトウェアのモジュール化を推進してデバッグの効率を向上させる方法等があります。

しかし、このような問題を解決するには従来の手法では限界があります。そこでリアルタイム OS[3]という新しい手法の導入が必要になってきます。

そこで、弊社はこの要求に答えるべく 32 ビットマイクロプロセッサ M32R **ファミリ**用にリアルタイム OS MR32R を開発しました。MR32R を導入することにより以下のような効果があります。

4. **ソフトウェアの再利用が容易になります。**

   リアルタイム OS を導入することにより、タイミングをリアルタイム OS を介してとることにより、タイミングに依存したプログラムが再利用できるようになります。

   また、プログラムをタスクというモジュールに分割しますので、自然と構造化プログラミングをおこなうようになります。すなわち再利用可能なプログラムを自然に作成するようになります。

5. **チームプログラミングがおこないやすくなります。**

   リアルタイム OS を導入することにより、プログラムがタスクという機能単位のモジュールに分割されますので、タスク単位で開発をおこなう技術者を振り分け開発からタスク単位でデバッグまでできるようになります。とくにリアルタイム OS を導入すると、プログラムが全てでき上がっていなくてもタスクさえ出来ていればその部分のデバッグを始めることが容易にできます。またタスク単位で技術者を割り振ることができますので、作業分担が容易におこなえます。

---

[3] OS：Operating System

6. ソフトウェアの独立性が高くなり、 プログラムをデバックしやすくなります。

リアルタイム OS を導入することにより、プログラムをタスクという独立した小さなモジュールに分割できますので、プログラムをデバッグする際ほとんどはその小さなモジュールに着目するだけでデバッグすることができます。

7. タイマ制御が簡単になります。

従来例えば、10mSec ごとにある処理を動作させるためには、マイクロコンピュータのタイマ機能を用いて定期的に割り込みを発生させて処理させていました。ところが、マイクロコンピュータのタイマの数には限りがありますのでタイマが足らなくなったら 1 本のタイマを複数の処理に使用するなどの手法を用いて解決していました。

ところがリアルタイム OS を導入することにより、リアルタイム OS の時間管理機能を使用して一定時間毎にある処理をさせるというプログラムを、マイクロコンピュータのタイマ機能を特に意識せずに作成することができます。また、同時にプログラマから見たとき疑似的にマイクロコンピュータに無限本数のタイマが搭載されたようにプログラムを作成することができます。

8. ソフトウェアの保守性が向上します。

リアルタイム OS を導入することにより開発したソフトウェアが小さなタスクと呼ばれるプログラムの集合で構成されます。これにより開発完了後保守をおこなう場合、小さなタスクだけを保守すればよくなり保守性が向上します。

9. ソフトウェアの信頼性が向上します。

リアルタイム OS を導入することにより、プログラムの評価、試験などがタスクという小さなモジュール単位でおこなえますので評価、試験が容易になりひいては信頼性が向上します。

10. マイクロコンピュータの性能を最大限生かすことができます。これにより応用製品の性能向上が望めます。

リアルタイム OS を導入することにより、入出力待ちなどのマイクロコンピュータのむだな動作を減少させることができます。これによりマイクロコンピュータの能力を最大限に引き出すことができます。ひいては応用製品の性能向上につながります。

2.2 TRON 仕様と MR32R

TRON 仕様とは The Realtime Operating system Nucleus 仕様の略で、リアルタイム・オペレーティングシステムの核となる部分の仕様を意味します。TRON 仕様の設計を中心とした TRON プロジェクトは、東京大学理学部 坂村健博士を中心として進められています。

この TRON プロジェクトで推進されているものの 1 つに ITRON 仕様があります。ITRON 仕様は Industrial TRON 仕様の略で、産業用組み込みシステムをターゲットとしたリアルタイム・オペレーティングシステムの仕様です。

ITRON 仕様は広い分野の応用に十分対応できるように数多くの機能を有しています。このため ITRON システムは比較的大きなメモリ容量と処理能力を必要とします。μITRON 仕様 V.2.0 は、この ITRON 仕様を処理速度を向上させるため仕様を整理し、必要十分な機能のみにサブセット化されたものです。μITRON 仕様 V.2.0 は以下の項目において ITRON 仕様のサブセットになっています。

1. **システムコールのタイムアウト機能がありません。**

2. **タスク、セマフォ等のオブジェクトは、システム作成時のみに生成[4]することができます。システム起動後に生成[5]することはできません。**

3. **メモリプールは固定長のみで、可変長のメモリプールは扱えません。**

4. **システムコール例外管理機能および CPU 例外管理機能がありません。**

現在、μITRON 仕様 V.3.0 が規定されています。このμITRON 仕様 V.3.0 はμITRON 仕様 V.2.0 と ITRON 仕様を統合し、接続機能を追加したものです。

MR32R は、このμITRON 仕様[6]に従って 32 ビットマイクロプロセッサ MR32R **ファミリ**用に開発された、リアルタイム・オペレーティングシステムです。

μITRON 仕様 V.3.0 では、各システムコールは、レベル R、レベル S、レベル E、レベル C に分けられています。

MR32R は、μITRON 仕様 V.3.0 で規定されたシステムコールのうち、レベル R、レベル S の全部と、レベル E の大部分をインプリメントしています。

MR32R の概略仕様を表 2-1に示します。

---

[4] 静的オブジェクト生成
[5] 動的オブジェクト生成
[6] **MR32R** V.1.00 はμITRON 仕様 V3.0 に準拠しています。

表 2-1　MR32R 概略仕様

項目	仕様
ターゲットマイクロプロセッサ	M32R ファミリ
最大タスク数	32767
タスクの優先度数	255
最大イベントフラグ数	32766
イベントフラグの幅	32 ビット
最大セマフォ数	32766
セマフォの形式	計数型
最大メイルボックス数	32766
メッセージサイズ	32 ビット
メイルボックスのバッファサイズ	4 バイト以上
最大メッセージバッファ数	32765
最大ランデブ用ポート数	32765
最大固定長メモリプール数	32766
最大可変長メモリプール数	32766
システムコール数	116
OS 核コードサイズ	約 4K～53K バイト
OS 核データサイズ	最小 81 バイト、1 タスクあたり（スタックを除く）44 バイト増加
OS 核記述言語	C 言語、アセンブリ言語

※ セマフォ数、イベントフラグ数、メールボックス数、メッセージバッファ数×2、ランデブ用ポート数×2、優先度数、TA_TFIFO 指定された優先度付きメイルボックス数、(最大優先度付きメイルボックス数-コンフィグレーションファイルで定義した最大優先度付きメイルボックス数+コンフィグレーションファイルで定義した TA_MPRI 指定のある優先度付きメイルボックス数) ×最大優先度数の合計が 32766 以下でなければなりません。

2.3 MR32R の特長

MR32R は以下に示す特長を持っています。

1. μITRON 仕様に準拠したリアルタイム・オペレーティングシステム

MR32R は ITRON 仕様をワンチップマイクロコンピュータでも実装できるように機能を整理したμITRON 仕様に基づいて開発されました。

μITRON 仕様は ITRON 仕様のサブセットですので ITRON 教科書として出版されている文献や ITRON セミナー等で得た知識をほとんどそのまま役立てることができます。また、ITRON 仕様に準拠したリアルタイム OS を用いて開発したアプリケーションプログラムを MR32R に移行するのは比較的容易に行えます。

2. 高速処理を実現

マイコンのアーキテクチャを活用し、高速処理を実現しています。

3. 必要モジュールのみを自動選択することにより常に最小サイズのシステムを構築

MR32R は M32R ファミリオブジェクトライブラリ形式で供給されています。

したがって、リンケージエディタのもつ機能により、数ある MR32R の機能モジュールのなかで使用しているモジュールのみを自動選択してシステムを生成します。このため、常に最小サイズのシステムが自動的に生成されます。

4. C コンパイラを用いて C 言語でアプリケーションプログラムが開発可能

C コンパイラ **M3T-CC32R,M3T-TW32R,D-CC/M32R** を用いて、MR32R のアプリケーションプログラムを C 言語で開発できます。また C 言語から MR32R の機能を呼び出すためのインターフェースライブラリが添付されています。

5. 上流工程ツール "コンフィグレータ"により、容易な開発手順

ROM 書き込み形式ファイルまでの作成を簡単な定義のみでおこなえるコンフィグレータを装備しています。これにより、どんなライブラリを結合する必要があるかなどを特に気にする必要はありません。

第 3 章　　MR32R 入門

本章では、MR32R を使用する上で必要な知識や考え方を説明し、MR32R が持つ機能の使用方法を説明します。

3.1 リアルタイム OS の考え方

本節では、リアルタイム OS の基本概念について説明します。

3.1.1 リアルタイム OS の必要性

近年半導体技術の進歩にともなってシングルチップマイクロコンピュータ(マイコン)の ROM 容量が増大してきています。

このような大 ROM 容量のマイクロコンピュータの出現によりそのプログラム開発が従来の方法では困難になってきています。図 3.1にプログラムサイズと開発期間（開発の困難さ）との関係を示します。この図 3.1はあくまでイメージ図ですが、プログラムのサイズが大きくなるに従い開発期間が指数関数的に長くなってきます。

例えば 32K バイトのプログラムを 1 個開発するより、8K バイトのプログラムを 4 個開発する方が簡単です。[7]

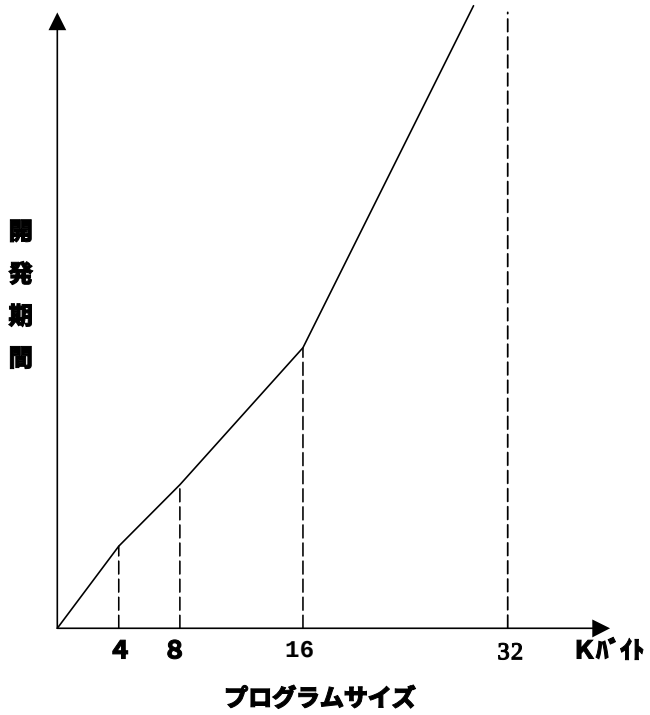


図 3.1 プログラムサイズと開発期間

そこで大きなプログラムを短期間に簡単に開発するための手法が必要になってきます。この方法として小さな ROM 容量のマイクロコンピュータを多く使う方法があります。たとえば、図 3.2にオーディオ機器システムを複数のマイクロコンピュータで構成した例を示します。

[7] ROM 詰めの必要がないことを前提とします

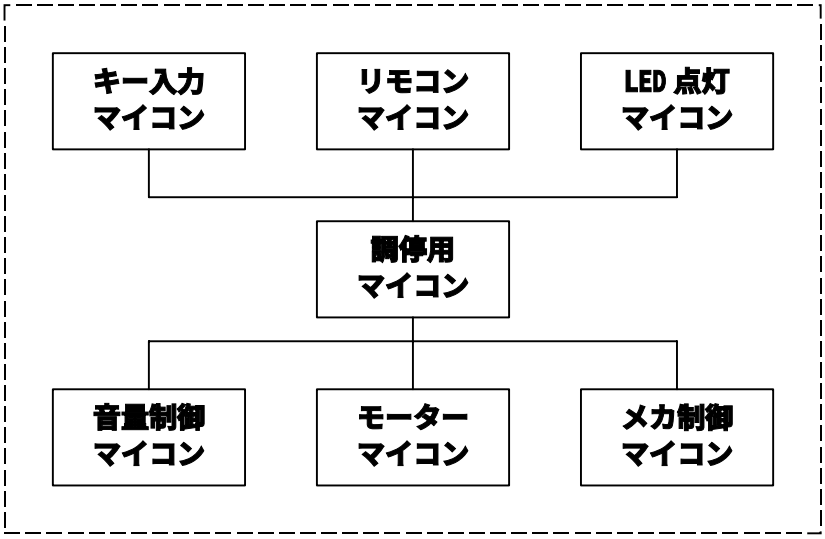


図 3.2　マイコンを多く使ったシステム例（オーディオ機器）

このように機能単位で別々のマイクロコンピュータを用いることは以下の利点があります。

1. ひとつひとつのプログラムが小さくなり、プログラム開発が容易になる。
2. 一度開発したソフトウェアを再利用することが非常に容易になる。[8]
3. 完全に機能ごとにプログラムが分離するので複数の技術者でプログラム開発が容易にできる。

この反面以下のような欠点があります。

1. 部品点数が多くなり製品の原価を上昇させる。
2. ハードウェア設計が複雑になる。
3. 製品の物理的サイズが大きくなる。

そこでそれぞれのマイクロコンピュータで動作しているプログラムを、1 つのマイクロコンピュータでソフトウェア的に、別々のマイクロコンピュータで動作しているように見せることのできるリアルタイム OS を採用すれば、上記の利点を残したままで欠点をすべて無くすことができます。
図 3.3に、図 3.2に示したシステムにリアルタイム OS を導入した場合のシステム例を示します。

[8] 例えば、図 3.2において、リモコンマイコンを別の製品にそのまま使用することができる。

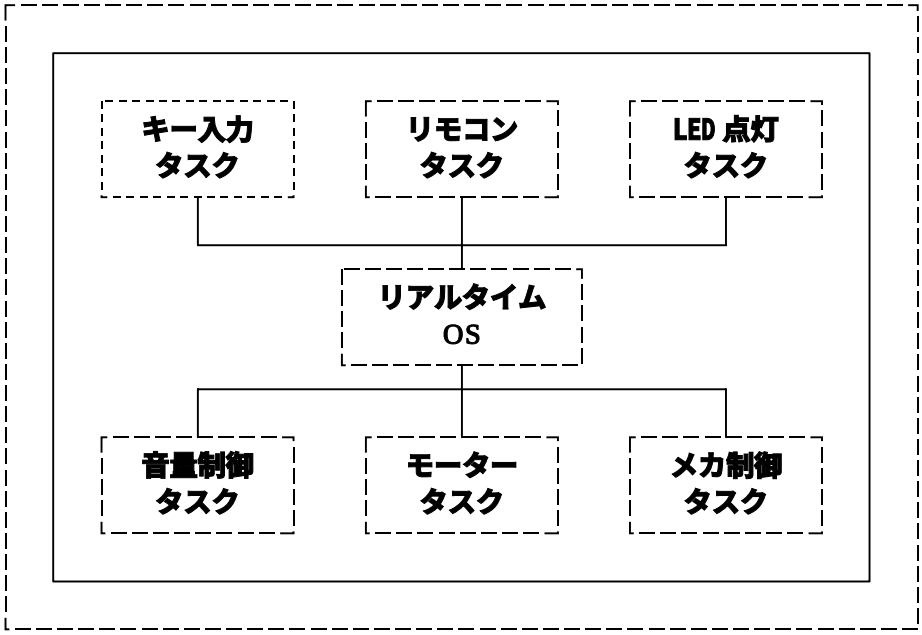


図 3.3　リアルタイム OS の導入システム例（オーディオ機器）

すなわちリアルタイム OS とは 1 個のマイクロコンピュータを、あたかも複数のマイクロコンピュータが動作しているように見せるソフトウェアです。複数のマイクロコンピュータに相当するひとつひとつのプログラムをリアルタイム OS 用語でタスクと呼びます。

3.1.2 リアルタイム OS の動作原理

リアルタイム OS は 1 個のマイクロコンピュータを、あたかも複数のマイクロコンピュータが動作しているように見せることのできるソフトウェアです。では 1 個のマイクロコンピュータをどのようにして複数あるように見せかけるのでしょうか？

それは、図 3.4に示すようにそれぞれのタスクを時分割で動作させるからです。つまり実行するタスクを一定時間ごとに切り替えて、複数のタスクが同時に実行しているように見せるのです。

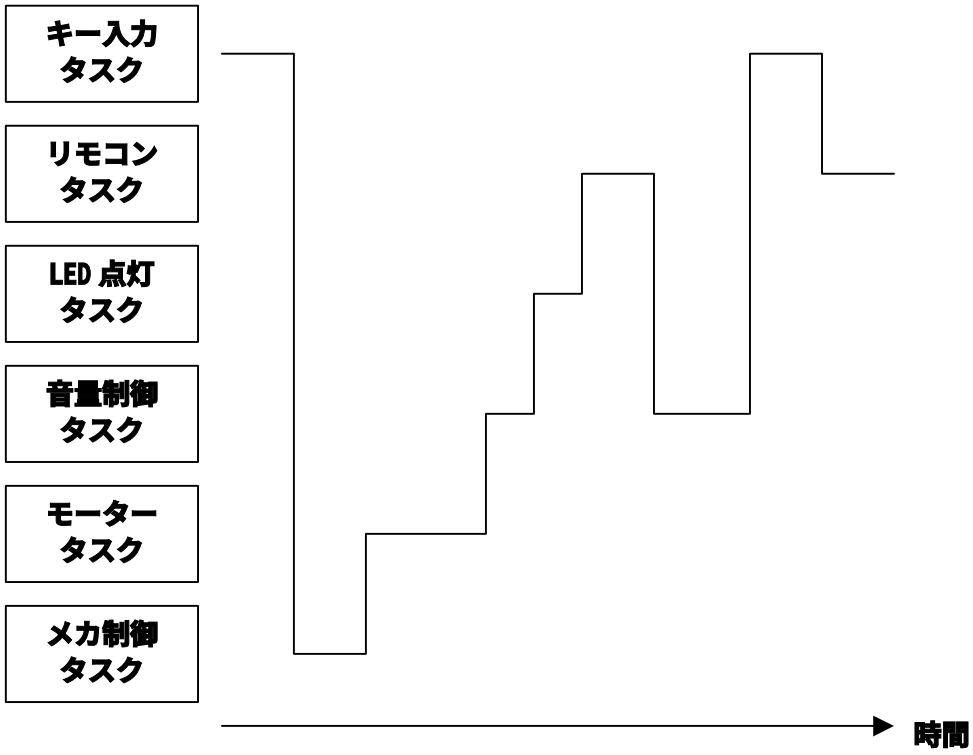


図 3.4 タスクの時分割動作

このようにタスクを一定時間ごとに切り替えて実行しています。このタスクを切り替えることをリアルタイム OS 用語でディスパッチと呼ぶこともあります。タスク切り替え(ディスパッチ)が発生する要因として以下のものがあります。

- 自分自身で切り替えを要求する。
- 割り込みなどの外的要因で切り替わる。

タスク切り替えが発生し、再度、そのタスクを実行するときには、中断していたところから再開します。 (図 3.5参照)

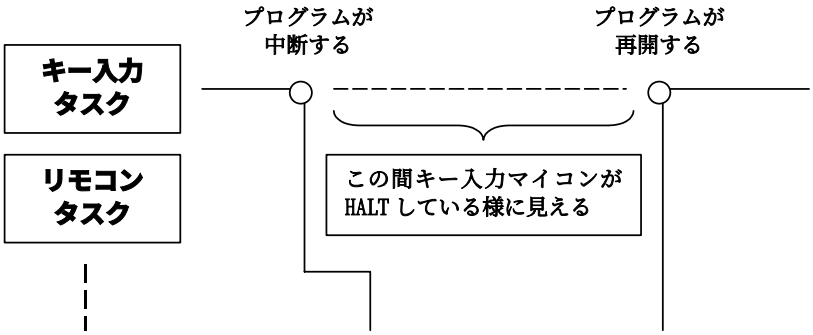


図 3.5　タスクの中断と再開

図 3.5においてキー入力タスクは、他のタスクに実行制御が移っている間、プログラマから見ればプログラムが中断しそのマイコンが HALT しているようにみえます。

タスクの実行は、中断した時点のレジスタ内容を復帰することにより、中断した時点の状態で再開されます。すなわちタスクの切り替えとは、現在実行中のタスクのレジスタの内容をそのタスクを管理するメモリ領域に退避し、切り替えるタスクのレジスタ内容を復帰することです。

すなわちリアルタイム OS を実現するには、タスクごとにレジスタを管理し、切り換えが発生する度にそのレジスタ内容を入れ換えることにより複数のマイクロコンピュータが存在しているように見せてやればよいということになります。（図 3.6参照）。

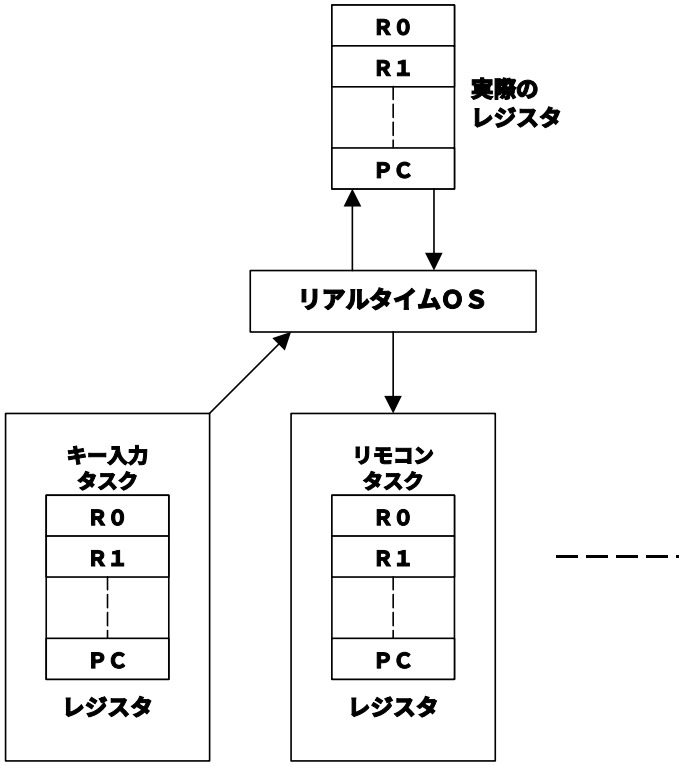


図 3.6　タスクの切り替え

図 3.7 は各タスクのレジスタをどのように管理しているか具体的に示したものです。

実際にはタスクごとに持つ必要のあるのはレジスタだけでなく、スタック領域もタスクごとに持つ必要があります。

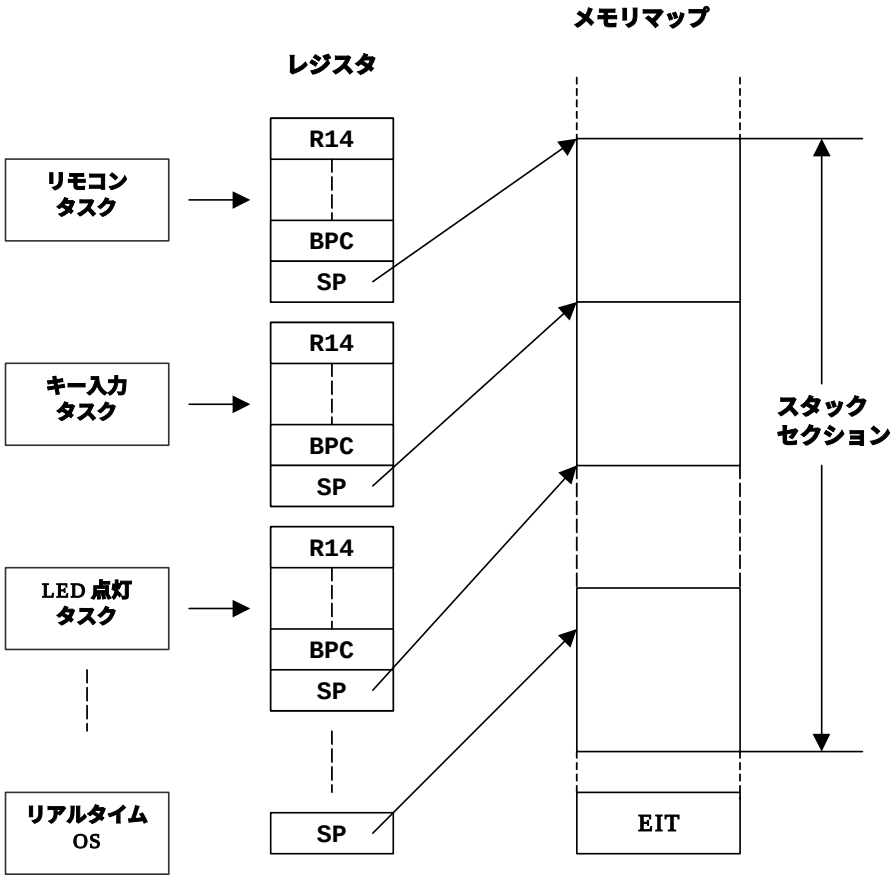


図 3.7　タスクのレジスタ領域

図 3.8は各タスクのレジスタおよびスタック領域を詳細に説明したものです。MR32R では各タスクのレジスタは図 3.8に示すようにスタック領域の中に格納され管理されています。図 3.8は、レジスタ格納後の状態を示しています。

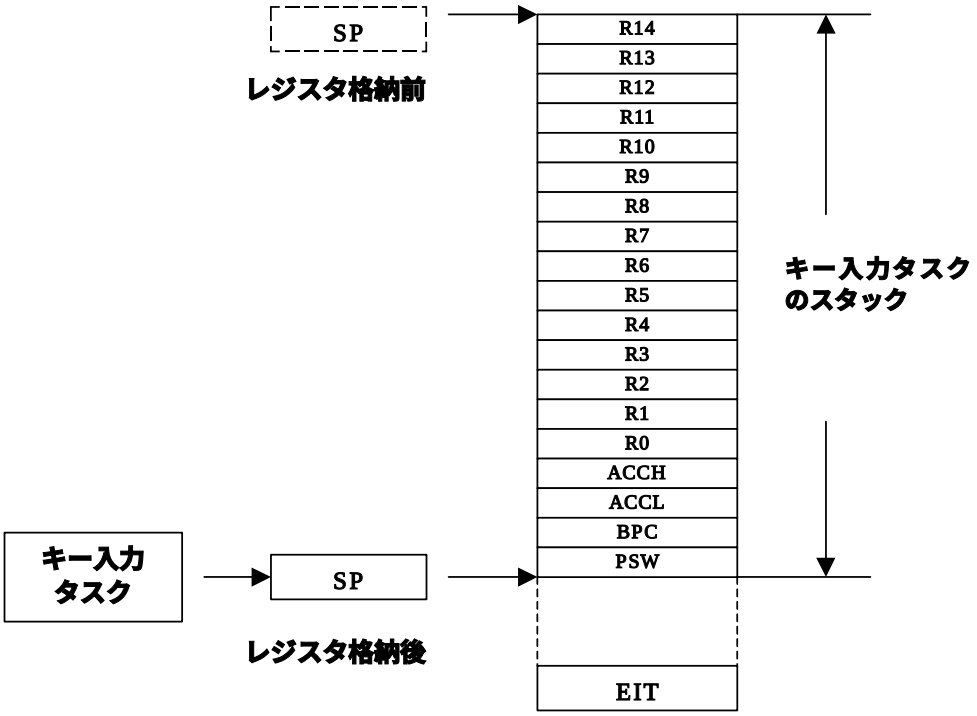


図 3.8　実際のレジスタとスタック領域の管理

3.2 システムコール

リアルタイム OS をプログラマはプログラム中でどのように使用するのでしょうか?

これにはリアルタイム OS の機能をプログラムから何らかの形で呼び出すことが必要です。こリアルタイム OS の機能を呼び出すことをシステムコールといいます。すなわちシステムコールのにより、タスクの起動などの処理を行なうことができます (図 3.9 参照)。

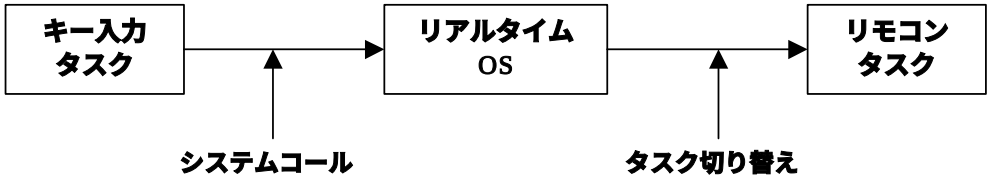


図 3.9 システムコール

このシステムコールは、C 言語で応用プログラムを記述する場合は関数呼び出しで実現します。すなわち、

```
sta_tsk(ID_main,3);
```

またアセンブリ言語で応用プログラムを記述する場合はアセンブラマクロ呼び出しにより実現します。すなわち、

```
sta_tsk ID_main,3
```

3.2.1 システムコール処理

システムコールが発行されると以下の手順により処理がおこなわれます。[9]

1. 現レジスタ内容を退避します。
2. スタックポインタをタスクのものからリアルタイム OS(システム)のものへ切り替えます。
3. システムコール要求にしたがった処理を行います。
4. 次に実行するタスクの選択をおこないます。
5. スタックポインタをタスクのものに切り替えます。
6. レジスタ内容を復帰してタスクの実行を再開します。

システムコールが発生してからタスク切り替えまでの処理の流れを図 3.10に示します。

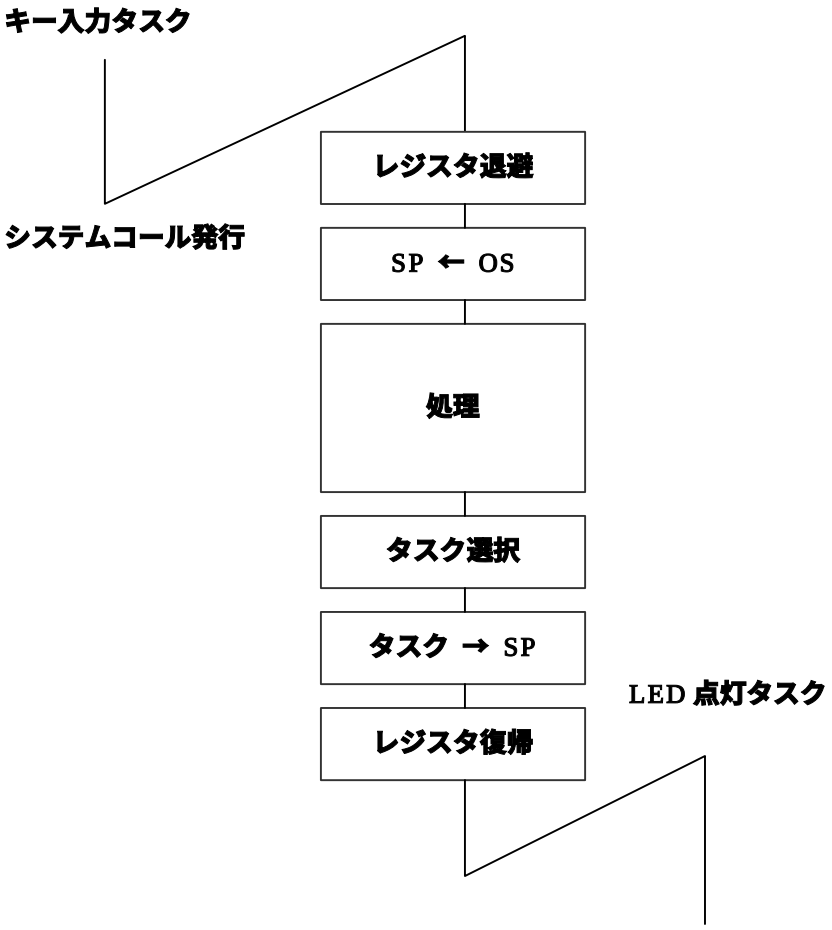


図 3.10 システムコールの処理の流れ

[9] タスク切り替えの発生しないシステムコールはこの限りではありません。

3.2.2 システムコールにおけるタスクの指定方法

各タスクの識別は、リアルタイム OS MR32R の内部では ID 番号でおこないます。
すなわち、"タスク ID 番号 1 のタスクを起動する"などというように管理されています。
しかし、プログラム中にタスクの番号を直接書き込むと非常に可読性の低いプログラムになってしまいます。たとえば、

```
sta_tsk(2,1);
```

とプログラム中に記述するとプログラマは絶えず ID 番号の 2 番のタスクは何かを知っている必要があります。また、他人がこのプログラムを見たときに ID 番号の 2 番のタスクが何かが一目では分かりません。
そこで MR32R ではタスクの識別をそのタスクの名前(関数もしくはシンボルの名前)で指定し、その名前からタスクの ID 番号への変換を MR32R に付属しているプログラム"コンフィグレータ cfg32r"が自動的におこないます。図 3.11にその様子を示します。

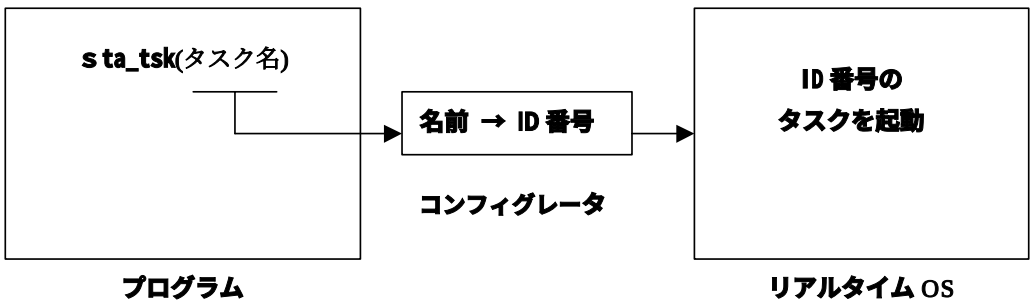


図 3.11 タスクの識別

これによりタスクの指定を以下のようにおこなえます。

```
sta_tsk(ID_task,1);
```

この例では、関数名"task()"もしくはシンボル名"task:"のタスクを起動するように指定しています。
なお、タスクの名前から ID 番号への変換は、プログラムを生成するときにおこないます。したがって、この機能による処理速度の低下はありません。

3.3 タスク

本節ではタスクをリアルタイム OS MR32R がどのように管理しているかを説明します。

3.3.1 タスクの状態

リアルタイム OS ではタスクを実行するべきか否かを、タスクの状態を管理することにより制御しています。例えば、図 3.12にキー入力タスクの実行制御と状態の関係を示します。キー入力が発生した場合はそのタスクを実行しなければなりません。すなわち、キー入力タスクが実行状態となります。またキー入力を待っているときはタスクを実行する必要はありません。すなわち、キー入力タスクは待ち状態になっています。

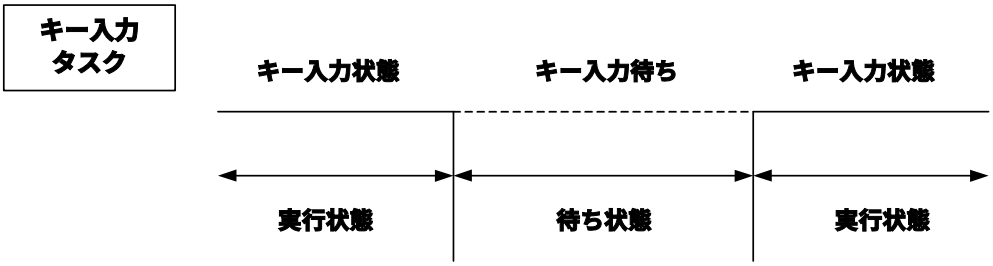


図 3.12 タスクの状態

MR32R では実行状態、待ち状態を含め以下の 6 つの状態を管理しています。

1. 実行状態 (RUN 状態)
2. 実行可能状態 (READY 状態)
3. 待ち状態 (WAIT 状態)
4. 強制待ち状態 (SUSPEND 状態)
5. 二重待ち状態 (WAIT-SUSPEND 状態)
6. 休止状態 (DORMANT 状態)
7. 未登録状態(NON-EXISTENT 状態)

タスクは上記の 7 つの状態を遷移していきます。図 3.13に、タスクの状態遷移図を示します。

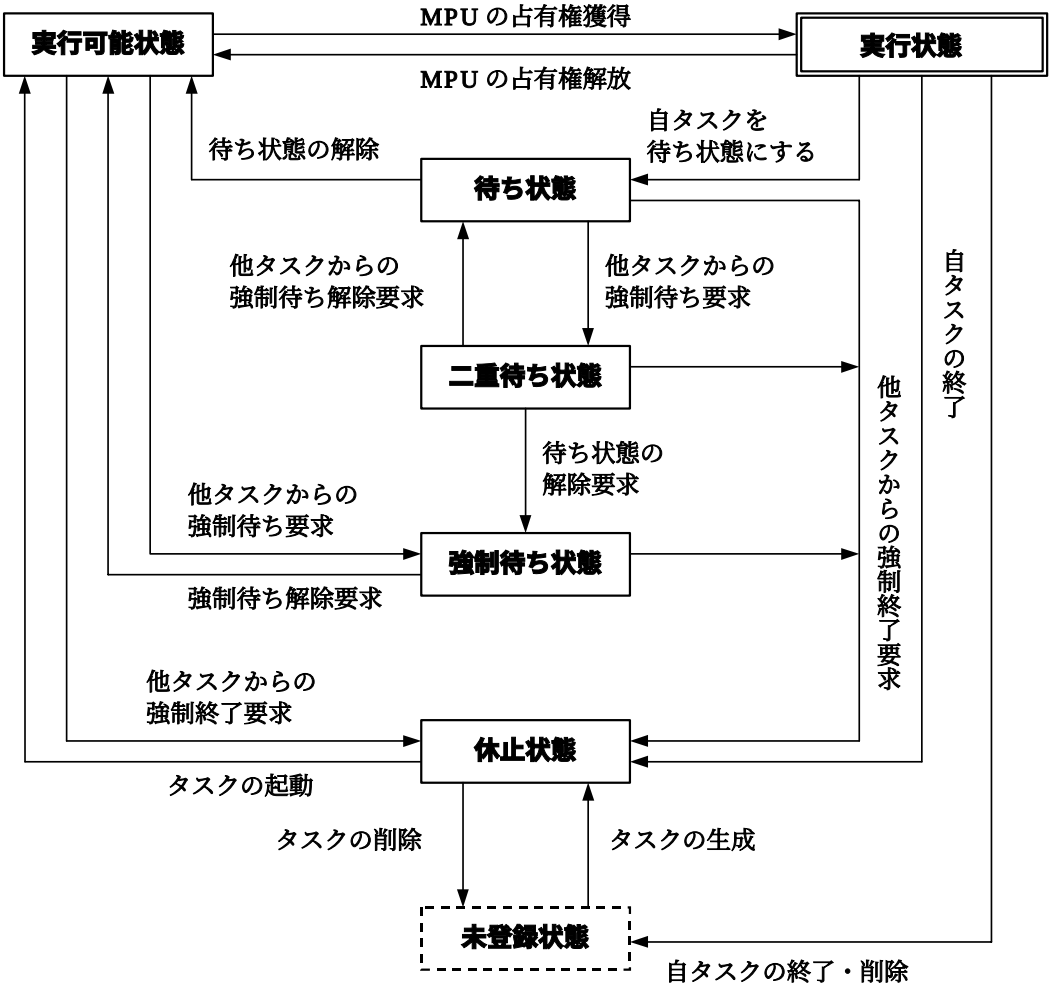


図 3.13 MR32R のタスク状態遷移図

1. 実行状態 (RUN 状態)

タスクが、まさに現在実行中の状態を実行状態といいます。マイクロコンピュータは１つしかないのですから当然実行状態にあるのは常に１つだけです。

現在実行状態のタスクが他の状態に移行するには、以下の事象のうちどれかが発生した場合です。

- 自分で自タスクを正常終了させた場合[10]
- 自分で待ち状態に入った場合
- 割り込み等の事象の発生により、その割り込みハンドラによって自タスクより優先度の高いタスクが実行可能状態になった場合
- 自タスクの優先度を変更することにより他の実行可能状態のタスクが自タスクより優先度が高くなった場合[11]

[10] ext_tsk システムコールによる
[11] chg_pri システムコールによる

- 割り込み等の事象の発生により自タスクもしくは他の実行可能状態のタスクの優先度が変更され、そのため他の実行可能状態のタスクが自タスクより優先度が高くなった場合[12]

上記の事象が発生すると再スケジュールされて実行状態と実行可能状態にあるタスクのなかで最も優先度の高いタスクが実行状態に移され、そのタスクのプログラムが実行されます。

2. 実行可能状態 (READY 状態)

タスクが実行される条件は整っているが、そのタスクより優先度の高いタスクもしくは同一優先度のタスクが実行されているために実行できずに実行待ち状態になっている状態を実行可能状態といいます。

実行可能状態であるタスクで、レディキュー[13]では 2 番目に実行される可能性のあるタスクが実行状態になるのは、以下の事象の内いずれかが発生した場合です。

- 実行状態のタスクが自分で正常終了した場合
- 実行状態のタスクが自分で待ち状態に入った場合
- 実行状態のタスクが自分で優先度を変更することにより実行可能状態のタスクが実行状態のタスクより優先度が高くなった場合[14]
- 割り込み等の事象の発生により実行状態のタスクの優先度が変更され、そのため実行可能状態のタスクが実行状態のタスクより優先度が高くなった場合[15]

3. 待ち状態 (WAIT 状態)

実行状態のタスクが自分自身を待ち状態に移行させる要求を出すことにより、タスクは実行状態から待ち状態に移行することができます。待ち状態は通常入出力装置の入出力動作完了待ちや他のタスクの処理待ちなどの状態として使用されます。

- 実行待ち状態に移行するには以下の方法があります。
- slp_tsk システムコールにより単純に待ち状態に移行します。この場合、他のタスクから明示的に待ち状態から解除されないと実行可能状態に移行しません。
- dly_tsk、tslp_tsk システムコールにより一定時間待ち状態に移行します。この場合、指定時間経過するかもしくは他のタスクから明示的に待ち状態を解除することにより実行可能状態に移行します。
- wai_flg、wai_sem、rcv_msg、snd_mbf、rcv_mbf、cal_por、acp_por、get_blf、get_blk、vrcv_mbx システムコールにより要求待ちで待ち状態に移行します。この場合、要求事項が満たされるかもしくは他のタスクから明示的に待ち状態を解除することにより実行可能状態に移行します。
- tslp_tsk、twai_flg、twai_sem、trcv_msg、tsnd_mbf、trcv_mbf、tcal_por、tacp_por、tget_blf、tget_blk、vtrcv_mbx システムコールは、slp_tsk、wai_flg、wai_sem、rcv_msg、snd_mbf、rcv_mbf、cal_por、acp_por、get_blf、get_blk、vrcv_mbx システムコールにタイムアウトを指定したシステムコールです。各システムコールの要求待ちで待ち状態に移行します。この場合、要求事項が満たされるかもしくは、指定時間が経過した場合、実行可能状態に移行します。

タスクが wai_flg、wai_sem、rcv_msg、snd_mbf、rcv_mbf、cal_por、acp_por、get_blf、get_blk、vsnd_mbx システムコール[16]により要求待ちで待ち状態になると、その要求事項によ

[12] ichg_pri システムコールによる
[13] レディーキューについては次節参照
[14] chg_pri システムコールによる
[15] icg_pri システムコールによる
[16] twai_flg,twai_sem,trcv_msg システムコールも含まれます。

り次の待ち行列のいずれかにつながります。

- イベントフラグ待ち行列
- セマフォ待ち行列
- メイルボックス待ち行列
- 送信メッセージバッファ待ち行列
- 受信メッセージバッファ待ち行列
- ランデブ用ポート呼出待ち行列
- ランデブ用ポート受付待ち行列
- 固定長メモリブロック待ち行列
- 可変長メモリブロック待ち行列
- 優先度付きメイルボックス待ち行列

4. 強制待ち状態 (SUSPEND 状態)

実行状態のタスクから sus_tsk システムコールが発行される、もしくはハンドラから isus_tsk システムコールが発行されると、システムコールにより指定された実行可能なタスクもしくは実行中のタスクは強制待ち状態になります。なお待ち状態のタスクが指定された場合は二重待ち状態になります。

強制待ち状態は入出力エラー等の発生により実行可能なタスクもしくは実行中のタスク[17]が処理を一時的に中断させるためにスケジューリングから外された状態です。すなわち実行可能状態のタスクに対して強制待ち要求が出された場合、そのタスクは実行待ち行列から外されます。

なお、強制待ち要求のキューイングは行いません。したがって強制待ち要求は実行状態、実行可能状態、待ち状態[18]にあるタスクにのみ行えます。すでに強制待ち状態にあるタスクに強制待ち要求した場合には、エラーコード E_QOVR が返されます。

5. 二重待ち状態 (WAIT-SUSPEND 状態)

待ち状態にあるタスクに強制待ちの要求が出された場合、タスクは二重待ち状態になります。wai_flg、wai_sem、rcv_msg、snd_mbf、rcv_mbf、cal_por、acp_por、get_blf、get_blk システムコールによる要求待ちで待ち状態にあるタスクに対して強制待ち要求が出された場合、そのタスクは要求待ち行列から外されず、単にそのタスクが二重待ち状態に移行するだけです。

また、二重待ち状態のタスクは待ち条件が解除されると強制待ち状態になります。待ち条件が解除されるには以下の場合が考えられます。

- wup_tsk、iwup_tsk システムコールにより起床する場合
- dly_tsk、tslp_tsk システムコールにより待ち状態になったタスクが時間経過により起床される場合
- wai_flg、wai_sem、rcv_msg、snd_mbf、rcv_mbf、cal_por、acp_por、get_blf、get_blk、

[17] ハンドラから isus_tsk システムコールにより実行タスクを強制待ち状態にした場合は、実行状態から直接強制待ち状態に移行されます。この場合のみ、例外的に実行状態から強制待ち状態に移行しますので注意してください。
[18] 待ち状態にあるタスクに対して強制待ち要求をおこなうと二重待ち状態になります。

vrcv_mbx システムコールにより待ち状態になったタスクの要求が満たされた場合

- rel_wai、irel_wai システムコールにより待ち状態が強制解除される場合二重待ち状態のタスクに強制待ち解除要求[19]がだされると待ち状態になります。なお、強制待ち状態にあるタスクが自分自身を待ち状態にする要求は出せないため、強制待ち状態から二重待ち状態への移行は発生しません。

6. 休止状態 (DORMANT 状態)

通常は、MR32R システムに登録されているが起動していない状態です。この状態になるには以下の 2 つの場合があります。

- タスクが起動をかけられるのを待っている場合
- タスクが正常終了[20]もしくは強制終了[21]により終了した場合

7. 未登録状態(NON-EXISTENT 状態)

タスクが生成される前、またはタスクが削除された後の MR32R システムに登録されていない状態です。タスクを生成[22]すると MR32R に登録され、休止状態に移行します。この状態になるには以下の 2 つの場合があります。

- 他タスクにより、休止状態のタスクが削除された場合[23]
- 実行状態のタスクが自タスクを終了し、さらに削除した場合[24]

[19] rsm_tsk, irsm_tsk システムコール
[20] ext_tsk システムコール
[21] ter_tsk システムコール
[22] cre_tsk システムコール
[23] del_tsk システムコール
[24] exd_tsk システムコール

3.3.2 タスクの優先度とレディキュー

リアルタイム OS では実行したいタスクが同時にいくつも発生することがあります。

このときにどのタスクを実行するかを判断することが必要になります。そこでタスクに実行の優先度をつけ、優先度の高いタスクから実行するようにします。すなわち、処理を素早くおこなう必要のあるタスクの優先度を高くしておけば実行したいときに素早く実行することができるようになります。

MR32R では同一の優先度を複数のタスクに与えることができます。そこで、実行可能になったタスクの実行順を制御するためにタスクの待ち行列（レディキュー)を生成します。

