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

RENESAS

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。

SH4AL-DSP

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

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

SH4AL-DSP 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、SH4AL-DSP 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、SH4AL-DSP の命令を理解していただくことを目的としています。ハードウェアの各機能については、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**「3. 命令セット」**に説明されています。各命令の詳細動作は、**「11. 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
DSP	Digital Signal Processor デジタルシグナルプロセッサ
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

図目次

付録

表目次

4. パイプライン動作

5. 例外処理

6. DSP ユニット

1. 概要

1.1 SH-4AL の特長

SH-4AL は、SH-4A から FPU 機能を削除した 32 ビット RISC（縮小命令セットコンピュータ）マイコンであり、SH-1、SH-2、SH-3 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します。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サイズをほぼ 50%縮小することができます。

なお SH-4AL では従来の SH-3 との互換性を鑑み SH-4A の L メモリを X/Y メモリとして表現し、大容量の内蔵 RAM を U メモリと表します。

SH-4AL の特長を**表** 1.1 に示します。

表 1.1　SH-4AL 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 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 • 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
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） 【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。
X/Y 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 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 • CPU アクセスでの記憶保護機構に加え、DSP アクセス専用の記憶保護機構をサポート 【注】X/Y 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。
U 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 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】U 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 SH4AL-DSP の特長

SH4AL-DSP は、SH-4AL に対して DSP ユニットをコプロセッサとして付加したものです。

SH4AL-DSP は基本的には SH-4AL CPU 部と同じ 16 ビット長の命令を持ち、DSP タイプの命令を並行処理するために、32 ビット長の DSP タイプの命令が追加されています。SH-4AL CPU は標準のノイマン型アーキテクチャですが、SH4AL-DSP は拡張ハーバード型アーキテクチャの DSP データバスを持っています。

SH4AL-DSP にて追加された DSP 機能の特長を表 1.2 に示します。

表 1.2 SH4AL-DSP の追加された特長

項目	特　　長
DSP ユニット	• 16 ビット命令、および 32 ビット命令の混在可能 • 32 または 40 ビットの内部データバスを内蔵 • 乗算器、ALU、バレルシフタに対応 • 16 ビット×16 ビットに対応する 32 ビット乗算器を内蔵 • 大容量の DSP データレジスタファイルをサポート 6 本の 32 ビットデータレジスタ 2 本の 40 ビットデータレジスタ • DSP データバス用の拡張ハーバードアーキテクチャをサポート 2 本のデータバス 1 本の命令バス • 最大 4 つの並列演算を実行可能 ALU、乗算、2 つのロード／ストア • 2 つのメモリアクセス用アドレスを生成するための 2 本のアドレスユニットを装備 • DSP データアドレッシングモードをサポート インクリメント、およびインデクシング（モジュロアドレッシングあり／なし） • ゼロオーバヘッドリピートループ制御に対応 • 条件付実行命令に対応 • ユーザ DSP モードおよび特権 DSP モードをサポート
X/Y メモリ （追加項目）	• 3 本の独立した読み出し／書き込みポート CPU からの 8／16／32 ビットアクセス DSP からの最大 2 つの 16 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

1.3 SH-4A から SH4AL-DSP への変更点

SH-4A から SH4AL-DSP への変更点を本マニュアルの章、節単位に示します。

表 1.3 SH-4A から SH4AL-DSP への変更点

章番号、章名	節番号	節名	変更点
全体構成	—	—	FPU の代わりに DSP を内蔵しています。
	—	—	L メモリを X/Y メモリと呼びます。
	—	—	U メモリを内蔵しています。
2. プログラミングモデル	2.1	データフォーマット	単精度／倍精度の浮動小数点フォーマットはサポートしません。
	2.2	レジスタの構成	処理モードにユーザ DSP モードと特権 DSP モードが追加となり、FPU レジスタの代わりに DSP 関連のレジスタが追加となります。
	2.2.4（1）	ステータスレジスタ（SR）	RC、DSP、DMY、DMX、RF ビットが追加となり、FD ビットがなくなります。
3. 命令セット	3.2	アドレッシングモード	クワッドワードサイズのアドレッシングがなくなり、DSP 関連データのアドレッシングが追加となります。
	3.3	命令セット	浮動小数点関連命令を削除し、DSP をサポートする CPU 命令を追加します。
	3.4～3.5		DSP データ転送命令と DSP 演算命令の命令セットを追加します。
4. パイプライン動作	—	—	浮動小数点パイプラインを削除し、DSP パイプラインを追加します。
5. 例外処理	—	—	一般 FPU 抑止例外とスロット FPU 抑止例外と FPU 例外がなくなります。
	5.6.2（6）	データアドレスエラー	データアドレスエラーの条件からクワッドワードアクセスを削除し、X/Y メモリの領域に対する条件を追加します。
	5.6.2（9）	一般不当命令例外	SR.DSP ビットが 0 のときの DSP 関連命令の実行を条件に追加します。
	5.6.2（10）	スロット不当命令例外	SR.DSP ビットが 0 のときの DSP 関連命令の実行を条件に追加します。
	5.6.3	互換リピート制御中の例外処理	互換リピート制御中の例外処理の扱いが追加となります。
6. DSP	—	—	FPU の代わりに DSP を内蔵しています。リピート機能も追加となります。
7. MMU	7.1.1	アドレス空間	ストアキュー領域、内蔵メモリ領域の代わりに U メモリと X/Y メモリに対する Uxy 領域をサポートします。
	7.2	レジスタの構成	PASCR レジスタの初期値が H'00000082 となります。
	7.2.5	MMUCR	MMUCR の 9 ビット目は SQMD ビットからリザーブビットになります。

章番号、章名	節番号	節名	変更点
8. キャッシュ	—	—	ストアキューはサポートしません。
	8.2	レジスタの説明	ストアキューに関連する QACR0,QACR1 レジスタはサポートしません。
	8.3.4	ライトスルーバッファ	ライトスルーバッファが 64 ビットから 32 ビットに変更となります。
9. X/Y メモリ	—	—	SH-4A の L メモリの名称を変更し、DSP データ転送用に専用バスでアクセスすることができる X/Y メモリを追加します。
10. U メモリ	—	—	命令／データを格納することが可能な大容量の内蔵メモリ（U メモリ）をサポートします。
11. 各命令の説明	11.2	CPU 命令（DSP 関係）	FPU 関係の CPU 命令の代わりに、DSP 関係の CPU 命令を追加します。
	11.3	DSP データ転送命令	FPU 命令の代わりに追加します。
	11.4	DSP 演算命令	FPU 命令の代わりに追加します。

1.4 SH3-DSP から SH4AL-DSP への変更点

SH3-DSP から SH4AL-DSP への変更点を、本マニュアルの章、節単位に示します。

表 1.4 SH3-DSP から SH4AL-DSP への変更点

章番号、章名	節番号	節名	変更点
1. 概要	ー	ー	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	レジスタの説明	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）を追加
			DSP ステータスレジスタ（DSR）に制御ビットを 8 ビット追加
	2.2.4 （1）	ステータスレジスタ（SR）	RF ビットの仕様変更
	2.2.4 （8）	繰り返し開始レジスタ（RS）、 繰り返し終了レジスタ（RE）	仕様変更
3. 命令セット	3.1(4)	遅延スロット	成立しない BF/S、BT/S の次の命令もディレイスロットと定義
	3.3	命令セット	CPU 命令として 24 命令を追加
			DSP 命令として 42 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 命令として 24 命令を追加
			DSP 命令として 42 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	5.2	レジスタの説明	割り込み事象レジスタ 2（INTEVT2）を廃止
	5.4	例外の種類と優先順位	TLB 無効例外は TLB ミス例外として扱う
			DMA アドレスエラー例外は割り込みとして定義し、一般例外からは削除する
			ディレイスロット付き条件分岐命令のディレイスロットで発生した例外については、分岐命令の成立・非成立にかかわらずディレイスロットとして扱う
			不当命令例外、スロット不当命令例外条件変更

章番号、章名	節番号	節名	変更点
6. DSP ユニット	6.2	DSP モードのリソース	処理モード、ステータスレジスタ（SR）の拡張、およびステータスレジスタ（SR）のリピートフラグビット（RF）と繰り返し開始レジスタ（RS）と繰り返し終了レジスタ（RE）の仕様変更
	6.3	CPU 拡張命令	拡張リピートに関する CPU 命令の使用法の説明追加
	6.4	DSP データ転送命令	DSP 命令の MOVX,MOVY 拡張命令の説明を追加
			モジュロアドレッシングで追加 CPU 命令使用法の説明追加
	6.5	DSP データ演算命令	DSR レジスタの追加 8 ビット、フラグアキュムレート機能の説明追加
			T ビットリンク機能の説明追加
			PSWAP 命令の追加
7. メモリマネジメントユニット	－	－	新規追加
8. キャッシュ	－	－	新規追加
9. X/Y メモリ	－	－	新規追加
10. U メモリ	－	－	新規追加
11. 各命令の説明	－	－	CPU 命令として 24 命令を追加
			DSP 命令として 42 命令を追加
			DSP 命令の動作内容の C 記述の関数を SH2-DSP と合わせ、SH3-DSP での関数から記載変更

2. プログラミングモデル

本章では、SH4AL-DSP のプログラミングモデルについて記述します。SH4AL-DSP は以下に示すレジスタとデータフォーマットを持っています。

2.1 データフォーマット

SH4AL-DSP 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。

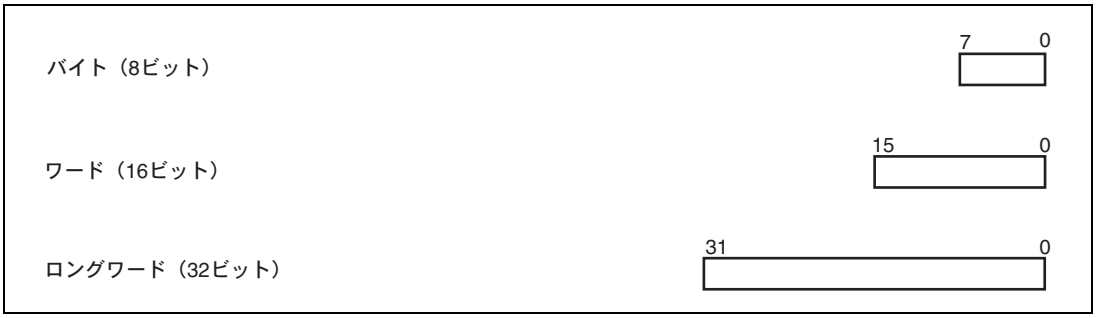


図 2.1 データフォーマット

2.2 レジスタの構成

2.2.1 特権モードとレジスタバンク

(1) 処理モード

CPU の処理モードは、ステータスレジスタ（SR）のモードビット（MD）および DSP ビット（DSP）により、次の表のように指定されます。

表 2.1 CPU 処理モード表

MD	DSP	処理モード	説明	
			特権保護されたリソースのアクセスや特権命令の実行	DSP 拡張機能
0	0	ユーザモード	不可	無効
0	1	ユーザ DSP モード	不可	有効
1	0	特権モード	可能	無効
1	1	特権 DSP モード	可能	有効

このように、DSP ビットによる DSP 拡張機能の制御は、MD ビットによる制御と独立に作用します。ただし、DSP ビットは特権モードでのみ値の変更が可能であり、DSP モードの変更を行うには特権モードや特権 DSP モードへの遷移が必要になります。

(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。

  SH4AL-DSPでDSP拡張機能が有効なとき、DSPタイプの命令では、16の汎用レジスタのうち8つのレジスタがX、Yデータメモリおよびオペランドバスを使うデータメモリ（シングルデータ）のアドレッシングに使われます。

  Xメモリをアクセスするためには、Xアドレスレジスタ（[Ax]）としてR4、R5を使い、Xインデックスレジスタ（[Ix]）としてR8を使います。Yメモリをアクセスするためには、Yアドレスレジスタ（[Ay]）としてR6、R7を使い、Yインデックスレジスタ（[Iy]）としてR9を使います。オペランドバスを使ってシングルデータをアクセスするためには、シングルデータアドレスレジスタ（[As]）としてR2、R3、R4、R5を使い、シングルデータインデックスレジスタ（[Is]）としてR8を使います。

  DSPタイプの命令はXとYデータメモリを同時にアクセスできます。XとYデータメモリのアドレスを指定するために、2組のアドレスポインタがあります。

(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（たとえば 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) DSP レジスタと DSP に関するシステムレジスタ

DSP レジスタとして 8 つのデータレジスタと 1 つのコントロールレジスタがあります。

DSP データレジスタは 2 本の 40 ビット長の A0、A1 レジスタと、6 本の 32 ビット長の M0、M1、X0、X1、Y0、Y1 レジスタがあります。A0、A1 レジスタにはそれぞれ 8 ビットのガードビット、A0G、A1G があります。

DSP に関するシステムレジスタには 32 ビット長の DSP ステータスレジスタ（DSR）、繰り返し開始レジスタ（RS）、繰り返し終了レジスタ（RE）、モジュロレジスタ（MOD）があります。

リセット後のレジスタの値を**表 2.2** に示します。

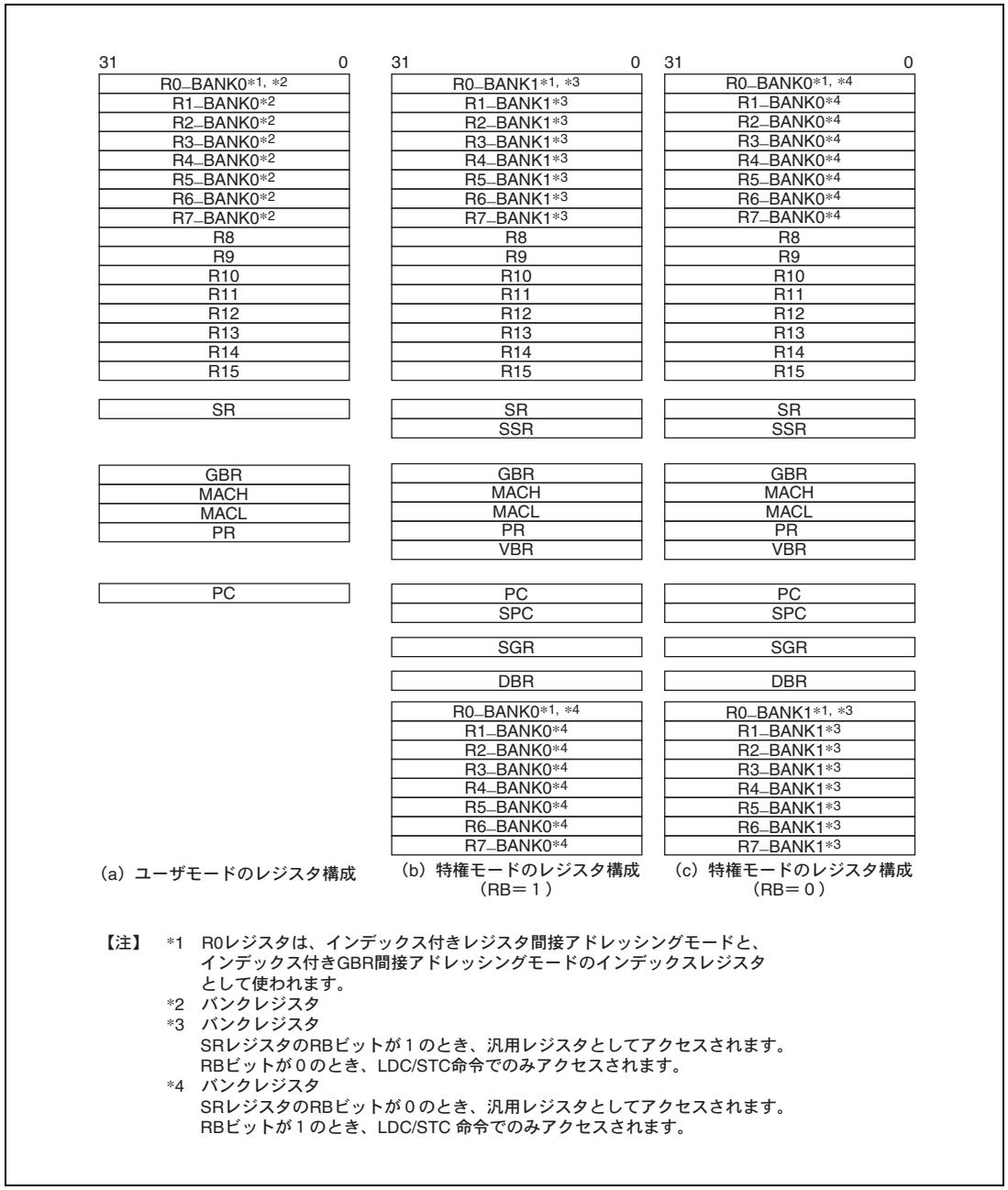
表 2.2 レジスタの初期値

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、IMASK は B'1111、その他はリザーブビットも含めて 0。
	GBR、SSR、SPC、SGR、DBR、RS、RE、MOD	不定
	VBR	H'00000000
システムレジスタ	MACH、MACL、PR	不定
	PC	H'A0000000
DSP レジスタ	A0、A0G、A1、A1G、M0、M1、X0、X1、Y0、Y1	不定
	DSR	H'0000

【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。SH4AL-DSP 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。SH4AL-DSP 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。

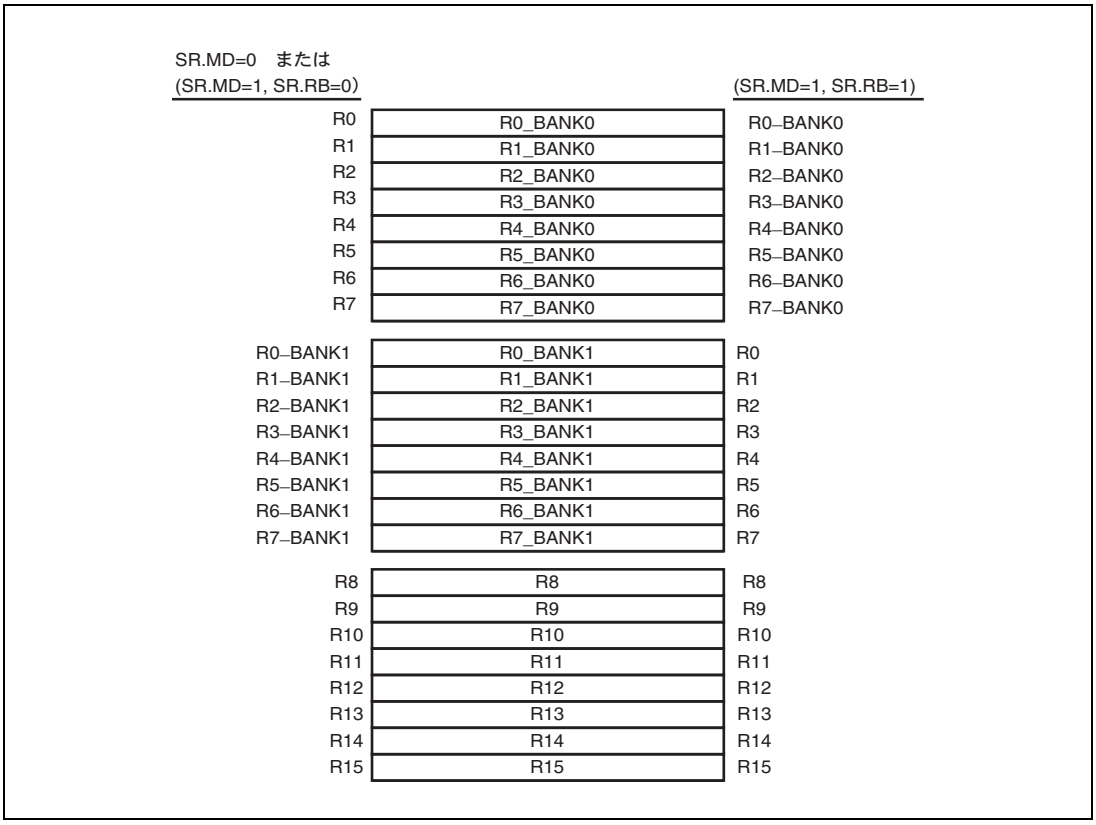


図 2.3 汎用レジスタ

【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での 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 DSP レジスタ

DSP データレジスタは 2 本の 40 ビット長の A0、A1 レジスタと、6 本の 32 ビット長の M0、M1、X0、X1、Y0、Y1 レジスタがあります。A0、A1 レジスタにはそれぞれ 8 ビットのガードビット A0G、A1G があります。

図 2.4 に DSP レジスタを示します。

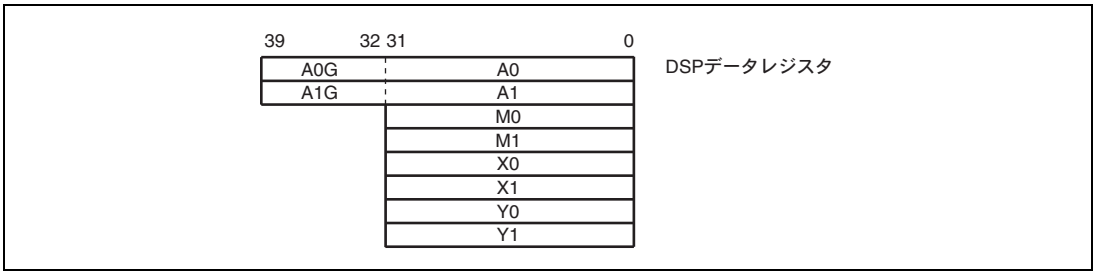


図 2.4 DSP レジスタの構成

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	RC											
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/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
	—	—	—	DSP	DMY	DMX	M	Q	IMASK				RF		S	T
初期値：	0	0	0	0	0	0	0	0	1	1	1	1	0	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	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、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 例外または割り込みにより 1 にセットされます。 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。
27～16	RC	すべて 0	R/W	リピートカウンタ 繰り返し（ループ）制御の繰り返し回数を指定します。（2～4096）
15～13	−	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。
12	DSP	0	R/W	DSP ビット 1 のとき DSP 命令が有効になります。
11	DMY	0	R/W	Y ポインタ用モジュロアドレッシング指定 1：Y メモリアドレスポインタ、Ay（R6、R7）に対し、モジュロアドレッシングモードが有効になります。
10	DMX	0	R/W	X ポインタ用モジュロアドレッシング指定 1：X メモリアドレスポインタ、Ax（R4、R5）に対し、モジュロアドレッシングモードが有効になります。
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。

ビット	ビット名	初期値	R/W	説　　明
7～4	IMASK	B'1111	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 の説明は「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	RF	B'00	R/W	リピートフラグビット リピートフラグビットは、リピート制御命令によって使用されます。これらのビットは、特権モード、特権 DSP モード、およびユーザ DSP モードで更新可能です。リセット状態に遷移することにより、0 に初期化されます。例外処理状態に遷移しても値は変化しません。 【注】　SH4AL-DSP では、SETRC 命令を用いたリピート制御（互換リピート制御）を拡張リピートの制御でエミュレーションしています。このため互換リピート制御中にリピートフラグビット（RF）の値が内部状態に応じて変化します。この仕様は従来の SH3-DSP シリーズと異なります。
1	S	0	R/W	S ビット MAC 命令および DSP 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、キャリー、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。

SR に対する LDC と STC 命令は、DSP ビットが 0 のときには、MD ビットが 1 の場合にのみ使用可能な命令ですが、DSP ビットが 1 のときには、ユーザ DSP モードにおいても使用可能になります。ただし、値を書き替えられる制御ビットは、RC、RF、DMX、および DMY に限定されます。LDC と STC 命令使用時のステータスレジスタ（SR）の詳細は、下記のとおりです。

- ユーザモード時は、SRに対するLDC命令とSTC命令は不当命令例外となります。
- 特権モードと特権DSPモードでは、SRの全ビットが更新できます。
- ユーザDSPモード時は、SRはSTC命令で読み出し可能です。

  ユーザDSPモード時は、SRへのLDC命令発行は可能ですが、DSP拡張ビットのみ更新できます。

表 2.3　各処理モードにおける SR の各ビットの動作説明

フィールド	特権モード	ユーザモード	特権 DSP モード	ユーザ DSP モード	専用命令による DSP 関連ビットへの アクセス
	MD=1 & DSP=0	MD=0 & DSP=0	MD=1 & DSP=1	MD=0 & DSP=1	
MD	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：NG	
RB	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：NG	
BL	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：NG	
RC	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：OK	SETRC、LDRC 命令
DSP	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：NG	
DMY	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：OK	SETDMY、CLRDMXY 命令
DMX	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：OK	SETDMX、CLRDMXY 命令
M	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：NG	
Q	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：NG	
IMASK	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：NG	
RF	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：OK	SETRC、LDRC 命令
S	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：NG	
T	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：NG	

【注】　M、Q、S、T ビットはユーザモードで専用命令によってセット／クリアが可能です。

【記号説明】

S　　：STC 命令

L　　：LDC 命令

OK　　：STC と LDC 動作を許可します。

不当命令：実行すると不当命令例外が発生します。

NG　　：前の値を保持します。変化しません。

(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）

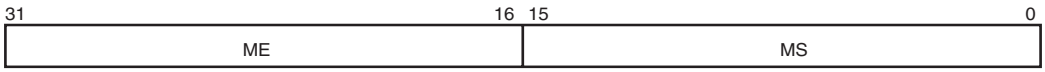
ユーザブレークデバッグ機能を有効にする場合（CBCR.UBDE＝1）、DBR は VBR の代わりにユーザブレークハンドラへの分岐先アドレスとして参照されます。

(8) 繰り返し開始レジスタ（RS）、繰り返し終了レジスタ（RE）

RS レジスタと RE レジスタはプログラムの繰り返し（ループ）を制御するために使います。SR レジスタの繰り返しカウンタ（RC：Repeat counter）に繰り返し回数を指定し、RS レジスタに繰り返し開始アドレスを指定し、RE レジスタに繰り返し終了アドレスを指定します。ただし、RS レジスタと RE レジスタに格納されるアドレスの値は、繰り返しの物理的な開始アドレス、終了アドレスとは値が必ずしも同じとは限りません。

また SH4AL-DSP では、SETRC 命令を用いたリピート制御（互換リピート制御）を LDRC 命令を用いた拡張リピート制御でエミュレーションしています。このため互換リピート制御中に RS レジスタと RE レジスタの値が内部状態に応じて変化します。この仕様は従来の SH3-DSP シリーズの互換リピート制御と異なっていますので、互換リピートを使用する際はリピート制御マクロ（REPEAT）を用いるか、SETRC 命令により 1 以上のリピート回数を設定する前には必ず LDRS および LDRE 命令を実行するようにしてください。

(9) モジュロレジスタ



- MS：モジュロ開始アドレス
- ME：モジュロ終了アドレス

MOD レジスタは繰り返しデータのバッファリングのためのモジュロアドレッシングに使います。SR レジスタの DMX または DMY でモジュロアドレッシングの指定をし、MOD レジスタの上位 16 ビットにモジュロ終了アドレス（ME）を指定し、下位 16 ビットにモジュロ開始アドレス（MS）を指定します。なお、DMX と DMY ビットは同時にモジュロアドレッシングを指定することはできません。モジュロアドレッシングは X、Y データ転送命令（MOVX、MOVY）のとき可能です。シングルデータ転送命令（MOVS）ではできません。

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) DSP ステータスレジスタ（DSR）

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
	AGT	AZ	AN	AV	TS			TC	GT	Z	N	V	CS			DC
初期値：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
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

表 2.4　DSR レジスタのビット

ビット	ビット名	初期値	R/W	機　　能
31～16	−	すべて 0	R	リザーブビット 常に 0 が読み出されます。書き込む値も 0 にしてください。
15	AGT	0	R/W	累積符合付き大ビット 1：演算結果が正（0 を除く）、またはオペランド 1 がオペランド 2 より大きいことを示します。 本ビットは演算結果により 0 クリアされることはありません。LDS 命令によって当該ビットへ 0 を書き込むまで 1 を保持します。また、DSR[11:8]（TS、TC）が 0000 のときは演算結果によって値が更新されることはありません。LDS 命令によって当該ビットへ書き込むまで値を保持します。
14	AZ	0	R/W	累積ゼロビット 1：演算結果が 0（ゼロ）、またはオペランド 1 がオペランド 2 と等しいことを示します。 本ビットは演算結果により 0 クリアされることはありません。LDS 命令によって当該ビットへ 0 を書き込むまで 1 を保持します。また、DSR[11:8]（TS、TC）が 0000 のときは演算結果によって値が更新されることはありません。LDS 命令によって当該ビットへ書き込むまで値を保持します。
13	AN	0	R/W	累積負値ビット 1：演算結果が負、またはオペランド 1 がオペランド 2 より小さいことを示します。 本ビットは演算結果により 0 クリアされることはありません。LDS 命令によって当該ビットへ 0 を書き込むまで 1 を保持します。また、DSR[11:8]（TS、TC）が 0000 のときは演算結果によって値が更新されることはありません。LDS 命令によって当該ビットへ書き込むまで値を保持します。
12	AV	0	R/W	累積オーバフロービット 1：演算結果がオーバフローしたことを示します。 本ビットは演算結果により 0 クリアされることはありません。LDS 命令によって当該ビットへ 0 を書き込むまで 1 を保持します。また、DSR[11:8]（TS、TC）が 0000 のときは演算結果によって値が更新されることはありません。LDS 命令によって当該ビットへ書き込むまで値を保持します。

ビット	ビット名	初期値	R/W	機　　能
11～9	TS	0	R/W	T ビット状態選択 TC ビットが 1 のとき、SR レジスタの T ビットに設定する演算結果状態を選択するモードを指定します。ただし、SR レジスタの S ビットが 1 のときもオーバフロー検出を行います。 000：キャリー／ボローモード 001：負値モード 010：ゼロモード 011：オーバフローモード 100：符合付き大モード 101：符号付き以上モード その他：リザーブ（設定禁止）
8	TC	0	R/W	TC ビット 1：SR レジスタの T ビットは DSP 命令実行時、DSR レジスタの TS ビットの状態により変化します。ただし、条件付き DSP 命令実行時は T ビットは変化しません。 0：SR レジスタの T ビットは DSP 命令に依存しません。
7	GT	0	R/W	符号付き大ビット 演算結果が正（0 を除く）、またはオペランド 1 がオペランド 2 より大きいことを示します。 1：演算結果が正、またはオペランド 1 がオペランド 2 より大きい
6	Z	0	R/W	ゼロビット 演算結果が 0（ゼロ）、またはオペランド 1 がオペランド 2 と等しいことを示します。 1：演算結果がゼロ（0）、または等しい
5	N	0	R/W	負値ビット 演算結果が負、またはオペランド 1 がオペランド 2 より小さいことを示します。 1：演算結果が負、またはオペランド 1 がオペランド 2 より小さい
4	V	0	R/W	オーバフロービット 演算結果がオーバフローしたことを示します。 1：演算結果がオーバフロー

ビット	ビット名	初期値	R/W	機　　能
3～1	CS	すべて 0	R/W	DC ビット状態選択 DC ビットに設定する演算結果状態を選択するためのモードを指定します。 000：キャリー／ボローモード 001：負値モード 010：ゼロモード 011：オーバフローモード 100：符合付き大モード 101：符号付き以上モード その他：リザーブ（設定禁止）
0	DC	0	R/W	DSP 状態ビット CS ビットで指定されたモードで演算結果の状態を設定します。 0：指定されたモードの状態が成立しない（不成立） 1：指定されたモードの状態が成立

【注】　*　PADDC/PSUBC 命令実行後の DC ビットは、CS ビットに関係なくキャリー／ボローモードで演算結果の状態を設定します。

2.3 メモリ割り付けレジスタ

各種制御レジスタの多くは以下のメモリ領域に割り付けられています。これらのメモリ領域に割り付けられたすべてのレジスタには、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。

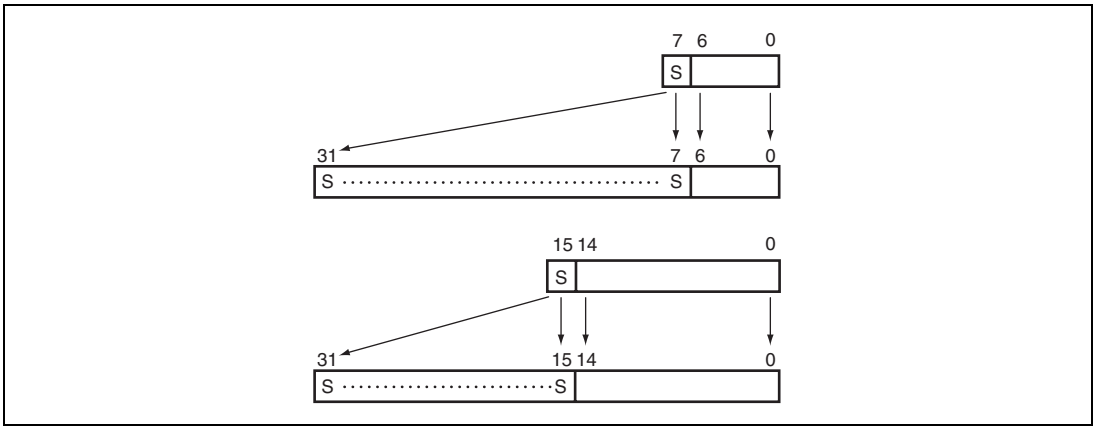


図 2.5 バイトデータ、ワードデータのレジスタ中のデータ形式

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 2.6 に示します。

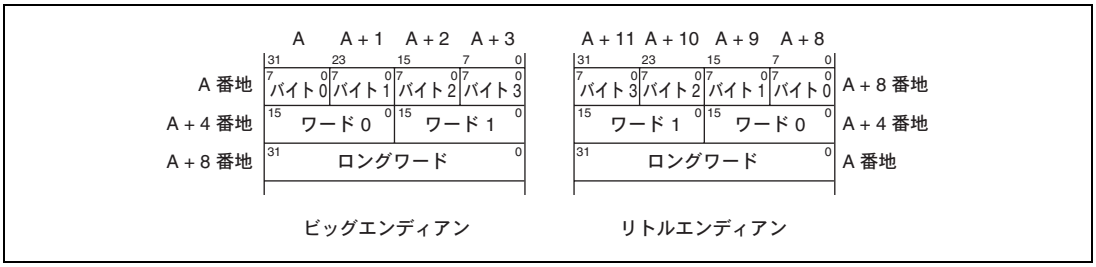


図 2.6 メモリ上のデータ形式

2.6 DSP タイプデータ形式

SH4AL-DSP には命令に対応して 3 つの異なるデータ形式があります。固定小数点データ形式、整数データ形式、論理データ形式です。

DSP タイプの固定小数点データ形式はビット 31 とビット 30 の間に 2 進小数点があります。ガードビット付き、ガードビットなし、乗算入力の 3 種類があり、それぞれ有効ビット長と表せる値の範囲が異なります。

DSP タイプの整数データ形式はビット 16 とビット 15 の間に 2 進小数点があります。ガードビット付き、ガードビットなし、シフト量の 3 種類があり、それぞれ有効ビット長と表せる値の範囲が異なります。算術シフト（PSHA）のシフト量は 7 ビットの領域で－64～＋63 までを表せますが、実際に有効なのは－32～＋32 までの値です。同様に論理シフト（PSHL）のシフト量は 6 ビットの領域ですが、実際に有効なのは－16～＋16 までの値です。

DSP タイプの論理データ形式は小数点がありません。

データ形式とデータの有効な長さは命令と DSP レジスタによって決まります。

3 つの DSP タイプのデータ形式とその 2 進小数点の位置、および参考として SH タイプのデータ形式を図 2.7 に示します。

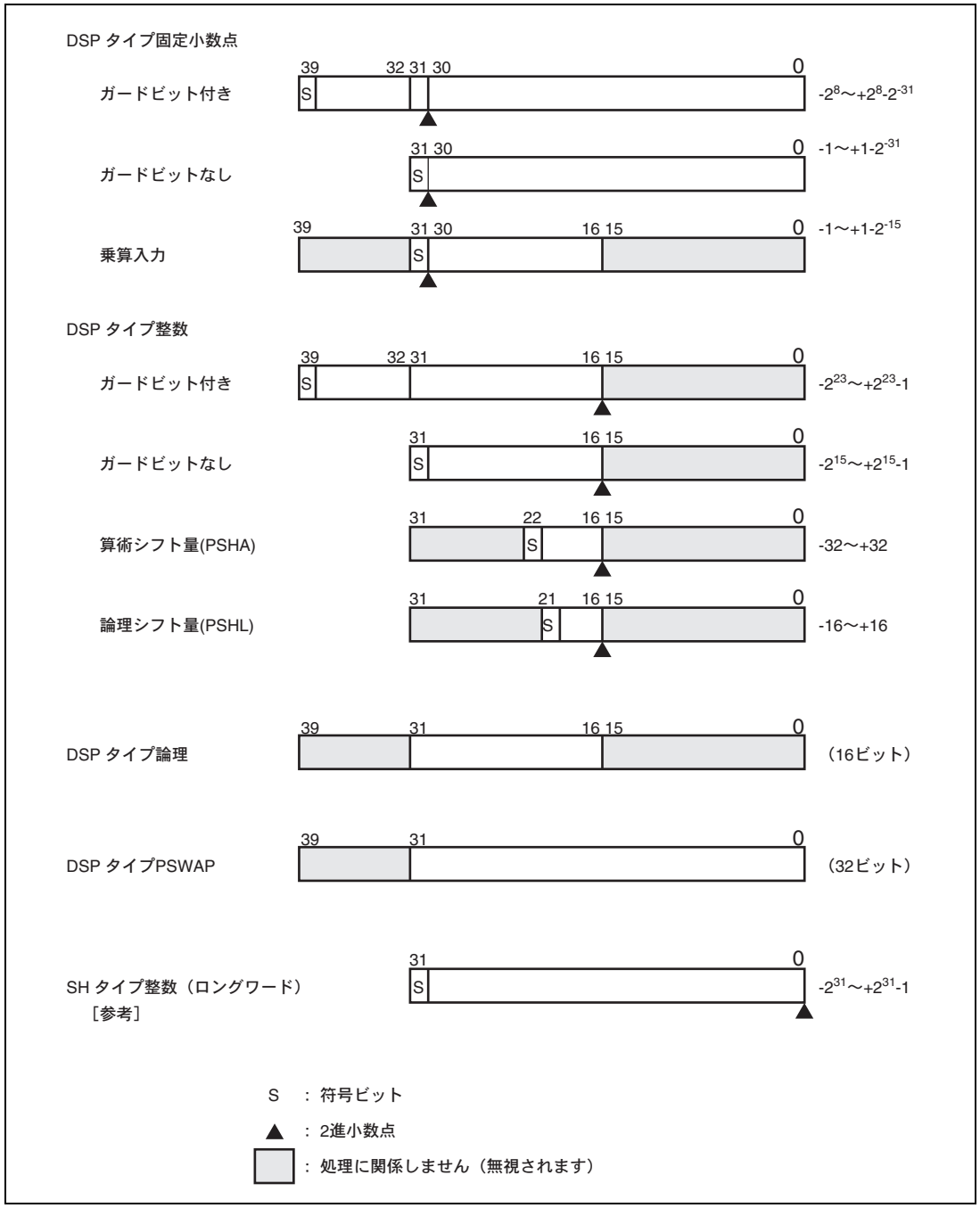


図 2.7 DSP タイプデータ形式

2.7 処理状態

処理状態には、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。

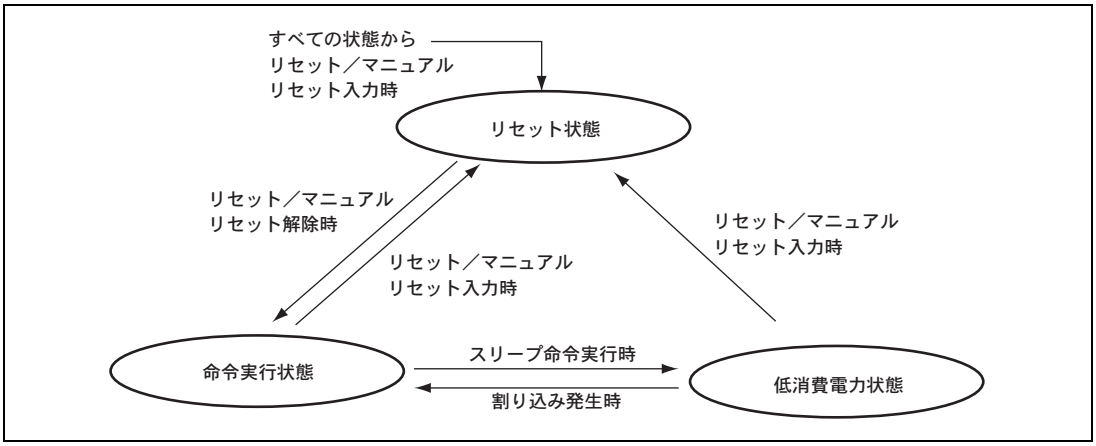


図 2.8 処理状態遷移図

2.8 使用上の注意事項

2.8.1 自己書き換えコードに対する注意事項

SH4AL-DSP 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

SH4AL-DSP の命令セットは固定長 16 ビット命令で実現されます。SH4AL-DSP 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）のデータサイズでメモリにアクセスします。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

SH4AL-DSP 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

SH4AL-DSP 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

遅延命令の次の命令は「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。

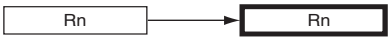
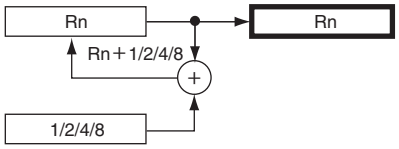
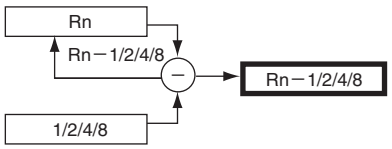
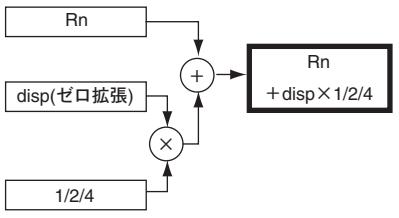
```
MOV.W    @(disp, PC), Rn
MOV.L    @(disp, PC), Rn
```

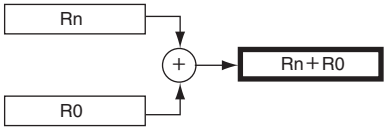
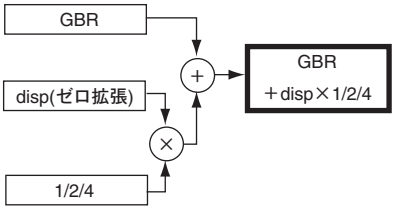
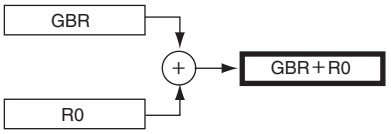
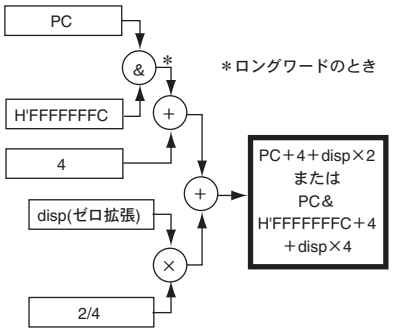
3.2 アドレッシングモード

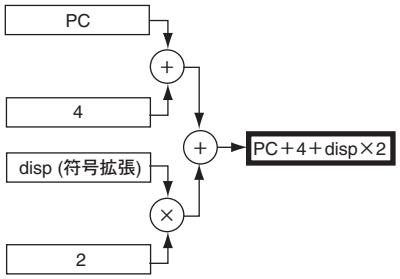
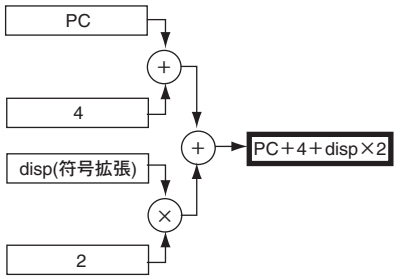
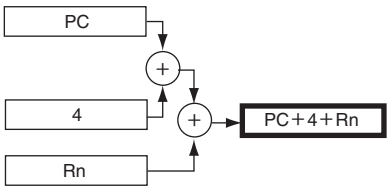
3.2.1 CPU アドレッシングモード

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 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
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 です。 	バイト：Rn－1→Rn ワード：Rn－2→Rn ロングワード：Rn－4→Rn Rn→EA （計算後の Rn で命令実行）
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

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
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
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×4→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
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
	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

3.2.2 DSP データアドレッシング

DSP 命令では 2 つの異なったメモリアクセスをします。1 つは X、Y データ転送命令（MOVX.W、MOVY.W）で、もう 1 つはシングルデータ転送命令（MOVS.W、MOVS.L）です。これらの 2 種類の命令のデータアドレッシングは異なります。

表 3.3 データ転送命令の概要

	ダブルデータ転送命令		シングルデータ転送命令
	MOVX.W MOVY.W	MOVX.W&NOPY NOPX&MOVY.W MOVX.L&NOPY NOPX&MOVY.L	MOVS.W MOVS.L
アドレスレジスタ	Ax：R4、R5 Ay：R6、R7	Axy：R4、R5、R0、R1 Ayx：R6、R7、R2、R3	As：R2、R3、R4、R5
インデックスレジスタ	Ix：R8 Iy：R9	Ix：R8 Iy：R9	Is：R8
アドレッシング	Nop/Inc(+2)/インデクス加算： ポストインクリメント	Nop/Inc(+2/+4)/インデクス加算： ポストインクリメント	Nop/Inc(+2,+4)/インデクス加算： ポストインクリメント
アドレッシング	—	—	Dec(-2,-4)：プリデクリメント
モジュロ アドレッシング	可能	可能	不可
データバス	X バス、Y バス	X バス、Y バス	オペランドバス
データ長	16 ビット（ワード）	16 ビット／32 ビット （ワード／ロングワード）	16 ビット／32 ビット （ワード／ロングワード）
バス競合	なし	なし	あり
メモリ	X、Y データメモリ	X、Y データメモリ	すべてのメモリ空間
ソースレジスタ	Da：A0、A1	Dax：A0、A1、X0、X1 Day：A0、A1、Y0、Y1	Ds：A0、A1、M0、M1、X0、X1、 Y0、Y1、A0G、A1G
デスティネーション レジスタ	Dx：X0、X1 Dy：Y0、Y1	Dxy：X0、X1、Y0、Y1 Dyx：Y0、Y1、X0、X1	Ds：A0、A1、M0、M1、X0、X1、 Y0、Y1、A0G、A1G

3.2.3 X、Y データアドレッシング

DSP 命令では MOVX.W、MOVY.W 命令を使って、X、Y データメモリを同時にアクセスすることができます。DSP 命令には同時に X、Y データメモリをアクセスするために 2 つのアドレスポインタがあります。DSP 命令にはポインタアドレッシングだけが可能で、イミディエイトアドレッシングはありません。アドレスレジスタは 2 つに分けられ、R4、R5 レジスタが X メモリのアドレスレジスタ（Ax）となり、R6、R7 レジスタが Y メモリのアドレスレジスタ（Ay）となります。X、Y データ転送命令には次の 3 つのアドレッシングがあります。

(1) 更新なし

Ax、Ay レジスタがアドレスポインタです。更新されません。

(2) インクリメント

Ax、Ay レジスタがアドレスポインタです。データ転送後それぞれ+2 または+4 が加算されます(ポスト更新)。

(3) インデクスレジスタ加算

Ax、Ay レジスタがアドレスポインタです。データ転送後それぞれ Ix、Iy レジスタの値が加算されます（ポスト更新）。それぞれのアドレスポインタにはインデクスレジスタがあります。R8 レジスタは X メモリアドレスレジスタ（Ax）のインデクスレジスタ（Ix）となり、R9 レジスタは Y メモリアドレスレジスタ（Ay）のインデクスレジスタ（Iy）となります。

X、Y データ転送命令はワードで処理します。X、Y データメモリを 16 ビットでアクセスします。そのためインクリメント処理は、アドレスレジスタに 2 を加えます。デクリメントさせるためには、−2 をインデクスレジスタに設定し加算インデクスレジスタアドレッシングを指定します。

X、Y データ転送のアドレッシングを図 3.1 に示します。

DSP 命令では X、Y データメモリを同時にアクセスできますが、一方の転送動作が不要の場合に転送機能を拡張することができます。（MOVX.W & NOPY、NOPX & MOVY.W）

この形式では 32 ビットデータを転送することもできます。（MOVX.L & NOPY、NOPX & MOVY.L）

このとき、R0、R1、R4、R5 が拡張 X メモリアドレスレジスタ Axy となり、R2、R3、R6、R7 が拡張 Y メモリアドレスレジスタ Ayx となります。

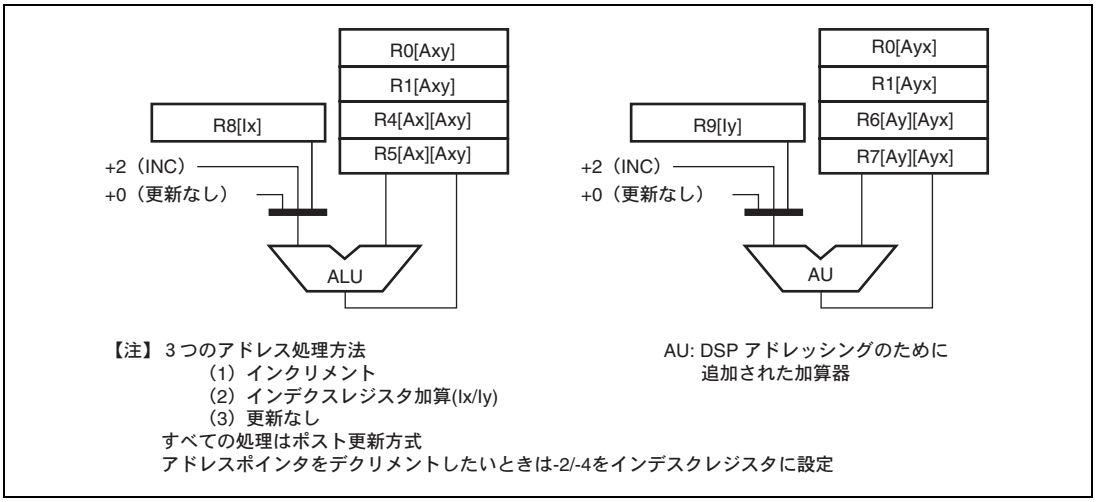


図 3.1　X、Y データ転送のアドレッシング

3.2.4 シングルデータアドレッシング

DSP 命令にはシングルデータ転送命令（MOVS.W、MOVS.L）があり、DSP レジスタにデータをロードし、DSP レジスタからデータをストアします。この命令で R2～R5 レジスタはシングルデータ転送のアドレスレジスタ（As）として使われます。

シングルデータ転送命令には次の 4 つのデータアドレッシングがあります。

(1) 更新なしアドレス

Asレジスタがアドレスポインタです。更新されません。

(2) インデクス加算

Asレジスタがアドレスポインタです。データ転送後Isレジスタの値が加算されます（ポスト更新）。

(3) インクリメントアドレス

Asレジスタがアドレスポインタです。データ転送後＋2または＋4が加算されます（ポスト更新）。

(4) デクリメントアドレス

Asレジスタがアドレスポインタです。データ転送前に－2、－4が加算（＋2または＋4が減算）されます（プレ更新）。

アドレスポインタ（As）は R8 レジスタをインデクスレジスタ（Is）として使います。シングルデータ転送のアドレッシングを**図 3.2** に示します。

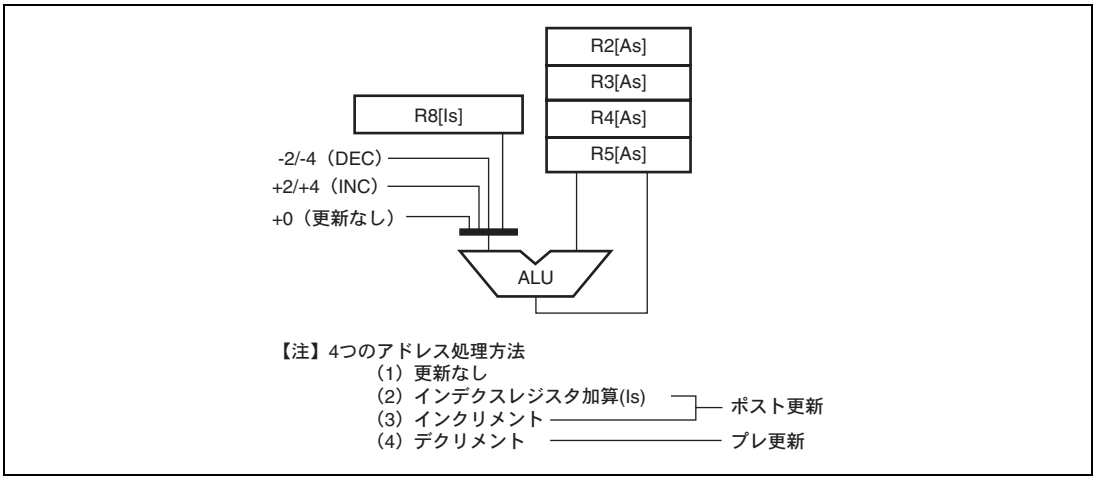


図 3.2 シングルデータ転送のアドレッシング

3.2.5 モジュロアドレッシング

SH4AL-DSP には、他の DSP と同様にモジュロアドレッシングモードがあります。このモードでもアドレスレジスタは同じように更新されます。アドレスポインタの値がすでに設定されたモジュロ終了アドレスになると、アドレスポインタはモジュロ開始アドレスになります。

モジュロアドレッシングは X、Y データ転送命令（MOVX.W、MOVY.W）にだけ有効です。SR レジスタの DMX ビットをセットすると X アドレスレジスタが、DMY ビットをセットすると Y アドレスレジスタがそれぞれモジュロアドレッシングモードになります。モジュロアドレッシングはどちらかの X、Y アドレスレジスタに対してだけ有効です。両方を同時にモジュロアドレッシングモードにすることはできません。したがって、DMX と DMY を同時にセットしないでください。万一同時にセットされた場合には、DMY 側のみ有効となります。

モジュロアドレス領域の開始と終了アドレスを指定するための MOD レジスタがあり、MOD レジスタは MS（Modulo Start： モジュロ開始）と、ME（Modulo End： モジュロ終了）を格納します。MOD レジスタ（MS、ME）の使用例を次に示します。*

```
                MOV.L ModAddr,Rn; Rn=ModEnd, ModStart
                LDC Rn,MOD;       ME=ModEnd, MS=ModStart
ModAddr:        .DATA.W           mEnd;             Lower 8bit of ModEnd
                .DATA.W           mStart;           Lower 8bit of ModStart

ModStart:       .DATA
                  :
ModEnd:         .DATA
```

MS、ME には開始、終了アドレスを指定して、そのあとで DMX 又は DMY ビットを 1 にセットします。アドレスレジスタの内容が ME と比較されます。もし ME と一致したら、開始アドレス MS をアドレスレジスタに格納します。アドレスレジスタの下位 16 ビットが ME と比較されます。最大のモジュロサイズは 64K バイトです。これは X、Y データメモリをアクセスするには十分です。モジュロアドレッシングのブロック図を**図 3.3** に示します。

【注】 * この仕様は将来変更される可能性があります。

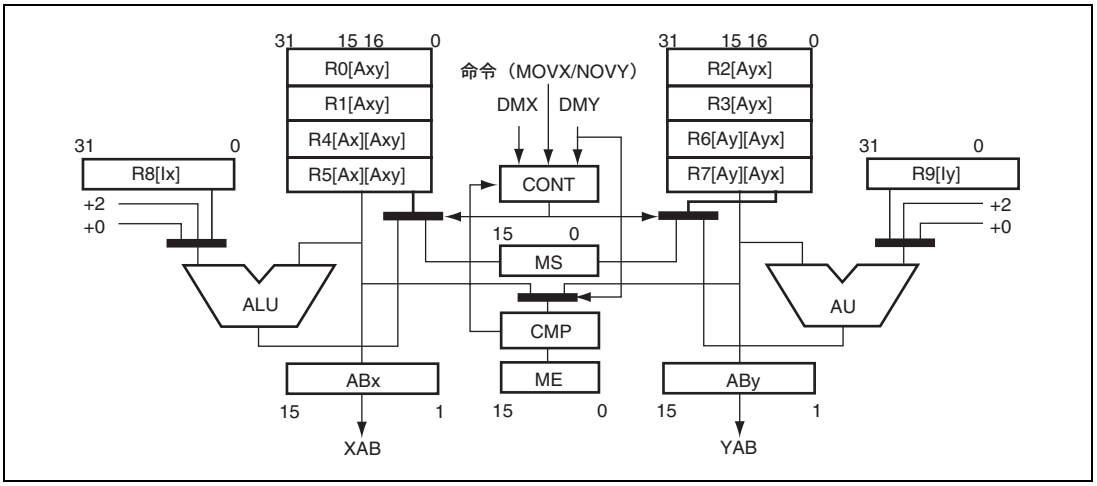


図 3.3　モジュロアドレッシング

モジュロアドレッシングの例を次に示します。

MS＝H'08; ME=H'0C; R4＝H'C008;

DMX＝1; DMY＝0;（アドレスレジスタ Ax（R4,R5）に対するモジュロアドレッシングの設定です）

以上の設定により R4 レジスタは次のように変化します。

R4: H'C008

Inc.　R4: H'C00A

Inc.　R4: H'C00C

Inc.　R4: H'C008（モジュロ終了アドレスになったので、モジュロ開始アドレスになります）

モジュロ開始、終了アドレスの上位 16 ビットは同じになるようデータを配置します。これはモジュロ開始アドレスがアドレスレジスタの下位 16 ビットだけを置き換えるからです。

【注】　DSP データアドレッシングに加算インデクスを使う場合は、アドレスポインタは ME と一致せずにその値を超えてしまうことがあります。この場合は、アドレスポインタはモジュロ開始アドレスには戻りません。

3.2.6　DSP アドレッシング動作

モジュロアドレッシングを含めて、パイプラインの実行ステージ（EX）での DSP アドレッシングの動作を次に示します。

```
if ( Opration is MOVX.W MOVY.W ) {
    ABx=Ax; ABy=Ay;
    /* memory access cycle uses ABx and ABy. The addresses to be used have not been
updated */

    /* Ax is one of R0,R1,R4,R5 */
    if {DMX==0 || (DMX==1 && DMY == 1 )} Ax=Ax+(+2 or R8[Ix] or +0);
    /* Inc,Index,Not-Update */
    else if (! not-update) Ax=modulo( Ax, (+2 or R8[Ix]) );
```

```
    /* Ay is one of R2,R3,R6,7 */
    if ( DMY==0 ) Ay=Ay+(+2 or R9[Iy] or +0); /* Inc,Index,Not-Update */
    else if (! not-update) Ay=modulo( Ay, (+2 or R9[Iy]) );
}
else if ( Operation is MOVS.W or MOVS.L ) {
  if ( Addressing is Nop, Inc, Add-index-reg ) {
    MAB=As;
    /* memory access cycle uses MAB. The address to be used has not been updated */

    /* As is one of R2~5 */
    As=As+(+2 or +4 or R8[Is] or +0); /* Inc,Index,Not-Update */
else { /* Decrement, Pre-update */
    /* As is one of R2~5 */
    As=As+(-2 or -4);
    MAB=As;
    /* memory access cycle uses MAB. The address to be used has been updated */
}

/* The value to be added to the address register depends on addressing operations.
For example, (+2 or R8[Ix] or +0) means that
       +2                    : if operation is increment
    R8[Ix]                   : if operation is add-index-reg
       +0                    : if operation is not-update
*/

function modulo ( AddrReg, Index ) {
    if ( AdrReg[15:0]==ME ) AdrReg[15:0]==MS;
    else AdrReg=AdrReg+Index;
    return AddrReg;
}
```

3.3 命令セット

表 3.5～表 3.13 に示す SuperH 命令の説明に使用する表記を表 3.4 に示します。

表 3.4 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） nnnn：レジスタ番号（Rn） 0000：R0 0001：R1 ： 1111：R15 mmm：レジスタ番号（Rm_BANK） nnn：レジスタ番号（Rn_BANK） 000：R0_BANK 001：R1_BANK ： 111：R7_BANK 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。
DC ビット	命令実行後の DC ビットの値	－：変更なし
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、SH4AL-DSP で新規に追加された命令です。

【注】 スケーリング（×1、×2、×4）は命令オペランドのサイズに応じて実行されます。

表 3.5 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	1001nnnndddddddd	—	—	—
MOV.L	@(disp*,PC),Rn	(disp×4+PC&H'FFFFFFFC+4)→Rn	1101nnnndddddddd	—	—	—
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 3.6 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 3.7　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 3.8　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 3.9　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	—	—	—

【注】 *　遅延スロット命令が 32 ビット命令のときには PC+6→PR

表 3.10 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	—	新規
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	—	新規
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	—	—	新規
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	—	—	新規
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	—	—	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	—	—	新規

命令		動作	命令コード	特権	T ビット	新規
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	—	新規
STC	DBR,Rn	DBR→Rn	0000nnnn11111010	特権	—	新規
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	—	新規
STC.L	DBR,@-Rn	Rn-4→Rn, DBR→(Rn)	0100nnnn11110010	特権	—	新規
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	—	—	—

【注】 * DSP モードではユーザモードで実行可能になります。

表 3.11　DSP をサポートする CPU 命令

命令		動作	命令コード	特権	T ビット	新規
LDC	Rm,MOD	Rm→MOD	0100mmmm01011110	—	—	—
LDC	Rm,RE	Rm→RE	0100mmmm01111110	—	—	—
LDC	Rm,RS	Rm→RS	0100mmmm01101110	—	—	—
LDC.L	@Rm+,MOD	(Rm)→MOD、Rm＋4→Rm	0100mmmm01010111	—	—	—
LDC.L	@Rm+,RE	(Rm)→RE、Rm＋4→Rm	0100mmmm01110111	—	—	—
LDC.L	@Rm+,RS	(Rm)→RS、Rm＋4→Rm	0100mmmm01100111	—	—	—
STC	MOD,Rn	MOD→Rn	0000nnnn01010010	—	—	—
STC	RE,Rn	RE→Rn	0000nnnn01110010	—	—	—
STC	RS,Rn	RS→Rn	0000nnnn01100010	—	—	—
STC.L	MOD,@-Rn	Rn－4→Rn、MOD→(Rn)	0100nnnn01010011	—	—	—
STC.L	RE,@-Rn	Rn－4→Rn、RE→(Rn)	0100nnnn01110011	—	—	—
STC.L	RS,@-Rn	Rn－4→Rn、RS→(Rn)	0100nnnn01100011	—	—	—
LDS	Rm,DSR	Rm→DSR	0100mmmm01101010	—	—	—
LDS.L	@Rm+,DSR	(Rm)→DSR、Rm＋4→Rm	0100mmmm01100110	—	—	—
LDS	Rm,A0	Rm→A0	0100mmmm01111010	—	—	—
LDS.L	@Rm+,A0	(Rm)→A0、Rm＋4→Rm	0100mmmm01110110	—	—	—
LDS	Rm,X0	Rm→X0	0100mmmm10001010	—	—	—
LDS.L	@Rm+,X0	(Rm)→X0、Rm＋4→Rm	0100mmmm10000110	—	—	—
LDS	Rm,X1	Rm→X1	0100mmmm10011010	—	—	—
LDS.L	@Rm+,X1	(Rm)→X1、Rm＋4→Rm	0100mmmm10010110	—	—	—
LDS	Rm,Y0	Rm→Y0	0100mmmm10101010	—	—	—
LDS.L	@Rm+,Y0	(Rm)→Y0、Rm＋4→Rm	0100mmmm10100110	—	—	—
LDS	Rm,Y1	Rm→Y1	0100mmmm10111010	—	—	—
LDS.L	@Rm+,Y1	(Rm)→Y1、Rm＋4→Rm	0100mmmm10110110	—	—	—
STS	DSR,Rn	DSR→Rn	0000nnnn01101010	—	—	—
STS.L	DSR,@-Rn	Rn－4→Rn、DSR→(Rn)	0100nnnn01100010	—	—	—
STS	A0,Rn	A0→Rn	0000nnnn01111010	—	—	—
STS.L	A0,@-Rn	Rn－4→Rn、A0→(Rn)	0100nnnn01110010	—	—	—
STS	X0,Rn	X0→Rn	0000nnnn10001010	—	—	—
STS.L	X0,@-Rn	Rn－4→Rn、X0→(Rn)	0100nnnn10000010	—	—	—
STS	X1,Rn	X1→Rn	0000nnnn10011010	—	—	—
STS.L	X1,@-Rn	Rn－4→Rn、X1→(Rn)	0100nnnn10010010	—	—	—
STS	Y0,Rn	Y0→Rn	0000nnnn10101010	—	—	—
STS.L	Y0,@-Rn	Rn－4→Rn、Y0→(Rn)	0100nnnn10100010	—	—	—
STS	Y1,Rn	Y1→Rn	0000nnnn10111010	—	—	—
STS.L	Y1,@-Rn	Rn－4→Rn、Y1→(Rn)	0100nnnn10110010	—	—	—

命令		動作	命令コード	特権	T ビット	新規
SETRC*2	Rm	Rm[11:0]→RC (SR[27:16])	0100mmmm00010100	—	—	—
SETRC*2	#imm	imm→RC(SR[23:16])、0→SR[27:24]	10000010iiiiiiii	—	—	—
LDRS	@(disp*1,pc)	disp×2+PC→RS	10001100dddddddd	—	—	—
LDRE	@(disp*1,pc)	disp×2+PC→RE	10001110dddddddd	—	—	—
LDRC	Rm	Rm[11:0]→RC (SR[27:16])、1→RE[0]	0100mmmm00110100	—	—	新規
LDRC	#imm	imm→RC (SR[23:16])、0→SR[27:24] 1→RE[0]	10001010iiiiiiii	—	—	新規
SETDMX		1→DMX(SR[10])、0→DMY(SR[11])	0000000010011000	—	—	新規
SETDMY		0→DMX(SR[10])、1→DMY(SR[11])	0000000011001000	—	—	新規
CLRDMXY		0→DMX(SR[10])、0→DMY(SR[11])	0000000010001000	—	—	新規

【注】 *1 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。

*2 SETRC 命令により 1 以上のリピート回数を設定する前に、必ず LDRS 命令と LSRE 命令を毎回実行するようにしてください。

3.4 DSP データ転送命令の命令セット

DSP データ転送命令は 2 つのグループに分けられます。ダブルデータ転送とシングルデータ転送です。ダブルデータ転送は DSP 演算命令と組み合わせて、DSP 並行処理命令することができます。並行処理命令は 32 ビット長で、A フィールドにダブルデータ転送命令が組み込まれます。並行処理命令でないダブルデータ転送とシングルデータ転送命令は 16 ビット長です。

ダブルデータ転送では X メモリと Y メモリを同時に並行してアクセスできます。それぞれ X、Y メモリデータアクセスから一つずつ命令を指定します。Ax ポインタは X メモリをアクセスするために使い、Ay ポインタは Y メモリをアクセスするために使います。ダブルデータ転送は X、Y メモリだけをアクセスできます。

シングルデータ転送はどこのエリアからでもアクセスできます。シングルデータ転送では Ax ポインタとその他の 2 つのポインタを As ポインタとして使います。

3.4.1 ダブルデータ転送命令

表 3.12 ダブルデータ転送命令（X メモリデータ）

命令	動作	命令コード	特権	DC ビット	T ビット	新規
NOPX	No Operation	1111000*0*0*00**	—	—	—	—
MOVX.W @Ax,Dx	(Ax)→Dx の MSW、0→Dx の LSW	111100A*D*0*01**	—	—	—	—
MOVX.W @Ax+,Dx	(Ax)→Dx の MSW、0→Dx の LSW、 Ax＋2→Ax	111100A*D*0*10**	—	—	—	—
MOVX.W @Ax+Ix,Dx	(Ax)→Dx の MSW、0→Dx の LSW、 Ax＋Ix→Ax	111100A*D*0*11**	—	—	—	—
MOVX.W @Axy,Dxy	(Axy)→Dxy の MSW、 0→Dxy の LSW	111100AADD000100	—	—	—	新規

命令	動作	命令コード	特権	DC ビット	T ビット	新規
MOVX.W @Axy+,Dxy	(Axy)→Dxy の MSW、 0→Dxy の LSW、 Axy+2→Axy	111100AADD001000	—	—	—	新規
MOVX.W @Axy+Ix,Dxy	(Axy)→Dxy の MSW、 0→Dxy の LSW, Axy+Ix→Axy	111100AADD001100	—	—	—	新規
MOVX.L @ Axy,Dxy	(Axy)→Dxy	111100AADD010100	—	—	—	新規
MOVX.L @ Axy+,Dxy	(Axy)→Dxy、 Axy+4→Axy	111100AADD011000	—	—	—	新規
MOVX.L @ Axy+Ix,Dxy	(Axy)→Dxy、Axy＋Ix→Axy	111100AADD011100	—	—	—	新規
MOVX.W Da,@Ax	Da の MSW→(Ax)	111100A*D*1*01**	—	—	—	—
MOVX.W Da,@Ax+	Da の MSW→(Ax)、Ax＋2→Ax	111100A*D*1*10**	—	—	—	—
MOVX.W Da,@Ax+Ix	Da の MSW→(Ax)、Ax＋Ix→Ax	111100A*D*1*11**	—	—	—	—
MOVX.W Dax,@Axy	Dax の MSW→(Axy)	111100AADD100100	—	—	—	新規
MOVX.W Dax,@Axy+	Dax の MSW→(Axy)、 Axy+2→Axy	111100AADD101000	—	—	—	新規
MOVX.W Dax,@Axy+Ix	Dax の MSW→(Axy)、 Axy+Ix→Axy	111100AADD101100	—	—	—	新規
MOVX.L Dax,@Axy	Dax→(Axy)	111100AADD110100	—	—	—	新規
MOVX.L Dax,@Axy+	Dax→(Axy)、 Axy+4→Axy	111100AADD111000	—	—	—	新規
MOVX.L Dax,@Axy+Ix	Dax→(Axy)、 Axy+Ix→Axy	111100AADD111100	—	—	—	新規
NOPY	No Operation	111100*0*0*0**00	—	—	—	—
MOVY.W @Ay,Dy	(Ay)→Dy の MSW、0→Dy の LSW	111100*A*D*0**01	—	—	—	—
MOVY.W @Ay+,Dy	(Ay)→Dy の MSW、 0→Dy の LSW、Ay＋2→Ay	111100*A*D*0**10	—	—	—	—
MOVY.W @Ay+Iy,Dy	(Ay)→Dy の MSW、 0→Dy の LSW、Ay＋Iy→Ay	111100*A*D*0**11	—	—	—	—
MOVY.W @Ayx,Dyx	(Ayx)→Dyx の MSW、 0→Dyx の LSW	111100AADD000001	—	—	—	新規
MOVY.W @Ayx+,Dyx	(Ayx)→Dyx の MSW、 0→Dyx の LSW、 Ayx+2→Ayx	111100AADD000010	—	—	—	新規
MOVY.W @Ayx+Ix,Dyx	(Ayx)→Dyx の MSW, 0→Dyx の LSW、 Ayx+Ix→Ayx	111100AADD000011	—	—	—	新規
MOVY.L @Ayx,Dyx	(Ayx)→Dyx	111100AADD100001	—	—	—	新規

命令	動作	命令コード	特権	DC ビット	T ビット	新規
MOVY.L @Ayx+,Dyx	(Ayx)→Dyx、 Ayx+4→Ayx	111100AADD100010	—	—	—	新規
MOVY.L @Ayx+Ix,Dyx	(Ayx)→Dyx、 Ayx+Ix→Ayx	111100AADD100011	—	—	—	新規
MOVY.W Da,@Ay	Da の MSW→(Ay)	111100*A*D*1**01	—	—	—	—
MOVY.W Da,@Ay+	Da の MSW→(Ay)、Ay＋2→Ay	111100*A*D*1**10	—	—	—	—
MOVY.W Da,@Ay+Iy	Da の MSW→(Ay)、Ay＋Iy→Ay	111100*A*D*1**11	—	—	—	—
MOVY.W Day,@Ayx	Day の MSW→(Ayx)	111100AADD010001	—	—	—	新規
MOVY.W Day,@Ayx+	Day の MSW→(Ayx)、 Ayx+2→Ayx	111100AADD010010	—	—	—	新規
MOVY.W Day,@Ayx+Iy	Day の MSW→(Ayx)、 Ayx+Iy→Ayx	111100AADD010011	—	—	—	新規
MOVY.L Day,@Ayx	Day→(Ayx)	111100AADD110001	—	—	—	新規
MOVY.L Day,@Ayx+	Day→(Ayx)、 Ayx+4→Ayx	111100AADD110010	—	—	—	新規
MOVY.L Day,@Ayx+Iy	Day→(Ayx)、 Ayx+Iy→Ayx	111100AADD110011	—	—	—	新規

表 3.13　シングルデータ転送命令

命令	動作	命令コード	特権	DC ビット	T ビット	新規
MOVS.W @-As,Ds	As－2→As、(As)→Ds の MSW、 0→Ds の LSW	111101AADDDD0000	—	—	—	—
MOVS.W @As,Ds	(As)→Ds の MSW、0→Ds の LSW	111101AADDDD0100	—	—	—	—
MOVS.W @As+,Ds	(As)→Ds の MSW、0→Ds の LSW、 As＋2→As	111101AADDDD1000	—	—	—	—
MOVS.W @As+Is,Ds	(As)→Ds の MSW、0→Ds の LSW、 As＋Is→As	111101AADDDD1100	—	—	—	—
MOVS.W Ds,@-As	As－2→As、Ds の MSW→(As)*	111101AADDDD0001	—	—	—	—
MOVS.W Ds,@As	Ds の MSW→(As)*	111101AADDDD0101	—	—	—	—
MOVS.W Ds,@As+	Ds の MSW→(As)、As＋2→As*	111101AADDDD1001	—	—	—	—
MOVS.W Ds,@As+Is	Ds の MSW→(As)、As＋Is→As*	111101AADDDD1101	—	—	—	—
MOVS.L @-As,Ds	As－4→As、(As)→Ds	111101AADDDD0010	—	—	—	—
MOVS.L @As,Ds	(As)→Ds	111101AADDDD0110	—	—	—	—
MOVS.L @As+,Ds	(As)→Ds、As＋4→As	111101AADDDD1010	—	—	—	—
MOVS.L @As+Is,Ds	(As)→Ds、As＋Is→As	111101AADDDD1110	—	—	—	—

命令	動作	命令コード	特権	DC ビット	T ビット	新規
MOVS.L Ds,@-As	As－4→As、Ds→(As)	111101AADDDD0011	—	—	—	—
MOVS.L Ds,@As	Ds→(As)	111101AADDDD0111	—	—	—	—
MOVS.L Ds,@As+	Ds→(As)、As＋4→As	111101AADDDD1011	—	—	—	—
MOVS.L Ds,@As+Is	Ds→(As)、As＋Is→As	111101AADDDD1111	—	—	—	—

【注】 * ソースオペランド Ds にガードビットレジスタ A0G、A1G を指定した場合は、データは LDB[7:0] バスに出力され、符号ビットが上位ビット [31:8]に出力されます。

DSP データ転送のオペランドとレジスタとの対応を表 3.14 に示します。CPU コアのレジスタはメモリアドレスを示すポインタアドレスとして使われます。

表 3.14 DSP データ転送のオペランドとレジスタとの対応

オペランド	SH（CPU コア）レジスタ									
	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9
Ax					0 （Yes）	1 （Yes）				
Ix（Is）									1 （Yes）	
Dx										
Ay							0 （Yes）	1 （Yes）		
Iy										1 （Yes）
Dy										
Da										
As			2 （Yes）	3 （Yes）	0 （Yes）	1 （Yes）				
Ds										

オペランド	DSP レジスタ									
	X0	X1	Y0	Y1	M0	M1	A0	A1	A0G	A1G
Ax										
Ix（Is）										
Dx	0 （Yes）	1 （Yes）								
Ay										
Iy										
Dy			0 （Yes）	1 （Yes）						
Da							0 （Yes）	1 （Yes）		
As										
Ds	8 （Yes）	9 （Yes）	A （Yes）	B （Yes）	C （Yes）	E （Yes）	7 （Yes）	5 （Yes）	F （Yes）	D （Yes）

オペランド	DSP レジスタ									
	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9
Axy	1 （Yes）	3 （Yes）			0 （Yes）	2 （Yes）				
Ayx			2 （Yes）	3 （Yes）			0 （Yes）	1 （Yes）		

オペランド	DSP レジスタ									
	X0	X1	Y0	Y1	M0	M1	A0	A1	A0G	A1G
Dxy	0 （Yes）	2 （Yes）	1 （Yes）	3 （Yes）						
Dyx	2 （Yes）	3 （Yes）	0 （Yes）	1 （Yes）						
Dax	1 （Yes）	3 （Yes）					0 （Yes）	2 （Yes）		
Day			2 （Yes）	3 （Yes）			0 （Yes）	1 （Yes）		

【注】　Yes：設定可能なレジスタ

3.5 DSP 演算命令の命令セット

DSP 演算命令は DSP ユニットで処理されるデジタル信号処理の命令です。これらの命令は 32 ビット長さの命令コードで、複数の命令を並行して実行します。命令コードは A フィールド、B フィールドの 2 つに分かれており、A フィールドにはパラレルデータ転送命令を指定し、B フィールドにはシングルまたはダブルデータ演算命令を指定します。命令は独立して指定することができ、実行も独立に並行して実行されます。A フィールドに指定するパラレルデータ転送命令はダブルデータ転送命令と全く同じです。

B フィールドのデータ演算命令は 3 つに分かれています。ダブルデータ演算命令、条件付きシングルデータ演算命令、無条件シングルデータ演算命令の 3 つです。DSP 演算命令の命令形式を**表 3.15** に示します。それぞれのオペランドは独立に DSP レジスタから選べます。DSP 演算命令のオペランドとレジスタの対応を**表 3.16** に示します。

表 3.15　DSP 演算命令の命令形式

分類		命令形式	命令
ダブルデータ演算命令 （6 オペランド）		ALUop. Sx, Sy, Du MLTop. Se, Sf, Dg	PADD PMULS, PCLR PMULS PSUB PMULS
条件付き シングルデータ 演算命令	3 オペランド	ALUop. Sx, Sy, Dz DCT ALUop. Sx, Sy, Dz DCF ALUop. Sx, Sy, Dz	PADD, PAND, POR, PSHA, PSHL, PXOR
		ALUop. Sx, Sy, Dz DCT ALUop. Sx, Sy, Dz DCF ALUop. Sx, Sy, Dz ALUop. Sy, Sx, Dz DCT ALUop. Sy, Sx, Dz DCF ALUop. Sy, Sx, Dz	PSUB
	2 オペランド	ALUop. Sx, Dz DCT ALUop. Sx, Dz DCF ALUop. Sx, Dz ALUop. Sy, Dz DCT ALUop. Sy, Dz DCF ALUop. Sy, Dz	PCOPY, PDEC, PDMSB, PINC, PLDS, PSTS, PNEG PABS, PRND, PSWAP
	1 オペランド	ALUop. Dz DCT ALUop. Dz DCF ALUop. Dz	PCLR, PSHA #imm, PSHL #imm
無条件 シングルデータ 演算命令	3 オペランド	ALUop. Sx, Sy, Du MLTop. Se, Sf, Dg	PADDC, PSUBC, PWADD, PWSB, PMULS
	2 オペランド	ALUop. Sx, Sy	PCMP

表 3.16　DSP 命令のオペランドとレジスタの対応

レジスタ	ALU、SFT 命令				乗算命令		
	Sx	Sy	Dz	Du	Se	Sf	Dg
A0	Yes		Yes	Yes			Yes
A1	Yes		Yes	Yes	Yes	Yes	Yes
M0		Yes	Yes				Yes
M1		Yes	Yes				Yes
X0	Yes		Yes	Yes	Yes	Yes	
X1	Yes		Yes		Yes		
Y0		Yes	Yes	Yes	Yes	Yes	
Y1		Yes	Yes			Yes	

並行命令を書くときは最初に B フィールドの命令を書いて、次に A フィールドの命令を書きます。並行処理プログラム例を図 3.4 に示します。

```
      PADD A0, M0, A0  PMULS X0, Y0, M0   MOVX.W @R4+, X0       MOVY.W @R6+, Y0 [;]
DCF   PINC X1, A1                         MOVX.W A0, @R5+R8     MOVY.W @R7+, Y0 [;]
      PCMP X1, M0                         MOVX.W @R4            [NOPY] [;]
```

図 3.4　並行処理プログラム例

ここで [ ] は省略可能を意味します。無操作命令 NOPX、NOPY は省略できます。';' は命令行の区切りですが、省略できます。もし区切り ';' を使うときはその後ろをコメント欄として使うことができます。

DSR レジスタの各状態コード（DC、N、Z、V、GT）は無条件の ALU 演算命令、シフト演算命令で常に更新されます。条件付き命令は条件が成立した場合でも状態コードを更新しません。乗算命令も状態コードを更新しません。DC ビットの定義は、DSR レジスタの CS ビットの指定によって決まります。

