九州大学学術情報リポジトリ
Kyushu University Institutional Repository

組込みシステムにおける割り込み制御に関する研究

南角，茂樹

https://doi.org/10.15017/1441262

出版情報：九州大学，2013，博士（工学），課程博士
バージョン：
権利関係：全文ファイル公表済

組込みシステムにおける
割り込み制御に関する研究

平成 26 年 1 月

南角　茂樹

目次

第 1 章 はじめに 1
1.1 組込みシステムとは 1
1.2 組込みシステムにおける並行性 3
1.3 組込みシステムにおける排他制御 3
1.4 組込みシステムのリアルタイム性 5
1.5 割り込みと割り込み処理 6
1.6 割り込み，割り込み処理，割り込み制御 10
1.7 割り込みとタスク 10
1.8 本論文の構成 10
第 2 章割り込み処理の現状分析と課題 12
2.1 組込みシステムにおける割り込み処理 12
2.2 割り込み処理の実行方式と問題点 13
2.2.1 タスクによる割り込み処理の実行 13
2.2.2 割り込み優先度をタスクの優先度に反映させる方式 13
2.2.3 単独のタスクですべての割り込み処理を実行する方式 14
2.2.4 最低優先度の割り込みですべての割り込み処理を実行する方式 15
2.2.5 ハードウェアの割り込み機能のみを使用した方式 15
2.2.6 フラグを利用した使用した方式 18
2.2.7 多重割り込み方式 19
2.3 割り込み処理の排他制御における問題点 20
2.4 優先度逆転による問題点 26
2.5 スタックオーバーフロー検出方式に関する問題点 27
2.5.1 スタック領域の用途とその問題点 30

2.5.2　割り込み処理のスタック領域の使い方とその問題点　32
2.6　本研究の位置付けと目的　33
第 3 章 関連研究　35
3.1　割り込み処理の制御に関する関連研究　35
3.1.1　割り込み処理の排他制御に関する関連研究　35
3.1.2　モニタに関する関連研究　36
3.1.3　CPU の割り込み優先度レベルの使用に関する関連研究　37
3.2　組込みシステムにおける割り込み処理の
スタックオーバーフローの検出に関する関連研究　37
3.2.1　スタックオーバーフローに関する関連研究概要　37
3.2.2　静的な方式　37
3.2.3　ARM 独自のハードウェアまたは MMU を用いる方式　38
3.2.3.1　TrustZone を用いた SafeG 方式　38
3.2.3.2　メモリ保護機能を搭載した RTOS(TOPPERS/HRP2)の
方式　39
3.2.3.3　ページテーブル書き換え方式　39
3.2.3.4　red zone 方式　39
3.2.3.5　ARM プロテクトドメイン方式　39
3.2.4　スタックオーバーフロー攻撃に対する方式　40
3.3　関連研究まとめ　40
第 4 章 割り込み処理の制御　42
4.1　提案方式の概要　42
4.2　割り込み処理に待ち状態を持たせてセマフォを実現する方式　43
4.2.1　REMON スケジューラ概要　43
4.2.2　ICB 詳細説明　45
4.2.3　REMON セマフォ概要　47
4.2.4　REMON の必要メモリ量　48
4.2.5　REMON スケジューラの動作　49
4.2.5.1　動作を示す記号の意味　49
4.2.5.2　割り込み発生時の共通処理　50
4.2.5.3　割り込み処理の終了処理　50
4.2.5.4　REMON セマフォに対する操作　53
4.2.5.5　処理時間が一定の P 操作　54
4.2.5.6　平均処理時間が短い P 操作　57
4.2.5.7　処理時間が一定の V 操作　58
4.2.5.8　平均処理時間が高速のセマフォに対する V 操作　60

4.2.6 REMON 全体構造 62
4.2.7 REMON による排他制御の動作 63
4.3 割り込み優先度レベルを利用した REMON の高速化 64
4.3.1 ハードウェア優先度利用の REMON スケジューラの動作 66
4.3.1.1 割り込み発生時の割り込み優先度レベル利用の共通処理 70
4.3.1.2 ハードウェア優先度を利用した優先度継承セマフォの P 操作 73
4.3.1.3 ハードウェア優先度を利用した優先度継承セマフォの V 操作 73
4.3.1.4 ハードウェア優先度を利用した優先度継承セマフォによる排他制御 75
4.4 割り込み処理のスタックオーバーフロー検出とスタックの再割り当て方式 76
4.4.1 割り込み処理のスタックオーバーフロー検出方式概要 76
4.4.2 マジックナンバの設定 77
4.4.3 スタックオーバーフロー検出方式 81
第 5 章 評価実験および考察 86
5.1 REMON の評価実験と考察 86
5.1.1 使用機材と実験方法 87
5.1.2 測定ツール REMON モニタの開発 88
5.1.3. DI/EI 方式との比較のための実験方法 89
5.1.4 DI/EI 方式との比較実験の結果 91
5.1.5 DI/EI 方式との比較結果の評価と考察 92
5.1.6 RTOS との比較実験の方法と結果 93
5.1.7 RTOS との比較実験の評価と考察 93
5.1.8 メモリ使用量の評価と考察 95
5.1.9 今後の展開に関する考察 96
5.2 割り込み優先度レベルを利用した REMON の評価実験と考察 96
5.2.1 使用機材と実験方法 97
5.2.2 割り込み優先度利用優先度継承セマフォの機能 97
5.2.3 割り込み優先度利用優先度継承機能マフォの評価と考察 99
75.2.4 ロジックアナライザによる処理時間の測定 101
5.2.5 ロジックアナライザによる処理時間の測定結果の評価と考察 102
5.2.6 CPU がハードウェアの優先度レベルを備えない場合の

考察 103
5.3 スタックオーバーフローに関する評価実験と考察 103
5.3.1 スタックオーバーフローに関する評価実験 104
5.3.2 スタックオーバーフローに関する評価実験の考察 106
5.3.3 スタックオーバーフロー検出の処理時間の評価実験と考察 106
5.3.4 スタックオーバーフローを原因とする割り込み処理の不具合判定時間の評価実験と考察 108
5.3.5 スタックオーバーフロー検出機能のメモリ容量に関する考察 110
5.3.6 マジックナンバの偽陽性に関する考察 111
第 6 章 おわりに 113
6.1 **成果** 113
6.2 **今後の課題** 113
謝辞 115
参考文献 116
本研究に関する著者の発表論文 119
本研究に関する著者の取得特許 121

図目次

図 1-1　組込みシステムのモデル　2
図 1-2　組込みシステムとしての人工衛星　2
図 1-3　排他制御を起こしたプログラム例　4
図 1-4　C 言語と機械語　5
図 1-5　CPU と割り込み　6
図 1-6　PSW　7
図 2-1　単独タスクによる割り込み処理の実行　14
図 2-2　CPU のビット数(2009 年度)　16
図 2-3　使用している OS の種類(2009 年度)　16
図 2-4　ハードウェアの割り込み機能のみを利用した方式　17
図 2-5　フラグを利用した方式　18
図 2-6　多重割り込み方式　19
図 2-7　割り込み禁止と許可を使用した排他制御方式　21
図 2-8　実製品における市場流失不具合件数　21
図 2-9　実製品における市場流失不具合割合　21
図 2-10　リアルタイム設計ミスの要因分析　22
図 2-11　RTOS の基本的な構造　23
図 2-12　割り込み処理の実行の流れ　25
図 2-13　優先度逆転　27
図 2-14　スマートキーレスエントリの不具合　29
図 2-15　スタックの使われ方　31
図 2-16　割り込み処理　32
図 3-1　モニタの構成　36
図 4-1　REMON の基本構造　44
図 4-2　REMON における割り込み処理の状態遷移　44
図 4-3　ICB の基本構造　46

図 4-4　セマフォ構造体　48
図 4-5　割り込み共通処理　52
図 4-6　割り込み処理の終了処理　53
図 4-7　処理時間一定の P 操作の処理　56
図 4-8　平均処理時間が高速の P 操作の処理　57
図 4-9　処理時間一定の V 操作の処理　59
図 4-10　平均処理時間が高速の V 操作の処理　61
図 4-11　REMON の全体構造　62
図 4-12　REMON セマフォによる排他制御動作　63
図 4-13　ハードウェア優先度を利用した REMON 全体図　68
図 4-14　ハードウェア優先度を利用した優先度継承セマフォを備えた REMON　68
図 4-15　割り込み優先度レベルを利用する REMON スケジューラの処理　71
図 4-16　割り込み優先度レベルを利用する優先度継承 P 操作の処理　72
図 4-17　割り込み優先度レベルを利用する優先度継承 V 操作の処理　74
図 4-18　優先度継承セマフォ利用時の排他制御の動作　75
図 4-19　スタック定義ブロック SDB の構造　78
図 4-20　スタックの初期化処理　80
図 4-21　マジックナンバが 4 個の場合の例　81
図 4-22　スタックオーバーフロー検出機能を備えた REMON の全体構成　82
図 4-23　スタックオーバーフロー検出機能動作　83
図 4-24　スタック再割り当て動作　84
図 5-1　評価実験に使用した CPU ボードとロジックアナライザ　87
図 5-2　REMON モニタ　88
図 5-3　排他制御をおこなわない多重割り込み　91
図 5-4　DI/EI による排他制御　91
図 5-5　REMON セマフォにより排他制御　91
図 5-6　REMON モニタによる評価　100
図 5-7　評価に使用したプログラム(A)　105
図 5-8　評価に使用したプログラム(B)　105

表目次

表 1-1　多重割り込みを許可しない場合の割り込み発生時の動作　8
表 1-2　多重割り込みを許可する場合の割り込み発生時の動作　9
表 1-3　タスクと優先度付き割り込み処理の比較　11
表 3-1　スタックオーバーフォローの検出方式比較　41
表 5-1　評価実験に使用した M16C/62A の仕様　88
表 5-2　定性的評価に使用した割り込み処理のスケジュール　90
表 5-3　定量的評価に使用した割り込み処理のスケジュール　90
表 5-4　DI/EI と REMON セマフォによる排他制御時間　92
表 5-5　REMON と JSP の処理時間の比較　94
表 5-6　REMON の使用メモリサイズ　96
表 5-7　測定した排他制御の方式（A）　101
表 5-8　測定項目（B）　101
表 5-9　REMON と TOPPERS/JSP の処理時間の測定結果　102
表 5-10　スタックオーバーフロー検出及び
スタック再割り当て方式の実験項目　104
表 5-11　割り当てサイズ　104
表 5-12　スタックオーバーフロー実験の結果　105
表 5-13　時間測定の実験時の設定　107
表 5-14　スタックオーバーフロー機能の時間測定の結果　108
表 5-15　スタックオーバーフローによる不具合の原因を
見つけるまでの時間　110
表 5-16　スタックオーバーフロー検出機能を備えた REMON の
メモリ容量　110
表 5-17　各種 CPU の LL/SC 系の命令　112

概要

現在，車に約 100 個使用される ECU(Electric Control Unit)はじめ，携帯電話，家電製品などの民生品，自動販売機，衛星機器，あるいは非接触型 IC カードなど，組込みシステムなしに生活は成り立たない．

組込みシステムとはソフトウェア，ハードウェア，機構(メカ)が協調し合って目的を達成するコンピュータシステムである．ただし汎用コンピュータシステムとは異なり，組込みシステムは現実世界の変化に制約時間以内に応答するというリアルタイム性が求められる．

組込みシステムは，現実世界の変化を捉えるのに各種センサを利用する．センサからの割り込みにより変化の発生を知る．いつどのような変化が起こるかをあらかじめ知ることが出来ない現実世界に対して，ソフトウェア（処理）の並列処理は，リアルタイムを満たすためにも重要である．

割り込みは，現実世界の変化を知るための信号として利用されるとともに，実行中のソフトウェアから，その変化に応答するソフトウェアである割り込み処理（割り込みハンドラ，または割り込みサービスルーチン)に処理を切り替えるための，ソフトウェアの切り替え機構としても利用される．

通常，汎用のコンピュータシステムにおいては，割り込みは隠蔽化されている．しかし，周期的なタイマー割り込みにより，アプリケーションソフトウェアからシステムソフトウェアに実行を切り替える，あるいは TSS(Time Sharing System)の実現に利用するなど，汎用のコンピュータシステムも割り込みなしでは成り立たない．

実行中のソフトウェアを外部から切り替える仕組みは割り込みのみである．コンピュータシステムにおいて，割り込みは非常に重要であり，割り込みによる処理の切り替えを利用して並行処理を実現している．

並列処理を実行する実体としては，RTOS(Real Time Operating System)を搭載している場合にはタスクやスレッドが利用され，RTOS 不使用の場合は割り込み

処理が利用される.

さらに，並行処理が実行される場合は，クリティカルセクション（CS：Critical Section）の排他制御機能が必要である．CS とは同時に実行するとデータの一貫性が失われるなどの不具合が発生する可能性がある一連の命令区間のことであり，CS の排他制御ができないとデータの不整合が発生する.

RTOS 使用時のタスク間の CS の排他制御にはセマフォが使用され、RTOS 不使用時の割り込み処理間の CS の排他制御には CPU の全外部割り込み禁止 DI(Disable Interrupt)と全外部割り込み許可 EI(Enable Interrupt)が使用される.

しかし現状の方式には次の問題点が存在する.

・RTOS を利用しない組込みシステムにおける排他制御の手段である，
　割り込み禁止および許可は，システム全体に影響を及ぼすため，
　割り込み応答性が損なわれリアルタイム性を損なう恐れがある.

・価格やメモリ搭載量などの理由で RTOS を利用できない組込みシステムが
　存在する.

・RTOS を利用しても，アプリケーションには不要な割り込みが
　余分に必要になり，さらに RTOS 内部の排他制御により，
　割り込み応答性が損なわれる.

・RTOS を搭載すると，並行処理を行う仕組みが割り込み処理と
　タスクの二重構造となり，システム設計が複雑になる.

この問題を解決するには、割り込み処理間の排他制御の影響が，関連する割り込み処理に限定される，リアルタイム性を損なわない，新しい排他制御方式が必要となる.

そこで，本論文では、主にシングルチップマイクロコンピュータを使用して，RTOS を使用しない組込みシステムに対して，割り込み処理間のリアルタイム性を損なわない CS の排他制御方式を提案する.

本提案方式は次の特徴を備える.

・割り込み処理間の排他制御の影響を関連する割り込み処理間のみに
　限定し，組込みシステムのリアルタイム性を向上させる.

・処理時間が一定で，リアルタイム設計に適する.

・CPU ハードウェアが割り込み優先度を備える場合には，平均処理速度を
　向上させる方式も可能である.

・MMU(Memory Management Unit)を使用しない組込みシステムに，
　割り込み処理のスタックオーバーフロー検出機能，およびメモリ
　再割り当て機能をあたえ，システムの信頼性を向上させる.

・並行処理を行う仕組みを，割り込み処理だけとしてシステム設計を
　容易にする.

・システムコールやタスク制御など複雑な機能をもつ RTOS を学習する
必要がないため、短時間に楽にリアルタイム制御を習得できる.

本提案の方式を割り込みスケジューラ REMON(Real-Time Embedded Monitor)と名付ける.

提案方式の REMON を実装して，M16C CPU を使用した実 CPU ボードを用いて評価を行った．評価には CPU ボードに REMON と比較対象として RTOS である TOPPERS/JSP を実装して，処理速度，排他制御などの処理時間の測定を行った．その結果 REMON セマフォの使用により，排他制御が関連しない割り込み処理には影響を及ぼさず，組込みシステムのリアルタイム性が向上することを確認した.

また REMON は TOPPERS/JSP と比較して，割り込み処理（タスク）起動時間や排他制御時間において高速である.

以上の評価により提案方式の有用性を示す.

なお，提案の割り込み処理の待ち状態を持たせてセマフォと同等の機能を実現して，リアルタイム性を向上させる方式は日本, 米国, ヨーロッパ，中国，台湾において特許を取得している.

第１章

はじめに

本章では，まず，組込みシステムに関する概要，および組込みシステムにおける，並行性，排他制御，リアルタイム性に関して述べる．次に，割り込み，割り込み処理に関して述べる．最後に，割り込み処理とタスクの比較を行う．

1.1　組込みシステムとは

現在，自動車，通信機器，家電機器など多くのものが製品内部にマイクロプロセッサを組込みこんだ組込みシステムであり，その需要が高まっている[1].

組込みシステムは現実世界の物理的な現象を対象としている，現実世界との相互作用により適切な処理を行うことが必要である．そのためにはセンサやアクチュエータとの連携が必須である.

組込みシステムは現実世界に変化があった場合は，速やかにそれを知り反応しなければならない．そのためには割り込みを使用することが多い.

図 1-1 に組込みシステムのモデルを示す．現実世界の変化をセンサが捉える．センサは現実世界の変化の発生を，割り込みによって組込みシステムに伝える.

組込みシステムは割り込みによって現実世界の変化の発生を知り，適切な計算を行い，その結果をアクチュエータにより，現実世界に反映する.

図 1-2 に図 1-1 で示したモデルの具体的な例を示す．人工衛星が軌道・姿勢制御を行う場合の例である．制御対象である人工衛星は，自分の姿勢向き

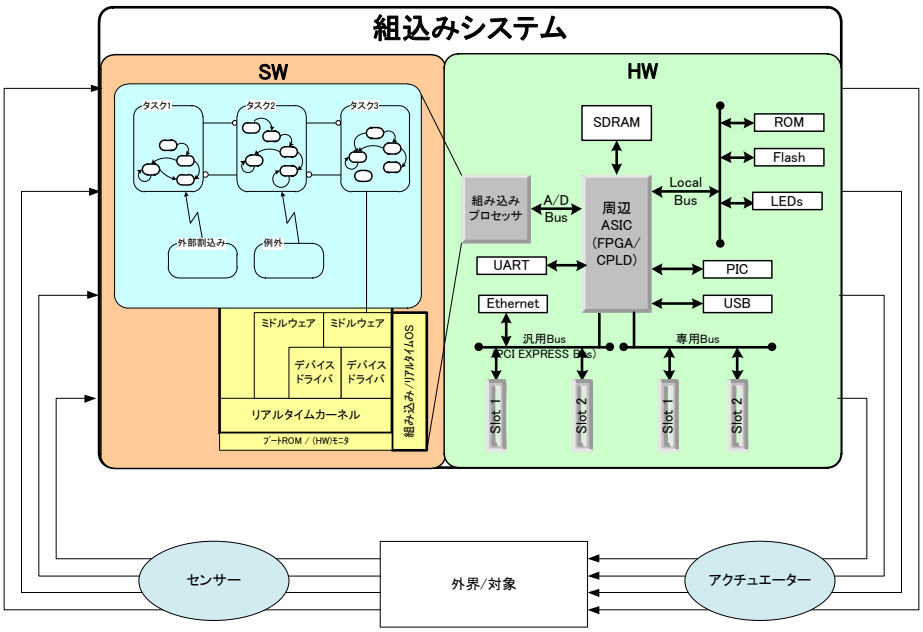


図 1-1　組込みシステムのモデル

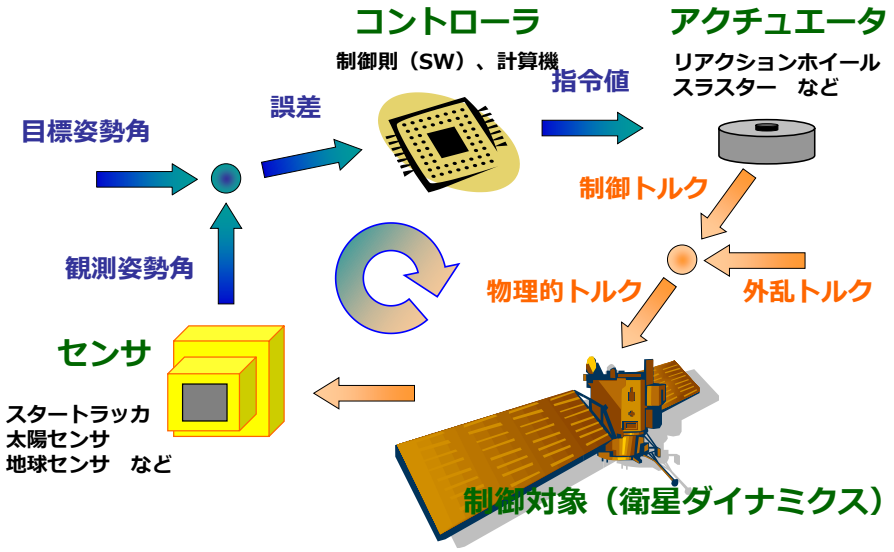


図 1-2　組込みシステムとしての人工衛星

などを，スタートラッカ， 太陽センサ，地球センサなどのセンサを通じて知る．そして軌道や姿勢が指令値に近づくための，リアクションホイールやスラスターなどのアクチュエータへの指令地を計算して指令する．この繰り返しにより，人工衛星は起動・姿勢を保つことができる．

この時，CPU（組込みプロセッサ，コントローラ，MCU(Micro Control

Unit)，シングルチップマイクロコンピュータ，マイコン，コンピュータなどとも呼ばれる）の動作を記述したものが組込みソフトウェアである．

組込みシステムに内蔵されている CPU は組込みソフトウェアの記述に従って動作を行う．

1.2 組込みシステムにおける並行性

1.1 節で述べたとおり，現実世界の変化に応答して動作するのが組込みシステムであり，現実世界の変化は次の特徴がある．

・現実世界の変化は非決定的に発生する．
・現実世界の変化への応答には時間制約がある．
・現実世界の変化は種類がある．
・複数の変化が同時に発生する可能性がある．

そのため，現実世界への応答を効果的に処理するためには，優先度を備えた並行処理（マルチタスク，コンカレント処理とも呼ばれる）を実現できることが必要になる．

1.3 組込みシステムにおける排他制御

並行処理（マルチタスク，コンカレント処理とも呼ばれる）間では，共有データの一貫性（coherency）が保たれなければならない．そのためには排他制御機能が必須となる．

図 1-3 に共有データの排他制御不足により不具合が発生したプログラムを示す．図における共有データは初期値 0 の flag である．この場合，割り込み A と割り込み B が両方とも発生した場合の flag の値は 0x03(0x01 OR 0x02)とならなければならないにもかかわらず，0x01 または 0x02 になる場合があった．実際の例では，flag の ON となっているビットによって，動作を行う他の並行処理が必要な動作を行わないという不具合が発生した．

組込みシステムでは性能や価格面から CISC(Complex Instruction Set Computer)系の CPU から RISC(Riduced Instruction Set Computer)系の CPU に変更する場合が多いが，その時に各社，様々な製品において同様の問題が発生している．原因は CISC 系の CPU においてはメモリ上のデータである flag に対する OR 演算は 1 命令であるのに対して，ロード／ストア アーキテクチャの RISC 系の CPU においては，C や C++言語においては，OR 演算は 1 実行命令であるにも関わらず，CPU が実行する命令は複数命令になっていること

```
volatile unsigned long flag = 0x00;

割り込み処理 A
{
     :
    flag |= 0x01;    // ビット 0 を ON
     :
}

割り込み処理 B
{
     :
    flag |= 0x02;    // ビット 1 を ON
     :
}
```

図 1-3　排他制御ミスを起こしたプログラム例

である．そしてその複数の命令実行中に処理の切り替えが発生すると flag の不整合が発生する．

図 1-4 に RISC 系 CPU(MIPS)における OR 命令の C(C++)言語における記述と機械語における記述を示す．CISC 系の CPU であれば，メモリ上の変数である flag に対して，直接 OR 演算を実行できるが，RISC ではまず flag の値をレジスタに持ってきて，そのレジスタに対して OR 演算を行い，最後にそのレジスタの値をメモリ上の flag 変数にコピーするという 3 命令になる．割り込み処理 A と割り込み処理 B におけるそれぞれの OR 命令が機械語レベルでは，それぞれ 3 命令となる．

割り込み処理のこの 3 命令の実行中に，処理の切り替えが発生して他の割り込み処理のこの 3 命令が実行されると flag の不整合が発生する．

この場合は flag が排他制御をしなければならないデータであり，両方の割り込み処理にある，C 言語では 1 命令，機械語では 3 命令の OR 命令が，それぞれクリティカルセクションであり，排他制御が必要な命令列となる．

```
// C や C++言語の OR 命令
    flag |= 0x01;    // ビット 0 を ON

// RISC（MIPS）における上記命令の機械語（アセンブラニーモニック）

    ld reg1, flag
    or reg1, 0x01
    st reg1, flag



// C や C++言語の OR 命令
    flag |= 0x02;    // ビット 1 を ON

// RISC（MIPS）における上記命令の機械語（アセンブラニーモニック）

    ld reg1, flag
    or reg1, 0x02
    st reg1, flag
```

図 1-4　C 言語と機械語

1.4　組込みシステムにおけるリアルタイム性

組込みシステムの，現実世界の変化に対する反応は一定時間以内でなければならない．また組込みシステムに求められるものは機能的な正確さに加えて時間的制約がある．言いかえると決められた時間内に処理を完了しなければならない．これがリアルタイム性である．

つまり，組込みシステムはコンピュータシステムとしての演算の正確性以外に，あらかじめ定められた（求められた）時間以内に演算結果を返すという，リアルタイム性も必要となる．

以上のように組込みシステムには

・並行性

・排他制御

・リアルタイム性

が必要となる．

1.5　割り込みと割り込み処理

組込みシステムは現実世界との相互作用により，現実の変化に応じて制約された時間以内に適切な処理を行うことが必要であり，センサからの割り込み信号を通じて現実世界の変化を認識する．

図 1-5 に CPU とセンサ及び割り込みの関係を示す．CPU の割り込み信号にセンサデバイスからの割り込み信号が入力されることによって，CPU はあらかじめ登録されている割り込み処理の実行を開始する．センサは割り込みを利用して次々に発生する現実世界の変化を CPU へ伝える．そのため多様なタイミングおよび種類の割り込みに対して，CPU は適切に対応しなければならない[2][3].

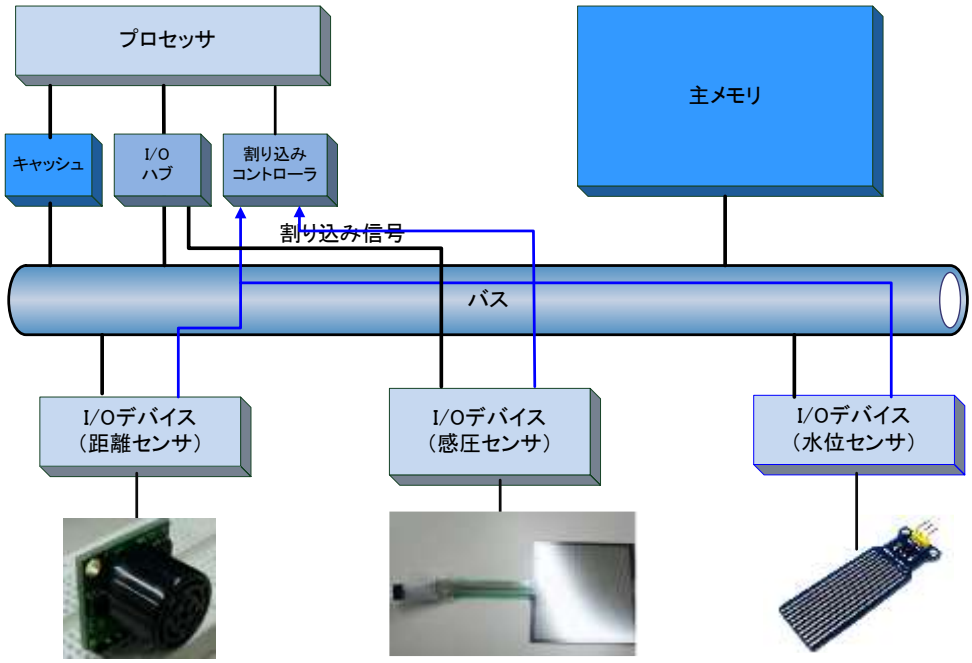


図 1-5　CPU と割り込み

割り込みにはソフトウェアから発行（実行）できるソフトウェア割り込みもあるが，本論文中では特に断らない限り，現実世界からの組込みシステムに対する何らかの処理を要求する手段である外部割り込み(ハードウェア割り込み，ペリフェラル割り込みとも言う)を割り込みと呼ぶ．さらにその要求に対して CPU ハードウェアが実行する処理を割り込み処理と呼ぶ.

図 1-6 に CPU ハードウェアが備える PSW（Processor Status Word）の一例

を示す．図は M16C/62A シリーズの flag レジスタである．

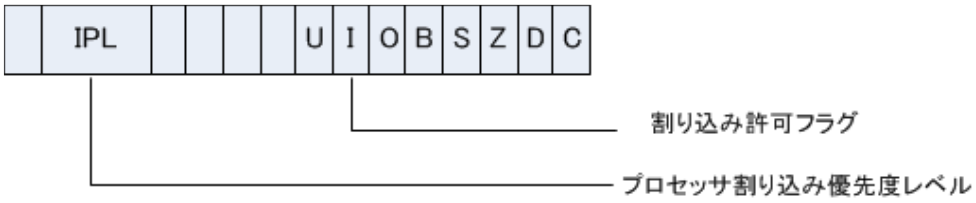


図 1-6　PSW

通常 CISC 系の PSW レジスタには，割り込みに関連する 2 種類のフラグ（ビット）が存在し，ハードウェアからも設定されるが，ソフトウェアからも設定することができる．

この例では，図における I フラグがすべての外部割込みの禁止と許可を行うフラグである．このフラグが ON の時はすべての優先度レベルの外部割込みが許可され，OFF の時は NMI（Non Maskable Interrupt）を除くすべての優先度の外部割込みが許可される．また IPL（Interrupt Privilege Level）は許可する割り込みのハードウェア優先度を指定するフラグである．M16C の場合割り込みレベルは 0 から 7 まであり，数字が大きいほど優先度が高い．0 はすべてのレベルの外部割込みを許可する非割り込み状態，7 は割り込みを禁止できない NMI として扱われる．両方のフラグでは I フラグが優先され，I フラグが ON，つまり割り込み許可状態に設定されている場合のみ IPL フラグの値が有効になる．IPL が 1-6 レベルに設定されている場合は，それ以下の割り込みが発生しても CPU ハードウェアがマスクして，割り込み処理（ソフトウェア）を実行させることはない．

表 1-1 に CISC 系の CPU における，多重割り込みを許可しない場合の I/O デバイス，CPU ハードウェア，割り込み処理(ソフトウェア)の動作を示す．

I フラグが OFF の場合は，割り込みが発生しても CPU ハードウェアによりマスクされ割り込み処理が実行されることはない．I フラグが ON の時に割り込みが発生すると CPU ハードウェアが I フラグを自動的に OFF してから割り込み処理が実行される．その割り込み処理実行中は，より優先度の高い割り込みが発生しても，CPU ハードウェアがマスクするため，その高い割り込みに対応した割り込み処理が実行されることはない．

実行中の割り込み処理が実行を終了して，割り込みからの復帰命令を実行すると CPU の状態が割り込み発生前の状態に戻るため，そこで改めて優先度の高い割り込み処理が呼び出される．

表 1-1　多重割り込みを許可しない場合の割り込み発生時の動作

I/O デバイス	CPU ハードウェア	割り込み処理 （ソフトウェア）
外部環境の変化により，設定に従って，CPU に割り込み信号を入力	各命令実行終了時に割り込み信号が入力されていないかチェックを行う． 割り込み信号が入力されており，PSW の割り込み許可フラグが ON の場合は，次に実行する命令を保持しておる PC（プログラムカウンタレジスタ）と PSW をスタックに保存する． その後 PSW の I フラグを OFF にして，IPL を発生した割り込みのレベルに設定する． 最後にあらかじめ登録されている割り込み処理の先頭アドレスを PC に設定する． この後，他の優先度が高い割り込みが発生しても CPU がマスクする．	CPU から呼び出されて，予め作成されている割り込み処理（ソフトウェア）を実行する． この間，他の割り込み処理によってプリエンプトされることはない． 割り込み処理終了時には，割り込みからの復帰命令を実行する． 割り込みからの復帰命令は，保存してある PSW と PC を復元する命令であり，この命令の実行により CPU は，割り込み発生前の状態に戻る．

表 1-2 に CISC 系の CPU における，多重割り込みを許可する場合の I/O デバイス，CPU ハードウェア，割り込み処理(ソフトウェア)の動作を示す．多重割り込みを許可する場合は，割り込み処理の中で，すべての割り込みを許可する，PSW の I フラグを ON にすることによって割り込み処理を有効にする．I フラグが ON の場合は IPL の値が有効になり，IPL の値より高いレベルの割り込みが発生した場合は，それに対応する割り込み処理によって，実行中の割り込み処理はプリエンプトされる．

表 1-2　多重割り込みを許可する場合の割り込み発生時の動作

I/O デバイス	CPU ハードウェア	割り込み処理 （ソフトウェア）
外部環境の変化により，設定に従って，CPU に割り込み信号を入力	各命令実行終了時に割り込み信号が入力されていないかチェックを行う． 割り込み信号が入力されており，PSW の割り込み許可フラグが ON の場合は，次に実行する命令を保持しておる PC（プログラムカウンタレジスタ）と PSW をスタックに保存する． その後 PSW の I フラグを OFF にして，IPL を発生した割り込みのレベルに設定する． 最後にあらかじめ登録されている割り込み処理の先頭アドレスを PC に設定する． 割り込み処理により割り込み許可フラグ I が ON に設定された後で，IPL に設定されているよりも高いレベルの割り込み信号が入力された場合は，次に実行する命令を保持しておる PC（プログラムカウンタレジスタ）と PSW をスタックに保存する． その後 PSW の I フラグを OFF にして，IPL を発生した割り込みのレベルに設定して，登録されている割り込み処理の先頭アドレスを PC に設定する．	CPU から呼び出されて，予め作成されている割り込み処理（ソフトウェア）を実行する． 割り込み禁止で実行しなければならない必要最小限の命令実行後に，割り込み許可フラグ I を ON にする，これで多重割り込みが許可される． 割り込み処理終了時には，割り込みからの復帰命令を実行する． 割り込みからの復帰命令は，保存してある PSW と PC を復元する命令であり，この命令の実行により CPU は，割り込み発生前の状態に戻る． 割り込み許可フラグ I を ON にした後は，より高い優先度の割り込みが発生した場合は，その割り込みに対応する割り込み処理によってプリエンプトされる．

1.6 割り込み，割り込み処理，割り込み制御

本論文中では「割り込み」とはセンサやアクチュエータなど，CPU カーネル以外のデバイスやスイッチから受け取るハードウェア的な信号（要求）とする．また，この「割り込み」要求の入力により，CPU ハードウェアが自動的に実行するソフトウェアを「割り込み処理」とする．

「割り込み制御」とは，従来方式では実現できなかった，割り込み処理のリアルタイム性を損なわない排他制御方式やスタックオーバーフローの検出方式の実現とする．

1.7 割り込みとタスク

表 1-3 に優先度付きの割り込み処理と RTOS のタスクの比較表を示す．

優先度付きの割り込み処理とタスクは，優先度を持った並行処理という意味では似た部分もあるが，割り込みの場合は優先度の処理が CPU ハードウェアで実行されるのに対し，タスクは RTOS というソフトウェアで実行される．また排他制御に関して，割り込み処理で使用される割り込み禁止／許可がシステム全体に影響を及ぼしリアルタイム性を損なうのに対して，タスクで使用されるセマフォなどは，その影響が関連するタスクだけに限定されるため，リアルタイム性を損なうことはない．

一方，RTOS はアプリケーションから見ると，RTOS 自体がオーバーヘッドであり，ブラックボックスである．そのため実際には，RTOS 自身のためのシステム割り込みや，RTOS 内部の排他制御のための割り込み禁止／許可のためにリアルタイム性を損なっているが，その影響が見えにくいという問題がある．

1.8 本論文の構成

本論文は，主に MMU や OS を使用しない組込みシステムにおける割り込み処理に関して，排他制御に伴うリアルタイム性の向上の研究，CPU ハードウェアを利用したリアルタイム性の向上の研究，割り込み処理のスタックオーバーフローの検出による割り込み処理の品質向上に関する研究で構成する．具体的には以下で構成する．

第 2 章では割り込み処理の現状分析と課題について述べる．第 3 章では割り込み処理に関する関連研究について述べる．第 4 章では上記の 3 つ研究に

表 1-3　タスクと優先度付きの割り込み処理の比較

	RTOS使用	RTOS未使用*
並行性の実現手段	タスク	割り込み処理
排他制御の手段	セマフォが代表だが他にもRTOSが提供する多くの手段がある	割り込み禁止および許可
優先度	ソフトウェア(RTOS)で付加するため，自由につけることが出来る 一般的には256(255)レベル レベル数はRTOSにおける重要なデータ構造の一つであるシステムREADYキューの構造に影響を与える	外部デバイス(PICなど)の追加も含めて，ハードウェアで付加する場合が多い ハードウェアに優先度がない場合はソフトウェアで優先度をつける場合もある ハードウェアによりレベルは異なるが16レベル程度の物が多い
排他制御の(悪)影響	関連するタスクのみに限られる 関連しない優先度の高いタスクには影響を与えない 優先度逆転などによるデッドロックに注意	影響がシステム全体，すべての割り込み処理に及ぶ 排他制御に無関係の優先度の高い割り込み処理も停止させる
並列実行実体の情報の保存場所	TCB RTOSはアドレスを知っている	割り込みスタック RTOS使用時も割り込み処理に関してはRTOSの管理外，保存アドレスもRTOSは管理外
排他制御の特徴	多種類に及ぶが使い方を誤ってもシステム全体に影響を及ぼすことはまれ ただしデッドロックには注意が必要	比較的簡単に使えるが，使い方を誤るとシステム全体の性能に影響を及ぼす 最悪システムダウンを招く

＊RTOS 使用時の割り込み処理も同じ

関して述べる．第 5 章では提案方式の評価と考察を行う．第 6 章では本論文をまとめ，今後の課題を述べる．

第２章

割り込み処理の現状分析と課題

本章では，まず，現在の組込みシステムにおける割り込み処理の実行方式に関して RTOS を使用する場合と使用しない場合について述べる．次に，割り込み処理の排他制御における問題について述べる．最後に，割り込み処理における優先度逆転の問題とスタックオーバーフローの問題について述べる．

2.1　組込みシステムにおける割り込み処理

1.4 節で述べたように，組込みシステムにおいては，予め定められている一定時間以内に処理を終えることが保証できることは重要である．これをリアルタイム性の保証とよぶ[4][5][6][7].

