システムソフトウェアと
オペレーティング・システム 82－5
（1999．8．2）

# 組込み機器向けOSの移植性を考慮した割込み機能の抽象化

砂川克志　岡本康明　最所圭三　福田晃

奈良先端科学技術大学院大学
情報科学研究科
〒630-0101 奈良県生駒市高山町 8916-5
*E-mail*:{katusi-s, yasua-o, sai, fukuda}@is.aist-nara.ac.jp

従来，組込みシステムにおいては使用するプロセッサ毎に特化した組込みプログラムが作成され，搭載されていた．このようなプログラムの生産性，再利用性は低く，そのため，ITRONに代表される組込みOSが開発されている．しかしながら，これらの組込みOSはハードウェアに特化して作られることが多く，OSそのものの移植性は必ずしも高くない．特に割込みに関しては，使用するプロセッサに大きく依存するので，移植の際に非常に大きな問題になる．そこで，割込み機能の抽象化を考え，どのようなプロセッサを使用するかをユーザに意識させないで開発できるインタフェースを提案する．割込み処理中のレジスタ情報などを保持する割込み構造体や，多重割込みを管理するデータ構造である割込みスタックを提案する．さらに，これらの割込み機能のインタフェースの適用例を示し，考察する．

# Definition of an Interface of Interrupt Handler for Easier Porting on Embedded Systems

Katsushi Sunagawa　Yasuaki Okamoto　Keizo Saisho　Akira Fukuda

Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5Takayama, Ikoma, Nara 630-0101, Japan

Programs for embedded systems are made depending on target processors and loaded into embedded systems. Operating systems for embedded systems such as ITRON have been developed to raise reusability of source cords and productivity of developing software. However, since even these operating systems are structured for specific target processors, portability of the systems is not so much. Interrupt particularly influences portability because management of interrupt includes operations depending on processors very much. Thus, an interface, which abstracts function of interrupt and makes uses develop programs for embedded systems with no awareness of that what processor is used, is proposed. Interrupt structures and an interrupt stack are also proposed. Interrupt structures keep information of context of interrupt handler such as contents of registers, and interrupt stack manages nested interrupts. Moreover, applications of the interface are shown and discussed.

# 1 はじめに

組込みシステムの機能がより多様化，複雑化，大規模化するにつれ，そのソフトウェアの生産性の向上が重要な課題となり，組込みシステム向けOSが必要になってきた．組込みシステムは，製造コストを下げるためにメモリの制約が特に厳しく，またリアルタイムシステムとしての機能が求められている．そのような要請に答えるべく，ITRONなどの組込みシステム向けOSが開発されている．

割込み処理はレジスタの退避やCPU依存の特定のアドレスへの分岐などを伴う．例えば，割込み発生時まで処理していたコードのプログラムカウンタの内容やスタックポインタの内容，必要な汎用レジスタの内容の保存など，CPU固有の操作が必要とされる．このため，既存の組込みシステムでは，割込み処理の大部分がプロセッサに依存する実装がなされている．組込みシステム用のOSにおいては，割込み機能に関する部分はCPU依存のコードが極めて多くなり，移植性に問題が出ている．ITRONの割込み仕様に関しても，実装依存として明確に定義していない部分が多く，OS自体の移植性は決して高いとはいえない．組込み機器は多様なCPUの上で動作するので，移植性の向上とコードの再利用による生産性に大きな影響を与える．

本研究においては，組み込みOSの割込み機能のコードの移植性と再利用性の向上を目的としている．そのため，割込み機能の抽象化によりこれらの目的を達成しようと考えている．具体的には，ハードウェアに依存する部分と，独立した部分の切り分けを行ない，その間のインタフェースを定義する．一方，組込みOSにはリアルタイム性が要求されている．このため，割込みの抽象化を行うとき，プロセッサのアーキテクチャの特性をできるかぎり生かし，OSの応答速度を落さないことが重要となる．このように，ハードウェア独立部を明確にし，インタフェースを定義することにより，このインタフェースを介してOSを高級言語で書くことができるようになる．この結果，コードの移植性および再利用性が向上すると同時に生産性も高まると考えている．