表 3.17　ALU 固定小数点算術演算命令

命令	動作	命令コード	特権	DC ビット	T ビット	新規
PABS Sx,Dz	もし Sx≧0 ならば Sx→Dz もし Sx＜0 ならば 0-Sx→Dz	111110********** 10001000xx00zzzz	—	更新	更新	—
PABS Sy,Dz	もし Sy≧0 ならば Sy→Dz もし Sy＜0 ならば 0-Sy→Dz	111110********** 1010100000yyzzzz	—	更新	更新	—
DCT PABS Sx,Dz	もし DC=1 & Sx≧0 ならば Sx→Dz もし DC=1 & Sx＜0 ならば 0-Sx→Dz もし DC＝0 ならば nop.	111110********** 10001010xx01zzzz	—	—	—	新規
DCT PABS Sy,Dz	もし DC=1 & Sy≧0 ならば Sy→Dz もし DC=1 & Sy＜0 ならば 0-Sy→Dz もし DC＝0 ならば nop.	111110********** 1010101001yyzzzz	—	—	—	新規
DCF PABS Sx,Dz	もし DC=0 & Sx≧0 ならば Sx→Dz もし DC=0 & Sx＜0 ならば 0-Sx→Dz もし DC＝1 ならば nop.	111110********** 10001011xx01zzzz	—	—	—	新規
DCF PABS Sy,Dz	もし DC=0 & Sy≧0 ならば Sy→Dz もし DC=0 & Sy＜0 ならば 0-Sy→Dz もし DC＝1 ならば nop.	111110********** 1010101101yyzzzz	—	—	—	新規
PADD Sx,Sy,Dz	Sx＋Sy→Dz	111110********** 10110001xxyyzzzz	—	更新	更新	—
DCT PADD Sx,Sy,Dz	もし DC＝1 ならば Sx＋Sy→Dz もし DC＝0 ならば nop	111110********** 10110010xxyyzzzz	—	—	—	—
DCF PADD Sx,Sy,Dz	もし DC＝0 ならば Sx＋Sy→Dz もし DC＝1 ならば nop	111110********** 10110011xxyyzzzz	—	—	—	—
PADD Sx,Sy,Du PMULS Se,Sf,Dg	Sx＋Sy→Du Se の MSW×Sf の MSW→Dg	111110********** 0111eeffxxyygguu	—	更新*[1]	更新*[1]	—
PADDC Sx,Sy,Dz	Sx＋Sy＋DC→Dz	111110********** 10110000xxyyzzzz	—	更新	更新	—
PCLR Dz	H'00000000→Dz	111110********** 1000110100000zzzz	—	更新	更新	—
DCT PCLR Dz	もし DC＝1 ならば H'00000000→Dz もし DC＝0 ならば nop.	111110********** 1000111000000zzzz	—	—	—	—

命令	動作	命令コード	特権	DC ビット	T ビット	新規
DCF PCLR Dz	もし DC＝0 ならば H'00000000→Dz もし DC＝1 ならば nop.	111110********** 100011110000zzzz	—	—	—	—
PCLR Du PMULS Se,Sf,Dg	H'00000000→Du Se の MSW× Sf の MSW→Dg	111110********** 0100eeff0001gguu	—	更新*2	更新*2	新規
PCMP Sx,Sy	Sx－Sy	111110********** 10000100xxyy0000	—	更新	更新	—
PCOPY Sx,Dz	Sx→Dz	111110********** 11011001xx00zzzz	—	更新	更新	—
PCOPY Sy,Dz	Sy→Dz	111110********** 1111100100yyzzzz	—	更新	更新	—
DCT PCOPY Sx,Dz	もし DC＝1 ならば Sx→Dz もし DC＝0 ならば nop.	111110********** 11011010xx00zzzz	—	—	—	—
DCT PCOPY Sy,Dz	もし DC＝1 ならば Sy→Dz もし DC＝0 ならば nop.	111110********** 1111101000yyzzzz	—	—	—	—
DCF PCOPY Sx,Dz	もし DC＝0 ならば Sx→Dz もし DC＝1 ならば nop	111110********** 11011011xx00zzzz	—	—	—	—
DCF PCOPY Sy,Dz	もし DC＝0 ならば Sy→Dz もし DC＝1 ならば nop	111110********** 1111101100yyzzzz	—	—	—	—
PNEG Sx,Dz	0－Sx→Dz	111110********** 11001001xx00zzzz	—	更新	更新	—
PNEG Sy,Dz	0－Sy→Dz	111110********** 1110100100yyzzzz	—	更新	更新	—
DCT PNEG Sx,Dz	もし DC＝1 ならば 0－Sx→Dz もし DC＝0 ならば nop.	111110********** 11001010xx00zzzz	—	—	—	—
DCT PNEG Sy,Dz	もし DC＝1 ならば 0－Sy→Dz もし DC＝0 ならば、nop.	111110********** 1110101000yyzzzz	—	—	—	—
DCF PNEG Sx,Dz	もし DC＝0 ならば 0－Sx→Dz もし DC＝1 ならば nop.	111110********** 11001011xx00zzzz	—	—	—	—
DCF PNEG Sy,Dz	もし DC＝0 ならば 0－Sy→Dz もし DC＝1 ならば nop.	111110********** 1110101100yyzzzz	—	—	—	—
PSUB Sx,Sy,Dz	Sx－Sy→Dz	111110********** 10100001xxyyzzzz	—	更新	更新	—
PSUB Sy,Sx,Dz	Sy－Sx→Dz	111110********** 10000101xxyyzzzz	—	更新	更新	新規
DCT PSUB Sx,Sy,Dz	もし DC＝1 ならば Sx－Sy→Dz もし DC＝0 ならば nop	111110********** 10100010xxyyzzzz	—	—	—	—
DCT PSUB Sy,Sx,Dz	もし DC＝1 ならば Sy－Sx→Dz もし DC＝0 ならば nop	111110********** 10000110xxyyzzzz	—	—	—	新規
DCF PSUB Sx,Sy,Dz	もし DC＝0 ならば Sx－Sy→Dz もし DC＝1 ならば nop	111110********** 10100011xxyyzzzz	—	—	—	—

命令	動作	命令コード	特権	DC ビット	T ビット	新規
DCF PSUB Sy,Sx,Dz	もし DC＝0 ならば Sy－Sx→Dz もし DC＝1 ならば nop	111110********** 10000111xxyyzzzz	—	—	—	新規
PSUB Sx,Sy,Du PMULS Se,Sf,Dg	Sx－Sy→Du Se の MSW×Sf の MSW→Dg	111110********** 0110eeffxxyygguu	—	更新*3	更新*3	—
PSUBC Sx,Sy,Dz	Sx－Sy－DC→Dz	111110********** 10100000xxyyzzzz	—	更新	更新	—

【注】 *1 PADD の演算結果に基づいて更新されます。

*2 PCLR の演算結果に基づいて更新されます。

*3 PSUB の演算結果に基づいて更新されます。

表 3.18　ALU 整数演算命令

命令	動作	命令コード	特権	DC ビット	T ビット	新規
PDEC Sx,Dz	Sx の MSW－1→Dz、 Dz の LSW クリア	111110********** 10001001xx00zzzz	—	更新	更新	—
PDEC Sy,Dz	Sy の MSW－1→Dz、 Dz の LSW クリア	111110********** 10101001xx00zzzz	—	更新	更新	—
DCT PDEC Sx,Dz	もし DC＝1 ならば Sx の MSW－1→Dz、 Dz の LSW クリア もし DC＝0 ならば nop.	111110********** 10001010xx00zzzz	—	—	—	—
DCT PDEC Sy,Dz	もし DC＝1 ならば Sy の MSW－1→Dz、 Dz の LSW クリア もし DC＝0 ならば nop.	111110********** 10101010xx00zzzz	—	—	—	—
DCF PDEC Sx,Dz	もし DC＝0 ならば Sx の MSW－1→Dz、 Dz の LSW クリア もし DC＝1 ならば nop.	111110********** 10001011xx00zzzz	—	—	—	—
DCF PDEC Sy,Dz	もし DC＝0 ならば Sy の MSW－1→Dz、Dz の LSW クリア もし DC＝1 ならば nop.	111110********** 10101011xx00zzzz	—	—	—	—
PINC Sx,Dz	Sx の MSW＋1→Dz、 Dz の LSW クリア	111110********** 10011001xx00zzzz	—	更新	更新	—
PINC Sy,Dz	Sy の MSW＋1→Dz、 Dz の LSW クリア	111110********** 1011100100yyzzzz	—	更新	更新	—
DCT PINC Sx,Dz	もし DC＝1 ならば Sx の MSW＋1→Dz、 Dz の LSW クリア もし DC＝0 ならば nop.	111110********** 10011010xx00zzzz	—	—	—	—
DCT PINC Sy,Dz	もし DC＝1 ならば Sy の MSW＋1→Dz、 Dz の LSW クリア もし DC＝0 ならば nop.	111110********** 1011101000yyzzzz	—	—	—	—
DCF PINC Sx,Dz	もし DC＝0 ならば Sx の MSW＋1→Dz、 Dz の LSW クリア もし DC＝1 ならば nop.	111110********** 10011011xx00zzzz	—	—	—	—
DCF PINC Sy,Dz	もし DC＝0 ならば Sy の MSW＋1→Dz、 Dz の LSW クリア もし DC＝1 ならば nop.	111110********** 1011101100yyzzzz	—	—	—	—

表 3.19　ALU 論理演算命令

命令	動作	命令コード	特権	DC ビット	T ビット	新規
PAND Sx,Sy,Dz	Sx & Sy→Dz、 Dz ガードビットと LSW クリア	111110********** 10010101xxyyzzzz	—	更新	更新	
DCT PAND Sx,Sy,Dz	もし DC＝1 ならば Sx&Sy→Dz、 Dz ガードビットと LSW クリア もし DC＝0 ならば nop	111110********** 10010110xxyyzzzz	—	—	—	
DCF PAND Sx,Sy,Dz	もし DC＝0 ならば Sx&Sy→Dz、 Dz ガードビットと LSW クリア もし DC＝1 ならば nop	111110********** 10010111xxyyzzzz	—	—	—	
POR Sx,Sy,Dz	Sx | Sy→Dz、 Dz のガードビットと LSW クリア	111110********** 10110101xxyyzzzz	—	更新	更新	
DCT POR Sx,Sy,Dz	もし DC＝1 ならば Sx | Sy→Dz、 Dz のガードビットと LSW クリア もし DC＝0 ならば nop.	111110********** 10110110xxyyzzzz	—	—	—	
DCF POR Sx,Sy,Dz	もし DC＝0 ならば Sx | Sy→Dz、 Dz のガードビットと LSW クリア もし DC＝1 ならば nop.	111110********** 10110111xxyyzzzz	—	—	—	
PXOR Sx,Sy,Dz	Sx ^ Sy→Dz、 Dz のガードビットと LSW クリア	111110********** 10100101xxyyzzzz	—	更新	更新	
DCT PXOR Sx,Sy,Dz	もし DC＝1 ならば Sx^Sy→Dz、 Dz のガードビットと LSW クリア もし DC＝0 ならば nop.	111110********** 10100110xxyyzzzz	—	—	—	
DCF PXOR Sx,Sy,Dz	もし DC＝0 ならば Sx^Sy→Dz、 Dz のガードビットと LSW クリア もし DC＝1 ならば nop.	111110********** 10100111xxyyzzzz	—	—	—	

表 3.20　固定小数点乗算命令

命令	動作	命令コード	特権	DC ビット	T ビット	新規
PMULS Se,Sf,Dg	Se の MSW×Sf の MSW→Dg	111110********** 0100eeff0000gg00	—	—	—	—

表 3.21　算術シフト演算命令

命令	動作	命令コード	特権	DC ビット	T ビット	新規
PSHA Sx,Sy,Dz	もし Sy≧0 ならば Sx<<Sy→Dz もし Sy<0 ならば Sx>>|Sy|→Dz	111110********** 10010001xxyyzzzz	—	更新	更新	—
DCT PSHA Sx,Sy,Dz	もし DC＝1 & Sy≧0 ならば Sx<<Sy→Dz もし DC＝1 & Sy<0 ならば Sx>>|Sy|→Dz もし DC＝0 ならば nop	111110********** 10010010xxyyzzzz	—	—	—	—
DCF PSHA Sx,Sy,Dz	もし DC＝0 & Sy≧0 ならば Sx<<Sy→Dz もし DC＝0 & Sy<0 ならば Sx>>|Sy|→Dz もし DC＝1 ならば nop	111110********** 10010011xxyyzzzz	—	—	—	—
PSHA #Imm,Dz	もし Imm≧0 ならば Dz<<Imm→Dz もし Imm<0 ならば Dz>>|Imm|→Dz	111110********** 00010iiiiiiizzzz	—	更新	更新	—

表 3.22　論理シフト演算命令

命令	動作	命令コード	特権	DC ビット	T ビット	新規
PSHL Sx,Sy,Dz	もし Sy≧0 ならば Sx<<Sy→Dz, Dz の LSW クリア もし Sy<0 ならば Sx>>|Sy|→Dz, Dz の LSW クリア	111110********** 10000001xxyyzzzz	—	更新	更新	—
DCT PSHL Sx,Sy,Dz	もし DC＝1 & Sy≧0 ならば Sx<<Sy→Dz, Dz の LSW クリア もし DC＝1 & Sy<0 ならば Sx>>|Sy|→Dz,Dz の LSW クリア もし DC＝0 ならば nop	111110********** 10000010xxyyzzzz	—	—	—	—
DCF PSHL Sx,Sy,Dz	もし DC＝0 & Sy≧0 ならば Sx<<Sy→Dz, Dz の LSW クリア もし DC＝0 & Sy<0 ならば Sx>>|Sy|→Dz,Dz の LSW クリア もし DC＝1 ならば nop	111110********** 10000011xxyyzzzz	—	—	—	—
PSHL #Imm,Dz	もし Imm≧0 ならば Dz<<Imm→Dz,Dz の LSW クリア もし Imm<0 ならば Dz>>|Imm|→Dz,Dz の LSW クリア	111110********** 00000iiiiiiizzzz	—	更新	更新	—

表 3.23 MSB 検出命令

命令	動作	命令コード	特権	DC ビット	T ビット	新規
PDMSB Sx,Dz	Sx の MSB 位置→Dz の MSW、 Dz の LSW をクリア	111110********** 10011101xx00zzzz	—	更新	更新	—
PDMSB Sy,Dz	Sy の MSB 位置→Dz の MSW、 Dz の LSW をクリア	111110********** 1011110100yyzzzz	—	更新	更新	—
DCT PDMSB Sx,Dz	もし DC＝1 ならば Sx の MSB 位置→Dz の MSW、 Dz の LSW をクリア もし DC＝0 ならば nop.	111110********** 10011110xx00zzzz	—	—	—	—
DCT PDMSB Sy,Dz	もし DC＝1 ならば Sy の MSB 位置→Dz の MSW、 Dz の LSW をクリア もし DC＝0 ならば nop.	111110********** 1011111000yyzzzz	—	—	—	—
DCF PDMSB Sx,Dz	もし DC＝0 ならば Sx の MSB 位置→Dz の MSW、 Dz の LSW をクリア もし DC＝1 ならば nop.	111110********** 10011111xx00zzzz	—	—	—	—
DCF PDMSB Sy,Dz	もし DC＝0 ならば Sy の MSB 位置→Dz の MSW、 Dz の LSW をクリア もし DC＝1 ならば nop.	111110********** 1011111100yyzzzz	—	—	—	—

表 3.24 丸め演算命令

命令	動作	命令コード	特権	DC ビット	T ビット	新規
PRND Sx,Dz	Sx＋H'00008000→Dz、 Dz の LSW クリア	111110********** 10011000xx00zzzz	—	更新	更新	—
PRND Sy,Dz	Sy＋H'00008000→Dz、 Dz の LSW クリア	111110********** 1011100000yyzzzz	—	更新	更新	—
DCT PRND Sx,Dz	もし DC＝1 ならば Sx＋H'00008000→Dz Dz の LSW クリア もし DC＝0 ならば nop.	111110********** 10011010xx01zzzz	—	—	—	新規
DCT PRND Sy,Dz	もし DC＝1 ならば Sy＋H'00008000→Dz Dz の LSW クリア もし DC＝0 ならば nop.	111110********** 1011101001yyzzzz	—	—	—	新規
DCF PRND Sx,Dz	もし DC＝0 ならば Sx＋H'00008000→Dz Dz の LSW クリア もし DC＝1 ならば nop.	111110********** 10011011xx01zzzz	—	—	—	新規

命令	動作	命令コード	特権	DC ビット	T ビット	新規
DCF PRND Sy,Dz	もし DC＝0 ならば Sy＋H'00008000→Dz Dz の LSW クリア もし DC＝1 ならば nop.	111110********** 1011101101yyzzzz	—	—	—	新規

表 3.25　スワップ命令

命令	動作	命令コード	特権	DC ビット	T ビット	新規
PSWAP Sx,Dz	Sx の LSW→Dz の MSW Sx の MSW→Dz の LSW	111110********** 10011101xx01zzzz	—	更新	更新	新規
PSWAP Sy,Dz	Sy の LSW→Dz の MSW Sy の MSW→Dz の LSW	111110********** 1011110101yyzzzz	—	更新	更新	新規
DCT PSWAP Sx,Dz	もし DC=1 ならば Sx の LSW→Dz の MSW Sx の MSW→Dz の LSW もし DC＝0 ならば nop.	111110********** 10011110xx01zzzz	—	—	—	新規
DCT PSWAP Sy,Dz	もし DC=1 ならば Sy の LSW→Dz の MSW Sy の MSW→Dz の LSW もし DC＝0 ならば nop.	111110********** 1011111001yyzzzz	—	—	—	新規
DCF PSWAP Sx,Dz	もし DC=0 ならば Sx の LSW→Dz の MSW Sx の MSW→Dz の LSW もし DC＝1 ならば nop.	111110********** 10011111xx01zzzz	—	—	—	新規
DCF PSWAP Sy,Dz	もし DC=0 ならば Sy の LSW→Dz の MSW Sy の MSW→Dz の LSW もし DC＝1 ならば nop.	111110********** 1011111101yyzzzz	—	—	—	新規

表 3.26　ローカルデータ移動命令

命令	動作	命令コード	特権	DC ビット	T ビット	新規
PLDS Dz,MACH	Dz→MACH	111110********** 1110110100 00zzzz	—	—	—	—
PLDS Dz,MACL	Dz→MACL	111110********** 1111110100 00zzzz	—	—	—	—
DCT PLDS Dz,MACH	もし DC＝1 ならば Dz→MACH もし DC＝0 ならば nop.	111110********** 1110111000 00zzzz	—	—	—	—
DCT PLDS Dz,MACL	もし DC＝1 ならば Dz→MACL もし DC＝0 ならば nop.	111110********** 1111111000 00zzzz	—	—	—	—
DCF PLDS Dz,MACH	もし DC＝0 ならば Dz→MACH もし DC＝1 ならば nop.	111110********** 1110111100 00zzzz	—	—	—	—
DCF PLDS Dz,MACL	もし DC＝0 ならば Dz→MACL もし DC＝1 ならば nop.	111110********** 1111111100 00zzzz	—	—	—	—
PSTS MACH,Dz	MACH→Dz	111110********** 1100110100 00zzzz	—	—	—	—
PSTS MACL,Dz	MACL→Dz	111110********** 1101110100 00zzzz	—	—	—	—
DCT PSTS MACH,Dz	もし DC＝1 ならば MACH→Dz もし DC＝0 ならば nop.	111110********** 1100111000 00zzzz	—	—	—	—
DCT PSTS MACL,Dz	もし DC＝1 ならば MACL→Dz もし DC＝0 ならば nop.	111110********** 1101111000 00zzzz	—	—	—	—
DCF PSTS MACH,Dz	もし DC＝0 ならば MACH→Dz もし DC＝1 ならば nop.	111110********** 1100111100 00zzzz	—	—	—	—
DCF PSTS MACL,Dz	もし DC＝0 ならば MACL→Dz もし DC＝1 ならば nop.	111110********** 1101111100 00zzzz	—	—	—	—

3.5.1 NOPX と NOPY の命令コード

DSP 演算命令と同時に並行処理されるデータ転送命令がないときは、データ転送命令に NOPX、NOPY 命令を書くかあるいは命令を省略することもできます。NOPX、NOPY 命令を書いても省略しても命令コードは同じです。NOPX と NOPY の命令コードの例を**表 3.27** に示します。

表 3.27 NOPX と NOPY の命令コードの例

命令	コード
PADD X0, Y0, A0 MOVX. W @R4+, X0 MOVY.W @R6+R9, Y0	1111100000001011 1011000100000111
PADD X0, Y0, A0 NOPX MOVY.W @R6+R9, Y0	1111100000000011 1011000100000111
PADD X0, Y0, A0 NOPX NOPY	1111100000000000 1011000100000111
PADD X0, Y0, A0 NOPX	1111100000000000 1011000100000111
PADD X0, Y0, A0	1111100000000000 1011000100000111
MOVX. W @R4+, X0 MOVY.W @R6+R9, Y0	1111000000001011
MOVX. W @R4+, X0 NOPY	1111000000001000
MOVS. W @R4+, X0	1111010010001000
NOPX MOVY.W @R6+R9, Y0	1111000000000011
MOVY.W @R6+R9, Y0	1111000000000011
NOPX NOPY	1111000000000000
NOP	0000000000001001

4. パイプライン動作

SH4AL-DSP 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。

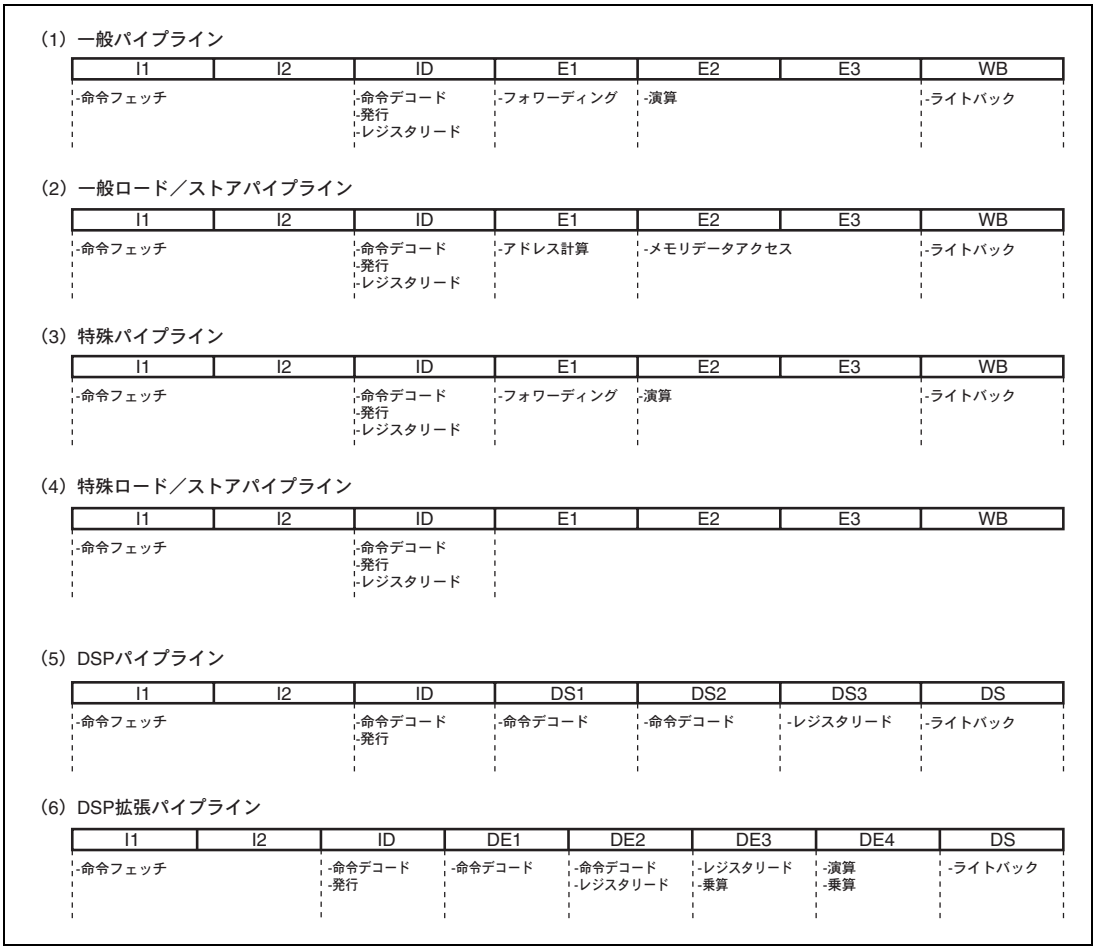


図 4.1　基本パイプライン

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
DE1 | DE2 | DE3 | DE4 | DS	DSP-EX パイプ占有
DS1 | DS2 | DS3 | DS	DSP-LS パイプ占有
ID	ID ステージをロック
└──	CPU と DSP 両方のパイプを占有

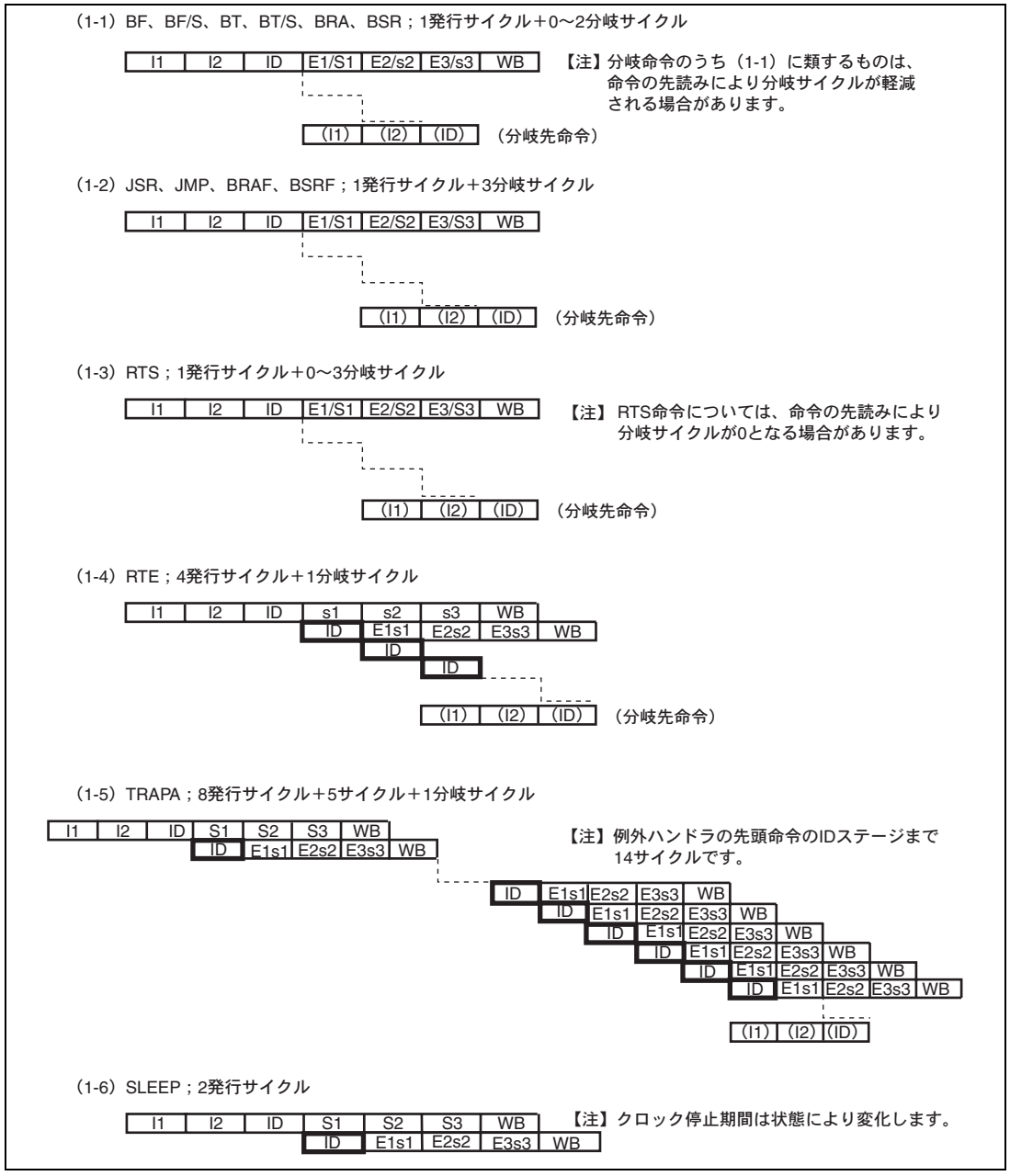


図 4.2　命令実行パターン（1）

（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
SETDMX、SETDMY、CLRDMXY

【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

（2-5）LDRS、LDRE（LSタイプ）；1発行サイクル

I1	I2	ID	s1	s2	s3	WB

（2-6）LDRC、SETRC（COタイプ）；2発行サイクル

I1	I2	ID	E1/s1	E2/s2	E3/s3	WB
			ID			

図 4.2　命令実行パターン（2）

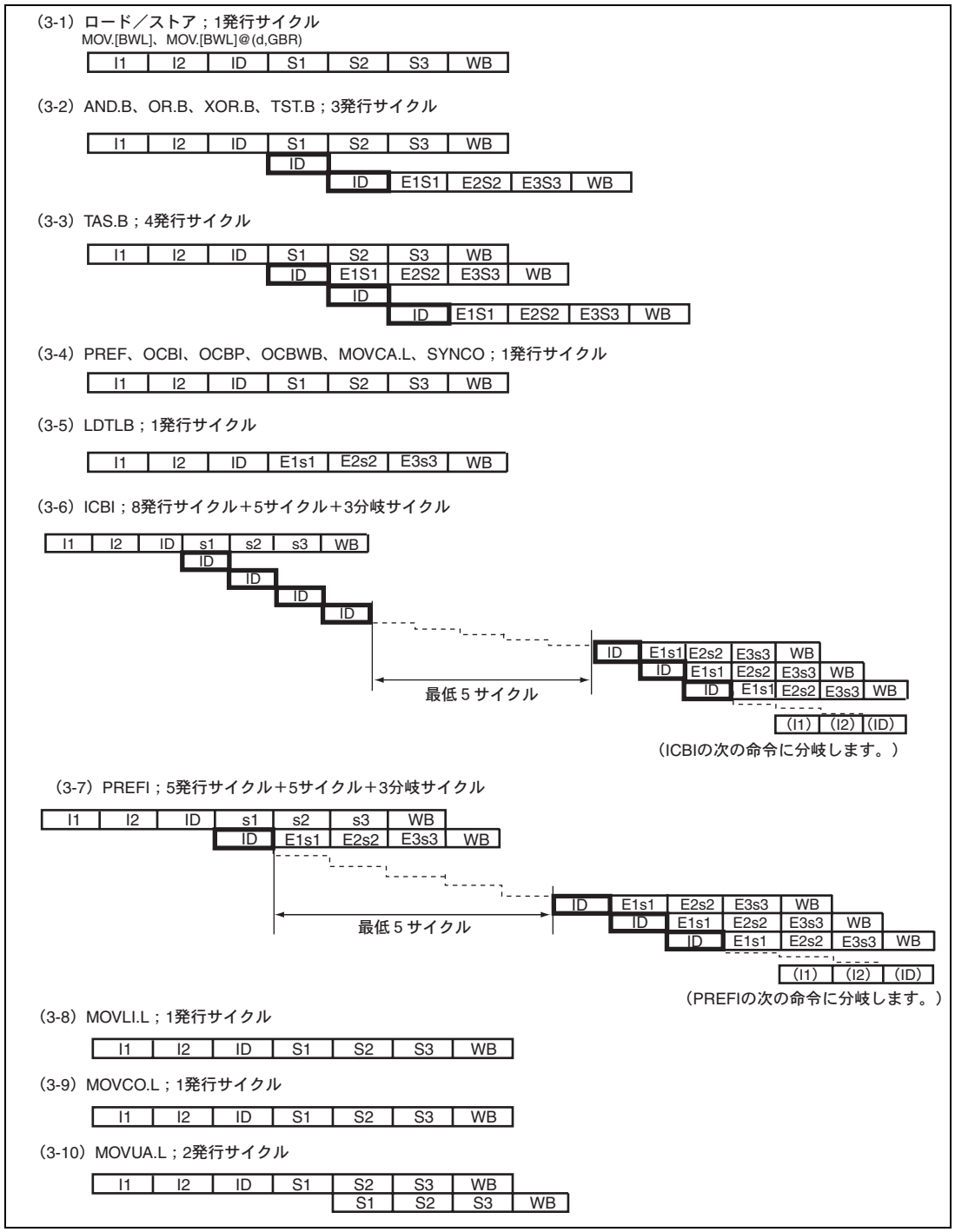


図 4.2　命令実行パターン（3）

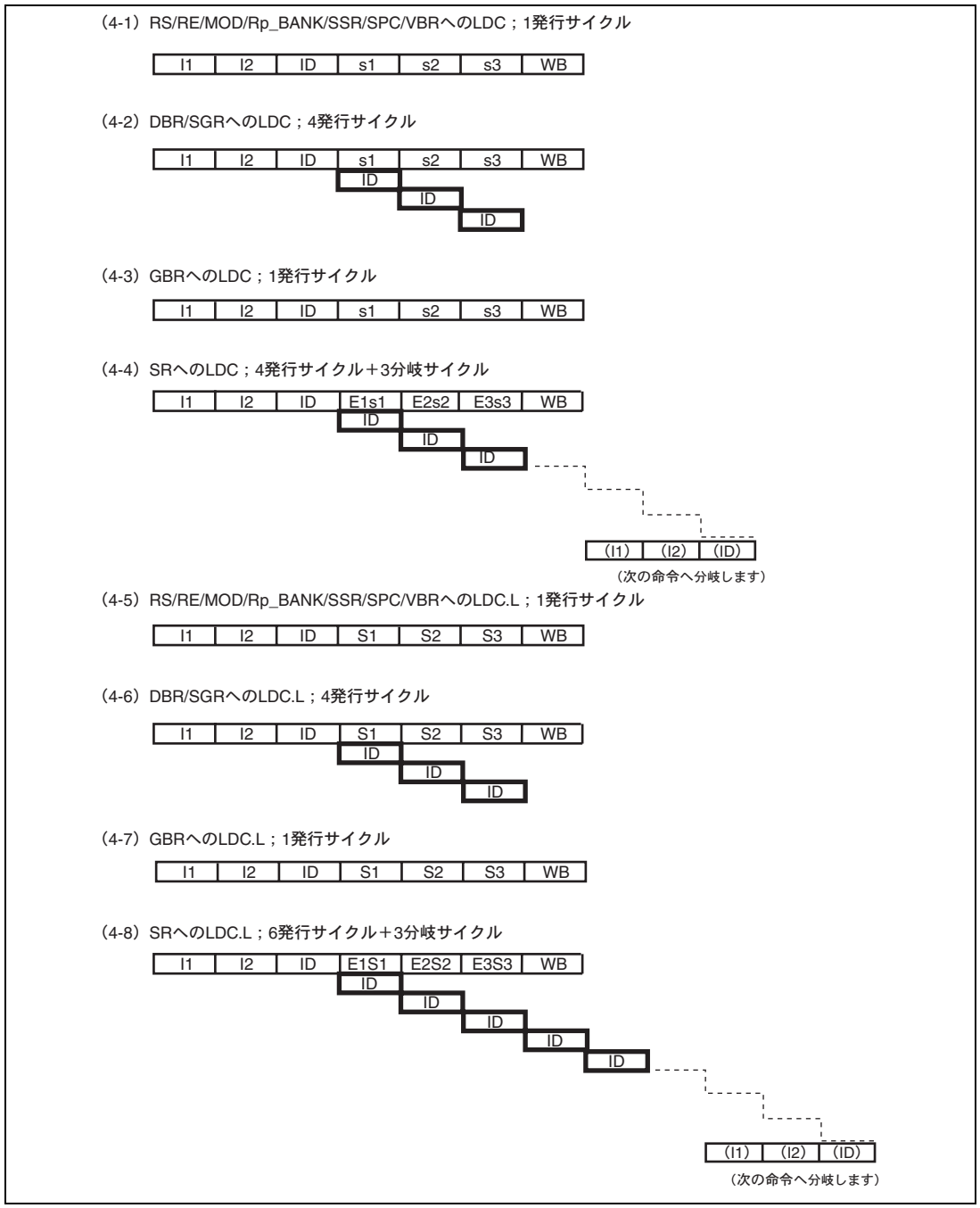


図 4.2　命令実行パターン（4）

（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）RS/RE/MODからのSTC；1発行サイクル

I1	I2	ID	s1	s2	s3	WB

（4-11）SRからのSTC；1発行サイクル

I1	I2	ID	E1s1	E2s2	E3s3	WB

（4-12）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

（4-13）RS/RE/MODからのSTC.L；1発行サイクル

I1	I2	ID	S1	S2	S3	WB

（4-14）SRからのSTC.L；1発行サイクル

I1	I2	ID	E1S1	E2S2	E3S3	WB

（4-15）PRへのLDS；1発行サイクル

I1	I2	ID	s1	s2	s3	WB

（4-16）PRへのLDS.L；1発行サイクル

I1	I2	ID	S1	S2	S3	WB

（4-17）PRからのSTS；1発行サイクル

I1	I2	ID	s1	s2	s3	WB

（4-18）PRからのSTS.L；1発行サイクル

I1	I2	ID	S1	S2	S3	WB

（4-19）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）

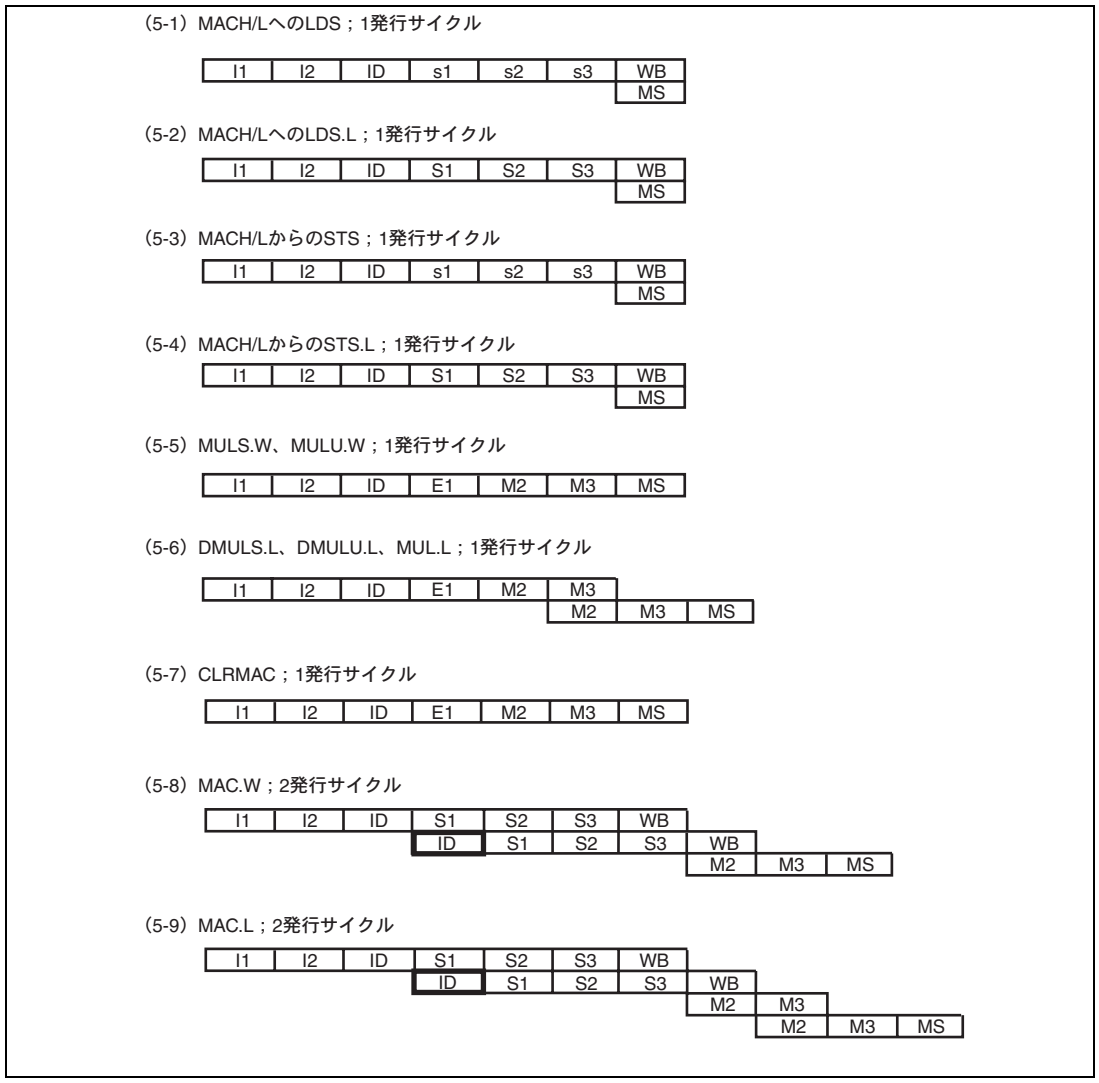


図 4.2　命令実行パターン（6）

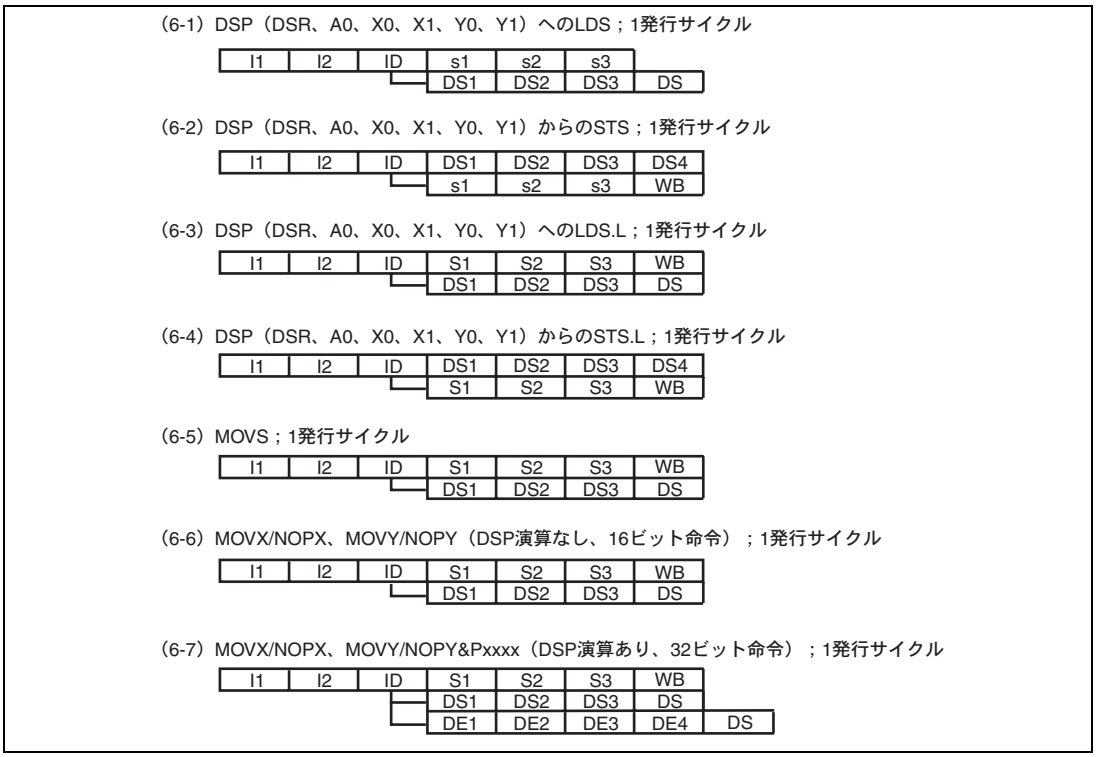


図 4.2　命令実行パターン（7）

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	ROTR	SHLR8
	ADDC	EXTS	SETDMX	SHLR16
	ADDV	EXTU	SETDMY	SUB
	AND #imm,R0	MOVT	SETS	SUBC
	AND Rm,Rn	MUL.L	SETT	SUBV
	CLRDMXY	MULS.W	SHAD	SWAP
	CLRMAC	MULU.W	SHAL	TST #imm,R0
	CLRS	NEG	SHAR	TST Rm,Rn
	CLRT	NEGC	SHLD	XOR #imm,R0
	CMP	NOT	SHLL	XOR Rm,Rn
	DIV0S	OR #imm,R0	SHLL2	XTRCT
	DIV0U	OR Rm,Rn	SHLL8	
	DIV1	ROTCL	SHLL16	
	DMUS.L	ROTCR	SHLR	
	DMULU.L	ROTL	SHLR2	
MT	MOV #imm,Rn	MOV Rm,Rn	NOP	
BR	BF	BRAF	BT	JSR
	BF/S	BSR	BT/S	RTS
	BRA	BSRF	JMP	
LS	LDC Rm,CR1	LDRE	MOVUA	STC.L CR2,@-Rn
	LDS Rm,SR1	LDRS	OCBI	STS.L SR1,@-Rn
	LDS Rm,SR2	MOV.[BWL] @adr,R	OCBP	STS SR1,Rn
	LDC.L @Rm+,CR1	MOV.[BWL] R,@adr	OCBWB	
	LDS.L @Rm+,SR1	MOVA	PREF	
	LDS.L @Rm+,SR2	MOVCA.L	STC CR2,Rn	
CO	AND.B #imm,@(R0,GBR)	LDC.L @Rm+,SGR	OR.B #imm,@(R0,GBR)	SYNCO
	ICBI	LDRC	PREFI	TAS.B
	LDC Rm,DBR	LDTLB	RTE	TRAPA
	LDC Rm,SR	MAC.L	SETRC	TST.B #imm,@(R0,GBR)
	LDC Rm,SGR	MAC.W	SLEEP	XOR.B #imm,@(R0,GBR)
	LDC.L @Rm+,DBR	MOVCO	STC SR,Rn	
	LDC.L @Rm+,SR	MOVLI	STC.L SR,@-Rn	
DSP-LS	MOVS.L	MOVX.W,	NOPX	
	MOVS.W	MOVY.W	NOPY	
DSP-CO	DSP 演算命令 Pxxx			

【記号説明】 R : Rm/Rn
@adr : アドレス
SR1 : MACH/MACL/PR
SR2 : DSR/A0/X0/X1/Y0/Y1
CR1 : RS/RE/MOD/GBR/Rp_BANK/SPC/SSR/VBR
CR2 : CR1/DBR/SGR

2 命令の同時実行は次の場合に限ります。

1. 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
2. **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
3. 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
4. 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
5. 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	CO	DSP-LS	DSP-CO
後行命令（addr+2）	EX	×	○	○	○		○	
	MT	○	○	○	○		○	
	BR	○	○	×	○		○	
	LS	○	○	○	×		×	
	CO					×	×	
	DSP-LS	○	○	○	×	×	×	
	DSP-CO							×

4.3 発行レートと実行ステート

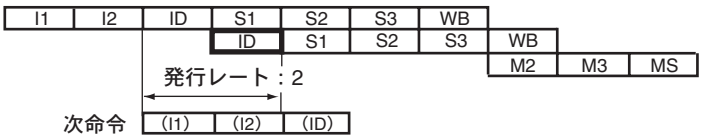
命令の発行レートと実行ステートを**表 4.4** に示します。**表 4.4** 中の命令グループは、**表 4.2** における命令の分類に対応します。また、本節に示す発行レートと実行ステートでは、パイプラインストールによるペナルティサイクルは考慮していません。

（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

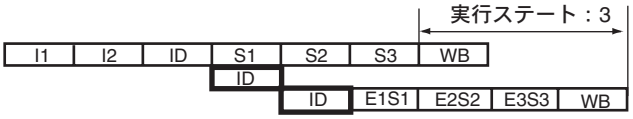


（2）実行ステート

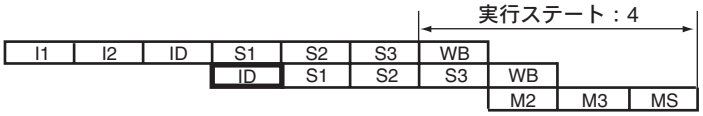
実行ステートは、命令がパイプラインを占有するサイクル数を次の基準で示します。

・CPU命令

（例）AND.B命令

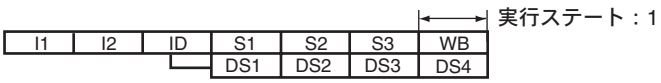


（例）MAC.W命令



・DSP命令

（例）MOVX、MOVY命令



（例）NOPX、NOPY、PADD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

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	72	MAC.L	@Rm+,@Rn+	CO	2	5	5-9
	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

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	108	SHLR	Rn	EX	1	1	2-1
	109	SHLR2	Rn	EX	1	1	2-1
	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

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	144	LDC.L	@Rm+,DBR	CO	4	4	4-6
	145	LDC.L	@Rm+,SGR	CO	4	4	4-6
	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	4-15
	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	4-16
	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	4-11
	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	4-12
	167	STC.L	SGR,@-Rn	LS	1	1	4-12
	168	STC.L	GBR,@-Rn	LS	1	1	4-12
	169	STC.L	Rp_BANK,@-Rn	LS	1	1	4-12
	170	STC.L	SR,@-Rn	CO	1	1	4-14
	171	STC.L	SSR,@-Rn	LS	1	1	4-12
	172	STC.L	SPC,@-Rn	LS	1	1	4-12
	173	STC.L	VBR,@-Rn	LS	1	1	4-12
	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	4-17
	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	4-18

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
DSP システム制御 命令	180	SETRC	#imm	CO	2	2	2-6
	181	SETRC	Rn	CO	2	2	2-6
	182	LDRS	@(disp,PC)	LS	1	1	2-5
	183	LDRE	@(disp,PC)	LS	1	1	2-5
	184	STC	MOD,Rn	LS	1	1	4-10
	185	STC	RS,Rn	LS	1	1	4-10
	186	STC	RE,Rn	LS	1	1	4-10
	187	STS	DSR,Rn	LS	1	1	6-2
	188	STS	A0,Rn	LS	1	1	6-2
	189	STS	X0,Rn	LS	1	1	6-2
	190	STS	X1,Rn	LS	1	1	6-2
	191	STS	Y0,Rn	LS	1	1	6-2
	192	STS	Y1,Rn	LS	1	1	6-2
	193	STS.L	DSR,@-Rn	LS	1	1	6-4
	194	STS.L	A0,@-Rn	LS	1	1	6-4
	195	STS.L	X0,@-Rn	LS	1	1	6-4
	196	STS.L	X1,@-Rn	LS	1	1	6-4
	197	STS.L	Y0,@-Rn	LS	1	1	6-4
	198	STS.L	Y1,@-Rn	LS	1	1	6-4
	199	STC.L	MOD,@-Rn	LS	1	1	4-13
	200	STC.L	RS,@-Rn	LS	1	1	4-13
	201	STC.L	RE,@-Rn	LS	1	1	4-13
	202	LDS.L	@Rn+,DSR	LS	1	1	6-3
	203	LDS.L	@Rn+,A0	LS	1	1	6-3
	204	LDS.L	@Rn+,X0	LS	1	1	6-3
	205	LDS.L	@Rn+,X1	LS	1	1	6-3
	206	LDS.L	@Rn+,Y0	LS	1	1	6-3
	207	LDS.L	@Rn+,Y1	LS	1	1	6-3
	208	LDC.L	@Rn+,MOD	LS	1	1	4-5
	209	LDC.L	@Rn+,RS	LS	1	1	4-5
	210	LDC.L	@Rn+,RE	LS	1	1	4-5
	211	LDS	Rn,DSR	LS	1	1	6-1
	212	LDS	Rn,A0	LS	1	1	6-1
	213	LDS	Rn,X0	LS	1	1	6-1
	214	LDS	Rn,X1	LS	1	1	6-1
	215	LDS	Rn,Y0	LS	1	1	6-1

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
DSP システム制御 命令	216	LDS	Rn,Y1	LS	1	1	6-1
	217	LDC	Rn,MOD	LS	1	1	4-1
	218	LDC	Rn,RS	LS	1	1	4-1
	219	LDC	Rn,RE	LS	1	1	4-1
	220	LDRC	#imm	CO	2	2	2-6
	221	LDRC	Rn	CO	2	2	2-6
	222	SETDMX		EX	1	1	2-1
	223	SETDMY		EX	1	1	2-1
	224	CLRMXY		EX	1	1	2-1
ダブルデータ 転送	225	NOPX		DSP-LS	1	1	6-6
	226	MOVX.W	@Ax,Dx	DSP-LS	1	1	6-6
	227	MOVX.W	@Ax+,Dx	DSP-LS	1	1	6-6
	228	MOVX.W	@Ax+Ix,Dx	DSP-LS	1	1	6-6
	229	MOVX.W	Da,@Ax	DSP-LS	1	1	6-6
	230	MOVX.W	Da,@Ax+	DSP-LS	1	1	6-6
	231	MOVX.W	Da,@Ax+Ix	DSP-LS	1	1	6-6
	232	MOVX.W	@Axy,Dxy	DSP-LS	1	1	6-6
	233	MOVX.W	@Axy+,Dxy	DSP-LS	1	1	6-6
	234	MOVX.W	@Axy+Ix,Dxy	DSP-LS	1	1	6-6
	235	MOVX.W	Dax,@Axy	DSP-LS	1	1	6-6
	236	MOVX.W	Dax,@Axy+	DSP-LS	1	1	6-6
	237	MOVX.W	Dax,@Axy+Ix	DSP-LS	1	1	6-6
	238	MOVX.L	@Axy,Dxy	DSP-LS	1	1	6-6
	239	MOVX.L	@Axy+,Dxy	DSP-LS	1	1	6-6
	240	MOVX.L	@Axy+Ix,Dxy	DSP-LS	1	1	6-6
	241	MOVX.L	Dax,@Axy	DSP-LS	1	1	6-6
	242	MOVX.L	Dax,@Axy+	DSP-LS	1	1	6-6
	243	MOVX.L	Dax,@Axy+Ix	DSP-LS	1	1	6-6
	244	NOPY		DSP-LS	1	1	6-6
	245	MOVY.W	@Ay,Dy	DSP-LS	1	1	6-6
	246	MOVY.W	@Ay+,Dy	DSP-LS	1	1	6-6
	247	MOVY.W	@Ay+Iy,Dy	DSP-LS	1	1	6-6
	248	MOVY.W	Da,@Ay	DSP-LS	1	1	6-6
	249	MOVY.W	Da,@Ay+	DSP-LS	1	1	6-6
	250	MOVY.W	Da,@Ay+Iy	DSP-LS	1	1	6-6
	251	MOVY.W	@Ayx,Dyx	DSP-LS	1	1	6-6

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
ダブルデータ 転送	252	MOVY.W	@Ayx+,Dyx	DSP-LS	1	1	6-6
	253	MOVY.W	@Ayx+Iy,Dyx	DSP-LS	1	1	6-6
	254	MOVY.W	Day,@Ayx	DSP-LS	1	1	6-6
	255	MOVY.W	Day,@Ayx+	DSP-LS	1	1	6-6
	256	MOVY.W	Day,@Ayx+Iy	DSP-LS	1	1	6-6
	257	MOVY.W	@Ayx,Dyx	DSP-LS	1	1	6-6
	258	MOVY.L	@Ayx+,Dyx	DSP-LS	1	1	6-6
	259	MOVY.L	@Ayx+Iy,Dyx	DSP-LS	1	1	6-6
	260	MOVY.L	Day,@Ayx	DSP-LS	1	1	6-6
	261	MOVY.L	Day,@Ayx+	DSP-LS	1	1	6-6
	262	MOVY.L	Day,@Ayx+Iy	DSP-LS	1	1	6-6
シングルデータ 転送	263	MOVS.W	@-As,Ds	DSP-LS	1	1	6-5
	264	MOVS.W	@As,Ds	DSP-LS	1	1	6-5
	265	MOVS.W	@As+,Ds	DSP-LS	1	1	6-5
	266	MOVS.W	@As+Is,Ds	DSP-LS	1	1	6-5
	267	MOVS.W	Ds,@-As	DSP-LS	1	1	6-5
	268	MOVS.W	Ds,@As	DSP-LS	1	1	6-5
	269	MOVS.W	Ds,@As+	DSP-LS	1	1	6-5
	270	MOVS.W	Ds,@As+Is	DSP-LS	1	1	6-5
	271	MOVS.L	@-As,Ds	DSP-LS	1	1	6-5
	272	MOVS.L	@As,Ds	DSP-LS	1	1	6-5
	273	MOVS.L	@As+,Ds	DSP-LS	1	1	6-5
	274	MOVS.L	@As+Is,Ds	DSP-LS	1	1	6-5
	275	MOVS.L	Ds,@-As	DSP-LS	1	1	6-5
	276	MOVS.L	Ds,@As	DSP-LS	1	1	6-5
	277	MOVS.L	Ds,@As+	DSP-LS	1	1	6-5
	278	MOVS.L	Ds,@As+Is	DSP-LS	1	1	6-5
DSP 演算	279	PABS	Sx,Dz	DSP-CO	1	1	6-7
	280	DCT PABS	Sx,Dz	DSP-CO	1	1	6-7
	281	DCF PABS	Sx,Dz	DSP-CO	1	1	6-7
	282	PABS	Sy,Dz	DSP-CO	1	1	6-7
	283	DCT PABS	Sy,Dz	DSP-CO	1	1	6-7
	284	DCF PABS	Sy,Dz	DSP-CO	1	1	6-7
	285	PADD	Sx,Sy,Du	DSP-CO	1	1	6-7
		PMULS	Se,Sf,Dg				
	286	PADD	Sx,Sy,Dz	DSP-CO	1	1	6-7

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
DSP 演算	287	DCT PADD	Sx,Sy,Dz	DSP-CO	1	1	6-7
	288	DCF PADD	Sx,Sy,Dz	DSP-CO	1	1	6-7
	289	PADDC	Sx,Sy,Dz	DSP-CO	1	1	6-7
	290	PAND	Sx,Sy,Dz	DSP-CO	1	1	6-7
	291	DCT PAND	Sx,Sy,Dz	DSP-CO	1	1	6-7
	292	DCF PAND	Sx,Sy,Dz	DSP-CO	1	1	6-7
	293	PCLR	Du	DSP-CO	1	1	6-7
		PMULS	Se,Sf,Dg				
	294	PCLR	Dz	DSP-CO	1	1	6-7
	295	DCT PCLR	Dz	DSP-CO	1	1	6-7
	296	DCF PCLR	Dz	DSP-CO	1	1	6-7
	297	PCMP	Sx,Sy	DSP-CO	1	1	6-7
	298	PCOPY	Sx,Dz	DSP-CO	1	1	6-7
	299	DCT PCOPY	Sx,Dz	DSP-CO	1	1	6-7
	300	DCF PCOPY	Sx,Dz	DSP-CO	1	1	6-7
	301	PCOPY	Sy,Dz	DSP-CO	1	1	6-7
	302	DCT PCOPY	Sy,Dz	DSP-CO	1	1	6-7
	303	DCF PCOPY	Sy,Dz	DSP-CO	1	1	6-7
	304	PDEC	Sx,Dz	DSP-CO	1	1	6-7
	305	DCT PDEC	Sx,Dz	DSP-CO	1	1	6-7
	306	DCT PDEC	Sx,Dz	DSP-CO	1	1	6-7
	307	PDEC	Sy,Dz	DSP-CO	1	1	6-7
	308	DCT PDEC	Sy,Dz	DSP-CO	1	1	6-7
	309	DCF PDEC	Sy,Dz	DSP-CO	1	1	6-7
	310	PDMSB	Sx,Dz	DSP-CO	1	1	6-7
	311	DCT PDMSB	Sx,Dz	DSP-CO	1	1	6-7
	312	DCF PDMSB	Sx,Dz	DSP-CO	1	1	6-7
	313	PDMSB	Sy,Dz	DSP-CO	1	1	6-7
	314	DCT PDMSB	Sy,Dz	DSP-CO	1	1	6-7
	315	DCF PDMSB	Sy,Dz	DSP-CO	1	1	6-7
	316	PINC	Sx,Dz	DSP-CO	1	1	6-7
	317	DCT PINC	Sx,Dz	DSP-CO	1	1	6-7
	318	DCF PINC	Sx,Dz	DSP-CO	1	1	6-7
	319	PINC	Sy,Dz	DSP-CO	1	1	6-7
	320	DCT PINC	Sy,Dz	DSP-CO	1	1	6-7
	321	DCF PINC	Sy,Dz	DSP-CO	1	1	6-7

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
DSP 演算	322	PLDS	Dz,MACH	DSP-CO	1	1	6-7
	323	DCT PLDS	Dz,MACH	DSP-CO	1	1	6-7
	324	DCF PLDS	Dz,MACH	DSP-CO	1	1	6-7
	325	PLDS	Dz,MACL	DSP-CO	1	1	6-7
	326	DCT PLDS	Dz,MACL	DSP-CO	1	1	6-7
	327	DCF PLDS	Dz,MACL	DSP-CO	1	1	6-7
	328	PMULS	Se,Sf,Dg	DSP-CO	1	1	6-7
	329	PNEG	Sx,Dz	DSP-CO	1	1	6-7
	330	DCT PNEG	Sx,Dz	DSP-CO	1	1	6-7
	331	DCF PNEG	Sx,Dz	DSP-CO	1	1	6-7
	332	PNEG	Sy,Dz	DSP-CO	1	1	6-7
	333	DCT PNEG	Sy,Dz	DSP-CO	1	1	6-7
	335	DCF PNEG	Sy,Dz	DSP-CO	1	1	6-7
	336	POR	Sx,Sy,Dz	DSP-CO	1	1	6-7
	337	DCT POR	Sx,Sy,Dz	DSP-CO	1	1	6-7
	338	DCF POR	Sx,Sy,Dz	DSP-CO	1	1	6-7
	339	PRND	Sx,Dz	DSP-CO	1	1	6-7
	340	DCT PRND	Sx,Dz	DSP-CO	1	1	6-7
	341	DCF PRND	Sx,Dz	DSP-CO	1	1	6-7
	342	PRND	Sy,Dz	DSP-CO	1	1	6-7
	343	DCT PRND	Sy,Dz	DSP-CO	1	1	6-7
	344	DCF PRND	Sy,Dz	DSP-CO	1	1	6-7
	345	PSHA	Sx,Sy,Dz	DSP-CO	1	1	6-7
	346	DCT PSHA	Sx,Sy,Dz	DSP-CO	1	1	6-7
	347	DCF SHA	Sx,Sy,Dz	DSP-CO	1	1	6-7
	348	PSHA	#imm,Dz	DSP-CO	1	1	6-7
	349	PSHL	Sx,Sy,Dz	DSP-CO	1	1	6-7
	350	DCT PSHL	Sx,Sy,Dz	DSP-CO	1	1	6-7
	351	DCF PSHL	Sx,Sy,Dz	DSP-CO	1	1	6-7
	352	PSHL	#imm,Dz	DSP-CO	1	1	6-7
	353	PSTS	MACH,Dz	DSP-CO	1	1	6-7
	354	DCT PSTS	MACH,Dz	DSP-CO	1	1	6-7
	355	DCF PSTS	MACH,Dz	DSP-CO	1	1	6-7
	356	PSTS	MACL,Dz	DSP-CO	1	1	6-7
	357	DCT PSTS	MACL,Dz	DSP-CO	1	1	6-7
	358	DCF PSTS	MACL,Dz	DSP-CO	1	1	6-7

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
DSP 演算	359	PSUB	Sx,Sy,Du	DSP-CO	1	1	6-7
		PMULS	Se,Sf,Dg				
	360	PSUB	Sx,Sy,Dz	DSP-CO	1	1	6-7
	361	DCT PSUB	Sx,Sy,Dz	DSP-CO	1	1	6-7
	362	DCF PSUB	Sx,Sy,Dz	DSP-CO	1	1	6-7
	363	PSUB	Sy,Sx,Dz	DSP-CO	1	1	6-7
	364	DCT PSUB	Sy,Sx,Dz	DSP-CO	1	1	6-7
	365	DCF PSUB	Sy,Sx,Dz	DSP-CO	1	1	6-7
	366	PSUBC	Sx,Sy,Dz	DSP-CO	1	1	6-7
	367	PSWAP	Sx,Dz	DSP-CO	1	1	6-7
	368	DCT PSWAP	Sx,Dz	DSP-CO	1	1	6-7
	369	DCF PSWAP	Sx,Dz	DSP-CO	1	1	6-7
	370	PSWAP	Sy,Dz	DSP-CO	1	1	6-7
	371	DCT PSWAP	Sy,Dz	DSP-CO	1	1	6-7
	372	DCF PSWAP	Sy,Dz	DSP-CO	1	1	6-7
	373	PXOR	Sx,Sy,Dz	DSP-CO	1	1	6-7
	374	DCT PXOR	Sy,Sx,Dz	DSP-CO	1	1	6-7
	375	DCF PXOR	Sx,Sy,Dz	DSP-CO	1	1	6-7

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。

SH4AL-DSP 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、リセットと一般例外事象による 12 ビットの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、割り込み要求による 14 ビットの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. 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。
6. 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*1	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	2	（VBR）	H'400	H'040
		命令 TLB 保護違反例外*2	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
		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	2	6	（VBR）	H'400	H'040
		データ TLB ミス例外（書き込み）*2	2	6	（VBR）	H'400	H'060
		データ TLB 保護違反例外（読み出し）*2	2	7	（VBR）	H'100	H'0A0
		データ TLB 保護違反例外（書き込み）*2	2	7	（VBR）	H'100	H'0C0
		初期ページ書き込み例外*2	2	8	（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*1	2	9	（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。

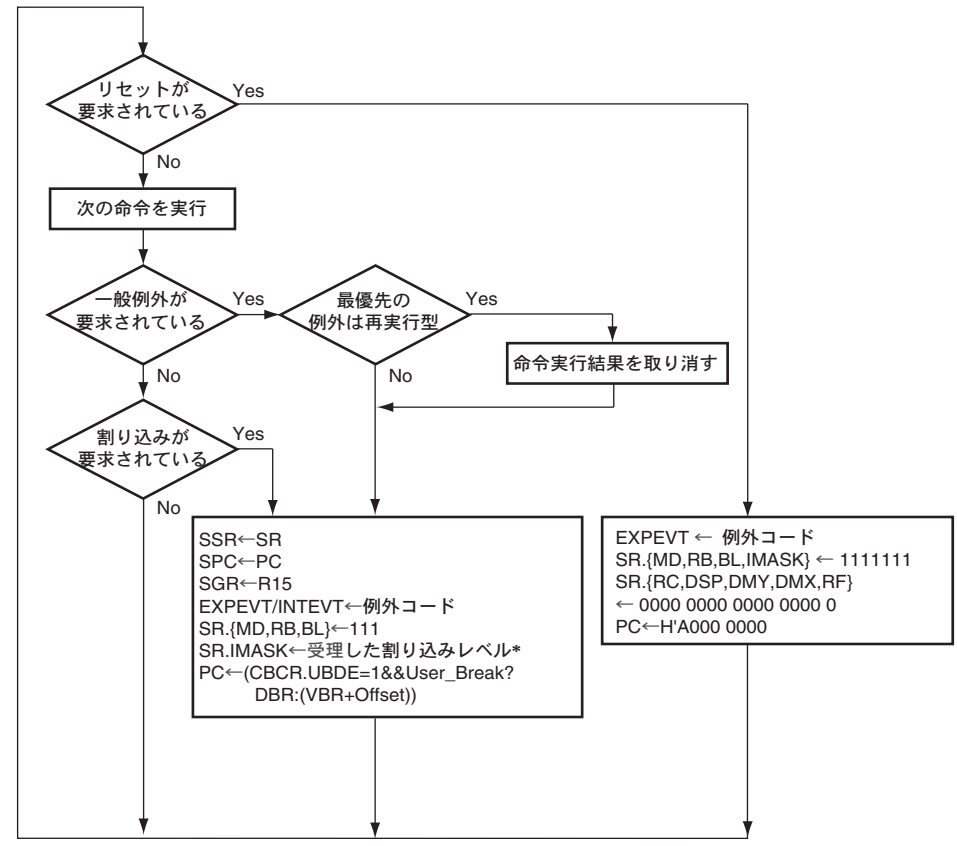
【注】 *1　CBCR.UBDE=1 のとき PC=DBR。その他は PC=VBR+H'100

*2　これらの例外コードは、メモリマネジメントユニット（MMU）を使用する場合に有効です。

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**「5.6.6 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。



【注】* 最優先の例外が割り込みのとき。なおIMASKを更新するかしないかをソフトウェアで設定可能です。

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、無条件トラップ例外の 3 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。

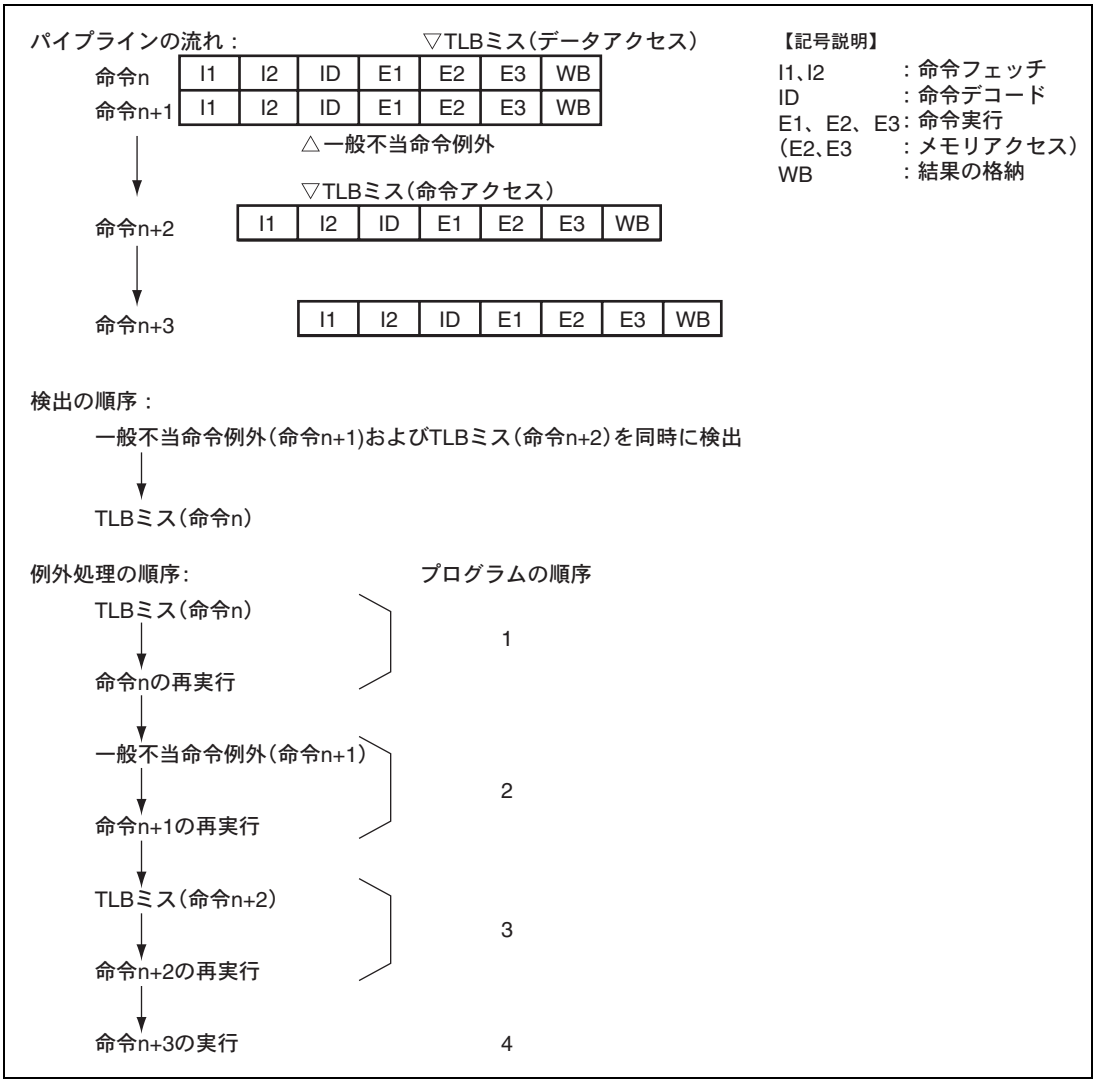


図 5.2 一般例外の受け付け順序の例

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
  - 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'A500 0000～H'A5FF 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　X/Yメモリ」**を参照してください。
  - X/Yメモリのアクセスが、Xバス保護制御レジスタ外またはYバス保護制御レジスタ外に発生した場合

- **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'A500 0000～H'A5FF 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 X/Y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」**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、MOD、RS、REをアクセスする命令を除きます。また、LDC、STCでSRをアクセスする命令はSR.DSPビットが0のときのみ特権命令です。

  - SRレジスタのDSPビットが0のときに、遅延スロット以外にあるDSP命令を実行

  - SRレジスタのDSPビットが0のときに、遅延スロット以外にあるDSPをサポートするCPU命令を実行

    DSPをサポートするCPU命令：MOD、RE、RSに対するLDC/STC

    A0、X0、X1、Y0、Y1に対するLDS/STS

    SETRC Rm、SETRC #imm、LDRS @(disp,PC)、LDRE @(disp,PC)、LDRC Rm、LDRC #imm、SETDMX、SETDMY、CLRDMXY

- **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、MOD、RS、REをアクセスする命令を除きます。また、LDC、STCでSRをアクセスする命令はSR.DSPビットが0のときのみ特権命令です。

  - 遅延スロット内のPC相対MOV命令、MOVA命令、LDRS命令、LDRE命令をデコード

  - SRレジスタのDSPビットが0のときに、遅延スロットにあるDSP命令を実行

  - SRレジスタのDSPビットが0のときに、遅延スロットにあるDSPをサポートするCPU命令を実行

    DSPをサポートするCPU命令：MOD、RE、RSに対するLDC/STC

    A0、X0、X1、Y0、Y1に対するLDS/STS

    SETRC Rm、SETRC #imm、LDRS @(disp,PC)、LDRE @(disp,PC)、LDRC Rm、LDRC #imm、SETDMX、SETDMY、CLRDMXY

- **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）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);
}
```

5.6.3 互換リピート制御中の例外処理

(1) 互換リピート制御中の例外処理における制限事項

従来の SH3-DSP では、互換リピート制御中に発生する例外要求に対する扱いや例外を受け付けた際の処理は、通常の状態とは異なり、場合によっては例外を受け付けなかったり、受け付けても正しく復帰できないというような特殊仕様がありましたが、SH4AL-DSP では、互換リピート制御中であっても通常どおり例外を受け付けることができます。

(2) 互換リピート制御中の禁止命令

従来の SH3-DSP ではリピート検出命令の次の命令からリピート最終命令の間に、以下に示す命令を配置できませんでしたが、SH4AL-DSP ではさらにリピート検出命令に以下に示す命令と遅延分岐命令の遅延スロットを配置できなくなります。また従来 SH3-DSP 互換リピート制御では、リピート検出命令の次の命令からリピート最終命令の間に、以下に示す命令を配置すると不当命令例外を発生しましたが、SH4AL-DSP では、リピート最終命令に以下に示す命令を配置する場合のみ不当命令例外を発生し、それ以外の場合は不当命令例外を発生しません。

- 分岐命令

  BRA、BSR、BT、BF、BT/S、BF/S、BSRF、RTS、BRAF、RTE、JSR、JMP

- リピート制御命令

  SETRC、LDRS、LDRE、LDRC

- SR、RS、REに対するロード命令

  LDC Rn,SR、LDC.L @Rn+,SR、LDC Rn,RE、LDC.L @Rn+,RE、LDC Rn,RS、LDC.L @Rn+,RS

(3) リピート検出命令の次命令以降への分岐および例外受理に関する制限

従来の SH3-DSP では、リピート検出命令の次命令以降に分岐した場合はリピートループが認識されませんでしたが、SH4AL-DSP ではリピートループが認識されることがあります。ただしリピート検出命令の次命令からリピート最終命令を分岐先に指定することは禁止します。また従来の SH3-DSP では例外ルーチンからの復帰も本制限に含まれていましたが、SH4AL-DSP では例外ルーチンの復帰は本制限に含まれません。

5.6.4 拡張リピート制御中の例外処理

(1) リピート最終命令での不当命令

リピート最終命令として次の命令を配置すると不当命令例外を発生します。

- 遅延分岐命令

  BRA、BSR、BT/S、BF/S、BSRF、RTS、BRAF、RTE、JSR、JMP

- リピート制御命令

  SETRC、LDRS、LDRE、LDRC

- SR、RS、REに対するロード命令

  LDC Rn,SR、LDC.L @Rn+,SR、LDC Rn,RE、LDC.L @Rn+,RE、LDC Rn,RS、LDC.L @Rn+,RS

【注】　非遅延の分岐命令（BT、BF）は、最終命令として配置してもかまいません。また、遅延分岐命令の遅延スロットがリピート最終命令となってもかまいません。これらの場合、分岐したときも分岐しなかったときも RC[11:0]の値は 1 減じられます。分岐しなかったときはリピート開始命令へ、分岐したときには分岐先へ制御が移行します。

5.6.5　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;
}
```

5.6.6 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付は行われません。

(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. DSP ユニット

6.1 概要

SH4AL-DSP では、DSP ユニットおよび DSP ユニットに直結された X/Y メモリを内蔵しており、それらを制御する拡張命令セットが提供されています。拡張される命令セットは、次の 4 つのグループに分けられます（**図 6.1**）。

（1） CPU ユニット用のシステム制御命令

DSP 機能が有効になると CPU ユニット用のシステム制御命令として以下の命令が利用できるようになります。

- リピートループを制御するための命令や、リピートループ制御用のコントロールレジスタに対するアクセス命令が追加されます。ゼロオーバヘッドリピート制御機構を使用することによりループ構造のプログラムを効率的に実行することができるようになります。本機能に関しては、**「6.3 CPU拡張命令」**で詳しく説明します。
- モジュロアドレッシングを制御する命令、およびコントロールレジスタをアクセスする命令が追加されます。循環構造を持つデータ構造にアクセスできる機能をモジュロアドレッシングと呼びます。これらの命令については、**「6.4 DSPデータ転送命令」**で詳しく説明します。
- DSPユニットのレジスタに対するアクセス命令が追加されます。DSPユニットの幾つかのレジスタをCPUユニットのシステムレジスタであるかのように操作することが可能になります。これらの命令については、**「6.4 DSPデータ転送命令」**で詳しく説明します。

【注】 **SH4AL-DSP では、SETRC 命令を用いたリピート制御（互換リピート制御）を LDRC 命令を用いた拡張リピート制御でエミュレーションしています。このため互換リピート制御中に RS レジスタ、RE レジスタ、SR レジスタの RF ビットの値が内部状態に応じて変化します。この仕様は従来の SH3-DSP シリーズの互換リピート制御と異なっていますので、互換リピートを使用する際はリピート制御マクロ（REPEAT）を用いるか、SETRC 命令により 1 以上のリピート回数を設定する前には必ず LDRS および LDRE 命令を実行するようにしてください。なお互換リピート制御には幾つか制約事項が存在するため、リピート制御を使用する場合は LDRC 命令を用いた拡張リピートの使用を強く推奨します。**

（2） DSP ユニットのレジスタと内蔵 X/Y メモリ間のデータ転送命令

DSP ユニットのレジスタと内蔵 X/Y メモリ間のデータ転送命令は、ダブルデータ転送命令とも呼ばれます。このグループの命令のコード長は、CPU 命令と同様に 16 ビットです。DSP ユニットと DSP ユニットに直結された内蔵 X/Y メモリのデータ転送を行います。このグループの命令は、DSP ユニット用の演算命令と組み合わせて記述することが可能です。このグループの命令については、**「6.4 DSP データ転送命令」**で詳しく説明します。

(3) DSP ユニットのレジスタと全論理アドレス空間の間のデータ転送命令

DSP ユニットのレジスタと全論理アドレス空間の間のデータ転送命令は、シングルデータ転送命令とも呼ばれます。このグループの命令のコード長は、CPU 命令同様に 16 ビットです。DSP ユニットと全論理アドレス空間の間でデータ転送をおこないます。このグループの命令については、**「6.4 DSP データ転送命令」**で詳しく説明します。

(4) DSP ユニット用の演算命令

DSP ユニット用の演算命令は、DSP データ演算命令とも呼ばれます。この命令は、DSP ユニットを用いたデジタル信号処理演算を高速に実行するために用意されています。この命令のコード長は、32 ビットです。DSP データ演算命令のフィールドは、A フィールドと B フィールドに分かれています。A フィールドにはダブルデータ転送命令の機能を記述することができ、B フィールドには ALU 演算命令、および乗算命令を記述することができます。記述されたこれらの命令は並列に実行され、同時に 4 つの処理（ALU 演算、乗算、および 2 つのデータ転送）を実行することができます。

