お客様各位

カタログ等資料中の旧社名の扱いについて

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

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

2010年4月1日
ルネサスエレクトロニクス株式会社

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

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

ご注意書き

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

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

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

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

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

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

User's Manual

32

RENESAS

改訂一覧は表紙をクリックして直接ご覧になれます。
改訂一覧は改訂箇所をまとめたものであり、詳細については、
必ず本文の内容をご確認ください。

SH-2E

ソフトウェアマニュアル

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

ルネサス エレクトロニクス
www.renesas.com

Rev.2.00　2006.05

安全設計に関するお願い

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

本資料ご利用に際しての留意事項

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

製品に関する一般的注意事項

1. NC 端子の処理

   【注意】NC端子には、何も接続しないようにしてください。

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

2. 未使用入力端子の処理

   【注意】未使用の入力端子は、ハイまたはローレベルに固定してください。

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

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

3. 初期化前の処置

   【注意】電源投入時は，製品の状態は不定です。

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

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

4. 未定義・リザーブアドレスのアクセス禁止

   【注意】未定義・リザーブアドレスのアクセスを禁止します。

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

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

はじめに

SH-2E 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-2E 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-2E 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-2E 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-2E 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-2E 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。

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 6 章　命令セット

第 7 章　各命令の説明

1. 特長

SH-2E 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-2E CPU の特長を**表 1.1** に示します。

表 1.1　SH-2E CPU の特長

項目	特長
アーキテクチャ	• ルネサス テクノロジオリジナルアーキテクチャ • 内部 32 ビット構成
汎用レジスタマシン	• 汎用レジスタ　32 ビット×16 本 • コントロールレジスタ　32 ビット×3 本 • システムレジスタ　32 ビット×4 本 • 浮動小数点レジスタ　32 ビット×16 本 • 浮動小数点システムレジスタ　32 ビット×2 本
命令セット	• RISC タイプの命令セット – 命令長は 16 ビット固定長、これによるコード効率の向上 – ロードストアアーキテクチャ（基本演算はレジスタ間で実行） – 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 – C 言語指向の命令セット
命令実行時間	• 基本命令は 1 命令／1 ステート（1 命令／1 システムクロックサイクル）
アドレス空間	• アーキテクチャ上は 4GB
乗算器内蔵	• 乗算器内蔵により、16×16→32 の乗算を 1～3 ステートで実行 16×16＋64→64 の積和演算を 3/(2)*ステートで実行 32×32→64 の乗算を 2～4*ステートで実行 32×32＋64→64 の積和演算を 3/(2～4) *ステートで実行
パイプライン	• 5 段パイプライン方式
処理状態	• プログラム実行状態 • 例外処理状態 • バス権解放状態 • リセット状態 • 低消費電力状態
低消費電力状態	• スリープモード • スタンバイモード

項目	特長
FPU	• 単精度浮動小数点フォーマット • IEEE754 規格のデータタイプのサブセット • 無効演算例外およびゼロによる除算例外（IEEE754 規格準拠） • ゼロ方向への丸め処理（IEEE754 規格準拠） • 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 • 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） • 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 • FDIV サポート • FLDI0／FLDI1（Load constant 0/1）サポート

【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。

2. レジスタ構成

レジスタは、汎用レジスタ（32 ビット×16 本）、コントロールレジスタ（32 ビット×3 本）、システムレジスタ（32 ビット×4 本）、浮動小数点レジスタ（32 ビット×16 本）、浮動小数点システムレジスタ（32 ビット×2 本）の 5 種類があります。

2.1 汎用レジスタ

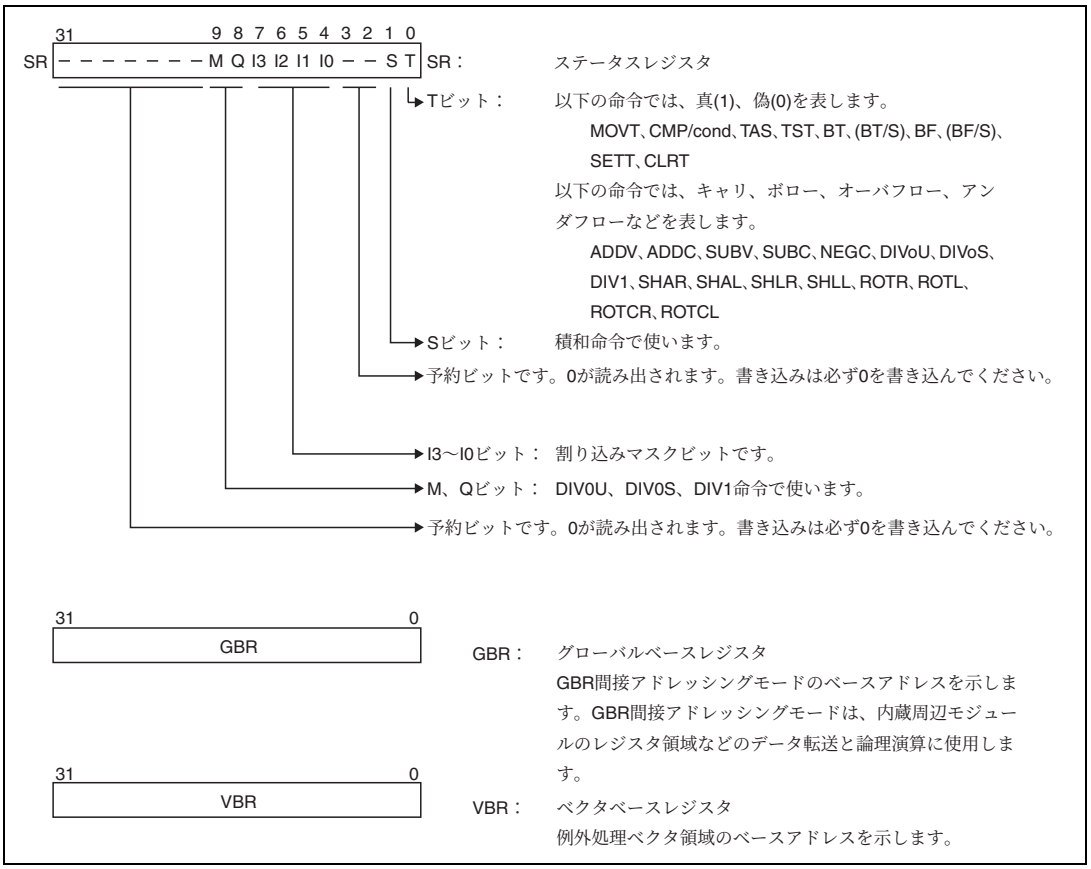
汎用レジスタ（Rn）は、32 ビットの長さで、R0 から R15 までの 16 本あります。汎用レジスタは、データ処理、アドレス計算に使われます。R0 は、インデックスレジスタとしても使用します。いくつかの命令では使用できるレジスタが R0 に固定されています。R15 は、ハードウェアスタックポインタ（SP）として使われます。例外処理でのステータスレジスタ（SR）とプログラムカウンタ（PC）の退避、回復は、R15 を用いてスタックを参照し行います。

【注】 *1 インデックス付きレジスタ間接、インデックス付きGBR間接アドレッシングモードのインデックスレジスタとしても使用します。
命令によっては、ソースまたはデスティネーションレジスタをR0に固定しているものがあります。

*2 R15は例外処理の中で、ハードウェアスタックポインタとして使用されます。

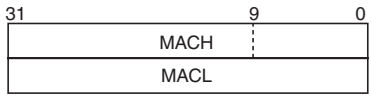
2.2 コントロールレジスタ

コントロールレジスタは 32 ビットの長さで、ステータスレジスタ（SR）、グローバルベースレジスタ（GBR）、ベクタベースレジスタ（VBR）の 3 本があります。SR は処理の状態を表します。GBR は GBR 間接アドレッシングモードのベースアドレスとして使用し、内蔵周辺モジュールのレジスタのデータ転送などに使用します。VBR は割り込みを含む例外処理ベクタ領域のベースアドレスとして使用します。



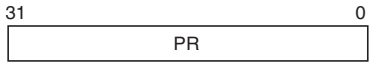
2.3 システムレジスタ

システムレジスタは 32 ビットの長さで、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）の 4 本があります。MACH、MACL は乗算または積和演算の結果を格納します。PR はサブルーチンプロシージャからの戻り先アドレスを格納します。PC は実行中のプログラムのアドレスを示し、処理の流れを制御します。



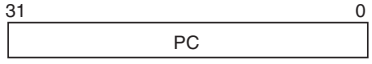
MACH*、MACL：積和レジスタ（MAC）

乗算、積和演算の結果の格納レジスタです。SH-2 CPUではMACHは32ビット有効です。SH-1 CPUではMACHは下位10ビットが有効であり、読み出すときは32ビット長まで符号拡張されます。



PR：プロシージャレジスタ

サブルーチンプロシージャからの戻り先アドレスの格納レジスタです。



PC：プログラムカウンタ

PCは現在実行中の命令の4バイト（2命令）先を示しています。

2.4 浮動小数点レジスタ

浮動小数点レジスタ（FRn）は 32 ビットの長さで、FR0～15 までの 16 本あります。浮動小数点レジスタは浮動小数点命令で使用します。FR0 は FMAC 命令のインデックスレジスタとして機能します。これらのレジスタは浮動小数点演算ユニット（FPU）に内蔵しています。

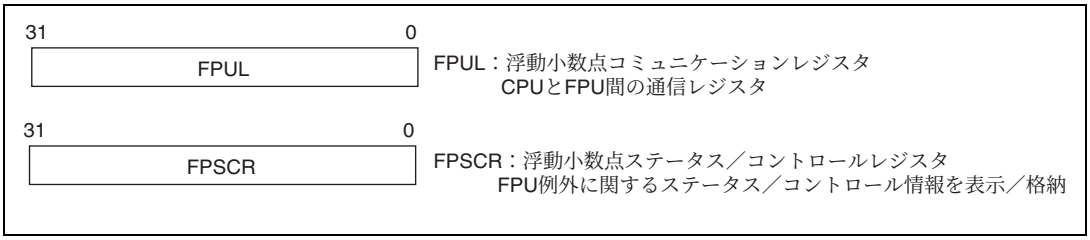
詳しくは、「**第 4 章　浮動小数点演算ユニット（FPU）**」を参照してください。

2.5 浮動小数点システムレジスタ

浮動小数点システムレジスタは 32 ビットの長さで、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）の 2 本があります。FPUL は CPU と浮動小数点ユニット（FPU）間の通信レジスタです。

FPSCR は FPU 例外に関するステータス／コントロール情報を表示／格納します。

これらのレジスタは浮動小数点演算ユニット（FPU）に内蔵しています。詳しくは、「**第 4 章　浮動小数点演算ユニット（FPU）**」を参照してください。



2.6 レジスタの初期値

リセット後のレジスタの値を表 2.1 に示します

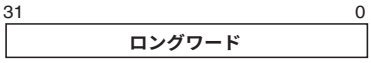
表 2.1 レジスタの初期値

区　分	レジスタ	初　期　値
汎用レジスタ	R0～R14	不定
	R15（SP）	ベクタアドレステーブル中の SP の値
コントロールレジスタ	SR	I3～I0 は 1111（H'F）、予約ビットは 0、その他は不定
	GBR	不定
	VBR	H'00000000
システムレジスタ	MACH、MACL、PR	不定
	PC	ベクタアドレステーブル中の PC の値
浮動小数点レジスタ	FR0～FR15	不定
浮動小数点システムレジスタ	FPUL	不定
	FPSCR	H'00040001

3. データ形式

3.1 レジスタのデータ形式

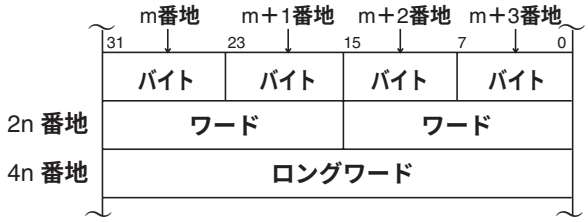
レジスタオペランドのデータサイズは常にロングワード（32 ビット）です。メモリ上のデータをレジスタへロードするとき、メモリオペランドのデータサイズがバイト（8 ビット）、もしくはワード（16 ビット）の場合は、ロングワードに符号拡張し、レジスタに格納します。



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。

バイトデータは任意番地に、ワードデータは 2n 番地から、ロングワードデータは 4n 番地から配置してください。その境界以外からアクセスすると、アドレスエラーが発生します。このとき、アクセスした結果は保証しません。特に、ハードウェアスタックポインタ（SP、R15）が指し示すスタックにはプログラムカウンタ（PC）とステータスレジスタ（SR）をロングワードで保持しますので、ハードウェアスタックポインタの値が 4n になるように設定してください。アドレスエラーについては、「**ハードウェアマニュアル**」を参照してください。



ビックエンディアン

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、「**第 5 章　命令の特長**」の「5.1（8）**イミディエイトデータ**」を参照してください。

4. 浮動小数点演算ユニット（FPU）

4.1 概要

SH-2E 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。

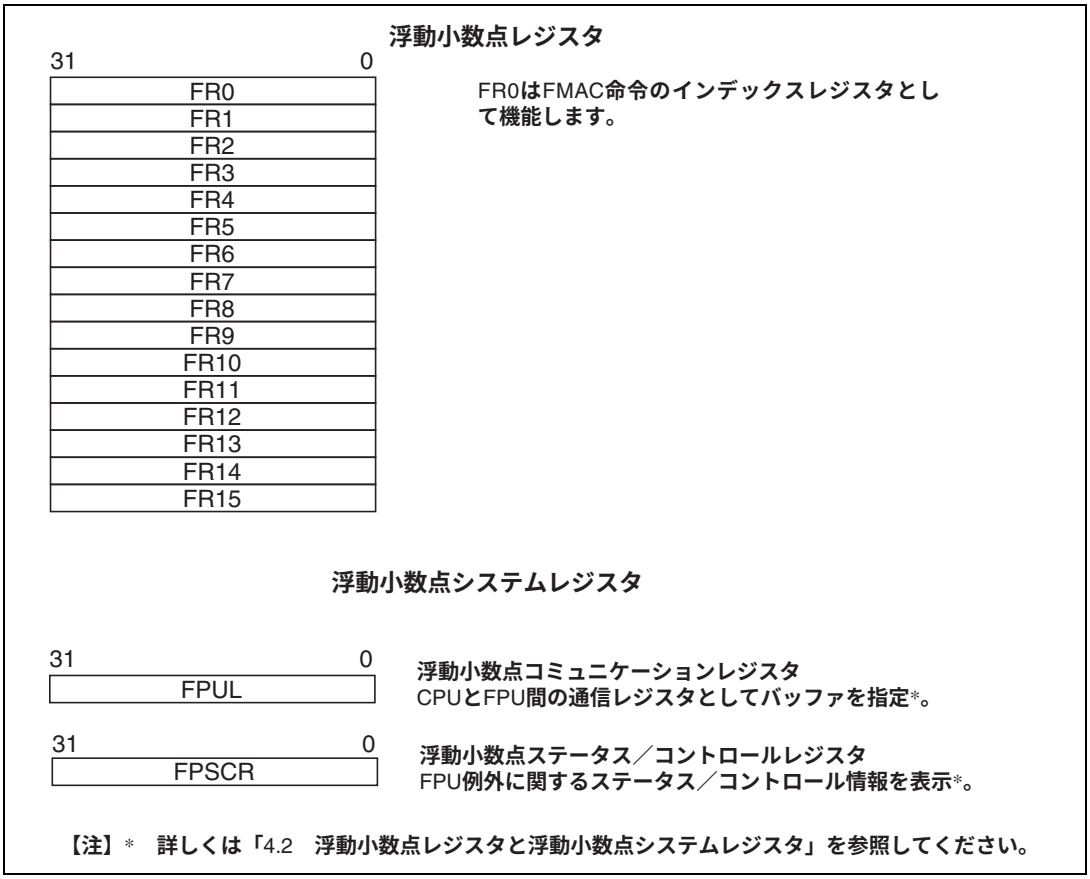


図 4.1 **レジスタの構成の概要（浮動小数点レジスタ、浮動小数点システムレジスタ）**

4.2 浮動小数点レジスタと浮動小数点システムレジスタ

4.2.1 浮動小数点レジスタファイル

SH-2E 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レジスタとして機能します。

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 本の通信レジスタ 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-2E ではこの通信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-2E 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、漸近的なアンダーフロー（非正規化数）、および FPU 例外に関する詳細情報の格納を制御します。FPU 自体を無効とするモジュールストップビットはモジュールスタンバイコントロールレジスタ（MSTCR）にあります。詳しくは「**ハードウェアマニュアル**」を参照してください。FPU はリセットスタート時は常にイネーブル状態です。

表 4.1 に起こりうる 5 種類の FPU 例外と対応するフラグを示します。さらに 6 番目のフラグとして FPU エラーがあり、これは浮動小数点ユニットが 5 種類以外のエラー状態を知らせるものです。

表 4.1　浮動小数点例外フラグ

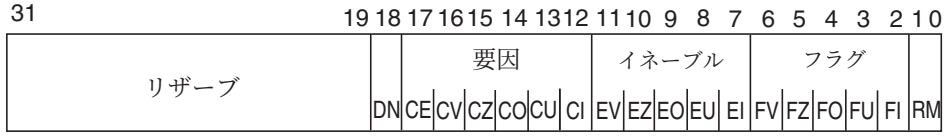
フラグ	意味	SH-2E でのサポート
E	FPU エラー	−
V	無効演算	○
Z	ゼロによる除算	○
O	オーバフロー（値は表現されない）	−
U	アンダフロー（値は表現されない）	−
I	不正確（結果は表現されない）	−

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ってのみ、リセットすることができます。



DN： 不正規化ビット

SH-2Eでは常に1で不正規化数のソースまたはデスティネーションオペランドは0になります。このビットはLDS命令でも書き込めません。

CV： 無効演算要因ビット

1のとき：現在の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のとき：無効演算例外が発生していないことを示します。

CZ： 0による除算要因ビット

1のとき：現在の命令の実行中に0による除算例外が発生したことを示します。

0のとき：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

1のとき：無効演算例外を発生許可

0のとき：無効演算例外は発生せず、結果としてqNaNを返します。

EZ： 0による除算イネーブル

1のとき：現在の命令の実行中に0による除算例外を発生許可

0のとき：0による除算例外は発生せず、結果として現在の式の符号（+もしくは-）を付けた無限値を返します。

FV： 無効演算例外フラグビット

1のとき：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のとき：無効演算例外は発生していないことを示します。

FZ： 0による除算例外フラグビット

1のとき：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のとき：0による除算例外が発生していないことを示します。

RM： 丸めモードビットSH-2Eでは、常に01でゼロ方向への丸めが（RZモード）行われていることを意味します。このビットはLDS命令でも書き込めません。

SH-2Eでは、要因フィールドEOUI（CE、CO、CU、およびCI）イネーブルフィールドのOUI（EO、EU、およびEI）、またフラグフィールド中のOUI（FO、FU、およびFI）の各ビットとリザーブ領域は"0"にプリセットされており、LDS命令を使用しても変更できません。

図 4.2 浮動小数点ステータス／コントロールレジスタ

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-2E は単精度浮動小数点演算をサポートしています。さらに IEEE754 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

指数はゲタばき表現（バイアス）されます。すなわち、
e=E+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）と非正規化数を表し、E_{max}+1 は、正負の無限大と非数（NaN）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。

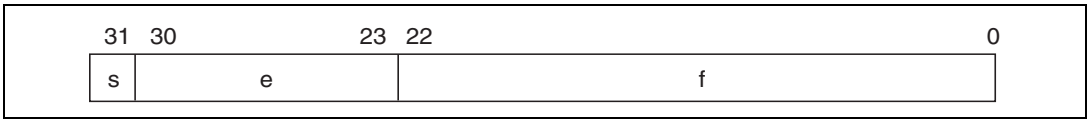


図 4.3 浮動小数点数のフォーマット

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!=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、$v=(-1)^s$(infinity)〔正または負の無限大〕
E_{min}<=E<=Emax ならば、$v=(-1)^s 2^E(1.f)$〔正規化数〕
E==E_{min}-1 かつ f!=0 ならば、$v=(-1)^s 2^{Emin}(0.f)$〔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 つのビットがセットされます。ビット 22 がセットされていれば、シグナリング NaN（sNaN）を示します。ビット 22 がリセットされていれば、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**図 4.4** に示します。図中のビット N はシグナリング NaN ではセットされ、クワイアット NaN ではリセットされます。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 つのビットはセットされています。

非数（NaN）では、符号ビットは、don't care となります。

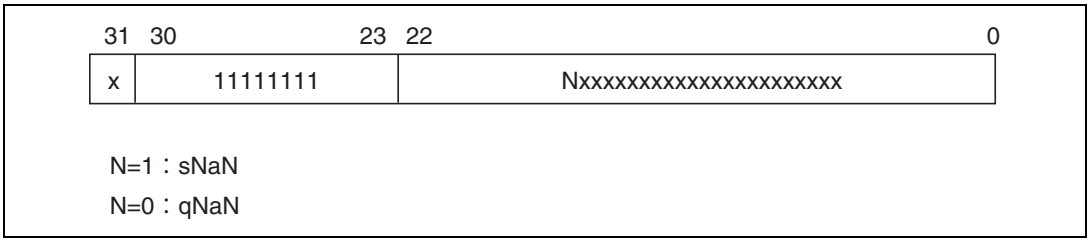


図 4.4 NaN ビットパターン

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 ビットがリセットされると、演算結果（出力）はクワイアット NaN（qNaN）となります。
- FPSCR レジスタの EV ビットがセットされると、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クワイアット NaN を入力し、かつシグナリング NaN 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、出力は常にクワイアット NaN となります。そしてこのとき例外は発生しません。

非数（NaN）が入力された場合の浮動小数点演算については、「**第 7 章　各命令の説明**」を参照してください。

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-2E 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

浮動小数点数の値の表現には、**表 4.2** 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 4.2 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」で説明したとおり**
+INF	0x7F800000
-INF	0xFF800000
qNaN（**クワイアット** NaN）	**「4.3.2 非数（NaN）」で説明したとおり**
sNaN（**シグナリング** NaN）	**「4.3.2 非数（NaN）」で説明したとおり**

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、イネーブル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、同一の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 ディスイネーブル状態の例外

イネーブルビット EV がセットされていなければ、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ければ、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。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

すべての FPU 例外は、同一の一般例外事象すなわち FPU 例外として、H'00000034 番地にベクタテーブルアドレスオフセットを持っています。

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-2E 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。詳しくは「**第 7 章　各命令の説明**」を参照してください。

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、Divide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Compare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やFPSCR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やFPSCRにアクセスする命令はFPULやFPSCR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。

5. 命令の特長

5.1 RISC 方式

命令は RISC 方式です。特長は次のとおりです。

（1）16ビット固定長命令

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。

（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。40MHz動作時、1ステートは25nsになります。

（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。イミディエイト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 5.1 ワードデータの符号拡張

SH-2E 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。

（4）ロードストアアーキテクチャ

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（ロードストアアーキテクチャ）。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。

（5）遅延分岐

無条件分岐命令などは、遅延分岐命令です。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、分岐時のパイプラインの乱れを軽減しています。

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。

表 5.2 遅延分岐命令

SH-2E 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

（6）乗算／積和演算

16×16→32の乗算を1～2ステート、16×16＋64→64の積和演算を2～3ステートで実行します。32×32→64の乗算や、32×32＋64→64の積和演算を2～4ステートで実行します。

（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。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 5.3 Tビット

SH-2E 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

（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命令コードの中に配置します。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。

表 5.4 イミディエイトデータによる参照

区分	SH-2E 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。

（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。命令実行時にイミディエイトデータをロードする方法で、この値をレジスタに転送し、レジスタ間接アドレッシングモードでデータを参照します。

表 5.5 絶対アドレスによる参照

区分	SH-2E 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

（10）16ビット／32ビットディスプレースメント

16ビットまたは32ビットディスプレース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ースメントの値をメモリ上のテーブルに配置しておきます。命令実行時にイミディエイトデータをロードする方法で、この値をレジスタに転送し、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データを参照します。

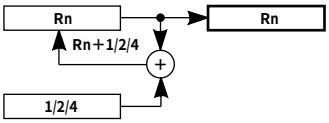
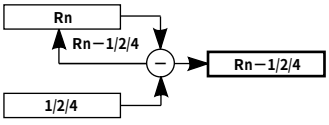
表 5.6　ディスプレースメントによる参照

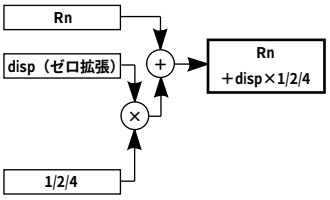
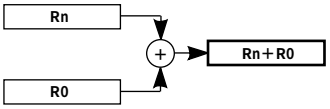
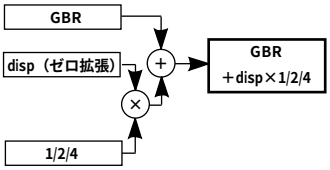
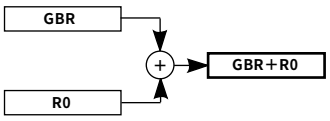
区分	SH-2E CPU	他の CPU の例
16 ビットディスプレースメント	MOV.W　@(disp,PC),R0 MOV.W　@(R0,R1),R2 ・・・・・・・・ .DATA.W　H'1234	MOV.W　@(H'1234,R1),R2

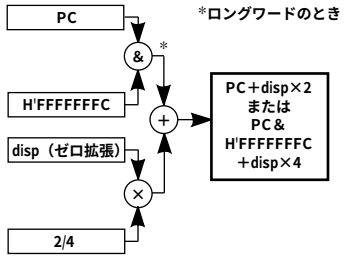
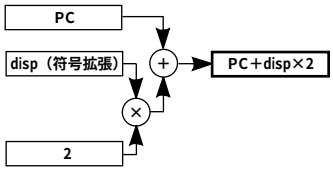
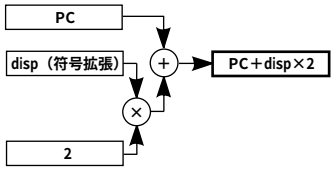
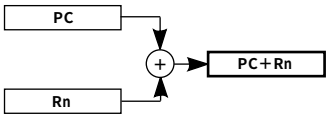
5.2　アドレッシングモード

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 5.7　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）

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

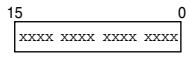
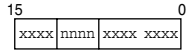
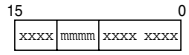
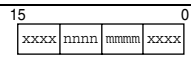
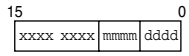
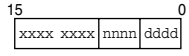
アドレッシングモード	命令フォーマット	実効アドレスの計算方法	計算式
ディスプレースメント付き PC 相対	@(disp:8,PC)	実効アドレスはレジスタ PC に 8 ビットディスプレースメント disp を加算した内容です。disp はゼロ拡張後、オペランドサイズによってワードで 2 倍、ロングワードで 4 倍します。 さらにロングワードのときは PC の下位 2 ビットをマスクします。 	ワード：PC＋disp×2 ロングワード： PC&H'FFFFFFFC＋disp×4
PC 相対	disp:8	実効アドレスはレジスタ PC に 8 ビットディスプレースメント disp を符号拡張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

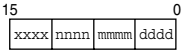
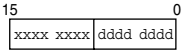
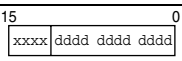
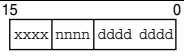
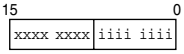
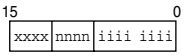
5.3 命令形式

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。

- xxxx ：命令コード
- mmmm ：ソースレジスタ
- nnnn ：デスティネーションレジスタ
- iiii ：イミディエイトデータ
- dddd ：ディスプレースメント

表 5.8 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		—	—	NOP	
n 形式		—	nnnn：レジスタ直接	MOV.T	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		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 レジスタ間接	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		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：* 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)
md 形式		mmmmdddd： ディスプレース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		R0（レジスタ直接）	nnnndddd： ディスプレースメント付きレジスタ間接	MOV.B	R0,@(disp,Rn)

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	
nmd 形式		mmmm：レジスタ直接	nnnndddd： ディスプレースメント付きレジスタ間接	MOV.L	Rm,@(disp,Rn)
		mmmmdddd： ディスプレース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		dddddddd： ディスプレースメント付き GBR 間接	R0（レジスタ直接）	MOV.L	@(disp,GBR),R0
		R0（レジスタ直接）	dddddddd： ディスプレースメント付き GBR 間接	MOV.L	R0,@(disp,GBR)
		dddddddd： ディスプレースメント付き PC 相対	R0（レジスタ直接）	MOVA	@(disp,PC),R0
		dddddddd：PC 相対	—	BF	label
d12 形式		dddddddddddd：PC 相対	—	BRA	label (label=disp+PC)
nd8 形式		dddddddd： ディスプレースメント付き PC 相対	nnnn：レジスタ直接	MOV.L	@(disp,PC),Rn
i 形式		iiiiiiii： イミディエイト	インデックス付き GBR 間接	AND.B	#imm,@(R0,GBR)
		iiiiiiii： イミディエイト	R0（レジスタ直接）	AND	#imm,R0
		iiiiiiii： イミディエイト	—	TRAPA	#imm
ni 形式		iiiiiiii： 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. 命令セット