以後，第2節では割込み機能を抽象化するインタフェースの概要と考慮すべき問題点を述べ，第3節では，割込み機能のインタフェースの定義を行なう．第4節では，2種類のプロセッサにおける割込み処理の具体例とそれに関する考察を述べる．

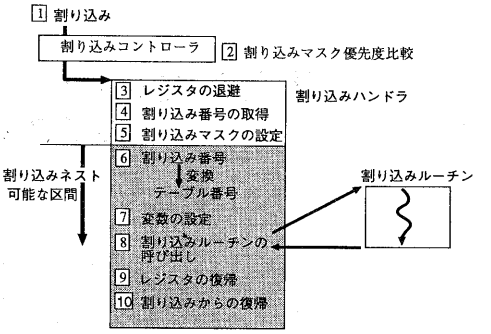


図 1: SH3の割込み処理の過程

# 2 割込み処理の概要と考慮すべき問題点

まず，日立SH3プロセッサにおける割込み処理の流れを説明する．次にSH3と他のプロセッサの割込み処理に関する相違点を指摘し，割込みインタフェース設計に向けての留意点を述べる．

日立SH3プロセッサにおける典型的な割込み処理の流れを図1に示す．以下図中の数字に対応して処理手順を示す．

1 外部機器からの割込み要求の発生

2 割込みコントローラでは，当該割込みの優先度より高い割込みが同時に入っていない，当該割込みがマスクされていない，の2つの条件が共に成り立った場合CPUに当該割込みを伝達する．

3 CPUのステータスレジスタとプログラムカウンタの内容が退避される

4 CPU固有の割込みコードがレジスタにセットされる．

5 割込みマスクがレジスタにセットされる．

6 割込み番号から割込み管理テーブルのオフセットを計算する

7 割込みハンドラに予め用意している割込みルーチン (ISR) のアドレスを設定する.

8 7で設定したアドレスを呼び出し，ISR内で処理を行なう

9 3で退避したレジスタを復帰する.

10 割り込まれる直前の処理に戻る.

次に，SH3と他のプロセッサにおける割込み処理の相違点を明らかにし，割込みインタフェースの設計に関する留意点を挙げる．以下の考察では，日立のSH1，SH2，SH3，Intel x86系プロセッサ，Strong ARMを念頭に置いている.

**インタフェース設計に関する留意点**

(1) 割込みマスクの制御
2の割込みコントローラでの割込み優先度およびマスクの方法は，割込みコントローラ毎に異なる．例えば，SH3では，割込みの優先度を各要因に対して0から15を指定でき，優先度が0の場合はマスクされる．これに対してIntelの割込みコントローラ8259Aでは，優先度とマスクは別々のレジスタを用いて制御される．この違いを吸収しなければならない.

(2) ISRの出口と入口でのレジスタの保存，復帰の方法の標準化
SH3，およびStrong ARMプロセッサは割込み用のレジスタ群を持っており，自動的にハードウェアが切替えるので，多重割込みが無い場合は明示的にレジスタを保存する必要はない．しかし，Intel x86系プロセッサではユーザが使用するレジスタを明示的に保存しなければならない．この違いを吸収する必要がある.

(3) 割込みベクタおよびISRへの分岐の管理方法の標準化
前述のように，SH3では複数の割込みに対して，例外要因判定ルーチンに分岐する．Strong ARMも同様である．SH1，SH2およびIntel x86系プロセッサでは，割込み要因とハードウェアで実現する割込みベクタによる分岐先が1対1に対応している.

(4) 割込み禁止，割込み許可の実現方法
SH1，SH2，SH3およびStrong ARMではSRレジスタに設定された優先度より大きな優先度を持つ割込みのみ受け付ける．これに対して，Intel x86系プロセッサでは，すべての割込みを受け付けるか受け付けないかの2つしか選べない．このため，他のプロセッサと同様に割込み優先度に応じて割込みを受け付けるかどうかを選択するには，割込みコントローラと協力しなければならない．例えば8259Aを用いる場合，8259Aのマスクレジスタの受け付けたい割込みに対応するビットをクリアする.