リアルタイム性の保証のためには，割り込みによって伝えられるさまざまな要求に対してその要求の重要度に応じて優先度をつけられることが必要である．というのは，優先度があれば，何らかの処理実行中に外部からより優先度が高い処理要求があった場合に，直ちにそれに応答する処理の実行が可能になるからである．その場合優先度が高い処理の実行中は，優先度が低い処理は一時停止状態となるが，優先度の高い処理終了後は，優先度の低い処理は再開できなければならない．これが並行処理である．処理に重要度をつけることは，システムを実現するエンジニアの責任である．

割り込みは現実世界の変化を組込みシステムに伝えるものであり，1.2 節で述べた現実世界の変化の特徴を伝えるための割り込みにも次のような特徴がある．

・割り込みは非決定的に発生する．

・割り込み処理には時間制約がある．

・複数の割り込みが同時に発生する可能性がある．

以上の割り込みの特徴から，割り込み処理を効果的に実現する手段として，優先度を備えた並行処理の実現手段は重要である[7][8][9][10].

2.2　割り込み処理の実行方式と問題点

現状使われているさまざまな割り込みの処理方式について説明する．

まず，RTOS を使用している場合の割り込み処理の実現方式を説明し，その後 RTOS を使用していない場合の割り込み処理の実現方式を説明する．

2.2.1　タスクによる割り込み処理の実行

組み込みシステムにおいては，使用している CPU の性能が比較的高く，更にメモリ容量も比較的多い場合は RTOS を搭載して並行処理を実現する．

RTOS が提供する機能であるタスク，スレッドあるいはプロセス（以後タスクと記す）を用いれば，容易に並行処理を実現することができる[11][12][13].

RTOS を利用して割り込み処理を実現する場合，割り込みに対する応答性能を確保するため，割り込み処理自体での処理は最小限にして処理本体はタスクで実行する方法が多い，タスク実行中は割り込みを許可しているからである．一方，割り込み処理は割り込み禁止状態で実施されるため，その割り込み処理が終了するまで，実行中の割り込み以外の割り込みに対する応答ができないためである．多重割り込みを許可している場合でも，同じ優先度以下の割り込みは禁止されている．

2.2.2　割り込み優先度をタスクの優先度に反映させる方式

各レベル（優先度の）割り込みに対して，ひとつの割り込み処理とひとつのタスクで実行する方式である．

割り込みが発生すると割り込み処理が呼び出されるが，そこでは必要な処理はほとんど実行せず，セマフォなど割り込み処理からタスクに知らせることが出来る手段を用いて，対応するタスクに対して処理の実行を依頼する方式である．この方式では必要な処理の大部分はそのタスクで実行する．割り込みの優先度はタスクの優先度に反映させる．

この方式の問題点は，タスクの数の増加よるシステムのオーバーヘッド

と，割り込み処理からタスクに通信する時同期をとらねばならないことによるオーバーヘッドの増加である．

2.2.3　単独のタスクですべての割り込み処理を実行する方式

割り込み処理から依頼されて必要な処理を行うタスクを一つにまとめたものである．図 2-1 にこの方式を示す．図において割り込み発生時に呼び出さ

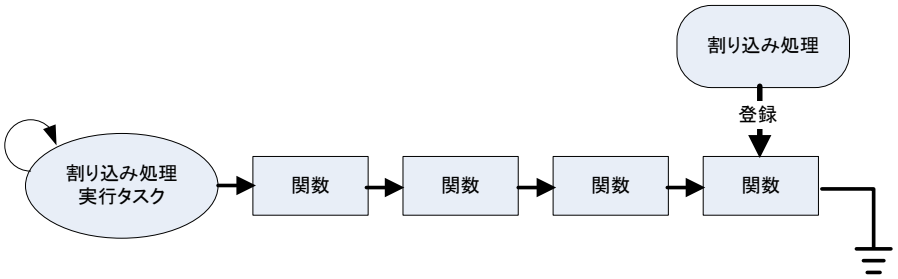


図 2-1　単独タスクによる割り込み処理の実行

れる割り込み処理では必要最低限の処理のみを行い，その他の処理は割り込み処理を行う最高優先度の専用のタスクに実行を依頼する．具体的には，割り込み発生時，割り込み処理を実行するタスクの要求処理キューに処理（関数）登録する．割り込み処理を実行するタスクは処理要求キューにつながれた処理を順次実施する．タスク実行中は割り込みを許可しているため，タスクが処理を実行している最中に発生した割り込みにも対応できる．たとえば VxWorks などの割り込みの処理はこの方法を採用している．

この方式の問題点は割り込み処理をタスクに依頼するための処理要求キューは FIFO 方式であるため，割り込みの優先度が処理の優先度に反映されないという点である．デッドロック発生を招く可能性があるため，キューを優先度順に構成することはできない．

割り込み処理をタスクで実行するという方法は，割り込み処理を直接制御する方法ではない．そのためハードウェアの不具合などの原因により不正割り込みが多数発生してスタックオーバーフローを起こしても，その原因の特定に時間を要するという問題が発生している．

2.2.4　最低優先度の割り込みですべての割り込み処理を実行する方式

この方式は前述の方式と同様であるが，最高優先度のタスクではなく，最低優先度の割り込み処理ですべての割り込み処理を実行する方式である．通常は優先度の低い割り込みは通常周期的なタイマー割り込みを利用する．多重割り込み許可状態で，最低優先度の割り込みを利用するため，割り込みに対する応答性は損なわれない．

この方式も前述の専用タスクで処理する方式と同様に，割り込みからの要求を保存するキューが FIFO 方式で実装されているため，処理が割り込みの優先度に対応できないという問題がある．

また処理の有無に関わらず周期的な割り込みを入力する必要があるため，それに対応した割り込み処理の実行が必要になる．さらに，ハードウェア的に割り込みに優先度がない CPU では実現できないという問題点もある．

2.2.5　ハードウェアの割り込み機能のみを使用した方式

ここまで RTOS を使用した場合の割り込み処理の排他制御方式に関して述べた．組込みシステムにおいては特に上位機種を中心に RTOS を使用することが多い．しかし内蔵メモリの容量の少ない特に 16 ビット以下のシングルチップマイクロコンピュータ（以後シングルチップマイコンと記す）を用いた組込みシステムにおいては，RTOS を搭載しない場合も多い．市場においては 16 ビット以下のシングルチップマイコンを使用した組込みシステム製品は一定のシェアを保っている．

図 2-2 に経済産業省 (独)情報処理推進機構調査の 2010 年度版組込みソフトウェア産業実態調査報告書[14]に記載されたプロセッサの仕様個数の図を示す．2009 年度においては 21.8%が 16 ビットと 8 ビットの CPU を合計すると 35.2%となる．また図 2-3 に同じく，組込みシステムが使用している OS の種類の図を示す．2009 年度においては 20.4%の種類の組込みシステムが OS を使用していない．2007 年度は 24.5% [15]，2008 年度は 28.0% [16]と 20 から 30%近くは RTOS を搭載していない．

シングルチップマイコンを用いた組込みシステムは低価格で大量製造され RTOS を使用しないことが多く，この調査が件数ベースであることから，個数ベースでは RTOS を使用しない組込みシステム製品の個数の割合は更に増加すると思われる．

そのため RTOS を搭載していない組込みシステムに関する研究を行うこと

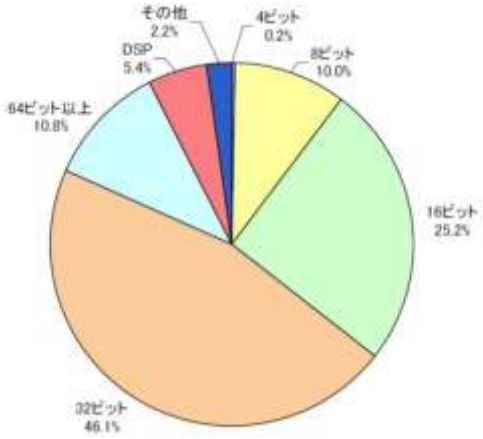


図 2-2　CPU のビット数(2009 年度)

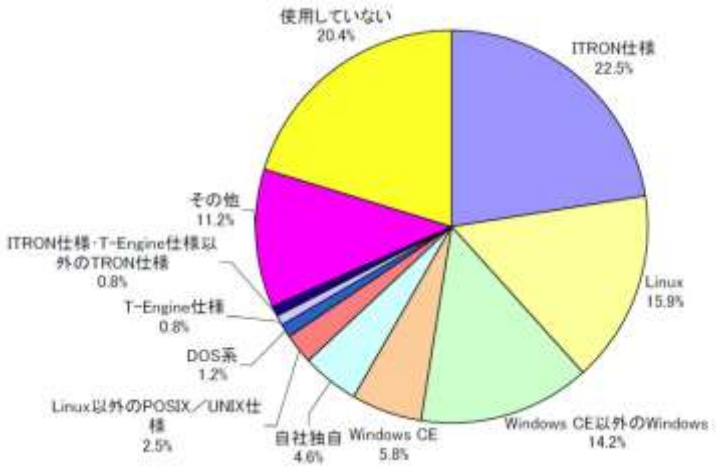


図 2-3　使用している OS の種類(2009 年度)

は重要となる．たとえば，業務用のエアコンでは 1 台の室外機で 10 台以上の室内機の温度設定や冷媒の送出のためのコンプレッサーの制御などを RTOS なしで行っている．室外機で割り込みによって行わねばならない制御は

・冷媒の送出制御（コンプレッサーの制御）
・全室外機動作の制御
・温度，湿度など外部条件のチェック
・通信のチェック
・設定データの反映
・電源低下のチェック
・EEPROM の書き込みタイムアウトのチェック

など多岐に渡る．現実に，RTOS を使用しない業務用エアコンの室外機は15 種類の割り込みにより処理を行っている．

また車に約 100 個使用される ECU（Electric Control Unit）においても，ほとんど RTOS は使用せず割り込み処理のみで処理を行う．ECU においてはバスからの割り込みが多数あるため，排他制御に苦労している．

ところで，組込みシステム実現のためには，並行処理の実現は必須であり，RTOS を搭載していない組込みシステムで並行処理を実現するためには割り込み処理を利用する．

図 2-4 に CPU が備える割り込み機能を単純に利用した方式を示す．

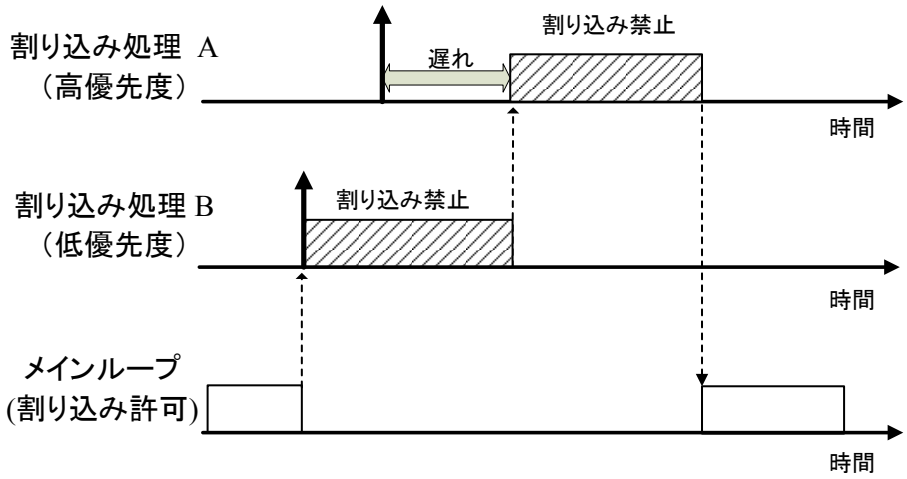


図 2-4　ハードウェアの割り込み機能のみを利用した方式

図において，上向きの矢印はその処理の実行要求元の割り込みが発生した時を示している．白抜きの四角は実際にその処理を実行していることを，また斜線部は割り込み禁止で，処理が実行されていることを示している．

メインループとは，常に実行している無限ループであり，すべての割り込みを許可している．つまり，非割り込み環境で実行している．メインループ実行時に，外部割り込みが発生すると CPU は，自動的に外部割り込みをすべて禁止状態にして割り込み処理を実行する．

この状態で新たな割り込みが発生した場合は，実行中の処理の優先度より，新たな割り込みの優先度が高い場合でも，新たに発生した割り込みは保留される．割り込み処理が割り込みからの復帰命令を呼び出され，CPU の状態が復帰すると，保留された割り込み処理が再開させられ，対応した割り込み処理を実行する．図 2-4 で「遅れ」と記述している区間がその割り込み処理が実行されない遅れ時間を示している．

2.2.6　フラグを利用した使用した方式

前述の方式では割り込み禁止時間が長く，割り込みに対する応答性が劣る．それを改善するのがフラグ方式である．図 2-5 にフラグ方式を示す.

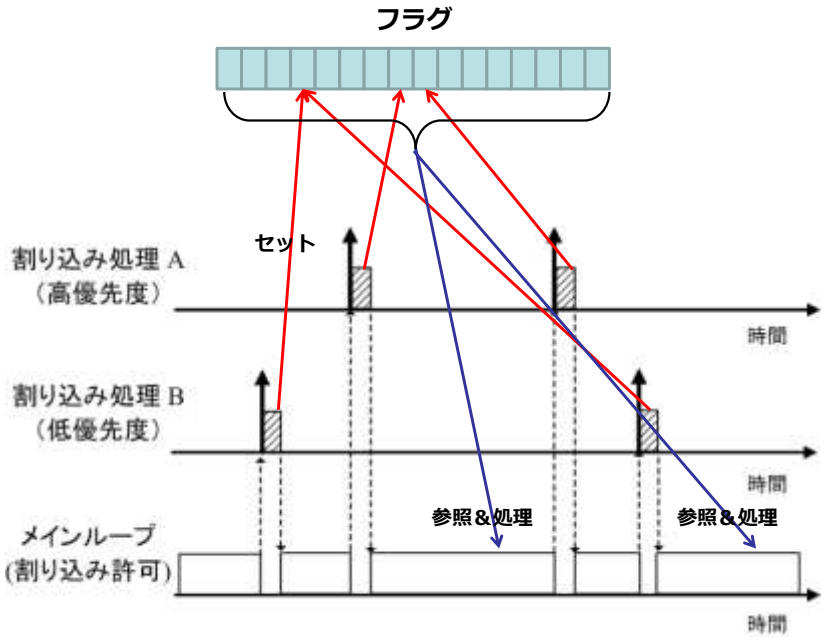


図 2-5　フラグを利用した方式

割り込みが発生した時，割り込み処理内で割り込みが発生したことを示すフラグをセットして処理を終了する．フラグのセットだけなので，処理時間は短く，従って割り込み禁止の時間は短い.

割り込みの種類によって別々のフラグを設ける．そして非割り込み状態で無限ループとして構成されているメインループが，フラグを順番に検査して，フラグがセットされている場合にそれに応じた処理を実行する.

この方式の長所は排他制御に伴う問題を考慮しなくてよいことであり，車載機器の ECU においてよく利用されている.

逆にこの方式の問題点は，処理が実行される周期がメインループの周期に依存することである．システムの機能増加に伴い，割り込みや割り込み処理の種類や処理内容が増えフラグのビットが増加する．それに伴いメインループで検査するフラグの書類や実行される処理が増加し，インループの 1 回の実行時間(処理周期)が長くなる.

その結果，割り込みの発生にメインループの処理が追いつかない場合があり，割り込みが複数回発生しフラグが複数回設定されるにも関わらず，それに対応した割り込み処理は 1 回しか実行されないという問題点がある.

さらに，割り込み処理の実行順序がメインループ内のフラグを検査する処理の位置と，割り込み発生時にメインループのどの場所を実行していたかに影響されるため，優先度が反映できないという問題がある．

また，ある処理を実行中に，最優先で処理しなくてはならない割り込みが発生しても，メインループのその割り込みに対応するフラグの検査をするまでその処理の実行を行うことはできないため，割り込みに対して優先度をつけることが出来ないという問題もある．

2.2.7　多重割り込み方式

CPU ハードウェアが割り込み優先度を備える場合はそれを利用し，RISC 系 CPU のように割り込み優先度がない場合は，外部に割り込みコントローラを設けて割り込みに優先度を設けるなどによる，割り込み信号の優先度を利用する方法である[17][18][19].

図 2-6 に多重割り込み方式を示す.

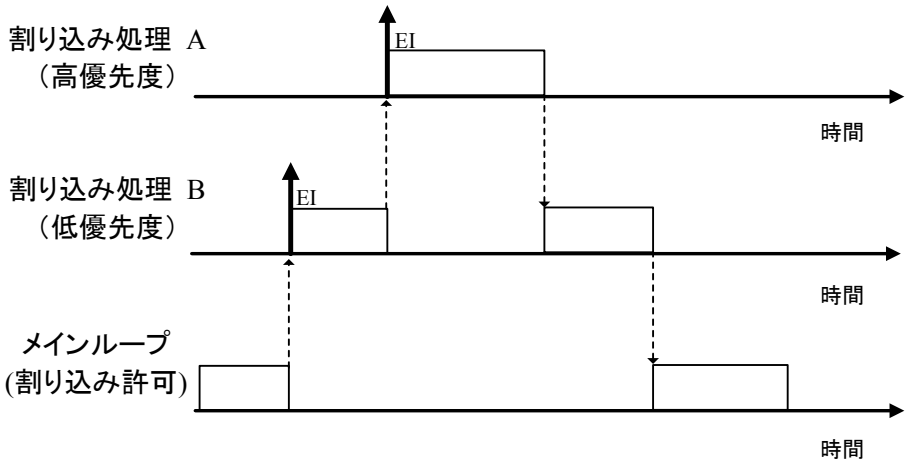


図 2-6　多重割り込み方式

図中の EI とは Enable Interrupt の意味で，ソフトウェアによる割り込み許可をしめす．通常割り込み許可および禁止は PSW における割り込み許可ビットの操作のみで実現できる．

優先度の低い割り込み処理中に，優先度が高い割り込みが発生した場合は，実行中の割り込みを一時停止させ，優先度の高い割り込みを実行し，それが終了した時に一時停止させていた割り込み処理を再開させる．

RTOS 未使用時の割り込み処理方式として最も良く使用される方法であり，RTOS 使用時の割り込み処理の実行方式としても一般的である．

2.3　割り込み処理の排他制御における問題

並行処理を実施する場合は，クリティカルセクション(以後 CS)の排他制御機能が必要となる．CS とは，同時に実行すると，データの一貫性が失われるなどの不具合が発生する可能性がある一連の命令区間のことである．

RTOS 使用時は，タスク間の CS の排他制御にはセマフォが使用される．

セマフォを使用すれば，複数の並行処理が同時に CS を実行しようとする場合のみ効果が表れるため，必要な区間のみの排他制御実現が可能である．しかし RTOS 不使用時，あるいは RTOS を使用している場合でも割り込み処理はセマフォを使用することが出来ないため，割り込み処理間の CS の排他制御には CPU の全外部割り込み禁止(Disable Interrupt - DI)と全外部割り込み許可(Enable Interrupt - EI)が使用される．

セマフォの排他制御の影響が，関連するタスク間に限定されるのに対し，DI/EI による割り込み処理の排他制御はシステム全体に及び、排他制御に無関係の部分や優先度が高い割り込み処理の実行までも阻害する．その結果システムのリアルタイム性が阻害される．

図 2-7 に割り込み禁止/許可を用いた排他制御の動作を示す．図において DI と示している部分が割り込み禁止命令の実行を示している．この命令の実行後は割り込み許可 EI が実行されるまで CPU は全ての外部割り込みを受け付けない．割り込みを受け付けないことによって全ての並行実行が禁止されるので，それを利用して排他制御を行う．

その影響を関連タスクのみに限定できるセマフォと異なり，割り込み禁止の影響はシステム全体に及ぶ．図に示すように本来はメインループと割り込み処理 B だけの排他制御が，両者よりも優先度の高い割り込み処理 A の実行も遅らせている．そのためシステムのリアルタイム性が保証できない．さらに本来の排他制御は割り込み処理 B がメインループと競合する部分だけであるのに，割り込み処理のそれ以外の部分の実行も遅らせるという問題もある．

図 2-8 にある実在の組込みシステム製品におけるフィールド不具合（製品を販売してから見つかった不具合）の原因の件数を示す．図 2-9 にその原因毎の割合を示す．リアルタイム設計時の不具合の割合が多い．図 2-10 にリアルタイム設計時の不具合の詳細要因を示す．

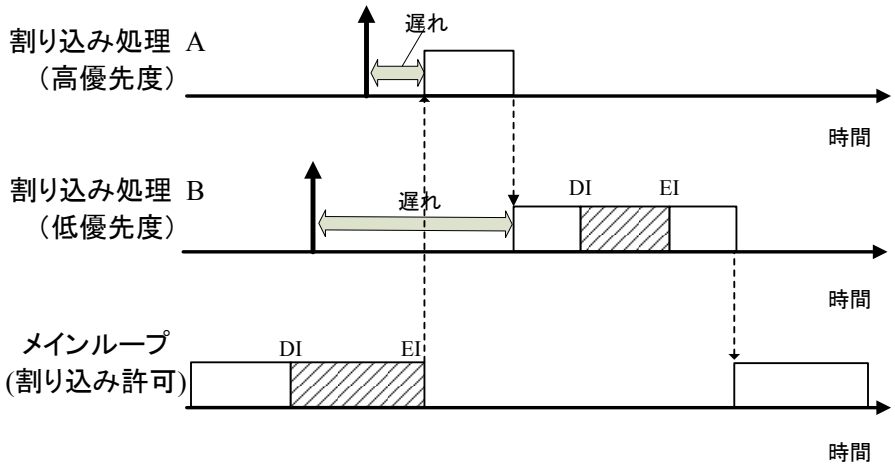


図 2-7　割り込み禁止と許可を使用した排他制御方式

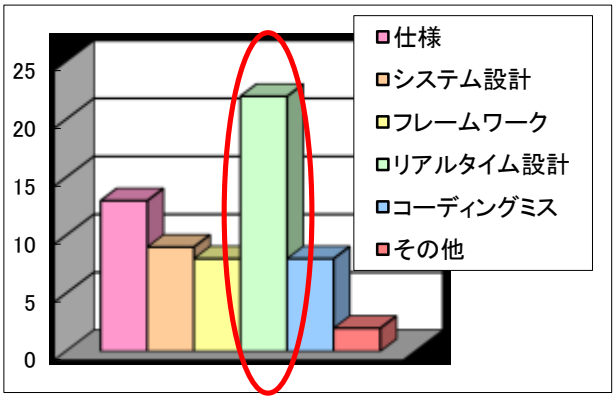


図 2-8　実製品における市場流失不具合件数

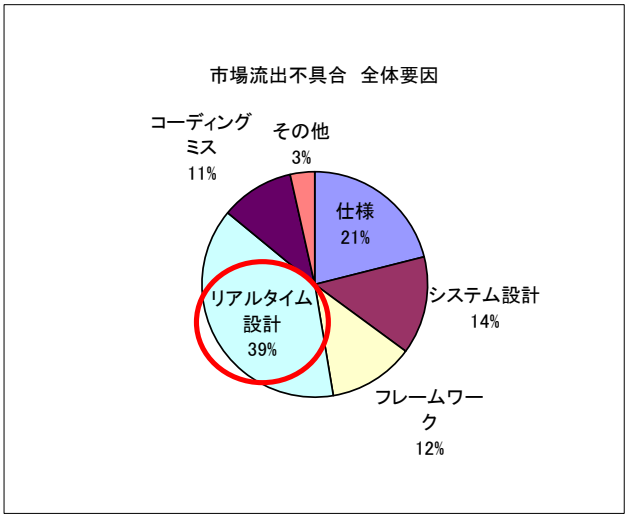


図 2-9　実製品における市場流失不具合割合

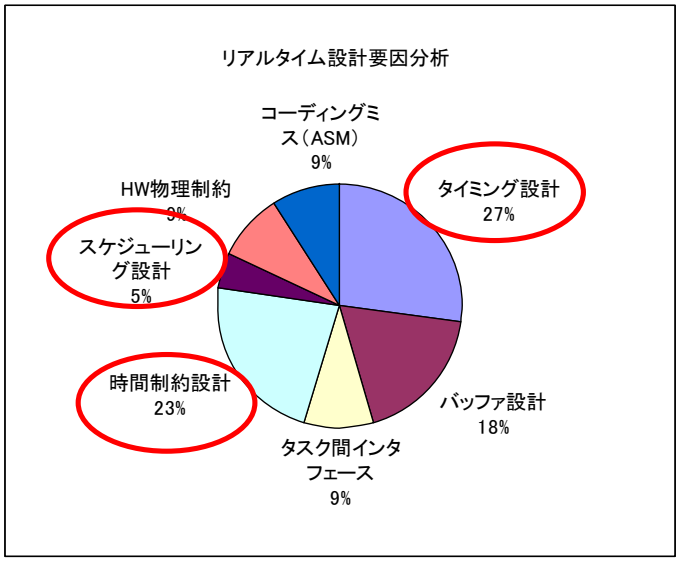


図 2-10　リアルタイム設計ミスの要因分析

図において赤字のまるで囲んでいるタイミング設計ミス，時間制約設計ミス，スケジューリング設計ミスは，その要因が割り込み禁止により，リアルタイム性が損なわれたためである．

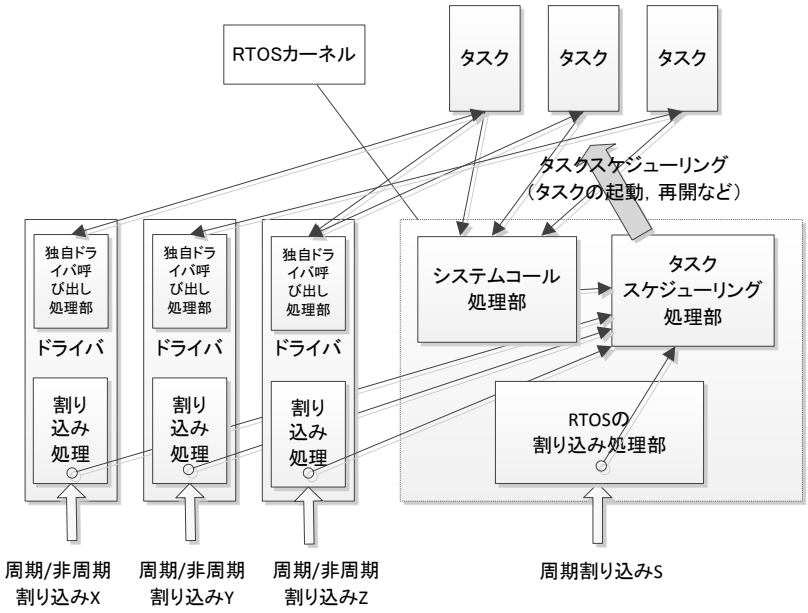
一方 RTOS を使用して排他制御を実施する場合にも問題がある．

図 2-11(a)，図 2-11(b)に RTOS の基本的な構造を示す．図 2-11(a)はμITRON 系のようにデバイスドライバの標準化を行っていない RTOS の例である．図 2-11(b)は VxWorks のようにデバイスドライバの標準化を行っている RTOS の例である．タスクと表示した矩形はタスクを示し，ドライバと表示した矩形はドライバー（割り込み処理）を示し，点線の四角で囲んだ部分が RTOS カーネルを示す．

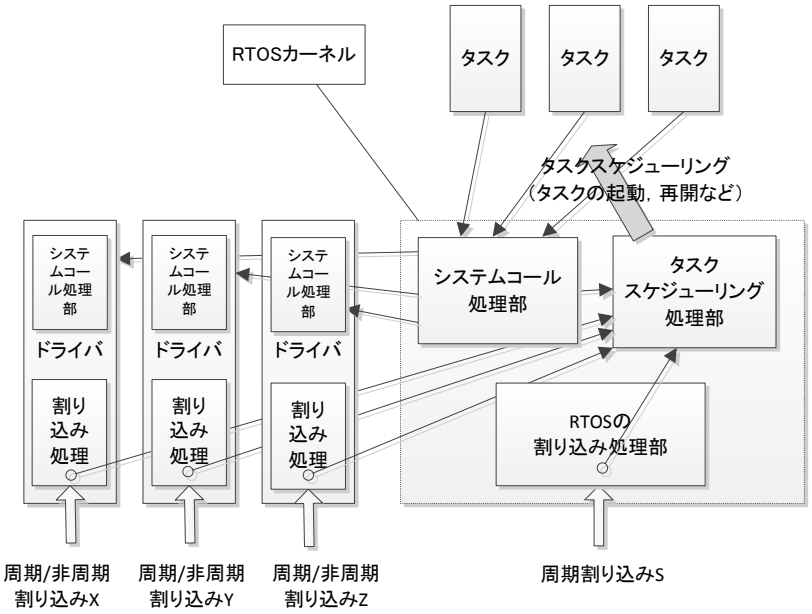
デバイスドライバはタスクから呼び出されタスクコンテキストで実行される部分と対象デバイスからの割り込みにより実行される割り込み処理から構成されるが，タスクコンテキストで実行されるのは処理の最初と最後だけであり，大部分の処理は割り込み処理で実行される．

I/O を標準化していないμITRON 系の RTOS の場合，タスクに対するドライバのインターフェースはさまざまであり，I/O を標準化している VxWorks のような RTOS の場合はタスクに対するドライバのインターフェースは統一されている．

しかし，デバイスドライバがタスクコンテキストで実行される部分と割り込み処理から構成されることは同じであり，割り込み処理は最後に RTOS の



(a)　I/O の標準化を行っていない RTOS の基本的な構造



(b)　I/O の標準化を行っている RTOS の基本的な構造

図 2-11　RTOS の基本的な構造

スケジューラを呼び出すことも同じである．タスクが自ら RTOS に対して処理を要求できるのは RTOS が提供するシステムコールの呼び出しのみであり，タスクの実行開始，プリエンプト，ディスパッチなどタスクのスケジュ

ーリングはRTOSによって実行される．

それに対してドライバは，CPU に対する周期的あるいは非周期的なハードウェア割り込みにより，CPUハードウェアによって直接実行が開始される．

ドライバは，セマフォ解放操作など，タスクと同期をとるために RTOS が提供している，システムコールの一部を呼び出すことは可能である．しかしドライバは，待ち状態にはなれないため，セマフォ獲得要求など“待ち状態”になる可能性のあるシステムコールを呼ぶ出すことはできない．

多重割り込みを使用している場合，ドライバ（割り込み処理）はより優先度の高い割り込みにより，実行を一時停止させられる（プリエンプトされる）場合はあるが，ドライバが再開させられる順序は，一時停止させられた順番の逆の順番のみである，つまりドライバの一時停止，再開は入れ子構造になっていなければならない．

タスクの場合は，プリエンプトされた順番と再開される順番には，このような制約はない．

図 2-12 に割り込み処理が呼び出される場合の入れ子構造を示す．割り込みが許可された状態で割り込み 1 が発生すると CPU ハードウェアが割り込み処理 1 の実行を開始する．割り込み処理の中で割り込みを許可することによって，新たに発生した，より優先度の高い割り込みが発生した場合は，その割り込み処理に移る．割り込み処理実行中の割り込 2 みによって割り込み処理 2 の実行が開始される．さらに割り込み 3 の発生によって割り込み処理 3 の実行が開始される．割り込み処理 3 の実行終了によって，割り込み処理 2 の実行が再開され，割り込み処理 2 の実行終了により割り込み処理 1 の実行が再開される．

以上のように割り込み処理は入れ子構造となる[20].

RTOS を使用していない場合，あるいは RTOS 使用時は割り込み処理中や RTOS 実行中に割り込まれた場合は，割り込み処理が処理を終了すると，割り込まれた箇所にもどり，実行を再開する．

RTOS使用時，タスク実行中に割り込まれた場合は，RTOSのスケジューラに制御は移る[21].

