お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

User's Manual

32

SH-4A

ソフトウェアマニュアル

ルネサス32ビットRISC マイクロコンピュータ
SuperH™ RISC engine ファミリ

改訂一覧は表紙をクリックして直接ご覧になれます。

改訂一覧は改訂箇所をまとめたものであり、
詳細については必ず本文の内容をご確認ください。

ルネサス エレクトロニクス
www.renesas.com

Rev.1.50　2004.10

安全設計に関するお願い

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

本資料ご利用に際しての留意事項

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

製品に関する一般的注意事項

1. NC 端子の処理

   【注意】NC端子には、何も接続しないようにしてください。

   NC(Non-Connection)端子は、内部回路に接続しない場合の他、テスト用端子やノイズ軽減などの目的で使用します。このため、NC端子には、何も接続しないようにしてください。

2. 未使用入力端子の処理

   【注意】未使用の入力端子は、ハイまたはローレベルに固定してください。

   CMOS製品の入力端子は、一般にハイインピーダンス入力となっています。未使用端子を開放状態で動作させると、周辺ノイズの誘導により中間レベルが発生し、内部で貫通電流が流れて誤動作を起こす恐れがあります。

   未使用の入力端子は、入力をプルアップかプルダウンによって、ハイまたはローレベルに固定してください。

3. 初期化前の処置

   【注意】電源投入時は、製品の状態は不定です。

   すべての電源に電圧が印加され、リセット端子にローレベルが入力されるまでの間、内部回路は不確定であり、レジスタの設定や各端子の出力状態は不定となります。この不定状態によってシステムが誤動作を起こさないようにシステム設計を行ってください。

   リセット機能を持つ製品は、電源投入後は、まずリセット動作を実行してください。

4. 未定義・リザーブアドレスのアクセス禁止

   【注意】未定義・リザーブアドレスのアクセスを禁止します。

   未定義・リザーブアドレスは、将来の機能拡張用の他、テスト用レジスタなどが割り付けられています。

   これらのレジスタをアクセスしたときの動作および継続する動作については、保証できませんので、アクセスしないようにしてください。

5. 各レジスタリザーブビットの読み出し／書き込み

   各モジュールで使用されるレジスタのリザーブビットは、その説明記述中に読み出し／書き込み値の指定が特にない限り以下のように取り扱ってください。

   読み出すと常に0が読み出されます。書き込む場合は、0を書き込むか、直前に読み出した値を書き込むかいずれかにしてください。

   直前に読み出した値を書き込むようにしておくと、将来このビットに拡張機能を割り当てることがある場合、その拡張機能に影響を与えない利点があります。

本書の構成

本書は、以下の構成で制作しています。

1. 製品に関する一般的注意事項
2. 本書の構成
3. はじめに
4. 目次
5. 概要
6. 各機能モジュールの説明
   - CPU およびシステム制御系
   - 内蔵周辺モジュール

     各モジュールの機能説明の構成は、モジュール毎に異なりますが、一般的には、
     ①特長、②入出力端子、③レジスタの説明、④動作説明、⑤使用上の注意事項、
     等の節で構成されています。

   > 本 LSI を用いた応用システムを設計する際、注意事項を十分確認の上設計してください。
   > 各章の本文中には説明に対する注意事項と、各章の最後には使用上の注意事項があります。
   > 必ずお読みください。(使用上の注意事項は必要により記載されます。)

7. レジスタ一覧
8. 索引

はじめに

SH-4A は、ルネサス テクノロジオリジナルの RISC 方式の CPU コアです。

対象者　本マニュアルは、SH-4A を用いた応用システムを設計するユーザーを対象としています。
本マニュアルを使用される読者には、電気回路、論理回路、マイクロコンピュータ、およびアセンブリ／C 言語プログラミングに関する基本的な知識を必要とします。

目的　本マニュアルは、SH-4A の命令を理解していただくことを目的としています。ハードウェアについては、当該製品のハードウェアマニュアルを参照してください。

読み方

- 機能全体を理解しようとするとき。
  → 目次にしたがって読んでください。
  本書は、大きく分類すると、CPU機能、システム制御機能、各命令の説明に順に構成されています。
- 各命令を理解したいとき。
  → 命令の体系と基本動作は**「第3章　命令セット」**に説明されています。各命令の詳細動作は、**「第10章　各命令の説明」**を読んでください。

凡例　レジスタ表記　：シリアルコミュニケーションなど、同一または類似した機能が複数チャネルに存在する場合に、次の表記を使用します。
XXX_N（XXX は基本レジスタ名称、N はチャネル番号）
ビット表記　：左側が上位ビット、右側が下位ビットの順に表記します。
数字の表記　：2 進数は B'XXXX、16 進数は H'XXXX、10 進数は XXXX で表します。
記号の表記　：ローアクティブの信号にはオーバーバー（$\overline{\text{XXXX}}$）を付けます。

略語の説明

ALU	Arithmetic Logic Unit 演算論理回路
ASID	Address Space Identifier アドレス空間識別子
CPU	Central Processing Unit 中央制御装置
FPU	Floating Point Unit 浮動小数点演算装置
LRU	Least Recently Used （仮想記憶ページ置き換えアルゴリズムの名前）
LSB	Least Significant Bit 最下位ビット
MMU	Memory Management Unit メモリマネジメントユニット
MSB	Most Significant Bit 最上位ビット
PC	Program Counter プログラムカウンタ
RISC	Reduced Instruction Set Computer 縮小命令セットコンピュータ
TLB	Translation Lookaside Buffer 変換ルックアサイドバッファ

目次

1. 概要 1-1

1.1 SH-4Aの特長 1-1
1.2 SH-4からSH-4Aへの変更点 1-3

2. プログラミングモデル 2-1

2.1 データフォーマット 2-1
2.2 レジスタの構成 2-2
2.2.1 特権モードとバンク 2-2
2.2.2 汎用レジスタ 2-5
2.2.3 浮動小数点レジスタ 2-6
2.2.4 コントロールレジスタ 2-9
2.2.5 システムレジスタ 2-11
2.3 メモリ割り付けレジスタ 2-14
2.4 レジスタのデータ形式 2-14
2.5 メモリ上でのデータ形式 2-15
2.6 処理状態 2-16
2.7 使用上の注意事項 2-17
2.7.1 自己書き換えコードに対する注意事項 2-17

3. 命令セット 3-1

3.1 実行環境 3-1
3.2 アドレッシングモード 3-3
3.3 命令セット 3-7

4. パイプライン動作 4-1

4.1 パイプライン 4-1
4.2 並列実行性 4-12
4.3 発行レートと実行ステート 4-15

5. 例外処理 5-1

5.1 概要 5-1
5.2 レジスタの説明 5-1
5.2.1 TRAPA 例外レジスタ（TRA） 5-2
5.2.2 例外事象レジスタ（EXPEVT） 5-2

5.2.3 割り込み事象レジスタ（INTEVT） 5-3
5.3 例外処理の機能 5-4
5.3.1 例外処理の流れ 5-4
5.3.2 例外処理ベクタアドレス 5-4
5.4 例外の種類と優先順位 5-5
5.5 例外フロー 5-6
5.5.1 例外フロー 5-6
5.5.2 例外要因の受け付け 5-7
5.5.3 例外要求と BL ビット 5-8
5.5.4 例外処理からの復帰 5-8
5.6 各例外の説明 5-9
5.6.1 リセット 5-9
5.6.2 一般例外 5-10
5.6.3 割り込み 5-20
5.6.4 複数回の例外が発生する場合の優先順位 5-22
5.7 注意事項 5-23

6. 浮動小数点ユニット（FPU） 6-1

6.1 概要 6-1
6.2 データフォーマット 6-2
6.2.1 浮動小数点フォーマット 6-2
6.2.2 非数（NaN） 6-4
6.2.3 非正規化数 6-5
6.3 レジスタ 6-6
6.3.1 浮動小数点レジスタ 6-6
6.3.2 浮動小数点ステータス／コントロールレジスタ（FPSCR） 6-8
6.3.3 浮動小数点通信レジスタ（FPUL） 6-10
6.4 丸め 6-10
6.5 浮動小数点例外 6-11
6.6 グラフィックサポート機能 6-13
6.6.1 ジオメトリック演算命令 6-13
6.6.2 ペア単精度データ転送 6-14

7. メモリマネジメントユニット（MMU） 7-1

7.1 MMUの概要 7-1
7.1.1 アドレス空間 7-3
7.2 レジスタの説明 7-8
7.2.1 ページテーブルエントリ上位レジスタ（PTEH） 7-9
7.2.2 ページテーブルエントリ下位レジスタ（PTEL） 7-10
7.2.3 変換テーブルベースレジスタ（TTB） 7-11

7.2.4 TLB 例外アドレスレジスタ（TEA） 7-11
7.2.5 MMU 制御レジスタ（MMUCR） 7-11
7.2.6 物理アドレス空間制御レジスタ（PASCR） 7-14
7.2.7 命令再フェッチ抑止制御レジスタ（IRMCR） 7-15
7.3 TLBの機能 7-17
7.3.1 共用 TLB（UTLB）の構成 7-17
7.3.2 命令 TLB（ITLB）の構成 7-19
7.3.3 アドレス変換方式 7-20
7.4 MMUの機能 7-22
7.4.1 MMU のハードウェア管理 7-22
7.4.2 MMU のソフトウェア管理 7-22
7.4.3 MMU の命令（LDTLB） 7-23
7.4.4 ハードウェア ITLB ミスハンドリング 7-24
7.4.5 シノニム問題の回避 7-24
7.5 MMU例外 7-25
7.5.1 命令 TLB 多重ヒット例外 7-25
7.5.2 命令 TLB ミス例外 7-25
7.5.3 命令 TLB 保護違反例外 7-26
7.5.4 データ TLB 多重ヒット例外 7-27
7.5.5 データ TLB ミス例外 7-28
7.5.6 データ TLB 保護違反例外 7-29
7.5.7 初期ページ書き込み例外 7-30
7.6 メモリ割り付けTLBの構成 7-31
7.6.1 ITLB アドレスアレイ 7-31
7.6.2 ITLB データアレイ 7-32
7.6.3 UTLB アドレスアレイ 7-33
7.6.4 UTLB データアレイ 7-34
7.7 32ビットアドレス拡張モード 7-35
7.7.1 32 ビットアドレス拡張モード概要 7-35
7.7.2 32 ビットアドレス拡張モードへの切り替え 7-35
7.7.3 特権空間マッピングバッファ（PMB）構成 7-36
7.7.4 PMB の機能 7-37
7.7.5 メモリ割り付け PMB の構成 7-38
7.7.6 32 ビットアドレス拡張モード使用時の注意事項 7-39

8. キャッシュ 8-1

8.1 特長 8-1
8.2 レジスタの説明 8-4
8.2.1 キャッシュ制御レジスタ（CCR） 8-5
8.2.2 キューアドレス制御レジスタ 0（QACR0） 8-6

8.2.3 キューアドレス制御レジスタ 1（QACR1） 8-7
8.2.4 内蔵メモリ制御レジスタ（RAMCR） 8-7
8.3 オペランドキャッシュの動作説明 8-9
8.3.1 読み出し動作 8-9
8.3.2 プリフェッチ動作 8-10
8.3.3 書き込み動作 8-11
8.3.4 ライトバックバッファ 8-12
8.3.5 ライトスルーバッファ 8-12
8.3.6 OC 2 ウェイモード 8-12
8.4 命令キャッシュの動作説明 8-13
8.4.1 読み出し動作 8-13
8.4.2 プリフェッチ動作 8-14
8.4.3 IC 2 ウェイモード 8-14
8.5 キャッシュ操作命令 8-15
8.5.1 キャッシュと外部メモリとのコヒーレンシ 8-15
8.5.2 プリフェッチ動作 8-16
8.6 メモリ割り付けキャッシュの構成 8-16
8.6.1 IC アドレスアレイ 8-17
8.6.2 IC データアレイ 8-18
8.6.3 OC アドレスアレイ 8-19
8.6.4 OC データアレイ 8-20
8.7 ストアキュー 8-21
8.7.1 SQ の構成 8-21
8.7.2 SQ への書き込み 8-21
8.7.3 外部メモリへの転送 8-21
8.7.4 SQ アクセスの例外判定 8-23
8.7.5 SQ からの読み出し 8-23
8.8 32ビットアドレス拡張モード使用時の注意事項 8-24

9. L メモリ 9-1

9.1 特長 9-1
9.2 レジスタの説明 9-2
9.2.1 内蔵メモリ制御レジスタ（RAMCR） 9-3
9.2.2 L メモリ転送元アドレスレジスタ 0（LSA0） 9-4
9.2.3 L メモリ転送元アドレスレジスタ 1（LSA1） 9-5
9.2.4 L メモリ転送先アドレスレジスタ 0（LDA0） 9-6
9.2.5 L メモリ転送先アドレスレジスタ 1（LDA1） 9-7
9.3 動作説明 9-8
9.3.1 CPU および FPU からのアクセス 9-8
9.3.2 SuperHyway バスマスタモジュールからのアクセス 9-8

9.3.3 ブロック転送 ... 9-8
9.4 Lメモリの保護機能 ... 9-9
9.5 使用上の注意 ... 9-10
9.5.1 ページ競合 ... 9-10
9.5.2 L メモリのコヒーレンシ ... 9-10
9.5.3 スリープモード ... 9-10
9.6 32ビットアドレス拡張モード使用時の注意事項 ... 9-10

10. 各命令の説明 ... 10-1

10.1 CPU命令 ... 10-2

10.1.1	ADD	ADD binary	算術演算命令 ... 10-4
10.1.2	ADDC	ADD with Carry	算術演算命令 ... 10-5
10.1.3	ADDV	ADD with (Vflag) overflow check	算術演算命令 ... 10-6
10.1.4	AND	AND logical	論理演算命令 ... 10-8
10.1.5	BF	Branch if False	分岐命令 ... 10-10
10.1.6	BF/S	Branch if False with delay Slot	分岐命令 ... 10-12
10.1.7	BRA	BRAnch	分岐命令 ... 10-14
10.1.8	BRAF	BRAnch Far	分岐命令 ... 10-16
10.1.9	BT	Branch if True	分岐命令 ... 10-17
10.1.10	BT/S	Branch if True with delay Slot	分岐命令 ... 10-19
10.1.11	CLRMAC	CLeaR MAC register システム	制御命令 ... 10-21
10.1.12	CLRS	CLeaR Sbit システム	制御命令 ... 10-22
10.1.13	CLRT	CLeaR Tbit システム	制御命令 ... 10-23
10.1.14	CMP/cond	CoMPare conditionally	算術演算命令 ... 10-24
10.1.15	DIV0S	DIVide(step0) as Signed	算術演算命令 ... 10-28
10.1.16	DIV0U	DIVide (step0) as Unsigned	算術演算命令 ... 10-29
10.1.17	DIV1	DIVide 1 step	算術演算命令 ... 10-30
10.1.18	DMULS.L	Double-length MULtiply as Signed	算術演算命令 ... 10-34
10.1.19	DMULU.L	Double-length MULtiply as Unsigned	算術演算命令 ... 10-36
10.1.20	DT	Decrement and Test	算術演算命令 ... 10-38
10.1.21	EXTS	EXTend as Signed	算術演算命令 ... 10-39
10.1.22	EXTU	EXTend as Unsigned	算術演算命令 ... 10-40
10.1.23	ICBI	Instruction Cache Block Invalidate	データ転送命令 ... 10-41
10.1.24	JMP	JuMP	分岐命令 ... 10-42
10.1.25	LDC	LoaD to Control register	システム制御命令 ... 10-43
10.1.26	LDS	LoaD to System register	システム制御命令 ... 10-47
10.1.27	LDTLB	LoaD PTEH/PTEL to TLB	システム制御命令 ... 10-49
10.1.28	MAC.L	Multiply and ACcumulate Long	算術演算命令 ... 10-51
10.1.29	MAC.W	Multiply and ACcumulate Word	算術演算命令 ... 10-55
10.1.30	MOV	MOVe data	データ転送命令 ... 10-58

10.1.31	MOV	MOVe constant value	データ転送命令 10-63
10.1.32	MOV	MOVe global data	データ転送命令 10-66
10.1.33	MOV	MOVe structure data	データ転送命令 10-69
10.1.34	MOVA	MOVe effective Address	データ転送命令 10-72
10.1.35	MOVCA.L	MOVe with Cache block Allocation	データ転送命令 10-73
10.1.36	MOVCO	MOVe COnditional	データ転送命令 10-74
10.1.37	MOVLI	MOVe LInked	データ転送命令 10-76
10.1.38	MOVT	MOVe Tbit	データ転送命令 10-77
10.1.39	MOVUA	MOVe UnAligned	データ転送命令 10-78
10.1.40	MUL.L	MULtiply Long	算術演算命令 10-80
10.1.41	MULS.W	MULtiply as Signed Word	算術演算命令 10-81
10.1.42	MULU.W	MULtiply as Unsigned Word	算術演算命令 10-82
10.1.43	NEG	NEGate	算術演算命令 10-83
10.1.44	NEGC	NEGate with Carry	算術演算命令 10-84
10.1.45	NOP	No OPeration	システム制御命令 10-85
10.1.46	NOT	NOT-logical complement	論理演算命令 10-86
10.1.47	OCBI	Operand Cache Block Invalidate	データ転送命令 10-87
10.1.48	OCBP	Operand Cache Block Purge	データ転送命令 10-88
10.1.49	OCBWB	Operand Cache Block Write Back	データ転送命令 10-89
10.1.50	OR	OR logical	論理演算命令 10-90
10.1.51	PREF	PREFetch data to cache	データ転送命令 10-92
10.1.52	PREFI	PREFetch Instruction cache block	データ転送命令 10-93
10.1.53	ROTCL	ROTate with Carry Left	シフト命令 10-94
10.1.54	ROTCR	ROTate with Carry Right	シフト命令 10-95
10.1.55	ROTL	ROTate Left	シフト命令 10-96
10.1.56	ROTR	ROTate Right	シフト命令 10-97
10.1.57	RTE	ReTurn from Exception	システム制御命令 10-98
10.1.58	RTS	ReTurn from Subroutine	分岐命令..... 10-100
10.1.59	SETS	SET Sbit	システム制御命令 10-102
10.1.60	SETT	SET Tbit	システム制御命令 10-103
10.1.61	SHAD	SHift Arithemetic Dynamically	シフト命令 10-104
10.1.62	SHAL	SHift Arithmetic Left	シフト命令 10-106
10.1.63	SHAR	SHift Arithmetic Right	シフト命令 10-107
10.1.64	SHLD	SHift Logical Dynamically	シフト命令 10-108
10.1.65	SHLL	SHift Logical Left	シフト命令 10-110
10.1.66	SHLLn n	bits SHift Logical Left	シフト命令 10-111
10.1.67	SHLR	SHift Logical Right	シフト命令 10-113
10.1.68	SHLRn	n bits SHift Logical Right	シフト命令 10-114
10.1.69	SLEEP	SLEEP	システム制御命令 10-116
10.1.70	STC	STore Control register	システム制御命令 10-117
10.1.71	STS	STore System register	システム制御命令 10-121

10.1.72 SUB SUBtract binary 算術演算命令 10-123
10.1.73 SUBC SUBtract with Carry 算術演算命令 10-124
10.1.74 SUBV SUBtract with (Vflag)underflow check 算術演算命令 10-125
10.1.75 SWAP SWAP register halves データ転送命令 10-127
10.1.76 SYNCO SYNChronize data Operation データ転送命令 10-129
10.1.77 TAS Test And Set 論理演算命令 10-130
10.1.78 TRAPA TRAP Always システム制御命令 10-132
10.1.79 TST TeST logical 論理演算命令 10-134
10.1.80 XOR eXclusive OR logical 論理演算命令 10-136
10.1.81 XTRCT eXTRaCT データ転送命令 10-138
10.2 CPU命令（FPU関係） 10-139
10.2.1 BSR Branch to SubRoutine 分岐命令 10-140
10.2.2 BSRF Branch to SubRoutine Far 分岐命令 10-142
10.2.3 JSR Jump to SubRoutine 分岐命令 10-144
10.2.4 LDC LoaD to Control register システム制御命令 10-146
10.2.5 LDS LoaD to FPU System register システム制御命令 10-147
10.2.6 STC STore Control register システム制御命令 10-149
10.2.7 STS Store from FPU System register システム制御命令 10-150
10.3 FPU命令 10-152
10.3.1 FABS Floating - point ABSolute value 浮動小数点命令 10-162
10.3.2 FADD Floating - point ADD 浮動小数点命令 10-163
10.3.3 FCMP Floating - point CoMPare 浮動小数点命令 10-166
10.3.4 FCNVDS Floating - point CoNVert Double to Single precision 浮動小数点命令 10-170
10.3.5 FCNVSD Floating - point CoNVert Single to Double precision 浮動小数点命令 10-173
10.3.6 FDIV Floating - point DIVide 浮動小数点命令 10-175
10.3.7 FIPR Floating - point Inner PRoduct 浮動小数点命令 10-179
10.3.8 FLDI0 Floating - point LoaD Immediate 0.0 浮動小数点命令 10-181
10.3.9 FLDI1 Floating - point LoaD Immediate 1.0 浮動小数点命令 10-182
10.3.10 FLDS Floating - point LoaD to System register 浮動小数点命令 10-183
10.3.11 FLOAT Floating - point convert from integer 浮動小数点命令 10-184
10.3.12 FMAC Floating - point Multiply and Accumulate 浮動小数点命令 10-186
10.3.13 FMOV Floating - point MOVe 浮動小数点命令 10-191
10.3.14 FMOV Floating - point MOVe extension 浮動小数点命令 10-195
10.3.15 FMUL Floating - point MULtiply 浮動小数点命令 10-198
10.3.16 FNEG Floating - point NEGate value 浮動小数点命令 10-202
10.3.17 FPCHG Pr-bit ChanGe 浮動小数点命令 10-203
10.3.18 FRCHG FR-bit CHanGe 浮動小数点命令 10-204
10.3.19 FSCA Floating Point Sine And Cosine Approximate 浮動小数点命令 10-205
10.3.20 FSCHG Sz-bit CHanGe 浮動小数点命令 10-207
10.3.21 FSQRT Floating - point SQuare RooT 浮動小数点命令 10-208
10.3.22 FSRRA Floating Ponit Square Reciprocal Approximate 浮動小数点命令 10-211

10.3.23 FSTS Floating - point Store System register 浮動小数点命令 10-213
10.3.24 FSUB Floating - point SUBtract 浮動小数点命令 10-214
10.3.25 FTRC Floating - point Truncate and Convert to integer 浮動小数点命令 10-217
10.3.26 FTRV Floating - point TRansform Vector 浮動小数点命令 10-220

11. レジスター覧 11-1

11.1 レジスタアドレス一覧（機能モジュールごと、マニュアル章番号順） 11-2
11.2 各動作モードにおけるレジスタの状態 11-3

付録 **付録-1**

A. CPU動作モードレジスタ（CPUOPM） 付録-1
B. 命令プリフェッチとその副作用について 付録-2
C. サブルーチン復帰投機実行 付録-3
D. バージョンレジスタ（PVR、PRR） 付録-4

本版で修正または追加された箇所 **改訂-1**

索引 **索引-1**

図目次

2. プログラミングモデル
図2.1 データフォーマット ... 2-1
図2.2 処理モード別のCPUレジスタ構成 ... 2-4
図2.3 汎用レジスタ ... 2-5
図2.4 浮動小数点レジスタ ... 2-8
図2.5 SZビットとエンディアンの関係 ... 2-13
図2.6 バイトデータ、ワードデータのレジスタ中のデータ形式 ... 2-14
図2.7 メモリ上のデータ形式 ... 2-15
図2.8 処理状態遷移図 ... 2-16

4. パイプライン動作
図4.1 基本パイプライン ... 4-1
図4.2 命令実行パターン（1） ... 4-3
図4.2 命令実行パターン（2） ... 4-4
図4.2 命令実行パターン（3） ... 4-5
図4.2 命令実行パターン（4） ... 4-6
図4.2 命令実行パターン（5） ... 4-7
図4.2 命令実行パターン（6） ... 4-8
図4.2 命令実行パターン（7） ... 4-9
図4.2 命令実行パターン（8） ... 4-10
図4.2 命令実行パターン（9） ... 4-11

5. 例外処理
図5.1 命令実行と例外処理 ... 5-6
図5.2 一般例外の受け付け順序の例 ... 5-7

6. 浮動小数点ユニット（FPU）
図6.1 単精度浮動小数点フォーマット ... 6-2
図6.2 倍精度浮動小数点フォーマット ... 6-2
図6.3 単精度のNaNビットパターン ... 6-4
図6.4 浮動小数点レジスタ ... 6-7
図6.5 SZビットとエンディアンの関係 ... 6-9

7. メモリマネジメントユニット（MMU）
図7.1 MMUの役割 ... 7-2
図7.2 仮想アドレス空間（MMUCR.AT＝0） ... 7-3
図7.3 仮想アドレス空間（MMUCR.AT＝1） ... 7-4
図7.4 P4領域 ... 7-5
図7.5 物理アドレス空間 ... 7-6
図7.6 UTLBの構成 ... 7-17
図7.7 ページサイズとアドレスの関係 ... 7-19
図7.8 ITLBの構成 ... 7-19

図7.9　UTLBを用いたメモリアクセスフロー 7-20
図7.10　ITLBを用いたメモリアクセスフロー 7-21
図7.11　LDTLB命令の動作 7-23
図7.12　メモリ割り付けITLBアドレスアレイ 7-32
図7.13　メモリ割り付けITLBデータアレイ 7-32
図7.14　メモリ割り付けUTLBアドレスアレイ 7-33
図7.15　メモリ割り付けUTLBデータアレイ 7-34
図7.16　物理アドレス空間（32ビットアドレス拡張モード） 7-35
図7.17　PMBの構成 7-36
図7.18　メモリ割り付けPMBアドレスアレイ 7-38
図7.19　メモリ割り付けPMBデータアレイ 7-39

8.　キャッシュ
図8.1　オペランドキャッシュの構成 8-2
図8.2　命令キャッシュの構成 8-3
図8.3　ライトバックバッファの構成 8-12
図8.4　ライトスルーバッファの構成 8-12
図8.5　メモリ割り付けICアドレスアレイ 8-18
図8.6　メモリ割り付けICデータアレイ 8-18
図8.7　メモリ割り付けOCアドレスアレイ 8-20
図8.8　メモリ割り付けOCデータアレイ 8-20
図8.9　ストアキューの構成 8-21

付録
図B.1　命令のプリフェッチ 付録-2

表目次

1. 概要
   表1.1 SH-4Aの特長 1-1
   表1.2 SH-4からSH-4Aへの変更点 1-3

2. プログラミングモデル
   表2.1 レジスタの初期値 2-3
   表2.2 FPU例外処理に関連するビットの割り付け 2-13

3. 命令セット
   表3.1 遅延分岐命令の実行順序 3-1
   表3.2 アドレッシングモードと実効アドレス 3-3
   表3.3 命令リストの表記 3-7
   表3.4 固定小数点転送命令 3-9
   表3.5 算術演算命令 3-10
   表3.6 論理演算命令 3-12
   表3.7 シフト命令 3-13
   表3.8 分岐命令 3-14
   表3.9 システム制御命令 3-15
   表3.10 浮動小数点単精度命令 3-17
   表3.11 浮動小数点倍精度命令 3-18
   表3.12 浮動小数点制御命令 3-18
   表3.13 浮動小数点グラフィック強化命令 3-19

4. パイプライン動作
   表4.1 命令実行パターン表記説明 4-2
   表4.2 命令グループ 4-12
   表4.3 先行・後行掛け合わせ表 4-14
   表4.4 発行レートと実行ステート 4-16

5. 例外処理
   表5.1 レジスタ構成 5-1
   表5.2 各処理モードにおけるレジスタの状態 5-1
   表5.3 例外一覧 5-5

6. 浮動小数点ユニット（FPU）
   表6.1 浮動小数点のフォーマットとパラメータ 6-3
   表6.2 浮動小数点の範囲 6-3
   表6.3 FPU例外処理に関連するビットの割り付け 6-10

7. メモリマネジメントユニット（MMU）
   表7.1 レジスタ構成 7-8
   表7.2 各処理状態におけるレジスタの状態 7-8

8. キャッシュ
表8.1 キャッシュの特長 8-1
表8.2 ストアキューの特長 8-1
表8.3 レジスタ構成 8-4
表8.4 各処理モードにおけるレジスタの状態 8-4

9. L メモリ
表9.1 Lメモリアドレス 9-1
表9.2 レジスタ構成 9-2
表9.3 各処理状態におけるレジスタの状態 9-2
表9.4 Lメモリへのアクセスに対する保護機能による例外 9-10

付録
表D.1 レジスタ構成 付録-4

1. 概要

1.1 SH-4A の特長

SH-4A は、SH-1、SH-2、SH-3、SH-4 マイクロコンピュータと命令セットレベルでの上位互換性を特長とする 32 ビット RISC（縮小命令セットコンピュータ）マイコンです。16 ビット固定長の命令セットにより、32 ビット命令に比較してプログラムコードのサイズをほぼ 50%縮小することができます。

SH-4A の特長を**表 1.1** に示します。

表 1.1　SH-4A の特長

項目	特　　長
CPU	• ルネサス テクノロジオリジナルアーキテクチャ • 32 ビット内部データバス • 汎用レジスタファイル： 16 本の 32 ビット汎用レジスタ（および 8 本の 32 ビットシャドウレジスタ） 7 本の 32 ビット制御レジスタ 4 本の 32 ビットシステムレジスタ • RISC タイプ命令セット（SH-1、SH-2、SH-3、SH-4 と上位互換性有り）： 命令長：　コードの効率改善のための 16 ビット固定長 ロードストアアーキテクチャ 遅延分岐命令 条件付き実行 C 言語に基づく命令セット • FPU を含む 2 命令同時実行型スーパスカラ • 命令実行時間：　最大 2 命令／サイクル • 仮想アドレス空間：　4G バイト • 空間識別子 ASID：　8 ビット、256 仮想アドレス空間 • 乗算器内蔵 • 7 段パイプライン

項目	特　　長
浮動小数点 ユニット （FPU）	• 浮動小数点コプロセッサ内蔵 • 単精度（32 ビット）および倍精度（64 ビット）をサポート • IEEE754 に準拠したデータタイプおよび例外をサポート • 丸めモード：近傍および 0 方向への丸め • 非正規化数の扱い：0 への切捨て、または IEEE754 に準拠のための割り込み発生 • 浮動小数点レジスタ：32 ビット×16 ワード×2 バンク （単精度×16 ワードまたは倍精度×8 ワード）×2 バンク • 32 ビット CPU-FPU 浮動小数点通信レジスタ（FPUL） • FMAC（乗算およびアキュムレート）命令をサポート • FDIV（除算）／FSQRT（平方根）命令をサポート • FLDI0／FLDI1（ロード定数 0/1）命令をサポート • 命令実行時間 レイテンシ（FADD/FSUB）：3 サイクル（単精度）、5 サイクル（倍精度） レイテンシ（FMAC/FMUL）：5 サイクル（単精度）、7 サイクル（倍精度） ピッチ（FADD/FSUB）：1 サイクル（単精度／倍精度） ピッチ（FMAC/FMUL）：1 サイクル（単精度）、3 サイクル（倍精度） 【注】： FMAC は単精度に対してのみサポートしています。 • 3D グラフィック命令（単精度のみ）： 4 次元ベクトル変換および行列演算（FTRV）、4 サイクル（ピッチ）、8 サイクル（レイテンシ） 4 次元ベクトル（FIPR）の内積、1 サイクル（ピッチ）、5 サイクル（レイテンシ） • 10 段パイプライン
メモリマネジメントユニット （MMU）	• 4G バイトのアドレス空間、256 のアドレス空間識別子（ASID 8 ビット） • 単一仮想記憶モードと多重仮想記憶モード • 複数のページサイズをサポート：　1K、4K、64K、1M バイト • 命令に対する 4 エントリのフルアソシアティブ TLB • 命令およびオペランドに対する 64 エントリのフルアソシアティブ TLB • ソフトウェアによる入換方法およびランダムカウンタ方式入換アルゴリズムをサポート • TLB の内容はアドレスマッピングにより直接アクセス可能
キャッシュ メモリ	• 命令キャッシュ（IC） 4 ウェイセットアソシアティブ 32 バイトブロック長 • オペランドキャッシュ（OC） 4 ウェイセットアソシアティブ 32 バイトブロック長 選択可能な書き込み方式（コピーバック／ライトスルー） • ストアキュー（32 バイト×2 エントリ） 【注】　命令キャッシュ、オペランドキャッシュのキャッシュ容量については、製品のハードウェアマニュアルを参照してください。

項目	特　　長
L メモリ	• 2 本の独立した読み出し／書き込みポート CPU からの 8／16／32／64 ビットアクセス 外部要求による 8／16／32／64 ビットおよび 16／32 バイトアクセス 【注】L メモリの容量については、製品のハードウェアマニュアルを参照してください。

1.2　SH-4 から SH-4A への変更点

SH-4 から SH-4A への変更点を、本マニュアルの章、節単位に示します。

表 1.2　SH-4 から SH-4A への変更点

章番号、章名	節番号	節名	変更点
1. 概要	−	−	全面変更 （個別の変更点は 2 章以降の差分で記載します）
2. プログラミングモデル	2.2	レジスタの構成	浮動小数点ステータス／コントロールレジスタ（FPSCR）　SZ=1, PR=1 での動作追加
3. 命令セット	3.3	命令セット	CPU 命令として 9 命令を追加
			FPU 命令として 3 命令を追加
4. パイプライン動作	4.1	パイプライン	パイプライン段数を 5 から 7 に変更
	4.2	並列実行性	CPU 命令して 9 命令を追加
			FPU 命令として 3 命令を追加
			命令のグループ分けと並列実行組合せ変更
	4.3	実行サイクル	実行サイクル数変更
5. 例外処理	—	—	特になし
6. FPU	6.3.2	浮動小数点ステータス／コントロールレジスタ（FPSCR）	SZ=1, PR=1 での動作追加およびエンディアン毎の動作説明追加
	6.5	浮動小数点例外	FPU 例外イネーブル設定時の FPU 例外検出条件の仕様を変更
7. メモリマネジメントユニット	7.1.1	アドレス空間	P4 領域の構成変更
			内蔵 RAM 空間を削除
	7.2	レジスタの説明	ページテーブルエントリアシストレジスタ（PTEA）の削除
			物理アドレス空間制御レジスタの追加
	7.2.6	物理アドレス空間制御レジスタ（PASCR）	新規追加
	7.2.7	命令再フェッチ抑止レジスタ（IRMCR）	新規追加
	7.3	TLB の機能	TLB から、空間属性ビット（SA[2:0] ）、タイミングコントロールビット（TC）を削除

章番号、章名	節番号	節名	変更点
7. メモリマネジメントユニット	7.4.5	シノニム問題の回避	キャッシュサイズ変更およびインデックスモード削除に伴い、該当ビット変更
	7.5.1、7.5.4	命令 TLB 多重ヒット例外およびデータ TLB 多重ヒット例外	ITLB ミスハンドリングによる UTLB 検索で多重ヒットになった場合も命令 TLB 多重ヒット例外に変更
	7.6	メモリ割り付け TLB の構成	ITLB および UTLB のデータアレイ 2 を削除
	7.6.3	UTLB アドレスアレイ	UTLB アドレスアレイに対する連想ライトではデータ TLB 多重ヒット例外を発生しないよう変更
			メモリ割り付けアドレスを H'F600 0000～H'F6FF FFFF から H'F600 0000～H'F60F FFFF へ変更
	7.6.4	UTLB データアレイ	メモリ割り付けアドレスを H'F700 0000～H'F77F FFFF から H'F700 0000～H'F70F FFFF へ変更
	7.7	32 ビットアドレス拡張モード	新規追加
8. キャッシュ	8.1	特長	命令キャッシュの容量を 32k バイトに変更
			方式を 4 ウェイセットアソシアティブに変更
	8.2	レジスタの説明	内蔵メモリ制御レジスタを追加
	8.2.1	キャッシュ制御レジスタ（CCR）	内容変更
	8.2.4	内蔵メモリ制御レジスタ（RAMCR）	新規追加
	8.3	オペランドキャッシュの動作説明	RAM モードおよび OC インデックスモードを削除
	8.3.6	OC 2 ウェイモード	新規追加
	8.4	命令キャッシュの動作説明	IC インデックスモードを削除
	8.4.3	IC 2 ウェイモード	新規追加
	8.5.1	キャッシュと外部メモリとのコヒーレンシ	ICBI 命令、PREFI 命令、および SYNCO 命令追加
	8.6	メモリ割り付けキャッシュの構成	容量の変更および 4 ウェイセットアソシアティブ化に伴い、エントリビットとウェイビット変更
	8.8	32 ビットアドレス拡張モード使用時の注意事項	新規追加
9. L メモリ	－	－	新規追加
10. 各命令の説明	－	－	CPU 命令として 9 命令追加
			FPU 命令として 3 命令追加

2. プログラミングモデル

本章では、SH-4A のプログラミングモデルについて記述します。SH-4A では以下に示すレジスタとデータ形式を持っています。

2.1 データフォーマット

SH-4A でサポートしているデータフォーマットを図 2.1 に示します。

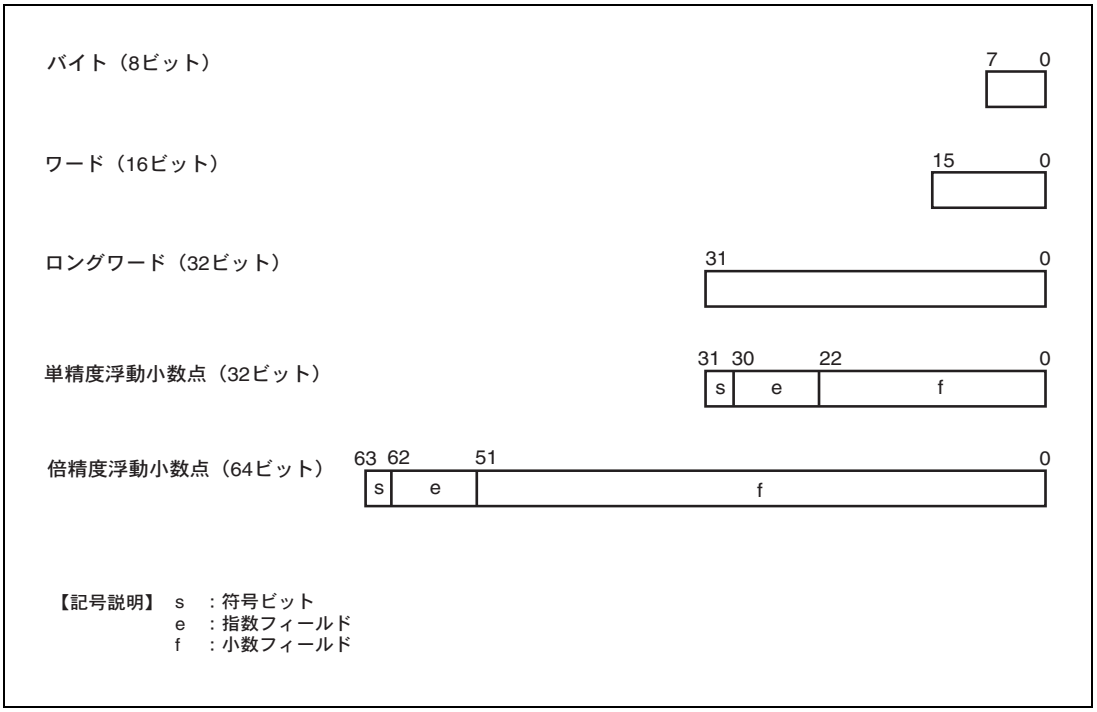


図 2.1 データフォーマット

2.2 レジスタの構成

2.2.1 特権モードとバンク

(1) 処理モード

処理モードにはユーザモードと特権モードの 2 つがあります。通常はユーザモードで動作し、例外が発生または割り込みを受け付けると特権モードになります。レジスタには、汎用レジスタ、システムレジスタ、コントロールレジスタ、および浮動小数点レジスタがあり、アクセスできるレジスタはそれぞれの処理モードで異なります。

(2) 汎用レジスタ

汎用レジスタには R0 から R15 までの 16 本のレジスタがあります。汎用レジスタ R0 から R7 は、バンクレジスタで、処理モードで切り替えることができます。

- 特権モードの場合

  ステータスレジスタ（SR）のレジスタバンクビット（RB）により、汎用レジスタとしてアクセスできるレジスタとできないレジスタが決められます。汎用レジスタとしてアクセスできないレジスタは、コントロールレジスタのロード命令（LDC）とストア命令（STC）でアクセスします。

  RBビットが1のとき、つまりバンク1が選ばれているときは、バンク1の汎用レジスタR0_BANK1からR7_BANK1とバンクに関係ないR8からR15との合計16本のレジスタが汎用レジスタR0からR15としてアクセスすることができ、バンク0の汎用レジスタR0_BANK0からR7_BANK0の8本のレジスタはLDC/STC命令でアクセスできます。

  RBビットが0のとき、つまりバンク0が選ばれているときは、バンク0の汎用レジスタR0_BANK0からR7_BANK0とバンクに関係ないR8からR15との合計16本のレジスタが汎用レジスタR0からR15としてアクセスすることができ、バンク1の汎用レジスタR0_BANK1からR7_BANK1の8本のレジスタはLDC/STC命令でアクセスできます。

- ユーザモードの場合

  バンク0の汎用レジスタR0_BANK0からR7_BANK0とバンクに関係ないR8からR15との合計16本のレジスタが汎用レジスタR0からR15としてアクセスすることができ、バンク1の汎用レジスタR0_BANK1からR7_BANK1の8本のレジスタはアクセスできません。

(3) コントロールレジスタ

コントロールレジスタには、処理モードで共通のグローバルベースレジスタ（GBR）とステータスレジスタ（SR）があり、特権モードでのみ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）、退避ジェネラルレジスタ 15（SGR）、デバッグベースレジスタ（DBR）があります。ステータスレジスタには、特権モードでのみアクセスできるビット（例えば RB ビット）があります。

(4) システムレジスタ

システムレジスタには、積和レジスタ（MACH/MACL）、プロシージャレジスタ（PR）、プログラムカウンタ（PC）があり、処理モードに関係しません。

(5) 浮動小数点レジスタと FPU に関するシステムレジスタ

浮動小数点レジスタには、FR0～FR15、XF0～XF15 の 32 本のレジスタがあります。FR0～FR15、XF0～XF15 をおのおの FPR0_BANK0～FPR15_BANK0、FPR0_BANK1～FPR15_BANK1 のいずれのバンクに割り付けるか選択できます。

また、FR0～FR15 は、DR0/2/4/6/8/10/12/14（倍精度浮動小数点レジスタ、またはレジスタペア）の 8 本、FV0/4/8/12（レジスタベクタ）の 4 本として使用でき、XF0～XF15 は、XD0/2/4/6/8/10/12/14（レジスタペア）の 8 本、XMTRX（レジスタ行列）の 1 本として使用できます。

FPU に関するシステムレジスタには、浮動小数点コミュニケーションレジスタ（FPUL）と浮動小数点ステータス／コントロールレジスタ（FPSCR）があり、FPU-CPU 間の通信や例外処理の設定を行います。

リセット後のレジスタの値を**表 2.1** に示します。

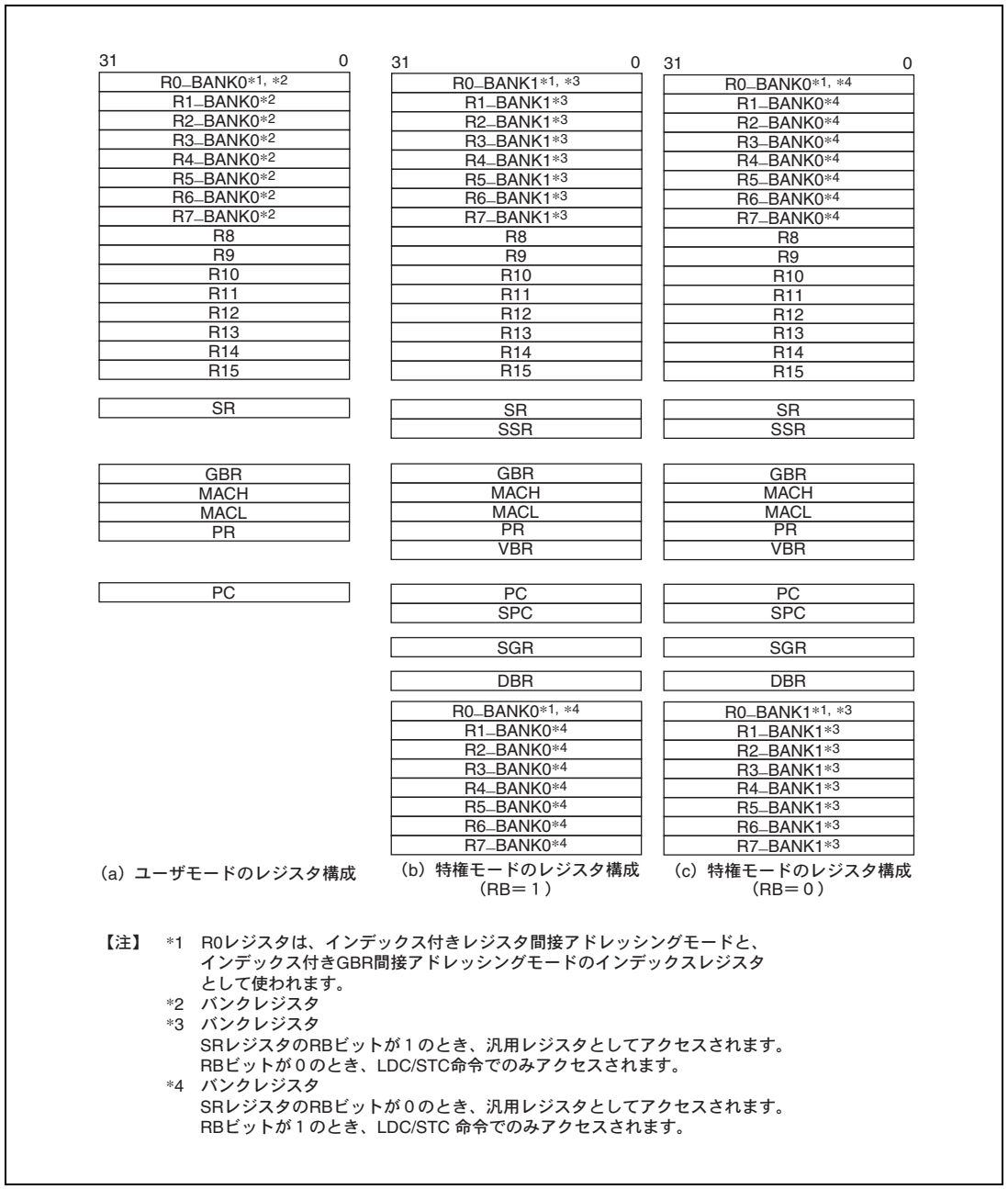
表 2.1　レジスタの初期値

区分	レジスタ	初期値*
汎用レジスタ	R0_BANK0～R7_BANK0、 R0_BANK1～R7_BANK1、 R8～R15	不定
コントロールレジスタ	SR	MD ビットは 1、RB ビットは 1、BL ビットは 1、FD ビットは 0、IMASK は B'1111、リザーブビットは 0、その他は不定
	GBR、SSR、SPC、SGR、DBR	不定
	VBR	H'00000000
システムレジスタ	MACH、MACL、PR	不定
	PC	H'A0000000
浮動小数点レジスタ	FR0～FR15、XF0～XF15、FPUL	不定
	FPSCR	H'00040001

【注】 * パワーオンリセット、マニュアルリセットで初期化されます。

処理モード別の CPU レジスタ構成を**図 2.2** に示します。

ユーザモードと特権モードは、ステータスレジスタの処理モードビット（MD）で切り替えます。



【注】 *1 R0レジスタは、インデックス付きレジスタ間接アドレッシングモードと、インデックス付きGBR間接アドレッシングモードのインデックスレジスタとして使われます。
*2 バンクレジスタ
*3 バンクレジスタ
SRレジスタのRBビットが1のとき、汎用レジスタとしてアクセスされます。
RBビットが0のとき、LDC/STC命令でのみアクセスされます。
*4 バンクレジスタ
SRレジスタのRBビットが0のとき、汎用レジスタとしてアクセスされます。
RBビットが1のとき、LDC/STC 命令でのみアクセスされます。

図 2.2 処理モード別の CPU レジスタ構成

2.2.2 汎用レジスタ

図 2.3 に処理モードと汎用レジスタの関係を示します。SH-4A には 24 本の 32 ビット汎用レジスタ（R0_BANK0～R7_BANK0、R0_BANK1～R7_BANK1、R8～R15）があります。ただし、これらのうち 16 本のレジスタのみ 1 つの処理モードで汎用レジスタ R0～R15 としてアクセスできます。SH-4A には特権モードとユーザモードの 2 つの処理モードがあります。R0～R7 はその 2 つのモードにより次のように割り当てられます。

- R0_BANK0～R7_BANK0

  ユーザモード（SR.MD＝0）では、常にR0～R7に割り当てられます。

  特権モード（SR.MD＝1）では、（SR.RB＝0）の場合に限りR0～R7に割り当てられます。

- R0_BANK1～R7_BANK1

  ユーザモードでは、アクセスできません。

  特権モードでは、（SR.RB＝1）の場合に限り、R0～R7に割り当てられます。

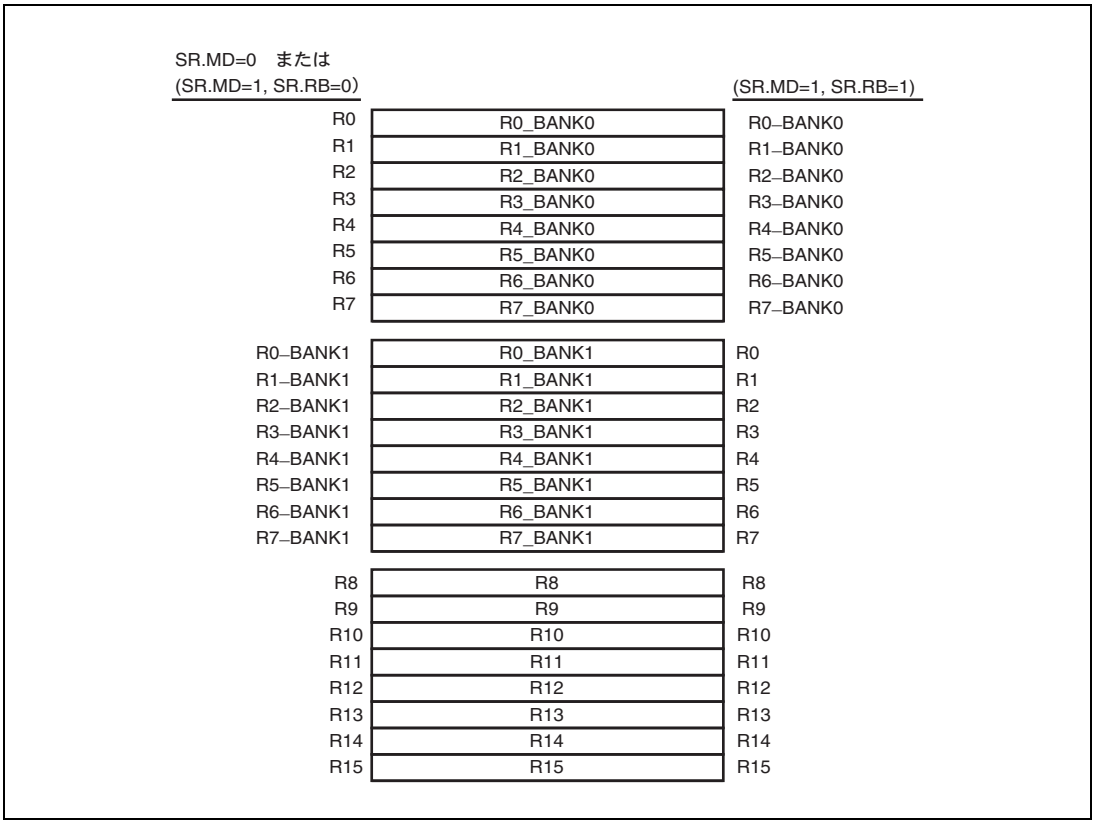


図 2.3 汎用レジスタ

【プログラミング上の注意】

ユーザモードの R0～R7 は R0_BANK0～R7_BANK0 に、例外・割り込み後の R0～R7 は R0_BANK1～R7_BANK1 に割り当てられるので、割り込みハンドラはユーザモードの R0～R7（R0_BANK0～R7_BANK0）を退避または復帰する必要はありません。

2.2.3 浮動小数点レジスタ

図 2.4 に浮動小数点レジスタを示します。32 本の 32 ビット浮動小数点レジスタがあります。これらは、2 つのバンクで構成され、FPR0_BANK0～FPR15_BANK0、FPR0_BANK1～FPR15_BANK1 があります。また、この 32 本レジスタは FR0～FR15、DR0/2/4/6/8/l0/12/14、FV0/4/8/12、XF0～XF15、XD0/2/4/6/8/l0/12/14、XMTRX として参照されます。FPRn_BANKi と参照名の対応は FPSCR の FR ビットによって決まります。**図 2.4** を参照してください。

（1） 浮動小数点レジスタ　FPRn_BANKi（32 レジスタ）

FPR0_BANK0、FPR l_BANK0、FPR2_BANK0、FPR3_BANK0、
FPR4_BANK0、FPR5_BANK0、FPR6_BANK0、FPR7_BANK0、
FPR8_BANK0、FPR9_BANK0、FPR10_BANK0、FPR11_BANK0、
FPR12_BANK0、FPR13_BANK0、FPR14_BANK0、FPR15_BANK0
FPR0_BANK1、FPR1_BANK1、FPR2_BANK1、FPR3_BANK1、
FPR4_BANK1、FPR5_BANK1、FPR6_BANK1、FPR7_BANK1、
FPR8_BANK1、FPR9_BANK1、FPR10_BANK1、FPR11_BANK1、
FPR12_BANK1、FPR13_BANK1、FPR14_BANK1、FPR15_BANK1

（2） 単精度浮動小数点レジスタ　FRi（16 レジスタ）

FPSCR.FR＝0 のとき、FR0～FR15 は FPR0_BANK0～FPR15_BANK0 に割り当てられます。
FPSCR.FR＝1 のとき、FR0～FR15 は FPR0_BANK1～FPR15_BANK1 に割り当てられます。

（3） 倍精度浮動小数点レジスタ、または単精度浮動小数点レジスタのペア　DRi（8 レジスタ）

DR レジスタは、2 つの FR レジスタから構成されます。
DR0＝{FR0、FR1}、DR2＝{FR2、FR3}、
DR4＝{FR4、FR5}、DR6＝{FR6、FR7}、
DR8＝{FR8、FR9}、DR10＝{FR10、FR11}、
DR12＝{FR12、FR13}、DR14＝{FR14、FR15}

（4） 単精度浮動小数点ベクトルレジスタ　FVi（4 レジスタ）

FV レジスタは 4 つの FR レジスタから構成されます。
FV0＝{FR0、FR1、FR2、FR3},
FV4＝{FR4、FR5、FR6、FR7},
FV8＝{FR8、FR9、FR10、FR11},
FVl2＝{FR12、FR13、FR14、FR15}

(5) 単精度浮動小数点拡張レジスタ　XFi（16 レジスタ）

FPSCR.FR＝0 のとき、XF0～XF15 は FPR0_BANK1～FPR15_BANK1 に割り当てられます。

FPSCR.FR＝1 のとき、XF0～XF15 は FPR0_BANK0～FPR15_BANK0 に割り当てられます。

(6) 単精度浮動小数点拡張レジスタのペア　XDi（8 レジスタ）

XD レジスタは 2 つの XF レジスタから構成されます。

XD0＝{XF0、XF1}、XD2＝{XF2、XF3}、

XD4＝{XF4、XF5}、XD6＝{XF6、XF7}、

XD8＝{XF8、XF9}、XD10＝{XF10、XF11}、

XD12＝{XF12、XF13}、XD14＝{XF14、XF15}

(7) 単精度浮動小数点拡張レジスタ行列　XMTRX

XMTRX は 16 本の XF レジスタから構成されます。

$$XMTRX = \begin{pmatrix} XF0 & XF4 & XF8 & XF12 \\ XF1 & XF5 & XF9 & XF13 \\ XF2 & XF6 & XF10 & XF14 \\ XF3 & XF7 & XF11 & XF15 \end{pmatrix}$$

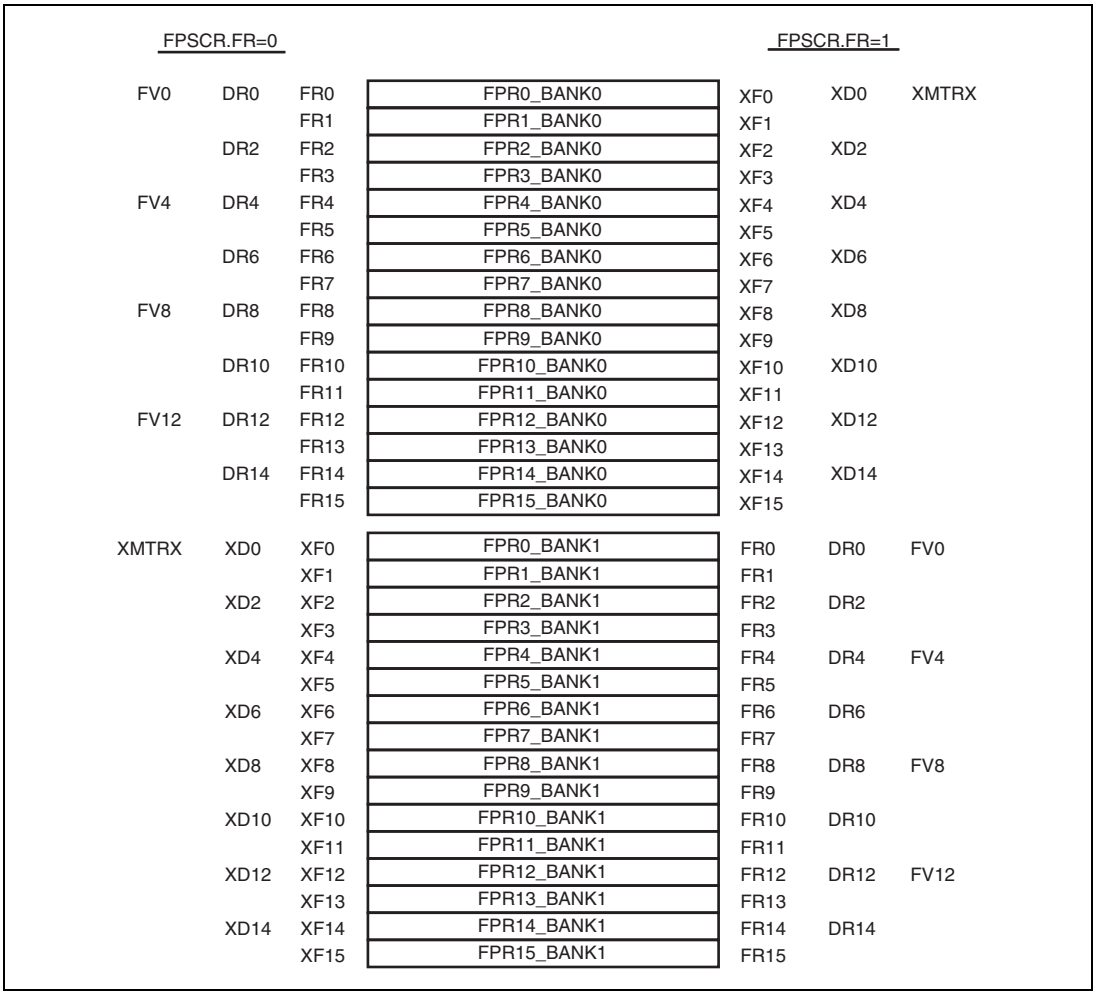


図 2.4　浮動小数点レジスタ

2.2.4 コントロールレジスタ

（1） ステータスレジスタ（SR）

ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	MD	RB	BL	—	—	—	—	—	—	—	—	—	—	—	—
初期値：	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R/W	R/W	R/W	R	R	R	R	R	R	R	R	R	R	R	R

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	FD	—	—	—	—	—	M	Q	IMASK				—	—	S	T
初期値：	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
R/W：	R/W	R	R	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W	R	R	R/W	R/W

ビット	ビット名	初期値	R/W	説　　明
31	－	0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
30	MD	1	R/W	処理モード 処理モードを選択します。 0：ユーザモード （命令の中には実行できない命令があります。また、リソースの中にはアクセスできないリソースがあります。） 1：特権モード 例外または割り込みにより 1 にセットされます。
29	RB	1	R/W	特権モードでの汎用レジスタバンク指定ビット 0：R0_BANK0～R7_BANK0 は汎用レジスタ R0～R7 としてアクセスでき、R0_BANK1～R7_BANK1 は LDC/STC 命令でアクセスできます。 1：R0_BANK1～R7_BANK1 は汎用レジスタ R0～R7 としてアクセスでき、R0_BANK0～R7_BANK0 は LDC/STC 命令でアクセスできます。 例外または割り込みにより 1 にセットされます。
28	BL	1	R/W	例外／割り込みブロックビット このビットが 1 のとき、割り込み要求はマスクされ、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に遷移します。 例外または割り込みにより 1 にセットされます。
27～16	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
15	FD	0	R/W	FPU ディスエーブルビット このビットが 1 のとき、FPU 命令は一般 FPU 抑止例外を発生させ、FPU 命令が遅延スロットにある場合、スロット FPU 抑止例外が発生します（FPU 命令：H'F***命令、FPUL/FPSCR に対する LDS(.L)/STS(.L)命令）。

ビット	ビット名	初期値	R/W	説　　明
14～10	−	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
9	M	0	R/W	M ビット DIV0S、DIV0U、DIV1 命令で使用します。
8	Q	0	R/W	Q ビット DIV0S、DIV0U、DIV1 命令で使用します。
7～4	IMASK	すべて 1	R/W	割り込みマスクレベル IMASK 以下のレベルの割り込みはマスクされます。また、割り込みが発生した場合に、IMASK が割り込み受け付けレベルに変化する動作と変化しない動作を、CPU 動作モードレジスタ（CPUOPM）を用いて切り替えることができます。CPUOPM の動作は、「付録 A.　CPU 動作モードレジスタ（CPUOPM）」を参照してください。
3、2	−	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
1	S	0	R/W	S ビット MAC 命令の飽和動作を指定します。
0	T	0	R/W	T ビット 真／偽条件、キャリ、ボロー、オーバフローまたはアンダフローなどを表します。 詳細は、「第 3 章　命令セット」を参照してください。

(2) 退避ステータスレジスタ（SSR）（32 ビット、特権保護、初期値＝不定）

SR の内容は例外または割り込みの発生時、SSR に退避されます。

(3) 退避プログラムカウンタ（SPC）（32 ビット、特権保護、初期値＝不定）

例外または割り込みの発生した命令のアドレスは SPC に退避されます。

(4) グローバルベースレジスタ（GBR）（32 ビット、初期値＝不定）

GBR は@(disp,GBR)、@(R0,GBR)アドレッシングのベースアドレスとして参照されます。

(5) ベクタベースレジスタ（VBR）（32 ビット、特権保護、初期値＝H'0000 0000）

VBR は例外および割り込み発生時、分岐先のベースアドレスとして参照されます。詳細については**「第 5 章　例外処理」**を参照してください。

(6) 退避ジェネラルレジスタ 15（SGR）（32 ビット、特権保護、初期値＝不定）

R15 の内容は例外または割り込みの発生時 SGR に退避されます。

(7) デバッグベースレジスタ（DBR）（32 ビット、特権保護、初期値＝不定）

ユーザブレークデバッグ機能を有効にする場合（CBCR.UBDE＝1）、DBR は VBR の代わりにユーザブレークハンドラへの分岐先アドレスとして参照されます。

2.2.5 システムレジスタ

(1) 積和上位レジスタ（MACH）（32 ビット、初期値＝不定）、
積和下位レジスタ（MACL）（32 ビット、初期値＝不定）

MACH／MACL は、MAC 命令の加算値として用いられます。また MAC 命令、MUL 命令の演算結果を格納するためにも用いられます。

(2) プロシージャレジスタ（PR）（32 ビット、初期値＝不定）

BSR、BSRF、JSR 命令を用いたサブルーチンコールの戻りアドレスは PR に格納されます。PR は、サブルーチンからの復帰命令（RTS）によって参照されます。

(3) プログラムカウンタ（PC）（32 ビット、初期値＝H'A000 0000）

PC は実行中の命令アドレスを示します。

(4) 浮動小数点ステータス／コントロールレジスタ（FPSCR）

ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	FR	SZ	PR	DN	Cause	
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Cause				Enable（EN）					Flag					RM	
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
R/W：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ビット	ビット名	初期値	R/W	説　　明
31～22	−	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
21	FR	0	R/W	浮動小数点レジスタバンク 0：FPR0_BANK0～FPR15_BANK0 は FR0～FR15 に、FPR0_BANK1～FPR15_BANK1 は XF0～XF15 に割り当てられます。 1：FPR0_BANK0～FPR15_BANK0 は XF0～XF15 に、FPR0_BANK1～FPR15～BANK1 は FR0～FR15 に割り当てられます。

ビット	ビット名	初期値	R/W	説　　明
20	SZ	0	R/W	転送サイズモード 0：FMOV 命令のデータサイズは 32 ビットです。 1：FMOV 命令のデータサイズは 32 ビットペア、または 64 ビットです。 SZ ビットおよび PR ビットとエンディアンとの関係については、図 2.5 を参照してください。
19	PR	0	R/W	精度モード 0：浮動小数点命令を単精度演算として実行します。 1：浮動小数点命令を倍精度演算として実行します（グラフィックサポート命令は未定義です）。 PR ビットおよび SZ ビットとエンディアンとの関係については、図 2.5 を参照してください。
18	DN	1	R/W	非正規化モード 0：非正規化数を非正規化数として扱います。 1：非正規化数を 0 として扱います。
17～12	Cause	すべて 0	R/W	FPU 例外要因フィールド
11～7	Enable（EN）	すべて 0	R/W	FPU 例外イネーブルフィールド
6～2	Flag	すべて 0	R/W	FPU 例外フラグフィールド FPU 演算命令を実行すると、FPU 例外要因フィールドは最初に 0 に設定されます。次に FPU 例外が発生すると、FPU 例外要因フィールドと FPU 例外フラグフィールドの該当ビットが 1 にセットされます。 FPU 例外フラグフィールドは、FPU 例外フラグフィールドが最後にクリアされたそれ以降に発生した例外のステータスを保持します。 各フィールドのビットの割り付けについては表 2.2 を参照してください。
1、0	RM	01	R/W	丸めモード 丸めの方法を選択します。 00：近傍への丸め 01：0 方向への丸め 10：リザーブ（設定禁止） 11：リザーブ（設定禁止）

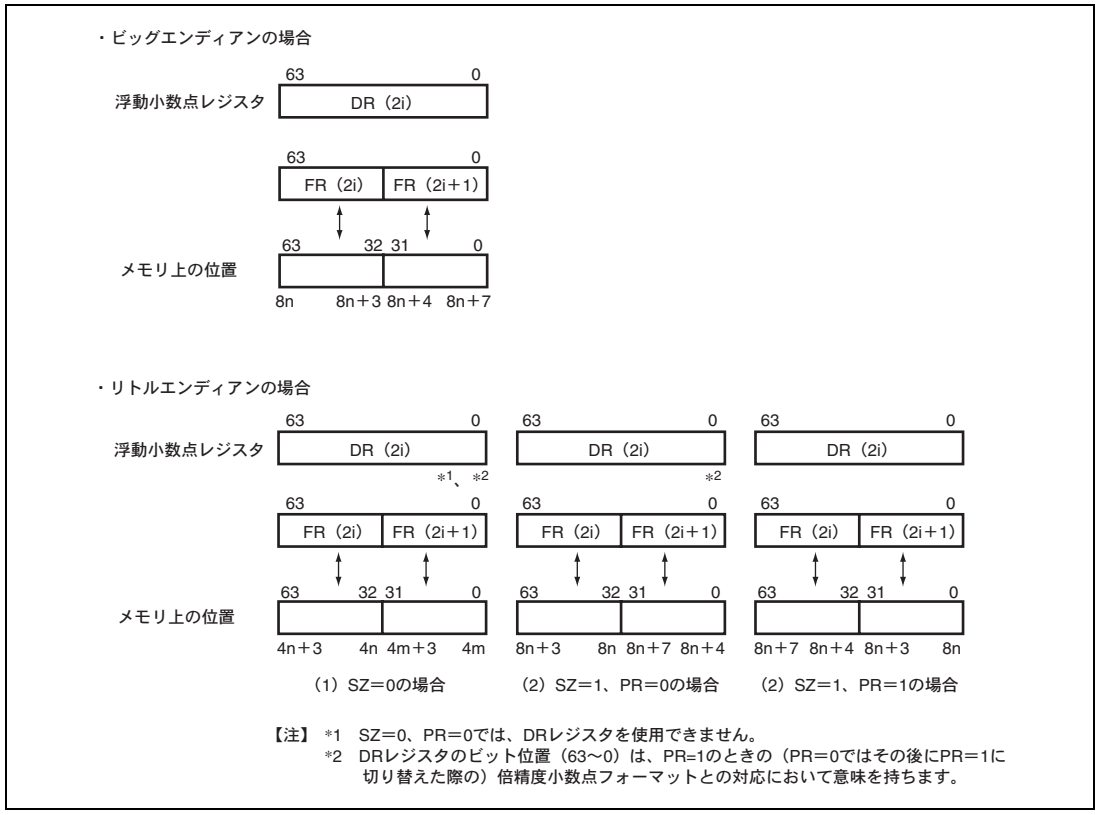


図 2.5 SZ ビットとエンディアンの関係

表 2.2 FPU 例外処理に関連するビットの割り付け

		FPU エラー (E)	無効演算 (V)	0 除算 (Z)	オーバフロー（O）	アンダフロー（U）	不正確 (I)
Cause	FPU 例外要因フィールド	ビット 17	ビット 16	ビット 15	ビット 14	ビット 13	ビット 12
Enable	FPU 例外イネーブルフィールド	なし	ビット 11	ビット 10	ビット 9	ビット 8	ビット 7
Flag	FPU 例外フラグフィールド	なし	ビット 6	ビット 5	ビット 4	ビット 3	ビット 2

(5) 浮動小数点通信レジスタ（FPUL）（32 ビット、初期値＝不定）

FPU レジスタと CPU レジスタ間のデータ転送は、FPUL を介して行われます。

2.3 メモリ割り付けレジスタ

制御レジスタのうち、以下のメモリ領域にマッピングされているものがあります。これらのメモリ領域に割り付けられたレジスタには、2 つのアドレスがあります。

H'1C00 0000～H'1FFF FFFF

H'FC00 0000～H'FFFF FFFF

以上 2 つの領域は次のように使用します。

- H'1C00 0000～H'1FFF FFFF

  この領域はMMUのアドレス変換機能を用いてアクセスしなければなりません。この領域のページ番号をTLBの該当フィールドに設定することでメモリ割り付けレジスタへアクセスできます。この領域に対して、MMUのアドレス変換機能を用いずにアクセスした場合の動作は保証されません。

- H'FC00 0000～H'FFFF FFFF

  ユーザモードで領域H'FC00 0000～H'FFFF FFFFにアクセスすると、アドレスエラーが発生します。ユーザモードではメモリ割り付けレジスタはアドレス変換によるアクセスで参照することができます。

【注】 **2 つの領域のレジスタが割り付けられていないアドレスにはアクセスしないでください。レジスタが割り付けられていないアドレスに対するアクセスの動作は不定になります。また、メモリ割り付けレジスタは一定のデータサイズでアクセスしなければなりません。不正なサイズでアクセスした場合も動作は不定になります。**

2.4 レジスタのデータ形式

レジスタオペランドのデータサイズは常にロングワード（32 ビット）です。メモリ上のデータをレジスタへロードするとき、メモリオペランドのデータサイズがバイト（8 ビット）、もしくはワード（16 ビット）の場合は、ロングワードに符号拡張し、レジスタに格納します。

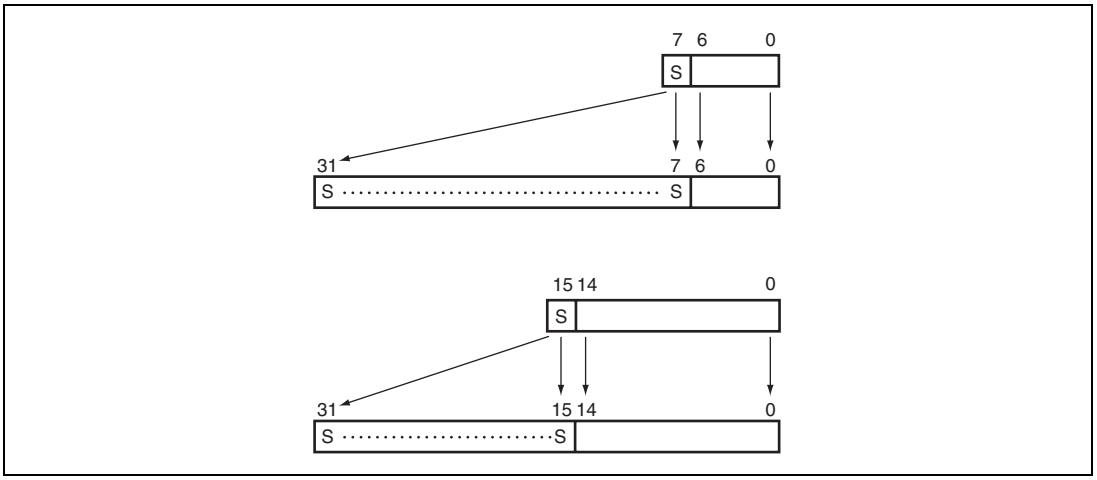


図 2.6 バイトデータ、ワードデータのレジスタ中のデータ形式

2.5 メモリ上でのデータ形式

バイト、ワード、ロングワードのデータ形式があります。メモリは 8 ビットのバイト、16 ビットのワード、32 ビットのロングワードいずれの形でもアクセスすることができます。32 ビットに満たないメモリオペランドは符号拡張されてレジスタに格納されます。

ワードオペランドはワード境界（2 バイト刻みの偶数番地：2n 番地）から、ロングワードオペランドはロングワード境界（4 バイト刻みの偶数番地：4n 番地）からアクセスしてください。これを守らない場合は、アドレスエラーになります。バイトオペランドはどの番地からでもアクセスできます。

データフォーマットは、ビッグエンディアンかリトルエンディアンのどちらかのバイト順を選択できます。エンディアンはパワーオンリセット時に外部ピンで設定してください。エンディアンは動的には変更できません。ただしビット位置は常に最上位（most-significant）から最下位（least-significant）へ左から右へ減少するように番号が付けられています。すなわち 32 ビットのロングワードでは、一番左のビット、ビット 31 が最上位ビットで、一番右のビット、ビット 0 が最下位ビットです。

メモリ上のデータ形式を**図 2.7** に示します。

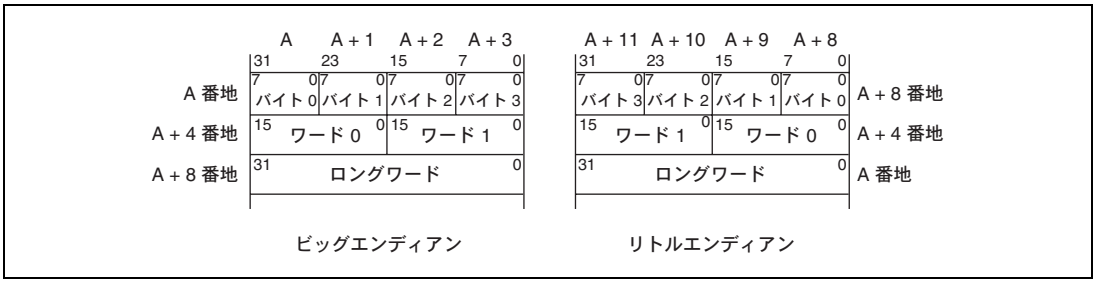


図 2.7 メモリ上のデータ形式

64 ビットのデータ形式については**図 2.5** を参照してください。

2.6 処理状態

処理状態には、大きく分けてリセット状態、命令実行状態、低消費電力状態の 3 種類があります。

(1) リセット状態

CPU がリセットされている状態です。リセット状態は、パワーオンリセット状態とマニュアルリセット状態に分類されます。

パワーオンリセット状態では、CPU の内部状態と内蔵周辺モジュールのレジスタが初期化されます。マニュアルリセット状態では、一部の内蔵周辺モジュールのレジスタと CPU の内部状態とが初期化されます。詳細は、ハードウェアマニュアルの各章のレジスタ構成を参照してください。

(2) 命令実行状態

CPU が順次プログラムを実行している状態です。命令実行状態には、一般のプログラム実行状態と例外処理状態があります。

(3) 低消費電力状態

CPU の動作が停止し消費電力が低い状態です。スリープ命令で低消費電力状態になります。スリープモード、およびスタンバイモードの 2 つのモードがあります。低消費電力状態の詳細は、当該製品ハードウェアマニュアルの**「低消費電力モード」**を参照してください。

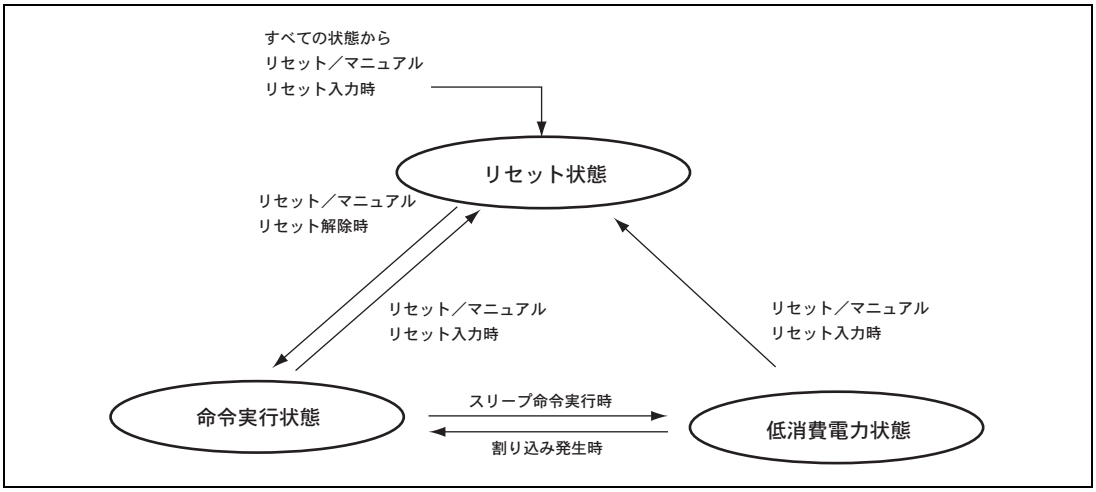


図 2.8 処理状態遷移図

2.7 使用上の注意事項

2.7.1 自己書き換えコードに対する注意事項

SH-4A は、処理を高速に行うために命令の先読みを行っています。このためメモリ上の命令列の書き換えを行った直後に当該命令を実行しようとすると、先読みバッファに格納された更新前の命令が実行される可能性があります。また、命令／オペランド分離方式のキャッシュを搭載するため、コヒーレンシにも注意する必要があります。確実に変更を反映させるためには、書き換えを行う命令と書き換えられた命令の実行の間に下記の命令列を実行するようにしてください。

(1) 書き換える命令がキャッシング不可能領域にある場合

```
SYNCO
ICBI @Rn
```

ICBI 命令の Rn で指定するアドレスは、アドレスエラーにならない範囲で任意のアドレスで構いません。

(2) 書き換える命令列がキャッシング可能（ライトスルー）領域にある場合

```
SYNCO
ICBI @Rn
```

書き換えた命令列に対応する命令キャッシュの領域すべてを ICBI 命令で無効化してください。ICBI はライン単位で行います。1 ラインは 32 バイトです。

(3) 書き換える命令列がキャッシング可能（コピーバック）領域にある場合

OCBP @Rm または OCBWB @Rm

```
SYNCO
ICBI @Rn
```

書き換えた命令列に対応するオペランドキャッシュの領域すべてを OCBP 命令または OCBWB 命令で主記憶に書き戻しを行い、その後 ICBI 命令で対応する命令キャッシュ領域の無効化を行ってください。ICBI/OCBP/OCBWB はライン単位で行います。1 ラインは 32 バイトです。

3. 命令セット

SH-4A の命令セットは固定長 16 ビット命令で実現されます。SH-4A はバイト（8 ビット）、ワード（16 ビット）、ロングワード（32 ビット）、クワッドワード（64 ビット）のデータサイズでメモリにアクセスします。単精度浮動小数点データ（32 ビット）は、ロングワードまたはクワッドワードサイズでメモリとのやりとりが可能です。倍精度浮動小数点データ（64 ビット）は、クワッドワードサイズでメモリとのやりとりが可能です。SH-4A がバイトサイズおよびワードサイズのデータをメモリからレジスタに移動するとデータは符号拡張されます。

3.1 実行環境

（1） PC

PC はその命令自身の命令アドレスを示します。

（2） ロード／ストアアーキテクチャ

SH-4A は基本的演算をレジスタで実行するロード／ストアアーキテクチャを特長としています。メモリで直接実行する論理 AND 演算のようなビット操作演算を除き、メモリアクセスを必要とする演算はレジスタにロードした後、レジスタで実行されます。

（3） 遅延分岐

SH-4A の分岐命令および RTE は、BF、BT の 2 つの分岐命令を除き遅延分岐です。遅延分岐では分岐命令の次の命令は分岐先命令の前に実行されます。

（4） 遅延スロット

遅延分岐後のこの実行スロットは「遅延スロット」と呼ばれます。たとえば、BRA 実行シーケンスは次のとおりです。

表 3.1 遅延分岐命令の実行順序

命令列				実行順序
	BRA	TARGET	（遅延分岐命令）	BRA
	ADD		（遅延スロット）	↓
	:			ADD
	:			↓
TARGET	target-inst		（分岐先命令）	target-inst

命令によっては遅延スロットで実行するとスロット不当命令例外を発生します。**「第 5 章 例外処理」**を参照してください。分岐が成立しなかった BF/S、BT/S の次の命令も遅延スロット命令です。

(5) Tビット

ステータスレジスタ（SR）のTビットは、比較演算の結果などを示すために使用し、条件付き分岐命令で参照します。たとえば、以下に条件付き分岐命令例を示します。

```
ADD     #1, R0          ;T ビットは ADD 演算で変更されません。
CMP/EQ  R1, R0          ;R0=R1 のとき T ビットは 1 にセットされる。
BT      TARGET          ;T ビット=1（R0=R1）のとき TARGET に分岐する。
```

RTE の遅延スロットで、ステータスレジスタ（SR）ビットは次のように参照されます。命令アクセスは変更の前に MD ビットを使用し、データアクセスは変更後の MD ビットにアクセスします。変更後の他のS、T、M、Q、FD、BL、RB ビットを遅延スロットの命令実行のために使用します。STC、STC.L SR 命令は、変更後すべての SR ビットにアクセスします。

(6) 定数値

8 ビットの定数値は命令コード、イミディエイト値で指定できます。また 16 ビット、32 ビットの定数値はメモリで定義することができ、PC 相対ロード命令で参照できます。

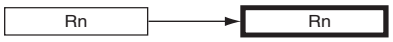
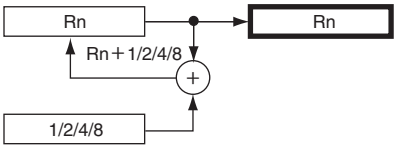
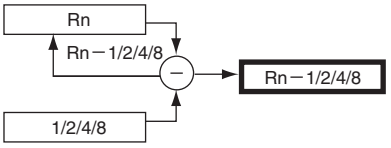
```
MOV.W   @(disp, PC), Rn
MOV.L   @(disp, PC), Rn
```

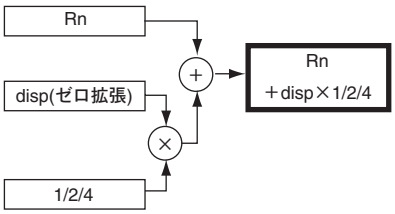
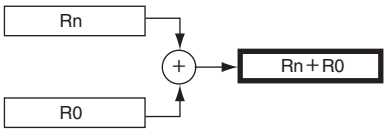
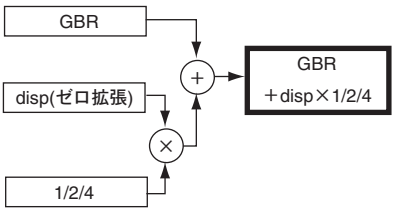
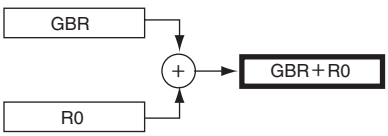
浮動小数点に対する PC 相対ロード命令はありません。ただし、単精度浮動小数点レジスタに対して FLDI0、FLDI1 命令を使用することによって、0.0 または 1.0 にセットすることができます。

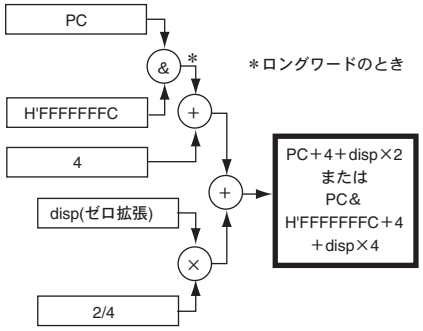
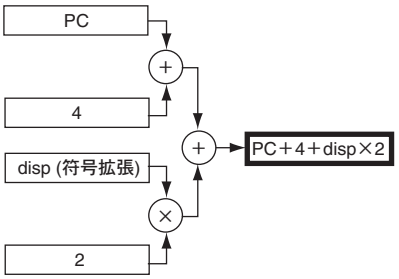
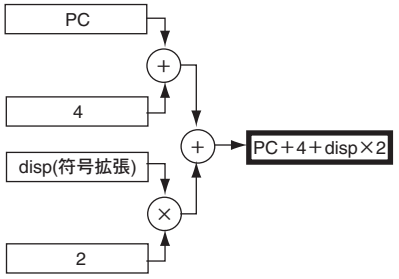
3.2 アドレッシングモード

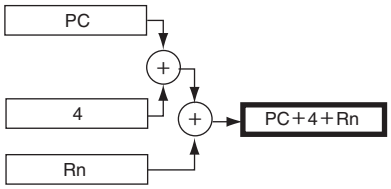
表 3.2 にアドレッシングモードと実効アドレスの計算を示します。仮想アドレス空間のある位置をアクセスすると（MMUCR.AT＝1）、実効アドレスは物理アドレスに変換されます。複数の仮想メモリ空間システムを選択した場合（MMUCR.SV＝0）、PTEH の最下位ビットもアクセスの ASID として参照されます。**「第 7 章　メモリマネジメントユニット（MMU）」**を参照してください。

表 3.2　アドレッシングモードと実効アドレス

アドレッシング モード	命令 フォーマット	実効アドレスの計算方法	計算式
レジスタ直接	Rn	実効アドレスはレジスタ Rn です。 （オペランドはレジスタ Rn の内容です。）	－
レジスタ間接	@Rn	実効アドレスはレジスタ Rn の内容です。 	Rn→EA（EA：実効アドレス）
ポストインクリメント レジスタ間接	@Rn＋	実効アドレスはレジスタ Rn の内容です。命令実行後 Rn に定数を加算します。定数はオペランドサイズがバイトのとき 1、ワードのとき 2、ロングワードのとき 4、クワッドワードのとき 8 です。 	Rn→EA 命令実行後 バイト：Rn＋1→Rn ワード：Rn＋2→Rn ロングワード：Rn+4→Rn クワッドワード：Rn+8→Rn
プリデクリメント レジスタ間接	@－Rn	実効アドレスは、あらかじめ定数を減算したレジスタ Rn の内容です。定数はバイトのとき 1、ワードのとき 2、ロングワードのとき 4、クワッドワードのとき 8 です。 	バイト：Rn－1→Rn ワード：Rn－2→Rn ロングワード：Rn－4→Rn クワッドワード：Rn－8→Rn Rn→EA （計算後の Rn で命令実行）

アドレッシング モード	命令 フォーマット	実効アドレスの計算方法	計算式
ディスプレースメント 付きレジスタ間接	@(disp:4, Rn)	実効アドレスはレジスタ Rn に 4 ビットディスプレースメント disp を加算した内容です。disp はゼロ拡張後、オペランドサイズによってバイトで 1 倍、ワードで 2 倍、ロングワードで 4 倍します。 	バイト：Rn＋disp→EA ワード：Rn＋disp×2→EA ロングワード：Rn＋disp×4→EA
インデックス付き レジスタ間接	@(R0, Rn)	実効アドレスはレジスタ Rn に R0 を加算した内容です。 	Rn＋R0→EA
ディスプレースメント 付き GBR 間接	@(disp:8, GBR)	実効アドレスはレジスタ GBR に 8 ビットディスプレースメント disp を加算した内容です。disp はゼロ拡張後、オペランドサイズによってバイトで 1 倍、ワードで 2 倍、ロングワードで 4 倍します。 	バイト：GBR＋disp→EA ワード：GBR＋disp×2→EA ロングワード：GBR＋disp×4→EA
インデックス付き GBR 間接	@(R0, GBR)	実効アドレスはレジスタ GBR に R0 を加算した内容です。 	GBR＋R0→EA

アドレッシング モード	命令 フォーマット	実効アドレスの計算方法	計算式
ディスプレースメント 付き PC 相対	@(disp:8, PC)	実効アドレスは PC＋4 に 8 ビットディスプレースメント disp を加算した内容です。disp はゼロ拡張後、オペランドサイズによってワードで 2 倍、ロングワードで 4 倍します。さらにロングワードのときは PC の下位 2 ビットをマスクします。 	ワード：PC＋4＋disp×2→EA ロングワード： PC & H'FFFFFFFC＋4＋disp×→EA
PC 相対	disp:8	実効アドレスは PC＋4 に 8 ビットディスプレースメント disp を符号拡張後 2 倍し、加算した内容です。 	PC＋4＋disp×2→Branch-Target
	disp:12	実効アドレスは PC＋4 に 12 ビットディスプレースメント disp を符号拡張後 2 倍し、加算した内容です。 	PC＋4＋disp×2→Branch-Target