(5) 多重割込み時の処理管理のインタフェースの標準化
多重割込みに対応する際，いずれのプロセッサも処理中の割込みのコンテキストを保存するため，ユーザが使用するレジスタを退避しなければならない．この部分を共通化する方法として，全てのレジスタを退避することが考えられる.

(6) 割込み要因の識別および割込みハンドラの標準化
SH3においては，例外処理ルーチンの先頭に一度ジャンプし，Strong ARMプロセッサは割込み用の特定の処理コードにジャンプする．一方，SH1，SH2，およびIntel x86系プロセッサは，ハードウェアによって用意されたベクタ機能により，割込み要因と1対1に対応したアドレスに自動的に分岐する．このため，割込みハンドラを要因別に分けるか否かという問題が生じる.

割込み処理を行なう場合，割込み機能を，プロセッサ，外部ハードウェア，ソフトウェアでどう分担するかが問題となる．またそれに応じて，多くのインタフェースの定義が考えられる．次節ではこれらの留意点を考慮して，移植性向上を目指したインタフェースの定義を行なう.

## 3 割込み機能を抽象化するインタフェースの定義

本節では前節で挙げた留意点を考察して，移植性およびユーザの利便性を向上するため，OSレベ

ルでの割込みインタフェースを定義する．以下の設計方針に従って割込みインタフェースを定義する．

- OS 側でインタフェースを標準化する
- 組込みシステムのハードウェアを OS のインタフェースに合わせて設計する

移植性向上という目的を達成することが上記の設計方針の第 1 の理由である．第 2 の理由は，組込みシステムのハードウェアは，システム機能に応じて設計されることが多いということである．

留意点 (1) について述べる．本研究では，上述の OS の設計方針より，組込みシステムの割込みコントローラは，マスク中の外部割込みが失われないものとする．即ち，ある優先度の割込みを処理中に別の低い優先度の外部割込みが発生すると，割込みコントローラによって外部割込みは保留される．そして，保留された外部割込みより高い優先度の割込みの処理が終了してから，保留された割込み信号のマスクが外され，割込み信号が CPU に到達する．また，受け付けた割込み要因は割込みコントローラで識別できるものとする．一般のコントローラではこれらの条件は満足されている．この仮定により，割込みの優先度マスクに関する処理はすべて外部または CPU 内部モジュールとして提供される割込みコントローラで処理することとなる．

次に留意点 (2) および (3) について併せて述べる．SH3 および Strong ARM では，OS の機能として割込み要因判定ルーチンを設け，割り込み要因判定ルーチンが割込みコントローラの割込み管理レジスタを見にいくことで対応する．よって，SH3 および Strong ARM では，1 つの割込みハンドラで割込みを受け付け，割込み要因判定ルーチンで割込み要因を判定し，割込み要因に対応した ISR を呼び出すようにする．一方，SH1，SH2，および Intel x86 系プロッセッサでは，実現方法が異なる．これらのプロセッサでは，ハードウェアで実現するベクタに従って分岐するので，SH3 や Strong ARM と同様の処理を行なうためには，分岐先において割込み要因を取得し，次にすべての分岐から割込み要因判定ルーチンにジャンプする．これにより，SH3 の割込み処理の概要 [4] で述べた割込み番号の取得が可能になる．後は SH3 および Strong ARM と同様の処理を行なうことができる．

ここで ISR の形態として，Managed ISR と Unmanaged ISR を定義する．Maneged ISR は ISR に分岐する前に OS が割込みコントローラの制御やすべてレジスタを管理する．これにより，ISR を高級言語で記述できるようになり，移植性が向上する．また，Unmanaged ISR においては，OS は各 Unmanaged ISR で共通化できる部分以外には関与しないで，ユーザの責任でレジスタの退避，復帰などを行なう．SH1，SH2，および Intel x86 系プロセッサやではハードウェアの特定のベクタに従ったの分岐先を Unmanaged ISR とすることで，ハードウェアの高速性を活かすことができる．しかし Unmanaged ISR においては，割込みコントローラなどの制御，一部のコードのアセンブラによる記述などを含むことが多く，移植性が低下する．