図 2-11(a)，図 2-11(b)に示すようにRTOSカーネルは

・システムコール処理部

・RTOSの割り込み処理部

・タスクスケジューリング処理部

から構成される．

システムコール処理部とはタスクやドライバから呼び出されるシステムコールの処理を行う部分である．タスクから呼び出された場合は，そのままタ

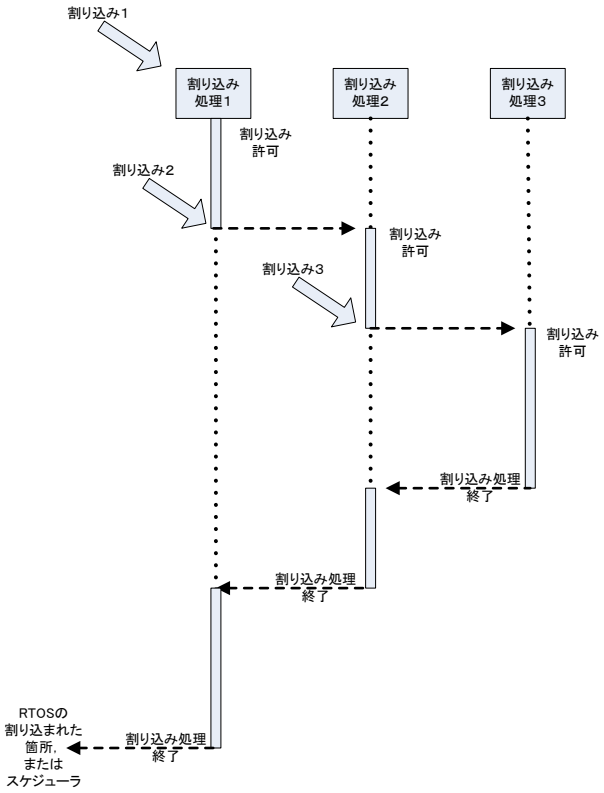


図 2-12　割り込み処理の実行の流れ

スクに復帰するものと，タスクスケジューリング部に制御が移る場合がある．

ドライバから，呼び出された場合は単なる関数と同じであり，通常スケジューラに制御が移ることはなく，そのままドライバに復帰する．

RTOS の割り込み処理部とは，RTOS に一定周期で入力する，システムチック，あるいはシステムクロックとも呼ばれるタイマー割り込みの割り込み処理である．RTOS はこの割り込みによって，一定周期でタスクに割り込んで，各種タスク制御をおこなうとともに，毎回の処理の最後では，他のドライバ（割り込み処理）と同様に，割り込み発生時の状況により割り込まれた部分に直接復帰するか，スケジューラに制御を移す．

タスクスケジューリング処理部は，システムコール処理部，あるいは他の割り込みが保留されていない場合には RTOS 割り込み処理部，あるいはドライバ処理の最後部分から呼び出されて，実行可能状態の最高優先度のタスクを起動したり，再開させたりする．

このように，RTOS を実装するには，タスクから RTOS への処理を切り替え，スリープ時間の基準などのため，周期的なタイマ割り込みが必要であ

る．これはシステム構成上アプリケーションに必要な割り込みではなく，RTOS のための割り込みである．

そのため組込みシステムを実現するための割り込みと以下の競合が発生する．

・割り込みの本数

・割り込み優先度レベル

・RTOS が排他制御を実現するための割り込み禁止/許可

割り込みの本数やレベルは特にそれらの数やレベルが少ないシングルチップマイクロコンピュータにおいては影響が大きい．

また RTOS の割り込みレベルを適切に設定することが難しい．たとえば，工作機械を制御する NC 装置では，他の割り込みを阻害しないように RTOS の割り込み優先度レベルは最低に設定していた．

また，RTOS 内部でもシステムレディキューを操作する場合など，RTOS と他の割り込み処理の排他制御のために，割り込み禁止を使用する．その割り込み処理はすべての割り込み処理に影響を与えるため，最悪処理時間の見積もりが必要な，リアルタイム設計を難しくする．

さらに RTOS を使用した場合は並行処理を実現する手段としては割り込み処理とタスクの二つの手段が存在することになり，組込みシステムのシステム設計を複雑にするという問題もある．

2.4　優先度逆転による問題点

排他制御のために使用するセマフォは，優先度の低い処理が，優先度の高い処理の実行を待たせることができるが，これはデータの不整合を防止するための例外的な仕組みであり，データの共有がない場合にまで使用すると優先度の低い処理の実行が，優先度の高い処理の実行より優先されることになり，リアルタイム性が阻害される．

そのため排他制御のためのセマフォの使用は CS 区間に限るわけであるが，直接データの共有がないにも関わらず，間接的に優先度の低い処理が優先度の高い処理の実行を阻害することがある．それが優先度逆転と呼ばれる[5].

図 2-13 に優先度逆転時の割り込み処理の動作を示す．高優先度の割り込み処理 A と低優先度の割り込み処理 C のセマフォによる排他制御実行中の中間優先度の割り込み処理 B の実行により，高優先度の割り込み処理 A の実行が遅らされる場合の動作である．

図において，上向きの矢印が割り込みの発生を，四角形が割り込み処理の

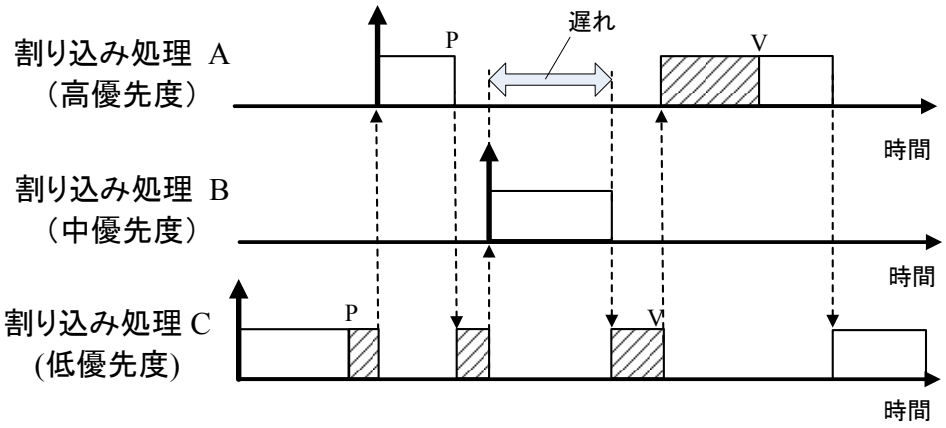


図 2-13　優先度逆転

実行を示し，斜線部が排他制御区間を示す．P が割り込み処理セマフォ獲得要求を示し，V が割り込み処理セマフォの解放要求を示す．遅れと記述している横線の矢印が優先度逆転による割り込み処理 A の実行の遅れを示している．

割り込み処理 C が P 操作によって排他制御開始後，高優先度の割り込みによって割り込み処理 A が実行を開始する．そして P 操作によって排他制御を開始しようとするが，セマフォは割り込み処理 C によってロックされているため，割り込み処理 A は待ち状態となり，割り込み処理 C が実行を再開させられる．その後割り込み処理 C は割り込み処理 B にプリエンプトされるため，割り込み処理 A がその分余分に遅らされる．これが優先度逆転である．高優先度の割り込み処理 A が，排他制御に無関係の，優先度の低い中間優先度の割り込み処理 B によって実行を余分に遅らされる．

優先度逆転により，高優先度の割り込みに対する応答性が阻害され，システムのリアルタイム性が低下する[5][6].

2.5　スタックオーバーフロー検出方式に関する問題点

前述のように，現実世界のさまざまな変化に対応するため，割り込みには優先度がつけられる．さらに，割り込み処理実行中に，より優先度の高い割り込みが発生した場合，実行中の割り込み処理を一時停止して，より優先度の高い割り込み処理に制御を移すという，多重割り込みを利用した多重処理が利用される．一方，約 30%の組込みシステムは，RTOS を使用していないた

め[14][15][16]，多重処理の実現のためにタスクではなく，割り込み処理を利用している．

ところでたとえば RTOS を使用していればタスクのスタックオーバーフローの検出は可能である[22]．しかし図 2-11 に示したようにタスクは完全に RTOS の制御下にあるのに対して，割り込み処理は RTOS よりも優先度が高いため，たとえ RTOS を使用していても割り込み処理のスタックオーバーフローの検出はできない．

さらにすべての割り込み処理がひとつのスタックを使用しているために，個々のスタックが明確でなく，スタックオーバーフローが発生しても，どの割り込み処理がスタックオーバーフローを発生させたのかの判定も難しい．

RTOS を使用していない場合も同様である．

図 2-14 に割り込み処理のスタックオーバーフローを原因として，自動車のスマートカードキーシステムで発生した人の車内閉じ込めを伝える記事を示す．2005 年に，自動車メーカー数社でスマートカードキーシステムにおいて，携帯電話などの電波の影響で，人間が車内に閉じ込められる不具合が発生した．これはスマートカードキー機能に対応する，ECU の割り込み処理が携帯電話の電波の影響でスタックオーバーフローを起こしたためである．

また NC 装置でも，ノイズによる不正割り込みが原因で，割り込み処理がスタックオーバーフローを起こした結果，システムダウンを起こし，原因の解明だけで 1 週間を要したことがあった．

タスクであれば，RTOS がスタックオーバーフローの検査を行うことが可能であるが，割り込み処理に関しては，従来は，スタックオーバーフローの検査を行うことはできなかった．さらに，タスクであれば，その実行パターンも限られるため，ある程度の予測が可能であり，予め試験をしておくこともできる．しかし，割り込みは，現実の環境に左右されるため，発生パターンの予測は難しく，予めスタックオーバーフローの試験をしておくことは難しい．

割り込み処理の処理においては，必要なスタック領域が，システム生成時に割り当てられる．しかし，割り込み処理ソフトウェア自体の不具合，ハードウェアの不具合，使用される環境による想定以上の頻度の割り込み処理呼び出しなどにより，割り当てられた領域以上のスタックを使用する場合がある．この場合，スタック以外のグローバル変数領域やプログラム領域が破壊される．これがスタックオーバーフローである．

一般的なコンピュータシステムでは，スタックオーバーフローの防止機能は MMU(Memory Management Unit)ハードウェアを備えた CPU と OS を用いるシステムが提供している[23]．予めスタックとして割り当てたメモリ領域

図 2-14　スマートキーレスエントリの不具合

以外をアクセスしようとした時に，MMU が検出して，OS に知らせる．OS はスタック領域を増やしたのち，スタックオーバーフローを起こした命令を再実行させる．そのためスタックオーバーフローは発生しない．しかし，前述のエアコン，車載機器など，コスト面，ノイズ対策などの点から，シングルチップマイクロコンピュータを使用する組込みシステムにおいては，割り込みレベルの少なさや，内蔵 RAM の容量の点からも RTOS を搭載することは難しい[24].

従って，一般的なコンピュータシステムの MMU と OS を使用したスタックオーバーフロー検出方式，スタック再割り当て方式は使用できない．

外部環境の変化に対応するため，割り込み処理が重要な意味を持つ組込みシステムにおいて，割り込み処理のスタックオーバーフローによる影響は重大である．

MMU を備えず RTOS を使用しない組込みシステムにおいても，割り込み処理のスタックオーバーフローを検出，防止できることは，組込みシステムの信頼性向上に有益である．

2.5.1　スタック領域の用途とその問題点

組込みシステムの開発言語として良く使用される C 言語や，C++言語のプログラムでは，グローバルメモリ領域はプログラム実行中，常時存在し使用される．また malloc や new により動的かつ明示的に獲得して使用されるヒープ領域も比較的長い期間使用される．これらの領域は，主にデータの保管場所として使用される．関数の配列など特別な使い方をする以外は，データあるいはデータのアドレスを保存する場合が一般的であり，命令のアドレスを保管することは少ない[25][26].

図 2-15 に C 言語や C++言語におけるスタック領域の使い方を示す.

図はある関数が上位関数から呼び出され，さらに下位関数を呼び出す様子を示しており，左側が命令メモリを右側がスタックを示す[25][26]．上位関数から呼び出された関数は最初に CPU のレジスタセットをスタック上に保存する．これは上位関数が使用しているレジスタセットを保存することによって，レジスタセットを自由に使用するためである．その後，関数内で使用する変数領域（ローカル変数）のためにスタック領域を確保する．ここまでが関数の処理を実行する前の準備段階である.

以後は関数として記述された命令列によってレジスタセットを適時使用しながら処理を行うが，通常は最初の方で上位関数からスタックを通じて渡される引数の受け取りを行う.

図では関数内でさらに下位関数の呼び出しを行う場合の動作も示している．下位関数を呼び出す場合，最初に下位関数に渡す引数をスタック上にコピーする．その後，関数呼び出し命令によって下位関数の実行を開始するが，その時関数の呼び出し命令によって，下位関数実行終了時に実行を再開する関数呼び出し命令の次の命令のアドレスがスタックに保存される．その後下位関数のアドレスが CPU の次に実行する命令のアドレスが保存するレジスターである PC（プログラムカウンタレジスタ）にコピーされる.

下位関数実行終了時には関数からの復帰命令（たとえば rts-リターンフロムサブルーチン）が実行される．この命令によりスタック上に保存されていた再開アドレスが取り出され，PC にコピーされる．これにより関数では下位関数を呼び出した次の命令から命令の実行が再開される．下位関数呼び出し後，通常は下位関数の戻り値の受け取りが実行される.

関数の処理終了時には，上位関数に戻す戻り値の設定を行い，関数で使用したスタック上のローカル変数領域の解放を行った後，スタック上に保存していた上位関数のレジスタセットの復元を行い，最後にスタック上に保存されていた再開アドレスが取り出され PC にコピーされ，上位関数の実行が再開

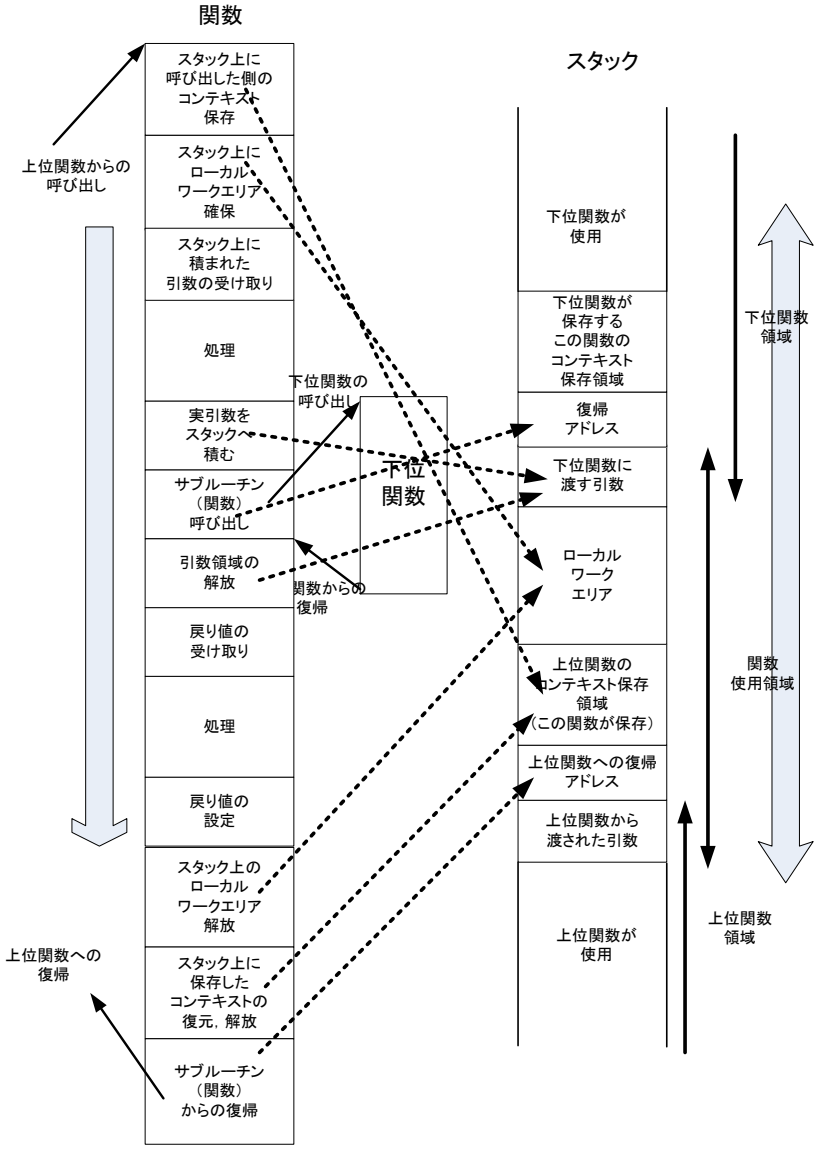


図 2-15　スタックの使われ方

される．

このように，スタックは，ローカルメモリ領域としての，グローバル変数領域と同じようにデータ領域しての使い方以外に，関数間のインターフェース領域としても使用され，関数から復帰した時に実行される命令のアドレスも保存される．

スタックには命令のアドレスも保存される．そのため，スタックオーバーフローにより命令のアドレスが破壊されると，以下のような不具合が発生する．

・不正アドレスの命令を実行して結果が不正になる．

・データであるのに命令として実行され不正命令例外などが発生する．

・メモリが実装されていないアドレスへのアクセスを行いアドレス例外などが発生する．

データが破壊される場合に対しては，データを使用する側でデータの正当性の検査を行い，データが不正な場合は，データを使用せずにエラー処理を行い，エラーの発生状況を記録して，それを次回に反映させるという対策をとる場合が多い．それに対して，命令のアドレスが破壊される場合は，エラーに対応するための命令自体を実行することができないため，対策が取りにくいという問題がある．

2.5.2　割り込み処理のスタック領域の使い方とその問題点

組込みシステムは，さまざまな現実世界の変化に対応するために，多重割り込みを利用した多重処理を行っている[2][3].

図 2-16 にあらためて割り込み処理の多重処理の動作を示す．図において，上向きの矢印が割り込み発生を，四角形が実行中の割り込み処理を示す.

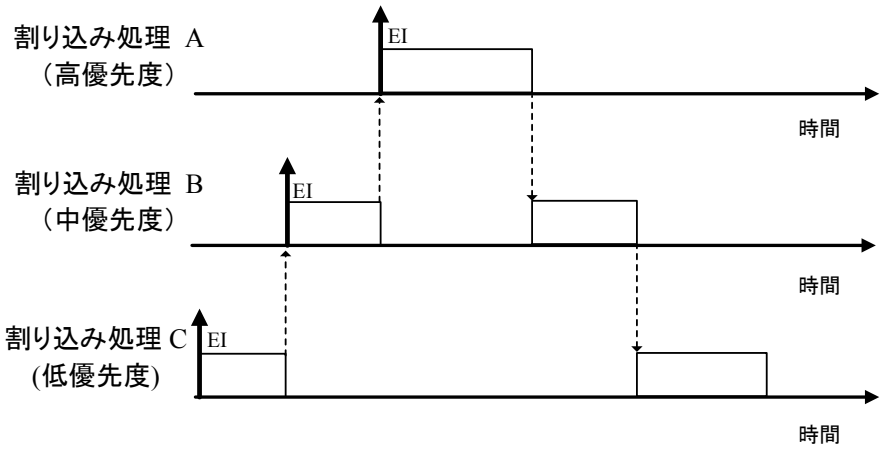


図 2-16　割り込み処理

低優先度の割り込みにより割り込み処理 C 処理実行中に，中優先度の割り込みが発生し割り込み処理 B にプリエンプトされる．さらに，割り込み処理 B 実行中に，高優先度の割り込みが発生し，割り込み処理 A にプリエンプトされた時の動作を示したものである．図では 3 個の割り込み処理が存在するが，すべての割り込み処理が同じスタック領域を共有する．割り込み処理 C がスタックを使用中に，割り込み処理 B にプリエンプトされた場合，使用しているスタック領域に続く領域が使用される．その割り込み処理 B が再び割り込み処理 A にプリエンプトされると，さらに連続するスタック領域が使用

される．そのため，割り込み処理のスタックオーバーフローが発生した場合は，どの割り込み処理がスタックオーバーフローを発生させた原因かの特定が困難である．

組込みシステムが，個々の割り込み処理のスタックオーバーフロー検出やスタックオーバーフロー発生時のスタックの再割り当て機能を備えることは，システム全体の信頼性向上につながる．

ソフトウェアから呼び出される関数と違い，割り込み処理はセンサーなどのハードウェアから割り込みによって呼び出されるため，呼び出されるタイミング，頻度などが環境の影響を受ける．そのため，スタックオーバーフローの発生に関して，すべての状況における試験を行っておくことは困難である．もし，割り込み処理が不具合を発生した場合，その原因がスタックオーバーフローであったとしても，同じ状況の再現が難しく，不具合原因の特定に時間がかかる．そこで，割り込み処理のスタックオーバーフローの原因特定が迅速に行えることも重要となる．

2.6　本研究の位置付けと目的

組込みシステムにおいては組込みリアルタイム RTOS を使用することが一般的であり，RTOS を使用した，さまざまな組込みシステムが作られ，各種研究もおこなわれてきた．

しかし，組込みシステム全体としては，30%近い製品が RTOS を使用していない[14][15][16]．RTOS を使用しない製品はシングルチップマイクロコンピュータを使用した製品と考えられ，数量も多いと考えられる．1 台の車に 100 個前後使用される ECU においても，大部分の ECU は RTOS を搭載していない．そのような車載 ECU を開発している部署においては，各種割り込みの排他制御に苦労している．

このように， RTOS を使用していない組込みシステムも RTOS を使用している組込みシステムに劣らず重要である．

RTOS を使用していない組込みシステムにおいては，並行処理はハードウェア割り込みと，割り込みにより CPU ハードウェアから呼び出される割り込み処理を用いて実現されている．

ところで，並行処理を実行する場合，リアルタイム性を向上させるためには排他制御機能が必須である[4][5][25]．RTOS を使用している場合は，RTOS が提供する機能出るセマフォを用いてリアルタイム性を損なうことがない排他制御が可能である．しかし RTOS 未使用の場合，従来はハードウェア割り

込み禁止及び許可を用いて実現していたが，この方法は前述のようにリアルタイム性を損なう．

そこで，組込みシステムを実現するうえで必須の機能である割り込み処理がリアルタイム性を損なわずに排他制御機能を実現できる方式を提案する．またCPUがハードウェアレベルの割り込み優先度を備えている場合には，それを利用して排他制御機能の性能向上の方式を提案する．さらに，MMU を使用しない組込みシステムにおいては，割り込み処理のスタックオーバーフローを検出する手段が存在しなかったため，品質の低下を招いていたが，MMU，OS を使用しない組込みシステムの割り込み処理のスタックオーバーフローを検出する手段および自動的にスタック再割り当てを行う方式を提案する．

本論文は，主に MMU を備えていないシングルチップマイクロコンピュータを用いた，RTOS 不使用の組込みシステムに対する以下の研究を行うものである．

・割り込み処理に待ち状態を持たせることによって，割り込み処理の効果的な排他制御手段を提案検証して，組込みシステムのリアルタイム性向上の研究

・CPU がハードウェア割り込み優先度を備える場合には，それを利用する方式を提案検証して，割り込み処理における排他制御機能の性能を向上の研究

・MMU，OS を使用しないで，割り込み処理のスタックオーバーフローを検出する方式を提案検証して，組込みシステムの品質向上の研究

第３章

関連研究

本章では，第 1 章で述べた対象分野，割り込み処理の排他制御，割り込み処理のスタックオーバーフローに関連する研究について述べる．

3.1　割り込み処理の制御に関する関連研究

3.1.1　割り込み処理の排他制御に関する関連研究

従来は，割り込み処理の制御(排他制御)に関する研究は現時点では存在しない．従来のアプローチが割り込み処理は割り込み禁止で必要最低限の処理のみを行い，処理の大部分はタスクで行うことを推奨してきたからと考えられる．そのためにプログラムのサイズが小さなことを目指してきた RTOS もいくつか存在する[27][28][29][30].

しかしそれらの RTOS も

・RTOS のための割り込みが必要である．

・RTOS のスケジューラも定期的なタイマー割り込みであり，RTOS 自身が必ずしも割り込みよりも優先度が高いとは限らず，効果的な割り込み制御が行えない．

・RTOS 自身の割り込みの優先度，および RTOS 内部の割り込み禁止を利用した排他制御による他の割り込みへの影響がわかりにくく，RTOS がブラックボックス化されている．

・割り込みよりも並列処理の実行単位が割り込み処理とタスクの二つが存在する二重構造となっており，エンジニアにとってシステム設計が難しくなる．

という問題がある．

本論文で提案する手法はこれらの点を解決することを目指したものである．

3.1.2　モニタに関する関連研究

従来 RTOS を使用しない組込みシステムの開発には，モニタを使用する場合もあった．モニタとは，開発時に組込みシステムに常駐させ，デバッガを行う場合に利用することが多い．

図 3-1 にモニタの構成例を示す．図に示したのは M16C のデバッガである．

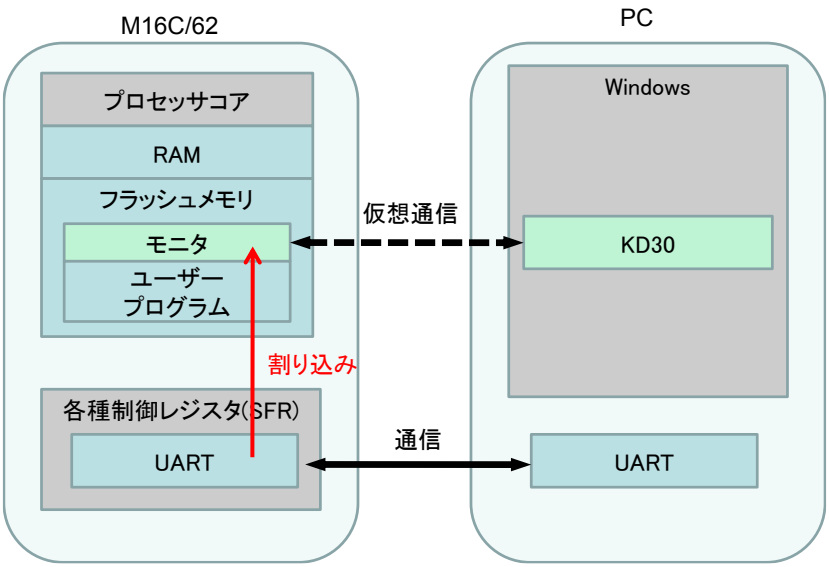


図 3-1　モニタの構成

この場合，モニタおよびユーザープログラム共に M16C 内蔵のフラッシュメモリに書き込まれている．ユーザープログラムは非割り込み環境で実行されている．

ホスト PC からの定期的な RS-232C 通信により，割り込みを発生させ，ユーザープログラムからモニタに制御を移す．

そしてモニタでそれまで実行していたユーザープログラムのレジスタやメモリの情報を表示する．さらに，ハードウェアブレークポイントを利用して，ユーザープログラムの実行を停止させてモニタに制御を移すこともできる．

しかしこのモニタはデバッグ対象が基本的に非割り込み状態で実行されるプログラムであり，割り込み処理には使用できない．

並行処理に対応する機能は持たず，セマフォなど並行処理の排他制御に使用できる機能も備えていない．

3.1.3　CPU の割り込み優先度レベルの使用に関する関連研究

従来は，CPU の割り込み優先度レベル利用に関する研究は現時点では存在しない．3.1.1 と同様従来の研究は，いかに RTOS を搭載して問題を解決するかであり，RTOS を搭載すれば必ずしも CPU の割り込み優先度を使用しなくとも，タスクの優先度で問題を解決できると考えられてきたからと思われる．しかし本論文で提案する手法のように，割り込み優先度を使用した方が，ソフトウェアの処理が不要になり，処理速度が向上する場合もある．

3.2　割り込み処理のスタックオーバーフローの検出に関する関連研究

3.2.1　スタックオーバーフローに関する関連研究概要

割り込み処理のスタックオーバーフローに関する研究も存在しない．割り込み処理に限定しない場合の組込みシステムにおけるオーバーフローの検出あるいは防止方式としては，静的なチェック方式，つまりコンパイル時にアプリケーションソフトウェアにオーバーフローをチェックするためのソフトウェアを組み込んでおく方式[31]，コンパイル時にスタック上の戻りアドレスやポインタなどをあらかじめ書き変えておく方式[32][33]，あるいは，ARM 独自のハードウェア機能である MPU（メモリプロテクションユニット）や MMU を用いた方式がある[34][35][36].

セキュリティ的なスタックオーバーフロー攻撃に対しては，メモリ内の汚染されたデータの伝搬を追跡し,悪意のあるコードが実行されているかどうかをチェックするという方式[37]やメモリオーバーフローにより，書き変えた命令の実行を検出するデバッガの方式[38]がある．

3.2.2　静的な方式

組込みシステムは通常オーバーフローを検出する仕組みやスワップ領域を備えていないため，仮想メモリを使用したメモリオーバーフローの対策が行えない．そこで，コンパイル時にメモリオーバーフローを検出するソフトウェアを，アプリケーションソフトウェアに組み込んでおき，実行時に検出する方式が Memory Overflow Protection[31]である．

他の静的なチェック方式としては，コンパイル時にスタック上の戻りアドレスをカナリーワードというマジックナンバに書き変えておき，戻りアドレスを使用する前にチェックを行う Stack-guard[32]，コンパイル時にすべての

ポインタを，バッファのメモリ領域を検査する特別なオブジェクトに書き変えておく dynamic buffer overflow detector[33] がある．

しかしこれらの静的な方式は，変数の別名やリンク情報の欠如など実行時のチェックではないため，警告が多く，さらに誤った警告も多いなどの問題があるとされている．

これらのアプリケーションソフトウェアをあらかじめ書き変えておく方式は，アプリケーションが明示的にメモリ領域を使用する場合のオーバーフロー対策を主な目的としており，アプリケーションが暗黙的に使用するスタック領域を使用する場合の対策は，考慮されていない．さらにアプリケーションのソースコードを必要とする．

3.2.3　ARM 独自のハードウェアまたは MMU を用いる方式

ARM 独自のハードウェア機能である MPU（メモリプロテクションユニット）や MMU を用いた方式である．

具体的には，タスクのスタックオーバーフロー対策として ARM の A シリーズに搭載している TrustZone を用いた方式(TOPPERS/SafeG)[34]，メモリ保護機能を持った RTOS(TOPPERS/HRP2)[35]，ページテーブル書き換え方式[36]，red zone 方式[36]，ARM プロテクトドメインを用いた方式[36]がある．

3.2.3.1　TrustZone を用いた SafeG 方式

TrustZone とは ARM の A シリーズに搭載されている機能であり，CPU に二種類の特権レベルを設け，リソースへのアクセス制限を実現するものである．特権レベル 1 では，すべてのリソースへのアクセスが可能であり，特権レベル 2 では一部のリソースへのアクセスを制限可能である．

TrustZone を用いた方式に，TOPPERS/SafeG（SafeG）がある．SafeG は，ARM 上で汎用 OS と RTOS を安全に同時実行可能なデュアル OS モニタである．RTOS はシステムすべてのリソースへのアクセス可能な Trust 領域（特権レベル 1）で動作する．

汎用 OS はリソースに対するアクセスが制限されている Non-Trust 領域（特権レベル 2）で動作する．

これにより，汎用 OS 上で動作しているプログラムによる，RTOS 用スタック領域への侵略を阻止することができる．しかし，TrustZone にはスタックオーバーフローを検出する機能は実装されていない．また TrustZone は，ARM A シリーズに固有な機能である．

3.2.3.2　メモリ保護機能を搭載した RTOS(TOPPERS/HRP2)の方式

RTOS である TOPPERS/HRP2[23]（HRP2）は TOPPERS/ASP にメモリ保護機能を拡張した RTOS であり，MMU を搭載したマイコンを対象として，ディスパッチャ内でメモリ保護領域の変更を行って，メモリ保護機能を実現している．

HRP2 とメモリ保護機能を持たない TOPPERS/ASP(以降 ASP)とのタスク切り替え時のオーバーヘッドを比較しているが，HRP2 のタスク切り替え時のオーバーヘッドは ASP に比べて最大 2 倍と報告されている．

3.2.3.3　ページテーブル書き換え方式

MMU を用いてタスク切り替え時に切り替え元のタスクスタックを無効にし，切り替え先のタスクスタックを有効にしてタスクのスタックオーバーフロー検出を行う[36].

タスク 1 を実行中はタスク 2 のスタック領域へのアクセスを行うことはできない．同様にタスク 2 を実行中はタスク 1 のスタック領域へのアクセスを行うことはできない.

この方式は，タスク切り替え時にページテーブルエントリの書き換えを行うためオーバーヘッドが大きくなる．さらにスタックオーバーフローには対応していない.

3.2.3.4　red zone 方式

ARM の MMU に搭載されている red zone 機能を用いてスタックオーバーフローの検出を行う[36].

具体的には各タスクスタック領域の終端アドレスに red zone と呼ぶアクセス禁止領域機能おく．タスク 1 実行中でもタスク 2 のタスクスタック領域へのアクセスを行うことが可能である.

この方式は，タスク切り替え時にページテーブルエントリの書き換えが不要なため，ページテーブル書き換え方式に比べてタスク切り替えを高速に行うことができるが，タスクスタック毎に 1 ページ分の領域が red zone のために余分に必要となる.

3.2.3.5　ARM プロテクトドメイン方式

ARM の MMU に搭載されているドメイン機能を用いてスタックオーバーフロー検出を行う[36].

具体的には，タスク切り替え時に有効なドメインを切り替えることでアクセスできるスタック領域の変更を行っている.

以上の 3.2.3.1 から 3.2.3.5 の方式は MMU を備えた ARM または MMU を備えた CPU でないと実現できない.

3.2.4　スタックオーバーフロー攻撃に対する方式

セキュリティ的なスタックオーバーフロー攻撃に対しては，メモリ内の汚染されたデータ（taint データ）の伝搬を追跡し,悪意のある命令が実行されているかどうかをチェックする Embedded TaintTracker[37]が報告されている.

これは，メモリ内の汚染された taint データの伝搬を追跡し,悪意のある命令が実行されているかどうかをチェックするために，従来は動的にコードを計測するエミュレータ上でプログラムを実行されていた機能を OS 上に移したものであり，MMU を使用する.

またメモリオーバーフローにより書き変えた悪意のある命令を実行させる攻撃に対する対策として，デバッガの共通プラットフォームを提供するための Stealth Breakpoints[38]が報告されている．これは書き換えられた後の悪意のある命令の実行を CPU ハードウェアが提供するシングルステップモードを利用して検出追跡するものである.

これらは，セキュリティ的な攻撃により汚染された命令の実行を監視するものであり，オーバーフロー自体を検出するものではない.

3.3　関連研究まとめ

表 3-1 にスタックオーバーフローの検出に関する提案方式と従来方式の比較を示す.

あらかじめチェックコードをアプリケーションに埋め込んでおく Memory Overflow Protection[31]，Stack-guard[32]，dynamic buffer overflow detector[33] はソースコードが必要であり，さらにアプリケーションが明示的にメモリを使用する部分を対象としており，関数呼び出しのように暗黙的に使用するスタックのオーバーフローは検出対象としていない.