このグループの命令については、**「6.5 DSP データ演算命令」**で詳しく説明します。

【注】 1. **32 ビット命令コードは、16 ビットの命令コードが 2 個連続したものとして扱われます。このため、32 ビット命令もワード境界から配置することができます。32 ビットの命令コードは、メモリ上にワードサイズ単位で、アドレス 2n、2n+2 の順番に格納してください。**

2. **リトルエンディアンの場合でも、命令コードの上位ワードと下位ワードがそれぞれワード単位でアクセスされるものとして、メモリ上に格納してください。**

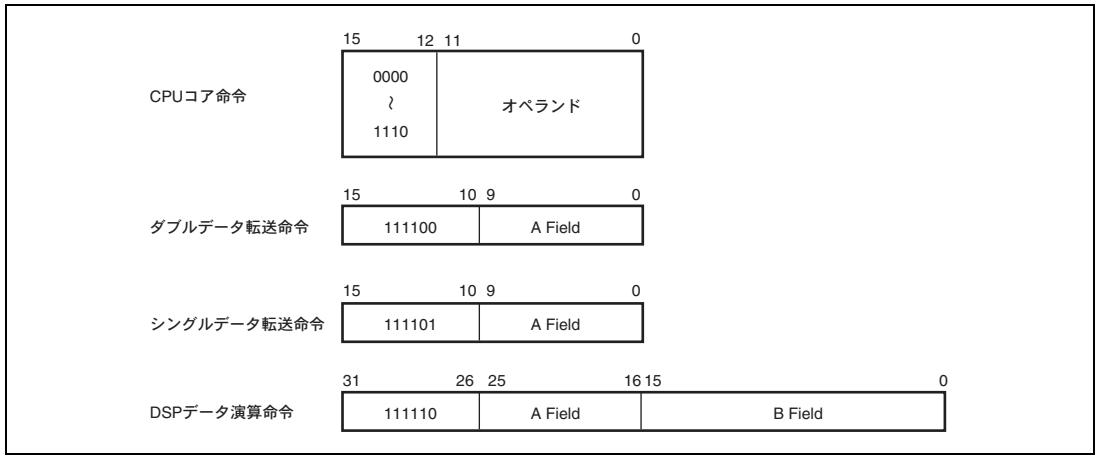


図 6.1 DSP 命令の命令形式

6.2 DSP モードのリソース

6.2.1 処理モード

CPU の処理モードは、ステータスレジスタ（SR）のモードビット（MD）および DSP ビット（DSP）により、次の表のように拡張されます。

表 6.1 処理モード

MD	DSP	処理モード	説明	
			特権保護されたリソースのアクセスや特権命令の実行	DSP 機能
0	0	ユーザモード	不可	無効
0	1	ユーザ DSP モード	不可	有効
1	0	特権モード	可能	無効
1	1	特権 DSP モード	可能	有効

このように、DSP ビットによる DSP 機能の制御は、MD ビットによる制御と独立に作用します。ただし、DSP ビットは特権モードでのみ値の変更が可能であり、DSP モードの変更を行うには特権モードや特権 DSP モードへの遷移が必要になります。

6.2.2 DSP モードのメモリマップ

DSP モードのときは、論理アドレス空間の P2 領域の一部がユーザ DSP モードでもアクセス可能になります。ユーザ DSP モードでアクセスするときは、この領域を Uxy 領域と呼びます。X/Y メモリは、この領域に配置され、ユーザ DSP モードでもアクセスが可能です。

表 6.2 論理アドレス空間

アドレス範囲	名称	保護	説明
H'A5000000 – H'A5FFFFFF	P2/Uxy	特権または DSP	16M バイト物理空間、キャッシング不可、アドレス変換不可 特権モード、特権 DSP モードおよびユーザ DSP モードでアクセス可能。

6.2.3 CPU のレジスタセット

DSP モードでは、CPU ユニットのステータスレジスタ（SR）に制御ビットが拡張され、リピートスタートレジスタ（RS）、リピートエンドレジスタ（RE）、およびモジュロレジスタ（MOD）の 3 つのコントロールレジスタが拡張されます。

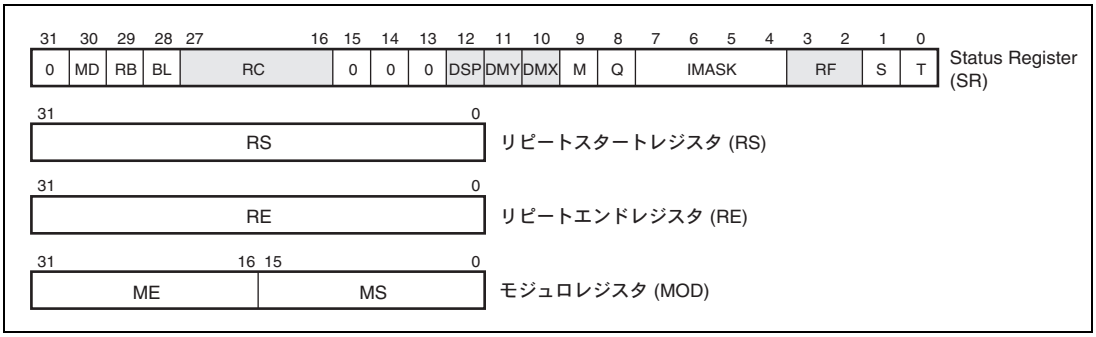


図 6.2 DSP モードでの CPU レジスタ

(1) ステータスレジスタ（SR）の拡張

DSP モードでは、以下に示す制御ビットが拡張されます。これらのビットを DSP 拡張ビットと呼びます。DSP 拡張ビットは、DSP モードでのみ有効です。

表 6.3 ステータスレジスタの拡張

ビット	ビット名	初期値	R/W	説　　明
31～28	—	—	—	拡張前の部分については、「第 2 章　プログラミングモデル」を参照してください。
27～16	RC	すべて 0	R/W	リピートカウンタ リピートカウンタは、リピート命令で制御されるリピートループの、残り実行回数を保持しています。このビットは、特権モード、特権 DSP モード、ユーザ DSP モードで更新可能です。リセット状態に遷移することにより 0 に初期化されます。例外処理状態に遷移しても値は変化しません。
15～13	—	—	—	拡張前の部分については、「第 2 章　プログラミングモデル」を参照してください。
12	DSP	0	R/W	DSP ビット DSP ビットは、DSP 機能の有効および無効を制御します。このビットに 1 を書くことで DSP 機能が有効になります。このビットは特権モードおよび特権 DSP モードでのみ更新可能で、ユーザ DSP モードでは更新できません。リセット状態に遷移することにより 0 に初期化されます。例外処理状態に遷移しても値は変化しません。
11 10	DMY DMX	0 0	R/W R/W	モジュロ制御ビット モジュロ制御ビットは、X/Y メモリへのアクセス命令でのモジュロアドレッシングの有効・無効を制御します。これらのビットは特権モード、特権 DSP モード、ユーザ DSP モードで更新可能です。リセット状態に遷移することにより 0 に初期化されます。例外処理状態に遷移しても値は変化しません。

ビット	ビット名	初期値	R/W	説　　明
9～4	—	—	—	拡張前の部分については、「第 2 章　プログラミングモデル」を参照してください。
3、2	RF	0	R/W	リピートフラグビット リピートフラグビットは、リピート制御命令によって使用されます。これらのビットは、特権モード、特権 DSP モード、およびユーザ DSP モードで更新可能です。リセット状態に遷移することにより、0 に初期化されます。例外処理状態に遷移しても値は変化しません。
1～0	—	—	—	拡張前の部分については、「第 2 章　プログラミングモデル」を参照してください。

(2)　リピートスタートレジスタ（RS）

リピートスタートレジスタは、リピート機能で制御されるリピートモジュールの先頭の命令アドレスを示します。リピートスタートレジスタは、DSP モードでアクセスできます。リセット状態に遷移したときの初期値は、不定です。例外処理状態に遷移しても値は変化しません。

また SH4AL-DSP では、SETRC 命令を用いたリピート制御（互換リピート制御）を LDRC 命令を用いた拡張リピート制御でエミュレーションしています。このため互換リピート制御中に RS レジスタの値が内部状態に応じて変化します。

(3)　リピートエンドレジスタ（RE）

リピートエンドレジスタには、リピートモジュールの最終命令の実行を検出するためのアドレスが格納されます。リピートエンドレジスタは、DSP モードでのみアクセスできます。リセット状態に遷移することにより、0 に初期化されます。例外処理状態に遷移しても値は変化しません。

また SH4AL-DSP では、SETRC 命令を用いたリピート制御（互換リピート制御）を LDRC 命令を用いた拡張リピート制御でエミュレーションしています。このため互換リピート制御中に RE レジスタの値が内部状態に応じて変化します。

(4)　モジュロレジスタ（MOD）

上位 16 ビットにモジュロアドレッシングの終了アドレスを、下位 16 ビットにモジュロアドレッシングの開始アドレスを格納します。MOD レジスタの上位 16 ビットを ME レジスタ、下位 16 ビットを MS レジスタと表現する場合もあります。モジュロレジスタは、DSP モードでのみアクセスできます。リセット状態に遷移したときの初期値は、不定です。例外処理状態に遷移しても値は変化しません。

これらのレジスタは、コントロールレジスタへのロード（LDC）およびストア（STC）命令でアクセスできます。RS、RE、および MOD に対する LDC と STC 命令は、特権 DSP モードとユーザ DSP モードで使用可能になります。

SR に対する LDC と STC 命令は、本来、MD ビットが 1 の場合にのみ使用可能な命令ですが、ユーザ DSP モードにおいても使用可能になります。ただし、値を書き替えられる制御ビットは、RC、RF、DMX、および DMY に限定されます。LDC と STC 命令使用時のステータスレジスタ（SR）の詳細は、下記のとおりです。

- ユーザモード時は、SRに対するLDC命令とSTC命令は不当命令例外となります。
- 特権モードと特権DSPモードでは、SRの全ビットが更新できます。
- ユーザDSPモード時は、SRはSTC命令で読み出し可能です。

  ユーザDSPモード時は、SRへのLDC命令発行は可能ですが、DSP拡張ビットのみ更新できます。

表 6.4　各処理モードにおける SR の各ビットの動作説明

フィールド	特権モード	ユーザモード	特権 DSP モード	ユーザ DSP モード	専用命令による DSP 関連ビットへの アクセス
	MD=1 & DSP=0	MD=0 & DSP=0	MD=1 & DSP=1	MD=0 & DSP=1	
MD	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：NG	
RB	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：NG	
BL	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：NG	
RC	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：OK	SETRC、LDRC 命令
DSP	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：NG	
DMY	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：OK	SETDMY、CLRDMXY 命令
DMX	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：OK	SETDMX、CLRDMXY 命令
M	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：NG	
Q	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：NG	
IMASK	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：NG	
RF	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：OK	SETRC、LDRC 命令
S	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：NG	
T	S：OK、L：OK	S、L：不当命令	S：OK、L：OK	S：OK、L：NG	

【注】　M、Q、S、T ビットはユーザモードで専用命令によってセット／クリアが可能です。

【記号説明】

S　　：STC 命令

L　　：LDC 命令

OK　　：STC と LDC 動作を許可します。

不当命令：実行すると不当命令例外が発生します。

NG　　：前の値を保持します。変化しません。

例外処理状態に遷移すると、DSP モードでの拡張ビットも含めた SR の全制御ビットが SSR へ待避されます。復帰時には、拡張 DSP ビットも含めて全制御ビットを回復してください。リピート制御を例外処理前の状態に復帰する必要がある場合には、RS と RE レジスタを例外処理前の値に回復してください。モジュロ制御を例外処理前の状態に復帰する必要がある場合には、MOD レジスタを例外処理前の値に回復してください。

6.2.4 DSP レジスタ

DSP ユニットは、8 つのデータレジスタ（A0、A1、X0、X1、Y0、Y1、M0、および M1）と 1 つのステータスレジスタ（DSR）を持っています。図 6.3 に DSP レジスタを示します。これらは、すべて 32 ビット幅のレジスタです。レジスタ A0 および A1 は、8 ビット幅のガードビットレジスタ（A0G および A1G）と組み合わせて、40 ビット幅のレジスタとしても使用されます。DSR は、DSP データ演算結果の状態（ゼロ、負、など）を保持し、また CPU の T ビットに類似した DC ビットを持っています。各ビットの詳細は、**「6.5 DSP データ演算命令」**を参照してください。

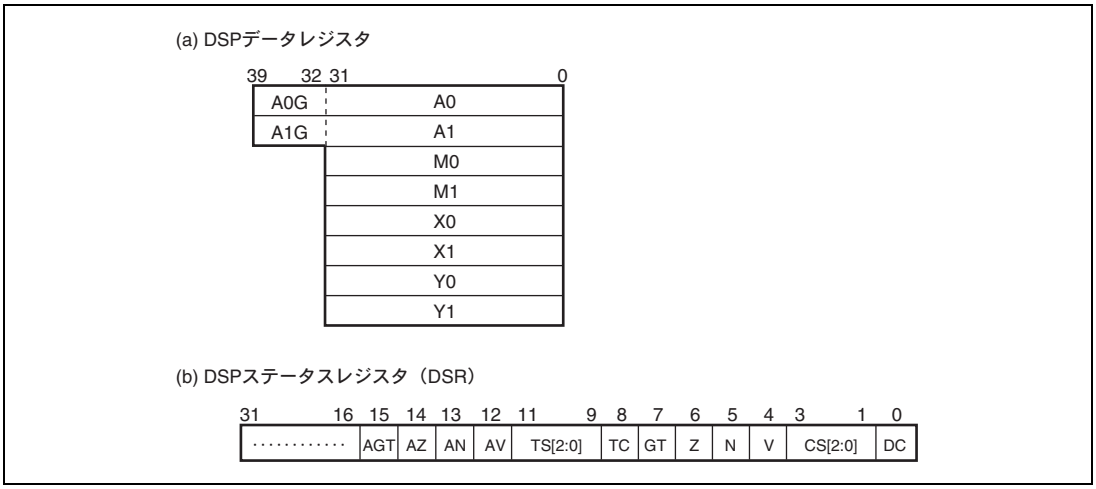


図 6.3 DSP レジスタの構成

6.3 CPU 拡張命令

リピート制御には、互換リピート制御と拡張リピート制御の 2 つが存在します。

6.3.1 互換リピート制御命令

DSP モードでは、リピートループを効率的に実行するための特別な制御機能が使用できます。この機能を使用することで、比較命令や分岐命令のオーバヘッドなしにループ構造のプログラムを実行することができます。

(1) リピートループのプログラム例

以下にリピートループを使用したプログラム例を示します。

- **(例1) 4命令以上のリピートループ**

```
                LDRS RptStart       ; RS レジスタに、リピート開始命令のアドレスをセットします。
                LDRE RptDtct+4      ; RE レジスタに、リピート検出命令のアドレス+4 をセットします。
                SETRC #4            ; SR レジスタの RC[11:0]フィールドにリピート回数（4）をセットします。
                Instr0              ; SETRC 命令から【リピート開始命令】までには少なくとも 1 個の命令が必要です。
RptStart:       instr1              ; 【リピート開始命令】
                ......              ;
                ......              ;
RptDtct:        instr(N-3)          ; リピート最終命令から 3 命令前の命令がリピート検出命令になります。
RptEnd2:        instr(N-2)          ;
RptEnd1:        instr(N-1)          ;
RptEnd:         instrN              ; 【リピート最終命令】
```

このプログラムの例では、RptStart のアドレスにある命令（instr1）から RptEnd のアドレスに配置された命令（instrN）までが 4 回繰り返し実行されます。繰り返し実行されるプログラム範囲をリピートループと呼び、その開始と終了命令をそれぞれリピート開始命令、およびリピート最終命令と呼びます。CPU は、命令を順次実行しながら、特定の命令の実行完了を検出することにより、リピートループの制御を開始します。この命令をリピート検出命令と呼びます。4 命令以上のリピートループでは、リピート最終命令から 3 命令前の命令がリピート検出命令になります。4 命令のリピートループでは RptStart 命令と RptDtct 命令は同じ命令になります。

リピートループの制御には、DSP モードで拡張されるコントロールレジスタ RE、RS および SR レジスタの RC[11:0]、RF[1:0]のビットフィールドが使用されます。また、これらのレジスタに値を設定するには、それぞれ LDRE、LDRS、SETRC 命令を使用します。

- リピートエンドレジスタ（RE）

  REには、LDRE命令で値をセットします。リピート検出命令のアドレス+4を設定します。4命令以上のリピートループでは、リピート最終命令から3命令前の命令がリピート検出命令になります。3命令以下のリピートループについては、後述します。

  なおSH4AL-DSPでは、SETRC命令を用いた互換リピート制御をLDRC命令を用いた拡張リピート制御（**「6.3.2 拡張リピート制御命令」**を参照）でエミュレーションしています。このため互換リピート制御中にREレジスタの値が内部状態に応じて変化します。

- リピートスタートレジスタ（RS）

  RSには、LDRS命令で値をセットします。4命令以上のリピートループでは、リピート開始命令のアドレスをセットします。3命令以下のリピートループでは特殊なアドレスを設定しますが、これについては後述します。

  なおSH4AL-DSPでは、SETRC命令を用いた互換リピート制御をLDRC命令を用いた拡張リピート制御でエミュレーションしています。このため互換リピート制御中にRSレジスタの値が内部状態に応じて変化します。

- リピートカウンタ（SRのRC[11:0]）

  SETRC命令により、繰り返し回数をセットします。リピートループ実行中は、繰り返しの残り回数を保持しています。

- リピートフラグ（SRのRF[1:0]）

  RFは、SETRC命令実行時に、RS、およびREレジスタに設定された値の関係から自動的に設定され、リピートループの命令数の情報を保持しています。また互換リピート制御中は、内部状態に応じて変化します。通常、ユーザが値を変更することはありません。書き込むときは、直前に読み出した値を書き込んでください。

CPU は、RE レジスタとプログラムカウンタ（PC）の値を常に比較しながら命令を実行します。PC は、命令アドレスの値を保持していますので、リピート検出命令実行時に両者が一致することで、リピート検出命令が検出されます。リピート検出命令の実行が分岐せずに完了し、かつ RC[11:0]>0 である場合にリピート制御が行なわれます。リピート最終命令の実行完了時に RC[11:0]>=2 であれば、RC[11:0]を 1 減じた後 RS レジスタに設定されたアドレスへ制御を移します。RC[11:0]==1（または 0）であれば RC[11:0]を 0 にしたのち、リピート最終命令の次の命令へ制御を移します。

リピートループを構成する命令数が 3、2、または 1 命令の場合のプログラム例を（例 2）（例 3）（例 4）に示します。これらの場合、リピート検出命令はリピート開始命令の直前の命令になります。また、RS レジスタにはリピート命令数を示す特殊な値を設定します。

- **（例2）3命令リピートループ**

```
            LDRS RptDtct+4    ; RS レジスタに、リピート検出命令のアドレス+4 をセットします。
            LDRE RptDtct+4    ; RE レジスタに、リピート検出命令のアドレス+4 をセットします。
            SETRC #4          ; SR レジスタの RC[11:0]フィールドにリピート回数（4）をセットします。
                              ; SETRC 命令実行時に、RE-RS==0 であれば 3 命令リピートと認識されます。
RptDtct:    instr0            ; リピート開始命令の直前の命令がリピート検出命令になります。
RptStart:   instr1            ; 【リピート開始命令】
            instr2            ;
RptEnd:     instr3            ; 【リピート最終命令】
```

- **（例3）2命令リピートループ**

```
            LDRS RptDtct+6    ; RS レジスタに、リピート検出命令のアドレス+6 をセットします。
            LDRE RptDtct+4    ; RE レジスタに、リピート検出命令のアドレス+4 をセットします。
            SETRC #4          ; SR レジスタの RC[11:0]フィールドにリピート回数（4）をセットします。
                              ; SETRC 実行時に RE-RS==-2 であれば 2 命令リピートと認識されます。
RptDtct:    instr0            ; リピート開始命令の直前の命令がリピート検出命令になります。
RptStart:   instr1            ; 【リピート開始命令】
RptEnd:     instr2            ; 【リピート最終命令】
```

- **（例4）1命令リピートループ**

```
            LDRS RptDtct+8    ; RS レジスタに、リピート検出命令のアドレス+8 をセットします。
            LDRE RptDtct+4    ; RE レジスタに、リピート検出命令のアドレス+4 をセットします。
            SETRC #4          ; SR レジスタの RC[11:0]フィールドにリピート回数（4）をセットします。
                              ; SETRC 実行時に RE-RS==-4 であれば 1 命令リピートと認識されます。
RptDtct:    instr0            ; リピート開始命令の直前の命令がリピート検出命令になります。
RptStart:
RptEnd:     instr1            ; 【リピート開始命令】==【リピート最終命令】
```

3、2、および 1 命令リピートの場合には、RS レジスタにリピートループ中の命令数を示すための特殊なアドレスを設定します。SETRC 命令を実行した際に、RE から RS を引いた結果が 0、-2、-4 のとき、それぞれ 3 命令、2 命令、1 命令のリピートループとして認識されます。

リピート検出命令の実行が分岐せずに完了し、かつ RC[11:0]>0 である場合には、リピート検出命令の次の命令をリピート開始命令として、認識されたリピート命令数分の命令を繰り返し実行します。リピート最終命令実行完了時に、RC[11:0]>=2 であれば、RC[11:0]を 1 減じた後リピート開始命令へ制御を移します。RC[11:0]==1（または 0）であれば RC[11:0]を 0 にしたのち、リピート最終命令の次の命令へ制御を移します。

【注】 RE-RS の値が正の場合に、CPU は 4 命令以上のリピートループと認識し、リピートループを制御します（4 命令以上のリピートループの場合、RE-RS の値は常に正の値になります。（例 1）のプログラム例を参考にしてください）。RE-RS の値が正の値、0、-2、-4 以外の値になった場合の動作は保証しません。

表 6.5 に、リピートスタートレジスタ（RS）、リピートエンドレジスタ（RE）に設定すべきアドレスをまとめます。

表 6.5　互換リピート制御 RS および RE のアドレス設定ルール

	リピートループ中の命令数			
	1	2	3	>=4
RS	RptStart0 +8	RptStart0 +6	RptStart0 +4	RptStart
RE	RptStart0 +4	RptStart0 +4	RptStart0 +4	RptEnd3+4

【注】 ここでは、次のラベルを使用しています。

RptStart：リピート開始命令のアドレス

RptStart0：リピート開始命令の 1 命令前の命令アドレス

RptEnd3：リピート最終命令の 3 つ前の命令アドレス

(2) 互換リピート制御命令およびリピート制御マクロ

リピートループを記述するには、前節で例示したように、LDRS および LDRE 命令でそれぞれ RS と RE レジスタに適切なアドレスを設定した後、SETRC 命令でリピート回数を指定してください。SETRC のオペランドとしては、8 ビットの即値または汎用レジスタが使用できます。RC に 256 を超える値を設定するには、SETRC Rm タイプの命令を使用してください。

表 6.6　互換リピート制御命令

命令	動　　作	実行ステート
LDRS @(disp,PC)	(disp×2+PC)を算出し、RS レジスタに値を設定します。	1
LDRE @(disp,PC)	(disp×2+PC)を算出し、RE レジスタに値を設定します。	1
SETRC* #imm	8 ビット定数 imm を SR レジスタの RC[11:0]に設定し、SR レジスタの RF[1:0]にリピート命令数を示す情報を設定します。 RC[11:0]には、0 から 255 までの値が設定できます。	1
SETRC* Rm	Rm レジスタの[11:0]を SR レジスタの RC[11:0]に設定し、SR レジスタの RF[1:0]にリピート命令数を示す情報を設定します。 RC[11:0]には、0 から 4095 までの値が設定できます。	1

【注】 * SETRC 命令により 1 以上のリピート回数を設定する前には必ず LDRS 命令と LDRE 命令を毎回実行するようにしてください。

RS および RE レジスタには、**表 6.5** に示したルールに従って**表 6.6** に示すリピート制御命令により適切なアドレスを設定する必要があります。SuperH アセンブラでは、この問題を処理するために、**表 6.7** に示すリピート制御マクロ（REPEAT）が提供されています。

表 6.7　互換リピート制御のリピート制御マクロ

命令	動　　作	実行 ステート
REPEAT RptStart, RptEnd, #imm	RptStart をリピート開始命令、RptEnd をリピート最終命令とし、8 ビットの即値#imm をリピート回数とするリピートループを設定します。適切に変換された LDRS、LDRE、および SETRC の 3 命令に展開されます。	3
REPEAT RptStart, RptEnd, Rm	RptStart をリピート開始命令、RptEnd をリピート最終命令とし、Rm の[11:0]をリピート回数とするリピートループを設定します。適切に変換された LDRS、LDRE、および SETRC の 3 命令に展開されます。	3

【注】　リピート制御マクロには、#imm や Rm を指定しないことで SETRC 命令の指定を独立に行う機能がありますが、SH4AL-DPS では SETRC 命令を用いて 1 以上のリピート回数を指定する前には必ず LDRS および LDRE 命令を実行する必要があるためこの制限に違反しないように注意してください。

リピート制御マクロを使用することで、前述した（例 1）～（例 4）は、それぞれ次に示す（例 5）～（例 8）の様に簡略に記述することができます。

- **（例5）4命令以上のリピートループの記述例…（例1）に示した命令列に展開されます。**

```
            REPEAT RptStart, RptEnd, #4
            instr0          ;
RptStart:   instr1          ; 【リピート開始命令】
            ......          ;
            ......          ;
            instr(N-3)      ;
            instr(N-2)      ;
            instr(N-1)      ;
RptEnd:     instrN          ; 【リピート最終命令】
```

- **（例6）3命令リピートループの記述例…（例2）に示した命令列に展開されます。**

```
            REPEAT RptStart, RptEnd, #4
            instr0          ;
RptStart:   instr1          ; 【リピート開始命令】
            instr2          ;
RptEnd:     instr3          ; 【リピート最終命令】
```

- **（例7）2命令リピートループ…（例3）に示した命令列に展開されます。**

```
            REPEAT RptStart, RptEnd, #4
            instr0              ;
RptStart:   instr1              ; 【リピート開始命令】
RptEnd:     instr2              ; 【リピート最終命令】
```

- **（例8）1命令リピートループ…（例4）に示した命令列に展開されます。**

```
            REPEAT RptStart, RptEnd, #4
            instr0              ;
RptStart:
RptEnd:     instr1              ; 【リピート開始命令】==【リピート最終命令】
```

DSP モードでは、RS および RE レジスタの値を操作するシステム制御命令（LDC と STC 命令）が拡張されます。**表 6.8** に DSP モードにおける拡張システム制御命令を示します。また、SR レジスタの RC[11:0]および RF[1:0]のビットフィールドは、SR レジスタに対する LDC と STC 命令で制御できます。これらの命令は、リピートループ中に例外を受け付ける場合に使用してください。RS と RE レジスタおよび SR レジスタの RC[11:0]と RF[1:0]のビットフィールドを退避した後、回復することでリピートループを再開することができます。ただし、リピートループ中の例外の受け付けには、制限事項があります。

表 6.8　DSP モード拡張システム制御命令

命令	動　　作	実行ステート
STC RS,Rn	RS→Rn	1
STC RE,Rn	RE→Rn	1
STC.L RS,@-Rn	Rn-4→Rn, RS→(Rn)	1
STC.L RE,@-Rn	Rn-4→Rn, RE→(Rn)	1
LDC.L @Rn+,RS	(Rn)→RS, Rn+4→Rn	1
LDC.L @Rn+,RE	(Rn)→RE, Rn+4→Rn	1
LDC Rn,RS	Rn→RS	1
LDC Rn,RE	Rn→RE	1

(3) 互換リピート制御中の制限事項

(a) リピート制御命令の配置

LDRS および LDRE 命令を実行した後に SETRC 命令を実行してください。SETRC 命令により 1 以上のリピート回数を再設定する場合、必ず LDRS および LDRE 命令を再実行する必要があります。また、SETRC 命令とリピート開始命令の間には少なくとも 1 命令が必要です。

(b) リピート検出命令に続く命令以降の禁止命令

従来の SH3-DSP ではリピート検出命令の次の命令からリピート最終命令の間に、以下に示す命令を配置できませんでしたが、SH4AL-DSP ではさらにリピート検出命令に以下に示す命令と遅延分岐命令の遅延スロットを配置できなくなります。また従来 SH3-DSP の互換リピート制御では、リピート検出命令の次の命令からリピート最終命令の間に以下に示す命令を配置すると不当命令例外を発生しましたが、SH4AL-DSP では、リピート最終命令に以下に示す命令を配置する場合のみ不当命令例外を発生し、それ以外の場合は不当命令例外は発生しません。

- 分岐命令

  BRA、BSR、BT、BF、BT/S、BF/S、BSRF、RTS、BRAF、RTE、JSR、JMP

- リピート制御命令

  SETRC、LDRS、LDRE、LDRC

- SR、RS、REに対するロード命令

  LDC Rn,SR、LDC.L @Rn+,SR、LDC Rn,RE、LDC.L @Rn+,RE、LDC Rn,RS、LDC.L @Rn+,RS

【注】 1～3 命令のリピートループの場合はリピートループ中の全命令が、4 命令以上のリピートループの場合はリピート終了命令を含む 3 命令がこの制約の範囲となります。

(c) リピートループ中の禁止命令（4 命令以上のリピートループ）

4 命令以上のリピートループのリピート開始命令からリピート検出命令までの間に、以下の命令を配置しないでください。配置した場合の動作は保証されません。

- リピート制御命令

  SETRC、LDRS、LDRE、LDRC

- SR、RS、REに対するロード命令

  LDC Rn,SR、LDC.L @Rn+,SR、LDC Rn,RE、LDC.L @Rn+,RE、LDC Rn,RS、LDC.L @Rn+,RS

【注】 多重のリピートループは保証されません。最内部のループをリピート制御命令で記述し、外部のループは DT および BF/S 命令等で実現してください。

(d) リピート検出命令の次命令以降への分岐および例外受理に関する制限

従来の SH3-DSP では、リピート検出命令の次命令以降に分岐した場合はリピートループが認識されませんでしたが、SH4AL-DSP ではリピートループが認識されることがあります。ただしリピート検出命令の次命令からリピート最終命令を分岐先に指定することは禁止します。また従来の SH3-DSP では例外ルーチンからの復帰も本制限に含まれていましたが、SH4AL-DSP では例外ルーチンの復帰は本制限に含まれません。

- リピートループ中で条件分岐命令を使用する場合は、リピート検出命令以前の命令を分岐先に指定してください。
- リピートループ中でサブルーチンコールを使用する場合は、サブルーチンコール命令の遅延スロット命令がリピート検出命令より前になるように配置してください。

ここでの分岐には、例外ルーチンからの復帰を含みます。

(e) リピート検出命令からの分岐

従来の SH3-DSP ではリピート検出命令が遅延分岐命令の遅延スロット命令である場合や分岐命令そのものである場合は、分岐命令で分岐しなかったときにリピートループが認識され、分岐したときにはリピート制御が行われず分岐先命令を実行していましたが、SH4AL-DSP ではリピート検出命令に分岐命令および遅延分岐命令の遅延スロットを配置することができません。

(f) リピートカウンタとリピート制御

CPU は、常にリピートエンドレジスタ（RE）と PC との比較を行ないながらプログラムを実行しています。SR レジスタの RC[11:0]が 0 以外で PC が RE に一致すると、リピート制御が機能します。

- RC>=2の場合は、リピート最終命令実行後、リピート開始命令に制御が移行します。最終命令の実行完了によりRCが1減じられます。（a）～（e）の制約がかかります。
- RC==1の場合は、リピート最終命令実行後、RCが0になり、後続命令へ制御が移行します。RC==1の場合も（a）～（e）の制約がかかります。
- RC==0の場合は、リピート検出命令を実行しても、リピート制御は機能しません。リピートループは通常の命令列として1回実行され、最終命令を実行してもリピート開始命令へは制御が移行しません。

6.3.2 拡張リピート制御命令

「6.3.1 **互換リピート制御命令**」で提供されるリピート制御機構には、幾つかの制約事項があります。この制約を軽減するためのリピート制御機能が拡張されています。これらの命令は、従来の SH-DSP アーキテクチャには存在しない命令で、互換性を重視する場合には従来の互換リピート制御命令を使用します。

(1) 拡張リピート制御命令のプログラム例

以下に拡張リピート制御命令を使用したプログラム例を示します。

- **(例1) 4命令以上のリピートループ**

```
            LDRS RptStart     ; RS レジスタに、リピート開始命令のアドレスをセットします。
            LDRE RptEnd       ; RE レジスタに、リピート最終命令のアドレスをセットします。
            LDRC #4           ; SR レジスタの RC[11:0]フィールドにリピート回数（4）をセットします。
            instr0            ; LDRC 命令から【リピート開始命令】までには少なくとも 1 個の命令が必要です。
RptStart:   instr1            ; 【リピート開始命令】
            ......            ;
            ......            ;
            instr(N-3)        ;
            instr(N-2)        ;
            instr(N-1)        ;
RptEnd:     instrN            ; 【リピート最終命令】
```

- **(例2) 3命令リピートループ**

```
            LDRS RptStart     ; RS レジスタに、リピート開始命令のアドレスをセットします。
            LDRE RptEnd       ; RE レジスタに、リピート最終命令のアドレスをセットします。
            LDRC #4           ; SR レジスタの RC[11:0]フィールドにリピート回数（4）をセットします。
            instr0            ; LDRC 命令とリピート開始命令の間には少なくとも 1 命令が必要です。
RptStart:   instr1            ; 【リピート開始命令】
            instr2            ;
RptEnd:     instr3            ; 【リピート最終命令】
```

- **（例3）2命令リピートループ**

```
                LDRS RptStart       ; RS レジスタに、リピート開始命令のアドレスをセットします。
                LDRE RptEnd         ; RE レジスタに、リピート最終命令のアドレスをセットします。
                LDRC #4             ;SR レジスタの RC[11:0]フィールドにリピート回数（4）をセットします。
                instr0              ; LDRC 命令とリピート開始命令の間には少なくとも 1 命令が必要です。
RptStart:       instr1              ; 【リピート開始命令】
RptEnd:         instr2              ; 【リピート最終命令】
```

- **（例4）1命令リピートループ**

```
                LDRS RptStart       ; RS レジスタに、リピート開始命令のアドレスをセットします。
                LDRE RptEnd         ; RE レジスタに、リピート最終命令のアドレスをセットします。
                LDRC #4             ;SR レジスタの RC[11:0]フィールドにリピート回数（4）をセットします。
                instr0              ; LDRC 命令とリピート開始命令の間には少なくとも 1 命令が必要です。
RptStart:
RptEnd:         instr1              ; 【リピート開始命令】=【リピート最終命令】
```

拡張リピート制御命令では、リピート命令数によらず、RS レジスタにリピート開始命令のアドレスを、RE レジスタにリピート最終命令のアドレスを格納します。また、SETRC 命令の代わりに LDRC 命令を用いることで、拡張リピート制御がおこなわれます。拡張リピート制御が行なわれている場合は、リピート最終命令を実行することでリピートループが認識できます。このため、分岐や例外に対する制約がありません。

(2) 拡張リピート制御命令

拡張リピートループを記述するには前節で例示したように、LDRS と LDRE 命令でそれぞれ RS と RE レジスタにそれぞれリピート先頭命令およびリピート最終命令を指定します。**表 6.9** に拡張リピート制御命令を示します。LDRS と LDRE 命令は、互換リピート制御用の命令をそのまま使用します。その後、LDRC 命令でリピート回数を指定してください。LDRC のオペランドとしては、8 ビットの即値または汎用レジスタが使用できます。RC に 256 を超える値を設定するには、LDRC Rm タイプの命令を使用してください。

表 6.9 拡張リピート制御命令

命令	動　　作	実行ステート
LDRS @(disp,PC)	(disp×2+PC)を算出し、RS レジスタに値を設定します。	1
LDRE @(disp,PC)	(disp×2+PC)を算出し、RE レジスタに値を設定します。	1
LDRC #imm	8 ビット定数 imm を SR レジスタの RC[11:0]に設定し、SR レジスタの RF[1:0]にリピート命令数を示す情報を設定します。RC[11:0]には、0 から 255 までの値を設定できます。 拡張リピート制御中を示すために、RE レジスタのビット 0 に 1 がセットされます。	1

命令	動　　作	実行ステート
LDRC Rm	Rm レジスタの[11:0]を SR レジスタの RC[11:0]に設定し、SR レジスタの RF[1:0]にリピート命令数を示す情報を設定します。RC[11:0]には、0 から 4095 までの値を設定できます。 拡張リピート制御中を示すために、RE レジスタのビット 0 に 1 がセットされます。	1

LDRC 命令を実行することで、以後 CPU は拡張リピートとして制御を行います。拡張リピート制御中であることを示すために、LDRC 命令の実行により RE レジスタのビット 0 に 1 がセットされます。例外処理等で RE レジスタの値を変更する場合は、ビット 0 も正確に退避した後に回復してください。SR レジスタの RC[11:0]、DSP、RF[1:0]のビットフィールドおよびREとRSレジスタを退避した後に回復することで拡張リピートとして処理に復帰することができます。

(3) 拡張リピート制御中の制限事項

(a) 拡張リピート制御命令の配置

LDRS および LDRE 命令を実行した後に LDRC 命令を実行してください。また、LDRC 命令とリピート開始命令の間には、少なくとも 1 命令が必要です。

(b) リピート最終命令での不当命令

リピート最終命令として次の命令を配置すると不当命令例外を発生します。

- 遅延分岐命令

  BRA、BSR、BT/S、BF/S、BSRF、RTS、BRAF、RTE、JSR、JMP

- リピート制御命令

  SETRC、LDRS、LDRE、LDRC

- SR、RS、REに対するロード命令

  LDC Rn,SR、LDC.L @Rn+,SR、LDC Rn,RE、LDC.L @Rn+,RE、LDC Rn,RS、LDC.L @Rn+,RS

【注】 非遅延の分岐命令（BT、BF）は、最終命令として配置してもかまいません。また、遅延分岐命令の遅延スロットがリピート最終命令となってもかまいません。これらの場合、分岐したときも分岐しなかったときも RC[11:0]の値は 1 減じられます。分岐しなかったときはリピート開始命令へ、分岐したときには分岐先へ制御が移行します。

(c) リピートカウンタとリピート制御

CPU は、常にリピートエンドレジスタ（RE）と PC-4（命令のアドレス）との比較を行ないながらプログラムを実行しています。RE レジスタのビット 0 が 1 であり、SR レジスタの RC[11:0]が 0 以外で PC-4[31:1]と RE[31:1]が一致すると、拡張リピート制御が行なわれます。

- RC>=2の場合は、リピート最終命令実行後、リピート開始命令に制御が移行します。最終命令の実行完了によりRCが1減じられます。
- RC==1の場合は、リピート最終命令実行後、RCが0になり、後続命令へ制御が移行します。
- RC==0の場合は、リピート検出命令を実行しても、リピート制御は機能しません。リピートループは通常の命令列として1回実行され、最終命令を実行してもリピート開始命令へは制御が移行しません。

6.4 DSP データ転送命令

DSP モードでは、DSP ユニットのレジスタに対するデータ転送命令が追加されます。追加されるデータ転送命令は、次の 3 種類に分類されます。

図 6.4 に DSP ユニットのレジスタとバス接続を示します。DSP ユニットは、X メモリと Y メモリに X バスと Y バスと呼ばれる専用バスで接続されており、これらのバスを用いたデータ転送命令を使用することで、X/Y メモリとの間で同時に 2 個のデータを転送することができます（ダブルデータ転送命令）。このダブルデータ転送命令は、DSP 演算命令と組み合わせて記述することができ、データ転送およびデータ演算を並列に実行することが可能です。

また、DSP ユニットは、オペランドバスと呼ばれる CPU が使用するバスとも接続されており、DSR を除く全レジスタは CPU の生成する論理アドレス空間すべてにアクセスすることができます（シングルデータ転送命令）。シングルデータ転送命令は、DSP 演算命令と組み合わせることはできず、また一度にアクセスできるデータは 1 個だけになります。

さらに DSP ユニットのレジスタのうち幾つかは、CPU のシステムレジスタとして扱われ、これらを制御するためのシステム制御命令が追加されています。CPU の汎用レジスタとの間は、データ転送用のバス（転送バス）で接続されています。

いずれのタイプのデータ転送命令でも、アクセスするアドレスは CPU が生成し、出力します。これらの命令に対しては CPU の汎用レジスタの幾つかがアドレス生成に使用され、また独特のアドレッシングモードを有します。

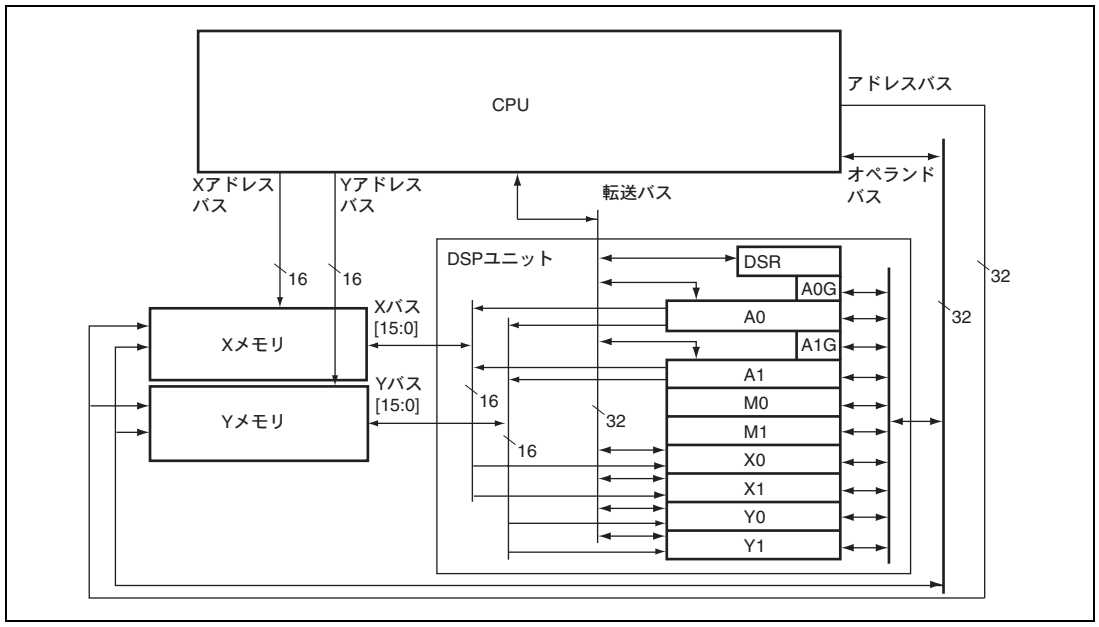


図 6.4 DSP レジスタとバスの接続

(1) ダブルデータ転送命令（MOVX.W、MOVY.W、MOVX.L、および MOVY.L）

ダブルデータ転送命令では、X メモリに対するアクセスおよび Y メモリに対するアクセスを同時に記述することができます。このアクセスにはそれぞれ X バスと Y バスと呼ばれる専用バスを使用します。CPU の命令フェッチは命令バスを使用するため、CPU の命令フェッチとの間でバス競合が生じることもありません。

X メモリに対するロード命令はデスティネーションオペランドとして X0 と X1 レジスタのどちらか 1 つを指定し、Y メモリに対するロード命令はデスティネーションオペランドとして Y0 と Y1 レジスタのどちらか 1 つを指定できます。X メモリと Y メモリのいずれのストア命令もソースオペランドとして A0 と A1 レジスタのどちらか 1 つを指定することができます。この種の命令はワードデータ（16 ビット）のみを扱います。ワードデータ転送命令を実行すると、レジスタオペランドの上位ワードが用いられます。ワードデータロードの場合は、データはデスティネーションレジスタの上位ワードにデータが読み込まれ、デスティネーションの下位側が自動的に 0 クリアされます。

ダブルデータ転送命令では X バス、Y バスでの転送をそれぞれ組み合わせて指定することができますが、一方の転送動作が不要の場合には転送機能を拡張することができます（MOVX.W&NOPY、NOPX&MOVY.W）。この形式では、使用されないバスを活用して 32 ビットのデータを転送することもできます（MOVX.L&NOPY、NOPX&MOVY.L）。X メモリからのロード命令（Y メモリ側は NOPY である必要があります）は、デスティネーションオペランドとして X0、X1、Y0 および Y1 を指定することができます。Y メモリからのロード命令（X メモリ側は NOPX である必要があります）は、デスティネーションオペランドとして Y0、Y1、X0、および X1 を指定することができます。また、X メモリへのストア命令ではソースオペランドとして X0、X1、A0 および A1 が、Y メモリへのストア命令では Y0、Y1、A0 および A1 が使用可能です。また、この種のダブルデータ転送命令では、アドレッシングに使用できるアドレスポインタの種類も通常のダブルデータ転送命令と比較して多くなっています。

ダブルデータ転送命令では、DSP データ演算命令を並行して記述することができます。ただし、実行する演算命令に条件付き命令を指定した場合でも、指定した条件はどのデータ転送命令に対しても影響しません。条件付き命令については、**「6.5 DSP データ演算命令」**を参照してください。

ダブルデータ転送命令は、X メモリか Y メモリのみアクセスすることができます。その他のメモリ空間はアクセスすることができません。また、X バス、Y バスは、それぞれ 16 ビット（64K バイト）のアドレス空間を持っており、オペランドバスのアドレス空間の H'A500 0000～H'A500 FFFF および H'A501 0000～H'A501 FFFF の範囲に対応します。この範囲は、P2/Uxy 領域に含まれるため、キャッシュやアドレス変換機構の影響を受けません。

(2) シングルデータ転送命令

この種の命令は、任意のメモリアドレス空間にアクセスできます。DSR を除く DSP ユニットのすべてのレジスタ*をソースオペランド、デスティネーションオペランドに指定することができます。ガードビットレジスタ、A0G、および A1G も独立したレジスタとして指定することができます。この種の命令では、オペランドバスを使用するので、CPU の扱うすべての論理アドレス空間をアクセスすることができます。キャッシャブル領域を指定しかつキャッシュが利用可能な場合には、キャッシングの対象になります。X と Y メモリは、論理アドレス空間の一部としてマッピングされており、シングルデータ転送命令でアクセスすることができます。

シングルデータ転送は、ワードとロングワードのいずれも扱うことができます。ワードデータ転送を実行するとき、レジスタオペランドの上位ワードが有効になります。ワードデータロードの場合は、データはデスティネーションレジスタの上位ワードに読み込まれ、デスティネーションの下位側は自動的に 0 でクリアされます。ガードビット部分がサポートされている場合には、符号ビットが拡張されて格納されます。ロングワードデータロードの場合は、データはデスティネーションレジスタの上位ワードと下位ワードに読み込まれ、ガードビットがあれば、符号ビットが拡張されて格納されます。ガードレジスタ ストアの場合は、符号ビットがオペランドバスの上位 24 ビットに拡張されてオペランドバスに読み出されます。

【注】 * DSR レジスタは、システムレジスタとして定義されているので、LDS、STS 命令でのデータの転送が可能です。すべてのデータ転送命令は、DSR レジスタのいずれのビットも更新しません。

(3) システム制御命令

DSP ユニットのレジスタの内 DSR、A0、X0、X1、Y0、Y1 レジスタは、CPU のシステムレジスタとして扱うことができ、STS と LDS 命令によって汎用レジスタやメモリとの間でデータ転送を行うことができます。**表 6.10** に、DSP モードの拡張システム制御命令を示します。これらのシステム制御命令は CPU レジスタの PR、MACH、MACL と全く同じように扱うことができ、アドレッシングも同一です。

表 6.10 DSP モード拡張システム制御命令

命令	動　　作	実行ステート
STS DSR,Rn	DSR → Rn	1
STS A0,Rn	A0 → Rn	1
STS X0,Rn	X0 → Rn	1
STS X1,Rn	X1 → Rn	1
STS Y0,Rn	Y0 → Rn	1
STS Y1,Rn	Y1 → Rn	1
STS.L DSR,@-Rn	Rn-4 → Rn, DSR →(Rn)	1
STS.L A0,@-Rn	Rn-4 → Rn, A0 →(Rn)	1
STS.L X0,@-Rn	Rn-4 → Rn, X0 →(Rn)	1
STS.L X1,@-Rn	Rn-4 → Rn, X1 →(Rn)	1
STS.L Y0,@-Rn	Rn-4 → Rn, Y0 →(Rn)	1
STS.L Y1,@-Rn	Rn-4 → Rn, Y1 →(Rn)	1
LDS.L @Rn+,DSR	(Rn) → DSR, Rn+4 → Rn	1
LDS.L @Rn+,A0	(Rn) → A0, Rn+4 → Rn	1
LDS.L @Rn+,X0	(Rn) → X0, Rn+4 → Rn	1
LDS.L @Rn+,X1	(Rn) → X1, Rn+4 → Rn	1
LDS.L @Rn+,Y0	(Rn) → Y0, Rn+4 → Rn	1
LDS.L @Rn+,Y1	(Rn) → Y1, Rn+4 → Rn	1
LDS Rn,DSR	Rn → DSR	1
LDS Rn,A0	Rn → A0	1

命令	動　　作	実行ステート
LDS Rn,X0	Rn → X0	1
LDS Rn,X1	Rn → X1	1
LDS Rn,Y0	Rn → Y0	1
LDS Rn,Y1	Rn → Y1	1

6.4.1　汎用レジスタ

DSP タイプの命令では、汎用レジスタ 16 本のうち 10 本のレジスタがダブルデータ転送命令とシングルデータ転送命令で特別なアドレスポインタおよびインデックスレジスタとして使用されます。DSP タイプ命令でのレジスタの目的を表すもう 1 つの記号を[　]内に示します。

- ダブルデータ転送命令（XメモリとYメモリに同時にアクセスする場合）

  ダブルデータ転送命令は、XとYデータメモリに同時にアクセスできます。XとYデータメモリのアドレスを指定するために、次の2つのアドレスポインタセットを用意しています。

	アドレスポインタ	インデックスレジスタ
X メモリ（MOVX.W）	R4,R5[Ax]	R8[Ix]
Y メモリ（MOVY.W）	R6,R7[Ay]	R9[Iy]

- ダブルデータ転送命令（XメモリとYメモリの片方にアクセスする場合）

  ダブルデータ転送命令で、XメモリとYメモリの片方の転送動作が不要の場合は、次のようにアドレスポインタセットを拡張して使用することができます。

	アドレスポインタ	インデックスレジスタ
X メモリ（MOVX.W/L&NOPY）	R4,R5,R0,R1[Axy]	R8[Ix]
Y メモリ（NOPX & MOVY.W/L）	R6,R7,R2,R3[Ayx]	R9[Iy]

- シングルデータ転送命令

  シングルデータ転送命令では、オペランドバスを使用してすべての論理アドレス空間をアクセスできます。次のアドレスポインタとインデックスレジスタを使用します。

	アドレスポインタ	インデックスレジスタ
全論理空間（MOVS.W/L）	R4,R5,R2,R3[As]	R8[Is]

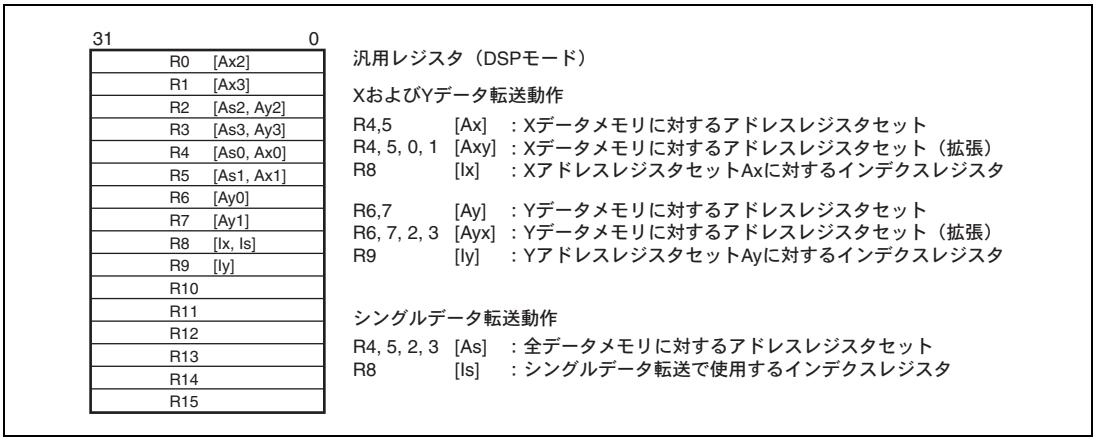


図 6.5 汎用レジスタ（DSP モード）

アセンブラでは R0、R1、R2、R3……R9 の記号名（シンボル）を使います。DSP データ転送命令では、次のようなレジスタの別名（エイリアス、alias）を使うこともできます。アセンブラで次のように書きます。

```
Ix:   .REG   (R8)
```

名前 Ix が R8 の別名になります。そのほか、次のように別名を付けます。

```
Ax0: .REG   (R4)
Ax1: .REG   (R5)
Ax2: .REG   (R0); この定義は、Y メモリ動作が NOPY の場合にのみ使用できます。
Ax3: .REG   (R1); この定義は、Y メモリ動作が NOPY の場合にのみ使用できます。
Ix:   .REG   (R8)
Ay0: .REG   (R6)
Ay1: .REG   (R7)
Ay2: .REG   (R2); この定義は、X メモリ動作が NOPX の場合にのみ使用できます。
Ay3: .REG   (R3); この定義は、X メモリ動作が NOPX の場合にのみ使用できます。
Iy:   .REG   (R9)
As0: .REG   (R4);
As1: .REG   (R5);
As2: .REG   (R2)
As3: .REG   (R3)
Is:   .REG   (R8);
```

6.4.2 DSP データアドレッシング

ダブルデータ転送命令およびシングルデータ転送命令の関係を**表 6.11** に示します。

表 6.11 データ転送命令の関係

	ダブルデータ転送命令		シングルデータ転送命令
	MOVX.W MOVY.W	MOVX.W&NOPY NOPX&MOVY.W MOVX.L&NOPY NOPX&MOVY.L	MOVS.W MOVS.L
アドレスレジスタ	Ax：R4、R5 Ay：R6、R7	Axy： R4、R5、R0、R1 Ayx： R6、R7、R2、R3	As：R2、R3、R4、R5
インデックス レジスタ	Ix：R8 Iy：R9	Ix：R8、Iy：R9	Is：R8
アドレッシング	Nop/Inc(+2)/インデクス加算： ポストインクリメント	Nop/Inc(+2/+4)/インデクス加算： ポストインクリメント	Nop/Inc(+2、+4)/インデクス加算： ポストインクリメント
アドレッシング	—	—	Dec(-2、-4)：プリデクリメント
モジュロ アドレッシング	可能	可能	不可
データバス	XDB、YDB	XDB、YDB	LDB
データ長	16 ビット（ワード）	16 ビット／32 ビット （ワード／ロングワード）	16 ビット／32 ビット （ワード／ロングワード）
バス競合	なし	なし	あり
メモリ	X、Y データメモリ	X、Y データメモリ	すべてのメモリ空間
ソースレジスタ	Da：A0、A1	Dax：A0、A1、X0、X1 Day：A0、A1、Y0、Y1	Ds：A0、A1、M0、M1、X0、X1、 Y0、Y1、A0G、A1G
デスティネーション レジスタ	Dx：X0、X1 Dy：Y0、Y1	Dxy：X0、X1、Y0、Y1 Dyx：Y0、Y1、X0、X1	Ds：A0、A1、M0、M1、X0、X1、 Y0、Y1、A0G、A1G

(1) ダブルデータ転送命令のアドレッシングモード

ダブルデータ転送命令には、次の 3 つのアドレッシングモードがあります。

- 更新なし

  AxとAyレジスタがアドレスポインタです。@Axと@Ayへのアクセスが行なわれ、AxとAyの値は更新されません。

- インクリメント

  AxとAyレジスタがアドレスポインタです。@Axと@Ayへのアクセス後、転送サイズに応じて+2または+4が加算されます（ポスト更新）。

- インデクスレジスタ加算

  AxとAyレジスタがアドレスポインタです。@Axと@Ayへのアクセス後、それぞれIxとIyレジスタの値が加算されます（ポスト更新）。ダブルデータ転送命令にデクリメントアドレッシングはありませんが、デクリメントさせるためには-2または-4をインデクスレジスタに設定し、インデクスレジスタ加算アドレッシングを指定します。

X と Y データアドレッシングの場合は、ワードアクセスの場合アドレスポインタのビット 0 が、ロングワードアクセスの場合ビット 0 と 1 が無効になります。X と Y データアドレッシングの場合は、アドレスポインタとインデックスレジスタのこれらのビットには 0 を書き込んでください。

X と Y バスを使用して X メモリと Y メモリへアクセスする場合は、Ax と Ay の上位ワードは無視されます。また、Ay+と Ay+Iy の結果は、Ay の下位ワードに格納され、上位ワードは元の値が保持されます。ただし、Ax+と Ax+Ix の演算は 32 ビットで行なわれ、上位ワードが変化する場合もあります。

ダブルデータ転送命令で X メモリと Y メモリの一方の転送動作が不要の場合、転送機能を拡張することができます。このときアドレスポインタは、Ax、Ay の代わりに Axy、Ayx が用いられます。アドレッシングモードは同じです。

(2) シングルデータ転送命令のアドレッシングモード

シングルデータ転送命令には、次の 4 つのデータアドレッシングモードがあります。

- 更新なし

  Asレジスタがアドレスポインタです。@Asへのアクセスが行なわれますが、Asは更新されません。

- インクリメント

  Asレジスタがアドレスポインタです。@Asへのアクセス後、転送サイズに応じて＋2または＋4が加算されます（ポスト更新）。

- インデクスレジスタ加算

  Asレジスタがアドレスポインタです。@Asへのアクセス後、Isレジスタの値が加算されます（ポスト更新）。

- デクリメント

  Asレジスタがアドレスポインタです。データ転送前に－2または－4が加算（＋2または＋4が減算）されます（プリ更新）。

シングルデータ転送命令では、アドレスの 32 ビットすべてが有効です。

6.4.3 モジュロアドレッシング

ダブルデータ転送命令では、モジュロアドレッシングを使用することができます。モジュロアドレッシングモードが設定されている場合は、アドレスポインタの値がすでに設定されたモジュロ終了アドレスになると、アドレスポインタはモジュロ開始アドレスになります。

モジュロアドレッシングの制御には、DSP モードで拡張されるモジュロレジスタ（MOD）および SR レジスタの DMX と DMY ビットを使用します。**表 6.12** に、モジュロアドレッシング制御命令を示します。

MOD レジスタにモジュロアドレス領域の開始と終了アドレスを格納します。MOD レジスタの下位ワードにモジュロ開始アドレス（MS）を、MOD レジスタの上位ワードにモジュロ終了アドレス（ME）を格納します。MOD レジスタに対する LDC 命令および STC 命令が拡張されます。

SR レジスタの DMX ビットをセットすると X アドレスレジスタが、DMY ビットをセットすると Y アドレスレジスタがそれぞれモジュロアドレッシングモードになります。モジュロアドレッシングは、X と Y アドレスレジスタどちらかに対してだけ有効です。両方を同時にモジュロアドレッシングモードにすることはできません。したがって、DMX と DMY を同時にセットしないでください。万一同時にセットされた場合には、DMY 側のみ有効となります*。DMX と DMY ビットは、SR レジスタに対する STC 命令および LDC 命令で設定できます。SETDMX、SETDMY、CLRDMXY 命令の使用を推奨します。

モジュロアドレッシング制御中に例外を受理した場合は、MOD レジスタおよび SR レジスタの DMX と DMY ビットを退避してください。復帰時にこれらを回復することにより、モジュロアドレッシング制御に復帰することができます。

【注】 * この仕様は、将来変更される可能性があります。

表 6.12 モジュロアドレッシング制御命令

命令	動　　作	実行ステート
STC MOD,Rn	MOD→Rn	1
STC.L MOD,@-Rn	Rn-4→Rn, MOD→(Rn)	1
LDC.L @Rn+,MOD	(Rn)→MOD, Rn+4→Rn	4
LDC Rn,MOD	Rn→MOD	4
SETDMX	1→SR.DMX, 0→SR.DMY	1
SETDMY	0→SR.DMX, 1→SR.DMY	1
CLRDMXY	0→SR.DMX, 0→SR.DMY	1

モジュロアドレッシングの使用例を以下に示します。

```
MOV.L #H'70047000,R10   ; MS=H'7000 ME=H'7004 として
LDC R10, MOD            ; MOD レジスタに ME:MS を設定します。
STC SR,R10              ;
MOV.L #H'FFFFF3FF,R11
MOV.L #H'00000400,R12
AND R11,R10             ;
OR R12,R10              ;
LDC R10,SR              ; SR.DMX=1, SR.DMY=0。X モジュロアドレッシングモードを設定。
MOV.L #H'A5007000,R4
MOVX.W @R4+,X0          ;R4: H'A5007000→H'A5007002
MOVX.W @R4+,X0          ;R4: H'A5007002→H'A5007004
MOVX.W @R4+,X0          ;R4: H'A5007004→H'A5007000（ME と一致したので、MS が設定
                        されます）
MOVX.W @R4+,X0          ;R4: H'A5007000→H'A5007002
```

MS と ME に開始と終了アドレスを指定した後に DMX または DMY ビットを 1 にセットします。

DMX または DMY で指定された X または Y データ転送命令が実行されると、アドレスレジスタの更新前の値が ME と比較されます*。データ転送の後、比較結果が ME と一致していた場合、アドレスレジスタの更新後の値として、MS のスタートアドレスが代入されます。

X または Y データ転送命令のアドレッシングタイプが「更新なし」の場合は、たとえ ME と一致しても MS への復帰は行なわれません。また、X または Y データ転送命令のアドレッシングタイプが「インデクスレジスタ加算」の場合は、アドレスポインタは ME と一致せずにその値を超えてしまうことがあります。この場合は、アドレスポインタはモジュロ開始アドレスには戻りません。

最大のモジュロサイズは、64K バイトです。これは、X と Y データメモリをアクセスするには十分です。

【注】 * モジュロアドレッシングに限らず、DSP 命令による X と Y データアドレッシング時は、アドレスポインタ、インデックスレジスタ、MS、および ME のビット 0 には、必ず 0 を書き込んでください。

上記プログラミング例では、SR レジスタの DMX ビットをセットするのに STC および LDC 命令を使用しています。これらの命令列の代わりに、SETDMX 命令により 1 命令で DMY ビットのクリアおよび DMX ビットのセットを行うことができます。SETDMY 命令も同様に DMX ビットをクリアし、DMY ビットをセットします。CLRDMXY 命令は、DMX および DMY ビットをクリアし、モジュロアドレッシングモードを無効化します。

6.4.4 メモリのデータ形式

DSP 命令で扱えるメモリのデータ形式は、ワードとロングワードに分けられます。MOVS.W 命令で 2n 以外のアドレスから始まるワードデータをアクセスしようとしたり、または MOVS.L、LDS.L、および STS.L 命令で 4n 以外のアドレスから始まるロングワードにアクセスしようとするとアドレスエラーが発生します。このような場合は、アクセスするデータは保証されません。

MOVX.W および MOVY.W により 2n 以外のアドレスから始まるワードデータをアクセスしたときは、MMUCR.AT＝1 かつ RAMCR.RP＝1 のときのみアドレスエラーが発生します。MMUCR.AT＝0 または RAMCR.RP＝0 のときはアドレスエラーが発生しないので、必ず 2n 番地境界にアドレスを設定してください。2n 番地以外のアドレスに設定した場合は、アクセスするデータは保証されません。

また、MOVX.L および MOVY.L により 4n 以外のアドレスから始まるロングワードデータをアクセスしたときは、MMUCR.AT＝1 かつ RAMCR.RP＝1 のときのみアドレスエラーが発生します。MMUCR.AT＝0 または RAMCR.RP＝0 のときは、アドレスエラーが発生しないので、必ず 4n 境界にアドレスを設定してください。4n 番地以外のアドレスに設定した場合は、アクセスするデータは保証されません。

6.4.5 ダブル、シングルデータ転送命令の命令フォーマット

ダブルデータ転送命令の命令形式を**表 6.13** と**表 6.14** に、シングルデータ転送命令の命令形式を**表 6.15** に示します。

表 6.14 は、X メモリか Y メモリの一方の転送命令が NOPX または NOPY のときの拡張命令です。一方が NOPX または NOPY 以外のときは使用できません。

表 6.13 ダブルデータ転送の命令形式（1）

分類	ニーモニック	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X メモリデータ転送	NOPX	1	1	1	1	0	0	0		0		0		0	0		
	MOVX.W @Ax,Dx							Ax		Dx		0		0	1		
	MOVX.W @Ax+,Dx													1	0		
	MOVX.W @Ax+Ix,Dx													1	1		
	MOVX.W Da,@Ax									Da		1		0	1		
	MOVX.W Da,@Ax+													1	0		
	MOVX.W Da,@Ax+Ix													1	1		
Y メモリデータ転送	NOPY	1	1	1	1	0	0		0		0		0			0	0
	MOVY.W @Ay,Dy								Ay		Dy		0			0	1
	MOVY.W @Ay+,Dy															1	0
	MOVY.W @Ay+Iy,Dy															1	1
	MOVY.W Da,@Ay										Da		1			0	1
	MOVY.W Da,@Ay+															1	0
	MOVY.W Da,@Ay+Iy															1	1

【注】 Ax：0＝R4、1＝R5　Ay：0＝R6、1＝R7　Dx：0＝X0、1＝X1　Dy：0＝Y0、1＝Y1　Da：0=A0、1=A1

表 6.14　ダブルデータ転送の命令形式（2）

分類	ニーモニック	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X メモリデータ転送	MOVX.W @Axy,Dxy	1	1	1	1	0	0	Axy		Dxy		0	0	0	1	0	0
	MOVX.W @Axy+,Dxy													1	0		
	MOVX.W @Axy+Ix,Dxy													1	1		
	MOVX.W Dax,@Axy									Dax		1	0	0	1		
	MOVX.W Dax,@Axy+													1	0		
	MOVX.W Dax,@Axy+Ix													1	1		
	MOVX.L @Axy,Dxy									Dxy		0	1	0	1		
	MOVX.L @Axy+,Dxy													1	0		
	MOVX.L @Axy+Ix,Dxy													1	1		
	MOVX.L Dax,@Axy									Dax		1	1	0	1		
	MOVX.L Dax,@Axy+													1	0		
	MOVX.L Dax,@Axy+Ix													1	1		
Y メモリデータ転送	MOVY.W @Ayx,Dyx	1	1	1	1	0	0	Ayx		Dyx		0	0	0	0	0	1
	MOVY.W @Ayx+,Dyx															1	0
	MOVY.W @Ayx+Iy,Dyx															1	1
	MOVY.W Day,@Ayx									Day		0	1			0	1
	MOVY.W Day,@Ayx+															1	0
	MOVY.W Day,@Ayx+Iy															1	1
	MOVY.L @Ayx,Dyx									Dyx		1	0			0	1
	MOVY.L @Ayx+,Dyx															1	0
	MOVY.L @Ayx+Iy,Dyx															1	1
	MOVY.L Day,@Ayx									Day		1	1			0	1
	MOVY.L Day,@Ayx+															1	0
	MOVY.L Day,@Ayx+Iy															1	1

【注】　Axy：R0,R1,R4,R5=(01,11,00,10)　Ix=R8　Dxy：X0,X1,Y0,Y1=(00,10,01,11)　Dax：A0,A1,X0,X1=(00,10,01,11)

Ayx：R2,R3,R6,R7=(10,11,00,01)　Iy=R9　Dyx：Y0,Y1,X0,X1=(00,01,10,11)　Day：A0,A1,Y0,Y1=(00,01,10,11)

表 6.15　シングルデータ転送命令の命令形式

分類	ニーモニック	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
シングルデータ転送	MOVS.W @-As,Ds	1	1	1	1	0	1	As		Ds		0:(*)		0	0	0	0
	MOVS.W @As,Ds							0:R4				1:(*)		0	1		
	MOVS.W @As+,Ds							1:R5				2:(*)		1	0		
	MOVS.W @As+Is,Ds							2:R2				3:(*)		1	1		
	MOVS.W Ds,@-As							3:R3				4:(*)		0	0	0	1
	MOVS.W Ds,@As											5:A1		0	1		
	MOVS.W Ds,@As+											6:(*)		1	0		
	MOVS.W Ds,@As+Is											7:A0		1	1		
	MOVS.L @-As,Ds											8:X0		0	0	1	0
	MOVS.L @As,Ds											9:X1		0	1		
	MOVS.L @As+,Ds											A:Y0		1	0		
	MOVS.L @As+Is,Ds											B:Y1		1	1		
	MOVS.L Ds,@-As											C:M0		0	0	1	1
	MOVS.L Ds,@As											D:A1G		0	1		
	MOVS.L Ds,@As+											E:M1		1	0		
	MOVS.L Ds,@As+Is											F:A0G		1	1		

【注】* システム予約コード

6.5　DSP データ演算命令

6.5.1　DSP レジスタ

SH4AL-DSP は、**図 6.3** に示すように DSP レジスタとして 8 つのデータレジスタ（A0、A1、X0、X1、Y0、Y1、M0、M1）と 1 つのコントロールレジスタ（DSR）を持っています。

表 6.16 および**表 6.17** は、DSP 命令で使用するレジスタのデータタイプを示します。命令コードの制限のため、表に示すレジスタの中には使用できない演算もあります。たとえば、PMULS はソースレジスタに A1 を使用できますが、A0 は使用できません。これらの表は、レジスタの選択性の詳細については省略しています。

表 6.16　DSP 命令のデスティネーションレジスタ

レジスタ		命令	ガードビット	レジスタビット	
			39 32	31 16	15 0
A0、A1	DSP 演算	固定小数点、PSHA、PMULS、PSWAP	（符号拡張）	40 ビット結果	
		整数、PDMSB	（符号拡張）	24 ビット結果	0クリア
		論理、PSHL	0 クリア	16 ビット結果	0 クリア
	データ転送	MOVS.W	符号拡張	16 ビットデータ	0 クリア
		MOVS.L	符号拡張	32 ビットデータ	
A0G、A1G	データ転送	MOVS.W	データ	更新しない	
		MOVS.L	データ	更新しない	
X0、X1 Y0、Y1 M0、M1	DSP 演算	固定小数点、PSHA、PMULS、PSWAP		32 ビット結果	
		整数、論理、PDMSB、PSHL		16 ビット結果	0 クリア
	データ転送	MOVX/Y.W、MOVS.W		16 ビットデータ	0 クリア
		MOVX/Y.L、MOVS.L		32 ビットデータ	

表 6.17　DSP 命令のソースレジスタ

レジスタ		命令	ガードビット	レジスタビット	
			39 32	31 16	15 0
A0、A1	DSP 演算	固定小数点、PDMSB、PSHA	40 ビットデータ		
		整数	24 ビットデータ		
		論理、PSHL、PMULS		16 ビットデータ	
		PSWAP		32 ビットデータ	
	データ転送	MOVX/Y.W、MOVS.W		16 ビットデータ	
		MOVS.L		32 ビットデータ	
A0G、A1G	データ転送	MOVS.W	データ		
		MOVS.L	データ		
X0、X1 Y0、Y1 M0、M1	DSP 演算	固定小数点、PDMSB、PSHA	符号*	32 ビットデータ	
		整数	符号*	16 ビットデータ	
		論理、PSHL、PMULS		16 ビットデータ	
		PSWAP		32 ビットデータ	
	データ転送	MOVS.W		16 ビットデータ	
		MOVX/Y.L、MOVS.L		32 ビットデータ	

【注】　*　データを符号拡張し、ALU に入力する。

DSR は、DSP データ演算結果の状態（ゼロ、負、など）を保持します。**表 6.18** に DSR レジスタの各ビットの説明を示します。DSR は、また CPU の T ビットに類似した DC ビットを持っており、状態フラグを示します。条件付き DSP データ演算命令は、この DC ビットに基づいて実行を制御します。この制御は、DSP ユニットの命令にのみ影響を与えます。すなわち、DSP レジスタのみを制御し、アドレスレジスタの更新、およびロードやストア命令などの CPU の命令を制御することはできません。DC 状態選択ビット（CS[2:0]）には、DC ビットに反映する条件を指定します。

PMULS、PLDS、PSTS、MOVX、MOVY、および MOVS を除く無条件 DSP タイプのデータ演算命令は条件フラグと DC ビットを更新しますが､CPU 命令はどれも DC ビットを更新しません｡条件付き DSP タイプ命令も DSR を更新することはありません。

表 6.18　DSR レジスタのビットの説明

ビット	ビット名	初期値	説　　明
31～16	—	すべて 0	リザーブビット 読み出すと常に 0 が読み出されます。書き込み時も常に 0 にしてください。
15	AGT	0	累積符号付き大ビット 1：演算結果が正（ゼロを除く）、またはオペランド 1 がオペランド 2 より大きいことを表します。
14	AZ	0	累積ゼロビット 1：演算結果が 0（ゼロ）、またはオペランド 1 がオペランド 2 と等しいことを示します。
13	AN	0	累積負ビット 1：演算結果が負、またはオペランド 1 がオペランド 2 より小さいことを表します。
12	AV	0	累積オーバフロービット 1：演算結果がオーバフローしたことを表します。
11～9	TS	すべて 0	T ビット状態選択 TC ビットが 1 のとき、SR レジスタの T ビットに設定する演算結果状態を選択するモードを指定します。ただし、SR レジスタの S ビットが 1 のときもオーバフロー検出を行います。 000：キャリー／ボローモード 001：負値モード 010：ゼロ値モード 011：オーバフローモード 100：符号付き大モード 101：符号付き以上モード 110：リザーブ（設定禁止） 111：リザーブ（設定禁止）
8	TC	0	TC ビット 0：SR レジスタの T ビットは DSP 命令に依存しません。 1：SR レジスタの T ビットは DSP 命令実行時、DSR レジスタの T S ビットの状態により変化します。T ビットに設定できる命令は、DC ビットを更新する命令に限られます。

ビット	ビット名	初期値	説　　明
7	GT	0	符号付き大ビット 演算結果が正（ゼロを除く）、またはオペランド 1 がオペランド 2 より大きいことを表します。 1：演算結果が正、またはオペランド 1 がオペランド 2 より大きい
6	Z	0	ゼロビット 演算結果が 0（ゼロ）、またはオペランド 1 がオペランド 2 と等しいことを表します。 1：演算結果が 0（ゼロ）、または等しい
5	N	0	負値ビット 演算結果が負、またはオペランド 1 がオペランド 2 より小さいことを表します。 1：演算結果が負、またはオペランド 1 がオペランド 2 より小さい
4	V	0	オーバフロービット 演算結果がオーバフローしたことを表します。 1：演算結果がオーバフロー
3〜1	CS	すべて 0	DC ビット状態選択ビット DC ビットに設定する演算結果状態を選択するためのモードを指定します。 000：キャリー／ボローモード 001：負値モード 010：ゼロモード 011：オーバフローモード 100：符号付き大モード 101：符号付き以上モード 110：リザーブ（設定禁止） 111：リザーブ（設定禁止）
0	DC	0	DSP 状態ビット CS ビットで指定されたモードで演算結果の状態を設定します。 0：指定されたモードの状態が成立しない 1：指定されたモードの状態が成立する PADDC または PSUBC 命令実行後の DC ビットは、CS ビットに関係なくキャリー／ボローモードで演算結果の状態を設定します。

DSR は、システムレジスタに割り当てられられます。DSR には、次のロードまたはストア命令が用意されています。

```
STS DSR,Rn;
STS.L DSR,@-Rn;
LDS Rn,DSR;
LDS.L @Rn+,DSR;
```

STS 命令で DSR を読み出すとき、上位ビット（ビット 31～ビット 16）はすべて 0 になります。

DSR レジスタの条件コードビット（DC）は、常に無条件の ALU またはシフト演算命令の結果に基づいて更新されます。乗算命令、PLDS、PSTS 命令、MOVX、MOVY、MOVS 命令、および条件付き命令の場合は、DC ビットを更新しません。DC ビットの更新は、DSR レジスタの CS[2:0]ビットにより行われます。表 6.19 に DC ビットの更新ルールについて示します。

表 6.19　DC ビットの更新ルール

CS[2:0]			条件モード	説　　明
0	0	0	キャリーまたは ボローモード	ALU 算術演算の結果、キャリーまたはボローが発生した場合は、DC ビットがセットされます。それ以外は 0 クリアされます。 シフト命令、PSHA または PSHL の実行時、最後にシフトアウトしたビットデータが DC ビットにコピーされます。 ALU 論理演算の実行時は、DC ビットは常に 0 クリアされます。
0	0	1	負値モード	ALU 算術演算または算術シフト（PSHA）演算の実行時は、ガードビット部分を含めて結果の MSB が DC ビットにコピーされます。 ALU 論理演算または論理シフト（PSHL）演算の実行時、ガードビット部分を除く結果の MSB が DC ビットにコピーされます。
0	1	0	ゼロ値モード	ALU またはシフト演算の結果がすべてゼロの場合は、DC ビットがセットされます。それ以外は 0 クリアされます。
0	1	1	オーバフローモード	ALU 算術演算または算術シフト（PSHA）の演算結果がガードビット部分を除いたデスティネーションレジスタの範囲を越える場合は、DC ビットがセットされます。それ以外は 0 クリアされます。 ALU 論理演算または論理シフト（PSHL）の演算の実行時は、DC ビットは常に 0 クリアされます。
1	0	0	符号付き大モード	このモードは符号付き以上モードに類似していますが、結果がすべて 0 の場合は、DC は 0 クリアされます。 DC = ~{ (負値^オーバレンジ) |ゼロ値}　　; 算術演算の場合 DC = 0　　; 論理演算の場合
1	0	1	符号付き以上モード	ALU またはシフト（PSHA）の算術演算の結果がガードビットを含んだデスティネーションレジスタの範囲を超える場合（オーバレンジと呼ぶ）は、定義は負値モードと同じになります。オーバレンジでない場合は、定義は負値モードの反転となります。 ALU またはシフト（PSHL）の論理演算の実行時は、DC ビットは常に 0 クリアされます。 DC = ~(負値^オーバレンジ)　　; 算術演算の場合 DC = 0　　; 論理演算の場合
1	1	0	リザーブ（設定禁止）	
1	1	1	リザーブ（設定禁止）	

DSR にはフラグを累積するフラグアキュムレート機能があり、DSR[11:8]（TS、TC）で指定します。DSR[11:8]≠0000 のとき、AGT、AZ、AN、および AV は演算結果によって値が更新されます。ただし、演算結果により 0 クリアされることはありません。本機能により、ある一連の処理の中で発生したフラグ結果を累積することができます。フラグアキュムレート機能の使用例を**図 6.6** に示します。

```
MOV.L     #H'7FFFFFFF,R0
LDS       R0,X0
LDS       R0,Y0
MOV.W     #H'0200,R0
LDS       R0,DSR          ;AGT,AZ,AN,AV=0000   TS,TC=0010   GT,Z,N,V=0000
PADD      X0,Y0,A0        ;AGT,AZ,AN,AV=1001   TS,TC=0010   GT,Z,N,V=1001
PCLR      A0              ;AGT,AZ,AN,AV=1101   TS,TC=0010   GT,Z,N,V=0100
```

図 6.6　フラグアキュムレート機能の使用例

DSR には DSP データ演算結果を SR レジスタの T ビットに反映する T ビットリンク機能があります。

DSR レジスタの TC ビットと TS ビットにより、T ビットに 1 をセットするか否かの制御をすることができます。

T ビットに設定できる命令は、DC ビットを更新する命令に限られます。PMULS、PSTS、MOVX、MOVY、MOVS および条件付き DSP データ演算命令は T ビットを更新しません。

本機能により CS ビットと TS ビットを独立に設定することで、2 つの条件での異なる処理を行うことが可能です。T ビットは DC ビットと類似していますが、T ビットと DC ビットの相違点は次の通りです。

- オーバフロー防止機能が有効のとき（S＝1）、CSビットでオーバフローモードを選択する場合、DCビットは0でクリアされますが、TSビットでオーバフローを選択する場合、オーバフロー検出結果がTビットに反映されます。
- PADDC、PSUBC命令実行後のDCビットは、CSビットに関係なくキャリー／ボローモードになりますが、TビットはTSビットで設定した状態が反映されます。

図 6.7 に T ビットリンク機能の使用例を示します。

DSR に、TC ビットをゼロ値モード、DC ビットをキャリー／ボローモードに設定します。

PADD 演算結果に従って T ビットが設定され、後続の BT 命令により TRGET_T に分岐します。

```
MOV.L     #H'0000 0000,R1
LDS       R1,X0
LDS       R1,Y0

MOV.L     #H'0000 0500,R0  ;  DSR TS,TC=0101,CS=000
LDS       R0,DSR

PADD      X,Y0,A0  ;  T=1 DC=0
BT        TRGET_T
```