アドレッシング モード	命令 フォーマット	実効アドレスの計算方法	計算式
PC 相対	Rn	実効アドレスは PC＋4 に Rn を加算した内容です。  	PC＋4＋Rn→Branch-Target
イミディエイト	#imm:8	TST, AND, OR, XOR 命令の 8 ビットイミディエイト imm はゼロ拡張します。	―
	#imm:8	MOV, ADD, CMP/EQ 命令の 8 ビットイミディエイト imm は符号拡張します。	―
	#imm:8	TRAPA 命令の 8 ビットイミディエイト imm はゼロ拡張後、4 倍します。	―

【注】 下記のディスプレースメント（disp）を伴うアドレッシングモードにおいて、本マニュアルのアセンブラ記述は、オペランドサイズに応じたスケーリング（×1、×2、×4）を行う前の値を書いています。これは、LSI の動作を明確にするためで、実際のアセンブラの記述は、各アセンブラの表記ルールを参照してください。

@（disp:4, Rn） ;ディスプレースメント付きレジスタ間接

@（disp:8, GBR） ;ディスプレースメント付き GBR 間接

@（disp:8, PC） ;ディスプレースメント付き PC 相対

disp: 8, disp:12 ;PC 相対

3.3 命令セット

表 3.4～表 3.13 に示す SuperH 命令の説明に使用する表記を表 3.3 に示します。

表 3.3 命令リストの表記

項目	フォーマット	説　　明
命令ニーモニック	OP.Sz SRC,DEST	OP：オペレーションコード Sz：サイズ SRC：ソースオペランド DEST：ソースおよび／またはデスティネーションオペランド Rm：ソースレジスタ Rn：デスティネーションレジスタ imm：イミディエイトデータ disp：ディスプレースメント
演算の要約		→、←：転送方向 (xx)：メモリオペランド M/Q/T：SR のフラグビット &：各ビットの論理積 |：各ビットの論理和 ^：各ビット排他的論理和 ~：各ビットの論理否定 <<n,>>n：n ビットシフト
命令コード	MSB←→LSB	mmmm：レジスタ番号（Rm、FRm） nnnn：レジスタ番号（Rn、FRn） 0000：R0, FR0 0001：R1, FR1 ： 1111：R15, FR15 mmm：レジスタ番号（DRm、XDm、Rm_BANK） nnn：レジスタ番号（DRn、XDn、Rn_BANK） 000：DR0、XD0、R0_BANK 001：DR2、XD2、R1_BANK ： 111：DR14、XD14、R7_BANK mm：レジスタ番号(FVm) nn：レジスタ番号(FVn)

項目	フォーマット	説　　明
命令コード	MSB←→LSB	00：FV0 01：FV4 10：FV8 11：FV12 iiii：イミディエイト値 dddd：ディスプレースメント
特権モード	−	「特権」と記載してある場合、特権モードでのみ実行可能です。
T ビット	命令実行後の T ビットの値	−：変更なし
新規	−	「新規」と記載してある場合は、SH-4A で新規に追加された命令です。

【注】　スケーリング（x1、x2、x4、x8）は命令オペランドのサイズに応じて実行されます。

表 3.4 固定小数点転送命令

命令		動作	命令コード	特権	T ビット	新規
MOV	#imm,Rn	imm→符号拡張→Rn	1110nnnniiiiiiii	—	—	—
MOV.W	@(disp*,PC),Rn	(disp×2+PC+4)→符号拡張→Rn	1001nnnnddddddd	—	—	—
MOV.L	@(disp*,PC),Rn	(disp×4+PC&H'FFFFFFFC+4)→Rn	1101nnnnddddddd	—	—	—
MOV	Rm,Rn	Rm→Rn	0110nnnnmmmm0011	—	—	—
MOV.B	Rm,@Rn	Rm→(Rn)	0010nnnnmmmm0000	—	—	—
MOV.W	Rm,@Rn	Rm→(Rn)	0010nnnnmmmm0001	—	—	—
MOV.L	Rm,@Rn	Rm→(Rn)	0010nnnnmmmm0010	—	—	—
MOV.B	@Rm,Rn	(Rm)→符号拡張→Rn	0110nnnnmmmm0000	—	—	—
MOV.W	@Rm,Rn	(Rm)→符号拡張→Rn	0110nnnnmmmm0001	—	—	—
MOV.L	@Rm,Rn	(Rm)→Rn	0110nnnnmmmm0010	—	—	—
MOV.B	Rm,@-Rn	Rn-1→Rn, Rm→(Rn)	0010nnnnmmmm0100	—	—	—
MOV.W	Rm,@-Rn	Rn-2→Rn, Rm→(Rn)	0010nnnnmmmm0101	—	—	—
MOV.L	Rm,@-Rn	Rn-4→Rn, Rm→(Rn)	0010nnnnmmmm0110	—	—	—
MOV.B	@Rm+,Rn	(Rm)→符号拡張→Rn, Rm+1→Rm	0110nnnnmmmm0100	—	—	—
MOV.W	@Rm+,Rn	(Rm)→符号拡張→Rn, Rm+2→Rm	0110nnnnmmmm0101	—	—	—
MOV.L	@Rm+,Rn	(Rm)→Rn, Rm+4→Rm	0110nnnnmmmm0110	—	—	—
MOV.B	R0,@(disp*,Rn)	R0→(disp+Rn)	10000000nnnndddd	—	—	—
MOV.W	R0,@(disp*,Rn)	R0→(disp×2+Rn)	10000001nnnndddd	—	—	—
MOV.L	Rm,@(disp*,Rn)	Rm→(disp×4+Rn)	0001nnnnmmmmdddd	—	—	—
MOV.B	@(disp*,Rm),R0	(disp+Rm)→符号拡張→R0	10000100mmmmdddd	—	—	—
MOV.W	@(disp*,Rm),R0	(disp×2+Rm)→符号拡張→R0	10000101mmmmdddd	—	—	—
MOV.L	@(disp*,Rm),Rn	(disp×4+Rm)→Rn	0101nnnnmmmmdddd	—	—	—
MOV.B	Rm,@(R0,Rn)	Rm→(R0+Rn)	0000nnnnmmmm0100	—	—	—
MOV.W	Rm,@(R0,Rn)	Rm→(R0+Rn)	0000nnnnmmmm0101	—	—	—
MOV.L	Rm,@(R0,Rn)	Rm→(R0+Rn)	0000nnnnmmmm0110	—	—	—
MOV.B	@(R0,Rm),Rn	(R0+Rm)→符号拡張→Rn	0000nnnnmmmm1100	—	—	—
MOV.W	@(R0,Rm),Rn	(R0+Rm)→符号拡張→Rn	0000nnnnmmmm1101	—	—	—
MOV.L	@(R0,Rm),Rn	(R0+Rm)→Rn	0000nnnnmmmm1110	—	—	—
MOV.B	R0,@(disp*,GBR)	R0→(disp+GBR)	11000000dddddddd	—	—	—
MOV.W	R0,@(disp*,GBR)	R0→(disp×2+GBR)	11000001dddddddd	—	—	—
MOV.L	R0,@(disp*,GBR)	R0→(disp×4+GBR)	11000010dddddddd	—	—	—
MOV.B	@(disp*,GBR),R0	(disp+GBR)→符号拡張→R0	11000100dddddddd	—	—	—
MOV.W	@(disp*,GBR),R0	(disp×2+GBR)→符号拡張→R0	11000101dddddddd	—	—	—
MOV.L	@(disp*,GBR),R0	(disp×4+GBR)→R0	11000110dddddddd	—	—	—
MOVA	@(disp*,PC),R0	disp×4+PC&H'FFFFFFFC+4→R0	11000111dddddddd	—	—	—

命令	動作	命令コード	特権	T ビット	新規
MOVCO.L R0,@Rn	LDST→T if(T==1)R0→(Rn) 0→LDST	0000nnnn01110011	—	LDST	新規
MOVLI.L @Rm,R0	1→LDST (Rm)→R0 ただし、割り込み／例外発生時 0→LDST	0000mmmm01100011	—	—	新規
MOVUA.L @Rm,R0	(Rm)→R0 非境界調整データのロード	0100mmmm10101001	—	—	新規
MOVUA.L @Rm+,R0	(Rm)→R0,Rm+4→Rm 非境界調整データのロード	0100mmmm11101001	—	—	新規
MOVT Rn	T→Rn	0000nnnn00101001	—	—	—
SWAP.B Rm,Rn	Rm→下位 2 バイトの 上下バイト交換→Rn	0110nnnnmmmm1000	—	—	—
SWAP.W Rm,Rn	Rm→上下ワード交換→Rn	0110nnnnmmmm1001	—	—	—
XTRCT Rm,Rn	Rm:Rn の中央 32 ビット→Rn	0010nnnnmmmm1101	—	—	—

【注】 * ルネサスのアセンブラでは、disp にスケーリング後（×1、×2、×4）の値を設定します。

表 3.5 算術演算命令

命令	動作	命令コード	特権	T ビット	新規
ADD Rm,Rn	Rn+Rm→Rn	0011nnnnmmmm1100	—	—	—
ADD #imm,Rn	Rn+imm→Rn	0111nnnniiiiiiii	—	—	—
ADDC Rm,Rn	Rn+Rm+T→Rn,キャリ→T	0011nnnnmmmm1110	—	キャリ	—
ADDV Rm,Rn	Rn+Rm→Rn,オーバフロー→T	0011nnnnmmmm1111	—	オーバフロー	—
CMP/EQ #imm,R0	R0=imm のとき 1→T それ以外のとき 0→T	10001000iiiiiiii	—	比較結果	—
CMP/EQ Rm,Rn	Rn=Rm のとき 1→T それ以外のとき 0→T	0011nnnnmmmm0000	—	比較結果	—
CMP/HS Rm,Rn	無符号で Rn≧Rm のとき 1→T それ以外のとき 0→T	0011nnnnmmmm0010	—	比較結果	—
CMP/GE Rm,Rn	有符号で Rn≧Rm のとき 1→T それ以外のとき 0→T	0011nnnnmmmm0011	—	比較結果	—
CMP/HI Rm,Rn	無符号で Rn>Rm のとき 1→T それ以外のとき 0→T	0011nnnnmmmm0110	—	比較結果	—
CMP/GT Rm,Rn	有符号で Rn>Rm のとき 1→T それ以外のとき 0→T	0011nnnnmmmm0111	—	比較結果	—
CMP/PZ Rn	Rn≧0 のとき 1→T それ以外のとき 0→T	0100nnnn00010001	—	比較結果	—

命令		動作	命令コード	特権	T ビット	新規
CMP/PL	Rn	Rn>0 のとき 1→T それ以外のとき 0→T	0100nnnn00010101	—	比較 結果	—
CMP/STR	Rm,Rn	いずれかのバイトが等しいとき 1→T それ以外のとき 0→T	0010nnnnmmmm1100	—	比較 結果	—
DIV1	Rm,Rn	1 ステップ除算(Rn÷Rm)	0011nnnnmmmm0100	—	計算 結果	—
DIV0S	Rm,Rn	Rn の MSB→Q, Rm の MSB→M, M^Q→T	0010nnnnmmmm0111	—	計算 結果	—
DIV0U		0→M/Q/T	0000000000011001	—	0	—
DMULS.L	Rm,Rn	符号付きで Rn×Rm→MAC, 32×32→64 ビット	0011nnnnmmmm1101	—	—	—
DMULU.L	Rm,Rn	符号なしで Rn×Rm→MAC, 32×32→64 ビット	0011nnnnmmmm0101	—	—	—
DT	Rn	Rn-1→Rn, Rn が 0 のとき 1→T Rn が 0 以外のとき 0→T	0100nnnn00010000	—	比較 結果	—
EXTS.B	Rm,Rn	Rm をバイトから符号拡張→Rn	0110nnnnmmmm1110	—	—	—
EXTS.W	Rm,Rn	Rm をワードから符号拡張→Rn	0110nnnnmmmm1111	—	—	—
EXTU.B	Rm,Rn	Rm をバイトからゼロ拡張→Rn	0110nnnnmmmm1100	—	—	—
EXTU.W	Rm,Rn	Rm をワードからゼロ拡張→Rn	0110nnnnmmmm1101	—	—	—
MAC.L	@Rm+,@Rn+	符号付きで (Rn)×(Rm)+MAC→MAC Rn+4→Rn, Rm+4→Rm 32×32＋64→64 ビット	0000nnnnmmmm1111	—	—	—
MAC.W	@Rm+,@Rn+	符号付きで (Rn)×(Rm)+MAC→MAC Rn+2→Rn, Rm+2→Rm 16×16+64→64 ビット	0100nnnnmmmm1111	—	—	—
MUL.L	Rm,Rn	Rn×Rm→MACL 32×32→32 ビット	0000nnnnmmmm0111	—	—	—
MULS.W	Rm,Rn	符号付きで Rn×Rm→MACL 16×16→32 ビット	0010nnnnmmmm1111	—	—	—
MULU.W	Rm,Rn	符号なしで Rn×Rm→MACL 16×16→32 ビット	0010nnnnmmmm1110	—	—	—
NEG	Rm,Rn	0-Rm→Rn	0110nnnnmmmm1011	—	—	—
NEGC	Rm,Rn	0-Rm-T→Rn, ボロー→T	0110nnnnmmmm1010	—	ボロー	—
SUB	Rm,Rn	Rn-Rm→Rn	0011nnnnmmmm1000	—	—	—
SUBC	Rm,Rn	Rn-Rm-T→Rn,ボロー→T	0011nnnnmmmm1010	—	ボロー	—
SUBV	Rm,Rn	Rn-Rm→Rn,アンダフロー→T	0011nnnnmmmm1011	—	アンダ フロー	—

表 3.6　論理演算命令

命令		動作	命令コード	特権	T ビット	新規
AND	Rm,Rn	Rn & Rm→Rn	0010nnnnmmmm1001	—	—	—
AND	#imm,R0	R0 & imm→R0	11001001iiiiiiii	—	—	—
AND.B	#imm,@(R0,GBR)	(R0+GBR) & imm→(R0+GBR)	11001101iiiiiiii	—	—	—
NOT	Rm,Rn	~Rm→Rn	0110nnnnmmmm0111	—	—	—
OR	Rm,Rn	Rn｜Rm→Rn	0010nnnnmmmm1011	—	—	—
OR	#imm,R0	R0｜imm→R0	11001011iiiiiiii	—	—	—
OR.B	#imm,@(R0,GBR)	(R0+GBR) | imm→(R0+GBR)	11001111iiiiiiii	—	—	—
TAS.B	@Rn	(Rn)が 0 のとき 1→T それ以外とき 0→T 両方に対して 1→(Rn)の MSB	0100nnnn00011011	—	テスト結果	—
TST	Rm,Rn	Rn & Rm,結果が 0 のとき 1→T それ以外のとき 0→T	0010nnnnmmmm1000	—	テスト結果	—
TST	#imm,R0	R0 & imm,結果が 0 のとき 1→T それ以外のとき 0→T	11001000iiiiiiii	—	テスト結果	—
TST.B	#imm,@(R0,GBR)	(R0+GBR)&imm, 結果が 0 のとき 1→T それ以外のとき 0→T	11001100iiiiiiii	—	テスト結果	—
XOR	Rm,Rn	Rn ^ Rm→Rn	0010nnnnmmmm1010	—	—	—
XOR	#imm,R0	R0 ^ imm→R0	11001010iiiiiiii	—	—	—
XOR.B	#imm,@(R0,GBR)	(R0+GBR) ^ imm→(R0+GBR)	11001110iiiiiiii	—	—	—

表 3.7　シフト命令

命令		動作	命令コード	特権	T ビット	新規
ROTL	Rn	T←Rn←MSB	0100nnnn00000100	—	MSB	—
ROTR	Rn	LSB→Rn→T	0100nnnn00000101	—	LSB	—
ROTCL	Rn	T←Rn←T	0100nnnn00100100	—	MSB	—
ROTCR	Rn	T→Rn→T	0100nnnn00100101	—	LSB	—
SHAD	Rm, Rn	Rm≧0 のとき Rn<<Rm→Rn, Rm<0 のとき Rn>>Rm→[MSB→Rn]	0100nnnnmmmm1100	—	—	—
SHAL	Rn	T←Rn←0	0100nnnn00100000	—	MSB	—
SHAR	Rn	MSB→Rn→T	0100nnnn00100001	—	LSB	—
SHLD	Rm, Rn	Rm≧0 のとき Rn<<Rm→Rn, Rm<0 のとき Rn>>Rm→[0→Rn]	0100nnnnmmmm1101	—	—	—
SHLL	Rn	T←Rn←0	0100nnnn00000000	—	MSB	—
SHLR	Rn	0→Rn→T	0100nnnn00000001	—	LSB	—
SHLL2	Rn	Rn<<2→Rn	0100nnnn00001000	—	—	—
SHLR2	Rn	Rn>>2→Rn	0100nnnn00001001	—	—	—
SHLL8	Rn	Rn<<8→Rn	0100nnnn00011000	—	—	—
SHLR8	Rn	Rn>>8→Rn	0100nnnn00011001	—	—	—
SHLL16	Rn	Rn<<16→Rn	0100nnnn00101000	—	—	—
SHLR16	Rn	Rn>>16→Rn	0100nnnn00101001	—	—	—

表 3.8　分岐命令

命令		動作	命令コード	特権	T ビット	新規
BF	label	T=0 のとき disp×2+PC+4→PC, T=1 のとき nop	10001011dddddddd	—	—	—
BF/S	label	遅延分岐,T=0 のとき disp×2+PC+4→PC, T=1 のとき nop	10001111dddddddd	—	—	—
BT	label	T=1 のとき disp×2+PC+4→PC, T=0 のとき nop	10001001dddddddd	—	—	—
BT/S	label	遅延分岐,T=1 のとき disp×2+PC+4→PC, T=0 のとき nop	10001101dddddddd	—	—	—
BRA	label	遅延分岐,disp×2+PC+4→PC	1010dddddddddddd	—	—	—
BRAF	Rn	遅延分岐,Rn+PC+4→PC	0000nnnn00100011	—	—	—
BSR	label	遅延分岐,PC+4→PR, disp×2+PC+4→PC	1011dddddddddddd	—	—	—
BSRF	Rn	遅延分岐,PC+4→PR, Rn+PC+4→PC	0000nnnn00000011	—	—	—
JMP	@Rn	遅延分岐,Rn→PC	0100nnnn00101011	—	—	—
JSR	@Rn	遅延分岐,PC+4→PR,Rn→PC	0100nnnn00001011	—	—	—
RTS		遅延分岐,PR→PC	0000000000001011	—	—	—

表 3.9 システム制御命令

命令		動作	命令コード	特権	T ビット	新規
CLRMAC		0→MACH,MACL	0000000000101000	—	—	—
CLRS		0→S	0000000001001000	—	—	—
CLRT		0→T	0000000000001000	—	0	—
ICBI	@Rn	論理アドレス Rn で示される命令キャッシュを無効化	0000nnnn11100011	—	—	新規
LDC	Rm,SR	Rm→SR	0100mmmm00001110	特権	LSB	—
LDC	Rm,GBR	Rm→GBR	0100mmmm00011110	—	—	—
LDC	Rm,VBR	Rm→VBR	0100mmmm00101110	特権	—	—
LDC	Rm,SGR	Rm→SGR	0100mmmm00111010	特権	—	新規
LDC	Rm,SSR	Rm→SSR	0100mmmm00111110	特権	—	—
LDC	Rm,SPC	Rm→SPC	0100mmmm01001110	特権	—	—
LDC	Rm,DBR	Rm→DBR	0100mmmm11111010	特権	—	—
LDC	Rm,Rn_BANK	Rm→Rn_BANK(n=0～7)	0100mmmm1nnn1110	特権	—	—
LDC.L	@Rm+,SR	(Rm)→SR,Rm+4→Rm	0100mmmm00000111	特権	LSB	—
LDC.L	@Rm+,GBR	(Rm)→GBR,Rm+4→Rm	0100mmmm00010111	—	—	—
LDC.L	@Rm+,VBR	(Rm)→VBR,Rm+4→Rm	0100mmmm00100111	特権	—	—
LDC.L	@Rm+,SGR	(Rm)→SGR,Rm+4→Rm	0100mmmm00110110	特権	—	新規
LDC.L	@Rm+,SSR	(Rm)→SSR,Rm+4→Rm	0100mmmm00110111	特権	—	—
LDC.L	@Rm+,SPC	(Rm)→SPC,Rm+4→Rm	0100mmmm01000111	特権	—	—
LDC.L	@Rm+,DBR	(Rm)→DBR,Rm+4→Rm	0100mmmm11110110	特権	—	—
LDC.L	@Rm+,Rn_BANK	(Rm)→Rn_BANK,Rm+4→Rm	0100mmmm1nnn0111	特権	—	—
LDS	Rm,MACH	Rm→MACH	0100mmmm00001010	—	—	—
LDS	Rm,MACL	Rm→MACL	0100mmmm00011010	—	—	—
LDS	Rm,PR	Rm→PR	0100mmmm00101010	—	—	—
LDS.L	@Rm+,MACH	(Rm)→MACH,Rm+4→Rm	0100mmmm00000110	—	—	—
LDS.L	@Rm+,MACL	(Rm)→MACL,Rm+4→Rm	0100mmmm00010110	—	—	—
LDS.L	@Rm+,PR	(Rm)→PR,Rm+4→Rm	0100mmmm00100110	—	—	—
LDTLB		PTEH/PTEL→TLB	0000000000111000	特権	—	—
MOVCA.L	R0,@Rn	(キャッシュブロックをフェッチせずに)R0→(Rn)	0000nnnn11000011	—	—	—
NOP		無操作	0000000000001001	—	—	—
OCBI	@Rn	オペランドキャッシュブロックを無効にする	0000nnnn10010011	—	—	—
OCBP	@Rn	オペランドキャッシュブロックをライトバックし無効にする	0000nnnn10100011	—	—	—

命令		動作	命令コード	特権	T ビット	新規
OCBWB	@Rn	オペランドキャッシュブロックをライトバックする	0000nnnn10110011	—	—	—
PREF	@Rn	(Rn)→オペランドキャッシュ	0000nnnn10000011	—	—	—
PREFI	@Rn	32 バイトの命令ブロックを命令キャッシュに読み込む	0000nnnn11010011	—	—	新規
RTE		遅延分岐,SSR/SPC→SR/PC	0000000000101011	特権	—	—
SETS		1→S	0000000001011000	—	—	—
SETT		1→T	0000000000011000	—	1	—
SLEEP		スリープもしくはスタンバイ	0000000000011011	特権	—	—
STC	SR,Rn	SR→Rn	0000nnnn00000010	特権	—	—
STC	GBR,Rn	GBR→Rn	0000nnnn00010010	—	—	—
STC	VBR,Rn	VBR→Rn	0000nnnn00100010	特権	—	—
STC	SSR,Rn	SSR→Rn	0000nnnn00110010	特権	—	—
STC	SPC,Rn	SPC→Rn	0000nnnn01000010	特権	—	—
STC	SGR,Rn	SGR→Rn	0000nnnn00111010	特権	—	—
STC	DBR,Rn	DBR→Rn	0000nnnn11111010	特権	—	—
STC	Rm_BANK,Rn	Rm_BANK→Rn(m=0～7)	0000nnnn1mmm0010	特権	—	—
STC.L	SR,@-Rn	Rn-4→Rn,SR→(Rn)	0100nnnn00000011	特権	—	—
STC.L	GBR,@-Rn	Rn-4→Rn,GBR→(Rn)	0100nnnn00010011	—	—	—
STC.L	VBR,@-Rn	Rn-4→Rn,VBR→(Rn)	0100nnnn00100011	特権	—	—
STC.L	SSR,@-Rn	Rn-4→Rn,SSR→(Rn)	0100nnnn00110011	特権	—	—
STC.L	SPC,@-Rn	Rn-4→Rn,SPC→(Rn)	0100nnnn01000011	特権	—	—
STC.L	SGR,@-Rn	Rn-4→Rn,SGR→(Rn)	0100nnnn00110010	特権	—	—
STC.L	DBR,@-Rn	Rn-4→Rn,DBR→(Rn)	0100nnnn11110010	特権	—	—
STC.L	Rm_BANK,@-Rn	Rn-4→Rn,Rm_BANK→(Rn) (m=0～7)	0100nnnn1mmm0011	特権	—	—
STS	MACH,Rn	MACH→Rn	0000nnnn00001010	—	—	—
STS	MACL,Rn	MACL→Rn	0000nnnn00011010	—	—	—
STS	PR,Rn	PR→Rn	0000nnnn00101010	—	—	—
STS.L	MACH,@-Rn	Rn-4→Rn,MACH→(Rn)	0100nnnn00000010	—	—	—
STS.L	MACL,@-Rn	Rn-4→Rn,MACL→(Rn)	0100nnnn00010010	—	—	—
STS.L	PR,@-Rn	Rn-4→Rn,PR→(Rn)	0100nnnn00100010	—	—	—
SYNCO		本命令以前のデータ操作を完了するまで、本命令以降の命令を開始しない	0000000010101011	—	—	新規
TRAPA	#imm	imm<<2→TRA,PC+2→SPC, SR→SSR,R15→SGR, 1→SR.MD/BL/RB,H'160→EXPEVT, VBR+H'0100→PC	11000011iiiiiiii	—	—	—

表 3.10 浮動小数点単精度命令

命令		動作	命令コード	特権	T ビット	新規
FLDI0	FRn	H'00000000→FRn	1111nnnn10001101	—	—	—
FLDI1	FRn	H'3F800000→FRn	1111nnnn10011101	—	—	—
FMOV	FRm,FRn	FRm→FRn	1111nnnnmmmm1100	—	—	—
FMOV.S	@Rm,FRn	(Rm)→FRn	1111nnnnmmmm1000	—	—	—
FMOV.S	@(R0,Rm),FRn	(R0+Rm)→FRn	1111nnnnmmmm0110	—	—	—
FMOV.S	@Rm+,FRn	(Rm)→FRn,Rm+4→Rm	1111nnnnmmmm1001	—	—	—
FMOV.S	FRm,@Rn	FRm→(Rn)	1111nnnnmmmm1010	—	—	—
FMOV.S	FRm,@-Rn	Rn-4→Rn,FRm→(Rn)	1111nnnnmmmm1011	—	—	—
FMOV.S	FRm,@(R0,Rn)	FRm→(R0+Rn)	1111nnnnmmmm0111	—	—	—
FMOV	DRm,DRn	DRm→DRn	1111nnn0mmm01100	—	—	—
FMOV	@Rm,DRn	(Rm)→DRn	1111nnn0mmmm1000	—	—	—
FMOV	@(R0,Rm),DRn	(R0+Rm)→DRn	1111nnn0mmmm0110	—	—	—
FMOV	@Rm+,DRn	(Rm)→DRn,Rm+8→Rm	1111nnn0mmmm1001	—	—	—
FMOV	DRm,@Rn	DRm→(Rn)	1111nnnnmmm01010	—	—	—
FMOV	DRm,@-Rn	Rn-8→Rn,DRm→(Rn)	1111nnnnmmm01011	—	—	—
FMOV	DRm,@(R0,Rn)	DRm→(R0+Rn)	1111nnnnmmm00111	—	—	—
FLDS	FRm,FPUL	FRm→FPUL	1111mmmm00011101	—	—	—
FSTS	FPUL,FRn	FPUL→FRn	1111nnnn00001101	—	—	—
FABS	FRn	FRn & H'7FFF FFFF→FRn	1111nnnn01011101	—	—	—
FADD	FRm,FRn	FRn+FRm→FRn	1111nnnnmmmm0000	—	—	—
FCMP/EQ	FRm,FRn	FRn=FRm のとき 1→T それ以外のとき 0→T	1111nnnnmmmm0100	—	比較結果	—
FCMP/GT	FRm,FRn	FRn>FRm のとき 1→T それ以外のとき 0→T	1111nnnnmmmm0101	—	比較結果	—
FDIV	FRm,FRn	FRn/FRm→FRn	1111nnnnmmmm0011	—	—	—
FLOAT	FPUL,FRn	(float)FPUL→FRn	1111nnnn00101101	—	—	—
FMAC	FR0,FRm,FRn	FR0×FRm+FRn→FRn	1111nnnnmmmm1110	—	—	—
FMUL	FRm,FRn	FRn×FRm→FRn	1111nnnnmmmm0010	—	—	—
FNEG	FRn	FRn ^ H'80000000→FRn	1111nnnn01001101	—	—	—
FSQRT	FRn	sqrt(FRn)→FRn*	1111nnnn01101101	—	—	—
FSUB	FRm,FRn	FRn - FRm→FRn	1111nnnnmmmm0001	—	—	—
FTRC	FRm,FPUL	(long)FRm→FPUL	1111mmmm00111101	—	—	—

【注】 * sqrt(FR n)は FRn の平方根を表します。

表 3.11 浮動小数点倍精度命令

命令		動作	命令コード	特権	T ビット	新規
FABS	DRn	DRn&H'7FFF FFFF FFFF FFFF→DRn	1111nnn001011101	—	—	—
FADD	DRm,DRn	DRn+DRm→DRn	1111nnn0mmm00000	—	—	—
FCMP/EQ	DRm,DRn	DRn=DRm のとき 1→T それ以外のとき 0→T	1111nnn0mmm00100	—	比較結果	—
FCMP/GT	DRm,DRn	DRn>DRm のとき 1→T それ以外のとき 0→T	1111nnn0mmm00101	—	比較結果	—
FDIV	DRm,DRn	DRn/DRm→DRn	1111nnn0mmm00011	—	—	—
FCNVDS	DRm,FPUL	double_to_float(DRm)→FPUL	1111mmm010111101	—	—	—
FCNVSD	FPUL,DRn	float_to_double(FPUL)→DRn	1111nnn010101101	—	—	—
FLOAT	FPUL,DRn	(float)FPUL→DRn	1111nnn000101101	—	—	—
FMUL	DRm,DRn	DRn×DRm→DRn	1111nnn0mmm00010	—	—	—
FNEG	DRn	DRn ^ H'8000 0000 0000 0000→DRn	1111nnn001001101	—	—	—
FSQRT	DRn	sqrt(DRn)→DRn*	1111nnn001101101	—	—	—
FSUB	DRm,DRn	DRn - DRm→DRn	1111nnn0mmm00001	—	—	—
FTRC	DRm,FPUL	(long)DRm→FPUL	1111mmm000111101	—	—	—

【注】 * sqrt(DRn)は DRn の平方根を表します。

表 3.12 浮動小数点制御命令

命令		動作	命令コード	特権	T ビット	新規
LDS	Rm,FPSCR	Rm→FPSCR	0100mmmm01101010	—	—	—
LDS	Rm,FPUL	Rm→FPUL	0100mmmm01011010	—	—	—
LDS.L	@Rm+,FPSCR	(Rm)→FPSCR,Rm+4→Rm	0100mmmm01100110	—	—	—
LDS.L	@Rm+,FPUL	(Rm)→FPUL,Rm+4→Rm	0100mmmm01010110	—	—	—
STS	FPSCR,Rn	FPSCR→Rn	0000nnnn01101010	—	—	—
STS	FPUL,Rn	FPUL→Rn	0000nnnn01011010	—	—	—
STS.L	FPSCR,@-Rn	Rn-4→Rn,FPSCR→(Rn)	0100nnnn01100010	—	—	—
STS.L	FPUL,@-Rn	Rn-4→Rn,FPUL→(Rn)	0100nnnn01010010	—	—	—

表 3.13 浮動小数点グラフィック強化命令

命令		動作	命令コード	特権	T ビット	新規
FMOV	DRm,XDn	DRm→XDn	1111nnn1mmm01100	—	—	—
FMOV	XDm,DRn	XDm→DRn	1111nnn0mmm11100	—	—	—
FMOV	XDm,XDn	XDm→XDn	1111nnn1mmm11100	—	—	—
FMOV	@Rm,XDn	(Rm)→XDn	1111nnn1mmmm1000	—	—	—
FMOV	@Rm+,XDn	(Rm)→XDn,Rm+8→Rm	1111nnn1mmmm1001	—	—	—
FMOV	@(R0,Rm),XDn	(R0+Rm)→XDn	1111nnn1mmmm0110	—	—	—
FMOV	XDm,@Rn	XDm→(Rn)	1111nnnnmmm11010	—	—	—
FMOV	XDm,@-Rn	Rn-8→Rn,XDm→(Rn)	1111nnnnmmm11011	—	—	—
FMOV	XDm,@(R0,Rn)	XDm→(R0+Rn)	1111nnnnmmm10111	—	—	—
FIPR	FVm,FVn	inner_product(FVm,FVn) →FR[n+3]	1111nnmm11101101	—	—	—
FTRV	XMTRX,FVn	transform_vector(XMTRX,FVn) →FVn	1111nn0111111101	—	—	—
FRCHG		～FRSCR.FR→FRSCR.FR	1111101111111101	—	—	—
FSCHG		～FPSCR.SZ→FPSCR.SZ	1111001111111101	—	—	—
FPCHG		～FPSCR.PR→FPSCR.PR	1111011111111101	—	—	新規
FSRRA	FRn	1/sqrt(FRn)→FRn*	1111nnnn01111101	—	—	新規
FSCA	FPUL,DRn	sin(FPUL)→FRn cos(FPUL)→FR[n+1]	1111nnn011111101	—	—	新規

【注】 * sqrt(FRn)は FRn の平方根を表します。

4. パイプライン動作

SH-4A は 2 命令並列型（2-ILP, Instruction-Level-Parallelism）のスーパスカラパイプライン処理マイクロプロセッサです。命令実行はパイプライン化され、2 つの命令を並行して実行できます。

4.1 パイプライン

図 4.1 に基本パイプラインを示します。通常、パイプラインは命令フェッチ（I1、I2）、デコード・レジスタリード（ID）、実行（E1、E2、E3）、ライトバック（WB）の 7 ステージから構成されます。1 つの命令は基本パイプラインの組み合わせとして実行されます。

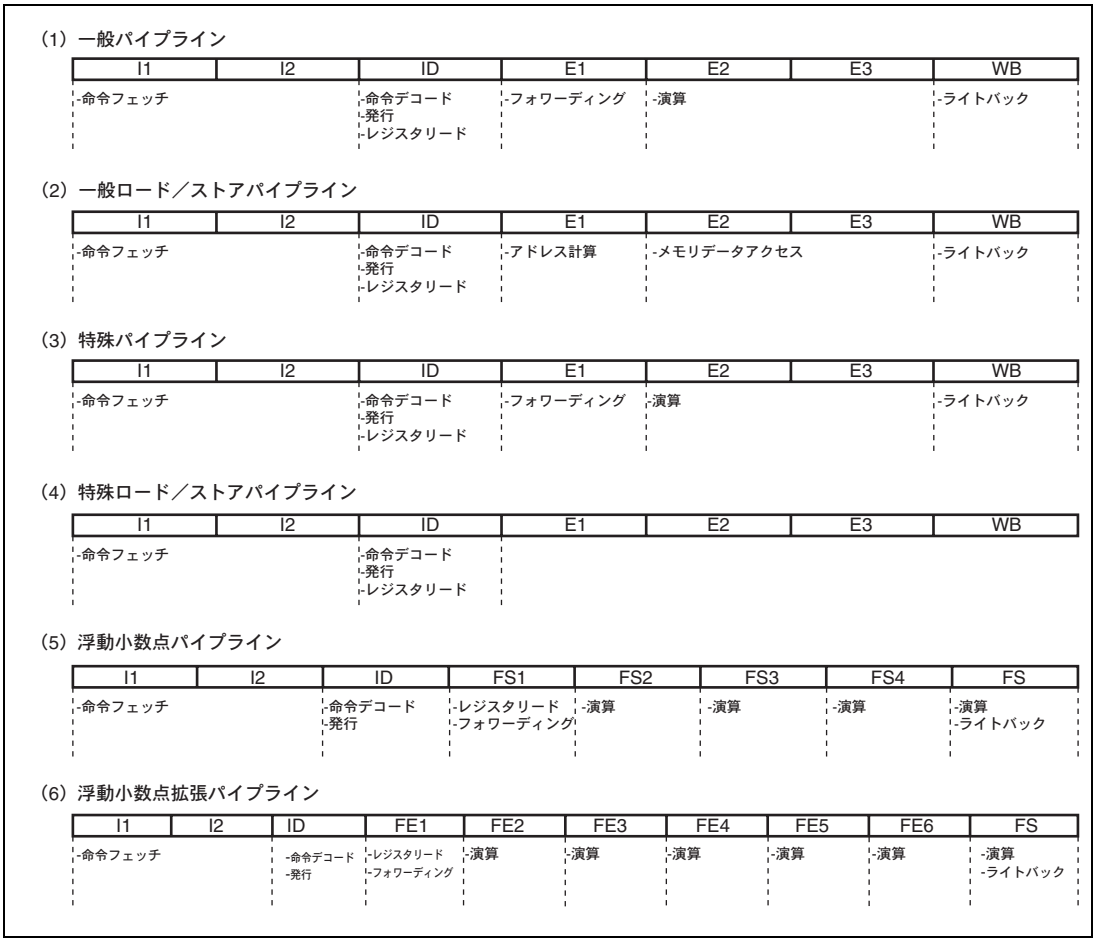


図 4.1　基本パイプライン

図 4.2 に命令実行パターンを示します。図 4.2 で使用する表記とその意味を以下に示します。

表 4.1　命令実行パターン表記説明

表　　記	意　　味
E1 E2 E3 WB	CPU EX パイプ占有
S1 S2 S3 WB	CPU LS パイプ占有 （メモリアクセスを伴う場合）
s1 s2 s3 WB	CPU LS パイプ占有 （メモリアクセスを伴わない場合）
E1/S1	CPU EX か LS の いずれか一方を占有
E1S1 、E1s1	CPU EX と LS の 両方を占有
M2 M3 MS	CPU MULT 演算器占有
FE1 FE2 FE3 FE4 FE5 FE6 FS	FPU-EX パイプ占有
FS1 FS2 FS3 FS4 FS	FPU-LS パイプ占有
ID	ID ステージをロック
└──	CPU と FPU 両方のパイプを占有

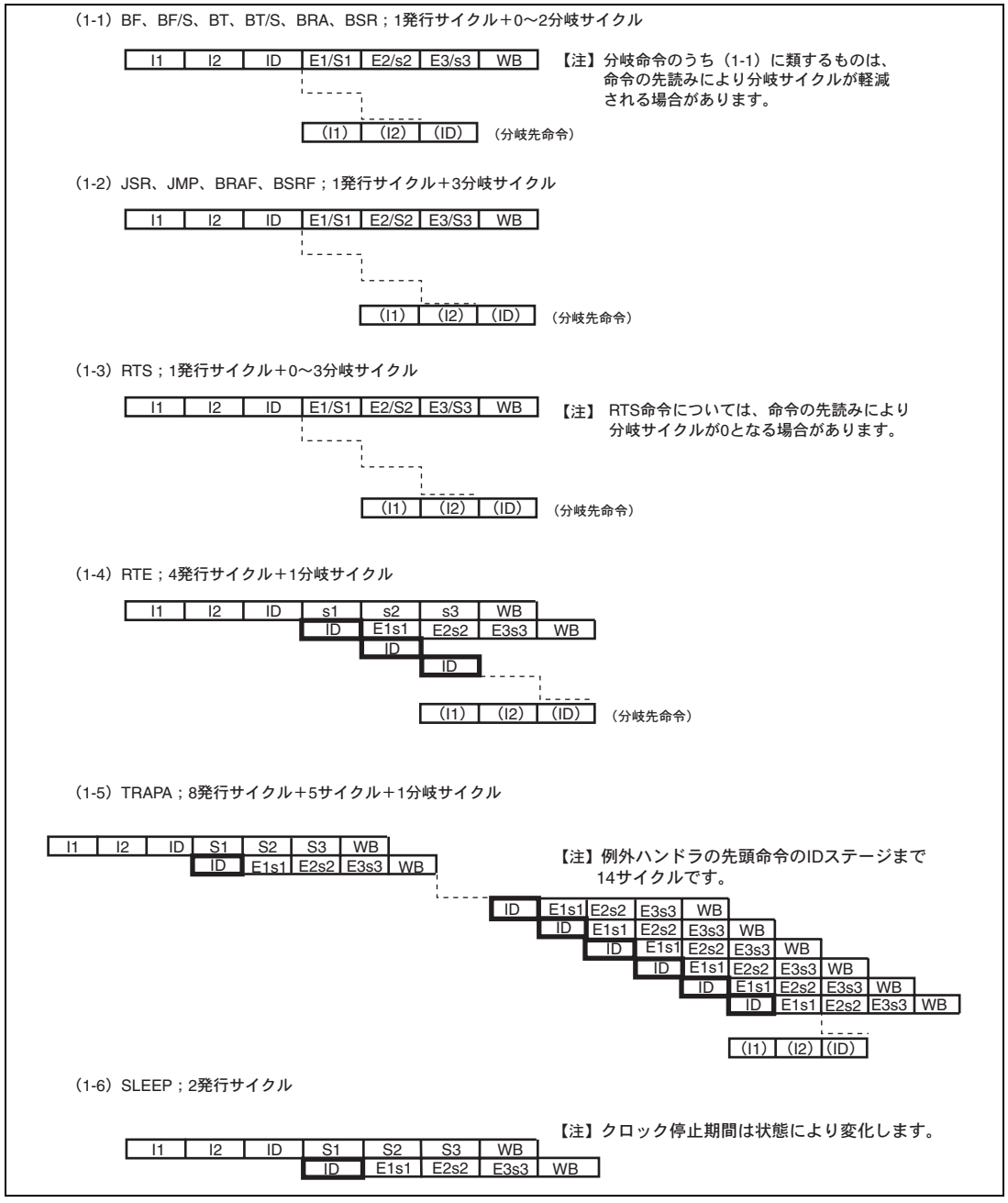


図 4.2　命令実行パターン（1）

（2-1）1ステップ演算（EXタイプ）；1発行サイクル

EXT[SU].[BW]、MOVT、SWAP、XTRCT、ADD*、CMP*、DIV*、DT、NEG*、SUB*、AND、AND#、NOT、OR、OR#、TST、TST#、XOR、XOR#、ROT*、SHA*、SHL*、CLRS、CLRT、SETS、SETT

【注】AND#、OR#、TST#、XOR#はGBR相対命令を除く

I1	I2	ID	E1	E2	E3	WB

（2-2）1ステップ演算（LSタイプ）；1発行サイクル

MOVA

I1	I2	ID	s1	s2	s3	WB

（2-3）1ステップ演算（MTタイプ）；1発行サイクル

MOV#、NOP

I1	I2	ID	E1/S1	E2/s2	E3/s3	WB

（2-4）MOV（MTタイプ）；1発行サイクル

MOV

I1	I2	ID	E1/s1	E2/s2	E3/S3	WB

図 4.2　命令実行パターン（2）

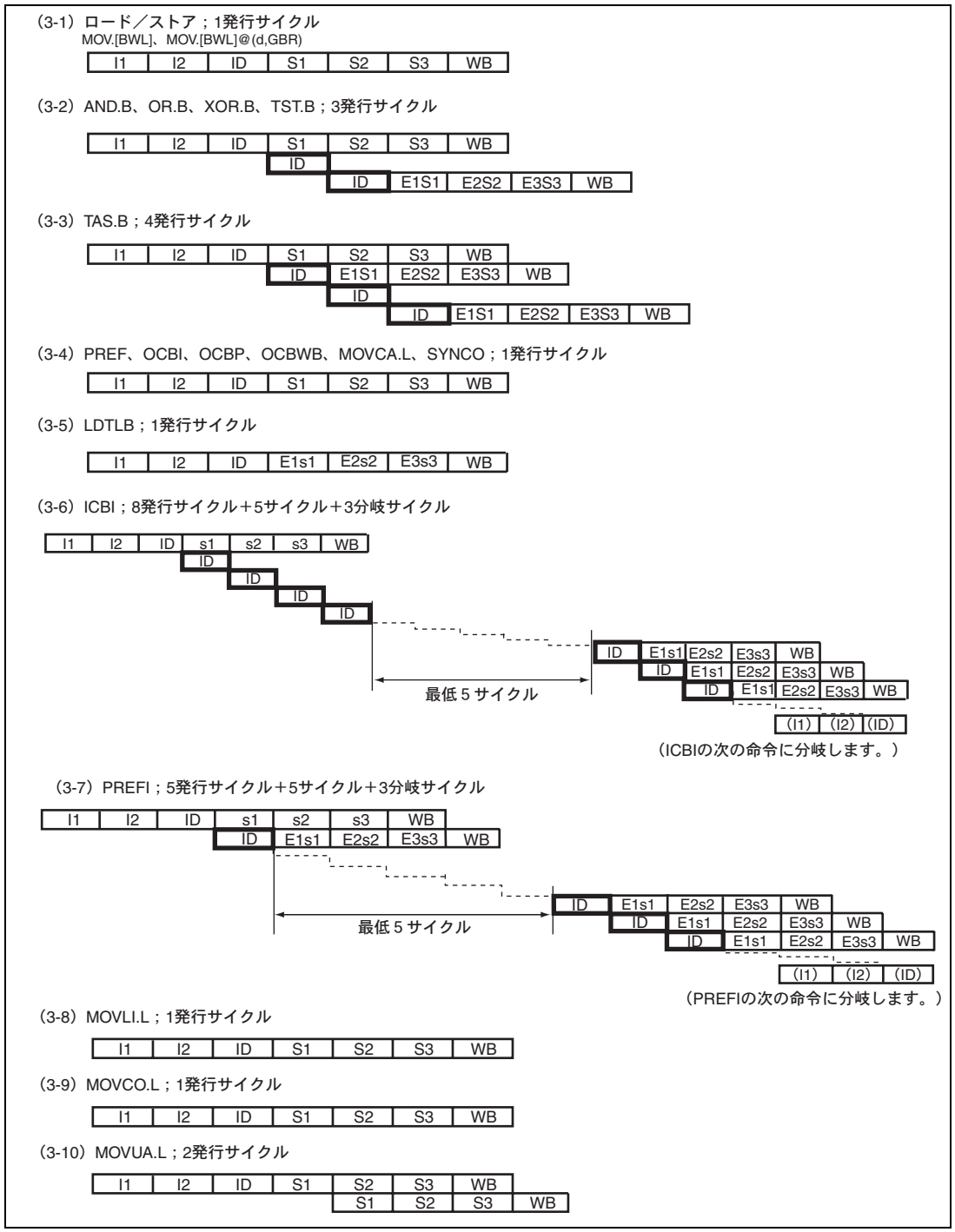


図 4.2　命令実行パターン（3）

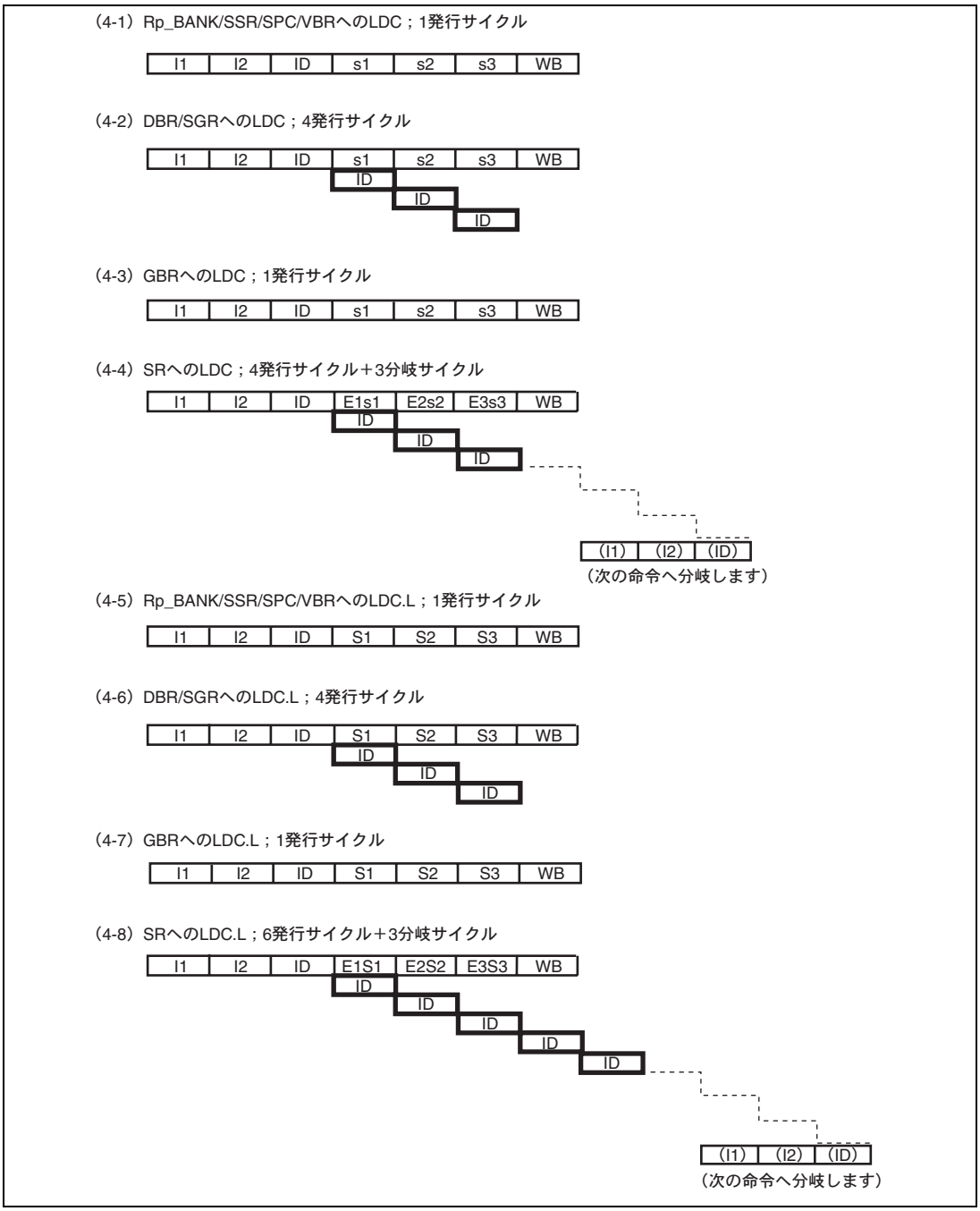


図 4.2　命令実行パターン（4）

（4-9）DBR/GBR/Rp_BANK/SSR/SPC/VBR/SGRからのSTC；1発行サイクル

I1	I2	ID	s1	s2	s3	WB

（4-10）SRからのSTC；1発行サイクル

I1	I2	ID	E1s1	E2s2	E3s3	WB

（4-11）DBR/GBR/Rp_BANK/SSR/SPC/VBR/SGRからのSTC.L；1発行サイクル

I1	I2	ID	S1	S2	S3	WB

（4-12）SRからのSTC.L；1発行サイクル

I1	I2	ID	E1S1	E2S2	E3S3	WB

（4-13）PRへのLDS；1発行サイクル

I1	I2	ID	s1	s2	s3	WB

（4-14）PRへのLDS.L；1発行サイクル

I1	I2	ID	S1	S2	S3	WB

（4-15）PRからのSTS；1発行サイクル

I1	I2	ID	s1	s2	s3	WB

（4-16）PRからのSTS.L；1発行サイクル

I1	I2	ID	S1	S2	S3	WB

（4-17）BSRF、BSR、JSRの遅延スロット命令（PRセット）；0発行サイクル

(I1)	(I2)	(ID)	(??1)	(??2)	(??3)	(WB)

【注】遅延スロット命令のE3ステージでPRの値が更新されます。
遅延スロットにPRからのSTS、STS.L命令が使用されている場合、
更新されたPRの値が使用されます。

図 4.2　命令実行パターン（5）

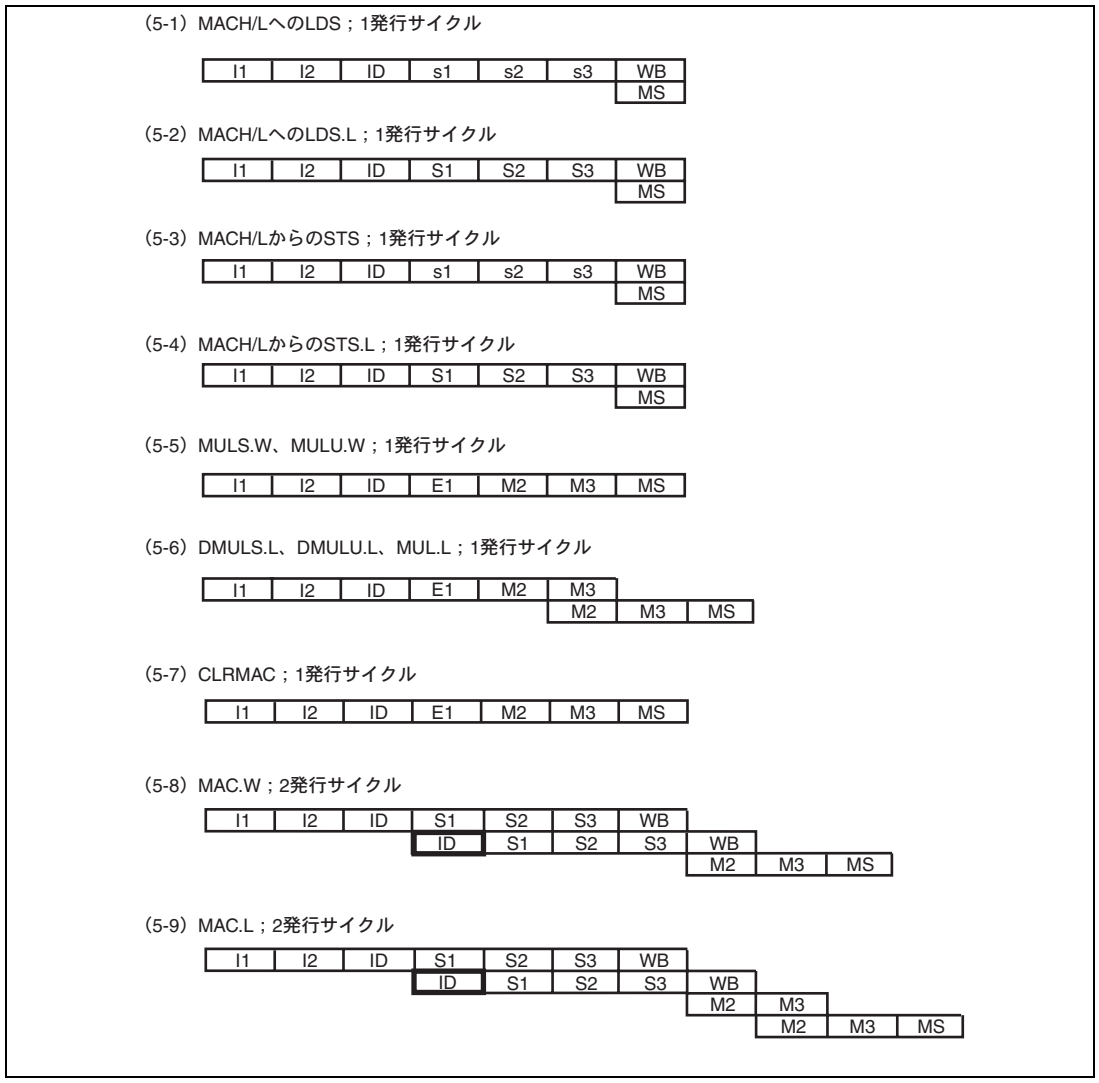


図 4.2　命令実行パターン（6）

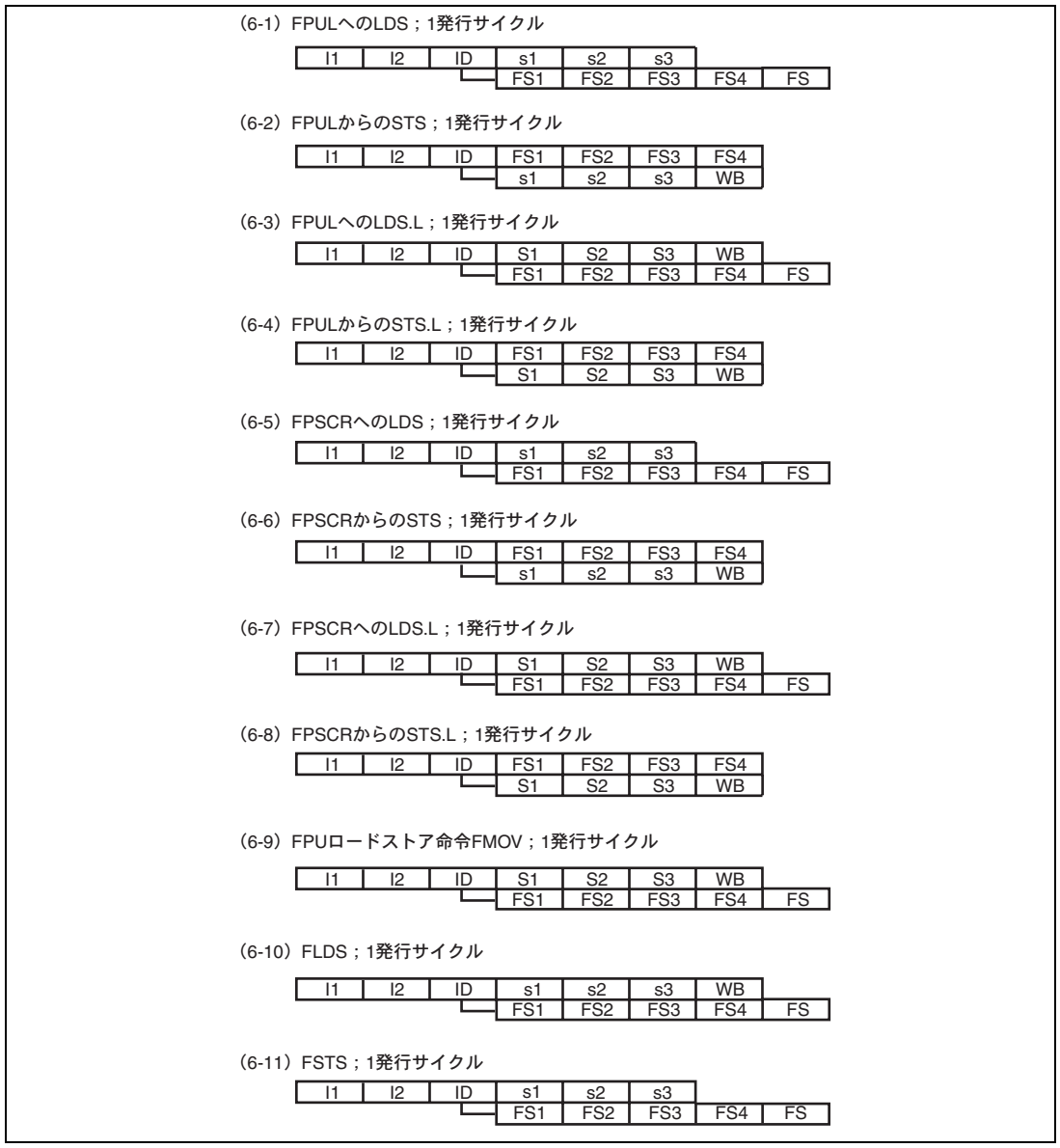


図 4.2　命令実行パターン（7）

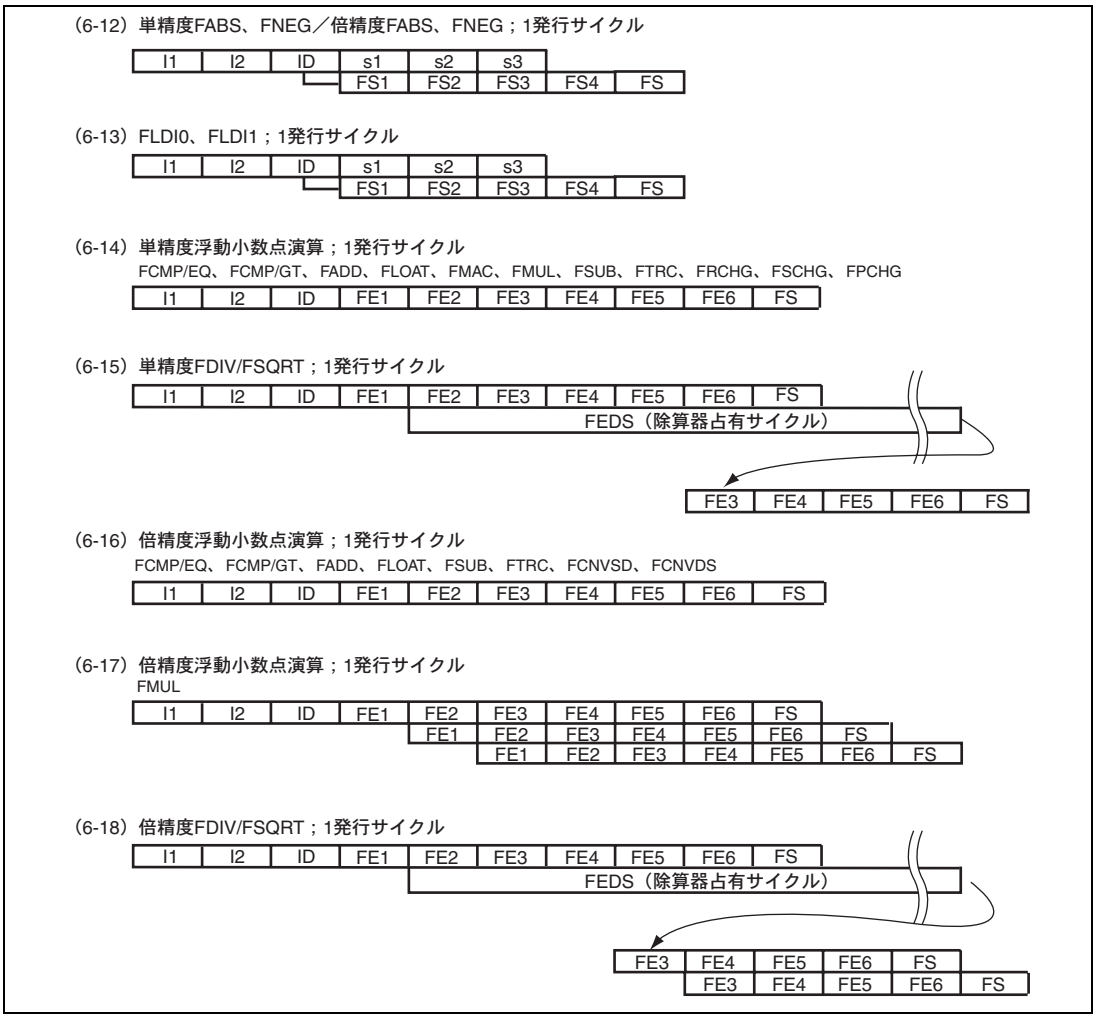


図 4.2　命令実行パターン（8）

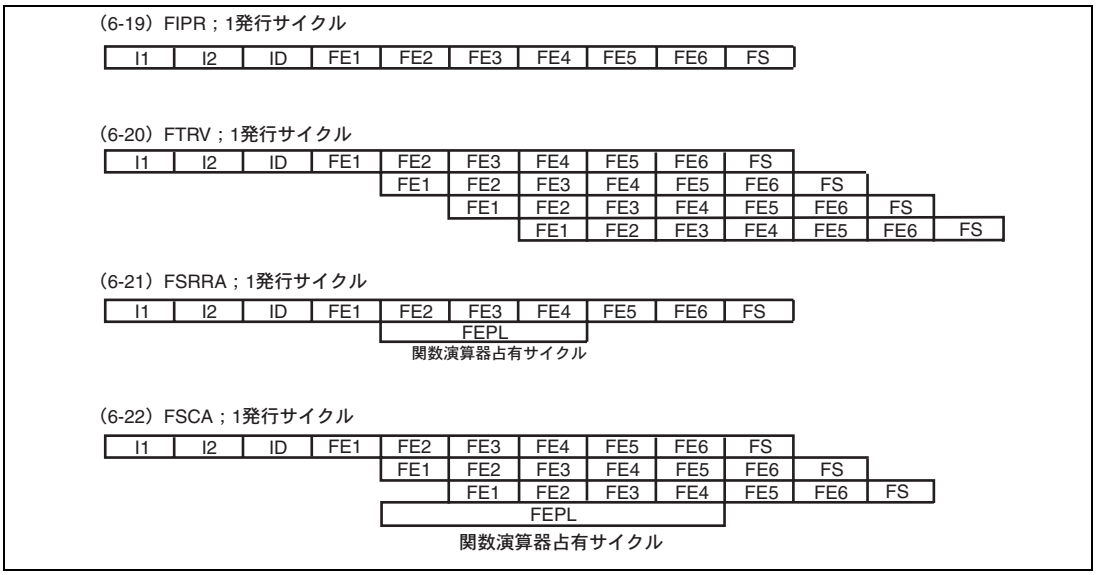


図 4.2　命令実行パターン（9）

4.2 並列実行性

命令は利用する内部機能ブロックにより、**表 4.2** に示すようなグループに分類されます。**表 4.3** に並列実行可能な 2 つの命令の組み合わせをグループごとに示します。たとえば、EX グループに分類された ADD と BR グループの BRA は並列実行できます。

表 4.2 命令グループ

命令グループ	命令			
EX	ADD	DT	ROTL	SHLR8
	ADDC	EXTS	ROTR	SHLR16
	ADDV	EXTU	SETS	SUB
	AND #imm,R0	MOVT	SETT	SUBC
	AND Rm,Rn	MUL.L	SHAD	SUBV
	CLRMAC	MULS.W	SHAL	SWAP
	CLRS	MULU.W	SHAR	TST #imm,R0
	CLRT	NEG	SHLD	TST Rm,Rn
	CMP	NEGC	SHLL	XOR #imm,R0
	DIV0S	NOT	SHLL2	XOR Rm,Rn
	DIV0U	OR #imm,R0	SHLL8	XTRCT
	DIV1	OR Rm,Rn	SHLL16	
	DMUS.L	ROTCL	SHLR	
	DMULU.L	ROTCR	SHLR2	
MT	MOV #imm,Rn	MOV Rm,Rn	NOP	
BR	BF	BRAF	BT	JSR
	BF/S	BSR	BT/S	RTS
	BRA	BSRF	JMP	
LS	FABS	FMOV.S FR,@adr	MOV.[BWL] @adr,R	STC CR2,Rn
	FNEG	FSTS	MOV.[BWL] R,@adr	STC.L CR2,@-Rn
	FLDI0	LDC Rm,CR1	MOVA	STS SR2,Rn
	FLDI1	LDC.L @Rm+,CR1	MOVCA.L	STS.L SR2,@-Rn
	FLDS	LDS Rm,SR1	MOVUA	STS SR1,Rn
	FMOV @adr,FR	LDS Rm,SR2	OCBI	STS.L SR1,@-Rn
	FMOV FR,@adr	LDS.L @adr,SR2	OCBP	
	FMOV FR,FR	LDS.L @Rm+,SR1	OCBWB	
	FMOV.S @adr,FR	LDS.L @Rm+,SR2	PREF	

命令グループ	命令			
FE	FADD	FDIV	FRCHG	FSCA
	FSUB	FIPR	FSCHG	FSRRA
	FCMP（S/D）	FLOAT	FSQRT	FPCHG
	FCNVDS	FMAC	FTRC	
	FCNVSD	FMUL	FTRV	
CO	AND.B #imm,@(R0,GBR)	LDC.L @Rm+,SR	PREFI	TRAPA
	ICBI	LDTLB	RTE	TST.B #imm,@(R0,GBR)
	LDC Rm,DBR	MAC.L	SLEEP	XOR.B #imm,@(R0,GBR)
	LDC Rm,SGR	MAC.W	STC SR,Rn	
	LDC Rm,SR	MOVCO	STC.L SR,@-Rn	
	LDC.L @Rm+,DBR	MOVLI	SYNCO	
	LDC.L @Rm+,SGR	OR.B #imm,@(R0,GBR)	TAS.B	

【記号説明】 R : Rm/Rn

@adr : アドレス

SR1 : MACH/MACL/PR

SR2 : FPUL/FPSCR

CR1 : GBR/Rp_BANK/SPC/SSR/VBR

CR2 : CR1/DBR/SGR

FR : FRm/FRn/DRm/DRn/XDm/XDn

2 命令の同時実行は次の場合に限ります。

addr（先行）とaddr+2（後行）の2命令で1Kバイト（最小のページサイズ）をまたがないこと

表4.3（先行・後行掛け合わせ表）で同時実行可能である（○となっている）こと

addrにある命令がそれ以前の命令とデータコンフリクトがないこと

addr+2にある命令がそれ以前の命令とデータコンフリクトがないこと

2命令とも有効であること

表 4.3　先行・後行掛け合わせ表

		先行命令（addr）					
		EX	MT	BR	LS	FE	CO
後行命令（addr+2）	EX	×	○	○	○	○	
	MT	○	○	○	○	○	
	BR	○	○	×	○	○	
	LS	○	○	○	×	○	
	FE	○	○	○	○	×	
	CO						×

4.3 発行レートと実行ステート

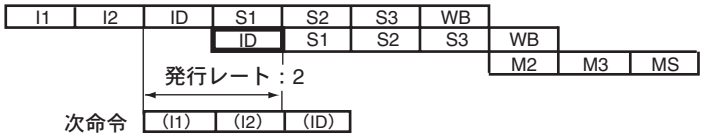
命令の発行レートと実行ステートを**表 4.4** に示します。**表 4.4** 中の命令グループは**表 4.2** における分類に対応します。また、本節に示す発行レートと実行ステートでは、パイプラインストールによるペナルティサイクルは考慮していません。

（1）発行レート

発行レートは、命令の発行と次の命令の発行の間隔を示します。

（例）AND.B命令

（例）MAC.W命令

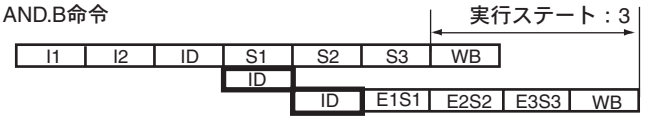


（2）実行ステート

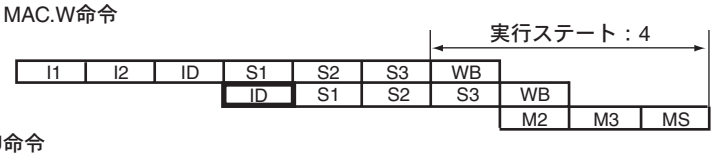
実行ステートは、命令がパイプラインを占有するサイクル数を次の基準で示します。

・CPU命令

（例）AND.B命令

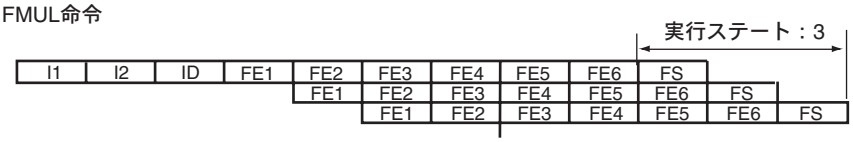


（例）MAC.W命令



・FPU命令

（例）FMUL命令



（例）FDIV命令

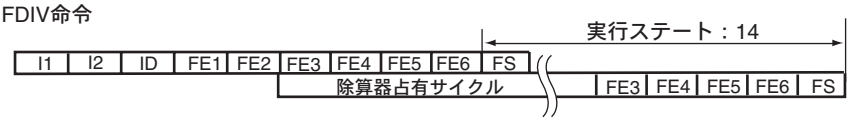


表 4.4　発行レートと実行ステート

機能分類	No.	命令		命令グループ	発行レート	実行ステート	実行パターン
データ転送命令	1	EXTS.B	Rm,Rn	EX	1	1	2-1
	2	EXTS.W	Rm,Rn	EX	1	1	2-1
	3	EXTU.B	Rm,Rn	EX	1	1	2-1
	4	EXTU.W	Rm,Rn	EX	1	1	2-1
	5	MOV	Rm,Rn	MT	1	1	2-4
	6	MOV	#imm,Rn	MT	1	1	2-3
	7	MOVA	@(disp,PC),R0	LS	1	1	2-2
	8	MOV.W	@(disp,PC),Rn	LS	1	1	3-1
	9	MOV.L	@(disp,PC),Rn	LS	1	1	3-1
	10	MOV.B	@Rm,Rn	LS	1	1	3-1
	11	MOV.W	@Rm,Rn	LS	1	1	3-1
	12	MOV.L	@Rm,Rn	LS	1	1	3-1
	13	MOV.B	@Rm+,Rn	LS	1	1	3-1
	14	MOV.W	@Rm+,Rn	LS	1	1	3-1
	15	MOV.L	@Rm+,Rn	LS	1	1	3-1
	16	MOV.B	@(disp,Rm),R0	LS	1	1	3-1
	17	MOV.W	@(disp,Rm),R0	LS	1	1	3-1
	18	MOV.L	@(disp,Rm),Rn	LS	1	1	3-1
	19	MOV.B	@(R0,Rm),Rn	LS	1	1	3-1
	20	MOV.W	@(R0,Rm),Rn	LS	1	1	3-1
	21	MOV.L	@(R0,Rm),Rn	LS	1	1	3-1
	22	MOV.B	@(disp,GBR),R0	LS	1	1	3-1
	23	MOV.W	@(disp, GBR),R0	LS	1	1	3-1
	24	MOV.L	@(disp, GBR),R0	LS	1	1	3-1
	25	MOV.B	Rm,@Rn	LS	1	1	3-1
	26	MOV.W	Rm,@Rn	LS	1	1	3-1
	27	MOV.L	Rm,@Rn	LS	1	1	3-1
	28	MOV.B	Rm,@-Rn	LS	1	1	3-1
	29	MOV.W	Rm,@-Rn	LS	1	1	3-1
	30	MOV.L	Rm,@-Rn	LS	1	1	3-1
	31	MOV.B	R0,@(disp,Rn)	LS	1	1	3-1
	32	MOV.W	R0,@(disp,Rn)	LS	1	1	3-1
	33	MOV.L	Rm,@(disp,Rn)	LS	1	1	3-1
	34	MOV.B	Rm,@(R0,Rn)	LS	1	1	3-1
	35	MOV.W	Rm,@(R0,Rn)	LS	1	1	3-1

機能 分類	No.	命令		命令 グループ	発行 レート	実行 ステート	実行 パターン
データ 転送命令	36	MOV.L	Rm,@(R0,Rn)	LS	1	1	3-1
	37	MOV.B	R0,@(disp,GBR)	LS	1	1	3-1
	38	MOV.W	R0,@(disp,GBR)	LS	1	1	3-1
	39	MOV.L	R0,@(disp,GBR)	LS	1	1	3-1
	40	MOVCA.L	R0,@Rn	LS	1	1	3-4
	41	MOVCO.L	R0,@Rn	CO	1	1	3-9
	42	MOVLI.L	@Rm,R0	CO	1	1	3-8
	43	MOVUA.L	@Rm,R0	LS	2	2	3-10
	44	MOVUA.L	@Rm+,R0	LS	2	2	3-10
	45	MOVT	Rn	EX	1	1	2-1
	46	OCBI	@Rn	LS	1	1	3-4
	47	OCBP	@Rn	LS	1	1	3-4
	48	OCBWB	@Rn	LS	1	1	3-4
	49	PREF	@Rn	LS	1	1	3-4
	50	SWAP.B	Rm,Rn	EX	1	1	2-1
	51	SWAP.W	Rm,Rn	EX	1	1	2-1
	52	XTRCT	Rm,Rn	EX	1	1	2-1
固定小数点 算術命令	53	ADD	Rm,Rn	EX	1	1	2-1
	54	ADD	#imm,Rn	EX	1	1	2-1
	55	ADDC	Rm,Rn	EX	1	1	2-1
	56	ADDV	Rm,Rn	EX	1	1	2-1
	57	CMP/EQ	#imm,R0	EX	1	1	2-1
	58	CMP/EQ	Rm,Rn	EX	1	1	2-1
	59	CMP/GE	Rm,Rn	EX	1	1	2-1
	60	CMP/GT	Rm,Rn	EX	1	1	2-1
	61	CMP/HI	Rm,Rn	EX	1	1	2-1
	62	CMP/HS	Rm,Rn	EX	1	1	2-1
	63	CMP/PL	Rn	EX	1	1	2-1
	64	CMP/PZ	Rn	EX	1	1	2-1
	65	CMP/STR	Rm,Rn	EX	1	1	2-1
	66	DIV0S	Rm,Rn	EX	1	1	2-1
	67	DIV0U		EX	1	1	2-1
	68	DIV1	Rm,Rn	EX	1	1	2-1
	69	DMULS.L	Rm,Rn	EX	1	2	5-6
	70	DMULU.L	Rm,Rn	EX	1	2	5-6
	71	DT	Rn	EX	1	1	2-1
	72	MAC.L	@Rm+,@Rn+	CO	2	5	5-9

機能 分類	No.	命令		命令 グループ	発行 レート	実行 ステート	実行 パターン
固定小数点 算術命令	73	MAC.W	@Rm+,@Rn+	CO	2	4	5-8
	74	MUL.L	Rm,Rn	EX	1	2	5-6
	75	MULS.W	Rm,Rn	EX	1	1	5-5
	76	MULU.W	Rm,Rn	EX	1	1	5-5
	77	NEG	Rm,Rn	EX	1	1	2-1
	78	NEGC	Rm,Rn	EX	1	1	2-1
	79	SUB	Rm,Rn	EX	1	1	2-1
	80	SUBC	Rm,Rn	EX	1	1	2-1
	81	SUBV	Rm,Rn	EX	1	1	2-1
論理命令	82	AND	Rm,Rn	EX	1	1	2-1
	83	AND	#imm,R0	EX	1	1	2-1
	84	AND.B	#imm,@(R0,GBR)	CO	3	3	3-2
	85	NOT	Rm,Rn	EX	1	1	2-1
	86	OR	Rm,Rn	EX	1	1	2-1
	87	OR	#imm,R0	EX	1	1	2-1
	88	OR.B	#imm,@(R0,GBR)	CO	3	3	3-2
	89	TAS.B	@Rn	CO	4	4	3-3
	90	TST	Rm,Rn	EX	1	1	2-1
	91	TST	#imm,R0	EX	1	1	2-1
	92	TST.B	#imm,@(R0,GBR)	CO	3	3	3-2
	93	XOR	Rm,Rn	EX	1	1	2-1
	94	XOR	#imm,R0	EX	1	1	2-1
	95	XOR.B	#imm,@(R0,GBR)	CO	3	3	3-2
シフト命令	96	ROTL	Rn	EX	1	1	2-1
	97	ROTR	Rn	EX	1	1	2-1
	98	ROTCL	Rn	EX	1	1	2-1
	99	ROTCR	Rn	EX	1	1	2-1
	100	SHAD	Rm,Rn	EX	1	1	2-1
	101	SHAL	Rn	EX	1	1	2-1
	102	SHAR	Rn	EX	1	1	2-1
	103	SHLD	Rm,Rn	EX	1	1	2-1
	104	SHLL	Rn	EX	1	1	2-1
	105	SHLL2	Rn	EX	1	1	2-1
	106	SHLL8	Rn	EX	1	1	2-1
	107	SHLL16	Rn	EX	1	1	2-1
	108	SHLR	Rn	EX	1	1	2-1
	109	SHLR2	Rn	EX	1	1	2-1

機能 分類	No.	命令		命令 グループ	発行 レート	実行 ステート	実行 パターン
シフト命令	110	SHLR8	Rn	EX	1	1	2-1
	111	SHLR16	Rn	EX	1	1	2-1
分岐命令	112	BF	disp	BR	1+0～2	1	1-1
	113	BF/S	disp	BR	1+0～2	1	1-1
	114	BT	disp	BR	1+0～2	1	1-1
	115	BT/S	disp	BR	1+0～2	1	1-1
	116	BRA	disp	BR	1+0～2	1	1-1
	117	BRAF	Rm	BR	1+3	1	1-2
	118	BSR	disp	BR	1+0～2	1	1-1
	119	BSRF	Rm	BR	1+3	1	1-2
	120	JMP	@Rn	BR	1+3	1	1-2
	121	JSR	@Rn	BR	1+3	1	1-2
	122	RTS		BR	1+0～3	1	1-3
システム制御命令	123	NOP		MT	1	1	2-3
	124	CLRMAC		EX	1	1	5-7
	125	CLRS		EX	1	1	2-1
	126	CLRT		EX	1	1	2-1
	127	ICBI	@Rn	CO	8+5+3	13	3-6
	128	SETS		EX	1	1	2-1
	129	SETT		EX	1	1	2-1
	130	PREFI		CO	5+5+3	10	3-7
	131	SYNCO	@Rn	CO	不定	不定	3-4
	132	TRAPA	#imm	CO	8+5+1	13	1-5
	133	RTE		CO	4+1	4	1-4
	134	SLEEP		CO	不定	不定	1-6
	135	LDTLB		CO	1	1	3-5
	136	LDC	Rm,DBR	CO	4	4	4-2
	137	LDC	Rm,SGR	CO	4	4	4-2
	138	LDC	Rm,GBR	LS	1	1	4-3
	139	LDC	Rm,Rp_BANK	LS	1	1	4-1
	140	LDC	Rm,SR	CO	4+3	4	4-4
	141	LDC	Rm,SSR	LS	1	1	4-1
	142	LDC	Rm,SPC	LS	1	1	4-1
	143	LDC	Rm,VBR	LS	1	1	4-1
	144	LDC.L	@Rm+,DBR	CO	4	4	4-6
	145	LDC.L	@Rm+,SGR	CO	4	4	4-6

機能 分類	No.	命令		命令 グループ	発行 レート	実行 ステート	実行 パターン
システム制御 命令	146	LDC.L	@Rm+,GBR	LS	1	1	4-7
	147	LDC.L	@Rm+,Rp_BANK	LS	1	1	4-5
	148	LDC.L	@Rm+,SR	CO	6+3	4	4-8
	149	LDC.L	@Rm+,SSR	LS	1	1	4-5
	150	LDC.L	@Rm+,SPC	LS	1	1	4-5
	151	LDC.L	@Rm+,VBR	LS	1	1	4-5
	152	LDS	Rm,MACH	LS	1	1	5-1
	153	LDS	Rm,MACL	LS	1	1	5-1
	154	LDS	Rm,PR	LS	1	1	4-13
	155	LDS.L	@Rm+,MACH	LS	1	1	5-2
	156	LDS.L	@Rm+,MACL	LS	1	1	5-2
	157	LDS.L	@Rm+,PR	LS	1	1	4-14
	158	STC	DBR,Rn	LS	1	1	4-9
	159	STC	SGR,Rn	LS	1	1	4-9
	160	STC	GBR,Rn	LS	1	1	4-9
	161	STC	Rp_BANK,Rn	LS	1	1	4-9
	162	STC	SR,Rn	CO	1	1	4-10
	163	STC	SSR,Rn	LS	1	1	4-9
	164	STC	SPC,Rn	LS	1	1	4-9
	165	STC	VBR,Rn	LS	1	1	4-9
	166	STC.L	DBR,@-Rn	LS	1	1	4-11
	167	STC.L	SGR,@-Rn	LS	1	1	4-11
	168	STC.L	GBR,@-Rn	LS	1	1	4-11
	169	STC.L	Rp_BANK,@-Rn	LS	1	1	4-11
	170	STC.L	SR,@-Rn	CO	1	1	4-12
	171	STC.L	SSR,@-Rn	LS	1	1	4-11
	172	STC.L	SPC,@-Rn	LS	1	1	4-11
	173	STC.L	VBR,@-Rn	LS	1	1	4-11
	174	STS	MACH,Rn	LS	1	1	5-3
	175	STS	MACL,Rn	LS	1	1	5-3
	176	STS	PR,Rn	LS	1	1	4-15
	177	STS.L	MACH,@-Rn	LS	1	1	5-4
	178	STS.L	MACL,@-Rn	LS	1	1	5-4
	179	STS.L	PR,@-Rn	LS	1	1	4-16
単精度 浮動小数点 命令	180	FLDI0	FRn	LS	1	1	6-13
	181	FLDI1	FRn	LS	1	1	6-13
	182	FMOV	FRm,FRn	LS	1	1	6-9

機能分類	No.	命令		命令グループ	発行レート	実行ステート	実行パターン
単精度浮動小数点命令	183	FMOV.S	@Rm,FRn	LS	1	1	6-9
	184	FMOV.S	@Rm+,FRn	LS	1	1	6-9
	185	FMOV.S	@(R0,Rm),FRn	LS	1	1	6-9
	186	FMOV.S	FRm,@Rn	LS	1	1	6-9
	187	FMOV.S	FRm,@-Rn	LS	1	1	6-9
	188	FMOV.S	FRm,@(R0,Rn)	LS	1	1	6-9
	189	FLDS	FRm,FPUL	LS	1	1	6-10
	190	FSTS	FPUL,FRn	LS	1	1	6-11
	191	FABS	FRn	LS	1	1	6-12
	192	FADD	FRm,FRn	FE	1	1	6-14
	193	FCMP/EQ	FRm,FRn	FE	1	1	6-14
	194	FCMP/GT	FRm,FRn	FE	1	1	6-14
	195	FDIV	FRm,FRn	FE	1	14	6-15
	196	FLOAT	FPUL,FRn	FE	1	1	6-14
	197	FMAC	FR0,FRm,FRn	FE	1	1	6-14
	198	FMUL	FRm,FRn	FE	1	1	6-14
	199	FNEG	FRn	LS	1	1	6-12
	200	FSQRT	FRn	FE	1	30	6-15
	201	FSUB	FRm,FRn	FE	1	1	6-14
	202	FTRC	FRm,FPUL	FE	1	1	6-14
	203	FMOV	DRm,DRn	LS	1	1	6-9
	204	FMOV	@Rm,DRn	LS	1	1	6-9
	205	FMOV	@Rm+,DRn	LS	1	1	6-9
	206	FMOV	@(R0,Rm),DRn	LS	1	1	6-9
	207	FMOV	DRm,@Rn	LS	1	1	6-9
	208	FMOV	DRm,@-Rn	LS	1	1	6-9
	209	FMOV	DRm,@(R0,Rn)	LS	1	1	6-9
倍精度浮動小数点命令	210	FABS	DRn	LS	1	1	6-12
	211	FADD	DRm,DRn	FE	1	1	6-16
	212	FCMP/EQ	DRm,DRn	FE	1	1	6-16
	213	FCMP/GT	DRm,DRn	FE	1	1	6-16
	214	FCNVDS	DRm,FPUL	FE	1	1	6-16
	215	FCNVSD	FPUL,DRn	FE	1	1	6-16
	216	FDIV	DRm,DRn	FE	1	14	6-18
	217	FLOAT	FPUL,DRn	FE	1	1	6-16

機能 分類	No.	命令		命令 グループ	発行 レート	実行 ステート	実行 パターン
倍精度	218	FMUL	DRm,DRn	FE	1	3	6-17
浮動小数点	219	FNEG	DRn	LS	1	1	6-12
命令	220	FSQRT	DRn	FE	1	30	6-18
	221	FSUB	DRm,DRn	FE	1	1	6-16
	222	FTRC	DRm,FPUL	FE	1	1	6-16
FPU	223	LDS	Rm,FPUL	LS	1	1	6-1
システム制御	224	LDS	Rm,FPSCR	LS	1	1	6-5
命令	225	LDS.L	@Rm+,FPUL	LS	1	1	6-3
	226	LDS.L	@Rm+,FPSCR	LS	1	1	6-7
	227	STS	FPUL,Rn	LS	1	1	6-2
	228	STS	FPSCR,Rn	LS	1	1	6-6
	229	STS.L	FPUL,@-Rn	LS	1	1	6-4
	230	STS.L	FPSCR,@-Rn	LS	1	1	6-8
グラフィクス	231	FMOV	DRm,XDn	LS	1	1	6-9
強化命令	232	FMOV	XDm,DRn	LS	1	1	6-9
	233	FMOV	XDm,XDn	LS	1	1	6-9
	234	FMOV	@Rm,XDn	LS	1	1	6-9
	235	FMOV	@Rm+,XDn	LS	1	1	6-9
	236	FMOV	@(R0,Rm),XDn	LS	1	1	6-9
	237	FMOV	XDm,@Rn	LS	1	1	6-9
	238	FMOV	XDm,@-Rn	LS	1	1	6-9
	239	FMOV	XDm,@(R0,Rn)	LS	1	1	6-9
	240	FIPR	FVm,FVn	FE	1	1	6-19
	241	FRCHG		FE	1	1	6-14
	242	FSCHG		FE	1	1	6-14
	243	FPCHG		FE	1	1	6-14
	244	FSRRA	FRn	FE	1	1	6-21
	245	FSCA	FPUL,DRn	FE	1	3	6-22
	246	FTRV	XMTRX,FVn	FE	1	4	6-20

5. 例外処理

5.1 概要

例外処理とは、リセット、一般例外、割り込みが検出されたときに、通常とは異なるプログラムで必要な処理を行うことをいいます。たとえば、実行中の命令の異常終了が発生した場合、適切な処置をすることで、元のプログラムに復帰したり、異常を報告して終了するといった制御が必要になります。このような機能をサポートするために、異常終了に対して、例外処理要求を発生させ、ユーザが作成した例外処理ルーチンに制御の流れが渡ることなどを総称して例外処理と呼びます。

SH-4A の例外処理は、リセット、一般例外、割り込みの 3 つに分類されます。

5.2 レジスタの説明

例外処理に関するレジスタ構成を**表 5.1** に示します。

表 5.1 レジスタ構成

名称	略称	R/W	P4 領域 アドレス*	エリア 7 アドレス*	アクセス サイズ
TRAPA 例外レジスタ	TRA	R/W	H'FF00 0020	H'1F00 0020	32
例外事象レジスタ	EXPEVT	R/W	H'FF00 0024	H'1F00 0024	32
割り込み事象レジスタ	INTEVT	R/W	H'FF00 0028	H'1F00 0028	32

【注】 * P4 領域アドレスは、仮想アドレス空間の P4 領域を用いた場合のものです。エリア 7 アドレスは、TLB を用いて物理アドレス空間のエリア 7 からアクセスするものです。

表 5.2 各処理モードにおけるレジスタの状態

名称	略称	パワーオン リセット	マニュアル リセット	スリープ	スタンバイ
TRAPA 例外レジスタ	TRA	不定	不定	保持	保持
例外事象レジスタ	EXPEVT	H'0000 0000	H'0000 0020	保持	保持
割り込み事象レジスタ	INTEVT	不定	不定	保持	保持

5.2.1 TRAPA 例外レジスタ（TRA）

TRAPA 例外レジスタ（TRA）は、TRAPA 命令の 8 ビットイミディエイトデータ（imm）が設定されるレジスタです。TRA は TRAPA 命令実行時にハードウェアにより自動的に設定されます。TRA はソフトウェアからも変更が可能です。

ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	—	—	TRACODE								—	—
初期値：	0	0	0	0	0	0	—	—	—	—	—	—	—	—	0	0
R/W：	R	R	R	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R	R

ビット	ビット名	初期値	R/W	説　　明
31～10	−	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
9～2	TRACODE	不定	R/W	TRAPA コード TRAPA 命令の 8 ビットイミディエイトデータが設定されます。
1、0	−	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。