6.1 分類順命令セット

命令を分類順に**表** 6.1 に示します。

表 6.1 **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の転送 周辺モジュールデータの転送 構造体データの転送	39
		MOVA	実行アドレスの転送	
		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 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	
		MULS	符号付き乗算	
		MULU	符号なし乗算	
		NEG	符号反転	
		NEGC	ボロー付き符号反転	
		SUB	2 進減算	
		SUBC	ボロー付き 2 減算	
		SUBV	アンダフロー付き 2 進減算	

分　類	命令の種類	オペコード	機　能	命令数
論理演算命令	6	AND	論理積演算	14
		NOT	ビット反転	
		OR	論理和演算	
		TAS	メモリテストとビットセット	
		TST	論理積演算のTビットセット	
		XOR	排他的論理和演算	
シフト命令	10	ROTL	1 ビット左回転	14
		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	算術的 1 ビット左シフト	
		SHAR	算術的 1 ビット右シフト	
		SHLL	論理的 1 ビット左シフト	
		SHLLn	論理的 n ビット左シフト	
		SHLR	論理的 1 ビット右シフト	
		SHLRn	論理的 n ビット右シフト	
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	11	CLRT	T ビットのクリア	31
		CLRMAC	MAC レジスタのクリア	
		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	
		NOP	無操作	
		RTE	例外処理からの復帰	
		SETT	T ビットのセット	
		SLEEP	低消費電力状態への遷移	
		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
浮動小数点演算命令	15	FABS	浮動小数点数絶対値	22
		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	
		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	
FPU に関する CPU 命令	2	LDS	浮動小数点システムレジスタへのロード	8
		STS	浮動小数点システムレジスタからのストア	
	計　79			172

命令の命令コード、動作、実行ステートを、以下の形式で分類順に説明します。

命令	命令コード	動作の概略	実行ステート	T ビット
ニーモニックで表示しています。 【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*2	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 0001 : R1 …… 1111 : R15 iiii : イミディエイトデータ dddd : ディスプレースメント	動作の概略を表示しています。 【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	ノーウェイトのときの値です。*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。
詳しくは「第 7 章　各命令の説明」を参照してください。

6.1.1 データ転送命令

表 6.2 データ転送命令

命令		命令コード	動作	実行ステート	T ビット
MOV	#imm, Rn	1110nnnniiiiiiii	imm→符号拡張→Rn	1	—
MOV.W	@(disp, PC), Rn	1001nnnndddddddd	(disp×2+PC) →符号拡張→Rn	1	—
MOV.L	@(disp, PC), Rn	1101nnnndddddddd	(disp×4+PC) →Rn	1	—
MOV	Rm, Rn	0110nnnnmmmm0011	Rm→Rn	1	—
MOV.B	Rm, @Rn	0010nnnnmmmm0000	Rm→ (Rn)	1	—
MOV.W	Rm, @Rn	0010nnnnmmmm0001	Rm→ (Rn)	1	—
MOV.L	Rm, @Rn	0010nnnnmmmm0010	Rm→ (Rn)	1	—
MOV.B	@Rm, Rn	0110nnnnmmmm0000	(Rm) →符号拡張→Rn	1	—
MOV.W	@Rm, Rn	0110nnnnmmmm0001	(Rm) →符号拡張→Rn	1	—
MOV.L	@Rm, Rn	0110nnnnmmmm0010	(Rm) →Rn	1	—
MOV.B	Rm, @- Rn	0010nnnnmmmm0100	Rn-1→Rn, Rm→ (Rn)	1	—
MOV.W	Rm, @- Rn	0010nnnnmmmm0101	Rn-2→Rn, Rm→ (Rn)	1	—
MOV.L	Rm, @- Rn	0010nnnnmmmm0110	Rn-4→Rn, Rm→ (Rn)	1	—
MOV.B	@Rm+, Rn	0110nnnnmmmm0100	(Rm) →符号拡張→Rn, Rm+1→Rm	1	—
MOV.W	@Rm+, Rn	0110nnnnmmmm0101	(Rm) →符号拡張→Rn, Rm+2→Rm	1	—
MOV.L	@Rm+, Rn	0110nnnnmmmm0110	(Rm) →Rn, Rm+4→Rm	1	—
MOV.B	R0, @(disp, Rn)	10000000nnnndddd	R0→ (disp+Rn)	1	—
MOV.W	R0, @(disp, Rn)	10000001nnnndddd	R0→ (disp×2+Rn)	1	—
MOV.L	Rm, @(disp, Rn)	0001nnnnmmmmdddd	Rm→ (disp×4+Rn)	1	—
MOV.B	@(disp, Rm), R0	10000100mmmmdddd	(disp+Rm) →符号拡張→R0	1	—
MOV.W	@(disp, Rm), R0	10000101mmmmdddd	(disp×2+Rm) →符号拡張→R0	1	—
MOV.L	@(disp, Rm), R0	0101nnnnmmmmdddd	(disp×4+Rm) →Rn	1	—
MOV.B	Rm, @(R0, Rn)	0000nnnnmmmm0100	Rm→ (R0+Rn)	1	—
MOV.W	Rm, @(R0, Rn)	0000nnnnmmmm0101	Rm→ (R0+Rn)	1	—
MOV.L	Rm, @(R0, Rn)	0000nnnnmmmm0110	Rm→ (R0+Rn)	1	—
MOV.B	@(R0, Rm), Rn	0000nnnnmmmm1100	(R0+Rm) →符号拡張→Rn	1	—
MOV.W	@(R0, Rm), Rn	0000nnnnmmmm1101	(R0+Rm) →符号拡張→Rn	1	—
MOV.L	@(R0, Rm), Rn	0000nnnnmmmm1110	(R0+Rm) →Rn	1	—
MOV.B	R0, @(disp, GBR)	11000000dddddddd	R0→ (disp+GBR)	1	—
MOV.W	R0, @(disp, GBR)	11000001dddddddd	R0→ (disp×2+GBR)	1	—
MOV.L	R0, @(disp, GBR)	11000010dddddddd	R0→ (disp×4+GBR)	1	—
MOV.B	@(disp, GBR), R0	11000100dddddddd	(disp+GBR) →符号拡張→R0	1	—
MOV.W	@(disp, GBR), R0	11000101dddddddd	(disp×2+GBR) →符号拡張→R0	1	—
MOV.L	@(disp, GBR), R0	11000110dddddddd	(disp×4+GBR) →R0	1	—
MOVA	@(disp, PC), R0	11000111dddddddd	disp×4+PC→R0	1	—
MOVT	Rn	0000nnnn00101001	T→Rn	1	—
SWAP.B	Rm, Rn	0110nnnnmmmm1000	Rm→下位 2 バイトの上下バイト交換→Rn	1	—
SWAP.W	Rm, Rn	0110nnnnmmmm1001	Rm→上下ワード交換→Rn	1	—
XTRCT	Rm, Rn	0010nnnnmmmm1101	Rm:Rn の中央 32 ビット→Rn	1	—

6.1.2 算術演算命令

表 6.3 算術演算命令

命　令		命令コード	動　作	実行ステート	T ビット
ADD	Rm, Rn	0011nnnnmmmm1100	Rn+Rm→Rn	1	—
ADD	#imm, Rn	0111nnnniiiiiiii	Rn+imm→Rn	1	—
ADDC	Rm, Rn	0011nnnnmmmm1110	Rn+Rm+T→Rn, キャリ→T	1	キャリ
ADDV	Rm, Rn	0011nnnnmmmm1111	Rn+Rm→Rn, オーバフロー→T	1	オーバフロー
CMP/EQ	#imm, R0	10001000iiiiiiii	R0=imm のとき 1→T	1	比較結果
CMP/EQ	Rm, Rn	0011nnnnmmmm0000	Rn=Rm のとき 1→T	1	比較結果
CMP/HS	Rm, Rn	0011nnnnmmmm0010	無符号で Rn≧Rm のとき 1→T	1	比較結果
CMP/GE	Rm, Rn	0011nnnnmmmm0011	有符号で Rn≧Rm のとき 1→T	1	比較結果
CMP/HI	Rm, Rn	0011nnnnmmmm0110	無符号で Rn>Rm のとき 1→T	1	比較結果
CMP/GT	Rm, Rn	0011nnnnmmmm0111	有符号で Rn>Rm のとき 1→T	1	比較結果
CMP/PL	Rn	0100nnnn00010101	Rn>0 のとき 1→T	1	比較結果
CMP/PZ	Rn	0100nnnn00010001	Rn≧0 のとき 1→T	1	比較結果
CMP/STR	Rm, Rn	0010nnnnmmmm1100	いずれかのバイトが等しいとき 1→T	1	比較結果
DIV1	Rm, Rn	0011nnnnmmmm0100	1 ステップ除算(Rn÷Rm)	1	計算結果
DIV0S	Rm, Rn	0010nnnnmmmm0111	Rn の MSB→Q, Rm の MSB→M, M ^ Q→T	1	計算結果
DIV0U		0000000000011001	0→M/Q/T	1	0
DMULS.L	Rm, Rn	0011nnnnmmmm1101	符号付きで Rn×Rm→MACH, MACL 32×32→64 ビット	2〜4*1	—
DMULU.L	Rm, Rn	0011nnnnmmmm0101	符号なしで Rn×Rm→MACH, MACL 32×32→64 ビット	2〜4*1	—
DT	Rn	0100nnnn00010000	Rn−1→Rn, Rn が 0 のとき 1→T Rn が 0 以外のとき 0→T	1	比較結果
EXTS.B	Rm, Rn	0110nnnnmmmm1110	Rm をバイトから符号拡張→Rn	1	—
EXTS.W	Rm, Rn	0110nnnnmmmm1111	Rm をワードから符号拡張→Rn	1	—
EXTU.B	Rm, Rn	0110nnnnmmmm1100	Rm をバイトからゼロ拡張→Rn	1	—
EXTU.W	Rm, Rn	0110nnnnmmmm1101	Rm をワードからゼロ拡張→Rn	1	—
MAC.L	@Rm+, @Rn+	0000nnnnmmmm1111	符号付きで(Rn)×(Rm)+MAC→MAC 32×32+64→64 ビット	3/(2〜4) *1	—
MAC.W	@Rm+, @Rn+	0100nnnnmmmm1111	符号付きで(Rn)×(Rm)+MAC→MAC 16×16+64→64 ビット	3/(2) *1	—
MUL.L	Rm, Rn	0000nnnnmmmm0111	Rn×Rm→MACL　32×32→32 ビット	2〜4*1	—
MULS.W	Rm, Rn	0010nnnnmmmm1111	符号付きで Rn×Rm→MACL 16×16→32 ビット	1〜3*1	—
MULU.W	Rm, Rn	0010nnnnmmmm1110	符号なしで Rn×Rm→MACL 16×16→32 ビット	1〜3*1	—
NEG	Rm, Rn	0110nnnnmmmm1011	0-Rm→Rn	1	—
NEGC	Rm, Rn	0110nnnnmmmm1010	0-Rm-T→Rn, ボロー→T	1	ボロー
SUB	Rm, Rn	0011nnnnmmmm1000	Rn-Rm→Rn	1	—
SUBC	Rm, Rn	0011nnnnmmmm1010	Rn-Rm-T→Rn, ボロー→T	1	ボロー
SUBV	Rm, Rn	0011nnnnmmmm1011	Rn-Rm→Rn, アンダフロー→T	1	オーバフロー

【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。

6.1.3 論理演算命令

表 6.4 論理演算命令

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

6.1.4 シフト命令

表 6.5 シフト命令

命令		命令コード	動作	実行ステート	T ビット
ROTL	Rn	0100nnnn00000100	T←Rn←MSB	1	MSB
ROTR	Rn	0100nnnn00000101	LSB→Rn→T	1	LSB
ROTCL	Rn	0100nnnn00100100	T←Rn←T	1	MSB
ROTCR	Rn	0100nnnn00100101	T→Rn→T	1	LSB
SHAL	Rn	0100nnnn00100000	T←Rn←0	1	MSB
SHAR	Rn	0100nnnn00100001	MSB→Rn→T	1	LSB
SHLL	Rn	0100nnnn00000000	T←Rn←0	1	MSB
SHLR	Rn	0100nnnn00000001	0→Rn→T	1	LSB
SHLL2	Rn	0100nnnn00001000	Rn<<2→Rn	1	—
SHLR2	Rn	0100nnnn00001001	Rn>>2→Rn	1	—
SHLL8	Rn	0100nnnn00011000	Rn<<8→Rn	1	—
SHLR8	Rn	0100nnnn00011001	Rn>>8→Rn	1	—
SHLL16	Rn	0100nnnn00101000	Rn<<16→Rn	1	—
SHLR16	Rn	0100nnnn00101001	Rn>>16→Rn	1	—

6.1.5 分岐命令

表 6.6 分岐命令

命　令		命令コード	動　作	実行ステート	T ビット
BF	label	10001011dddddddd	T=0 のとき disp×2+PC→PC, T=1 のとき nop	3/1*2	—
BF/S	label	10001111dddddddd	遅延分岐、T=0 のとき disp×2+PC→PC,T=1 のとき nop	3/1*2	—
BT	label	10001001dddddddd	T=1 のとき disp×2+PC→PC, T=0 のとき nop	3/1*2	—
BT/S	label	10001101dddddddd	遅延分岐、T=1 のとき disp×2+PC→PC,T=0 のとき nop	2/1*2	—
BRA	label	1010dddddddddddd	遅延分岐、disp×2+PC→PC	2	—
BRAF	Rm	0000mmmm00100011	遅延分岐、Rm+PC→PC	2	—
BSR	label	1011dddddddddddd	遅延分岐、PC→PR, disp×2+PC→PC	2	—
BSRF	Rm	0000mmmm00000011	遅延分岐、PC→PR, Rm+PC→PC	2	—
JMP	@Rm	0100mmmm00101011	遅延分岐、Rm→PC	2	—
JSR	@Rm	0100mmmm00001011	遅延分岐、PC→PR, Rm→PC	2	—
RTS		0000000000001011	遅延分岐、PR→PC	2	—

【注】 *2　分岐しないときは 1 ステートになります。

6.1.6 システム制御命令

表 6.7 システム制御命令

命令		命令コード	動作	実行ステート	T ビット
CLRT		0000000000001000	0→T	1	0
CLRMAC		0000000000101000	0→MACH,MACL	1	—
LDC	Rm, SR	0100mmmm00001110	Rm→SR	1	LSB
LDC	Rm, GBR	0100mmmm00011110	Rm→GBR	1	—
LDC	Rm, VBR	0100mmmm00101110	Rm→VBR	1	—
LDC.L	@Rm+, SR	0100mmmm00000111	(Rm) →SR, Rm+4→Rm	3	LSB
LDC.L	@Rm+, GBR	0100mmmm00010111	(Rm) →GBR, Rm+4→Rm	3	—
LDC.L	@Rm+, VBR	0100mmmm00100111	(Rm) →VBR, Rm+4→Rm	3	—
LDS	Rm, MACH	0100mmmm00001010	Rm→MACH	1	—
LDS	Rm, MACL	0100mmmm00011010	Rm→MACL	1	—
LDS	Rm, PR	0100mmmm00101010	Rm→PR	1	—
LDS.L	@Rm+, MACH	0100mmmm00000110	(Rm) →MACH, Rm+4→Rm	1	—
LDS.L	@Rm+, MACL	0100mmmm00010110	(Rm) →MACL, Rm+4→Rm	1	—
LDS.L	@Rm+, PR	0100mmmm00100110	(Rm) →PR, Rm+4→Rm	1	—
NOP		0000000000001001	無操作	1	—
RTE		0000000000101011	遅延分岐、スタック領域→PC/SR	4	—
SETT		0000000000011000	1→T	1	1
SLEEP		0000000000011011	スリープ	3 *3	—
STC	SR, Rn	0000nnnn00000010	SR→Rn	1	—
STC	GBR, Rn	0000nnnn00010010	GBR→Rn	1	—
STC	VBR, Rn	0000nnnn00100010	VBR→Rn	1	—
STC.L	SR, @- Rn	0100nnnn00000011	Rn-4→Rn, SR→ (Rn)	2	—
STC.L	GBR, @- Rn	0100nnnn00010011	Rn-4→Rn, GBR→ (Rn)	2	—
STC.L	VBR, @- Rn	0100nnnn00100011	Rn-4→Rn, VBR→ (Rn)	2	—
STS	MACH, Rn	0000nnnn00001010	MACH→Rn	1	—
STS	MACL, Rn	0000nnnn00011010	MACL→Rn	1	—
STS	PR, Rn	0000nnnn00101010	PR→Rn	1	—
STS.L	MACH, @-Rn	0100nnnn00000010	Rn-4→Rn, MACH→ (Rn)	1	—
STS.L	MACL, @-Rn	0100nnnn00010010	Rn-4→Rn, MACL→ (Rn)	1	—
STS.L	PR, @-Rn	0100nnnn00100010	Rn-4→Rn, PR→ (Rn)	1	—
TRAPA	#imm	11000011iiiiiiii	PC/SR→スタック領域、(imm×4+VBR) →PC	8	—

【注】 *3 スリープ状態に遷移するまでのステート数です。

・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。

6.1.7 浮動小数点命令

表 6.8 浮動小数点命令

命　令		命令コード	動　作	実行ステート	T ビット
FABS	FRn	1111nnnn01011101	|FRn|→FRn	1	—
FADD	FRm, FRn	1111nnnnmmmm0000	FRn+FRm→FRn	1	—
FCMP/EQ	FRm, FRn	1111nnnnmmmm0100	(FRn=FRm)? 1:0→T	1	比較結果
FCMP/GT	FRm, FRn	1111nnnnmmmm0101	(FRn>FRm)? 1:0→T	1	比較結果
FDIV	FRm, FRn	1111nnnnmmmm0011	FRn/FRm→FRn	13	—
FLDI0	FRn	1111nnnn10001101	0×00000000→FRn	1	—
FLDI1	FRn	1111nnnn10011101	0×3F800000→FRn	1	—
FLDS	FRm, FPUL	1111mmmm00011101	FRm→FPUL	1	—
FLOAT	FPUL,FRn	1111nnnn00101101	(float)FPUL→FRn	1	—
FMAC	FR0,FRm,FRn	1111nnnnmmmm1110	FR0×FRm+FRn→FRn	1	—
FMOV	FRm, FRn	1111nnnnmmmm1100	FRm→FRn	1	—
FMOV.S	@(R0, Rm), FRn	1111nnnnmmmm0110	(R0+Rm) →FRn	1	—
FMOV.S	@Rm+, FRn	1111nnnnmmmm1001	(Rm) →FRn, Rm+=4	1	—
FMOV.S	@Rm, FRn	1111nnnnmmmm1000	(Rm) →FRn	1	—
FMOV.S	FRm, @(R0,Rn)	1111nnnnmmmm0111	FRm→ (R0+Rn)	1	—
FMOV.S	FRm, @-Rn	1111nnnnmmmm1011	Rn-=4, FRm→ (Rn)	1	—
FMOV.S	FRm, @Rn	1111nnnnmmmm1010	FRm→ (Rn)	1	—
FMUL	FRm, FRn	1111nnnnmmmm0010	FRn×FRm→FRn	1	—
FNEG	FRn	1111nnnn01001101	-FRn→FRn	1	—
FSTS	FPUL,FRn	1111nnnn00001101	FPUL→FRn	1	—
FSUB	FRm, FRn	1111nnnnmmmm0001	FRn-FRm→FRn	1	—
FTRC	FRm, FPUL	1111mmmm00111101	(long)FRm→FPUL	1	—

6.1.8 FPU に関する CPU 命令

表 6.9 FPU に関する CPU 命令

命　令		命令コード	動　作	実行ステート	T ビット
LDS	Rm,FPSCR	0100mmmm01101010	Rm→FPSCR	1	—
LDS	Rm,FPUL	0100mmmm01011010	Rm→FPUL	1	—
LDS.L	@Rm+, FPSCR	0100mmmm01100110	@Rm→FPSCR, Rm+=4	1	—
LDS.L	@Rm+, FPUL	0100mmmm01010110	@Rm→FPUL, Rm+=4	1	—
STS	FPSCR, Rn	0000nnnn01101010	FPSCR→Rn	1	—
STS	FPUL,Rn	0000nnnn01011010	FPUL→Rn	1	—
STS.L	FPSCR,@-Rn	0100nnnn01100010	Rn-=4, FPCSR→@Rn	1	—
STS.L	FPUL,@-Rn	0100nnnn01010010	Rn-=4, FPUL→@Rn	1	—

6.2 アルファベット順命令セット

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 6.10 アルファベット順命令セット

命令		命令コード	動作	実行ステート	T ビット
ADD	#imm,Rn	0111nnnniiiiiiii	Rn+imm→Rn	1	—
ADD	Rm,Rn	0011nnnnmmmm1100	Rn+Rm→Rn	1	—
ADDC	Rm,Rn	0011nnnnmmmm1110	Rn+Rm+T→Rn, キャリ→T	1	キャリ
ADDV	Rm,Rn	0011nnnnmmmm1111	Rn+Rm→Rn, オーバフロー→T	1	オーバフロー
AND	#imm,R0	11001001iiiiiiii	R0 & imm→R0	1	—
AND	Rm,Rn	0010nnnnmmmm1001	Rn & Rm→Rn	1	—
AND.B	#imm,@(R0,GBR)	11001101iiiiiiii	(R0+GBR) & imm→ (R0+GBR)	3	—
BF	label	10001011dddddddd	T=0 のとき disp×2+PC→PC, T=1 のとき nop	3/1*2	—
BF/S	label	10001111dddddddd	T=0 のとき disp×2+PC→PC, T=1 のとき nop	2/1*2	—
BRA	label	1010dddddddddddd	遅延分岐、disp×2+PC→PC	2	—
BRAF	Rm	0000mmmm00100011	遅延分岐、Rm+PC→PC	2	—
BSR	label	1011dddddddddddd	遅延分岐、PC→PR, disp×2+PC→PC	2	—
BSRF	Rm	0000mmmm00000011	遅延分岐、PC→PR, Rm+PC→PC	2	—
BT	label	10001001dddddddd	T=1 のとき disp×2+PC→PC, T=0 のとき nop	3/1*2	—
BT/S	label	10001101dddddddd	T=1 のとき disp×2+PC→PC, T=0 のとき nop	2/1*2	—
CLRMAC		0000000000101000	0→MACH, MACL	1	—
CLRT		0000000000001000	0→T	1	0
CMP/EQ	#imm,R0	10001000iiiiiiii	R0=imm のとき 1→T	1	比較結果
CMP/EQ	Rm,Rn	0011nnnnmmmm0000	Rn=Rm のとき 1→T	1	比較結果
CMP/GE	Rm,Rn	0011nnnnmmmm0011	有符号で Rn≧Rm のとき 1→T	1	比較結果
CMP/GT	Rm,Rn	0011nnnnmmmm0111	有符号で Rn>Rm のとき 1→T	1	比較結果
CMP/HI	Rm,Rn	0011nnnnmmmm0110	無符号で Rn>Rm のとき 1→T	1	比較結果
CMP/HS	Rm,Rn	0011nnnnmmmm0010	無符号で Rn≧Rm のとき 1→T	1	比較結果
CMP/PL	Rn	0100nnnn00010101	Rn>0 のとき 1→T	1	比較結果
CMP/PZ	Rn	0100nnnn00010001	Rn≧0 のとき 1→T	1	比較結果
CMP/STR	Rm,Rn	0010nnnnmmmm1100	いずれかのバイトが等しいとき 1→T	1	比較結果
DIV0S	Rm,Rn	0010nnnnmmmm0111	Rn の MSB→Q, Rm の MSB→M, M ^ Q→T	1	計算結果
DIV0U		0000000000011001	0→M/Q/T	1	0
DIV1	Rm,Rn	0011nnnnmmmm0100	1 ステップ除算(Rn÷Rm)	1	計算結果
DMULS.L	Rm,Rn	0011nnnnmmmm1101	符号付きで Rn×Rm→MACH, MACHL	2〜4*1	—
DMULU.L	Rm,Rn	0011nnnnmmmm0101	符号なしで Rn×Rm→MACH, MACL	2〜4*1	—
DT	Rn	0100nnnn00010000	Rn−1→Rn,Rn が 0 のとき 1→T Rn が 0 以外のとき 0→T	1	比較結果
EXTS.B	Rm,Rn	0110nnnnmmmm1110	Rm をバイトから符号拡張→Rn	1	—
EXTS.W	Rm,Rn	0110nnnnmmmm1111	Rm をワードから符号拡張→Rn	1	—
EXTU.B	Rm,Rn	0110nnnnmmmm1100	Rm をバイトからゼロ拡張→Rn	1	—

命　令		命令コード	動　作	実行ステート	Tビット
EXTU.W	Rm,Rn	0110nnnnmmmm1101	Rm をワードからゼロ拡張→Rn	1	—
FABS	FRn	1111nnnn01011101	|FRn|→FRn	1	—
FADD	FRm, FRn	1111nnnnmmmm0000	FRn+FRm→FRn	1	—
FCMP/EQ	FRm, FRn	1111nnnnmmmm0100	(FRn=FRm)? 1:0→T	1	比較結果
FCMP/GT	FRm, FRn	1111nnnnmmmm0101	(FRn>FRm)? 1:0→T	1	比較結果
FDIV	FRm, FRn	1111nnnnmmmm0011	FRn/FRm→FRn	13	—
FLDI0	FRn	1111nnnn10001101	0×00000000→FRn	1	—
FLDI1	FRn	1111nnnn10011101	0×3F800000→FRn	1	—
FLDS	FRm, FPUL	1111mmmm00011101	FRm→FPUL	1	—
FLOAT	FPUL,FRn	1111nnnn00101101	(float)FPUL→FRn	1	—
FMAC	FR0,FRm,FRn	1111nnnnmmmm1110	FR0×FRm+FRn→FRn	1	—
FMOV	FRm, FRn	1111nnnnmmmm1100	FRm→FRn	1	—
FMOV.S	@(R0, Rm), FRn	1111nnnnmmmm0110	(R0+Rm) →FRn	1	—
FMOV.S	@Rm+, FRn	1111nnnnmmmm1001	(Rm) →FRn, Rm+=4	1	—
FMOV.S	@Rm, FRn	1111nnnnmmmm1000	(Rm) →FRn	1	—
FMOV.S	FRm, @(R0,Rn)	1111nnnnmmmm0111	FRm→ (R0+Rn)	1	—
FMOV.S	FRm, @-Rn	1111nnnnmmmm1011	Rn-=4, FRm→ (Rn)	1	—
FMOV.S	FRm, @Rn	1111nnnnmmmm1010	FRm→ (Rn)	1	—
FMUL	FRm, FRn	1111nnnnmmmm0010	FRn×FRm→FRn	1	—
FNEG	FRn	1111nnnn01001101	-FRn→FRn	1	—
FSTS	FPUL,FRn	1111nnnn00001101	FPUL→FRn	1	—
FSUB	FRm, FRn	1111nnnnmmmm0001	FRn-FRm→FRn	1	—
FTRC	FRm, FPUL	1111mmmm00111101	(long)FRm→FPUL	1	—
JMP	@Rm	0100mmmm00101011	遅延分岐、Rm→PC	2	—
JSR	@Rm	0100mmmm00001011	遅延分岐、PC→PR, Rm→PC	2	—
LDC	Rm,GBR	0100mmmm00011110	Rm→GBR	1	—
LDC	Rm,SR	0100mmmm00001110	Rm→SR	1	LSB
LDC	Rm,VBR	0100mmmm00101110	Rm→VBR	1	—
LDC.L	@Rm+,GBR	0100mmmm00010111	(Rm) →GBR, Rm+4→Rm	3	—
LDC.L	@Rm+,SR	0100mmmm00000111	(Rm) →SR, Rm+4→Rm	3	LSB
LDC.L	@Rm+,VBR	0100mmmm00100111	(Rm) →VBR, Rm+4→Rm	3	—
LDS	Rm,FPSCR	0100mmmm01101010	Rm→FPSCR	1	—
LDS	Rm,FPUL	0100mmmm01011010	Rm→FPUL	1	—
LDS	Rm,MACH	0100mmmm00001010	Rm→MACH	1	—
LDS	Rm,MACL	0100mmmm00011010	Rm→MACL	1	—
LDS	Rm,PR	0100mmmm00101010	Rm→PR	1	—
LDS.L	@Rm+, FPSCR	0100mmmm01100110	@Rm→FPSCR, Rm+=4	1	—
LDS.L	@Rm+, FPUL	0100mmmm01010110	@Rm→FPUL, Rm+=4	1	—
LDS.L	@Rm+,MACH	0100mmmm00000110	(Rm) →MACH, Rm+4→Rm	1	—
LDS.L	@Rm+,MACL	0100mmmm00010110	(Rm) →MACL, Rm+4→Rm	1	—
LDS.L	@Rm+,PR	0100mmmm00100110	(Rm) →PR, Rm+4→Rm	1	—
MAC.L	@Rm+,@Rn+	0000nnnnmmmm1111	符号付きで(Rn)×(Rm)+MAC→MAC	3/(2〜4)※1	—
MAC.W	@Rm+,@Rn+	0100nnnnmmmm1111	符号付きで(Rn)×(Rm)+MAC→MAC	3/(2) ※1	—