図 6.7　T ビットリンク機能の使用例

6.5.2 DSP データ演算命令の命令セット

DSP データ演算命令は、DSP ユニットで処理されるデジタル信号処理の命令です。これらの命令は 32 ビット長の命令コードで、複数の命令を並列に実行します。命令コードは A フィールド、B フィールドの 2 つに分かれており、A フィールドにはダブルデータ転送命令を指定し、B フィールドにはシングルまたはダブルデータ演算命令を指定します。命令は独立して指定することができ、実行も独立に実行されます。

B フィールドのデータ演算命令は 3 つに分かれています。ダブルデータ演算命令、条件付きシングルデータ演算命令、および無条件シングルデータ演算命令の 3 つです。DSP 演算命令の命令形式を**表 6.20** に示します。それぞれのオペランドは独立に DSP レジスタから選べます。DSP 演算命令のオペランドとレジスタの対応を**表 6.21** に示します。

表 6.20 DSP 演算命令の命令形式

分類	命令形式
ダブルデータ演算命令	`ALUop. Sx, Sy, Du MLTop. Se, Sf, Dg`

条件付きシングルデータ演算命令	`DCT ALUop. Sx, Sy, Dz`
	`DCF ALUop. Sx, Sy, Dz`
	`DCT ALUop. Sx, Dz`
	`DCF ALUop. Sx, Dz`
	`DCT ALUop. Sy, Dz`
	`DCF ALUop. Sy, Dz`

無条件シングルデータ演算命令	`ALUop. Sx, Sy, Dz`
	`ALUop. Sx, Dz`
	`ALUop. Sy, Dz`
	`MLTop. Se, Sf, Dg`

表 6.21 DSP 命令のオペランドとレジスタの対応

レジスタ	ALU、シフト演算				乗算演算		
	Sx	Sy	Dz	Du	Se	Sf	Dg
A0	Yes		Yes	Yes			Yes
A1	Yes		Yes	Yes	Yes	Yes	Yes
M0		Yes	Yes				Yes
M1		Yes	Yes				Yes
X0	Yes		Yes	Yes	Yes	Yes	
X1	Yes		Yes		Yes		
Y0		Yes	Yes	Yes	Yes	Yes	
Y1		Yes	Yes			Yes	

DSP データ演算命令を書くときは、最初に B フィールドの命令を書いて、次に A フィールドの命令を書きます。DSP データ演算命令による並行処理プログラム例を**図 6.8** に示します。

```
       PADD  A0, M0, A0   PMULS  X0, Y0, M0    MOVX.W  @R4+,    X0    MOVY.W  @R6+,  Y0  [;]
DCF    PINC  M1, A1                           MOVX.W  @R5+R8,  X0    MOVY.W  @R7+,  Y1  [;]
       PCMP  M1, M0                           MOVX.W  @R4,     X1    [NOPY]  [;]
```

図 6.8　DSP データ演算命令による並行処理プログラムの例

ここで、[　]は省略可能な部分を表します。

NOPX と NOPY のノーオペレーション命令は、省略可能です。DSP データ演算命令の B フィールドの詳細は、**表 6.39** を参照してください。

- **条件付き演算とデータ転送**

このクラスに属する命令の中には前記のように、条件付きで実行することができるものがあります。ただし、指定した条件は命令の B フィールドに対してのみ有効であって、並行して指定したデータ転送命令には有効ではありません。**図 6.9** に例を示します。

```
DCT    PADD   X0,     Y0,     A0          MOVX.W    @R4+,   X0    MOVY.W   A0,   @R6+R9    [;]
  ＜条件が真の場合＞
実行前：  X0=H'33333333,      Y0=H'55555555,      A0=H'123456789A,
          R4=H'00008000,      R6=H'00005000,      R9=H'00000004
            (R4)=1111      (R6)=2222
実行後：  X0=H'11110000,      Y0=H'55555555,      A0=H'0088888888
          R4=H'00008002,      R6=H'00005004,      R9=H'00000004
            (R4)=1111      (R6)=3456

  ＜条件が偽の場合＞
実行前：  X0=H'33333333,      Y0=H'55555555,      A0=H'123456789A
          R4=H'00008000,      R6=H'00005000,      R9=H'00000004
            (R4)=1111      (R6)=2222
実行後：  X0=H'11110000,      Y0=H'55555555,      A0=H'123456789A
          R4=H'00008002,      R6=H'00005004,      R9=H'00000004
            (R4)=1111      (R6)=3456
```

図 6.9　条件付き演算とデータ転送命令の例

- **NOPXおよびNOPYの命令コードの割当て**

DSP 演算命令と同時に並行処理されるデータ転送命令がないときは、データ転送命令に NOPX または NOPY 命令を書くか、あるいは命令を省略することもできます。NOPX または NOPY 命令を書くかあるいは省略しても命令コードは同じです。NOPX と NOPY の命令コードの例を**表 6.22** に示します。

表 6.22　NOPX と NOPY の命令コードの例

命令			コード
PADD X0,Y0,A0	MOVX.W @R4+,X0	MOVY.W @R6+R9,Y0	1111100000001011 1011000100000111
PADD X0,Y0,A0	NOPX	MOVY.W @R6+R9,Y0	1111100000000011 1011000100000111
PADD X0,Y0,A0	NOPX	NOPY	1111100000000000 1011000100000111
PADD X0,Y0,A0	NOPX		1111100000000000 1011000100000111
PADD X0,Y0,A0			1111100000000000 1011000100000111
	MOVX.W @R4+,X0	MOVY.W @R6+R9,Y0	1111000000001011
	MOVX.W @R4+,X0	NOPY	1111000000001000
	MOVS.W @R4+,X0 NOPX	 MOVY.W @R6+R9,Y0	1111010010001000 1111000000000011
		MOVY.W @R6+R9,Y0	1111000000000011
	NOPX	NOPY	1111000000000000
NOP			0000000000001001

6.5.3 SP タイプデータ形式

SH4AL-DSP は、命令によって異なるデータ形式を持っています。ここでは DSP タイプ命令用のデータ形式について説明します。

図 6.10 に 2 進小数点の位置の異なる 3 つの DSP タイプのデータ形式を、また参考として、ビット 0 の右側に 2 進小数点を持つ CPU タイプのデータ形式を示します。

DSP タイプ固定小数点データ形式は、ビット 31 とビット 30 の間に 2 進小数点があります。DSP タイプ整数形式は、ビット 16 とビット 15 の間に 2 進小数点があります。DSP タイプ論理形式には、2 進小数点はありません。データ形式の有効なデータ長は、命令および DSP レジスタによって異なります。

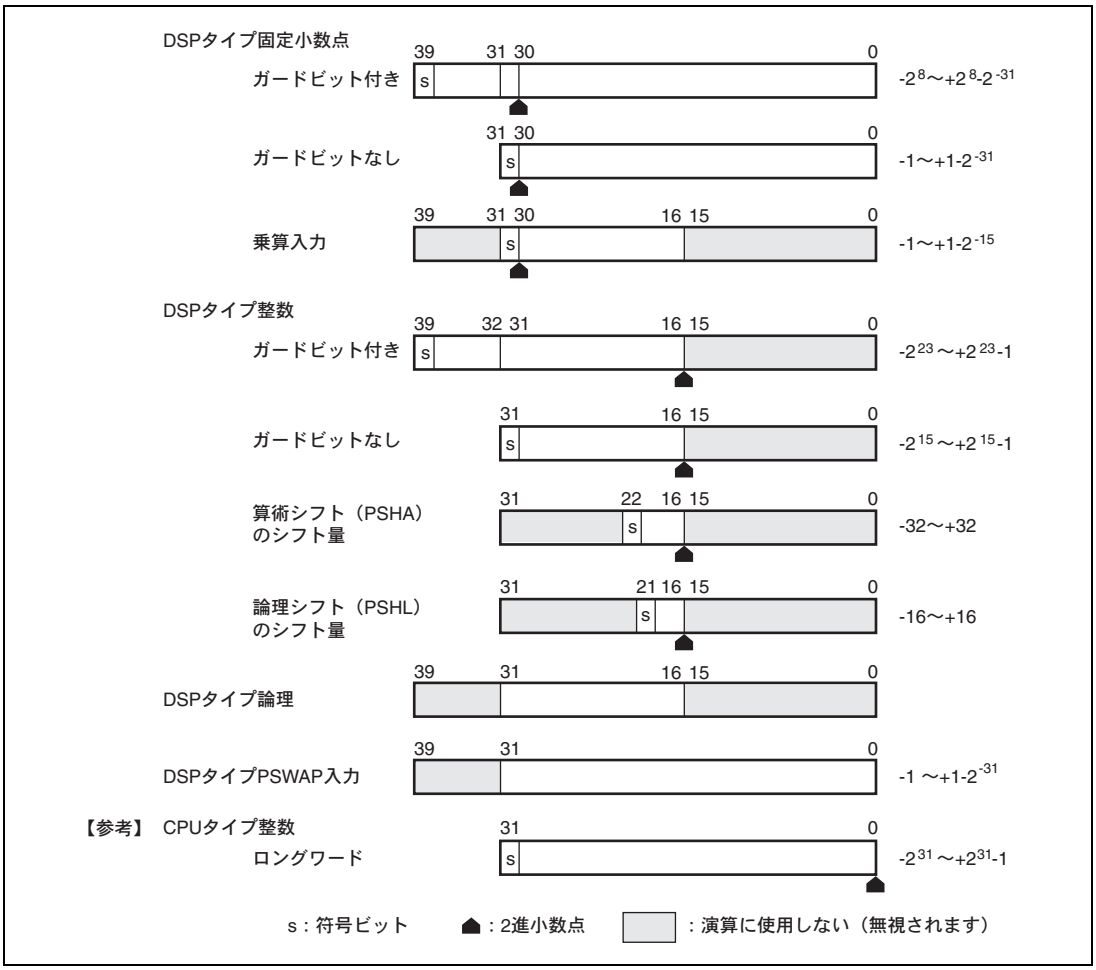


図 6.10 データ形式

算術シフト（PSHA）命令のシフト量は、−64～＋63 を表す 7 ビットのフィールドを持っていますが、−32～＋32 が有効な数です。また論理シフトのシフト量も 6 ビットのフィールドを持っていますが、−16～＋16 が有効な数です。無効な数値を指定した場合の結果は保証されません。

6.5.4 ALU 固定小数点算術演算

図 6.11 に ALU 固定小数点算術演算フローを示します。**表** 6.23 はこの演算の種々のタイプを示し、**表** 6.24 は各オペランドとレジスタの対応を示します。

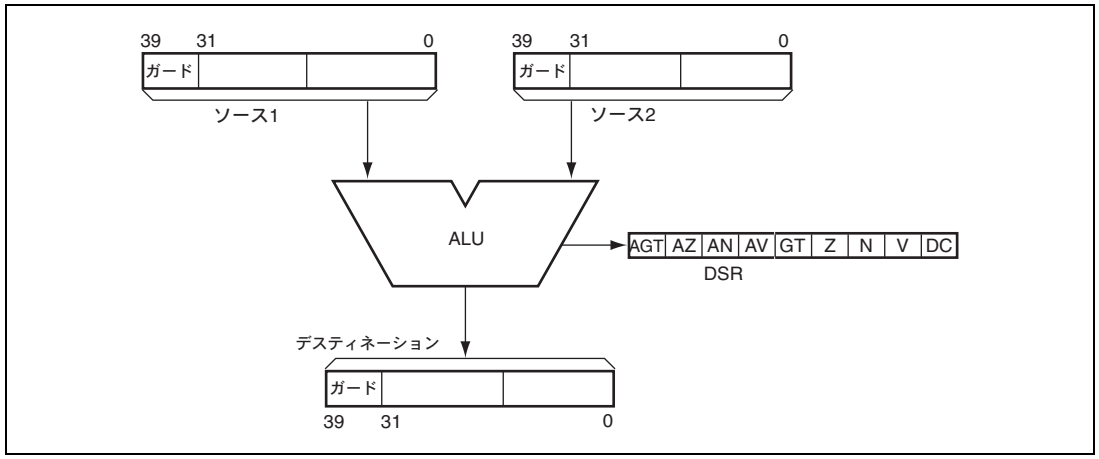


図 6.11 ALU 固定小数点算術演算フロー

ALU 固定小数点算術演算は、基本的に 40 ビット演算であり、32 ビットの基本精度部分および 8 ビットのガードビット部分から構成されます。したがって、ガードビット部分を提供していないレジスタをソースオペランドに指定すると、符号ビットがガードビット部分にコピーされます。ガードビット部分を提供していないレジスタをデスティネーションオペランドに指定すると、演算結果の下位 32 ビットがデスティネーションレジスタに入力されます。

ALU 固定小数点算術演算は、レジスタ間で実行されます。各ソースオペランドおよびデスティネーションオペランドは、DSP レジスタの 1 つから独立して選択されます。ガードビットを持つレジスタをオペランドに指定すると、ガードビットも含めてこれらの演算が実行されます。

表 6.23　ALU 固定小数点算術演算の種類

ニーモニック	機能	ソース 1	ソース 2	デスティネーション
PADD	加算	Sx	Sy	Dz（Du）
PSUB	減算	Sx	Sy	Dz（Du）
PADDC	キャリー付き加算	Sx	Sy	Dz
PSUBC	ボロー付き減算	Sx	Sy	Dz
PCMP	比較	Sx	Sy	−
PCOPY	データコピー	Sx	all 0	Dz
		all 0	Sy	Dz
PABS	絶対値	Sx	all 0	Dz
		all 0	Sy	Dz
PNEG	符号反転	Sx	all 0	Dz
		all 0	Sy	Dz
PCLR	クリア	all 0	all 0	Dz（Du）

表 6.24　オペランドとレジスタの対応

レジスタ	Sx	Sy	Dz	Du
A0	Yes		Yes	Yes
A1	Yes		Yes	Yes
M0		Yes	Yes	
M1		Yes	Yes	
X0	Yes		Yes	Yes
X1	Yes		Yes	
Y0		Yes	Yes	Yes
Y1		Yes	Yes	

図 6.12 に示すように、ALU 演算と同じラインでプログラムされたデータロード命令によりメモリから読み込まれたデータは、データロード命令のデスティネーションオペランドが ALU 演算のソースオペランドと同一であってもこの演算用のソースオペランドとしては使用されません。この場合は、前の命令の結果が ALU 演算のソースオペランドとして用いられた後にデータロード演算のデスティネーションオペランドとして更新されます。

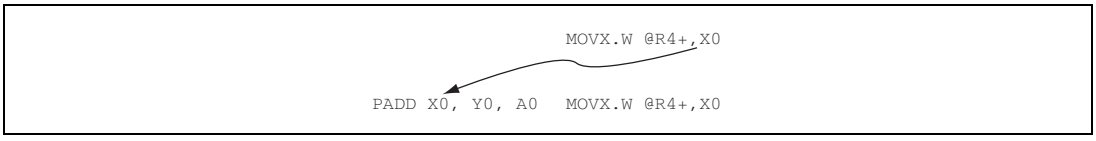


図 6.12　演算シーケンスの例

ALU 算術演算を実行するたびに、DSR の DC、N、Z、V、GT、AN、AZ、AV、および AGT ビットは基本的に演算結果に従って更新されます。また、DSR[11:8]≠0000 のとき、AGT、AZ、AN、AV ビットは演算結果に従って累積されます。ただし、条件付き命令の場合は、指定条件が真で演算が実行されてもこれらのビットは更新されません。無条件命令の場合は、これらは演算結果に従って常に更新されます。DC ビットの定義は、DSR レジスタの CS[2:0]（条件選択）で指定します。DC ビットは、以下のようになります。

(1) キャリー／ボローモード（CS[2:0]＝000 の場合）

DC ビットは、ガードビット部分を除いた演算結果の最上位ビットからキャリーまたはボローが発生したことを示します。いくつかの例を図 6.13 に示します。このモードがデフォルトです。PABS および PNEG 命令では、入力データが負のとき LSB（Least significant bit）に 1 を加算するため、キャリービットが発生する場合があります。

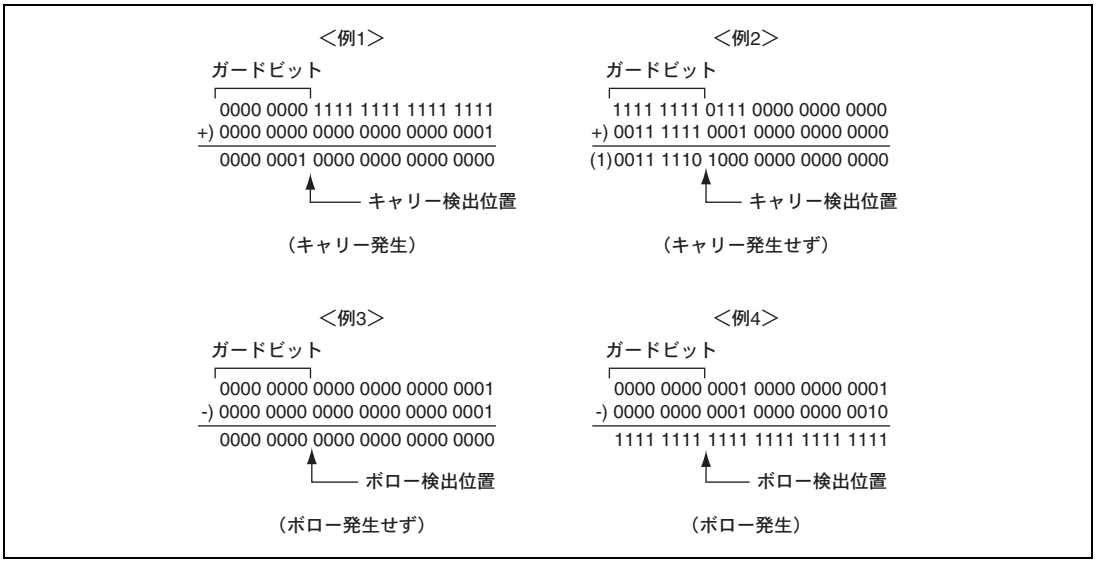


図 6.13　キャリー／ボローモードでの DC ビット生成の例

(2) 負値モード（CS[2:0]＝001 の場合）

DC ビットは、演算結果の MSB（Most significant bit）と同じ状態を示します。結果が負の数のときは、DC ビットは 1 を示します。結果が 0 または正の数のときは、DC ビットは 0 を示します。ALU は常に 40 ビットの算術演算を実行するので、正か負かを検出する符号ビットはデスティネーションオペランドに関係なく常に演算結果の MSB から得られます。いくつかの例を図 6.14 に示します。

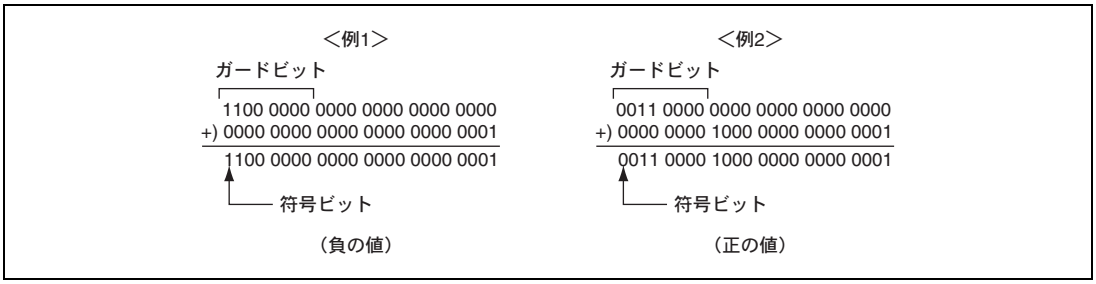


図 6.14　負値モードでの DC ビット生成の例

(3) ゼロ値モード（CS[2:0]＝010 の場合）

DC ビットは、演算結果がゼロであるか否かを示します。結果がゼロの場合は、DC ビットは 1 を示します。結果がゼロでない場合は、DC ビットは 0 を示します。

(4) オーバフローモード（CS[2:0]＝011 の場合）

DC ビットは、結果にオーバフローが発生したか否かを示します。ガードビットを除き演算の結果がデスティネーションレジスタの範囲を越える場合は、DC ビットが 1 にセットされます。ガードビットがある場合でも、DC ビットはガードビットがない場合の結果を示します。したがって、ガードビットの部分が大きな数を表すために使用される場合は、DC ビットは常に 1 にセットされます。オーバフローモードでの DC ビット生成の例を図 6.15 に示します。

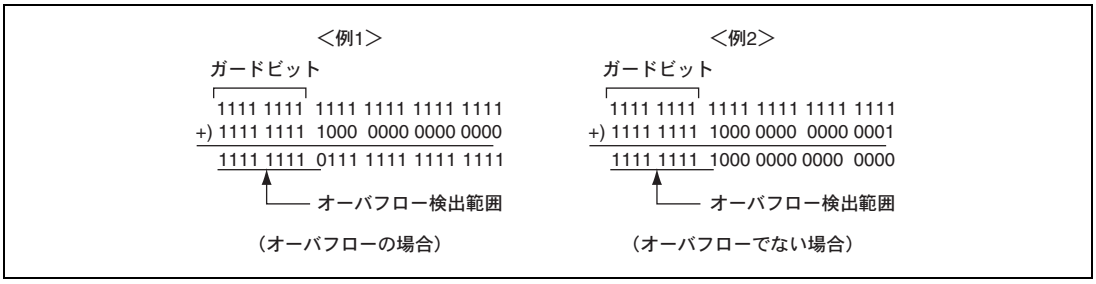


図 6.15　オーバフローモードでの DC ビット生成の例

(5) 符号付き大モード（CS[2:0]＝100 の場合）

DC ビットは、比較演算 PCMP の結果、ソース 1 データ（符号付き）がソース 2 データ（符号付き）より大きいかどうかを示します。ソース 1 データがソース 2 データより大きい場合は、比較演算の結果は正の値なので、このモードは前述の負値モードに類似しています。ただし、ソース 1 データがソース 2 データより大きくても、比較演算の結果がガードビットを含めたデスティネーションオペランドの範囲を越える場合（「オーバレンジ」と呼ぶ）は、結果の符号ビットは負の値を示します。この条件モードでは、この特殊な場合を考慮した上で DC ビットを更新します。次の式は、この条件を得る定義を示します。

DC = ~{ (負値^オーバレンジ) |ゼロ値}

PCMP 演算をこの条件モードで実行する場合は、DC ビットの結果は CPU 命令の CMP/GT 演算の T ビットの結果と同じです。このモードでは、PCMP 命令以外でも上記定義に従って DC ビットが更新されます。

(6) 符号付き以上モード（CS[2:0]＝101 の場合）

DC ビットは、比較演算 PCMP の結果、ソース 1 データ（符号付き）がソース 2 データ（符号付き）以上であるかどうかを示します。このモードは前述の「符号付き大モード」と類似していますが、このモードには等しい場合も含まれます。次の式は、この条件を得る定義を示します。

DC = ~(負値^オーバレンジ)

PCMP 演算をこの条件モードで実行する場合は、DC ビットの結果は CPU 命令の CMP/GE 演算の T ビット結果と同じです。このモードでは、PCMP 命令以外でも上記定義に従って DC ビットが更新されます。

N ビットは、CS[2:0]ビットが負値モードとしてセットされる DC ビットと常に同じ状態を示します。上記の負値モード部分を参照してください。Z ビットは、CS[2:0]ビットがゼロ値モードとしてセットされる DC ビットと常に同じ状態を示します。上記のゼロ値モード部分を参照してください。V ビットは、CS[2:0]ビットがオーバフローモードとしてセットされる DC ビットと常に同じ状態を示します。上記のオーバフローモード部分を参照してください。GT ビットは、CS[2:0]ビットが符号付き大モードとしてセットされる DC ビットと常に同じ状態を示します。上記の符号付き大モード部分を参照してください。

【注】 DC ビットは、PADDC と PSUBC 命令では、CS[2:0]の状態に関係なく常にキャリー／ボローモードとなります。

- **オーバフローの防止機能**

SR レジスタの S ビットは、DSP ユニットのどの ALU 固定小数点算術演算に対しても有効です。詳細については、**「6.5.12 オーバフロー防止機能」**を参照してください。

6.5.5 ALU 整数演算

図 6.16 に ALU 整数演算フローを示します。表 6.25 にこの演算の種類を示します。各オペランドのレジスタとの対応は、表 6.24 に示した ALU 固定小数点算術演算と同じです。

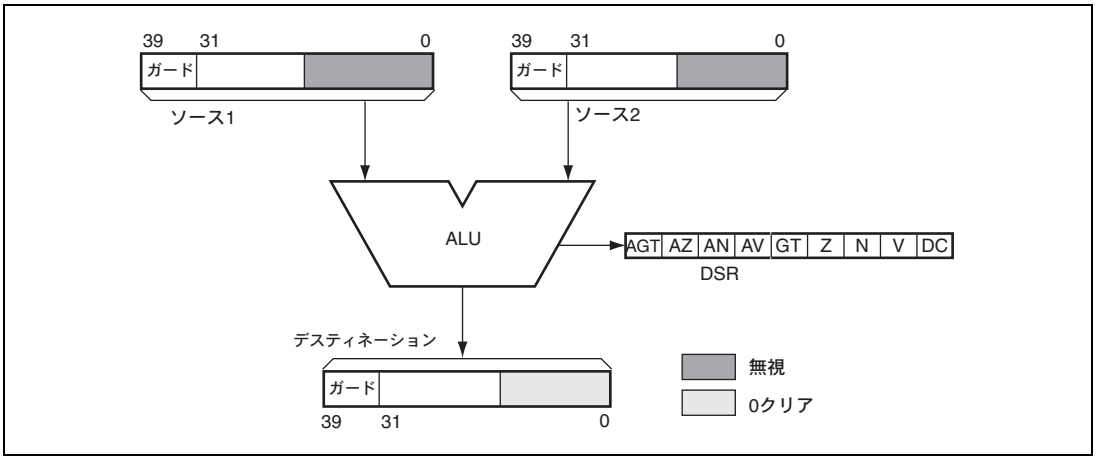


図 6.16　ALU 整数演算フロー

表 6.25　ALU 整数演算の種類

ニーモニック	機能	ソース 1	ソース 2	デスティネーション
PINC	1 ずつインクリメント	Sx	＋1	Dz
		＋1	Sy	Dz
PDEC	1 ずつデクリメント	Sx	－1	Dz
		－1	Sy	Dz

ALU 整数演算は、基本的に 24 ビット演算、すなわち上位 16 ビットの基本精度および 8 ビットのガードビット部分からなります。したがって、ガードビット部分を提供していないレジスタをソースオペランドに指定すると、符号ビットはガードビット部分にコピーされます。ガードビット部分を提供していないレジスタをデスティネーションオペランドに指定すると、演算結果のガードビットを除いた上位ワードがデスティネーションレジスタの上位ワードに入力されます。

ALU 整数演算では、ソースオペランドの下位ワードは無視され、デスティネーションオペランドの下位ワードは自動的にクリアされます。ガードビット部分がサポートされている場合は、ALU 整数演算で有効です。その他は、基本的に ALU 固定小数点演算の演算と同じです。ただし、表 6.25 に示すように、この種の演算は 2 種類の命令しか提供されません。したがって、第 2 オペランドは、実質的には＋1 か－1 かのいずれかとなります。ワードデータを DSP ユニットのレジスタに読み込むと、上位ワードデータとして入力されます。ガードビットがあるレジスタをオペランドに指定すると、ガードビットも有効です。

ALU 整数演算を実行するたびに、DSR レジスタの DC、N、Z、V、GT、AN、AZ、AV、および AGT ビットは、基本的に演算結果に従って更新されます。また、DSR[11:8]≠0000 のとき、AGT、AZ、AN、AV ビットは演算結果に従って累積されます。これは固定小数点演算と同じですが、各ソースオペランドとデスティネーションオペランドの下位ワードはそれらを生成するためには使用しません。詳細については、**「6.5.4 ALU 固定小数点算術演算」**を参照してください。

条件付き命令の場合は、指定した条件が真であり演算が実行されてもこれらのビットは更新されません。無条件命令の場合、これらは、演算結果に従って常に更新されます。詳細については、**「6.5.4 ALU 固定小数点算術演算」**を参照してください。

- **オーバフローの防止機能**

SR レジスタの S ビットは、DSP ユニットのすべての ALU 整数演算で有効です。詳細については、**「6.5.12 オーバフロー防止機能」**を参照してください。

6.5.6 ALU 論理演算

図 6.16 に ALU 論理演算フローを示します。**表 6.26** にこの演算の種類を示します。各オペランドのレジスタとの対応は、**表 6.24** に示した ALU 固定小数点算術演算と同じです。

ALU 論理演算は、レジスタ間で実行します。各ソースオペランドおよびデスティネーションオペランドは、DSP レジスタの 1 つから独立して選択されます。**図 6.17** に示すように、この種の演算は、各オペランドの上位ワードのみを使用します。ソースオペランドの下位ワードとガードビットは無視され、デスティネーションオペランドの下位ワードとガードビットは自動的にクリアされます。

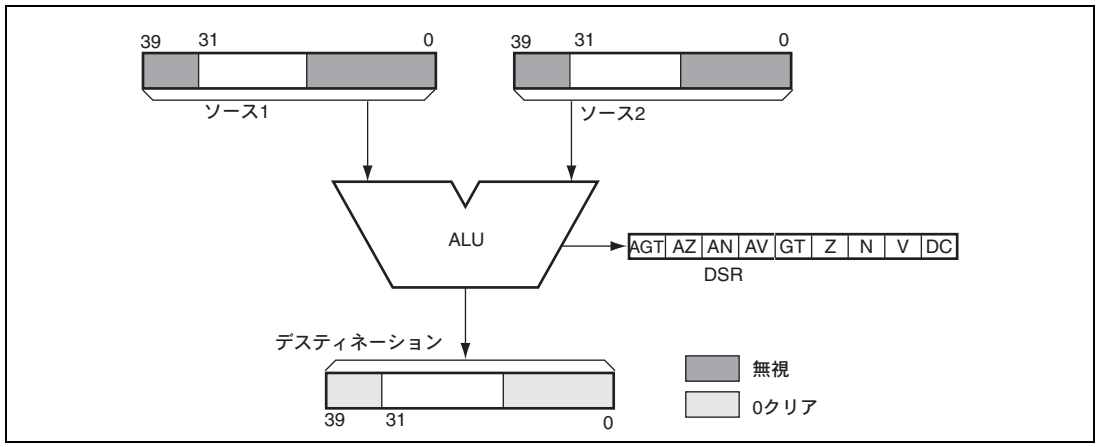


図 6.17 ALU 論理演算フロー

表 6.26 ALU 論理演算の種類

ニーモニック	機能	ソース 1	ソース 2	デスティネーション
PAND	論理 AND	Sx	Sy	Dz
POR	論理 OR	Sx	Sy	Dz
PXOR	論理排他的 OR	Sx	Sy	Dz

ALU 論理演算を実行するたびに、DSR レジスタの DC、N、Z、V、GT、AN、AZ、AV、および AGT ビットは、基本的に演算結果に従って更新されます。また、DSR[11:8]≠0000 のとき、AGT、AZ、AN、AV ビットは演算結果に従って累積されます。条件付き命令の場合は、指定条件が真で演算が実行されてもこれらのビットは更新されません。無条件命令の場合は、これらは演算結果に従って常に更新されます。DC ビットの定義は、DSR レジスタの CS[2:0]ビット（条件選択ビット）で指定します。DC ビットの結果は、次のとおりです。

1. キャリーまたはボローモード（CS[2:0]＝000の場合）

   DCビットは常に0にクリアされます。

2. 負値モード（CS[2:0]＝001の場合）

   演算結果のビット31の値がDCビットに読み込まれます。

3. ゼロ値モード（CS[2:0]＝010の場合）

   演算結果がゼロのときDCビットは1にセットされ、それ以外は0にクリアされます。

4. オーバフローモード（CS[2:0]＝011の場合）

   DCビットは常に0にクリアされます。

5. 符号付き大モード（CS[2:0]＝100の場合）

   DCビットは常に0にクリアされます。

6. 符号付き、以上モード（CS[2:0]＝101の場合）

   DCビットは常に0にクリアされます。

N ビットは、CS[2:0]ビットが負値モードとしてセットされる DC ビットと常に同じ状態を示します。上記の負値モード部分を参照してください。Z ビットは、CS[2:0]ビットがゼロ値モードとしてセットされる DC ビットと常に同じ状態を示します。上記のゼロ値モード部分を参照してください。V ビットは、CS[2:0]ビットがオーバフローモードとしてセットされる DC ビットと常に同じ状態を示しますが、本命令では、常に 0 クリアされます。上記のオーバフローモード部分を参照してください。GT ビットは、CS[2:0]ビットが符号付き大モードとしてセットされる DC ビットと常に同じ状態を示します。上記の符号付き大モード部分を参照してください。

6.5.7 固定小数点乗算

図 6.18 に乗算命令のフローを示します。表 6.27 にこの演算の種類を示します。表 6.28 に各オペランドとレジスタの対応を示します。DSP ユニットの乗算は、シングルワード符号付き単精度乗算です。

倍精度乗算が必要な場合は、CPU のダブルワード乗算命令を使用します

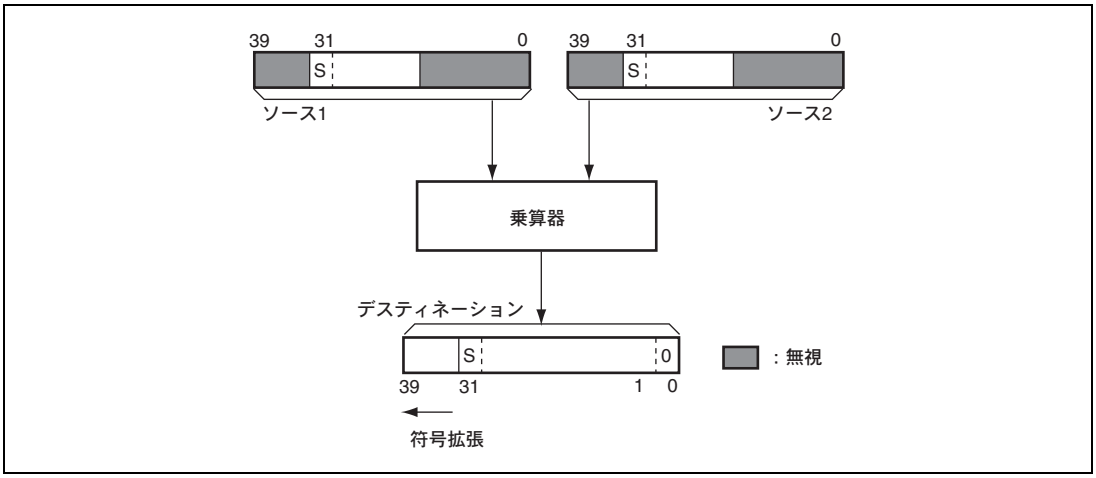


図 6.18 固定小数点乗算フロー

表 6.27 固定小数点乗算の種類

ニーモニック	機能	ソース 1	ソース 2	デスティネーション
PMULS	符号付き乗算	Se	Sf	Dg

表 6.28 固定小数点乗算のオペランドとレジスタの対応

レジスタ	Se	Sf	Dg
A0			Yes
A1	Yes	Yes	Yes
M0			Yes
M1			Yes
X0	Yes	Yes	
X1	Yes		
Y0	Yes	Yes	
Y1		Yes	

【注】 乗算は、基本的に 32 ビットの演算結果を生成します。したがって、ガードビット部分を提供するレジスタをデスティネーションオペランドに指定するとガードビット部分には、演算結果のビット 31 がコピーされます。

DSP ユニット側の乗算は、整数ではなく固定小数点演算です。したがって、乗数および被乗数それぞれの上位ワードが**図 6.18** に示すように乗算器に入力されます。固定小数点乗算結果は MSB に揃えられ、固定小数点乗算結果の LSB は常に 0 になります。

乗算演算は、常に無条件で実行されますが、DSR レジスタの DC、N、Z、V、GT、AN、AZ、AV、および AGT の条件コードビットは更新されません。

- **オーバフローの防止機能**

SR レジスタの S ビットは DSP ユニットのこの乗算に対して有効です。詳細については、**「6.5.12　オーバフロー防止機能」**を参照してください。

S ビットが 0 の場合は、H'8000*H'8000（（－1.0）*（－1.0））演算を符号付き固定小数点乗算として実行するときだけオーバフローが発生します。結果は、H'00 8000 0000 ですが、（＋1.0）を意味しません。S ビットが 1 の場合は、オーバフロー防止機能が働いて結果は H'00 7FFF FFFF となります。

6.5.8　シフト演算

シフト演算は、シフト量オペランドとしてレジスタ値またはイミディエイト値を使用することができます。他のソースオペランドとデスティネーションオペランドは、レジスタで指定します。シフト演算には、算術シフトおよび論理シフトの 2 種類があります。**表 6.29** にこの演算の種類を示します。イミディエイトオペランドを除き、各オペランドのレジスタとの対応は、**表 6.24** に示すように ALU 固定小数点算術演算と同じです。

表 6.29　シフト演算の種類

ニーモニック	機能	ソース 1	ソース 2	デスティネーション
PSHA Sx, Sy, Dz	算術シフト	Sx	Sy	Dz
PSHL Sx, Sy, Dz	論理シフト	Sx	Sy	Dz
PSHA #Imm1, Dz	イミディエイト付き算術シフト	Dz	Imm1	Dz
PSHL #Imm2, Dz	イミディエイト付き論理シフト	Dz	Imm2	Dz

-32<=Imm1<=+32, -16<=Imm2<=+16

（1）　算術シフト：

図 6.19 に算術シフト演算フローを示します。

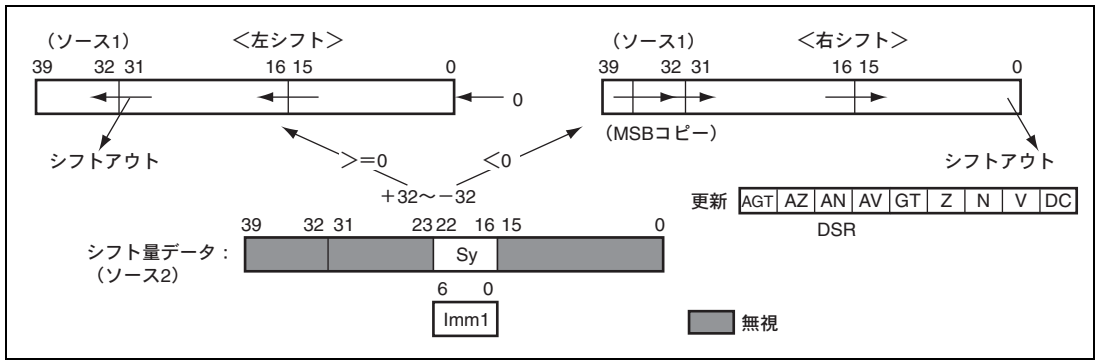


図 6.19　算術シフト演算フロー

算術シフト演算は、基本的に 40 ビット演算、すなわち 32 ビットの基本精度、8 ビットのガードビット部分から構成されます。したがって、ガードビット部分を提供していないレジスタをソースオペランドに指定すると、符号ビットがガードビット部分にコピーされます。ガードビット部分を提供していないレジスタをデスティネーションオペランドに指定すると、演算結果の下位 32 ビットがデスティネーションレジスタに入力されます。

この算術シフト演算においては、ソース 1 オペランドとデスティネーションオペランドは全ビット有効になります。シフト量は、整数部分としてソース 2 オペランドで指定します。ソース 2 オペランドは、レジスタまたはイミディエイトオペランドで指定することができます。利用可能なシフト範囲は、−32 から＋32 までです。ここで負の値は右シフト、正の値は左シフトを意味します。ソース 2 オペランドとしては、−64 から＋63 までを指定することができますが、無効なシフト値が指定された場合は、結果は保証されません。イミディエイトオペランド命令を持つシフトの場合は、ソース 1 オペランドはデスティネーションのレジスタと同じレジスタでなければなりません。

算術シフト演算を実行するたびに、DSR レジスタの DC、N、Z、V、GT、AN、AZ、AV、および AGT ビットは、基本的に演算結果に従って更新されます。また、DSR[11:8]≠0000 のとき、AGT、AZ、AN、AV ビットは演算結果に従って累積されます。ただし、条件付き命令の場合は、指定条件が真で演算が実行されてもこれらのビットは更新されません。無条件の命令の場合は、これらは演算結果に従って常に更新されます。DC ビットの定義は DSR レジスタの CS[2:0]ビット（条件選択ビット）で指定します。DC ビットの結果は、次のとおりです。

1. キャリー／ボローモード（CS[2:0]＝000の場合）

   DCビットは、演算結果として最後にシフトアウトしたデータを示します。

2. 負値モード（CS[2:0]＝001の場合）

   DCビットは、演算結果が負の値のとき1にセットされ、ゼロまたは正の値のときに0クリアされます。

3. ゼロ値モード（CS[2:0]＝010の場合）

   DCビットは、演算結果がゼロのとき1にセットされます。それ以外は、0クリアされます。

4. オーバフローモード（CS[2:0]＝011の場合）

   オーバフローが発生したときに1にセットされます。

5. 符号付き大モード（CS[2:0]＝100の場合）

   DCビットは、常に0にクリアされます。

6. 符号付き以上モード（CS[2:0]＝101の場合）

   DCビットは、常に0にクリアされます。

N ビットは、CS[2:0]ビットが負値モードとしてセットされる DC ビットと常に同じ状態を示します。上記の負値モード部分を参照してください。Z ビットは、CS[2:0]ビットがゼロ値モードとしてセットされる DC ビットと常に同じ状態を示します。上記のゼロ値モード部分を参照してください。V ビットは、CS[2:0]ビットがオーバフローモードとしてセットされる DC ビットと常に同じ状態を示します。上記のオーバフローモード部分を参照してください。GT ビットは、CS[2:0]ビットが符号付き大モードとしてセットされる DC ビットと常に同じ状態を示します。上記の符号付き大モード部分を参照してください。

- **オーバフローの防止機能**

SR レジスタの S ビットは、DSP ユニットのどの算術シフト演算に対しても有効です。詳細については、**「6.5.12 オーバフロー防止機能」**を参照してください。

(2) 論理シフト：

図 6.20 に論理シフト演算フローを示します。

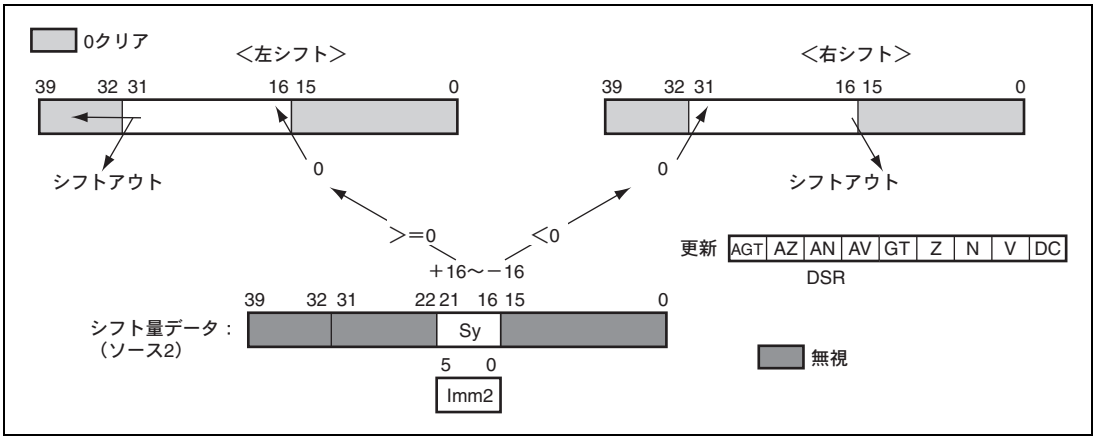


図 6.20 論理シフト演算フロー

図 6.20 に示すように、論理シフト演算は、ソース 1 の上位ワードとデスティネーションオペランドを使用します。ソースオペランドの下位ワードとガードビット部分は無視され、デスティネーションオペランドの下位ワードとガードビット部分は、ALU 論理演算同様、0 にクリアされます。シフト量は、整数データとしてソース 2 オペランドに指定します。ソース 2 オペランドでは、レジスタまたはイミディエイトオペランドに指定できます。利用可能なシフト範囲は、−16 から+16 です。ここで、負の値は右シフト、正の値は左シフトを意味します。任意のソース 2 オペランドは、−32 から+31 までを指定することができますが、無効なシフト値を指定すると、結果は保証されません。イミディエイトオペランド命令を持つシフトの場合は、ソース 1 オペランドはデスティネーションのレジスタと同じレジスタでなければなりません。

論理シフト演算を実行するたびに、DSR レジスタの DC、N、Z、V、GT、AN、AZ、AV、および AGT ビットは、基本的に演算結果に従って更新されます。また、DSR[11:8]≠0000 のとき、AGT、AZ、AN、AV ビットは演算結果に従って累積されます。条件付き演算の場合は、指定条件が真で演算が実行されてもこれらのビットは更新されません。無条件の演算の場合は、これらは演算結果で常に更新されます。DC ビットの定義は、DSR レジスタの CS[2:0]ビット（条件選択ビット）で指定します。DC ビットの結果は、次のとおりです。

1. キャリー／ボローモード（CS[2:0]＝000の場合）

   DCビットは、演算結果として最後にシフトアウトしたデータを示します。

2. 負値モード（CS[2:0]＝001の場合）

   DCビットは、演算結果のビット31の値が格納されます。

3. ゼロ値モード（CS[2:0]＝010の場合）

   DCビットは、演算結果がゼロのとき1にセットされます。それ以外は、0にクリアされます。

4. オーバフローモード（CS[2:0]＝011の場合）

   DCビットは、常に0にクリアされます。

5. 符号付き大モード（CS[2:0]＝100の場合）

   DCビットは、常に0にクリアされます。

6. 符号付き以上モード（CS[2:0]＝101の場合）

   DCビットは、常に0にクリアされます。

N ビットは、CS[2:0]ビットが負値モードとしてセットされる DC ビットと常に同じ状態を示します。上記の負値部分を参照してください。Z ビットは、CS[2:0]ビットがゼロ値モードとしてセットされる DC ビットと常に同じ状態を示します。上記のゼロ値モード部分を参照してください。V ビットは、CS[2:0]ビットがオーバフローモードとしてセットされる DC ビットと常に同じ状態を示しますが、この演算では常にクリアされます。GT ビットも同じです。

6.5.9 MSB 検出命令

MSB 検出命令（PDMSB：Detect Most Significant Bit）は、正規化のためのシフト量を計算するために使用されます。**図 6.21** に PDMSB 命令のフローを、**表 6.30** に演算の定義を示します。**表 6.31** にこの演算の種類を示します。各オペランドのレジスタとの対応は、**表 6.24** に示した ALU 固定小数点算術演算と同じです。

MSB 検出命令の結果は、ALU 整数演算と同様、基本的に 24 ビット、すなわち上位 16 ビットの基本精度と 8 ビットのガードビット部分です。ガードビット部分を提供していないレジスタをデスティネーションオペランドに指定すると、演算結果の上位ワードがデスティネーションレジスタに入力されます。

図 6.21 に示すように、PDMSB 命令はソースオペランドとしてフルサイズのデータを使用しますが、正規化用のシフト量データは**「6.5.8　シフト演算」**で述べたように整数データでなければならないので、デスティネーションオペランドは整数演算結果と見なされます。

PDMSB 演算を実行するたびに、DSR レジスタの DC、N、Z、V、GT、AN、AZ、AV、および AGT ビットは、基本的に演算結果に従って更新されます。また、DSR[11:8]≠0000 のとき、AGT、AZ、AN、AV ビットは演算結果に従って累積されます。条件付き命令の場合は、指定条件が真で演算が実行されてもこれらのビットは更新されません。無条件命令の場合は、これらは演算結果で常に更新されます。

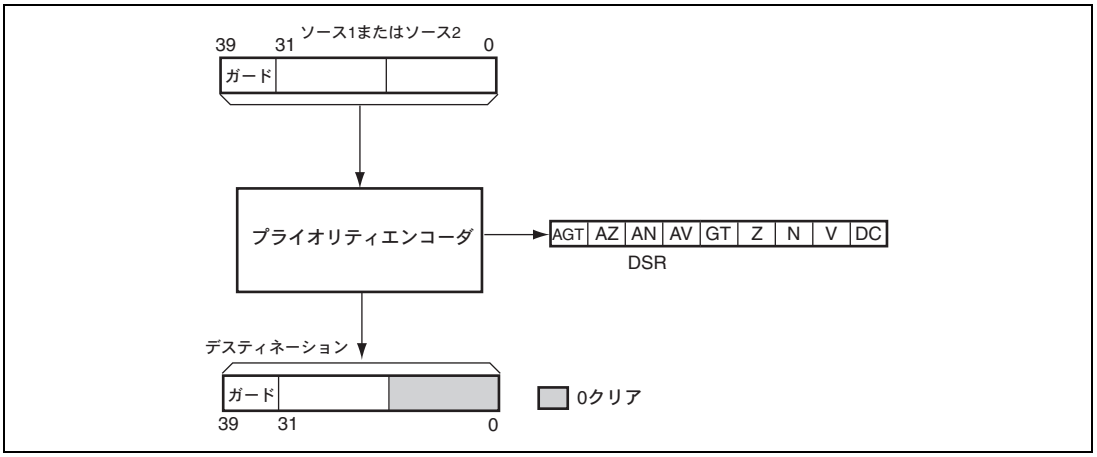


図 6.21　PDMSB 演算フロー

DC ビットの定義は DSR レジスタの CS[2:0] ビット（条件選択ビット）で選択します。DC ビットの結果は、次のとおりです。

1. キャリー／ボローモード（CS[2:0]＝000の場合）

   DCビットは、常に0にクリアされます。

2. 負値モード（CS[2:0]＝001の場合）

   DCビットは、演算結果が負の値のとき1にセットされ、ゼロまたは正の値のとき0にクリアされます。

3. ゼロ値モード（CS[2:0]＝010の場合）

   DCビットは、演算結果がゼロのとき1にセットされます。それ以外は、0にクリアされます。

4. オーバフローモード（CS[2:0]＝011の場合）

   DCビットは、常に0にクリアされます。

5. 符号付き大モード（CS[2:0]＝100の場合）

   DCビットは、演算結果が正の値のときに1にセットされます。それ以外は、0にクリアされます。

6. 符号付き以上モード（CS[2:0]＝101の場合）

   DCビットは、演算結果が正またはゼロのとき1にセットされます。それ以外は、0にクリアされます。

表 6.30　PDMSB 命令の定義

ソースデータ														デスティネーションの結果							
ガードビット					上位ワード					下位ワード				ガードビット	上位ワード						
39	38	−	33	32	31	30	29	28	−	3	2	1	0	39～32	31～22	21	20	19	18	17	16	10 進数
0	0	−	0	0	0	0	0	0	−	0	0	0	0	all 0	all 0	0	1	1	1	1	1	+31
0	0	−	0	0	0	0	0	0	−	0	0	0	1	all 0	all 0	0	1	1	1	1	0	+30
0	0	−	0	0	0	0	0	0	−	0	0	1	*	all 0	all 0	0	1	1	1	0	1	+29
0	0	−	0	0	0	0	0	0	−	0	1	*	*	all 0	all 0	0	1	1	1	0	0	+28
		:							:									:				
0	0	−	0	0	0	0	0	1	−	*	*	*	*	all 0	all 0	0	0	0	0	1	0	+2
0	0	−	0	0	0	0	1	*	−	*	*	*	*	all 0	all 0	0	0	0	0	0	1	+1
0	0	−	0	0	0	1	*	*	−	*	*	*	*	all 0	all 0	0	0	0	0	0	0	0
0	0	−	0	0	1	*	*	*	−	*	*	*	*	all 1	all 1	1	1	1	1	1	1	−1
0	0	−	0	1	*	*	*	*	−	*	*	*	*	all 1	all 1	1	1	1	1	1	0	−2
		:							:									:				
0	1	−	*	*	*	*	*	*	−	*	*	*	*	all 1	all 1	1	1	1	0	0	0	−8
1	0	−	*	*	*	*	*	*	−	*	*	*	*	all 1	all 1	1	1	1	0	0	0	−8
		:							:									:				
1	1	−	1	0	*	*	*	*	−	*	*	*	*	all 1	all 1	1	1	1	1	1	0	−2
1	1	−	1	1	0	*	*	*	−	*	*	*	*	all 1	all 1	1	1	1	1	1	1	−1
1	1	−	1	1	1	0	*	*	−	*	*	*	*	all 0	all 0	0	0	0	0	0	0	0
1	1	−	1	1	1	1	0	*	−	*	*	*	*	all 0	all 0	0	0	0	0	0	1	+1
1	1	−	1	1	1	1	1	0	−	*	*	*	*	all 0	all 0	0	0	0	0	1	0	+2
		:							:									:				
1	1	−	1	1	1	1	1	1	−	1	0	*	*	all 0	all 0	0	1	1	1	0	0	+28
1	1	−	1	1	1	1	1	1	−	1	1	0	*	all 0	all 0	0	1	1	1	0	1	+29
1	1	−	1	1	1	1	1	1	−	1	1	1	0	all 0	all 0	0	1	1	1	1	0	+30
1	1	−	1	1	1	1	1	1	−	1	1	1	1	all 0	all 0	0	1	1	1	1	1	+31

【注】　*　Don't care ビットを意味します。

表 6.31　PDMSB 命令の種類

ニーモニック	機能	ソース 1	ソース 2	デスティネーション
PDMSB	MSB 検出	Sx	−	Dz
		−	Sy	Dz

N ビットは、CS[2:0]ビットが負値モードとしてセットされる DC ビットと常に同じ状態を示します。上記の負値モード部分を参照してください。Z ビットは、CS[2:0]ビットがゼロ値モードとしてセットされる DC ビットと常に同じ状態を示します。上記のゼロ値モード部分を参照してください。V ビットは、CS[2:0]ビットがオーバフローモードとしてセットされる DC ビットと常に同じ状態を示しますが、本命令では、常に 0 クリアされます。上記のオーバフローモード部分を参照してください。GT ビットは、CS[2:0]ビットが符号付き大モードとしてセットされる DC ビットと常に同じ状態を示します。上記の符号付き大モード部分を参照してください。

6.5.10 丸め演算

DSP ユニットは、32 ビットから 16 ビットに丸める丸め機能を提供します。ガードビットがある場合は、40 ビットから 24 ビットに丸めます。丸め命令を実行するときは、ソースオペランドに H'00008000 が加算された後、下位ワードが 0 クリアされます。**図** 6.22 に丸め演算フローを示します。**図** 6.23 に丸め演算の定義を示します。また、**表** 6.32 に演算の種類を示します。各オペランドのレジスタとの対応は、**表** 6.24 に示した ALU 固定小数点算術演算と同じです。

図 6.23 に示すように、丸め演算は、ソースオペランド、デスティネーションオペランド両方に対してフルサイズデータを使用します。

丸め演算を実行するたびに、DSR レジスタの DC、N、Z、V、GT、AN、AZ、AV、および AGT ビットは、基本的に演算結果に従って更新されます。また、DSR[11:8]≠0000 のとき、AGT、AZ、AN、AV ビットは演算結果に従って累積されます。条件付き命令の場合は、指定条件が真で演算が実行されてもこれらのビットは更新されません。無条件命令の場合は、これらは演算結果で常に更新されます。DC ビットの定義は、DSR レジスタの CS[2:0]（条件選択）ビットで指定します。これらの状態コードビットの結果は ALU 固定小数点算術演算と同じです。

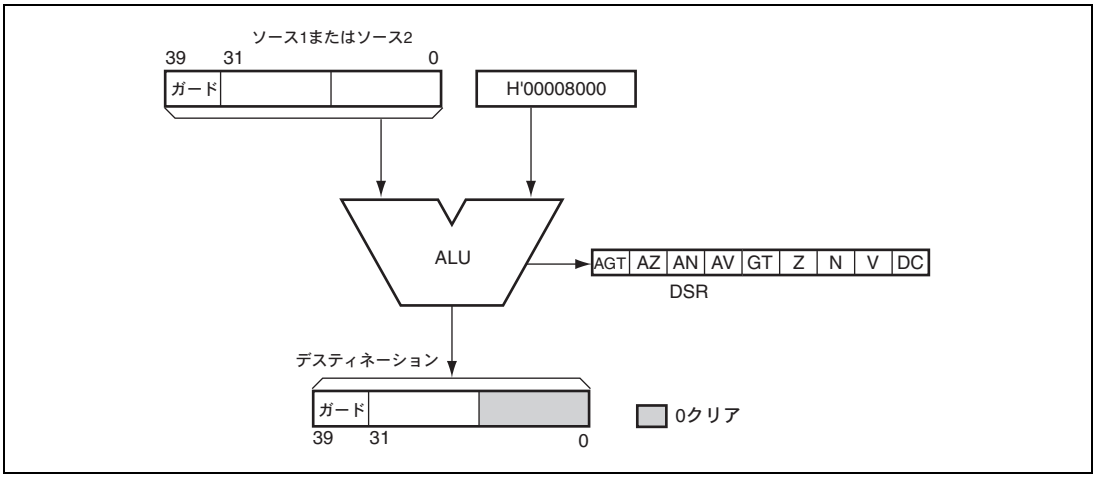


図 6.22 丸め演算フロー

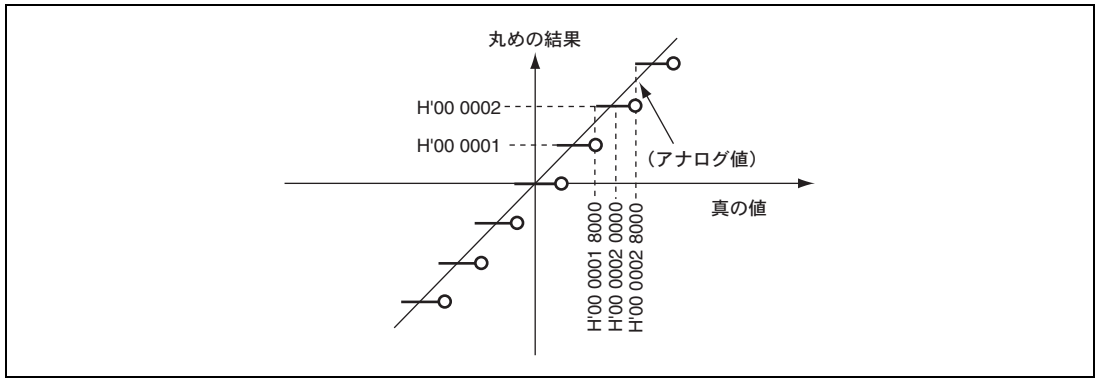


図 6.23　丸め演算の定義

表 6.32　丸め演算の種類

ニーモニック	機能	ソース 1	ソース 2	デスティネーション
PRND	丸め	Sx	−	Dz
		−	Sy	Dz

- **オーバフロー防止機能**

SR レジスタの S ビットは DSP ユニットの任意の丸め演算で有効です。詳細については、**「6.5.12　オーバフロー防止機能」**を参照してください。

6.5.11　スワップ命令

スワップ命令（PSWAP）は、上位ワードと下位ワードを交換するために使用されます。ソースオペランドにガードビットがある場合は無視されます。デスティネーションオペランドにガードビットがあるときは、符号拡張されます。**図 6.24** に PSWAP 命令の演算のフローを示します。**表 6.33** にこの演算の種類を示します。各オペランドのレジスタとの対応は、**表 6.24** に示した ALU 固定小数点演算と同じです。

PSWAP 演算を実行するたびに、DSR レジスタの DC、N、Z、V、GT ビットは、基本的に演算結果に従って更新されます。また、DSR[11:8]≠0000 のとき、AGT、AZ、AN、AV ビットは演算結果に従って更新されます。

DC ビットの定義は、DSR レジスタの CS[2:0]ビット（条件選択ビット）で選択します。DC ビットの結果は次のとおりです。

1. キャリー／ボローモード（CS[2:0]＝000の場合）

   DCビットは、常に0にクリアされます。

2. 負値モード（CS[2:0]＝001の場合）

   DCビットは、演算結果が負の値のとき1にセットされ、ゼロまたは正の値のとき0にクリアされます。

3. ゼロ値モード（CS[2:0]＝010の場合）

   DCビットは、演算結果がゼロのとき1にセットされます。それ以外は、0にクリアされます。

4. オーバフローモード（CS[2:0]＝011の場合）

   DCビットは、常に0にクリアされます。

5. 符号付き大モード（CS[2:0]＝100の場合）

   DCビットは、演算結果が正の値のときに1にセットされます。それ以外は、0にクリアされます。

6. 符号付き以上モード（CS[2:0]＝101の場合）

   DCビットは、演算結果が正またはゼロのとき1にセットされます。それ以外は、0にクリアされます。

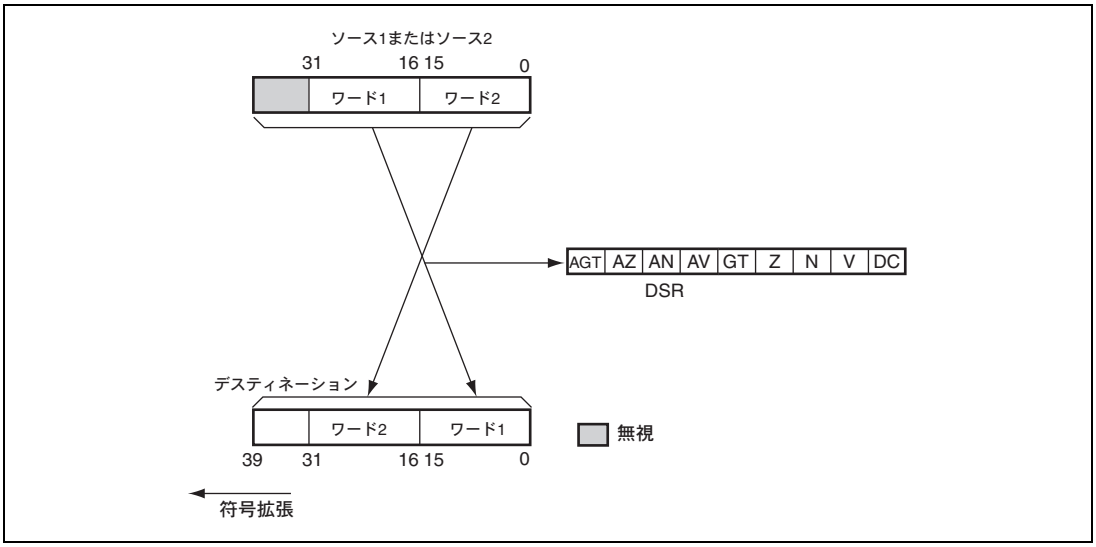


図 6.24　PSWAP 演算フロー

表 6.33　PSWAP 命令の種類

ニーモニック	機能	ソース 1	ソース 2	デスティネーション
PSWAP	上位ワード、下位ワード交換	Sx	−	Dz
		−	Sy	Dz

- **オーバフロー防止機能**

SR レジスタの S ビットは、DSP ユニットの任意の PSWAP 演算で有効です。詳細については、**「6.5.12　オーバフロー防止機能」**を参照してください。

6.5.12 オーバフロー防止機能

SR レジスタの S ビットは、従来の SH 乗算および MAC 演算を含め DSP ユニットで実行する算術演算に有効です。SH の CPU コアの SR レジスタの S ビットは、オーバフロー防止機能イネーブルビットとして使用します。演算結果がガードビット部分のない 2 の補数の表記範囲を越える場合は、算術演算はオーバフローします。**表 6.34** に**「6.5.7 固定小数点乗算」**で説明した符号付き×符号付き固定小数点乗算を含めて、固定小数点算術演算に対するオーバフロー保護の定義を示します。**表 6.35** には整数算術演算に対するオーバフロー防止機能の定義を示します。整数算術演算の飽和値の下位ワードは Don't care です。下位ワードの値は、保証されません。

オーバフロー防止機能が有効になっているときは、オーバフローは発生することはありません。したがって、V ビットは、0 にクリアされます。CS[2:0]ビットでオーバフローモードを選択するときも DC ビットは 0 にクリアされます。ただし、TS[2:0]ビットでオーバフローモードを選択する場合は、S ビットが 1 のときもオーバフロー検出を行います。

表 6.34 固定小数点算術用演算のオーバフロー防止機能の定義

符号	オーバフロー条件	固定値	16 進表記
正	結果＞$1-2^{-31}$	$1-2^{-31}$	00 7FFF FFFF
負	結果＜−1	−1	FF 8000 0000

表 6.35 整数算術演算用オーバフロー防止機能の定義

符号	オーバフロー条件	固定値	16 進表記
正	結果＞$2^{15}-1$	$2^{15}-1$	00 7FFF ****
負	結果＜-2^{15}	-2^{15}	FF 8000 ****

【注】* Don't care を意味します。

6.5.13 ローカルデータ移動命令

SH4AL-DSP には、CPU の乗算／積和演算（MAC）をサポートするための MACL と MACH の 2 つレジスタがあります。これらのレジスタは、他の DSP レジスタとのローカルデータ移動命令により、テンポラリレジスタとして活用することができます。**図 6.25** にローカルデータ移動命令のフローを示します。**表 6.36** にはこの命令の種類を示します。

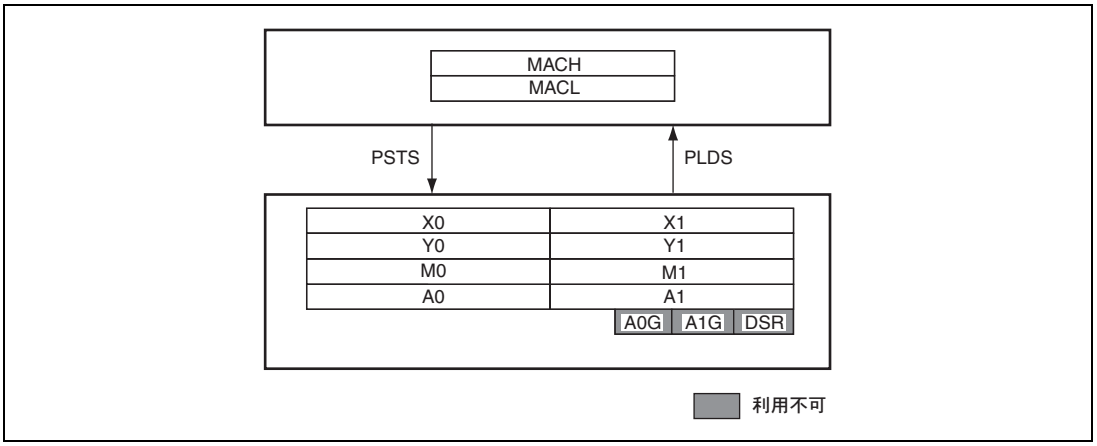


図 6.25 ローカルデータ移動命令のフロー

表 6.36 ローカルデータ移動命令の種類

ニーモニック	機能	オペランド
PLDS	DSP レジスタから MACL/H へのデータ移動	Dz
PSTS	MACL/H から DSP レジスタへのデータ移動	Dz

この命令は、他の転送命令と非常に似ています。A0 および A1 レジスタのいずれかを PSTS のデスティネーションオペランドとして指定すると、符号ビットが該当するガードビット部分 A0G または A1G に符号拡張されて格納されます。命令結果にかかわらず、DSR レジスタの DC、N、Z、V、GT の条件コードビットは、更新されません。また、AGT、AZ、AN、AV ビットも更新されません。この命令は、条件付きとしても動作します。ローカルデータ移動命令は、MOVX と MOVY で並行して指定することができます。

6.5.14 並行処理命令の命令フォーマット

並行処理命令は DSP ユニットを使ったデジタル信号処理を効率よく実行するための命令です。32 ビット長で、同時に並行して 4 つの処理、ALU 演算、乗算、2 つのデータ転送ができます。

並行処理命令は A フィールドと B フィールドに分かれています。A フィールドはデータ転送命令を定義し、B フィールドは ALU 演算命令、乗算命令を定義します。これらの命令は独立に定義することができ、処理は独立に、しかも同時に並行して実行されます。A フィールドの並行データ転送命令を**表 6.37** および**表 6.38** に、B フィールドの ALU 演算命令、乗算命令を**表 6.39** に示します。A フィールドの命令は、**表 6.13**、**表 6.14** のダブルデータ転と同じです。

表 6.38 は X メモリか Y メモリの一方の転送命令が NOPX または NOPY のときの拡張命令です。一方が NOPX または NOPY 以外のときは使用できません。

表 6.37 A フィールドの並行データ転送命令（1）

分類	ニーモニック	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15~0
X メモリデータ転送	NOPX	1	1	1	1	1	0	0		0		0		0	0			B フィールド
	MOVX.W @Ax,Dx							Ax		Dx		0		0	1			
	MOVX.W @Ax+,Dx													1	0			
	MOVX.W @Ax+Ix,Dx													1	1			
	MOVX.W Da,@Ax									Da		1		0	1			
	MOVX.W Da,@Ax+													1	0			
	MOVX.W Da,@Ax+Ix													1	1			
Y メモリデータ転送	NOPY	1	1	1	1	1	0		0		0		0			0	0	B フィールド
	MOVY.W @Ay,Dy								Ay		Dy		0			0	1	
	MOVY.W @Ay+,Dy															1	0	
	MOVY.W @Ay+Iy,Dy															1	1	
	MOVY.W Da,@Ay										Da		1			0	1	
	MOVY.W Da,@Ay+															1	0	
	MOVY.W Da,@Ay+Iy															1	1	

【注】 Ax：0＝R4、1＝R5

Ay：0＝R6、1＝R7

Dx：0＝X0、1＝X1

Dy：0＝Y0、1＝Y1

Da：0＝A0、1＝A1

表 6.38　A フィールドの並行データ転送命令（2）

分類	ニーモニック	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15～0
X メモリデータ転送	MOVX.W @Axy,Dxy	1	1	1	1	1	0	Axy		Dxy		0	0	0	1	0	0	B フィールド
	MOVX.W @Axy+,Dxy													1	0			
	MOVX.W @Axy+Ix,Dxy													1	1			
	MOVX.W Dax,@Axy									Dax		1	0	0	1			
	MOVX.W Dax,@Axy+													1	0			
	MOVX.W Dax,@Axy+Ix													1	1			
	MOVX.L @Axy,Dxy									Dxy		0	1	0	1			
	MOVX.L @Axy+,Dxy													1	0			
	MOVX.L @Axy+Ix,Dxy													1	1			
	MOVX.L Dax,@Axy									Dax		1	1	0	1			
	MOVX.L Dax,@Axy+													1	0			
	MOVX.L Dax,@Axy+Ix													1	1			
Y メモリデータ転送	MOVY.W @Ayx,Dyx	1	1	1	1	1	0	Ayx		Dyx		0	0	0	0	0	1	B フィールド
	MOVY.W @Ayx+,Dyx															1	0	
	MOVY.W @Ayx+Iy,Dyx															1	1	
	MOVY.W Day,@Ayx									Day		0	1			0	1	
	MOVY.W Day,@Ayx+															1	0	
	MOVY.W Day,@Ayx+Iy															1	1	
	MOVY.L @Ayx,Dyx									Dyx		1	0			0	1	
	MOVY.L @Ayx+,Dyx															1	0	
	MOVY.L @Ayx+Iy,Dyx															1	1	
	MOVY.L Day,@Ayx									Day		1	1			0	1	
	MOVY.L Day,@Ayx+															1	0	
	MOVY.L Day,@Ayx+Iy															1	1	

【注】　Axy：R0、R1、R4、R5＝（01、11、00、10）

Ix＝R8

Dxy：X0、X1、Y0、Y1＝（00、10、01、11）

Dax：A0、A1、X0、X1＝（00、10、01、11）

Ayx：R2、R3、R6、R7＝（10、11、00、01）

Iy＝R9

Dyx：Y0、Y1、X0、X1＝（00、01、10、11）

Day：A0、A1、Y0、Y1＝（00、01、10、11）

表 6.39　B フィールドの ALU 演算命令、乗算命令

分類	ニーモニック	31	30	29	28	27	26	25～16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Imm シフト命令	PSHL #imm,Dz	1	1	1	1	1	0	A フィールド	0	0	0	0	0	-16 <= Imm <= + 16							Dz			
	PSHA #imm,Dz								0	0	0	1	0	-32 <= Imm <= + 32										
	リザーブ								0	0	0		1											
									0	0	1													
並行命令	PMULS Se,Sf,Dg								0	1	0	0	Se 0：X0 1：X1 2：Y0 3：A1		Sf 0：Y0 1：Y1 2：X0 3：A1		0	0	0	0	Dg 0：M0 1：M1 2：A0 3：A1		0	0
	PCLR Du PMULS Se,Sf,Dg								0	1	0	0					0	0	0	1			Du 0：X0 1：Y0 2：A0 3：A1	
	リザーブ								0	1	0	1					Sx 0：X0 1：X1 2：A0 3：A1		Sy 0：Y0 1：Y1 2：M0 3：M1					
	PSUB Sx,Sy,Du PMULS Se,Sf,Dg								0	1	1	0												
	PADD Sx,Sy,Du PMULS Se,Sf,Dg								0	1	1	1												
無条件命令	リザーブ								1	0	0	0	0	0	0	0	Sx		Sy		Dz			
									1	0	0	1	0	0	0	0								
	PSUBC Sx,Sy,Dz								1	0	1	0	0	0	0	0								
	PADDC Sx,Sy,Dz								1	0	1	1	0	0	0	0								
	PCMP Sx,Sy								1	0	0	0	0	1	0	0					0	0	0	0
	リザーブ								1	0	0	1	0	1	0	0					Dz 0：リザーブ 1：リザーブ 2：リザーブ 3：リザーブ 4：リザーブ 5：A1 6：リザーブ 7：A0 8：X0 9：X1 A：Y0 B：Y1 C：M0 D：リザーブ E：M1 F：リザーブ			
									1	0	1	0	0	1	0	0								
									1	0	1	1	0	1	0	0								
	PABS Sx,Dz								1	0	0	0	1	0	0	0			0	0				
	PRND Sx,Dz								1	0	0	1	1	0	0	0			0	0				
	PABS Sy,Dz								1	0	1	0	1	0	0	0	0	0	Sy					
	PRND Sy,Dz								1	0	1	1	1	0	0	0	0	0						
	リザーブ								1	0	0	0	1	1	0	0								
									1	0	0	1	1	1	0	0								
									1	0	1	0	1	1	0	0								
									1	0	1	1	1	1	0	0								

分類	ニーモニック	31	30	29	28	27	26	25～16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
条件付命令	[if cc] PSHL Sx,Sy,Dz	1	1	1	1	1	0	A フィールド	1	0	0	0	0	0	Ifcc 01:無条件 10:DCT 11:DCF		Sx		Sy		Dz 0：リザーブ 1：リザーブ 2：リザーブ 3：リザーブ 4：リザーブ 5：A1 6：リザーブ 7：A0 8：X0 9：X1 A：Y0 B：Y1 C：M0 D：リザーブ E：M1 F：リザーブ			
	[if cc] PSHA Sx,Sy,Dz								1	0	0	1	0	0										
	[if cc] PSUB Sx,Sy,Dz								1	0	1	0	0	0										
	[if cc] PADD Sx,Sy,Dz								1	0	1	1	0	0										
	[if cc] PSUB Sy,Sx,Dz								1	0	0	0	0	1										
	[if cc] PAND Sx,Sy,Dz								1	0	0	1	0	1										
	[if cc] PXOR Sx,Sy,Dz								1	0	1	0	0	1										
	[if cc] POR Sx,Sy,Dz								1	0	1	1	0	1										
	[if cc] PDEC Sx,Dz								1	0	0	0	1	0					0	0				
	[if cc] PINC Sx,Dz								1	0	0	1	1	0					0	0				
	[if cc] PDEC Sy,Dz								1	0	1	0	1	0			0	0	Sy					
	[if cc] PINC Sy,Dz								1	0	1	1	1	0			0	0						
	[if cc] PABS Sx,Dz								1	0	0	0	1	0	Ifcc 00:無条件 10:DCT 11:DCF		Sx		0	1				
	[if cc] PRND Sx,Dz								1	0	0	1	1	0					0	1				
	[if cc] PABS Sy,Dz								1	0	1	0	1	0			0	1	Sy					
	[if cc] PRND Sy,Dz								1	0	1	1	1	0			0	1						
	[if cc] PCLR Dz								1	0	0	0	1	1	Ifcc 01:無条件 10:DCT 11:DCF		0	0	0	0				
	[if cc] PDMSB Sx,Dz								1	0	0	1	1	1			Sx		0	0				
	[if cc] PSWAP Sx,Dz								1	0	0	1	1	1					0	1				
	リザーブ								1	0	1	0	1	1					Sy					
	[if cc] PDMSB Sy,Dz								1	0	1	1	1	1			0	0						
	[if cc] PSWAP Sy,Dz								1	0	1	1	1	1			0	1						
	リザーブ								1	1					0	0	Sx							
	リザーブ								1	1	0	0	1	0	Ifcc 01:無条件 10:DCT 11:DCF									
									1	1	0	1	1	0										
									1	1	1	0	1	0										
									1	1	1	1	1	0										
									1	1	0	0	1	1										
									1	1	0	1	1	1										
									1	1	1	0	1	1										
									1	1	1	1	1	1										
	[if cc] PNEG Sx,Dz								1	1	0	0	1	0					0	0				
	[if cc] PCOPY Sx,Dz								1	1	0	1	1	0					0	0				
	[if cc] PNEG Sy,Dz								1	1	1	0	1	0			0	0	Sy					
	[if cc] PCOPY Sy,Dz								1	1	1	1	1	0			0	0						

分類	ニーモニック	31	30	29	28	27	26	25～16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
条件付命令	[if cc] PSTS MACH,Dz	1	1	1	1	1	0	A フィールド	1	1	0	0	1	1	Ifcc 01：無条件 10：DCT 11：DCF		0	0	0	0	Dz 0：リザーブ 1：リザーブ 2：リザーブ 3：リザーブ 4：リザーブ 5：A1 6：リザーブ 7：A0 8：X0 9：X1 A：Y0 B：Y1 C：M0 D：リザーブ E：M1 F：リザーブ			
	[if cc] PSTS MACL,Dz								1	1	0	1	1	1			0	0	0	0				
	[if cc] PLDS Dz,MACH								1	1	1	0	1	1			0	0	0	0				
	[if cc] PLDS Dz,MACL								1	1	1	1	1	1			0	0	0	0				

6.5.15 オペランドの競合

同一のデスティネーションオペランドを複数の DSP データ演算命令による並行処理で指定すると、データの競合が発生します。**表 6.40** に各命令のオペランドとレジスタの対応を示します。

表 6.40 競合の発生するオペランドとレジスタとの対応

		DSPレジスタ							
		X0	X1	Y0	Y1	M0	M1	A0	A1
Xメモリロード	Ax								
	Ix								
	Dx	(*)	(*)						
	Dxy	(*)	(*)	(*)	(*)				
Yメモリロード	Ay								
	Iy								
	Dy			(*)	(*)				
	Dyx	(*)	(*)	(*)	(*)				
6オペランドALU演算	Sx	*	*					*	*
	Sy			*	*	*	*		
	Du	(*)		(*)				(*)	(*)
3オペランド乗算	Se	*	*	*					*
	Sf	*		*	*				*
	Dg					*	*	(*)	(*)
3オペランドALU演算	Sx	*	*					*	*
	Sy			*	*	*	*		
	Dz	(*)	(*)	(*)	(*)	*	*	*	*

（Dx、Dxy、DyxとDuとDzの競合）（Dy、Dxy、DyxとDuとDzの競合）　（DuとDgの競合）

【注】 * オペランドに対する設定可能レジスタ

(*) オペランド競合

オペランド競合には、次の 3 つの場合があります。

- **ALU演算と乗算命令が同じデスティネーションオペランド（DuおよびDg）を指定する場合**
- **X側ロードとALU命令が同じデスティネーションオペランド（Dx、DxyおよびDuまたはDz）を指定する場合**
- **Y側ロードとALU命令が同じデスティネーションオペランド（Dy、DyxおよびDuまたはDz）を指定する場合**

競合した場合の結果は、保証されません。

6.5.16 プログラミング上の注意

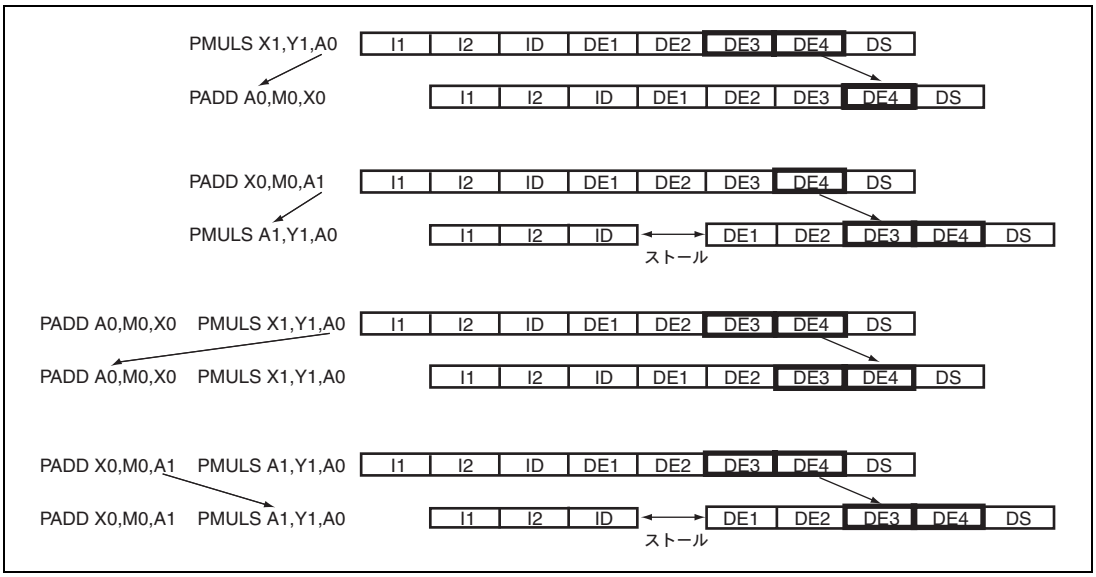


図 6.26 PADD 命令と PMULS 命令の実行例

PMULS 命令は、演算の実行に DE3、DE4 ステージを使用し、PMULS 命令以外の DSP データ演算命令は演算の実行に DE4 ステージを使用します。どちらも実行ステートは 1 サイクルです。