図 3.14[25]にレディキューの構造を示します。レディキューは優先度ごとに管理され、タスクが接続されている最も優先度の高い待ち行列の先頭タスクを実行状態にします。[26]

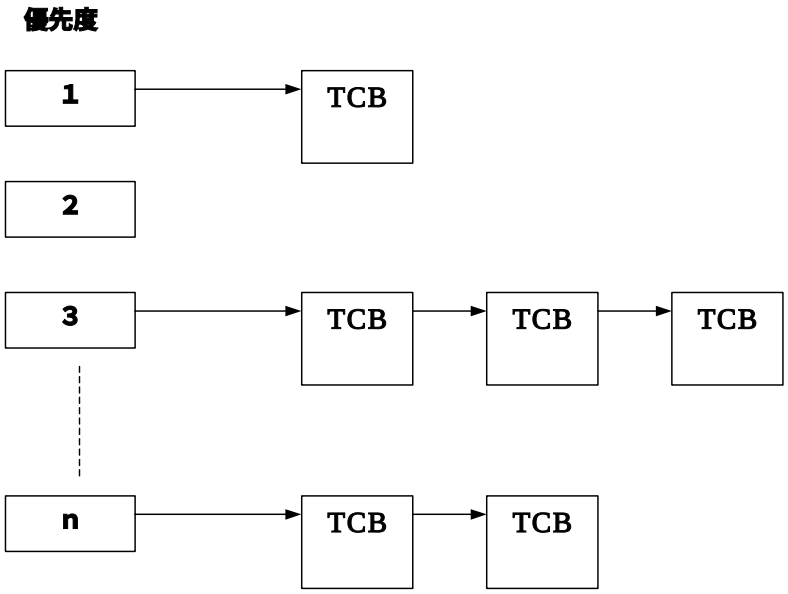


図 3.14 レディーキュー（実行待ち状態）

[25] TCB:タスクコントロールブロックについては次節で述べます。
[26] 実行状態のタスクはレディキューにつながれたままです。

3.3.3 タスクコントロールブロック (TCB)

タスクコントロールブロック(TCB)とは、リアルタイム OS がそれぞれのタスクの状態や優先度などを管理するデータブロックのことを言います。

MR32R ではタスクの以下の情報をタスクコントロールブロックとして管理しています。

- タスク接続ポインタ
  レディキューなどを構成するときに使用するタスク接続用ポインタ
- タスクの状態
- タスクの優先度
- タスク属性
  タスクのスタック領域が内蔵 RAM あるいは外部 RAM かの情報をこの領域に格納します。
- タスクの拡張情報
  タスク生成時に設定する、タスクの拡張情報がこの領域に格納されます。
- タスクのレジスタ情報など[27]を格納したスタック領域のポインタ (現在の SP レジスタの値)
- 起床要求カウンタ
  タスクの起床要求カウンタを蓄積する領域
- メモリブロックサイズ
  タスクが可変長メモリブロック待ち状態にある場合、この領域に要求するメモリブロックサイズを格納します。
- ランデブ待ちビットパターン
  ランデブ呼出である場合に、呼出側選択条件ビットパターンを、また、受付待ち状態である場合に、受付選択条件ビットパターンがこの領域に格納されます。
- フラグ待ちモード
  イベントフラグ待ちの時の待ちモード
- フラグ待ちパターン
  タスクがイベントフラグ待ち状態の場合、フラグ待ちパターンが格納されます。
- タイマキュー接続ポインタ
  タイムアウト機能を使用した場合に使用する領域です。タイマキューを構成する時に使用するタスクの接続用ポインタを格納する領域です。
- タイムアウトカウンタ
  タスクがタイムアウト待ち状態である場合に、残りの待ち時間が格納されます。
- 例外マスク
  例外マスク値を格納します。強制例外ハンドラを使用しない場合[28]、この領域は確保されません。

タスクコントロールブロックを図 3.15に示します。

---

[27] これをタスクコンテキストと呼びます。

[28] 強制例外ハンドラを使用、未使用の指定はコンフィグレーションファイルの system 定義で指定します。

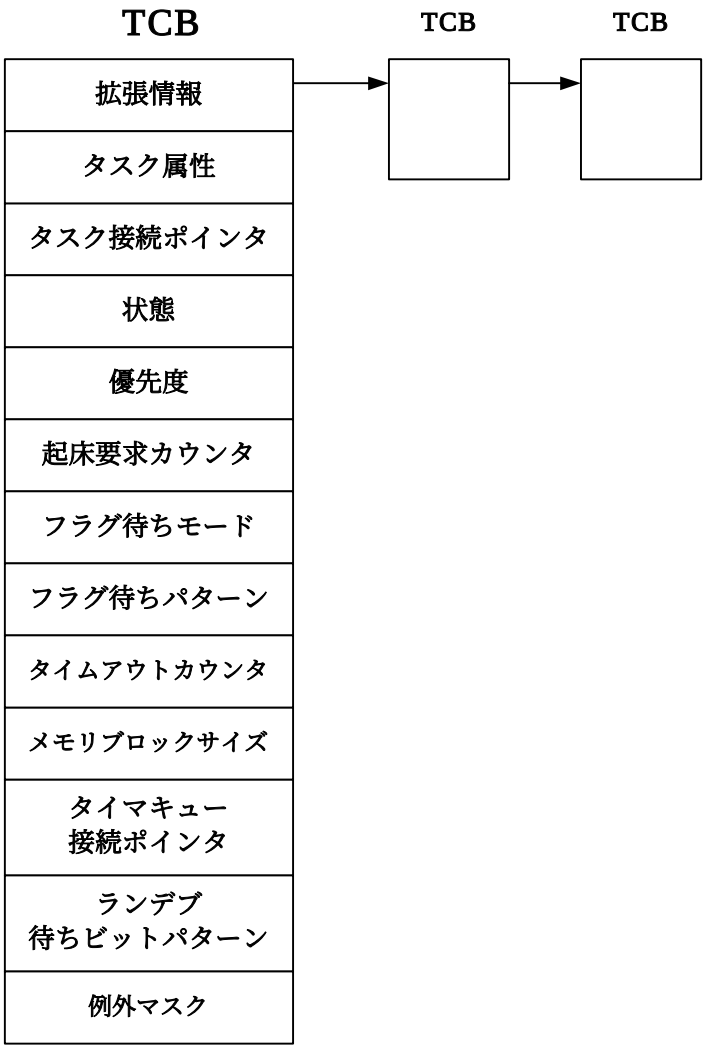


図 3.15 タスクコントロールブロック

3.4 ハンドラ

3.4.1 タスクとハンドラの違い

タスクは MR32R が実行制御するプログラムの単位ですから、タスクはそれぞれ独立したコンテキスト(プログラムカウンタ、スタックポインタ、ステータスレジスタ、その他のレジスタ)を持っています。したがって、あるタスクからあるタスクに実行を移す[29]にはこのコンテキストを切り替える必要があります。この処理には時間を要します。

割り込みなどの処理は、高速に応答する必要がありますので MR32R にはコンテキストを切り替えずに処理する機能があります。すなわち割り込まれたタスクのコンテキスト(レジスタ)をそのまま使用してプログラムを動作させることができます。このプログラムのことをハンドラと呼びます。ハンドラは割り込まれたタスクのコンテキスト(レジスタ)をそのまま使用しますので必ずハンドラの先頭で割り込まれたタスクのコンテキストをメモリに退避し、タスクに復帰するときにはその退避したコンテキストをもとに戻さなくてはなりません。

1. **割り込みハンドラ**

   ハードウェア割り込みにより起動されるプログラムを割り込みハンドラと呼びます。割り込みハンドラの起動は、割り込み入り口処理を行った後に、割り込み処理を行います。割り込み入り口処理は、MR32R カーネル内で行いますので、ユーザは、コンフィグレーションファイルに登録するだけでかまいません。割り込み入り口処理については、5.6.1 を参照してください。

2. **周期起動ハンドラ**

   周期起動ハンドラはあらかじめ設定された時間毎に周期的に起動されるプログラムです。設定された周期起動ハンドラを無効にするか有効にするかは周期起動ハンドラの活性状態の変更[30]によりおこないます。

3. **アラームハンドラ**

   アラームハンドラは、 指定した時刻に起動されるハンドラです。なお、set_tim システムコールにより、すでに実行されたアラームハンドラが再び起動されることはありません。また、まだ起動されていないアラームハンドラの起動時刻よりも後の時刻を設定した場合は、すべてのアラームハンドラは、起動されなくなります。

周期起動ハンドラとアラームハンドラはシステムクロック割り込み(タイマ割り込み)ハンドラからサブルーチンコールで呼び出されます(図 3.16参照)。したがって、周期起動ハンドラ、アラームハンドラはシステムクロック割り込みハンドラの一部として動作します。なお、周期起動ハンドラ、アラームハンドラが呼び出されるときは、システムクロック割り込みの割り込み優先レベルの状態で実行されます。

[29] このことをディスパッチもしくはタスク切り替えと呼びます。
[30] act_cyc システムコール

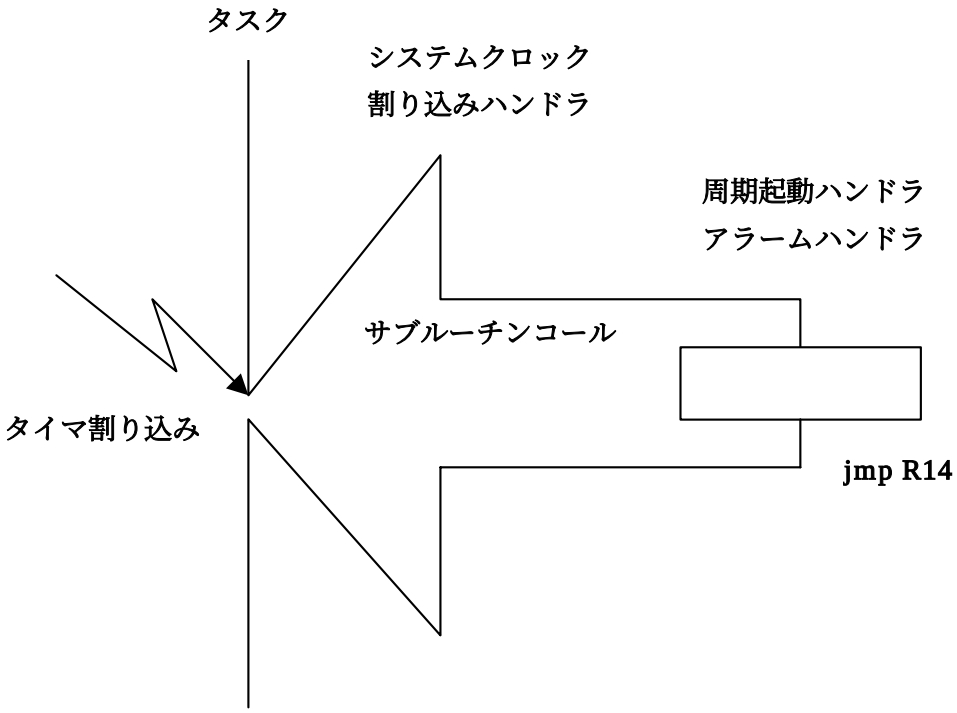


図 3.16 周期起動ハンドラ、アラームハンドラの起動

3.4.2 ハンドラ専用のシステムコール

MR32R では以下に示すシステムコールはハンドラからのみ発行できます。

なお、ret_int システムコールは、割り込みハンドラ専用です。したがって周期起動ハンドラ、アラームハンドラからは発行できません。

表 3-1 ハンドラから発行できるシステムコール

システムコール名		機能
ichg_pri	Change Task Priority	タスクの優先度を変更する
irot_rdq	Rotate Ready Queue	タスクのレディキューを回転する
irel_wai	Release Task Wait	タスクの待ち状態を強制解除する
isus_tsk	Suspend Task	タスクを強制待ち状態へ移行する
irsm_tsk	Resume Task	強制待ち状態のタスクを再開する
iwup_tsk	Wakeup Task	待ち状態のタスクを起床する
iset_flg	Set EventFlag	イベントフラグをセットする
isig_sem	Signal Semaphore	セマフォに対する信号操作
isnd_msg	Send Message to Mailbox	メッセージを送信する
ista_tsk	Start Task	タスクを起動する
irel_blf	Release Fixed-size memory block	固定長メモリーブロックを解放する
ret_int	Return from Interrupt Handler	割込みハンドラから復帰する
visnd_mbx	Send Message	メッセージを送信する(優先度付き)

3.5 MR32R カーネルの構成

3.5.1 モジュール構成

MR32R カーネルは、図 3.17に示すモジュールから構成されています。これらの個々のモジュールはそれぞれのモジュールの機能を実現する関数群より構成されています。

MR32R カーネルはライブラリ形式で提供されシステム生成時に必要な機能のみがリンクされます。すなわちこれらのモジュールを構成する関数群の中で使用している関数のみをリンケージエディタの機能によりリンクします。ただし、スケジューラとタスク管理の一部および時間管理の一部は必須機能関数ですので常時リンクされます。

アプリケーションプログラムはユーザーが作成するプログラムで、タスク・割り込みハンドラ・アラームハンドラおよび周期起動ハンドラ[31]から構成されます。

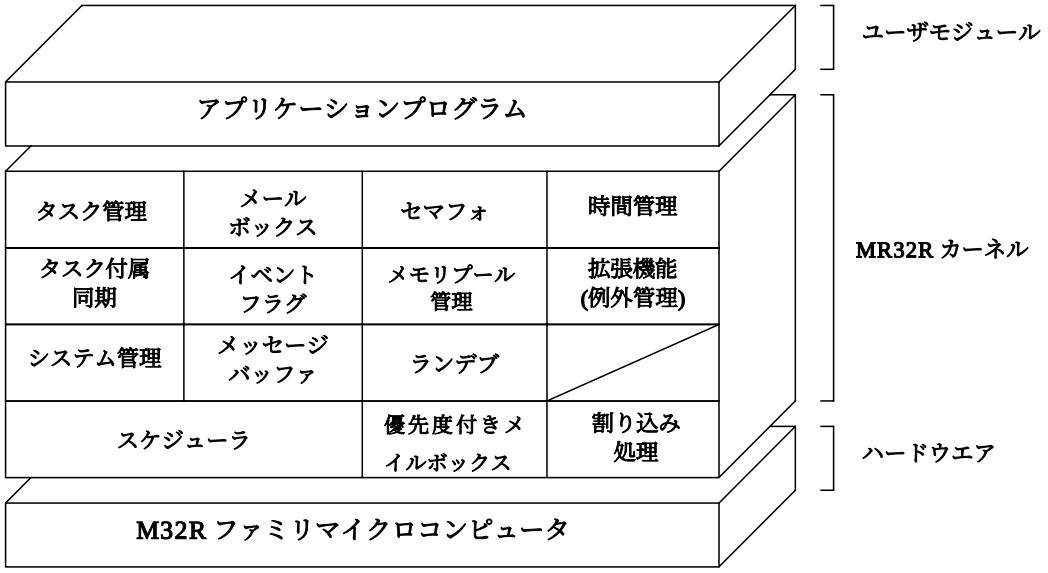


図 3.17 MR32R の構成

[31] 詳細は第3.5.12節を参照。

3.5.2 モジュール概要

MR32R カーネルを構成する各モジュールの概要を説明します。

- スケジューラ
  タスクの持つ優先度に基づいて、タスクの処理待ち行列を形成し、その待ち行列の先頭にある優先度の高い(優先度の値の小さい)タスクの処理を実行するよう制御をおこないます。

- タスク管理
  実行・実行可能・待ち・強制待ち等のタスク状態の管理をおこないます。

- タスク付属同期
  他タスクからタスクの状態を変化させることによりタスク間の同期をとります。

- 割り込み管理
  割り込みハンドラからの復帰処理をおこないます。

- 時間管理
  MR32R カーネルで使用するシステムクロックの設定、タイムアウトの処理、ユーザーの作成したアラームハンドラ[32]、周期起動ハンドラ[33]の起動をおこないます。

- システム管理
  MR32R カーネルのバージョン番号やシステムの動的な情報を報告します。

- 同期・通信
  タスク間の同期をとったりタスク間の通信をおこなうための機能です。以下の 3 つの機能モジュールが用意されています。
  - ♦ イベントフラグ
    MR32R 内部で管理されているフラグが立っているか否かによりタスクを実行するかしないかを制御します。これによりタスク間の同期をとることができます。
  - ♦ セマフォ
    MR32R 内部で管理されているセマフォカウンタ値によりタスクを実行するかしないかを制御します。これによりタスク間の同期をとることができます。
  - ♦ メイルボックス
    タスク間のデータ通信をデータの先頭アドレスを渡すことにより行います。

- 拡張同期・通信
  タスク間の同期をとったりタスク間の通信をおこなうための機能です。以下の 2 つの機能モジュールが用意されています。
  - ♦ メッセージバッファ
    タスク間の同期と通信を同時に行う機能で、メイルボックスとは異なり、可変長のメッセージ内容をコピーして送受信する機能です。
  - ♦ ランデブ
    タスク間の待ち合わせを行う機能です。

- メモリプール管理機能
  タスクまたはハンドラが使用するメモリ領域の動的な獲得および解放を行います。

---

[32] 指定時刻に一回のみ起動されるハンドラです。
[33] 周期的に起動されるハンドラです。

- 拡張機能
  強制例外機能、OS 資源のリセット機能を提供します。

- 拡張機能(優先度付きメイルボックス)
  優先度順のメッセージの送受信などの機能を提供します。

3.5.3 タスク管理機能

タスク管理機能とは、タスクの起動・終了・優先度の変更等のタスク操作をおこなう機能です。
MR32R カーネルが提供するタスク管理機能のシステムコールには、次のものがあります。

- タスクを生成する（cre_tsk)
  あるタスクから指定された ID のタスクを生成します。

- タスクを削除する（del_tsk)
  あるタスクから指定された ID のタスクを削除します。

- 自タスクを終了させ、削除する（exd_tsk)
  自タスクを正常終了させ、さらに自タスクを削除します。すなわち、自タスクを未登録状態に移行し、スタック領域を解放します。

- タスクを起動する（sta_tsk)
  あるタスクから、他タスクを起動することにより、起動対象となるタスクの状態を休止状態から実行可能状態もしくは実行状態に移行します。

- ハンドラからタスクを起動する（ista_tsk)
  ハンドラからタスクを起動することにより、起動対象となるタスクの状態を休止状態から実行可能状態に移行します。

- 自タスクを終了する（ext_tsk)
  自タスクを終了するとタスクの状態が休止状態になります。これにより再起動されるまで、このタスクは実行しません。

- 他タスクを強制的に終了させる（ter_tsk)
  休止状態以外の他のタスクを強制的に終了させ休止状態にします。
  タスクを強制的に終了させた後に再度そのタスクを起動するとそのタスクがリセットしたように振る舞います。 (図 3.18参照)

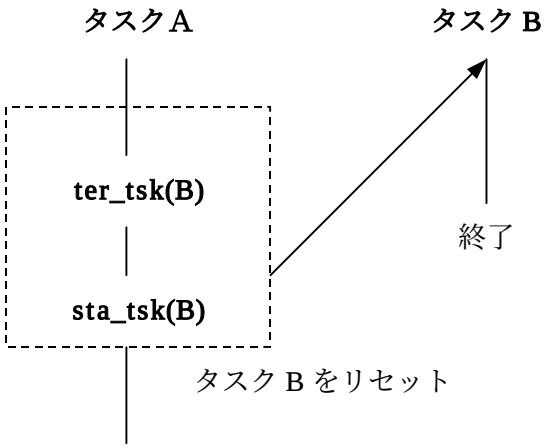


図 3.18 タスクのリセット

- タスクの優先度を変更する（chg_pri，ichg_pri）
  タスクの優先度を変更するとそのタスクが実行可能状態もしくは実行状態であるときは、レディキューも更新されます。（図 3.19参照）

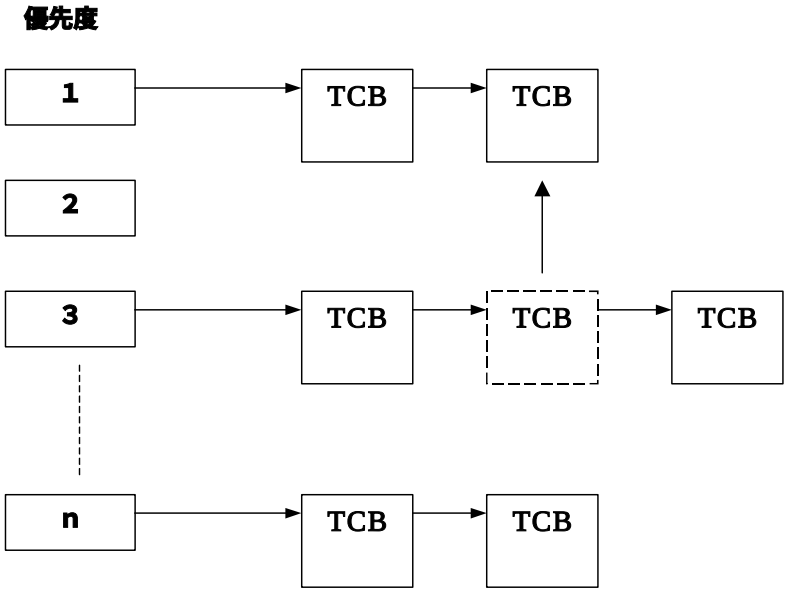


図 3.19 優先度の変更

- タスクの実行待ち行列を回転する（rot_rdq，irot_rdq）
  本システムコールにより TSS(タイムシェアリングシステム）を実現することができます。すなわち、一定周期でレディキューを回転すれば、TSS で必要なラウンドロビンスケジューリングを実現することができます。（図 3.20参照）

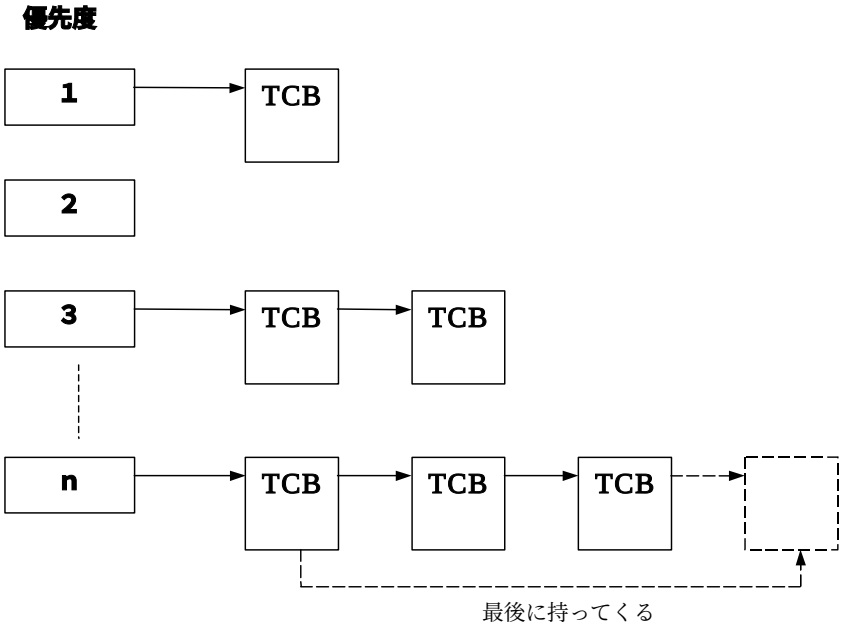


図 3.20 rot_rdq システムコールによるレディキューの操作

- タスクの待ち状態を強制解除する (rel_wai, irel_wai)
  タスクの待ち状態を強制的に解除します。解除される待ち状態は以下の条件により待ちに入ったタスクです。
  - タイムアウト待ち状態
  - slp_tsk システムコールによる (+タイムアウト有)待ち状態
  - イベントフラグ (+タイムアウト有)待ち状態
  - セマフォ (+タイムアウト有)待ち状態
  - メッセージ (+タイムアウト有)待ち状態
  - 送信 (+タイムアウト有)待ち状態(メッセージバッファ)
  - 受信 (+タイムアウト有)待ち状態(メッセージバッファ)
  - 呼出 (+タイムアウト有)待ち状態(ランデブ)
  - 受付 (+タイムアウト有)待ち状態(ランデブ)
  - メモリブロック (+タイムアウト有)待ち状態(固定長、可変長)

- 自タスクの ID を得る (get_tid)
  自タスクの ID 番号を得ます。ハンドラから発行した場合は、ID 番号の替わりに 0(ゼロ)が得られます。

- タスクの状態を参照する (ref_tsk)
  対象タスクの状態およびその優先度等を参照します。

3.5.4 タスク付属同期機能

タスク付属同期機能とは、タスク間の同期をとるためにタスクを待ち状態（もしくは強制待ち状態・二重待ち状態)にしたり、待ち状態になったタスクを起床させたりする機能です。

MR32R カーネルが提供するタスク付属同期システムコールには次のものがあります。

- タスクを強制待ち状態に移行する（sus_tsk，isus_tsk)
- 強制待ち状態のタスクを再開する（rsm_tsk，irsm_tsk)

  タスクの実行を強制的に待たせたり、実行を再開したりします。実行可能状態のタスクを強制待ちすれば強制待ち状態になり、待ち状態のタスクを強制待ちすれば二重待ち状態になります。（図 3.21参照)

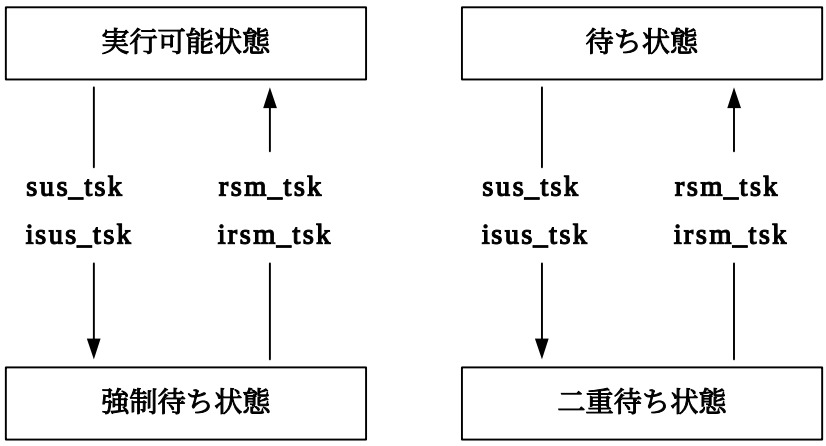


図 3.21 タスクの強制待ちと再開

- タスクを待ち状態に移行する（slp_tsk,tslp_tsk)
- 待ち状態のタスクを起床する（wup_tsk，iwup_tsk)

  slp_tsk、tslp_tsk システムコールにより待ち状態に入ったタスクを起床させます。

  slp_tsk、tslp_tsk システムコール以外の条件で待ち状態にあるタスクは起床できません。

  slp_tsk、tslp_tsk システムコール以外の条件で待ちに入ったタスクや休止状態を除く他の状態[34]のタスクに対して wup_tsk、iwup_tsk システムコールにより起床要求をおこなうと、この起床要求だけが蓄積されます。

  したがって、例えば実行状態のタスクに対して起床要求をおこなうと、この起床要求が一時的に記憶されます。そして、その実行状態のタスクが slp_tsk、tslp_tsk システムコールにより待ち状態に入ろうとした時、蓄積された起床要求が有効になり、待ち状態にならずに再び実行を続けます。（図 3.22参照)
- タスクの起床要求を無効にする（can_wup)

  蓄積された起床要求をクリアします。（図 3.23参照)

[34] 待ち状態であっても以下の条件で待っているタスクは起床されませんのでご注意ください。

- ♦ イベントフラグ待ち状態、セマフォ待ち状態、メッセージ待ち状態、タイムアウト待ち状態、メッセージバッファ送信待ち、メッセージバッファ受信待ち、固定長メモリプール獲得待ち、可変長メモリプール獲得待ち、ランデブ受付待ち、ランデブ呼び出し待ち、ランデブ終了待ち

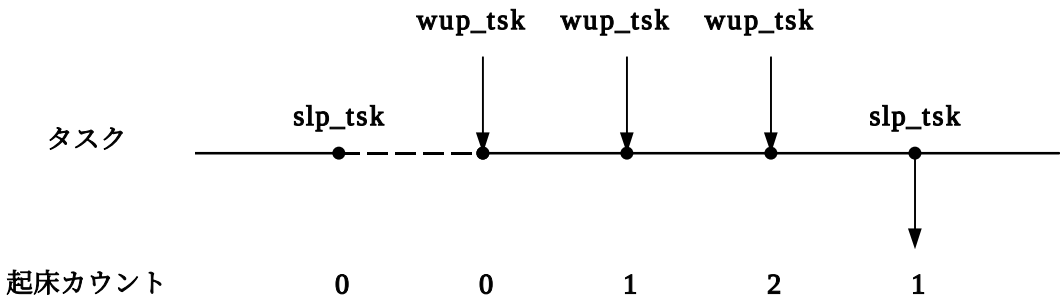


図 3.22 起床要求の蓄積

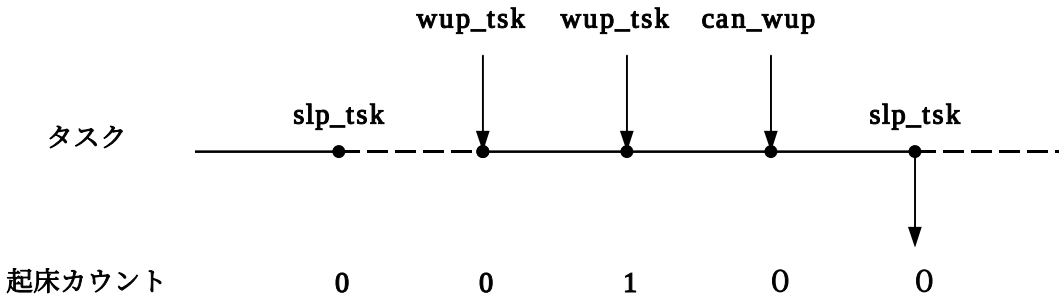


図 3.23 起床要求のキャンセル

3.5.5 同期・通信機能 (イベントフラグ)

イベントフラグは複数タスクの実行同期をとるための MR32R 内部に持つ機構です。イベントフラグは、フラグ待ちパターンと 32 ビットのイベントフラグ(ビットパターン)によりタスクの実行制御をおこないます。タスクは、設定したフラグ待ちの条件が満たされるまで待ちます。

MR32R カーネルが提供するイベントフラグのシステムコールには次のものがあります。

- イベントフラグを生成する (cre_flg)
  あるタスクから指定された ID のイベントフラグを生成します。

- イベントフラグを削除する (del_flg)
  あるタスクから指定された ID のイベントフラグを削除します。

- イベントフラグをセットする (set_flg, iset_flg)
  イベントフラグをセットします。これにより、このイベントフラグの待ちパターンを待っていたタスクは待ち解除されます。

- イベントフラグをクリアする (clr_flg)
  イベントフラグをクリアします。

- イベントフラグを待つ (wai_flg, twai_flg)
  イベントフラグがあるパターンにセットされるのを待ちます。イベントフラグを待つ時のモードは、 以下に示す 3 種類があります。
  - ♦ AND 待ち
    指定されたビットが全てセットされるのを待ちます。
  - ♦ OR 待ち
    指定されたビットの内いずれか 1 ビットがセットされるのを待ちます。
  - ♦ クリア指定
    AND 待ちか OR 待ちの条件が満たされた場合にフラグをクリアします。

- イベントフラグを得る (pol_flg)
  イベントフラグがあるパターンになっているか否かを調べます。このシステムコールではタスクは待ち状態に移行しません。

- イベントフラグの状態を得る (ref_flg)
  対象イベントフラグのビットパターンや待ちタスクの有無を参照します。

図 3.24に wai_flg と set_flg システムコールを使用したイベントフラグによるタスクの実行制御の例を示します。

イベントフラグは複数のタスクを一度に起床できるという特徴があります。

図 3.24では、タスク A からタスク F までの 6 個のタスクがつながっています。

そして、set_flg システムコールによって、フラグパターンを 0x0F にすると、待ち条件にあっているタスクがキューの前から順にはずされていきます。この図で待ち条件を満たすタスクはタスク A、タスク C、タスク E、タスク F です。このうち、タスク A、タスク C、タスク E はキューからはずされますが、タスク E は、クリア指定で待っていますので、タスク E がキューからはずされた時点でフラグがクリアされます。したがって、タスク F はキューからはずされません。

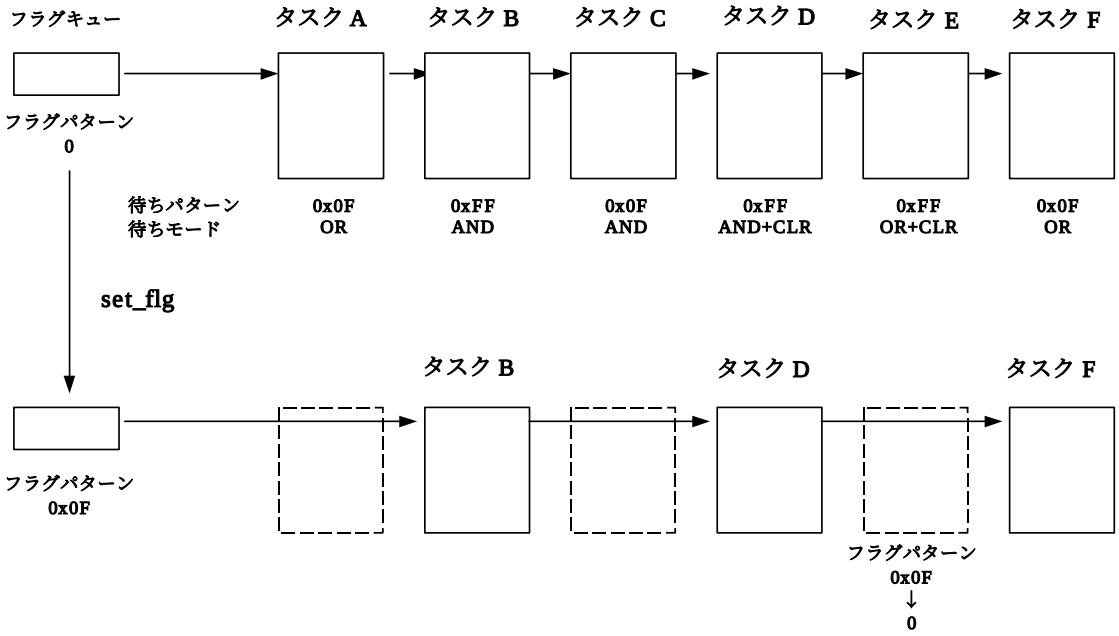


図 3.24 イベントフラグによるタスクの実行制御

3.5.6 同期・通信機能 (セマフォ)

セマフォは複数のタスクで共有する装置などの資源の競合を防ぐための機能です。例えば図 3.25に示すような場合、すなわち通信回線が 3 本しかないシステムに 4 つのタスクが回線を獲得しようと競合した場合に、通信回線を競合することなくタスクに接続することがセマフォを用いるとできます。

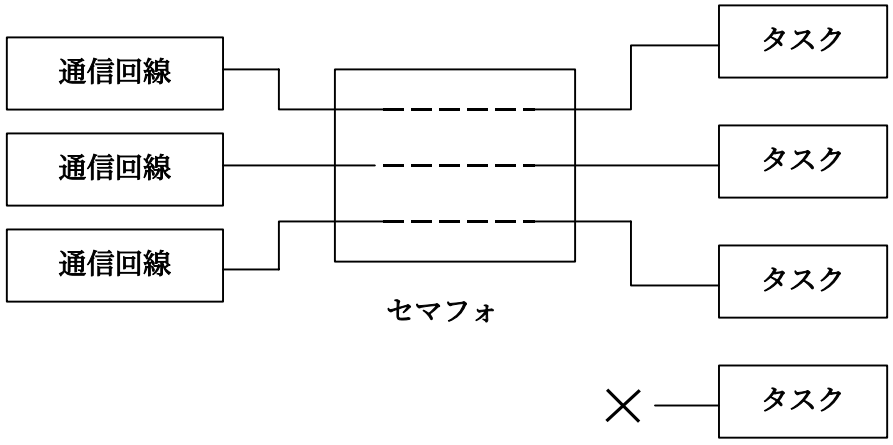


図 3.25 セマフォによる排他制御

セマフォは内部にセマフォカウンタと呼ばれる計数値を持っており、そのセマフォカウンタに基づきセマフォの獲得・解放をおこなうことによって資源の競合を防ぎます。 (図 3.26参照)

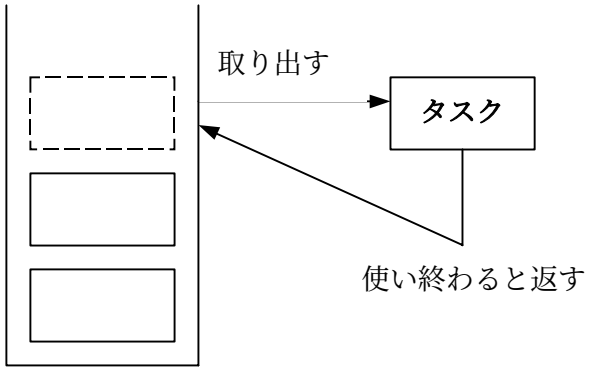


図 3.26 セマフォカウンタ

MR32R カーネルが提供するセマフォ同期のシステムコールには次のものがあります。

- セマフォを生成する (cre_sem)
  あるタスクから指定された ID のセマフォを生成します。
- セマフォを削除する (del_sem)
  あるタスクから指定された ID のセマフォを削除します。

- セマフォに対する信号操作（sig_sem，isig_sem）
  セマフォに信号をおくります。すなわち、セマフォを待っているタスクがあればそのタスクを起床し、なければセマフォカウンタを１増やします。

- セマフォ獲得操作（wai_sem，twai_sem）
  セマフォを待ちます。セマフォカウンタが０であればセマフォを得ることができませんので待ち状態になります。

- セマフォ獲得操作（preq_sem）
  セマフォを得ます。得るべきセマフォがなければ待ち状態に入らずにエラーコードをかえします。

- セマフォの状態を参照する（ref_sem）
  対象セマフォの状態を参照します。対象セマフォのカウント値や待ちタスクの有無を参照します。

wai_sem、sig_sem システムコールを用いたタスクの実行制御の例を図 3.27に示します。

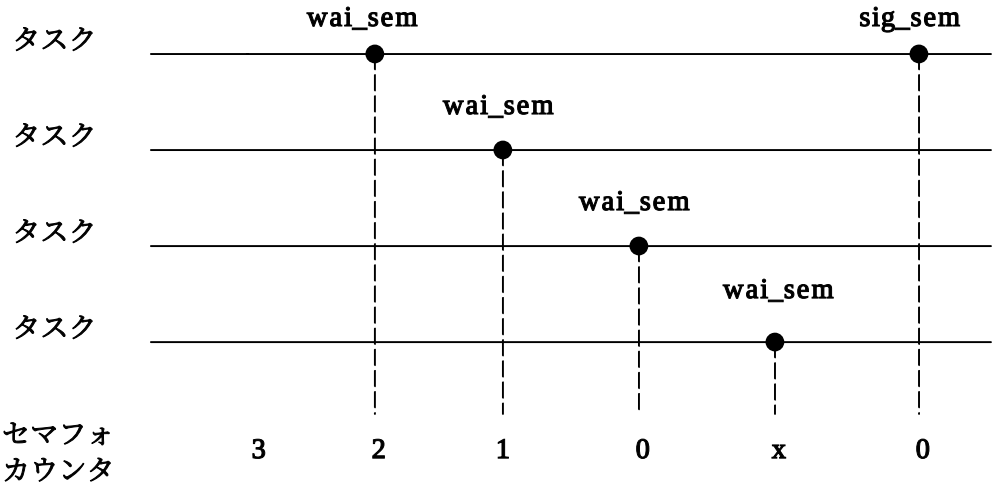


図 3.27 セマフォによるタスクの実行制御

3.5.7 同期・通信機能 (メイルボックス)

メイルボックスとはタスク間でデータの通信をおこなう機構です。例えば、図 3.28においてタスクAがメッセージをメイルボックスに投函しタスクBがそのメッセージをメイルボックスから取り出すことができます。

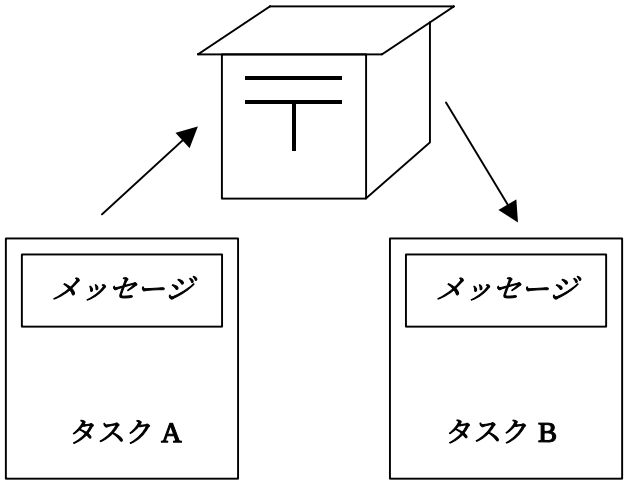


図 3.28 メイルボックス

このメイルボックスに投函できるメッセージ幅は 32 ビットのデータです[35]。

MR32R では、このデータをメッセージパケットの先頭アドレスとして使用することを標準としていますが[36]、単なるデータとして使用することも可能です[37]。

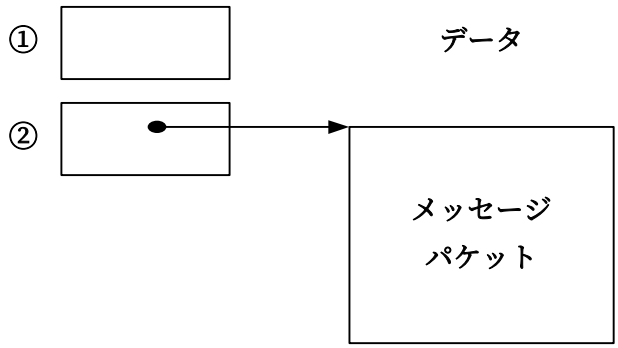


図 3.29 メッセージの意味

メイルボックスにはメッセージを蓄積する機能があります。蓄積されたメッセージは FIFO[38]でメッセージが取り出されます。ただし、メイルボックスに蓄積できるメッセージの数には制限があります。

このメイルボックスに蓄積できるメッセージの最大数をメッセージキューのサイズと呼びます。

[35] メッセージサイズの指定は、コンフィグレーションファイルのシステム定義で行います。
[36] ITRON 仕様では、メッセージパケットの先頭アドレスとして使用することを標準としています。
[37] この場合、システムコールの引数であるデータにキャストして、ポインタに型変換しなければなりません
[38] ファーストインファーストアウト

(図 3.30参照)

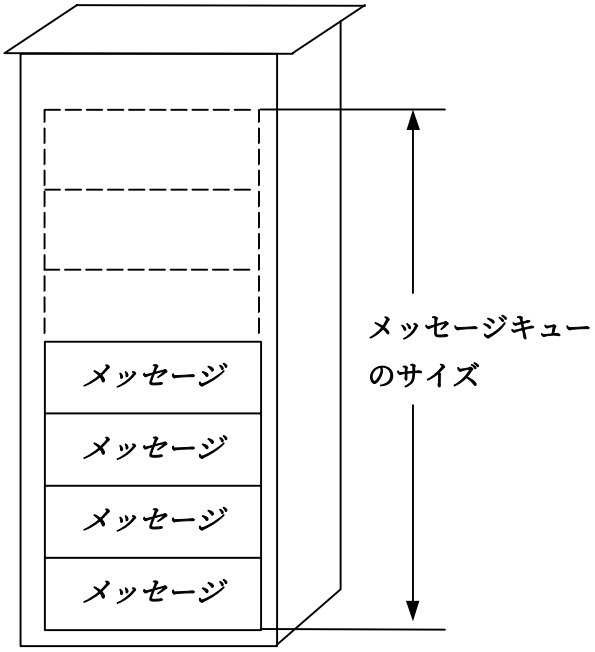


図 3.30 メッセージキューのサイズ

MR32R カーネルが提供するメイルボックスのシステムコールには次のものがあります。

- メイルボックスを生成します (cre_mbx)
  あるタスクから指定された ID のメイルボックスを生成します。

- メイルボックスを削除します (del_mbx)
  あるタスクから指定された ID のメイルボックスを削除します。

- メッセージを送信します (snd_msg, isnd_msg)
  メッセージを送信します。すなわち、メッセージをメイルボックスに投函します。

- メッセージを受信します (rcv_msg, trcv_msg)
  メッセージを受信します。すなわち、メッセージをメイルボックスから取り出します。このときメッセージがメイルボックスに投函されていなければ、投函されるまで待ち状態になります。

- メッセージを受信します (prcv_msg)
  メッセージを受信します。rcv_msg システムコールと異なるのは、メイルボックスにメッセージがない場合は待ち状態にならずにエラーコードを返すところです。

- メイルボックスの状態を参照します (ref_mbx)
  対象メイルボックスにメッセージが入るのを待っているタスクの有無やメイルボックスに入っている先頭のメッセージを参照します。

3.5.8 拡張同期・通信機能 (メッセージバッファ)

メッセージバッファは、メイルボックスの機能と同様にメッセージを渡すことにより同期と通信を同時に行う機能です。メイルボックス機能と異なる点は、メッセージ内容そのもののコピーが、送信相手のタスクに送信されます。

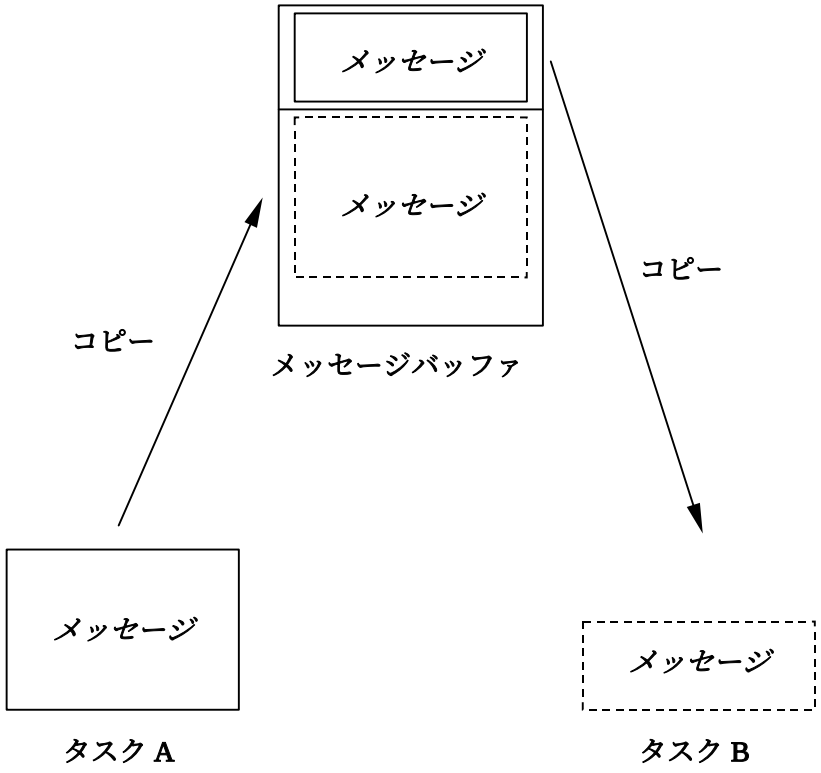


図 3.31 メッセージバッファ

図 3.31が示すようにメッセージバッファには、メイルボックスと同様にメッセージを蓄積します。蓄積されたメッセージは、FIFO 順でメッセージが取り出されます。

メッセージバッファ機能を使用し、メッセージを送信する場合、MR32R はバッファにまず、4 バイトのメッセージのサイズ情報を書き込んだ後、メッセージを書き出すようになっています。直前に送信されたメッセージの終端が 4 バイト境界にない場合は、次の 4 バイト境界よりメッセージのサイズ情報を書き出します。