命　令		命令コード	動　作	実行ステート	Tビット
MOV	#imm,Rn	1110nnnniiiiiiii	imm→符号拡張→Rn	1	—
MOV	Rm,Rn	0110nnnnmmmm0011	Rm→Rn	1	—
MOV.B	@(disp,GBR),R0	11000100dddddddd	(disp+GBR) →符号拡張→R0	1	—
MOV.B	@(disp,Rm),R0	10000100mmmmdddd	(disp+Rm) →符号拡張→R0	1	—
MOV.B	@(R0,Rm),Rn	0000nnnnmmmm1100	(R0+Rm) →符号拡張→Rn	1	—
MOV.B	@Rm+,Rn	0110nnnnmmmm0100	(Rm) →符号拡張→Rn, Rm+1→Rm	1	—
MOV.B	@Rm,Rn	0110nnnnmmmm0000	(Rm) →符号拡張→Rn	1	—
MOV.B	R0,@(disp,GBR)	11000000dddddddd	R0→ (disp+GBR)	1	—
MOV.B	R0,@(disp,Rn)	10000000nnnndddd	R0→ (disp+Rn)	1	—
MOV.B	Rm,@(R0,Rn)	0000nnnnmmmm0100	Rm→ (R0+Rn)	1	—
MOV.B	Rm,@-Rn	0010nnnnmmmm0100	Rn－1→Rn, Rm→ (Rn)	1	—
MOV.B	Rm,@Rn	0010nnnnmmmm0000	Rm→ (Rn)	1	—
MOV.L	@(disp,GBR),R0	11000110dddddddd	(disp×4+GBR) →R0	1	—
MOV.L	@(disp,PC),Rn	1101nnnndddddddd	(disp×4+PC) →Rn	1	—
MOV.L	@(disp,Rm),Rn	0101nnnnmmmmdddd	(disp×4+Rm) →Rn	1	—
MOV.L	@(R0,Rm),Rn	0000nnnnmmmm1110	(R0+Rm) →Rn	1	—
MOV.L	@Rm+,Rn	0110nnnnmmmm0110	(Rm) →Rn, Rm+4→Rm	1	—
MOV.L	@Rm,Rn	0110nnnnmmmm0010	(Rm) →Rn	1	—
MOV.L	R0,@(disp,GBR)	11000010dddddddd	R0→ (disp×4+GBR)	1	—
MOV.L	Rm,@(disp,Rn)	0001nnnnmmmmdddd	Rm→ (disp×4+Rn)	1	—
MOV.L	Rm,@(R0,Rn)	0000nnnnmmmm0110	Rm→ (R0+Rn)	1	—
MOV.L	Rm,@-Rn	0010nnnnmmmm0110	Rn－4→Rn, Rm→ (Rn)	1	—
MOV.L	Rm,@Rn	0010nnnnmmmm0010	Rm→ (Rn)	1	—
MOV.W	@(disp,GBR),R0	11000101dddddddd	(disp×2+GBR) →符号拡張→R0	1	—
MOV.W	@(disp,PC),Rn	1001nnnndddddddd	(disp×2+PC) →符号拡張→Rn	1	—
MOV.W	@(disp,Rm),R0	10000101mmmmdddd	(disp×2+Rm) →符号拡張→R0	1	—
MOV.W	@(R0,Rm),Rn	0000nnnnmmmm1101	(R0+Rm) →符号拡張→Rn	1	—
MOV.W	@Rm+,Rn	0110nnnnmmmm0101	(Rm) →符号拡張→Rn, Rm+2→Rm	1	—
MOV.W	@Rm,Rn	0110nnnnmmmm0001	(Rm) →符号拡張→Rn	1	—
MOV.W	R0,@(disp,GBR)	11000001dddddddd	R0→ (disp×2+GBR)	1	—
MOV.W	R0,@(disp,Rn)	10000001nnnndddd	R0→ (disp×2+Rn)	1	—
MOV.W	Rm,@(R0,Rn)	0000nnnnmmmm0101	Rm→ (R0+Rn)	1	—
MOV.W	Rm,@-Rn	0010nnnnmmmm0101	Rn－2→Rn, Rm→ (Rn)	1	—
MOV.W	Rm,@Rn	0010nnnnmmmm0001	Rm→ (Rn)	1	—
MOVA	@(disp,PC),R0	11000111dddddddd	disp×4+PC→R0	1	—
MOVT	Rn	0000nnnn00101001	T→Rn	1	—
MUL.L	Rm,Rn	0000nnnnmmmm0111	Rn×Rm→MACL	2〜4*1	—
MULS.W	Rm,Rn	0010nnnnmmmm1111	符号付きで Rn×Rm→MACL	1〜3*1	—
MULU.W	Rm,Rn	0010nnnnmmmm1110	符号なしで Rn×Rm→MACL	1〜3*1	—
NEG	Rm,Rn	0110nnnnmmmm1011	0－Rm→Rn	1	—
NEGC	Rm,Rn	0110nnnnmmmm1010	0－Rm－T→Rn, ボロー→T	1	ボロー
NOP		0000000000001001	無操作	1	—
NOT	Rm,Rn	0110nnnnmmmm0111	~Rm→Rn	1	—

命　令		命令コード	動　作	実行ステート	Tビット
OR	#imm,R0	11001011iiiiiiii	R0 | imm→R0	1	—
OR	Rm,Rn	0010nnnnmmmm1011	Rn | Rm→Rn	1	—
OR.B	#imm,@(R0,GBR)	11001111iiiiiiii	(R0+GBR) | imm→ (R0+GBR)	3	—
ROTCL	Rn	0100nnnn00100100	T←Rn←T	1	MSB
ROTCR	Rn	0100nnnn00100101	T→Rn→T	1	LSB
ROTL	Rn	0100nnnn00000100	T←Rn←MSB	1	MSB
ROTR	Rn	0100nnnn00000101	LSB→Rn→T	1	LSB
RTE		0000000000101011	遅延分岐、スタック領域→PC/SR	4	LSB
RTS		0000000000001011	遅延分岐、PR→PC	2	—
SETT		0000000000011000	1→T	1	1
SHAL	Rn	0100nnnn00100000	T←Rn←0	1	MSB
SHAR	Rn	0100nnnn00100001	MSB→Rn→T	1	LSB
SHLL	Rn	0100nnnn00000000	T←Rn←0	1	MSB
SHLL2	Rn	0100nnnn00001000	Rn<<2→Rn	1	—
SHLL8	Rn	0100nnnn00011000	Rn<<8→Rn	1	—
SHLL16	Rn	0100nnnn00101000	Rn<<16→Rn	1	—
SHLR	Rn	0100nnnn00000001	0→Rn→T	1	LSB
SHLR2	Rn	0100nnnn00001001	Rn>>2→Rn	1	—
SHLR8	Rn	0100nnnn00011001	Rn>>8→Rn	1	—
SHLR16	Rn	0100nnnn00101001	Rn>>16→Rn	1	—
SLEEP		0000000000011011	スリープ	3	—
STC	GBR,Rn	0000nnnn00010010	GBR→Rn	1	—
STC	SR,Rn	0000nnnn00000010	SR→Rn	1	—
STC	VBR,Rn	0000nnnn00100010	VBR→Rn	1	—
STC.L	GBR,@-Rn	0100nnnn00010011	Rn−4→Rn, GBR→ (Rn)	2	—
STC.L	SR,@-Rn	0100nnnn00000011	Rn−4→Rn, SR→ (Rn)	2	—
STC.L	VBR,@-Rn	0100nnnn00100011	Rn−4→Rn, VBR→ (Rn)	2	—
STS	FPSCR, Rn	0000nnnn01101010	FPSCR→Rn	1	—
STS	FPUL,Rn	0000nnnn01011010	FPUL→Rn	1	—
STS	MACH,Rn	0000nnnn00001010	MACH→Rn	1	—
STS	MACL,Rn	0000nnnn00011010	MACL→Rn	1	—
STS	PR,Rn	0000nnnn00101010	PR→Rn	1	—
STS.L	FPSCR,@-Rn	0100nnnn01100010	Rn-=4, FPCSR→@Rn	1	—
STS.L	FPUL,@-Rn	0100nnnn01010010	Rn-=4, FPUL→@Rn	1	—
STS.L	MACH,@-Rn	0100nnnn00000010	Rn−4→Rn, MACH→ (Rn)	1	—
STS.L	MACL,@-Rn	0100nnnn00010010	Rn−4→Rn, MACL→ (Rn)	1	—
STS.L	PR,@-Rn	0100nnnn00100010	Rn−4→Rn, PR→ (Rn)	1	—
SUB	Rm,Rn	0011nnnnmmmm1000	Rn−Rm→ Rn	1	—
SUBC	Rm,Rn	0011nnnnmmmm1010	Rn−Rm−T→Rn, ボロー→T	1	ボロー
SUBV	Rm,Rn	0011nnnnmmmm1011	Rn−Rm→Rn, アンダフロー→T	1	アンダフロー
SWAP.B	Rm,Rn	0110nnnnmmmm1000	Rm→下位2バイトの上下バイト交換→Rn	1	—
SWAP.W	Rm,Rn	0110nnnnmmmm1001	Rm→上下ワード交換→Rn	1	—
TAS.B	@Rn	0100nnnn00011011	(Rn)が0のとき 1→T, 1→MSBof(Rn)	4	テスト結果

命　令		命令コード	動　作	実行ステート	T ビット
TRAPA	#imm	11000011iiiiiiii	PC/SR→スタック領域, (imm×4+VBR)→PC	8	—
TST	#imm,R0	11001000iiiiiiii	R0 & imm, 結果が 0 のとき 1→T	1	テスト結果
TST	Rm,Rn	0010nnnnmmmm1000	Rn & Rm, 結果が 0 のとき 1→T	1	テスト結果
TST.B	#imm,@(R0,GBR)	11001100iiiiiiii	(R0+GBR) & imm, 結果が 0 のとき 1→T	3	テスト結果
XOR	#imm,R0	11001010iiiiiiii	R0 ^ imm→R0	1	—
XOR	Rm,Rn	0010nnnnmmmm1010	Rn ^ Rm→Rn	1	—
XOR.B	#imm,@(R0,GBR)	11001110iiiiiiii	(R0+GBR) ^ imm→ (R0+GBR)	3	—
XTRCT	Rm,Rn	0010nnnnmmmm1101	Rm と Rn の中央 32 ビット→Rn	1	—

【注】 *1　通常ステートを示します。
*2　分岐しないときは 1 ステートになります。

7. 各命令の説明

7.1 命令説明のフォーム

以下の形式でアルファベット順に説明します。

命令の名称　命令の機能（英文）　：命令の分類

命令の機能　　**（遅延分岐命令、または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。

(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。

```
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と、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。

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

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

各レジスタの本体

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

SR の構造の定義

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

SR 内ビットの定義

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

エラー表示関数

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。**

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

@ (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ていないコードは一般不当命令として扱われ、不当命令例外処理を発生します。**

また、FPU **をモジュールストップビットにより停止状態にしたときは、浮動小数点命令および** FPU **に関する** CPU **命令は不当命令として扱われます。**

*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分岐命令であると（これをスロット不当命令といいます）、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**

7.2 CPU 命令

7.2.1 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	0111nnnniiiiiiii	1	—

(1) **説明**

汎用レジスタ Rn の内容と Rm とを加算し、結果を Rn に格納します。
汎用レジスタ Rn と 8 ビットのイミディエイトデータとの加算も可能です。
8 ビットのイミディエイトデータは 32 ビットに符号拡張しますので減算との兼用が可能です。

(2) **動作内容**

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

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

(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'FFFFFFFF
```

7.2.2 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'FFFFFFFF
                ;実行後 T=1,R1=H'00000000
ADDC  R2,R0     ;実行前 T=1,R0=H'00000000,R2=H'00000000
                ;実行後 T=0,R0=H'00000001
```

7.2.3 ADDV ADD with (Vflag)overflow check：算術演算命令

オーバフロー付き 2 進加算

書式	動作概略	命令コード	実行ステート	T ビット
ADDV Rm,Rn	Rn+Rm→Rn, オーバフロー→T	0011nnnnmmmm1111	1	オーバフロー

(1) 説明

汎用レジスタ Rn の内容と Rm とを加算し、結果を Rn に格納します。オーバフローが発生すると、T ビットをセットします。

(2) ■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
```

7.2.4 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	11001001iiiiiiii	1	—
AND.B #imm,@(R0,GBR)	(R0+GBR) & imm → (R0+GBR)	11001101iiiiiiii	3	—

(1) 説明

汎用レジスタ Rn の内容と Rm の論理積をとり、結果を Rn に格納します。

汎用レジスタ R0 とゼロ拡張した 8 ビットのイミディエイトデータとの論理積、もしくはインデックス付き GBR 間接アドレッシングモードで 8 ビットのメモリと 8 ビットのイミディエイトデータとの論理積が可能です。

(2) 注意

AND #imm,R0 では演算の結果、R0 の上位 24 ビットは常にクリアされます。

(3) 動作内容

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

ANDI(long i)    /* AND #imm,R0 */
{
    R[0]&=(0x000000FF & (long)i);
    PC+=2;
}

ANDM(long i)    /* AND.B #imm,@(R0,GBR) */
{
    long temp;

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

(4) 使用例

```
AND   R0,R1                ;実行前 R0=H'AAAAAAAA,R1=H'55555555
                           ;実行後 R1=H'00000000
AND   #H'0F,R0             ;実行前 R0=H'FFFFFFFF
                           ;実行後 R0=H'0000000F
AND.B #H'80,@(R0,GBR)      ;実行前 @(R0,GBR)=H'A5
                           ;実行後 @(R0,GBR)=H'80
```

7.2.5 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。

(3) 動作内容

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

    if ((d&0x80)==0) disp=(0x000000FF & (long)d);
    else disp=(0xFFFFFF00 | (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                     ; ←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
```

7.2.6 BF/S Branch if False with delay Slot：分岐命令

条件付き遅延分岐 **遅延分岐命令**

書式	動作概略	命令コード	実行ステート	T ビット
BF/S label	T=0 **のとき** disp×2+PC→PC, T=1 **のとき** nop	10001111dddddddd	2/1	—

(1) 説明

T ビットを参照する条件付き遅延分岐命令です。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、アドレスエラーと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。

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。

(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=(0xFFFFFF00 | (long)d);
    if (T==0) {
        PC=PC+(disp<<1)+4;
        Delay_Slot(temp+2);
    }
    else PC+=2;
}
```

(4) **使用例**

```
        CLRT                    ;常に T=0
        BT/S  TRGET_T           ;T=0 のため分岐しません。
        NOP                     ;
        BF/S  TRGET_F           ;T=0 のため TRGET 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。

7.2.7 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	1010ddddddddddddd	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。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、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、アドレスエラーと割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

(3) 動作内容

```
BRA(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);
}
```

(4) 使用例

```
    BRA  TRGET              ;TRGET へ分岐します。
    ADD  R0,R1              ;分岐に先立ち実行します。
    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。

7.2.8 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 Rm 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、アドレスエラーと割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

(3) 動作内容

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

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

(4) 使用例

```
    MOV.L #(TRGET-BRAF_PC),R0      ;ディスプレースメントを設定します。
    BRAF  R0                       ;TRGET へ分岐します。
    ADD   R0,R1                    ;分岐に先立ち実行します。
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。

7.2.9 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	1011dddddddddddd	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との間には、アドレスエラーと割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

(3) 動作内容

```
BSR(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。

7.2.10 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 Rm 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、アドレスエラーと割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

(3) 動作内容

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

(4) 使用例

```
MOV.L #(TRGET-BSRF_PC),R0      ;ディスプレースメントを設定します。
        BSRF  R0               ;TRGET へ分岐します。
        MOV   R3,R4            ;分岐に先立ち実行します。
      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            ;
         ・・・・・
         ・・・・・
      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。

7.2.11 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 abel	T=1 **のとき** disp×2+PC→PC, T=0 **のとき** nop	10001001dddddddd	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。

(3) 動作内容

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

    if ((d&0x80)==0) disp=(0x000000FF & (long)d);
    else disp=(0xFFFFFF00 | (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
```

7.2.12 BT/S Branch if True with delay Slot：分岐命令

条件付き遅延分岐 **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 ビットを参照する条件付き遅延分岐命令です。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、アドレスエラーと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。

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。

(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=(0xFFFFFF00 | (long)d);
    if (T==1) {
        PC=PC+(disp<<1)+4;
        Delay_Slot(temp+2);
    }
    else PC+=2;
}
```

(4) 使用例

```
      SETT                    ;常に T=1
      BF/S TRGET_F            ;T=1 のため分岐しません。
      NOP                     ;
      BT/S TRGET_T            ;T=1 のため TRGET_T に分岐します。
      ADD R0,R1               ;分岐に先立ち実行します。
      NOP                     ; ←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。

7.2.13 CLRMAC CLeaR MAC register：システム制御命令

MAC レジスタのクリア

書式	動作概略	命令コード	実行ステート	T ビット
CLRMAC	0→MACH, MACL	0000000000101000	1	—

(1) 説明

MACH、MACL レジスタをクリアします。

(2) 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+        ;
```

7.2.14 CLRT CLeaR Tbit：システム制御命令

T ビットのクリア

書式	動作概略	命令コード	実行ステート	T ビット
CLRT	0→T	0000000000001000	1	0

(1) 説明

T ビットをクリアします。

(2) 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
```

7.2.15 CMP/cond CoMPare conditionally：算術演算命令

比較

書式	動作概略	命令コード	実行ステート	T ビット
CMP/EQ Rm,Rn	Rn=Rm **のとき** 1→T	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	**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。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。

ニーモニック	説明
CMP/EQ Rm,Rn	Rn=Rm **のとき** T=1
CMP/GE Rm,Rn	**有符号値として** Rn≧Rm **のとき** T=1
CMP/GT Rm,Rn	**有符号値として** Rn>Rm **のとき** T=1
CMP/HI Rm,Rn	**無符号値として** Rn>Rm **のとき** T=1
CMP/HS Rm,Rn	**無符号値として** Rn≧Rm **のとき** T=1
CMP/PL Rn	Rn>0 **のとき** T=1
CMP/PZ Rn	Rn≧0 **のとき** T=1
CMP/STR Rm,Rn	**いずれかのバイトが等しいとき** T=1
CMP/EQ #imm,R0	R0=imm **のとき** T=1

(2) 動作内容

```
CMPEQ(long m, long n)    /* CMP_EQ Rm,Rn */
{
    if (R[n]==R[m]) T=1;
    else T=0;
    PC+=2;
}
```

```
CMPGE(long m, long n)    /* CMP_GE Rm,Rn */
{
    if ((long)R[n]>=(long)R[m]) T=1;
    else T=0;
    PC+=2;
}

CMPGT(long m, long n)    /* CMP_GT Rm,Rn */
{
    if ((long)R[n]>(long)R[m]) T=1;
    else T=0;
    PC+=2;
}

CMPHI(long m, long n)    /* CMP_HI Rm,Rn */
{
    if ((unsigned long)R[n]>(unsigned long)R[m]) T=1;
    else T=0;
    PC+=2;
}

CMPHS(long m, long n)    /* CMP_HS Rm,Rn */
{
    if ((unsigned long)R[n]>=(unsigned long)R[m]) T=1;
    else T=0;
    PC+=2;
}

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

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

CMPSTR(long m, long n)    /* CMP_STR Rm,Rn */
{
    unsigned long temp;
    long HH,HL,LH,LL;

    temp=R[n]^R[m];
    HH=(temp>>12)&0x000000FF;
    HL=(temp>>8)&0x000000FF;
    LH=(temp>>4)&0x000000FF;
    LL=temp&0x000000FF;
    HH=HH&&HL&&LH&&LL;
    if (HH==0) T=1;
    else T=0;
    PC+=2;
}
CMPIM(long i)    /* CMP_EQ #imm,R0 */
{
    long imm;

    if ((i&0x80)==0) imm=(0x000000FF & (long i));
    else imm=(0xFFFFFF00 | (long i));
    if (R[0]==imm) T=1;
    else T=0;
    PC+=2;
}
```

(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="XYCZ"
    BT      TRGET_T        ;T=1 なので分岐します。
```

7.2.16 DIV0S DIVide(step0) as Signed：算術演算命令

符号付き除算の初期化

書式	動作概略	命令コード	実行ステート	T ビット
DIV0S Rm,Rn	Rn の MSB→Q, Rm の MSB→M, M ^ Q→T	0010nnnnmmmm0111	1	計算結果

(1) 説明

符号付き除算の初期設定をします。本命令に続けて 1 桁分の除算をする DIV1 命令などを組み合せて、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。

7.2.17 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。

7.2.18 DIV1 DIVide 1 step：算術演算命令

除算

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

(1) 説明

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

1 ステップ除算とは、被除数を左に 1 ビットシフトし、それから除数を減算し、結果の正負によって商のビットを Q ビットに反映するという処理を実行します。

割り算で余りを求めるには、DIV1 命令を用いて商を求めたあと、

（被除数）－（除数）×（商）=（余り）

として求めてください。なお、周辺機能として除算器を搭載している SH7600 シリーズでは、除算器の機能により余りを求めることができます。

ゼロ除算とオーバフローの検出は用意していません。除算の前にゼロ除算とオーバフロー除算をチェックしてください。剰余の演算は用意していません。除数と求められた商との積を求めて、被除数から減算して剰余を求めてください。

最初に、DIV0S または DIV0U で初期設定します。DIV1 を除数のビット数分繰り返します。商が 17 ビット以上必要なとき、ROTCL を DIV1 の前に置きます。詳しい 除算のシーケンスは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;
            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
XOR      R2,R2          ;R2=0
MOV      R1,R3          ;
ROTCL    R3             ;
SUBC     R2,R1          ;被除数が負のとき、-1 する。
DIV0S    R0,R1          ;フラグの初期化
.arepeat  16            ;
DIV1     R0,R1          ;16 回繰り返し
.aendr                  ;
EXTS.W   R1,R1          ;
ROTCL    R1             ;R1=商(1 の補数表現)
ADDC     R2,R1          ;商の MSB が 1 のとき、+1 して 2 の補数表現に変換
EXTS.W   R1,R1          ;R1=商(2 の補数表現)
```

(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 回繰り返し
DIV1     R0,R1          ;
.aendr                  ;
ROTCL    R2             ;R2=商(1 の補数表現)
ADDC     R3,R2          ;商の MSB が 1 のとき、+1 して 2 の補数表現に変換
                        ;R2=商(2 の補数表現)
```

7.2.19 DMULS.L Double-length MULtiply as Signed：算術演算命令

符号付き倍精度乗算

書式	動作概略	命令コード	実行ステート	T ビット
DMULS.L Rm,Rn	**符号付きで** Rn×Rm→MACH, MACL	0011nnnnmmmm1101	2〜4	—

(1) 説明

汎用レジスタ Rn の内容と Rm を 32 ビットで乗算し、結果の 64 ビットを MACH レジスタと MACL レジスタに格納します。演算は符号付き算術演算で行います。

(2) 動作内容

```
DMULS(long m, long n)    /* DMULS.L Rm,Rn */
{
    unsigned long RnL,RnH,RmL,RmH,Res0,Res1,Res2;
    unsigned long temp0,temp1,temp2,temp3;
    long tempm,tempn,fnLmL;

    tempn=(long)R[n];
    tempm=(long)R[m];
    if (tempn<0) tempn=0-tempn;
    if (tempm<0) tempm=0-tempm;
    if ((long)(R[n]^R[m])<0) fnLmL=-1;
    else fnLmL=0;

    temp1=(unsigned long)tempn;
    temp2=(unsigned long)tempm;

    RnL=temp1&0x0000FFFF;
    RnH=(temp1>>16)&0x0000FFFF;
    RmL=temp2&0x0000FFFF;
    RmH=(temp2>>16)&0x0000FFFF;

    temp0=RmL*RnL;
    temp1=RmH*RnL;
    temp2=RmL*RnH;
```

```
    temp3=RmH*RnH;

    Res2=0
    Res1=temp1+temp2;
    if (Res1<temp1) Res2+=0x00010000;

    temp1=(Res1<<16)&0xFFFF0000;
    Res0=temp0+temp1;
    if (Res0<temp0) Res2++;

    Res2=Res2+((Res1>>16)&0x0000FFFF)+temp3;

    if (fnLmL<0) {
        Res2=~Res2;
        if (Res0==0)
            Res2++;
        else
            Res0=(~Res0)+1;
    }

    MACH=Res2;
    MACL=Res0;
    PC+=2;
}
```

(3) 使用例

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

7.2.20 DMULU.L Double-length MULtiply as Unsigned：算術演算命令

符号なし倍精度乗算

書式	動作概略	命令コード	実行ステート	T ビット
DMULU.L Rm,Rn	**符号なしで** Rn×Rm→MACH, MACL	0011nnnnmmmm0101	2〜4	—

(1) 説明

汎用レジスタ Rn の内容と Rm を 32 ビットで乗算し、結果の 64 ビットを MACH レジスタと MACL レジスタに格納します。演算は符号なし算術演算で行います。

(2) 動作内容

```
DMULU(long m, long n)    /* DMULU.L Rm,Rn */
{
    unsigned long RnL,RnH,RmL,RmH,Res0,Res1,Res2;
    unsigned long temp0,temp1,temp2,temp3;

    RnL=R[n]&0x0000FFFF;
    RnH=(R[n]>>16)&0x0000FFFF;

    RmL=R[m]&0x0000FFFF;
    RmH=(R[m]>>16)&0x0000FFFF;

    temp0=RmL*RnL;
    temp1=RmH*RnL;
    temp2=RmL*RnH;
    temp3=RmH*RnH;

    Res2=0
    Res1=temp1+temp2;
    if (Res1<temp1) Res2+=0x00010000;

temp1=(Res1<<16)&0xFFFF0000;
    Res0=temp0+temp1;
    if (Res0<temp0) Res2++;
    Res2=Res2+((Res1>>16)&0x0000FFFF)+temp3;

    MACH=Res2;
    MACL=Res0;
    PC+=2;
}
```

(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
```

7.2.21 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 以外のとき 0→T	0100nnnn00010000	1	比較結果

(1) 説明

汎用レジスタ Rn の内容を 1 デクリメントして、結果を 0（ゼロ）と比較します。結果が 0 のとき T ビットを 1 にセットします。結果が 0 以外のとき、T ビットを 0 にセットします。

(2) 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 RS            ;R5 の値をディリリメントし、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)。
```

7.2.22 EXTS EXTend as Signed：算術演算命令

符号拡張

書式		動作概略	命令コード	実行ステート	T ビット
EXTS.B	Rm,Rn	Rm をバイトから符号拡張→Rn	0110nnnnmmmm1110	1	—
EXTS.W	Rm,Rn	Rm をワードから符号拡張→Rn	0110nnnnmmmm1111	1	—

(1) 説明

汎用レジスタ Rm の内容を符号拡張して、結果を Rn に格納します。

バイト指定のとき、Rn のビット 8 からビット 31 に Rm のビット 7 の内容を転送します。ワード指定のとき、Rn のビット 16 からビット 31 に Rm のビット 15 の内容を転送します。

(2) 動作内容

```
EXTSB(long m, long n)    /* EXTS.B Rm,Rn */
{
    R[n]=R[m];
    if ((R[m]&0x00000080)==0) R[n]&=0x000000FF;
    else R[n]|=0xFFFFFF00;
    PC+=2;
}

EXTSW(long m, long n)    /* EXTS.W Rm,Rn */
{
    R[n]=R[m];
    if ((R[m]&0x00008000)==0) R[n]&=0x0000FFFF;
    else R[n]|=0xFFFF0000;
    PC+=2;
}
```

(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
```

7.2.23 EXTU EXTend as Unsigned：算術演算命令

ゼロ拡張

書式	動作概略	命令コード	実行ステート	T ビット
EXTU.B Rm,Rn	Rm をバイトからゼロ拡張→Rn	0110nnnnmmmm1100	1	—
EXTU.W Rm,Rn	Rm をワードからゼロ拡張→Rn	0110nnnnmmmm1101	1	—