このため、先行の DSP 演算命令のデスティネーションレジスタを PMULS 命令のソースレジスタとして指定すると、1 サイクルのストールが発生します。

図 6.26 に PADD 命令と PMULS 命令の実行例を示します。

先行の PADD 命令のデスティネーションレジスタ A1 と同じレジスタを PMULS 命令のソースレジスタとして指定すると、1 サイクルのストールが発生します。

並行処理命令である、PADD PMULS、PSUB PMULS でも同様です。

7. メモリマネジメントユニット（MMU）

SH4AL-DSP 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、SH4AL-DSP 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。

SH4AL-DSP 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、SH4AL-DSP 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。

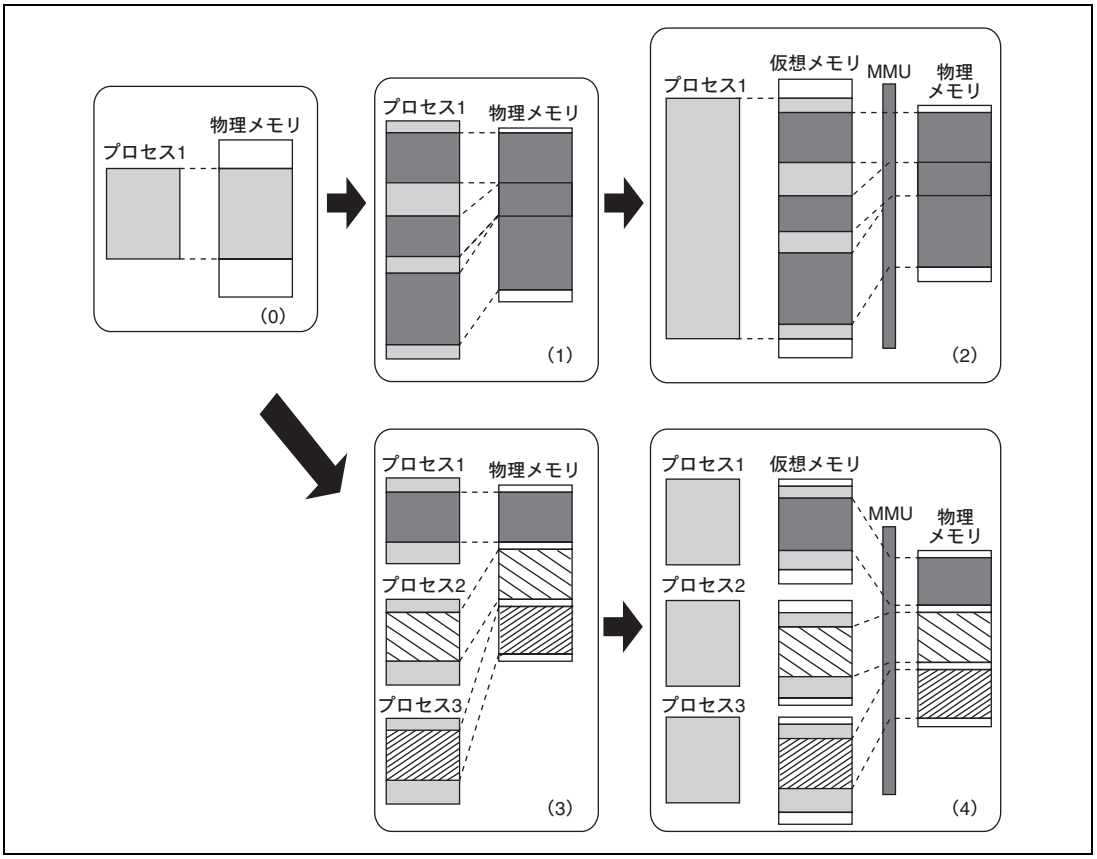


図 7.1　MMU の役割

7.1.1 アドレス空間

（1） 仮想アドレス空間

SH4AL-DSP 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 SR レジスタの DSP ビットが 1 か、あるいは内蔵メモリ制御レジスタ（RAMCR）の RMD ビットが 1 の場合、Uxy 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、Uxy 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。

MMU 制御レジスタ（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。

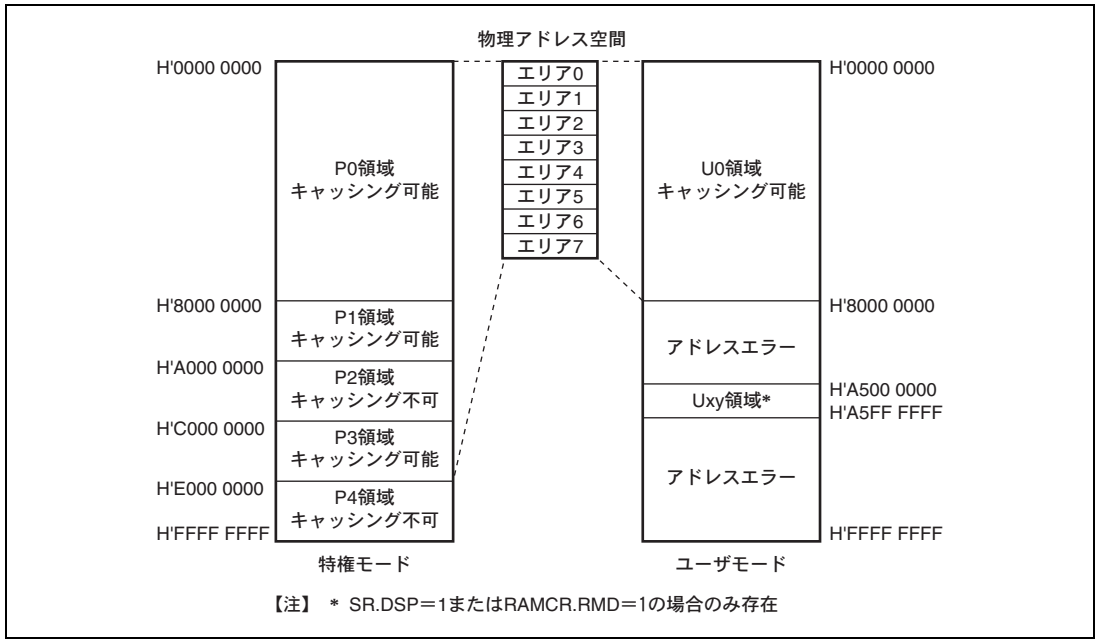


図 7.2 仮想アドレス空間（MMUCR.AT＝0）

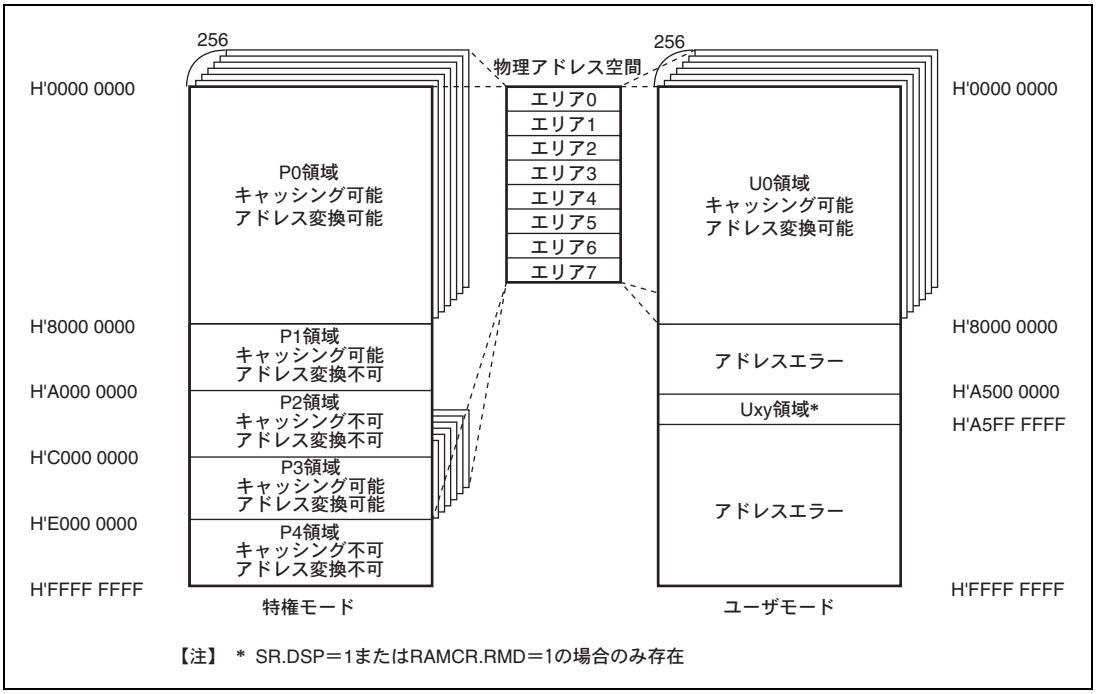


図 7.3　仮想アドレス空間（MMUCR.AT＝1）

(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 1 およびエリア 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 SH4AL-DSP 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。

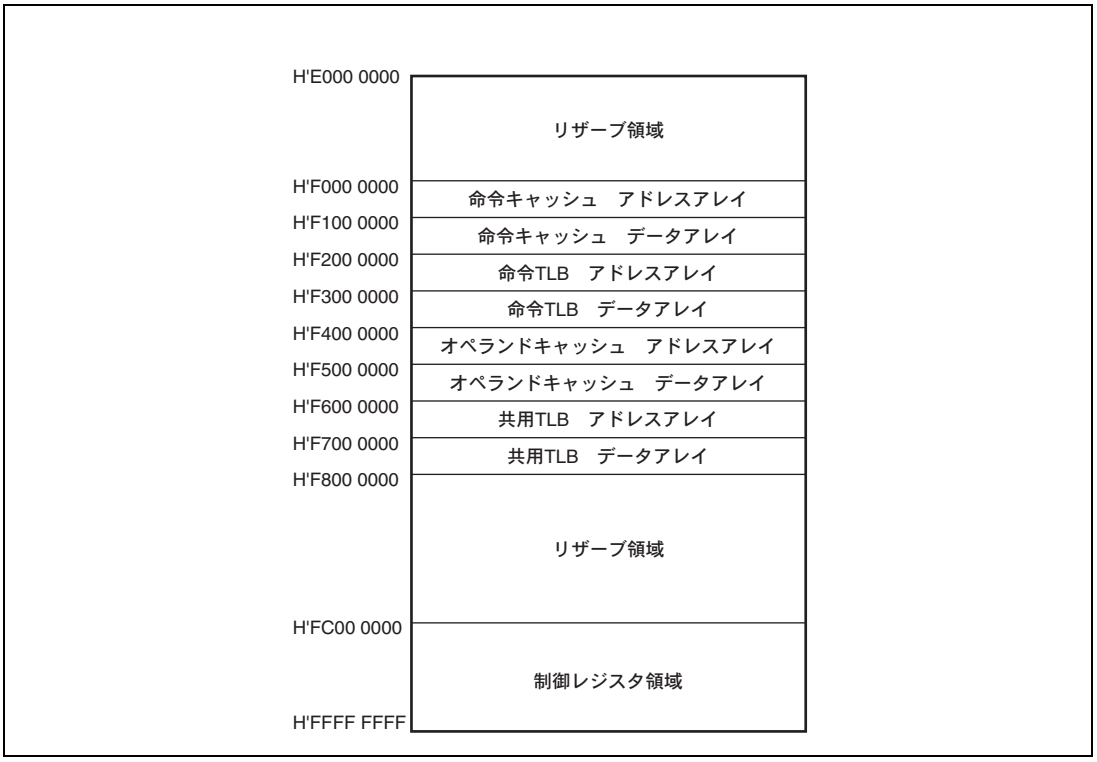


図 7.4　P4 領域

H'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'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'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。詳細は当該製品ハードウェ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。

(e)　Uxy 領域

Uxy 領域はユーザモードにおいて、SR レジスタの DSP ビットが 1 か、あるいは RAMCR レジスタの RMD ビットが 1 のときに使用可能となる、SH4AL-DSP の内蔵メモリにマッピングされる領域です。ユーザモードにおいて、SR.DSP ビットが 0 かつ RAMCR.RMD ビットが 0 のときにこの領域にアクセスすると、アドレスエラーとなります。この領域は 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は行えません。Uxy 領域の詳細に関しては「第 9 章　X/Y メモリ」をおよび「第 10 章　U メモリ」を参照してください。

(2) 物理アドレス空間

SH4AL-DSP 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ェアマニュアルの「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。SH4AL-DSP 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 0082	H'0000 0082	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.R2＝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 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、このレジスタへは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）

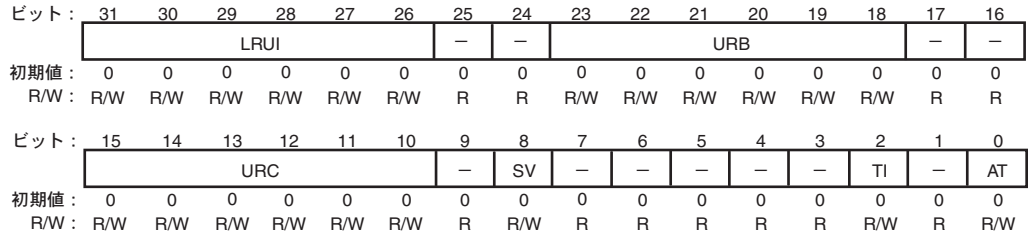
MMUCR の各ビットは以下に示すように MMU の設定を行います。このため MMUCR の書き換えは P1、P2 領域のプログラムで行うようにしてください。

MMUCR レジスタを更新後、P0、P3、U0 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	－	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	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	1	0	0	0	0	0	1	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	H'82	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。

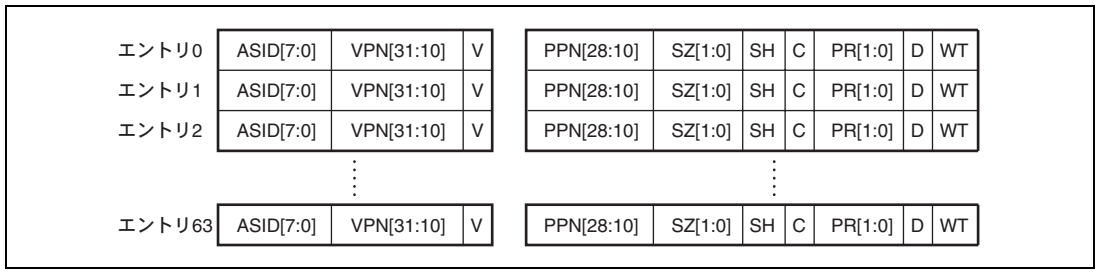


図 7.6　UTLB の構成

【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

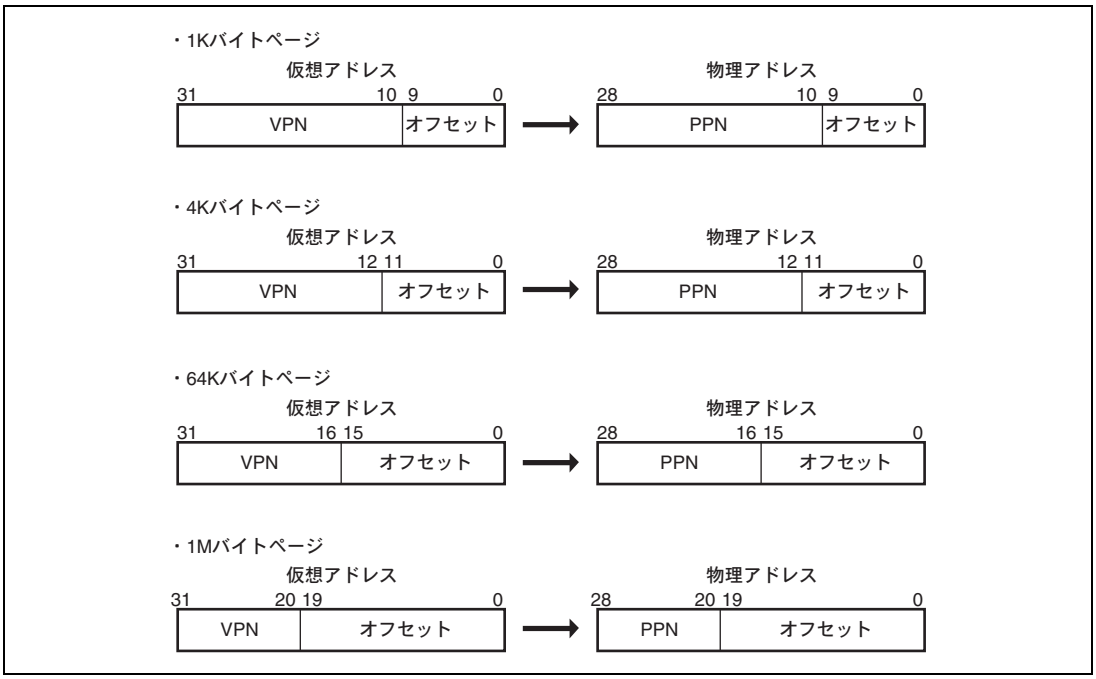


図 7.7　ページサイズとアドレスの関係

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。

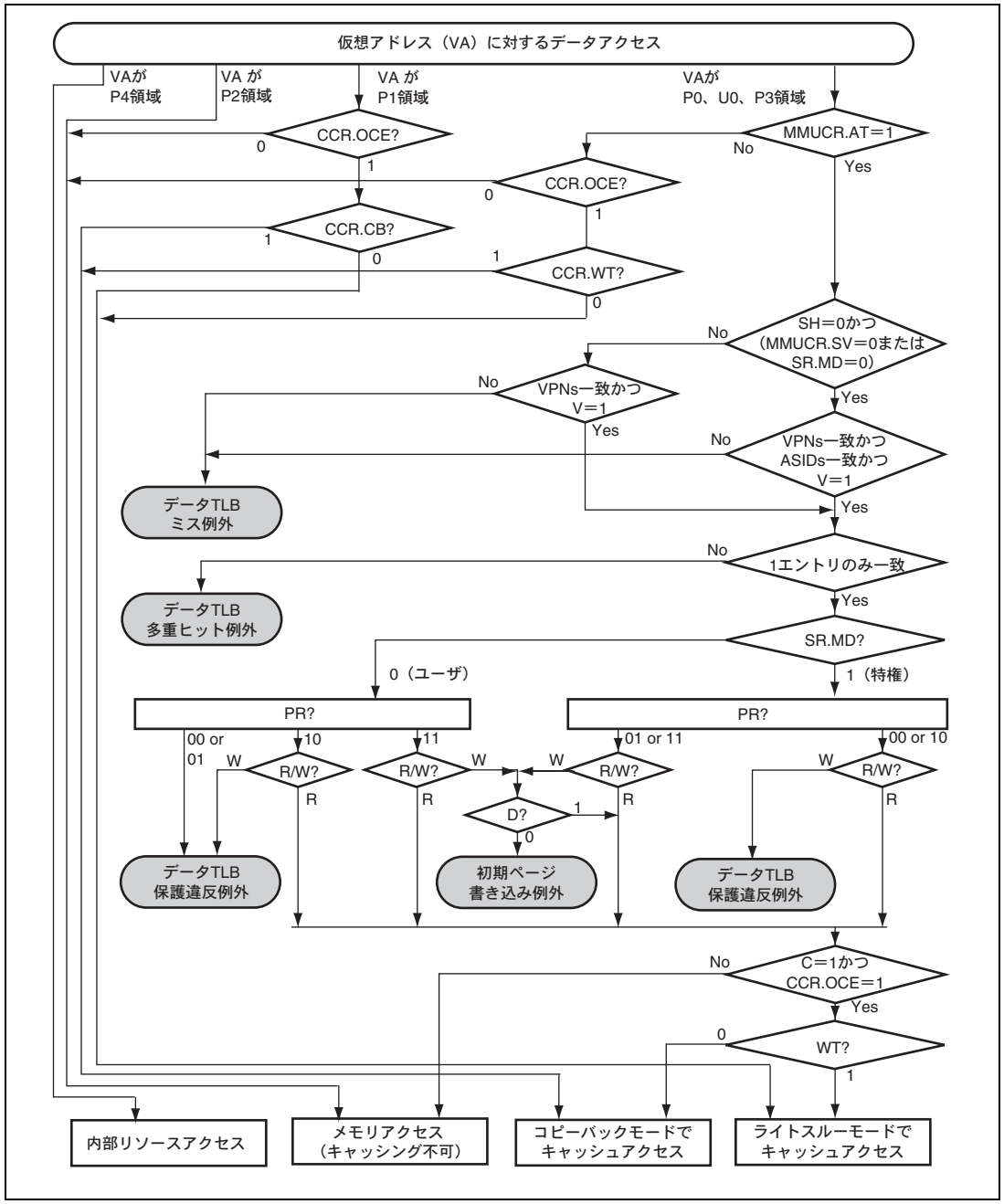


図 7.9　UTLB を用いたメモリアクセスフロー

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。

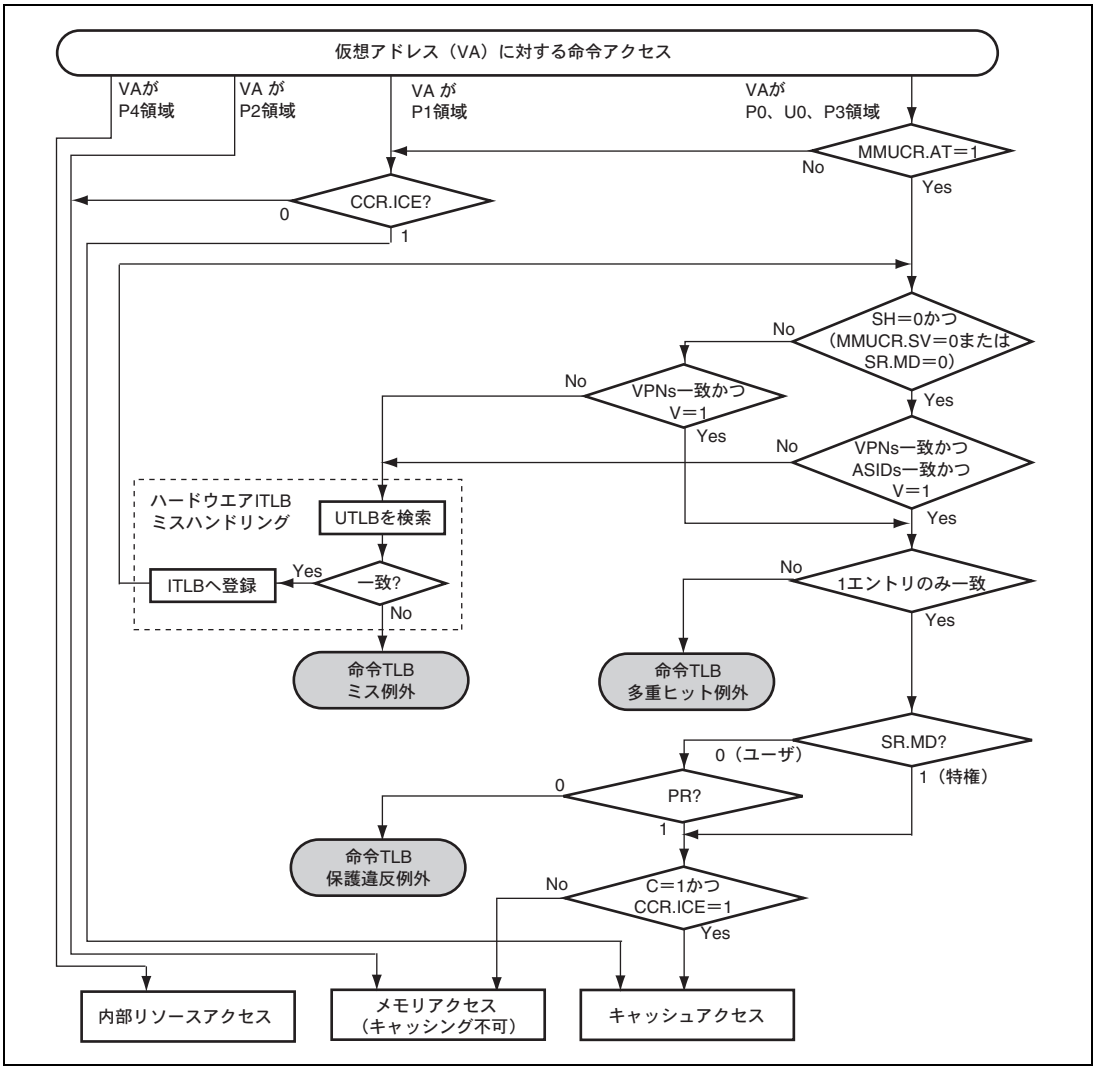


図 7.10　ITLB を用いたメモリアクセスフロー

7.4 MMU の機能

7.4.1 MMU のハードウェア管理

SH4AL-DSP 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、SH4AL-DSP 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。

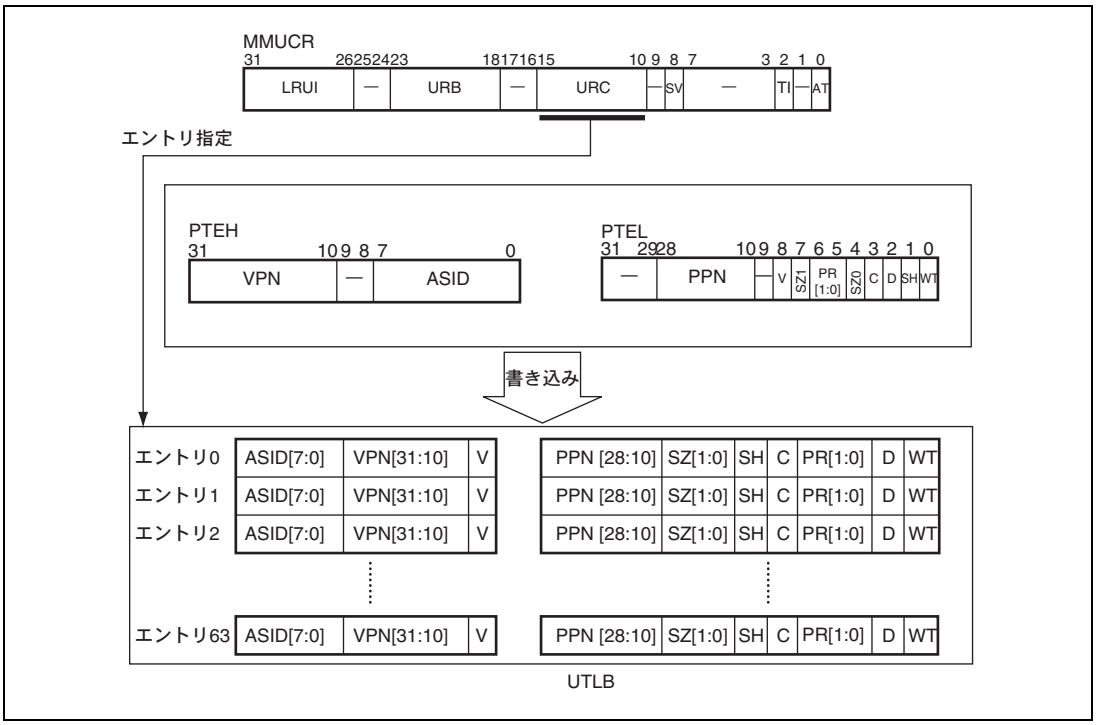


図 7.11　LDTLB 命令の動作

7.4.4 ハードウェア ITLB ミスハンドリング

SH4AL-DSP 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。SH4AL-DSP 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。

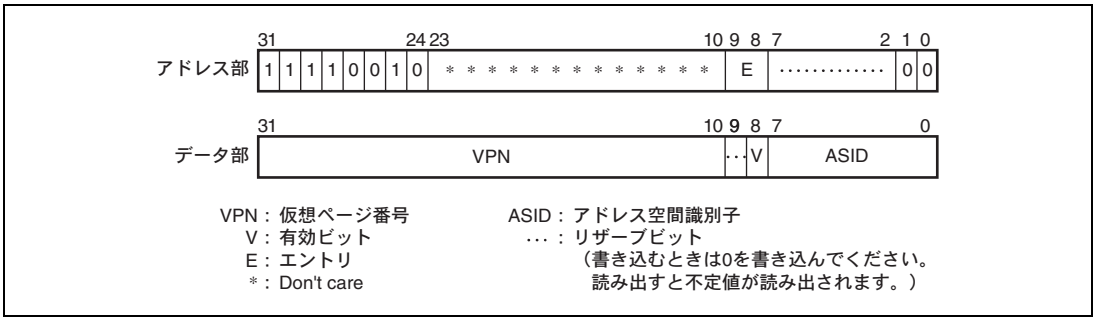


図 7.12　メモリ割り付け ITLB アドレスアレイ

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。

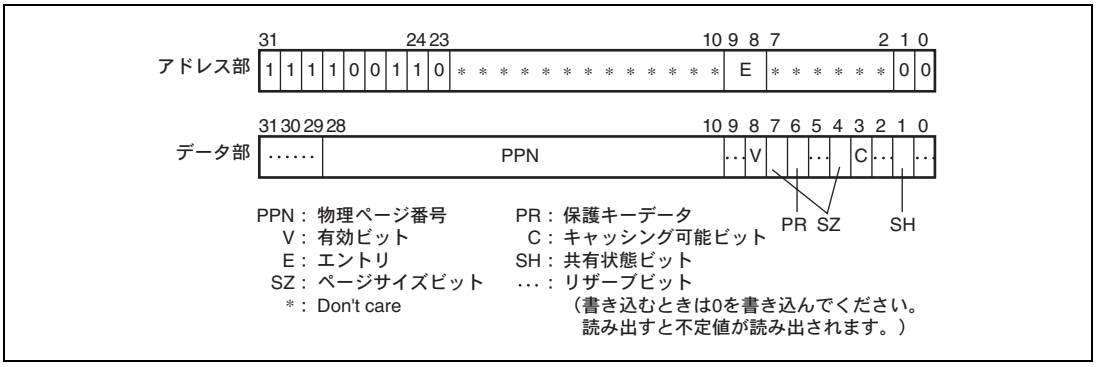


図 7.13　メモリ割り付け ITLB データアレイ

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にミスした場合は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。

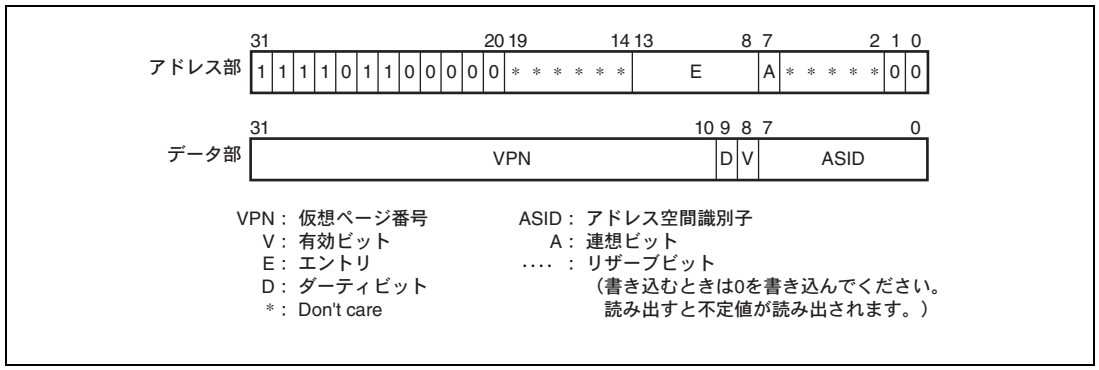


図 7.14　メモリ割り付け UTLB アドレスアレイ

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。

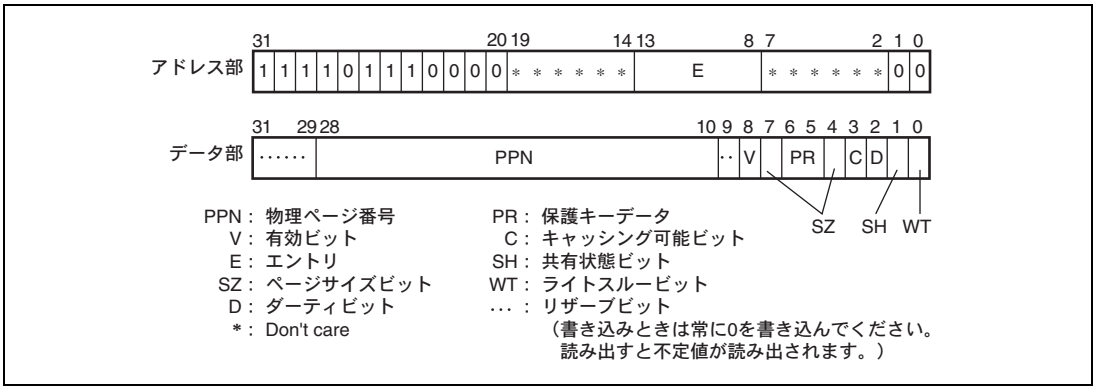


図 7.15　メモリ割り付け UTLB データアレイ

8. キャッシュ

SH4AL-DSP 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。

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

SH4AL-DSP 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。

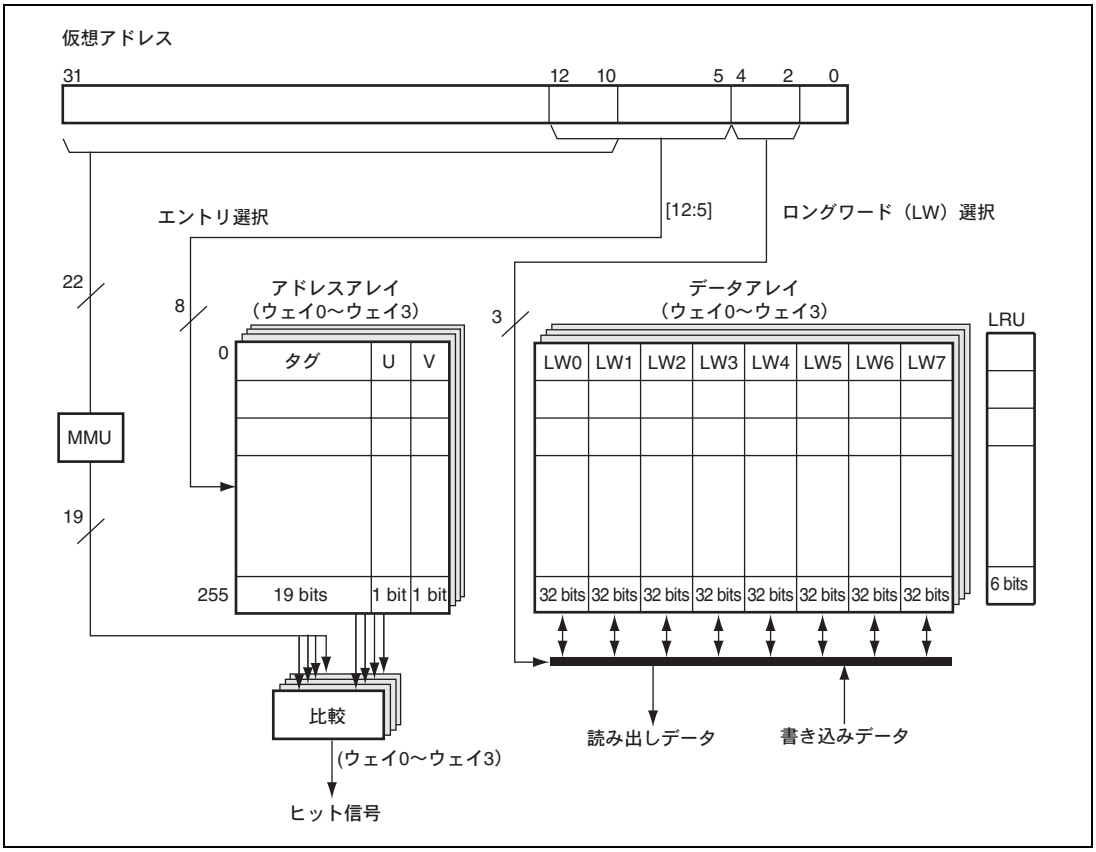


図 8.1　オペランドキャッシュ（OC）の構成

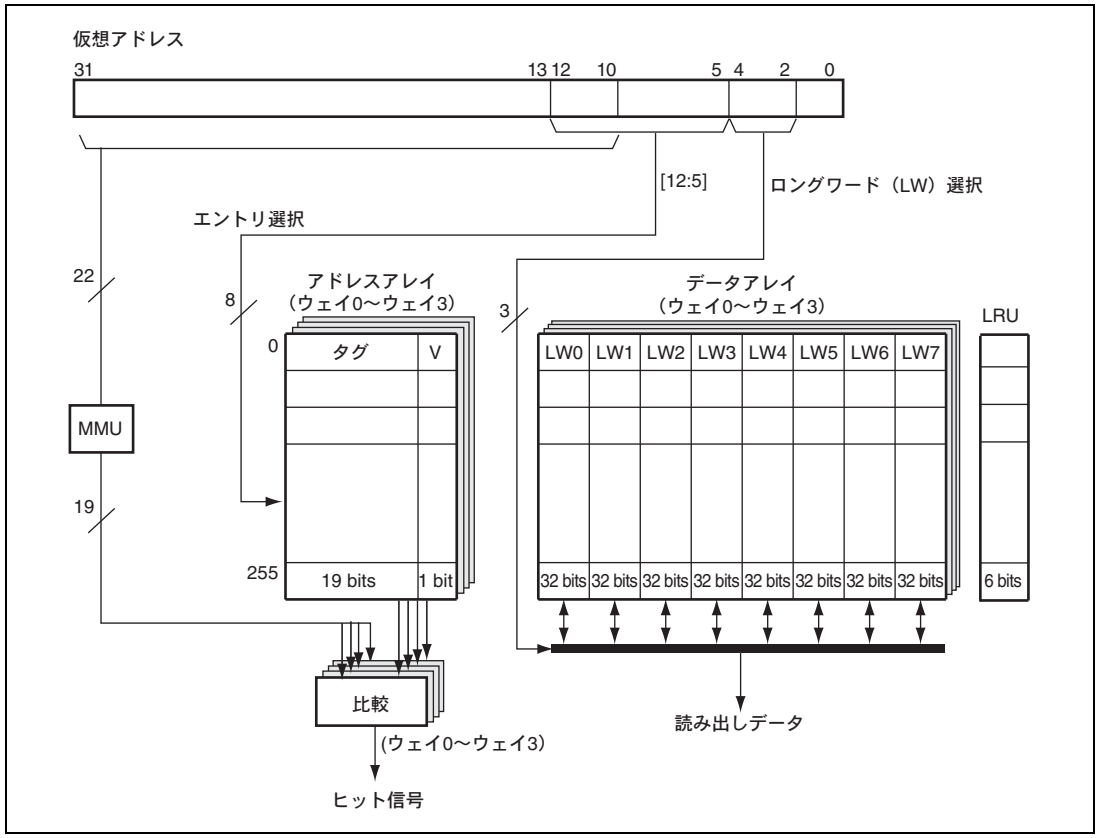


図 8.2　命令キャッシュ（IC）の構成

(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 8.2 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
キャッシュ制御レジスタ	CCR	R/W	H'FF00 001C	H'1F00 001C	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 8.3 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
キャッシュ制御レジスタ	CCR	H'0000 0000	H'0000 0000	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

ビット	ビット名	初期値	R/W	説　　明
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。
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　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、X/Y メモリ領域または U 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可能領域、X/Yメモリ領域またはU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　X/Y メモリの保護機能」および「10.4　U 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　X/Y メモリの保護機能」および「10.4　U 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出来ます。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出来ます。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出来ます。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出来ます。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出来ます。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出来ます。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

SH4AL-DSP 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。

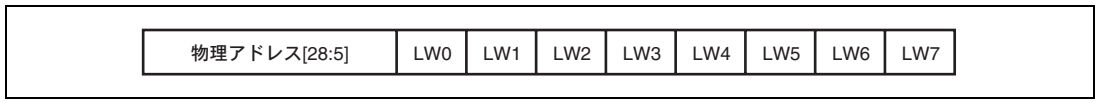


図 8.3 ライトバックバッファの構成

8.3.5 ライトスルーバッファ

SH4AL-DSP 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 32 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。

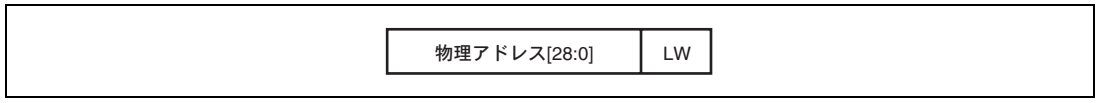


図 8.4 ライトスルーバッファの構成

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.OCI 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出来ます。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出来ます。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 IC2W 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にしてください。

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。SH4AL-DSP 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 11 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とき、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とき、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、SH4AL-DSP 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 11 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 のどれかの方法により行ってください。

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。

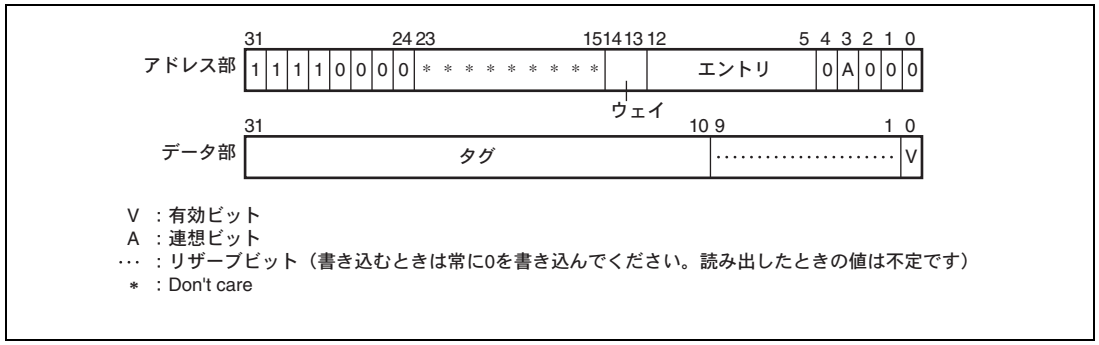


図 8.5　メモリ割り付け IC アドレスアレイ

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。

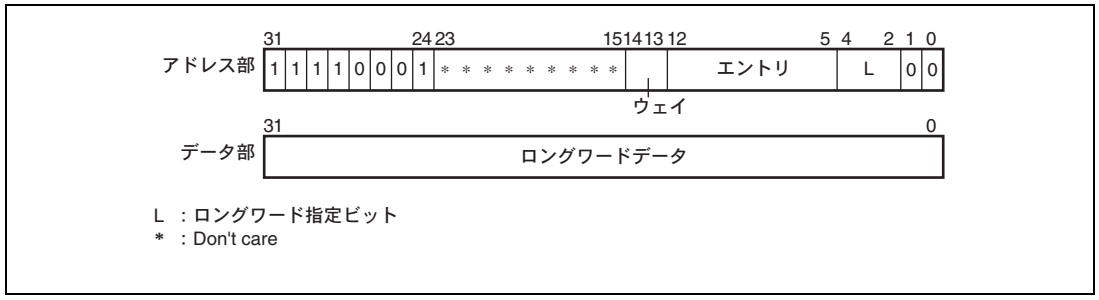


図 8.6　メモリ割り付け IC データアレイ

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。

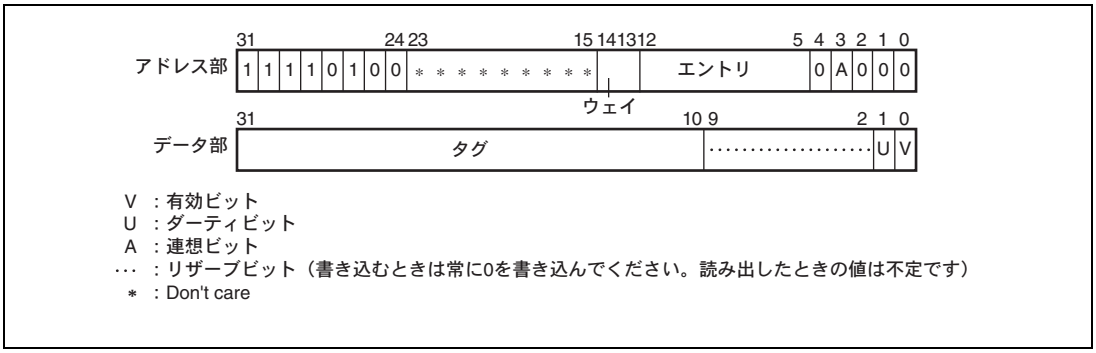


図 8.7　メモリ割り付け OC アドレスアレイ

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。

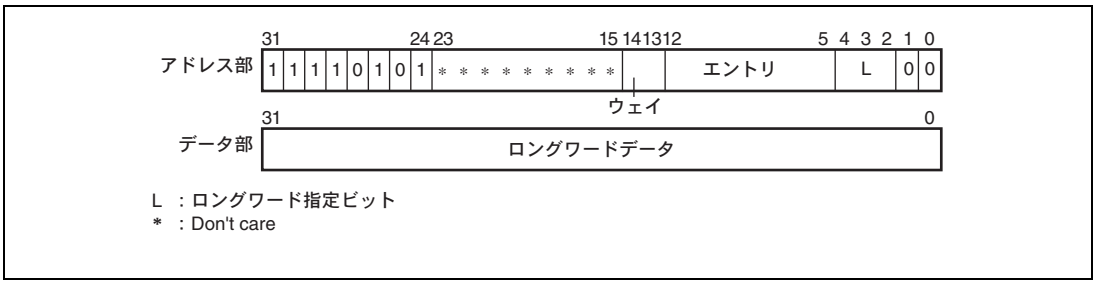


図 8.8　メモリ割り付け OC データアレイ

9. X/Y メモリ

SH4AL-DSP は X/Y 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】 X/Y 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：

  X/Y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：

  Xメモリが2ページ（ページ0、1）とYメモリが2ページ（ページ0、1）の合計4ページ存在します。

- メモリマップ：

  本メモリは、仮想アドレス空間、物理アドレス空間およびXバスとYバスのアドレス空間にそれぞれ配置されています。

  仮想アドレス空間内では、**表9.1**に示されるアドレスに配置されています。これらのアドレスは、CPUの動作モードによって、P2（SR.MD＝1の場合）、あるいはUxy（SR.MD＝0かつSR.DSP＝1の場合）と呼ばれる領域に含まれています。

表 9.1 X/Y メモリ仮想アドレス

ページ	メモリサイズ（4 ページ合計）			
	16K バイト	32K バイト	64K バイト	128K バイト
X メモリ ページ 0	H'A5007000 ～H'A5007FFF	H'A5006000 ～H'A5007FFF	H'A5004000 ～H'A5007FFF	H'A5000000 ～H'A5007FFF
X メモリ ページ 1	H'A5008000 ～H'A5008FFF	H'A5008000 ～H'A5009FFF	H'A5008000 ～H'A500BFFF	H'A5008000 ～H'A500FFFF
Y メモリ ページ 0	H'A5017000 ～H'A5017FFF	H'A5016000 ～H'A5017FFF	H'A5014000 ～H'A5017FFF	H'A5010000 ～H'A5017FFF
Y メモリ ページ 1	H'A5018000 ～H'A5018FFF	H'A5018000 ～H'A5019FFF	H'A5018000 ～H'A501BFFF	H'A5018000 ～H'A501FFFF

一方、物理アドレス空間内では、エリア1の一部に配置されています。物理アドレス空間からアクセスを行う場合、**表9.1**に示すアドレスの上位3ビットを0としたアドレスを使用します。

XバスとYバスのアドレス空間は16ビットのアドレス空間なので、**表9.1**に示すXメモリとYメモリのアドレスのそれぞれ上位16ビットを無視したアドレスを使用します。

- ポート：

  各ページは5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。Xメモリは SuperHywayバス、キャッシュ・RAM内蔵バス、Xバス、オペランドバスおよび命令バスと、Yメモリは SuperHywayバス、キャッシュ・RAM内蔵バス、Yバス、オペランドバスおよび命令バスと接続されています。仮想アドレス空間からのアクセスにはオペランドバスおよび命令バス、物理アドレス空間からのアクセスにはキャッシュ・RAM内蔵バス、XバスとYバスのアドレス空間からのアクセスにはXバスとYバス、SuperHywayバスマスタモジュールからのアクセスにはSuperHywayバスが使用されます。

- 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XメモリではSuperHywayバス、キャッシュ・RAM内蔵バス、Xバス、オペランドバス、命令バスとなり、YメモリではSuperHywayバス、キャッシュ・RAM内蔵バス、Y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

X/Y 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 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
X メモリ転送元アドレスレジスタ	XSA	R/W	H'FF00 0050	H'1F00 0050	32
Y メモリ転送元アドレスレジスタ	YSA	R/W	H'FF00 0054	H'1F00 0054	32
X メモリ転送先アドレスレジスタ	XDA	R/W	H'FF00 0058	H'1F00 0058	32
Y メモリ転送先アドレスレジスタ	YDA	R/W	H'FF00 005C	H'1F00 005C	32
X バス保護制御レジスタ	XPR	R/W	H'FF00 0060	H'1F00 0060	32
Y バス保護制御レジスタ	YPR	R/W	H'FF00 0064	H'1F00 0064	32
X バス例外アドレスレジスタ	XEA	R/W	H'FF00 0068	H'1F00 0068	32
Y バス例外アドレスレジスタ	YEA	R/W	H'FF00 006C	H'1F00 006C	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
X メモリ転送元アドレスレジスタ	XSA	不定	不定	保持	保持
Y メモリ転送元アドレスレジスタ	YSA	不定	不定	保持	保持
X メモリ転送先アドレスレジスタ	XDA	不定	不定	保持	保持
Y メモリ転送先アドレスレジスタ	YDA	不定	不定	保持	保持
X バス保護制御レジスタ	XPR	H'0000 00FC	保持	保持	保持
Y バス保護制御レジスタ	YPR	H'0000 00FC	保持	保持	保持
X バス例外アドレスレジスタ	XEA	不定	保持	保持	保持
Y バス例外アドレスレジスタ	YEA	不定	保持	保持	保持

9.2.1 内蔵メモリ制御レジスタ（RAMCR）

RAMCR は X/Y メモリの保護機能の制御を行います。

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 仮想アドレス空間からの X/Y メモリへのアクセス権を指定します。 0：SR.DSP＝0 の場合、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） SR.DSP＝1 の場合、ユーザ/特権アクセスが可能 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、X バスアドレス空間および Y バスアドレス空間からの X/Y メモリへのアクセスに対して、ITLB、UTLB、XPR および YPR 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　X/Y 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 X メモリ転送元アドレスレジスタ（XSA）

XSA は、MMUCR.AT＝0 または RAMCR.RP＝0 のときに、X メモリ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：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	XSADR												
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
	XSADR						—	—	—	—	XSSZ					
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	XSADR	不定	R/W	X メモリブロック転送元アドレス MMUCR.AT＝0 または RAMCR.RP＝0 のとき、X メモリ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	XSSZ	不定	R/W	X メモリ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、X メモリ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、XSADR の値を使用するかを選択します。 XSSZ[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 XSADR 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 Y メモリ転送元アドレスレジスタ（YSA）

YSA は、MMUCR.AT＝0 または RAMCR.RP＝0 のときに、Y メモリ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：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	YSADR												
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
	YSADR						—	—	—	—	YSSZ					
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	YSADR	不定	R/W	Y メモリブロック転送元アドレス MMUCR.AT＝0 または RAMCR.RP＝0 のとき、Y メモリ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	YSSZ	不定	R/W	Y メモリ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、Y メモリ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、YSADR の値を使用するかを選択します。 YSSZ[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 YSADR 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 X メモリ転送先アドレスレジスタ（XDA）

XDA は、MMUCR.AT＝0 または RAMCR.RP＝0 のときに、X メモリ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。

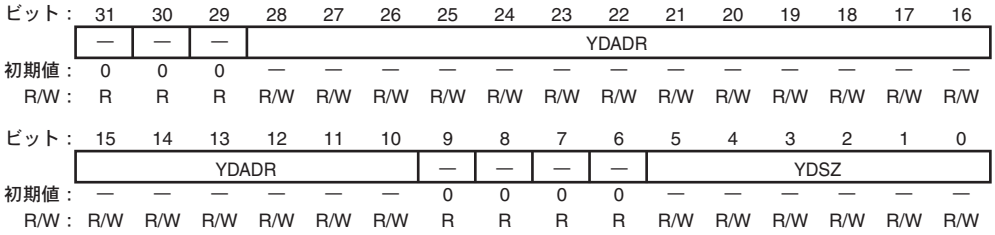
ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	XDADR												
初期値：	0	0	0	—	—	—	—	—	—	—	—	—	—	—	—	—
R/W：	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	XDADR						—	—	—	—	XDSZ					
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	XDADR	不定	R/W	X メモリブロック転送先アドレス MMUCR.AT＝0 または RAMCR.RP＝0 のとき、X メモリ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	XDSZ	不定	R/W	X メモリ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、X メモリ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、XDADR の値を使用するかを選択します。 XDSZ[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 XDADR 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 Y メモリ転送先アドレスレジスタ（YDA）

YDA は、MMUCR.AT＝0 または RAMCR.RP＝0 のときに、Y メモリ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	YDADR	不定	R/W	Y メモリブロック転送先アドレス MMUCR.AT＝0 または RAMCR.RP＝0 のとき、Y メモリ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	YDSZ	不定	R/W	Y メモリ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、Y メモリ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、YDADR の値を使用するかを選択します。 YDSZ[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 YDADR 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。

9.2.6 X バス保護制御レジスタ（XPR）

XPR は MMUCR.AT＝1 かつ RAMCR.RP＝1 のときに、ユーザプロセスによる X バスからの X メモリデータ転送命令（MOVX）アクセスが可能な領域を指定します。X バスアドレス空間のうち、

アドレス領域 {SXADR, B'00 0000 0000}～{EXADR, B'11 1111 1111}

へのアクセスが許可されます。ユーザモードでの XPR に設定された領域以外に対して X バスからアクセスするとアドレスエラー例外が発生します。

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
	SXADR						—	—	EXADR						—	—
初期値：	0	0	0	0	0	0	0	0	1	1	1	1	1	1	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	R

ビット	ビット名	初期値	R/W	説　　明
31～16	－	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。
15～10	SXADR	すべて 0	R/W	アクセス許可 X バス先頭アドレス アクセスを許可する X バスアドレス領域の先頭アドレス上位 6 ビット
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。
7～2	EXADR	すべて 1	R/W	アクセス許可 X バス最終アドレス アクセスを許可する X バスアドレス領域の最終アドレス上位 6 ビット
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。

9.2.7 Y バス保護制御レジスタ（YPR）

YPR は MMUCR.AT＝1 かつ RAMCR.RP＝1 のときに、ユーザプロセスによる Y バスからの Y メモリデータ転送命令（MOVY）アクセスが可能な領域を指定します。Y バスアドレス空間のうち、

アドレス領域 {SYADR, B'00 0000 0000}～{EYADR, B'11 1111 1111}

へのアクセスが許可されます。ユーザモードでの YPR に設定された領域以外に対して Y バスからアクセスするとアドレスエラー例外が発生します。

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
	SYADR						—	—	EYADR						—	—
初期値：	0	0	0	0	0	0	0	0	1	1	1	1	1	1	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	R

ビット	ビット名	初期値	R/W	説　　明
31～16	－	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。
15～10	SYADR	すべて 0	R/W	アクセス許可 Y バス先頭アドレス アクセスを許可する Y バスアドレス領域の先頭アドレス上位 6 ビット
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。
7～2	EYADR	すべて 1	R/W	アクセス許可 Y バス最終アドレス アクセスを許可する Y バスアドレス領域の最終アドレス上位 6 ビット
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。

9.2.8 X バス例外アドレスレジスタ（XEA）

XEA へは、MMUCR.AT＝1 かつ RAMCR.RP＝1 のときに、ユーザモードでの X バスアクセスによるアドレスエラー例外発生後に、アドレスエラーとなった X バスのアドレスがハードウェアにより設定されます。

ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	XEF	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
初期値：	—	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R/W	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
	XEA															
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	ビット名	初期値	R/W	説　　明
31	XEF	不定	R/W	X バスアドレスエラー検出フラグ XEF ビットは TEA、XEA、YEA のいずれかがハードウェアにて更新される例外が発生した場合に更新され、MOVX 命令による X バスアクセスがアドレスエラーになったときには 1 が、それ以外のときには 0 がハードウェアにより設定されます。
30～16	—	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。
15～0	XEA	不定	R/W	X バスアドレスエラー検出アドレス MOVX 命令による X バスアクセスによりアドレスエラー例外となったアドレスがハードウェアにより設定されます。

9.2.9 Y バス例外アドレスレジスタ（YEA）

YEA へは、MMUCR.AT＝1 かつ RAMCR.RP＝1 のときに、ユーザモードでのアクセスによるアドレスエラー例外発生後に、アドレスエラーとなった Y バスのアドレスがハードウェアにより設定されます。

ビット：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	YEF	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
初期値：	—	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R/W	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
	YEA															
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	ビット名	初期値	R/W	説　　明
31	YEF	不定	R/W	Y バスアドレスエラー検出フラグ YEF ビットは TEA、XEA、YEA のいずれかがハードウェアにて更新される例外が発生した場合に更新され、MOVY 命令による Y バスアクセスがアドレスエラーになった時には 1 が、それ以外のときには 0 がハードウェアにより設定されます。
30～16	—	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。
15～0	YEA	不定	R/W	Y バスアドレスエラー検出アドレス MOVY 命令による Y バスアクセスによりアドレスエラー例外となったアドレスがハードウェアにより設定されます。

9.3 動作説明

9.3.1 CPU からのアクセス

CPU からのアクセス手段として、仮想アドレスにより命令バスおよびオペランドバスから直接アクセスを行う方法と、MMU を用いて物理アドレスに変換後キャッシュ・RAM 内蔵バスからアクセスを行う方法があります。命令バスおよびオペランドバスからのアクセスは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。キャッシュ・RAM 内蔵バスからのアクセスは複数サイクル必要となります。CPU の動作モードに応じてそれぞれ以下の様になります。

- **特権モードおよび特権DSPモード（SR.MD＝1）**

  このモードでは、P2領域から直接アクセスすることができます。また、MMUを使用してP0、P3領域の仮想アドレスを本メモリへマッピングすることができます。

- **ユーザDSPモード（SR.MD＝0かつSR.DSP＝1）**

  このモードでは、Uxy領域から直接アクセスすることができます。また、MMUを使用してU0領域の仮想アドレスを本メモリへマッピングすることができます。

- **ユーザモード（SR.MD＝0かつSR.DSP＝0）**

  このモードでは、MMUを使用してU0領域の仮想アドレスを本メモリへマッピングすることができます。またRAMCRレジスタのRMDビットが1の場合には、Uxy領域から直接アクセスすることが出来ます。

9.3.2 DSP からのアクセス

DSP からのアクセスは命令の種類によりアクセス方法が異なります。

X データ転送命令および Y データ転送命令は常に X バスおよび Y バスからのアクセスになります。この場合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。また X バスからの X メモリアクセスと、Y バスからの Y メモリアクセスは同時に行うことが出来ます。

シングルデータ転送命令はアクセス手段として、仮想アドレスによりオペランドバスから直接アクセスを行う方法と、MMU を用いて物理アドレスに変換後キャッシュ・RAM 内蔵バスからアクセスを行う方法があります。オペランドバスからのアクセスは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。キャッシュ・RAM 内蔵バスからのアクセスは複数サイクル必要となります。CPU の動作モードに応じてそれぞれ以下の様になります。

- **特権DSPモード（SR.MD＝1かつSR.DSP＝1）**

  このモードでは、P2領域より直接アクセスすることができます。また、MMUを使用してP0、P3領域の仮想アドレスを本メモリへマッピングすることができます。

- **ユーザDSPモード（SR.MD＝0かつSR.DSP＝1）**

  このモードでは、Uxy領域より直接アクセスすることができます。また、MMUを使用してU0領域の仮想アドレスを本メモリへマッピングすることができます。

9.3.3 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、常に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。**表 9.1** に示すアドレスの上位 3 ビットを 0 としたアドレスを使用してください。

9.3.4 ブロック転送

X/Y 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 X/Y 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 X/Y 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 X/Y メモリへのブロック転送が開始されます。

X/Y 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 X/Y メモリ領域のアドレスに対して発行することにより、X/Y 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。

X/Y 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。

- **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X/Y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。

  X/Y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X/Yメモリへブロック転送が行われます。

  X/Y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。X/Y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。

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

  XSAレジスタのXSADRビットにXメモリ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、XSSZ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、

XSADR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。

XDAレジスタのXDADRビットにXメモリ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、XDSZ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、XDADR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。

Yメモリに対するブロック転送の設定も、Xメモリと同様にYSAおよびYDAに対して行います。

X/Yメモリ領域へのPREF命令が発行されると、XSAレジスタまたはYSA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X/Yメモリへブロック転送が行われます。

X/Yメモリ領域へのOCBWB命令が発行されると、XDAレジスタまたはYDA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。X/Y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 X/Y メモリの保護機能

SH4AL-DSP では、X/Y 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からのアクセスに対する保護機能**

  SR.DSP＝0かつRAMCR.RMD＝0のとき、ユーザモードでのUxy領域へ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、X/Yメモリ領域（特権モードまたは特権DSPモードのときP2領域の一部、ユーザDSPモードのときUxy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。

- **DSPからのアクセスに対する保護機能**

  シングルデータ転送命令に対する保護機能は、CPUからのアクセスの場合と同じです。

  Xデータ転送命令およびYデータ転送命令に対しては、MMUCR.AT＝1かつRAMCR.RP＝1のときに保護機能が働きます。この保護機能は、XPRレジスタおよびYPRレジスタに、現プロセスがアクセス可能な領域をソフトウェアにより登録することで行います。したがってプロセス切り替え時に、ソフトウェアで必要に応じてXPRレジスタおよびYPRレジスタの設定を変更してください。XPRレジスタおよびYPRレジスタに設定したアドレス範囲から外れるXデータ転送命令、Yデータ転送命令が実行された場合、アドレスエラー例外を発生し、そのアドレスをXEAレジスタおよびYEAレジスタに記録します。

  またMMUCR.AT＝1かつRAMCR.RP＝1のとき、MOVX命令やMOVY命令によりワードサイズのデータアクセスを2n番地以外に実行したりロングワードサイズのデータアクセスを4n番地以外に実行するとアドレスエラー例外が発生します。この場合のアドレスエラー例外を除き以上のX/Yメモリの保護機能を**表9.4**にまとめます。

表 9.4 X/Y 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.DSP	SR.MD	RAMCR.RMD	必ず発生する例外			起こり得る例外		
					CPU からのアクセス	MOVS 命令	MOVX 命令 MOVY 命令	CPU からのアクセス	MOVS 命令	MOVX 命令 MOVY 命令
0	x	0	0	0	アドレスエラー例外	不当命令例外	不当命令例外			
				1		不当命令例外	不当命令例外			
			1	x		不当命令例外	不当命令例外			
		1	x	x						
1	0	0	0	0	アドレスエラー例外	不当命令例外	不当命令例外			
				1		不当命令例外	不当命令例外			
			1	x		不当命令例外	不当命令例外			
		1	x	x						
	1	0	0	0	アドレスエラー例外	不当命令例外	不当命令例外			
				1		不当命令例外	不当命令例外	MMU 例外		
			1	x		不当命令例外	不当命令例外	MMU 例外		
		1	0	x				MMU 例外	MMU 例外	アドレスエラー例外
			1	x				MMU 例外	MMU 例外	

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メモリ、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 バス競合

キャッシュ・RAM 内蔵バスは命令アクセスとオペランドアクセスの共有バスです。このためキャッシュ・RAM 内蔵バス経由のアクセスは、命令アクセスとオペランドアクセスのバス競合が発生する場合があります。バス競合が発生するとメモリアクセスの性能低下を招きますので、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 CPU による本メモリアクセスでは、キャッシュ・RAM 内蔵バス経由を避け P2 領域または Uxy 領域より直接アクセスすることによってキャッシュ・RAM 内蔵バス上での競合は回避されます。

9.5.3 MMU とキャッシュの設定

CPU と DSP からキャッシュを利用してキャッシュ・RAM 内蔵バス経由で本メモリにアクセスした場合には、動作を保証しません。キャッシュを有効（CCR.ICE＝1 または CCR.OCE＝1）にして使用する場合には、P2 または Uxy 領域から命令バスおよびオペランドバス経由で直接アクセスするか、P0、P3、U0 領域からのアクセスでは MMU を有効（MMUCR.AT＝1）にして、ページ属性にキャッシング不可（C ビット＝0）を設定し、キャッシュを用いないキャッシュ・RAM 内蔵バス経由のアクセスとして使用してください。ただし、キャッシュ・RAM 内蔵バス経由のアクセスは、複数サイクル数必要になります（必要なサイクル数はバスの動作状態などにより変化します）。高い性能が必要なプログラムでは、P2 または Uxy 領域からアクセスすることを推奨します。以上の関係を**表 9.5** および**表 9.6** にまとめます。

表 9.5 MMU、キャッシュの設定（命令アクセス）

設定		仮想アドレス領域とアクセスの可否			
CCR.ICE	MMUCR.AT	P0、U0	P1	P2、Uxy	P3
0	0	○	○	◎	○
0	1	○	○	◎	○
1	0	×	×	◎	×
1	1	△	×	◎	△

表 9.6 MMU、キャッシュの設定（オペランドアクセス）

設定		仮想アドレス領域とアクセスの可否			
CCR.OCE	MMUCR.AT	P0、U0	P1	P2、Uxy	P3
0	0	○	○	◎	○
0	1	○	○	◎	○
1	0	×	×	◎	×
1	1	△	×	◎	△

【注】 ◎：可（推奨）

○：可

△：可（ただし、MMU のページ属性を C ビット=0 に設定すること）

×：不可

9.5.4 X/Y メモリのコヒーレンシ

X/Y メモリに命令を配置する場合、X/Y 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（X/Y 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。

また同一アドレスに対して、命令バス、オペランドバス、X バスまたは Y バスからのアクセスと、キャッシュ・RAM 内蔵バスからのアクセスの両方を行わないでください。行った場合のコヒーレンシは保証されません。

9.5.5 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。

10. U メモリ

SH4AL-DSP は U 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】 U 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。

10.1 特長

- 容量：

  128Kバイト、256Kバイト、512Kバイト、1Mバイトから選択可能です。

- アクセス方法：

  命令フェッチやランダムなアクセスに適したキャッシャブルアクセスと、リードバッファを用いてシーケンシャルなオペランドアクセスに最適化した非キャッシャブルアクセスが可能です。

- メモリマップ：

  Uメモリは、仮想アドレス空間および物理アドレス空間で、それぞれ**表10.1**に示されるアドレスに配置されています。

  仮想アドレス空間のアドレスには、CPUの動作モードによって、P2（SR.MD＝1の場合）、あるいはUxy（SR.MD＝0かつSR.DSP＝1の場合）領域からアクセス可能です。このアドレスを用いたアクセスは常に非キャッシャブルアクセスとなります。

  物理アドレス空間のアドレスには、U0、P0、P1またはP3領域からアクセス可能です。このアドレスを用いたアクセスが、キャッシャブルアクセスとなるか、非キャッシャブルアクセスとなるかは、CCRレジスタ、MMUCRレジスタおよびTLBの設定に従います。

表 10.1 U メモリアドレス

アドレス空間	メモリサイズ			
	128K バイト	256K バイト	512K バイト	1M バイト
仮想アドレス	H'A55F0000 ～H'A560FFFF	H'A55E0000 ～H'A561FFFF	H'A55C0000 ～H'A563FFFF	H'A5580000 ～H'A567FFFF
物理アドレス	H'055F0000 ～H'0560FFFF	H'055E0000 ～H'0561FFFF	H'055C0000 ～H'0563FFFF	H'05580000 ～H'0567FFFF

- ポート：

  Uメモリは3本の独立した読み出し／書き込みポートを持ち、オペランドバス、キャッシュ・RAM内蔵バスおよびSuperHywayバスと接続されています。非キャッシャブルのオペランドアクセスにはオペランドバス、命令フェッチおよびキャッシャブルのオペランドアクセスにはキャッシュ・RAM内蔵バス、SuperHywayバスマスタモジュールからのアクセスにはSuperHywayバスが使用されます。

- 優先順位：

  Uメモリ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、キャッシュ・RAM内蔵バス、オペランドバスとなります。

10.2 レジスタの説明

U メモリに関するレジスタは以下のとおりです。

表 10.2 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

【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 10.3 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
内蔵メモリ制御レジスタ	RAMCR	H'0000 0000	H'0000 0000	保持	保持

10.2.1 内蔵メモリ制御レジスタ（RAMCR）

RAMCR は U メモリの保護機能の制御を行います。

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 仮想アドレス空間からの U メモリへのアクセス権を指定します。 0:SR.DSP=0 の場合、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） SR.DSP＝1 の場合、ユーザ/特権アクセスが可能 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 U メモリへのアクセスに対して、ITLB および 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「10.4　U 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。

10.3　動作説明

10.3.1　CPU からのアクセス

(1) 命令フェッチ

CPU からの命令フェッチ手段として、非キャッシャブルアクセスとキャッシャブルアクセスがあります。

非キャッシャブルアクセスでは、キャッシュ・RAM 内蔵バス経由のアクセスとなり、一回の命令フェッチに複数サイクルかかります。キャッシャブルアクセスでは、外部メモリと同じように U メモリの内容を IC にキャッシングします。キャッシャブルアクセスではキャッシュヒットの場合、1 サイクルで命令フェッチ可能なため、高い性能を要求するプログラムではキャッシャブルアクセスを推奨します。ただしこの場合 IC と U メモリのコヒーレンシはソフトウェアにより保証してください。たとえばキャッシャブルで走行するプログラムを書きかえる場合には、書き換え後に ICBI 命令により当該部分を無効化するか、または CCR.ICI＝1 書き込みにより IC 全部を無効化してから、当該プログラムへ分岐するようにしてください。

CPU からの命令フェッチが、非キャッシャブルアクセスになるか、キャッシャブルアクセスになるかは外部メモリの場合と同様です。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。

(2) オペランドアクセス

CPU からのオペランドアクセス手段として、非キャッシャブルアクセスとキャッシャブルアクセスがあります。

非キャッシャブルのリードアクセスは、リードバッファを経由したアクセスとなります。リードバッファは 1 ライン 32 バイトのバッファ 2 本で構成されており、それまでに非キャッシャブルでリードアクセスしたラインを最大 2 ラインまで保持しています。非キャッシャブルのリードアクセスでは、リードバッファにヒットした場合、1 サイクルでアクセス可能です。リードバッファにミスした場合、U メモリから要求されたデータを含む 32 バイトを読み出し、CPU へ返すとともに、リードバッファを更新します。このアクセスには複数サイクルかかります。2 本あるリードバッファのどちらを更新するかは LRU アルゴリズムを用いて決定します。非キャッシャブルのライトアクセスでは U メモリを直接更新するとともに、当該ラインがリードバッファに保持されていた場合には、無効化を行います。DMAC などの SuperHyway バスマスタモジュールが U メモリを書き換えた場合にもリードバッファの無効化をハードウェアが行いますので、ソフトウェアでコヒーレンシを保証する必要はありません。

キャッシャブルアクセスでは、外部メモリと同じように U メモリの内容を OC にキャッシングします。この場合、OC と U メモリのコヒーレンシはソフトウェアにより保証してください。たとえばキャッシャブルのライトアクセスを行った領域を DMAC により読み出す場合には、あらかじめ OCBP 命令または OCBWB 命令によりライトバックを行っておくか、DMAC から PURGE または FLUSH トランザクションを発行することによりコヒーレンシを保証してください。

CPU からのオペランドアクセスが、非キャッシャブルアクセスになるか、キャッシャブルアクセスになるかは外部メモリの場合と同様です。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。

10.3.2 DSP からのアクセス

DSP からの U メモリに対するアクセスは、シングルデータ転送命令のみ可能です。X データ転送命令および Y データ転送命令ではアクセスできません。

シングルデータ転送命令によるアクセスは CPU からのオペランドアクセスと同様です。

10.3.3 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、常に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。**表 10.1** に示す物理アドレスを使用してください。

10.4 U メモリの保護機能

SH4AL-DSP では、U 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からのアクセスに対する保護機能

  SR.DSP＝0かつRAMCR.RMD=0のとき、ユーザモードでのUxy領域へ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、Uメモリ領域（特権モードまたは特権DSPモードのときP2領域の一部、ユーザDSPモードのときUxy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。この場合、アドレス変換は行いません。ただし、Uメモリ領域をマッピングするページのPPNフィールドには、マッピングされるUメモリの物理アドレスを登録してください。

- DSPからのアクセスに対する保護機能

  シングルデータ転送命令に対する保護機能は、CPUからのアクセスの場合と同じです。

  Xデータ転送命令およびYデータ転送命令ではUメモリにアクセスできません。

  以上を**表10.4**にまとめます。

表 10.4 U 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.DSP	SR.MD	RAMCR.RMD	必ず発生する例外	起こり得る例外
0	*	0	0	0	アドレス エラー例外	−
				1	−	−
			1	*	−	−
		1	*	*	−	−
1	0	0	0	0	アドレス エラー例外	−
				1	−	−
			1	*	−	−
		1	*	*	−	−
	1	0	0	0	アドレス エラー例外	−
				1	−	MMU 例外
			1	*	−	MMU 例外
		1	*	*	−	MMU 例外

【記号説明】 *：Don't care

10.5 使用上の注意

10.5.1 スリープモード

スリープモード中も、DMAC などの SuperHyway バスマスタモジュールから本メモリへアクセス可能です。

11. 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。

11.1 CPU 命令

【注】 CPU 命令のうち、DSP 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、「11.2 CPU 命令（DSP 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

11.1.1 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
```

11.1.2 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
```

11.1.3 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
```

11.1.4 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。

11.1.5 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

11.1.6 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

11.1.7 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

11.1.8 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

11.1.9 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

11.1.10 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

11.1.11 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+      ;
```

11.1.12 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
```

11.1.13 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
```

11.1.14 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。
```

11.1.15 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。

11.1.16 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。

11.1.17 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)
```

11.1.18 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
```

11.1.19 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
```

11.1.20 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)。
```

11.1.21 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
```

11.1.22 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
```

11.1.23 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

11.1.24 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

11.1.25 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

11.1.26 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

11.1.27 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

11.1.28 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

11.1.29 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

11.1.30 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）

11.1.31 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

11.1.32 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）

11.1.33 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）

11.1.34 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

11.1.35 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

11.1.36 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

11.1.37 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

11.1.38 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
```

11.1.39 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）

11.1.40 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
```

11.1.41 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
```

11.1.42 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
```

11.1.43 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
```

11.1.44 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
```

11.1.45 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。
```

11.1.46 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
```

11.1.47 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。

11.1.48 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。

11.1.49 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。

11.1.50 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。

11.1.51 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

11.1.52 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

11.1.53 ROTCL　ROTate with Carry Left　シフト命令

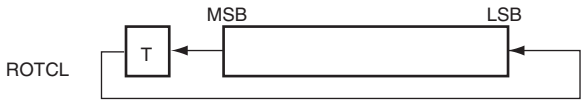
T ビット付き

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
```

11.1.54 ROTCR ROTate with Carry Right シフト命令

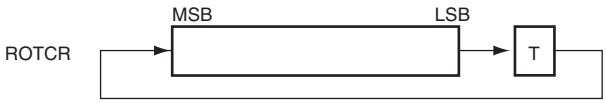
T ビット付き

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
```

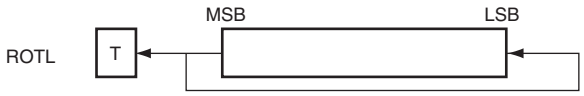
11.1.55 ROTL ROTate Left シフト命令

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
```

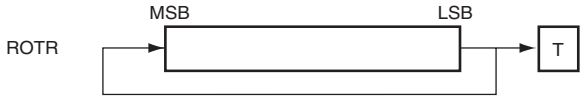
11.1.56 ROTR ROTate Right シフト命令

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
```

11.1.57 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。たと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

11.1.58 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

11.1.59 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
```

11.1.60 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
```

11.1.61 SHAD SHift Arithemetic Dynamically シフト命令

ダイナミック算術

シフト

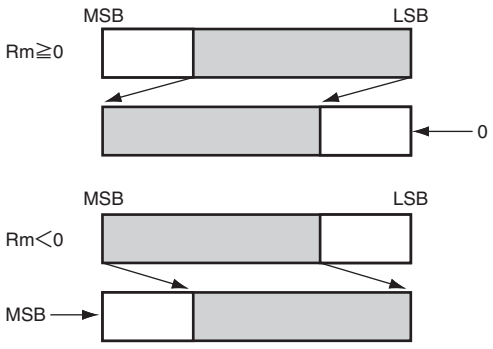
書式	動作概略	命令コード	実行ステート	T ビット
SHAD Rm, Rn	Rm≧0 のとき、Rn<<Rm→Rn Rm<0 のとき、 Rn>>Rm→[MSB→Rn]	0100nnnnmmmm1100	1	—

(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）のときは 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
```

11.1.62 SHAL SHift Arithmetic Left シフト命令

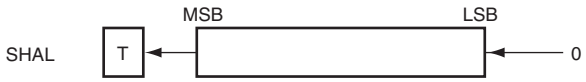
1 ビット左算術

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
```

11.1.63 SHAR SHift Arithmetic Right シフト命令

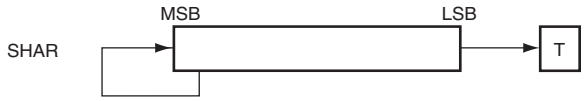
1 ビット右算術

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
```

11.1.64 SHLD SHift Logical Dynamically シフト命令

ダイナミック論理

シフト

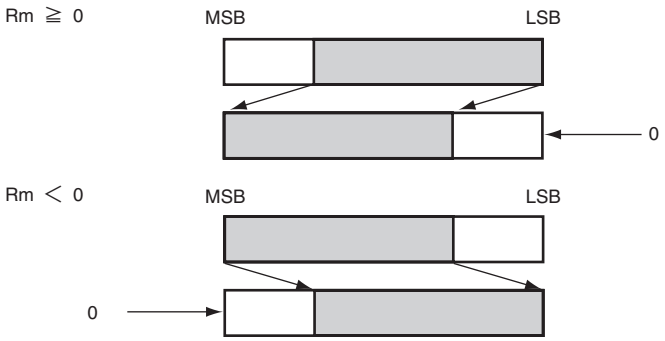
書式	動作概略	命令コード	実行ステート	T ビット
SHLD Rm, Rn	Rm≧0 のとき、Rn<<Rm→Rn Rm<0 のとき、 Rn>>Rm→[0→Rn]	0100nnnnmmmm1101	1	—

(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）のときは 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
```

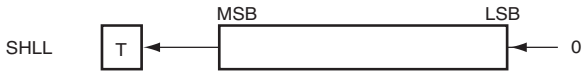
11.1.65 SHLL SHift Logical Left シフト命令

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
```

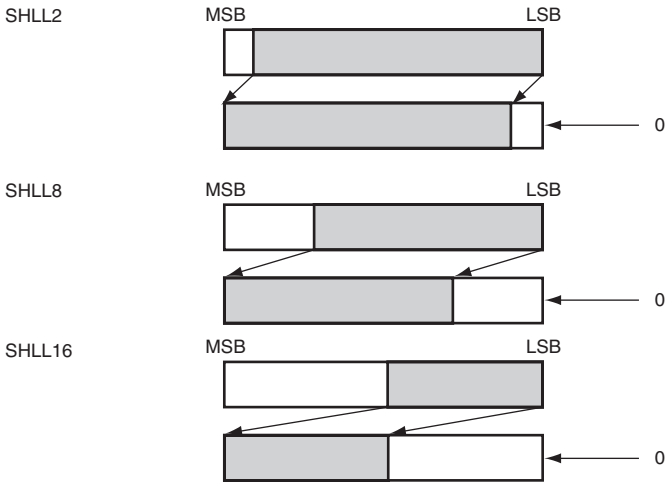
11.1.66 SHLLn　　n bits SHift Logical Left　　シフト命令

