

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ひ申し上げます。

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事務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、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と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。

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。

# SH-3、SH-3E、SH3-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

## 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 テクノロジー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 テクノロジー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 テクノロジー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 テクノロジーは、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 テクノロジー半導体製品のご購入に当たりますとは、事前にルネサス テクノロジー、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 テクノロジーホームページ (<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。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 テクノロジー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 テクノロジーは、適用可否に対する責任を負いません。
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 テクノロジー、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 テクノロジー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 テクノロジー、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。



---

## はじめに

---

SH-3、SH-3E、SH3-DSP は、RISC タイプの CPU により、高性能な演算処理を実現し、システム構成に必要な周辺機能を集積すると同時に、携帯用機器に不可欠な低消費電力を同時に実現する新世代シングルチップ RISC マイコンです。

SH-3、SH-3E、SH3-DSP の CPU は、RISC (Reduced instruction set computer) タイプの命令セットを持っており、基本命令は 1 命令 1 ステート (1 システムクロックサイクル) で動作するので、命令実行速度が向上しています。また内部 32 ビット構成を採っていてデータ処理能力を強化しています。

また、SH-3E は単精度浮動小数点演算をサポートし、さらに PCAPI 完全準拠の倍精度浮動小数点演算もエミュレーションします。SH-3E の命令は IEEE754 規格に対応した浮動小数点演算のサブセットとなっています。

このソフトウェアマニュアルは、SH-3、SH-3E、SH3-DSP の命令の詳細について記載しています。命令の動作やアーキテクチャを知るために使ってください。SH-3、SH-3E、SH3-DSP の特長であるパイプラインの動作についても述べてあります。

ハードウェアについては、ハードウェアマニュアルをごらんください。





---

## マニュアルの構成

---

このマニュアルの構成を表 1 に示します。また、記載項目と記載章節との関係を表 2 に示します。

表 1 マニュアルの構成

区分	章名	内容
概要	1. 特長	CPU の特長
アーキテクチャ (1)	2. プログラミングモデル	汎用レジスタ、コントロールレジスタ、システムレジスタの種類と構成
	3. データ形式	レジスタとメモリ上のデータ形式
	4. 浮動小数点演算ユニット	FPU のレジスタ構成、FPU 例外
	5. DSP の演算機能とデータ転送	固定小数点演算、整数演算、論理演算、乗算、シフト演算、飽和演算などの DSP 演算の概要、繰り返し制御
命令の概要	6. 命令の特長	命令の特長、アドレッシングモード、命令形式
	7. 命令セット	分類順の命令概要、アルファベット順命令一覧
命令の詳細	8. 各命令の説明	アルファベット順の各命令の動作
アーキテクチャ (2)	9. 処理状態	低消費電力モードなどの処理状態

表2 項目と関連する章名

区分	項目	章名
概要と特長	CPUの特長	1. 特長
	命令の特長	6.1 RISCタイプ命令セット
	パイプライン	10.1 パイプラインの基本構成 10.2 スロットとパイプラインの流れ
アーキテクチャ	レジスタの構成	2. プログラミングモデル
	データ形式	3. データ形式
	浮動小数点演算ユニット	4. 浮動小数点演算ユニット
	DSP	5. DSPの演算機能とデータ転送
	処理状態、リセット状態、 例外処理状態、バス権解放状態、 プログラム実行状態、 低消費電力モード、スリープモード、 スタンバイモード	9. 処理状態
	パイプライン動作	10. パイプライン動作
命令の概要	命令の特長	6. 命令の特長
	アドレッシングモード	6.2 アドレッシングモード
	命令形式	6.3 命令形式
命令一覧	命令セット	7.1 分類順命令セット
		7.2 アルファベット順命令セット
命令の詳細	命令の動作詳細	8. 各命令の説明
		10.7 各命令のパイプラインの動作
	命令実行ステート数	10.3 命令実行ステート数

---

## 本版で改訂された箇所

---

修正項目	ページ	修正箇所
全体	-	社名変更による変更 (修正前) 日立製作所 → (修正後) ルネサス テクノロジ



---

# 目次

---

## 第 1 章 概要

- 1.1 SH-3 CPU の特長 ..... 1-1
- 1.2 SH3-DSP の特長 ..... 1-3

## 第 2 章 プログラミングモデル

- 2.1 概要 ..... 2-1
- 2.2 汎用レジスタ ..... 2-8
- 2.3 コントロールレジスタ ..... 2-10
- 2.4 システムレジスタ ..... 2-11
- 2.5 レジスタの初期値 ..... 2-12

## 第 3 章 データ形式

- 3.1 レジスタのデータ形式 ..... 3-1
- 3.2 メモリ上でのデータ形式 ..... 3-2
- 3.3 イミディエイトデータのデータ形式 ..... 3-3
- 3.4 DSP タイプデータ形式 (SH3-DSP のみ) ..... 3-3

## 第 4 章 浮動小数点演算ユニット (SH-3E のみ)

- 4.1 概要 ..... 4-1
- 4.2 浮動小数点レジスタと FPU システムレジスタ ..... 4-2
  - 4.2.1 浮動小数点レジスタ ..... 4-2
  - 4.2.2 浮動小数点コミュニケーションレジスタ (FPUL) ..... 4-2
  - 4.2.3 浮動小数点ステータス / コントロールレジスタ (FPSCR) ..... 4-2
- 4.3 浮動小数点フォーマット ..... 4-4
  - 4.3.1 浮動小数点数フォーマット ..... 4-4
  - 4.3.2 非数 (NaN) ..... 4-4
  - 4.3.3 非正規化数の値 ..... 4-5
  - 4.3.4 その他の特殊な値について ..... 4-5
- 4.4 浮動小数点例外モデル ..... 4-6
  - 4.4.1 イネーブル状態の例外 ..... 4-6
  - 4.4.2 ディスエーブル状態の例外 ..... 4-6
  - 4.4.3 FPU の例外事象とコード ..... 4-6
  - 4.4.4 メモリ内の浮動小数点データの配置 ..... 4-6
  - 4.4.5 特殊オペランドを伴う算術演算 ..... 4-6

4.5	CPU との同期化 .....	4-7
<b>第 5 章 DSP の演算機能とデータ転送 (SH3-DSP のみ)</b>		
5.1	ALU 固定小数点演算 .....	5-2
5.2	ALU 整数演算 .....	5-7
5.3	ALU 論理演算 .....	5-9
5.4	固定小数点乗算 .....	5-11
5.5	シフト演算 .....	5-13
5.5.1	算術シフト演算 .....	5-13
5.5.2	論理シフト演算 .....	5-15
5.6	MSB 検出命令 .....	5-18
5.7	丸め処理 .....	5-21
5.8	状態選択ビット (CS) と DSP 状態ビット (DC) .....	5-24
5.9	オーバフロー防止機能 (飽和演算) .....	5-25
5.10	データ転送 .....	5-26
5.10.1	X、Y メモリデータ転送 .....	5-26
5.10.2	シングルデータ転送 .....	5-27
5.11	オペランド競合 .....	5-30
5.12	DSP 繰り返し (ループ) 制御 .....	5-31
5.12.1	注意事項 .....	5-33
5.13	条件付き命令とデータ転送 .....	5-37
<b>第 6 章 命令の特長</b>		
6.1	RISC タイプ命令セット .....	6-1
6.1.1	16 ビット固定長命令 .....	6-1
6.1.2	1 命令 / 1 ステート .....	6-1
6.1.3	データサイズ .....	6-1
6.1.4	ロードストアアーキテクチャ .....	6-1
6.1.5	遅延分岐 .....	6-2
6.1.6	乗算 / 積和演算 .....	6-2
6.1.7	T ビット .....	6-2
6.1.8	イミディエイトデータ .....	6-3
6.1.9	絶対アドレス .....	6-3
6.1.10	16 ビット / 32 ビットディスプレイメント .....	6-3
6.1.11	特権命令 .....	6-4
6.2	CPU 命令のアドレッシングモード .....	6-5
6.3	DSP データアドレッシング (SH3-DSP のみ) .....	6-8
6.3.1	X、Y データアドレッシング .....	6-8
6.3.2	シングルデータアドレッシング .....	6-9
6.3.3	モジュロアドレッシング .....	6-10
6.3.4	DSP アドレッシング動作 .....	6-11
6.4	CPU 命令の命令形式 .....	6-13
6.5	DSP 命令の命令形式 (SH3-DSP のみ) .....	6-15
6.5.1	ダブル、シングルデータ転送命令 .....	6-16

6.5.2	並行処理命令 .....	6-17
<b>第7章 命令セット</b>		
7.1	分類順命令セット .....	7-1
7.1.1	データ転送命令 .....	7-5
7.1.2	算術演算命令 .....	7-6
7.1.3	論理演算命令 .....	7-8
7.1.4	シフト命令 .....	7-9
7.1.5	分岐命令 .....	7-9
7.1.6	システム制御命令 .....	7-10
7.1.7	浮動小数点命令 (SH-3Eのみ) .....	7-12
7.1.8	FPU システムレジスタに関連する CPU 命令 (SH-3Eのみ) .....	7-13
7.1.9	DSP 機能をサポートする CPU 命令 (SH3-DSPのみ) .....	7-13
7.2	アルファベット順命令セット .....	7-15
7.3	DSP データ転送命令の命令セット (SH3-DSPのみ) .....	7-23
7.3.1	ダブルデータ転送命令 (X メモリデータ) .....	7-23
7.3.2	ダブルデータ転送命令 (Y メモリデータ) .....	7-24
7.3.3	シングルデータ転送命令 .....	7-24
7.4	DSP 演算命令の命令セット (SH3-DSPのみ) .....	7-26
7.4.1	ALU 算術演算命令 .....	7-28
7.4.2	ALU 論理演算命令 .....	7-32
7.4.3	固定小数点乗算命令 .....	7-32
7.4.4	シフト演算命令 .....	7-33
7.4.5	システム制御命令 .....	7-34
7.4.6	NOPX と NOPY の命令コード .....	7-35
<b>第8章 各命令の説明</b>		
8.1	命令説明のフォーム .....	8-1
8.2	命令の説明 (SH-3、SH-3E、SH3-DSP に共通する CPU 命令の説明) .....	8-5
8.2.1	ADD ADD binary : 算術演算命令 .....	8-5
8.2.2	ADDC ADD with Carry : 算術演算命令 .....	8-6
8.2.3	ADDV ADD with V flag overflow check : 算術演算命令 .....	8-7
8.2.4	AND AND logical : 論理演算命令 .....	8-8
8.2.5	BF Branch if False : 分岐命令 .....	8-10
8.2.6	BF/S Branch if False with delay Slot : 分岐命令 .....	8-11
8.2.7	BRA BRAnch : 分岐命令 .....	8-13
8.2.8	BRAF BRAnch Far : 分岐命令 .....	8-15
8.2.9	BSR Branch to SubRoutine : 分岐命令 .....	8-16
8.2.10	BSRF Branch to SubRoutine Far : 分岐命令 .....	8-18
8.2.11	BT Branch if True : 分岐命令 .....	8-20
8.2.12	BT/S Branch if True with delay Slot : 分岐命令 .....	8-21
8.2.13	CLRMAC CLeaR MAC register : システム制御命令 .....	8-23
8.2.14	CLRS CLeaR Sbit : システム制御命令 .....	8-24
8.2.15	CLRT CLeaR Tbit : システム制御命令 .....	8-25
8.2.16	CMP/cond CoMPare conditionally : 算術演算命令 .....	8-26
8.2.17	DIV0S DIVide (step0) as Signed : 算術演算命令 .....	8-29
8.2.18	DIV0U DIVide(step0) as Unsigned : 算術演算命令 .....	8-30

8.2.19	DIV1	DIVide 1 step : 算術演算命令	8-31
8.2.20	DMULS.L	Double-length MULTiPLY as Signed : 算術演算命令	8-35
8.2.21	DMULU.L	Double-length MULTiPLY as Unsigned : 算術演算命令	8-37
8.2.22	DT	Decrement and Test : 算術演算命令	8-39
8.2.23	EXTS	EXTend as Signed : 算術演算命令	8-40
8.2.24	EXTU	EXTend as Unsigned : 算術演算命令	8-41
8.2.25	JMP	JuMP : 分岐命令	8-42
8.2.26	JSR	Jump to SubRoutine : 分岐命令	8-43
8.2.27	LDC	LoaD to Control register : システム制御命令	8-45
8.2.28	LDRE	LoaD effective address to RE register : システム制御命令	8-50
8.2.29	LDRS	LoaD effective address to RS register : システム制御命令	8-51
8.2.30	LDS	LoaD to System register : システム制御命令	8-52
8.2.31	LDTLB	Load PTEH/PTEL to TLB : システム制御命令	8-56
8.2.32	MAC.L	MULTiPLY and ACCumulate Long : 算術演算命令	8-57
8.2.33	MAC	MULTiPLY and ACCumulate Word : 算術演算命令	8-60
8.2.34	MOV	MOVe data : データ転送命令	8-62
8.2.35	MOV	MOVe immediate data : データ転送命令	8-67
8.2.36	MOV	MOVe peripheral data : データ転送命令	8-69
8.2.37	MOV	MOVe structure data : データ転送命令	8-72
8.2.38	MOVA	MOVe effective Address : データ転送命令	8-75
8.2.39	MOVT	MOVe T bit : データ転送命令	8-76
8.2.40	MUL.L	MULTiPLY Long : 算術演算命令	8-77
8.2.41	MULS.W	MULTiPLY as Signed Word : 算術演算命令	8-78
8.2.42	MULU.W	MULTiPLY as Unsigned Word : 算術演算命令	8-79
8.2.43	NEG	NEGate : 算術演算命令	8-80
8.2.44	NEGC	NEGate with Carry : 算術演算命令	8-81
8.2.45	NOP	No Operation : システム制御命令	8-82
8.2.46	NOT	NOT-logical complement : 論理演算命令	8-83
8.2.47	OR	OR logical : 論理演算命令	8-84
8.2.48	PREF	PREFetch data to the cache : システム制御命令	8-86
8.2.49	ROTCL	ROTate with Carry Left : シフト命令	8-87
8.2.50	ROTCR	ROTate with Carry Right : シフト命令	8-88
8.2.51	ROTL	ROTate Left : シフト命令	8-89
8.2.52	ROTR	ROTate Right : シフト命令	8-90
8.2.53	RTE	ReTurn from Exception : システム制御命令	8-91
8.2.54	RTS	ReTurn from SubRoutine : 分岐命令	8-93
8.2.55	SETRC	SET reপরat count RC : システム制御命令	8-95
8.2.56	SETS	SET Sbit : システム制御命令	8-97
8.2.57	SETT	SET T bit : システム制御命令	8-98
8.2.58	SHAD	SHift Arithmetic Dynamically : シフト命令	8-99
8.2.59	SHAL	SHift Arithmetic Left : シフト命令	8-101
8.2.60	SHAR	SHift Arithmetic Right : シフト命令	8-102
8.2.61	SHLD	SHift Logical Dynamically : シフト命令	8-103
8.2.62	SHLL	SHift Logical Left : シフト命令	8-105
8.2.63	SHLLn	n bits SHift Logical Left : シフト命令	8-106
8.2.64	SHLR	SHift Logical Right : シフト命令	8-108
8.2.65	SHLRn	n bits SHift Logical Right : シフト命令	8-109



8.2.66	SLEEP	SLEEP : システム制御命令	8-111
8.2.67	STC	Store Control register : システム制御命令	8-112
8.2.68	STS	STore System register : システム制御命令	8-117
8.2.69	SUB	SUBtract binary : 算術演算命令	8-121
8.2.70	SUBC	SUBtract with Carry : 算術演算命令	8-122
8.2.71	SUBV	SUBtract with(Vflag)underflow check : 算術演算命令	8-123
8.2.72	SWAP	SWAP register halves : データ転送命令	8-124
8.2.73	TAS	Test And Set : 論理演算命令	8-126
8.2.74	TRAPA	TRAP Always : システム制御命令	8-127
8.2.75	TST	TeST logical : 論理演算命令	8-128
8.2.76	XOR	eXclusive OR logical : 論理演算命令	8-130
8.2.77	XTRCT	eXTRaCT : データ転送命令	8-132
8.3	浮動小数点命令と FPU に関連する CPU 命令 (SH-3E のみ)		8-133
8.3.1	FABS	Floating point ABSolute value : 浮動小数点命令	8-135
8.3.2	FADD	Floating point ADD : 浮動小数点命令	8-136
8.3.3	FCMP	Floating point Compare : 浮動小数点命令	8-139
8.3.4	FDIV	Floating point DIVide : 浮動小数点命令	8-143
8.3.5	FLDIO	Floating point LoaD Immediate 0 : 浮動小数点命令	8-145
8.3.6	FLDII	Floating point LoaD Immediate 1 : 浮動小数点命令	8-146
8.3.7	FLDS	Floating point LoaD to System register : 浮動小数点命令	8-147
8.3.8	FLOAT	FLOAting point Convert from Integer : 浮動小数点命令	8-148
8.3.9	FMAC	Floating point MUltyply ACcumulate : 浮動小数点命令	8-149
8.3.10	FMOV	Floating point MOVe : 浮動小数点命令	8-152
8.3.11	FMUL	Floating point MUltyply : 浮動小数点命令	8-156
8.3.12	FNEG	Floating point NEGate : 浮動小数点命令	8-158
8.3.13	FSQRT	Floating point SQUare RooT : 浮動小数点命令	8-159
8.3.14	FSTS	Floating point STore from System register : 浮動小数点 命令	8-161
8.3.15	FSUB	Floating point SUBtract : 浮動小数点命令	8-162
8.3.16	FTRC	Floating point TRuncate and Convert to integer : 浮動小数点命令	8-165
8.3.17	LDS	Load to FPU System register : FPU に関する CPU 命令	8-167
8.3.18	STS	STore from FPU System register : FPU に関する CPU 命令	8-169
8.4	DSP データ転送命令の説明 (SH3-DSP のみ)		8-171
8.4.1	MOVS	MOVe Single data between memory and dsp register : DSP データ転送命令	8-177
8.4.2	MOVX	MOVe between X memory and dsp register : DSP データ転送命令	8-179
8.4.3	MOVY	MOVe between Y memory and dsp register : DSP データ転送命令	8-180
8.4.4	NOPX	No access OPeration for X memory : DSP データ転送命令	8-181
8.4.5	NOPY	No access OPeration for Y memory : DSP データ転送命令	8-182
8.5	DSP 演算命令の説明		8-183
8.5.1	PABS	ABSolute : DSP 算術演算命令	8-195
8.5.2	[if cc] PADD	ADDition with Condition : DSP 算術演算命令	8-198
8.5.3	PADD PMULS	ADDition & MUltyply Signed by Signed : DSP 算術演算命令	8-201
8.5.4	PADDC	ADDition with Carry : DSP 算術演算命令	8-203
8.5.5	[if cc] PAND	logical AND : DSP 論理演算命令	8-205
8.5.6	[if cc] PCLR	CLear : DSP 算術演算命令	8-208
8.5.7	PCMP	CoMPare two data : DSP 算術演算命令	8-210

8.5.8	[if cc] PCOPY COPY with Condition : DSP 算術演算命令 .....	8-212
8.5.9	[if cc] PDEC DECrement by 1 : DSP 算術演算命令 .....	8-215
8.5.10	[if cc] PDMSB Detect MSB with Condition : DSP 算術演算命令 .....	8-218
8.5.11	[if cc] PINC INCrement by 1 with Condition : DSP 算術演算命令 .....	8-221
8.5.12	[if cc] PLDS LoaD System register : DSP システム制御命令 .....	8-224
8.5.13	PMULS MULTiply Signed by Signed : DSP 算術演算命令 .....	8-226
8.5.14	[if cc] PNEG NEGate : DSP 算術演算命令 .....	8-227
8.5.15	[if cc] POR logical OR : DSP 論理演算命令 .....	8-230
8.5.16	PRND RouNDing : DSP 算術演算命令 .....	8-233
8.5.17	[if cc] PSHA SHift Arithmetically with Condition : DSP 算術シフト命令 .....	8-235
8.5.18	[if cc] PSHL SHift Logically with condition : DSP 論理シフト命令 .....	8-240
8.5.19	[if cc] PSTS STore System register : DSP システム制御命令 .....	8-245
8.5.20	[if cc] PSUB SUBtract with Condition : DSP 算術演算命令 .....	8-248
8.5.21	PSUBPMULS : SUBtraction & MULTiply Signedby Signed DSP 算術演算命令 .....	8-251
8.5.22	PSUBC SUBtract with Carry : DSP 算術演算命令 .....	8-253
8.5.23	[if cc] PXOR logical eXclusive OR : DSP 論理演算命令 .....	8-255

## 第9章 処理状態

9.1	処理状態 .....	9-1
9.1.1	リセット状態 .....	9-2
9.1.2	例外処理状態 .....	9-2
9.1.3	プログラム実行状態 .....	9-2
9.1.4	低消費電力状態 .....	9-2
9.1.5	バス権解放状態 .....	9-2
9.2	低消費電力状態 .....	9-3
9.2.1	スリープモード .....	9-3
9.2.2	スタンバイモード .....	9-3
9.2.3	モジュールスタンバイ機能 .....	9-3
9.2.4	ハードウェアスタンバイモード .....	9-4

## 第10章 パイプライン動作

10.1	パイプラインの基本構成 .....	10-1
10.1.1	5段パイプライン .....	10-1
10.1.2	スロットとパイプラインの流れ .....	10-2
10.1.3	1スロットの実行にかかるステート数 .....	10-2
10.1.4	命令実行ステート数 .....	10-3
10.2	競合の発生 .....	10-4
10.2.1	命令フェッチ (IF) とメモリアクセス (MA) の競合 .....	10-4
10.2.2	メモリロード命令による競合 .....	10-7
10.2.3	SR更新命令による競合 .....	10-8
10.2.4	乗算器アクセスによる競合 .....	10-8
10.2.5	FPUの競合 (SH-3Eのみ) .....	10-9
10.2.6	DSPデータ演算命令とストア命令の競合 (SH3-DSPのみ) .....	10-11
10.2.7	DSPレジスタ間転送とメモリ・ロード/ストア動作の競合 (SH3-DSPのみ) .....	10-12
10.3	プログラミングの指針 .....	10-13
10.3.1	競合の種類と命令との対応 .....	10-13

10.3.2	命令実行速度の向上 .....	10-16
10.3.3	ステート数 .....	10-16
10.4	各命令のパイプラインの動作 .....	10-17
10.4.1	データ転送命令 .....	10-27
10.4.2	算術演算命令 .....	10-30
10.4.3	論理演算命令 .....	10-35
10.4.4	シフト命令 .....	10-38
10.4.5	分岐命令 .....	10-40
10.4.6	システム制御命令 .....	10-44
10.4.7	例外処理 .....	10-55
10.4.8	FPU 命令のパイプライン (SH-3E のみ) .....	10-58
10.4.9	DSP データ転送命令 (SH3-DSP のみ) .....	10-60
10.4.10	DSP 演算命令 (SH3-DSP のみ) .....	10-64

## 付 録

A.	命令コード .....	1
A.1	アドレッシングモード別命令セット .....	1
A.2	命令形式別命令セット .....	15
A.3	オペレーションコードマップ .....	29
B.	パイプライン動作と競合 .....	33



---

# 1. 概要

---

## 1.1 SH-3 CPU の特長

SH-3、SH-3E、SH3-DSP の CPU は、RISC (Reduced instruction set computer) タイプの命令セットを持っており、基本命令は 1 命令 1 ステート (1 システムクロックサイクル) で動作するので、命令実行速度が飛躍的に向上しています。また内部 32 ビット構成を採用しておりデータ処理能力を強化しています。

SH-3、SH-3E、SH3-DSP CPU の特長を表 1.1 に示します。

表 1.1 SH-3、SH-3E、SH3-DSP CPU の特長

項目	特長
アーキテクチャ	<ul style="list-style-type: none"><li>ルネサス テクノロジオリジナルアーキテクチャ</li><li>内部 32 ビット構成</li></ul>
汎用レジスタマシン	<ul style="list-style-type: none"><li>汎用レジスタ 32 ビット×16 本 (バンクレジスタ 32 ビット×8 本)</li><li>コントロールレジスタ 32 ビット×5 本</li><li>システムレジスタ 32 ビット×4 本 (SH-3)、 32 ビット×6 本 (SH-3E)</li></ul>
命令セット	<ul style="list-style-type: none"><li>命令長は 16 ビット固定長、これによるコード効率の向上</li><li>ロード・ストア・アーキテクチャ (基本 / 論理演算はレジスタ間で実行)</li><li>遅延分岐方式の採用で、分岐時のパイプラインの乱れを軽減</li><li>C 言語指向の命令セット</li></ul>
命令実行時間	<ul style="list-style-type: none"><li>基本命令は 1 命令 / 1 ステート</li></ul>
アドレス空間	<ul style="list-style-type: none"><li>論理アドレス空間 最大 4G バイト</li></ul>
乗算器内蔵	<ul style="list-style-type: none"><li>32×32→64 の乗算を 2 (~5) ステートで実行、 32×32+64→64 の積和演算を 2 (~5) ステートで実行</li></ul>
パイプライン	<ul style="list-style-type: none"><li>5 段パイプライン方式</li></ul>
処理状態	<ul style="list-style-type: none"><li>リセット状態</li><li>例外処理状態</li><li>プログラム実行状態</li><li>低消費電力状態</li><li>バス権解放状態</li></ul>
低消費電力状態	<ul style="list-style-type: none"><li>スリープモード</li><li>スタンバイモード</li><li>モジュールスタンバイ機能</li></ul>

## 1. 概要

---

項目	特長
FPU (SH-3E のみ)	<ul style="list-style-type: none"><li>• 単精度浮動小数点フォーマット</li><li>• IEEE754 規格のデータタイプのサブセット</li><li>• 無効演算例外およびゼロによる除算例外 (IEEE754 規格準拠)</li><li>• ゼロ方向への丸め処理 (IEEE754 規格準拠)</li><li>• 汎用レジスタファイル 32 ビット×16 の浮動小数点レジスタ</li><li>• 基本命令 実行ピッチ : 1 サイクル / レイテンシー : 2 サイクル ( FADD / FSUB / FMUL )</li><li>• FMAC (浮動小数点数積和演算) 実行ピッチ : 1 サイクル / レイテンシー : 2 サイクル</li><li>• FDIV / FSQRT サポート</li><li>• FLDI0 / FLDI1 ( Load constant 0/1 ) サポート</li></ul>

## 1.2 SH3-DSP の特長

SH3 CPU は、16 ビット長のみの命令を持っています。SH3-DSP は基本的には同じ 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。SH3 CPU は標準のノイマン型アーキテクチャですが、SH3-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。

SH3-DSP に追加された特長を表 1.2 に示します。

表 1.2 SH3-DSP の追加された特長

項目	特長
DSP ユニット	<ul style="list-style-type: none"> <li>乗算器</li> <li>算術演算器 (ALU : Arithmetic Logic Unit)</li> <li>パレルシフタ</li> <li>DSP レジスタ</li> <li>MSB 検知</li> </ul>
乗算器	<ul style="list-style-type: none"> <li>16 ビット×16 ビット→32 ビット (符号付き固定小数点)</li> <li>1 サイクル乗算器</li> </ul>
DSP レジスタ	<ul style="list-style-type: none"> <li>40 ビットデータレジスタ×2 本</li> <li>32 ビットデータレジスタ×6 本</li> <li>モジュロレジスタ (MOD、32 ビット) をコントロールレジスタに追加</li> <li>リピートカウンタ (RC) をステータスレジスタ (SR) に追加</li> <li>繰り返し開始レジスタ (RS、32 ビット)、繰り返し終了レジスタ (RE、32 ビット) をコントロールレジスタに追加</li> </ul>
DSP データバス	<ul style="list-style-type: none"> <li>拡張ハーバード型アーキテクチャ</li> <li>2 つのデータバスおよび 1 つの命令バスを同時にアクセス</li> </ul>
内蔵メモリ	<ul style="list-style-type: none"> <li>16K バイト RAM</li> </ul>
並行処理	<ul style="list-style-type: none"> <li>最大 4 つの並行処理</li> <li>ALU 演算、乗算、および 2 つのロードまたはストア</li> </ul>
アドレス演算器	<ul style="list-style-type: none"> <li>2 つのアドレス演算器</li> <li>2 つのメモリをアクセスするためのアドレス演算</li> </ul>
DSP データ アドレッシングモード	<ul style="list-style-type: none"> <li>インクリメント、デクリメントおよびインデクス</li> <li>それぞれモジュロアドレッシング付きまたはなし</li> </ul>
繰り返し制御	<ul style="list-style-type: none"> <li>ゼロオーバーヘッド繰り返し (ループ) 制御</li> </ul>
命令セット	<ul style="list-style-type: none"> <li>16 ビット長または 32 ビット長 <ul style="list-style-type: none"> <li>16 ビット長 (ロードまたはストアだけの場合)</li> <li>32 ビット長 (ALU 演算、乗算を含む場合)</li> </ul> </li> <li>DSP レジスタをアクセスする SH マイコン命令を追加</li> </ul>
パイプライン	<ul style="list-style-type: none"> <li>5 段パイプライン方式</li> <li>最後の第 5 ステージは WB ステージと DSP ステージ兼用</li> </ul>

## 1. 概要

---



---

## 2. プログラミングモデル

---

### 2.1 概要

#### (1) 処理モード

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

#### (2) 汎用レジスタ

汎用レジスタには、R0 から R15 まで 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。

特権モードのとき、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、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 命令でアクセスします。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 とバンクに依存しない 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 命令でアクセスします。

ユーザモードのときは、バンク 0 の汎用のレジスタ R0\_BANK0 から R7\_BANK0 とバンクに依存しない 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。

SH3-DSP で DSP 拡張機能が有効なとき、DSP タイプの命令では、16 の汎用レジスタの内の 8 つのレジスタが X、Y データメモリおよび L 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。L 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。

## 2. プログラミングモデル

---

### (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) 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) あり、処理モードに関係しません。

### (5) 浮動小数点レジスタと FPU に関するシステムレジスタ (SH-3E のみ)

浮動小数点レジスタには FR0 から FR15 までの 16 本のレジスタがあります。これらは単精度浮動小数点数演算のソース/ディスティネーションレジスタとして使用します。

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

処理モード別のレジスタ構成を図 2.1、図 2.2 に示します。FPU レジスタの詳細は「第 4 章 浮動小数点演算ユニット」を参照してください。

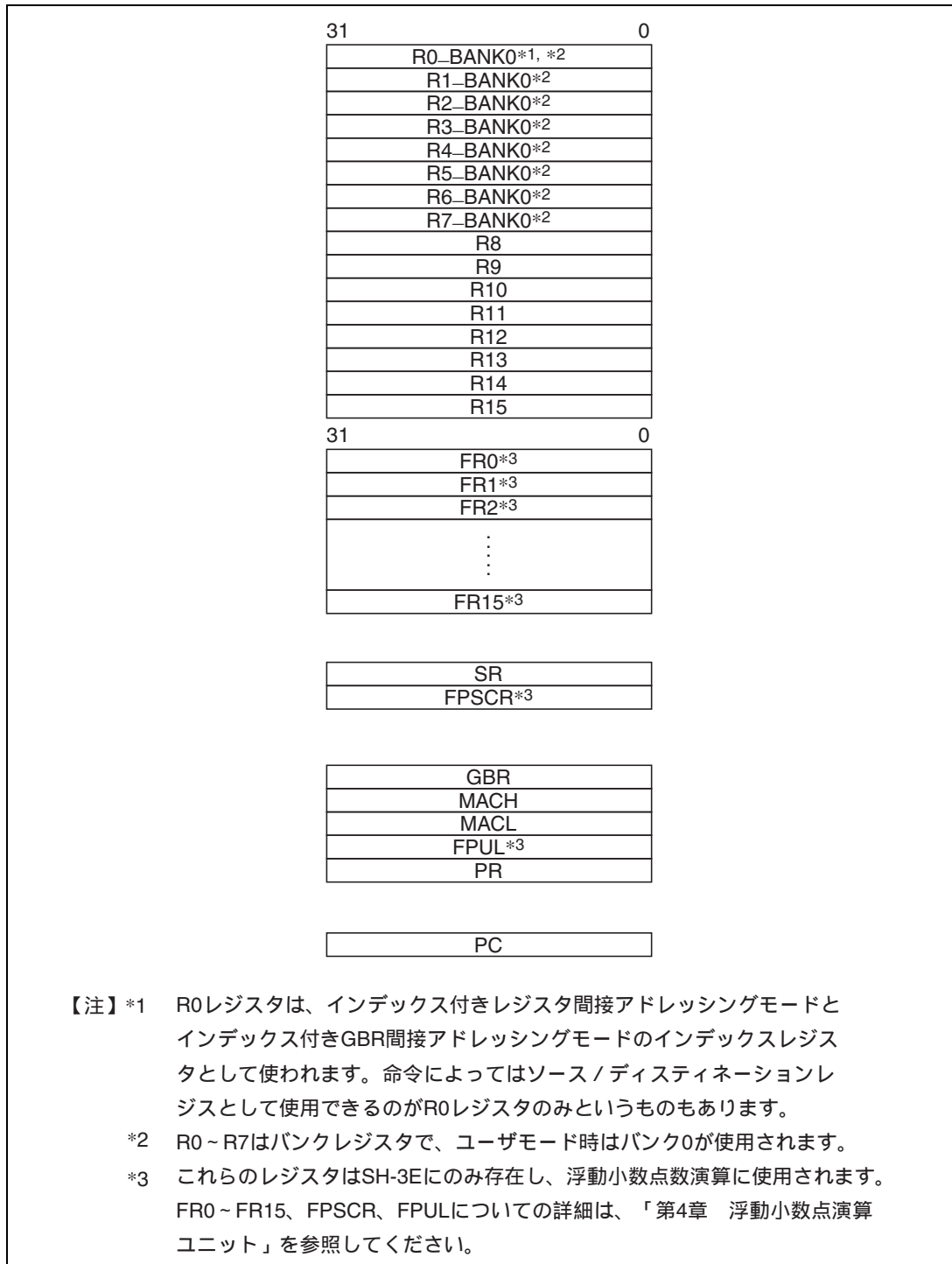


図 2.1 ユーザモード時のプログラミングモデル

## 2. プログラミングモデル

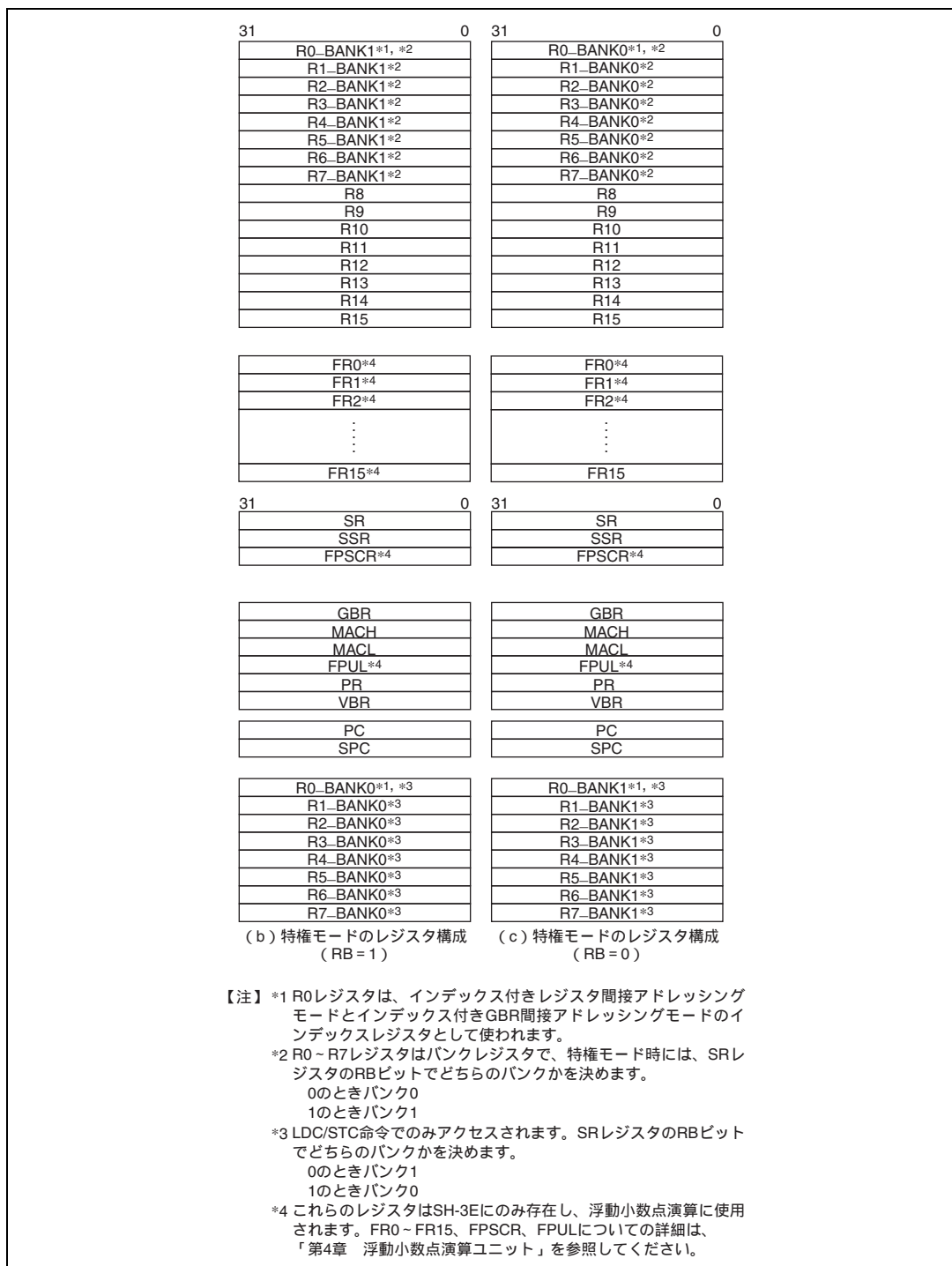


図 2.2 特権モード時のレジスタ構成

## (6) DSP レジスタと DSP に関するコントロールレジスタ (SH3-DSP のみ)

DSP ユニットには 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 データレジスタは、DSP 命令のオペランドとして DSP データのデータ転送、データ処理に使われます。DSP データレジスタをアクセスする命令には、DSP データ処理、X、Y データ転送処理、の 3 つのタイプがあります。

コントロールレジスタは 32 ビット長の DSP ステータスレジスタ (DSR: DSP status register) で、演算結果を表します。DSR レジスタには演算結果を表すビット、符号付き大ビット (GT: Signed greater than)、ゼロビット (Z: Zero value)、負値ビット (N: Negative value)、オーバフロービット (V: overflow)、DSP 状態ビット (DC: DSP condition) と、DC ビットの設定を制御する状態選択ビット (CS: Condition select) があります。

DC ビットは状態フラグの一つを表し、SuperH マイコン CPU コアの T ビットとよく似ています。条件付き DSP タイプ命令の場合、DSP データ処理は、DC ビットに従って実行が制御されます。この制御は DSP ユニットでの実行にだけ関係し、DSP レジスタだけが更新されます。アドレス計算や、ロード/ストア命令などの SuperH マイコンの CPU コアの実行命令には関係しません。コントロールビット CS (ビット 2 から 0) は DC ビットを設定する状態を指定します。

DSP タイプ命令には、無条件 DSP タイプ命令と条件付き DSP タイプ命令があります。無条件 DSP タイプのデータ処理は、PMULS、MOVX、MOVY、MOVZ 命令を除いて、状態ビットと DC ビットを更新します。条件付き DSP タイプ命令は DC ビットの状態によって実行されますが、実行された場合も実行されない場合も DSR レジスタは更新されません。

DSP レジスタを図 2.3 に示します。DSR レジスタのビットの機能を表 2.1 に示します。

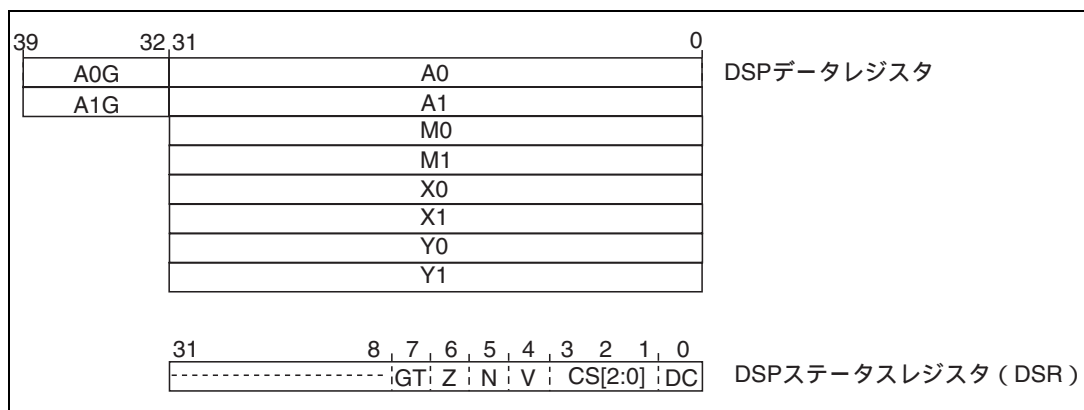


図 2.3 DSP レジスタの構成

## 2. プログラミングモデル

表 2.1 DSR レジスタのビット

ビット	名称 (略称)	機能
31~8	予約ビット	0: 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。
7	符号付き大ビット (GT)	演算結果が正 (ゼロをのぞく)、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	ゼロビット (Z)	演算結果がゼロ (0)、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	負値ビット (N)	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	オーバフロービット (V)	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
3~1	状態選択ビット (CS)	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。 110、111 は指定しないでください。 000: キャリ/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
0	DSP 状態ビット (DC)	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

DSR レジスタは CPU コア命令ではシステムレジスタとして取り扱われます。DSR レジスタとのデータ転送は次のようにロード/ストア命令があります。

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

A0、X0、X1、Y0、Y1 レジスタも CPU コア命令ではシステムレジスタとして取り扱われます。

DSP に関するコントロールレジスタには、繰り返し開始レジスタ (RS: Repeat start register)、繰り返し終了レジスタ (RE: Repeat end register)、モジュロレジスタ (MOD: Modulo register) の 3 本があります。

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。

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。  
図 2.5 にコントロールレジスタを示します。

## 2.2 汎用レジスタ

汎用レジスタの構成を図 2.4 に示します。

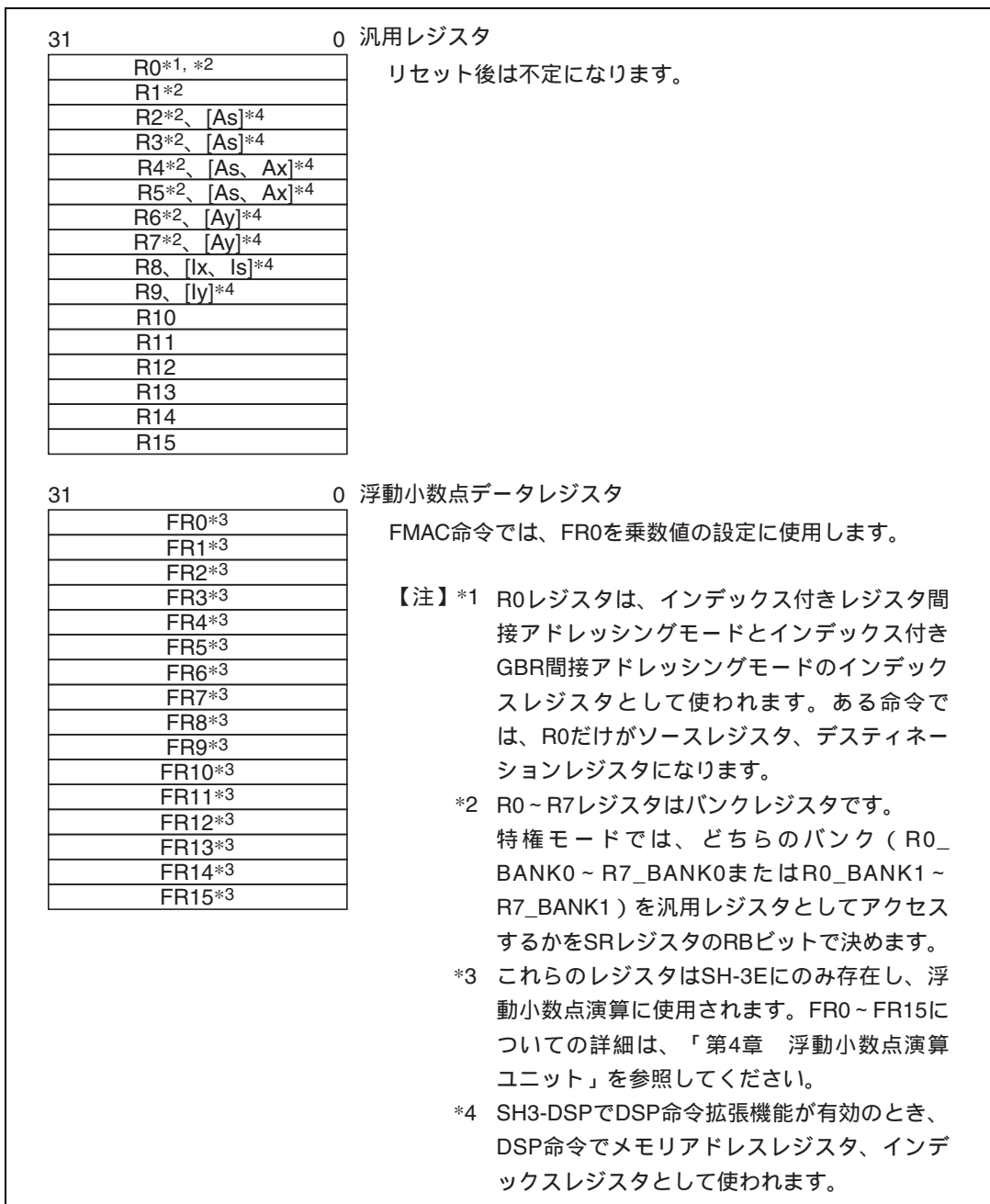


図 2.4 レジスタの構成の概要（汎用レジスタ）



アセンブラでは R2、R3、．．．、R9 の記号名 (シンボル) を使います。もし DSP タイプの命令のためのレジスタの役割を明示した名前にしたいときは、レジスタの別名 (エイリアス、alias) を使います。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)
```

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

```
Ay0: .REG (R6)
```

```
Ay1: .REG (R7)
```

```
Ix: .REG (R9)
```

```
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);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。
```