留意点 (4) について述べる．割込み番号が取得できれば，割込み要因に対応する割込み番号と，割込みに対する目的の処理を記述する ISR のエントリを 1 対 1 で対応付けることができる．この 1 対 1 の対応づけは，割込み管理テーブルとして，OS 側でソフトウェアにより管理する．割込みテーブルには，割込み番号と，当該割込みの分岐先のエントリ，分岐先の ISR の属性を含むようにする．分岐先の属性には，当該 ISR が Managed ISR であるか Unmanaged ISR であるかを指定する．Managed ISR の場合は特別に用意したコンテキスト保存ルーチンにジャンプし，レジスタを全て保存した上で ISR に移行する．このようにインタフェースを分けることにより，高速処理を望むユーザのニーズに答えつつ，移植性の向上に寄与することができる．(図 2)

留意点 (5) について説明する．本インターフェースにおいては，割込みを受け付中には，それより高い優先度を持つ割込みのみ受け付けるようにする．SH1，SH2，SH3 および Strong ARM では，SR レジスタに処理中の割込みの優先度を設定することにより実現できる．また，ある割込みが受け付けられた場合，その割込みの優先度が自動的に SR レジスタに設定されるので，特別な処理は不要である．しかし Intel x86 系では，割込みコントローラと協力しなければ実現できない．Intel x86 系プロセッサでは，割込みが入った場合は，割込み禁止になっているので，割込みコントローラの割込みマスクレジスタの受け付けたい割込み以外のビットをマ

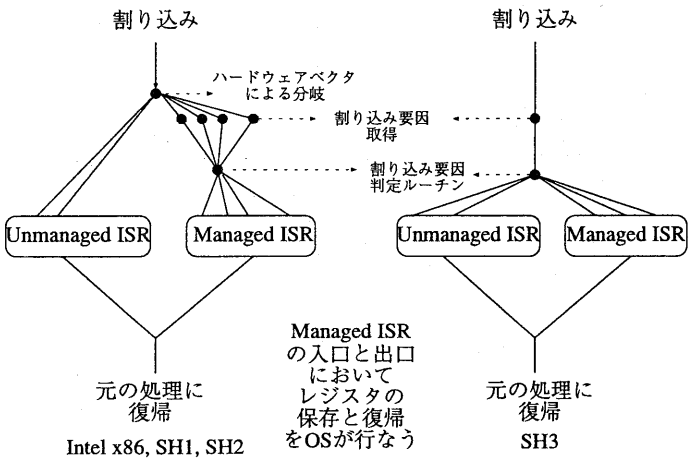


図 2: 割込み要因の取得と ISR

スクした後割込みを可にする．

最後に留意点 (6) について説明する．これまでの組込み OS では，多重割込みを処理する際， ISR が様々なレジスタの退避，復帰を意識しなければならなかった．本研究では，割込み構造体と割込みスタックを提案する．これらを用いて以下の手順で多重割込みを処理する．割込み処理中に割込みが発生した場合，割込み構造体のアドレスを割込みスタックのエントリに保持する．そして，処理中の割込み処理が終れば，自動的に OS が割込みスタックに割込みタスクが残っているかどうかを調べる．そのタスクが残っていれば，その割込みタスクを処理する．もし，割込みスタックが空であれば，割り込まれた元のコードに復帰する．

ユーザは多重割込みを許可する場合でも，ISR を高級言語記述できる．OS は割込みスタックの情報を参照することにより，割込み処理終了時に多重割込みの有無を知ることが出来る． 割込み構造体と割込みスタックの詳細は以下の通りである．