OPPERS/SafeG[34]はハイブリッド OS 環境において汎用 OS 上で動作しているプログラムによる，RTOS 用スタック領域への侵略を阻止することができるが，スタックオーバーフローが発生した際の検出を行う機能は搭載して

いない．またこの方式は ARM の A シリーズに搭載している TrustZone を使用しているため，たとえ ARM であっても，他のシリーズでは利用できない．TOPPERS/HRP2[35]は ARM や SH などで利用可能であるが，MMU が必要である．

ページテーブル書き換え方式[36]は，スタック領域の完全な保護を行うことが可能であるが，MMU が必要であり，処理の切り替え時に，ページテーブルの書き換えも必要になる．

red zone[36]方式はページテーブル切り替え方式に比べて処理の切り替え処理は高速であるが，ARM の MMU の機能を使用する．

ARM のプロテクトドメイン[36]を用いた方式も MMU 機能を使用する．

Embedded TaintTracker[37]や Stealth Breakpoints[38]は書き換えられた命令の実行を検出するものであり，スタックオーバーフロー検出はできない．

これら従来の方式に対して，今回提案のマジックナンバ方式は MMU およびソースコードを使用せず，割り込み処理のスタックオーバーフローの検出を行うことを目的とする．

表 3-1　スタックオーバーフォローの検出方式比較

	スタックオーバーフローの検出	ソースコード不必要	MMU 不使用	ARM CPU 固有機能を使用しない
マジックナンバー方式	○	○	○	○
Memory Overflow Protection	×	×	○	○
Stack-guard	×	×	○	○
Dynamic Buffer Overflow Detector	×	×	○	○
TrustZone/SafeG	×	○	○	×
HPR	○	○	×	○
page table	○	○	×	○
red zone	○	○	×	×
ARM protect domain	○	○	×	×
Embedded TaintTracker	×	×	×	○
Stealth Breakpoints	×	○	○	○

第４章

割り込み処理の制御

本章では本研究で提案する割り込み処理の排他制御方式，割り込み処理のハードウェア優先度を利用した実行方式と優先度継承方式，そして割り込み処理におけるスタックオーバーフローの検出方式とスタックの再割り当て方式に関して述べる．

4.1　提案方式の概要

第 2 章で述べたように，組込みシステムにおいて割り込み処理は非常に重要である．なぜならば，割り込みは組込みシステムにおいて並行処理を実現する唯一の手段であるからである．そして並行処理を実施する場合，並行処理間の CS の排他制御機能は必須である．

割り込み処理の CS の排他制御実現の方法としては，すべての外部割り込みの禁止および許可しか存在しなかった[2][3]．しかしこの方法は，一度割り込み禁止を開始すると，新たに発生した割り込みをその優先度とは無関係にすべて保留する．その結果，優先度が高い，排他制御に無関係な割り込み，無関係の割り込み処理まで保留してしまい，システムのリアルタイム性を阻害する．特に RTOS を使用していない組込みシステムにおいては，並行性を実現する手段としては割り込み処理しか存在しないためその影響は重大である．

本論文では，特にリソース制約が厳しく，RTOS を搭載できないため割り込み処理により並行処理を実現している，シングルチップマイクロコンピュータを使用する組込みシステムを主な対象として，割り込み処理間の効率的な排他制御方式を提案する．

提案方式は，割り込み処理にも独立した実行環境を持たせることにより，割り込み処理に“待ち状態”を含む“状態”を持たせることを可能にして，RTOS がタスクに対して排他制御手段として提供しているセマフォと同等の

機能を，割り込み処理が使用できる方式である．

本方式の骨子は，従来割り込み処理では利用できなかったセマフォ同等の機能を，割り込み処理で利用可能としたことであり，多重割り込み使用時，割り込処理間の CS の排他制御にセマフォ同等の機能を利用することにより，リアルタイム性を向上させることを狙いとする．提案方式を使用すると，割り込み処理の排他制御が効率的に実現できるため，リアルタイム性の向上が期待できる[6][9][24].

提案方式は，RTOS を使用する場合と異なり，専用の割り込みを必要としない．また RTOS が提供するセマフォがタスクの切り替えが発生する場合と発生しない場合で処理時間が異なるのに対して，割り込み処理の切り替えが発生する場合も，発生しない場合も処理時間が等しいため，最悪処理時間の見積もりが必要なリアルタイム設計も容易である．

提案方式は，RISC タイプのように外部割込みに優先度レベルがない CPU に対しても適用可能である．一方 CISC タイプのように外部割込みが優先度レベルを持つ場合は，それを利用して処理速度を向上させることも可能となる．ただし処理速度を向上させたタイプは，前述は処理時間の一定性は失われるため，用途に応じて使い分けることができる．

提案方式は割り込み処理間の優先度逆転を防ぐための，優先度継承機能も備えているが，これに関しても CPU が備えている割り込み優先度を利用して処理時間の向上を図った方式を提案する[39].

さらに，その発生が現実の環境に左右され，現場での問題が大きい，割り込み処理のスタックオーバーフローの検出と，MMU を使用しないスタックの再割り当て方式に関する提案を行う[40].

4.2　割り込み処理に待ち状態を持たせてセマフォを実現する方式

4.2.1　REMON スケジューラ概要

提案方式を REMON と呼ぶ．図 4-1 に REMON 基本構造を示す.

割り込み処理ごとに状態を持たせるために，割り込み処理毎に一時停止時の各種情報を保存するためのデータ構造である ICB(Interrupt Control Block)およびスタックを持たせる.

割り込みが発生するごとに REMON スケジューラが，実行中の割り込み処理を再開させるのに必要なデータを ICB に保存しておく.

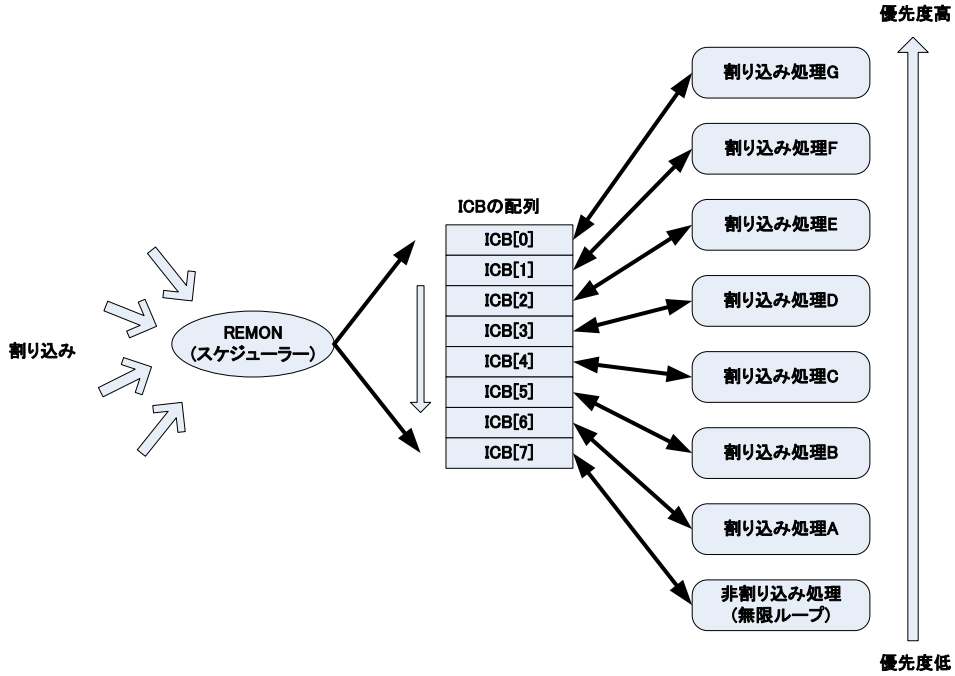


図 4-1　REMON の基本構造

割り込み処理を実行させる場合は，REMON スケジューラが ICB の配列を検索して，実行可能状態の割り込み処理のうち最高優先度の割り込み処理を実行させる．

図 4-2 に割り込み処理が持つ「状態」および状態遷移図を示す．初期状態は未実行状態である「DORMANT（停止状態）」である．この状態から割り込みが発生すると「READY（実行可能状態）」となる．実行可能状態の割り込み処理のうち最高優先度の割り込み処理が「RUNNING（実行状態）」となる．さらに割り込み処理が後述のセマフォの解放待ちになった場合は「WAITTING（待ち状態）」となる．

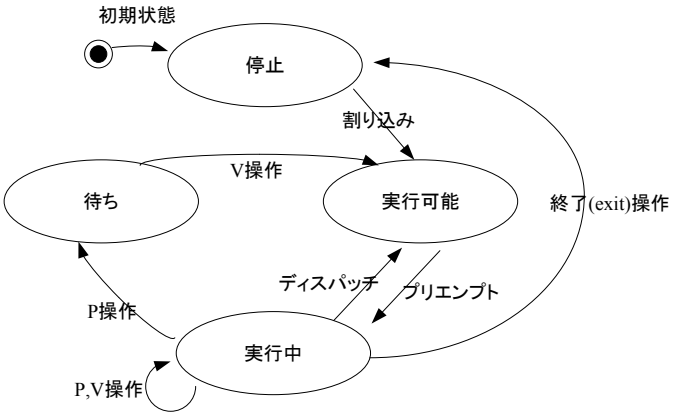


図 4-2　REMON における割り込み処理の状態遷移

ICB は配列構造であり，REMON スケジューラは割り込みが発生するたびに，その配列の先頭から実行可能状態の ICB を検索してそれに関連づけられた割り込み処理を起動，または再開させる．つまり配列内で先頭に近い位置におかれた ICB に関連付けられた割り込み処理ほど優先度が高くなる．

これを利用して RISC など割り込みに優先度がない CPU においても，割り込み処理に優先度を付加することができる．

この ICB の配列のインデック番号を割り込み処理の番号として使用する．実行中の割り込み処理は，より優先度が高い割り込みが発生するとプリエンプトされ実行可能状態となる．

実行中の割り込み処理は後述のセマフォ獲得要求の P 操作によってセマフォを獲得できなかった場合は待ち状態となる．待ち状態の割り込み処理は実行中の割り込み処理の後述のセマフォ解放の V 操作によって再び実行可能状態となる．なお，図には実行中の割り込み処理の P 操作や V 操作によっても状態が変化しない場合があるが，これは P 操作や V 操作にも関わらず処理の切り替えが発生しなかった場合を示している．

組込みシステムにおいては，割り込みの数が少ない場合，割り込み制御用のデバイスである割り込みコントローラを増設して，割り込みの数を増加させる[2][40].

しかし，REMON の主な対象としている，シングルチップマイクロコンピュータを使用するような，比較的小規模なシステムにおいては，ノイズ対策，サイズ，価格などの理由で，別途割り込みコントローラを増設することが難しく，割り込みも貴重な資源となる．RTOS を使用する場合は，タスクから RTOS に定期的に制御を移すための，RTOS 専用の割り込みや，RTOS 内部の排他制御のための割り込み禁止が必要であるが，REMON においては，それらは不要である．

4.2.2　ICB 詳細説明

図 4-3 に割り込み処理に状態を含む，独立した実行環境を持たせるために必要な ICB の構造を示す．

各割り込み処理に関連づけられた ICB は，一時停止後の再開に必要なデータ領域であり，以下のデータ領域を備えている．

・割り込み処理の状態
・割り込み要求回数
・割り込み処理開始アドレス
・割り込み処理のスタックの初期アドレス(最高位アドレス)
・CPU コンテキストの保存領域

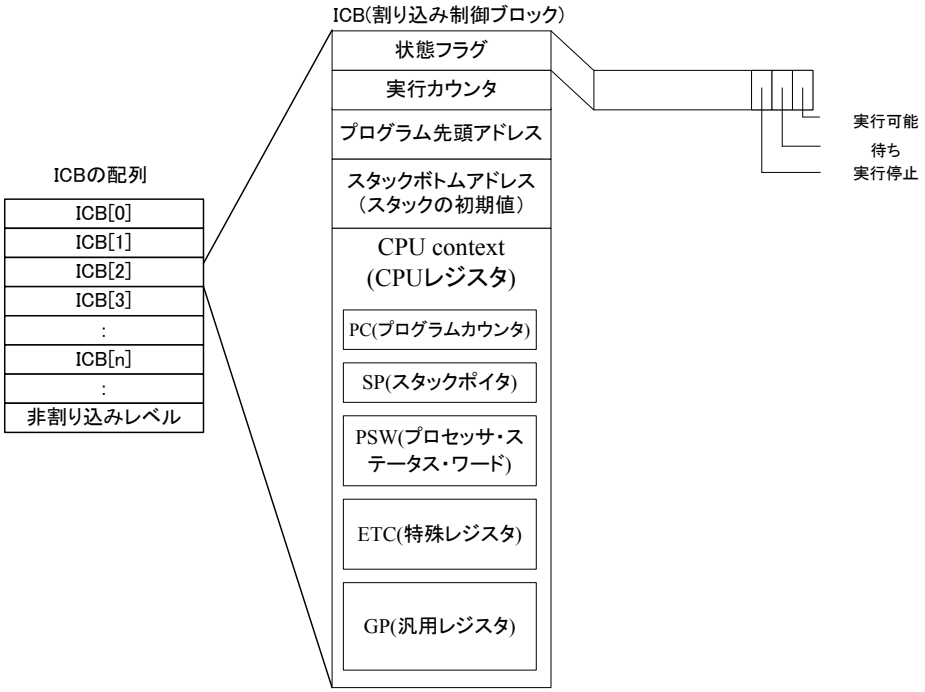


図 4-3　ICB の基本構造

割り込み処理が実行中に一時停止する時は，ICB にその時のレジスタの内容など CPU コンテキストを保存することによって，のちに続きを再開できる．割り込み処理ごとに ICB を持つことにより，割り込み処理ごとに状態を持つことができる．割り込み処理が“待ち状態”を含む状態を持つことで割り込み処理もタスクが使用するセマフォと同等の機能が使用できるようになる．スタックに関しても個々の割り込み処理に割り当てておく．

割り込み処理状態領域には 4.1.1 で説明した，各々の割り込み処理の状態を保存する．

実行カウンタとは，後述の割り込み発生回数を保存する．割り込み禁止状態で割り込み処理実行中に，同じ割り込みが連続して発生した場合 CPU ハードウェアは 1 回だけ割り込みが発生したことを記録できるため，実行中の割り込み処理終了時に，CPU ハードウェアが続けて同じ割り込み処理を実行させることができる．しかし割り込み禁止状態で割り込み処理実行中に同じ割り込みが 2 回以上発生しても，CPU ハードウェアが記録できるのは 1 回分のみであるため，結果的に 2 回目以降の割り込みが無視されることになる．それに対して，提案の REMON では割り込み禁止時間がきわめて短いため，割り込みを無視する時間も短く，2 回以上の割り込みの発生に対応できる場合が多い．

ICB のプログラムの先頭アドレス領域には，各割り込み処理の先頭アドレ

スを設定しておく．割り込みが発生して，新たに割り込み処理を実行する場合はこのアドレスから処理を開始する．

実行中に一時停止した時に，再開するアドレスは ICB の CPU コンテキスト領域にある，プログラムカウンタレジスタの保存領域に保存される．

ICB の CPU コンテキスト保存領域は，CPU のプログラムカウンタ(PC)，スタックポインタ(SP)，CPU の状態を保持するプロセッサステータスワード(PSW)，汎用レジスタなどを保存する領域である．

PSW にはすべての外部割込みの禁止／許可，優先度別の割り込みの禁止／許可を設定するビットなどがあり，ソフトウェアでも変更できる．

割り込み処理実行中に割り込みが発生して REMON スケジューラに処理が切り替わると，その時点の CPU コンテキストを ICB に保存して，さらにそのフラグを待ち状態変更する．この割り込み処理を再開させる場合は，ICB に保存したデータを復元する．スタックのボトムアドレスはスタックの初期値（最高位アドレス）が格納される領域である．

新たな割り込みが発生して，割り込み処理を最初から実行させる場合，その割り込み処理専用のスタック領域の初期アドレスとしてこの値をスタックポインタレジスタに REMON が設定する．

割り込み処理が実行中に一時停止させられる場合，スタックポインタレジスタの値は CPU コンテキス領域に保存され，再開される場合はそちらを使用する．再度この割り込み処理を最初から実行させる場合は，このスタックのボトムアドレスが使用される．

4.2.3　REMON セマフォ概要

図 4-4 に割り込み処理がセマフォを使用できるために必要なセマフォ制御構造体 SCB(Semaphore Control Block)を示す．

セマフォ制御構造体はセマフォの値を保持する領域，セマフォを所有（ロック）している割り込み処理番号，セマフォ解放待ちの割り込み処理番号を格納する領域を備える．

セマフォの値とは 0 以上の整数であるが，現状は単純な構造とするためにバイナリセマフォのみを実現しており 0 か 1 である．

また同じく単純な構造とするためセマフォ解放待ちの割り込み処理は最大 1 個までと制限している．

複数の割り込み処理がセマフォの解放を待てるようにするには，優先度順の待ちであればセマフォ解放待ちの領域をビットマップとすればよい．

もし FIFO 待ちを実現するのであれば，この部分をリスト構造にすればよいが，通常はそこまで複雑にする必要はないと判断している．

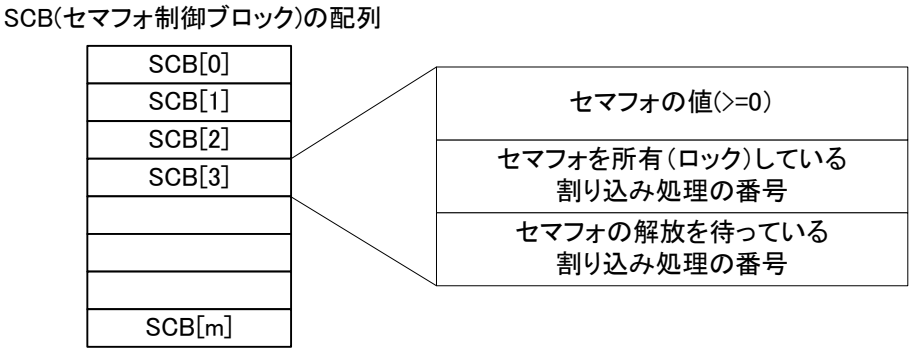


図 4-4　セマフォ構造体

このセマフォ構造体も配列になっており，配列のインデックス番号をセマフォ番号として使用する．実行状態の割り込み処理は，セマフォ番号で対象のセマフォを指定して P 操作または V 操作を行うことができる．

P 操作実行時，セマフォの値が 1 であればその値を 0 にしてそのまま実行を続ける．その値が 0 であれば，P 操作を実行した割り込み処理は実行状態から待ち状態になる．

V 操作実行時，対象のセマフォの解放待ち割り込み処理がない場合は，セマフォの値を 1 にしてそのまま実行を続ける．解放待ち割り込み処理が存在する場合は，対象の割り込み処理を待ち状態から実行可能状態に変更して，あらためて REMON スケジューラを呼び出す．

4.2.4　REMON の必要メモリ量

従来方式でも，通常は多重割り込みを許可しているため，割り込み処理実行中の，より優先度の高い割り込みの発生により，新たな割り込み処理が実行される．さらに，その割り込み処理終了後には，割り込まれた割り込み処理の続きの命令から，実行を再開させる．

そのため，割り込みスタックなどに CPU のレジスタの内容などの，実行を再開するための CPU コンテキストなどのデータを保存しておく．

状態フラグ，実行カウンタなど一部のデータは増加するが，REMON で ICB に保存される情報と，このスタックに保存される情報は，ほとんど同じ内容である．そのため，保存場所をスタックから ICB に変更しても，必要なメモリ容量，保存時間が大きく増加するわけではない．また ICB の配列によって割り込み処理の優先度を実現できる優先度方式は，割り込みにハードウェア的に優先度がない CPU に，優先度を与える場合にも適用できる[24][39][41].

さらに従来方式では割り込み処理は各々のローカルメモリとして一つのス

タックを共有しており，多重割り込み処理実行時，スタックの容量は各割り込み処理が使用する最大サイズの合計分が必要となる．

それに対して REMON では割り込みの種類ごとにスタック領域を，割り当てておくが，必要なスタックの合計容量は，ほぼ同じとなる．なお，REMON は，現在割り込み処理のみによって実現しているシステムに対する，割り込み処理間の新しい排他制御方式による，リアルタイム性の向上を狙いとしている．そのため一つの割り込みレベルに対する割り込み処理は一つとしており，全体としてみると，メモリ容量の増加は，ICB の一部増加分とセマフォ構造体などによる増加のみと考えられる．

4.2.5　REMON スケジューラの動作

4.2.5.1　動作を示す記号の意味

まず，アルゴリズムの動作を説明する図で使用する記号の意味を説明する．

 は関数，サブルーチンなど何らかの処理の入り口を示す．

 は何らかの処理を示す．

 は判定を示す，具体的には C 言語や C++言語では if 文や switch 文などである．

 は処理の繰り返しを示す．

 は関数の呼び出しを示す．

 は処理の区切りを示す，具体的には C 言語や C++言語では }である．

 はジャンプを示す，具体的には C 言語や C++言語では goto 文などである．

 は処理の終了を示す，具体的には C 言語や C++言語では } や return 文などである．

処理の流れは 線で繋がれている．

4.2.5.2　割り込み発生時の共通処理

図 4-5(a)にハードウェア割り込み発生時に最初に実行される割り込み共通処理を図 4-5 (b)に REMON スケジューラの動作を示す.

REMON では非割り込み環境で実行する処理を無限ループの構造として，さらにその処理を割り込み処理の最も優先度の低い処理として扱う.

また実行中の割り込み処理番号は整数型の変数 runningISR#に格納して，割り込み処理を実行させる時には，その変数に割り込み処理番号を保存する．外部割り込みが発生すると図 4-5(a)に示す REMON 割り込み共通処理部が呼び出される.

共通処理部では割り込み発生時に実行されていた割り込み処理の番号を runningISR#から知り，ハードウェアにより自動的に保存されるものも含めて CPU コンテキスをその割り込み処理に対応 ICB に保存する．そして発生した割り込みから，実行する割り込み処理を決定する.

そして，対応する ICB の状態を実行可能状態にして，実行カウンタを 1 増やす.

REMON は割り込み禁止区間が短いため，割り込みを受け付けられない時間も短く，発生した割り込みを取り逃す可能性は少ない．その後，図 4-5(b)に示す REMON スケジューラに制御を移す.

REMON スケジューラは実行カウンタの回数だけ割り込み処理を実行するので，割り込みが発生した回数，割り込み処理を実行できる.

スケジューラは配列の先頭から実行可能状態の ICB を探索し，対応する割り込み処理を実行する．ICB のうち特に PC と PSW をスタックにコピーしているのは，CPU の割り込みからの復帰命令は，1 命令でこの 2 つの値を復元できるからである.

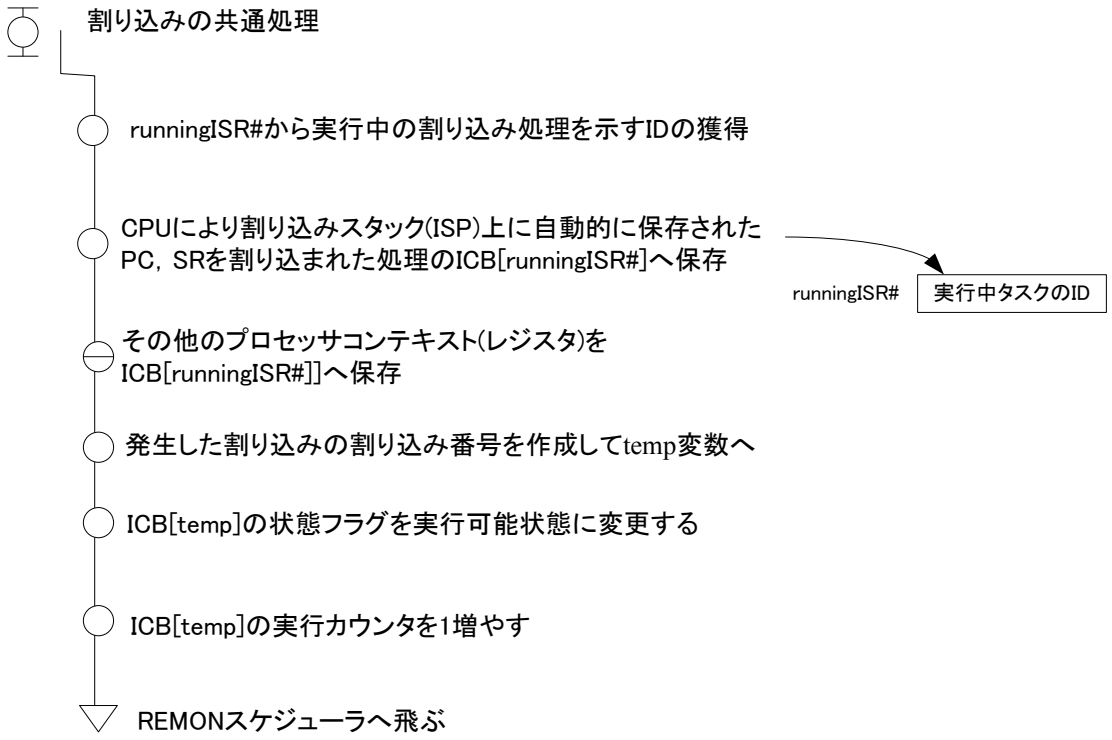
EMON スケジューラが，スタック上にコピーした PSW のすべての割り込みを許可して，割り込みレベルマスクもすべての割り込みを許可する最低レベルに下げてから，割り込みからの復帰命令を呼び出すことにより，実行される割り込み処理はすべての割り込みを受け付ける状態で実行される.

4.2.5.3　割り込み処理の終了処理

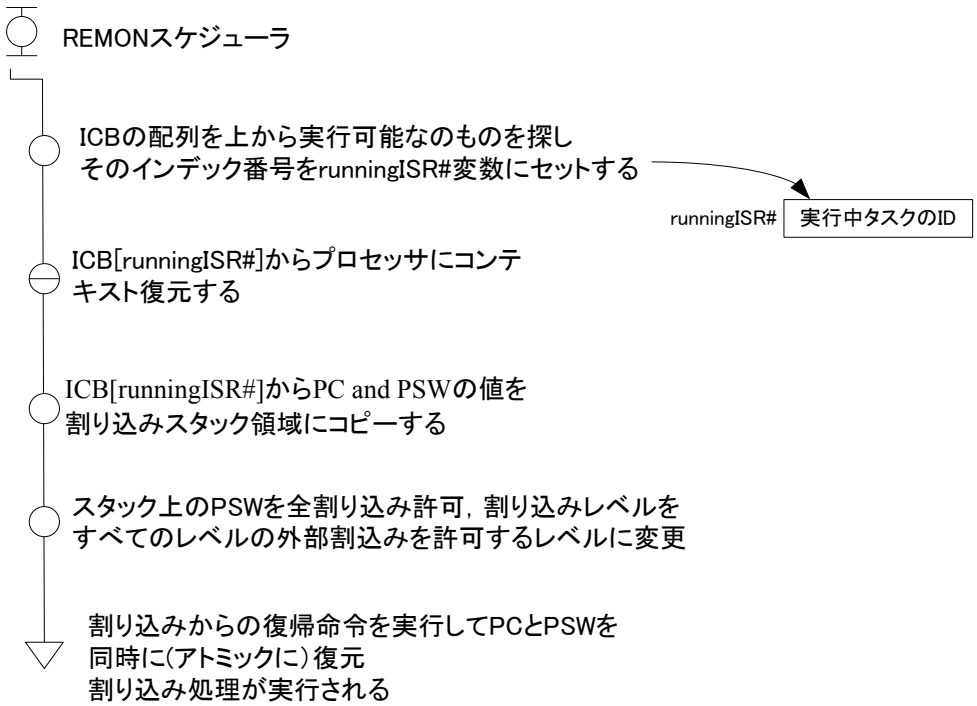
図 4-6 に割り込み処理終了時に呼び出す exit 処理の動作を示す．割り込み処理はその終了時に exit 処理を呼び出す．ここでは ICB の呼び出しカウンタの処理，状態フラグや先頭アドレスの初期化などを行う．組込みシステムにおいては，ハードウェアも新規開発する場合が多く，ハードウェアの不具合などにより，不正な割り込みが，異常な頻度で発生して，システムが制御不

能になる場合がある．

実行カウンタを利用して，異常な頻度の割り込みが発生しても，その割り込み処理の実行回数の上限を設定しておけば，それを超える割り込み実行要求がある場合は，システムエラーとして検知する，あるいは無視するなどに利用できる．呼び出しカウンタを 1 減らしても，1 以上であれば実行した回数以上に呼び出されたことを示している．この場合割り込み処理は実行可能状態のままとする．この割り込み処理は再びスケジューラが呼び出して実行させる．



(a) 割り込み共通処理



(b) スケジューラ

図 4-5　割り込み共通処理

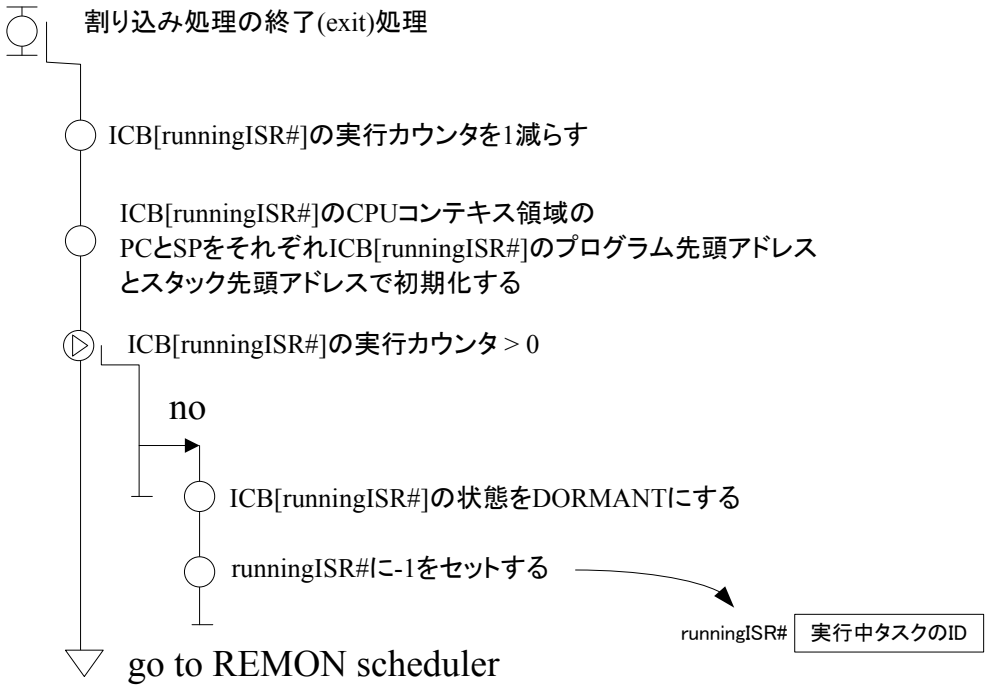


図 4-6　割り込み処理の終了処理

4.2.5.4　REMON セマフォに対する操作

REMON では割り込み処理間の排他制御を実現するために，RTOS がタスクに対して提供するセマフォと同等の機能を，割り込み処理が使用して排他制御を行うことを提案する.

以下割り込み処理が使用できるセマフォと同等の排他制御に用いる仕組みを REMON セマフォ（RTOS がタスクに対して提供しているセマフォと混同しない文中においては単にセマフォ）と呼ぶ.

REMON セマフォに対する操作としては,

・セマフォがロック（所有）されていない場合は，セマフォの値を 0 にしてそのまま実行を続け，セマフォがロックされている場合は，セマフォが解放されるまで待つ P 操作

・セマフォ解放を待っている割り込み処理がいる場合は，その割り込み処理を実行可能状態にして，セマフォ解放を待っている割り込み処理がいない場合はセマフォの値を 1 にして，そのまま処理を続ける V 操作

を提案する.

さらに P 操作， V 操作各々に対して

・割り込み処理の切り替えの有無に関わらず処理時間が一定のセマフォ操作

・割り込み処理の切り替えがない場合の処理時間を高速にして，平均処理速度を向上させたセマフォ操作

を提案する．

P 操作において，割り込み処理の切り替えが発生する場合とは，実行中の最高優先度の割り込み処理が P 操作を実行した時，対象のセマフォがすでに他の割り込み処理に所有（ロック）されており，自分自身が実行状態から．待ち状態に移行され，次の優先度の割り込み処理に切替わる場合のことである．

セマフォが他の割り込み処理に所有されていない場合は，P 操作を呼び出した割り込みそのまま処理を続行する．

V 操作において割り込み処理の切り替えが発生する場合とは，V 操作によって待ち状態から実行可能状態に移行した割り込み処理の優先度が，V 操作を実行した割り込み処理の優先度よりも高い場合である．この時は実行状態の割り込み処理の切り替えが発生する．一方切り替えが発生しない場合とはセマフォの解放待ちの割り込み処理がない場合である．なお実行可能になった割り込み処理の優先度が低い場合も，処理の切り替えは発生しないが，処理時間は切り替えが発生する場合と同じである．

割り込み処理の切り替えの有無に関わらず処理時間が一定のセマフォ操作は最悪実行時間の見積もりが重要なリアルタイムの保証が重要な場合に適している．

一方厳密なリアルタイム性よりも，平均処理時間が高速であることが重要な組込みシステムにとっては，平均処理時間が短いことが重要な場合に適している．

以下に，4.2.5.5 に 処理時間が一定の P 操作

4.2.5.6 に 平均処理時間が短い P 操作

4.2.5.7 に 処理時間が一定の V 操作

4.2.5.8 に平均処理時間が短い V 操作

の提案方式を説明する．

4.2.5.5　処理時間が一定の P 操作

図 4-7 に REMON セマフォに対する処理時間が一定の P 操作の処理を示す．この処理は割り込み処理の切り替えの有無に関わらず実行時間の変化がないようにしたものである．

CS の命令を実行する前に，割り込み処理（アプリケーション）はセマフォに対して P 操作を実行する．P 操作はソフトウェア割り込みによって，REMON の P 操作の実態（systemP）を呼び出す．ソフトウェア割り込みは

NMI(Non Maskable Interrupt – マスク不能割り込み)であるため，全割り込みを禁止している割り込み処理からも実行可能である．P 操作の実態部では最初に P 操作を実行した割り込み処理の ICB にデータを保存する．その後，セマフォの値によって処理を切り替える．

セマフォの値が 1 の場合は，その値を 0 にした後，セマフォ構造体のセマフォを所有（ロック）している割り込み処理の番号領域に P 操作を実行した割り込み処理の番号を格納する．

セマフォ構造体のセマフォ解放を待っている割り込み処理番号領域は，待っている割り込み処理がないことを示すためにマイナスの値を設定する．

セマフォの値が 0 ということは，そのセマフォが他の割り込み処理によって既に所有（ロック）されていることを示すため，実行中の P 操作を実行していた割り込み処理に対応する ICB の状態を待ち状態に変更するとともに，セマフォ構造体のセマフォ解放を待っている割り込み処理番号領域に割り込み処理番号をコピーする．

セマフォの値に関わらず，上記処理終了後は REMON スケジューラに飛び，実行可能状態の割り込み処理のうち優先度が最高の割り込み処理が実行される．

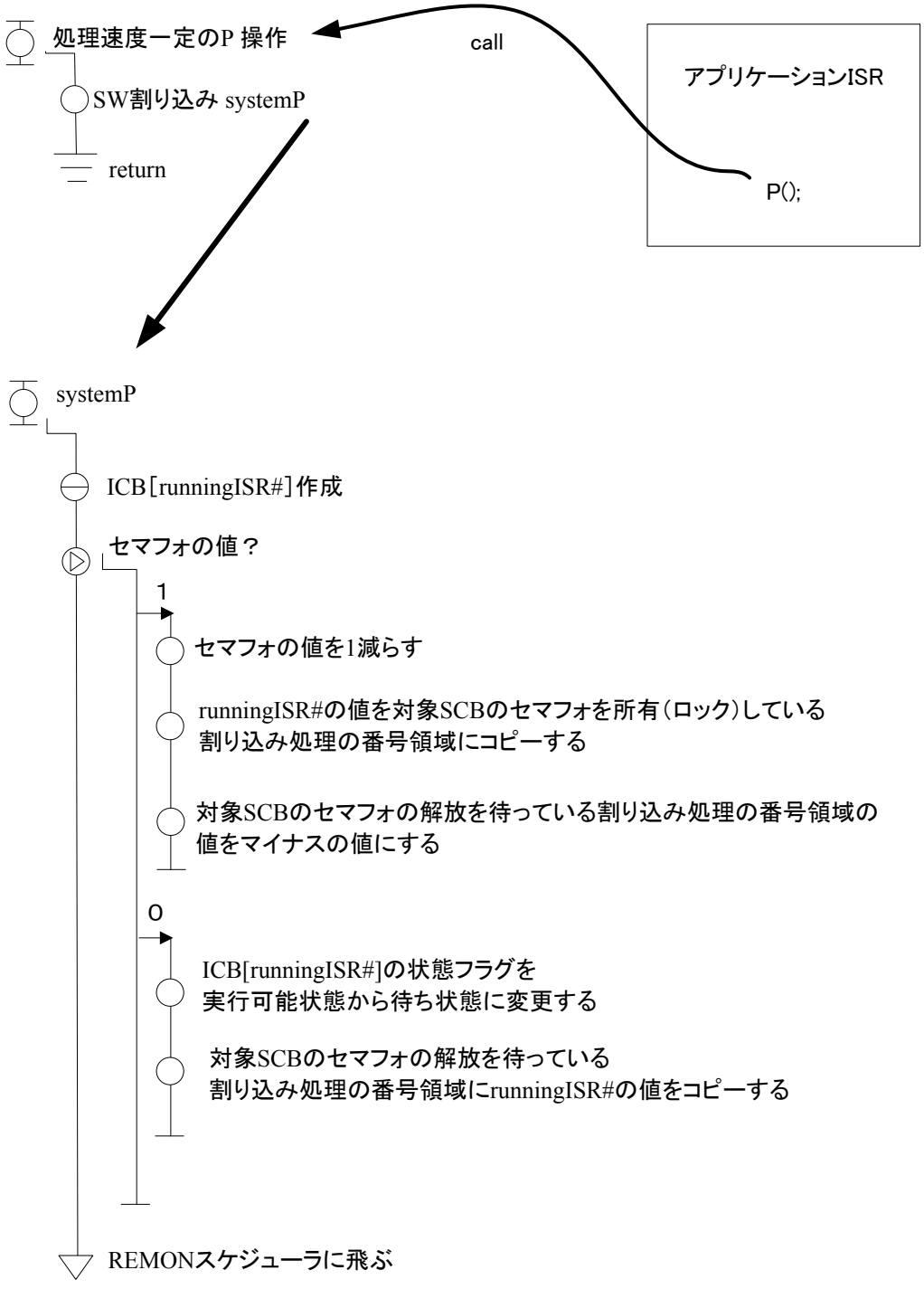


図 4-7　処理時間一定の P 操作の処理

4.2.5.6　平均処理時間が短い P 操作

処理の切り替えがない場合の処理時間を短縮して，平均実行時間を短縮した P 操作の処理を図 4-8 に示す．この P 操作は平均処理時間が短いことが重要な場合に適している．図においてセマフォの値が 1 の時は，その値を 0 にしてそのまま処理を続行する．