そのため、ひとつのメッセージを送信すると、「アライメント調整分+サイズ情報(4 バイト)+メッセージサイズ」分だけバッファを使用します。

なお、受信したメッセージにメッセージのサイズ情報は含まれません。

[例]
　メッセージ終端が 4 バイト境界にないメッセージ A を送信後、12 バイトのメッセージ B を送信した場合、次の 4 バイト境界にサイズ情報が書き込まれた後、メッセージ B が書き込まれます。

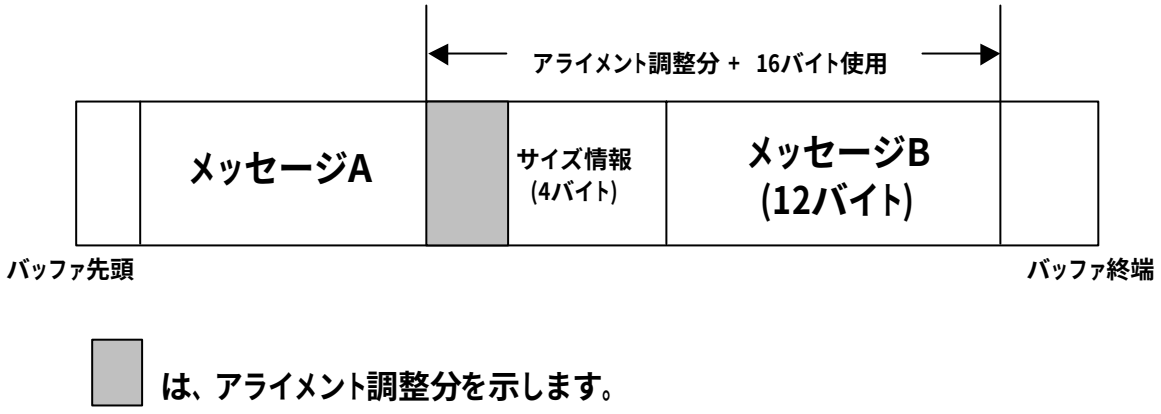


図 3.32 メッセージの送信例

MR32R カーネルが提供するメッセージバッファのシステムコールには、次のものがあります。

- メッセージバッファを生成する（cre_mbf）
  あるタスクから指定された ID のメッセージバッファを生成します。

- メッセージバッファを削除する（del_mbf）
  あるタスクから指定された ID のメッセージバッファを削除します。

- メッセージを送信する（snd_mbf、tsnd_mbf）
  メッセージバッファにメッセージを送信します。すなわち、メッセージをメッセージバッファにコピーします。メッセージバッファでは、転送するメッセージのサイズは一定ではありません。メッセージバッファの空きサイズによっては、メッセージを転送できたり、できなくなる場合が出てきます。メッセージバッファの空きサイズが足りない場合は、送信待ち状態になります。(図 3.33参照)

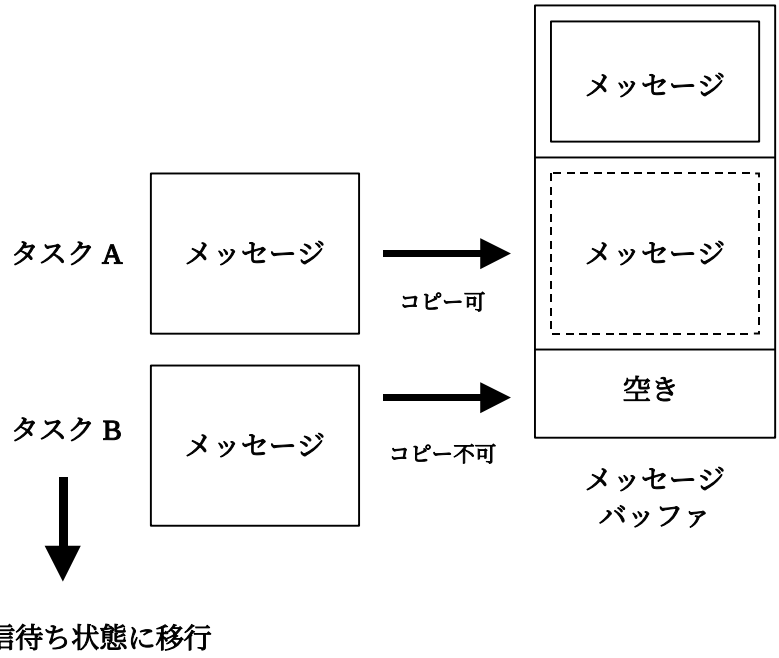


図 3.33 メッセージの送信

- メッセージを送信する（psnd_mbf）
  メッセージバッファにメッセージを送信します。snd_mbf、tsnd_mbf と異なるのは、メッセージバッファに空きサイズが足りない場合、送信待ち状態にはならず、エラーコードを返すところです[39]。

[39] エラーコードは、E_TMOUT を返します。

- メッセージを受信する (rcv_mbf、trcv_mbf)
  メッセージを受信します。メッセージをメッセージバッファから取り出します。メッセージバッファにメッセージがない場合は受信待ち状態に移行します。
  メッセージをメッセージバッファから受信すると、メッセージバッファの空きサイズが大きくなります。送信待ちタスクがある場合、その待ちタスクが送信するメッセージのサイズよりも空きサイズの方が大きい場合は、メッセージをメッセージバッファに送信し、送信待ち状態から実行(RUN)状態あるいは実行可能(READY)状態に移行します。

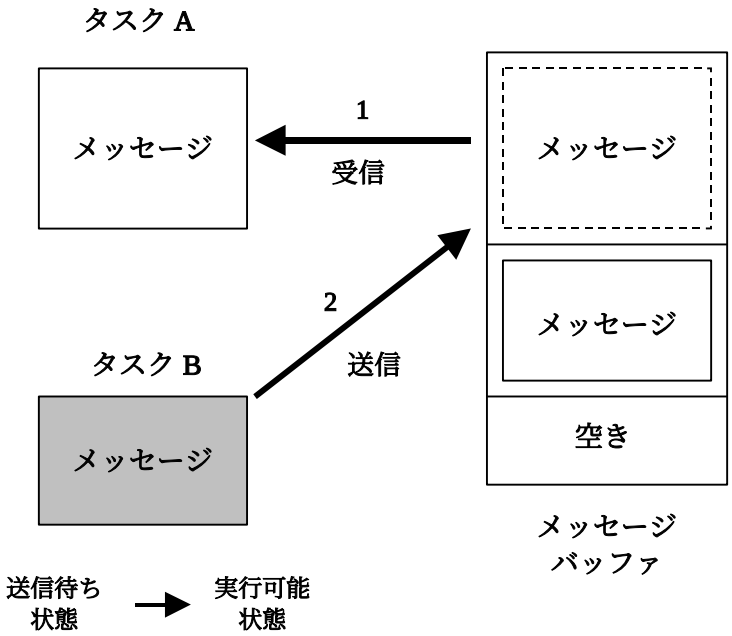


図 3.34 メッセージの受信

状態は、図 3.34では、タスク A が実行(RUN)状態、タスク B が送信待ち状態中で、

1. **タスク A がメッセージバッファからメッセージを受信します。**

   メッセージバッファからメッセージを受信することで、メッセージバッファの空き領域が大きくなります。

2. **タスク B は、メッセージバッファにメッセージを送信します。**

   メッセージバッファの空き領域が足りないため、送信待ち状態中であったタスク B は、タスク A のメッセージ受信により、メッセージバッファにメッセージを送信します。タスク B の状態は、送信待ち状態から実行可能状態に移行します。

- メッセージを受信する (prcv_mbf)
  メッセージを受信します。rcv_mbf、trcv_mbf と異なるところは、メッセージバッファにメッセージがない場合は、受信待ち状態に移行せずにエラーコードを返すところです[40]。

[40] エラーコード E_TMOUT を返します。

- メッセージバッファの状態を参照する（ref_mbf）
  対象メッセージバッファに対する送信待ちタスクの有無、受信待ちタスクの有無、次に受信するメッセージのサイズなどを参照します。

3.5.9 拡張同期・通信機能 (ランデブ)

ランデブ機能は、ポートと呼ばれる窓口を介してタスク(呼出)とタスク(受付)が待ち合わせを行い、待ち合わせ(ランデブ)が成立すると、互いのメッセージの交換を行う機能です。(待ち合わせ(ランデブ)の成立とは、呼出側のビットパターンと受付側のビットパターンとの論理積の結果が 0 以外であれば、ランデブ成立となり、待ち合わせが成立したことになります。)

本機能は、リアルタイム OS 上でのサーバ・クライアント形式の実現に有用です。

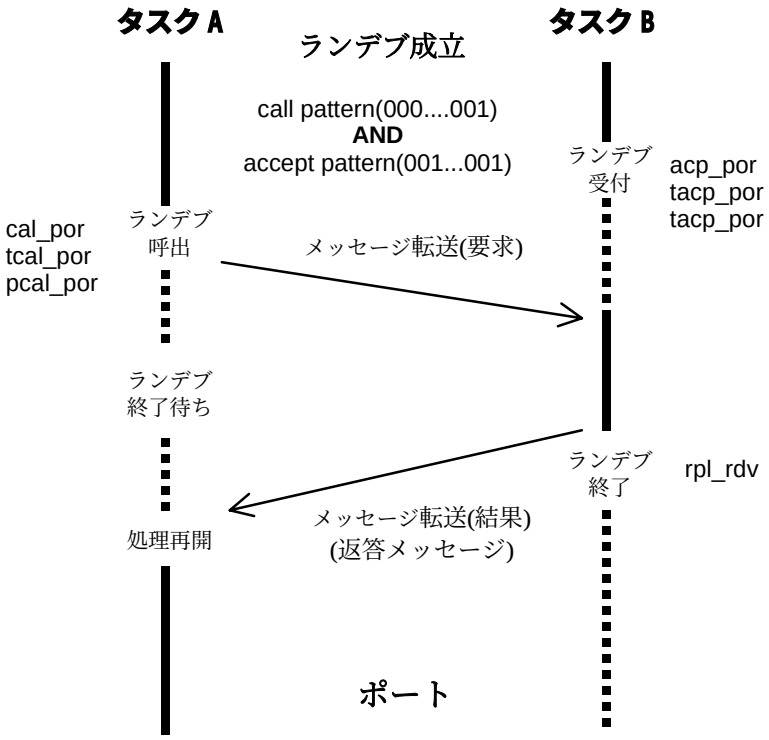


図 3.35 ランデブ

MR32R カーネルが提供するランデブのシステムコールには次のものがあります。

- ランデブ用ポートを生成します (cre_por)
  あるタスクから指定された ID のランデブ用ポートを生成します。

- ランデブ用ポートを削除します (del_por)
  あるタスクから指定された ID のランデブ用ポートを削除します。

- ポートに対するランデブの呼出 (cal_por、tcal_por)
  cal_por、tcal_por システムコールを発行し、指定したポートの受付待ち状態のタスクとランデブが成立するかどうかをチェックします。ランデブが成立すれば、受付待ち状態のタスクに対して、メッセージを転送し、受付待ち状態から実行可能状態に移行します。
  指定したポートに受付待ちタスクがない場合や受付待ちタスクがあってもランデブ成立条件が満たされない場合には、cal_por、tcal_por を発行したタスクは、呼出待ち状態となり、呼出待ち行列につながれます。

- ポートに対するランデブの呼び出し (pcal_por)
  cal_por、tcal_por システムコールと同様の処理を行いますが、指定したポートに受付待ちタスクがない場合や受付待ちタスクがあってもランデブ成立条件が満たされない場合は、呼出待ち状態には移行せず、エラーコードを返します。

- ポートに対するランデブの受付 (acp_por、tacp_por)
  acp_por、tacp_por システムコールを発行し、指定したポートの呼出待ち状態のタスクとランデブが成立するかどうかをチェックします。ランデブが成立すれば、呼出待ち状態であったタスクは、呼出待ち行列からはずれ、ランデブ終了待ち状態へ移行します。なお、ランデブ受付側タスク(acp_por を発行し他タスク)が、同時に複数のランデブを行うこともできます。図 3.36は、異なるポートでの複数のランデブを表わしています。また、同一ポートに対しても複数のランデブは行うことはできます。
  指定したポートに呼出待ちタスクがない場合や呼出待ちタスクがあってもランデブ成立条件が満たされない場合には、acp_por、tacp_por を発行したタスクは、受付待ち状態となり、受付待ち行列につながれます。

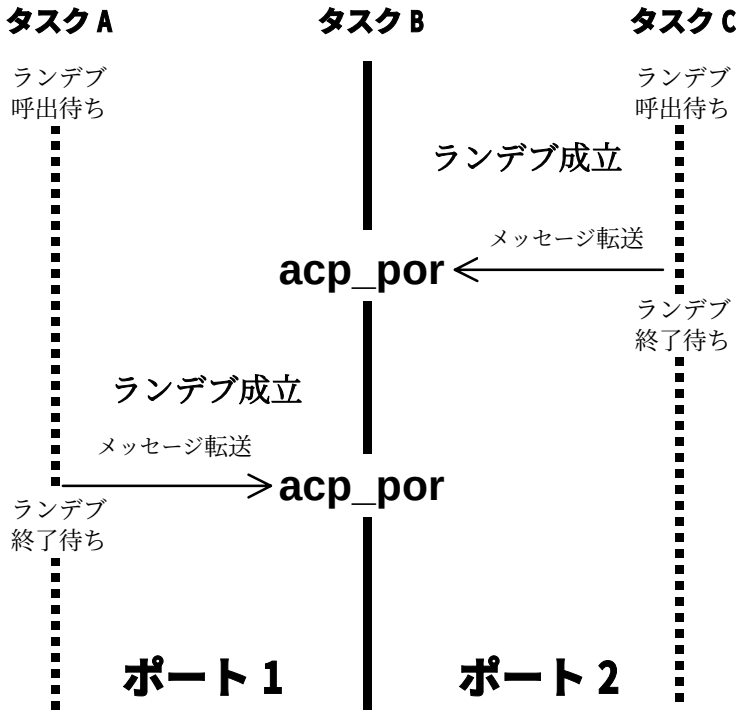


図 3.36 複数のランデブ

- ポートに対するランデブの受付 (pacp_por)
  acp_por、tacp_por システムコールと同様の処理を行いますが、指定したポートに呼出待ちタスクがない場合や呼出待ちタスクがあってもランデブ成立条件が満たされない場合は、受付待ち状態には移行せず、エラーコードを返します。

- ポートに対するランデブの回送 (fwd_por)
  本システムコール(fwd_por)を発行すると、現在、ランデブ中の呼出側タスクとのランデブ

状態を解除し、指定した別のポートに対してランデブ呼出を行います。(本システムコールは、acp_por、tacp_por あるいは pacp_por システムコールを実行した後の状態で発行しなければなりません。)この後の処理は、cal_por、tcal_por あるいは pcal_por システムコールの処理と同様の処理を行います。

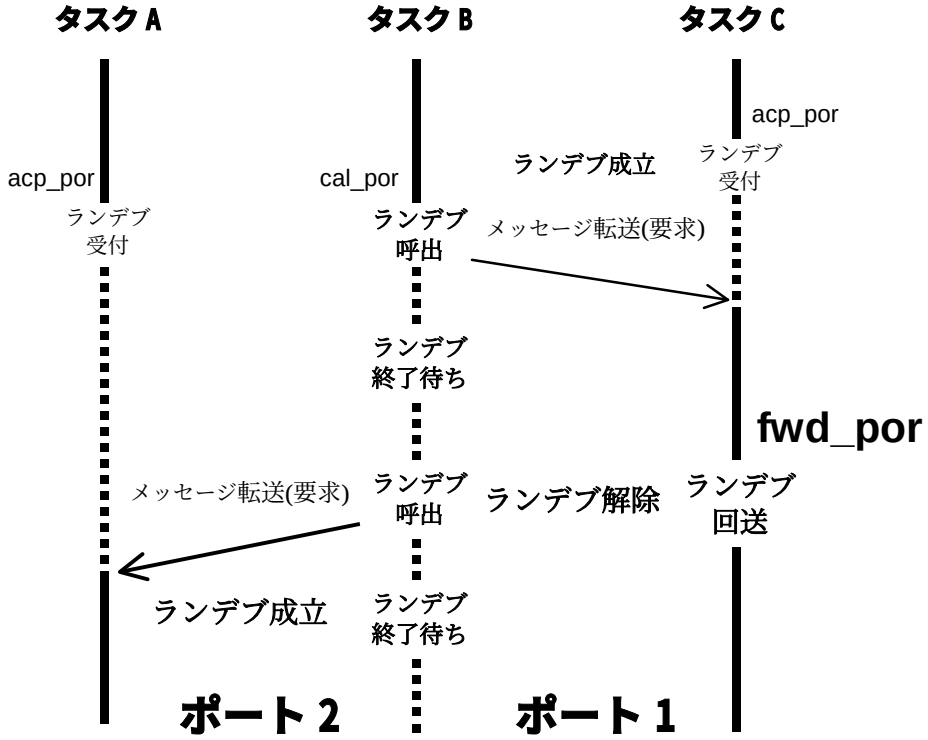


図 3.37 ランデブの回送

- ランデブの返答（rpl_rdv）
  本システムコール(rpl_rdv)を発行すると呼出側タスクに対して返答メッセージを返し、ランデブを終了します。本システムコールは、acp_por、tacp_por あるいは pacp_por システムコール発行後の状態でなければなりません。返答メッセージを受け取った呼出側タスクは、ランデブ終了待ち状態から実行可能状態に移行します。

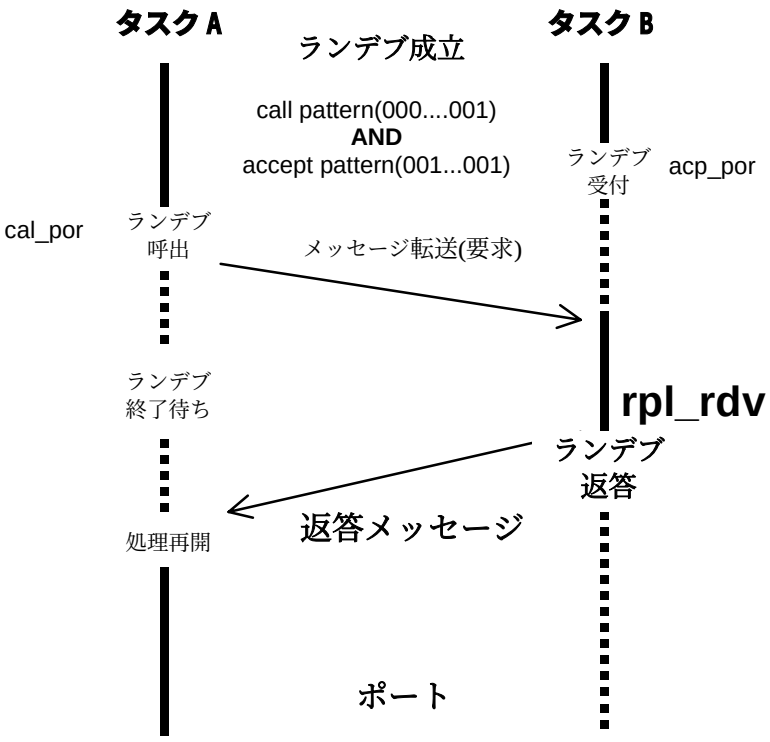


図 3.38 ランデブの返答

- ランデブ用ポートの参照（ref_por）
  指定した ID のランデブ用ポートに対する呼び出し待ちタスクの有無や受付待ちタスクの有無を参照します。

3.5.10 割り込み管理機能

割り込み管理機能は、外部割り込みの発生に対して、実時間で処理をおこなう機能を提供します。MR32R カーネルが提供する割り込み管理システムコールには次のものがあります。

- 割り込みハンドラを定義します (def_int)
  割り込みハンドラを指定された割り込みベクタに定義します。

- 割り込みハンドラから復帰します (ret_int)
  ret_int システムコールは、割り込みハンドラから復帰するとき、必要ならばスケジューラを起動し、タスク切り替えをおこないます。
  本機能は、ハンドラ関数の終了時に自動で呼び出されます。従って、この場合、本システムコールを呼び出す必要はありません。

- 割り込みおよびタスクのディスパッチを禁止します (loc_cpu)
  loc_cpu システムコールは、外部割り込みとタスクのディスパッチを禁止します。

- 割り込みおよびタスクのディスパッチを許可します (unl_cpu)
  unl_cpu システムコールは、外部割り込みとタスクのディスパッチを許可します。従って、loc_cpu システムコールによる割り込みおよびディスパッチの禁止状態が、このシステムコールの発行により解除されます。

図 3.39に割り込み処理の流れをしめします。なお、タスク選択からレジスタ復帰までの処理をスケジューラと呼びます。

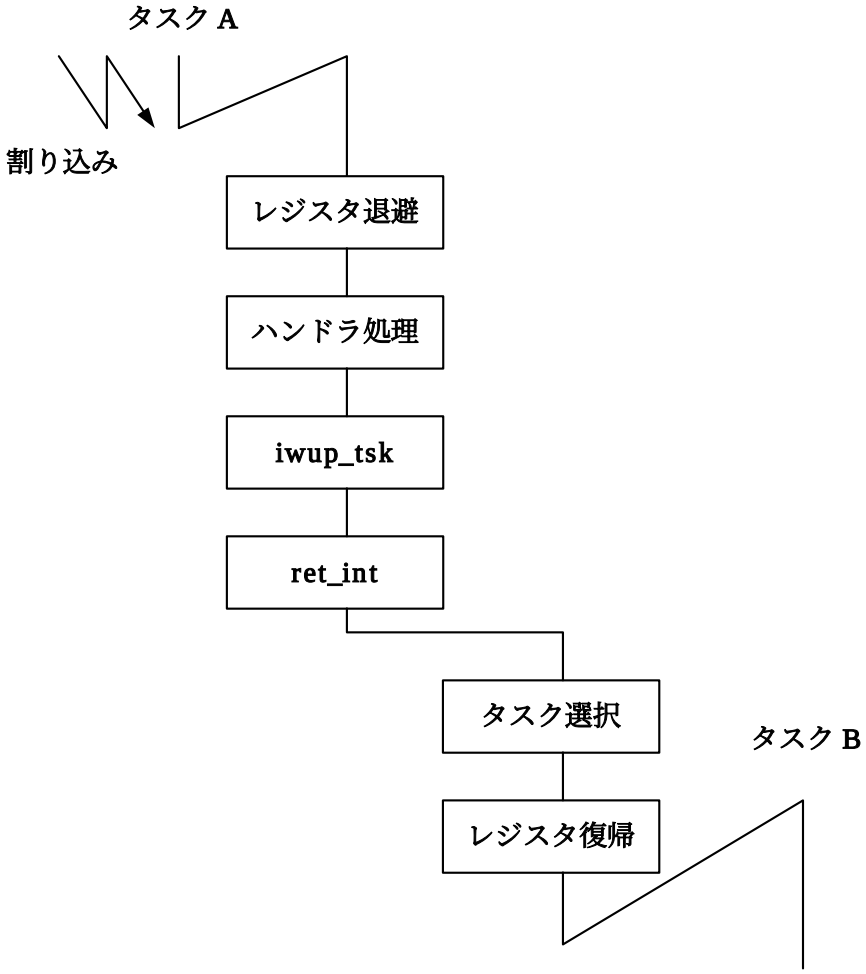


図 3.39 割り込み処理の流れ

3.5.11 メモリプール管理機能

メモリプール管理機能はシステムのメモリ空間(RAM 空間)を動的に管理する機能を提供します。

特定のメモリ領域(メモリプール)を管理し、そのメモリプールからタスクあるいはハンドラの必要とするメモリブロックを動的に確保し、また不用になったメモリブロックをメモリプールに解放します。

MR32R では、固定長と可変長の 2 つのメモリプール管理機能をサポートしています。

固定長メモリプール管理機能

メモリプールから獲得できるメモリブロックサイズが決まっていることを固定長といいます。

獲得するメモリブロックサイズは、cre_mpf システムコールあるいはコンフィグレーションファイルで指定します。

MR32R カーネルが提供する固定長メモリプール管理システムコールには次のものがあります。

- 固定長メモリプールを生成する (cre_mpf)
  あるタスクから指定された ID の固定長メモリプールを生成します。

- 固定長メモリプールを削除する (del_mpf)
  あるタスクから指定された ID の固定長メモリプールを生成します。

- メモリブロックを獲得する (get_blf、tget_blf)
  指定された ID の固定長メモリプールからメモリブロックを獲得します。空きメモリブロックが、指定された固定長メモリプールにない場合、このシステムコールを発行したタスクは待ち状態に移行し、待ち行列につながれます。

- メモリブロックを獲得する (pget_blf)
  指定された ID の固定長メモリプールからメモリブロックを獲得します。get_blf、tget_blf と異なるのは、空きメモリブロックがメモリプールにない場合は、待ち状態に移行せず、エラーコードを返します。

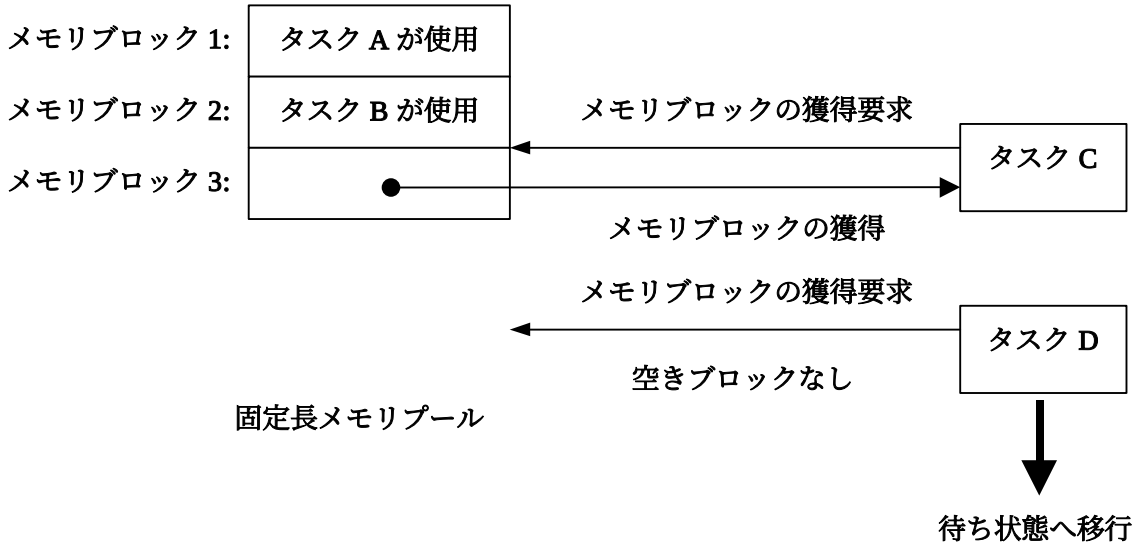


図 3.40 固定長メモリプールの獲得

- メモリブロックを解放する（rel_blf,irel_blf）
  獲得しているメモリブロックを解放します。指定された固定長メモリプールに対する待ち状態のタスクがある場合には、待ち行列の先頭につながれたタスクに解放したメモリブロックを割り当てます。この時のタスクの状態は、待ち状態から実行可能(READY)状態に移行します。また、待ち状態のタスクがない場合は、メモリプールにメモリブロックを返却します。

- メモリプールの状態を参照する（ref_mpf）
  対象メモリプールの空きブロック数やブロックサイズを参照します。

可変長メモリプール管理機能

メモリプールから獲得できるメモリブロックサイズが任意に指定可能なことを可変長といいます。
MR32R では、メモリを 4 種類の固定長ブロックサイズで管理しています。
4種類の各々のサイズは、ユーザが獲得するメモリブロックの最大サイズからMR32Rが計算します。メモリブロックの最大サイズは、cre_mpl システムコールあるいはコンフィグレーションファイルで指定します。

例

```
variable_memorypool[]{

        name            = mpl1;
        max_memsize     = 400; <---- 最大獲得サイズ
        heap_size       = 5000;

};
```

上記のように、可変長メモリプール定義を行った場合、4 種類の固定長ブロックサイズは、max_memsize 定義値から 60,120,240,480 となります。
ユーザが要求したメモリは、指定サイズをもとに MR32R が計算を行い 4 種類の固定長メモリブロックサイズの中から最適なサイズを選択し、メモリを割り当てます。この 4 種類以外のサイズのメモリブロックを割り当てることはありません。
4 種類のブロックサイズは(下記 abcd)は、下記の計算式から算出されます。

```
a = (((max_memsize + (12-1))/96)+1)*12
b = a*2
c = a*4
d = a*8
```

MR32R カーネルが提供する可変長メモリプール管理システムコールには次のものがあります。

- 可変長メモリプールを生成する（cre_mpl）
  あるタスクから指定された ID の可変長メモリプールを生成します。

- 可変長メモリプールを生成する（del_mpl）
  あるタスクから指定された ID の可変長メモリプールを削除します。

- メモリブロックを獲得する（get_blk、tget_blk）
  メモリブロックを可変長メモリプールから獲得します。ユーザが指定したブロックサイズは、4 種類のブロックサイズのうちから最適なブロックサイズに丸めて、丸めたサイズ分のメモリをメモリプールから獲得します。指定されたサイズのメモリブロックがメモリプールにない場合、このシステムコールを発行したタスクは、待ち状態に移行し、待ち行列につながれます。

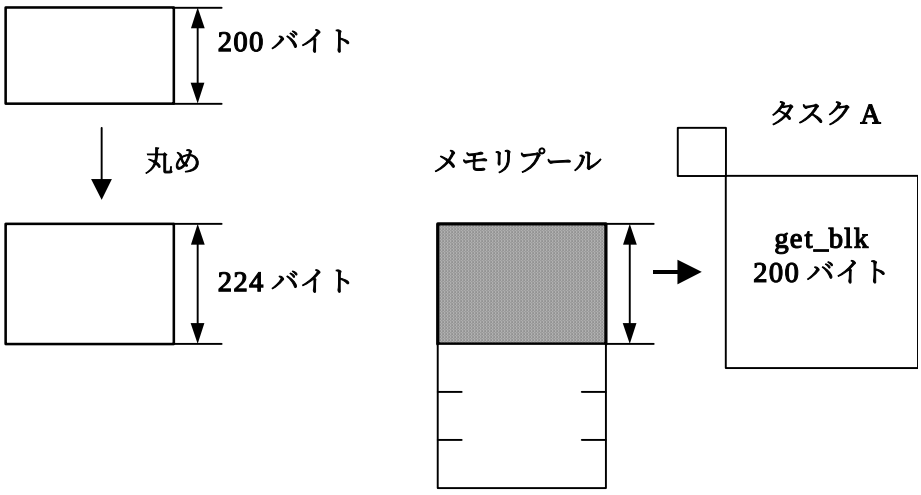


図 3.41 get_blk 処理

図 3.41では、タスク A が 200 バイトのメモリブロックを要求している例です。MR32R では、200 バイトのサイズを 224 バイトに丸め、224 バイトのメモリブロックをメモリプールから切り出し、タスク A に割り当てている様子を示しています。

- メモリブロックを獲得する（pget_blk）
  メモリブロックを可変長メモリプールから獲得します。get_blk、tget_blk システムコールと異なるところは、指定されたサイズのメモリブロックがメモリプールにない場合は、待ち状態に移行せず、エラーコードを返します。

- メモリブロックを解放する（rel_blk）
  get_blk、tget_blk あるいは tget_blk で獲得したメモリブロックを解放します。指定されたメモリプールのメモリブロックを待っているタスクがない場合は、メモリプールにメモリブロックを返却します。

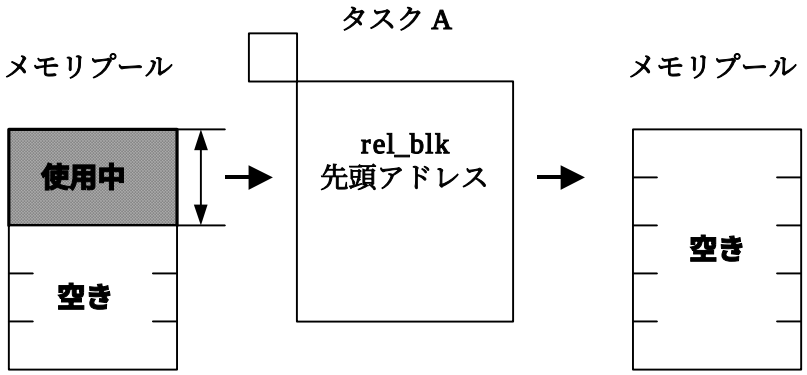


図 3.42 rel_blk 処理

次に、メモリブロック解放時に対象メモリプールのメモリブロックを待っているタスクがある場合は、待ち行列の先頭タスクから要求サイズを調べ、条件を満足[41]すれば要求サイズ分のメモリを切り出し、タスクに割り当てて、待ち状態から実行可能(READY)状態へ移行します。さらに、次に接続されているタスクの要求サイズと解放したメモリの残りのサイズとを比較していきます。もし、要求を満足しないタスクがあれば、その時点で、メモリブロック割り当てを終了します。

[41]ここでの条件の満足とは、メモリの要求サイズが解放するよりも小さい場合をいいます。

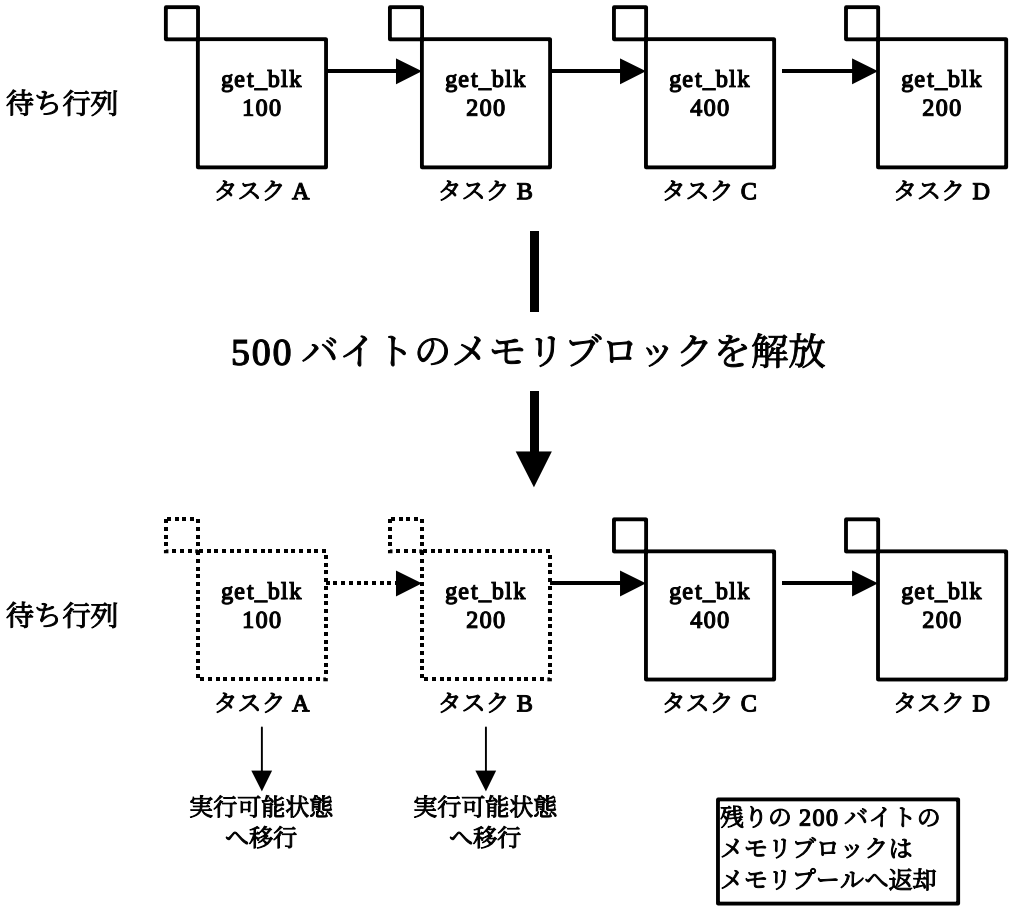


図 3.43 メモリブロックの解放

- メモリプールの状態を参照する（ref_mpl）
  メモリプールの空き領域の合計サイズやすぐに獲得できる最大の空き領域のサイズを参照します。

3.5.12 時間管理機能

時間管理機能はシステムの時刻を管理し、時刻の読みだし[42]、時刻の設定[43]機能、タイムアウトの処理や特定時刻に起動するアラームハンドラや定期的に起動する周期起動ハンドラの機能を提供します。

MR32R カーネルはシステムクロックとしてタイマを一つ必要とします。

MR32R カーネルが提供する時間管理システムコールには次のものがあります。

- タスクを一定時間待ち状態に移行します (dly_tsk)

  タスクを一定時間待たせます。図 3.44に dly_tsk システムコールにより 10msec 間タスクの実行を待たせる例を示します。

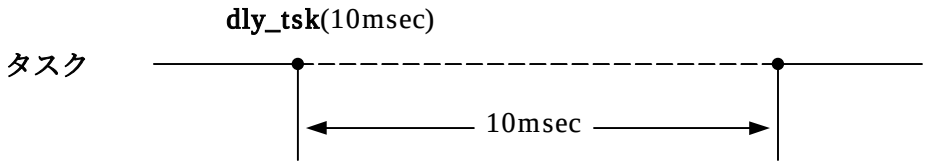


図 3.44 dly_tsk システムコール

- 待ち状態にタイムアウト値を指定すれば、一定時間待ち状態に移行します

  タスクを待ち状態に移行するシステムコール[44]にタイムアウトを指定することができます。システムコール名は、tslp_tsk、twai_flg、twai_sem、trcv_msg、tsnd_mbf、trcv_mbf、tcal_por、tacp_por、tget_blf、tget_blk です。タイムアウトの指定時間が経過するまでに待ち解除条件が満たされない場合、エラーコード E_TMOUT を返し、待ち状態が解除されます。待ち解除条件が満たされた場合は、エラーコードは E_OK を返します。

  タイムアウトの基準時間は、MR32R のシステムクロックの時間を基準としています。

[42] get_tim システムコール
[43] set_tim システムコール
[44] 強制待ち状態を除きます。

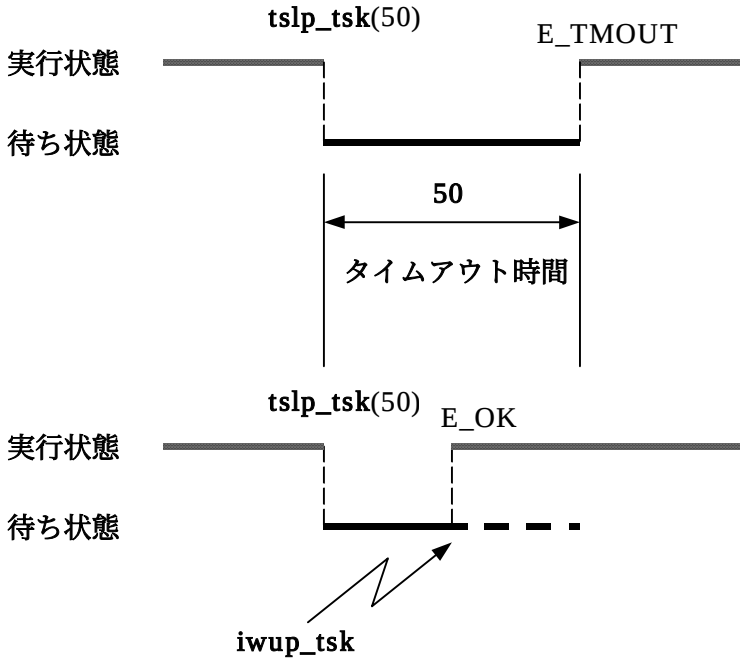


図 3.45 タイムアウト処理

- システム時刻を設定する (set_tim)

- システム時刻の値を読みだす (get_tim)
  システム時刻はリセット後のシステムクロック割り込みの発生回数をカウントし、48 ビットのデータで表します。

- 周期起動ハンドラを定義する (def_cyc)
  指定された ID 番号の周期起動ハンドラを定義します。

- 周期起動ハンドラの活性制御をおこなう (act_cyc)
  周期起動ハンドラは一定間隔で起動するプログラムです。(図 3.46参照) システムクロック割り込みの発生回数をカウントすることによって、一定時間で起動します。周期起動ハンドラは、システムコールで活性状態を指定することによって制御されます。例えば、TCY_ON を指定して活性状態を OFF から ON に変更したり(図 3.47参照)、TCY_INI_ON を指定して、ハンドラのカウント値を初期化する 図 3.48参照)ことができます。

- 周期起動ハンドラの状態を参照する (ref_cyc)
  対象周期起動ハンドラの活性状態やつぎの起動までの残り時間を参照します。

- アラームハンドラの状態を参照する (ref_alm)
  対象アラームハンドラの起動までの残り時間を参照します。

なお、システムクロックは必須機能ではありません。したがって下記のシステムコールおよび時間管理機能を使用しなければ、タイマを MR32R 用に占有する必要がありません。

1. システムクロックの設定、読みだし[45]
2. 周期起動ハンドラ
3. アラームハンドラ
4. dly_tsk システムコール
5. タイムアウト機能を使用したシステムコール[46]

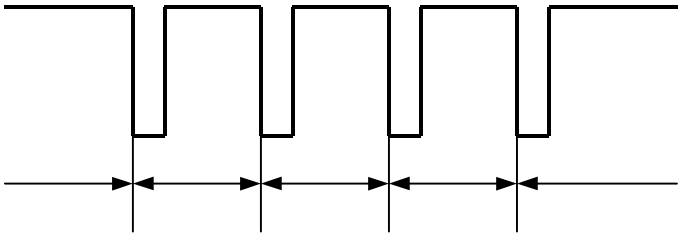


図 3.46 周期起動ハンドラ

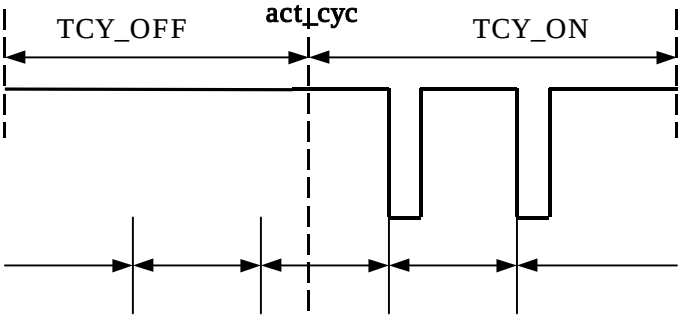


図 3.47 周期起動ハンドラ:活性状態 TCY_ON を指定

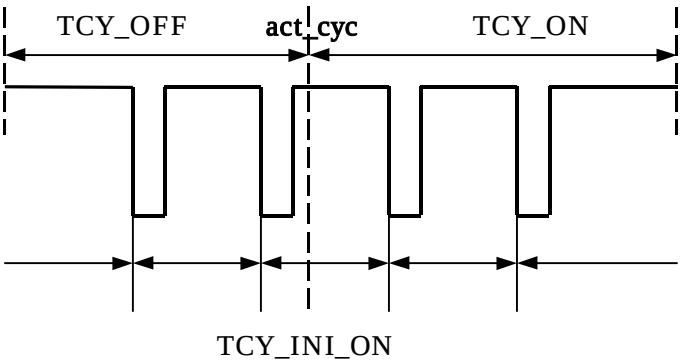


図 3.48 周期起動ハンドラ:活性状態 TCY_INI_ON を指定

[45] set_tim, get_tim システムコール

[46] tslp_tsk、twai_flg、twai_flg、trcv_msg、tsnd_mbf、trcv_mbf、tcal_por、tacp_por、tget_blf、tget_blk など

3.5.13 システム管理機能

- MR32R のバージョンを得る(get_ver)
  MR32R のバージョンを get_ver システムコールにより得ることができます。
  このバージョンは TRON 仕様で標準化された形式で得ることができます。このシステムコールでは以下の情報を得ることができます。

  - ♦ メーカー名
    ルネサステクノロジを示す番号
  - ♦ 形式番号
    製品識別番号
  - ♦ 仕様書バージョン
    μITRON 仕様であることを表す番号とこの製品のもとになったμITRON 仕様書のバージョン番号
  - ♦ 製品バージョン
    MR32R のバージョン番号
  - ♦ 製品管理情報
    製品のリリース番号、リリース日等の情報
  - ♦ MPU 情報
    MR32R ファミリマイクロコンピュータを示す番号
  - ♦ バリエーション記述子
    MR32R で利用できる機能番号

- システムの状態を参照する (ref_sys)
  CPU や OS の状態を参照します。下記に示す状態を参照できます。
  - ♦ システムの状態
    現在、タスクあるいはタスク独立部のどちらを実行しているか、タスク実行中であれば、ディスパッチを禁止中か、割り込み禁止中かなどを知ることができます。
  - ♦ 実行中のタスク ID
  - ♦ 実行中のタスクの優先度
  - ♦ 実行中のタスクあるいはタスク独立部の PSW(プロセッサステータスワード)値

- 例外ハンドラを定義する (def_exc)
  例外ハンドラについて、以下に示す内容を定義します。
  - ♦ 例外ハンドラ属性
    例外ハンドラが使用するスタックを内蔵 RAM にするか外部 RAM にするかを指定します。
  - ♦ 例外ハンドラアドレス
    例外ハンドラのエントリアドレスを指定します。
  - ♦ タスク ID 番号
    例外ハンドラを定義するタスクの ID 番号を指定します。
  - ♦ スタックサイズ
    例外ハンドラが使用するスタックのサイズを指定します。

3.5.14 拡張機能

拡張機能は、μITRON V.3.0 仕様外の機能です。

例外管理機能とは、プログラム処理中に、例外[47]が発生した場合に、それを処理するハンドラとして例外ハンドラ[48]を定義し、実行する機能をいいます。

例外ハンドラは、タスク毎に定義します。また、タスクと同様のシステムコールを発行することができますので、より優先度の高いタスクが READY 状態になった場合や例外ハンドラから待ち状態に移行するシステムコールを発行した場合は、他のタスクにディスパッチされます。

MR32R カーネルが提供する例外管理のシステムコールには次のものがあります。

- 強制例外ハンドラを起動する(vras_fex)
  指定した ID のタスクに対して強制例外ハンドラを起動します。

- 例外マスクをクリアする(vclr_ems)
  指定した ID のタスクに対して例外マスクをクリアします。

- 例外マスクをセットする(vset_ems)
  指定した ID のタスクに対して例外マスクをセットします。

- 例外ハンドラからタスク復帰する(vret_exc)
  例外ハンドラからタスクに処理が復帰します。

- メールボックスをクリアする(vrst_msg)
  メールボックスをクリアします。

- メッセージバッファをクリアする(vrst_mbf)
  メッセージバッファをクリアします。送信待ちのタスクがある場合は、待ち状態を解除し、エラーコード EV_RST を返します。

- 固定長メモリプールをリセットする(vrst_blf)
  固定長メモリプールをリセットします。待ち状態のタスクがある場合は、待ち状態を解除し、エラーコード EV_RST を返します。

- 可変長メモリプールをリセットする(vrst_blk)
  可変長メモリプールをリセットします。待ち状態のタスクがある場合は、待ち状態を解除し、エラーコード EV_RST を返します。

---

[47] システム全体に異常が検出された場合など

[48] **MR32R** では、強制例外ハンドラのみが定義できます。その他の例外ハンドラ(強制終了ハンドラ、CPU 例外ハンドラ)は、定義できません。

3.5.15 拡張機能 (優先度付きメイルボックス)