(1) 説明

汎用レジスタ Rm の内容をゼロ拡張して、結果を Rn に格納します。

バイト指定のとき、Rn のビット 8 からビット 31 に 0 を転送します。ワード指定のとき、Rn のビット 16 からビット 31 に 0 を転送します。

(2) 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
```

7.2.24 JMP JuMP：分岐命令

無条件分岐 **遅延分岐命令**

書式	動作概略	命令コード	実行ステート	Tビット
JMP @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、アドレスエラーと割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

(3) 動作内容

```
JMP(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               ;分岐に先立ち実行します。
        .align  4
JMP_TABLE: .data.l  TRGET           ;ジャンプテーブル
      ・・・・・・・・・・・
TRGET:    ADD     #1,R1             ; ←分岐先
```

【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。

7.2.25 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→PR, 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、アドレスエラーと割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

(3) 動作内容

```
JSR(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。

7.2.26 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 **割り込み禁止命令**

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	1	LSB
LDC Rm,GBR	Rm→GBR	0100mmmm00011110	1	—
LDC Rm,VBR	Rm→VBR	0100mmmm00101110	1	—
LDC.L @Rm+,SR	(Rm) →SR, Rm+4→Rm	0100mmmm00000111	3	LSB
LDC.L @Rm+,GBR	(Rm) →GBR, Rm+4→Rm	0100mmmm00010111	3	—
LDC.L @Rm+,VBR	(Rm) →VBR, Rm+4→Rm	0100mmmm00100111	3	—

(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 に格納します。

(2) 注意

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。（アドレスエラーは受け付けます。）

(3) 動作内容

```
LDCSR(long m)    /* LDC Rm,SR */
{
    SR=R[m]&0x000003F3;
    PC+=2;
}

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

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

```
LDCMSR(long m)    /* LDC.L @Rm+,SR */
{
    SR=Read_Long(R[m])&0x000003F3;
    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;
}
```

(4) **使用例**

```
LDC    R0,SR                ;実行前  R0=H'FFFFFFFF,SR=H'00000000
                            ;実行後  SR=H'000003F3
LDC.L  @R15+,GBR            ;実行前 R15=H'10000000
                            ;実行後 R15=H'10000004,GBR=@H'10000000
```

7.2.27 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 **割り込み禁止命令**

書式	動作概略	命令コード	実行ステート	Tビット
LDS Rm,MACH	Rm→MACH	0100mmmm00001010	1	—
LDS Rm,MACL	Rm→MACL	0100mmmm00011010	1	—
LDS Rm,PR	Rm→PR	0100mmmm00101010	1	—
LDS.L @Rm+,MACH	(Rm) →MACH, Rm+4→Rm	0100mmmm00000110	1	—
LDS.L @Rm+,MACL	(Rm) →MACL, Rm+4→Rm	0100mmmm00010110	1	—
LDS.L @Rm+,PR	(Rm) →PR, Rm+4→Rm	0100mmmm00100110	1	—

(1) 説明

ソースオペランドをシステムレジスタ MACH、MACL、PR に格納します。

(2) 注意

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。（アドレスエラーは受け付けます。）

(3) 動作内容

```
LDSMACH(long m)    /* LDS Rm,MACH */
{
    MACH=R[m];
    PC+=2;
}

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

LDSPR(long m)    /* LDS Rm,PR */
{
    PR=R[m];
    PC+=2;
}
```

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

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

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

(4) **使用例**

```
LDS    R0,PR                ;実行前  R0=H'12345678,PR=H'00000000
                            ;実行後  PR=H'12345678
LDS.L  @R15+,MACL           ;実行前 R15=H'10000000
                            ;実行後 R15=H'10000004,MACL=@H'10000000
```

7.2.28 MAC.L Multiply and ACcumulate Long：算術演算命令

倍精度積和演算

書式	動作概略	命令コード	実行ステート	T ビット
MAC.L @Rm+,@Rn+	**符号付きで** (Rn)×(Rm)+MAC→MAC	0000nnnnmmmm1111	3/(2〜4)	—

(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'00007FFFFFFFFFFF（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;
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++;
    Res2+=(MACH&0x0000FFFF);

    if(((long)Res2<0)&&(Res2<0xFFFF8000)){
        Res2=0x00008000;
        Res0=0x00000000;
    }
    if(((long)Res2>0)&&(Res2>0x00007FFF)){
        Res2=0x00007FFF;
```

```
            Res0=0xFFFFFFFF;
        };

        MACH=Res2;
        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.l  H'1234ABCD          ;
                                    .data.l  H'5678EF01          ;
TBLN                                .data.l  H'0123ABCD          ;
                                    .data.l  H'4567DEF0          ;
```

7.2.29 MAC.W Multiply and ACcumulate Word：算術演算命令

積和演算

書式	動作概略	命令コード	実行ステート	T ビット
MAC.W @Rm+,@Rn+ MAC @Rm+,@Rn+	**符号付きで** (Rn)×(Rm)+MAC→MAC	0100nnnnmmmm1111	3/(2)	—

(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。オーバフローが発生すると、MACH レジスタの LSB を 1 にセットします。結果が負の方向にオーバフローしたときは H'80000000（最小値）を、正の方向にオーバフローしたときは H'7FFFFFFF（最大値）を、MACL レジスタに格納します。

(2) 動作内容

```
MACW(long m, long n)    /* MAC.W @Rm+,@Rn+ */
{
    long tempm,tempn,dest,src,ans;
    unsigned long templ;
    tempn=(long)Read_Word(R[n]);
    R[n]+=2;
    tempm=(long)Read_Word(R[m]);
    R[m]+=2;
    templ=MACL;
    tempm=((long)(short)tempn*(long)(short)tempm);
    if ((long)MACL>=0) dest=0;
    else dest=1;
    if ((long)tempm>=0) {
        src=0;
        tempn=0;
    }
    else {
        src=1;
```

```
            tempn=0xFFFFFFFF;
        }
        src+=dest;
        MACL+=tempm;
        if ((long)MACL>=0) ans=0;
        else ans=1;
        ans+=dest;
        if (S==1) {
            if (ans==1) {
                if (src==0) MACL=0x7FFFFFFF;
                if (src==2) MACL=0x80000000;
            }
        }
        else {
            MACH+=tempn;
            if (templ>MACL) MACH+=1;
        }
        PC+=2;
    }
```

(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.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             ;
```

7.2.30 MOV MOVe data：データ転送命令

データ転送

書式		動作概略	命令コード	実行ステート	Tビット
MOV	Rm,Rn	Rm→Rn	0110nnnnmmmm0011	1	—
MOV.B	Rm,@Rn	Rm→ (Rn)	0010nnnnmmmm0000	1	—
MOV.W	Rm,@Rn	Rm→ (Rn)	0010nnnnmmmm0001	1	—
MOV.L	Rm,@Rn	Rm→ (Rn)	0010nnnnmmmm0010	1	—
MOV.B	@Rm,Rn	(Rm) →**符号拡張**→Rn	0110nnnnmmmm0000	1	—
MOV.W	@Rm,Rn	(Rm) →**符号拡張**→Rn	0110nnnnmmmm0001	1	—
MOV.L	@Rm,Rn	(Rm) →Rn	0110nnnnmmmm0010	1	—
MOV.B	Rm,@-Rn	Rn-1→Rn, Rm→ (Rn)	0010nnnnmmmm0100	1	—
MOV.W	Rm,@-Rn	Rn-2→Rn, Rm→ (Rn)	0010nnnnmmmm0101	1	—
MOV.L	Rm,@-Rn	Rn-4→Rn, Rm→ (Rn)	0010nnnnmmmm0110	1	—
MOV.B	@Rm+,Rn	(Rm) →**符号拡張**→Rn, Rm+1→Rm	0110nnnnmmmm0100	1	—
MOV.W	@Rm+,Rn	(Rm) →**符号拡張**→Rn, Rm+2→Rm	0110nnnnmmmm0101	1	—
MOV.L	@Rm+,Rn	(Rm) →Rn, Rm+4→Rm	0110nnnnmmmm0110	1	—
MOV.B	Rm,@(R0,Rn)	Rm→ (R0+Rn)	0000nnnnmmmm0100	1	—
MOV.W	Rm,@(R0,Rn)	Rm→ (R0+Rn)	0000nnnnmmmm0101	1	—
MOV.L	Rm,@(R0,Rn)	Rm→ (R0+Rn)	0000nnnnmmmm0110	1	—
MOV.B	@(R0,Rm),Rn	(R0+Rm) →**符号拡張**→Rn	0000nnnnmmmm1100	1	—
MOV.W	@(R0,Rm),Rn	(R0+Rm) →**符号拡張**→Rn	0000nnnnmmmm1101	1	—
MOV.L	@(R0,Rm),Rn	(R0+Rm) →Rn	0000nnnnmmmm1110	1	—

(1) 説明

ソースオペランドをデスティネーションへ転送します。オペランドがメモリのときは転送するデータサイズをバイト／ワード／ロングワードの範囲で指定できます。ソースオペランドがメモリのときは、ロードされたデータをロングワードに符号拡張後レジスタに格納します。

(2) 動作内容

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

MOVBS(long m, long n)    /* MOV.B Rm,@Rn */
{
```

```
    Write_Byte(R[n],R[m]);
    PC+=2;
}

MOVWS(long m, long n)    /* MOV.W Rm,@Rn */
{
    Write_Word(R[n],R[m]);
    PC+=2;
}

MOVLS(long m, long n)    /* MOV.L Rm,@Rn */
{
    Write_Long(R[n],R[m]);
    PC+=2;
}

MOVBL(long m, long n)    /* MOV.B @Rm,Rn */
{
    R[n]=(long)Read_Byte(R[m]);
    if ((R[n]&0x80)==0) R[n]&=0x000000FF;
    else R[n]|=0xFFFFFF00;
    PC+=2;
}

MOVWL(long m, long n)    /* MOV.W @Rm,Rn */
{
    R[n]=(long)Read_Word(R[m]);
    if ((R[n]&0x8000)==0) R[n]&=0x0000FFFF;
    else R[n]|=0xFFFF0000;
    PC+=2;
}

MOVLL(long m, long n)    /* MOV.L @Rm,Rn */
{
    R[n]=Read_Long(R[m]);
    PC+=2;
}
```

```
MOVBM(long m, long n)    /* MOV.B Rm,@-Rn */
{
    Write_Byte(R[n]-1,R[m]);
    R[n]-=1;
    PC+=2;
}

MOVWM(long m, long n)    /* MOV.W Rm,@-Rn */
{
    Write_Word(R[n]-2,R[m]);
    R[n]-=2;
    PC+=2;
}

MOVLM(long m, long n)    /* MOV.L Rm,@-Rn */
{
    Write_Long(R[n]-4,R[m]);
    R[n]-=4;
    PC+=2;
}

MOVBP(long m, long n)    /* MOV.B @Rm+,Rn */
{
    R[n]=(long)Read_Byte(R[m]);
    if ((R[n]&0x80)==0) R[n]&=0x000000FF;
    else R[n]|=0xFFFFFF00;
    if (n!=m) R[m]+=1;
    PC+=2;
}

MOVWP(long m, long n)    /* MOV.W @Rm+,Rn */
{
    R[n]=(long)Read_Word(R[m]);
    if ((R[n]&0x8000)==0) R[n]&=0x0000FFFF;
    else R[n]|=0xFFFF0000;
    if (n!=m) R[m]+=2;
```

```
    PC+=2;
}

MOVLP(long m, long n)    /* MOV.L @Rm+,Rn */
{
    R[n]=Read_Long(R[m]);
    if (n!=m) R[m]+=4;
    PC+=2;
}

MOVBS0(long m, long n)    /* MOV.B Rm,@(R0,Rn) */
{
    Write_Byte(R[n]+R[0],R[m]);
    PC+=2;
}

MOVWS0(long m, long n)    /* MOV.W Rm,@(R0,Rn) */
{
    Write_Word(R[n]+R[0],R[m]);
    PC+=2;
}

MOVLS0(long m, long n)    /* MOV.L Rm,@(R0,Rn) */
{
    Write_Long(R[n]+R[0],R[m]);
    PC+=2;
}

MOVBL0(long m, long n)    /* MOV.B @(R0,Rm),Rn */
{
    R[n]=(long)Read_Byte(R[m]+R[0]);
    if ((R[n]&0x80)==0) R[n]&=0x000000FF;
    else R[n]|=0xFFFFFF00;
    PC+=2;
}
```

```
MOVWL0(long m, long n)    /* MOV.W @(R0,Rm),Rn */
{
    R[n]=(long)Read_Word(R[m]+R[0]);
    if ((R[n]&0x8000)==0) R[n]&=0x0000FFFF;
    else R[n]|=0xFFFF0000;
    PC+=2;
}

MOVLL0(long m, long n)    /* MOV.L @(R0,Rm),Rn */
{
    R[n]=Read_Long(R[m]+R[0]);
    PC+=2;
}
```

(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
```

7.2.31 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	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	—

(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]=(0xFFFFFF00 | (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;
}

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

    disp=(0x000000FF & (long)d);
    R[n]=Read_Long((PC&0xFFFFFFFC)+(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  @(1,PC),R3        ;R3=H'12345678
100E  IMM.data.w  H'9ABC          ;
1010      .data.w  H'1234         ;
1012  NEXT                       JMP    @R3   ;BRA の分岐先
1014     CMP/EQ #0,R0             ;←MOV.L 命令でアドレス計算に用いる PC の位置
         .align  4                ;
1018     .data.l H'12345678       ;
```

7.2.32 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のとき 8 ビットディスプレースメントはゼロ拡張するだけですので、＋255 バイトまでの範囲が指定できます。ワードサイズのとき 8 ビットディスプレースメントはゼロ拡張後 2 倍しますので、＋510 バイトまでの範囲が指定できます。ロングワードサイズのとき 8 ビットディスプレースメントはゼロ拡張後 4 倍しますので、＋1020 バイトまでの範囲が指定できます。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変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;
    else R[0]|=0xFFFFFF00;
    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;
}

MOVLLG(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
```

7.2.33 MOV MOVe structure data：データ転送命令

比較構造体データの転送

書式		動作概略	命令コード	実行ステート	Tビット
MOV.B	R0,@(disp,Rn)	R0→ (disp+Rn)	10000000nnnndddd	1	—
MOV.W	R0,@(disp,Rn)	R0→ (disp×2+Rn)	10000001nnnndddd	1	—
MOV.L	Rm,@(disp,Rn)	Rm→ (disp×4+Rn)	0001nnnnmmmmdddd	1	—
MOV.B	@(disp,Rm),R0	(disp+Rm) →**符号拡張**→R0	10000100mmmmdddd	1	—
MOV.W	@(disp,Rm),R0	(disp×2+Rm) →**符号拡張**→R0	10000101mmmmdddd	1	—
MOV.L	@(disp,Rm),Rn	(disp×4+Rm) →Rn	0101nnnnmmmmdddd	1	—

(1) 説明

ソースオペランドをデスティネーションへ転送します。構造体、スタック内のデータアクセスに最適です。データサイズをバイト、ワード、またはロングワードの範囲で指定できますが、バイトまたはワードのときはレジスタが R0 固定になります。

データがバイトサイズのとき 4 ビットディスプレースメントはゼロ拡張するだけですので、＋15 バイトまでの範囲が指定できます。ワードサイズのとき 4 ビットディスプレースメントはゼロ拡張後 2 倍しますので、＋30 バイトまでの範囲が指定できます。ロングワードサイズのとき 4 ビットディスプレースメントはゼロ拡張後 4 倍しますので、＋60 バイトまでの範囲が指定できます。メモリオペランドに届かないときは前述の@(R0,Rn)モードを使う必要があります。

ソースオペランドがメモリのときは、ロードされたデータをロングワードに符号拡張後レジスタへ格納します。

(2) 注意

バイト／ワードデータをロードするときデスティネーションレジスタが 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変えることによって最適化が可能です。

```
MOV.B   @(2,R1),R0              MOV.B   @(2,R1),R0
AND     #80,R0          ->      ADD     #20,R1
ADD     #20,R1          ->      AND     #80,R0
```

(3) 動作内容

```
MOVBS4(long d, long n)    /* MOV.B R0,@(disp,Rn) */
{
    long disp;

    disp=(0x0000000F & (long)d);
    Write_Byte(R[n]+disp,R[0]);
    PC+=2;
}

MOVWS4(long d, long n)    /* MOV.W R0,@(disp,Rn) */
{
    long disp;

    disp=(0x0000000F & (long)d);
    Write_Word(R[n]+(disp<<1),R[0]);
    PC+=2;
}

MOVLS4(long m, long d, long n)    /* MOV.L Rm,@(disp,Rn) */
{
    long disp;

    disp=(0x0000000F & (long)d);
    Write_Long(R[n]+(disp<<2),R[m]);
    PC+=2;
}

MOVBL4(long m, long d)    /* MOV.B @(disp,Rm),R0 */
{
    long disp;

    disp=(0x0000000F & (long)d);
    R[0]=Read_Byte(R[m]+disp);
    if ((R[0]&0x80)==0) R[0]&=0x000000FF;
    else R[0]|=0xFFFFFF00;
    PC+=2;
```

```
}

MOVWL4(long m, long d)    /* MOV.W @(disp,Rm),R0 */
{
    long disp;

    disp=(0x0000000F & (long)d);
    R[0]=Read_Word(R[m]+(disp<<1));
    if ((R[0]&0x8000)==0) R[0]&=0x0000FFFF;
    else R[0]|=0xFFFF0000;
    PC+=2;
}

MOVLL4(long m, long d, long n)    /* MOV.L @(disp,Rm),Rn */
{
    long disp;

    disp=(0x0000000F & (long)d);
    R[n]=Read_Long(R[m]+(disp<<2));
    PC+=2;
}
```

(4) 使用例

```
MOV.L  @(2,R0),R1          ;実行前 @(R0+8)=H'12345670
                           ;実行後 R1=@H'12345670
MOV.L  R0,@(H'F,R1)        ;実行前 R0=H'FFFF7F80
                           ;実行後 @(R1+60)=H'FFFF7F80
```

7.2.34 MOVA MOVe effective Address：データ転送命令

実効アドレスの転送

書式	動作概略	命令コード	実行ステート	T ビット
MOVA @(disp,PC),R0	disp×4+PC→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&0xFFFFFFFC)+(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                    ;
```

7.2.35 MOVT MOVe Tbit：データ転送命令

T ビットの転送

書式	動作概略	命令コード	実行ステート	T ビット
MOVT Rn	T→Rn	0000nnnn00101001	1	—

(1) 説明

T ビットを汎用レジスタ Rn に格納します。T＝1 のとき Rn＝1、T＝0 のとき Rn＝0 になります。

(2) 動作内容

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

(3) 使用例

```
XOR       R2,R2             ;R2=0
CMP/PZ    R2                ;T=1
MOVT      R0                ;R0=1
CLRT                        ;T=0
MOVT      R1                ;R1=0
```

7.2.36 MUL.L MULtiply Long：算術演算命令

倍精度乗算

書式	動作概略	命令コード	実行ステート	T ビット
MUL.L Rm,Rn	Rn×Rm→MACL	0000nnnnmmmm0111	2〜4	—

(1) 説明

汎用レジスタ Rn の内容と Rm を 32 ビットで乗算し、結果の下位側 32 ビットを MACL レジスタに格納します。MACH の内容は変化しません。

(2) 動作内容

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

(3) 使用例

```
MUL.L  R0,R1              ;実行前 R0=H'FFFFFFFE,R1=H'00005555
                          ;実行後 MACL=H'FFFF5556
STS   MACL,R0             ;演算結果を得る
```

7.2.37 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×Rm→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
```

7.2.38 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×Rm→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'FFFFAAAA
                           ;実行後 MACL=H'00015554
STS   MACL,R0              ;演算結果を得る
```

7.2.39 NEG NEGate：算術演算命令

符号反転

書式	動作概略	命令コード	実行ステート	T ビット
NEG Rm,Rn	0-Rm→Rn	0110nnnnmmmm1011	1	—

(1) 説明

汎用レジスタ Rm の内容の 2 の補数をとり、結果を Rn に格納します。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'FFFFFFFF
```

7.2.40 NEGC NEGate with Carry：算術演算命令

ボロー付き符号反転

書式	動作概略	命令コード	実行ステート	T ビット
NEGC Rm,Rn	0-Rm-T→Rn, ボロー→T	0110nnnnmmmm1010	1	ボロー

(1) 説明

0 から汎用レジスタ Rm の内容と T ビットを減算し、結果を Rn に格納します。演算の結果によってボローを T ビットに反映します。32 ビットを超える値の符号反転を行うとき使用します。

(2) 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'FFFFFFFF,T=1
NEGC  R0,R0             ;実行前 R0=H'00000000,T=1
                        ;実行後 R0=H'FFFFFFFF,T=1
```

7.2.41 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。
```

7.2.42 NOT NOT-logical complement：論理演算命令

ビット反転

書式	動作概略	命令コード	実行ステート	T ビット
NOT Rm,Rn	~Rm→Rn	0110nnnnmmmm0111	1	—

(1) 説明

汎用レジスタ Rm の内容の 1 の補数をとり、結果を Rn に格納します。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
```

7.2.43 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
```

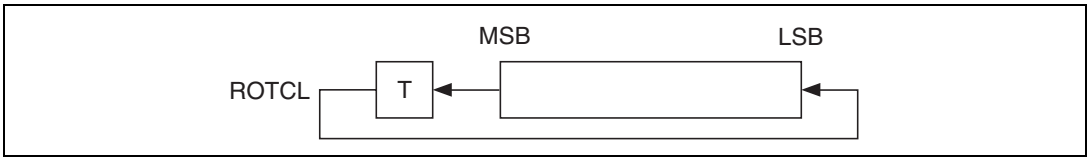
7.2.44 ROTCL ROTate with Carry Left：シフト命令

T ビット付き 1 ビット左回転

書式	動作概略	命令コード	実行ステート	T ビット
ROTCL Rn	T←Rn←T	0100nnnn00100100	1	MSB

(1) 説明

汎用レジスタ Rn の内容を左方向に T ビットを含めて 1 ビットローテート（回転）し、結果を Rn に格納します。ローテートしてオペランドの外に出てしまったビットは、T ビットへ転送します。



(2) 動作内容

```
ROTCL(long n)    /* ROTCL Rn */
{
    long temp;

    if ((R[n]&0x80000000)==0) temp=0;
    else temp=1;
    R[n]<<=1;
    if (T==1) R[n]|=0x00000001;
    else R[n]&=0xFFFFFFFE;
    if (temp==1) T=1;
    else T=0;
    PC+=2;
}
```

(3) 使用例

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

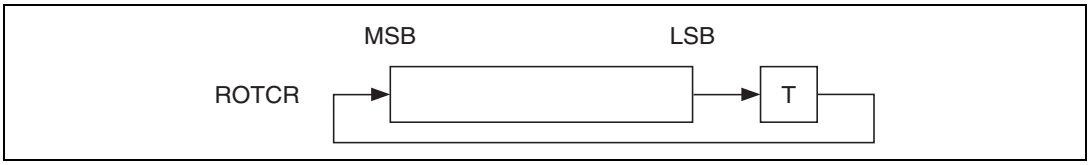
7.2.45 ROTCR ROTate with Carry Right：シフト命令

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
```

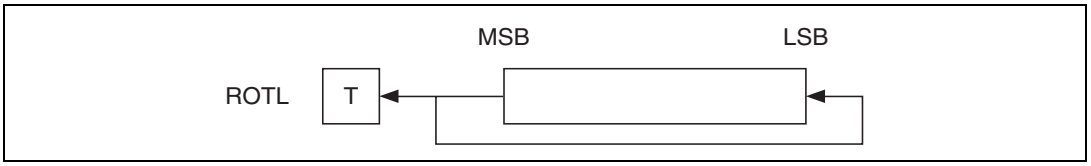
7.2.46 ROTL ROTate Left：シフト命令

1 ビット左回転

書式	動作概略	命令コード	実行ステート	T ビット
ROTL Rn	T←Rn←MSB	0100nnnn00000100	1	MSB

(1) 説明

汎用レジスタ Rn の内容を左方向に 1 ビットローテート（回転）し、結果を Rn に格納します。ローテートしてオペランドの外に出てしまったビットは、T ビットへ転送します。



(2) 動作内容

```
ROTL(long n)    /* ROTL Rn */
{
    if ((R[n]&0x80000000)==0) T=0;
    else T=1;
    R[n]<<=1;
    if (T==1) R[n]|=0x00000001;
    else R[n]&=0xFFFFFFFE;
    PC+=2;
}
```

(3) 使用例

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

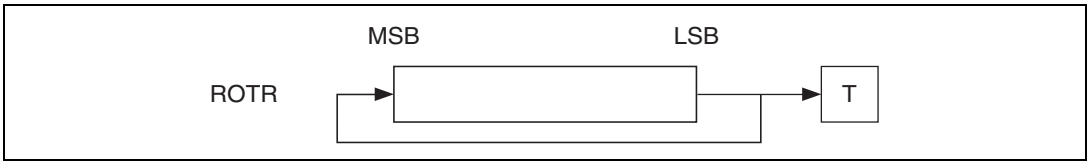
7.2.47 ROTR ROTate Right：シフト命令

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
```

7.2.48 RTE ReTurn from Exception：システム制御命令

例外処理からの復帰 **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
RTE	スタック領域→PC/SR	0000000000101011	4	—

(1) 説明

割り込みルーチンから復帰します。すなわち、PC と SR をスタックから復帰し、復帰した PC の示すアドレスから処理を続行します。

(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、アドレスエラーと割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

(3) 動作内容

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

    temp=PC;
    PC=Read_Long(R[15])+4;
    R[15]+=4;
    SR=Read_Long(R[15])&0x000003F3;
    R[15]+=4;
    Delay_Slot(temp+2);
}
```

(4) 使用例

```
RTE                 ;元のルーチンへ復帰します。
ADD    #8,R14       ;分岐に先立ち実行します。
```

【注】 遅延分岐においては分岐という動作そのものは、スロット命令の実行後に発生しますが、命令の実行（レジスタの更新など）は、あくまでも遅延分岐命令→遅延スロット命令の順に行われます。たとえば遅延スロットで分岐先アドレスが格納されたレジスタを変更しても、変更前のレジスタ内容が分岐先アドレスとなります。

7.2.49 RTS ReTurn from SubRoutine：分岐命令

サブルーチンプロシージャからの復帰 **遅延分岐命令**

書式	動作概略	命令コード	実行ステート	T ビット
RTS	PR→PC	0000000000001011	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 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、アドレスエラーと割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

(3) 動作内容

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

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

(4) 使用例

```
        MOV.L       TABLE,R3    ;R3=TRGET のアドレス
        JSR         @R3         ;TRGET へ分岐します。
        NOP                     ;JSR に先立ち実行します。
        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。

7.2.50 SETT SET Tbit：システム制御命令

T ビットのセット

書式	動作概略	命令コード	実行ステート	T ビット
SETT	1→T	0000000000011000	1	1

(1) 説明

T ビットをセットします。

(2) 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
```

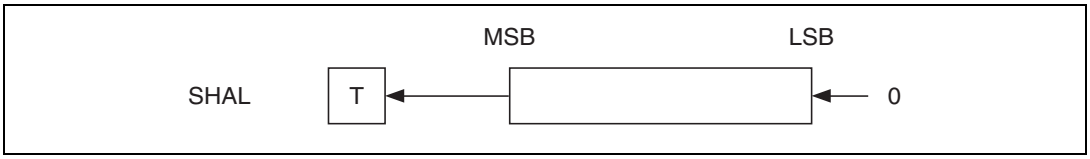
7.2.51 SHAL SHift Arithmetic Left：シフト命令

算術的 1 ビット左シフト

書式	動作概略	命令コード	実行ステート	T ビット
SHAL Rn	T←Rn←0	0100nnnn00100000	1	MSB

(1) 説明

汎用レジスタ Rn の内容を左方向に算術的に 1 ビットシフトし、結果を Rn に格納します。シフトしてオペランドの外に出てしまったビットは、T ビットへ転送します。



(2) 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
```

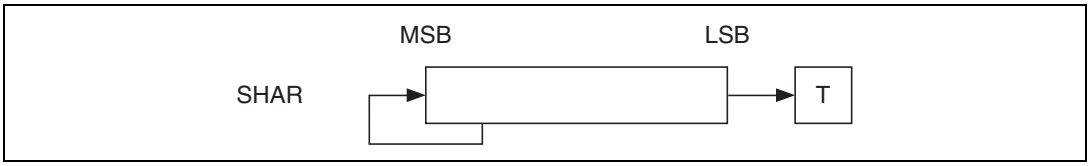
7.2.52 SHAR SHift Arithmetic Left/Right：シフト命令

算術的 1 ビット右シフト

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
```