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
```

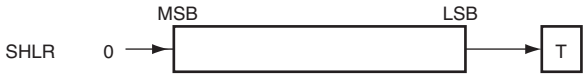
11.1.67 SHLR SHift Logical Right シフト命令

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
```

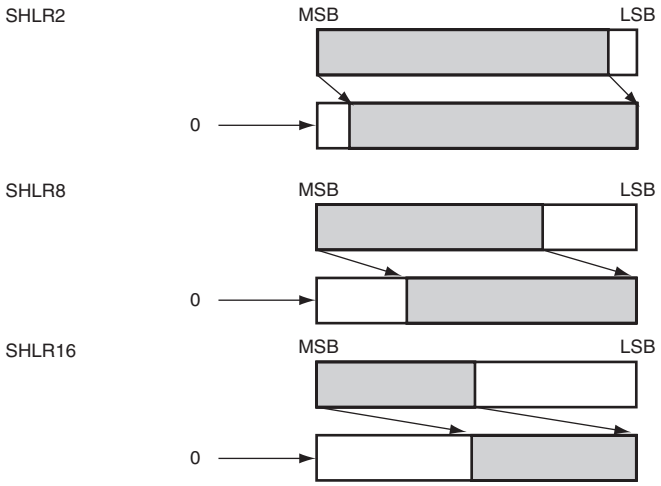
11.1.68 SHLRn n bits SHift Logical Right シフト命令

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
```

11.1.69 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

11.1.70 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

11.1.71 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

11.1.72 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
```

11.1.73 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
```

11.1.74 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
```

11.1.75 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
```

11.1.76 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

11.1.77 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。

11.1.78 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

11.1.79 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。

11.1.80 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。

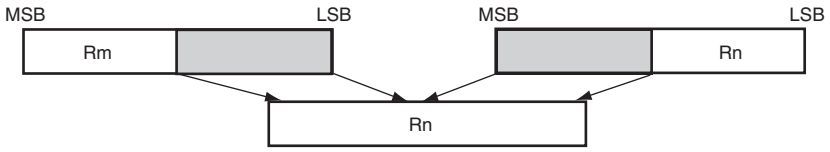
11.1.81 XTRCT eXTRaCT データ転送命令

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
```

11.2 CPU 命令（DSP 関係）

CPU 命令のうち、DSP をサポートする CPU 命令、および SH4AL-DSP、SH-4A で機能の一部に差分のある命令を本節に記載します。

11.2.1 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+4/6→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+4/6）を格納します。BSR の遅延スロット命令が 32 ビット DSP 命令のときに+6、それ以外のときに+4 となります。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);
     if(is_32bit_instruction(temp+2))
          PR = PC + 6;
     else  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

11.2.2 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+4/6→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+4/6）を格納します。BSRF の遅延スロット命令が 32 ビット DSP 命令のときに+6、それ以外のときに+4 となります。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;
     if(is_32bit_instruction(temp+2))
          PR = PC +6;
     else PR = PC + 4;
     PC = PC + 4 + R[n];
     Delay_Slot(temp+2);
}
```

(4) 発生する可能性がある例外

- スロット不当命令例外

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。
```

11.2.3 CLRDMXY Clear DMX DMY システム制御命令

モジュロアドレッシングモードの解除

書式	動作概略	命令コード	実行ステート	T ビット
CLRDMXY	0→DMX(SR[10]) 0→DMY(SR[11])	0000000010001000	1	—

(1) 説明

SR の DMX ビットおよび DMY ビットのリセットを行い、X ポインタ用モジュロアドレッシングモードを解除します。

(2) 注意

SR レジスタの DSP ビットが 0 のときに、遅延スロット以外で CLRDMXY を実行すると一般不当命令例外となり，遅延スロットで実行するとスロット不当命令例外となります。

(3) 動作内容

```
CLRDMXY( ) /* CLRDMXY */
{
        SR.DMX = 0;
        SR.DMY = 0;
        PC += 2;
}
```

(4) 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

11.2.4 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+4/6→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+4/6）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。JSR の遅延スロット命令が 32 ビット DSP 命令のときに+6、それ以外のときに+4 となります。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;
     if(is_32bit_instruction(temp+2))
          PR = PC +6;
     else 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

11.2.5 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 （一部特権命令）

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	Rm, MOD	Rm→MOD	0100mmmm01011110	1	—
LDC	Rm, RE	Rm→RE	0100mmmm01111110	1	—
LDC	Rm, RS	Rm→RS	0100mmmm01101110	1	—
LDC.L	@Rm+, SR	(Rm)→SR, Rm+4→Rm	0100mmmm00000111	4	LSB
LDC.L	@Rm+, MOD	(Rm)→MOD, Rm+4→Rm	0100mmmm01010111	1	—
LDC.L	@Rm+, RE	(Rm)→RE, Rm+4→Rm	0100mmmm01110111	1	—
LDC.L	@Rm+, RS	(Rm)→RS, Rm+4→Rm	0100mmmm01100111	1	—

(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

LDC Rm, SR、LDC.L @Rm, SR は SR レジスタの DSP ビットが 0 のときに特権命令となり、SR レジスタの全ビットの更新ができます。DSP ビットが 1（特権 DSP モードあるいはユーザ DSP モード）のときは特権命令ではなくなり、その結果、ユーザ DSP モードで命令実行が可能です。ただしユーザ DSP モードでは、SR レジスタの一部のビット（DSP 関連ビット）のみ書き込みができます。

これ以外の 6 つの命令は、ユーザモードで実行することができます。

遅延分岐命令直後の遅延スロットに LDC Rm, SR 命令、LDC.L @Rm+, SR 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。

(3) 動作内容

```
LDCSR(long m)     /* LDC Rm,SR : Privileged conditionally */
{
     if(SR.MD==1)
     {
        SR = R[m] & 0x7FFF1FFF;
     }
     else if(SR.DSP==1)
     {
        SR = SR & 700003F3 | R[m] & 0x0FFF1C0C;
     }
```

```
        PC += 2;
}

LDCMOD(long m)    /* LDC Rm,MOD */
{
MOD = R[m];
PC += 2;
}

LDCRE(long m)     /* LDC Rm,RE */
{
RE = R[m];
PC += 2;
}

LDCRS(long m)     /* LDC Rm,RS */
{
RS = R[m];
PC += 2;
}

LDCMSR (long m)   /* LDC.L @Rm+,SR : Privileged conditionally */
{
    if(SR.MD==1)
    {
        SR = Read_Long(R[m]) & 0x7FFF1FFF
    }
     else if(SR.DSP==1)
     {
        SR = SR & 700003F3 | Read_Long(R[m]) & 0x0FFF1C0C;
     }
     R[m] += 4;
     PC += 2;
}

LDCMMOD(long m)   /*LDC.L @Rm+,MOD */
{
MOD = Read_Long(R[m]);
```

```
R[m] += 4;
PC += 2;
}

LDCMRE(long m)    /*LDC.L @Rm+,RE */
{
RE = Read_Long(R[m]);
R[m] += 4;
PC += 2;
}

LDCMRS(long m)    /*LDC.L @Rm+,RS */
{
RS = Read_Long(R[m]);
R[m] += 4;
PC += 2;
}
```

(4) 使用例

```
LDC     R0,SR           ;実行前    R0=H'FFFFFFFF,SR=H'00000000
                        ;実行後    SR=H'70001FFF,T=1
LDC.L   @R15+,GBR       ;実行前    R15=H'10000000,
                        ;@R15+ H'12345678,GBR=H'EDCBA987
                        ;実行後    R15=H'10000004,GBR=@H'10000000
```

(5) 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

11.2.6 LDRC LoaD RC register システム制御命令

RC カウンタの設定および拡張リピート制御中であることを示す RE[0]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
LDRC Rm	Rm[11:0]→RC(SR[27:16])、 1→RE[0]	0100mmmm00110100	2	—
LDRC #imm	0→RC(SR[27:24]) imm→RC(SR[23:16]) 1→RE[0]	10001010iiiiiiii	2	—

(1) 説明

Rm レジスタ[11:0]もしくは 8 ビット定数 imm を SR の RC[11:0]に設定し、拡張リピート制御中を示す RE[0]に 1 をセットします。

繰り返し回数を SR レジスタの RC カウンタに設定します。オペランドがレジスタの場合は下位 12 ビットで繰り返し回数を指定します。イミディエイトデータの場合は、8 ビットで繰り返し回数を指定します。このとき RC の上位 4 ビットは、ゼロ詰めされます。

また、繰り返し制御フラグを SR レジスタの RF1,RF0 にセットします。

LDRC 命令の実行とともに RE レジスタの LSB に 1 がセットされ、次の命令以降が拡張リピート制御の対象となります。拡張リピート制御についての詳細は**「6.3.2 拡張リピート制御命令」**を参照してください。

(2) 注意

LDRC 命令の使用についてはいくつかの制限があります。詳しくは、**「6.3.2 拡張リピート制御命令」**を参照してください。

(3) 動作内容

```
LDRC(long m) /* LDRC Rm */
{
     long temp;
     temp = (R[m] & 0x00000FFF)<<16;
     SR &= 0xF000FFFF;
     SR |= temp;
     RF1 = Repeat_Control_Flag1;
     RF0 = Repeat_Control_Flag0;
     PC += 2;
     RE[0] = 1;
}
```

```
LDRCI(long i) /* LDRC #imm*/
{
     long temp;
     temp = (R[m] & 0x000000FF)<<16;
     SR &= 0xF000FFFF;
     SR| = temp;
     RF1 = Repeat_Control_Flag1;
     RF0 = Repeat_Control_Flag0;
     PC += 2;
     RE[0] = 1;
}
```

(4) 発生する可能性のある例外

- 一般不当命令例外
- スロット不当命令例外

11.2.7 LDRE LoaD effective address to RE register システム制御命令

繰り返し終了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
LDRE @(disp*,PC)	disp×2 ＋PC→RE	10001110dddddddd	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

ソースオペランドの実効アドレス値を繰り返し終了レジスタ RE に格納します。実効アドレスは PC にディスプレースメントを加えたアドレスです。PC は、本命令の 4 バイト後のアドレスです。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、-256 バイトから+254 バイトの範囲になります。

(2) 注意

RE レジスタに指定する実効アドレス値は実際の繰り返し終了アドレスとは異なります。詳しくは、**「第 6 章 DSP ユニット」**内の**「表 6.5 RS および RE のアドレス設定ルール」**を参照してください。

本命令を遅延分岐命令の直後に配置すると、PC は分岐先の"先頭アドレス＋2"になります。

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。また、本命令をリピートループ中で使用する場合にはいくつかの制限があります。詳しくは、「6.3.1 **互換リピート制御命令**」および「6.3.2 **拡張リピート制御命令**」を参照してください。

(3) 動作内容

```
LDRS(long d)      /* LDRE @(disp,PC) */
{
     long disp;

     if ((d&0x80) == 0) disp = (0x000000FF & (long)d);
     else disp = (0xFFFFFF00 | (long)d);
     RE = PC + (disp<<1);
     PC += 2;
}
```

(4) 発生する可能性のある例外

- 一般不当命令例外
- スロット不当命令例外

11.2.8 LDS LoaD to DSP System register システム制御命令

DSP 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,DSR	Rm→DSR	0100mmmm01101010	1	—
LDS	Rm,A0	Rm→A0	0100mmmm01111010	1	—
LDS	Rm,X0	Rm→X0	0100mmmm10001010	1	—
LDS	Rm,X1	Rm→X1	0100mmmm10011010	1	—
LDS	Rm,Y0	Rm→Y0	0100mmmm10101010	1	—
LDS	Rm,Y1	Rm→Y1	0100mmmm10111010	1	—
LDS.L	@Rm+,DSR	(Rm)→DSR,Rm+4→Rm	0100mmmm01100110	1	—
LDS.L	@Rm+,A0	(Rm)→A0,Rm+4→Rm	0100mmmm01110110	1	—
LDS.L	@Rm+,X0	(Rm)→X0,Rm+4→Rm	0100mmmm10000110	1	—
LDS.L	@Rm+,X1	(Rm)→X1,Rm+4→Rm	0100mmmm10010110	1	—
LDS.L	@Rm+,Y0	(Rm)→Y0,Rm+4→Rm	0100mmmm10100110	1	—
LDS.L	@Rm+,Y1	(Rm)→Y1,Rm+4→Rm	0100mmmm10110110	1	—

(1) 説明

ソースオペランドを DSP システムレジスタ DSR、および DSP データレジスタ A0、X0、X1、Y0、Y1 に格納します。

(2) 注意

特になし

(3) 動作内容

```
LDSDSR(long m)    /* LDS Rm,DSR */
{
    DSR = R[m] & 0x0000000F;
    PC += 2;
}

LDSA0(long m)     /* LDS Rm,A0 */
{
    A0 = R[m];
    if((A0&0x80000000) == 0) A0G = 0x00;
    else A0G = 0xFF;
```

```
    PC += 2;
}

LDSX0(long m)     /* LDS Rm,X0 */
{
    X0 = R[m];
    PC += 2;
}

LDSX1(long m)     /* LDS Rm,X1 */
{
    X1 = R[m];
    PC += 2;
}

LDSY0(long m)     /* LDS Rm,Y0 */
{
    Y0 = R[m];
    PC += 2;
}

LDSY1(long m)     /* LDS Rm,Y1 */
{
    Y1 = R[m];
    PC += 2;
}

LDSMDSR(long m)   /* LDS.L @Rm+,DSR */
{
    DSR = Read_Long(R[m])&0x0000000F;
    R[m] += 4;
    PC += 2;
}

LDSMA0(long m)    /* LDS.L @Rm+,A0 */
{
    A0 = Read_Long(R[m]);
    if ((A0&0x80000000) == 0) A0G = 0x00;
```

```
    else A0G = 0xFF;
    R[m] += 4;
    PC += 2;
}

LDSMX0(long m)    /* LDS.L @Rm+,X0 */
{
    X0 = Read_Long (R[m]);
    R[m] += 4;
    PC += 2;
}
LDSMX1(long m)    /* LDS.L @Rm+,X1 */
{
    X1 = Read_Long (R[m]);
    R[m] += 4;
    PC += 2;
}

LDSMY0(long m)    /* LDS.L @Rm+,Y0 */
{
    Y0 = Read_Long(R[m]);
    R[m] += 4;
    PC += 2;
}

LDSMY1(long m)    /* LDS.L @Rm+,Y1 */
{
    Y1 = 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        R15=H'10000004,MACL=@H'10000000
```

(5) 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

11.2.9 LDRS LoaD effective address to RS register システム制御命令

繰り返し開始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
LDRS @(disp*,PC)	disp×2+PC→RS	10001100dddddddd	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

ソースオペランドの実効アドレス値を繰り返し開始レジスタ RS に格納します。実効アドレスは PC にディスプレースメントを加えたアドレスです。PC は、本命令の 4 バイト後のアドレスです。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、-256 バイトから+254 バイトの範囲になります。

(2) 注意

繰り返し（ループ）プログラムが 3 命令以下のときは、RS レジスタに指定する実効アドレス値は実際の繰り返し開始アドレスとは異なります。詳しくは、**「表 6.5 RS および RE のアドレス設定ルール」**を参照してください。

本命令を遅延分岐命令の直後に配置すると、PC は分岐先の"先頭アドレス＋2"になります。

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。また、本命令をリピートループ中で使用する場合にはいくつかの制限があります。詳しくは、「6.3.1 **互換リピート制御命令**」および「6.3.2 **拡張リピート制御命令**」を参照してください。

(3) 動作内容

```
LDRS(long d)    /* LDRS @(disp,PC) */
{
      long disp;

      if ((d&0x80) == 0) disp = (0x000000FF & (long)d);
      else disp =(0xFFFFFF00 | (long)d);
      RS = PC + (disp<<1);
      PC += 2;
}
```

(4) 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

11.2.10 SETDMX Set DMX システム制御命令

X ポインタ用モジュロアドレッシングモードの設定

書式	動作概略	命令コード	実行ステート	T ビット
SETDMX	1→DMX(SR[10]) 0→DMY(SR[11])	0000000010011000	1	—

(1) 説明

SR の DMX ビットのセットおよび DMY ビットのリセットを行い、X ポインタ用モジュロアドレッシングモードを有効にします。

(2) 注意

SR レジスタの DSP ビットが 0 のときに、遅延スロット以外で SETDMX を実行すると一般不当命令例外となり，遅延スロットで実行するとスロット不当命令例外となります。

(3) 動作内容

```
SETDMX( ) /* SETDMX */
{
     SR.DMX = 1;
     SR.DMY = 0;
     PC += 2;
}
```

(4) 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

11.2.11 SETDMY Set DMY システム制御命令

Ｙポインタ用モジュロアドレッシングモードの設定

書式	動作概略	命令コード	実行ステート	Ｔビット
SETDMY	0→DMX(SR[10]) 1→DMY(SR[11])	0000000011001000	1	―

(1) 説明

SR の DMX ビットのリセットおよび DMY ビットのセットを行い、Y ポインタ用モジュロアドレッシングモードを有効にします。

(2) 注意

SR レジスタの DSP ビットが 0 のときに､遅延スロット以外で SETDMY を実行すると一般不当命令例外となり，遅延スロットで実行するとスロット不当命令例外となります。

(3) 動作内容

```
SETDMY( ) /* SETDMY */
{
        SR.DMX = 0;
        SR.DMY = 1;
        PC += 2;
}
```

(4) 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

11.2.12 SETRC SET repeat count RC システム制御命令

RC カウンタの設定および繰り返し制御フラグの設定

書式	動作概略	命令コード	実行ステート	T ビット
SETRC Rm	Rm [11:0] →RC(SR[27:16])	0100mmmm00010100	2	—
SETRC #imm	imm→RC (SR [23:16])、0→RC(SR[27:24])	10000010iiiiiiii	2	—

(1) 説明

繰り返し回数を SR レジスタの RC カウンタに設定します。オペランドがレジスタの場合は、下位 12 ビットで繰り返し回数を指定します。イミディエイトデータの場合は、8 ビットで繰り返し回数を指定します。このとき RC の上位 4 ビットは、ゼロ詰めされます。

また、繰り返し制御フラグを SR レジスタの RF1、RF0 ビットにセットします。

SETRC 命令の使用についてはいくつかの制限があります。詳しくは、**「6.3.1 互換リピート制御命令」**を参照してください。

(2) 注意

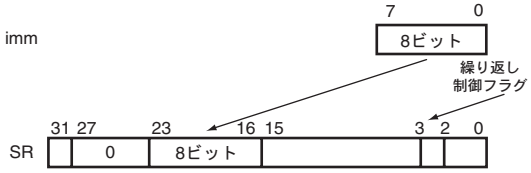
特になし

(3) 動作内容

```
SETRC(long m)     /* SETRC Rm */
{
      long temp;

      temp=(R[m] & 0x00000FFF)<<16;
      SR&=0xF000FFFF;
      SR|=temp;
      RF1=Repeat_Control_Flag1;
      RF0=Repeat_Control_Flag0;
      PC+=2;
}

SETRCI(long i)    /* SETRC #imm */
{
      long temp;

      temp=((long)i & 0x000000FF)<<16;
```

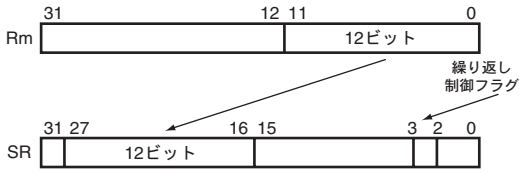
```
        SR&=0xF000FFFF;
        SR|=temp;
        RF1=Repeat_Control_Flag1;
        RF0=Repeat_Control_Flag0;
        PC+=2;
}
```

SETRC #imm



1=< imm =< 255

SETRC Rm



1 =< RM[11:0] =< 4095

(4) 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

11.2.13 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 （一部特権命令）

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 MOD,Rn	MOD→Rn	0000nnnn01010010	1	—
STC RE,Rn	RE→Rn	0000nnnn01110010	1	—
STC RS,Rn	RS→Rn	0000nnnn01100010	1	—
STC.L SR,@-Rn	Rn-4→Rn, SR→(Rn)	0100nnnn00000011	1	—
STC.L MOD,@-Rn	Rn-4→Rn、MOD→(Rn)	0100nnnn01010011	1	—
STC.L RE,@-Rn	Rn-4→Rn、RE→(Rn)	0100nnnn01110011	1	—
STC.L RS,@-Rn	Rn-4→Rn、RS→(Rn)	0100nnnn01100011	1	—

(1) 説明

コントロールレジスタ SR、MOD、RE、RS のデータをデスティネーションに格納します。

(2) 注意

STC SR,Rn と STC.L SR,@Rn-は SR レジスタの DSP ビットが 0 のとき特権命令です。DSP ビットが 1 のときには特権命令ではなく、すべてのビットを読み出すことができます。これ以外の 6 つの命令は、ユーザモードで実行することができます。

(3) 動作内容

```
STCSR(long n)       /*STC SR,Rn */
{
        R[n] = SR;
        PC += 2;
}

STCMOD(long n)      /* STC MOD,Rn */
{
        R[n] = MOD
        PC += 2;
}

STCRE(long n)       /* STC RE, Rn */
{
```

```
        R[n] = RE;
        PC += 2;
}

STCRS(long n)       /* STC RS,Rn */
{
        R[n] = RS;
        PC += 2;
}

STCMSR(long n)      /* STC.L SR,@-Rn */
{
        R[n] -= 4;
        Write_Long (R[n],SR);
        PC += 2;
}

STCMMOD(long n)     /* STC.L MOD,@-Rn */
{
        R[n] -= 4;
        Write_Long (R[n],MOD);
        PC += 2;
}

STCMRE(long n)      /* STC.L RE,@-Rn */
{
        R[n] -= 4;
        Write_Long (R[n],RE);
        PC += 2;
}

STCMRS(long n)      /* STC.L RS, @-Rn */
{
        R[n] -= 4;
        Write_Long (R[n],RS);
        PC += 2;
}
```

(4) 使用例

```
STC  SR,R0              ;実行前     R0=H'FFFFFFFF,SR=H'00000000
                        ;実行後     R0=H'00000000
STC.L  GBR,@-R15        ;実行前     R15=H'10000004,GBR=H'12345678
                        ;実行後     R15=H'10000000,(R15)=H'12345678
```

(5) 発生する可能性のある例外

- 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

11.2.14 STS STore from DSP System register システム制御命令

DSP 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	DSR,Rn	DSR→Rn	0000nnnn01101010	1	—
STS	A0,Rn	A0→Rn	0000nnnn01111010	1	—
STS	X0,Rn	X0→Rn	0000nnnn10001010	1	—
STS	X1,Rn	X1→Rn	0000nnnn10011010	1	—
STS	Y0,Rn	Y0→Rn	0000nnnn10101010	1	—
STS	Y1,Rn	Y1→Rn	0000nnnn10111010	1	—
STS.L	DSR,@-Rn	Rn-4→Rn, DSR→(Rn)	0100nnnn01100010	1	—
STS.L	A0,@-Rn	Rn-4→Rn, A0→(Rn)	0100nnnn01110010	1	—
STS.L	X0,@-Rn	Rn-4→Rn, X0→(Rn)	0100nnnn10000010	1	—
STS.L	X1,@-Rn	Rn-4→Rn, X1→(Rn)	0100nnnn10010010	1	—
STS.L	Y0,@-Rn	Rn-4→Rn, Y0→(Rn)	0100nnnn10100010	1	—
STS.L	Y1,@-Rn	Rn-4→Rn, Y1→(Rn)	0100nnnn10110010	1	—

(1) 説明

DSP システムレジスタ DSR、および DSP データレジスタ A0、X0、X1、Y0、Y1 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

```
STSDSR (long n)       /* STS DSR,Rn */
{
      R[n] = DSR;
      PC += 2;
}

STSA0 (long n)        /* STS A0,Rn */
{
      R[n] = A0;
      PC += 2;
}
```

```
STSX0 (long n)        /* STS X0,Rn */
{
      R[n] =  X0;
      PC += 2;
}

STSX1(long n)         /* STS X1,Rn */
{
      R[n] = X1;
      PC += 2;
}

STSY0(long n)         /* STS Y0,Rn */
{
      R[n] = Y0;
      PC += 2;
}

STSY1(long n)         /* STS Y1,Rn */
{
      R[n] = Y1;
      PC += 2;
}

STSMDSR(long n)       /* STS.L DSR,@-Rn */
{
      R[n] -= 4;
      Write_Long(R[n],DSR);
      PC += 2;
}

STSMA0(long n)        /* STS.L A0,@-Rn */
{
      R[n] -= 4;
      Write_Long(R[n],A0);
      PC += 2;
}
```

```
STSMX0(long n)        /* STS.L X0,@-Rn */
{
      R[n] -= 4;
      Write_Long(R[n],X0);
      PC += 2;
}

STSMX1(long n)        /* STS.L X1,@-Rn */
{
      R[n] -= 4;
      Write_Long(R[n],X1);
      PC += 2;
}

STSMY0(long n)        /* STS.L Y0,@-Rn */
{
      R[n] -= 4;
      Write_Long(R[n],Y0);
      PC += 2;
}

STSMY1(long n)        /* STS.L Y1,@-Rn */
{
      R[n] -= 4;
      Write_Long(R[n],Y1);
      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) 発生する可能性のある例外

- 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

11.3 DSP データ転送命令

本節では、ダブルデータ転送（MOVX.W、MOVY.W、MOVX.L、MOVY.L）、シングルデータ転送（MOVS.W、MOVS.L）の動作を説明します。

11.3.1 ダブルデータ転送（MOVX.W、MOVY.W、MOVX.L、MOVY.L）

この命令は XDB バス、YDB バスを使って X/Y メモリをアクセスします。X/Y メモリ以外のエリアはアクセスできません。メモリアクセスはワード単位のアクセスです。独立したバスを使用するため、命令フェッチ（LDB バスを使用）とはアクセス競合が発生しません。

ダブルデータ転送命令では、DSP データ演算命令を並行して記述することができます。ただし、同時並行して実行されるデータ演算命令が条件付命令であっても、ダブルデータ転送命令は条件に関係なく実行されます。

XDB バス、YDB バスでの転送を組み合わせて指定することができますが、一方の転送動作が不要の場合に転送機能を拡張することができます。1 つは、デスティネーションオペランドを拡張する 16 ビット転送命令 MOVX.W&NOPY または NOPX&MOVY.W です。もう 1 つは、32 ビット転送命令 MOVX.L&NOPY または NOPX&MOVY.L です。この形式では、アドレッシングに使用できるアドレスポインタの種類も通常のダブルデータ転送命令と比較して多くなっています。

ダブルデータ転送のロード、ストアの動作を**図 11.1** に示します。

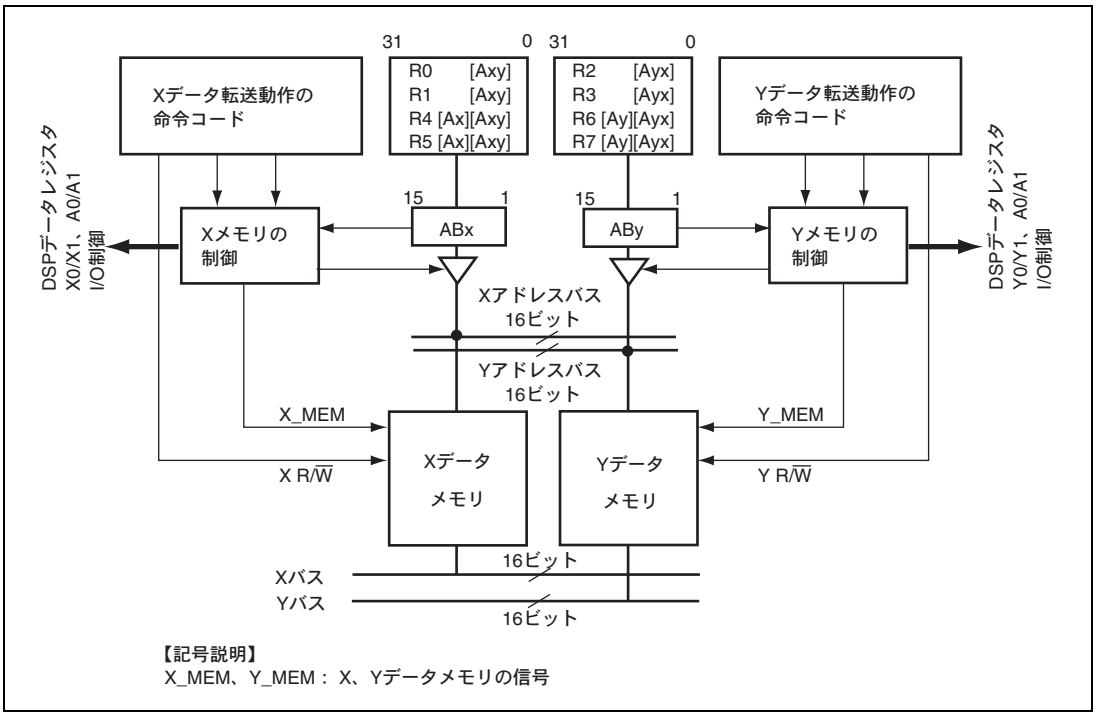


図 11.1 ダブルデータ転送のロード、ストアの動作

X/Y メモリデータ転送の動作を次に示します。

```
if( MOVX ) {
     X_MEM = 1; XAB = ABx;
          if( W/L is word access ) {                    /* MOVX.W */
               if( X R/W is load operation ) {
               Dxy[31:16] = XDB;                        /* load */
               Dxy[15:0] = 0x0000;
          } else {
               XDB = Dx[31:16];                         /* write */
          }
     } else {
          if( X R/W is load operation) {                /* MOVX.L */
               Dxy[31:16] = XDB;                        /* load */
               Dxy[15:0] = YDB;
          } else {
               XDB = Dax[31:16];
               YDB = Dax[15:0];                         /* write */
          }
     }
}
else if( MOVY ) {
     Y_MEM = 1, YAB = ABy;
     if( W/L is word access ) {                         /* MOVY.W */
          if( Y R/W is load operation ) {
               Dyx[31:16] = YDB;                        /* load */
               Dyx[15:0] = 0x0000;
          } else {
               YDB = Day[31:16];                        /* write */
          }
     } else {
          if( Y R/W is load operation) {                /* MOVY.L */
               Dyx[31:16] = XDB;                        /* load */
               Dyx[15:0] = YDB;
          } else {
               XDB = Day[31:16];                        /* write */
               YDB = Day[15:0];
          }
```

```
        }
    }
    else { X_MEM or Y_MEM = 0; XAB or YAB = Unknown; }       /* NOPX or NOPY */
```

11.3.2 シングルデータ転送（MOVS.W、MOVS.L）

シングルデータ転送はDSPレジスタにロード、ストアする命令で、システムレジスタのロード、ストア命令と似ています。メモリとDSPレジスタとの間のデータ転送をLDBバスを使って転送します。

シングルデータ転送はワード長、ロングワード長のデータが転送できます。

シングルデータ転送のロード、ストアの動作を図11.2に示します。

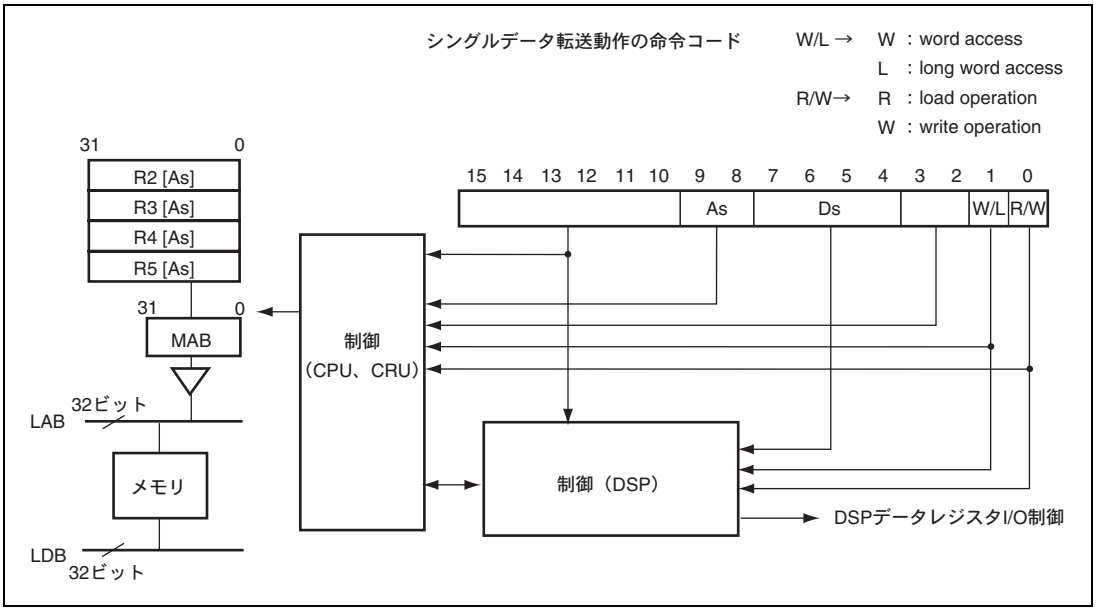


図 11.2 シングルデータ転送のロード、ストアの動作

シングルデータ転送のロード、ストア動作を次に示します。

```
if(MOVS) {
          LAB = MAB;
          if(W/L is word access) {                         /* MOVS.W*/
               if(R/W is load operation) {                 /* load */
                    if( Ds!=A0G && Ds!=A1G) {
                         Ds[31:16] = LDB[15:0];
                         Ds[15:0] = 0x0000;
                         if(DS==A0) A0G[7:0] = LDB[15];
                         if(DS==A1) A1G[7:0] = LDB[15];
                    } else {
                         Ds[7:0] = LDB[7:0];
                    }
          } else {       /* store */
                    if(Ds!=A0G && Ds!=A1G) {
                         LDB[15:0] = Ds[31:16];
                    } else {
                         LDB[15:0] = Ds[7:0] ( with 8bit sign extention )
                    }
               }
          } else {       /* MOVS.L */
               if(R/W is load operation) {                 /* load */
                    if(Ds != A0G && Ds !=A1G) {
                         Ds[31:0] = LDB[31:0];
                         if(DS == A0) A0G[7:0] = LDB[31];
                         if(DS == A1) A1G[7:0] = LDB[31];
                    } else {
                         Ds[7:0] = LDB[7:0];
                    }
          } else {       /* store */
                    if(Ds!=A0G && Ds!=A1G) {
                         LDB[31:0] = Ds[31:0];
                    } else {
                         LDB[31:0] = Ds[7:0] (with 24bit sign extention)
                    }
               }
          }
     }
```

11.3.3 MOVS MOVe Single data between memory and dsp register DSP データ転送命令

シングルデータ転送

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
MOVS.W @-As,Ds	As－2→As、(As)→MSW of Ds、0→LSW of Ds	111101AADDDD0000	1	−	−
MOVS.W @As,Ds	(As)→MSW of Ds、0→LSW of Ds	111101AADDDD0100	1	−	−
MOVS.W @As+,Ds	(As)→MSW of Ds、0→LSW of Ds、As+2→As	111101AADDDD1000	1	−	−
MOVS.W @As+Is,Ds	(As)→MSW of Ds、0→LSW of Ds、As+Is→As	111101AADDDD1100	1	−	−
MOVS.W Ds,@-As	As-2→As、MSW of Ds→(As)	111101AADDDD0001	1	−	−
MOVS.W Ds,@As	MSW of Ds→(As)	111101AADDDD0101	1	−	−
MOVS.W Ds,@As+	MSW of Ds→(As)、As＋2→As	111101AADDDD1001	1	−	−
MOVS.W Ds,@As+Is	MSW of Ds→(As)、As+Is→As	111101AADDDD1101	1	−	−
MOVS.L @-As,Ds	As-4→As、(As)→Ds	111101AADDDD0010	1	−	−
MOVS.L @As,Ds	(As)→Ds	111101AADDDD0110	1	−	−
MOVS.L @As+,Ds	(As)→Ds、As+4→As	111101AADDDD1010	1	−	−
MOVS.L @As+Is,Ds	(As)→Ds、As+Is→As	111101AADDDD1110	1	−	−
MOVS.L Ds,@-As	As-4→As、Ds→(As)	111101AADDDD0011	1	−	−
MOVS.L Ds,@As	Ds→(As)	111101AADDDD0111	1	−	−
MOVS.L Ds,@As+	Ds→(As)、As+4→As	111101AADDDD1011	1	−	−
MOVS.L Ds,@As+Is	Ds→(As)、As+Is→As	111101AADDDD1111	1	−	−

(1) 説明

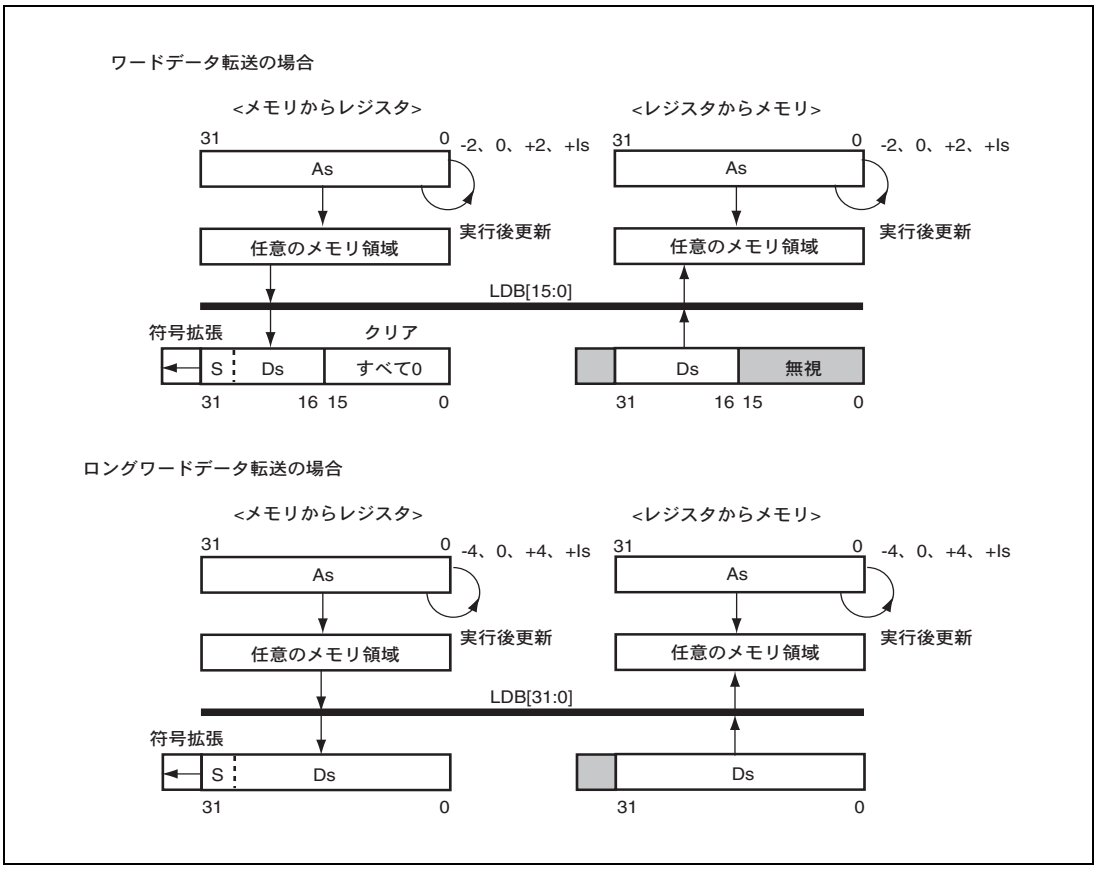
ソースオペランドのデータをデスティネーションオペランドへ転送します。メモリからレジスタへ、レジスタからメモリへ転送します。ワード長、ロングワード長を指定できます。ワード転送の場合、ソースオペランドがメモリでデスティネーションオペランドがレジスタのときは、ワードデータはレジスタの上位ワードにロードされ、下位ワードは 0 でクリアされます。ソースオペランドがレジスタでデスティネーションオペランドがメモリのときは、レジスタの上位ワードがワードデータとしてストアされます。ロングワード転送の場合はロングワー

ドデータが転送されます。レジスタがデスティネーションオペランドでガードビットがある場合は、符号が拡張されてガードビットに格納されます。

(2) 注意

ガードビットレジスタ、A0G、A1G の 1 つがストア処理のソースオペランドのときは、データは最下位 8 ビット（ビット 7〜0）に出力され、上位 24 ビット（ビット 31〜8）は符号拡張されます。

(3) 動作内容



(4) 使用例

```
MOVS.W      @R4＋,A0            ;実行前：  R4=H'00000400,(R4)=H'8765,
                                           A0=H'123456789A
                                実行後：  R4=H'00000402, A0=H'FF87650000
MOVS.L      A1,@-R3             ;実行前：  R3=H'00000800, A1=H'123456789A
                                実行後：  R3=H'000007FC,
                                           (H'000007FC)=H'3456789A
```

(5) 発生する可能性のある例外

- 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

11.3.4 MOVX MOVe between X memory and dsp register DSP データ転送命令

X メモリデータ転送

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
MOVX.W @Ax,Dx	(Ax)→Dx の MSW、 0→Dx の LSW	111100A*D*0*01**	1	−	−
MOVX.W @Ax+,Dx	(Ax)→Dx の MSW、 0→Dx の LSW、 Ax＋2→Ax	111100A*D*0*10**	1	−	−
MOVX.W @Ax+Ix,Dx	(Ax)→Dx の MSW、 0→Dx の LSW、 Ax＋Ix→Ax	111100A*D*0*11**	1	−	−
MOVX.W @Axy,Dxy	(Axy)→Dxy の MSW、 0→Dxy の LSW	111100AADD000100	1	−	−
MOVX.W @Axy+,Dxy	(Axy)→Dxy の MSW、 0→ Dxy の LSW、 Axy+2→Axy	111100AADD001000	1	−	−
MOVX.W @Axy+Ix,Dxy	(Axy)→Dxy の MSW、 0→ Dxy の LSW Axy+Ix→Axy	111100AADD001100	1	−	−
MOVX.L @Axy,Dxy	(Axy)→Dxy	111100AADD010100	1	−	−
MOVX.L @Axy+,Dxy	(Axy)→Dxy、 Axy+4→Axy	111100AADD011000	1	−	−
MOVX.L @Axy+Ix,Dxy	(Axy)→Dxy Axy＋Ix→Axy	111100AADD011100	1	−	−
MOVX.W Da,@Ax	Da の MSW→(Ax)	111100A*D*1*01**	1	−	−
MOVX.W Da,@Ax+	Da の MSW→(Ax)、 Ax＋2→Ax	111100A*D*1*10**	1	−	−
MOVX.W Da,@Ax+Ix	Da の MSW→(Ax)、 Ax＋Ix→Ax	111100A*D*1*11**	1	−	−
MOVX.W Dax,@Axy	Dax の MSW→(Axy)	111100AADD100100	1	−	−
MOVX.W Dax,@Axy+	Dax の MSW→(Axy)、 Axy+2→Axy	111100AADD101000	1	−	−
MOVX.W Dax,@Axy+Ix	Dax の MSW→(Axy)、 Axy＋Ix→Axy	111100AADD101100	1	−	−

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
MOVX.L Dax,@Axy	Dax→(Axy)	111100AADD110100	1	−	−
MOVX.L Dax,@Axy+	Dax→(Axy)、Axy+4→Axy	111100AADD111000	1	−	−
MOVX.L Dax,@Axy+Ix	Dax→(Axy)、Axy+Ix→Axy	111100ADDD111100	1	−	−

【注】 命令コードの「*」部分は、MOVY 命令指定領域です。

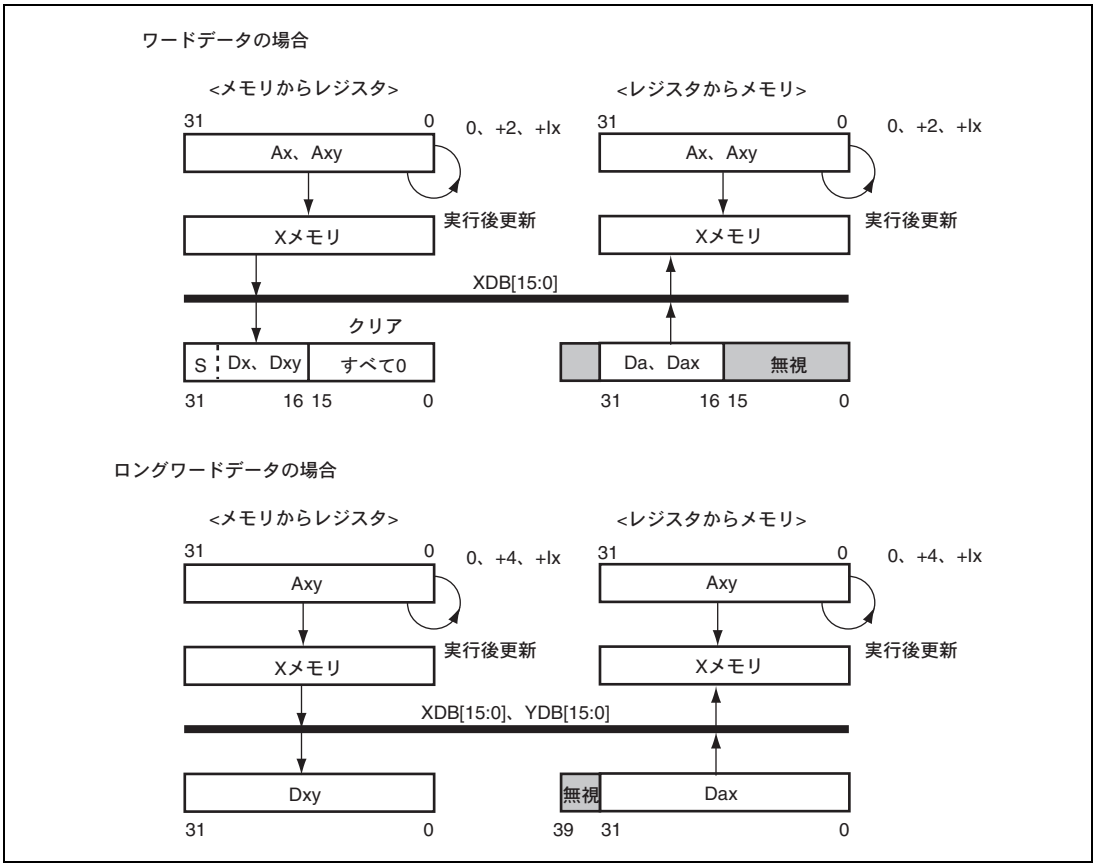
（1） 説明

ソースオペランドのデータをデスティネーションオペランドへ転送します。メモリからレジスタへ、レジスタからメモリへ転送します。X メモリとワード長、ロングワード長が指定できます。

ソースオペランドがメモリ、デスティネーションオペランドがレジスタで、ワード長のロードのときは、ワードデータはレジスタの上位ワードにロードされ、下位ワードは 0 でクリアされます。ソースオペランドがレジスタ、デスティネーションオペランドがメモリで、ワード長のストアのときは、レジスタの上位ワードがワードデータとしてストアされます。Y メモリとのロードストアを NOPY とすることにより、MOVX.W のソースオペランド、デスティネーションオペランドを拡張することができます。

ソースオペランドがメモリ、デスティネーションオペランドがレジスタで、ロングワード長のロードのときは、ロングワードデータはレジスタにロードされます。ソースオペランドがレジスタ、デスティネーションオペランドがメモリで、ロングワード長のストアのときは、レジスタがロングワードデータとしてストアされます。このとき、Y メモリとのロードストアは NOPY となります。

(2) 動作内容



(3) 使用例

```
MOVX.W @R4＋,X0          ;実行前：R4=H'A5007000,(R4)=H'5555, X0=H'12345678
                          実行後：R4=H'A5007002, X0=H'55550000
MOVX.L @R4＋,X0          ;実行前：R4=H'A5007000,(R4)=H'55555555, X0=H'12345678
                          実行後：R4=H'A5007004, X0=H'55555555
```

(4) 発生する可能性のある例外

- 一般不当命令例外
- データアドレスエラー

11.3.5 MOVY MOVe between Y memory and dsp register DSP データ転送命令

Y メモリデータ転送

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
MOVY.W @Ay,Dy	(Ay)→Dy の MSW、 0→Dy の LSW	111100*A*D*0**01	1	−	−
MOVY.W @Ay+,Dy	(Ay)→Dy の MSW、 0→Dy の LSW、 Ay＋2→Ay	111100*A*D*0**10	1	−	−
MOVY.W @Ay+Iy,Dy	(Ay)→Dy の MSW、 0→Dy の LSW、 Ay＋Iy→Ay	111100*A*D*0**11	1	−	−
MOVY.W @Ayx,Dyx	(Ayx)→Dyx の MSW、 0→Dyx の LSW	111100AADD000001	1	−	−
MOVY.W @Ayx+,Dyx	(Ayx)→Dyx の MSW、 0→Dyx の LSW、 Ayx+2→Ayx	111100AADD000010	1	−	−
MOVY.W @Ayx+Iy,Dyx	(Ayx)→Dyx の MSW、 0→Dyx の LSW、 Ayx+Ix→Ayx	111100AADD000011	1	−	−
MOVY.L @Ayx,Dyx	(Ayx)→Dyx	111100AADD100001	1	−	−
MOVY.L @Ayx+,Dyx	(Ayx)→Dyx、 Ayx+4→Ayx	111100AADD100010	1	−	−
MOVY.L @Ayx+Iy,Dyx	(Ayx)→Dyx、 Ayx＋Ix→Ayx	111100AADD100011	1	−	−
MOVY.W Da,@Ay	Da の MSW→(Ay)	111100*A*D*1**01	1	−	−
MOVY.W Da,@Ay+	Da の MSW→(Ay)、 Ay＋2→Ay	111100*A*D*1**10	1	−	−
MOVY.W Da,@Ay+Iy	Da の MSW→(Ay)、 Ay＋Iy→Ay	111100*A*D*1**11	1	−	−
MOVY.W Day,@Ayx	Day の MSW→(Ayx)	111100AADD010001	1	−	−
MOVY.W Day,@Ayx+	Day の MSW→(Ayx)、 Ayx+2→Ayx	111100AADD010010	1	−	−

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
MOVY.W Day,@Ayx+ly	Day の MSW→(Ayx)、Ayx＋ly→Ayx	111100AADD010011	1	−	−
MOVY.L Day,@Ayx	Day→(Ayx)	111100AADD110001	1	−	−
MOVY.L Day,@Ayx+	Day→(Ayx)、Ayx+4→Ayx	111100AADD110010	1	−	−
MOVY.L Day,@Ayx+ly	Day→(Ayx)、Ayx+ly→Ayx	111100ADDD110011	1	−	−

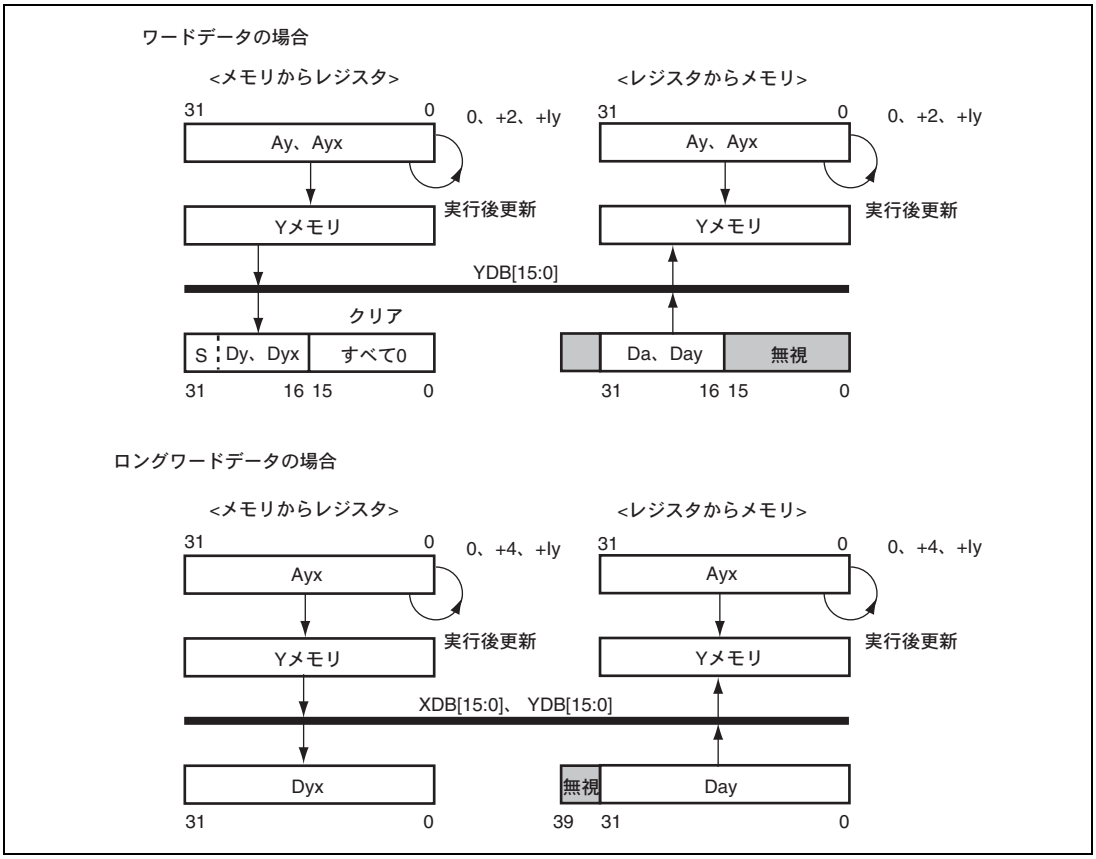
【注】 命令コードの「*」部分は、MOVX 命令の指定領域です。

(1) 説明

ソースオペランドがメモリ、デスティネーションオペランドがレジスタで、ワード長のロードのときは、ワードデータはレジスタの上位ワードにロードされ、下位ワードは 0 でクリアされます。ソースオペランドがレジスタ、デスティネーションオペランドがメモリで、ワード長のストアのときは、レジスタの上位ワードがワードデータとしてストアされます。X メモリとのロードストアを NOPX とすることにより、MOVY.W のソースオペランド、デスティネーションオペランドを拡張することができます。

ソースオペランドがメモリ、デスティネーションオペランドがレジスタで、ロングワード長のロードのときは、ロングワードデータはレジスタにロードされます。ソースオペランドがレジスタ、デスティネーションオペランドがメモリで、ロングワード長のストアのときは、レジスタがロングワードデータとしてストアされます。このとき、X メモリとのロードストアは NOPX となります。

(2) 動作内容



(3) 使用例

```
MOVY.W A0, @R6＋R9          ;実行前：R6=H'A5017000, R9=H'00000004, A0=H'123456789A
                             実行後：R6=H'A5017002, (H'A5017000)=H'3456
MOVY.L A0, @R6＋R9          ;実行前：R6=H'A5017000, R9=H'00000004, A0=H'123456789A
                             実行後：R6=H'A5017004, (H'A5017000)=H'3456789A
```

(4) 発生する可能性のある例外

- 一般不当命令例外
- データアドレスエラー

11.3.6 NOPX No access OPeration for X memory DSP データ転送命令

X メモリ無操作

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
NOPX	No Operation	1111000*0*0*00**	1	−	−

【注】 命令コードの「*」部分は、MOVY 命令の指定領域です。

(1) 説明

X メモリのアクセスについて無操作です。

このニーモニックは省略可能です。

11.3.7 NOPY No access OPeration for Y memory DSP データ転送命令

Y メモリ無操作

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
NOPY	No Operation	111100*0*0*0**00	1	−	−

【注】 命令コードの「*」部分は、MOVX 命令の指定領域です。

(1) 説明

Y メモリのアクセスについて無操作です。

このニーモニックは省略可能です。

11.4 DSP 演算命令

DSP 演算命令の 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。

(1) DSP レジスタ

DSP レジスタの名称は下記の DSP_Register_Set という名前のユニオンを基に定義します。このユニオンには、レジスタへのアクセス方法に応じて 5 種類の変数もしくは構造体・ユニオンが定義されています。

- 32ビットのロングワードデータ格納用レジスタの定義 → ①
- 16ビットのワードデータ格納用レジスタの定義 → ②
- 32ビット長レジスタのMSBをアクセスする領域と、8ビット長レジスタ（ガードビット）のMSBをアクセスする領域の定義 → ③
- ガードビットレジスタを格納するための領域の定義 → ④
- DSRレジスタをアクセスするための領域の定義 → ⑤

各々のロングワードは 11 個の DSP レジスタ（A0、A1、M0、M1、X0、X1、Y0、Y1、A0G、A1G、DSR）に対応します。

```
/* Union DSP_Register_Set の定義 */

union {
     unsigned long int uli[11];
     unsigned short int usi[22];
     struct {
          struct {
               unsigned short int usi[2];
          } ee[11];
     } dd;
     struct {
          struct {
               union {
                    unsigned long int uli;  …… ①
                    unsigned short int usi[2];……②
                    struct {
                         unsigned msb: 1;
                         unsigned : 23;
                         unsigned g_msb: 1;
                         unsigned : 7;
```

```
                } bb; ····· ③
                struct {
                    unsigned : 24;
                    unsigned lsb8: 8;
                } cc; ····· ④
            } mm;
        } a0, a1, m0, m1, x0, x1, y0, y1, a0g, a1g;
        union {
            unsigned long int uli;
            struct {
                unsigned Reserve: 16;
                unsigned agt: 1;
                unsigned az: 1;
                unsigned an: 1;
                unsigned av: 1;
                unsigned ts: 3;
                unsigned tc: 1;
                unsigned gt: 1;
                unsigned z: 1;
                unsigned n: 1;
                unsigned v: 1;
                unsigned cs: 3;
                unsigned dc: 1;
            } aa;
        } dsr; ····· ⑤
    } name;
    struct {
        unsigned short int a[2][2];
        unsigned short int m[2][2];
        unsigned short int x[2][2];
        unsigned short int y[2][2];
        unsigned short int ag[2][2];
        unsigned short int dsr[2];
    } word;
} DSP_Register_Set;
```

上記のユニオン DSP_Register_Set を用いて以下のように DSP レジスタの名称を定義します。

```
/* DSP Register 名の定義 */

#define A0          DSP_Register_Set.name.a0.mm.uli
#define A0_HW       DSP_Register_Set.name.a0.mm.uli[0]
#define A0_LW       DSP_Register_Set.name.a0.mm.uli[1]
#define A0_MSB      DSP_Register_Set.name.a0.mm.bb.msb

#define A1          DSP_Register_Set.name.a1.mm.uli
#define A1_HW       DSP_Register_Set.name.a1.mm.uli[0]
#define A1_LW       DSP_Register_Set.name.a1.mm.uli[1]
#define A1_MSB      DSP_Register_Set.name.a1.mm.bb.msb

#define M0          DSP_Register_Set.name.m0.mm.uli
#define M0_HW       DSP_Register_Set.name.m0.mm.uli[0]
#define M0_LW       DSP_Register_Set.name.m0.mm.uli[1]
#define M0_MSB      DSP_Register_Set.name.m0.mm.bb.msb

#define M1          DSP_Register_Set.name.m1.mm.uli
#define M1_HW       DSP_Register_Set.name.m1.mm.uli[0]
#define M1_LW       DSP_Register_Set.name.m1.mm.uli[1]
#define M1_MSB      DSP_Register_Set.name.m1.mm.bb.msb

#define X0          DSP_Register_Set.name.x0.mm.uli
#define X0_HW       DSP_Register_Set.name.x0.mm.uli[0]
#define X0_LW       DSP_Register_Set.name.x0.mm.uli[1]
#define X0_MSB      DSP_Register_Set.name.x0.mm.bb.msb

#define X1          DSP_Register_Set.name.x1.mm.uli
#define X1_HW       DSP_Register_Set.name.x1.mm.uli[0]
#define X1_LW       DSP_Register_Set.name.x1.mm.uli[1]
#define X1_MSB      DSP_Register_Set.name.x1.mm.bb.msb

#define Y0          DSP_Register_Set.name.y0.mm.uli
#define Y0_HW       DSP_Register_Set.name.y0.mm.uli[0]
#define Y0_LW       DSP_Register_Set.name.y0.mm.uli[1]
#define Y0_MSB      DSP_Register_Set.name.y0.mm.bb.msb

#define Y1          DSP_Register_Set.name.y1.mm.uli
```

```
#define Y1_HW        DSP_Register_Set.name.y1.mm.uli[0]
#define Y1_LW        DSP_Register_Set.name.y1.mm.uli[1]
#define Y1_MSB       DSP_Register_Set.name.y1.mm.bb.msb

#define A0G          DSP_Register_Set.name.a0g.mm.uli
#define A0G_HW       DSP_Register_Set.name.a0g.mm.uli[0]
#define A0G_LW       DSP_Register_Set.name.a0g.mm.uli[1]
#define A0G_LSB8     DSP_Register_Set.name.a0g.mm.cc.lsb8
#define A0G_MSB      DSP_Register_Set.name.a0g.mm.bb.g_msb

#define A1G          DSP_Register_Set.name.a1g.mm.uli
#define A1G_HW       DSP_Register_Set.name.a1g.mm.uli[0]
#define A1G_LW       DSP_Register_Set.name.a1g.mm.uli[1]
#define A1G_LSB8     DSP_Register_Set.name.a1g.mm.cc.lsb8
#define A1G_MSB      DSP_Register_Set.name.a1g.mm.bb.g_msb

#define DSR          DSP_Register_Set.name.dsr.uli
#define DSPAGTBIT    DSP_Register_Set.name.dsr.aa.agt
#define DSPAZBIT     DSP_Register_Set.name.dsr.aa.az
#define DSPANBIT     DSP_Register_Set.name.dsr.aa.an
#define DSPAVBIT     DSP_Register_Set.name.dsr.aa.av
#define DSPTSBITS    DSP_Register_Set.name.dsr.aa.ts
#define DSPTCBIT     DSP_Register_Set.name.dsr.aa.tc
#define DSPGTBIT     DSP_Register_Set.name.dsr.aa.gt
#define DSPZBIT      DSP_Register_Set.name.dsr.aa.z
#define DSPNBIT      DSP_Register_Set.name.dsr.aa.n
#define DSPVBIT      DSP_Register_Set.name.dsr.aa.v
#define DSPCSBITS    DSP_Register_Set.name.dsr.aa.cs
#define DSPDCBIT     DSP_Register_Set.name.dsr.aa.dc
#define DSP_REG      DSP_Register_Set.uli
#define DSP_REG_WD   DSP_Register_Set.uli
```

(2) ALU の入出力と演算結果を表す変数

ALU の入出力は下記の DSP_ALU_Set という名前のユニオンを基に定義します。このユニオンは 6 個のロングワードで構成されます。このうち 3 つのロングワードは 2 つの入力と 1 つの出力（src1、src2、dst）に対応します。残りの 3 つのロングワードはこれら 2 つの入力と 1 つの出力のガードビット用（src1g、src2g、dstg）です。

```
/* Union DSP_ALU_Set の定義 */

union {
     unsigned long int     uli[6];
     unsigned short int    usi[12];
     struct {
          struct {
               unsigned msb:              1;
               unsigned:  31;
          } src1, src2, dst;
          struct {
               union {
                    unsigned long int   uli;
                    struct {
                         unsigned:       24;
                         unsigned bit7:  1;
                         unsigned:       7;
                    } a;
                    struct {
                         unsigned:       24;
                         unsigned lsb8:  8;
                    } b;
               } u;
          } src1g, src2g, dstg;
     } n;
} DSP_ALU_Set;
```

上記のユニオン DSP_ALU_Set を用いて以下のように ALU 入出力の名称を定義します。

```
/* DSP 演算命令における ALU の入出力の定義 */

#define DSP_ALU_SRC1      DSP_ALU_Set.uli[0]
#define DSP_ALU_SRC2      DSP_ALU_Set.uli[1]
#define DSP_ALU_DST       DSP_ALU_Set.uli[2]

#define DSP_ALU_SRC1G     DSP_ALU_Set.uli[3]
#define DSP_ALU_SRC2G     DSP_ALU_Set.uli[4]
#define DSP_ALU_DSTG      DSP_ALU_Set.uli[5]

#define DSP_ALU_SRC1_HW   DSP_ALU_Set.usi[0]
#define DSP_ALU_SRC2_HW   DSP_ALU_Set.usi[2]
#define DSP_ALU_DST_HW    DSP_ALU_Set.usi[4]

#define DSP_ALU_SRC1_MSB  DSP_ALU_Set.n.src1.msb
#define DSP_ALU_SRC2_MSB  DSP_ALU_Set.n.src2.msb
#define DSP_ALU_DST_MSB   DSP_ALU_Set.n.dst.msb

#define DSP_ALU_SRC1G_BIT7   DSP_ALU_Set.n.src1g.u.a.bit7
#define DSP_ALU_SRC2G_BIT7   DSP_ALU_Set.n.src2g.u.a.bit7
#define DSP_ALU_DSTG_BIT7    DSP_ALU_Set.n.dstg.u.a.bit7

#define DSP_ALU_SRC1G_LSB8   DSP_ALU_Set.n.src1g.u.b.lsb8
#define DSP_ALU_SRC2G_LSB8   DSP_ALU_Set.n.src2g.u.b.lsb8
#define DSP_ALU_DSTG_LSB8    DSP_ALU_Set.n.dstg.u.b.lsb8
```

さらに演算結果の状態が「オーバフロー」「オーバレンジ」を表す変数を、上記までの定義を使って以下のように定義します。これらの変数は、各命令の動作内容の説明の中で DSR レジスタ内の DC ビットを計算するのに使われます。

1. 加算時のオーバレンジ定義

```
#define PLUS_OP_G_OV ((~DSP_ALU_SRC1G_BIT7 && ~DSP_ALU_SRC2G_BIT7 &&
                      DSP_ALU_DSTG_BIT7) || (DSP_ALU_SRC1G_BIT7 &&
                      DSP_ALU_SRC2G_BIT7 && ~DSP_ALU_DSTG_BIT7))
```

2. 減算時のオーバレンジ定義

```
#define MINUS_OP_G_OV ((~DSP_ALU_SRC1G_BIT7 && DSP_ALU_SRC2G_BIT7 &&
                      DSP_ALU_DSTG_BIT7) || (DSP_ALU_SRC1G_BIT7 &&
                      ~DSP_ALU_SRC2G_BIT7 && ~DSP_ALU_DSTG_BIT7))
```

3. 正の方向へオーバフローしない条件の定義

```
#define POS_NOT_OV ((DSP_ALU_DSTG_LSB8==0x00) && (DSP_ALU_DST_MSB==0x0))
```

4. 負の方向へオーバフローしない条件の定義

```
#define NEG_NOT_OV ((DSP_ALU_DSTG_LSB8==0xFF) && (DSP_ALU_DST_MSB==0x1))
```

(3) 乗算器の入出力

乗算器の入出力は下記の DSP_MUL_Set という名前のユニオンを基に定義します。このユニオンは 4 個のロングワードで構成されます。2 つの入力にはロングワードが 1 つずつ割り当てられていますが、両者とも上位 16 ビット（usi[0]、usi[2]）のみが使われます。出力にはガードビット用を含めた 2 つのロングワード（dst、dstg）が対応します。

```
/* Union DSP_MUL_Set の定義 */

union {
     unsigned long int  uli[4];
     struct {
          unsigned short int  usi[4];
          struct {
               unsigned msb:  1;
               unsigned:      31;
          } dst;
          struct {
               unsigned:      24;
               unsigned lsb8: 8;
          } dstg;
     } aa;
} DSP_MUL_Set;
```

上記のユニオン DSP_MUL_Set を用いて以下のように乗算器入出力の名称を定義します。

```
/* DSP 演算命令における乗算器入出力の定義 */

#define DSP_MLT_SRC1       DSP_MUL_Set.aa.usi[0]
#define DSP_MLT_SRC2       DSP_MUL_Set.aa.usi[2]
#define DSP_MLT_DST        DSP_MUL_Set.uli[2]
#define DSP_MLT_DST_MSB    DSP_MUL_Set.aa.dst.msb
#define DSP_MLT_DSTG       DSP_MUL_Set.uli[3]
#define DSP_MLT_DSTG_LSB8 DSP_MUL_Set.aa.dstg.lsb8
```

(4) その他の命令動作内容説明で使われる変数など

DCT、DCF という条件が指定できる DSP 演算命令の動作内容説明のときには以下の変数を使用しています。

```
#define DSP_UNCONDITIONAL_UPDATE  (!EX_DCT && !EX_DCF)
#define DSP_CONDITION_MATCH       ((EX_DCT && DSPDCBIT) ||(EX_DCF && !DSPDCBIT))
#define DSP_CONDITION_NOT_MATCH   ((EX_DCT && !DSPDCBIT)||(EX_DCF && DSPDCBIT))
```

上記の定義において EX_DCT と EX_DCF はそれぞれ命令に DCT、DCF という条件が指定されている場合に真となる変数です。また、DSPDCBIT は「（1）DSP レジスタ」を参照のこと。

DSP の算術演算は SR レジスタの飽和ビットが 1 のとき、飽和処理を行います。この飽和ビットを動作内容説明のときには SBIT と呼びます。

(5) 演算命令の命令コード

DSP の命令コードを以後の説明で使えるよう、一部を下記の「DSP_Instruction_code」を基に定義します。第 1 オペランドが Sx 側か Sy 側かの指定と、ソースおよびデスティネーションレジスタの指定、並列演算命令時のソースおよびデスティネーションレジスタの指定を定義します。

```
/* Union DSP_Instruction_code の定義 */
    union{
        unsigned short int usi;
        union{
            unsigned short int usi;
            struct {
                unsigned: 4;
                unsigned se: 2;
                unsigned sf: 2;
                unsigned sx: 2;
                unsigned sy: 2;
                unsigned dg: 2;
                unsigned du: 2;
            } para
            struct {
                unsigned: 2;
                unsigned xy: 1;
                unsigned: 3;
                unsigned dctf: 2;
                unsigned sx: 2;
                unsigned sy: 2;
                unsigned dz: 4;
```

```
            } sxy
            struct {
                unsigned: 16;
            } lw
        } dspcode
    } DSP_Instruction_code
```

上記のユニオン「DSP_Instruction_code」を用いて、以下の用に使用レジスタ、および条件付き実行、第 1 オペランドが X 側か Y 側かを定義します。

```
/* XY の判定。EX2_DSP_BIT13->0：X 側、1：Y 側 */
#define EX2_DSP_BIT13   DSP_Instruction_code.dspcode.sxy.xy
                                                /* 命令コードの[13]に相当 */

/* ソースレジスタの判定。
   Sx = X0,X1.A0,A1, Sy = Y0,Y1,M0,M1, Se = X0,X1,Y0,A1, Sf = Y0,Y1,X0,A1 */

#define EX2_SX          DSP_Instruction_code.dspcode.sxy.sx
                                                /* 命令コードの[7:6]に相当 */
#define EX2_SY          DSP_Instruction_code.dspcode.sxy.sy
                                                /* 命令コードの[5:4]に相当 */
#define EX2_SE          DSP_Instruction_code.dspcode.para.se
                                                /* 命令コードの[11:10]に相当 */
#define EX2_SF          DSP_Instruction_code.dspcode.para.sf
                                                /* 命令コードの[9:8]に相当 */
#define EX2_LW          DSP_Instruction_code.dspcode.lw
                                                /* 命令コードの[15: 0]に相当 */
/* デスティネーションレジスタの判定。
    Dz = A1,A0,X0,X1,Y0,Y1,M0,M1, Dg = M0,M1,A0,A1, Du = X0,Y0,A0,A1 */

#define EX2_DZ          DSP_Instruction_code.dspcode.sxy.dz
                                                /* 命令コードの[3:0]に相当 */
#define EX2_DG          DSP_Instruction_code.dspcode.para.dg
                                                /* 命令コードの[3:2]に相当 */
#define EX2_DU          DSP_Instruction_code.dspcode.para.du
                                                /* 命令コードの[1:0]に相当 */
```

上記までの指定を用いて、デスティネーションレジスタを示す変数を別に定義します。この変数は、各命令の説明のときに、レジスタを指定するために用いられます。

```
/* Destination_Register_Number.c */
{
     unsigned short int ex2_dz_no;
     switch ( EX2_DZ ) {
          case 0x7: ex2_dz_no = 0;              /* A0 */
          case 0x5: ex2_dz_no = 1;              /* A1 */
          case 0xC: ex2_dz_no = 2;              /* M0 */
          case 0xE: ex2_dz_no = 3;              /* M1 */
          case 0x8: ex2_dz_no = 4;              /* X0 */
          case 0x9: ex2_dz_no = 5;              /* X1 */
          case 0xA: ex2_dz_no = 6;              /* Y0 */
          case 0xB: ex2_dz_no = 7;              /* Y1 */
     }
}
```

(6) DSR レジスタ

DSR レジスタ内の DC ビットは、DSP 演算命令の演算結果と状態選択ビット（CS）の指定に従って更新されます。DSR レジスタ内には、DC ビットのほかに、4 つの状態を示すフラグ（V、N、Z、GT）があります。また、フラグの累積状態を示すフラグ（AV、AN、AZ、AGT）があります。SR レジスタの T ビットは、DSR レジスタの TS ビットの指定に従って更新されます。オーバフロー防止機能が有効のとき（S=1）、CS ビットでオーバフローモードを選択する場合、DC ビットは 0 でクリアされますが、TS ビットでオーバフローモードを選択する場合、オーバフロー検出結果が T ビットに反映されます。

各ビットの動作内容については下記のようになります。後述の DSP 演算命令ごとの動作説明では、下記の動作内容がサブルーチンモジュールとして使われています。以下の説明において、CPUSRTBIT は、ステータスレジスタ（SR）の S ビットを表すものと定義します。

- **動作内容1：固定小数点ボローDCビット**

```
/*  fixed_pt_dc_always_borrow.c*/
{
     unsigned short int  borrow_bit, negative_bit, zero_bit, overflow_bit,
                         overflow_avbit, overrange_bit;

     DSPDCBIT = borrow_bit;
     DSPGTBIT = ~((negative_bit ^ overrange_bit) | zero_bit);
     DSPZBIT = zero_bit;
     DSPNBIT = negative_bit;
     DSPVBIT = overflow_bit;
     if(DSPTSBITS != 0x0) {
```

```
        switch (DSPTSBITS) {
            case 0x0: if(DSPTCBIT == 1) CPUSRTBIT = borrow_bit;
                      break;
            case 0x1: DSPANBIT = DSPANBIT | nagative_bit;
                      if(DSPTCBIT == 1) CPUSRTBIT = negative_bit;
                      break;
            case 0x2: DSPAZBIT = DSPAZBIT | zero_bit;
                      if(DSPTCBIT == 1) CPUSRTBIT = zero_bit;
                      break;
            case 0x3: DSPAVBIT = DSPAVBIT | overflow_avbit;
                      if(DSPTCBIT == 1) CPUSRTBIT = overflow_bit;
                      break;
            case 0x4: DSPAGTBIT = DSPAGTBIT | DSPGTBIT;
                      if(DSPTCBIT == 1) CPUSRTBIT = DSPGTBIT;
                      break;
            case 0x5: if(DSPTCBIT == 1 ) CPUSRTBIT = ~(negative_bit ^ overflow_bit);
                      break;
        }
    }
}
```

- **動作内容2：固定小数点キャリDCビット**

```
/*  fixed_pt_dc_always_carry.c*/
{
    unsigned short int  borrow_bit, negative_bit, zero_bit, overflow_bit,
                        overflow_avbit, overrange_bit;

    DSPDCBIT = carry_bit;
    DSPGTBIT = ~((negative_bit ^ overrange_bit) | zero_bit);
    DSPZBIT = zero_bit;
    DSPNBIT = negative_bit;
    DSPVBIT = overflow_bit;
    if(DSPTSBITS != 0x0) {
        switch (DSPTSBITS) {
            case 0x0: if(DSPTCBIT == 1) CPUSRTBIT = carry_bit;
                      break;
            case 0x1: DSPANBIT = DSPANBIT | nagative_bit;
                      if(DSPTCBIT == 1) CPUSRTBIT = negative_bit;
```

```
                    break;
            case 0x2: DSPAZBIT = DSPAZBIT | zero_bit;
                    if(DSPTCBIT == 1) CPUSRTBIT = zero_bit;
                    break;
            case 0x3: DSPAVBIT = DSPAVBIT | overflow_bit;
                    if(DSPTCBIT == 1) CPUSRTBIT = overflow_bit;
                    break;
            case 0x4: DSPAGTBIT = DSPAGTBIT | DSPGTBIT;
                    if(DSPTCBIT == 1) CPUSRTBIT = DSPGTBIT;
                    break;
            case 0x5: if(DSPTCBIT == 1) CPUSRTBIT = ~(negative_bit^overflow_bit);
                    break;
        }
    }
}
```

- **動作内容3：固定小数点負値DCビット**

```
/* fixed_pt_minus_dc.c */
{
    unsigned short int borrow_bit, negative_bit, zero_bit, overflow_bit,
                       overflow_avbit, overrange_bit;
    switch (DSPCSBITS) {
            case 0x0: DSPDCBIT = borrow_bit;
                      break;
            case 0x1: DSPDCBIT = negative_bit;
                      break;
            case 0x2: DSPDCBIT = zero_bit;
                      break;
            case 0x3: DSPDCBIT = overflow_bit;
                      break;
            case 0x4: DSPDCBIT = ~((negative_bit ^ overrange_bit) | zero_bit);
                      break;
            case 0x5: DSPDCBIT = ~(negative_bit ^ overrange_bit);
                      break;
    }
DSPGTBIT = ~((negative_bit ^ overrange_bit) | zero_bit);
DSPZBIT = zero_bit;
DSPNBIT = negative_bit;
```

```
DSPVBIT = overflow_bit;
if(DSPTSBITS != 0x0) {
          switch ( DSPTSBITS ) {
              case 0x0: if(DSPTCBIT == 1) CPUSRTBIT = borrow_bit;
                        break;
              case 0x1: DSPANBIT = DSPANBIT | nagative_bit;
                        if(DSPTCBIT == 1) CPUSRTBIT = negative_bit;
                        break;
              case 0x2: DSPAZBIT = DSPAZBIT | zero_bit;
                        if(DSPTCBIT == 1) CPUSRTBIT = zero_bit;
                        break;
              case 0x3: DSPAVBIT = DSPAVBIT | overflow_avbit;
                        if(DSPTCBIT == 1) CPUSRTBIT = overflow_avbit;
                        break;
              case 0x4: DSPAGTBIT = DSPAGTBIT | DSPGTBIT;
                        if(DSPTCBIT == 1) CPUSRTBIT = DSPGTBIT;
                        break;
              case 0x5: if(DSPTCBIT == 1) CPUSRTBIT = ~(negative_bit^overrange_bit);
                        break;
              }
          }
}
```

- **動作内容4：固定小数点オーバフロー・オーバレンジ防止機能（飽和演算）**

```
/* fixed_pt_overflow_protection.c */
{
     unsigned short int       overflow_bit, overflow_avbit, overrange_bit;

     overflow_avbit = overflow_bit  | overrange_bit;
     if(SBIT && (overflow_bit || overrange_bit))) {
          if(DSP_ALU_DSTG_BIT7 == 0){
               if(DSP_ALU_DSTG_LSB8 != 0x0) || (DSP_ALU_DST_MSB != 0x0)) {
                    DSP_ALU_DSTG != 0x0;
                    DSP_ALU_DST = 0x7FFFFFFF;
               }
     } else {
          if((DSP_ALU_DSTG_LSB8 != 0xFF) || (DSP_ALU_DST_MSB != 0x1)) {
               DSP_ALU_DSTG = 0xFF;
```

```
                DSP_ALU_DST = 0x80000000;
            }
        }
        overflow_bit = 0;
        overrange_bit = 0;
        }
}
```

- **動作内容5：固定小数点正値DCビット**

```
/* fixed_pt_plus_dc_bit.c */
{
        unsigned short int       overflow_bit, overflow_avbit, overrange_bit;
        switch (DSPCSBITS) {
              case 0x0: DSPDCBIT = carry_bit;
                        break;
              case 0x1: DSPDCBIT = negative_bit;
                        break;
              case 0x2: DSPDCBIT = zero_bit;
                        break;
              case 0x3: DSPDCBIT = overflow_bit;
                        break;
              case 0x4: DSPDCBIT = ~((negative_bit ^ overrange_bit) | zero_bit);
                        break;
              case 0x5: DSPDCBIT = ~(negative_bit ^ overrange_bit);
                        break;
        }
        DSPGTBIT = ~((negative_bit ^ overrange_bit) | zero_bit);
        DSPZBIT = zero_bit;
        DSPNBIT = negative_bit;
        DSPVBIT = overflow_bit;
        if(DSPTSBITS != 0x0) {
             switch (DSPTSBITS) {
                 case 0x0: if(DSPTCBIT == 1) CPUSRTBIT = carry_bit;
                           break;
                 case 0x1: DSPANBIT = DSPANBIT | nagative_bit;
                           if(DSPTCBIT == 1) CPUSRTBIT = negative_bit;
                           break;
                 case 0x2: DSPAZBIT = DSPAZBIT | zero_bit;
```