メイルボックスは、送信されたメッセージを入れるためのメッセージキューとメッセージの受信を待つタスクの待ち行列を持ちます。
送信するメッセージは、メッセージの先頭番地のみでメッセージ内容のコピーは行いません。

【従来のメイルボックスとの相違点】

カーネルは、メッセージキューをリンクリストで管理します。アプリケーション側でリンクリストに用いるためのヘッダ領域を用意しなければいけません。これをメッセージヘッダと呼びます。メッセージヘッダと実際にアプリケーションが使用するメッセージを格納する領域をメッセージパケットと呼びます。カーネルはメッセージキューヘッダの内容を書き換えて管理しています。アプリケーション側からは書き換えることはできません。
メッセージヘッダのデータ型は以下の通り定義しています。

T_MSG: **メイルボックスのメッセージヘッダ**
T_MSG_PRI: **メイルボックスの優先度付きメッセージヘッダ**

メッセージキューに入れることのできるメッセージのサイズは、アプリケーション側でヘッダ領域を確保するため、制限はありません。また、送信するためにタスクが待ち状態になることはありません。
メッセージに優先度を設定し、優先度の高いメッセージから受信することができます。さらに、メッセージ待ち状態のタスクがメッセージを受信する際、優先度の高いタスクからメッセージを受信することもできます。

- 優先度付きメイルボックスを生成する(vcre_mbx)
  優先度付きメイルボックスを生成します。

- 優先度付きメイルボックスを削除する(vdel_mbx)
  優先度付きメイルボックスを削除します。

- メッセージを送信する(vsnd_mbx,visnd_mbx)
  メッセージを送信します。送信するメッセージに優先度をつけ、優先度の高いメッセージ(優先度の小さい)からタスクに受信させることができます。

- メッセージを受信する(vrcv_mbx,vtrcv_mbx,vprcv_mbx)
  メッセージを受信します。送信されたメッセージがない場合は、待ち状態に移行します。待ち状態に移行したタスクが、メッセージを受信する際、タスクの優先度の高い(優先度の小さい)タスクから順にメッセージを受信させることもできます。

- 優先度付きメイルボックスをクリアする(vrst_mbx)
  優先度付きメイルボックスをクリアします。

- 優先度付きメイルボックスの状態を参照する(vref_mbx)
  優先度付きメイルボックスの状態を参照します。

3.6 タスク、ハンドラから発行できるシステムコール一覧

システムコールにはタスクから発行できるものとハンドラから発行できるもの、その両方から発行できるものがあります。

表 3-2にその一覧を示します。

表 3-2　タスク、ハンドラから発行できるシステムコール一覧

システムコール	タスク 例外ハンドラ	割り込みハンドラ 周期起動ハンドラ アラームハンドラ
cre_tsk	○	×
del_tsk	○	×
sta_tsk	○	×
ista_tsk	×	○
ext_tsk	○	×
exd_tsk	○	×
ter_tsk	○	×
dis_dsp	○	×
ena_dsp	○	×
chg_pri	○	×
ichg_pri	×	○
rot_rdq	○	×
irot_rdq	×	○
rel_wai	○	×
irel_wai	×	○
get_tid	○	○
ref_tsk	○	○
sus_tsk	○	×
isus_tsk	×	○
rsm_tsk	○	×
irsm_tsk	×	○
slp_tsk	○	×
tslp_tsk	○	×
wup_tsk	○	×
iwup_tsk	×	○
can_wup	○	○
cre_flg	○	×
del_flg	○	×
set_flg	○	×
iset_flg	×	○
clr_flg	○	○
wai_flg	○	×
twai_flg	○	×
pol_flg	○	○
ref_flg	○	○

システムコール	タスク 例外ハンドラ	割り込みハンドラ 周期起動ハンドラ アラームハンドラ
cre_sem	○	×
del_sem	○	×
sig_sem	○	×
isig_sem	×	○
wai_sem	○	×
twai_sem	○	×
preq_sem	○	○
ref_sem	○	○
cre_mbx	○	×
del_mbx	○	×
snd_msg	○	×
isnd_msg	×	○
rcv_msg	○	×
trcv_msg	○	×
prcv_msg	○	○
ref_mbx	○	○
cre_mbf	○	×
del_mbf	○	×
snd_mbf	○	×
tsnd_mbf	○	×
psnd_mbf	○	×
rcv_mbf	○	×
trcv_mbf	○	×
prcv_mbf	○	×
ref_mbf	○	○
cre_por	○	×
del_por	○	×
cal_por	○	×
tcal_por	○	×
pcal_por	○	×
acp_por	○	×
tacp_por	○	×
pacp_por	○	×
fwd_por	○	×
rpl_rdv	○	×
ref_por	○	○

システムコール	タスク 例外ハンドラ	割り込みハンドラ 周期起動ハンドラ アラームハンドラ
def_int	○	×
ret_int	×	○[49]
loc_cpu	○	×
unl_cpu	○	×
cre_mpf	○	×
del_mpf	○	×
get_blf	○	×
tget_blf	○	×
pget_blf	○	○
rel_blf	○	×
irel_blf	×	○
ref_mpf	○	○
cre_mpl	○	×
del_mpl	○	×
get_blk	○	×
tget_blk	○	×
pget_blk	○	×
rel_blk	○	×
ref_mpl	○	○
set_tim	○	○
get_tim	○	○
dly_tsk	○	×
act_cyc	○	○
ref_cyc	○	○
ref_alm	○	○
get_ver	○	○
ref_sys	○	○
def_exc	○	×
vclr_ems	○	×
vset_ems	○	×
vret_exc	○※	×
vras_fex	○	×
vrst_msg	○	○
vrst_mbf	○	×
vrst_blf	○	×
vrst_blk	○	×
vcre_mbx	○	×
vdel_mbx	○	×
def_cyc	○	×

[49] C 言語を用いて記述された割り込みハンドラからは発行できません。

システムコール	タスク 例外ハンドラ	割り込みハンドラ 周期起動ハンドラ アラームハンドラ
vsnd_mbx	○	×
visnd_mbx	×	○
vrcv_mbx	○	×
vtrcv_mbx	○	×
vprcv_mbx	○	○
vrst_mbx	○	×
vref_mbx	○	○

※:vret_exc は、タスクからは発行しないでください。

第 4 章　　アプリケーション作成手順概要

本章ではMR32Rを用いてアプリケーションプログラムを開発する手順の概要について説明します。

4.1 概要

MR32Rのアプリケーションプログラムは一般的に以下に示す手順で開発します。

1. **アプリケーションプログラムのコーディング**
   C 言語もしくはアセンブリ言語を用いてアプリケーションプログラムをコーディングします。

2. **割り込み制御用プログラムの作成**
   割り込みコントローラ、タイマなどに依存する部分の処理のプログラムを作成します。本製品では、M32R 評価用ボード M3A-2131 に対応した、プログラム ipl.ms(**M3T-CC32R** 対応)および ipl.s(**M3T-TW32R**、D-CC/M32R 対応)を同梱しております。

3. **コンフィグレーションファイル作成**
   タスクのエントリーアドレスやスタックサイズなどを定義したコンフィグレーションファイルをエディタを用いて作成します。

4. **コンフィグレータ実行**
   コンフィグレーションファイルからシステムデータ定義ファイル (sys_rom.inc、sys_ram.inc)、インクルードファイル (mr32r.inc、id.h)およびシステム生成手順記述ファイル (makefile) を作成します。

5. **システム生成**
   make[50]コマンドを実行してシステムを生成します。

6. **ROM 書き込み**
   作成された ROM 書き込み形式ファイルにより、ROM に書き込みます。もしくはデバッガに読み込ませてデバッグを行います。

図 4.1にシステム生成の詳細フローを示します。

[50] make コマンドは、UNIX 標準もしくは準拠のコマンドのみ使用可能です。

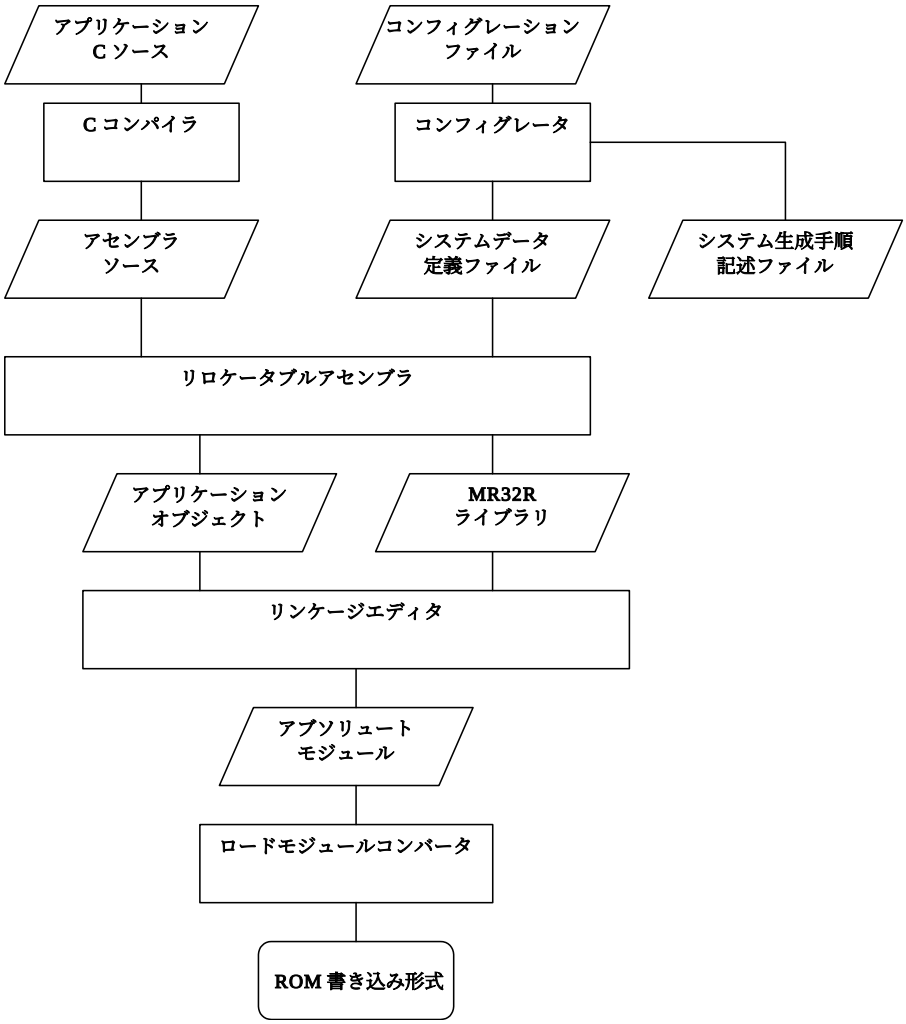


図 4.1　MR32R システム生成詳細フロー

4.2 開発手順例

この節では MR32R のアプリケーション例をもとに開発手順の概要について説明します。

4.2.1 アプリケーションプログラムのコーディング

図 4.2にレーザビームプリンタの動作をシミュレーションするプログラムを示します。このレーザービームプリンタのシミュレーションプログラムを記述したファイルの名前を"lbp.c"とします。このプログラムは以下の３つのタスクと１つの割り込みハンドラから構成されます。

- メインタスク
- イメージ展開タスク
- プリンタエンジンタスク
- セントロニクスインターフェース割り込みハンドラ

このプログラムでは MR32R ライブラリ中の以下の機能を利用します。

- sta_tsk()
  タスク起動をおこないます｡引数は起動すべきタスクを選択するための ID 番号を与えます。ＩＤ番号はコンフィグレータの生成するファイル"id.h"をインクルードすることにより名前(文字列)でタスクを指定することができます。[51]
- wai_flg()
  イベントフラグが立つまで待ちます。この例ではセントロニクスインターフェースから１ページ分データがバッファに溜まるのを待つために使用しています。
- wup_tsk()
  指定タスクを待ち状態から起床します。プリンタエンジンタスクを起動するために使用しています。
- slp_tsk()
  タスクを実行状態から待ち状態にします。この例ではプリンタエンジンタスクをイメージ展開待ちにするため使用しています。
- iset_flg()
  イベントフラグを立てます。この例では、１ページ分のデータ入力完了をイメージ展開タスクに知らせるために使用しています。

[51] すなわち、コンフィグレータがコンフィグレーションファイルに記述されている情報をもとに ID 番号を名前（文字列）に置き換えます。

```
#include <mr32r.h>
#include "id.h"


void main() /* main task */
{
    printf("LBP シミュレーション開始¥n");
    sta_tsk(ID_idle,1);     /* アイドルタスク起動 */
    sta_tsk(ID_image,1);    /* イメージ展開タスク起動 */
    sta_tsk(ID_printer,1);  /* プリンタエンジンタスク起動 */
}
void image() /* イメージ展開タスク */
{
    while(1){
            wai_flg(&flgptn,ID_pagein,waiptn,TWF_ANDW+TWF_CLR);/* 1ページ入力
待ち */

            printf("ビットマップ展開処理¥n");
            wup_tsk(ID_printer);    /* プリンタエンジンタスク起床 */
    }
}
void printer() /* プリンタエンジンタスク */
{
    while(1){
            slp_tsk();
            printf("プリンタエンジン動作¥n");
    }
}
void sent_in() /* セントロニクスインターフェースハンドラ */
{


    /* セントロニクスインターフェースからの入力処理 */
    if ( /* 1ページ入力完了 */ )
            iset_flg(ID_pagein,setptn);
}
```

図 4.2　プログラム例

4.2.2 コンフィグレーションファイル作成

タスクのエントリーアドレスやスタックサイズなどを定義したコンフィグレーションファイルを作成します。図 4.3 にレーザービームプリンタシュミレーションプログラムのコンフィグレーションファイル（ファイル名 "lbp.cfg"）を示します。

```
// System Definition
system{
    stack_size              = 1024;
    priority                = 5;
};
//System Clock Definition
clock{
    timer_clock             = 33.3MHz;
    timer                   = MFT00;
    file_name               = m32102.tpl;
    IPL                     = 4;
    unit_time               = 10ms;
    initial_time            = 0:0:0;
};
//Task Definition
task[1]{
    entry_address           = main();
    stack_size              = 512;
    priority                = 1;
    initial_start           = ON;
};
task[2]{
    entry_address           = image();
    stack_size              = 512;
    priority                = 2;
};
task[3]{
    entry_address           = printer();
    stack_size              = 512;
    priority                = 4;
};
task[4]{
    entry_address           = idle();
    stack_size              = 256;
    priority                = 5;
};
//Eventflag Definition
flag[1]{
     name                   = pagein;
};
//Interrupt Vector Definition
interrupt_vector[16]                = __sys_timer;
interrupt_vector[23]                = sent_in();
```

図 4.3 コンフィグレーションファイル例

4.2.3 コンフィグレータ実行

コンフィグレータ cfg32r を実行して、コンフィグレーションファイルからシステムデータ定義ファイル(sys_rom.inc, sys_ram.inc)、インクルードファイル(mr32r.inc, id.h)およびシステム生成手順記述ファイル(makefile)を作成します。

```
A> cfg32r  -mv  lbp.cfg
MR32R system configurator V.3.50.00 (for CC32R)
COPYRIGHT(C) 2001(2003) RENESAS TECHNOLOGY CORPORATION
AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED.
MR32R version ==> V.3.50 Release 1
A>
```

図 4.4 コンフィグレータ実行

4.2.4 システム生成

make コマンド[52]を実行してシステムを生成します。

```
C> make –f  makefile
as32R –g crt0mr.ms
cc32R –c task.c
lnk32R lnk32R.sub

C>
```

図 4.5 システム生成

4.2.5 ROM 書き込み

ロードモジュールコンバータ LMC32R(**M3T-CC32R 対応**),m32r-elf-objcopy(**M3T-TW32R 対応**), ddump (**D-CC/M32R 対応**) により、アブソリュートモジュールファイルを ROM 書き込み形式に変換し、ROM に書き込みます。

[52] make コマンドは MS-DOS 標準のものと、UNIX 標準もしくは準拠のものとがあります。**MR32R** では、UNIX 標準もしくは UNIX 準拠の make コマンドのみ使用できます。Windows 版を使用する場合は、UNIX 互換の make コマンド（GNU make コマンドなど）を使用してください。UNIX 互換の make コマンド対応状況については、リリースノートを参照下さい。本章では、UNIX 互換の make コマンドを実行する場合を例として説明します。

第 5 章　　アプリケーション作成手順詳細

5.1 C 言語によるコーディング方法

本節では、C 言語を用いてアプリケーションプログラムを記述する方法について述べます。

5.1.1 タスクの記述方法

C 言語を用いてタスクを記述する場合、以下の項目に注意してください。

1. **タスクは関数として記述します。**
   そのタスクを MR32R に登録するにはコンフィグレーションファイルに関数名を記述します。例えば関数名 “task()” をタスク ID 番号 3 で登録するには以下のようにおこないます。

```
Task[3]{
        entry_address   = task();
        stack_size      = 100;
        priority        = 3;
};
```

2. **ファイル先頭で必ずシステムディレクトリのなかの “mr32r.h” とカレントディレクトリ内の “id.h” をインクルードしてください。すなわちファイルの先頭で以下の 2 行を必ず記述してください。**

```
#include <mr32r.h>
#include “id.h”
```

3. **タスク開始関数の戻り値はありません。したがって、void 型で宣言してください。**
4. **スタティック宣言をおこなった関数はタスクとして登録できません。**
5. **タスク開始関数の出口では、かならず ext_tsk()を記述してください。**
6. **タスク開始関数を無限ループで記述することも可能です。**

```
#include <mr32r.h>
#include “id.h”
void task(void)
{
        /* 処理 */
        ext_tsk();
}
```

図 5.1 C 言語で記述したタスクの例

```
#include <mr32r.h>

#include “id.h”

void task(void)
{
        for(;;){
                /* 処理 */

        }
}
```

図 5.2　C言語で記述した無限ループタスクの例

7. **タスクを指定する場合はタスクの関数名に“ID_”を追加した文字列で指定してください[53]。**

```
wup_tsk(ID_main);
```

8. **イベントフラグ、セマフォ、メイルボックスを指定する場合は、コンフィグレーションファイルで定義したそれぞれの名前に”ID_”を追加した文字列で指定してください。**

例えば、コンフィグレーションファイルで以下のようにイベントフラグを定義した場合は、

```
flag[1]{
        name    = abc;
};
```

このイベントフラグを指定するには以下のようにおこなってください。

```
set_flg(ID_abc,&setptn);
```

9. **周期起動ハンドラ、アラームハンドラを指定する場合は、そのハンドラの開始関数名に“ID_”を追加した文字列で指定してください。例えば、周期起動ハンドラ“cyc()”を指定する場合は、以下のようにおこなってください。**

```
act_cyc(ID_cyc,TCY_ON);
```

10. **タスクを** ter_tsk() **システムコールなどで終了した後で** sta_tsk() **システムコールで再起動した場合は、タスク自身は初期状態から開始します[54]が、外部変数、スタティック変数はタスクの開始にともなっての初期化はされません。外部変数、スタティック変数の初期化は** MR32R **が立ち上がる前に起動されるスタートアッププログラムでのみおこないます。**

11. MR32R **システム起動時に起動されるタスクは、コンフィグレーションファイルで設定します。**

12. **変数の記憶クラスについて**

C言語の変数は MR32R から見て表 5-1に示す扱いになります。

[53] コンフィグレータがタスクの ID 番号をタスクを指定するための文字列に変換するためのファイル”id.h”を生成します。すなわち、タスクの開始関数名に”ID_”を付加した文字列をそのタスクの ID 番号に変換するための#define 宣言を”id.h”で行います。

[54] タスクの開始関数から初期優先度でなおかつ起床カウントがクリアされた状態で開始します。

表 5-1　C 言語における変数の扱い

変数の記憶クラス	扱い
グローバル変数	すべてのタスクの共有変数
関数外のスタティック変数	同一ファイル内のタスクの共有変数
オート変数 レジスタ変数 関数内のスタティック変数	タスク固有の変数

5.1.2 割り込みハンドラの記述方法

C言語を用いて割り込みハンドラを記述する場合、以下の点に注意してください。

1. **割り込みハンドラは関数として記述します。**[55]
2. **割り込みハンドラ開始関数の戻り値および引き数は、必ずvoid型で宣言してください。**
3. **ファイルの先頭で必ずシステムディレクトリのなかの"mr32r.h" とカレントディレクトリ内の"id.h"をインクルードしてください。**
4. **スタティック宣言をおこなった関数は、割り込みハンドラとしては登録できません。**

```
#include <mr32r.h>
#include “id.h”
void int_handler(void)
{
        /* 処理 */
        iwup_tsk(ID_main);
}
```

図 5.3 C言語で記述した割り込みハンドラの例

[55] ハンドラと関数名との対応はコンフィグレーションファイルにより行います。

5.1.3 周期起動ハンドラ、アラームハンドラの記述方法

C 言語を用いて周期起動ハンドラおよびアラームハンドラを記述する場合、以下の点に注意してください。

1. **周期起動ハンドラおよびアラームハンドラは関数として記述します。**[56]
2. **関数の戻り値および引き数を、void 型で宣言してください。**
3. **ファイルの先頭で必ずシステムディレクトリのなかの”mr32r.h”とカレントディレクトリ内の “id.h”をインクルードしてください。**
4. **スタティック宣言をおこなった関数は周期起動ハンドラおよびアラームハンドラとしては登録できません。**
5. **周期起動ハンドラおよびアラームハンドラはシステムクロックの割り込みハンドラからサブルーチン呼び出しにより起動されます。**

```
#include <mr32r.h>

#include “id.h”

void cychand(void)
{
        /* 処理 */
}
```

図 5.4 C 言語で記述した周期起動ハンドラの例

[56] ハンドラと関数名との対応は、コンフィグレーションファイルにより行います。

5.1.4 例外(強制例外)ハンドラの記述方法

C言語を用いて例外ハンドラを記述する場合、以下の点に注意してください。

1. **例外ハンドラは関数として記述します。**
2. **関数の戻り値を、void型で宣言してください。**
3. **関数の引数は、例外情報パケットへのポインタ(T_EXC *pk_exc)、例外発生時のレジスタ情報パケットへのポインタ(T_REGS *pk_regs)、EIT 情報パケットへのポインタ(T_EIT *pk_eit)を指定してください。**
4. **ファイルの先頭で必ずシステムディレクトリのなかの"mr32r.h"とカレントディレクトリ内の "id.h"をインクルードしてください。**
5. **スタティック宣言をおこなった関数は例外ハンドラとして登録できません。**
6. **ハンドラの終了は、vret_exc システムコールで終了してください[57]。**

   Vret_exc システムコールを発行することで、例外ハンドラに対応したタスクに対応したタスクに処理が移行します。

```
#include <mr32r.h>
#include "id.h"
T_EXC exc;
T_REGS regs;
T_EIT eit;

/* プロトタイプ宣言 */
void exc_handler(T_EXC *, T_REGS *, T_EIT *);

void exc_handler(T_EXC &exc, T_REGS &regs, T_EIT &eit)
{

            /* 処理 */

            vret_exc();

}
```

図 5.5 C言語で記述した例外ハンドラの例

[57] ハンドラの終了に ext_tsk システムコールを発行することもできます。この場合、例外ハンドラに対応したタスクも終了しますので、ご注意ください。

5.2 アセンブリ言語によるコーディング方法

本節では、アセンブリ言語を用いてアプリケーションを記述する方法について述べます。

5.2.1 タスクの記述方法

アセンブリ言語を用いてタスクを記述する場合、以下の項目に注意してください。

1. **ファイルの先頭で必ず“mr32r.inc”をインクルードしてください。**
2. **タスクの開始アドレスを示すシンボルは外部シンボル宣言[58]をおこなってください。**
3. **タスクは無限ループか ext_tsk システムコールで終了してください。**

```
        .include  “mr32r.inc” ----- (1)
        .global    task     ----- (2)

task:
                ; 処理
        bra   task       ----- (3)
```

図 5.6 アセンブリ言語で記述した無限ループタスクの例

```
        .include  mr32r.inc
        .global   task

task:
                ; 処理
        ext_tsk
```

図 5.7 アセンブリ言語で記述した ext_tsk で終了するタスクの例

4. **タスク起動時のレジスタの初期値は、PC、PSW と起動コードを格納したレジスタ以外は不定です。**

   起動コードを格納したレジスタは、以下の通りです。

   - M32R ファミリクロスツール **M3T-CC32R** をご使用の場合
     R2、R4 レジスタに格納します
   - M32R ファミリクロスツール **D-CC/M32R** をご使用の場合
     R0、R2 レジスタに格納します
   - M32R ファミリ GNU クロスツール **M3T-TW32R** をご使用の場合
     R0、R2 レジスタに格納します

5. **タスクを指定する場合はタスクの開始シンボル名に”ID_”を追加した文字列で指定してください。**[59]

```
        wup_tsk  ID_task
```

[58] .GLOBAL 疑似命令を使用してください。
[59] コンフィグレータがタスクの ID 番号をタスクを指定するために文字列に変換するための命令をファイル”mr32r.inc”に生成します。

6. **イベントフラグ、 セマフォ、メイルボックスを指定する場合は、コンフィグレーションファイルで定義したそれぞれの名前に"ID_"を追加した文字列で指定してください。**

例えば、 コンフィグレーションファイルで以下のようにセマフォを定義した場合、

```
semaphore[1]{
        name            = abc;
};
```

このセマフォを指定するには以下のようにおこなってください。

```
Sig_sem  ID_abc
```

7. **周期起動ハンドラ、アラームハンドラを指定する場合は、そのハンドラの開始シンボル名に"ID_"を追加した文字列で指定してください。例えば、周期起動ハンドラ"cyc"を指定する場合は、以下のようにおこなってください。**

```
act_cyc  ID_cyc,TCY_ON
```

8. MR32R **システム起動時に起動されるタスクは、コンフィグレーションファイルで設定します。**[60]

[60] コンフィグレータがタスクの ID 番号を、タスクを指定するための文字列に変換するための命令をファイル"mr32r.inc"に生成します。すなわち、タスクの開始シンボル名に"ID_"を付加した文字列をそのタスク ID 番号に変換するための.EQU 宣言を"mr32r.inc"でおこないます。

5.2.2 割り込みハンドラの記述方法

アセンブリ言語を用いて割り込みハンドラを記述する場合、以下の項目に注意してください。

1. **ファイルの先頭で必ず"mr32r.inc"をインクルードしてください。**
2. **割り込みハンドラの開始アドレスを示すシンボルは外部宣言（グローバル宣言）[61]をおこなってください。**
3. **ハンドラ内で使用するレジスタは、ハンドラの入口でセーブし、使用後復帰して下さい。**
4. ret_int **システムコールにて復帰してください。**

```
                .include  “mr32r.inc”          ------(1)
                .global   inth                 ------(2)

        inth:
                ;  使用レジスタ退避             ------(3)
                iwup_tsk  ID_task1
                        :
                        処     理
                        :

                ;  使用レジスタ復帰             ------(3)

                ret_int                        ------(4)
```

図 5.8　割り込みハンドラの例

[61] .GLOBAL 疑似命令を使用してください。

5.2.3 周期起動ハンドラ、アラームハンドラの記述方法

アセンブリ言語を用いて周期起動ハンドラおよびアラームハンドラを記述する場合、以下の点に注意してください。

1. **ファイルの先頭で必ず”mr32r.inc”をインクルードしてください。**
2. **ハンドラの開始アドレスを示すシンボルは外部宣言（グローバル宣言）[62]をおこなってください。**
3. **周期起動ハンドラ、アラームハンドラは全て jmp 命令にて復帰してください。**

例えば、

```
.include        mr32r.inc       ----- (1)
.global         cychand         ----- (2)

cychand:
        st      R14,@-R15
                :
        ; ハンドラ処理
                :
        ld      R14,@R15+

        jmp     R14             ----- (3)
```

図 5.9 アセンブリ言語で記述したハンドラの例

[62] .GLOBAL 疑似命令を使用してください。

5.2.4 例外(強制例外)ハンドラの記述方法

アセンブリ言語を用いて例外ハンドラを記述する場合、以下の点に注意してください。

1. **ファイルの先頭で必ず"mr32r.inc"をインクルードしてください。**
2. **ハンドラの開始アドレスを示すシンボルは外部宣言（グローバル宣言)をおこなってください。**
3. **ハンドラの終了は、vret_exc システムコールで終了してください[63]。**

   Vret_exc システムコールを発行することで、例外ハンドラに対応したタスクに対応したタスクに処理が移行します。

例外ハンドラ起動時に渡されるパラメータは、以下のレジスタにセットされています。

レジスタ名	値
R0	T_EXC *pk_exc
R1	T_REGS *pk_regs
R2	T_EIT *pk_eit

```
.include        “mr32r.inc”     ----- (1)
.global         exc_handler     ----- (2)

exc_handler:

                :
        ; ハンドラ処理
                :

        vret_exc                        ----- (3)
```

図 5.10 アセンブリ言語で記述した例外ハンドラの例

[63] ハンドラの終了に ext_tsk システムコールを発行することもできます。この場合、例外ハンドラに対応したタスクも終了しますので、ご注意ください。

第 6 章　　アプリケーション作成時の注意事項

6.1 ディスパッチ遅延について

MR32R では、ディスパッチ遅延に関するシステムコールが 4 つあります。

- dis_dsp
- ena_dsp
- loc_cpu
- unl_cpu

これらのシステムコールを使用し、一時的にディスパッチを遅延した場合のタスクの扱いについて以下に記述します。

1. ディスパッチ遅延中の実行タスクがプリエンプトされる場合

ディスパッチが禁止されている間は、実行中のタスクがプリエンプトされるべき状況となっても、新たに実行すべき状態となったタスクにはディスパッチされません。実行するべきタスクへのディスパッチは、ディスパッチ禁止状態が解除されるまで遅延されます。ディスパッチ遅延中は、

- 実行中のタスクは RUN 状態であり、レディキューにつながれている。
- ディスパッチ禁止解除後に実行するタスクは、READY 状態であり、(タスクがつながれている中で)最高優先度のレディキューにつながれている。

2. ディスパッチ遅延中の isus_tsk,irsm_tsk

また、ディスパッチ禁止状態で起動された割り込みハンドラから、実行中のタスク(dis_dsp を発行したタスク)に対して isus_tsk を発行し SUSPEND 状態へ移行させようとした場合､タスク状態の遷移はディスパッチ禁止状態が解除されるまで遅延されます｡ディスパッチ遅延中は、

- 実行中のタスクの状態の扱いは、リアルタイム OS 内部では、ディスパッチ遅延解除後の状態として扱います。そのため、実行中のタスクに対して発行された isus_tsk では、実行中のタスクをレディキューからはずし、SUSPEND 状態に移行します。エラーコードは E_OK を返します。この後、実行中のタスクに対して irsm_tsk が発行されると、実行中のタスクをレディキューにつなぎ、エラーコードは E_OK を返します。ただし、ディスパッチ遅延が解除されるまでタスクの切り替えは起こりません。
- ディスパッチ禁止解除後に実行するタスクは、レディキューにつながれています。

3. ディスパッチ遅延中の rot_rdq, irot_rdq

ディスパッチ遅延中に､rot_rdq(TPRI_RUN=0)を発行した場合､自タスクの持つ優先度のレディキューを回転させます。また、irot_rdq(TPRI_RUN=0)を発行した場合、実行中のタスクが持つ優先度のレディキューが回転します。この場合、実行中のタスクは該当レディキューにはつながれていない場合があります。 (ディスパッチ遅延中に、実行タスクに対し isus_tsk が発行された場合など。)

4. 注意事項

- dis_dsp,loc_cpu により、ディスパッチが禁止されている状態で、自タスクを待ち状態に移す可能性のあるシステムコール (slp_tsk,wai_sem など)は発行できません。
- loc_cpu により割り込みおよびディスパッチを禁止した状態で ena_dsp,dis_dsp は発行できません。
- dis_dsp を何回か発行して、その後、ena_dsp を 1 回発行しただけでディスパッチ禁止状態は解除されます。
  上記の状態遷移をまとめると表 6-1のようになります。

表 6-1 dis_dsp,loc_cpu に関する割り込み、ディスパッチの状態遷移

状態番号	状態の内容		dis_dsp を実行	ena_dsp を実行	loc_cpu を実行	unl_cpu を実行
	割り込み	ディスパッチ				
1	許可	許可	→ 2	→ 1	→ 3	→ 1
2	許可	禁止	→ 2	→ 1	→ 3	→ 1
3	禁止	禁止	×	×	→ 3	→ 1

6.2 初期起動タスクについて

MR32R では、システム起動時に READY 状態からスタートするタスクを指定できます。この指定はコンフィグレーションファイルで設定を行います。
設定方法の詳細については、133ページを参照して下さい。

6.3 アラームハンドラ使用時の注意事項

MR32R では、アラームハンドラ使用時に以下のような仕様となっていますのでご注意ください。

- 一度起動されたアラームハンドラは、set_tim システムコールで起動時刻以前に設定されたとしても再度起動する事はありません。
- set_tim システムコールにより、まだ起動していないアラームハンドラの起動時刻より後に時刻が設定された場合、設定時刻以降のアラームハンドラが起動する事はありません。必ず、起動時刻の早いアラームハンドラから順に起動させるようにしてください。

6.4 動的生成・削除機能について

6.4.1 動的生成・削除機能におけるメモリ確保の仕組み

MR32R では動的生成機能(cre_tsk,cre_mpf,cre_mpl など)を使用する際、可変長メモリプール機能と同様に４つのサイズのメモリブロックのうち最適なサイズを割り当てるようにしています。対象となるシステムコールをに示します。

表 6-2 動的生成・削除システムコール一覧

システムコール	用途
cre_tsk,del_tsk,exd_tsk	スタック領域
def_exc	例外ハンドラ用スタック領域
cre_mbx,del_mbx	メイルボックス領域
cre_mbf,del_mbf	メッセージバッファ領域
cre_mpf,del_mpf	固定長メモリプール領域
cre_mpl,del_mpl	可変長メモリプール領域

4 つのメモリブロックのサイズ(a,b,c,d)も可変長メモリプールと同様に、以下のように求めることができます。以下のサイズは、管理情報を含むため実際にはこれより 12 バイト少ない値になります。

```
A =(((max_memsize +(12-1))/96)+1)*12
b =a*2
c =a*4
d =a*8
```

以下に例を挙げます。

【例】

- **ユーザープログラム内で"__MR_EXT__"を指定して生成したタスク**

  タスク A ： 256 バイトのスタック
  タスク B ： 1024 バイトのスタック

この場合、ID=3 のタスクのスタックサイズが最大となるので max_memsize = 1024 以上を指定します。コンフィグレーションファイルの ext_memstk 定義を max_memsize = 1024 とした場合、4 つのメモリブロックのサイズは上記計算式より以下のようになります。

種類	サイズ	実使用可能サイズ
a	132	120
b	264	252
c	528	516
d	1056	1040

(差分の 12 バイトは管理領域として使用)

タスク A のスタックサイズは、256 バイトとなりますのでサイズ c(=528)のブロックが割り当てられ、タスク B のスタックサイズ(1024 バイト)は d(=1056)バイト割り当てられるようになります。
したがって、要求サイズ、実使用サイズ、無駄なサイズの関係は以下のようになります。

	要求サイズ	実使用サイズ	無駄なサイズ
タスク A	256	528	272
タスク B	1024	1056	32
Total	1280	1584	304

従って、コンフィグレーションファイルには、下記のように記載すればメモリ不足となる事はありません。

- **コンフィグレーションファイルの記述**

```
ext_memstk{
        max_memsize     = 1024;
        all_memsize     = 1584;
};
```

しかし、この場合、無駄な領域が 304 バイトできます。
無駄な領域を減らすためには、max_memsize を調整する必要があります。例えば、max_memsize=1048 とすれば、a=144 b=288 c=576 d=1152 となり、

	要求サイズ	実使用サイズ	無駄なサイズ
タスク A	256	288	32
タスク B	1024	1152	128
Total	1280	1440	160

と無駄なサイズを減らすことが可能です。

6.4.2 動的生成・削除機能使用時の注意事項

- 本方式では、フラグメンテーションは、発生しにくくなっていますが、全く発生しないわけではありません。そのため、空き領域のサイズは十分あっても連続した空き領域がないため、メモリが割り当てられないこともあるので注意してください。
- cre_mpl を使用して生成した可変長メモリプールは、生成時に指定したメモリサイズではなく、4 種類のサイズのメモリブロックから OS が割り当てたメモリブロックのサイズ分をメモリプール領域として利用します。従って、コンフィグレーションファイルでメモリプールサイズとして指定した値を cre_mpl の引数にした場合、cre_mpl を使用した方が実際に使用されるメモリプールサイズが大きくなることがあります。

6.5 TRAP 命令の使用について

MR32R では、TRAP 命令の割り込み番号を表 6-3に示すようにシステムコール発行のため予約しています。そのため、ユーザアプリケーションでソフトウェア割り込みを使用する場合は、7～12 以外の割り込みを使用して下さい。

表 6-3　割り込み番号の割り当て

割り込み番号	使用するシステムコール
7	タスクからのみ発行可能なシステムコール
8	タスク独立部からのみ発行可能なシステムコール タスク、タスク独立部の両方から発行可能なシステムコール
9～12	拡張のため予約

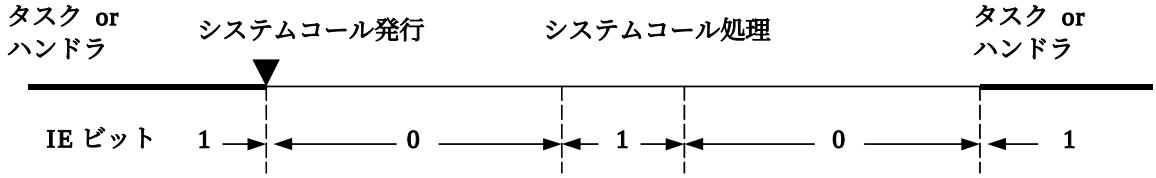
6.6 割り込みについて

6.6.1 割り込み制御方法

システムコール内の割り込み禁止/許可の制御は、PSW の IE ビット操作により行っています。
システムコール内での IE ビットは、クリアされており、割り込みを禁止しています。全ての割り込みを許可できる箇所では、システムコール発行時の IE ビットに戻します。
図 6.1に、システムコール内での IE ビットの状態を示します。

- タスクからのみ発行できるシステムコールの場合

・システムコール発行前の IE ビットが 1 の場合



・システムコール発行前の IE ビットが 0 の場合

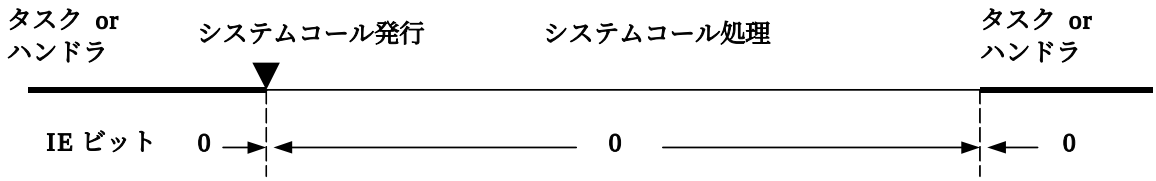


図 6.1 システムコール内での割り込み制御

図 6.1に示すように IE ビットは、システムコール内で変化します。そのため、ユーザアプリケーション内で割り込みを禁止したい場合、IE ビットの操作によって割り込みを禁止にする方法はお奨めできません。
割り込みの制御は、以下に示す二つの方法をお奨めします。

1. **禁止にしたい割り込みの割り込み制御レジスタを変更する。**
2. loc_cpu ～ unl_cpu **を使用する。**

6.6.2 割り込みハンドラの処理手順

MR32R では、図 6.2のような手順で割り込みハンドラの処理を行います。

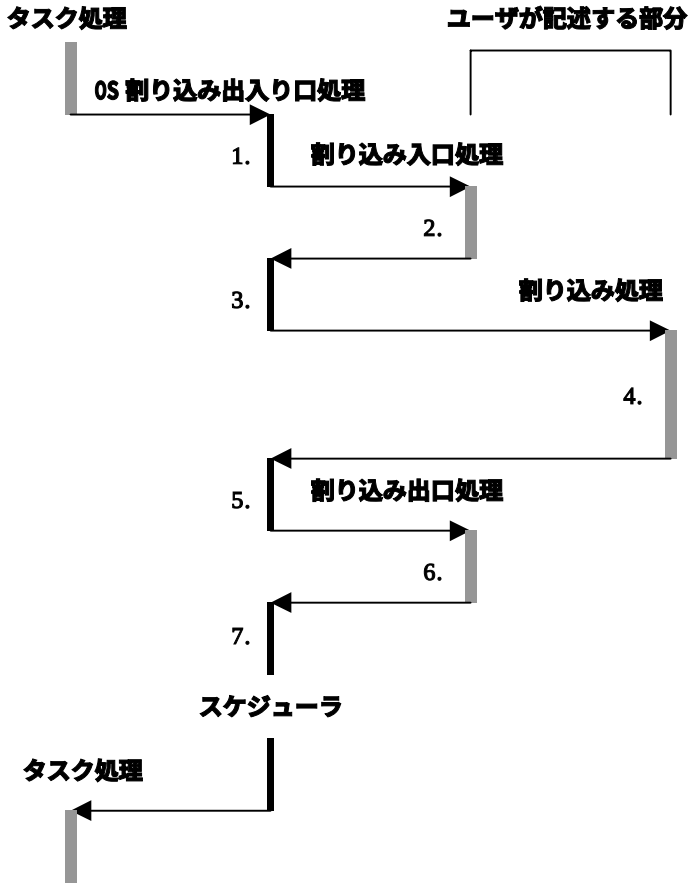


図 6.2 割り込みハンドラの処理手順

1. レジスタを保存し、ユーザーの割り込み入り口処理ルーチンをコールする。
2. ユーザの割り込み入り口処理ルーチン
   - 割り込みレベルを読み、スタックに退避します。
   - 割り込み要因を読み、割り込みのジャンプ先アドレスをスタックに退避します。
3. スタック上に積まれた割り込みハンドラのアドレスをコールする。
4. 割り込み処理ルーチン
5. ユーザの割り込み出口処理ルーチンをコールする。
6. 割り込み出口処理
   - 割り込みレベルを元に戻します。
7. レジスタを復帰し、スケジューラへジャンプする。

上記処理の2と6は、割込み制御プログラム(ipl.ms あるいは ipl.s)内でユーザが記述する必要があり

ます。(詳細は、8.1割り込み制御プログラムについてを参照ください。)その他の部分については、OSで処理を行います。

6.6.3 ハンドラ実行時の割り込みの受付について

ハンドラ実行時の割り込み可否の状態は以下のようになっています。

- 周期起動ハンドラ、アラームハンドラ
  周期起動ハンドラ・アラームハンドラは、割り込み許可状態で起動されます。
- 割り込みハンドラ
  割り込みハンドラは、デフォルトでは割り込み禁止状態で起動されます。多重割り込みを許可する場合は、「6.6.4 多重割り込みの許可方法」を参照ください。

6.6.4 多重割り込みの許可方法

多重割り込みを許可するには下記に示す処理を行ってください。**ただし、OS のシステムクロック処理ルーチン(__sys_timer)を呼び出す際は、必ず割り込み禁止状態で呼び出してください。**

- すべての割り込みに対して多重割り込みを許可する場合
  1. 割り込み入口処理の 2.のルーチン内で、割り込み許可ビットをセットします。
  2. 割り込み出口処理の 6.のルーチン内で、割り込み許可ビットをクリアします。
- ある特定の割り込みだけ多重割り込みを許可する場合
  1. 多重割り込みを許可する割り込みハンドラの先頭で割り込み許可ビットをセットします。
  2. 多重割り込みを許可する割り込みハンドラの最後で割り込み許可ビットをクリアします。

6.7 OS デバッグ機能使用手順

OS デバッグ機能(タスクトレース、システムコールトレース、システムコール発行)を使用するための手順および注意事項を以下に示します。

6.7.1 OS デバッグ機能使用手順

1. コンフィグレーションファイルのシステム定義項目を編集します。
   定義方法についての詳細は、6.1 コンフィグレーションファイルの作成方法を参照してください。
2. フックルーチンのライブラリ mrdbg.lib(M3T-TW32R 使用の場合は libmrdbg.a)をリンクするようにします。
3. フックルーチンのセクション(MR_KERNEL)バッファ領域のセクション(MR_dbg_RAM)を任意の領域に配置します。
4. システムコール発行機能を使用する場合は、発行するシステムコールもコンフィグレーションファイルの使用システムコール定義で"YES"を指定します
5. 割り込みハンドラの定義でシステムクロック割り込みを__sys_timer としていたものを __Dbg_sys_timer に変更してください。
   (__Dbg_sys_timer は、システムコール発行機能用のフックルーチンです)

6.7.2 OS デバッグ機能使用時の注意事項

- デバッグ機能を使用する場合、システムクロック割り込みの間隔を 10ms 以上に設定するとトレース結果の時間軸がおかしくなる場合があります。(切り替えの要因、順序などは、正しく表示されます。) そのため、システムクロック割り込み間隔は、10ms 以内に設定するようお願いします。
- タスクトレース、システムコールトレース機能に対応していないデバッガ(PD32000)では使用することはできません。
- システムコール発行機能で発行可能なシステムコールは、"get_tid"を除く、割り込みハンドラから発行可能なシステムコールです。タスクからのみ発行可能なシステムコールは、システムコール発行機能では、使用できません。
- フックルーチンをリンクすることにより、OS の処理時間・割り込み禁止時間が長くなります。
- システムコールトレース、タスクトレース機能を制御するために"__Dbg_mode"という 1 バイトのデータで OS とデバッガとやりとりを行っています。標準では、スタートアップファイル内で、トレースを行わないように OS がデータを書き換えています。そのため、プログラムをダウンロードし、デバッガの MR トレースウインドウを開いても、トレースができません。OS がデータを書き換えてから MR トレースウインドウを開くか、いったん OS がデータを書き換えたところでブレークしてから再度実行してください。デフォルトでトレースを ON にするには、スタートアップファイルを以下のように変更してから使用してください。**M3T-CC32R** 使用時の変更例を以下に示します。

[変更後]

```
        ;       seth    R11,#high(__REL_BASE11)
        ;       or3     R11,R11,#low(__REL_BASE11)
        ;       seth    R12,#high(__REL_BASE12)
        ;       or3     R12,R12,#low(__REL_BASE12)
        ;       seth    R13,#high(__REL_BASE13)
        ;       or3     R13,R13,#low(__REL_BASE13)


        .AIF    \&__Dbg_flg gt 0
        ld24    r1,#__Dbg_mode
        ldi     r2,#0               ←R2 レジスタの設定値を変更する
        stb     r2,@r1                 この例では"AFTER"モードに設定
        .AENDI
```

[変更前]

```
        ;       seth    R11,#high(__REL_BASE11)
        ;       or3     R11,R11,#low(__REL_BASE11)
        ;       seth    R12,#high(__REL_BASE12)
        ;       or3     R12,R12,#low(__REL_BASE12)
        ;       seth    R13,#high(__REL_BASE13)
        ;       or3     R13,R13,#low(__REL_BASE13)


                .AIF    \&__Dbg_flg gt 0
                ld24    r1,#__Dbg_mode
        ;       ldi     r2,#0
                ldi     r2,#4
                stb     r2,@r1
                .AENDI
```

R2 レジスタに設定する値は、"AFTER"モードの場合は「0」"Break"モードの場合は「2」にします。

6.8 システムクロックの設定について

6.8.1 システムクロック処理の登録について

MR32R では、システムクロックを使用する場合、システムクロック処理(__sys_timer というラベルのアセンブラルーチン)を割り込みハンドラとして登録する必要があります。
登録は、通常の割り込みハンドラと同様にコンフィグレーションファイルで定義します。

例えば、割り込み要因番号 0 から 63 までの 64 の割り込み要因が存在し、割り込み要因番号 16 をシステムクロックとする場合以下のようにコンフィグレーションファイルに記述します。

```
【例】interrupt_vector[16]      =       __sys_timer;
```