```
As3: .REG (R3);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。
```

```
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。
```

## 2. プログラミングモデル

### 2.3 コントロールレジスタ

コントロールレジスタの構成を図 2.5 に示します。



図 2.5 レジスタの構成の概要 (コントロールレジスタ)

## 2.4 システムレジスタ

システムレジスタの構成を図 2.6 に示します。  
システムレジスタは LDS / STS 命令でアクセスします。

システムレジスタ	
31	0 積和上位、下位レジスタ (MACH/L) 乗算、積和演算の結果を格納。 リセット後は不定になります。
	MACH
	MACL
31	0 浮動小数点コミュニケーションレジスタ (FPUL) CPUとFPU間のコミュニケーションレジスタと してバッファの役割をもちます。 リセット後は不定になります。
	FPUL*
31	0 プロシージャレジスタ (PR) サブルーチンプロシージャから戻り先アドレスを格納。 リセット後は不定になります。
	PR
31	0 プログラムカウンタ (PC) 現在実行中の命令の開始アドレスの4番地 (2命令) 先を表示。 リセットでH'A000 0000に初期化されます。
	PC
31	0 浮動小数点ステータス/コントロールレジスタ (FPSCR) 浮動小数点演算のステータスの格納、および情報制御を行います。 リセット後は不定になります。
	FPSCR*

【注】 FPUL、FPSCRについては「第4章 浮動小数点演算ユニット」を参照してください。

図 2.6 システムレジスタの構成

## 2.5 レジスタの初期値

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 ~ R15	不定
	FR0 ~ FR15* <sup>1</sup>	不定
コントロールレジスタ	SR	I3 ~ I0 は 1111(H'F)、MD は 1、RB は 1、BL は 1、RC、DMY、DMX は 0 (SH3-DSP のみ)、予約ビットは 0、その他は不定
	GBR	不定
	VBR	H'0000 0000
	SSR、SPC	不定
	RS* <sup>2</sup> 、RE* <sup>2</sup>	不定
	MOD* <sup>2</sup>	不定
システムレジスタ	MACH、MACL、PR、FPSCR* <sup>1</sup> 、FPUL* <sup>1</sup>	不定
	PC	H'A000 0000
DSP レジスタ* <sup>2</sup>	A0、A0G、A1、A1G、M0、M1、X0、X1、Y0、Y1	
	DSR	H'0000 0000

- 【注】 \*1 これらのレジスタは、SH-3E のみ存在し、浮動小数点演算に使用されます。  
FR0 ~ FR15、FPSCR、FPUL についての詳細は、「4.2 浮動小数点レジスタと FPU システムレジスタ」を参照してください。
- \*2 これらのレジスタは、SH3-DSP にのみ存在します。

---

## 3. データ形式

---

### 3.1 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) (図 3.1) 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。

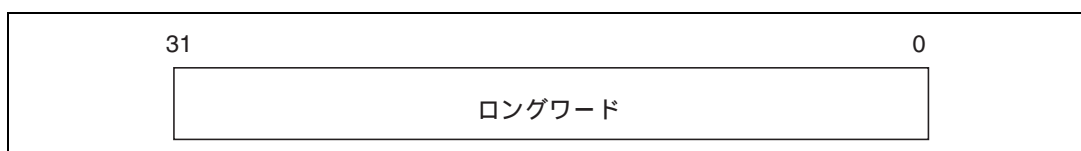


図 3.1 ロングワードオペランド

### 3.2 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。

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）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）からアクセスしてください。これを守らない場合には、アドレスエラーになります。バイト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、またはリトルエンディアンのバイト順を選択可能です。パワーオンリセット時に外部ピン（MD5ピン）で設定してください。MD5ピンがローレベルの場合ビッグエンディアンに、MD5ピンがハイレベルの場合リトルエンディアンに設定されます。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3.2に示します。リトルエンディアンモードのときは、バイト長（8ビット）で書き込んだデータはバイト長で読み出してください。ワード長（16ビット）で書き込んだデータはワード長で読み出してください。

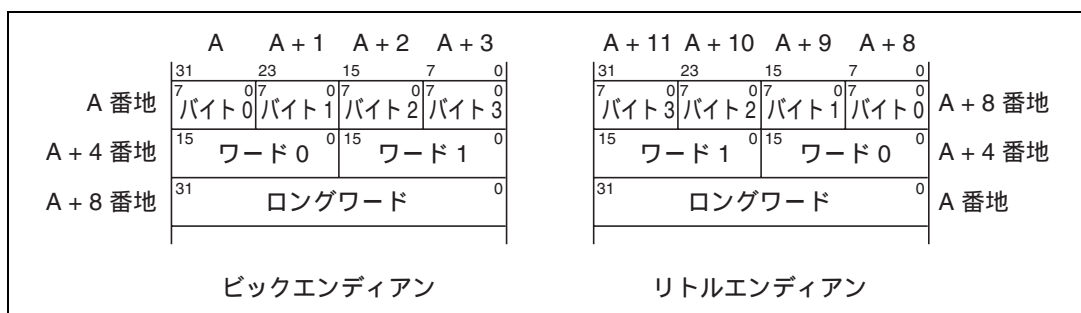


図3.2 メモリ上のデータ形式

### 3.3 イミディエイトデータのデータ形式

バイトのイミディエイトデータは命令コードの中に配置します。

MOV、ADD、CMP/EQ 命令ではイミディエイトデータを符号拡張後、レジスタとロングワードで演算します。一方、TST、AND、OR、XOR 命令ではイミディエイトデータをゼロ拡張後、ロングワードで演算します。したがって、AND 命令でイミディエイトデータを用いると、デスティネーションレジスタの上位 24 ビットは常にクリアされます。

ワードとロングワードのイミディエイトデータは命令コードの中に配置せず、メモリ上のテーブルに配置します。メモリ上のテーブルは、ディスプレースメント付き PC 相対アドレッシングモードを使ったイミディエイトデータのデータ転送命令 (MOV) で、参照します。

具体例については、「6.1.8 イミディエイトデータ」を参照してください。

### 3.4 DSP タイプデータ形式 (SH3-DSP のみ)

SH3-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 進少数点の位置、および参考として SH タイプのデータ形式を図 3.3 に示します。

### 3. データ形式

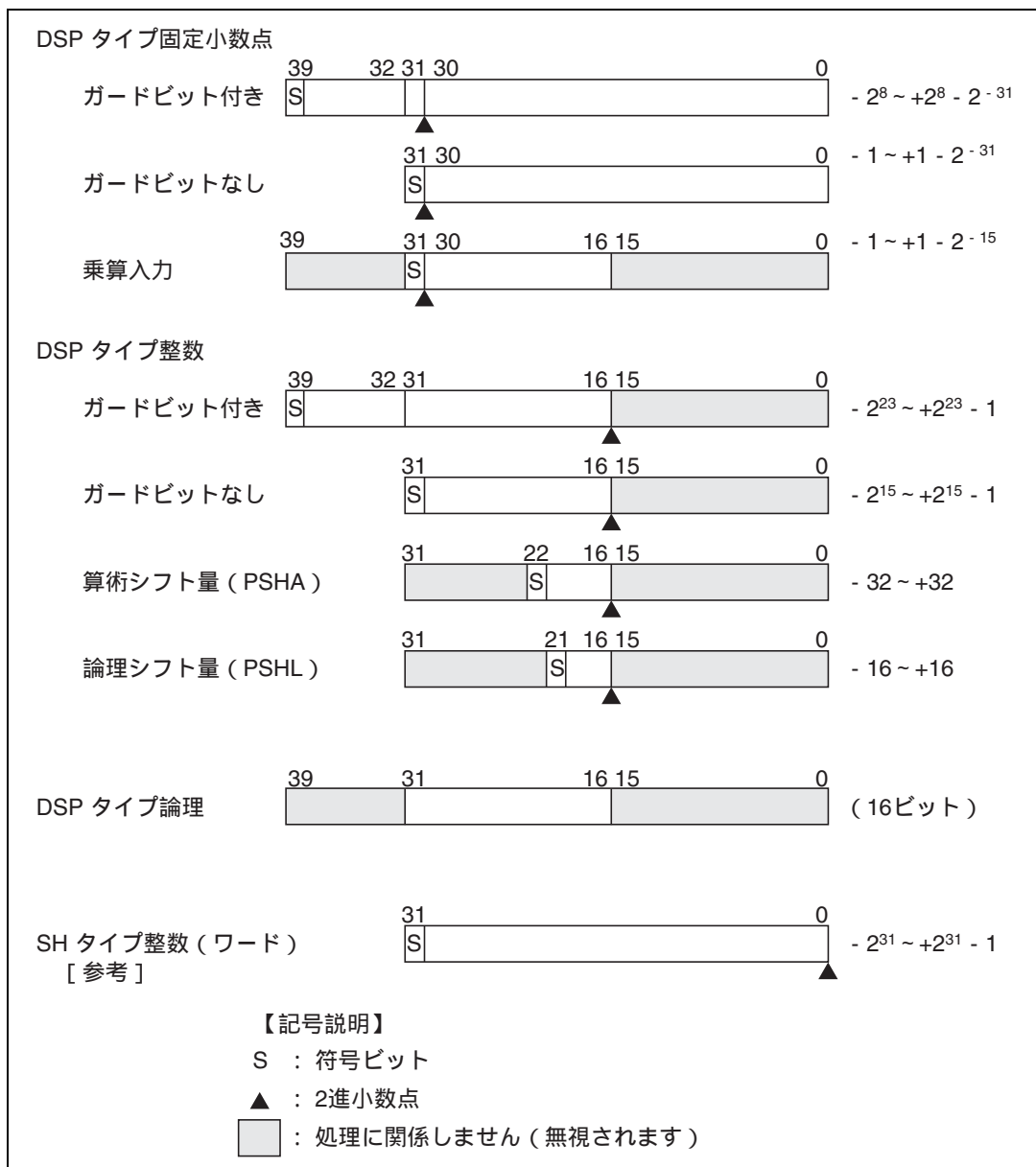


図 3.3 DSP タイプデータ形式



---

## 4. 浮動小数点演算ユニット (SH-3E のみ)

---

### 4.1 概要

SH-3E は、浮動小数点演算ユニット (FPU) を内蔵しています。FPU のレジスタ構成を図 4.1 に示します。

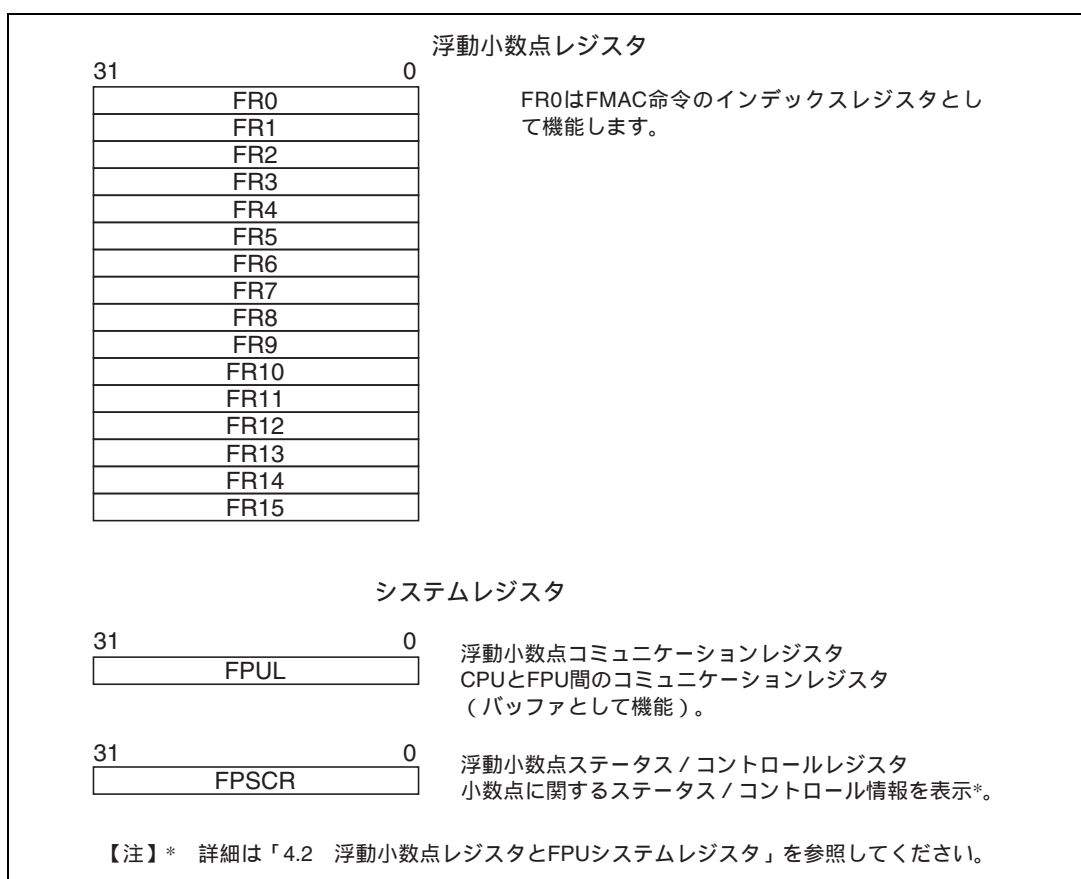


図 4.1 レジスタの構成の概要 (浮動小数点レジスタ、FPU 関連のシステムレジスタ)

## 4.2 浮動小数点レジスタと FPU システムレジスタ

### 4.2.1 浮動小数点レジスタ

SH-3E は 16 本の 32 ビット単精度浮動小数点レジスタをもっています。レジスタ指定は常に 4 ビットで行います。アセンブリ言語では、浮動小数点レジスタは、FR0、FR1、FR2、・・・などのように指定します。FR0 は FMAC 命令のインデックスとして機能します。

### 4.2.2 浮動小数点コミュニケーションレジスタ (FPUL)

FPU と CPU 間で転送される情報は、整数ユニットの MACL、MACH に類似した 1 本のコミュニケーションレジスタ FPUL を介して転送されます。整数形式と浮動小数点形式とは異なるため、SH-3E ではこのコミュニケーションレジスタを設けています。32 ビット FPUL はシステムレジスタで、CPU 側からは LDS、STS 命令によりアクセスされます。

### 4.2.3 浮動小数点ステータス / コントロールレジスタ (FPSCR)

SH-3E は、浮動小数点ステータス / コントロールレジスタ (FPSCR) を備えており、このレジスタは、LDS、STS 命令によりアクセスするシステムレジスタとして機能します (図 4.2)。FPSCR は、ユーザプログラムによる書き込みが可能です。FPSCR は、プロセスコンテキストの一部であり、コンテキスト切り替え時にはセーブする必要があります。また、プロシジャコール時にも、セーブする必要がある場合があります。

FPSCR は、32 ビットのレジスタで、丸めモード、非正規化数の扱い方、および FPU 例外に関する詳細制御情報を示します。

SH-3E では、これらの扱いに関しては以下のモードのみサポートします。

- 丸めモード : 0 方向への丸めモードのみサポート
- 非正規化数の取り扱い :  
非正規化数がソースまたはデスティネーションオペランドにある場合、この値は常に 0 とみなされます。
- FPU 例外 :  
ゼロ除算例外 (Divide by Zero : Z)  
無効演算例外 (Invalid : V) をサポートします。

31					19	18	17	16	15	14				12	11	10	9				7	6	5	4				2	1	0
										要因		イネーブル				フラグ														
0	—————										0	1	0	CV	CZ	0	0	0	EV	EZ	0	0	0	FV	FZ	0	0	0	0	1

【記号説明】

CV : 無効演算要因ビット  
 1 : 現在の命令実行中に無効演算例外が発生したことを示します。  
 0 : 無効演算例外が発生していないことを示します。

CZ : ゼロ除算要因ビット  
 1 : 現在の命令実行中に0による除算例外が発生したことを示します。  
 0 : 0による除算例外が発生していないことを示します。

EV : 無効演算例外イネーブルビット  
 1 : 無効演算例外の発生を許可  
 0 : 無効演算例外は発生を許可せず、結果としてqNaNを返します。

EZ : ゼロ除算例外イネーブルビット  
 1 : 0による除算例外の発生を許可  
 0 : 0による除算例外の発生を許可せず、結果としてqNaNを返します。

FV : 無効演算例外フラグビット  
 1 : 命令の実行中に無効演算例外が発生したことを示します。  
 0 : 無効演算例外は発生していないことを示します。

FZ : ゼロ除算例外フラグビット  
 1 : 命令の実行中に0による除算例外が発生したことを示します。  
 0 : 0による除算例外は発生していないことを示します。

上記ビット以外の各ビットはすべて図のように予約されており、この値はLDS命令でも書き換えることはできません。

図 4.2 浮動小数点ステータス/コントロールレジスタ

要因フィールド中のビットは、そのとき実行中の命令の例外要因を示します。要因ビットは浮動小数点命令によって変更されます。これらのビットは、単一の命令の実行期間中に例外状態が発生するか否かにより、“1”または“0”になります。

イネーブルフィールド中のビットは、イネーブルにする例外の種類を指定します。すなわち例外処理に流れを変更することを可能にします。イネーブルビットと対応する要因ビットが、そのとき実行中の命令よりセットされれば、例外が発生します。

フラグフィールド中のビットは一連の命令の実行中に発生したすべての例外を、累積して格納するのに使用されます。

これらのビットは、いったん命令によってセットされると、その後の命令によってリセットされません。このフィールド中のビットは、FPSCR に対して明示的にストア動作を行うことによるのみ、リセットすることができます。

浮動小数点例外取扱の詳細は「4.4 浮動小数点例外モデル」を参照してください。

## 4.3 浮動小数点フォーマット

### 4.3.1 浮動小数点数フォーマット

SH-3E は単精度浮動小数点演算をサポートしています。浮動小数点フォーマットは IEEE754 浮動小数点規格完全準拠です。

浮動小数点数は、次の 3 つのフィールドにより構成されます。

- 符号部 s
- 指数部 e
- 仮数部 f

指数はバイアスされます。すなわち、

$$e = E + \text{bias}$$

の形式をとります。

バイアスされていない指数 E の範囲は、 $E_{\min} - 1$  から  $E_{\max} + 1$  となります。2 つの値 ( $E_{\min} - 1$  と  $E_{\max} + 1$ ) は以下のように識別されます。 $E_{\min} - 1$  は、ゼロ (符号は正負の双方とも) および非正規化数を表し、 $E_{\max} + 1$  は、正負の無限大および非数 (NaN: Not a Number) を表します。単精度演算では、バイアス値は 127、 $E_{\min}$  は -126、そして  $E_{\max}$  は 127 となります。

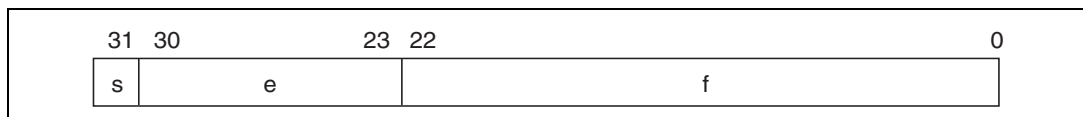


図 4.3 浮動小数点数のフォーマット

浮動小数点数の値  $v$  は、次のように決定されます。

$E = E_{\max} + 1$  かつ  $f \neq 0$  ならば、符号 s に関係なく  $v$  は非数 (NaN)

$E = E_{\max} + 1$  かつ  $f = 0$  ならば、 $v = (-1)^s (\text{infinity})$  [ 正または負の無限大 ]

$E_{\min} \leq E \leq E_{\max}$  ならば、 $v = (-1)^s 2^E (1.f)$  [ 正規化数 ]

$E = E_{\min} - 1$  かつ  $f \neq 0$  ならば、 $v = (-1)^s 2^{E_{\min}} (0.f)$  [ 非正規化数 ]

$E = E_{\min} - 1$  かつ  $f = 0$  ならば、 $v = (-1)^s 0$  [ 正または負のゼロ ]

### 4.3.2 非数 (NaN)

単精度演算値における非数 (NaN) の表現では、ビット 22 ~ 0 のうち少なくとも 1 つのビットが 1 になります。ビット 22 が 1 であれば、シグナリング NaN (sNaN) を示します。ビット 22 が 0 であれば、その値はクワイアット NaN (qNaN) です。

非数 (NaN) のビットパターンを下図に示します。図中のビット N は sNaN では 1 になり、qNaN では 0 になります。x は don't care のビットを示しています。ただし、ビット 22 ~ 0 のうち少なくとも 1 つのビットは 1 になります。

非数 (NaN) では、符号ビットは、don't care になります。

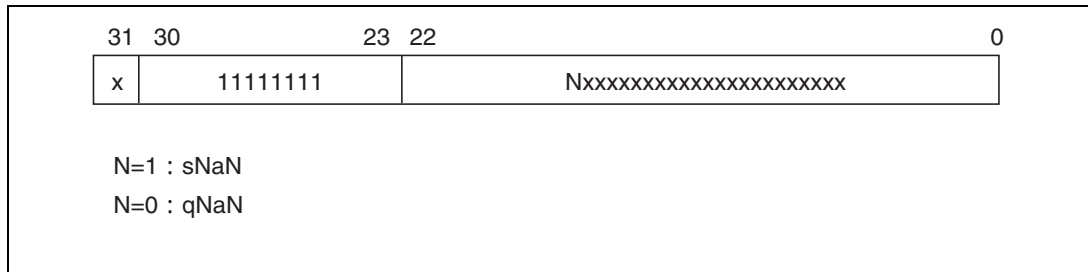


図 4.4 NaN ビットパターン

浮動小数点値を生成する演算に非数 (sNaN) を入力した場合、

- FPSCR レジスタの EV ビットが 0 であれば、演算結果 (出力) は qNaN となります。
- FPSCR レジスタの EV ビットが 1 であれば、無効演算例外が発生します。この場合は、演算のデスティネーション側のレジスタの内容は変更されません。

浮動小数点値を生成する演算に qNaN を入力し、かつ sNaN がその演算に入力されていない場合、FPSCR レジスタの EV ビットのセットとは無関係に、出力は常に qNaN となります。そしてこのとき例外は発生しません。

### 4.3.3 非正規化数の値

非正規化数の浮動小数点数の値は、バイアスされた指数が 0、仮数部がノン-ゼロでヒドゥンビットが 0 として表現されます。SH-3E の浮動小数点演算ユニットでは、非正規化数 (オペランドソースまたは演算結果) は、値を生成する浮動小数点演算 (コピー以外の演算) では画一的に 0 にフラッシュされます。

### 4.3.4 その他の特殊な値について

SH-3E は IEEE754 規格に定められた浮動小数点表現に準拠しています。IEEE754 規格で規定されている浮動小数点数の値の表現には、表 4.1 に示すように 7 種類の異なる種類の特殊な値があります。

表 4.1 IEEE754 規格で規定されている単精度における特殊な値の表現

値	表現
+0.0	0x00000000
-0.0	0x80000000
非正規化数	「4.3.3 非正規化数の値」参照
+INF	0x7F800000
-INF	0xFF800000
qNaN (クワイアット NaN)	「4.3.2 非数 (NaN)」参照
sNaN (シグナリング NaN)	「4.3.2 非数 (NaN)」参照

## 4.4 浮動小数点例外モデル

### 4.4.1 イネーブル状態の例外

無効演算およびゼロによる除算例外の双方は、FPSCR 内の該当例外のイネーブルビット (EV または EZ ビット) をセットすることでイネーブル状態になります。FPU により発生する例外はすべて、FPU 例外事象としてマッピングされています。個々の例外の意味は、システムレジスタ FPSCR を読み出し、そこに保持されている情報を解析して、ソフトウェアにより決定することになります。

### 4.4.2 ディスエーブル状態の例外

FPSCR 内のイネーブルビット EV がセットされていない場合は、無効演算は結果として qNaN を生成します (FCMP と FTRC を除く)。イネーブルビット EZ がセットされていない場合は、ゼロによる除算は現在の式の符号 (+もしくは-) を付けた無限値を返します。

他の IEEE754 規格に定められている不正確 (inexact)、オーバフロー、アンダーフローの浮動小数点例外は、SH-3E ではサポートしていません。この様な場合 SH-3E は以下の動作をします。

- オーバフローは、フォーマットにおいて絶対値が表現可能な最大値となる有限数で、かつ正しい符号をもった数を生成します。アンダーフローは、正しい符号をもったゼロを生成します。もし演算結果が不正確である場合は、デスティネーションレジスタは、その不正確な結果を格納することになります。

### 4.4.3 FPU の例外事象とコード

FPU 例外 (無効除算およびゼロによる除算例外) は、同一の一般例外事象すなわち FPU 例外として、H'0x0120 番地にマッピングされます。また、システムレジスタ FPUL、FPSCR に関するロードおよびストア命令では、通常メモリ管理一般例外が発生します。

### 4.4.4 メモリ内の浮動小数点データの配置

単精度浮動小数点データは、4 バイト境界のメモリ上に配置されます。すなわち、SH-3E のロング整数と同一の形式で配置されます。

### 4.4.5 特殊オペランドを伴う算術演算

特殊オペランド (qNaN、sNaN、+INF、-INF、+0、-0) を伴う算術演算はすべて、IEEE754 規格の規定に従っています。

## 4.5 CPU との同期化

### (1) CPU との同期化

浮動小数点演算命令と CPU 命令は、プログラム順序に従って順番に実行されていきますが、実行サイクルの相違により動作完了がプログラムの順番どおりにならない場合があります。浮動小数点演算命令が FPU リソースのみをアクセスする場合は、CPU との同期化は必要ありませんし、FPU 命令に続く CPU 命令は、FPU 動作の完了以前に動作を終えることができます。それゆえ、最適化されたプログラムにおいては、FDiv のような長い実行サイクルを要する浮動小数点演算命令の実行サイクルを見かけ上隠すことが可能です。一方、CPU リソースにアクセスする FCMP のような浮動小数点演算命令は、プログラム順序を保証する同期化が必要になります。

### (2) 同期化を必要とする浮動小数点命令

ロード、ストア、比較、および FPUL にアクセスする命令は、CPU リソースにアクセスするため、同期化が必要となります。ロード、ストア命令は、汎用レジスタを参照します。ポストインクリメントロードとプリデクリメントストアは、汎用レジスタの内容を変更します。比較は T ビットを変更します。FPUL にアクセスする命令は FPUL を参照するか、内容を変更します。これらの参照と変更は CPU と同期をとる必要があります。

### (3) 例外発生時のプログラム順序の保持

浮動小数点命令は、続く CPU 命令が動作完了する前に動作完了することはありません。また、FPU 例外は、続く CPU 命令が動作完了する以前に検出され、FPU 例外が発生した場合には、続く CPU 命令はキャンセルされます。

浮動小数点命令実行中に、続く命令で例外が発生した場合、その浮動小数点命令は実行を続け、その間 FPU リソースは他の命令によりアクセスされません。他の命令によるアクセスは、浮動小数点演算命令の完了を待って行われます。これはプログラム順序を保証するためです。

#### 4. 浮動小数点演算ユニット (SH-3E のみ)

---



---

## 5. DSP の演算機能とデータ転送 (SH3-DSP のみ)

---

DSP の演算とデータ転送には次のものがあります。

- (1) ALU固定小数点演算：  
固定小数点データ40ビット（ガードビット付き）または32ビット（ガードビットなし）の固定小数点算術演算です。加減算、比較命令などがあります。
- (2) ALU整数演算：  
整数データ24ビット（ガードビット付き）または16ビット（ガードビットなし）の整数算術演算です。インクリメント、デクリメント命令があります。
- (3) ALU論理演算：  
論理データ16ビットの論理演算です。論理積、論理和、排他的論理和があります。
- (4) 固定小数点乗算：  
固定小数点データ上位16ビットの固定小数点乗算（算術演算）です。DCビットなどの状態ビットは更新されません。
- (5) シフト演算：  
算術シフト演算と論理シフト演算があります。算術シフト演算は固定小数点データ40ビット（ガードビット付き）または32ビット（ガードビットなし）の算術シフトです。論理シフト演算は論理データ16ビットの論理演算です。算術シフト演算のシフト量は - 32 ~ + 32（負は右シフト、正は左シフト）、論理シフト演算のシフト量は - 16 ~ + 16です。
- (6) MSB検出命令：  
データを正規化するためのシフト量を求める演算です。固定小数点データ40ビット（ガードビット付き）または32ビット（ガードビットなし）のMSBビット位置を整数データ24ビット（ガードビット付き）または16ビット（ガードビットなし）で求めます。
- (7) 丸め演算：  
固定小数点データ40ビット（ガードビット付き）を24ビットに、または32ビット（ガードビットなし）を16ビットに丸めます。
- (8) データ転送：  
X、Yメモリから16ビットデータをロード、ストアするX、Yデータ転送と、すべてのメモリから16ビット、または32ビットデータをロード、ストアするシングルデータ転送とがあります。X、Yデータ転送は2つの処理を同時に並行して処理することができます。DCビットなどの状態ビットは更新されません。

演算命令には、無条件演算命令と、DCビットを判定して実行する条件付き命令があります。DCビットなどの状態ビットは、条件付き命令では更新されません。DCビットなどの状態ビットは、算術演算、論理演算、算術シフト、論理シフトで、それぞれ設定が異なります。MSB検出命令、丸め演算でのDCビットなどの状態ビットの設定は、算術演算として設定されます。

算術演算にはオーバフロー防止機能（飽和演算）があります。SRレジスタのSビットで飽和演算を指定すると、演算結果がオーバフローしたとき最大値（正）または最小値（負）が格納されます。Sビット機能は、ALU、シフト、乗算のすべての算術演算に有効です。

## 5.1 ALU 固定小数点演算

### (1) 演算機能

ALU 固定小数点算術演算は、基本の精度 32 ビットにガードビット 8 ビットを加えた 40 ビットで演算されます。ソースオペランドがガードビットなしのレジスタのときは、その符号ビットがガードビットに拡張されて転記されます。デスティネーションオペランドがガードビットのないレジスタのときは、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格納されます。

ALU 固定小数点算術演算はレジスタ間で実行されます。ソースおよびデスティネーションレジスタはそれぞれ独立に DSP レジスタから選べます。選ばれたレジスタにガードビットがあるときは、ガードビットを含めてこれらの演算が実行されます。これらの演算の実行はパイプラインの流れの最後の DSP ステージで実行されます。

ALU 算術演算が実行されるときはいつでも、DSR レジスタの DC、N、Z、V、GT ビットが演算結果によって更新されます。しかし条件付き命令の場合は、指定された状態になっても状態ビットは更新されません。無条件命令の場合は、演算結果に従って更新されます。

DC ビットに反映させる状態は、CS [2:0] ビットによって選択されます。ただし、PADDCC 命令と PSUBC 命令の DC ビットは、CS ビットの設定に関係なく更新されます。PADDCC 命令ではキャリフラグとして更新され、PSUBC 命令ではボローフラグとして更新されます。

ALU 固定小数点算術演算の流れを図 5.1 に示します。

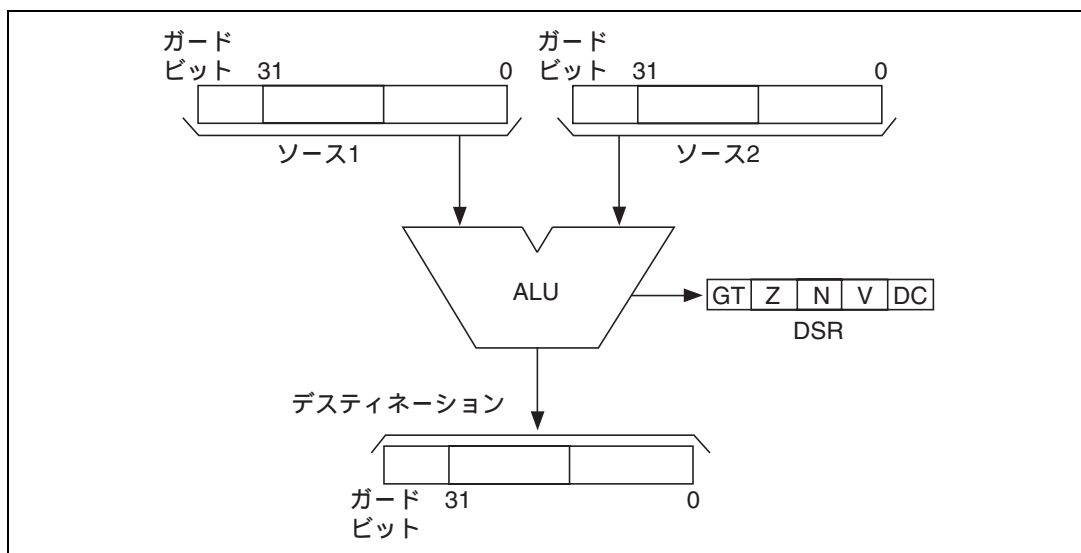


図 5.1 ALU 固定小数点算術演算の流れ

メモリ読み出しのデスティネーションオペランドと ALU 演算のソースオペランドを同じにし、ALU 演算と同じ行にデータ転送命令のプログラムを書いた場合は、メモリアクセスステージ (MA) でメモリからロードされるデータは、ALU 演算命令のソースオペランドとしては使われません。この場合、先に実行された命令の結果が ALU 演算のソースオペランドとして使われ、そのあとで、データロード命令のデスティネーションオペランドとして更新されます。この流れを図 5.2 に示します。

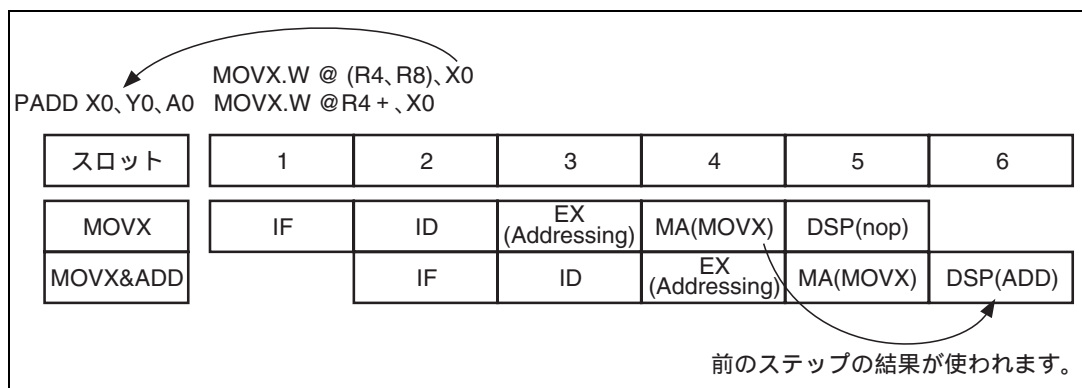


図 5.2 処理の流れの例

## (2) 命令とオペランド

ALU 固定小数点算術演算の種類を表 5.1 に示します。それぞれのオペランドとレジスタとの対応を表 5.2 に示します。

表 5.1 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	データ複写	Sx	—	Dz
		—	Sy	Dz
PABS	絶対値	Sx	—	Dz
		—	Sy	Dz
PNEG	符号反転	Sx	—	Dz
		—	Sy	Dz
PCLR	ゼロクリア	—	—	Dz

5. DSP の演算機能とデータ転送 (SH3-DSP のみ)

表 5.2 ALU 固定小数点算術演算のオペランドとレジスタとの対応

オペランド	X0	X1	Y0	Y1	M0	M1	A0	A1
Sx	Yes	Yes					Yes	Yes
Sy			Yes	Yes	Yes	Yes		
Dz	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Du	Yes		Yes				Yes	Yes

【注】 Yes : オペランドに使えるレジスタです

Du : 乗算と組み合わせられる場合のオペランドです

(3) DC ビット

DC ビットは、DSR レジスタの CS2 ~ CS0 ビット (Condition selection、状態選択) の指定に従って、次のようになります。

(a) キャリ / ボローモード : CS2 ~ CS0 = 000

DC ビットは演算の結果、MSB (Most significant bit) ビットからキャリまたはボローが発生したことを表します。ガードビットは関係ありません。このモードはデフォルト (default) です。キャリとボローの発生例を図 5.3 に示します。

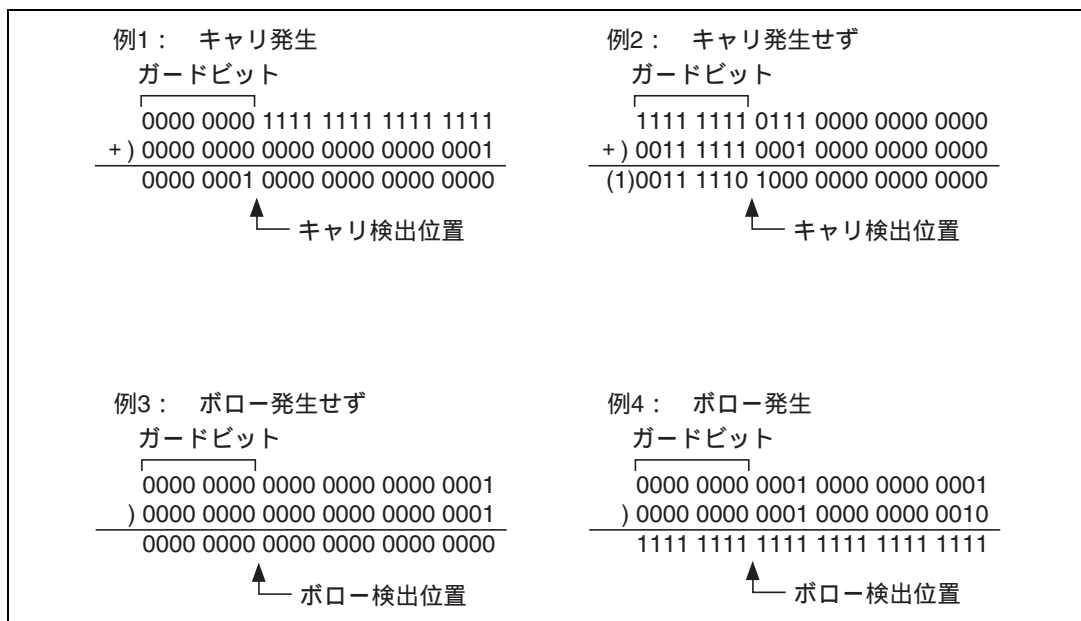


図 5.3 キャリとボローの発生例

## (b) 負値モード： CS2 ~ CS0 = 001

DC ビットは演算結果の MSB ビットの値と同じです。結果が負の値のとき DC ビットは 1 になります。ゼロまたは正の値のとき DC ビットは 0 になります。ALU 算術演算は常に 40 ビットで演算します。そのため正か負かの符号ビットは、デスティネーションオペランドの MSB ビットではなく、演算結果のガードビットを含めた MSB ビットで判定されます。正負の判定例を図 5.4 に示します。このモードの DC ビットは、状態ビットの N ビットの値と同じです。

<p>例1： 負の値</p> <p>ガードビット</p> <pre> 1100 0000 0000 0000 0000 0000 +) 0000 0000 0000 0000 0000 0001 ----- 1100 0000 0000 0000 0000 0001 </pre> <p>↑ 符号ビット</p>	<p>例2： 正の値</p> <p>ガードビット</p> <pre> 0011 0000 0000 0000 0000 0000 +) 0000 0000 1000 0000 0000 0001 ----- 0011 0000 1000 0000 0000 0001 </pre> <p>↑ 符号ビット</p>
-------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------

図 5.4 正負の判定例

## (c) ゼロ値モード： CS2 ~ CS0 = 010

DC ビットは演算結果がゼロかどうかを表します。結果がゼロのとき DC ビットは 1 になり、結果がゼロでないとき DC ビットは 0 になります。このモードの DC ビットは、状態ビットの Z ビットの値と同じです。

## (d) オーバフローモード： CS2 ~ CS0 = 011

DC ビットは演算の結果、オーバフローが発生したかどうかを表します。演算の結果がガードビットを除いて、デスティネーションレジスタの範囲を超えた場合、DC ビットが 1 にセットされます。DC ビットは、ガードビットがあっても、ガードビットがないと考えてオーバフローを判定します。そのため大きな数がガードビットを使う場合は DC ビットが常に 1 にセットされます。このモードの DC ビットは、状態ビットの V ビットの値と同じです。オーバフローの判定例を図 5.5 に示します。

<p>例1： オーバフロー発生</p> <p>ガードビット</p> <pre> 1111 1111 1111 1111 1111 1111 +) 1111 1111 1000 0000 0000 0000 ----- 1111 1111 0111 1111 1111 1111 </pre> <p>↑ オーバフロー検出範囲</p>	<p>例2： オーバフロー発生せず</p> <p>ガードビット</p> <pre> 1111 1111 1111 1111 1111 1111 +) 1111 1111 1000 0000 0000 0001 ----- 1111 1111 1000 0000 0000 0000 </pre> <p>↑ オーバフロー検出範囲</p>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------

図 5.5 オーバフローの判定例

## (e) 符号付き大モード： CS2 ~ CS0 = 100

DC ビットは比較命令 PCMP の判定結果、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表します。ソース 1 データがソース 2 データより大きいときは比較の結果が正の値になるため、このモードは負値モードと似ています。しかしソース 1 データがソース 2 データ

タより大きいときでも、デスティネーションオペランドの範囲を超えたときは、比較の結果の符号が負の値になります。DC ビットはこの場合更新されます。このモードの DC ビットは、状態ビットの GT ビットの値と同じです。このモードでの DC ビットを式で定義すると次のようになります。ただし、VR は結果がガードビット領域も含めてデスティネーションオペランドの表示範囲を超えた場合に真となる値です。

$$\text{DC ビット} = \sim \{ (\text{N ビット} \quad \text{VR}) \mid \text{Z ビット} \}$$

DC ビットは、このモードで PCMP 命令を実行させると、SH コア命令の CMP/GT 命令の結果を表す T ビットと同じ値になります。このモードでは、PCMP 命令以外でも上記定義に従って DC ビットは更新されます。

### (f) 符号付き以上モード： CS2 ~ CS0 = 101

DC ビットは、比較命令 PCMP の実行結果、ソース 1 データ (符号付き) がソース 2 データ (符号付き) より大きいかまたは等しいか、あるいはそうでないかを表します。そのため PCMP 命令は、このモードで DC ビットを判定するまえに、実行されます。このモードは、等しいかどうかを除いて符号付き大モードと似ています。このモードでの DC ビットを式で定義すると次のようになります。ただし、VR は結果がガードビット領域も含めてデスティネーションオペランドの表示範囲を超えた場合に真となる値です。

$$\text{DC ビット} = \sim (\text{N ビット} \quad \text{VR})$$

DC ビットは、このモードで PCMP 命令を実行させると、SH コア命令の CMP/GE 命令の結果を表す T ビットと同じ値になります。このモードでは、PCMP 命令以外でも上記定義に従って DC ビットは更新されます。

### (4) 状態ビット

状態ビットは次のように設定されます。

N ビット (Negative bit、負値ビット) は CS ビットで負値モードを指定したときの DC ビットの値と同じです。演算結果が負の値のとき N ビットは 1 になります。ゼロまたは正の値のとき N ビットは 0 になります。

Z ビット (Zero bit、ゼロビット) は CS ビットでゼロ値モードを指定したときの DC ビットの値と同じです。結果がゼロのとき Z ビットは 1 になり、結果がゼロでないとき Z ビットは 0 になります。

V ビット (Overflow bit、オーバフロービット) V ビットは CS ビットでオーバフローモードを指定したときの DC ビットの値と同じです。演算の結果、カードビットを除くデスティネーションレジスタの範囲を超えた場合、V ビットが 1 にセットされます。それ以外は 0 にクリアされます。

GT ビット (Greater than bit、符号付き大ビット) は CS ビットで符号付き大モードを指定したときの DC ビットの値と同じです。比較の結果、ソース 1 データがソース 2 データより大きいとき、GT ビットが 1 にセットされます。それ以外は 0 にクリアされます。

### (5) オーバフロー防止機能 (飽和演算)

SR レジスタの S ビットを 1 にセットすると、DSP ユニットで実行されるすべての ALU 算術演算で、オーバフロー防止機能が実行されます。演算結果がオーバフローしたとき最大値 (正) または最小値 (負) が格納されます。

## 5.2 ALU 整数演算

ALU 整数演算は基本的には、上位ワード (上位 16 ビット、ビット 31~16) とガードビット 8 ビットとの 24 ビットの演算です。ALU 整数算術演算では、ソースオペランドの下位ワード (下位 16 ビット、ビット 15~0) 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 0 クリアされます。ソースオペランドにガードビットがない場合は符号ビットがガードビットとして拡張されて格納されます。デスティネーションオペランドにガードビットがない場合は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格納されます。

整数演算は基本的に ALU 固定小数点算術演算と同じです。整数演算の演算命令はインクリメントとデクリメント命令の 2 種類しかなく、第 2 オペランドは実質的には +1 か -1 です。16 ビットの整数データ (ワードデータ) が DSP レジスタにロードされ、上位ワードに格納されます。そしては DSP レジスタの上位ワードを使って演算されます。ガードビットがある場合は、ガードビットも有効です。これらの動作は、パイプラインの流れの DSP ステージと名付けられた最終ステージで行われます。

ALU 整数算術演算が実行される時は、DSR レジスタの DC、N、Z、V、GT ビットは、基本的には演算の結果で更新されます。これは ALU 固定小数点演算と同じです。

条件付き命令の場合は、指定された条件が成立し、命令が実行されたときでも、条件ビットとフラグは更新されません。無条件命令の場合は、演算結果に従って常に更新されます。

ALU 整数演算の流れを図 5.6 に示します。

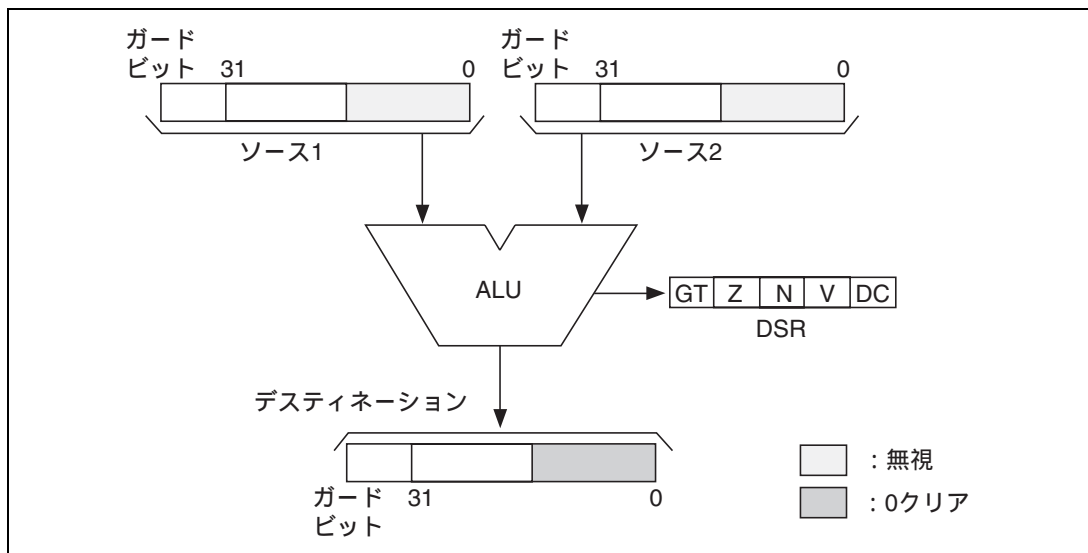


図 5.6 ALU 整数演算の流れ

## 5. DSP の演算機能とデータ転送 (SH3-DSP のみ)

---

ALU 整数演算の種類を表 5.3 に示します。それぞれのオペランドのレジスタとの対応を表 5.4 に示します。

表 5.3 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 インクリメント	Sx	( +1 )	Dz
		( +1 )	Sy	Dz
PDEC	1 デクリメント	Sx	( -1 )	Dz
		( -1 )	Sy	Dz

表 5.4 ALU 整数演算のオペランドとレジスタとの対応

オペランド	X0	X1	Y0	Y1	M0	M1	A0	A1
Sx	Yes	Yes					Yes	Yes
Sy			Yes	Yes	Yes	Yes		
Dz	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

【注】 Yes : オペランドに使えるレジスタです

オーバーフロー防止機能 (飽和演算) の実行は、SR レジスタの S ビットを 1 にセットします。DSP エンジンで実行される ALU 整数算術演算に対して、オーバーフロー防止機能を指定できます。演算結果がオーバーフローしたとき最大値 (正) または最小値 (負) が格納されます。



## 5.3 ALU 論理演算

### (1) 演算機能

ALU 論理演算もレジスタ間で実行されます。それぞれのソース、デスティネーションオペランドは、独立に、DSP レジスタの一つを選べます。このタイプの演算はそれぞれのオペランドの上位ワードだけを使います。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 0 クリアされます。これらの動作は、パイプラインの流れの DSP ステージと名付けられた最終ステージで行われます。

ALU 論理演算が実行されると、DSR レジスタの DC ビット、N、Z、V、GT フラグは、基本的には演算の結果で更新されます。条件付き命令の場合は、指定された条件が成立し、命令が実行されたときでも、条件ビットとフラグは更新されません。無条件命令の場合は、演算結果に従って常に更新されます。DC ビットは CS ビットの指定に従って更新されます。ALU 論理演算の流れを図 5.7 に示します。

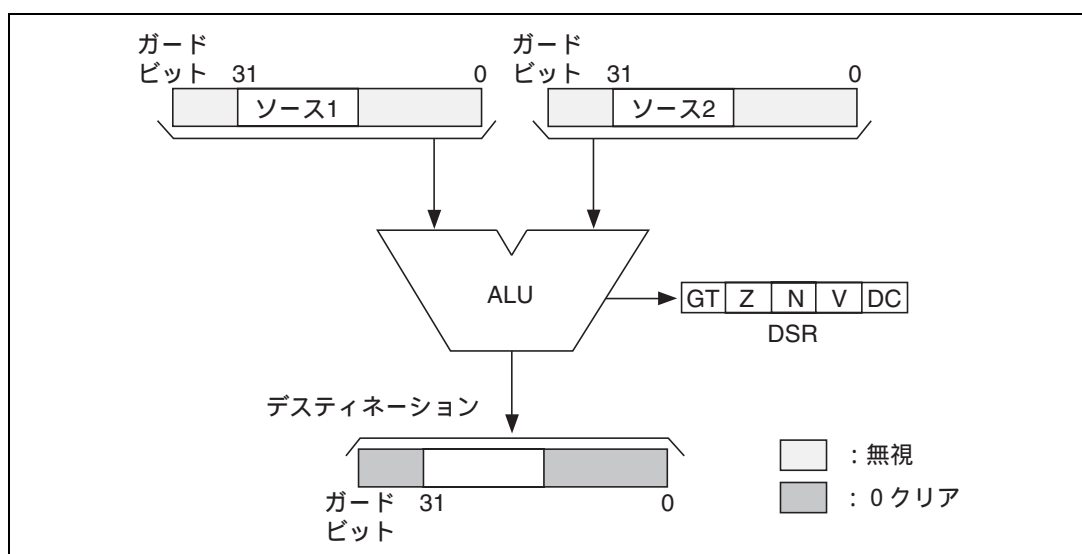


図 5.7 ALU 論理演算の流れ

### (2) 命令とオペランド

ALU 論理演算の種類を表 5.5 に示します。それぞれのオペランドのレジスタとの対応は、ALU 固定小数点演算と同じで、表 5.6 に示します。

表 5.5 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	論理積	Sx	Sy	Dz
POR	論理和	Sx	Sy	Dz
PXOR	排他的論理和	Sx	Sy	Dz

## 5. DSP の演算機能とデータ転送 (SH3-DSP のみ)

---

表 5.6 ALU 論理演算のオペランドとレジスタとの対応

オペランド	X0	X1	Y0	Y1	M0	M1	A0	A1
Sx	Yes	Yes					Yes	Yes
Sy			Yes	Yes	Yes	Yes		
Dz	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

【注】 Yes : オペランドに使えるレジスタです

### (3) DC ビット

論理演算の DC ビットは次のように設定されます。

#### (a) キャリ/ボローモード : CS2 ~ CS0 = 000

DC ビットは常に 0 にクリアされます。

#### (b) 負値モード : CS2 ~ CS0 = 001

DC ビットは演算結果のビット 31 の値になります。このモードの DC ビットは N ビットの値と同じです。

#### (c) ゼロ値モード : CS2 ~ CS0 = 010

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。このモードの DC ビットは、Z ビットの値と同じです。

#### (d) オーバフローモード : CS2 ~ CS0 = 011

DC ビットは常に 0 にクリアされます。このモードの DC ビットは、V ビットの値と同じです。

#### (e) 符号付き大モード : CS2 ~ CS0 = 100

DC ビットは常に 0 にクリアされます。このモードの DC ビットは、GT ビットの値と同じです。

#### (f) 符号付き以上モード : CS2 ~ CS0 = 101

DC ビットは常に 0 にクリアされます。

### (4) 状態ビット

状態ビットは次のように設定されます。

- N ビットは演算結果のビット 31 の値になります。
- Z 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。
- V ビットは常に 0 にクリアされます。
- GT ビットは常に 0 にクリアされます。

## 5.4 固定小数点乗算

DSP 命令の乗算は、符号付き単精度乗算です。この演算は 1 サイクルで処理が完了します。もし倍精度の乗算が必要な場合は、従来の SuperH マイコンの倍精度乗算を使います。

乗算の結果は基本的に 32 ビットの演算結果が得られます。デスティネーションオペランドにガードビットを持つレジスタを指定した場合は、符号拡張されます。

DSP 命令の乗算は整数の計算ではなく、固定小数点算術演算です。そのため、それぞれ、定数と被乗数の上位ワードが、MAC 演算器に入力されます。従来の SuperH マイコンの乗算では、両方のオペランドの下位ワードが MAC 演算器に入力されます。演算結果は SuperH マイコンの場合と異なります。SuperH マイコンの乗算の結果はデスティネーションの LSB に合わせられますが、固定小数点乗算の演算結果は MSB に合わせられます。そのため固定小数点乗算の演算結果の LSB は常に 0 になります。

固定小数点乗算の流れを図 5.8 に示します。

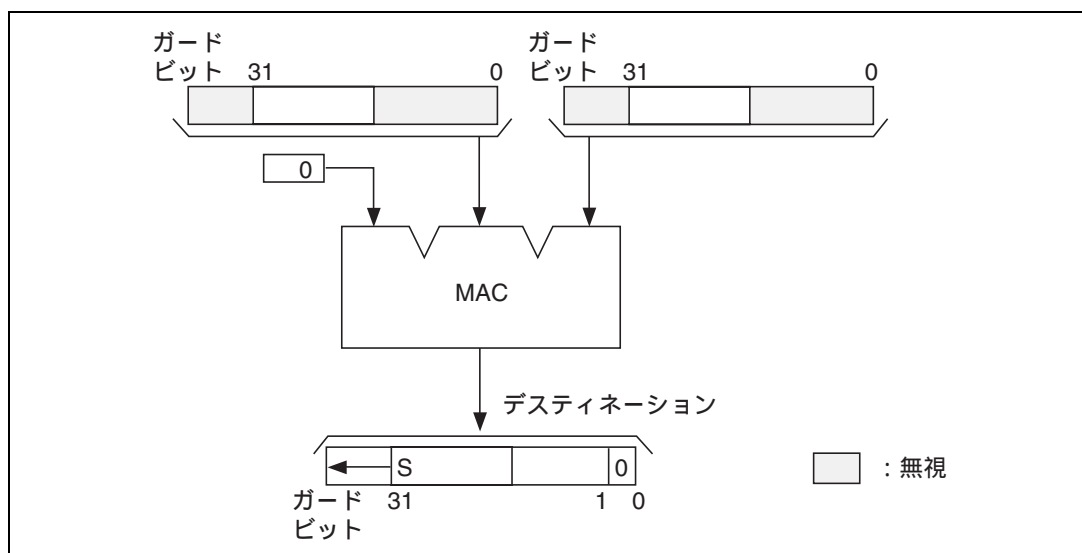


図 5.8 固定小数点乗算の流れ

## 5. DSP の演算機能とデータ転送 (SH3-DSP のみ)

---

固定小数点乗算の種類を表 5.7 に示します。それぞれのオペランドのレジスタとの対応を表 5.8 に示します。

表 5.7 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 5.8 固定小数点乗算のオペランドとレジスタとの対応

オペランド	X0	X1	Y0	Y1	M0	M1	A0	A1
Sx	Yes	Yes					Yes	Yes
Sy			Yes	Yes	Yes	Yes		
Dz	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

【注】 Yes : オペランドに使えるレジスタです

DSP 命令の固定小数点乗算は 16 ビット × 16 ビットの単精度演算を 1 サイクルで完了します。これ以外の乗算は従来の SuperH マイコンの乗算と同じです。

乗算命令は DSR レジスタのどの状態ビット、DC、N、Z、V、GT ビットも更新しません。

オーバーフロー防止機能（飽和演算）は DSP 命令の乗算でも有効です。SR レジスタの S ビットを 1 にセットして指定します。演算結果の値がオーバーフローまたはアンダフローしたときそれぞれ最大値または最小値になります。DSP 命令の固定小数点乗算では、H'8000 × H'8000 ( (-1.0) × (-1.0) ) の場合だけ、オーバーフローが発生します。S ビットが 0 のとき、この演算結果は H'8000 0000 となり、これは -1.0 を意味し、正しい値 +1.0 になりません。S ビットが 1 のとき、オーバーフロー防止機能が働いて、この結果は H'00 7FFF FFFF となります。

## 5.5 シフト演算

シフト演算のシフト量は、レジスタで指定するかまたは直接イミディエイト値で指定します。その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す。シフト演算には算術シフトと論理シフトの 2 種類があります。演算の種類を表 5.9 に示します。イミディエイトオペランドを除いたそれぞれのオペランドのレジスタとの対応は ALU 固定小数点演算と同じです。対応を表 5.10 に示します。

表 5.9 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 #lmm, Dz	イミディエイトデータ付き算術シフト	Dz	lmm1	Dz
PSHL #lmm, Dz	イミディエイトデータ付き論理シフト	Dz	lmm2	Dz

- 32<=lmm1<= + 32、 - 16<=lmm2<= + 16

表 5.10 シフト演算のオペランドとレジスタとの対応

オペランド	X0	X1	Y0	Y1	M0	M1	A0	A1
Sx	Yes	Yes					Yes	Yes
Sy			Yes	Yes	Yes	Yes		
Dz	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

【注】 Yes : オペランドに使えるレジスタです

### 5.5.1 算術シフト演算

#### (1) 演算機能

ALU 算術シフト演算は、32 ビット精度とガードビット 8 ビットとの 40 ビットの演算です。基本的にはレジスタ間で実行されます。ソースオペランドにガードビットがない場合は符号ビットがガードビットとして転記されます。デスティネーションオペランドにガードビットがない場合は演算結果の下位 32 ビットがデスティネーションレジスタに格納されます。

この算術シフトではソース 1 オペランドとデスティネーションオペランドはすべてのビットが有効です。シフト量を指定するソース 2 オペランドは整数データです。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します。有効なシフト量は - 32 から + 32 までです。ここで、負の値は右のシフトを意味し、正の値は左のシフトを意味します。ソース 2 オペランドとして - 64 から + 63 までを指定することはできませんが、有効なシフト量は - 32 から + 32 までです。無効な数値を指定した場合の結果は保証されません。シフト量をイミディエイト値で指定した場合は、ソース 1 オペランドはデスティネーションオペランドと同じでなければなりません。この演算動作は、固定小数点演算と同じように、パイプラインの流れの最後の DSP ステージで実行されます。

算術シフト演算が実行される時は、DSR レジスタの DC、N、Z、V、GT ビットは、基本的には演算の結果に従って更新されます。これは ALU 固定小数点演算と同じです。条件付き命令の場合は、指定された条件が成立し、命令が実行されたときでも、状態ビットは更新されません。無条件命令の場合は、演算結果に従って常に更新されます。

算術シフト演算の流れを図 5.9 に示します。

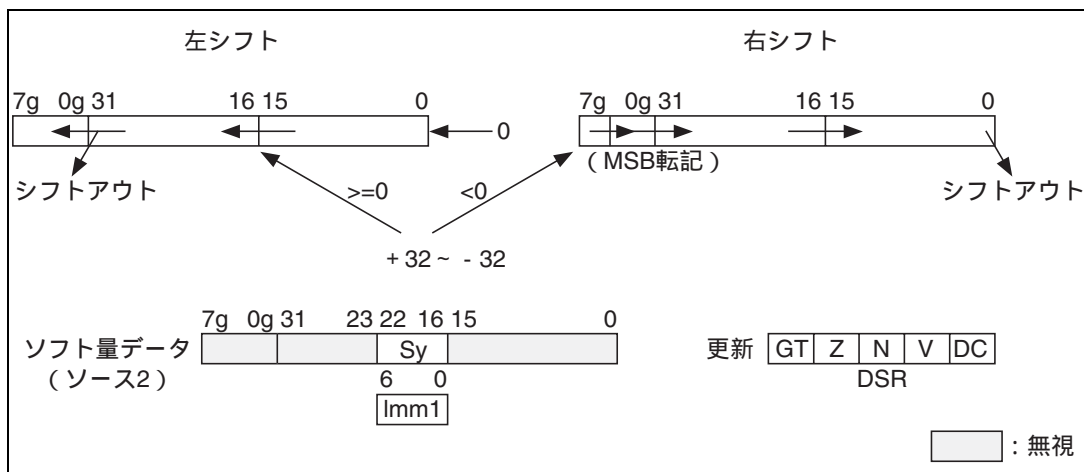


図 5.9 算術シフト演算の流れ

(2) DC ビット

DC ビットは CS ビットで指定されたモードに従って、次のように更新されます。

(a) キャリ/ボローモード:  $CS2 \sim CS0 = 000$

DC ビットは演算の結果、最後にシフトして押し出されたビットの値になります。

(b) 負値モード:  $CS2 \sim CS0 = 001$

ビットは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。このモードの DC ビットは N ビットの値と同じです。

(c) ゼロ値モード:  $CS2 \sim CS0 = 010$

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。このモードの DC ビットは、Z ビットの値と同じです。

(d) オーバフローモード:  $CS2 \sim CS0 = 011$

オーバフローが発生したとき 1 にセットされます。このモードの DC ビットは、V ビットの値と同じです。

(e) 符号付き超モード:  $CS2 \sim CS0 = 100$

DC ビットは常に 0 にクリアされます。このモードの DC ビットは、GT ビットの値と同じです。

(f) 符号付き以上モード:  $CS2 \sim CS0 = 101$

DC ビットは常に 0 にクリアされます。

### (3) 状態ビット

状態ビットは次のように更新されます。

N ビットは ALU 固定小数点算術演算の結果と同じです。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。

Z ビットは ALU 固定小数点算術演算の結果と同じです。演算結果がゼロのとき 1 にセットされ、それ以外は 0 にクリアされます。

V ビットは ALU 固定小数点算術演算の結果と同じです。オーバーフローが発生したとき 1 にセットされます。

GT ビットは常に 0 にクリアされます。

### (4) オーバフロー防止機能 (飽和演算)

SR レジスタの S ビットを 1 にセットすると、DSP ユニットで実行される算術シフト演算で、オーバーフロー防止機能が実行されます。演算結果がオーバーフローしたとき最大値 (正) または最小値 (負) が格納されます。

## 5.5.2 論理シフト演算

### (1) 演算機能

論理シフト演算は、ソース 1 オペランドとデスティネーションオペランドの上位ワードを使います。オペランドのガードビットと下位ワードは、ALU 論理演算と同じに、無視されます。シフト量を指定するソース 2 オペランドは整数データです。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します。有効なシフト量は -16 から +16 までです。ここで、負の値は右のシフトを意味し、正の値は左のシフトを意味します。ソース 2 オペランドとして -32 から +31 までを指定することはできませんが、有効なシフト量は -16 から +16 までですので、無効な数値を指定した場合の結果は保証されません。シフト量をイミディエイト値で指定した場合は、ソース 1 オペランドはデスティネーションオペランドと同じでなければなりません。この演算動作は、パイプラインの流れの最後の DSP ステージで実行されます。

論理シフト演算が実行されるときは、DSR レジスタの DC、N、Z、V、GT ビットは、基本的には演算の結果に従って更新されます。これは ALU 論理演算と同じです。条件付き命令の場合は、指定された条件が成立し、命令が実行されたときでも、状態ビットは更新されません。無条件命令の場合は、演算結果に従って常に更新されます。

論理シフト演算の流れを図 5.10 に示します。

## 5. DSP の演算機能とデータ転送 (SH3-DSP のみ)

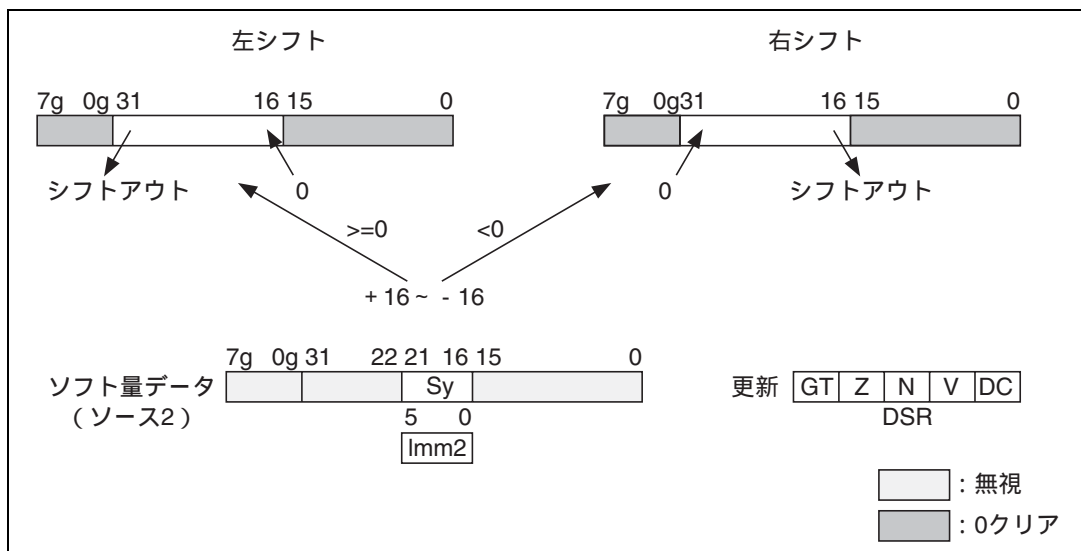


図 5.10 論理シフト演算の流れ

### (2) DC ビット

CS ビットで指定されたモードに従って、次のように更新されます。

#### (a) キャリ/ボロモード： CS2 ~ CS0 = 000

DC ビットは演算の結果、最後にシフトして押し出されたビットの値になります。

#### (b) 負値モード： CS2 ~ CS0 = 001

DC ビットは演算結果のビット 31 の値が格納されます。このモードの DC ビットは N ビットの値と同じです。

#### (c) ゼロ値モード： CS2 ~ CS0 = 010

DC ビットは演算結果がすべてゼロのとき 1 にセットされます。それ以外は 0 にクリアされます。このモードの DC ビットは、Z ビットの値と同じです。

#### (d) オーバフローモード： CS2 ~ CS0 = 011

DC ビットは常に 0 にクリアされます。このモードの DC ビットは、V ビットの値と同じです。

#### (e) 符号付き超モード： CS2 ~ CS0 = 100

DC ビットは常に 0 にクリアされます。このモードの DC ビットは、GT ビットの値と同じです。

#### (f) 符号付き以上モード： CS2 ~ CS0 = 101

DC ビットは常に 0 にクリアされます。



(3) 状態ビット

状態ビットは次のように更新されます。

- N ビットは ALU 論理演算の結果と同じです。演算結果のビット 31 の値が格納されます。
- Z ビットは ALU 論理演算の結果と同じです。演算結果がすべてゼロのとき 1 にセットされます。それ以外は 0 にクリアされます。
- V ビットは常に 0 にクリアされます。
- GT ビットは常に 0 にクリアされます。

## 5.6 MSB 検出命令

### (1) 演算機能

MSB 検出命令 (PDMSB: Most Significant Bit Detection) は、データを正規化するためのシフト量を求めるものです。

演算結果は、ALU 整数演算と同じに、基本的には上位 16 精度と 8 ビットのガードビットとの 24 ビットが有効です。デスティネーションオペランドがガードビットのないレジスタの場合は、デスティネーションレジスタ上位 16 ビットに格納されます。

MSB 検出命令はソースオペランドのすべてのビットを対象にしていますが、演算結果は整数データとして求められます。これは、正規化のためのシフト量は、算術シフト演算の整数データが必要なためです。この演算動作は、固定小数点演算と同じように、パイプラインの流れの最後の DSP ステージで実行されます。

PDMSB 命令が実行される時は、DSR レジスタの DC、N、Z、V、GT ビットは、基本的には演算の結果に従って更新されます。条件付き命令の場合は、指定された条件が成立し、命令が実行されたときでも、状態ビットは更新されません。無条件命令の場合は、演算結果に従って常に更新されます。

MSB 検出命令の流れを図 5.11 に示します。ソースデータとデスティネーションデータとの関係を表 5.11 に示します。

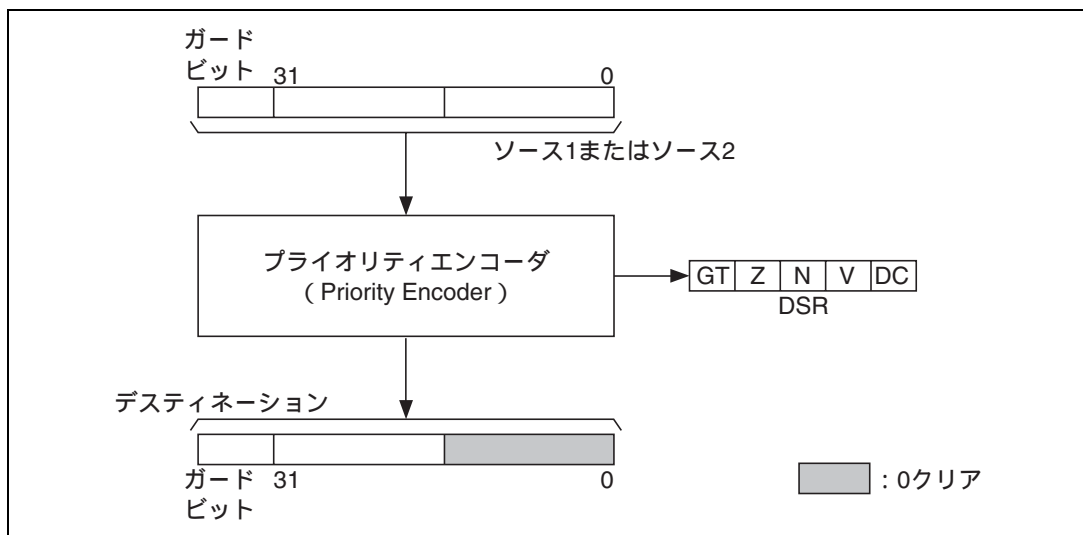


図 5.11 MSB 検出の流れ

表 5.11 ソースデータとデスティネーション結果との関係

ソースデータ										結果												
ガードビット				上位ワード				下位ワード		ガード	上位ワード											
7g	6g	---	1g	0g	31	30	29	28	---	3	2	1	0	7g~0g	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、影響なし

## (2) 命令とオペランド

MSB 検出命令の種類を表 5.12 に示します。それぞれのオペランドのレジスタとの対応は、ALU 固定小数点演算と同じです。その対応を表 5.13 に示します。

表 5.12 MSB 検出の種類

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

## 5. DSP の演算機能とデータ転送 (SH3-DSP のみ)

---

表 5.13 MSB 検出のオペランドとレジスタとの対応

オペランド	X0	X1	Y0	Y1	M0	M1	A0	A1
Sx	Yes	Yes					Yes	Yes
Sy			Yes	Yes	Yes	Yes		
Dz	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

【注】 Yes : オペランドに使えるレジスタです

### (3) DC ビット

DC ビットは CS ビットで指定されたモードに従って、次のように更新されます。

#### (a) キャリ/ボローモード : CS2~CS0 = 000

DC ビットは常に 0 にクリアされます。

#### (b) 負値モード : CS2~CS0 = 001

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。このモードの DC ビットは N ビットの値と同じです。

#### (c) ゼロ値モード : CS2~CS0 = 010

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。このモードの DC ビットは、Z ビットの値と同じです。

#### (d) オーバフローモード : CS2~CS0 = 011

常に 0 にクリアされます。このモードの DC ビットは、V ビットの値と同じです。

#### (e) 符号付き大モード : CS2~CS0 = 100

演算結果が正の値のとき 1 にセットされます。それ以外は 0 にクリアされます。このモードの DC ビットは、GT ビットの値と同じです。

#### (f) 符号付き以上モード : CS2~CS0 = 101

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

### (4) 状態ビット

状態ビットは次のように更新されます。

- N ビットは ALU 整数演算の結果と同じです。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。
- Z ビットは ALU 整数演算の結果と同じです。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。
- V ビットは常に 0 にクリアされます。
- GT ビットは ALU 整数演算の結果と同じです。演算結果が正の値のとき 1 にセットされます。それ以外は 0 にクリアされます。

## 5.7 丸め処理

### (1) 演算機能

DSP エンジンには 32 ビットの数値を 16 ビットに丸める機能があります。ガードビットがある場合は、40 ビットの数値を 24 ビットに丸めます。丸めの命令が実行されると、H'0000 8000 がソースオペランドに加えられ、そのあとで下位ワードは 0 でクリアされます。

丸め処理はソースオペランドとデスティネーションオペランドともにすべてのビットデータを使います。この演算動作は、固定小数点演算と同じように、パイプラインの流れの最後の DSP ステージで実行されます。

丸め処理命令は無条件命令です。そのため、DSR レジスタの DC、N、Z、V、GT ビットは、常に演算の結果に従って更新されます。

丸め処理の流れを図 5.12 に示します。丸め処理の定義を図 5.13 に示します。

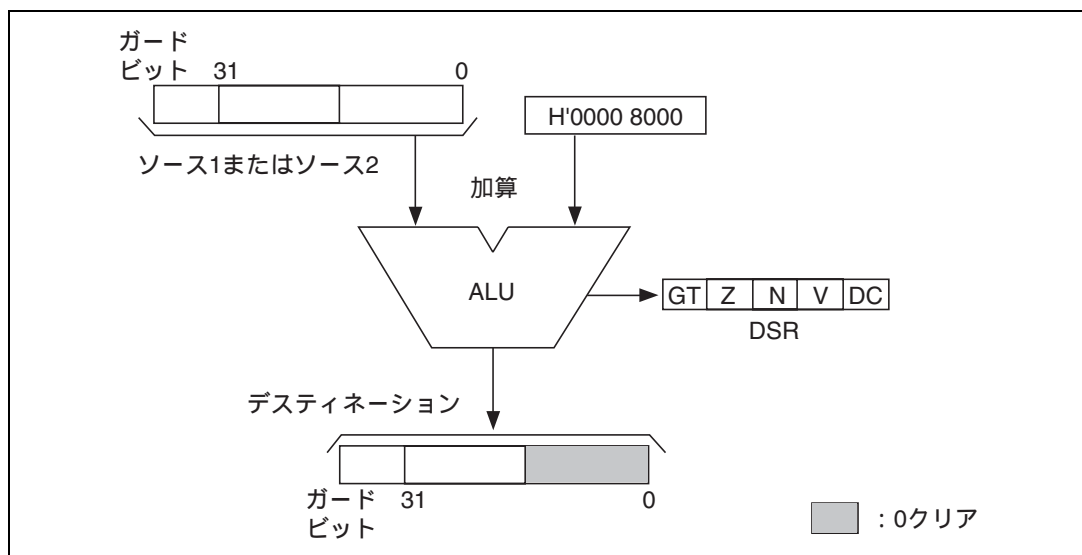


図 5.12 丸め処理の流れ

## 5. DSP の演算機能とデータ転送 (SH3-DSP のみ)

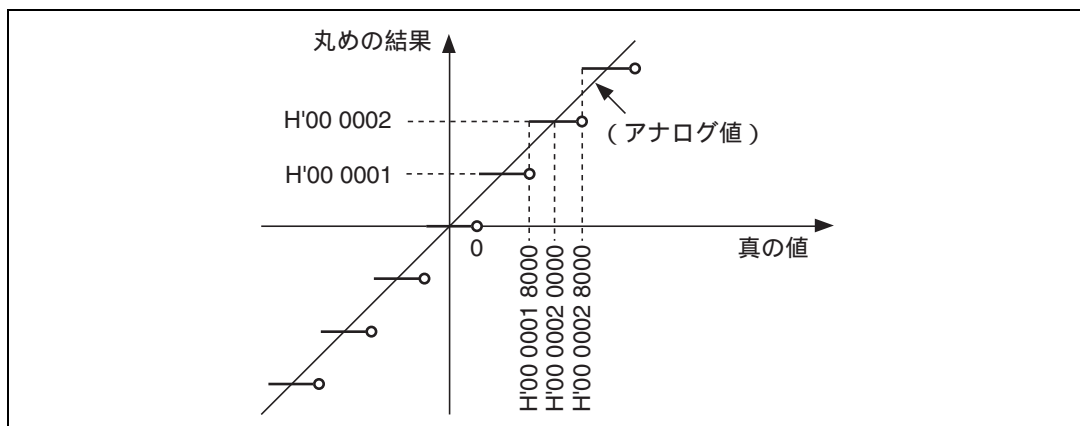


図 5.13 丸め処理の定義

### (2) 命令とオペランド

命令の種類を表 5.14 に示します。オペランドとレジスタの対応は ALU 固定小数点演算と同じです。対応を表 5.15 に示します。

表 5.14 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
PRND	丸め処理	Sx	—	Dz
		—	Sy	Dz

表 5.15 固定小数点乗算のオペランドとレジスタとの対応

オペランド	X0	X1	Y0	Y1	M0	M1	A0	A1
Sx	Yes	Yes					Yes	Yes
Sy			Yes	Yes	Yes	Yes		
Dz	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

【注】 Yes : オペランドに使えるレジスタです

### (3) DC ビット

DC ビットは CS ビットで指定されたモードに従って、次のように更新されます。状態ビットの更新は ALU 固定小数点算術演算と同じです。

#### (a) キャリ/ボローモード : CS2 ~ CS0 = 000

DC ビットは演算の結果、MSB ビットからキャリまたはボローが発生したとき 1 にセットされます。それ以外は 0 にクリアされます。

#### (b) 負値モード : CS2 ~ CS0 = 001

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。このモードの DC ビットは N ビットの値と同じです。

(c) ゼロ値モード : CS2 ~ CS0 = 010

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。このモードの DC ビットは、Z ビットの値と同じです。

(d) オーバフローモード : CS2 ~ CS0 = 011

DC ビットはオーバフローが発生すると 1 にセットされます。それ以外は 0 にクリアされます。このモードの DC ビットは、V ビットの値と同じです。

(e) 符号付き大モード : CS2 ~ CS0 = 100

演算結果が正の値のとき 1 にセットされます。それ以外は 0 にクリアされます。このモードの DC ビットは、GT ビットの値と同じです。

(f) 符号付き以上モード : CS2 ~ CS0 = 101

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

(4) 状態ビット

状態ビットは次のように更新されます。状態ビットの更新は ALU 固定小数点算術演算と同じです。

- N ビットは ALU 固定小数点算術演算の結果と同じです。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。
- Z ビットは ALU 固定小数点算術演算の結果と同じです。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。
- V ビットは ALU 固定小数点算術演算の結果と同じです。オーバフローが発生したとき 1 にセットされます。それ以外は 0 にクリアされます。
- GT ビットは ALU 固定小数点算術演算 ALU 整数演算の結果と同じです。演算結果が正の値のとき 1 にセットされます。それ以外は 0 にクリアされます。

(5) オーバフロー防止機能 (飽和演算)

SR レジスタの S ビットを 1 にセットすると、DSP ユニットで実行されるすべての丸め処理に対して、オーバフロー防止機能を実行します。演算結果の値がオーバフローしたときそれぞれ正の最大値または負の最小値になります。

## 5.8 状態選択ビット (CS) と DSP 状態ビット (DC)

DSP 命令には無条件命令と条件付き命令があります。無条件命令は DSP 状態ビット (DC) に関係なく実行され、条件付き命令は DC ビットを判定して実行するかしないかが決まります。無条件命令では DSR レジスタの DC ビットおよび状態ビット (N、Z、V、GT) は ALU 演算またはシフト演算の結果によって更新されます。条件付き命令は実行するしないにかかわらず、DC ビットおよび状態ビット (N、Z、V、GT) を更新しません。DC ビットは状態選択ビット (CS) の指定に従って更新されます。更新は、算術演算、論理演算、算術シフト、論理シフトによってそれぞれ異なります。CS ビットと DC ビットの関係を表 5.16 に示します。

表 5.16 状態選択ビット (CS) と DSP 状態ビット (DC)

CS ビット			状態モード	説明
2	1	0		
0	0	0	キャリ/ボロモード	ALU 算術演算の結果キャリまたはボロが生じたとき、DC ビットは 1 にセットされます。それ以外は 0 にクリアされます。 論理演算では DC ビットは常に 0 にクリアされます。 シフト演算 (PSHA、PSHL 命令) のとき、最後にシフトアウトされた (外に出た) ビットが DC ビットに転記されます。
0	0	1	負値モード	ALU 算術演算または算術シフト (PSHA) 演算のとき、ガードビットを含めて、結果の MSB ビットが DC ビットに転記されます。 ALU 論理演算または論理シフト (PSHL) 演算のとき、ガードビットを除いて、結果の MSB ビットが DC ビットに転記されます。
0	1	0	ゼロ値モード	ALU 演算またはシフト演算の結果がすべてゼロ (0) のとき、DC ビットは 1 にセットされます。それ以外は 0 にクリアされます。
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) 演算のとき、ガードビットを除いて、演算結果がデスティネーションレジスタの値の範囲を超えたとき、DC ビットは 1 にセットされます。それ以外は 0 にクリアされます。 ALU 論理演算または論理シフト (PSHL) 演算のとき、DC ビットは常に 0 にクリアされます。
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と似ていますが、演算結果がゼロ (0) のとき DC ビットは 0 にクリアされます。ガードビット部分を含めても演算結果が表現可能範囲を超えたとき、真となる状態を VR とすると以下のように計算されます。 DC ビット = $\sim \{(N \text{ ビット} \wedge VR)   Z \text{ ビット}\}$ ; 算術演算の場合 DC ビット = 0; 論理演算の場合
1	0	1	符号付き以上モード	ALU 算術演算または算術シフト (PSHA) 演算のとき、かつ結果がオーバフローしないとき、負値モードの DC ビットを反転した値になります。ガードビット部分を含めても結果が表現可能範囲を超えたとき、負値モードの DC ビットと同じ値になります。 ALU 論理演算または論理シフト (PSHL) 演算のとき、DC ビットは常に 0 にクリアされます。ガードビット部分を含めても演算結果が表現可能範囲を超えたとき、真となる状態を VR とすると、以下のように計算されます。 DC ビット = $\sim (N \text{ ビット} \wedge VR)$ ; 算術演算の場合 DC ビット = 0; 論理演算の場合
1	1	0	予約コード	
1	1	1		



## 5.9 オーバフロー防止機能 (飽和演算)

オーバフロー防止機能 (飽和演算) は SR レジスタの S ビットで指定します。この機能は DSP ユニットで実行されるすべての算術演算、および従来の SH-1、SH-2 マイコンで実行される積和演算に有効です。演算結果がガードビットを除いて表現できる 2 の補数の範囲を超えたときオーバフローが発生します。

固定小数点算術演算のオーバフローの定義を表 5.17 に示し、整数算術演算のオーバフローの定義を表 5.18 に示します。従来の SuperH マイコンからサポートされている積和命令 (MAC) は 64 ビットレジスタ (MACH、MACL) で演算しているため、オーバフローの値および最大値、最小値は異なり、従来と全く同じ定義になっています。

表 5.17 固定小数点算術演算のオーバフローの定義

符号	オーバフロー状態	最大値 / 最小値	16 進表示
正	結果 $> 1 - 2^{-31}$	$1 - 2^{-31}$	007FFFFFFF
負	結果 $< -1$	-1	FF80000000

表 5.18 整数算術演算のオーバフローの定義

符号	オーバフロー状態	最大値 / 最小値	16 進表示
正	結果 $> 2^{15} - 1$	$2^{15} - 1$	007FFF****
負	結果 $< -2^{15}$	$-2^{15}$	FF8000****

【注】\* : ' Don't care '、影響なし

オーバフロー防止機能を指定した場合はオーバフローは発生しません。このとき、オーバフロービット (V) はセットされません。CS ビットでオーバフローモードを指定したときの DC ビットもセットされません。

## 5.10 データ転送

SH-DSP は DSP ユニットで DSP レジスタと内蔵メモリとの間で最大 2 つのデータを同時に並行して転送することができます。SH-DSP には次の 3 つのデータ転送があります。

- (1) X、Yメモリデータ転送： XDBバス、YDBバスを使ってX、Yメモリとデータ転送
- (a) ダブルデータ転送： データ転送だけ、どちらか一方にのみの転送も可
- (b) パラレルデータ転送： ALU演算や乗算と並行処理をしながらのデータ転送

- (2) シングルデータ転送： LDBバスを使って内蔵メモリとデータ転送  
データ転送命令は DSR レジスタの状態ビットを更新しません。  
それぞれの機能を表 5.19 に示します。

表 5.19 データ転送の機能

種類	使用バス	転送データ長	ALU 演算との並行処理	データ転送の並行処理	命令長
X、Yメモリ データ転送	XDB バス YDB バス	16 ビット	なし (ダブル)	なし (XDB バスか YDB バス)	16 ビット
				あり (XDB バスと YDB バス)	16 ビット
			あり (パラレル)	なし (XDB バスか YDB バス)	32 ビット
				あり (XDB バスと YDB バス)	32 ビット
シングル データ転送	LDB バス	32 ビット 16 ビット	なし	なし	16 ビット

### 5.10.1 X、Yメモリデータ転送

X、Yメモリデータ転送は 2 つのデータ転送を同時に並行して実行することができ、データ転送と DSP データ演算を同時に並行して実行することができます。DSP データ演算と転送を同時に並行して実行させるには 32 ビットの命令コードが必要です。これをパラレルデータ転送と呼びます。X、Yメモリデータ転送だけを実行する場合は 16 ビットの命令コードです。これをダブルデータ転送と呼びます。

データ転送は、Xメモリデータ転送と、Yメモリデータ転送があります。Xメモリデータは X0、X1 レジスタのどちらかにロードされ、Yメモリデータは Y0、Y1 レジスタのどちらかにロードされます。X0、X1、Y0、Y1 レジスタがデスティネーションレジスタになります。デスティネーションレジスタの上位ワードにデータが転送され、下位ワードは自動的に 0 にクリアされます。A0、A1 レジスタの一方をソースレジスタとして、X、Yメモリにデータをストアすることができます。これらのデータ転送はすべてワードデータ (16 ビット) です。ソースレジスタの上位ワードからデータが転送されます。

同時に並行して実行する演算命令に条件付き命令を指定しても、データ転送命令は影響を受けません。

X、Yメモリデータ転送は X、Yメモリのみをアクセスし、他のメモリアリアはアクセスできません。

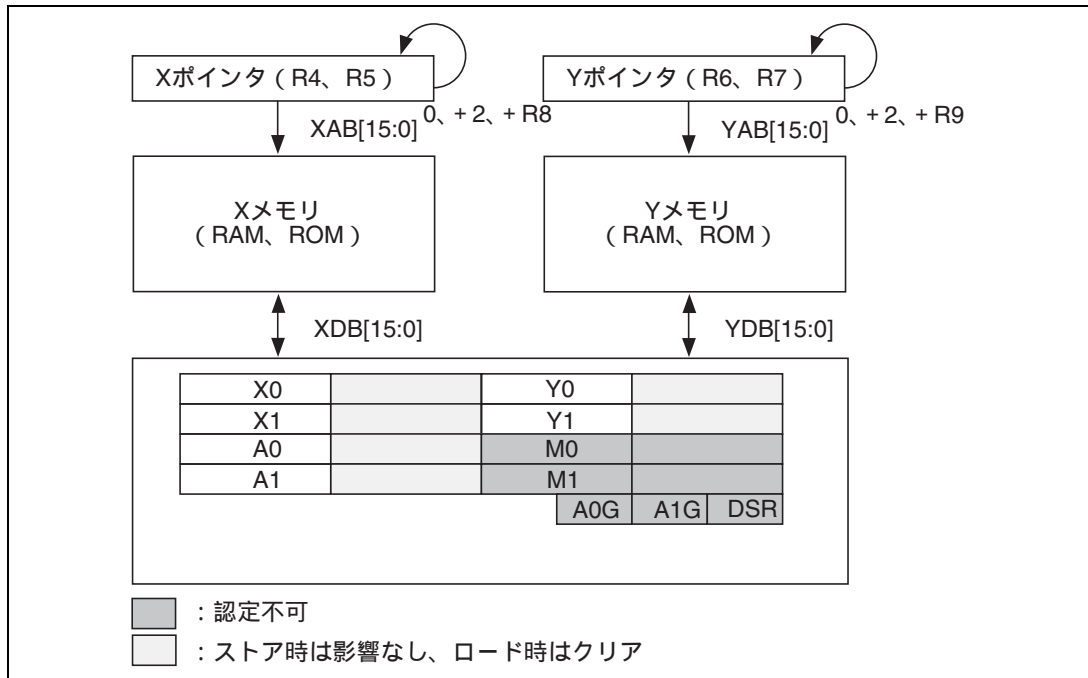


図 5.14 X、Yメモリデータ転送の流れ

### 5.10.2 シングルデータ転送

シングルデータ転送は1つのデータ転送だけを実行します。16ビットの命令コードです。シングルデータ転送はALU演算と同時に並行処理はできません。XメモリをアクセスするXポインタと追加された2つのポインタが有効となり、Yポインタは無効です。従来のSuperHマイコンと同様に、シングルデータ転送は外部エリアを含むすべてのメモリエリアをアクセスできます。DSRレジスタを除く\*DSPレジスタがソースオペランド、デスティネーションオペランドに指定できます。ガードビットレジスタ、A0G、A1Gは独立したレジスタとしてオペランドに指定できます。シングルデータ転送ではXAB、XDB、YAB、YDBバスの代わりにLAB、LDBバスを使うので、LDBバスでデータ転送と命令フェッチの競合が発生します。

シングルデータ転送はワードデータとロングワードデータを取り扱います。ワードデータ転送ではレジスタの上位ワードが有効です。レジスタにデータがロードされる時は上位ワードにロードされ、下位ワードは自動的に0でクリアされ、ガードビットがあればガードビットには符号ビットが拡張されて格納されます。レジスタからストアされる時は上位ワードがストアされます。

ロングワードが転送される時は32ビットが有効になります。ロードされる時ガードビットがあれば、ガードビットには符号ビットが拡張されて格納されます。

ガードビットレジスタがストアされる時は、上位24ビットに符号が拡張されて、LDBバスに読み出されます。ガードビットレジスタ、A0G、A1GレジスタがMOV.S.W命令のデスティネーションレジスタとしてワードデータがロードされる時は、下位バイトがレジスタに書き込まれます。

【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命令でデータの転送が可能です。

5. DSP の演算機能とデータ転送 (SH3-DSP のみ)

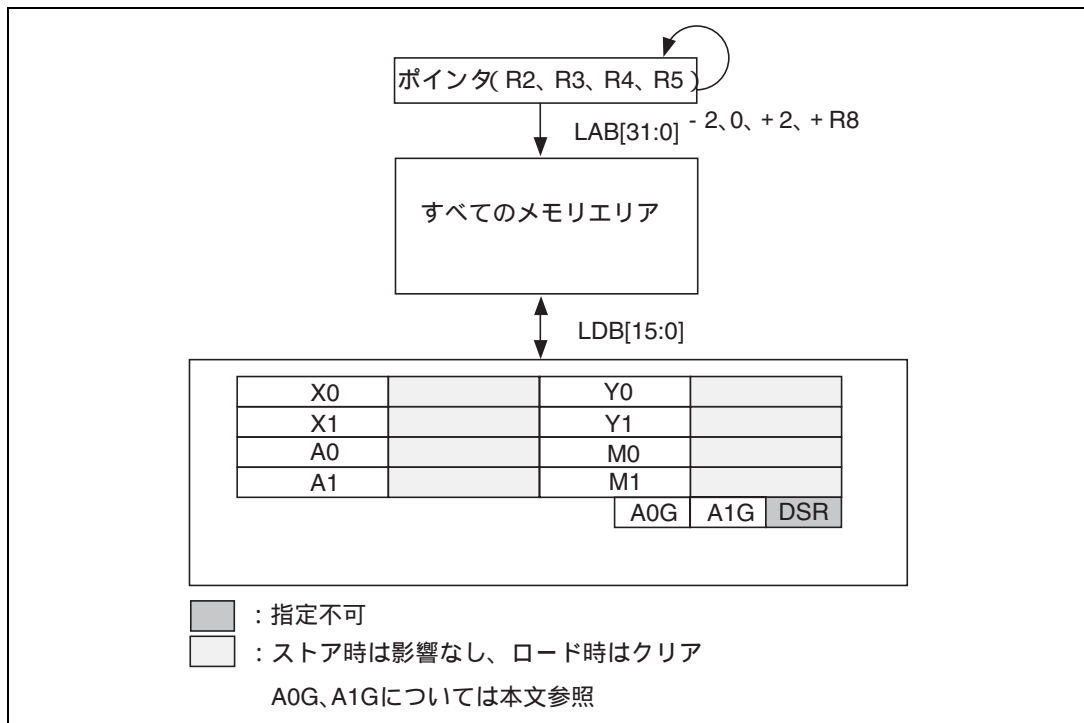


図 5.15 シングルデータ転送の流れ (ワード)

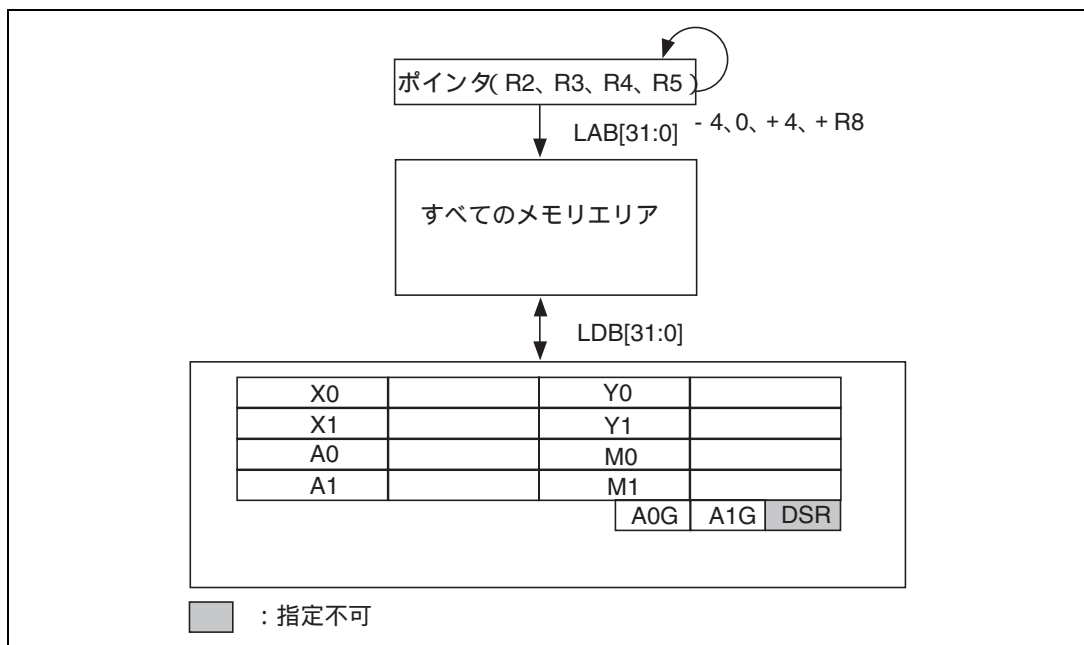


図 5.16 シングルデータ転送の流れ (ロングワード)

データ転送はパイプラインの MA ステージで実行され、DSP 演算は DSP ステージで実行されます。演算機能をストアする命令がデータ演算命令のすぐ次の命令行にある場合は、データ演算命令が終わらない内に次のデータストア命令が始まるため、1つのストールサイクルが挿入されます。このオーバーヘッドサイクルは、1つの命令をデータ演算命令とデータ転送命令の間に追加することによって避けることができます。この例を図 5.17 に示します。

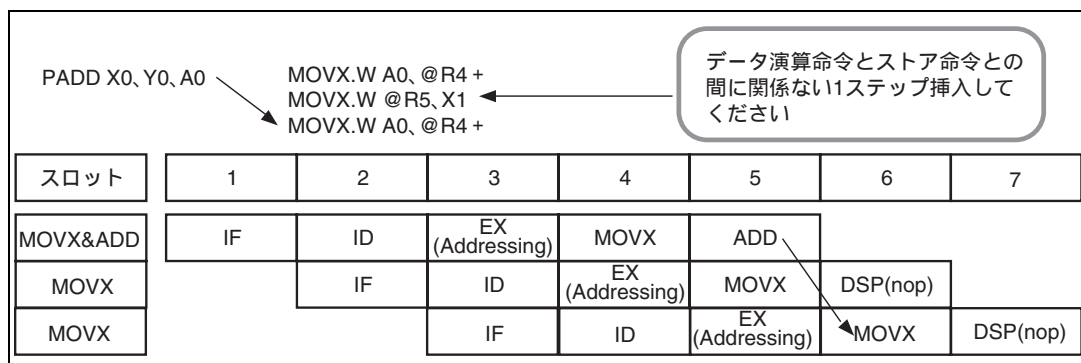


図 5.17 演算とデータストアの命令実行例

## 5.11 オペランド競合

2つ以上の並行処理命令でデスティネーションオペランドに同じレジスタを指定するとデータの競合が発生します。データの競合は次の3つの場合が考えられます。

- (1) ALU演算と乗算で同じデスティネーションオペランドを指定した場合 (Du、Dg)
- (2) XメモリロードとALU演算で同じデスティネーションオペランドを指定した場合 (Dx、Du、Dz)
- (3) YメモリロードとALU演算で同じデスティネーションオペランドを指定した場合 (Dy、Du、Dz)

もし競合した場合の結果は保証されません。競合の発生するオペランドとレジスタの対応を表5.20に示します。

これらの競合を検出できるアセンブラがありますので、機能を選択してアセンブラをお使いください。

表 5.20 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	*	*						
Yメモリ ロード	Ay								
	Iy								
	Dy			*	*				
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、Du、Dz の競合 ) ( Dy、Du、Dz の競合 )

↑            ↑

( Du、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

## 5.12 DSP 繰り返し (ループ) 制御

SH3-DSP は、効率よく繰り返し (ループ) 制御を行うための特別な機構を持っています。SETRC 命令で、繰り返しカウンタ (RC、12 ビット) に繰り返し回数を格納し、RC が 1 になるまで繰り返しプログラム (ループ) を反復する実行モードを設定します。繰り返し動作が終了すると、RC の内容は 0 になります。

繰り返し開始アドレスレジスタ (RS) は、繰り返しループの開始アドレスを格納しています。繰り返し終了レジスタ (RE) は、繰り返し終了アドレスを格納しています。(例外があります。「5.12.1 (1) プログラミングの実際」を参照してください。) 繰り返しカウンタ (RC) は、繰り返し回数を格納しています。この繰り返し制御を実行する手順は次のようになります。

- #1 繰り返し開始アドレスを RS レジスタに設定します。
  - #2 繰り返し終了アドレスを RE レジスタに設定します。
  - #3 繰り返し回数を RC カウンタに設定します。
  - #4 繰り返しプログラム (ループ) を開始します。
- #1 と #2 を実行するために次の命令を使います。

```
LDRS @(disp,PC);および
LDRE @(disp,PC);
```

#3 と #4 を実行するために SETEC 命令を使います。SETEC 命令のオペランドはイミディエイティブまたは汎用レジスタで繰り返し回数を指定します。

```
SETRC #imm; #imm→RC,enable repeat control
SETRC Rm; Rm→RC,enable repeat control
```

#imm は 8 ビットで RC カウンタは 12 ビットです。そのため RC カウンタに 256 以上の数値を指定したいときは、Rm レジスタを使って設定します。プログラム例を次に示します。

```
LDRS RptStart;
LDRE RptEnd;
SETRC #imm;      RC=#imm
instr0;
; instr1~5 executes repeatedly
RptStart:  instr1;
           instr2;
           instr3;
           instr4;
RptEnd:   instr5;
           instr6;
```

この繰り返し命令には次のようないくつかの制限があります。

- (1) SETRC 命令と繰り返しプログラム (ループ) の最初の命令の間には少なくとも 1 命令が必要です。
- (2) LDRS、LDRE 命令を実行したあとで、SETRC 命令を実行してください。

## 5. DSP の演算機能とデータ転送 (SH3-DSP のみ)

- (3) 繰り返しプログラム (ループ) が4命令以上の場合は、繰り返し開始アドレス (先述の例ではinstr1のアドレス) がロングワード境界にないときには、繰り返しのたびに1サイクルのストール (実行待ちサイクル) が発生します。
- (4) 繰り返しプログラム (ループ) が3命令以下の場合、分岐命令 (BRA、BSR、BT、BF、BT/S、BF/S、BSRF、RTS、BRAf、RTE、JSR、JMP)、繰り返し制御命令 (SETRC、LDRS、LDRE)、SR、RS、およびREのロード命令、TRAPAは使えません。もし記述すると、不当命令例外処理が起動され、表5.21に示すアドレス値がSPCにストアされます。

表 5.21 SPC にストアされる値 (1)

条件	位置	SPC にストアされるアドレス
RC>=2	任意	RptStart
RC=1	任意	不正な命令のプログラムアドレス

- (5) 繰り返しプログラム (ループ) が4命令以上の場合、分岐命令 (BRA、BSR、BT、BF、BT/S、BF/S、BSRF、RTS、BRAf、RTE、JSR、JMP)、繰り返し制御命令 (SETRC、LDRS、LDRE)、SR、RS、およびREのロード命令、TRAPAは繰り返しプログラム (ループ) 内の最後の3命令には使えません。もし記述すると、不当命令例外処理が起動され、表5.22に示すアドレス値がSPCにストアされます。繰り返し制御命令 (SETRC、LDRS、LDRE)、SR、RS、およびREのロード命令の場合には、繰り返しモジュールの他の位置にも記述できません。記述した場合の動作は保証しません。

表 5.22 SPC にストアされる PC 値 (2)

条件	位置	SPC にストアされるアドレス
RC>=2	instr3	不正な命令のプログラムアドレス
	instr4	RptStart-4
	instr5	RptStart-2
RC=1	任意	不正な命令のプログラムアドレス

- (6) 繰り返しプログラム (ループ) が3命令以下の場合には、PC相対命令 (MOVA (disp、PC)、R0など) は繰り返しプログラム (ループ) の最初の命令 (instr1) だけに使えます。
- (7) 繰り返しプログラム (ループ) が4命令以上の場合には、PC相対命令 (MOVA (disp、PC)、R0など) は繰り返しプログラム (ループ) の最後の2命令には使えません。
- (8) SH3-DSPに繰り返し有効フラグはありませんが、RCカウンタが0のとき繰り返しは無効になります。RCカウンタが0でなく、PCカウンタがREレジスタの内容と一致したとき、繰り返しが開始されます。RCカウンタを0に設定すると、繰り返しプログラム (ループ) は無効ですが繰り返しモジュールを1回だけ実行し、RCが1の場合と同様に繰り返しプログラム (ループ) の開始命令には戻りません。RCカウンタを1に設定すると繰り返しモジュールを1回だけ実行し、繰り返しプログラム (ループ) の開始命令には戻りませんが、RCカウンタはゼロになります。
- (9) 繰り返しプログラム (ループ) が4命令以上の場合には、サブルーチン呼び出しと戻り命令を含む分岐命令は、分岐先アドレスとしての"inst3"から"inst5"までの命令には使えません。もしこれを実行すると繰り返し制御は正しく動作しません。繰り返しプログラム (ループ) が3命令以下のとき、分岐先が"RptStart"またはそれより先のアドレスの場合には繰り返し制御は正しく動作せず、SRレジスタ内のRCの内容は更新されません。



- (10) 繰り返し実行中は、割り込みは制限されます。詳細は、図5.18を参照してください。この図のそれぞれのケースのフローがEXの各ステージを示しています。割り込みの最初のEXステージは、通常、命令のEXステージが終了した直後に開始します。これらを図では"A"で示しています。図中の"B"は割り込みが受け付けられない場所を示しています。

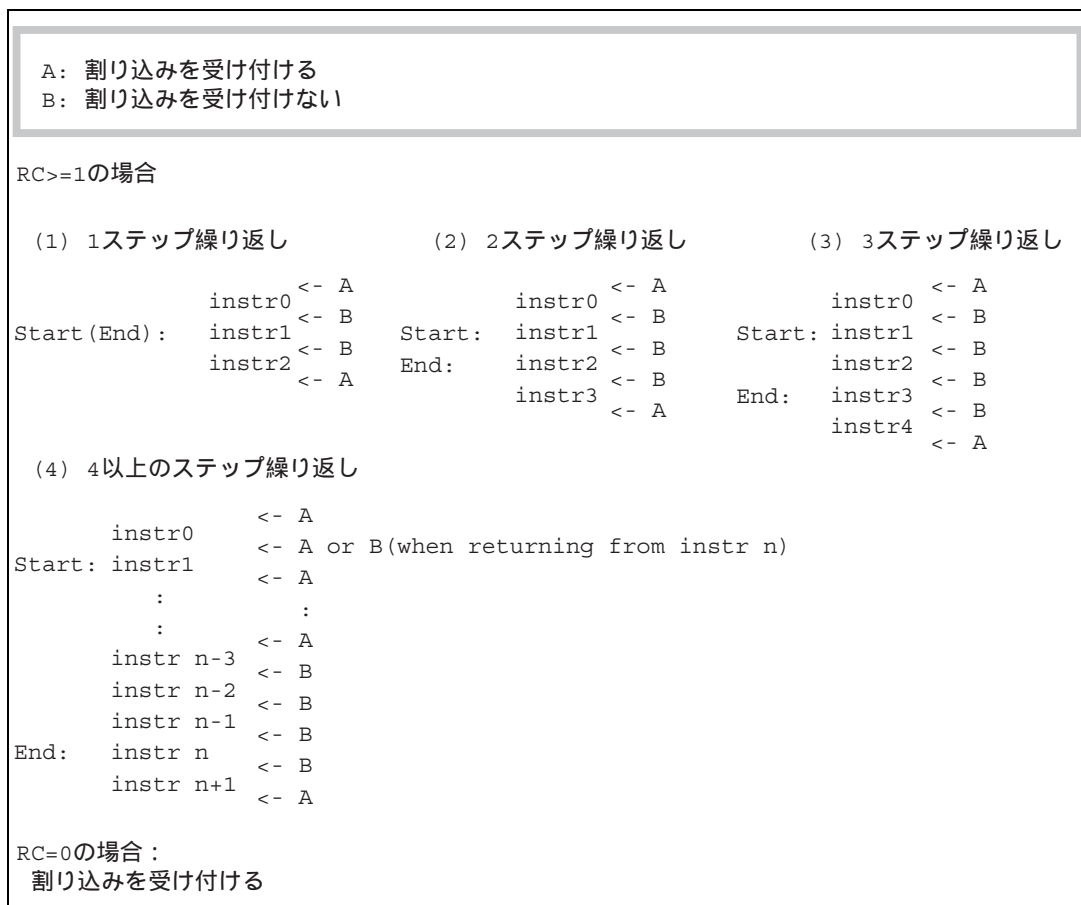


図 5.18 繰り返しモジュールでの割り込み受け付けの制限

## 5.12.1 注意事項

### (1) プログラミングの実際

繰り返し開始レジスタ (RS) と繰り返し終了レジスタ (RE) は、繰り返し開始アドレスと繰り返し終了アドレスをそれぞれ格納しています。これらのレジスタに格納されているアドレスは繰り返しプログラム (ループ) 内の命令の数によって変わります。この規則を次に示します。

Repeat\_Start: 繰り返し開始命令のアドレス

Repeat\_Start0: 繰り返し開始命令の1つ上の命令のアドレス

Repeat\_Start3: 繰り返し終了命令の3つ上の命令のアドレス

5. DSP の演算機能とデータ転送 (SH3-DSP のみ)

表 5.23 RS および RE 設定規則

	繰り返しプログラム (ループ) 内の命令の数			
	1	2	3	>=4
RS	Repeat_start0 + 8	Repeat_start0 + 6	Repeat_start0 + 4	Repeat_Start
RE	Repeat_start0 + 4	Repeat_start0 + 4	Repeat_start0 + 4	Repeat_End3 + 4

このテーブルに基づいて、さまざまなケースを想定した実際の繰り返しプログラム (ループ) のプログラミング例を次に示します。

ケース 1: 1 繰り返し命令の場合

```

LDRS          RptStart0+8;
LDRE          RptStart0+4;
SETRC         RptCount;
- - - -
RptStart0:instr0;
RptStart:instr1;          繰り返し命令
instr2;

```

ケース 2: 2 繰り返し命令の場合

```

LDRS          RptStart0+6;
LDRE          RptStart0+4;
SETRC         RptCount;
- - - -
RptStart0:instr0;
RptStart:instr1;          繰り返し命令 1
RptEnd:instr2;           繰り返し命令 2
instr3;

```

ケース 3: 3 繰り返し命令の場合

```

LDRS          RptStart0+4;
LDRE          RptStart0+4;
SETRC         RptCount;
- - - -
RptStart0:instr0;
RptStart:instr1;          繰り返し命令 1
instr2;                   繰り返し命令 2
RptEnd: instr3;           繰り返し命令 3
instr4;

```

ケース 4: 4 繰り返し命令以上の場合

```

LDRS          RptStart;

```

```

LDRE          RptEnd3+4;
SETRC         RptCount;
- - - -
RptStart0:instr0;
RptStart:instr1;          繰り返し命令 1
        instr2;          繰り返し命令 2
        instr3;          繰り返し命令 3
-----
RptEnd3:instrN-3;        繰り返し命令 N-3
        instrN-2;        繰り返し命令 N-2
        instrN-1;        繰り返し命令 N-1
RptEnd:instrN;          繰り返し命令 N
                        instrN+1

```

上記の例はこの繰り返しプログラム(ループ)シーケンスをプログラミングするためのテンプレートとして用いることができます。拡張命令"REPEAT"で、これらの複雑なラベリングとオフセットの問題を簡素化できます。詳細を次に示します。

## (2) 拡張命令 REPEAT

拡張命令 REPEAT で、表 5.23 に示したラベリングとオフセットの取り扱いを簡単にできます。次に使用するラベルを示します。

RptStart : 繰り返しプログラム(ループ)の先頭命令のアドレス

RptEnd : 繰り返しプログラム(ループ)の最終命令のアドレス

RptCount : 繰り返し回数イミディエイト番号

この命令は次のように使用します。

Repeat count は、イミディエイト値#Imm またはレジスタ間接値 Rn として指定できます。

ケース 1: 1 繰り返し命令の場合

```

REPEAT RptStart, RptStart, RptCount
- - - -
        instr0;
RptStart:instr1;          繰り返し命令 1
        instr2;

```

ケース 2: 2 繰り返し命令の場合

```

REPEAT RptStart, RptStart, RptCount
- - - -
        instr0;
RptStart:instr1;          繰り返し命令 1
RptEnd:instr2;           繰り返し命令 2

```

## 5. DSP の演算機能とデータ転送 (SH3-DSP のみ)

---

ケース 3: 3 繰り返し命令の場合

```
REPEAT RptStart, RptEnd, RptCount
    - - - -
        instr0;
RptStart:instr1;          繰り返し命令 1
        instr2;          繰り返し命令 2
RptEnd:   instr3;        繰り返し命令 3
```

ケース 4: 4 繰り返し命令以上の場合

```
REPEAT RptStart, RptEnd, RptCount
    - - - -
        instr0;
RptStart:instr1;          繰り返し命令 1
        instr2;          繰り返し命令 2
        instr3;          繰り返し命令 3
-----
        instrN-3;        繰り返し命令 N-3
        instrN-2;        繰り返し命令 N-2
        instrN-1;        繰り返し命令 N-1
RptEnd:instrN;           繰り返し命令 N
        instrN+1
```

それぞれのケースでの拡張の結果は、(1) のケース番号に対応します。

### 5.13 条件付き命令とデータ転送

データ演算命令には無条件命令と、条件付き命令があります。両者とも並行して実行するデータ転送命令を指定することができますが、条件が不成立の場合でもデータ転送命令には影響せず、常に実行されます。

条件付き命令とデータ転送の例を図 5.19 に示します。