5.2.2 例外事象レジスタ（EXPEVT）

例外事象レジスタ（EXPEVT）には、12 ビットのリセットと一般例外事象による例外コードが設定されます。例外コードは例外受け付け時にハードウェアにより自動的に設定されます。EXPEVT はソフトウェアからも変更が可能です。

ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	EXPCODE											
初期値：	0	0	0	0	0	0	0	0	0	0	0/1	0	0	0	0	0
R/W：	R	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ビット	ビット名	初期値	R/W	説　　明
31～12	−	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
11～0	EXPCODE	H'000 または H'020	R/W	例外コード リセット、一般例外の例外コードが設定されます。詳細は表 5.3 を参照してください。

5.2.3　割り込み事象レジスタ（INTEVT）

割り込み事象レジスタ（INTEVT）には、14 ビットの割り込み要求による例外コードが設定されます。例外コードは例外受け付け時にハードウェアにより自動的に設定されます。INTEVT はソフトウェアからも変更が可能です。

ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	INTCODE													
初期値：	0	0	—	—	—	—	—	—	—	—	—	—	—	—	—	—
R/W：	R	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ビット	ビット名	初期値	R/W	説　　明
31～14	−	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
13～0	INTCODE	不定	R/W	例外コード 割り込みの例外コードが設定されます。詳細は表 5.3 を参照してください。

5.3 例外処理の機能

5.3.1 例外処理の流れ

例外処理では、プログラムカウンタ（PC）、ステータスレジスタ（SR）、R15 の内容がそれぞれ退避プログラムカウンタ（SPC）、退避ステータスレジスタ（SSR）、退避ジェネラルレジスタ（SGR）に退避され、ベクタアドレスに従って対応する例外処理ルーチンの実行を開始します。例外処理ルーチンとは、ユーザによって、個々の例外の内容に応じて作成されたプログラムです。例外処理ルーチンを終了させ、元のプログラムに戻るためには、例外処理からの復帰命令（RTE）を実行します。本命令によって、PC と SR の内容が復帰し、例外などが発生した時点での通常処理ルーチンに戻ることができます。なお、SGR の内容は RTE 命令では R15 に書き戻されません。

基本的な例外処理の流れは次のようになります。SR のビットの意味の詳細は、**「第 2 章　プログラミングモデル」**を参照してください。

1. PC、SRおよびR15の内容がそれぞれSPC、SSRおよびSGRに退避されます。
2. SRのブロックビット（BL）が1に設定されます。
3. SRのモードビット（MD）が1に設定されます。
4. SRのレジスタバンクビット（RB）が1に設定されます。
5. リセット時、SRのFPUディスエーブルビット（FD）が0に設定されます。
6. 例外コードは、例外要因の例外事象レジスタ（EXPEVT）、または割り込み事象レジスタ（INTEVT）のビット13～0に書き込まれます。
7. 決められた例外処理のベクタアドレスに分岐して、例外処理ルーチンを開始します。

5.3.2 例外処理ベクタアドレス

リセットベクタアドレスは H'A000 0000 に固定されています。例外、割り込みのベクタアドレスはベクタベースアドレスに各事象のオフセット値を加えたアドレスです。ベクタベースアドレスはベクタベースレジスタ（VBR）にソフトウェアで設定します。たとえば、TLB ミス例外のオフセットは H'0000 0400 ですから、VBR に H'9C08 0000 を設定しておくと、例外処理ベクタアドレスは H'9C08 0400 になります。例外処理ベクタアドレスでさらに例外が発生すると、二重例外となり、回復が困難になりますので、ベクタアドレスはアドレス変換の対象とならない P1、P2 領域のアドレスを指定してください。

5.4 例外の種類と優先順位

表 5.3 に、例外の種類、優先順位、ベクタアドレス、および例外／割り込みコードを示します。

表 5.3 例外一覧

例外区分	実行形態	例外	優先レベル	優先順位	例外遷移先		例外コード
					ベクタベース	オフセット	
リセット	中断型	パワーオンリセット	1	1	H'A000 0000	－	H'000
		マニュアルリセット	1	2	H'A000 0000	－	H'020
		H-UDI リセット	1	1	H'A000 0000	－	H'000
		命令 TLB 多重ヒット例外	1	3	H'A000 0000	－	H'140
		データ TLB 多重ヒット例外	1	4	H'A000 0000	－	H'140
一般例外	再実行型	命令実行前ユーザブレーク*	2	0	（VBR/DBR）	H'100/－	H'1E0
		命令アドレスエラー	2	1	（VBR）	H'100	H'0E0
		命令 TLB ミス例外	2	2	（VBR）	H'400	H'040
		命令 TLB 保護違反例外	2	3	（VBR）	H'100	H'0A0
		一般不当命令例外	2	4	（VBR）	H'100	H'180
		スロット不当命令例外	2	4	（VBR）	H'100	H'1A0
		一般 FPU 抑止例外	2	4	（VBR）	H'100	H'800
		スロット FPU 抑止例外	2	4	（VBR）	H'100	H'820
		データアドレスエラー（読み出し）	2	5	（VBR）	H'100	H'0E0
		データアドレスエラー（書き込み）	2	5	（VBR）	H'100	H'100
		データ TLB ミス例外（読み出し）	2	6	（VBR）	H'400	H'040
		データ TLB ミス例外（書き込み）	2	6	（VBR）	H'400	H'060
		データ TLB 保護違反例外（読み出し）	2	7	（VBR）	H'100	H'0A0
		データ TLB 保護違反例外（書き込み）	2	7	（VBR）	H'100	H'0C0
		FPU 例外	2	8	（VBR）	H'100	H'120
		初期ページ書き込み例外	2	9	（VBR）	H'100	H'080
	完了型	無条件トラップ（TRAPA）	2	4	（VBR）	H'100	H'160
		命令実行後ユーザブレーク*	2	10	（VBR/DBR）	H'100/－	H'1E0
割り込み	完了型	ノンマスカブル割り込み	3	－	（VBR）	H'600	H'1C0
		一般割り込み要求	4	－	（VBR）	H'600	－

優先度　：まず優先レベルで順位付けし、同一レベル内を優先順位で順位付けします（より小さい数値が、優先度が高くなります）。

例外遷移先：リセットでは H'A000 0000、その他では（VBR＋オフセット）へ制御が移ります。

例外コード：リセット、一般例外では EXPEVT、割込みでは INTEVT に格納されます。

【注】 * CBCR.UBDE=1 のとき PC=DBR。その他は PC=VBR+H'100

5.5 例外フロー

5.5.1 例外フロー

図 5.1 に、命令実行と例外処理の基本動作を概念的に示します。ここでは説明の都合上、命令を 1 命令ずつ逐次的に実行することを基本として説明しています。**図 5.1** には、例外種別（リセット、一般例外、割り込み）間の優先順位が表されています。なお**図 5.1** では、例外成立時のレジスタ設定を SSR、SPC、SGR、EXPEVT/INTEVT、SR、および PC に限っていますが、例外によってはこの他にもハードウェアによって自動的に設定されるレジスタがあります。詳細は、**「5.6 各例外の説明」**を参照してください。また、遅延分岐命令と遅延スロット命令を実行中の例外処理や、2 回データアクセスが発生する命令については**「5.6.4 複数回の例外が発生する場合の優先順位」**を参照してください。

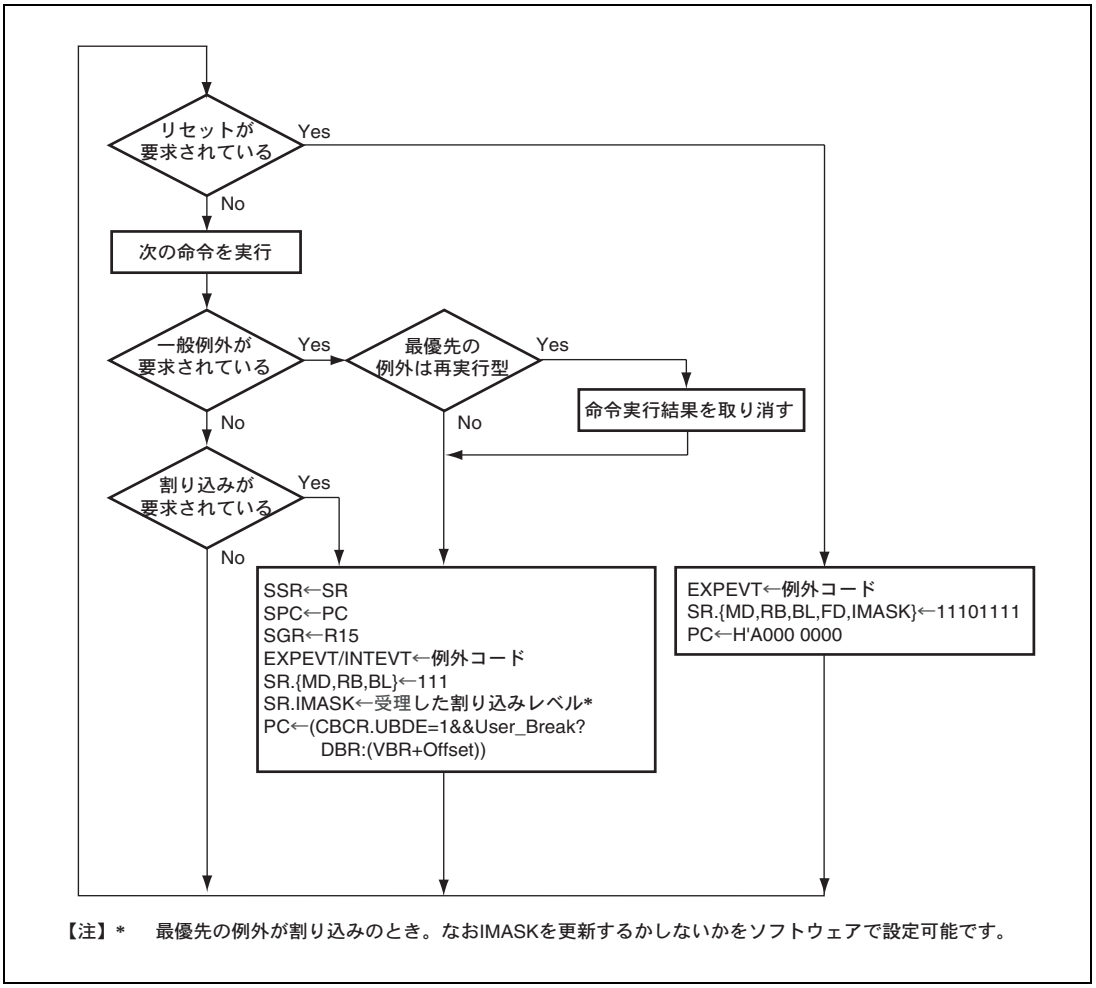


図 5.1 命令実行と例外処理

5.5.2 例外要因の受け付け

2 つ以上の例外が同時に発生したときに受け付ける例外を決定するため、すべての例外には優先順位が決められています。一般例外の中の一般不当命令例外、スロット不当命令例外、一般 FPU 抑止例外、スロット FPU 抑止例外、無条件トラップ例外の 5 つは、それぞれの命令解析の過程で検出され、命令パイプラインの中では同時に発生しない例外です。このため優先順位は同じ値になっています。一般例外は命令実行に従った順序で検出されます。しかし、例外処理は命令の流れの順序（プログラム順）に従って処理されます。つまり、先の命令の例外が、後続の命令の例外よりも優先されて受け付けられます。一般例外の受け付け順序の例を図 5.2 に示します。

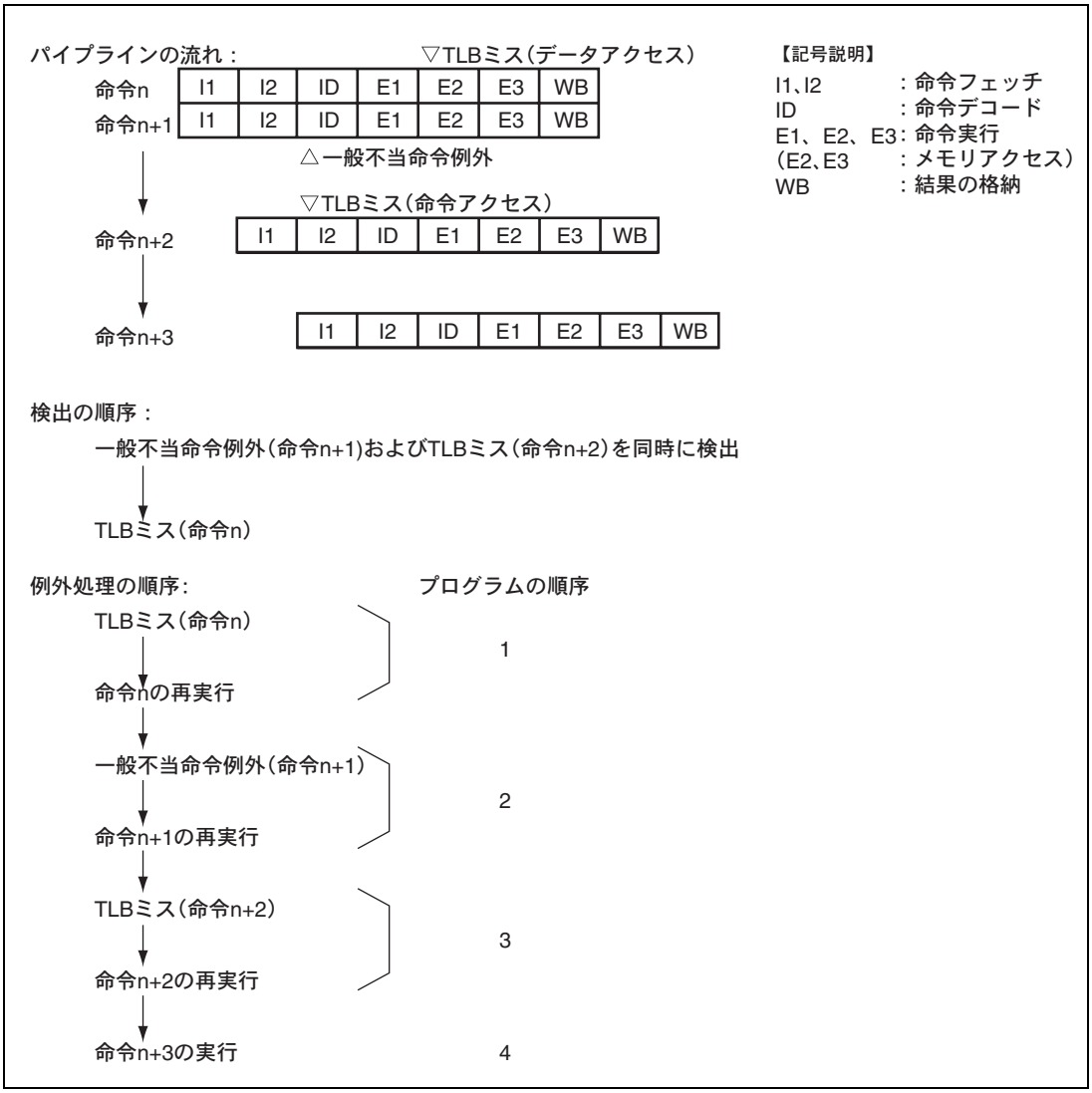


図 5.2 一般例外の受け付け順序の例

5.5.3 例外要求と BL ビット

SR の BL ビットが 0 のとき、例外、割り込みを受け付けます。

SR の BL ビットが 1 のときに、ユーザブレークを除く例外が発生した場合には、CPU の内部レジスタ、他のモジュールのレジスタは、マニュアルリセット後の状態になり、リセットと同アドレス（H'A000 0000）に分岐します。ユーザブレークが発生した場合の動作については当該製品ハードウェアマニュアルの**「ユーザブレークコントローラ」**を参照してください。また、通常の割り込みが発生した場合には、割り込み要求は保留され、ソフトウェアで BL ビットが 0 にクリアされてから受け付けられます。ノンマスカブル割り込み（NMI）が発生した場合は、保留するか、受け付けるかをソフトウェアによって設定可能です。

このように、通常は例外状態を多重に受け付け可能にするためには、SPC と SSR を退避させ、その後 SR の BL ビットを 0 クリアします。

5.5.4 例外処理からの復帰

例外処理からの復帰は、RTE 命令を使用します。RTE 命令により、SPC が PC に、SSR が SR に回復され、SPC のアドレスに分岐して、例外処理ルーチンから復帰します。もし、メモリに SPC、SSR を退避していた場合には、SR の BL ビットを 1 にセットしてから、SPC と SSR を回復し、RTE 命令を発行してください。

5.6 各例外の説明

個別の例外処理動作について、発生要因、発生時の遷移先アドレス、遷移時のプロセッサの動作を説明します。

5.6.1 リセット

(1) パワーオンリセット

- **条件：**

  パワーオンリセット要求

- **動作：**

  EXPEVTにH'000を設定し、CPUおよび内蔵周辺モジュールの初期化を行った後リセットベクタ（H'A0000000）に分岐します。詳細は、各章のレジスタの説明を参照してください。電源投入時には必ずパワーオンリセットを行ってください。

(2) マニュアルリセット

- **条件：**

  マニュアルリセット要求

- **動作：**

  EXPEVTにH'020を設定し、CPUおよび内蔵周辺モジュールの初期化を行った後リセットベクタ（H'A0000000）に分岐します。パワーオンリセットとマニュアルリセットでは初期化されるレジスタが異なります。詳細は、各章のレジスタの説明を参照してください。

(3) H-UDI リセット

- **要因：**SDIR. TI[7:4]がB'0110（ネゲート）、またはB'0111（アサート）
- **遷移先アドレス：**H'A000 0000
- **遷移時動作：**

  例外コードH'000をEXPEVTにセットします。VBR、 SRの初期化を行い、PC＝H'A000 0000に分岐します。

  CPUおよび内蔵周辺モジュールの初期化を行います。詳細は、ハードウェアマニュアルの各章のレジスタの説明を参照してください。

(4) 命令 TLB 多重ヒット例外

- **要因：**ITLBのアドレスが多重に一致
- **遷移先アドレス：**H'A000 0000
- **遷移時動作：**

  本例外を発生させた仮想アドレス（32ビット）をTEAに、対応する仮想ページ番号（22ビット）をPTEH[31：10]にセットします。PTEHのASIDは本例外発生時のASIDを示します。

  例外コードH'140をEXPEVTにセットします。VBR、SRの初期化を行い、PC＝H'A000 0000に分岐します。

  CPUおよび内蔵周辺モジュールの初期化をマニュアルリセットの場合と同様に行います。詳細は、ハードウェアマニュアルの各章のレジスタの説明を参照してください。

(5) データ TLB 多重ヒット例外

- **要因：UTLBのアドレスが多重に一致**
- **遷移先アドレス**：H'A000 0000
- **遷移時動作**：

  本例外を発生させた仮想アドレス（32ビット）をTEAに、対応する仮想ページ番号（22ビット）をPTEH[31：10]にセットします。PTEHのASIDは本例外発生時のASIDを示します。

  例外コードH'140をEXPEVTにセットします。VBR、 SRの初期化を行い、PC＝H'A000 0000に分岐します。

  CPUおよび内蔵周辺モジュールの初期化をマニュアルリセットの場合と同様に行います。詳細は、ハードウェアマニュアルの各章のレジスタの説明を参照してください。

5.6.2 一般例外

(1) データ TLB ミス例外

- **要因：UTLBのアドレス比較の結果、アドレスが不一致**
- **遷移先アドレス**：VBR + H'0000 0400
- **遷移時動作**：

  本例外を発生させた仮想アドレス（32ビット）をTEAに、対応する仮想ページ番号（22ビット）をPTEH[31:10]にセットします。PTEHのASIDは本例外発生時のASIDを示します。

  本例外を発生させた命令のPC、SRをそれぞれSPC、SSRに退避し、そのときのR15をSGRに退避します。

  読み出しの場合は例外コードH'040を、書き込みの場合は例外コードH'060をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC＝VBR＋H'0400に分岐します。

  TLBミス処理高速化のために、他の例外とオフセットを分けています。

```
Data_TLB_miss_exception()
{
            TEA = EXCEPTION_ADDRESS;
            PTEH.VPN = PAGE_NUMBER;
            SPC = PC;
            SSR = SR;
            SGR = R15;
            EXPEVT = read_access ? H'00000040 : H'00000060;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000400;
}
```

(2) 命令 TLB ミス例外

- **要因：ITLBのアドレス比較の結果、アドレスが不一致**
- **遷移先アドレス：VBR + H'0000 0400**
- **遷移時動作：**

  本例外を発生させた仮想アドレス（32ビット）をTEAに、対応する仮想ページ番号（22ビット）をPTEH[31:10]にセットします。PTEHのASIDは本例外発生時のASIDを示します。

  本例外を発生させた命令のPC、SRをそれぞれSPC、SSRに退避し、そのときのR15をSGRに退避します。

  例外コードH'040をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC＝VBR＋H'0400に分岐します。

  TLBミス処理高速化のために、他の例外とオフセットを分けています。

```
ITLB_miss_exception()
{
            TEA = EXCEPTION_ADDRESS;
            PTEH.VPN = PAGE_NUMBER;
            SPC = PC;
            SSR = SR;
            SGR = R15;
            EXPEVT = H'00000040;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000400;
}
```

(3) 初期ページ書き込み例外

- **要因：ストアアクセスでTLBにヒットしたが、ダーティビットD = 0**
- **遷移先アドレス：VBR + H'0000 0100**
- **遷移時動作：**

  本例外を発生させた仮想アドレス（32ビット）をTEAに、対応する仮想ページ番号（22ビット）をPTEH[31:10]にセットします。PTEHのASIDは本例外発生時のASIDを示します。

  本例外を発生させた命令のPC、SRをそれぞれSPC、SSRに退避し、そのときのR15をSGRに退避します。

  例外コードH'080をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC＝VBR＋H'0100に分岐します。

```
Initial_write_exception()
{
            TEA = EXCEPTION_ADDRESS;
            PTEH.VPN = PAGE_NUMBER;
            SPC = PC;
            SSR = SR;
            SGR = R15;
            EXPEVT = H'00000080;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000100;
}
```

(4) データ TLB 保護違反例外

- **要因：アクセスが以下に示すUTLBのプロテクション情報（PRビット）に反する。**

PR	特権モード	ユーザモード
00	読み出しのみ可	アクセス不可
01	読み出し／書き込み可	アクセス不可
10	読み出しのみ可	読み出しのみ可
11	読み出し／書き込み可	読み出し／書き込み可

- **遷移先アドレス**：VBR + H'0000 0100
- **遷移時動作**：

  本例外を発生させた仮想アドレス（32ビット）をTEAに、対応する仮想ページ番号（22ビット）をPTEH[31:10]にセットします。PTEHのASIDは本例外発生時のASIDを示します。

  本例外を発生させた命令のPC、SRをそれぞれSPC、SSRに退避し、そのときのR15をSGRに退避します。

  読み出しの場合には例外コードH'0A0を、書き込みの場合には例外コードH'0C0をEXPEVTにセットします。

  SRのBLビット、MDビット、RBビットを1にセットし、PC＝VBR＋H'0100に分岐します。

```
Data_TLB_protection_violation_exception()
{
            TEA = EXCEPTION_ADDRESS;
            PTEH.VPN = PAGE_NUMBER;
            SPC = PC;
            SSR = SR;
            SGR = R15;
            EXPEVT = read_access ? H'000000A0 : H'000000C0;
```

```
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000100;
}
```

(5) 命令 TLB 保護違反例外

- 要因：アクセスが以下に示すITLBのプロテクション情報（PRビット）に反する。

PR	特権モード	ユーザモード
0	アクセス可	アクセス不可
1	アクセス可	アクセス可

- 遷移先アドレス：VBR + H'0000 0100
- 遷移時動作：

  本例外を発生させた仮想アドレス（32ビット）をTEAに、対応する仮想ページ番号（22ビット）をPTEH[31:10]にセットします。PTEHのASIDは本例外発生時のASIDを示します。

  本例外を発生させた命令のPC、SRをそれぞれSPC、SSRに退避し、そのときのR15をSGRに退避します。

  例外コードH'0A0をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC＝VBR＋H'0100に分岐します。

```
ITLB_protection_violation_exception()
{
            TEA = EXCEPTION_ADDRESS;
            PTEH.VPN = PAGE_NUMBER;
            SPC = PC;
            SSR = SR;
            SGR = R15;
            EXPEVT = H'000000A0;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000100;
}
```

(6) データアドレスエラー

- **要因：**
  - ワードデータをワード境界以外（2n＋1）からアクセス
  - ロングワードデータをロングワードデータ境界以外（4n＋1、4n＋2、4n＋3）からアクセス
  - クワッドワードをクワッドワードデータ境界以外（8n＋1、8n＋2、8n＋3、8n＋4、8n＋5、8n＋6、8n＋7）からアクセス
  - ユーザモードでの領域H'8000 0000～H'FFFF FFFFへのアクセス

    ただし、H'E000 0000～H'E3FF FFFFおよびH'E500 0000～H'E5FF FFFFは、それぞれユーザモードからアクセスする設定が可能です。詳しくは**「第7章　メモリマネジメントユニット（MMU）」**および**「第9章　L メモリ」**を参照してください。
- **遷移先アドレス**：VBR + H'0000 0100
- **遷移時動作**：

  本例外を発生させた仮想アドレス（32ビット）をTEAに、対応する仮想ページ番号(22ビット)をPTEH[31:10]にセットします。PTEHのASIDは本例外発生時のASIDを示します。

  本例外を発生させた命令のPC、SRをそれぞれSPC、SSRに退避し、そのときのR15をSGRに退避します。

  読み出しの場合は例外コードH'0E0を、書き込みの場合は例外コードH'100をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC＝VBR＋H'0100に分岐します。詳細は**「第7章　メモリマネジメントユニット（MMU）」**を参照してください。

```
Data_address_error()
{
            TEA = EXCEPTION_ADDRESS;
            PTEH.VPN = PAGE_NUMBER;
            SPC = PC;
            SSR = SR;
            SGR = R15;
            EXPEVT = read_access? H'000000E0: H'00000100;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000100;
}
```

(7) 命令アドレスエラー

- **要因：**
  - ワード境界以外（2n＋1）から命令フェッチ
  - ユーザモードでの領域H'8000 0000～H'FFFF FFFFから命令フェッチ

    ただし、H'E500 0000～H'E5FF FFFFはユーザモードからアクセスする設定が可能です。詳しくは**「第9章　L メモリ」**を参照してください。
- **遷移先アドレス：**VBR + H'0000 0100
- **遷移時動作：**

  本例外を発生させた仮想アドレス（32ビット）をTEAに、対応する仮想ページ番号（22ビット）をPTEH[31:10]にセットします。PTEHのASIDは本例外発生時のASIDを示します。

  本例外を発生させた命令のPC、SRをそれぞれSPC、SSRに退避し、そのときのR15をSGRに退避します。

  例外コードH'0E0をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC＝VBR＋H'0100に分岐します。詳細は**「第7章　メモリマネジメントユニット（MMU）」**を参照してください。

```
Instruction_address_error()
{
            TEA = EXCEPTION_ADDRESS;
            PTEH.VPN = PAGE_NUMBER;
            SPC = PC;
            SSR = SR;
            SGR = R15;
            EXPEVT = H'000000E0;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000100;
}
```

(8) 無条件トラップ

- **要因：TRAPA命令の実行**
- **遷移先アドレス：**VBR + H'0000 0100
- **遷移時動作：**

  処理完了型の例外のため、TRAPA命令の次の命令のPCをSPCに退避します。TRAPA命令実行時のSR、R15をSSR、SGRに退避します。TRAPA命令中の8ビットのイミディエイトを4倍して、TRA[9:0]にセットします。例外コードH'160をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC＝VBR＋H'0100に分岐します。

```
TRAPA_exception()
{
            SPC = PC + 2;
            SSR = SR;
            SGR = R15;
            TRA = imm << 2;
            EXPEVT = H'00000160;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000100;
}
```

(9) 一般不当命令例外

- **要因：**

  - 遅延スロット以外にある未定義命令をデコード

    遅延分岐命令：JMP、JSR、BRA、BRAF、BSR、BSRF、RTS、RTE、BT/S、BF/S

    未定義命令：H'FFFD

  - 遅延スロット以外にある特権命令をユーザモードでデコード

    特権命令：LDC、STC、RTE、LDTLB、SLEEP、

    ただし、LDC、STCでGBRをアクセスする命令を除く

- **遷移先アドレス**：VBR + H'0000 0100

- **遷移時動作**：

  本例外を発生させた命令のPC、SRをそれぞれSPC、SSRに退避し、そのときのR15をSGRに退避します。

  例外コードH'180をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC＝VBR＋H'0100に分岐します。なお、H'FFFD以外の未定義コードをデコードした場合には動作を保証しません。

```
General_illegal_instruction_exception()
{
            SPC = PC;
            SSR = SR;
            SGR = R15;
            EXPEVT = H'00000180;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000100;
}
```

（10）スロット不当命令例外

- **要因：**
  - 遅延スロットにある未定義命令をデコード

    遅延分岐命令：JMP、JSR、BRA、BRAF、BSR、BSRF、RTS、RTE、BT/S、BF/S

    未定義命令：H'FFFD
  - 遅延スロット内のPCを書き換える命令をデコード

    PCを書き換える命令：JMP、JSR、BRA、BRAF、BSR、BSRF、RTS、RTE、BT、BF、BT/S、BF/S、TRAPA、LDC Rm,SR、LDC.L @Rm＋,SR、ICBI、PREFI
  - 遅延スロット内の特権命令をユーザモードでデコード

    特権命令：LDC、STC、RTE、LDTLB、SLEEP

    ただし、LDC、STCでGBRをアクセスする命令を除く
  - 遅延スロット内のPC相対MOV命令、MOVA命令をデコード
- **遷移先アドレス**：VBR + H'0000 0100
- **遷移時動作**：

  直前の遅延分岐命令のPCをSPCに退避します。本例外発生時のSR、R15をSSR、SGRに退避します。

  例外コードH'1A0をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC＝VBR＋H'0100に分岐します。なお、H'FFFD以外の未定義命令をデコードした場合には動作を保証しません。

```
Slot_illegal_instruction_exception()
{
            SPC = PC - 2;
            SSR = SR;
            SGR = R15;
            EXPEVT = H'000001A0;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000100;
}
```

（11）一般 FPU 抑止例外

- **要因：遅延スロット以外にあるFPU命令*をSR.FD=1でデコード**
- **遷移先アドレス：**VBR + H'0000 0100
- **遷移時動作：**

  本例外を発生させた命令のPC、SRをそれぞれSPC、SSRに退避し、そのときのR15をSGRに退避します。

  例外コードH'800をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC＝VBR＋H'0100に分岐します。

```
General_fpu_disable_exception()
{
            SPC = PC;
            SSR = SR;
            SGR = R15;
            EXPEVT = H'00000800;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000100;
}
```

【注】 * FPU 命令とは命令コードの最初の 4 ビットが F である命令（ただし、未定義命令 H'FFFD を除く）と、FPUL、FPSCR に対する LDS、STS、LDS.L、STS.L 命令です。

（12）スロット FPU 抑止例外

- **要因：遅延スロットにあるFPU命令をSR.FD=1でデコード**
- **遷移先アドレス：**VBR + H'0000 0100
- **遷移時動作：**

  直前の遅延分岐命令のPCをSPCに退避します。本例外発生時のSR、R15をSSR、SGRに退避します。

  例外コードH'820をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC＝VBR＋H'0100に分岐します。

```
Slot_fpu_disable_exception()
{
            SPC = PC - 2;
            SSR = SR;
            SGR = R15;
            EXPEVT = H'00000820;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000100;
}
```

（13）命令実行前ユーザブレーク／命令実行後ユーザブレーク

- **要因：ユーザブレークポイントコントローラに設定したブレーク条件が成立**
- **遷移先アドレス：**VBR + H'0000 0100、またはDBR
- **遷移時動作：**

  命令実行後ブレークの場合、ブレークポイントを設定した命令の直後の命令のPCをSPCに退避します。命令実行前ブレークの場合、ブレークポイントを設定した命令のPCをSPCに退避します。

  ブレーク発生時のSR、R15をSSR、SGRに退避します。例外コードH'1E0をEXPEVTにセットします。

  SRのBLビット、MDビット、RBビットを1にセットし、PC＝VBR＋H'0100に分岐します。ただし、PC=DBRに分岐することも可能です。

  データブレークを設定した場合のPCについてなど、詳細は当該製品ハードウェアマニュアルの**「ユーザブレークコントローラ」**を参照してください。

```
User_break_exception()
{
            SPC = (pre_execution break? PC : PC + 2);
            SSR = SR;
            SGR = R15;
            EXPEVT = H'000001E0;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = (CBCR.UBDE==1 ? DBR : VBR + H’00000100);
}
```

（14）FPU 例外

- **要因：浮動小数点演算実行による例外**
- **遷移先アドレス**：VBR + H'0000 0100
- **遷移時動作**：

  本例外を発生させた命令のPC、SRをそれぞれSPC、SSRに退避し、そのときのR15をSGRに退避します。例外コードH'120をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC=VBR+H'0100に分岐します。

```
FPU_exception()
{
            SPC = PC;
            SSR = SR;
            SGR = R15;
            EXPEVT = H'00000120;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000100;
}
```

5.6.3 割り込み

（1） NMI（ノンマスカブル割り込み）

- **要因：NMI端子のエッジ検出**
- **遷移先アドレス**：VBR＋H'0000 0600
- **遷移時動作**：

  本割り込みを受け付けた命令の直後のPC、SRを、それぞれSPC、SSRに退避し、そのときのR15をSGRに退避します。

  例外コードH'1C0をINTEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC＝VBR＋H'0600に分岐します。本割り込みは、SRのBLビットが0のときはSRの割り込みマスクビットによってマスクされず、最優先で受け付けられます。SRのBLビットが1のとき本割り込みがマスクされるか、受け付けるかをソフトウェアによって設定可能です。詳細は当該製品ハードウェアマニュアルの**「割り込みコントローラ」**を参照してください。

```
NMI()
{
            SPC = PC;
            SSR = SR;
            SGR = R15;
            INTEVT = H'000001C0;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            If(cond)SR.IMASK = B'1111;
            PC = VBR + H'00000600;
}
```

(2) 一般割り込み要求

- **要因：**

  SRの割り込みマスクビットが割り込み要求の割り込みレベルより小さく、かつSRのBLが0（命令の切れ目で受け付けます。）

- **遷移先アドレス：**VBR + H'0000 0600

- **遷移時動作：**

  受け付けた命令の直後のPCをSPCにセットします。受け付けた時点のSR、R15をSSR、SGRにセットします。各割り込み要因に対応したコードをINTEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、VBR＋H'0600に分岐します。詳細は当該製品ハードウェアマニュアルの**「割り込みコントローラ」**を参照してください。

```
Module_interruption()
{
            SPC = PC;
            SSR = SR;
            SGR = R15;
            INTEVT = H'00000400 ~ H'00003FE0;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            if(cond)SR.IMASK = level_of_accepted_interrupt();
            PC = VBR + H'00000600;
}
```

5.6.4 複数回の例外が発生する場合の優先順位

メモリを 2 回アクセスする命令や、不可分である遅延付き分岐命令と遅延スロット命令などでは、複数回例外が発生します。この場合、通常の例外優先順位と異なるので、注意が必要です。

(1) メモリを 2 回アクセスする命令

MAC 命令やメモリ―メモリ間論理演算命令、TAS 命令、MOVUA 命令は 1 つの命令でデータ転送が 2 回あるため、それぞれのデータ転送時に例外の発生を検出します。そのため、以下の順位で判定します。

1. 1回目のデータ転送のデータアドレスエラー
2. 1回目のデータ転送のTLBミス
3. 1回目のデータ転送のTLB保護違反
4. 1回目のデータ転送の初期ページ書き込み例外
5. 2回目のデータ転送のデータアドレスエラー
6. 2回目のデータ転送のTLBミス
7. 2回目のデータ転送のTLB保護違反
8. 2回目のデータ転送の初期ページ書き込み例外

(2) 不可分である遅延付き分岐命令と遅延スロット命令

遅延付き分岐命令と遅延スロット命令は不可分であるため、1 つの命令として扱われます。そのため、それぞれの命令における例外についても、優先順位が通常と異なります。遅延スロット命令が 1 回のデータ転送しか持たない場合の順位を示します。

1. 遅延付き分岐命令における優先レベル1、2の中断型および再実行型例外をチェックします。
2. 遅延スロット命令における優先レベル1、2の中断型および再実行型例外をチェックします。
3. 遅延付き分岐命令における優先レベル2の完了型例外をチェックします。
4. 遅延スロット命令における優先レベル2の完了型例外をチェックします。
5. 遅延付き分岐命令における優先レベル3と遅延スロット命令における優先レベル3をチェックします（この2つの間の優先順位はありません）。
6. 遅延付き分岐命令における優先レベル4と遅延スロット命令における優先レベル4をチェックします（この2つの間の優先順位はありません）。

遅延スロット命令が 2 回目のデータ転送を持つ場合、2.において、（1）のように 2 回チェックを行います。

なお、受け付けた例外（最も優先度が高い例外）が遅延スロット命令の再実行型例外である場合、分岐命令の PR レジスタ書き込み動作（BSR、BSRF、JSR の PC→PR 動作）は抑止されません。ただし、その場合の PR レジスタの内容は保証されません。

5.7 注意事項

(1) 例外処理からの復帰

1. SRのBLビットをソフトウェアでチェックしてください。メモリにSPC、SSRを退避していた場合には、SRのBLビットを1にしてからそれらを回復してください。
2. RTE命令を発行してください。RTE命令により、SPCがPCに、SSRがSRにセットされ、SPCのアドレスに分岐して、例外処理から復帰します。

(2) SR.BL＝1 のときに例外または割り込みが発生した場合

1. 例外

   ユーザブレークを除く例外が発生した場合には、マニュアルリセットが発生します。このときEXPEVTは、H'0000 0020となり、SPC、SSRの各レジスタは不定値となります。

2. 割り込み

   通常の割り込みが発生した場合には、割り込み要求は保留され、ソフトウェアでSRのBLビットが0にクリアされてから受け付けられます。ノンマスカブル割り込み（NMI）が発生した場合は、保留するか、受け付けるかをソフトウェアによって設定可能です。

   ただし、スリープまたはスタンバイ状態では、SRのBLビットが1であっても、割り込みを受け付けます。

(3) 例外発生時の SPC

1. 再実行型の例外

   例外が発生した命令のPCがSPCにセットされ、例外処理から復帰後に再実行されます。ただし、遅延スロット命令で発生した場合、直前の遅延分岐命令の条件が成立する、しないに関係なく遅延分岐命令のPCがSPCにセットされます。

2. 完了型の例外、割り込み

   例外が発生した命令の次の命令のPCがSPCにセットされます。ただし、遅延スロット付き分岐命令で発生した場合、分岐先のPCがSPCにセットされます。

(4) RTE 命令の遅延スロット

1. RTE命令の遅延スロットに配置された命令は、SSRに退避されていた値がSRに復帰されたのち実行されます。命令アクセスに関する例外の受け付け判定は復帰前のSRの値に応じて決定され、その他の例外の受け付け判定は復帰後とのSRによる処理モードやBLビットに依存して決定されます。完了型の例外に関してはRTEの分岐先の実行前に受け付けられますが、再実行型の例外が発生すると動作が保証されません。
2. RTE命令の遅延スロットに配置された命令では、ユーザブレークの受け付けは行われません。

(5) SR レジスタ値変更と例外の受け付け

1. LDC命令によりSRレジスタのMDやBLビットを操作した場合は、その次命令から新しいSRレジスタの値で例外の受け付けを再判定します*。完了型例外では次命令の実行後に例外が受け付けられますが、完了型例外のうち、割り込みに関しては次命令の実行前に受け付けを行います。

【注】 * **SR に対する LDC 命令が実行されると、後続命令への命令フェッチが再び行われ、新しい SR の値で命令フェッチ例外の再評価が行われます。**

6. 浮動小数点ユニット（FPU）

6.1 概要

FPU には次のような特長があります。

- IEEE754規格に準拠
- 32本の単精度浮動小数点レジスタ（16本の倍精度レジスタとしても参照できます）
- 2つの丸めモード：近傍および0方向への丸め
- 2つの非正規化数処理モード：0へのフラッシュと非正規化数の扱い
- 6つの例外要因：

  FPUエラー、無効演算、0による除算、オーバフロー、アンダフロー、不正確
- 包括命令：

  単精度、倍精度、グラフィックサポート、システム制御
- SH-4Aで下記の3命令を追加しました。

  FSRRA、FSCA、FPCHG

SR の FD ビットを 1 にセットすると、浮動小数点ユニット（FPU）は使用できなくなり、FPU 命令を実行しようとすると FPU 抑止例外（一般 FPU 抑止例外またはスロット FPU 抑止例外）が発生します。

6.2 データフォーマット

6.2.1 浮動小数点フォーマット

浮動小数点は次の 3 つのフィールドから構成されています。

- 符号ビット（s）
- 指数フィールド（e）
- 小数フィールド（f）

SH-4A は**図 6.1** と**図 6.2** に示すフォーマットを用いて単精度、倍精度浮動小数点を扱うことができます。

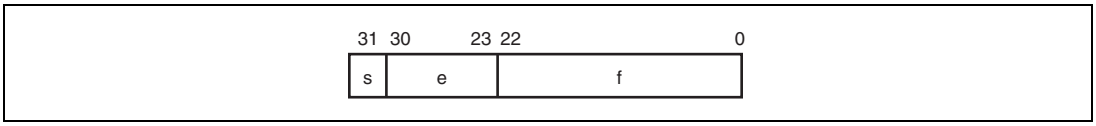


図 6.1 単精度浮動小数点フォーマット

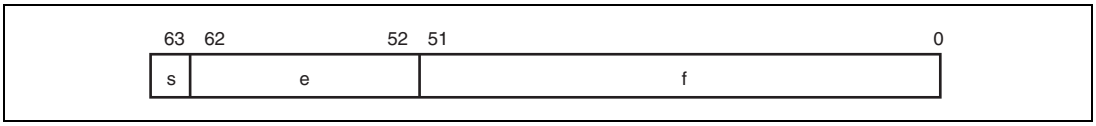


図 6.2 倍精度浮動小数点フォーマット

指数は次のようにバイアス付きで表します。

e＝E＋バイアス

バイアスのない指数 E の範囲は、E_{min} - 1 から E_{max} + 1 までです。E_{min} - 1 と E_{max} + 1 の 2 つの値は次のように区別します。E_{min} - 1 は 0（正、負両方の符号）と非正規化数を表し、E_{max} + 1 は正または負の無限大または非数（NaN）を表します。**表 6.1** に浮動小数点のフォーマットとパラメータを示します。

表 6.1 浮動小数点のフォーマットとパラメータ

パラメータ	単精度	倍精度
総ビット幅	32 ビット	64 ビット
符号ビット（s）	1 ビット	1 ビット
指数フィールド（e）	8 ビット	11 ビット
小数フィールド（f）	23 ビット	52 ビット
精度	24 ビット	53 ビット
バイアス	+127	+1023
E_{max}	+127	+1023
E_{min}	-126	-1022

浮動小数点の数値 v は次のようにして決められます。

- $E = E_{max} + 1$かつ$f \neq 0$の場合、vは符号sに関係なく非数（NaN）です。
- $E = E_{max} + 1$かつ$f = 0$の場合、vは$(-1)^s$（無限）「正または負の無限」です。
- $E_{min} \leqq E \leqq E_{max}$の場合、vは$(-1)^s 2^E$（1.f）「正規化数」です。
- $E = E_{min} - 1$かつ$f \neq 0$の場合、vは$(-1)^s 2^{Emin}$（0.f）「非正規化数」です。
- $E = E_{min} - 1$かつ$f = 0$の場合、vは$(-1)^s 0$「正または負の0」です。

表 6.2 に 16 進数による各タイプの範囲を示します。シグナリング非数とクワイアット非数については、**「6.2.2 非数（NaN）」**を、非正規化数については**「6.2.3 非正規化数」**を参照してください。

表 6.2 浮動小数点の範囲

タイプ	単精度	倍精度
シグナリング非数	H'7FFFFFFF～H'7FC00000	H'7FFFFFFF FFFFFFFF～H'7FF80000 00000000
クワイアット非数	H'7FBFFFFF～H'7F800001	H'7FF7FFFF FFFFFFFF～H'7FF00000 00000001
正の無限大	H'7F800000	H'7FF00000 00000000
正の正規化数	H'7F7FFFFF～H'00800000	H'7FEFFFFF FFFFFFFF～H'00100000 00000000
正の非正規化数	H'007FFFFF～H'00000001	H'000FFFFF FFFFFFFF～H'00000000 00000001
正のゼロ	H'00000000	H'00000000 00000000
負のゼロ	H'80000000	H'80000000 00000000
負の非正規化数	H'80000001～H'807FFFFF	H'80000000 00000001～H'800FFFFF FFFFFFFF
負の正規化数	H'80800000～H'FF7FFFFF	H'80100000 00000000～H'FFEFFFFF FFFFFFFF
負の無限大	H'FF800000	H'FFF00000 00000000
クワイアット非数	H'FF800001～H'FFBFFFFF	H'FFF00000 00000001～H'FFF7FFFF FFFFFFFF
シグナリング非数	H'FFC00000～H'FFFFFFFF	H'FFF80000 00000000～H'FFFFFFFF FFFFFFFF

6.2.2 非数（NaN）

図 6.3 に非数（NaN）のビットパターンを示します。次の場合の値は NaN です。

- 符号ビット　　：don't care
- 指数フィールド：すべてのビットが1
- 小数フィールド：少なくとも1ビットが1

NaN は、小数フィールドの MSB が 1 の場合はシグナリング非数（sNaN）であり、0 の場合はクワイアット非数（qNaN）です。

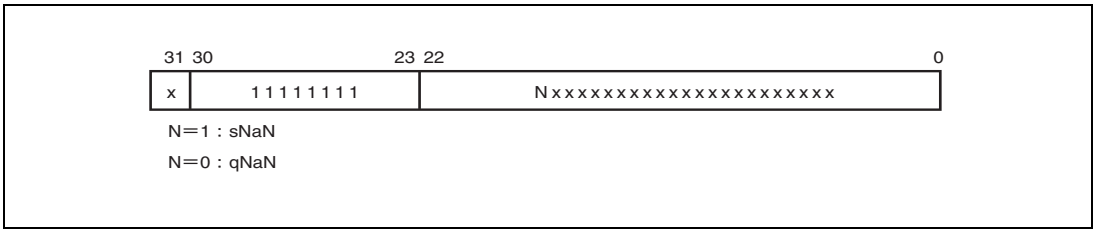


図 6.3　単精度の NaN ビットパターン

sNaN をレジスタ・レジスタ間の転送命令 FABS または FNEG 以外の浮動小数点値を生成する演算の入力データとすると、

- FPSCRレジスタのEN.Vビットが0の場合、演算結果（出力）はqNaNになります。
- FPSCRレジスタのEN.Vビットが1の場合、無効演算例外が発生します。この場合、演算のデスティネーションレジスタの内容は変更しません。

レジスタ・レジスタ間の転送命令には、下記の 3 命令があります。

- FMOV FRm,FRn
- FLDS FRm,FPUL
- FSTS FPUL,FRn

浮動小数点値を生成する演算で qNaN を入力し、その演算に sNaN を入力していない場合、FPSCR レジスタの EN.V ビットの設定に関係なく出力は常に qNaN です。この場合、例外は発生しません。

演算結果として SH-4A が生成する qNaN の値は、常に次のような値になります。

- 単精度qNaN：H'7FBFFFFF
- 倍精度qNaN：H'7FF7FFFF FFFFFFFF

非数（NaN）を入力した場合の浮動小数点演算の詳細については**「第 10 章　各命令の説明」**を参照してください。

6.2.3 非正規化数

非正規化数の浮動小数点値は、指数フィールドは 0 として、小数フィールドは 0 以外の値として表現します。

FPU のステータスレジスタ FPSCR の DN ビットが 1 の場合、非正規化数（ソースオペランドまたは演算結果）は、（レジスタ・レジスタ間の転送命令、FNEG、FABS 以外の演算の）値を生成する浮動小数点演算で正のゼロまたは負のゼロになります。

FPSCR の DN ビットが 0 の場合、非正規化数（ソースオペランドまたは演算結果）はそのまま処理されます。非正規化数を入力する場合の浮動小数点演算の詳細については、**「第 10 章　各命令の説明」**を参照してください。

6.3 レジスタ

6.3.1 浮動小数点レジスタ

図 6.4 に浮動小数点レジスタの構成を示します。32 本の 32 ビット浮動小数点レジスタがあります。これらは、2 つのバンクで構成され、FPR0_BANK0～FPR15_BANK0、FPR0_BANK1～FPR15_BANK1 があります。また、この 32 本レジスタは FR0～FR15、DR0/2/4/6/8/l0/12/14、FV0/4/8/12、XF0～XF15、XD0/2/4/6/8/l0/12/14、XMTRX として参照されます。FPRn_BANKi と参照名の対応は FPSCR の FR ビットによって決まります。

（1） 浮動小数点レジスタ　FPRn_BANKi（32 レジスタ）

FPR0_BANK0～FPR15_BANK0

FPR0_BANK1～FPR15_BANK1

（2） 単精度浮動小数点レジスタ　FRi（16 レジスタ）

FPSCR.FR＝0 のとき、FR0～FR15 は FPR0_BANK0～FPR15_BANK0 に割り当てられます。

FPSCR.FR＝1 のとき、FR0～FR15 は FPR0_BANK1～FPR15_BANK1 に割り当てられます。

（3） 倍精度浮動小数点レジスタ、または単精度浮動小数点レジスタのペア　DRi（8 レジスタ）

DR レジスタは、2 つの FR レジスタから構成されます。

DR0＝{FR0、FR1}、DR2＝{FR2、FR3}、

DR4＝{FR4、FR5}、DR6＝{FR6、FR7}、

DR8＝{FR8、FR9}、DR10＝{FR10、FR11}、

DR12＝{FR12、FR13}、DR14＝{FR14、FR15}

（4） 単精度浮動小数点ベクトルレジスタ　FVi（4 レジスタ）

FV レジスタは 4 つの FR レジスタから構成されます。

FV0＝{FR0、FR1、FR2、FR3},

FV4＝{FR4、FR5、FR6、FR7},

FV8＝{FR8、FR9、FR10、FR11},

FVl2＝{FR12、FR13、FR14、FR15}

（5） 単精度浮動小数点拡張レジスタ　XFi（16 レジスタ）

FPSCR.FR＝0 のとき、XF0～XF15 は FPR0_BANK1～FPR15_BANK1 に割り当てられます。

FPSCR.FR＝1 のとき、XF0～XF15 は FPR0_BANK0～FPR15_BANK0 に割り当てられます。

（6） 単精度浮動小数点拡張レジスタのペア　XDi（8 レジスタ）

XD レジスタは 2 つの XF レジスタから構成されます。

XD0＝{XF0、XF1}、XD2＝{XF2、XF3}、

XD4＝{XF4、XF5}、XD6＝{XF6、XF7}、

XD8＝{XF8、XF9}、XD10＝{XF10、XF11}、
XD12＝{XF12、XF13}、XD14＝{XF14、XF15}

(7) 単精度浮動小数点拡張レジスタ行列　XMTRX

XMTRX は 16 本の XF レジスタから構成されます。

$$XMTRX = \begin{pmatrix} XF0 & XF4 & XF8 & XF12 \\ XF1 & XF5 & XF9 & XF13 \\ XF2 & XF6 & XF10 & XF14 \\ XF3 & XF7 & XF11 & XF15 \end{pmatrix}$$

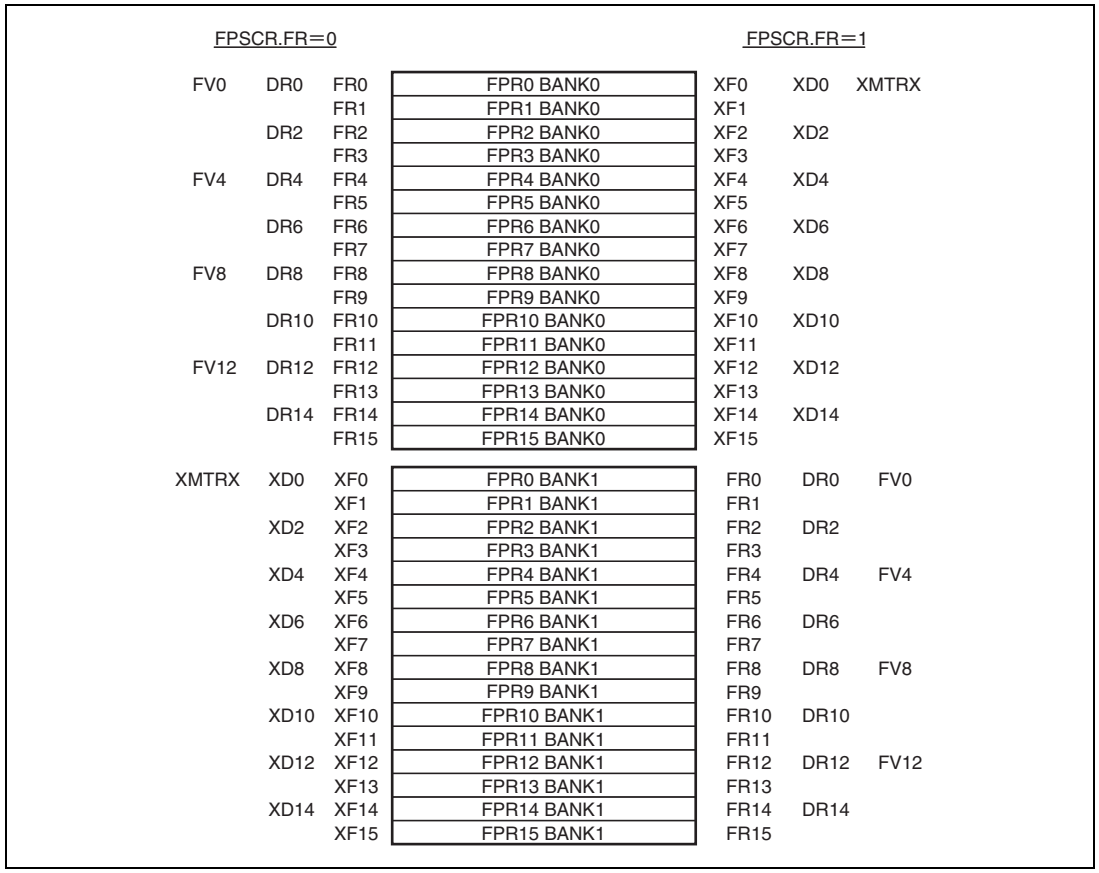


図 6.4　浮動小数点レジスタ

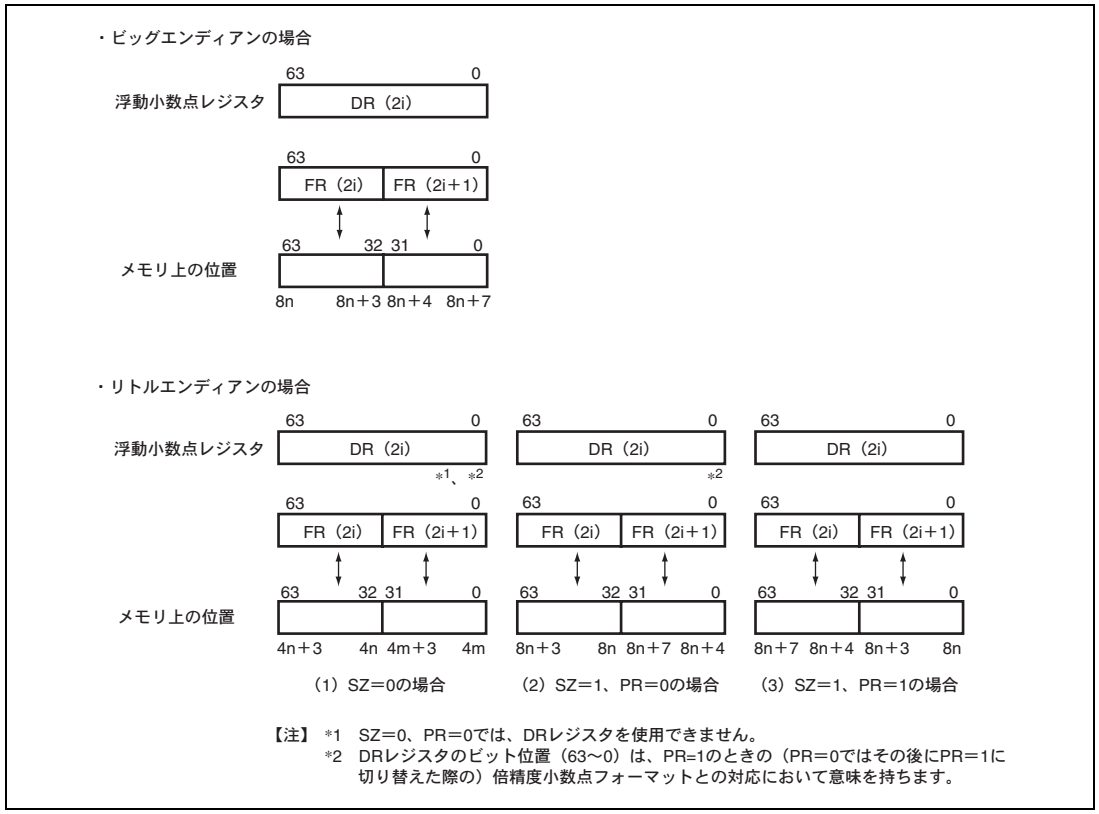
6.3.2 浮動小数点ステータス／コントロールレジスタ（FPSCR）

ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	FR	SZ	PR	DN	Cause	
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Cause				Enable（EN）					Flag					RM	
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
R/W：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ビット	ビット名	初期値	R/W	説　　明
31～22	−	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
21	FR	0	R/W	浮動小数点レジスタバンク 0：FPR0_BANK0～FPR15_BANK0 は FR0～FR15 に、FPR0_BANK1～FPR15_BANK1 は XF0～XF15 に割り当てられます。 1：FPR0_BANK0～FPR15_BANK0 は XF0～XF15 に、FPR0_BANK1～FPR15～BANK1 は FR0～FR15 に割り当てられます。
20	SZ	0	R/W	転送サイズモード 0：FMOV 命令のデータサイズは 32 ビットです。 1：FMOV 命令のデータサイズは 32 ビットペア、または 64 ビットです。 SZ ビットおよび PR ビットとエンディアンとの関係については、図 6.5 を参照してください。
19	PR	0	R/W	精度モード 0：浮動小数点命令を単精度演算として実行します。 1：浮動小数点命令を倍精度演算として実行します（グラフィックサポート命令は未定義です）。 PR ビットおよび SZ ビットとエンディアンとの関係については、図 6.5 を参照してください。
18	DN	1	R/W	非正規化モード 0：非正規化数を非正規化数として扱います。 1：非正規化数を 0 として扱います。

ビット	ビット名	初期値	R/W	説　　明
17～12	Cause	すべて 0	R/W	FPU 例外要因フィールド
11～7	Enable（EN）	すべて 0	R/W	FPU 例外イネーブルフィールド
6～2	Flag	すべて 0	R/W	FPU 例外フラグフィールド FPU 演算命令を実行すると、FPU 例外要因フィールドは最初に 0 に設定されます。次に FPU 例外が発生すると、FPU 例外要因フィールドと FPU 例外フラグフィールドの該当ビットが 1 にセットされます。 FPU 例外フラグフィールドは、FPU 例外フラグフィールドが最後にクリアされたそれ以降に発生した例外のステータスを保持します。 各フィールドのビットの割り付けについては表 6.3 を参照してください。
1、0	R	B'01	R/W	丸めモード 丸めの方法を選択します。 00：近傍への丸め 01：0 方向への丸め 10：リザーブ（設定禁止） 11：リザーブ（設定禁止）



【注】 ＊1　SZ＝0、PR＝0では、DRレジスタを使用できません。
＊2　DRレジスタのビット位置（63～0）は、PR=1のときの（PR＝0ではその後にPR＝1に切り替えた際の）倍精度小数点フォーマットとの対応において意味を持ちます。

図 6.5　SZ ビットとエンディアンの関係

表 6.3　FPU 例外処理に関連するビットの割り付け

		FPU エラー（E）	無効演算（V）	0 除算（Z）	オーバフロー（O）	アンダフロー（U）	不正確（I）
Cause	FPU 例外要因フィールド	ビット 17	ビット 16	ビット 15	ビット 14	ビット 13	ビット 12
Enable	FPU 例外イネーブルフィールド	なし	ビット 11	ビット 10	ビット 9	ビット 8	ビット 7
Flag	FPU 例外フラグフィールド	なし	ビット 6	ビット 5	ビット 4	ビット 3	ビット 2

6.3.3　浮動小数点通信レジスタ（FPUL）

FPU と CPU 間の情報伝達は FPUL レジスタを介して行われます。FPUL レジスタは 32 ビットのシステムレジスタで、LDS、STS 命令によって CPU からもアクセスします。たとえば、汎用レジスタ R1 に格納した整数を単精度浮動小数点に変換する処理フローは次のとおりです。

R1 →　（LDS 命令）→　FPUL　→　（単精度 FLOAT 命令）　→　FR1

6.4　丸め

浮動小数点命令において、丸めは中間結果から最終演算結果を生成する際に実行されます。したがって、FMAC、FTRV、FIPR のような組み合わせ命令の結果は、FADD、FSUB、FMUL などの基本命令だけを用いた結果とは異なります。FMAC は 1 度、FADD、FSUB および FMUL は 2 度というように丸めの回数が異なるためです。

丸めには 2 つの方法があり、使用する方法は FPSCR の RM フィールドで決まります。

RM=00：近傍への丸め

RM=01：0 方向への丸め

（1）近傍への丸め

演算結果はもっとも近い表現可能な値に丸められます。もっとも近い表現可能な値が 2 つある場合、LSB が 0 の方を選択します。

丸め前の値が 2^{Emax}（$2-2^{-p}$）以上であれば丸め前と同じ符号の無限となります。ここで Emax、p は単精度でそれぞれ 127、24、倍精度で 1023、53 です。

（2）0 方向への丸め

丸め前の値の丸めビット以下の桁は切り捨てられます。

ただし、丸め前の値が表現可能な最大絶対値数よりも絶対値が大きい場合、丸め前と同じ符号の表現可能な最大絶対値の数になります。

6.5 浮動小数点例外

FPU 関連の例外は次のとおりです。

(1) 一般 FPU 抑止／スロット FPU 抑止例外

SR.FD＝1 のときに FPU 命令を実行すると発生します。FPU 命令が遅延スロット以外にある場合は一般 FPU 抑止例外が、FPU 命令が遅延スロットにある場合はスロット FPU 抑止例外が発生します。

(2) FPU 例外

例外要因は次のとおりです。

- FPUエラー（E）：

  FPSCR.DN＝0かつ非正規化数の入力時

- 無効演算（V）：

  NaN入力のような無効な演算の場合

- 0による除算（Z）：

  除数0による除算

- オーバフロー（O）：

  演算結果がオーバフローする場合

- アンダフロー（U）：

  演算結果がアンダフローする場合

- 不正確例外（I）：

  丸めが発生する場合

FPSCR の FPU 例外要因フィールドには上記 E、V、Z、O、U、I のすべてに該当するビットが含まれ、FPSCR のフラグおよびイネーブルフィールドには V、Z、O、U、I に該当するビットが含まれていますが E に該当するビットは含まれていません。このように FPU エラーはディスエーブルにすることができません。

FPU 例外が発生すると、FPU 例外要因フィールドの該当するビットは 1 にセットされ FPU 例外フラグフィールドに該当するビットに 1 が累積されます。FPU 例外が発生しない場合、FPU 例外要因フィールドの該当するビットは 0 にクリアされ、FPU 例外フラグフィールドに該当するビットは変更されません。

(3) FPU 例外処理

FPU 例外は次の場合に発生します。

- FPUエラー（E）：

  FPSCR.DN＝0かつ非正規化数を扱えない命令への非正規化数の入力時

- 無効演算（V）

  ：FPSCR.EN.V＝1かつ（命令=FTRVまたは無効演算）の場合

- 0による除算（Z）

  ：FPSCR.EN.Z＝1かつ除数0による除算またはFSRRAの入力が0の場合

- オーバフロー（O）

  ：FPSCR.EN.O＝1かつ演算結果がオーバフローする可能性のある場合

- アンダフロー（U）

  ：FPSCR.EN.U＝1かつ演算結果がアンダフローする可能性のある場合

- 不正確例外（I）

  ：FPSCR.EN.I＝1かつ演算結果が不正確になる可能性のある命令

FPU 演算に起因するすべての例外事象は、同一の例外事象として割り付けられています。例外の意味内容は、システムレジスタ FPSCR を読み出して、保持されている情報を解釈することでソフトウェアにより決定します。

また、いかなる FPU 例外処理動作によっても、デスティネーションレジスタは変更されません。

上記以外で FPU 例外要因が発生すると、V、Z、O、U、I に対する該当ビットを 1 にセットし、演算結果としてデフォルト値を生成します。

- 無効演算（V）：

  結果としてqNaNを生成します。

- 0による除算（Z）：

  丸め前と同じ符号付きの無限大を生成します。

- オーバフロー（O）：

  0方向への丸めのとき、丸め前と同じ符号付き最大正規化数を生成します。

  近傍への丸めのとき、丸め前と同じ符号付き無限大を生成します。

- アンダフロー（U）：

  FPSCR.DN＝0のとき、丸め前と同じ符号付き非正規化数、または丸め前と同じ符号付き0を生成します。

  FPSCR.DN＝1のとき、丸め前と同じ符号付き0を生成します。

- 不正確例外（I）：

  不正確な結果を生成します。

6.6 グラフィックサポート機能

SH-4A は 2 種類のグラフィック機能をサポートしています。1 つはジオメトリック演算用の命令であり、もう一つは高速データ転送を可能にするペア単精度転送命令です。

6.6.1 ジオメトリック演算命令

ジオメトリック演算命令は最小のハードウェアで高速演算を可能とするため、SH-4A は 4 つの乗算の部分的演算結果のうち相対的に小さな値を無視します。したがって、演算結果には以下に示す誤差が生じます。

最大誤差＝MAX（各乗算結果×$2^{-MIN（乗数の有効数字桁数－1、被乗数の有効数字桁数－1）}$）＋MAX（結果値×$2^{-23}$、$2^{-149}$）

ただし、有効数字桁数は正規化数が 24、非正規化数が 23（小数部のリーディングゼロの桁数）となります。

将来の SuperH シリーズでの演算誤差は保証しますが、異なるプロセッサコア間の同一の演算結果は保証しません。

（1） FIPR FVm,FVn（m、n：0、4、8、12）

この命令の用途例を以下に示します。

- 内積（m≠n）：

  一般的に、この演算はポリゴン表面の輝度や表面／裏面を判定するために使用されます。

- 各要素の平方和（m=n）：

  一般的に、この演算はベクトルの長さを得るために使用されます。

FIPR 命令は不正確例外を検出しないため、命令を実行すると、FPU 例外要因フィールドおよび FPU 例外フラグフィールドの不正確例外（I）ビットが常に 1 にセットされます。したがって、FPU 例外イネーブルフィールドの I ビットがセットされていれば、FPU 例外処理が実行されます。

（2） FTRV XMTRX、FVn（n：0、4、8、12）

この命令の用途例を以下に示します。

- 行列（4×4）・ベクトル（4）：

  一般的に、この演算は、視点の変更、角度の変更、または移動といったベクトル変換（4次元）に使用されます。基本的に、角度＋平行移動のためのアフィン変換処理は、4×4行列を必要とします。したがって、SH-4A は4次元演算をサポートしています。

- 行列（4×4）×行列（4×4）：

  この演算を行うためには、FTRV命令を4回実行する必要があります。

FIRV 命令は不正確例外を検出しないため、命令を実行すると、FPU 例外要因フィールドおよび FPU 例外フラグフィールドの不正確例外（I）ビットが常に 1 にセットされます。したがって、イネーブルフィールドの I ビッ

トがセットされていれば、FPU 例外処理が実行されます。また、FTRV 命令の実行の際、レジスタ内のすべてのデータタイプを実行前にチェックすることができません。FPU 例外イネーブルフィールドの V ビットがセットされていると、FPU 例外処理が実行されます。

（3） FRCHG

この命令はバンクレジスタを変更します。例えば、FTRV 命令を使用する場合、背後にあるバンク上に行列の要素を設定する必要があります。しかし、変換行列の要素自体を作成するには、前面にあるバンクのレジスタを使用する方が簡単です。FPSCR に対する LDS 命令を使用すると、この命令は FPU の状態を維持するために、4～5 サイクルを費やします。FRCHG 命令では FPSCR.FR ビットの変更を 1 サイクルで行うことができます。

6.6.2 ペア単精度データ転送

強力なジオメトリック演算命令に加えて、SH-4A は高速データ転送命令をサポートしています。
FPSCR.SZ＝1 のとき、ペア単精度データ転送命令によるデータ転送を行えます。

- FMOV DRm/XDm、DRn/XDRn（m、n：0、2、4、6、8、10、12、14）
- FMOV DRm/XDm、@Rn（m：0、2、4、6、8、10、12、14、n：0～15）

  これらの命令により、2つの単精度（2×32ビット）データを転送することができます。つまり、これらの命令の転送性能が2倍となります。

- FSCHG

  この命令はFPSCRのSZビットの値を変更します。ペア単精度データ転送を行うか行わないかを高速に切り換えることができます。

7. メモリマネジメントユニット（MMU）

SH-4A は、8 ビットのアドレス空間識別子と 32 ビットの仮想アドレス空間から 29 ビットの物理アドレス空間を扱うことができます。仮想アドレスから物理アドレスへのアドレス変換は、SH-4A に内蔵されたメモリマネジメントユニット（MMU：Memory Management Unit）を用いて行います。MMU は変換ルックアサイドバッファ（TLB：Translation Lookaside Buffer）にユーザ作成のアドレス変換テーブルの情報をキャッシングすることにより、高速にアドレス変換を行います。

SH-4A は命令 TLB（ITLB）を 4 エントリ、共用 TLB（UTLB）を 64 エントリ内蔵しており ITLB には UTLB のコピーがハードウェアにより格納されます。アドレス変換方式はページング方式で、4 種類（1K/4K/64K/1M バイト）のページサイズをサポートしています。また特権モード、ユーザモードのそれぞれにおいて、仮想アドレス空間へのアクセス権を設定し、記憶保護を行うことができます。

7.1 MMU の概要

MMU とは物理メモリを有効に利用するために考え出された機能です。**図 7.1（0）**に示すように、プロセスのサイズが物理メモリより少ない場合、プロセスのすべてを物理メモリへマッピングすることが可能です。しかしプロセスのサイズが増大し、物理メモリに収まらない場合、プロセスを分割して実行に必要な部分を随時物理メモリへマッピングする必要が生じます（**図 7.1（1）**）。この物理メモリへのマッピングをプロセス自身が考えながら実行していては、プロセスにかかる負担が増大します。この負担を軽減するために物理メモリへのマッピングを一括して行おうとして生まれた考え方が仮想記憶方式です（**図 7.1（2）**）。仮想記憶方式では物理メモリに比べて十分に大きな仮想メモリを用意します。プロセスはこの仮想メモリにマッピングされます。このためプロセスは仮想メモリ上での動作だけを考えていればよくなります。仮想メモリから物理メモリへのマッピングには、MMU が用いられます。通常、OS が MMU を管理しており、プロセスが必要とする仮想メモリを円滑に物理メモリへマッピングできるように物理メモリの入れ換えを行います。物理メモリの入れ換えは 2 次記憶などとの間で行われます。

こうして生まれた仮想記憶方式は、複数のプロセスが同時に走行するタイムシェアリングシステム（TSS）の上で威力を発揮します（**図 7.1（3）**）。TSS 上で走行する複数のプロセスが、おのおの物理メモリへのマッピングを意識しながら動作していたのでは効率が上がりません。この効率を上げ、各プロセスの負担を減らすために仮想記憶方式は使われます（**図 7.1（4）**）。この仮想記憶方式ではプロセスごとに仮想メモリが割り当てられます。MMU は複数の仮想メモリを効率よく物理メモリへマッピングする働きをします。さらに、あるプロセスが別のプロセスの物理メモリに誤ってアクセスしないように、MMU には記憶保護の機能も備わっています。

MMU を用いて仮想メモリから物理メモリへアドレス変換を行うとき、その変換情報が MMU に登録されていなかったり、別のプロセスの仮想メモリへ誤ってアクセスしたりすることがあります。そのとき MMU は例外を発生させて、物理メモリのマッピングを変更し、新たなアドレス変換情報を登録します。

MMU の機能はソフトウェアのみでも実現可能ですが、プロセスが物理メモリへアクセスするたびにソフトウェアで変換を行っていたのでは効率が悪くなります。そのためハードウェア上にアドレス変換のためのバッファ（TLB）を用意し、頻繁に使用されるアドレス変換情報は TLB に置いておきます。TLB はアドレス変換情報のためのキャッシュといえます。しかしキャッシュと違いアドレス変換に失敗したとき、つまり例外が発生したときのアドレス変換情報の入れ換えは通常ソフトウェアで行います。このためソフトウェアで柔軟にメモリ管理を行うことが可能となります。

MMU が仮想メモリから物理メモリへのマッピングをする方式として、固定長のアドレス変換を用いる方式（ページング方式）と可変長のアドレス変換を用いる方式（セグメント方式）があります。ページング方式では固定サイズのページと呼ばれるアドレス空間が変換の単位となります。

以下、SH-4A では仮想メモリ上のアドレス空間のことを仮想アドレス空間、物理メモリ上のアドレス空間のことを物理アドレス空間と呼ぶことにします。

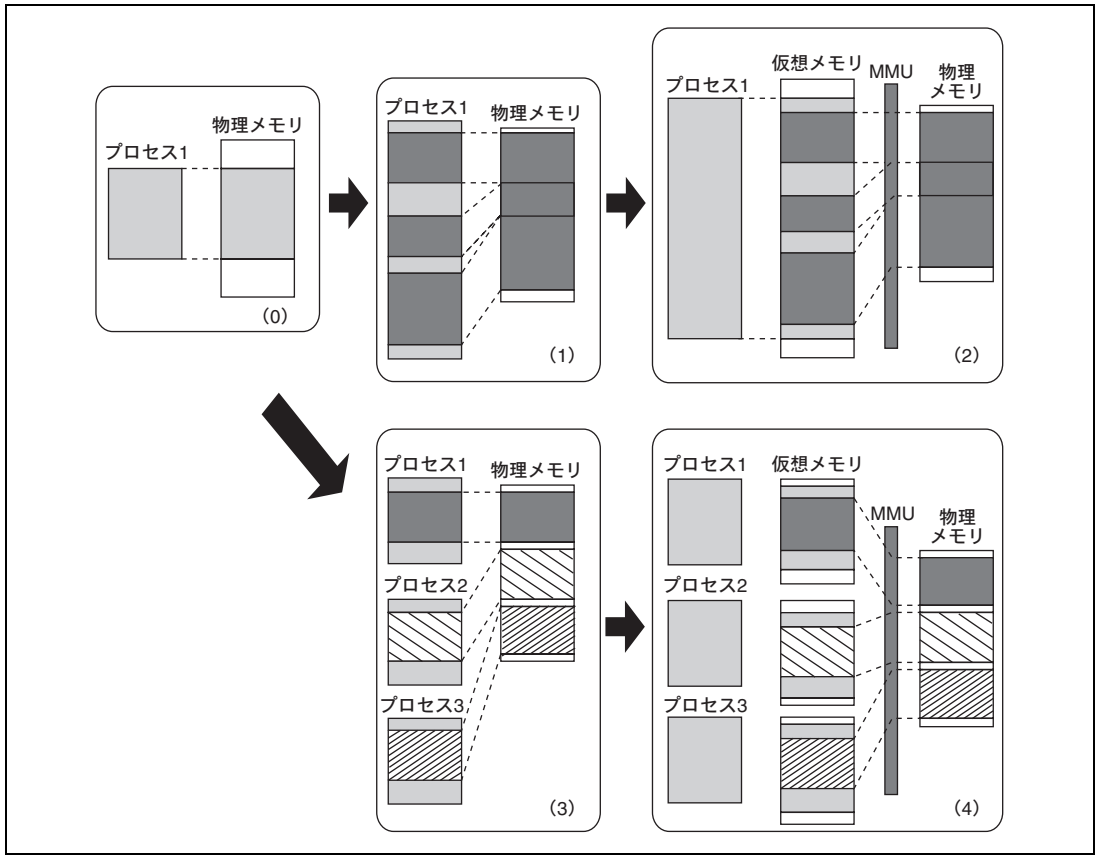


図 7.1　MMU の役割

7.1.1 アドレス空間

（1） 仮想アドレス空間

SH-4A は 32 ビットの仮想アドレス空間をサポートし、4G バイトのアドレス空間をアクセスできます。仮想アドレス空間は**図 7.2**、**図 7.3** に示すとおり、いくつかの領域に分かれています。特権モードでは P0 領域から P4 領域の 4G バイトの空間をアクセスすることが可能です。ユーザモードでは U0 領域の 2G バイトの空間をアクセス可能です。また MMU 制御レジスタ（MMUCR）の SQMD ビットが 0 の場合、ストアキュー領域の 64M バイトの空間もアクセス可能になり、内蔵メモリ制御レジスタ（RAMCR）の RMD ビットが 1 の場合、内蔵メモリ領域の 16M バイトの空間もアクセス可能になります。ユーザモードで U0 領域、ストアキュー領域、内蔵メモリ領域以外をアクセスした場合、アドレスエラーとなります。

MMUCR の AT ビットを 1 にし、MMU をイネーブルにしたとき、これらの領域のうち、P0、P3、U0 領域は、任意の物理アドレス空間へ 1K/4K/64K/1M バイトページ単位でマッピングすることができます。また 8 ビットのアドレス空間識別子を用いることにより、P0、P3、U0 領域を 256 個まで増やすことが可能です。仮想アドレス空間から 29 ビットの物理アドレス空間へのマッピングには TLB を用います。

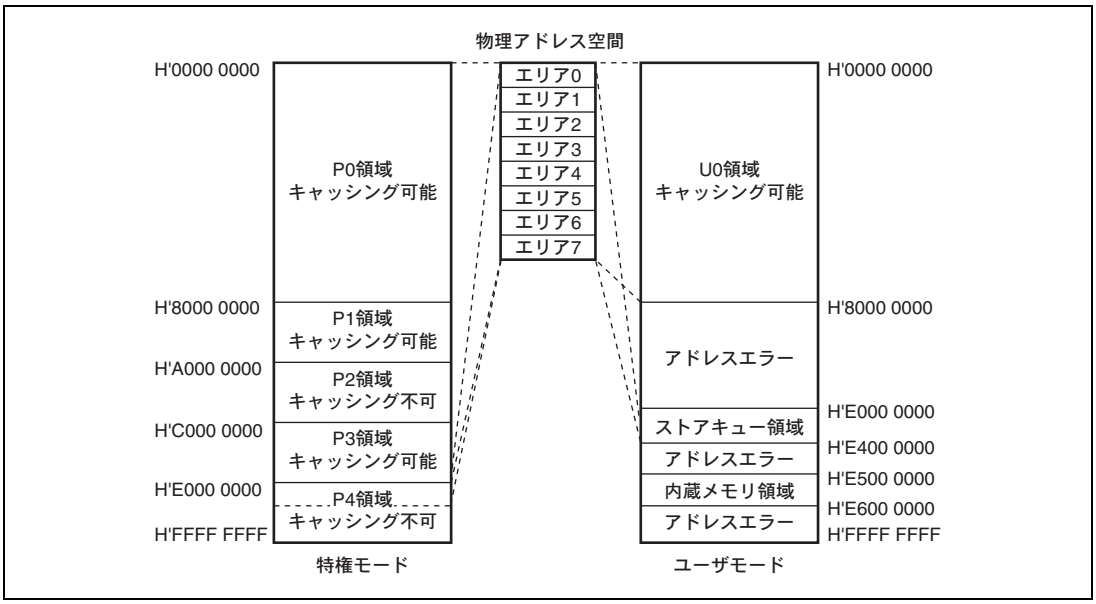


図 7.2　仮想アドレス空間（MMUCR.AT＝0）

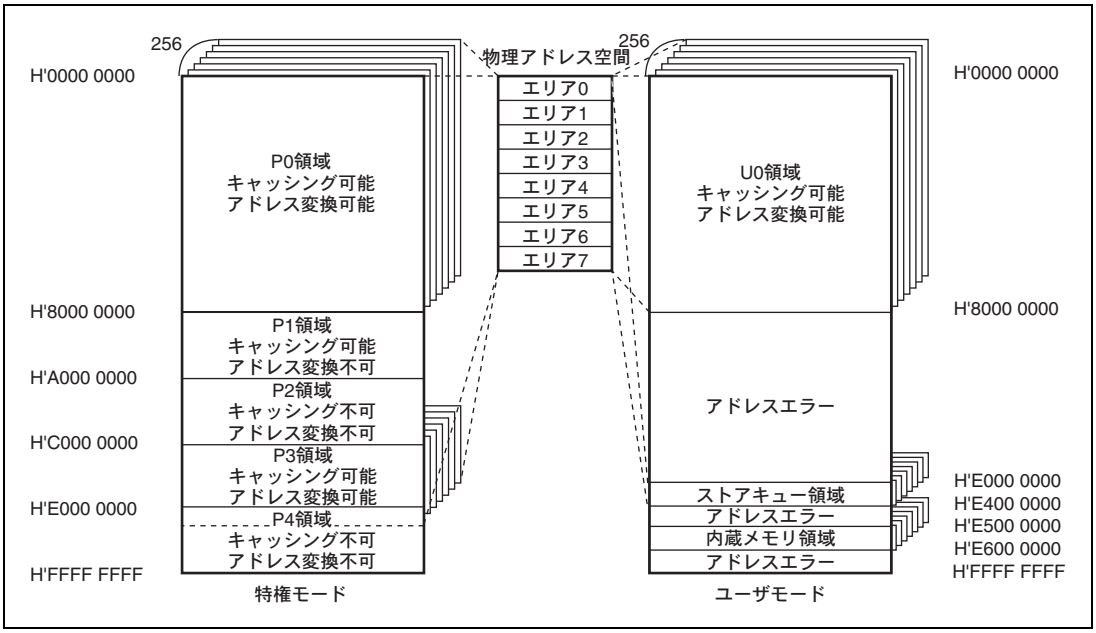


図 7.3　仮想アドレス空間（MMUCR.AT＝1）

(a)　P0、P3、U0 領域

P0、P3、U0 領域は TLB を用いたアドレス変換とキャッシュを用いたアクセスが可能な領域です。

MMU がディスエーブルの場合、アドレスの上位 3 ビットを 0 にしたものが対応する物理アドレス空間のアドレスとなります。キャッシュを用いるか否かはキャッシュコントロールレジスタ（CCR）に従います。キャッシュを用いた場合、ライトアクセスにおけるコピーバック方式とライトスルー方式の切り替えは、CCR の WT ビットに従います。

MMU がイネーブルの場合、これらの領域は TLB を用いて 1K/4K/64K/1M バイトページ単位に任意の物理アドレス空間へマッピングできます。CCR がキャッシュイネーブル状態であり、かつ TLB エントリの当該ページのキャッシング可能ビット（C ビット）が 1 のとき、キャッシュを用いたアクセスが行えます。キャッシュを用いた場合、ライトアクセスにおけるコピーバック方式とライトスルー方式の切り替えは、TLB の WT ビットに従います。

これらの領域を、TLB により物理アドレス空間のエリア 7 に存在する制御レジスタ領域にマッピングする場合、当該ページの C ビットは 0 にしてください。

(b)　P1 領域

P1 領域は TLB を用いたアドレス変換が行えませんが、キャッシュを用いたアクセスは可能な領域です。

MMU がイネーブルか否かにかかわらず、アドレスの上位 3 ビットを 0 にしたものが対応する物理アドレス空間のアドレスとなります。キャッシュを用いるか否かは CCR に従います。キャッシュを用いた場合、ライトアクセスにおけるコピーバック方式とライトスルー方式の切り替えは、CCR の CB ビットに従います。

(c)　P2 領域

P2 領域は TLB を用いたアドレス変換とキャッシュを用いたアクセスが行えない領域です。

MMU がイネーブルか否かにかかわらず、アドレスの上位 3 ビットを 0 にしたものが対応する物理アドレス空間のアドレスとなります。

（d） P4 領域

P4 領域は SH-4A の内部リソースにマッピングされる領域です。この領域は、ストアキューと内蔵メモリ領域を除いて TLB を用いたアドレス変換ができません。また、この領域はキャッシュを用いたアクセスが行えません。P4 領域の詳細を図 7.4 に示します。

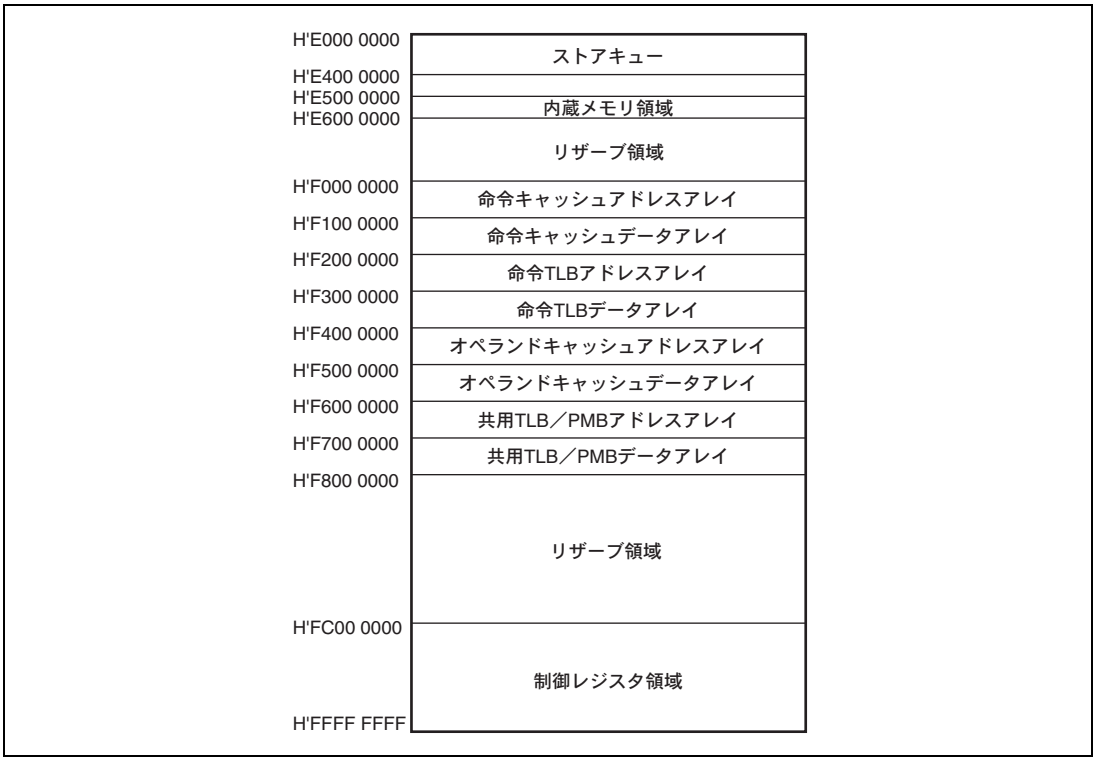


図 7.4 P4 領域

H'E000 0000～H'E3FF FFFF までは、ストアキュー（SQ）にアクセスするための領域です。ユーザモードでのアクセス権は MMUCR の SQMD ビットで指定します。詳細は「8.7 ストアキュー」を参照してください。

H'E500 0000～H'E5FF FFFF までは、内蔵メモリをアクセスするための領域です。ユーザモードでのアクセス権は RAMCR レジスタの RMD ビットで指定します。詳細は「第 9 章 L メモリ」を参照してください。

H'F000 0000～H'F0FF FFFF までは、命令キャッシュのアドレスアレイを直接アクセスするための領域です。詳細は「8.6.1 IC アドレスアレイ」を参照してください。

H'F100 0000～H'F1FF FFFF までは、命令キャッシュのデータアレイを直接アクセスするための領域です。詳細は「8.6.2 IC データアレイ」を参照してください。

H'F200 0000～H'F2FF FFFF までは、命令 TLB のアドレスアレイを直接アクセスするための領域です。詳細は「7.6.1 ITLB アドレスアレイ」を参照してください。

H'F300 0000～H'F37F FFFF までは、命令 TLB のデータアレイを直接アクセスするための領域です。詳細は「7.6.2 ITLB データアレイ」を参照してください。

H'F400 0000～H'F4FF FFFF までは、オペランドキャッシュのアドレスアレイを直接アクセスするための領域です。詳細は**「8.6.3　OC アドレスアレイ」**を参照してください。

H'F500 0000～H'F5FF FFFF までは、オペランドキャッシュのデータアレイを直接アクセスするための領域です。詳細は**「8.6.4　OC データアレイ」**を参照してください。

H'F600 0000～H'F60F FFFF までは、共用 TLB のアドレスアレイを直接アクセスするための領域です。詳細は**「7.6.3　UTLB アドレスアレイ」**を参照してください。

H'F610 0000～H'F61F FFFF までは、PMB のアドレスアレイを直接アクセスするための領域です。詳細は、**「7.7.5 メモリ割り付け PMB の構成」**を参照してください。

H'F700 0000～H'F70F FFFF までは、共用 TLB のデータアレイを直接アクセスするための領域です。詳細は、**「7.6.4 UTLB データアレイ」**を参照してください。

H'F710 0000～H'F71F FFFF までは、PMB のデータアレイを直接アクセスするための領域です。詳細は、**「7.7.5 メモリ割り付け PMB の構成」**を参照してください。

H'FC00 0000～H'FFFF FFFF までは内蔵周辺モジュールの制御レジスタの領域です。詳細はハードウエアマニュアルの各章のレジスタ説明の項を参照してください。

（2） 物理アドレス空間

SH-4A は 29 ビットの物理アドレス空間をサポートします。物理アドレス空間は**図 7.5** に示すとおり 8 つの領域に分かれています。エリア 7 はリザーブ領域です。詳細は当該製品ハードウエアマニュアルの**「バスステートコントローラ」**の章を参照してください。

TLB を用いて物理アドレス空間のエリア 7 をアクセスする場合のみ、エリア 7 の H'1C00 0000～H'1FFF FFFF までの領域がリザーブ領域ではなくなり、仮想アドレス空間の P4 領域に含まれる制御レジスタ領域と等価になります。