一方セマフォの値が 0 の場合は，そのセマフォはすでに他の割り込み処理に所有されているということである．P 操作を発行した割り込み処理の ICB の状態フラグを実行可能状態から待ち状態としたのち，ソフトウェア割り込み命令によりスケジューラに制御を移す．

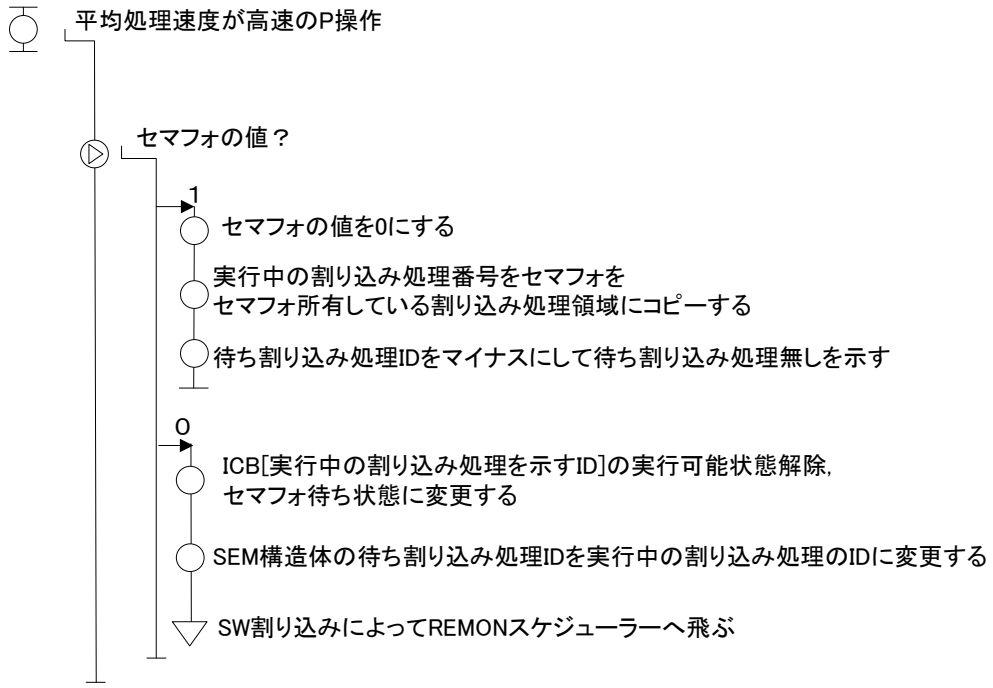


図 4-8　平均処理時間が高速の P 操作の処理

4.2.5.7　処理時間が一定の V 操作

図 4-9 に REMON セマフォに対する V 操作の処理を示す．これも P 操作同様に割り込み処理の切り替えの有無に関わらず処理時間を一定としたものである．CS の命令終了した時，割り込み処理（アプリケーション）はセマフォに対して V 操作を実行する．

V 操作はソフトウェア割り込みによって，REMON の P 操作の実態（systemV）を呼び出す．V 操作の実態部では最初に V 操作を実行した割り込み処理の ICB を作成する．その後，セマフォ解放待ちの割り込み処理の有無によって処理を切り替える．

セマフォ解放待ちの割り込み処理がない場合は，そのセマフォの値を 1 にした後，セマフォ構造体のセマフォを所有（ロック）して割り込み処理の番号領域にマイナスの値を設定して，このセマフォを所有している割り込み処理がないことを示す．

セマフォ解放待ちの割り込み処理がある場合は，セマフォの解放を待っている割り込み処理番号の ICB の状態フラグを，待ち状態から実行可能状態に変更する．

そして，セマフォ構造体のセマフォ解放を待っている割り込み処理番号領域の値を，セマフォを所有している割り込み処理番号領域にコピーする．

セマフォ構造体のセマフォ解放を待っている割り込み処理番号領域は，待っている割り込み処理がないことを示すためにマイナスの値を設定する．

最後にセマフォの値に関わらず，上記処理終了後は REMON スケジューラに飛び，実行可能状態の割り込み処理のうち優先度が最高の割り込み処理が実行される．

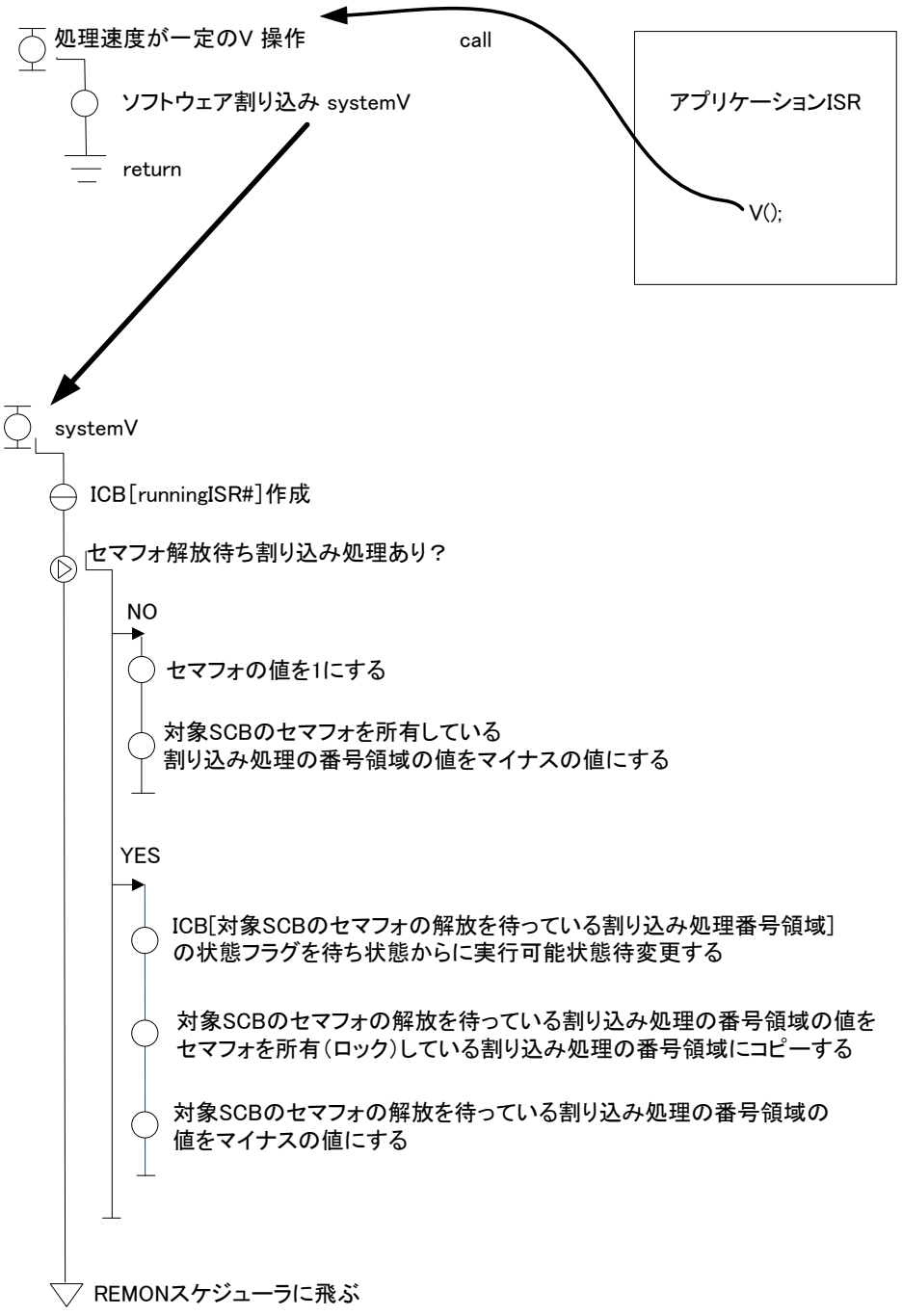


図 4-9　処理時間一定の V 操作の処理

4.2.5.8　平均処理時間が高速のセマフォに対する V 操作

処理の切り替えがない場合の処理時間を短縮して，平均実行時間を短縮した V 操作の処理を図 4-10 に示す．この方式の V 操作は平均処理時間が短いことが重要な場合に適している．

V 処理もセマフォ解放待ちの割り込み処理の有無によって処理を切り替える．セマフォ解放待ちの割り込み処理がない場合は，そのセマフォの値を 1 にした後，セマフォ構造体のセマフォを所有（ロック）したのち，割り込み処理の番号領域にマイナスの値を設定しこのセマフォを所有している割り込み処理がないことを示す．そして，最後に呼び出し元に復帰する．

セマフォ解放待ちの割り込み処理がある場合は，セマフォの解放を待っている割り込み処理番号の ICB の状態フラグを待ち状態から実行可能状態に変更する．そして，セマフォ構造体のセマフォ解放を待っている割り込み処理番号領域の値を，セマフォを所有している割り込み処理番号領域にコピーする．セマフォ構造体のセマフォ解放を待っている割り込み処理番号領域は，待っている割り込み処理がないことを示すためにマイナスの値を設定する．

最後にソフトウェア割り込み命令によりスケジューラに制御を移す．

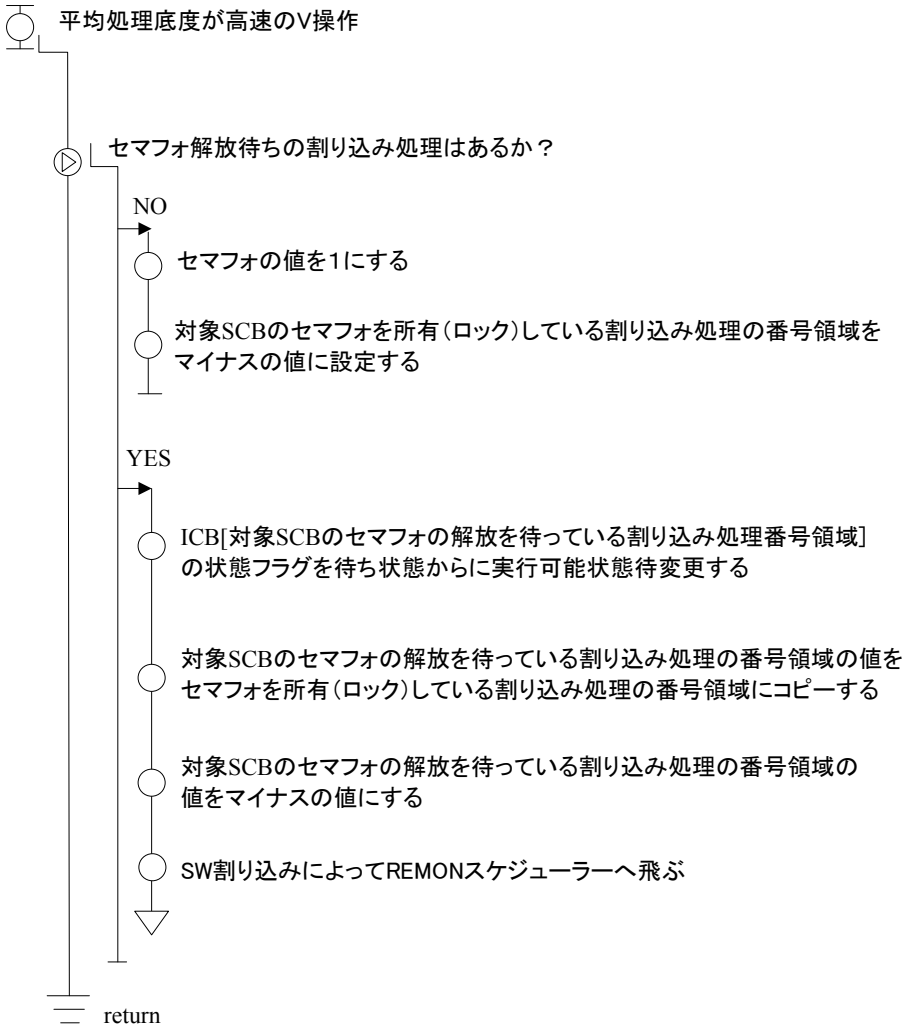


図 4-10　平均処理時間が高速の V 操作の処理

4.2.6　REMON 全体構造

図 4-11 に REMON 全体の構造を示す．割り込み発生時や，P 操作や V 操作から呼び出されると REMON スケジューラが ICB の配列の中から実行可能状態の割り込み処理を探す．この時常に配列の先頭から探すために，配列番号が小さなものほど優先度が高いことになる．また ICB 配列の最後には，非割り込み状態で実行される処理に対応しているが，この処理は無限ループの構造としておくことにより，実行可能状態の処理が少なくとも 1 つは存在することになる．

割り込み処理は REMON セマフォを使用して，排他制御を行う．

今回はシステム全体の構造も単純化することによって，品質の良い組込みシステムを構築することを目指すために，セマフォは単純なバイナリセマフォとすることを提案した．またセマフォの解放を待てる割り込み処理は最大一つとする．この制約を除くのは SCB の所有および待ち割り込み処理番号を示す部分を番号でなく，ビットマップにすればよい．

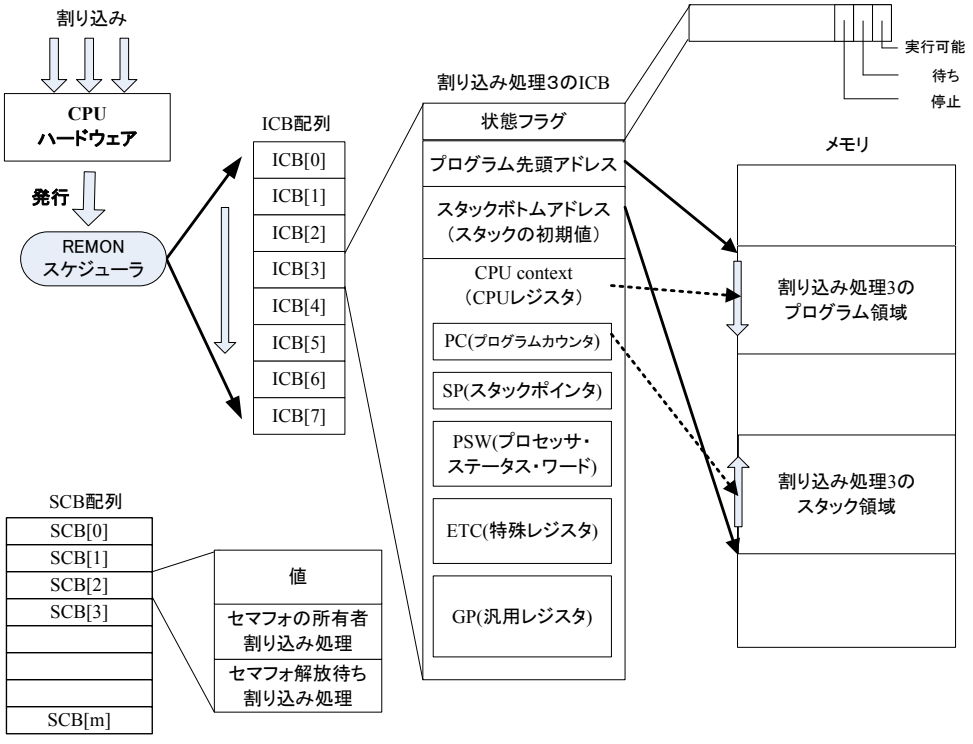


図 4-11　REMON の全体構造

4.2.7　REMONによる排他制御の動作

図 4-12 に REMON セマフォを用いて割り込み処理間で排他制御を行った場合の動作を示す．図における割り込み発生のタイミングや，割り込み処理の処理時間図 2-7 に示したものと同じである．斜線部は P 操作と V 操作によって囲まれた，排他制御区間を示す．

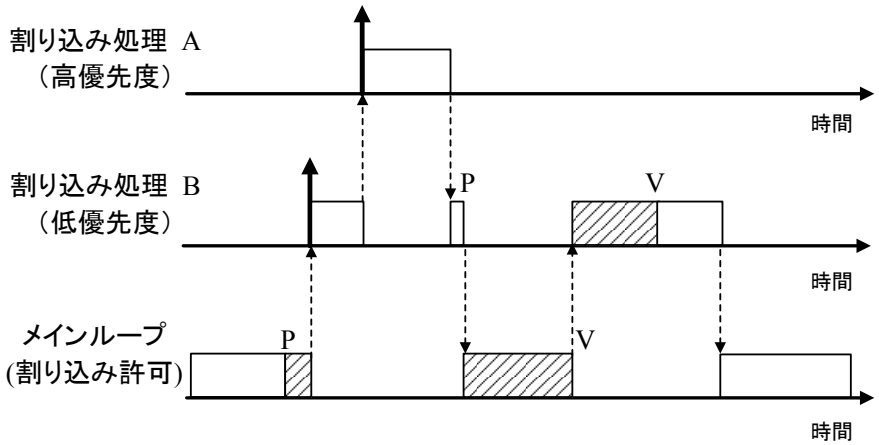


図 4-12　REMON セマフォによる排他制御動作

メインループ実行時 P 操作を実行して CS の実行を開始する．その最中に低優先度の割り込みが発生すると直ちに低優先度の割り込み処理 B が実行を開始する．そして割り込み処理 B 実行中に，高優先度の割り込みが発生すると，直ちに高優先度の割り込み処理 A が実行を開始する．

割り込み処理 A は他の割り込み処理間の排他制御の影響を受けない．割り込み処理 A の終了処理呼び出しにより，割り込み処理 A は終了する．それに伴って，REMON は割り込み処理 B の実行を再開させる，割り込み処理 B は CS を実行する前に P 操作を呼ぶ出すことによって，割り込み処理 B は待ち状態となり，メインループに制御が移る．

メインループが CS の実行を終え V 操作を呼び出すと，割り込み処理 B が実行可能状態に復帰して，メインループをプリエンプトして実行を再開する．割り込み処理 B の CS 実行終了時は V 操作を呼び出すが，セマフォ解放待ちの処理はないため，セマフォの値を 1 にしたのち，処理の切り替えは発生せずそのまま最後まで実行を続け終了処理の呼び出しにより，メインループの実行が再開する．

図 2-7 で示した，DI/EI による排他制御と比較すると，REMON セマフォによる排他制御では，優先度の高い割り込み処理 A の終了が，図 3 の割り込み処理 A の終了より，早くなっている．REMON セマフォの利用により，CS の

排他制御が実現でき，リアルタイム性が向上する．

4.3　割り込み優先度レベルを利用した REMON の高速化

割り込み処理は独立した実行環境を持たず，セマフォを使用することができなかったため，割り込み処理間の排他制御には CPU が備える外部割込みの割り込み禁止（DI）命令と許可（EI）命令が使用されてきた．しかし DI は，すべての外部割込みを禁止するため，影響はシステム全体に及ぶ．つまり，DI/EI による排他制御は，無関係の割り込み処理の実行を遅らせ，組込みシステムのリアルタイム性を阻害するという問題があった[3][4].

この問題は，割り込み処理に対して RTOS のセマフォと同等の機能を提供する割り込みスケジューラ REMON(Real-Time Embedded Monitor)によって解決された[6][24][40][41]．REMON を使用した割り込み処理間の排他制御は，組込みシステムのリアルタイム性を損なわない．さらに，4.1 節で説明した REMON は割り込み処理優先度の判定など割り込み処理に関するすべての制御をソフトウェアで行っている．この方式はハードウェア的な割り込み優先度を持たない RISC 系の CPU にも適用可能であり，汎用性は高い．

割り込みの優先度に関わらず，すべての割り込みを許可した状態で割り込み処理を実行させるために，割り込みを禁止する時間が短い．それを利用して同じレベルの割り込みが短時間に複数回発生した時も，割り込み発生回数を記録しておくため，割り込みを取り逃がす可能性が少なく，また，処理の切り替えの有無に関わらず処理時間を一定にする方式が可能である．しかし 4.1 節でも述べたように，適用分野によっては，常に処理時間が一定であることよりも，平均処理時間が高速であるほうが望ましいカーナビゲーションシステムのような分野も存在する．そのような分野の組込みシステムにおいて，CISC 系の CPU のようにハードウェア的に割り込み優先度を備える CPU を使用している場合，ハードウェア割り込み優先度を利用すれば，REMON は実行させる割り込み処理を選択するスケジューラの処理時間が短縮され，性能の向上が可能である．

そこで RTOS 不使用の組込みシステムにおいて，ハードウェア的に割り込み優先度を備える CPU の使用を前提とした，割り込み優先度を利用して平均処理速度を向上させた REMON を提案する．

この方式は，新たな割り込みが発生した時，実行させる割り込み処理の選択時に CPU ハードウェアが備える割り込み優先度を利用する．

本提案方式では，REMON は割り込み処理ごとに，割り込み処理を実行させる時のハードウェアレベルの割り込み優先度を設定しておく．割り込み処理を実行させる時，REMON は CPU が備えるプロセッサステータスワード（PSW）の割り込みレベルを設定する領域に上記の値を設定してから割り込み処理に制御を移す.

発生した割り込みの優先度レベルとは独立して，REMON が割り込み処理を実行させる時の割り込み優先度を設定できることが重要である．この方式により，REMON は高優先度の割り込み処理がセマフォ待ちになった時に，そのセマフォをロックしている低優先度の割り込み処理を実行させることができる.

新たな割り込みが発生した時，CPU ハードウェアは，PSW に設定された割り込みレベルと，発生した割り込みのレベルの比較を行う．そして発生した割り込みのレベルが，PSW に設定された割り込みレベル以下の場合，CPU ハードウェアは，発生した割り込みを自動的にマスクして，REMON の呼び出しを行わない．その結果 REMON の処理時間は 0 となり，REMON の平均処理時間が短縮される[39].

さらに 2.4 節で説明した割り込み処理セマフォの優先度逆転の問題は，セマフォをロックしている割り込み処理に，セマフォの解放を待っている割り込み処理の優先度を引き継がせて実行させる優先度継承機能を割り込み処理セマフォが備えれば解決可能である.

優先度継承セマフォにより，高優先度の割り込み処理と低優先度の割り込み処理の排他制御を実行中に，排他制御に無関係の中間優先度の割り込み処理が実行されるため，高優先度の割り込み処理の実行が間接的に阻害されるという優先度逆転の問題が解決される[42][43].

RTOS がタスクに対して提供する優先度継承セマフォと同等機能の，ハードウェア割り込み優先度を利用した，REMON 優先度継承セマフォを提案する.

REMON 優先度継承セマフォを使用すれば，割り込み処理は割り込み処理間の優先度逆転を解決することが可能になる.

CISC 系の CPU はハードウェア割り込み優先度を備えており，それを利用すれば REMON において，優先度継承機能を備えた割り込み処理用セマフォを効率的に実現できる.

ハードウェア割り込み優先度の利用により REMON の割り込み回数の記録機能や，処理速度の一定性は失われるが，カーナビゲーションシステムのように，状況によらず処理時間が一定であることよりも，平均処理時間が短い方が適している組込みシステムも存在するため，適用分野によって REMON

を使い分けることが可能となる．

提案方式では，割り込み処理セマフォを利用して排他制御を行う時，最初に割り込み処理セマフォが他の割り込み処理にロックされているかどうか検査する．そして割り込み処理セマフォがロックされている場合には，ロックしている割り込み処理の優先度と現在の割り込み処理の優先度を比較する．

割り込み処理セマフォをロックしている割り込み処理の優先度が，現在割り込み処理の優先度より低い場合は，実行中の高い優先度のままで，割り込み処理セマフォをロックしたままプリエンプトされた低優先度の割り込み処理の実行を再開させる．

優先度を継承した割り込み処理実行中は，中間優先の割り込みが発生しても，CPU ハードウェアが自動的に割り込みをマスクするため，優先度逆転は発生しない．優先度を継承した割り込み処理は排他制御区間が終了して，割り込み処理セマフォを解放するとともに，本来の優先度にもどり，高優先度の割り込み処理に再びプリエンプトされ，高優先度の割り込み処理が実行を再開する．

ハードウェア割り込み優先度を利用した，割り込み処理用の優先度継承セマフォにより，排他制御実行時の高優先度の割り込みに対する応答性が向上し，組込みシステムのリアルタイム性が向上する．

本方式の狙いは，CPU ハードウェア優先度を利用して，低優先度の割り込み発生時の REMON の処理をなくし，割り込み処理セマフォの優先度継承機能をソフトウェアのみで実現する方式に比べて，平均処理時間を短縮することである．

そして，排他制御時に割り込み処理が優先度を継承することにより，排他制御に無関係の割り込み処理が，高優先度の割り込み処理の実行を間接的に遅らせることを防止して，高優先度の割り込みに対する応答性を向上させることを目的とする．

4.3.1　ハードウェア優先度利用の REMON スケジューラの動作

図 4-13 にハードウェア優先度を利用した REMON の全体構造を示す．

さらに，図 4-14 にハードハードウェア優先度を利用した優先度継承セマフォを備えた REMON の全体構造を示す．図 4-13 はほとんど図 4-14 と同じ動作となるので，図 4-14 を中心に説明する．

図 4-14 における ICB が，割り込み処理の実行環境データを保存する領域である．個々の割り込み処理が個別の ICB に関連付けられている．

ICB の状態フラグは割り込み処理の状態を示すが，優先度継承セマフォをしなえる場合は優先度継承中を意味するビット（PRIORITY INHERITED）が追加されている．

ICB の割り込み優先度領域には，割り込み処理実行時に REMON スケジューラによって CPU の PSW にコピーされる優先度を設定しておく．通常はこの優先度は関連付けられた割り込みの割り込み優先度と同じにしておく．つまり割り込み優先度 3 の割り込み処理 3 に関連付けられた ICB[3]の割り込み優先度領域は 3 にしておく．

ICB の割り込み発生時に実行させる割り込み処理の先頭アドレス，スタックの初期値，及び割り込み処理が待ち状態になった時に CPU のレジスタを保存するための CPU コンテキス領域などは 3.1 節で説明した REMON と同じ意味である．

割り込み処理実行中に割り込みが発生すると，REMON スケジューラに処理が切り替わる．REMON スケジューラは，その時点の CPU コンテキストを ICB に保存して，ICB のフラグを必要に応じて変更する．REMON が割り込み処理を再開させる場合は，ICB に保存したデータを復元する．

外部割り込みが発生すると CPU ハードウェアから REMON スケジューラが呼び出される．しかし CPU 割り込み優先度の利用により REMON が呼び出されるのは PSW の割り込み優先度レベルに設定されている割り込み優先度より，優先度が高い割り込みのみとなる．つまり割り込み発生時に REMON が呼び出される回数が平均すると半分になる．

実行中の割り込み処理よりも優先度の高い割り込みの発生により REMON スケジューラが呼び出される．REMON スケジューラは，最初に割り込み発生時に実行されていた割り込み処理の CPU コンテキスをその割り込み処理の対応 ICB に保存する．

次に，REMON スケジューラは，実行する割り込み処理を決定する．具体的には REMON スケジューラは ICB 配列の先頭から実行可能な ICB を探索して，最初に見つけた ICB をもとに実行する割り込み処理を決定する．

REMON が優先度継承セマフォを実現する場合，ICB 配列ではなく優先度配列経由で，実行可能な ICB を検索して実行する割り込み処理を決定する．

最後に REMON スケジューラは ICB の割り込み優先度を CPU の PSW の割り込み優先度に設定したのち割り込み処理に制御を移す．ここで，強調しておくが REMON 不使用の多重処理では，割り込み処理の実行時にはハードウェア割り込み優先度が利用される．しかしその方式では，低優先度の割り込み処理がセマフォをロックしている時に，高優先度の割り込み処理がセマフォ待ちになった場合，低優先度の割り込み処理が実行されないため，シス

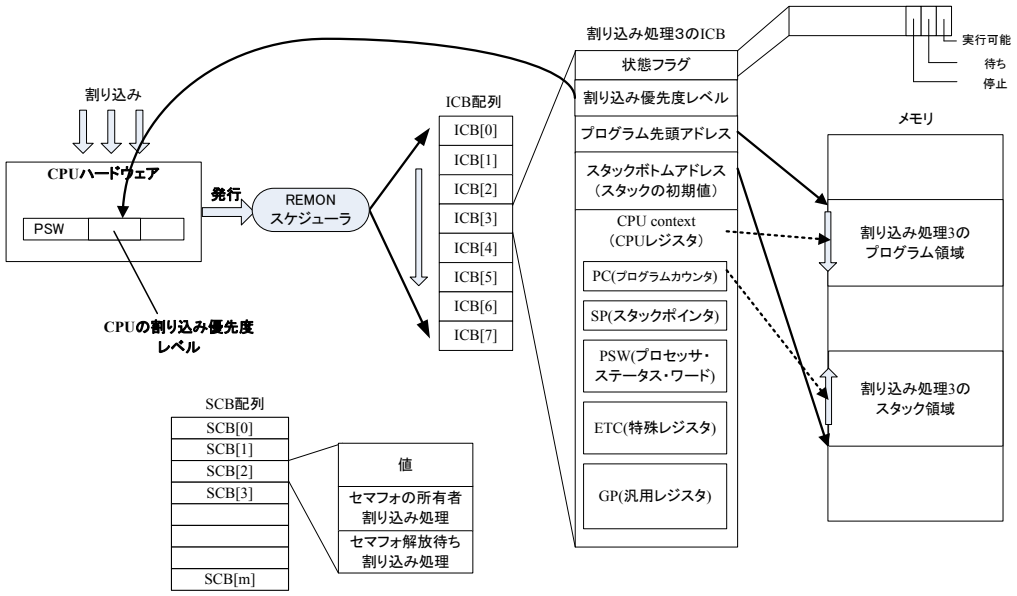


図 4-13　ハードウェア優先度を利用した REMON 全体図

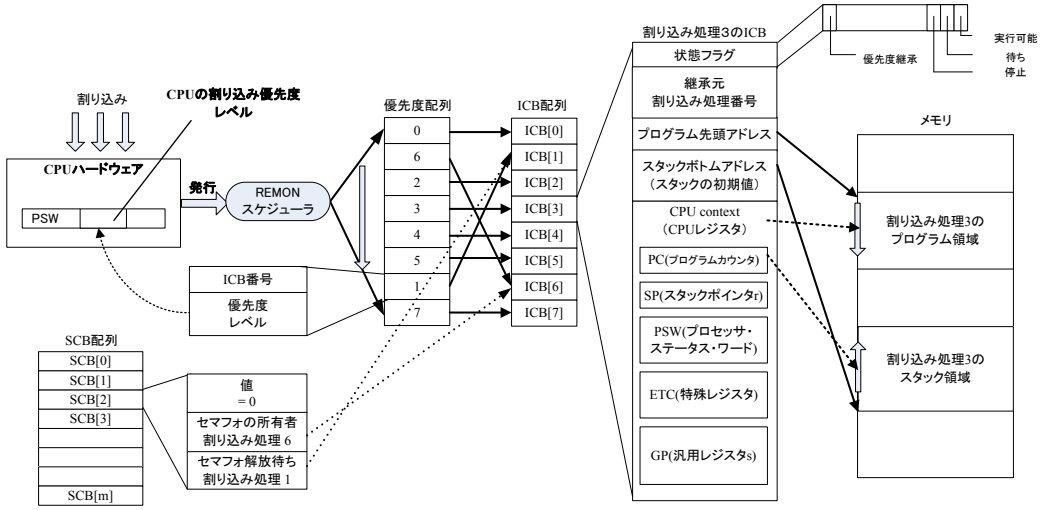


図 4-14　ハードウェア優先度を利用した優先度継承セマフォを備えた REMON

テムがデッドロック状態となる.

REMON はハードウェア割り込み優先度と割り込み処理実行時の優先度を独立させるために，発生した割り込みの優先度にかかわらず REMON は割り込み処理を実行させることができる.

図 4-14 における優先度配列は優先度継承 REMON セマフォ実現のため追加されたデータである．優先度配列は割り込み処理の優先度を決めるための配列であり，割り込み処理の優先度を変更するために使用される．

優先度配列には REMON がどの ICB を検索するのかの指定と，その割り込み処理を実行する時に PSW に設定するハードウェア割り込み優先度を設定しておく．優先度継承がない場合，優先度配列には ICB のインデックスがそのまま格納されている．優先度を継承する場合は，優先度配列内の ICB のインデックスが入れ替えられる．REMON では，ICB を調べる順番により優先度が決まるが，優先度配列を利用することにより，ICB の配列の位置を変更しなくとも，ICB を調べる順番を変更することができる．

ICB が REMON に調べられる順番を変更されることは，割り込み処理の優先度が変更されるのと同じ意味を持つ．図における優先度配列は低優先度の割り込み処理 6 が高優先度の割り込み処理 1 と優先度を交換している時の状態を示している．

ICB の状態領域には，従来の割り込み処理の状態を示すビット，新しく追加された優先度の継承を示す優先度継承ビットがある．

実行状態によって，あるいは優先度を継承して実行する場合は，このビットが REMON によって必要に応じて設定される．

ICB は配列になっており，割り込みが発生すると REMON は，優先度配列経由で ICB の配列の中から実行可能状態の割り込み処理を探して実行させる．

割り込み処理を実行させる時，REMON は CPU が備えるプロセッサステータスワード（PSW）の割り込みレベルを設定する領域に優先度配列内のハードウェア割り込み優先度の値を設定してから割り込み処理に制御を移す．

新たな割り込みが発生した時，CPU ハードウェアは，PSW に設定された割り込みレベルと，発生した割り込みのレベルの比較を行う．

発生した割り込みのレベルが，PSW に設定された割り込みレベル以下の場合，CPU ハードウェアが発生した割り込みを自動的にマスクするため，REMON が呼び出されることはない．

発生した割り込み優先度が実行中の割り込み処理の優先度より高く割り込み処理がプリエンプトされた場合や，P 操作によって実行を待たされる場合など割り込み処理の実行状態が変わる時は，REMON は実行していた割り込み処理の実行環境を ICB の CPU コンテキスト保存領域に保存する．そして REMON は ICB に保存した実行環境データを復元することによって割り込み処理を再開させる．また SCB も配列になっており，図 4-14 は優先度の低い割り込み処理 6 がセマフォ 2 を先に獲得した後で，優先度の高い割り込み処

理 1 が同じセマフォ 2 に対して獲得要求を発行し待ち状態になった後の様子を示す.