```
DCT PADD X0, Y0, A0 MOVX.W @R4+, X0 MOVS.W A0, @R6+R9 ;
```

[ 条件が真のとき ]

実行前 : X0=H'33333333, Y0=H'55555555, A0=H'123456789A,  
R4=H'00008000, R6=H'00008232, R1=H'00000004  
(R4)=H'1111, (R6)=H'2222

実行後 : X0=H'11110000, Y0=H'55555555, A0=H'0088888888,  
R4=H'00008002, R6=H'00008236, R1=H'00000004  
(R4)=H'1111, (R6)=H'1234

[ 条件が偽のとき ]

実行前 : X0=H'33333333, Y0=H'55555555, A0=H'123456789A,  
R4=H'00008000, R6=H'00008232, R1=H'00000004  
(R4)=H'1111, (R6)=H'2222

実行後 : X0=H'11110000, Y0=H'55555555, A0=H'123456789A,  
R4=H'00008002, R6=H'00008236, R1=H'00000004  
(R4)=H'1111, (R6)=H'1234

図 5.19 条件付き命令とデータ転送の例



---

## 6. 命令の特長

---

### 6.1 RISC タイプ命令セット

命令は RISC タイプです。特長は次のとおりです。

#### 6.1.1 16 ビット固定長命令

SH-3CPU の命令長はすべて 16 ビット固定長です。これによりプログラムのコード効率が向上します。

SH3-DSP は SH-3 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。DSP の詳細については、「第 5 章 DSP の演算機能とデータ転送」を参照してください。

#### 6.1.2 1 命令 / 1 ステート

パイプライン方式を採用し、基本命令は、1 命令を 1 ステートで実行できます。

#### 6.1.3 データサイズ

演算の基本的なデータサイズはロングワードです。メモリのアクセスサイズは、バイト / ワード / ロングワードを選択できます。メモリのバイトとワードのデータは符号拡張後、ロングワードで演算されます (表 6.1)。リテラルデータは算術演算では符号拡張後、論理演算ではゼロ拡張後、ロングワードで演算されます。

表 6.1 ワードデータの符号拡張

SH-3、SH-3E、SH3-DSP の CPU	説明	他の CPU の例
MOV.W @ (disp, PC), R1 ADD R1, R0 ..... .DATA.W H'1234	32 ビットに符号拡張され、R1 は H'00001234 になります。 次に ADD 命令で演算されます。	ADD.W #H'1234, R0

【注】 @ (disp, PC) でイミディエイトデータを参照します。

#### 6.1.4 ロードストアアーキテクチャ

SH-3、SH-3E、SH3-DSP はロードストアアーキテクチャを採用し、基本演算はレジスタ間で実行します。メモリとの演算は、レジスタにデータをロードし実行します。ただし、AND などのビットを操作する命令は直接メモリに対して実行します。

### 6.1.5 遅延分岐

無条件分岐は、遅延分岐となっています。遅延分岐命令の場合、遅延分岐命令の直後の命令を実行してから、分岐します。これにより、分岐のために起こるパイプラインの乱れを最小限におさえます（表 6.2）。

表 6.2 遅延分岐命令

SH-3、SH-3E、SH3-DSP シリーズの CPU	説明	他の CPU の例
BRA TRGET ADD R1, R0	TRGET に分岐する前に ADD を実行します。	ADD.W R1, R0 BRA TRGET

【注】 遅延分岐の分岐動作そのものは、スロット命令実行後に発生します。しかし、レジスタの更新などの分岐動作を除く命令の実行は、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ているレジスタの内容を変更しても、分岐先アドレスは更新前のレジスタ内容のままです。

### 6.1.6 乗算 / 積和演算

16 ビット×16 ビット→32 ビットの乗算を 1~3 ステート（SH3-DSP では 1、2 ステート）、32 ビット×32 ビット→64 ビットの乗算を 2~5 ステート（SH3-DSP では 2、3 ステート）で実行します。32 ビット×32 ビット+64 ビット→64 ビットの積和演算は、MAC 命令では 2~5 ステート（SH3-DSP では 2~4 ステート）、FMAC 命令\*では 1 ステートで実行します。

【注】\* FMAC 命令は、SH-3E でのみ使用可能です（浮動小数点演算命令）。

### 6.1.7 T ビット

比較結果はステータスレジスタ（SR）の T ビットに反映し、その真、偽によって条件分岐します（表 6.3）。必要最小限の命令によってのみ T ビットを変化させ、処理速度を向上させています。

表 6.3 T ビット

SH-3、SH-3E、SH3-DSP の CPU	説明	他の CPU の例
CMP/GE R1, R0 BT TRGET0 BF TRGET1	R0 R1 のとき T ビットがセットされます。 R0 R1 のとき TRGET0 へ R0<R1 のとき TRGET1 へ分岐します。	CMP.W R1, R0 BGE TRGET0 BLT TRGET1
ADD #-1, R0 CMP/EQ #0, R0 BT TRGET	ADD では T ビットが変化しません。 R0=0 のとき T ビットがセットされます。 R0=0 のとき分岐します。	SUB.W #1, R0 BEQ TRGET



### 6.1.8 イミディエイトデータ

バイトのイミディエイトデータは、直接命令コード中に挿入されます。16ビット固定長の命令コードを保持するため、ワードとロングワードのイミディエイトデータは直接命令コード中に挿入されず、メインメモリ上のテーブルに配置します。メモリ上のテーブルはディスプレイメント付き PC 相対アドレッシングモードを使ったイミディエイトデータのデータ転送命令 (MOV) で参照します (表 6.4)。

表 6.4 イミディエイトデータによる参照

区分	SH-3、SH-3E、SH3-DSP の CPU	他の CPU の例
8 ビットイミディエイト	MOV #H'12,R0	MOV.B #H'12,R0
16 ビットイミディエイト	MOV.W @(disp,PC),R0 ..... .DATA.W H'1234	MOV.W #H'1234,R0
32 ビットイミディエイト	MOV.L @(disp,PC),R0 ..... .DATA.L H'12345678	MOV.L #H'12345678,R0

【注】 @(disp,PC)でイミディエイトデータを参照します。

### 6.1.9 絶対アドレス

絶対アドレスでデータを参照するときは、あらかじめ絶対アドレスの値を、メモリ上のテーブルに配置しておきます。絶対アドレスの値はレジスタに転送され、ロードされた絶対アドレスをもとに、インデックス付きレジスタ間接アドレッシングモードでオペランドはアクセスされます。

表 6.5 絶対アドレスによる参照

区分	SH-3、SH-3E、SH3-DSP の CPU	他の CPU の例
絶対アドレス	MOV.L @(disp,PC),R1 MOV.B @R1,R0 ..... .DATA.L H'12345678	MOV.B @H'12345678,R0

### 6.1.10 16 ビット / 32 ビットディスプレイメント

16ビットまたは32ビットディスプレイメントでデータを参照するときは、あらかじめディスプレイメントの値をメモリ上のテーブルに配置しておきます。命令実行時にディスプレイメントデータをロードする方法で、この値はレジスタに転送され、インデックス付きレジスタ間接アドレッシングモードでオペランドはアクセスされます。

表 6.6 16 ビット / 32 ビットディスプレイメント

区分	SH-3、SH-3E、SH3-DSP の CPU	他の CPU の例
16 ビット ディスプレイメント	MOV.W @(disp,PC),R0 MOV.W @(R0,R1),R2 ..... .DATA.W H'1234	MOV.W @(H'1234,R1),R2

### 6.1.11 特権命令

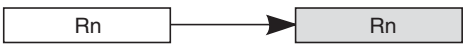
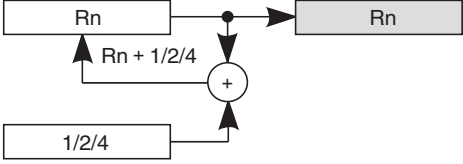
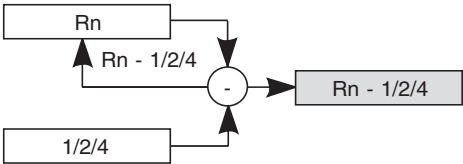
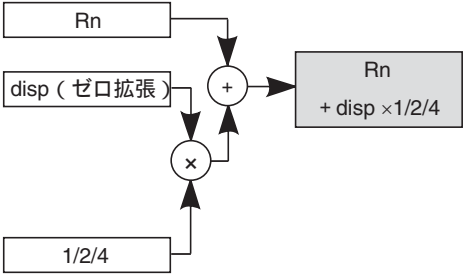
プロセッサはユーザモードと特権モードの2つのオペレーションモードをもっています。次に挙げる特権命令がユーザモードで使用された場合、不当命令例外が検出されます。

- LDC
- STC
- RTE
- LDTLB
- SLEEP

## 6.2 CPU 命令のアドレッシングモード

アドレッシングモードと実効アドレスの計算方法は次のとおりです（表 6.7）。

表 6.7 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
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 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 (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 ワード：Rn+disp × 2 ロングワード： Rn+disp × 4

## 6. 命令の特長

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)	<p>実効アドレスはレジスタ Rn に R0 を加算した内容です。</p>	Rn+R0
ディスプレイースメント付き GBR 間接	@(disp:8,GBR)	<p>実効アドレスはレジスタ GBR に 8 ビットディスプレイースメント 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。</p>	バイト : GBR+disp ワード : GBR+disp × 2 ロングワード : GBR+disp × 4
インデックス付き GBR 間接	@(R0,GBR)	<p>実効アドレスはレジスタ GBR に R0 を加算した内容です。</p>	GBR+R0
ディスプレイースメント付き PC 間接	@(disp:8,PC)	<p>実効アドレスはレジスタ PC に 8 ビットディスプレイースメント 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。</p> <p style="text-align: right;">* ロングワードのとき</p>	ワード : PC+disp × 2 ロングワード : PC&H'FFFFFFFC +disp × 4

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	<p>実効アドレスはレジスタ PC に 8 ビットディスプレイメント disp を符号拡張後 2 倍し、加算した内容です。</p>	PC+disp × 2
	disp:12	<p>実効アドレスはレジスタ PC に 12 ビットディスプレイメント disp を符号拡張後 2 倍し、加算した内容です。</p>	PC+disp × 2
PC 相対 ( 続き )	Rn	<p>実効アドレスはレジスタ PC に R0 を加算した内容です。</p>	PC+R0
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。	

 実効アドレス

### 6.3 DSP データアドレッシング (SH3-DSP のみ)

DSP 命令では 2 つの異なる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 (MOV.S.W、MOV.S.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。データ転送命令の概要を表 6.8 に示します。

表 6.8 データ転送命令の概要

	X、Y データ転送処理 (MOVX.W、MOVY.W)	シングルデータ転送処理 (MOV.S.W、MOV.S.L)
アドレスレジスタ	Ax : R4、R5、Ay : R6、R7	As : R2、R3、R4、R5
インデックスレジスタ	Ix : R8、Iy : R9	Is : R8
アドレッシング	Nop/Inc(+2)/インデックス加算： ポスト更新	Nop/Inc(+2,+4)/インデックス加算： ポスト更新  Dec(-2,-4)：プレ更新
モジュロアドレッシング	可能	不可
データバス	XDB、YDB	LDB
データ長	16bit (ワード)	16bit/32bit (ワード/ロングワード)
バス競合	なし	あり
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	Dx、Dy : A0、A1	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	Ds : A0/A1、M0/M1、X0/X1、Y0/Y1、A0G、A1G

#### 6.3.1 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トがあります。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) 更新なしアドレスレジスタ：  
Ax、Ay レジスタがアドレスポイントです。更新されません。
- (2) 加算インデックスレジスタ：  
Ax、Ay レジスタがアドレスポイントです。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)。
- (3) インクリメントアドレスレジスタ：  
Ax、Ay レジスタがアドレスポイントです。データ転送後それぞれ +2 が加算されます (ポスト更新)。

それぞれのアドレスポイントにはインデッ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) のインデックスレジスタ (Ix) となり、R9 レジスタは Y メモリアドレスレジスタ (Ay) のインデックスレジスタ (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 をインデックスレジスタに設定し加算インデッ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 6.1 に示します。

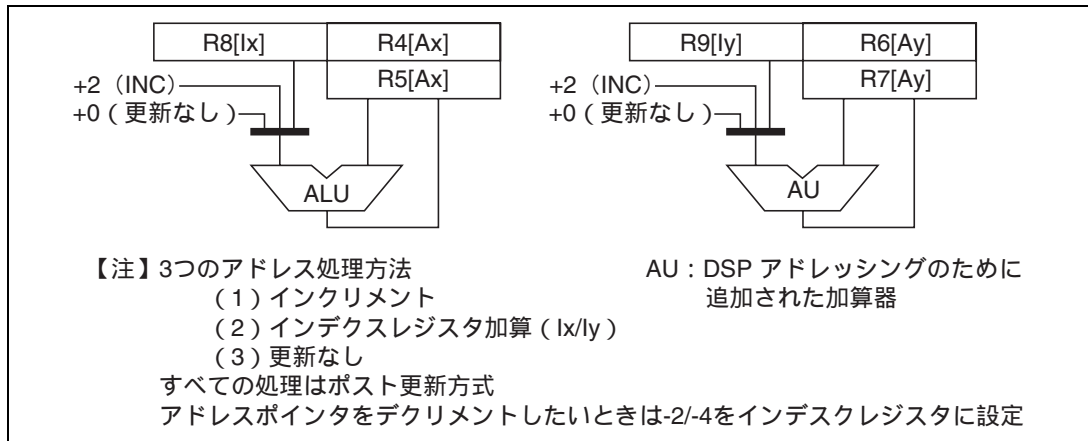


図 6.1 X、Y データ転送のアドレッシング

### 6.3.2 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 ( $A_s$ ) 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) 更新なしアドレスレジスタ：  
 $A_s$ レジスタがアドレスポインタです。更新されません。
- (2) 加算インデクスレジスタ：  
 $A_s$ レジスタがアドレスポインタです。データ転送後  $I_s$ 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レジスタ：  
 $A_s$ 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レジスタ：  
 $A_s$ 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 ( $A_s$ ) は R8 レジスタをインデクスレジスタ ( $I_s$ ) 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 6.2 に示します。

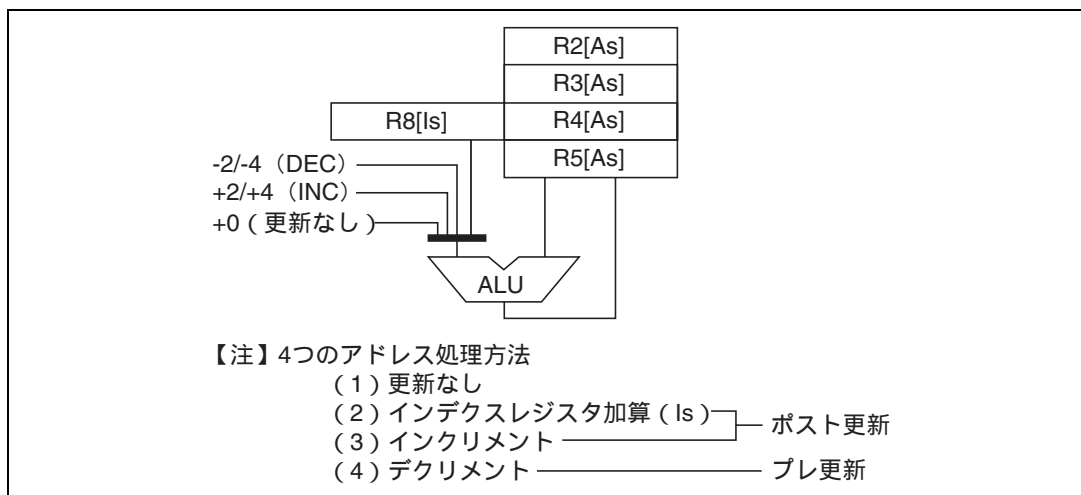


図 6.2 シングルデータ転送のアドレッシング

### 6.3.3 モジュロアドレッシング

SH3-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 または 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



のモジュールサイズは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。モジュールアドレッシングのブロック図を図 6.3 に示します。

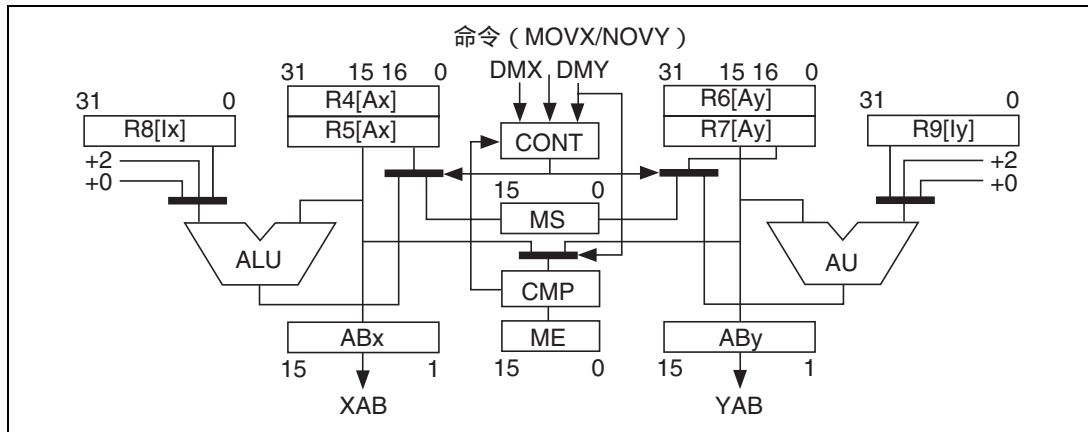


図 6.3 モジュールアドレッシング

モジュール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) に対するモジュール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

(モジュール終了アドレスになったので、モジュール開始アドレスになります)

モジュール開始、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。これはモジュール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ックスを使う場合は、アドレスポインタは ME と一致せず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ール開始アドレスには戻りません。

### 6.3.4 DSP アドレッシング動作

モジュール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 ( Operation is MOVX.W MOVY.W ) {
    ABx=Ax; ABY=Ay;
```

## 6. 命令の特長

---

```
/* memory access cycle uses ABx and ABy. The addresses to be used have not been
updated */

/* Ax is one of R4,5 */
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 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 ( AddrReg[15:0]==ME ) AddrReg[15:0]==MS;
  else AddrReg=AddrReg+Index;
  return AddrReg;
}
```

## 6.4 CPU 命令の命令形式

表 6.9 に、命令形式とソースオペランドとデスティネーションオペランドの意味を示します。命令コードによりオペランドの意味が異なります。記号は次のとおりです。

x x x x : 命令コード  
 m m m m : ソースレジスタ  
 n n n n : デスティネーションレジスタ  
 i i i i : イミディエイトデータ  
 d d d d : ディスプレースメント

表 6.9 命令形式

命令形式		ソースオペランド	デスティネーション オペランド	命令の例
0 形式	15 0 xxxx xxxx xxxx xxxx			NOP
n 形式	15 0 xxxx   nnnn   xxxx xxxx		nnnn: レジスタ直接	MOVT Rn
		コントロールレジスタまたはシステムレジスタ	nnnn: レジスタ直接	STS MACH, Rn
		コントロールレジスタまたはシステムレジスタ	nnnn: プリデクリメントレジスタ間接	STC.L SR, @-Rn
m 形式	15 0 xxxx   mmmm   xxxx xxxx	mmmm: レジスタ直接	コントロールレジスタまたはシステムレジスタ	LDC Rm, SR
		mmmm: ポストインクリメントレジスタ直接	コントロールレジスタまたはシステムレジスタ	LDC.L @Rm+, SR
		mmmm: レジスタ間接		JMP @Rm
		mmmm: Rm を用いた PC 相対		BRAF Rm
nm 形式	15 0 xxxx   nnnn   mmmm   xxxx	mmmm: レジスタ直接	nnnn: レジスタ直接	ADD Rm, Rn
		mmmm: レジスタ直接	nnnn: レジスタ間接	MOV.L Rm, @Rn
		mmmm: ポストインクリメントレジスタ間接 (積和演算) nnnn: * mmmm: ポストインクリメントレジスタ間接 (積和演算)	MACH, MACL	MAC.W @Rm+, @Rn+
		mmmm: ポストインクリメントレジスタ間接	nnnn: レジスタ直接	MOV.L @Rm+, Rn
		mmmm: レジスタ直接	nnnn: プリデクリメントレジスタ間接	MOV.L Rm, @-Rn
		mmmm: レジスタ直接	nnnn: インデックス付きレジスタ間接	MOV.L Rm, @(R0, Rn)

## 6. 命令の特長

命令形式		ソースオペランド	デスティネーション オペランド	命令の例
md 形式	15 xxxx xxxx mmmm dddd	0 mmmmdddd : ディスプレイメント付き レジスタ間接	R0 (レジスタ直接)	MOV.B @(disp,Rm),R0
nd4 形式	15 xxxx xxxx nnnn dddd	0 R0 (レジスタ直接)	nnnndddd : ディスプレイメント付き レジスタ間接	MOV.B R0,@(disp,Rn)
nmd 形式	15 xxxx nnnn mmmm dddd	0 mmmm : レジスタ直接	nnnndddd : ディスプレイメント付き レジスタ間接	MOV.L Rm,@(disp,Rn)
		mmmmdddd : ディスプレイメント付き レジスタ間接	nnnn : レジスタ直接	MOV.L @(disp,Rm),Rn
d 形式	15 xxxx xxxx dddd dddd	0 dddddddd : ディスプレイメント付き GBR 間接	R0 (レジスタ直接)	MOV.L @(disp,GBR),R0
		R0 (レジスタ直接)	dddddddd : ディスプレイメント付き GBR 間接	MOV.L R0,@(disp,GBR)
		dddddddd : ディスプレイメント付き PC 相対	R0 (レジスタ直接)	MOVA @(disp,PC),R0
		dddddddd : PC 相対		BF label
d12 形式	15 xxxx dddd dddd dddd	0 dddddddddddd : PC 相対		BRA label (label=disp+PC)
nd8 形式	15 xxxx nnnn dddd dddd	0 dddddddd : ディスプレイメント付き PC 相対	nnnn : レジスタ直接	MOV.L @(disp,PC),Rn
i 形式	15 xxxx xxxx iiii iiii	iiiiiii : イミディエイト	インデックス付き GBR 間接	AND.B #imm,@(R0,GBR)
		iiiiiii : イミディエイト	R0 (レジスタ直接)	AND #imm,R0
		iiiiiii : イミディエイト		TRAPA #imm
ni 形式	15 xxxx nnnn iiii iiii	0 iiiiiii : イミディエイト	nnnn : レジスタ直接	ADD #imm,Rn

【注】 \* 積和命令では nnnn は、ソースレジスタです。

## 6.5 DSP 命令の命令形式 (SH3-DSP のみ)

SH3-DSP にはデジタル信号処理のための新しい命令が追加されています。新しい命令は次の 2 つに分けられます。

- (1) メモリとDSPレジスタのダブル、シングルデータ転送命令 (16ビット長)
  - (2) DSPエンジンで処理される並行処理命令 (32ビット長)
- それぞれの命令形式を図 6.4 に示します。

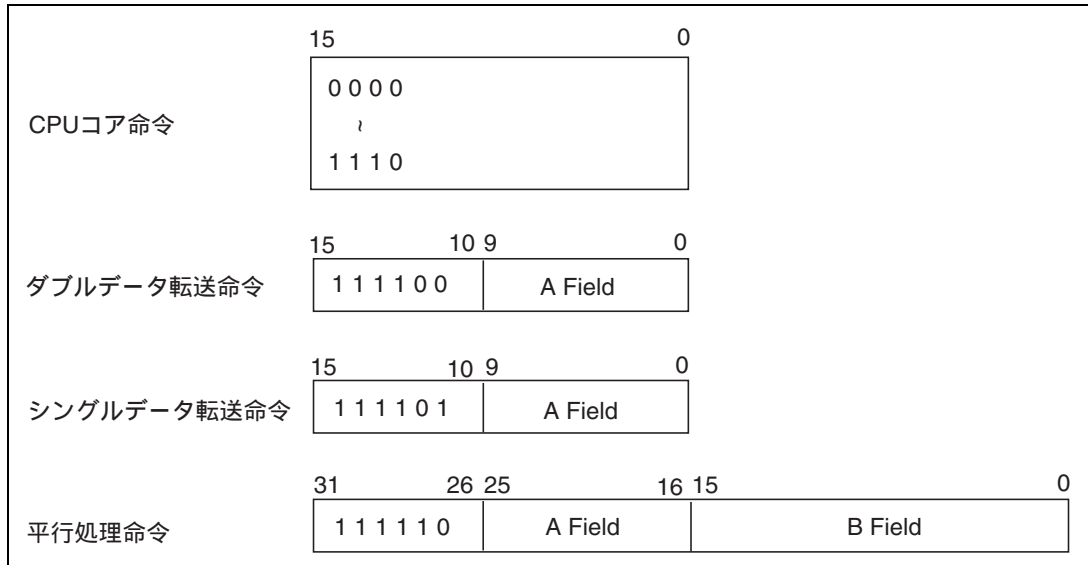


図 6.4 DSP 命令の命令形式

## 6. 命令の特長

### 6.5.1 ダブル、シングルデータ転送命令

ダブルデータ転送命令の命令形式を表 6.10 に、シングルデータ転送命令の命令形式を表 6.11 に示します。

表 6.10 ダブルデータ転送の命令形式

分類	ニーモニック	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
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+lx,Dx													1	1		
	MOVX.W Da,@Ax									Da		1		0	1		
	MOVX.W Da,@Ax+													1	0		
MOVX.W Da,@Ax+lx													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+ly,Dy															1	1
	MOVY.W Da,@Ay										Da		1			0	1
	MOVY.W Da,@Ay+															1	0
MOVY.W Da,@Ay+ly															1	1	

【記号説明】

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 6.11 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:(*1)		0	0	0	0
	MOVS.W @As,Ds							0:R4				1:(*1)		0	1		
	MOVS.W @As+,Ds							1:R5				2:(*1)		1	0		
	MOVS.W @As+ls,Ds							2:R2				3:(*1)		1	1		
	MOVS.W Ds,@-As							3:R3				4:(*1)		0	0		1
	MOVS.W Ds,@As											5:A1		0	1		
	MOVS.W Ds,@As+											6:(*1)		1	0		
	MOVS.W Ds,@As+ls											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+ls,Ds											B:Y1		1	1		
	MOVS.L Ds,@-As											C:M0		0	0		1
	MOVS.L Ds,@As											D:A1G		0	1		
MOVS.L Ds,@As+											E:M1		1	0			
MOVS.L Ds,@As+ls											F:A0G		1	1			

【注】 \*1 システム予約コード

## 6.5.2 並行処理命令

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 6.12 に、B フィールドの ALU 演算命令、乗算命令を表 6.13 に示します。A フィールドの命令は、表 6.10 のダブルデータ転送と同じです。

表 6.12 A フィールドの並行データ転送命令

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	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	1	0	0	0	0	0	0	0	0	0																			
	MOVX.W @Ax, Dx							Ax		Dx		0																						
	MOVX.W @Ax+, Dx												0																					
	MOVX.W @Ax+lx, Dx																																	
	MOVX.W Da, @Ax										Da		1																					
	MOVX.W Da, @Ax+																																	
	MOVX.W Da, @Ax+lx																																	
Y メモリ データ 転送	NOPY							0	0	0	0	0	0	0	0																			
	MOVY.W @Ay, Dy							Ay		Dy		0																						
	MOVY.W @Ay+, Dy																																	
	MOVY.W @Ay+ly, Dy																																	
	MOVY.W Da, @Ay										Da		1																					
	MOVY.W Da, @Ay+																																	
	MOVY.W Da, @Ay+ly																																	

【記号説明】

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

## 6. 命令の特長

表 6.13 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	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																										
Imm. シフト	PSHL #Imm, Dz PSHA #Imm, Dz	1	1	1	1	1	0	Aフィールド										0	0	0	0	-16<=Imm<=+16	Dz	0	0	0	1	0	-32<=Imm<=+32																														
	予約																	0	0	0	1																																						
6オペランド パラレル 命令	PMULS Sx, Sf, Dg 予約 PSUB Sx, Sy, Du PMULS Sx, Sf, Dg PADD Sx, Sy, Du PMULS Sx, Sf, Dg																	0	1	0	0	Se	Sf	Sx	Sy	Dg	Du	0	1	0	1	0:X0	0:Y0	0:X0	0:Y0	0:M0	0:X0	1:X1	1:Y1	1:X1	1:Y1	1:M1	1:Y0	2:Y0	2:X0	2:A0	2:M0	2:A0	2:A0	3:A1	3:A1	3:A1	3:M1	3:A1	3:A1				
3オペランド 命令	予約 PSUBC Sx, Sy, Dz PADDC Sx, Sy, Dz PCMP Sx, Sy 予約 PABS Sx, Dz PRND Sx, Dz PABS Sy, Dz PRND Sy, Dz 予約	1	0	0	0											0	0	0	0																					Dz	0:(*1)	1:(*1)	2:(*1)	3:(*1)	4:(*1)	5:A1	6:(*1)	7:A0	8:X0	9:X1	A:Y0	B:Y1	C:M0	D:(*1)	E:M1	F:(*1)			
条件付き 3オペランド 命令	[if cc] PSHL Sx, Sy, Dz [if cc] PSHA Sx, Sy, Dz [if cc] PSUB Sx, Sy, Dz [if cc] PADD Sx, Sy, Dz 予約 [if cc] PAND Sx, Sy, Dz [if cc] PXOR Sx, Sy, Dz [if cc] POR Sx, Sy, Dz [if cc] PDEC Sx, Dz [if cc] PINC Sx, Dz [if cc] PDEC Sy, Dz [if cc] PINC Sy, Dz [if cc] PCLR Dz [if cc] PDMSB Sx, Dz 予約 [if cc] PDMSB Sy, Dz [if cc] PNEG Sx, Dz [if cc] PCOPY Sx, Dz [if cc] PNEG Sy, Dz [if cc] PCOPY Sy, Dz 予約 [if cc] PSTS MACH, Dz [if cc] PSTS MACL, Dz [if cc] PLDS Dz, MACH [if cc] PLDS Dz, MACL (*2) 予約															0	0	0	0	fcc																																							
	予約															0	0	1	1																																								
	予約	1	1	1	1	1	1											0	0	1	0																																						

【注】 \*1 システム予約コード

\*2 [if cc]: DCT (DCビット真)、DCF (DCビット偽) またはなし (無条件命令)



## 7. 命令セット

### 7.1 分類順命令セット

SH-3 は 68、また SH-3E は 84 の基本命令をもち、表 7.1 に示すような 7 つの分類に分けることができます。表 7.3 ~ 表 7.10 は各命令の動作説明、命令コード、実行ステートと機能の一覧表です。

表 7.1 命令の分類

分類	命令の種類	オペコード	機能	命令数
データ転送命令	5	MOV	データ転送 イミディエイトデータ転送 周辺モジュールデータ転送 構造データ転送	39
		MOVA	実効アドレスの転送	
		MOVT	T ビットの転送	
		SWAP	上位と下位の交換	
		XTRCT	連結レジスタの中央切り出し	
算術演算命令	21	ADD	2 進加算	33
		ADDC	キャリ付き 2 進加算	
		ADDV	オーバフロー付き 2 進加算	
		CMP/cond	比較	
		DIV1	除算	
		DIV0S	符号付き除算の初期化	
		DIV0U	符号なし除算の初期化	
		DMULS	符号付き倍精度乗算	
		DMULU	符号なし倍精度乗算	
		DT	デクリメントとテスト	
		EXTS	符号拡張	
		EXTU	ゼロ拡張	
		MAC	積和演算、倍精度積和演算	
		MUL	倍精度乗算 (32 × 32 ビット)	
		MULS	符号付き乗算 (16 × 16 ビット)	
		MULU	符号なし乗算 (16 × 16 ビット)	
		NEG	符号反転	
		NEGC	ポロー付き符号反転	
		SUB	2 進減算	
		SUBC	キャリ付き 2 進減算	
SUBV	アンダフロー付き 2 進減算			
論理演算命令	6	AND	論理積演算	14
		NOT	ビット反転	

## 7. 命令セット

分類	命令の種類	オペコード	機能	命令数
論理演算命令	6	OR	論理和演算	14
		TAS	メモリテストとビットセット	
		TST	論理積演算とTビットセット	
		XOR	排他的論理和演算	
シフト命令	12	ROTL	1ビット左回転	16
		ROTR	1ビット右回転	
		ROTCL	Tビット付き1ビット左回転	
		ROTCR	Tビット付き1ビット右回転	
		SHAL	1ビット左算術シフト	
		SHAR	1ビット右算術シフト	
		SHLL	1ビット左論理シフト	
		SHLLn	nビット左論理シフト	
		SHLR	1ビット右論理シフト	
		SHLRn	nビット右論理シフト	
		SHAD	ダイナミック算術シフト	
		SHLD	ダイナミック論理シフト	
分岐命令	9	BF	条件分岐、遅延付き条件分岐 (T=0 で分岐)	11
		BT	条件分岐、遅延付き条件分岐 (T=1 で分岐)	
		BRA	無条件分岐	
		BRAF	無条件分岐	
		BSR	サブルーチンプロシージャへの分岐	
		BSRF	サブルーチンプロシージャへの分岐	
		JMP	無条件分岐	
		JSR	サブルーチンプロシージャへの分岐	
		RTS	サブルーチンプロシージャからの復帰	
システム制御命令	15	CLRT	Tビットのクリア	83 (75)*
		CLRMAC	MACレジスタのクリア	
		CLRS	Sビットのクリア	
		LDC*	コントロールレジスタへのロード	
		LDS	システムレジスタへのロード	
		LDTLB	TLBにPTEをロード	
		NOP	無操作	
		PREF	データキャッシュへのプリフェッチ	
		RTE	例外処理からの復帰	
		SETS	Sビットのセット	
		SETT	Tビットのセット	
		SLEEP	低消費電力モードへの遷移	
		STC	コントロールレジスタからのストア	
		STS*	システムレジスタからのストア	
TRAPA	トラップ例外処理			

分類	命令の種類	オペコード	機能	命令数
浮動小数点命令 (SH-3Eのみ)	16	FABS	浮動小数点数絶対値	23
		FADD	浮動小数点数加算	
		FCMP	浮動小数点数比較	
		FDIV	浮動小数点数除算	
		FLDI0	浮動小数点数ロード イミディエイト0	
		FLDI1	浮動小数点数ロード イミディエイト1	
		FLDS	システムレジスタ FPUL への浮動小数点数ロード	
		FLOAT	整数から浮動小数点数への変換	
		FMAC	浮動小数点数積和演算	
		FMOV	浮動小数点数転送	
		FMUL	浮動小数点数乗算	
		FNEG	浮動小数点数符号反転	
		FSQRT	浮動小数点数平方根	
		FSTS	システムレジスタ FPUL からの浮動小数点数ストア	
		FSUB	浮動小数点数減算	
FTRC	浮動小数点数の整数への切り捨て変換			
	計 84			計 219 (188)*

【注】 \* LDS、STS 命令には、FPU のシステムレジスタへのロード/ストア命令が含まれています。これらの命令は SH-3E でのみ使用可能です。( ) 内の命令数は、SH-3E の命令を除いた値です。

## 7. 命令セット

表 7.2 に分類順に、命令コード、動作、および実行ステート数を示します。

表 7.3 ~ 表 7.10 では、実行に必要な最小の実行ステート数を示しています。実際には、命令フェッチがデータアクセスと競合した場合やロード命令（メモリ→レジスタ）のデスティネーションレジスタが次命令によって使用される場合には、命令実行ステート数は増加します。

表 7.2 命令リストの表記

命令	オペレーション	命令コード	特権	命令実行ステート	T ビット
ニーモニックで表示していません。  【記号説明】 OP.Sz SRC,DEST OP: オペコード Sz: サイズ SRC: ソース DEST: デスティネーション  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	動作の概略を表示していません。  【記号説明】 →,←: 転送方向 (xx): メモリオペランド M/Q/T: SR 内のフラグビット  &: ビットごとの論理積  : ビットごとの論理和 ^: ビットごとの排他的論理和 ~: ビットごとの論理否定 <<n: 左 n ビットシフト >>n: 右 n ビットシフト	MSB←→LSB の順で表示しています。  【記号説明】 mmmm: ソースレジスタ nnnn: デスティネーションレジスタ  0000: R0, FR0 0001: R1, FR1 ..... 1111: R15, FR15 iii: イミディエイトデータ dddd: ディスプレースメント*2	特権命令を示しています。	表に示した実行ステートは最少値です。*1	命令実行後の、T ビットの値を表示しています。  【記号説明】 : 変化しない

- 【注】 \*1 命令の実行ステートについて表に示した実行ステートは最少値です。実際は、
- (1) 命令フェッチとデータアクセスの競合が起こる場合
  - (2) ロード命令（メモリ→レジスタ）のデスティネーションレジスタと、その直後の命令が使うレジスタが同一な場合などの条件により、命令実行ステート数は増加します。
- \*2 命令のオペランドサイズなどに応じてスケーリング（×1、×2、×4）されます。詳細は「第 8 章 各命令の説明」を参照してください。

## 7.1.1 データ転送命令

表 7.3 データ転送命令

命令	動作	命令コード	特権	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)→符号拡張→Rn	1001nnnndddddddd		1	
MOV.L @(disp,PC),Rn	(disp×4+PC)→Rn	1101nnnndddddddd		1	
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 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)	0001nnnnmmmmddd		1	
MOV.B @(disp,Rm),R0	(disp+Rm)→符号拡張→R0	10000100mmmmddd		1	
MOV.W @(disp,Rm),R0	(disp×2+Rm)→符号拡張→R0	10000101mmmmddd		1	
MOV.L @(disp,Rm),Rn	(disp×4+Rm)→Rn	0101nnnnmmmmddd		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	
MOV.B R0,@(disp,GBR)	R0→(disp+GBR)	11000000ddddddd		1	
MOV.W R0,@(disp,GBR)	R0→(disp×2+GBR)	11000001ddddddd		1	
MOV.L R0,@(disp,GBR)	R0→(disp×4+GBR)	11000010ddddddd		1	
MOV.B @(disp,GBR),R0	(disp+GBR)→符号拡張→R0	11000100ddddddd		1	
MOV.W @(disp,GBR),R0	(disp×2+GBR)→符号拡張→R0	11000101ddddddd		1	
MOV.L @(disp,GBR),R0	(disp×4+GBR)→R0	11000110ddddddd		1	
MOVA @(disp,PC),R0	disp×4+PC→R0	11000111ddddddd		1	
MOVT Rn	T→Rn	0000nnnn00101001		1	
PREF @Rn	(Rn)→キャッシュ	0000nnnn10000011		1/2*	

## 7. 命令セット

命令	動作	命令コード	特権	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→REG	0110nnnnmmmm1000		1	
SWAP.W Rm,Rn	Rm→上下ワード交換→Rn	0110nnnnmmmm1001		1	
XTRCT Rm,Rn	Rm:Rn の中央 32 ビット→Rn	0010nnnnmmmm1101		1	

【注】 \* SH3-DSP では 2 ステートになります。

### 7.1.2 算術演算命令

表 7.4 算術演算命令

命令	動作	命令コード	特権	実行 ステート	Tビット
ADD Rm,Rn	Rn+Rm→Rn	0011nnnnmmmm1100		1	
ADD #imm,Rn	Rn+imm→Rn	0111nnnniiiiiii		1	
ADDC Rm,Rn	Rn+Rm+T→Rn, キャリ→T	0011nnnnmmmm1110		1	キャリ
ADDV Rm,Rn	Rn+Rm→Rn, オーバフロー→T	0011nnnnmmmm1111		1	オーバ フロー
CMP/EQ #imm,R0	R0=imm のとき 1→T	10001000iiiiiii		1	比較結果
CMP/EQ Rm,Rn	Rn=Rm のとき 1→T	0011nnnnmmmm0000		1	比較結果
CMP/HS Rm,Rn	無符号で Rn Rm のとき 1→T	0011nnnnmmmm0010		1	比較結果
CMP/GE Rm,Rn	有符号で Rn Rm のとき 1→T	0011nnnnmmmm0011		1	比較結果
CMP/HI Rm,Rn	無符号で Rn>Rm のとき 1→T	0011nnnnmmmm0110		1	比較結果
CMP/GT Rm,Rn	有符号で Rn>Rm のとき 1→T	0011nnnnmmmm0111		1	比較結果
CMP/PZ Rn	Rn 0 のとき 1→T	0100nnnn00010001		1	比較結果
CMP/PL Rn	Rn>0 のとき 1→T	0100nnnn00010101		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	0010nnnnmmmm1100		1	比較結果
DIV1 Rm,Rn	1 ステップ除算 (Rn ÷ Rm)	0011nnnnmmmm0100		1	計算結果
DIV0S Rm,Rn	Rn の MSB→Q, Rm の MSB→M, M^Q→T	0010nnnnmmmm0111		1	計算結果
DIV0U	0→M/Q/T	000000000011001		1	0
DMULS.L Rm,Rn	符号付きで Rn × Rm→MACH,MACL32 × 32→64 ビット	0011nnnnmmmm1101		2 (~ 5/4)* <sup>1</sup>	
DMULU.L Rm,Rn	符号なしで Rn × Rm→MACH,MACL 32 × 32→64 ビット	0011nnnnmmmm0101		2 (~ 5/4)* <sup>1</sup>	
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
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	
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	
MAC.L @Rm+,@Rn+	符号付きで (Rn) × (Rm)+MAC→MAC	0000nnnnmmmm1111		2 (~ 5/4)* <sup>1</sup>	

## 7. 命令セット

命令		動作	命令コード	特権	実行 ステート	Tビット
MAC.W	@Rm+, @Rn+	符号付きで $(Rn) \times (Rm) + MAC \rightarrow MAC$ 16 × 16 + 64 → 64 ビット	0100nnnnmmmm1111		2 (~ 5) <sup>*1</sup>	
MUL.L	Rm, Rn	$Rn \times Rm \rightarrow MACL$ 32 × 32 → 32 ビット	0000nnnnmmmm0111		2 (~ 5/4) <sup>*1</sup>	
MULS.W	Rm, Rn	符号付きで $Rn \times Rm \rightarrow MAC$ 16 × 16 → 32 ビット	0010nnnnmmmm1111		1 (~ 3) <sup>*2</sup>	
MULU.W	Rm, Rn	符号なしで $Rn \times Rm \rightarrow MAC$ 16 × 16 → 32 ビット	0010nnnnmmmm1110		1 (~ 3) <sup>*2</sup>	
NEG	Rm, Rn	0-Rm → Rn	0110nnnnmmmm1011		1	
NEGC	Rm, Rn	0-Rm-T → Rn, ボロー → T	0110nnnnmmmm1010		1	ボロー
SUB	Rm, Rn	Rn-Rm → Rn	0011nnnnmmmm1000		1	
SUBC	Rm, Rn	Rn-Rm-T → Rn, ボロー → T	0011nnnnmmmm1010		1	ボロー
SUBV	Rm, Rn	Rn-Rm → Rn, アンダフロー → T	0011nnnnmmmm1011		1	アンダ フロー

【注】 \*1 実行ステートの通常最小値は2ですが、その命令の実行直後に MAC レジスタから演算結果を読みこむときには5ステート (SH3-DSP では4ステート) が必要です。

\*2 実行ステートの通常最小値は1ですが、MUL 命令の実行直後に MAC レジスタから演算結果を読みこむときには3ステートが必要です。

## 7. 命令セット

### 7.1.3 論理演算命令

表 7.5 論理演算命令

命令	動作	命令コード	特権	実行 ステート	Tビット
AND Rm,Rn	$Rn \& Rm \rightarrow Rn$	0010nnnnmmmm1001		1	
AND #imm,R0	$R0 \& imm \rightarrow R0$	11001001iiiiiii		1	
AND.B #imm,@(R0,GBR)	$(R0+GBR) \& imm \rightarrow (R0+GBR)$	11001101iiiiiii		3	
NOT Rm,Rn	$\sim Rm \rightarrow Rn$	0110nnnnmmmm0111		1	
OR Rm,Rn	$Rn   Rm \rightarrow Rn$	0010nnnnmmmm1011		1	
OR #imm,R0	$R0   imm \rightarrow R0$	11001011iiiiiii		1	
OR.B #imm,@(R0,GBR)	$(R0+GBR)   imm \rightarrow (R0+GBR)$	11001111iiiiiii		3	
TAS.B @Rn	(Rn)が0のとき 1→T, 1→MSB of (Rn)	0100nnnn00011011		3/4*	テスト 結果
TST Rm,Rn	$Rn \& Rm$ , 結果が0のとき 1→T	0010nnnnmmmm1000		1	テスト 結果
TST #imm,R0	$R0 \& imm$ , 結果が0のとき 1→T	11001000iiiiiii		1	テスト 結果
TST.B #imm,@(R0,GBR)	$(R0+GBR) \& imm$ , 結果が0のとき 1→T	11001100iiiiiii		3	テスト 結果
XOR Rm,Rn	$Rn \wedge Rm \rightarrow Rn$	0010nnnnmmmm1010		1	
XOR #imm,R0	$R0 \wedge imm \rightarrow R0$	11001010iiiiiii		1	
XOR.B #imm,@(R0,GBR)	$(R0+GBR) \wedge imm \rightarrow (R0+GBR)$	11001110iiiiiii		3	

【注】 \* SH3-DSP では4ステートになります。



## 7.1.4 シフト命令

表 7.6 シフト命令

命令	動作	命令コード	特権	実行 ステート	Tビット
ROTL Rn	$T \leftarrow Rn \leftarrow MSB$	0100nnnn00000100		1	MSB
ROTR Rn	$LSB \rightarrow Rn \rightarrow T$	0100nnnn00000101		1	LSB
ROTCL Rn	$T \leftarrow Rn \leftarrow T$	0100nnnn00100100		1	MSB
ROTCR Rn	$T \rightarrow Rn \rightarrow T$	0100nnnn00100101		1	LSB
SHAD Rm, Rn	Rn 0 のとき、 $Rn \ll Rm \rightarrow Rn$ Rn<0 のとき、 $Rn \gg Rm \rightarrow [MSB \rightarrow Rn]$	0100nnnnmmmm1100		1	
SHAL Rn	$T \leftarrow Rn \leftarrow 0$	0100nnnn00100000		1	MSB
SHAR Rn	$MSB \rightarrow Rn \rightarrow T$	0100nnnn00100001		1	LSB
SHLD Rm, Rn	Rn 0 のとき、 $Rn \ll Rm \rightarrow Rn$ Rn<0 のとき、 $Rn \gg Rm \rightarrow [0 \rightarrow Rn]$	0100nnnnmmmm1101		1	
SHLL Rn	$T \leftarrow Rn \leftarrow 0$	0100nnnn00000000		1	MSB
SHLR Rn	$0 \rightarrow Rn \rightarrow T$	0100nnnn00000001		1	LSB
SHLL2 Rn	$Rn \ll 2 \rightarrow Rn$	0100nnnn00001000		1	
SHLR2 Rn	$Rn \gg 2 \rightarrow Rn$	0100nnnn00001001		1	
SHLL8 Rn	$Rn \ll 8 \rightarrow Rn$	0100nnnn00011000		1	
SHLR8 Rn	$Rn \gg 8 \rightarrow Rn$	0100nnnn00011001		1	
SHLL16 Rn	$Rn \ll 16 \rightarrow Rn$	0100nnnn00101000		1	
SHLR16 Rn	$Rn \gg 16 \rightarrow Rn$	0100nnnn00101001		1	

## 7.1.5 分岐命令

表 7.7 分岐命令

命令	動作	命令コード	特権	実行 ステート	Tビット
BF label	T=0 のとき $disp \times 2 + PC \rightarrow PC$ , T=1 のとき nop	10001011dddddddd		3/1*	
BF/S label	遅延分岐、T=0 のとき $disp \times 2 + PC \rightarrow PC$ , T=1 のとき nop	10001111dddddddd		2/1*	
BT label	T=1 のとき $disp \times 2 + PC \rightarrow PC$ , T=0 のとき nop	10001001dddddddd		3/1*	
BT/S label	T=1 のとき $disp \times 2 + PC \rightarrow PC$ , T=0 のとき nop	10001101dddddddd		2/1*	
BRA label	遅延分岐、 $disp \times 2 + PC \rightarrow PC$	1010dddddddddddd		2	
BRAF Rm	$Rn + PC \rightarrow PC$	0000mmmm00100011		2	
BSR label	遅延分岐、 $PC \rightarrow PR$ , $disp \times 2 + PC \rightarrow PC$	1011dddddddddddd		2	
BSRF Rm	$PC \rightarrow PR$ , $Rn + PC \rightarrow PC$	0000mmmm00000011		2	
JMP @Rm	遅延分岐、 $Rn \rightarrow PC$	0100mmmm00101011		2	
JSR @Rm	遅延分岐、 $PC \rightarrow PR$ , $Rn \rightarrow PC$	0100mmmm00001011		2	
RTS	遅延分岐、 $PR \rightarrow PC$	000000000001011		2	

【注】 \* 分岐しないときは 1 ステートになります。

## 7. 命令セット

### 7.1.6 システム制御命令

表 7.8 システム制御命令

命令	動作	命令コード	特権	実行 状態	Tビット
CLRMACH	0→MACH,MACL	000000000101000		1	
CLRS	0→S	000000001001000		1	
CLRT	0→T	000000000001000		1	0
LDC Rm,SR	Rm→SR	0100mmmm00001110	特権	5	LSB
LDC Rm,GBR	Rm→GBR	0100mmmm00011110		1/3* <sup>1</sup>	
LDC Rm,VBR	Rm→VBR	0100mmmm00101110	特権	1/3* <sup>1</sup>	
LDC Rm,SSR	Rm→SSR	0100mmmm00111110	特権	1/3* <sup>1</sup>	
LDC Rm,SPC	Rm→SPC	0100mmmm01001110	特権	1/3* <sup>1</sup>	
LDC Rm,R0_BANK	Rm→R0_BANK	0100mmmm10001110	特権	1/3* <sup>1</sup>	
LDC Rm,R1_BANK	Rm→R1_BANK	0100mmmm10011110	特権	1/3* <sup>1</sup>	
LDC Rm,R2_BANK	Rm→R2_BANK	0100mmmm10101110	特権	1/3* <sup>1</sup>	
LDC Rm,R3_BANK	Rm→R3_BANK	0100mmmm10111110	特権	1/3* <sup>1</sup>	
LDC Rm,R4_BANK	Rm→R4_BANK	0100mmmm11001110	特権	1/3* <sup>1</sup>	
LDC Rm,R5_BANK	Rm→R5_BANK	0100mmmm11011110	特権	1/3* <sup>1</sup>	
LDC Rm,R6_BANK	Rm→R6_BANK	0100mmmm11101110	特権	1/3* <sup>1</sup>	
LDC Rm,R7_BANK	Rm→R7_BANK	0100mmmm11111110	特権	1/3* <sup>1</sup>	
LDC.L @Rm+,SR	(Rm)→SR, Rm+4→Rm	0100mmmm00000111	特権	7	LSB
LDC.L @Rm+,GBR	(Rm)→GBR, Rm+4→Rm	0100mmmm00010111		1/5* <sup>2</sup>	
LDC.L @Rm+,VBR	(Rm)→VBR, Rm+4→Rm	0100mmmm00100111	特権	1/5* <sup>2</sup>	
LDC.L @Rm+,SSR	(Rm)→SSR, Rm+4→Rm	0100mmmm00110111	特権	1/5* <sup>2</sup>	
LDC.L @Rm+,SPC	(Rm)→SPC, Rm+4→Rm	0100mmmm01000111	特権	1/5* <sup>2</sup>	
LDC.L @Rm+,R0_BANK	(Rm)→R0_BANK, Rm+4→Rm	0100mmmm10000111	特権	1/5* <sup>2</sup>	
LDC.L @Rm+,R1_BANK	(Rm)→R1_BANK, Rm+4→Rm	0100mmmm10010111	特権	1/5* <sup>2</sup>	
LDC.L @Rm+,R2_BANK	(Rm)→R2_BANK, Rm+4→Rm	0100mmmm10100111	特権	1/5* <sup>2</sup>	
LDC.L @Rm+,R3_BANK	(Rm)→R3_BANK, Rm+4→Rm	0100mmmm10110111	特権	1/5* <sup>2</sup>	
LDC.L @Rm+,R4_BANK	(Rm)→R4_BANK, Rm+4→Rm	0100mmmm11000111	特権	1/5* <sup>2</sup>	
LDC.L @Rm+,R5_BANK	(Rm)→R5_BANK, Rm+4→Rm	0100mmmm11010111	特権	1/5* <sup>2</sup>	
LDC.L @Rm+,R6_BANK	(Rm)→R6_BANK, Rm+4→Rm	0100mmmm11100111	特権	1/5* <sup>2</sup>	
LDC.L @Rm+,R7_BANK	(Rm)→R7_BANK, Rm+4→Rm	0100mmmm11110111	特権	1/5* <sup>2</sup>	
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	
LDTLB	PTEH/PTEL→TLB	000000000111000	特権	1	
NOP	無操作	000000000001001		1	

## 7. 命令セット

命令	動作	命令コード	特権	実行 ステート	Tビット
PREF @Rm	(Rn)→cache	0000mmmm10000011		1	
RTE	遅延分岐、SSR/SPC→SR/PC	0000000000101011	特権	4	
SETS	1→S	0000000001011000		1	
SETT	1→T	0000000000011000		1	1
SLEEP	スリープ	0000000000011011	特権	4* <sup>3</sup>	
STC SR,Rn	SR→Rn	0000nnnn00000010	特権	1	
STC GBR,Rn	GBR→Rn	0000nnnn00010010		1	
STC VBR,Rn	VBR→Rn	0000nnnn00100010	特権	1	
STC SSR, Rn	SSR→Rn	0000nnnn00110010	特権	1	
STC SPC,Rn	SPC→Rn	0000nnnn01000010	特権	1	
STC R0_BANK,Rn	R0_BANK→Rn	0000nnnn10000010	特権	1	
STC R1_BANK,Rn	R1_BANK→Rn	0000nnnn10010010	特権	1	
STC R2_BANK,Rn	R2_BANK→Rn	0000nnnn10100010	特権	1	
STC R3_BANK,Rn	R3_BANK→Rn	0000nnnn10110010	特権	1	
STC R4_BANK,Rn	R4_BANK→Rn	0000nnnn11000010	特権	1	
STC R5_BANK,Rn	R5_BANK→Rn	0000nnnn11010010	特権	1	
STC R6_BANK,Rn	R6_BANK→Rn	0000nnnn11100010	特権	1	
STC R7_BANK,Rn	R7_BANK→Rn	0000nnnn11110010	特権	1	
STC.L SR,@-Rn	Rn-4→Rn, SR→(Rn)	0100nnnn00000011	特権	1/2* <sup>4</sup>	
STC.L GBR,@-Rn	Rn-4→Rn, GBR→(Rn)	0100nnnn00010011		1/2* <sup>4</sup>	
STC.L VBR,@-Rn	Rn-4→Rn, VBR→(Rn)	0100nnnn00100011	特権	1/2* <sup>4</sup>	
STC.L SSR,@-Rn	Rn-4→Rn, SSR→(Rn)	0100nnnn00110011	特権	1/2* <sup>4</sup>	
STC.L SPC,@-Rn	Rn-4→Rn, SPC→(Rn)	0100nnnn01000011	特権	1/2* <sup>4</sup>	
STC.L R0_BANK,@-Rn	Rn-4→Rn, R0_BANK→(Rn)	0100nnnn10000011	特権	2	
STC.L R1_BANK,@-Rn	Rn-4→Rn, R1_BANK→(Rn)	0100nnnn10010011	特権	2	
STC.L R2_BANK,@-Rn	Rn-4→Rn, R2_BANK→(Rn)	0100nnnn10100011	特権	2	
STC.L R3_BANK,@-Rn	Rn-4→Rn, R3_BANK→(Rn)	0100nnnn10110011	特権	2	
STC.L R4_BANK,@-Rn	Rn-4→Rn, R4_BANK→(Rn)	0100nnnn11000011	特権	2	
STC.L R5_BANK,@-Rn	Rn-4→Rn, R5_BANK→(Rn)	0100nnnn11010011	特権	2	
STC.L R6_BANK,@-Rn	Rn-4→Rn, R6_BANK→(Rn)	0100nnnn11100011	特権	2	
STC.L R7_BANK,@-Rn	Rn-4→Rn, R7_BANK→(Rn)	0100nnnn11110011	特権	2	
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	
TRAPA #imm	PC/SR→SPC/SSR, #imm <<2→TRA, 0x160→EXPEVT, VBR+ H'0100→PC	1100001111111111		6/8* <sup>5</sup>	

【注】 命令の実行ステートについて表に示した実行ステートは最小値です。実際は、

- (1) 命令フェッチとデータアクセスの競合が起こる場合
- (2) ロード命令(メモリ→レジスタ)のデスティネーションレジスタと、その直後の命令が使うレジスタが同一な場合

## 7. 命令セット

などの条件により、命令実行ステート数は増加します。

- \*1 SH3-DSP では 3 ステートになります。
- \*2 SH3-DSP では 5 ステートになります。
- \*3 スリープ状態に遷移するまでのステート数です。
- \*4 SH3-DSP では 2 ステートになります。
- \*5 SH3-DSP では 8 ステートになります。

### 7.1.7 浮動小数点命令 (SH-3E のみ)

表 7.9 浮動小数点命令

命令	動作	命令コード	特権	実行ステート	Tビット
FABS FRn	FRn →FRn	1111nnnn01011101		1	
FADD FRm,FRn	FRn + FRm → FRn	1111nnnnmmmm0000		1	
FCMP/EQ FRm,FRn	(FRn == FRm)? 1:0 → T	1111nnnnmmmm0100		1	比較結果
FCMP/GT FRm,FRn	(FRn > FRm)? 1:0 → T	1111nnnnmmmm0101		1	比較結果
FDIV FRm,FRn	FRn /FRm → FRn	1111nnnnmmmm0011		13	
FLDI0 FRn	H'00000000 → FRn	1111nnnn10001101		1	
FLDI1 FRn	H'3F800000 → FRn	1111nnnn10011101		1	
FLDS FRm,FPUL	FRm → FPUL	1111nnnn00011101		1	
FLOAT FPUL,FRn	(float)FPUL → FRn	1111nnnn00101101		1	
FMAC FR0,FRm,FRn	FR0 × FRm + FRn → FRn	1111nnnnmmmm1110		1	
FMOV FRm,FRn	FRm → FRn	1111nnnnmmmm1100		1	
FMOV.S @(R0,Rm),FRn	(R0 + Rm) → FRn	1111nnnnmmmm0110		1	
FMOV.S @Rm+,FRn	(Rm) → FRn, Rm+4 → Rm	1111nnnnmmmm1001		1	
FMOV.S @Rm,FRn	(Rm) → FRn	1111nnnnmmmm1000		1	
FMOV.S FRm,@(R0,Rn)	(FRm) → (R0+Rn)	1111nnnnmmmm0111		1	
FMOV.S FRm,@-Rn	Rn-4 → Rn, FRm→(Rn)	1111nnnnmmmm1011		1	
FMOV.S FRm,@Rn	FRm→(Rn)	1111nnnnmmmm1010		1	
FMUL FRm,FRn	FRn × FRm → FRn	1111nnnnmmmm0010		1	
FNEG FRn	-FRn → FRn	1111nnnn01001101		1	
FSQRT FRn	FRn → FRn	1111nnnn01101101		13	
FSTS FPUL,FRn	FPUL → FRn	1111nnnn00001101		1	
FSUB FRm,FRn	FRn - FRm → FRn	1111nnnnmmmm0001		1	
FTRC FRm,FPUL	(long)FRm → FPUL	1111nnnn00111101		1	

## 7.1.8 FPU システムレジスタに関連する CPU 命令 (SH-3E のみ)

表 7.10 FPU に関連する CPU 命令

命令	動作	命令コード	特権	実行 ステート	Tビット
LDS Rm,FPSCR	Rm→FPSCR	0100nnnn01101010		1	
LDS Rm,FPUL	Rm→FPUL	0100nnnn01011010		1	
LDS.L @Rm+,FPSCR	@Rm → FPSCR ,Rm+4→Rm	0100nnnn01100110		1	
LDS.L @Rm+,FPUL	@Rm → FPUL , Rm+4→Rm	0100nnnn01010110		1	
STS FPSCR, Rn	FPSCR→Rn	0000nnnn01101010		1	
STS FPUL, Rn	FPUL→Rn	0000nnnn01011010		1	
STS.L FPSCR,@- Rn	Rn-4→Rn, FPSCR→@Rn	0100nnnn01100010		1	
STS.L FPUL,@-Rn	Rn-4→Rn, FPUL→@Rn	0100nnnn01010010		1	

## 7.1.9 DSP 機能をサポートする CPU 命令 (SH3-DSP のみ)

DSP 機能をサポートするために CPU コア命令にいくつかのシステム制御命令が追加されました。繰り返し制御、モジュロアドレッシングをサポートする RS、RE、MOD レジスタが追加され、RC カウンタが SR レジスタに追加されました。これらにアクセスするため、LDC、STC 命令が追加されました。DSP レジスタの DSR、A0、X0、X1、Y0 および Y1 レジスタにアクセスするために、LDS、STS 命令が追加されました。

SR レジスタの繰り返しカウンタ (RC、ビット 27~16) に値を設定する SETRC 命令が追加されました。SETRC 命令のオペランドがイミディエイトのときは、8 ビットのイミディエイトデータが SR レジスタのビット 23~16 に格納され、ビット 27~24 は 0 にクリアされます。オペランドがレジスタのときは、レジスタのビット 11~0 の 12 ビットが SR レジスタのビット 27~16 に格納されます。

繰り返し開始アドレス、繰り返し終了アドレスを RS、RE レジスタに設定する命令は、LDC 命令のほかに、LDRS、LDRE 命令を追加しました。

追加された命令を表 7.11 に示します。

## 7. 命令セット

表 7.11 追加された CPU 命令

命令	動作	命令コード	実行 ステート	T ビット
LDC Rm,MOD	Rm→MOD	0100mmmm01011110	3	
LDC Rm,RE	Rm→RE	0100mmmm01111110	3	
LDC Rm,RS	Rm→RS	0100mmmm01101110	3	
LDC.L @Rm+,MOD	(Rm)→MOD、Rm + 4→Rm	0100mmmm01010111	5	
LDC.L @Rm+,RE	(Rm)→RE、Rm + 4→Rm	0100mmmm01110111	5	
LDC.L @Rm+,RS	(Rm)→RS、Rm + 4→Rm	0100mmmm01100111	5	
STC MOD,Rn	MOD→Rn	0000nnnn01010010	1	
STC RE,Rn	RE→Rn	0000nnnn01110010	1	
STC RS,Rn	RS→Rn	0000nnnn01100010	1	
STC.L MOD,@-Rn	Rn - 4→Rn、MOD→(Rn)	0100nnnn01010011	2	
STC.L RE,@-Rn	Rn - 4→Rn、RE→(Rn)	0100nnnn01110011	2	
STC.L RS,@-Rn	Rn - 4→Rn、RS→(Rn)	0100nnnn01100011	2	
LDS Rm,DSR	Rm→DSR	0100mmmm01101010	1	
LDS.L @Rm+,DSR	(Rm)→DSR、Rm + 4→Rm	0100mmmm01100110	1	
LDS Rm,A0	Rm→A0	0100mmmm01110110	1	
LDS.L @Rm+,A0	(Rm)→A0、Rm + 4→Rm	0100mmmm01100110	1	
LDS Rm,X0	Rm→X0	0100mmmm01110110	1	
LDS.L @Rm+,X0	(Rm)→X0、Rm + 4→Rm	0100mmmm01100110	1	
LDS Rm,X1	Rm→X1	0100mmmm01110110	1	
LDS.L @Rm+,X1	(Rm)→X1、Rm + 4→Rm	0100mmmm01100110	1	
LDS Rm,Y0	Rm→Y0	0100mmmm01110110	1	
LDS.L @Rm+,Y0	(Rm)→Y0、Rm + 4→Rm	0100mmmm01100110	1	
LDS Rm,Y1	Rm→Y1、Rm + 4→Rm	0100mmmm01110110	1	
LDS.L @Rm+,Y1	(Rm)→Y1、Rm + 4→Rm	0100mmmm01100110	1	
STS DSR,Rn	DSR→Rn	0000nnnn01101010	1	
STS.L DSR,@-Rn	Rn - 4→Rn、DSR→(Rn)	0100nnnn01100010	1	
STS A0,Rn	A0→Rn	0000nnnn01111010	1	
STS.L A0,@-Rn	Rn - 4→Rn、A0→(Rn)	0100nnnn01110010	1	
STS X0,Rn	X0→Rn	0000nnnn01111010	1	
STS.L X0,@-Rn	Rn - 4→Rn、X0→(Rn)	0100nnnn01110010	1	
STS X1,Rn	X1→Rn	0000nnnn01111010	1	
STS.L X1,@-Rn	Rn - 4→Rn、X1→(Rn)	0100nnnn01110010	1	
STS Y0,Rn	Y0→Rn	0000nnnn10101010	1	
STS.L Y0,@-Rn	Rn - 4→Rn、Y0→(Rn)	0100nnnn10100010	1	
STS Y1,Rn	Y1→Rn	0000nnnn10111010	1	
STS.L Y1,@-Rn	Rn - 4→Rn、Y1→(Rn)	0100nnnn10110010	1	
SETRC Rm	Rm[11:0]→RC (SR[27:16])	0100mmmm00010100	3	
SETRC #imm	imm→RC(SR[23:16]),zeros→SR[27:24]	10000101iiiiiii	3	
LDRS @(disp,pc)	disp × 2+PC→RS	10001100dddddddd	3	
LDRE @(disp,pc)	disp × 2+PC→RE	10001110dddddddd	3	

## 7.2 アルファベット順命令セット

表 7.12 に命令の命令コードと実行ステートを、アルファベット順に示します。

表 7.12 アルファベット順命令セット

命令	動作	命令コード	特権	実行ステート	T ビット
ADD #imm,Rn	Rn+imm→Rn	0111nnnniiiiiii		1	
ADD Rm,Rn	Rn+Rm→Rn	0011nnnnmmmm1100		1	
ADDC Rm,Rn	Rn+Rm+T→Rn, キャリ→T	0011nnnnmmmm1110		1	キャリ
ADDV Rm,Rn	Rn+Rm→Rn, オーバフロー→T	0011nnnnmmmm1111		1	オーバーフロー
AND #imm,R0	R0 & imm → R0	11001001iiiiiii		1	
AND Rm,Rn	Rn & Rm → Rn	0010nnnnmmmm1001		1	
AND.B #imm,@(R0,GBR)	(R0+GBR) & imm → (R0+GBR)	11001101iiiiiii		3	
BF label	T=0 のとき disp+PC→PC, T=1 のとき nop	10001011ddddddd		3/1*2	
BF/S label	T=0 のとき disp+PC→PC, T=1 のとき nop	10001111ddddddd		2/1*2	
BRA label	遅延分岐、disp+PC→PC	1010ddddddddddd		2	
BRAF Rm	遅延分岐、Rn+PC→PC	0000mmmm00100011		2	
BSR label	遅延分岐、PC→PR, disp+PC→PC	1011ddddddddddd		2	
BSRF Rm	遅延分岐、PC→PR, Rn+PC→PC	0000mmmm00000011		2	
BT label	T=1 のとき disp+PC→PC, T=0 のとき nop	10001001ddddddd		3/1*2	
BT/S label	T=1 のとき disp+PC→PC, T=0 のとき nop	10001101ddddddd		2/1*2	
CLRMACH	0→MACH,MACL	000000000101000		1	
CLRS	0→S	000000001001000		1	
CLRT	0→T	000000000001000		1	0
CMP/EQ #imm,R0	R0=imm のとき 1→T	10001000iiiiiii		1	比較結果
CMP/EQ Rm,Rn	Rn=Rm のとき 1→T	0011nnnnmmmm0000		1	比較結果
CMP/GE Rm,Rn	有符号で Rn Rm のとき 1→T	0011nnnnmmmm0011		1	比較結果
CMP/GT Rm,Rn	有符号で Rn>Rm のとき 1→T	0011nnnnmmmm0111		1	比較結果
CMP/HI Rm,Rn	無符号で Rn>Rm のとき 1→T	0011nnnnmmmm0110		1	比較結果
CMP/HS Rm,Rn	無符号で Rn Rm のとき 1→T	0011nnnnmmmm0010		1	比較結果
CMP/PL Rn	Rn>0 のとき 1→T	0100nnnn00010101		1	比較結果
CMP/PZ Rn	Rn 0 のとき 1→T	0100nnnn00010001		1	比較結果
CMP/STR Rm,Rn	Rn と Rm バイトが等しいとき 1→T	0010nnnnmmmm1100		1	比較結果
DIV0S Rm,Rn	Rn の MSB→Q, Rm の MSB→M, M^Q→T	0010nnnnmmmm0111		1	計算結果
DIV0U	0→M/Q/T	000000000011001		1	0
DIV1 Rm,Rn	1 ステップ除算 (Rn ÷ Rm)	0011nnnnmmmm0100		1	計算結果
DMULS.L Rm,Rn	符号付きで Rn × Rm→MACH,MACL	0011nnnnmmmm1101		2(～5)*1	

## 7. 命令セット

命令	動作	命令コード	特権	実行 ステート	Tビット
DMULU.L Rm,Rn	符号なしで Rn x Rm → MACH, MACL	0011nnnnmmmm0101		2(~5)* <sup>1</sup>	
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
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	
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	
FABS FRn * <sup>3</sup>	FRn  → FRn	1111nnnn01011101		1	
FADD FRm ,FRn * <sup>3</sup>	FRn + FRm → FRn	1111nnnnmmmm0000		1	
FCMP/EQ FRm ,FRn * <sup>3</sup>	(FRn == FRm)? 1:0 → T	1111nnnnmmmm0100		1	比較結果
FCMP/GT FRm ,FRn * <sup>3</sup>	(FRn > FRm) ? 1:0 → T	1111nnnnmmmm0101		1	比較結果
FDIV FRm ,FRn * <sup>3</sup>	FRn /FRm → FRn	1111nnnnmmmm0011		13	
FLDI0 FRn * <sup>3</sup>	H'00000000 → FRn	1111nnnn10001101		1	
FLDI1 FRn * <sup>3</sup>	H'3F800000 → FRn	1111nnnn10011101		1	
FLDS FRm ,FPUL * <sup>3</sup>	FRm → FPUL	1111nnnn00011101		1	
FLOAT FPUL, FRn * <sup>3</sup>	(float)FPUL → FRn	1111nnnn00101101		1	
FMAC FR0 ,FRm ,FRn * <sup>3</sup>	FR0 x FRm + FRn → FRn	1111nnnnmmmm1110		1	
FMOV FRm ,FRn * <sup>3</sup>	FRm → FRn	1111nnnnmmmm1100		1	
FMOV.S @(R0,Rm),FRn	(R0 + Rm) → FRn	1111nnnnmmmm0110		1	
FMOV.S @Rm+,FRn	(Rm) → FRn, Rm+4 =Rm	1111nnnnmmmm1001		1	
FMOV.S @Rm, FRn	(Rm) → FRn	1111nnnnmmmm1000		1	
FMOV.S FRm ,@(R0,Rn)	(FRm)→(R0+Rn)	1111nnnnmmmm0111		1	
FMOV.S FRm ,@-Rn	Rn-4→Rn, FRm→(Rn)	1111nnnnmmmm1011		1	
FMOV.S FRm ,@Rn	FRm→(Rn)	1111nnnnmmmm1010		1	
FMUL FRm ,FRn	FRn x FRm → FRn	1111nnnnmmmm0010		1	
FNEG FRn	-FRn → FRn	1111nnnn01001101		1	
FSQRT FRn	FRn → FRn	1111nnnn01101101		13	
FSTS FPUL, FRn	FPUL → FRn	1111nnnn00001101		1	
FSUB FRm, FRn	FRn - FRm → FRn	1111nnnnmmmm0001		1	
FTRC FRm, FPUL	(long)FRm → FPUL	1111nnnn00111101		1	
JMP @Rm	遅延分岐、Rm→PC	0100mmmm00101011		2	
JSR @Rm	遅延分岐、PC→PR, Rm→PC	0100mmmm00001011		2	
LDC Rm,GBR	Rm→GBR	0100mmmm00011110		1/3* <sup>4</sup>	
LDC Rm,SR	Rm→SR	0100mmmm00001110	特権	5	LSB
LDC Rm,VBR	Rm→VBR	0100mmmm00101110	特権	1/3* <sup>4</sup>	
LDC Rm,SSR	Rm→SSR	0100mmmm00111110	特権	1/3* <sup>4</sup>	
LDC Rm,SPC	Rm→SPC	0100mmmm01001110	特権	1/3* <sup>4</sup>	
LDC Rm,MOD	Rm→MOD	0100mmmm01011110	特権	3	
LDC Rm,RE	Rm→RE	0100mmmm01101110	特権	3	
LDC Rm,RS	Rm→RS	0100mmmm01101110	特権	3	



## 7. 命令セット

命令	動作	命令コード	特権	実行 ステート	Tビット
LDC Rm,R0_BANK	Rm→R0_BANK	0100mmmm10001110	特権	1/3* <sup>4</sup>	
LDC Rm,R1_BANK	Rm→R1_BANK	0100mmmm10011110	特権	1/3* <sup>4</sup>	
LDC Rm,R2_BANK	Rm→R2_BANK	0100mmmm10101110	特権	1/3* <sup>4</sup>	
LDC Rm,R3_BANK	Rm→R3_BANK	0100mmmm10111110	特権	1/3* <sup>4</sup>	
LDC Rm,R4_BANK	Rm→R4_BANK	0100mmmm11001110	特権	1/3* <sup>4</sup>	
LDC Rm,R5_BANK	Rm→R5_BANK	0100mmmm11011110	特権	1/3* <sup>4</sup>	
LDC Rm,R6_BANK	Rm→R6_BANK	0100mmmm11101110	特権	1/3* <sup>4</sup>	
LDC Rm,R7_BANK	Rm→R7_BANK	0100mmmm11111110	特権	1/3* <sup>4</sup>	
LDC.L @Rm+,GBR	(Rm)→GBR, Rm+4→Rm	0100mmmm00010111		1/5* <sup>5</sup>	
LDC.L @Rm+,SR	(Rm)→SR, Rm+4→Rm	0100mmmm00000111	特権	7	LSB
LDC.L @Rm+,VBR	(Rm)→VBR, Rm+4→Rm	0100mmmm00100111	特権	1/5* <sup>5</sup>	
LDC.L @Rm+,SSR	(Rm)→SSR, Rm+4→Rm	0100mmmm00110111	特権	1/5* <sup>5</sup>	
LDC.L @Rm+,SPC	(Rm)→SPC, Rm+4→Rm	0100mmmm01000111	特権	1/5* <sup>5</sup>	
LDC.L @Rm+,MOD	(Rm)→MOD, Rm+4→Rm	0100mmmm01010111	特権	5	
LDC.L @Rm+,RE	(Rm)→RE, Rm+4→Rm	0100mmmm01110111	特権	5	
LDC.L @Rm+,RS	(Rm)→RS, Rm+4→Rm	0100mmmm01100111	特権	5	
LDC.L @Rm+,R0_BANK	(Rm)→R0_BANK, Rm+4→Rm	0100mmmm10000111	特権	1/5* <sup>5</sup>	
LDC.L @Rm+,R1_BANK	(Rm)→R1_BANK, Rm+4→Rm	0100mmmm10010111	特権	1/5* <sup>5</sup>	
LDC.L @Rm+,R2_BANK	(Rm)→R2_BANK, Rm+4→Rm	0100mmmm10100111	特権	1/5* <sup>5</sup>	
LDC.L @Rm+,R3_BANK	(Rm)→R3_BANK, Rm+4→Rm	0100mmmm10110111	特権	1/5* <sup>5</sup>	
LDC.L @Rm+,R4_BANK	(Rm)→R4_BANK, Rm+4→Rm	0100mmmm11000111	特権	1/5* <sup>5</sup>	
LDC.L @Rm+,R5_BANK	(Rm)→R5_BANK, Rm+4→Rm	0100mmmm11010111	特権	1/5* <sup>5</sup>	
LDC.L @Rm+,R6_BANK	(Rm)→R6_BANK, Rm+4→Rm	0100mmmm11100111	特権	1/5* <sup>5</sup>	
LDC.L @Rm+,R7_BANK	(Rm)→R7_BANK, Rm+4→Rm	0100mmmm11110111	特権	1/5* <sup>5</sup>	
LDRE @(disp,pc)	disp x 2+pc→RE	10001110ddddddd		3	
LDRS @(disp,pc)	disp x 2+pc→RS	10001100ddddddd		3	
LDS Rm,FPSCR * <sup>3</sup>	Rm→FPSCR	0100nnnn01101010		1	
LDS Rm,FPUL * <sup>3</sup>	Rm→FPUL	0100nnnn01011010		1	
LDS Rm,MACH	Rm→MACH	0100mmmm00001010		1	
LDS Rm,MACL	Rm→MACL	0100mmmm00011010		1	
LDS Rm,PR	Rm→PR	0100mmmm00101010		1	
LDS Rm,A0	Rm→DSR	0100mmmm01101010		1	
LDS Rm,DSR	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+,FPSCR * <sup>3</sup>	@Rm → FPSCR, Rm+4→Rn	0100nnnn01100110		1	
LDS.L @Rm+,FPUL * <sup>3</sup>	@Rm → FPUL, Rm+4→Rn	0100nnnn01010110		1	
LDS.L @Rm+,MACH	(Rm)→MACH, Rm+4→Rm	0100mmmm00000110		1	

## 7. 命令セット

命令	動作	命令コード	特権	実行 ステート	Tビット
LDS.L @Rm+,MACL	(Rm)→MACL, Rm+4→Rm	0100mmmm00010110		1	
LDS.L @Rm+,PR	(Rm)→PR, Rm+4→Rm	0100mmmm00100110		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	
LDTLB	PTCH/PTCL→TLB	000000000111000	特権	1	
MAC.L @Rm+,@Rn+	符号付きで (Rn) x (Rm)+MAC→MAC	0000nnnnmmmm1111		2(~5)* <sup>1</sup>	
MAC.W @Rm+,@Rn+	符号付きで (Rn) x (Rm)+MAC→MAC	0100nnnnmmmm1111		2(~5)* <sup>1</sup>	
MOV #imm,Rn	#imm→符号拡張→Rn	1110nnnniiiiiii		1	
MOV Rm,Rn	Rm→Rn	0110nnnnmmmm0011		1	
MOV.B @(disp,GBR),R0	(disp+GBR)→符号拡張→R0	11000100ddddddd		1	
MOV.B @(disp,Rm),R0	(disp+Rm)→符号拡張→R0	10000100mmmmddd		1	
MOV.B @(R0,Rm),Rn	(R0+Rm)→符号拡張→Rn	0000nnnnmmmm1100		1	
MOV.B @Rm+,Rn	(Rm)→符号拡張→Rn, Rm+1→Rm	0110nnnnmmmm0100		1	
MOV.B @Rm,Rn	(Rm)→符号拡張→Rn	0110nnnnmmmm0000		1	
MOV.B R0,@(disp,GBR)	R0→(disp+GBR)	11000000ddddddd		1	
MOV.B R0,@(disp,Rn)	R0→(disp+Rn)	10000000nnnnddd		1	
MOV.B Rm,@(R0,Rn)	Rm→(R0+Rn)	0000nnnnmmmm0100		1	
MOV.B Rm,@-Rn	Rn-1→Rn, Rm→(Rn)	0010nnnnmmmm0100		1	
MOV.B Rm,@Rn	Rm→(Rn)	0010nnnnmmmm0000		1	
MOV.L @(disp,GBR),R0	(disp+GBR)→R0	11000110ddddddd		1	
MOV.L @(disp,PC),Rn	(disp+PC)→Rn	1101nnnnddddddd		1	
MOV.L @(disp,Rm),Rn	(disp+Rm)→Rn	0101nnnnmmmmddd		1	
MOV.L @(R0,Rm),Rn	(R0+Rm)→Rn	0000nnnnmmmm1110		1	
MOV.L @Rm+,Rn	(Rm)→Rn, Rm+4→Rm	0110nnnnmmmm0110		1	
MOV.L @Rm,Rn	(Rm)→Rn	0110nnnnmmmm0010		1	
MOV.L R0,@(disp,GBR)	R0→(disp+GBR)	11000101ddddddd		1	
MOV.L Rm,@(disp,Rn)	Rm→(disp+Rn)	0001nnnnmmmmddd		1	
MOV.L Rm,@(R0,Rn)	Rm→(R0+Rn)	0000nnnnmmmm0110		1	
MOV.L Rm,@-Rn	Rn-4→Rn, Rm→(Rn)	0010nnnnmmmm0110		1	
MOV.L Rm,@Rn	Rm→(Rn)	0010nnnnmmmm0010		1	
MOV.W @(disp,GBR),R0	(disp+GBR)→符号拡張→R0	11000101ddddddd		1	
MOV.W @(disp,PC),Rn	(disp+PC)→符号拡張→Rn	1001nnnnddddddd		1	
MOV.W @(disp,Rm),R0	(disp+Rm)→符号拡張→R0	10000101mmmmddd		1	
MOV.W @(R0,Rm),Rn	(R0+Rm)→符号拡張→Rn	0000nnnnmmmm1101		1	
MOV.W @Rm+,Rn	(Rm)→符号拡張→Rn, Rm+2→Rm	0110nnnnmmmm0101		1	

## 7. 命令セット

命令	動作	命令コード	特権	実行 ステート	Tビット
MOV.W @Rm,Rn	(Rm)→符号拡張→Rn	0110nnnnmmmm0001		1	
MOV.W R0,@(disp,GBR)	R0→(disp+GBR)	11000001ddddddd		1	
MOV.W R0,@(disp,Rn)	R0→(disp+Rn)	10000001nnnnddd		1	
MOV.W Rm,@(R0,Rn)	Rm→(R0+Rn)	0000nnnnmmmm0101		1	
MOV.W Rm,@-Rn	Rn-2→Rn, Rm→(Rn)	0010nnnnmmmm0101		1	
MOV.W Rm,@Rn	Rm→(Rn)	0010nnnnmmmm0001		1	
MOVA @(disp,PC),R0	disp+PC→R0	11000111ddddddd		1	
MOVT Rn	T→Rn	0000nnnn00101001		1	
MUL.L Rm,Rn	Rn × Rm→MAC	0000nnnnmmmm0111		2(～5)* <sup>1</sup>	
MULS.W Rm,Rn	符号付きで Rn × Rm→MAC	0010nnnnmmmm1111		1(～3)* <sup>1</sup>	
MULU.W Rm,Rn	符号なしで Rn × Rm→MAC	0010nnnnmmmm1110		1(～3)* <sup>1</sup>	
NEG Rm,Rn	0-Rm→Rn	0110nnnnmmmm1011		1	
NEGC Rm,Rn	0-Rm-T→Rn, ボロ→T	0110nnnnmmmm1010		1	ボロ→
NOP	無操作	000000000001001		1	
NOT Rm,Rn	~Rm → Rn	0110nnnnmmmm0111		1	
OR #imm,R0	R0   imm → R0	11001011iiiiiii		1	
OR Rm,Rn	Rn   Rm → Rn	0010nnnnmmmm1011		1	
OR.B #imm,@(R0,GBR)	(R0+GBR)   imm → (R0+GBR)	11001111iiiiiii		3	
PREF @Rm	(Rm)→キャッシュ	0000mmmm10000011		1/2* <sup>6</sup>	
ROTCL Rn	T←Rn←T	0100nnnn00100100		1	MSB
ROTCR Rn	T→Rn→T	0100nnnn00100101		1	LSB
ROTL Rn	T←Rn←MSB	0100nnnn00000100		1	MSB
ROTR Rn	LSB→Rn→T	0100nnnn00000101		1	LSB
RTE	遅延分岐、SSR/ SPC、SR/PC	000000000101011	特権	4	
RTS	遅延分岐、PR→PC	000000000001011		2	
SETRC Rm	Rm の下位 12 ビット→RC(SR のビット 27～16)、繰り返し制御フラグ→RF1, RF0	0100mmmm00010100		3	
SETRC #imm	imm→RC(SR のビット 23～16)、繰り返し制御フラグ→RF1, RF0	10000010iiiiiii		3	
SETS	1→S	000000001011000		1	
SETT	1→T	000000000011000		1	1
SHAD Rm, Rn	Rn 0 のとき、Rn<<Rm→Rn Rn<0 のとき、Rn>>Rm→[MSB→Rn]	0100nnnnmmmm1100		1	
SHAL Rn	T←Rn←0	0100nnnn00100000		1	MSB
SHAR Rn	MSB→Rn→T	0100nnnn00100001		1	LSB
SHLD Rm, Rn	Rn 0 のとき、Rn<<Rm→Rn Rn<0 のとき、Rn>>Rm→[0→Rn]	0100nnnnmmmm1101		1	
SHLL Rn	T←Rn←0	0100nnnn00000000		1	MSB
SHLL2 Rn	Rn<<2 → Rn	0100nnnn00001000		1	
SHLL8 Rn	Rn<<8 → Rn	0100nnnn00011000		1	

## 7. 命令セット

命令	動作	命令コード	特権	実行 ステート	Tビット
SHLL16 Rn	Rn<<16 → Rn	0100nnnn00101000		1	
SHLR Rn	0→Rn→T	0100nnnn00000001		1	LSB
SHLR2 Rn	Rn>>2 → Rn	0100nnnn00001001		1	
SHLR8 Rn	Rn>>8 → Rn	0100nnnn00011001		1	
SHLR16 Rn	Rn>>16 → Rn	0100nnnn00101001		1	
SLEEP	スリープ	000000000011011	特権	4	
STC GBR,Rn	GBR→Rn	0000nnnn00010010		1	
STC SR,Rn	SR→Rn	0000nnnn00000010	特権	1	
STC VBR,Rn	VBR→Rn	0000nnnn00100010	特権	1	
STC SSR,Rn	SSR→Rn	0000nnnn00110010	特権	1	
STC SPC,Rn	SPC→Rn	0000nnnn01000010	特権	1	
STC MOD,Rn	MOD→Rn	0000nnnn01010010		1	
STC RE,Rn	RE→Rn	0000nnnn01110010		1	
STC RS,Rn	RS→Rn	0000nnnn01100010		1	
STC R0_BANK, Rn	R0_BANK→Rn	0000nnnn10000010	特権	1	
STC R1_BANK, Rn	R1_BANK→Rn	0000nnnn10010010	特権	1	
STC R2_BANK, Rn	R2_BANK→Rn	0000nnnn10100010	特権	1	
STC R3_BANK, Rn	R3_BANK→Rn	0000nnnn10110010	特権	1	
STC R4_BANK, Rn	R4_BANK→Rn	0000nnnn11000010	特権	1	
STC R5_BANK, Rn	R5_BANK→Rn	0000nnnn11010010	特権	1	
STC R6_BANK, Rn	R6_BANK→Rn	0000nnnn11100010	特権	1	
STC R7_BANK, Rn	R7_BANK→Rn	0000nnnn11110010	特権	1	
STC.L GBR,@-Rn	Rn-4→Rn, GBR→(Rn)	0100nnnn00010011		1/2* <sup>6</sup>	
STC.L SR,@-Rn	Rn-4→Rn, SR→(Rn)	0100nnnn00000011	特権	1/2* <sup>6</sup>	
STC.L VBR,@-Rn	Rn-4→Rn, VBR→(Rn)	0100nnnn00100011	特権	1/2* <sup>6</sup>	
STC.L SSR,@-Rn	Rn-4→Rn, SSR→(Rn)	0100nnnn00110011	特権	1/2* <sup>6</sup>	
STC.L SPC,@-Rn	Rn-4→Rn, SPC→(Rn)	0100nnnn01000011	特権	1/2* <sup>6</sup>	
STC.L MOD,@-Rn	Rn-4→Rn, MOD→(Rn)	0100nnnn01010011	特権	2	
STC.L RE,@-Rn	Rn-4→Rn, RE→(Rn)	0100nnnn01110011	特権	2	
STC.L RS,@-Rn	Rn-4→Rn, RS→(Rn)	0100nnnn01100011	特権	2	
STC.L R0_BANK,@-Rn	Rn-4→Rn, R0_BANK→(Rn)	0100nnnn10000011	特権	2	
STC.L R1_BANK,@-Rn	Rn-4→Rn, R1_BANK→(Rn)	0100nnnn10010011	特権	2	
STC.L R2_BANK,@-Rn	Rn-4→Rn, R2_BANK→(Rn)	0100nnnn10100011	特権	2	
STC.L R3_BANK,@-Rn	Rn-4→Rn, R3_BANK→(Rn)	0100nnnn10110011	特権	2	
STC.L R4_BANK,@-Rn	Rn-4→Rn, R4_BANK→(Rn)	0100nnnn11000011	特権	2	
STC.L R5_BANK,@-Rn	Rn-4→Rn, R5_BANK→(Rn)	0100nnnn11010011	特権	2	
STC.L R6_BANK,@-Rn	Rn-4→Rn, R6_BANK→(Rn)	0100nnnn11100011	特権	2	
STC.L R7_BANK,@-Rn	Rn-4→Rn, R7_BANK→(Rn)	0100nnnn11110011	特権	2	
STS FPSCR, Rn * <sup>3</sup>	FPSCR→Rn	0000nnnn01101010		1	
STS FPUL, Rn * <sup>3</sup>	FPUL→Rn	0000nnnn01011010		1	

## 7. 命令セット

命令	動作	命令コード	特権	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 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 FPSCR,@-Rn <sup>*3</sup>	Rn-4→Rn, FPSCR→@Rn	0100nnnn01100010		1	
STS.L FPUL,@-Rn <sup>*3</sup>	Rn-4→Rn, FPUL→@Rn	0100nnnn01010010		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	
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	
SUB Rm,Rn	Rn-Rm→Rn	0011nnnnmmmm1000		1	
SUBC Rm,Rn	Rn-Rm-T→Rn, ボロー→T	0011nnnnmmmm1010		1	ボロー
SUBV Rm,Rn	Rn-Rm→Rn, アンダフロー→T	0011nnnnmmmm1011		1	アンダ フロー
SWAP.B Rm,Rn	Rm→下位2バイトの上下バイト交換 →Rm	0110nnnnmmmm1000		1	
SWAP.W Rm,Rn	Rm→上下ワード交換→Rn	0110nnnnmmmm1001		1	
TAS.B @Rn	(Rn)が0のとき 1→T, 1→MSBof(Rn)	0100nnnn00011011		3/4 <sup>*7</sup>	テスト 結果
TRAPA #imm	PC→SPC, SR→SSR, (#imm)<<2→TRA, VBR+H'0100→PC	11000011iiiiiiii		6/8 <sup>*8</sup>	
TST #imm,R0	R0 & imm, 結果が0のとき 1→T	11001000iiiiiiii		1	テスト 結果
TST Rm,Rn	Rn & Rm, 結果が0のとき 1→T	0010nnnnmmmm1000		1	テスト 結果
TST.B #imm,@(R0,GBR)	(R0+GBR)&imm,結果が0のとき 1→T	11001100iiiiiiii		3	テスト 結果
XOR #imm,R0	R0 ^ imm → R0	11001010iiiiiiii		1	
XOR Rm,Rn	Rn ^ Rm → Rn	0010nnnnmmmm1010		1	
XOR.B #imm,@(R0,GBR)	(R0+GBR) ^ imm → (R0+GBR)	11001110iiiiiiii		3	
XTRCT Rm,Rn	Rm:Rnの中央32ビット→Rn	0010nnnnmmmm1101		1	

【注】 \*1 通常最小実行ステートを示します。前後の命令との競合関係により実行ステート数は変わります。

\*2 分岐しないときは1ステートになります。

## 7. 命令セット

---

- \*3 浮動小数点命令と FPU に関連する CPU 命令を示します。  
これらは SH-3E でのみ使用可能です。
- \*4 SH3-DSP では 3 ステートになります。
- \*5 SH3-DSP では 5 ステートになります。
- \*6 SH3-DSP では 2 ステートになります。
- \*7 SH3-DSP では 4 ステートになります。
- \*8 SH3-DSP では 8 ステートになります。

## 7.3 DSP データ転送命令の命令セット (SH3-DSP のみ)

DSP データ転送命令を分類別に表 7.13 に示します。

表 7.13 DSP データ転送命令の分類

分類	命令の種類	オペコード	機能	命令数
ダブルデータ転送命令	4	NOPX	X メモリ無操作	14
		MOVX	X メモリデータ転送	
		NOPY	Y メモリ無操作	
		MOVY	Y メモリデータ転送	
シングルデータ転送命令	1	MOVS	シングルデータ転送	16
	計 5			計 30

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、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。