図 7.5　物理アドレス空間

（3） アドレス変換

MMU を使用するとき、仮想アドレス空間はページという単位に分割され、そのページ単位で物理アドレスに変換されます。外部メモリ上のアドレス変換テーブルには、仮想アドレスに対応する物理アドレスや、記憶保護コードなどの付加情報が格納され、TLB にはアドレス変換の高速化のために、外部メモリ上のアドレス変換テーブルの内容がキャッシングされます。SH-4A では命令のアクセスには ITLB を、データのアクセスには UTLB を用います。P4 領域以外へのアクセスが発生するとそのアクセスされた仮想アドレスが物理アドレスへ変換されます。その仮想アドレスが P1、P2 領域に属する場合、TLB をアクセスせずに物理アドレスが一意に決定されます。その仮想アドレスが P0、U0、P3 領域に属する場合には、仮想アドレスで TLB が検索され、その仮想アドレスが TLB に登録されている場合には、TLB ヒットとなり、TLB から対応する物理アドレスが読み出されます。またアクセスされた仮想アドレスが TLB に登録されていない場合には、TLB ミス例外が発生し、処理が TLB ミス例外処理ルーチンへ移ります。TLB ミス例外処理ルーチンでは、外部メモリ上のアドレス変換テーブルを検索し、対応する物理アドレス、ページ管理情報を TLB に登録します。そして例外処理ルーチンから復帰後、TLB ミス例外を発生させた命令を再実行します。

（4） 単一仮想記憶モードと多重仮想記憶モード

仮想記憶方式には、単一仮想記憶方式と多重仮想記憶方式があり、MMUCR の SV ビットにより選択が可能です。単一仮想記憶方式では、複数のプロセスが仮想アドレス空間を排他的に使用しながら同時に走行し、ある仮想アドレスに対応する物理アドレスは一意に定まります。多重仮想記憶方式では、複数のプロセスが仮想アドレス空間を共有して使用しながら走行するため、ある仮想アドレスはプロセスにより異なった物理アドレスに変換され得ます。単一仮想記憶方式と多重仮想記憶方式との動作上の違いは、TLB のアドレス比較の方式（**「7.3.3　アドレス変換方式」**参照）のみです。

（5） アドレス空間識別子（ASID）

多重仮想記憶モードの場合、8 ビットのアドレス空間識別子（ASID）は仮想アドレス空間を共有しながら同時に走行する複数のプロセスを区別するために用いられます。ASID は 8 ビットで、ソフトウェアが MMU 内の PTEH に現在走行中のプロセスの ASID をセットすることで設定可能です。また ASID によってプロセスを切り替えの際に TLB をパージしないで済みます。

単一仮想記憶モードの場合、ASID は仮想アドレス空間を排他的に使用しながら同時に走行する複数のプロセスの記憶保護のために用いられます。

【注】　単一仮想記憶モードの設定で、ASID が異なる同一の仮想ページ番号（VPN）を持つエントリを複数同時に TLB に設定してはいけません。

7.2 レジスタの説明

MMU 処理に関するレジスタを以下に示します。

表 7.1 レジスタ構成

名称	略称	R/W	P4 領域 アドレス*	エリア 7 アドレス*	サイズ
ページテーブルエントリ上位レジスタ	PTEH	R/W	H'FF00 0000	H'1F00 0000	32
ページテーブルエントリ下位レジスタ	PTEL	R/W	H'FF00 0004	H'1F00 0004	32
変換テーブルベースレジスタ	TTB	R/W	H'FF00 0008	H'1F00 0008	32
TLB 例外アドレスレジスタ	TEA	R/W	H'FF00 000C	H'1F00 000C	32
MMU 制御レジスタ	MMUCR	R/W	H'FF00 0010	H'1F00 0010	32
物理アドレス空間制御レジスタ	PASCR	R/W	H'FF00 0070	H'1F00 0070	32
命令再フェッチ抑止制御レジスタ	IRMCR	R/W	H'FF00 0078	H'1F00 0078	32

【注】 * P4 領域アドレスは、仮想アドレス空間の P4 領域を用いた場合のものです。エリア 7 アドレスは、TLB を用いて物理アドレス空間のエリア 7 からアクセスするものです。

表 7.2 各処理状態におけるレジスタの状態

名称	略称	パワーオン リセット	マニュアル リセット	スリープ	スタンバイ
ページテーブルエントリ上位レジスタ	PTEH	不定	不定	保持	保持
ページテーブルエントリ下位レジスタ	PTEL	不定	不定	保持	保持
変換テーブルベースレジスタ	TTB	不定	不定	保持	保持
TLB 例外アドレスレジスタ	TEA	不定	保持	保持	保持
MMU 制御レジスタ	MMUCR	H'0000 0000	H'0000 0000	保持	保持
物理アドレス空間制御レジスタ	PASCR	H'0000 0000	H'0000 0000	保持	保持
命令再フェッチ抑止制御レジスタ	IRMCR	H'0000 0000	H'0000 0000	保持	保持

7.2.1 ページテーブルエントリ上位レジスタ（PTEH）

PTEH は仮想ページ番号（VPN）とアドレス空間識別子（ASID）から構成されています。VPN は MMU 例外またはアドレスエラー例外が発生した際に、ハードウェアにより例外を発生させた仮想アドレスの VPN が設定されます。VPN はページサイズによって異なりますが、例外発生時にハードウェアにより設定される VPN は例外を発生させた仮想アドレスの上位 22 ビットとなります。VPN の設定はソフトウェアにより行うことも可能です。ASID には現在実行中のプロセスの番号をソフトウェアにより設定します。ASID がハードウェアにより更新されることはありません。この VPN と ASID は、LDTLB 命令により UTLB に登録されます。

PTEH レジスタの ASID フィールドを更新後、更新後の ASID 値を使用する P0、P3、U0 領域へのアクセス（命令フェッチを含む）を行う前に、以下の 1～3 のいずれかを実行してください。

1. RTE命令による分岐を実行してください。この場合、分岐先はP0、P3、U0領域でかまいません。
2. 任意のアドレス（キャッシング不可領域でもよい）に対して、ICBI命令を実行してください。
3. PTEH更新の前にあらかじめIRMCR.RE2＝0（初期値）と設定されていた場合には、特定の命令の実行は不要です。しかしこの方法では、PTEH更新命令の次命令を命令フェッチからやり直すため、CPUの処理性能が低下しますのでご注意ください。

ただし、方法 3 は今後の SuperH シリーズでは保証されない可能性があります。今後の SuperH シリーズでの互換性を保証するためには、1 または 2 を用いることを推奨します。

ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	VPN															
初期値：	－	－	－	－	－	－	－	－	－	－	－	－	－	－	－	－
R/W：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	VPN						－	－	ASID							
初期値：	－	－	－	－	－	－	0	0	－	－	－	－	－	－	－	－
R/W：	R/W	R/W	R/W	R/W	R/W	R/W	R	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ビット	ビット名	初期値	R/W	説　　明
31～10	VPN	－	R/W	仮想ページ番号
9、8	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
7～0	ASID	－	R/W	アドレス空間識別子

7.2.2 ページテーブルエントリ下位レジスタ（PTEL）

PTEL は LDTLB 命令により UTLB へ登録する物理ページ番号とページ管理情報を格納するために使用されます。本レジスタはソフトウェアの指示がない限り内容が変更されることはありません。

ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	PPN												
初期値：	0	0	0	—	—	—	—	—	—	—	—	—	—	—	—	—
R/W：	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	PPN						—	V	SZ1	PR1	PR0	SZ0	C	D	SH	WT
初期値：	—	—	—	—	—	—	0	—	—	—	—	—	—	—	—	—
R/W：	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ビット	ビット名	初期値	R/W	説　　明
31～29	−	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
28～10	PPN	−	R/W	物理ページ番号
9	−	0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
8	V	−	R/W	ページ管理情報 各ビットの意味は、共用 TLB（UTLB）の対応するビットと同じです。 詳細は「7.3　TLB の機能」を参照してください。
7	SZ1	−	R/W	
6	PR1	−	R/W	
5	PR0	−	R/W	
4	SZ0	−	R/W	
3	C	−	R/W	
2	D	−	R/W	
1	SH	−	R/W	
0	WT	−	R/W	

7.2.3 変換テーブルベースレジスタ（TTB）

TTB は、現在使用しているページテーブルのベースアドレスの格納用などの用途に使用します。TTB はソフトウェアの指示がない限り内容が変更されることはありません。本レジスタはソフトウェアで自由に使用可能です。

ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	TTB															
初期値：	－	－	－	－	－	－	－	－	－	－	－	－	－	－	－	－
R/W：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TTB															
初期値：	－	－	－	－	－	－	－	－	－	－	－	－	－	－	－	－
R/W：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

7.2.4 TLB 例外アドレスレジスタ（TEA）

TEA は、MMU 例外またはアドレスエラー例外発生後に、例外を発生させた仮想アドレスが格納されます。このレジスタはソフトウェアにより変更することは可能です。

ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	TEA								MMU例外／アドレスエラーを発生させた仮想アドレス							
初期値：	－	－	－	－	－	－	－	－	－	－	－	－	－	－	－	－
R/W：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TEA								MMU例外／アドレスエラーを発生させた仮想アドレス							
初期値：	－	－	－	－	－	－	－	－	－	－	－	－	－	－	－	－
R/W：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

7.2.5 MMU 制御レジスタ（MMUCR）

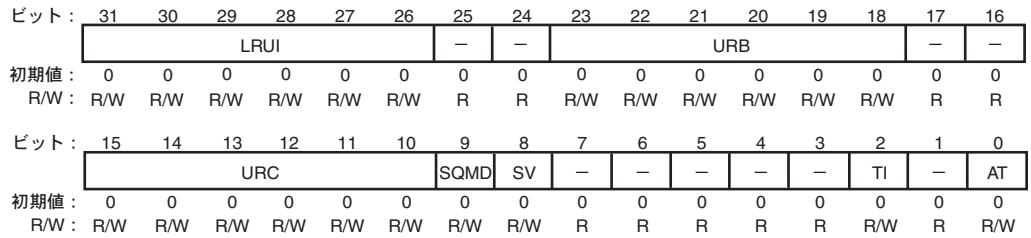
MMUCR の各ビットは以下に示すように MMU の設定を行います。このため MMUCR の書き換えは P1、P2 領域のプログラムで行うようにしてください。

MMUCR レジスタを更新後、P0、P3、U0、ストアキュー領域へのアクセス（命令フェッチを含む）を行う前に、以下の 1～3 のどれかを実行してください。

1. RTE命令による分岐を実行してください。この場合、分岐先はP0、P3、U0領域でかまいません。
2. 任意のアドレス（キャッシング不可領域でもよい）に対して、ICBI命令を実行してください。
3. MMUCR更新の前にあらかじめIRMCR.R2＝0（初期値）と設定されていた場合には、特定の命令シーケンスは不要です。しかしこの方法では、MMUCR更新命令の次命令を命令フェッチからやり直すため、CPUの処理性能が低下しますのでご注意ください。

ただし、方法 3 は今後の SuperH シリーズでは保証されない可能性があります。今後の SuperH シリーズでの互換性を保証するためには、1 または 2 を用いることを推奨します。

MMUCR はソフトウェアにより変更可能です。ただし LRUI ビットと URC ビットはハードウェアにより更新されることもあります。



ビット	ビット名	初期値	R/W	説　　明
31～26	LRUI	すべて 0	R/W	入れ換えを行う ITLB エントリを示す LRU ビット ITLB ミス発生時に入れ換える ITLB のエントリを決めるため、LRU 方式（Least Recently Used）を用います。LRUI ビットを用いて ITLB の追い出すエントリを確定できます。 LRUI は、以下のアルゴリズムで更新が行われます。 なお、以下の「x」は更新を行わないことを意味します。 000xxx：ITLB のエントリ 0 を用いたとき 1xx00x：ITLB のエントリ 1 を用いたとき x1x1x0：ITLB のエントリ 2 を用いたとき xx1x11：ITLB のエントリ 3 を用いたとき xxxxxx：上記以外 また LRUI が以下の状態のとき、対応する ITLB のエントリが ITLB ミスにより更新されます。なお、下表で設定禁止の値にはソフトウェアの責任で設定しないようにしてください。またパワーオンリセット、マニュアルリセット後に LRUI は 0 に初期化されるので、ハードウェアの更新によって LRUI が設定禁止の値になることはありません。 なお、以下の「x」は Don't care を意味します。 111xxx：ITLB のエントリ 0 が更新される 0xx11x：ITLB のエントリ 1 が更新される x0x0x1：ITLB のエントリ 2 が更新される xx0x00：ITLB のエントリ 3 が更新される 上記以外：設定禁止
25、24	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
23～18	URB	すべて 0	R/W	入れ換えを行う UTLB エントリの境界を示すビット URB≠0 のときに有効となります。
17、16	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。

ビット	ビット名	初期値	R/W	説　　明
15～10	URC	すべて 0	R/W	LDTLB 命令により入れ換えを行う UTLB エントリを示すためのランダムカウンタ UTLB へのアクセスが発生するたびにインクリメントされます。ただし URB＞0 の場合、URC＝URB の条件が成立すると URC は 0 にクリアされます。またソフトウェアにより URC＞URB となる値が URC に書き込まれた場合、最初は URC＝H'3F になるまで URB を超えてインクリメントされますので注意してください。なお URC は、LDTLB 命令によってカウントアップされません。
9	SQMD	0	R/W	ストアキューモードビット ストアキューへのアクセス権を指定します。 0：ユーザ／特権アクセスが可能 1：特権アクセスが可能（ユーザアクセスの場合はアドレスエラー例外）
8	SV	0	R/W	単一仮想記憶モード／多重仮想記憶モード切り替えビット このビットを変更するときは、必ず TI ビットにも 1 を書き込んでください。 0：多重仮想記憶モード 1：単一仮想記憶モード
7～3	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
2	TI	0	R/W	TLB 無効化ビット このビットに 1 を書き込むと、UTLB/ITLB の有効ビットをすべて 0 にクリアします。読み出すと常に 0 が読み出されます。
1	－	0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
0	AT	0	R/W	アドレス変換有効ビット MMU のイネーブル（有効）とディスエーブル（無効）を指定します。 0：MMU ディスエーブルにする 1：MMU イネーブルにする AT ビットが 0 の状態では MMU 例外は発生しません。このため MMU を使用しないソフトウェアでは AT ビットを 0 の状態で使用してください。

7.2.6 物理アドレス空間制御レジスタ（PASCR）

PASCR は物理アドレス空間の動作を制御します。

ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	—	—	—	—	UB							
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ビット	ビット名	初期値	R/W	説　　明
31～8	—	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
7～0	UB	すべて 0	R/W	エリア（64M バイト）ごとのバッファドライト制御 キャッシュを使わない書き込みのバスアクセスが完了するまで次の CPU からのバスアクセスを待たせるかをエリアごとに指定します。 0：CPU は書き込みのバスアクセスの完了を待たずに次のバスアクセスを行います 1：CPU は書き込みのバスアクセスの完了を待ってから次のバスアクセスを行います UB[7]：制御レジスタ領域に対応 UB[6]：エリア 6 に対応 UB[5]：エリア 5 に対応 UB[4]：エリア 4 に対応 UB[3]：エリア 3 に対応 UB[2]：エリア 2 に対応 UB[1]：エリア 1 に対応 UB[0]：エリア 0 に対応

7.2.7 命令再フェッチ抑止制御レジスタ（IRMCR）

IRMCR は特定のリソースが変更された場合に、次の命令を命令フェッチからやり直すかどうかを制御します。特定のリソースとは、制御レジスタの一部、TLB、キャッシュを示します。

初期状態ではリソース変更後、次の命令の命令フェッチをやり直すように設定されています。しかしこの状態では、リソースの変更を一回行うごとに命令フェッチのやり直しが起こり、CPU の処理性能が低下します。そのため IRMCR の各ビットを 1 に設定し、必要なリソースの変更をまとめて行ったうえで、特定の命令を実行し、変更後のリソースを使用するプログラムの実行へ移るようにすることを推奨します。

特定のシーケンスに関しては、各リソースの説明を参照してください。

ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	—	—	—	—	—	—	—	R2	R1	LT	MT	MC
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R/W	R/W	R/W	R/W	R/W

ビット	ビット名	初期値	R/W	説　　明
31～5	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
4	R2	0	R/W	レジスタ変更後再フェッチ抑止 2 MMUCR、PASCR、CCR、RAMCR、PTEH の各レジスタが変更された場合に、次命令の再フェッチを行うかどうかを制御します。 0：再フェッチを行います 1：再フェッチを行いません
3	R1	0	R/W	レジスタ変更後再フェッチ抑止 1 アドレス H'FF200000～H'FF2FFFFF に存在するレジスタが変更された場合に、次命令の再フェッチを行うかどうかを制御します。 0：再フェッチを行います 1：再フェッチを行いません
2	LT	0	R/W	LDTLB 実行後再フェッチ抑止 LDTLB 命令を実行後に、次命令の再フェッチを行うかどうかを制御します。 0：再フェッチを行います 1：再フェッチを行いません

ビット	ビット名	初期値	R/W	説　　明
1	MT	0	R/W	メモリ割り付け TLB ライト後再フェッチ抑止 MMUCR.AT＝1 の状態で、メモリ割り付け ITLB/UTLB ライトを行った後に、次命令の再フェッチを行うかどうかを制御します。 0：再フェッチを行います 1：再フェッチを行いません
0	MC	0	R/W	メモリ割り付け IC ライト後再フェッチ抑止 CCN.ICE＝1 の状態で、メモリ割り付け IC ライトを行った後に、次命令の再フェッチを行うかどうかを制御します。 0：再フェッチを行います 1：再フェッチを行いません

7.3 TLB の機能

7.3.1 共用 TLB（UTLB）の構成

UTLB は次の 2 つの目的のために使用されます。

1. データアクセスのとき、仮想アドレスを物理アドレスへ変換する。
2. 命令TLBミスのとき、ITLBへ登録するアドレス変換情報のテーブル。

このため共用 TLB と呼ばれます。UTLB には外部メモリ上に置かれるアドレス変換テーブルの情報がキャッシングされます。アドレス変換テーブルには仮想ページ番号とアドレス空間識別子、それに対応する物理ページ番号とページ管理情報が格納されています。**図 7.6** に UTLB の構成を示します。UTLB はフルアソシアティブ方式の 64 エントリで構成されています。**図 7.7** にページサイズとアドレスの関係を示します。

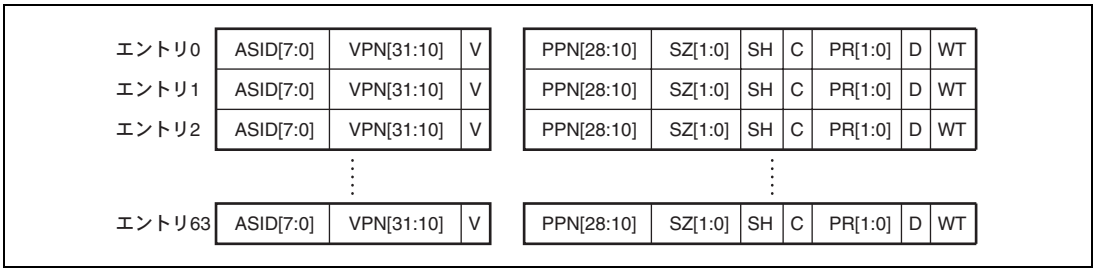


図 7.6 UTLB の構成

【記号説明】

VPN：仮想ページ番号　1K バイトページのとき、仮想アドレスの上位 22 ビット
4K バイトページのとき、仮想アドレスの上位 20 ビット
64K バイトページのとき、仮想アドレスの上位 16 ビット
1M バイトページのとき、仮想アドレスの上位 12 ビット

ASID：アドレス空間識別子　仮想ページをアクセスできるプロセスを示します。
単一仮想記憶モードかつユーザモードか、多重仮想記憶モードのときで、SH ビットが 0 ならアドレス比較の際に PTEH 中の ASID と比較されます。

SH：共有状態ビット　0：複数のプロセスでページを共有しません。
1：複数のプロセスでページを共有します。

SZ[1:0]：ページサイズビット　ページサイズを指定します。
00：1K バイトページ
01：4K バイトページ
10：64K バイトページ
11：1M バイトページ

V：有効ビット　エントリが有効かどうかを示します。
0：無効
1：有効
パワーオンリセット時に 0 にクリアされます。
マニュアルリセット時には変化しません。

PPN：物理ページ番号　物理アドレスの上位 22 ビット
1K バイトページのときは PPN[28:10]が有効です。
4K バイトページのときは PPN[28:12]が有効です。
64K バイトページのときは PPN[28:16]が有効です。
1M バイトページのときは PPN[28:20]が有効です。
また PPN の設定においてはシノニム問題に注意してください
（**「7.4.5　シノニム問題の回避」**参照）。

PR[1:0]：保護キーデータ　ページのアクセス権をコードで表した 2 ビットデータ
00：特権モードで読み出しのみ可能
01：特権モードで読み出し／書き込み可能
10：特権／ユーザモードで読み出しのみ可能
11：特権／ユーザモードで読み出し／書き込み可能

C：キャッシング可能ビット　ページがキャッシング可能かどうか示します。
0：キャッシング不可能。
1：キャッシング可能。
制御レジスタ空間のマッピングを行う場合、このビットは 0 にしてください。

D：ダーティビット　ページに書き込みが行われたかどうかを示します。
0：書き込みが行われていない。
1：書き込みが行われた。

WT：ライトスルービット　キャッシュへの書き込みモードを指定します。
0：コピーバックモード
1：ライトスルーモード

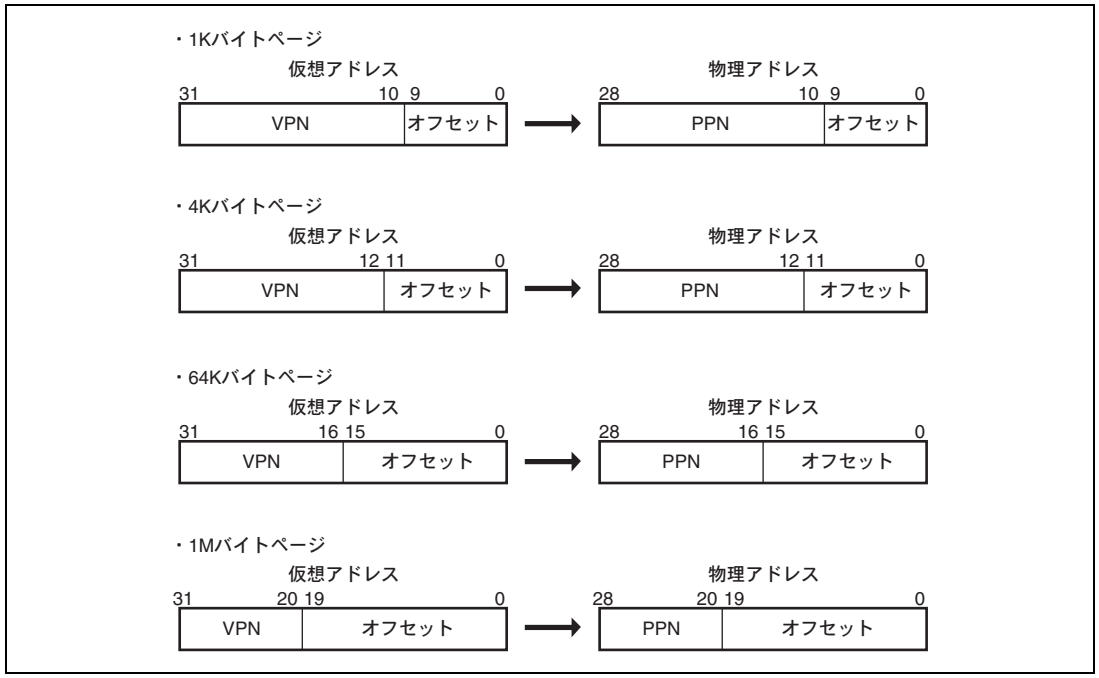


図 7.7　ページサイズとアドレスの関係

7.3.2　命令 TLB（ITLB）の構成

ITLB は命令アクセスのとき、仮想アドレスを物理アドレスへ変換するために用いられます。ITLB には UTLB 上に置かれるアドレス変換テーブルの情報がキャッシングされます。図 7.8 に ITLB の構成を示します。ITLB はフルアソシアティブの 4 エントリで構成されています。

エントリ0	ASID[7:0]	VPN[31:10]	V	PPN[28:10]	SZ[1:0]	SH	C	PR
エントリ1	ASID[7:0]	VPN[31:10]	V	PPN[28:10]	SZ[1:0]	SH	C	PR
エントリ2	ASID[7:0]	VPN[31:10]	V	PPN[28:10]	SZ[1:0]	SH	C	PR
エントリ3	ASID[7:0]	VPN[31:10]	V	PPN[28:10]	SZ[1:0]	SH	C	PR

【注】　1. D、WTビットをサポートしません。
2. PRビットが1ビットになり、UTLBのPRビットの上位1ビットに対応します。

図 7.8　ITLB の構成

7.3.3 アドレス変換方式

図 7.9 に、UTLB を用いたメモリアクセスのフローを示します。

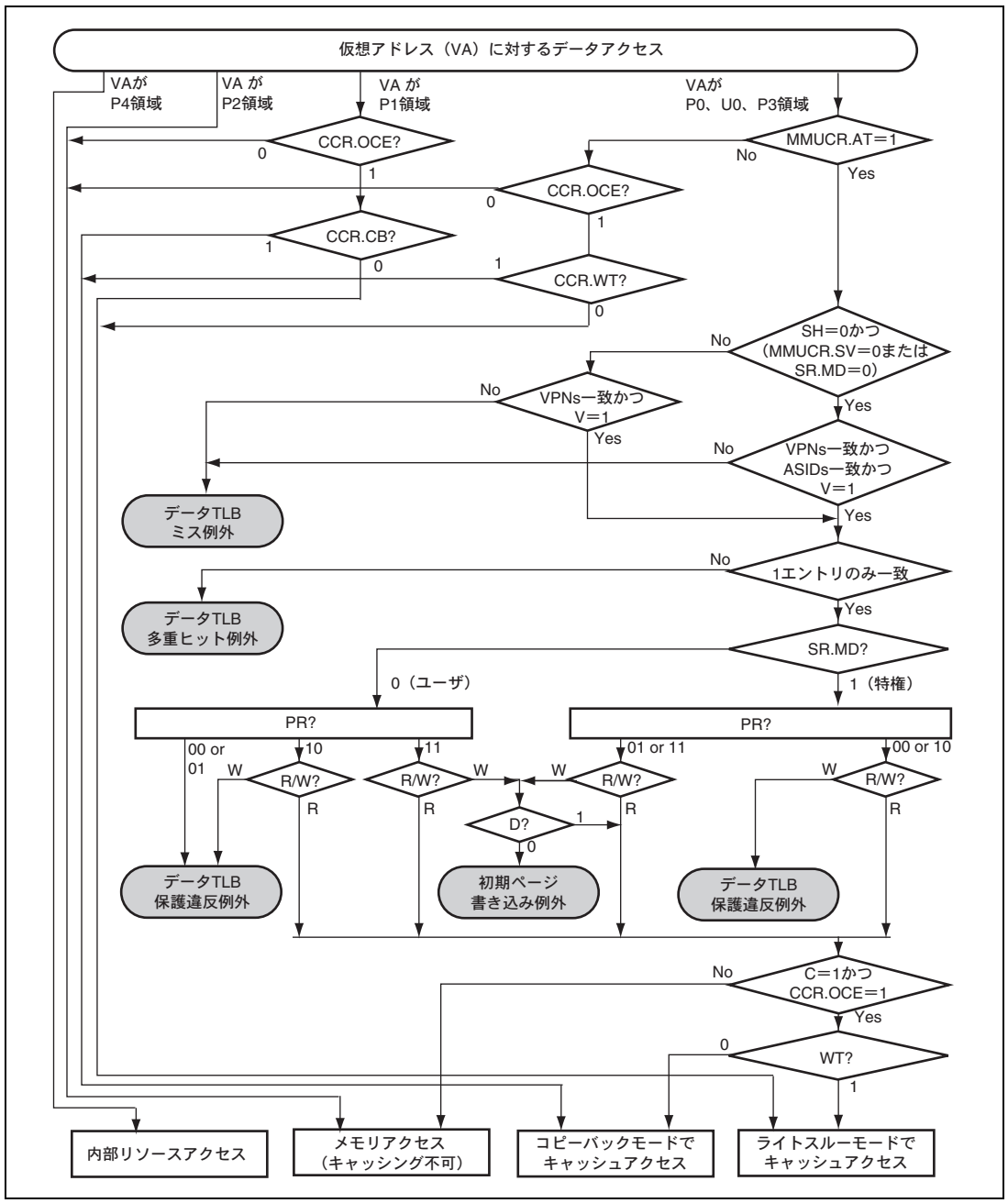


図 7.9　UTLB を用いたメモリアクセスフロー

図 7.10 に ITLB を用いたメモリアクセスのフローを示します。

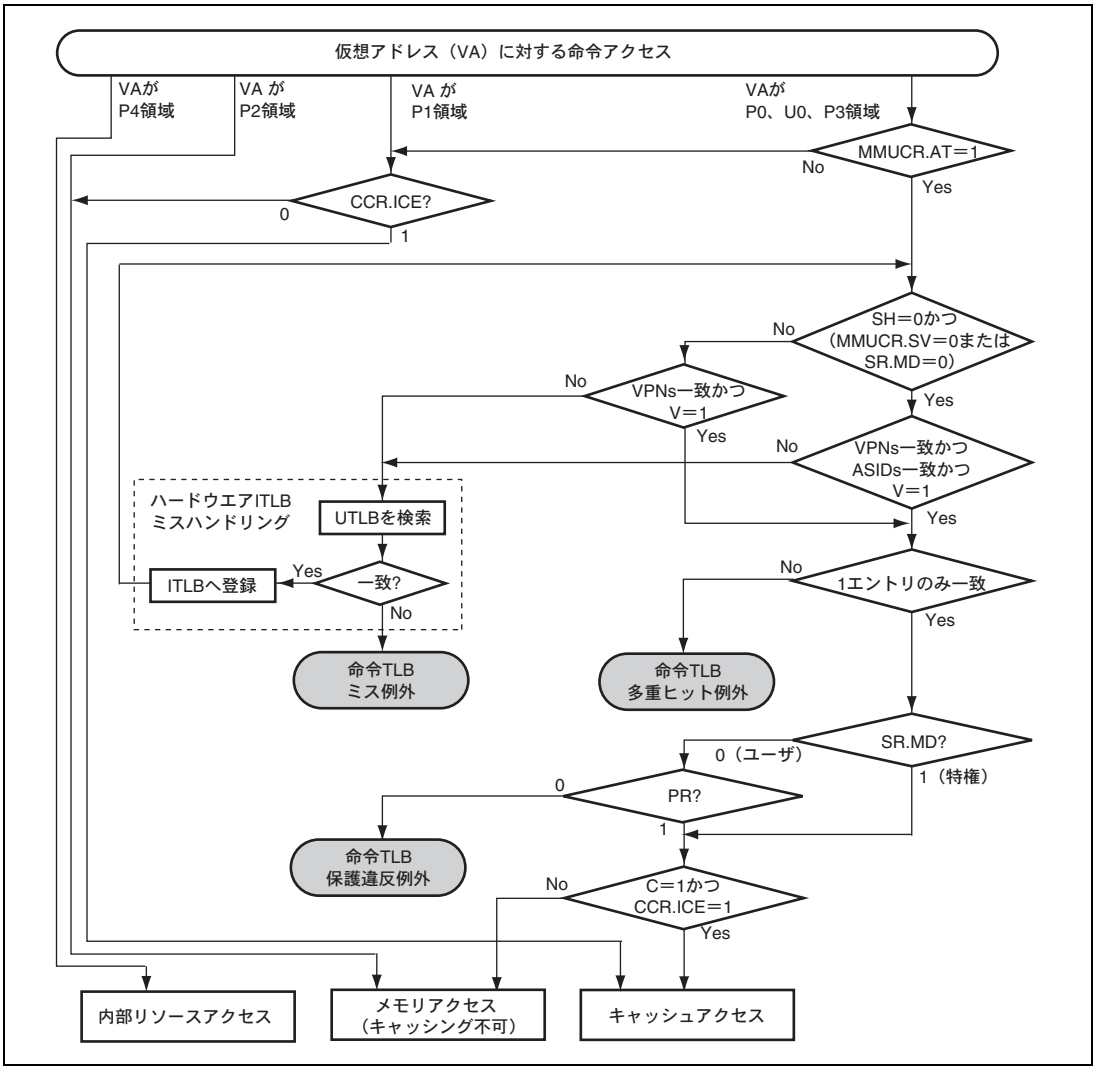


図 7.10　ITLB を用いたメモリアクセスフロー

7.4 MMU の機能

7.4.1 MMU のハードウェア管理

SH-4A がサポートする MMU の機能として次のものがあります。

1. ソフトウェアがアクセスする仮想アドレスをデコードし、MMUCRの設定に従ってUTLB、ITLBを制御してアドレス変換を行います。
2. アドレス変換の際に読み出されたページ管理情報をもとに、キャッシュへのアクセス状態を判定します（C、WTビット）。
3. データアクセス、命令アクセスにおいて正常にアドレス変換が行われなかった場合、MMU例外の発生によりソフトウェアに通知します。
4. 命令アクセスでITLBにアドレス変換情報が登録されていないとき、UTLBを検索します。必要なアドレス変換情報がUTLBに登録されていた場合、MMUCRのLRUIビットに従い、ITLBにそのアドレス変換情報をコピーします。

7.4.2 MMU のソフトウェア管理

MMU に対するソフトウェアの処理として次のものがあります。

1. MMU関連レジスタの設定。一部ハードウェアにより自動的に更新されるものもあります。
2. TLBエントリの登録、削除、読み出し。UTLBエントリの登録にはLDTLB命令を用いる方法と、メモリ割り付けUTLBに直接書き込む方法があります。ITLBエントリの登録はメモリ割り付けITLBに直接書き込む方法しかありません。UTLB、ITLBエントリの削除と読み出しは、メモリ割り付けUTLB、ITLBをアクセスすることで可能です。
3. MMU例外処理。MMU例外が発生したときにハードウェア側から設定された情報を元に処理を行います。

7.4.3 MMU の命令（LDTLB）

UTLB エントリを登録する命令として TLB ロード命令（LDTLB）があります。LDTLB 命令が発行されると、SH-4A は PTEH と PTEL の内容を URC ビットが指し示す UTLB エントリにコピーします。LDTLB 命令により ITLB エントリの更新は行われませんので、UTLB エントリから追い出されたアドレス変換情報が ITLB エントリに残る可能性があります。LDTLB 命令はアドレス変換情報を変更する命令のため、必ず P1、P2 領域のプログラムで発行するようにしてください。LDTLB 命令実行後、TLB が有効な領域へのアクセス（命令フェッチを含む）を行う前に、以下の 1.～3.のどれかを実行してください。

1. RTE命令による分岐を実行してください。この場合、分岐先はTLBが有効な領域で構いません。
2. 任意のアドレス（キャッシング不可領域でもよい）に対して、ICBI命令を実行してください。
3. LDTLB命令実行前にあらかじめIRMCR.LT=0（初期値）と設定されていた場合には、特定の命令シーケンスは不要です。しかしこの方法では、LDTLB命令の次命令を命令フェッチからやり直すため、CPUの処理性能が低下しますのでご注意ください。

ただし、方法 3.は今後の SuperH シリーズでは保証されない可能性があります。今後の SuperH シリーズでの互換性を保証するためには、1.または 2.を用いることを推奨します。

図 7.11 に LDTLB 命令の動作を示します。

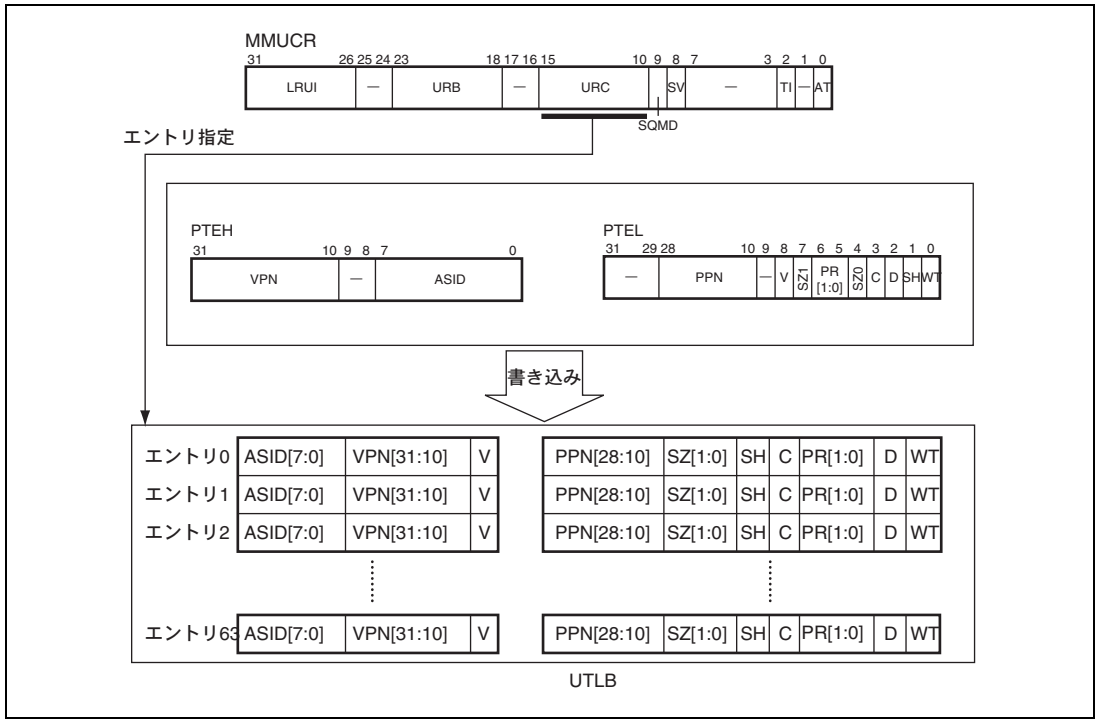


図 7.11　LDTLB 命令の動作

7.4.4 ハードウェア ITLB ミスハンドリング

SH-4A は命令アクセスの際、ITLB を検索して必要なアドレス変換情報を見つけられなかった（ITLB ミス）場合、ハードウェアにより UTLB を検索し、必要なアドレス変換情報があれば ITLB への登録を行います。これをハードウェア ITLB ミスハンドリングと呼びます。UTLB を検索しても必要なアドレス変換情報が見つからない場合、命令 TLB ミス例外を発生し、処理をソフトウェアへ移します。

7.4.5 シノニム問題の回避

TLB エントリに 1K、4K バイトページを登録するときにシノニム問題が発生する可能性があります。シノニム問題とは、複数の仮想アドレスが 1 つの物理アドレスにマッピングされる場合に、キャッシュの複数のエントリに同一の物理アドレスのデータが登録されてしまい、データの一致性を保証できなくなるという問題です。この問題は命令 TLB や命令キャッシュではデータの読み出ししか行わないため発生しません。SH-4A ではオペランドキャッシュの高速動作のために仮想アドレスの[12:5]を用いて、エントリの指定を行います。しかし 1K バイトページでは仮想アドレスの[12:10]が、4K バイトページでは仮想アドレスの[12]がアドレス変換の対象になります。このため変換後の物理アドレスの[12:10]と仮想アドレスの[12:10]が異なる可能性があります。

このため UTLB エントリへのアドレス変換情報の登録には以下の制限が生じます。

1. 複数の1Kバイトページの UTLBエントリが同一の物理アドレスに変換されるアドレス変換情報をUTLBに登録するとき、VPN[12:10]は必ず等しくなるようにしてください。
2. 複数の4Kバイトページの UTLBエントリが同一の物理アドレスに変換されるアドレス変換情報をUTLBに登録するとき、VPN[12]は必ず等しくなるようにしてください。
3. 1Kバイトページの UTLBエントリの物理アドレスを、異なるページサイズの UTLBエントリで使用しないでください。
4. 4Kバイトページの UTLBエントリの物理アドレスを、異なるページサイズの UTLBエントリで使用しないでください。

上記の制限はキャッシュを用いたアクセスを行う場合に限定されます。

【注】 **将来の SuperH RISC engine ファミリ拡張に備えて、複数のアドレス変換情報が同一の物理メモリを使用する場合、VPN[20:10]を等しくなるようにしてください。また異なるページサイズのアドレス変換情報で同一の物理アドレスを使用しないでください。**

7.5 MMU 例外

MMU 例外には、命令 TLB 多重ヒット例外、命令 TLB ミス例外、命令 TLB 保護違反例外、データ TLB 多重ヒット例外、データ TLB ミス例外、データ TLB 保護違反例外、初期ページ書き込み例外の 7 つの例外があります。各例外の発生条件については**図 7.9** と**図 7.10** を参照してください。

7.5.1 命令 TLB 多重ヒット例外

命令 TLB 多重ヒット例外は、命令アクセスした仮想アドレスに一致する ITLB エントリが複数存在した場合に発生します。ハードウェア ITLB ミスハンドリングにより UTLB を検索する際に UTLB で多重ヒットが発生した場合も、命令 TLB 多重ヒット例外となります。

命令 TLB 多重ヒット例外が発生するとリセットになり、キャッシュのコヒーレンシは保証しません。

- **ハードウェア処理**

  命令TLB多重ヒット例外のとき、ハードウェアは次の処理を行います。

1. 例外の発生した仮想アドレスをTEAに設定します。
2. 例外コードH'140をEXPEVTに設定します。
3. リセット処理ルーチン（H'A000 0000）に分岐します。

- **ソフトウェア処理（リセットルーチン）**

  リセット処理ルーチンで多重ヒットを発生させたITLBエントリを確認します。この例外はプログラムのデバッグ時に用いるためのもので、通常はこの例外を発生させないでください。

7.5.2 命令 TLB ミス例外

命令 TLB ミス例外は、ハードウェア ITLB ミスハンドリングにより UTLB エントリに命令アクセスした仮想アドレスに対応するアドレス変換情報が見つからなかったときに発生します。命令 TLB ミス例外のハードウェアで行われる処理と、ソフトウェアで行う処理は次のとおりです。これはデータ TLB ミス例外時の処理と同じです。

- **ハードウェア処理**

  命令TLBミス例外のとき、ハードウェアは次の処理を行います。

1. 例外が発生した仮想アドレスのVPNをPTEHに設定します。
2. 例外の発生した仮想アドレスをTEAに設定します。
3. 例外コードH'040を、EXPEVTに設定します。
4. 例外が発生した命令のアドレスを指すPCの値をSPCに設定します。もし例外が遅延スロットで発生した場合は、遅延分岐命令のアドレスを指すPCの値をSPCに設定します。
5. 例外が発生したときのSRの内容をSSRに設定します。そのときのR15をSGRに設定します。

6. SRのMDビットを1に設定し、特権モードに切り替えます。
7. SRのBLビットを1に設定し、これ以降の例外要求をマスクします。
8. SRのRBビットを1に設定します。
9. VBRの内容にオフセットH'0000 0400を加えたアドレスに分岐し、命令TLBミス例外処理ルーチンを開始します。

- **ソフトウェア処理（命令TLBミス例外処理ルーチン）**

  外部メモリのページテーブルを検索し、必要なページテーブルエントリを割り当てるのはソフトウェアの責任です。必要なページテーブルエントリを探して割り当てるために、ソフトウェアでは次のように処理してください。

1. 外部メモリのアドレス変換テーブルに記録されているページテーブルエントリのPPN、PR、SZ、C、D、SH、V、WTの各ビットの値を、PTELに書き込みます。
2. エントリ置き換えで置き換えられるエントリをソフトウェアで指定する場合、その値をMMUCRのURCに書き込みます。このときURCがURBを超えるような場合、LDTLB命令発行後に適切な値に変更してください。
3. LDTLB命令を実行させ、PTEH、PTELの内容をTLBに書き込みます。
4. 最後に、例外処理からの復帰命令（RTE）を実行させ、例外処理ルーチンを終わらせ、制御を通常の流れに戻してください。ただし、LDTLB命令の次の命令以降にRTE命令を発行してください。

7.5.3 命令 TLB 保護違反例外

命令 TLB 保護違反例外は、命令アクセスした仮想アドレスに一致するアドレス変換情報が ITLB エントリに存在するにもかかわらず、実際のアクセスタイプが PR ビットで指定されるアクセス権で許されていない場合に発生します。命令 TLB 保護違反例外のハードウェアで行われる処理と、ソフトウェアで行う処理は次のとおりです。

- **ハードウェア処理**

  命令TLB保護違反例外のとき、ハードウェアは次の処理を行います。

1. 例外が発生した仮想アドレスのVPNをPTEHに設定します。
2. 例外の発生した仮想アドレスをTEAに設定します。
3. 例外コードH'0A0をEXPEVTに設定します。
4. 例外が発生した命令のアドレスを指すPCの値をSPCに設定します。もし例外が遅延スロットで発生した場合は、遅延分岐命令のアドレスを指すPCの値をSPCに設定します。
5. 例外が発生したときのSRの内容をSSRに設定します。そのときのR15をSGRに設定します。
6. SRのMDビットを1に設定し、特権モードに切り替えます。
7. SRのBLビットを1に設定し、これ以降の例外要求をマスクします。

8. SRのRBビットを1に設定します。
9. VBRの内容にオフセットH'0000 0100を加えたアドレスに分岐し、命令TLB保護違反例外処理ルーチンを開始します。

- **ソフトウェア処理（命令TLB保護違反例外処理ルーチン）**

  命令TLB保護違反を解決し、例外処理からの復帰命令（RTE）を実行させ、例外処理ルーチンを終わらせ、制御を通常の流れに戻してください。ただしLDTLB命令の次の命令以降にRTE命令を発行してください。

7.5.4 データ TLB 多重ヒット例外

データ TLB 多重ヒット例外は、データアクセスした仮想アドレスに一致する UTLB エントリが複数存在した場合に発生します。

データ TLB 多重ヒット例外が発生するとリセットになり、キャッシュのコヒーレンシは保証しません。また例外発生以前の UTLB 内の PPN の内容は壊れることがあります。

- **ハードウェア処理**

  データTLB多重ヒット例外のとき、ハードウェアは次の処理を行います。

1. 例外の発生した仮想アドレスをTEAに設定します。
2. 例外コードH'140をEXPEVTに設定します。
3. リセット処理ルーチン（H'A000 0000）に分岐します。

- **ソフトウェア処理（リセットルーチン）**

  リセット処理ルーチンで多重ヒットを発生させたUTLBエントリを確認します。この例外はプログラムのデバッグ時に用いるためのもので、通常はこの例外を発生させないでください。

7.5.5 データ TLB ミス例外

データ TLB ミス例外は、データアクセスした仮想アドレスに対応するアドレス変換情報が UTLB 内に見つからなかったときに発生します。データ TLB ミス例外のハードウェアで行われる処理と、ソフトウェアで行う処理は次のとおりです。

- **ハードウェア処理**

  データTLBミス例外のとき、ハードウェアは次の処理を行います。

1. 例外が発生した仮想アドレスのVPNをPTEHに設定します。
2. 例外の発生した仮想アドレスをTEAに設定します。
3. 読み出しのとき例外コードH'040を、書き込みのとき例外コードH'060を、EXPEVTに設定します（OCBP、OCBWB：読み出し； OCBI、MOVCA.L：書き込み）。
4. 例外が発生した命令のアドレスを指すPCの値をSPCに設定します。もし例外が遅延スロットで発生した場合は、遅延分岐命令のアドレスを指すPCの値をSPCに設定します。
5. 例外が発生したときのSRの内容をSSRに設定します。そのときのR15をSGRに設定します。
6. SRのMDビットを1に設定し、特権モードに切り替えます。
7. SRのBLビットを1に設定し、これ以降の例外要求をマスクします。
8. SRのRBビットを1に設定します。
9. VBRの内容にオフセットH'0000 0400を加えたアドレスに分岐し、データTLBミス例外処理ルーチンを開始します。

- **ソフトウェア処理（データTLBミス例外処理ルーチン）**

  外部メモリのページテーブルを検索し、必要なページテーブルエントリを割り当てるのはソフトウェアの責任です。必要なページテーブルエントリを探して割り当てるために、ソフトウェアでは次のように処理してください。

1. 外部メモリのアドレス変換テーブルに記録されているページテーブルエントリのPPN、PR、SZ、C、D、SH、V、WTの各ビットの値を、PTELに書き込みます。
2. エントリ置き換えで置き換えられるエントリをソフトウェアで指定する場合、その値をMMUCRのURCに書き込みます。このときURCがURBを超えるような場合、LDTLB命令発行後に適切な値に変更してください。
3. LDTLB命令を実行させ、PTEH、PTELの内容をUTLBに書き込みます。
4. 最後に、例外処理からの復帰命令（RTE）を実行させ、例外処理ルーチンを終わらせ、制御を通常の流れに戻してください。ただし、LDTLB命令の次の命令以降にRTE命令を発行してください。

7.5.6 データ TLB 保護違反例外

データ TLB 保護違反例外は、データアクセスした仮想アドレスに一致するアドレス変換情報が UTLB エントリに存在するにもかかわらず、実際のアクセスタイプが PR ビットで指定されるアクセス権で許されていない場合に発生します。データ TLB 保護違反例外のハードウェアで行われる処理と、ソフトウェアで行う処理は次のとおりです。

- **ハードウェア処理**

  データTLB保護違反例外のとき、ハードウェアは次の処理を行います。

1. 例外が発生した仮想アドレスのVPNをPTEHに設定します。
2. 例外の発生した仮想アドレスをTEAに設定します。
3. 読み出しのとき例外コードH'0A0を、書き込みのとき例外コードH'0C0を、EXPEVTに設定します（OCBP、OCBWB：読み出し； OCBI、MOVCA.L：書き込み）。
4. 例外が発生した命令のアドレスを指すPCの値をSPCに設定します。もし例外が遅延スロットで発生した場合は、遅延分岐命令のアドレスを指すPCの値をSPCに設定します。
5. 例外が発生したときのSRの内容をSSRに設定します。そのときのR15をSGRに設定します。
6. SRのMDビットを1に設定し、特権モードに切り替えます。
7. SRのBLビットを1に設定し、これ以降の例外要求をマスクします。
8. SRのRBビットを1に設定します。
9. VBRの内容にオフセットH'0000 0100を加えたアドレスに分岐し、データTLB保護違反例外処理ルーチンを開始します。

- **ソフトウェア処理（データTLB保護違反例外処理ルーチン）**

  データTLB保護違反を解決し、例外処理からの復帰命令（RTE）を実行させ、例外処理ルーチンを終わらせ、制御を通常の流れに戻してください。ただしLDTLB命令の次の命令以降にRTE命令を発行してください。

7.5.7 初期ページ書き込み例外

初期ページ書き込み例外は、データアクセス（書き込み）した仮想アドレスに一致するアドレス変換情報がUTLBエントリに存在し、アクセス権も許されているにもかかわらず、Dビットが0であった場合に発生します。初期ページ書き込み例外のハードウェアで行われる処理と、ソフトウェアで行う処理は次のとおりです。

- **ハードウェア処理**

  初期ページ書き込み例外のとき、ハードウェアは次の処理を行います。

1. 例外が発生した仮想アドレスのVPNをPTEHに設定します。
2. 例外の発生した仮想アドレスをTEAに設定します。
3. 例外コードH'080をEXPEVTに設定します。
4. 例外が発生した命令のアドレスを指すPCの値をSPCに設定します。もし例外が遅延スロットで発生した場合は、遅延分岐命令のアドレスを指すPCの値をSPCに設定します。
5. 例外が発生したときのSRの内容をSSRに設定します。そのときのR15をSGRに設定します。
6. SRのMDビットを1に設定し、特権モードに切り替えます。
7. SRのBLビットを1に設定し、これ以降の例外要求をマスクします。
8. SRのRBビットを1に設定します。
9. VBRの内容にオフセットH'0000 0100を加えたアドレスに分岐し、初期ページ書き込み例外処理ルーチンを開始します。

- **ソフトウェア処理（初期ページ書き込み例外処理ルーチン）**

  ソフトウェアの責任で、次のように処理してください。

1. 外部メモリから必要なページテーブルエントリを探し出します。
2. 外部メモリのページテーブルエントリのDビットに1を書き込んでください。
3. 外部メモリに記憶されているページテーブルエントリのPPN、PR、SZ、C、D、WT、SH、Vのビットの値をPTELに書き込みます。
4. エントリ置き換えで置き換えられるエントリをソフトウェアで指定する場合、その値をMMUCRのURCに書き込みます。このときURCがURBを超えるような場合、LDTLB命令発行後に適切な値に変更してください。
5. LDTLB命令を実行させ、PTEH、PTELの内容をUTLBに書き込みます。
6. 最後に、例外処理からの復帰命令（RTE）を実行させ、例外処理ルーチンを終わらせ、制御を通常の流れに戻してください。ただし、LDTLB命令の次の命令以降にRTE命令を発行してください。

7.6 メモリ割り付け TLB の構成

ITLB および UTLB をソフトウェアで管理するために、特権モードのとき、P2 領域のプログラムから MOV 命令によって ITLB および UTLB の内容の読み出し、書き込みが可能です。別の領域のプログラムからアクセスする場合、動作の保証はありません。

メモリ割り付け TLB アクセス後、P2 領域以外へのアクセス（命令フェッチを含む）を行う前に、以下の 1～3 のどれかを実行してください。

1. RTE命令による分岐を実行してください。この場合、分岐先はP2領域以外でかまいません。
2. 任意のアドレス（キャッシング不可領域でもよい）に対して、ICBI命令を実行してください。
3. メモリ割り付けTLBアクセスの前にあらかじめIRMCR.MT＝0（初期値）と設定されていた場合には、特定の命令シーケンスは不要です。しかしこの方法では、MMUCR更新命令の次命令を命令フェッチからやり直すため、CPUの処理性能が低下しますのでご注意ください。

ただし、方法 3 は今後の SuperH シリーズでは保証されない可能性があります。今後の SuperH シリーズでの互換性を保証するためには、1 または 2 を用いることを推奨します。

ITLB および UTLB は仮想アドレス空間の P4 領域に割り付けられています。ITLB では VPN、V、ASID をアドレスアレイとして、PPN、V、SZ、PR、C、SH をデータアレイとしてアクセス可能です。

UTLB では VPN、D、V、ASID をアドレスアレイとして、PPN、V、SZ、PR、C、D、WT、SH をデータアレイとしてアクセス可能です。V と D はアドレスアレイ側からとデータアレイ側からの両方からアクセスできるようになっています。アクセスサイズはロングワードサイズのみ可能です。この領域に対して命令フェッチは行えません。リザーブビットに対しては、書き込み値として 0 を指定してください。読み出し値は保証しません。

7.6.1 ITLB アドレスアレイ

ITLB のアドレスアレイは P4 領域の H'F200 0000～H'F2FF FFFF に割り付けられています。アドレスアレイのアクセスには、32 ビットのアドレス部の指定（読み出し／書き込み時）と 32 ビットのデータ部の指定（書き込み時）が必要です。アドレス部はアクセスするエントリを選択するための情報を指定し、データ部にはアドレスアレイに書き込む VPN、V、ASID を指定します。

アドレス部は、[31:24]が ITLB アドレスアレイを示す H'F2 になっており、[9:8]でエントリを選択するようになっています。アドレス部[1:0]はロングワードアクセスのため 0 を指定してください。

データ部は、[31:10]が VPN を、[8]が V を、[7:0]が ASID を示します。

ITLB アドレスアレイに対しては以下の 2 種類の操作が可能です。

1. ITLBアドレスアレイ　読み出し
   アドレス部に設定されたエントリに対応するITLBエントリから、データ部へVPN、V、ASIDを読み出します。
2. ITLBアドレスアレイ　書き込み
   アドレス部に設定されたエントリに対応するITLBエントリに対して、データ部で指定されたVPN、V、ASIDを書き込みます。

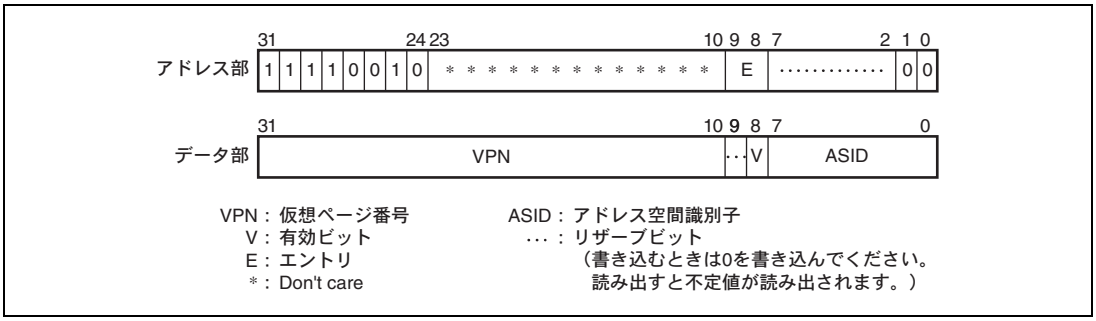


図 7.12　メモリ割り付け ITLB アドレスアレイ

7.6.2　ITLB データアレイ

ITLB のデータアレイは P4 領域の H'F300 0000～H'F37F FFFF に割り付けられています。データアレイのアクセスには、32 ビットのアドレス部の指定（読み出し／書き込み時）と 32 ビットのデータ部の指定（書き込み時）が必要です。アドレス部はアクセスするエントリを選択するための情報を指定し、データ部にはデータアレイ 1 に書き込む PPN、V、SZ、PR、C、SH を指定します。

アドレス部は、[31:23]が ITLB データアレイを示す H'F30 になっており、[9:8]でエントリを選択するようになっています。

データ部は、[28:10]が PPN を、[8]が V を、[7]、[4]が SZ を、[6]が PR を、[3]が C を、[1]が SH を示します。

ITLB データアレイに対しては以下の 2 種類の操作が可能です。

1. ITLBデータアレイ　読み出し

   アドレス部に設定されたエントリに対応するITLBエントリから、データ部へPPN、V、SZ、PR、C、SHを読み出します。

2. ITLBデータアレイ　書き込み

   アドレス部に設定されたエントリに対応するITLBエントリに対して、データ部で指定されたPPN、V、SZ、PR、C、SHを書き込みます。

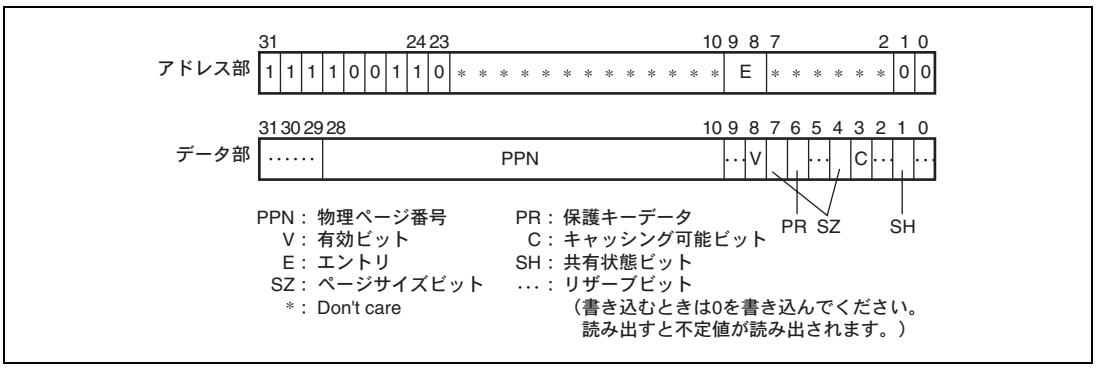


図 7.13　メモリ割り付け ITLB データアレイ

7.6.3 UTLB アドレスアレイ

UTLB のアドレスアレイは P4 領域の H'F600 0000～H'F60F FFFF に割り付けられています。アドレスアレイのアクセスには、32 ビットのアドレス部の指定（読み出し／書き込み時）と 32 ビットのデータ部の指定（書き込み時）が必要です。アドレス部はアクセスするエントリを選択するための情報を指定し、データ部にはアドレスアレイに書き込む VPN、D、V、ASID を指定します。

アドレス部は、[31:20]が UTLB アドレスアレイを示す H'F60 になっており、[13:8]でエントリを選択するようになっています。アドレス部[7]の連想ビット（A ビット）は、UTLB アドレスアレイへの書き込みのときのアドレス比較の有無を指定します。

データ部は、[31:10]が VPN を、[9]が D を、[8]が V を、[7:0]が ASID を示します。

UTLB アドレスアレイに対しては以下の 3 種類の操作が可能です。

1. UTLBアドレスアレイ　読み出し
   アドレス部に設定されたエントリに対応するUTLBエントリから、データ部へVPN、D、V、ASIDを読み出します。読み出す場合、アドレス部に指定される連想ビットは1でも0でも連想動作は行いません。
2. UTLBアドレスアレイ　書き込み（連想なし）
   アドレス部に設定されたエントリに対応するUTLBエントリに対して、データ部で指定されたVPN、D、V、ASIDを書き込みます。アドレス部のAビットは0にしてください。
3. UTLBアドレスアレイ　書き込み（連想あり）
   アドレス部のAビットが1で書き込みのとき、データ部で指定されたVPNとPTEH.ASIDを用い、UTLBの全エントリとの間で比較が行われます。比較は通常のアドレス比較の規則に従いますが、UTLBにミスした場合、例外は発生せずノーオペレーションとなります。比較によりデータ部で指定したVPNに対応するUTLBエントリが存在した場合、そのエントリに対してデータ部で指定したDとVを書き込みます。この連想動作はITLBに対しても同時に行われ、ITLB内に一致するエントリが存在した場合はそのエントリに対してVを書き込みます。UTLBでの比較でノーオペレーションとなってもITLBで一致していればITLB側にのみ書き込みは行います。またUTLBとITLBの両方で一致した場合、UTLBの情報がITLBへも書き込まれます。

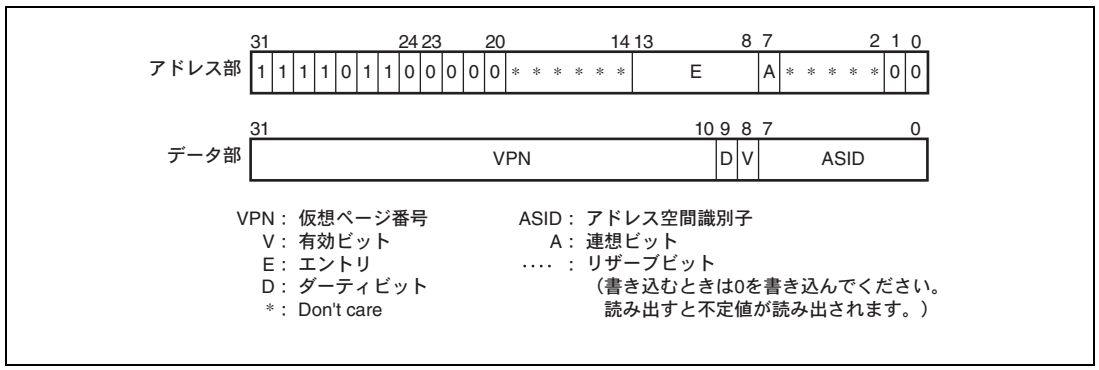


図 7.14　メモリ割り付け UTLB アドレスアレイ

7.6.4 UTLB データアレイ

UTLB のデータアレイは P4 領域の H'F700 0000～H'F70F FFFF に割り付けられています。データアレイのアクセスには、32 ビットのアドレス部の指定（読み出し／書き込み時）と 32 ビットのデータ部の指定（書き込み時）が必要です。アドレス部はアクセスするエントリを選択するための情報を指定し、データ部にはデータアレイに書き込む PPN、V、SZ、PR、C、D、SH、WT を指定します。

アドレス部は、[31:20]が UTLB データアレイを示す H'F70 になっており、[13:8]でエントリを選択するようになっています。

データ部は、[28:10]が PPN を、[8]が V を、[7]、[4]が SZ を、[6:5]が PR を、[3]が C を、[2]が D を、[1]が SH を、[0]が WT を示します。

UTLB データアレイに対しては以下の 2 種類の操作が可能です。

1. UTLBデータアレイ　読み出し
   アドレス部に設定されたエントリに対応するUTLBエントリから、データ部へPPN、V、SZ、PR、C、D、SH、WTを読み出します。
2. UTLBデータアレイ　書き込み
   アドレス部に設定されたエントリに対応するUTLBエントリに対して、データ部で指定されたPPN、V、SZ、PR、C、D、SH、WTを書き込みます。

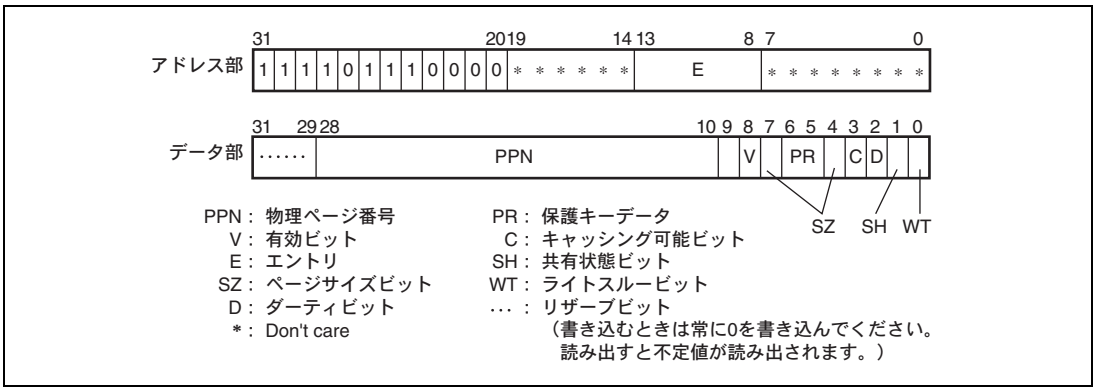


図 7.15　メモリ割り付け UTLB データアレイ

7.7 32 ビットアドレス拡張モード

SH-4A は PASCR レジスタの SE ビットを 1 に設定することで、29 ビットの物理アドレス空間を扱う 29 ビットアドレスモードから、32 ビットの物理アドレス空間を扱う 32 ビットアドレス拡張モードに変更することができます。

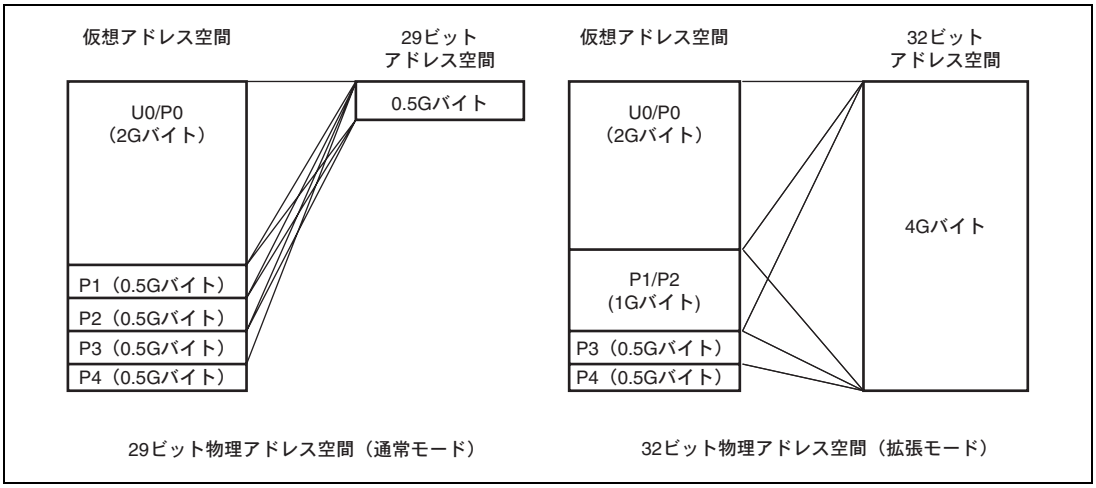


図 7.16　物理アドレス空間（32 ビットアドレス拡張モード）

7.7.1 32 ビットアドレス拡張モード概要

32 ビットアドレス拡張モードでは、29 ビットアドレスモードではアドレス変換対象外である P1/P2 領域の仮想アドレスを、32 ビットの物理アドレス空間にマッピングする特権空間マッピングバッファ（PMB）を導入します。また、既存の TLB（UTLB/ITLB）のアドレス変換対象領域についても、UTLB/ITLB の PPN フィールドの上位 3 ビットを拡張して、TLB 変換後のアドレスが 32 ビットの物理アドレスを扱えるようになります。

また、キャッシュの動作は、29 ビットアドレスモードでは固定的に P1 領域はキャッシング可能、P2 領域はキャッシング不可ですが、32 ビットアドレス拡張モードでは P1、P2 領域とも PMB の C ビットおよび WT ビットに従うようになります。

7.7.2 32 ビットアドレス拡張モードへの切り替え

SH-4A は、パワーオンリセット後は 29 ビットアドレスモードです。PASCR レジスタの SE ビットに 1 を書き込むことで、32 ビットアドレス拡張モードへと遷移します。32 ビットアドレス拡張モードでは MMU の動作は次の様になります。

1. MMUCR.AT＝0のとき、U0/P0/P3領域の仮想アドレスはそのまま32ビット物理アドレスとなります。P1/P2領域のアドレスはPMBマッピング情報に従いアドレス変換されます。
2. MMUCR.AT＝1のとき、U0/P0/P3領域の仮想アドレスはTLB変換情報に従い32ビット物理アドレスに変換されます。P1/P2領域のアドレスはPMBマッピングの情報に従いアドレス変換されます。

3. 制御レジスタ領域（H'FC00 0000～H'FFFF FFFF）は、MMUCR.ATにかかわらず、物理アドレスの[31:29]がB'111となります。制御レジスタ領域をUTLBに登録してアクセスする場合には、PPN[31:29]にはB'111を設定してください。

7.7.3 特権空間マッピングバッファ（PMB）構成

32 ビットアドレス拡張モードでは、P1/P2 領域の仮想アドレスは PMB マッピング情報に従いアドレス変換されます。PMB は 16 エントリで各エントリは以下の構成です。

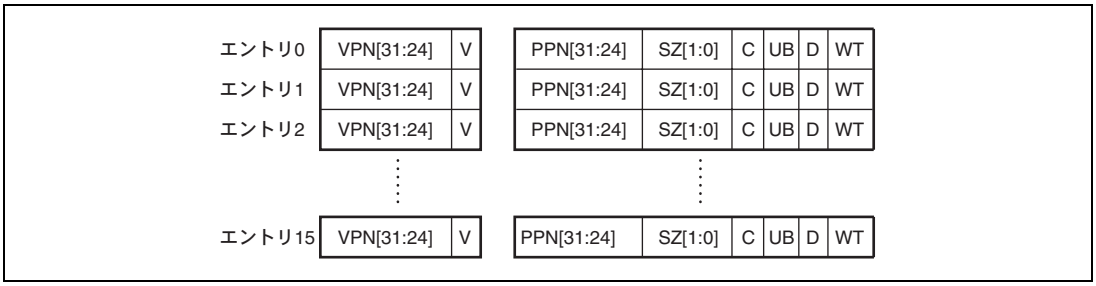


図 7.17　PMB の構成

【記号説明】

VPN：仮想ページ番号

16M バイトページのとき、仮想アドレスの上位 8 ビット

64M バイトページのとき、仮想アドレスの上位 6 ビット

128M バイトページのとき、仮想アドレスの上位 5 ビット

512M バイトページのとき、仮想アドレスの上位 3 ビット

SZ　：ページサイズビット

ページサイズを指定します。

00：16M バイトページ

01：64M バイトページ

10：128M バイトページ

11：512M バイトページ

V　：有効ビット

エントリが有効かどうかを示します。

0：無効

1：有効

パワーオンリセット時に 0 にクリアされます。

マニュアルリセット時に変化しません。

PPN：物理ページ番号

物理アドレスの上位 8 ビット

16M バイトページのとき、PPN[31:24]が有効

64M バイトページのとき、PPN[31:26]が有効

128M バイトページのとき、PPN[31:27]が有効

512M バイトページのとき、PPN[31:29]が有効

C　：キャッシング可能ビット

ページがキャッシング可能かどうかを示します。

0：キャッシング不可能

1：キャッシング可能

WT ：ライトスルービット

キャッシュへの書き込みモードを指定します。

0：コピーバックモード

1：ライトスルーモード

UB ：バッファドライトビット

バッファドライトするかどうかを指定します。

0：バッファドライト（ライト完了を待たずに後続命令のデータアクセスを開始する）

1：アンバッファドライト（ライト完了を待ち後続命令のデータアクセスを開始する）

7.7.4　PMB の機能

SH-4A がサポートする PMB の機能を以下に示します。

1. PMBへの書き込みはメモリ割り付けライトでのみ行えます。LDTLBでの登録はできません。
2. PMBマッピング対象であるP1/P2領域のアクセスするアドレスは必ずPMB登録されていることをソフトウェアで保証してください。PMBに変換情報がないP1/P2領域のアドレスにアクセスがあった場合、SH-4AはTLBリセットとなります。このとき、TEAにはTLBリセットの原因となったP1/P2領域へのアクセスアドレスが、EXPEVTにはコードH'140が格納されます。
3. SH-4AはPMBが多重ヒットを起こした場合の動作を保証しません。ソフトウェアは十分注意してPMBマッピング情報を登録してください。
4. PMBには連想ライトの機能はありません。
5. PMBにはPRフィールドは存在せず、リード／ライトのプロテクションを施すことはできません。PMBのアドレス変換対象はP1/P2アドレスなので、ユーザモードでのアクセスではアドレスエラー例外が発生します。
6. ITLBにはハードウェアITLBミスハンドリングによりUTLBとPMBの両方のエントリが混在して登録されます。ただしVPN[31:30]が10か否かで、UTLBから登録されたものかPMBから登録されたものか識別できます。PMBのエントリがITLBに登録される際に、PMBに存在しないフィールドであるASIDにはH'00、PRには01、SHには1が登録されます。

7.7.5 メモリ割り付け PMB の構成

PMB をソフトウェアで管理するために、特権モードのとき、P1/P2 領域のプログラムから MOV.L 命令によって PMB の内容の読み出し、書き込みが可能です。PMB のアドレスアレイは P4 領域の H'F610 0000～H'F61F FFFF に、PMB のデータアレイは P4 領域の H'F710 0000～H'F71F FFFF に割り付けられています。PMB では VPN、V をアドレスアレイとして、PPN、V、SZ、C、WT、UB をデータアレイとしてアクセス可能です。V はアドレスアレイ側からとデータアレイ側からの両方からアクセスできるようになっています。PMB メモリ割り付けアクセスを実行するプログラムは、PMB.C＝0 に設定したページの領域に配置してください。

1. PMBアドレスアレイリード
   アドレスとして[31:20]にPMBアドレスアレイを示すH'F61、[11:8]にエントリを指定してメモリ読み出しを行うと、[31:24]にVPN、[8]にVが読み出されます。
2. PMBアドレスアレイライト
   アドレスとして[31:20]にPMBアドレスアレイを示すH'F61、[11:8]にエントリを指定し、データとして[31:24]にVPN、[8]にVを指定してメモリ書き込みを行うと、指定したエントリに書き込まれます。
3. PMBデータアレイリード
   アドレスとして[31:20]にPMBデータアレイを示すH'F71、[11:8]にエントリを指定してメモリ読み出しを行うと、[31:24]にPPN、[9]にUB、[8]にV、[7][4]にSZ、[3]にC、[0]にWTが読み出されます。
4. PMBデータアレイライト
   アドレスとして[31:20]にPMBデータアレイを示すH'F71、[11:8]にエントリを指定し、データとして[31:24]にPPN、[9]にUB、[8]にV、[7][4]にSZ、[3]にC、[0]にWTを指定してメモリ書き込みを行うと、指定したエントリに書き込まれます。

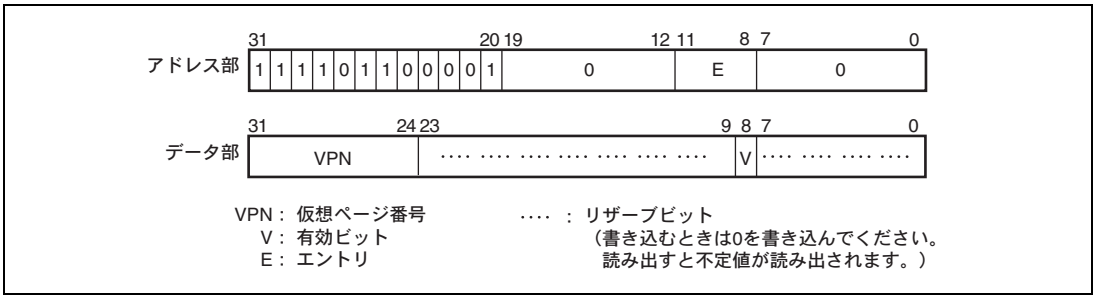


図 7.18 メモリ割り付け PMB アドレスアレイ

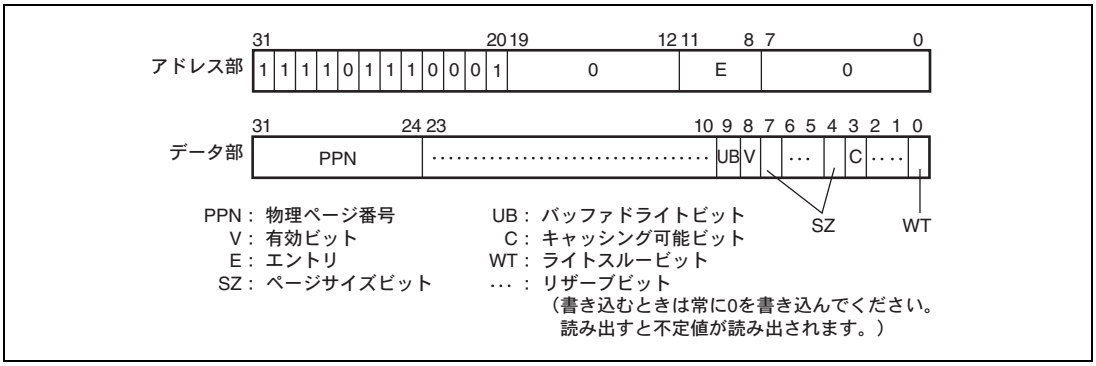


図 7.19　メモリ割り付け PMB データアレイ

7.7.6　32 ビットアドレス拡張モード使用時の注意事項

32 ビットアドレス拡張モードを使用する場合、本章ですでに述べた事項が以下のように拡張または変更されますので、注意してください。

（1） PASCR.SE

制御レジスタ PASCR[31]に SE ビットが追加になります。また、UB[6:0]は無効になります（UB[7]は 32 ビットアドレス拡張モードでも有効です）。

バッファドライトになるか否かは、P1/P2 領域に対するライトでは PMB の UB ビットにより制御されます。P0/P3/U0 領域に対するライトでは、MMU がイネーブルの場合 TLB の UB ビットにより制御され、MMU がディスエーブルの場合、常にバッファドライトになります。

ビット	ビット名	初期値	R/W	説　　明
31	SE	0	R/W	アドレスモード 0：29 ビットアドレスモード 1：32 ビットアドレス拡張モード
30～8	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
7～0	UB	すべて 0	R/W	エリア（64M バイト）ごとのバッファドライト制御 キャッシュを使わない書き込み時に CPU が書き込みの完了を待つかどうかをエリアごとに指定します。 0：CPU は書き込みの完了を待ちません 1：CPU は書き込みが完了するまでストールして待ちます。 UB [7]：制御レジスタ領域のバッファドライト制御 UB [6:0]：エリア（64M バイト）ごとのバッファドライト制御（32 ビットアドレス拡張モードでは無効）

（2） ITLB

ITLB の PPN フィールドが[31:10]へ拡張されます。

（3） UTLB

UTLB の各エントリに PMB の UB ビットと同じ意味の UB ビットが追加になります。

UB ：バッファドライトビット

バッファドライトするかどうかを指定します。

0：バッファドライト（ライト完了を待たずに後続処理を開始する）

1：アンバッファドライト（ライト完了を待ち後続処理を開始する）

UB ビットはメモリ割り付け TLB アクセスではデータアレイのビット[9]でリード／ライトが行えます。

（4） PTEL

UTLB と同様に PTEL レジスタのビット[9]に PMB の UB ビットと同じ意味の UB ビットが追加になります。

この UB ビットは LDTLB 命令によって UTLB の UB ビットへ書き込まれます。また PPN フィールドが[31:10]に拡張されます。

（5） CCR.CB

CCR レジスタの CB ビットは無効になります。P1 領域に対するキャッシャブルライトがコピーバックモードになるか、ライトスルーモードになるかは、PMB の WT ビットに従います。

（6） IRMCR.MT

IRMCR の MT ビットが、メモリ割り付け PMB ライトに対しても有効になります。

（7） QACR0、QACR1

QACR0、QACR1 レジスタの AREA0[4:2]、AREA[4:2]がそれぞれ AREA0[7:2]、AREA1[7:2]に拡張され、物理アドレス 31～26 に対応します。

（8） LSA0、LSA1、LDA0、LDA1

L0SADR、L1SADR、L0DADR、L1DADR がそれぞれ[31:0]に拡張されます。

また、32 ビットアドレスモード使用時にはソフトウェアは以下の点に注意してください。

1. SEビットの切り替えはパワーオンリセットまたはマニュアルリセット後のブートルーチンで0から1への切り替えのみサポートされています。
2. SEビット切り替え後、そのプログラムの配置されている領域自体がPMBアドレス変換対象となりますので、SEビットの切り替えに先立ってPMBへの登録が必要です。例外ハンドラなど、P1/P2領域へのアクセスされる可能性のあるアドレスについても必ずPMBへの登録を行ってください。
3. SEビットを切り替えるMOV.L命令の前にあるオペランドメモリアクセスが外部メモリアクセスを起こす場合、両アドレスモードでアクセスされる外部メモリ空間アドレスが異ならないようにしてください。

4. PMBの登録時にVビットがアドレスアレイとデータアレイの両方にマッピングされていることに注意してください。すなわち、1回目の一方への書き込みではV＝0を、2回目の他方への書き込みではV＝1を選んでください。

8. キャッシュ

SH-4A は命令用に 32K バイトの命令キャッシュ（IC）を、データ用に 32K バイトのオペランドキャッシュ（OC）を内蔵しています。

【注】 **命令キャッシュ、オペランドキャッシュの容量については、製品のハードウェアマニュアルを参照してください。本マニュアルではおのおの 32K バイトのケースについて説明します。**

8.1 特長

キャッシュの特長を**表 8.1** に示します。

SH-4A では、外部メモリへの高速な書き込みを行うために 32 バイト×2 のストアキュー（SQ）をサポートします。SQ の特長を**表 8.2** に示します。

表 8.1 キャッシュの特長

項目	命令キャッシュ	オペランドキャッシュ
容量	32K バイトキャッシュ	32K バイトキャッシュ
方式	4 ウェイセットアソシアティブ、 仮想アドレスインデックス／物理アドレスタグ	4 ウェイセットアソシアティブ、 仮想アドレスインデックス／物理アドレスタグ
ラインサイズ	32 バイト	32 バイト
エントリ数	256 エントリ／ウェイ	256 エントリ／ウェイ
書き込み方式	−	コピーバック／ライトスルー選択可能
置換方式	LRU（Least Recently Used）アルゴリズム	LRU（Least Recently Used）アルゴリズム

表 8.2 ストアキューの特長

項目	ストアキュー
容量	2×32 バイト
アドレス	H'E000 0000～H'E3FF FFFF
ライト	ストア命令（1 サイクルライト）
ライトバック	プリフェッチ命令（PREF 命令）
アクセス権	MMU ディスエーブル時：MMU 制御レジスタ（MMUCR）の SQMD ビットによる MMU イネーブル時：個々のページ PR による

SH-4A のオペランドキャッシュは 4 ウェイセットアソシアティブ方式で、おのおののウェイは 256 本のキャッシュラインから構成されます。**図 8.1** にオペランドキャッシュの構成を示します。

命令キャッシュは 4 ウェイセットアソシアティブ方式で、おのおののウェイは 256 本のキャッシュラインから構成されます。**図 8.2** に命令キャッシュの構成を示します。

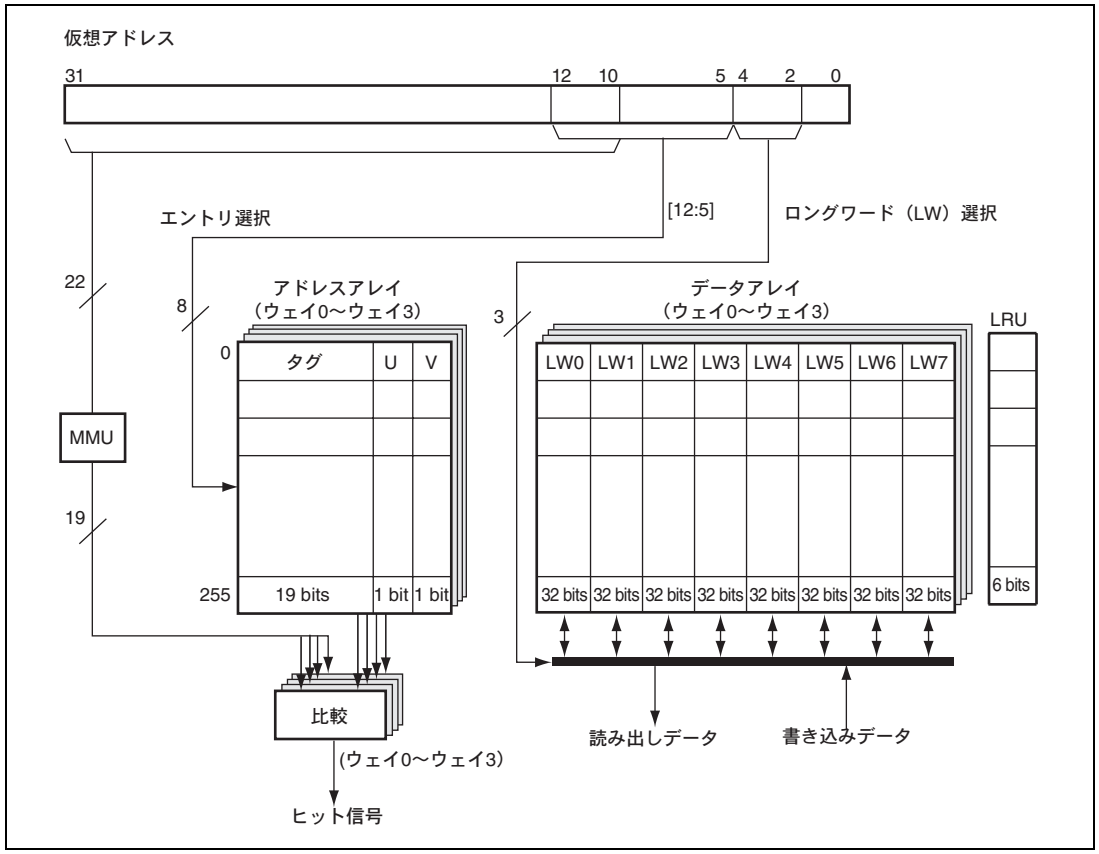


図 8.1　オペランドキャッシュの構成

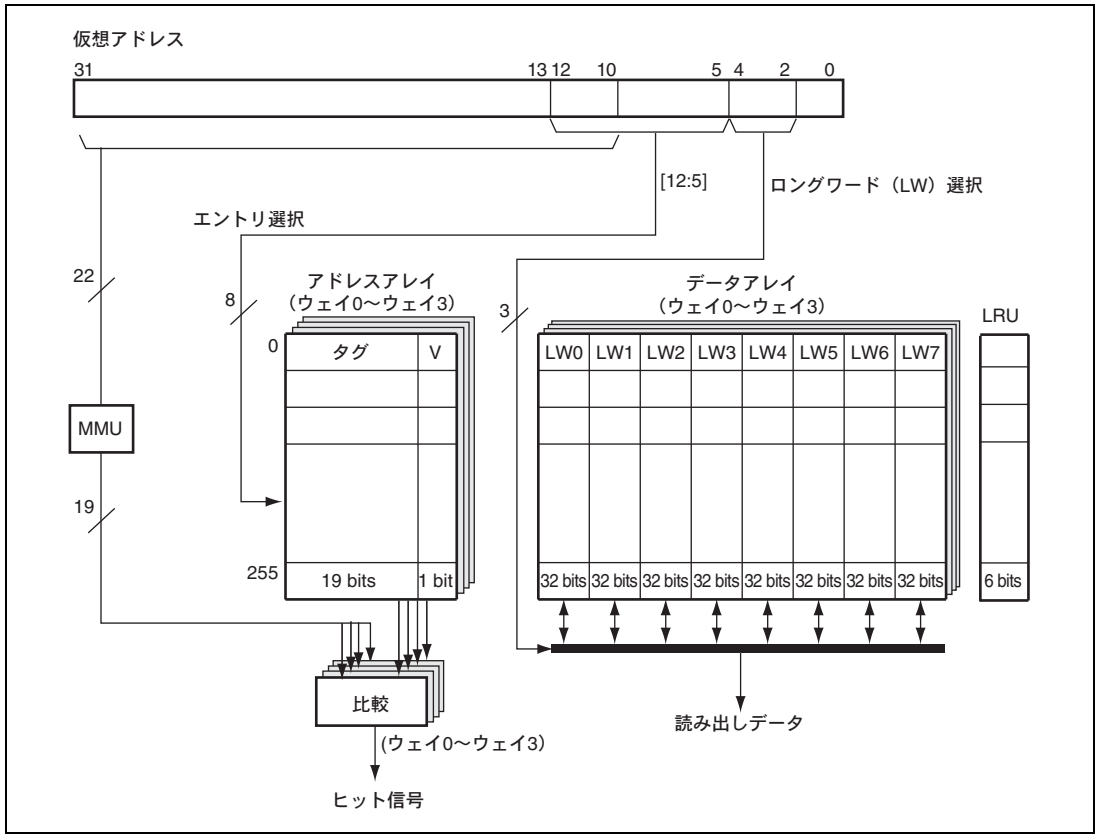


図 8.2　命令キャッシュの構成

(1) タグ

キャッシュされるデータラインの物理アドレス29ビットの上位19ビットを格納します。タグはパワーオンリセット、マニュアルリセットで初期化されません。

(2) V ビット（有効ビット）

キャッシュラインに有効なデータが格納されているか否かを示します。このビットが1のとき、そのキャッシュラインのデータは有効となります。Vビットはパワーオンリセットで0に初期化されますが、マニュアルリセットでは値を保持します。

(3) U ビット（ダーティビット）

コピーバックモードでキャッシュを使用中に、キャッシュラインへデータを書き込んだとき、Uビットが1になります。つまりUビットはキャッシュライン中のデータと外部メモリ中のデータとの不一致を示します。メモリ割り付けキャッシュ（**「8.6　メモリ割り付けキャッシュの構成」**参照）をアクセスすることによりUビットを書き換えない限り、ライトスルーモードでキャッシュを使用中はUビットが1になることはありません。Uビットはパワーオンリセットで0に初期化されますが、マニュアルリセットでは値を保持します。

(4) データ部

データ部には1キャッシュラインあたり32バイト（256ビット）のデータが格納されます。データアレイはパワーオンリセット、マニュアルリセットで初期化されません。

(5) LRU 部

4ウェイセットアソシアティブ方式では、エントリアドレスが同じデータを4つまでキャッシュに登録できます。エントリを登録するとき、4つのウェイのうち、どのウェイに登録するかをLRUビットが示します。LRUビットは各エントリ6ビットからなり、ハードウェアで制御します。ウェイ選択のアルゴリズムとして、最も以前にアクセスされたウェイを選ぶLRU（Least Recently Used）アルゴリズムを使用しています。LRUビットは、パワーオンリセットで0に初期化されますが、マニュアルリセットでは初期化されません。LRUビットは、ソフトウェアでは読み書きできません。

8.2 レジスタの説明

キャッシュに関連するレジスタを以下に示します。

表 8.3 レジスタ構成

名称	略称	R/W	P4 領域 アドレス*	エリア 7 アドレス*	サイズ
キャッシュ制御レジスタ	CCR	R/W	H'FF00 001C	H'1F00 001C	32
キューアドレス制御レジスタ 0	QACR0	R/W	H'FF00 0038	H'1F00 0038	32
キューアドレス制御レジスタ 1	QACR1	R/W	H'FF00 003C	H'1F00 003C	32
内蔵メモリ制御レジスタ	RAMCR	R/W	H'FF00 0074	H'1F00 0074	32

【注】 * P4 領域アドレスは、仮想アドレス空間の P4 領域を用いた場合のものです。エリア 7 アドレスは、TLB を用いて物理アドレス空間のエリア 7 からアクセスするものです。

表 8.4 各処理モードにおけるレジスタの状態

名称	略称	パワーオン リセット	マニュアル リセット	スリープ	スタンバイ
キャッシュ制御レジスタ	CCR	H'0000 0000	H'0000 0000	保持	保持
キューアドレス制御レジスタ 0	QACR0	不定	不定	保持	保持
キューアドレス制御レジスタ 1	QACR1	不定	不定	保持	保持
内蔵メモリ制御レジスタ	RAMCR	H'0000 0000	H'0000 0000	保持	保持

8.2.1 キャッシュ制御レジスタ（CCR）

CCR は、キャッシュの動作モードの選択、キャッシュの全エントリの無効化、キャッシュへの書き込みモードの選択を行います。

CCR の書き換えは、キャッシング不可の P2 領域のプログラムのみで行わなければなりません。CCR 更新後、キャッシング可能領域へのアクセス（命令フェッチを含む）を行う前に、以下の 1～3 のどれかを実行してください。

1. RTE命令による分岐を実行してください。この場合、分岐先はキャッシング可能領域でかまいません。
2. 任意のアドレス（キャッシング不可領域でもよい）に対して、ICBI命令を実行してください。
3. CCR更新の前にあらかじめIRMCR.R2＝0（初期値）と設定されていた場合には、特定の命令シーケンスは不要です。しかしこの方法では、CCR更新命令の次命令を命令フェッチからやり直すため、CPUの処理性能が低下しますのでご注意ください。

ただし、方法 3 は今後の SuperH シリーズでは保証されない可能性があります。今後の SuperH シリーズでの互換性を保証するためには、1 または 2 を用いることを推奨します。

ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	－	－	－	－	－	－	－	－	－	－	－	－	－	－	－	－
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

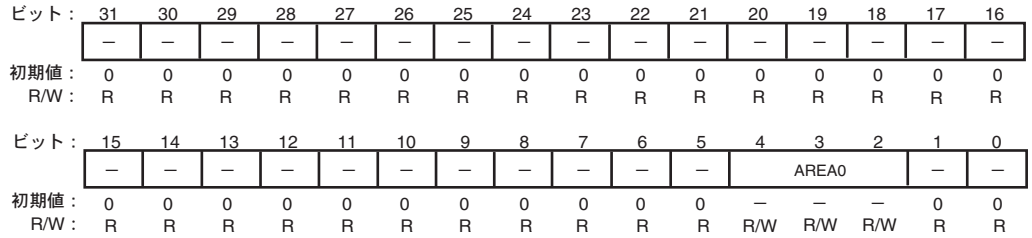
ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	－	－	－	－	ICI	－	－	ICE	－	－	－	－	OCI	CB	WT	OCE
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R/W	R	R	R/W	R	R	R	R	R/W	R/W	R/W	R/W

ビット	ビット名	初期値	R/W	説　　明
31～12	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
11	ICI	0	R/W	IC 無効化ビット このビットに 1 を書き込むと IC の全エントリの V ビットを 0 にします。読み出すと常に 0 が読み出されます。
10、9	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
8	ICE	0	R/W	IC 有効ビット IC の使用を選択します。ただしアドレス変換が行われる場合は、ページ管理情報の C ビットも 1 でなければ IC を使用できません。 0：IC を使用しない 1：IC を使用する
7～4	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。

ビット	ビット名	初期値	R/W	説　　明
3	OCI	0	R/W	OC 無効化ビット このビットに 1 を書き込むと OC の全エントリの V、U ビットを 0 にします。読み出すと常に 0 が読み出されます。
2	CB	0	R/W	コピーバックビット P1 領域のキャッシュへの書き込みモードを示します。 0：ライトスルーモード 1：コピーバックモード
1	WT	0	R/W	ライトスルーモード P0、U0、P3 領域のキャッシュへの書き込みモードを示します。ただし、アドレス変換が行われる場合は、ページ管理情報の WT ビットの値を優先します。 0：コピーバックモード 1：ライトスルーモード
0	OCE	0	R/W	OC 有効ビット OC の使用を選択します。ただしアドレス変換が行われる場合は、ページ管理情報の C ビットも 1 でなければ OC を使用できません。 0：OC を使用しない 1：OC を使用する

8.2.2　キューアドレス制御レジスタ 0（QACR0）

QACR0 は、MMU がディスエーブルのとき、ストアキュー0（SQ0）がマップされているエリアを設定します。



ビット	ビット名	初期値	R/W	説　　明
31～5	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
4～2	AREA0	不定	R/W	MMU がディスエーブルのとき、SQ0 に対する物理アドレス 28～26 を生成します。
1、0	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。

8.2.3 キューアドレス制御レジスタ 1（QACR1）

QACR1 は、MMU がディスエーブルのとき、ストアキュー1（SQ1）がマップされているエリアを設定します。

ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	−	−	−	−	−	−	−	−	−	−	−	AREA1			−	−
初期値：	0	0	0	0	0	0	0	0	0	0	0	−	−	−	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R/W	R/W	R/W	R	R

ビット	ビット名	初期値	R/W	説　　明
31〜5	−	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
4〜2	AREA1	不定	R/W	MMU がディスエーブルのとき、SQ1 に対する物理アドレス 28〜26 を生成します。
1、0	−	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。

8.2.4 内蔵メモリ制御レジスタ（RAMCR）

RAMCR は IC および OC のウェイ数の制御を行います。

RAMCR への書き換えは、キャッシング不可の P2 領域のプログラムで行われなければなりません。RAMCR 更新後、キャッシング可能領域または L メモリ領域へのアクセス（命令フェッチを含む）を行う前に、以下の 1〜3 のどれかを実行してください。

1. RTE命令による分岐を実行してください。この場合、分岐先はキャッシング不可領域またはLメモリ領域でかまいません。
2. 任意のアドレス（キャッシング不可領域でもよい）に対して、ICBI命令を実行してください。
3. RAMCR更新の前にあらかじめIRMCR.R2＝0（初期値）と設定されていた場合には、特定の命令シーケンスは不要です。しかしこの方法では、RAMCR更新命令の次命令を命令フェッチからやり直すため、CPUの処理性能が低下しますのでご注意ください。

ただし、方法 3 は今後の SuperH シリーズでは保証されない可能性があります。今後の SuperH シリーズでの互換性を保証するためには、1 または 2 を用いることを推奨します。

ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	−	−	−	−	−	−	RMD	RP	IC2W	OC2W	−	−	−	−	−	−
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R/W	R/W	R/W	R/W	R	R	R	R	R	R

ビット	ビット名	初期値	R/W	説　　明
31〜10	−	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
9	RMD	0	R/W	内蔵メモリアクセスモードビット 詳細は「9.4　L メモリの保護機能」を参照してください。
8	RP	0	R/W	内蔵メモリ保護有効ビット 詳細は「9.4　L メモリの保護機能」を参照してください。
7	IC2W	0	R/W	IC 2 ウェイモードビット 0：IC は 4 ウェイ動作 1：IC は 2 ウェイ動作 詳細は「8.4.3　IC 2 ウェイモード」を参照してください。
6	OC2W	0	R/W	OC 2 ウェイモードビット 0：OC は 4 ウェイ動作 1：OC は 2 ウェイ動作 詳細は「8.3.6　OC 2 ウェイモード」を参照してください。
5〜0	−	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。

8.3 オペランドキャッシュの動作説明

8.3.1 読み出し動作

オペランドキャッシュ（OC）が有効（CCR.OCE=1）かつキャッシング可能な領域からデータを読み出す場合、OC は以下のように動作します。

1. 仮想アドレスのビット[12:5]でインデックスされる各ウェイのキャッシュラインから、タグ、Vビット、UビットおよびLRUビットを読み出します。
2. 仮想アドレスをMMUにより変換した物理アドレスのビット[28:10]と、各ウェイから読み出したタグを比較し、

- タグが一致かつVビットが1のウェイが存在する場合 → 3.
- タグが一致かつVビットが1のウェイが存在せず、LRUビットにより選択された置換対象ウェイのUビットが0の場合 → 4.
- タグが一致かつVビットが1のウェイが存在せず、LRUビットにより選択された置換対象ウェイのUビットが1の場合 → 5.