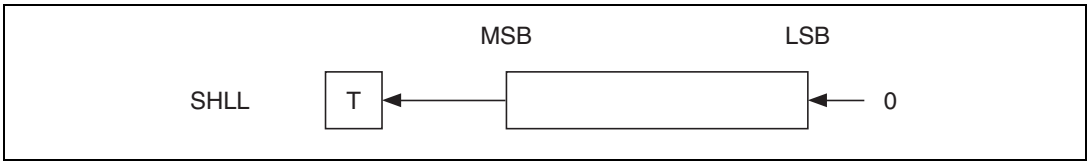
7.2.53 SHLL SHift Logical Left：シフト命令

論理的 1 ビット左シフト

書式	動作概略	命令コード	実行ステート	T ビット
SHLL Rn	T←Rn←0	0100nnnn00000000	1	MSB

(1) 説明

汎用レジスタ Rn の内容を左方向に論理的に 1 ビットシフトし、結果を Rn に格納します。シフトしてオペランドの外に出てしまったビットは、T ビットへ転送します。



(2) 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
```

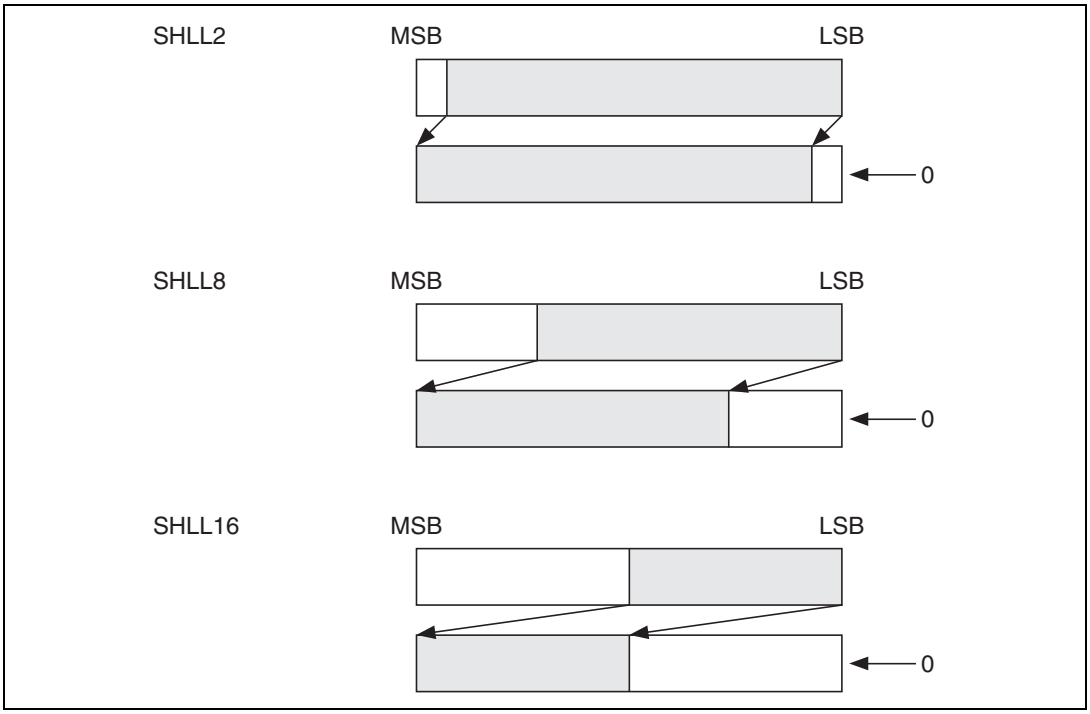
7.2.54 SHLLn n bits SHift Logical Left：シフト命令

論理的 n ビット　　左シフト

書式		動作概略	命令コード	実行ステート	T ビット
SHLL2	Rn	Rn<<2→Rn	0100nnnn00001000	1	—
SHLL8	Rn	Rn<<8→Rn	0100nnnn00011000	1	—
SHLL16	Rn	Rn<<16→Rn	0100nnnn00101000	1	—

(1) 説明

汎用レジスタ Rn の内容を左方向に論理的に 2/8/16 ビットシフトし、結果を Rn に格納します。シフトしてオペランドの外に出てしまったビットは捨てます。



(2) 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
```

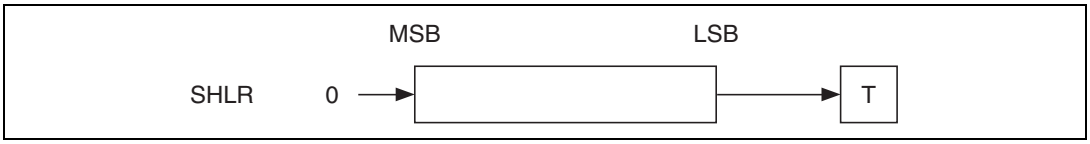
7.2.55 SHLR SHift Logical Right：シフト命令

論理的 1 ビット右シフト

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
```

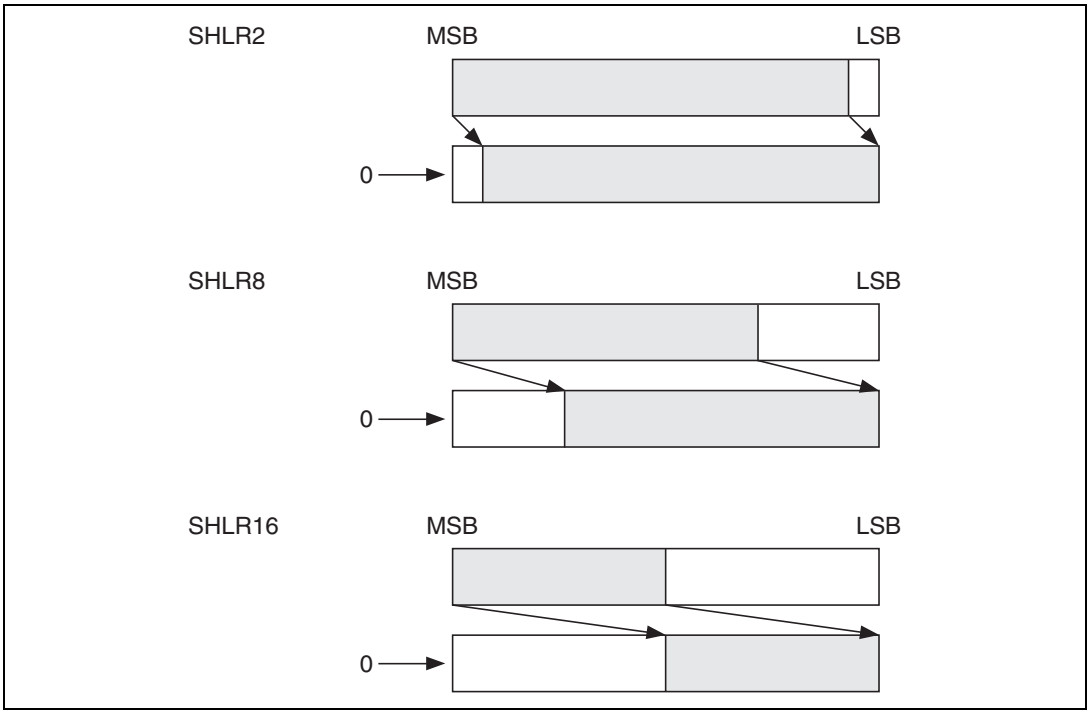
7.2.56 SHLRn n bits SHift Logical Right：シフト命令

論理的 n ビット右シフト

書式		動作概略	命令コード	実行ステート	T ビット
SHLR2	Rn	Rn>>2→Rn	0100nnnn00001001	1	—
SHLR8	Rn	Rn>>8→Rn	0100nnnn00011001	1	—
SHLR16	Rn	Rn>>16→Rn	0100nnnn00101001	1	—

(1) **説明**

汎用レジスタ Rn の内容を右方向に論理的に 2/8/16 ビットシフトし、結果を Rn に格納します。シフトしてオペランドの外に出てしまったビットは捨てます。



(2) 動作内容

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

SHLR8(long n)    /* SHLR8 Rn */
{
    R[n]>>=8;
    R[n]&=0x00FFFFFF;
    PC+=2;
}

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

(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
```

7.2.57 SLEEP SLEEP：システム制御命令

低消費電力状態への遷移

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	3	—

(1) 説明

CPU を低消費電力状態にします。

低消費電力状態では、CPU の内部状態を保持し、直後の命令の実行を停止し、割り込み要求の発生を待ちます。要求が発生すると、低消費電力状態から抜けます。

(2) 注意

実行ステートの 3 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               ;低消費電力状態への遷移
```

7.2.58 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 **割り込み禁止命令**

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.L SR,@-Rn	Rn-4→Rn, SR→ (Rn)	0100nnnn00000011	2	—
STC.L GBR,@-Rn	Rn-4→Rn, GBR→ (Rn)	0100nnnn00010011	2	—
STC.L VBR,@-Rn	Rn-4→Rn, VBR→ (Rn)	0100nnnn00100011	2	—

(1) 説明

コントロールレジスタ SR、GBR、または VBR をデスティネーションに格納します。

(2) 注意

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。（アドレスエラーは受け付けます。）

(3) 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;
}

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;
}
```

(4) 使用例

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

7.2.59 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 **割り込み禁止命令**

書式	動作概略	命令コード	実行ステート	T ビット
STS MACH,Rn	MACH→Rn	0000nnnn00001010	1	—
STS MACL,Rn	MACL→Rn	0000nnnn00011010	1	—
STS PR,Rn	PR→Rn	0000nnnn00101010	1	—
STS.L MACH,@-Rn	Rn-4→Rn, MACH→ (Rn)	0100nnnn00000010	1	—
STS.L MACL,@-Rn	Rn-4→Rn, MACL→ (Rn)	0100nnnn00010010	1	—
STS.L PR,@-Rn	Rn-4→Rn, PR→ (Rn)	0100nnnn00100010	1	—

(1) 説明

システムレジスタ MACH、MACL、または PR をデスティネーションに格納します。

(2) 注意

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（アドレスエラーは受け付けます）。

(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;
}

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

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

STSMMACL(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;
}
```

(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
```

7.2.60 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
```

7.2.61 SUBC SUBtract with Carry：算術演算命令

ボロー付き 2 進減算

書式	動作概略	命令コード	実行ステート	T ビット
SUBC Rm,Rn	Rn-Rm-T→Rn, ボロー→T	0011nnnnmmmm1010	1	ボロー

(1) 説明

汎用レジスタ Rn の内容から Rm と T ビットを減算し、結果を Rn に格納します。演算の結果によってボローを T ビットに反映します。32 ビットを超える減算を行うとき使用します。

(2) 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
```

7.2.62 SUBV SUBtract with (Vflag)underflow check ：算術演算命令

アンダフロー付き 2 進減算

書式	動作概略	命令コード	実行ステート	T ビット
SUBV Rm,Rn	Rn-Rm→Rn, アンダフロー→T	0011nnnnmmmm1011	1	アンダフロー

(1) 説明

汎用レジスタ Rn の内容から Rm を減算し、結果を Rn に格納します。アンダフローが発生すると、T ビットをセットします。

(2) 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'7FFFFFFE
                    ;実行後 R3=H'80000000,T=1
```

7.2.63 SWAP SWAP register halves：データ転送命令

上位と下位の交換

書式	動作概略	命令コード	実行ステート	T ビット
SWAP.B Rm,Rn	Rm→**下位 2 バイトの上下バイト交換**→Rn	0110nnnnmmmm1000	1	—
SWAP.W Rm,Rn	Rm→**上下ワード交換**→Rn	0110nnnnmmmm1001	1	—

(1) 説明

汎用レジスタ Rm の内容の上位と下位を交換して、結果を Rn に格納します。

バイト指定のとき、Rm のビット 0 からビット 7 の 8 ビットと、ビット 8 からビット 15 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 0 からビット 15 の 16 ビットと、ビット 16 からビット 31 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]&0xffff0000;
    temp1=(R[m]&0x000000ff)<<8;
    R[n]=(R[m]>>8)&0x000000ff;
    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
```

7.2.64 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	4	テスト結果

(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。

(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。
```

7.2.65 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	PC/SR→**スタック領域**, (imm×4+VBR) →PC	11000011iiiiiiii	8	—

(1) 説明

トラップ例外処理を開始します。すなわち PC と SR をスタックに退避し、指定ベクタの内容で表されるアドレスへ分岐します。ベクタは 8 ビットイミディエートデータをゼロ拡張後 4 倍したメモリアドレスそのものです。PC は次命令の先頭アドレスです。

RTE と組み合せて、システムコールに使います。

(2) 動作内容

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

    imm=(0x000000FF & i);
    R[15]-=4;
    Write_Long(R[15],SR);
    R[15]-=4;
    Write_Long(R[15],PC-2);
    PC=Read_Long(VBR+(imm<<2))+4;
}
```

(3) 使用例

アドレス

```
VBR+H'80    .data.l    10000000    ;
            .......
            TRAPA      #H'20       ;VBR+H'80 番地の内容のアドレスに分岐します。
            TST        #0,R0       ; ←トラップルーチンからの戻り先
            .......                (スタックした PC の内容）です。
            .......
10000000    XOR        R0,R0       ; ← トラップルーチンの入り口
10000002    RTE                    ;上記 TST に戻ります。
10000004    NOP                    ;RTE に先立ち実行します。
```

7.2.66 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	0010nnnnmmmm1000	1	**テスト結果**
TST #imm,R0	R0 & imm, **結果が 0 のとき** 1→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	11001100iiiiiiii	3	**テスト結果**

(1) 説明

汎用レジスタ Rn の内容と Rm の論理積をとり、結果がゼロのとき T ビットをセットします。結果がゼロでないとき T ビットをクリアします。Rn の内容は変更しません。

汎用レジスタ R0 とゼロ拡張した 8 ビットのイミディエイトデータとの論理積、もしくはインデックス付き GBR 間接アドレッシングモードで 8 ビットのメモリと 8 ビットのイミディエイトデータとの論理積が可能です。R0、もしくはメモリの内容は変更しません。

(2) 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
```

7.2.67 XOR eXclusive OR logical：論理演算命令

排他的論理和演算

書式	動作概略	命令コード	実行ステート	T ビット
XOR Rm,Rn	Rn ^ Rm → Rn	0010nnnnmmmm1010	1	—
XOR #imm,R0	R0 ^ imm → R0	11001010iiiiiiii	1	—
XOR.B #imm,@(R0,GBR)	(R0+GBR) ^ imm→ (R0+GBR)	11001110iiiiiiii	3	—

(1) 説明

汎用レジスタ Rn の内容と Rm の排他的論理和をとり、結果を Rn に格納します。

汎用レジスタ R0 とゼロ拡張した 8 ビットのイミディエイトデータとの排他的論理和、もしくはインデックス付き GBR 間接アドレッシングモードで 8 ビットのメモリと 8 ビットのイミディエイトデータとの排他的論理和が可能です。

(2) 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
```

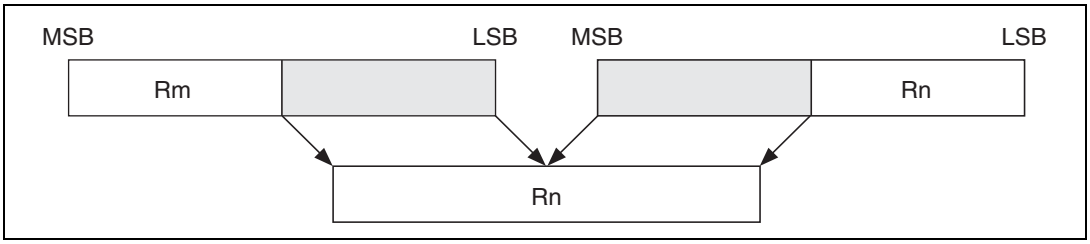
7.2.68 XTRCT eXTRaCT：データ転送命令

連結レジスタの中央切り出し

書式	動作概略	命令コード	実行ステート	T ビット
XTRCT Rm,Rn	Rm:Rn の中央 32 ビット→Rn	0010nnnnmmmm1101	1	—

(1) **説明**

汎用レジスタ Rm と Rn とを連結した 64 ビットの内容から中央の 32 ビットを切り出し、結果を Rn に格納します。



(2) **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
```

7.3 浮動小数点命令と FPU に関する CPU 命令

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); }
```

```
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;
}
```

7.3.1 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	実行ステート	T ビット
FABS FRn	|FRn|→FRn	1111nnnn01011101	1	ー

(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);                berak;
 }
 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*/
```

7.3.2 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	実行ステート	T ビット
FADD FRm,FRn	FRn+FRm→FRn	1111nnnnmmmm0000	1	ー

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

FADD 特殊ケース

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

(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*/
```

7.3.3 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	実行ステート	T ビット
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

(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;
                            default     :                   break;
        }                                                   break;
        PZERO               :
        NZERO               :case(data_type_of(FRn))        {
                            PZERO       :
                            NZERO       :return(EQ);        break;
                            PINF        :return(GT);        break;
                            NINF        :return(NOTGT);     break;
                            default     :                   break;
        }                                                   break;
        PINF                :case(data_type_of(FRn))        {
                            PINF        :return(EQ)         break;
                            default     :return(NOTGT);     break;
        }                                                   break;
        NINF                :case(data_type_of(FRn))        {
                            NINF        :return(EQ);        break;
                            default     :return(GT);        break;
        }                                                   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			GT	!GT		
+0		EQ					
-0							
+INF	!GT			EQ			
-INF	GT				EQ		
qNaN						UO	
sNaN							Invalid

【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 通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-2E 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 命令を組み合わせて使用することで、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）
    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
```

7.3.4 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	実行ステート	T ビット
FDIV FRm, FRn	FRn/FRm→FRn	1111nnnnmmmm0011	13	ー

(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;
      }                                                               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;
      }                                                               break;
      PINF   :
      NINF   :
```

```
        case(data_type_of(FRn)){
                PINF        :
                NINF        :  invalid(FRn);                                    break;
                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
```

7.3.5 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	実行ステート	T ビット
FLDI0 FRn	H'00000000→FRn	1111nnnn10001101	1	ー

(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
```

7.3.6 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	実行ステート	T ビット
FLDI1 FRn	H'3F800000→FRn	1111nnnn10011101	1	ー

(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*/
```

7.3.7 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	実行ステート	T ビット
FLDS FRm,FPUL	FRm→FPUL	1111nnnn00011101	1	ー

(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
```

7.3.8 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	実行ステート	T ビット
FLOAT FPUL,FRn	(float)FPUL→FRn	1111nnnn00101101	1	ー

(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
    FLDI0       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*/
```

7.3.9 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	実行ステート	T ビット
FMAC FR0,FRm,FRn	FR0×FRm+FRn→FRn	1111nnnnmmmm1110	1	ー

(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   */
}
```

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				+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						+INF		
	0					Invalid			
	+INF			Invalid		+INF			
	-INF	Invalid	+INF				+INF		
-INF	+NORM	-INF					-INF		
	-NORM								
	0								
	+INF	Invalid		Invalid			-INF		
	-INF	-INF				-INF	Invalid		
qNaN	0					Invalid			
	INF			Invalid					
	!sNaN								
!NaN	qNaN						qNaN		
All types	sNaN								
sNaN	All types								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**

```
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*/
```

7.3.10 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	実行ステート	Tビット
1. FMOV FRm, FRn	FRm→FRn	1111nnnnmmmm1100	1	−
2. FMOV.S @Rm, FRn	(Rm) →FRn	1111nnnnmmmm1000	1	−
3. FMOV.S FRm, @Rn	FRm→ (Rn)	1111nnnnmmmm1010	1	−
4. FMOV.S @Rm+,FRn	(Rm) →FRn, Rm+=4	1111nnnnmmmm1001	1	−
5. FMOV.S FRm,@-Rn	Rn-=4, FRm→ (Rn)	1111nnnnmmmm1011	1	−
6. FMOV.S @(R0,Rm), FRn	(R0+Rm) →FRn	1111nnnnmmmm0110	1	−
7. FMOV.S FRm,@(R0,Rn)	FRm→ (R0+Rn)	1111nnnnmmmm0111	1	−

(1) **説明**

1. 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。
2. 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。
3. 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。
4. 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。
5. 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。
6. 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。
7. 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

(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
```

7.3.11 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	実行ステート	T ビット
FMUL FRm,FRn	FRn×FRm→FRn	1111nnnnmmmm0010	1	ー

(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;
        }                                                                 break;
        PZERO         :
        NZERO         :
        case(data_type_of(FRn))       {
                    PINF      :
                    NINF      :  invalid(FRn);                            break;
                    default   :  zero(FRn,sign_of(FRm)^sign_of(FRn));     break;
        }                                                                 break;
        PINF          :
        NINF          :
        case(data_type_of(FRn))       {
                    PZERO     :
```

```
                    NZERO     :  invalid(FRn);                              break;
                    default   :inf  (FRn,sign_of(FRm)^sign_of(FRn));        break
          }                                                                 break;
    }
    pc += 2;
}
```

FMUL 特殊ケース

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

(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*/
```

7.3.12 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	実行ステート	T ビット
FNEG FRn	-FRn→FRn	1111nnnn01001101	1	ー

(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*/
```

7.3.13 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	実行ステート	T ビット
FSTS FPUL,FRn	FPUL→FRn	1111nnnn00001101	1	ー

(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
FLDI0     FR5                                   ;FR5=0
LDS       R2,FPUL           ;FSTS 命令実行後:   ;R2=H'00000002
FSTS      FPUL, R5                              ;FR5= H'00000002
```

7.3.14 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	実行ステート	T ビット
FSUB FRm, FRn	FRn-FRm→FRn	1111nnnnmmmm0001	1	ー

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

FSUB **特殊ケース**

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

(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*/
```

7.3.15 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	実行ステート	T ビット
FTRC FRm, FPUL	(long)FRm→FPUL	1111nnnn00111101	1	ー

(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 & 0x7FFFFF;
  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       */
```

```
    }
  }
  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 rarge	+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
```

7.3.16 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	実行ステート	T ビット
1. LDS Rm, FPUL	Rm→FPUL	0100nnnn01011010	1	−
2. LDS.L @Rm+,FPUL	(Rm) →FPUL, Rm+=4	0100nnnn01010110	1	−
3. LDS Rm,FPSCR	Rm→FPSCR	0100nnnn01101010	1	−
4. LDS.L @Rm+,FPSCR	(Rm) →FPSCR, Rm+=4	0100nnnn01100110	1	−