### 7.3.1 ダブルデータ転送命令 (X メモリデータ)

表 7.14 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	実行 ステート	DC ビット
NOPX	No Operation	1111000*0*0*00**	1	
MOVX.W @Ax,Dx	(Ax)→MSW of Dx, 0→LSW of Dx	111100A*D*0*01**	1	
MOVX.W @Ax+,Dx	(Ax)→MSW of Dx, 0→LSW of Dx, Ax + 2→Ax	111100A*D*0*10**	1	
MOVX.W @Ax+lx,Dx	(Ax)→MSW of Dx, 0→LSW of Dx, Ax + lx→Ax	111100A*D*0*11**	1	
MOVX.W Da,@Ax	MSW of Da→(Ax)	111100A*D*1*01**	1	
MOVX.W Da,@Ax+	MSW of Da→(Ax), Ax + 2→Ax	111100A*D*1*10**	1	
MOVX.W Da,@Ax+lx	MSW of Da→(Ax), Ax + lx→Ax	111100A*D*1*11**	1	

## 7. 命令セット

### 7.3.2 ダブルデータ転送命令 (Yメモリデータ)

表 7.15 ダブルデータ転送命令 (Yメモリデータ)

命令	動作	命令コード	実行 ステート	DC ビット
NOPY	No Operation	111100*0*0*0**00	1	
MOVY.W @Ay,Dy	(Ay)→MSW of Dy、0→LSW of Dy	111100* $\Delta$ *D*0**01	1	
MOVY.W @Ay+,Dy	(Ay)→MSW of Dy、0→LSW of Dy、Ay + 2→Ay	111100* $\Delta$ *D*0**10	1	
MOVY.W @Ay+ly,Dy	(Ay)→MSW of Dy、0→LSW of Dy、Ay + ly→Ay	111100* $\Delta$ *D*0**11	1	
MOVY.W Da,@Ay	MSW of Da→(Ay)	111100* $\Delta$ *D*1**01	1	
MOVY.W Da,@Ay+	MSW of Da→(Ay)、Ay + 2→Ay	111100* $\Delta$ *D*1**10	1	
MOVY.W Da,@Ay+ly	MSW of Da→(Ay)、Ay + ly→Ay	111100* $\Delta$ *D*1**11	1	

### 7.3.3 シングルデータ転送命令

表 7.16 シングルデータ転送命令

命令	動作	命令コード	実行 ステート	DC ビット
MOVS.W @-As,Ds	As - 2→As、(As)→MSW of Ds、0→LSW of Ds	111101AADDDDD0000	1	
MOVS.W @As,Ds	(As)→MSW of Ds、0→LSW of Ds	111101AADDDDD0100	1	
MOVS.W @As+,Ds	(As)→MSW of Ds、0→LSW of Ds、As + 2→As	111101AADDDDD1000	1	
MOVS.W @As+ls,Ds	(As)→MSW of Ds、0→LSW of Ds、As + lx→As	111101AADDDDD1100	1	
MOVS.W Ds,@-As	As - 2→As、MSW of Ds→(As)*	111101AADDDDD0001	1	
MOVS.W Ds,@As	MSW of Ds→(As)*	111101AADDDDD0101	1	
MOVS.W Ds,@As+	MSW of Ds→(As)、As + 2→As*	111101AADDDDD1001	1	
MOVS.W Ds,@As+ls	MSW of Ds→(As)、As + lx→As*	111101AADDDDD1101	1	
MOVS.L @-As,Ds	As - 4→As、(As)→Ds	111101AADDDDD0010	1	
MOVS.L @As,Ds	(As)→Ds	111101AADDDDD0110	1	
MOVS.L @As+,Ds	(As)→Ds、As + 4→As	111101AADDDDD1010	1	
MOVS.L @As+ls,Ds	(As)→Ds、As + lx→As	111101AADDDDD1110	1	
MOVS.L Ds,@-As	As - 4→As、Ds→(As)	111101AADDDDD0011	1	
MOVS.L Ds,@As	Ds→(As)	111101AADDDDD0111	1	
MOVS.L Ds,@As+	Ds→(As)、As + 4→As	111101AADDDDD1011	1	
MOVS.L Ds,@As+ls	Ds→(As)、As + lx→As	111101AADDDDD1111	1	

【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 7.17 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 7.17 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(As2)	R3(As3)	R4(Ax0) (As0)	R5(Ax1) (As1)	R6(Ay0)	R7(Ay1)	R8(Ix) (Is)	R9(Iy)
Ax					Yes	Yes				
Ix ( Is )									Yes	
Dx										
Ay							Yes	Yes		
Iy										Yes
Dy										
Da										
As			Yes	Yes	Yes	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	Yes	Yes								
Ay										
Iy										
Dy			Yes	Yes						
Da							Yes	Yes		
As										
Ds	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

【注】 Yes : 設定可能なレジスタ

## 7.4 DSP 演算命令の命令セット (SH3-DSP のみ)

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 7.18 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を表 7.19 に示します。

表 7.18 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, 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, PSUB, PXOR
	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
	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	3 オペランド	ALUop. Sx, Sy, Du MLTop. Se, Sf, Dg	PADDC, PSUBC, PWADD, PWSB, PMULS
演算命令	2 オペランド	ALUop. Sx, Dz ALUop. Sy, Dz ALUop. Sx, Sy	PCMP, PABS, PRND

表 7.19 DSP 命令のオペランドとレジスタの対応

レジスタ	ALU、BPU 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 7.1 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 7.1 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。

DSP 演算命令を分類別に表 7.20 に示します。

表 7.20 DSP 演算命令の分類

分類	命令の種類	オペコード	機能	命令数	
A L U 算 術 演 算 命 令	ALU 固定小数点演算命令	11	PABS	絶対値演算	28
			PADD	加算	
			PADD PMULS	加算と符号付き乗算	
			PADDC	キャリ付き加算	
			PCLR	クリア	
			PCMP	比較	
			PCOPY	転記	
			PNEG	符号反転	
			PSUB	減算	
			PSUB PMULS	減算と符号付き乗算	
			PSUBC	ボロー付き減算	
ALU 整数演算命令	2	PDEC	デクリメント	12	
		PINC	インクリメント		
MSB 検出命令	1	PDMSB	MSB 検出	6	
丸め演算命令	1	PRND	丸め演算	2	
ALU 論理演算命令	3	PAND	論理積演算	9	
		POR	論理和演算		
		PXOR	排他的論理和演算		
固定小数点乗算命令	1	PMULS	符号付き乗算	1	
シ フ ト	算術シフト演算命令 論理シフト演算命令	1	PSHA	算術シフト	4
			PSHL	論理シフト	4
システム制御命令	2	PLDS	システムレジスタのロード	12	
		PSTS	システムレジスタからのストア		
	計	25		計	78

## 7. 命令セット

### 7.4.1 ALU 算術演算命令

#### (1) ALU 固定小数点演算命令

表 7.21 ALU 固定小数点演算命令

命令	動作	命令コード	実行 ステート	DC ビット
PABS Sx,Dz	もし Sx = 0 ならば Sx→Dz もし Sx < 0 ならば 0 - Sx→Dz	111110***** 10001000xx00zzzz	1	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	1	更新
PADD Sx,Sy,Dz	Sx + Sy→Dz	111110***** 10110001xxyyzzzz	1	更新
DCT PADD Sx,Sy,Dz	もし DC = 1 ならば Sx + Sy→Dz もし 0 ならば nop	111110***** 10110010xxyyzzzz	1	
DCF PADD Sx,Sy,Dz	もし DC = 0 ならば Sx + Sy→Dz もし 1 ならば nop	111110***** 10110011xxyyzzzz	1	
PADD Sx,Sy,Du PMULS Se,Sf,Dg	Sx + Sy→Du Se の上位ワード × Sf の上位ワード→Dg	111110***** 0111eefxxyygguu	1	更新
PADDC Sx,Sy,Dz	Sx + Sy + DC→Dz	111110***** 10110000xxyyzzzz	1	更新
PCLR Dz	H'00000000→Dz	111110***** 100011010000zzzz	1	更新
DCT PCLR Dz	もし DC = 1 ならば H'00000000→Dz もし 0 ならば nop.	111110***** 100011100000zzzz	1	
DCF PCLR Dz	もし DC = 0 ならば H'00000000→Dz もし 1 ならば nop.	111110***** 100011110000zzzz	1	
PCMP Sx,Sy	Sx - Sy	111110***** 10000100xxyy0000	1	更新
PCOPY Sx,Dz	Sx→Dz	111110***** 11011001xx00zzzz	1	更新
PCOPY Sy,Dz	Sy→Dz	111110***** 1111100100yyzzzz	1	更新
DCT PCOPY Sx,Dz	もし DC = 1 ならば Sx→Dz もし 0 ならば nop.	111110***** 11011010xx00zzzz	1	
DCT PCOPY Sy,Dz	もし DC = 1 ならば Sy→Dz もし 0 ならば nop.	111110***** 1111101000yyzzzz	1	
DCF PCOPY Sx,Dz	もし DC = 0 ならば Sx→Dz もし 1 ならば nop	111110***** 11011011xx00zzzz	1	
DCF PCOPY Sy,Dz	もし DC = 0 ならば Sy→Dz もし 1 ならば nop	111110***** 1111101100yyzzzz	1	
DCT PCOPY Sy,Dz	もし DC = 1 ならば Sy→Dz もし 0 ならば nop.	111110***** 1111101000yyzzzz	1	
DCF PCOPY Sx,Dz	もし DC = 0 ならば Sx→Dz もし 1 ならば nop	111110***** 11011011xx00zzzz	1	
DCF PCOPY Sy,Dz	もし DC = 0 ならば Sy→Dz もし 1 ならば nop	111110***** 1111101100yyzzzz	1	

## 7. 命令セット

命令	動作	命令コード	実行 ステート	DCビット
PNEG Sx,Dz	0 - Sx→Dz	111110***** 11001001xx00zzzz	1	更新
PNEG Sy,Dz	0 - Sy→Dz	111110***** 1110100100yyzzzz	1	更新
DCT PNEG Sx,Dz	もし DC = 1 ならば 0 - Sx→Dz もし 0 ならば nop.	111110***** 11001010xx00zzzz	1	
DCT PNEG Sy,Dz	もし DC = 1 ならば 0 - Sy→Dz もし 0 ならば、nop.	111110***** 1110101000yyzzzz	1	
DCF PNEG Sx,Dz	もし DC = 0 ならば 0 - Sx→Dz もし 1 ならば nop.	111110***** 11001011xx00zzzz	1	
DCF PNEG Sy,Dz	もし DC = 0 ならば 0 - Sy→Dz もし 1 ならば nop.	111110***** 1110101100yyzzzz	1	
PSUB Sx,Sy,Dz	Sx - Sy→Dz	111110***** 10100001xxyyzzzz	1	更新
DCT PSUB Sx,Sy,Dz	もし DC = 1 ならば Sx - Sy→Dz もし 0 ならば nop	111110***** 10100010xxyyzzzz	1	
DCF PSUB Sx,Sy,Dz	もし DC = 0 ならば Sx - Sy→Dz もし 1 ならば nop	111110***** 10100011xxyyzzzz	1	
PSUB Sx,Sy,Du PMULS Se,Sf,Dg	Sx - Sy→Du Se の上位ワード × Sf の上位ワード→Dg	111110***** 0110eefxxyygguu	1	更新
PSUBC Sx,Sy,Dz	Sx - Sy - DC→Dz	111110***** 10100000xxyyzzzz	1	更新

## 7. 命令セット

### (2) ALU 整数演算命令

表 7.22 ALU 整数演算命令

命令	動作	命令コード	実行 ステート	DC ビット
PDEC Sx,Dz	Sx の上位ワード - 1→Dz の上位ワード Dz の下位ワードをクリア	111110***** 10001001xx00zzzz	1	更新
PDEC Sy,Dz	Sy の上位ワード - 1→Dz の上位ワード Dz の下位ワードをクリア	111110***** 10101001xx00zzzz	1	更新
DCT PDEC Sx,Dz	もし DC = 1 ならば Sx の上位ワード - 1→Dz の上位ワード Dz の下位ワードクリア もし 0 ならば nop.	111110***** 10001010xx00zzzz	1	
DCT PDEC Sy,Dz	もし DC = 1 ならば Sy の上位ワード - 1→Dz の上位ワード Dz の下位ワードクリア もし 0 ならば nop.	111110***** 10101010xx00zzzz	1	
DCF PDEC Sx,Dz	もし DC = 0 ならば Sx の上位ワード - 1→Dz の上位ワード Dz の下位ワードクリア もし 1 ならば nop.	111110***** 10001011xx00zzzz	1	
DCF PDEC Sy,Dz	もし DC = 0 ならば Sy の上位ワード - 1→Dz の上位ワード Dz の下位ワードクリア もし 1 ならば nop.	111110***** 10101011xx00zzzz	1	
PINC Sx,Dz	Sx の上位ワード + 1→Dz の上位ワード Dz の下位ワードクリア	111110***** 10011001xx00zzzz	1	更新
PINC Sy,Dz	Sy の上位ワード + 1→Dz の上位ワード Dz の下位ワードクリア	111110***** 1011100100yyzzzz	1	更新
DCT PINC Sx,Dz	もし DC = 1 ならば Sx の上位ワード + 1→Dz の上位ワード Dz の下位ワードクリア もし 0 ならば nop.	111110***** 10011010xx00zzzz	1	
DCT PINC Sy,Dz	もし DC = 1 ならば Sy の上位ワード + 1→Dz の上位ワード Dz の下位ワードクリア もし 0 ならば nop.	111110***** 1011101000yyzzzz	1	
DCF PINC Sx,Dz	もし DC = 0 ならば Sx の上位ワード + 1→Dz の上位ワード Dz の下位ワードクリア もし 1 ならば nop.	111110***** 10011011xx00zzzz	1	
DCF PINC Sy,Dz	もし DC = 0 ならば Sy の上位ワード + 1→Dz の上位ワード Dz の下位ワードクリア もし 1 ならば nop.	111110***** 1011101100yyzzzz	1	

## (3) MSB 検出命令

表 7.23 MSB 検出命令

命令	動作	命令コード	実行 ステート	DC ビット
PDMSB Sx,Dz	Sx データの MSB 位置→Dz の上位ワード Dz の下位ワードクリア	111110***** 10011101xx00zzzz	1	更新
PDMSB Sy,Dz	Sy データの MSB 位置→Dz の上位ワード Dz の下位ワードクリア	111110***** 1011110100yyzzzz	1	更新
DCT PDMSB Sx,Dz	もし DC = 1 ならば Sx データの MSB 位置→Dz の上位ワード Dz の下位ワードクリア もし 0 ならば nop.	111110***** 10011110xx00zzzz	1	
DCT PDMSB Sy,Dz	もし DC = 1 ならば Sy データの MSB 位置→Dz の上位ワード Dz の下位ワードクリア もし 0 ならば nop.	111110***** 1011111000yyzzzz	1	
DCF PDMSB Sx,Dz	もし DC = 0 ならば Sx データの MSB 位置→Dz の上位ワード Dz の下位ワードクリア もし 1 ならば nop.	111110***** 10011111xx00zzzz	1	
DCF PDMSB Sy,Dz	もし DC = 0 ならば Sy データの MSB 位置→Dz の上位ワード Dz の下位ワードクリア もし 1 ならば nop.	111110***** 1011111100yyzzzz	1	

## (4) 丸め演算命令

表 7.24 丸め演算命令

命令	動作	命令コード	実行 ステート	DC ビット
PRND Sx,Dz	Sx + H'00008000→Dz Dz の下位ワードクリア	111110***** 10011000xx00zzzz	1	更新
PRND Sy,Dz	Sy + H'00008000→Dz Dz の下位ワードクリア	111110***** 1011100000yyzzzz	1	更新

## 7. 命令セット

### 7.4.2 ALU 論理演算命令

表 7.25 ALU 論理演算命令

命令	動作	命令コード	実行 ステート	DC ビット
PAND Sx,Sy,Dz	Sx & Sy → Dz Dz の下位ワードクリア	111110***** 10010101xxyyzzzz	1	更新
DCT PAND Sx,Sy,Dz	もし DC = 1 ならば Sx&Sy→Dz Dz の下位ワードクリア もし 0、nop.	111110***** 10010110xxyyzzzz	1	
DCF PAND Sx,Sy,Dz	もし DC = 0 ならば Sx&Sy→Dz Dz の下位ワードクリア もし 1 ならば nop.	111110***** 10010111xxyyzzzz	1	
POR Sx,Sy,Dz	Sx   Sy → Dz Dz の下位ワードクリア	111110***** 10110101xxyyzzzz	1	更新
DCT POR Sx,Sy,Dz	もし DC = 1 ならば Sx Sy→Dz Dz の下位ワードクリア 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 の下位ワードクリア もし 1 ならば nop.	111110***** 10110111xxyyzzzz	1	
PXOR Sx,Sy,Dz	Sx ^ Sy → Dz Dz の下位ワードクリア	111110***** 10100101xxyyzzzz	1	更新
DCT PXOR Sx,Sy,Dz	もし DC = 1 ならば Sx^Sy→Dz Dz の下位ワードクリア 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 の下位ワードクリア もし 1 ならば nop.	111110***** 10100111xxyyzzzz	1	

### 7.4.3 固定小数点乗算命令

表 7.26 固定小数点乗算命令

命令	動作	命令コード	実行 ステート	DC ビット
PMULS Se,Sf,Dg	Se の上位ワード × Sf の上位ワード → Dg	111110***** 0100eef0000gg00	1	



## 7.4.4 シフト演算命令

## (1) 算術シフト演算命令

表 7.27 算術シフト演算命令

命令	動作	命令コード	実行 状態	DC 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
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***** 00000iiiiiiiizzzz	1	更新

## (2) 論理シフト演算命令

表 7.28 論理シフト演算命令

命令	動作	命令コード	実行 状態	DC ビット
PSHL Sx,Sy,Dz	もし Sy = 0 ならば Sx < < Sy → Dz, Dz の下位ワードクリア もし Sy < 0 ならば Sx > > Sy → Dz, Dz の下位ワードクリア	111110***** 10000001xxyyzzzz	1	更新
DCT PSHL Sx,Sy,Dz	もし DC = 1 & Sy = 0 ならば Sx < < Sy → Dz, Dz の下位ワードクリア もし DC = 1 & Sy < 0 ならば Sx > > Sy → Dz, Dz の下位ワードクリア 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 の下位ワードクリア もし DC = 0 & Sy < 0 ならば Sx > > Sy → Dz, Dz の下位ワードクリア 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 の下位ワードクリア もし Imm < 0 ならば Dz > > Imm → Dz, Dz の下位ワードクリア	111110***** 00010iiiiiiiizzzz	1	更新

## 7. 命令セット

### 7.4.5 システム制御命令

表 7.29 システム制御命令

命令	動作	命令コード	実行 ステート	DC ビット
PLDS Dz,MACH	Dz→MACH	111110***** 111011010000zzzz	1	
PLDS Dz,MACL	Dz→MACL	111110***** 111111010000zzzz	1	
DCT PLDS Dz,MACH	もし DC = 1 ならば Dz→MACH もし 0 ならば nop.	111110***** 111011100000zzzz	1	
DCT PLDS Dz,MACL	もし DC = 1 ならば Dz→MACL もし 0 ならば nop.	111110***** 111111100000zzzz	1	
DCF PLDS Dz,MACH	もし DC = 0 ならば Dz→MACH もし 1 ならば nop.	111110***** 111011110000zzzz	1	
DCF PLDS Dz,MACL	もし DC = 0 ならば Dz→MACL もし 1 ならば nop.	111110***** 111111110000zzzz	1	
PSTS MACH,Dz	MACH→Dz	111110***** 110011010000zzzz	1	
PSTS MACL,Dz	MACL→Dz	111110***** 110111010000zzzz	1	
DCT PSTS MACH,Dz	もし DC = 1 ならば MACH→Dz もし 0 ならば nop.	111110***** 110011100000zzzz	1	
DCT PSTS MACL,Dz	もし DC = 1 ならば MACL→Dz もし 0 ならば nop.	111110***** 110111100000zzzz	1	
DCF PSTS MACH,Dz	もし DC = 0 ならば MACH→Dz もし 1 ならば nop.	111110***** 110011110000zzzz	1	
DCF PSTS MACL,Dz	もし DC = 0 ならば MACL→Dz もし 1 ならば nop.	111110***** 110111110000zzzz	1	





---

## 8. 各命令の説明

---

命令の解説に使用するフォーマットについて説明します。実際の解説は次節からとなります。

### 8.1 命令説明のフォーム

命令の機能

(遅延分岐命令、または割り込み禁止命令の表示)

---

書式	動作概略	命令コード	実行 ステート	Tビット
アセンブラの入力書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ビットの値を表示しています

【注】 8.2節ではSH-3、SH-3E、SH3-DSP共通のCPU命令を、8.3節ではSH-3Eのみ使用可能な浮動小数点命令を、8.4節ではSH3-DSPのみ使用可能なDSPデータ転送命令を、8.5節ではSH3-DSPのみ使用可能なDSP演算命令を説明します。

浮動小数点命令の実行サイクルはレイテンシとピッチの2つの値により定義されます。

レイテンシは演算結果の値を生成するのに必要なサイクルを示し、ピッチは次の命令を開始するために要する待ちサイクルを示します。ほとんどのCPU命令では、レイテンシとピッチは同一であり、これらの実行サイクルは1つのサイクルとして示されています。

#### (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特に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。ここでは以下の資源の使用を仮定しています。

```
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。

## 8. 各命令の説明

---

```
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\_4) のスロット命令に実行を移します。これはたとえば “Delay\_Slot(4);” のとき、4 番地ではなく 0 番地の命令に実行が移ることを意味します。また、この関数から以下の命令に実行が移されようとする、その直前に以下の命令をスロット不当命令として検出します。(遅延スロット命令が以下の命令だと、スロット不当命令となります。)

- (1) 未定義命令
- (2) PCを書き換える命令：  
BF, BT, BRA, BSR, JMP, JSR, RTS, RTE, TRAPA, BF/S, BT/S, BRAF, BSRF, LDC Rm, SR, LDS.L @Rm+, SR
- (3) ユーザモードで遅延スロットが特権命令：

特権命令：LDC, STC, RTE, LDTLB, SLEEP  
ただし、LDC/STC で GBR をアクセスする命令を除く

```
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    dummy0:4;
    unsigned long    RC0:12;
    unsigned long    dummy1:4;
    unsigned long    DMY0:1;
    unsigned long    DMX0:1;
    unsigned long    M0:1;
    unsigned long    Q0:1;
    unsigned long    I0:4;
    unsigned long    RF10:1;
    unsigned long    RF00:1;
    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)
#define RF1 ((*(struct SR0 *)(&SR)).RF10)
#define RF0 ((*(struct SR0 *)(&SR)).RF00)
```

## SR 内ビットの定義

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

## エラー表示関数

これ以外に、PC は現在実行中の命令の 4 バイト (2 命令) 先を示しているものと仮定しています。これは、たとえば “PC=4;” は 4 番地ではなく 0 番地の命令に実行が移ることを意味します。

## (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、「クロスアセンブラ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.l 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
.arepeat 16 16 回繰り返し展開
.arepeat 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 Ver 1.0 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。

- \*1 下記のディスプレイースメント (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) を行う前の値を記載しています。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ご参照ください。
  - @(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
- \*2 命令コード 16 ビットのうち命令として割り当てられないコード、またはユーザモードでの特権命令 (GBR をアクセスする命令は除く) は、一般不当命令として扱われ、不当命令例外処理が発生します。

例 H'FFFF [ 一般不当命令 ]

- \*3 BRA, BT/S などの遅延分岐命令の次命令が一般不当命令または PC を書き換える命令(分岐命令など)であると(これをスロット不当命令といいます)、スロット不当命令例外処理を発生します。

例 1 ....

BRA LABEL

.data.W H'FFFF ←スロット不当命令

.... [ H'FFFF は本来一般不当命令 ]

例 2 RTE

BT/S LABEL ←スロット不当命令

- \*4 SH3-DSP では、3 命令以下の繰り返しプログラム(ループ)内または、4 命令以上の繰り返しプログラム(ループ)内の最後の 3 命令に、一般不当命令、PC を書き換える命令(分岐命令など)または、SR、RS、RE レジスタを書き換える命令(SETRC、LDRS、LDRE、LDC)が存在すると、不当命令例外処理を発生します。詳しくは、「5.12 DSP 繰り返し(ループ)制御」を参照してください。



## 8.2 命令の説明 (SH-3、SH-3E、SH3-DSP に共通する CPU 命令の説明)

### 8.2.1 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	0111nnnniiiiiii	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) 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]+=(0xFFFFF00 | (long)i);
    PC+=2;
}
```

#### (3) 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'FFFFFFF
```

## 8. 各命令の説明

---

### 8.2.2 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) 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;
}
```

#### (3) 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'FFFFFFF
                                           ;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
```

### 8.2.3 ADDV ADD with V flag overflow check : 算術演算命令 オーバーフロー付き 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, オーバーフロー→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。オーバーフローが発生すると、T ビットをセットします。

#### (2) 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;
}
```

#### (3) 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
```

## 8. 各命令の説明

---

### 8.2.4 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	11001001iiiiiii	1	
AND.B #imm,@(R0,GBR)	(R0+GBR) & imm → (R0+GBR)	11001101iiiiiii	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
```

## 8. 各命令の説明

### 8.2.5 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 disp × 2 + PC → PC, T=1 のとき nop	10001011dddddddd	3/1	

#### (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、次の命令を実行します。逆に T=0 のとき、分岐します。

分岐先は 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 2 命令後の先頭アドレスです。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。分岐先に届かないときは BRA 命令などとの組み合わせで対応する必要があります。

#### (2) 注意

分岐するときは 3 ステート、分岐しないときは 1 ステートになります。

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せん。

#### (3) 動作内容

```
BF(long d) /* BF disp */
{
    long disp;

    if ((d&0x80)==0) disp=(0x000000FF & (long)d);
    else disp=(0xFFFFFFFF | (long)d);
    if (T==0) PC=PC+(disp<<1)+4;
    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
NOP                 ;←BF 命令で分岐先アドレス計算に用いる PC の位置
TRGET_F:            ;←BF 命令の分岐先
```

## 8.2.6 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 label	T=0 のとき disp × 2+PC→PC, T=1 のとき nop	100011111ddddddd	2/1	

## (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、次の命令を実行します。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 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 2 命令後の先頭アドレスです。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。分岐先に届かないときは BRA 命令などとの組み合わせで対応する必要があります。

## (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。

分岐するときは 2 ステート、分岐しないときは 1 ステートになります。

分岐成立の場合でかつ、直後の命令が、一般不当命令、または PC を書き換える命令（分岐命令など）の場合、それをスロット不当命令として認識します。

分岐不成立の場合、スロット不当命令はありません。

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。

## (3) 動作内容

```

BFS(long d) /* BFS disp */
{
    long disp;
    unsigned long temp;

    temp=PC;
    if ((d&0x80)==0) disp=(0x000000FF & (long)d);
    else disp=(0xFFFFFFFF00 | (long)d);
    if (T==0) {
        PC=PC+(disp<<1)+4;
        Delay_Slot(temp+2);
    }
    else PC+=2;
}

```

## 8. 各命令の説明

---

### (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 に分岐します。
ADD R0,R1           ;分岐に先立ち実行します。
NOP                 ;←BF/S 命令で分岐先アドレス計算に用いる PC の位置
TRGET_F             ;←BF/S 命令の分岐先
```

【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。



## 8.2.7 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	disp × 2+PC→PC	1010ddddddddddd	2	

## (1) 説明

無条件の遅延分岐命令です。分岐先はPCにディスプレイメントを加えたアドレスです。PCは、本命令の2命令後の先頭アドレスです。12ビットディスプレイ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、分岐先アドレスをMOV命令でレジスタに転送した上で、JMP命令への変更が必要です。

## (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。

遅延分岐命令直後の遅延スロットに本命例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

```

BRA(long d) /* BRA disp */
{
    unsigned long temp;
    long disp;

    if ((d&0x800)==0) disp=(0x00000FFF & d);
    else disp=(0xFFFFF000 | d);
    temp=PC;
    PC=PC+(disp<<1)+4;
    Delay_Slot(temp+2);
}

```

## 8. 各命令の説明

---

### (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                ;←BRA 命令で分岐先アドレス計算に用いる PC の位置
TRGET:             ;←BRA 命令の分岐先
```

【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。

## 8.2.8 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 Rm	Rm+PC→PC	0000mmmm00100011	2	

## (1) 説明

無条件の遅延分岐命令です。分岐先は PC に汎用レジスタ Rn の内容の 32 ビットを加えたアドレスです。PC は、本命令の 2 命令後の先頭アドレスです。

## (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。

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せん。

## (3) 動作内容

```
BRAF(long m) /* BRAF Rm */
{
    unsigned long temp;

    temp=PC;
    PC+=R[m];
    Delay_Slot(temp+2);
}
```

## (4) 使用例

```
MOV.L #(TARGET-BSRF_PC),R0 ;ディスプレイメントを設定します。
BRAF TRGET ;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: ;←BRAF 命令で分岐先アドレス計算に用いる PC の位置
NOP
TRGET: ;←BRAF 命令の分岐先
```

**【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。

## 8.2.9 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→PR, disp × 2+PC→PC	1011ddddddddddd	2	

## (1) 説明

指定され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。PC の内容を PR に退避し、PC にディスプレースメントを加えたアドレスへ分岐します。PC は、本命令の 2 命令後の先頭アドレスです。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、分岐先アドレスを MOV 命令でレジスタに転送した上で、JSR 命令への変更が必要です。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、分岐します。

本命令と直後の命令の間には、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。

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。

本命令の直後の命令で使用される PR は、本命令で更新したものです。

また、直後の命令で命令フェッチ以外の再実行型例外が発生した場合、PR は本命令で更新されています。本命令を再実行することで回復してください。

## (3) 動作内容

```
BSR(long d)          /* BSR disp */
{
    long disp;

    if ((d&0x800)==0) disp=(0x00000FFF & d);
    else disp=(0xFFFFF000 | d);
    PR=PC;
    PC=PC+(disp<<1)+4;
    Delay_Slot(PR+2);
}
```

## (4) 使用例

```
BSR TRGET          ;TRGET へ分岐します。
MOV R3,R4          ;分岐に先立ち MOV 命令を実行します。
ADD R0,R1          ;←BSR で分岐先アドレス計算に用いる PC の位置であり
                  ;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 命令を実行します。
```

【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。

## 8. 各命令の説明

### 8.2.10 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 Rm	PC→PR, Rm+PC→PC	0000mmmm00000011	2	

#### (1) 説明

指定され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。PCの内容をPRに退避します。分岐先はPC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。PCは、本命令の2命令後の先頭アドレスです。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。

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。

本命令の直後の命令で使用されるPRは、本命令で更新したものです。

また、直後の命令で命令フェッチ以外の再実行型例外が発生した場合、PRは本命令で更新されています。本命令を再実行することで回復してください。

#### (3) 動作内容

```
BSRF(long m) /* BSRF Rm */
{
  PR=PC;
  PC+=R[m];
  Delay_Slot(PR+2);
}
```

#### (4) 使用例

```
MOV.L #(TARGET-BSRF_PC), R0 ;ディスプレイメントを設定します
BSRF @R0 ;TARGETへ分岐します。
MOV R3, R4 ;分岐に先立ちMOV命令を実行します。
BSRF_PC: ;←BSRF命令で分岐先アドレス計算に用いるPCの位置
ADD R0, R1 ;
.....
.....
TARGET: ;←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。
```

【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。

## 8. 各命令の説明

### 8.2.11 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 disp×2+PC→PC, T=0 のとき nop	10001001ddddddd	3/1	

#### (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、次の命令を実行します。

分岐先は 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 2 命令後の先頭アドレスです。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。分岐先に届かないときは BRA 命令などとの組み合わせで対応する必要があります。

#### (2) 注意

分岐するときは 3 ステート、分岐しないときは 1 ステートになります。

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。

#### (3) 動作内容

```
BT(long d) /* BT disp */
{
    long disp;

    if ((d&0x80)==0) disp=(0x000000FF & (long)d);
    else disp=(0xFFFFFFFF0 | (long)d);
    if (T==1) PC=PC+(disp<<1)+4;
    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                 ;←BT 命令で分岐先アドレス計算に用いる PC の位置
TRGET_T:            ;←BT 命令の分岐先
```



## 8.2.12 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 disp × 2+PC→PC, T=0 のとき nop	10001101dddddddd	2/1	

## (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、次の命令を実行した後で分岐します。T=0 のとき、次の命令を実行します。

分岐先は 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 2 命令後の先頭アドレスです。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。分岐先に届かないときは BRA 命令などとの組み合わせで対応する必要があります。

## (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。

分岐するときは 2 ステート、分岐しないときは 1 ステートになります。

分岐成立の場合でかつ、直後の命令が、一般不当命令、または PC を書き換える命令（分岐命令など）の場合、それをスロット不当命令として認識します。

分岐不成立の場合、スロット不当命令はありません。

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。

## (3) 動作内容

```

BTS(long d) /* BTS disp */
{
    long disp;
    unsigned long temp;

    temp=PC;
    if ((d&0x80)==0) disp=(0x000000FF & (long)d);
    else disp=(0xFFFFFFFF0 | (long)d);
    if (T==1) {
        PC=PC+(disp<<1)+4;
        Delay_Slot(temp+2);
    }
    else PC+=2;
}

```

## 8. 各命令の説明

---

### (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                 ;←BT/S 命令で分岐先アドレス計算に用いる PC の位置
TRGET_T:           ;←BT/S 命令の分岐先
```

【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更新しても、変更前のレジスタ内容が分岐先アドレスとなります。

## 8.2.13 CLRMAC CLear MAC register : システム制御命令

MACレジスタのクリア

システム制御命令

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) 動作内容

```
CLRMAC ( ) /* CLRMAC */
{
    MACH=0;
    MACL=0;
    PC+=2;
}
```

## (3) 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+ ;
```

## 8. 各命令の説明

---

### 8.2.14 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) 動作内容

```
CLRS ( ) /* CLRS */  
{  
    S=0;  
    PC+=2;  
}
```

#### (3) 使用例

```
CLRS ;実行前 S=1  
      ;実行後 S=0
```

## 8.2.15 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	00000000000001000	1	0

#### (1) 説明

Tビットをクリアします。

#### (2) 動作内容

```
CLRT( ) /* CLRT */
{
    T=0;
    PC+=2;
}
```

#### (3) 使用例

```
CLRT ;実行前 T=1
      ;実行後 T=0
```

## 8. 各命令の説明

### 8.2.16 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 状態	Tビット
CMP/EQ Rm,Rn	Rn=Rm のとき 1→T	0011nnnnmmmm0000	1	比較結果
CMP/GE Rm,Rn	有符号で Rn Rm のとき 1→T	0011nnnnmmmm0011	1	比較結果
CMP/GT Rm,Rn	有符号で Rn>Rm のとき 1→T	0011nnnnmmmm0111	1	比較結果
CMP/HIRm,Rn	無符号で Rn>Rm のとき 1→T	0011nnnnmmmm0110	1	比較結果
CMP/HS Rm,Rn	無符号で Rn Rm のとき 1→T	0011nnnnmmmm0010	1	比較結果
CMP/PL Rn	Rn>0 のとき 1→T	0100nnnn00010101	1	比較結果
CMP/PZ Rn	Rn 0 のとき 1→T	0100nnnn00010001	1	比較結果
CMP/STR Rm,Rn	Rn と Rm のいずれかのバイトが等しいとき 1→T	0010nnnnmmmm1100	1	比較結果
CMP/EQ #imm,R0	R0=imm のとき 1→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。下表に示す 8 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。

CMP ニーモニック

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/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) 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;
}
```

## 8. 各命令の説明

---

```
    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)>>12;
    HL=(temp&0x00FF0000)>>8;
    LH=(temp&0x0000FF00)>>4;
    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=(0xFFFFF00 | (long i));
    if (R[0]==imm) T=1;
    else T=0;
    PC+=2;
}
```

### (3) 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="XYZC"
BT TRGET_T ;T=1 なので分岐します。
```



## 8.2.17 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 状態	Tビット
DIV0S Rm,Rn	Rn の MSB→Q, Rm の MSB→M, M^Q→T	0010nnnnnnmmmm0111	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) 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;
}
```

#### (3) 使用例

DIV1の使用例を参照してください。

## 8. 各命令の説明

---

### 8.2.18 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) 動作内容

```
DIV0U( ) /* DIV0U */  
{  
    M=Q=T=0;  
    PC+=2;  
}
```

#### (3) 使用例

DIV1の使用例を参照してください。

## 8.2.19 DIV1 DIVide 1 step : 算術演算命令 除算

書式	動作概略	命令コード	実行 ステート	Tビット
DIV1 Rm,Rn	1ステップ除算 (Rn ÷ Rm)	0011nnnnmmmm0100	1	計算結果

### (1) 説明

汎用レジスタ Rn の 32 ビットの内容 (被除数) を Rm の内容 (除数) で 1 桁分の除算 (1 ステップ除算) を実行する命令です。本命令単独でまたは他の命令と組み合わせて繰り返し実行し商を求めます。この繰り返し中は、指定したレジスタと M、Q、T ビットを書き換えしないでください。

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、

$$(\text{余り}) = (\text{被除数}) - (\text{除数}) \times (\text{商})$$

として求めてください。

ゼロ除算とオーバフローの検出および剰余の演算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。

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下記の使用例を参考にしてください。

### (2) 動作内容

```
DIV1(long m, long n) /* DIV1 Rm,Rn */
{
    unsigned long tmp0;
    unsigned char old_q, tmp1;

    old_q=Q;
    Q=(unsigned char)((0x80000000 & R[n])!=0);
    R[n]<<=1;
    R[n]|=(unsigned long)T;
    switch(old_q){
    case 0:switch(M){
        case 0:tmp0=R[n];
            R[n]-=R[m];
            tmp1=(R[n]>tmp0);
            switch(Q){
            case 0:Q=tmp1;
                break;
```

## 8. 各命令の説明

---

```
        case 1:Q=(unsigned char) (tmp1==0);
            break;
        }
        break;
    case 1:tmp0=R[n];
        R[n]+=R[m];
        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]+=R[m];
        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]-=R[m];
        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;
}

```

## (3) 使用例 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       ;オーバーフローチェック
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

```

## (4) 使用例 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       ;オーバーフローチェック
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

```

## (5) 使用例 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

```

## 8. 各命令の説明

---

```
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 回繰り返す
. 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)
```

### (6) 使用例 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 回繰り返す
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)
```

## 8.2.20 DMULS.L Double-length MULtiplly 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 $R_n \times R_m \rightarrow MACH, MACL$	0011nnnnmmmm1101	2(~5)	

#### (1) 説明

汎用レジスタ  $R_n$  の内容と  $R_m$  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) 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
```

## 8. 各命令の説明

---

```
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;
}
```

### (3) 使用例

DMULS	R0,R1	;実行前 R0=H'FFFFFFFE,R1=H'00005555 ;実行後 MACH=H'FFFFFFF,MACL=H'FFFF5556
STS	MACH,R0	;演算結果(上位)を得る
STS	MACL,R0	;演算結果(下位)を得る



## 8.2.21 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 $R_n \times R_m \rightarrow MACH, MACL$	0011nnnnmmmm0101	2(~5)	

### (1) 説明

汎用レジスタ  $R_n$  の内容と  $R_m$  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) 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;
```

## 8. 各命令の説明

---

```
    MACL=Res0;  
    PC+=2;  
}
```

### (3) 使用例

DMULU	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, R0	;演算結果(下位)を得る

## 8.2.22 DT Decrement and Test : 算術演算命令

## デクリメントとテスト

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以外るとき 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 以外るとき、T ビットを 0 にセットします。

## (2) 動作内容

```
DT(long n) /* DT Rn */
{
    R[n]--;
    if (R[n]==0) T=1;
    else T=0;
    PC+=2;
}
```

## (3) 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)。
```

## 8. 各命令の説明

---

### 8.2.23 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	0110nnnnnnmmmm1110	1	
EXTS.W Rm,Rn	Rm をワードから符号拡張→Rn	0110nnnnnnmmmm1111	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 の内容をコピーします。

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 の内容をコピーします。

#### (2) 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]|=0xFFFFFFFF00;  
    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;
```

```
}
```

#### (3) 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
```

## 8.2.24 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	0110nnnnnnmmmm1100	1	
EXTU.W Rm,Rn	Rm をワードからゼロ拡張→Rn	0110nnnnnnmmmm1101	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 を書き込みます。ワード指定のとき、Rn のビット 16 からビット 31 に 0 を書き込みます。

### (2) 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;
}
```

### (3) 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
```

## 8. 各命令の説明

### 8.2.25 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 @Rm	Rm→PC	0100mmmm00101011	2	

#### (1) 説明

レジスタ間接で指定したアドレスへ無条件に遅延分岐します。分岐先は汎用レジスタ Rm の内容の 32 ビットデータで表されるアドレスです。

#### (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。

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せん。

#### (3) 動作内容

```
JMP(long m)          /* JMP @Rm */
{
    unsigned long temp;

    temp=PC;
    PC=R[m]+4;
    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.1 TRGET   ;ジャンプテーブル
.....
TRGET:   ADD      #1, R1   ;←分岐先
```

**【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。

## 8.2.26 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 @Rm	PC→Rm, Rm→PC	0100mmmm00001011	2	

## (1) 説明

レジスタ間接で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。PCの内容をPRに退避し、汎用レジスタRmの内容の32ビットデータで表されるアドレスへ分岐します。

退避されるPCは、本命令の2命令後の先頭アドレスです。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。

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。

本命令の直後の命令で使用されるPRは、本命令で更新したものです。

また、直後の命令で命令フェッチ以外の再実行型例外が発生した場合、PRは本命令で更新されません。本命令を再実行することで回復してください。

## (3) 動作内容

```
JSR(long m)          /* JSR @Rm */
{
    PR=PC;
    PC=R[m]+4;
    Delay_Slot(PR+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。
```

【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。



## 8.2.27 LDC Load to Control register : システム制御命令

コントロールレジスタへのロード

(特権命令)

書式	動作概略	命令コード	実行 ステート	Tビット
LDC Rm,SR	Rm→SR	0100mmmm00001110	5	LSB
LDC Rm,GBR	Rm→GBR	0100mmmm00011110	1/3 <sup>*1</sup>	
LDC Rm,VBR	Rm→VBR	0100mmmm00101110	1/3 <sup>*1</sup>	
LDC Rm,SSR	Rm→SSR	0100mmmm00111110	1/3 <sup>*1</sup>	
LDC Rm,SPC	Rm→SPC	0100mmmm01001110	1/3 <sup>*1</sup>	
LDC Rm,MOD	Rm→MOD	0100mmmm01011110	3	
LDC Rm,RE	Rm→RE	0100mmmm01111110	3	
LDC Rm,RS	Rm→RS	0100mmmm01101110	3	
LDC Rm,R0_BANK	Rm→R0_BANK	0100mmmm10001110	1/3 <sup>*1</sup>	
LDC Rm,R1_BANK	Rm→R1_BANK	0100mmmm10011110	1/3 <sup>*1</sup>	
LDC Rm,R2_BANK	Rm→R2_BANK	0100mmmm10101110	1/3 <sup>*1</sup>	
LDC Rm,R3_BANK	Rm→R3_BANK	0100mmmm10111110	1/3 <sup>*1</sup>	
LDC Rm,R4_BANK	Rm→R4_BANK	0100mmmm11001110	1/3 <sup>*1</sup>	
LDC Rm,R5_BANK	Rm→R5_BANK	0100mmmm11011110	1/3 <sup>*1</sup>	
LDC Rm,R6_BANK	Rm→R6_BANK	0100mmmm11101110	1/3 <sup>*1</sup>	
LDC Rm,R7_BANK	Rm→R7_BANK	0100mmmm11111110	1/3 <sup>*1</sup>	
LDC.L @Rm+,SR	(Rm)→SR, Rm+4→Rm	0100mmmm00000111	7	
LDC.L @Rm+,GBR	(Rm)→GBR, Rm+4→Rm	0100mmmm00010111	1/5 <sup>*2</sup>	
LDC.L @Rm+,VBR	(Rm)→VBR, Rm+4→Rm	0100mmmm00100111	1/5 <sup>*2</sup>	
LDC.L @Rm+,SSR	(Rm)→SSR, Rm+4→Rm	0100mmmm00110111	1/5 <sup>*2</sup>	
LDC.L @Rm+,SPC	(Rm)→SPC, Rm+4→Rm	0100mmmm01000111	1/5 <sup>*2</sup>	
LDC.L @Rm+,MOD	(Rm)→MOD, Rm+4→Rm	0100mmmm01010111	5	
LDC.L @Rm+,RE	(Rm)→RE, Rm+4→Rm	0100mmmm01101111	5	
LDC.L @Rm+,RS	(Rm)→RS, Rm+4→Rm	0100mmmm01100111	5	
LDC.L @Rm+,R0_BANK	(Rm)→R0_BANK, Rm+4→Rm	0100mmmm10000111	1/5 <sup>*2</sup>	
LDC.L @Rm+,R1_BANK	(Rm)→R1_BANK, Rm+4→Rm	0100mmmm10010111	1/5 <sup>*2</sup>	
LDC.L @Rm+,R2_BANK	(Rm)→R2_BANK, Rm+4→Rm	0100mmmm10100111	1/5 <sup>*2</sup>	
LDC.L @Rm+,R3_BANK	(Rm)→R3_BANK, Rm+4→Rm	0100mmmm10110111	1/5 <sup>*2</sup>	
LDC.L @Rm+,R4_BANK	(Rm)→R4_BANK, Rm+4→Rm	0100mmmm11000111	1/5 <sup>*2</sup>	
LDC.L @Rm+,R5_BANK	(Rm)→R5_BANK, Rm+4→Rm	0100mmmm11010111	1/5 <sup>*2</sup>	
LDC.L @Rm+,R6_BANK	(Rm)→R6_BANK, Rm+4→Rm	0100mmmm11100111	1/5 <sup>*2</sup>	
LDC.L @Rm+,R7_BANK	(Rm)→R7_BANK, Rm+4→Rm	0100mmmm11110111	1/5 <sup>*2</sup>	

【注】 \*1 SH3-DSP では 3 ステートになります。

\*2 SH3-DSP では 5 ステートになります。

## 8. 各命令の説明

---

### (1) 説明

ソースオペランドをコントロールレジスタ SR、GBR、VBR、SSR、SPC、MOD、RE、RS または R0\_BANK ~ R7\_BANK に格納します。LDC Rm,GBR と LDC.L @Rm+, GBR を除いた LDC と LDC.L 命令は、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。ただし LDC Rm,GBR と LDC.L @Rm+, GBR はユーザモードでも使うことができます。

Rm\_BANK オペランドは、SR レジスタの RB ビットで指定されます。RB ビットが 1 のとき、Rn オペランドとして R0\_BANK1 レジスタ ~ R7\_BANK1 レジスタおよび R8 レジスタ ~ R15 レジスタが使用され、Rm\_BANK オペランドとして R0\_BANK0 レジスタ ~ R7\_BANK0 レジスタが使用されます。RB ビットが 0 のとき、Rn オペランドとして R0\_BANK0 レジスタ ~ R7\_BANK0 レジスタおよび R8 レジスタ ~ R15 レジスタが使用され、Rm\_BANK オペランドとして R0\_BANK1 レジスタ ~ R7\_BANK1 レジスタが使用されます。

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。

### (2) 動作内容

```
LDCSR (long m)                /* LDC Rm, SR */
{
    SR=R[m] &0x0FFF0FFF;
    PC+=2;
}

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

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

LDCSSR (long m)              /* LDC Rm, SSR */
{
    SSR=R[m] &0x700003F3;
    PC+2;
}

LDCPSC (long m)              /*LDC Rm, SPC */
{
```

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

LDCRn_BANK(long m)          /*LDC Rm,Rn_BANK */
{                             /*n=0-7 */
    Rn_BANK=R [m] ;
    PC+=2;
}

LDCMSR(long m)              /* LDC.L @Rm+,SR */
{
    SR=Read_Long (R [m] ) &0x0FFF0FFF;
    R [m] +=4;
    PC+=2;
}

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

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

LDCMSSR(long m)             /*LDC.L @Rm+,SSR */
{
    SSR=Read_Long (R [m] ) &0x700003F3;
    R [m] +=4;
    PC+=2;
}

LDCMSPC(long m)             /*LDC.L @Rm+,SPC */
{
```

## 8. 各命令の説明

---

```
    SPC=Read_Long (R [m] ) ;
    R [m] +=4 ;
    PC+=2 ;
}
LDCMRn_BANK (Long m)          /*LDC.L @Rm+, Rn_BANK */
                               /*n=0-7 */
{
    Rn_BANK=Read_Long (R [m] ) ;
    R [m] +=4 ;
    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 ;
}

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;  
}
```

### (3) 使用例

```
LDC    R0,SR           ;実行前  R0=H'FFFFFFFF,SR=H'00000000  
                          ;実行後  SR=H'700003F3,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'12345678
```

## 8. 各命令の説明

---

### 8.2.28 LDRE Load effective address to RE 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
LDRE @(disp,PC)	disp × 2 + PC → RS	10001100dddddddd	3	

#### (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 にディスプレイ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 ビットディスプレイ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開始アドレスとは異なります。詳しくは、表 5.23 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。

#### (3) 動作内容

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

    if ((d&0x80)==0) disp=(0x000000FF & (long)d);
    else disp=(0xFFFFFFFF0 | (long)d);
    RE=PC+(disp<<1);
    PC+=2;
}
```

#### (4) 使用例

```
LDRS STA          ; set repeat start address to RS.
LDRE END          ; set repeat end address to RE.
SETRC #32        ; repeat 32 times from inst.A to inst.C.
inst.0           ;
STA: inst.A       ;
inst.B           ;
.....
END: inst.C       ;
inst.E           ;
.....
```

## 8.2.29 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	3	

### (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 にディスプレイ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 ビットディスプレイ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 5.23 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。

### (3) 動作内容

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

    if ((d&0x80)==0) disp=(0x000000FF & (long)d);
    else disp=(0xFFFFF00 | (long)d);
    RS=PC+(disp<<1);
    PC+=2;
}
```

### (4) 使用例

```
LDRS STA          ; set repeat start address to RS.
LDRE END          ; set repeat end address to RE.
SETRC #32         ; repeat 32 times from inst.A to inst.C.
inst.0            ;
STA: inst.A       ;
inst.B           ;
.....
END: inst.C      ;
inst.D          ;
.....
```

## 8. 各命令の説明

### 8.2.30 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 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+,MACH	(Rm)→MACH, Rm+4→Rm	0100mmmm00000110	1	
LDS.L @Rm+,MACL	(Rm)→MACL, Rm+4→Rm	0100mmmm00100110	1	
LDS.L @Rm+,PR	(Rm)→PR, Rm+4→Rm	0100mmmm00010110	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システムレジスタ MACH、MACL、PR、DSR、A0、X0、X1、Y0、Y1 に格納します。

#### (2) 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;
}

LDSRPR(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;
}

LDSMACL(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;
}

LSDSDR(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 */
{
```

## 8. 各命令の説明

---

```
    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;
}

LDSMSDR(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;
}

```

**(3) 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

## 8. 各命令の説明

### 8.2.31 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 レジスタを TLB (Translation Lookaside Buffer) に格納します。TLB は PTEH レジスタに書き込まれている仮想アドレスでインデクスされます。格納される TLB のウェイは MMU 制御レジスタ (MMUCR) の 2 ビットの RC フィールドで指定されます。

LDTLB 命令は特権命令であり、特権モードでだけ使われます。もしユーザーモードで使われた場合は不当命令例外が発生します。

【注】 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 (MMUCR.AT=0)、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、各製品のハードウェアマニュアルの MMU を参照してください)。本命令を例外ハンドラ中で使用する場合には、そのハンドラを終了させる RTE 命令の 2 命令以上に使用するようにしてください。

#### (2) 動作内容

```
LDTLB( ) /*LDTLB */
{
    TLB_tag=PTEH;
    TLB_data=PTEL;
    PC+=2;
}
```

#### (3) 使用例

```
MOV L @R0, R1 ;ページテーブルエントリ上位を R1 にロード
MOV L R1,@R2 ;R1 を PTEH にロード、R2 は PTEH のアドレス (H'FFFFFFF0)
MOV.L @R3, R4 ;ページテーブルエントリ下位を R4 にロード
MOV.L R4, @R5 ;R4 を PTEL にロード、R5 は PTEL のアドレス (H'FFFFFFF4)
LDTLB ;PTEH, PTEL レジスタを TLB にロード
```

## 8.2.32 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	2(~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ごとにそれぞれ、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'00007FFFFFFF(最大値) までに制限します。

#### (2) 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;
```

## 8. 各命令の説明

---

```
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 (fnLm<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 & 0x0007FFF;

    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;
}

```

## (3) 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.1     H'1234ABCD ;
       .data.1     H'5678EF01 ;
TBLN   .data.1     H'0123ABCD ;
       .data.1     H'4567DEF0 ;

```

### 8.2.33 MAC 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	2(~5)	

#### (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ごとにそれぞれ、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。オーバー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ーフローしたときは、H'80000000 (最小値) を、正の方向にオーバーフローしたときは 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+=tempn;
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

```

MOV      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  ;

```

## 8. 各命令の説明

### 8.2.34 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.WRm,@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.WRm,@-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.WRm,@(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) 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] |= 0xFFFFF00;
    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;
}
```

## 8. 各命令の説明

---

```
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]|=0xFFFFF00;
    if (n!=m) R[m]++;
    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]++;
    PC+=2;
}

MOVLP(long m, long n)     /* MOV.L @Rm+,Rn */
{
    R[n]=Read_Long(R[m]);
    if (n!=m) R[m]++;
    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]|=0xFFFFF00;
    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;
}
```

## 8. 各命令の説明

---

### (3) 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 R1=@H'10000004
MOV.W  @(R0,R2),R1    ;実行前 R2=H'00000004,R0=H'10000000
                          ;実行後 R1=@H'10000004
```

## 8.2.35 MOV MOVE immediate data : 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	1110nnnniiiiiii	1	
MOV.W@(disp,PC),Rn	(disp × 2+PC)→符号拡張→Rn	1001nnnnddddddd	1	
MOV.L@(disp,PC),Rn	(disp × 4+PC)→Rn	1101nnnnddddddd	1	

#### (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、PC にディスプレースメントを加えたアドレスに格納されたテーブル内のデータを参照します。

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+510バイトまでの範囲になります。PCは本命令の2命令後の先頭アドレスです。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+1020バイトまでの範囲になります。PCは本命令の2命令後の先頭アドレスですが、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

テーブルはモジュール後端あるいは無条件分岐命令の1命令後への配置が最適です。510バイト/1020バイト以内に無条件分岐命令がないなどの理由で最適配置が不可能なときは、BRA命令でテーブルを飛び越す対策が必要です。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。

#### (3) 動作内容

```

MOVI(long i, long n)      /* MOV #imm,Rn */
{
    if ((i&0x80)==0) R[n]=(0x000000FF & (long)i);
    else R[n]=(0xFFFFF00 | (long)i);
    PC+=2;
}

MOVWI(long d, long n) /* MOV.W @(disp,PC),Rn */
{
    long disp;
    disp=(0x000000FF & (long)d);
    R[n]=(long)Read_Word(PC+(disp<<1));
    if ((R[n]&0x8000)==0) R[n]&=0x0000FFFF;
    else R[n]|=0xFFFF0000;
    PC+=2;
}

```

## 8. 各命令の説明

---

```
MOVLI(long d, long n)/* MOV.L @(disp,PC),Rn */
{
    long disp;

    disp=(0x000000FF & (long)d);
    R[n]=Read_Long((PC&0xFFFFF0)+ (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@(H'08,PC)の意味
1004	ADD	#-1,R0	;
1006	TST	R0,R0	;MOV.W命令でアドレス計算に用いるPCの位置
1008	MOVT	R13	;
100A	BRA	NEXT	;遅延分岐命令
100C	MOV.L	@(4,PC),R3	;R3=H'12345678
100E	IMM	.data.w	H'9ABC
1010		.data.w	H'1234
1012	NEXT	JMP	@R3
1014		CMP/EQ	#0,R0
		.align	4
1018		.data.l	H'12345678



## 8.2.36 MOV MOVE peripheral data : データ転送命令

周辺モジュールデータの転送

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	

## (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。内蔵周辺モジュール領域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、レジスタが R0 固定になります。

GBR には、内蔵周辺モジュールのベースアドレスを設定します。

内蔵周辺モジュールのデータがバイトサイズるとき 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。+255 バイトまでの範囲が指定できます。ワードサイズるとき 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るとき 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。メモリオペランドに届かないときは GBR を汎用レジスタに転送した後、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

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。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 @(12,GBR),R0	MOV.B @(12,GBR),R0
AND #80,R0	ADD #20,R1
ADD #20,R1	AND #80,R0

## (3) 動作内容

```
MOVBLG(long d /* MOV.B @(disp,GBR),R0 */
{
    long disp;

    disp=(0x000000FF & (long)d);
    R[0]=(long)Read_Byte(GBR+disp);
    if ((R[0]&0x80)==0) R[0]&=0x000000FF;
```

## 8. 各命令の説明

---

```
        else R[0] |= 0xFFFFF00;
        PC+=2;
    }

MOVWLG(long d) /* MOV.W @(disp,GBR),R0 */
{
    long disp;

    disp=(0x000000FF & (long)d);
    R[0]=(long)Read_Word(GBR+(disp<<1));
    if ((R[0]&0x8000)==0) R[0]&=0x0000FFFF;
    else R[0] |= 0xFFFF0000;
    PC+=2;
}

MOVLG(long d) /* MOV.L @(disp,GBR),R0 */
{
    long disp;
    disp=(0x000000FF & (long)d);
    R[0]=Read_Long(GBR+(disp<<2));
    PC+=2;
}

MOVBSG(long d) /* MOV.B R0,@(disp,GBR) */
{
    long disp;
    disp=(0x000000FF & (long)d);
    Write_Byte(GBR+disp,R[0]);
    PC+=2;
}

MOVWSG(long d) /* MOV.W R0,@(disp,GBR) */
{
    long disp;
    disp=(0x000000FF & (long)d);
    Write_Word(GBR+(disp<<1),R[0]);
    PC+=2;
}

MOVLSG(long d) /* MOV.L R0,@(disp,GBR) */
{
    long disp;
    disp=(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'FFFF7F80
```

## 8.2.37 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)	10000000nnnnndddd	1	
MOV.W R0,@(disp,Rn)	R0→(disp×2+Rn)	10000001nnnnndddd	1	
MOV.L Rm,@(disp,Rn)	Rm→(disp×4+Rn)	0001nnnnmmmmndddd	1	
MOV.B @(disp,Rm),R0	(disp+Rm)→符号拡張→R0	10000100mmmmndddd	1	
MOV.W @(disp,Rm),R0	(disp×2+Rm)→符号拡張→R0	10000101mmmmndddd	1	
MOV.L @(disp,Rm),Rn	(disp×4+Rm)→Rn	0101nnnnmmmmndddd	1	

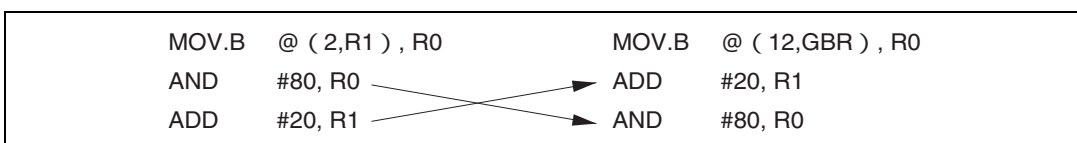
## (1) 説明

ソースオペランドをデスティネーションへ転送します。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るとき 4 ビットディスプレイ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。ワードサイズるとき 4 ビットディスプレイ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るとき 4 ビットディスプレイ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タが 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。



## (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]|=0xFFFFF00;
    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;
}
```

## 8. 各命令の説明

---

```
}  
  
MOVLL4(long m,long d,long n)  
    /* MOV.L @(disp,Rm),Rn */  
{  
    long disp;  
  
    disp=(0x0000000F & (long)d);  
    R[0]=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'3C,R1)      ;実行前 R0=H'FFFF7F80  
                               ;実行後@(R1+60)=H'FFFF7F80
```

### 8.2.38 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→R0	11000111dddddddd	1	

#### (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 倍しますので、オペランドとの相対距離は PC+1020 バイトまでの範囲になります。PC は本命令の 2 命令後の先頭アドレスですが、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

本命令が遅延分岐命令の直後に配置されているとき、PC は分岐先の “先頭アドレス+2” になります。

#### (3) 動作内容

```
MOVA(long d)      /* MOVA @(disp,PC),R0 */
{
    long disp;

    disp=(0x000000FF & (long)d);
    R[0]=(PC&0xFFFFF0)+(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          ;←MOVA 命令のアドレス計算時、PC の本来の位置
    .align 4
100C STR:.sdata "XYZP12"
    .....

2002 BRA TRGET          ;遅延分岐命令
2004 MOVA @(0,PC),R0    ;TRGET のアドレス+2→R0
2006 NOP                ;
```

## 8. 各命令の説明

---

### 8.2.39 MOV T MOVE T bit : 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
MOV T 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) 動作内容

```
MOV T(long n) /* MOV T Rn */  
{  
  R[n] = (0x00000001 & SR);  
  PC+=2;  
}
```

#### (3) 使用例

```
XOR      R2, R2      ;R2=0  
CMP/PZ   R2          ;T=1  
MOV T    R0          ;R0=1  
CLR T  
MOV T    R1          ;R1=0
```



## 8.2.40 MUL.L MULtipliy 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	$Rn \times Rm \rightarrow MACL$	0000nnnnmmmm0111	2(~5)	

#### (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。MACL の内容は変化しません。

#### (2) 動作内容

```
MULL(long m, long n) /* MUL.L Rm,Rn */
{
  MACL=R[n]*R[m];
  PC+=2;
}
```

#### (3) 使用例

```
MULL    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
```

## 8. 各命令の説明

---

### 8.2.41 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 MULS Rm,Rn	符号付きで $Rn \times Rm \rightarrow MACL$	0010nnnnmmmm1111	1(~3)	

#### (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) 動作内容

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

#### (3) 使用例

```
MULS 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
```

## 8.2.42 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 MULU Rm,Rn	符号なしで $Rn \times Rm \rightarrow MACL$	0010nnnnmmmm1110	1(~3)	

#### (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) 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;
}
```

#### (3) 使用例

```
MULU R0,R1 ;実行前 R0=H'00000002,R1=H'FFFFFFAA
;実行後 MACL=H'00015554
STS MACL,R0 ;演算結果を得る
```

## 8. 各命令の説明

---

### 8.2.43 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 の補数を取り、結果を Rn に格納します。すなわち 0 から Rm を減算し、結果を Rn に格納します。

#### (2) 動作内容

```
NEG(long m, long n) /* NEG Rm,Rn */  
{  
  R[n]=0-R[m];  
  PC+=2;  
}
```

#### (3) 使用例

```
NEG R0,R1           ;実行前      R0=H'00000001  
                    ;実行後      R1=H'FFFFFFF
```

## 8.2.44 NEGC NEGate with Carry : 算術演算命令

ポロ-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, ポロ-→T	0110nnnnmmmm1010	1	ポロ-

## (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ポロ-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) 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;
}
```

## (3) 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'FFFFFFF,T=1
NEGC R0,R0    ;実行前 R0=H'00000000,T=1
              ;実行後 R0=H'FFFFFFF,T=1
```

## 8. 各命令の説明

---

### 8.2.45 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) 動作内容

```
NOP( ) /* NOP */  
{  
    PC+=2;  
}
```

#### (3) 使用例

NOP ;1 サイクルの時間が過ぎます。

## 8.2.46 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 の補数を取り、結果を Rn に格納します。すなわち Rm のビットを反転して Rn に格納します。

#### (2) 動作内容

```
NOT(long m, long n) /* NOT Rm,Rn */
{
  R[n] = ~R[m];
  PC+=2;
}
```

#### (3) 使用例

```
NOT R0,R1 ;実行前 R0=H'AAAAAAAA
          ;実行後 R1=H'55555555
```

## 8. 各命令の説明

---

### 8.2.47 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) 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;
}
```



## (3) 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
```

## 8. 各命令の説明

### 8.2.48 PREF PREFetch data to the cache : システム制御命令

#### 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 @Rm	(Rm & 0xfffff0)→Cache (Rm & 0xfffff0+4)→Cache (Rm & 0xfffff0+8)→Cache (Rm & 0xfffff0+C)→Cache	0000mmmm10000011	1-3	

#### (1) 説明

ソフトウェア用のプリフェッチ命令でデータをキャッシュに書き込みます。

Rn レジスタでアドレスを指定したデータを含むキャッシュ1ライン分の16バイトデータがキャッシュに書き込まれます。

この命令でアドレスに関するエラーは発生しません。エラーの場合には、この命令はNOP（無操作）命令として取り扱われます。

ディステネーションは内蔵キャッシュです。そのためこの命令は実際にはNOP命令と同様に、レジスタの値や処理状態を変更させることはありません。

#### (2) 動作内容

```
PREF(long m) /* PREF */  
{  
    PC+=2;  
}
```

#### (3) 使用例

```
MOV.L    SOFT_PF, R1    ;SOFT_PF のアドレス→R1  
PREF    @R1            ;SOFT_PF のデータを内蔵キャッシュへロード  
  
        .align        4  
SOFT_PF: .data.1      H'12345678  
        .data.1      H'9ABCDEF0  
        .data.1      H'AAAA5555  
        .data.1      H'5555AAAA
```

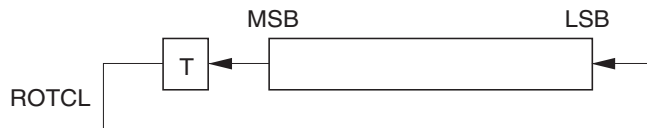
## 8.2.49 ROTCL ROTate with Carry Left : シフト命令

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	$T \leftarrow Rn \leftarrow T$	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) 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]&=0xFFFFF0;
    if (temp==1) T=1;
    else T=0;
    PC+=2;
}

```

## (3) 使用例

```

ROTCL R0 ;実行前 R0=H'80000000,T=0
          ;実行後 R0=H'00000000,T=1

```

## 8. 各命令の説明

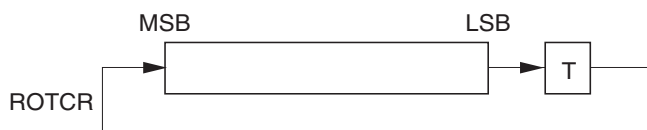
### 8.2.50 ROTCR ROTate with Carry Right : シフト命令

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) 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;
}
```

#### (3) 使用例

```
ROTCR R0 ;実行前 R0=H'00000001,T=1
          ;実行後 R0=H'80000000,T=1
```

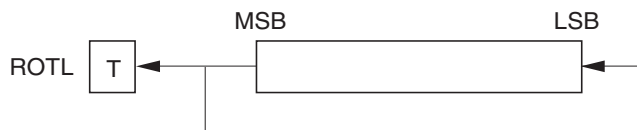
## 8.2.51 ROTL ROTate Left : シフト命令

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	$T \leftarrow Rn \leftarrow \text{MSB}$	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) 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]&=0xFFFFF0;
    PC+=2;
}
```

## (3) 使用例

```
ROTL R0 ;実行前 R0=H'80000000,T=0
        ;実行後 R0=H'00000001,T=1
```

## 8. 各命令の説明

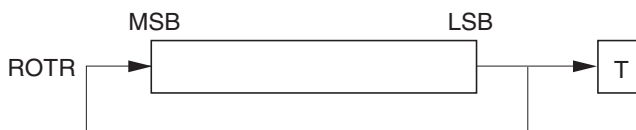
### 8.2.52 ROTR ROTate Right : シフト命令

#### 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) 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;
}
```

#### (3) 使用例

```
ROTR R0 ;実行前 R0=H'00000001,T=0
        ;実行後 R0=H'80000000,T=1
```

## 8.2.53 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。

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。

本命令の直後の遅延スロットで実行される命令は、本命令で復帰した SR が使用されます。

本命令の直後の遅延スロットで実行される命令で例外を発生させないでください。また、レジスタ SR の MD、BL ビットを操作する命令と、その次の命令は MMU がディスエーブル状態か固定物理アドレス空間 (P1、P2 空間) で使用してください。

## (3) 動作内容

```
RTE( ) /* RTE */
{
    unsigned long temp;
    temp=PC;
    PC=SPC;
    SR=SSR;
    Delay_Slot(temp+2);
}
```

## 8. 各命令の説明

---

### (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 ADD を実行します。

**【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。



## 8.2.54 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	00000000000001011	2	

## (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、BSRF 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。

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。

本命令で使用する PR は、本命令より前で設定しなければなりません。

直後の命令で設定した PR は、本命令で使用されません。

## (3) 動作内容

```
RTS ( ) /* RTS */
{
    unsigned long temp;

    temp=PC;
    PC=PR+4;
    Delay_Slot(temp+2);
}
```

## 8. 各命令の説明

---

### (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                  ;分岐に先立ち NOP 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)
.....
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。
```

【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。

## 8.2.55 SETRC SET reপরত 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 の下位 12 ビット→RC(SR のビット 27~16)、繰り返し制御フラグ→RF1、RF0	0100mmmm00010100	3	
SETRC #imm	imm→RC (SR のビット 23~16)、繰り返し制御フラグ→RF1、RF0	10000010iiiiiii	3	

## (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、「5.12 DSP 繰り返し (ループ) 制御」を参照してください。

## (2) 動作内容

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

    temp=(R[m] & 0x0000FFF)<<16;
    SR&=0xF00FFFF;
    SR|=temp;
    RF1=Repeat_Control_Flag1;
    RF0=Repeat_Control_Flag0;
    PC+=2;
}

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

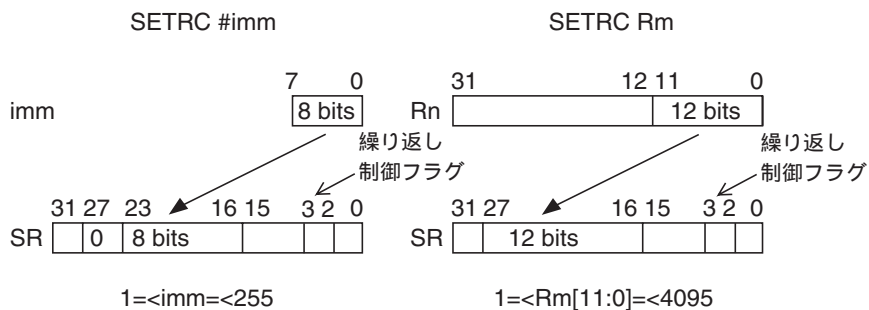
    temp=((long)i & 0x000000FF)<<16;
    SR&=0xF00FFFF;
    SR|=temp;
    RF1=Repeat_Control_Flag1;
```

## 8. 各命令の説明

```

RF0=Repeat_Control_Flag0;
PC+=2;
}

```



### (3) 使用例

```

LDRS STA ; set repeat start address to RS.
LDRE END ; set repeat end address to RE.
SETRC #32 ; repeat 32 times from inst.A to inst.C.
inst.0 ;
STA: inst.A ;
inst.B ;
.....
END: inst.C ;
inst.D ;
.....

```

## 8.2.56 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

#### (1) 説明

Sビットを1にセットします。

#### (2) 動作内容

```
SETS( ) /* SETS */
{
    S=1;
    PC+=2;
}
```

#### (3) 使用例

```
SETS          ;実行前 S=0
              ;実行後 S=1
```

## 8. 各命令の説明

---

### 8.2.57 SETT SET T bit : 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1にセットします。

#### (2) 動作内容

```
SETT( ) /* SETT */  
{  
    T=1;  
    PC+=2;  
}
```

#### (3) 使用例

```
SETT      ;実行前 T=0  
          ;実行後 T=1
```

## 8.2.58 SHAD SHift Arithmetic Dynamically : シフト命令

### ダイナミック算術シフト

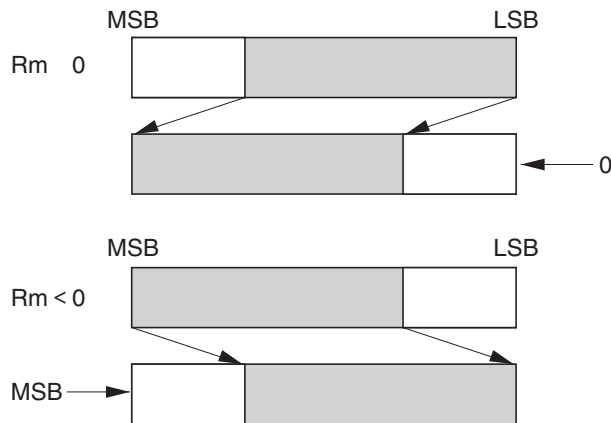
書式	動作概略	命令コード	実行 ステート	Tビット
SHAD Rm, Rn	Rm ≥ 0 のとき、 $Rn \ll Rm \rightarrow Rn$ Rm < 0 のとき、 $Rn \gg Rm \rightarrow$ [MSB→Rn]	0100nnnnmmmm1100	1	

#### (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。
- 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) 動作内容

```
SHAD(long m,n) /*SHAD Rm,Rn */
{
    long cont,sgn;
    sgn=R[m]&0x80000000;
    cnt=R[m]&0x0000001F;
    if (sgn==0) R[n]<<=cnt;
    else R[n]=(long)R[n]>>((-cnt+1)&0x1F);/* shift arithmetic right*/
    PC+=2;
}
```

## 8. 各命令の説明

---

### (3) 使用例

SHAD R1,R2

;実行前 R1=H'FFFFFFEC,

R2=H'80180000

;実行後 R1=H'FFFFFFEC,

R2=H'FFFFFF801

SHAD R3,R4

;実行前 R3=H'00000014,

R4=H'FFFFFF801

;実行後 R3=H'00000014,

R4=H'80100000



## 8.2.59 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	$T \leftarrow Rn \leftarrow 0$	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) 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;
}
```

#### (3) 使用例

```
SHAL R0 ;実行前 R0=H'80000001,T=0
        ;実行後 R0=H'00000002,T=1
```

## 8. 各命令の説明

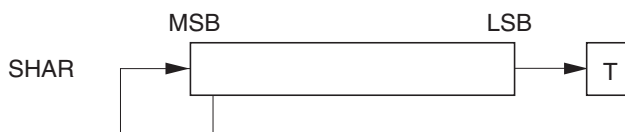
### 8.2.60 SHAR SHift Arithmetic Right : シフト命令

#### 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) 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;
}
```

#### (3) 使用例

```
SHAR R0 ;実行前 R0=H'80000001,T=0
        ;実行後 R0=H'C0000000,T=1
```

## 8.2.61 SHLD SHift Logical Dynamically : シフト命令

### ダイナミック論理シフト

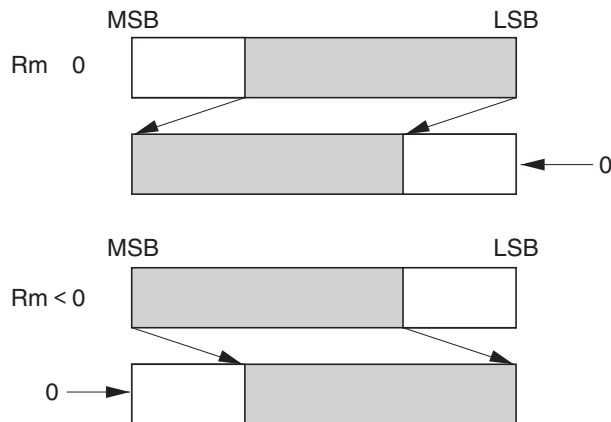
書式	動作概略	命令コード	実行 ステート	Tビット
SHAD Rm, Rn	Rm = 0 のとき、 $Rn \ll Rm \rightarrow Rn$ Rm < 0 のとき、 $Rn \gg Rm \rightarrow [0 \rightarrow Rn]$	0100nnnnmmmm1101	1	

#### (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。Tビットは最後にシフトされたビットです。

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には上位に 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) 動作内容

```
SHLD(long m, n) /*SHLD Rm, Rn */
{
    long cont, sgn;
    sgn=R[m] &0x80000000;
    cnt=R[m] &0x0000001F;
    if (sgn==0)    R[n] <<=cnt;
    else R[n]=(unsigned long)R[n] >> ((~cnt+1) &0x1F);
    PC+=2;
}
```

## 8. 各命令の説明

---

### (3) 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'FFFFFF801
	;実行後	R3=H'00000014, R4=H'80100000

## 8.2.62 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	$T \leftarrow Rn \leftarrow 0$	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) 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;
}
```

## (3) 使用例

```
SHLL R0 ;実行前 R0=H'80000001,T=0
        ;実行後 R0=H'00000002,T=1
```

## 8. 各命令の説明

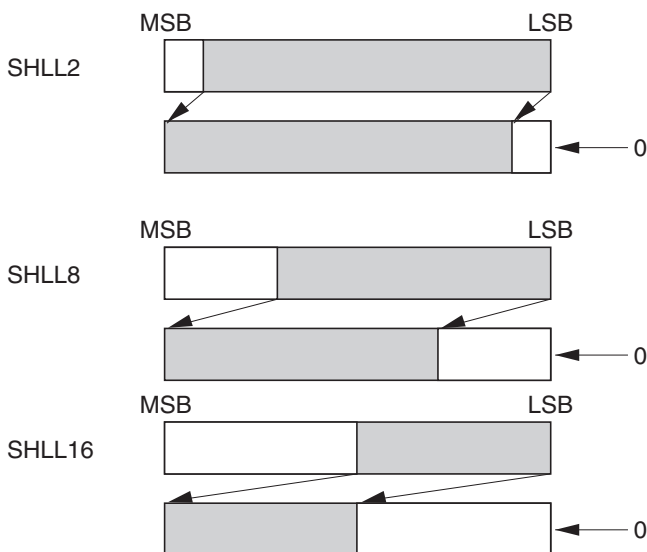
### 8.2.63 SHLLn n bits SHift Logical Left : シフト命令

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	$Rn \ll 2 \rightarrow Rn$	0100nnnn00001000	1	
SHLL8 Rn	$Rn \ll 8 \rightarrow Rn$	0100nnnn00011000	1	
SHLL16 Rn	$Rn \ll 16 \rightarrow Rn$	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) 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;  
    }
```

### (3) 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
```

## 8. 各命令の説明

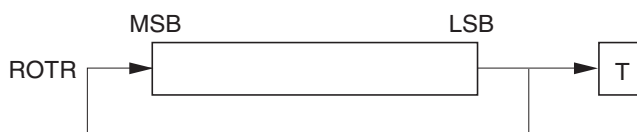
### 8.2.64 SHLR SHift Logical Right : シフト命令

#### 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) 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;
}
```

#### (3) 使用例

```
SHLR R0 ;実行前 R0=H'80000001,T=0
        ;実行後 R0=H'40000000,T=1
```



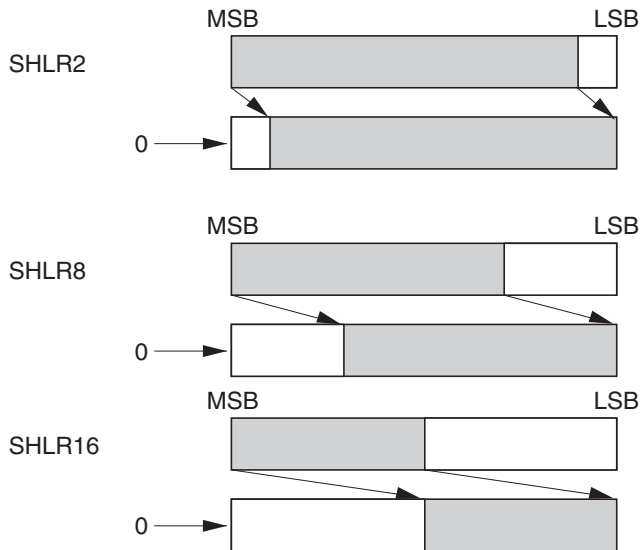
## 8.2.65 SHLRn n bits SHift Logical Right : シフト命令

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	$Rn \gg 2 \rightarrow Rn$	0100nnnn00001001	1	
SHLR8 Rn	$Rn \gg 8 \rightarrow Rn$	0100nnnn00011001	1	
SHLR16 Rn	$Rn \gg 16 \rightarrow Rn$	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) 動作内容

```
SHLR2(long n)                                /* SHLR2 Rn */
{
    R[n] >>= 2;
    R[n] &= 0x3FFFFFFF;
    PC+=2;
}
```

```
SHLR8(long n)                                /* SHLR8 Rn */
{
```

## 8. 各命令の説明

---

```
        R[n] >>=8;
        R[n] &=0x00FFFFFF;
        PC+=2;
    }

SHLR16 (long n)                               /* SHLR16 Rn */
{
    R[n] >>=16;
    R[n] &=0x0000FFFF;
    PC+=2;
}
```

### (3) 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
```

## 8.2.66 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	0000000000011011	4	

## (1) 説明

CPU を低消費電力モードにします。

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、低消費電力モードから抜けます。

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

実行ステートの 4 は、スリープモードに遷移するまでのステート数です。

## (3) 動作内容

```
SLEEP ( )    /* SLEEP */
{
    PC-=2;
    wait_for_exception;
}
```

## (4) 使用例

```
SLEEP      ;低消費電力モードへの遷移
```

## 8. 各命令の説明

### 8.2.67 STC Store Control register : システム制御命令

コントロールレジスタからのストア

(特権命令)

書式	動作概略	命令コード	実行 ステート	Tビット
STC SR,Rn	SR→Rn	0000nnnn00000010	1	
STC 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 MOD,Rn	MOD→Rn	0000nnnn01010010	1	
STC RE,Rn	RE→Rn	0000nnnn01110010	1	
STC RS,Rn	RS→Rn	0000nnnn01100010	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 SR,@-Rn	Rn-4→Rn, SR→(Rn)	0100nnnn00000011	1/2*	
STC.L GBR,@-Rn	Rn-4→Rn, GBR→(Rn)	0100nnnn00010011	1/2*	
STC.L VBR,@-Rn	Rn-4→Rn, VBR→(Rn)	0100nnnn00100011	1/2*	
STC.L SSR,@-Rn	Rn-4→Rn, SSR→(Rn)	0100nnnn00110011	1/2*	
STC.L SPC,@-Rn	Rn-4→Rn, SPC→(Rn)	0100nnnn01000011	1/2*	
STC.L MOD,@-Rn	Rn - 4→Rn, MOD→(Rn)	0100nnnn01010011	2	
STC.L RE,@-Rn	Rn - 4→Rn, RE→(Rn)	0100nnnn01110011	2	
STC.L RS,@-Rn	Rn - 4→Rn, RS→(Rn)	0100nnnn01100011	2	
STC.L R0_BANK,@-Rn	Rn-4→Rn, R0_BANK→(Rn)	0100nnnn10000011	2	
STC.L R1_BANK,@-Rn	Rn-4→Rn, R1_BANK→(Rn)	0100nnnn10010011	2	
STC.L R2_BANK,@-Rn	Rn-4→Rn, R2_BANK→(Rn)	0100nnnn10100011	2	
STC.L R3_BANK,@-Rn	Rn-4→Rn, R3_BANK→(Rn)	0100nnnn10110011	2	
STC.L R4_BANK,@-Rn	Rn-4→Rn, R4_BANK→(Rn)	0100nnnn11000011	2	
STC.L R5_BANK,@-Rn	Rn-4→Rn, R5_BANK→(Rn)	0100nnnn11010011	2	
STC.L R6_BANK,@-Rn	Rn-4→Rn, R6_BANK→(Rn)	0100nnnn11100011	2	
STC.L R7_BANK,@-Rn	Rn-4→Rn, R7_BANK→(Rn)	0100nnnn11110011	2	

【注】 \* SH3-DSP では2ステートになります。

## (1) 説明

コントロールレジスタ SR、GBR、VBR、SSR、SPC、MOD、RE、RS または R0~R7\_BANK のデータをディスティネーションに格納します。STC GBR, Rn と STC.L GBR, @-Rn を除く STC と STC.L 命令は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。ただし、STC GBR, Rn と STC.L GBR, @-Rn はユーザモードでも使うことができます。

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、Rn オペランドとして R0\_BANK1 レジスタ~R7\_BANK1 レジスタおよび R8 レジスタ~R15 レジスタが使用され、Rm\_BANK オペランドとして R0\_BANK0 レジスタ~R7\_BANK0 レジスタが使用されます。RB ビットが 0 のとき、Rn オペランドとして R0\_BANK0 レジスタ~R7\_BANK0 レジスタおよび R8 レジスタ~R15 レジスタが使用され、Rm\_BANK オペランドとして R0\_BANK1 レジスタ~R7\_BANK1 レジスタが使用されます。

## (2) 動作内容