3. キャッシュヒット

   ヒットしたウェイのデータ部から、仮想アドレスのビット[4:0]でインデックスされるデータをアクセスサイズに応じて読み出します。またヒットしたウェイが最新となるようにLRUビットを更新します。

4. キャッシュミス（書き戻しなし）

   仮想アドレスに対応する物理アドレス空間から、置換対象ウェイのキャッシュラインへデータを読み込みます。データの読み込みはキャッシュミスしたデータを含むクワッドワード（8バイト）から順にラップアラウンド方式で行い、該当するデータがキャッシュへ到着した時点で、CPUへ読み出しデータを返します。残りのキャッシュ1ライン分のデータが読み込まれている間、CPUは次の処理を実行することができます。キャッシュに1ライン分のデータの読み込みが完了した時点で、物理アドレスによるタグを登録し、Vビットに1を、Uビットに0を書き込みます。また置換したウェイが最新となるようにLRUビットを更新します。

5. キャッシュミス（書き戻しあり）

   置換対象ウェイのキャッシュラインのタグとデータ部をライトバックバッファへ退避します。その後、仮想アドレスに対応する物理アドレス空間から、置換対象ウェイのキャッシュラインへデータを読み込みます。データの読み込みはキャッシュミスしたデータを含むクワッドワード（8バイト）から順にラップアラウンド方式で行い、該当するデータがキャッシュへ到着した時点で、CPUへ読み出しデータを返します。残りのキャッシュ1ライン分のデータが読み込まれている間、CPUは次の処理を実行することができます。キャッシュに1ライン分のデータの読み込みが完了した時点で、物理アドレスによるタグを登録し、Vビットに1を、Uビットに0を書き込みます。また置換したウェイが最新となるようにLRUビットを更新します。その後、ライトバックバッファのデータを外部メモリへ書き戻します。

8.3.2 プリフェッチ動作

オペランドキャッシュ（OC）が有効（CCR.OCE=1）かつキャッシング可能な領域からデータを OC にプリフェッチする場合、OC は以下のように動作します。

1. 仮想アドレスのビット[12:5]でインデックスされる各ウェイのキャッシュラインから、タグ、Vビット、UビットおよびLRUビットを読み出します。
2. 仮想アドレスをMMUにより変換した物理アドレスのビット[28:10]と、各ウェイから読み出したタグを比較し、

- タグが一致かつVビットが1のウェイが存在する場合 → 3.
- タグが一致かつVビットが1のウェイが存在せず、LRUビットにより選択された置換対象ウェイのUビットが0の場合 → 4.
- タグが一致かつVビットが1のウェイが存在せず、LRUビットにより選択された置換対象ウェイのUビットが1の場合 → 5.

3. キャッシュヒット

   ヒットしたウェイが最新となるようにLRUビットを更新します。

4. キャッシュミス（書き戻しなし）

   仮想アドレスに対応する物理アドレス空間から、置換対象ウェイのキャッシュラインへデータを読み込みます。データの読み込みはキャッシュミスしたデータを含むクワッドワード（8バイト）から順にラップアラウンド方式で行います。プリフェッチ動作ではCPUがデータの到着を待つことはなく、キャッシュ1ライン分のデータが読み込まれている間、CPUは次の処理を実行することができます。キャッシュに1ライン分のデータの読み込みが完了した時点で、物理アドレスによるタグを登録し、Vビットに1を、Uビットに0を書き込みます。また置換したウェイが最新となるようにLRUビットを更新します。

5. キャッシュミス（書き戻しあり）

   置換対象ウェイのキャッシュラインのタグとデータ部をライトバックバッファへ退避します。その後、仮想アドレスに対応する物理アドレス空間から、置換対象ウェイのキャッシュラインへデータを読み込みます。データの読み込みはキャッシュミスしたデータを含むクワッドワード（8バイト）から順にラップアラウンド方式で行います。プリフェッチ動作ではCPUがデータの到着を待つことはなく、キャッシュ1ライン分のデータが読み込まれている間、CPUは次の処理を実行することができます。キャッシュに1ライン分のデータの読み込みが完了した時点で、物理アドレスによるタグを登録し、Vビットに1を、Uビットに0を書き込みます。また置換したウェイが最新となるようにLRUビットを更新します。その後、ライトバックバッファのデータを外部メモリへ書き戻します。

8.3.3 書き込み動作

オペランドキャッシュ（OC）が有効（CCR.OCE=1）かつキャッシング可能な領域に対してデータが書き込まれる場合、OC は以下のように動作します。

1. 仮想アドレスのビット[12:5]でインデックスされる各ウェイのキャッシュラインから、タグ、Vビット、UビットおよびLRUビットを読み出します。
2. 仮想アドレスをMMUにより変換した物理アドレスのビット[28:10]と、各ウェイから読み出したタグの比較、および対象となる領域の属性から、

	コピーバック	ライトスルー
• タグが一致かつVビットが1のウェイが存在する場合	→ 3.	→ 4.
• タグが一致かつVビットが1のウェイが存在せず、LRUビットにより選択された置換対象ウェイのUビットが0の場合	→ 5.	→ 7.
• タグが一致かつVビットが1のウェイが存在せず、LRUビットにより選択された置換対象ウェイのUビットが1の場合	→ 6.	→ 7.

3. キャッシュヒット（コピーバック）

   ヒットしたウェイのデータ部の、仮想アドレスのビット[4:0]でインデックスされるデータ位置に対し、アクセスサイズに応じて書き込みます。またUビットに1を書き込み、ヒットしたウェイが最新となるようにLRUビットを更新します。

4. キャッシュヒット（ライトスルー）

   ヒットしたウェイのデータ部の、仮想アドレスのビット[4:0]でインデックスされるデータ位置に対し、アクセスサイズに応じて書き込むとともに、仮想アドレスに対応する外部メモリに対しても書き込みを行います。またヒットしたウェイが最新となるようにLRUビットを更新します。この場合、Uビットは更新されません。

5. キャッシュミス(コピーバック、書き戻しなし)

   置換対象ウェイのデータ部の、仮想アドレスのビット[4:0]でインデックスされるデータ位置に対し、アクセスサイズに応じて書き込みます。また仮想アドレスに対応する物理アドレス空間から、置換対象ウェイのキャッシュラインへデータを読み込みます(ただし、すでに書き込み済みのキャッシュミスしたデータを除く)。データの読み込みはキャッシュミスしたデータを含むクワッドワード(8バイト)から順にラップアラウンド方式で行います。キャッシュ1ライン分のデータが読み込まれている間、CPUは次の処理を実行することができます。キャッシュに1ライン分のデータの読み込みが完了した時点で、物理アドレスによるタグを登録し、Vビットに1を、Uビットに1を書き込みます。また置換したウェイが最新となるようにLRUビットを更新します。

6. キャッシュミス（コピーバック、書き戻しあり）

   置換対象ウェイのキャッシュラインのタグとデータ部をライトバックバッファへ退避します。その後、置換対象ウェイのデータ部の、仮想アドレスのビット[4:0]でインデックスされるデータ位置に対し、アクセスサイズに応じて書き込みます。また仮想アドレスに対応する物理アドレス空間から、置換対象ウェイのキャッシュラインへデータを読み込みます（ただし、すでに書き込み済みのキャッシュミスしたデータを除く）。

データの読み込みはキャッシュミスしたデータを含むクワッドワード（8バイト）から順にラップアラウンド方式で行います。キャッシュ1ライン分のデータが読み込まれている間、CPUは次の処理を実行することができます。キャッシュに1ライン分のデータの読み込みが完了した時点で、物理アドレスによるタグを登録し、Vビットに1を、Uビットに1を書き込みます。また置換したウェイが最新となるようにLRUビットを更新します。その後、ライトバックバッファのデータを外部メモリへ書き戻します。

7. キャッシュミス（ライトスルー）

   仮想アドレスに対応した外部メモリへ、指定されたアクセスサイズで書き込みを行います。この場合、キャッシュへの書き込みは行われません。タグ、Vビット、Uビット、LRUビットも更新されません。

8.3.4 ライトバックバッファ

SH-4A は、キャッシュミスによりダーティなキャッシュのエントリを外部メモリに追い出す必要が生じた場合、キャッシュへのデータの読み込みを優先させ性能を向上させるために、追い出すキャッシュラインのデータを格納するためのライトバックバッファを内蔵しています。ライトバックバッファはキャッシュ 1 ライン分のデータと追い出す先の物理アドレスで構成されます。

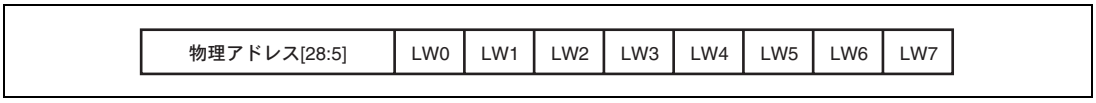


図 8.3 ライトバックバッファの構成

8.3.5 ライトスルーバッファ

SH-4A は、ライトスルーモード時のデータの書き込みや、キャッシング不可能な領域に対する書き込み動作において、書き込みデータを保持するための 64 ビットのバッファを内蔵しています。これにより CPU はライトスルーバッファへの書き込みが完了すると、外部メモリへの書き込みの完了を待たずに次の動作へ移ります。

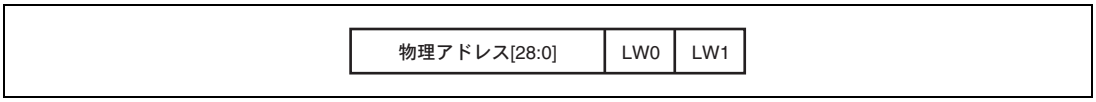


図 8.4 ライトスルーバッファの構成

8.3.6 OC 2 ウェイモード

RAMCR レジスタの OC2W ビットを 1 にセットすると、OC のウェイ 0 とウェイ 1 のみを使用する OC 2 ウェイモードとなり、消費電力を低減できます。本モードではメモリ割り付け OC アクセスも含め、ウェイ 0 とウェイ 1 のみが使用されます。

OC2W ビットの書き換えは P2 領域のプログラムで行ってください。また、書き換える時点ですでに OC に有効なラインが登録されている場合には、OC2W ビットを書き換える前に、必要に応じてソフトウェアにより書き戻しを行った後、CCR レジスタの OCI ビットに 1 を書き込み、OC の全エントリを無効にしてください。

8.4 命令キャッシュの動作説明

8.4.1 読み出し動作

命令キャッシュ（IC）が有効（CCR.ICE=1）かつキャッシング可能な領域から命令フェッチを行う場合、IC は以下のように動作します。

1. 仮想アドレスのビット[12:5]でインデックスされる各ウェイのキャッシュラインから、タグ、Vビットおよび LRUビットを読み出します。
2. 仮想アドレスをMMUにより変換した物理アドレスのビット[28:10]と、各ウェイから読み出したタグを比較し、

- タグが一致かつVビットが1のウェイが存在する場合　→ 3.
- タグが一致かつVビットが1のウェイが存在しない場合　→ 4.

3. キャッシュヒット

   ヒットしたウェイのデータ部から、仮想アドレスのビット[4:3]でインデックスされるデータを命令として読み出します。またヒットしたウェイが最新となるようにLRUビットを更新します。

4. キャッシュミス

   仮想アドレスに対応する物理アドレス空間から、LRUビットにより選択された置換対象ウェイのキャッシュラインへデータを読み込みます。データの読み込みはキャッシュミスしたデータを含むクワッドワード（8バイト）から順にラップアラウンド方式で行い、該当するデータがキャッシュへ到着した時点で、CPUへ読み出しデータを命令として返します。残りのキャッシュ1ライン分のデータが読み込まれている間、CPUは次の処理を実行することができます。キャッシュに1ライン分のデータの読み込みが完了した時点で、物理アドレスによるタグを登録し、Vビットに1を書き込みます。また置換したウェイが最新となるようにLRUビットを更新します。

8.4.2 プリフェッチ動作

命令キャッシュ（IC）が有効（CCR.ICE=1）かつキャッシング可能な領域から、命令を IC にプリフェッチする場合、IC は以下のように動作します。

1. 仮想アドレスのビット[12:5]でインデックスされる各ウェイのキャッシュラインから、タグ、Vビットおよび LRUビットを読み出します。
2. 仮想アドレスをMMUにより変換した物理アドレスのビット[28:10]と、各ウェイから読み出したタグを比較し、

- タグが一致かつVビットが1のウェイが存在する場合　　→ 3.
- タグが一致かつVビットが1のウェイが存在しない場合　　→ 4.

3. キャッシュヒット

   ヒットしたウェイが最新となるようにLRUビットを更新します。

4. キャッシュミス

   仮想アドレスに対応する物理アドレス空間から、置換対象ウェイのキャッシュラインへデータを読み込みます。データの読み込みはキャッシュミスしたデータを含むクワッドワード（8バイト）から順にラップアラウンド方式で行います。プリフェッチ動作ではCPUがデータの到着を待つことはなく、キャッシュ1ライン分のデータが読み込まれている間、CPUは次の処理を実行することができます。キャッシュに1ライン分のデータの読み込みが完了した時点で、物理アドレスによるタグを登録し、Vビットに1を書き込みます。また置換したウェイが最新となるようにLRUビットを更新します。

8.4.3 IC 2 ウェイモード

RAMCR レジスタの IC 2W ビットを 1 にセットすると、IC のウェイ 0 とウェイ 1 のみを使用する IC 2 ウェイモードとなり、消費電力を低減できます。本モードではメモリ割り付け IC アクセスも含め、ウェイ 0 とウェイ 1 のみが使用されます。

IC2W ビットの書き換えは P2 領域のプログラムで行うようにしてください。また、書き換える時点ですでに IC に有効なラインが登録されている場合には、IC2W ビットを書き換える前に、CCR レジスタの ICI ビットに 1 を書き込み、IC の全エントリを無効化してください。

8.5 キャッシュ操作命令

8.5.1 キャッシュと外部メモリとのコヒーレンシ

キャッシュと外部メモリとのコヒーレンシはソフトウェアで保証してください。SH-4A ではキャッシュを操作する命令として次の 6 命令をサポートしています。各命令の詳細は**「第 10 章　各命令の説明」**を参照してください。

- **オペランドキャッシュインバリデイト命令**：OCBI @Rn

  オペランドキャッシュの無効化（書き戻しなし）
- **オペランドキャッシュパージ命令**：OCBP @Rn

  オペランドキャッシュの無効化（書き戻しあり）
- **オペランドキャッシュライトバック命令**：OCBWB @Rn

  オペランドキャッシュの書き戻し
- **オペランドキャッシュアロケート命令**：MOVCA.L R0,@Rn

  オペランドキャッシュの確保
- **命令キャッシュインバリデイト命令**：ICBI @Rn

  命令キャッシュの無効化
- **オペランドアクセス同期命令**：SYNCO

  データ転送の完了待ち

またオペランドキャッシュのコヒーレンシ制御のために、SuperHyway バスからの PURGE および FLUSH トランザクションを受け付けることが可能です。PURGE/FLUSH トランザクションで与えられるアドレスは物理アドレスです。そのため MMU がイネーブルの場合、キャッシュシノニム問題を回避するため、以下の制限事項が生じます。

- 1Kバイトのページサイズを使用しないでください。

(1) PURGE トランザクション

オペランドキャッシュがイネーブルの時、オペランドキャッシュを検索し、ヒットしたエントリを無効化します。無効化されるラインがダーティであれば外部メモリへ書き戻しを行います。ミスした場合にはノーオペレーションです。

(2) FLUSH トランザクション

オペランドキャッシュがイネーブルの時、オペランドキャッシュを検索し、ヒットしたエントリがあり、かつダーティであれば外部メモリへ書き戻しを行います。ヒットしたエントリの無効化は行いません。ミスした場合またはヒットしたエントリがダーティでなかった場合にはノーオペレーションです。

8.5.2 プリフェッチ動作

キャッシュミスにより発生するキャッシュフィルのペナルティを削減するために、SH-4A ではプリフェッチ命令をサポートしています。読み出し動作、書き込み動作によりキャッシュミスの発生することがわかっていた場合、プリフェッチ命令によりあらかじめキャッシュへデータをフィルしておき、読み出し動作、書き込み動作においてキャッシュミスを発生させないようにできます。これによりソフトウェアの性能が向上します。すでにキャッシュに格納されているデータに対して、プリフェッチ命令を実行したり、プリフェッチしようとしたアドレスが UTLB にミスした場合やプロテクションに違反した場合は、ノーオペレーションとなり例外を発生させません。プリフェッチ命令の詳細は**「第 10 章　各命令の説明」**を参照してください。

- プリフェッチ命令（OC）　：PREF @Rn
- プリフェッチ命令（IC）　：PREFI @Rn

8.6 メモリ割り付けキャッシュの構成

IC、OC をソフトウェアで管理するために、特権モードのとき、P2 領域のプログラムから MOV 命令によって IC の内容の読み出し／書き込みが可能です。他の領域のプログラムからのアクセスは保証しません。この場合、P0、U0、P1、P3 領域への分岐は、以下の 1～3 のどれかの方法で行ってください。

1. RTE命令による分岐を実行してください。
2. 任意のアドレス（キャッシング不可領域でもよい）に対して、ICBI命令を実行した後、P0、U0、P1、P3領域への分岐を行ってください。
3. メモリ割り付けICへのアクセスの前に、あらかじめIRMCR.MC＝0（初期値）と設定されていた場合には、特定の命令シーケンスは不要です。しかしこの方法では、メモリ割り付けICアクセス命令の次命令を命令フェッチからやり直すため、CPUの処理性能が低下しますのでご注意ください。

ただし、方法 3 は今後の SuperH シリーズでは保証されない可能性があります。今後の SuperH シリーズでの互換性を保証するためには、1 または 2 を用いることを推奨します。

また、特権モードのとき、P1、P2 領域のプログラムから MOV 命令によって OC の内容の読み出し／書き込みが可能です。他の領域のプログラムからのアクセスは保証しません。IC、OC は仮想アドレス空間の P4 領域に割り付けられています。IC のアドレスアレイ／データアレイ、OC のアドレスアレイ／データアレイともにデータアクセスのみ可能でアクセスサイズはロングワード固定です。この領域に対して命令フェッチは行えません。予約ビットには 0 を設定するようにしてください。予約ビットの読み出し値は不定です。

8.6.1 IC アドレスアレイ

IC のアドレスアレイは P4 領域の H'F000 0000～H'F0FF FFFF に割り付けられています。アドレスアレイのアクセスには 32 ビットのアドレス部の指定（読み出し／書き込み時）と 32 ビットのデータ部の指定が必要です。アドレス部ではアクセスするウェイとエントリを指定し、データ部には書き込みタグと V ビットを指定します。

アドレス部は[31:24]が IC アドレスアレイを示す H'F0 になっており、[14:13]でウェイ、[12:5]でエントリを指定するようになっています。アドレス部[3]の連想ビット（A ビット）は IC アドレスアレイへの書き込みのときに連想を行うかどうかを指定します。アクセスはロングワードサイズ固定なのでアドレス部[1:0]は 0 を指定してください。

データ部は[31:10]がタグを、[0]が V ビットを示します。IC アドレスアレイのタグは 19 ビットのためデータ部[31:29]は連想を行わない書き込みのときには使用されません。データ部[31:29]は連想を行う書き込みのときのみ仮想アドレスの指定のため用います。

IC アドレスアレイに対しては次の 3 種類の操作が可能です。

(1) IC アドレスアレイ　読み出し

アドレス部に設定されたウェイとエントリに対応するICエントリから、データ部へタグとVビットを読み出します。読み出す場合アドレス部に指定される連想ビットは1でも0でも連想動作は行いません。

(2) IC アドレスアレイ　書き込み（連想なし）

アドレス部に設定されたウェイとエントリに対応するICエントリに対して、データ部で指定されたタグとVビットを書き込みます。アドレス部のAビットは0にしてください。

(3) IC アドレスアレイ　書き込み（連想あり）

アドレス部のAビットが1で書き込みのとき、アドレス部で指定されたエントリに格納されている各ウェイのタグとデータ部で指定されたタグとの間で一致判定が行われます。アドレス部[14:13]のウェイ番号は使用されません。このときMMUがイネーブルなら、データ部[31:10]で指定した仮想アドレスをITLBを用い物理アドレスに変換してから一致判定を行います。アドレスが一致しそのウェイのVビットが1であったなら、データ部で指定したVビットをICのエントリに書き込みます。それ以外の場合はノーオペレーションとなります。本動作はICの特定のエントリの無効化に用いられます。アドレス変換の際にITLBにミスした場合や、一致判定で不一致になった場合、例外は発生せずノーオペレーションとなり書き込みは行われません。

【注】　本機能は今後の SuperH シリーズではサポートされない可能性があります。ITLB ミスハンドリングや命令 TLB ミス例外の通知を行い、確実に IC の操作が可能な ICBI 命令の使用を推奨します。

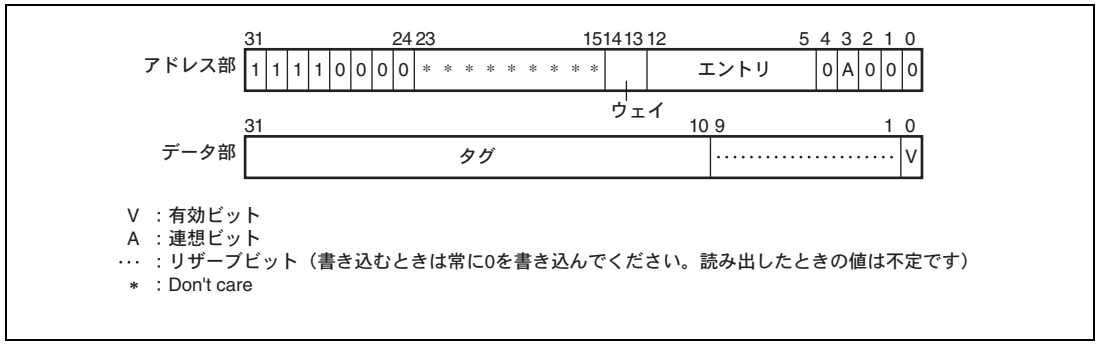


図 8.5　メモリ割り付け IC アドレスアレイ

8.6.2　IC データアレイ

IC のデータアレイは P4 領域の H'F100 0000～H'F1FF FFFF に割り付けられています。データアレイのアクセスには 32 ビットのアドレス部の指定（読み出し／書き込み時）と 32 ビットのデータ部の指定が必要です。アドレス部ではアクセスするウェイとエントリを指定し、データ部には書き込むロングワードデータを指定します。

アドレス部は[31:24]が IC データアレイを示す H'F1 になっており、[14:13]でウェイ、[12:5]でエントリを指定するようになっています。アドレス部[4:2]はエントリ内のロングワードデータの指定に用います。アクセスはロングワードサイズ固定なのでアドレス部[1:0]は 0 を指定してください。

データ部はロングワードデータの指定に用います。

IC データアレイに対しては次の 2 種類の操作が可能です。

(1) IC データアレイ　読み出し

アドレス部に設定されたウェイとエントリに対応するICエントリのうち、アドレス部のロングワード指定ビットで指定されたデータから、データ部へロングワードデータを読み出します。

(2) IC データアレイ　書き込み

アドレス部に設定されたウェイとエントリに対応するICエントリのうち、アドレス部のロングワード指定ビットで指定されたデータに対して、データ部で指定されたロングワードデータを書き込みます。

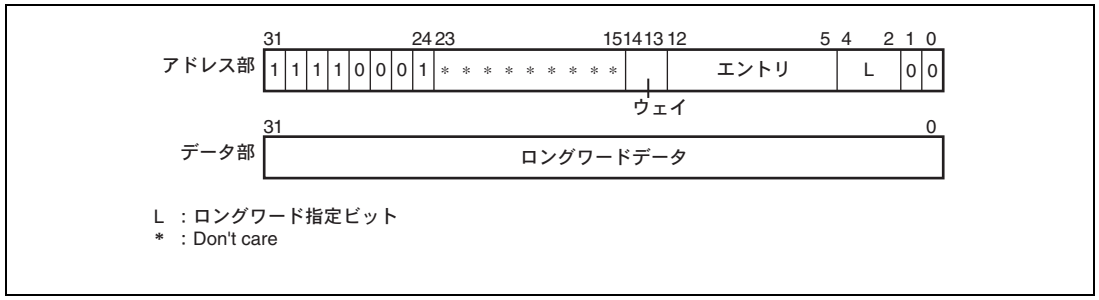


図 8.6　メモリ割り付け IC データアレイ

8.6.3 OC アドレスアレイ

OC のアドレスアレイは P4 領域の H'F400 0000～H'F4FF FFFF に割り付けられています。アドレスアレイのアクセスには 32 ビットのアドレス部の指定（読み出し／書き込み時）と 32 ビットのデータ部の指定が必要です。アドレス部ではアクセスするウェイとエントリを指定し、データ部には書き込みタグと U ビットと V ビットを指定します。

アドレス部は[31:24]が OC アドレスアレイを示す H'F4 になっており、[14:13]でウェイ、[12:5]でエントリを指定するようになっています。アドレス部[3]の連想ビット（A ビット）は OC アドレスアレイへの書き込みのときに連想を行うかどうかを指定します。アクセスはロングワードサイズ固定ですのでアドレス部[1:0]は 0 を指定してください。

データ部は[31:10]がタグを、[1]が U ビットを、[0]が V ビットを示します。OC アドレスアレイのタグは 19 ビットのため、データ部[31:29]は連想を行わない書き込みのときには使用されません。データ部[31:29]は連想を行う書き込みのときのみ仮想アドレスの指定のため用います。

OC アドレスアレイに対しては次の 3 種類の操作が可能です。

(1) OC アドレスアレイ　読み出し

アドレス部に設定されたウェイとエントリに対応するOCエントリから、データ部へタグとUビットとVビットを読み出します。読み出す場合、アドレス部に指定される連想ビットは1でも0でも連想動作は行いません。

(2) OC アドレスアレイ　書き込み（連想なし）

アドレス部に設定されたウェイとエントリに対応するOCエントリに対して、データ部で指定されたタグとUビットとVビットを書き込みます。アドレス部のAビットは0にしてください。

書き込みをUビットが1、Vビットが1のキャッシュラインに対して行った場合、そのキャッシュラインの書き戻しを行った後、データ部で指定されたタグとUビットとVビットを書き込みます。

(3) OC アドレスアレイ　書き込み（連想あり）

アドレス部のAビットが1で書き込みのとき、アドレス部で指定されたエントリに格納されている各ウェイのタグとデータ部で指定されたタグとの間で一致判定が行われます。ビット[14:13]のウェイ番号は使用されません。このときMMUがイネーブルなら、データ部[31:10]で指定した仮想アドレスをUTLBを用い物理アドレスに変換してから一致判定を行います。アドレスが一致しそのウェイのVビットが1であったなら、データ部で指定したUビットとVビットをOCのエントリに書き込みます。それ以外の場合はノーオペレーションとなります。本動作はOCの特定のエントリの無効化に用いられます。このときOCのエントリのUビットが1で、Vビットに0もしくはUビットに0を書き込んだ場合、書き戻しが発生します。アドレス変換の際にUTLBにミスした場合や、一致判定で不一致になった場合、例外は発生せずノーオペレーションとなり書き込みは行われません。

【注】　本機能は今後の SuperH シリーズではサポートされない可能性があります。データ TLB ミス例外の通知を行い、確実に OC の操作が可能な OCBI/OCBP/OCBWB 命令の使用を推奨します。

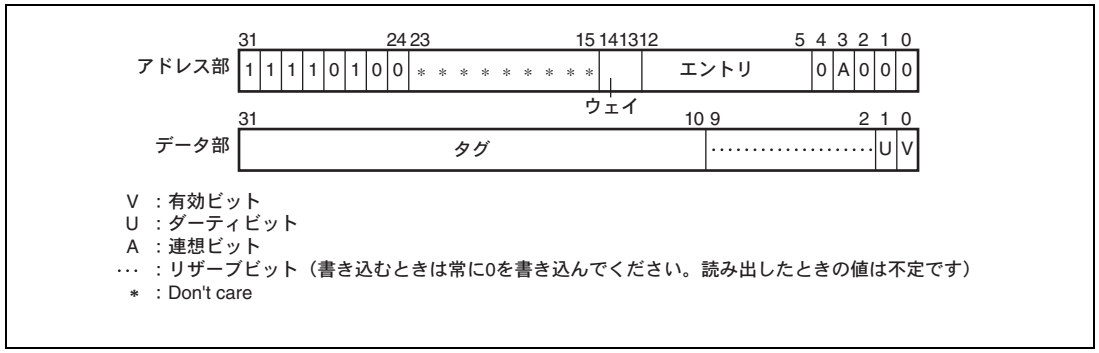


図 8.7　メモリ割り付け OC アドレスアレイ

8.6.4　OC データアレイ

OC のデータアレイは P4 領域の H'F500 0000～H'F5FF FFFF に割り付けられています。データアレイのアクセスには 32 ビットのアドレス部の指定（読み出し／書き込み時）と 32 ビットのデータ部の指定が必要です。アドレス部ではアクセスするウェイとエントリを指定し、データ部には書き込むロングワードデータを指定します。

アドレス部は[31:24]が OC データアレイを示す H'F5 になっており、[14:13]でウェイ、[12:5]でエントリを指定するようになっています。アドレス部[4:2]はエントリ内のロングワードデータの指定に用います。アクセスはロングワードサイズ固定なのでアドレス部[1:0]は 0 を指定してください。

データ部はロングワードデータの指定に用います。

OC データアレイに対しては次の 2 種類の操作が可能です。

(1) OC データアレイ　読み出し

アドレス部に設定されたウェイとエントリに対応するOCエントリのうち、アドレス部のロングワード指定ビットで指定されたデータから、データ部へロングワードデータを読み出します。

(2) OC データアレイ　書き込み

アドレス部に設定されたウェイとエントリに対応するOCエントリのうち、アドレス部のロングワード指定ビットで指定されたデータに対して、データ部で指定されたロングワードデータを書き込みます。この書き込みによりアドレスアレイ側のUビットは1になりません。

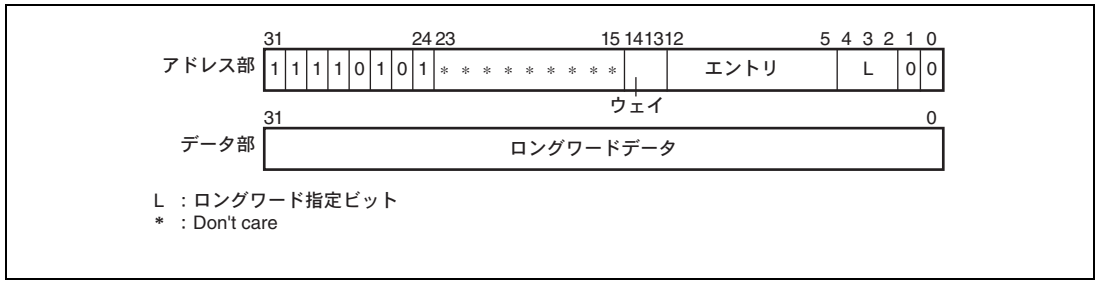


図 8.8　メモリ割り付け OC データアレイ

8.7 ストアキュー

SH-4A では、外部メモリへの高速な書き込みを行うために 32 バイト×2 のストアキュー（SQ）をサポートします。

8.7.1 SQ の構成

SQ は図 8.9 に示すとおり、32 バイトの SQ0 と 32 バイトの SQ1 から成り立っています。SQ0、SQ1 はそれぞれ独立に設定することが可能です。

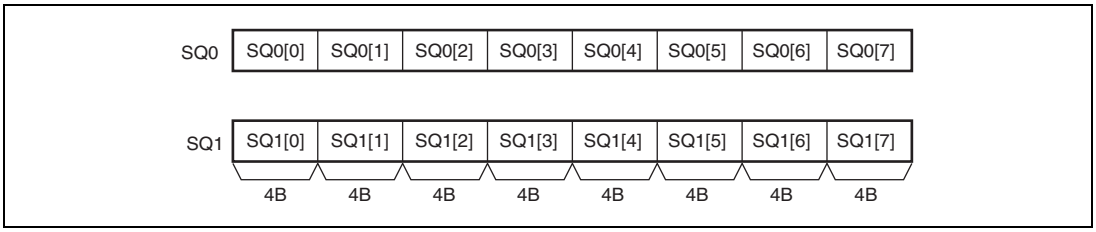


図 8.9 ストアキューの構成

8.7.2 SQ への書き込み

SQ への書き込みは P4 領域の H'E000 0000～H'E3FF FFFC に対するストア命令で行うことができます。アクセスサイズはロングワード、もしくはクワッドワードが可能です。このアドレスは以下の意味を持ちます。

[31:26]	：111000	：ストアキュー指定
[25:6]	：Don't care	：外部メモリへの転送・アクセス権で使用
[5]	：0/1	：0:SQ0 指定　1:SQ1 指定
[4:2]	：LW 指定	：SQ0、SQ1 内のロングワード位置を指定
[1:0]	：00	：0 固定

8.7.3 外部メモリへの転送

SQ から外部メモリへの転送は、プリフェッチ命令（PREF）により行えます。PREF 命令を P4 領域の H'E000 0000～H'E3FF FFFC に対して発行することにより、SQ から外部メモリへの転送が開始します。転送は 32 バイト固定で、開始アドレスは必ず 32 バイト境界となります。一方の SQ を外部メモリへ転送中に、もう一方の SQ への書き込みはペナルティサイクルなしに行えますが、外部メモリへ転送中の SQ への書き込みは外部メモリへの転送が完了するまで待たされます。

SQ の転送先の物理アドレス[28:0]は MMU イネーブル／ディスエーブルにより次のように指定します。

(1) MMU イネーブル（MMUCR.AT＝1）の場合

UTLBのVPNにSQ領域（H'E000 0000～H'E3FF FFFF）を、PPNに転送先の物理アドレスを設定します。ASID、V、SZ、SH、PR、Dビットは通常のアドレス変換と同様の意味を持ちますが、C、WTビットはこのページに関しては意味を持ちません。

SQ領域へのプリフェッチ命令が発行されると、アドレス変換を行い、SZビットの指定に従い物理アドレス[28:10]を生成します。物理アドレスの[9:5]についてはMMUディスエーブルと同様にアドレス変換前のアドレスから生成します。物理アドレスの[4:0]は0固定です。SQから外部メモリへの転送はこのアドレスに対して行われます。

(2) MMU ディスエーブル（MMUCR.AT＝0）の場合

PREF命令を発行するアドレスにSQ領域（H'E000 0000～H'E3FF FFFF）を指定します。このアドレス[31:0]は次の意味を持ちます。

[31:26]	：111000	：ストアキュー指定
[25:6]	：アドレス	：転送先物理アドレス[25:6]
[5]	：0/1	：0:SQ0指定　1:SQ1指定　かつ　転送先物理アドレス[5]
[4:2]	：Don't care	：プリフェッチのときは意味を持たない。
[1:0]	：00	：0固定

上記のアドレスから生成できない物理アドレス[28:26]は、QACR0、QACR1から生成します。

QACR0[4:2]	：SQ0に対する物理アドレス[28:26]
QACR1[4:2]	：SQ1に対する物理アドレス[28:26]

物理アドレスの[4:0]は、バースト転送の開始が32バイト境界のため常に0固定となります。

8.7.4 SQ アクセスの例外判定

SQ への書き込み、および外部メモリへの転送（PREF 命令）の例外判定は MMU イネーブル／ディスエーブルにより次のように行われます。なお、SQ への書き込みで例外が発生した場合、SQ の内容は元の値が保証されます。SQ から外部メモリへの転送で例外が発生した場合、外部メモリへの転送は抑止されます。

（1） MMU イネーブル（MMUCR.AT＝1）の場合

UTLBに登録されたアドレス変換情報とSQMDビットに従います。SQへの書き込みはライトタイプ、SQから外部メモリへの転送（PREF命令）はリードタイプとして例外判定が行われ、TLBミス例外、保護違反例外が発生します。ただし、SQMDビットによりSQへのアクセスを特権モードのみ許可している場合、ユーザモードでアドレス変換に成功してもアドレスエラーとなります。

（2） MMU ディスエーブル（MMUCR.AT＝0）の場合

SQMDビットに従います。

0：特権／ユーザアクセス可能

1：特権アクセス可能

SQMDビットが1のときに、ユーザモードでSQ領域をアクセスするとアドレスエラーが発生します。

8.7.5 SQ からの読み出し

SH-4A では、特権モードのとき、SQ からの読み出しを P4 領域の H'FF00 1000～H'FF00 103C に対するロード命令で行うことができます。アクセスサイズはロングワードでのみアクセス可能です。

[31:6]	：H'FF00 1000	：ストアキュー指定
[5]	：0/1	：0：SQ0 指定、1：SQ1 指定
[4:2]	：LW 指定	：SQ0、SQ1 内のロングワード位置を指定
[1:0]	：00	：0 固定

8.8 32 ビットアドレス拡張モード使用時の注意事項

32 ビットアドレス拡張モードでは、本章ですでに述べた事項が以下のように拡張されます。

1. ICおよびOCのタグが[28:10]の19ビットから、[31:10]の22ビットに拡張されます。
2. ICを操作する命令（メモリ割り付けICアクセスおよびCCR.ICI書き込み）を配置する領域は、P1またはP2領域とし、PMBの当該エントリのキャッシング可能ビット（Cビット）を0にしてください。
3. QACR0レジスタのAREA0ビットおよびQACR1レジスタのAREA1ビットがそれぞれ[4:2]の3ビットから[7:2]の6ビットに拡張されます。

9. Lメモリ

SH-4A はL メモリモジュールを内蔵しており、命令やデータを格納することができます。

【注】 Lメモリの容量については、製品のハードウェアマニュアルを参照してください。

9.1 特長

- 容量：

  Lメモリ合計で、16Kバイト、32Kバイト、64Kバイト、128Kバイトから選択可能です。

- ページ：

  Lメモリは2ページ（ページ0および1）に分かれています。

- メモリマップ：

  本メモリは、仮想アドレス空間、物理アドレス空間とも、**表9.1**に示されるアドレスに配置されています。

表 9.1 Lメモリアドレス

ページ	メモリサイズ（2 ページ合計）			
	16K バイト	32K バイト	64K バイト	128K バイト
L メモリ ページ 0	H'E500E000 ～H'E500FFFF	H'E500C000 ～H'E500FFFF	H'E5008000 ～H'E500FFFF	H'E5000000 ～H'E500FFFF
L メモリ ページ 1	H'E5010000 ～H'E5011FFF	H'E5010000 ～H'E5013FFF	H'E5010000 ～H'E5017FFF	H'E5010000 ～H'E501FFFF

- ポート：

  各ページは3本の独立した読み出し／書き込みポートを持ち、各バスと接続されています。命令フェッチには命令バスが、オペランドアクセスにはオペランドバスが、SuperHywayバスマスタモジュールからのアクセスにはSuperHywayバスがそれぞれ使用されます。

- 優先順位：

  同じページに対して異なるバスから同時にアクセス要求があった場合には、優先順位に従ってアクセスが処理されます。優先順位は高い順にSuperHywayバス、オペランドバス、命令バスとなります。

9.2 レジスタの説明

L メモリに関するレジスタは以下のとおりです。

表 9.2 レジスタ構成

名称	略称	R/W	P4 領域 アドレス*	エリア 7 アドレス*	サイズ
内蔵メモリ制御レジスタ	RAMCR	R/W	H'FF00 0074	H'1F00 0074	32
L メモリ転送元アドレスレジスタ 0	LSA0	R/W	H'FF00 0050	H'1F00 0050	32
L メモリ転送元アドレスレジスタ 1	LSA1	R/W	H'FF00 0054	H'1F00 0054	32
L メモリ転送先アドレスレジスタ 0	LDA0	R/W	H'FF00 0058	H'1F00 0058	32
L メモリ転送先アドレスレジスタ 1	LDA1	R/W	H'FF00 005C	H'1F00 005C	32

【注】 * P4 領域アドレスは、仮想アドレス空間の P4 領域を用いた場合のものです。エリア 7 アドレスは、TLB を用いて物理アドレス空間のエリア 7 からアクセスするものです。

表 9.3 各処理状態におけるレジスタの状態

名称	略称	パワーオン リセット	マニュアル リセット	スリープ	スタンバイ
内蔵メモリ制御レジスタ	RAMCR	H'0000 0000	H'0000 0000	保持	保持
L メモリ転送元アドレスレジスタ 0	LSA0	不定	不定	保持	保持
L メモリ転送元アドレスレジスタ 1	LSA1	不定	不定	保持	保持
L メモリ転送先アドレスレジスタ 0	LDA0	不定	不定	保持	保持
L メモリ転送先アドレスレジスタ 1	LDA1	不定	不定	保持	保持

9.2.1 内蔵メモリ制御レジスタ（RAMCR）

RAMCR は L メモリの保護機能の制御を行います。

ビット名：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

ビット名：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	—	—	RMD	RP	IC2W	OC2W	—	—	—	—	—	—
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R/W	R/W	R/W	R/W	R	R	R	R	R	R

ビット	ビット名	初期値	R/W	説　　明
31～10	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
9	RMD	0	R/W	内蔵メモリアクセスモードビット 仮想アドレス空間からの L メモリへのアクセス権を指定します。 0：特権アクセスが可能(ユーザアクセスの場合はアドレスエラー例外) 1：ユーザ／特権アクセスが可能
8	RP	0	R/W	内蔵メモリ保護有効ビット 仮想アドレス空間からの L メモリへのアクセスに対して、ITLB、UTLB を用いた保護機能の使用を選択します。 0：保護機能を使用しない 1：保護機能を使用する 詳細は「9.4　L メモリの保護機能」を参照してください。
7	IC2W	0	R/W	IC 2 ウェイモードビット 詳細は「8.4.3　IC 2 ウェイモード」を参照してください。
6	OC2W	0	R/W	OC 2 ウェイモードビット 詳細は「8.3.6　OC 2 ウェイモード」を参照してください。
5～0	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。

9.2.2 L メモリ転送元アドレスレジスタ 0（LSA0）

LSA0 は、MMUCR.AT＝0 または RAMCR.RP＝0 のときに、L メモリページ 0 へのブロック転送において、転送元の物理アドレスを指定します。

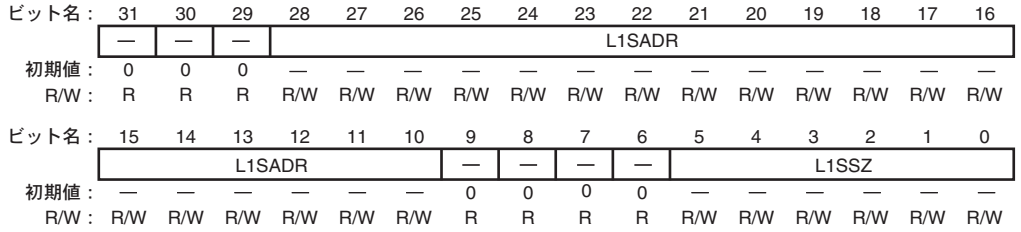
ビット名：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	L0SADR												
初期値：	0	0	0	—	—	—	—	—	—	—	—	—	—	—	—	—
R/W：	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ビット名：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	L0SADR						—	—	—	—	L0SSZ					
初期値：	—	—	—	—	—	—	0	0	0	0	—	—	—	—	—	—
R/W：	R/W	R/W	R/W	R/W	R/W	R/W	R	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W

ビット	ビット名	初期値	R/W	説　　明
31～29	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
28～10	L0SADR	不定	R/W	L メモリページ 0 ブロック転送元アドレス MMUCR.AT＝0 または RAMCR.RP＝0 のとき、L メモリページ 0 に対するブロック転送の転送元となる物理アドレスを指定します。
9～6	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
5～0	L0SSZ	不定	R/W	L メモリページ 0 ブロック転送元アドレス選択ビット MMUCR.AT＝0 または RAMCR.RP＝0 のとき、L メモリページ 0 に対するブロック転送の転送元となる物理アドレスのうちビット 15～10 に関して、オペランドアドレスを使用するか、L0SADR の値を使用するかを選択します。L0SSZ[5:0]が転送元物理アドレスの[15:10]に対応します。 0：転送元物理アドレスにオペランドアドレスを使用します。 1：転送元物理アドレスに L0SADR の値を使用します。 • 設定可能な値 111111　転送元の物理アドレスを 1K バイト単位で設定する場合 111110　転送元の物理アドレスを 2K バイト単位で設定する場合 111100　転送元の物理アドレスを 4K バイト単位で設定する場合 111000　転送元の物理アドレスを 8K バイト単位で設定する場合 110000　転送元の物理アドレスを 16K バイト単位で設定する場合 100000　転送元の物理アドレスを 32K バイト単位で設定する場合 000000　転送元の物理アドレスを 64K バイト単位で設定する場合 上記以外は設定禁止です。

9.2.3 L メモリ転送元アドレスレジスタ 1（LSA1）

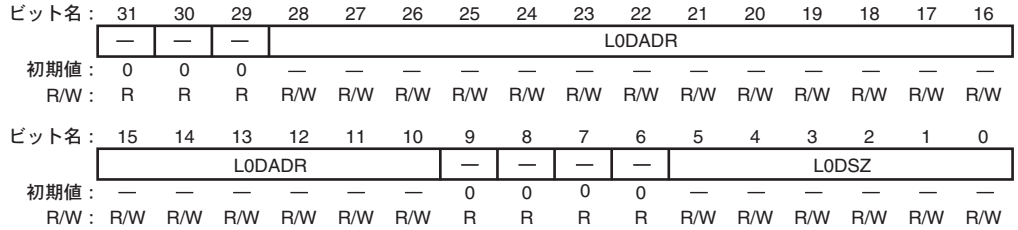
LSA1 は、MMUCR.AT＝0 または RAMCR.RP＝0 のときに、L メモリページ 1 へのブロック転送において、転送元の物理アドレスを指定します。



ビット	ビット名	初期値	R/W	説　　明
31～29	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
28～10	L1SADR	不定	R/W	L メモリページ 1 ブロック転送元アドレス MMUCR.AT＝0 または RAMCR.RP＝0 のとき、L メモリページ 1 に対するブロック転送の転送元となる物理アドレスを指定します。
9～6	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
5～0	L1SSZ	不定	R/W	L メモリページ 1 ブロック転送元アドレス選択ビット MMUCR.AT＝0 または RAMCR.RP＝0 のとき、L メモリページ 1 に対するブロック転送の転送元となる物理アドレスのうちビット 15～10 に関して、オペランドアドレスを使用するか、L1SADR の値を使用するかを選択します。L1SSZ[5:0]が転送元物理アドレスの[15:10]に対応します。 0：転送元物理アドレスにオペランドアドレスを使用します。 1：転送元物理アドレスに L1SADR の値を使用します。 • 設定可能な値 111111　転送元の物理アドレスを 1K バイト単位で設定する場合 111110　転送元の物理アドレスを 2K バイト単位で設定する場合 111100　転送元の物理アドレスを 4K バイト単位で設定する場合 111000　転送元の物理アドレスを 8K バイト単位で設定する場合 110000　転送元の物理アドレスを 16K バイト単位で設定する場合 100000　転送元の物理アドレスを 32K バイト単位で設定する場合 000000　転送元の物理アドレスを 64K バイト単位で設定する場合 上記以外は設定禁止です。

9.2.4 L メモリ転送先アドレスレジスタ 0（LDA0）

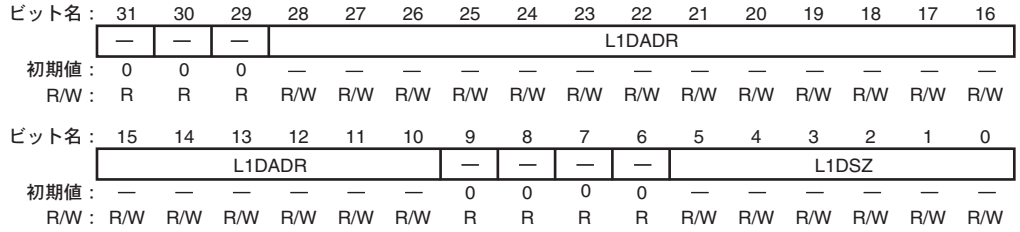
LDA0 は、MMUCR.AT＝0 または RAMCR.RP＝0 のときに、L メモリページ 0 へのブロック転送において、転送先の物理アドレスを指定します。



ビット	ビット名	初期値	R/W	説　　明
31～29	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
28～10	L0DADR	不定	R/W	L メモリページ 0 ブロック転送先アドレス MMUCR.AT＝0 または RAMCR.RP＝0 のとき、L メモリページ 0 に対するブロック転送の転送先となる物理アドレスを指定します。
9～6	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
5～0	L0DSZ	不定	R/W	L メモリページ 0 ブロック転送先アドレス選択ビット MMUCR.AT＝0 または RAMCR.RP＝0 のとき、L メモリページ 0 に対するブロック転送の転送先となる物理アドレスのうちビット 15～10 に関して、オペランドアドレスを使用するか、L0DADR の値を使用するかを選択します。L0DSZ[5:0]が転送先物理アドレスの[15:10]に対応します。 0：転送先物理アドレスにオペランドアドレスを使用します。 1：転送先物理アドレスに L0DADR の値を使用します。 • 設定可能な値 111111　転送先の物理アドレスを 1K バイト単位で設定する場合 111110　転送先の物理アドレスを 2K バイト単位で設定する場合 111100　転送先の物理アドレスを 4K バイト単位で設定する場合 111000　転送先の物理アドレスを 8K バイト単位で設定する場合 110000　転送先の物理アドレスを 16K バイト単位で設定する場合 100000　転送先の物理アドレスを 32K バイト単位で設定する場合 000000　転送先の物理アドレスを 64K バイト単位で設定する場合 上記以外は設定禁止です。

9.2.5 L メモリ転送先アドレスレジスタ 1（LDA1）

LDA1 は、MMUCR.AT＝0 または RAMCR.RP＝0 のときに、L メモリページ 1 へのブロック転送において、転送先の物理アドレスを指定します。



ビット	ビット名	初期値	R/W	説　　明
31～29	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
28～10	L1DADR	不定	R/W	L メモリページ 1 ブロック転送先アドレス MMUCR.AT＝0 または RAMCR.RP＝0 のとき、L メモリページ 1 に対するブロック転送の転送先となる物理アドレスを指定します。
9～6	－	すべて 0	R	リザーブビット 本ビットの読み出し／書き込みに関しては「製品に関する一般的注意事項」を参照してください。
5～0	L1DSZ	不定	R/W	L メモリページ 1 ブロック転送先アドレス選択ビット MMUCR.AT＝0 または RAMCR.RP＝0 のとき、L メモリページ 1 に対するブロック転送の転送先となる物理アドレスのうちビット 15～10 に関して、オペランドアドレスを使用するか、L1DADR の値を使用するかを選択します。L1DSZ[5:0]が転送先物理アドレスの[15:10]に対応します。 0:転送先物理アドレスにオペランドアドレスを使用します。 1:転送先物理アドレスに L1DADR の値を使用します。 • 設定可能な値 111111　転送先の物理アドレスを 1K バイト単位で設定する場合 111110　転送先の物理アドレスを 2K バイト単位で設定する場合 111100　転送先の物理アドレスを 4K バイト単位で設定する場合 111000　転送先の物理アドレスを 8K バイト単位で設定する場合 110000　転送先の物理アドレスを 16K バイト単位で設定する場合 100000　転送先の物理アドレスを 32K バイト単位で設定する場合 000000　転送先の物理アドレスを 64K バイト単位で設定する場合 上記以外は設定禁止です。

9.3 動作説明

9.3.1 CPU および FPU からのアクセス

CPU および FPU からのアクセスは仮想アドレスにより、命令バスまたはオペランドバスから行います。ページ競合が発生しない限り 1 サイクルアクセスになります。

9.3.2 SuperHyway バスマスタモジュールからのアクセス

DMAC などの SuperHyway バスマスタモジュールからの本メモリへのアクセスは、物理アドレスバスである SuperHyway バスからのアクセスとなりますが、仮想アドレスと同じアドレスを使用してください。

9.3.3 ブロック転送

L メモリと外部メモリの間で、キャッシュを介さずに、ブロック転送により高速にデータ転送を行うことができます。

外部メモリから L メモリへの転送は、プリフェッチ命令（PREF）により行えます。PREF 命令を仮想アドレス空間の L メモリ領域のアドレスに対して発行することにより、外部メモリから L メモリへのブロック転送が開始されます。

L メモリから外部メモリへの転送は、ライトバック命令（OCBWB）により行えます。OCBWB 命令を仮想アドレス空間の L メモリ領域のアドレスに対して発行することにより、L メモリから外部メモリへのブロック転送が開始されます。

いずれの転送も転送サイズは 32 バイト固定で、開始アドレスは必ず 32 バイト境界となるため、レジスタ Rn により指示されるアドレスの下位 5 ビットは無視され、常にすべて 0 として扱われます。またいずれの場合もブロック転送中に他のページやキャッシュに対するアクセスが可能ですが、転送中のページにアクセスした場合、転送が終了するまで CPU はストールします。

L メモリと転送を行う外部メモリの物理アドレス[28:0]は MMU イネーブル／ディスエーブルにより次のように指定します。

（1） MMU イネーブル（MMUCR.AT＝1）かつ RAMCR.RP＝1 の場合

UTLB の VPN フィールドに L メモリ領域のアドレスを、PPN フィールドに転送元（PREF 命令の場合）または転送先（OCBWB 命令の場合）の物理アドレスを設定します。ASID、V、SZ、SH、PR、D ビットは通常のアドレス変換と同様の意味を持ちますが、C、WT ビットはこのページに関しては意味を持ちません。

L メモリ領域への PREF 命令が発行されると、アドレス変換を行い、SZ ビットの指定に従い物理アドレス[28:10]を生成します。物理アドレスの[9:5]についてはアドレス変換前の仮想アドレスから生成します。物理アドレスの[4:0]は 0 固定です。この物理アドレスで指定される外部メモリから L メモリへブロック転送が行われます。

L メモリ領域への OCBWB 命令が発行されると、アドレス変換を行い、SZ ビットの指定に従い物理アドレス[28:10]を生成します。物理アドレスの[9:5]についてはアドレス変換前の仮想アドレスから生成します。物理アドレスの[4:0]は 0 固定です。L メモリからこの物理アドレスで指定される外部メモリへブロック転送が行われます。

PREF 命令、OCBWB 命令はリードタイプとして MMU 例外の判定が行われ、必要に応じて TLB ミス例外、保護違反例外が発生します。例外が発生した場合、ブロック転送は抑止されます。

（2） MMU ディスエーブル（MMUCR.AT＝0）または RAMCR.RP＝0 の場合

LSA0 レジスタの L0SADR ビットに L メモリページ 0 へのブロック転送の転送元となる物理アドレスを設定し、L0SSZ ビットに、転送元の物理アドレスのビット 15～10 として PREF 命令で指定された仮想アドレスを使用するか、L0SADR の値を使用するかをソフトウェアにより設定します。すなわち転送元の領域を 1K バイト～64K バイト単位で設定可能です。

LDA0 レジスタの L0DADR ビットに L メモリページ 0 からのブロック転送の転送先となる物理アドレスを設定し、L0DSZ ビットに、転送先の物理アドレスのビット 15～10 として OCBWB 命令で指定された仮想アドレスを使用するか、L0DADR の値を使用するかをソフトウェアにより設定します。すなわち転送先の領域を 1K バイト～64K バイト単位で設定可能です。

L メモリページ 1 に対するブロック転送の設定も、ページ 0 と同様に LSA1 および LDA1 に対して行います。

L メモリ領域への PREF 命令が発行されると、LSA0 レジスタまたは LSA1 レジスタの指定に従い物理アドレス[28:10]を生成します。物理アドレスの[9:5]については仮想アドレスから生成します。物理アドレスの[4:0]は 0 固定です。この物理アドレスで指定される外部メモリから L メモリへブロック転送が行われます。

L メモリ領域への OCBWB 命令が発行されると、LDA0 レジスタまたは LDA1 レジスタの指定に従い物理アドレス[28:10]を生成します。物理アドレスの[9:5]については仮想アドレスから生成します。物理アドレスの[4:0]は 0 固定です。L メモリからこの物理アドレスで指定される外部メモリへブロック転送が行われます。

9.4 L メモリの保護機能

SH-4A では、L メモリに対して、内蔵メモリ制御レジスタ RAMCR の内蔵メモリアクセスモードビット（RMD）と内蔵メモリ保護有効ビット（RP）を使用して以下の保護機能を実現します。

- CPUおよびFPUからのアクセスに対する保護機能

  RAMCR.RMD＝0のとき、ユーザモードでのアクセスをアドレスエラー例外と判定します。

  またMMUCR.AT＝1かつRAMCR.RP＝1のときは、アドレスエラー例外の判定に加えて、P4領域の一部であるLメモリ領域もP0/P3/U0領域と同じようにMMU例外の判定を行います。

  以上を**表9.4**にまとめます。

表 9.4　L メモリへのアクセスに対する保護機能による例外

MMUCR.AT	RAMCR.RP	SR.MD	RAMCR.RMD	必ず発生する例外	起こり得る例外
0	*	0	0	アドレスエラー例外	−
			1	−	−
		1	*	−	−
1	0	0	0	アドレスエラー例外	−
			1	−	−
		1	*	−	−
	1	0	0	アドレスエラー例外	−
			1	−	MMU 例外
		1	*	−	MMU 例外

【記号説明】　*：Don't care

9.5　使用上の注意

9.5.1　ページ競合

同じページに対して異なるバスから同時にアクセス要求が発生した場合は、ページ競合となります。各アクセスは正しく完了しますが、このような競合はメモリアクセスの性能低下を招きます。したがって、できるだけ競合が起こらないようにソフトウェアでの対策を推奨いたします。たとえば各バスごとに異なるページをアクセスすると競合は発生しません。

9.5.2　L メモリのコヒーレンシ

L メモリに命令を配置する場合、L メモリに命令を書き込んだ後、以下のシーケンスを実行してから書き換え後の命令への分岐を行ってください。

- SYNCO
- ICBI @Rn

この場合、ICBI 命令の対象はアドレスエラー例外にならない範囲で任意のアドレスでよく（L メモリのアドレスでもよい）、キャッシュヒット／ミスどちらでも構いません。

9.5.3　スリープモード

スリープモード中は、DMAC などの SuperHyway バスマスタモジュールから本メモリへのアクセスは行えません。

9.6　32 ビットアドレス拡張モード使用時の注意事項

32 ビットアドレス拡張モードでは、LSA0 レジスタの L0SADR ビット、LSA1 レジスタの L1SADR ビット、LDA0 レジスタの L0DADR ビット、LDA1 レジスタの L1DADR ビットがそれぞれ[28:10]の 19 ビットから[31:10]の 22 ビットに拡張されます。

10. 各命令の説明

本章では、サポートする命令の動作を説明します。説明は以下の形式で、命令単位にアルファベット順に行います。

命令の名称 **命令の機能（英文）** **命令の分類**

命令の機能 （遅延分岐命令または特権命令の表示）

書式	動作概略	命令コード	実行ステート	Tビット
アセンブラの入力形式で表示しています。 imm、dispは、数値、式またはシンボルになります。	動作の概略を表示しています。	MSB⇔LSBの順で表示しています。	ノーウェイトのときの値です。	命令実行後のTビットの状態を表示しています。

（1）説明

動作の説明を行います。

（2）注意

命令を使用する上で注意が必要なことを説明します。

（3）動作内容

Cで動作内容を表示しています。

（4）使用例

アセンブラニーモニックで例を示し、命令の実行前後の状態を表示しています。
イタリック字体（例：*.align*）はアセンブラ制御命令であることを示します。アセンブラ制御命令の意味は次のようになります。詳しくは「C/C++コンパイラ、アセンブラ、最適化リンケージエディタ」ユーザーズマニュアルを参照してください。

.org	ロケーションカウンタ設定
.data.W	ワード整数データ確保
.data.1	ロングワード整数データ確保
.sdata	文字列データ確保
.align 2	2バイト境界調整
.align 4	4バイト境界調整
.align 32	32バイト境界調整
.arepeat 16	16回繰り返し展開
.aerpeat 32	32回繰り返し展開
.aendr	回数指定繰り返し展開終了

【注】SHシリーズクロスアセンブラVer1.0では、条件付きアセンブラ機能をサポートしておりません。

（5） 発生する可能性のある例外 （使用例の説明がない場合（4）項になります。）

命令を実行したときに、発生する可能性のある例外を列挙しています。ただし命令TLB多重ヒット例外、命令TLBミス例外、命令TLB保護違反例外、命令アドレスエラーについては全命令で発生する可能性があるため、ここでは省略します。またオーバフロー／アンダーフロー例外に関しては、その詳細な発生条件も説明しています。

10.1 CPU 命令

【注】 CPU 命令のうち、FPU をサポートする CPU 命令、および SH-4A、SH4AL-DSP で機能の一部に差分のある命令は、「10.2 CPU 命令（FPU 関係）」に記載し、それ以外の命令を本節に記載します。

CPU 命令の C を用いた動作内容の説明では、以下の資源と関数を使用します。

```
char     8-bit integer
short    16-bit integer
int      32-bit integer
long     64-bit integer
float    single precision floating point number(32 bits)
double   double precision floating point number(64 bits)
```

データのタイプです。

```
unsigned char  Read_Byte(unsigned long Addr);
unsigned short Read_Word(unsigned long Addr);
unsigned long  Read_Long(unsigned long Addr);
```

アドレス Addr のそれぞれのサイズの内容を返します。2n 番地以外からのワード、4n 番地以外からのロングワードの読み込みはアドレスエラーとして検出します。

```
unsigned char Write_Byte(unsigned long Addr, unsigned long Data);
unsigned short Write_Word(unsigned long Addr, unsigned long Data);
unsigned long Write_Long(unsigned long Addr, unsigned long Data);
```

アドレス Addr にデータ Data をそれぞれのサイズで書き込みます。2n 番地以外へのワード、4n 番地以外へのロングワードの書き込みはアドレスエラーとして検出します。

```
Delay_Slot(unsigned long Addr)
```

アドレス（Addr）のスロット命令に実行を移します。

```
unsigned long R[16];
unsigned long SR,GBR,VBR;
unsigned long MACH,MACL,PR;
unsigned long PC;
```

各レジスタの本体

```
struct SR0 {
  unsigned long dummy0:22;
  unsigned long    M0:1;
  unsigned long    Q0:1;
  unsigned long    I0:4;
  unsigned long dummy1:2;
  unsigned long    S0:1;
  unsigned long    T0:1;
};
```

SR の構造の定義

```
#define M ((*(struct SR0 *)(&SR)).M0)
#define Q ((*(struct SR0 *)(&SR)).Q0)
#define S ((*(struct SR0 *)(&SR)).S0)
#define T ((*(struct SR0 *)(&SR)).T0)
```

SR 内ビットの定義

```
Error( char *er );
```

エラー表示関数

10.1.1 ADD　　ADD binary　　算術演算命令

2 進加算

書式	動作概略	命令コード	実行ステート	T ビット
ADD　Rm,Rn	Rn+Rm→Rn	0011nnnnmmmm1100	1	—
ADD　#imm,Rn	Rn+imm→Rn	0111nnnniiiiiiii	1	—

(1) 説明

汎用レジスタ Rn の内容と Rm とを加算し、結果を Rn に格納します。

汎用レジスタ Rn と 8 ビットのイミディエイトデータとの加算も可能です。

8 ビットのイミディエイトデータは 32 ビットに符号拡張しますので減算との兼用が可能です。

(2) 注意

特になし

(3) 動作内容

```
ADD(long m, long n)     /* ADD Rm,Rn */
{
    R[n] += R[m];
    PC += 2;
}

ADDI(long i, long n)    /* ADD #imm,Rn */
{
    if((I & 0x80) == 0)
         R[n] += (0x000000FF & (long)i);
    else R[n] += (0xFFFFFF00 | (long)i);
    PC += 2;
}
```

(4) 使用例

```
ADD  R0,R1          ;実行前 R0=H'7FFFFFFF,R1=H'00000001
                    ;実行後 R1=H'80000000
ADD  #H'01,R2       ;実行前 R2=H'00000000
                    ;実行後 R2=H'00000001
ADD  #H'FE,R3       ;実行前 R3=H'00000001
                    ;実行後 R3=H'FFFFFFFF
```

10.1.2 ADDC ADD with Carry 算術演算命令

キャリ付き 2 進加算

書式	動作概略	命令コード	実行ステート	T ビット
ADDC Rm,Rn	Rn+Rm+T→Rn, キャリ→T	0011nnnnmmmm1110	1	キャリ

(1) 説明

汎用レジスタ Rn の内容と Rm と T ビットを加算し、結果を Rn に格納します。演算の結果によってキャリを T ビットに反映します。32 ビットを超える加算を行うとき使用します。

(2) 注意

特になし

(3) 動作内容

```
ADDC(long m, long n)    /* ADDC Rm,Rn */
{
     unsigned long tmp0,tmp1;

     tmp1 = R[n] + R[m];
     tmp0 = R[n];
     R[n] = tmp1 + T;
     if(tmp0>tmp1) T = 1;
     else T = 0;
     if(tmp1>R[n]) T = 1;
     PC += 2;
}
```

(4) 使用例

```
CLRT              ;R0:R1(64 ビット)+R2:R3(64 ビット)=R0:R1(64 ビット)
ADDC  R3,R1       ;実行前 T=0,R1=H'00000001,R3=H'FFFFFFFF
                  ;実行後 T=1,R1=H'00000000
ADDC  R2,R0       ;実行前 T=1,R0=H'00000000,R2=H'00000000
                  ;実行後 T=0,R0=H'00000001
```

10.1.3 ADDV ADD with (Vflag) overflow check 算術演算命令

オーバフロー付き
2 進加算

書式	動作概略	命令コード	実行ステート	T ビット
ADDV Rm,Rn	Rn+Rm→Rn, オーバフロー→T	0011nnnnmmmm1111	1	オーバフロー

(1) 説明

汎用レジスタ Rn の内容と Rm とを加算し、結果を Rn に格納します。オーバフローが発生すると、T ビットをセットします。

(2) 注意

特になし

(3) 動作内容

```
ADDV(long m, long n)    /* ADDV Rm,Rn */
{
     long dest,src,ans;

     if((long)R[n]>=0) dest = 0;
     else dest = 1;
     if((long)R[m]>=0) src = 0;
     else src = 1;
     src += dest;
     R[n] += R[m];
     if((long)R[n]>=0) ans = 0;
     else ans = 1;
     ans += dest;
     if(src==0 || src==2) {
          if(ans==1) T = 1;
          else T = 0;
     }
     else T = 0;
     PC += 2;
}
```

(4) 使用例

```
ADDV  R0,R1        ;実行前 R0=H'00000001,R1=H'7FFFFFFE,  T=0
                   ;実行後 R1=H'7FFFFFFF, T=0
ADDV  R0,R1        ;実行前 R0=H'00000002,R1=H'7FFFFFFE,  T=0
                   ;実行後 R1=H'80000000, T=1
```

10.1.4 AND AND logical 論理演算命令

論理積演算

書式	動作概略	命令コード	実行ステート	T ビット
AND Rm,Rn	Rn&Rm→Rn	0010nnnnmmmm1001	1	—
AND #imm,R0	R0&imm→R0	11001001iiiiiiii	1	—
AND.B #imm,@(R0,GBR)	(R0+GBR)&imm→(R0+GBR)	11001101iiiiiiii	3	—

(1) 説明

汎用レジスタ Rn の内容と Rm の論理積をとり、結果を Rn に格納します。

汎用レジスタ R0 とゼロ拡張した 8 ビットのイミディエイトデータとの論理積、もしくはインデックス付き GBR 間接アドレッシングモードで 8 ビットのメモリと 8 ビットのイミディエイトデータとの論理積が可能です。

(2) 注意

AND #imm,R0 では演算の結果、R0 の上位 24 ビットは常にクリアされます。

(3) 動作内容

```
AND(long m, long n)    /* AND Rm,Rn */
{
     R[n] &= R[m];
     PC += 2;
}

ANDI(long i)    /* AND #imm,R0 */
{
     R[0] &= (0x000000FF & (long)i);
     PC += 2;
}

ANDM(long i)    /* AND.B #imm,@(R0,GBR) */
{
     long temp;

     temp = (long)Read_Byte(GBR+R[0]);
     temp &= (0x000000FF & (long)i);
     Write_Byte(GBR+R[0],temp);
```

```
        PC += 2;
    }
```

(4) 使用例

```
    AND   R0,R1                ;実行前 R0=H'AAAAAAAA,R1=H'55555555
                               ;実行後 R1=H'00000000
    AND   #H'0F,R0             ;実行前 R0=H'FFFFFFFF
                               ;実行後 R0=H'0000000F
    AND.B #H'80,@(R0,GBR)      ;実行前 @(R0,GBR)=H'A5
                               ;実行後 @(R0,GBR)=H'80
```

(5) 発生する可能性がある例外

AND.B 命令のみ以下の例外が発生する可能性があります。

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- 初期ページ書き込み例外
- データアドレスエラー

例外処理において、本命令によるデータアクセスは、バイトロード、バイトストアとして扱われます。

10.1.5 BF Branch if False 分岐命令

条件分岐

書式	動作概略	命令コード	実行ステート	T ビット
BF label	T=0 のとき PC+4+disp×2→PC, T=1 のとき nop	10001011dddddddd	1	—

(1) 説明

T ビットを参照する条件付き分岐命令です。T=1 のときは分岐しません。逆に T=0 のとき、分岐します。分岐先はアドレス（PC+4+ディスプレースメント×2）です。PC ソース値は BF の命令アドレスです。8 ビットディスプレースメントは符号拡張後2倍しますので､分岐先との相対距離は-256バイトから+254バイトの範囲になります。

(2) 注意

分岐先に届かないときは BRA、JMP 命令などとの組み合わせで対応する必要があります。

(3) 動作内容

```
BF(int d)    /* BF disp */
{
    int disp;

    if((d&0x80)==0)
         disp = (0x000000FF & d);
    else disp = (0xFFFFFF00 | d);
    if(T==0)
         PC = PC + 4 + (disp<<1);
    else PC += 2;
}
```

(4) 使用例

```
    CLRT              ;常に T=0
    BT   TRGET_T      ;T=0 のため分岐しません。
    BF   TRGET_F      ;T=0 のため TRGET_F へ分岐します。
    NOP               ;
    NOP               ;
TRGET_F:              ;←BF 命令の分岐先
```

(5) 発生する可能性がある例外

- スロット不当命令例外

10.1.6 BF/S Branch if False with delay Slot 分岐命令

遅延付き条件分岐 遅延分岐命令

書式	動作概略	命令コード	実行ステート	T ビット
BF/S label	T=0 のとき PC+4+disp×2→PC, T=1 のとき nop	10001111dddddddd	1	—

(1) 説明

T ビットを参照する遅延付き条件分岐命令です。T=1 のとき、次の命令を実行し、分岐しません。T=0 のとき、次の命令を実行した後で分岐します。

分岐先はアドレス（PC+4+ディスプレースメント×2）です。PC ソース値は BF/S の命令アドレスです。8 ビットディスプレースメントは符号拡張後 2 倍しますので、分岐先との相対距離は-256 バイトから+254 バイトの範囲になります。

(2) 注意

遅延分岐命令ですので、分岐成立時には本命令の直後の命令を先に実行してから分岐先の命令を実行します。

本命令と直後の命令との間には、割り込みを受け付けません。

直後の命令が、分岐命令の場合、それをスロット不当命令として認識します。

遅延分岐命令直後の遅延スロットに本命令が配置されたときには、スロット不当命令として認識します。

分岐先に届かないときは BRA、JMP 命令などとの組み合わせで対応する必要があります。

(3) 動作内容

```
BFS(int d)    /* BFS disp */
{
      int disp;
      unsigned int temp;

      temp = PC;
      if((d&0x80)==0)
            disp = (0x000000FF & d);
      else disp = (0xFFFFFF00 | d);
      if(T==0)
            PC = PC + 4 + (disp<<1);
      else PC += 4;
      Delay_Slot(temp+2);
}
```

(4) 使用例

```
      CLRT                ;常に T=0
      BT/S TRGET_T        ;T=0 のため分岐しません。
      NOP                 ;
      BF/S TRGET_F        ;T=0 のため TRGET-F に分岐します。
      ADD  R0,R1          ;分岐に先立ち実行します。
      NOP                 ;
  TRGET_F:                ;←BF/S 命令の分岐先
```

(5) 発生する可能性がある例外

- スロット不当命令例外

10.1.7 BRA BRAnch 分岐命令

無条件分岐 **遅延分岐命令**

書式	動作概略	命令コード	実行ステート	T ビット
BRA label	PC+4+disp×2→PC	1010ddddddddddd	1	—

(1) 説明

無条件の遅延分岐命令です。分岐先はアドレス（PC+4+ディスプレースメント×2）です。PC ソース値は BRA の命令アドレスです。12 ビットディスプレースメントは符号拡張後 2 倍しますので、分岐先との相対距離は-4096 バイトから+4094 バイトの範囲になります。分岐先に届かないときは、JMP 命令によってこの分岐が可能になります。

(2) 注意

遅延分岐命令ですので、本命令の直後の命令を先に実行してから、分岐先の命令を実行します。

本命令と直後の命令との間には、割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

(3) 動作内容

```
BRA(int d)    /* BRA disp */
{
      int disp;
      unsigned int temp;

      temp = PC;
      if((d&0x800)==0)
            disp = (0x00000FFF & d);
      else disp = (0xFFFFF000 | d);
      PC = PC + 4 + (disp<<1);
      Delay_Slot(temp+2);
}
```

(4) 使用例

```
          BRA  TRGET     ;TRGET へ分岐します。
          ADD  R0,R1     ;分岐に先立ち ADD を実行します。
          NOP            ;
TRGET:                   ;←BRA 命令の分岐先
```

(5) 発生する可能性がある例外

- スロット不当命令例外

10.1.8 BRAF BRAnch Far 分岐命令

無条件分岐 **遅延分岐命令**

書式	動作概略	命令コード	実行ステート	T ビット
BRAF Rn	PC+4+Rn→PC	0000nnnn00100011	1	—

(1) 説明

無条件の遅延分岐命令です。分岐先はアドレス（PC+4+Rn）です。分岐先アドレスは PC に 4 と汎用レジスタ Rn の内容の 32 ビットを加えたアドレスです。

(2) 注意

遅延分岐命令ですので、本命令の直後の命令を先に実行してから、分岐先の命令を実行します。

本命令と直後の命令との間には、割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

(3) 動作内容

```
BRAF(int n)   /* BRAF Rn */
{
    unsigned int temp;

    temp = PC;
    PC = PC + 4 + R[n];
    Delay_Slot(temp+2);
}
```

(4) 使用例

```
    MOV.L #(TRGET-BRAF_PC),R0   ;ディスプレースメントを設定します。
    BRAF  R0                    ;TRGET へ分岐します。
    ADD   R0,R1                 ;分岐に先立ち ADD を実行します。
BRAF_PC:                        ;
    NOP
TRGET:                          ;←BRAF 命令の分岐先
```

(5) 発生する可能性がある例外

- スロット不当命令例外

10.1.9 BT Branch if True 分岐命令

条件分岐

書式	動作概略	命令コード	実行ステート	T ビット
BT label	T=1 のとき PC+4+disp×2→PC, T=0 のとき nop	10001001dddddddd	1	—

(1) 説明

T ビットを参照する条件付き分岐命令です。T=1 のとき、分岐します。逆に T=0 のとき、分岐しません。

分岐先はアドレス（PC+4+ディスプレースメント×2）です。PC ソース値は BT の命令アドレスです。8 ビットディスプレースメントは符号拡張後 2 倍しますので、分岐先との相対距離は-256 バイトから+254 バイトの範囲になります。

(2) 注意

分岐先に届かないときは BRA、JMP 命令などとの組み合わせで対応する必要があります。

(3) 動作内容

```
BT(int d)    /* BT disp */
{
    int disp;

    if((d&0x80)==0)
         disp = (0x000000FF & d);
    else disp = (0xFFFFFF00 | d);
    if(T==1)
         PC = PC + 4 + (disp<<1);
    else PC += 2;
}
```

(4) 使用例

```
    SETT              ;常に T=1
    BF   TRGET_F      ;T=1 のため分岐しません。
    BT   TRGET_T      ;T=1 のため TRGET_T へ分岐します。
    NOP               ;
    NOP               ;
TRGET_T:              ;←BT 命令の分岐先
```

(5) 発生する可能性がある例外

- スロット不当命令例外

10.1.10 BT/S　Branch if True with delay Slot　分岐命令

遅延付き条件分岐　遅延分岐命令

書式	動作概略	命令コード	実行ステート	T ビット
BT/S　label	T=1 のとき PC+4+disp×2→PC, T=0 のとき nop	10001101dddddddd	1	―

(1) 説明

T ビットを参照する遅延付き条件分岐命令です。T=1 のとき、分岐します。T=0 のとき、分岐しません。

PC ソース値は BT/S の命令アドレスです。8 ビットディスプレースメントは符号拡張後 2 倍しますので、分岐先との相対距離は-256 バイトから+254 バイトの範囲になります。

(2) 注意

遅延分岐命令ですので、分岐成立時には本命令の直後の命令を先に実行してから分岐先の命令を実行します。

本命令と直後の命令との間には、割り込みを受け付けません。

直後の命令が、分岐命令の場合、それをスロット不当命令として認識します。分岐先に届かないときは BRA、JMP 命令などとの組み合わせで対応する必要があります。

(3) 動作内容

```
BTS(int d)    /* BTS disp */
{
    int disp;
    unsigned temp;

    temp = PC;
    if((d&0x80)==0)
         disp = (0x000000FF & d);
    else disp = (0xFFFFFF00 | d);
    if(T==1)
         PC = PC + 4 + (disp<<1);
    else PC += 4;
    Delay_Slot(temp+2);
}
```

(4) 使用例

```
    SETT                ;常に T=1
    BF/S TRGET_F        ;T=1 のため分岐しません。
    NOP                 ;
    BT/S TRGET_T        ;T=1 のため TRGET_T に分岐します。
    ADD R0,R1           ;分岐に先立ち実行します。
    NOP                 ;
TRGET_T:                ;←BT/S 命令の分岐先
```

(5) 発生する可能性がある例外

- スロット不当命令例外

10.1.11 CLRMAC CLeaR MAC register システム制御命令

MAC レジスタのクリア

書式	動作概略	命令コード	実行ステート	T ビット
CLRMAC	0→MACH,MACL	0000000000101000	1	—

(1) 説明

MACH、MACL レジスタをクリアします。

(2) 注意

特になし

(3) 動作内容

```
CLRMAC( )    /* CLRMAC */
{
     MACH = 0;
     MACL = 0;
     PC += 2;
}
```

(4) 使用例

```
     CLRMAC                ;MAC レジスタをクリアして初期化します。
     MAC.W  @R0+,@R1+      ;積和演算
     MAC.W  @R0+,@R1+      ;
```

10.1.12 CLRS CLeaR Sbit システム制御命令

S ビットのクリア

書式	動作概略	命令コード	実行ステート	T ビット
CLRS	0→S	0000000001001000	1	−

(1) 説明

S ビットを 0 にクリアします。

(2) 注意

特になし

(3) 動作内容

```
CLRS( )    /* CLRS */
{
     S = 0;
     PC += 2;
}
```

(4) 使用例

```
     CLRS      ;実行前 S=1
               ;実行後 S=0
```

10.1.13 CLRT CLeaR Tbit システム制御命令

Tビットのクリア

書式	動作概略	命令コード	実行ステート	Tビット
CLRT	0→T	0000000000001000	1	0

(1) 説明

Tビットをクリアします。

(2) 注意

特になし

(3) 動作内容

```
CLRT( )    /* CLRT */
{
     T = 0;
     PC += 2;
}
```

(4) 使用例

```
CLRT          ;実行前 T=1
              ;実行後 T=0
```

10.1.14 CMP/cond　　CoMPare conditionally　　算術演算命令

比較

書式		動作概略	命令コード	実行ステート	T ビット
CMP/EQ	Rm,Rn	Rn=Rm のとき 1→T、 それ以外のとき 0→T	0011nnnnmmmm0000	1	比較結果
CMP/GE	Rm,Rn	有符号で Rn≧Rm のとき 1→T、 それ以外のとき 0→T	0011nnnnmmmm0011	1	比較結果
CMP/GT	Rm,Rn	有符号で Rn>Rm のとき 1→T、 それ以外のとき 0→T	0011nnnnmmmm0111	1	比較結果
CMP/HI	Rm,Rn	無符号で Rn>Rm のとき 1→T、 それ以外のとき 0→T	0011nnnnmmmm0110	1	比較結果
CMP/HS	Rm,Rn	無符号で Rn≧Rm のとき 1→T、 それ以外のとき 0→T	0011nnnnmmmm0010	1	比較結果
CMP/PL	Rn	Rn>0 のとき 1→T、 それ以外のとき 0→T	0100nnnn00010101	1	比較結果
CMP/PZ	Rn	Rn≧0 のとき 1→T、 それ以外のとき 0→T	0100nnnn00010001	1	比較結果
CMP/STR	Rm,Rn	いずれかのバイトが等しいとき 1→T、 それ以外のとき 0→T	0010nnnnmmmm1100	1	比較結果
CMP/EQ	#imm,R0	R0=imm のとき 1→T、 それ以外のとき 0→T	10001000iiiiiiii	1	比較結果