システムクロック処理に関する注意事項

- **テンプレートファイルとして M32160.tpl または M32180.tpl を指定した場合**
  テンプレートファイルとして m32160.tpl または m32180.tpl を指定した場合、タイマ割り込みは、同じ割り込みエントリを使用しますので、どのタイマ割り込みが入ったか判定するルーチンは別途ユーザー側で用意する必要があります。コンフィグレーションファイルの割り込み定義で設定したルーチンがその判定ルーチンにあたります。システムクロック割り込みが発生した場合、この判定ルーチンでシステムクロック割り込みが発生したか判定し、システムクロック処理(__sys_timer)を呼び出すようにします。
  (__sys_timer を呼び出す際は、「bra 命令」ではなく「bl 命令」で呼び出すようにしてください。)
- システムクロック(__sys_timer)を呼び出す際は、外部割込みを禁止した状態(PSW レジスタの IE ビットをクリアした状態)で呼び出すようにしてください。

6.8.2 タイマの自動設定について

タイマの自動設定の仕組み

MR32R では、いくつかのマイコンについては、タイマの初期化や、ICU の設定などを、コンフィグレータによって自動的に出力できるようになっています。

その仕組みの概要は次のとおりです。まずコンフィグレーションファイルに、テンプレートファイル・タイマの種類・割り込み間隔・入力周波数を設定します。コンフィグレータは、環境変数 LIB32R にあるテンプレートファイルをコンフィグレータの実行ディレクトリに"timer.inc"というファイル名でコピーすると同時に、タイマの種類・割り込み間隔・入力周波数などをシンボル定義し、"mr32r.inc"に出力します。"timer.inc"は、ipl.ms ファイルでインクルードされるようになっており、"mr32r.inc"に出力されたシンボル定義をもとにタイマの設定を行います。
(従来から使用されていた"ipl.ms"ファイルは、そのままご使用いただくことができます。製品に添付されているテンプレートファイルおよびその対応機種は、表 7-3をごらんください。製品に添付されていない機種については、ユーザで作成する必要があります。)

タイマの自動設定機能に関する注意事項

- **タイマの自動設定機能を使用する場合**
  コンフィグレータの起動オプションに"-i"オプションをつけて実行する必要があります。
  コンフィグレータを実行したディレクトリにすでにタイマ設定ファイル"timer.inc"がコピーされている場合は、あらたにコピーされることはありません。

タイマの自動設定において、タイマカウンタ設定値のオーバーフローのチェックを行っていません。

- **タイマの自動設定機能を使用しない場合**
  TM を使用せずにコマンドラインで使用し、“-m”オプションをつけて makefile を自動的に生成する場合は、割り込み制御プログラムを作成してからコンフィグレータを実行してください。割り込み制御プログラムを作成せずに"-m"オプション付きで実行した場合、"timer.inc"をインクルードするように設定された割り込み制御プログラムがコピーされてしまい、"timer.inc”がないためにコンフィグレータがエラーを出力します。
  (ユーザが作成する際の割り込み制御プログラムは、サンプルプログラムが格納されているディレクトリにあるものを参考にしてください。)

6.9 コンパイラ依存の注意事項

6.9.1 M3T-CC32R 使用の場合

- **M3T-CC32R のベースレジスタ機能使用方法**

M3T-CC32R V.3.00 でサポートされたベースレジスタ機能を使用する場合、下記の手順で使用します。
(ベースレジスタ機能の詳細については、「M3T-CC32R ユーザーズマニュアル<<C コンパイラ編>>」拡張機能リファレンスを参照ください。)

1. ベースアドレスを決定します。
2. アクセス制御ファイルを作成します。
3. **ベースシンボルの定義を行います。**
   ベースシンボルの定義は、OS のスタートアップルーチンで行います。
   【例】R11=0x00FC8000,R12=0x00F88000,R13=未使用に設定する場合
   OS に添付されているスタートアップルーチン(crt0mr.ms または start.ms)の 48 行目から 66 行目を下記のように変更します。

```
        .global  __REL_BASE11
        .global  __REL_BASE12
        .global  __REL_BASE13
__REL_BASE11:    .equ     0x00FC8000        ← ①ベースシンボルを定義します。
__REL_BASE12:    .equ     0x00F88000
__REL_BASE13:    .equ     0

__START:
        seth     r1,#high(__Sys_Sp)
        or3      r1,r1,#low(__Sys_Sp)
        addi     r1,#-4
        mvtc     r1,SPI                ;SPI initialize
        mvtc     r1,SPU
        ldi      r0,#-1
        st       r0,@r1
        ldi      r0,#NULL
        mvtc     r0,PSW                ;PSW initialize

;       seth     R11,#high(__REL_BASE11)    ← ②スタートアップルーチンからC言語で
;       or3      R11,R11,#low(__REL_BASE11)      記述された関数を呼び出す場合、ベースレ
;       seth     R12,#high(__REL_BASE12)          ジスタの設定を行います。
;       or3      R12,R12,#low(__REL_BASE12)
;       seth     R13,#high(__REL_BASE13)
;       or3      R13,R13,#low(__REL_BASE13)
```

(注意)

- タスク・割り込みハンドラ起動時のベースレジスタの設定は、OS カーネルが行いますので、スタートアップルーチンから C 言語で記述された関数(I/O の初期設定関数など)を呼び出さない場合、ベースレジスタの設定は不要です。
- **ベースレジスタとして使用しないレジスタがある場合、ベースレジスタ機能をまったく使用しない場合であっても、R11～R13 までのベースシンボルが定義されている必要があります。ただし、その値はどのような値でもかまいません。**

4. コンパイル・リンクを行います。

6.9.2 M3T-TW32R 使用の場合

- **TM との組み合わせに関する注意事項**
  M3T-TW32R と TM と組み合わせて使用される場合、デフォルトで OS カーネルのライブラリのサーチパスが登録されません。ユーザーが TM にパスを登録する必要があります。その手順は以下のとおりです。

  1. TM のメニューの"Environment"から"Project Settings"を選択します。
  2. "Project Settings"画面の"LIBRARY"タブを選択します。
  3. "Path"ボタンを選択してライブラリのパスを設定してください。

  ライブラリは、インストール時に指定したディレクトリ下の"lib32rg"ディレクトリに格納されています。(c:\mtool にインストールした場合は、c:\mtool\lib32rg が該当するディレクトリにあたります。)

6.9.3 D-CC/M32R 使用の場合

- makefile の依存関係出力に関する注意事項
  Wind River Systems,Inc 製 C コンパイラ D-CC/M32R 使用時にコンフィグレータの起動オプション`-m`を指定し、makefile を生成させた場合、コンフィグレータは、コンパイラのシステムインクルードファイルの依存関係を出力する事ができず、ワーニングを出力します。
  これを回避するためには、以下の作業[64]が必要です。

  1. OS カーネルインクルードファイル`mr32r.h`をコンパイラのシステムインクルードディレクトリにコピーします。

  【例】 copy %INC32RG%\mr32r.h %DIABLIB%\include

  2. 環境変数 INC32RG をコンパイラのシステムインクルードディレクトリに変更します。

  【例】 set INC32RG=%DIABLIB%\include

[64] 例は、Windows95 を使用時のものです。

6.10 メモリ配置

6.10.1 M3T-CC32R 使用時のセクション配置

ここでは、M3T-CC32R を使用する場合、MR32R で使用するセクションについて説明します。

● C 言語使用時に使用するセクション

- **P セクション**
  ユーザのアプリケーションプログラムが配置されます。

- **B セクション**
  初期値なしのデータが配置されます。
  このセクションは、RAM 上に配置する必要があります。

- **C セクション**
  定数データが配置されます。

- **D セクション**
  初期値ありのデータが配置されます。
  このセクションは、ROM から RAM へ転送して使用します。

● MR32R で使用するセクション

- **SYS_STACK セクション**
  システムスタック領域です。MR32R カーネル内部、割り込みハンドラなどで使用します。
  このセクションは、RAM 上に配置する必要があります。

- **INT_USR_STACK セクション**
  内蔵 RAM に割り当てたユーザスタック領域です。
  このセクションは、RAM 上に配置する必要があります。

- **EXT_USR_STACK セクション**
  外部 RAM に割り当てたユーザスタック領域です。
  このセクションは、RAM 上に配置する必要があります。

- **MR_KERNEL・MR_KERNEL2 セクション**
  MR32R カーネルが配置されます。

- **MR_RAM セクション**
  MR32R が使用する内蔵 RAM のデータが配置されます。
  このセクションは、RAM 上に配置する必要があります。

- **EXT_MR_RAM セクション**
  MR32R が使用する外部 RAM のデータが配置されます。
  このセクションは、RAM 上に配置する必要があります。

- **MR_ROM セクション**
  MR32R が使用する固定データが配置されます。
  動的生成機能を使用する場合は、ROM から RAM へ転送して使用します。

- **MR_HEAP セクション**
  MR32R が使用する内蔵 RAM のヒープ領域です。可変長メモリプール機能を使用する場合、このセクションを使用します。
  このセクションは、RAM 上に配置する必要があります。

- **EXT_MR_HEAP セクション**
  MR32R が使用する外部 RAM のヒープ領域です。可変長メモリプール機能を使用する場合、このセクションを使用します。
  このセクションは、RAM 上に配置する必要があります。

● **割り込みベクタ関連のセクション**

- **Int_Vector セクション**

- **EIT_Vector セクション**
  EIT ベクタ領域を格納するセクションです。

- **RESET_VECT セクション**
  リセットベクタ領域です。

- **INTERRUPT_VECT セクション**
  コンフィグレータにより出力された割り込みベクタテーブルが格納されているセクションです。
  def_int システムコール使用時は、ROM から RAM へ転送して使用します。

6.10.2 M3T-TW32R、D-CC/M32R 使用時のセクション配置

ここでは、**M3T-TW32R,D-CC/M32R** を使用する場合、MR32R で使用するセクションについて説明します。

● **C 言語使用時に使用するセクション**

- **.text セクション**
  ユーザのアプリケーションプログラムが配置されます。

- **.bss、.sbss セクション**
  初期値なしのデータが配置されます。
  このセクションは、RAM 上に配置する必要があります。

- **.rodata セクション**
  定数データが配置されます。

- **.sdata、.data セクション**
  初期値ありのデータが配置されます。
  このセクションは、ROM から RAM へ転送して使用します。

● **MR32R で使用するセクション**

- **.SYS_STACK セクション**
  システムスタック領域です。MR32R カーネル内部、割り込みハンドラなどで使用します。
  このセクションは、RAM 上に配置する必要があります。

- **.INT_USR_STACK セクション**
  内蔵 RAM に割り当てたユーザスタック領域です。
  このセクションは、RAM 上に配置する必要があります。

- **.EXT_USR_STACK セクション**
  外部 RAM に割り当てたユーザスタック領域です。
  このセクションは、RAM 上に配置する必要があります。

- **.MR_KERNEL・.MR_KERNEL2 セクション**
  MR32R カーネルが配置されます。

- **.MR_RAM セクション**
  MR32R が使用する内蔵 RAM データが配置されます。
  このセクションは、RAM 上に配置する必要があります。

- **.EXT_MR_RAM セクション**
  MR32R が使用する外部 RAM データが配置されます。
  このセクションは、RAM 上に配置する必要があります。

- **.MR_ROM セクション**
  MR32R が使用する固定データが配置されます。
  動的生成機能を使用する場合は、ROM から RAM へ転送して使用します。

- **.MR_HEAP セクション**
  MR32R が使用する内蔵 RAM のヒープ領域です。固定長および可変長メモリプール機能を使用する場合、このセクションを使用します。
  このセクションは、RAM 上に配置する必要があります。

- **.EXT_MR_HEAP セクション**
  MR32R が使用する外部 RAM ヒープ領域です。固定長および可変長メモリプール機能を使用する場合、このセクションを使用します。
  このセクションは、RAM 上に配置する必要があります。

● **割り込みベクタ関連のセクション**

- **.Int_Vector セクション**

- **.EIT_Vector セクション**
  EIT ベクタ領域を格納するセクションです。

- **.RESET_VECT セクション**
  リセットベクタ領域です。

- **.INTERRUPT_VECT セクション**
  コンフィグレータにより出力された割り込みベクタテーブルが格納されているセクションです。
  def_int システムコール使用時は、ROM から RAM へ転送して使用します

6.10.3 メモリモデルについて

MR32R では以下のような二種類のメモリモデルがあります。

- ラージモデル
  MR32R のカーネル領域やデータ領域を 16MB を超える空間に配置可能なモデルです。M3T-CC32R 対応カーネルのみラージモデルを用意しています。ラージモデル対応時のセクション配置については、下記の制限があります。

  1. EVB(EIT ベクタベースレジスタ)をサポートしていないもしくは、16MB を超える領域に EVB を設定できないマイコンでは、6.10.4に示す処理が必要です。

- 非ラージモデル
  MR32R のカーネル領域やデータ領域の配置について以下の制限があります。

  16MB を越えた空間に配置可能な領域

  INT_USR_STACK, EXT_USR_STACK, SYS_STACK セクション
  EXT_MR_RAM, MR_HEAP, EXT_MR_HEAP セクション

  (b)16MB を越えた空間に配置不可能な領域

  MR_RAM, MR_ROM, INTERRUPT_VECTOR, MR_Dbg_RAM セクション
  MR_KERNEL,MR_KERNEL2 セクション

  ※ EVB(EIT ベクタベースレジスタ)をサポートしていないもしくは、16MB を超える領域に EVB を設定できないマイコンでは、6.10.4に示す処理が必要です。
  ※ MR_KERNEL と MR_KERNEL2 は相対 16MB 空間内に配置しなければいけません。

  アプリケーションが使用するコード領域(P セクションや.text セクションなど)やデータ領域(B,D,C セクションや.bss,.data,.rodata セクションなど)の配置に関しては,ご使用になるコンパイラに依存します。また,コンパイラオプションの変更・リンクする標準ライブラリの変更・標準ライブラリの再構築が必要となる場合があります。詳細については,ご使用のコンパイラのマニュアル等ドキュメントを参照ください。

6.10.4 カーネル領域の 16MB を超える空間への配置について

MR32R では、システムコールの呼び出しに TRAP 命令を使用しています。しかし、TRAP は、4 バイトのエントリしかないため、直接 16MB を超える空間にジャンプする事ができません。従って、EVB(EIT ベクタベースレジスタ)をサポートしていないもしくは、16MB を超える領域に EVB を設定できないマイコンでは、①いったん 16MB 空間内にジャンプし、そこから 16MB を越える空間に jmp,jl 命令によってジャンプする必要があります。また、これらの命令を使用するにはレジスタを１つ使用しますので、②このレジスタの退避・復帰処理が必要となります。

MR32R カーネル領域(MR_KERNEL,MR_KERNEL2 セクション)を 16MB を越える空間に配置するためには、上記①②の処理が必要となります。H'2000000 番地以降に MR32R カーネル領域を配置した場合の TRAP7 使用システムコール処理の例を下図に示します。

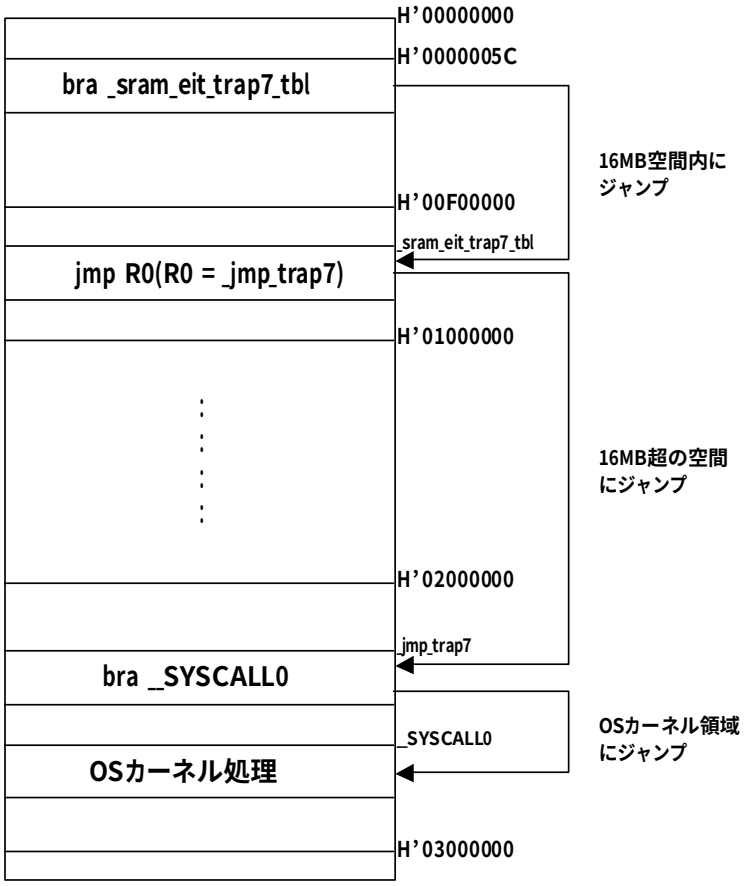


図 6.3 OS カーネルの 16MB 超領域への配置

以下に詳細手順を示します。

(1)スタートアップルーチンを変更し、TRAP,EI ベクタのジャンプ先を変更します。

[M3T-CC32R 記述例]

```
.global _sram_eit_trap7_tbl,_sram_eit_trap8_tbl
                :
bra     _sram_eit_trap7_tbl             ; TRAP7
bra     _sram_eit_trap8_tbl             ; TRAP8
                :
.section        Int_Vector,code,align=4
st      r0,@-r15
seth    r0,#high(_jmp_ei)
or3     r0,r0,#low(_jmp_ei)
jmp     r0
```

(2)スタートアップルーチンに 2 段目ジャンプ用のコードを追記します。このセクションは、16MB 空間内に配置します。

[M3T-CC32R 記述例]

```
.section        ROM_MR_EIT,code,align=4
.section        MR_EIT,code,align=4
.export _sram_eit_trap7_tbl
_sram_eit_trap7_tbl:
st      r0,@-r15
seth    r0,#high(_jmp_trap7)
or3     r0,r0,#low(_jmp_trap7)
jmp     r0

.export _sram_eit_trap8_tbl
_sram_eit_trap8_tbl:
st      r0,@-r15
seth    r0,#high(_jmp_trap8)
or3     r0,r0,#low(_jmp_trap8)
jmp     r0
```

(3)スタートアップルーチンに 3 段目ジャンプ用のコードを追記します。このセクションは、OS カーネル領域と同じセクション(MR_KERNEL)とします。

[M3T-CC32R 記述例]

```
.section        MR_KERNEL
_jmp_trap7:
.AIF    ¥&__Dbg_flg eq 0
.global __SYSCALL0
ld      r0,@r15+
bra     __SYSCALL0
.AELSE
.global __Dbg_entry0
ld      r0,@r15+
bra __Dbg_entry0
.AENDI

_jmp_trap8:
.AIF    ¥&__Dbg_flg eq 0
.global __SYSCALL1
ld      r0,@r15+
bra __SYSCALL1
.AELSE
.global __Dbg_entry1
ld      r0,@r15+
bra __Dbg_entry1
.AENDI

_jmp_ei:
.global __int_entry
ld      r0,@r15+
bra __int_entry
```

第 7 章　　コンフィグレータの使用方法

7.1 コンフィグレーションファイルの作成方法

アプリケーションプログラムのコーディング、スタートアッププログラムの修正が終わると、そのアプリケーションプログラムを MR32R システムに登録する必要があります。これをおこなうのがコンフィグレーションファイルです。

7.1.1 コンフィグレーションファイル内の表現形式

この節ではコンフィグレーションファイル内における定義データの表現形式について説明します。

コメント文

'//'から行の終わりまではコメント文とみなし、処理の対象になりません。

文の終わり

';'で文を終ります。

数値

数値は以下の形式で入力できます。

1. 16 進数

   数値の先頭に'0x'か'0X'を付加します。または、数値の最後に'h'か'H'を付加します。後者の場合でかつ先頭が英文字 (A～F)で始まる場合は先頭に必ず'0'を付加してください。なおここで使用する数値表現で英文字 (A～F)は大文字・小文字を識別しません。[65]

2. 10 進数

   23 のように整数のみで表します。ただし'0'で始めることはできません。

3. 8 進数

   数値の先頭に'0'を付加するか数値の最後に'O'もしくは'o'を付加します。

4. 2 進数

   数値の最後に'B'または'b'を付加します。ただし'0'で始めることはできません。

表 7-1 数値表現例

16 進数	0xf12 0Xf12 0a12h 0a12H 12h 12H
10 進数	32
8 進数	017 17o 17O
2 進数	101110b 101010B

また数値内に演算子を記述できます。使用できる演算子を表 7-2に示します。

[65] 数値表現内の'A'～'F','a'～'f'を除いて全ての文字は、大文字・小文字の区別を行います。

表 7-2　演算子

演算子	優先度	演算方向
()	高	左から右
- (単項マイナス)		右から左
* / %		左から右
+ - (二項マイナス)	低	左から右

数値の例を以下に示します。

- 123
- 123 + 0x23
- (234 + 3) * 2
- 100B + 0aH

シンボル

シンボルは数字、英大文字、英小文字、'_'（アンダースコア)、'?'より構成される数字以外の文字で始まる文字列で表されます。
シンボルの例を以下に示します。

- _TASK1
- IDLE3

関数名

関数名は数字、英大文字、英小文字、'_' (アンダースコア)、'$'(ドル記号)より構成される数字以外の文字で始まり、' ()'で終わる文字列で表されます。
C 言語で記述した関数名の例を以下に示します。

- main()
- func()

アセンブリ言語で記述した場合はモジュールの先頭ラベルを関数名とします。

周波数

周波数は数字と'.'(ピリオド) から構成され'MHz'で終わる文字列で表されます。小数点以下は 6 桁が有効です。なお周波数は 10 進数のみで記述可能です。
周波数の例を以下に示します。

- 16MHz
- 8.1234MHz

なお、周波数は'.'で始まってはいけません。

時間

時間は数字と'. ' (ピリオド) から構成され'ms'または's'で終わる文字列で表されます。有効桁数は'ms'の場合小数点以下 3 桁です。's'の場合は小数点以下 6 桁です。なお時間は 10 進数のみで記述可能です。

時間の例を以下に示します。

- 0.23s
- 10ms
- 10.5ms

なお時間は'.'(ピリオド)で始まってはいけません。

時刻

時刻は 3 つの 16 ビットの数値を':'でつないだ形で表現される 48 ビットのデータです。たとえば、

- 23:0x021:00B

なお、3 つの数字の内、上位の 1 つもしくは上位の 2 つを省略した場合はその位置の数字は 0 とみなされます。すなわち、'12' は'0:12'と等価です。

7.1.2 コンフィグレーションファイルの定義項目

コンフィグレーションファイルでは以下の項目[66]の定義をおこないます。

- システム定義
- システムクロック定義
- 最大項目定義
- 動的生成用ユーザースタック領域定義
- 動的生成用メールボックス領域定義
- 動的生成用メッセージバッファ領域定義
- 動的生成用固定長メモリプール領域定義
- 動的生成用可変長メモリプール領域定義
- タスク定義
- イベントフラグ定義
- セマフォ定義
- メイルボックス定義
- メッセージバッファ定義
- ランデブ定義
- 優先度付きメイルボックス定義
- 固定長メモリープール定義
- 可変長メモリープール定義
- 周期起動ハンドラ定義
- アラームハンドラ定義
- 割り込みベクタ定義

[66] タスク定義以外の項目は、省略することができます。省略した場合にはデフォルトコンフィグレーションファイルの定義が参照されます。

● システム定義

<< 形式 >>

```
// System Definition
system{
    stack_size      = システムスタックサイズ ;
    priority        = 優先度の最大値 ;
    exc_handler     = 例外ハンドラ ;
    debug           = デバッグ機能;
    debug_buffer    = デバッグ機能で使用するバッファのサイズ;
 };
```

<< 内容 >>

1. システムスタックサイズ

【 定義形式 】数値

【 定義範囲 】1 以上

システムコール処理および割り込み処理で使用するスタックサイズの合計を定義します。

2. 優先度の最大値（最低優先度の値）

【 定義形式 】数値

【 定義範囲 】1 ～ 255

MR32R のアプリケーションプログラムの使用する優先度の最大値を定義します。すなわち使用している優先度の最も大きい値を設定してください。[67]

3. 例外ハンドラ

【 定義形式 】シンボル

【 定義範囲 】YES or NO

ユーザアプリケーションプログラムで例外ハンドラ(強制例外ハンドラ)を定義している場合は、YES を設定し、定義していない場合は、NO を設定してください。

4. デバッグ機能

【 定義形式 】シンボル

【 定義範囲 】YES or NO

OS デバッグ機能に対応したデバッガ(PD32R など)で MR トレース機能、システムコール発行機能を使用する場合は YES、使用しない場合は、NO を指定します。YES を指定することにより、OS デバッグ機能を実現するためのフックルーチンを呼び出します。

[67] **MR32R** の優先度は、値が大きいほど優先度は低くなります。

5. デバッグ機能に使用するバッファサイズ

【 定義形式 】数値

【 定義範囲 】0 以上(4 の倍数)

デバッグ機能で使用するバッファのサイズを指定します。指定したサイズは 4 の倍数に丸められます。4096 バイト確保した場合、少なくとも約 60 回分のタスク切り替えがトレースできます。

● 動的生成用ユーザスタック領域定義(内蔵 RAM)

<< 形式 >>

```
int_memstk{
    max_memsize     = 生成するタスクのスタックサイズの最大値;
    all_memsize     = タスク生成用ユーザースタック領域(内蔵RAM)のサイズ;
};
```

<< 内容 >>

1. 生成するタスクのスタックサイズの最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_tsk システムコールの発行により生成するタスクあるいは def_exc により定義した例外ハンドラの中でスタックサイズが最も大きい値を指定します。ここでの指定は、スタックに使用する領域を内蔵 RAM に指定したタスクの中で、最も大きいサイズを指定してください。

2. タスク生成用ユーザースタック領域(内蔵 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

タスク生成用ユーザースタック領域のサイズを指定してください。
ここで指定されたサイズの領域は、タスク生成時に必要となるタスクのスタック領域(内蔵 RAM)を確保するために必要となります。
cre_tsk システムコールで指定されたスタックサイズは、このタスク生成用ユーザースタック領域から指定サイズ分のメモリを確保します。

● 動的生成用ユーザスタック領域定義(外部 RAM)

<< 形式 >>

```
ext_memstk{
    max_memsize     = 生成するタスクのスタックサイズの最大値;
    all_memsize     = タスク生成用ユーザースタック領域(外部RAM)のサイズ;
};
```

<< 内容 >>

1. 生成するタスクのスタックサイズの最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_tsk システムコールの発行により生成するタスクあるいは def_exc により定義した例外ハンドラの中でスタックサイズが最も大きい値を指定します。ここでの指定は、スタックに使用する領域を外部 RAM に指定したタスクの中で、最も大きいサイズを指定してください。

2. ユーザースタック領域(外部 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

タスク生成用ユーザースタック領域のサイズを指定してください。
ここで指定されたサイズの領域は、タスク生成時にタスクのスタック領域(外部 RAM)を確保するために必要となります。
cre_tsk システムコールで指定されたスタックサイズは、このタスク生成用ユーザースタック領域から指定サイズ分のメモリを確保します。

● 動的生成用メールボックス領域定義(内蔵 RAM)

<< 形式 >>

```
int_memmbx{
    max_memsize     = 生成するメールボックスサイズの最大値;
    all_memsize     = 動的生成用メールボックス領域(内蔵RAM)のサイズ;
};
```

<< 内容 >>

1. 生成するメールボックスサイズの最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_mbx システムコールの発行により生成するメールボックスの中でサイズが最も大きい値を指定します。ここでの指定は、内蔵 RAM を指定したメールボックスの中で、最も大きいサイズを指定してください。

2. 動的生成用メールボックス領域(内蔵 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

動的生成用メールボックス領域のサイズを指定してください。
ここで指定されたサイズの領域は、cre_mbx システムコール発行時にメールボックス領域(内蔵 RAM)を確保するために必要となります。
cre_mbx システムコールで指定されたメールボックスのサイズは、この動的生成用メール

ボックス領域から指定サイズ分のメモリを確保します。

● 動的生成用メールボックス領域定義(外部 RAM)

<< 形式 >>

```
ext_memmbx{
    max_memsize     = 生成するメールボックスサイズの最大値;
    all_memsize     = メールボックス領域(外部RAM)のサイズ;
};
```

<< 内容 >>

1. 生成するメールボックスサイズの最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_mbx システムコールの発行により生成するメールボックスの中でサイズが最も大きい値を指定します。ここでの指定は、外部 RAM を指定したメールボックスの中で、最も大きいサイズを指定してください。

2. メールボックス領域(外部 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

動的生成用メールボックス領域のサイズを指定してください。
ここで指定されたサイズの領域は、cre_mbx システムコール発行時にメールボックス領域(外部 RAM)を確保するために必要となります。
cre_mbx システムコールで指定されたメールボックスのサイズは、この動的生成用メールボックス領域から指定サイズ分のメモリを確保します。

● 動的生成用メッセージバッファ領域定義(内蔵 RAM)

<< 形式 >>

```
int_memmbf{
    max_memsize     = 生成するメッセージバッファサイズの最大値;
    all_memsize     = 動的生成用メッセージバッファ領域(内蔵RAM)のサイズ;
};
```

<< 内容 >>

1. 生成するメッセージバッファサイズの最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_mbf システムコールの発行により生成するメッセージバッファの中でサイズが最も大きい値を指定します。ここでの指定は、内蔵 RAM を指定したメッセージバッファの中で、最

も大きいサイズを指定してください。

2. メッセージバッファ領域(内蔵 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

動的生成用メッセージバッファ領域のサイズを指定してください。
ここで指定されたサイズの領域は、cre_mbf システムコール発行時にメッセージバッファ領域(内蔵 RAM)を確保するために必要となります。
cre_mbf システムコールで指定されたメッセージバッファのサイズは、この動的生成用メッセージバッファ領域から指定サイズ分のメモリを確保します。

● 動的生成用メッセージバッファ領域定義(外部 RAM)

<< 形式 >>

```
ext_memmbf{
    max_memsize     = 生成するメッセージバッファサイズの最大値;
    all_memsize     = 動的生成用メッセージバッファ領域(外部RAM)のサイズ;
};
```

<< 内容 >>

1. 生成するメッセージバッファサイズの最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_mbf システムコールの発行により生成するメッセージバッファの中でサイズが最も大きい値を指定します。ここでの指定は、外部 RAM を指定したメッセージバッファの中で、最も大きいサイズを指定してください。

2. メッセージバッファ領域(外部 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

動的生成用メッセージバッファ領域のサイズを指定してください。
ここで指定されたサイズの領域は、cre_mbf システムコール発行時にメッセージバッファ領域(外部 RAM)を確保するために必要となります。
cre_mbf システムコールで指定されたメッセージバッファのサイズは、この動的生成用メッセージバッファ領域から指定サイズ分のメモリを確保します。

● 動的生成用固定長メモリプール領域定義(内蔵 RAM)

<< 形式 >>

```
int_memmpf{
    max_memsize     = 生成する固定長メモリプール領域の最大値;
    all_memsize     = 動的生成用固定長メモリプール領域(内蔵RAM)のサイズ;
};
```

<< 内容 >>

1. 生成する固定長メモリプール領域の最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_mpf システムコールの発行により生成する固定長メモリプールの中でサイズが最も大きい値を指定します。ここでの指定は、内蔵 RAM を指定した固定長メモリプールの中で、最も大きいサイズを指定してください。

2. 動的生成用固定長メモリプール領域(内蔵 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

動的生成用固定長メモリプール領域のサイズを指定してください。
ここで指定されたサイズの領域は、cre_mpf システムコール発行時に固定長メモリプール領域(内蔵 RAM)を確保するために必要となります。
cre_mpf システムコールで指定された固定長メモリプールのサイズは、この動的生成用固定長メモリプール領域から指定サイズ分のメモリを確保します。

● 動的生成用固定長メモリプール領域定義(外部 RAM)

<< 形式 >>

```
ext_memmpf{
    max_memsize     = 生成する固定長メモリプール領域の最大値;
    all_memsize     = 動的生成用固定長メモリプール領域(外部RAM)のサイズ;
};
```

<< 内容 >>

1. 生成する固定長メモリプール領域の最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_mpf システムコールの発行により生成する固定長メモリプールの中でサイズが最も大きい値を指定します。ここでの指定は、外部 RAM を指定した固定長メモリプールの中で、最も大きいサイズを指定してください。

2. 動的生成用固定長メモリプール領域(外部 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

動的生成用固定長メモリプール領域のサイズを指定してください。
ここで指定されたサイズの領域は、cre_mpf システムコール発行時に固定長メモリプール領域(外部 RAM)を確保するために必要となります。
cre_mpf システムコールで指定された固定長メモリプールのサイズは、この動的生成用固定長メモリプール領域から指定サイズ分のメモリを確保します。

● 動的生成用可変長メモリプール領域定義(内蔵 RAM)

<< 形式 >>

```
int_memmpl{
    max_memsize     = 生成する可変長メモリプール領域の最大値;
    all_memsize     = 動的生成用可変長メモリプール領域(内蔵RAM)のサイズ;
};
```

<< 内容 >>

1. 生成する可変長メモリプール領域の最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_mpl システムコールの発行により生成する可変長メモリプールの中でサイズが最も大きい値を指定します。ここでの指定は、内蔵 RAM を指定した可変長メモリプールの中で、最も大きいサイズを指定してください。

2. 動的生成用可変長メモリプール領域(内蔵 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

動的生成用可変長メモリプール領域のサイズを指定してください。
ここで指定されたサイズの領域は、cre_mpf システムコール発行時に可変長メモリプール領域(内蔵 RAM)を確保するために必要となります。
cre_mpf システムコールで指定された可変長メモリプールのサイズは、この動的生成用可変長メモリプール領域から指定サイズ分のメモリを確保します。

● 動的生成用可変長メモリプール領域定義(外部 RAM)

<< 形式 >>

```
ext_memmpl{
    max_memsize     = 生成する可変長メモリプール領域の最大値;
    all_memsize     = 動的生成用可変長メモリプール領域(外部RAM)のサイズ;
};
```

<< 内容 >>

1. 生成する可変長メモリプール領域の最大値

【 定義形式 】数値

【 定義範囲 】1 以上

cre_mpl システムコールの発行により生成する可変長メモリプールの中でサイズが最も大きい値を指定します。ここでの指定は、外部 RAM を指定した可変長メモリプールの中で、最も大きいサイズを指定してください。

2. 動的生成用可変長メモリプール領域(外部 RAM)のサイズ

【 定義形式 】数値

【 定義範囲 】24 以上

動的生成用可変長メモリプール領域のサイズを指定してください。
ここで指定されたサイズの領域は、cre_mpl システムコール発行時に可変長メモリプール領域(外部 RAM)を確保するために必要となります。
cre_mpl システムコールで指定された可変長メモリプールのサイズは、この動的生成用可変長メモリプール領域から指定サイズ分のメモリを確保します。

● システムクロック定義

<< 形式 >>

```
// System Clock Definition
clock{
    timer_clock     = タイマのクロック;
    timer           = タイマモード;
    IPL             = システムクロック割り込み優先レベル;
    unit_time       = システムクロックの単位時間;
    initial_time    = システム時刻の初期値;
    file_name       = タイマ設定用テンプレートファイル名;
};
```

<< 内容 >>

1. タイマのクロック

【 定義形式 】周波数 (MHz)

【 定義範囲 】なし

タイマに供給されるクロックの周波数を MHz 単位で定義します。
従来のバージョンとの互換性のために、mpu_clock も使用することができます。

2. タイマモード

【 定義形式 】シンボル

【 定義範囲 】MSA2000, OTHER, NOTIMER, その他シンボル

タイマのモードを MSA2000、OTHER、NOTIMER、その他のシンボルで定義します。

MSA2000 は、M3A-2000G01 のタイマ 0 を使用する場合に、システムクロックを使用しない場合は、"NOTIMER"を定義します。ここで定義されたシンボルは、"mr32r.inc"ファイルにシンボルとして定義され、"timer.inc"のタイマの初期設定部分でマクロ展開される際に使用されます。

指定可能なシンボルは、表 7-3を参照ください。

3. システムクロック割り込み優先レベル

【 定義形式 】数値

【 定義範囲 】0 ～ 6

システムクロック用タイマ割り込みの優先レベルを定義します。

システムクロックの割り込みハンドラ処理中は、ここで定義した割り込みレベルより低いレベルの割り込みは受け付けられません。

4. システムクロックの単位時間

【 定義形式 】時間 (ms 単位)

【 定義範囲 】0.1ms 以上

システムクロックの単位時間 (システムクロック割り込み発生間隔)を ms 単位で定義します。

5. システム時刻の初期値

【 定義形式 】時刻

【 定義範囲 】0 : 0 : 0 ～ 0x7FFF : 0xFFFF : 0xFFFF

システム時刻の初期値を定義します。システム時刻を用いた機能(set_tim、get_tim、アラームハンドラ)を使用しない場合は、この項目を設定する必要はありません。本定義をおこなわないことにより自動的にシステムクロック割り込みハンドラの処理が最適化されます。ただし、デフォルトコンフィグレーションファイルでデフォルト値を定義した場合、最適化は行いませんので注意してください。

6. テンプレートファイル名

【 定義形式 】シンボル

【 定義範囲 】なし

使用するテンプレートファイル名を指定します。指定したファイル名のファイルがコンフィグレータ実行ディレクトリに"timer.inc"というファイル名でコピーされます。標準添付のテンプレートファイルとマイコン・周辺機能との対応は、以下のとおりです。また、下表の定義可能なシンボルは、各マイコン・周辺のタイマの名称と同じです。

表 7-3　使用マイコンとテンプレートファイル名との対応

使用マイコンまたは周辺	テンプレートファイル名	定義可能なシンボル
M32120	m32120.tpl	MFT00,MFT01…MFT13
M32160	m32160.tpl	TOP0,TOP1,TOP2…・TOP5
M32180	M32180.tpl	TOP0,TOP1,TOP2, …TOP10
Hint 社 HM1	hint1.tpl	T0,T1
M32104	M32104.tpl	MFT00,MFT01,MFT02
M32102	m32102.tpl	MFT00,MFT01…MFT13

● **最大項目数定義**

ここでの設定は、複数のアプリケーションの中で、各定義の最大数を定義します[68]。

<< 形式 >>

```
// Max Definition
maxdefine{
    max_task        = 最大タスク定義数;
    max_flag        = 最大イベントフラグ定義数;
    max_mbx         = 最大メイルボックス定義数;
    max_sem         = 最大セマフォ定義数;
    max_mbf         = 最大メッセージバッファ定義数;
    max_mpf         = 最大固定長メモリプール定義数;
    max_mpl         = 最大可変長メモリプール定義数;
    max_por         = 最大ランデブ定義数;
    max_vmbx        = 最大優先度付きメイルボックス定義数;
    max_cyh         = 最大周期起動ハンドラ定義数;
    max_alh         = 最大アラームハンドラ定義数;
    max_int         = 最大割り込みハンドラ定義数;
};
```

[68] タスクやオブジェクトを生成するシステムコール(cre_xxx)をご使用の場合、必ず、生成するタスクやオブジェクトの最大数を定義してください。

<< 内容 >>

1. 最大タスク定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 32767

タスク定義の最大数を定義します。

2. 最大イベントフラグ定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 32766

イベントフラグ定義の最大数を定義します。

3. 最大メイルボックス定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 32766

メイルボックス定義の最大数を定義します。

4. 最大セマフォ定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 32766

セマフォ定義の最大数を定義します。

5. 最大メッセージバッファ定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 32765

メッセージバッファ定義の最大数を定義します。

6. 最大固定長メモリプール定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 32766

固定長メモリプール定義の最大数を定義します。

7. 最大可変長メモリプール定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 32766

可変長メモリプール定義の最大数を定義します。

8. 最大ランデブ定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 32765

ランデブ定義の最大数を定義します。

9. 最大優先度付きメイルボックス定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 32765

優先度付きメイルボックス定義の最大数を定義します。

10. 最大周期起動ハンドラ定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 32767

周期起動ハンドラ定義の最大数を定義します。

11. 最大アラームハンドラ定義

【 定義形式 】数値

【 定義範囲 】1 ～ 32767

アラームハンドラ定義の最大数を定義します。

12. 最大割り込みハンドラ定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 255

割り込みハンドラ定義の最大数を定義します。コンフィグレータによってここに指定した値の分だけ割り込みベクタテーブルが確保されます。

● タスク定義

<< 形式 >>

```
 // Tasks Definition
task[ID番号]{
    entry_address   = タスクの開始アドレス;
    stack_size      = タスクのユーザースタックサイズ;
    stack_area      = スタックの配置;
    stack_section   = スタックのセクション名;
    priority        = タスクの初期優先度;
    initial_start   = 初期起動状態;
};
                :
                :
```

ID 番号は 1～32767 の範囲でなければなりません。ID 番号は省略可能です。

省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>
タスク ID 番号ごとに以下の定義をおこないます。

1. タスク開始アドレス

【 定義形式 】シンボル、または、関数名

【 定義範囲 】なし

タスクの入り口アドレスを定義します。

2. ユーザースタックサイズ

【 定義形式 】数値

【 定義範囲 】80 以上

タスクごとのユーザースタックサイズを定義します。ユーザースタックとは、各々のタスクが使用するスタック領域です。MR32R ではユーザー用のスタック領域をタスクごとに割り当てる必要があり最低で 80 バイトが必要です。
また、本定義によるタスクに対して例外ハンドラを定義する場合は、96 バイト分のスタックがさらに必要になります。
ここで指定した値は、4 の倍数に丸めて割り当てられます。

3. スタックの配置

【 定義形式 】シンボル

【 定義範囲 】__MR_INT or __MR_EXT

ユーザースタックの配置を指定します。ユーザースタックを内蔵 RAM に配置するか、外部 RAM に配置するかを指定します。

★ 内蔵 RAM に配置する場合
__MR_INT または INTERNAL を指定してください。。

★ 外部 RAM に配置する場合
__MR_EXT または EXTERNAL を指定してください。

本項目を省略した場合は、__MR_INT が設定されます。
なお、4の項目 stack_section の指定を行う場合は、本項目の指定は省略してください。

4. スタックのセクション名

【 定義形式 】シンボル

【 定義範囲 】なし

タスクのスタックを配置するセクション名を指定します。ここで定義したセクションは、必ず、セクションファイルにて配置を行って下さい。
なお、3の項目 stack_area の指定を行う場合、本項目の指定は省略してください。

5. タスクの初期優先度

【 定義形式 】数値

【 定義範囲 】1 ～ (システム定義の優先度の最大値)

タスクの起動時の優先度を定義します。
MR32R の優先度は、値が小さいほど、優先度としては高くなります。

6. 初期起動状態

【 定義形式 】シンボル

【 定義範囲 】ON or OFF

タスクの初期起動状態を定義します。
ON を指定すると、システムの初期起動時に READY 状態になります。

● イベントフラグ定義

<< 形式 >>

```
// Event Flag Definition
flag[ID番号]{
   name            = 名前;
};
            :
            :
```

ID 番号は 1～32766 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

イベントフラグ ID 番号ごとに以下の定義をおこないます。

1. 名前

【 定義形式 】シンボル

【 定義範囲 】なし

プログラム中でイベントフラグを指定する時の名前を定義します。

● セマフォ定義

<< 形式 >>

```
// Semaphore Definition
semaphore[ID番号]{
    name            = 名前;
    max_count       = セマフォのカウンタ最大値;
    initial_count   = セマフォのカウンタ初期値;
};
            :
            :
```

ID 番号は 1～32766 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

セマフォ ID 番号ごとに以下の定義をおこないます。

1. **名前**

【 定義形式 】シンボル

【 定義範囲 】なし

プログラム中でセマフォを指定する時の名前を定義します。

2. **セマフォカウンタ最大値**

【 定義形式 】数値

【 定義範囲 】0 ～ 0x7FFFFFFF

セマフォカウンタの最大値を定義します。

3. **セマフォカウンタ初期値**

【 定義形式 】数値

【 定義範囲 】0 ～ セマフォカウンタ最大値

セマフォカウンタの初期値を定義します。

● メイルボックス定義

<< 形式 >>

```
// Mailbox Definition
mailbox[ID番号]{
    name            = 名前;
    mbx_area        = メイルボックスの配置;
    buffer_size     = メイルボックスの最大メッセージ数;
};
            :
            :
```

ID 番号は 1～32766 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

メイルボックス ID 番号ごとに以下の項目の定義をおこないます。

1. 名前

【 定義形式 】シンボル

【 定義範囲 】なし

プログラム中でメイルボックスを指定する時の名前を定義します。

2. メイルボックスの配置

【 定義形式 】シンボル

【 定義範囲 】__MR_INT or __MR_EXT

メイルボックスの配置を指定します。メイルボックスを内蔵 RAM に配置するか、外部 RAM に配置するかを指定します。

★ 内蔵 RAM に配置する場合
__MR_INT または INTERNAL を指定してください。。

★ 外部 RAM に配置する場合
__MR_EXT または EXTERNAL を指定してください。。

本項目を省略した場合は、__MR_INT が設定されます。

3. 最大メッセージ数

【 定義形式 】数値

【 定義範囲 】1 以上

メイルボックスに蓄えることのできる最大メッセージ数を定義します。これを越えてメッセージを蓄えようとするとエラーがかえってきます。

● メッセージバッファ定義

<< 形式 >>

```
 // MessageBuffer Definition
message_buffer[ID番号]{
    name            = 名前;
    mbf_area        = メッセージバッファの配置;
    buffer_size     = メッセージバッファのサイズ;
};
            :
            :
```

ID 番号は 1～32765 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

メッセージバッファ ID 番号ごとに以下の項目の定義をおこないます。

1. 名前

【 定義形式 】シンボル

【 定義範囲 】なし

プログラム中でメッセージバッファを指定する時の名前を定義します。

2. メッセージバッファの配置

【 定義形式 】シンボル

【 定義範囲 】__MR_INT or __MR_EXT

メッセージバッファの配置を指定します。メッセージバッファを内蔵 RAM に配置するか、外部 RAM に配置するかを指定します。

★ 内蔵 RAM に配置する場合

__MR_INT または INTERNAL を指定してください。。

★ 外部 RAM に配置する場合

__MR_EXT または EXTERNAL を指定してください。。

本項目を省略した場合は、__MR_INT が設定されます。

3. メッセージバッファサイズ

【 定義形式 】数値

【 定義範囲 】0 以上

メッセージバッファのサイズを指定します。
なお、ここで指定した値は、4 の倍数に丸めて割り当てられます。

● 優先度付きメイルボックス定義

<< 形式 >>

```
// VMailbox Definition
vmailbox[ID番号]{
    name            = 名前;
    wait_queue      = メッセージを受信する順序;
    message_queue   = メッセージキューへのつなぎ方;
    maxpri          = メッセージの最大優先度;
};
          :
          :
```

<< 内容 >>

4. ID 番号

【 定義形式 】数値

【 定義範囲 】1～32765
ID 番号は 1～32765 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

5. 名前

【 定義形式 】シンボル

【 定義範囲 】なし

プログラム中でメイルボックスを指定する時の名前を定義します。

6. メッセージを受信する順序

【 定義形式 】シンボル

【 定義範囲 】FIFO or PRI

メッセージを受信する順序を指定します。

★ FIFO
最初にメッセージを待ったタスクから順にメッセージを受信します。

★ PRI
優先度の高いタスクから順にメッセージを受信します。

7. メッセージキューへのつなぎ方

【 定義形式 】シンボル

【 定義範囲 】FIFO or PRI

★ FIFO
最初に送信したメッセージから順に受信されます。

★ PRI
優先度の高いメッセージから順に受信されます。

8. メッセージの最大優先度

【 定義形式 】数値

【 定義範囲 】1～255

メッセージの最大優先度を指定します。指定した優先度を超える値のメッセージを使用してもエラーとはなりませんので注意してください。

● ランデブ定義

<< 形式 >>

```
// Rendezvous Definition
rendezvous[ID番号]{
   name            = 名前;
};
          :
          :
```