```

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

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

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

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

STCSPC(long n)              /* STC SPC, Rn */
{
    R[n]=SPC;

```

## 8. 各命令の説明

---

```
        PC+=2;
    }

STCRm_BANK(long n)          /* STC Rm_BANK,Rn */
                              /* n=0-7 */
{
    R[n]=Rm_BANK;
    PC+=2;
}

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

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

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

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

STCMSPC(long n)             /* STC.L SPC,@-Rn */
```

```
{
    R[n] -=4;
    Write_Long(R[n], SPC);
    PC+=2;
}

STCMRm_BANK(long n)          /* STC.L Rm_BANK, @-Rn */
                              /* n=0-7 */
{
    R[n] -=4;
    Write_Long(R[n], Rm_BANK);
    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;
}

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

STCMRE(long n)               /* STC.L RE, @-Rn */
{
```

## 8. 各命令の説明

---

```
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;
}
```

### (3) 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



## 8.2.68 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 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 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	
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

システムレジスタ MACH、MACL、PR、DSR、A0、X0、X1、Y0、Y1 をデスティネーションに格納します。

#### (2) 注意

システムレジスタが MACH のときは、MACH の 32 ビットをそのまま格納します。

#### (3) 動作内容

```
STSMACH(long n) /* STS MACH,Rn */
```

```
{
```

```
    R[n]=MACH;
```

```
    PC+=2;
```

```
}
```

```
STSMACL(long n) /* STS MACL,Rn */
```

```
{
```

```
    R[n]=MACL;
```

```
    PC+=2;
```

```
}
```

## 8. 各命令の説明

---

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

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

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

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;
}

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

STSMX0(long n)         /* STS.L X0,@-Rn */
{
```

## 8. 各命令の説明

---

```
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
```

## 8.2.69 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) 動作内容

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

#### (3) 使用例

```
SUB    R0,R1                ;実行前 R0=H'00000001,R1=H'80000000
                                ;実行後 R1=H'7FFFFFFF
```

## 8. 各命令の説明

---

### 8.2.70 SUBC SUBtract with Carry : 算術演算命令

ボロ付き 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→T	0011nnnnmmmm1010	1	ボロ

#### (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) 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;
}
```

#### (3) 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
```

## 8.2.71 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 1 にセットします。

#### (2) 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;
}

```

#### (3) 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'7FFFFFFFE
                   ;実行後  R3=H'80000000,T=1

```

## 8. 各命令の説明

### 8.2.72 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	0110nnnnnnmmmm1000	1	
SWAP.W Rm,Rn	交換→Rn Rm→上下ワード交換→Rn	0110nnnnnnmmmm1001	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) 動作内容

```
SWAPB(long m, long n)                                /* SWAP.B Rm,Rn */
{
    unsigned long temp0,temp1;

    temp0=R[m]&0xFFFF000;
    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;
}
```



## (3) 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
```

## 8. 各命令の説明

---

### 8.2.73 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 1→MSB of (Rn)	0100nnnn00011011	3/4*	テスト結果

【注】 \* SH3-DSP では4ステートになります。

#### (1) 説明

汎用レジスタ Rn の内容をアドレスとし、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。

【注】 この命令の Rn の内容をアドレスとするメモリ領域は、ノンキャッシュブル領域としてください。

#### (2) 動作内容

```
TAS(long n)                                     /* TAS.B @Rn */
{
    long temp;

    temp=(long)Read_Byte(R[n]);                 /* Bus Lock enable */
    if (temp==0) T=1;
    else T=0;
    temp|=0x00000080;
    Write_Byte(R[n],temp);                       /* Bus Lock disable */
    PC+=2;
}
```

#### (3) 使用例

```
LOOP
TAS.B    @R7                ;R7=1000
BF      LOOP                ;1000番地がゼロになるまでループします。
```

## 8.2.74 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→TRA,PC→SPC,SR→SSR, 1→SR.MD/BL/RB, 0x160→EXPEVT, VBR+H'00000100→PC	11000011iiiiiii	6/8*	

【注】 \* SH3-DSP では 8 ステートになります。

## (1) 説明

トラップ例外処理を開始します。PC と SR の値が、SPC と SSR 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。プログラムはアドレス (VBR+H'00000100) に分岐します。

TRAPA 命令と RTE 命令と組み合わせてシステムコールに使います。

## (2) 注意

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せん。

## (3) 動作内容

```
TRAPA(long i) /* TRAPA #imm */
{
    long imm;

    imm=(0x000000FF & i);
    TRA=imm<<2;
    SSR=SR;
    SPC=PC;
    SR.MD=1;
    SR.BL=1;
    SR.RB=1;
    EXPEVT=0x00000160;
    PC=VBR+H'00000100;
}
```

## 8. 各命令の説明

### 8.2.75 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 1 にセットします。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) 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;  
}
```

### (3) 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
```

## 8.2.76 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	$Rn \wedge Rm \rightarrow Rn$	0010nnnnmmmm1010	1	
XOR #imm,R0	$R0 \wedge imm \rightarrow R0$	11001010iiiiiii	1	
XOR.B #imm,@(R0,GBR)	$(R0+GBR) \wedge imm \rightarrow (R0+GBR)$	11001110iiiiiii	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) 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) */
{
    long temp;

    temp=(long)Read_Byte(GBR+R[0]);
    temp^=(0x000000FF & (long)i);
    Write_Byte(GBR+R[0],temp);
    PC+=2;
}
```

## (3) 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
```

## 8. 各命令の説明

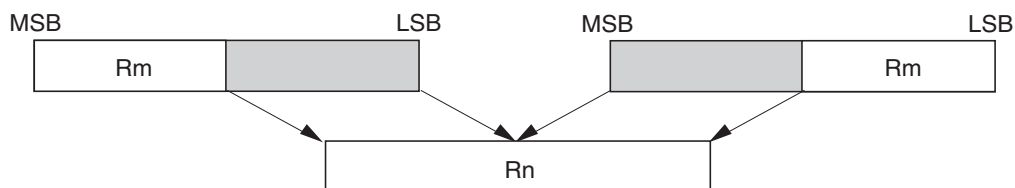
### 8.2.77 XTRCT eXTRaCT : データ転送命令

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	0010nnnnnnmmmm1101	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) 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;
}
```

#### (3) 使用例

```
XTRCT R0,R1                                           ;実行前 R0=H'01234567,R1=H'89ABCDEF
                                           ;実行後 R1=H'456789AB
```



## 8.3 浮動小数点命令とFPUに関連するCPU命令 (SH-3Eのみ)

FPU演算の動作内容説明に使用されている関数の内容は以下のとおりです。

```
long FPSCR;
int T;

int load_long(long *adress, *data)
{
    /* This function is defined in CPU part */
}
int store_long(long *adress, *data)
{
    /* This function is defined in CPU part */
}
int sign_of(long *src)
{
    return(*src >> 31);
}
int data_type_of(long *src)
{
    float abs;

    abs = *src & 0x7fffffff;
    if(abs < 0x00800000) {
        if(sign_of (src) == 0) return(PZERO);
        else return(NZERO);
    }
    else if((0x00800000 <= abs) && (abs < 0x7f800000))
        return(NORM);
    else if(0x7f800000 == abs) {
        if(sign_of (src) == 0) return(PINF);
        else return(NINF);
    }
    else if(0x00400000 & abs) return(sNaN);
    else return(qNaN);
}
clear_cause_VZ(){ FPSCR &= (~CAUSE_V & ~CAUSE_Z); }
```

## 8. 各命令の説明

---

```
set_V(){ FPSCR |= (CAUSE_V | FLAG_V); }
set_Z(){ FPSCR |= (CAUSE_Z | FLAG_Z); }

invalid(float *dest)
{
    set_V();
    if((FPSCR & ENABLE_V) == 0) qnan(dest);
}
dz(float *dest, int sign)
{
    set_Z();
    if((FPSCR & ENABLE_Z) == 0) inf (dest,sign);
}
zero(float *dest, int sign)
{
    if(sign == 0)                *dest = 0x00000000;
    else                        *dest = 0x80000000;
}
int(float *dest, int sign)
{
    if(sign == 0)                *dest = 0x7f800000;
    else                        *dest = 0xff800000;
}
qnan(float *dest)
{
    *dest = 0x7fbfffff;
}
```

### 8.3.1 FABS Floating point ABSolute value : 浮動小数点命令 浮動小数点数絶対値

書式	動作概略	命令コード	レイテンシー	ピッチ	Tビット
FABS FRn	FRn  → FRn	1111nnnn01011101	2	1	-

#### (1) 説明

浮動小数点レジスタ FRn の内容の（浮動小数点数としての）算術的な絶対値をとります。演算結果は、FRn に書き込まれます。

#### (2) 動作内容

```
FABS(float *FRn) /* FABS FRn */
{
  clear_cause_VZ();
  case(data_type_of(FRn))
  {
    NORM:      if(sign_of(FRn) == 0) *FRn = *FRn;
               else *FRn = -*FRn;
               break;

    PZERO     :
    NZERO     : zero(FRn, 0);
               break;

    PINF      :
    NINF      : inf(FRn, 0);
               break;

    qnan      : qnan(FRn);
               break;

    sNaN      : invalid(FRn);
               break;
  }
  pc += 2;
}
```

#### FABS 特殊ケース

FRn	NORM	+0	-0	+INF	-INF	qNaN	sNaN
FABS(FRn)	ABS	+0	+0	+INF	+INF	qNaN	Invalid

【注】 非正規化数の値は、ゼロとして扱われます。

#### (3) 例外

無効演算 ( Invalid Operation )

#### (4) 使用例

```
FABS      FR2          ;浮動小数点絶対値
           ;実行前 FR2=H' C0800000/*10 進数の-4*/
           ;実行後 FR2=H' 40800000/*10 進数の 4*/
```

## 8.3.2 FADD Floating point ADD : 浮動小数点命令

## 浮動小数点数加算

書式	動作概略	命令コード	レイテンシー	ピッチ	Tビット
FADD FRm,FRn	FRn+FRm→FRn	1111nnnnmmmm0000	2	1	-

## (1) 説明

浮動小数点レジスタ FRm と FRn の内容を、(浮動小数点数として)算術的に加算します。演算結果は、FRn に書き込まれます。

## (2) 動作内容

```

FADD ( float *FRm,FRn )                               /* FADD FRm,FRn */
{
    clear_cause_VZ();
    if((data_type_of(FRm) == sNaN)                    ||
        (data_type_of(FRn) == sNaN))                  invalid(FRn);
    else if((data_type_of(FRm) == qNaN)                ||
        (data_type_of(FRn) == qNaN))                  qnan(FRn);
    else case(data_type_of(FRm))                       {
NORM:
        case(data_type_of(FRn))                         {
            PINF      :          inf (FRn,0);           break;
            NINF      :          inf (FRn,1);           break;
            default   :          *FRn = *FRn + *FRm;    break;
        }
    }
PZERO:
        case(data_type_of(FRn))                         {
            NORM      :          *FRn = *FRn + *FRm;    break;
            PZERO     :
            NZERO     :          zero (FRn,0);          break;
            PINF      :          inf (FRn,0);           break;
            NINF      :          inf (FRn,1);           break;
        }
    }
NZERO:
        case(data_type_of(FRn)) {
            NORM      :          *FRn = *FRn + *FRm;    break;
            PZERO     :          zero (FRn,0);          break;
            NZERO     :          zero (FRn,1);          break;
            PINF      :          inf (FRn,0);           break;
        }
    }
}

```

```

    NINF      :          inf (FRn, 1);                break;
  }
  PINF:
    case (data_type_of (FRn))                       {
      NINF      :          invalid (FRn);            break;
      default   :          inf (FRn, 0);             break;
    }
    NINF:
      case (data_type_of (FRn))                     {
        PINF      :          invalid (FRn);          break;
        default   :          inf (FRn, 1);           break;
      }
  }
  pc += 2;
}

```

FADD 特殊ケース

FRm	FRn						
	NORM	+0	-0	+INF	-INF	qNaN	sNaN
NORM	ADD				-INF	qNaN	Invalid
+0	+0		+INF		Invalid		
-0	-0						
+INF				Invalid	-INF		
-INF	-INF			Invalid	-INF		
qNaN							
sNaN							

【注】 非正規化数の値は、ゼロとして扱われます。

### (3) 例外

無効演算 (Invalid Operation)

## 8. 各命令の説明

---

### (4) 使用例

```
FADD    FR2, FR3    ;浮動小数点数加算
;実行前：    FR2=H'40400000/*10 進数の3*/
;            FR3=H'3F800000/*10 進数の1*/
;実行後：    FR2=H'40400000
;            FR3=H'40800000/*10 進数の4*/

FADD    FR5, FR4    ;
;実行前：    FR5=H'40400000/*10 進数の3*/
;            FR4=H'C0000000/*10 進数の-2*/
;実行後：    FR5=H'40400000
;            FR4=H'3F800000/*10 進数の1*/
```

### 8.3.3 FCMP Floating point Compare : 浮動小数点命令

#### 浮動小数点数比較

書式	動作概略	命令コード	レイテンシー	ピッチ	Tビット
FCMP/EQ FRm,FRn	(FRn==FRm)? 1:0→T	1111nnnnmmmm0100	2	1	比較結果
FCMP/GT FRm,FRn	(FRn>FRm)? 1:0→T	1111nnnnmmmm0101	2	1	比較結果

#### (1) 説明

浮動小数点レジスタ FRm と FRn の内容を、(浮動小数点数として)算術的に比較します。演算結果(真偽)は、Tビットに書き込まれます。

#### (2) 動作内容

```

FCMP_EQ(float *FRm,FRn) /* FCMP/EQ FRm,FRn */
{
    clear_cause_VZ();
    if (fcmp_chk(FRm,FRn) == INVALID) {fcmp_invalid(0); }
    else if (fcmp_chk(FRm,FRn) == EQ)          T = 1;
    else                                       T = 0;
    pc += 2;
}

FCMP_GT(float *FRm,FRn) /* FCMP/GT FRm,FRn */
{
    clear_cause_VZ();
    if (fcmp_chk(FRm,FRn) == INVALID) || {fcmp_chk (FRm, FRn) == UO}){
        fcmp_invalid(0); }
    else if (fcmp_chk(FRm,FRn) == GT)          T = 1;
    else                                       T = 0;
    pc += 2;
}

fcmp_chk(float *FRm,*FRn)
{
    if((data_type_of(FRm) == sNaN) ||
        (data_type_of(FRn) == sNaN))          return(INVALID);
    else if((data_type_of(FRm) == qNaN) ||
        (data_type_of(FRn) == qNaN))          return(UO);
    else case(data_type_of(FRm)) {
        NORM :case(data_type_of(FRn)) {
            PINF :return(GT);          break;
            NINF :return(NOTGT); break;
        }
    }
}

```

## 8. 各命令の説明

---

```

                                default          :                break;
    }
    PZERO                        :                break;

    NZERO                        :case(data_type_of(FRn))      {
    PZERO                        :                break;
    NZERO                        :return(EQ);                break;
    PINF                          :return(GT);                break;
    NINF                          :return(NOTGT); break;
    default                        :                break;
    }
    PINF                          :case(data_type_of(FRn))      {
    PINF                          :return(EQ)                break;
    default                        :return(NOTGT); break;
    }
    NINF                          :case(data_type_of(FRn))      {
    NINF                          :return(EQ);                break;
    default                        :return(GT);                break;
    }
    }
    if(*FRn == *FRm)              return(EQ);
    else if(*FRn > *FRm)         return(GT);
    else                          return(NOTGT);
}
fcmp_invalid(int cmp_flag)
{
    set_V();
    if((FPSCR & ENABLE_V) == 0) T = cmp_flag;
}

```



FCMP 特殊ケース

FRm	FRn							
	NORM	+0	-0	+INF	-INF	qNaN	sNaN	
NORM	CMP	EQ		GT	!GT	UO	Invalid	
+0								
-0								
+INF	!GT			EQ		UO	Invalid	
-INF	GT				EQ			
qNaN							UO	Invalid
sNaN								

【注】 \*1 FCMP/EQ のときは、UO。FCMP/GT のときは invalid。

\*2 非正規化数の値は、ゼロとして扱われます。

### (3) 例外

#### 無効演算 ( Invalid Operation )

【注】 IEEE 規格では 4 とおりの互いに独立の関係にある比較を定義していますが、SH-3E は FCMP/EQ と FCMP/GT のみをサポートしています。しかしながら、BT/BF 命令とこれら 2 種類の FCMP 命令を組み合わせることで、すべての比較条件をサポートすることができます。

(FRm = FRn) fcmp/eq FRm, FRn ; bt

(FRm != FRn) fcmp/eq FRm, FRn ; bf

(FRm < FRn) fcmp/gt FRm, FRn ; bt

(FRm <= FRn) fcmp/gt FRn, FRm ; bt

(FRm > FRn) fcmp/gt FRn, FRm ; bt

(FRm >= FRn) fcmp/gt FRm, FRn ; bf

Unorder FRm, FRn fcmp/eq FRm, FRm ; bf

### (4) 使用例

FCMP/EQ:

FLDI1 FR6 ;FR6=H'3F800000/\*10 進数の 1\*/

FLDI1 FR7 ;FR7=H'3F800000

CLRT ;T ビット=0

FCMP/EQ FR6, FR7 ;浮動小数点比較 Equal

BF TRGET\_F ;分岐しない (T=1)

NOP

BT/S TRGET\_T ;分岐する

FADD FR6, FR7 ;遅延スロット、FR7=H'40000000/\*10 進数の 2\*/

NOP

TRGET\_F FCMP/EQ FR6, FR7

BT/S TRGET\_T ;分岐しない (T=0)

## 8. 各命令の説明

---

```
FLDI1          FR7          ;遅延スロット
TRGET_T FCMP/EQ  FR6,FR7    ;T ビット = 0
BF TRGET_F      ;初回のみ分岐する。
NOP            ;FR6=FR7=H'3F800000/*10 進数の 1*/
.END

FCMP/GT:
FLDI1          FR2          ;FR2=H'3F800000/*10 進数の 1*/
FLDI1          FR7
FADD           FR2,FR7    ;FR7=H'40000000/*10 進数の 2*/
CLRT          ;T ビット = 0
FCMP/GT       FR2,FR7    ;浮動小数点比較 Greater Than
BT/S          TRGET_T    ;分岐する (T=1)
FLDI1          FR7
TRGET_T FCMP/GT  FR2,FR7  ;T ビット = 0
BT           TRGET_T    ;分岐しない (T=0)
.END
```

### 8.3.4 FDIV Floating point DIVide : 浮動小数点命令 浮動小数点数除算

書式	動作概略	命令コード	レイテンシー	ピッチ	Tビット
FDIV FRm, FRn	FRn/FRm→FRn	1111nnnnmmmm0011	14	13	-

#### (1) 説明

浮動小数点レジスタ FRn の内容を浮動小数点レジスタ FRm の内容によって、(浮動小数点数として)算術的に除算します。演算結果は、FRn に書き込まれます。

#### (2) 動作内容

```

FDIV(float *FRm,*FRn)                                /* FDIV FRm,FRn */
{
    clear_cause_VZ();
    if((data_type_of(FRm) == sNaN) | |
        (data_type_of(FRn) == sNaN))                invalid(FRn);
    else if((data_type_of(FRm) == qNaN) | |
        (data_type_of(FRn) == qNaN))                qnan(FRn);
    else case((data_type_of(FRm)
NORM :
case(data_type_of(FRn))
    PINF      :
    NINF      : inf(FRn,sign_of(FRm)^sign_of(FRn));      break;
    default   : *FRn =*FRn / *FRm;                          break;
    }
PZERO :
NZERO :
case(data_type_of(FRn))
    PZERO     :
    NZERO     : invalid(FRn);                                break;
    PINF      :
    NINF      : inf(FN,Sign_of(FRm)^sign_of(FRn));        break;
    default   : dz(FRn,sign_of(FRm)^sign_of(FRn));        break;
    }
PINF :
NINF :
case(data_type_of(FRn))
    PINF      :
    NINF      : invalid(FRn);                                break;

```

## 8. 各命令の説明

```

        default :zero (FRn,sign_of (FRm) ^sign_of (FRn));           break
                                                                break;
    }
    pc += 2;
}

```

FDIV 特殊ケース

FRm	FRn						
	NORM	+0	-0	+INF	-INF	qNaN	sNaN
NORM	DIV	0		INF		qNaN	Invalid
+0	DZ	Invalid					
-0							
+INF	0	+0	-0	Invalid			
-INF		-0	+0				
qNaN	qNaN						
sNaN	Invalid						

【注】 非正規化数の値は、ゼロとして扱われます。

### (3) 例外

無効演算 (Invalid Operation)、ゼロによる除算

### (4) 使用例

```

FDIV      FR6, FR5      ;浮動小数点数除算
                                ;実行前:  ;FR5=H'40800000/*10 進数の4*/
                                ;          ;FR6=H'40400000/*10 進数の3*/
                                ;実行後:  ;FR5=H'3FAAAAAA/*10 進数の1.33...*/
                                ;          ;FR6=H'40400000

```

### 8.3.5 FLDI0 Floating point Load Immediate 0 : 浮動小数点命令

浮動小数点ロードイミディエイト 0

書式	動作概略	命令コード	レイテンシー	ピッチ	Tビット
FLDI0 FRn	H'00000000→FRn	1111nnnn10001101	2	1	-

#### (1) 説明

浮動小数点レジスタ FRn に、浮動小数点数の 0 (0x00000000) をロードします。

#### (2) 動作内容

```
FLDI0(float *FRn)                                /* FLDI0 FRn */
{
    *FRn = 0x00000000;
    pc += 2;
}
```

#### (3) 例外

なし

#### (4) 使用例

```
FLDI0    FR1                ;イミディエイト 0 をロード
;実行前:                    FR1=x (don't care)
;実行後:                    FR1=00000000
```

## 8. 各命令の説明

---

### 8.3.6 FLDI1 Floating point Load Immediate 1 : 浮動小数点命令 浮動小数点ロードイミディエイト 1

---

書式	動作概略	命令コード	レイテンシー	ピッチ	Tビット
FLDI1 FRn	H'3F800000→FRn	1111nnnn10011101	2	1	-

#### (1) 説明

浮動小数点レジスタ FRn に、浮動小数点数の 1 (0x3F800000) をロードします。

#### (2) 動作内容

```
FLDI1(float *FRn)                                /* FLDI1 FRn */
{
    *FRn = 0x3F800000;
    pc += 2;
}
```

#### (3) 例外

なし

#### (4) 使用例

```
FLDI1      FR2                                ;イミディエイト 1 をロード
;実行前:      FR2=x (don't care)
;実行後:      FR2=H' 3F800000/*10 進数の 1*/
```

### 8.3.7 FLDS Floating point Load to System register : 浮動小数点命令 システムレジスタへの浮動小数点数ロード

書式	動作概略	命令コード	レイテンシー	ピッチ	Tビット
FLDS FRm,FPUL	FRm→FPUL	1111nnnn00011101	2	1	-

#### (1) 説明

浮動小数点レジスタ FRm の内容を、システムレジスタ FPUL にコピーします。

#### (2) 動作内容

```

FLDS(float *FRm,*FPUL)                               /* FLDS FRm,FPUL */
{
    *FPUL = *FRm;
    pc += 2;
}

```

#### (3) 例外

なし

#### (4) 使用例

```

;FLDS および FSTS 命令実行前:
FLDI1    FR6          ;FR6=H'3F800000/*10 進数の 1*/
FLDI0    FR2          ;FR2=0
;FLDS および FSTS 命令実行後:
FLDS     FR6, FPUL    ;FPUL=H'3F800000
FSTS     FPUL, FR2    ;FR2= H'3F800000

```

### 8.3.8 FLOAT FLOATing point Convert from Integer : 浮動小数点命令 整数から浮動小数点数への変換

書式	動作概略	命令コード	レイテンシー	ピッチ	Tビット
FLOAT FPUL,FRn	(float)FPUL→FRn	1111nnnn00101101	2	1	-

#### (1) 説明

FPULの内容を整数値として解釈して、その値を浮動小数点数に変換します。演算結果は、浮動小数点レジスタFRnに書き込まれます。

#### (2) 動作内容

```

FLOAT(int, *FPUL, float *FRn)          /* FLOAT FRn */
{
    clear_cause_VZ();
    *FRn = (float)*FPUL;
    pc += 2;
}

```

#### (3) 例外

なし

#### (4) 使用例

```

;整数値を浮動小数点数値に変換
;FLOAT 命令実行前:
MOV.L    #H'00000003, R1    ;          R1=H'00000003
FLDIO    FR2                ;          FR2=0
;FLOAT 命令実行後:
LDS      R1, FPUL           ;          FPUL=H'00000003
FLOAT    FPUL, FR2         ;          FR2=H'40400000/*10進数の3*/

```



### 8.3.9 FMAC Floating point Multiply ACcumulate : 浮動小数点命令 浮動小数点数積和演算

書式	動作概略	命令コード	レイテンシー	ピッチ	Tビット
FMAC FR0,FRm,FRn	$FR0 \times FRm + FRn$ →FRn	1111nnnnmmmm1110	2	1	-

#### (1) 説明

浮動小数点レジスタ FR0 と FRm の内容を、(浮動小数点数として)算術的に乗算します。この演算結果に対して、浮動小数点レジスタ FRn の内容を加算し、その結果を FRn に書き込みます。

#### (2) 動作内容

```

FMAC(float *FR0,*FRm,*FRn)                               /* FMAC FR0,FRm,FRn */
{
long      tmp_FPSCR;
float     *tmp_FMUL = *FRm;
          FMUL(F0,tmp_FMUL);
          pc -=2;                                           /* correct pc */
          tmp_FPSCR = FPSCR;                                /* save cause field for FR0*FRm
          */
          FADD(tmp_FMUL,FRn);
          FPSCR |= tmp_FPSCR;                               /* reflect cause field for F0*FRm
          */
}

```

8. 各命令の説明

FMAC 特殊ケース

FRn	FR0	FRm								
		+NORM	-NORM	+0	-0	+INF	-INF	qNaN	sNaN	
NORM	NORM	MAC				INF				
	0					Invalid				
	+INF	+INF	-INF	Invalid		+INF	-INF			
	-INF	-INF	+INF			-INF	+INF			
+0	NORM	MAC				INF				
	0					Invalid				
	+INF	+INF	-INF	Invalid		+INF	-INF			
	-INF	-INF	+INF			-INF	+INF			
-0	+NORM	MAC			+0	-0	+INF	-INF		
	-NORM				-0	+0	-INF	+INF		
	+0	+0	-0	+0	-0	Invalid				
	-0	-0	+0	-0	+0					
	+INF	+INF	-INF	Invalid		+INF	-INF			
	-INF	-INF	+INF			-INF	+INF			
+INF	+NORM	+INF				Invalid				
	-NORM					Invalid				
	0					Invalid				
	+INF	Invalid			Invalid		+INF			
	-INF	Invalid	+INF							+INF
-INF	+NORM	-INF				Invalid				
	-NORM					Invalid				
	0					Invalid				
	+INF	Invalid	Invalid		Invalid		-INF			
	-INF	-INF					-INF			Invalid
qNaN	0					Invalid				
	INF					Invalid				
	!sNaN									
!NaN	qNaN							qNaN		
All types	sNaN									
sNaN	All types									

【注】 非正規化数の値は、ゼロとして扱われます。

(3) 例外

無効演算 ( Invalid Operation )

## (4) 使用例

```
FMAC FR0, FR3, FR5 ;浮動小数点数積和演算
                    FR0*FR3+FR5->FR5
;実行前:           FR0=H'40000000/*10 進数の 2*/
;                  FR3=H'40800000/*10 進数の 4*/
;                  FR5=H'3F800000/*10 進数の 1*/
;実行後:           FR0=H'40000000/*10 進数の 2*/
;                  FR3=H'40800000/*10 進数の 4*/
;                  FR5=H'41100000/*10 進数の 9*/

FMAC FR0, FR0, FR5 ;FR0*FR0+FR5->FR5
;実行前:           FR0=H'40000000/*10 進数の 2*/
;                  FR5=H'3F800000/*10 進数の 1*/
;実行後:           FR0=H'40000000/*10 進数の 2*/
;                  FR5=H'40A00000/*10 進数の 5*/

FMAC FR0, FR5, FR0 ;FR0*FR5+FR0->FR5
;実行前:           FR0=H'40000000/*10 進数の 2*/
;                  FR5=H'40A00000/*10 進数の 5*/
;実行後:           FR0=H'41400000/*10 進数の 12*/
;                  FR5=H'40A00000/*10 進数の 5*/
```

## 8. 各命令の説明

### 8.3.10 FMOV Floating point MOVE : 浮動小数点命令

#### 浮動小数点数転送

書式	動作概略	命令コード	レイテンシー	ピッチ	Tビット
1. FMOV FRm, FRn	FRm → FRn	1111nnnnmmmm1100	2	1	-
2. FMOV.S @Rm, FRn	(Rm) → FRn	1111nnnnmmmm1000	2	1	-
3. FMOV.S FRm, @Rn	FRm → (Rn)	1111nnnnmmmm1010	2	1	-
4. FMOV.S @Rm+,FRn	(Rm) → FRn, Rm+=4	1111nnnnmmmm1001	2	1	-
5. FMOV.S FRm,@-Rn	Rn-=4, FRm→(Rn)	1111nnnnmmmm1011	2	1	-
6. FMOV.S @(R0,Rm), FRn	(R0+Rm)→FRn	1111nnnnmmmm0110	2	1	-
7. FMOV.S FRm,@(R0,Rn)	FRm→(R0+Rn)	1111nnnnmmmm0111	2	1	-

#### (1) 説明

- 浮動小数点レジスタFRmの内容を浮動小数点レジスタFRnに転送します。
- 汎用レジスタRmにより指定されたアドレスのメモリ内容を、浮動小数点レジスタFRnにロードします。
- 浮動小数点レジスタFRmの内容を、汎用レジスタRnにより指定されたアドレスのメモリ位置にストアします。
- 汎用レジスタRmにより指定されたアドレスのメモリ内容を、浮動小数点レジスタFRnにロードします。ロードが正常に完了した後、Rmの値が4インクリメントされます。
- 浮動小数点レジスタFRmの内容を、汎用レジスタRn-4により指定されたアドレスのメモリ位置にストアします。ストアが正常に完了した後、デクリメントされた値(Rn-4)がRnの値となります。
- 汎用レジスタRmとR0により指定されたアドレスのメモリ内容を、浮動小数点レジスタFRnにロードします。
- 浮動小数点レジスタFRmの内容を、汎用レジスタRnとR0により指定されたアドレスのメモリ位置にストアします。

#### (2) 動作内容

```

FMOV(float *FRm,*FRn)      /* FMOV.S FRm,FRn */
{
    *FRn = *FRm;
    pc += 2;
}

FMOV_LOAD(long *Rm,float *FRn) /* FMOV @Rm,FRn */
{
    if (load_long(Rm,FRn) !=Address_Error)
        load_long(Rm,FRn);
    pc += 2;
}

FMOV_STORE(float *FRm,long *Rn) /* FMOV.S FRm,@Rn */

```

```
{
    if (store_long(FRm,tmp_address) !=Address_Error)
        store_long(FRm,Rn);
    pc += 2;
}
FMOV_RESTORE(long *Rm,float *FRn) /* FMOV.S @Rm+,FRn */
{
    if (load_long(Rm,FRn) !=Address_Error)
        *Rm += 4;
    pc += 2;
}
FMOV_SAVE(float *FRm,long *Rn) /*FMOV.S FRm,@-Rn */
{
    long *tmp_address =*Rn -4;
    if (store_long(FRm,tmp_address) !=Address_Error)
        Rn = tmp_address;
    pc += 2;
}
FMOV_LOAD_index(long *Rm, long *R0, float *FRn) /* FMOV.S @(R0,Rm),FRn*/
{
    if (load_long(&(*Rm+*R0),FRn), != Address_Error);
    pc += 2;
}
FMOV_STORE_index(float *FRm,long *R0, long *Rn) /* FMOV.S FRm,@(R0,Rn) */
{
    if (store_long(FRm,&(*Rn+*R0)), != Address_Error);
    pc += 2;
}
```

### (3) 例外

アドレスエラー

## 8. 各命令の説明

---

### (4) 使用例

```
FMOV.S    @R1, FR2    ;ロード
;実行前:    @R1=H'00ABCDEF
;           FR2=0
;実行後:    @R1=H'00ABCDEF
;           FR2=H'00ABCDEF

FMOV.S    FR2, @R3    ;ストア
;実行前:    @R3=0
;           FR2=H'40800000
;実行後:    @R3=H'40800000
;           FR2=H'40800000

FMOV.S    @R3+, FR3   ;リストア
;実行前:    R3=H'0C700028
;           @R3=H'40800000
;           FR3=0
;実行後:    R3=H'0C70002C
;           FR3=H'40800000

FMOV.S    FR4, @-R3   ;セーブ
;実行前:    R3=H'0C700044
;           @R3=0
;           FR4=H'01234567
;実行後:    R3=H'0C700040
;           @R3=H'01234567
;           FR4=H'01234567

FMOV.S    @(R0, R3), FR4 ;インデックス付きロード
;実行前:    R0=H'00000004
;           R3=H'0C700040
;           @H'0C700044=H'00ABCDEF
;           FR=4
;実行後:    R0=H'00000004
;           R3=H'0C700040
;           FR4=H'00ABCDEF

FMOV.S    FR5, @(R0, R3) ;インデックス付きストア
;実行前:    R0=H'00000028
```

```

; R3=H'0C700040
; @H'0C700068=0
; FR5=H'76543210
;実行後: R0=H'00000028
; R3=H'0C700040
; @H'0C700068=H'76543210

FMOV.S    FR5, FR6    ;レジスタファイル内
;実行前:  FR5=H'76543210
;          FR6=x(don't care)
;実行後:  FR5=H'76543210
;          FR6=H'76543210
```

## 8.3.11 FMUL Floating point MULTiply : 浮動小数点命令

## 浮動小数点数乗算

書式	動作概略	命令コード	レイテンシー	ピッチ	Tビット
FMUL FRm,FRn	FRn × FRm → FRn	1111nnnnmmmm0010	2	1	-

## (1) 説明

浮動小数点レジスタ FRm と FRn の内容を、(浮動小数点数として)算術的に乗算します。演算結果は、FRn に書き込まれます。

## (2) 動作内容

```

FMUL(float *FRm,*FRn)      /* FMUL FRm,FRn */
{
    clear_cause_VZ();
    if((data_type_of(FRm) == sNaN)    ||
        (data_type_of(FRn) == sNaN))    invalid(FRn);
    else if((data_type_of(FRm) == qNaN) ||
        (data_type_of(FRn) == qNaN))    qnan(FRn);
    else case(data_type_of(FRm)      {
        NORM      :
        case(data_type_of(FRn))      {
            PINF      :
            NINF      :  inf(FRn,sign_of(FRm)^sign_of(FRn));    break;
            default   :  *FRn=(*FRn)*(*FRm);                      break;
        }
        PZERO      :
        NZERO      :
        case(data_type_of(FRn))      {
            PINF      :
            NINF      :  invalid(FRn);                            break;
            default   :  zero(FRn,sign_of(FRm)^sign_of(FRn));    break;
        }
        PINF      :
        NINF      :
        case(data_type_of(FRn))      {
            PZERO      :
            NZERO      :  invalid(FRn);                            break;
            default   :  inf (FRn,sign_of(FRm)^sign_of(FRn));    break;
        }
    }
}

```



```

}
pc += 2;
}

```

FMUL 特殊ケース

FRm	FRn						
	NORM	+0	-0	+INF	-INF	qNaN	sNaN
NORM	MUL	0		INF		qNaN	Invalid
+0	0	+0	-0	Invalid			
-0		-0	+0				
+INF	INF	Invalid		+INF	-INF		
-INF				-INF	+INF		
qNaN							
sNaN							Invalid

【注】 非正規化数の値は、ゼロとして扱われます。

## (3) 例外

無効演算 (Invalid Operation)

## (4) 使用例

```

FMUL      FR2, FR3      ;浮動小数点数乗算
;実行前:      FR2=H' 40000000/*10 進数の 2*/
;              FR3=H' 40800000/*10 進数の 4*/
;実行後:      FR2=H' 40000000
;              FR3=H' 41000000/*10 進数の 8*/

```

## 8. 各命令の説明

### 8.3.12 FNEG Floating point NEGate : 浮動小数点命令

浮動小数点数符号反転

書式	動作概略	命令コード	レイテンシー	ピッチ	Tビット
FNEG FRn	-FRn→FRn	1111nnnn01001101	2	1	-

#### (1) 説明

浮動小数点レジスタ FRn の内容を、(浮動小数点数として)算術的に符号を反転します。  
演算結果は、FRn に書き込まれます。

#### (2) 動作内容

```
FNEG(float *FRn)          /* FNEG FRn */
{
    clear_cause_VZ();
    case(data_type_of(FRn)) {
        qNaN   :   qnan(FRn); break;
        sNaN   :   invalid(FRn); break;
        default:   *FRn = -(*FRn); break;
    }
    pc += 2;
}
```

#### FNEG 特殊ケース

FRn	NORM	+0	-0	+INF	-INF	qNaN	sNaN
FNEG(FRn)	NEG	-0	+0	-INF	+INF	qNaN	Invalid

【注】 非正規化数の値は、ゼロとして扱われます。

#### (3) 例外

無効演算 (Invalid Operation)

#### (4) 使用例

```
FNEG      FR2          ;浮動小数点数符号反転
;実行前:          FR2=H'40800000/*10 進数の4*/
;実行後:          FR2=H'C0800000/*10 進数の-4*/
```

### 8.3.13 FSQRT Floating point Square Root: 浮動小数点命令 浮動小数点数平方根

書式	動作概略	命令コード	レイテンシー	ピッチ	Tビット
FSQRT FRn	$\sqrt{FRn} \rightarrow FRn$	1111nnnn01101101	14	13	-

#### (1) 説明

浮動小数点レジスタ FRn の内容の（浮動小数点数としての）算術的な平方根をとります。演算結果は、FRn に書き込まれます。

#### (2) 動作内容

```
FSQRT(float *FRn)          /* FSQRT FRn */
{
    clear_cause_VZ();
    case(data_type_of(FRn)) {
    NORM                    :  if(sign_of(FRn) == 0)
                                *FRn = sqrt(*FRn);
                                else invalid(FRn);          break;
    PZERO                   :
    NZERO                   :
    PINF                    :          *FRn = *FRn;          break;
    NINF                    :          invalid(FRn);          break;
    qNaN                    :          qnan(FRn);             break;
    sNaN                    :          invalid(FRn);          break;
    }
    pc += 2;
}
```

#### FSQRT 特殊ケース

FRn	+NORM	-NORM	+0	-0	+INF	-INF	qNaN	sNaN
FSQRT(FRn)	SQRT	Invalid	+0	-0	+INF	Invalid	qNaN	Invalid

【注】 非正規化数の値は、ゼロとして扱われます。

#### (3) 例外

無効演算 (Invalid Operation)

## 8. 各命令の説明

---

### (4) 使用例

```
FSQRT      FR4      ;浮動小数点数平方根
              ;実行前:      ;FR4=H'40400000/*10進数の3*/
              ;実行後:      ;FR4=H'3FDDB3D7/*10進数の1.7320*/
```

### 8.3.14 FSTS Floating point STore from System register : 浮動小数点命令

システムレジスタからの浮動小数点数ストア

書式	動作概略	命令コード	レイテンシー	ピッチ	Tビット
FSTS FPUL,FRn	FPUL→FRn	1111nnnn00001101	2	1	-

#### (1) 説明

システムレジスタ FPUL の内容を浮動小数点レジスタ FRn にコピーします。

#### (2) 動作内容

```
FSTS(float *FRn,*FPUL)      /* FSTS FPUL,FRn */
{
    *FRn = *FPUL;
    pc += 2;
}
```

#### (3) 例外

なし

#### (4) 使用例

```
MOV.L    #H'00000002, R2    ;FSTS 命令実行前:    ;R2=H'00000002
FLDIO    FR5                ;FR5=0
LDS      R2,FPUL            ;FSTS 命令実行後:    ;R2=H'00000002
FSTS     FPUL, R5           ;FR5= H'00000002
```

## 8.3.15 FSUB Floating point SUBtract : 浮動小数点命令

## 浮動小数点数減算

書式	動作概略	命令コード	レイテンシー	ピッチ	Tビット
FSUB FRm, FRn	FRn-FRm→FRn	1111nnnnmmmm0001	2	1	-

## (1) 説明

浮動小数点レジスタ FRn の内容から浮動小数点レジスタ FRm の内容を、(浮動小数点数として)算術的に減算します。演算結果は、FRn に書き込まれます。

## (2) 動作内容

```

FSUB (float *FRm, FRn) /* FSUB FRm, FRn */
{
    clear_cause_VZ();
    if((data_type_of(FRm) == sNaN) | |
        (data_type_of(FRn) == sNaN) | | invalid(FRn);
        else if((data_type_of(FRm) == qNaN) | |
            (data_type_of(FRn) == qNaN) | | qnan(FRn);
    else case(data_type_of(FRm)) {
        NORM :
        case(data_type_of(FRn)) {
            PINF : inf(FRn, 0); break;
            NINF : inf(FRn, 1); break;
            default : *FRn = *FRn - *FRm; break;
        }
        PZERO :
        case(data_type_of(FRn)) {
            NORM : *FRn = *FRn - *FRm; break;
            PZERO : zero(FRn, 0); break;
            NZERO : zero(FRn, 1); break;
            PINF : inf(FRn, 0); break;
            NINF : inf(FRn, 1); break;
        }
        NZERO :
        case(data_type_of(FRn)) {
            NORM : *FRn = *FRn - *FRm; break;
            PZERO :
            NZERO : zero(FRn, 0); break;
            PINF : inf(FRn, 0); break;
        }
    }
}

```

```

NINF      :          inf(FRn,1);          break;
}
PINF      :
case(data_type_of(FRn)) {
NINF      :          invalid(FRn);       break;
default   :          inf(FRn,1);       break;
}
NINF :
case(data_type_of(FRn)) {
PINF      :          invalid(FRn);       break;
default   :          inf(FRn,0);       break;
}
}
pc += 2;
}

```

FSUB 特殊ケース

FRm	FRn						
	NORM	+0	-0	+INF	-INF	qNaN	sNaN
NORM	SUB			+INF	-INF	qNaN	Invalid
+0			-0				
-0	+0						
+INF	-INF			Invalid			
-INF	+INF				Invalid		
qNaN							
sNaN							

【注】 非正規化数の値は、ゼロとして扱われます。

### (3) 例外

無効演算 (Invalid Operation) 例外

## 8. 各命令の説明

---

### (4) 使用例

```
FSUB      FR0, FR3      ;浮動小数点数減算
;実行前:                ;FR0=H'3F800000/*10 進数の 1*/
;                        ;FR3=H'40E00000/*10 進数の 7*/
;実行後:                ;FR0=H'3F800000/*10 進数の 1*/
;                        ;FR3=H'40C00000/*10 進数の 6*/

FSUB      FR3, FR2      ;
;実行前:                ;FR2=H'40800000/*10 進数の 4*/
;                        ;FR3=H'40C00000/*10 進数の 6*/
;実行後:                ;FR2=H'C0000000/*10 進数の -2*/
;                        ;FR3=H'40C00000/*10 進数の 6*/
```



### 8.3.16 FTRC Floating point TRuncate and Convert to integer : 浮動小数点命令

浮動小数点数から整数への切り捨て変換

書式	動作概略	命令コード	レイテンシー	ピッチ	Tビット
FTRC FRm, FPUL	(long)FRm→FPUL	1111nnnn00111101	2	1	-

#### (1) 説明

浮動小数点レジスタ FRm の内容を浮動小数点数として解釈し、その値の小数部を切り捨てて整数に変換します。演算結果は、FRn に書き込まれます。

#### (2) 動作内容

```
#define N_INT_RANGE 0xCF000000          /* 01.000000 * 2^16 */
#define P_INT_RANGE 0x47FFFFFF          /* 1.fffffe * 2^30 */

FTRC(float *FRm,int *FPUL)             /* FTRC FRm,FPUL */
{
    clear_cause_VZ();
    case(ftrc_type_of(FRm)) {
        NORM      :    *FPUL = (long)(*FRm); break;
        PINF      :    ftrc_invalid(0);      break;
        NINF      :    ftrc_invalid(1);      break;
    }
    pc += 2;
}

int ftrc_type_of(long *src)
{
    long abs;
    abs = *src & 0x7FFFFFFF;
    if(sign_of(src) == 0) {
        if(abs > 0x7F800000) return(NINF); /* NaN */
        else if(abs > P_INT_RANGE) return(PINF); /* out of range,+INF */
        else return(NORM); /* +0,+NORM */
    }
    else {
        if(*src > N_INT_RANGE) return(NINF); /* out of range ,+INF,NaN */
        else return(NORM); /* -0,-NORM */
    }
}
}
```

## 8. 各命令の説明

```

ftrc_invalid(long *dest,int sign)
{
    set_V();
    if((FPSCR & ENABLE_V) == 0) {
        if(sign == 0)    *dest = 0x7FFFFFFF;
        else             *dest = 0x80000000;
    }
}

```

FTRC 特殊ケース

FRn	NORM	+0	-0	positive out of range	negative out of range	+INF	-INF	qNaN	sNaN
FTRC (FRn)	TRC	0	0	7FFFFFFF	80000000	Invalid +MAX	Invalid -MAX	Invalid -MAX	Invalid -MAX

【注】 非正規化数の値は、ゼロとして扱われます。

### (3) 例外

無効演算 (invalid Operation)

### (4) 使用例

```

MOV.L    #H'402ED9EB, R2
LDS      R2, FPUL
FSTS     FPUL, FR6      ;FR6=H'402ED9EB/*10 進数の 2.7320*/
FTRC     FR6, FPUL
STS      FPUL, R2      ;R2=H'00000002/*10 進数の 2*/
                        ;FTRC および STS 命令実行前:
                        ;    R2=H'402ED9EB
                        ;    FR6=H'402ED9EB
                        ;FTRC および STS 命令実行後:
                        ;    R2=H'00000002
                        ;    FR6=H'402ED9EB

```

## 8.3.17 LDS Load to FPU System register : FPU に関する CPU 命令

FPU システムレジスタへの転送

書式	動作概略	命令コード	レイテンシー	ピッチ	Tビット
1. LDS Rm, FPUL	Rm→FPUL	0100nnnn01011010	2	1	-
2. LDS.L@Rm+,FPUL	(Rm)→FPUL, Rm+=4	0100nnnn01010110	2	1	-
3. LDS Rm,FPSCR	Rm→FPSCR	0100nnnn01101010	3	1	-
4. LDS.L @Rm+,FPSCR	(Rm)→FPSCR, Rm+=4	0100nnnn01100110	3	1	-

## (1) 説明

- 汎用レジスタRmの内容をシステムレジスタFPULに転送します。
- 汎用レジスタRmにより指定されたアドレスのメモリ内容を、システムレジスタFPULにロードします。ロードが正常に完了した後、Rmの値が4インクリメントされます。
- 汎用レジスタRmの内容をシステムレジスタFPSCRに転送します。FPSCR中のあらかじめ値が定義されているビットは変更されません。
- 汎用レジスタRmにより指定されたアドレスのメモリ内容を、システムレジスタFPSCRにロードします。ロードが正常に完了した後、Rmの値が4インクリメントされます。FPSCR中のあらかじめ値が定義されているビットは、変更されません。

## (2) 動作内容

```
#define FPSCR_MASK 0x00018C60
LDS(long *Rm,*FPUL) /* LDS Rm,FPUL */
{
    *FPUL = *Rm;
    pc += 2;
}
LDS_RESTORE(long *Rm, *FPUL) /* LDS.L @Rm+,FPUL */
{
    if(load_long(Rm,FPUL) != Address_Error) *Rm += 4 ;
    pc += 2;
}

LDS(long *Rm,*FPSCR) /* LDS Rm,FPSCR */
{
    *FPSCR = *Rm & FPSCR_MASK;
    pc += 2;
}
LDS_RESTORE(long *Rm, *FPSCR) /* LDS.L @Rm+,FPSCR */
{
```

## 8. 各命令の説明

---

```
long *tmp_FPSCR;
if(load_long(Rm, tmp_FPSCR) != Address_Error){
    *FPSCR =*tmp_FPSCR & FPSCR_MASK;
    *Rm += 4 ;
}
pc += 2;
}
```

### (3) 例外

アドレスエラー

### (4) 使用例

#### ・LDS

##### 例 1

```
MOV.L    #H'12345678, R2      ;LDS および FSTS 命令実行前:
;                               R2=H'12345678
FLDI0    FR3                  ;                               FR3=0
LDS      R2, FPUL             ;LDS および FSTS 命令実行後:
;                               R2=H'12345678
FSTS     FPUL, FR3           ;                               FR3= H'12345678
```

##### 例 2

```
MOV.L    #H'00040801, R4      ;LDS 命令実行後:
LDS      R4, FPSCR           ;FPSCR=00040801
```

#### ・LDS.L

##### 例 1

```
LDI0     FR0                  ;LDS.L および FSTS 命令実行前:
MOV.L    #H'87654321, R4      ;                               FR0=0
MOV.L    #H'0C700128, R8      ;                               R8=0C700128
MOV.L    R4, @R8              ;LDS.L および FSTS 命令実行後:
LDS.L    @R8+, FPUL           ;                               FR0=87654321
FSTS     FPUL, FR0           ;                               R8=0C70012C
```

##### 例 2

```
MOV.L    #H'00040C01, R4      ;LDS.L 命令実行前:
MOV.L    #H'0C700134, R8      ;                               R8=0C700134
MOV.L    R4, @R8              ;LDS.L 命令実行後:
;                               R8=0C700138
LDS.L    @R8+, FPSCR         ;                               FPSCR=00040C01
```

### 8.3.18 STS STore from FPU System register : FPU に関する CPU 命令

FPU システムレジスタからの転送

書式	動作概略	命令コード	レイテンシー	ピッチ	Tビット
1. STS FPUL,Rn	FPUL→Rn	0000nnnn01011010	2	1	-
2. STS.L FPUL,@-Rn	Rn -= 4, FPUL→@(Rn)	0100nnnn01010010	2	1	-
3. STS FPSCR,Rn	FPSCR→Rn	0000nnnn01101010	3	1	-
4. STS.L FPSCR,@-Rn	Rn -= 4, FPSCR→@(Rn)	0100nnnn01100010	3	1	-

#### (1) 説明

1. システムレジスタFPULの内容を汎用レジスタRnに転送します。
2. システムレジスタFPULの内容を、汎用レジスタRn-4により指定されたアドレスのメモリ位置にストアします。ストアが正常に完了した後、デクリメントされた値がRnの値となります。
3. システムレジスタFPSCRの内容を汎用レジスタRnに転送します。
4. システムレジスタFPSCRの内容を、汎用レジスタRn-4により指定されたアドレスのメモリ位置にストアします。ストアが正常に完了した後、デクリメントされた値がRnの値となります。

#### (2) 動作内容

```

STS(long *FPUL,*Rn)                                     /* STS.L FPUL,Rn */
{
    *Rn = *FPUL;
    pc += 2;
}

STS_SAVE(long *FPUL,*Rn)                               /* STS.L FPUL,@-Rn */
{
    long *tmp_address = *Rn - 4;
    if(store_long(FPUL,tmp_address) != Address_Error)
        Rn = tmp_address;
    pc += 2;
}

STS(long *FPSCR,*Rn)                                   /* STS FPSCR,Rn */
{
    *Rn = *FPSCR;
    pc += 2;
}

STS_RESTORE long *FPSCR,*Rn)                          /* STS.L FPSCR,@-Rn */
{
    long *tmp_address = *Rn - 4;
    if(store_long(FPSCR tmp_address) != Address_Error)

```

## 8. 各命令の説明

---

```
Rn = tmp_address
pc += 2;
}
```

### (3) 例外

アドレスエラー

### (4) 使用例

#### STS

##### 例 1

```
MOV.L #H'12ABCDEF, R12
LDS.L @R12, FPUL
STS   FPUL, R13
```

```
; STS 命令実行後:
;     R13 = 12ABCDEF
```

##### 例 2

```
STS   FPSCR, R2
```

```
; STS 命令実行後:
; FPSCR のその時点の内容が R2 レジスタにストアされます。
```

#### STS.L

##### 例 1

```
MOV.L #H'0C700148, R7
STS   FPUL, @-R7
```

```
; STS.L 命令実行前:
;     R7 = H'0C700148
; STS.L 命令実行後:
;     R7 = H'0C700144, FPUL の内容は H'0C700144 番地の
;     メモリにセーブされます。
;     location H'0C700144
```

##### 例 2

```
MOV.L #H'0C700154, R8
STS.L FPSCR, @-R8
```

```
; STS.L 命令実行後:
;     FPSCR の内容は H'0C700150 番地のメモリにセーブされます
```

## 8.4 DSP データ転送命令の説明 (SH3-DSP のみ)

DSP データ転送命令を、アルファベット順に示します。

表 8.1 アルファベット順 DSP データ転送命令

命令	動作	命令コード	実行 ステート	DC ビット
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+lx,Ds	(As)→Ds、As + ls→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+ls	Ds→(As)、As + ls→As	111101AADDDD1111	1	-
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+ls,Ds	(As)→MSW of Ds、0→LSW of Ds、 As + ls→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+ls	MSW of Ds→(As)、As + ls→As	111101AADDDD1101	1	-
MOVX.W @Ax,Dx	(Ax)→MSW of Dx、0→LSW of Dx	111100A*D*0*01**	1	-
MOVX.W @Ax+,Dx	(Ax)→MSW of Dx、0→LSW of Dx、 Ax + 2→Ax	111100A*D*0*10**	1	-
MOVX.W @Ax+lx,Dx	(Ax)→MSW of Dx、0→LSW of Dx、 Ax + lx→Ax	111100A*D*0*11**	1	-
MOVX.W Da,@Ax	MSW of Da→(Ax)	111100A*D*1*01**	1	-
MOVX.W Da,@Ax+	MSW of Da→(Ax)、Ax + 2→Ax	111100A*D*1*10**	1	-
MOVX.W Da,@Ax+lx	MSW of Da→(Ax)、Ax + lx→Ax	111100A*D*1*11**	1	-
MOVY.W @Ay,Dy	(Ay)→MSW of Dy、0→LSW of Dy	111100*A*D*0**01	1	-
MOVY.W @Ay+,Dy	(Ay)→MSW of Dy、0→LSW of Dy、 Ay + 2→Ay	111100*A*D*0**10	1	-
MOVY.W @Ay+ly,Dy	(Ay)→MSW of Dy、0→LSW of Dy、 Ay + ly→Ay	111100*A*D*0**11	1	-
MOVY.W Da,@Ay	MSW of Da→(Ay)	111100*A*D*1**01	1	-
MOVY.W Da,@Ay+	MSW of Da→(Ay)、Ay + 2→Ay	111100*A*D*1**10	1	-
MOVY.W Da,@Ay+ly	MSW of Da→(Ay)、Ay + ly→Ay	111100*A*D*1**11	1	-
NOPX	No Operation	1111000*0*0*00**	1	-
NOPY	No Operation	111100*0*0*0**00	1	-

【注】 MSW : オペランドの上位ワード

LSW : オペランドの下位ワード

## 8. 各命令の説明

X、Y データ転送 (MOVX.W、MOVY.W)、シングルデータ転送 (MOV.S.W、MOV.S.L) の動作を説明します。

### (1) X、Y データ転送 (MOVX.W、MOVY.W)

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。

同時並行して実行されるデータ演算命令が条件付き命令であっても、X、Y データ転送命令は条件に関係なく実行されます。

X、Y データ転送のロード、ストアの動作を図 8.1 に示します。

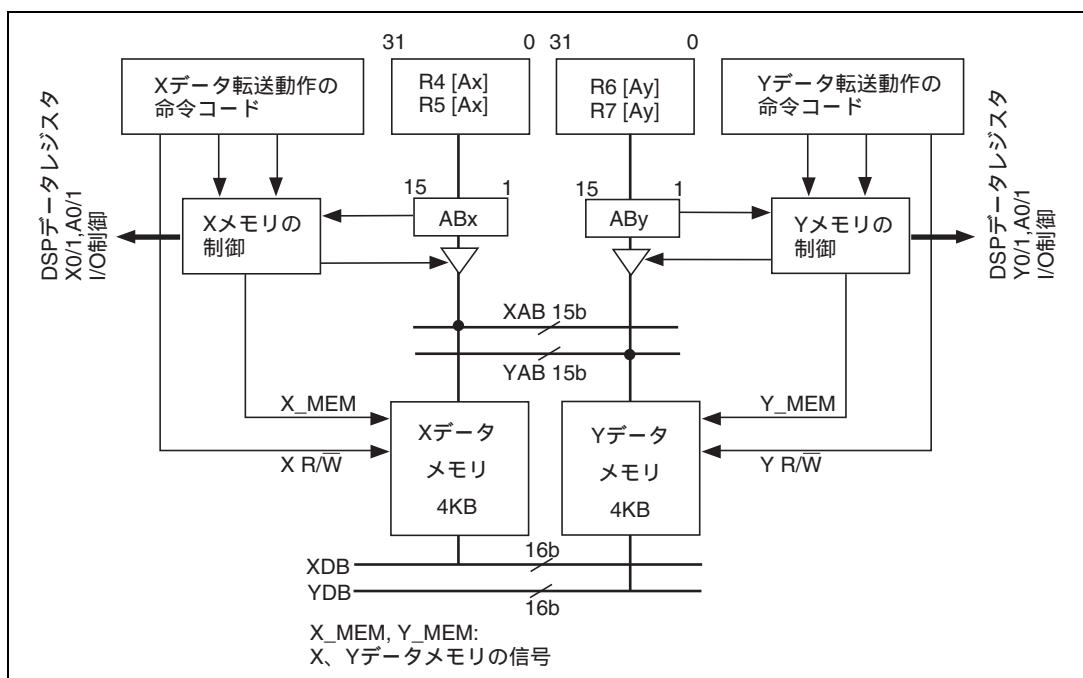


図 8.1 X、Y データ転送のロード、ストアの動作



X メモリデータ転送の動作を次に示します。Y メモリデータ転送も同様です。

```

if ( !Nop ) {
    X_MEM=1; XAB=ABx;X R/W=1
    if ( load opeation ) {
        Dx[31:16]=XDB;
        Dx[15:0] =0x0000; /* Dx is X0 or X1 */
    }
    else { XDB=Dx[31:16];X R/W=0; }
}
/* Dx is A0 or A1 */
else { X_MEM=0; XAB=Unknown; }

```

## (2) シングルデータ転送 (MOV.S.W、MOV.S.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。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。

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 8.2 に示します。

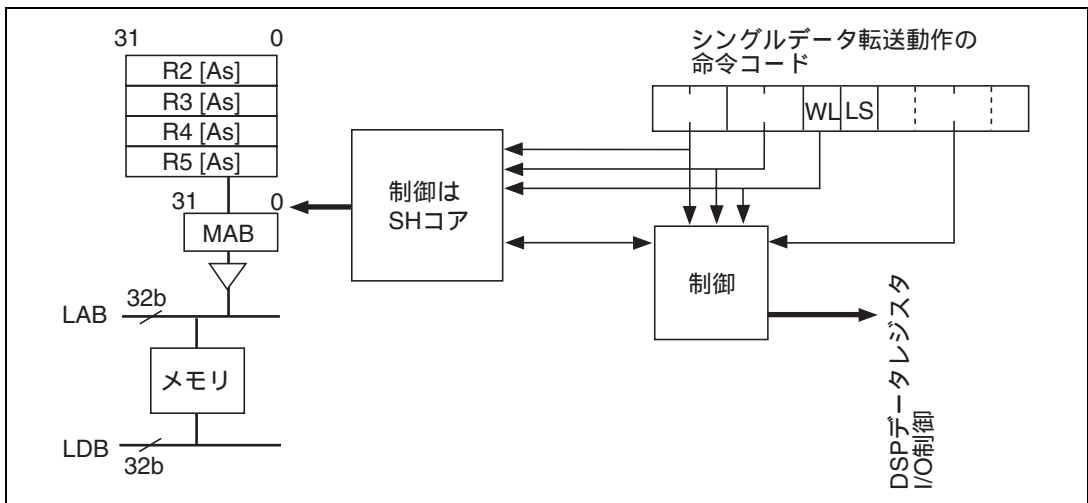


図 8.2 シングルデータ転送のロード、ストアの動作

## 8. 各命令の説明

---

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。

```
LAB = MAB;
if ( Ms!=NLS && W/L is word access ) { /* MOV.S.W */
    if (LS==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]; /* Ds is A0G or A1G */
    }
    else { /* Store */
        if (Ds!=A0G && Ds!=A1G) LDB[15:0] = Ds[31:16];
        /* Ds is A0G or A1G */
        else LDB[15:0] = Ds[7:0] with 8bit sign-extention;
    }
}
else if ( MA!=NLS && W/L is long-word access ) { /* MOV.S.L */
    if (LS==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]; /* Ds is A0G or A1G */
    }
    else { /* Store */
        if (Ds!=A0G && Ds!=A1G) LDB[31:0] = Ds[31:0];
        /* Ds is A0G or A1G */
        else LDB[31:0] = Ds[7:0] with 24bit sign-extention;
    }
}
}
```

次の形式で命令を説明します。

命令の名称      命令の機能（英文）  
命令の機能

命令の分類

書式	動作概略	命令コード	実行ステート	DCビット
アセンブラの入力書式を表示します。	動作の概略を表示します。	MSB↔LSBの順に表示します。	DSP命令はすべて1です。	命令実行後の、DCビットの状態を表示します。

- 書式

[if cc] OP.Sz SRC1,SRC2,DEST  
 [if cc] : 条件、(無条件、DCTまたはDCF)  
 OP : オペレーションニーモニック  
 Sz : サイズ  
 SRC1 : ソース1オペランド  
 SRC2 : ソース2オペランド  
 DEST : デスティネーションオペランド

- 動作概略

→,← : 転送方向  
 (xx) : メモリオペランド  
 DC : DSR内のフラグビット  
 & : ビットごとの論理積  
 | : ビットごとの論理和  
 ^ : ビットごとの排他的論理和  
 ~ : ビットごとの論理否定  
 <<n : 左nビットシフト  
 >>n : 右nビットシフト  
 MSW : 上位ワード(ビット31~16)  
 LSW : 下位ワード(ビット15~0)  
 [n1:n2] : ビットn1~n2

- 命令コード

ソースレジスタ、デスティネーションレジスタを表します。

Xデータ転送命令

A(Ax): 0=R4, 1=R5  
 D(デスティネーション、Dx): 0=X0, 1=X1  
 D(ソース、Da): 0=A0, 1=A1

Yデータ転送命令

A(Ay): 0=R6, 1=R7  
 D(デスティネーション、Dy): 0=Y0, 1=Y1  
 D(ソース、Da): 0=A0, 1=A1

シングルデータ転送命令

AA(As): 0=R4, 1=R5, 2=R2, 3=R3  
 DDDD(Ds): 5=A1, 7=A0, 8=X0, 9=X1, A=Y0, B=Y1, C=M0, D=A1G, E=M1,  
 F=A0G

## 8. 各命令の説明

---

### DSP演算命令

iiiiii(imm): PSHAの場合は-32 ~ +32、PSHLの場合は-16 ~ +16

ee(Se): 0=X0, 1=X1, 2=Y0, 3=A1

ff(Sf): 0=Y0, 1=Y1, 2=X0, 3=A1

xx(Sx): 0=X0, 1=X1, 2=A0, 3=A1

yy(Sy): 0=Y0, 1=Y1, 2=M0, 3=M1

gg(Dg): 0=M0, 1=M1, 2=A0, 3=A1

uu(Du): 0=X0, 1=Y0, 2=A0, 3=A1

zzzz(Dz): 5=A1, 7=A0, 8=X0, 9=X1, A=Y0, B=Y1, C=M0, E=M1

- DC ビット

更新 : 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。

: 更新されません。

#### (1) 説明

命令の動作を説明します。

#### (2) 注意

命令を使う上で特に注意が必要なことを説明します。

#### (3) 動作内容

C 言語で動作内容を表示します。

#### (4) 使用例

アセンブラモニターで例を示し、命令の実行前後の状態を表示します。

### 8.4.1 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
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+ls,Ds	(As)→MSW of Ds、0→LSW of Ds、 As + ls→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+ls	MSW of Ds→(As)、As + ls→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+ls,Ds	(As)→Ds、As + ls→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+ls	Ds→(As)、As + ls→As	111101AADDDD1111	1	-

#### (1) 説明

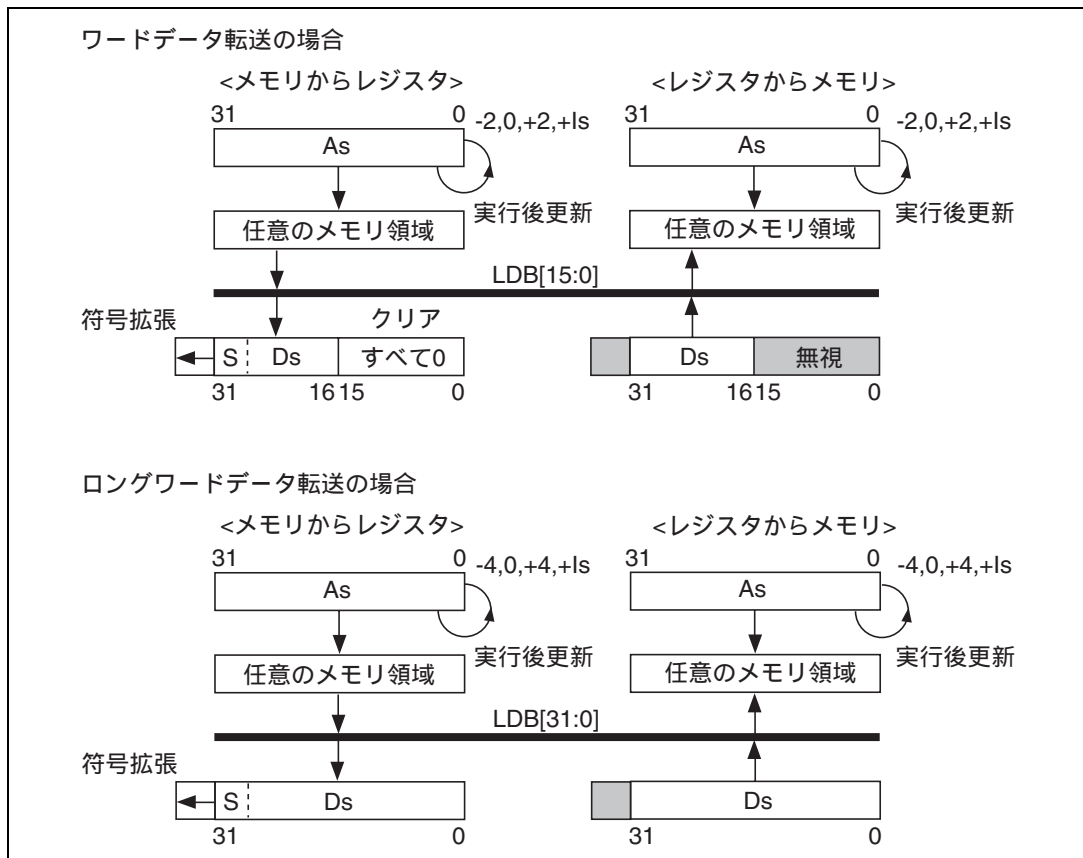
ソースオペランドのデータをデスティネーションオペランドへ転送します。メモリからレジスタへ、レジスタからメモリへ転送します。ワード長、ロングワード長を指定できます。ワード転送の場合、ソースオペランドがメモリでデスティネーションオペランドがレジスタのときは、ワードデータはレジスタの上位ワードにロードされ、下位ワードは0でクリアされます。ソースオペランドがレジスタでデスティネーションオペランドがメモリのときは、レジスタの上位ワードがワードデータとしてストアされます。ロングワード転送の場合はロングワードデータが転送されます。レジスタがデスティネーションオペランドでガードビットがある場合は、符号が拡張さ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

## 8. 各命令の説明

### (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
  
```

## 8.4.2 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

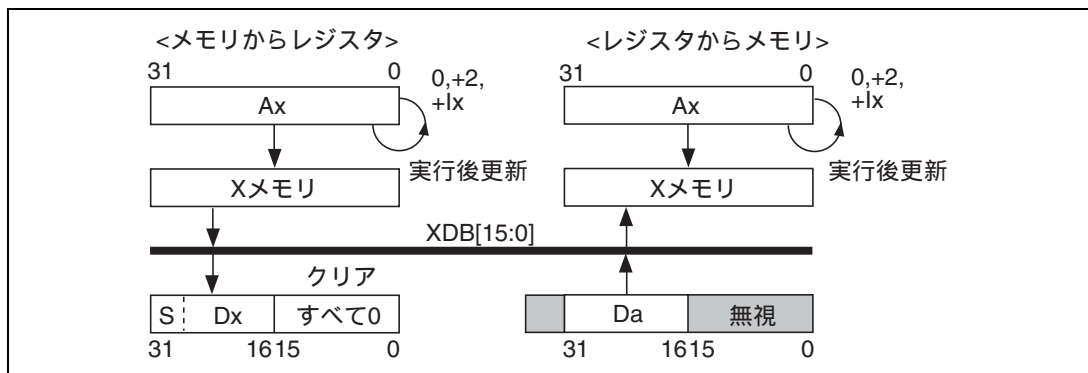
書式	動作概略	命令コード	実行 ステート	DC ビット
MOVX.W @Ax,Dx	(Ax)→MSW of Dx、0→LSW of Dx	111100A*D*0*01**	1	-
MOVX.W @Ax+,Dx	(Ax)→MSW of Dx、0→LSW of Dx、 Ax+2→Ax	111100A*D*0*10**	1	-
MOVX.W @Ax+lx,Dx	(Ax)→MSW of Dx、0→LSW of Dx、 Ax+lx→Ax	111100A*D*0*11**	1	-
MOVX.W Da,@Ax	MSW of Da→(Ax)	111100A*D*1*01**	1	-
MOVX.W Da,@Ax+	MSW of Da→(Ax)、Ax+2→Ax	111100A*D*1*10**	1	-
MOVX.W Da,@Ax+lx	MSW of Da→(Ax)、Ax+lx→Ax	111100A*D*1*11**	1	-

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メモリとワード長だけが指定できます。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場合は、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。

#### (2) 動作内容



#### (3) 使用例

```
MOVX.W @R4 + ,X0 ;実行前: R4=H'08010000, (R4)=H'5555, X0=H'12345678
                  実行後: R4=H'08010002, X0=H'55550000
```

## 8. 各命令の説明

### 8.4.3 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

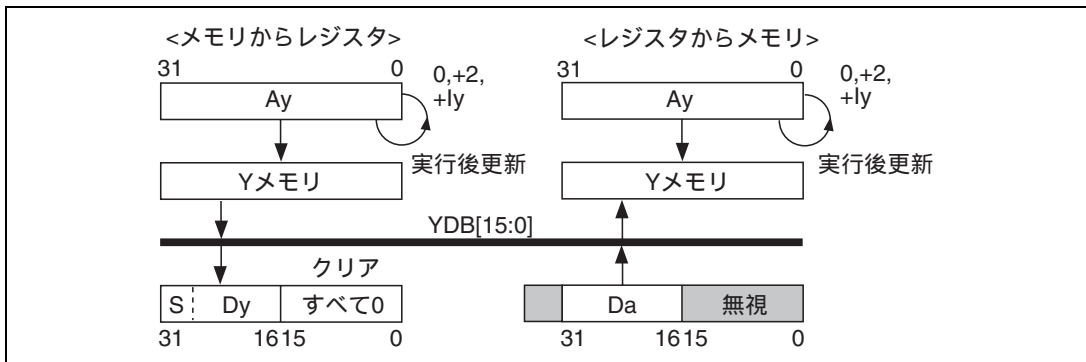
書式	動作概略	命令コード	実行 ステート	DC ビット
MOVY.W @Ay,Dy	(Ay)→MSW of Dy、0→LSW of Dy	111100*A*D*0**01	1	-
MOVY.W @Ay+,Dy	(Ay)→MSW of Dy、0→LSW of Dy、 Ay+2→Ay	111100*A*D*0**10	1	-
MOVY.W @Ay+ly,Dy	(Ay)→MSW of Dy、0→LSW of Dy、 Ay+ly→Ay	111100*A*D*0**11	1	-
MOVY.W Da,@Ay	MSW of Da→(Ay)	111100*A*D*1**01	1	-
MOVY.W Da,@Ay+	MSW of Da→(Ay)、Ay+2→Ay	111100*A*D*1**10	1	-
MOVY.W Da,@Ay+ly	MSW of Da→(Ay)、Ay+ly→Ay	111100*A*D*1**11	1	-

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。Yメモリとワード長だけが指定できます。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。

#### (2) 動作内容



#### (3) 使用例

```
MOVY.W A0,@R6+R9 ;実行前：R6=H'08020000, R9=H'00000006,
                    A0=H'123456789A
                    実行後：R6=H'08020006, (H'08020000)=H'3456
```



#### 8.4.4 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
NOPX	No Operation	1111000*0*0*00**	1	-

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。

## 8. 各命令の説明

---

### 8.4.5 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
NOPY	No Operation	111100*0*0*0**00	1	-

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。

## 8.5 DSP 演算命令の説明

DSP 演算命令の詳細説明を、アルファベット順に示します。説明の形式、記号については「8.4 DSP データ転送命令の説明」を参照してください。

表 8.2 アルファベット順 DSP 演算命令

命令	動作	命令コード	実行 ステート	DC ビット
PABS Sx,Dz	もし Sx = 0 ならば Sx→Dz もし Sx < 0 ならば 0 - Sx→Dz	111110***** 10001000xx00zzzz	1	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	1	更新
PADD Sx,Sy,Dz	Sx + Sy→Dz	111110***** 10110001xxyyzzzz	1	更新
DCT PADD Sx,Sy,Dz	もし DC = 1 ならば Sx + Sy→Dz もし 0 ならば nop.	111110***** 10110010xxyyzzzz	1	-
DCF PADD Sx,Sy,Dz	もし DC = 0 ならば Sx + Sy→Dz もし 1 ならば nop.	111110***** 10110011xxyyzzzz	1	-
PADD Sx,Sy,Du PMULS Se,Sf,Dg	Sx + Sy→Du MSW of Se × MSW of Sf→Dg	111110***** 0111eefxxyygguu	1	更新* <sup>1</sup>
PADDC Sx,Sy,Dz	Sx + Sy + DC→Dz	111110***** 10110000xxyyzzzz	1	更新
PAND Sx,Sy,Dz	Sx & Sy→Dz、LSW&ガードビット クリア	111110***** 10010101xxyyzzzz	1	更新
DCT PAND Sx,Sy,Dz	もし DC = 1 ならば Sx&Sy→Dz、 LSW&ガードビットクリア もし 0 ならば nop.	111110***** 10010110xxyyzzzz	1	-
DCF PAND Sx,Sy,Dz	もし DC = 0 ならば Sx&Sy→Dz、 LSW&ガードビットクリア もし 1 ならば nop.	111110***** 10010111xxyyzzzz	1	-
PCLR Dz	H'00000000→Dz	111110***** 100011010000zzzz	1	更新
DCT PCLR Dz	もし DC = 1 ならば H'00000000→Dz もし 0 ならば nop.	111110***** 100011100000zzzz	1	-
DCF PCLR Dz	もし DC = 0 ならば H'00000000→Dz もし 1 ならば nop.	111110***** 100011110000zzzz	1	-
PCMP Sx,Sy	Sx - Sy	111110***** 10000100xxyy0000	1	更新
PCOPY Sx,Dz	Sx→Dz	111110***** 11011001xx00zzzz	1	更新
PCOPY Sy,Dz	Sy→Dz	111110***** 1111100100yyzzzz	1	更新
DCT PCOPY Sx,Dz	もし DC = 1 ならば Sx→Dz もし 0 ならば nop.	111110***** 11011010xx00zzzz	1	-

## 8. 各命令の説明

命令	動作	命令コード	実行 状態	DCビット
DCT PCOPY S <sub>y</sub> ,D <sub>z</sub>	もし DC = 1 ならば S <sub>y</sub> →D <sub>z</sub> もし 0 ならば nop.	111110***** 1111101000yyzzzz	1	-
DCF PCOPY S <sub>x</sub> ,D <sub>z</sub>	もし DC = 0 ならば S <sub>x</sub> →D <sub>z</sub> もし 1 ならば nop.	111110***** 11011011xx00zzzz	1	-
DCF PCOPY S <sub>y</sub> ,D <sub>z</sub>	もし DC = 0 ならば S <sub>y</sub> →D <sub>z</sub> もし 1 ならば nop.	111110***** 1111101100yyzzzz	1	-
PDEC S <sub>x</sub> ,D <sub>z</sub>	MSW of S <sub>x</sub> - 1→D <sub>z</sub> 、D <sub>z</sub> の LSW クリア	111110***** 10001001xx00zzzz	1	更新
PDEC S <sub>y</sub> ,D <sub>z</sub>	MSW of S <sub>y</sub> - 1→D <sub>z</sub> 、D <sub>z</sub> の LSW クリア	111110***** 10101001xx00zzzz	1	更新
DCT PDEC S <sub>x</sub> ,D <sub>z</sub>	もし DC = 1 ならば MSW of S <sub>x</sub> - 1→D <sub>z</sub> 、D <sub>z</sub> の LSW クリア もし 0 ならば nop.	111110***** 10001010xx00zzzz	1	-
DCT PDEC S <sub>y</sub> ,D <sub>z</sub>	もし DC = 1 ならば MSW of S <sub>y</sub> - 1→D <sub>z</sub> 、D <sub>z</sub> の LSW クリア もし 0 ならば nop.	111110***** 10101010xx00zzzz	1	-
DCF PDEC S <sub>x</sub> ,D <sub>z</sub>	もし DC = 0 ならば MSW of S <sub>x</sub> - 1→D <sub>z</sub> 、D <sub>z</sub> の LSW クリア もし 1 ならば nop.	111110***** 10001011xx00zzzz	1	-
DCF PDEC S <sub>y</sub> ,D <sub>z</sub>	もし DC = 0 ならば MSW of S <sub>y</sub> - 1→D <sub>z</sub> 、D <sub>z</sub> の LSW クリア もし 1 ならば nop.	111110***** 10101011xx00zzzz	1	-
PDMSB S <sub>x</sub> ,D <sub>z</sub>	S <sub>x</sub> の内容を正規化するに必要な シフト量→MSW of D <sub>z</sub> 、D <sub>z</sub> の LSW クリア	111110***** 10011101xx00zzzz	1	更新
PDMSB S <sub>y</sub> ,D <sub>z</sub>	S <sub>y</sub> の内容を正規化するに必要な シフト量→MSW of D <sub>z</sub> 、D <sub>z</sub> の LSW クリア	111110***** 1011110100yyzzzz	1	更新
DCT PDMSB S <sub>x</sub> ,D <sub>z</sub>	もし DC = 1 ならば S <sub>x</sub> の内容を正規 化するに必要なシフト量→MSW of D <sub>z</sub> 、D <sub>z</sub> の LSW クリア もし 0 ならば nop.	111110***** 10011110xx00zzzz	1	-
DCT PDMSB S <sub>y</sub> ,D <sub>z</sub>	もし DC = 1 ならば S <sub>y</sub> の内容を正規 化するに必要なシフト量→MSW of D <sub>z</sub> 、D <sub>z</sub> の LSW クリア もし 0 ならば nop.	111110***** 1011111000yyzzzz	1	-
DCF PDMSB S <sub>x</sub> ,D <sub>z</sub>	もし DC = 0 ならば S <sub>x</sub> の内容を正規 化するに必要なシフト量→MSW of D <sub>z</sub> 、D <sub>z</sub> の LSW クリア もし 1 ならば nop.	111110***** 10011111xx00zzzz	1	-
DCF PDMSB S <sub>y</sub> ,D <sub>z</sub>	もし DC = 0 ならば S <sub>y</sub> の内容を正規 化するに必要なシフト量→MSW of D <sub>z</sub> 、D <sub>z</sub> の LSW クリア もし 1 ならば nop.	111110***** 1011111100yyzzzz	1	-

命令	動作	命令コード	実行 ステート	DCビット
PINC Sx,Dz	MSW of Sx + 1→MSW of Dz、 Dz の LSW クリア	111110***** 10011001xx00zzzz	1	更新
PINC Sy,Dz	MSW of Sy + 1→MSW of Dz、 Dz の LSW クリア	111110***** 1011100100yyzzzz	1	更新
DCT PINC Sx,Dz	もし DC = 1 ならば MSW of Sx + 1→MSW of Dz、 Dz の LSW クリア もし 0 ならば nop.	111110***** 10011010xx00zzzz	1	-
DCT PINC Sy,Dz	もし DC = 1 ならば MSW of Sy + 1→MSW of Dz、 Dz の LSW クリア もし 0 ならば nop.	111110***** 1011101000yyzzzz	1	-
DCF PINC Sx,Dz	もし DC = 0 ならば MSW of Sx + 1→MSW of Dz、 Dz の LSW クリア もし 1 ならば nop.	111110***** 10011011xx00zzzz	1	-
DCF PINC Sy,Dz	もし DC = 0 ならば MSW of Sy + 1→MSW of Dz、 Dz の LSW クリア もし 1 ならば nop.	111110***** 1011101100yyzzzz	1	-
PLDS Dz,MACH	Dz→MACH	111110***** 111011010000zzzz	1	-
PLDS Dz,MACL	Dz→MACL	111110***** 111111010000zzzz	1	-
DCT PLDS Dz,MACH	もし DC = 1 ならば Dz→MACH もし 0 ならば nop.	111110***** 111011100000zzzz	1	-
DCT PLDS Dz,MACL	もし DC = 1 ならば Dz→MACL もし 0 ならば nop.	111110***** 111111100000zzzz	1	-
DCF PLDS Dz,MACH	もし DC = 0 ならば Dz→MACH もし 1 ならば nop.	111110***** 111011110000zzzz	1	-
DCF PLDS Dz,MACL	もし DC = 0 ならば Dz→MACL もし 1 ならば nop.	111110***** 111111110000zzzz	1	-
PMULS Se,Sf,Dg	MSW of Se * MSW of Sf→Dg	111110***** 0100eeff0000gg00	1	-
PNEG Sx,Dz	0 - Sx→Dz	111110***** 11001001xx00zzzz	1	更新
PNEG Sy,Dz	0 - Sy→Dz	111110***** 1110100100yyzzzz	1	更新
DCT PNEG Sx,Dz	もし DC = 1 ならば 0 - Sx→Dz もし 0 ならば nop.	111110***** 11001010xx00zzzz	1	-
DCT PNEG Sy,Dz	もし DC = 1 ならば 0 - Sy→Dz もし 0 ならば nop.	111110***** 1110101000yyzzzz	1	-
DCF PNEG Sx,Dz	もし DC = 0 ならば 0 - Sx→Dz もし 1 ならば nop.	111110***** 11001011xx00zzzz	1	-
DCF PNEG Sy,Dz	もし DC = 0 ならば 0 - Sy→Dz もし 1 ならば nop.	111110***** 1110101100yyzzzz	1	-

## 8. 各命令の説明

命令	動作	命令コード	実行 ステート	DC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
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	-
PRND Sx,Dz	Sx + H'00008000 → Dz clear LSW of Dz	111110***** 10011000xx00zzzz	1	更新
PRND Sy,Dz	Sy + H'00008000 → Dz clear LSW of Dz	111110***** 101110000yyzzzz	1	更新
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
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***** 00010iiiiiiiizzzz	1	更新
PSHL Sx,Sy,Dz	もし Sy < 0、Sx > >   Sy   → Dz ならば Dz のガードビットと LSW クリア もし Sy = 0 ならば Sx < < Sy → Dz.	111110***** 10000001xxyyzzzz	1	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 クリア もし DC = 1 & Sy < 0 ならば Sx > >   Sy   → Dz 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 クリア もし DC = 0 & Sy < 0 ならば Sx > >   Sy   → Dz もし DC = 1 ならば nop.	111110***** 10000011xxyyzzzz	1	-

命令	動作	命令コード	実行 ステート	DC ビット
PSHL #Imm,Dz	もし Imm = 0 ならば Dz << Imm → Dz、Dz のガードビットと LSW クリア もし Imm < 0 ならば Dz >>  Imm  → Dz	111110***** 00000iiiiiiiizzzz	1	更新
PSTS MACH,Dz	MACH → Dz	111110***** 110011010000zzzz	1	-
PSTS MACL,Dz	MACL → Dz	111110***** 110111010000zzzz	1	-
DCT PSTS MACH,Dz	もし DC = 1 ならば MACH → Dz もし 0 ならば nop.	111110***** 110011100000zzzz	1	-
DCT PSTS MACL,Dz	もし DC = 1 ならば MACL → Dz もし 0 ならば nop.	111110***** 110111100000zzzz	1	-
DCF PSTS MACH,Dz	もし DC = 0 ならば MACH → Dz もし 1 ならば nop.	111110***** 110011110000zzzz	1	-
DCF PSTS MACL,Dz	もし DC = 0 ならば MACL → Dz もし 1 ならば nop.	111110***** 110111110000zzzz	1	-
PSUB Sx,Sy,Dz	Sx - Sy → Dz	111110***** 10100001xxyyzzzz	1	更新
DCT PSUB Sx,Sy,Dz	もし DC = 1 ならば Sx-Sy → Dz もし 0 ならば nop.	111110***** 10100010xxyyzzzz	1	-
DCF PSUB Sx,Sy,Dz	もし DC = 0 ならば Sx-Sy → Dz もし 1 ならば nop.	111110***** 10100011xxyyzzzz	1	-
PSUB Sx,Sy,Du PMULS Se,Sf,Dg	Sx - Sy → Du MSW of Se × MSW of Sf → Dg	111110***** 0110eefxxyygguu	1	更新*2
PSUBC Sx,Sy,Dz	Sx - Sy - DC → Dz	111110***** 10100000xxyyzzzz	1	更新
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
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	-

【注】 \*1 PADD の演算結果に基づいて更新されます。

\*2 PSUB の演算結果に基づいて更新されます。

## 8. 各命令の説明

---

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、4 つの状態を示すフラグ (V、N、Z、GT) があります。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 演算ごとの命令の動作内容の説明では、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。

### 動作内容 (1) 固定小数点ボロ-DC ビット