(1) **説明**

1. 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。
2. 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。
3. 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。
4. 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 */
{
        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
```

7.3.17 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	実行ステート	T ビット
1. STS	FPUL,Rn	FPUL→Rn	0000nnnn01011010	1	−
2. STS.L	FPUL,@-Rn	Rn -= 4, FPUL→@(Rn)	0100nnnn01010010	1	−
3. STS	FPSCR,Rn	FPSCR→Rn	0000nnnn01101010	1	−
4. STS.L	FPSCR,@-Rn	Rn -= 4, FPSCR→@(Rn)	0100nnnn01100010	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)
        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. パイプライン動作

各命令のパイプライン動作を説明します。これは、CPU の命令実行ステート数（システムクロックサイクル数）の算出をするための情報です。

8.1 パイプラインの基本構成

パイプラインは、次の 5 つのステージから構成されます。

●IF	：命令フェッチ	プログラムが格納されているメモリから命令を取り込みます。
●ID	：命令デコード	取り込んだ命令を解読します。
●EX	：命令実行	解読結果に従い、データ演算やアドレス計算を行います。
●MA	：メモリアクセス	メモリのデータアクセスを行います。 メモリアクセスを伴う命令で発生しますが一部例外があります。
●WB	：ライトバック	メモリアクセスした結果（データ）をレジスタに戻します。 メモリロードを伴う命令で発生しますが一部例外があります。

図 8.1 に示すように、命令の各ステージは、命令の実行とともに流れていき、パイプラインを構成します。ある瞬間を捉えれば、5 つの命令が同時に実行されることになります。基本的なパイプラインの流れを**図** 8.1 に示します。1 つのステージが実行される期間をスロットと呼び“ ←→ ”で表します。各命令は少なくとも 3 ステージ構造になっています。

IF、ID、EX の 3 ステージはすべての命令に存在しますが、命令によっては、MA、WB がない場合もあります。また、MA が 2 回あったり、乗算器をアクセスする（mm）があったり、パイプラインの流れ方も、命令の種類によって変化します。IF と MA との競合などが発生することもあり、競合が発生するとパイプラインの流れが変わります。

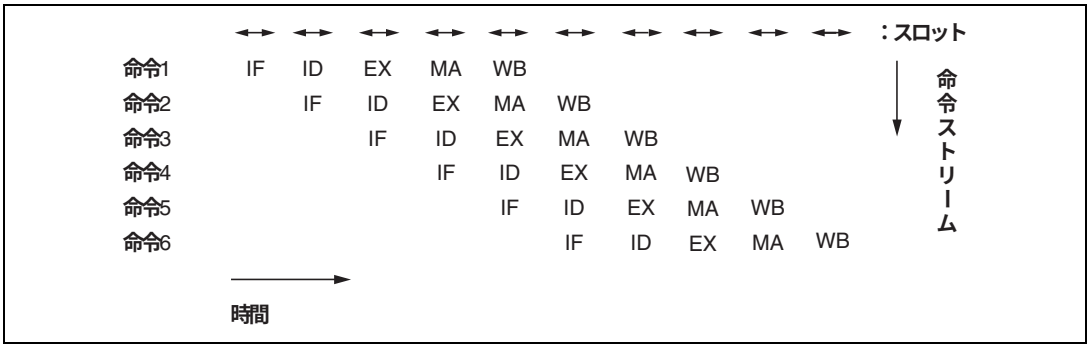


図 8.1 **パイプラインの基本構成**

(a) FPU パイプライン

FPU パイプラインの各ステージの長さは CPU パイプラインの各ステージの長さと同じです。両方のパイプラインは第一ステージの命令フェッチ（IF）を共有します。FPU パイプラインにはさらに 4 つのステージがあります。

●DF	：FPU デコード	取り込んだ命令を解読します。
●E1	：FPU 実行第 1 ステージ	浮動小数点演算を初期化します。
●E2	：FPU 実行第 2 ステージ	浮動小数点演算を完了します。
●SF	：FPU ストア	FPU レジスタに結果を書き込みます。

すべての命令は CPU パイプラインと FPU パイプラインの両方のパイプラインを通ります。そして命令によって、CPU パイプライン、または両方のパイプラインによる動作を行います。

浮動小数点命令および FPU に関する CPU 命令では FPU パイプラインと CPU パイプラインが並行して同時に動作します。

それ以外の純粋な CPU 命令に対しては、FPU パイプラインは動作せず、CPU パイプラインのみの動作を行います。

詳しくは「8.8 **各命令のパイプライン動作**」を参照してください。

8.2 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（**図**8.2）。ただし、WBはMA直後に実行されますので、ある命令のMAとWBが同一スロット内で実行されることがあります。

X	←→	←―→	←→	←→	←→	←→	←→	←→	←→	：スロット
命令1	IF	ID	EX	MA	WA					
命令2		IF		ID	EX	MA	MB			

【注】 命令1のID、EXが1つのスロットで実行されている。

図 8.2 **ありえないパイプラインの流れ（1）**

（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。（**図**8.3）。

X	←→	←→	←→	←→	←→	←→	←→	←→	←→	：スロット
命令1	IF	ID	EX	MA	WB					
命令2	IF	ID	EX	MA	WB					
命令3		IF	ID	EX	MA	WB				
命令4			IF	ID	EX	MA	WB			
命令5			IF	ID	EX	MA	WB			

【注】 1スロット内で別の命令の同じステージが実行されている。

図 8.3 **ありえないパイプラインの流れ（2）**

(3) 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。

(a) S＝（1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

(b) 各ステージの実行ステート数は……
- ●IF ：命令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、**図**8.4は、命令1、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示しています。"——"はストールを表しています。

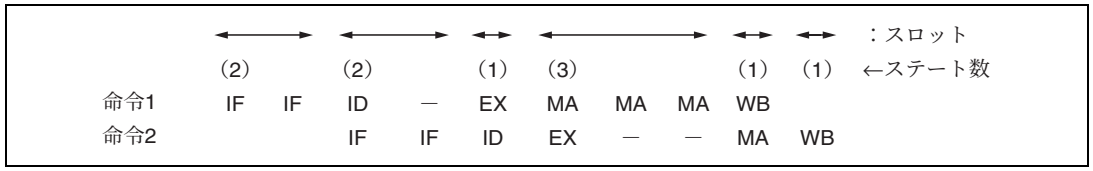


図 8.4 **複数サイクルかかるスロット**

8.3 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 断実行開始から、次に続く命令 2 の EX 断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。

たとえば、**図** 8.5 のような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 5 ステージですから、命令 1 の実行時間は 5 ステートということになります。また、命令 2 と命令 3 の EX ステージの間隔は 1 ステートですから、命令 2 の実行時間は 1 ステートということになります。

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 をおいて、命令 3 と命令 4 の EX ステージの間隔から算出してください（**図** 8.5 の場合、命令 3 の実行時間は 1 ステートになります）。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、「8.4 **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 8.5 の命令 1～命令 3 までの実行時間は合計 5＋1＋1＝7 ステートとなります。

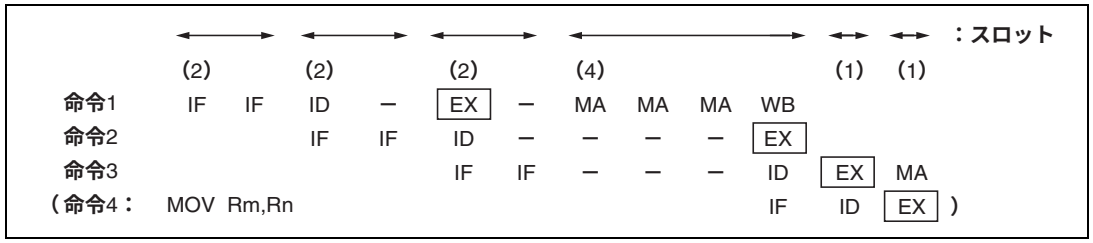


図 8.5 **命令実行ステート数の数え方の例**

8.4 命令フェッチ（IF）とメモリアクセス（MA）の競合

(1) 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にいきますので、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にいこうとすると、そのスロットは**図** 8.6 に示すようにスプリットします。なお、WB がある場合は MA 終了直後に実行されます。

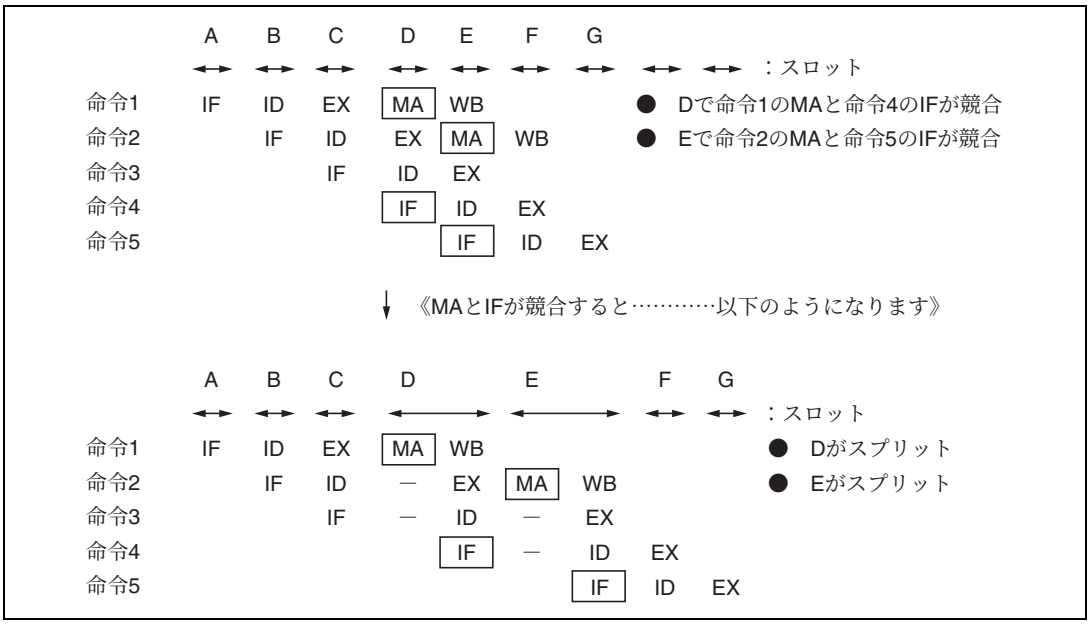


図 8.6 IF と MA の競合時の動作

MA と IF が競合したスロットはスプリットされ、まず前半で、MA が優先して実行され（WB がある場合はその MA 終了直後に実行されます）、後半で他の EX、ID、IF が同時に実行されます。たとえば、**図** 8.6 の場合、スロット D では命令 1 の MA が優先し、命令 2 の EX、命令 3 の ID、命令 4 の IF の 3 つは、あとから同時に実行されることになります。またスロット E では、命令 2 の MA が優先し、命令 3 の EX、命令 4 の ID、命令 5 の IF は、あとから同時に実行されます。

MA と IF が競合したスロットでかかるステート数は、MA でのメモリアクセスサイクル数と IF でのメモリアクセスサイクル数の和になります。

(2) メモリに配置された命令の位置と、IF の関係

SH-2E はメモリから命令を 32 ビットでアクセスします。SH-2E 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。IF がフェッチュするのが 1 命令か 2 命令かは、その命令がメモリのどこに配置されているかによります（ワード／ロングワード境界）。

ロングワード境界にある場合、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になります。

すなわち、メモリに配置されている命令のうち、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** 8.7 に示します。

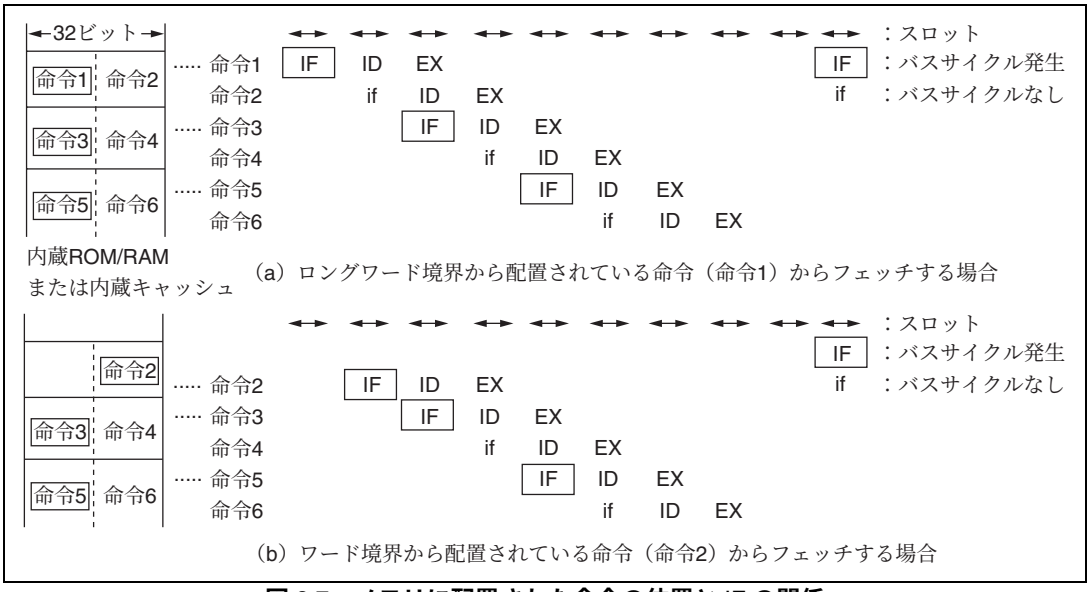
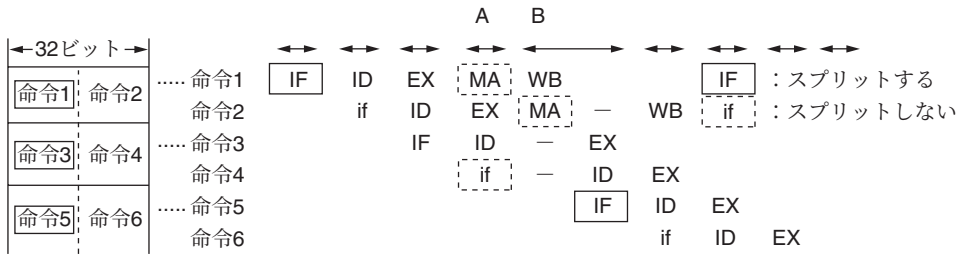


図 8.7 **メモリに配置された命令の位置と** IF **の関係**

(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（2）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、スロットのストリップは発生しません。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** 8.8 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。



【注】 スロットAでは、MAとifの競合なのでスプリットしませんが、スロットBでは、MAとIFの競合なのでスプリットする。

図 8.8 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

8.5 メモリロード命令によるパイプラインへの影響

メモリからのロード命令は、デスティネーションレジスタへのデータ戻しがパイプライン最後の WB 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 WB ステージが来る前に、命令 2 の EX ステージが来ることになります。

このとき、ロード命令 1 のデスティネーションレジスタを命令 2 が使おうとすると、まだそのレジスタの内容が準備できていないので、命令 1 の MA と命令 2 の EX があるスロットはスプリットされます。ロード命令 1 のデスティネーションレジスタが、命令 2 のソースでなくデスティネーションと同じでもスロットはスプリットされます。

なお、ロード命令 1 のデスティネーションがステータスレジスタ（SR）で、その中のフラグを命令 2 が取り込んで使用する場合（たとえば、ADDC など）もスプリットが発生します。

ただし、次の場合はスプリットしません。

（1） 命令2がロード命令で、そのデスティネーションが、ロード命令1のデスティネーションと同じ場合。
（2） 命令2が、MAC @Rm+、@Rn+の場合、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。

このスプリットが発生したスロットのステート数は、MA のサイクル数＋IF（または if）のサイクル数になります。その様子を**図 8.9** 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。

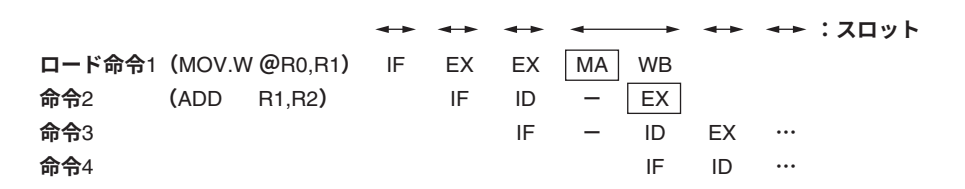


図 8.9 メモリロード命令によるパイプラインへの影響

8.6 FPU による競合

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（FR0～FR15、FPUL）が、次に続く命令によってリード（ソースレジスタとして使用）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** 8.10）。

スロット	←→	←→	←→	←→	←→	←→	←→	←→	←→
浮動小数点数算術演算命令（FADD FR1, FR2）	IF	DF	E1	E2	SF				
次浮動小数点命令（FMOV FR2, FR3）		IF	DF	ー	E1	E2	SF		

図 8.10 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は、2 スロット期間分実行が遅延します（**図** 8.11）。

スロット	←→	←→	←→	←→	←→	←→	←→	←→	←→
命令1（LDS R2, FPSCR）	IF	DF	E1	E2	SF				
命令2（FADD FR4, FR5）		IF	DF	ー	ー	E1	E2	SF	

図 8.11 FPU **による競合** 2

STS または STS.L 命令を使用して FPSCR の値をリードすると、2 スロット期間分実行が遅延します（**図** 8.12）。

スロット	←→	←→	←→	←→	←→	←→	←→	←→	←→
命令1（FADD FR6, FR9）	IF	DF	E1	E2	SF				
命令2（STS FPSCR, R3）		IF	DF	ー	ー	E1	E2	SF	

図 8.12 FPU **による競合** 3

FDIV 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および FPU に関する CPU 命令が E1 ステージに入ることはありません。FDIV 命令が E1 ステージを終了する前に、他の浮動小数点命令や FPU に関する CPU 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 命令が SF ステージを終了した後に E1 ステージに入ります（**図** 8.13）。

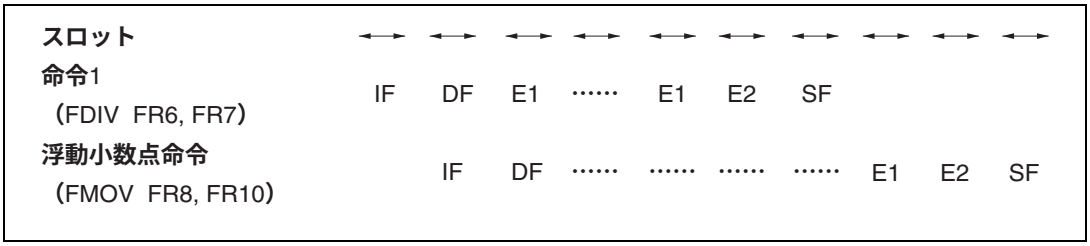


図 8.13 FPU による競合 4

8.7 プログラミングの指針

プログラムを作る場合は、命令実行速度を向上させるため、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、A＝00）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を使わない命令を配置します。
（3）乗算器を使う命令が連続しないように配置します。乗算器からの結果の取り出しのためのMACH、MACLレジスタへのアクセスも乗算器を使う命令に連続しないように配置します。
（4）浮動小数点数演算命令の直前の命令は浮動小数点数演算命令のデスティネーションレジスタを使用しないようにします。
（5） FDIV命令の後14命令以内に浮動小数点命令およびFPUに関するCPU命令をできるだけ置かないようにします。

8.8 各命令のパイプライン動作

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 と競合するとスロットがスプリットしますが、そのスプリットの状況については、特別な場合を除き、以下の図では表していません。スロットがスプリットすると判断された場合は、「**8.4 命令フェッチ（IF）とメモリアクセス（MA）の競合**」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。

命令のステージ段数と実行ステート数を以下の様式で表示します。

命令のステージ段数と実行ステート数の様式

分類	区分	ステージ段数	実行ステート	競合	命令
機能別の分類です。	命令を動作の違いで区分しています。	命令のステージの段数です。	競合がない場合の実行ステート数です。	発生する競合を表します。	対応する命令をニーモニックで表示します。

表 8.1 命令のステージ段数と実行ステート数

分類	区分	ステージ段数	実行ステート	競合	命令	
データ転送命令	レジスター レジスタ間 転送命令	3	1	—	MOV	#imm,Rn
					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	5	1	・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 ・MA は IF と競合します	MOV.W	@(disp,PC),Rn
					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	4	1	・MA は IF と競合します	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	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)

分類	区分	ステージ段数	実行ステート	競合	命令	
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）	3	1	—	ADD	Rm,Rn
					ADD	#imm,Rn
					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/PZ	Rn
					CMP/PL	Rn
					CMP/STR	Rm,Rn
					DIV1	Rm,Rn
					DIV0S	Rm,Rn
					DIV0U	
					DT	Rn
					EXTS.B	Rm,Rn
					EXTS.W	Rm,Rn
					EXTU.B	Rm,Rn
					EXTU.W	Rm,Rn
					NEG	Rm,Rn
					NEGC	Rm,Rn
					SUB	Rm,Rn
					SUBC	Rm,Rn
					SUBV	Rm,Rn
	積和命令	7	3/(2)*1	・こ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 ・MA は IF と競合します	MAC.W	@Rm+,@Rn+
	倍精度積和命令	9	3/(2〜4)*1	・こ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 ・MA は IF と競合します	MAC.L	@Rm+,@Rn+
	乗算命令	6	1〜3*1	・こ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 ・MA は IF と競合します	MULS.W	Rm,Rn
					MULU.W	Rm,Rn
	倍精度乗算命令	9	2〜4*1	・こ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 ・MA は IF と競合します	DMULS.L	Rm,Rn
					DMULU.L	Rm,Rn
					MUL.L	Rm,Rn

分類	区分	ステージ段数	実行ステート	競合	命令	
論理演算命令	レジスタ−レジスタ間論理演算命令	3	1	—	AND	Rm,Rn
					AND	#imm,R0
					NOT	Rm,Rn
					OR	Rm,Rn
					OR	#imm,R0
					TST	Rm,Rn
					TST	#imm,R0
					XOR	Rm,Rn
					XOR	#imm,R0
	メモリ論理演算命令	6	3	• MA は IF と競合します	AND.B	#imm,@(R0,GBR)
					OR.B	#imm,@(R0,GBR)
					TST.B	#imm,@(R0,GBR)
					XOR.B	#imm,@(R0,GBR)
	TAS 命令	6	4	• MA は IF と競合します	TAS.B	@Rn
シフト命令	シフト命令	3	1	—	ROTL	Rn
					ROTR	Rn
					ROTCL	Rn
					ROTCR	Rn
					SHAL	Rn
					SHAR	Rn
					SHLL	Rn
					SHLR	Rn
					SHLL2	Rn
					SHLR2	Rn
					SHLL8	Rn
					SHLR8	Rn
					SHLL16	Rn
					SHLR16	Rn
分岐命令	条件分岐命令	3	3/1*2	—	BF	label
					BT	label
	遅延付き条件分岐命令	3	2/1*2	—	BF/S	label
					BT/S	label
	無条件分岐命令	3	2	—	BRA	label
					BRAF	Rm
					BSR	label
					BSRF	Rm
					JMP	@Rm
					JSR	@Rm
					RTS	

分類	区分	ステージ段数	実行ステート	競合	命令	
システム制御命令	システム制御 ALU 命令	3	1	—	CLRT	
					LDC	Rm,SR
					LDC	Rm,GBR
					LDC	Rm,VBR
					LDS	Rm,PR
					NOP	
					SETT	
					STC	SR,Rn
					STC	GBR,Rn
					STC	VBR,Rn
					STS	PR,Rn
	LDC.L 命令	5	3	・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 ・MA は IF と競合します	LDC.L	@Rm+,SR
					LDC.L	@Rm+,GBR
					LDC.L	@Rm+,VBR
	STC.L 命令	4	2	・MA は IF と競合します	STC.L	SR,@-Rn
					STC.L	GBR,@-Rn
					STC.L	VBR,@-Rn
	LDS.L 命令（PR）	5	1	・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 ・MA は IF と競合します	LDS.L	@Rm+,PR
	STS.L 命令（PR）	4	1	・MA は IF と競合します	STS.L	PR,@-Rn
	レジスタ→ MAC 転送命令	4	1	・乗算器との競合を起こします ・MA は IF と競合します	CLRMAC	
					LDS	Rm,MACH
					LDS	Rm,MACL
	メモリ→MAC 転送命令	4	1	・乗算器との競合を起こします ・MA は IF と競合します	LDS.L	@Rm+,MACH
					LDS.L	@Rm+,MACL
	MAC→レジスタ転送命令	5	1	・乗算器との競合を起こします ・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、競合します ・MA は IF と競合します	STS	MACH,Rn
					STS.L	MACL,Rn
	MAC→メモリ転送命令	4	1	・乗算器との競合を起こします ・MA は IF と競合します	STS.L	MACH,@-Rn
					STS.L	MACL,@-Rn
	RTE 命令	5	4	—	RTE	
	TRAP 命令	9	8	—	TRAPA	#imm
	SLEEP 命令	3	3	—	SLEEP	

分類	区分	ステージ段数	実行ステート	競合	命令	
FPU 関する CPU 命令	FPUL ロード命令	5(FPU パイプライン)	1	・次命令が FPUL をリードする命令のとき、競合が発生します ・CPU パイプラインの MA は IF と競合します	LDS	Rm,FPUL
		4(CPU パイプライン)			LDS.L	@Rm+,FPUL
	FPSCR ロード命令	5(FPU パイプライン)	1	・図 8.11 に示したように競合が発生します	LDS	Rm,FPSCR
		4(CPU パイプライン)			LDS.L	@Rm+,FPSCR
	FPUL ストア命令（STS）	4(FPU パイプライン)	1	・次命令に Rn を使う命令を置くと競合します ・CPU パイプラインの MA は IF と競合します	STS	FPUL,Rn
		5(CPU パイプライン)				
	FPUL ストア命令（STS.L）	4(FPU パイプライン)	1	・CPU パイプラインの MA は IF と競合します	STS.L	FPUL,@-Rn
		4(CPU パイプライン)				
	FPSCR ストア命令（STS）	4(FPU パイプライン)	1	・図 8.12 に示したように競合が発生します ・次命令に Rn を使う命令を置くと競合します ・CPU パイプラインの MA は IF と競合します	STS	FPSCR,Rn
		5(CPU パイプライン)				
	FPSCR ストア命令（STS.L）	4(FPU パイプライン)	1	・図 8.12 に示したように競合が発生します ・CPU パイプラインの MA は IF と競合します	STS.L	FPSCR,@-Rn
		4(CPU パイプライン)				
浮動小数点命令	浮動小数点レジスタ―レジスタ間転送命令	5(FPU パイプライン)	1	・次命令がデスティネーションレジスタをリードする命令のとき、競合が発生します	FLDS	FRm,FPUL
		3(CPU パイプライン)			FMOV	FRm,FRn
					FSTS	FPUL,FRn
	浮動小数点レジスタ―イミディエイト命令	5(FPU パイプライン)	1	・次命令がデスティネーションレジスタをリードする命令のとき、競合が発生します	FLDI0	FRn
		3(CPU パイプライン)			FLDI1	FRn
	浮動小数点レジスタロード命令	5(FPU パイプライン)	1	・次命令がデスティネーションレジスタをリードする命令のとき、競合が発生します ・CPU パイプラインの MA は IF と競合します	FMOV.S	@Rm,FRn
		4(CPU パイプライン)			FMOV.S	@Rm+,FRn
					FMOV.S	@(R0,Rm),FRn
	浮動小数点レジスタストア命令	4(FPU パイプライン)	1	・CPU パイプラインの MA は IF と競合します	FMOV.S	FRm,@Rn
		4(CPU パイプライン)			FMOV.S	FRm,@-Rn
					FMOV.S	FRm,@ (R0,Rn)

分類	区分	ステージ段数	実行ステート	競合	命令	
浮動小数点命令	浮動小数点演算命令（FDIV 以外）	5(FPU パイプライン)	1	・次命令がデスティネーションレジスタをリードする命令のとき、競合が発生します	FABS	FRn
		3(CPU パイプライン)			FADD	FRm,FRn
					FLOAT	FPUL,FRn
					FMAC	FR0,FRm,FRn
					FMUL	FRm,FRn
					FNEG	FRn
					FSUB	FRm,FRn
					FTRC	FRm,FPUL
	浮動小数点演算命令（FDIV）	17(FPU パイプライン)	13	・図 8.13 に示したように競合が発生します	FDIV	FRm,FRn
		3(CPU パイプライン)				
	浮動小数点比較命令	3(FPU パイプライン)	1	—	FCMP/EQ	FRm,FRn
		3(CPU パイプライン)			FCMP/GT	FRm,FRn

【注】 *1　通常実行ステートを示します。（ ）内の値は、命令との競合関係による実行ステート数です。
【注】 *2　分岐しないときは 1 ステートになります。

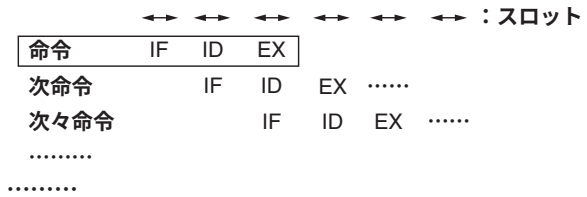
8.8.1 データ転送命令

(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
XTRACT      Rm,Rn
```

●パイプライン



●動作説明

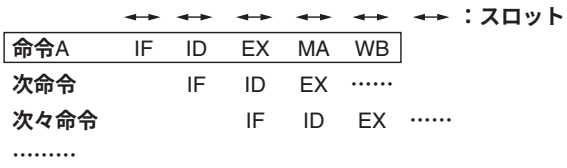
パイプラインは、IF、ID、EX の 3 段で終了します。EX ステージで、ALU を通してデータ転送を行います。

(2) メモリロード命令

●命令の種類

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

●パイプライン



●動作説明

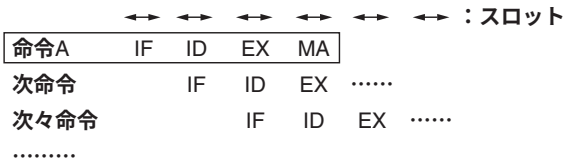
パイプラインは、IF、ID、EX、MA、WB の 5 段です。この命令の直後に、この命令のデスティネーションレジスタを使う命令を置くと、競合が発生します（「8.5 **メモリロード命令によるパイプラインへの影響**」参照）。

(3) メモリストア命令

●命令の種類

```
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       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)
```

●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。

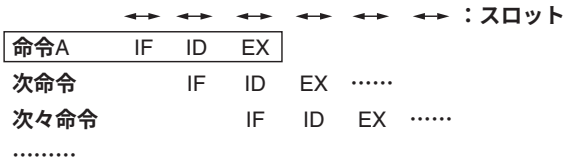
8.8.2 算術演算命令

(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
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/PZ      Rn
CMP/PL      Rn
CMP/STR     Rm,Rn
DIV1        Rm,Rn
DIV0S       Rm,Rn
DIV0U
DT          Rn
EXTS.B      Rm,Rn
EXTS.W      Rm,Rn
EXTU.B      Rm,Rn
EXTU.W      Rm,Rn
NEG         Rm,Rn
NEGC        Rm,Rn
SUB         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 の 3 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。

(2) 積和命令

●命令の種類

```
MAC.W       @Rm+,@Rn+
```

●パイプライン

	←→	←→	←→	←→	←→	←→	←→	←→ : スロット
MAC.W	IF	ID	EX	MA	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、MA、mm、mm の 7 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 2 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、「8.4 **命令フェッチ（IF）とメモリアクセス（MA）の競合**」で説明したようにスロットがスプリットします。

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 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、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。

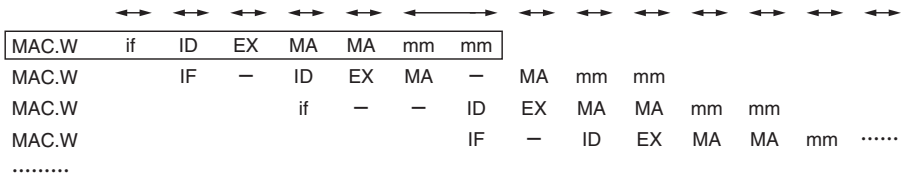
しかし、MAC.W 命令の後ろに乗算器を使う命令がくる場合、乗算器の競合が発生しますので、通常の動作と異なってきます。この場合は、以下のケースが考えられます。

（a） MAC.W命令の直後に、MAC.W命令が連続してくる場合
（b） MAC.W命令の直後に、MAC.L命令が連続してくる場合
（c） MAC.W命令の直後に、MULS.W命令が連続してくる場合
（d） MAC.W命令の直後に、DMULS.L命令が連続してくる場合
（e） MAC.W命令の直後に、STS（レジスタ）命令がくる場合
（f） MAC.W命令の直後に、STS.L（メモリ）命令がくる場合
（g） MAC.W命令の直後に、LDS（レジスタ）命令がくる場合
（h） MAC.W命令の直後に、LDS.L（メモリ）命令がくる場合

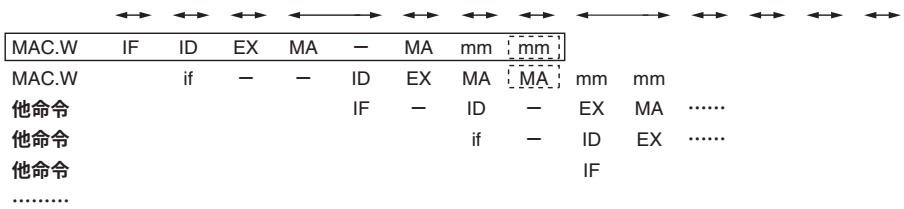
（a）MAC.W命令の直後に、MAC.W命令が連続してくる場合

MAC.W命令の2番目のMAと、その前の乗算系命令によって発生したmmとは競合しません。

MAC.Wの連続により、MAとIFの競合で命令実行がずれる場合もあります。下図を参照してください。この図については、MAとIFの競合を考慮して書いてあります。



MAC.W命令の2番目のMAとIFが競合すると、通常どおりスロットがスプリットします。下図を参照してください。この図については、MAとIFの競合を考慮して書いてあります。

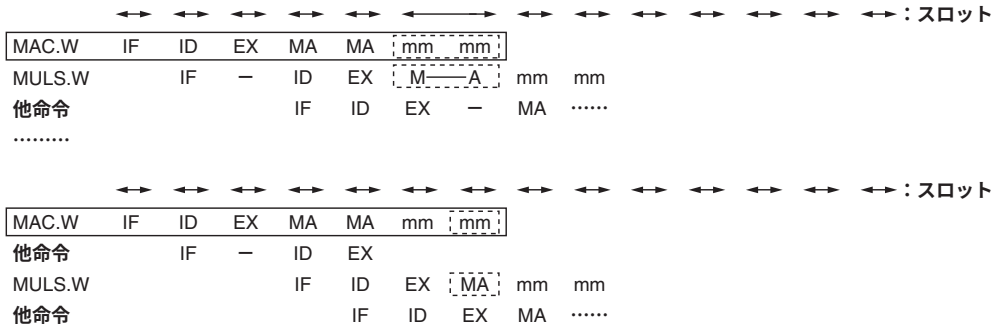


（b）MAC.W命令の直後に、MAC.L命令が連続してくる場合

MAC.W命令の2番目のMAと、その前の乗算系命令によって発生したmmとは競合しません。

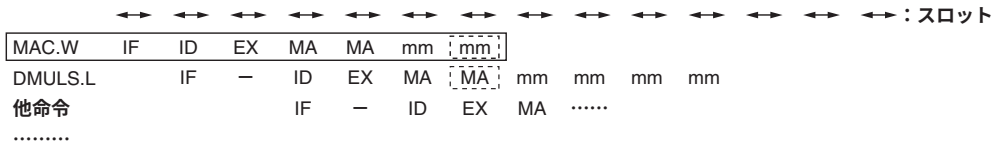
（c）MAC.W命令の直後に、MULS.W命令が連続してくる場合

MULS.W命令には、乗算器アクセスのためのMAステージがあります。MAC.W命令の乗算器の動作中（mm）にMULS.WのMAが競合すると、そのMAはmmが終了するまで引き伸ばされ（下図のM――A）、一つのスロットを形成します。MAC.WとMULS.Wの間に乗算器と無関係な命令が1つ以上はいるとMAC.WとMULS.Wの競合によるストールはなくなります。MAC.W命令の乗算器の動作中（mm）にMULS.WのMAは競合しません。なお、MULS.WのMAもIFと競合するとスロットがスプリットします。