OS は，多重割込みが生じた際，割込み構造体を割込みスタックに積む．割込み構造体は，割込み時に使用するすべてのレジスタ情報と，当該割込みを処理する ISR の属性を保持する構造体である．割込み構造体は，後述のように割込みプロシージャ，割込みタスクの 2 つの形態を取る． 割込みスタックの各エントリは，割込み構造体と割込み構造体の ID を保持する．ISR 属性には，当該割込みに対応する ISR が Unmanaged ISR か，Managed ISR かを指定する．Managed ISR の場合は OS は当該処理の ISR を呼び出す前に，すべてのレジスタの退避を行なうが，Unmanaged ISR の場合はレジスタの退避を行なわない．また，多重割り込み実現方法として，割込み構造体には 2 通りの実装法が考えられる．割込みタスクとするか，割込みプロシージャの形とするかである．(図 3)

- 割込みプロシージャ
  割込みプロシージャは割り込まれたタスクに割り当てられたスタック領域を使用する．もし割込みプロシージャの実行中に割込みが発生した場合，次の割込みプロシージャによってスタックがさらに深くなってしまう．従って，割込まれる可能性のあるすべてのタスクは，割込みプロシージャのネストが最も深くなってもあふれない大きさのスタックを持つ必要がある．このため，割込まれる可能性のあるタスクに対して全体としては無駄なメモリを割り当てることになる可能性もある．ただし，スタック切替の際のレジスタの保存，復帰の点で，後述の割込みタスクより高速となる．

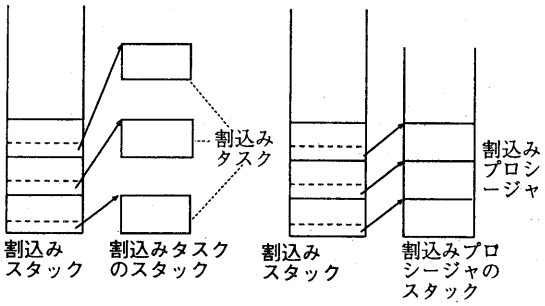


図 3: 割込みタスクと割込みプロシージャ

- 割込みタスク
  割込みタスクは，割り込まれたタスクとは別のスタックを用意する．従って，自タスクが使用するだけのスタック領域を保持すれば良い．一方，スタック切替を必要とするので，レジスタの退避，復帰の時間が割込みプロシージャより長くなる．

ISR が終了すれば，OS は割込みスタックを調べる．割込み構造体がスタックになければ，割込み処理を終了する．現在の ISR が Managed ISR か Unmanaged ISR かを調べる．Manageded ISR であれば全ての使用したレジスタを保存してから，割り込まれる前のコードに復帰する．

# 4 割込み機能インタフェースの適用と考察

## 4.1 日立 SH3 への適用

日立の SH3 のレジスタ構成は，システムレジスタとコントロールレジスタ，汎用レジスタから成る．まず，割込み処理に関わるレジスタを説明する．システムレジスタにはプログラムカウンタ (PC) が含まれる．コントロールレジスタにはステータスレジスタ (SR) がある．また，特権モードのみでアクセスできる退避ステータスレジスタ (SSR)，退避プログラムカウンタ (SPC)，ベクタベースレジスタ (VBR) がある．SR には割込みを CPU 側でブロックするためのブロックビット (BL) と，プロセッサの特権モード，ユーザモードを切替えるモードビット (MB) がある．汎用レジスタとして R0 ～ R15 までの 16 本のレジスタがある．汎用レジスタ R0 ～ R7 はバンクレジスタで，SR のレジスタバンクビット RB が 1 の時バンク 1 の汎用レジスタ R1_BANK0 ～ R1_BANK7 と，バンクに関係ない R8 ～ R15 の合計 16 のレジスタを汎用レジスタとして使用可能であり，バンク 0 の汎用レジスタ R0_BANK0 ～ R0_BANK7 は LDC/STC という命令を用いてアクセスする．ユーザモードの時は，R0_BANK0 ～ R1_BANK7 を R0 ～ R7 として，加えて R8 ～ R15 のレジスタの合計 16 個のレジスタを汎用レジスタとして使用できる．ただし R1_BANK0 ～ R1_BANK7 はユーザモードでは使用できない．以下に，SH3 における割込み処理の具体例を挙げる．