```
/* SH-DSP: DSP Engine: fixed_pt_dc_always_borrow.c
   Set DSR's DC Bit to borrow bit regardless the status of CS[2:0] bits */

{
  /* DC update policy: don't care the status of DSPCSBITS */
  DSPDCBIT = borrow_bit;
  DSPGTBIT = ~((negative_bit ^ overflow_bit) | zero_bit);
  DSPZBIT = zero_bit;
  DSPNBIT = negative_bit;
  DSPVBIT = overflow_bit;
}
```

### 動作内容 (2) 固定小数点キャリ DC ビット

```
/* SH-DSP: DSP Engine: fixed_pt_dc_always_carry.c
   Set DSR's DC Bit to carry bit regardless the status of CS[2:0] bits */

{
  /* DC update policy: don't care the status of DSPCSBITS */
  DSPDCBIT = carry_bit;
  DSPGTBIT = ~((negative_bit ^ overflow_bit) | zero_bit);
  DSPZBIT = zero_bit;
  DSPNBIT = negative_bit;
  DSPVBIT = overflow_bit;
}
```

### 動作内容 (3) 固定小数点負値 DC ビット

```
/* SH-DSP: DSP Engine: fixed_pt_minus_dc_bit.c
   Fixed Point Minus(-) Operation: Set DC Bit in DSR */

{
  switch (DSPCSBITS) {
    case 0x0: /* Borrow Mode */
      DSPDCBIT = borrow_bit;
      break;
  }
```



```

case 0x1:          /* Negative Value Mode */
    DSPDCBIT = negative_bit;
    break;
case 0x2:          /* Zero Value Mode */
    DSPDCBIT = zero_bit;
    break;
case 0x3:          /* Overflow Mode */
    DSPDCBIT = overflow_bit;
    break;
case 0x4:          /* Signed Greater Than Mode */
    DSPDCBIT = ~((negative_bit ^ overflow_bit) | zero_bit);
    break;
case 0x5:          /* Signed Greater Than or Equal Mode */
    DSPDCBIT = ~(negative_bit ^ overflow_bit);
    break;
case 0x6:          /* Reserved */
case 0x7:          /* Reserved */
    break;
}
DSPGTBIT = ~((negative_bit ^ overflow_bit) | zero_bit);
DSPZBIT = zero_bit;
DSPNBIT = negative_bit;
DSPVBIT = overflow_bit;
}

```

#### 動作内容 (4) 固定小数点オーバーフロー防止機能 (飽和演算)

```

/* SH-DSP: DSP Engine: Set to maximum non-overflow value if overflow
fixed_pt_overflow_protection.c */

{
    if(SBIT && overflow_bit) { /* Overflow Protection Enable & overflow */
        if(DSP_ALU_DSTG_BIT7==0) { /* positive value */
            if((DSP_ALU_DSTG_LSB8!=0x0) || (DSP_ALU_DST_MSB!=0)) {
                DSP_ALU_DSTG= 0x0;
                DSP_ALU_DST = 0x7fffffff;
            }
        }
    }
    else { /* negative value */
        if((DSP_ALU_DSTG_LSB8!=0xff) || (DSP_ALU_DST_MSB!=1)) {

```

## 8. 各命令の説明

---

```
DSP_ALU_DSTG= 0xff;
DSP_ALU_DST = 0x80000000;
    }
}
overflow_bit = 0; /* No more overflow when protected */
}
}
```

### 動作内容 (5) 固定小数点正值 DC ビット

```
/* SH-DSP: DSP Engine: fixed_pt_plus_dc_bit.c
Fixed Point Plus(+) Operation: Set DC Bit in DSR */
{
switch (DSPCSBITS) {
case 0x0:          /* Carry Mode */
    DSPDCBIT = carry_bit;
    break;
case 0x1:          /* Negative Value Mode */
    DSPDCBIT = negative_bit;
    break;
case 0x2:          /* Zero Value Mode */
    DSPDCBIT = zero_bit;
    break;
case 0x3:          /* Overflow Mode */
    DSPDCBIT = overflow_bit;
    break;
case 0x4:          /* Signed Greater Than Mode */
    DSPDCBIT = ~((negative_bit ^ overflow_bit) | zero_bit);
    break;
case 0x5:          /* Signed Greater Than or Equal Mode */
    DSPDCBIT = ~(negative_bit ^ overflow_bit);
    break;
case 0x6:          /* Reserved */
case 0x7:          /* Reserved */
    break;
}
DSPGTBIT = ~((negative_bit ^ overflow_bit) | zero_bit);
DSPZBIT = zero_bit;
DSPNBIT = negative_bit;
DSPVBIT = overflow_bit;
```

```
}

```

#### 動作内容 (6) 固定小数点演算無条件 DC ビット更新

```
/* SH-DSP: DSP Engine: Fixed Point Unconditional Update
fixed_pt_unconditional_update.c
    1. Write back to the Destination Register
    2. update negative_bit and zero_bit. */
/* negative_bit = MSB of ALU's 40-bit result.
zero_bit      = if(ALU's 40-bit result==0)
sign-extend to A0/1G[31:8] */

{
    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) 整数正值 DC ビット

```
/* SH-DSP: DSP Engine: integer_minus_dc_bit.c
Integer Minus(-) Operation: Set DC Bit in DSR */

#include "fixed_pt_minus_dc_bit.c"

```

#### 動作内容 (8) 整数オーバーフロー防止機能 (飽和演算)

```
/* SH-DSP: DSP Engine: Set to maximum non-overflow value if overflow
integer_overflow_protection.c */

#include "fixed_pt_overflow_protection.c"

```

#### 動作内容 (9) 整数正值 DC ビット

```
/* SH-DSP: DSP Engine: integer_plus_dc_bit.c

```

## 8. 各命令の説明

---

Integer Plus(+) Operation: Set DC Bit in DSR \*/

```
#include "fixed_pt_plus_dc_bit.c"
```

### 動作内容 (10) 整数無条件 DC ビット更新

```
/* SH-DSP: DSP Engine: Integer Operation Unconditional Update
integer_unconditional_update.c
    1. Write back to the Destination Register
    2. update negative_bit and zero_bit.
negative_bit = MSB of ALU's 24-bit(g-bit and hw) result.
zero_bit = if(ALU's g-bit & hw==0)
Spec 1.1: Clear ALU Integer operation's LSW. */

{
    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;
    }
    negative_bit = DSP_ALU_DSTG_BIT7;
    zero_bit = (DSP_ALU_DST_HW==0) & (DSP_ALU_DSTG_LSB8==0);
}
```

### 動作内容 (11) 論理演算 DC ビット

```
/* SH-DSP: DSP Engine: logical_dc_bit.c
Logical Operation: Set DC Bit in DSR */

{
    switch (DSPCSBITS) {
        case 0x0:          /* Carry Mode */
            DSPDCBIT = 0;
            break;
        case 0x1:          /* Negative Value Mode */
            DSPDCBIT = negative_bit;
    }
}
```

```

        break;
case 0x2:          /* Zero Value Mode */
    DSPDCBIT = zero_bit;
    break;
case 0x3:          /* Overflow Mode */
    DSPDCBIT = 0;
    break;
case 0x4:          /* Signed Greater Than Mode */
    DSPDCBIT = 0;
    break;
case 0x5:          /* Signed Greater Than or Equal Mode */
    DSPDCBIT = 0;
    break;
case 0x6:          /* Reserved */
case 0x7:          /* Reserved */
    break;
}

    DSPGTBIT = 0;
    DSPZBIT = zero_bit;
    DSPNBIT = negative_bit;
    DSPVBIT = 0;
}

```

#### 動作内容 (12) シフト演算 DC ビット

```

/* SH-DSP: DSP Engine: Shift_dc_bit.c
Shift Operation: Set DC Bit in DSR */

{
switch (DSPCSBITS) {
case 0x0:          /* Carry Mode */
    DSPDCBIT = carry_bit;
    break;
case 0x1:          /* Negative Value Mode */
    DSPDCBIT = negative_bit;
    break;
case 0x2:          /* Zero Value Mode */
    DSPDCBIT = zero_bit;
    break;
case 0x3:          /* Overflow Mode */

```

## 8. 各命令の説明

---

```
    DSPDCBIT = overflow_bit;
    break;
case 0x4:          /* Signed Greater Than Mode */
    DSPDCBIT = 0;
    break;
case 0x5:          /* Signed Greater Than or Equal Mode */
    DSPDCBIT = 0;
    break;
case 0x6:          /* Reserved */
case 0x7:          /* Reserved */
    break;
}
DSPGTBIT = 0;
DSPZBIT = zero_bit;
DSPNBIT = negative_bit;
DSPVBIT = overflow_bit;
}
```

## 8.5.1 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
PABS Sx,Dz	もし Sx ≥ 0 ならば Sx→Dz もし Sx < 0 ならば 0 - Sx→Dz	111110***** 10001000xx0zzzz	1	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***** 101010000yyzzzz	1	更新

## (1) 説明

絶対値を求めます。もし S<sub>x</sub>、S<sub>y</sub> オペランドの内容が正の値のときは S<sub>x</sub>、S<sub>y</sub> 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。

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。

## (2) 動作内容

```

{
    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 */

```

## 8. 各命令の説明

---

```
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= PLUS_OP_G_OV || !(POS_NOT_OV || NEG_NOT_OV);
#include "fixed_pt_overflow_protection.c"
#include "fixed_pt_unconditional_update.c"

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

break;
```



## (3) 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'DDDDDDD

実行後：X1=H'22222223

DC ビットは CS [ 2:0 ] の状態に従って更新。

## 8. 各命令の説明

### 8.5.2 [if cc] PADD ADDition 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
PADD Sx,Sy,Dz	$Sx + Sy \rightarrow Dz$	111110***** 10110001xxyyzzzz	1	更新
DCT PADD Sx,Sy,Dz	もし DC = 1 ならば $Sx + Sy \rightarrow Dz$ もし 0 ならば nop.	111110***** 10110010xxyyzzzz	1	-
DCF PADD Sx,Sy,Dz	もし DC = 0 ならば $Sx + Sy \rightarrow Dz$ もし 1 ならば nop.	111110***** 10110011xxyyzzzz	1	-

#### (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せん。DSR レジスタの N、Z、V、GT ビットも更新されません。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) 動作内容

```
{
  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);
DSP_ALU_DSTG_LSB8 = DSP_ALU_SRC1G_LSB8 + DSP_ALU_SRC2G_LSB8 + carry_bit;

overflow_bit= PLUS_OP_G_OV || !(POS_NOT_OV || NEG_NOT_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;

```

## 8. 各命令の説明

---

### (3) 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。

### 8.5.3 PADD PMULS ADDition & MULtiplly Signed by 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
PADD Sx,Sy,Du PMULS Se,Sf,Dg	Sx + Sy→Du MSW of Se × MSW of Sf→Dg	111110***** 0111eefxxyygguu	1	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。

#### (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

```
{
    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= PLUS_OP_G_OV || !(POS_NOT_OV || NEG_NOT_OV);
#include "../d_3operand.d/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;
    }
}
```

## 8. 各命令の説明

---

```
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 "../d_3operand.d/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。

### 8.5.4 PADDC ADDition 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
PADDC Sx,Sy,Dz	$Sx + Sy + DC \rightarrow Dz$	111110***** 10110000xxyyzzzz	1	キャリ

#### (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。

#### (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

```
{
  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;
  }
}
```

## 8. 各命令の説明

---

```
    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= PLUS_OP_G_OV || !(POS_NOT_OV || NEG_NOT_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。



### 8.5.5 [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
PAND Sx,Sy,Dz	Sx & Sy→Dz、Dz ガードビットと LSW クリア	111110***** 10010101xxyyzzzz	1	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	111110***** 10010110xxyyzzzz	1	-
DCF PAND Sx,Sy,Dz	もし 0 ならば nop. もし DC=0 ならば Sx&Sy→Dz、Dz ガードビットと LSW クリア もし 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。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

```
{
  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) {
```

## 8. 各命令の説明

---

```
    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。

## 8. 各命令の説明

### 8.5.6 [if cc] PCLR CLearR : 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
PCLR Dz	H'00000000→Dz	111110***** 100011010000zzzz	1	更新
DCT PCLR Dz	もし DC=1 ならば H'00000000→Dz	111110***** 100011100000zzzz	1	-
DCF PCLR Dz	もし 0 ならば nop. もし DC=0 ならば H'00000000→Dz もし 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。DSR レジスタの Z ビットは 1 にセットされます。N、V、GT ビットは 0 にクリアされます。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) 動作内容

```
{ /* 0 + 0 -> Dz */
    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;

#include "fixed_pt_plus_dc_bit.c"
    }
    else if(DSP_CONDITION_MATCH) { /* conditional operation and match */
        DSP_REG[ex2_dz_no] = 0x0;
    }
}

break;
```

(3) 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。

## 8. 各命令の説明

---

### 8.5.7 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
PCMP Sx,Sy	Sx - Sy	111110***** 10000100xxyy0000	1	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。

#### (2) 動作内容

```
{
  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= MINUS_OP_G_OV || !(POS_NOT_OV || NEG_NOT_OV);

#include "fixed_pt_overflow_protection.c"
#include "fixed_pt_minus_dc_bit.c"
}

break;

```

### (3) 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。

```

## 8. 各命令の説明

### 8.5.8 [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
PCOPY Sx,Dz	Sx→Dz	111110***** 11011001xx00zzzz	1	更新
PCOPY Sy,Dz	Sy→Dz	111110***** 1111100100yyzzzz	1	更新
DCT PCOPY Sx,Dz	もし DC = 1 ならば Sx→Dz もし 0 ならば nop.	111110***** 11011010xx00zzzz	1	-
DCT PCOPY Sy,Dz	もし DC = 1 ならば Sy→Dz もし 0 ならば nop.	111110***** 1111101000yyzzzz	1	-
DCF PCOPY Sx,Dz	もし DC = 0 ならば Sx→Dz もし 1 ならば nop.	111110***** 11011011xx00zzzz	1	-
DCF PCOPY Sy,Dz	もし DC = 0 ならば Sy→Dz もし 1 ならば nop	111110***** 1111101100yyzzzz	1	-

#### (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。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) 動作内容

```
{
  /* Sx + 0 -> Dz */
  if (EX2_DSP_BIT13==0) {
    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= PLUS_OP_G_OV || !(POS_NOT_OV || NEG_NOT_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;

```

## 8. 各命令の説明

---

```
        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;
```

### (3) 使用例

```
PCOPY X0,A0 NOPX NOPY
```

```
;実行前:X0=H'55555555, A0=H'FFFFFFFF
```

```
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。

### 8.5.9 [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
PDEC Sx,Dz	MSW of Sx - 1→Msw of Dz、Dz の LSW クリア	111110***** 10001001xx00zzzz	1	更新
PDEC Sy,Dz	MSW of Sy - 1→Msw of Dz、Dz の LSW クリア	111110***** 1010100100yyzzzz	1	更新
DCT PDEC Sx,Dz	もし DC = 1 ならば MSW of Sx - 1→Msw of Dz、Dz の LSW クリア	111110***** 10001010xx00zzzz	1	-
DCT PDEC Sy,Dz	もし 0 ならば nop. もし DC = 1 ならば MSW of Sy - 1→Msw of Dz、Dz の LSW クリア	111110***** 1010101000yyzzzz	1	-
DCF PDEC Sx,Dz	もし 0 ならば nop. もし DC = 0 ならば MSW of Sx - 1→Msw of Dz、Dz の LSW クリア	111110***** 10001011xx00zzzz	1	-
DCF PDEC Sy,Dz	もし 1 ならば nop. もし DC = 0 ならば MSW of Sy - 1→Msw of Dz、Dz の LSW クリア	111110***** 1010101100yyzzzz	1	-
	もし 1 ならば nop.			

#### (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。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

```
{
    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;
```

## 8. 各命令の説明

---

```
        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 = PLUS_OP_G_OV || !(POS_NOT_OV || NEG_NOT_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。

## 8. 各命令の説明

### 8.5.10 [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
PDMSB Sx,Dz	Sx の内容を正規化するのに必要なシフト量→MSW of Dz、Dz の LSW クリア	111110***** 10011101xx00zzzz	1	更新
PDMSB Sy,Dz	Sy の内容を正規化するのに必要なシフト量→MSW of Dz、Dz の LSW クリア	111110***** 1011110100yyzzzz	1	更新
DCT PDMSB Sx,Dz	もし DC = 1 ならば Sx の内容を正規化するのに必要なシフト量 →MSW of Dz、Dz の LSW クリア もし 0 ならば nop	111110***** 10011110xx00zzzz	1	-
DCT PDMSB Sy,Dz	もし DC = 1 ならば Sy の内容を正規化するのに必要なシフト量 →MSW of Dz、Dz の LSW クリア もし 0 ならば nop	111110***** 1011111000yyzzzz	1	-
DCF PDMSB Sx,Dz	もし DC = 0 ならば Sx の内容を正規化するのに必要なシフト量 →MSW of Dz、Dz の LSW クリア もし 1 ならば nop	111110***** 10011111xx00zzzz	1	-
DCF PDMSB Sy,Dz	もし DC = 0 ならば Sy の内容を正規化するのに必要なシフト量 →MSW of Dz、Dz の LSW クリア もし 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。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) 動作内容

```
{
    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; }
}
}

```

## 8. 各命令の説明

---

```
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;
#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;
```

### (3) 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。



### 8.5.11 [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
PINC Sx,Dz	MSW of Sx + 1 → MSW of Dz、Dz の LSW クリア	111110***** 10011001xx00zzzz	1	更新
PINC Sy,Dz	MSW of Sy + 1 → MSW of Dz、Dz の LSW クリア	111110***** 1011100100yyzzzz	1	更新
DCT PINC Sx,Dz	もし DC = 1 ならば MSW of Sx + 1 → MSW of Dz、Dz の LSW クリア	111110***** 10011010xx00zzzz	1	-
DCT PINC Sy,Dz	もし 0 ならば nop. もし DC = 1 ならば MSW of Sy + 1 → MSW of Dz、Dz の LSW クリア	111110***** 1011101000yyzzzz	1	-
DCF PINC Sx,Dz	もし 0 ならば MSW of Sx + 1 → MSW of Dz、Dz の LSW クリア	111110***** 10011011xx00zzzz	1	-
DCF PINC Sy,Dz	もし 1 ならば nop. もし DC = 0 ならば MSW of Sy + 1 → DzMSW of Dz、Dz の LSW クリア	111110***** 1011101100yyzzzz	1	-
	もし 1 ならば nop.			

#### (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せん。DSR レジスタの N、Z、V、GT ビットも更新されません。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

```
{
    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;
```

## 8. 各命令の説明

---

```
        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);
DSP_ALU_DSTG_LSB8 = DSP_ALU_SRC1G_LSB8 + DSP_ALU_SRC2G_LSB8 +
carry_bit;

overflow_bit= PLUS_OP_G_OV || !(POS_NOT_OV || NEG_NOT_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

<pre>PINC X0,M0 NOPX NOPY ;</pre>	<pre>実行前: X0=H'0052330F, M0=H'12345678 実行後: X0=H'0052330F, M0=H'00530000</pre>
<pre>OPINC X1,X1 NOPX NOPY ;</pre>	<pre>実行前: X1=H'FC342855 実行後: X1=H'FC350000</pre>

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。

## 8. 各命令の説明

### 8.5.12 [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
PLDS Dz,MACH	Dz→MACH	111110***** 111011010000zzzz	1	-
PLDS Dz,MACL	Dz→MACL	111110***** 111111010000zzzz	1	-
DCT PLDS Dz,MACH	もし DC = 1 ならば Dz→MACH もし 0 ならば nop.	111110***** 111011100000zzzz	1	-
DCT PLDS Dz,MACL	もし DC = 1 ならば Dz→MACL もし 0 ならば nop.	111110***** 111111100000zzzz	1	-
DCF PLDS Dz,MACH	もし DC = 0 ならば Dz→MACH もし 1 ならば nop.	111110***** 111011110000zzzz	1	-
DCF PLDS Dz,MACL	もし DC = 0 ならば Dz→MACL もし 1 ならば nop.	111110***** 111111110000zzzz	1	-

#### (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 ビットはいずれも更新されません。

#### (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 */
  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
```

## 8. 各命令の説明

---

### 8.5.13 PMULS MULtiplied Signed by 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
PMULS Se,Sf,Dg	MSW of Se * MSW of Sf→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 ビットはいずれも更新されません。

#### (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) 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'FFFFFFF0
```

## 8.5.14 [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
PNEG Sx,Dz	0 - Sx→Dz	111110***** 11001001xx00zzzz	1	更新
PNEG Sy,Dz	0 - Sy→Dz	111110***** 1110100100yyzzzz	1	更新
DCT PNEG Sx,Dz	もし DC=1 ならば 0 - Sx→Dz もし 0 ならば nop.	111110***** 11001010xx00zzzz	1	-
DCT PNEG Sy,Dz	もし DC=1 ならば 0 - Sy→Dz もし 0 ならば nop.	111110***** 1110101000yyzzzz	1	-
DCF PNEG Sx,Dz	もし DC=0 ならば 0 - Sx→Dz もし 1 ならば nop.	111110***** 11001011xx00zzzz	1	-
DCF PNEG Sy,Dz	もし DC=0 ならば 0 - Sy→Dz もし 1 ならば nop.	111110***** 1110101100yyzzzz	1	-

## (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。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) 動作内容

```
{
  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;
```

## 8. 各命令の説明

---

```
        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= MINUS_OP_G_OV || !(POS_NOT_OV || NEG_NOT_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;
```

### (3) 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'FFAAAAAAB  
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。
```

## 8.5.15 [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
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
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	111110***** 10110110xxyyzzzz	1	-
DCF POR Sx,Sy,Dz	もし 0 ならば nop. もし DC=0 ならば Sx   Sy → Dz、Dz のガードビットと LSW クリア	111110***** 10110111xxyyzzzz	1	-
	もし 1 ならば nop.			

## (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。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

```
{
  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;

```

## 8. 各命令の説明

---

### (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。

## 8.5.16 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
PRND Sx,Dz	Sx + H'00008000 → Dz clear LSW of Dz	111110***** 10011000xx00zzzz	1	更新
PRND Sy,Dz	Sy + H'00008000 → Dz clear LSW of Dz	111110***** 1011100000yyzzzz	1	更新

## (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。

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。

## (2) 動作内容

```

{
    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;
        }
    }
}

```

## 8. 各命令の説明

---

```
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= PLUS_OP_G_OV || !(POS_NOT_OV || NEG_NOT_OV);

#include "fixed_pt_overflow_protection.c"
#include "fixed_pt_unconditional_update.c"
#include "fixed_pt_plus_dc_bit.c"
}

break;
```

### (3) 使用例

```
PRND X0,M0 NOPX NOPY ; 実行前: X0=H'0052330F, M0=H'12345678
                      実行後: X0=H'0052330F, M0=H'00520000
PRND X1,X1 NOPX NOPY ; 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。
```

## 8.5.17 [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
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
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

### (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。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) 動作内容

<レジスタオペランドによる場合>

```
{
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;
```

## 8. 各命令の説明

---

```
        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)==0x1);
    }

    else { /* Right Shift 0< cnt <=32 */
        char cnt = ((~DSP_ALU_SRC2_HW & MASK003F)+1);
        if(cnt > 32) {
```



```

    printf("ΰnPSHA Sz,Sy,Dz Error! shift -%2X exceed range.ΰn",cnt);
    exit();
}
if((cnt>8) && DSP_ALU_SRC1G_BIT7) { /* MSB copy */
    DSP_ALU_DST=((DSP_ALU_SRC1>>8) | (DSP_ALU_SRC1<<
(32-8)));
    DSP_ALU_DST=(long) DSP_ALU_DST >> (cnt-8);
}
else {
    DSP_ALU_DST=((DSP_ALU_SRC1>>cnt) | (DSP_ALU_SRC1<<
(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 */
/* #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>

```
{
```

## 8. 各命令の説明

---

```
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)==0x1);
}

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 */
/* #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;

```

### (3) 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。

```

### 8.5.18 [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
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	111110***** 10000001xxyyzzzz	1	更新
DCT 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	111110***** 10000010xxyyzzzz	1	-
DCF 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***** 10000011xxyyzzzz	1	-
PSHL #Imm,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***** 00000iiiiiiizzzz	1	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。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) 動作内容

&lt;レジスタオペランドによる場合&gt;

```

{
    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);

```

## 8. 各命令の説明

---

```
        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;
#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;
    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) AOG = 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;
#include "shift_dc_bit.c"
}
}

break;

```

## 8. 各命令の説明

---

### (3) 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。



### 8.5.19 [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
PSTS MACH,Dz	MACH→Dz	111110***** 110011010000zzzz	1	-
PSTS MACL,Dz	MACL→Dz	111110***** 110111010000zzzz	1	-
DCT PSTS MACH,Dz	もし DC = 1 ならば MACH→Dz もし 0 ならば nop.	111110***** 110011100000zzzz	1	-
DCT PSTS MACL,Dz	もし DC = 1 ならば MACL→Dz もし 0 ならば nop.	111110***** 110111100000zzzz	1	-
DCF PSTS MACH,Dz	もし DC = 0 ならば MACH→Dz もし 1 ならば nop.	111110***** 110011110000zzzz	1	-
DCF PSTS MACL,Dz	もし DC = 0 ならば MACL→Dz もし 1 ならば nop.	111110***** 110111110000zzzz	1	-

#### (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 ビットはいずれも更新されません。

#### (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 */
{
  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 */

```

## 8. 各命令の説明

---

```
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
```

## 8. 各命令の説明

### 8.5.20 [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
PSUB Sx,Sy,Dz	Sx - Sy→Dz	111110***** 10100001xxyyzzzz	1	更新
DCT PSUB Sx,Sy,Dz	もし DC = 1 ならば Sx - Sy→Dz	111110***** 10100010xxyyzzzz	1	-
DCF PSUB Sx,Sy,Dz	もし 0 ならば nop. もし DC = 0 ならば Sx - Sy→Dz もし 1 ならば nop.	111110***** 10100011xxyyzzzz	1	-

#### (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、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。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) 動作内容

```
{
    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;

overflow_bit= MINUS_OP_G_OV || !(POS_NOT_OV || NEG_NOT_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;

```

## 8. 各命令の説明

---

### (3) 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。

## 8.5.21 PSUBPMULS : SUBtraction & MULtipliy Signedby 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
PSUB Sx,Sy,Du PMULS Se,Sf,Dg	Sx - Sy→Du MSW of Se × MSW of Sf→Dg	111110***** 0110eefxxyygguu	1	更新

### (1) 説明

S<sub>x</sub> オペランドの内容から S<sub>y</sub> オペランドの内容を減算し、結果を D<sub>u</sub> オペランドへ格納します。S<sub>e</sub>、S<sub>f</sub> 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 D<sub>g</sub> 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) 動作内容

```
{
    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= MINUS_OP_G_OV || !(POS_NOT_OV || NEG_NOT_OV);
#include "../d_3operand.d/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:
```

## 8. 各命令の説明

---

```
        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 "../d_3operand.d/fixed_pt_minus_dc_bit.c"
}

break;
```

### (3) 使用例

```
PSUB A0,M0,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
```



## 8.5.22 PSUBC SUBtract with Carry : DSP 算術演算命令

ポロ-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
PSUBC Sx,Sy,Dz	Sx - Sy - DC→Dz	111110***** 10100000xxyyzzzz	1	ポロ-

## (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 ビットはポロ-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。

## (2) 注意

PADDC 命令実行後の DC ビットは、CS ビットに関係なく、ポロ-フラグとして更新されます。

## (3) 動作内容

```
{
  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;
  }
}
```

## 8. 各命令の説明

---

```
    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= MINUS_OP_G_OV || !(POS_NOT_OV || NEG_NOT_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'DDDDDDE, 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'DDDDDDD, DC=1
```

## 8.5.23 [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
PXOR Sx,Sy,Dz	Sx ^ Sy → Dz、Dz のガードビットと LSW クリア	111110***** 10100101xyyzzzz	1	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	111110***** 10100110xyyzzzz	1	-
DCF PXOR Sx,Sy,Dz	もし 0 ならば nop. 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***** 10100111xyyzzzz	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。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

```
{
  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;
  }
}
```

## 8. 各命令の説明

---