ID 番号は 1～32765 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

ランデブ用ポート ID 番号ごとに以下の定義をおこないます。

1. 名前

【 定義形式 】シンボル

【 定義範囲 】なし

プログラム中でランデブ用ポートを指定する時の名前を定義します。

● 固定長メモリプール定義

<< 形式 >>

```
// Fixed Memory Pool Definition
memorypool[ID番号]{
    name            = 名前;
    mpf_area        = 固定長メモリプールの配置;
    section         = セクション名;
    num_block       = メモリプールのブロック数;
    siz_block       = メモリプールのブロックサイズ;
};
            :
            :
```

ID 番号は、1～32766 の範囲でなければなりません。ID 番号は、省略可能です。
省略した場合は番号を小さい方から順に自動的に割り当てます。

<< 内容 >>

メモリプール ID 番号ごとに以下の定義をおこないます。

1. 名前

【 定義形式 】シンボル

【 定義範囲 】なし

プログラム中でメモリプールを指定する時の名前を指定します。

2. 固定長メモリプールの配置

【 定義形式 】シンボル

【 定義範囲 】__MR_INT or __MR_EXT

固定長メモリプールの配置を指定します。固定長メモリプールを内蔵 RAM に配置するか、外部 RAM に配置するかを指定します。

★ 内蔵 RAM に配置する場合

__MR_INT または INTERNAL を指定してください。。

★ 外部 RAM に配置する場合

__MR_EXT または EXTERNAL を指定してください。。

本項目を省略した場合は、__MR_INT が設定されます。
なお、3の項目 section の指定を行う場合、本項目の指定は、行わないでください。

3. セクション名

【 定義形式 】シンボル

【 定義範囲 】なし

メモリプールを配置するセクションの名前を定義します。ここで定義したセクションは、必ず、セクションファイルにて配置を行って下さい。
なお、2の項目 mpf_area の指定を行う場合、本項目の指定は、行わないでください。

4. ブロック数

【 定義形式 】数値

【 定義範囲 】1～32

メモリプールのブロック総数を定義します。

5. サイズ

【 定義形式 】数値

【 定義範囲 】1 以上

メモリプールの 1 ブロック当たりのサイズを定義します。この定義によりメモリプールとして使用する RAM 容量は、 (ブロック数)×(サイズ)バイトです。

● 可変長メモリプール定義

<< 形式 >>

```
// Variable-Size Memory Pool Definition
 variable_memorypool[ID番号]{
    name            = 名前;
    mpl_area        = 可変長メモリプールの配置;
    max_memsize     = 確保するメモリブロックサイズの最大値;
    heap_size       = メモリプールのサイズ;
 };
            :
            :
```

ID 番号は、1～32766 の範囲でなければなりません。ID 番号は、省略可能です。
省略した場合は番号を小さい方から順に自動的に割り当てます。

<< 内容 >>

1. 名前

【 定義形式 】シンボル

【 定義範囲 】なし

プログラム中でメモリプールを指定する時の名前を指定します。

2. 確保するメモリブロックサイズの最大値

【 定義形式 】数値

【 定義範囲 】0 以上

アプリケーションプログラム中で、確保しているメモリブロックサイズの最大値を指定します。

3. 可変長メモリプールの配置

【 定義形式 】シンボル

【 定義範囲 】__MR_INT or __MR_EXT

可変長メモリプールの配置を指定します。可変長メモリプールを内蔵 RAM に配置するか、外部 RAM に配置するかを指定します。

★ 内蔵 RAM に配置する場合

__MR_INT または INTERNAL を指定してください。。

★ 外部 RAM に配置する場合

__MR_EXT または EXTERNAL を指定してください。。

本項目を省略した場合は、__MR_INT が設定されます。

4. メモリプールのサイズ

【 定義形式 】数値

【 定義範囲 】192 以上

メモリプールのサイズを指定します。ここで、指定された数値は、4 の倍数に丸めて、割り当てられます。

● 周期起動ハンドラ定義

<< 形式 >>

```
// Cyclic Handlar Definition
 cyclic_hand[ID番号]{
    interval_counter        = 周期起動ハンドラの周期間隔;
    mode                    = 周期起動ハンドラのモード;
    entry_address           = 周期起動ハンドラの開始アドレス;
 };
            :
            :
```

ID 番号は 1～32767 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

周期起動ハンドラ ID 番号ごとに以下の項目の定義をおこないます。

1. 周期間隔

【 定義形式 】数値

【 定義範囲 】1 ～ 65535

周期起動ハンドラの周期起動間隔を定義します。ここで定義する時間の単位はシステムクロック定義項目において定義したシステムクロックの単位時間です。例えば、システムクロックの単位時間が 10ms のときに、10 秒間隔で周期起動しようとすると、この値を 1000 に設定します。

2. モード

【 定義形式 】シンボル

【 定義範囲 】TCY_OFF または TCY_ON

周期起動ハンドラの初期モードを定義します。ここでは以下に示す２つのモードの何れかを定義します。

- TCY_OFF
  act_cyc システムコールを発行することで動作を始めるモード。
- TCY_ON
  システムの立ち上げと同時に動作を始めるモード。

3. 開始アドレス

【 定義形式 】シンボル、または、関数名

【 定義範囲 】なし

周期起動ハンドラの開始アドレスを定義します。

● アラームハンドラ定義

<< 形式 >>

```
// Alarm Handlar Definition
   alarm_hand[ID番号]{
   time            = アラームハンドラの起動時刻;
   entry_address   = アラームハンドラの開始アドレス;
};
          ⋮
          ⋮
```

ID 番号は 1～32767 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

アラームハンドラ ID 番号ごとに以下の項目の定義をおこないます。

1. 起動時刻

【 定義形式 】時刻

【 定義範囲 】0 : 0 : 0 ～ 0x7FFF : 0xFFFF : 0xFFFF

アラームハンドラの起動時刻を定義します。

2. 開始アドレス

【 定義形式 】シンボル、または、関数名

【 定義範囲 】なし

アラームハンドラの開始アドレスを定義します。

● 使用システムコール定義

<< 形式 >>

```
 // Systemcall Definition
    systemcall{
            システムコール名          = YES or NO;
};
            :
            :
```

ユーザプログラム中で使用されているシステムコール名を設定します。

<< 内容 >>

1. 使用システムコール

【 定義形式 】シンボル

【 定義範囲 】YES or NO

以下のシステムコール名を定義できます。

cre_tsk,del_tsk,sta_tsk,ista_tsk,ext_tsk,exd_tsk,ter_tsk,dis_dsp,ena_dsp,chg_pri,ichg_pri,rot_rdq,irot_rdq,rel_wai,irel_wai,get_tid,ref_tsk,sus_tsk,isus_tsk,rsm_tsk,irsm_tsk,slp_tsk,wup_tsk,iwup_tsk,can_wup,cre_flg,del_flg,set_flg,iset_flg,clr_flg,wai_flg,twai_flg,pol_flg,ref_flg,cre_sem,del_sem,sig_sem,isig_sem,wai_sem,twai_sem,preq_sem,ref_sem,cre_mbx,del_mbx,snd_msg,isnd_msg,rcv_msg,trcv_msg,prcv_msg,ref_mbx,cre_mbf,del_mbf,snd_mbf,tsnd_mbf,psnd_mbf,rcv_mbf,trcv_mbf,prcv_mbf,ref_mbf,cre_por,del_por,cal_por,tcal_por,pcal_por,acp_por,tacp_por,pacp_por,fwd_por,rpl_rdv,ref_por,def_int,ret_int,loc_cpu,unl_cpu,cre_mpf,del_mpf,get_blf,tget_blf,pget_blf,rel_blf,ref_mpf,cre_mpl,del_mpl,get_blk,tget_blk,pget_blk,rel_blk,ref_mpl,set_tim,get_tim,dly_tsk,act_cyc,ref_cyc,ref_alm,get_ver,ref_sys,def_exc,vclr_ems,vset_ems,vras_fex,irel_blf,vrst_blf,vrst_blk,vrst_msg,vrst_mbf,vcre_mbx,vdel_mbx,vsnd_mbx,visnd_mbx,vrcv_mbx,vtrcv_mbx,vprcv_mbx,vref_mbx,vrst_mbx

● 割り込みベクタ定義

<< 形式 >>

```
// Interrupt Vector Definition
interrupt_vector[割り込み要因番号]    = 割り込みハンドラの開始アドレス;
          :
          :
```

<< 内容 >>

1. 割り込みハンドラの開始アドレス

【 定義形式 】シンボルまたは関数名

【 定義範囲 】なし

割り込みハンドラの開始アドレスを指定します。[]内には、割り込み要因番号を指定してください。コンフィグレータによって割り込みハンドラ開始アドレスが、割り込みベクタテーブルの該当する割り込み要因番号の箇所に出力されます。

システムクロックを使用する場合、システムクロック割り込みハンドラ(__sys_timer)の定義が必要です。

【例】interrupt_vector[10]　　=　　__sys_timer;

7.1.3 コンフィグレーションファイル例

以下にコンフィグレーションファイルの例を示します。

```
//************************************************************************
//
//  COPYRIGHT(C) 2001(2003) RENESAS TECHNOLOGY CORPORATION
//  AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
//************************************************************************

// System Definition
system{
    stack_size      = 0x5000;
    priority        = 10;
    exc_handler     = NO;
};

//Max Definition
maxdefine{
    max_task        =       5;
    max_flag        =       1;
    max_mbx         =       1;
    max_sem         =       1;
    max_mem         =       1;
    max_cyh         =       2;
    max_alh         =       1;
    max_int         =       16;
};

//System Clock Definition
clock{
    timer_clock     =       33.3MHz;
    IPL             =       4;
    unit_time       =       1ms;
    timer           =       MFT00;
    file_name       =       m32102.tpl;
    initial_time    =       1:0x10ffff;
};

//Task Definition
task[1]{
    initial_start   =       ON;
    entry_address   =       task1();
    stack_size      =       0x1000;
    stack_area      =       __MR_INT;
    priority        =       2;
};

task[2]{
    entry_address   =       task2();
    stack_size      =       0x1000;
    stack_section   =       TASK2_STK;
    priority        =       3;
};

task[3]{
    entry_address   =       task3();
    stack_size      =       0x1000;
    stack_area      =       __MR_EXT;
    priority        =       4;
};

// Event Flag Definition
flag[1]{
    name            =       flg1;
};
// Semaphore Definiton
semaphore[1]{
    name            =       sem1;
    max_count       =       0x7ff;
    initial_count   =       0x0;
};
// MailBox Definition
```

```
mailbox[1]{
    name            =       mbx1;
    mbx_area        =       __MR_EXT;
    buffer_size     =       0x10;
};
// MessageBuffer Definition
message_buffer[1]{
    name            =       mbf;
    mbf_area        =       __MR_INT;
    buffer_size     =       0x10;
};
// Rendezvous Definition
rendezvous[1]{
    name            =       por1;
};
// Fixed-Memory pool Definition
memorypool[1]{
    name            =       mpf1;
    mpf_area        =       __MR_EXT;
    num_block       =       4;
    siz_block       =       0x100;
};
memorypool[2]{
    name            =       mpf2;
    section         =       MPF2_POOL;
    num_block       =       4;
    siz_block       =       0x100;
};
//Variable Memory  pool    Definition
variable_memorypool[]{
    name            =       mpl1;
    mpl_area        =       __MR_INT;
    max_memsize     =       0x500;
    heap_size       =       0x6000;
};
//Vmailbox
vmailbox[]{
    name            = vmbx1;
    message_pri     = FIFO;
    wait_pri        = PRI;
};

systemcall{
    cre_tsk         =       NO;
    del_tsk         =       NO;
    sta_tsk         =       YES;
    ter_tsk         =       YES;
    chg_pri         =       YES;
    rot_rdq         =       YES;
    rel_wai         =       YES;
    ena_dsp         =       YES;
    sus_tsk         =       YES;
    rsm_tsk         =       YES;
    slp_tsk         =       YES;
    wup_tsk         =       YES;
    set_flg         =       YES;
    wai_flg         =       YES;
    sig_sem         =       YES;
    wai_sem         =       YES;
    snd_msg         =       YES;
    rcv_msg         =       YES;
    snd_mbf         =       YES;
    rcv_mbf         =       YES;
    acp_por         =       YES;
    cal_por         =       YES;
    rpl_rdv         =       YES;
    get_blf         =       YES;
    rel_blf         =       YES;
    get_blk         =       YES;
    rel_blk         =       YES;
    unl_cpu         =       YES;
    dly_tsk         =       YES;
    ista_tsk        =       YES;
```

```
        ichg_pri        =       YES;
        irot_rdq        =       YES;
        irel_wai        =       YES;
        get_tid         =       YES;
        isus_tsk        =       YES;
        irsm_tsk        =       YES;
        iwup_tsk        =       YES;
        can_wup         =       YES;
        iset_flg        =       YES;
        clr_flg         =       YES;
        pol_flg         =       YES;
        isig_sem        =       YES;
        preq_sem        =       YES;
        isnd_msg        =       YES;
        prcv_msg        =       YES;
        set_tim         =       YES;
        get_tim         =       YES;
        act_cyc         =       YES;
        get_ver         =       YES;
        ret_int         =       NO;
        dis_dsp         =       YES;
        loc_cpu         =       YES;
        ext_tsk         =       YES;
        exd_tsk         =       NO;
    };
    //InterruptVector  Definition
    interrupt_vector[16]       =       __sys_timer;
   //
    //  End of Configuration
    //
```

7.2 コンフィグレータの実行

7.2.1 コンフィグレータ概要

コンフィグレータはコンフィグレーションファイルで定義した内容をアセンブリ言語のインクルードファイル等に変換するツールです。コンフィグレータの動作概要を図 6.1 に示します。

1. コンフィグレータの実行には以下の入力ファイルが必要です。

- コンフィグレーションファイル (XXXX.cfg)
  システムの初期設定項目を記述したファイルです。カレントディレクトリに作成します。

- デフォルトコンフィグレーションファイル (default.cfg)
  コンフィグレーションファイルで値の設定を省略した場合にこのファイルに書き込まれている値を設定します環境変数 "LIB32R"で示されるディレクトリ、もしくは、カレントディレクトリに置きます。両方のディレクトリに存在する場合は、カレントディレクトリのファイルが優先されます。

- makefile のテンプレートファイル[69] (makefile.ews, makefile.dos, makefile, Makefile)
  makefile[70]を生成する場合にテンプレートのファイルとして使用するファイルです。 (第7.2.4項参照)

- mr32r.inc テンプレートファイル (mr32r.inc)
  インクルードファイル mr32r.inc のテンプレートとなるファイルです。
  環境変数 "LIB32R"で示されるディレクトリに存在します。

- タイマ設定用テンプレートファイル(xxx.tpl)

  コンフィグレーションファイルで指定したファイル名のタイマ設定用テンプレートを環境変数"LIB32R"ディレクトリからコンフィグレータを実行したディレクトリに"timer.inc"というファイル名でコピーします。この"timer.inc"は、ipl.ms(または ipl.s)でインクルードされています。

- MR32R バージョンファイル (version)
  MR32R のバージョンを記述したファイルです。環境変数 "LIB32R" で示されるディレクトリに存在します。コンフィグレータはこのファイルを読み込み、起動メッセージに MR32R のバージョン情報を出力します。

2. コンフィグレータの実行によって以下のファイルが出力されます。

コンフィグレータが出力したファイルには、ユーザのデータ定義を行わないで下さい。データ定義を行った後で、コンフィグレータを起動するとユーザの定義したデータは失われます。

- システムデータ定義ファイル (sys_rom.inc)
  システムの設定を定義しているファイルです。

- インクルードファイル (mr32r.inc)
  mr32r.inc はアセンブリ言語用のインクルードファイルです。

- システム生成手順記述ファイル (makefile)
  システムを自動生成するためのファイルです。

---

[69] EWS 版では makefile.ews,DOS 版では makefile.dos を使用します。
[70] この makefile は、UNIX 標準もしくは準拠の make コマンドで処理可能なシステム生成手順記述ファイルです。

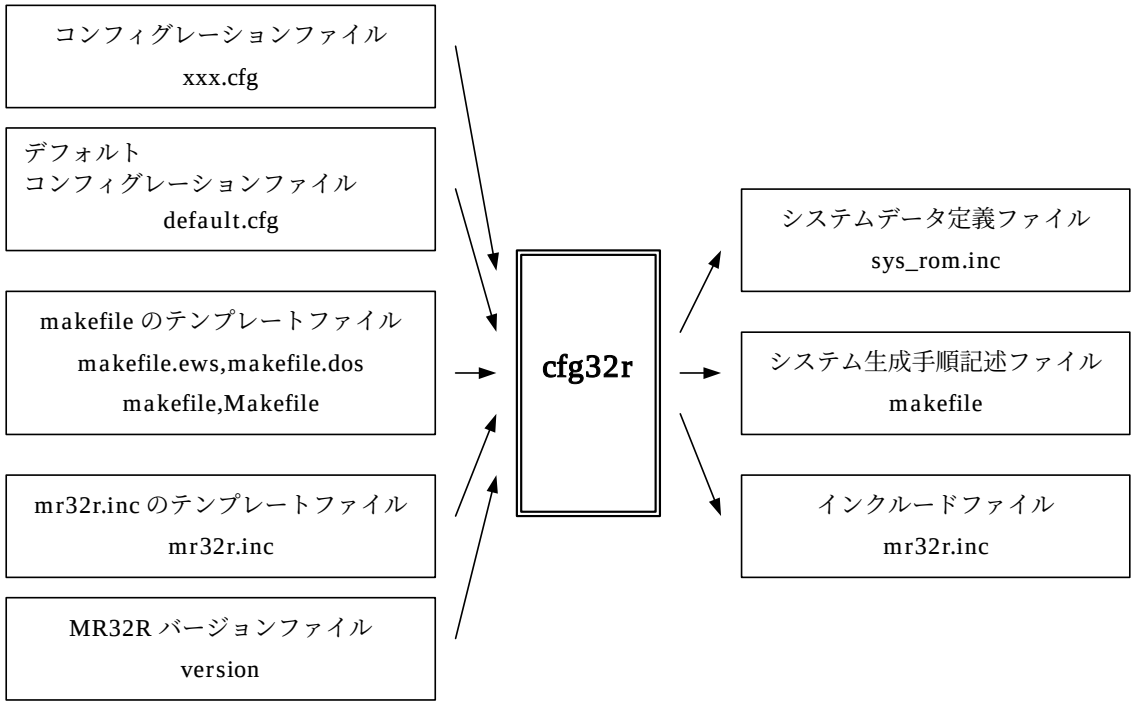


図 7.1　コンフィグレータ動作概要

7.2.2 コンフィグレータの環境設定

コンフィグレータを実行するにあたって環境変数"LIB32R"が正しく設定されているかを確認してください。

環境変数"LIB32R"で示すディレクトリ下には以下のファイルがないと正常に実行できません。

- デフォルトコンフィグレーションファイル (default.cfg)
  カレントディレクトリにコピーして使用することもできます。その場合はカレントディレクトリのファイルを優先して使用します。

- システム RAM 領域定義データベースファイル (sys_ram.inc)
- mr32r.inc のテンプレートファイル (mr32r.inc)
- セクション定義ファイル (section あるいは m32r.cmd)
- makefile のテンプレートファイル (makefile.ews または makefile.dos)
- MR32R バージョンファイル (version)
- スタートアップファイル (crt0mr.ms,starut.ms または crt0mr.s)
- 割込み制御プログラム(ipl.ms または ipl.s)
- タイマ設定用テンプレートファイル(xxx.tpl)

7.2.3 コンフィグレータ起動方法

コンフィグレータは以下の形式で起動します。

```
A> cfg32r [-vmVi] コンフィグレーションファイル名
```

コンフィグレーションファイル名は、 通常拡張子 (.cfg) かまたは拡張子 (.cfg) を除いたファイル名を指定します。

コマンドオプション

-v オプション
コマンドのオプションの説明と詳細なバージョンを表示します。

-V オプション
コマンドが生成するファイルの作成状況を表示します。

-i オプション
タイマの自動設定用ファイル(“timer.inc”)を生成します。カレントディレクトリにすでに存在するときは、生成しません。

-m オプション
UNIX 標準もしくは準拠のシステム生成手順記述ファイル (makefile)を作成します。指定がない場合は makefile を作成しません。[71]
また、カレントディレクトリにスタートアップファイルとセクション定義ファイル がない場合は、環境変数 LIB32R が示すディレクトリにある、サンプルスタートアップファイルとセクション定義ファイルをカレントディレクトリにコピーします。

[71] UNIX 標準もしくは準拠の”makefile”には、”clean”ターゲットによりワークファイルを削除する機能があります。すなわち、make コマンドで生成されたオブジェクトファイルなどを削除するには以下のように行います。
> make clean

7.2.4 makefile 生成機能

コンフィグレータは以下の手順で makefile を生成します。

1. ソースファイルの依存関係を調べます。

コンフィグレータはカレントディレクトリの拡張子が.c と.s あるいは.ms のファイルをそれぞれ、C 言語とアセンブリ言語として、それらがインクルードするファイルなどの依存関係を調べます。

したがって、ソースファイルを記述する場合には次の 2 点に注意してください。

- ソースファイルはカレントディレクトリに置かなければなりません。
- ソースファイルの拡張子はＣ言語では'.c'を、アセンブリ言語では'.s'あるいは'.ms'を使用してください。

2. makefile へファイルの依存関係を書き込みます。

カレントディレクトリの "makefile"、または"Makefile"または、環境変数"LIB32R"で示されるディレクトリの"makefile.ews"または"makefile.dos"を、テンプレートファイルとして、カレントディレクトリに"makefile"を作成します。

7.2.5 コンフィグレータ実行上の注意

以下にコンフィグレータ実行上の注意点を示します。

- コンフィグレータを再度かけ直した場合は、必ず make clean を実行するかオブジェクトファイル（拡張子.o あるいは.mo）を全て消去してから make コマンドを実行してください。リンク時等にエラーが発生する場合があります。
- スタートアッププログラム名、およびセクション定義ファイル名は変更しないでください。変更した場合、コンフィグレータ実行時にエラーが発生します。
- コンフィグレータ cfg32r では、UNIX 標準、もしくは準拠の makefile しか生成しません。

7.2.6 コンフィグレータのエラーと対処方法

以下のメッセージが表示された場合はコンフィグレータが正常に終了していませんのでコンフィグレーションファイルを修正の上、再度コンフィグレータを実行してください。

エラーメッセージ

cfg32r Error : syntax error near line 16 (test.cfg)
コンフィグレーションファイルに文法エラーがあります。

cfg32r Error : not enough memory
メモリが足りません。

cfg32r Error : illegal option --> <x>
コンフィグレータのコマンドオプションに誤りがあります。

cfg32r Error : illegal argument --> <xx>
コンフィグレータの起動形式に誤りがあります。

cfg32r Error : can't write open <XXXX>
XXXX ファイルが作成できません。ディレクトリの属性やディスクの残り容量を確認してください。

cfg32r Error : can't open <XXXX>
XXXX ファイルにアクセスできません。XXXX ファイルの属性や、存在を確認してください。

cfg32r Error : can't open version file
環境変数"LIB32R"の示すディレクトリの下に MR32R バージョンファイル"version"がありません。

cfg32r Error : can't open default configuration file
デフォルトコンフィグレーションファイルがアクセスできません。環境変数 "LIB32R"の示すディレクトリ、またはカレントディレクトリに"default.cfg"が必要です。

cfg32r Error : can't open configuration file <test.cfg>
コンフィグレーションファイルがアクセスできません。コンフィグレータの起動形式を確認の上、正しいファイル名を指定してください。

cfg32r Error : illegal XXXX --> <xx> near line 212 (test.cfg)
定義項目 XXXX の数値または ID 番号が間違っています。定義範囲を確認してください。

cfg32r Error : Unknown XXXX --> <xx> near line 23 (test.cfg)
定義項目 XXXX のシンボル定義が間違っています。定義範囲を確認してください。

cfg32r Error : too big XXXX's ID number --> <6> (test.cfg)
XXXX 定義の ID 番号に、定義したオブジェクトの総数を超える値が設定されています。ID 番号がオブジェクトの総数を超えることはありません。

cfg32r Error : too big task[x]'s priority --> <16> near line 100 (test.cfg)
ID 番号 x のタスク定義項目の初期優先度が、システム定義項目の優先度値を越えています。

cfg32r Error : XXXX is not defined (test.cfg)
コンフィグレーションファイルで XXXX の項目の定義が必要です。

cfg32r Error : system's default is not defined
デフォルトコンフィグレーションファイルで定義が必要な項目です。

cfg32r Error : double definition <XXXX> near line 17 (test.cfg)
項目 XXXX は既に定義されています。確認の上、重複定義を削除してください。

cfg32r Error : double definition XXXX[x] near line 77 (default.cfg)

cfg32r Error : double definition XXXX[x] (test.cfg) near line 77 (test.cfg)
項目 XXXX において ID 番号 x は既に登録されています。ID 番号を変更するか重複定義を削除してください。

cfg32r Error : you must define XXXX near line 107 (test.cfg)
XXXX は、省略できない項目です。

cfg32r Error : you must define SYMBOL near line 30 (test.cfg)
省略できないシンボルです。

cfg32r Error : start-up-file (XXXX) not found
カレントディレクトリにスタートアップファイル XXXX が見つかりません。スタートアップファイル"start.s"または "crt0mr.ms"が、カレントディレクトリに必要です。

cfg32r Error : bad start-up-file(XXXX)
カレントディレクトリに不要なスタートアップファイルがあります。

cfg32r Error : no source file
カレントディレクトリにソースファイルがありません。

cfg32r Error : zero divide error near line 28 (test.cfg)
演算式で 0(ゼロ) 除算が発生しました。

cfg32r Error : “max_memsize” must set XX or less in YYYY definition near line 28 (test.cfg)
YYYY 定義の”max_memsize”の項目に XX バイト以下のサイズをセットしてください。

cfg32r Error : can’t define both “stack_area” keyword and “stack_section” keyword in task definnition near line 28 (test.cfg)
タスク定義において”stack_area”と”stack_section”の両方が定義されています。どちらか片方だけを定義してください。

警告メッセージ

以下のメッセージは警告ですので、内容が理解できていれば無視してもかまいません。

cfg32r Warning : system is not defined (test.cfg)

cfg32r Warning : system.XXXX is not defined (test.cfg)
コンフィグレーションファイルでシステム定義またはシステム定義項目 XXXX が省略されています。

cfg32r Warning : system.message_size is not defined (test.cfg)
システム定義中のメッセージサイズ定義項目が省略されています。メイルボックス機能のメッセージサイズ(16 または 32)を指定して下さい。

cfg32r Warning : task[x].XXXX is not defined near line 32 (test.cfg)
ID 番号 x のタスク定義項目 XXXX が省略されています。

cfg32r Warning : Already definition XXXX near line 20 (test.cfg)
XXXX は既に定義されています。定義内容は無視されます。確認の上、重複定義を削除してください。

cfg32r Warning : specified variable memorypool.max`memsize 120 (test.cfg)
可変長メモリプール定義項目の最大メモリサイズが 120 より、小さいためその値を 120 に設定しました。

cfg32r Warning : interrupt_vector[x]'s default is not defined (default.cfg)
デフォルトコンフィグレーションファイルでベクタ番号 x の割り込みベクタ定義が抜けています。

cfg32r Warning : interrupt_vector[x]'s default is not defined near line 213 (test .cfg)
コンフィグレーションファイルのベクタ番号 x の割り込みベクタは、デフォルトコンフィグレーションファイルに定義されていません。

cfg32r Warning : Initial Start Task not defined
コンフィグレーションファイルで、初期起動タスクの定義がないためタスク ID 番号 1 のタスクを初期起動タスクとして定義しました。

cfg32r Warning : specified XXXX_YYYstk 180 near line 213 (test.cfg)
メモリの確保に必要な最低限のサイズ(180 バイト)を確保しました。

cfg32r Warning : XXX is less than real definition number
最大項目数定義の"XXX"で指定した値が実際にコンフィグレーションファイルで定義した値に置き換えます。

その他のメッセージ

以下のメッセージは makefile を生成する場合にのみ出力される警告メッセージです。コンフィグレータは要因となった部分を読み飛ばして makefile を生成します。

cfg32r Error : test.c(line 11): include format error.
ファイル読み込みの書式が間違っています。正しい書式に書き直してください。

cfg32r Warning : test.c(line 12): can't find <XXXX>

cfg32r Warning : test.c(line 13): can't find "XXXX"
インクルードファイル XXXX が見つかりません。ファイル名および存在を確認して下さい。

cfg32r Warning : over character number of including path-name
インクルードファイルのパス名が 255 文字を超えています。

第 8 章　　各設定ファイルのカスタマイズ方法

8.1 割り込み制御プログラムについて

MR32R では、「図 6.2　　割り込みハンドラの処理手順」で示すように割り込みコントローラの設定やタイマの設定に関係する部分は、OS カーネル内には、含まれていません。これらの設定に関しては、"ipl.ms(または ipl.s)"ファイルで行います。MR32R では、「6.8.2 タイマの自動設定について」にあるようにいくつかのマイコンや周辺機能に対応したテンプレートがあり、そのテンプレートを使用することができます。

しかし、使用しているマイコンに対応したテンプレートがない場合は、"ipl.ms(または ipl.s)"ファイルをユーザが作成する必要があります。

8.1.1 割り込み制御プログラムの処理内容

割り込み処理プログラムでは、表 8-1に示すラベルのアセンブラルーチンを記述します。それぞれのアセンブラルーチンでどのような処理を記述する必要があるのか以下に説明します。

表 8-1　割り込み制御プログラムの概要

ラベル	概要	レジスタの保証
__sys_timer_init	タイマの初期化を行います	不要
__SAVE_IPL_to_STACK	割り込みステータスをスタックに退避します。	必要
__RESTORE_IPL_from_STACK	割り込みステータスをスタックに退避します。	必要
__set_ipl	起動時の割り込み優先レベルを設定します。	不要
__Int_Dummy	ダミーの割り込みルーチンです。	必要
__SYS_DUMMY	ダミーのシステムコールルーチンです。	不要

- __sys_timer_init

  このルーチンは、スタートアップルーチン内で呼び出され、システム時刻の設定とシステムクロックの初期化を行います。
  システム時刻の設定部分は、マイコンや周辺機能に依存せず、次節からのサンプルプログラム内のＩの部分のように記述します。
  タイマの初期化は、使用するタイマに応じて任意の間隔で割り込みが発生するように記述します。
  ユーザがタイマ初期化を記述する際は、コンフィグレーションファイルのシステムクロック定義の項目"timer"に OTHER を指定します。
  そのときに "timer_clock"，"IPL"，"file_name"，"unit_time"に指定されている値は無視されます。

- __SAVE_IPL_to_STACK

  割り込みステータスと割り込みハンドラの入り口アドレスを図 8.1にようにスタックに積んで OS 処理に戻ります。
  このルーチンは、割り込みが発生した時に、図 6.2で示す OS 割り込み出入り口処理から呼び出されます。
  割り込みハンドラの入り口アドレスは、コンフィグレータによって出力されるアドレステーブルから読み出します。
  アドレステーブルは、INTERRUPT_VECTOR または.INTERRUPT_VECTOR セクション(先頭ラベルが__INT_VECTOR で定義されています。)に割り込み要因０からコンフィグレーションファイルの最大項目定義の max_int で指定された値までの分用意されます。例えば、「max_int = 63;」を指定した場合、割り込み要因番号 0 から割り込み要因番号 63 までの 64 のエントリが用意されます。
  また、割り込みハンドラのアドレスは、コンフィグレーションファイルの割り込みハンドラ

定義で記述した割り込み要因番号のエントリに対応する割り込みハンドラの入り口アドレスが出力されます。例えば、「interrupt_vector[16] = __sys_timer;」と指定した場合、割り込み要因番号 16 のエントリに__sys_timer のラベルが埋め込まれます。
従って、__INT_VECTOR[割り込み要因番号]を読み出すことによって割り込み要因番号に対応する割り込みハンドラの入り口アドレスを取得することができます。

- __RESTORE_IPL_from_STACK
  図 8.1にようにスタックに積まれている割り込みステータスを復帰し、OS 処理に戻ります。このルーチンは、割り込みが発生した時に、図 6.2で示す OS 割り込み出入り口処理から呼び出されます。

- __set_ipl
  割り込みマスクレベルを設定します。このルーチンは、スタートアップルーチンから呼び出されます。

- __Int_Dummy[72]
  割り込みハンドラとして登録されていない割り込みが発生した場合、このルーチンが呼び出されます。

- __SYS_DUMMY
  コンフィグレーションファイルのシステムコール定義で使用しないと定義したシステムコールを使用した場合、このルーチンが呼び出されます。

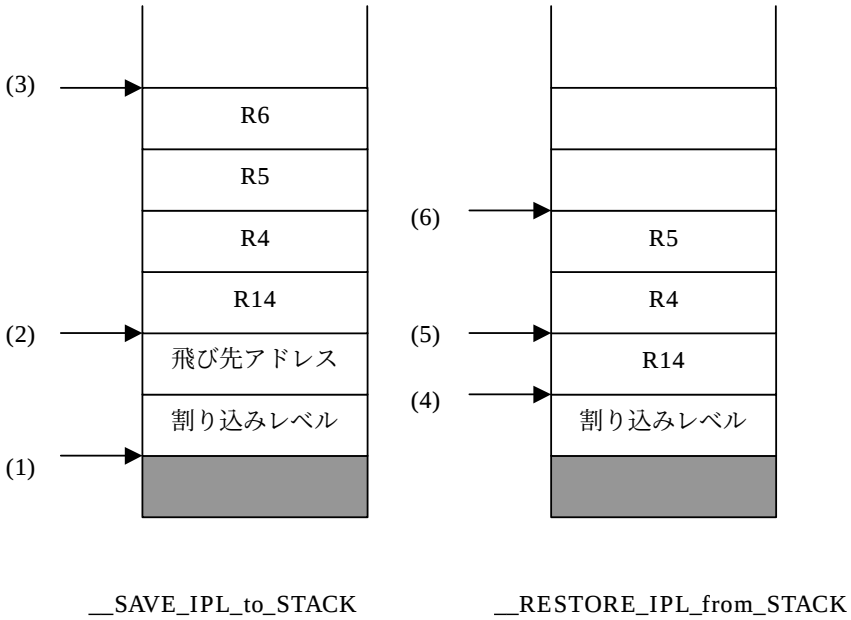


図 8.1　割り込み制御プログラムのスタックの状況

[72] 割り込みハンドラとして登録されていない割り込み要因番号の割り込みは、割り込みハンドラアドレステーブルに__Int_Dummy が埋め込まれます。

8.1.2 割り込み制御プログラム例 (M3T-CC32R 対応)

```
;***********************************************************************
;
;   COPYRIGHT(C) 2001(2003) RENESAS TECHNOLOGY CORPORATION
;   AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
;
;
;   $Id: ipl.s,v 1.2 2001/04/18 06:24:29 inui Exp $
;
;***********************************************************************

    .include        "mr32r.inc"
    .section        MR_KERNEL,code

ICUISTS:    .equ    0x00EFF004
ICUIMASK:   .equ    0x00EFF01C
MFTCR:              .equ    0x00EFC000
MFTMOD:             .equ    0x00EFC100
MFTRLD:             .equ    0x00EFC10C
ICUCR:              .equ    0x00EFF23C
__write_bit: .equ   0x8000
__ena_bit:  .equ    0x0080
__timer_div: .equ   0x3
__timer_count:      .equ    0x6596

    .global __sys_timer_init,__stmr_ipl,__stmr_src,__stmr_cnt
    .global __D_Sys_TIME_H,__Sys_time,__stmr_ctr
    .global __D_Sys_TIME_L

;タイマを使用する場合、タイマの初期化をここで行います。
;このルーチンは、スタートアッププログラム(crt0mr.msまたは、start.ms)
;から呼ばれます。
    .AIF    &¥USE_TIMER
__sys_timer_init:

;システムクロックの初期化を行います。この部分を変更しないでください。
    st      r14,@-r15
    ld24    r4,#__D_Sys_TIME_H
    lduh    r5,@r4
    ld24    r6,#__Sys_time
    sth     r5,@r6
    ld24    r4,#__D_Sys_TIME_L
    ld      r5,@r4
    st      r5,@(2,r6)

;OSのシステムクロックに使用するタイマの初期設定を行います。
;MFT        control register setting
    seth    r1,#shigh(MFTCR)
    ld      r0,@(low(MFTCR),r1)
    ld24    r2,#__write_bit
    or      r0,r2
    st      r0,@(low(MFTCR),r1)
;MFT mode register setting
    seth    r1,#shigh(MFTMOD)
    ld24    r0,#0x8000
    or3     r0,r0,#__timer_div
    st      r0,@(low(MFTMOD),r1)
;MFT reload register setting
    seth    r1,#shigh(MFTRLD)
    ld24    r0,#__timer_count
    st      r0,@(low(MFTRLD),r1)

;MFT interrupt control register setting(IEN=1, ILEVEL=3)
    seth    r1,#shigh(ICUCR)
    ldi     r0,#0x1100
    or3     r0,r0,#3
    st      r0,@(low(ICUCR),r1)

```

```
 68 ;MFT control register setting
 69      seth    r1,#shigh(MFTCR)
 70      ld      r0,@(low(MFTCR),r1)
 71      ld24    r2,#(__write_bit | __ena_bit)
 72      or      r0,r2
 73      st      r0,@(low(MFTCR),r1)
 74
 75      ld      r14,@r15+
 76      jmp     r14
 77      .aendi
 78
 79 ;割り込み制御レジスタを退避し、割り込みの飛びさきアドレスをスタック上に
 80 ;積み、OSカーネルに戻ります。
 81 ;割り込み入り口処理でOSカーネルから呼び出されますのでこのラベル名は
 82 ;変更しないでください。
 83 ;
 84      .global __SAVE_IPL_to_STACK,__INT_VECTOR
 85  __SAVE_IPL_to_STACK:
 86
 87 ;割り込み制御レジスタをスタックに退避し、割り込み要因番号を読みます。
 88      addi    r15,#-8                                 ------------(1)
 89      st      r14,@-r15                               ------------(2)
 90      st      r4,@-r15
 91      st      r5,@-r15
 92      st      r6,@-r15
 93      seth    r4,#shigh(ICUISTS) )                    ------------(3)
 94      ld      r5,@(low(ICUISTS),r4)
 95      st      r5,@(D'20,r15)
 96      srli    r5,#20
 97
 98 ;__INT_VECTORからテーブル検索し、割り込み飛び先アドレスをスタックに積みます。
 99 ;(このテーブルは、コンフィグレータによって、sys_rom.incファイルにINTERRUPT_VECTOR
100 ;セクションとして出力され、任意のアドレスに配置することができます。)
101      ld24    r6,#__INT_VECTOR
102      add     r6,r5
103      ld      r6,@r6
104      st      r6,@(16,r15)                            ------------(3)
105      ld      r6,@r15+
106      ld      r5,@r15+
107      ld      r4,@r15+
108      ld      r14,@r15+
109      jmp     r14                                     ------------(2)
110
111 ;割り込み制御レジスタを復帰し、OSカーネルに戻ります。
112 ;割り込み出口処理でOSカーネルから呼び出されますのでこのラベル名は
113 ;変更しないでください。
114      .global __RESTORE_IPL_from_STACK
115 __RESTORE_IPL_from_STACK:                            ------------(5)
116      st      r14,@-r15
117      st      r4,@-r15
118      st      r5,@-r15
119      ld      r5,@(12,r15)                            ------------(6)
120      seth    r4,#shigh(ICUIMASK)
121      st      r5,@(low(ICUIMASK),r4)
122      ld      r5,@r15+
123      ld      r4,@r15+
124      ld      r14,@r15+                               ------------(5)
125      addi    r15,#4
126      jmp     r14                                     ------------(4)
127
128 ;割り込み優先レベルの初期設定を行います。
129 ;スタートアップファイルから呼び出されます。
130      .global __set_ipl
131 __set_ipl:
132      ldi     r5,#0x07
133      seth    r4,#shigh(ICUIMASK)
134      sth     r5,@(low(ICUIMASK),r4)
135      jmp     r14
136
```

```
137 ;ダミーの割り込みルーチンです。
138 ;コンフィグレーションファイルに割り込みハンドラ定義されていない
139 ;割り込みが入った場合、このルーチンが呼び出されます。
140
141     .global __Int_Dummy
142 __Int_Dummy:
143     jmp     r14
144
145 ;コンフィグレーションファイルのシステムコール定義で"NO"を
146 ;指定したにもかかわらず、ユーザプログラム中でそのシステムコールを
147 ;使用した場合にこのルーチンが呼び出されます。
148     .global __SYS_DUMMY
149 __SYS_DUMMY
150     jmp     r14
151
```

8.1.3 割り込み制御プログラム例 (M3T-TW32R 対応)

```
 1 ;************************************************************************
 2 ;
 3 ;   COPYRIGHT(C) 2001(2003) RENESAS TECHNOLOGY CORPORATION
 4 ;   AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
 5 ;
 6 ;
 7 ;   $Id: ipl.s,v 1.2 2001/04/18 06:24:29 inui Exp $
 8 ;
 9 ;************************************************************************
10
11      .include        "mr32r.inc"
12      .section        .MR_KERNEL,"ax"
13
14      .equ    ICUISTS,        0x00F7F004
15      .equ    ICUIMASK,       0x00F7F01C
16      .equ    MFTCR,          0x00F7C000
17      .equ    MFTMOD,         0x00F7C080
18      .equ    MFTOS,          0x00F7C088
19      .equ    MFTRLD,         0x00F7C08C
20      .equ    ICUCR,          0x00F7F23C
21      .equ    __write_bit,    0x8000
22      .equ    __ena_bit,      0x0080
23
24      .equ    __timer_count,  0x6596
25      .equ    __timer_div,    0x3
26
27 ;タイマを使用する場合、タイマの初期化をここで行います。
28 ;このルーチンは、スタートアッププログラム(crt0mr.msまたは、start.ms)
29 ;から呼ばれます。
30 .ifdef       USE_TIMER
31      .global __sys_timer_init,__stmr_ipl,__stmr_src,__stmr_cnt
32      .global __D_Sys_TIME_H,__Sys_time,__stmr_ctr
33
34 __sys_timer_init:
35
36 ;システムクロックの初期化を行います。この部分を変更しないでください。
37      st      R14,@-R15
38      ld24    R4,#__D_Sys_TIME_H
39      ldh     R5,@R4
40      ld24    R6,#__Sys_time
41      sth     R5,@R6
42      ld24    R4,#__D_Sys_TIME_L
43      ld      R5,@R4
44      st      R5,@(2,R6)
45
46 ;OSのシステムクロックに使用するタイマの初期設定を行います。
47
48      ld24    R10,#__timer_count
49      ldi     R12,#__timer_div
50 ;MFT         control register setting
51      ld24    r1,#MFTCR
52      ld      r0,@r1
53      ld24    r2,#__write_bit
54      or      r0,r2
55      st      r0,@r1
56 ;MFT mode register setting
57      ld24    r1,#MFTMOD
58      ld24    r0,#0x8000
59      or      r0,r12
60      st      r0,@r1
61 ;MFT port output status register setting
62      ld24    r1,MFTOS
63      ldi     r0,#0
64      st      r0,@r1
65 ;MFT reload register setting
66      ld24    r1,#MFTRLD
67      addi    r10,#-1
68      st      r10,@r1
```

```
 69
 70 ;MFT interrupt control register setting(IEN=1, ILEVEL=3)
 71      ld24    r1,#ICUCR
 72      ldi     r0,#0x1100
 73      or3     r0,r0,#__stmr_ipl
 74      st      r0,@r1
 75
 76 ;MFT control register setting
 77      ld24    r1,#MFTCR
 78      ld24    r2,#(__write_bit | __ena_bit)
 79      or      r0,r2
 80      st      r0,@r1
 81
 82      ld      r14,@r15+
 83      jmp     r14
 84      .endif
 85
 86 ;割り込み制御レジスタを退避し、割り込みの飛びさきアドレスをスタック上に
 87 ;積み、OSカーネルに戻ります。
 88 ;割り込み入り口処理でOSカーネルから呼び出されますのでこのラベル名は
 89 ;変更しないでください。
 90 ;
 91
 92      .global __SAVE_IPL_to_STACK,__INT_VECTOR
 93  __SAVE_IPL_to_STACK:                                  ------------(1)
 94      addi    R15,#-8                                   ------------(2)
 95      st      R14,@-R15
 96      st      R4,@-R15
 97      st      R5,@-R15
 98      st      R6,@-R15
 99
100 ;割り込み制御レジスタをスタックに退避し、割り込み要因番号を読みます。
101
102      ld24    r4,#ICUISTS                              ------------(3)
103      ld      r5,@r4
104      st      r5,@(20,r15)
105      srli    r5,#20
106
107 ;__INT_VECTORからテーブル検索し、割り込み飛び先アドレスをスタックに積みます。
108 ;(このテーブルは、コンフィグレータによって、sys_rom.incファイルにINTERRUPT_VECTOR
109 ;セクションとして出力され、任意のアドレスに配置することができます。)
110
111      ld24    r6,#__INT_VECTOR
112      add     r6,r5
113      ld      r6,@r6
114      st      r6,@(16,r15)
115      ld      r6,@r15+                                 ------------(3)
116      ld      r5,@r15+
117      ld      r4,@r15+
118      ld      r14,@r15+
119      jmp     r14                                      ------------(2)
120
121 ;割り込み制御レジスタを復帰し、OSカーネルに戻ります。
122 ;割り込み出口処理でOSカーネルから呼び出されますのでこのラベル名は
123 ;変更しないでください。
124
125      .global __RESTORE_IPL_from_STACK
126 __RESTORE_IPL_from_STACK:                              ------------(5)
127      st      r14,@-r15
128      st      r4,@-r15
129      st      r5,@-r15
130      ld      r5,@(12,r15)                             ------------(6)
131      ld24    r4,#ICUIMASK
132      st      r5,@r4
133      ld      r5,@r15+                                 ------------(5)
134      ld      r4,@r15+
135      ld      r14,@r15+
136      addi    r15,#4
137      jmp     r14                                      ------------(4)
138
```

```
139 ;割り込み優先レベルの初期設定を行います。
140 ;スタートアップファイルから呼び出されます。
141     .global __set_ipl
142 __set_ipl:
143     ldi     r5,#0x07
144     ld24    r4,#ICUIMASK
145     sth     r5,@r4
146     jmp     r14
147
148 ;ダミーの割り込みルーチンです。
149 ;コンフィグレーションファイルに割り込みハンドラ定義されていない
150 ;割り込みが入った場合、このルーチンが呼び出されます。
151
152     .global __Int_Dummy
153 __Int_Dummy:
154     jmp     R14
155
156 ;コンフィグレーションファイルのシステムコール定義で"NO"を
157 ;指定したにもかかわらず、ユーザプログラム中でそのシステムコールを
158 ;使用した場合にこのルーチンが呼び出されます。
159
160     .global __SYS_DUMMY
161 __SYS_DUMMY
162     jmp     r14
163
```