```
                    if(DSPTCBIT == 1) CPUSRTBIT = zero_bit;
                    break;
               case 0x3: DSPAVBIT = DSPAVBIT | overflow_avbit;
                    if(DSPTCBIT == 1) CPUSRTBIT = overflow_avbit;
                    break;
               case 0x4: DSPAGTBIT = DSPAGTBIT | DSPGTBIT;
                    if(DSPTCBIT == 1) CPUSRTBIT = DSPGTBIT;
                    break;
               case 0x5: if(DSPTCBIT == 1) CPUSRTBIT = ~(negative_bit^overrange_bit);
                    break;
          }
     }
}
```

- **動作内容6：固定小数点演算無条件DCビット更新**

```
/* fixed_pt_unconditional_update.c */
{
     unsigned short int   negative_bit, zero_bit, ex2_dz_no;

#include "Destination_Register_Number.c"

     DSP_REG[ex2_dz_no] = DSP_ALU_DST;
     if(ex2_dz_no == 0) {
          A0G = DSP_ALU_DSTG & MASK000000FF;
          if(DSP_ALU_DSTG_BIT7) A0G = A0G | MASKFFFFFF00;
     }
     else if(ex2_dz_no == 1) {
          A1G = DSP_ALU_DSTG & MASK000000FF;
          if(DSP_ALU_DSTG_BIT7) A1G = A1G | MASKFFFFFF00;
     }
     negative_bit = DSP_ALU_DSTG_BIT7;
     zero_bit = (DSP_ALU_DST == 0) & (DSP_ALU_DSTG_LSB8 == 0);
}
```

- **動作内容7：整数負値DCビット**

```
/* integer_minus_dc_bit.c */
#include "fixed_pt_minus_dc_bit.c"
```

- **動作内容8：整数オーバフロー・オーバレンジ防止機能（飽和演算）**

```
/* integer_overflow_protection.c */
#include "fixed_pt_overflow_protection.c"
```

- **動作内容9：整数正値DCビット**

```
/* integer_plus_dc_bit.c */
#include "fixed_pt_plus_dc_bit.c"
```

- **動作内容10：整数無条件DCビット更新**

```
/* integer_unconditional_update.c */
{
     unsigned short int   negative_bit, zero_bit, ex2_dz_no;

#include "Destination_Register_Number.c"

     DSP_REG[ex2_dz_no*2] = DSP_ALU_DST_HW;
     DSP_REG[ex2_dz_no*2+1] = 0x0;
     if(ex2_dz_no == 0) {
          A0G = DSP_ALU_DSTG & MASK000000FF;
          if(DSP_ALU_DSTG_BIT7) A0G = A0G | MASKFFFFFF00;
     }
     else if(ex2_dz_no == 1) {
          A1G = DSP_ALU_DSTG & MASK000000FF;
          if(DSP_ALU_DSTG_BIT7) A1G = A1G | MASKFFFFFF00;
     }
     negative_bit = DSP_ALU_DSTG_BIT7;
     zero_bit = (DSP_ALU_DST == 0) & (DSP_ALU_DSTG_LSB8 == 0);
}
```

- **動作内容11：論理演算DCビット**

```
/* logical_dc_bit.c */
{
     unsigned short int       negative_bit, zero_bit;
     switch (DSPCSBITS) {
          case 0x0: DSPDCBIT = 0;
```

```
                    break;
          case 0x1: DSPDCBIT = negative_bit;
                    break;
          case 0x2: DSPDCBIT = zero_bit;
                    break;
          case 0x3: DSPDCBIT = 0;
                    break;
          case 0x4: DSPDCBIT = 0;
                    break;
          case 0x5: DSPDCBIT = 0;
                    break;
     }
     DSPGTBIT = 0;
     DSPZBIT = zero_bit;
     DSPNBIT = negative_bit;
     DSPVBIT = 0;
     if(DSPTSBITS != 0x0) {
          switch (DSPTSBITS) {
              case 0x0: if(DSPTCBIT == 1) CPUSRTBIT = 0;
                        break;
              case 0x1: DSPANBIT = DSPANBIT | nagative_bit;
                        if(DSPTCBIT == 1) CPUSRTBIT = negative_bit;
                        break;
              case 0x2: DSPAZBIT = DSPAZBIT | zero_bit;
                        if(DSPTCBIT == 1) CPUSRTBIT = zero_bit;
                        break;
              case 0x3: DSPAVBIT = DSPAVBIT;
                        if(DSPTCBIT == 1) CPUSRTBIT = 0;
                        break;
              case 0x4: DSPAGTBIT = DSPAGTBIT;
                        if(DSPTCBIT == 1) CPUSRTBIT = 0;
                        break;
              case 0x5: if(DSPTCBIT == 1) CPUSRTBIT = 0;
                        break;
          }
     }
}
```

- **動作内容12：シフト演算DCビット**

```
/* shift_dc_bit.c */
{
     unsigned short int  carry_bit, zero_bit, negative_bit, overflow_bit,
                         overflow_avbit;
     switch (DSPCSBITS) {
          case 0x0:  DSPDCBIT = carry_bit;
                     break;
          case 0x1:  DSPDCBIT = negative_bit;
                     break;
          case 0x2:  DSPDCBIT = zero_bit;
                     break;
          case 0x3:  DSPDCBIT = overflow_bit;
                     break;
          case 0x4:  DSPDCBIT = 0;
                     break;
          case 0x5:  DSPDCBIT = 0;
                     break;
     }
     DSPGTBIT = 0;
     DSPZBIT = zero_bit;
     DSPNBIT = negative_bit;
     DSPVBIT = overflow_bit;
     if(DSPTSBITS != 0x0) {
          switch (DSPTSBITS) {
              case 0x0:  if(DSPTCBIT == 1) CPUSRTBIT = cary_bit;
                         break;
              case 0x1:  DSPANBIT = DSPANBIT | nagative_bit;
                         if(DSPTCBIT == 1) CPUSRTBIT = negative_bit;
                         break;
              case 0x2:  DSPAZBIT = DSPAZBIT | zero_bit;
                         if(DSPTCBIT == 1) CPUSRTBIT = zero_bit;
                         break;
              case 0x3:  DSPAVBIT = DSPAVBIT | overflow_avbit;
                         if(DSPTCBIT == 1) CPUSRTBIT = 0;
                         break;
              case 0x4: DSPAGTBIT = DSPAGTBIT;
                         if(DSPTCBIT == 1) CPUSRTBIT = 0;
```

```
                        break;
                case 0x5: if(DSPTCBIT == 1) CPUSRTBIT = 0;
                        break;
            }
        }
    }
```

11.4.1 [if cc] PABS　　ABSolute　　DSP 算術演算命令

絶対値演算

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PABS Sx,Dz	もし Sx≧0 ならば Sx→Dz	111110**********	1	更新	更新
	もし Sx＜0 ならば 0－Sx→Dz	10001000xx00zzzz			
PABS Sy,Dz	もし Sy≧0 ならば Sy→Dz	111110**********	1	更新	更新
	もし Sy＜0 ならば 0－Sy→Dz	1010100000yyzzzz			
DCT PABS Sx,Dz	もし DC=1 & Sx≧0 ならば	111110**********	1	－	－
	Sx→Dz	10001010xx01zzzz			
	もし DC=1 & Sx＜0 ならば				
	0－Sx→Dz				
	もし DC=0 ならば nop.				
DCT PABS Sy,Dz	もし DC=1 & Sy≧0 ならば	111110**********	1	－	－
	Sy→Dz	1010101001yyzzzz			
	もし DC=1 & Sy＜0 ならば				
	0－Sy→Dz				
	もし DC=0 ならば nop.				
DCF PABS Sx,Dz	もし DC=0 & Sx≧0 ならば	111110**********	1	－	－
	Sx→Dz	10001011xx01zzzz			
	もし DC=0 & Sx＜0 ならば				
	0－Sx→Dz				
	もし DC=1 ならば nop.				
DCF PABS Sy,Dz	もし DC=0 & Sy≧0 ならば	111110**********	1	－	－
	Sy→Dz	1010101101yyzzzz			
	もし DC=0 & Sy＜0 ならば				
	0－Sy→Dz				
	もし DC=1 ならば nop.				

(1) 説明

絶対値を求めます。もし Sx、Sy オペランドの内容が正の値のときは Sx、Sy オペランドの内容を Dz オペランドへ転記します。もし負の値のときは符号を反転して Dz オペランドへ格納します。

条件が指定されていない場合は、DSR レジスタの DC ビットは CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

条件が指定されている場合は、条件が真であっても、DC、N、Z、V、GT、AN、AZ、AV、AGT ビットは更新されません。SR レジスタの T ビットも更新されません。

(2) 注意

特にありません。

(3) 動作内容

```
{
    unsigned short int ex2_dz_no, carry_bit, borrow_bit, overflow_bit, overrange_bit;
    DSP_ALU_SRC1  = 0;
    DSP_ALU_SRC1G = 0;
    if(EX2_DSP_BIT13 == 0) {        /* 0 +/- Sx -> Dz */
          switch (EX2_SX) {
               case 0x0:  DSP_ALU_SRC2  = X0;
                          if(DSP_ALU_SRC2_MSB) DSP_ALU_SRC2G = 0xff;
                          else DSP_ALU_SRC2G = 0x0;
                          break;
               case 0x1:  DSP_ALU_SRC2 = X1;
                          if(DSP_ALU_SRC2_MSB) DSP_ALU_SRC2G = 0xff;
                          else DSP_ALU_SRC2G = 0x0;
                          break;
               case 0x2:  DSP_ALU_SRC2  = A0;
                          DSP_ALU_SRC2G = A0G;
                          break;
               case 0x3:  DSP_ALU_SRC2  = A1;
                          DSP_ALU_SRC2G = A1G;
                          break;
          }
    }
```

```
    else  {    /* 0 +/- Sy -> Dz */
          switch (EX2_SY) {
               case 0x0:  DSP_ALU_SRC2  = Y0;
                          break;
               case 0x1:  DSP_ALU_SRC2  = Y1;
                          break;
               case 0x2:  DSP_ALU_SRC2  = M0;
                          break;
               case 0x3:  DSP_ALU_SRC2  = M1;
                          break;
          }
          if(DSP_ALU_SRC2_MSB)  DSP_ALU_SRC2G = 0xff;
          else DSP_ALU_SRC2G = 0x0;
    }

    if(DSP_ALU_SRC2G_BIT7 == 0) {    /* positive value */
          DSP_ALU_DST = 0x0 + DSP_ALU_SRC2;
          carry_bit = 0;
          DSP_ALU_DSTG_LSB8= 0x0 + DSP_ALU_SRC2G_LSB8 + carry_bit;
    }
    else {                    /* negative value */
          DSP_ALU_DST = 0x0 - DSP_ALU_SRC2;
          borrow_bit = 1;
          DSP_ALU_DSTG_LSB8 = 0x0 - DSP_ALU_SRC2G_LSB8 - borrow_bit;
    }
    overflow_bit = !(POS_NOT_OV || NEG_NOT_OV);
    overrange_bit = PLUS_OP_G_OV;
#include "fixed_pt_overflow_protection.c"
    if(DSP_UNCONDITIONAL_UPDATE){    /* ubconditional operation */
#include "fixed_pt_unconditional_update.c"

          if(DSP_ALU_SRC2G_BIT7 == 0) {
#include "fixed_pt_plus_dc_bit.c"
          }
          else{
               overflow_bit = !(POS_NOT_OV || NEG_NOT_OV);
               overrange_bit = MINUS_OP_G_OV;
#include "fixed_pt_minus_dc_bit.c"
```

```
            }
        }
        else  if(DSP_CONDITION_MATCH){    /* conditional operation and match */
              DSP_REG[ex2_dz_no] = DSP_ALU_DST;
                   if(ex2_dz_no == 0){
                             A0G = DSP_ALU_DSTG & MASK000000FF;
                             If(DSP_ALU_DSTG_BIT7) A0G = A0G | MASKFFFFFF00;
                   }
                   else if(ex2_dz_no == 1){
                             A1G = DSP_ALU_DSTG & MASK000000FF;
                             If(DSP_ALU_DSTG_BIT7) A1G = A1G | MASKFFFFFF00;
                   }
              }
        }
    }

    break;
```

(4) 使用例

```
PABS X0,M0  NOPX  NOPY          ;実行前：X0=H'33333333, M0=H'12345678
                                 実行後：X0=H'33333333, M0=H'33333333
PABS X1,X1  NOPX  NOPY          ;実行前：X1=H'DDDDDDDD
                                 実行後：X1=H'22222223
                                 DC ビットは CS[2:0]の状態に従って更新。
```

11.4.2 [if cc] PADD ADD with Condition DSP 算術演算命令

条件付き加算

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PADD Sx,Sy,Dz	Sx＋Sy→Dz	111110**********	1	更新	更新
		10110001xxyyzzzz			
DCT PADD Sx,Sy,Dz	もし DC＝1 ならば Sx＋Sy→Dz	111110**********	1	−	−
	もし DC＝0 ならば nop.	10110010xxyyzzzz			
DCF PADD Sx,Sy,Dz	もし DC＝0 ならば Sx＋Sy→Dz	111110**********	1	−	−
	もし DC＝1 ならば nop.	10110011xxyyzzzz			

(1) 説明

Sx、Sy オペランドの内容を加算し、その結果を Dz オペランドへ格納します。DCT、DCF の条件が指定されている場合は、条件が真のとき命令が実行されます。条件が偽のとき命令は実行されません。

条件が指定されていない場合は DSR レジスタの DC ビットは CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

条件が指定されている場合は、条件が真であっても、DC、N、Z、V、GT、AN、AZ、AV、AGT ビットは更新されません。SR レジスタの T ビットも更新されません。

(2) 注意

特にありません。

(3) 動作内容

```
{
    unsigned short int   ex2_dz_no, carry_bit, overflow_bit, overrange_bit;
    switch (EX2_SX) {
        case 0x0:  DSP_ALU_SRC1  = X0;
                   if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                   else    DSP_ALU_SRC1G = 0x0;
                   break;
        case 0x1:  DSP_ALU_SRC1  = X1;
                   if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                   else    DSP_ALU_SRC1G = 0x0;
                   break;
```

```
            case 0x2:  DSP_ALU_SRC1  = A0;
                       DSP_ALU_SRC1G = A0G;
                       break;
            case 0x3:  DSP_ALU_SRC1  = A1;
                       DSP_ALU_SRC1G = A1G;
                       reak;
       }
       switch (EX2_SY) {
            case 0x0:  DSP_ALU_SRC2  = Y0;
                       break;
            case 0x1:  DSP_ALU_SRC2  = Y1;
                       break;
            case 0x2:  DSP_ALU_SRC2  = M0;
                       break;
            case 0x3:  DSP_ALU_SRC2  = M1;
                       break;
       }
       if(DSP_ALU_SRC2_MSB) DSP_ALU_SRC2G = 0xff;
       else   DSP_ALU_SRC2G = 0x0;

       DSP_ALU_DST = DSP_ALU_SRC1 + DSP_ALU_SRC2;
       carry_bit = ((DSP_ALU_SRC1_MSB | DSP_ALU_SRC2_MSB) & !DSP_ALU_DST_MSB) |
                   (DSP_ALU_SRC1_MSB & DSP_ALU_SRC2_MSB);
       DSP_ALU_DSTG_LSB8 = DSP_ALU_SRC1G_LSB8 + DSP_ALU_SRC2G_LSB8 + carry_bit;

       overflow_bit= !(POS_NOT_OV || NEG_NOT_OV);
       overrange_bit = PLUS_OP_G_OV;
#include "fixed_pt_overflow_protection.c"

       if(DSP_UNCONDITIONAL_UPDATE) { /* unconditional operation */
#include "fixed_pt_unconditional_update.c"
#include "fixed_pt_plus_dc_bit.c"
       }
       else if(DSP_CONDITION_MATCH) { /* conditional operation and match */
                 DSP_REG[ex2_dz_no] = DSP_ALU_DST;
                 If(ex2_dz_no==0) {
                      A0G = DSP_ALU_DSTG & MASK000000FF;
                      If(DSP_ALU_DSTG_BIT7) A0G = A0G | MASKFFFFFF00;
```

```
                }
            }
        }
        else if(ex2_dz_no==1) {
                A1G = DSP_ALU_DSTG & MASK000000FF;
                If(DSP_ALU_DSTG_BIT7) A1G = A1G | MASKFFFFFF00;
            }
        }
    }

    break;
```

(4) 使用例

```
PADD X0,Y0,A0  NOPX  NOPY      ;実行前： X0=H'22222222, Y0=H'33333333,
                                          A0=H'123456789A
                                実行後： X0=H'22222222, Y0=H'33333333,
                                          A0=H'0055555555
```

無条件実行の場合、DC ビットは演算直前の CS[2:0]ビットの状態に従って更新。

11.4.3 PADD PMULS ADD & MULtiply as Signed DSP 算術演算命令

加算と符号付き乗算

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PADD Sx,Sy,Du PMULS Se,Sf,Dg	Sx＋Sy→Du Se の MSW×Sf の MSW→Dg	111110********** 0111eeffxxyygguu	1	更新	更新

(1) 説明

Sx、Sy オペランドの内容を加算し、結果を Du オペランドへ格納します。Se、Sf オペランドの上位ワードの内容を符号付きとして乗算し、結果を Dg オペランドに格納します。この 2 つの処理は同時に並行して実行されます。

DSR レジスタの DC ビットは ALU 演算の結果と CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも ALU 演算の結果に従って更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

(2) 注意

PMULS は固定小数点乗算ですので、ソースデータが同じでも MULS とは演算結果が異なります。

(3) 動作内容

```
{
/* Multiplier Sources assignment */
     unsigned short int  carry_bit, negatve_bit, zero_bit, overflow_bit,
                         overrange_bit;
     switch (EX2_SE){
          case 0x0:  DSP_MLT_SRC1 = X0;
                     break;
          case 0x1:  DSP_MLT_SRC1 = X1;
                     break;
          case 0x2:  DSP_MLT_SRC1 = Y0;
                     break;
          case 0x3:  DSP_MLT_SRC1 = A1;
                     break;
     }
     switch (EX2_SF){
          case 0x0:  DSP_MLT_SRC2 = Y0;
                     break;
```

```
          case 0x1:  DSP_MLT_SRC2 = Y1;
                     break;
          case 0x2:  DSP_MLT_SRC2 = X0;
                     break;
          case 0x3:  DSP_MLT_SRC2 = X1;
                     break;
     }

/* ALU Sources assignment */
     switch (EX2_SX){
          case 0x0:  DSP_ALU_SRC1 = X0;
                     if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                     else     DSP_ALU_SRC1G = 0x0;
                     break;
          case 0x1:  DSP_ALU_SRC1 = X1;
                     if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                     else     DSP_ALU_SRC1G = 0x0;
                     break;
          case 0x2:  DSP_ALU_SRC1 = A0;
                     DSP_ALU_SRC1G = A0G;
                     break;
          case 0x3:  DSP_ALU_SRC1 = A1;
                     DSP_ALU_SRC1G = A1G;
                     break;
     }
     switch (EX2_SY){
          case 0x0:  DSP_ALU_SRC2 = Y0;
                     break;
          case 0x1:  DSP_ALU_SRC2 = Y1;
                     break;
          case 0x2:  DSP_ALU_SRC2 = M0;
                     break;
          case 0x3:  DSP_ALU_SRC2 = M1;
                     break;
     }
     if(DSP_ALU_SRC2_MSB) DSP_ALU_SRC2G = 0xff;
     else   DSP_ALU_SRC2G = 0x0;
```

```
/* Multiplier Operation */
     if((SBIT == 1) && (DSP_MLT_SRC1_HW == 0x8000) && (DSP_MLT_SRC2_HW == 0x8000)
     {
          DSP_MLT_DST = 0x7fffffff;             /* overflow protection */
     }
     else {
          DSP_MLT_DST = ((long)(short)DSP_MLT_SRC1_HW
                        * (long)(short)DSP_MLT_SRC2_HW << 1;
     }
     if(DSP_MLT_DST_MSB) DSP_MLT_DSTG_LSB8 = 0xff;
     else DSP_MLT_DTSG_LSB8 = 0x0;

/* Multiplier Destination assignment */
     switch (EX2_DG){
          case 0x0:  M0 = DSP_MLT_DST;
                     break;
          case 0x1:  M1 = DSP_MLT_DST;
                     break;
          case 0x2:  A0 = DSP_MLT_DST;
                     if(DSP_MLT_DSTG_LSB8 = 0x0) A0G = 0x0;
                     else A0G = 0xffffffff;
                     break;
          case 0x3:  A1 = DSP_MLT_DST;
                     if(DSP_MLT_DSTG_LSB8 = 0x0) A1G = 0x0;
                     else A1G = 0xffffffff;
                     break;
     }

/* ALU operation */
     DSP_ALU_DST = DSP_ALU_SRC1 + DSP_ALU_SRC2;
     carry_bit = ((DSP_ALU_SRC1_MSB | DSP_ALU_SRC2_MSB) & !DSP_ALU_DST_MSB) |
                 (DSP_ALU_SRC1_MSB & DSP_ALU_SRC2_MSB);
     DSP_ALU_DSTG_LSB8 = DSP_ALU_SRC1G_LSB8 + DSP_ALU_SRC2G_LSB8 + carry_bit;
     overflow_bit= !(POS_NOT_OV || NEG_NOT_OV);
     overrange_bit = PLUS_OP_G_OV;
#include "fixed_pt_overflow_protection.c"
     switch (EX2_DU) {
          case 0x0:  X0  = DSP_ALU_DST;
```

```
                    negative_bit = DSP_ALU_DSTG_BIT7
                    zero_bit = (DSP_ALU_DST==0)&(DSP_ALU_DSTG_LSB8==0);
                    break;
          case 0x1: Y0  = DSP_ALU_DST;
                    negative_bit = DSP_ALU_DSTG_BIT7;
                    zero_bit = (DSP_ALU_DST == 0) & (DSP_ALU_DSTG_LSB8 == 0);
                    break;
          case 0x2: A0  = DSP_ALU_DST;
                    A0G = DSP_ALU_DSTG & MASK000000FF;
                    If(DSP_ALU_DSTG_BIT7) A0G = A0G | MASKFFFFFF00;
                    negative_bit = DSP_ALU_DSTG_BIT7;
                    zero_bit = (DSP_ALU_DST == 0) & (DSP_ALU_DSTG_LSB8 == 0);
                    break;
          case 0x3: A1  = DSP_ALU_DST;
                    A1G = DSP_ALU_DSTG & MASK000000FF;
                    If(DSP_ALU_DSTG_BIT7) A1G = A1G | MASKFFFFFF00;
                    negative_bit = DSP_ALU_DSTG_BIT7;
                    zero_bit = (DSP_ALU_DST == 0) & (DSP_ALU_DSTG_LSB8 == 0);
                    break;
     }
#include "fixed_pt_plus_dc_bit.c"
}

break;
```

(4) 使用例

```
PADD A0,M0,A0 PMULS X0,Y0,M0  NOPX  NOPY;
```

実行前：X0=H'00020000,Y0=H'00030000,
M0=H'22222222,A0=H'0055555555

実行後：X0=H'00020000,Y0=H'00030000,
M0=H'0000000C,A0=H'0077777777

DCビットはCS[2:0]の状態に従ってPADD動作の結果に基づいて更新。

11.4.4 PADDC ADD with Carry DSP 算術演算命令

キャリ付き加算

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PADDC Sx,Sy,Dz	Sx＋Sy＋DC→Dz	111110********** 10110000xxyyzzzz	1	キャリ	更新

(1) 説明

Sx、Sy オペランドの内容と DC ビットを加算し、Dz オペランドへ格納します。

DSR レジスタの DC ビットはキャリフラグとして更新されます。DSR レジスタの N、Z、V、GT ビットも更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AGT、AZ、AN、AV の各ビットにも演算結果が反映され、結果が累積されます。

(2) 注意

PADDC 命令実行後の DC ビットは、CS ビットに関係なく、キャリフラグとして更新されます。

(3) 動作内容

```
{
     unsigned short int   carry_bit, overflow_bit, overrange_bit;
     switch (EX2_SX) {
          case 0x0:  DSP_ALU_SRC1 = X0;
                     if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                     else DSP_ALU_SRC1G = 0x0;
                     break;
          case 0x1:  DSP_ALU_SRC1  = X1;
                     if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                     else DSP_ALU_SRC1G = 0x0;
                     break;
          case 0x2:  DSP_ALU_SRC1  = A0;
                     DSP_ALU_SRC1G = A0G;
                     break;
          case 0x3:  DSP_ALU_SRC1  = A1;
                     DSP_ALU_SRC1G = A1G;
                     break;
     }
```

```
    switch (EX2_SY) {
        case 0x0:  DSP_ALU_SRC2  = Y0;
                   break;
        case 0x1:  DSP_ALU_SRC2  = Y1;
                   break;
        case 0x2:  DSP_ALU_SRC2  = M0;
                   break;
        case 0x3:  DSP_ALU_SRC2  = M1;
                   break;
    }
    if(DSP_ALU_SRC2_MSB) DSP_ALU_SRC2G = 0xff;
    else   DSP_ALU_SRC2G = 0x0;

    DSP_ALU_DST = DSP_ALU_SRC1 + DSP_ALU_SRC2 + DSPDCBIT;
    carry_bit = ((DSP_ALU_SRC1_MSB | DSP_ALU_SRC2_MSB) & !DSP_ALU_DST_MSB) |
                (DSP_ALU_SRC1_MSB & DSP_ALU_SRC2_MSB);
    DSP_ALU_DSTG_LSB8 = DSP_ALU_SRC1G_LSB8 + DSP_ALU_SRC2G_LSB8 + carry_bit;
    overflow_bit= !(POS_NOT_OV || NEG_NOT_OV);
    overrange_bit = PLUS_OP_G_OV;
#include "fixed_pt_overflow_protection.c"

#include "fixed_pt_unconditional_update.c"
#include "fixed_pt_dc_always_carry.c"
}

break;
```

(4) 使用例

CS[2:0]=***:CS ビットの状態に関係なく、常に Carry or Borrow Mode として動作します。

```
PADDC X0,Y0,M0  NOPX  NOPY ;   実行前:X0=H'B3333333, Y0=H'55555555
                                      M0=H'12345678, DC=0
                               実行後:X0=H'B3333333, Y0=H'55555555
                                      M0=H'08888888, DC=1
PADDC X0,Y0,M0  NOPX  NOPY ;   実行前:X0=H'33333333, Y0=H'55555555
                                      M0=H'12345678, DC=1
                               実行後:X0=H'33333333, Y0=H'55555555
                                      M0=H'88888889, DC=0
                               DC ビットは CS[2:0]ビットの状態に従って更新。
```

11.4.5 [if cc] PAND logical AND DSP 論理演算命令

条件付き論理積演算

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PAND Sx,Sy,Dz	Sx & Sy→Dz、Dz ガードビットと LSW クリア	111110********** 10010101xxyyzzzz	1	更新	更新
DCT PAND Sx,Sy,Dz	もし DC＝1 ならば Sx&Sy→Dz、Dz ガードビットと LSW クリア もし DC＝0 ならば nop.	111110********** 10010110xxyyzzzz	1	−	−
DCF PAND Sx,Sy,Dz	もし DC＝0 ならば Sx&Sy→Dz、Dz ガードビットと LSW クリア もし DC＝1 ならば nop.	111110********** 10010111xxyyzzzz	1	−	−

(1) 説明

Sx オペランドの上位ワードの内容と Sy オペランドの上位ワードの内容との論理積を演算し、その結果を Dz オペランドの上位ワードへ格納し、Dz オペランドの下位ワードを 0 でクリアします。Dz がガードビットを持つレジスタの場合は、ガードビットも 0 でクリアします。DCT、DCF の条件が指定されている場合は、条件が真のとき命令が実行されます。条件が偽のとき命令は実行されません。

条件が指定されていない場合は、DSR レジスタの DC ビットは CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

条件が指定されている場合は、条件が真であっても、DC、N、Z、V、GT ビットは更新されません。

(2) 注意

デスティネーションレジスタの下位ワードの内容とガードビットの内容は DC ビットの更新には無視されます。

(3) 動作内容

```
{
    unsigned short int    carry_bit, negative_bit, overflow_bit, overrange_bit;
    switch (EX2_SX) {
        case 0x0:  DSP_ALU_SRC1 = X0;
                   break;
        case 0x1:  DSP_ALU_SRC1 = X1;
                   break;
```

```
            case 0x2:   DSP_ALU_SRC1 = A0;
                        break;
            case 0x3:   DSP_ALU_SRC1 = A1;
                        break;
        }
        switch (EX2_SY) {
            case 0x0:   DSP_ALU_SRC2 = Y0;
                        break;
            case 0x1:   DSP_ALU_SRC2 = Y1;
                        break;
            case 0x2:   DSP_ALU_SRC2 = M0;
                        break;
            case 0x3:   DSP_ALU_SRC2 = M1;
                        break;
        }

        DSP_ALU_DST_HW = DSP_ALU_SRC1_HW & DSP_ALU_SRC2_HW;

        if(DSP_UNCONDITIONAL_UPDATE) {   /* unconditional operation */
            DSP_REG_WD[ex2_dz_no * 2] = DSP_ALU_DST_HW;
            DSP_REG_WD[ex2_dz_no * 2 + 1] = 0x0;      /* clear LSW */
            if(ex2_dz_no == 0)       A0G = 0x0; /* clear Guard bits */
            else if(ex2_dz_no == 1)  A1G = 0x0;

            carry_bit    = 0x0;
            negative_bit = DSP_ALU_DST_MSB;
            zero_bit     = (DSP_ALU_DST_HW == 0);
            overflow_bit = 0x0;
    #include "logical_dc_bit.c"
        }
        else if(DSP_CONDITION_MATCH) {   /* conditional operation and match */
            DSP_REG_WD[ex2_dz_no * 2] = DSP_ALU_DST_HW;
            DSP_REG_WD[ex2_dz_no * 2 + 1] = 0x0;      /* clear LSW */
            if(ex2_dz_no == 0)       A0G = 0x0; /* clear Guard bits */
            else if(ex2_dz_no == 1)  A1G = 0x0;
        }
    }
```

```
break;
```

(4) 使用例

```
PAND X0,Y0,A0  NOPX  NOPY ;
```

実行前:X0=H'33333333, Y0=H'55555555
A0=H'123456789A

実行後:X0=H'33333333, Y0=H'55555555
A0=H'0011110000

無条件実行の場合、DC ビットは CS[2:0]の状態に従って更新。

11.4.6 [if cc] PCLR CLeaR DSP 算術演算命令

条件付きクリア

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PCLR Dz	H'00000000→Dz	111110********** 100011010000zzzz	1	更新	更新
DCT PCLR Dz	もし DC＝1 ならば H'00000000→Dz もし DC＝0 ならば nop.	111110********** 100011100000zzzz	1	−	−
DCF PCLR Dz	もし DC＝0 ならば H'00000000→Dz もし DC＝1 ならば nop.	111110********** 100011110000zzzz	1	−	−

(1) 説明

Dz オペランドの内容を 0 でクリアします。DCT、DCF の条件が指定されている場合は、条件が真のとき命令が実行されます。条件が偽のとき命令は実行されません。

条件が指定されていない場合は、DSR レジスタの DC ビットは CS ビットの指定に従って更新されます。また、SR レジスタの S ビットは、TC ビットが 1 のときに TS ビットの指定に従って更新されます。DSR レジスタの Z ビットは 1 にセットされ、N、V、GT ビットは 0 にクリアされます。TS ビットがすべて 0 でないときは、累積ビットも更新され、AZ ビットに 1 がセットされます。

条件が指定されている場合は、条件が真であっても、DC、N、Z、V、GT ビットは更新されません。

(2) 注意事項

特になし

(3) 動作内容

```
{    /* 0 + 0 -> Dz */
     unsigned short int     ex2_dz_no, carry_bit, negative_bit, zero_bit,
                            overflow_bit, overrange_bit;
     if(DSP_UNCONDITIONAL_UPDATE) { /* unconditional operation */
          DSP_REG[ex2_dz_no] = 0x0;
          if(ex2_dz_no == 0)       A0G = 0x0;
          else if(ex2_dz_no == 1)  A1G = 0x0;

          carry_bit     = 0;
          negative_bit  = 0;
```

```
            zero_bit      = 1;
            overflow_bit  = 0;
            overrange_bit = 0:

  #include "fixed_pt_plus_dc_bit.c"
       }
       else if(DSP_CONDITION_MATCH) { /* conditional operation and match */
            DSP_REG[ex2_dz_no] = 0x0;
       }
  }

  break;
```

(4) 使用例

```
PCLR A0  NOPX  NOPY ;
```

実行前:A0=H'FF87654321

実行後:A0=H'0000000000

無条件実行の場合、DC ビットは CS[2:0]の状態に従って更新。

11.4.7 PCLR PMULS Clear & MULtiply as Signed DSP 算術演算命令

クリアと符号付き乗算

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PCLR Du PMULS Se,Sf,Dg	H'00000000→Du Se の MSW×Sf の MSW→Dg	111110********** 0100eeff0001gguu	1	更新	更新

(1) 説明

Du オペランドの内容を 0 でクリアします。Se、Sf オペランドの上位ワードの内容を符号付きとして乗算し、結果を Dg オペランドに格納します。この 2 つの処理は同時に並行して実行されます。

DSR レジスタの DC ビットは ALU 演算の結果と CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも ALU 演算の結果に従って更新されます。

(2) 注意

PMULS は固定小数点乗算ですので、ソースデータが同じでも MULS とは演算結果が異なります。

(3) 動作内容

```
{
/* Multiplier Sources assignment */
     unsigned short int     carry_bit, negative_bit, zero_bit, overflow_bit,
                            overrange_bit;
     switch (EX2_SE){
          case 0x0:  DSP_MLT_SRC1 = X0;
                     break;
          case 0x1:  DSP_MLT_SRC1 = X1;
                     break;
          case 0x2:  DSP_MLT_SRC1 = Y0;
                     break;
          case 0x3:  DSP_MLT_SRC1 = A1;
                     break;
     }
     switch (EX2_SF){
          case 0x0:  DSP_MLT_SRC2 = Y0;
                     break;
          case 0x1:  DSP_MLT_SRC2 = Y1;
                     break;
```

```
          case 0x2:  DSP_MLT_SRC2 = X0;
                     break;
          case 0x3:  DSP_MLT_SRC2 = A1;
                     break;
     }

/* Multiplier Operation */
     if((SBIT == 1) && (DSP_MLT_SRC1_HW == 0x8000) && (DSP_MLT_SRC2_HW == 0x8000)
     {
        DSP_MLT_DST = 0x7fffffff;              /* overflow protection */
     }
     else{
       DSP_MLT_DST = ((long)(short)DSP_MLT_SRC1_HW
                     * (long)(short)DSP_MLT_SRC2_HW) << 1;
}
     if(DSP_MLT_DST_MSB) DSP_MLT_DSTG_LSB8 = 0xff;
     else DSP_MLT_DTSG_LSB8 = 0x0;

/* Multiplier Destination assignment */
     switch(EX2_DG){
          case 0x0:  M0 = DSP_MLT_DST;
                     break;
          case 0x1:  M1 = DSP_MLT_DST;
                     break;
          case 0x2:  A0 = DSP_MLT_DST;
                     if(DSP_MLT_DSTG_LSB8 = 0x0) A0G = 0x0;
                     else A0G = 0xffffffff;
                     break;
          case 0x3:  A1 = DSP_MLT_DST;
                     if(DSP_MLT_DSTG_LSB8 = 0x0) A1G = 0x0;
                     else A1G = 0xffffffff;
                     break;
     }

/ * ALU operation */
     DSP_ALU_DST = 0x0;
     carry_bit = 0;
     negative_bit = 0;
```

```
        zero_bit = 1;
        overflow_bit = 0;
        overrange_bit = 0;

    #include "fixed_pt_overflow_protection.c"
        switch(EX2_DU){
            case 0x0:
                        X0 = DSP_ALU_DST;
                        break;
            case 0x1:
                        Y0 = DSP_ALU_DST;
                        break
            case 0x2:
                        A0 = DSP_ALU_DST;
                        A0G = 0x0;
                        break;
            case 0x3:
                        A1 = DSP_ALU_DST;
                        A1G = 0x0;
                        break;
        }
    #include "fixed_pt_plus_dc_bit.c"
    }

    break;
```

(4) 使用例

```
PCLR A0  PMULS X0,Y0,M0  NOPX  NOPY ;
                              実行前:X0=H'00020000,Y0=H'00030000,
                                    M0=H'22222222,A0=H'0055555555
                              実行後:X0=H'00020000,Y0=H'00030000,
                                    M0=H'0000000C,A0=H'0000000000
                              DC ビットは CS[2:0]の状態に従って PCLR 結果に基づいて更新。
```

11.4.8 PCMP CoMPare two data DSP 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	DC ビット	T ビット
PCMP Sx,Sy	Sx－Sy	111110********** 10000100xxyy0000	1	更新	更新

(1) 説明

Sx オペランドの内容から Sy オペランドの内容を減算します。

DSR レジスタの DC ビットは CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

(2) 注意

特にありません。

(3) 動作内容

```
{
     unsigned short int carry_bit, borrow_bit, negative_bit, zero_bit,
                        overflow_bit, overrange_bit;
     switch (EX2_SX){
          case 0x0:  DSP_ALU_SRC1  = X0;
                     if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                     else  DSP_ALU_SRC1G = 0x0;
                     break;
          case 0x1:  DSP_ALU_SRC1  = X1;
                     if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                     else  DSP_ALU_SRC1G = 0x0;
                     break;
          case 0x2:  DSP_ALU_SRC1  = A0;
                     DSP_ALU_SRC1G = A0G;
                     break;
          case 0x3:  DSP_ALU_SRC1  = A1;
                     DSP_ALU_SRC1G = A1G;
                     break;
```

```
        }
        switch (EX2_SY){
             case 0x0:  DSP_ALU_SRC2  = Y0;
                        break;
             case 0x1:  DSP_ALU_SRC2  = Y1;
                        break;
             case 0x2:  DSP_ALU_SRC2  = M0;
                        break;
             case 0x3:  DSP_ALU_SRC2  = M1;
                        break;
        }

        if(DSP_ALU_SRC2_MSB)  DSP_ALU_SRC2G = 0xff;
        else   DSP_ALU_SRC2G = 0x0;
        DSP_ALU_DST = DSP_ALU_SRC1 - DSP_ALU_SRC2;
        carry_bit =((DSP_ALU_SRC1_MSB | !DSP_ALU_SRC2_MSB) && !DSP_ALU_DST_MSB) |
                   (DSP_ALU_SRC1_MSB & !DSP_ALU_SRC2_MSB);
        borrow_bit = !carry_bit;
        DSP_ALU_DSTG_LSB8 = DSP_ALU_SRC1G_LSB8 - DSP_ALU_SRC2G_LSB8 - borrow_bit;

        negative_bit = DSP_ALU_DSTG_BIT7;
        zero_bit = (DSP_ALU_DST==0) & (DSP_ALU_DSTG_LSB8==0);
        overflow_bit = !(POS_NOT_OV || NEG_NOT_OV);
        overrange_bit = MINUS_OP_G_OV;
    #include "fixed_pt_overflow_protection.c"
    #include "fixed_pt_minus_dc_bit.c"
    }

    break;
```

(4) 使用例

```
PCMP X0,Y0  NOPX  NOPY          ;実行前:X0=H'22222222, Y0=H'33333333
                                 実行後:X0=H'22222222, Y0=H'33333333
                                       N=1,Z=0,V=0,GT=0
                                DC ビットはCS[2:0]の状態に従って更新。
```

11.4.9 [if cc] PCOPY COPY with Condition DSP 算術演算命令

条件付き転記

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PCOPY Sx,Dz	Sx→Dz	111110**********	1	更新	更新
		11011001xx00zzzz			
PCOPY Sy,Dz	Sy→Dz	111110**********	1	更新	更新
		1111100100yyzzzz			
DCT PCOPY Sx,Dz	もし DC＝1 ならば Sx→Dz もし DC＝0 ならば nop.	111110**********	1	−	−
		11011010xx00zzzz			
DCT PCOPY Sy,Dz	もし DC＝1 ならば Sy→Dz もし DC＝0 ならば nop.	111110**********	1	−	−
		1111101000yyzzzz			
DCF PCOPY Sx,Dz	もし DC＝0 ならば Sx→Dz もし DC＝1 ならば nop.	111110**********	1	−	−
		11011011xx00zzzz			
DCF PCOPY Sy,Dz	もし DC＝0 ならば Sy→Dz もし DC＝1 ならば nop	111110**********	1	−	−
		1111101100yyzzzz			

(1) 説明

Sx、Sy オペランドの内容を Dz オペランドへ格納します。DCT、DCF の条件が指定されている場合は、条件が真のとき命令が実行されます。条件が偽のとき命令は実行されません。

条件が指定されていない場合は DSR レジスタの DC ビットは CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

条件が指定されている場合は、条件が真であっても、DC、N、Z、V、GT ビットは更新されません。

(2) 注意

特にありません。

(3) 動作内容

```
{    /* Sx + 0 -> Dz */
     unsigned short int     ex2_dz_no, carry_bit, overflow_bit, overrange_bit;
     if(EX2_DSP_BIT13==0) {         /* Sx + 0 -> Dz */
          switch (EX2_SX) {
               case 0x0:  DSP_ALU_SRC1  = X0;
                          if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                          else  DSP_ALU_SRC1G = 0x0;
```

```
                    break;
          case 0x1:  DSP_ALU_SRC1  = X1;
                    if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                    else  DSP_ALU_SRC1G = 0x0;
                    break;
          case 0x2:  DSP_ALU_SRC1  = A0;
                    DSP_ALU_SRC1G = A0G;
                    break;
          case 0x3:  DSP_ALU_SRC1  = A1;
                    DSP_ALU_SRC1G = A1G;
                    break;
     }
     DSP_ALU_SRC2 = 0;
     DSP_ALU_SRC2G= 0;
}
else  {   /* 0 + Sy -> Dz */
     DSP_ALU_SRC1 = 0;
     DSP_ALU_SRC1G = 0;
     switch (EX2_SY) {
          case 0x0:  DSP_ALU_SRC2  = Y0;
                    break;
          case 0x1:  DSP_ALU_SRC2  = Y1;
                    break;
          case 0x2:  DSP_ALU_SRC2  = M0;
                    break;
          case 0x3:  DSP_ALU_SRC2  = M1;
                    break;
     }
     if(DSP_ALU_SRC2_MSB)  DSP_ALU_SRC2G = 0xff;
     else  DSP_ALU_SRC2G = 0x0;
}

DSP_ALU_DST = DSP_ALU_SRC1 + DSP_ALU_SRC2;
carry_bit = ((DSP_ALU_SRC1_MSB | DSP_ALU_SRC2_MSB) & !DSP_ALU_DST_MSB) |
            (DSP_ALU_SRC1_MSB & DSP_ALU_SRC2_MSB);
DSP_ALU_DSTG_LSB8 = DSP_ALU_SRC1G_LSB8 + DSP_ALU_SRC2G_LSB8 + carry_bit;
overflow_bit = !(POS_NOT_OV || NEG_NOT_OV);
overrange_bit = PLUS_OP_G_OV;
```

```
#include "fixed_pt_overflow_protection.c"

     if(DSP_UNCONDITIONAL_UPDATE) { /* unconditional operation */
#include "fixed_pt_unconditional_update.c"
#include "fixed_pt_plus_dc_bit.c"
     }
     else if(DSP_CONDITION_MATCH) { /* conditional operation and match */
          DSP_REG[ex2_dz_no] = DSP_ALU_DST;
          If(ex2_dz_no==0) {
               A0G = DSP_ALU_DSTG & MASK000000FF;
               if(DSP_ALU_DSTG_BIT7) A0G = A0G | MASKFFFFFF00;
          }
          else if(ex2_dz_no==1) {
               A1G = DSP_ALU_DSTG & MASK000000FF;
               If(DSP_ALU_DSTG_BIT7) A1G = A1G | MASKFFFFFF00;
          }
     }
}

break;
```

(4) 使用例

```
PCOPY X0,A0  NOPX  NOPY        ;実行前:X0=H'55555555, A0=H'FFFFFFFFFF
                                実行後:X0=H'55555555, A0=H'0055555555
                                無条件実行の場合、DC ビットは CS[2:0]の状態に従って更新。
```

11.4.10 [if cc] PDEC DECrement by 1 DSP 算術演算命令

条件付きデクリメント

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PDEC Sx,Dz	Sx の MSW－1→Dz の MSW、 Dz の LSW クリア	111110********** 10001001xx00zzzz	1	更新	更新
PDEC Sy,Dz	Sy の MSW－1→Dz の MSW、 Dz の LSW クリア	111110********** 1010100100yyzzzz	1	更新	更新
DCT PDEC Sx,Dz	もし DC＝1 ならば Sx の MSW－1→Dz の MSW、 Dz の LSW クリア もし DC＝0 ならば nop.	111110********** 10001010xx00zzzz	1	－	－
DCT PDEC Sy,Dz	もし DC＝1 ならば Sy の MSW－1→Dz の MSW、 Dz の LSW クリア もし DC＝0 ならば nop.	111110********** 1010101000yyzzzz	1	－	－
DCF PDEC Sx,Dz	もし DC＝0 ならば Sx の MSW－1→Dz の MSW、 Dz の LSW クリア もし DC＝1 ならば nop.	111110********** 10001011xx00zzzz	1	－	－
DCF PDEC Sy,Dz	もし DC＝0 ならば Sy の MSW－1→Dz の MSW、 Dz の LSW クリア もし DC＝1 ならば nop.	111110********** 1010101100yyzzzz	1	－	－

(1) 説明

Sx、Sy オペランドの上位ワードの内容から 1 を減算し、その結果を Dz オペランドの上位ワードへ格納し、Dz オペランドの下位ワードを 0 でクリアします。DCT、DCF の条件が指定されている場合は、条件が真のとき命令が実行されます。条件が偽のとき命令は実行されません。

条件が指定されていない場合は DSR レジスタの DC ビットは CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

条件が指定されている場合は、条件が真であっても、DC、N、Z、V、GT ビットは更新されません。

(2) 注意

デスティネーションレジスタの下位ワードの内容は DC ビットの更新には無視されます。

(3) 動作内容

```
{
    unsigned short int  ex2_dz_no, carry_bit, borrow_bit, overflow_bit,
                        overrange_bit;
     DSP_ALU_SRC2 = 0x1;
     DSP_ALU_SRC2G= 0x0;
     if(EX2_DSP_BIT13==0) {  /* MSW of Sx -1 -> Dz */
          switch (EX2_SX) {
               case 0x0: DSP_ALU_SRC1  = X0;
                         if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                         else  DSP_ALU_SRC1G = 0x0;
                         break;
               case 0x1: DSP_ALU_SRC1  = X1;
                         if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                         else  DSP_ALU_SRC1G = 0x0;
                         break;
               case 0x2: DSP_ALU_SRC1  = A0;
                         DSP_ALU_SRC1G = A0G;
                         break;
               case 0x3: DSP_ALU_SRC1  = A1;
                         DSP_ALU_SRC1G = A1G;
                         break;
          }
     }
     else {   /* MSW of Sy -1 -> Dz */
          switch (EX2_SY) {
               case 0x0:  DSP_ALU_SRC1  = Y0;
                          break;
               case 0x1:  DSP_ALU_SRC1  = Y1;
                          break;
               case 0x2:  DSP_ALU_SRC1  = M0;
                          break;
               case 0x3:  DSP_ALU_SRC1  = M1;
                          break;
          }

          if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
          else   DSP_ALU_SRC1G = 0x0;
```

```
}

    DSP_ALU_DST_HW = DSP_ALU_SRC1_HW - 1;
    carry_bit = ((DSP_ALU_SRC1_MSB | !DSP_ALU_SRC2_MSB) && !DSP_ALU_DST_MSB) |
                (DSP_ALU_SRC1_MSB & !DSP_ALU_SRC2_MSB);
    borrow_bit = !carry_bit;
    DSP_ALU_DSTG_LSB8 = DSP_ALU_SRC1G_LSB8 - DSP_ALU_SRC2G_LSB8 - borrow_bit;
    overflow_bit = !(POS_NOT_OV || NEG_NOT_OV);
    overrange_bit = MINUS_OP_G_OV;
#include "integer_overflow_protection.c"

    if(DSP_UNCONDITIONAL_UPDATE) { /* unconditional operation */
#include "integer_unconditional_update.c"
#include "integer_minus_dc_bit.c"
    }
    else if(DSP_CONDITION_MATCH) { /* conditional operation and match */
              DSP_REG_WD[ex2_dz_no*2] = DSP_ALU_DST_HW;
              DSP_REG_WD[ex2_dz_no*2+1] = 0x0;    /* clear LSW */
              if(ex2_dz_no==0) {
                    A0G = DSP_ALU_DSTG & MASK000000FF;
                    If(DSP_ALU_DSTG_BIT7) A0G = A0G | MASKFFFFFF00;
              }
              else  if(ex2_dz_no==1) {
                    A1G = DSP_ALU_DSTG & MASK000000FF;
                    If(DSP_ALU_DSTG_BIT7) A1G = A1G | MASKFFFFFF00;
              }
         }
    }
}

break;
```

(4) 使用例

```
PDEC X0,M0  NOPX  NOPY          ;実行前: X0=H'0052330F, M0=H'12345678
                                 実行後: X0=H'0052330F, M0=H'00510000
PDEC X1,X1  NOPX  NOPY          ;実行前: X1=H'FC342855
                                 実行後: X1=H'FC330000
                                 無条件実行の場合、DC ビットは CS[2:0]の状態に従って更新。
```

11.4.11 [if cc] PDMSB Detect MSB with Condition DSP 算術演算命令

条件付き MSB 検出

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PDMSB Sx,Dz	Sx の MSB 位置→Dz の MSW、Dz の LSW クリア	111110********** 10011101xx00zzzz	1	更新	更新
PDMSB Sy,Dz	Sy の MSB 位置→Dz の MSW、Dz の LSW クリア	111110********** 1011110100yyzzzz	1	更新	更新
DCT PDMSB Sx,Dz	もし DC＝1 ならば Sx の MSB 位置→Dz の MSW、Dz の LSW クリア もし DC＝0 ならば nop	111110********** 10011110xx00zzzz	1	−	−
DCT PDMSB Sy,Dz	もし DC＝1 ならば Sy の MSB 位置→Dz の MSW、Dz の LSW クリア もし DC＝0 ならば nop	111110********** 1011111000yyzzzz	1	−	−
DCF PDMSB Sx,Dz	もし DC＝0 ならば Sx の MSB 位置→Dz の MSW、Dz の LSW クリア もし DC＝1 ならば nop	111110********** 10011111xx00zzzz	1	−	−
DCF PDMSB Sy,Dz	もし DC＝0 ならば Sy の MSB 位置→Dz の MSW、Dz の LSW クリア もし DC＝1 ならば nop.	111110********** 1011111100yyzzzz	1	−	−

(1) 説明

Sx、Sy オペランドのビットの並びが最初に変わる位置を探し、そのビット位置を Dz オペランドへ格納します。DCT、DCF の条件が指定されている場合は、条件が真のとき命令が実行されます。条件が偽のとき命令は実行されません。

条件が指定されていない場合は DSR レジスタの DC ビットは CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

条件が指定されている場合は、条件が真であっても、DC、N、Z、V、GT ビットは更新されません。

(2) 注意

特にありません。

(3) 動作内容

```
{
      unsigned short int    ex2_dz_no, carry_bit, overflow_bit, overrange_bit;
      DSP_ALU_SRC2 = 0x0;
      DSP_ALU_SRC2G= 0x0;
      if(EX2_DSP_BIT13==0) {                    /*  msb(Sx) -> Dz */
           switch (EX2_SX) {
                case 0x0:  DSP_ALU_SRC1  = X0;
                           if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                           else  DSP_ALU_SRC1G = 0x0;
                           break;
                case 0x1:  DSP_ALU_SRC1 = X1;
                           if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                           else  DSP_ALU_SRC1G = 0x0;
                           break;
                case 0x2:  DSP_ALU_SRC1 = A0;
                           DSP_ALU_SRC1G = A0G;
                           break;
                case 0x3:  DSP_ALU_SRC1 = A1;
                           DSP_ALU_SRC1G = A1G;
                           break;
           }
      }
      else {    /* msb(Sy) -> Dz */
           switch (EX2_SY) {
                case 0x0:  DSP_ALU_SRC1 = Y0;
                           break;
                case 0x1:  DSP_ALU_SRC1 = Y1;
                           break;
                case 0x2:  DSP_ALU_SRC1 = M0;
                           break;
                case 0x3:  DSP_ALU_SRC1  = M1;
                           break;
           }
           if(DSP_ALU_SRC1_MSB)  DSP_ALU_SRC1G = 0xff;
```

```
            else  DSP_ALU_SRC1G = 0x0;
      }
      {
      short int i;
      unsigned char msb, src1g;
      unsigned long src1 = DSP_ALU_SRC1;
      msb = DSP_ALU_SRC1G_BIT7;
      src1g = (DSP_ALU_SRC1G_LSB8 << 1);
      for (i=38; ((msb == (src1g >> 7)) && (i>= 32));i--) { src1g <<= 1; }
      if(i==31)  {
            for (i; ((msb == (src1 >> 31)) && (i>=0));i--) { src1 <<= 1; }
      }

            DSP_ALU_DST = 0x0;
            DSP_ALU_DST_HW = (short int) (30-i);
            if(DSP_ALU_DST_MSB)  DSP_ALU_DSTG_LSB8 = 0xff;
            else  DSP_ALU_DSTG_LSB8 = 0x0;
      }

      carry_bit = 0;

      if(DSP_UNCONDITIONAL_UPDATE) { /* unconditional operation */
      overflow_bit = 0;
      overrange_bit = 0;
#include "integer_unconditional_update.c"
#include "integer_plus_dc_bit.c"
      }
      else if(DSP_CONDITION_MATCH) { /* conditional operation and match */
            DSP_REG_WD[ex2_dz_no*2] = DSP_ALU_DST_HW;
            DSP_REG_WD[ex2_dz_no*2+1] = 0x0;       /* clear LSW */
            If(ex2_dz_no==0) {
                  A0G = DSP_ALU_DSTG & MASK000000FF;
                  If(DSP_ALU_DSTG_BIT7) A0G = A0G | MASKFFFFFF00;
            }
            else if(ex2_dz_no==1) {
                  A1G = DSP_ALU_DSTG & MASK000000FF;
                  If(DSP_ALU_DSTG_BIT7) A1G = A1G | MASKFFFFFF00;
            }
```

```
        }
    }

    break;
```

(4) 使用例

```
PDMSB X0,M0  NOPX  NOPY        ;実行前：X0=H'0052330F, M0=H'12345678
                                実行後：X0=H'0052330F, M0=H'00080000
PDMSB X1,X1  NOPX  NOPY        ;実行前：X1=H'FC342855
                                実行後：X1=H'00050000
                                無条件実行の場合、DC ビットは CS[2:0]の状態に従って更新。
```

11.4.12 [if cc] PINC INCrement by 1 with Condition DSP 算術演算命令

条件付きインクリメント

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PINC Sx,Dz	Sx の MSW＋1→Dz の MSW、Dz の LSW クリア	111110********** 10011001xx00zzzz	1	更新	更新
PINC Sy,Dz	Sy の MSW＋1→Dz の MSW、Dz の LSW クリア	111110********** 1011100100yyzzzz	1	更新	更新
DCT PINC Sx,Dz	もし DC＝1 ならば Sx の MSW＋1→Dz の MSW、Dz の LSW クリア もし DC＝0 ならば nop.	111110********** 10011010xx00zzzz	1	−	−
DCT PINC Sy,Dz	もし DC＝1 ならば Sy の MSW＋1→Dz の MSW、Dz の LSW クリア もし DC＝0 ならば nop.	111110********** 1011101000yyzzzz	1	−	−
DCF PINC Sx,Dz	もし DC＝0 ならば Sx の MSW＋1→Dz の MSW、Dz の LSW クリア もし DC＝1 ならば nop.	111110********** 10011011xx00zzzz	1	−	−
DCF PINC Sy,Dz	もし DC＝0 ならば Sy の MSW＋1→Dz の MSW、Dz の LSW クリア もし DC＝1 ならば nop.	111110********** 1011101100yyzzzz	1	−	−

(1) 説明

Sx、Sy オペランドの上位ワードの内容に 1 を加算し、その結果を Dz オペランドの上位ワードへ格納し、Dz オペランドの下位ワードを 0 でクリアします。DCT、DCF の条件が指定されている場合は、条件が真のとき命令が実行されます。条件が偽のとき命令は実行されません。

条件が指定されていない場合は DSR レジスタの DC ビットは CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

条件が指定されている場合は、条件が真であっても、DC、N、Z、V、GT ビットは更新されません。

(2) 注意

デスティネーションレジスタの下位ワードの内容は DC ビットの更新には無視されます。

(3) 動作内容

```
{
     unsigned short int     ex2_dz_no, carry_bit, overflow_bit, overrange_bit;
     DSP_ALU_SRC2 = 0x1;
     DSP_ALU_SRC2G= 0x0;
     if(EX2_DSP_BIT13==0) {/* MSW of Sx +1 -> Dz */
          switch (EX2_SX) {
               case 0x0: DSP_ALU_SRC1= X0;
                         if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                         elseDSP_ALU_SRC1G = 0x0;
                         break;
               case 0x1: DSP_ALU_SRC1 = X1;
                         if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                         else DSP_ALU_SRC1G = 0x0;
                         break;
               case 0x2: DSP_ALU_SRC1 = A0;
                         DSP_ALU_SRC1G = A0G;
                         break;
               case 0x3: DSP_ALU_SRC1 = A1;
                         DSP_ALU_SRC1G = A1G;
                         break;
          }
     }
     else {     /* MSW of Sy +1 -> Dz */
          switch (EX2_SY) {
               case 0x0: DSP_ALU_SRC1  = Y0;
                         break;
               case 0x1: DSP_ALU_SRC1  = Y1;
                         break;
               case 0x2: DSP_ALU_SRC1  = M0;
                         break;
               case 0x3: DSP_ALU_SRC1  = M1;
                         break;
            }
            if(DSP_ALU_SRC1_MSB)DSP_ALU_SRC1G = 0xff;
            else  DSP_ALU_SRC1G = 0x0;
     }
```

```
      DSP_ALU_DST_HW = DSP_ALU_SRC1_HW + 1;
      carry_bit = ((DSP_ALU_SRC1_MSB | DSP_ALU_SRC2_MSB) & !DSP_ALU_DST_MSB) |
                  (DSP_ALU_SRC1_MSB & DSP_ALU_SRC2_MSB);
      DSP_ALU_DSTG_LSB8 = DSP_ALU_SRC1G_LSB8 + DSP_ALU_SRC2G_LSB8 + carry_bit;
      overflow_bit = !(POS_NOT_OV || NEG_NOT_OV);
      overrange_bit = PLUS_OP_G_OV;
#include "integer_overflow_protection.c"
      if(DSP_UNCONDITIONAL_UPDATE) { /* unconditional operation */
#include "integer_unconditional_update.c"
#include "integer_plus_dc_bit.c"
      }
      else if(DSP_CONDITION_MATCH) { /* conditional operation and match */
            DSP_REG_WD[ex2_dz_no*2] = DSP_ALU_DST_HW;
            DSP_REG_WD[ex2_dz_no*2+1] = 0x0;       /* clear LSW */
            If(ex2_dz_no==0) {
                  A0G = DSP_ALU_DSTG & MASK000000FF;
                  If(DSP_ALU_DSTG_BIT7) A0G = A0G | MASKFFFFFF00;
            }
            else if(ex2_dz_no==1) {
                  A1G = DSP_ALU_DSTG & MASK000000FF;
                  If(DSP_ALU_DSTG_BIT7) A1G = A1G | MASKFFFFFF00;
            }
      }
}

break;
```

(4) 使用例

```
PINC X0,M0  NOPX  NOPY  ;
PINC X1,X1  NOPX  NOPY  ;
```

実行前：X0=H'0052330F, M0=H'12345678
実行後：X0=H'0052330F, M0=H'00530000
実行前：X1=H'FC342855
実行後：X1=H'FC350000
無条件実行の場合、DC ビットは CS[2:0]の状態に従って更新。

11.4.13 [if cc] PLDS LoaD System register DSP システム制御命令

条件付き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	DC ビット	T ビット
PLDS Dz,MACH	Dz→MACH	111110**********	1	−	−
		111011010000zzzz			
PLDS Dz,MACL	Dz→MACL	111110**********	1	−	−
		111111010000zzzz			
DCT PLDS Dz,MACH	もし DC＝1 ならば Dz→MACH	111110**********	1	−	−
	もし DC＝0 ならば nop.	111011100000zzzz			
DCT PLDS Dz,MACL	もし DC＝1 ならば Dz→MACL	111110**********	1	−	−
	もし DC＝0 ならば nop.	111111100000zzzz			
DCF PLDS Dz,MACH	もし DC＝0 ならば Dz→MACH	111110**********	1	−	−
	もし DC＝1 ならば nop.	111011110000zzzz			
DCF PLDS Dz,MACL	もし DC＝0 ならば Dz→MACL	111110**********	1	−	−
	もし DC＝ならば nop.	111111110000zzzz			

(1) 説明

Dz オペランドの内容を、MACH、MACL レジスタへ格納します。DCT、DCF の条件が指定されている場合は、条件が真のとき命令が実行されます。条件が偽のとき命令は実行されません。

DSR レジスタの DC、N、Z、V、GT、AN、AZ、AV、AGT ビットはいずれも更新されません。また、SR レジスタの T ビットも更新されません。

(2) 注意

PLDS と MOVX、MOVY は並列に指定できますが、実行には 2 サイクルかかる場合があります。

(3) 動作内容

```
{    /* Dz -> MACH */
     unsigned short int     ex2_dz_no;
     if(DSP_UNCONDITIONAL_UPDATE) { /* unconditional operation */
          MACH = DSP_REG[ex2_dz_no] ;
     }
     else if(DSP_CONDITION_MATCH) { /* conditional operation and match */
          MACH = DSP_REG[ex2_dz_no] ;
     }
}
```

```
break;

/*   SH-DSP: DSP Engine:  Local Data Move Operation: LoaD System register
     plds_macl.c
     rev 1.0 24 May 1995, EY  */

{    /* Dz -> MACL */
     if(DSP_UNCONDITIONAL_UPDATE) { /* unconditional operation */
          MACL = DSP_REG[ex2_dz_no] ;
     }
     else if(DSP_CONDITION_MATCH) { /* conditional operation and match */
          MACL = DSP_REG[ex2_dz_no] ;
     }
}

break;
```

(4) 使用例

```
PLDS A0,MACH  NOPX  NOPY       ;実行前: A0=H'123456789A,
                                        MACH=H'66666666
                                実行後: A0=H'123456789A,
                                        MACH=H'3456789A
```

11.4.14 PMULS MULtiply as Signed DSP 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	DC ビット	T ビット
PMULS Se,Sf,Dg	Se の MSW×Sf の MSW→Dg	111110********** 0100eeff0000gg00	1	−	−

(1) 説明

Se、Sf オペランドの上位ワードの内容を符号付きとして乗算し、結果を Dg オペランドに格納します。

DSR レジスタの DC、N、Z、V、GT、AN、AZ、AV、AGT ビットはいずれも更新されません。また、SR レジスタの T ビットも更新されません。

(2) 注意

PMULS は固定小数点乗算ですので、ソースデータが同じでも MULS とは演算結果が異なります。

(3) 動作内容

```
{
     switch(EX2_SE){
          case 0x0:  DSP_MLT_SRC1 = X0;
                     break;
          case 0x1:  DSP_MLT_SRC1 = X1;
                     break;
          case 0x2:  DSP_MLT_SRC1 = Y0;
                     break;
          case 0x3:  DSP_MLT_SRC1 = A1;
                     break;
}
     switch(EX2_SF){
          case 0x0:  DSP_MLT_SRC2 = Y0;
                     break;
          case 0x1:  DSP_MLT_SRC2 = Y1;
                     break;
          case 0x2:  DSP_MLT_SRC2 = X0;
                     break;
          case 0x3:  DSP_MLT_SRC2 = A1;
                     break;
```

```
}

/* Multiplier Operation */
     if((SBIT==1) && (DSP_MLT_SRC1_HW==0x8000) && (DSP_MLT_SRC2_HW===0x8000)
     {
          DSP_MLT_DST = 0x7fffffff;             /* overflow protection */
     }
     else{
     DSP_MLT_DST = ((long)(short) DSP_MLT_SRC1_HW
                  * (long)(short) DSP_MLT_SRC2_HW) << 1;
     }
     if(DSP_MLT_DST_MSB) DSP_MLT_DSTG_LSB8 = 0xff;
     else DSP_MLT_DSTG_LSB8 = 0x00;

/* Multiplier Destination assignment */
     switch (EX2_DG) {
          case 0x0:  M0 = DSP_MLT_DST;
                     break;
          case 0x1:  M1 = DSP_MLT_DST;
                     break;
          case 0x2:  A0 = DSP_MLT_DST;
                     if(DSP_MLT_DSTG_LSB8 = 0x0) A0G = 0x0;
                     else  A0G = 0xffffffff;
                     break;
          case 0x3:  A1 = DSP_MLT_DST;
                     if(DSP_MLT_DSTG_LSB8 = 0x0) A1G = 0x00;
                     else  A0G = 0xffffffff;
                     break;
     }

}
break;
```

(4) 使用例

```
PMULS X0,Y0,M0  NOPX  NOPY  ;  実行前:X0=H'00010000,Y0=H'00020000,
                                      M0=H'33333333
                               実行後:X0=H'00010000,Y0=H'00020000,
                                      M0=H'00000004
PMULS X1,Y1,A0  NOPX  NOPY  ;  実行前:X1=H'FFFE2222,Y1=H'0001AAAA,
                                      A0=H'4444444444
                               実行後:X1=H'FFFE2222,Y1=H'0001AAAA,
                                      A0=H'FFFFFFFFFC
```

11.4.15 [if cc] PNEG NEGate DSP 算術演算命令

条件付き符号反転

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PNEG Sx,Dz	0－Sx→Dz	111110**********	1	更新	更新
		11001001xx00zzzz			
PNEG Sy,Dz	0－Sy→Dz	111110**********	1	更新	更新
		1110100100yyzzzz			
DCT PNEG Sx,Dz	もし DC＝1 ならば 0－Sx→Dz	111110**********	1	－	－
	もし DC＝0 ならば nop.	11001010xx00zzzz			
DCT PNEG Sy,Dz	もし DC＝1 ならば 0－Sy→Dz	111110**********	1	－	－
	もし DC＝0 ならば nop.	1110101000yyzzzz			
DCF PNEG Sx,Dz	もし DC＝0 ならば 0－Sx→Dz	111110**********	1	－	－
	もし DC＝1 ならば nop.	11001011xx00zzzz			
DCF PNEG Sy,Dz	もし DC＝0 ならば 0－Sy→Dz	111110**********	1	－	－
	もし DC＝1 ならば nop.	1110101100yyzzzz			

(1) 説明

符号を反転します。0 から Sx、Sy オペランドの内容を減算して Dz オペランドへ格納します。DCT、DCF の条件が指定されている場合は、条件が真のとき命令が実行されます。条件が偽のとき命令は実行されません。

条件が指定されていない場合は DSR レジスタの DC ビットは CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

条件が指定されている場合は、条件が真であっても、DC、N、Z、V、GT ビットは更新されません。

(2) 注意

特にありません。

(3) 動作内容

```
{
    unsigned short int ex2_dz_no, carry_bit, borrow_bit, overflow_bit,
                       overrange_bit;
      DSP_ALU_SRC1 = 0;
      DSP_ALU_SRC1G= 0;
      if(EX2_DSP_BIT13==0) { /* 0 - Sx -> Dz */
```

```
        switch (EX2_SX) {
            case 0x0: DSP_ALU_SRC2  = X0;
                      if(DSP_ALU_SRC2_MSB) DSP_ALU_SRC2G = 0xff;
                      else  DSP_ALU_SRC2G = 0x0;
                      break;
            case 0x1: DSP_ALU_SRC2  = X1;
                      if(DSP_ALU_SRC2_MSB) DSP_ALU_SRC2G = 0xff;
                      else  DSP_ALU_SRC2G = 0x0;
                      break;
            case 0x2: DSP_ALU_SRC2  = A0;
                      DSP_ALU_SRC2G = A0G;
                      break;
            case 0x3: DSP_ALU_SRC2  = A1;
                      DSP_ALU_SRC2G = A1G;
                      break;
        }
    }
    else  {             /* 0 - Sy -> Dz */
        switch (EX2_SY) {
            case 0x0: DSP_ALU_SRC2  = Y0;
                      break;
            case 0x1: DSP_ALU_SRC2  = Y1;
                      break;
            case 0x2: DSP_ALU_SRC2  = M0;
                      break;
            case 0x3: DSP_ALU_SRC2  = M1;
                      break;
        }
        if(DSP_ALU_SRC2_MSB) DSP_ALU_SRC2G = 0xff;
        else  DSP_ALU_SRC2G = 0x0;
    }

    DSP_ALU_DST = DSP_ALU_SRC1 - DSP_ALU_SRC2;
    carry_bit = ((DSP_ALU_SRC1_MSB | !DSP_ALU_SRC2_MSB) && !DSP_ALU_DST_MSB) |
                DSP_ALU_SRC1_MSB & !DSP_ALU_SRC2_MSB);
    borrow_bit = !carry_bit;
    DSP_ALU_DSTG_LSB8 = DSP_ALU_SRC1G_LSB8 - DSP_ALU_SRC2G_LSB8 - borrow_bit;
    overflow_bit= !(POS_NOT_OV || NEG_NOT_OV);
```

```
      overrange_bit = MINUS_OP_G_OV;
#include "fixed_pt_overflow_protection.c"

      if(DSP_UNCONDITIONAL_UPDATE) { /* unconditional operation */
#include "fixed_pt_unconditional_update.c"
#include "fixed_pt_minus_dc_bit.c"
      }
      else if(DSP_CONDITION_MATCH) { /* conditional operation and match */
            DSP_REG[ex2_dz_no] = DSP_ALU_DST;
            If(ex2_dz_no==0) {
                  A0G = DSP_ALU_DSTG & MASK000000FF;
                  If(DSP_ALU_DSTG_BIT7) A0G = A0G | MASKFFFFFF00;
            }
            else   if(ex2_dz_no==1) {
                  A1G = DSP_ALU_DSTG & MASK000000FF;
                  If(DSP_ALU_DSTG_BIT7) A1G = A1G | MASKFFFFFF00;
            }
      }
}

break;
```

(4) 使用例

```
PNEG X0,A0  NOPX  NOPY        ;実行前: X0=H'55555555,A0=H'A987654321
                               実行後: X0=H'55555555,A0=H'FFAAAAAAAB
PNEG Y1,Y1  NOPX  NOPY        ;実行前: Y1=H'99999999
                               実行後: Y1=H'66666667
                               無条件実行の場合、DC ビットは CS[2:0]の状態に従って更新。
```

11.4.16 [if cc] POR　　logical OR　　DSP 論理演算命令

条件付き論理和演算

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
POR Sx,Sy,Dz	Sx | Sy→Dz、Dz のガードビットと LSW クリア	111110********** 10110101xxyyzzzz	1	更新	更新
DCT POR Sx,Sy,Dz	もし DC＝1 ならば Sx | Sy→Dz、Dz のガードビットと LSW クリア もし 0 ならば nop.	111110********** 10110110xxyyzzzz	1	−	−
DCF POR Sx,Sy,Dz	もし DC＝0 ならば Sx | Sy→Dz、Dz のガードビットと LSW クリア もし 1 ならば nop.	111110********** 10110111xxyyzzzz	1	−	−

(1) 説明

Sx オペランドの上位ワードの内容と Sy オペランドの上位ワードの内容との論理和を演算し、その結果を Dz オペランドの上位ワードへ格納し、Dz オペランドの下位ワードを 0 でクリアします。Dz オペランドがガードビットを持つレジスタの場合は、ガードビットも 0 でクリアします。DCT、DCF の条件が指定されている場合は、条件が真のとき命令が実行されます。条件が偽のとき命令は実行されません。

条件が指定されていない場合は DSR レジスタの DC ビットは CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

条件が指定されている場合は、条件が真であっても、DC、N、Z、V、GT ビットは更新されません。

(2) 注意

デスティネーションレジスタの下位ワードの内容とガードビットの内容は DC ビットの更新には無視されます。

(3) 動作内容

```
{
    unsigned short int ex2_dz_no, carry_bit,negative_bit, zero_bit, overflow_bit,
overrange_bit;
     switch (EX2_SX) {
          case 0x0:  DSP_ALU_SRC1  = X0;
                     break;
          case 0x1:  DSP_ALU_SRC1  = X1;
                     break;
```

```
            case 0x2:   DSP_ALU_SRC1  = A0;
                        break;
            case 0x3:   DSP_ALU_SRC1  = A1;
                        break;
        }

        switch (EX2_SY) {
            case 0x0:   DSP_ALU_SRC2  = Y0;
                        break;
            case 0x1:   DSP_ALU_SRC2  = Y1;
                        break;
            case 0x2:   DSP_ALU_SRC2  = M0;
                        break;
            case 0x3:   DSP_ALU_SRC2  = M1;
                        break;
        }

        DSP_ALU_DST_HW = DSP_ALU_SRC1_HW | DSP_ALU_SRC2_HW;

        if(DSP_UNCONDITIONAL_UPDATE) { /* unconditional operation */
            DSP_REG_WD[ex2_dz_no*2] = DSP_ALU_DST_HW;
            DSP_REG_WD[ex2_dz_no*2+1] = 0x0;      /* clear LSW */
            if(ex2_dz_no==0)        A0G = 0x0;     /* clear Guard bits */
            else if(ex2_dz_no==1)  A1G = 0x0;
            carry_bit = 0x0;
            negative_bit = DSP_ALU_DST_MSB;
            zero_bit = (DSP_ALU_DST_HW==0);
            overflow_bit = 0x0;
            overrange_bit = 0x0;
#include "logical_dc_bit.c"
        }
        else if(DSP_CONDITION_MATCH) { /* conditional operation and match */
            DSP_REG_WD[ex2_dz_no*2] = DSP_ALU_DST_HW;
            DSP_REG_WD[ex2_dz_no*2+1] = 0x0;  /* clear LSW */
            if(ex2_dz_no==0)A0G = 0x0;       /* clear Guard bits */
            else if(ex2_dz_no==1)  A1G = 0x0;
        }
}
```

```
break;
```

(4) 使用例

```
POR X0,Y0,A0  NOPX  NOPY       ;実行前:X0=H'33333333, Y0=H'55555555
                                        A0=H'123456789A
                                 実行後:X0=H'33333333, Y0=H'55555555
                                        A0=H'0077770000
```

無条件実行の場合、DC ビットは CS[2:0]の状態に従って更新。

11.4.17 [if cc] PRND RouNDing DSP 算術演算命令

丸め演算

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PRND Sx,Dz	Sx＋H'00008000→Dz	111110**********	1	更新	更新
	Dz の LSW クリア	10011000xx00zzzz			
PRND Sy,Dz	Sy＋H'00008000→Dz	111110**********	1	更新	更新
	Dz の LSW クリア	1011100000yyzzzz			
DCT PRND Sx,Dz	もし DC＝1 ならば	111110**********	1	−	−
	Sx＋H'00008000→Dz	10011010xx01zzzz			
	Dz の LSW クリア				
	もし DC＝0 ならば nop.				
DCT PRND Sy,Dz	もし DC＝1 ならば	111110**********	1	−	−
	Sy＋H'00008000→Dz	1011101001yyzzzz			
	Dz の LSW クリア				
	もし DC＝0 ならば nop.				
DCF PRND Sx,Dz	もし DC＝0 ならば	111110**********	1	−	−
	Sx＋H'00008000→Dz	10011011xx01zzzz			
	Dz の LSW クリア				
	もし DC＝1 ならば nop.				
DCF PRND Sy,Dz	もし DC＝0 ならば	111110**********	1	−	−
	Sy＋H'00008000→Dz	1011101101yyzzzz			
	Dz の LSW クリア				
	もし DC＝1 ならば nop.				

(1) 説明

丸めを行います。Sx、Sy オペランドの内容にイミディエイトデータ H'00008000 を加算し、結果の上位ワードを Dz オペランドへ格納し、Dz オペランドの下位ワードを 0 でクリアします。

条件が指定されていない場合は、DSR レジスタの DC ビットは CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

条件が指定されている場合は、条件が真であっても、DC、N、Z、V、GT、AN、AZ、AV、AGT ビットは更新されません。SR レジスタの T ビットも更新されません。

(2) 注意

特にありません。

(3) 動作内容

```
{
     unsigned short int  ex2_dz_no, carry_bit, negative_bit, zero_bit, overflow_bit,
                         overrange_bit;
     DSP_ALU_SRC2 = 0x00008000;
     DSP_ALU_SRC2G= 0x0;
     if(EX2_DSP_BIT13==0) {       /* Sx + H'00008000 -> Dz; clr Dz LW */
          switch (EX2_SX) {
               case 0x0: DSP_ALU_SRC1  = X0;
                         if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                         else  DSP_ALU_SRC1G = 0x0;
                         break;
               case 0x1: DSP_ALU_SRC1  = X1;
                         if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                         else  DSP_ALU_SRC1G = 0x0;
                         break;
               case 0x2: DSP_ALU_SRC1  = A0;
                         DSP_ALU_SRC1G = A0G;
                         break;
               case 0x3: DSP_ALU_SRC1  = A1;
                         DSP_ALU_SRC1G = A1G;
                         break;
          }
     }

     else  {             /* Sy + H'00008000 -> Dz; clr Dz LW */
          switch (EX2_SY) {
               case 0x0: DSP_ALU_SRC1  = Y0;
                         break;
               case 0x1: DSP_ALU_SRC1  = Y1;
                         break;
               case 0x2: DSP_ALU_SRC1  = M0;
                         break;
               case 0x3: DSP_ALU_SRC1  = M1;
                         break;
```

```
          }
     if(DSP_ALU_SRC1_MSB)  DSP_ALU_SRC1G = 0xff;
     else        DSP_ALU_SRC1G = 0x0;
     }

     DSP_ALU_DST = (DSP_ALU_SRC1 + DSP_ALU_SRC2) & MASKFFFF0000;
     carry_bit = ((DSP_ALU_SRC1_MSB | DSP_ALU_SRC2_MSB) &
               !DSP_ALU_DST_MSB) | (DSP_ALU_SRC1_MSB & DSP_ALU_SRC2_MSB);
     DSP_ALU_DSTG_LSB8 = DSP_ALU_SRC1G_LSB8 + DSP_ALU_SRC2G_LSB8 + carry_bit;
     overflow_bit= !(POS_NOT_OV || NEG_NOT_OV);
     overrange_bit = PLUS_OP_G_OV;
#include "fixed_pt_overflow_protection.c"

     if(DSP_UNCONDITIONAL_UPDATE) {  /* inconditional operation * /
#include "fixed_pt_unconditional_update.c"
#include "fixed_pt_plus_dc_bit.c"
}
     else if(DSP_CONDITION_MATCH) {  /* conditional operation and match * /
          DSP_REG[ex2_dz_no] = DSP_ALU DST;
          If(ex2_dz_no==0) {
               A0G = DSP_ALU_DSTG & MASK000000FF;
               If(DSP_ALU_DSTG_BIT7)  A0G = A0G | MASKFFFFFF00;
          }
          else if(ex2_dz_no==1){
               A1G = DSP_ALU_DSTG & MASK000000FF;
               If(DSP_ALU_DSTG_BIT7)  A1G = A1G | MASKFFFFFF00;
          }
     }
}
break;
```

(4) 使用例

```
PRND X0,M0  NOPX  NOPY  ;
PRND X1,X1  NOPX  NOPY  ;
```

実行前:X0=H'0052330F, M0=H'12345678
実行後:X0=H'0052330F, M0=H'00520000
実行前:X1=H'FC34C087
実行後:X1=H'FC350000
DC ビットは CS[2:0]の状態に従って更新。

11.4.18 [if cc] PSHA SHift Arithmetically with Condition DSP 算術シフト命令

条件付き算術シフト

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PSHA Sx,Sy,Dz	もし Sy≧0 ならば Sx＜＜Sy→Dz もし Sy＜0 ならば Sx＞＞｜Sy｜→Dz	111110********** 10010001xxyyzzzz	1	更新	更新
DCT PSHA Sx,Sy,Dz	もし DC＝1 & Sy≧0 ならば Sx＜＜Sy→Dz もし DC＝1 & Sy＜0 ならば Sx＞＞｜Sy｜→Dz もし DC＝0 ならば nop	111110********** 10010010xxyyzzzz	1	−	−
DCF PSHA Sx,Sy,Dz	もし DC＝0 & Sy≧0 ならば Sx＜＜Sy→Dz もし DC＝0 & Sy＜0 ならば Sx＞＞｜Sy｜→Dz もし DC＝1 ならば nop	111110********** 10010011xxyyzzzz	1	−	−
PSHA #Imm,Dz	もし Imm≧0 ならば Dz＜＜Imm→Dz もし Imm＜0 ならば Dz＞＞｜Imm｜→Dz	111110********** 00010iiiiiiizzzz	1	更新	更新

(1) 説明

Sx または Dz オペランドの内容を算術的にシフトし、その結果を Dz オペランドへ格納します。シフト量は Sy オペランドまたはイミディエイト値 Imm オペランドで指定します。シフト量が正の値のとき左にシフトします。負の値のとき右にシフトします。DCT、DCF の条件が指定されている場合は、条件が真のとき命令が実行されます。条件が偽のとき命令は実行されません。

条件が指定されていない場合は DSR レジスタの DC ビットは CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

条件が指定されている場合は、条件が真であっても、DC、N、Z、V、GT ビットは更新されません。

(2) 注意

特にありません。

(3) 動作内容

- レジスタオペランドによる場合