```
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

```
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。



## 9. 処理状態

### 9.1 処理状態

処理状態にはリセット状態、例外処理状態、バス権解放状態、プログラム実行状態、低消費電力状態の5種類があります。各状態間の遷移を図9.1に示します。

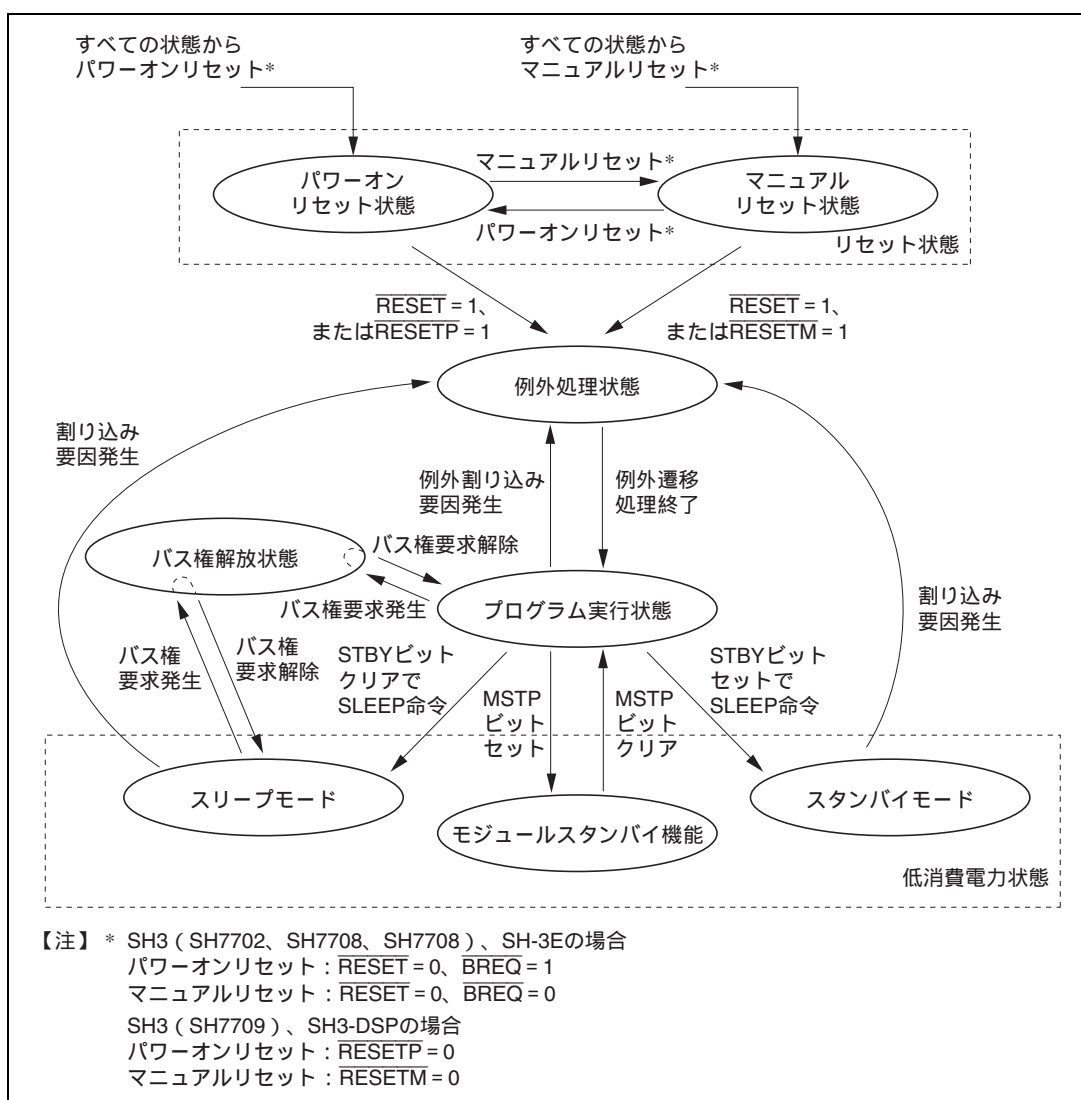


図9.1 処理状態の状態遷移図

### 9.1.1 リセット状態

CPU がリセットされている状態です。SH3 (SH7702、SH7708、SH7707)、SH-3E では、 $\overline{\text{RESET}}$  端子がローレベルになるとリセット状態になります。 $\overline{\text{BREQ}}$  端子がハイレベルのときパワーオンリセット状態になり、 $\overline{\text{BREQ}}$  端子がローレベルのとき、マニュアルリセット状態になります。SH3 (SH7709)、SH3-DSP では、 $\overline{\text{RESETP}}$  端子がローレベルのときパワーオンリセット状態、 $\overline{\text{RESETM}}$  端子がローレベルのときマニュアルリセット状態になります。

### 9.1.2 例外処理状態

リセット、一般例外、割り込みの例外処理要因によって、CPU が処理状態の流れを変えるときの過渡的な状態です。

リセットの場合は、H'A000 0000 に分岐してユーザが作成した例外処理プログラムの実行を開始します。

一般例外、割り込みの場合は、プログラムカウンタ (PC) を退避プログラムカウンタ (SPC) に、ステータスレジスタ (SR) を退避ステータスレジスタ (SSR) に退避します。ベクタベースアドレスの内容とベクタオフセットの和で求められたユーザ作成の例外サービスルーチンの開始アドレスに分岐して、プログラムの実行を開始します。

### 9.1.3 プログラム実行状態

CPU が順次プログラムを実行している状態です。

### 9.1.4 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、スタンバイモードの2つのモードおよびモジュールスタンバイ機能があります。詳細は「9.2 低消費電力状態」を参照してください。

### 9.1.5 バス権解放状態

CPU がバス権を要求したデバイスにバスを解放している状態です。



## 9.2 低消費電力状態

通常のプログラム実行状態に加えて CPU には低消費電力状態があり、この状態では CPU 動作は停止し消費電力が少なくなります (表 9.1)。低消費電力状態には、スリープモード、スタンバイモードの 2 つのモード、およびモジュールスタンバイ機能があります。

### 9.2.1 スリープモード

スタンバイコントロールレジスタ (STBCR) のスタンバイビット (STBY) を 0 にクリアして、SLEEP 命令を実行すると、CPU はスリープモードになります。スリープモードでは、CPU の動作は停止しますが、CPU、キャッシュのレジスタの内容は保持されます。内蔵周辺モジュールは動作を続けます。

スリープモードからの復帰は、リセット、割り込みによって行われ、例外処理状態を経て通常のプログラム実行状態へ遷移します。

### 9.2.2 スタンバイモード

スタンバイコントロールレジスタ (STBCR) のスタンバイビット (STBY) を 1 にセットして、SLEEP 命令を実行すると、スタンバイモードになります。スタンバイモードでは、CPU、内蔵周辺モジュール、および発振器の機能が停止します。ただし、CPU、キャッシュのレジスタの内容は保持されます。

スタンバイモードからの復帰は、リセット、割り込みにより行われます。リセットの場合は、発振安定時間経過後、例外処理状態を経て通常のプログラム実行状態へ遷移します。割り込みの場合は、WDT に設定された発振安定時間経過後、例外処理状態を経て通常のプログラム実行状態へ遷移します。

本モードでは、発振器が停止しますから、消費電力は著しく低減されます。

### 9.2.3 モジュールスタンバイ機能

タイマ (TMU)、リアルタイムクロック (RTC)、シリアルコミュニケーションインタフェース (SCI) には、モジュールスタンバイ機能があります。

スタンバイコントロールレジスタ (STBCR) のモジュールストップビットに 1 をセットすることで、それぞれ対応したモジュールへのクロック供給を停止させることができます。この機能を使用することで、通常のプログラム実行状態ならびにスリープモード時の消費電力を低減させることができます。

モジュールスタンバイ機能の内蔵周辺モジュールの各外部端子状態は、各モジュールにより異なります。TMU の外部端子は、停止前の状態を保持します。SCI の外部端子はリセット状態になります。

モジュールスタンバイ機能の解除は、MSTP ビットを 0 にクリアするか、またはリセットにより行います。

## 9. 処理状態

### 9.2.4 ハードウェアスタンバイモード

CA 端子をローレベルに設定することにより、ハードウェアスタンバイモードに遷移します。ハードウェアスタンバイモードでは、SLEEP 命令によって遷移するスタンバイモードと同様に、RTC クロックで動作するモジュール以外のすべてのモジュールが停止します。

表 9.1 低消費電力モード

モード	移行方法	LSI の状態							解除方法
		発振器	CPU	CPU レジスタ	内蔵 メモリ	内蔵周辺 モジュール	端子	外部 メモリ	
スリープ モード	STBCR の STBY ビットを“0”にクリアした状態で、SLEEP 命令を実行	動作	停止	保持	保持	動作	保持	リフレッシュ	1.割り込み 2.リセット
スタンバイ モード	STBCR の STBY ビットを“1”にセットした状態で、SLEEP 命令を実行	停止	停止	保持	保持	停止*	保持	セルフ リフレッシュ	1.割り込み 2.リセット
モジュール スタンバイ 機能	STBCR の MSTP ビットを“1”にセット	動作	動作	保持	保持	指定された モジュールが 停止	保持	リフレッシュ	1.MSTP ビットを “0”にクリア 2.リセット
ハード ウェア スタンバイ	CA 端子をローレベルにする	停止	停止	保持	保持	停止*	保持	セルフ リフレッシュ	パワーオン リセット

【注】 \* それぞれの周辺モジュールによって異なります。詳しくは SH-3、SH-3E の「ハードウェアマニュアル」を参照してください。



## 10. パイプライン動作

### 10.1.2 スロットとパイプラインの流れ

1つのステージが実行される期間をスロットと呼びます。このスロットについて以下に示すルールがあります。

- (1) 命令の各ステージ (IF、ID、EX、MA、WB) は、必ず1スロットで実行されます。1スロット内で2つ以上のステージを実行することはありません。
- (2) 1スロットには別の命令の異なるステージが最大1つずつ設定されます。1スロット内で、別の命令の同じステージが実行されることはありません。  
ありえないパイプラインの流れを図 10.2、図 10.3 に示します。

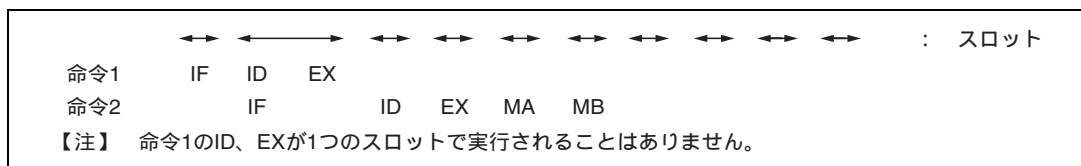


図 10.2 ありえないパイプラインの流れ (1)

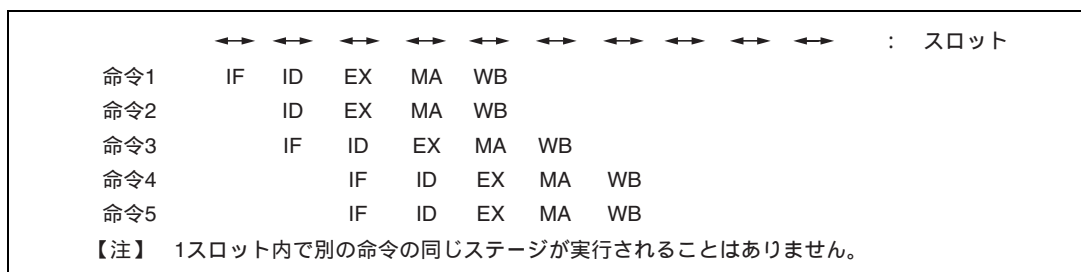


図 10.3 ありえないパイプラインの流れ (2)

### 10.1.3 1スロットの実行にかかるステート数

1スロットの実行にかかるステート数(システムクロックサイクル数)Sは次の条件で算出します。

$S = (1 \text{ スロット内に含まれる各命令のステージのうちの最長ステート数})$

すなわち、最も長いステージによって、他の短いステージを持つ命令はストールすることになります。

各ステージの実行ステート数は.....

IF: フェッチのためのメモリアクセスクロック数

ID: 常に1ステート

EX: 常に1ステート

MA: データアクセスのためのメモリアクセスクロック数

WB: 常に1ステート

たとえば、命令1、命令2のIF(命令フェッチのためのメモリアクセス)に2サイクル、命令1のMA(データアクセスのためのメモリアクセス)に3サイクル、他は1サイクルかかるとした場合のパイプラインの流れを図10.4に示します。“ ”はストールを表します。

メモリアクセスクロック数については、ハードウェアマニュアルを参照してください。

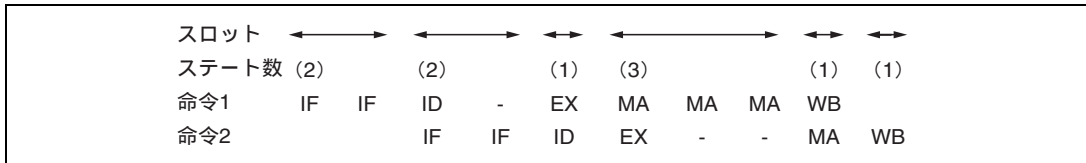


図 10.4 複数サイクルかかるスロット

### 10.1.4 命令実行ステート数

各命令の実行ステート数は、EXステージの実行間隔で数えます。命令1のEX段実行開始から、次に続く命令2のEX段開始時点までのステート数が、命令1の実行時間となります。命令実行ステート数の数え方の例を図10.5に示します。

この例のパイプラインの流れの場合、命令1と命令2のEXステージの間隔は2ステートですから、命令1の実行時間は2ステートになります。また、命令2と命令3のEXステージの間隔は3ステートですから、命令2の実行時間は3ステートになります。もし、プログラムが、命令3で終了しているなら、命令3の実行時間は、命令3の次に仮想的に命令4として、MOV Rm,Rnを置いて、命令3と命令4のEXステージの間隔から算出します。この例の場合、命令3の実行時間は2ステートになります。命令1~命令3までの実行時間は合計2+3+2=7ステートとなります。

この例では、命令1のMAと命令4のIFとが競合しています。MAとIFの競合時の動作については、「10.2.1 命令フェッチ(IF)とメモリアクセス(MA)の競合」を参照してください。

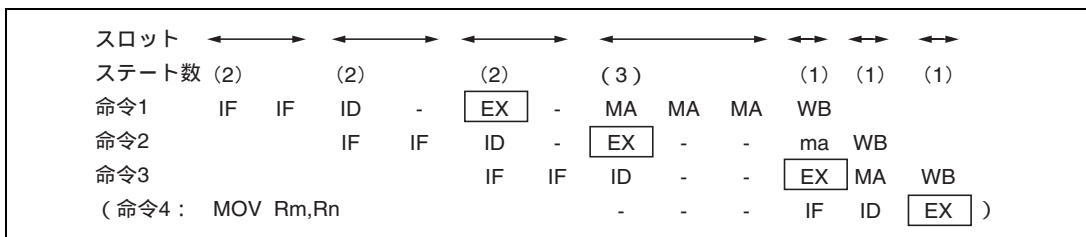


図 10.5 命令実行ステート数の数え方の例

## 10.2 競合の発生

競合は、次の7つの場合に発生します。競合が発生すると、そのステージはストールされ、次以降のスロットで実行されます。

- (1) 命令フェッチ (IF) とメモリアクセス (MA) の競合
- (2) メモリロード命令による競合
- (3) SR更新命令による競合
- (4) 乗算器アクセスの競合
- (5) FPUの競合 (SH-3Eのみ)
- (6) DSPデータ演算命令とストア命令による競合 (SH3-DSPのみ)
- (7) DSPレジスタ間転送とメモリ・ロード/ストア動作の競合 (SH3-DSPのみ)

### 10.2.1 命令フェッチ (IF) とメモリアクセス (MA) の競合

#### (1) メモリに配置された命令の位置と、IF の関係

SuperH マイコンはメモリから命令を 32 ビットでアクセスします。SH マイコンの命令はすべて 16 ビット固定長なので、基本的に IF ステージの 1 回のアクセスで 2 命令持ってくることができます。

このとき、IF で 1 回命令フェッチすると 2 命令持ってこれますので、次命令の IF では、メモリから命令をフェッチするバスサイクルは発生しません。さらにその次の命令の IF は命令を 2 つ分取り込み、またその次の命令の IF ではバスサイクルは発生しないこととなります。

すなわち、メモリに配置されている命令の内、ロングワード境界から配置されている命令 (命令アドレスの下位 2 ビットが 00 の位置:  $A1=0$ 、 $A0=0$ ) の IF が、2 命令フェッチを行います。その次の命令の IF はバスサイクルを発生しません。このバスサイクルを発生しない IF を “if” と小文字で表します。必ず if は 1 ステートです。

一方、分岐などにより、ワード境界から配置されている命令 (命令アドレスの下位 2 ビットが 10 の位置:  $A1=1$ 、 $A0=0$ ) からフェッチする場合、その IF のバスサイクルは、その命令 1 個しか取り込むことができません。したがって、その次の命令の IF はバスサイクルを発生することになりますが、その IF からは 2 命令取り込みを行います。以上の動作について、図 10.6 に示します。

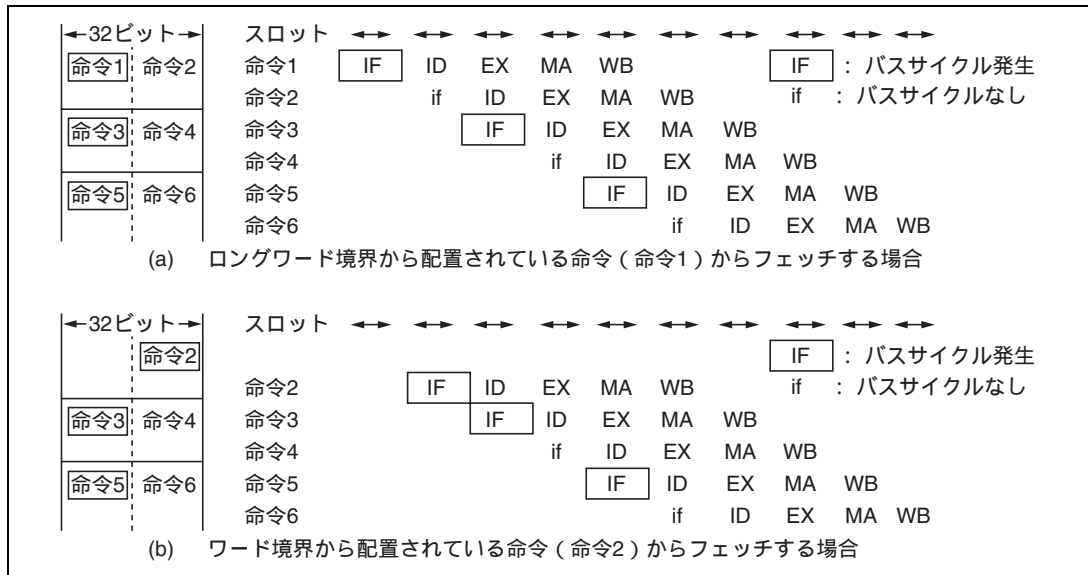


図 10.6 メモリに配置された命令の位置と IF の関係

## (2) IF と MA の競合時の基本動作

IF ステージと MA ステージはともにメモリをアクセスしていきますので、同時に動作できません。IF ステージと MA ステージが同じスロット内でメモリをアクセスしようとする、IF ステージはストールされ、次のスロットで実行されます。しかし、次のスロットで再び MA ステージと競合した場合、さらにストールされ、その次のスロットで実行されます。IF と MA の競合時の動作を図 10.7 に示します。

## 10. パイプライン動作



図 10.7 IF と MA の競合時の動作



図 10.7(a) は MA ステージが連続しない場合に、IF と MA が競合したときの動作です。スロット D で IF と MA が競合しています。この場合、IF ステージはストールされ、次のスロット E で実行されます。スロット E で ma と IF が競合しますが、ma ステージではバスサイクルが発生しないため、IF ステージのストールは発生しません。

図 10.7(b) は MA ステージが連続する場合に、IF と MA が競合したときの動作です。スロット D とスロット E に MA があり、スロット D の IF と競合しています。この場合、スロット D の ID と IF はストールされ、スロット E で実行されます。しかし、スロット E で再び IF と MA の競合が発生するため、再び IF ステージはストールされ、次のスロット F で実行されます。

### (3) メモリに配置された命令の位置と、IF、MA の競合動作との関係

命令がメモリに配置されている場合、上記(1)で示したように、バスサイクルが発生しない命令フェッチステージ(“if”と小文字で表します)が存在します。この if と MA が競合したときは、IF と MA の競合と異なり、if ステージのストールは発生しません。すなわち、if と MA は同時実行可能です。このスロットの実行にかかるステート数は MA によるメモリアクセスサイクル数になります。この様子を図 10.8 に示します。

プログラムとして、なるべく MA と IF が競合しないで、MA と if が競合するように書くと、命令実行速度が向上します。すなわち、IF、ID、EX、MA、(WB) という 4(5) 段のパイプラインを持つ命令は、メモリのロングワード境界(命令アドレスの下位 2 ビットが 00 の位置: A1=0、A0=0)から配置すると、その命令の MA ステージがその後の if と同一スロットになり、ストールが発生しないようになります。

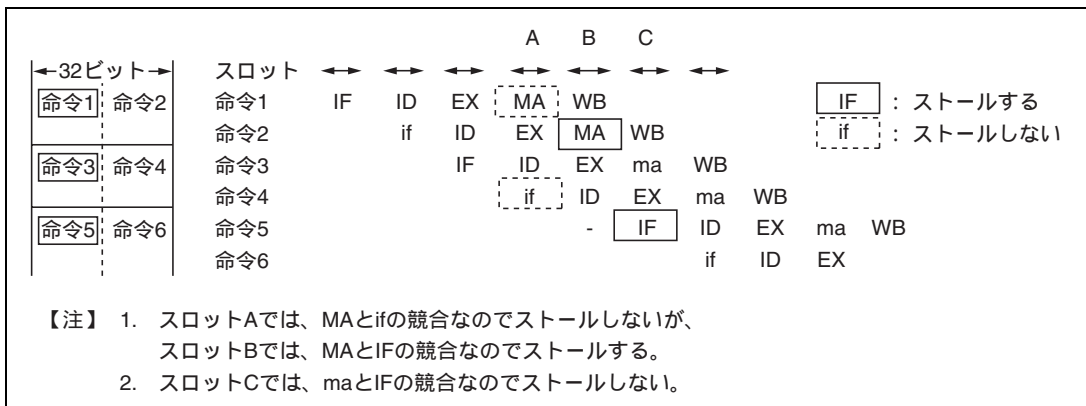


図 10.8 メモリに配置された命令の位置と、IF、MA の競合動作との関係

## 10.2.2 メモリロード命令による競合

メモリからのロード命令は、メモリへのデータアクセスがパイプラインの MA ステージで行われます。そのロード命令(ロード命令 1 とします)と、その直後の命令(命令 2 とします)に着目すると、ロード命令 1 の MA ステージが終了する前に、命令 2 の EX ステージが始まることとなります。

このとき、ロード命令 1 でメモリロードしたデータを命令 2 が使おうとすると、まだそのデータが準備できていないので、命令 2 の EX ステージはストールされます。

ただし、命令 2 が MAC @Rm+,@Rn+で、Rm とロード命令 1 のデスティネーションが同じだった場合はストールしません。

## 10. パイプライン動作

ストールが発生したときのパイプラインの様子を図 10.9 に示します。

したがって、ロード命令の直後に、その結果を使う命令を配置するようなプログラムを書くと、実行速度が低下します。ロード命令の結果を使う命令は、ロード命令の 2 命令以後に置くと速度が低下しません。

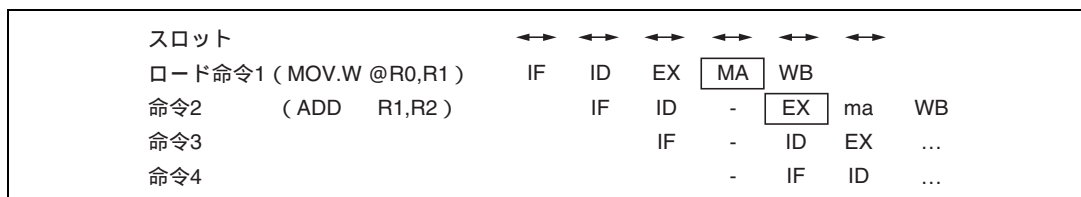


図 10.9 メモリロード命令によるパイプラインへの影響

### 10.2.3 SR 更新命令による競合

ステータスレジスタ (SR) の M、Q、S、および T ビットを書き換える命令 (SR 更新命令) は、SR の書き換えをパイプラインの WB ステージで行います。この命令の直後に SR を読み出す命令 (命令 2) が来ると、まだそのデータが準備できていないので、命令 2 の EX ステージは SR の書き換えが終了するまでストールされます。ただし、SR の全ビットを書き換える LDC Rm,SR や LDC.L@Rm+,SR や RTE では、競合によるストールは発生しません。また、SR を読み出す命令には、STC SR,Rn と STC.L SR,@-Rn と TRAPA があります。ストールが発生したときのパイプラインの様子を図 10.10 に示します。

このように、SR 更新命令の直後に、SR を読み出す命令を配置するようなプログラムを書くと実行速度が低下します。SR を読み出す命令は、SR 更新命令の 3 命令以後に置くと速度が低下しません。

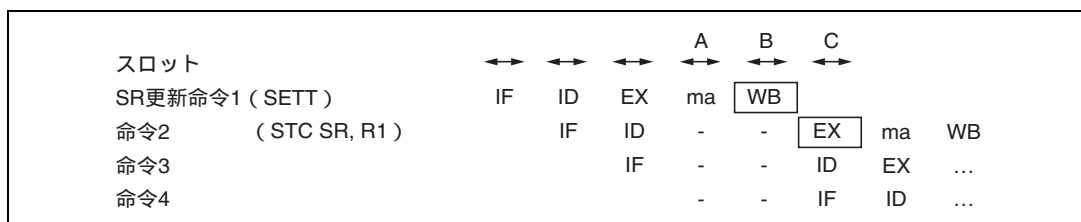


図 10.10 SR 更新命令によるパイプラインへの影響

### 10.2.4 乗算器アクセスによる競合

乗算系命令 (積和命令、乗算命令)、積和レジスタ (MACH、MACL) をアクセスする命令は、乗算器アクセスで競合が発生します。

乗算系命令では、乗算器は最後の MA 終了後スロットに関係なく倍精度 (64 ビット) 系乗算命令と積和演算命令では 3 ステートの間動作し、単精度 (32 ビット) 系乗算命令では 2 ステートの間動作します。

乗算系命令 (積和命令、乗算命令) の MA (2 つあるときは最初の MA) がその前の乗算系命令の乗算器アクセス (mm) と競合した場合には、その MA のパスサイクルは mm が終了するまで引き伸ばされ、その引き伸ばされた MA は一つのスロットになります。

積和レジスタ (MACH、MACL) をアクセスする命令の MA は乗算器をアクセスするので、乗算系

命令と同様に、その MA のバスサイクルは前の乗算系命令の mm が終了するまで引き伸ばされ、その引き伸ばされた MA は一つのスロットになります。特に、積和レジスタ (MACH、MACL) を読み出す命令 (STS、STS.L) は mm が終了してから 1 ステート経過するまで引き伸ばされ、その引き伸ばされた MA は一つのスロットになります。

また、MA が 2 つある命令の次命令の ID は 1 スロット分後ろにストールされます。

乗算系命令、積和レジスタアクセス命令は、MA を持っているので、IF との競合も発生します。

乗算器アクセスの競合の例を図 10.11、図 10.12 に示します。ここでは MA と IF の競合は考慮していません。



図 10.11 乗算器アクセスの競合の例 (1) (MAC.L 命令と MAC.L 命令の競合)

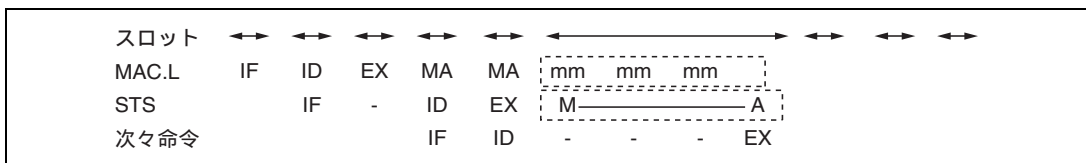


図 10.12 乗算器アクセスの競合の例 (2) (MAC.L 命令と STS 命令の競合)

### 10.2.5 FPU の競合 (SH-3E のみ)

LDS および STS 命令による CPU と FPU 間のデータ転送に加え、浮動小数点数のロードおよびストアでも CPU パイプラインの MA ステージを使用します。したがって、これらの命令は、IF ステージと競合します。

浮動小数点数算術演算命令、FMOV 命令、または浮動小数点数ロード命令の結果を格納するレジスタが、次に続く命令によってリードされた場合、その命令 (次に続く命令) は 1 スロット期間分実行が遅延します (図 10.13)。

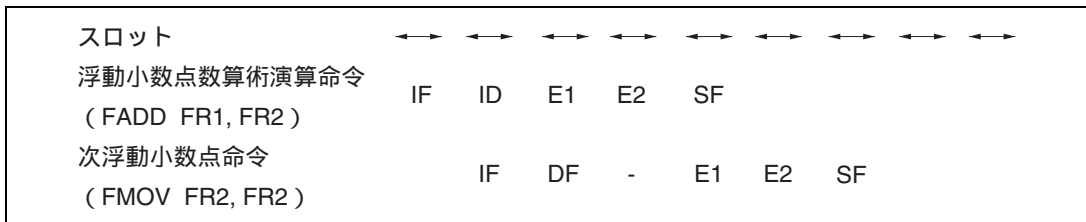


図 10.13 FPU による競合 1

LDS または LDS.L 命令を使用して、FPSCR の値を変更すると次に続く命令 (その命令が浮動小数点数算術演算命令ならば) は、1 スロット期間分実行が遅延します (図 10.14)。

## 10. パイプライン動作

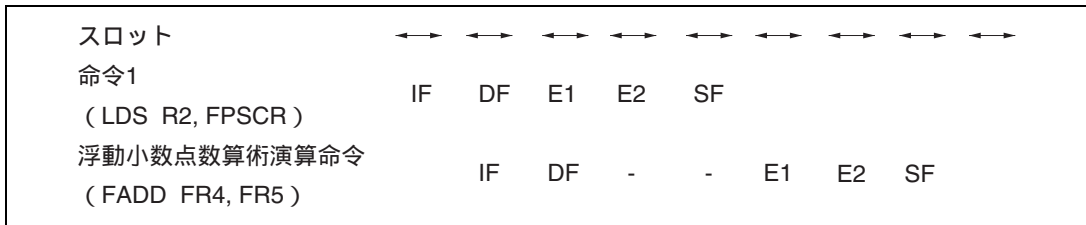


図 10.14 FPU による競合 2

先立つ命令が浮動小数点数算術演算命令のとき、(STS または STS.L 命令を使用して) FPSCR の値をリードすると、1 スロット期間分実行が遅延します (図 10.15)。

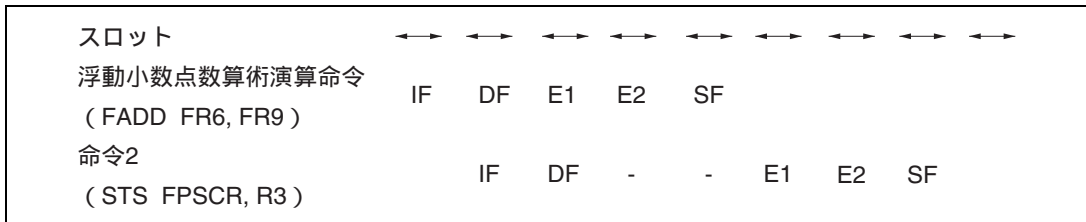


図 10.15 FPU による競合 3

FDIV および FSQRT 命令は E1 ステージに 13 サイクルを要します。この期間中は、他の浮動小数点命令が E1 ステージに入ることはありません。FDIV および FSQRT 命令が E1 ステージを終了する前に、他の浮動小数点命令が出現した場合、その浮動小数点命令は所定のスロット期間実行を遅延し、FDIV および FSQRT 命令が E2 ステージに移行した後に E1 ステージに入ります (図 10.16)。

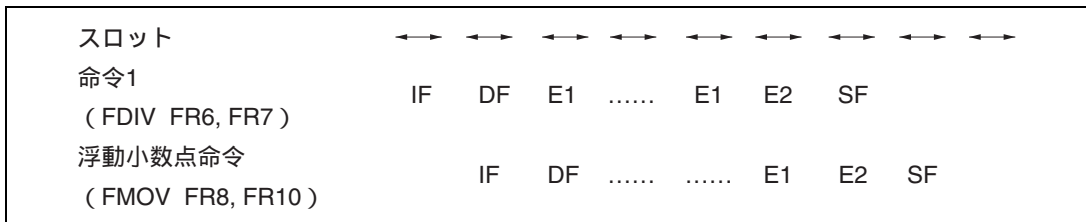


図 10.16 FPU による競合 4

ただし、前の FDIV もしくは FSQRT 命令と次の FPU 演算が同一のレジスタを使用して競合が発生する場合には以下のように FDIV および FSQRT 命令が SF 命令実行の次のサイクルで E1 ステージに入ります。

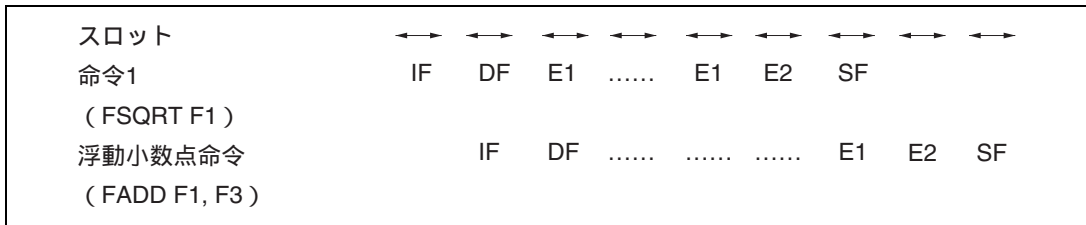


図 10.17 FPU による競合 5

### 10.2.6 DSP データ演算命令とストア命令の競合 (SH3-DSP のみ)

DSP ユニットで DSP 演算を実行し、その結果を次の命令でメモリにストアするとき、メモリロード命令と同じ競合が発生します。このとき、先行する命令の WB/DSP ステージのデータ演算が終了するまで、後続の命令の MA ステージのデータストアは引き延ばされます。ただし、CPU コアでは演算が EX ステージで実行されるのでストールサイクルは発生しません。DSP ユニットのデータ演算命令とストア命令の関係を図 10.18 に示し、CPU コアの関係を図 10.19 に示します。

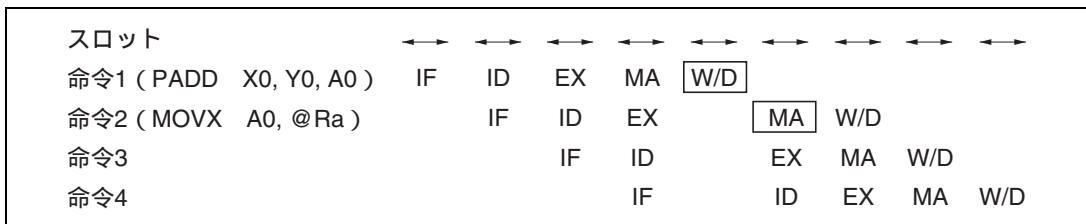


図 10.18 DSP ユニットでの演算命令とストア命令の関係

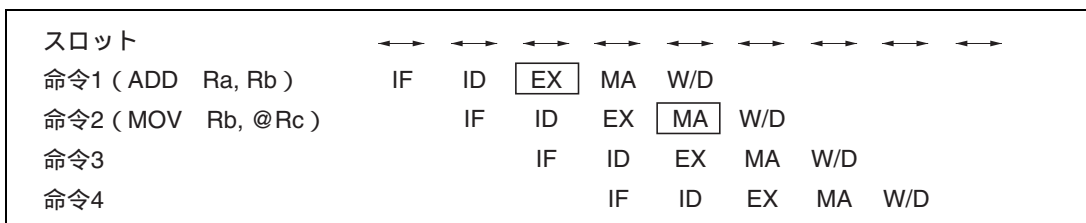


図 10.19 CPU コアでの演算命令とストア命令の関係

### 10.2.7 DSP レジスタ間転送とメモリ・ロード/ストア動作の競合 (SH3-DSP のみ)

DSP ユニットには、従来の SuperH マイコンの乗算/積和演算命令の実行に使用されるレジスタ (MACH、MACL) と、DSP 演算命令で使用されるレジスタ (A0、A1、M0、M1、X0、X1、Y0、Y1) との間でデータ転送を行うための命令 (STS.L、LDS.L) があります。この命令と並列に MOVX.W でメモリロードを行うと、競合が発生します。また、この命令の直後に MOVX.W、MOV.S.W または MOV.S.L でメモリストアを行うと、競合が発生します。

この命令は定義上、他の DSP 演算命令と同じ命令コード領域に割り付けられているので、ダブルデータ転送命令 (MOVX.W、MOVY.W) と並列に記述することができます。しかし動作のためのハードウェアを MOVX.W と共用しているため、STS.L または LDS.L と MOVX.W (ロード) を並列記述すると、WB/DSP ステージで競合が生じます。この場合、まず STS.L または LDS.L が先に実行され、MOVX.W (ロード) は 1 スロット分後ろにストールされます。

MOV.S.W、MOV.S.L も MOVX.W と同じハードウェアを使用しますが、DSP 演算命令と並列に記述できないため、先述の競合はあり得ません。しかしこの命令の直後に MOVX.W、MOV.S.W または MOV.S.L でストア動作を実行する場合は、SYS.L または LDS.L の WB/DSP ステージとストア動作の MA ステージとに競合が発生します。

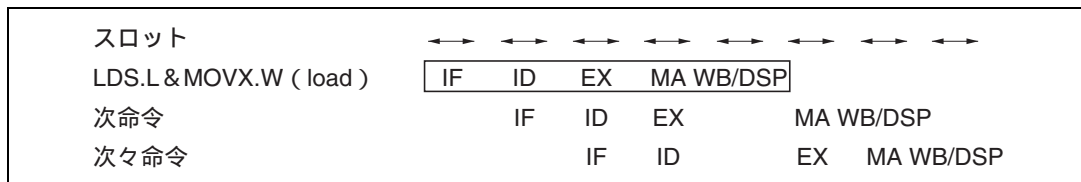


図 10.20 DSP レジスタ間転送とメモリロード並列動作の競合

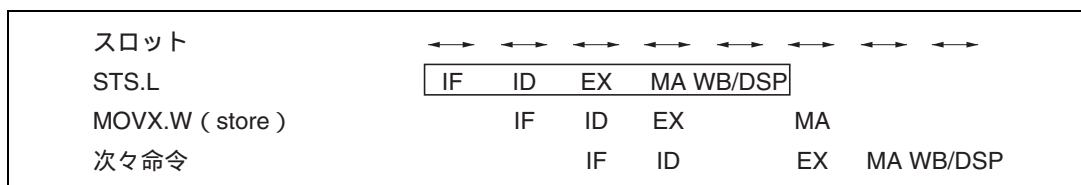


図 10.21 DSP レジスタ間転送直後のメモリストア動作の競合

## 10.3 プログラミングの指針

### 10.3.1 競合の種類と命令との対応

競合の種類と命令との対応をまとめると次のようになります。

- (1) 競合が発生しない命令
- (2) メモリアクセス (MA) があり、命令フェッチ (IF) と競合する命令
- (3) SRに対するライトバック (WB) があり、SR更新による競合を起こすことがある命令
- (4) メモリアクセス (MA) があり命令フェッチ (IF) と競合し、その上メモリのライトバック (WB) がありメモリロードの競合を起こすことがある命令
- (5) メモリアクセス (MA) があり命令フェッチ (IF) と競合し、その上SRに対するライトバック (WB) がありSR更新による競合を起こすことがある命令
- (6) メモリアクセス (MA) があり命令フェッチ (IF) と競合し、その上乘算器をアクセス (mm) し乗算器の競合を起こすことがある命令
- (7) メモリアクセス (MA) があり命令フェッチ (IF) と競合し、乗算器をアクセス (mm) し乗算器の競合を起こすことがあり、その上ライトバック (WB) がありメモリロードの競合を起こすことがある命令
- (8) MOVX.W、MOVS.WあるいはMOVS.L命令との競合を起こすことがある命令

## 10. パイプライン動作

競合の種類と命令との対応を表 10.1 に示します。

表 10.1 競合の種類と命令との対応

競合	実行 ステート	ステージ 段数	命令
競合しません	1	5	レジスタ間転送命令
	1	5	レジスタ間算術演算命令(乗算系命令を除く)
	1	5	レジスタ間論理演算命令
	1	5	シフト命令
	3/1	3	条件分岐命令
	2/1	3	遅延付き条件分岐命令
	2	3	無条件分岐命令
	2	5	無条件分岐命令 (PR)
	1	5	システム制御命令
	1	3	NOP 命令
	5	5	LDC 命令 (SR)
	7	7	LDC.L 命令 (SR)
	4	5	RTE 命令
	6	6	TRAP 命令
	4	6	SLEEP 命令
• MA は IF と競合します	1	4	メモリストア命令
	1	5	メモリストア命令 (プリデクリメント)
	1	4	キャッシュ命令
	3	6	メモリ論理演算命令
	1	4	LDTLB 命令
	1	5	STS.L 命令 (PR)
	1	5	STC.L 命令 (除くバンクレジスタ)
	2	6	STC.L 命令 (バンクレジスタ)
• DSP 演算との競合を起こします	1	4	MOVX.W (ストア)、 MOVY.W (ストア) 命令
	1	5	SR 更新レジスタ間算術演算命令 (乗算系命令を除く)
• SR 更新による競合を起こします	1	5	SR 更新レジスタ間論理演算命令
	1	5	SR 更新シフト命令
	1	5	SR 更新システム制御命令
	1	5	メモリロード命令
• MA は IF と競合します • メモリロードの競合を起こします	1	5	LDS.L 命令 (PR)
	1	5	LDC.L 命令



競合	実行 ステート	ステージ 段数	命令
<ul style="list-style-type: none"> <li>MA は IF と競合します</li> <li>SR 更新による競合を起こします</li> </ul>	3	7	SR 更新メモリ論理演算命令
	3	7	TAS 命令
<ul style="list-style-type: none"> <li>MA は IF と競合します</li> <li>乗算器との競合を起こします</li> </ul>	2 (~5) *	8	積和命令
	2 (~5) *	8	倍精度積和命令
	1 (~3) *	6	乗算命令 (PMULS を除く)
	2 (~5) *	8	倍精度乗算命令
	1	4	レジスタ→MAC 転送命令
	1	4	メモリ→MAC 転送命令
	1	5	MAC→メモリ転送命令
<ul style="list-style-type: none"> <li>MA は IF と競合します</li> <li>DSP 演算との競合を起こします</li> </ul>	1	4	MOVX.W (ストア)、 MOVS.L (ストア) 命令
<ul style="list-style-type: none"> <li>MA は IF と競合します</li> <li>メモリロードの競合を起こします</li> <li>乗算器との競合を起こします</li> <li>DSP 演算との競合を起こします</li> </ul>	1	5	MAC/DSP→レジスタ転送命令
<ul style="list-style-type: none"> <li>MOVX.W、MOVS.W あるいは MOVS.L 命令との競合を起こします</li> </ul>	1	5	PLDS、PSTS 命令

【注】 \* 通常実行ステートを示します。( )内の値は、前後の命令との競合関係による実行ステートです。

### 10.3.2 命令実行速度の向上

プログラムを作る場合は、競合ができるだけ発生しないようにすると、命令実行速度が向上します。競合を発生しないよう次のようにプログラムを作ります。

- (1) メモリアクセス (MA) を持つ命令を、メモリのロングワード境界 (命令アドレスの下位2ビットが00の位置: A1=0, A0=0) から配置します。これにより、MAと命令フェッチ (IF) との競合が発生しないようにします。
- (2) メモリからのロード命令の直後の命令には、ロード命令のデスティネーションレジスタと同じレジスタ使わない命令を配置します。これによりライトバック (WB) によるメモリロードの競合を発生しないようにします。
- (3) SRのM、Q、SおよびTビットを書き換える命令の直後およびその次には、SRを読み出さない命令を配置します。これにより、SR更新命令による競合を発生しないようにします。
- (4) 乗算器を使う命令が連続しないように配置します (PMULS命令を除く)。これにより乗算器アクセス (mm) による乗算器の競合が発生しないようになります。
- (5) DSPユニットでのデータ演算直後に演算結果を格納したレジスタの内容を、メモリあるいはCPUコアのレジスタへデータ転送をしないようにします。何か別の命令を間にはさむことにより、競合が発生しないようになります。
- (6) DSPユニットでのPLDS/PSTS命令直後にMOVX.W、MOV.S.WあるいはMOV.S.Lによるメモリストアを行わないようにします。また、PLDS/PSTS命令とMOVX.Wによるメモリストア命令をパラレルに記述しないようにします。

### 10.3.3 ステート数

基本命令は1命令を1ステートで実行するように設計されています。1ステート命令の中には、競合の発生しない命令と競合の発生する命令があります。

競合が発生しなくとも1命令が2ステート以上になる命令もあります。分岐命令などで分岐先アドレスを変更する命令、メモリ論理演算命令やいくつかのシステム制御命令のようにメモリアクセスを2回以上実行する命令、および乗算命令、積和命令のようにメモリアクセスと乗算器アクセスを持つ命令は、2ステート以上になります。

1命令が2ステート以上になる命令にも、競合の発生しない命令と競合の発生する命令があります。

効率の良いプログラムを作るためには、競合を避けて命令実行速度が向上させると同時に、ステート数の少ない命令を使う配慮が必要です。

## 10.4 各命令のパイプラインの動作

各命令のパイプラインの動作を説明します。これに、前述のルールを併せることで、プログラムのパイプラインの流れ方、および命令実行ステート数を算出することができます。

後述のパイプラインの図において、“命令 A”とあるのが、説明しようとしている命令です。また、命令フェッチステージの IF と if は区別せずに IF と書いてあります。この IF が MA と競合するとストールが発生しますが、そのストールの状況については、特別な場合を除き、後述の図では表していません。スロットがストールすると判断された場合は、「10.2 競合の発生」のルールに基づいてパイプラインの動作を考慮してください。

命令の実行ステート数とステージ段数を以下の様式で表示します。

命令の実行ステート数とステージ段数の表様式

分類	区分	命令	実行ステート	ステージ段数	競合
機能別の分類です。	命令を動作の違いで区分しています。	対応する命令を二モーニックで表示します。	競合がない場合の実行ステート数です。	命令のステージの段数です。	発生する競合を示します。

## 10. パイプライン動作

表 10.2 命令のステージ段数と実行ステート数

分類	区分	命令	実行ステート	ステージ段数	競合
データ転送命令	レジスタ - レジスタ間転送命令	MOV #imm,Rn	1	5	
		MOV Rm,Rn			
		MOVA @(disp,PC),R0			
		MOVT Rn			
		SWAP.B Rm,Rn			
		SWAP.W Rm,Rn			
		XTRCT Rm,Rn			
メモリロード命令		MOV.W @(disp,PC),Rn	1	5	<ul style="list-style-type: none"> <li>この命令の直後に、この命令のデスティネーションレジスタを使う命令を置くと、競合します。</li> <li>MA は IF と競合します。</li> </ul>
		MOV.L @(disp,PC),Rn			
		MOV.B @Rm,Rn			
		MOV.W @Rm,Rn			
		MOV.L @Rm,Rn			
		MOV.B @Rm+,Rn			
		MOV.W @Rm+,Rn			
		MOV.L @Rm+,Rn			
		MOV.B @(disp,Rm),R0			
		MOV.W @(disp,Rm),R0			
		MOV.L @(disp,Rm),Rn			
		MOV.B @(R0,Rm),Rn			
		MOV.W @(R0,Rm),Rn			
		MOV.L @(R0,Rm),Rn			
		MOV.B @(disp,GBR),R0			
		MOV.W @(disp,GBR),R0			
		MOV.L @(disp,GBR),R0			
メモリストア命令		MOV.B Rm,@Rn	1	4	<ul style="list-style-type: none"> <li>MA は IF と競合します。</li> </ul>
		MOV.W Rm,@Rn			
		MOV.L Rm,@Rn			
		MOV.B R0,@(disp,Rn)			
		MOV.W R0,@(disp,Rn)			
		MOV.L Rm,@(disp,Rn)			
		MOV.B Rm,@(R0,Rn)			
		MOV.W Rm,@(R0,Rn)			
		MOV.L Rm,@(R0,Rn)			
		MOV.B R0,@(disp,GBR)			
		MOV.W R0,@(disp,GBR)			
MOV.L R0,@(disp,GBR)					
メモリストア命令 (プリデクリメント)		MOV.B Rm,@-Rn	1	5	<ul style="list-style-type: none"> <li>MA は IF と競合します。</li> </ul>
		MOV.W Rm,@-Rn			
		MOV.L Rm,@-Rn			
キャッシュ命令		PREF @Rn	1/2* <sup>1</sup>	4	<ul style="list-style-type: none"> <li>MA は IF と競合します。</li> </ul>

分類	区分	命令	実行 ステート	ステージ 段数	競合	
算術 演算命令	レジスタ間 算術演算 命令 (乗算系命 令を除く)	ADD Rm,Rn	1	5		
		ADD #imm,Rn				
		EXTS.B Rm,Rn				
		EXTS.W Rm,Rn				
		EXTU.B Rm,Rn				
		EXTU.W Rm,Rn				
		NEG Rm,Rn				
	SUB Rm,Rn					
	SR 更新 レジスタ間 算術演算 命令 (乗算系命 令を除く)	ADDC Rm,Rn	1	5	<ul style="list-style-type: none"> <li>この命令の直後あるいはその次に SR を読み出す命令を置くと競合します。</li> </ul>	
		ADDV Rm,Rn				
CMP/EQ #imm,R0						
CMP/EQ Rm,Rn						
CMP/HS Rm,Rn						
CMP/GE Rm,Rn						
CMP/HI Rm,Rn						
CMP/GT Rm,Rn						
CMP/PL Rn						
CMP/PZ Rn						
CMP/STR Rm,Rn						
DIV1 Rm,Rn						
DIV0S Rm,Rn						
DIV0U Rn						
DT Rm,Rn						
NEGC Rm,Rn						
SUBC Rm,Rn						
SUBV Rm,Rn						
積和命令	MAC.W @Rm+,@Rn+	2(～5) <sup>*2</sup>	8	<ul style="list-style-type: none"> <li>乗算器との競合を起こします。</li> <li>MA は IF と競合します。</li> </ul>		
	MAC.L @Rm+,@Rn+	2(～5) <sup>*2</sup>				
	MULS.W Rm,Rn	1(～3) <sup>*2</sup>			6	<ul style="list-style-type: none"> <li>乗算器との競合を起こします。</li> <li>MA は IF と競合します。</li> </ul>
	MULU.W Rm,Rn					
倍精度 積和命令	DMULS.L Rm,Rn	2(～5) <sup>*2</sup>	8	<ul style="list-style-type: none"> <li>乗算器との競合を起こします。</li> <li>MA は IF と競合します。</li> </ul>		
	DMULU.L Rm,Rn					
論理 演算命令	レジスタ間 論理演算 命令	AND Rm,Rn	1	5		
		AND #imm,R0				
		NOT Rm,Rn				
		OR Rm,Rn				
		OR #imm,R0				
		XOR Rm,Rn				
		XOR #imm,R0				
	SR 更新 レジスタ間 論理演算 命令	TST Rm,Rn	1	5	<ul style="list-style-type: none"> <li>この命令の直後あるいはその次に SR を読み出す命令を置くと競合します。</li> </ul>	
		TST #imm,R0				

## 10. パイプライン動作

分類	区分	命令	実行 ステート	ステージ 段数	競合
論理 演算命令	メモリ 論理演算 命令	AND.B #imm,@(R0,GBR) OR.B #imm,@(R0,GBR) XOR.B #imm,@(R0,GBR)	3	6	<ul style="list-style-type: none"> <li>MA は IF と競合します。</li> </ul>
	SR 更新 メモリ論理 演算命令	TST.B #imm,@(R0,GBR)	3	7	<ul style="list-style-type: none"> <li>この命令の直後あるいはその次に SR を読み出す命令を置くと競合します。</li> <li>MA は IF と競合します。</li> </ul>
	TAS 命令	TAS.B @Rn	3/4* <sup>3</sup>	7	<ul style="list-style-type: none"> <li>MA は IF と競合します。</li> </ul>
シフト 命令	シフト命令	SHLL2 Rn SHLR2 Rn SHLL8 Rn SHLR8 Rn SHLL16 Rn SHLR16 Rn SHAD Rm,Rn SHLD Rm,Rn	1	5	
	SR 更新 シフト命令	ROTL Rn ROTR Rn ROTCL Rn ROTCR Rn SHAL Rn SHAR Rn SHLL Rn SHLR Rn	1	5	<ul style="list-style-type: none"> <li>この命令の直後あるいはその次に SR を読み出す命令を置くと競合します。</li> </ul>
分岐命令	条件分岐 命令	BF lavel BT lavel	3/1 * <sup>4</sup>	3	
	遅延付き条 件分岐命令	BF/S lavel BT/S lavel	2/1 * <sup>4</sup>	3	
	無条件 分岐命令	BRA label BRAf Rm JMP @Rm RTS	2	3	
	無条件 分岐命令 (PR)	BSR lavel BSRF Rm JSR @Rm	2	5	
システム 制御命令	システム 制御	LDC Rm,GBR LDC Rm,VBR LDC Rm,SSR LDC Rm,SPC	1/3* <sup>5</sup>	5	
		LDC Rm,MOD LDC Rm,RE LDC Rm,RS	3	5	

分類	区分	命令	実行 ステート	ステージ 段数	競合
システム 制御命令	システム 制御	LDC Rm,R0_BANK	1/3* <sup>5</sup>	5	
		LDC Rm,R1_BANK			
		LDC Rm,R2_BANK			
		LDC Rm,R3_BANK			
		LDC Rm,R4_BANK			
		LDC Rm,R5_BANK			
		LDC Rm,R6_BANK			
		LDC Rm,R7_BANK			
	SETRC Rm	3	5		
	SETRC #imm				
	LDRE @(disp, PC)				
	LDRS @(disp, PC)				
LDS Rm,PR	1	5			
STC SR,Rn					
STC GBR,Rn					
STC VBR,Rn					
STC SSR, Rn					
STC SPC,Rn					
STC MOD,Rn					
STC RE,Rn					
STC RS,Rn					
STC R0_BANK,Rn					
STC R1_BANK,Rn					
STC R2_BANK,Rn					
STC R3_BANK,Rn					
STC R4_BANK,Rn					
STC R5_BANK,Rn					
STC R6_BANK,Rn					
STC R7_BANK,Rn					
STS PR,Rn					
SR 更新 システム 制御命令	CLRS CLRT SETS SETT	1	5	<ul style="list-style-type: none"> <li>この命令の直後あるいはその次に SR を読み出す命令を置くと競合します。</li> </ul>	
LDTLB 命令	LDTLB	1	4	<ul style="list-style-type: none"> <li>MA は IF と競合します。</li> </ul>	
NOP 命令	NOP	1	3		
LDC 命令 (SR)	LDC Rm,SR	5	5		
LDC.L 命令 (SR)	LDC.L @Rm+,SR	7	7		
LDS.L 命令 (PR)	LDS.L @Rm+,PR	1	5	<ul style="list-style-type: none"> <li>この命令の直後に、この命令のデスティネーションレジスタを使う命令を置くと、競合します。</li> <li>MA は IF と競合します。</li> </ul>	

## 10. パイプライン動作

分類	区分	命令	実行 ステート	ステージ 段数	競合
システム 制御命令	STS.L 命令 (PR)	STS.L PR, @-Rn	1	5	<ul style="list-style-type: none"> <li>MA は IF と競合します。</li> </ul>
	LDC.L 命令	LDC.L @Rm+,GBR	1/5* <sup>6</sup>	5	<ul style="list-style-type: none"> <li>この命令の直後に、この命令のデスティネーションレジスタを使う命令を置くと、競合します。</li> <li>MA は IF と競合します。</li> </ul>
		LDC.L @Rm+,VBR			
		LDC.L @Rm+,SSR			
		LDC.L @Rm+,SPC			
		LDC.L @Rm+,MOD	5	5	
		LDC.L @Rm+,RE	1/5* <sup>6</sup>	5	
	LDC.L @Rm+,RS				
	LDC.L @Rm+,R0_BANK				
	LDC.L @Rm+,R1_BANK				
	LDC.L @Rm+,R2_BANK				
	LDC.L @Rm+,R3_BANK				
	LDC.L @Rm+,R4_BANK				
	LDC.L @Rm+,R5_BANK				
LDC.L @Rm+,R6_BANK					
LDC.L @Rm+,R7_BANK					
STC.L 命令 (除くバンクレジスタ)	STC.L SR, @-Rn	1/2* <sup>7</sup>	5	<ul style="list-style-type: none"> <li>MA は IF と競合します。</li> </ul>	
	STC.L GBR, @-Rn				
	STC.L VBR, @-Rn				
	STC.L SSR, @-Rn				
	STC.L SPC, @-Rn				
STC.L MOD, @-Rn	2	5			
STC.L RE, @-Rn					
STC.L RS, @-Rn					
STC.L 命令 (バンクレジスタ)	STC.L R0_BANK, @-Rn	2	6	<ul style="list-style-type: none"> <li>MA は IF と競合します。</li> </ul>	
	STC.L R1_BANK, @-Rn				
	STC.L R2_BANK, @-Rn				
	STC.L R3_BANK, @-Rn				
	STC.L R4_BANK, @-Rn				
	STC.L R5_BANK, @-Rn				
	STC.L R6_BANK, @-Rn				
STC.L R7_BANK, @-Rn					
レジスタ→ MAC/DSP 転送命令	CLRMAC	1	4	<ul style="list-style-type: none"> <li>乗算器との競合を起こします。</li> <li>MA は IF と競合します。</li> </ul>	
	LDS Rm, MACH				
	LDS Rm, MACL				
	LDS Rm, DSR				
	LDS Rm, A0				
	LDS Rm, X0				
	LDS Rm, X1				
	LDS Rm, Y0				
	LDS Rm, Y1				



分類	区分	命令	実行 ステート	ステージ 段数	競合
システム 制御命令	メモリ→ MAC/DSP 転送命令	LDS.L @Rm+,MACH	1	4	<ul style="list-style-type: none"> <li>乗算器との競合を起こします。</li> <li>MA は IF と競合します。</li> </ul>
		LDS.L @Rm+,MACL			
		LDS.L @Rm+,DSR			
		LDS.L @Rm+,A0			
		LDS.L @Rm+,X0			
		LDS.L @Rm+,X1			
		LDS.L @Rm+,Y0			
	LDS.L @Rm+,Y1				
	MAC/DSP→ レジスタ 転送命令	STS MACH,Rn	1	5	<ul style="list-style-type: none"> <li>乗算器との競合を起こします。</li> <li>この命令の直後に、この命令のデステイネーションレジスタを使う命令を置くと、競合します。</li> <li>MA は IF と競合します。</li> </ul>
		STS MACL,Rn			
		STS DSR, Rn			
		STS A0,Rn			
STS X0,Rn					
STS X1,Rn					
STS Y0,Rn					
STS Y1,Rn					
MAC/DSP→ メモリ 転送命令	STS.L MACH,@-Rn	1	5	<ul style="list-style-type: none"> <li>乗算器との競合を起こします。</li> <li>MA は IF と競合します。</li> </ul>	
	STS.L MACL,@-Rn				
	STS.L DSR,@-Rn				
	STS.L A0,@-Rn				
	STS.L X0,@-Rn				
	STS.L X1,@-Rn				
	STS.L Y0,@-Rn				
STS.L Y1,@-Rn					
RTE 命令	RTE	4	5		
TRAP 命令	TRAPA #imm	6/8* <sup>8</sup>	6/8* <sup>8</sup>		
SLEEP 命令	SLEEP	4	6		
レジスタ→ DSP 転送命令	CLRMACH	1	4	<ul style="list-style-type: none"> <li>乗算器との競合を起こします。</li> <li>MA は IF と競合します。</li> </ul>	
	LDS Rm,MACH				
	LDS Rm,MACL	1	4		
	LDS Rm,DSR				
	LDS Rm,A0				
	LDS Rm,X0				
	LDS Rm,X1				
LDS Rm,Y0					
LDS Rm,Y1					
メモリ→ DSP 転送命令	LDS.L @Rm+,MACH	1	4	<ul style="list-style-type: none"> <li>乗算器との競合を起こします。</li> <li>MA は IF と競合します。</li> </ul>	
	LDS.L @Rm+,MACL				
	DS.L @Rm+,DSR	1	4		
	DS.L @Rm+,A0				
	DS.L @Rm+,X0				
	DS.L @Rm+,X1				
	DS.L @Rm+,Y0				
DS.L @Rm+,Y1					

## 10. パイプライン動作

分類	区分	命令	実行 ステート	ステージ 段数	競合
システム 制御命令	DSP→ レジスタ 転送命令	STS MACH,Rn	1	5	<ul style="list-style-type: none"> <li>乗算器との競合を起こします。</li> <li>この命令の直後に、この命令のデステイネーションレジスタを使う命令を置くと、競合します。</li> <li>MA は IF と競合します。</li> <li>DSP 演算との競合を起こします。</li> </ul>
		STS MACL,Rn			
		STS DSR,Rn			
		STS A0,Rn			
		STS X0,Rn			
		STS X1,Rn			
		STS Y0,Rn			
STS Y1,Rn					
DSP→ メモリ 転送命令	STS.L MACH,@-Rn	STS.L MACL,@-Rn	1	4	<ul style="list-style-type: none"> <li>乗算器との競合を起こします。</li> <li>MA は IF と競合します。</li> </ul>
		STS.L DSR,@-Rn			
DSP→ メモリ 転送命令	STS.L A0,@-Rn	STS.L X0,@-Rn	1	4	
		STS.L X1,@-Rn			
		STS.L Y0,@-Rn			
		STS.L Y1,@-Rn			
		RTE 命令			
TRAP 命令	TRAPA #imm	8	9		
SLEEP 命令	SLEEP	3	3		
DSP データ 転送命令	Xメモリ ロード命令	NOPX	1	5	
		MOVX.W @Ax,Dx			
		MOVX.W @Ax+,Dx			
	Xメモリ ストア命令	MOVX.W Da,@Ax	1	4	<ul style="list-style-type: none"> <li>DSP 演算との競合を起こします。</li> </ul>
		MOVX.W Da,@Ax+			
	MOVX.W Da@Ax+Ix				
Yメモリ ロード命令	NOPY	1	5		
	MOYV.W @Ay,Dy				
	MOYV.W @Ay+,Dy				
	MOYV.W @Ay+Ix,Dy				
Yメモリ ストア命令	MOVY.W Da,@Ay	1	4	<ul style="list-style-type: none"> <li>DSP 演算との競合を起こします。</li> </ul>	
	MOVY.W Da,@Ay+				
	MOVY.W Da@Ay+Iy				
シングル ロード命令		MOV.S.W @-As,Ds	1	5	<ul style="list-style-type: none"> <li>MA は IF と競合します。</li> </ul>
		MOV.S.W @As,Ds			
		MOV.S.W @As+,Ds			
		MOV.S.W @As+Is,Ds			
		MOV.S.L @-As,Ds			
		MOV.S.L @As,Ds			
		MOV.S.L @As+,Ds			
		MOV.S.L @As+Is,Ds			

10. パイプライン動作

分類	区分	命令	実行 ステート	ステージ 段数	競合
DSP データ 転送命令	シングル ストア命令	MOVS.W Ds,@-As MOVS.W Ds,@As MOVS.W Ds,@As+ MOVS.W Ds,@As+Is MOVS.L Ds,@-As MOVS.L Ds,@As MOVS.L Ds,@As+ MOVS.L Ds,@As+Is	1	5	<ul style="list-style-type: none"> <li>• MA は IF と競合します。</li> <li>• DSP 演算との競合を起こします。</li> </ul>
DSP 演算命令	ALU 算術 演算命令	PADD Sx, Sy,Dz(Du) DCT PADD Sx, Sy,Dz DCF PADD Sx, Sy,Dz PSUB Sx, Sy,Dz(Du) DCT PSUB Sx, Sy,Dz DCF PSUB Sx, Sy,Dz PCOPY Sx, Dz DCT PCOPY Sx, Dz DCF PCOPY Sx, Dz PCOPY Sy, Dz DCT PCOPY Sy, Dz DCF PCOPY Sy, Dz PDMSB Sx,Dz DTC PDMSB Sx,Dz DCF PDMSB Sx,Dz PDMSB Sy,Dz DCT PDMSB Sy,Dz DCF PDMSB Sy,Dz PINC Sx,Dz DCT PINC Sx,Dz DCF PINC Sx,Dz PINC Sy,Dz DCT PINC Sy,Dz DCF PINC Sy,Dz PNEG Sx,Dz DCT PNEG Sx,Dz DCF PNEG Sx,Dz PNEG Sy,Dz DCT PNEG Sy,Dz DCF PNEG Sy,Dz PDEC Sx,Dz DTC PDEC Sx,Dz DCF PDEC Sx,Dz PDEC Sy,Dz DTC PDEC Sy,Dz DCF PDEC Sy,Dz PCLR Dz DCT PCLR Dz DCF PCLR Dz	1	5	

## 10. パイプライン動作

分類	区分	命令	実行 ステート	ステージ 段数	競合
DSP 演算命令	ALU 算術 演算命令	PADDC Sx,Sy,Dz PSUBC Sx,Sy,Dz PCMP Sx,Sy, PABS Sx,Dz PABS Sy,Dz PRNDSx,Dz PRNDSy,Dz	1	5	
	ALU 論理 演算命令	DCT POR Sx,Sy,Dz DCF POR Sx,Sy,Dz PAND Sx,Sy,Dz DCT PAND Sx,Sy,Dz DCF PAND Sx,Sy,Dz PXOR Sx,Sy,Dz DCT PXOR Sx,Sy,Dz DCF PXOR Sx,Sy,Dz	1	5	
	シフト命令	PSHA Sx,Sy,Dz DCT PSHA Sx,Sy,Dz DCF PSHA Sx,Sy,Dz PSHA #imm,Dz PSHL Sx,Sy,Dz DCT PSHL Sx,Sy,Dz DCF PSHL Sx,Sy,Dz PSHL #imm,Dz	1	5	
	乗算命令	PMULS Se,Sf,Dg	1	5	
	レジスタ間 転送命令	DTC PSTS MACH,Dz DCF PSTS MACH,Dz PSTS MAACL,Dz DCT PSTS MAACL,Dz DCF PSTS MAACL,Dz PLDS Dz,MACH DCT PLDS Dz,MACH DCF PLDS Dz,MACH PLDS Dz,MAACL DCT PLDS Dz,MAACL DCF PLDS Dz,MAACL	1	5	<ul style="list-style-type: none"> <li>MOVX.W,MOVVS.W,MOVVS.L との競合 します。</li> </ul>

- 【注】 \*1 SH3-DSP では 2 ステートになります。  
 \*2 通常実行ステートを示します。  
 ( ) 内の値は、後続の命令との競合関係による実行ステート数です。  
 \*3 SH3-DSP では 4 ステートになります。  
 \*4 分岐しないときは 1 ステートになります。  
 \*5 SH3-DSP では 3 ステートになります。  
 \*6 SH3-DSP では 5 ステートになります。  
 \*7 SH3-DSP では 2 ステートになります。  
 \*8 SH3-DSP では 8 ステート、8 ステージ段数になります。

### 10.4.1 データ転送命令

#### (1) レジスタ - レジスタ間転送命令

##### 命令の種類

MOV	#imm, Rn
MOV	Rm, Rn
MOVA	@(disp, PC), R0
MOVT	Rn
SWAP.B	Rm, Rn
SWAP.W	Rm, Rn
XTRCT	Rm, Rn

##### パイプライン

	スロット	←→	←→	←→	←→	←→
命令A	IF	ID	EX	ma	WB	
次命令		IF	ID	EX	.....	
次々命令			IF	ID	EX	.....
.....						

##### 動作説明

パイプラインは、IF、ID、EX、ma、WBの5段で終了します。maステージでは、何もせず転送するデータが保持され、WBステージでデータをレジスタに書き込みます。

#### (2) メモリロード命令

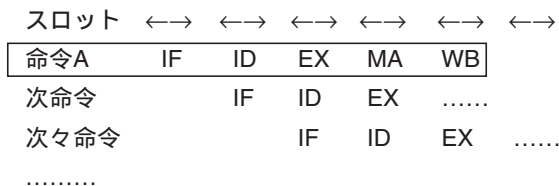
##### 命令の種類

MOV.W	@(disp, PC), Rn	MOV.B	@(disp, Rm), R0
MOV.L	@(disp, PC), Rn	MOV.W	@(disp, Rm), R0
MOV.B	@Rm, Rn	MOV.L	@(disp, Rm), Rn
MOV.W	@Rm, Rn	MOV.B	@(R0, Rm), Rn
MOV.L	@Rm, Rn	MOV.W	@(R0, Rm), Rn
MOV.B	@Rm+, Rn	MOV.L	@(R0, Rm), Rn
MOV.W	@Rm+, Rn	MOV.B	@(disp, GBR), R0
MOV.L	@Rm+, Rn	MOV.W	@(disp, GBR), R0
		MOV.L	@(disp, GBR), R0

## 10. パイプライン動作

---

### パイプライン



### 動作説明

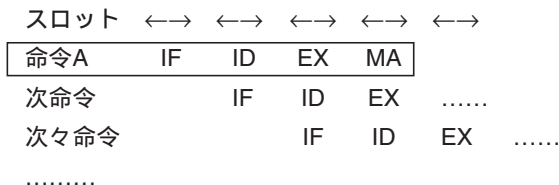
パイプラインは、IF、ID、EX、MA、WBの5段です。この命令の直後に、この命令のデスティネーションレジスタを使う命令を置くと、競合が発生します（「10.2.2 メモリロード命令による競合」参照）。また、これらの命令のMAはIFと競合します（「10.2.1 命令フェッチ（IF）とメモリアクセス（MA）の競合」参照）。

### (3) メモリストア命令

#### 命令の種類

MOV.B	Rm,@Rn	MOV.B	Rm,@(R0,Rn)
MOV.W	Rm,@Rn	MOV.W	Rm,@(R0,Rn)
MOV.L	Rm,@Rn	MOV.L	Rm,@(R0,Rn)
MOV.B	R0,@(disp,Rn)	MOV.B	R0,@(disp,GBR)
MOV.W	R0,@(disp,Rn)	MOV.W	R0,@(disp,GBR)
MOV.L	Rm,@(disp,Rn)	MOV.L	R0,@(disp,GBR)

### パイプライン



### 動作説明

パイプラインは、IF、ID、EX、MAの4段で終了します。レジスタへのデータの戻しがないのでWBステージはありません。これらの命令のMAはIFと競合します（「10.2.1 命令フェッチ（IF）とメモリアクセス（MA）の競合」参照）。

## (4) メモリストア命令 (プリデクリメント)

## 命令の種類

MOV.B	Rm, @-Rn
MOV.W	Rm, @-Rn
MOV.L	Rm, @-Rn

## パイプライン

スロット	←→	←→	←→	←→	←→	
命令A	IF	ID	EX	MA	WB	
次命令		IF	ID	EX	.....	
次々命令			IF	ID	EX	.....
.....						

## 動作説明

パイプラインは、IF、ID、EX、MA、WBの5段で終了します。WBステージで、デクリメントした値をレジスタへ書き込みます。これらの命令 MA は IF と競合します(「10.2.1 命令フェッチ(IF)とメモリアクセス(MA)の競合」参照)。

## (5) キャッシュ命令

## 命令の種類

PREF	@Rn
------	-----

## パイプライン

スロット	←→	←→	←→	←→	←→	
命令A	IF	ID	EX	MA		
次命令		IF	ID	EX	.....	
次々命令			IF	ID	EX	.....
.....						

## 動作説明

パイプラインは、IF、ID、EX、MAの4段で終了します。レジスタへのデータの戻しがないのでWBステージはありません。この命令のMAはIFと競合します(「10.2.1 命令フェッチ(IF)とメモリアクセス(MA)の競合」参照)。

SH3-DSPでは、次命令のIDは1スロット分後ろにストールされます。

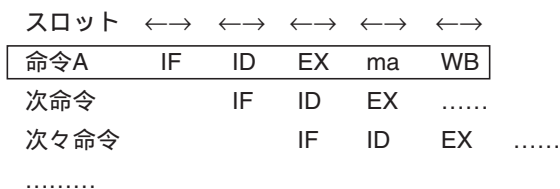
### 10.4.2 算術演算命令

(1) レジスタ間算術演算命令（乗算系命令を除く）

#### 命令の種類

ADD	Rm, Rn
ADD	#imm, Rn
EXTS.B	Rm, Rn
EXTS.W	Rm, Rn
EXTU.B	Rm, Rn
EXTU.W	Rm, Rn
NEG	Rm, Rn
SUB	Rm, Rn

#### パイプライン



#### 動作説明

パイプラインは、IF、ID、EX、ma、WB の5段で終了します。ma ステージでは、何もせず演算結果が保持され、WB ステージで結果をレジスタに書き込みます。

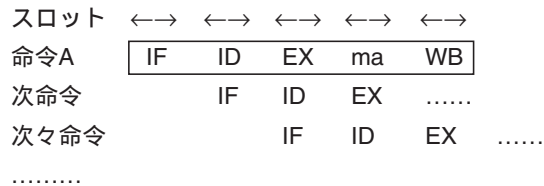


## (2) SR 更新レジスタ間算術演算命令 (乗算系命令除く)

## 命令の種類

ADDC	Rm, Rn
ADDV	Rm, Rn
CMP/EQ	#imm, R0
CMP/EQ	Rm, Rn
CMP/HS	Rm, Rn
CMP/GE	Rm, Rn
CMP/HI	Rm, Rn
CMP/GT	Rm, Rn
CMP/PL	Rn
CMP/PZ	Rn
CMP/STR	Rm, Rn
DIV1	Rm, Rn
DIV0S	Rm, Rn
DIV0U	
DT	Rn
NEGC	Rm, Rn
SUBC	Rm, Rn
SUBV	Rm, Rn

## パイプライン



## 動作説明

パイプラインは、IF、ID、EX、ma、WB の5段で終了します。ma ステージでは、何もせず演算結果が保持され、WB ステージで結果をレジスタに書き込みます。この命令の直後、あるいは、その次にSRを読み出す命令を置くと競合が発生します(「10.2.3 SR更新命令による競合」参照)。

## 10. パイプライン動作

### (3) 積和命令

#### 命令の種類

MAC.W @Rm+, @Rn+

#### パイプライン

スロット	←→	←→	←→	←→	←→	←→	←→	←→
命令A	IF	ID	EX	MA	MA	mm	mm	mm
次命令		IF		ID	EX	MA	WB	
次々命令				IF	ID	EX	MA	WB
.....								

#### 動作説明

パイプラインは、IF、ID、EX、MA、MA、mm、mm、mm の 8 段\*で終了します。2 番目の MA は、メモリ読み込みとともに乗算器のアクセスも行います。mm は乗算器が動作している状態を表しています。mm は最後の MA 終了後、スロットに関係なく 3 ステートの間、動作します。また、MAC.W 命令の次命令の ID は 1 スロット分後ろにストールされます。MAC.W 命令の 2 個の MA も、IF と競合する場合は、「10.2.1 命令フェッチ (IF) とメモリアクセス (MA) の競合」で説明したように IF ステージがストールします。

MAC.W 命令の後ろに乗算器を使わない命令がくる場合は、MAC.W 命令は IF、ID、EX、MA、MA、WB の 6 段パイプライン命令とみなして考えてかまいません。すなわち、この場合は、次命令の ID が 1 スロット分ストールするだけで、あとは、通常のパイプライン動作と同様になります。

しかし、MAC.W 命令の後ろに乗算器を使う命令がくる場合、乗算器の競合が発生しますので、通常の動作と異なってきます（「10.2.4 乗算器アクセスによる競合」参照）。

【注】\* SH3-DSP では、IF、ID、EX、MA、MA、mm、mm の 7 段

### (4) 倍精度積和命令

#### 命令の種類

MAC.L @Rm+, @Rn+

#### パイプライン

スロット	←→	←→	←→	←→	←→	←→	←→	←→
命令A	IF	ID	EX	MA	MA	mm	mm	mm
次命令		IF		ID	EX	MA	WB	
次々命令				IF	ID	EX	MA	WB
.....								

## 動作説明

パイプラインは、IF、ID、EX、MA、MA、mm、mm、mm の 8 段\*で終了します。2 番目の MA は、メモリ読み込みとともに乗算器のアクセスも行います。mm は乗算器が動作している状態を表しています。mm は最後の MA 終了後、スロットに関係なく 3 ステートの間、動作します。また、MAC.L 命令の次命令の ID は 1 スロット分後ろにストールされます。MAC.L 命令の 2 個の MA も、IF と競合する場合は、「10.2.1 命令フェッチ (IF) とメモリアクセス (MA) の競合」で説明したように IF ステージがストールします。

MAC.L 命令の後ろに乗算器を使わない命令がくる場合は、MAC.L 命令は IF、ID、EX、MA、MA、WB の 6 段パイプライン命令とみなして考えてかまいません。すなわち、この場合は、次命令の ID が 1 スロット分ストールするだけで、あとは、通常のパイプライン動作と同様になります。

しかし、MAC.L 命令の後ろに乗算器を使う命令がくる場合、乗算器の競合が発生しますので、通常の動作と異なってきます（「10.2.4 乗算器アクセスによる競合」参照）。

【注】\* SH3-DSP では、IF、ID、EX、MA、MA、mm、mm、mm、mm の 9 段

## (5) 乗算命令

## 命令の種類

MULS.W Rm, Rn  
MULU.W Rm, Rn

## パイプライン

	スロット	←→	←→	←→	←→	←→	←→	←→	←→
命令A		IF	ID	EX	MA	mm	mm		
次命令			IF	ID	EX	MA	WB		
次々命令				IF	ID	EX	MA	WB	
.....									

## 動作説明

パイプラインは、IF、ID、EX、MA、mm、mm の 6 段で終了します。MA は、乗算器のアクセスを行います。mm は乗算器が動作している状態を表しています。mm は MA 終了後、スロットに関係なく 2 ステートの間、動作します。MULS.W 命令の MA も、IF と競合する場合は「10.2.1 命令フェッチ (IF) とメモリアクセス (MA) の競合」で説明したように IF ステージがストールします。

MULS.W 命令の後ろに乗算器を使わない命令がくる場合は、MULS.W 命令は IF、ID、EX、MA の 4 段パイプライン命令とみなして考えてかまいません。すなわち、この場合は、通常のパイプライン動作と同様になります。

しかし、MULS.W 命令の後ろに乗算器を使う命令がくる場合、乗算器の競合が発生しますので、通常の動作と異なってきます（「10.2.4 乗算器アクセスによる競合」参照）。

## 10. パイプライン動作

---

### (6) 倍精度の乗算命令

#### 命令の種類

DMULS.L	Rm, Rn
DMULU.L	Rm, Rn
MUL.L	Rm, Rn

#### パイプライン

スロット	←→	←→	←→	←→	←→	←→	←→	←→
命令A	IF	ID	EX	MA	MA	mm	mm	mm
次命令		IF		ID	EX	MA	WB	
次々命令				IF	ID	EX	MA	WB
.....								

#### 動作説明

パイプラインは、IF、ID、EX、MA、MA、mm、mm、mmの8段\*で終了します。MAは、乗算器のアクセスを行います。mmは乗算器が動作している状態を表しています。mmはMA終了後、スロットに関係なく3ステートの間、動作します。また、DMULS.L命令の次命令のIDは1スロット分後ろにストールされます。DMULS.L命令の2個のMAも、IFと競合する場合は「10.2.1 命令フェッチ(IF)とメモリアクセス(MA)の競合」で説明したようにIFステージがストールします。

DMULS.L命令の後ろに乗算器を使わない命令がくる場合は、DMULS.L命令はIF、ID、EX、MA、MAの5段パイプライン命令とみなして考えてかまいません。すなわち、この場合は、通常のパイプライン動作と同様になります。

しかし、DMULS.L命令の後ろに乗算器を使う命令がくる場合、乗算器の競合が発生しますので、通常の動作と異なってきます(「10.2.4 乗算器アクセスによる競合」参照)。

【注】\* SH3-DSPでは、IF、ID、EX、MA、MA、mm、mm、mm、mmの9段

### 10.4.3 論理演算命令

#### (1) レジスタ - レジスタ間論理演算命令

##### 命令の種類

AND	Rm, Rn
AND	#imm, R0
NOT	Rm, Rn
OR	Rm, Rn
OR	#imm, R0
XOR	Rm, Rn
XOR	#imm, R0

##### パイプライン

スロット	↔	↔	↔	↔	↔
命令A	IF	ID	EX	ma	WB
次命令		IF	ID	EX	.....
次々命令			IF	ID	EX
.....					

##### 動作説明

パイプラインは、IF、ID、EX、ma、WB の5段で終了します。ma ステージでは、何もせず演算結果が保持され、WB ステージで結果をレジスタに書き込みます。

#### (2) SR 更新レジスタ間論理演算命令

##### 命令の種類

TST	Rm, Rn
TST	#imm, R0

##### パイプライン

スロット	↔	↔	↔	↔	↔
命令A	IF	ID	EX	ma	WB
次命令		IF	ID	EX	.....
次々命令			IF	ID	EX
.....					

## 10. パイプライン動作

---

### 動作説明

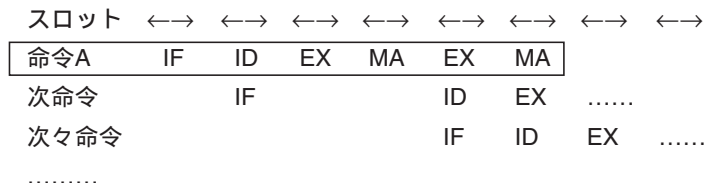
パイプラインは、IF、ID、EX、ma、WB の5段で終了します。ma ステージでは、何もせず演算結果が保持され、WB ステージで結果をSRのTビットに書き込みます。この命令の直後、あるいは、その次にSRを読み出す命令を置くと競合が発生します(「10.2.3 SR更新命令による競合」参照)。

### (3) メモリ論理演算命令

#### 命令の種類

AND.B        #imm,@(R0,GBR)  
OR.B         #imm,@(R0,GBR)  
XOR.B        #imm,@(R0,GBR)

#### パイプライン



### 動作説明

パイプラインは、IF、ID、EX、MA、EX、MA の6段で終了します。次命令のIDは2スロット分ストールされます。これらの命令のMAはIFと競合します(「10.2.1 命令フェッチ(IF)とメモリアクセス(MA)の競合」参照)。

### (4) SR更新メモリ論理演算命令

#### 命令の種類

TST.B        #imm,@(R0,GBR)

#### パイプライン



## 動作説明

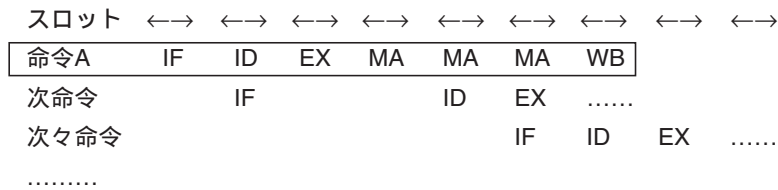
パイプラインは、IF、ID、EX、MA、EX、MA、WB の7段で終了します。WB ステージで結果をSRのTビットに書き込みます。TST命令のMAはIFと競合します（「10.2.1 命令フェッチ（IF）とメモリアクセス（MA）の競合」参照）。また、この命令の直後、あるいは、その次にSRを読み出す命令を置くと競合が発生します（「10.2.3 SR更新命令による競合」参照）。

## (5) TAS 命令

## 命令の種類

TAS.B @Rn

## パイプライン



## 動作説明

パイプラインは、IF、ID、EX、MA、MA、MA、WB の7段で終了します。次命令のIDは2スロット分ストールされます。TAS命令のMAはIFと競合します（「10.2.1 命令フェッチ（IF）とメモリアクセス（MA）の競合」参照）。また、この命令の直後、あるいは、その次にSRを読み出す命令を置くと競合が発生します（「10.2.3 SR更新命令による競合」参照）。

SH3-DSPでは次命令のIDは3スロット分ストールされます。

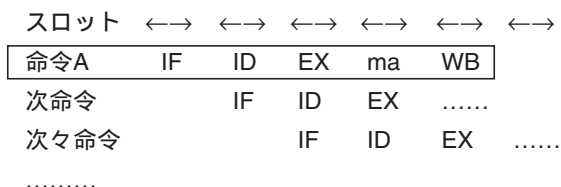
### 10.4.4 シフト命令

#### (1) シフト命令

##### 命令の種類

SHLL2	Rn
RnSHLR2	Rn
SHLL8	Rn
SHLR8	Rn
SHLL16	Rn
SHLR16	Rn
SHAD	Rm, Rn
SHLD	Rm, Rn

##### パイプライン



##### 動作説明

パイプラインは、IF、ID、EX、ma、WB の5段で終了します。ma ステージでは、何もせずシフト結果が保持され、WB ステージで結果をレジスタに書き込みます。

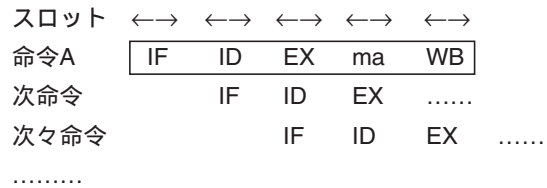
#### (2) SR 更新シフト命令

##### 命令の種類

ROTL	Rn
ROTR	Rn
ROTCL	Rn
ROTCLR	Rn
SHAL	Rn
SHAR	Rn
SHLL	Rn
SHLR	Rn



## パイプライン



## 動作説明

パイプラインは、IF、ID、EX、ma、WB の5段で終了します。ma ステージでは、何もせず結果が保持され、WB ステージで結果をレジスタに書き込みます。この命令の直後、あるいは、その次に SR を読み出す命令を置くと競合が発生します（「10.2.3 SR 更新命令による競合」参照）。

### 10.4.5 分岐命令

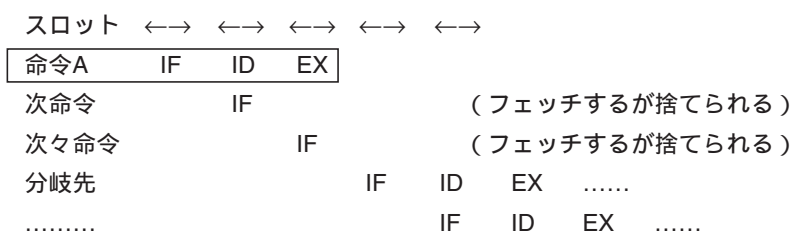
#### (1) 条件分岐命令

##### 命令の種類

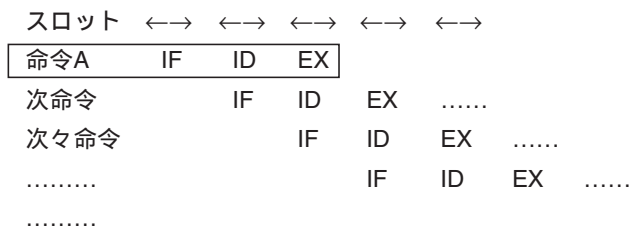
BF            label  
 BT            label

##### パイプライン

##### (a) 条件が成立したとき



##### (b) 条件が成立しないとき



##### 動作説明

パイプラインは、IF、ID、EX の3段で終了します。ID ステージで条件判断を行います。条件分岐命令は遅延分岐ではありません。

##### (a) 条件が成立したとき

EXステージで、分岐先アドレスを計算します。条件分岐命令(命令A)の次命令と次々命令は、フェッチしますが捨てられます。分岐先命令は、命令AのEXステージがあるスロットの次のスロットからフェッチを開始します。

##### (b) 条件が成立しないとき

IDステージで条件が成立しないと判断したら、EXステージでは何もせずに進みます。次命令も続けてフェッチし実行していきます。

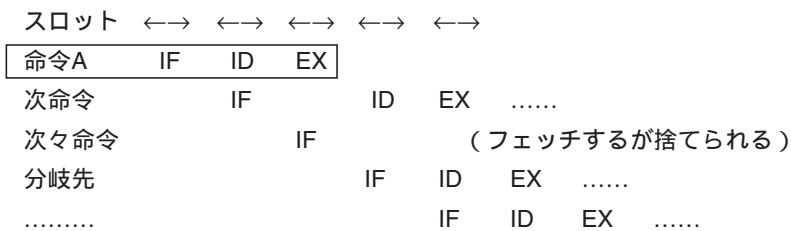
## (2) 遅延付き条件分岐命令

## 命令の種類

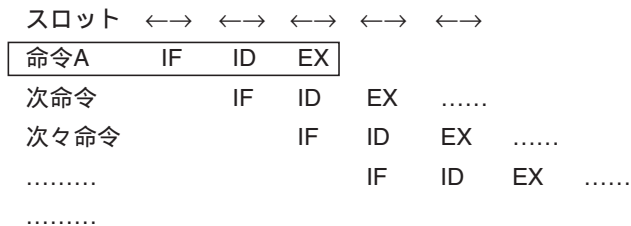
BF/S            label  
BT/S            label

## パイプライン

## (a) 条件が成立したとき



## (b) 条件が成立しないとき



## 動作説明

パイプラインは、IF、ID、EX の 3 段で終了します。ID ステージで条件判断を行います。

## (a) 条件が成立したとき

EX ステージで、分岐先アドレスを計算します。条件分岐命令 (命令A) の次命令すなわち遅延スロット命令はフェッチされ実行されますが、次々命令はフェッチされても捨てられます。分岐先命令は、命令AのEXステージがあるスロットの次のスロットからフェッチを開始します。

## (b) 条件が成立しないとき

ID ステージで条件が成立しないと判断したら、EX ステージでは何もせずに進みます。次命令も続けてフェッチし実行していきます。

## 10. パイプライン動作

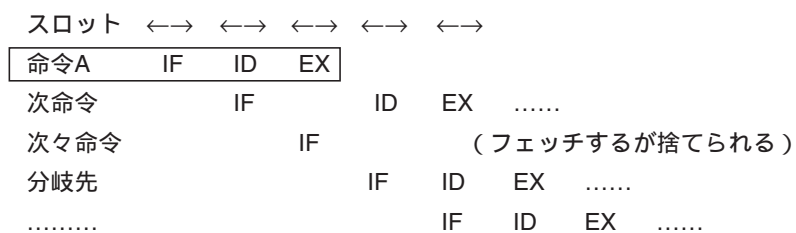
---

### (3) 無条件分岐命令

#### 命令の種類

BRA	label
BRAF	Rm
JMP	@Rm
RTS	

#### パイプライン



#### 動作説明

パイプラインは、IF、ID、EX の 3 段で終了します。無条件分岐命令は遅延分岐です。

EX ステージで、分岐先アドレスを計算します。無条件分岐命令（命令 A）の次命令すなわち遅延スロット命令は、フェッチされ実行されますが、次々命令はフェッチされても捨てられます。分岐先命令は、命令 A の EX ステージがあるスロットの次のスロットからフェッチを開始します。

## (4) 無条件分岐命令 (PR)

## 命令の種類

BSR	label
BSRF	Rm
JSR	@Rm

## パイプライン

スロット	←→	←→	←→	←→	←→		
命令A	IF	ID	EX	ma	WB		
次命令		IF		ID	EX	.....	
次々命令			IF			(フェッチするが捨てられる)	
.....				IF	ID	EX	.....

## 動作説明

パイプラインは、IF、ID、EX、ma、WB の 5 段で終了します。無条件分岐命令は遅延分岐です。

EX ステージで、分岐先アドレスを計算します。ma ステージでは、何もせず戻り先アドレスが保持され、WB ステージで PR に書き込みます。無条件分岐命令 (命令 A) の次命令すなわち遅延スロット命令は、フェッチされ実行されますが、次々命令はフェッチされても捨てられます。分岐先命令は、命令 A の EX ステージがあるスロットの次のスロットからフェッチを開始します。

### 10.4.6 システム制御命令

#### (1) システム制御 ALU 命令

命令の種類

LDC	Rm, GBR	STC	SR, Rn	LDRE	@(disp, pc)
LDC	Rm, VBR	STC	GBR, Rn	LDRS	@(disp, pc)
LDC	Rm, SSR	STC	VBR, Rn	SETRC	Rm
LDC	Rm, SPC	STC	SSR, Rn	SETRC	#imm
LDC	Rm, MOD	STC	SPC, Rn		
LDC	Rm, RE	STC	MOD, Rn		
LDC	Rm, RS	STC	RE, Rn		
LDC	Rm, R0_BANK	STC	RS, Rn		
LDC	Rm, R1_BANK	STC	R0_BANK, Rn		
LDC	Rm, R2_BANK	STC	R1_BANK, Rn		
LDC	Rm, R3_BANK	STC	R2_BANK, Rn		
LDC	Rm, R4_BANK	STC	R3_BANK, Rn		
LDC	Rm, R5_BANK	STC	R4_BANK, Rn		
LDC	Rm, R6_BANK	STC	R5_BANK, Rn		
LDC	Rm, R7_BANK	STC	R6_BANK, Rn		
LDS	Rm, PR	STC	R7_BANK, Rn		
		STS	PR, Rn		

パイプライン

スロット	←→	←→	←→	←→	←→
命令A	IF	ID	EX	ma	WB
次命令		IF	ID	EX	.....
次々命令			IF	ID	EX
.....					

動作説明

パイプラインは、IF、ID、EX、ma、WB の 5 段で終了します。EX ステージで、ALU を通してデータ演算は完結します。ma ステージでは、何もせず転送するデータが保持され、WB ステージでデータをレジスタに書き込みます。

SH3-DSP では、LDC、LDRE、LDRS、SETRC 命令の次命令の ID は 2 スロット分ストールされます。

## (2) SR 更新システム制御命令

## 命令の種類

CLRS  
CLRT  
SETS  
SETT

## パイプライン

スロット	←→	←→	←→	←→	←→		
命令A	IF	ID	EX	ma	WB		
次命令		IF	ID	EX	.....		
次々命令			IF	ID	EX	.....	
.....				IF	ID	EX	.....

## 動作説明

パイプラインは、IF、ID、EX、ma、WB の5段で終了します。ma ステージでは、何もせず転送するデータが保持され、WB ステージでデータをレジスタに書き込みます。この命令の直後、あるいは、その次に SR を読み出す命令を置くと競合が発生します(「10.2.3 SR 更新命令による競合」参照)。

## (3) LDTLB 命令

## 命令の種類

LDTLB

## パイプライン

スロット	←→	←→	←→	←→	←→	
命令A	IF	ID	EX	MA		
次命令		IF	ID	EX	.....	
次々命令			IF	ID	EX	.....
.....						

## 動作説明

パイプラインは、IF、ID、EX、MA の4段で終了します。レジスタへのデータの戻しがないので WB ステージはありません。この命令の MA は IF と競合します(「10.2.1 命令フェッチ (IF) とメモリアクセス (MA) の競合」参照)。

## 10. パイプライン動作

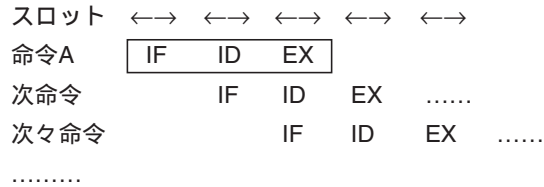
---

### (4) NOP 命令

命令の種類

NOP

パイプライン



動作説明

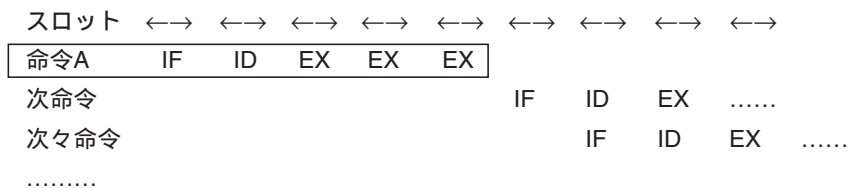
パイプラインは、IF、ID、EX の 3 段で終了します。

### (5) LDC 命令 (SR)

命令の種類

LDC                    Rm, SR

パイプライン



動作説明

パイプラインは、IF、ID、EX、EX、EX の 5 段で終了します。最後の EX ステージでデータを SR に書き込みます。次命令の IF は命令 A の最後の EX があるスロットの次のスロットから開始されません。

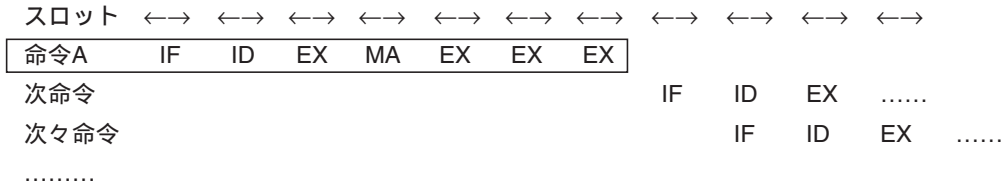
### (6) LDC.L 命令 (SR)

命令の種類

LDC.L                    @Rm+, SR



## パイプライン



## 動作説明

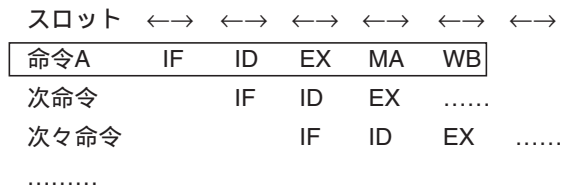
パイプラインは、IF、ID、EX、MA、EX、EX、EX の 7 段で終了します。最後の EX ステージでデータを SR に書き込みます。次命令の IF は命令 A の最後の EX があるスロットの次のスロットから開始されます。

## (7) LDS.L 命令 (PR)

## 命令の種類

LDS.L @Rm+, PR

## パイプライン



## 動作説明

パイプラインは、IF、ID、EX、MA、WB の 5 段で終了します。この命令直後に、この命令のデスティネーションレジスタを使う命令を置くと競合が発生します（「10.2.2 メモリロード命令による競合」参照）。また、この命令の MA は IF と競合します（「10.2.1 命令フェッチ (IF) とメモリアクセス (MA) の競合」参照）。

## (8) STS.L 命令 (PR)

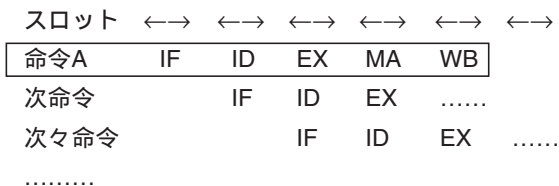
## 命令の種類

STS.L PR, @-Rn

## 10. パイプライン動作

---

### パイプライン



### 動作説明

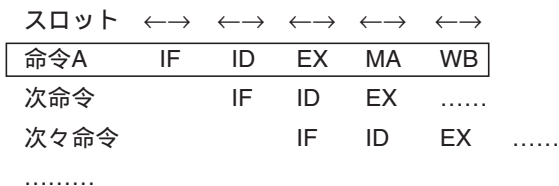
パイプラインは、IF、ID、EX、MA、WBの5段で終了します。WBステージでは、デクリメントした値をレジスタへ書き込みます。この命令のMAはIFと競合します(「10.2.1 命令フェッチ(IF)とメモリアクセス(MA)の競合」参照)。

### (9) LDC.L 命令

#### 命令の種類

LDC.L	@Rm+,GBR	LDC.L	@Rm+,R0_BANK
LDC.L	@Rm+,VBR	LDC.L	@Rm+,R1_BANK
LDC.L	@Rm+,SSR	LDC.L	@Rm+,R2_BANK
LDC.L	@Rm+,SPC	LDC.L	@Rm+,R3_BANK
LDC.L	@Rm+,MOD	LDC.L	@Rm+,R4_BANK
LDC.L	@Rm+,RE	LDC.L	@Rm+,R5_BANK
LDC.L	@Rm+,RS	LDC.L	@Rm+,R6_BANK
		LDC.L	@Rm+,R7_BANK

### パイプライン



### 動作説明

パイプラインは、IF、ID、EX、MA、WBの5段で終了します。この命令直後に、この命令のデスティネーションレジスタを使う命令を置くと競合が発生します(「10.2.2 メモリロード命令による競合」参照)。また、この命令のMAはIFと競合します(「10.2.1 命令フェッチ(IF)とメモリアクセス(MA)の競合」参照)。

SH3-DSPでは、次命令のIDは4スロット分ストールされます。

## (10) STC.L 命令 (除くバンクレジスタ)

## 命令の種類

STC.L	SR, @-Rn
STC.L	GBR, @-Rn
STC.L	VBR, @-Rn
STC.L	SSR, @-Rn
STC.L	SPC, @-Rn
STC.L	MOD, @-Rn
STC.L	RE, @-Rn
STC.L	RS, @-Rn

## パイプライン

スロット	←→	←→	←→	←→	←→	
命令A	IF	ID	EX	MA	WB	
次命令		IF	ID	EX	.....	
次々命令			IF	ID	EX	.....
.....						

## 動作説明

パイプラインは、IF、ID、EX、MA、WB の 5 段で終了します。WB ステージでは、デクリメントした値をレジスタへ書き込みます。これらの命令の MA は IF と競合します（「10.2.1 命令フェッチ (IF) とメモリアクセス (MA) の競合」参照）。

SH3-DSP では、次命令の ID は 1 スロット分ストールされます。

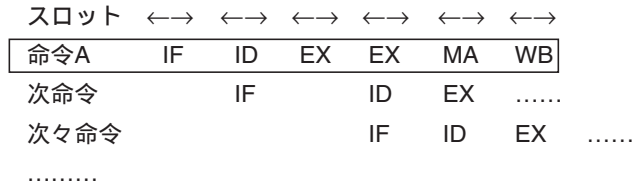
## (11) STC.L 命令 (バンクレジスタ)

## 命令の種類

STC.L	R0_BANK, @-Rn
STC.L	R1_BANK, @-Rn
STC.L	R2_BANK, @-Rn
STC.L	R3_BANK, @-Rn
STC.L	R4_BANK, @-Rn
STC.L	R5_BANK, @-Rn
STC.L	R6_BANK, @-Rn
STC.L	R7_BANK, @-Rn

## 10. パイプライン動作

### パイプライン



### 動作説明

パイプラインは、IF、ID、EX、EX、MA、WB の6段で終了します。次命令のIDは1スロット分ストールされます。これらの命令のMAはIFと競合します（「10.2.1 命令フェッチ（IF）とメモリアクセス（MA）の競合」参照）。

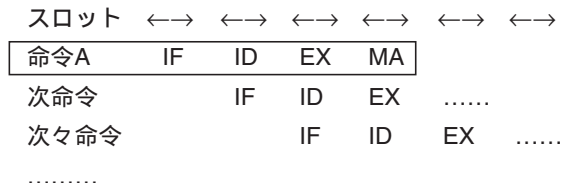
### (12) レジスタ→MAC/DSP 転送命令

#### 命令の種類

##### CLRMAC

LDS	Rm, MACH
LDS	Rm, MACL
LDS	Rm, DSR
LDS	Rm, A0
LDS	Rm, X0
LDS	Rm, X1
LDS	Rm, Y0
LDS	Rm, Y1

### パイプライン



### 動作説明

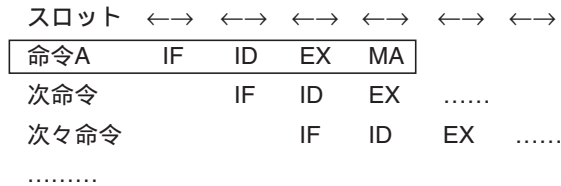
パイプラインは、IF、ID、EX、MA の4段で終了します。MAは乗算器アクセスのためのステージです。このMAはIFと競合を起こします（「10.2.1 命令フェッチ（IF）とメモリアクセス（MA）の競合」参照）。また、これらの命令の後ろに乗算器を使う命令がくる場合、乗算器の競合が発生します（「10.2.4 乗算器アクセスによる競合」参照）。

## (13) メモリ→MAC/DSP 転送命令

## 命令の種類

LDS.L	@Rm+, MACH	LDS.L	@Rm+, DSR
LDS.L	@Rm+, MACL	LDS.L	@Rm+, A0
		LDS.L	@Rm+, X0
		LDS.L	@Rm+, X1
		LDS.L	@Rm+, Y0
		LDS.L	@Rm+, Y1

## パイプライン



## 動作説明

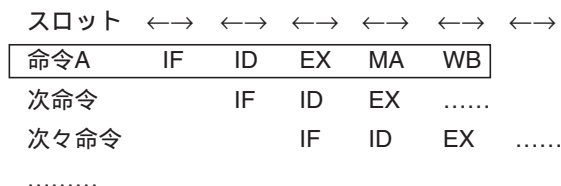
パイプラインは、IF、ID、EX、MA の4段で終了します。MA はメモリアクセスと乗算器アクセスのためのステージです。このMA はIF と競合を起こします（「10.2.1 命令フェッチ (IF) とメモリアクセス (MA) の競合」参照）。また、これらの命令の後ろに乗算器を使う命令がくる場合、乗算器の競合が発生します（「10.2.4 乗算器アクセスによる競合」参照）。

## (14) MAC/DSP→レジスタ転送命令

## 命令の種類

STS	MACH, Rn	STS	DSR, Rn
STS	MACL, Rn	STS	A0, Rn
		STS	X0, Rn
		STS	X1, Rn
		STS	Y0, Rn
		STS	Y1, Rn

## パイプライン



## 10. パイプライン動作

---

### 動作説明

パイプラインは、IF、ID、EX、MA、WB の5段で終了します。MA は乗算器アクセスのためのステージです。この MA は IF と競合を起こします(「10.2.1 命令フェッチ(IF)とメモリアクセス(MA)の競合」参照)。また、これらの命令の直後に、この命令のデスティネーションレジスタを使う命令を置いたり、乗算器を使う命令を置くと競合が発生します(「10.2.2 メモリロード命令による競合」および「10.2.4 乗算器アクセスによる競合」参照)。

### (15) MAC/DSP→メモリ転送命令

#### 命令の種類

STS.L	MACH,@-Rn	STS.L	DSR,@-Rn
STS.L	MACL,@-Rn	STS.L	A0,@-Rn
		STS.L	X0,@-Rn
		STS.L	X1,@-Rn
		STS.L	Y0,@-Rn
		STS.L	Y1,@-Rn

### パイプライン

スロット	←→	←→	←→	←→	←→	←→
命令A	IF	ID	EX	MA	WB	
次命令		IF	ID	EX	.....	
次々命令			IF	ID	EX	.....
.....						

### 動作説明

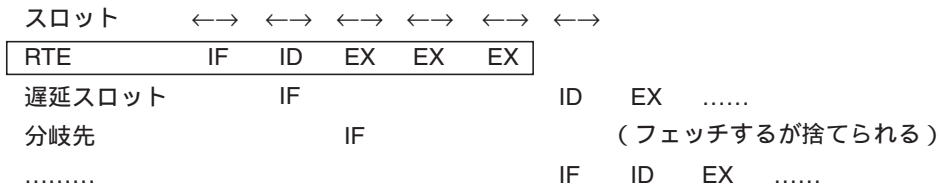
パイプラインは、IF、ID、EX、MA、WB の5段で終了します。MA は乗算器アクセスのためのステージです。この MA は IF と競合を起こします。(「10.2.1 命令フェッチ(IF)とメモリアクセス(MA)の競合」参照)。また、これらの命令の後ろに乗算器を使う命令がくる場合、乗算器の競合が発生します(「10.2.4 乗算器アクセスによる競合」参照)。

## (16) RTE 命令

## 命令の種類

RTE

## パイプライン



## 動作説明

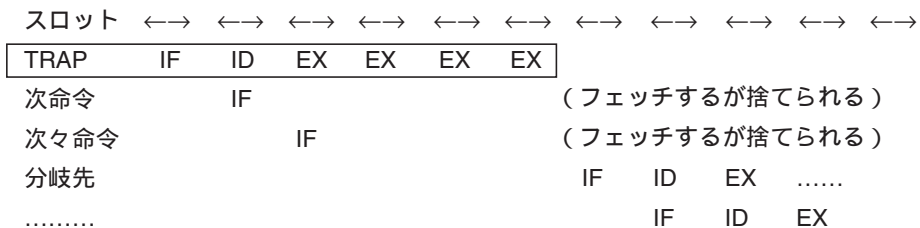
パイプラインは、IF、ID、EX、EX、EX の 5 段で終了します。RTE は遅延分岐命令です。RTE 命令の次命令すなわち遅延スロット命令はフェッチされ実行されますが、次々命令はフェッチされても捨てられます。分岐先命令の IF は RTE の最後の EX のあるスロットの次のスロットから開始されます。

## (17) TRAP 命令

## 命令の種類

TRAPA #imm

## パイプライン



## 動作説明

パイプラインは、IF、ID、EX、EX、EX、EX の 6 段\*で終了します。TRAP 命令は遅延分岐命令ではありません。TRAP 命令の次命令と次々命令はフェッチされますが、実行されずに捨てられます。分岐先命令の IF は TRAP 命令の最後の EX のあるスロットの次のスロットから開始されます。

【注】\* SH3-DSP では、IF、ID、EX、EX、EX、EX、EX、EX の 8 段

## 10. パイプライン動作

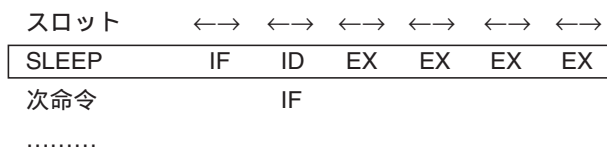
---

### (18) SLEEP 命令

命令の種類

SLEEP

パイプライン



動作説明

パイプラインは、IF、ID、EX、EX、EX、EX の 6 段で終了します。次命令の IF まででは発行されません。SLEEP 命令を実行後、スリープモードまたはスタンバイモードに入ります。



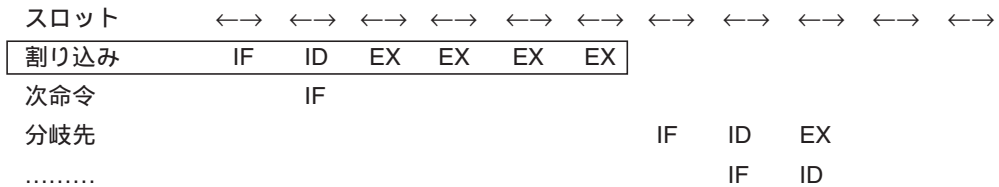
## 10.4.7 例外処理

### (1) 割り込み例外処理

命令の種類

割り込み例外処理

パイプライン



動作説明

割り込みは命令の ID ステージで受け付けられ、その ID ステージ以降を割り込み例外処理のシーケンスに置き換えます。

パイプラインは、IF、ID、EX、EX、EX、EX の 6 段で終了します。割り込み例外処理は遅延分岐ではありません。割り込み例外処理では、オーバランフェッチ (IF) が起こります。分岐先命令は割り込み例外処理の最後の EX があるスロットの次のスロットから IF を開始します。

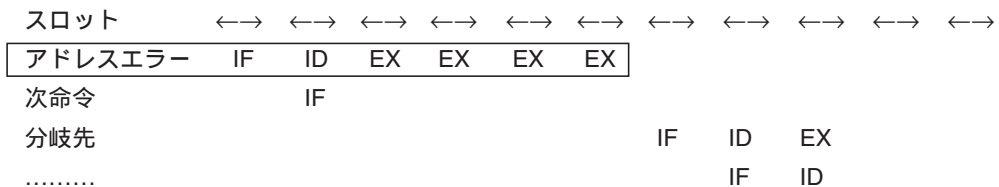
割り込み要因には、NMI、IRL、内蔵周辺モジュールによる割り込みがあります。詳細については、ハードウェアマニュアルを参照してください。

### (2) アドレスエラー例外処理

命令の種類

アドレスエラー例外処理

パイプライン



## 10. パイプライン動作

### 動作説明

アドレスエラーは命令の ID ステージで受け付けられ、その ID ステージ以降をアドレスエラー例外処理のシーケンスに置き換えます。

パイプラインは、IF、ID、EX、EX、EX、EX の 6 段で終了します。アドレスエラー例外処理は遅延分岐ではありません。アドレスエラー例外処理では、オーバランフェッチ (IF) が起こります。分岐先命令はアドレスエラー例外処理の最後の EX があるスロットの次のスロットから IF を開始します。

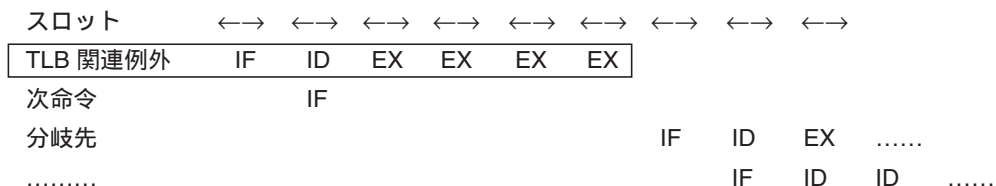
アドレスエラーの発生要因には、命令フェッチによるものとデータの読み出しあるいは書き込みによるものがあります。奇数アドレスから命令をフェッチすると命令フェッチのアドレスエラーとなり、ワードデータをワード境界以外からアクセス、ロングワードデータをロングワード境界以外からアクセスするとデータの読み出しあるいは書き込みのアドレスエラーとなります。詳細については、ハードウェアマニュアルを参照してください。

### (3) TLB 関連例外処理

#### 命令の種類

#### TLB 関連例外処理

#### パイプライン



### 動作説明

TLB 関連例外は命令の ID ステージで受け付けられ、その ID ステージ以降を TLB 関連例外処理のシーケンスに置き換えます。

パイプラインは、IF、ID、EX、EX、EX、EX の 6 段で終了します。TLB 関連例外処理は遅延分岐ではありません。TLB 関連例外処理では、オーバランフェッチ (IF) が起こります。分岐先命令は TLB 関連例外処理の最後の EX があるスロットの次のスロットから IF を開始します。

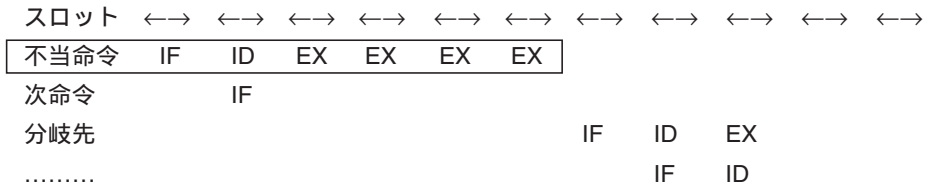
TLB 関連例外の発生要因には、TLB ミス、TLB インバリッド、TLB イニシャルライト、TLB 保護例外があります。詳細については、ハードウェアマニュアルを参照してください。

## (4) 不当命令例外処理

命令の種類

不当命令例外処理

パイプライン



動作説明

不当命令は命令の ID ステージで受け付けられ、その ID ステージ以降を不当命令例外処理のシーケンスに置き換えます。

パイプラインは、IF、ID、EX、EX、EX、EX の 6 段で終了します。不当命令例外処理は遅延分岐ではありません。不当命令例外処理では、オーバランフェッチ (IF) が起こります。分岐先命令は不当命令例外処理の最後の EX があるスロットの次のスロットから IF を開始します。

不当命令例外処理要因には、一般不当命令によるものとスロット不当命令によるものがあります。遅延分岐命令直後のスロット (遅延スロットと呼ぶ) 以外に配置されている未定義コードをデコードすると一般不当命令例外処理となります。遅延スロットに配置されている未定義コードをデコードする、または遅延スロットにプログラムカウンタを書き換える命令を配置し、それをデコードするか、または、ユーザモードで遅延スロットに特権命令を配置し、それをデコードするとスロット不当命令となります。詳細については、ハードウェアマニュアルを参照してください。

10.4.8 FPU 命令のパイプライン (SH-3E のみ)

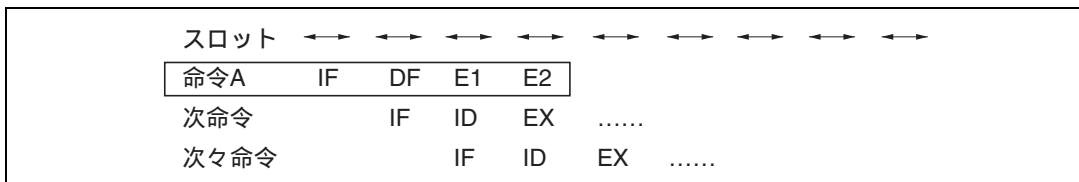


図 10.22 浮動小数点レジスタ - レジスタ間転送時の FPU パイプライン

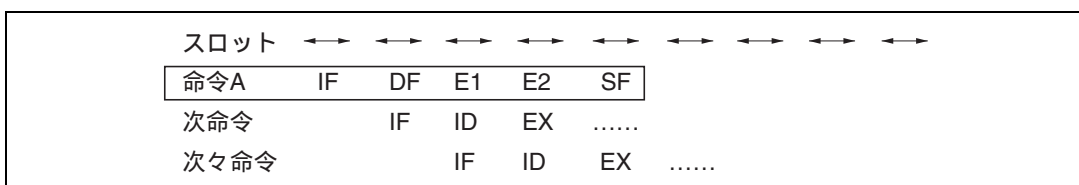


図 10.23 浮動小数点ロード時の FPU パイプライン

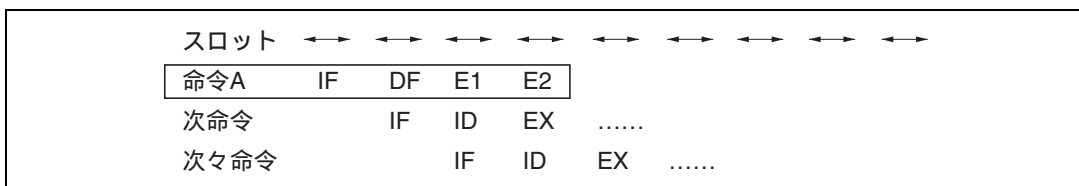


図 10.24 浮動小数点ストア時の FPU パイプライン

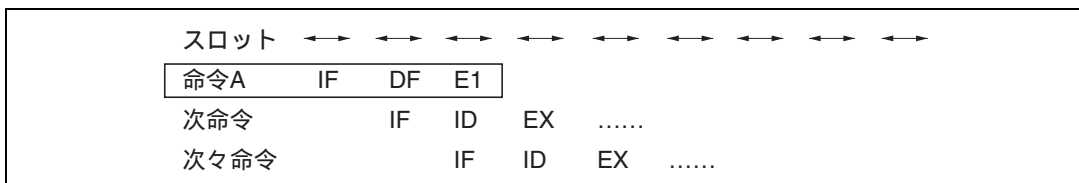


図 10.25 浮動小数点比較時の FPU パイプライン

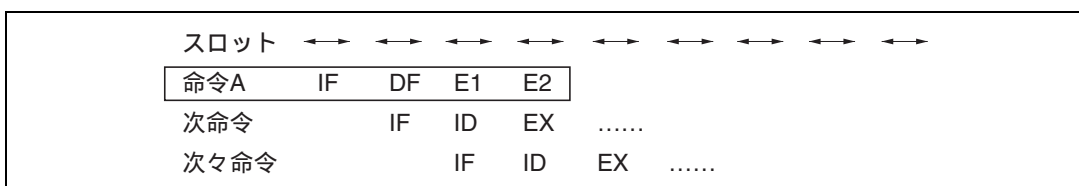


図 10.26 浮動小数点数算術演算命令 (FDIV と FSQRT を除く) の FPU パイプライン



図 10.27 FDIV および FSQRT 命令の FPU パイプライン

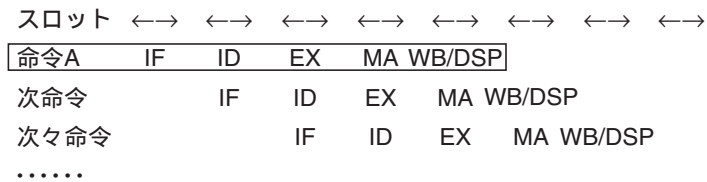
### 10.4.9 DSP データ転送命令 (SH3-DSP のみ)

#### (1) Xメモリ/Yメモリロード命令

##### 命令の種類

NOPX  
 MOVX.W @Ax, Dx  
 MOVX.W @Ax+, Dx  
 MOVX.W @Ax+Ix, Dx

##### パイプライン



##### 動作説明

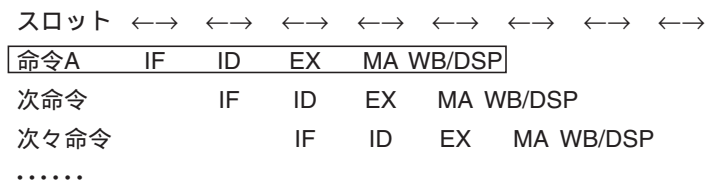
パイプラインは、IF、ID、EX、MA、WB/DSPの5段です。このデータ転送はXバスを経由して実行されるため、別命令のIFとは競合しません。

#### (2) Yメモリロード命令

##### 命令の種類

NOPY  
 MOVY.W @Ay, Dy  
 MOVY.W @Ay+, Dy  
 MOVY.W @Ay+Iy, Dy

##### パイプライン



## 動作説明

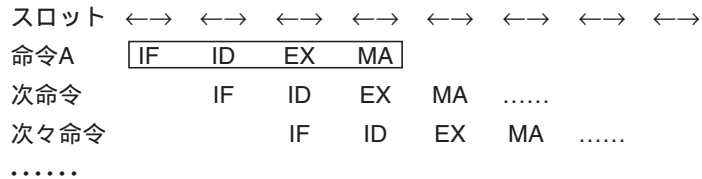
パイプラインは、IF、ID、EX、MA、WB/DSP の5 段です。このデータ転送は Y バスを經由して実行されるため、別命令の IF とは競合しません。

## (3) X メモリストア命令

## 命令の種類

MOVX.W     Da, @Ax  
 MOVX.W     Da, @Ax+  
 MOVX.W     Da, @Ax+Ix

## パイプライン



## 動作説明

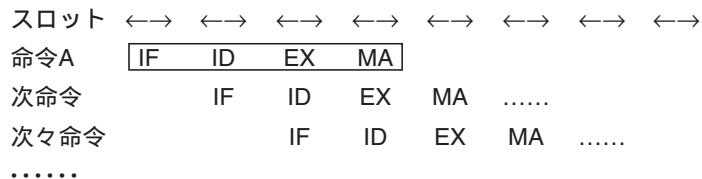
パイプラインは、IF、ID、EX、MA の4 段です。DSP 演算命令の直後にこの命令で DSP 演算結果をストアしようとする、競合が発生します（「10.2.6 DSP データ演算命令とストア命令の競合」を参照）。

## (4) Y メモリストア命令

## 命令の種類

MOVY.W     Da, @Ay  
 MOVY.W     Da, @Ay+  
 MOVY.W     Da, @Ay+Iy

## パイプライン



## 10. パイプライン動作

---

### 動作説明

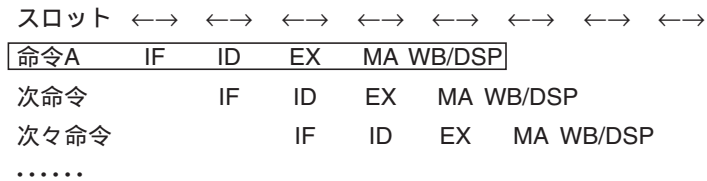
パイプラインは、IF、ID、EX、MA の4段です。DSP 演算命令の直後にこの命令で DSP 演算結果をストアしようとする、競合が発生します（「10.2.6 DSP データ演算命令とストア命令の競合」を参照）。

### (5) シングルロード命令

#### 命令の種類

MOVS.W	@-As, Ds
MOVS.W	@As, Ds
MOVS.W	@As+, Ds
MOVS.W	@As+Is, Ds
MOVS.L	@-As, Ds
MOVS.L	@As, Ds
MOVS.L	@As+, Ds
MOVS.L	@As+Is, Ds

#### パイプライン



### 動作説明

パイプラインは、IF、ID、EX、MA、WB/DSP の5段です。この命令のデスティネーションレジスタを使う命令を置いて、競合は発生しません。

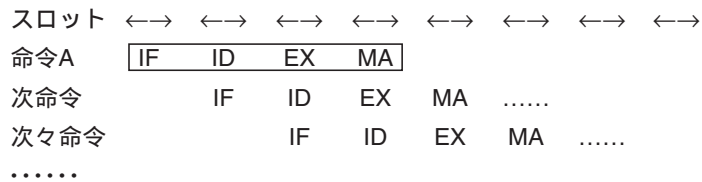


## (6) シングルストア命令

## 命令の種類

MOVS.W	Ds, @-As
MOVS.W	Ds, @As,
MOVS.W	Ds, @As+
MOVS.W	Ds, @As+Is
MOVS.L	Ds, @-As
MOVS.L	Ds, @As,
MOVS.L	Ds, @As+
MOVS.L	Ds, @As+I

## パイプライン



## 動作説明

パイプラインは、IF、ID、EX、MA の4段です。DSP 演算命令の直後にこの命令で DSP 演算結果をストアしようとする、競合が発生します（「10.2.6 DSP データ演算命令とストア命令の競合」を参照）。

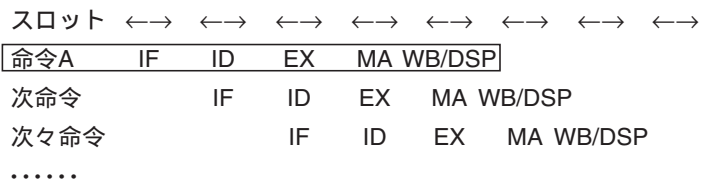
### 10.4.10 DSP 演算命令 (SH3-DSP のみ)

#### (1) ALU 算術演算命令

##### 命令の種類

	PADD Sx, Sy, Dz (Du)		PNEG Sx, Dz
DCT	PADD Sx, Sy, Dz	DCT	PNEG Sx, Dz
DCF	PADD Sx, Sy, Dz	DCF	PNEG Sx, Dz
	PSUB Sx, Sy, Dz (Du)		PNEG Sy, Dz
DCT	PSUB Sx, Sy, Dz	DCT	PNEG Sy, Dz
DCF	PSUB Sx, Sy, Dz	DCF	PNEG Sy, Dz
	PCOPY Sx, Dz		PDEC Sx, Dz
DCT	PCOPY Sx, Dz	DCT	PDEC Sx, Dz
DCF	PCOPY Sx, Dz	DCF	PDEC Sx, Dz
	PCOPY Sy, Dz		PDEC Sy, Dz
DCT	PCOPY Sy, Dz	DCT	PDEC Sy, Dz
DCF	PCOPY Sy, Dz	DCF	PDEC Sy, Dz
	PDMSB Sx, Dz		PCLR Dz
DCT	PDMSB Sx, Dz	DCT	PCLR Dz
DCF	PDMSB Sx, Dz	DCF	PCLR Dz
	PDMSB Sy, Dz		PADDC Sx, Sy, Dz
DCT	PDMSB Sy, Dz		PSUBC Sx, Sy, Dz
DCF	PDMSB Sy, Dz		PCMP Sx, Sy
	PINC Sx, Dz		PABS Sx, Dz
DCT	PINC Sx, Dz		PABS Sy, Dz
DCF	PINC Sx, Dz		PRND Sx, Dz
	PINC Sy, Dz		PRND Sy, Dz
DCT	PINC Sy, Dz		
DCF	PINC Sy, Dz		

##### パイプライン



## 動作説明

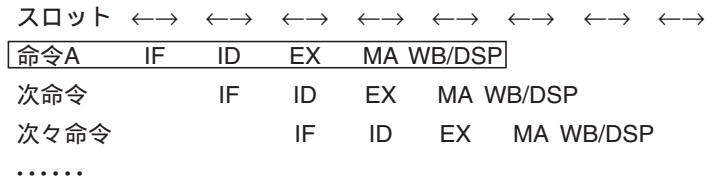
パイプラインは、IF、ID、EX、MA、WB/DSP の 5 段です。条件付き演算命令では条件が不成立の場合、WB/DSP ステージが無動作（ノー・オペレーション）となりますが、パイプラインは変わりません。

## (2) ALU 論理演算命令

## 命令の種類

	POR Sx, Sy, Dz
DCT	POR Sx, Sy, Dz
DCF	POR Sx, Sy, Dz
	PAND Sx, Sy, Dz
DCT	PAND Sx, Sy, Dz