1. PC と SR の内容がそれぞれ SPC と SSR に退避される
2. SR のブロックビット (BL) が 1 に設定される (BL=1 の時，割込み要求をマスク．ユーザブレーク以外の一般例外が発生するとプロセッサはリセット状態へ移行する)
3. SR のモードビット (MB) が 1 に設定される
4. SR のレジスタバンクビット (RB) が 1 に設定される
5. 例外コードが例外事象レジスタ (EXPEVT) または割込み事象レジスタ (INTEVT) のビット 11-0 に書き込まれる
6. 割込み要因を判定するため，INTEVT レジスタの値をオフセットとして各割込み要因の割込み処理ルーチンに分岐する．

7. 各割込み処理ルーチン中で，該当割込み要因をクリアする．

8. SPC，SSR および保存すべきすべてのレジスタを割込みタスク構造体として退避し，当該割込み構造体の情報を割込みスタックに格納する．

9. 多重割込みを許可するため，SR の BL ビットをクリアする．

10. ベクタベースレジスタ (VBR) にソフトウェア設定したアドレスと 0x00000600 の和である例外処理ルーチンの先頭へ一旦ジャンプし，割込み管理テーブルで設定された割込み要因に対応するエントリのアドレスに分岐する．

11. 実際の割込み対応する ISR を呼び出す．このとき当該割込み処理が Managed ISR であれば ISR を呼び出す前にレジスタを退避する．

12. 13 の処理の間に割込みが入らないように，SR の BL ビットを 1 にする．

13. 割込みスタックを調べ，処理すべき割込み構造体があれば割込みマスクを更新後，BL ビットをクリアしてその割込み処理に復帰する．

14. 割込みスタックが空であれば，割込みマスクを更新後，SR の BL ビットをクリアして，割り込んだ元のコードへ復帰する．

ISR への入口と出口のレジスタの退避，復帰，ベクタによる目的の ISR の呼び出しおよび多重割込みの管理が OS 側で提供する処理である．ここで，ステップの 1 番から 5 番まではハードウェアで自動的に行なわれる． 割込み構造体と割込みスタックのインタフェースを実現するために OS で提供する 6 番から 9 番までの処理が，割込みハンドラの ISR の「入口の処理」である．また，11 番と 12 番の機能は，ISR の「出口の処理」である．この「入口の処理」および「出口の処理」では，原則としてユーザがレジスタを意識しないですむような設計にする．即ち，レジスタの退避/復帰は OS にすべて任される．また，割込み構造体と割込みスタックによる抽象化により，ユーザがレジスタを意識することなく，多重割込みは OS 側で管理される．それを実現するためのレジスタの退避/復帰も OS 側が管理する．

## 4.2 Intel Strong ARM 1100 への適用

Strong ARM のレジスタは，R0 から R15 までの計 16 個の汎用レジスタがある．R15 はプログラムカウンタ（PC）として使用され，R14 は PC の退避用として使用される．R13 は通常スタックポインタとして使用される． プロセッサの状態を設定する Current Program Status Register(CPSR) がある．プロサッセのモードはユーザ 32 モード，(Fast Interrupt Request)FIQ32 モード，スーパバイザ 32 モード，IRQ(Interrupt Request)32 モード，未定義 32 モードの 5 つのモードがある．各モード毎に CPSR を退避するための Stored Program Status Register(SPSR) があり，非ユーザモードへ移行する時は，関連する SPSR に CPSR が保存される．FIQ モードでは R8_FIQ から R14_FIQ が，IRQ モードでは，R13_IRQ および R14_IRQ がバンクレジスタとして使用される．CPSR の I ビットがセットされると IRQ 割込みが禁止され，F ビットがセットされると FIQ 割込みが禁止される．

以下に，IRQ 割込みの処理の例を示す．

1. R15(PC) と CPSR の内容がそれぞれ新しいモードの R14 バンクレジスタと SPSR_IRQ とに退避される．

2. IRQ 割込みを禁止するため，CPSR の I ビットが 1 に設定される．

3. CPSR のモードが IRQ モードに設定される．

4. IRQ 割込みベクタに従い，0x00000018 番地へ一旦分岐し，そこからさらに割込み要因判定ルーチンへ分岐する．

5. 割込み要因判定ルーチンで割込みコントローラのフラグレジスタをを調べることにより，どのユニットから割込みが入ったか特定する．ユニットを特定後，割り込んだユニット内のステータスレジスタを調べて，当該ユニットの内の割込み要因を特定する．