8.1.4 割り込み制御プログラム例 (D-CC/M32R 対応)

```
 1 ;***************************************************************************
 2 ;
 3 ;   COPYRIGHT(C) 2001(2003) RENESAS TECHNOLOGY CORPORATION
 4 ;   AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
 5 ;
 6 ;
 7 ;   $Id: ipl.s,v 1.2 2001/04/18 06:24:29 inui Exp $
 8 ;
 9 ;***************************************************************************
10     .include        "mr32r.inc"
11     .section        .MR_KERNEL,"ax"
12
13     .equ    ICUISTS,        0x00F7F004
14     .equ    ICUIMASK,       0x00F7F01C
15     .equ    MFTCR,          0x00F7C000
16     .equ    MFTMOD,         0x00F7C080
17     .equ    MFTOS,          0x00F7C088
18     .equ    MFTRLD,         0x00F7C08C
19     .equ    ICUCR,          0x00F7F23C
20     .equ    __write_bit,    0x8000
21     .equ    __ena_bit,      0x0080
22
23     .equ    __timer_count,  0x6596
24     .equ    __timer_div,    0x3
25
26 ;タイマを使用する場合、タイマの初期化をここで行います。
27 ;このルーチンは、スタートアッププログラム(crt0mr.msまたは、start.ms)
28 ;から呼ばれます。
29 .ifdef       USE_TIMER
30     .global __sys_timer_init,__stmr_ipl,__stmr_src,__stmr_cnt
31     .global __D_Sys_TIME_H,__Sys_time,__stmr_ctr
32
33 __sys_timer_init:
34
35 ;システムクロックの初期化を行います。この部分を変更しないでください。
36
37     st      R14,@-R15
38     ld24    R4,#__D_Sys_TIME_H
39     ldh     R5,@R4
40     ld24    R6,#__Sys_time
41     sth     R5,@R6
42     ld24    R4,#__D_Sys_TIME_L
43     ld      R5,@R4
44     st      R5,@(2,R6)
45
46 ;OSのシステムクロックに使用するタイマの初期設定を行います。
47
48     ld24    R10,#__timer_count
49     ldi     R12,#__timer_div
50 ;MFT         control register setting
51     ld24    r1,#MFTCR
52     ld      r0,@r1
53     ld24    r2,#__write_bit
54     or      r0,r2
55     st      r0,@r1
56 ;MFT mode register setting
57     ld24    r1,#MFTMOD
58     ld24    r0,#0x8000
59     or      r0,r12
60     st      r0,@r1
61 ;MFT port output status register setting
62     ld24    r1,MFTOS
63     ldi     r0,#0
64     st      r0,@r1
65 ;MFT reload register setting
66     ld24    r1,#MFTRLD
67     addi    r10,#-1
68     st      r10,@r1
```

```
 69
 70 ;MFT interrupt control register setting(IEN=1, ILEVEL=3)
 71      ld24    r1,#ICUCR
 72      ldi     r0,#0x1100
 73      or3     r0,r0,#__stmr_ipl
 74      st      r0,@r1
 75
 76 ;MFT control register setting
 77      ld24    r1,#MFTCR
 78      ld24    r2,#(__write_bit | __ena_bit)
 79      or      r0,r2
 80      st      r0,@r1
 81
 82      ld      r14,@r15+
 83      jmp     r14
 84      .endif
 85
 86 ;割り込み制御レジスタを退避し、割り込みの飛びさきアドレスをスタック上に
 87 ;積み、OSカーネルに戻ります。
 88 ;割り込み入り口処理でOSカーネルから呼び出されますのでこのラベル名は
 89 ;変更しないでください。
 90 ;
 91
 92      .global __SAVE_IPL_to_STACK,__INT_VECTOR
 93  __SAVE_IPL_to_STACK:                                  ------------(1)
 94      addi    R15,#-8                                   ------------(2)
 95      st      R14,@-R15
 96      st      R4,@-R15
 97      st      R5,@-R15
 98      st      R6,@-R15
 99
100 ;割り込み制御レジスタをスタックに退避し、割り込み要因番号を読みます。
101
102      ld24    r4,#ICUISTS                              ------------(3)
103      ld      r5,@r4
104      st      r5,@(20,r15)
105      srli    r5,#20
106
107 ;__INT_VECTORからテーブル検索し、割り込み飛び先アドレスをスタックに積みます。
108 ;(このテーブルは、コンフィグレータによって、sys_rom.incファイルにINTERRUPT_VECTOR
109 ;セクションとして出力され、任意のアドレスに配置することができます。)
110
111      ld24    r6,#__INT_VECTOR
112      add     r6,r5
113      ld      r6,@r6
114      st      r6,@(16,r15)
115      ld      R6,@R15+                                 ------------(3)
116      ld      R5,@R15+
117      ld      R4,@R15+
118      ld      R14,@R15+
119      jmp     R14                                      ------------(2)
120
121 ;割り込み制御レジスタを復帰し、OSカーネルに戻ります。
122 ;割り込み出口処理でOSカーネルから呼び出されますのでこのラベル名は
123 ;変更しないでください。
124
125      .global __RESTORE_IPL_from_STACK
126 __RESTORE_IPL_from_STACK:                              ------------(5)
127      st      R14,@-R15
128      st      R4,@-R15
129      st      R5,@-R15
130      ld      R5,@(12,R15)                             ------------(6)
131      ld24    r4,#ICUIMASK
132      st      r5,@r4
133      ld      R5,@R15+
134      ld      R4,@R15+
135      ld      R14,@R15+                                ------------(5)
136      addi    R15,#4
137      jmp     R14                                      ------------(4)
```

```
138
139 ;割り込み優先レベルの初期設定を行います。
140 ;スタートアップファイルから呼び出されます。
141
142     .global __set_ipl
143 __set_ipl:
144     ldi     r5,#0x07
145     ld24    r4,#ICUIMASK
146     sth     r5,@r4
147     jmp     r14
148
149 ;ダミーの割り込みルーチンです。
150 ;コンフィグレーションファイルに割り込みハンドラ定義されていない
151 ;割り込みが入った場合、このルーチンが呼び出されます。
152
153     .global __Int_Dummy
154 __Int_Dummy:
155     jmp     R14
156
157 ;コンフィグレーションファイルのシステムコール定義で"NO"を
158 ;指定したにもかかわらず、ユーザプログラム中でそのシステムコールを
159 ;使用した場合にこのルーチンが呼び出されます。
160
161     .global __SYS_DUMMY
162 __SYS_DUMMY
163     jmp     r14
164
```

8.2 MR32R スタートアッププログラムのカスタマイズ方法

MR32R には、以下に示す 2 種類のスタートアッププログラムが用意されています。

- start.ms(**M3T-TW32R** 使用時は、start.s)
  アセンブリ言語を使って、プログラムを作成した時に使用するスタートアッププログラムです。

- crt0mr.ms(**M3T-TW32R** 使用時は、crt0mr.s)
  C 言語を使って、プログラムを作成した時に使用するスタートアッププログラムです。
  “start.ms”に C 言語の初期化ルーチンを追加したものです。

スタートアッププログラムは以下のようなことを行っています。

- スタックポインタの設定
- リセット後のプロセッサの初期化
- EIT ベクタエントリ、プログラム領域などの ROM から RAM への転送
- C 言語の変数の初期化(crt0mr.ms、crt0mr.s のみ)
- システムクロックの設定
- MR32R のデータ領域の初期化

このスタートアッププログラムは、環境変数 "LIB32R"の示すディレクトリからカレントディレクトリへコピーして下さい[73]。なお、必要があればスタートアップファイルを修正、あるいは追加して下さい。

[73] カレントディレクトリにスタートアッププログラムがない場合、コンフィグレータ cfg32r を"-m"オプション付きで実行すると、環境変数``LIB32R''の示すディレクトリからカレントディレクトリへコピーされます。

8.2.1 ROM から RAM へのデータ転送について

MR32R では、"mr32r.inc"ファイルで定義されている"DOWNLOAD"マクロを利用し、ROM から RAM への転送処理を行うことができます。製品添付のスタートアップルーチンでは、初期値ありのデータ領域と MR_ROM(または.MR_ROM)セクションの転送処理を行っています。

ダウンロード用のマクロを使用した場合のセクションファイル(section,m32r.cmd)の変更方法・記述方法については次節で説明します。

● M3T-CC32R を使用している場合

```
DOWNLOAD      reg1,reg2,reg3,reg4,section,ROM_section
```

reg1,reg2,reg3 には、マクロ内で使用するレジスタを指定します。
section には、ダウンロードするセクション名を指定し,ROM_section には、前述の section に指定したセクション名の頭に ROM_をつけたものを指定します。
以下に R0,R1,R2,R3 レジスタを使用し、DATA セクションを転送する場合のマクロ記述を示します[74]。

```
DOWNLOAD      R0,R1,R2,R3,DATA,ROM_DATA
```

● M3T-TW32R を使用している場合

```
DOWNLOAD reg1,reg2,reg3,reg4,start_section,end_section,rom_section
```

reg1,reg2,reg3,reg4 には、マクロ内で使用するレジスタを指定します。start_section には、ダウンロード先の先頭アドレス、end_section には、ダウンロード先の最後のアドレス、rom_section には、ダウンロード元の先頭アドレスを指定します。通常は、これらのアドレスはリンカスクリプトファイル`m32r.cmd'内で設定するシンボルを使用します。m32r.cmd 内で以下のように記述された.data 領域を R0,R1,R2,R3 レジスタを使用し、転送する場合のマクロ記述をしめします。

```
.data (ADDR(.MR_RAM)+SIZEOF(.MR_RAM)) : AT( LOADADDR(.rodata) + SIZEOF(.rodata) )
{
  __data_start = .;
  *(.data)
  *(.gnu.linkonce.d*)
  CONSTRUCTORS
  __data_end = .;
}
__rom_data = LOADADDR(.data);
```

```
DOWNLOAD      R0,R1,R2,R3,__data_start,__data_end,__rom_data
```

[74] この場合、ROM_DATA セクションは、リンカによって自動的に生成されるセクションで、スタートアップファイルでセクションの定義を行う必要があります。

● D-CC/M32R を使用している場合

DOWNLOAD reg1,reg2,reg3,reg4,start_section,end_section,rom_section

reg1,reg2,reg3,reg4 には、マクロ内で使用するレジスタを指定します。
start_section には、ダウンロード先の先頭アドレス、end_section には、ダウンロード先の最後のアドレス、rom_section には、ダウンロード元の先頭アドレスを指定します。
通常は、これらのアドレスはリンカスクリプトファイル`m32r.cmd'内で設定するシンボルを使用します。
m32r.cmd 内で以下のように記述された.data 領域を R0,R1,R2,R3 レジスタを使用し、転送する場合のマクロ記述をしめします。

```
__START_data = .;
  .data LOAD(__ROM_data): {
    *(.data)
  }
  __END_data = .;

   DOWNLOAD  R0,R1,R2,R3,__START_data,__END_data,__ROM_data
```

8.2.2 ROM から RAM への転送を止める方法について

ROM から RAM に転送する必要のないセクションは、転送処理を解除します。転送する必要のあるセクション、転送する必要のないセクションは下記のとおりです。

転送する必要のあるセクション

- D(.sdata,.data) **初期値ありのデータ領域**

転送する必要のないセクション

- P(.text) **ユーザプログラム領域**
- C(.rodata) **定数データ領域**
- MR_KERNEL(.MR_KERNEL) **OS カーネル領域**

転送が必要かどうかは条件による場合

- MR_ROM(.MR_ROM) **OS データ領域**
  タスクや OS 資源の動的生成削除を使用する場合は、ROM から RAM への転送処理が必要になります。
- INTERRUPT_VECTOR(.INTERRUPT_VECTOR) **割り込みベクタテーブル領域**
  アドレスが固定され、そのアドレスが ROM 領域に割り込みベクタテーブルを配置しなければならない場合、転送処理を削除しなければいけません。
- EIT_Vector(.EIT_Vector) **EIT ベクタエントリ領域**
  EIT ベクタ領域が ROM の場合、転送処理を削除しなければいけません。
- Int_Vector(.Int_Vector) **割り込みベクタエントリ領域**
  Int_Vector ベクタ領域が ROM の場合、転送処理を削除しなければいけません。

転送処理を解除するには以下の手順で行います。

1. M3T-CC32R を使用する場合は、スタートアッププログラムの不要となったセクションを削除します。(削除するセクションは、ROM_P,ROM_C など転送しないセクション名の頭に ROM_がついたセクションです。)
2. スタートアップファイルに記述されている転送しないセクションの部分の DOWNLOAD マクロを削除します。
3. セクションファイル(section)、リンカスクリプト(m32r.cmd)で転送用の記述をしてある部分を変更します。(変更方法については、各クロスツールのマニュアルを参照してください。

8.2.3 C 言語用(M3T-CC32R 対応)スタートアッププログラム(crt0mr.ms)

図 8.2に非ラージモデルC言語用(M3T-CC32R対応)スタートアッププログラムcrt0mr.msの詳細を示します。

```
 1 ;***********************************************************************
 2 ;
 3 ;   MR32R start up program for C language (for CC32R)
 4 ;   COPYRIGHT(C) 2001(2003) RENESAS TECHNOLOGY CORPORATION
 5 ;   AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
 6 ;
 7 ;
 8 ; $Id: crt0mr.ms,v 1.5 2001/04/23 07:05:17 inui Exp $
 9 ;
10 ;***********************************************************************
11 ;OSを組み込むのに必要なファイルをインクルードします。
12 ;この部分は変更しないでください。
13
14      .include        "mr32r.inc"
15      .include        "sys_rom.inc"
16      .include        "sys_ram.inc"
17      .include        "mrtable.inc"
18
19      .global         __Sys_Sp,__sys_timer,__int_entry,__START
20      .global         __D_INIT_START,__sys_timer_init,__start_up_exit
21      .global         __TOP_USER_STACK,$_init_usr_memblk,$usr_getmem
22      .global         __SYSCALL0,__SYSCALL1,__TOP_HEAP,$_init_memblk
23      .global         __set_ipl
24      .section        B,data
25      .section        D,data
26      .section        P,code
27      .section        C,data
28      .section        ROM_D,data
29      .section        ROM_INTERRUPT_VECTOR,data
30      .section        INTERRUPT_VECTOR,data
31      .section        MR_HEAP,data,align=4
32      .section        EXT_MR_HEAP,data,align=4
33      .section        INT_USR_STACK,data,align=4
34      .section        EXT_USR_STACK,data,align=4
35      .section        EXT_MR_RAM,data,align=4
36      .section        MR_KERNEL2,code,align=4
37      .section        MR_KERNEL,code,align=4
38      .section        ROM_MR_ROM,data,align=4
39      .section        MR_ROM,data,align=4
40      .section        OS_DEBUG,code,align=4
41
42 ;スタートアップファイルのセクション定義します。
43
44      .section        START_UP,code,align=4
45 NULL:        .EQU    0
46
47 ;ベースレジスタ機能を使用する場合、各シンボルに値を設定してください。
48 ;ベースレジスタ機能を使用しない場合、使用しないレジスタについても
49 ;ダミーで値を設定する必要があります。
50
51      .global __REL_BASE11
52      .global __REL_BASE12
53      .global __REL_BASE13
54      .global __REL_BASE
55 __REL_BASE11:        .equ    0
56 __REL_BASE12:        .equ    0
57 __REL_BASE:  .equ    0
58 __REL_BASE13:        .equ    __REL_BASE
59
60 ;ここからスタートアッププログラムを開始します。
61
62 __START:
63
```

```
64 ;システムスタックポインタの設定を行います。
65
66      seth    r1,#high(__Sys_Sp)
67      or3     r1,r1,#low(__Sys_Sp)
68      addi    r1,#-4
69      mvtc    r1,SPI                ;SPI initialize
70      mvtc    r1,SPU
71      ldi     r0,#-1
72      st      r0,@r1
73      ldi     r0,#NULL
74      mvtc    r0,PSW                ;PSW initialize
75
76 ;ベースレジスタ機能を使用し、スタートアップルーチンからC言語で記述された関数を呼び
77 ;出す場合、R11～R13レジスタにベースレジスタの設定を行います。
78
79 ;    seth    R11,#high(__REL_BASE11)
80 ;    or3     R11,R11,#low(__REL_BASE11)
81 ;    seth    R12,#high(__REL_BASE12)
82 ;    or3     R12,R12,#low(__REL_BASE12)
83 ;    seth    R13,#high(__REL_BASE13)
84 ;    or3     R13,R13,#low(__REL_BASE13)
85
86      .AIF    ¥&__Dbg_flg gt 0
87      ld24    r1,#__Dbg_mode
88 ;    ldi     r2,#0
89      ldi     r2,#4
90      stb     r2,@r1
91      .AENDI
92
93 ;マイコンの動作モードの設定などを行います。
94 ;
95 ;    Description below,if you need
96 ;    (ex. Master/Slavemode initialize,Power management initialize)
97 ;
98
99 ;初期値なしのデータをゼロクリアします。
100 ;このマクロは、"mr32r.inc"で定義されており、ユーザも自由に使用することができます。
101
102     RAM_CLEAR r0,r1,r2,B
103
104 ;外部ROM領域から内蔵RAM領域へ転送します。
105 ;このマクロもユーザが自由に使用することができます。
106
107     DOWNLOAD r0,r1,r2,r3,D,ROM_D
108
109 ;C言語の標準関数の初期化を行います。
110 ;
111
112 ;Initialize standard library
113 ;
114
115 ;    .global __init_mem,__init_stdio
116 ;    bl      __init_mem
117 ;    bl      __init_stdio
118
119     .global __OS_INIT
120
121 ;OSの初期化を行います。
122 __OS_INIT:
123
124     DOWNLOAD r0,r1,r2,r3,MR_ROM,ROM_MR_ROM
125     DOWNLOAD r0,r1,r2,r3,INTERRUPT_VECTOR,ROM_INTERRUPT_VECTOR
126 ;
127 ;Initialize OS system area
128 ;
129
130     INITIALIZE
131
132 ;OSデバッグ機能使用時の初期化をします。;
133
```

```
134     .AIF    ¥&__Dbg_flg gt 0
135     ld24    r1,#__Dbg_buffer_start
136     ld24    r0,#__Dbg_addr
137     st      r1,@r0
138     ld24    r1,#__Dbg_cnt
139     ldi     r0,#0
140     st      r0,@r1
141     ld24    r1,#__Dbg_mode
142     ldb     r0,@r1
143     ldi     r2,#1
144     or      r0,r2
145     stb     r0,@r1
146     .aendi
147
148 ;'ipl.ms'内の__set_iplを呼び出し、割り込み優先レベルの初期設定を
149 ;行います。
150
151     bl      __set_ipl
152
153 ;タイマを使用する場合、'ipl.ms'内の__sys_timer_initを呼び出し、
154 ;タイマの初期化を行います。
155
156     .AIF    ¥&USE_TIMER gt 0
157     bl      __sys_timer_init
158     .AENDI
159
160 ;    .global __init_abort
161 ;    bl      __init_abort
162     ldi     r4,#2
163
164 ;
165 ;Start task
166 ;
167
168 ;初期起動タスクを起動します。
169 ;この部分は変更しないでください。
170
171 start_task:
172     ld24    r5,#__D_INIT_START-2
173     add     r5,r4
174     lduh    r1,@r5
175     beqz    r1,start_task_end
176     ldi     r2,#NULL
177     ldi     r0,#ISTA_TSK
178     TRAP    #8
179     addi    r4,#2
180     bra     start_task
181
182 start_task_end:
183     ldi     r5,#NULL
184     ld24    r4,#__enq_dsp
185     stb     r5,@r4
186     bra     __start_up_exit
187
188 ;この部分は変更しないでください。
189
190     .AIF    ¥&__Dbg_flg eq 0
191     .section        MR_KERNEL,code,align=4
192     .align  4
193     .global __Dbg_idle
194     .global __Dbg_RUNtsk,__Dbg_int_entry,__Dbg_int_exit
195     .global __Dbg_ent_handler,__Dbg_ext_handler,__Dbg_int_exit2
196     .global __Dbg_sys_exit,__Dbg_sys_exit2,__Dbg_sys_timer
197 __Dbg_idle:
198 __Dbg_RUNtsk:
199 __Dbg_int_entry:
200 __Dbg_int_exit:
201 __Dbg_int_exit2:
202 __Dbg_ent_handler:
203 __Dbg_ext_handler:
204 __Dbg_sys_exit:
```

```
205 __Dbg_sys_exit2:
206 __Dbg_sys_timer:
207     .AENDI
208
209     .aif ¥&__MR_EXC_HANDLER
210     .global __DEF_EXC_HDR
211 __DEF_EXC_HDR:
212     vret_exc
213     .aendi
214
215     .AIF    ¥&__Dbg_flg gt 0
216     .section        MR_Dbg_RAM,data
217     .align  4
218     .global __Dbg_mode
219     .global __Dbg_buffer_start
220     .global __Dbg_buffer_end
221     .global __Dbg_cnt
222     .global __Dbg_addr
223     .global __Dbg_sys_iss
224 __Dbg_buffer_start:
225     .res.b  __Dbg_buffer_size
226 __Dbg_buffer_end:
227 __Dbg_addr:  .res.w  1
228 __Dbg_cnt:   .res.w  1
229 __Dbg_sys_iss:       .res.w  7
230 __Dbg_mode:  .res.b  1
231     .aelse
232     .section        MR_Dbg_RAM,data
233     .align  4
234     .global __Dbg_mode
235     .global __Dbg_buffer_start
236     .global __Dbg_buffer_end
237     .global __Dbg_cnt
238     .global __Dbg_addr
239     .global __Dbg_sys_iss
240 __Dbg_buffer_start:
241 __Dbg_buffer_end:
242 __Dbg_addr:
243 __Dbg_cnt:
244 __Dbg_mode:
245 __Dbg_sys_iss:
246     .aendi
247
248 ;***************  EIT Vector AREA  ********************
249
250     .section        EIT_Vector,CODE,ALIGN=4
251     bra     SBI_hdr:24              ;SBI    H'20
252     .RES.W  3
253     bra     RIE_hdr:24              ;RIE    H'30
254     .RES.W  3
255     bra AE_hdr:24           ;AE             H'40
256     .RES.W  3
257     rte                             ;TRAP0
258     nop
259     rte                             ;TRAP1
260     nop
261     rte                             ;TRAP2
262     nop
263     rte                             ;TRAP3
264     nop
265     rte                             ;TRAP4
266     nop
267     rte                             ;TRAP5
268     nop
269     rte                             ;TRAP6
270     nop
271     .AIF    ¥&__Dbg_flg eq 0
272     bra     __SYSCALL0              ;TRAP7
273     bra     __SYSCALL1              ;TRAP8
274     .AELSE
275     .global __Dbg_entry0,__Dbg_entry1
276     bra     __Dbg_entry0            ;TRAP7
277     bra     __Dbg_entry1            ;TRAP8
```

```
278     .AENDI
279     rte                             ;TRAP9
280     nop
281     rte                             ;TRAP10
282     nop
283     rte                             ;TRAP11
284     nop
285     rte                             ;TRAP12
286     nop
287     rte                             ;TRAP13
288     nop
289     rte                             ;TRAP14
290     nop
291     rte                             ;TRAP15
292     nop
293     .section        Int_Vector,code,align=4
294     bra     __int_entry
295
296     .SECTION        RESET_VECT,code,align=4
297     bra     __START
298     .section        MR_KERNEL,code,align=4
299 SBI_hdr:
300     bra     SBI_hdr
301 RIE_hdr:
302     bra     RIE_hdr
303 AE_hdr:
304     bra_AE_hdr
305     .end
```

図 8.2 C 言語用スタートアッププログラム(M3T-CC32R 対応)

8.2.4 C 言語用(M3T-TW32R 対応)スタートアッププログラム(crt0mr.s)

図 8.3に C 言語用(M3T-TW32R 対応)スタートアッププログラム crt0mr.s の詳細を示します。

```
 1 ;************************************************************************
 2 ;
 3 ;   MR32R start up program for C language (for TW32R)
 4 ;   COPYRIGHT(C) 2001(2003) RENESAS TECHNOLOGY CORPORATION
 5 ;   AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
 6 ;
 7 ;
 8 ; $Id: crt0mr.s,v 1.4 2001/04/23 07:05:18 inui Exp $
 9 ;
10 ;************************************************************************
11 ;OSを組み込むのに必要なファイルをインクルードします。
12 ;この部分は変更しないでください。
13
14      .include        "mr32r.inc"
15      .include        "sys_rom.inc"
16      .include        "sys_ram.inc"
17      .include        "mrtable.inc"
18
19      .global         __Sys_Sp,__sys_timer,__START
20      .global         __D_INIT_START,__sys_timer_init,__start_up_exit
21      .global         __TOP_USER_STACK,_init_usr_memblk,usr_getmem
22      .global         __SYSCALL0,__SYSCALL1,__TOP_HEAP,_init_memblk
23
24 ;スタートアップファイルのセクション定義します。
25
26      .section        .STARTUP,"awx"
27      .balign         4
28      .equ            NULL,0
29
30 ;ここからスタートアッププログラムを開始します。
31
32 __START:
33
34 ;システムスタックポインタの設定を行います。
35
36      seth    r1,#high(__Sys_Sp)
37      or3     r1,r1,#low(__Sys_Sp)
38      addi    r1,#-4
39      mvtc    r1,CR2                  ;SPI initialize
40      mvtc    r1,CR3
41      ldi     r0,#-1
42      st      r0,@r1
43      ldi     r0,#NULL
44      mvtc    r0,CR0                  ;PSW initialize
45
46      .if     __Dbg_flg
47      ld24    r1,#__Dbg_mode
48 ;    ldi     r2,#0
49      ldi     r2,#4
50      stb     r2,@r1
51      .endif
52
53 ;マイコンの動作モードの設定などを行います。
54
55 ;
56 ;    Description below,if you need
57 ;    (ex. Master/Slavemode initialize,Power management initialize)
58 ;
59 ;
60 ;    Initialize section
61 ;
62
63 ;初期値なしのデータをゼロクリアします。
64 ;このマクロは、"mr32r.inc"で定義されており、ユーザも自由に使用することができます。
65
```

```
 66      RAM_CLEAR R0,R1,R2,__sbss_start,__sbss_end
 67      RAM_CLEAR R0,R1,R2,__bss_start,__bss_end
 68
 69
 70 ;外部ROM領域から内蔵RAM領域へ転送します。
 71 ;このマクロもユーザが自由に使用することができます。
 72
 73      DOWNLOAD  R0,R1,R2,R3,__mr_rom_start,__mr_rom_end,__rom_mr_rom
 74      DOWNLOAD  R0,R1,R2,R3,__data_start,__data_end,__rom_data
 75      DOWNLOAD  R0,R1,R2,R3,__sdata_start,__sdata_end,__rom_sdata
 76
 77 ;C言語の標準関数の初期化を行います。
 78
 79 ;
 80 ;    initialize C language library
 81 ;
 82 ;    .global _init_mem,_init_stdio,__OS_INIT
 83 ;    bl      _init_mem
 84 ;    bl      _init_stdio
 85
 86 ;OSの初期化を行います。
 87
 88 __OS_INIT:
 89 ;
 90 ;Initialize OS system area
 91 ;
 92      INITIALIZE
 93
 94 ;OSデバッグ機能使用時の初期化をします。
 95
 96      .if     __Dbg_flg
 97      ld24    r1,#__Dbg_buffer_start
 98      ld24    r0,#__Dbg_addr
 99      st      r1,@r0
100      ld24    r1,#__Dbg_cnt
101      ldi     r0,#0
102      st      r0,@r1
103      ld24    r1,#__Dbg_mode
104      ldb     r0,@r1
105      ldi     r2,#1
106      or      r0,r2
107      stb     r0,@r1
108      .endif
109
110 ;'ipl.ms'内の__set_iplを呼び出し、割り込み優先レベルの初期設定を
111 ;行います。
112
113      bl      __set_ipl
114
115 ;タイマを使用する場合、'ipl.ms'内の__sys_timer_initを呼び出し、
116 ;タイマの初期化を行います。
117
118      .ifdef  USE_TIMER
119      bl      __sys_timer_init
120      .endif
121
122 ;    bl      __init_abort
123
124 ;
125 ;Start task
126 ;
127
128 ;初期起動タスクを起動します。
129 ;この部分は変更しないでください。
130
131      ldi     R4,#2
132 start_task:
133      ld24    R5,#__D_INIT_START
134      add     R5,R4
```

```
135     lduh    R1,@(-2,R5)
136     beqz    R1,start_task_end
137     ldi     R2,#NULL
138     ldi     R0,#ISTA_TSK
139     TRAP    #8
140     addi    R4,#2
141     bra     start_task
142 start_task_end:
143     ldi     R5,#NULL
144     ld24    R4,#__enq_dsp
145     stb     R5,@R4
146     bra     __start_up_exit
147
148 ;この部分は変更しないでください。
149
150     .if !(__Dbg_flg )
151     .section        .MR_KERNEL,"ax"
152     .align  4
153     .global __Dbg_idle
154     .global __Dbg_RUNtsk,__Dbg_int_entry,__Dbg_int_exit
155     .global __Dbg_ent_handler,__Dbg_ext_handler,__Dbg_int_exit2
156     .global __Dbg_sys_exit,__Dbg_sys_exit2,__Dbg_sys_timer
157 __Dbg_idle:
158 __Dbg_RUNtsk:
159 __Dbg_int_entry:
160 __Dbg_int_exit:
161 __Dbg_int_exit2:
162 __Dbg_ent_handler:
163 __Dbg_ext_handler:
164 __Dbg_sys_exit:
165 __Dbg_sys_exit2:
166 __Dbg_sys_timer:
167     .endif
168
169     .if     __Dbg_flg
170     .section        MR_Dbg_RAM,"aw"
171     .align  4
172     .global __Dbg_mode
173     .global __Dbg_buffer_start
174     .global __Dbg_buffer_end
175     .global __Dbg_cnt
176     .global __Dbg_addr
177 __Dbg_buffer_start:
178     .space  __Dbg_buffer_size
179 __Dbg_buffer_end:
180 __Dbg_addr:  .space  1*4
181 __Dbg_cnt:   .space  1*4
182 __Dbg_sys_iss:       .space  7*4
183 __Dbg_mode:  .space  1
184     .else
185     .section        .MR_Dbg_RAM,"aw"
186     .align  4
187     .global __Dbg_mode
188     .global __Dbg_buffer_start
189     .global __Dbg_buffer_end
190     .global __Dbg_cnt
191     .global __Dbg_addr
192     .global __Dbg_sys_iss
193 __Dbg_buffer_start:
194 __Dbg_buffer_end:
195 __Dbg_addr:
196 __Dbg_cnt:
197 __Dbg_mode:
198 __Dbg_sys_iss:
199     .endif
200 ;***************  EIT Vector AREA ****************************************
    ****
201
202     .section        .EIT_Vector,"awx"
203     .balign         4
204     bra     SBI_hdr:24      ;SBI H'10
205     .space  12
206     bra     RIE_hdr:24      ;RIE H'20
```

```
207     .space  12
208     bra     AE_HDR:24       ;AE H'30
209     .space  12
210     rte                             ;TRAP0
211     nop
212     rte                             ;TRAP1
213     nop
214     rte                             ;TRAP2
215     nop
216     rte                             ;TRAP3
217     nop
218     rte                             ;TRAP4
219     nop
220     rte                             ;TRAP5
221     nop
222     rte                             ;TRAP6
223     nop
224     .if !(__Dbg_flg )
225     bra     __SYSCALL0              ;TRAP7
226     bra     __SYSCALL1              ;TRAP8
227     .else
228     .global  __Dbg_entry0,__Dbg_entry1
229     bra     __Dbg_entry0
230     bra     __Dbg_entry1
231     .endif
232     rte                             ;TRAP9
233     nop
234     rte                             ;TRAP10
235     nop
236     rte                             ;TRAP11
237     nop
238     rte                             ;TRAP12
239     nop
240     rte                             ;TRAP13
241     nop
242 ;   rte                             ;TRAP14
243 ;   nop
244 ;   rte                             ;TRAP15
245 ;   nop
246     .section        .Int_Vector,"awx"
247     .balign         4
248     .global __int_entry
249     bra     __int_entry
250
251     .section        .MR_KERNEL,"ax"
252
253 .if  __MR_EXC_HANDLER
254     .global __DEF_EXC_HDR
255 __DEF_EXC_HDR:
256     vret_exc
257 .endif
258 SBI_hdr:
259     bra     SBI_hdr
260 RIE_hdr:
261     bra     RIE_hdr
262 AE_hdr:
263     bra_AE_hdr
264
265     .section        .RESET_VECT,"ax"
266     .balign         4
267 reset:
268     bra     __START
```

図 8.3　C 言語用スタートアッププログラム(M3T-TW32R 対応)

8.2.5 C 言語用(D-CC/M32R 対応)スタートアッププログラム(crt0mr.s)

図 8.4に C 言語用(D-CC/M32R 対応)スタートアッププログラム crt0mr.s の詳細を示します。

```
 1 ;***********************************************************************
 2 ;
 3 ;   MR32R start up program for C language (for DCC/M32R)
 4 ;   COPYRIGHT(C) 2001(2003) RENESAS TECHNOLOGY CORPORATION
 5 ;   AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
 6 ;
 7 ;
 8 ; $Id: crt0mr.s,v 1.3 2001/04/23 07:05:17 inui Exp $
 9 ;
10 ;***********************************************************************
11 ;OSを組み込むのに必要なファイルをインクルードします。
12 ;この部分は変更しないでください。
13
14      .include        "mr32r.inc"
15      .include        "sys_rom.inc"
16      .include        "sys_ram.inc"
17      .include        "mrtable.inc"
18
19      .global         __Sys_Sp,__sys_timer,__START
20      .global         __D_INIT_START,__sys_timer_init,__start_up_exit
21      .global         __TOP_USER_STACK,_init_usr_memblk,usr_getmem
22      .global         __SYSCALL0,__SYSCALL1,__TOP_HEAP,_init_memblk
23
24 ;スタートアップファイルのセクション定義します。
25
26      .section        .STARTUP,"awx"
27      .balign         4
28      .equ            NULL,0
29
30 ;ここからスタートアッププログラムを開始します。
31
32 __START:
33
34 ;システムスタックポインタの設定を行います。
35
36      seth    r1,#__Sys_Sp@h
37      or3     r1,r1,#__Sys_Sp@l
38      addi    r1,#-4
39      mvtc    r1,CR2                  ;SPI initialize
40      mvtc    r1,CR3
41      ldi     r0,#-1
42      st      r0,@r1
43      ldi     r0,#NULL
44      mvtc    r0,CR0                  ;PSW initialize
45
46      .if     __Dbg_flg
47      ld24    r1,#__Dbg_mode
48 ;    ldi     r2,#0
49      ldi     r2,#4
50      stb     r2,@r1
51      .endif
52
53 ;マイコンの動作モードの設定などを行います。
54
55 ;
56 ;    Description below,if you need
57 ;    (ex. Master/Slavemode initialize,Power management initialize)
58 ;
59 ;
60 ;    Initialize section
61 ;
62
63 ;初期値なしのデータをゼロクリアします。
64 ;このマクロは、"mr32r.inc"で定義されており、ユーザも自由に使用することができます。
65
```

```
 66      RAM_CLEAR R0,R1,R2,__START_bss,__END_bss
 67
 68
 69 ;外部ROM領域から内蔵RAM領域へ転送します。
 70 ;このマクロもユーザが自由に使用することができます。
 71
 72      RAM_CLEAR R0,R1,R2,__START_bss,__END_bss
 73      DOWNLOAD  R0,R1,R2,R3,__START_MR_ROM,__END_MR_ROM,__ROM_MR_ROM
 74      DOWNLOAD  R0,R1,R2,R3,__START_data,__END_data,__ROM_data
 75
 76 ;C言語の標準関数の初期化を行います。
 77
 78 ;
 79 ;    initialize C language library
 80 ;
 81
 82 ;OSの初期化を行います。
 83
 84 __OS_INIT:
 85 ;
 86 ;Initialize OS system area
 87 ;
 88      INITIALIZE
 89
 90 ;OSデバッグ機能使用時の初期化をします。
 91
 92      .if     __Dbg_flg
 93      ld24    r1,#__Dbg_buffer_start
 94      ld24    r0,#__Dbg_addr
 95      st      r1,@r0
 96      ld24    r1,#__Dbg_cnt
 97      ldi     r0,#0
 98      st      r0,@r1
 99      ld24    r1,#__Dbg_mode
100      ldb     r0,@r1
101      ldi     r2,#1
102      or      r0,r2
103      stb     r0,@r1
104      .endif
105
106 ;'ipl.ms'内の__set_iplを呼び出し、割り込み優先レベルの初期設定を
107 ;行います。
108
109      bl      __set_ipl
110
111 ;タイマを使用する場合、'ipl.ms'内の__sys_timer_initを呼び出し、
112 ;タイマの初期化を行います。
113
114      .ifdef  USE_TIMER
115      bl      __sys_timer_init
116      .endif
117
118 ;    bl      __init_abort
119
120 ;
121 ;Start task
122 ;
123
124 ;初期起動タスクを起動します。
125 ;この部分は変更しないでください。
126
127      ldi     R4,#2
128 start_task:
129      ld24    R5,#__D_INIT_START
130      add     R5,R4
131      lduh    R1,@(-2,R5)
132      beqz    R1,start_task_end
133      ldi     R2,#NULL
134      ldi     R0,#ISTA_TSK
```

```
135     TRAP    #8
136     addi    R4,#2
137     bra     start_task
138 start_task_end:
139     ldi     R5,#NULL
140     ld24    R4,#__enq_dsp
141     stb     R5,@R4
142     bra     __start_up_exit
143
144 ;ダミーの関数です。
145 ;リンクするライブラリによって、このシンボルが必要となる場合があります。
146     .global __init
147 __init:
148     jmp     r14
149
150 ;この部分は変更しないでください。
151
152     .if !(__Dbg_flg )
153     .section        .MR_KERNEL,"ax"
154     .align  4
155     .global __Dbg_idle
156     .global __Dbg_RUNtsk,__Dbg_int_entry,__Dbg_int_exit
157     .global __Dbg_ent_handler,__Dbg_ext_handler,__Dbg_int_exit2
158     .global __Dbg_sys_exit,__Dbg_sys_exit2,__Dbg_sys_timer
159 __Dbg_idle:
160 __Dbg_RUNtsk:
161 __Dbg_int_entry:
162 __Dbg_int_exit:
163 __Dbg_int_exit2:
164 __Dbg_ent_handler:
165 __Dbg_ext_handler:
166 __Dbg_sys_exit:
167 __Dbg_sys_exit2:
168 __Dbg_sys_timer:
169     .endif
170
171     .if     __Dbg_flg
172     .section        MR_Dbg_RAM,"aw"
173     .align  4
174     .global __Dbg_mode
175     .global __Dbg_buffer_start
176     .global __Dbg_buffer_end
177     .global __Dbg_cnt
178     .global __Dbg_addr
179 __Dbg_buffer_start:
180     .space  __Dbg_buffer_size
181 __Dbg_buffer_end:
182 __Dbg_addr:  .space  1*4
183 __Dbg_cnt:   .space  1*4
184 __Dbg_sys_iss:       .space  7*4
185 __Dbg_mode:  .space  1
186     .else
187     .section        .MR_Dbg_RAM,"aw"
188     .align  4
189     .global __Dbg_mode
190     .global __Dbg_buffer_start
191     .global __Dbg_buffer_end
192     .global __Dbg_cnt
193     .global __Dbg_addr
194     .global __Dbg_sys_iss
195 __Dbg_buffer_start:
196 __Dbg_buffer_end:
197 __Dbg_addr:
198 __Dbg_cnt:
199 __Dbg_mode:
200 __Dbg_sys_iss:
201     .endif
202 ;***************  EIT Vector AREA ****************************************
203
204     .section        .EIT_Vector,"awx"
205     .align          4
206     bra     SBI_hdr:24      ;SBI H'10
```

```
207     .space  12
208     bra     RIE_hdr:24      ;RIE H'20
209     .space  12
210     bra     AE_HDR:24       ;AE H'30
211     .space  12
212     rte                             ;TRAP0
213     nop
214     rte                             ;TRAP1
215     nop
216     rte                             ;TRAP2
217     nop
218     rte                             ;TRAP3
219     nop
220     rte                             ;TRAP4
221     nop
222     rte                             ;TRAP5
223     nop
224     rte                             ;TRAP6
225     nop
226     .if !(__Dbg_flg )
227     bra     __SYSCALL0              ;TRAP7
228     bra     __SYSCALL1              ;TRAP8
229     .else
230     .global  __Dbg_entry0,__Dbg_entry1
231     bra     __Dbg_entry0
232     bra     __Dbg_entry1
233     .endif
234     rte                             ;TRAP9
235     nop
236     rte                             ;TRAP10
237     nop
238     rte                             ;TRAP11
239     nop
240     rte                             ;TRAP12
241     nop
242     rte                             ;TRAP13
243     nop
244 ;   rte                             ;TRAP14
245 ;   nop
246 ;   rte                             ;TRAP15
247 ;   nop
248     .section        .Int_Vector,"awx"
249     .align          4
250     .global __int_entry
251     bra     __int_entry
252
253     .section        .MR_KERNEL,"ax"
254
255 .if  __MR_EXC_HANDLER
256     .global __DEF_EXC_HDR
257 __DEF_EXC_HDR:
258     vret_exc
259 .endif
260
261 SBI_hdr:
262     bra     SBI_hdr
263 RIE_hdr:
264     bra     RIE_hdr
265 AE_hdr:
266     bra_AE_hdr
267
268     .section        .RESET_VECT,"ax"
269     .align          4
270 reset:
271     bra     __START;
```

図 8.4　C 言語用スタートアッププログラム(D-CC/M32R 対応)

8.3 セクションファイルのカスタマイズ

8.3.1 M3T-CC32R を使用する場合

アプリケーションプログラムのデータのメモリ配置方法について説明します。

メモリ配置を設定するためには、MR32R が提供しているセクションファイル(section)で設定します。

このファイルは、リンカのセクション配置オプションとしてリンカに渡されます。

ユーザのシステムに合わせて、セクション配置、開始アドレスの設定を変更して下さい。[75]

通常は、-SEC の後にセクション名を記述していきます。外部 ROM から内蔵 DRAM に転送する場合は、転送先セクション名の頭に@をつけ、転送元のセクション名はそのまま記述します。

次に例を示します。なお、図中の矢印は、矢印方向へ(ROM から RAM へ)転送することを表します。

```
-SEC
RESET_VECTOR,EIT_Vector=10,Int_Vector=80,MR_KERNEL=100,MR_KERNEL2,START_UP,OS_DEBUG,P,D,C,
MR_ROM,INTERRUPT_VECTOR,MR_RAM=0F00000,@MR_ROM,@INTERRUPT_VECTOR,MR_Dbg_RAM,SYS_STACK,INT_
USR_STACK,MR_HEAP,B,D,EXT_MR_RAM=1000000,EXT_USR_STACK,EXT_MR_HEAP
```

[75] 詳細は、CC32R ユーザーズマニュアル セクション結合機能をご覧ください。

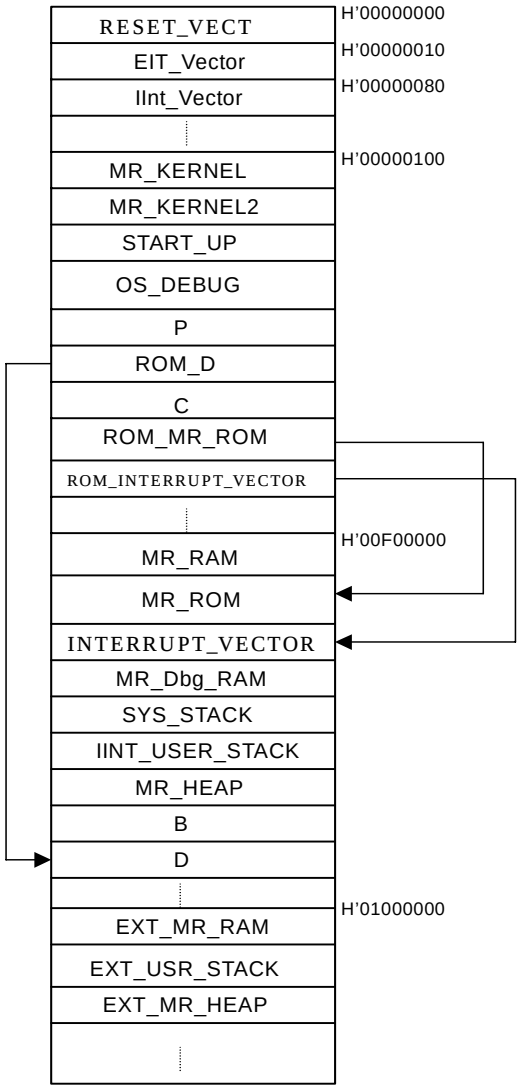


図 8.5　サンプルセクションファイルのメモリ配置

8.3.2 M3T-TW32R を使用する場合

メモリ配置を設定するためには、MR32R が提供しているリンカスクリプト(m32r.cmd)で設定します。このファイルは、リンカ(m32r-elf-ld)実行時に利用されます。ここでは、MR32R で定義されているシンボル・および製品に添付しているサンプルファイルの内容について説明します。なお、リンカスクリプトの記述方法の詳細については、**M3T-TW32R** に付属している「ユーザーズガイド」またはリンカ(m32r-elf-ld)に関するオンラインドキュメントをご覧ください。

サンプルリンカスクリプトのメモリイメージは、図 7.3 で示す M3T-CC32R 用のものと同じになります。

リンカスクリプトで定義されているシンボル

- EIT_OF_RAM
  TRAP で使用する領域の開始番地を表わしています。この値を変更することはありません。
- INT_OF_RAM
  割り込みベクタ領域の開始番地を表わしています。
- START_OF_RAM
  内蔵 RAM 領域の先頭をあらわします。
- START_OF_MR_RAM
  MR32R の内蔵 RAM データ領域の開始番地を表わしています。
- START_OF_EXT_OF_RAM
  MR32R の外部 RAM データ領域の開始番地をあらわしています。
- RESET
  リセットベクタの開始番地を表わしています。