SCB の所有割り込み処理領域には 6 が，待ち割り込み処理領域には 1 が REMON によって設定されている．また優先度配列も 1 と 6 が入れ替えられている．なお優先配列のうち ICB 番号のみが入れ替え，優先度配列内の優先度レベルは入れ替えないため，CPU に設定する優先度レベルは，優先度レベルの高い割り込み処理と同じになる.

4.3.1.1　割り込み発生時の割り込み優先度レベル利用の共通処理

以下 REMON の動作に関して説明するがハードウェア優先度を利用する場合は優先度の高い割り込み処理中に優先度の低い割り込みが発生しても，ハードウェアレベルでマスクされソフトウェアが呼び出されることがないため，切り替えの有無により実行時間が一定となる方式は存在しない．また，ソフトウェアが呼び出されることがないため同じ割り込みが複数回発生した時の記録もできない.

図 4-15 に割り込み優先度レベルを利用した REMON のスケジューラの動作を示す．割り込み共通処理に関しては 3.1 節で説明した REMON と同じなので省略する．スケジューラでは，スタック上にコピーした PSW の割り込み優先度レベルを実行させる割り込み処理に関連付けられた ICB の割り込み優先度レベルの値に設定してから，割り込み割り込みからの復帰命令により割り込み処理を実行する.

そのため実行される割り込み処理は，設定された割り込み優先度レベル以下の割り込みが発生しても，CPU が割り込みをブロックするため，REMON スケジューラが呼び出されることはない.

実行中の割り込み処理よりも優先度の高い割り込みが発生した時のみ REMON スケジューラが呼び出される.

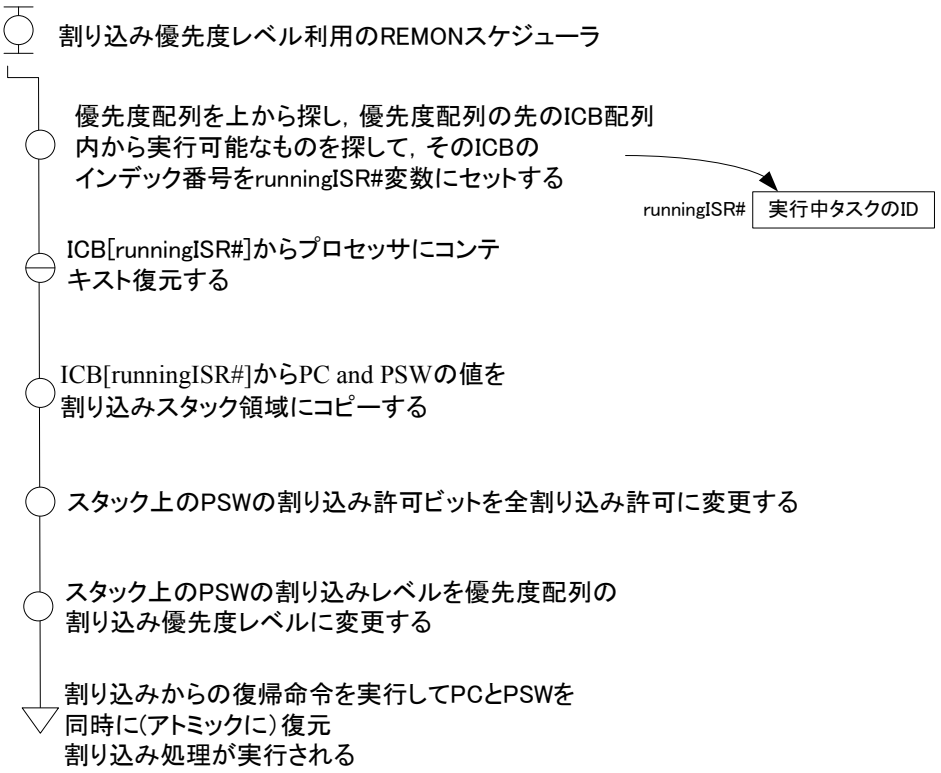


図 4-15　割り込み優先度レベルを利用する REMON スケジューラの処理

4.3.1.2　ハードウェア優先度を利用した優先度継承セマフォの P 操作

図 4-16 にハードウェア優先度を利用した優先度継承機能を備えたセマフォに対する P 操作の方式を示す．SCB のセマフォの値が 1 の場合，REMON はセマフォの値を 0 にする．そして SCB のセマフォ所有者領域に実行中の割り込み処理 ID をコピーする．そして，SCB のセマフォ解放待ち領域に，解放待ち割り込み処理がないことを示す負の値を書き込み，P 操作が呼び出された場所に復帰する．

セマフォの値が 0 の時，つまりセマフォがロックされており，かつセマフォをロックしている割り込み処理の優先度が実行中の割り込み処理の優先度より低い場合は，実行中の割り込み処理の ICB の状態を実行可能状態から待ち状態に変更して，実行中の割り込み処理とセマフォをロックしている割り込み処理の優先度配列のインデックス領域の値を入れ替える．

REMON スケジューラは優先度配列に従って ICB を調べ，最初に見つけた実行可能状態の割り込み処理を実行させるため，この入れ替えによって，セマフォをロックしている割り込み処理が，セマフォ解放待ち状態の高優先度

の割り込み処理の優先度で実行される．

REMON は優先度を継承した割り込み処理の ICB の状態領域の優先度継承ビットをオンにしたのち，ソフトウェア割り込みによって REMON スケジューラに制御を移す．セマフォをロックしている割り込み処理が実行を再開する時は，セマフォの解放を待っている割り込み処理の割り込み優先度で実行されるため中間優先度の割り込みが発生しても，CPU ハードウェアがその割り込みをマスクするため，REMON が呼び出されることはなく，優先度逆転は発生しない．

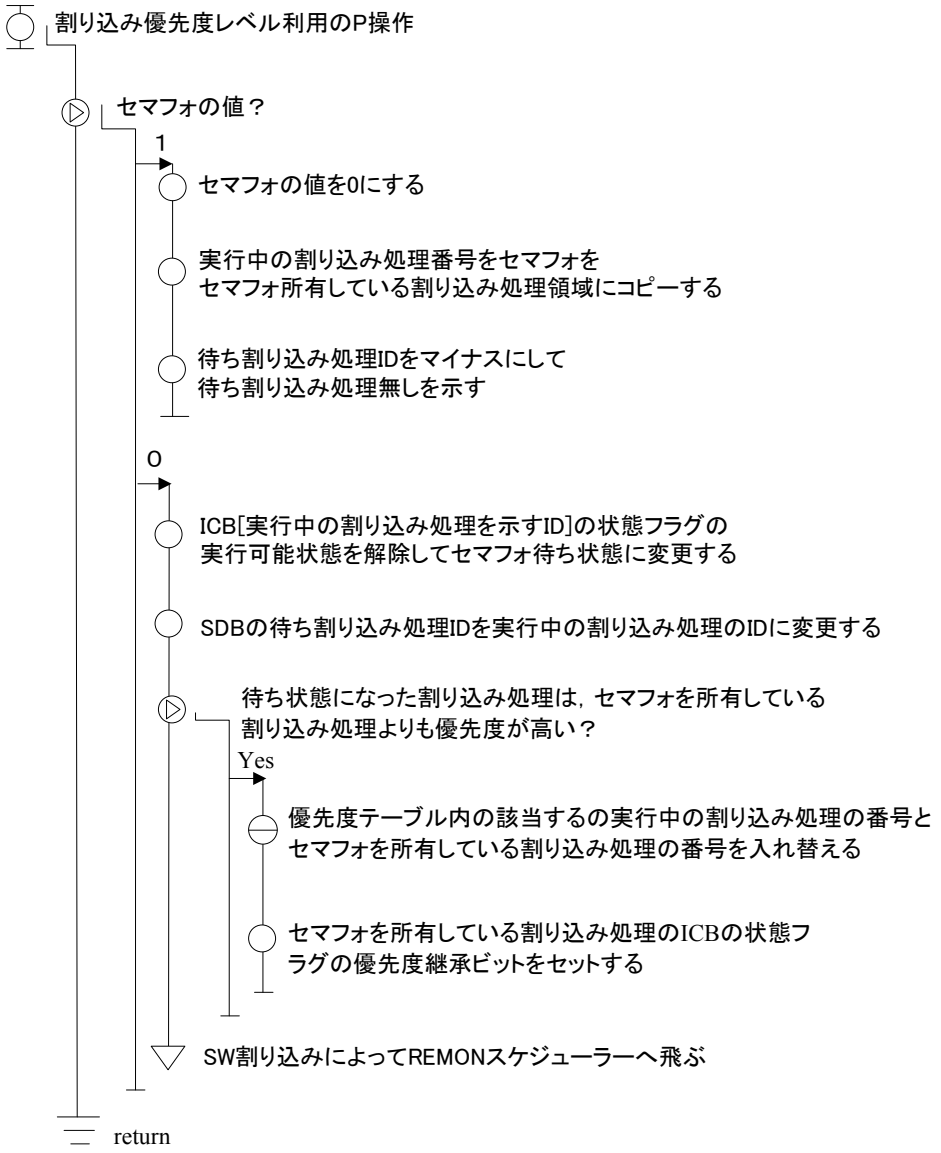


図 4-16　割り込み優先度レベルを利用する優先度継承 P 操作の処理

4.3.1.3　ハードウェア優先度を利用した優先度継承セマフォのV操作

図 4-17 にハードウェア優先度を利用した優先度継承機能を備えたセマフォに対する V 操作の方式を示す．セマフォが 0 の場合，REMON は SCB の待ち割り込み処理領域によりセマフォの解放を待っている割り込み処理の有無を調べる．

解放待ちの割り込み処理がない場合，REMON は SCB のセマフォの値の領域を 1 にするとともに，セマフォの所有者の領域を負の値にして，セマフォを所有している割り込み処理がないことを示した後，V 操作が呼び出された場所に制御を戻す．セマフォ解放待ちの割り込み処理が存在する場合，REMON はセマフォ解放待ちの割り込み処理の ICB の状態を実行可能にして，SCB の解放待ち領域の割り込み処理 ID 領域の値をセマフォ所有者領域にコピーする．最後に SCB の解放待ち領域を負の値として，セマフォ解放待ち割り込み処理がないことを示す．

さらに REMON は，今まで実行していた割り込み処理の ICB の状態を示す領域の優先度継承ビットにより，いままで実行されていた割り込み処理が，優先度継承状態で実行されていたかどうかを判定する．

もし優先度を継承して実行していた場合は継承元と継承先の割り込み処理の優先度配列のインデックス領域の値を再び入れ替える．この入れ替えにより優先度がもとに戻る．そして，優先度を継承して実行していた割り込み処理の ICB の状態領域の優先度継承ビットをクリアする．

最後に，ソフトウェア割り込みによって REMON スケジューラに制御を移す．

以上の処理により，ハードウェア優先度を利用した優先度継承機能を備えた割り込み処理セマフォが実現できる．

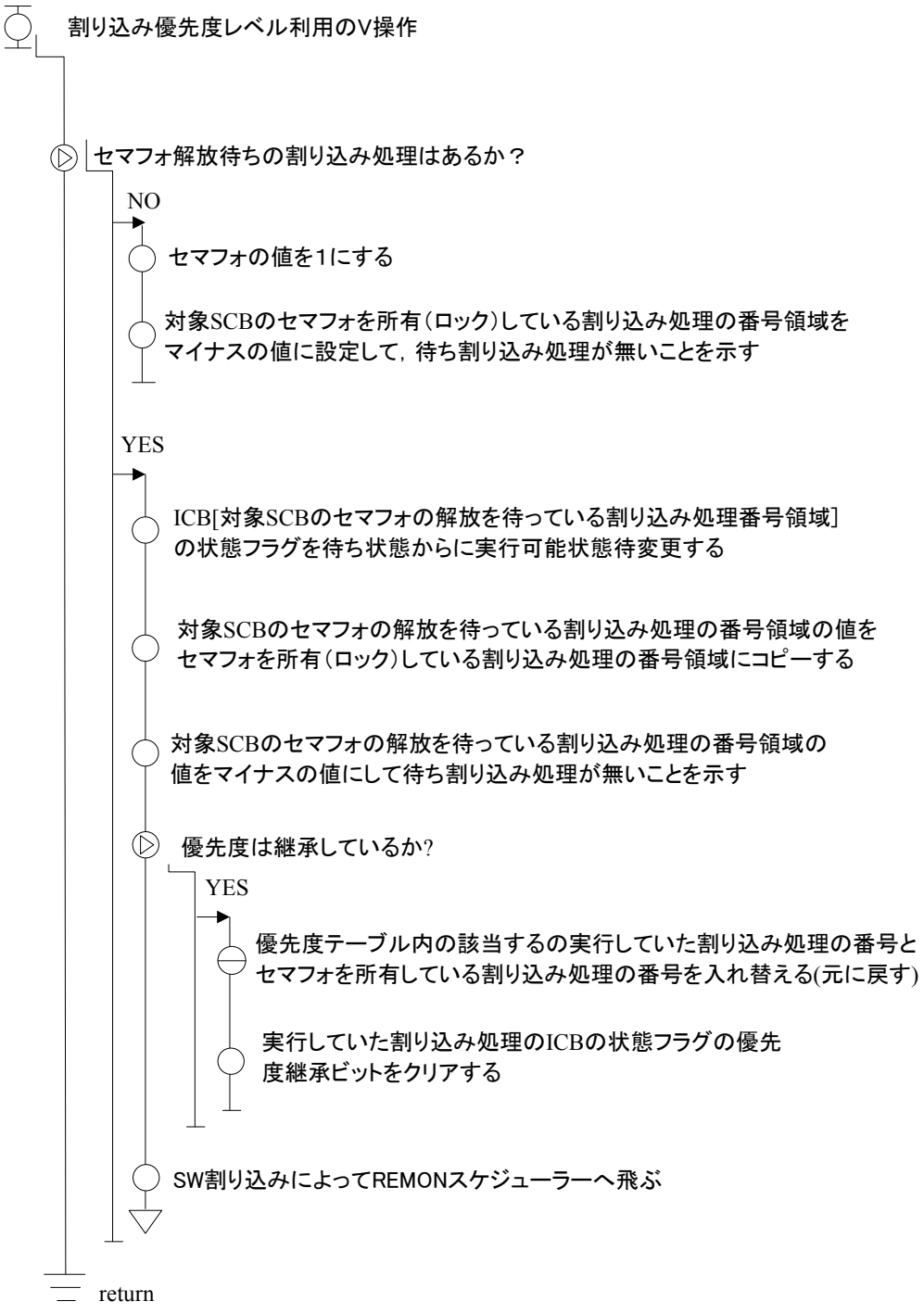


図 4-17　割り込み優先度レベルを利用する優先度継承 V 操作の処理

4.3.1.4　ハードウェア優先度を利用した優先度継承セマフォによる排他制御

図 4-18 にハードウェア優先度利用の優先度継承機能を備えたセマフォを使用した場合の割り込み処理の排他制御動作を示す.

割り込みが発生するタイミング，各割り込み処理の処理時間は図 4-13 に示したものと同じである．黒塗りの四角形が優先度を継承して割り込み処理 A と同じ優先度で実行される割り込み処理 C の排他制御処理を示す．図において，割り込み処理 C が P 操作によって，排他制御を開始した後，高優先度の割り込みによって，割り込み処理 A が実行を開始する．そして割り込み処理 A が P 操作によって排他制御を開始しようとするが，そのセマフォはすでに割り込み処理 C によってロックされている．そのため REMON は割り込み処理 A を待ち状態に移行させ，優先度を割り込み処理 A の割り込み優先度レベルのままで割り込み処理 C の実行を再開させる.

割り込み処理 C の実行中に，中優先度の割り込みが発生するが，CPU ハードウェアが自動的に割り込みをマスクするため REMON は呼び出されず割り込み処理 B が実行されることはない．つまり優先度逆転は発生しない.

割り込み処理 C が排他制御を終え V 操作を実行すると，割り込み処理 C は本来の優先度にもどり，割り込み処理 A が実行を再開する.

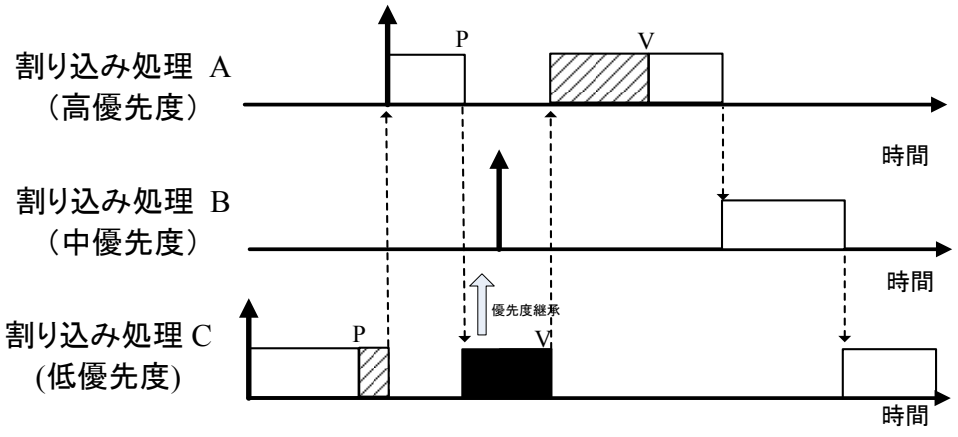


図 4-18　優先度継承セマフォ利用時の排他制御の動作

4.4　割り込み処理のスタックオーバーフロー検出とスタックの再割り当て方式

本節では組込みシステムにおいて，OS および MMU を使用せず，個別の割り込み処理のスタックオーバーフローを検出する方式，およびスタックの再割り当て方式を提案する[40].

提案方式の狙いは，従来 OS と MMU なしでは検出できなかった，組込みシステムにおけるスタックオーバーフロー検出を，REMON の割り込み処理単位に実行環境を持たせる機能[24]を利用して，実現することである．さらに，OS と MMU を使用しないで，スタックオーバーフロー発生時の自動的なスタックの再割り当てを実現すること，そして割り込み処理のスタックオーバーフロー発生時の原因特定時間を短縮することである．

これらにより，割り込み処理のスタックオーバーフローを原因とする不具合発生を減少させ，組込みシステムの信頼性向上を狙いとする．

4.4.1　割り込み処理のスタックオーバーフロー検出方式概要

従来の多重割り込み方式ではすべての割り込み処理が一つのスタックを共有している．

割り込み処理は，新たな割り込みが発生時してプリエンプトされると，実行を再開するためのデータである実行環境データがスタックに保存される．

スタックの使用順序は，割り込みの発生順という外的要因で決まるため，実行環境データの保存場所も毎回変化する．そのためスタックに保存された逆順でしかデータを復元できず，割り込み処理はプリエンプトされた逆の順番（入れ子の順番）でしか再開させられない．つまり割り込み処理は独立した実行環境は持てない．

割り込み処理が独立した実行環境を持てないとセマフォなどの排他制御の手段を使用できない．なぜなら割り込み処理がセマフォ獲得要求操作をしてロックされた場合，その再開は他の割り込み処理のセマフォ解放操作によるため，再開の順番は割り込み処理が一時停止させられた順番とは無関係であるからである．

提案方式は REMON を利用して個々の割り込み処理に独立した実行環境を与えることにより，割り込み処理の一時停止動作を可能にしている(11).

REMON は個別の割り込み処理に個別のスタックも含めた独立した実行環

境を与える．そして，割り込みが発生するごとに，REMON スケジューラがスタックも含めた割り込み処理の実行環境の入れ替えを行い，割り込み処理を切り替える[24][40].

各割り込み処理の実行環境を保存するために，REMON は個々の割り込み処理に ICB というデータを関連づけている．REMON はこの ICB を使用して，個別の割り込み処理に独立した実行環境を持たせることができる．組込みシステムの信頼性向上のために REMON を利用した，割り込み処理のスタックオーバーフロー検出機能を提案する．

提案方式では，システム起動時に割り込み処理毎のスタックの終端部（下位アドレス部分）にマジックナンバと呼ぶビットパターンをシステム初期化時に書き込んでおく．そしてスケジューラが実行される毎にすべての割り込み処理のマジックナンバが書き換えられていないどうかを検査して，マジックナンバが書き換えられている場合はスタックオーバーフローが発生したと判断する．マジックナンバを各割り込み処理のスタックの終端部だけでなくスタックの途中にも配置すれば，割り込み処理のスタックの使用状況も知ることができる[40].

4.4.2　マジックナンバの設定

電源投入時の割り込み処理毎のスタックへのマジックナンバの設定，各割り込み処理のスタックの使用状況のログの記録などのために，スタック定義ブロック SDB (Stack Definition Block – SDB)を REMON に設ける．

図 4-19 に SDB の構造を示す．SDB にはシステム全体のスタック領域の先頭(最下位)アドレス，スタックの合計容量，各割り込み処理に配置するマジックナンバの個数を示す領域，スタックオーバーフロー発生時のエラー番号，各割り込み処理のスタック使用状況を記録する領域，そして，マジックナンバのバイト数，マジックナンバ，および割り込み処理毎のスタック容量を示す領域を備える．このうちエラー番号，スタック使用状況は，REMON が実行開始時に設定する領域である．これ以外はあらかじめ設定しておく領域である．

システムの初期化時，SDB の設定に従って，スタックの割り当て，スタックへのマジックナンバの書き込みを行う．マジックナンバは，各割り込み処理のスタックの使用状況を把握するため，スタックごとに複数個配置してもよい.

REMON は実行されるたびに，すべての割り込み処理の，すべてのマジックナンバを検査して，SDB の割り込み処理の使用状況を示す領域に記録す

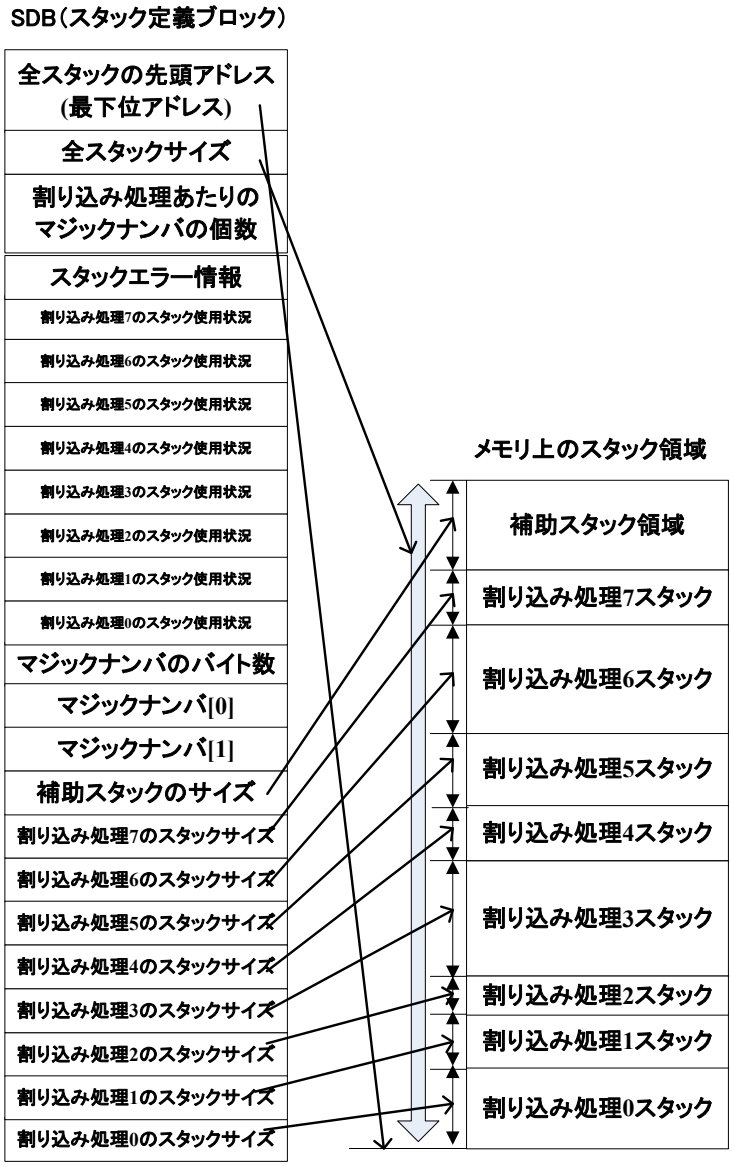


図 4-19　スタック定義ブロック SDB の構造

る．

割り込み処理のスタックの途中のマジックナンバが，書き替えられていてもそれはスタックオーバーフローではない．スタックの終端(最低位)のアドレスにおかれたマジックナンバが書き換えられた時がスタックオーバーフローである．

図 4-20 に電源投入時の初期化動作を示す．最初に各割り込み処理のスタック容量の合計が全スタック容量を超えていないか計算して，残った領域を予備スタック領域とする．そして SDB の割り込み処理のスタック容量をもとに，各割り込み処理に，スタック領域の高位アドレス側から，スタックの割り当てを行い，ICB のスタックボトム(先頭)領域に設定する．次いでマジッ

クナンバを，各割り込み処理に指定された個数だけ等間隔に設定する．これらのマジックナンバがどこまで書き替えられたかにより，スタックの概略の使用状況を知ることができる．

提案方式はスタックのすべての領域にマジックナンバを書き込むのではなく，スタックの一部に書き込むことによって，スタックの最大使用容量の検査を高速に行える効果がある．

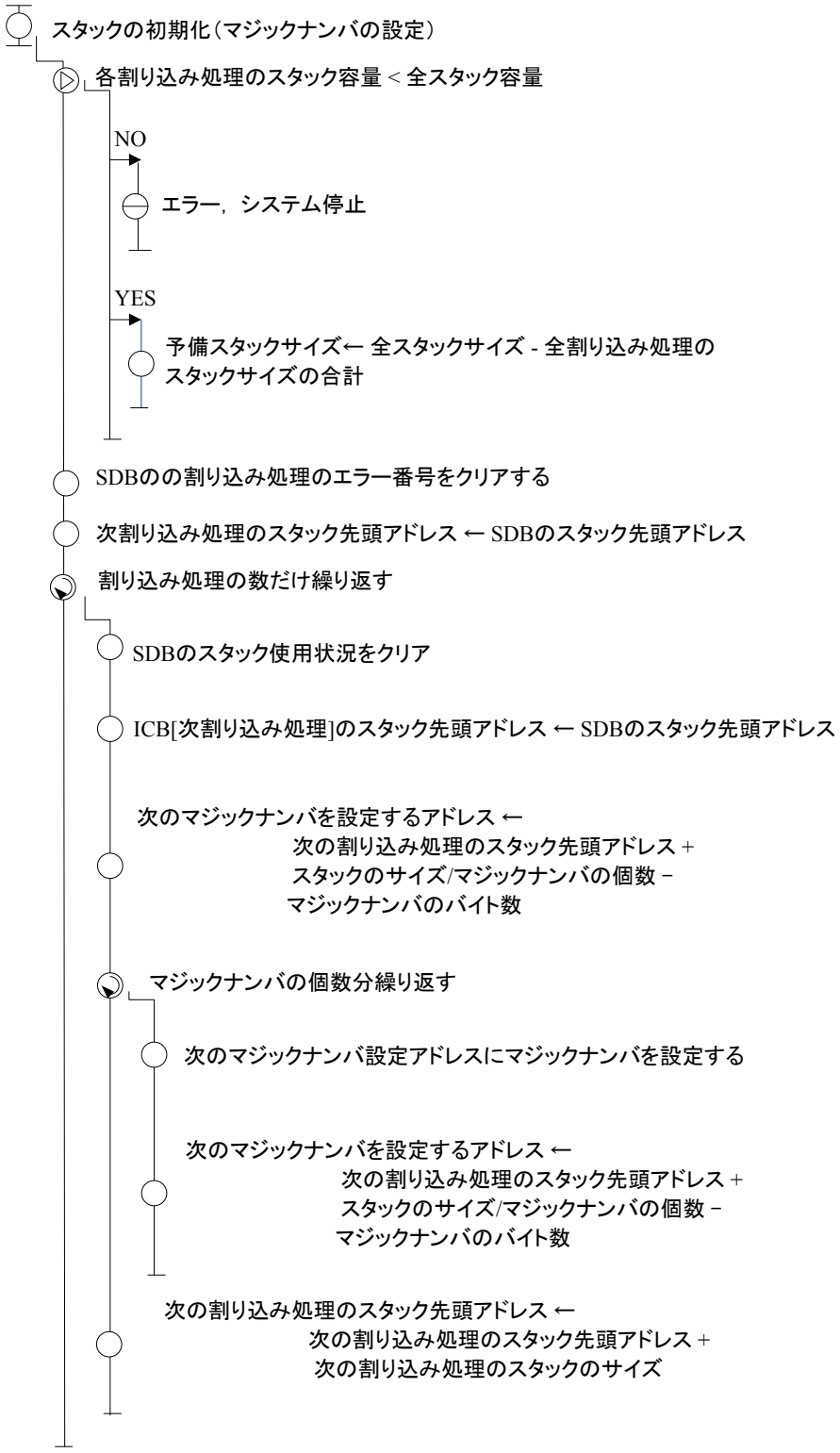


図 4-20　スタックの初期化処理

図 4-21 にマジックナンバの個数が 4 の場合の割り込み処理のスタックを示す.

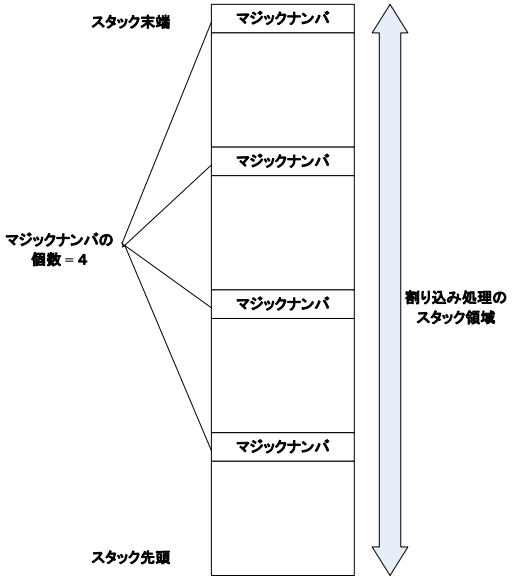


図 4-21　マジックナンバが 4 個の場合の例

4.4.3　スタックオーバーフロー検出方式

図 4-22 にスタックオーバーフロー検出機能を備えた REMON の全体構成を示す.

割り込みによって REMON スケジューラが実行され，割り込み処理を切り替える．REMON 実行時に，その時点の割り込み処理の実行環境を ICB に保存する．この割り込み処理を再開させる場合は，ICB に保存したデータを使用する．REMON は実行されるたびに，スタックオーバーフローが発生していないか検査を行う．スタックオーバーフローを検出した場合は，スタック再割り当てなどのスタックオーバーフロー発生時の処理を実行する．またスタックポインタレジスタ(SP)が，すでにスタック領域を超えている場合も，スタックオーバーフローが発生しているため，スタックオーバーフロー発生時の処理を行う.