6. R14， SPSR および保存すべき全てのレジスタを割込みタスク構造体として退避し，当該割込み構造体の情報を割込みスタックに格納する．

7. 多重割込みを許すため，CPSR の I ビットをクリアする．

8. 割込み管理テーブルで指定した割込み要因に対応するISRを呼び出す．このとき，当該割込み処理がManaged ISRであればISRを呼び出す前にOSがレジスタを退避する．

9. 10および11の処理中に割込みが入らないように，CPSRのIビットを1にする．

10. 割込みスタックを調べ，処理すべき割込みタスクがあれば，割込みマスクの更新後，Iビットをクリアして，その割込み処理に復帰する．

11. 割込みスタックが空であれば，割込みマスクの更新後，Iビットをクリアして，割り込んだ元のコードへ復帰する．

ここで，ステップの1番から4番までがハードウェアで自動的に行なわれる．5番から7番までの処理がOS側で提供する機能であるISRの「入口の処理」である．また，9番から11番までの機能は，ISRの「出口の処理」である．ここでも、割り込みスタックと割込み構造体を構成し，多重割込みを管理する．

### 4.3 考察

・ベクタテーブルについて
各プロセッサとも，プロセッサが外部から割込みを受け付けると，外部割り込みに対応して一度特定のアドレスに分岐する．その後、割込み要因判定ルーチンに直接又は間接にジャンプし，ベクタテーブルを引いて，割込み要因に対応するISRを呼び出す．SH2からSH3への移行で見られているように，組込み用途向けプロセッサは，割込みの構成の変化にも柔軟に対応できるため，割込み要因判定ルーチンをソフトウェアで構成するものが増えると思われる．これらのプロセッサに対して，割込み管理テーブルは有効である．一方，Intel x86系プロセッサに対しても割込み管理テーブルを設けることにより，割込み要因とISRのエントリをユーザがソフトウェアで統一的に記述可能となる．もちろん，ベクタテーブルで指定されるISRは，割込みに対して処理すべき目的の処理のみを高級言語で記述できる．

・割込みスタックと割込み構造体の有効性と問題点
割込みスタックと割込み構造体を設けることにより，ユーザはレジスタを気にすることなく多重割込みに対処できる．また，ユーザは割込みに対する目的の処理のみを高級言語で記述でき，移植性を向上することが出来る．多重割込みを許す場合，さらなる割込みに対して常に大量のレジスタ情報の保存が必要となる．従ってバンクレジスタの恩恵をなかなか受けることが出来ないという問題点が出てくる．Unmanaged Interruptを使用することが考えられるが，移植性が落ちる．

## 5 おわりに

本研究では，割込み機能の抽象化インタフェースとして，Managed ISRおよびUnmanaged ISR，割込み管理テーブル，割込み構造体，割込みスタックを提案し，ユーザにプロセッサやレジスタを意識させない機構を提案した．これにより移植性を向上できる．今後はバンクレジスタの存在を考慮した割込み機能の抽象化インタフェースを含めて，さらなる処理の高速化の可能性を探っていく．

## 参考文献


[1] 田中義照 “組込みシステム向け オペレーティングシステム の構成” RTP'99.

[2] “日立 SuperH RISC engine SH7708 シリーズ ハードウェアマニュアル”.

[3] 監修 坂村 健 “$\mu$ ITRON3．0 標準ハンドブック改訂新版”，パーソナルメディア， 1997.

[4] “ARM RISC Chip A programmer's Guide”，シミュレーションメディア，1995.

[5] “Strong ARM SA-1100 Portable Communications Microcontroller DATA SHEET V 1. 2”，Digital Equipment Corporation.

[6] W.B. スルヤント “80386 の使い方”，オーム社， 1987.

[7] 吉田 俊郎 “80386 システムプログラム”，オーム社， 1989.