(1) 説明

汎用レジスタ Rn と Rm とを比較し、その結果、指定された条件(cond)が成立していると T ビットをセットします。条件が不成立のときは T ビットをクリアします。Rn の内容は変化しません。9 条件が指定できます。PZ と PL の 2 条件については Rn と 0 との比較になります。

EQ の条件については符号拡張した 8 ビットのイミディエイトデータと R0 との比較も可能です。R0 の内容は変化しません。

ニーモニック	説　　明
CMP/EQ Rm,Rn	Rn=Rm のとき T=1
CMP/GE Rm,Rn	有符号値として Rn≧Rm のとき T=1
CMP/GT Rm,Rn	有符号値として Rn>Rm のとき T=1
CMP/HI Rm,Rn	無符号値として Rn>Rm のとき T=1
CMP/HS Rm,Rn	無符号値として Rn≧Rm のとき T=1
CMP/PL Rn	Rn>0 のとき T=1
CMP/PZ Rn	Rn≧0 のとき T=1
CMP/STR Rm,Rn	いずれかのバイトが等しいとき T=1
CMP/EQ #imm,R0	R0=imm のとき T=1

(2) 注意

特になし

(3) 動作内容

```
CMPEQ(long m, long n)    /* CMP_EQ Rm,Rn */
{
    if(R[n]==R[m]) T = 1;
    else T = 0;
    PC += 2;
}
CMPGE(long m, long n)    /* CMP_GE Rm,Rn */
{
    if((long)R[n]>=(long)R[m]) T = 1;
    else T = 0;
    PC += 2;
}

CMPGT(long m, long n)    /* CMP_GT Rm,Rn */
{
    if((long)R[n]>(long)R[m]) T = 1;
    else T = 0;
    PC += 2;
}

CMPHI(long m, long n)    /* CMP_HI Rm,Rn */
{
    if((unsigned long)R[n] > (unsigned long)R[m]) T = 1;
```

```
    else T = 0;
    PC += 2;
}

CMPHS(long m, long n)    /* CMP_HS Rm,Rn */
{
    if((unsigned long)R[n]>=(unsigned long)R[m]) T = 1;
    else T = 0;
    PC += 2;
}

CMPPL(long n)            /* CMP_PL Rn */
{
    if((long)R[n]>0) T = 1;
    else T = 0;
    PC += 2;
}

CMPPZ(long n)    /* CMP_PZ Rn */
{
    if((long)R[n]>=0) T = 1;
    else T = 0;
    PC += 2;
}

CMPSTR(long m, long n)    /* CMP_STR Rm,Rn */
{
    unsigned long temp;
    long HH,HL,LH,LL;

    temp = R[n] ^ R[m];
    HH = (temp & 0xFF000000) >> 24;
    HL = (temp & 0x00FF0000) >> 16;
    LH = (temp & 0x0000FF00) >> 8;
    LL = temp & 0x000000FF;
    HH = HH && HL && LH && LL;
    if(HH==0) T = 1;
    else T = 0;
```

```
    PC += 2;
}

CMPIM(long i)    /* CMP_EQ #imm,R0 */
{
    long imm;

    if((i&0x80)==0) imm = (0x000000FF & (long i));
    else imm = (0xFFFFFF00 | (long i));
    if(R[0]==imm) T = 1;
    else T = 0;
    PC += 2;
}
```

(4) 使用例

```
CMP/GE  R0,R1       ;R0=H'7FFFFFFF,R1=H'80000000
BT      TRGET_T     ;T=0 なので分岐しません。
CMP/HS  R0,R1       ;R0=H'7FFFFFFF,R1=H'80000000
BT      TRGET_T     ;T=1 なので分岐します。
CMP/STR R2,R3       ;R2="ABCD",R3="XYCZ"
BT      TRGET_T     ;T=1 なので分岐します。
```

10.1.15 DIV0S DIVide(step0) as Signed 算術演算命令

符号付き除算の
初期化

書式	動作概略	命令コード	実行ステート	T ビット
DIV0S Rm,Rn	Rn の MSB→Q, Rm の MSB→M, M^Q→T	0010nnnnmmmm0111	1	計算結果

(1) 説明

符号付き除算の初期設定をします。本命令に続けて 1 桁分の除算をする DIV1 命令などを組み合わせて、繰り返し除算を行い、商を求めます。詳しくは DIV1 の説明を参照してください。

(2) 注意

特になし

(3) 動作内容

```
DIV0S(long m, long n)    /* DIV0S Rm,Rn */
{
    if((R[n] & 0x80000000)==0) Q = 0;
    else Q = 1;
    if((R[m] & 0x80000000)==0) M = 0;
    else M = 1;
    T = !(M==Q);
    PC += 2;
}
```

(4) 使用例

DIV1 の使用例を参照してください。

10.1.16 DIV0U　　DIVide (step0) as Unsigned　　算術演算命令

符号なし除算の
初期化

書式	動作概略	命令コード	実行ステート	T ビット
DIV0U	0→M/Q/T	0000000000011001	1	0

(1) 説明

符号なし除算の初期設定をします。本命令に続けて 1 桁分の除算をする DIV1 命令などを組み合わせて、繰り返し除算を行い、商を求めます。詳しくは DIV1 の説明を参照してください。

(2) 注意

特になし

(3) 動作内容

```
DIV0U( )        /* DIV0U */
{
    M = Q = T = 0;
    PC += 2;
}
```

(4) 使用例

DIV1 の使用例を参照してください。

10.1.17 DIV1 DIVide 1 step 算術演算命令

除算

書式	動作概略	命令コード	実行ステート	T ビット
DIV1 Rm,Rn	1 ステップ除算 (Rn÷Rm)	0011nnnnmmmm0100	1	計算結果

(1) 説明

汎用レジスタ Rn の 32 ビットの内容(被除数)を Rm の内容(除数)で 1 桁分の除算(1 ステップ除算)を実行する命令です。本命令単独でまたは他の命令と組み合わせて繰り返し実行し商を求めます。この繰り返し中は、指定したレジスタと M、Q、T ビットを書き替えないでください。

1 ステップ除算とは、被除数を左に 1 ビットシフトし、それから除数を減算し、結果の正負によって商のビットを Q ビットに反映するという処理を実行します。

割り算で余りを求めるには、DIV1 命令を用いて商を求めた後、

（余り）＝（被除数）－（除数）×（商）

として求めてください。

ゼロ除算とオーバフローの検出は用意していません。除算の前にゼロ除算とオーバフロー除算をチェックしてください。剰余の演算は用意していません。除数と求められた商との積を求めて、被除数から減算して剰余を求めてください。

最初に、DIV0S または DIV0U で初期設定します。DIV1 を除数のビット数分繰り返します。商が 17 ビット以上必要なとき、ROTCL を DIV1 の前に置きます。詳しい 除算のシーケンスは使用例を参考にしてください。

(2) 注意

特になし

(3) 動作内容

```
DIV1(long m, long n)    /* DIV1 Rm,Rn */
{
   unsigned long tmp0, tmp2;
   unsigned char old_q, tmp1;

   old_q = Q;
   Q = (unsigned char)((0x80000000 & R[n]) != 0);
   tmp2 = R[m];
   R[n] <<= 1;
   R[n] |= (unsigned long)T;
```

```
switch(old_q){
  case 0:  switch(M){
             case 0:  tmp0 = R[n];
                      R[n] -= tmp2;
                      tmp1 = (R[n]>tmp0);
                      switch(Q){
                        case 0:  Q = tmp1;
                                 break;
                        case 1:  Q = (unsigned char)(tmp1==0);
                                 break;
                      }
                      break;
             case 1:  tmp0 = R[n];
                      R[n] += tmp2;
                      tmp1 = (R[n]<tmp0);
                      switch(Q){
                         case 0:  Q = (unsigned char)(tmp1==0);
                                  break;
                         case 1:  Q = tmp1;
                                  break;
                      }
                      break;
           }
           break;
  case 1:  switch(M){
             case 0:  tmp0 = R[n];
                      R[n] += tmp2;
                      tmp1 = (R[n]<tmp0);
                      switch(Q){
                        case 0:  Q = tmp1;
                                 break;
                        case 1:  Q = (unsigned char)(tmp1==0);
                                 break;
                      }
                      break;
             case 1:  tmp0 = R[n];
                      R[n] -= tmp2;
```

```
                        tmp1 = (R[n]>tmp0);
                        switch(Q){
                          case 0:  Q = (unsigned char)(tmp1==0);
                                   break;
                          case 1:  Q = tmp1;
                                   break;
                        }
                        break;
                }
                break;
    }
    T = (Q==M);
    PC += 2;
}
```

(4) 使用例

- **例1**

```
                                ;R1(32 ビット)÷R0(16 ビット)=R1(16 ビット):符号なし
SHLL16      R0                  ;除数を上位 16 ビット、下位 16 ビットを 0 に設定
TST         R0,R0               ;ゼロ除算チェック
BT          ZERO_DIV            ;
CMP/HS      R0,R1               ;オーバフローチェック
BT          OVER_DIV            ;
DIV0U                           ;フラグの初期化
.arepeat    16                  ;
DIV1        R0,R1               ;16 回繰り返し
.aendr                          ;
ROTCL       R1                  ;
EXTU.W      R1,R1               ;R1=商
```

- **例2**

```
                                ;R1:R2(64 ビット)÷R0(32 ビット)=R2(32 ビット):符号なし
TST         R0,R0               ;ゼロ除算チェック
BT          ZERO_DIV            ;
CMP/HS      R0,R1               ;オーバフローチェック
BT          OVER_DIV            ;
DIV0U                           ;フラグの初期化
.arepeat    32                  ;
```

```
ROTCL     R2          ;32 回繰り返し
DIV1      R0,R1       ;
.aendr                ;
ROTCL     R2          ;R2=商
```

- 例3

```
                      ;R1(16 ビット)÷R0(16 ビット)=R1(16 ビット):符号付き
SHLL16    R0          ;除数を上位 16 ビット、下位 16 ビットを 0 に設定
EXTS.W    R1,R1       ;被除数は符号拡張して 32 ビット
XOR       R2,R2       ;R2=0
MOV       R1,R3       ;
ROTCL     R3          ;
SUBC      R2,R1       ;被除数が負のとき、-1 する。
DIV0S     R0,R1       ;フラグの初期化
.arepeat  16          ;
DIV1      R0,R1       ;16 回繰り返し
.aendr                ;
EXTS.W    R1,R1       ;
ROTCL     R1          ;R1=商(1 の補数表現)
ADDC      R2,R1       ;商の MSB が 1 のとき、+1 して 2 の補数表現に変換
EXTS.W    R1,R1       ;R1=商(2 の補数表現)
```

- 例4

```
                      ;R2(32 ビット)÷R0(32 ビット)=R2(32 ビット):符号付き
MOV       R2,R3       ;
ROTCL     R3          ;
SUBC      R1,R1       ;被除数は符号拡張して 64 ビット(R1:R2)
XOR       R3,R3       ;R3=0
SUBC      R3,R2       ;被除数が負のとき、-1 して 1 の補数表現に変換
DIV0S     R0,R1       ;フラグの初期化
.arepeat  32          ;
ROTCL     R2          ;32 回繰り返し
DIV1      R0,R1       ;
.aendr                ;
ROTCL     R2          ;R2=商(1 の補数表現)
ADDC      R3,R2       ;商の MSB が 1 のとき、+1 して 2 の補数表現に変換
                      ;R2=商(2 の補数表現)
```

10.1.18 DMULS.L Double-length MULtiply as Signed 算術演算命令

符号付き倍精度乗算

書式	動作概略	命令コード	実行ステート	T ビット
DMULS.L Rm,Rn	符号付きで Rn×Rm→MAC	0011nnnnmmmm1101	2	—

(1) 説明

汎用レジスタ Rn の内容と Rm を 32 ビットで乗算し、結果の 64 ビットを MACH レジスタと MACL レジスタに格納します。演算は符号付き算術演算で行います。

(2) 注意

特になし

(3) 動作内容

```
DMULS(long m, long n)  /* DMULS.L Rm,Rn */
{
    unsigned long RnL,RnH,RmL,RmH,Res0,Res1,Res2;
    unsigned long temp0,temp1,temp2,temp3;
    long tempm,tempn,fnLmL;

    tempn = (long)R[n];
    tempm = (long)R[m];
    if(tempn<0) tempn = 0-tempn;
    if(tempm<0) tempm = 0-tempm;
    if((long)(R[n]^R[m])<0) fnLmL = -1;
    else fnLmL = 0;

    temp1 = (unsigned long)tempn;
    temp2 = (unsigned long)tempm;

    RnL = temp1 & 0x0000FFFF;
    RnH = (temp1>>16) & 0x0000FFFF;
    RmL = temp2 & 0x0000FFFF;
    RmH = (temp2>>16) & 0x0000FFFF;
```

```
    temp0 = RmL*RnL;
    temp1 = RmH*RnL;
    temp2 = RmL*RnH;
    temp3 = RmH*RnH;

    Res2 = 0;
    Res1 = temp1 + temp2;
    if(Res1<temp1) Res2 += 0x00010000;
    temp1 = (Res1<<16) & 0xFFFF0000;
    Res0 = temp0 + temp1;
    if(Res0<temp0) Res2++;

    Res2 = Res2 + ((Res1>>16) & 0x0000FFFF) + temp3;

    if(fnLmL<0) {
         Res2 = ~Res2;
         if(Res0==0) Res2++;
         else  Res0 = (~Res0)+1;
    }

    MACH = Res2;
    MACL = Res0;
    PC += 2;
}
```

(4) 使用例

```
DMULS.L   R0,R1          ;実行前 R0=H'FFFFFFFE,R1=H'00005555
                         ;実行後 MACH=H'FFFFFFFF,MACL=H'FFFF5556
STS       MACH,R0        ;演算結果(上位)を得る
STS       MACL,R1        ;演算結果(下位)を得る
```

10.1.19 DMULU.L Double-length MULtiply as Unsigned 算術演算命令

符号なし倍精度乗算

書式	動作概略	命令コード	実行ステート	T ビット
DMULU.L Rm,Rn	符号なしで Rn×Rm→MAC	0011nnnnmmmm0101	2	—

(1) 説明

汎用レジスタ Rn の内容と Rm を 32 ビットで乗算し、結果の 64 ビットを MACH レジスタと MACL レジスタに格納します。演算は符号なし算術演算で行います。

(2) 注意

特になし

(3) 動作内容

```
DMULU(long m, long n)  /* DMULU.L Rm,Rn */
{
    unsigned long RnL,RnH,RmL,RmH,Res0,Res1,Res2;
    unsigned long temp0,temp1,temp2,temp3;

    RnL = R[n] & 0x0000FFFF;
    RnH = (R[n]>>16) & 0x0000FFFF;

    RmL =R [m] & 0x0000FFFF;
    RmH = (R[m]>>16) & 0x0000FFFF;

    temp0 = RmL*RnL;
    temp1 = RmH*RnL;
    temp2 = RmL*RnH;
    temp3 = RmH*RnH;

    Res2=0
    Res1 = temp1 + temp2;
    if(Res1<temp1) Res2 += 0x00010000;

    temp1 = (Res1<<16)&0xFFFF0000;
```

```
    Res0 = temp0 + temp1;
    if(Res0<temp0) Res2++;

    Res2 = Res2 + ((Res1>>16)&0x0000FFFF) + temp3;

    MACH = Res2;
    MACL = Res0;
    PC += 2;
}
```

(4) 使用例

```
DMULU.L     R0,R1           ;実行前 R0=H'FFFFFFFE,R1=H'00005555
                            ;実行後 MACH=H'00005554,MACL=H'FFFF5556
STS         MACH,R0         ;演算結果(上位)を得る
STS         MACL,R1         ;演算結果(下位)を得る
```

10.1.20 DT　Decrement and Test　算術演算命令

デクリメント

テスト

書式	動作概略	命令コード	実行ステート	T ビット
DT　Rn	Rn-1→Rn,Rn が 0 のとき 1→T Rn が 0 以外のとき 0→T	0100nnnn00010000	1	比較結果

(1) 説明

汎用レジスタ Rn の内容を 1 デクリメントして、結果を 0(ゼロ)と比較します。結果が 0 のとき T ビットを 1 にセットします。結果が 0 以外のとき、T ビットを 0 にセットします。

(2) 注意

特になし

(3) 動作内容

```
DT(long n)  /* DT Rn */
{
    R[n]--;
    if(R[n]==0) T = 1;
    else T = 0;
    PC += 2;
}
```

(4) 使用例

```
    MOV #4,R5          ;ループ回数を設定します。
LOOP:
    ADD R0,R1          ;
    DT  R5             ;R5 の値をデクリメントし、0 になったかどうか判定します。
    BF  LOOP           ;T=0 なら LOOP へ分岐します(この例では 4 回ループします)。
```

10.1.21 EXTS EXTend as Signed 算術演算命令

符号拡張

書式	動作概略	命令コード	実行ステート	T ビット
EXTS.B Rm,Rn	Rm をバイトから符号拡張→Rn	0110nnnnmmmm1110	1	—
EXTS.W Rm,Rn	Rm をワードから符号拡張→Rn	0110nnnnmmmm1111	1	—

(1) 説明

汎用レジスタ Rm の内容を符号拡張して、結果を Rn に格納します。

バイト指定のとき、Rn のビット 8 からビット 31 に Rm のビット 7 の内容を転送します。ワード指定のとき、Rn のビット 16 からビット 31 に Rm のビット 15 の内容を転送します。

(2) 注意

特になし

(3) 動作内容

```
EXTSB(long m, long n)          /* EXTS.B Rm,Rn */
{
    R[n] = R[m];
    if((R[m]&0x00000080)==0) R[n] &= 0x000000FF;
    else R[n] |= 0xFFFFFF00;
    PC += 2;
}
EXTSW(long m, long n)          /* EXTS.W Rm,Rn */
{
    R[n] = R[m];
    if((R[m]&0x00008000)==0) R[n] &= 0x0000FFFF;
    else R[n] |= 0xFFFF0000;
    PC += 2;
}
```

(4) 使用例

```
EXTS.B      R0,R1     ;実行前 R0=H'00000080
                      ;実行後 R1=H'FFFFFF80
EXTS.W      R0,R1     ;実行前 R0=H'00008000
                      ;実行後 R1=H'FFFF8000
```

10.1.22 EXTU EXTend as Unsigned 算術演算命令

ゼロ拡張

書式	動作概略	命令コード	実行ステート	T ビット
EXTU.B Rm,Rn	Rm をバイトからゼロ拡張→Rn	0110nnnnmmmm1100	1	—
EXTU.W Rm,Rn	Rm をワードからゼロ拡張→Rn	0110nnnnmmmm1101	1	—

(1) 説明

汎用レジスタ Rm の内容をゼロ拡張して、結果を Rn に格納します。

バイト指定のとき、Rn のビット 8 からビット 31 に 0 を転送します。ワード指定のとき、Rn のビット 16 からビット 31 に 0 を転送します。

(2) 注意

特になし

(3) 動作内容

```
EXTUB(long m, long n)          /* EXTU.B Rm,Rn */
{
    R[n] = R[m];
    R[n] &= 0x000000FF;
    PC += 2;
}

EXTUW(long m, long n)          /* EXTU.W Rm,Rn */
{
    R[n] = R[m];
    R[n] &= 0x0000FFFF;
    PC += 2;
}
```

(4) 使用例

```
EXTU.B      R0,R1     ;実行前 R0=H'FFFFFF80
                      ;実行後 R1=H'00000080
EXTU.W      R0,R1     ;実行前 R0=H'FFFF8000
                      ;実行後 R1=H'00008000
```

10.1.23 ICBI Instruction Cache Block Invalidate データ転送命令

命令キャッシュブロックの無効化

書式	動作概略	命令コード	実行ステート	T ビット
ICBI @Rn	論理アドレス Rn で示される命令キャッシュを無効化	0000nnnn11100011	13	—

(1) 説明

Rn の内容を実効アドレスとして命令キャッシュをアクセスします。キャッシュにヒットした場合、対応するキャッシュブロックを無効（V bit=0）にします。このとき、書き戻しはしません。キャッシングミスの場合や、キャッシング不可領域へのアクセスの場合は無効化を行いません。

(2) 注意

特になし

(3) 動作内容

```
ICBI(int n) /* ICBI @Rn */
{
    invalidate_instruction_cache_block(R[n]);
    PC += 2;
}
```

(4) 使用例

プログラムを RAM 上で書き換えて実行するプログラムでは、命令キャッシュが書き換え前の古い命令を持ったまま実行しないように、当該命令キャッシュブロックを ICBI 命令で無効にします。

(5) 発生する可能性がある例外

無効化を行わない場合でも、下記例外が発生する可能性があります。

- 命令TLB多重ヒット例外
- 命令TLBミス例外
- 命令TLB保護違反例外
- 命令アドレスエラー
- スロット不当命令例外

10.1.24 JMP　　JuMP　　分岐命令

無条件分岐　　**遅延分岐命令**

書式	動作概略	命令コード	実行ステート	T ビット
JMP　@Rn	Rn→PC	0100nnnn00101011	1	―

(1) 説明

Rn で指定したアドレスへ無条件に遅延分岐します。

(2) 注意

遅延分岐命令ですので、本命令の直後の命令を先に実行してから、分岐先の命令を実行します。

本命令と直後の命令との間には、割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

(3) 動作内容

```
JMP(int n)  /* JMP @Rn */
{
    unsigned int temp;

    temp = PC;
    PC = R[n];
    Delay_Slot(temp+2);
}
```

(4) 使用例

```
            MOV.L   JMP_TABLE,R0    ;R0=TRGET のアドレス
            JMP     @R0             ;TRGET へ分岐します。
            MOV     R0,R1           ;分岐に先立ち MOV を実行します。
            .align  4
JMP_TABLE:  .data.l  TRGET          ;ジャンプテーブル
        ...........
TRGET:      ADD     #1,R1           ;←分岐先
```

(5) 発生する可能性がある例外

- スロット不当命令例外

10.1.25 LDC LoaD to Control register システム制御命令

コントロールレジスタへのロード （特権命令）

書式		動作概略	命令コード	実行ステート	T ビット
LDC	Rm, GBR	Rm→GBR	0100mmmm00011110	1	—
LDC	Rm, VBR	Rm→VBR	0100mmmm00101110	1	—
LDC	Rm, SGR	Rm→SGR	0100mmmm00111010	4	—
LDC	Rm, SSR	Rm→SSR	0100mmmm00111110	1	—
LDC	Rm, SPC	Rm→SPC	0100mmmm01001110	1	—
LDC	Rm, DBR	Rm→DBR	0100mmmm11111010	4	—
LDC	Rm, R0_BANK	Rm→R0_BANK	0100mmmm10001110	1	—
LDC	Rm, R1_BANK	Rm→R1_BANK	0100mmmm10011110	1	—
LDC	Rm, R2_BANK	Rm→R2_BANK	0100mmmm10101110	1	—
LDC	Rm, R3_BANK	Rm→R3_BANK	0100mmmm10111110	1	—
LDC	Rm, R4_BANK	Rm→R4_BANK	0100mmmm11001110	1	—
LDC	Rm, R5_BANK	Rm→R5_BANK	0100mmmm11011110	1	—
LDC	Rm, R6_BANK	Rm→R6_BANK	0100mmmm11101110	1	—
LDC	Rm, R7_BANK	Rm→R7_BANK	0100mmmm11111110	1	—
LDC.L	@Rm+, GBR	(Rm)→GBR, Rm+4→Rm	0100mmmm00010111	1	—
LDC.L	@Rm+, VBR	(Rm)→VBR, Rm+4→Rm	0100mmmm00100111	1	—
LDC.L	@Rm+, SGR	(Rm)→SGR, Rm+4→Rm	0100mmmm00110110	4	—
LDC.L	@Rm+, SSR	(Rm)→SSR, Rm+4→Rm	0100mmmm00110111	1	—
LDC.L	@Rm+, SPC	(Rm)→SPC, Rm+4→Rm	0100mmmm01000111	1	—
LDC.L	@Rm+, DBR	(Rm)→DBR, Rm+4→Rm	0100mmmm11110110	4	—
LDC.L	@Rm+, R0_BANK	(Rm)→R0_BANK, Rm+4→Rm	0100mmmm10000111	1	—
LDC.L	@Rm+, R1_BANK	(Rm)→R1_BANK, Rm+4→Rm	0100mmmm10010111	1	—
LDC.L	@Rm+, R2_BANK	(Rm)→R2_BANK, Rm+4→Rm	0100mmmm10100111	1	—
LDC.L	@Rm+, R3_BANK	(Rm)→R3_BANK, Rm+4→Rm	0100mmmm10110111	1	—
LDC.L	@Rm+, R4_BANK	(Rm)→R4_BANK, Rm+4→Rm	0100mmmm11000111	1	—
LDC.L	@Rm+, R5_BANK	(Rm)→R5_BANK, Rm+4→Rm	0100mmmm11010111	1	—
LDC.L	@Rm+, R6_BANK	(Rm)→R6_BANK, Rm+4→Rm	0100mmmm11100111	1	—
LDC.L	@Rm+, R7_BANK	(Rm)→R7_BANK, Rm+4→Rm	0100mmmm11110111	1	—

(1) 説明

ソースオペランドをコントロールレジスタ GBR、VBR、SSR、SPC、DBR、SGR、または、R0_BANK～R7_BANK に格納します。

(2) 注意

LDC Rm, GBR と LDC.L @Rm+,GBR を除いた LDC, LDC.L 命令は、特権モードだけで使うことができます。もしユーザモードで使用された場合は、不当命令例外が発生します。ただし、LDC Rm,GBR と LDC.L @Rm+,GBR は、ユーザモードでも使用することができます。

LDC Rm, Rn_BANK, LDC.L @Rm, Rn_BANK 命令の Rn_BANK は、SR レジスタの RB ビットが 0 のとき、Rn_BANK1 を指し、RB ビットが 1 のとき、Rn_BANK0 を指します。

(3) 動作内容

```
LDCGBR(int m)        /* LDC Rm,GBR */
{
      GBR = R[m];
      PC += 2;
}
LDCVBR(int m)        /* LDC Rm,VBR : Privileged */
{
      VBR = R[m];
      PC += 2;
}
 LDCSGR(int m)       /* LDC Rm,SGR : Privileged */
{
      SGR = R[m];
      PC += 2;
}
LDCSSR(int m)        /* LDC Rm,SSR : Privileged */
{
      SSR = R[m];
      PC += 2;
}
LDCSPC(int m)        /* LDC Rm,SPC : Privileged */
{
      SPC = R[m];
      PC += 2;
}
LDCDBR(int m)        /* LDC Rm,DBR : Privileged */
{
      DBR = R[m];
      PC += 2;
}
```

```
LDCRn_BANK(int m)    /* LDC Rm,Rn_BANK : Privileged */
                     /* n=0-7 */
{
      Rn_BANK = R[m];
      PC += 2;
}
LDCMGBR(int m)       /* LDC.L @Rm+,GBR */
{
      GBR = Read_Long(R[m]);
      R[m] += 4;
      PC += 2;
}
LDCMVBR(int m)       /* LDC.L @Rm+,VBR : Privileged */
{
      VBR = Read_Long(R[m]);
      R[m] += 4;
      PC += 2;
}
LDCMSGR(int m)       /* LDC.L @Rm+,SGR : Privileged */
{
      SGR = Read_Long(R[m]);
      R[m] += 4;
      PC += 2;
}
LDCMSSR(int m)       /* LDC.L @Rm+,SSR : Privileged */
{
      SSR = Read_Long(R[m]);
      R[m] += 4;
      PC += 2;
}
LDCMSPC(int m)       /* LDC.L @Rm+,SPC : Privileged */
{
      SPC = Read_Long(R[m]);
      R[m] += 4;
      PC += 2;
}
LDCMDBR(int m)       /* LDC.L @Rm+,DBR : Privileged */
{
```

```
        DBR = Read_Long(R[m]);
        R[m] += 4;
        PC += 2;
}
LDCMRn_BANK(Long m)  /* LDC.L @Rm+,Rn_BANK : Privileged */
                     /*n=0-7 */
{
        Rn_BANK=Read_Long(R[m]);
        R[m] += 4;
        PC += 2;
}
```

(4) 発生する可能性がある例外

- データTLB多重ヒット例外
- 一般不当命令例外
- スロット不当命令例外
- データTLBミス例外
- データTLB保護違反例外
- データアドレスエラー

10.1.26 LDS LoaD to System register システム制御命令

システムレジスタへのロード

書式	動作概略	命令コード	実行ステート	Tビット
LDS Rm,MACH	Rm→MACH	0100mmmm00001010	1	—
LDS Rm,MACL	Rm→MACL	0100mmmm00011010	1	—
LDS Rm,PR	Rm→PR	0100mmmm00101010	1	—
LDS.L @Rm+,MACH	(Rm)→MACH, Rm＋4→Rm	0100mmmm00000110	1	—
LDS.L @Rm+,MACL	(Rm)→MACL, Rm＋4→Rm	0100mmmm00010110	1	—
LDS.L @Rm+,PR	(Rm)→PR, Rm＋4→Rm	0100mmmm00100110	1	—

(1) 説明

ソースオペランドをシステムレジスタ MACH、MACL、PR に格納します。

(2) 注意

特になし

(3) 動作内容

```
LDSMACH(long m)     /* LDS Rm,MACH  */
{
      MACH = R[m];
      PC += 2;
}
LDSMACL(long m)     /* LDS Rm,MACL  */
{
      MACL = R[m];
      PC += 2;
}
LDSPR(long  m)      /* LDS Rm,PR  */
{
      PR = R[m];
      PC += 2;
}
LDSMMACH(long  m)   /* LDS.L @Rm+,MACH  */
{
      MACH = Read_Long(R[m]);
```

```
        R[m] += 4;
        PC += 2;
}
LDSMMACL(long  m)   /* LDS.L @Rm+,MACL  */
{
        MACL = Read_Long(R[m]);
        R[m] += 4;
        PC += 2;
}
LDSMPR(long  m)     /* LDS.L @Rm+,PR  */
{
        PR=Read_Long(R[m]);
        R[m] += 4;
        PC += 2;
}
```

(4) 使用例

```
      LDS     R0、PR          ;実行前 R0=H'12345678、PR=H'00000000
                              ;実行後 PR=H'12345678
      LDS.L   @R15+,MACL      ;実行前 R15=H'10000000
                              ;実行後 P15=H'10000004,MACL=(H'10000000)
```

(5) 発生する可能性がある例外

LDS.L 命令でのみ以下の例外が発生する可能性があります。

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- データアドレスエラー

10.1.27 LDTLB LoaD PTEH/PTEL to TLB システム制御命令

TLB へのロード （特権命令）

書式	動作概略	命令コード	実行ステート	T ビット
LDTLB	PTEH/PTEL→TLB	0000000000111000	1	—

(1) 説明

PTEH/PTEL レジスタ内容を MMUCR.URC（MMC 制御レジスタのランダムカウンタフィールド）で指定した TLB (Translation Lookaside Buffer) に格納します。

LDTLB 命令は特権命令であり、特権モードでだけ使われます。もしユーザモードで使われた場合は不当命令例外が発生します。

(2) 注意

本命令は PTEH/PTEL レジスタを TLB にロードする命令なので、MMU がディスエーブル状態か、MMU がイネーブル状態で論理空間の P1 または P2 空間で本命令を使用するようにしてください（詳しくは**「第 7 章　メモリマネジメントユニット（MMU）」**を参照してください）。本命令の発行後、LDTLB 命令と領域 P0、U0、P3 へのアクセスに関わる命令、すなわち BRAF、BSRF、JMP、JSR、RTS、RTE の発行の間には少なくとも 1 つの命令がなければなりません。

(3) 動作内容

```
LDTLB( )   /*LDTLB */
{
    TLB[MMUCR. URC] .ASID=PTEH & 0x000000FF;
    TLB[MMUCR. URC] .VPN=(PTEH & 0xFFFFFC00) >> 10;
    TLB[MMUCR. URC] .PPN=(PTEH & 0x1FFFFC00) >> 10;
    TLB[MMUCR. URC] .SZ=(PTEL & 0x00000080) >> 6 | (PTEL & 0x00000010) >> 4;
    TLB[MMUCR. URC] .SH=(PTEH & 0x00000002) >> 1;
    TLB[MMUCR. URC] .PR=(PTEH & 0x00000060) >> 5;
    TLB[MMUCR. URC] .WT=(PTEH & 0x00000001);
    TLB[MMUCR. URC] .C=(PTEH & 0x00000008) >> 3;
    TLB[MMUCR. URC] .D=(PTEH & 0x00000004) >> 2;
    TLB[MMUCR. URC] .V=(PTEH & 0x00000100) >> 8;

    PC += 2;
}
```

(4) 使用例

```
MOV    @R0,R1        ;ページテーブルエントリ上位を R1 にロード
MOV    R1,@R2        ;R1 を PTEH にロード、R2 は PTEH のアドレス(H'FF000000)
LDTLB                ;PTEH,PTEL レジスタを TLB にロード
```

(5) 発生する可能性がある例外

- 一般不当命令例外
- スロット不当命令例外

10.1.28 MAC.L Multiply and ACcumulate Long 算術演算命令

倍精度積和演算

書式	動作概略	命令コード	実行ステート	T ビット
MAC.L @Rm+,@Rn+	符号付きで (Rn)×(Rm)+MAC→MAC Rn+4→Rn, Rm+4→Rm	0000nnnnmmmm1111	5	—

(1) 説明

汎用レジスタ Rm と Rn の内容をアドレスとする 32 ビットオペランドを符号付きで乗算し、結果の 64 ビットと MAC レジスタの内容とを加算し、結果を MAC レジスタに格納します。各オペランドを読み出す毎にそれぞれ、Rm を+4、Rn を+4 します。

S ビットが 0 のときは、連結した MACH、MACL レジスタに結果の 64 ビットを格納します。

S ビットが 1 のときは、MAC レジスタとの加算は LSB から 48 番目のビットで飽和演算になります。飽和演算では、MAC レジスタの下位 48 ビットのみが有効となり結果の範囲を H'FFFF800000000000(最小値)から H'00007FFFFFFFFFFF(最大値)までに制限します。

(2) 注意

特になし

(3) 動作内容

```
MACL(long m, long n)   /* MAC.L @Rm+,@Rn+ */
{
    unsigned long RnL,RnH,RmL,RmH,Res0,Res1,Res2;
    unsigned long temp0,temp1,temp2,temp3;
    long tempm,tempn,fnLmL;

    tempn = (long)Read_Long(R[n]);
    R[n] += 4;
    tempm = (long)Read_Long(R[m]);
    R[m] += 4;

    if((long)(tempn^tempm)<0) fnLmL = -1;
    else fnLmL = 0;
    if(tempn<0) tempn = 0-tempn;
```

```
    if(tempm<0) tempm = 0-tempm;

    temp1 = (unsigned long)tempn;
    temp2 = (unsigned long)tempm;

    RnL = temp1 & 0x0000FFFF;
    RnH = (temp1>>16) & 0x0000FFFF;
    RmL = temp2 & 0x0000FFFF;
    RmH = (temp2>>16)& 0x0000FFFF;
    temp0 = RmL * RnL;
    temp1 = RmH * RnL;
    temp2 = RmL * RnH;
    temp3 = RmH * RnH;

    Res2 = 0;

Res1 = temp1 + temp2;
if(Res1<temp1) Res2 += 0x00010000;

temp1 = (Res1<<16) & 0xFFFF0000;
Res0 = temp0 + temp1;
if(Res0<temp0) Res2++;

Res2 = Res2 + ((Res1>>16) & 0x0000FFFF) + temp3;

if(fnLmL<0){
    Res2 = ~Res2;
    if(Res0==0) Res2++;
    else Res0 = (~Res0)+1;
}
if(S==1){
    Res0 = MACL + Res0;
    if(MACL>Res0) Res2++;
    if(MACH&0x00008000);
    else Res2 += MACH | 0xFFFF0000;
         Res2 += MACH & 0x00007FFF;

    if(((long)Res2<0)&&(Res2<0xFFFF8000)){
```

```
        Res2 = 0xFFFF8000;
        Res0 = 0x00000000;
    }
    if(((long)Res2>0)&&(Res2>0x00007FFF)){
        Res2 = 0x00007FFF;
        Res0 = 0xFFFFFFFF;
    };

    MACH = (Res2 & 0x0000FFFF) | (MACH & 0xFFFF0000);
    MACL = Res0;
}
    else {
        Res0 = MACL + Res0;
        if(MACL>Res0) Res2++;
        Res2 += MACH;

        MACH = Res2;
        MACL = Res0;
    }
    PC += 2;
}
```

(4) 使用例

```
        MOVA    TBLM,R0         ;テーブルのアドレスを得る
        MOV     R0,R1           ;
        MOVA    TBLN,R0         ;テーブルのアドレスを得る
        CLRMAC                  ;MAC レジスタの初期化
        MAC.L   @R0+,@R1+       ;
        MAC.L   @R0+,@R1+       ;
        STS     MACL,R0         ;結果を R0 に得る
        ............
        .align  2               ;
TBLM    .data.l H'1234ABCD      ;
        .data.l H'5678EF01      ;
TBLN    .data.l H'0123ABCD      ;
        .data.l H'4567DEF0      ;
```

(5) 発生する可能性がある例外

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- データアドレスエラー

10.1.29 MAC.W Multiply and ACcumulate Word 算術演算命令

積和演算

書式	動作概略	命令コード	実行ステート	Tビット
MAC.W @Rm+,@Rn+ MAC @Rm+,@Rn+	符号付きで (Rn)×(Rm)+MAC→MAC Rn+2→Rn, Rm+2→Rm	0100nnnnmmmm1111	4	—

(1) 説明

汎用レジスタ Rm と Rn の内容をアドレスとする 16 ビットオペランドを符号付きで乗算し、結果の 32 ビットと MAC レジスタの内容とを加算し、結果を MAC レジスタに格納します。各オペランドを読み出す毎にそれぞれ、Rm を+2、Rn を+2 します。

S ビットが 0 のとき、16×16+64→64 ビットの積和演算となり、連結した MACH、MACL レジスタに結果の 64 ビットを格納します。

S ビットが 1 のとき、16×16+32→32 ビットの積和演算となり、MAC レジスタとの加算は飽和演算になります。飽和演算では、MACL レジスタのみが有効となり結果の範囲を H'80000000(最小値)から H'7FFFFFFF(最大値)までに制限します。オーバフローが発生すると、MACH レジスタの LSB を 1 にセットします。結果が負の方向にオーバフローしたときは H'80000000(最小値)を、正の方向にオーバフローしたときは H'7FFFFFFF(最大値)を、MACL レジスタに格納します。

(2) 注意

S ビットが 0 のとき、16×16+64→64 ビットの積和演算を行います。

(3) 動作内容

```
MACW(long m, long n)   /* MAC.W @Rm+,@Rn+ */
{
    long tempm,tempn,dest,src,ans;
    unsigned long templ;
    tempn = (long)Read_Word(R[n]);
    R[n] += 2;
    tempm = (long)Read_Word(R[m]);
    R[m] += 2;
    templ=MACL;
    tempm = ((long)(short)tempn*(long)(short)tempm);
    if((long)MACL>=0) dest = 0;
```

```
    else dest = 1;
    if((long)tempm>=0) {
       src=0;
       tempn = 0;
    }
    else {
       src=1;
       tempn = 0xFFFFFFFF;
    }
    src += dest;
    MACL += tempm;
    if((long)MACL>=0) ans = 0;
    else ans = 1;
    ans += dest;
    if(S==1) {
       if(ans==1) {
          if(src==0) MACL = 0x7FFFFFFF;
          if(src==2) MACL = 0x80000000;
       }
    }
    else {
       MACH += tempn;
       if(templ>MACL) MACH += 1;

    }
    PC += 2;
}
```

(4) 使用例

```
        MOVA    TBLM,R0         ;テーブルのアドレスを得る
        MOV     R0,R1           ;
        MOVA    TBLN,R0         ;テーブルのアドレスを得る
        CLRMAC                  ;MAC レジスタの初期化
        MAC.W   @R0+,@R1+       ;
        MAC.W   @R0+,@R1+       ;
        STS     MACL,R0         ;結果を R0 に得る
        ............
        .align  2               ;
```

```
TBLM    .data.w   H'1234        ;
        .data.w   H'5678        ;
TBLN    .data.w   H'0123        ;
        .data.w   H'4567        ;
```

(5) 発生する可能性がある例外

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- データアドレスエラー

10.1.30 MOV　　MOVe data　　データ転送命令

データ転送

書式		動作概略	命令コード	実行ステート	Tビット
MOV	Rm,Rn	Rm→Rn	0110nnnnmmmm0011	1	—
MOV.B	Rm,@Rn	Rm→(Rn)	0010nnnnmmmm0000	1	—
MOV.W	Rm,@Rn	Rm→(Rn)	0010nnnnmmmm0001	1	—
MOV.L	Rm,@Rn	Rm→(Rn)	0010nnnnmmmm0010	1	—
MOV.B	@Rm,Rn	(Rm)→符号拡張→Rn	0110nnnnmmmm0000	1	—
MOV.W	@Rm,Rn	(Rm)→符号拡張→Rn	0110nnnnmmmm0001	1	—
MOV.L	@Rm,Rn	(Rm)→Rn	0110nnnnmmmm0010	1	—
MOV.B	Rm,@-Rn	Rn-1→Rn, Rm→(Rn)	0010nnnnmmmm0100	1	—
MOV.W	Rm,@-Rn	Rn-2→Rn, Rm→(Rn)	0010nnnnmmmm0101	1	—
MOV.L	Rm,@-Rn	Rn-4→Rn, Rm→(Rn)	0010nnnnmmmm0110	1	—
MOV.B	@Rm+,Rn	(Rm)→符号拡張→Rn, Rm+1→Rm	0110nnnnmmmm0100	1	—
MOV.W	@Rm+,Rn	(Rm)→符号拡張→Rn, Rm+2→Rm	0110nnnnmmmm0101	1	—
MOV.L	@Rm+,Rn	(Rm)→Rn, Rm+4→Rm	0110nnnnmmmm0110	1	—
MOV.B	Rm,@(R0,Rn)	Rm→(R0+Rn)	0000nnnnmmmm0100	1	—
MOV.W	Rm,@(R0,Rn)	Rm→(R0+Rn)	0000nnnnmmmm0101	1	—
MOV.L	Rm,@(R0,Rn)	Rm→(R0+Rn)	0000nnnnmmmm0110	1	—
MOV.B	@(R0,Rm),Rn	(R0+Rm)→符号拡張→Rn	0000nnnnmmmm1100	1	—
MOV.W	@(R0,Rm),Rn	(R0+Rm)→符号拡張→Rn	0000nnnnmmmm1101	1	—
MOV.L	@(R0,Rm),Rn	(R0+Rm)→Rn	0000nnnnmmmm1110	1	—

(1) 説明

ソースオペランドをデスティネーションへ転送します。オペランドがメモリのときは転送するデータサイズをバイト/ワード/ロングワードの範囲で指定できます。ソースオペランドがメモリのときは、ロードされたデータをロングワードに符号拡張後レジスタに格納します。

(2) 注意

特になし

(3) 動作内容

```
MOV(long m, long n)   /* MOV Rm,Rn */
{
    R[n] = R[m];
    PC += 2;
```

```
}

MOVBS(long m, long n) /* MOV.B Rm,@Rn */
{
    Write_Byte(R[n],R[m]);
    PC += 2;
}

MOVWS(long m, long n) /* MOV.W Rm,@Rn */
{
    Write_Word(R[n],R[m]);
    PC += 2;
}

MOVLS(long m, long n) /* MOV.L Rm,@Rn */
{
    Write_Long(R[n],R[m]);
    PC += 2;
}

MOVBL(long m, long n) /* MOV.B @Rm,Rn */
{
    R[n] = (long)Read_Byte(R[m]);
    if((R[n]&0x80)==0) R[n] &= 0x000000FF;
    else R[n] |= 0xFFFFFF00;
    PC += 2;
}

MOVWL(long m, long n) /* MOV.W @Rm,Rn */
{
    R[n] = (long)Read_Word(R[m]);
    if((R[n]&0x8000)==0) R[n] &= 0x0000FFFF;
    else R[n] |= 0xFFFF0000;
    PC += 2;
}

MOVLL(long m, long n) /* MOV.L @Rm,Rn */
}
```

```
    R[n] = Read_Long(R[m]);
    PC += 2;
}

MOVBM(long m, long n) /* MOV.B Rm,@-Rn */
{
    Write_Byte(R[n]-1,R[m]);
    R[n] -= 1;
    PC += 2;
}

MOVWM(long m, long n) /* MOV.W Rm,@-Rn */
{
    Write_Word(R[n]-2,R[m]);
    R[n] -= 2;
    PC += 2;
}

MOVLM(long m, long n)   /* MOV.L Rm,@-Rn */
{
    Write_Long(R[n]-4,R[m]);
    R[n] -= 4;
    PC += 2;
}

MOVBP(long m, long n) /* MOV.B @Rm+,Rn */
{
    R[n] = (long)Read_Byte(R[m]);
    if((R[n]&0x80)==0) R[n] &= 0x000000FF;
    else R[n] |= 0xFFFFFF00;
    if(n != m) R[m] += 1;
    PC += 2;
}
MOVWP(long m, long n)    /* MOV.W @Rm+,Rn */
{
    R[n] = (long)Read_Word(R[m]);
    if((R[n]&0x8000)==0) R[n] &= 0x0000FFFF;
    else R[n] |= 0xFFFF0000;
```

```
    if(n != m) R[m] += 2;
    PC += 2;
}

MOVLP(long m, long n)   /* MOV.L @Rm+,Rn */
{
    R[n] = Read_Long(R[m]);
    if(n != m) R[m] += 4;
    PC += 2;
}

MOVBS0(long m, long n) /* MOV.B Rm,@(R0,Rn) */
{
    Write_Byte(R[n]+R[0],R[m]);
    PC += 2;
}

MOVWS0(long m, long n) /* MOV.W Rm,@(R0,Rn) */
{
    Write_Word(R[n]+R[0],R[m]);
   PC += 2;
}

MOVLS0(long m, long n) /* MOV.L Rm,@(R0,Rn) */
{
    Write_Long(R[n]+R[0],R[m]);
    PC += 2;
}

MOVBL0(long m, long n) /* MOV.B @(R0,Rm),Rn */
{
   R[n] = (long)Read_Byte(R[m]+R[0]);
   if((R[n]&0x80)==0) R[n] &= 0x000000FF;
   else R[n] |= 0xFFFFFF00;
   PC += 2;
}

MOVWL0(long m, long n)   /* MOV.W @(R0,Rm),Rn */
```

```
{
    R[n] = (long)Read_Word(R[m]+R[0]);
    if((R[n]&0x8000)==0) R[n] &= 0x0000FFFF;
    else R[n] |= 0xFFFF0000;
    PC += 2;
}

MOVLL0(long m, long n)   /* MOV.L @(R0,Rm),Rn */
{
    R[n] = Read_Long(R[m]+R[0]);
    PC += 2;
}
```

(4) 使用例

```
MOV       R0,R1          ;実行前 R0=H'FFFFFFFF,R1=H'00000000
                         ;実行後 R1=H'FFFFFFFF
MOV.W     R0,@R1         ;実行前 R0=H'FFFF7F80
                         ;実行後(R1)=H'7F80
MOV.B     @R0,R1         ;実行前(R0)=H'80,R1=H'00000000
                         ;実行後 R1=H'FFFFFF80
MOV.W     R0,@-R1        ;実行前 R0=H'AAAAAAAA,(R1)=H'FFFF7F80
                         ;実行後 R1=H'FFFF7F7E,(R1)=H'AAAA
MOV.L     @R0+,R1        ;実行前 R0=H'12345670
                         ;実行後 R0=H'12345674,R1=(H'12345670)
MOV.B     R1,@(R0,R2)    ;実行前 R2=H'00000004,R0=H'10000000
                         ;実行後(H'10000004)=R1
MOV.W     @(R0,R2),R1    ;実行前 R2=H'00000004,R0=H'10000000
                         ;実行後 R1=(H'10000004)
```

(5) 発生する可能性がある例外

MOV Rm,Rn 命令を除き以下の例外が発生する可能性があります。

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- データアドレスエラー
- 初期ページ書き込み例外（メモリへ書き込む命令のみ）

10.1.31 MOV　　MOVe constant value　　データ転送命令

イミディエイト

データの転送

書式		動作概略	命令コード	実行ステート	T ビット
MOV	#imm,Rn	imm→符号拡張→Rn	1110nnnniiiiiiii	1	—
MOV.W	@(disp*,PC),Rn	(disp×2+PC+4)→符号拡張→Rn	1001nnnndddddddd	1	—
MOV.L	@(disp*,PC),Rn	(disp×4+PC&H'FFFFFFFC+4)→Rn	1101nnnndddddddd	1	—

【注】* ルネサスのアセンブラでは、disp にスケーリング後（×1、×2、×4）の値を設定します。

(1) 説明

ロングワードに符号拡張したイミディエイトデータを汎用レジスタ Rn に格納します。データがワードまたはロングワードのときは、データはアドレス（PC+4+ディスプレースメント×2）または（PC+4+ディスプレースメント×4）のメモリ位置から格納されます。

データがワードのとき、8 ビットディスプレースメントはゼロ拡張後 2 倍しますので、テーブルとの相対距離は PC+4+510 バイトまでの範囲になります。PC は本命令の命令アドレスです。

データがロングワードのとき、8 ビットディスプレースメントはゼロ拡張後 4 倍しますので、オペランドとの相対距離は PC+4+1020 バイトまでの範囲になります。 PC は本命令の命令アドレスですが、下位 2 ビットを B'00 に補正した値をアドレス計算に使用します。

(2) 注意

PC 相対ロード命令を遅延スロットで実行するとスロット不当命令例外が発生します。

(3) 動作内容

```
MOVI(int i, int n)  /* MOV #imm,Rn */
{
     if((i&0x80)==0) R[n]=(0x000000FF & i);
     else R[n] = (0xFFFFFF00 | i);
     PC += 2;
}

MOVWI(d, n) /* MOV.W @(disp,PC),Rn */
{
     unsigned int disp;

     disp = (unsigned int)(0x000000FF & d);
     R[n] = (int)Read_Word(PC+4+(disp<<1));
```

```
        if((R[n]&0x8000)==0) R[n] &= 0x0000FFFF;
        else R[n] |= 0xFFFF0000;
        PC += 2;
    }

        MOVLI(int d, int n)/* MOV.L @(disp,PC),Rn */
    {
        unsigned int disp;

        disp = (unsigned int)(0x000000FF & (int)d);
        R[n] = Read_Long((PC&0xFFFFFFFC)+4+(disp<<2));
        PC += 2;
    }
```

(4) 使用例

アドレス

```
1000        MOV       #H'80,R1      ;R1=H'FFFFFF80
1002        MOV.W     IMM,R2        ;R2=H'FFFF9ABC  IMM は(PC+4+H'08) の意味
1004        ADD       #-1,R0        ;
1006        TST       R0,R0         ;
1008        MOV.L     @(3*,PC),R3   ;R3=H'12345678
100A        BRA       NEXT          ;遅延分岐命令
100C        NOP
100E IMM    .data.w   H'9ABC        ;
1010        .data.w   H'1234        ;
1012 NEXT   JMP       @R3           ;BRA の分岐先
1014        CMP/EQ    #0,R0         ;
            .align    4             ;
1018        .data.l   H'12345678    ;
101C        .data.l   H'9ABCDEF0    ;
```

【注】*　ルネサスのアセンブラでは、disp にスケーリング後（×1、×2、×4）の値を設定します。

(5) 発生する可能性がある例外

PC 相対ロード命令でのみ以下の例外が発生する可能性があります。

- データTLB多重ヒット例外
- スロット不当命令例外
- データTLBミス例外

- データTLB保護違反例外
- データアドレスエラー

10.1.32 MOV MOVe global data データ転送命令

グローバルデータの転送

書式		動作概略	命令コード	実行ステート	T ビット
MOV.B	@(disp*,GBR),R0	(disp+GBR)→符号拡張→R0	11000100dddddddd	1	—
MOV.W	@(disp*,GBR), R0	(disp×2+GBR)→符号拡張→R0	11000101dddddddd	1	—
MOV.L	@(disp*,GBR),R0	(disp×4+GBR)→R0	11000110dddddddd	1	—
MOV.B	R0,@(disp*,GBR)	R0→(disp+GBR)	11000000dddddddd	1	—
MOV.W	R0,@(disp*,GBR)	R0→(disp×2+GBR)	11000001dddddddd	1	—
MOV.L	R0,@(disp*,GBR)	R0→(disp×4+GBR)	11000010dddddddd	1	—

【注】 * ルネサスのアセンブラでは、disp にスケーリング後（×1、×2、×4）の値を設定します。

(1) 説明

ソースオペランドをデスティネーションへ転送します。データサイズをバイト、ワード、またはロングワードの範囲で指定できますが、レジスタが R0 固定になります。転送データがバイトサイズのとき 8 ビットディスプレースメントはゼロ拡張するだけですので、+255 バイトまでの範囲が指定できます。ワードサイズのとき 8 ビットディスプレースメントはゼロ拡張後 2 倍しますので、+510 バイトまでの範囲が指定できます。ロングワードサイズのとき 8 ビットディスプレースメントはゼロ拡張後 4 倍しますので、+1020 バイトまでの範囲が指定できます。

ソースオペランドがメモリのときは、ロードされたデータをロングワードに符号拡張後レジスタへ格納します。

(2) 注意

ロードするときデスティネーションレジスタが R0 固定です。

(3) 動作内容

```
MOVBLG(int d)  /* MOV.B @(disp,GBR),R0 */
{
        unsigned int disp;

        disp = (unsigned int)(0x000000FF & d);
        R[0] = (int)Read_Byte(GBR+disp);
        if((R[0]&0x80)==0) R[0] &= 0x000000FF;
        else R[0] |= 0xFFFFFF00;
        PC += 2;
}

MOVWLG(int d)  /* MOV.W @(disp,GBR),R0 */
```

```
{
        unsigned int disp;

        disp = (unsigned int)(0x000000FF & d);

        R[0] = (int)Read_Word(GBR+(disp<<1));
        if((R[0]&0x8000)==0) R[0] &= 0x0000FFFF;
        else R[0] |= 0xFFFF0000;
        PC += 2;
}

MOVLLG(int d)  /* MOV.L @(disp,GBR),R0 */
{

        unsigned int disp;

        disp = (unsigned int)(0x000000FF & d);
        R[0] = Read_Long(GBR+(disp<<2));
        PC += 2;
}
MOVBSG(int d) /* MOV.B R0,@(disp,GBR) */
{
        unsigned int disp;

        disp = (unsigned int)(0x000000FF & d);
        Write_Byte(GBR+disp,R[0]);
        PC += 2;
}

MOVWSG(int d) /* MOV.W R0,@(disp,GBR) */
{
        unsigned int disp;

        disp = (unsigned int)(0x000000FF & d);
        Write_Word(GBR+(disp<<1),R[0]);
        PC += 2;
}
```

```
MOVLSG(int d) /* MOV.L R0,@(disp,GBR) */
{
        unsigned int disp;

        disp = (unsigned int)(0x000000FF & (long)d);
        Write_Long(GBR+(disp<<2),R[0]);
        PC += 2;
}
```

(4) 使用例

```
MOV.L  @(2*,GBR),R0      ;実行前 (GBR+8)=H'12345670
                         ;実行後 R0=H'12345670
MOV.B  R0,@(1*,GBR)      ;実行前 R0=H'FFFF7F80
                         ;実行後 (GBR+1)=H'80
```

【注】* ルネサスのアセンブラでは、disp にスケーリング後（×1、×2、×4）の値を設定します。

(5) 発生する可能性がある例外

- データTLB多重ヒット例外
- スロット不当命令例外
- データTLBミス例外
- データTLB保護違反例外
- データアドレスエラー
- 初期ページ書き込み例外（メモリへ書き込む命令のみ）

10.1.33 MOV MOVe structure data データ転送命令

構造体データの転送

書式	動作概略	命令コード	実行ステート	T ビット
MOV.B R0,@(disp*,Rn)	R0→(disp+Rn)	10000000nnnndddd	1	—
MOV.W R0,@(disp*,Rn)	R0→(disp×2+Rn)	10000001nnnndddd	1	—
MOV.L Rm,@(disp*,Rn)	Rm→(disp×4+Rn)	0001nnnnmmmmdddd	1	—
MOV.B @(disp*,Rm),R0	(disp+Rm)→符号拡張→R0	10000100mmmmdddd	1	—
MOV.W @(disp*,Rm),R0	(disp×2+Rm)→符号拡張→R0	10000101mmmmdddd	1	—
MOV.L @(disp*,Rm),Rn	(disp×4+Rm)→Rn	0101nnnnmmmmdddd	1	—

【注】* ルネサスのアセンブラでは、disp にスケーリング後（×1、×2、×4）の値を設定します。

(1) 説明

ソースオペランドをデスティネーションへ転送します。構造体、スタック内のデータアクセスに最適です。データサイズをバイト、ワード、またはロングワードの範囲で指定できますが、バイトまたはワードのときはレジスタが R0 固定になります。

データがバイトサイズのとき 4 ビットディスプレースメントはゼロ拡張するだけですので、+15 バイトまでの範囲が指定できます。ワードサイズのとき 4 ビットディスプレースメントはゼロ拡張後 2 倍しますので、+30 バイトまでの範囲が指定できます。ロングワードサイズのとき 4 ビットディスプレースメントはゼロ拡張後 4 倍しますので、+60 バイトまでの範囲が指定できます。メモリオペランドに届かないときは前述の@(R0,Rn)モードを使う必要があります。

ソースオペランドがメモリのときは、ロードされたデータをロングワードに符号拡張後レジスタへ格納します。

(2) 注意

バイト/ワードデータをロードするときデスティネーションレジスが R0 固定です。したがって、直後の命令で R0 を参照しようとしてもロード命令の実行完了まで待たされます。これは命令の順序を替えることによって最適化が可能です。

```
MOV.B  @(2,R1),R0          MOV.B  @(2,R1),R0
AND    #80,R0      ──►     ADD    #20,R1
ADD    #20,R1      ──►     AND    #80,R0
```

(3) 動作内容

```
MOVBS4(long d), long n   /* MOV.B R0,@(disp,Rn) */
{
    long disp;
    disp = (0x0000000F & (long)d);
    Write_Byte(R[n]+disp,R[0]);
    PC += 2;
}

MOVWS4(long d, long n)  /* MOV.W R0,@(disp,Rn) */
{
    long disp;

    disp = (0x0000000F & (long)d);
    Write_Word(R[n]+(disp<<1),R[0]);
    PC += 2;
}

MOVLS4(long m, long d, long n)                  /* MOV.L Rm,@(disp,Rn) */
{
    long disp;

    disp = (0x0000000F & (long)d);
    Write_Long(R[n]+(disp<<2),R[m]);
    PC += 2;
}

MOVBL4(long m, long d)  /* MOV.B @(disp,Rm),R0 */
{
    long disp;

    disp = (0x0000000F & (long)d);
    R[0] = Read_Byte(R[m]+disp);
    if((R[0]&0x80)==0) R[0] &= 0x000000FF;
    else R[0] |= 0xFFFFFF00;
    PC += 2;
}
```

```
MOVWL4(long m, long d) /* MOV.W @(disp,Rm),R0 */
{
   long disp;

   disp = (0x0000000F & (long)d);
   R[0] = Read_Word(R[m]+(disp<<1));
   if((R[0]&0x8000)==0) R[0] &= 0x0000FFFF;
   else R[0] |= 0xFFFF0000;
   PC += 2;
}

MOVLL4(long m, long d, long n) /* MOV.L @(disp,Rm),Rn */
{
   long disp;

   disp = (0x0000000F & (long)d);
   R[n] = Read_Long(R[m]+(disp<<2));
   PC += 2;
}
```

(4) 使用例

```
MOV.L   @(2*,R0),R1      ;実行前 (R0+8)=H'12345670
                         ;実行後 R1=H'12345670
MOV.L   R0,@(H'F*,R1)    ;実行前 R0=H'FFFF7F80
                         ;実行後 (R1+60)=H'FFFF7F80
```

【注】*　ルネサスのアセンブラでは、disp にスケーリング後（×1、×2、×4）の値を設定します。

(5) 発生する可能性がある例外

- データTLB多重ヒット例外
- スロット不当命令例外
- データTLBミス例外
- データTLB保護違反例外
- データアドレスエラー
- 初期ページ書き込み例外（メモリへ書き込む命令のみ）

10.1.34 MOVA MOVe effective Address データ転送命令

実効アドレスの転送

書式	動作概略	命令コード	実行ステート	T ビット
MOVA @(disp*,PC),R0	Disp×4+PC&H'FFFFFFFC+4→R0	11000111dddddddd	1	—

【注】* ルネサスのアセンブラでは、disp にスケーリング後（×1、×2、×4）の値を設定します。

(1) 説明

汎用レジスタ R0 にソースオペランドの実効アドレスを格納します。8 ビットディスプレースメントはゼロ拡張後 4 倍します。PC は本命令の命令アドレスですが、下位 2 ビットを B'00 に補正した値をアドレス計算に使用します。

(2) 注意

本命令を遅延スロットで実行すると、スロット不当命令例外が発生します。

(3) 動作内容

```
MOVA(int d)             /* MOVA @(disp,PC),R0 */
{
        unsigned int disp;

        disp = (unsigned int)(0x000000FF & d);
        R[0] = (PC&0xFFFFFFFC) + 4 + (disp<<2);
        PC += 2;
}
```

(4) 使用例

```
アドレス      .org    H'1006
1006          MOVA    STR*,R0      ;STR のアドレス→R0
1008          MOV.B   @R0,R1       ;R1="X"  ←PC 下位 2 ビット補正後の位置
100A          ADD     R4,R5
              .align  4
100C  STR:    .sdata  "XYZP12"
```

【注】* ルネサスのアセンブラでは、disp にスケーリング後（×1、×2、×4）の値を設定します。

(5) 発生する可能性がある例外

- スロット不当命令例外

10.1.35 MOVCA.L MOVe with Cache block Allocation データ転送命令

キャッシュブロックの確保

書式	動作概略	命令コード	実行ステート	T ビット
MOVCA.L R0,@Rn	（キャッシュブロックをフェッチせずに）R0→(Rn)	0000nnnn11000011	1	—

(1) 説明

汎用レジスタ R0 を、実効アドレス Rn で示されているメモリに格納します。この命令は他の格納命令とは以下の点で異なります。

アクセスされたメモリがライトバック方式を選択していた場合で、かつキャッシュミスが起きた場合、キャッシュブロックは確保されますが、ブロックリードは行わず、R0 のデータの書き込みを、そのキャッシュブロックに行います。他のキャッシュブロックの内容は未定義です。

(2) 注意

特にありません。

(3) 動作内容

```
MOVCAL(int n)     /*MOVCA.L  R0,@Rn */
{
    if((is_write_back_memory(R[n]))
          && (look_up_in_operand_cache(R[n]) == MISS))
                allocate_operand_cache_block(R[n]);
    Write_Long(R[n],R[0]);
    PC += 2;
}
```

(4) 発生する可能性がある例外

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- 初期ページ書き込み例外
- データアドレスエラー

10.1.36 MOVCO MOVe COnditional データ転送命令

Read-Modify-Write がアトミックに完了した
場合にストアする条件付ストア

書式	動作概略	命令コード	実行ステート	T ビット
MOVCO.L R0,@Rn	LDST→T if(T==1) R0→(Rn) 0→LDST	0000nnnn01110011	1	LDST

(1) 説明

MOVLI 命令と MOVCO 命令を組み合わせて、シングルプロセッサ上のアトミックな Read-Modify-Write を実現します。

本命令は LDST フラグの値を T ビットにコピーし、T が 1 の場合に R0 の値をアドレス Rm にストアし、T が 0 の場合はストアしません。最後に LDST フラグを 0 にクリアします。LDST フラグは割り込みや例外が発生すると 0 にクリアされますので、MOVLI 命令と MOVCO 命令との間に割り込みや例外が発生しなかった場合に MOVCO 命令のストアが実行されます。

(2) 注意

特になし

(3) 動作内容

```
MOVCO(long n) /* MOVCO Rn,@Rn */
{
    T = LDST;
    if(T==1)
            Write_Long(R[n],R[0]);
    LDST = 02;
    PC += 2
}
```

(4) 使用例

```
Retry:   MOVLI.L   @Rn,R0        ;アトミックなインクリメント
         ADD       #1,R0
         MOVCO.L   R0,@Rn
         BF        Retry         ;MOVLI、MOVCO 間に割り込み／例外発生すると再実行
         NOP
```

(5) 発生する可能性がある例外

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- 初期ページ書き込み例外
- データアドレスエラー

10.1.37 MOVLI MOVe LInked データ転送命令

アトミックな Read-Modify-Write
を開始するロード

書式	動作概略	命令コード	実行ステート	T ビット
MOVLI.L @Rm,R0	1→LDST, (Rm)→R0 ただし、割り込み／例外発生時 0→LDST	0000nnnn01100011	1	—

(1) 説明

MOVLI 命令と MOVCO 命令を組み合わせて、シングルプロセッサ上のアトミックな Read-Modify-Write を実現します。

本命令は LDST フラグに 1 をセットし、Rm の指す 4 バイトデータを R0 に読み込みます。割り込みや例外が発生すると、LDST は 0 にクリアされます。

MOVLI 命令が 1 にセットした LDST が割り込みや例外により 0 にクリアされずに MOVCO 命令を実行した場合、MOVCO 命令はストアを実行し T ビットに 1 をセットします。LDST が 0 にクリアされた状態で MOVCO 命令を実行した場合、ストアを実行せず、T ビットに 0 を設定します。

(2) 注意

特になし

(3) 動作内容

```
MOVLINK(long m) /* MOVLI Rm,@Rn */
{
    LDST = 1;
    R[0] = Read_Long(R[m]);
    PC += 2
}
```

(4) 使用例

MOVCO 命令を参照してください。

(5) 発生する可能性がある例外

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- データアドレスエラー

10.1.38 MOVT　　MOVe Tbit　　データ転送命令

Ｔビットの転送

書式	動作概略	命令コード	実行ステート	Ｔビット
MOVT　Rn	T→Rn	0000nnnn00101001	1	―

(1) 説明

T ビットを汎用レジスタ Rn に格納します。T=1 のとき Rn=1、T=0 のとき Rn=0 になります。

(2) 注意

特になし

(3) 動作内容

```
MOVT(long n)              /* MOVT Rn */
{
      R[n] = (0x00000001 & SR);
      PC += 2;
}
```

(4) 使用例

```
XOR       R2,R2       ;R2=0
CMP/PZ    R2          ;T=1
MOVT      R0          ;R0=1
CLRT                  ;T=0
MOVT      R1          ;R1=0
```

10.1.39 MOVUA MOVe UnAligned データ転送命令

非境界調整データ転送

書式	動作概略	命令コード	実行ステート	Tビット
MOVUA.L @Rm,R0	(Rm)→R0 非境界調整データをロード	0100nnnn10101001	2	—
MOVUA.L @Rm+,R0	(Rm)→R0、Rm+4→Rm 非境界調整データをロード	0100nnnn11101001	2	—

(1) 説明

Rm の内容を実効アドレスとして、メモリにあるロングワードデータを R0 にロードします。ロングワードデータをロングワード境界のアドレス（4n）だけでなく、ロングワード境界でないアドレス（4n+1、4n+2、4n+3）からロードできます。ロングワード境界でないアドレス（4n+1、4n+2、4n+3）をアクセスしてもデータアドレスエラー例外が発生しません。

(2) 注意

特になし

(3) 動作内容

```
MOVUAL(int m) /* MOVUA.L Rm,R0*/
{
     Read_Unaligned_Long(R0,R[m]);
     PC += 2;
}
MOVUALP(int m) /* MOVUA.L Rm+,R0*/
{
     Read_Unaligned_Long(R0,R[m]);
     if(m != 0) R[m] += 4;
     PC += 2;
}
```

(4) 使用例

```
MOVUA.L  @R1,R0      ;実行前 R1=H'00001001、R0=H'00000000
                     ;実行後 R0=(H'00001001)
MOVUA.L  @R1+,R0     ;実行前 R1=H'00001007、R0=H'00000000
                     ;実行後 R1=H'0000100B、R0=(H'00001007)
```

```
;ソースオペランドが@R0 である特別な場合
MOVUA.L   @R0,R0       ;実行前 R0=H'00001001
                       ;実行後 R0=(H'00001001)
MOVUA.L   @R0+,R0      ;実行前 R0=H'00001001
                       ;実行後 R0=(H'00001001)
```

(5) 発生する可能性がある例外

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- データアドレスエラー例外（ユーザモードで特権領域をアクセスしたとき）

10.1.40 MUL.L MULtiply Long 算術演算命令

倍精度乗算

書式	動作概略	命令コード	実行ステート	Tビット
MUL.L Rm,Rn	Rn×Rm→MACL	0000nnnnmmmm0111	2	—

(1) 説明

汎用レジスタ Rn の内容と Rm を 32 ビットで乗算し、結果の下位側 32 ビットを MACL レジスタに格納します。MACH の内容は変化しません。

(2) 注意

特になし

(3) 動作内容

```
MULL(long m, long n) /* MUL.L Rm,Rn */
{
    MACL = R[n]*R[m];
    PC += 2;
}
```

(4) 使用例

```
MUL.L   R0,R1       ;実行前  R0=H'FFFFFFFE,R1=H'00005555
                    ;実行後  MACL=H'FFFF5556
STS     MACL,R0     ;演算結果を得る
```

10.1.41 MULS.W MULtiply as Signed Word 算術演算命令

符号付き乗算

書式	動作概略	命令コード	実行ステート	T ビット
MULS.W Rm,Rn	符号付きで Rn×Rm→MACL	0010nnnnmmmm1111	1	—

(1) 説明

汎用レジスタ Rn の内容と Rm を 16 ビットで乗算し、結果の 32 ビットを MACL レジスタに格納します。演算は符号付き算術演算で行います。MACH の内容は変化しません。

(2) 注意

特になし

(3) 動作内容

```
MULS(long m, long n) /* MULS Rm,Rn */
{
    MACL = ((long)(short)R[n]*(long)(short)R[m]);
    PC += 2;
}
```

(4) 使用例

```
MULS.W   R0,R1       ;実行前 R0=H'FFFFFFFE,R1=H'00005555
                     ;実行後 MACL=H'FFFF5556
STS      MACL,R0     ;演算結果を得る
```

10.1.42 MULU.W MULtiply as Unsigned Word 算術演算命令

符号なし乗算

書式	動作概略	命令コード	実行ステート	Tビット
MULU.W Rm,Rn	符号なしで Rn×Rm→MACL	0010nnnnmmmm1110	1	—

(1) 説明

汎用レジスタ Rn の内容と Rm を 16 ビットで乗算し、結果の 32 ビットを MACL レジスタに格納します。演算は符号なし算術演算で行います。MACH の内容は変化しません。

(2) 注意

特になし

(3) 動作内容

```
MULU(long m, long n) /* MULU Rm,Rn */
{
    MACL = ((unsigned long)(unsigned short)R[n]*
    (unsigned long)(unsigned short)R[m];
      PC += 2;
}
```

(4) 使用例

```
MULU.W   R0,R1       ;実行前 R0=H'00000002,R1=H'FFFFAAAA
                     ;実行後 MACL=H'00015554
STS      MACL,R0     ;演算結果を得る
```

10.1.43 NEG　　NEGate　　算術演算命令

符号反転

書式	動作概略	命令コード	実行ステート	T ビット
NEG Rm,Rn	0-Rm→Rn	0110nnnnmmmm1011	1	—

(1) 説明

汎用レジスタ Rm の内容の 2 の補数をとり、結果を Rn に格納します。即ち 0 から Rm を減算し、結果を Rn に格納します。

(2) 注意

特になし

(3) 動作内容

```
NEG(long m, long n) /* NEG Rm,Rn */
{
    R[n] = 0-R[m];
    PC += 2;
}
```

(4) 使用例

```
NEG R0,R1    ;実行前 R0=H'00000001
             ;実行後 R1=H'FFFFFFFF
```

10.1.44 NEGC NEGate with Carry 算術演算命令

ボロー付き符号反転

書式	動作概略	命令コード	実行ステート	T ビット
NEGC Rm,Rn	0-Rm-T→Rn, ボロー→T	0110nnnnmmmm1010	1	ボロー

(1) 説明

0 から汎用レジスタ Rm の内容と T ビットを減算し、結果を Rn に格納します。演算の結果によってボローを T ビットに反映します。32 ビットを超える値の符号反転を行うとき使用します。

(2) 注意

特になし

(3) 動作内容

```
NEGC(long m, long n) /* NEGC Rm,Rn */
{
     unsigned long temp;

     temp = 0-R[m];
     R[n] = temp-T;
     if(0<temp) T = 1;
     else T = 0;
     if(temp<R[n]) T = 1;
     PC += 2;
}
```

(4) 使用例

```
CLRT                ;R0:R1(64 ビット)の符号反転
NEGC   R1,R1        ;実行前 R1=H'00000001,T=0
                    ;実行後 R1=H'FFFFFFFF,T=1
NEGC   R0,R0        ;実行前 R0=H'00000000,T=1
                    ;実行後 R0=H'FFFFFFFF,T=1
```

10.1.45 NOP　No OPeration　システム制御命令

無操作

書式	動作概略	命令コード	実行ステート	T ビット
NOP	無操作	0000000000001001	1	—

(1) 説明

PC のインクリメントのみを行い、次の命令に実行を移します。

(2) 注意

特になし

(3) 動作内容

```
NOP(  ) /* NOP */
{
    PC += 2;
}
```

(4) 使用例

```
NOP          ;1 実行ステート分の時間が過ぎます。
```

10.1.46 NOT　NOT-logical complement　論理演算命令

ビット反転

書式	動作概略	命令コード	実行ステート	Tビット
NOT　Rm,Rn	~Rm→Rn	0110nnnnmmmm0111	1	—

(1) 説明

汎用レジスタ Rm の内容の 1 の補数をとり、結果を Rn に格納します。即ち Rm のビットを反転して Rn に格納します。

(2) 注意

特になし

(3) 動作内容

```
NOT(long m, long n) /* NOT Rm,Rn */
{
        R[n] = ~R[m];
        PC += 2;
}
```

(4) 使用例

```
NOT  R0,R1          ;実行前 R0=H'AAAAAAAA
                    ;実行後 R1=H'55555555
```

10.1.47 OCBI Operand Cache Block Invalidate データ転送命令

キャッシュブロックの無効化

書式	動作概略	命令コード	実行ステート	T ビット
OCBI @Rn	オペランドキャッシュブロックを無効にする	0000nnnn10010011	1	—

(1) 説明

実効アドレス Rn で示されている内容を使用して、データをアクセスします。キャッシュにヒットした場合、対応するキャッシュブロックを無効(Vbit=0)にします。このとき、たとえライトバック方式で、未書き込み情報あり(U bit=1)の場合でも、書き戻しはしません。キャッシュミスの場合や、非キャッシュ領域へのアクセスの場合は、動作しません。

(2) 注意

特になし

(3) 動作内容

```
OCBI(int n)       /* OCBI @Rn */
{
  invalidate_operand_cache_block(R[n]);
  PC += 2;
}
```

(4) 発生する可能性がある例外

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- 初期ページ書き込み例外
- データアドレスエラー

OCBI が動作しない場合でも、上記例外が発生しますので注意してください。

10.1.48 OCBP Operand Cache Block Purge データ転送命令

キャッシュブロックのパージ

書式	動作概略	命令コード	実行ステート	T ビット
OCBP @Rn	オペランドキャッシュブロックをライトバックし無効にする	0000nnnn10100011	1	—

(1) 説明

実効アドレス Rn で示されている内容を使用して、データをアクセスします。キャッシュにヒットして未書き込み情報あり(U bit=1)の場合 、対応するキャッシュブロックを外部メモリに書き戻して、そのブロックを無効(Vbit=0)にします。このとき、未書き込み情報 無し(U bit=0)の場合、単にそのブロックを無効にします。キャッシュミスの場合や、非キャッシュ領域へのアクセスの場合は、動作しません。

(2) 注意

特になし

(3) 動作内容

```
OCBP(int n)       /* OCBP @Rn */
{
  if(is_dirty_block(R[n]))  write_back(R[n])
  invalidate_operand_cache_block(R[n]);
  PC += 2;
}
```

(4) 発生する可能性がある例外

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- データアドレスエラー

OCBP が動作しない場合でも、上記例外が発生しますので注意してください。

10.1.49 OCBWB Operand Cache Block Write Back データ転送命令

キャッシュブロックの書き戻し

書式	動作概略	命令コード	実行ステート	T ビット
OCBWB @Rn	オペランドキャッシュブロックをライトバックする	0000nnnn10110011	1	—

(1) 説明

実効アドレス Rn で示されている内容を使用して、データをアクセスします。キャッシュにヒットして未書き込み情報あり(U bit=1)の場合 、対応するキャッシュブロックを外部メモリに書き戻して、そのブロックをクリーン(Ubit=0)にします。そのほかの場合つまり、キャッシュミスの場合や、すでにクリーンな場合、非キャッシュ領域へのアクセスの場合などは動作しません。

(2) 注意

特になし

(3) 動作内容

```
OCBWB(int n)      /* OCBWB @Rn */
    {
      if(is_dirty_block(R[n]))  write_back(R[n]);
      PC += 2;
    }
```

(4) 発生する可能性がある例外

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- データアドレスエラー

OCBWB が動作しない場合でも、上記例外が発生しますので注意してください。

10.1.50 OR　　OR logical　　論理演算命令

論理和演算

書式	動作概略	命令コード	実行ステート	T ビット
OR Rm,Rn	Rn	Rm → Rn	0010nnnnmmmm1011	1	—
OR #imm,R0	R0	imm → R0	11001011iiiiiiii	1	—
OR.B #imm,@(R0,GBR)	(R0+GBR)	imm → (R0+GBR)	11001111iiiiiiii	3	—

(1) 説明

汎用レジスタ Rn の内容と Rm の論理和をとり、結果を Rn に格納します。

汎用レジスタ R0 とゼロ拡張した 8 ビットのイミディエイトデータとの論理和、もしくはインデックス付き GBR 間接アドレッシングモードで 8 ビットのメモリと 8 ビットのイミディエイトデータとの論理和が可能です。

(2) 注意

特になし

(3) 動作内容

```
OR(long m, long n) /* OR Rm,Rn */
{
        R[n] |= R[m];
        PC += 2;
}

ORI(long i)  /* OR #imm,R0 */
{
        R[0] |= (0x000000FF & (long)i);
        PC += 2;
}

ORM(long i) /* OR.B #imm,@(R0,GBR) */
{
       long temp;

       temp = (long)Read_Byte(GBR+R[0]);
       temp |= (0x000000FF & (long)i);
       Write_Byte(GBR+R[0],temp);
```

```
        PC += 2;
}
```

(4) 使用例

```
OR      R0,R1               ;実行前 R0=H'AAAA5555,R1=H'55550000
                            ;実行後 R1=H'FFFF5555
OR      #H'F0,R0            ;実行前 R0=H'00000008
                            ;実行後 R0=H'000000F8
OR.B    #H'50,@(R0,GBR)     ;実行前 (R0+GBR)=H'A5
                            ;実行後 (R0+GBR)=H'F5
```

(5) 発生する可能性がある例外

OR.B 命令でのみ以下の例外が発生する可能性があります。

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- 初期ページ書き込み例外
- データアドレスエラー

例外処理において、本命令によるデータアクセスはバイトロード、バイトストアとして扱われます。

10.1.51 PREF PREFetch data to cache データ転送命令

データキャッシュへのプリフェッチ

書式	動作概略	命令コード	実行ステート	T ビット
PREF @Rn	(Rn)→オペランドキャッシュ	0000nnnn10000011	1	—

(1) 説明

32 バイト境界で始まる 32 バイトのデータブロックをオペランドキャッシュに読み込みます。Rn で指定したアドレスの下位 5 ビットは 0 にマスクされます。

この命令でデータアドレスエラーは発生しません。またデータ TLB 多重ビット例外を除く MMU 例外も発生しません。これらの例外条件に合致した場合には、NOP（無操作）命令として取り扱われます。

(2) 注意

特になし

(3) 動作内容

```
PREF(int n) /* PREF @Rn */
{
      PC += 2;
}
```

(4) 使用例

```
          MOV.L SOFT_PF,R1      ;R1 のアドレスは SOFT_PF
          PREF @R1              ;SOFT_PF のデータを内蔵キャッシュへロード

          .align    32
SOFT_PF:  .data.l   H'12345678
          .data.l   H'9ABCDEF0
          .data.l   H'AAAA5555
          .data.l   H'5555AAAA
          .data.l   H'11111111
          .data.l   H'22222222
          .data.l   H'33333333
          .data.l   H'44444444
```

(5) 発生する可能性のある例外

- データTLB多重ヒット例外

10.1.52 PREFI PREFetch Instruction cache block データ転送命令

命令キャッシュブロックの
プリフェッチ

書式	動作概略	命令コード	実行ステート	T ビット
PREFI @Rn	32 バイトの命令を命令キャッシュに読み込む	0000nnnn11010011	10	—

(1) 説明

32 バイト境界で始まる 32 バイトの命令ブロックを命令キャッシュに読み込みます。Rn で指定したアドレスの下位 5 ビットは 0 とみなします。

この命令でデータアドレスエラーおよび MMU 例外は発生しません。これらの例外条件に合致した場合には、NOP（無操作）命令として取り扱われます。

プリフェッチしようとしたアドレスが UTLB にミスした場合やプロテクションに違反した場合には、NOP（無操作）命令として取り扱われ、TLB 例外を発生させません。

(2) 注意

特になし

(3) 動作内容

```
PREFI(int n) /* PREFI @Rn */
{
      prefetch_instruction_cache_block(R[n]);
      PC += 2;
}
```

(4) 使用例

```
          MOVA    WakeUp,R0   ;WakeUp のアドレス→0
          PREFI   @R0         ;SLEEP 命令解除後の命令をプリフェッチする
          SLEEP
WakeUp：
          NOP
```

SLEEP 命令発行前に SLEEP 状態から復帰したときに実行する命令をキャッシュにプリフェッチします。

(5) 発生する可能性がある例外

- スロット不当命令例外

10.1.53 ROTCL ROTate with Carry Left シフト命令

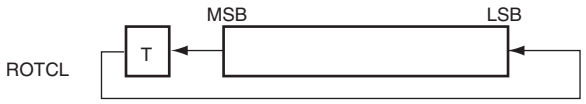
T ビット付き

1 ビット左回転

書式	動作概略	命令コード	実行ステート	T ビット
ROTCL Rn	T←Rn←T	0100nnnn00100100	1	MSB

(1) 説明

汎用レジスタ Rn の内容を左方向に T ビットを含めて 1 ビットローテート(回転)し、結果を Rn に格納します。ローテートしてオペランドの外に出てしまったビットは、T ビットへ転送します。



(2) 注意

特になし

(3) 動作内容

```
ROTCL(long n) /* ROTCL Rn */
{
        long temp;

        if((R[n]&0x80000000)==0) temp = 0;
        else temp = 1;
        R[n] <<= 1;
        if(T==1) R[n] |= 0x00000001;
        else R[n] &= 0xFFFFFFFE;
        if(temp==1) T = 1;
        else T = 0;
        PC += 2;
}
```

(4) 使用例

```
ROTCL  R0       ;実行前 R0=H'80000000,T=0
                ;実行後 R0=H'00000000,T=1
```

10.1.54 ROTCR ROTate with Carry Right シフト命令

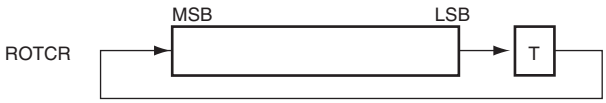
T ビット付き

1 ビット右回転

書式	動作概略	命令コード	実行ステート	T ビット
ROTCR Rn	T→Rn→T	0100nnnn00100101	1	LSB

(1) 説明

汎用レジスタ Rn の内容を右方向に T ビットを含めて 1 ビットローテート(回転)し、結果を Rn に格納します。ローテートしてオペランドの外に出てしまったビットは、T ビットへ転送します。



(2) 注意

特になし

(3) 動作内容

```
ROTCR(long n) /* ROTCR Rn */
{
        long temp;

        if((R[n]&0x00000001)==0) temp = 0;
        else temp = 1;
        R[n] >>= 1;
        if(T==1) R[n] |= 0x80000000;
        else R[n] &= 0x7FFFFFFF;
        if(temp==1) T = 1;
        else T = 0;
        PC += 2;
}
```

(4) 使用例

```
ROTCR R0     ;実行前 R0=H'00000001,T=1
             ;実行後 R0=H'80000000,T=1
```

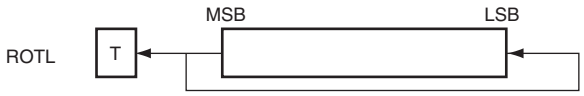
10.1.55 ROTL ROTate Left シフト命令

1 ビット左回転

書式	動作概略	命令コード	実行ステート	T ビット
ROTL Rn	T←Rn←MSB	0100nnnn00000100	1	MSB

(1) 説明

汎用レジスタ Rn の内容を左方向に 1 ビットローテート(回転)し、結果を Rn に格納します。ローテートしてオペランドの外に出てしまったビットは、T ビットへ転送します。



(2) 注意

特になし

(3) 動作内容

```
ROTL(long n)  /* ROTL Rn */
{
        if((R[n]&0x80000000)==0) T = 0;
        else T = 1;
        R[n] <<= 1;
        if(T==1) R[n] |= 0x00000001;
        else R[n] &= 0xFFFFFFFE;
        PC += 2;
}
```

(4) 使用例

```
ROTL R0      ;実行前 R0=H'80000000,T=0
             ;実行後 R0=H'00000001,T=1
```

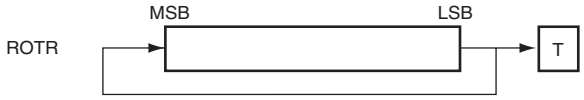
10.1.56 ROTR ROTate Right シフト命令

1 ビット右回転

書式	動作概略	命令コード	実行ステート	T ビット
ROTR Rn	LSB→Rn→T	0100nnnn00000101	1	LSB

(1) 説明

汎用レジスタ Rn の内容を右方向に 1 ビットローテート(回転)し、結果を Rn に格納します。ローテートしてオペランドの外に出てしまったビットは、T ビットへ転送します。



(2) 注意

特になし

(3) 動作内容

```
ROTR(long n) /* ROTR Rn */
{
      if((R[n]&0x00000001)==0) T = 0;
      else T = 1;
      R[n]>>=1;
      if(T==1) R[n] |= 0x80000000;
      else R[n] &= 0x7FFFFFFF;
      PC += 2;
}
```

(4) 使用例

```
ROTR R0      ;実行前 R0=H'00000001,T=0
             ;実行後 R0=H'80000000,T=1
```

10.1.57 RTE　ReTurn from Exception　システム制御命令

例外処理からの復帰　（特権命令）

遅延分岐命令

書式	動作概略	命令コード	実行ステート	T ビット
RTE	SSR→SR, SPC→PC	0000000000101011	4	—

(1) 説明

例外、割り込み処理ルーチンから復帰します。PC と SR の値を SPC と SSR から回復させます。プログラムは回復された PC の値で指定されるアドレスから続行されます。RTE 命令は特権命令なので特権モードでだけ使うことができます。もしユーザモードで使われた場合は不当命令例外が発生します。

(2) 注意

遅延分岐命令なので、この RTE 命令の次の命令を先に実行してから、分岐先の命令を実行します。

本命令と直後の命令との間には、割り込みを受け付けません。本命令の遅延スロット内の命令によって例外が発生してはなりません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。遅延分岐命令直後の遅延スロットに本命令が配置されたときは、スロット不当命令として認識します。RTE の遅延スロット中の命令によってアクセスした SR の内容は、RTE によって SSR から復帰した値です。ただし、RTE の実行前に定義済みの SR、MD の値は RTE の遅延スロット内の命令をフェッチするために使用します。

(3) 動作内容

```
RTE( ) /* RTE */
{
        unsigned int temp;
        temp = PC;
        SR = SSR;
        PC = SPC;
        Delay_Slot(temp+2);
}
```

(4) 使用例

```
RTE          ;元のルーチンへ復帰します。
ADD #8,R14   ;分岐に先立ち実行します。
```

【注】 遅延分岐においては分岐という動作そのものは、スロット命令の実行後に発生しますが、命令の実行（レジスタの更新など）は、あくまでも遅延分岐命令→遅延スロット命令の順に行われます。例えば遅延スロットで分岐先アドレスが格納されたレジスタを変更しても、変更前のレジスタ内容が分岐先アドレスとなります。

(5) 発生する可能性がある例外

- スロット不当命令例外
- 一般不当命令例外

10.1.58 RTS ReTurn from Subroutine 分岐命令

サブルーチンプロシージャからの復帰　　遅延分岐命令

書式	動作概略	命令コード	実行ステート	T ビット
RTS	PR→PC	0000000000001011	1	—

(1) 説明

サブルーチンプロシージャから復帰します。すなわち、PC を PR から復帰し、復帰した PC の示すアドレスから処理を続行します。本命令によって、BSR および JSR 命令でコールされたサブルーチンプロシージャからコール元へ戻ることができます。

(2) 注意

遅延分岐命令ですので、本命令の直後の命令を先に実行してから、分岐先の命令を実行します。

本命令と直後の命令との間には、割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

PR を復帰する命令は RTS 命令に先行しなければなりません。この復帰命令は RTS の遅延スロットであってはなりません。

(3) 動作内容

```
RTS( ) /* RTS */
{
        unsigned int temp;

        temp = PC;
        PC = PR;
        Delay_Slot(temp+2);
}
```

(4) 使用例

```
                MOV.L     TABLE,R3    ;R3=TRGET のアドレス
                JSR       @R3         ;TRGET へ分岐します。
                NOP                   ;分岐前に NOP を実行します。
                ADD       R0,R1       ;←サブルーチンプロシージャからの戻り先(PR の内容)
                ........
TABLE:          .data.l   TRGET       ;ジャンプテーブル
                ........
```

```
TRGET:      MOV       R1,R0       ;←プロシージャの入り口
            RTS                   ;PR の内容→PC
            MOV       #12,R0      ;分岐に先立ち MOV を実行します。
```