スタックオーバーフローが発生して，一時的に割り当てられた以上のスタックを使用しても，再び領域内に戻る場合もある．この場合，REMON 実行時に，SP がスタック領域内にあっても，スタックオーバーフローは発生している．そのためスタックオーバーフローを検出するためには，マジックナンバの検査も必要である．図は例としてマジックナンバの個数が 4 の時の，割

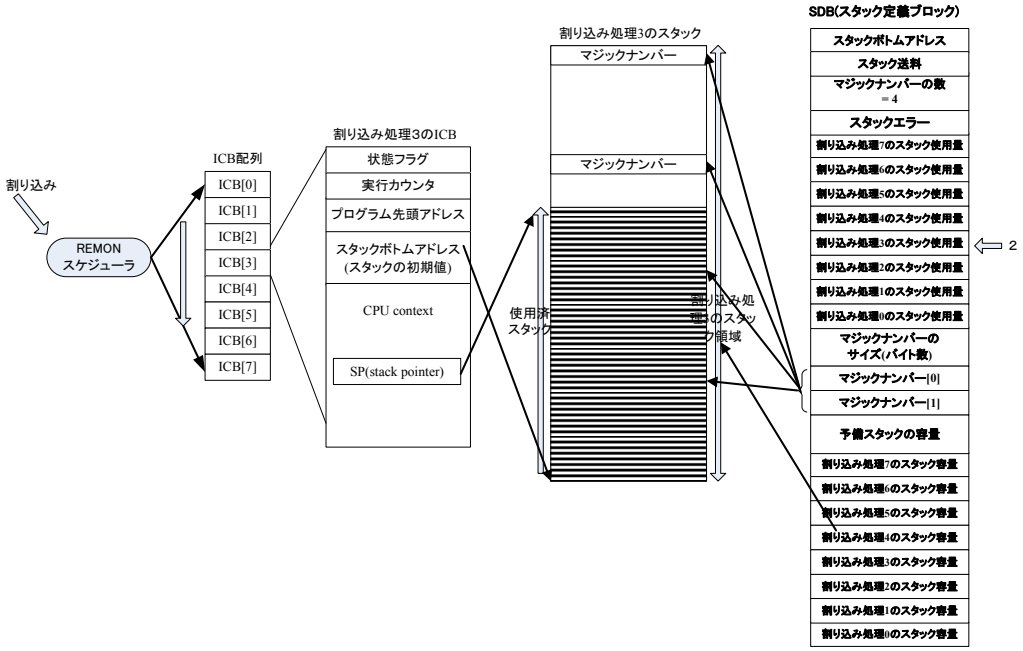


図 4-22　スタックオーバーフロー検出機能を備えた REMON の全体構成

り込み処理 3 のスタックを示している．スタックはハッチングした部分が，使用しているスタック領域を示す．割り込み処理 3 のスタックは，マジックナンバが 2 個書き変えられているため，SDB の割り込み処理 3 のスタック使用状況には，2 を記録する．

図 4-23 に REMON のスタックオーバーフロー検出動作を示す．

割り込みが発生して，REMON が実行され，割り込み処理の切り替えを行う時に，マジックナンバが変更されていないかどうかの検査を行う．

書き換えられているマジックナンバがあった場合は，SDB のスタック使用状況領域に記録する．

この領域を調べれば，各割り込み処理のスタック使用状況を容易に知ることができるため，どの割り込み処理がスタックオーバーフローを起こしたのかの特定を迅速に行うことが可能になる．

書き換えられているマジックナンバが，その割り込み処理のスタクの終端アドレス（最下位アドレス）に置かれたものである場合は，スタックオーバーフローの発生を示す．

外部環境に起因する外部割込みの発生は，環境によっては一時的に想定以上の割り込みが発生する場合があり，その場合はスタックを一時的に増やす対策も有効である．

環境によって発生する割り込みは，組込みシステム製品の設置場所によっ

図 4-23　スタックオーバーフロー検出機能動作

て異なるため，あらかじめ特定の割り込み処理にスタックを余分に割り当てておくことは，特に内蔵メモリ容量に制約があるシングルチップマイクロコンピュータにおいては難しい．そのため，発生したスタックオーバーフローに応じてスタックの容量を増やすことは，環境によるスタックオーバーフロー再発を防止する効果が期待できる．

図 4-24 にスタック再割り当ての動作を示す．スタックオーバーフロー発生時，まだスタックの再割り当てが行われていない場合，スタックオーバーフローを発生させた割り込み処理に割り当てたスタックのサイズを予備領域(aux)分増やして，SDB の値を変更する．そして，SDB のエラー番号記録領域を，スタック再割り当て済みに設定し，ソフトウェアリセット命令を実行して，システムを再起動させる．

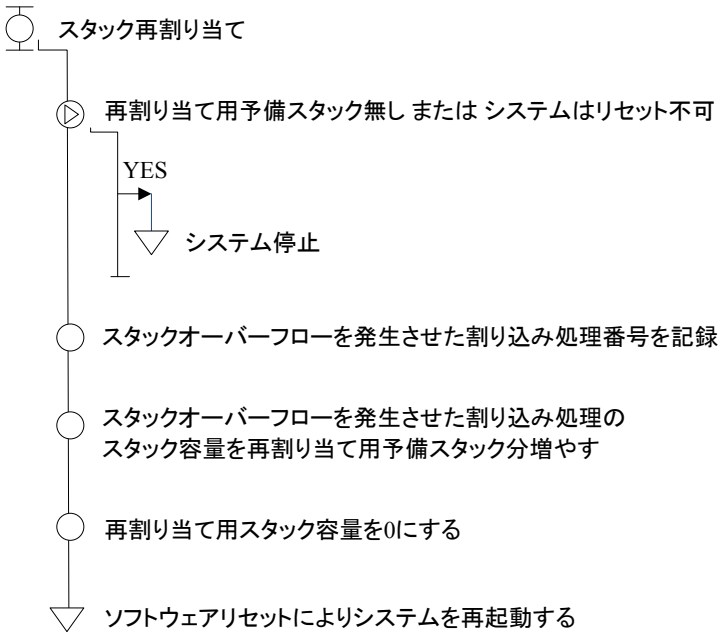


図 4-24　スタック再割り当て動作

割り込み処理のスタックオーバーフロー発生時に，当該割り込み処理のスタック領域を増やしてシステムを再起動させるこの方式は宇宙機器やパチンコなどの娯楽機器のように，定期的にリセット信号を入力することによって，システムの再起動を行い，システムの信頼性を確保しているシステムにおいては，特に有効である．

再起動できない組込みシステムの場合には，たとえば EEPROM やフラッシュメモリなど不揮発性のメモリに SDB の内容を書き込み，スタックオーバーフロー発生時の情報を保存する．

割り込み発生時は実行する命令が強制的に決まるため，不揮発性メモリに書き込む命令を確実に実行することが可能になり，迅速な不具合原因の特定を行うことを可能になる．スタックの予備領域が 0 の場合は，再割り当てができないため，システムが誤動作しないように，システムを停止させる

第５章

評価実験および考察

本章ではREMONおよびRTOSをCPUボードに実装して行った，提案システムの検証に関して述べる．

5.1　REMONの評価実験と考察

提案方式 REMON を使用することにより，従来割り込み処理の排他制御に使用していた DI/EI 方式に比較してリアルタイム性が向上すること，およびRTOS を使用する場合と比較して，性能が低下しないことを評価確認するため次の実験を行った．

・REMONのセマフォを用いて排他制御を行った場合と，DI/EIを使用して排他制御を行った場合との時間の測定

・REMON を用いた場合と RTOS を使用した場合のタスク起動時間，排他制御に要する時間の測定

また，割り込み処理のハードウェア優先度利用の優先度継承機能を備えたセマフォ動作確認のため，DI/EI 方式による排他制御，優先度継承機能を備えていないセマフォ方式による排他制御との動作比較を行った．ハードウェア優先度利用の優先度継承機能を備えた REMON セマフォの使用により，高優先度の割り込みに対する応答性が向上することの確認が目的である．さらにハードウェア優先度利用の優先度継承機能を備えた REMON セマフォ方式，優先度継承機能を備えていない従来の REMON セマフォ方式，およびRTOS のセマフォ方式の処理時間の測定を行った．ハードウェア優先度利用の優先度継承方式を備えたセマフォの性能確認が目的である．そして割り込み処理のスタックオーバーフロー検出処理の動作確認，スタックオーバーフロー検出時のスタックの再割り当て処理の動作確認，リアルタイム性の観点からの処理時間，スタックオーバーフローの検出機能追加による処理時間の

増加の評価，およびスタックオーバーフローの検出機能による不具原因特定までの時間短縮の確認のための実験を行った．

REMON には処理の切り替えの有無に関わらず処理時間が一定の REMON と処理の切り替えの有無で処理時間は異なるが平均処理時間が短い REMON があるが，処理の切り替えの有無で処理時間は異なるが平均処理時間が短い REMON を特別 REMON と呼ぶ．

5.1.1　使用機材と実験方法

図 5-1 に実験機材の写真を示す．機材としてルネサスエレクトロニクス社製 M30620FCAFP(M16C/62A シリーズ)シングルチップマイクロコンピュータを搭載した OAKS 電子株式会社製の OAKS16-62P BoardKit および OAKS16-LCD ボードを使用した．図の右側がこの CPU ボードである．

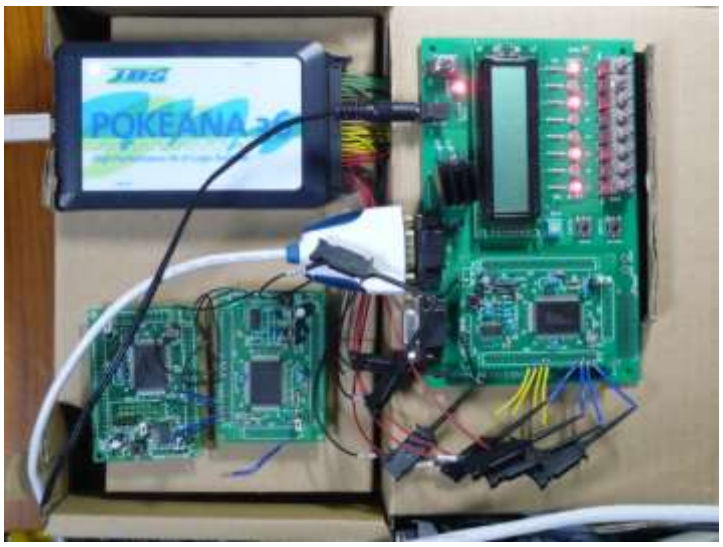


図 5-1　評価実験に使用した CPU ボードとロジックアナライザ

表 5-1 に M30620FCAFP の主な仕様を示す．16MHz の周波数で実行させたため，評価ボードにおける最短命令実行時間は 62.5ns である．時間測定は，同図左上に示す JDS 社の POKEANA36 ロジックアナライザを使用した．時間測定に使用したロジックアナライザは 800MHz で信号のサンプリングを行ったので，最少測定単位は 0.25ns となる．

時間測定は測定区間を汎用ポートの ON/OFF 命令で囲み，その時間をロジックアナライザで測定することにより実施した．すべての測定は 100 回実施して，その平均値とした．

表 5-1　評価実験に使用した M16C/62A の仕様

動作周波数	16MHz
フラッシュメモリのサイズ	128KB
SRAM のサイズ	10KB
割り込みレベル	0-7

比較対象の RTOS としては TOPPERS/JSP カーネル（JSP）を使用した．これは TOPPERS プロジェクトにより開発・公開されている μITRON4.0 仕様のスタンダードプロファイル[44]に準拠したオープンソースの RTOS である．測定には，最新バージョンの 1.4.3 のソースコードを使用した．JSP は排他制御の手段としてイベントフラグも推奨しているため，イベントフラグを用いた場合の排他制御の時間測定も実施した．

以上 3 種類のソフトウェアをそれぞれ単独で前記ボードで動作させ時間測定を行った．

5.1.2　測定ツール REMON モニタの開発

ロジックアナライザによる時間測定は正確であるが，割り込み処理間の動作関係を直観的に判断するのは難しい．そこで PC 上の画面に，割り込み処理の動作関係表示するツールとして REMON モニタを開発した．図 5-2 に REMON モニタの表示を示す．

REMON モニタとは，実行する割り込み処理を REMON が切り替えるたびに割り込み処理番号をシリアル通信によりホスト PC に送信して，割り込み処理の動きをホスト PC で画面に表示するツールである。

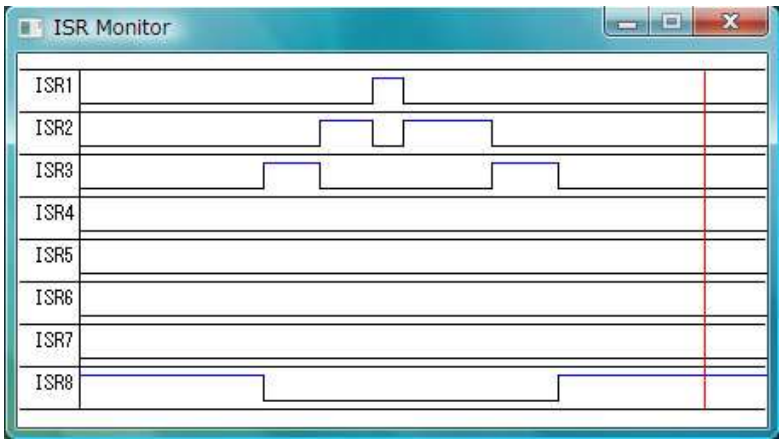


図 5-2　REMON モニタ

REMON スケジューラが，実行する割り込み処理を切り替えるたびに，その割り込み処理 ID をシリアル通信によりホスト PC に送信し，ホスト側ではデータを受信するたびに実行された割り込み処理を画面で表示する．

また REMON 不使用の場合は，割り込み処理の先頭と最後で，割り込み処理番号をシリアル通信によりホスト PC に送信することを利用して割り込み処理の動きを表示する。

REMON モニタは PC との通信に速度の遅いシリアル通信を使用していること，PC が細かい時間の計測には不適切であることから，時間を長く設定して割り込み処理の定性的な動作確認のみに使用した。

REMON モニタを用いると割り込み処理が実行したり，切り替わったりする様子を，視覚的に確認することができる．ただしこのモニタは PC との通信に速度の遅いシリアル通信を使用していること，演算を PC で行っていることなどの理由で，測定値の変動が大きい．そのため REMON モニタは定性的な動作確認のみに使用した．一方，ロジックアナライザは測定値に変動がなく，測定誤差も少ないため処理時間の測定など定量的測定に使用した．

5.1.3.　DI/EI 方式との比較のための実験方法

排他制御を DI/EI を使用した場合と，REMON セマフォを使用した場合の，各割り込み処理の終了時間などを測定した．優先度の低い割り込み処理同士の排他制御が，優先度が高い割り込み処理に影響を与えないことを確認するための実験であり，次の方法で実験を行った．

1) 最初に優先度低（割り込み優先度レベル 4）の割り込みを発生させ実行優先度低の割り込み処理 3 の実行を開始させる．

2) 上記 1)の実行中に優先度中の割り込み（割り込み優先度レベ 5）発生させ実行優先度中の割り込み処理 2 の実行を開始させる．

3) 上記 1), 2)の実行中に優先度高（割り込み優先度レベ 6）の割り込みを発生させ，実行優先度高の割り込み処理 1 の実行を開始させる．

4) 排他制御は割り込み処理 2 と割り込み処理 3 の間で行う．

表 5-2 に定性的評価に使用した上記のタイミングを示す．表 5-3 に定性的評価に使用した上記のタイミングを示す．両者の割合は等しくしたが，定性的評価に用いた時間の長さは定量的評価に用いたものの 10^6 倍としている．

表 5-2，表 5-3 の実行開始時間は低優先度の割り込みを入れて割り込み処理 3 が実行を開始した時間を 0 として，そこからの相対時間である．総実行時間は各割り込み処理が CPU を使用する時間である．排他制御の開始時間，排他制御の終了時間は割り込み処理 2 及び 3 各々の処理における開始時間から

表 5-2　定性的評価に使用した割り込み処理のスケジュール (単位：秒)

	割り込み処理 1	割り込み処理 2	割り込み処理 3
実行開始時間	7.0	3.2	0
総実行時間	1.8	9.0	7.2
排他制御開始時間	-	10.0	18.0
排他制御終了時間	-	24.0	36.0

表 5-3　定量的評価に使用した割り込み処理のスケジュール (単位：μs)

	割り込み処理 1	割り込み処理 2	割り込み処理 3
実行開始時間	701.75	320.69	0
総実行時間	181.1	900.33	721.54
排他制御開始時間	-	100.21	180.44
排他制御終了時間	-	239.92	360

の経過時間である.

5.1.4　DI/EI 方式との比較実験の結果

最初に実行結果を REMON モニタで示す．図 5-3 に三つの割り込み処理が多重割り込み許可状態にあるが、排他制御をおこなわない時の動作を示す．図 5-4 に DI/EI を用いて排他制御を実行した時の動作を示す．図において DI が全割り込み禁止命令を，EI が全割り込み許可命令の実行を示す．図 5-5 に REMON のセマフォを用いて排他制御を行った時の動作を示す．図 5-5 において，P が P 操作の，V が V 操作の実行を示す．

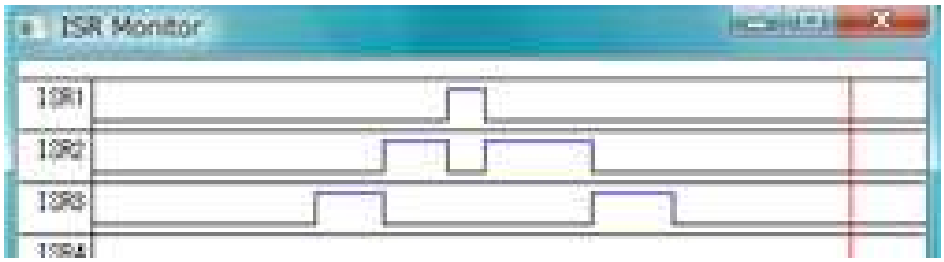


図 5-3　排他制御をおこなわない多重割り込み

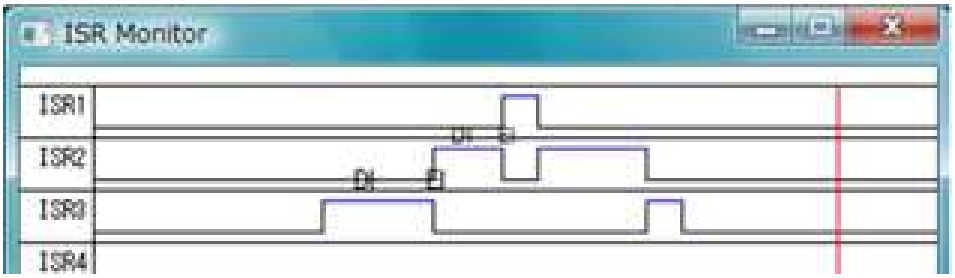


図 5-4　DI/EI による排他制御

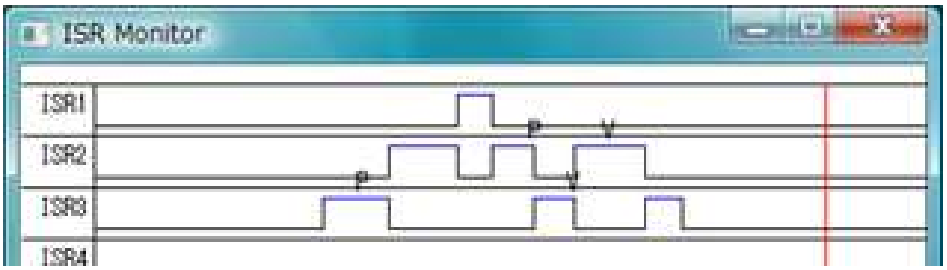


図 5-5　REMON セマフォにより排他制御

次に表 5-4 に DI/EI と REMON セマフォによる割り込み処理の排他制御時間を測定した結果を示す.

各割り込み処理の処理開始時刻，処理終了時刻を測定している．割り込み処理 2 と割り込み処理 3 に関しては，排他制御の開始時刻と排他制御の終了時刻も測定している.

表 5-4　DI/EI と REMON セマフォによる排他制御時間（単位：μs）

	割り込み処理 1		割り込み処理 2		割り込み処理 3	
	DI/EI	REMON セマフォ	DI/EI	REMON セマフォ	DI/EI	REMON セマフォ
実行開始時間	900.33	702.75	561.84	320.69	0	0
排他制御開始時間	-	-	640.65	625.221	180.44	180.44
排他制御終了時間	-	-	880.57	945.181	540.44	868.1
実行終了時間	1061.67	882.85	1621.87	1645.09	1802.97	1849.41

5.1.5　DI/EI 方式との比較結果の評価と考察

排他制御をおこなわない場合（図 5-3）と，DI/EI を用いて排他制御を実行した場合（図 5-4）を比較すると，割り込み処理 3 の割り込み禁止により，割り込み処理 3 よりも優先度の高い割り込み処理 2 の実行が遅らされていることがわかる.

REMON のセマフォを用いて排他制御を行った場合（図 5-5）から割り込み処理 2 が P 操作を行った時にはじめて先に P 操作を行っていた優先度の低い割り込み処理である割り込み処理 3 に制御が移ることから REMON のセマフォが正しく動作していることがわかる.

次に表 5-4 から優先度が最も高い処理である割り込み処理 1 の終了時間が 178.82μs（= 1061.67μs–882.85μs）早くなっている.

また起動要求から終了するまでの時間を比較すると

・REMON を使用した場合，181.1μs（= 882.85μs–701.75μs）

・REMON を使用しない場合，359.92μs（= 1061.67μs–701.75μs）

となる.

以上の結果より REMON の使用により，システムのリアルタイム性が改善されたことがわかる.

従来の割り込み禁止と許可を用いて割り込み処理の排他制御を実現してい

た部分の REMON 対応への変更は容易である．割り込みを禁止している部分を P 操作に，割り込みを許可している部分を V 操作に置き換えればよい．REMON の使用により，リアルタイム性の向上を期待できる[24].

5.1.6　RTOS との比較実験の方法と結果

REMON においては並行実行の単位は割り込み処理であるが，JSP に対応させるため，JSP と比較している節では REMON の割り込み処理もタスクと呼ぶ.

REMON と JSP におけるタスク起動時間とセマフォに対する操作の処理時間を測定した．JSP に関してはイベントフラグに対する操作の処理時間も測定した．タスクの起動時間としては優先度の低いタスクが，優先度の高いタスクを，タスク起動のサービスコール関数を呼び出して起動した場合，つまりタスクスイッチが伴う場合，優先度が逆で，タスクスイッチが伴わない場合があるが，両方の時間を測定した.

通常の REMON は，タスクスイッチの有無が，処理時間に影響を与えない方式としている．セマフォに対する P 操作と V 操作も，タスクスイッチを伴う場合と，伴わない場合があるが，両方の処理時間を測定した.

最後に P 操作と V 操作の組み合わせで行う，排他制御の処理時間を測定した．排他制御に関してもタスクスイッチを伴う場合と伴わない場合がある.

タスクスイッチを伴う排他制御とは優先度の高いタスクが P 操作で待ちになった後，優先度の低いタスクが V 操作を行った場合である.

タスクスイッチを伴わない排他制御とは優先度の低いタスクが P 操作で待ちになった後，優先度の高いタスクが V 操作を行う場合である．これも両方の処理時間を測定した．さらに JSP の排他制御の処理時間が，2 つのタスクの優先度が同じで処理の切り替えが発生しない場合でも異なるためそれも測定した．表 5-5 に REMON および特別 REMON と JSP の処理時間を測定した結果を示す.

5.1.7　RTOS との比較実験の評価と考察

REMON と JSP のタスク起動時間や排他制御などの時間測定結果を示した表 5-5 から REMON と JSP の処理時間を比較すると，タスク起動の場合，JSP ではタスク切り替えが発生しない場合で 33.277μs，タスク切り替えが発生する場合 36.704μs である．REMON はどちらの場合も 14.53μs と REMON の方が JSP よりも 2 倍以上高速である．またタスクスイッチが発生する排他制御では約 1.48 倍，タスクスイッチが発生しない排他制御では約 1.36 倍高

表 5-5　REMON と JSP の処理時間の比較（単位：μs）

機能	REMON	特別 REMON	TOPPERS/JSP
タスク起動 （処理の切り替えなし）	14.53	-	33.277
タスク起動 （処理の切り替えあり）	14.53	-	36.704
P 操作 (単独) （処理の切り替えなし）	19.027	4.491	8.472
V 操作 (単独) （処理の切り替えなし）	17.717	3.181	11.091
wai_flg (P 操作相当)	-	-	18.757
set_flg (V 操作相当)	-	-	10.219
セマフォによる排他制御 （処理の切り替えなし）	22.287	23.221	28.51
セマフォによる排他制御 (同じ優先度のため 処理の切り替えなし)	-	-	28.229
セマフォによる排他制御 （処理の切り替えあり）	22.287	21.865	33.091
イベントフラグによる排他制御 (処理の切り替えなし)	-	-	35.333
イベントフラグによる排他制御 (同じ優先度のため 処理の切り替えなし)	-	-	33.21
イベントフラグによる排他制御 (処理の切り替えあり)	-	-	40.882

速である．

JSP は READY 状態のタスクをリストで保持している．リスト方式は，データを読み込まないと次のデータの場所がわからないため，あらかじめ次のデータの場所がわかっている配列方式に比較すると，処理時間が多くかかる場合が多い．JSP における，タスク起動やセマフォ操作は，上記 READY 状態のタスクリストに対する操作が必要である．

一方 REMON では，上記操作は配列内の ICB のフラグ部に対するビット操作のみである．これが REMON の方が高速となった原因と考えられる．

JSP がタスクの状態によって処理時間が変化するのに対して REMON では処理時間は変化しない．リアルタイム性を保証した設計を行うためには，処

理時間の変化が少ない方が容易である．つまり処理時間に変化がないREMONの方が，リアルタイム設計が容易となる可能性がある．

排他制御で処理の切り替えが発生しない場合に相当するのが，表における処理の切り替えを伴わないP操作，および処理の切り替えを伴わないV操作である．この場合はREMONの19.027μs, 17.717μsに対して，JSPは8.472μs, 11.091μsと高速である．これはJSPではタスクスイッチが必要かどうかを最初に判定して，タスクスイッチが必要でない場合は，タスクスイッチの処理を省略するためである．

REMONでは処理時間を一定にするために，タスクスイッチの処理を行ってからその必要性の判断を実行している．

そしてJSP同様の処理として，タスクスイッチの必要がない場合の処理時間を短縮したのが，特別REMONである．特別REMONでは，P操作とV操作がそれぞれ4.491μs, 3.181μsとJSPよりも高速である．表示が中心のカーナビゲーションシステムのような，厳密なリアルタイム性よりも，高い平均性能を求めるような組込みシステムの場合は，特別REMONの方が適している場合もある．

5.1.8　メモリ使用量の評価と考察

表5-6にREMONのメモリ使用サイズを示す．初期化処理のデータサイズは8レベルの割り込み処理用スタック領域およびICB領域を含むため大きくなっているが，割り込み処理用のスタック領域はREMONの使用の有無にかかわらず必要な領域である．またセマフォのデータサイズには16個のセマフォ構造体の領域を含んでいる．これらの処理を含めてもメモリサイズの合計はコード1274バイト，データ580バイトである．

RAMサイズの大部分は，REMONを使用しなくても必要な，CPUコンテキストの保存領域とスタック領域である．また，OSEKで定義されている同時に実行されないタスクと同様に，同時に実行されない割り込み処理を定義すれば，その割り込み処理間でスタックの共有も可能となりスタックの使用量削減の可能性がある．ただしその場合，現状のシステム構成の見直しが必要になる．REMONは，現状のシステム構成を変更せずに，リアルタイム性を向上させることを，主な目的としており，スタックの共有機能は備えていない．

表 5-6　REMON の使用メモリサイズ（単位：バイト）

機能	初期化	スケジューラ	処理（タスク）	セマフォ	合計
命令	127	248	240	514	1129
データ	470	4	28	74	576

5.1.9　今後の展開に関する考察

（１）従来は組込みシステムの課題はすべて RTOS を搭載して解決するという傾向があった．たとえば割り込み処理の登録を共通化するためだけに RTOS を搭載する場合もあった．しかし RTOS の搭載により，割り込み，実行速度およびメモリなどのオーバーヘッドが増加するだけでなく，ブラックボックス部分も増えることをきちんと認識して，本当に RTOS を搭載して解決するのが最良の方法なのかを検討する必要がある．

本提案方式のように，割り込み処理間でリアルタイム性を損なわない排他制御方式の導入のみで十分な場合も多いと考えられる．

従来 RTOS を使用せず割り込み処理のみで実現していた組込みシステムのみならず，RTOS 使用していた組込みシステムにおいても，RTOS 使用によってブラックボックス部分が増加して，システムの品質が損なわれていないか検証する必要がある．

（２）現状 REMON はデバッグ機能をサポートしていない．しかしハードウェアブレークポイントを利用したデバッガ割り込みにより REMON のデバッガ処理部に制御が移るように構成して，ICB の状態にデバッグ中フラグを設ければデバッガの重要な機能であるブレークポイント機能が実現可能であると考えられる．その時にデバッグ中フラグが ON になっている ICB の CPU コンテキストを表示すればよい．REMON では，一時停止時の割り込み処理毎の情報を ICB に保存しておくため割り込み処理毎の状態の表示も実現可能と考えられる．

5.2　割り込み優先度レベルを利用した REMON の評価実験と考察

ハードウェア優先度利用の優先度継承機能を備えた割り込み処理セマフォの使用により，高優先度の割り込みに対する応答性が向上することの確認を目的として，割り込み処理のハードウェア優先度利用の優先度継承機能を備

えたセマフォ動作確認のため，DI/EI 方式による排他制御，優先度継承機能を備えていないセマフォ方式による排他制御との動作比較を行った．

さらにハードウェア優先度利用の優先度継承方式を備えたセマフォの，RTOS と比較しての性能確認が目的である．ハードウェア優先度利用の優先度継承機能を備えた割り込み処理セマフォ方式，優先度継承機能を備えていない従来の割り込み処理セマフォ方式，および RTOS のセマフォ方式の処理時間の測定を行った．

5.2.1　使用機材と実験方法

ハードウェア優先度利用の優先度継承機能を備えた割り込み処理セマフォによる排他制御方式，優先度継承機能を備えていない割り込み処理セマフォによる排他制御方式，および TOPPERS/JSP カーネル（JSP）のセマフォによる排他制御方式の性能比較のために，それぞれを同じ CPU ボードに実装して，動作の確認と処理時間の測定を行った．JSP とは TOPPERS プロジェクトにより開発・公開されている μITRON4.0 仕様[44]に準拠したオープンソースの RTOS である．なお，JSP のセマフォは優先度継承機能を備えていない．評価実験には 5.1 節で使用したものと同じ CPU ボードを使用した．またほぼ同じ仕様の REMON モニタも使用した．

処理時間の測定には，測定誤差が少なく，測定値に変動がなく，細かい時間まで測定可能な 5.1 節で使用したのと同じロジックアナライザを使用した． 800MHz で信号のサンプリングを行ったので，最少測定単位は 0.25ns である．

5.2.2　割り込み優先度利用優先度継承セマフォの機能確認

3 つの割り込み処理の動作を REMON モニタで表示することにより動作の確認を行った．図 5-6(a)に実験時の 3 つの割り込み処理の割り込みと動作の時間を示す．最初に CPU の内蔵タイマからの低優先度の割り込みにより割り込み処理 C が起動される．割り込み処理 C は実行を開始して 1.5 秒後に DI または P 操作により排他制御を開始する．割り込み処理 C は 1.5 秒の排他制御区間実行後に EI または V 操作により排他制御区間を終了する．その後さらに 1.5 秒処理を実行して終了する．

低優先度の割り込み発生後 3.5 秒経過してから内蔵タイマからの中間優先度の割り込みにより割り込み処理 B が起動され，2 秒実行して終了する．

低優先度の割り込み発生後 2 秒経過してから内蔵タイマからの高優先度の割り込みにより割り込み処理 A が起動される．割り込み処理 A は実行を開始

して 1 秒後に DI または P 操作により排他制御を開始して，1 秒の排他制御区間実行後に，EI または V 操作により排他制御を終了する．その後さらに 1 秒実行後処理を終了する．

なお図 5-6(a)は各割り込み処理の実行のタイミングを示すためだけのものである．複数の割り込み処理が同時に実行されているように見えるが，現実には同時に実行される割り込み処理は一つであり，図のタイミングで割り込み処理が実行されることはない．

図 5-6(a)で示されたタイミングに従って，各方式の排他制御を行った時の割り込み処理の動作を REMON モニタにより示す．図 5-6(b)に排他制御を実施しない場合の割り込み処理の動作を示す．図 4-6(b)は割り込みが発生して割り込み処理が実行可能になるタイミングを知るために使用する．図 5-6(c)に DI および EI により排他制御を実行した時の割り込み処理の動作を示す．図 5-6(d)に優先度継承機能を備えていない REMON セマフォの P 操作と V 操作の排他制御実行時の割り込み処理の動作を示す．

最後に図 5-6(e)にハードウェア割り込み優先度利用の優先度継承セマフォを使用した REMON セマフォの P 操作と V 操作の排他制御実行時の割り込み処理の動作を示す．

5.2.3　割り込み優先度利用優先度継承セマフォの評価と考察

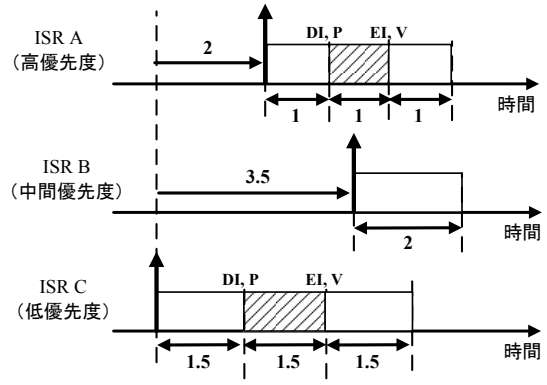
図 5-6(c)を図 5-6(b)と比較すると，割り込み処理 C が DI により排他制御を開始したのち，高優先度の割り込みが発生するが，DI が実行されているため割り込み処理 A の実行開始は遅らされる．そして割り込み処理 C が EI により排他制御を終了して，割り込み処理 A が実行を開始する．割り込み処理 A が排他制御を開始するまでは，割り込み処理 A は直ちに実行が開始されねばならないにも関わらず，開始が遅らされる．

図 5-6(d)を図 5-6(b)と比較すると，割り込み処理 C が P 操作により排他制御開始後の高優先度の割り込みにより割り込み処理 A が実行を開始する．そして割り込み処理 A も P 操作により排他制御を開始した時，割り込み処理セマフォがすでにロックされているため，待ち状態に移行させられ，割り込み処理 C が実行を再開する．割り込み処理 C の実行中に，中間優先度の割り込みにより割り込み処理 B が実行を開始して割り込み処理 C の実行をプリエンプトする．これが結果的に割り込み処理 A の再開を遅らせる優先度逆転である．割り込み処理 B が実行する間割り込み処理 A は実行を遅らされる．そのため割り込み処理 B が割り込み処理 A より先に終了しているが，これは，中間優先度の割り込み処理が高優先度の割り込み処理 A の実行を阻害するためである．

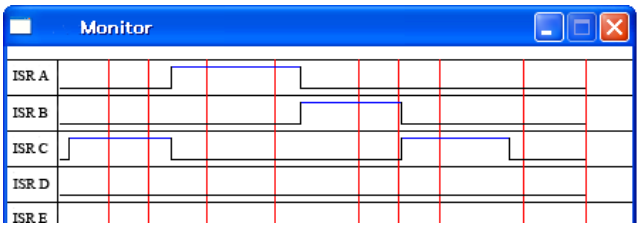
図 5-6(e)を図 5-6(b)と比較すると，割り込み処理 C が P 操作により排他制御開始後，高優先度の割り込みにより割り込み処理 A が実行を開始している．そして，割り込み処理 A も P 操作により排他制御を開始した時，割り込み処理セマフォがすでにロックされているため，待ち状態に移行させられ，割り込み処理 C が実行を再開する．割り込み処理 C の実行中に，中間優先度の割り込みが発生するが，割り込み処理 C が実行を続けることから，優先度を継承していることがわかる．割り込み処理 C が排他制御区間を終了して V 操作を発行すると割り込み処理 C は本来の優先度に戻る．その後は優先度に従って，各割り込み処理が動作する．以上の動作からハードウェア優先度利用の優先度継承セマフォが正しく動作していることが確認できる．割り込み処理 A が実行を待たされるのは，必要最小限の区間のみである．

図 5-6(c)と図 5-6(e)を比較すると，図 5-6(e)の方が割り込み処理 A の終了が早く，ハードウェア優先度利用の優先度継承方式割り込み処理セマフォ方式のほうが DI/EI 方式よりも，高優先度の割り込み応答性がすぐれていることがわかる．

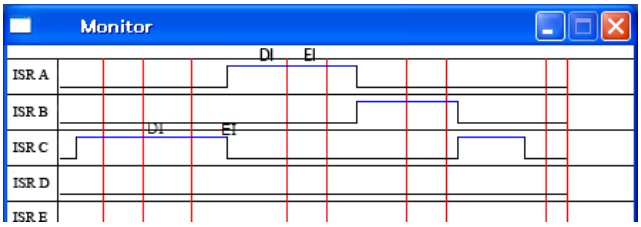
図 5-6(d)と図 5-6(e)を比較すると，図 5-6(e)の方が割り込み処理 A の再開が割り込み処理 B の実行よりも早く，さらに割り込み処理 A が割り込み処理 B



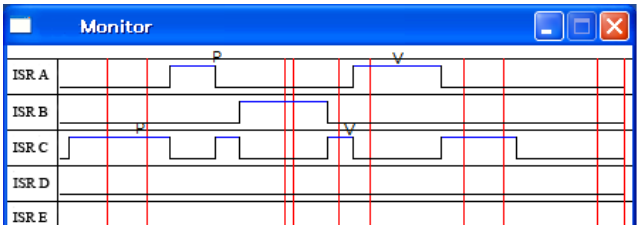
(a)　割り込み処理 のタイムチャート (単位 秒)



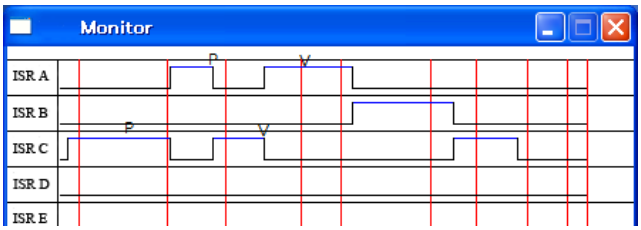
(b)　排他制御をおこなわない場合



(c)　DI/EI による排他制御



(d)　優先度継承機能のない REMON セマフォによる排他制御



(e)　ハードウェア優先度利用の優先度継承セマフォにより排他制御

図 5-6　REMON モニタによる評価

よりも先に終了しており，ハードウェア優先度利用の優先度継承機能割り込み処理セマフォ方式の方が優先度継承機能を持たない割り込み処理セマフォ方式よりも，高優先度の割り込み応答性がすぐれていることがわかる．

以上の割り込み処理の動作から REMON のハードウェア優先度利用の優先度継承割り込み処理セマフォ方式が正しく動作しており， DI/EI 方式や優先度継承機能を持たない割り込み処理セマフォ方式の排他制御と比較して，高優先度の割り込み処理の無関係な処理を遅らせることがなく，リアルタイム性にすぐれていることがわかる．

5.2.4　ロジックアナライザによる処理時間の測定

表 5-7 に示す排他制御方式（A）で表 5-8 に示す項目の処理時間（B）を測定した．（B）の項目において割り込み処理と記述している部分は JSP ではタスクを意味する．また JSP の場合は P 処理とは wai_sem の呼び出しを，V 処理とは sig_sem の呼び出しを意味する．

表 5-9 にロジックアナライザで測定結果を示す．

表 5-7　測定した排他制御の方式（A）

（A）	排他制御の方式
（１）	ハードウェア割り込み優先度を利用した REMON の優先度継承割り込み処理セマフォを使用する方式
（２）	REMON の優先度継承機能を使用しない割り込み処理セマフォを使用する方式
（３）	TOPPERS/JSP のセマフォを使用する方式