DCF	PAND Sx, Sy, Dz
	PXOR Sx, Sy, Dz
DCT	PXOR Sx, Sy, Dz
DCF	PXOR Sx, Sy, Dz

## パイプライン



## 動作説明

パイプラインは、IF、ID、EX、MA、WB/DSP の 5 段です。条件付き演算命令では条件が不成立の場合、WB/DSP ステージが無動作（ノー・オペレーション）となりますが、パイプラインは変わりません。

## 10. パイプライン動作

---

### (3) ALU 論理演算命令

#### 命令の種類

	PSHA Sx, Sy, Dz
DCT	PSHA Sx, Sy, Dz
DCF	PSHA Sx, Sy, Dz
	PSHA #Imm, Dz
	PSHL Sx, Sy, Dz
DCT	PSHL Sx, Sy, Dz
DCF	PSHL Sx, Sy, Dz
	PSHL #Imm, Dz

#### パイプライン

スロット	←→	←→	←→	←→	←→	←→	←→	←→
命令A	IF	ID	EX	MA	WB/DSP			
次命令		IF	ID	EX	MA	WB/DSP		
次々命令			IF	ID	EX	MA	WB/DSP	
.....								

#### 動作説明

パイプラインは、IF、ID、EX、MA、WB/DSP の5段です。条件付き演算命令では条件が不成立の場合、WB/DSP ステージが無動作（ノー・オペレーション）となりますが、パイプラインは変わりません。

### (4) 符号付き乗算命令

#### 命令の種類

PMULS Se, Sf, Dg

#### パイプライン

スロット	←→	←→	←→	←→	←→	←→	←→	←→
命令A	IF	ID	EX	MA	WB/DSP			
次命令		IF	ID	EX	MA	WB/DSP		
次々命令			IF	ID	EX	MA	WB/DSP	
.....								

## 動作説明

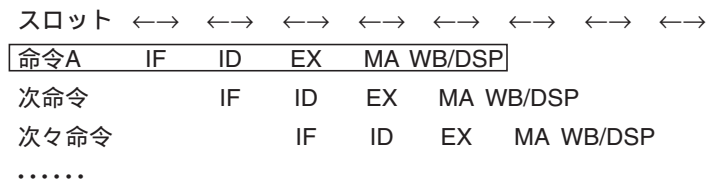
パイプラインは、IF、ID、EX、MA、WB/DSP の 5 段です。

## (5) レジスタ間転送命令

## 命令の種類

	PSTS MACH, Dz
DCT	PSTS MACH, Dz
DCF	PSTS MACH, Dz
	PSTS MACL, Dz
DCT	PSTS MACL, Dz
DCF	PSTS MACL, Dz
	PLDS Dz, MACH
DCT	PLDS Dz, MACH
DCF	PLDS Dz, MACH
	PLDS Dz, MACL
DCT	PLDS Dz, MACL
DCF	PLDS Dz, MACL

## パイプライン



## 動作説明

パイプラインは、IF、ID、EX、MA、WB/DSP の 5 段です。条件付き演算命令では条件が不成立の場合、WB/DSP ステージが無動作（ノー・オペレーション）となりますが、パイプラインは変わりません。この命令と並列に MOVX.W、MOV.S.W または MOV.S.L でメモリロードを行うと、競合が発生します。また、この命令の直後に MOVX.W、MOV.S.W または MOV.S.L でメモリストアを行うと、競合が発生します。（「10.2.7 DSP レジスタ間転送とメモリ・ロード/ストア動作の競合」を参照）。



# 付 録

## A. 命令コード

### A.1 アドレッシングモード別命令セット

表 A.1 アドレッシングモード別命令セット

アドレッシングモード	区分	命令の例	種類
オペランドなし		NOP	11 ( 11 )
レジスタ直接	デスティネーションオペランドのみ	MOVT Rn	23 ( 18 )
	ソースとデスティネーションオペランド	ADD Rm,Rn	44 ( 36 )
	コントロールレジスタまたはシステムレジスタへの転送	LDC Rm,SR	19 ( 16 )
	コントロールレジスタまたはシステムレジスタからの転送	STS MACH,Rn	19 ( 16 )
レジスタ間接	デスティネーションオペランドのみ	JMP @Rn	4 ( 4 )
	レジスタ直接とのデータ転送	MOV.L Rm,@Rn	8 ( 6 )
ポストインクリメント レジスタ間接	積和演算	MAC.W @Rm+,@Rn+	2 ( 2 )
	レジスタ直接からのデータ転送	MOV.L @Rm+,Rn	4 ( 3 )
	コントロールレジスタまたはシステムレジスタへのロード	LDC.L @Rm+,SR	18 ( 16 )
プリデクリメント レジスタ間接	レジスタ直接からのデータ転送	MOV.L Rm,@-Rn	4 ( 3 )
	コントロールレジスタまたはシステムレジスタからのストア	STC.L SR,@-Rn	18 ( 16 )
ディスプレイースメント 付きレジスタ間接	レジスタ直接とのデータ転送	MOV.L Rm,@(disp,Rn)	6 ( 6 )
インデックス付き レジスタ間接	レジスタ直接とのデータ転送	MOV.L Rm,@(R0,Rn)	8 ( 6 )
ディスプレイースメント 付き GBR 間接	レジスタ直接とのデータ転送	MOV.L R0,@(disp,GBR)	6 ( 6 )
インデックス付き GBR 間接	イミディエイトデータの転送	AND.B #imm,@(R0,GBR)	4 ( 4 )
ディスプレイースメント 付き PC 相対	レジスタ直接へのデータ転送	MOV.L @(disp,PC),Rn	3 ( 3 )
Rn を用いた PC 相対	分岐命令	BRAF Rn	2 ( 2 )
PC 相対	分岐命令	BRA label	6 ( 6 )
イミディエイト	レジスタへロード	FLDI0 FRn	2 ( 0 )
	レジスタ直接との算術論理演算	ADD #imm,Rn	7 ( 7 )
	例外処理ベクタの指定	TRAPA #imm	1 ( 1 )
			計 219 ( 188 )

【注】 ( ) の外の数字は SH-3E の、 ( ) 内の数字は SH-3 の命令数を示します。

## (1) オペランドなし

表 A.2 オペランドなし

命令	動作	命令コード	実行 ステート	Tビット
CLRS	0→S	0000000001001000	1	
CLRT	0→T	0000000000001000	1	0
CLRMAC	0→MACH,MACL	0000000000101000	1	
DIV0U	0→M/Q/T	0000000000011001	1	0
LDTLB	PTEH/PTEL→TLB	0000000000111000	1	
NOP	無操作	0000000000001001	1	
RTE	遅延分岐、SSR/SPC→SR/PC	0000000000101011	4	
RTS	遅延分岐、PR→PC	0000000000001011	2	
SETS	1→S	0000000001011000	1	
SETT	1→T	0000000000011000	1	1
SLEEP	スリープ	0000000000011011	4	



## (2) レジスタ直接

表 A.3 デスティネーションオペランドのみ

命令	動作	命令コード	実行 ステート	Tビット
CMP/PL Rn	Rn>0 とき 1→T	0100nnnn00010101	1	比較結果
CMP/PZ Rn	Rn 0 とき 1→T	0100nnnn00010001	1	比較結果
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
FABS FRn*	abs(FRn)→FRn	1111nnnn01011101	1	
FLOAT FPUL, FRn*	(float)FPUL→FRn	1111nnnn00101101	1	
FNEG FRn*	-1.0 × FRn→FRn	1111nnnn01001101	1	
FSQRT FRn*	sqrt(FRn)→FRn	1111nnnn01101101	13	
FTRC FRm, FPUL*	(long)FRm→FPUL	1111nnnn00111101	1	
MOVT Rn	T→Rn	0000nnnn00101001	1	
ROTL Rn	T←Rn←MSB	0100nnnn00000100	1	MSB
ROTR Rn	LSB→Rn→T	0100nnnn00000101	1	LSB
ROTCL Rn	T←Rn←T	0100nnnn00100100	1	MSB
ROTCR Rn	T→Rn→T	0100nnnn00100101	1	LSB
SHAL Rn	T←Rn←0	0100nnnn00100000	1	MSB
SHAR Rn	MSB→Rn→T	0100nnnn00100001	1	LSB
SHLL Rn	T←Rn←0	0100nnnn00000000	1	MSB
SHLR Rn	0→Rn→T	0100nnnn00000001	1	LSB
SHLL2 Rn	Rn<<2 → Rn	0100nnnn00001000	1	
SHLR2 Rn	Rn>>2 → Rn	0100nnnn00001001	1	
SHLL8 Rn	Rn<<8 → Rn	0100nnnn00011000	1	
SHLR8 Rn	Rn>>8 → Rn	0100nnnn00011001	1	
SHLL16 Rn	Rn<<16 → Rn	0100nnnn00101000	1	
SHLR16 Rn	Rn>>16 → Rn	0100nnnn00101001	1	

【注】 \* 浮動小数点数演算命令または FPU に関連する CPU 命令です。これらの命令は SH-3E でのみ使用可能です。

表 A.4 ソースとデスティネーションオペランド

命令	動作	命令コード	実行 ステート	Tビット
ADD Rm,Rn	$Rn+Rm \rightarrow Rn$	0011nnnnmmmm1100	1	
ADDC Rm,Rn	$Rn+Rm+T \rightarrow Rn$ , キャリ $\rightarrow T$	0011nnnnmmmm1110	1	キャリ
ADDV Rm,Rn	$Rn+Rm \rightarrow Rn$ , オーバフロー $\rightarrow T$	0011nnnnmmmm1111	1	オーバ フロー
AND Rm,Rn	$Rn \& Rm \rightarrow Rn$	0010nnnnmmmm1001	1	
CMP/EQ Rm,Rn	$Rn=Rm$ のとき $1 \rightarrow T$	0011nnnnmmmm0000	1	比較結果
CMP/HS Rm,Rn	無符号で $Rn > Rm$ のとき $1 \rightarrow T$	0011nnnnmmmm0010	1	比較結果
CMP/GE Rm,Rn	有符号で $Rn > Rm$ のとき $1 \rightarrow T$	0011nnnnmmmm0011	1	比較結果
CMP/HL Rm,Rn	無符号で $Rn > Rm$ のとき $1 \rightarrow T$	0011nnnnmmmm0110	1	比較結果
CMP/GT Rm,Rn	有符号で $Rn > Rm$ のとき $1 \rightarrow T$	0011nnnnmmmm0111	1	比較結果
CMP/STR Rm,Rn	いずれかのバイトが等しいとき $1 \rightarrow T$	0010nnnnmmmm1100	1	比較結果
DIV1 Rm,Rn	1ステップ除算 ( $Rn \div Rm$ )	0011nnnnmmmm0100	1	計算結果
DIV0S Rm,Rn	$Rn$ の MSB $\rightarrow Q$ , $Rm$ の MSB $\rightarrow M$ , $M^{\wedge}Q \rightarrow T$	0010nnnnmmmm0111	1	計算結果
DMULS.L Rm,Rn	符号付きで $Rn \times Rm \rightarrow$ MACH, MACL	0011nnnnmmmm1101	$2(\sim 5)^{*2}$	
DMULU.L Rm,Rn	符号なしで $Rn \times Rm \rightarrow$ MACH, MACL	0011nnnnmmmm0101	$2(\sim 5)^{*2}$	
EXTS.B Rm,Rn	$Rm$ をバイトから符号拡張 $\rightarrow Rn$	0110nnnnmmmm1110	1	
EXTS.W Rm,Rn	$Rm$ をワードから符号拡張 $\rightarrow Rn$	0110nnnnmmmm1111	1	
EXTU.B Rm,Rn	$Rm$ をバイトからゼロ拡張 $\rightarrow Rn$	0110nnnnmmmm1100	1	
EXTU.W Rm,Rn	$Rm$ をワードからゼロ拡張 $\rightarrow Rn$	0110nnnnmmmm1101	1	
FADD FRm,FRn <sup>*1</sup>	$FRm + FRn \rightarrow FRn$	1111nnnnmmmm0000	1	
FCMP/EQ FRm,FRn <sup>*1</sup>	$FRn = FRm$ , $1 \rightarrow T$	1111nnnnmmmm0100	1	比較結果
FCMP/GT FRm,FRn <sup>*1</sup>	$FRn > FRm$ , $1 \rightarrow T$	1111nnnnmmmm0101	1	比較結果
FDIV FRm,FRn <sup>*1</sup>	$FRn / FRm \rightarrow FRm$	1111nnnnmmmm0011	13	
FMAC FR0,FRm FRn <sup>*1</sup>	$(FR0 \times FRm) + FRn \rightarrow FRn$	1111nnnnmmmm1110	1	
FMOV FRm,FRn <sup>*1</sup>	$FRm \rightarrow FRn$	1111nnnnmmmm1100	1	
FMUL FRm,FRn <sup>*1</sup>	$FRn \times FRm \rightarrow FRn$	1111nnnnmmmm0010	1	
FSUB FRm,FRn <sup>*1</sup>	$FRn - FRm \rightarrow FRn$	1111nnnnmmmm0001	1	
MOV Rm,Rn	$Rm \rightarrow Rn$	0110nnnnmmmm0011	1	
MUL.L Rm,Rn	$Rn \times Rm \rightarrow MAC$	0000nnnnmmmm0111	$2(\sim 5)^{*2}$	
MULS.W Rm,Rn	符号付きで $Rn \times Rm \rightarrow MAC$	0010nnnnmmmm1111	$1(\sim 3)^{*2}$	
MULU.W Rm,Rn	符号なしで $Rn \times Rm \rightarrow MAC$	0010nnnnmmmm1110	$1(\sim 3)^{*2}$	
NEG Rm,Rn	$0-Rm \rightarrow Rn$	0110nnnnmmmm1011	1	
NEGC Rm,Rn	$0-Rm-T \rightarrow Rn$ , ボロ $\rightarrow T$	0110nnnnmmmm1010	1	ボロ
NOT Rm,Rn	$\sim Rm \rightarrow Rn$	0110nnnnmmmm0111	1	
OR Rm,Rn	$Rn   Rm \rightarrow Rn$	0010nnnnmmmm1011	1	
SHAD Rm,Rn	$Rn < 0$ のとき、 $Rn \ll Rm \rightarrow Rn$ $Rn < 0$ のとき、 $Rn \gg Rm \rightarrow (MSB \rightarrow) Rn$	0100nnnnmmmm1100	1	

命令		動作	命令コード	実行 ステート	Tビット
SHLD	Rm,Rn	Rn 0 のとき、 $Rn \ll Rm \rightarrow Rn$ Rn<0 のとき、 $Rn \gg Rm \rightarrow (0 \rightarrow)Rn$	0100nnnnmmmm1101	1	
SUB	Rm,Rn	$Rn - Rm \rightarrow Rn$	0011nnnnmmmm1000	1	
SUBC	Rm,Rn	$Rn - Rm - T \rightarrow Rn$ , ボロー $\rightarrow T$	0011nnnnmmmm1010	1	ボロー
SUBV	Rm,Rn	$Rn - Rm \rightarrow Rn$ , アンダフロー $\rightarrow T$	0011nnnnmmmm1011	1	アンダ フロー
SWAP.B	Rm,Rn	Rm $\rightarrow$ 下位 2 バイトの上下バイト交 換 $\rightarrow Rn$	0110nnnnmmmm1000	1	
SWAP.W	Rm,Rn	Rm $\rightarrow$ 上下ワード交換 $\rightarrow Rn$	0110nnnnmmmm1001	1	
TST	Rm,Rn	Rn & Rm, 結果が 0 のとき 1 $\rightarrow T$	0010nnnnmmmm1000	1	テスト結果
XOR	Rm,Rn	$Rn \wedge Rm \rightarrow Rn$	0010nnnnmmmm1010	1	
XTRCT	Rm,Rn	Rm:Rn の中央 32 ビット $\rightarrow Rn$	0010nnnnmmmm1101	1	

【注】 \*1 浮動小数点数演算命令または FPU に関連する CPU 命令です。これらの命令は SH-3E でのみ使用可能です。

\*2 通常最小実行ステートを示します。前後の命令との競合関係により実行ステート数は変わります。

表 A.5 コントロールレジスタまたはシステムレジスタへの転送

命令	動作	命令コード	実行 ステート	T ビット	
FLDS	FRm, FPUL* <sup>1</sup>	FRm→FPUL	1111mmmm00011101	1	
LDC	Rm,SR	Rm→SR	0100mmmm00001110	5	LSB
LDC	Rm,GBR	Rm→GBR	0100mmmm00011110	1/3* <sup>2</sup>	
LDC	Rm,VBR	Rm→VBR	0100mmmm00101110	1/3* <sup>2</sup>	
LDC	Rm,SSR	Rm→SSR	0100mmmm00111110	1/3* <sup>2</sup>	
LDC	Rm,SPC	Rm→SPC	0100mmmm01001110	1/3* <sup>2</sup>	
LDC	Rm,MOD* <sup>3</sup>	Rm→MOD	0100mmmm01011110	3	
LDC	Rm,RE* <sup>3</sup>	Rm→RE	0100mmmm01111110	3	
LDC	Rm,RS* <sup>3</sup>	Rm→RS	0100mmmm01101110	3	
LDC	Rm,R0_BANK	Rm→R0_BANK	0100mmmm10001110	1/3* <sup>2</sup>	
LDC	Rm,R1_BANK	Rm→R1_BANK	0100mmmm10011110	1/3* <sup>2</sup>	
LDC	Rm,R2_BANK	Rm→R2_BANK	0100mmmm10101110	1/3* <sup>2</sup>	
LDC	Rm,R3_BANK	Rm→R3_BANK	0100mmmm10111110	1/3* <sup>2</sup>	
LDC	Rm,R4_BANK	Rm→R4_BANK	0100mmmm11001110	1/3* <sup>2</sup>	
LDC	Rm,R5_BANK	Rm→R5_BANK	0100mmmm11011110	1/3* <sup>2</sup>	
LDC	Rm,R6_BANK	Rm→R6_BANK	0100mmmm11101110	1/3* <sup>2</sup>	
LDC	Rm,R7_BANK	Rm→R7_BANK	0100mmmm11111110	1/3* <sup>2</sup>	
LDS	Rm, FPSCR* <sup>1</sup>	Rm→FPSCR	0100mmmm01101010	1	
LDS	Rm, FPUL* <sup>1</sup>	Rm→FPUL	0100mmmm01011010	1	
LDS	Rm,MACH	Rm→MACH	0100mmmm00001010	1	
LDS	Rm,MACL	Rm→MACL	0100mmmm00011010	1	
LDS	Rm,PR	Rm→PR	0100mmmm00101010	1	
LDS	Rm,DSR* <sup>3</sup>	Rm→DSR	0100mmmm01101010	1	
LDS	Rm,A0* <sup>3</sup>	Rm→A0	0100mmmm01111010	1	
LDS	Rm,X0* <sup>3</sup>	Rm→X0	0100mmmm10001010	1	
LDS	Rm,X1* <sup>3</sup>	Rm→X1	0100mmmm10011010	1	
LDS	Rm,Y0* <sup>3</sup>	Rm→Y0	0100mmmm10101010	1	
LDS	Rm,Y1* <sup>3</sup>	Rm→Y1	0100mmmm10111010	1	
SETRC	Rm* <sup>3</sup>	Rm の下位 12 ビット →RC(SR のビット 27~ 16)、繰り返し制御フラグ →RF1、RF0	0100mmmm00010100	3	

【注】 \*1 浮動小数点数演算命令または FPU に関連する CPU 命令です。これらの命令は SH-3E でのみ使用可能です。

\*2 SH3-DSP では 3 ステートになります。

\*3 DSP 機能をサポートするための CPU 命令です。これらの命令は SH3-DSP でのみ使用可能です。

表 A.6 コントロールレジスタまたはシステムレジスタからの転送

命令	動作	命令コード	実行 ステート	T ビット
FSTS FPUL, FRn* <sup>1</sup>	FPUL→FRn	1111nnnn01011010	1	
STC SR,Rn	SR→Rn	0000nnnn00000010	1	
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 MOD,Rn* <sup>2</sup>	MOD→Rn	0000nnnn01010010	1	
STC RE,Rn* <sup>2</sup>	RE→Rn	0000nnnn01110010	1	
STC RS,Rn* <sup>2</sup>	RS→Rn	0000nnnn01100010	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	
STS FPSCR, Rn* <sup>1</sup>	FPSCR→Rn	1111nnnn01101010	1	
STS FPUL, Rn* <sup>1</sup>	FPUL→Rn	1111nnnn01011010	1	
STS MACH,Rn	MACH→Rn	0000nnnn01011010	1	
STS MACL,Rn	MACL→Rn	0000nnnn00011010	1	
STS PR,Rn	PR→Rn	0000nnnn00101010	1	
STS DSR,Rn* <sup>2</sup>	DSR→Rn	0000nnnn01101010	1	
STS A0,Rn* <sup>2</sup>	A0→Rn	0000nnnn01111010	1	
STS X0,Rn* <sup>2</sup>	X0→Rn	0000nnnn10001010	1	
STS X1,Rn* <sup>2</sup>	X1→Rn	0000nnnn10011010	1	
STS Y0,Rn* <sup>2</sup>	Y0→Rn	0000nnnn10101010	1	
STS Y1,Rn* <sup>2</sup>	Y1→Rn	0000nnnn10111010	1	

【注】 \*1 浮動小数点数演算命令または FPU に関連する CPU 命令です。これらの命令は SH-3E でのみ使用可能です。

\*2 DSP 機能をサポートするための CPU 命令です。これらの命令は SH3-DSP でのみ使用可能です。

## (3) レジスタ間接

表 A.7 ソースオペランドのみ

命令	動作	命令コード	実行 ステート	Tビット
JMP @Rm	遅延分岐、Rm→PC	0100mmmm00101011	2	
JSR @Rm	遅延分岐、PC→Rm, Rm→PC	0100mmmm00001011	2	
PREF @Rm	(Rn)→キャッシュ	0000mmmm10000011	1/2*	

【注】 \* SH3-DSP では 2 ステートになります。

表 A.8 デスティネーションオペランドのみ

命令	動作	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, 1→MSB of (Rn)	0100nnnn00011011	3/4*	テスト結果

【注】 \* SH3-DSP では 4 ステートになります。

表 A.9 レジスタ直接とのデータ転送

命令	動作	命令コード	実行 ステート	Tビット
FMOV.S FRm, @Rn*	FRm→(FRn)	1111nnnnmmmm1010	1	
FMOV.S @Rm, FRn*	(Rm)→FRn	1111nnnnmmmm1000	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	

【注】 \* 浮動小数点数演算命令または FPU に関連する CPU 命令です。これらの命令は SH-3E でのみ使用可能です。

## (4) ポストインクリメントレジスタ間接

表 A.10 積和演算

命令	動作	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	2(~5)*	
MAC.W @Rm+,@Rn+	符号付きで (Rn) × (Rm)+MAC→MAC,Rn+2→Rn, Rm+2→Rn	0100nnnnmmmm1111	2(~5)*	

【注】 \* 通常最小実行ステートを示します。前後の命令との競合関係により実行ステート数は変わります。

表 A.11 レジスタ直接からのデータ転送

命令	動作	命令コード	実行 ステート	Tビット
FMOV.S @Rm+,FRn*	(Rm)→FRn, Rm + 4→Rm	1111nnnnmmmm1001	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	

【注】 \* 浮動小数点数演算命令または FPU に関連する CPU 命令です。これらの命令は SH-3E でのみ使用可能です。

表 A.12 コントロールレジスタまたはシステムレジスタへのロード

命令	動作	命令コード	実行 ステート	T ビット
LDC.L @Rm+,SR	(Rm)→SR, Rm+4→Rm	0100mmmm00000111	7	LSB
LDC.L @Rm+,GBR	(Rm)→GBR, Rm+4→Rm	0100mmmm00010111	1/5* <sup>2</sup>	
LDC.L @Rm+,VBR	(Rm)→VBR, Rm+4→Rm	0100mmmm00100111	1/5* <sup>2</sup>	
LDC.L @Rm+,SSR	(Rm)→SSR, Rm+4→Rm	0100mmmm00110111	1/5* <sup>2</sup>	
LDC.L @Rm+,SPC	(Rm)→SPC, Rm+4→Rm	0100mmmm01000111	1/5* <sup>2</sup>	
LDC.L @Rm+,MOD* <sup>3</sup>	(Rm)→MOD, Rm+4→Rm	0100mmmm01010111	5	
LDC.L @Rm+,RE* <sup>3</sup>	(Rm)→RE, Rm+4→Rm	0100mmmm01110111	5	
LDC.L @Rm+,RS* <sup>3</sup>	(Rm)→RS, Rm+4→Rm	0100mmmm01100111	5	
LDC.L @Rm+,R0_BANK	(Rm)→R0_BANK, Rm+4→Rm	0100mmmm10000111	1/5* <sup>2</sup>	
LDC.L @Rm+,R1_BANK	(Rm)→R1_BANK, Rm+4→Rm	0100mmmm10010111	1/5* <sup>2</sup>	
LDC.L @Rm+,R2_BANK	(Rm)→R2_BANK, Rm+4→Rm	0100mmmm10100111	1/5* <sup>2</sup>	
LDC.L @Rm+,R3_BANK	(Rm)→R3_BANK, Rm+4→Rm	0100mmmm10110111	1/5* <sup>2</sup>	
LDC.L @Rm+,R4_BANK	(Rm)→R4_BANK, Rm+4→Rm	0100mmmm11000111	1/5* <sup>2</sup>	
LDC.L @Rm+,R5_BANK	(Rm)→R5_BANK, Rm+4→Rm	0100mmmm11010111	1/5* <sup>2</sup>	
LDC.L @Rm+,R6_BANK	(Rm)→R6_BANK, Rm+4→Rm	0100mmmm11100111	1/5* <sup>2</sup>	
LDC.L @Rm+,R7_BANK	(Rm)→R7_BANK, Rm+4→Rm	0100mmmm11110111	1/5* <sup>2</sup>	
LDS.L @Rm+,FPSCR* <sup>1</sup>	(Rm)→FPSCR, Rm+4→Rm	0100mmmm01100110	1	
LDS.L @Rm+,FPUL* <sup>1</sup>	(Rm)→FPUL, Rm+4→Rm	0100mmmm01010110	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	
LDS.L @Rm+,DSR* <sup>3</sup>	(Rm)→DSR, Rm+4→Rm	0100mmmm01100110	1	
LDS.L @Rm+,AO* <sup>3</sup>	(Rm)→AO, Rm+4→Rm	0100mmmm01110110	1	
LDS.L @Rm+,X0* <sup>3</sup>	(Rm)→X0, Rm+4→Rm	0100mmmm10000110	1	
LDS.L @Rm+,X1* <sup>3</sup>	(Rm)→X1, Rm+4→Rm	0100mmmm10010110	1	
LDS.L @Rm+,Y0* <sup>3</sup>	(Rm)→Y0, Rm+4→Rm	0100mmmm10100110	1	
LDS.L @Rm+,Y1* <sup>3</sup>	(Rm)→Y1, Rm+4→Rm	0100mmmm10110110	1	

【注】 \*1 浮動小数点数演算命令または FPU に関連する CPU 命令です。これらの命令は SH-3E でのみ使用可能です。

\*2 SH3-DSP では 5 ステートになります。

\*3 DSP 機能をサポートするための CPU 命令です。これらの命令は SH3-DSP でのみ使用可能です。



## (5) プリデクリメントレジスタ間接

表 A.13 レジスタ直接からのデータ転送

命令	動作	命令コード	実行 ステート	Tビット
FMOV.S FRm,@-Rn* <sup>1</sup>	Rn-4→Rn, FRm→(Rn)	1111nnnnmmmm1011	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	

表 A.14 コントロールレジスタまたはシステムレジスタからのストア

命令	動作	命令コード	実行 ステート	Tビット
STC.L SR,@-Rn	Rn-4→Rn, SR→(Rn)	0100nnnn00000011	1/2* <sup>2</sup>	
STC.L GBR,@-Rn	Rn-4→Rn, GBR→(Rn)	0100nnnn00010011	1/2* <sup>2</sup>	
STC.L VBR,@-Rn	Rn-4→Rn, VBR→(Rn)	0100nnnn00100011	1/2* <sup>2</sup>	
STC.L SSR,@-Rn	Rn-4→Rn, SSR→(Rn)	0100nnnn00110011	1/2* <sup>2</sup>	
STC.L SPC,@-Rn	Rn-4→Rn, SPC→(Rn)	0100nnnn01000011	1/2* <sup>2</sup>	
STC.L MOD,@-Rn* <sup>3</sup>	Rn - 4→Rn, MOD→(Rn)	0100nnnn01010011	2	
STC.L RE,@-Rn* <sup>3</sup>	Rn - 4→Rn, RE→(Rn)	0100nnnn01110011	2	
STC.L RS,@-Rn* <sup>3</sup>	Rn - 4→Rn, RS→(Rn)	0100nnnn01100011	2	
STC.L R0_BANK,@-Rn	Rn-4→Rn, R0_BANK→(Rn)	0100nnnn10000011	2	
STC.L R1_BANK,@-Rn	Rn-4→Rn, R1_BANK→(Rn)	0100nnnn10010011	2	
STC.L R2_BANK,@-Rn	Rn-4→Rn, R2_BANK→(Rn)	0100nnnn10100011	2	
STC.L R3_BANK,@-Rn	Rn-4→Rn, R3_BANK→(Rn)	0100nnnn10110011	2	
STC.L R4_BANK,@-Rn	Rn-4→Rn, R4_BANK→(Rn)	0100nnnn11000011	2	
STC.L R5_BANK,@-Rn	Rn-4→Rn, R5_BANK→(Rn)	0100nnnn11010011	2	
STC.L R6_BANK,@-Rn	Rn-4→Rn, R6_BANK→(Rn)	0100nnnn11100011	2	
STC.L R7_BANK,@-Rn	Rn-4→Rn, R7_BANK→(Rn)	0100nnnn11110011	2	
STS.L FPSCR,@-Rn* <sup>1</sup>	Rn-4→Rn, FPSCR→(Rn)	0100nnnn01100010	1	
STS.L FPUL,@-Rn* <sup>1</sup>	Rn-4→Rn, FPUL→(Rn)	0100nnnn01010010	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	
STS.L DSR,@-Rn* <sup>3</sup>	Rn - 4→Rn, DSR→(Rn)	0100nnnn01100010	1	
STS.L A0,@-Rn* <sup>3</sup>	Rn - 4→Rn, A0→(Rn)	0100nnnn01110010	1	
STS.L X0,@-Rn* <sup>3</sup>	Rn - 4→Rn, X0→(Rn)	0100nnnn10000010	1	
STS.L X1,@-Rn* <sup>3</sup>	Rn - 4→Rn, X1→(Rn)	0100nnnn10010010	1	
STS.L Y0,@-Rn* <sup>3</sup>	Rn - 4→Rn, Y0→(Rn)	0100nnnn10100010	1	
STS.L Y1,@-Rn* <sup>3</sup>	Rn - 4→Rn, Y1→(Rn)	0100nnnn10110010	1	

【注】 \*1 浮動小数点数演算命令またはFPUに関連するCPU命令です。これらの命令はSH-3Eでのみ使用可能です。

\*2 SH3-DSPでは2ステートになります。

\*3 DSP機能をサポートするためのCPU命令です。これらの命令はSH3-DSPでのみ使用可能です。

## (6) ディスプレースメント付きレジスタ間接

表 A.15 ディスプレースメント付きレジスタ間接

命令	動作	命令コード	実行 ステート	Tビット
MOV.B R0,@(disp,Rn)	R0→(disp+Rn)	10000000nnnnndddd	1	
MOV.W R0,@(disp,Rn)	R0→(disp+Rn)	10000001nnnnndddd	1	
MOV.L Rm,@(disp,Rn)	Rm→(disp+Rn)	0001nnnnmmmmndddd	1	
MOV.B @(disp,Rm),R0	(disp+Rm)→符号拡張→R0	10000100mmmmndddd	1	
MOV.W @(disp,Rm),R0	(disp+Rm)→符号拡張→R0	10000101mmmmndddd	1	
MOV.L @(disp,Rm),Rn	(disp+Rm)→Rn	0101nnnnmmmmndddd	1	

## (7) インデックス付きレジスタ間接

表 A.16 インデックス付きレジスタ間接

命令	動作	命令コード	実行 ステート	Tビット
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	
FMOV.S FRm,@(R0,Rn)*	FRm→(R0+Rn)	1111nnnnmmmm0111	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	
FMOV.S @(R0,FRm),FRm*	(R0+Rm)→FRm	1111nnnnmmmm0110	1	

【注】 \* 浮動小数点数演算命令または FPU に関連する CPU 命令です。これらの命令は SH-3E でのみ使用可能です。

## (8) ディスプレースメント付き GBR 間接

表 A.17 ディスプレースメント付き GBR 間接

命令	動作	命令コード	実行 ステート	Tビット
MOV.B R0,@(disp,GBR)	R0→(disp+GBR)	11000000ddddddd	1	
MOV.W R0,@(disp,GBR)	R0→(disp×2+GBR)	11000001ddddddd	1	
MOV.L R0,@(disp,GBR)	R0→(disp×4+GBR)	11000010ddddddd	1	
MOV.B @(disp,GBR),R0	(disp+GBR)→符号拡張→R0	11000100ddddddd	1	
MOV.W @(disp,GBR),R0	(disp×2+GBR)→符号拡張→R0	11000101ddddddd	1	
MOV.L @(disp,GBR),R0	(disp×4+GBR)→R0	11000110ddddddd	1	

## (9) インデックス付き GBR 間接

表 A.18 インデックス付き GBR 間接

命令	動作	命令コード	実行 ステート	Tビット
AND.B #imm,@(R0,GBR)	(R0+GBR) & imm→(R0+GBR)	11001101iiiiiii	3	
OR.B #imm,@(R0,GBR)	(R0+GBR)   imm→(R0+GBR)	11001111iiiiiii	3	
TST.B #imm,@(R0,GBR)	(R0+GBR)&imm,結果が0のとき 1→T	11001100iiiiiii	3	テスト 結果
XOR.B #imm,@(R0,GBR)	(R0+GBR) ^ imm→(R0+GBR)	11001110iiiiiii	3	

## (10) ディスプレースメント付き PC 相対

表 A.19 ディスプレースメント付き PC 相対

命令	動作	命令コード	実行 ステート	Tビット
MOV.W @(disp,PC),Rn	(disp × 2+PC)→符号拡張→Rn	1001nnnnddddddd	1	
MOV.L @(disp,PC),Rn	(disp × 4+PC)→Rn	1101nnnnddddddd	1	
MOVA @(disp,PC),R0	disp × 4+PC→R0	11000111ddddddd	1	
LDRS @(disp,pc)*	disp × 2+PC→RS	10001100ddddddd	3	
LDRE @(disp,pc)*	disp × 2+PC→RE	10001110ddddddd	3	

【注】 \* SH3-DSP の命令

## (11) PC 相対

表 A.20 Rn を用いた PC 相対

命令	動作	命令コード	実行 ステート	Tビット
BRAF Rm	遅延分岐、Rm+PC→PC	0000mmmm00100011	2	
BSRF Rm	遅延分岐、PC→PR, Rm+PC→PC	0000mmmm00000011	2	

表 A.21 PC 相対

命令	動作	命令コード	実行 ステート	Tビット
BF label	T=0 のとき disp × 2+PC→PC, T=1 のとき nop	10001011ddddddd	3/1*	
BF/S label	遅延分岐、T=0 のとき disp × 2+PC→PC, T=1 のとき nop	10001111ddddddd	2/1*	
BT label	T=1 のとき disp × 2+PC→PC, T=0 のとき nop	10001001ddddddd	3/1*	
BT/S label	遅延分岐、T=1 のとき disp × 2+PC→PC, T=0 のとき nop	10001101ddddddd	2/1*	
BRA label	遅延分岐、disp × 2+PC→PC	1010ddddddddddd	2	
BSR label	遅延分岐、PC→PR, disp × 2+PC→PC	1011ddddddddddd	2	

【注】 \* 分岐しないときは 1 ステートになります。

## (12) イミディエイト

表 A.22 レジスタへのロード

命令	動作	命令コード	実行 ステート	Tビット
FLDI0 FRn*	0.0→FRn	1111nnnn10001101	1	
FLDI1 FRn*	1.0→FRn	1111nnnn10011101	1	

【注】 \* 浮動小数点数演算命令または FPU に関連する CPU 命令です。これらの命令は SH-3E でのみ使用可能です。

表 A.23 イミディエイトとのレジスタ直接算術論理演算

命令	動作	命令コード	実行 ステート	Tビット
ADD #imm,Rn	Rn+#imm→Rn	0111nnnniiiiiiii	1	
AND #imm,R0	R0 & imm → R0	11001001iiiiiiii	1	
CMP/EQ #imm,R0	R0=imm のとき 1→T	10001000iiiiiiii	1	比較結果
MOV #imm,Rn	#imm→符号拡張→Rn	1110nnnniiiiiiii	1	
OR #imm,R0	R0   imm → R0	11001011iiiiiiii	1	
TST #imm,R0	R0 & imm, 結果が 0 のとき 1→T	11001000iiiiiiii	1	テスト 結果
XOR #imm,R0	R0 ^ imm → R0	11001010iiiiiiii	1	

表 A.24 例外処理ベクタの指定

命令	動作	命令コード	実行 ステート	Tビット
TRAPA #imm	imm→TRA, PC→SPC, SR→SSR, 1→SR.MD/BL/RB, 0x160→ EXPEVT, VBR + H'00000100→PC	11000011iiiiiiii	6/8*	

【注】 \* SH3-DSP では 8 ステートになります。

表 A.25 コントロールレジスタへのロード

命令	動作	命令コード	実行 ステート	Tビット
SETRC #imm*	imm→RC(SR[23:16]),zeros→SR[27:24]	10000010iiiiiiii	3	

【注】 \* SH3-DSP の命令

## A.2 命令形式別命令セット

表 A.25 ~ 表 A.55 に命令の命令コードと実行ステートを、命令形式別に示します。

表 A.25 命令形式別命令セット

命令形式	区分	命令の例	種類		
			SH-3	SH-3E	SH3-DSP
0 形式		NOP	11	11	11
n 形式	レジスタ直接	MOV.T Rn	18	18	18
	レジスタ直接 (コントロールレジスタまたはシステムレジスタとのストア)	STS MACH,Rn	16	18	25
	レジスタ間接	TAS @Rn	1	1	1
	プリデクリメントレジスタ間接	STC.L SR,@-Rn	16	18	25
	浮動小数点命令	FABS FRn	-	7	-
m 形式	レジスタ直接 (コントロールレジスタまたはシステムレジスタとのロード)	LDC Rm,SR	16	18	26
	Rm を用いた PC 相対	BRAF Rm	2	2	2
	レジスタ間接	JMP @Rm	2	2	2
	ポストインクリメントレジスタ間接	LDC.L @Rm+,SR	16	18	25
	浮動小数点命令	FLDS FRm,FPUL	-	2	-
nm 形式	レジスタ直接	ADD Rm,Rn	36	36	36
	レジスタ間接	MOV.L Rm,@Rn	6	6	6
	ポストインクリメントレジスタ間接 (積和演算)	MAC.W @Rm+,@Rn+	2	2	2
	ポストインクリメントレジスタ間接	MOV.L @Rm+,Rn	3	3	3
	プリデクリメントレジスタ間接	MOV.L Rm,@-Rn	3	3	3
	インデックス付きレジスタ間接	MOV.L Rm,@(R0,Rn)	6	6	6
	浮動小数点命令	FADD FRm,FRn	-	14	-
md 形式	ディスプレイースメント付きレジスタ間接	MOV.B @(disp,Rm),R0	2	2	2
nd4 形式	ディスプレイースメント付きレジスタ間接	MOV.B R0,@(disp,Rn)	2	2	2
nmd 形式	ディスプレイースメント付きレジスタ間接	MOV.L Rm,@(disp,Rn)	2	2	2
d 形式	ディスプレイースメント付き GBR 間接	MOV.L R0,@(disp,GBR)	6	6	6
	ディスプレイースメント付き PC 相対	MOVA @(disp,PC),R0	1	1	3
	PC 相対	BF label	4	4	4
d12 形式	PC 相対	BRA label	2	2	2
nd8 形式	ディスプレイースメント付き PC 相対	MOV.L @(disp,PC),Rn	2	2	2
i 形式	インデックス付き GBR 間接	AND.B #imm,@(R0,GBR)	4	4	4
	イミディエイト (レジスタ直接との算術論理演算)	AND #imm,R0	5	5	5
	イミディエイト (例外処理ベクタの指定)	TRAPA #imm	1	1	1
	コントロールレジスタへのロード	SETRC #imm	-	-	1
ni 形式	イミディエイト (レジスタ直接との算術演算とデータ転送)	ADD #imm,Rn	2	2	2
		計	187	218	227

## (1) 0 形式

表 A.26 0 形式

命令	動作	命令コード	実行 ステート	Tビット
CLRS	0→S	0000000001001000	1	
CLRT	0→T	0000000000001000	1	0
CLRMACH	0→MACH,MACL	0000000000101000	1	
DIV0U	0→M/Q/T	000000000011001	1	0
LDTLB	PTEH/PTEL→TLB	000000000111000	1	
NOP	無操作	000000000001001	1	
RTE	遅延分岐、SSR→SR, SPC→PC	000000000101011	4	
RTS	遅延分岐、PR→PC	000000000001011	2	
SETS	1→S	000000001011000	1	
SETT	1→T	000000000011000	1	1
SLEEP	スリープ	000000000011011	4*	

【注】 \* スリープ状態に移るまでのステート数です。

## (2) n 形式

表 A.27 レジスタ直接

命令	動作	命令コード	実行 ステート	Tビット
CMP/PL Rn	Rn>0 とき 1→T	0100nnnn00010101	1	比較結果
CMP/PZ Rn	Rn 0 とき 1→T	0100nnnn00010001	1	比較結果
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
MOVT Rn	T→Rn	0000nnnn00101001	1	
ROTL Rn	T←Rn←MSB	0100nnnn00000100	1	MSB
ROTR Rn	LSB→Rn→T	0100nnnn00000101	1	LSB
ROTCL Rn	T←Rn←T	0100nnnn00100100	1	MSB
ROTCR Rn	T→Rn→T	0100nnnn00100101	1	LSB
SHAL Rn	T←Rn←0	0100nnnn00100000	1	MSB
SHAR Rn	MSB→Rn→T	0100nnnn00100001	1	LSB
SHLL Rn	T←Rn←0	0100nnnn00000000	1	MSB
SHLR Rn	0→Rn→T	0100nnnn00000001	1	LSB
SHLL2 Rn	Rn<<2 → Rn	0100nnnn00001000	1	
SHLR2 Rn	Rn>>2 → Rn	0100nnnn00001001	1	
SHLL8 Rn	Rn<<8 → Rn	0100nnnn00011000	1	
SHLR8 Rn	Rn>>8 → Rn	0100nnnn00011001	1	
SHLL16 Rn	Rn<<16 → Rn	0100nnnn00101000	1	
SHLR16 Rn	Rn>>16 → Rn	0100nnnn00101001	1	

表 A.28 レジスタ直接 (コントロールレジスタまたはシステムレジスタとのストア)

命令	動作	命令コード	実行 ステート	Tビット
STC SR,Rn	SR→Rn	0000nnnn00000010	1	
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 MOD,Rn* <sup>2</sup>	MOD→Rn	0000nnnn01010010	1	
STC RE,Rn* <sup>2</sup>	RE→Rn	0000nnnn01110010	1	
STC RS,Rn* <sup>2</sup>	RS→Rn	0000nnnn01100010	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	
STS FPSCR,Rn * <sup>1</sup>	FPSCR→Rn	0000nnnn01101010	1	
STS FPUL,Rn * <sup>1</sup>	FPUL→Rn	0000nnnn01011010	1	
STS MACH,Rn	MACH→Rn	0000nnnn00001010	1	
STS MACL,Rn	MACL→Rn	0000nnnn00011010	1	
STS PR,Rn	PR→Rn	0000nnnn00101010	1	
STS DSR,Rn* <sup>2</sup>	DSR→Rn	0000nnnn01101010	1	
STS A0,Rn* <sup>2</sup>	A0→Rn	0000nnnn01111010	1	
STS X0,Rn* <sup>2</sup>	X0→Rn	0000nnnn10001010	1	
STS X1,Rn* <sup>2</sup>	X1→Rn	0000nnnn10011010	1	
STS Y0,Rn* <sup>2</sup>	Y0→Rn	0000nnnn10101010	1	
STS Y1,Rn* <sup>2</sup>	Y1→Rn	0000nnnn10111010	1	

【注】 \*<sup>1</sup> SH-3E の命令\*<sup>2</sup> SH3-DSP の命令

表 A.29 レジスタ間接

命令	動作	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, 1→MSB of (Rn)	0100nnnn00011011	3/4*	テスト結果

【注】 \* SH3-DSP では4ステートになります。

表 A.30 プリデクリメントレジスタ間接

命令	動作	命令コード	実行 状態	Tビット
STC.L SR,@-Rn	Rn-4→Rn, SR→(Rn)	0100nnnn00000011	1/2* <sup>2</sup>	
STC.L GBR,@-Rn	Rn-4→Rn, GBR→(Rn)	0100nnnn00010011	1/2* <sup>2</sup>	
STC.L VBR,@-Rn	Rn-4→Rn, VBR→(Rn)	0100nnnn00100011	1/2* <sup>2</sup>	
STC.L SSR,@-Rn	Rn-4→Rn, SSR→(Rn)	0100nnnn00110011	1/2* <sup>2</sup>	
STC.L SPC,@-Rn	Rn-4→Rn, SPC→(Rn)	0100nnnn01000011	1/2* <sup>2</sup>	
STC.L MOD,@-Rn* <sup>3</sup>	Rn - 4→Rn, MOD→(Rn)	0100nnnn01010011	2	
STC.L RE,@-Rn* <sup>3</sup>	Rn - 4→Rn, RE→(Rn)	0100nnnn01110011	2	
STC.L RS,@-Rn* <sup>3</sup>	Rn - 4→Rn, RS→(Rn)	0100nnnn01100011	2	
STC.L R0_BANK,@-Rn	Rn-4→Rn, R0_BANK→(Rn)	0100nnnn10000011	2	
STC.L R1_BANK,@-Rn	Rn-4→Rn, R1_BANK→(Rn)	0100nnnn10010011	2	
STC.L R2_BANK,@-Rn	Rn-4→Rn, R2_BANK→(Rn)	0100nnnn10100011	2	
STC.L R3_BANK,@-Rn	Rn-4→Rn, R3_BANK→(Rn)	0100nnnn10110011	2	
STC.L R4_BANK,@-Rn	Rn-4→Rn, R4_BANK→(Rn)	0100nnnn11000011	2	
STC.L R5_BANK,@-Rn	Rn-4→Rn, R5_BANK→(Rn)	0100nnnn11010011	2	
STC.L R6_BANK,@-Rn	Rn-4→Rn, R6_BANK→(Rn)	0100nnnn11100011	2	
STC.L R7_BANK,@-Rn	Rn-4→Rn, R7_BANK→(Rn)	0100nnnn11110011	2	
STS.L FPSCR,@-Rn* <sup>1</sup>	Rn-4→Rn, FPSCR→@ Rn	0100nnnn01100010	1	
STS.L FPUL,@-Rn* <sup>1</sup>	Rn-4→Rn, FPUL→@Rn	0100nnnn01010010	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	
STS.L DSR,@-Rn* <sup>3</sup>	Rn - 4→Rn, DSR→(Rn)	0100nnnn01100010	1	
STS.L A0,@-Rn* <sup>3</sup>	Rn - 4→Rn, A0→(Rn)	0100nnnn01110010	1	
STS.L X0,@-Rn* <sup>3</sup>	Rn - 4→Rn, X0→(Rn)	0100nnnn10000010	1	
STS.L X1,@-Rn* <sup>3</sup>	Rn - 4→Rn, X1→(Rn)	0100nnnn10010010	1	
STS.L Y0,@-Rn* <sup>3</sup>	Rn - 4→Rn, Y0→(Rn)	0100nnnn10100010	1	
STS.L Y1,@-Rn* <sup>3</sup>	Rn - 4→Rn, Y1→(Rn)	0100nnnn10110010	1	

【注】 \*1 SH-3E の命令

\*2 SH3-DSP では 2 ステートになります。

\*3 SH3-DSP の命令



表 A.31 浮動小数点命令 (SH-3E のみ)

命令	動作	命令コード	実行 ステート	Tビット
FABS FRn	FRn →FRn	1111nnnn01011101	1	
FLDI0 FRn	H'00000000→FRn	1111nnnn10001101	1	
FLDI1 FRn	H'3F800000→FRn	1111nnnn10011101	1	
FLOAT FPUL,FRn	(float)FPUL→FRn	1111nnnn00101101	1	
FNEG FRn	−FRn→FRn	1111nnnn01001101	1	
FSQRT FRn	FRn→FRn	1111nnnn01101101	13	
FSTS FPUL,FRn	FPUL→FRn	1111nnnn00001101	1	

## (3) m 形式

表 A.32 レジスタ直接 (コントロールレジスタまたはシステムレジスタへのロード)

命令	動作	命令コード	実行 ステート	T ビット
LDC Rm,SR	Rm→SR	0100mmmm00001110	5	LSB
LDC Rm,GBR	Rm→GBR	0100mmmm00011110	1/3* <sup>2</sup>	
LDC Rm,VBR	Rm→VBR	0100mmmm00101110	1/3* <sup>2</sup>	
LDC Rm,SSR	Rm→SSR	0100mmmm00111110	1/3* <sup>2</sup>	
LDC Rm,SPC	Rm→SPC	0100mmmm01001110	1/3* <sup>2</sup>	
LDC Rm,MOD* <sup>3</sup>	Rm→MOD	0100mmmm01011110	3	
LDC Rm,RE* <sup>3</sup>	Rm→RE	0100mmmm01111110	3	
LDC Rm,RS* <sup>3</sup>	Rm→RS	0100mmmm01101110	3	
LDC Rm,R0_BANK	Rm→R0_BANK	0100mmmm10001110	1/3* <sup>2</sup>	
LDC Rm,R1_BANK	Rm→R1_BANK	0100mmmm10011110	1/3* <sup>2</sup>	
LDC Rm,R2_BANK	Rm→R2_BANK	0100mmmm10101110	1/3* <sup>2</sup>	
LDC Rm,R3_BANK	Rm→R3_BANK	0100mmmm10111110	1/3* <sup>2</sup>	
LDC Rm,R4_BANK	Rm→R4_BANK	0100mmmm11001110	1/3* <sup>2</sup>	
LDC Rm,R5_BANK	Rm→R5_BANK	0100mmmm11011110	1/3* <sup>2</sup>	
LDC Rm,R6_BANK	Rm→R6_BANK	0100mmmm11101110	1/3* <sup>2</sup>	
LDC Rm,R7_BANK	Rm→R7_BANK	0100mmmm11111110	1/3* <sup>2</sup>	
LDS Rm,FPSCR* <sup>1</sup>	Rm→FPSCR	0100mmmm01101010	1	
LDS Rm,FPUL* <sup>1</sup>	Rm→FPUL	0100mmmm01011010	1	
LDS Rm,MACH	Rm→MACH	0100mmmm00001010	1	
LDS Rm,MACL	Rm→MACL	0100mmmm00011010	1	
LDS Rm,PR	Rm→PR	0100mmmm00101010	1	
LDS Rm,DSR* <sup>3</sup>	Rm→DSR	0100mmmm01101010	1	
LDS Rm,A0* <sup>3</sup>	Rm→A0	0100mmmm01111010	1	
LDS Rm,X0* <sup>3</sup>	Rm→X0	0100mmmm10001010	1	
LDS Rm,X1* <sup>3</sup>	Rm→X1	0100mmmm10011010	1	
LDS Rm,Y0* <sup>3</sup>	Rm→Y0	0100mmmm10101010	1	
LDS Rm,Y1* <sup>3</sup>	Rm→Y1	0100mmmm10111010	1	
SETRC Rm* <sup>3</sup>	Rm の下位 12 ビット→RC (SR のビット 27~16)、繰り返し制 御フラグ→RF1、RF0	0100mmmm00010100	3	

【注】 \*1 SH-3E の命令

\*2 SH3-DSP では 3 ステートになります。

\*3 SH3-DSP の命令

表 A.33 Rm を用いた PC 相対

命令	動作	命令コード	実行 ステート	T ビット
BRAF Rm	遅延分岐、Rm+PC→PC	0000mmmm00100011	2	
BSRF Rm	遅延分岐、PC→PR, Rm+PC→PC	0000mmmm00000011	2	

表 A.34 レジスタ間接

命令	動作	命令コード	実行 ステート	Tビット
JMP @Rm	遅延分岐、Rm→PC	0100mmmm00101011	2	
JSR @Rm	遅延分岐、PC→PR, Rm→PC	0100mmmm00001011	2	

表 A.35 ポストインクリメントレジスタ間接

命令	動作	命令コード	実行 ステート	Tビット
LDC.L @Rm+,SR	(Rm)→SR, Rm+4→Rm	0100mmmm00000111	7	LSB
LDC.L @Rm+,GBR	(Rm)→GBR, Rm+4→Rm	0100mmmm00010111	1/5 <sup>*2</sup>	
LDC.L @Rm+,VBR	(Rm)→VBR, Rm+4→Rm	0100mmmm00100111	1/5 <sup>*2</sup>	
LDC.L @Rm+,SSR	(Rm)→SSR, Rm+4→Rm	0100mmmm00110111	1/5 <sup>*2</sup>	
LDC.L @Rm+,SPC	(Rm)→SPC, Rm+4→Rm	0100mmmm01000111	1/5 <sup>*2</sup>	
LDC.L @Rm+,MOD <sup>*3</sup>	(Rm)→MOD, Rm+4→Rm	0100mmmm01010111	5	
LDC.L @Rm+,RE <sup>*3</sup>	(Rm)→RE, Rm+4→Rm	0100mmmm01110111	5	
LDC.L @Rm+,RS <sup>*3</sup>	(Rm)→RS, Rm+4→Rm	0100mmmm01100111	5	
LDC.L @Rm+,R0_BANK	(Rm)→R0_BANK, Rm+4→Rm	0100mmmm10000111	1/5 <sup>*2</sup>	
LDC.L @Rm+,R1_BANK	(Rm)→R1_BANK, Rm+4→Rm	0100mmmm10010111	1/5 <sup>*2</sup>	
LDC.L @Rm+,R2_BANK	(Rm)→R2_BANK, Rm+4→Rm	0100mmmm10100111	1/5 <sup>*2</sup>	
LDC.L @Rm+,R3_BANK	(Rm)→R3_BANK, Rm+4→Rm	0100mmmm10110111	1/5 <sup>*2</sup>	
LDC.L @Rm+,R4_BANK	(Rm)→R4_BANK, Rm+4→Rm	0100mmmm11000111	1/5 <sup>*2</sup>	
LDC.L @Rm+,R5_BANK	(Rm)→R5_BANK, Rm+4→Rm	0100mmmm11010111	1/5 <sup>*2</sup>	
LDC.L @Rm+,R6_BANK	(Rm)→R6_BANK, Rm+4→Rm	0100mmmm11100111	1/5 <sup>*2</sup>	
LDC.L @Rm+,R7_BANK	(Rm)→R7_BANK, Rm+4→Rm	0100mmmm11110111	1/5 <sup>*2</sup>	
LDS.L @Rm+,FPSCR <sup>*1</sup>	@Rm→FPSCR, Rm+4→Rm	0100nnnn01100110	1	
LDS.L @Rm+,FPUL <sup>*1</sup>	@Rm→FPUL, Rm+4→Rm	0100nnnn01010110	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	
LDS.L @Rm+,DSR <sup>*3</sup>	(Rm)→DSR, Rm+4→Rm	0100mmmm01100110	1	
LDS.L @Rm+,A0 <sup>*3</sup>	(Rm)→A0, Rm+4→Rm	0100mmmm01110110	1	
LDS.L @Rm+,X0 <sup>*3</sup>	(Rm)→X0, Rm+4→Rm	0100mmmm10000110	1	
LDS.L @Rm+,X1 <sup>*3</sup>	(Rm)→X1, Rm+4→Rm	0100mmmm10010110	1	
LDS.L @Rm+,Y0 <sup>*3</sup>	(Rm)→Y0, Rm+4→Rm	0100mmmm10100110	1	
LDS.L @Rm+,Y1 <sup>*3</sup>	(Rm)→Y1, Rm+4→Rm	0100mmmm10110110	1	

【注】 \*1 SH-3E の命令

\*2 SH3-DSP では 5 ステートになります。

\*3 SH3-DSP の命令

表 A.36 浮動小数点命令 (SH-3E のみ)

命令		動作	命令コード	実行 ステート	Tビット
FLDS	FRm,FPUL	FRm→FPUL	1111nnnn00011101	1	
FTRC	FRm,FPUL	(long)FRm→FPUL	1111nnnn00111101	1	

## (4) nm 形式

表 A.37 レジスタ直接

命令	動作	命令コード	実行 ステート	Tビット
ADD Rm,Rn	$Rn+Rm \rightarrow Rn$	0011nnnnmmmm1100	1	
ADDC Rm,Rn	$Rn+Rm+T \rightarrow Rn$ , キャリ→T	0011nnnnmmmm1110	1	キャリ
ADDV Rm,Rn	$Rn+Rm \rightarrow Rn$ , オーバフロー→T	0011nnnnmmmm1111	1	オーバ フロー
AND Rm,Rn	$Rn \& Rm \rightarrow Rn$	0010nnnnmmmm1001	1	
CMP/EQ Rm,Rn	$Rn=Rm$ のとき 1→T	0011nnnnmmmm0000	1	比較結果
CMP/HS Rm,Rn	無符号で $Rn > Rm$ のとき 1→T	0011nnnnmmmm0010	1	比較結果
CMP/GE Rm,Rn	有符号で $Rn > Rm$ のとき 1→T	0011nnnnmmmm0011	1	比較結果
CMP/HI Rm,Rn	無符号で $Rn > Rm$ のとき 1→T	0011nnnnmmmm0110	1	比較結果
CMP/GT Rm,Rn	有符号で $Rn > Rm$ のとき 1→T	0011nnnnmmmm0111	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	0010nnnnmmmm1100	1	比較結果
DIV1 Rm,Rn	1 ステップ除算 ( $Rn \div Rm$ )	0011nnnnmmmm0100	1	計算結果
DIV0S Rm,Rn	$Rn$ の MSB→Q, $Rm$ の MSB→M, $M \wedge Q \rightarrow T$	0010nnnnmmmm0111	1	計算結果
DMULS.L Rm,Rn	符号付きで $Rn \times Rm \rightarrow MACH, MACL$	0011nnnnmmmm1101	2(~5)*	
DMULU.L Rm,Rn	符号なしで $Rn \times Rm \rightarrow MACH, MACL$	0011nnnnmmmm0101	2(~5)*	
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	
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	
MOV Rm,Rn	$Rm \rightarrow Rn$	0110nnnnmmmm0011	1	
MUL.L Rm,Rn	$Rn \times Rm \rightarrow MAC$	0000nnnnmmmm0111	2(~5)*	
MULS Rm,Rn	符号付きで $Rn \times Rm \rightarrow MAC$	0010nnnnmmmm1111	1(~3)*	
MULU Rm,Rn	符号なしで $Rn \times Rm \rightarrow MAC$	0010nnnnmmmm1110	1(~3)*	
NEG Rm,Rn	$0-Rm \rightarrow Rn$	0110nnnnmmmm1011	1	
NEGC Rm,Rn	$0-Rm-T \rightarrow Rn$ , ボロ→T	0110nnnnmmmm1010	1	ボロ
NOT Rm,Rn	$\sim Rm \rightarrow Rn$	0110nnnnmmmm0111	1	
OR Rm,Rn	$Rn   Rm \rightarrow Rn$	0010nnnnmmmm1011	1	
SHAD Rm, Rn	$Rm < 0$ のとき、 $Rn \ll Rm \rightarrow Rn$ $Rm < 0$ のとき、 $Rn \gg Rm \rightarrow [MSB \rightarrow Rn]$	0100nnnnmmmm1100	1	
SHLD Rm, Rn	$Rm < 0$ のとき、 $Rn \ll Rm \rightarrow Rn$ $Rm < 0$ のとき、 $Rn \gg Rm \rightarrow [0 \rightarrow Rn]$	0100nnnnmmmm1101	1	
SUB Rm,Rn	$Rn-Rm \rightarrow Rn$	0011nnnnmmmm1000	1	
SUBC Rm,Rn	$Rn-Rm-T \rightarrow Rn$ , ボロ→T	0011nnnnmmmm1010	1	ボロ
SUBV Rm,Rn	$Rn-Rm \rightarrow Rn$ , アンダフロー→T	0011nnnnmmmm1011	1	アンダ フロー
SWAP.B Rm,Rn	$Rm \rightarrow$ 下位 2 バイトの上下バイト 交換→ $Rn$	0110nnnnmmmm1000	1	
SWAP.W Rm,Rn	$Rm \rightarrow$ 上下ワード交換→ $Rn$	0110nnnnmmmm1001	1	
TST Rm,Rn	$Rn \& Rm$ , 結果が 0 のとき 1→T	0010nnnnmmmm1000	1	テスト 結果

命令	動作	命令コード	実行 ステート	Tビット
XOR Rm,Rn	$Rn \wedge Rm \rightarrow Rn$	0010nnnnmmmm1010	1	
XTRCT Rm,Rn	Rm:Rn の中央 32 ビット→Rn	0010nnnnmmmm1101	1	

【注】 \* 通常最小実行ステートを示します。前後の命令との競合関係により実行ステート数は変わります。

表 A.38 レジスタ間接

命令	動作	命令コード	実行 ステート	Tビット
MOV.B Rm,@Rn	$Rm \rightarrow (Rn)$	0010nnnnmmmm0000	1	
MOV.W Rm,@Rn	$Rm \rightarrow (Rn)$	0010nnnnmmmm0001	1	
MOV.L Rm,@Rn	$Rm \rightarrow (Rn)$	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	

表 A.39 ポストインクリメントレジスタ間接 (積和演算)

命令	動作	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	2(~5)*	
MAC.W @Rm+,@Rn+	符号付きで (Rn) × (Rm)+MAC→MAC, Rn+2→Rn, Rm+2→Rm	0100nnnnmmmm1111	2(~5)*	

【注】 \* 通常最小実行ステートを示します。前後の命令との競合関係により実行ステート数は変わります。

表 A.40 ポストインクリメントレジスタ間接

命令	動作	命令コード	実行 ステート	Tビット
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	

表 A.41 プリデクリメントレジスタ間接

命令	動作	命令コード	実行 ステート	Tビット
MOV.B Rm,@-Rn	$Rn-1 \rightarrow Rn, Rm \rightarrow (Rn)$	0010nnnnmmmm0100	1	
MOV.W Rm,@-Rn	$Rn-2 \rightarrow Rn, Rm \rightarrow (Rn)$	0010nnnnmmmm0101	1	
MOV.L Rm,@-Rn	$Rn-4 \rightarrow Rn, Rm \rightarrow (Rn)$	0010nnnnmmmm0110	1	

表 A.42 インデックス付きレジスタ間接

命令	動作	命令コード	実行 ステート	Tビット
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	

表 A.43 浮動小数点 (SH-3E のみ)

命令	動作	命令コード	実行 ステート	Tビット
FADD FRm,FRn	FRn+FRm→FRn	1111nnnnmmmm0000	1	
FCMP/EQ FRm,FRn	(FRn=FRm)? 1:0→T	1111nnnnmmmm0100	1	比較結果
FCMP/GT FRm,FRn	(FRn>FRm)? 1:0→T	1111nnnnmmmm0101	1	比較結果
FDIV FRm,FRn	FRn/FRm→FRn	1111nnnnmmmm0011	13	
FMAC FR0,FRm,FRn	FR0×FRm+FRn→FRn	1111nnnnmmmm1110	1	
FMOV FRm,FRn	FRm→FRn	1111nnnnmmmm1100	1	
FMOV.S @(R0,Rm),FRn	(R0+Rm)→FRn	1111nnnnmmmm0110	1	
FMOV.S @Rm+,FRn	(Rm)→FRn,Rm+4→Rm	1111nnnnmmmm1001	1	
FMOV.S @Rm,FRn	(Rm)→FRn	1111nnnnmmmm1000	1	
FMOV.S FRm,@(R0,Rn)	(FRm)→(R0+Rn)	1111nnnnmmmm0111	1	
FMOV.S FRm,@-Rn	Rn-4→Rn, FRm→(Rn)	1111nnnnmmmm1011	1	
FMOV.S FRm,@Rn	FRm→(Rn)	1111nnnnmmmm1010	1	
FMUL FRm,FRn	FRn×FRm→FRn	1111nnnnmmmm0010	1	
FSUB FRm,FRn	FRn-FRm→FRn	1111nnnnmmmm0001	1	

## (5) md 形式

表 A.44 md 形式

命令	動作	命令コード	実行 ステート	Tビット
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	

## (6) nd4 形式

表 A.45 nd4 形式

命令	動作	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	

## (7) nmd 形式

表 A.46 nmd 形式

命令	動作	命令コード	実行 ステート	Tビット
MOV.L Rm,@(disp,Rn)	Rm→(disp×4+Rn)	0001nnnnmmmmddddd	1	
MOV.L @(disp,Rm),Rn	(disp×4+Rm)→Rn	0101nnnnmmmmddddd	1	

## (8) d 形式

表 A.47 ディスプレースメント付き GBR 間接

命令	動作	命令コード	実行 ステート	Tビット
MOV.B R0,@(disp,GBR)	R0→(disp+GBR)	11000000ddddddddd	1	
MOV.W R0,@(disp,GBR)	R0→(disp×2+GBR)	11000001ddddddddd	1	
MOV.L R0,@(disp,GBR)	R0→(disp×4+GBR)	11000010ddddddddd	1	
MOV.B @(disp,GBR),R0	(disp+GBR)→符号拡張→R0	11000100ddddddddd	1	
MOV.W @(disp,GBR),R0	(disp×2+GBR)→符号拡張→R0	11000101ddddddddd	1	
MOV.L @(disp,GBR),R0	(disp×4+GBR)→R0	11000110ddddddddd	1	

表 A.48 ディスプレースメント付き PC 相対

命令	動作	命令コード	実行 ステート	Tビット
MOVA @(disp,PC),R0	disp×4+PC→R0	11000111ddddddddd	1	
LDRS @(disp,pc)*	disp×2+PC→RS	10001100ddddddddd	3	
LDRE @(disp,pc)*	disp×2+PC→RE	10001110ddddddddd	3	

【注】 \* SH3-DSP の命令

表 A.49 PC 相対

命令	動作	命令コード	実行 ステート	Tビット
BF label	T=0 のとき disp×2+PC→PC, T=1 のとき nop	10001011ddddddddd	3/1*	
BF/S label	遅延分岐、T=0 のとき disp×2+PC→PC, T=1 のとき nop	10001111ddddddddd	2/1*	
BT label	T=1 のとき disp×2+PC→PC, T=0 のとき nop	10001001ddddddddd	3/1*	
BT/S label	遅延分岐、T=1 のとき disp×2+PC→PC, T=0 のとき nop	10001101ddddddddd	2/1*	

【注】 \* 分岐しないときは 1 ステートになります。



## (9) d12 形式

表 A.50 d12 形式

命令	動作	命令コード	実行 ステート	T ビット
BRA label	遅延分岐、 $\text{disp} \times 2 + \text{PC} \rightarrow \text{PC}$	1010ddddddddddd	2	
BSR label	遅延分岐、 $\text{PC} \rightarrow \text{PR}$ , $\text{disp} \times 2 + \text{PC} \rightarrow \text{PC}$	1011ddddddddddd	2	

## (10) nd8 形式

表 A.51 nd8 形式

命令	動作	命令コード	実行 ステート	T ビット
MOV.W @(disp,PC),Rn	$(\text{disp} \times 2 + \text{PC}) \rightarrow \text{符号拡張} \rightarrow \text{Rn}$	1001nnnnddddddd	1	
MOV.L @(disp,PC),Rn	$(\text{disp} \times 4 + \text{PC}) \rightarrow \text{Rn}$	1101nnnnddddddd	1	

## (11) i 形式

表 A.52 インデックス付き GBR 間接

命令	動作	命令コード	実行 ステート	T ビット
AND.B #imm,@(R0,GBR)	$(\text{R0} + \text{GBR}) \& \text{imm} \rightarrow (\text{R0} + \text{GBR})$	11001101iiiiiii	3	
OR.B #imm,@(R0,GBR)	$(\text{R0} + \text{GBR})   \text{imm} \rightarrow (\text{R0} + \text{GBR})$	11001111iiiiiii	3	
TST.B #imm,@(R0,GBR)	$(\text{R0} + \text{GBR}) \& \text{imm}$ , 結果が 0 のとき 1→T	11001100iiiiiii	3	テスト 結果
XOR.B #imm,@(R0,GBR)	$(\text{R0} + \text{GBR}) \wedge \text{imm} \rightarrow (\text{R0} + \text{GBR})$	11001110iiiiiii	3	

表 A.53 イミディエイト (レジスタ直接との算術論理演算)

命令	動作	命令コード	実行 ステート	T ビット
AND #imm,R0	$\text{R0} \& \text{imm} \rightarrow \text{R0}$	11001001iiiiiii	1	
CMP/EQ #imm,R0	$\text{R0} = \text{imm}$ のとき 1→T	10001000iiiiiii	1	比較結果
OR #imm,R0	$\text{R0}   \text{imm} \rightarrow \text{R0}$	11001011iiiiiii	1	
TST #imm,R0	$\text{R0} \& \text{imm}$ , 結果が 0 のとき 1→T	11001000iiiiiii	1	テスト 結果
XOR #imm,R0	$\text{R0} \wedge \text{imm} \rightarrow \text{R0}$	11001010iiiiiii	1	

表 A.54 イミディエイト (例外処理ベクタの指定)

命令	動作	命令コード	実行 ステート	T ビット
TRAPA #imm	$\text{imm} \rightarrow \text{TRA}$ , $\text{PC} \rightarrow \text{SPC}$ , $\text{SR} \rightarrow \text{SSR}$ , 1→SR.MD/BL/RB, $0 \times 160 \rightarrow \text{EXPEVT}$ , $\text{VBR} + \text{H}'00000100 \rightarrow \text{PC}$	11000011iiiiiii	6/8*	

【注】 \* SH3-DSP では 8 ステートになります。

表 A.55 コントロールレジスタへのロード (SH3-DSP のみ)

命令	動作	命令コード	実行 ステート	Tビット
SETRC #imm	imm→RC(SR[23:16]),zeros→SR[27:24]	10000010iiiiiiii	3	

(12) ni 形式

表 A.56 ni 形式

命令	動作	命令コード	実行 ステート	Tビット
ADD #imm,Rn	Rn+imm→Rn	0111nnnniiiiiiii	1	
MOV #imm,Rn	imm→符号拡張→Rn	1110nnnniiiiiiii	1	

## A.3 オペレーションコードマップ

表 A.57 オペレーションコードマップ

命令コード				Fx: 0000	Fx: 0001	Fx: 0010	Fx: 0011 ~ 1111
MSB		LSB		MD: 00	MD: 01	MD: 10	MD: 11
0000	Rn	Fx	0000				
0000	Rn	Fx	0001				
0000	Rn	00MD	0010	STC SR,Rn	STC GBR,Rn	STC VBR,Rn	STC SSR,Rn
0000	Rn	01MD	0010	STC SPC,Rn	STC MOD,Rn* <sup>2</sup>	STC RS,Rn* <sup>2</sup>	STC RE,Rn* <sup>2</sup>
0000	Rn	10MD	0010	STC R0_BANK,Rn	STC R1_BANK,Rn	STC R2_BANK,Rn	STC R3_BANK,Rn
0000	Rn	11MD	0010	STC R4_BANK,Rn	STC R5_BANK,Rn	STC R6_BANK,Rn	STC R7_BANK,Rn
0000	Rm	00MD	0011	BSRF Rm		BRAF Rm	
0000	Rm	10MD	0011	PREF @Rm			
0000	Rn	Rm	01MD	MOV.B Rm,@(R0,Rn)	MOV.W Rm,@(R0,Rn)	MOV.L Rm,@(R0,Rn)	MUL.L Rm,Rn
0000	0000	00MD	1000	CLRT	SETT	CLRMAC	LDTLB
0000	0000	01MD	1000	CLRS	SETS		
0000	0000	Fx	1001	NOP	DIV0U		
0000	0000	Fx	1010				
0000	0000	Fx	1011	RTS	SLEEP	RTE	
0000	Rn	Fx	1000				
0000	Rn	Fx	1001			MOVT Rn	
0000	Rn	00MD	1010	STS MACH,Rn	STS MACL,Rn	STS PR,Rn	
0000	Rn	01MD	1010		STS FPUL,Rn* <sup>1</sup>	STS FPSCR,Rn* <sup>1</sup> STS DSR,Rn* <sup>2</sup>	STS A0,Rn* <sup>2</sup>
0000	Rn	10MD	1010	STS X0,Rn* <sup>2</sup>	STS X1,Rn* <sup>2</sup>	STS Y0,Rn* <sup>2</sup>	STS Y1,Rn* <sup>2</sup>
0000	Rn	Fx	1011				
0000	Rn	RM	11MD	MOV.B @(R0,Rm), Rn	MOV.W @(R0,Rm), Rn	MOV.L @(R0,Rm), Rn	MAC.L @Rm+, @Rn+
0001	Rn	Rm	disp	MOV.L Rm,@(disp:4,Rn)			
0010	Rn	Rm	00MD	MOV.B Rm,@Rn	MOV.W Rm,@Rn	MOV.L Rm,@Rn	
0010	Rn	Rm	01MD	MOV.B Rm,@-Rn	MOV.W Rm,@-Rn	MOV.L Rm,@-Rn	DIV0S Rm,Rn
0010	Rn	Rm	10MD	TST Rm,Rn	AND Rm,Rn	XOR Rm,Rn	OR Rm,Rn
0010	Rn	Rm	11MD	CMP/STR Rm,Rn	XTRCT Rm,Rn	MULU.W Rm,Rn	MULS.W Rm,Rn
0011	Rn	Rm	00MD	CMP/EQ Rm,Rn		CMP/HS Rm,Rn	CMP/GE Rm,Rn
0011	Rn	Rm	01MD	DIV1 Rm,Rn	DMULU.L Rm,Rn	CMP/HI Rm,Rn	CMP/GT Rm,Rn
0011	Rn	Rm	10MD	SUB Rm,Rn		SUBC Rm,Rn	SUBV Rm,Rn
0011	Rn	Rm	11MD	ADD Rm,Rn	DMULU.L Rm,Rn	ADDC Rm,Rn	ADDV Rm,Rn
0100	Rn	Fx	0000	SHLL Rn	DT Rn	SHAL Rn	
0100	Rn	Fx	0001	SHLR Rn	CMP/PZ Rn	SHAR Rn	
0100	Rn	00MD	0010	STS.L MACH,@-Rn	STS.L MACL,@-Rn	STS.L PR,@-Rn	
0100	Rn	01MD	0010		STS.L FPUL,@-Rn* <sup>1</sup>	STS.L DSR,@-Rn* <sup>2</sup> STS.L FPSCR,@-Rn* <sup>1</sup>	STS.L A0,@-Rn* <sup>2</sup>
0100	Rn	10MD	0010	STS.L X0,@-Rn* <sup>2</sup>	STS.L X1,@-Rn* <sup>2</sup>	STS.L Y0,@-Rn* <sup>2</sup>	STS.L Y1,@-Rn* <sup>2</sup>
0100	Rn	00MD	0011	STC.L SR,@-Rn	STC.L GBR,@-Rn	STC.L VBR,@-Rn	STC.L SSR,@-Rn
0100	Rn	01MD	0011	STC.L SPC,@-Rn	STC.L MOD,@-Rn* <sup>2</sup>	STC.L RS,@-Rn* <sup>2</sup>	STC.L RE,@-Rn* <sup>2</sup>

付 録

命令コード				Fx: 0000	Fx: 0001	Fx: 0010	Fx: 0011 ~ 1111
MSB		LSB		MD: 00	MD: 01	MD: 10	MD: 11
0100	Rn	10MD	0011	STC.L R0_BANK,@-Rn	STC.L R1_BANK,@-Rn	STC.L R2_BANK,@-Rn	STC.L R3_BANK,@-Rn
0100	Rn	11MD	0011	STC.L R4_BANK,@-Rn	STC.L R5_BANK,@-Rn	STC.L R6_BANK,@-Rn	STC.L R7_BANK,@-Rn
0100	Rm/ Rn	Fx	0100	ROTL Rn	SETRC Rm	ROTCL Rn	
0100	Rn	Fx	0101	ROTR Rn	CMP/PL Rn	ROTCR Rn	
0100	Rm	00MD	0110	LDS.L @Rm+,MACH	LDS.L @Rm+,MACL	LDS.L @Rm+,PR	
0100	Rm	01MD	0110		LDS.L @Rm+,FPUL* <sup>1</sup>	LDS.L @Rm+,DSR* <sup>2</sup>	LDS.L @Rm+,A0* <sup>2</sup>
0100	Rm	10MD	0110	LDS.L @Rm+,X0* <sup>2</sup>	LDS.L @Rm+,X1* <sup>2</sup>	LDS.L @Rm+,Y0* <sup>2</sup>	LDS.L @Rm+,Y1* <sup>2</sup>
0100	Rm	00MD	0111	LDC.L @Rm+,SR	LDC.L @Rm+,GBR	LDC.L @Rm+,VBR	LDC.L @Rm+,SSR
0100	Rm	01MD	0111	LDC.L @Rm+,SPC	LDC.L @Rm+,MOD* <sup>2</sup>	LDC.L @Rm+,RS* <sup>2</sup>	LDC.L @Rm+,RE* <sup>2</sup>
0100	Rm	10MD	0111	LDC.L @Rm+,R0_BANK	LDC.L @Rm+,R1_BANK	LDC.L @Rm+,R2_BANK	LDC.L @Rm+,R3_BANK
0100	Rm	11MD	0111	LDC.L @Rm+,R4_BANK	LDC.L @Rm+,R5_BANK	LDC.L @Rm+,R6_BANK	LDC.L @Rm+,R7_BANK
0100	Rn	Fx	1000	SHLL2 Rn	SHLL8 Rn	SHLL16 Rn	
0100	Rn	Fx	1001	SHLR2 Rn	SHLR8 Rn	SHLR16 Rn	
0100	Rm	00MD	1010	LDS Rm,MACH	LDS Rm,MACL	LDS Rm,PR	
0100	Rm	01MD	1010		LDS Rm,FPUL* <sup>1</sup>	LDS Rm,DSR* <sup>2</sup>	LDS Rm,A0* <sup>2</sup>
0100	Rm	10MD	1010	LDS Rm,X0* <sup>2</sup>	LDS Rm,X1* <sup>2</sup>	LDS Rm,Y0* <sup>2</sup>	LDS Rm,Y1* <sup>2</sup>
0100	Rm	Fx	1011	JSR @Rm	TAS.B @Rn	JMP @Rm	
0100	Rn	Rm	1100	SHAD Rm,Rn			
0100	Rn	Rm	1101	SHLD Rm,Rn			
0100	Rm	00MD	1110	LDC Rm,SR	LDC Rm,GBR	LDC Rm,VBR	LDC Rm,SSR
0100	Rm	01MD	1110	LDC Rm,SPC	LDC Rm,MOD* <sup>2</sup>	LDC Rm,RS* <sup>2</sup>	LDC Rm,RE* <sup>2</sup>
0100	Rm	10MD	1110	LDC Rm,R0_BANK	LDC Rm,R1_BANK	LDC Rm,R2_BANK	LDC Rm,R3_BANK
0100	Rm	11MD	1110	LDC Rm,R4_BANK	LDC Rm,R5_BANK	LDC Rm,R6_BANK	LDC Rm,R7_BANK
0100	Rn	Rm	1111	MAC.W @Rm+,@Rn+			
0101	Rn	Rm	disp	MOV.L @(disp:4,Rm),Rn			
0110	Rn	Rm	00MD	MOV.B @Rm,Rn	MOV.W @Rm,Rn	MOV.L @Rm,Rn	MOV Rm,Rn
0110	Rn	Rm	01MD	MOV.B @Rm+,Rn	MOV.W @Rm+,Rn	MOV.L @Rm+,Rn	NOT Rm,Rn
0110	Rn	Rm	10MD	SWAP.B @Rm,Rn	SWAP.W @Rm,Rn	NEGC Rm,Rn	NEG Rm,Rn
0110	Rn	Rm	11MD	EXTU.B Rm,Rn	EXTU.W Rm,Rn	EXTS.B Rm,Rn	EXTS.W Rm,Rn
0111	Rn	imm		ADD #imm:8,Rn			
1000	00MD	Rn	disp	MOV.B R0,@(disp:4,Rn)	MOV.W R0,@(disp:4,Rn)	SETRC #imm* <sup>2</sup>	
1000	01MD	Rm	disp	MOV.B @(disp:4,Rm),R0	MOV.W @(disp:4,Rm),R0		
1000	10MD	imm/disp		CMP/EQ #imm:8,R0	BT disp:8		BF label:8
1000	11MD	imm/disp		LDRS @(disp:PC)* <sup>2</sup>	BT/S disp:8	LDRE @(disp:PC)* <sup>2</sup>	BF/S label:8
1001	Rn	disp		MOV.W @(disp:8,PC),Rn			
1010		disp		BRA label:12			
1011		disp		BSR label:12			

命令コード			Fx: 0000	Fx: 0001	Fx: 0010	Fx: 0011 ~ 1111	
MSB	LSB		MD: 00	MD: 01	MD: 10	MD: 11	
1100	00MD	imm/disp	MOV.B R0, @(disp:8,GBR)	MOV.W R0, @(disp:8,GBR)	MOV.L R0, @(disp:8,GBR)	TRAPA #imm:8	
1100	01MD	disp	MOV.B @(disp:8,GBR),R0	MOV.W @(disp:8,GBR),R0	MOV.L @(disp:8,GBR),R0	MOVA @(disp:8,PC),R0	
1100	10MD	imm	TST #imm:8,R0	AND #imm:8,R0	XOR #imm:8,R0	OR #imm:8,R0	
1100	11MD	imm	TST.B #imm:8,@(R0,GBR)	AND.B #imm:8,@(R0,GBR)	XOR.B #imm:8,@(R0,GBR)	OR.B #imm:8,@(R0,GBR)	
1101	Rn	disp	MOV.L @(disp:8,PC),Rn				
1110	Rn	imm	MOV #imm:8,Rn				
1111	Rn	Rm	00MD	FADD FRm,FRn* <sup>1</sup>	FSUB FRm,FRn* <sup>1</sup>	FMUL FRm,FRn* <sup>1</sup>	FDIV FRm,FRn* <sup>1</sup>
1111	Rn	Rm	01MD	FCMP/EQ FRm,FRn* <sup>1</sup>	FCMP/GT FRm,FRn* <sup>1</sup>	FMOV.S @(R0,Rm),FRm* <sup>1</sup>	FMOV.S FRm,@(R0,Rn)* <sup>1</sup>
1111	Rn	Rm	10MD	FMOV.S @Rm,FRn* <sup>1</sup>	FMOV.S @Rm+,FRn* <sup>1</sup>	FMOV.S FRm,@Rn* <sup>1</sup>	FMOV.S FRm,@-Rn* <sup>1</sup>
1111	Rn	Rm	1100	FMOV FRm,FRn* <sup>1</sup>			
1111	Rn	00MD	1101	FSTS FPUL,FRn* <sup>1</sup>	FLDS FRn,FPUL* <sup>1</sup>	FLOAT FPUL,FRn* <sup>1</sup>	FTRC FRn,FPUL* <sup>1</sup>
1111	Rn	01MD	1101	FNEG FRn* <sup>1</sup>	FABS FRn* <sup>1</sup>	FSQRT FRn* <sup>1</sup>	
1111	Rn	10MD	1101	FLDI0 FRn* <sup>1</sup>	FLDI1 FRn* <sup>1</sup>		
1111	Rn	Rm	1110	FMAC FR0,FRm,FRn* <sup>1</sup>			
命令コード			Fx: 0000	Fx: 0001	Fx: 0010	Fx: 0011 ~ 1111	
MSB	LSB		MD: 00	MD: 01	MD: 10	MD: 11	
1111	00**	****	(MOVX.W, MOVY.W, DSP デュアルデータ転送命令) (SH3-DSP)				
1111	01**	****	(MOV.S.W, MOV.S.L, DSP シングルデータ転送命令) (SH3-DSP)				
1111	10**	****	(DSP 並行処理命令、A フィールド: MOVX.W, MOVY.W, DSP デュアルデータ転送命令、B フィールド: PSHL ~ PLDS、DSP 演算命令) (SH3-DSP)				
1111	11**	****					

【注】 \*1 浮動小数点数演算命令または FPU に関連する CPU 命令です。これらの命令は SH-3E でのみ使用可能です。

\*2 DSP 機能をサポートするための CPU 命令です。これらの命令は SH3-DSP でのみ使用可能です。

表 A.58 DSP 演算命令オペレーションコードマップ (B フィールド)

命令コード				cc: 00	cc: 01 ( Un-condi-tional )	cc: 10 ( DCT )	cc: 11 ( DCF )
MSB			LSB				
0000	0	lmm	zzzz	PSHL #lmm, Dz			
0000	1***	****	****				
0001	0	lmm	zzzz	PSHA #lmm, Dz			
0001	1***	****	****				
001*	****	****	****				
0100	eeff	xxyy	gguu	PMULS Se, Sf, Dg			
0101	****	****	****				
0110	eeff	xxyy	gguu	PSUB Sx, Sy, Du	PMULS Se, Sf, Dg		
0111	eeff	xxyy	gguu	PADD Sx, Sy, Du	PMULS Se, Sf, Dg		
1000	00cc	xxyy	zzzz		[if cc] PSHL Sx, Sy, Dz		
1000	01cc	xxyy	zzzz	PCMP Sx, Sy			
1000	10cc	xxyy	zzzz	PABS Sx, Dz	[if cc] PDEC Sx, Dz		
1000	11cc	xxyy	zzzz		[if cc] PCLR Dz		
1001	00cc	xxyy	zzzz		[if cc] PSHA Sx, Sy, Dz		
1001	01cc	xxyy	zzzz		[if cc] PAND Sx, Sy, Dz		
1001	10cc	xxyy	zzzz	PRND Sx, Dz	[if cc] PINC Sx, Dz		
1001	11cc	xxyy	zzzz		[if cc] PDMSB Sx, Dz		
1010	00cc	xxyy	zzzz	PSUBC Sx, Sy, Dz	[if cc] PSUB Sx, Sy, Dz		
1010	01cc	xxyy	zzzz		[if cc] PXOR Sx, Sy, Dz		
1010	10cc	xxyy	zzzz	PABS Sy, Dz	[if cc] PDEC Sy, Dz		
1010	11cc	xxyy	zzzz				
1011	00cc	xxyy	zzzz	PADDC Sx, Sy, Dz	[if cc] PADD Sx, Sy, Dz		
1011	01cc	xxyy	zzzz		[if cc] POR Sx, Sy, Dz		
1011	10cc	xxyy	zzzz	PRND Sy, Dz	[if cc] PINC Sy, Dz		
1011	11cc	xxyy	zzzz		[if cc] PDMSB Sy, Dz		
1100	0***	****	****				
1100	10cc	xxyy	zzzz		[if cc] PNEG Sx, Dz		
1100	11cc	xxyy	zzzz		[if cc] PSTS MACH, Dz		
1101	0***	****	****				
1101	10cc	xxyy	zzzz		[if cc] PCOPY Sx, Dz		
1101	11cc	xxyy	zzzz		[if cc] PSTS MACL, Dz		
1110	0***	****	****				
1110	10cc	xxyy	zzzz		[if cc] PNEG Sy, Dz		
1110	11cc	xxyy	zzzz		[if cc] PLDS Dz, MACH		
1111	0***	****	****				
1111	10cc	xxyy	zzzz		[if cc] PCOPY Sy, Dz		
1111	11cc	xxyy	zzzz		[if cc] PLDS Dz, MACL		

## B. パイプライン動作と競合

SH-3、SH-3E、SH3-DSP の基本命令は 1 ステート (1 システムクロックサイクル) で実行されるようになっています。命令の実行に 2 ステート以上必要なときは、たとえば、分岐命令により分岐先アドレスが変わったとき、もしくは MA と IF の間の競合により実行ステート数が増えたときなどです。表 B.1 に競合のタイプと命令による実行ステート数とステージ数の一覧を示します。また、競合のない命令、さらに競合がなくとも 2 ステート以上の必要とする命令もあわせて表示します。

命令の競合は次のように起こります。

- (1) レジスタ間のオペレーションや転送が 1 ステート、競合なしで実行されます。
- (2) 競合は起こりませんが、命令はさらに 2 ステート以上要求します。
- (3) 実行サイクル数 (ステート数) が増えるにつれ、競合が起こります。競合の組み合わせとしては、次のようなものがあります。
  - MA が IF と競合。
  - MA が IF と競合。また、MA はメモリのロードとも同様に競合する場合があります。
  - MA が IF と競合。また、MA は乗算器とも同様に競合する場合があります。
  - MA が IF と競合。また、MA はメモリのロードとも同様に競合する場合があります、さらに、乗算器とも競合する場合があります。

表 B.1 競合の種類と命令との対応

競合	実行 ステート	ステージ 段数	命令
なし	1	3	レジスタ間転送命令 レジスタ間演算 (乗算系命令を除く) レジスタ間論理演算命令 シフト命令、動的シフト命令 システム制御 ALU 命令
	2	3	無条件分岐
	2/1	3	遅延付き条件分岐命令
	3/1	3	条件分岐
	4	3	SLEEP 命令
	4	5	RTE 命令
	5	5	LDC 命令 (SR)、レジスタ→SR
	6/8* <sup>2</sup>	6/8* <sup>2</sup>	TRAP 命令
• MA は IF と競合します	1	4	メモリストア命令 STS.L 命令 (PR) キャッシュ命令 STC.L 命令 (除くバンクレジスタ)
	1/2* <sup>2</sup>		
	2	5	STC.L 命令 (バンクレジスタ)
	3 3/4* <sup>2</sup>	6	メモリ論理演算 TAS 命令
	7	7	LDC.L 命令 (SR)、メモリ→SR
• MA は IF と競合します • メモリロードの競合を起こします	1	5	メモリロード命令 LDS.L 命令 (PR) LDC.L 命令
	1/5* <sup>2</sup>		
• MA は IF と競合します • 乗算器との競合を起こします	1	4	レジスタ→MAC 転送命令 メモリ→MAC 転送命令 MAC→メモリ転送命令
	1(~3)* <sup>1</sup>	6	乗算命令 (PMULS を除く)
	2(~5)* <sup>1</sup>	7	積和命令
	2(~5)* <sup>1</sup>	9	倍精度積和命令
	2(~5)* <sup>1</sup>	9	倍精度乗算命令
• MA は IF と競合します • 乗算器との競合を起こします • メモリロードの競合を起こします • DSP 演算との競合を起こします	1	5	MAC/DSP→レジスタ転送命令

【注】 \*1 通常実行ステートを示します。( )内の値は、前後の命令との競合関係による実行ステートです。

\*2 SH3-DSP では右側の数字のステート数、ステージ段数になります。



---

ルネサス32ビットRISCマイクロコンピュータ  
ソフトウェアマニュアル  
SH-3、SH-3E、SH3-DSP

発行年月日 1996年3月 第1版  
2006年5月26日 Rev.5.00  
発行 株式会社ルネサステクノロジ 営業統括部  
〒100-0004 東京都千代田区大手町 2-6-2  
編集 株式会社ルネサスソリューションズ  
グローバルストラテジックコミュニケーション本部  
カスタマサポート部

株式会社ルネサス テクノロジ 営業統括部 〒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

# RENESAS

<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	〒190-0023	立川市柴崎町2-2-23 (第二高島ビル2F)	(042) 524-8701
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	〒970-8026	いわき市平小太郎町4-9 (平小太郎ビル)	(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	支	社	〒460-0008	名古屋市中区栄4-2-29 (名古屋広小路プレイス)	(052) 249-3330
関	西	支	社	〒541-0044	大阪府中央区伏見町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
島	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

■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：コンタクトセンター E-Mail: [csc@renesas.com](mailto:csc@renesas.com)

# SH-3、SH-3E、SH3-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

RJJ09B0345-0500