(5) 発生する可能性がある例外

- スロット不当命令例外

10.1.59 SETS　　SET Sbit　　システム制御命令

S ビットのセット

書式	動作概略	命令コード	実行ステート	T ビット
SETS	1→S	0000000001011000	1	−

(1) 説明

S ビットを 1 にセットします。

(2) 注意

特になし

(3) 動作内容

```
SETS( ) /* SETS */
{
        S=1;
        PC += 2;
}
```

(4) 使用例

```
SETS            ;実行前 S=0
                ;実行後 S=1
```

10.1.60 SETT　　SET Tbit　　システム制御命令

Ｔビットのセット

書式	動作概略	命令コード	実行ステート	Ｔビット
SETT	1→T	0000000000011000	1	1

(1) 説明

Ｔビットをセットします。

(2) 注意

特になし

(3) 動作内容

```
SETT( )  /* SETT */
{
        T = 1;
        PC += 2;
}
```

(4) 使用例

```
SETT        ;実行前 T=0
            ;実行後 T=1
```

10.1.61 SHAD SHift Arithemetic Dynamically シフト命令

ダイナミック算術

シフト

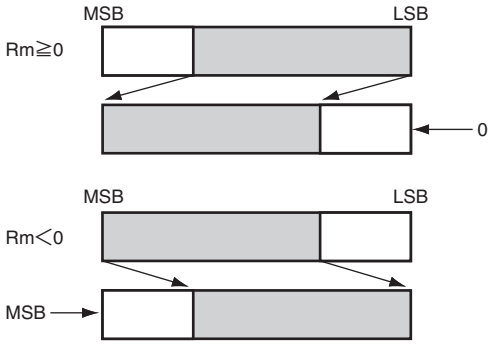
書式	動作概略	命令コード	実行ステート	T ビット
SHAD Rm, Rn	Rm≧0 のとき、Rn<<Rm→Rn Rm<0 のとき、 Rn>>Rm→[MSB→Rn]	0100nnnnmmmm1100	1	—

(1) 説明

汎用レジスタ Rn の内容を算術的にシフトします。汎用レジスタ Rm がシフトの方向とシフトするビット数を指定します。

Rm レジスタの値が正のとき左へシフトし、負のとき右へシフトします。右にシフトするときは上位に MSB が追加されます。

シフトするビット数は Rm レジスタの下位 5 ビット (ビット 4〜0) で指定されます。値が負（MSB=1）のとき、Rm レジスタは 2 の補数で表されています。左シフトのシフト量は 0〜31 で、右シフトのシフト量は 1〜32 です。



(2) 注意

特になし

(3) 動作内容

```
SHAD(int m,n) /*SHAD Rm,Rn */
{
        int sgn=R[m] & 0x80000000;
        if(sgn==0)
                R[n] <<= (R[m] & 0x1F);
        else if((R[m] & 0x1F) == 0) {
                if((R[n] & 0x80000000) == 0)
                     R[n] = 0;
                else
                     R[n] = 0xFFFFFFFF;
        }
        else  R[n] = (long)R[n] >> ((~R[m] & 0x1F)+1);
        PC += 2;
}
```

(4) 使用例

```
SHAD  R1,R2         ;実行前 R1=H'FFFFFFEC,R2=H'80180000
                    ;実行後 R1=H'FFFFFFEC,R2=H'FFFFF801
SHAD  R3,R4         ;実行前 R3=H'00000014,R4=H'FFFFF801
                    ;実行後 R3=H'00000014,R4=H'80100000
```

10.1.62 SHAL SHift Arithmetic Left シフト命令

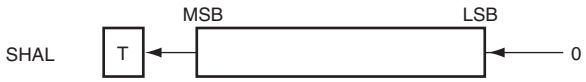
1 ビット左算術

シフト

書式	動作概略	命令コード	実行ステート	T ビット
SHAL Rn	T←Rn←0	0100nnnn00100000	1	MSB

(1) 説明

汎用レジスタ Rn の内容を左方向に算術的に 1 ビットシフトし、結果を Rn に格納します。シフトしてオペランドの外に出てしまったビットは、T ビットへ転送します。



(2) 注意

特になし

(3) 動作内容

```
SHAL(long n) /* SHAL Rn (Same as SHLL) */
{
        if((R[n]&0x80000000)==0) T = 0;
        else T = 1;
        R[n] <<= 1;
        PC += 2;
}
```

(4) 使用例

```
SHAL R0      ;実行前 R0=H'80000001,T=0
             ;実行後 R0=H'00000002,T=1
```

10.1.63 SHAR SHift Arithmetic Right シフト命令

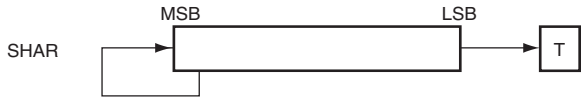
1 ビット右算術

シフト

書式	動作概略	命令コード	実行ステート	T ビット
SHAR Rn	MSB→Rn→T	0100nnnn00100001	1	LSB

(1) 説明

汎用レジスタ Rn の内容を右方向に算術的に 1 ビットシフトし、結果を Rn に格納します。シフトしてオペランドの外に出てしまったビットは、T ビットへ転送します。



(2) 注意

特になし

(3) 動作内容

```
SHAR(long n) /* SHAR Rn */
{
        long temp;

        if((R[n]&0x00000001)==0) T = 0;
        else T = 1;
        if((R[n]&0x80000000)==0) temp = 0;
        else temp = 1;
        R[n] >>= 1;
        if(temp==1) R[n] |= 0x80000000;
        else R[n] &= 0x7FFFFFFF;
        PC += 2;
}
```

(4) 使用例

```
SHAR R0      ;実行前 R0=H'80000001,T=0
             ;実行後 R0=H'C0000000,T=1
```

10.1.64 SHLD SHift Logical Dynamically シフト命令

ダイナミック論理

シフト

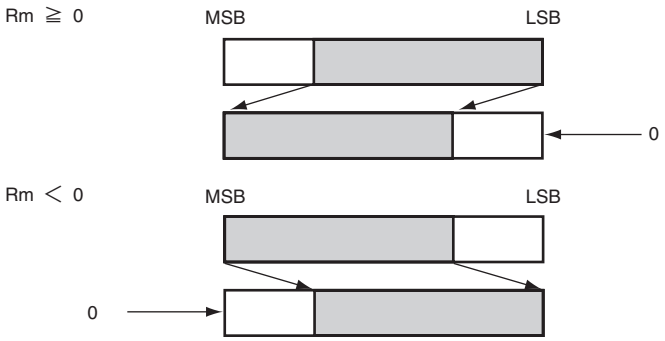
書式	動作概略	命令コード	実行ステート	T ビット
SHLD Rm, Rn	Rm≧0 のとき、Rn<<Rm→Rn Rm<0 のとき、 Rn>>Rm→[0→Rn]	0100nnnnmmmm1101	1	—

(1) 説明

汎用レジスタ Rn の内容を論理的にシフトします。汎用レジスタ Rm がシフトの方向とシフトするビット数を指定します。

Rm レジスタの値が正のとき左へシフトし、負のとき右へシフトします。右にシフトするときは上位に 0 が追加されます。

シフトするビット数は Rm レジスタの下位 5 ビット（ビット 4～0）で指定されます。値が負（MSB=1）のとき、Rm レジスタは 2 の補数で表されています。左シフトのシフト量は 0～31 で、右シフトのシフト量は 1～32 です。



(2) 注意

特になし

(3) 動作内容

```
SHLD(int m,n)/*SHLD Rm,Rn */
{
    int sgn = R[m] & 0x80000000;
    if(sgn == 0)
            R[n] <<= (R[m] & 0x1F);
    else if((R[m] & 0x1F) == 0)
            R[n] = 0;
    else
            R[n] = (unsigned)R[n] >> ((~R[m] & 0x1F) + 1);
    PC += 2;
}
```

(4) 使用例

```
SHLD R1, R2          ;実行前   R1=H'FFFFFFEC, R2=H'80180000
                     ;実行後   R1=H'FFFFFFEC, R2=H'00000801
SHLD R3, R4          ;実行前   R3=H'00000014, R4=H'FFFFF801
                     ;実行後   R3=H'00000014, R4=H'80100000
```

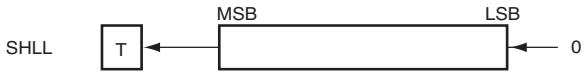
10.1.65 SHLL SHift Logical Left シフト命令

1 ビット左論理
シフト

書式	動作概略	命令コード	実行ステート	T ビット
SHLL Rn	T←Rn←0	0100nnnn00000000	1	MSB

(1) 説明

汎用レジスタ Rn の内容を左方向に論理的に 1 ビットシフトし、結果を Rn に格納します。シフトしてオペランドの外に出てしまったビットは、T ビットへ転送します。



(2) 注意

特になし

(3) 動作内容

```
SHLL(long n) /* SHLL Rn  (Same as SHAL) */
{
    if((R[n]&0x80000000)==0) T = 0;
    else T = 1;
    R[n] <<= 1;
    PC += 2;
}
```

(4) 使用例

```
SHLL R0      ;実行前          R0=H'80000001,T=0
             ;実行後          R0=H'00000002,T=1
```

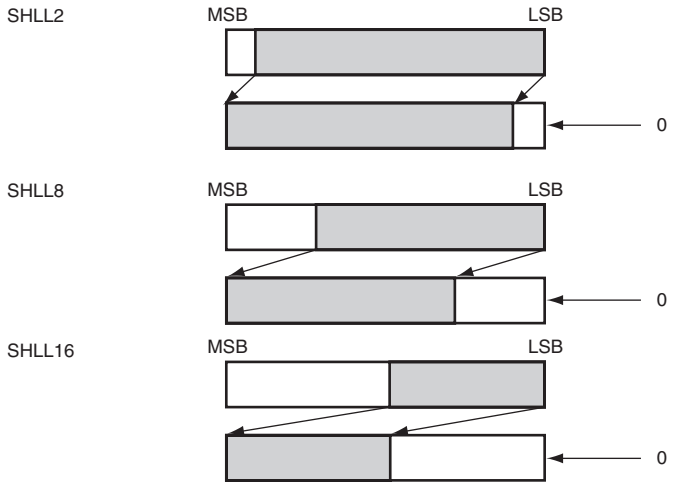
10.1.66 SHLLn n bits SHift Logical Left シフト命令

n ビット左論理
シフト

書式	動作概略	命令コード	実行ステート	T ビット
SHLL2 Rn	Rn<<2 → Rn	0100nnnn00001000	1	—
SHLL8 Rn	Rn<<8 → Rn	0100nnnn00011000	1	—
SHLL16 Rn	Rn<<16 → Rn	0100nnnn00101000	1	—

(1) 説明

汎用レジスタ Rn の内容を左方向に論理的に 2/8/16 ビットシフトし、結果を Rn に格納します。シフトしてオペランドの外に出てしまったビットは捨てます。



(2) 注意

特になし

(3) 動作内容

```
SHLL2(long n) /* SHLL2 Rn */
{
    R[n] <<= 2;
    PC += 2;
}

SHLL8(long n) /* SHLL8 Rn */
{
    R[n] <<= 8;
    PC += 2;
}
SHLL16(long n) /* SHLL16 Rn */
{
    R[n] <<= 16;
    PC += 2;
}
```

(4) 使用例

```
SHLL2  R0    ;実行前  R0=H'12345678
             ;実行後  R0=H'48D159E0
SHLL8  R0    ;実行前  R0=H'12345678
             ;実行後  R0=H'34567800
SHLL16  R0   ;実行前  R0=H'12345678
             ;実行後  R0=H'56780000
```

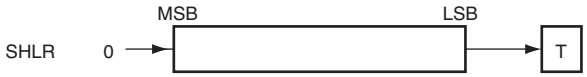
10.1.67 SHLR SHift Logical Right シフト命令

1 ビット右論理
シフト

書式	動作概略	命令コード	実行ステート	T ビット
SHLR Rn	0→Rn→T	0100nnnn00000001	1	LSB

(1) 説明

汎用レジスタ Rn の内容を右方向に論理的に 1 ビットシフトし、結果を Rn に格納します。シフトしてオペランドの外に出てしまったビットは、T ビットへ転送します。



(2) 注意

特になし

(3) 動作内容

```
SHLR(long n)   /* SHLR Rn */
{
    if ((R[n]&0x00000001)==0) T = 0;
    else T = 1;
    R[n] >>= 1;
    R[n] &= 0x7FFFFFFF;
    PC += 2;
}
```

(4) 使用例

```
SHLR  R0     ;実行前  R0=H'80000001,T=0
             ;実行後  R0=H'40000000,T=1
```

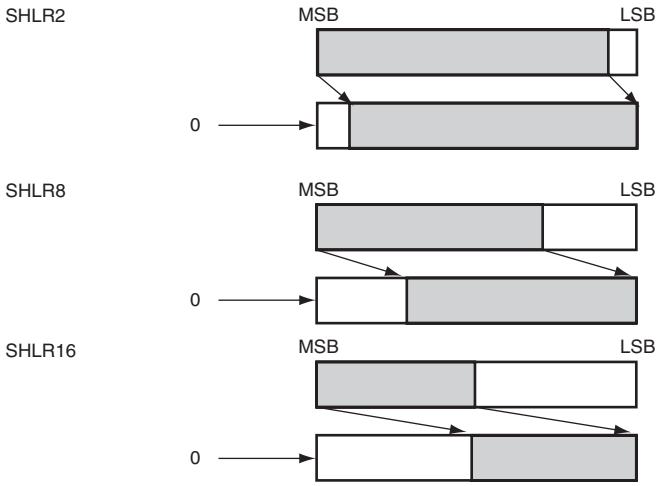
10.1.68 SHLRn n bits SHift Logical Right シフト命令

n ビット右論理
シフト

書式	動作概略	命令コード	実行ステート	T ビット
SHLR2 Rn	Rn>>2→Rn	0100nnnn00001001	1	—
SHLR8 Rn	Rn>>8→Rn	0100nnnn00011001	1	—
SHLR16 Rn	Rn>>16→Rn	0100nnnn00101001	1	—

(1) 説明

汎用レジスタ Rn の内容を右方向に論理的に 2/8/16 ビットシフトし、結果を Rn に格納します。シフトしてオペランドの外に出てしまったビットは捨てます。



(2) 注意

特になし

(3) 動作内容

```
SHLR2(long n)         /* SHLR2 Rn */
{
    R[n] >>= 2;
    R[n] &= 0x3FFFFFFF;
    PC += 2;
}
SHLR8(long n)         /* SHLR8 Rn */
{
    R[n] >>= 8;
    R[n] &= 0x00FFFFFF;
    PC += 2;
}

SHLR16(long n)        /* SHLR16 Rn */
{
    R[n] >>= 16;
    R[n] &= 0x0000FFFF;
    PC += 2;
}
```

(4) 使用例

```
SHLR2  R0   ;実行前  R0=H'12345678
            ;実行後  R0=H'048D159E
SHLR8  R0   ;実行前  R0=H'12345678
            ;実行後  R0=H'00123456
SHLR16  R0  ;実行前  R0=H'12345678
            ;実行後  R0=H'00001234
```

10.1.69 SLEEP　　SLEEP　　システム制御命令

低消費電力モードへの遷移　　(特権命令)

書式	動作概略	命令コード	実行ステート	T ビット
SLEEP	スリープもしくはスタンバイ	0000000000011011	不定	—

(1) 説明

CPU を低消費電力状態にします。

低消費電力モードでは、CPU の内部状態を保持し、直後の命令の実行を停止し、割り込み要求の発生を待ちます。要求が発生すると、低消費電力状態から抜けます。

SLEEP 命令は特権命令なので、特権モードでだけ使うことができます。もしユーザモードで使われた場合は、不当命令例外が発生します。

(2) 注意

SLEEP の性能は STBCR（スタンバイコントロールレジスタ）に依存します。当該製品ハードウェアマニュアルの**「低消費電力モード」**を参照してください。

(3) 動作内容

```
SLEEP( )    /* SLEEP */
{
    Sleep_standby();
}
```

(4) 使用例

```
SLEEP          ;低消費電力モードへの遷移
```

(5) 発生する可能性がある例外

- スロット不当命令例外
- 一般不当命令例外

10.1.70 STC STore Control register システム制御命令

コントロールレジスタからのストア （特権命令）

書式	動作概略	命令コード	実行ステート	T ビット
STC GBR, Rn	GBR→Rn	0000nnnn00010010	1	—
STC VBR, Rn	VBR→Rn	0000nnnn00100010	1	—
STC SSR , Rn	SSR→Rn	0000nnnn00110010	1	—
STC SPC, Rn	SPC→Rn	0000nnnn01000010	1	—
STC SGR, Rn	SGR→Rn	0000nnnn00111010	1	—
STC DBR , Rn	DBR→Rn	0000nnnn11111010	1	—
STC R0_BANK, Rn	R0_BANK→Rn	0000nnnn10000010	1	—
STC R1_BANK, Rn	R1_BANK→Rn	0000nnnn10010010	1	—
STC R2_BANK, Rn	R2_BANK→Rn	0000nnnn10100010	1	—
STC R3_BANK, Rn	R3_BANK→Rn	0000nnnn10110010	1	—
STC R4_BANK, Rn	R4_BANK→Rn	0000nnnn11000010	1	—
STC R5_BANK, Rn	R5_BANK→Rn	0000nnnn11010010	1	—
STC R6_BANK, Rn	R6_BANK→Rn	0000nnnn11100010	1	—
STC R7_BANK, Rn	R7_BANK→Rn	0000nnnn11110010	1	—
STC.L GBR, @-Rn	Rn-4→Rn, GBR→(Rn)	0100nnnn00010011	1	—
STC.L VBR, @-Rn	Rn-4→Rn, VBR→(Rn)	0100nnnn00100011	1	—
STC.L SSR , @-Rn	Rn-4→Rn, SSR→(Rn)	0100nnnn00110011	1	—
STC.L SPC, @-Rn	Rn-4→Rn, SPC→(Rn)	0100nnnn01000011	1	—
STC.L SGR, @-Rn	Rn-4→Rn, SGR→(Rn)	0100nnnn00110010	1	—
STC.L DBR , @-Rn	Rn-4→Rn, DBR→(Rn)	0100nnnn11110010	1	—
STC.L R0_BANK, @-Rn	Rn-4→Rn, R0_BANK→(Rn)	0100nnnn10000011	1	—
STC.L R1_BANK, @-Rn	Rn-4→Rn, R1_BANK→(Rn)	0100nnnn10010011	1	—
STC.L R2_BANK, @-Rn	Rn-4→Rn, R2_BANK→(Rn)	0100nnnn10100011	1	—
STC.L R3_BANK, @-Rn	Rn-4→Rn, R3_BANK→(Rn)	0100nnnn10110011	1	—
STC.L R4_BANK, @-Rn	Rn-4→Rn, R4_BANK→(Rn)	0100nnnn11000011	1	—
STC.L R5_BANK, @-Rn	Rn-4→Rn, R5_BANK→(Rn)	0100nnnn11010011	1	—
STC.L R6_BANK, @-Rn	Rn-4→Rn, R6_BANK→(Rn)	0100nnnn11100011	1	—
STC.L R7_BANK, @-Rn	Rn-4→Rn, R7_BANK→(Rn)	0100nnnn11110011	1	—

(1) 説明

コントロールレジスタ GBR、VBR、SSR、SPC、SGR、DBR、Rm_BANK(m=0～7)をデスティネーションに格納します。Rm_BANK オペランドは SR レジスタの RB ビットで指定します。RB ビットが 1 のとき Rm_BANK0 レジスタが、RB ビットが 0 のとき Rm_BANK1 レジスタが STC/STC.L 命令でアクセスされます。

(2) 注意

STC GBR,Rn/STC.L GBR,@-Rn を除く STC/STC.L 命令は特権モードの場合だけ使用可能です。ユーザモードで使用すると、不当命令例外が発生します。

(3) 動作内容

```
STCGBR(int n)     /* STC GBR,Rn */
{
      R[n] = GBR;
      PC += 2;
}
STCVBR(int n)     /* STC VBR,Rn : Privileged */
{
      R[n] = VBR;
      PC += 2;
}
STCSSR(int n)     /* STC SSR,Rn : Privileged */
{
      R[n] = SSR;
      PC += 2;
}
STCSPC(int n)     /* STC SPC,Rn : Privileged */
{
      R[n] = SPC;
      PC += 2;
}
STCSGR(int n)     /* STC SGR,Rn : Privileged */
{
      R[n] = SGR;
      PC += 2;
}
STCDBR(int n)     /* STC DBR,Rn : Privileged */
{
      R[n] = DBR;
      PC += 2;
}
STCRm_BANK(int n) /* STC Rm_BANK,Rn : Privileged */
                  /* m=0-7 */
{
```

```
      R[n] = Rm_BANK;
      PC += 2;
}
STCMGBR(int n)    /* STC.L GBR,@-Rn */
{
      R[n] -= 4;
      Write_Long(R[n],GBR);
      PC += 2;
}
STCMVBR(int n)    /* STC.L VBR,@-Rn : Privileged */
{
      R[n] -= 4;
      Write_Long(R[n],VBR);
      PC += 2;
}
STCMSSR(int n)    /* STC.L SSR,@-Rn : Privileged */
{
      R[n] -= 4;
      Write_Long(R[n],SSR);
      PC += 2;
}
STCMSPC(int n)    /* STC.L SPC,@-Rn : Privileged */
{
      R[n] -= 4;
      Write_Long(R[n],SPC);
      PC += 2;
}
STCMSGR(int n)    /* STC.L SGR,@-Rn : Privileged */
{
      R[n] -= 4;
      Write_Long(R[n],SGR);
      PC += 2;
}
STCMDBR(int n)    /* STC.L DBR,@-Rn : Privileged */
{
      R[n] -= 4;
      Write_Long(R[n],DBR);
      PC += 2;
```

```
}
STCMRm_BANK(int n)     /* STC.L Rm_BANK,@-Rn : Privileged */
                                /* m=0-7 */
{
      R[n] -= 4;
      Write_Long(R[n],Rm_BANK)
      PC += 2;
}
```

(4) 発生する可能性がある例外

- データTLB多重ヒット例外
- 一般不当命令例外
- スロット不当命令例外
- データTLBミス例外
- データTLB保護違反例外
- 初期ページ書き込み例外
- データアドレスエラー

10.1.71 STS　　STore System register　　システム制御命令

システムレジスタからのストア

書式		動作概略	命令コード	実行ステート	Tビット
STS	MACH,Rn	MACH→Rn	0000nnnn00001010	1	—
STS	MACL,Rn	MACL→Rn	0000nnnn00011010	1	—
STS	PR,Rn	PR→Rn	0000nnnn00101010	1	—
STS.L	MACH,@-Rn	Rn-4→Rn, MACH→(Rn)	0100nnnn00000010	1	—
STS.L	MACL,@-Rn	Rn-4→Rn, MACL→(Rn)	0100nnnn00010010	1	—
STS.L	PR,@-Rn	Rn-4→Rn, PR→(Rn)	0100nnnn00100010	1	—

(1) 説明

システムレジスタ MACH、MACL、PR をデスティネーションに格納します。

(2) 注意

特になし

(3) 動作内容

```
STSMACH(int n)      /* STC MACH,Rn */
    {
      R[n] = MACH;
      PC += 2;
    }
STSMACL(int n)      /* STC MACL,Rn */
    {
      R[n] = MACL;
      PC += 2;
    }
STSPR(int n)     /* STS PR,Rn */
    {
      R[n] = PR;
      PC += 2;
    }
STSMACH(int n)      /* STS.L MACH,@-Rn */
    {
      R[n] -= 4;
```

```
      Write_Long(R[n],MACH);
PC += 2;
    }
STSMACL(int n)      /* STS.L MACL,@-Rn */
    {
      R[n] -= 4;
      Write_Long(R[n],MACL);
PC += 2;
    }
STSMPR(int n)     /* STS.L PR,@-Rn */
    {
      R[n] -= 4;
Write_Long(R[n],PR);
PC += 2;
    }
```

(4) 使用例

```
STS     MACH,R0          ;実行前 R0=H'FFFFFFFF,MACH-H'00000000
                         ;実行後 R0=H'00000000
STS.L   PR,@-R15         ;実行前 R15=H'10000004
                         ;実行後 R15=H'10000000,(R15)=PR
```

(5) 発生する可能性がある例外

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- 初期ページ書き込み例外
- データアドレスエラー

10.1.72 SUB　SUBtract binary　算術演算命令

2 進減算

書式	動作概略	命令コード	実行ステート	T ビット
SUB Rm,Rn	Rn-Rm→Rn	0011nnnnmmmm1000	1	—

(1) 説明

汎用レジスタ Rn の内容から Rm を減算し、結果を Rn に格納します。イミディエイトデータとの減算は ADD #imm,Rn を使います。

(2) 注意

特になし

(3) 動作内容

```
SUB(long m, long n) /* SUB Rm,Rn */
{
   R[n] -= R[m];
   PC += 2;
}
```

(4) 使用例

```
SUB    R0,R1 ;実行前  R0=H'00000001,R1=H'80000000
             ;実行後  R1=H'7FFFFFFF
```

10.1.73 SUBC SUBtract with Carry 算術演算命令

ボロー付き 2 進減算

書式	動作概略	命令コード	実行ステート	T ビット
SUBC Rm,Rn	Rn-Rm-T→Rn, ボロー→T	0011nnnnmmmm1010	1	ボロー

(1) 説明

汎用レジスタ Rn の内容から Rm と T ビットを減算し、結果を Rn に格納します。演算の結果によってボローを T ビットに反映します。32 ビットを超える減算を行うとき使用します。

(2) 注意

特になし

(3) 動作内容

```
SUBC(long m, long n) /* SUBC Rm,Rn */
{
    unsigned long tmp0,tmp1;

    tmp1 = R[n]-R[m];
    tmp0 = R[n];
    R[n] = tmp1-T;
    if(tmp0<tmp1) T = 1;
    else T = 0;
    if(tmp1<R[n]) T = 1;
    PC += 2;
}
```

(4) 使用例

```
CLRT            ;R0:R1(64 ビット)-R2:R3(64 ビット)=R0:R1(64 ビット)
SUBC  R3,R1     ;実行前  T=0,R1=H'00000000,R3=H'00000001
                ;実行後  T=1,R1=H'FFFFFFFF
SUBC  R2,R0     ;実行前  T=1,R0=H'00000000,R2=H'00000000
                ;実行後  T=1,R0=H'FFFFFFFF
```

10.1.74 SUBV SUBtract with (Vflag)underflow check 算術演算命令

アンダフロー付き 2 進減算

書式	動作概略	命令コード	実行ステート	T ビット
SUBV Rm,Rn	Rn-Rm→Rn, アンダフロー→T	0011nnnnmmmm1011	1	アンダフロー

(1) 説明

汎用レジスタ Rn の内容から Rm を減算し、結果を Rn に格納します。アンダフローが発生すると、T ビットをセットします。

(2) 注意

特になし

(3) 動作内容

```
SUBV(long m, long n) /* SUBV Rm,Rn */
{
    long dest,src,ans;

    if((long)R[n]>=0) desT = 0;
    else desT = 1;
    if((long)R[m]>=0) src = 0;
    else src = 1;
    src += dest;
    R[n] -= R[m];
    if((long)R[n]>=0) anS = 0;
    else ans = 1;
    ans += dest;
    if(src==1) {
        if(ans==1) T = 1;
        else T = 0;
    }
    else T = 0;
    PC += 2;
}
```

(4) 使用例

```
SUBV  R0,R1    ;実行前  R0=H'00000002,R1=H'80000001
               ;実行後  R1=H'7FFFFFFF,T=1
SUBV  R2,R3    ;実行前  R2=H'FFFFFFFE,R3=H'7FFFFFFE
               ;実行後  R3=H'80000000,T=1
```

10.1.75 SWAP　　SWAP register halves　　データ転送命令

上位と下位の交換

書式	動作概略	命令コード	実行ステート	T ビット
SWAP.B Rm,Rn	Rm→下位 2 バイトの上下バイト交換→Rn	0110nnnnmmmm1000	1	—
SWAP.W Rm,Rn	Rm→上下ワード交換→Rn	0110nnnnmmmm1001	1	—

(1) 説明

汎用レジスタ Rm の内容の上位と下位を交換して、結果を Rn に格納します。

バイト指定のとき、Rm のビット 15 からビット 8 の 8 ビットと、ビット 7 からビット 0 の 8 ビットを交換します。Rn の上位 16 ビットには Rm の上位 16 ビットをそのまま転送します。

ワード指定のとき、Rm のビット 31 からビット 16 の 16 ビットと、ビット 15 からビット 0 の 16 ビットを交換します。

(2) 注意

特になし

(3) 動作内容

```
SWAPB(long m, long n)      /* SWAP.B Rm,Rn */
{
   unsigned long temp0,temp1;

   temp0 = R[m]&0xFFFF0000;
   temp1 = (R[m]&0x000000FF)<<8;
   R[n] = (R[m]&0x0000FF00)>>8;
   R[n] = R[n]|temp1|temp0;
   PC += 2;
}

SWAPW(long m, long n)      /* SWAP.W Rm,Rn */
{
   unsigned long temp;

   temp = (R[m]>>16)&0x0000FFFF;
   R[n] = R[m]<<16;
```

```
    R[n] |= temp;
    PC += 2;
}
```

(4) 使用例

```
SWAP.B  R0,R1          ;実行前  R0=H'12345678
                       ;実行後  R1=H'12347856
SWAP.W  R0,R1          ;実行前  R0=H'12345678
                       ;実行後  R1=H'56781234
```

10.1.76 SYNCO SYNChronize data Operation データ転送命令

データ操作の同期

書式	動作概略	命令コード	実行ステート	Tビット
SYNCO	本命令に先行するバスアクセスの完了まで、本命令以降の命令によるデータアクセスを開始しません。	0000000010101011	不定	—

(1) 説明

本命令はデータ操作の同期に使用します。本命令を実行すると、本命令に先行するバスアクセスの完了を待ってから、本命令より後の命令によるデータアクセスを開始します。

(2) 注意

バスアクセスによるデータ更新結果がバス以外により CPU に通知されるような場合、すなわちアドレスマップされたハードウェアレジスタの反映などについては、SYNCO 命令の挿入だけでは順序付けが保証されないことがあります。この場合、順序保証のための条件は各レジスタの項目を個別に参照してください。

(3) 動作内容

```
SYNCO /* SYNCO*/
{
      synchronize_data_operaiton();
      PC += 2;
}
```

(4) 使用例

1. 他メモリユーザと共有したメモリへのアクセスの順序付け
2. すべてのライトバッファのフラッシュ
3. メモリアクセスのマージ、消滅の防止
4. キャッシュ操作命令の完了待ち

10.1.77 TAS　Test And Set　論理演算命令

メモリテストと
ビットセット

書式	動作概略	命令コード	実行ステート	T ビット
TAS.B @Rn	(Rn)が 0 のとき 1→T,それ以外 0→T 両方に対して 1→(Rn)の MSB	0100nnnn00011011	4	テスト結果

(1) 説明

汎用レジスタ Rn の内容で指定したメモリ領域に対し、本命令は該当するキャッシュブロックをパージし、そのアドレスの示すバイトデータを読み込み、そのデータがゼロのとき T=1、ゼロでないとき T=0 とします。その後、ビット 7 を 1 にセットして同じアドレスへ書き込みます。この間、バス権は解放しません。

この場合、パージ動作は次のように実行します。

パージ動作は実効アドレスとして汎用レジスタ Rn の内容によりデータにアクセスします。キャッシュヒットが存在し、該当するキャッシュブロックがダーティ（U ビット=1）の場合、そのキャッシュブロックの内容は外部メモリにライトバックされた後、キャッシュブロックは無効になります（V ビット=0）。キャッシュヒットが存在し、該当するキャッシュブロックがクリーン（U ビット=0）の場合、キャッシュブロックは無効になるだけです（V ビット=0）。キャッシュミスが発生した場合、またはアクセスするメモリ位置がキャッシュ不可の場合、パージは実行されません。

TAS.B の 2 つのメモリアクセスは自動的に実行されます。TAS.B の 2 つのアクセスの間では他のメモリアクセスは実行されません。

(2) 注意

特になし

(3) 動作内容

```
TAS(int n) /* TAS.B @Rn */
{
    int temp;

    temp = (int)Read_Byte(R[n]); /* Bus Lock */
    if(temp==0) T = 1;
    else T = 0;
    temp |= 0x00000080;
    Write_Byte(R[n],temp);    /* Bus unlock */
    PC += 2;
}
```

(4) 発生する可能性がある例外

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- 初期ページ書き込み例外
- データアドレスエラー

例外処理において、本命令によるデータアクセスはバイトロード、バイトストアとして扱われます。

10.1.78 TRAPA TRAP Always システム制御命令

トラップ例外処理

書式	動作概略	命令コード	実行ステート	T ビット
TRAPA #imm	Imm<<2→TRA, PC+2→SPC, SR→SSR,R15→SGR, 1→SR.MD/BL/RB, H'160→EXPEVT, VBR+H'0100→PC	11000011iiiiiiii	13	—

(1) 説明

トラップ例外処理を開始します。（PC+2）と SR と R15 の値が SPC と SSR と SGR に退避され、8 ビットイミディエートデータが TRA レジスタ（ビット 9～2）に格納されます。処理モードは特権モード（SR の MD ビットが 1）に切り替わり、SR の BL ビットと RB ビットが 1 になります。これにより、例外と割り込みの要求をマスクして受け付けず、BANK1 レジスタ（R0_BANK1～R7_BANK1）が選択されます。例外コード H'160 が EXPEVT レジスタ（ビット 11～0）に書き込まれます。プログラムは VBR レジスタとオフセット H'00000100 の和で表されるアドレス（VBR+H'00000100）に分岐します。

(2) 注意

特になし

(3) 動作内容

```
TRAPA(int i) /* TRAPA #imm */
{
    int imm;
    imm = (0x000000FF & i);
    TRA = imm << 2;
    SSR = SR;
    SPC = PC + 2;
    SGR = R15;
    SR.MD = 1;
    SR.BL = 1;
    SR.RB = 1;
    EXPEVT = H'00000160;
    PC = VBR + H'00000100;
}
```

(4) 発生する可能性がある例外

- スロット不当命令例外
- 無条件トラップ

10.1.79 TST　TeST logical　論理演算命令

論理積演算の
T ビットセット

書式	動作概略	命令コード	実行ステート	T ビット
TST Rm,Rn	Rn & Rm、結果が 0 のとき 1→T その他 0→T	0010nnnnmmmm1000	1	テスト結果
TST #imm,R0	R0 & imm、結果が 0 のとき 1→T その他 0→T	11001000iiiiiiii	1	テスト結果
TST.B #imm,@(R0,GBR)	(R0+GBR)&imm、結果が 0 のとき 1→T　その他 0→T	11001100iiiiiiii	3	テスト結果

(1) 説明

汎用レジスタ Rn の内容と Rm の論理積をとり、結果がゼロのとき T ビットをセットします。結果がゼロでないとき T ビットをクリアします。Rn の内容は変更しません。

汎用レジスタ R0 とゼロ拡張した 8 ビットのイミディエイトデータとの論理積、もしくはインデックス付き GBR 間接アドレッシングモードで 8 ビットのメモリと 8 ビットのイミディエイトデータとの論理積が可能です。R0、もしくはメモリの内容は変更しません。

(2) 注意

特になし

(3) 動作内容

```
TST(long m, long n) /* TST Rm,Rn */
{
    if((R[n]&R[m])==0) T = 1;
    else T = 0;
    PC += 2;
}

TSTI(long i)                    /* TST #imm,R0 */
{
    long temp;

    temp = R[0] & (0x000000FF & (long)i);
    if(temp==0) T = 1;
    else T = 0;
```

```
    PC += 2;
}
TSTM(long i) /* TST.B #imm,@(R0,GBR) */
{
    long temp;

    temp = (long)Read_Byte(GBR+R[0]);
    temp &= (0x000000FF & (long)i);
    if(temp==0) T = 1;
    else T = 0;
    PC += 2;
}
```

(4) 使用例

```
TST     R0,R0               ;実行前R0=H'00000000
                            ;実行後T=1
TST     #H'80,R0            ;実行前R0=H'FFFFFF7F
                            ;実行後T=1
TST.B   #H'A5,@(R0,GBR)     ;実行前(R0,GBR)=H'A5
                            ;実行後T=0
```

(5) 発生する可能性がある例外

TST.B 命令のみで以下の例外が発生する可能性があります。

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- 初期ページ書き込み例外
- データアドレスエラー

例外処理において、本命令によるデータアクセスはバイトロード、バイトストアとして扱われます。

10.1.80 XOR　eXclusive OR logical　論理演算命令

排他的論理和演算

書式	動作概略	命令コード	実行ステート	T ビット
XOR Rm,Rn	Rn ^ Rm → Rn	0010nnnnmmmm1010	1	—
XOR #imm,R0	R0 ^ imm → R0	11001010iiiiiiii	1	—
XOR.B #imm,@(R0,GBR)	(R0+GBR)^imm → (R0+GBR)	11001110iiiiiiii	3	—

(1) 説明

汎用レジスタ Rn の内容と Rm の排他的論理和をとり、結果を Rn に格納します。

汎用レジスタ R0 とゼロ拡張した 8 ビットのイミディエイトデータとの排他的論理和、もしくはインデックス付き GBR 間接アドレッシングモードで 8 ビットのメモリと 8 ビットのイミディエイトデータとの排他的論理和が可能です。

(2) 注意

特になし

(3) 動作内容

```
XOR(long m, long n) /* XOR Rm,Rn */
{
    R[n] ^= R[m];
    PC += 2;
}

XORI(long i)  /* XOR #imm,R0 */
{
    R[0] ^= (0x000000FF & (long)i);
    PC += 2;
}

XORM(long i)  /* XOR.B #imm,@(R0,GBR) */
{
    int temp;

    temp = (long)Read_Byte(GBR+R[0]);
    temp ^= (0x000000FF & (long)i);
```

```
    Write_Byte(GBR+R[0],temp);
    PC += 2;
}
```

(4) 使用例

```
XOR R0,R1               ;実行前  R0=H'AAAAAAAA,R1=H'55555555
                        ;実行後  R1=H'FFFFFFFF
XOR #H'F0,R0            ;実行前  R0=H'FFFFFFFF
                        ;実行後  R0=H'FFFFFF0F
XOR.B #H'A5,@(R0,GBR)   ;実行前  (R0,GBR)=H'A5
                        ;実行後  (R0,GBR)=H'00
```

(5) 発生する可能性がある例外

XOR.B 命令のみで以下の例外が発生する可能性があります。

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- 初期ページ書き込み例外
- データアドレスエラー

例外処理において、本命令によるデータアクセスはバイトロード、バイトストアとして扱われます。

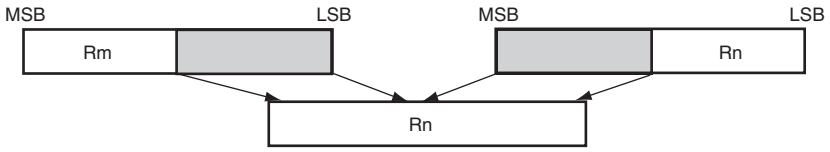
10.1.81 XTRCT eXTRaCT データ転送命令

連結レジスタの
中央切り出し

書式	動作概略	命令コード	実行ステート	T ビット
XTRCT Rm,Rn	Rm:Rn の中央 32 ビット→Rn	0010nnnnmmmm1101	1	—

(1) 説明

汎用レジスタ Rm と Rn とを連結した 64 ビットの内容から中央の 32 ビットを切り出し、結果を Rn に格納します。



(2) 注意

特になし

(3) 動作内容

```
XTRCT(long m, long n)     /* XTRCT Rm,Rn */
{
    unsigned long temp;

    temp = (R[m]<<16) & 0xFFFF0000;
    R[n] = (R[n]>>16) & 0x0000FFFF;
    R[n] |= temp;
    PC += 2;
}
```

(4) 使用例

```
XTRCT  R0,R1          ;実行前  R0=H'01234567,R1=H'89ABCDEF
                      ;実行後  R1=H'456789AB
```

10.2 CPU 命令（FPU 関係）

CPU 命令のうち、FPU をサポートする CPU 命令、および SH-4A、SH4AL-DSP で機能の一部に差分のある命令を本節に記載します。

10.2.1 BSR　Branch to SubRoutine　分岐命令

サブルーチンプロシージャへの分岐　遅延分岐命令

書式	動作概略	命令コード	実行ステート	T ビット
BSR　label	PC+4→PR, PC+4+disp×2→PC	1011ddddddddddd	1	—

(1) 説明

アドレス（PC+4+ディスプレースメント×2）に分岐し、PR にアドレス（PC+4）を格納します。PC ソース値は BSR の命令アドレスです。12 ビットディスプレースメントは符号拡張後 2 倍しますので、分岐先との相対距離は−4096 バイトから+4094 バイトの範囲になります。分岐先に届かないときは、JSR 命令によってこの分岐が可能になります。

(2) 注意

遅延分岐命令ですので、本命令の直後の命令を先に実行してから、分岐先の命令を実行します。

本命令と直後の命令との間には、割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

(3) 動作内容

```
BSR(int d)   /* BSR disp */
{
     int disp;
     unsigned int temp;

     temp = PC;
     if((d & 0x800) == 0)
          disp =  (0x00000FFF & d);
     else  disp =  (0xFFFFF000 | d);
     PR = PC + 4;
     PC = PC + 4 + (disp << 1);
     Delay_Slot(temp + 2);
}
```

(4) 使用例

```
        BSR    TRGET          ;TRGET へ分岐します。
        MOV    R3,R4          ;分岐に先立ち MOV を実行します。
        ADD    R0,R1          ;サブルーチンプロシージャからの戻り先(PR の内容)です。
    .....
    .....
TRGET:                        ;←プロシージャの入り口
        MOV    R2,R3          ;
        RTS                   ;上記 ADD 命令に戻ります。
        MOV    #1,R0          ;分岐に先立ち MOV を実行します。
```

(5) 発生する可能性がある例外

- スロット不当命令例外

10.2.2 BSRF Branch to SubRoutine Far 分岐命令

サブルーチンプロシージャへの分岐 遅延分岐命令

書式	動作概略	命令コード	実行ステート	T ビット
BSRF Rn	PC+4→PR, PC+4+Rn→PC	0000nnnn00000011	1	—

(1) 説明

アドレス（PC+4+Rn）に分岐し、PR にアドレス（PC+4）を格納します。PC ソース値は BSRF の命令アドレスです。分岐先は PC+4 に汎用レジスタ Rn の内容の 32 ビットデータを加えたアドレスです。

(2) 注意

遅延分岐命令ですので、本命令の直後の命令を先に実行してから、分岐先の命令を実行します。

本命令と直後の命令との間には、割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

(3) 動作内容

```
BSRF(int n)     /* BSRF Rn */
{
     unsigned int temp;

     temp = PC;
     PR = PC + 4;
     PC = PC + 4 + R[n];
     Delay_Slot(temp + 2);
}
```

(4) 使用例

```
        MOV.L  #(TRGET-BSRF_PC),R0    ;ディスプレースメントを設定します。
        BRSF   R0                     ;TRGET へ分岐します。
        MOV    R3,R4                  ;分岐に先立ち MOV を実行します。
BSRF_PC:                              ;
        ADD    R0,R1                  ;
        .....
TRGET:                                ;←プロシージャの入り口
        MOV    R2,R3                  ;
        RTS                           ;上記 ADD 命令に戻ります。
        MOV    #1,R0                  ;分岐に先立ち MOV を実行します。
```

(5) 発生する可能性がある例外

- スロット不当命令例外

10.2.3 JSR Jump to SubRoutine 分岐命令

サブルーチンプロシージャへの分岐 遅延分岐命令

書式	動作概略	命令コード	実行ステート	T ビット
JSR @Rn	PC+4→PR, Rn→PC	0100nnnn00001011	1	―

(1) 説明

本命令の直後の命令の実行後指定したアドレスのサブルーチンプロシージャへ遅延分岐します。戻り先アドレス（PC+4）を PR に退避し、汎用レジスタ Rn で表されるアドレスへ分岐します。RTS と組み合わせて、サブルーチンプロシージャコールに使用します。

(2) 注意

遅延分岐命令ですので、本命令の直後の命令を先に実行してから、分岐先の命令を実行します。

本命令と直後の命令との間には、割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

(3) 動作内容

```
JSR(int n)  /* JSR @Rn */
{
    unsigned int temp;

    temp = PC;
    PR = PC + 4;
    PC = R[n];
    Delay_Slot(temp + 2);
}
```

(4) 使用例

```
            MOV.L    JSR_TABLE,R0    ;R0=TRGET のアドレス
            JSR      @R0             ;TRGET へ分岐します。
            XOR      R1,R1           ;分岐に先立ち XOR を実行します。
            ADD      R0,R1           ;←プロシージャからの戻り先
            .......                  (PR の内容)です。
            .align   4
JSR_TABLE:  .data.l  TRGET           ;ジャンプテーブル
```

```
TRGET:    NOP                 ;←プロシージャの入り口
          MOV     R2,R3       ;
          RTS                 ;上記 ADD 命令に戻ります。
          MOV     #70,R1      ;RTS に先立ち MOV を実行します。
```

(5) 発生する可能性がある例外

- スロット不当命令例外

10.2.4 LDC LoaD to Control register システム制御命令

コントロールレジスタへのロード （特権命令）

書式	動作概略	命令コード	実行ステート	Tビット
LDC Rm, SR	Rm→SR	0100mmmm00001110	4	LSB
LDC.L @Rm+, SR	(Rm)→SR, Rm+4→Rm	0100mmmm00000111	4	LSB

(1) 説明

ソースオペランドをコントロールレジスタ SR に格納します。

(2) 注意

特権モードでのみ使用することができます。もしユーザモードで使用された場合は、不当命令例外が発生します。

(3) 動作内容

```
LDCSR(int m)      /* LDC Rm,SR : Privileged */
{
     SR = R[m] & 0x700083F3;
     PC += 2;
}
LDCMSR(int m)     /* LDC.L @Rm+,SR: Privileged */
{
     SR = Read_Long(R[m]) & 0x700083F3;
     R[m] += 4;
     PC += 2;
}
```

(4) 発生する可能性がある例外

- データTLB多重ヒット例外
- 一般不当命令例外
- スロット不当命令例外
- データTLBミス例外
- データTLB保護違反例外
- データアドレスエラー

10.2.5 LDS LoaD to FPU System register システム制御命令

FPU システム
レジスタへのロード

書式	動作概略	命令コード	実行ステート	T ビット
LDS Rm,FPUL	Rm→FPUL	0100mmmm01011010	1	—
LDS.L @Rm+,FPUL	(Rm)→FPUL, Rm＋4→Rm	0100mmmm01010110	1	—
LDS Rm,FPSCR	Rm→FPSCR	0100mmmm01101010	1	—
LDS.L @Rm+,FPSCR	(Rm)→FPSCR, Rm＋4→Rm	0100mmmm01100110	1	—

(1) 説明

ソースオペランドを FPU システムレジスタ FPUL、FPSCR に格納します。

(2) 注意

特になし

(3) 動作内容

```
#define FPSCR_MASK 0x003FFFFF

LDSFPUL(int  m,int *FPUL)     /* LDS Rm,FPUL  */
{
      *FPUL = R[m];
      PC += 2;
}
LDSMFPUL(int  m,int *FPUL)    /* LDS.L @Rm+,FPUL  */
{
      *FPUL = Read_Long(R[m]);
      R[m] += 4;
      PC += 2;
}
LDSFPSCR(int  m)       /* LDS Rm,FPSCR  */
{
      FPSCR = R[m] & FPSCR_MASK;
      PC += 2;
}
LDSMFPSCR(int  m)      /* LDS.L @Rm+,FPSCR  */
{
```

```
        FPSCR = Read_Long(R[m]) & FPSCR_MASK;
        R[m] += 4;
        PC += 2;
}
```

(4) 発生する可能性がある例外

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- データアドレスエラー

10.2.6 STC STore Control register システム制御命令

コントロールレジスタからのストア （特権命令）

書式	動作概略	命令コード	実行ステート	T ビット
STC SR, Rn	SR→Rn	0000nnnn00000010	1	—
STC.L SR, @-Rn	Rn-4→Rn, SR→(Rn)	0100nnnn00000011	1	—

(1) 説明

コントロールレジスタ SR をデスティネーションに格納します。

(2) 注意

特権モードでのみ使用可能です。ユーザモードで使用すると、不当命令例外が発生します。

(3) 動作内容

```
STCSR(int n)      /* STC SR,Rn : Privileged */
{
      R[n] = SR;
      PC += 2;
}
STCMSR(int n)     /* STC.L SR,@-Rn : Privileged */
{
      R[n] -= 4;
      Write_Long(R[n],SR);
      PC += 2;
}
```

(4) 発生する可能性がある例外

- データTLB多重ヒット例外
- 一般不当命令例外
- スロット不当命令例外
- データTLBミス例外
- データTLB保護違反例外
- 初期ページ書き込み例外
- データアドレスエラー

10.2.7 STS Store from FPU System register システム制御命令

FPU システムレジスタからのストア

書式		動作概略	命令コード	実行ステート	T ビット
STS	FPUL,Rn	FPUL→Rn	0000nnnn01011010	1	—
STS	FPSCR,Rn	FPSCR→Rn	0000nnnn01101010	1	—
STS.L	FPUL,@-Rn	Rn-4→Rn, FPUL→(Rn)	0100nnnn01010010	1	—
STS.L	FPSCR,@-Rn	Rn-4→Rn, FPSCR→(Rn)	0100nnnn01100010	1	—

(1) 説明

FPU システムレジスタ FPUL、FPSCR をデスティネーションに格納します。

(2) 注意

特になし

(3) 動作内容

```
STSFPUL(int  n, int *FPUL)    /* STS FPUL,Rn */
{
     R[n] = *FPUL;
     PC += 2;
}
STSMFPUL(int  n, int *FPUL)   /* STS.L FPUL,@-Rn  */
{
     R[n] -= 4;
     Write_Long(R[n],*FPUL) ;
     PC += 2;
}
STSFPSCR(int  n)              /* STS FPSCR,Rn  */
{
     R[n] = FPSCR & 0x003FFFFF;
     PC += 2;
}
STSMFPSCR(int  n)             /* STS.L FPSCR,@-Rn  */
{
     R[n] -= 4;
     Write_Long(R[n],FPSCR & 0x003FFFFF)
```

```
    PC += 2;
}
```

(4) 使用例

```
STS
Example 1:
MOV.L       #H'12ABCDEF, R12
LDS         R12, FPUL
STS         FPUL, R13
                            ; After executing the STS instruction:
                            ; R13 = 12ABCDEF
Example 2:
STS         FPSCR, R2
                            ; After executing the STS instruction:
                            ; The current content of FPSCR is stored in register R2
```

STS.L

```
Example 1:
MOV.L       #H'0C700148, R7
STS.L       FPUL, @-R7
                            ; Before executing the STS.L instruction:
                            ; R7 = 0C700148
                            ; After executing the STS.L instruction:
                            ; R7 = 0C700144, and the content of FPUL is saved at memory
                            ; location 0C700144.
Example 2:
MOV.L       #H'0C700154, R8
STS.L       FPSCR, @-R8
                            ; After executing the STS.L instruction:
                            ; The content of FPSCR is saved at memory location 0C700150.
```

(5) 発生する可能性のある例外

- データTLB多重ヒット例外
- データTLBミス例外
- データTLB保護違反例外
- 初期ページ書き込み例外
- データアドレスエラー

10.3 FPU 命令

FPU 命令の C を用いた動作内容の説明では、CPU 命令の動作内容の説明で用いた資源や関数に加え、以下の資源と関数を使用します。

- 浮動小数点用の定義文です。

```
#define PZERO        0
#define NZERO        1
#define DENORM       2
#define NORM         3
#define PINF         4
#define NINF         5
#define qNaN         6
#define sNaN         7
#define EQ           0
#define GT           1
#define LT           2
#define UO           3
#define INVALID      4
#define FADD         0
#define FSUB         1

#define CAUSE         0x0003f000       /* FPSCR(bit17-12) */
#define SET_E         0x00020000       /* FPSCR(bit17) */
#define SET_V         0x00010040       /* FPSCR(bit16,6) */
#define SET_Z         0x00008020       /* FPSCR(bit15,5) */
#define SET_O         0x00004010       /* FPSCR(bit14,4) */
#define SET_U         0x00002008       /* FPSCR(bit13,3) */
#define SET_I         0x00001004       /* FPSCR(bit12,2) */
#define ENABLE_VOUI   0x00000b80       /* FPSCR(bit11,9-7) */
#define ENABLE_V      0x00000800       /* FPSCR(bit11) */
#define ENABLE_Z      0x00000400       /* FPSCR(bit10) */
#define ENABLE_OUI    0x00000380       /* FPSCR(bit9-7) */
#define ENABLE_I      0x00000080       /* FPSCR(bit7) */
#define FLAG          0x0000007C       /* FPSCR(bit6-2) */

#define FPSCR_FR      FPSCR>>21&1
#define FPSCR_PR      FPSCR>>19&1
#define FPSCR_DN      FPSCR>>18&1
#define FPSCR_I       FPSCR>>12&1
#define FPSCR_RM      FPSCR&1
#define FR_HEX        frf.l[ FPSCR_FR]
```

```
#define FR      frf.f[ FPSCR_FR]
#define DR_HEX  frf.l[ FPSCR_FR]
#define DR      frf.d[ FPSCR_FR]
#define XF_HEX  frf.l[~FPSCR_FR]
#define XF      frf.f[~FPSCR_FR]
#define XD      frf.d[~FPSCR_FR]

union {
     int    l[2][16];
     float   f[2][16];
     double d[2][8];
} frf;
int FPSCR;

int sign_of(int n)
{
     return(FR_HEX[n]>>31);
}
int data_type_of(int n) {
int abs;
     abs = FR_HEX[n] & 0x7fffffff;
     if(FPSCR_PR==0) {/* 単精度 */
          if(abs<0x00800000){
               if((FPSCR_DN == 1) || (abs == 0x00000000)){
                    if(sign_of(n) == 0) {zero(n, 0); return(PZERO);}
                    else                {zero(n, 1); return(NZERO);}
               }
               else return(DENORM);
          }
          else if(abs<0x7f800000)         return(NORM);
     else if(abs==0x7f800000) {
                    if(sign_of(n)==0)     return(PINF);
                    else                  return(NINF);
               }
               else if(abs<0x7fc00000)    return(qNaN);
                    else                  return(sNaN);
          }
     else {    /* 倍精度 */
          if(abs<0x00100000){
               if((FPSCR_DN==1) || ((abs==0x00000000)
                    && (FR_HEX[n+1]==0x00000000){
```

```
                if(sign_of(n)==0)   {zero(n,0); return(PZERO);}
                else                {zero(n,1); return(NZERO);}
            }
            else return(DENORM);
        }
}
        else if(abs < 0x7ff00000)  return(NORM);
        else if((abs == 0x7ff00000) &&
               (FR_HEX[n+1] == 0x00000000)) {
           if(sign_of(n) == 0)return(PINF);
           else                return(NINF);
        }
        else if(abs < 0x7ff80000) return(qNaN);
        else                      return(sNaN);
     }
}
void register_copy(int m,n)
{
                        FR[n] = FR[m];
     if(FPSCR_PR == 1)  FR[n+1] = FR[m+1];
}
void normal_faddsub(int m,n,type)
{
union {
     float f;
     int l;
}    dstf,srcf;
union {
     double d;
     int l[2];
}    dstd,srcd;
union {              /* “long double” のフォーマット:*/
     long double x; /*1-bit 符号*/
     int l[4];   /*15-bit 指数*/
}    dstx;       /*112-bit 小数*/
     if(FPSCR_PR == 0) {
        if(type == FADD) srcf.f =  FR[m];
        else           srcf.f = -FR[m];
        dstd.d = FR[n]; /* 単精度から倍精度への変換*/
        dstd.d += srcf.f;
```

```
            if(((dstd.d == FR[n]) && (srcf.f != 0.0)) ||
               ((dstd.d == srcf.f) && (FR[n] != 0.0))) {
               set_I();
               if(sign_of(m) ^ sign_of(n)) {
                   dstd.l[1] -= 1;
                    if(dstd.l[1] == 0xffffffff) dstd.l[0] -= 1;
               }
            }
            if(dstd.l[1] & 0x1fffffff) set_I();
            dstf.f += srcf.f; /* 近傍への丸め */
            if(FPSCR_RM == 1) {
               dstd.l[1] &= 0xe0000000; /* 0 への丸め */
               dstf.f = dstd.d;
            }
            check_single_exception(&FR[n],dstf.f);
         } else {
            if(type == FADD)    srcd.d =  DR[m>>1];
            else                srcd.d = -DR[m>>1];
            dstx.x = DR[n>>1]; /* 倍精度から拡張倍精度への変換 */
            dstx.x += srcd.d;
            if(((dstx.x == DR[n>>1]) && (srcd.d != 0.0)) ||
               ((dstx.x == srcd.d) && (DR[n>>1] != 0.0)) ) {
               set_I();
               if(sign_of(m)^ sign_of(n)) {
                  dstx.l[3] -= 1;
                  if(dstx.l[3] == 0xffffffff) {dstx.l[2] -= 1;
                  if(dstx.l[2] == 0xffffffff) {dstx.l[1] -= 1;
                  if(dstx.l[1] == 0xffffffff) {dstx.l[0] -= 1;}}}
               }
            }
            if((dstx.l[2] & 0x0fffffff) || dstx.l[3]) set_I();
            dst.d += srcd.d; /*近傍への丸め */
            if(FPSCR_RM == 1) {
               dstx.l[2] &= 0xf0000000; /* 0 への丸め */
               dstx.l[3]  = 0x00000000;
               dst.d = dstx.x;
            }
            check_double_exception(&DR[n>>1] ,dst.d);
         }
      }
```

```
void normal_fmul(int m,n)
{
union {
      float f;
      int l;
}     tmpf;
union {
      double d;
      int l[2];
}     tmpd;
union {
      long double x;
      int l[4];
}     tmpx;
      if(FPSCR_PR == 0) {
         tmpd.d = FR[n]; /* 単精度から倍精度 */
         tmpd.d *= FR[m]; /* 正確に作成 */
         tmpf.f *= FR[m]; /* 近傍への丸め */
         if(tmpf.f != tmpd.d) set_I();
         if((tmpf.f > tmpd.d) && (FPSCR_RM == 1)) {
            tmpf.l -= 1; /* 0 への丸め */
         }
         check_single_exception(&FR[n],tmpf.f);
      } else {
         tmpx.x = DR[n>>1]; /* 単精度から倍精度 */
         tmpx.x *= DR[m>>1]; /* 正確に作成 */
         tmpd.d *= DR[m>>1]; /* 近傍への丸め */
         if(tmpd.d != tmpx.x) set_I();
         if(tmpd.d > tmpx.x) && (FPSCR_RM == 1)) {
            tmpd.l[1] -= 1; /* 0 への丸め */
            if(tmpd.l[1] == 0xffffffff) tmpd.l[0] -= 1;
         }
         check_double_exception(&DR[n>>1], tmpd.d);
      }
}
void fipr(int m,n)
{
union {
      double d;
      int l[2];
```

```
}     mlt[4];

float dstf;
      if((data_type_of(m) == sNaN) || (data_type_of(n) == sNaN) ||
         (data_type_of(m+1) == sNaN) || (data_type_of(n+1) == sNaN) ||
         (data_type_of(m+2) == sNaN) || (data_type_of(n+2) == sNaN) ||
         (data_type_of(m+3) == sNaN) || (data_type_of(n+3) == sNaN) ||
         (check_product_invalid(m,n)) ||
         (check_product_invalid(m+1,n+1)) ||
         (check_product_invalid(m+2,n+2)) ||
         (check_product_invalid(m+3,n+3)) )  invalid(n+3);
      else if((data_type_of(m) == qNaN)|| (data_type_of(n) == qNaN)||
         (data_type_of(m+1) == qNaN) || (data_type_of(n+1) == qNaN) ||
         (data_type_of(m+2) == qNaN) || (data_type_of(n+2) == qNaN) ||
         (data_type_of(m+3) == qNaN) || (data_type_of(n+3) == qNaN)) qnan(n+3);
      else if(check_ positive_infinity() &&
              (check_ negative_infinity())          invalid(n+3);
      else if(check_ positive_infinity())inf(n+3,0);
      else if(check_ negative_infinity())inf(n+3,1);
      else {
         for(i=0;i<4;i++) {
            /* FPSCR_DN == 1 なら、0 にする) */
            if(data_type_of(m+i) == PZERO)        FR[m+i] = +0.0;
            else if(data_type_of(m+i) == NZERO)   FR[m+i] = -0.0;
            if(data_type_of(n+i) == PZERO)        FR[n+i] = +0.0;
            else if(data_type_of(n+i) == NZERO)   FR[n+i] = -0.0;
            mlt[i].d = FR[m+i];
            mlt[i].d *= FR[n+i];

      /* 正確には、FIPR では、下位 18bit を切り捨てているので、ここに記述したものは
         ハードウェアとは異なりより単純にしたものです。 */
         mlt[i].l[1] &= 0xff000000;
         mlt[i].l[1] |= 0x00800000;
         }
         mlt[0].d += mlt[1].d + mlt[2].d + mlt[3].d;
         mlt[0].l[1] &= 0xff800000;
         dstf = mlt[0].d;
         set_I();
         check_single_exception(&FR[n+3],dstf);
      }
```

```
}
void check_single_exception(float *dst,result)
{
union {
      float f;
      int l;
}     tmp;
float abs;
      if(result < 0.0)tmp.l = 0xff800000; /* -無限大 */
      else              tmp.l = 0x7f800000; /* +無限大 */
      if(result == tmp.f) {
         set_O(); set_I();
         if(FPSCR_RM == 1){
            tmp.l -= 1; /* 正規化数の最大値 */
            result = tmp.f;
         }
      }
      if(result < 0.0)abs = -result;
      else              abs =  result;
      tmp.l = 0x00800000; /* 正規化数の最小値 */
      if(abs < tmp.f) {
         if((FPSCR_DN == 1) && (abs != 0.0)) {
            set_I();
            if(result < 0.0)result = -0.0; /* 非正規化数を0にする。 */
            else              result =  0.0;
         }
         if(FPSCR_I == 1) set_U();
      }
      if(FPSCR & ENABLE_OUI) fpu_exception_trap();
      else                   *dst = result;
}
void check_double_exception(double *dst,result)
{
union {
      double d;
      int l[2];
}     tmp;
double abs;
      if(result < 0.0)tmp.l[0] = 0xfff00000; /* -無限大 */
      else              tmp.l[0] = 0x7ff00000; /* +無限大 */
```

```
                tmp.l[1] = 0x00000000;
      if(result == tmp.d)
         set_O(); set_I();
         if(FPSCR_RM == 1) {
            tmp.l[0] -= 1;
            tmp.l[1] = 0xffffffff;
            result = tmp.d; /* 正規化数の最大値 */
         }
      }
      if(result < 0.0)abs = -result;
      else        abs =  result;
      tmp.l[0] = 0x00100000; /* 正規化数の最小値 */
      tmp.l[1] = 0x00000000;
      if(abs < tmp.d) {
         if((FPSCR_DN == 1) && (abs != 0.0)) {
            set_I();
            if(result < 0.0)result = -0.0; /* 非正規化数を 0 にする。 */
               else           result =  0.0;
         }
         if(FPSCR_I == 1) set_U();
      }
      if(FPSCR & ENABLE_OUI)fpu_exception_trap();
      else                     *dst = result;
}
int check_product_invalid(int m,n)
{
      return(check_product_infinity(m,n)  &&
            ((data_type_of(m) == PZERO) || (data_type_of(n) == PZERO) ||
             (data_type_of(m) == NZERO) || (data_type_of(n) == NZERO)));
}
int check_ product_infinity(int m,n)
{
      return((data_type_of(m) == PINF) || (data_type_of(n) == PINF) ||
             (data_type_of(m) == NINF) || (data_type_of(n) == NINF));
}
int check_ positive_infinity(int m,n)
{
  return(((check_ product_infinity(m,n) && (~sign_of(m)^ sign_of(n))) ||
      ((check_ product_infinity(m+1,n+1) && (~sign_of(m+1)^ sign_of(n+1))) ||
      ((check_ product_infinity(m+2,n+2) && (~sign_of(m+2)^ sign_of(n+2))) ||
```

```
      ((check_ product_infinity(m+3,n+3) && (~sign_of(m+3)^ sign_of(n+3))));
}
int check_ negative_infinity(int m,n)
{
  return(((check_ product_infinity(m,n) && (sign_of(m)^ sign_of(n))) ||
      ((check_ product_infinity(m+1,n+1) && (sign_of(m+1)^ sign_of(n+1))) ||
      ((check_ product_infinity(m+2,n+2) && (sign_of(m+2)^ sign_of(n+2))) ||
      ((check_ product_infinity(m+3,n+3) && (sign_of(m+3)^ sign_of(n+3))));
}
void clear_cause () {FPSCR &= ~CAUSE;}
void set_E() {FPSCR |= SET_E; fpu_exception_trap();}
void set_V() {FPSCR |= SET_V;}
void set_Z() {FPSCR |= SET_Z;}
void set_O() {FPSCR |= SET_O;}
void set_U() {FPSCR |= SET_U;}
void set_I() {FPSCR |= SET_I;}
void invalid(int n)
{
      set_V();
      if((FPSCR & ENABLE_V) == 0 qnan(n);
      else    fpu_exception_trap();
}

void dz(int n,sign)
{
      set_Z();
      if((FPSCR & ENABLE_Z) == 0 inf(n,sign);
      else    fpu_exception_trap();
}
void zero(int n,sign)
{
      if(sign == 0)        FR_HEX [n]   = 0x00000000;
      else                 FR_HEX [n]   = 0x80000000;
      if(FPSCR_PR==1)      FR_HEX [n+1] = 0x00000000;
}
void inf(int n,sign) {
      if(FPSCR_PR==0) {
         if(sign == 0)     FR_HEX [n]   = 0x7f800000;
         else              FR_HEX [n]   = 0xff800000;
      } else {
         if(sign == 0)     FR_HEX [n]   = 0x7ff00000;
```

```
        else            FR_HEX [n]   = 0xfff00000;
                        FR_HEX [n+1] = 0x00000000;
    }
}
void qnan(int n)
{
    if(FPSCR_PR==0)     FR[n]   = 0x7fbfffff;
    else {              FR[n]   = 0x7ff7ffff;
                        FR[n+1] = 0xffffffff;
    }
}
```

10.3.1 FABS Floating - point ABSolute value 浮動小数点命令

浮動小数点絶対値

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FABS FRn	FRn & H'7FFFFFFF→FRn	1111nnnn01011101	1	—
1	FABS DRn	DRn & H'7FFFFFFFFFFFFFFF → DRn	1111nnn001011101	1	—

(1) 説明

浮動小数点レジスタ FRn/DRn の内容の最上位ビットを 0 にクリアして、結果を FRn/DRn に格納します。

FPSCR の cause/flag 部分は更新されません。

(2) 注意

特になし

(3) 動作内容

```
void FABS (int n){
    FR[n] = FR[n] & 0x7fffffff;
    pc += 2;
}
/* 精度に依存せず、同じ動作を行います。 */
```

(4) 発生する可能性がある例外

なし

10.3.2 FADD Floating - point ADD 浮動小数点命令

浮動小数点加算

PR	書式	動作概略	命令コード	実行ステート	Tビット
0	FADD FRm,FRn	FRn＋FRm→FRn	1111nnnnmmmm0000	1	―
1	FADD DRm,DRn	DRn＋DRm→DRn	1111nnn0mmm00000	1	―

(1) 説明

- FPSCR.PR=0の場合

  FRnとFRmの内容の2つの単精度浮動小数点数を算術加算し、結果をFRnに格納します。

- FPSCR.PR=1の場合

  DRnとDRmの内容の2つの倍精度浮動小数点数を算術加算し、結果をDRnに格納します。

  FPSCR.enable.Iがセットされている場合、FPU例外トラップが、例外の発生いかんに関わらず発生します。FPSCR.enable.O/Uがセットされている場合は、実際にFPU例外要因が発生したとき、および命令固有の特殊条件が成立したときにFPU例外トラップが発生します。なお当該特殊条件を本節の末尾に示します。例外発生時は、FPSCR.cause FPSCR.flagには、例外の正しい情報が反映され、FRn/DRnは更新されません。ソフトウェアで適切な処理を行ってください。

(2) 注意

特になし

(3) 動作内容

```
void FADD (int m,n)
{
     pc += 2;
     clear_cause();
     if((data_type_of(m)==sNaN) || (data_type_of(n)==sNaN)) invalid(n);
     else  if((data_type_of(m)==qNaN) || (data_type_of(n)==qNaN)) qnan(n);
           else if((data_type_of(m)==DENORM)
                   || (data_type_of(n)==DENORM)) set_E();
                else switch (data_type_of(m)){
                       case NORM:  switch (data_type_of(n)){
                                     case NORM:  normal_faddsub(m,n,ADD);
                                                 break;
                                     case PZERO:
                                     case NZERO: register_copy(m,n);
```

```
                                        break;
                                  default:    break;
                                }
                                break;
                        case PZERO: switch (data_type_of(n)){
                                     case NZERO:  zero(n,0);  break;
                                     default:     break;
                                    }
                                    break;
                        case NZERO: break;
                        case PINF:  switch (data_type_of(n)){
                                      case NINF:  invalid(n); break;
                                      default:    inf(n,0);   break;
                                    }
                                   break;
                        case NINF: switch (data_type_of(n)){
                                     case PINF:    invalid(n); break;
                                     default:      inf(n,1);   break;
                                    }
                                    break;
                    }
    }
```

FADD 特殊ケース

FADD	FRn,DRm							
FRm,DRm	+NORM	-NORM	+0	–0	+inf	–inf	qNaN	sNaN
+NORM	FADD							
-NORM								
+0			+0					
–0				–0		–inf		
+inf					+inf	invalid		
–inf				–inf	invalid	–inf		
qNaN							qNaN	
sNaN								invalid

【注】 FPSCR.DN=1 のとき、DENORM を 0 として扱います。

FPSCR.DN=0 のとき、DENORM は NORM と同様に計算されます。

(4) 発生する可能性がある例外とオーバフロー／アンダフロー例外トラップの発生条件

- FPUエラー
- 無効演算
- オーバフロー

  オーバフロー**例外トラップの発生条件**

  ・FPSCR.PR=0の場合：FRnとFRmが同符合かつ少なくとも一方の指数部がH'FE

  ・FPSCR.PR=1の場合：DRnとDRmが同符合かつ少なくとも一方の指数部がH'7FE
- アンダフロー

  アンダフロー例外トラップの発生条件

  ・FPSCR.PR=0の場合：FRnとFRmが異符合かつ双方の指数部がH'18以下

  ・FPSCR.PR=1の場合：DRnとDRmが異符合かつ双方の指数部がH'035以下
- 不正確例外

10.3.3 FCMP Floating - point CoMPare 浮動小数点命令

浮動小数点比較

No.	PR	書式	動作概略	命令コード	実行ステート	Tビット
1	0	FCMP/EQ FRm,FRn	FRn=FRm のとき 1→T それ以外のとき 0→T	1111nnnnmmmm0100	1	1/0
2	1	FCMP/EQ DRm,DRn	DRn=DRm のとき 1→T それ以外のとき 0→T	1111nnn0mmm00100	1	1/0
3	0	FCMP/GT FRm,FRn	FRn>FRm のとき 1→T それ以外のとき 0→T	1111nnnnmmmm0101	1	1/0
4	1	FCMP/GT DRm,DRn	DRn>DRm のとき 1→T それ以外のとき 0→T	1111nnn0mmm00101	1	1/0

(1) 説明

1. FPSCR.PR=0の場合：FRnとFRmの内容の2つの単精度浮動小数点数を算術比較し、等しい場合にTビットに1を、他の場合に0を格納します。
2. FPSCR.PR=1の場合：DRnとDRmの内容の2つの倍精度浮動小数点数を算術比較し、等しい場合にTビットに1を、他の場合に0を格納します。
3. FPSCR.PR=0の場合：FRnとFRmの内容の2つの単精度浮動小数点数を算術比較し、 FRn>FRmの場合にTビットに1を、他の場合に0を格納します。
4. FPSCR.PR=1の場合：DRnとDRmの内容の2つの倍精度浮動小数点数を算術比較し、 DRn>DRmの場合にTビットに1を、他の場合に0を格納します。

(2) 注意

特になし

(3) 動作内容

```
void FCMP_EQ(int m,n) /* FCMP/EQ  FRm,FRn */
{
    pc += 2;
    clear_cause();
    if(fcmp_chk(m,n) == INVALID) fcmp_invalid();
    else if(fcmp_chk(m,n) == EQ)   T = 1;
    else                           T = 0;
```

```
}
void FCMP_GT(int m,n) /* FCMP/GT  FRm,FRn */
{
    pc += 2;
    clear_cause();
    if((fcmp_chk(m,n)==INVALID) || (fcmp_chk(m,n)==UO)) fcmp_invalid();
    else if(fcmp_chk(m,n)==GT)  T = 1;
         else                   T = 0;
}
int fcmp_chk (int m,n)
{
    if((data_type_of(m) == sNaN) ||
       (data_type_of(n) == sNaN))       return(INVALID);
    else if((data_type_of(m) == qNaN) ||
             (data_type_of(n) == qNaN))     return(UO);
    else switch(data_type_of(m)){
          case NORM: switch(data_type_of(n)){
                       case PINF:  return(GT); break;
                       case NINF:  return(LT); break;
                       default:    break;
                     }     break;
          case PZERO:
          case NZERO: switch(data_type_of(n)){
                       case PZERO:
                       case NZERO: return(EQ); break;
                       default:    break;
                     }     break;
         case PINF :   switch(data_type_of(n)){
                       case PINF: return(EQ);  break;
                       default:   return(LT);  break;
                     }     break;
         case NINF:    switch(data_type_of(n)){
                       case NINF:  return(EQ);  break;
                       default:    return(GT);  break;
                     }     break;
        }
    if(FPSCR_PR==0) {
            if(FR[n]==FR[m]) return(EQ);
```

```
            else if(FR[n] > FR[m]) return(GT);
                 else return(LT);
    }
    else {
            if(DR[n>>1] == DR[m>>1]) return(EQ);
            else if(DR[n>>1] > DR[m>>1]) return(GT);
                 else return(LT);
    }
}
void fcmp_invalid()
{
        set_V();
        if((FPSCR&ENABLE_V) == 0)  T = 0;
        else fpu_exception_trap();
}
```

FCMP 特殊ケース

FCMP/EQ	FRn,DRn							
FRm,DRm	NORM	DNORM	+0	–0	+INF	–INF	qNaN	sNaN
NORM	CMP							
DNORM								
+0			EQ					
–0								
+INF					EQ			
–INF						EQ		
qNaN							!EQ	
sNaN								Invalid

【注】 DN=1 の場合、非正規化数の値は 0 として扱う。

FCMP/GT	FRn,DRn							
FRm,DRm	NORM	DENORM	+0	–0	+INF	–INF	qNaN	sNaN
NORM	CMP				GT	!GT		
DENORM								
+0			!GT					
–0								
+INF	!GT				!GT			
–INF	GT					!GT		
qNaN							UO	
sNaN								Invalid

【注】 DN=1 の場合、非正規化数の値は 0 として扱う。

UO はアンオーダードです。アンオーダードは!GT と扱います。

(4) 発生する可能性がある例外

- 無効演算

10.3.4 FCNVDS Floating - point CoNVert Double to Single precision 浮動小数点命令

倍精度単精度変換

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	—	—	—	—	—
1	FCNVDS DRm,FPUL	(float)DRm →FPUL	1111mmm010111101	1	—

(1) 説明

FPSCR.PR=1 の場合：DRm 内の倍精度浮動小数点数を単精度浮動小数点数に変換し FPUL に格納します。

FPSCR.enable.I がセットされている場合、FPU 例外トラップが、例外の発生如何に関わらず発生します。FPSCR.enable.O/U がセットされている場合で、実際に FPU 例外要因が発生したとき、および命令固有の特殊条件が成立したときに FPU 例外トラップが発生します。なお、当該特殊条件を本節の末尾に示します。

例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FPUL は更新されません。ソフトウェアで適切な処理を行ってください。

(2) 注意

特になし

(3) 動作内容

```
void FCNVDS(int m, float *FPUL){
    case((FPSCR.PR){
        0: undefined_operation();   /* reserved */
        1: fcnvds(m, *FPUL); break; /* FCNVDS */
    }
}
void fcnvds(int m, float *FPUL)
{
    pc += 2;
    clear_cause();
    case(data_type_of(m)){
        NORM  :
        PZERO :
        NZERO :    normal_ fcnvds(m, *FPUL); break;
        DENORM :   set_E();
        PINF  :    *FPUL = 0x7f800000; break;
        NINF  :    *FPUL = 0xff800000; break;
```

```
        qNaN  :     *FPUL = 0x7fbfffff; break;
        sNaN  :       set_V();
                        if((FPSCR & ENABLE_V) == 0) *FPUL = 0x7fbfffff;
                        else fpu_exception_trap();   break;
    }
}
void normal_fcnvds(int m,float *FPUL)
{
int sign;
float abs;
union {
      float f;
      int l;
}     dstf,tmpf;
union {
      double d;
      int l[2];
}     dstd;
      dstd.d = DR[m>>1];
      if(dstd.l[1] & 0x1fffffff)) set_I();
      if(FPSCR_RM == 1) dstd.l[1] &= 0xe0000000; /* round toward zero */
      dstf.f = dstd.d;
      check_single_exception(FPUL,dstf.f);
}
```

FCNVDS 特殊ケース

DRn	+NORM	–NORM	+0	–0	+INF	–INF	qNaN	sNaN
FCNVDS(DRn FPUL)	FCNVDS	FCNVDS	+0	–0	+INF	–INF	qNaN	Invalid

【注】 DN=1 の場合、非正規化数の値は 0 として扱われます。

(4) 発生する可能性がある例外とオーバフロー／アンダフロー例外トラップの発生条件

- FPUエラー
- 無効演算
- オーバフロー

  オーバフロー例外トラップの発生条件

  ・DRnの指数部がH'47E以上
- アンダフロー

  アンダフロー例外トラップの発生条件

  ・DRnの指数部がH'380以下
- 不正確例外

10.3.5 FCNVSD Floating - point CoNVert Single to Double precision 浮動小数点命令

単精度倍精度変換

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	—	—	—	—	—
1	FCNVSD FPUL, DRn	(double)FPUL→DRn	1111nnn010101101	1	—

(1) 説明

FPSCR.PR=1 の場合：FPUL の内容を単精度浮動小数点数と解釈し、倍精度浮動小数点数に変換し、結果を DRn に格納します。

(2) 注意

特になし

(3) 動作内容

```
void FCNVSD(int n, float *FPUL){
    pc += 2;
    clear_cause();
    case((FPSCR_PR){
        0: undefined_operation();   /* reserved */
        1: fcnvsd(n,*FPUL); break;  /* FCNVSD */
    }
}
void fcnvsd(int n, float *FPUL)
{
    case(fpul_type(*FPUL)){
        PZERO :
        NZERO :
        PINF  :
        NINF  :   DR[n>>1] = *FPUL; break;
        DENORM :  set_E();      break;
        qNaN  :        qnan(n);      break;
        sNaN  :        invalid(n);      break;
    }
}
int fpul_type(int *FPUL)
```

```
{
int abs;
      abs = *FPUL & 0x7fffffff;
      if(abs < 0x00800000){
         if((FPSCR_DN==1) || (abs==0x00000000)){
            if(sign_of(src) == 0)   return(PZERO);
            else                    return(NZERO);
         }
         else                       return(DENORM);
      }
      else if(abs<0x7f800000)       return(NORM);
      else if(abs==0x7f800000) {
            if(sign_of(src) == 0)   return(PINF);
            else                    return(NINF);
      }
      else if(abs<0x7fc00000)       return(qNaN);
      else                          return(sNaN);
}
```

FCNVSD 特殊ケース

FRn	+NORM	–NORM	+0	–0	+INF	–INF	qNaN	sNaN
FCNVSD(FPUL FRn)	+NORM	–NORM	+0	–0	+INF	–INF	qNaN	Invalid

【注】 DN=1 の場合、非正規化数の値は 0 として扱われます。

(4) 発生する可能性がある例外

- FPUエラー
- 無効演算

10.3.6 FDIV Floating - point DIVide 浮動小数点命令

浮動小数点除算

PR	書式	動作概略	命令コード	実行ステート	Tビット
0	FDIV FRm,FRn	FRn/FRm→FRn	1111nnnnmmmm0011	14	—
1	FDIV DRm,DRn	DRn/DRm→DRn	1111nnn0mmm00011	30	—

(1) 説明

- FPSCR.PR=0の場合

  FRnとFRmの内容の2つの単精度浮動小数点数を算術除算し、結果をFRnに格納します。

- FPSCR.PR=1の場合

  DRnとDRmの内容の2つの倍精度浮動小数点数を算術除算し、結果をDRnに格納します。

FPSCR.enable.Iがセットされている場合、FPU例外トラップが、例外の発生如何に関わらず発生します。

FPSCR.enable.O/Uがセットされている場合で、実際にFPU例外要因が発生したとき、および命令固有の特殊条件が成立したときにFPU例外トラップが発生します。なお、当該特殊条件を本節の末尾に示します。

例外発生時は、FPSCR.cause FPSCR.flagには、例外の正しい情報が反映され、FRn/DRnは更新されません。ソフトウェアで適切な処理を行ってください。

(2) 注意

特になし

(3) 動作内容

```
void FDIV(int m,n)     /* FDIV FRm,FRn */
{
    pc += 2;
    clear_cause();
    if((data_type_of(m) == sNaN) || (data_type_of(n) == sNaN)) invalid(n);
    else if((data_type_of(m) == qNaN) || (data_type_of(n) == qNaN)) qnan(n);
         else switch (data_type_of(m)){
                case NORM: switch (data_type_of(n)){
                                case PINF:
                                case NINF: inf(n,sign_of(m)^sign_of(n));
                                           break;
                                case PZERO:
                                case NZERO:  zero(n,sign_of(m)^sign_of(n));
```

```
                                                                    break;
                        case DENORM: set_E();
                                     break;
                        default:     normal_fdiv(m,n);
                                     break;
                    } break;
            case PZERO: switch (data_type_of(n)){
                        case PZERO:
                        case NZERO:  invalid(n);
                                     break;
                        case PINF:
                        case NINF:   break;
                        default:     dz(n,sign_of(m)^sign_of(n));
                                     break;
                    } break;
            case NZERO: switch (data_type_of(n)){
                        case PZERO:
                        case NZERO:  invalid(n);break;
                        case PINF:   inf(n,1);break;
                        case NINF:   inf(n,0);break;
            default:                 dz(FR[n],sign_of(m)^sign_of(n));
                                     break;
                    } break;
            case DENORM:  set_E();
                        break;
            case PINF :
            case NINF : switch (data_type_of(n)){
                        case DENORM: set_E();
                                     break;
                        case PINF:
                        case NINF:   invalid(n);
                                     break;
                        default:     zero(n,sign_of(m)^sign_of(n))
                                     break
                    } break;
        }
    }
void normal_fdiv(int m,n)
```

```
{
union {
      float f;
      int l;
}     dstf,tmpf;
union {
      double d;
      int l[2];
}     dstd,tmpd;
union {
      int double x;
      int l[4];
}     tmpx;
      if(FPSCR_PR == 0) {
           tmpf.f = FR[n];        /* save destination value */
           dstf.f /= FR[m];       /* round toward nearest or even */
           tmpd.d = dstf.f;       /* convert single to double */
           tmpd.d *= FR[m];
           if(tmpf.f != tmpd.d) set_I();
           if((tmpf.f < tmpd.d) && (FPSCR_RM == 1)) dstf.l -= 1;
                                  /* round toward zero */
           check_single_exception(&FR[n], dstf.f);
      }
      else {
           tmpd.d = DR[n>>1];     /* save destination value */
           dstd.d /= DR[m>>1];    /* round toward nearest or even */
           tmpx.x = dstd.d;       /* convert double to int double */
           tmpx.x *= DR[m>>1];
           if(tmpd.d != tmpx.x) set_I();
           if((tmpd.d < tmpx.x) && (FPSCR_RM == 1)) {
               dstd.l[1] -= 1;    /* round toward zero */
               if(dstd.l[1] == 0xffffffff) dstd.l[0] -= 1;
           }
           check_double_exception(&DR[n>>1], dstd.d);
      }
}
```

FDIV 特殊ケース

FDIV	FRn,DRn									
FRm,DRm	+NORM	-NORM	+DENORM	–DENORM	+0	-0	+inf	–inf	qNaN	sNaN
+NORM	FDIV				+0	-0	+inf	-inf		
-NORM					-0	+0	-inf	+inf		
+DENORM					+0	-0	+inf	-inf		
–DENORM				Error	-0	+0	-inf	+inf		
+0							+inf	-inf		
-0				DZ		invalid	-inf	DZ+inf		
+inf	+0	-0	+0	-0	+0	-0				
–inf	-0	+0	-0	+0	-0	+0		invalid		
qNaN									qNaN	
sNaN										invalid

【注】 FPSCR.DN=1 のとき、DENORM を 0 として扱います。

(4) 発生する可能性がある例外とオーバフロー／アンダフロー例外トラップの発生条件

- FPUエラー
- 無効演算
- 0による除算
- オーバフロー

  オーバフロー例外トラップの発生条件

  ・FPSCR.PR=0の場合：FRnの指数部－FRmの指数部＋H'7FがH'FF以上

  ・FPSCR.PR=1の場合：DRnの指数部－DRmの指数部＋H'3FFがH'7FF以上
- アンダフロー

  アンダフロー例外トラップの発生条件

  ・FPSCR.PR=0の場合：FRnの指数部－FRmの指数部＋H'7FがH'01以下

  ・FPSCR.PR=1の場合：DRnの指数部－DRmの指数部＋H'3FFがH'001以下
- 不正確例外

10.3.7 FIPR Floating - point Inner PRoduct 浮動小数点命令

浮動小数点内積

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FIPR FVm,FVn	inner_product(FVm, FVn)→FR[n+3]	1111nnmm11101101	1	—
1	—	—	—	—	—

【注】 FV0 = {FR0, FR1, FR2, FR3}

FV4 = {FR4, FR5, FR6, FR7}

FV8 = {FR8, FR9, FR10, FR11}

FV12 = {FR12, FR13, FR14, FR15}

(1) 説明

- FPSCR.PR=0の場合

FVnとFVmで示される4次元の単精度浮動小数点数ベクタを算術内積し、結果をFR[n+3]に格納します。

FIPR命令は正確さよりも高速化のための命令です。そのため、FADDやFMULを組み合わせて使用した場合と、結果が異なります。FIPRは次の順序で実行します。

1. 各項をそれぞれ乗算します。結果は28ビットです。
2. それらの結果をアライメントします。このとき、30ビット以内に入るように丸めます。
3. アライメントした結果を加算します。
4. 正規化と丸めを行います。

以下の場合、特別な処理になります。

5. 入力値にsNaNがある場合、無効例外になります。
6. 乗算する入力値で0と無限大の組み合わせがある場合、無効例外になります。
7. 上記以外で、入力値にqNaNが含まれる場合、結果はqNaNになります。
8. 上記以外に入力値に無限大がある場合、

- もしも乗算した結果が2つ以上無限大になり、符号が異なる場合、無効例外になります。
- 上記以外の場合、正しい無限大が格納されます。

9. 入力値にsNaN、qNaN、無限大が含まれない場合は、通常と同じ処理を行います。

FPSCR.enable.U/I がセットされている場合、FPU 例外トラップが、例外の発生如何に関わらず発生します。FPSCR.enable.O がセットされている場合、実際に FPU 例外要因が発生したとき、および命令固有の特殊条件が成立したときに FPU 例外トラップが発生します。なお、当該特殊条件を本節の末尾に示します。

例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FR[n+3]は更新されません。

ソフトウェアで適切な処理を行ってください。

(2) 注意

特になし

(3) 動作内容

```
void FIPR(int m,n)   /* FIPR FVm,FVn */
{
      if(FPSCR_PR == 0)    {
         pc += 2;
         clear_cause();
         fipr(m,n);
      }
      else  undefined_operation();
}
```

(4) 発生する可能性がある例外とオーバフロー例外トラップの発生条件

- 無効演算
- オーバフロー

  オーバフロー例外トラップの発生条件

  次の数の少なくとも1つがH'FC以上

  ・FRnの指数部＋FRmの指数部

  ・FR(n+1)の指数部＋FR(m+1)の指数部

  ・FR(n+2)の指数部＋FR(m+2)の指数部

  ・FR(n+3)の指数部＋FR(m+3)の指数部
- アンダフロー
- 不正確例外

10.3.8 FLDI0 Floating - point LoaD Immediate 0.0 浮動小数点命令

0.0 ロード

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FLDI0 FRn	0x00000000→FRn	1111nnnn10001101	1	—
1	—	—	—	—	—

(1) 説明

FPSCR.PR=0 の場合、浮動小数点数の 0.0（0x00000000)を FRn に格納します。

(2) 注意

特になし

(3) 動作内容

```
void FLDI0(int n)
{
    FR[n] = 0x00000000;
    pc += 2;
}
```

(4) 発生する可能性がある例外

なし

10.3.9 FLDI1 Floating - point LoaD Immediate 1.0 浮動小数点命令

1.0 ロード

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FLDI1 FRn	0x3F800000→FRn	1111nnnn10011101	1	—
1	—	—	—	—	—

(1) 説明

FPSCR.PR=0 の場合、浮動小数点数の 1.0（0x3F800000)を FRn に格納します。

(2) 注意

特になし

(3) 動作内容

```
void FLDI1(int n)
{
    FR[n] = 0x3F800000;
    pc += 2;
}
```

(4) 発生する可能性がある例外

なし

10.3.10 FLDS Floating - point LoaD to System register 浮動小数点命令

システムレジスタへの転送

書式	動作概略	命令コード	実行ステート	T ビット
FLDS FRm,FPUL	FRm → FPUL	1111mmmm00011101	1	—

(1) 説明

浮動小数点レジスタ FRm の内容をシステムレジスタである FPUL に格納します。

(2) 注意

特になし

(3) 動作内容

```
void FLDS(int m,float *FPUL)
{
    *FPUL = FR[m];
    pc += 2;
}
```

(4) 発生する可能性がある例外

なし

10.3.11 FLOAT Floating - point convert from integer 浮動小数点命令

整数浮動小数点数変換

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FLOAT FPUL,FRn	(float)FPUL→FRn	1111nnnn00101101	1	—
1	FLOAT FPUL,DRn	(double)FPUL→DRn	1111nnn000101101	1	—

(1) 説明

FPSCR.PR=0 の場合：FPUL の内容を 32bit 整数とみなして、単精度浮動小数点数に変換し、 結果を FRn に格納します。

FPSCR.PR=1 の場合： FPUL の内容を 32bit 整数とみなして、倍精度浮動小数点数に変換し、 結果を DRn に格納します。

FPSCR.enable.I=1 で、FPSCR.PR=0 の場合、FPU 例外トラップが、例外の発生如何に関わらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn は更新されません。ソフトウェアで適切な処理を行ってください。

(2) 注意

特になし

(3) 動作内容

```
void FLOAT(int n,float *FPUL)
{
union {
      double d;
      int l[2];
}     tmp;
      pc += 2;
      clear_cause();
      if(FPSCR.PR==0){
         FR[n] = *FPUL; /* convert from integer to float */
         tmp.d = *FPUL;
         if(tmp.l[1] & 0x1fffffff) inexact();
      } else {
         DR[n>>1] = *FPUL; /* convert from integer to double */
      }
```

```
}
```