8.3.3 サンプルリンカスクリプト(M3T-TW32R 対応)

```
 1 /*
 2  * m32r.cmd -- Linker script for M32R & MTM
 3  * This script is based on `m32r-elf/lib/ldscripts/m32relf.x'.
 4  * @(#) m32r.cmd 1.2@(#) 99/03/16
 5  * $Id: m32r.cmd,v 1.2 2001/04/18 06:24:31 inui Exp $
 6  */
 7
 8 /*[System Calls(You selected)]*/
 9 /*
10  * Users are not allowed to remove and modify the below
11  * blocks because MTM should handle these blocks:
12  *   - MTM_INPUT
13  *   - MTM_SEARCH_DIR
14  *   - MTM_GROUP
15  */
16 /* この部分は、統合化環境TMが使用しますので修正しないでください。  */
17
18 /*[MTM_INPUT_TOP]*/
19 /*[MTM_INPUT_END]*/
20 /*[MTM_SEARCH_DIR_TOP]*/
21
22 /*[MTM_SEARCH_DIR_END]*/
23 /*[MTM_GROUP_TOP]*/
24 /*[MTM_GROUP_END]*/
25
26
27 OUTPUT_FORMAT("elf32-m32r", "elf32-m32r", "elf32-m32r")
28 OUTPUT_ARCH(m32r)
29 ENTRY(__START)
30 /* Do we need any of these for elf?
31    __DYNAMIC = 0;    */
32
33 /* Memory Layout
34  *   This layout is just example.  Applying this example, it
35  *   is able to check a linking process exactly.
36  *
37  */
38 MEMORY
39 {
40     in_rom   : org = 0, len = 4M
41     in_ram1  : org = 0xf00000, len = 64K
42     out_ram1 : org = 0x1000000, len = 4M
43 }
44
45 /* Section Layout
46  *
47  *   <section name>  <placed in>     <placed in while executing>
48  *   .reset          ROM             ROM
49  *   .EIT_Vector     ROM             ROM
50  *   .Int_Vector     ROM             ROM
51  *   .text           ROM             ROM
52  *   .rodata         ROM             ROM
53  *   .data           ROM             RAM
54  *   .sdata          ROM             RAM
55  *   .sbss           RAM             RAM
56  *   .bss            RAM             RAM
57  *   .heap           RAM             RAM
58  *   .spu_stack              RAM             RAM
59  *   .spi_stack              RAM             RAM
60  */
61
62 SECTIONS
63 {
64 /* メモリ配置を決定するシンボルを定義します。*/
65
66   RESET = 0;
67   EIT_OF_ROM = 0x10;
68   INT_OF_ROM = 0x80;
69   START_OF_ROM = 0x100;
```

```
 70   START_OF_RAM = 0xf00000;
 71   START_OF_EXT_RAM = 0x1000000;
 72
 73 /* ロケーションカウンタを RESETにセットします。*/
 74
 75   . = RESET;
 76   /* reset vector sections */
 77   .reset (RESET) :
 78   {
 79      *(.RESET_VECT)
 80   }
 81
 82 /*  EIT_Vectorの配置アドレスをEIT_OF_ROMに定義します。。 */
 83
 84   .EIT_Vector (EIT_OF_ROM) :
 85   {
 86     __eit_start = .;
 87     *(.EIT_Vector)
 88     __eit_end = .;
 89   } = 0
 90
 91   PROVIDE (EIT_Vector = .);
 92
 93 /* .Int_Vectorの配置アドレスをINT_OF_ROMに定義します。*/
 94
 95   .Int_Vector (INT_OF_ROM) :
 96   {
 97     __Int_start = .;
 98     *(.Int_Vector)
 99     __Int_end = .;
100   } = 0
101
102 /* .MR_KERNELの配置アドレスをSTART_OF_ROMに定義します。*/
103
104   .MR_KERNEL (START_OF_ROM) :
105   {
106     __mr_kernel_start = .;
107     *(.MR_KERNEL)
108     *(.MR_KERNEL2)
109     *(.OS_DEBUG)
110     __mr_kernel_end = .;
111   } =0
112
113 /* .STARTUPの配置を定義します。*/
114
115   .STARTUP (.) :
116   {
117     *(.STARTUP)
118   }
119
120 /* .textの配置を定義します。*/
121
122   .text (.) :
123   {
124     __text_start = .;
125     *(.text)
126     /* .gnu.warning sections are handled specially by elf32.em.  */
127     *(.gnu.warning)
128     *(.gnu.linkonce.t*)
129     __text_end = .;
130   } =0
131   _etext = .;
132   PROVIDE (etext = .);
133
134 /* .rodataの配置を定義します。*/
135
136   .rodata (.) :
137   {
138      __rodata_start = .;
139      *(.rodata)
140      *(.gnu.linkonce.r*)
```

```
141     __rodata_end = .;
142   } =0
143
144 /* .MR_RAMの配置アドレスをSTART_OF_MR_RAMに定義します。*/
145
146   .MR_RAM (START_OF_MR_RAM) :
147  {
148     *(.MR_RAM)
149     *(.MR_Dbg_RAM)
150   }
151
152   /* Adjust the address for the data segment.  We want to adjust up to
153      the same address within the page on the next page up.  */
154
155 /* .dataの配置を定義します。*/
156
157   .data (ADDR(.MR_RAM)+SIZEOF(.MR_RAM)) : AT( LOADADDR(.rodata) + SIZEOF(.roda
    ta) )
158   {
159     __data_start = .;
160     *(.data)
161     *(.gnu.linkonce.d*)
162     CONSTRUCTORS
163     __data_end = .;
164   }
165   __rom_data = LOADADDR(.data);
166
167   /* We want the small data sections together, so single-instruction offsets
168      can access them all, and initialized data all before uninitialized, so
169      we can shorten the on-disk segment size.  */
170
171 /* .sdataの配置を定義します。*/
172
173   .sdata (ADDR(.data)+SIZEOF(.data)) : AT( LOADADDR(.data) + SIZEOF(.data) )
174   {
175     __sdata_start = .;
176     *(.sdata)
177     __sdata_end = .;
178   }
179   __rom_sdata = LOADADDR(.sdata);
180
181 /* .MR_ROMの配置を定義します。*/
182
183   .MR_ROM (ADDR(.sdata)+SIZEOF(.sdata)) : AT( LOADADDR(.sdata) + SIZEOF(.sdata
    ) )
184   {
185     __mr_rom_start = .;
186     *(.MR_ROM)
187     *(.INTERRUPT_VECTOR)
188     __mr_rom_end = .;
189   }
190   __rom_mr_rom = LOADADDR(.MR_ROM);
191
192   _edata  =  .;
193   PROVIDE (edata = .);
194
195 /* .SYS_STACK(システムスタック領域)のセクション配置を決定します。  */
196
197   .SYS_STACK (.) :
198   {           /* Interrupt stack sections */
199      __spi_start = .;
200      *(.SYS_STACK)
201      __spi_end = .;
202   }
203
204 /* .INT_USR_STACK(内蔵RAMのユーザスタック領域)のセクション配置を決定します。 */
205
206   .INT_USR_STACK (.)  :
207   {           /* User stack sections */
208      __int_spu_start = .;
209      *(.INT_USR_STACK)
```

```
210     __int_spu_end = .;
211   }
212
213 /* .MR_HEAP(内蔵RAMの可変長メモリプール領域)のセクション配置を決定します。  */
214
215   .MR_HEAP (.)  :
216   {                   /* Heap space sections */
217     __int_mpl_start = .;
218     *(.MR_HEAP)
219     __int_mpl_end = .;
220   }
221
222   .sbss (.) :
223     {
224     __sbss_start = .;
225     *(.sbss)
226     *(.scommon)
227     __sbss_end = .;
228     }
229   .bss (.) :
230     {
231     __bss_start = .;
232     *(.dynbss)
233     *(.bss)
234     *(COMMON)
235     __bss_end = .;
236     }
237   _end = . ;
238   PROVIDE (end = .);
239
240 /* .EXT_MR_RAM(MR32Rの外部RAMデータ領域)のセクション配置を決定します。  */
241
242   .EXT_MR_RAM (START_OF_EXT_RAM) :
243   {
244     *(.EXT_MR_RAM)
245   }
246
247 /* .EXT_USR_STACK(ユーザスタックの外部RAM領域)のセクション配置を決定します。  */
248
249   .EXT_USR_STACK (.)  :
250   {          /* User stack sections */
251     __ext_spu_start = .;
252     *(.EXT_USR_STACK)
253     __ext_spu_end = .;
254   }
255
256 /* .EXT_MR_HEAP(可変長メモリプールの外部RAM領域)のセクション配置を決定します。  */
257
258   .EXT_MR_HEAP (.) :
259   {                   /* Heap space sections */
260     __ext_mpl_start = .;
261     *(.EXT_MR_HEAP)
262     __ext_mpl_end = .;
263   }
264
265 /* デバッグ情報を配置します。この部分は削除しないでください  */
266
267   /* Stabs debugging sections.  */
268   .stab 0 : { *(.stab) }
269   .stabstr 0 : { *(.stabstr) }
270   .stab.excl 0 : { *(.stab.excl) }
271   .stab.exclstr 0 : { *(.stab.exclstr) }
272   .stab.index 0 : { *(.stab.index) }
273   .stab.indexstr 0 : { *(.stab.indexstr) }
274   .comment 0 : { *(.comment) }
275   /* DWARF debug sections.
276      Symbols in the DWARF debugging sections are relative to the beginning
277      of the section so we begin them at 0.  */
278   /* DWARF 1 */
279   .debug          0 : { *(.debug) }
280   .line           0 : { *(.line) }
281   /* GNU DWARF 1 extensions */
```

```
282  .debug_srcinfo  0 : { *(.debug_srcinfo) }
283  .debug_sfnames  0 : { *(.debug_sfnames) }
284  /* DWARF 1.1 and DWARF 2 */
285  .debug_aranges  0 : { *(.debug_aranges) }
286  .debug_pubnames 0 : { *(.debug_pubnames) }
287  /* DWARF 2 */
288  .debug_info     0 : { *(.debug_info) }
289  .debug_abbrev   0 : { *(.debug_abbrev) }
290  .debug_line     0 : { *(.debug_line) }
291  .debug_frame    0 : { *(.debug_frame) }
292  .debug_str      0 : { *(.debug_str) }
293  .debug_loc      0 : { *(.debug_loc) }
294  .debug_macinfo  0 : { *(.debug_macinfo) }
295  /* SGI/MIPS DWARF 2 extensions */
296  .debug_weaknames 0 : { *(.debug_weaknames) }
297  .debug_funcnames 0 : { *(.debug_funcnames) }
298  .debug_typenames 0 : { *(.debug_typenames) }
299  .debug_varnames  0 : { *(.debug_varnames) }
300 }
```

8.3.4 D-CC/M32R を使用する場合

メモリ配置を設定するためには、MR32R が提供しているリンカスクリプト(m32r.cmd)で設定します。このファイルは、リンカ(dld)実行時に利用されます。ここでは、MR32R で定義されているシンボル・および製品に添付しているサンプルファイルの内容について説明します。なお、リンカスクリプトの記述方法の詳細については、**D-CC/M32R** に付属しているドキュメントをご覧ください。

サンプルリンカスクリプトのメモリイメージは、図 7.4 で示す M3T-CC32R 用のものと同じになります。

8.3.5 サンプルリンカスクリプト(DCC/M32R 対応)

```
 1 /*
 2  * m32r.cmd -- Linker script for M32R
 3  * @(#) m32r.cmd 1.0@(#) 2000/05/18
 4  * $Id: m32r.cmd,v 1.2 2001/04/18 06:24:30 inui Exp $
 5  */
 6
 7 /* Memory Layout
 8  *   This layout is just example.  Applying this example, it
 9  *   is able to check a linking process exactly.
10  *
11  */
12 /*
13 in_rom1,in_rom2などそれぞれの領域に対して、開始アドレス・領域のサイズを定義します。
14 */
15
16 MEMORY
17 {
18     in_rom1  : org = 0x0,            len = 0x10
19     in_rom2  : org = 0x10,           len = 0x70
20     in_rom3  : org = 0x80,           len = 0x10
21     in_rom4  : org = 0x100,          len = 0x3fff00
22     in_ram1  : org = 0xf00000        len = 0x10000
23     out_ram1 : org = 0x1000000,      len = 0x400000
24 }
25
26 /* Section Layout
27  *
28  *   <section name>  <placed in>     <placed in while executing>
29  *   .reset          ROM             ROM
30  *   .EIT_Vector     ROM             ROM
31  *   .Int_Vector     ROM             ROM
32  *   .text           ROM             ROM
33  *   .data           ROM             RAM
34  *   .bss            RAM             RAM
35  *   .heap           RAM             RAM
36  *   .spu_stack              RAM             RAM
37  *   .spi_stack              RAM             RAM
38  */
39
40 /* セクション配置を行います。*/
41
42 SECTIONS
43 {
44   GROUP : {
45     .RESET_VECT : { *(.EIT_Vector) }
46   } > in_rom1
47
48   GROUP : {
49     .EIT_Vector : { *(.EIT_Vector) }
50   } > in_rom2
51
52   GROUP : {
53     .Int_Vector : {  *(.Int_Vector) }
54   } > in_rom3
55
56   GROUP : {
57     .MR_KERNEL : {
58       *(.MR_KERNEL)
59       *(.MR_KERNEL2)
60       *(.OS_DEBUG)
61     }
62     .text : {
63       *(.text)
64       *(.text2)
65       *(.init)
66       *(.fini)
67       *(.rodata)
68     }
69     .STARTUP : {*(.STARTUP)}
```

```
 70   } > in_rom4
 71
 72   GROUP :{
 73
 74     .MR_RAM : {*(.MR_RAM) *(.MR_Dbg_RAM)}
 75
 76 /* .dataセクションの配置を行います。セクションの開始アドレスを__START_data
 77 終了アドレスを__END_dataというシンボルで定義します。転送元アドレスは、__ROM_dataと
 78 しています。転送元アドレス__ROM_dataは、ファイルの最後で定義しています。*/
 79
 80     __START_data = .;
 81     .data LOAD(__ROM_data): {
 82       *(.data)
 83     }
 84     __END_data = .;
 85
 86 /* .MR_ROMセクションの配置を行います。セクションの開始アドレスを__START_MR_ROM
 87 終了アドレスを__END_MR_ROMというシンボルで定義します。転送元アドレスは、__ROM_MR_ROMと
 88 しています。転送元アドレス__ROM_MR_ROMは、ファイルの最後で定義しています。*/
 89
 90     __START_MR_ROM = .;
 91     .MR_ROM LOAD(__ROM_MR_ROM): {
 92       *(.MR_ROM)
 93       *(.INTERRUPT_VECTOR)
 94     }
 95     __END_MR_ROM = .;
 96
 97     .bss : {
 98       __START_bss = .;
 99       *(.sbss)
100       *(.bss)
101       *[COMMON]
102       __END_bss = .;
103     }
104     __SP_INIT = .;
105     .SYS_STACK : {*(.SYS_STACK)}
106     __SP_END = .;
107     .INT_USR_STACK : {*(.INT_USR_STACK)}
108     .MR_HEAP : {*(.MR_HEAP)}
109       __HEAP_START=.;
110   } > in_ram1
111
112   GROUP :{
113     .EXT_MR_RAM :{*(.EXT_MR_RAM)}
114     .EXT_USR_STACK :{*(.EXT_USR_STACK)}
115     .EXT_MR_HEAP :{*(.EXT_MR_HEAP)}
116   } > in_ram2
117
118 }
119
120 /*  必要なシンボルを定義します。 */
121
122 __ROM_data = addr(.START_UP) + sizeof(.START_UP);
123 __ROM_MR_ROM = __ROM_data + sizeof(.data);
124 __BSS_START = __START_bss;
125 __BSS_END = __END_bss;
126 __DATA_ROM = __ROM_data;
127 __DATA_RAM = __START_data;
128 __DATA_END = __END_data;
129
130 __HEAP_END = addr(in_ram2) + sizeof(in_ram2);
131
```

8.4 makefile の編集

8.4.1 M3T-CC32R を使用する場合

コンフィグレータが生成した、makefile を編集し、コンパイルオプションやライブラリなどを設定します。以下にその設定方法を示します。

1. cc32R コマンドオプション

   C コンパイラのコマンドオプションは"CFLAGS"に定義します。"-c"オプションは必ず指定して下さい。

2. as32R コマンドオプション

   アセンブラのコマンドオプションは"ASFLAGS"に定義します。

3. lnk32R コマンドオプション

   リンカのコマンドオプションは"LDFLAGS"に定義します。特に指定しなければならないオプションはありません。

4. ライブラリの指定

   ライブラリの指定は、まず、ライブラリのパス名を'echo "-L ライブラリのパス" >> lnk32R.sub'と記述します。
   次に指定するライブラリを'echo "-l ライブラリのパス" >> lnk32R.sub'と記述し、ライブラリを指定します。
   以下に、smp32r ディレクトリにある smp.lib を指定して例を示します。

```
lnk32r.sub:makefile
        echo -o $(PROGRAM) > lnk32R.sub
        echo -L $(LIB32R) >> lnk32R.sub
        echo -l c32Rmr.lib >> lnk32R.sub
        echo -l mr32R.lib >> lnk32R.sub
  # この後にライブラリの指定を行います。
  # ライブラリのパスを指定します。
        echo -L c:¥mtool¥mr32r¥smp32r >> lnk32R.sub
        echo -l smp.lib >> lnk32R.sub
```

8.4.2 D-CC/M32R を使用する場合

コンフィグレータが生成した、makefile を編集し、コンパイルオプションやライブラリなどを設定します。以下にその設定方法を示します。

5. DCC コマンドオプション

   C コンパイラのコマンドオプションは"CFLAGS"に定義します。"-c"オプションは必ず指定して下さい。

6. DAS コマンドオプション

   アセンブラのコマンドオプションは"ASFLAGS"に定義します。

7. DLD コマンドオプション

   リンカのコマンドオプションは"LDFLAGS"に定義します。

8. ライブラリの指定

   ライブラリの指定は、まず、ライブラリのパス名を'echo "-L ライブラリのパス" >> tmp.cmd と記述します。
   次に指定するライブラリを'echo "-l ライブラリのパス" >> tmp.cmd と記述し、ライブラ

リを指定します。

以下に、smp32r ディレクトリにある smp.lib を指定して例を示します。

```
$(PROGRAM).x: $(ALLOBJS) makefile m32r.cmd
      @echo¥$(LDFLAGS) > tmp.cmd
# この後にライブラリの指定を行います。
# ライブラリのパスを指定します。
      @echo –L c:¥mtool¥mr32r¥smp32r >> tmp.cmd
      @echo –l smp.lib >> tmp.cmd
      @echo¥$(LDFLAGS) > tmp.cmd
      @echo¥$(OBJS1)  >> tmp.cmd
      @echo¥$(OBJS2)  >> tmp.cmd
```

第 9 章　　アプリケーション作成の手引き

9.1 ハンドラからのシステムコールの処理手順

ハンドラからのシステムコール発行はタスクからのシステムコールと異なり、システムコール発行時にタスク切り替えは発生しません。タスク切り替えが発生するのはハンドラからの復帰時です。

ハンドラからのシステムコール処理手順は大きく分けて以下の３通りがあります。

1. タスク実行中に割り込んだハンドラからのシステムコール
2. システムコール処理中に割り込んだハンドラからのシステムコール
3. ハンドラ実行中に割り込んだ（多重割り込み）ハンドラからのシステムコール

9.1.1 タスク実行中に割り込んだハンドラからのシステムコール

スケジューリング（タスク切り替え)は ret_int システムコールによりおこなわれます。(図 9.1参照)

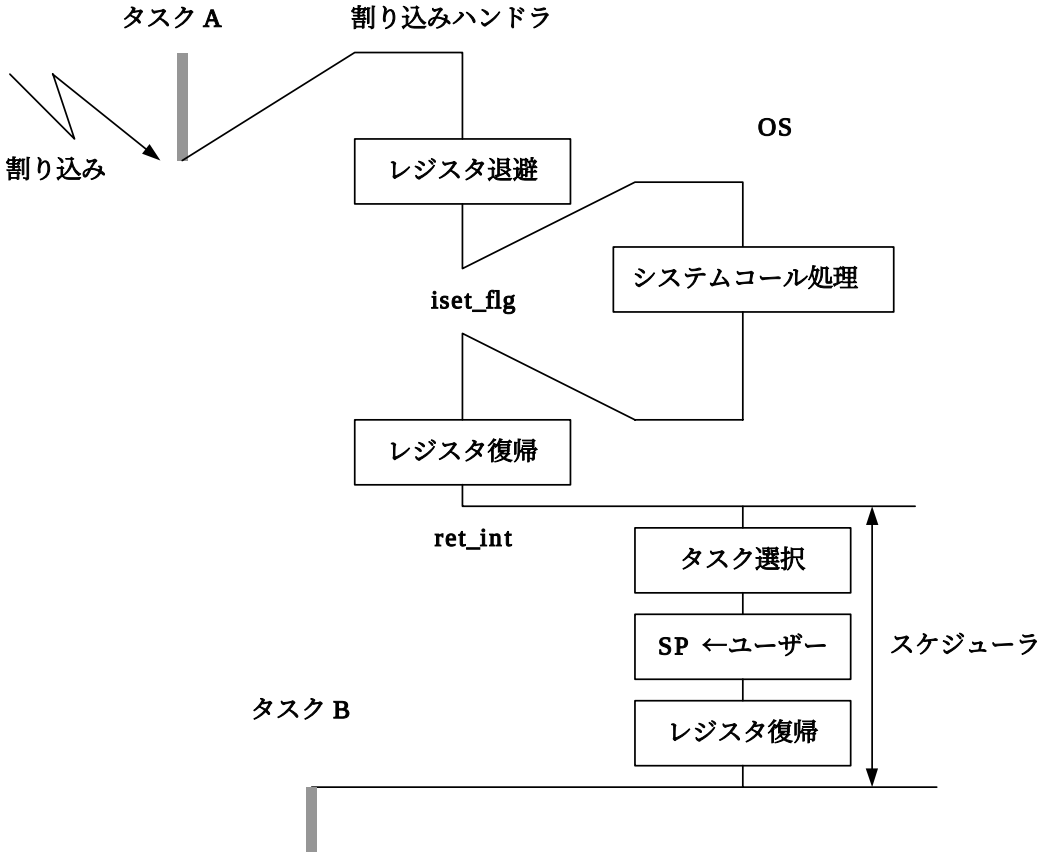


図 9.1　タスク実行中に割り込んだ割り込みハンドラからのシステムコール処理手順

9.1.2 システムコール処理中に割り込んだハンドラからのシステムコール

スケジューリング (タスク切り替え)は割り込まれたシステムコール処理に戻った後におこなわれます。 (図 9.2参照)

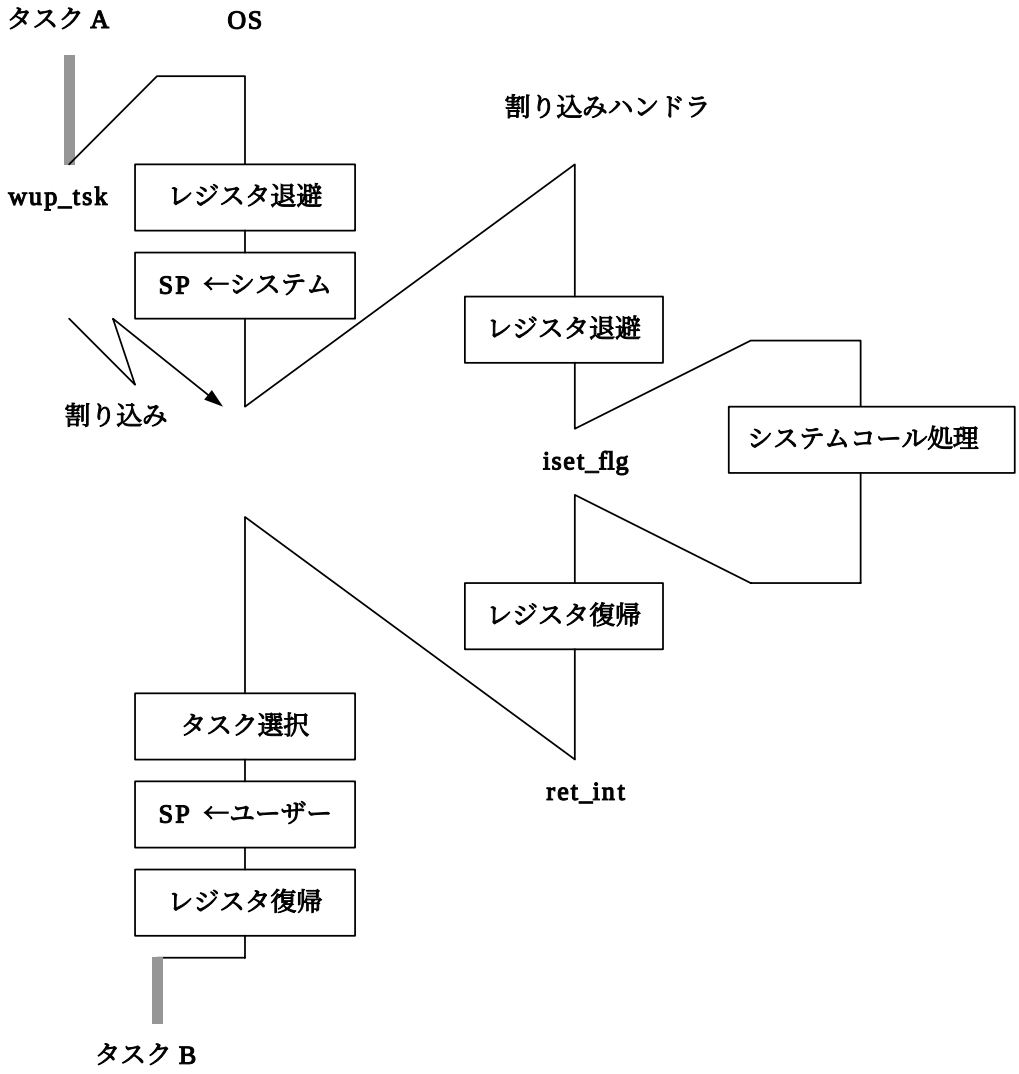


図 9.2 システムコール処理中に割り込んだ割り込みハンドラからのシステムコール処理手順

9.1.3 ハンドラ実行中に割り込んだハンドラからのシステムコール

ハンドラ (以後ハンドラ A と呼びます。)実行中に割り込みが発生した場合を考えます。ハンドラ A 実行中に割り込んだハンドラ (以後ハンドラ B と呼びます。)が、発行したシステムコールによりタスク切り替えが必要になった場合は、ハンドラ B から復帰するシステムコール (ret_int システムコール)では、ハンドラ A に戻るだけでタスク切り替えはおこないません。

ハンドラ A からの ret_int システムコールによりタスク切り替えが行われます。 (図 9.3参照)

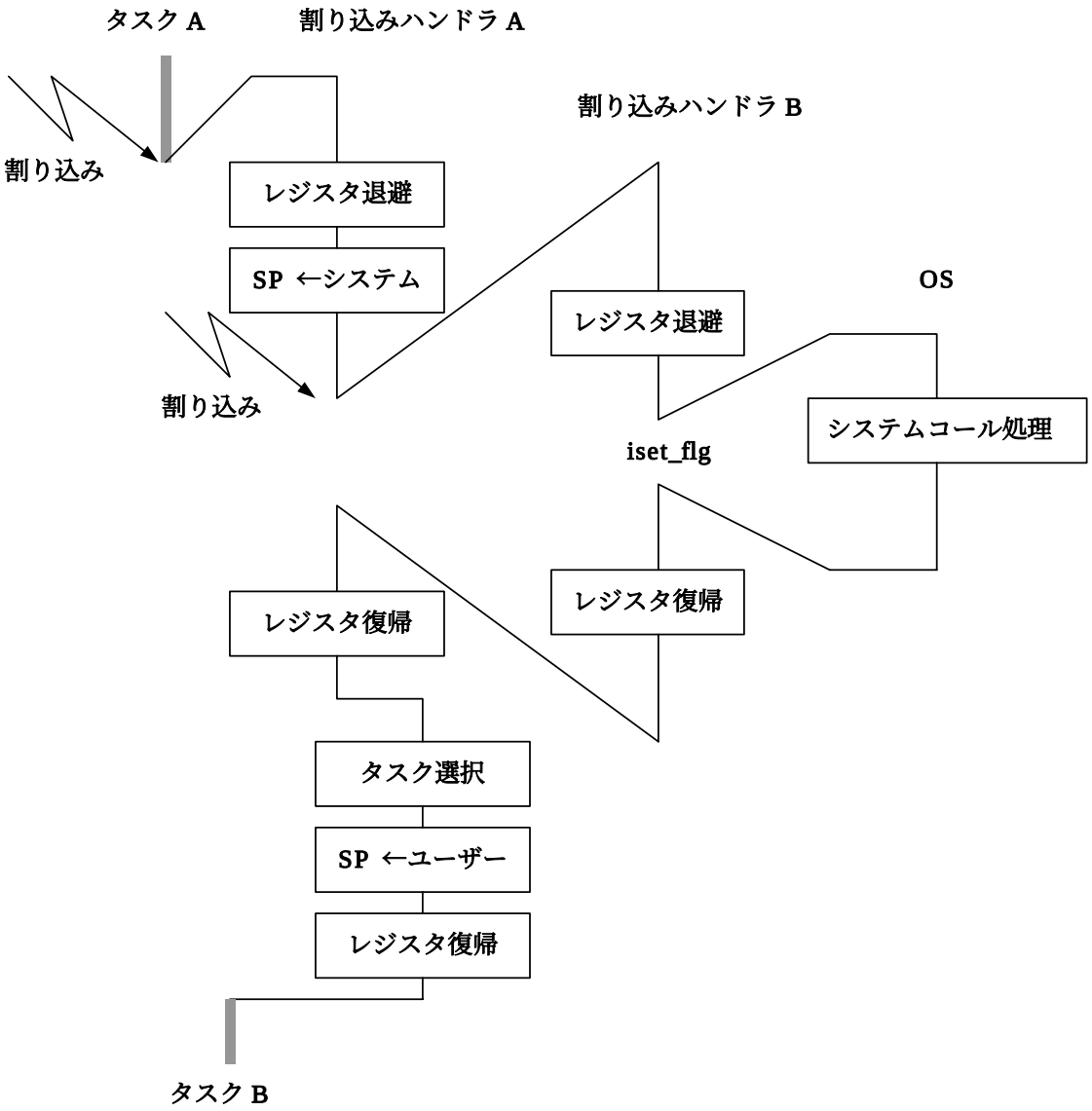


図 9.3 多重割り込みハンドラからのシステムコール処理手順

9.2 システムの使用する RAM 容量の計算方法

MR32R カーネルがタスクなどを管理するためのデータは、MR_RAM セクションに配置されます。オブジェクトの動的生成機能を使用する場合、MR_ROM セクションを RAM に転送して使用する必要がありますので MR_ROM セクションと MR_RAM セクションの使用サイズの合計が RAM 使用量になります。MR_RAM セクションにおいて MR32R が使用する RAM 容量は次の表で計算することができます。ただし、システムおよびタスクの使用するスタックは含まれていません。

また、従来、MR_ROM セクションに配置されていた割り込みハンドラアドレステーブルは、INTERRUPT_VECTOR セクションに配置されるようになりましたのでご注意ください。このセクションは、def_int を使用する場合は、RAM 領域に転送する必要があります。

スタックサイズの計算はリファレンスマニュアルを参照して下さい。

表 9-1 MR_RAM セクションのサイズ算出方法

領域名	バイト数(各領域で 4 バイトのアライメント調整が必要です)
システム管理領域	24+4×(優先度数+最大タスク数[76]×2+最大フラグ数[76]+最大セマフォ数[76]+最大メールボックス数[76]+最大固定長メモリプール数[76]+最大可変長メモリプール数[76]+最大メッセージバッファ数[76]×2+最大ランデブポート数[76]×2+最大優先度数×TA_TPRI 指定の優先度付きメイルボックス数+TA_TFIFO 指定の優先度付きメイルボックス+最大優先度数×(最大優先度付きメイルボックス数-定義済み優先度付きメイルボックス)+2)
タスク管理領域	37×最大タスク数[76](例外ハンドラ未使用時)
	43×最大タスク数[76](例外ハンドラ使用時)
イベントフラグ管理領域	11×最大イベントフラグ数[76]+(最大イベントフラグ数-1)/8+4
セマフォ管理領域	8×最大セマフォ数[76]+(最大セマフォ数[76]-1)/8+4
メールボックス管理領域	17×最大メールボックス数[76]+(最大メールボックス数[76]-1)/8+4
メールボックス領域	メールボックスのバッファサイズの合計
メッセージバッファ管理領域	21×最大メッセージバッファ数[76]+(最大メッセージバッファ数[76]-1)/8+4
メッセージバッファ領域	メッセージバッファのバッファサイズの合計
ランデブ用ポート管理領域	6×最大ランデブ用ポート数[76]+(最大ランデブ用ポート数[76]-1)/8+4
優先度付きメイルボックス管理領域	9×最大優先度付きメイルボックス数[76]+(最大優先度付きメイルボックス数[76]-1)/8+4
固定長メモリプール管理領域	9×最大固定長メモリプール数[76]+(最大固定長メモリプール数[76]-1)/8+4
可変長メモリプール管理領域	81×最大可変長メモリプール数[76]+(最大可変長メモリプール数[76]-1)/8+4
周期起動ハンドラ管理領域	5×周期起動ハンドラ数
アラームハンドラ管理領域	1
タスク管理領域(動的生成時に)[77]	68(内蔵 RAM 使用時) 68(外部 RAM 使用時)
メールボックス管理領域(動的生成時)[77]	72(内蔵 RAM 使用時) 72(外部 RAM 使用時)
メッセージバッファ管理領域(動的生成時)[77]	72(内蔵 RAM 使用時) 72(外部 RAM 使用時)

[76] コンフィグレーションファイルの最大項目数定義で指定した値です。この項目で値を指定していない場合は、コンフィグレーションファイルで静的に定義した値を指します。

[77] 動的生成機能を使用した場合(cre_tsk,del_tsk,cre_mbx など)に確保されます。

固定長メモリプール管理領域 (動的生成時)[77]	72(内蔵 RAM 使用時) 72(外部 RAM 使用時)
可変長メモリプール管理領域 (動的生成時)[77]	72(内蔵 RAM 使用時) 72(外部 RAM 使用時)

(注) システム管理領域は、必ず確保されます。また、その他の領域については、最大項目数定義の該当オブジェクトの最大値を 0 に指定した場合・記述を省略し、デフォルト値を使用する場合は、確保されません。

たとえば、max_sem = 0;と定義し、セマフォを定義していない場合は、セマフォ管理領域はメモリを消費しません。

表 9-2 MR_ROM セクションのサイズ算出方法

領域名	バイト数(各領域で 4 バイトのアライメント調整が必要です)
システム時刻管理領域	6
システム管理領域	(初期起動タスク数+1)×2
タスク管理領域	16×最大タスク数[76]
イベントフラグ管理領域	(最大イベントフラグ数[76]-1)/8+4
セマフォ管理領域	8×最大セマフォ数[76]+(最大セマフォ数[76]-1)/8+4
メールボックス管理領域	9×最大メールボックス数[76]+(最大メールボックス数[76]-1)/8+4
メッセージバッファ管理領域	9×最大メッセージバッファ数[76]+(最大メッセージバッファ数[76]-1)/8+4
ランデブ用ポート管理領域	(最大ランデブ用ポート数[76]-1)/8+4
優先度付きメイルボックス管理領域	6×最大優先度付きメイルボックス数[76]+(最大優先度付きメイルボックス数[76]-1)/8+4
固定長メモリプール管理領域	13×最大固定長メモリプール数[76]+(最大固定長メモリプール数[76]-1)/8+4
可変長メモリプール管理領域	17×最大可変長メモリプール数[76]+(最大可変長メモリプール数[76]-1)/8+4
周期起動ハンドラ管理領域	9×周期起動ハンドラ数
アラームハンドラ管理領域	12×アラームハンドラ数
バージョン情報管理領域	20(get_ver を使用する場合に確保されます。)
システムコールテーブル	420

(注) システム管理領域・システム時刻管理領域は、必ず確保されます。また、その他の領域については、最大項目数定義の該当オブジェクトの最大値を 0 に指定した場合・記述を省略し、デフォルト値を使用する場合は、確保されません。

たとえば、max_sem = 0;と定義し、セマフォを定義していない場合は、セマフォ管理領域はメモリを消費しません。

表 9.3 INTERRUPT_VECTOR セクションのサイズ算出方法

領域名	バイト数
割り込みハンドラアドレステーブル	(最大割り込みハンドラ定義数[78])×4

[78] 最大項目数定義で指定した、最大割り込み要因数です。

9.3 スタックについて

9.3.1 システムスタックとユーザースタック

MR32R のスタックにはシステムスタックとユーザースタックがあります。

- ユーザースタック
  タスクごとに1つずつ存在するスタックです。したがってMR32Rを用いてアプリケーションを記述する場合はタスクごとのスタック領域を確保する必要があります。

- システムスタック
  MR32R 内部（システムコール処理中）に使用されるスタックです。MR32R ではシステムコールをタスクが発行するとスタックをユーザースタックからシステムスタックに切り替えます。（図 9.4　システムスタックとユーザースタックを参照して下さい。）
  システムスタックは、割り込みスタック(SPI)を使用します。

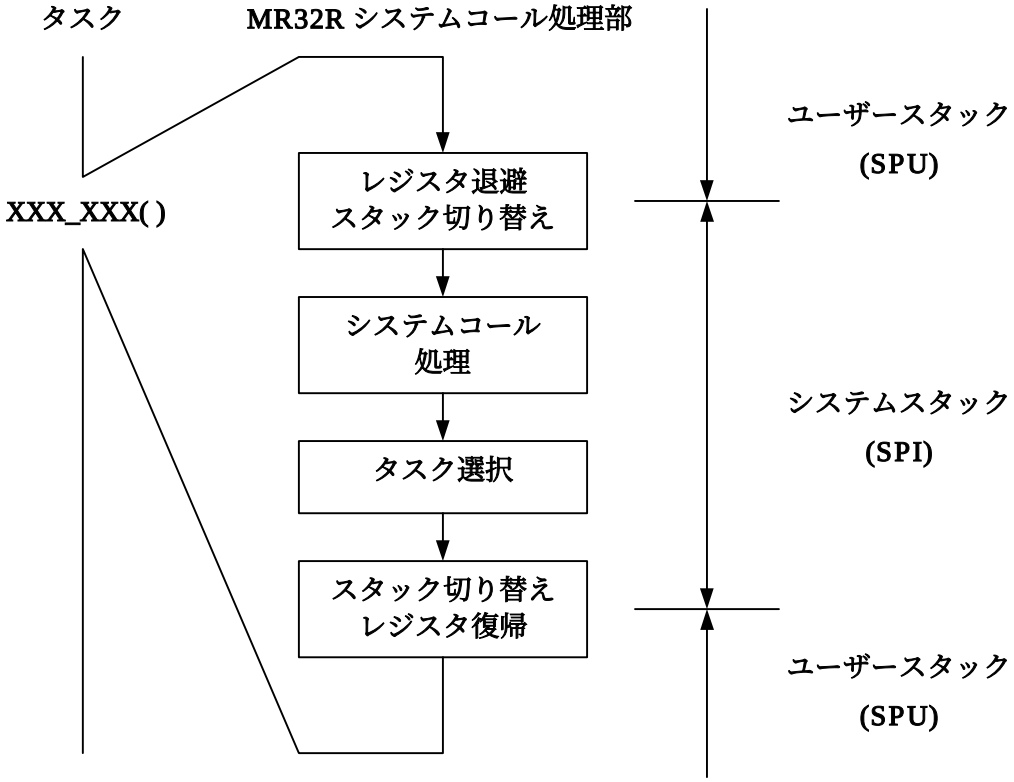


図 9.4　システムスタックとユーザースタック

第 10 章　　付録

10.1サンプルプログラム

10.1.1 サンプルプログラム概要

MR32R の応用例として、M32102 評価用ボードからシリアル出力させるプログラムを示します。

表 10-1 サンプルプログラムの関数一覧

関数名	種類	ID 番号	優先度	機能
Main()	タスク	1	1	task1、task2 を起床させます。
Task2()	タスク	2	2	メッセージ 1 を出力します。
Task3()	タスク	3	3	メッセージ 2 を出力します。
Alh1()	ハンドラ			メッセージ 1 をメールボックスに送ります。
Chy1()	ハンドラ			task2()を起床します。

メッセージ 1 : “Hello World!!”
メッセージ 2 : “MR32R is a Real Time OS for M32R”

以下に、処理内容を説明します。

- main タスクは、シリアルの初期設定を行い、task1、task2 を起動し、自タスクを終了させます。

- task1 は、次の順で動作します。
  1. メールボックス mbx1 から、メッセージ 1 を受け取り、メールボックス mbx2 からは mbx1 から受け取ったメッセージ 1 のサイズを受け取ります。
  2. 次に、シリアルに受け取ったメッセージを出力します。
  3. その後、システムクロックが 100 回カウントするまで待ち状態に移行し、実行状態に戻ると周期起動ハンドラを停止させ、再びシステムクロックが 100 回カウントするまで待ち状態に移行します。
  4. 実行状態に戻ると、メッセージ 1 を出力し、周期起動ハンドラを有効にすると同時に、周期カウンタもクリアします。
  5. 3、4を繰り返します。

- task2 は、次の順で動作します。
  1. 待ち状態に入り、周期起動ハンドラから起床されるのを待ちます。
  2. 起床されると、メッセージ 2 のサイズだけ、メモリを確保し、確保できれば、確保した領域にメッセージ 2 をコピーします。
  3. コピーされたメッセージ 2 を出力します。
  4. メッセージ 2 を出力すると、メッセージ 2 の入っていた領域を開放します。
  5. 1～4を繰り返します。

- alh1 は、mbx1 にメッセージ 1 を送り、mbx2 にそのサイズを送ります。

- cyh1 は、システムクロックが 10 回カウントされる毎に起動し、task2 を起床します。

10.1.2 サンプルプログラムソース

```
 1 /************************************************************************
 2 *                       MR32R  smaple program
 3 *
 4 * COPYRIGHT(C) 2001(2003) RENESAS TECHNOLOGY CORPORATION
 5 * AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
 6 *
 7 *
 8 *     $Id: smp.c,v 1.2 2001/05/11 10:08:37 inui Exp $
 9 ************************************************************************/
10
11 #include <mr32r.h>
12 #include "id.h"
13
14 void OutputMessage( char *,INT);
15
16 #define     P5MOD           (volatile UH *)(0x00EF106A)
17 #define     SIO0CR          (volatile UH *)(0x00EFD002)
18 #define     SIO0MOD0        (volatile UH *)(0x00EFD006)
19 #define     SIO0MOD1        (volatile UH *)(0x00EFD00A)
20 #define     SIO0STS         (volatile UH *)(0x00EFD00E)
21 #define     SIO0TRCR        (volatile UH *)(0x00EFD012)
22 #define     SIO0BAUR        (volatile UH *)(0x00EFD016)
23 #define     SIO0RBAUR       (volatile UH *)(0x00EFD01A)
24 #define     SIO0TXB         (volatile UB *)(0x00EFD01F)
25
26 void main(void)
27 {
28     *P5MOD = ((0x00ff & 0) | 0x5500);
29     *SIO0CR = 0x0300;
30     *SIO0MOD0 = 0x0100;
31     *SIO0MOD1 = 0x0800;
32     *SIO0BAUR = 34;
33     *SIO0RBAUR = 12;
34     *SIO0TRCR = 0x0000;
35     *SIO0CR = 0x0303;
36
37     sta_tsk(ID_task1,0);
38     sta_tsk(ID_task2,0);
39
40     ext_tsk();
41 }
42 void task1(void)
43 {
44     INT length;
45     char *message1;
46     INT message2;
47
48     rcv_msg((PT_MSG *)&message1,ID_mbx1);
49     rcv_msg((PT_MSG *)&message2,ID_mbx2);
50     OutputMessage(message1,message2);
51
52     while(1){
53             dly_tsk(100);
54             act_cyc(ID_cyh1,TCY_OFF);
55             dly_tsk(100);
56
57             OutputMessage(message1,message2);
58
59             act_cyc(ID_cyh1,TCY_INI_ON);
60     }
61 }
62
63 void task2(void)
64 {
65     INT length;
66     ER ercd;
67     char *message1,*string,*tmp;
68     INT message2,i;
69     INT message4 = sizeof("MR32R is a Real Time OS for M32R¥n");
70     while(1){
```

```
 71              slp_tsk();
 72              ercd = pget_blk( (VP *)&string, ID_memblk1,message4);
 73              if( ercd == E_OK ){
 74                      char *message3 = "MR32R is a Real Time OS for M32R¥n";
 75                      tmp = string;
 76                      for( i=0;i<message4;i++){
 77                              *string = *message3;
 78                              *string++;
 79                              *message3++;
 80                      }
 81                      string = tmp;
 82                      OutputMessage(string,message4);
 83                      string = tmp;
 84                      rel_blk( ID_memblk1, string);
 85              }
 86      }
 87 }
 88
 89 void alh1(void)
 90 {
 91      INT message2;
 92      char *message1;
 93      message1 = "Hellow world !!¥n";
 94      message2 = sizeof("Hellow world !!¥n");
 95      isnd_msg(ID_mbx1,(PT_MSG)message1);
 96      isnd_msg(ID_mbx2,(PT_MSG)message2);
 97 }
 98 void cyh1(void)
 99 {
100      iwup_tsk(ID_task2);
101 }
102
103 void OutputMessage(char *message,INT length)
104 {
105      INT i;
106      for(i=0;i<length;i++){
107              while( !((*SIO0STS)&0x0001));
108              *SIO0TXB = *message++;
109      }
110 }
111
```

10.1.3 コンフィグレーションファイル

```
 1 //************************************************************************
 2 //
 3 //   Copyright 2001 MITSUBISHI ELECTRIC CORPORATION
 4 //   AND MITSUBISHI ELECTRIC SEMICONDUCTOR APPLICATION ENGINEERING CORPORATION
 5 //
 6 //   MR32R System Configuration File.
 7 //   smp.cfg
 8 //      $Id: smp.cfg,v 1.2 2001/04/18 06:24:29 inui Exp $
 9 //************************************************************************
10
11 system{
12      stack_size      = 0x1000;
13      priority        = 4;
14      exc_handler     = NO;
15      debug           = NO;
16      debug_buffer    = 0;
17 };
18 //
19 maxdefine{
20      max_task        = 3;
21      max_int         = 32;
22      max_alh =1;
23 };
24 //
25 clock{
26 //   mpu_clock       = 33.3MHz;
27 //   IPL             = 3;
28 //   unit_time       = 100ms;
29      mode            = OTHER;
30 //   mode            = MFT00;
31 //   file_name       = m32102.tpl;
32      initial_time    = 0:0:0;
33 };
34 //
35 task[]{
36      entry_address   = main();
37      stack_size      = 0x1000;
38      stack_area      = INTERNAL;
39      priority        = 1;
40      initial_start   = ON;
41 };
42 task[]{
43      entry_address   = task1();
44      stack_size      = 0x1000;
45      stack_area      = INTERNAL;
46      priority        = 2;
47 };
48 task[]{
49      entry_address   = task2();
50      stack_area      = INTERNAL;
51      stack_size      = 0x1000;
52      priority        = 3;
53 };
54 //
55 mailbox[]{
56      name            = mbx1;
57      mbx_area        = INTERNAL;
58      buffer_size     = 10;
59 };
60 mailbox[]{
61      name            = mbx2;
62      mbx_area        = INTERNAL;
63      buffer_size     = 10;
64 };
65 //
66 variable_memorypool[]{
67      name            = memblk1;
68      max_memsize     = 0x100;
69      heap_size       = 0x300;
70 };
```

```
 71 cyclic_hand[]{
 72     interval_counter = 10;
 73     mode             = TCY_ON;
 74     entry_address    = cyh1();
 75 };
 76 //
 77 alarm_hand[]{
 78     time             = 0:00:5;
 79     entry_address    = alh1();
 80 };
 81
 82 systemcall{
 83     cre_tsk          = NO;
 84     del_tsk          = NO;
 85     sta_tsk          = YES;
 86     ter_tsk          = NO;
 87     chg_pri          = NO;
 88     rot_rdq          = NO;
 89     rel_wai          = NO;
 90     ena_dsp          = NO;
 91     sus_tsk          = NO;
 92     rsm_tsk          = NO;
 93     slp_tsk          = YES;
 94     wup_tsk          = NO;
 95     set_flg          = NO;
 96     wai_flg          = NO;
 97     sig_sem          = NO;
 98     wai_sem          = NO;
 99     snd_msg          = NO;
100     rcv_msg          = YES;
101     pget_blf         = NO;
102     rel_blf          = NO;
103     pget_blk         = YES;
104     rel_blk          = YES;
105     unl_cpu          = NO;
106     dly_tsk          = YES;
107     ista_tsk         = YES;
108     ichg_pri         = NO;
109     irot_rdq         = NO;
110     irel_wai         = NO;
111     get_tid          = NO;
112     isus_tsk         = NO;
113     irsm_tsk         = NO;
114     iwup_tsk         = YES;
115     can_wup          = NO;
116     iset_flg         = NO;
117     clr_flg          = NO;
118     pol_flg          = NO;
119     isig_sem         = NO;
120     preq_sem         = NO;
121     isnd_msg         = YES;
122     prcv_msg         = YES;
123     set_tim          = NO;
124     get_tim          = NO;
125     act_cyc          = YES;
126     get_ver          = NO;
127     ret_int          = YES;
128     dis_dsp          = NO;
129     loc_cpu          = NO;
130     ext_tsk          = YES;
131     exd_tsk          = NO;
132 };
133 interrupt_vector[16] = __sys_timer;
134 //
135 // End of Configuraton
136 //
137
```

索引

M3T-MR32R V.3.50 ユーザーズマニュアル

Rev. 1.00
03.06.16
RJJ10J0116-0100Z

M3T-MR32R V.3.50
ユーザーズマニュアル

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668

RJJ10J0116-0100Z