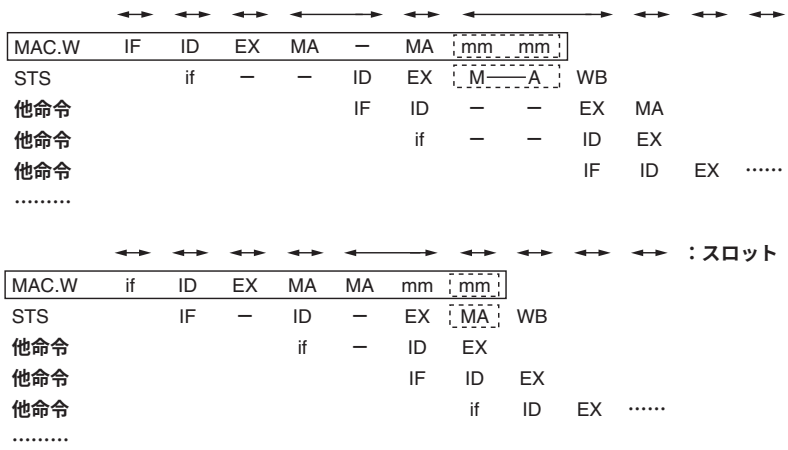
（d）MAC.W命令の直後に、DMULS.L命令が連続してくる場合

MULS.W命令には、乗算器アクセスのためのMAステージがありますが、MAC.W命令の乗算器の動作中（mm）にMULS.WのMAは競合しません。なお、MULS.WのMAもIFと競合するとスロットがスプリットします。



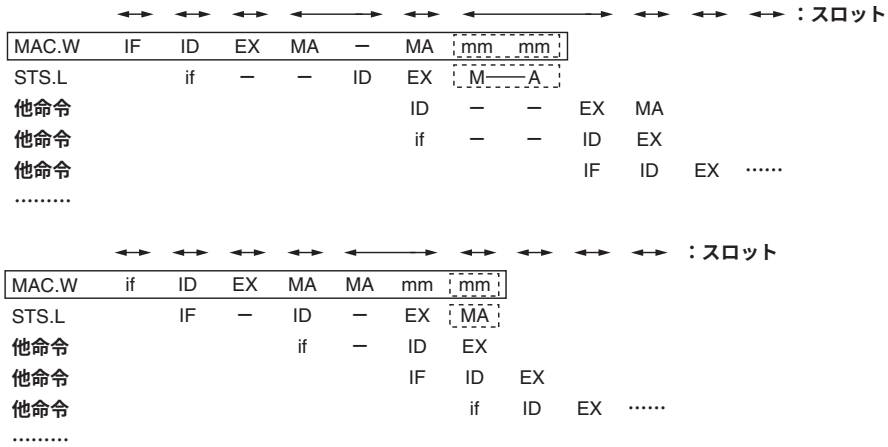
（e）MAC.W命令の直後に、STS（レジスタ）命令がくる場合

STS命令で、MACレジスタの内容を汎用レジスタにストアする場合、後述のとおり、STS命令には乗算器のアクセスのためのMAステージがはいります。乗算器の動作中（mm）にSTSのMAが競合すると、そのMAはmmが終了するまで引き伸ばされ（下図のM――A）、一つのスロットを形成します。また、このSTSのMAはIFと競合します。その様子を下図に示します。これらの図については、MAとIFの競合を考慮して書いてあります。



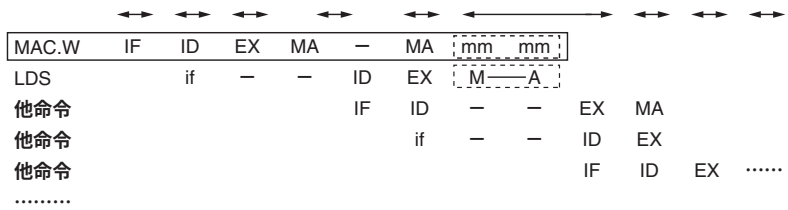
（f）MAC.W命令の直後に、STS.L（メモリ）命令がくる場合

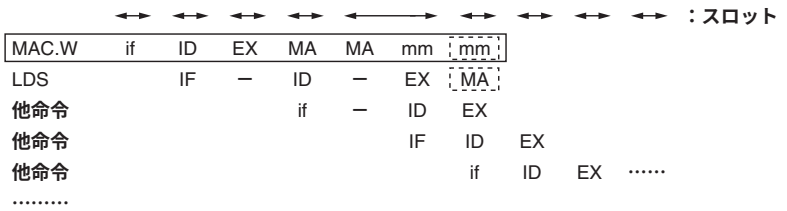
STS命令で、MACレジスタの内容をメモリにストアする場合、後述のとおり、STS命令には乗算器のアクセスと、メモリライトのためのMAステージがはいります。これらの図については、MAとIFの競合を考慮して書いてあります。



（g）MAC.W命令の直後に、LDS（レジスタ）命令がくる場合

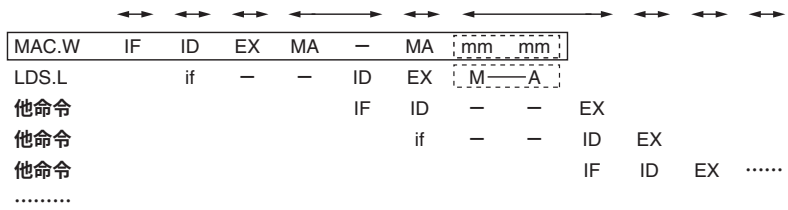
LDS命令で、MACレジスタの内容を汎用レジスタからロードする場合、後述のとおり、LDS命令には乗算器のアクセスのためのMAステージがはいります。乗算器の動作中（mm）にLDSのMAが競合すると、そのMAはmmが終了するまで引き伸ばされ（下図のM――A）、一つのスロットを形成します。また、このLDSのMAはIFと競合します。その様子を下図に示します。これらの図については、MAとIFの競合を考慮して書いてあります。

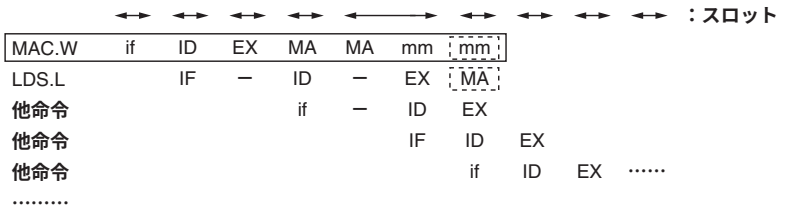




（h）MAC.W命令の直後に、LDS.L（メモリ）命令がくる場合

LDS命令で、MACレジスタの内容をメモリからロードする場合、後述のとおり、LDS命令にはメモリのアクセスと乗算器のアクセスのためのMAステージがはいります。乗算器の動作中（mm）にLDSのMAが競合すると、そのMAはmmが終了するまで引き伸ばされ（下図のM――A）、一つのスロットを形成します。また、このLDSのMAはIFと競合します。その様子を下図に示します。これらの図については、MAとIFの競合を考慮して書いてあります。





(3) 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
MAC.L	IF	ID	EX	MA	MA	mm	mm	mm	mm
次命令		IF	ー	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、mm の 9 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 4 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、「8.4 **命令フェッチ（IF）とメモリアクセス（MA）の競合**」で説明したようにスロットがスプリットします。

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 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、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。

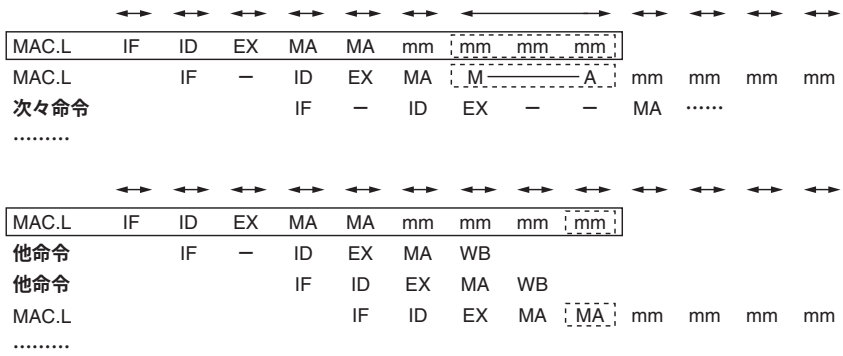
しかし、MAC.L 命令の後ろに乗算器を使う命令がくる場合、乗算器の競合が発生しますので、通常の動作と異なってきます。この場合は、以下のケースが考えられます。

（a） MAC.L命令の直後に、MAC.L命令が連続してくる場合
（b） MAC.L命令の直後に、MAC.W命令が連続してくる場合
（c） MAC.L命令の直後に、DMULS.L命令が連続してくる場合
（d） MAC.L命令の直後に、MULS.W命令が連続してくる場合
（e） MAC.L命令の直後に、STS（レジスタ）命令がくる場合
（f） MAC.L命令の直後に、STS.L（メモリ）命令がくる場合
（g） MAC.L命令の直後に、LDS（レジスタ）命令がくる場合
（h） MAC.L命令の直後に、LDS.L（メモリ）命令がくる場合

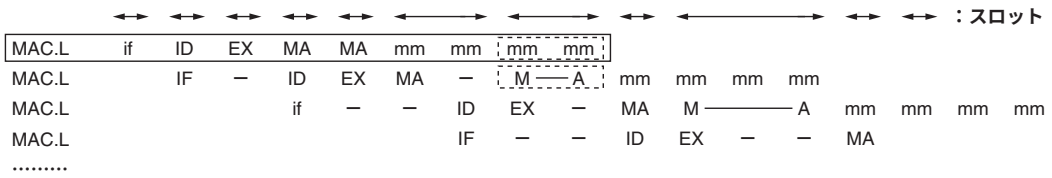
（a）MAC.L命令の直後に、MAC.L命令が連続してくる場合

MAC.L命令の2番目のMAが、その前の乗算系命令によって発生したmmと競合した場合、そのMAのバスサイクルはmmが終了するまで引き伸ばされ（下図のM――A）、その引き伸ばされたMAは一つのスロットになります。

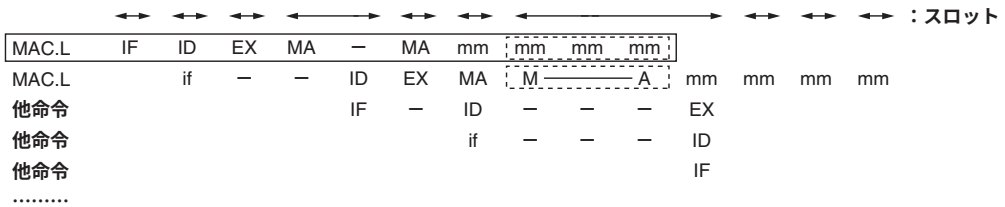
MAC.L命令とMAC.L命令の間に、乗算器と無関係な命令が2つ以上入ると、MAC.L命令どうしの乗算器の競合によるストールはなくなります。



MAC.Lの連続により、MAとIFの競合で命令実行がずれても乗算器の競合が少なくなる場合があります。下図を参照してください。この図については、MAとIFの競合を考慮して書いてあります。



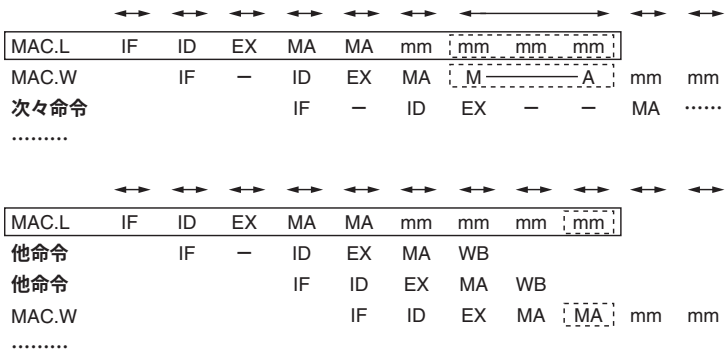
MAC.L命令の2番目のMAがmm終了まで引き伸ばされている場合、このMAとIFが競合すると、通常どおりスロットがスプリットします。下図を参照してください。この図については、MAとIFの競合を考慮して書いてあります。



(b) MAC.L命令の直後に、MAC.W命令が連続してくる場合

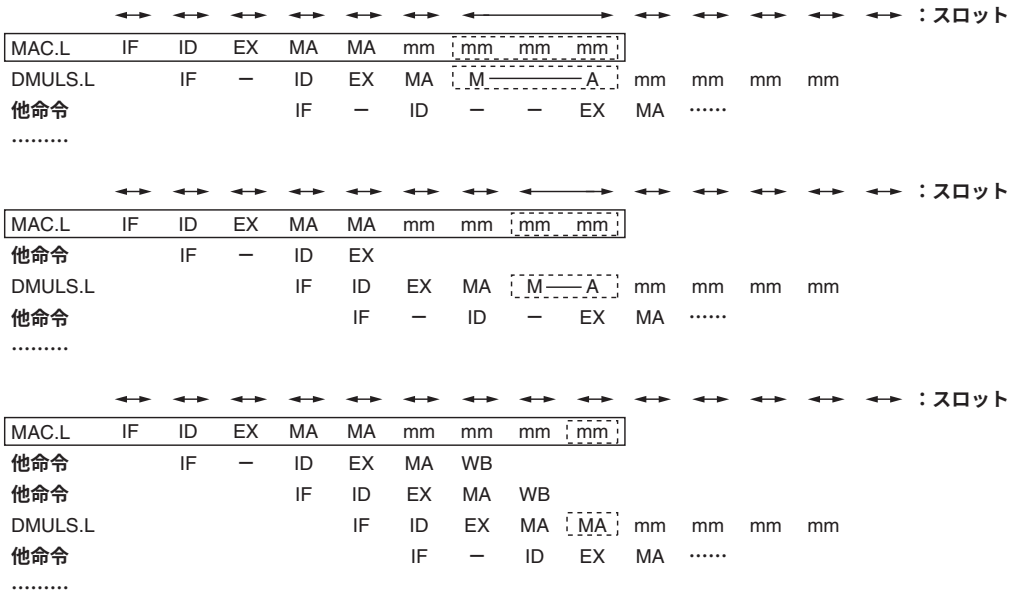
MAC.W命令の2番目のMAが、その前の乗算系命令によって発生したmmと競合した場合、そのMAのバスサイクルはmmが終了するまで引き伸ばされ（下図のM――A）、その引き伸ばされたMAは一つのスロットになります。

MAC.L命令とMAC.W命令の間に、乗算器と無関係な命令が2つ以上入ると、MAC.L命令とMAC.W命令の乗算器の競合によるストールはなくなります。



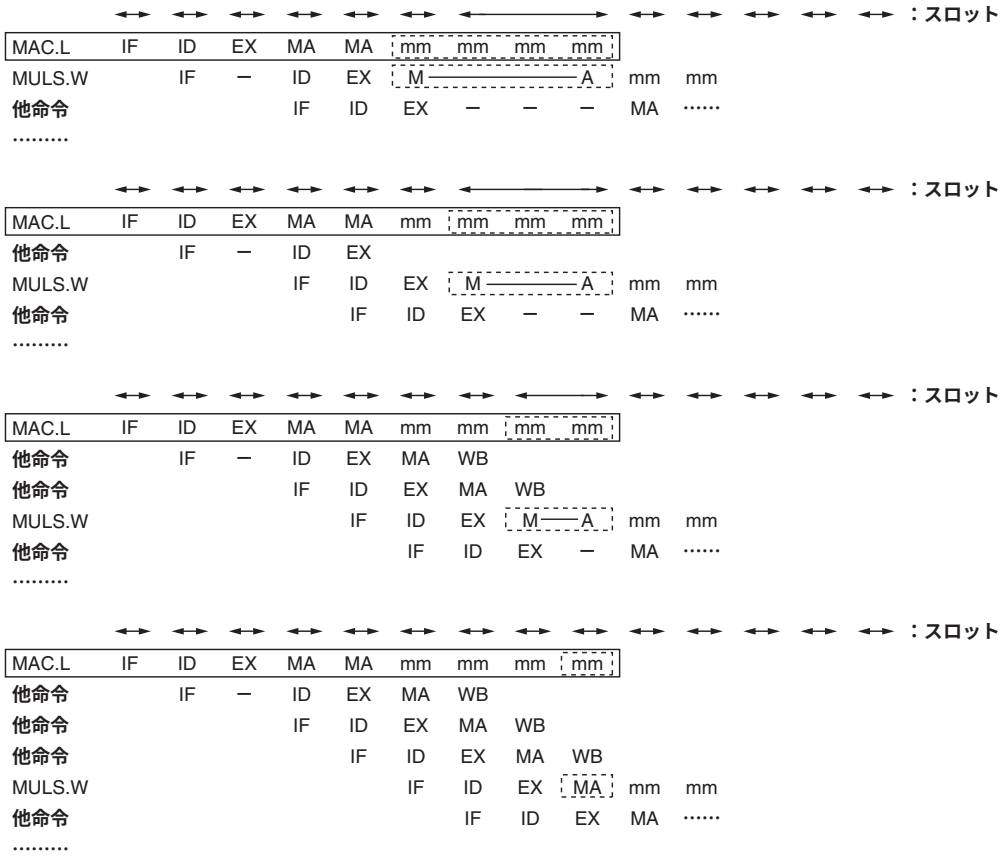
(c) MAC.L命令の直後に、DMULS.L命令が連続してくる場合

DMULS.L命令には、乗算器アクセスのためのMAステージがあります。MAC.L命令の乗算器の動作中（mm）にDMULS.L命令の2番目のMAが競合すると、そのMAはmmが終了するまで引き伸ばされ（下図のM――A）、一つのスロットを形成します。MAC.LとDMULS.Lの間に乗算器と無関係な命令が2つ以上はいるとMAC.LとDMULS.Lの競合によるストールはなくなります。なお、DMULS.LのMAもIFと競合するとスロットがスプリットします。



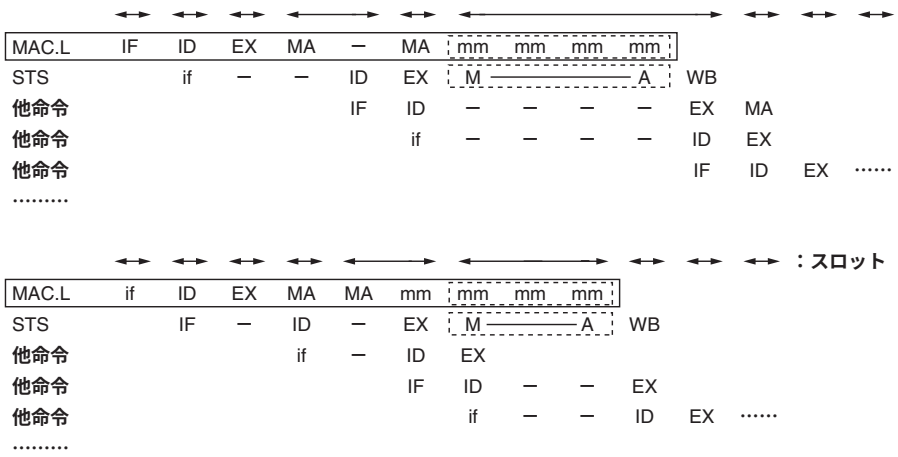
（d）MAC.L命令の直後に、MULS.W命令が連続してくる場合

MULS.W命令には、乗算器アクセスのためのMAステージがあります。MAC.L命令の乗算器の動作中（mm）にMULS.WのMAが競合すると、そのMAはmmが終了するまで引き伸ばされ（下図のM――A）、一つのスロットを形成します。MAC.LとMULS.Wの間に乗算器と無関係な命令が3つ以上はいるとMAC.LとMULS.Wの競合によるストールはなくなります。なお、MULS.WのMAもIFと競合するとスロットがスプリットします。



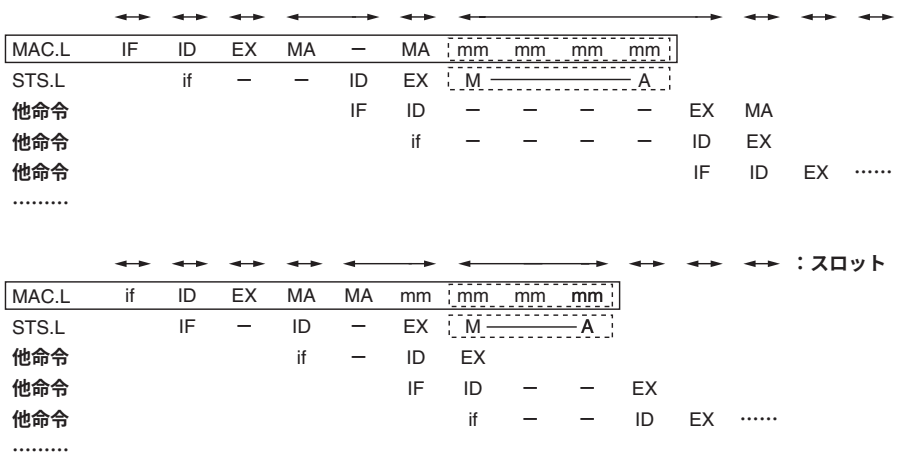
(e) MAC.L命令の直後に、STS（レジスタ）命令がくる場合

STS命令で、MACレジスタの内容を汎用レジスタにストアする場合、後述のとおり、STS命令には乗算器のアクセスのためのMAステージがはいります。乗算器の動作中（mm）にSTSのMAが競合すると、そのMAはmmが終了するまで引き伸ばされ（下図のM――A）、一つのスロットを形成します。また、このSTSのMAはIFと競合します。その様子を下図に示します。これらの図については、MAとIFの競合を考慮して書いてあります。



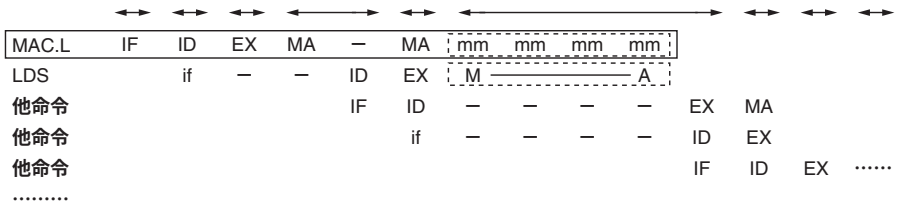
(f) MAC.L命令の直後に、STS.L（メモリ）命令がくる場合

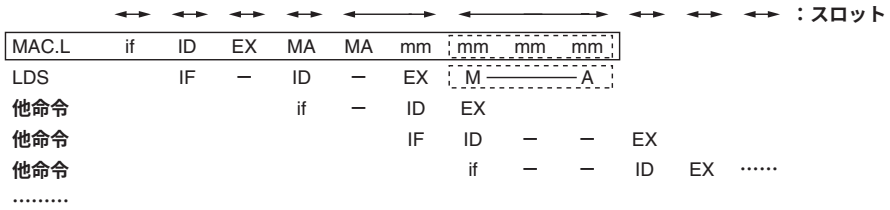
STS命令で、MACレジスタの内容をメモリにストアする場合、後述のとおり、STS命令には乗算器のアクセスと、メモリライトのためのMAステージがはいります。また、このSTSのMAはIFと競合します。その様子を下図に示します。これらの図については、MAとIFの競合を考慮して書いてあります。



（g）MAC.L命令の直後に、LDS（レジスタ）命令がくる場合

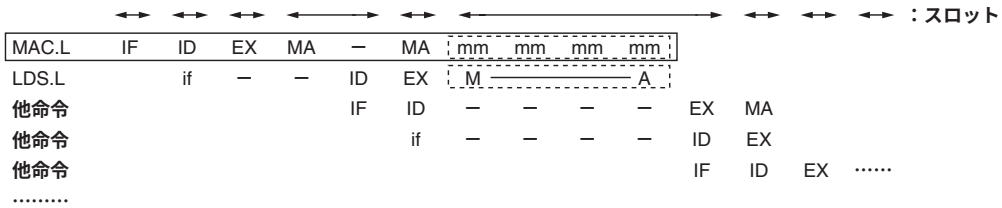
LDS命令で、MACレジスタの内容を汎用レジスタからロードする場合、後述のとおり、LDS命令には乗算器のアクセスのためのMAステージがはいります。乗算器の動作中（mm）にLDSのMAが競合すると、そのMAはmmが終了するまで引き伸ばされ（下図のM――A）、一つのスロットを形成します。また、このLDSのMAはIFと競合します。その様子を下図に示します。これらの図については、MAとIFの競合を考慮して書いてあります。

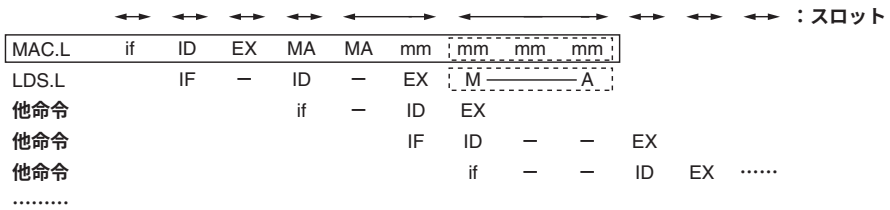




（h）MAC.L命令の直後に、LDS.L（メモリ）命令がくる場合

LDS命令で、MACレジスタの内容をメモリからロードする場合、後述のとおり、LDS命令にはメモリのアクセスと乗算器のアクセスのためのMAステージがはいります。乗算器の動作中（mm）にLDSのMAが競合すると、そのMAはmmが終了するまで引き伸ばされ（下図のM――A）、一つのスロットを形成します。また、このLDSのMAはIFと競合します。その様子を下図に示します。これらの図については、MAとIFの競合を考慮して書いてあります。



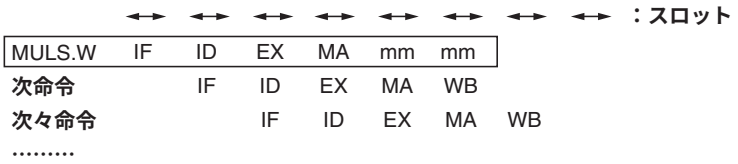


(4) 乗算命令

●命令の種類

```
MULS.W      Rm,Rn
MULU.W      Rm,Rn
```

●パイプライン



●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「8.4 **命令フェッチ（IF）とメモリアクセス（MA）の競合**」で説明したようにスロットがスプリットします。

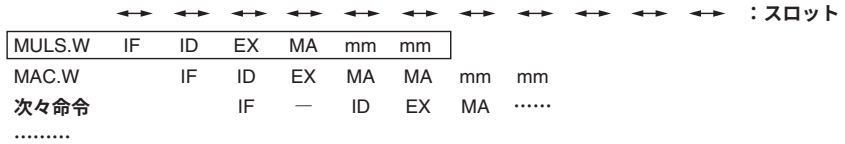
MULS.W 命令の後ろに乗算器を使わない命令がくる場合は、MULS.W 命令は IF、ID、EX、MA の 4 段パイプライン命令とみなして考えてかまいません。すなわち、この場合は、通常のパイプライン動作と同様になります。

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。この場合は、以下のケースが考えられます。

（a） MULS.W命令の直後に、MAC.W命令が連続してくる場合
（b） MULS.W命令の直後に、MAC.L命令が連続してくる場合
（c） MULS.W命令の直後に、MULS.W命令が連続してくる場合
（d） MULS.W命令の直後に、DMULS.L命令が連続してくる場合
（e） MULS.W命令の直後に、STS（レジスタ）命令がくる場合
（f） MULS.W命令の直後に、STS.L（メモリ）命令がくる場合
（g） MULS.W命令の直後に、LDS（レジスタ）命令がくる場合
（h） MULS.W命令の直後に、LDS.L（メモリ）命令がくる場合