(4) 発生する可能性がある例外

- 不正確例外：FPSCR.PR = 1の場合、発生しません。

10.3.12 FMAC Floating - point Multiply and Accumulate 浮動小数点命令

浮動小数点積和

PR	書式	動作概略	命令コード	実行ステート	Tビット
0	FMAC FR0,FRm,FRn	FR0×FRm＋FRn→FRn	1111nnnnmmmm1110	1	—
1	—	—	—	—	—

(1) 説明

FPSCR.PR=0 の場合：FR0 と FRm の内容の 2 つの単精度浮動小数点数を算術乗算し、さらに、FRn の内容を算術加算し、結果を FRn に格納します。

FPSCR.enable.I がセットされている場合、FPU 例外トラップが、例外の発生いかんに関わらず発生します。FPSCR.enable.O/U がセットされている場合で、実際に FPU 例外要因が発生したとき、および命令固有の特殊条件が成立したときに FPU 例外トラップが発生します。なお、当該特殊条件を本節の末尾に示します。

例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn は更新されません。ソフトウェアで適切な処理を行ってください。

(2) 注意

特になし

(3) 動作内容

```
void FMAC(int m,n)
{
      pc += 2;
      clear_cause();
      if(FPSCR_PR == 1) undefined_operation();
      else if((data_type_of(0) == sNaN) ||
              (data_type_of(m) == sNaN) ||
              (data_type_of(n) == sNaN)) invalid(n);
      else if((data_type_of(0) == qNaN) ||
              (data_type_of(m) == qNaN)) qnan(n);
      else if((data_type_of(0) == DENORM) ||
              (data_type_of(m) == DENORM)) set_E();
      else switch (data_type_of(0){
              case NORM: switch (data_type_of(m)){
              case PZERO:
              case NZERO: switch (data_type_of(n)){
```

```
            case DENORM:  set_E();  break;
            case qNaN:    qnan(n);  break;
            case PZERO:
            case NZERO:  zero(n,sign_of(0)^ sign_of(m)^sign_of(n));  break;
            default:       break;
            }
        case PINF:
case NINF: switch (data_type_of(n)){
     case DENORM: set_E();  break;
     case qNaN:   qnan(n);  break;
     case PINF:
     case NINF:   if(sign_of(0)^ sign_of(m)^sign_of(n))  invalid(n);
                  else  inf(n,sign_of(0)^ sign_of(m));   break;
     default:           inf(n,sign_of(0)^ sign_of(m));   break;
     }
case NORM: switch (data_type_of(n)){
               case DENORM:  set_E();            break;
               case qNaN:    qnan(n);            break;
               case PINF:
               case NINF:    inf(n,sign_of(n));  break;
               case PZERO:
               case NZERO:
               case NORM:    normal_fmac(m,n);   break;
           }   break;
case PZERO:
case NZERO: switch (data_type_of(m)){
               case PINF:
               case NINF:  invalid(n); break;
               case PZERO:
               case NZERO:
               case NORM: switch (data_type_of(n)){
               case DENORM: set_E();    break;
               case qNaN:   qnan(n);    break;
               case PZERO:
               case NZERO: zero(n,sign_of(0)^ sign_of(m)^sign_of(n));  break;
               default:    break;
               }     break;
}       break;
```

```
        case PINF :
        case NINF : switch (data_type_of(m)){
             case PZERO:
             case NZERO:    invalid(n);  break;
             default: switch (data_type_of(n)){
             case DENORM:   set_E();  break;
             case qNaN:     qnan(n);  break;
             default:       inf(n,sign_of(0)^sign_of(m)^sign_of(n)); break
             }     break;
          }     break;
       }
}
void normal_fmac(int m,n)
{
union {
        int double x;
        int l[4];
}     dstx,tmpx;
float dstf,srcf;
        if((data_type_of(n) == PZERO)|| (data_type_of(n) == NZERO))
                srcf = 0.0; /* flush denormalized value */
        else    srcf = FR[n];
        tmpx.x = FR[0]; /* convert single to int double */
        tmpx.x *= FR[m]; /* exact product */
        dstx.x = tmpx.x + srcf;
        if(((dstx.x == srcf) && (tmpx.x != 0.0)) ||
            ((dstx.x == tmpx.x) && (srcf != 0.0))) {
            set_I();
            if(sign_of(0)^ sign_of(m)^ sign_of(n))  {
                dstx.l[3] -= 1; /* correct result */
                if(dstx.l[3] == 0xffffffff) dstx.l[2] -= 1;
                if(dstx.l[2] == 0xffffffff) dstx.l[1] -= 1;
                if(dstx.l[1] == 0xffffffff) dstx.l[0] -= 1;
            }
            else    dstx.l[3] |= 1;
        }
        if((dstx.l[1] & 0x01ffffff) || dstx.l[2] || dstx.l[3]) set_I();
        if(FPSCR_RM == 1) {
```

```
            dstx.l[1] &= 0xfe000000; /* round toward zero */
            dstx.l[2]  = 0x00000000;
            dstx.l[3]  = 0x00000000;
        }
        dstf = dstx.x;
        check_single_exception(&FR[n],dstf);
}
```

FMAC 特殊ケース

FMAC		FRm							
FRn	FR0	+NORM	-NORM	+0	–0	+inf	–inf	qNaN	sNaN
NORM	+NORM	FMAC				+inf	-inf		
	-NORM					-inf	+inf		
	+0					invalid			
	-0								
	+inf	+inf	-inf	invalid		+inf	-inf		
	-inf	-inf	+inf			-inf	+inf		
+0	+NORM	FMAC		invalid		+inf	-inf		
	-NORM					-inf	+inf		
	+0	+0				invalid			
	-0								
	+inf	+inf	-inf	invalid		+inf	-inf		
	-inf	-inf	+inf			-inf	+inf		
-0	+NORM	FMAC		+0	-0	+inf	-inf		
	-NORM			-0	+0	-inf	+inf		
	+0	+0	-0	+0	-0	invalid			
	-0	-0	+0	-0	+0				
	+inf	+inf	-inf	invalid		+inf	-inf		
	-inf	-inf	+inf			-inf	+inf		
+inf	+NORM					+inf	invalid		
	-NORM					invalid	+inf		
	+0					invalid			
	-0				+inf				
	+inf	+inf	invalid	invalid		+inf	invalid		
	-inf	invalid	+inf			invalid	+inf	qNaN	invalid

FMAC		FRm							
FRn	FR0	+NORM	-NORM	+0	–0	+inf	–inf	qNaN	sNaN
-inf	+NORM					invalid	-inf		
	-NORM					-inf	invalid		
	+0								
	-0				-inf		invalid		
	+inf	invalid	-inf			invalid	-inf		
	-inf	-inf	invalid		invalid	-inf	invalid		
qNaN	+NORM								
	-NORM								
	+0								
	-0						invalid		
	+inf								
	-inf				invalid				
!sNaN	qNaN							qNaN	
all types	sNaN								
sNaN	all types								invalid

【注】 FPSCR.DN=1 のとき、DENORM を 0 として扱います。

FPSCR.DN=0 のとき、DENORM は NORM と同様に計算されます。

(4) 発生する可能性がある例外とオーバフロー／アンダフロー例外トラップの発生条件

- FPUエラー
- 無効演算
- オーバフロー

  オーバフロー例外トラップの発生条件

  次の数の少なくとも一方がH'FD以上

  ・FR0の指数部＋FRmの指数部

  ・FRnの指数部
- アンダフロー

  アンダフロー例外トラップの発生条件

  次の数の少なくとも一方がH'2E以下

  ・FR0の指数部＋FRmの指数部

  ・FRnの指数部
- 不正確例外

10.3.13 FMOV Floating - point MOVe 浮動小数点命令

浮動小数点転送

No.	SZ	書式	動作概略	命令コード	実行ステート	T ビット
1	0	FMOV FRm,FRn	FRm→FRn	1111nnnnmmmm1100	1	—
2	1	FMOV DRm,DRn	DRm→DRn	1111nnn0mmm01100	1	—
3	0	FMOV.S FRm,@Rn	FRm→(Rn)	1111nnnnmmmm1010	1	—
4	1	FMOV DRm,@Rn	DRm→(Rn)	1111nnnnmmm01010	1	—
5	0	FMOV.S @Rm,FRn	(Rm)→FRn	1111nnnnmmmm1000	1	—
6	1	FMOV @Rm,DRn	(Rm)→DRn	1111nnn0mmmm1000	1	—
7	0	FMOV.S @Rm+,FRn	(Rm)→FRn,Rm+4→Rm	1111nnnnmmmm1001	1	—
8	1	FMOV @Rm+,DRn	(Rm)→DRn,Rm+8→Rm	1111nnn0mmmm1001	1	—
9	0	FMOV.S FRm,@-Rn	Rn-4→Rn,FRm→(Rn)	1111nnnnmmmm1011	1	—
10	1	FMOV DRm,@-Rn	Rn-8→Rn,DRm→(Rn)	1111nnnnmmm01011	1	—
11	0	FMOV.S @(R0,Rm),FRn	(R0+Rm)→FRn	1111nnnnmmmm0110	1	—
12	1	FMOV @(R0,Rm),DRn	(R0+Rm)→DRn	1111nnn0mmmm0110	1	—
13	0	FMOV.S FRm, @(R0,Rn)	FRm→(R0+Rn)	1111nnnnmmmm0111	1	—
14	1	FMOV DRm, @(R0,Rn)	DRm→(R0+Rn)	1111nnnnmmm00111	1	—

(1) 説明

1. FRmの内容をFRnに転送します。
2. DRmの内容をDRnに転送します。
3. FRmの内容をRnが示すアドレスのメモリに転送します。
4. DRmの内容をRnが示すアドレスのメモリに転送します。
5. Rmが示すアドレスのメモリの内容をFRnに転送します。
6. Rmが示すアドレスのメモリの内容をDRnに転送します。
7. Rmが示すアドレスのメモリの内容をFRnに転送し、Rmに4を加算します。
8. Rmが示すアドレスのメモリの内容をDRnに転送し、Rmに8を加算します。
9. Rnから4を減算し、 FRmの内容をそのRnが示すアドレスのメモリに転送します。
10. Rnから8を減算し、 DRmの内容をそのRnが示すアドレスのメモリに転送します。
11. (R0+Rm)が示すアドレスのメモリの内容をFRnに転送します。
12. (R0+Rm)が示すアドレスのメモリの内容をDRnに転送します。
13. FRmの内容を(R0+Rn)が示すアドレスのメモリに転送します。

14. DRmの内容を(R0+Rn)が示すアドレスのメモリに転送します。

(2) 注意

特になし

(3) 動作内容

```
void FMOV(int m,n)                /* FMOV FRm,FRn */
{
    FR[n] = FR[m];
    pc += 2;
}
void FMOV_DR(int m,n)           /* FMOV DRm,DRn */
{
    DR[n>>1] = DR[m>>1];
    pc += 2;
}
void FMOV_STORE(int m,n)       /* FMOV.S FRm,@Rn */
{
    store_int(FR[m],R[n]);
    pc += 2;
}
void FMOV_STORE_DR(int m,n)   /* FMOV DRm,@Rn */
{
    store_quad(DR[m>>1],R[n]);
    pc += 2;
}
 void FMOV_LOAD(int m,n)       /* FMOV.S @Rm,FRn */
{
    load_int(R[m],FR[n]);
    pc += 2;
}
void FMOV_LOAD_DR(int m,n)    /* FMOV @Rm,DRn */
{
    load_quad(R[m],DR[n>>1]);
    pc += 2;
}
void FMOV_RESTORE(int m,n)     /* FMOV.S @Rm+,FRn */
{
```

```
    load_int(R[m],FR[n]);
    R[m] += 4;
    pc += 2;
}
void FMOV_RESTORE_DR(int m,n) /* FMOV @Rm+,DRn */
{
    load_quad(R[m],DR[n>>1]) ;
    R[m] += 8;
    pc += 2;
}
void FMOV_SAVE(int m,n)       /* FMOV.S FRm,@-Rn */
{
    store_int(FR[m],R[n]-4);
    R[n] -= 4;
    pc += 2;
}
void FMOV_SAVE_DR(int m,n)    /* FMOV DRm,@-Rn */
{
    store_quad(DR[m>>1],R[n]-8);
    R[n] -= 8;
    pc += 2;
}
void FMOV_INDEX_LOAD(int m,n)  /* FMOV.S @(R0,Rm),FRn */
{
    load_int(R[0] + R[m],FR[n]);
    pc += 2;
}
void FMOV_INDEX_LOAD_DR(int m,n) /*FMOV @(R0,Rm),DRn */
{
    load_quad(R[0] + R[m],DR[n>>1]);
    pc += 2;
}
void FMOV_INDEX_STORE(int m,n)  /*FMOV.S FRm,@(R0,Rn)*/
{
    store_int(FR[m],R[0]+R[n]);
    pc += 2;
}
void FMOV_INDEX_STORE_DR(int m,n)/*FMOV DRm,@(R0,Rn)*/
```

```
{
    store_quad(DR[m>>1],R[0]+R[n]);
    pc += 2;
}
```

(4) 発生する可能性がある例外

- データTLBミス例外
- データ保護違反例外
- 初期ページ書き込み例外
- データアドレスエラー

10.3.14 FMOV Floating - point MOVe extension 浮動小数点命令

浮動小数点転送

No.	SZ	書式	動作概略	命令コード	実行ステート	T ビット
1	1	FMOV XDm,@Rn	XRm→(Rn)	1111nnnnmmm11010	1	—
2	1	FMOV @Rm,XDn	(Rm)→XDn	1111nnn1mmmm1000	1	—
3	1	FMOV @Rm+,XDn	(Rm)→XDn,Rm+8→Rm	1111nnn1mmmm1001	1	—
4	1	FMOV XDm,@-Rn	Rn-8→ Rn,XDm→(Rn)	1111nnnnmmm11011	1	—
5	1	FMOV @(R0,Rm),XDn	(R0+Rm)→XDn	1111nnn1mmmm0110	1	—
6	1	FMOV XDm,@(R0,Rn)	XDm→(R0+Rn)	1111nnnnmmm10111	1	—
7	1	FMOV XDm,XDn	XDm→XDn	1111nnn1mmm11100	1	—
8	1	FMOV XDm,DRn	XDm→DRn	1111nnn0mmm11100	1	—
9	1	FMOV DRm,XDn	DRm→XDn	1111nnn1mmm01100	1	—

(1) 説明

1. XDmの内容をRnが示すアドレスのメモリに転送します。
2. Rmが示すアドレスのメモリの内容をXDnに転送します。
3. Rmが示すアドレスのメモリの内容をXDnに転送し、Rmに8を加算します。
4. Rnから8を減算し、 XDmの内容をそのRnが示すアドレスのメモリに転送します。
5. (R0+Rm)が示すアドレスのメモリの内容をXDnに転送します。
6. XDmの内容を(R0+Rn)が示すアドレスのメモリに転送します。
7. XDmの内容をXDnに転送します。
8. XDmの内容をDRnに転送します。
9. DRmの内容をXDnに転送します。

(2) 注意

特になし

(3) 動作内容

```
void FMOV_STORE_XD(int m,n)      /* FMOV XDm,@Rn */
{
    store_quad(XD[m>>1],R[n]);
    pc += 2;
}
```

```
void FMOV_LOAD_XD(int m,n)      /* FMOV @Rm,XDn */
{
     load_quad(R[m],XD[n>>1]);
     pc += 2;
}
void FMOV_RESTORE_XD(int m,n)   /* FMOV @Rm+,XDn */
{
     load_quad(R[m],XD[n>>1]);
     R[m] += 8;
     pc += 2;
}
void FMOV_SAVE_XD(int m,n)      /* FMOV XDm,@-Rn */
{
     store_quad(XD[m>>1],R[n]-8);
     R[n] -= 8;
     pc += 2;
}
void FMOV_INDEX_LOAD_XD(int m,n)  /* FMOV @(R0,Rm),XDn */
{
     load_quad(R[0] + R[m],XD[n>>1]);
     pc += 2;
}
void FMOV_INDEX_STORE_XD(int m,n)   /* FMOV XDm,@(R0,Rn) */
{
    store_quad(XD[m>>1], R[0] + R[n]);
    pc += 2;
}
 void FMOV_XDXD(int m,n)       /* FMOV XDm,XDn */
{
    XD[n>>1] = XD[m>>1];
    pc += 2;
}
void FMOV_XDDR(int m,n)       /* FMOV XDm,DRn */
{
    DR[n>>1] = XD[m>>1];
    pc += 2;
}
void FMOV_DRXD(int m,n)       /* FMOV DRm,XDn */
```

```
{
    XD[n>>1] = DR[m>>1];
    pc += 2;
}
```

(4) 発生する可能性がある例外

- データTLBミス例外
- データ保護違反例外
- 初期ページ書き込み例外
- データアドレスエラー

10.3.15 FMUL Floating - point MULtiply 浮動小数点命令

浮動小数点乗算

PR	書式	動作概略	命令コード	実行ステート	Tビット
0	FMUL FRm,FRn	FRn×FRm→FRn	1111nnnnmmmm0010	1	—
1	FMUL DRm,DRn	DRn×DRm→DRn	1111nnn0mmm00010	3	—

(1) 説明

- FPSCR.PR=0の場合

  FRnとFRmの内容の2つの単精度浮動小数点数を算術乗算し、結果をFRnに格納します。

- FPSCR.PR=1の場合

  DRnとDRmの内容の2つの倍精度浮動小数点数を算術乗算し、結果をDRnに格納します。

FPSCR.enable.I がセットされている場合、FPU 例外トラップが、例外の発生如何に関わらず発生します。FPSCR.enable.O/U がセットされている場合、実際に FPU 例外要因が発生したとき、および命令固有の特殊条件が成立したときに FPU 例外トラップが発生します。なお、当該特殊条件を本節の末尾に示します。

例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

(2) 注意

特になし

(3) 動作内容

```
void FMUL(int m,n)
{
    pc += 2;
       clear_cause();
       if((data_type_of(m) == sNaN) ||
           (data_type_of(n) == sNaN)) invalid(n);
       else if((data_type_of(m) == qNaN) ||
                (data_type_of(n) == qNaN)) qnan(n);
       else if((data_type_of(m) == DENORM) ||
                (data_type_of(n) == DENORM)) set_E();
       else switch (data_type_of(m){
               case NORM: switch (data_type_of(n)){
               case PZERO:
```

```
            case NZERO:    zero(n,sign_of(m)^sign_of(n));  break;
            case PINF:
            case NINF      inf(n,sign_of(m)^sign_of(n));  break;
            default:       normal_fmul(m,n);break;
      }      break;
         case PZERO:
         case NZERO: switch (data_type_of(n)){
            case PINF:
            case NINF:       invalid(n);break;
            default:         zero(n,sign_of(m)^sign_of(n));break;
      }     break;
         case PINF :
         case NINF : switch (data_type_of(n)){
            case PZERO:
            case NZERO:invalid(n);break;
             default:
      inf(n,sign_of(m)^sign_of(n));break
      }     break;
      }
}
```

FMUL 特殊ケース（FPSCR.PR=0 の場合）

FMUL	FRn							
FRm	+NORM	-NORM	+0	–0	+inf	–inf	qNaN	sNaN
+NORM	FMUL		+0	–0	+inf	-inf		
-NORM			–0	+0	-inf	+inf		
+0	+0	–0	+0	–0				
–0	–0	+0	–0	+0		invalid		
+inf	+inf	-inf			+inf	-inf		
–inf	-inf	+inf		invalid	-inf	+inf		
qNaN							qNaN	
sNaN								invalid

【注】 FPSCR.DN=0 のとき、DENORM は NORM と同様に計算されます。

FMUL 特殊ケース（FPSCR.PR=1 の場合）

FMUL	DRn									
DRm	+NORM	-NORM	+DENORM	–DENORM	+0	-0	+inf	–inf	qNaN	sNaN
+NORM	FMUL				+0	-0	+inf	-inf		
-NORM					-0	+0	-inf	+inf		
+DENORM					+0	-0	+inf	-inf		
–DENORM				Error	-0	+0	-inf	+inf		
+0	+0	-0	+0	-0	+0	-0				
-0	-0	+0	-0	+0	-0	+0		invalid		
+inf	+inf	-inf	+inf	-inf			+inf	-inf		
–inf	-inf	+inf	-inf	+inf		invalid	-inf	+inf		
qNaN									qNaN	
sNaN										invalid

【注】 FPSCR.DN=1 のとき、DENORM を 0 として扱います。

(4) 発生する可能性がある例外とオーバフロー／アンダフロー例外トラップの発生条件

- FPUエラー
- 無効演算
- オーバフロー

  オーバフロー例外トラップの発生条件

  ・FPSCR.PR=0の場合：FRnの指数部＋FRmの指数部－H'7FがH'FE以上

  ・FPSCR.PR=1の場合：DRnの指数部＋DRmの指数部－H'3FFがH'7FE以上

- アンダフロー

  アンダフロー例外トラップの発生条件

  ・FPSCR.PR=0で

  FRn、FRmの双方が正規化数の場合：FRnの指数部＋FRmの指数部－H'7FがH'00以下

  FRn、FRmの少なくとも一方が非正規化数の場合：FRnの指数部＋FRmの指数部－H'7FがH'18以下

  ・FPSCR.PR=1の場合：DRnの指数部＋DRmの指数部－H'3FFがH'000以下

- 不正確例外

10.3.16 FNEG Floating - point NEGate value 浮動小数点命令

浮動小数点符号反転

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FNEG FRn	FRn ^ H'80000000→FRn	1111nnnn01001101	1	—
1	FNEG DRn	DRn ^ H'8000000000000000→DRn	1111nnn001001101	1	—

(1) 説明

浮動小数点レジスタ FRn/DRn の内容の最上位ビット(符号ビット) を反転して、結果を FRn/DRn に格納します。FPSCR の cause/flag 部分は更新されません。

(2) 注意

特になし

(3) 動作内容

```
void FNEG (int n){
    FR[n] = -FR[n];
    pc += 2;
}

/* 精度に依存せず、同じ動作を行います。 */
```

(4) 発生する可能性がある例外

なし

10.3.17 FPCHG Pr-bit ChanGe 浮動小数点命令

PR ビット反転

書式	動作概略	命令コード	実行ステート	T ビット
FPCHG	~FPSCR.PR → FPSCR.PR	1111011111111101	1	—

(1) 説明

浮動小数点状態レジスタ FPSCR の PR ビットを反転します。FPSCR の PR ビットを換えると、単精度演算と倍精度演算を切り替えることができます。

(2) 注意

特になし

(3) 動作内容

```
void FPCHG(){/* FPCHG */}
{
    FPSCR ^= 0x00080000; /* bit 19 */
    PC += 2;
}
```

(4) 発生する可能性がある例外

なし

10.3.18 FRCHG FR-bit CHanGe 浮動小数点命令

FR ビット反転

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FRCHG	~FRSCR.FR→FRSCR.FR	1111101111111101	1	—
1	—	—	—	—	—

(1) 説明

浮動小数点状態レジスタ FPSCR の FR ビットを反転します。FPSCR の FR ビットを換えると、FPR0_BANK0 から FPR15_BANK0 および FPR0_BANK1 から FPR15_BANK1 の中の、FR0 から FR15 が、XR0 から XR15 になり、XR0 から XR15 が、FR0 から FR15 になります。FPSCR.FR=0 のとき、FPR0_BANK0 から FPR15_BANK0 が FR0 から FR15 に対応し、FPR0_BANK1 から FPR15_BANK1 が XR0 から XR15 に対応します。FPSCR.FR=1 のとき、FPR0_BANK1 から FPR15_BANK1 が FR0 から FR15 に対応し、FPR0_BANK0 から FPR15_BANK0 が XR0 から XR15 に対応します。

(2) 注意

特になし

(3) 動作内容

```
void FRCHG()   /* FRCHG */
{
      if(FPSCR_PR == 0){
           FPSCR ^= 0x00200000; /* bit 21 */
           PC += 2;
      }
      else undefined_operation();
}
```

(4) 発生する可能性がある例外

なし

10.3.19 FSCA Floating Point Sine And Cosine Approximate 浮動小数点命令

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FSCA FPUL,DRn	sin(FPUL)→FRn cos(FPUL)→FR[n+1]	1111nnn011111101	3	—
1	—	reserved	1111nnnn11111101	—	—

(1) 説明

FPUL の示す角度の sin および cos の近似値（絶対誤差±2 ^ -21 未満）を単精度浮動小数点として求め、sin 値を FRn、cos 値を FR[n+1]に書き込みます。近似演算命令のため、常に不正確例外を要求します（入力が 0 の場合でも不正確とします）。

FPSCR.enable.I がセットされている場合、FPU 例外トラップが発生します。例外発生時は、FPSCR.cause、FPSCR.flag には、例外の正しい情報が反映され、FRn/FR[n+1]は更新されません。ソフトウェアで適切な処理を行ってください。

(2) 注意

特になし

(3) 動作内容

```
void FSCA(int n)
{
     float angle;
     long offset = 0x00010000;
     long fraction = 0x0000ffff;
     case(FPSCR_PR){
          0: clear_cause ();
                    set_I();
   /* extract sub-rotation (fraction) part */
                    fraction &= FPUL;
   /* convert to float */
                    angle = fraction;
   /* convert to radian */
                    angle = 2*M_PI*angle / offset;
                    FR[n] ~= sin(angle);
                    FR[n+1] ~= cos(angle);
```

```
                    pc += 2; break;
            1: undefined_operation();     /* reserved */
        }
    }
```

(4) ソースオペランドのデータ形式：角度の指定方法

2 の補数表現である符号付小数点数で表現される回転数で角度指定を行います。sin/cos の結果は単精度浮動小数点数です。

0x7FFF FFFF～0x0000 0001	：360×2^{15}-$360/2^{16}$ ～ $360/2^{16}$ 度
0x0000 0000	：0 度
0xFFFF FFFF～0x8000 0000	：$-360/2^{16}$ ～ -360×2^{15} 度

(5) 発生する可能性のある例外

- 不正確例外

10.3.20 FSCHG Sz-bit CHanGe 浮動小数点命令

SZ ビット反転

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FSCHG	～FPSCR.SZ→FPSCR.SZ	1111001111111101	1	―

(1) 説明

浮動小数点状態レジスタ FPSCR の SZ ビットを反転します。FPSCR の SZ ビットを換えると、FMOV 命令のデータ転送が、単精度データ 1 つか、ペアかが切り替わります。FPSCR.SZ=0 のとき、FMOV 命令は単精度データを一つ転送します。FPSCR.SZ=1 のとき、FMOV 命令は単精度データをペアで 2 つ転送します。

(2) 注意

特になし

(3) 動作内容

```
void FSCHG()   /* FSCHG */
{
     if(FPSCR_PR == 0){
          FPSCR ^= 0x00100000; /* bit 20 */
          PC += 2;
     }
     else undefined_operation();
}
```

(4) 発生する可能性がある例外

なし

10.3.21 FSQRT Floating - point SQuare RooT 浮動小数点命令

浮動小数点平方根

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FSQRT FRn	sqrt(FRn)*→FRn	1111nnnn01101101	14	—
1	FSQRT DRn	sqrt(DRn)*→DRn	1111nnn001101101	30	—

【注】 * sqrt(FRn)は FRn の平方根、sqrt(DRn)は DRn の平方根を表します。

(1) 説明

- FPSCR.PR=0の場合

  FRnの内容の単精度浮動小数点数の算術平方根を求め、結果をFRnに格納します。

- FPSCR.PR=1の場合

  DRnの内容の倍精度浮動小数点数の算術平方根を求め、結果をDRnに格納します。

FPSCR.enable.I がセットされている場合、FPU 例外トラップが、例外の発生如何に関わらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

(2) 注意

特になし

(3) 動作内容

```
void FSQRT(int n){
     pc += 2;
     clear_cause();
     switch(data_type_of(n)){
          case NORM  :  if(sign_of(n) == 0) normal_fsqrt(n);
                        else    invalid(n); break;
          case DENORM:  if(sign_of(n) == 0) set_E();
                        else    invalid(n); break;
          case PZERO :
          case NZERO :
          case PINF  :  break;
          case NINF  :  invalid(n); break;
          case qNaN  :  qnan(n); break;
```

```
          case sNaN  :  invalid(n); break;
     }
}
void normal_fsqrt(int n)
{
union {
      float f;
      int l;
}     dstf,tmpf;
union {
      double d;
      int l[2];
}     dstd,tmpd;
union {
      int double x;
      int l[4];
}     tmpx;
      if(FPSCR_PR == 0) {
         tmpf.f = FR[n]; /* save destination value */
         dstf.f = sqrt(FR[n]); /* round toward nearest or even */
         tmpd.d = dstf.f; /* convert single to double */
         tmpd.d *= dstf.f;
         if(tmpf.f != tmpd.d) set_I();
         if((tmpf.f < tmpd.d) && (FPSCR_RM == 1))
            dstf.l -= 1; /* round toward zero */
         if(FPSCR & ENABLE_I)fpu_exception_trap();
         else                    FR[n] = dstf.f;
      } else {
         tmpd.d = DR[n>>1]; /* save destination value */
         dstd.d = sqrt(DR[n>>1]); /* round toward nearest or even */
         tmpx.x = dstd.d; /* convert double to int double */
         tmpx.x *= dstd.d;
         if(tmpd.d != tmpx.x) set_I();
         if((tmpd.d < tmpx.x) && (FPSCR_RM == 1)) {
            dstd.l[1] -= 1; /* round toward zero */
            if(dstd.l[1] == 0xffffffff) dstd.l[0] -= 1;
         }
         if(FPSCR & ENABLE_I)fpu_exception_trap();
```

```
            else                                DR[n>>1] = dstd.d;
        }
    }
```

FSQRT 特殊ケース

FRn	+NORM	–NORM	+DENORM	–DENORM	+0	–0	+INF	–INF	qNaN	sNaN
FSQRT(FRn)	SQRT	Invalid	Error	Error	+0	–0	+INF	Invalid	qNaN	Invalid

【注】 DN=1 の場合、非正規化数の値は 0 として扱われます。

（4） 発生する可能性がある例外

- FPUエラー
- 無効演算
- 不正確例外

10.3.22 FSRRA Floating Ponit Square Reciprocal Approximate 浮動小数点命令

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FSRRA FRn	1/sqrt(FRn)*→FRn	1111nnnn01111101	1	—
1	—	reserved	1111nnnn01111101		

【注】* sqrt(FRn)は FRn の平方根を表します。

(1) 説明

FRn の内容である単精度浮動小数点の算術的な平方根の逆数の近似値（絶対誤差±2 ^ -21 未満）を FRn に書き込みます。近似値演算命令のため、入力が正規化数の場合は不正確例外を要求します。その他の場合は、不正確例外を要求しません。

FPSCR.enable.I がセットされている場合、FPU 例外トラップが発生します。例外発生時は、FPSCR.cause、FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

(2) 注意

特になし

(3) 動作内容

```
vaid FSRRA(int n){
     case(FPSCR_PR){
          0:       fsrra_single(n);        break;
          1:       undefined_operation();  break;
     }
     PC += 2;
}
fsrra_single(int n)
{    clear_cause();
     case(data_type_of(n)){
          NORM:    if(sign_of(n)==0) {
                        Set_I();
                        FR[n] ~= 1/sqrt(FR[n]);
                   }
                   else invalid(n);        break;
          DENORM:  if(sign_of(n)==0)
                   fpu_error();            break;
```

```
                    else invalid(n);        break;
            PZERO:
            NZERO:  dz(n,sign_of(n));       break;
            PINF:   FR[n]=0;                break;
            NINF:   invalid(n);             break;
            qNAN:   qnan(n);                break;
            sNAN    invalid(n);             break;
        }
    }
```

FSRRA 特殊ケース

FRn	+NORM	–NORM	+DENORM	–DENORM	+0	–0	+INF	–INF	qNaN	sNaN
FSRRA(FRn)	1/SQRT	Invalid	Error	Invalid	DZ	DZ	+0	Invalid	qNaN	Invalid

【注】 DN=1 の場合、非正規化数の値は 0 として扱われます。

(4) 発生する可能性がある例外

- FPUエラー
- 無効演算
- 0による除算
- 不正確例外

10.3.23 FSTS　Floating - point Store System register　浮動小数点命令

システムレジスタからの転送

書式	動作概略	命令コード	実行ステート	T ビット
FSTS　FPUL,FRn	FPUL→FRn	1111nnnn00001101	1	—

(1) 説明

システムレジスタ FPUL の内容を浮動小数点レジスタ FRn に転送します。

(2) 注意

特になし

(3) 動作内容

```
void FSTS(int n, float *FPUL)
{
    FR[n] = *FPUL;
    pc += 2;
}
```

(4) 発生する可能性がある例外

なし

10.3.24 FSUB Floating - point SUBtract 浮動小数点命令

浮動小数点減算

PR	書式	動作概略	命令コード	実行ステート	Tビット
0	FSUB FRm,FRn	FRn-FRm→FRn	1111nnnnmmmm0001	1	—
1	FSUB DRm,DRn	DRn-DRm→DRn	1111nnn0mmm00001	1	—

(1) 説明

- FPSCR.PR=0の場合

  FRnとFRmの内容の2つの単精度浮動小数点数を算術減算し、結果をFRnに格納します。

- FPSCR.PR=1の場合

  DRnとDRmの内容の2つの倍精度浮動小数点数を算術減算し、結果をDRnに格納します。

FPSCR.enable.I がセットされている場合、FPU 例外トラップが、例外の発生如何に関わらず発生します。FPSCR.enable.O/U がセットされている場合、実際に FPU 例外要因が発生したとき、および命令固有の特殊条件が成立したときに FPU 例外トラップが発生します。なお、当該特殊条件を本節の末尾に示します。

例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

(2) 注意

特になし

(3) 動作内容

```
void FSUB (int m,n)
{
     pc += 2;
     clear_cause();
     if((data_type_of(m) == sNaN) ||
        (data_type_of(n) == sNaN)) invalid(n);
     else if((data_type_of(m) == qNaN) ||
             (data_type_of(n) == qNaN)) qnan(n);
     else if((data_type_of(m) == DENORM) ||
             (data_type_of(n) == DENORM)) set_E();
     else switch (data_type_of(m)){
        case NORM: switch          pe_of(n)){
```

```
            case NORM:     normal_faddsub(m,n,SUB); break;
            case PZERO:
            case NZERO:register_copy(m,n); FR[n] = -FR[n];break;
            default:       break;
          }          break;
         case PZERO:    break;
         case NZERO: switch (data_type_of(n)){
            case NZERO:zero(n,0); break;
            default:     break;
          }          break;
         case PINF: switch (data_type_of(n)){
            case PINF:     invalid(n);    break;
            default:       inf(n,1);      break;
          }          break;
         case NINF: switch (data_type_of(n)){
            case NINF:     invalid(n);    break;
            default:       inf(n,0);      break;
          }          break;
      }
  }
```

FSUB 特殊ケース

FSUB	FRn,DRn							
FRm,DRm	+NORM	-NORM	+0	–0	+inf	–inf	qNaN	sNaN
+NORM	FSUB							
-NORM								
+0				–0				
–0			+0		+inf			
+inf				–inf	invalid	–inf		
–inf					+inf	invalid		
qNaN							qNaN	
sNaN								invalid

【注】 FPSCR.DN=1 のとき、DENORM を 0 として扱います。

FPSCR.DN=0 のとき、DENORM は NORM と同様に計算されます。

(4) 発生する可能性がある例外とオーバフロー／アンダフロー例外トラップの発生条件

- FPUエラー
- 無効演算
- オーバフロー

  オーバフロー例外トラップの発生条件

  ・FPSCR.PR=0の場合：FRnとFRmが異符合かつ少なくとも一方の指数部がH'FE

  ・FPSCR.PR=1の場合：DRnとDRmが異符合かつ少なくとも一方の指数部がH'7FE
- アンダフロー

  アンダフロー例外トラップの発生条件

  ・FPSCR.PR=0の場合：FRnとFRmが同符合かつ双方の指数部がH'18以下

  ・FPSCR.PR=1の場合：DRnとDRmが同符合かつ双方の指数部がH'035以下
- 不正確例外

10.3.25 FTRC Floating - point Truncate and Convert to integer 浮動小数点命令整数への変換

PR	書式	動作概略	命令コード	実行ステート	Tビット
0	FTRC FRm,FPUL	(long)FRm→FPUL	1111mmmm00111101	1	—
1	FTRC DRm,FPUL	(long)DRm→FPUL	1111mmm000111101	1	—

(1) 説明

- FPSCR.PR=0の場合

  FRmの内容の単精度浮動小数点数を32ビット整数に変換し、結果をFPULに格納します。

- FPSCR.PR=1の場合

  DRmの内容の倍精度浮動小数点数を32ビット整数に変換し、結果をFPULに格納します。

  丸めモードは常に切り捨てになります。

(2) 注意

特になし

(3) 動作内容

```
#define N_INT_SINGLE_RANGE 0xcf000000 & 0x7fffffff  /* -1.000000 * 2^31 */
#define P_INT_SINGLE_RANGE 0x4effffff  /* 1.fffffe * 2^30 */
#define N_INT_DOUBLE_RANGE 0xc1e0000000200000 & 0x7fffffffffffffff
#define P_INT_DOUBLE_RANGE 0x41e0000000000000

void FTRC(int m,int *FPUL)
{
    pc += 2;
    clear_cause();
    if(FPSCR.PR==0){
        case(ftrc_single_ type_of(m)){
        NORM:    *FPUL = FR[m];    break;
        PINF:    ftrc_invalid(0, *FPUL); break;
        NINF:    ftrc_invalid(1, *FPUL); break;
        }
    }
    else{               /* case FPSCR.PR=1 */
        case(ftrc_double_type_of(m)){
```

```
        NORM:    *FPUL = DR[m>>1]; break;
        PINF:    ftrc_invalid(0, *FPUL); break;
        NINF:    ftrc_invalid(1, *FPUL); break;
        }
    }
}
int ftrc_single_type_of(int m)
{
       if(sign_of(m) == 0){
            if(FR_HEX[m] > 0x7f800000)     return(NINF);      /* NaN */
            else if(FR_HEX[m] > P_INT_SINGLE_RANGE)
                       return(PINF);       /* out of range,+INF */
            else       return(NORM);       /* +0,+NORM        */
       } else {
            if((FR_HEX[m] & 0x7fffffff) > N_INT_SINGLE_RANGE)
                       return(NINF);       /* out of range ,+INF,NaN*/
            else       return(NORM);       /* -0,-NORM           */
    }
}
int ftrc_double_type_of(int m)
{
       if(sign_of(m)==0){
            if((FR_HEX[m] > 0x7ff00000) ||
              ((FR_HEX[m] == 0x7ff00000) &&
               (FR_HEX[m+1] != 0x00000000)))  return(NINF);     /* NaN */
            else if(DR_HEX[m>>1] >= P_INT_DOUBLE_RANGE)
                        return(PINF);      /* out of range,+INF */
            else        return(NORM);      /* +0,+NORM        */
       } else   {
            if((DR_HEX[m>>1] & 0x7fffffffffffffff) >= N_INT_DOUBLE_RANGE)
                        return(NINF);      /* out of range ,+INF,NaN*/
            else        return(NORM);      /* -0,-NORM           */
       }
}
void ftrc_invalid(int sign,int *FPUL)
{
    set_V();
     if((FPSCR&ENABLE_V) == 0){
```

```
            if(sign == 0) *FPUL = 0x7fffffff;
            else          *FPUL = 0x80000000;
        }
        else fpu_exception_trap();
    }
```

FTRC 特殊ケース

FRn,DRn	NORM	+0	–0	Positive Out of Range	Negative Out of Range	+INF	–INF	qNaN	sNaN
FTRC (FRn,DRn)	TRC	0	0	Invalid +MAX	Invalid –MAX	Invalid +MAX	Invalid –MAX	Invalid –MAX	Invalid –MAX

【注】 DN=1 の場合、非正規化数の値は 0 として扱われます。

(4) 発生する可能性がある例外

- 無効演算

10.3.26 FTRV Floating - point TRansform Vector 浮動小数点命令

ベクトル変換

PR	書式	動作概略	命令コード	実行ステート	Tビット
0	FTRV XMTRX,FVn	transform_vector(XMTRX, FVn)→ FVn	1111nn0111111101	4	—
1	—	—	—	—	—

(1) 説明

- FPSCR.PR=0の場合

  XMTRXで示される、浮動小数点レジスタXF0からXF15の内容を4行4列の行列とし、FVnで示される、浮動小数点レジスタFR[n]から FR[n+3]の内容を4次元のベクトルとして、行列とベクトルの乗算を行い、結果をFVnに格納します。

XMTRX FVn FVn

$$\begin{pmatrix} XF[0] & XF[4] & XF[8] & XF[12] \\ XF[1] & XF[5] & XF[9] & XF[13] \\ XF[2] & XF[6] & XF[10] & XF[14] \\ XF[3] & XF[7] & XF[11] & XF[15] \end{pmatrix} \times \begin{pmatrix} FR[n] \\ FR[n+1] \\ FR[n+2] \\ FR[n+3] \end{pmatrix} \rightarrow \begin{pmatrix} FR[n] \\ FR[n+1] \\ FR[n+2] \\ FR[n+3] \end{pmatrix}$$

FTRV 命令は正確さよりも高速化のための命令です。そのため、FADD や FMUL を組み合わせて使用した場合と、結果が異なります。FTRV は次の順序で実行します。

1. 各項をそれぞれ乗算します。結果は28ビットです。
2. それらの結果をアライメントします。このとき、30ビット以内に入るように丸めます。
3. アライメントした結果を加算します。
4. 正規化と丸めを行います。

以下の場合、特別な処理になります。

1. 入力値にsNaNがある場合、無効例外になります。
2. 乗算する入力値で0と無限大の組み合わせがある場合、無効演算例外になります。
3. 上記以外で、入力値にqNaNが含まれる場合、結果はqNaNになります。
4. 上記以外に入力値に無限大がある場合、

- もしも乗算した結果が2つ以上無限大になり、符号が異なる場合、無効演算例外になります。

- 上記以外の場合、正しい無限大が格納されます。

1. 入力値にsNaN、qNaN、無限大が含まれない場合は、通常と同じ処理を行います。

FPSCR.enable.V/O/U/I がセットされている場合、FPU 例外トラップが、例外の発生如何に関わらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FVn は更新されません。ソフトウェアで適切な処理を行ってください。

(2) 注意

特になし

(3) 動作内容

```
void FTRV (int n)     /* FTRV FVn */
{
float saved_vec[4],result_vec[4];
int saved_fpscr;
int dst,i;
      if(FPSCR_PR == 0)      {
          PC += 2;
          clear_cause();
          saved_fpscr = FPSCR;
          FPSCR &= ~ENABLE_VOUI; /* mask VOUI enable */
          dst = 12 - n;   /* select other vector than FVn */
          for (i=0;i<4;i++) saved_vec [i] = FR[dst+i];
          for (i=0;i<4;i++) {
              for(j=0;j<4;j++)FR[dst+j] = XF[i+4j];
              fipr(n,dst);
              saved_fpscr |= FPSCR & (CAUSE|FLAG) ;
              result_vec[i] = FR[dst+3];
          }
          for (i=0;i<4;i++) FR[dst+i] = saved_vec[i];
          FPSCR = saved_fpscr;
          If(FPSCR & ENABLE_VOUI) fpu_exception_trap();
          Else for (i=0;i<4;i++)    FR[n+i] = result_vec[i];
      }
      else undefined_operation();
}
```

(4) 発生する可能性がある例外

- 無効演算
- オーバフロー
- アンダフロー
- 不正確例外

11. レジスター覧

本章では、内蔵 I/O レジスタについて、各章で説明された内容を一覧表の形でまとめて説明しています。

1. レジスタアドレス一覧（機能モジュールごと、マニュアル章番号順）

- 機能モジュールごとに、マニュアルの章番号の順に表記しています。
- 本リストに記載されていないリザーブアドレスは、アクセスしないでください。

2. 各動作モードにおけるレジスタの状態

- 「レジスタアドレス一覧（機能モジュールごと、マニュアル章番号順）」の並びに従って、レジスタの状態を表記しています。
- 初期化時の各ビットの状態は、該当する章のレジスタ説明を参照してください。
- 基本的な動作モード時のレジスタの状態を示しており、内蔵モジュール固有のリセットなどがある場合は、内蔵モジュールの章を参照してください。

11.1 レジスタアドレス一覧（機能モジュールごと、マニュアル章番号順）

【注】 未定義およびリザーブアドレスのアクセスは、禁止します。これらのレジスタをアクセスした時の動作および継続する動作については保証できませんので、アクセスしないようにしてください。

モジュール名	名　称	略称	R/W	P4 領域 アドレス*	エリア 7 アドレス*	アクセス サイズ
例外処理	TRAPA 例外レジスタ	TRA	R/W	H'FF00 0020	H'1F00 0020	32
	例外事象レジスタ	EXPEVT	R/W	H'FF00 0024	H'1F00 0024	32
	割り込み事象レジスタ	INTEVT	R/W	H'FF00 0028	H'1F00 0028	32
MMU	ページテーブルエントリ上位レジスタ	PTEH	R/W	H'FF00 0000	H'1F00 0000	32
	ページテーブルエントリ下位レジスタ	PTEL	R/W	H'FF00 0004	H'1F00 0004	32
	変換テーブルベースレジスタ	TTB	R/W	H'FF00 0008	H'1F00 0008	32
	TLB 例外アドレスレジスタ	TEA	R/W	H'FF00 000C	H'1F00 000C	32
	MMU 制御レジスタ	MMUCR	R/W	H'FF00 0010	H'1F00 0010	32
	物理アドレス空間制御レジスタ	PASCR	R/W	H'FF00 0070	H'1F00 0070	32
	命令再フェッチ抑止制御レジスタ	IRMCR	R/W	H'FF00 0078	H'1F00 0078	32
キャッシュ	キャッシュ制御レジスタ	CCR	R/W	H'FF00 001C	H'1F00 001C	32
	キューアドレス制御レジスタ 0	QACR0	R/W	H'FF00 0038	H'1F00 0038	32
	キューアドレス制御レジスタ 1	QACR1	R/W	H'FF00 003C	H'1F00 003C	32
	内蔵メモリ制御レジスタ	RAMCR	R/W	H'FF00 0074	H'1F00 0074	32
L メモリ	L メモリ転送元アドレスレジスタ 0	LSA0	R/W	H'FF00 0050	H'1F00 0050	32
	L メモリ転送元アドレスレジスタ 1	LSA1	R/W	H'FF00 0054	H'1F00 0054	32
	L メモリ転送先アドレスレジスタ 0	LDA0	R/W	H'FF00 0058	H'1F00 0058	32
	L メモリ転送先アドレスレジスタ 1	LDA1	R/W	H'FF00 005C	H'1F00 005C	32

【注】 * P4 領域アドレスは、仮想アドレス空間の P4 領域を用いた場合のものです。エリア 7 アドレスは。TLB を用いて物理アドレス空間のエリア 7 からアクセスするものです。

11.2 各動作モードにおけるレジスタの状態

【注】* レジスタの初期値は、各モジュールの章を参照してください。また、初期値が不定のレジスタについても値が保持されないため、初期化と表現しています。

モジュール名	名称	略称	パワーオンリセット	マニュアルリセット	スリープ	スタンバイ
例外処理	TRAPA 例外レジスタ	TRA	不定	不定	保持	保持
	例外事象レジスタ	EXPEVT	H'0000 0000	H'0000 0020	保持	保持
	割り込み事象レジスタ	INTEVT	不定	不定	保持	保持
MMU	ページテーブルエントリ上位レジスタ	PTEH	不定	不定	保持	保持
	ページテーブルエントリ下位レジスタ	PTEL	不定	不定	保持	保持
	変換テーブルベースレジスタ	TTB	不定	不定	保持	保持
	TLB 例外アドレスレジスタ	TEA	不定	保持	保持	保持
	MMU 制御レジスタ	MMUCR	H'0000 0000	H'0000 0000	保持	保持
	物理アドレス空間制御レジスタ	PASCR	H'0000 0000	H'0000 0000	保持	保持
	命令再フェッチ抑止制御レジスタ	IRMCR	H'0000 0000	H'0000 0000	保持	保持
キャッシュ	キャッシュ制御レジスタ	CCR	H'0000 0000	H'0000 0000	保持	保持
	キューアドレス制御レジスタ 0	QACR0	不定	不定	保持	保持
	キューアドレス制御レジスタ 1	QACR1	不定	不定	保持	保持
	内蔵メモリ制御レジスタ	RAMCR	H'0000 0000	H'0000 0000	保持	保持
L メモリ	L メモリ転送元アドレスレジスタ 0	LSA0	不定	不定	保持	保持
	L メモリ転送元アドレスレジスタ 1	LSA1	不定	不定	保持	保持
	L メモリ転送先アドレスレジスタ 0	LDA0	不定	不定	保持	保持
	L メモリ転送先アドレスレジスタ 1	LDA1	不定	不定	保持	保持

付録

A. CPU 動作モードレジスタ（CPUOPM）

CPUOPM は、CPU の動作モードを切り替えるために使用します。本レジスタは P4 領域の H'FF2F0000 あるいはエリアアドレスの H'1F2F0000 から 32 ビットサイズで読み出し／書き込みが可能です。本レジスタへ書き込む際には、必ずリザーブビットに初期値を書き込むようにしてください。リザーブビットに初期値以外の値を書き込んだ場合の動作は保証されません。

CPUOPM の更新は、CPU 以外の SuperHyway バスマスタからのアクセスでなく、CPU のストア命令で行ってください。また、CPUOPM 更新後、一度 CPUOPM を読み出した後で、以下の 1.または 2.のどちらかを実行してください。

1. RTE命令による分岐を実行してください。
2. 任意のアドレス（キャッシング不可領域でもよい）に対して、ICBI命令を実行してください。

1.または 2.の実行後、CPU は更新後の CPUOPM の値を用いて動作することが保証されます。

ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	—	—	—	—	—	—	RABD	—	INTMU	—	—	—
初期値：	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R/W	R	R/W	R	R	R

ビット	ビット名	初期値	R/W	説　　明
31～6	—	H'000000F	R	リザーブビット 書き込み時は、必ず初期値を書き込むようにしてください。
5	RABD	1	R/W	サブルーチン復帰投機実行ビット 0：サブルーチンからの復帰時に命令フェッチを投機的に発行します。本ビットを 0 に設定する場合は、「付録 C.　サブルーチン復帰投機実行」を参照してください。 1：サブルーチンからの復帰時に命令フェッチを投機的に発行しません。
4	—	0	R	リザーブビット 書き込み時は、必ず初期値を書き込むようにしてください。
3	INTMU	0	R/W	割り込み動作モード切り替えビット 0：割り込みを受理しても SR.IMASK の値は変化しません。 1：割り込みを受理した場合、受け付けたレベルを SR.IMASK の値に自動的に設定します。
2～0	—	000	R	リザーブビット 書き込み時は、必ず初期値を書き込むようにしてください。

B. 命令プリフェッチとその副作用について

SH-4A は、先読みした命令を保持するためのバッファを内部に設けており、常に命令の先読みを行っています。したがって、各メモリ空間の最終 64 バイト領域にプログラムを配置しないでください。その領域にプログラムを配置した場合、メモリエリアを超えて命令の先読みのためのバスアクセスが発生する場合があります。

以下にこれが問題となるケースを示します。

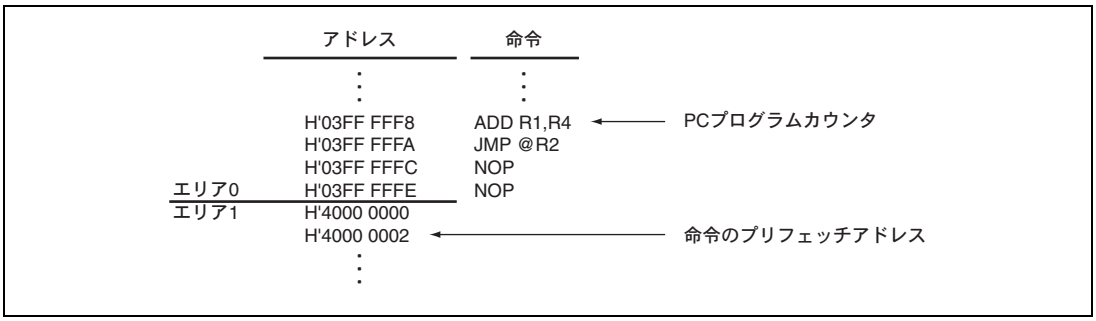


図 B.1　命令のプリフェッチ

図 B.1 では、PC（プログラムカウンタ）が指し示す命令（ADD）と、H'0400 0002 番地の命令フェッチが同時に行われるケースを想定しています。また、プログラムは、後続の JMP 命令、ディレイスロット命令の実行後、エリア 1 以外の領域に分岐するものと仮定します。

この場合、プログラムのフローから想定し得ないエリア 1 へのバスアクセス（命令のプリフェッチ）が発生する可能性があります。

(1) 命令のプリフェッチの副作用

1. 命令プリフェッチが引き起こす外部バスアクセスが原因でその領域に接続されたFIFOなどの外部デバイスが誤動作する場合が考えられます。
2. 命令プリフェッチが引き起こす外部バス要求に応答するデバイスが存在しない場合、ハングアップの原因になります。

(2) 回避方法

1. MMUを用いることで、これら不当な命令フェッチを回避することが可能です。
2. 各エリア最終64バイトの領域にプログラムを配置しないことで、回避することが可能です。

C. サブルーチン復帰投機実行

SH-4A はサブルーチンからの復帰時に命令フェッチを投機的に発行する仕組みを内部に持っています。サブルーチンからの復帰時に命令フェッチを投機的に発行することにより、復帰時の実行サイクルを短縮することができます。この機能は CPU 動作モードレジスタ（CPUOPM）のビット 5（RABD）の値を 0 に設定すると有効になります。しかしサブルーチンからの復帰時に命令フェッチを投機的に発行すると、プログラム上アクセスするはずのないアドレスに対する命令フェッチが起きる場合があります。その結果、想定し得ないエリアへのバスアクセスが発生したり、内部的に命令アドレスエラーが発生して誤動作を引き起こす可能性があります。想定し得ないエリアへのバスアクセスが発生することによる副作用は、**「付録 B.（1） 命令のプリフェッチの副作用」**を参照してください。

使用条件：

サブルーチン復帰投機実行の機能を有効にする場合、サブルーチンからの復帰は JSR/BSR/BSRF 命令で PR に設定した戻りアドレスに対して、RTS 命令を使って行うようにしてください。これによりプログラム上アクセスするはずのないアドレスに対するアクセスを抑止でき、誤動作を回避することが可能です。

D. バージョンレジスタ（PVR、PRR）

SH-4A は、プロセッサコアのバージョンと製品のバージョンを示す読み出し専用のレジスタを内蔵しています。これらのレジスタの値を用いることにより、ソフトウェアからプロセッサのバージョンおよび製品を区別することができ拡張性の高いシステムを構築することが可能となります。バージョンレジスタの値は製品ごとに異なるため、詳細は各製品のハードウェアマニュアルを参照するか、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。

【注】 PVR レジスタのビット 7～ビット 0 と、PRR レジスタのビット 3～ビット 0 の値は必ずマスクをし、ソフトウェアに影響を与えないようにしてください。

表 D.1 レジスタ構成

名称	略称	R/W	P4 領域 アドレス	エリア 7 アドレス	サイズ
プロセッサバージョンレジスタ	PVR	R	H'FF00 0030	H'1F00 0030	32
プロダクトレジスタ	PRR	R	H'FF00 0044	H'1F00 0044	32

- PVR

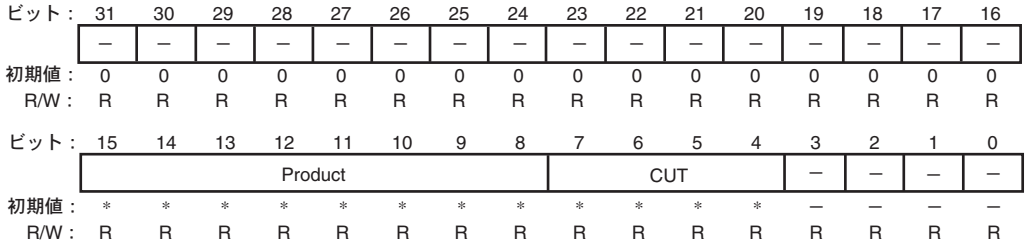
ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	CHIP								VER							
初期値：	0	0	0	1	0	0	0	0	*	*	*	*	*	*	*	*
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CUT								−	−	−	−	−	−	−	−
初期値：	*	*	*	*	*	*	*	*	−	−	−	−	−	−	−	−
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

ビット	ビット名	初期値	R/W	説　　明
31～24	CHIP	H'10	R	プロセッサファミリの種別を示します。 SH-4A シリーズでは、必ず、H'10 が読み出されます。
23～16	VER	*	R	バージョンを示します。 SH-4A シリーズに大幅な機能拡張を行う場合に変更します。 本マニュアルでは、VER が H'20 の場合のみを対象としています。
15～8	CUT	*	R	バージョンを示します。 SH-4A シリーズに小規模な修正を行う場合に変更します。 本ビットは製品により異なります。
7～0	−	不定	R	不定値が読み出されます。 ソフトウェアからは読み出し後に必ずマスクをして使用してください。

【注】 * 本ビットの値は製品により異なります。

- PRR

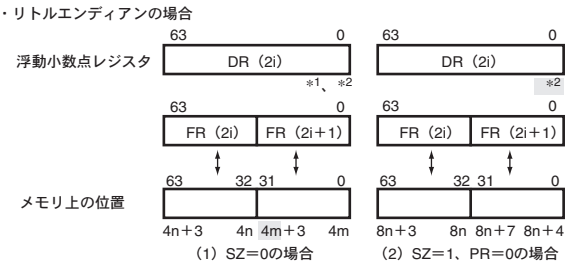


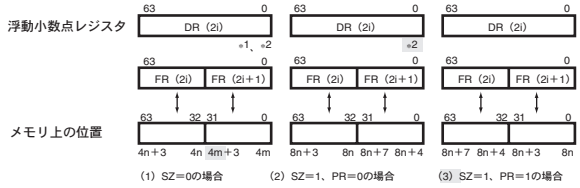
ビット	ビット名	初期値	R/W	説　　明
31～16	—	すべて 0	R	すべて 0 固定です。
15～8	Product	*	R	製品種別を示します。 本ビットの値は、製品により異なります。
7～4	CUT	*	R	バージョンを示します。 製品に小規模な修正を行う場合に変更します。本ビットは製品により異なります。
3～0	—	不定	R	不定値が読み出されます。 ソフトウェアからは読み出し後に必ずマスクをして使用してください。

【注】　*　本ビットの値は製品により異なります。

本版で修正または追加された箇所

<table>
<tr><th>項　　目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>はじめに</td><td>—</td><td>修正。
SH-4A は、ルネサス テクノロジオリジナルの RISC 方式の CPU をコア~~にして、システム構成に必要な周辺機能を集積した RISC マイコン~~です。</td></tr>
<tr><td>1.1　SH-4A の特長</td><td>1-1</td><td>修正。
SH-4A は、SH-1、SH-2、SH-3、~~SH-3E、~~SH-4 マイクロコンピュータと命令セットレベルでの上位互換性を特長とする 32 ビット RISC（縮小命令セットコンピュータ）マイコンです。</td></tr>
<tr><td>表 1.1　SH-4A の特長
CPU</td><td>1-1</td><td>修正。
• RISC タイプ命令セット（SH-1、SH-2、SH-3、SH-4 と上位互換性有り）：</td></tr>
<tr><td>L メモリ</td><td>1-3</td><td>追加。
• 2 本の独立した読み出し／書き込みポート
CPU からの 8／16／32／64 ビットアクセス
外部要求による 8／16／32／64 ビットおよび 16／32 バイトアクセス
【注】L メモリの容量については、製品のハードウェアマニュアルを参照してください。</td></tr>
<tr><td>表 1.2　SH-4 から SH-4A への変更点</td><td>1-3</td><td>修正。
<table>
<tr><th>章番号、章名</th><th>節番号</th><th>節名</th><th>変更点</th></tr>
<tr><td>3.　命令セット</td><td>3.3</td><td>命令セット</td><td>CPU 命令として 9 命令を追加</td></tr>
<tr><td></td><td></td><td></td><td>FPU 命令として 3 命令を追加</td></tr>
<tr><td>4.　パイプライン動作</td><td>4.2</td><td>並列実行性</td><td>CPU 命令して 9 命令を追加</td></tr>
<tr><td></td><td></td><td></td><td>FPU 命令として 3 命令を追加</td></tr>
</table></td></tr>
<tr><td></td><td>1-4</td><td>追加。
<table>
<tr><th>章番号、章名</th><th>節番号</th><th>節名</th><th>変更点</th></tr>
<tr><td>7.　メモリマネジメントユニット</td><td>7.6.4</td><td>UTLB データアレイ</td><td>メモリ割り付けアドレスを H'F700 0000～H'F77F FFFF から H'F700 0000～H'F70F FFFF へ変更</td></tr>
<tr><td></td><td>7.7</td><td>32 ビットアドレス拡張モード</td><td>新規追加</td></tr>
<tr><td>8.　キャッシュ</td><td>8.5.1</td><td>キャッシュと外部メモリとのコヒーレンシ</td><td>ICBI 命令、PREFI 命令、および SYNCO 命令追加</td></tr>
<tr><td></td><td>8.6</td><td>メモリ割り付け
キャッシュの構成</td><td>容量の変更および 4 ウェイセットアソシアティブ化に伴い、エントリビットとウェイビット変更</td></tr>
<tr><td></td><td>8.8</td><td>32 ビットアドレス拡張モード使用時の注意事項</td><td>新規追加</td></tr>
<tr><td>10.　各命令の説明</td><td>—</td><td>—</td><td>CPU 命令として 9 命令追加</td></tr>
<tr><td></td><td></td><td></td><td>FPU 命令として 3 命令追加</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>項　　目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>2.2.4　コントロールレジスタ
（1）ステータスレジスタ（SR）</td><td>2-9</td><td>修正。
<table>
<tr><th>ビット</th><th>ビット名</th><th>説　　明</th></tr>
<tr><td>15</td><td>FD</td><td>FPU ディスエーブルビット
このビットが 1 のとき、FPU 命令は一般 FPU 抑止例外を発生させ、FPU 命令が遅延スロットにある場合、スロット FPU 抑止例外が発生します（FPU 命令：H'F***命令、FPUL/FPSCR に対する LDS(.L)/STS(.L)命令）。</td></tr>
</table></td></tr>
<tr><td>2.2.5　システムレジスタ
（4）浮動小数点ステータス／コントロールレジスタ（FPSCR）</td><td>2-12</td><td>修正。
<table>
<tr><th>ビット</th><th>ビット名</th><th>初期値</th><th>R/W</th></tr>
<tr><td>1、0</td><td>RM</td><td>01</td><td>R/W</td></tr>
</table></td></tr>
<tr><td>図 2.5　SZ ビットとエンディアンの関係</td><td>2-13</td><td>追加。

</td></tr>
<tr><td>2.7　使用上の注意事項</td><td>2-17</td><td>追加。</td></tr>
<tr><td>図 4.2　命令実行パターン（7）</td><td>4-9</td><td>修正。
（6-3）FPUL への LDS.L；1 発行サイクル
（6-5）FPSCR への LDS；1 発行サイクル
（6-7）FPSCR への LDS.L；1 発行サイクル</td></tr>
<tr><td>表 4.2　命令グループ</td><td>4-12、
4-13</td><td>修正。
<table>
<tr><th>命令
グループ</th><th colspan="4">命令</th></tr>
<tr><td>LS</td><td>FABS
FNEG
FLDI0
FLDI1
FLDS
FMOV @adr,FR
FMOV FR,@adr
FMOV FR,FR
FMOV.S @adr,FR</td><td>FMOV.S FR,@adr
FSTS
LDC Rm,CR1
LDC.L @Rm+,CR1
LDS Rm,SR1
LDS Rm,SR2
LDS.L @adr,SR2
LDS.L @Rm+,SR1
LDS.L @Rm+,SR2</td><td>MOV.[BWL] @adr,R
MOV.[BWL] R,@adr
MOVA
MOVCA.L
MOVUA
OCBI
OCBP
OCBWB
PREF</td><td>STC CR2,Rn
STC.L CR2,@-Rn
STS SR2,Rn
STS.L SR2,@-Rn
STS SR1,Rn
STS.L SR1,@-Rn</td></tr>
<tr><td>CO</td><td>AND.B
#imm,@(R0,GBR)
ICBI
LDC Rm,DBR
LDC Rm,SGR
LDC Rm,SR
LDC.L @Rm+,DBR
LDC.L @Rm+,SGR</td><td>LDC.L @Rm+,SR
LDTLB
~~LDS Rm,FPSCR~~
~~LDS.L @Rm+,FPSCR~~
MAC.L
MAC.W
MOVCO</td><td>MOVLI
OR.B
#imm,@(R0,GBR)
PREFI
RTE
SLEEP
STC SR,Rn
STC.L SR,@-Rn</td><td>SYNCO
TAS.B
TRAPA
TST.B
#imm,@(R0,GBR)
XOR.B
#imm,@(R0,GBR)</td></tr>
</table></td></tr>
</table>

項　　目	ページ	修正箇所
5.6.1（3）H-UDI リセット	5-9	修正。 • 要因：SDIR. TI[7:4]が B'0110（ネゲート）、または B'0111（アサート）
5.7（4）RTE 命令の遅延スロット	5-23	削除。 1. RTE 命令の遅延スロットに配置された命令は、SSR に退避されていた値が SR に復帰されたのち実行されます。命令アクセスに関する例外の受け付け判定は復帰前の SR の値に応じて決定され、その他の例外の受け付け判定は復帰後との SR による処理モードや BL ビットに依存して決定されます。完了型の例外に関しては RTE の分岐先の実行前に受け付けられますが、再実行型の例外が発生すると動作が保証されません。 ~~RTE 命令の遅延スロットに配置された命令は、SSR に退避されていた値が SR に復帰されたのち実行されます。命令アクセスに関する例外の受け付け判定は復帰前の SR の値に応じて決定され、その他の例外の受け付け判定は復帰後との SR による処理モードや BL ビットに依存して決定されます。完了型の例外に関しては RTE の分岐先の実行前に受け付けられますが、再実行型の例外が発生すると動作は保証されません。~~
図 6.5　SZ ビットとエンディアンの関係	6-9	修正。 ・リトルエンディアンの場合 
（3）FPU 例外処理	6-12	• 0 による除算（Z） ：FPSCR.EN.Z＝1 かつ除数 0 による除算または FSRRA の入力が 0 の場合
図 7.4　P4 領域	7-5	修正。 

<table>
<tr><th>項　　目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>7.1.1　アドレス空間
（d）P4 領域</td><td>7-6</td><td>追加。
H'F610 0000～H'F61F FFFF までは、PMB のアドレスアレイを直接アクセスするための領域です。詳細は、「7.7.5　メモリ割り付け PMB の構成」を参照してください。
H'F700 0000～H'F70F FFFF までは、共用 TLB のデータアレイを直接アクセスするための領域です。詳細は、「7.6.4　UTLB データアレイ」を参照してください。
H'F710 0000～H'F71F FFFF までは、PMB のデータアレイを直接アクセスするための領域です。詳細は、「7.7.5　メモリ割り付け PMB の構成」を参照してください。</td></tr>
<tr><td>7.2.6　物理アドレス空間制御レジスタ（PASCR）</td><td>7-14</td><td>修正。
<table>
<tr><th>ビット</th><th>ビット名</th><th>説　　明</th></tr>
<tr><td>7～0</td><td>UB</td><td>エリア（64M バイト）ごとのバッファドライト制御
キャッシュを使わない書き込みのバスアクセスが完了するまで次の CPU からのバスアクセスを待たせるかをエリアごとに指定します。
0：CPU は書き込みのバスアクセスの完了を待たずに次のバスアクセスを行います
1：CPU は書き込みのバスアクセスの完了を待ってから次のバスアクセスを行います</td></tr>
</table></td></tr>
<tr><td>7.2.7　命令再フェッチ抑止制御レジスタ（IRMCR）</td><td>7-15</td><td>修正。
<table>
<tr><th>ビット</th><th>ビット名</th><th>初期値</th><th>R/W</th></tr>
<tr><td>31～5</td><td>－</td><td>すべて 0</td><td>R</td></tr>
<tr><td>4</td><td>R2</td><td>0</td><td>R/W</td></tr>
<tr><td>3</td><td>R1</td><td>0</td><td>R/W</td></tr>
<tr><td>2</td><td>LT</td><td>0</td><td>R/W</td></tr>
</table></td></tr>
<tr><td>7.7　32 ビットアドレス拡張モード</td><td>7-35～7-41</td><td>追加。</td></tr>
<tr><td>8.　キャッシュ</td><td>8-1</td><td>【注】追加。</td></tr>
<tr><td>8.5.1　キャッシュと外部メモリとのコヒーレンシ</td><td>8-15</td><td>削除。
• 1K バイトのページサイズを使用しないでください。
• ~~オペランドキャッシュの容量が 64KB の場合、4KB のページサイズでは仮想アドレス[13]と物理アドレス[13]を一致させてください。~~
• ~~オペランドキャッシュの容量が 128KB の場合、4KB のページサイズでは仮想アドレス[14:13]と物理アドレス[14:13]を一致させてください。~~</td></tr>
<tr><td>7.6.1　IC アドレスアレイ</td><td>8-17</td><td>【注】追加。</td></tr>
<tr><td>7.6.3　OC アドレスアレイ</td><td>8-19</td><td>【注】追加。</td></tr>
<tr><td>8.7.3　外部メモリへの転送
（1）MMU イネーブル（MMUCR.AT＝1）の場合</td><td>8-22</td><td>削除。
ASID、V、SZ、SH、PR、D ビットは通常のアドレス変換と同様の意味を持ちますが、C、WT ビットはこのページに関しては意味を持ちません。
~~SQ を用いて PCMCIA インタフェースのエリアへのデータ転送はできません。~~</td></tr>
</table>

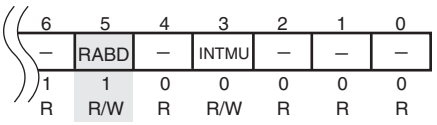
<table>
<tr><th>項　　目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>8.8　32 ビットアドレス拡張モード使用時の注意事項</td><td>8-24</td><td>追加。</td></tr>
<tr><td>9.　L メモリ</td><td>9-1</td><td>【注】追加。</td></tr>
<tr><td>9.6　32 ビットアドレス拡張モード使用時の注意事項</td><td>9-10</td><td>追加。</td></tr>
<tr><td>10.1　CPU 命令</td><td>10-2</td><td>修正。
float　　single precision floating point number(32 bits)</td></tr>
<tr><td>10.1.3　ADDV</td><td>10-6</td><td>修正。
ADD with (Vflag) overflow check</td></tr>
<tr><td>10.1.4　AND
（5）発生する可能性がある例外</td><td>10-9</td><td>追加。
例外処理において、本命令によるデータアクセスは、バイトロード、バイトストアとして扱われます。</td></tr>
<tr><td>10.1.29　MAC.W
（3）動作内容</td><td>10-55、
10-56</td><td>修正。
<pre>MACW(long m, long n)   /* MAC.W @Rm+,@Rn+ */
{
    long tempm,tempn,dest,src,ans;
    unsigned long templ;
    tempn = (long)Read_Word(R[n]);
    R[n] += 2;
    tempm = (long)Read_Word(R[m]);
    R[m] += 2;
    templ=MACL;
    tempm = ((long)(short)tempn*(long)(short)tempm);
    if((long)MACL>=0) dest = 0;
    else dest = 1;
    if((long)tempm>=0) {
        src=0;
        tempn = 0;
    }
    else {
        src=1;
        tempn = 0xFFFFFFFF;
    }
    src += dest;
    MACL += tempm;
    if((long)MACL>=0) ans = 0;
    else ans = 1;</pre></td></tr>
<tr><td>10.1.32　MOV
（3）動作内容</td><td>10-66</td><td>修正。
<pre>MOVBLG(int d) /* MOV.B @(disp,GBR),R0 */</pre></td></tr>
<tr><td>10.1.33　MOV
（3）動作内容</td><td>10-70</td><td>修正。
<pre>MOVBS4(long d), long n   /* MOV.B R0,@(disp,Rn) */</pre></td></tr>
<tr><td>10.1.36　MOVCO</td><td>10-74</td><td>修正。
<table>
<tr><th>書式</th><th>動作概略</th></tr>
<tr><td>MOVCO.L　　R0,@Rn</td><td>LDST→T
if(T==1) R0→(Rn)
0→LDST</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>項　　目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>10.1.50　OR
（5）発生する可能性のある例外</td><td>10-91</td><td>追加。
例外処理において、本命令によるデータアクセスはバイトロード、バイトストアとして扱われます。</td></tr>
<tr><td>10.1.51　PREF
（1）説明</td><td>10-92</td><td>修正。
この命令でデータアドレスエラーは発生しません。またデータ TLB 多重ヒット例外を除く MMU 例外も発生しません。これらの例外条件に合致した場合には、NOP（無操作）命令として取り扱われます。</td></tr>
<tr><td>（3）動作内容</td><td>10-92</td><td>修正。
PREF(int n) /* PREF @Rn */</td></tr>
<tr><td>（5）発生する可能性のある例外</td><td>10-92</td><td>追加。
• データ TLB 多重ヒット例外</td></tr>
<tr><td>10.1.52　PREFI
（1）説明</td><td>10-93</td><td>修正。
この命令でデータアドレスエラーおよび MMU 例外は発生しません。これらの例外条件に合致した場合には、NOP（無操作）命令として取り扱われます。</td></tr>
<tr><td>（3）動作内容</td><td>10-93</td><td>修正。
PREFI(int n) /* PREFI @Rn */</td></tr>
<tr><td>10.1.70　STC
（3）動作内容</td><td>10-118</td><td>修正。
<pre>STCGBR(int n) /* STC GBR,Rn */
{
        R[n] = GBR;</pre></td></tr>
<tr><td>10.1.76　SYNCO</td><td>10-129</td><td>修正。
<table><tr><th>書式</th><th>動作概略</th></tr><tr><td>SYNCO</td><td>本命令に先行するバスアクセスの完了まで、本命令以降の命令によるデータアクセスを開始しません。</td></tr></table></td></tr>
<tr><td>（1）説明</td><td>10-129</td><td>修正。
本命令はデータ操作の同期に使用します。本命令を実行すると、本命令に先行するバスアクセスの完了を待ってから、本命令より後の命令によるデータアクセスを開始します。</td></tr>
<tr><td>（2）注意</td><td>10-129</td><td>修正。
バスアクセスによるデータ更新結果がバス以外により CPU に通知されるような場合、すなわちアドレスマップされたハードウェアレジスタの反映などについては、SYNCO 命令の挿入だけでは順序付けが保証されないことがあります。この場合、順序保証のための条件は各レジスタの項目を個別に参照してください。</td></tr>
</table>

<table>
<tr><th>項　　目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>（4）使用例</td><td>10-129</td><td>削除。
1. 他メモリユーザと共有したメモリへのアクセスの順序付け
<s>2. アドレスマップされたハードウェアレジスタへのアクセスの順序付け</s>
2. すべてのライトバッファのフラッシュ
3. メモリアクセスのマージ、消滅の防止
4. キャッシュ操作命令の完了待ち</td></tr>
<tr><td>10.1.77　TAS
（4）発生する可能性がある例外</td><td>10-131</td><td>修正。
例外処理において、本命令によるデータアクセスはバイトロード、バイトストアとして扱われます。</td></tr>
<tr><td>10.1.79　TST
（5）発生する可能性がある例外</td><td>10-135</td><td>追加。
例外処理において、本命令によるデータアクセスはバイトロード、バイトストアとして扱われます。</td></tr>
<tr><td>10.1.80　XOR
（5）発生する可能性がある例外</td><td>10-137</td><td>追加。
例外処理において、本命令によるデータアクセスはバイトロード、バイトストアとして扱われます。</td></tr>
<tr><td>10.3.6　FDIV</td><td>10-175</td><td>修正。
<table>
<tr><th>PR</th><th>書式</th><th>実行ステート</th><th>T ビット</th></tr>
<tr><td>0</td><td>FDIV　FRm,FRn</td><td>14</td><td>—</td></tr>
<tr><td>1</td><td>FDIV　DRm,DRn</td><td>30</td><td>—</td></tr>
</table>
</td></tr>
<tr><td>（3）動作内容</td><td>10-177</td><td>修正。
if((tmpf.f < tmpd.d) && (FPSCR_RM == 1)) dstf.l -= 1;
　　:
if((tmpd.d < tmpx.x) && (FPSCR_RM == 1)) {</td></tr>
<tr><td>10.3.7　FIPR</td><td>10-179</td><td>修正。
例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FR[n+3]は更新されません。</td></tr>
<tr><td>10.3.11　FLOAT</td><td>10-184</td><td>修正。
例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn は更新されません。</td></tr>
</table>

<table>
<tr><th>項　　目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td rowspan="2">10.3.12　FMAC</td><td>10-186</td><td>修正。
例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn は更新されません。</td></tr>
<tr><td>10-189</td><td>修正。
<table>
<tr><th colspan="2">FMAC</th><th colspan="3">FRm</th></tr>
<tr><th>FRn</th><th>FR0</th><th>+NORM</th><th>qNaN</th><th>sNaN</th></tr>
<tr><td rowspan="6">+inf</td><td>+NORM</td><td rowspan="5"></td><td rowspan="5"></td><td rowspan="5"></td></tr>
<tr><td>-NORM</td></tr>
<tr><td>+0</td></tr>
<tr><td>-0</td></tr>
<tr><td>+inf</td><td>+inf</td></tr>
<tr><td>-inf</td><td>invalid</td><td colspan="2"></td></tr>
</table></td></tr>
<tr><td>10.3.14　FMOV
（3）動作内容</td><td>10-196</td><td>修正。
<pre>
void FMOV_STORE_XD(int m,n)   /* FMOV XDm,@Rn */
{
     store_quad(XD[m>>1],R[n]);
     pc += 2;
}
void FMOV_LOAD_XD(int m,n)   /* FMOV @Rm,XDn */
{
               load_quad(R[m],XD[n>>1]);
               pc += 2;
}
void FMOV_RESTORE_XD(int m,n)   /* FMOV @Rm+,XDn */
{
               load_quad(R[m],XD[n>>1]);
               R[m] += 8;
               pc += 2;
}
</pre></td></tr>
<tr><td>10.3.15　FMUL
（4）発生する可能性がある例外とオーバフロー／アンダフロー例外トラップの発生条件</td><td>10-201</td><td>修正。
● アンダフロー
アンダフロー例外トラップの発生条件
・FPSCR.PR=0 で
FRn、FRm の双方が正規化数の場合：FRn の指数部＋FRm の指数部－H'7F が H'00 以下
FRn、FRm の少なくとも一方が非正規化数の場合：FRn の指数部＋FRm の指数部－H'7F が H'18 以下
・FPSCR.PR=1 の場合：DRn の指数部＋DRm の指数部－H'3FF が H'000 以下</td></tr>
<tr><td>10.3.17　FPCHG</td><td>10-203</td><td>修正。
<table>
<tr><th>書式</th><th>動作概略</th></tr>
<tr><td>FPCHG</td><td>~FPSCR.PR → FPSCR.PR</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>項　　目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>10.3.19　FSCA
（1）説明</td><td>10-205</td><td>修正。
FPUL の示す角度の sin および cos の近似値（絶対誤差±2 ^ -21 未満）を単精度浮動小数点として求め、sin 値を FRn、cos 値を FR[n+1]に書き込みます。近似演算命令のため、常に不正確例外を要求します（入力が 0 の場合でも不正確とします）。
FPSCR.enable.I がセットされている場合、FPU 例外トラップが発生します。例外発生時は、FPSCR.cause、FPSCR.flag には、例外の正しい情報が反映され、FRn/FR[n+1]は更新されません。ソフトウェアで適切な処理を行ってください。</td></tr>
<tr><td>10.3.21　FSQRT</td><td>10-208</td><td>修正。
<table>
<tr><th>PR</th><th>書式</th><th>命令コード</th><th>実行
ステート</th></tr>
<tr><td>0</td><td>FSQRT　FRn</td><td>1111nnnn01101101</td><td>14</td></tr>
<tr><td>1</td><td>FSQRT　DRn</td><td>1111nnn001101101</td><td>30</td></tr>
</table>
</td></tr>
<tr><td>（3）動作内容</td><td>10-209</td><td>修正。
if((tmpf.f < tmpd.d) && (FPSCR_RM == 1))
:
if((tmpd.d < tmpx.x) && (FPSCR_RM == 1)) {</td></tr>
<tr><td>10.3.22　FSRRA</td><td>10-211</td><td>追加。
<table>
<tr><th>PR</th><th>書式</th><th>動作概略</th></tr>
<tr><td>0</td><td>FSRRA　FRn</td><td>1/sqrt(FRn)*→FRn</td></tr>
<tr><td>1</td><td>—</td><td>reserved</td></tr>
</table>
</td></tr>
<tr><td>（1）説明</td><td>10-211</td><td>修正。
FRn の内容である単精度浮動小数点の算術的な平方根の逆数の近似値（絶対誤差±2 ^ -21 未満）を FRn に書き込みます。</td></tr>
<tr><td>10.3.26　FTRV</td><td>10-221</td><td>修正。
例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FVn は更新されません。</td></tr>
<tr><td>11.　レジスター覧</td><td>11-1</td><td>削除。
1. レジスタアドレス一覧（機能モジュールごと、マニュアル章番号順）
• 機能モジュールごとに、マニュアルの章番号の順に表記しています。
• 本リストに記載されていないリザーブアドレスは、アクセスしないでください。
~~• アドレスは、16 ビットまたは 32 ビットの場合、ビッグエンディアンを前提として MSB 側のアドレスを表記しています。~~</td></tr>
</table>

<table>
<tr><th>項　　目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>付録 A.　CPU 動作モードレジスタ（CPUOPM）</td><td>付録-1</td><td>追加。
CPUOPM は、CPU の動作モードを切り替えるために使用します。本レジスタは P4 領域の H'FF2F0000 あるいはエリア 7 アドレスの H'1F2F0000 から 32 ビットサイズで読み出し／書き込みが可能です。本レジスタへ書き込む際には、必ずリザーブビットに初期値を書き込むようにしてください。リザーブビットに初期値以外の値を書き込んだ場合の動作は保証されません。
CPUOPM の更新は、CPU 以外の SuperHyway バスマスタからのアクセスでなく、CPU のストア命令で行ってください。また、CPUOPM 更新後、一度 CPUOPM を読み出した後で、以下の 1.または 2.のどちらかを実行してください。
1.　RTE 命令による分岐を実行してください。
2.　任意のアドレス（キャッシング不可領域でもよい）に対して、ICBI 命令を実行してください。
1.または 2.の実行後、CPU は更新後の CPUOPM の値を用いて動作することが保証されます。</td></tr>
<tr><td>付録 A.　CPU 動作モードレジスタ（CPUOPM）</td><td>付録-1</td><td>追加。
 

<table>
<tr><th>ビット</th><th>ビット名</th><th>初期値</th><th>R/W</th><th>説　　明</th></tr>
<tr><td>31～6</td><td>—</td><td>H'000000F</td><td>R</td><td>リザーブビット
書き込み時は、必ず初期値を書き込むようにしてください。</td></tr>
<tr><td>5</td><td>RABD</td><td>1</td><td>R/W</td><td>サブルーチン復帰投機実行ビット
0：サブルーチンからの復帰時に命令フェッチを投機的に発行します。本ビットを 0 に設定する場合は、「付録 C.　サブルーチン復帰投機実行」を参照してください。
1：サブルーチンからの復帰時に命令フェッチを投機的に発行しません。</td></tr>
<tr><td>4</td><td>—</td><td>0</td><td>R</td><td>リザーブビット
書き込み時は、必ず初期値を書き込むようにしてください。</td></tr>
<tr><td>3</td><td>INTMU</td><td>0</td><td>R/W</td><td>割り込み動作モード切り替えビット
0:割り込みを受理しても SR.IMASK の値は変化しません。
1：割り込みを受理した場合、受け付けたレベルを SR.IMASK の値に自動的に設定します。</td></tr>
</table></td></tr>
<tr><td>付録 C.　サブルーチン復帰投機実行</td><td>付録-3</td><td>追加。</td></tr>
<tr><td>付録 D.　バージョンレジスタ（PVR、PRR）</td><td>付録-4</td><td>追加。</td></tr>
</table>

索引

0 による除算 6-11
32 ビットアドレス拡張 7-35
FPU エラー 6-11
FPU に関するシステムレジスタ 2-3
FPU 例外 5-20, 6-11
FPU 例外処理 6-12
H-UDI リセット 5-9
NMI（ノンマスカブル割り込み） 5-20
T ビット 3-2
X/Y メモリ 9-1
アドレッシングモード 3-3
アンダフロー 6-11
一般 FPU 抑止／スロット FPU 抑止例外 6-11
一般 FPU 抑止例外 5-18
一般不当命令例外 5-16
一般割り込み要求 5-21
オーバフロー 6-11
キャッシュ 8-1
共用 TLB 7-17
固定小数点転送命令 3-9
コントロールレジスタ 2-2
算術演算命令 3-10
ジオメトリック演算命令 6-13
実行ステート 4-15
システムレジスタ 2-3
システム制御命令 3-15
実効アドレス 3-3
シフト命令 3-13
初期ページ書き込み例外 5-11
処理モード 2-2
スロット FPU 抑止例外 5-18
スロット不当命令例外 5-17
単精度浮動小数点レジスタ 2-6
単精度浮動小数点拡張レジスタ 2-7
単精度浮動小数点拡張レジスタ行列 2-7
単精度浮動小数点ベクトルレジスタ 2-6
遅延スロット 3-1
遅延分岐 3-1
低消費電力状態 2-16
データ TLB ミス例外 5-10
データ TLB 多重ヒット例外 5-10
データ TLB 保護違反例外 5-12
データアドレスエラー 5-14
特権空間マッピングバッファ（PMB）構成 7-36
特権モード 2-2
倍精度浮動小数点レジスタ 2-6
パイプライン動作 4-1
発行レート 4-15
パワーオンリセット 5-9
汎用レジスタ 2-2
ビッグエンディアン 2-15
符号拡張 2-14
不正確例外 6-11
浮動小数点レジスタ 2-3, 2-6
浮動小数点制御命令 3-18
浮動小数点単精度命令 3-17
浮動小数点倍精度命令 3-18
プログラミングモデル 2-1
分岐命令 3-14
ペア単精度データ転送命令 6-14
マニュアルリセット 5-9
丸め 6-10
無効演算 6-11
無条件トラップ 5-15
命令 TLB 7-19
命令 TLB 多重ヒット例外 5-9
命令 TLB 保護違反例外 5-13
命令 TLB ミス例外 5-11
命令アドレスエラー 5-15
命令実行後ユーザブレーク 5-19
命令実行状態 2-16
命令実行前ユーザブレーク 5-19
メモリマネジメントユニット 7-1
メモリ割り付け PMB の構成 7-38
ユーザモード 2-2
リセット状態 2-16

リトルエンディアン 2-15
例外処理 5-1
レジスタ
CCR 8-5
CPUOPM 付録-1
DBR 2-11
EXPEVT 5-2
FPSCR 2-11, 6-8
FPUL 2-13, 6-10
GBR 2-10
INTEVT 5-3
IRMCR 7-15
LDA 0 9-6
LDA1 9-7
LSA0 9-4
LSA1 9-5
MACH 2-11
MACL 2-11
MMUCR 7-11
PASCR 7-14
PC 2-11
PR 2-11
PRR 付録-4
PTEH 7-9
PTEL 7-10
PVR 付録-4
QACR0 8-6
QACR1 8-7
RAMCR 8-7, 9-3
SGR 2-10
SPC 2-10
SR 2-9
SSR 2-10
TEA 7-11
TRA 5-2
TTB 7-11
VBR 2-10
レジスタの状態 11-1
ロード／ストアアーキテクチャ 3-1
論理演算命令 3-12
浮動小数点グラフィック強化命令 3-19

ルネサス32ビットRISCマイクロコンピュータ
ソフトウェアマニュアル
SH-4A

発行年月日　2003年9月9日　Rev.1.00
　　2004年10月29日　Rev.1.50
発　　行　株式会社ルネサス テクノロジ 営業企画統括部
　　〒100-0004 東京都千代田区大手町 2-6-2
編　　集　株式会社ルネサス小平セミコン 技術ドキュメント部

株式会社ルネサス テクノロジ 営業企画統括部　〒100-0004 東京都千代田区大手町2-6-2 日本ビル

営業お問合せ窓口
株式会社ルネサス販売

http://www.renesas.com

本社	〒100-0004	千代田区大手町2-6-2 (日本ビル)	(03) 5201-5350
京浜支社	〒212-0058	川崎市幸区鹿島田890-12 (新川崎三井ビル)	(044) 549-1662
西東京支社	〒190-0023	立川市柴崎町2-2-23 (第二高島ビル2F)	(042) 524-8701
札幌支店	〒060-0002	札幌市中央区北二条西4-1 (札幌三井ビル5F)	(011) 210-8717
東北支社	〒980-0013	仙台市青葉区花京院1-1-20 (花京院スクエア13F)	(022) 221-1351
いわき支店	〒970-8026	いわき市平小太郎町4-9 (損保ジャパンいわき第二ビル3F)	(0246) 22-3222
茨城支店	〒312-0034	ひたちなか市堀口832-2 (日立システムプラザ勝田1F)	(029) 271-9411
新潟支店	〒950-0087	新潟市東大通1-4-2 (新潟三井物産ビル3F)	(025) 241-4361
松本支社	〒390-0815	松本市深志1-2-11 (昭和ビル7F)	(0263) 33-6622
中部営業本部	〒460-0008	名古屋市中区栄3-13-20 (栄センタービル4F)	(052) 261-3000
浜松支店	〒430-7710	浜松市板屋町111-2（浜松アクトタワー10F）	(053) 451-2131
西部営業本部	〒541-0044	大阪市中央区伏見町4-1-1 (明治安田生命大阪御堂筋ビル)	(06) 6233-9500
北陸支社	〒920-0031	金沢市広岡3-1-1 (金沢パークビル8F)	(076) 233-5980
広島支店	〒730-0036	広島市中区袋町5-25 (広島袋町ビルディング8F)	(082) 244-2570
鳥取支店	〒680-0822	鳥取市今町2-251 (日本生命鳥取駅前ビル)	(0857) 21-1915
九州支社	〒812-0011	福岡市博多区博多駅前2-17-1 (ヒロカネビル本館5F)	(092) 481-7695
鹿児島支店	〒890-0053	鹿児島市中央町12-2 (明治安田生命鹿児島中央町ビル)	(099) 284-1748

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：カスタマサポートセンタ　E-Mail: csc@renesas.com

Colophon 3.0

SH-4A
ソフトウェアマニュアル

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668

RJJ09B0090-0150Z