表 5-8　測定項目（B）

（B）	測定項目
（１）	割り込み処理の切り替えを伴わない割り込み処理の起動時間
（２）	割り込み処理の切り替えを伴う割り込み処理の起動時間
（３）	ロックされていないセマフォに対する P 処理の時間
（４）	解放待ちの割り込み処理がない場合の V 処理の時間
（５）	割り込み処理の切り替えを伴わない，P 処理と V 処理の組み合わせでおこなう排他制御の処理時間
（６）	割り込み処理の切り替えを伴う，P 処理と V 処理の組み合わせでおこなう排他制御の処理時間

表 5-9　REMON と TOPPERS/JSP の処理時間の測定結果（単位：μs）

処理	ハードウェアを利用した優先度継承機能を備えた REMON	優先度継承機能を持たない REMON	優先度継承機能を持たない TOPPERS/JSP Kernel
割り込み処理起動（処理切り替えなし）	0	0	33.277
割り込み処理起動（処理切り替えあり）	15.157	14.843	36.704
P 操作 (単独)（処理切り替えなし）	4.491	4.491	8.472
V 操作 (単独)（処理切り替えあり）	3.181	3.181	11.091
排他制御（処理切り替えなし）	22.468	21.865	28.510
排他制御（処理切り替えあり）	25.094	23.221	33.091

5.2.5　ロジックアナライザによる処理時間の測定結果の評価と考察

表 5-9 の測定結果から，割り込み処理の切り替えが発生しない割り込み処理起動処理は，CPU ハードウェアが割り込みをマスクするため REMON が呼び出されることはなく，ハードウェア割り込み優先度を利用する REMON の場合処理時間は発生しない．JSP の処理時間は 33.277μs である．

割り込み処理の切り替えが発生する割り込み処理起動処理は，ハードウェア優先度利用の優先度継承を使用すると優先度継承機能を使用しないより 0.314μs（= 15.157μs – 14.843μs），割合では 2%処理時間が増加する．

JSP との比較では，REMON の処理時間は 21,547μs（= 15.157μs - 36.704μs）減少する，JSP の処理時間の 41%である．

割り込み処理の切り替えが発生しない P 操作，V 操作に関しては，優先度継承機能の有無に関わらず REMON の処理時間は同じである．

P 操作と V 操作の組み合わせで実現する排他制御に関する処理時間において，割り込み処理の切り替えが発生しない場合，REMON による処理時間は，ハードウェア優先度利用の優先度継承機能を使用すると優先度継承機能を使用しないより 0.603μs（= 22.468μs – 21.865μs），割合では 3%処理時間が増加している．しかし JSP との比較では，優先度継承機能を使用する

REMONの処理時間は0.997μs（= 22.468μs - 28.510μs）減少しており，割合ではJSPの処理時間の79%である．

割り込み処理の切り替えが発生する場合，REMONによる処理時間は，ハードウェア優先度利用の優先度継承機能を使用すると優先度継承機能を使用しないより1.873μs（= 25.094μs – 23.221μs），割合では8%処理時間が増加している．しかしJSPとの比較では，優先度継承機能を使用するREMONの処理時間7.997μs（= 25.094μs - 33.091μs）減少しており，割合ではJSPの処理時間の76%である．

以上の結果から，優先度継承機能を備えたREMONは実用上問題がない処理速度を備えており，割り込み処理セマフォへの優先度継承機能追加に有効である．

5.2.6　CPUがハードウェアの優先度レベルを備えない場合の考察

ここまではCPUが割り込みに対してハードウェア優先度を持つ場合に関して述べた．しかしRISC系などCPUが割り込みに対してハードウェア優先度を持たない場合もある．

CPUがハードウェア優先度を持たない場合もREMONを利用すればICBの並び順により，割り込みに対して優先度を与えることが可能である．この場合，たとえば優先度継承時はICBを入れ替えることによって，優先度継承機能をソフトウェアのみで実現することは可能であるが，処理速度の向上は今後の研究課題である．

5.3　スタックオーバーフローに関する評価実験と考察

割り込み処理のスタックオーバーフロー検出処理の動作確認，スタックオーバーフロー検出時のスタックの再割り当て処理の動作確認，リアルタイム性の観点からの処理時間，スタックオーバーフローの検出機能追加による処理時間の増加の評価，およびスタックオーバーフローの検出機能による不具原因特定までの時間短縮の確認のため評価実験を行った．

割り込み処理のスタックオーバーフロー検出及びスタック再割り当ての評価実験にも，5.1節および5.2節と同じ機材により評価実験を行った．

本提案方式を実証するため，同じターゲットボードにスタックオーバーフ

ロー検出機能を持たない従来の REMON，スタックオーバーフロー制御機能を持つ提案方式の REMON およびリアルタイム性の比較のため TOPPERS/JSP カーネルを実装して評価実験を行った。5.1 節および 5.2 節と同じく M30620FCAFP は周波数 16MHz で動作させたため，最短命令実行時間は 62.5ns である．時間測定には POKEANA36 ロジックアナライザを使用した．ロジックアナライザは 800MHz で信号のサンプリングを行ったので最少測定単位は 0.25ns となる．

5.3.1　スタックオーバーフローに関する評価実験

表 5-10 に示す 2 種類の実験を行った．

スタックオーバーフロー検出処理が正しく動作すること，およびスタックオーバーフロー検出時のスタック再割り当て処理が正しく動作することを確認するためである．

表 5-10　スタックオーバーフロー検出及びスタック再割り当て方式の実験項目

	実験項目
（A）	多回数の再帰関数の呼び出しによるスタックオーバーフロー発生実験
（B）	大量のローカル変数宣言によるスタックオーバーフロー発生実験

表 5-11 に実験時の割り込み処理の初期スタックの容量，マジックナンバのサイズ，再割り当て時のスタック容量，スタック容量からマジックナンバが使用する容量を除いた実際にスタックとして使用できる容量を示す．

表 5-11　割り当てサイズ（単位：バイト）

実験	スタック サイズ	マジックナンバの サイズ	再割り当て スタック容量	有効な スタックの容量
（１）	128	4	256	252
（２）	4096	4	8192	8188

図 5-7 に(A)の実験に使用したプログラムを示す．階乗を求める再帰関数の多回数の呼び出しである．再帰関数は呼び出されるたびに，引数および戻り

アドレスの保存にスタックを使用する．そのため，再帰関数の呼び出しを繰り返すことにより，スタックオーバーフローが発生する．

図 5-8 に(B)の実験に使用したプログラムを示す．スタック領域以上のローカル変数を宣言するプログラムである．ローカル変数領域としては，スタックを使用する．そのため，スタック領域以上のローカル変数を宣言すると，関数実行時に，スタックオーバーフローが発生する．

結果を表 5-12 に示す．

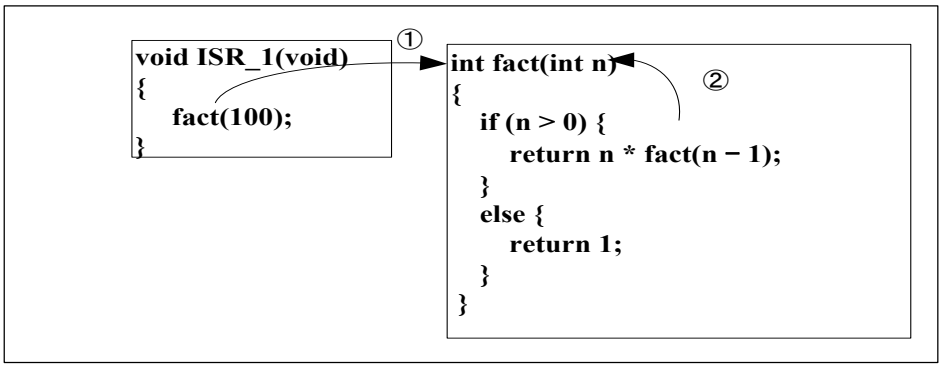


図 5-7　評価に使用したプログラム(A)

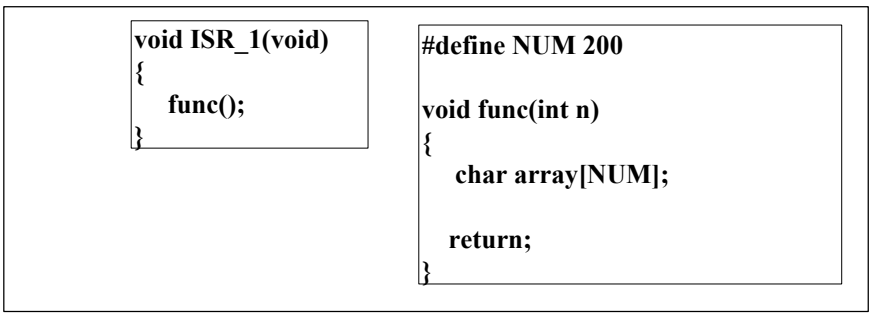


図 5-8　評価に使用したプログラム(B)

表 5-12　スタックオーバーフロー実験の結果

条件	スタック再割り当て機能なしの REMON	スタック再割り当て機能付きの REMON (マジックナンバー1 個)	スタック再割り当て機能付きの REMON (マジックナンバー4 個)
(A)-(1)	スタックオーバーフロー 20 回目に発生	スタックオーバーフロー 42 回目に発生	スタックオーバーフロー 42 回目に発生
(A)-(2)	スタックオーバーフロー 682 回目に発生	スタックオーバーフロー 1365 回目に発生	スタックオーバーフロー 1365 回目に発生
(B)	スタックオーバーフロー 発生	スタックオーバーフロー 発生せず	スタックオーバーフロー 発生せず

5.3.2　スタックオーバーフローに関する評価実験結果の考察

表 5-12 の(A)および(B)の実験結果から，従来の REMON 方式では(A)-1 では 20 回目に，(A)-2 では 682 回目にスタックオーバーフローが発生し、スタック再割り当て処理を追加した提案方式では(A)-1 では 42 回目に，(A)-2 では 1365 回目にスタックオーバーフローが発生している．

(A)の実験では図 9 の①②のどちらの呼び出しも，関数を呼び出すと PC と状態レジスタの保存のために 4 バイト，int 型の引数として 2 バイト，合計 6 バイトのスタックが使われる．(A)-(1)の場合は①最初の関数呼び出しのバイト+ 1 回の再起呼び出し 6 + 6 バイト × 20 = 126 バイトとなった時点でスタックオーバーフローが発生し，スタック再割り当て後は 6 + 6 × 42 = 258 バイトになった時点でスタックオーバーフローが発生している．(A)-(2)の場合は同じく，①6+ 6 バイト × 682 = 4098 バイトとなった時点でスタックオーバーフローが発生し，スタック再割り当て後は 6 + 6 × 1364 = 8190 バイトになった時点でスタックオーバーフローが発生している．つまり，スタックオーバーフローを検出して，スタックの再割り当てを実行している．

(B)の実験では，従来の方式では発生したスタックオーバーフローが、スタック再割り当て処理を追加した方式では発生しなくなった． つまり，スタックオーバーフローを検出して，スタックの再割り当てを実行している．

以上の 2 つの実験結果からスタックオーバーフロー検出処理，およびスタックの再割り当て処理が正しく動作することが確認できた．

評価対象のプログラムのサイズは小さいため，キャッシュを備えた CPU であれば，すべて命令キャッシュに収まると考えられる．しかし今回の評価対象はスタックというデータであり，スタックは常に新しい領域を使用していくため，たとえ CPU がデータキャッシュを備えていても，データキャッシュのデータが使用されることはない．そのため命令キャッシュの有無はスタックオーバーフロー発生までの時間には関係するが，データキャッシュの有無はスタックオーバーフローの発生回数には無関係である．

5.3.3　スタックオーバーフロー検出の処理時間の評価実験と考察

REMON を利用してスタックオーバーフロー検出機能を追加した場合の処理時間の増加を確認するために実験を行った．

表 5-13 に実験時の割り込み処理の個数，スタック，マジックナンバのサイ

表 5-13　時間測定の実験時の設定

実験条件	割り込み処理の個数	スタックのサイズ	マジックナンバのバイト数	マジックナンバの個数
	8	256 バイト	4	1 及び 4

ズ，設定したマジックナンバの個数を示す.

表 5-14 にスタックオーバーフロー検出機能を持たない従来方式の REMON，スタックオーバーフロー制御機能を持つ提案方式の REMON，そして JSP を使用した場合の割り込み処理の起動時間，セマフォ処理時間，排他制御処理時間をマジックナンバの個数 1 個と 4 個の場合で測定した結果を示す．なお JSP においては割り込み処理とはタスクを意味する.

割り込み処理間の優先度の関係により，割り込み処理の切り替えが発生する場合と発生しない場合があり，それぞれの処理時間を測定した.

初期化の時間は，設置するマジックナンバの個数が 1 個の場合は 6.654μs, 4 個の場合 9.419μs である．スタックの再割り当て処理時間は同じく 6.248μs, 6.323μs である．ある程度時間がかかることが許容される初期化，スタックの再割り当て時であり，これらは許容時間であると考えられる．スタックオーバーフロー検出による処理時間は，設置するマジックナンバの個数が 1 個の場合は約 1.2μs の増加，割合では 5%から 8%処理時間が増加している．マジックナンバが 4 個の場合は，約 2.7μs，12%から 22%の増加である.

特に処理時間が重要な組込みシステムの場合は，マジックナンバの個数は 1 個の方が適している.

割り込み処理の起動時間はスタックチェックの有無に関わらず JSP のタスク起動時間の半分以下であり，リアルタイム性の観点からも処理時間に問題はない.

セマフォを同期のために使用する場合に使用する単独の P 操作，V 操作に関しては JSP の 2 倍程度処理時間を要しており改良の余地がある．しかし P 操作と V 操作を組み合わせる排他制御実行時の処理時間は，JSP よりも 11%から 24%短縮されており，処理時間に問題はない.

表 5-14　スタックオーバーフロー機能の時間測定の結果（単位：μs）

機能	スタックオーバーフロー機能なしの REMON	スタックオーバーフロー機能を備えた REMON	スタックオーバーフロー機能を備えた REMON	TOPPERS /JSP カーネル
マジックンナンバの個数	-	1	4	-
スタックの初期化時間	-	6.654	9.419	-
スタック再割り当て時間	-	6.248	6.323	-
割り込み処理起動(処理の切り替えなし)	14.530	15.731	17.223	33.277
割り込み処理起動(処理の切り替えあり)	14.530	15.732	17.221	36.704
P 操作(処理の切り替えなし)	19.027	20.226	21.725	8.472
V 操作(処理の切り替えあり)	17.717	18.917	20.417	11.091
セマフォによる排他制御(処理の切り替えなし)	22.287	23.486	24.987	28.510
セマフォによる排他制御(処理の切り替えなし，同じ優先度の処理間)	-	-	-	28.229
セマフォによる排他制御(処理の切り替えあり)	22.287	23.486	24.987	33.091

5.3.4　スタックオーバーフローによる不具合判定時間の評価実験と考察

割り込み処理のスタックオーバーフローが原因の不具合をわざと発生させ，その原因特定までにかかる時間を測定した．不具合の原因が割り込み処理のオーバーフローと判明している場合とそうでない場合の，被験者による，不具合原因の特定までに要する時間の測定を行った．被験者は研究室の学生であり，2 名は組込みシステムの基礎の学習のみを終えた学部学生(a), (b)（以後初級被験者と記す），2 名は組込みシステム研究を 2 年間行った大学院生(c)，(d)（以後中級被験者と記す）である．

メモリ配置はスタック領域の上（下位アドレス）をグローバル変数領域としため，スタックオーバーフローが発生するとグローバル変数が破壊され不具合が発生する．デバッグには，実験に使用した OAKS16-62P BoardKi に付属の KD30 というデバッガを使用させた．これはホスト PC からの RS-232C 割り込みにより定期的に KD30 を起動させる仕組みを使用したデバッガである．両レベルの被験者の各一人には今回の提案手法を使用しない従来の方式，各一人には今回の提案方法を使用した方式で不具合原因の特定を行わせた．

今回の提案方式では，ターゲット側には割り込み処理のスタックオーバーフローが発生した時には RS-232C でホスト側に，どの割り込み処理がスタックオーバーフローを発生したかを通信した後，無限ループに入る処理をターゲット側に組み込んだ．ホスト側の PC ではターミナルソフトウェア（TeraTerm を使用）で受信した情報を表示した．

実験は次に示す（１），（２）のスタックオーバーフローを発生させるプログラムを実行させ不具合の原因を見つけるまでの時間を 5 分単位で測定した．

（１）2 種類のタイマー割り込みを利用する．優先度の低い 1 秒間隔のタイマー割り込みにより，グローバル変数領域に置いたデータにより，LED の点灯パターンを 1 秒毎に変化させる．優先度の高いイマー割り込みによる割り込み処理が，徐々にスタックオーバーフローを起こしてグローバル変数領域にある LED の点灯データを破壊する．その結果 LED の変化が不正になる不具合が発生する．

（２）前述の（１）と同じ動作を行わせるが，LED の点灯を関数で行い，その関数のアドレスをグローバル変数領域においた関数ポインタ経由で行う．優先度の高いタイマー割り込みにより割り込み処理が徐々にスタックオーバーフローを起こしてグローバル変数領域にある関数ポインタを破壊する．その結果システムダウンが発生して，LED が変化しなくなる．KD30 も実行しなくなる．

表 5-15 にこれらの不具合の原因を特定するまでに，各被験者がかかった時間を 5 分単位で示す．

(1)のケースでは初級被験者においては，従来の方法で原因の解決に 310 分かかっていたものが 90 分（29%）に，中級被験者で 130 分が 20 分（15%）

に，短縮された．(2)のケースでは初級被験者では，システムダウンによりデバッガも動かなくなったため，結局 10 時間の制限時間内では不具合の原因がわからなかった，それに対して提案手法を使用すると 210 分で原因が判

明した．中級被験者でも従来手法では 455 分かかっていたものが，35 分(8%)に短縮された．

以上のように割り込み処理のスタックオーバーフロー発生を示すことによる，不具合原因の特定時間短縮の有効性が確認された．

表 5-15　スタックオーバーフローによる不具合の原因を見つけるまでの時間（単位：分）

実験番号	スタックオーバーフロー検出機能	(a)	(b)	(c)	(d)
（１）	使用	90	-	-	20
（１）	未使用	-	310	130	-
（２）	使用	-	210	35	-
（２）	未使用	10 時間以内に発見できず	-	-	455

5.3.5　スタックオーバーフロー検出機能のメモリ容量に関する考察

表 5-16 にスタックオーバーフロー検出機能を持たない REMON 方式，スタックオーバーフロー制御機能を追加した REMON 方式，およびスタックオーバーフロー検出機能を持たない REMON 機能を組み込んでいる業務用エアコン室外機実製品におけるコードサイズとデータサイズを示す．

スタックオーバーフロー制御はコードサイズで 198 バイト，データサイズで 64 バイト，割合ではそれぞれ 16%，11%の増加である．

スタックオーバーフロー制御機能追加のためのデータの増加は SDB のみであり，増加したデータサイズは妥当である．実製品においても実装されている内蔵メモリの 0.28%（ROM），2%（RAM）に相当し，妥当なサイズであるとの評価を得た．

表 5-16　REMON のメモリ容量

機能	スタックオーバーフロー検出機能を持たない REMON	スタックオーバーフロー検出機能を持った REMON	実システム
命令のメモリ容量	1274 バイト	1472 バイト	510KB/516KB
データのメモリ容量	580 バイト	644 バイト	30KB/31KB

5.3.6　マジックナンバの偽陽性に関する考察

これまでに述べたマジックナンバを用いる方式では，スタックオーバーフローを起こして書き込んだ値が偶然マジックナンバと同じである場合は，マジックナンバが書き換えられているにも関わらず，書き替えられていないと判断するという偽陽性の問題がある．すなわち，スタックオーバーフローが発生したにも関わらずそれを検出することができない場合がある．しかし，最近の CPU には備えられている Load-Link（以後 LL），Store-Conditional（以後 SC）系の命令を使用すれば，偽陽性の問題を解決可能であると考えられる．表 5-17 に LL，SC 系の命令を備える CPU と具体的な命令を示す．

LL 命令，SC 命令は，本来はマルチ CPU 環境における，CPU 間の排他制御に使用する一対の命令である．LL 命令は，指定されたアドレスの現在の内容を返すとともに，そのアドレスのメモリへのアクセス監視を開始する命令である．その後の SC 命令は同じアドレスに，LL 命令実行後に書き込みがされてない時だけ，新しい値を書き込む．そして書き替えに成功したかどうかをプロセッサステータスワードなどで示す．つまり LL 命令，SC 命令の組み合わせで，特定のアドレスのメモリにアクセスがあったかどうかを知ることができる．

LL, SC 命令は，メモリの内容を比較するのではなく，CPU ハードウェアが，同じメモリアドレスへのアクセスがあったかどうかを判断するために，偽陽性の問題が発生しない．LL, SC 命令を用いてスタックオーバーフロー発生を検出するには，たとえば次のようにする．

マジックナンバを 2 つ用意しておき，REMON スケジューラが，これから実行する割り込み処理の第 1 のマジックナンバを格納している，スタックの終端アドレスに，LL 命令を実行する．

REMON スケジューラが，別の割り込み処理に処理を切り替える時に，今まで実行していた割り込み処理のスタックの終端アドレスに，第 2 のマジックナンバで SC 命令を実行する．SC 命令の失敗により，スタックの終端アドレスへのアクセス発生を検出した場合は，スタックオーバーフローが発生したと判定する．この方式を用いれば，マジックナンバが同じ値に書き変えられた場合でも，マジックナンバの書き換えが検出でき，スタックオーバーフローを検出できると考えられる．ただし ARM のように LL, SC 系の命令が実装依存の場合もあるため，個別の検証は必要である．

表 5-17　各種 CPU の LL/SC 系の命令

CPU	LL 系の命令	SC 系の命令
MIPS	LL	SC
PPC	LWARX	STWCX.
ARM	LDREX	STREX

第６章

おわりに

本章では，本研究により達成できた成果をまとめ，さらに今後の課題について述べる．

6.1 成果

本研究により以下の成果を実現できた．

・REMON により，割り込み処理においても，セマフォ同等機能を使用することができるようになり，割り込み処理間の排他制御におけるリアルタイム性の向上を実現できた．

・CPU ハードウェアが備える割り込み優先度を利用した，割り込み処理用の優先度継承セマフォの処理時間の短縮を実現できた．

・MMU を使用しない組込みシステムにおいても，REMON を使用した提案方式により，割り込み処理のスタックオーバーフローの検出，およびスタックの再割り当てが実現できた．また，スタックオーバーフローの検出は再現性の低い不具合の原因特定に有効であることを確認した．

・REMON では，割り込み処理とタスクが一体化されるため、システム設計が単純化される．REMON を使用することで高品質の組込みシステムの設計が容易になると期待できる．

以上のように本研究により，組込みシステムにおけるリアルタイム性および品質の向上が期待できる．

6.2　今後の課題

本研究の主な対象は RTOS を使用せず，割り込み処理のみで処理を行っている組込みシステムである．しかし RTOS 使用している組込みシステムにおいても，RTOS 使用によってブラックボックス部分が増加して，システムのリアルタイム性が損なわれている可能性がある．

組込みシステム特有のリソース制約などの条件を満たしつつ，リアルタイム性と品質を両立させることが今後の研究課題である．

また，割り込み処理用のデバッガの実現，CPU がハードウェア的に割り込み優先度を持たない場合の優先度継承割り込み処理セマフォの処理速度の向上，ロードリンク命令，ストアコンディショナル命令を使用した MMU を使用しないスタックオーバーフロー発生の検出方式の実現も今後の研究課題である．

謝辞

本研究を進めるにあたり，適切なご助言とご指導を賜り，また多くのご支援 を頂戴いたしました九州大学大学院システム情報科学府 福田 晃教授に深く感 謝の意を表します.

本論文をまとめるにあたり，貴重なる御助言とご指導を賜りました九州大学大学院システム情報科学府 村上 和彰教授ならびに鵜林 尚靖教授に心より感謝の意を表します.

東京電機大学 小 泉 寿 男 教授には，福田教授とともに論文の作成において細部にわたるご指導を頂き，様々なご助言をいただきましたことに深く感謝いたします.

三菱電機株式会社時代，助言協力を頂いた吉田 利夫氏，竹垣 盛一氏，田中 輝明氏， 南出 英明氏， 藤本 堅太氏， 井上 禎一郎氏に感謝いたします.

三菱電機メカトロニクスソフトウエア株式会社において，協力をいただいた岩橋正実氏，川上 敏弘氏に感謝いたします.

大阪電気通信大学において，論文作成の応援をいただいた登尾 啓史教授に感謝いたします.

様々な面で協力していただいた，九州大学 大鶴 陽子さんに感謝いたします.

また，本論文をまとめるにあたり，様々な協力をしてくれた大阪電気通信大学組み込みリアルタイム研究室（南角研）の学生諸君に感謝します.

最後に私を支えてくれる妻 知美， 子 哲俊，悠佳に感謝します.

参考文献

[1] 高田広章：「組込みシステム開発技術の現状と展望」，情報処理学会論文誌，Vol.42, No.4, pp.930-938, 2011.

[2] 中 森 章：「マイクロプロセッサ・アーキテクチャ入門」，CQ 出版，Interface 増刊，2004.

[3] J ulio Sanchez and Maria P. Canton：”Embedded Systems Circuits and Programming,” CRC Press, 2012.

[4] Phillip A. Laplante and Seppo J. Ovaska：”Real-Time Systems Design and Analysis -4th Edition,” Wiley IEEE Press, 2012.

[5] Alan Burns and Andy Wellings：”Real-Time Systems and Programming Languages,” Addison-Wesley, 1997.

[6] Giorgio C. Buttazzo：”Hard Real-Time Computing Systems,” Springer, 2011.

[7] 南角 茂樹：「組込みシステムにおける応答性能の保証」，システム/制御/情報学会 システム/制御/情報，Vol.51, No.9, pp.388-392, 2007.

[8] Dan Ionescu and Aurel Cornell：”Real-Time Systems Modeling, Design, and Applications,” World Scientific, 2007.

[9] Jack Ganssle：”The Art of Designing Embedded Systems – 2nd Edition,” Elsevier, 2008.

[10] 南角 茂樹：「組み込みシステムとリアルタイム OS」，CQ 出版, Interface, Vol.37, No.4, pp.46-51, 2011.

[11] Dan Ionescu and Aurel Cornell：”Real-Time Systems Modeling, Design, and Applications,” World Scientific, 2007.

[12] James F. Readya：”A Real-Time Operating System for Embedded Microprocessor Applications,” IEEE Micro, Vol.6, No.4, pp.8-17, 1986.

[13] John A. Stankovic and Krithi Ramamritham：”The Design of the Spring Kernel,” Proc. of Real-Time Systems Symposium, pp.146-157, 1987.

[14] 経済産業省 (独)情報処理推進機構：「2010 年版組込みソフトウェア産業実態調査報告書」, 2010.

[15] 経済産業省 (独)情報処理推進機構：「2009 年版組込みソフトウェア産業実態調査報告書」, 2009.

[16] 経済産業省 (独)情報処理推進機構：「2008 年版組込みソフトウェア産業実態調査報告書」, 2008.

[17] Phillip A. Laplante and Seppo J. Ovaska：”Real-Time Systems Design and Analysis – 4th Edition,” Wiley IEEE Press, 2012.

[18] Giorgio C. Buttazzo：”Hard Real-Time Computing Systems,” Springer, 2011.

[19] Ed Lipiansky : ”Embedded Systems Hardware for Software Engineers,” Wiley IEEE Press, 2012.

[20] Marilyn Wolf : ”Computers as Components – Princeples of Embedded Computing System Design 3rd Edition”, Elsevier (2012)

[21] 南角 茂樹：「Web 連載 ウインドリバースクエア-南角先生の組込み講座」http://www.windriver.com/japan/web_magazine/nankaku/index.html，2011-2013.

[22] VxWorks プログラマーズガイド 5.4 DOC-13006-ZD-00, Wind River Systems Inc. ,2000.

[23] Andrew S. Tanenbaum : ”Modern Operating Systems – Third Edition,”, Prentice Hall, 2008.

[24] 南角 茂樹，水篠 公範，小泉 寿男，福田 晃：「組込みシステム用割り込みスケジューラ REMON」，電気学会論文誌(電子・情報・システム部門誌), Vol.133, No.2, pp.316-325, 2013.

[25] D.A.Patterson and J.L Hennesy : ”Computer Organaization and Design,” Addison Wesly, 1996.

[26] M.K.McKusick, K.Bostic, M.J.Karels, and J.S.Quareterman : ”4.3BSD Operating System,” Elsevier,2012.

[27] D.A.Patterson and J.L.Hennesy : ”Computer Organaization and Design”, Elsevier, 2012.

[28] Jane W.S. Liu：”Real-Time Stsyems,” Prentice Hall, 2000.

[29] 坂田 史郎：「ユビキタス技術 センサネットワーク」，オーム社, 2006.

[30] Jason Hill, Robert Szewczyk, Philip Levis, Sam Madden, Cameron Whitehouse, Joseph Polastre, David Gay, Cory Sharp, Matt Welsh, Eric Brewer, and David Culler：”TinyOS: An Operating System for Sensor Networks,”Ambient Intelligence, pp.115-148, 2004.

[31] Supra Biswas, Thomas Carley, Matthew Simpson, Bhuvan Middha, and Rajeev Barua : “Memory Overflow Protection for Embedded Systems Using Run-Time Checks, Reuse, and Compression,” ACM Transactions on Embedded Systems, Vol.5, No.4, PP.719-752, 2006.

[32] C.Cowan, C.Pu, D.Maier, J.Walpole, P.Bakke, S.Beattie, A.Grier, P.Wagle, Q.Zhang, and H.Hinton : ”Stack-guard : Automatic adaptive detection and prevention of buffer-overflow attacks,” 7th Usenix Security Symposium, pp.63-78, 1998.

[33] O.Ruwase and M.S.Lam : “A practical dynamic buffer overflow detector,” Proc. of the 11th Annual Network and Distributed Systems Security Symposium, pp.159-169, 2004.

[34] 中嶋健一郎，本田晋也，手嶋茂晴，高田広章：「セキュリティ支援ハードウェアによるハイブリッド OS システムの高信頼化」，情報処理学会研究報告 EMB, 組込みシステム, pp.1-pp7, 2008.

[35] 中嶋健一郎，山田真大，長尾卓哉，山崎二三雄，武井千春，本田晋也，高田広章：「ARMv6 アーキテクチャを用いたメモリ保護 RTOS のユーザスタック保護の設計と評価」，情報処理学会研究報告 EMB, 組込みシステム, pp.1-11, 2009.

[36] 石川拓也，本田晋也，高田広章：「静的なメモリ配置を行うメモリ保護機能を持ったリアルタイム OS」，日本ソフトウェア科学会論文誌（コンピュータソフトウェア），Vol.29, No.4, pp.161-181, 2012.

[37] Yuan-Cheng Lai, Ying-Dar Lin, Fan-Cheng Wu, Tze-Yau Huang, and Frank C.Lin : “Embedded TaintTracker: Lightweight Run-Time Tracking of Taint Data against Buffer Overflow Attacks,” IEICE Transactions on Information and Systems Vol.E94.D, No.11, Year , pp. 2129-2138, 2011.

[38] Amit Vasudevan and Ramesh Yerraballi : “Stealth Breakpoints”, Proc. of the 21th Annual Computer Security and Applications Conference, pp.381-392, 2005.

[39] 南角 茂樹，川上 博行，小泉 寿男，福田 晃：「ハードウェア割り込み優先度を利用した割り込み処理の優先度継承セマフォの実現方式」，電気学会論文誌（電子・情報・システム部門誌）, Vol.133, No.11, pp.2053-2061, 2013.

[40] 南角 茂樹，川上 博行，小泉 寿男，福田 晃：「割り込みスケジューラ REMON のスタックオーバーフローの制御機能」，電気学会論文誌（電子・情報・システム部門誌）, Vol.133, No.8, pp.1509-1520, 2013.

[41] 岩橋 正実・川上 敏弘・松井 賢治・井上 禎一朗・南角 茂樹：「REMON (Real-Time Embedded Monitor) の開発」, MSW 技法, No.20, pp.47-51, 2007.

[42] Wei Zhao, Krithi Ramamritham, and John A. Stankovic：“Preemptive Scheduling Under Time and Resource Constraints,” IEEE Trans. Computers, Vol.36, No.8, pp.949-960, 1987.

[43] Joseph Y-T. Leung and Jennifer Whitehead：“On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks,” Performance Evaluation, Vol.2, No.4, pp.237-250, 1982.

[44] 坂村 健：「μITRON4.0 標準ハンドブック」，パーソナルメディア , 2001.

本研究に関する著者の発表論文

1. Shigeki Nankaku, Keiji Asada, Hisao Koizumi, and Akira Fukuda : “The Stack Overflow Control Mechanism with the Interrupt Scheduler REMON,” Proc. of the 2012 International Conference on Embedded Systems and Intelligent technology (ICESIT'12), pp.117-119, 2012 .

2. 南角 茂樹，水篠 公範，小泉 寿男，福田 晃：「組込みシステム用割り込みスケジューラ REMON」，電気学会論文誌（電子・情報・システム部門誌），Vol.133, No.2, pp.316-325, 2013.

3. Shigeki Nankaku, Jun Sawamoto, Hiroyuki Kawakami, Hisao Koizumi, and Akira Fukuda : "Development and Evaluation of an Interrupt Scheduler using CPU Hardware Interrupt Priority Levels," Proc. of the 2nd International Conference on Systems, Control, Power, Robotics (SCOPRO'13), pp.151-156, 2013.

4. Shigeki Nankaku, Hisao Koizumi, and Akira Fukuda : "Strengthening Interrupt Controls in Embedded Systems by Cooperation between Windows CE and REMON," Proc. of the 2013 International Conference on Software Engineering Research & Practice, pp.419-425, 2013.

5. 南角 茂樹，川上 博行，小泉 寿男，福田 晃：「割り込みスケジューラ REMON のスタックオーバーフローの制御機能」，電気学会論文誌（電子・情報・システム部門誌）, Vol.133, No.8, pp.1509-1520, 2013.

6. 南角 茂樹，川上 博行，小泉 寿男，福田 晃：「ハードウェア割り込み優先度を利用した割り込み処理の優先度継承セマフォの実

現方式」，電気学会論文誌（電子・情報・システム部門誌），
Vol.133, No.11, pp.2053-2061 , 2013.

本研究に関する著者の取得特許

１． 日本：南角茂樹，井上禎一郎，岩橋正美，川上敏弘：「リアルタイム組込み簡易モニタプログラム」，特許登録番号 4068106

２． 米国：Nankaku Shigeki; Inoue Teiichiro; Iwahashi Masami; Kawakami Toshihiro：”Real-time Embedded Simple Monitor Method and Computer Product,” United States Patent NO. 7,472,214

３． ヨーロッパ：Shigeki Nankaku; Teiichiro Inoue; Masami Iwahashi; Toshihiro Kawakami：” Method of Real-Time Embedded Simple Monitor,“ Bibliographic data: HK1101434 (A1)

４．中国：南角茂树；井上祯一郎；岩桥正实；川上敏弘：「实时内部简易监视器」，申请号 200610006414.0

５． 台湾：南角茂樹 Nankaku, Shigeki ;井上禎一郎 Inoue, Teiichiro ;岩橋正實 Iwahashi, Masami ;川上敏弘 Kawakami, Toshihiro：「即時嵌入式簡易監視方法、即時嵌入式簡易監視程式產品以及電腦可讀取即時嵌入式簡易監視程式的記錄媒體」“Real-Time Embedded Simple Monitor Method, Real-Time Embedded Simple Monitor Program Product and Real-Time Embedded Simple Monitor Program by Computer-Readable Storage Medium,” 證書号 I306216