```
{
     unsigned short int     ex2_dz_no, carry_bit, overflow_bit, overrange_bit;
     switch (EX2_SX) {
          case 0x0:  DSP_ALU_SRC1 = X0;
                     if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                     else            DSP_ALU_SRC1G = 0x0;
                     break;
          case 0x1:  DSP_ALU_SRC1   = X1;
                     if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                     else  DSP_ALU_SRC1G = 0x0;
                     break;
          case 0x2:  DSP_ALU_SRC1   = A0;
                     DSP_ALU_SRC1G  = A0G;
                     break;
          case 0x3:  DSP_ALU_SRC1   = A1;
                     DSP_ALU_SRC1G  = A1G;
                     break;
     }
     switch (EX2_SY) {
          case 0x0:  DSP_ALU_SRC2   = Y0 & MASK007F0000;
                     break;
          case 0x1:  DSP_ALU_SRC2   = Y1 & MASK007F0000;
                     break;
          case 0x2:  DSP_ALU_SRC2   = M0 & MASK007F0000;
                     break;
          case 0x3:  DSP_ALU_SRC2   = M1 & MASK007F0000;
                     break;
     }
     if(DSP_ALU_SRC2_MSB)           DSP_ALU_SRC2G = 0xff;
     else                           DSP_ALU_SRC2G = 0x0;

     if((DSP_ALU_SRC2_HW & MASK0040) == 0) { /* Left Shift 0 <= cnt <= 32 */
          char cnt = (DSP_ALU_SRC2_HW & MASK003F);
```

```
            if(cnt > 32) {
               printf("¥nPSHA Sz,Sy,Dz Error! Shift %2X exceed range.¥n",cnt);
               exit();
            }
            DSP_ALU_DST  = DSP_ALU_SRC1 << cnt;
            DSP_ALU_DSTG = ((DSP_ALU_SRC1G << cnt) |
                           (DSP_ALU_SRC1 >> (32-cnt))) & MASK000000FF;
            carry_bit = DSP_ALU_DSTG & MASK00000001;
      }

      else  {                          /* Right Shift 0< cnt <=32 */
           char cnt = ((~DSP_ALU_SRC2_HW & MASK003F)+1);
           if(cnt > 32) {
                printf("¥nPSHA Sz,Sy,Dz Error! shift -%2X exceed range.¥n",cnt);
                exit();
           }
           if((cnt>8) && DSP_ALU_SRC1G_BIT7) { /* MSB copy */
                DSP_ALU_DST =((DSP_ALU_SRC1 >> 8) | (DSP_ALU_SRC1G << (32-8)));
                DSP_ALU_DST =(long) DSP_ALU_DST >> (cnt-8);
           }
           else {
                DSP_ALU_DST =((DSP_ALU_SRC1 >> cnt) | (DSP_ALU_SRC1G << (32-cnt)));
           }
           DSP_ALU_DSTG_LSB8 = (char) DSP_ALU_SRC1G_LSB8 >> cnt-- ;
           carry_bit = (DSP_ALU_SRC1 >> cnt) & MASK00000001;
      }

      overflow_bit = !(POS_NOT_OV || NEG_NOT_OV); /* do overflow detection */
      overrange_bit = 0x0;
#include "fixed_pt_overflow_protection.c" /* do overflow protection; V=0 */

      if(DSP_UNCONDITIONAL_UPDATE) { /* unconditional operation */
#include "fixed_pt_unconditional_update.c"
#include "shift_dc_bit.c"
      }
      else if(DSP_CONDITION_MATCH) { /* conditional operation and match */
           DSP_REG[ex2_dz_no] = DSP_ALU_DST;
           If(ex2_dz_no == 0) {
```

```
            A0G = DSP_ALU_DSTG & MASK000000FF;
            If(DSP_ALU_DSTG_BIT7) A0G = A0G | MASKFFFFFF00;
        }
        else if(ex2_dz_no == 1) {
            A1G = DSP_ALU_DSTG & MASK000000FF;
            If DSP_ALU_DSTG_BIT7) A1G = A1G | MASKFFFFFF00;
        }
    }
}

break;
```

- **イミディエイトオペランドによる場合**

```
{
    unsigned short int    ex2_dz_no, carry_bit, overflow_bit, overrange_bit;
    unsigned short tmp_imm;
    DSP_ALU_SRC1=DSP_REG[ex2_dz_no];
    switch (ex2_dz_no) {
        case 0x0:  DSP_ALU_SRC1G = A0G;
                   break;
        case 0x1:  DSP_ALU_SRC1G = A1G;
                   break;
        default:   if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                   else  DSP_ALU_SRC1G = 0x0;
    }

    tmp_imm = ((EX2_LW >> 4) & MASK0000007F); /* bit[10:4] */
    if((tmp_imm & MASK0040)==0) {  /* Left Shift 0 <= cnt <= 32 */
        char cnt = (tmp_imm & MASK003F);
        if(cnt > 32) {
            printf("¥nPSHA Dz,#Imm,Dz Error! #Imm=%7X exceed range¥n",tmp_imm);
            exit();
        }
        DSP_ALU_DST  = DSP_ALU_SRC1 << cnt;
        DSP_ALU_DSTG = ((DSP_ALU_SRC1G << cnt) |
                        (DSP_ALU_SRC1 >> (32-cnt))) & MASK000000FF;
        carry_bit = DSP_ALU_DSTG & MASK00000001;
    }
```

```
    else   {                          /* Right Shift 0< cnt <=32 */
         char cnt = ((~tmp_imm & MASK003F) + 1);
         if(cnt>32) {
              printf("¥nPSHL Dz,#Imm,Dz Error! #Imm = %7X exceed range¥n",tmp_imm);
              exit();
         }
         if((cnt>8) && DSP_ALU_SRC1G_BIT7) { /* MSB copy */
         DSP_ALU_DST = ((DSP_ALU_SRC1 >> 8) | (DSP_ALU_SRC1G << (32-8)));
         DSP_ALU_DST = (long) DSP_ALU_DST >> (cnt-8);
         }
         else {
           DSP_ALU_DST = ((DSP_ALU_SRC1 >> cnt) | (DSP_ALU_SRC1G << (32-cnt)));
         }
         DSP_ALU_DSTG_LSB8 = (char) DSP_ALU_SRC1G_LSB8 >>  cnt--;
         carry_bit = (((DSP_ALU_SRC1 >> cnt) & MASK00000001) == 0x1);
    }

    overflow_bit = !(POS_NOT_OV || NEG_NOT_OV); /* do overflow detection */
    overrange_bit = 0x0;
#include "fixed_pt_overflow_protection.c"  /* do overflow protection; V=0 */

    { /* unconditional operation */
#include "fixed_pt_unconditional_update.c"
#include "shift_dc_bit.c"
    }
}

break;
```

(4) 使用例

```
PSHA X0,Y0,A0  NOPX  NOPY      ;実行前:X0=H'88888888, Y0=H'00020000,
                                        A0=H'123456789A
                                実行後:X0=H'88888888, Y0=H'00020000,
                                        A0=H'FE22222222
PSHA X0,Y0,X0  NOPX  NOPY      ;実行前:X0=H'33333333, Y0=H'FFFF0000,
                                実行後:X0=H'19999999, Y0=H'FFFE0000,
PSHA #-5,A1  NOPX  NOPY        ;実行前:A1=H'AAAAAAAAAA
                                実行後:A1=H'FD55555555
                                無条件実行の場合、DC ビットはCS[2:0]の状態に従って更新。
```

11.4.19 [if cc] PSHL SHift Logically with condition DSP 論理シフト命令

条件付き論理シフト

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PSHL Sx,Sy,Dz	もし Sy≧0 ならば Sx<<Sy→Dz、Dz のガードビットと LSW をクリア もし Sy<0 ならば Sx>>|Sy|→Dz、Dz のガードビットと LSW をクリア	111110********** 10000001xxyyzzzz	1	更新	更新
DCT PSHL Sx,Sy,Dz	もし DC=1 & Sy≧0 ならば Sx<<Sy→Dz、Dz のガードビットと LSW をクリア もし DC=1 & Sy<0 ならば Sx>>|Sy|→Dz、Dz のガードビットと LSW をクリア もし DC=0 ならば nop	111110********** 10000010xxyyzzzz	1	−	−
DCF PSHL Sx,Sy,Dz	もし DC=0 & Sy≧0 ならば Sx<<Sy→Dz、Dz のガードビットと LSW をクリア もし DC=0 & Sy<0 ならば Sx>>|Sy|→Dz、Dz のガードビットと LSW をクリア もし DC=1 ならば nop	111110********** 10000011xxyyzzzz	1	−	−
PSHL #Imm,Dz	もし Imm≧0 ならば Dz<<Imm→Dz、Dz のガードビットと LSW をクリア もし Imm<0 ならば Dz>>|Imm|→Dz	111110********** 00000iiiiiiizzzz	1	更新	更新

(1) 説明

Sx または Dz オペランドの上位ワードの内容を論理的にシフトし、その結果を Dz オペランドの上位ワードへ格納し、Dz オペランドの下位ワードを 0 でクリアします。Dz オペランドがガードビットを持つレジスタの場合は、ガードビットも 0 でクリアします。シフト量は Sy オペランドまたはイミディエイト値 Imm オペランドで指定します。シフト量が正の値のとき左にシフトします。負の値のとき右にシフトします。DCT、DCF の条件が指定されている場合は、条件が真のとき命令が実行されます。条件が偽のとき命令は実行されません。

条件が指定されていない場合は DSR レジスタの DC ビットは CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

条件が指定されている場合は、条件が真であっても、DC、N、Z、V、GT ビットは更新されません。

(2) 注意

特にありません。

(3) 動作内容

- レジスタオペランドによる場合

```
{
     unsigned short int  ex2_dz_no, carry_bit, negative_bit, zero_bit, overflow_bit,
                         overrange_bit;
     switch (EX2_SX) {
          case 0x0:  DSP_ALU_SRC1  = X0;
                     break;
          case 0x1:  DSP_ALU_SRC1  = X1;
                     break;
          case 0x2:  DSP_ALU_SRC1  = A0;
                     break;
          case 0x3:  DSP_ALU_SRC1  = A1;
                     break;
     }
     switch (EX2_SY) {
          case 0x0:  DSP_ALU_SRC2  = Y0 & MASK003F0000;
                     break;
          case 0x1:  DSP_ALU_SRC2  = Y1 & MASK003F0000;
                     break;
          case 0x2:  DSP_ALU_SRC2  = M0 & MASK003F0000;
                     break;
          case 0x3:  DSP_ALU_SRC2  = M1 & MASK003F0000;
                     break;
     }
     if((DSP_ALU_SRC2_HW & MASK0020) == 0) { /* Left Shift 0 <= cnt <= 16 */
          char cnt = (DSP_ALU_SRC2_HW & MASK001F);
          if(cnt>16) {
               printf ("PSHL Sx,Sy,Dz Error! Shift %2X exceed range¥n",cnt);
               exit();
          }
          DSP_ALU_DST_HW = DSP_ALU_SRC1_HW << cnt--;
          carry_bit = (((DSP_ALU_SRC1_HW << cnt) & MASK8000) == 0x8000);
     }
     else  {   /* Right Shift 0<cnt<=16 */
```

```
            char cnt = ((~DSP_ALU_SRC2_HW & MASK000F) + 1);
            if(cnt > 16) {
                 printf ("PSHL Sx,Sy,Dz Error! Shift -%2X exceed range¥n",cnt);
                 exit();
            }
             DSP_ALU_DST_HW = DSP_ALU_SRC1_HW >> cnt--;
             carry_bit = (((DSP_ALU_SRC1_HW >> cnt) & MASK0001) == 0x1);
      }

      if(DSP_UNCONDITIONAL_UPDATE) { /* unconditional operation */
             DSP_REG_WD[ex2_dz_no*2] = DSP_ALU_DST_HW;
             DSP_REG_WD[ex2_dz_no*2 + 1] = 0x0;     /* clear LSW */
             if(ex2_dz_no==0)A0G = 0x0;           /* clear Guard bits */
             else if(ex2_dz_no == 1)A1G = 0x0;

             negative_bit = DSP_ALU_DST_MSB;
             zero_bit = (DSP_ALU_DST_HW==0);
             overflow_bit = 0x0;
             overrange_bit = 0x0;
#include "shift_dc_bit.c"
      }
      else if(DSP_CONDITION_MATCH) { /* conditional operation and match */
             DSP_REG_WD[ex2_dz_no*2] = DSP_ALU_DST_HW;
             DSP_REG_WD[ex2_dz_no*2 + 1] = 0x0;     /* clear LSW */
             if(ex2_dz_no == 0) A0G = 0x0;          /* clear Guard bits */
             else if(ex2_dz_no == 1) A1G = 0x0;
      }
}

break;
```

- **イミディエイトオペランドによる場合**

```
{
     unsigned short tmp_imm;
     unsigned short int  ex2_dz_no, carry_bit, negative_bit, zero_bit, overflow_bit,
                         overrange_bit;
     DSP_ALU_SRC1=DSP_REG[ex2_dz_no];
     switch (ex2_dz_no) {
          case 0x0:  DSP_ALU_SRC1G = A0G;
                     break;
          case 0x1:  DSP_ALU_SRC1G = A1G;
                     break;
          default:   if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                     else DSP_ALU_SRC1G = 0x0;
     }

     tmp_imm = ((EX2_LW >> 4) & MASK0000003F); /* bit[9:4] */

     if((tmp_imm & MASK0020)==0) { /* Left Shift 0<= cnt <16 */
          char cnt = (tmp_imm & MASK001F);
          if(cnt > 16) {
               printf("PSHL Dz,#Imm,Dz Error! #Imm=%6X exceed range¥n",tmp_imm);
               exit();
          }
          DSP_ALU_DST_HW = DSP_ALU_SRC1_HW << cnt--;
          carry_bit = (((DSP_ALU_SRC1_HW << cnt) & MASK8000) == 0x8000);
     }

     else   { /* Right Shift 0< cnt <=16 */
            char cnt = ((~tmp_imm & MASK001F)+1);
            if(cnt > 16) {
                printf("PSHL Dz,#Imm,Dz Error! #Imm=%6X exceed range¥n",tmp_imm);
                exit();
            }
            DSP_ALU_DST_HW = DSP_ALU_SRC1_HW >> cnt--;
            carry_bit = (((DSP_ALU_SRC1_HW >> cnt) & MASK0001) == 0x1);
     }

     { /* unconditional operation */
```

```
            DSP_REG_WD[ex2_dz_no*2] = DSP_ALU_DST_HW;
            DSP_REG_WD[ex2_dz_no*2+1] = 0x0; /* clear LSW */
            if(ex2_dz_no == 0) A0G = 0x0;    /* clear Guard bits */
            else if(ex2_dz_no==1) A1G = 0x0;

            negative_bit = DSP_ALU_DST_MSB;
            zero_bit = (DSP_ALU_DST_HW == 0);
            overflow_bit = 0x0;
            overrange_bit = 0x0;
#include "shift_dc_bit.c"
    }
}

break;
```

(4) 使用例

```
PSHL X0,Y0,A0  NOPX  NOPY      ;実行前:X0=H'22222222, Y0=H'00030000,
                                       A0=H'123456789A
                                実行後:X0=H'22222222, Y0=H'00030000,
                                       A0=H'0011100000
PSHL X1,Y1,X1  NOPX  NOPY      ;実行前:X1=H'CCCCCCCC, Y1=H'FFFE0000
                                実行後:X1=H'33330000, Y1=H'FFFE0000
PSHL #7,A1  NOPX  NOPY         ;実行前:A1=H'55555555
                                実行後:A1=H'AA800000
                                無条件実行の場合、DC ビットは CS[2:0]の状態に従って更新。
```

11.4.20 [if cc] PSTS STore System register DSP システム制御命令

条件付き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	DC ビット	T ビット
PSTS MACH,Dz	MACH→Dz	111110**********	1	−	−
		110011010000zzzz			
PSTS MACL,Dz	MACL→Dz	111110**********	1	−	−
		110111010000zzzz			
DCT PSTS MACH,Dz	もし DC＝1 ならば MACH→Dz	111110**********	1	−	−
	もし DC＝0 ならば nop.	110011100000zzzz			
DCT PSTS MACL,Dz	もし DC＝1 ならば MACL→Dz	111110**********	1	−	−
	もし DC＝0 ならば nop.	110111100000zzzz			
DCF PSTS MACH,Dz	もし DC＝0 ならば MACH→Dz	111110**********	1	−	−
	もし DC＝1 ならば nop.	110011110000zzzz			
DCF PSTS MACL,Dz	もし DC＝0 ならば MACL→Dz	111110**********	1	−	−
	もし DC＝1 ならば nop.	110111110000zzzz			

(1) 説明

MACH、MACL レジスタの内容を、Dz オペランドへ格納します。DCT、DCF の条件が指定されている場合は、条件が真のとき命令が実行されます。条件が偽のとき命令は実行されません。

DSR レジスタの DC、N、Z、V、GT、AN、AZ、AV、AGT ビットはいずれも更新されません。また、SR レジスタの T ビットも更新されません。

(2) 注意

PSTS と MOVX、MOVY は並列に指定できますが、実行には 2 サイクルかかる場合があります。

(3) 動作内容

```
      /* MACH -> Dz */
  {
      unsigned short int     ex2_dz_no;
      if(DSP_UNCONDITIONAL_UPDATE) { /* unconditional operation */
            DSP_REG[ex2_dz_no] = MACH;
            if(ex2_dz_no == 0) {
                  A0G = DSP_ALU_DSTG & MASK000000FF;
                  if(DSP_ALU_DSTG_BIT7) A0G = A0G | MASKFFFFFF00;
            }
```

```
            else if(ex2_dz_no==1) {
                A1G = DSP_ALU_DSTG & MASK000000FF;
                if(DSP_ALU_DSTG_BIT7) A1G = A1G | MASKFFFFFF00;
            }
        }

        else if(DSP_CONDITION_MATCH) { /* conditional operation and match */
            DSP_REG[ex2_dz_no] = MACH;
            If(ex2_dz_no == 0) {
                A0G = DSP_ALU_DSTG & MASK000000FF;
                If(DSP_ALU_DSTG_BIT7) A0G = A0G | MASKFFFFFF00;
            }
            else if(ex2_dz_no == 1) {
                A1G = DSP_ALU_DSTG & MASK000000FF;
                If(DSP_ALU_DSTG_BIT7) A1G = A1G | MASKFFFFFF00;
            }
        }
    }

    break;

        /* MACL -> Dz */
    {
        if(DSP_UNCONDITIONAL_UPDATE) { /* unconditional operation */
            DSP_REG[ex2_dz_no] = MACL;
            If(ex2_dz_no==0) {
                A0G = DSP_ALU_DSTG & MASK000000FF;
                If(DSP_ALU_DSTG_BIT7) A0G = A0G | MASKFFFFFF00;
            }
            else if(ex2_dz_no==1) {
                A1G = DSP_ALU_DSTG & MASK000000FF;
                If(DSP_ALU_DSTG_BIT7) A1G = A1G | MASKFFFFFF00;
            }
        }
        else if(DSP_CONDITION_MATCH) { /* conditional operation and match */
            DSP_REG[ex2_dz_no] = MACL;
            If(ex2_dz_no == 0) {
                A0G = DSP_ALU_DSTG & MASK000000FF;
```

```
                If(DSP_ALU_DSTG_BIT7) A0G = A0G | MASKFFFFFF00;
            }
            else if(ex2_dz_no == 1) {
                A1G = DSP_ALU_DSTG & MASK000000FF;
                If(DSP_ALU_DSTG_BIT7) A1G = A1G | MASKFFFFFF00;
            }
        }
    }

    break;
```

(4) 使用例

```
PSTS MACH,A0  NOPX  NOPY        ;実行前:A0=H'123456789A, MACH=H'88888888
                                 実行後:A0=H'FF88888888, MACH=H'88888888
```

11.4.21 [if cc] PSUB SUBtract with Condition DSP 算術演算命令

条件付き減算

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PSUB Sx,Sy,Dz	Sx−Sy→Dz	111110**********	1	更新	更新
		10100001xxyyzzzz			
PSUB Sy,Sx,Dz	Sy−Sx→Dz	111110**********	1	更新	更新
		10000101xxyyzzzz			
DCT PSUB Sx,Sy,Dz	もし DC＝1 ならば	111110**********	1	−	−
	Sx−Sy→Dz	10100010xxyyzzzz			
	もし DC＝0 ならば nop.				
DCT PSUB Sy,Sx,Dz	もし DC＝1 ならば	111110**********	1	−	−
	Sy−Sx→Dz	10000110xxyyzzzz			
	もし DC＝0 ならば nop.				
DCF PSUB Sx,Sy,Dz	もし DC＝0 ならば	111110**********	1	−	−
	Sx−Sy→Dz	10100011xxyyzzzz			
	もし DC＝1 ならば nop.				
DCF PSUB Sy,Sx,Dz	もし DC＝0 ならば	111110**********	1	−	−
	Sy−Sx→Dz	10000111xxyyzzzz			
	もし DC＝1 ならば nop.				

(1) 説明

Sx、Sy オペランドでの減算を行い、その結果を Dz オペランドへ格納します。DCT、DCF の条件が指定されている場合は、条件が真のとき命令が実行されます。条件が偽のとき命令は実行されません。

条件が指定されていない場合は DSR レジスタの DC ビットは CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

条件が指定されている場合は、条件が真であっても、DC、N、Z、V、GT ビットは更新されません。

(2) 注意

特にありません。

(3) 動作内容

```
{
    unsigned long int temp1;
    unsigned short int     ex2_dz_no, carry_bit, borrow_bit, overflow_bit,
```

```
                        overrange_bit;
    switch (EX2_SX) {
        case 0x0:  DSP_ALU_SRC1 = X0;
                   if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                   else   DSP_ALU_SRC1G = 0x0;
                   break;
        case 0x1:  DSP_ALU_SRC1 = X1;
                   if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                   else   DSP_ALU_SRC1G = 0x0;
                   break;
        case 0x2:  DSP_ALU_SRC1   = A0;
                   DSP_ALU_SRC1G = A0G;
                   break;
        case 0x3:  DSP_ALU_SRC1  = A1;
                   DSP_ALU_SRC1G = A1G;
                   break;
    }
    switch (EX2_SY) {
        case 0x0:  DSP_ALU_SRC2  = Y0;
                   break;
        case 0x1:  DSP_ALU_SRC2  = Y1;
                   break;
        case 0x2:  DSP_ALU_SRC2  = M0;
                   break;
        case 0x3:  DSP_ALU_SRC2  = M1;
                   break;
    }
    if(DSP_ALU_SRC2_MSB)   DSP_ALU_SRC2G = 0xff;
    else  DSP_ALU_SRC2G = 0x0;

if(EX2_DSP_BIT13==0) {      /* Sy - Sx - Dz */
    temp1 = DSP_ALU_SRC1;
    DSP_ALU_SRC1 = DSP_ALU_SRC2;
    DSP_ALU SRC2 = temp1;

    DSP_ALU_SRC1G = DSP_ALU_SRC2G;
    If(DSP_ALU_SRC2_MSB)   DSP_APU_SRC2G = 0xff;
    Else  DSP_ALU_SRC2G = 0xff;
```

```
}

     DSP_ALU_DST = DSP_ALU_SRC1 - DSP_ALU_SRC2;
     carry_bit = ((DSP_ALU_SRC1_MSB | !DSP_ALU_SRC2_MSB) && !DSP_ALU_DST_MSB) |
                 (DSP_ALU_SRC1_MSB & !DSP_ALU_SRC2_MSB);
     borrow_bit = !carry_bit;
     DSP_ALU_DSTG_LSB8 = DSP_ALU_SRC1G_LSB8 - DSP_ALU_SRC2G_LSB8 - borrow_bit;
     overflow_bit = !(POS_NOT_OV || NEG_NOT_OV);
     overrange_bit = MINUS_OP_G_OV;
#include "fixed_pt_overflow_protection.c"
     if(DSP_UNCONDITIONAL_UPDATE) { /* unconditional operation */
#include "fixed_pt_unconditional_update.c"
#include "fixed_pt_minus_dc_bit.c"
     }
     else if(DSP_CONDITION_MATCH) { /* conditional operation and match */
          DSP_REG[ex2_dz_no] = DSP_ALU_DST;
          If(ex2_dz_no==0) {
              A0G = DSP_ALU_DSTG & MASK000000FF;
              If(DSP_ALU_DSTG_BIT7) A0G = A0G | MASKFFFFFF00;
          }
          else if(ex2_dz_no==1) {
              A1G = DSP_ALU_DSTG & MASK000000FF;
              If(DSP_ALU_DSTG_BIT7) A1G = A1G | MASKFFFFFF00;
          }
     }
}

break;
```

(4) 使用例

```
PSUB X0,Y0,A0  NOPX  NOPY      ;実行前:X0=H'55555555, Y0=H'33333333,
                                        A0=H'123456789A
                                実行後:X0=H'55555555, Y0=H'33333333,
                                        A0=H'0022222222
                                無条件実行の場合、DC ビットは CS[2:0]の状態に従って更新。
```

11.4.22 PSUB PMULS SUBtract & MULtiply as Signed DSP 算術演算命令

減算と符号付き乗算

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PSUB Sx,Sy,Du PMULS Se,Sf,Dg	Sx－Sy→Du Se の MSW × Sf の MSW→Dg	111110********** 0110eeffxxyygguu	1	更新	更新

(1) 説明

S x オペランドの内容から Sy オペランドの内容を減算し、結果を Du オペランドへ格納します。Se、Sf オペランドの上位ワードの内容を符号付きとして乗算し、結果を Dg オペランドに格納します。この 2 つの処理は同時に並行して実行されます。

DSR レジスタの DC ビットは ALU 演算の結果と CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも ALU 演算の結果に従って更新されます。

また、TC ビットが 1 のときは、SR レジスタの T ビットが ALU 演算の結果と TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

(2) 注意

特にありません。

(3) 動作内容

```
{
/* Multiplier Source assignment */
    unsigned short int  ex2_dz_no, carry_bit, borrow_bit, negative_bit, zero_bit,
                        overflow_bit, overrange_bit;
     switch (EX2_SE) {
           case 0x0: DSP_MLT_SRC1  = X0;
                     break;
           case 0x1: DSP_MLT_SRC1  = X1;
                     break;
           case 0x2: DSP_MLT_SRC1  = Y0;
                     break;
           case 0x3: DSP_MLT_SRC1  = A1;
                     break;
     }
```

```
      switch (EX2_SF) {
            case 0x0: DSP_MLT_SRC2  = Y0;
                      break;
            case 0x1: DSP_MLT_SRC2  = Y1;
                      break;
            case 0x2: DSP_MLT_SRC2  = X0;
                      break;
            case 0x3: DSP_MLT_SRC2  = A1;
                      break;
            }

/* ALU Source assignment */
      switch (EX2_SX) {
           case 0x0:  DSP_ALU_SRC1  = X0;
                      if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                      else  DSP_ALU_SRC1G = 0x0;
                      break;
           case 0x1:  DSP_ALU_SRC1  = X1;
                      if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                      else  DSP_ALU_SRC1G = 0x0;
                      break;
           case 0x2:  DSP_ALU_SRC1  = A0;
                      DSP_ALU_SRC1G = A0G;
                      break;
           case 0x3:  DSP_ALU_SRC1  = A1;
                      DSP_ALU_SRC1G = A1G;
                      break;
      }
      switch (EX2_SY) {
           case 0x0:  DSP_ALU_SRC2  = Y0;
                      break;
           case 0x1:  DSP_ALU_SRC2  = Y1;
                      break;
           case 0x2:  DSP_ALU_SRC2  = M0;
                      break;
           case 0x3:  DSP_ALU_SRC2  = M1;
                      break;
      }
```

```
     if(DSP_ALU_SRC2_MSB)  DSP_APU_SRC2G = 0xff;
     else  DSP_ALU_SRC2G = 0xff;

/* Multiplier Operation * /
     if((SBIT==1) && (DSP_MLT_SRC1_HW==0x8000) && (DSP_MLT_SRC2_HW==0x8000))
     {
          DSP_MLT_DST = 0x7fffffff;/* overflow protection */
     }
     else {
       DSP_MLT_DST = ((long)(short) DSP_MLT_SRC1_HW
                         * (long)(short) DSP_MLT_SRC2_HW)) << 1;
     }
     if(DSP_MLT_DST_MSB) DSP_MLT_DSTG_LSB8 = 0xff;
     else DSP_MLT_DSTG_LSB8 = 0x0;

/* Multiplier Destination assignment */
     switch(EX2_DG){
          case 0x0:  M0 = DSP_MLT_DST;
                     break;
          case 0x1:  M1 = DSP_MLT_DST;
                     break;
          case 0x2:  A0 = DSP_MLT_DST;
                     if(DSP_MLT_DSTG_LSB8 = 0x0) A0G = 0x0
                     else A0G = 0xffffffff;
                     break;
          case 0x3:  A1 = DSP_MLT_DST;
                     if(DSP_MLT_DSTG_LSB8 = 0x0) A1G = 0x0
                     else  A1G = 0xffffffff;
                     break;
     }

/* ALU operation */
     DSP_ALU_DST = DSP_ALU_SRC1 - DSP_ALU_SRC2;
     carry_bit = ((DSP_ALU_SRC1_MSB | !DSP_ALU_SRC2_MSB)&& !DSP_ALU_DST_MSB)|
                 (DSP_ALU_SRC1_MSB & !DSP_ALU_SRC2_MSB);
     borrow_bit = !carry_bit;
     DSP_ALU_DSTG_LSB8 = DSP_ALU_SRC1G_LSB8 - DSP_ALU_SRC2G_LSB8 - borrow_bit;
     overflow_bit= !(POS_NOT_OV || NEG_NOT_OV);
```

```
        overrange_bit = MINUS_OP_G_OV;
    #include "fixed_pt_overflow_protection.c"
        switch (EX2_DU) {
            case 0x0:  X0  = DSP_ALU_DST;
                       negative_bit = DSP_ALU_DST_MSB;
                       zero_bit = (DSP_ALU_DST==0);
                       break;
            case 0x1:  Y0  = DSP_ALU_DST;

                       negative_bit = DSP_ALU_DST_MSB;
                       zero_bit = (DSP_ALU_DST==0);
                       break;
            case 0x2:  A0  = DSP_ALU_DST;
                       A0G = DSP_ALU_DSTG & MASK000000FF;
                       if(DSP_ALU_DSTG_BIT7) A0G = A0G | MASKFFFFFF00;
                       negative_bit = DSP_ALU_DSTG_BIT7;
                       zero_bit = (DSP_ALU_DST==0) & (DSP_ALU_DSTG_LSB8==0);
                       break;
            case 0x3:  A1  = DSP_ALU_DST;
                       A1G = DSP_ALU_DSTG & MASK000000FF;
                       if(DSP_ALU_DSTG_BIT7) A1G = A1G | MASKFFFFFF00;
                       negative_bit = DSP_ALU_DSTG_BIT7;
                       zero_bit = (DSP_ALU_DST == 0) & (DSP_ALU_DSTG_LSB8 == 0);
                       break;
        }

    #include "fixed_pt_minus_dc_bit.c"
    }

    break;
```

(4) 使用例

```
PSUB M0,A0,A0  PMULS X0,Y0,M0  NOPX  NOPY ;
```

実行前: X0=H'00020000,
Y0=H'FFFE0000,
M0=H'33333333,
A0=H'0022222222

実行後: X0=H'00020000,
Y0=H'FFFE0000,
M0=H'FFFFFFF8,
A0=H'0055555555

11.4.23 PSUBC SUBtract with Carry DSP 算術演算命令

ボロー付き減算

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PSUBC Sx,Sy,Dz	Sx−Sy−DC→Dz	111110********** 10100000xxyyzzzz	1	ボロー	更新

(1) 説明

Sx オペランドの内容から Sy オペランドの内容と DC ビットを減算し、Dz オペランドへ格納します。

DSR レジスタの DC ビットはボローフラグとして更新されます。DSR レジスタの N、Z、V、GT ビットも更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

(2) 注意

PADDC 命令実行後の DC ビットは、CS ビットに関係なく、ボローフラグとして更新されます。

(3) 動作内容

```
{
    unsigned short int carry_bit, borrow_bit, overflow_bit, overrange_bit;
     switch (EX2_SX) {
          case 0x0:  DSP_ALU_SRC1  = X0;
                     if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                     else  DSP_ALU_SRC1G = 0x0;
                     break;
          case 0x1:  DSP_ALU_SRC1  = X1;
                     if(DSP_ALU_SRC1_MSB) DSP_ALU_SRC1G = 0xff;
                     else  DSP_ALU_SRC1G = 0x0;
                     break;
          case 0x2:  DSP_ALU_SRC1= A0;
                     DSP_ALU_SRC1G= A0G;
                     break;
          case 0x3:  DSP_ALU_SRC1= A1;
                     DSP_ALU_SRC1G= A1G;
                     break;
     }
```

```
    switch (EX2_SY) {
        case 0x0:  DSP_ALU_SRC2 = Y0;
                   break;
        case 0x1:  DSP_ALU_SRC2 = Y1;
                   break;
        case 0x2:  DSP_ALU_SRC2 = M0;
                   break;
        case 0x3:  DSP_ALU_SRC2 = M1;
                   break;
    }
    if(DSP_ALU_SRC2_MSB) DSP_ALU_SRC2G = 0xff;
    else   DSP_ALU_SRC2G = 0x0;

    DSP_ALU_DST = DSP_ALU_SRC1 - DSP_ALU_SRC2 - DSPDCBIT;
    carry_bit = ((DSP_ALU_SRC1_MSB | !DSP_ALU_SRC2_MSB) && !DSP_ALU_DST_MSB) |
                (DSP_ALU_SRC1_MSB & !DSP_ALU_SRC2_MSB);
    borrow_bit = !carry_bit;
    DSP_ALU_DSTG_LSB8 = DSP_ALU_SRC1G_LSB8 - DSP_ALU_SRC2G_LSB8 - borrow_bit;
    overflow_bit= !(POS_NOT_OV || NEG_NOT_OV);
    overrange_bit = MINUS_OP_G_OV;
#include "fixed_pt_overflow_protection.c"
#include "fixed_pt_unconditional_update.c"
#include "fixed_pt_dc_always_borrow.c"
}

break;
```

(4) 使用例

```
CS[2:0]=***: Always Carry or Borrow Mode
PSUBC X0,Y0,M0  NOPX  NOPY ;    実行前: X0=H'33333333, Y0=H'55555555
                                        M0=H'12345678, DC=0
                                実行後: X0=H'33333333, Y0=H'55555555
                                        M0=H'DDDDDDDE, DC=1
PSUBC X0,Y0,M0  NOPX  NOPY ;    実行前: X0=H'33333333, Y0=H'55555555
                                        M0=H'12345678, DC=1
                                実行後: X0=H'33333333, Y0=H'55555555
                                        M0=H'DDDDDDDD, DC=1
```

11.4.24 [if cc] PSWAP SWAP word DSP 論理演算命令

上位ワードと下位ワードの交換

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PSWAP Sx, Dz	Sx の LSW→Dz の MSW	111110**********	1	更新	更新
	Sx の MSW→Dz の LSW	10011101xx01zzzz			
PSWAP Sy, Dz	Sy の LSW→Dz の MSW	111110**********	1	更新	更新
	Sy の MSW→Dz の LSW	1011110101yyzzzz			
DCT PSWAP Sx,Dz	もし DC＝1 ならば	111110**********	1	−	−
	Sx の LSW→Dz の MSW	10011110xx01zzzz			
	Sx の MSW→Dz の LSW				
	もし 0 ならば nop.				
DCT PSWAP Sy,Dz	もし DC＝1 ならば	111110**********	1	−	−
	Sy の LSW→Dz の MSW	1011111001yyzzzz			
	Sy の MSW→Dz の LSW				
	もし 0 ならば nop.				
DCF PSWAP Sx,Dz	もし DC＝0 ならば	111110**********	1	−	−
	Sx の LSW→Dz の MSW	10011111xx01zzzz			
	Sx の MSW→Dz の LSW				
	もし 1 ならば nop.				
DCF PSWAP Sy,Dz	もし DC＝0 ならば	111110**********	1	−	−
	Sy の LSW→Dz の MSW	1011111101yyzzzz			
	Sy の MSW→Dz の LSW				
	もし 1 ならば nop.				

(1) 説明

Sx または Sy オペランドの上位ワードと下位ワードの内容を交換し、その結果を Dz オペランドへ格納します。DCT、DCF の条件が指定されている場合は、条件が真のとき命令が実行されます。条件が偽のとき命令は実行されません。

条件が指定されていない場合は、DSR レジスタの DC ビットは CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

条件が指定されている場合は、条件が真であっても、DC、N、Z、V、GT ビットは更新されません。

(2) 注意

特にありません。

(3) 動作内容

```
{
     unsigned short int     ex2_dz_no, carry_bit, negative_bit, zero_bit,
                            overflow_bit, overrange_bit;
     DSP_ALU_SRC2 = 0;
     DSP_ALU_SRC2G= 0;
     If(EX2_DSP_BIT13==0) { /* SWAP Sx */
          switch (EX2_SX) {
               case 0x0:  DSP_ALU_SRC1 = X0;
                          if(DSP_ALU_SRC2_MSB) DSP_ALU SRC1G = 0xff;
                          else  DSP_ALU_SRC1G = 0x0;
                          break;
               case 0x1:  DSP_ALU_SRC1 = X1;
                          if(DSP_ALU_SRC2_MSB) DSP_ALU SRC1G = 0xff;
                          else DSP_ALU_SRC1G = 0x0;
                          break;
               case 0x2:  DSP_ALU_SRC1 = A0;
                          DSP_ALU_SRC1G = A0G;
                          break;
               case 0x3:  DSP_ALU_SRC1 = A1;
                          DSP_ALU_SRC1G = A1G;
                          break;
          }
     }

     else {               / * SWAP Sy */
          switch (EX2_SY) {
               case 0x0:  DSP_ALU_SRC1 = Y0;
                          break;
               case 0x1:  DSP_ALU_SRC1 = Y1;
                          break;
               case 0x2:  DSP_ALU_SRC1 = M0;
                          break;
               case 0x3;  DSP_ALU_SRC1 = M1;
                          break;
```

```
            }
            if(DSP_ALU_SRC1_MSB)DSP_ALU_SRC1G = 0xff;
            else DSP_ALU_SRC1G = 0x0;
        }
        DSP_ALU_DST = ((DSP_ALU_SRC1 & 0x0000FFFF) << 16) |
                      (DSP_ALU SRC1 & 0xFFFF0000) >> 16));
        if(DSP_ALU_DST_MSB) DSP_ALU_DSTG = 0xff;
        else                DSP_ALU_DSTG = 0x0;

        carry_bit = 0;
        negative_bit = DSP_ALU DSTG_BIT7;
        zero_bit = (DSP_ALU_DST==0)
        overflow_bit = 0;
        overrange_bit = 0;
    #include "fixed_pt_overflow_protection.c"

        if(DSP_UNCONDITIONAL_UPDATE) { /* unconditional operation */
    #include "fixed_pt_unconditional_update.c"
    #include "fixed_pt_plus_dc_bit.c"
        }
        else if(DSP_CONDITION_MATCH) { /* conditional operation and match */
            DSP_REG[ex2_dz_no] = DSP_ALU_DST;
            if(ex2_dz_no == 0) {
                A0G = DSP_ALU_DSTG & MASK000000FF;
                if(DSP_ALU_DSTG_BIT7) A0G = A0G | MASKFFFFFF00;
            }
            else if(ex2_dz_no==1) {
                A1G = DSP_ALU_DSTG & MASK000000FF;
                If(DSP_ALU_DSTG_BIT7) A1G = A1G | MASKFFFFFF00;
            }
        }
    }
    break;
```

(4) 使用例

```
PSWAP X0,M0  NOPX NOPY          ;実行前:X0=H'12345678, M0=H'33333333
                                 実行後:X0=H'12345678, M0=H'56781234
                                 DC ビットは CS[2:0]の状態に従って更新。
```

11.4.25 [if cc] PXOR logical eXclusive OR DSP 論理演算命令

条件付き排他的論理和演算

書式	動作概略	命令コード	実行ステート	DC ビット	T ビット
PXOR Sx,Sy,Dz	Sx ^ Sy→Dz、Dz のガードビットと LSW クリア	111110********** 10100101xxyyzzzz	1	更新	更新
DCT PXOR Sx,Sy,Dz	もし DC＝1 ならば Sx ^ Sy→Dz、Dz のガードビットと LSW クリア もし 0 ならば nop.	111110********** 10100110xxyyzzzz	1	−	−
DCF PXOR Sx,Sy,Dz	もし DC＝0 ならば Sx ^ Sy→Dz、Dz のガードビットと LSW クリア もし 1 ならば nop.	111110********** 10100111xxyyzzzz	1	−	−

(1) 説明

Sx オペランドの上位ワードの内容と Sy オペランドの上位ワードの内容との排他的論理和を演算し、その結果を Dz オペランドの上位ワードへ格納し、Dz オペランドの下位ワードを 0 でクリアします。Dz オペランドがガードビットを持つレジスタの場合は、ガードビットも 0 でクリアします。DCT、DCF の条件が指定されている場合は、条件が真のとき命令が実行されます。条件が偽のとき命令は実行されません。

条件が指定されていない場合は DSR レジスタの DC ビットは CS ビットの指定に従って更新されます。DSR レジスタの N、Z、V、GT ビットも更新されます。また、TC ビットが 1 のときは、SR レジスタの T ビットが TS ビットの指定に従って更新されます。TC ビットと TS ビットの計 4 ビットがすべて 0 でないときは、累積ビットが有効となり、AN、AZ、AV、AGT の各ビットにも演算結果が反映され、結果が累積されます。

条件が指定されている場合は、条件が真であっても、DC、N、Z、V、GT ビットは更新されません。

(2) 注意

デスティネーションレジスタの下位ワードの内容とガードビットの内容は DC ビットの更新には無視されます。

(3) 動作内容

```
{
    unsigned short int     ex2_dz_no, carry_bit, negative_bit, zero_bit,
                           overflow_bit, overrange_bit;
    switch (EX2_SX) {
        case 0x0: DSP_ALU_SRC1 = X0;
                  break;
        case 0x1: DSP_ALU_SRC1 = X1;
                  break;
```

```
            case 0x2: DSP_ALU_SRC1 = A0;
                      break;
            case 0x3: DSP_ALU_SRC1 = A1;
                      break;
      }
      switch (EX2_SY) {
            case 0x0: DSP_ALU_SRC2 = Y0;
                      break;
            case 0x1: DSP_ALU_SRC2 = Y1;
                      break;
            case 0x2: DSP_ALU_SRC2 = M0;
                      break;
            case 0x3: DSP_ALU_SRC2 = M1;
                      break;
      }

      DSP_ALU_DST_HW = DSP_ALU_SRC1_HW ^ DSP_ALU_SRC2_HW;

      If(DSP_UNCONDITIONAL_UPDATE) { /* unconditional operation */
            DSP_REG_WD[ex2_dz_no*2] = DSP_ALU_DST_HW;
            DSP_REG_WD[ex2_dz_no*2+1] = 0x0;  /* clear LSW */
            if(ex2_dz_no==0)A0G = 0x0; /* clear Guard bits */
            else if(ex2_dz_no==1)  A1G = 0x0;

            carry_bit = 0x0;
            negative_bit = DSP_ALU_DST_MSB;
            zero_bit = (DSP_ALU_DST_HW==0);
            overflow_bit = 0x0;
            overrange_bit = 0x0;
#include "logical_dc_bit.c"
      }
      else if(DSP_CONDITION_MATCH) { /* conditional operation and match */
            DSP_REG_WD[ex2_dz_no * 2] = DSP_ALU_DST_HW;
            DSP_REG_WD[ex2_dz_no * 2 + 1] = 0x0; /* clear LSW */
            if(ex2_dz_no == 0)A0G = 0x0;    /* clear Guard bits */
            else if(ex2_dz_no == 1)  A1G = 0x0;
      }
}
```

```
break;
```

(4) 使用例

```
PXOR X0,Y0,A0  NOPX  NOPY;
```

実行前： X0=H'33333333, Y0=H'55555555
A0=H'123456789A

実行後： X0=H'33333333, Y0=H'55555555
A0=H'0066660000

無条件実行の場合、DC ビットは CS[2:0]の状態に従って更新。

12. 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。

12.1 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）

アクセスサイズは、ビット数を表しています。

【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
	内蔵メモリ制御レジスタ	RAMCR	R/W	H'FF00 0074	H'1F00 0074	32
X/Y メモリ	X メモリ転送元アドレスレジスタ	XSA	R/W	H'FF00 0050	H'1F00 0050	32
	Y メモリ転送元アドレスレジスタ	YSA	R/W	H'FF00 0054	H'1F00 0054	32
	X メモリ転送先アドレスレジスタ	XDA	R/W	H'FF00 0058	H'1F00 0058	32
	Y メモリ転送先アドレスレジスタ	YDA	R/W	H'FF00 005C	H'1F00 005C	32
	X バス保護制御レジスタ	XPR	R/W	H'FF00 0060	H'1F00 0060	32
	Y バス保護制御レジスタ	YPR	R/W	H'FF00 0064	H'1F00 0064	32
	X バス例外アドレスレジスタ	XEA	R/W	H'FF00 0068	H'1F00 0068	32
	Y バス例外アドレスレジスタ	YEA	R/W	H'FF00 006C	H'1F00 006C	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。

12.2 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 0082	H'0000 0082	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
	内蔵メモリ制御レジスタ	RAMCR	H'0000 0000	H'0000 0000	保持	保持
X/Y メモリ	X メモリ転送元アドレスレジスタ	XSA	不定	不定	保持	保持
	Y メモリ転送元アドレスレジスタ	YSA	不定	不定	保持	保持
	X メモリ転送先アドレスレジスタ	XDA	不定	不定	保持	保持
	Y メモリ転送先アドレスレジスタ	YDA	不定	不定	保持	保持
	X バス保護制御レジスタ	XPR	H'0000 00FC	保持	保持	保持
	Y バス保護制御レジスタ	YPR	H'0000 00FC	保持	保持	保持
	X バス例外アドレスレジスタ	XEA	不定	保持	保持	保持
	Y バス例外アドレスレジスタ	YEA	不定	保持	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
	—	—	—	—	—	—	—	—	—	—	—	—	INTMU	—	—	—
初期値：	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R/W	R	R	R

ビット	ビット名	初期値	R/W	説　　明
31～4	—	H'0000032	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

SH4AL-DSP 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。

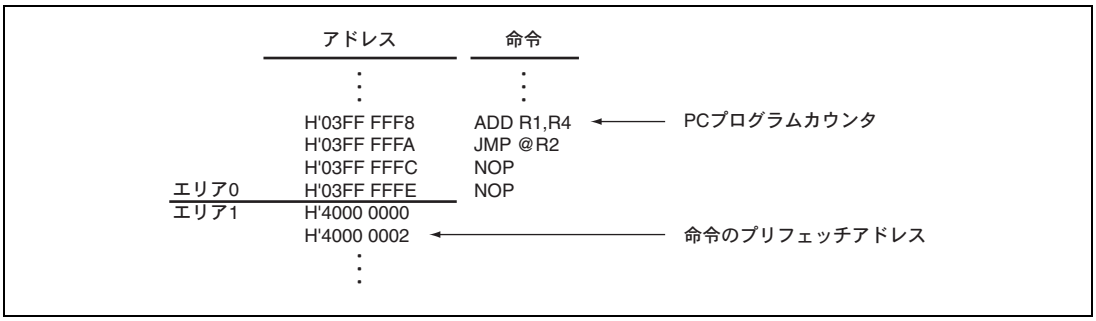


図 B.1　命令のプリフェッチ

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. バージョンレジスタ（PVR、PRR）

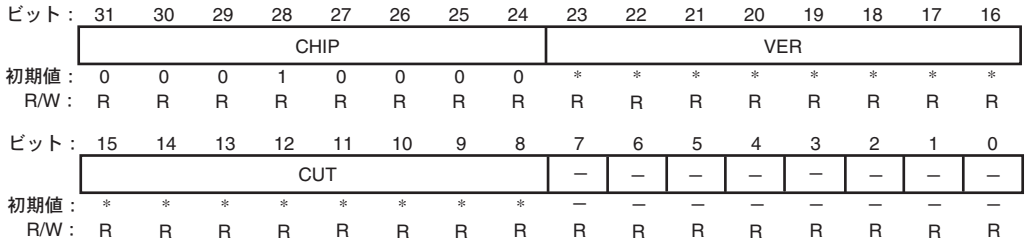
SH4AL-DSP は、プロセッサコアのバージョンと製品のバージョンを示す読み出し専用のレジスタを内蔵しています。これらのレジスタの値を用いることにより、ソフトウェアからプロセッサのバージョンおよび製品を区別することができ拡張性の高いシステムを構築することが可能となります。バージョンレジスタの値は製品ごとに異なるため、詳細は各製品のハードウェアマニュアルを参照するか、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。

【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 C.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

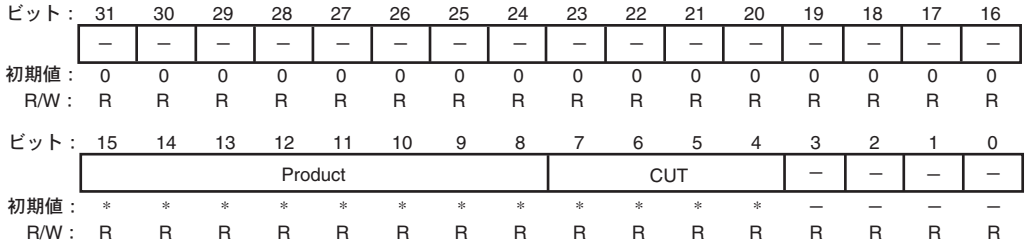
- PVR



ビット	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。 SH4AL-DSP シリーズでは、必ず、H'10 が読み出されます。バージョンを示します。
23～16	VER	*	R	バージョンを示します。 SH4AL-DSP 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。 SH4AL-DSP 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。
SH4AL-DSP 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　SH4AL の特長</td><td>1-1</td><td>修正。
SH-4AL は、SH-4A から FPU 機能を削除した 32 ビット RISC（縮小命令セットコンピュータ）マイコンであり、SH-1、SH-2、SH-3 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します。</td></tr>
<tr><td>表 1.1　SH4AL 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 と上位互換性有り）：</td></tr>
<tr><td>X/Y メモリ</td><td>1-2</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】X/Y 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>U メモリ</td><td>1-2</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 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】U 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　SH4AL-DSP の追加された特長
X/Y メモリ</td><td>1-3</td><td>修正。
• 3 本の独立した読み出し／書き込みポート
CPU からの 8／16／32 ビットアクセス
DSP からの最大 2 つの 16 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</td></tr>
<tr><td>表 1.3　SH-4A から SH4AL-DSP への変更点</td><td>1-4</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>5.　例外処理</td><td>5.6.3</td><td>互換リピート制御中の例外処理</td><td>互換リピート制御中の例外処理の扱いが追加となります。</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>表 1.4　SH3-DSP から SH4AL-DSP への変更点</td><td>1-6</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 rowspan="2">3.　命令セット</td><td rowspan="2">3.3</td><td rowspan="2">命令セット</td><td>CPU 命令として 24 命令を追加</td></tr>
<tr><td>DSP 命令として 42 命令を追加</td></tr>
<tr><td rowspan="2">4.　パイプライン動作</td><td rowspan="2">4.2</td><td rowspan="2">並列実行性</td><td>CPU 命令として 24 命令を追加</td></tr>
<tr><td>DSP 命令として 42 命令を追加</td></tr>
</table></td></tr>
<tr><td></td><td>1-7</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 rowspan="2">11.　各命令の説明</td><td rowspan="2">－</td><td rowspan="2">－</td><td>CPU 命令として 24 命令を追加</td></tr>
<tr><td>DSP 命令として 42 命令を追加</td></tr>
</table></td></tr>
<tr><td>2.6　コントロールレジスタ
（1）ステータスレジスタ（SR）</td><td>2-8</td><td>修正。
<table>
<tr><th>ビット</th><th>ビット名</th><th>説　　明</th></tr>
<tr><td>1</td><td>S</td><td>S ビット
MAC 命令および DSP 命令の飽和動作を指定します。</td></tr>
</table></td></tr>
<tr><td>2.8　使用上の注意事項</td><td>2-14</td><td>追加。</td></tr>
<tr><td>表 3.3　データ転送命令の概要</td><td>3-6</td><td>修正。
<table>
<tr><th rowspan="2"></th><th colspan="2">ダブルデータ転送命令</th><th>シングルデータ転送命令</th></tr>
<tr><td>MOVX.W
MOVY.W</td><td>MOVX.W&NOPY
NOPX&MOVY.W
MOVX.L&NOPY
NOPX&MOVY.L</td><td>MOVS.W
MOVS.L</td></tr>
<tr><td>モジュロアドレッシング</td><td>可能</td><td>可能</td><td>不可</td></tr>
<tr><td>データバス</td><td>X バス、Y バス</td><td>X バス、Y バス</td><td>オペランドバス</td></tr>
</table></td></tr>
<tr><td>表 3.11　DSP をサポートする CPU 命令</td><td>3-22</td><td>【注】 *2　SETRC 命令により 1 以上のリピート回数を設定する前に、必ず LDRS 命令と LSRE 命令を毎回実行するようにしてください。</td></tr>
<tr><td>図 4.2　命令実行パターン（2）</td><td>4-4</td><td>修正。
（2-6）LDRC、SETRC（CO タイプ）；2 発行サイクル</td></tr>
</table>

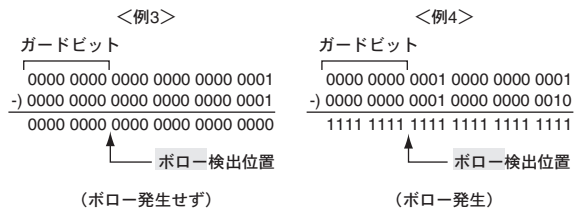
<table>
<tr><th>項　　目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>表 4.2　命令グループ</td><td>4-11</td><td>削除と追加。
<table>
<tr><th>命令
グループ</th><th colspan="2">命　　令</th></tr>
<tr><td>EX</td><td>ROTR
SETDMX
SETDMY
~~SETRC~~
SETS
SETT
SHAD
SHAL
SHAR
SHLD
SHLL
SHLL2
SHLL8
SHLL16
SHLR
SHLR2</td><td>SHLR8
SHLR16
SUB
SUBC
SUBV
SWAP
TST #imm,R0
TST Rm,Rn
XOR #imm,R0
XOR Rm,Rn
XTRCT</td></tr>
<tr><td>CO</td><td>OR.B #imm,@(R0,GBR)
PREFI
RTE
SETRC
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>
<tr><td>表 4.4　発行レートと実行ステート</td><td>4-18</td><td>修正。
<table>
<tr><th>機能
分類</th><th>No.</th><th colspan="2">命　　令</th><th>実行
パターン</th></tr>
<tr><td rowspan="4">システム制御
命令</td><td>176</td><td>STS</td><td>PR,Rn</td><td>4-17</td></tr>
<tr><td>177</td><td>STS.L</td><td>MACH,@-Rn</td><td>5-4</td></tr>
<tr><td>178</td><td>STS.L</td><td>MACL,@-Rn</td><td>5-4</td></tr>
<tr><td>179</td><td>STS.L</td><td>PR,@-Rn</td><td>4-18</td></tr>
</table></td></tr>
<tr><td>5.2.2　例外事象レジスタ（EXPEVT）</td><td>5-3</td><td>修正。
<table>
<tr><th>ビット</th><th>ビット名</th><th>説　　明</th></tr>
<tr><td>31～12</td><td>－</td><td>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。</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>表 5.3　例外一覧</td><td>5-5</td><td>修正。
<table>
<tr><th>例外
区分</th><th>実行
形態</th><th>例外</th><th>優先
レベル</th><th>優先
順位</th></tr>
<tr><td rowspan="4">一般例外</td><td rowspan="2">再実行型</td><td>データ TLB 保護違反例外
（書き込み）*2</td><td>2</td><td>7</td></tr>
<tr><td>初期ページ書き込み例外*2</td><td>2</td><td>8</td></tr>
<tr><td rowspan="2">完了型</td><td>無条件トラップ（TRAPA）</td><td>2</td><td>4</td></tr>
<tr><td>命令実行後ユーザブレーク*1</td><td>2</td><td>9</td></tr>
</table></td></tr>
<tr><td>5.6.1（3）H-UDI リセット</td><td>5-9</td><td>修正。
• 要因：SDIR. TI[7:4]が B'0110（ネゲート）、または B'0111（アサート）</td></tr>
<tr><td>（10）H-UDI リセット</td><td>5-18</td><td>削除。
- 遅延スロット内の PC 相対 MOV 命令、MOVA 命令をデコード
~~- 遅延スロット内の PC 相対 MOV 命令、MOVA 命令をデコード~~</td></tr>
<tr><td>（11）命令実行前ユーザブレーク／命令実行後ユーザブレーク</td><td>5-19</td><td>削除。
（11）命令実行前ユーザブレーク／命令実行前後ユーザブレーク</td></tr>
<tr><td>5.6.3　互換リピート制御中の例外処理</td><td>5-20</td><td>差し替え。</td></tr>
<tr><td>5.6.4　拡張リピート制御中の例外処理</td><td>5-20</td><td>追加。</td></tr>
<tr><td>5.7（4）RTE 命令の遅延スロット</td><td>5-25</td><td>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。~~</td></tr>
<tr><td>6.3　CPU 拡張命令</td><td>6-8</td><td>追加。
リピート制御には、互換リピート制御と拡張リピート制御の 2 つが存在します。</td></tr>
</table>

項　　目	ページ	修正箇所
6.3.1　互換リピート制御命令	6-8	修正。 6.3.1　互換リピート制御命令 このプログラムの例では、RptStart のアドレスにある命令（instr1）から RptEnd のアドレスに配置された命令（instrN）までが 4 回繰り返し実行されます。繰り返し実行されるプログラム範囲をリピートループと呼び、その開始と終了命令をそれぞれリピート開始命令、およびリピート最終命令と呼びます。
	6-9	修正。 ● リピートエンドレジスタ（RE） なお SH4AL-DSP では、SETRC 命令を用いた互換リピート制御を LDRC 命令を用いた拡張リピート制御（「6.3.2　拡張リピート制御命令」を参照）でエミュレーションしています。このため互換リピート制御中に RE レジスタの値が内部状態に応じて変化します。 ● リピートスタートレジスタ（RS） なお SH4AL-DSP では、SETRC 命令を用いた互換リピート制御を LDRC 命令を用いた拡張リピート制御でエミュレーションしています。このため互換リピート制御中に RS レジスタの値が内部状態に応じて変化します。 CPU は、RE レジスタとプログラムカウンタ（PC）の値を常に比較しながら命令を実行します。PC は、命令アドレス+4 の値を保持していますので、リピート検出命令実行時に両者が一致することで、リピート検出命令が検出されます。リピート検出命令の実行が分岐せずに完了し、かつ RC[11:0]>0 である場合にリピート制御が行なわれます。リピート最終命令の実行完了時に RC[11:0]>=2 であれば、RC[11:0]を 1 減じた後 RS レジスタに設定されたアドレスへ制御を移します。RC[11:0]==1（または 0）であれば RC[11:0]を 0 にしたのち、リピート最終命令の次の命令へ制御を移します。
	6-11	修正。 リピート検出命令の実行が分岐せずに完了し、かつ RC[11:0]>0 である場合には、リピート検出命令の次の命令をリピート開始命令として、認識されたリピート命令数分の命令を繰り返し実行します。リピート最終命令実行完了時に、RC[11:0]>=2 であれば、RC[11:0]を 1 減じた後リピート開始命令へ制御を移します。RC[11:0]==1（または 0）であれば RC[11:0]を 0 にしたのち、リピート最終命令の次の命令へ制御を移します。
	6-11	修正。 表 6.5　互換リピート制御 RS および RE のアドレス設定ルール 【注】RptEnd3：リピート最終命令の 3 つ前の命令アドレス （2）互換リピート制御命令およびリピート制御マクロ 表 6.6　互換リピート制御命令 【注】*　SETRC 命令により 1 以上のリピート回数を設定する前には必ず LDRS 命令と LDRE 命令を毎回実行するようにしてください。

<table>
<tr><th>項　　目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>6.3.1　互換リピート制御命令</td><td>6-12</td><td>修正。
表 6.7　互換リピート制御のリピート制御マクロ<table><tr><th>命令</th><th>動　　作</th></tr><tr><td>REPEAT RptStart, RptEnd, #imm</td><td>RptStart をリピート開始命令、RptEnd をリピート最終命令とし、・・</td></tr><tr><td>REPEAT RptStart, RptEnd, Rm</td><td>RptStart をリピート開始命令、RptEnd をリピート最終命令とし、・・</td></tr></table>【注】追加。</td></tr>
<tr><td>（3）互換リピート制御中の制限事項
（b）リピート検出命令に続く命令以降の禁止命令</td><td>6-14</td><td>修正。
（3）互換リピート制御中の制限事項
（b）リピート検出命令に続く命令以降の禁止命令
従来の SH3-DSP ではリピート検出命令の次の命令からリピート最終命令の間に、以下に示す命令を配置できませんでしたが、SH4AL-DSP ではさらにリピート検出命令に以下に示す命令と遅延分岐命令の遅延スロットを配置できなくなります。また従来 SH3-DSP の互換リピート制御では、リピート検出命令の次の命令からリピート最終命令の間に以下に示す命令を配置すると不当命令例外を発生しましたが、SH4AL-DSP では、リピート最終命令に以下に示す命令を配置する場合のみ不当命令例外を発生し、それ以外の場合は不当命令例外は発生しません。</td></tr>
<tr><td>（d）リピート検出命令の次命令以降への分岐および例外受理に関する制限</td><td>6-15</td><td>修正。
従来の SH3-DSP では、リピート検出命令の次命令以降に分岐した場合はリピートループが認識されませんでしたが、SH4AL-DSP ではリピートループが認識されることがあります。ただしリピート検出命令の次命令からリピート最終命令を分岐先に指定することは禁止します。また従来の SH3-DSP では例外ルーチンからの復帰も本制限に含まれていましたが、SH4AL-DSP では例外ルーチンの復帰は本制限に含まれません。

ここでの分岐には、例外ルーチンからの復帰を含みます。~~復帰アドレスがリピート検出命令の次命令以降になる例外が発生するとリピート制御が正しく復帰されません。~~</td></tr>
<tr><td>（e）リピート検出命令からの分岐</td><td>6-15</td><td>修正。
従来の SH3-DSP ではリピート検出命令が遅延分岐命令の遅延スロット命令である場合や分岐命令そのものである場合は、分岐命令で分岐しなかったときにリピートループが認識され、分岐したときにはリピート制御が行われず分岐先命令を実行していましたが、SH4AL-DSP ではリピート検出命令に分岐命令および遅延分岐命令の遅延スロットを配置することができません。</td></tr>
<tr><td>（f）リピートカウンタとリピート制御</td><td>6-15</td><td>修正。
（a）～（e）の制約がかかります。</td></tr>
</table>

<table>
<tr><th>項　　目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>6.3.2　拡張リピート制御命令</td><td>6-16</td><td>修正。
「6.3.1　互換リピート制御命令」で提供されるリピート制御機構には、幾つかの制約事項があります。この制約を軽減するためのリピート制御機能が拡張されています。これらの命令は、従来の SH-DSP アーキテクチャには存在しない命令で、互換性を重視する場合には従来の互換リピート制御命令を使用します。~~この意味で従来のリピート制御命令を互換リピート制御命令と呼びます。~~</td></tr>
<tr><td>（2）拡張リピート制御命令</td><td>6-17</td><td>修正。
拡張リピートループを記述するには前節で例示したように、LDRS と LDRE 命令でそれぞれ RS と RE レジスタにそれぞれリピート先頭命令およびリピート最終命令を指定します。</td></tr>
<tr><td>6.4.1　汎用レジスタ</td><td>6-23</td><td>削除。
As0: .REG　(R4);~~これは、シングルデータ転送のために別名が必要なときの定義です。~~
As1: .REG　(R5);~~これは、シングルデータ転送のために別名が必要なときの定義です。~~
As2: .REG　(R2)
As3: .REG　(R3)
Is:　.REG　(R8);~~これは、シングルデータ転送のために別名が必要なときの定義です。~~</td></tr>
<tr><td>6.5.1　DSP データ</td><td>6-35</td><td>修正。
DSR に、TC ビットをゼロ値モード、DC ビットをキャリー／ボローモードに設定します。
PADD 演算結果に従って T ビットが設定され、後続の BT 命令により TRGET_T に分岐します。</td></tr>
<tr><td>表 6.20　DSP 演算命令の命令形式</td><td>6-36</td><td>修正。
<table>
<tr><th>分類</th><th>命令形式</th></tr>
<tr><td>ダブルデータ演算命令</td><td>ALUop. Sx, Sy, Du　MLTop. Se, Sf, Dg</td></tr>
</table>
</td></tr>
<tr><td>表 6.23　ALU 固定小数点算術演算の種類</td><td>6-41</td><td>修正。
<table>
<tr><th>ニーモニック</th><th>機能</th><th>デスティネーション</th></tr>
<tr><td>PCLR</td><td>クリア</td><td>Dz（Du）</td></tr>
</table>
</td></tr>
<tr><td>図 6.13　キャリー／ボローモードでの DC ビット生成の例</td><td>6-42</td><td>修正。
 
</td></tr>
</table>

<table>
<tr><th>項　　目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>図 6.14　負値モードでの DC ビット生成の例</td><td>6-43</td><td>修正。
＜例1＞
ガードビット
1100 0000 0000 0000 0000 0000
+) 0000 0000 0000 0000 0000 0001
1100 0000 0000 0000 0000 0001
符号ビット
（負の値）</td></tr>
<tr><td>6.5.6　ALU 論理演算</td><td>6-47</td><td>修正。
V ビットは、CS[2:0]ビットがオーバフローモードとしてセットされる DC ビットと常に同じ状態を示しますが、本命令では、常に 0 クリアされます。</td></tr>
<tr><td>6.5.7　固定小数点乗算</td><td>6-49</td><td>削除。
乗算演算は、常に無条件で実行されますが、DSR レジスタの DC、N、Z、V、GT、AN、AZ、AV、および AGT の条件コードビットは更新されません。~~また、AGT、AZ、AN、AV ビットも更新されません。~~</td></tr>
<tr><td>表 6.30　PDMSB 命令の定義</td><td>6-54</td><td>修正。
<table><tr><th>ソースデータ</th><th>デスティネーションの結果</th></tr></table></td></tr>
<tr><td>6.5.9　MSB 検出命令</td><td>6-55</td><td>修正。
V ビットは、CS[2:0]ビットがオーバフローモードとしてセットされる DC ビットと常に同じ状態を示しますが、本命令では、常に 0 クリアされます。</td></tr>
<tr><td>6.5.12　オーバフロー防止機能</td><td>6-58</td><td>修正。
ただし、TS[2:0]ビットでオーバフローモードを選択する場合は、S ビットが 1 のときもオーバフロー検出を行います。</td></tr>
<tr><td>表 6.37　A フィールドの並行データ転送命令（1）</td><td>6-60</td><td>修正。
<table><tr><th>分類</th><th>ニーモニック</th><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th></tr><tr><td rowspan="7">Y メモリデータ転送</td><td>NOPY</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr><tr><td>MOVY.W @Ay,Dy</td><td rowspan="6"></td><td rowspan="6"></td><td rowspan="6"></td><td rowspan="6"></td><td rowspan="6"></td><td rowspan="6"></td></tr><tr><td>MOVY.W @Ay+,Dy</td></tr><tr><td>MOVY.W @Ay+Iy,Dy</td></tr><tr><td>MOVY.W Da,@Ay</td></tr><tr><td>MOVY.W Da,@Ay+</td></tr><tr><td>MOVY.W Da,@Ay+Iy</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>表 6.38　A フィールドの並行データ転送命令（2）</td><td>6-61</td><td>X メモリデータ転送
Ax → Axy
Y メモリデータ転送
Ix → Iy
Ay → Ayx
【注】 Axy：R0、R1、R4、R5＝（01、11、00、10）
Ayx：R2、R3、R6、R7＝（10、11、00、01）</td></tr>
<tr><td>表 6.39　B フィールドの ALU 演算命令、乗算命令</td><td>6-61</td><td>修正。
表 6.39　B フィールドの ALU 演算命令、乗算命令
<table>
<tr><th>分類</th><th>ニーモニック</th><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th><th>25～16</th></tr>
<tr><td>Imm
シフト
命令</td><td>PSHL #imm,Dz
PSHA #imm,Dz</td><td colspan="6">1 1 1 1 1 0</td><td>A
フィールド</td></tr>
</table></td></tr>
<tr><td></td><td>6-63</td><td>追加。
<table>
<tr><th>分類</th><th>ニーモニック</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="4">条件付
命令</td><td>[if cc] PABS Sx,Dz</td><td colspan="2" rowspan="4">Ifcc
00：無条件
10：DCT
11：DCF</td><td colspan="2" rowspan="2">Sx</td><td>0 1</td><td colspan="4" rowspan="4"></td></tr>
<tr><td>[if cc] PRND Sx,Dz</td><td>0 1</td></tr>
<tr><td>[if cc] PABS Sy,Dz</td><td colspan="2">0 1</td><td rowspan="2">Sy</td></tr>
<tr><td>[if cc] PRND Sy,Dz</td><td colspan="2">0 1</td></tr>
</table></td></tr>
<tr><td>表 6.40　競合の発生するオペランドとレジスタとの対応</td><td>6-64</td><td>修正。
<table>
<tr><th colspan="2" rowspan="2"></th><th colspan="5">DSPレジスタ</th></tr>
<tr><th>X0</th><th>X1</th><th></th><th>A0</th><th>A1</th></tr>
<tr><td rowspan="3">3オペランド
乗算</td><td>Se</td><td>*</td><td>*</td><td></td><td></td><td>*</td></tr>
<tr><td>Sf</td><td>*</td><td></td><td></td><td></td><td>*</td></tr>
<tr><td>Dg</td><td></td><td></td><td></td><td>*</td><td>*</td></tr>
<tr><td rowspan="3">3オペランド
ALU演算</td><td>Sx</td><td>*</td><td>*</td><td></td><td>*</td><td>*</td></tr>
<tr><td>Sy</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Dz</td><td>*</td><td>*</td><td></td><td>*</td><td>*</td></tr>
</table>
（Dx、Dxy、DyxとDuとDzの競合）　（DuとDgの競合）</td></tr>
</table>

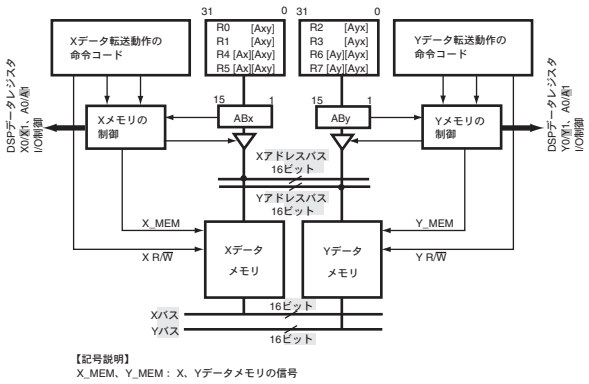
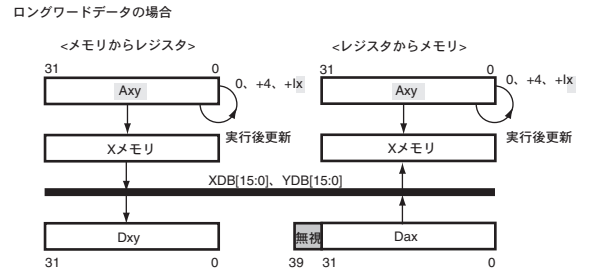
<table>
<tr><th>項　　目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>7.2.6　物理アドレス空間制御レジスタ（PASCR）</td><td>7-14</td><td>修正。<table><tr><th>ビット</th><th>ビット名</th><th>説　　明</th></tr><tr><td>7～0</td><td>UB</td><td>エリア（64M バイト）のバッファドライト制御
キャッシュを使わない書き込みのバスアクセスが完了するまで次の CPU からのバスアクセスを待たせるかをエリアごとに指定します。
0：CPU は書き込み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>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>8-14</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>9.　X/Y メモリ</td><td>9-1</td><td>【注】追加。</td></tr>
<tr><td>10.　U メモリ</td><td>10-1</td><td>【注】追加。</td></tr>
<tr><td>10.4　U メモリの保護機能
• CPU からのアクセスに対する保護機能</td><td>10-5</td><td>追加。
・・・この場合、アドレス変換は行いません。ただし、U メモリ領域をマッピングするページの PPN フィールドには、マッピングされる U メモリの物理アドレスを登録してください。</td></tr>
<tr><td>11.1　CPU 命令</td><td>11-2</td><td>修正。
float　　single precision floating point number(32 bits)</td></tr>
<tr><td>11.1.3　ADDV</td><td>11-6</td><td>修正。
ADD with (Vflag) overflow check</td></tr>
<tr><td>11.1.4　AND
（5）発生する可能性がある例外</td><td>11-9</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>
</table>

<table>
<tr><th>項　　目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>11.1.29　MAC.W
（3）動作内容</td><td>11-55、
11-56</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>11.1.32　MOV
（3）動作内容</td><td>11-66</td><td>修正。
<pre>
MOVBLG(int d) /* MOV.B @(disp,GBR),R0 */
</pre></td></tr>
<tr><td>11.1.33　MOV
（3）動作内容</td><td>11-70</td><td>修正。
<pre>
MOVBS4(long d), long n   /* MOV.B R0,@(disp,Rn) */
</pre></td></tr>
<tr><td>11.1.36　MOVCO</td><td>11-74</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>
<tr><td>11.1.50　OR
（5）発生する可能性のある例外</td><td>11-91</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>11.1.51　PREF
（1）説明</td><td>11-92</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。</td></tr>
<tr><td>（3）動作内容</td><td>11-92</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>11-92</td><td>追加。
• データ TLB 多重ヒット例外</td></tr>
</table>

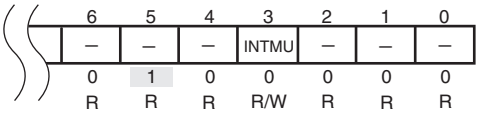
<table>
<tr><th>項　　目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>11.1.52　PREFI
（1）説明</td><td>11-93</td><td>修正。
この命令でデータアドレスエラーおよびMMU例外は発生しません。これらの例外条件に合致した場合は、NOP（無操作）命令として取り扱われます。</td></tr>
<tr><td>（3）動作内容</td><td>11-93</td><td>修正。
PREFI(int n) /* PREFI @Rn */</td></tr>
<tr><td>11.1.70　STC
（3）動作内容</td><td>11-118</td><td>修正。
STCGBR(int n) /* STC GBR,Rn */
{
R[n] = GBR;</td></tr>
<tr><td>11.1.76　SYNCO</td><td>11-129</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>11-129</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>11-129</td><td>差し替え。
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>
<tr><td>（4）使用例</td><td>11-129</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>11.1.77　TAS
（4）発生する可能性がある例外</td><td>11-131</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>11.1.79　TST
（5）発生する可能性がある例外</td><td>11-135</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>11.1.80　XOR
（5）発生する可能性がある例外</td><td>11-137</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>
</table>

項　　目	ページ	修正箇所

11.2.5　LDC
（4）使用例
	11-149	修正。
		`LDC　R0,SR　;実行前　R0=H'FFFFFFFF,SR=H'00000000`
		`;実行後　SR=H'70001FFF,T=1`
		`LDC.L　@R15+,GBR　;実行前　R15=H'10000000,`
		`;@R15+ H'12345678,GBR=H'EDCBA987`
		`;実行後　R15=H'10000004,GBR=@H'10000000`

図 11.1　ダブルデータ転送のロード、ストアの動作	11-169	修正。 
11.3.2　シングルデータ転送（MOVS.W、MOVS.L）	11-171	削除。 メモリとDSPレジスタとの間のデータ転送をLDBバスを使って転送します。~~CPUコアの命令と同じようにデータアクセスと命令アクセスの競合が発生します。~~
11.3.4　MOVX （2）動作内容	11-178	修正。 

<table>
<tr><th>項　　目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>11.3.5　MOVY</td><td>11-179</td><td>修正。
<table>
<tr><th>書式</th><th>動作概略</th><th>命令コード</th></tr>
<tr><td>MOVY.W @Ayx,Dyx</td><td>(Ayx)→Dyx の MSW、
0→Dyx の LSW</td><td>111100AADD000001</td></tr>
<tr><td>MOVY.W @Ayx+,Dyx</td><td>(Ayx)→Dyx の MSW、
0→Dyx の LSW、
Ayx+2→Ayx</td><td>111100AADD000010</td></tr>
<tr><td>MOVY.W @Ayx+Iy,Dyx</td><td>(Ayx)→Dyx の MSW、
0→Dyx の LSW、
Ayx+Ix→Ayx</td><td>111100AADD000011</td></tr>
<tr><td>MOVY.L @Ayx,Dyx</td><td>(Ayx)→Dyx</td><td>111100AADD100001</td></tr>
<tr><td>MOVY.L @Ayx+,Dyx</td><td>(Ayx)→Dyx、
Ayx+4→Ayx</td><td>111100AADD100010</td></tr>
<tr><td>MOVY.L @Ayx+Iy,Dyx</td><td>(Ayx)→Dyx、
Ayx＋Ix→Ayx</td><td>111100AADD100011</td></tr>
</table></td></tr>
<tr><td>（3）使用例</td><td>11-180</td><td>修正。
MOVY.W A0, @R6＋R9　;実行前：R6=H'A5017000, R9=H'00000004, A0=H'123456789A
実行後：R6=H'A5017002, (H'A5017000)=H'3456789A</td></tr>
<tr><td>11.4　DSP 演算命令
（5）演算命令の命令コード</td><td>11-192</td><td>修正。
/* ソースレジスタの判定。
Sx = X0,X1.A0,A1, Sy = Y0,Y1,M0,M1, Se = X0,X1,Y0,A1, Sf = Y0,Y1,X0,A1 */
#define EX2_SX　DSP_Instruction_code.dspcode.sxy.sx
/* 命令コードの[7:6]に相当 */
#define EX2_SY　DSP_Instruction_code.dspcode.sxy.sy
/* 命令コードの[5:4]に相当 */</td></tr>
<tr><td>（6）DSR レジスタ</td><td>11-193</td><td>修正。
SR レジスタの T ビットは、DSR レジスタの TS ビットの指定に従って更新されます。オーバフロー防止機能が有効のとき（S=1）、CS ビットでオーバフローモードを選択する場合、DC ビットは 0 でクリアされますが、TS ビットでオーバフローモードを選択する場合、オーバフロー検出結果が T ビットに反映されます。</td></tr>
<tr><td>11.4.2　[if cc] PADD</td><td>11-207</td><td>修正。
ADD with Condition</td></tr>
<tr><td>11.4.3　PADD PMULS</td><td>11-207</td><td>修正。
ADD & MULtiply as Signed</td></tr>
<tr><td>11.4.4　PADDC</td><td>11-214</td><td>修正。
ADD with Carry</td></tr>
<tr><td>11.4.7　PCLR PMULS</td><td>11-222</td><td>修正。
Clear & MULtiply as Signed</td></tr>
</table>

<table>
<tr><th>項　　目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>11.4.12　[if cc] PINC
（4）使用例</td><td>11-237</td><td>削除。
0PINC X1,X1　NOPX　NOPY　;実行前: X1=H'FC342855</td></tr>
<tr><td>11.4.14　PMULS</td><td>11-242</td><td>修正。
MULtiply as Signed</td></tr>
<tr><td>11.4.22　PSUB PMULS</td><td>11-271</td><td>修正。
SUBtract & MULtiply as Signed</td></tr>
<tr><td>12.　レジスター覧</td><td>12-1</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>
<tr><td>付録 A.　CPU 動作モードレジスタ（CPUOPM）</td><td>付録-1</td><td>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>修正。
 

<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>31～4</td><td>—</td><td>H'0000032</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.　バージョンレジスタ（PVR、PRR）</td><td>付録-3</td><td>追加。</td></tr>
</table>

索引

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
SH4AL-DSP

発行年月日　2003年9月10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

SH4AL-DSP
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

RJJ09B0087-0150Z