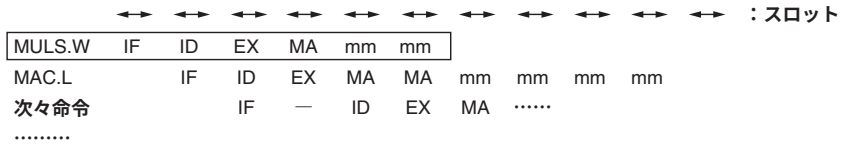
（a）MULS.W命令の直後に、MAC.W命令が連続してくる場合

MAC.W命令の2番目のMAと、その前の乗算系命令によって発生したmmとは競合しません。



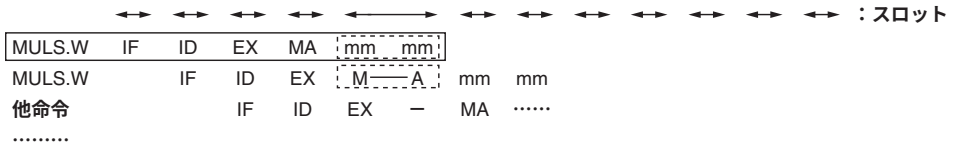
（b）MULS.W命令の直後に、MAC.L命令が連続してくる場合

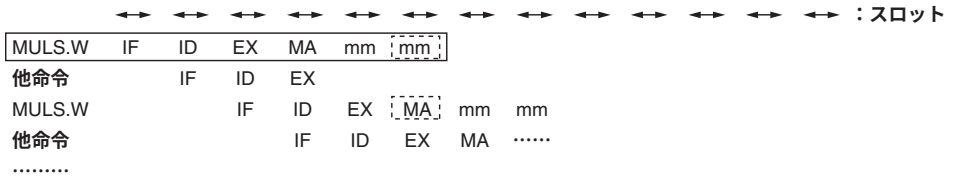
MAC.W命令の2番目のMAと、その前の乗算系命令によって発生したmmとは競合しません。



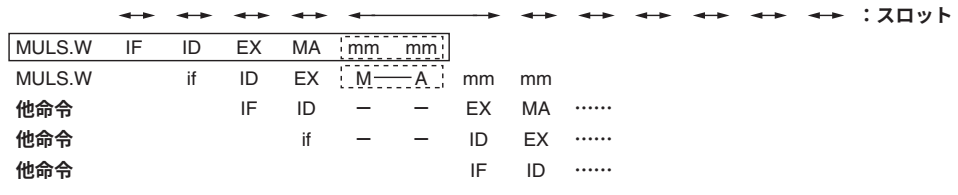
（c）MULS.W命令の直後に、MULS.W命令が連続してくる場合

MULS.W命令には、乗算器アクセスのためのMAステージがあります。MULS.W命令の乗算器の動作中（mm）に他のMULS.WのMAが競合すると、そのMAはmmが終了するまで引き伸ばされ（下図のM――A）、一つのスロットを形成します。MULS.WとMULS.Wの間に乗算器と無関係な命令が1つ以上はいるとMULS.WとMULS.Wの競合によるストールはなくなります。なお、MULS.WのMAもIFと競合するとスロットがスプリットします。



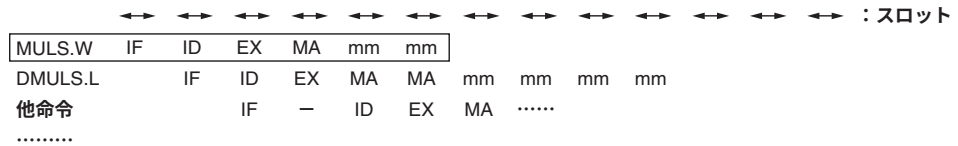


MULS.W命令のMAがmm終了まで引き伸ばされている場合、このMAとIFが競合すると、通常どおりスロットがスプリットします。下図を参照してください。この図については、MAとIFの競合を考慮して書いてあります。



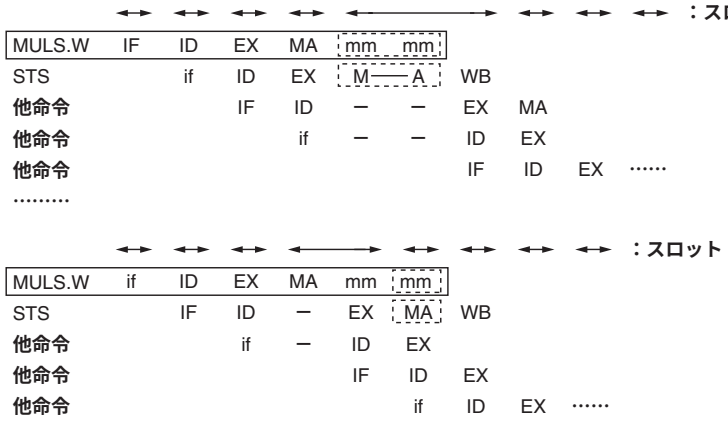
（d）MULS.W命令の直後に、DMULS.L命令が連続してくる場合

DMULS.L命令の2番目のMAは乗算器アクセスを行いますがMULS.W命令によって発生したmmとは競合しません。



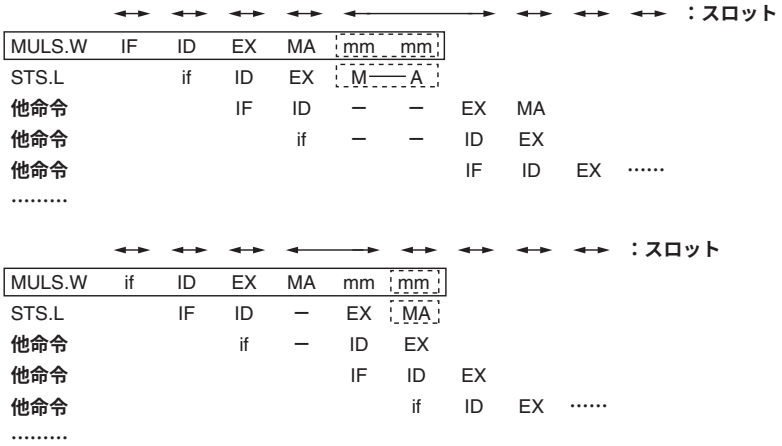
（e）MULS.W命令の直後に、STS（レジスタ）命令がくる場合

STS命令で、MACレジスタの内容を汎用レジスタにストアする場合、後述のとおり、STS命令には乗算器のアクセスのためのMAステージがはいります。乗算器の動作中（mm）にSTSのMAが競合すると、そのMAはmmが終了するまで引き伸ばされ（下図のM――A）、一つのスロットを形成します。また、このSTSのMAはIFと競合します。その様子を下図に示します。これらの図については、MAとIFの競合を考慮して書いてあります。



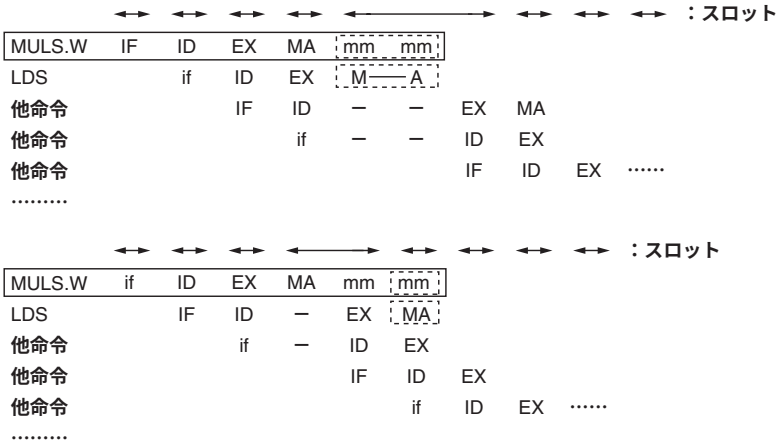
（f）MULS.W命令の直後に、STS.L（メモリ）命令がくる場合

STS命令で、MACレジスタの内容をメモリにストアする場合、後述のとおり、STS命令には乗算器のアクセスと、メモリライトのためのMAステージがはいります。また、このSTSのMAはIFと競合します。その様子を下図に示します。これらの図については、MAとIFの競合を考慮して書いてあります。



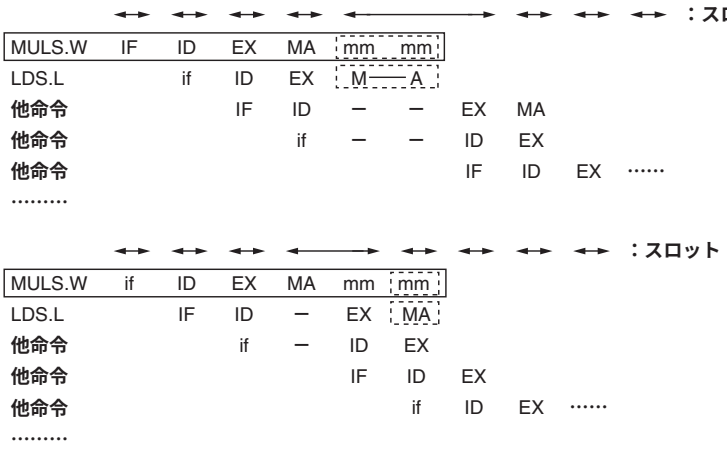
（g）MULS.W命令直後に、LDS（レジスタ）命令が来る場合

LDS命令で、MACレジスタの内容を汎用レジスタからロードする場合、後述のとおり、LDS命令には乗算器のアクセスするためのMAステージがはいります。乗算器の動作中（mm）にLDSのMAが競合すると、そのMAはmmが終了するまで引き伸ばされ（下図のM——A）、一つのスロットを形成します。また、このLDSのMAはIFと競合します。その様子を下図に示します。これらの図については、MAとIFの競合を考慮して書いてあります。



（h）MULS.W命令の直後に、LDS.L（メモリ）命令が来る場合

LDS命令で、MACレジスタの内容をメモリからロードする場合、後述のとおり、LDS命令には乗算器のアクセスのためのMAステージがはいります。乗算器の動作中（mm）にLDSのMAが競合すると、そのMAはmmが終了するまで引き伸ばされ（下図のM――A）、一つのスロットを形成します。また、このLDSのMAはIFと競合します。その様子を下図に示します。これらの図については、MAとIFの競合を考慮して書いてあります。

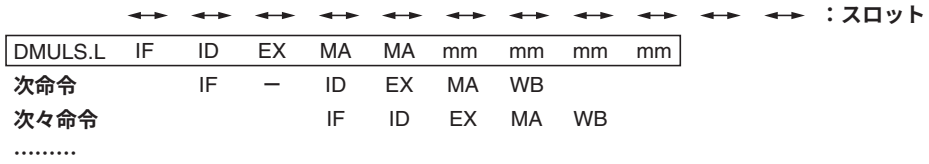


(5) 倍精度の乗算命令

●命令の種類

```
DMULS.L     Rm,Rn
DMULU.L     Rm,Rn
MUL.L       Rm,Rn
```

●パイプライン



●動作説明

パイプラインは、IF、ID、EX、MA、MA、mm、mm、mm、mm の 9 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 4 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（積和命令参照）。DMULS.L 命令の 2 個の MA も、IF と競合する場合は「8.4 **命令フェッチ（IF）とメモリアクセス（MA）の競合**」で説明したようにスロットがスプリットします。

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。

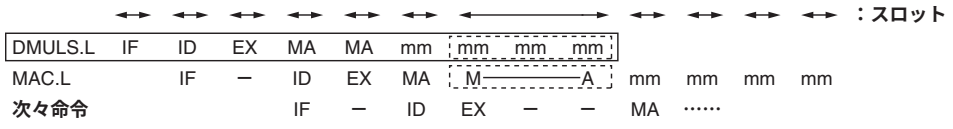
しかし、DMULS.L 命令の後ろに乗算器を使う命令がくる場合、乗算器の競合が発生しますので、通常の動作と異なってきます。この場合は、以下のケースが考えられます。

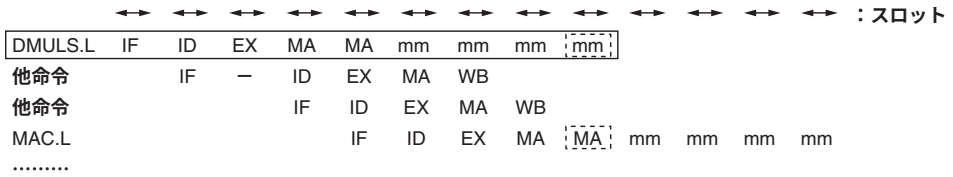
（a） DMULS.L命令の直後に、MAC.L命令が連続してくる場合
（b） DMULS.L命令の直後に、MAC.W命令が連続してくる場合
（c） DMULS.L命令の直後に、DMULS.L命令が連続してくる場合
（d） DMULS.L命令の直後に、MULS.W命令が連続してくる場合
（e） DMULS.L命令の直後に、STS（レジスタ）命令がくる場合
（f） DMULS.L命令の直後に、STS.L（メモリ）命令がくる場合
（g） DMULS.L命令の直後に、LDS（レジスタ）命令がくる場合
（h） DMULS.L命令の直後に、LDS.L（メモリ）命令がくる場合

（a）DMULS.L命令の直後に、MAC.L命令が連続してくる場合

MAC.L命令の2番目のMAが、その前の乗算系命令によって発生したmmと競合した場合、そのMAのバスサイクルはmmが終了するまで引き伸ばされ（下図のM――A）、その引き伸ばされたMAは一つのスロットになります。

DMULS.L命令とMAC.L命令の間に、乗算器と無関係な命令が2つ以上入ると、DMULS.LとMAC.L命令どうしの乗算器の競合によるストールはなくなります。

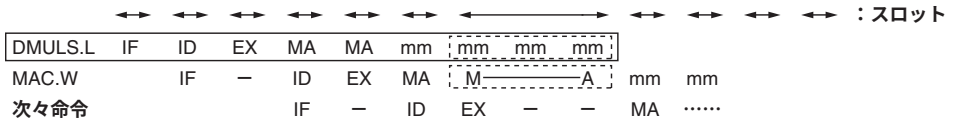


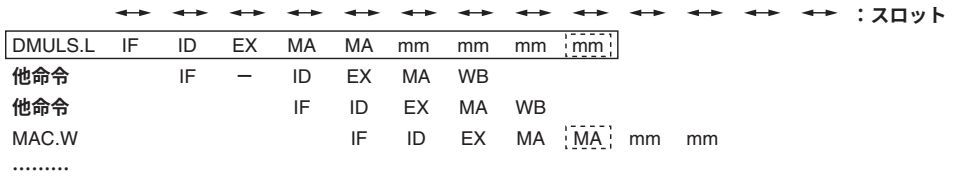


（b）DMULS.L命令の直後に、MAC.W命令が連続してくる場合

MAC.W命令の2番目のMAが、その前の乗算系命令によって発生したmmと競合した場合、そのMAのバスサイクルはmmが終了するまで引き伸ばされ（下図のM――A）、その引き伸ばされたMAは一つのスロットになります。

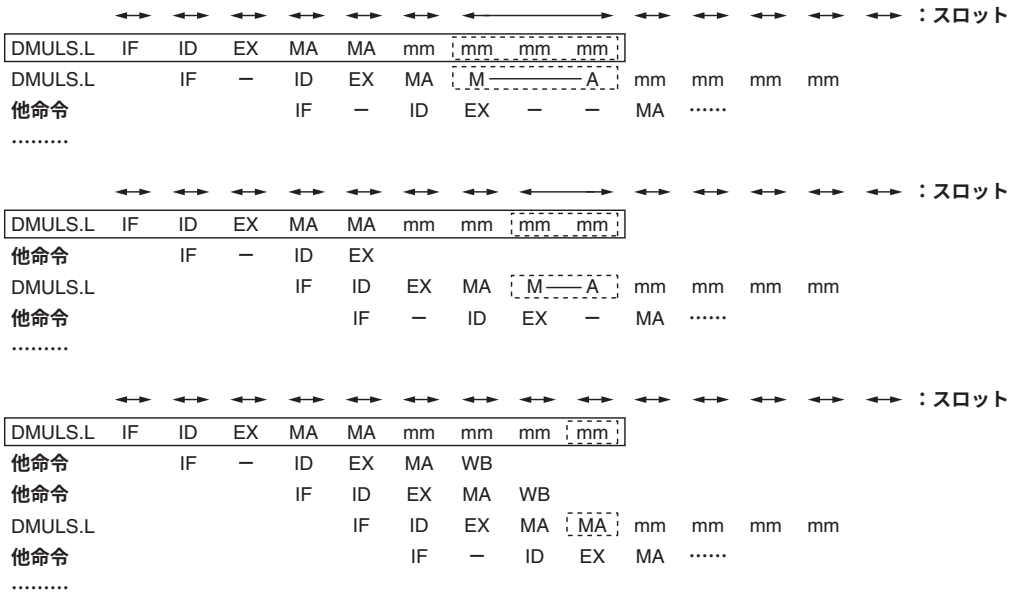
DMULS.L命令とMAC.W命令の間に、乗算器と無関係な命令が2つ以上入ると、DMULS.LとMAC.W命令どうしの乗算器の競合によるストールはなくなります。



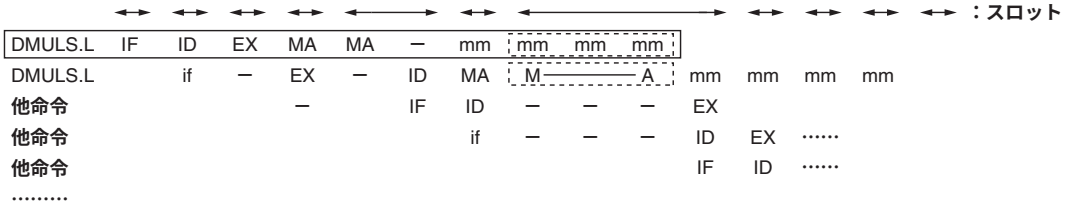


（c）DMULS.L命令の直後に、DMULS.L命令が連続してくる場合

DMULS.L命令には、乗算器アクセスのためのMAステージがあります。DMULS.L命令の乗算器の動作中（mm）に他のDMULS.LのMAが競合すると、そのMAはmmが終了するまで引き伸ばされ（下図のM――Aの）、一つのスロットを形成します。DMULS.LとDMULS.Lの間に乗算器と無関係な命令が2つ以上はいるとDMULS.LとDMULS.Lの競合によるストールはなくなります。なお、DMULS.LのMAもIFと競合するとスロットがスプリットします。

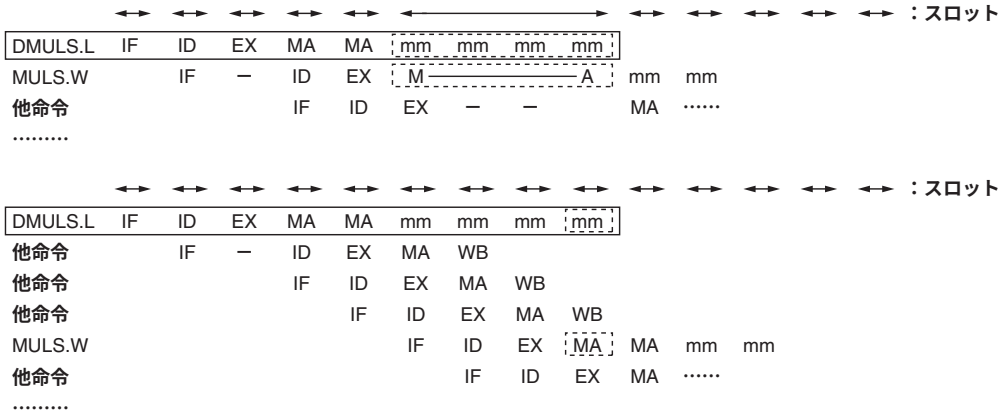


DMULS.L命令のMAがmm終了まで引き伸ばされている場合、このMAとIFが競合すると、通常どおりスロットがスプリットします。下図を参照してください。この図については、MAとIFの競合を考慮して書いてあります。

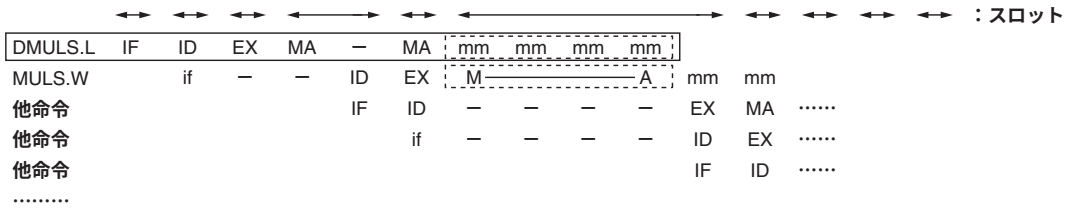


（d）DMULS.L命令の直後に、MULS.W命令が連続してくる場合

MULS.W命令には、乗算器アクセスのためのMAステージがあります。DMULS.L命令の乗算器の動作中（mm）に他のMULS.WのMAが競合すると、そのMAはmmが終了するまで引き伸ばされ（下図のM――Aの）、一つのスロットを形成します。DMULS.LとMULS.Wの間に乗算器と無関係な命令が3つ以上はいるとDMULS.LとDMULS.Lの競合によるストールはなくなります。なお、MULS.WのMAもIFと競合するとスロットがスプリットします。

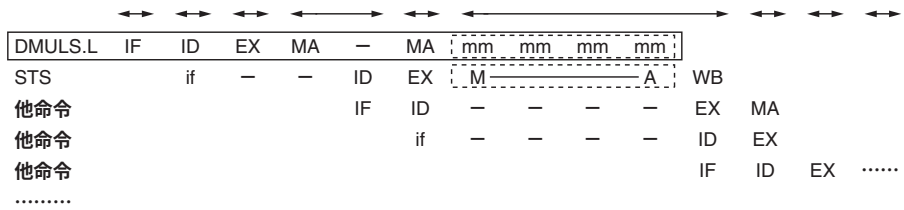


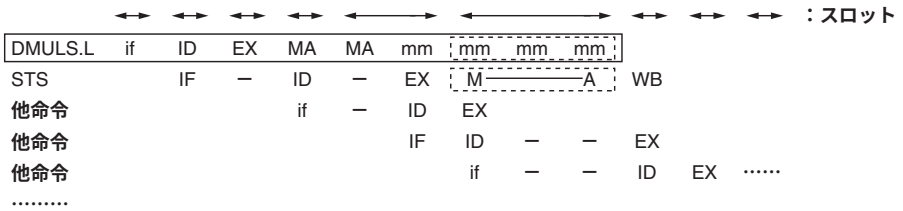
DMULS.L命令のMAがmm終了まで引き伸ばされている場合、このMAとIFが競合すると、通常どおりスロットがスプリットします。下図を参照してください。この図については、MAとIFの競合を考慮して書いてあります。



（e）DMULS.L命令の直後に、STS（レジスタ）命令がくる場合

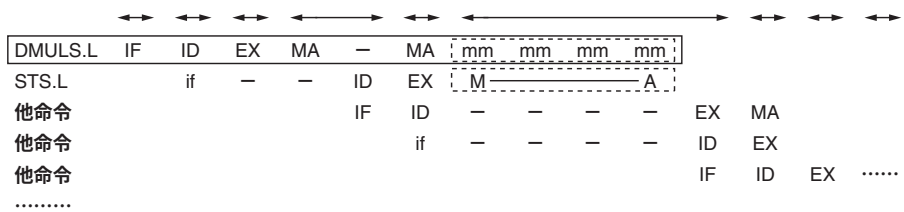
STS命令で、MACレジスタの内容を汎用レジスタにストアする場合、後述のとおり、STS命令には乗算器のアクセスのためのMAステージがはいります。乗算器の動作中（mm）にSTSのMAが競合すると、そのMAはmmが終了するまで引き伸ばされ（下図のM――A）、一つのスロットを形成します。また、このSTSのMAはIFと競合します。その様子を下図に示します。これらの図については、MAとIFの競合を考慮して書いてあります。

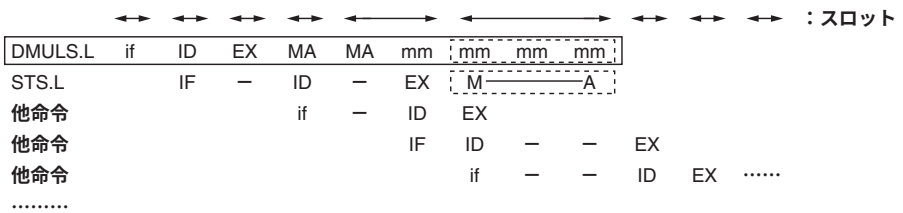




（f）DMULS.L命令の直後に、STS.L（メモリ）命令がくる場合

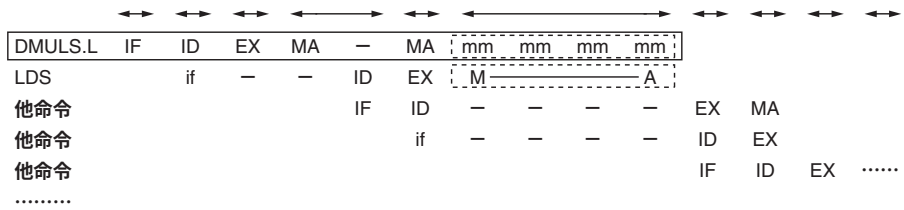
STS命令で、MACレジスタの内容をメモリにストアする場合、後述のとおり、STS命令には乗算器のアクセスと、メモリライトのためのMAステージがはいります。また、このSTSのMAはIFと競合します。その様子を下図に示します。これらの図については、MAとIFの競合を考慮して書いてあります。

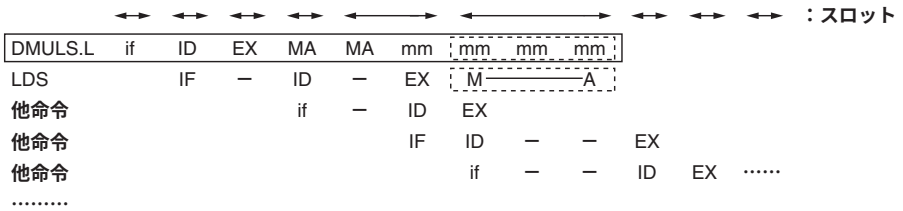




（g）DMULS.L命令の直後に、LDS（レジスタ）命令がくる場合

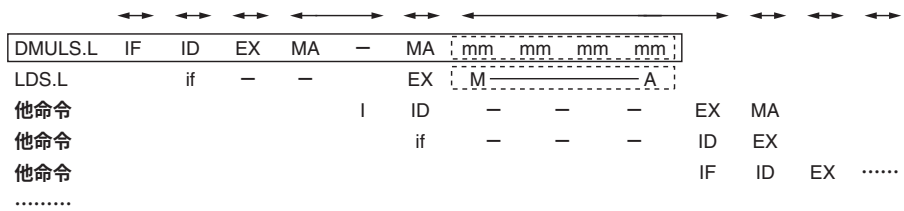
LDS命令で、MACレジスタの内容を汎用レジスタからロードする場合、後述のとおり、LDS命令には乗算器のアクセスのためのMAステージがはいります。乗算器の動作中（mm）にLDSのMAが競合すると、そのMAはmmが終了するまで引き伸ばされ（下図のM――A）、一つのスロットを形成します。また、このLDSのMAはIFと競合します。その様子を下図に示します。これらの図については、MAとIFの競合を考慮して書いてあります。

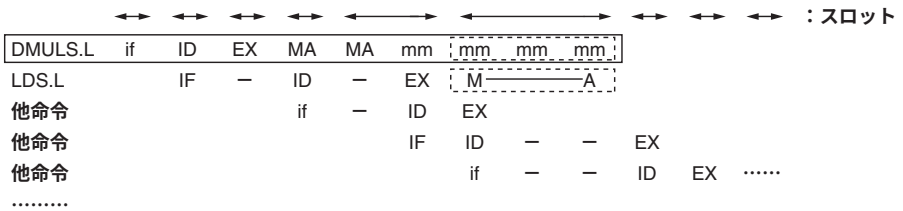




（h）DMULS.L命令の直後に、LDS.L（メモリ）命令がくる場合

LDS命令で、MACレジスタの内容をメモリからロードする場合、後述のとおり、LDS命令にはメモリのアクセスと乗算器のアクセスのためのMAステージがはいります。乗算器の動作中（mm）にLDSのMAが競合すると、そのMAはmmが終了するまで引き伸ばされ（下図のM――A）、一つのスロットを形成します。また、このLDSのMAはIFと競合します。その様子を下図に示します。これらの図については、MAとIFの競合を考慮して書いてあります。





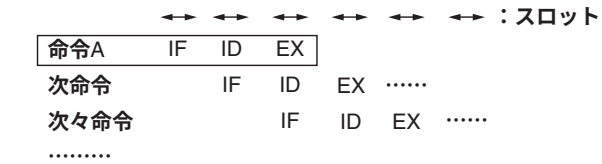
8.8.3 論理演算命令

(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
TST         Rm,Rn
TST         #imm,R0
XOR         Rm,Rn
XOR         #imm,R0
```

●パイプライン



●動作説明

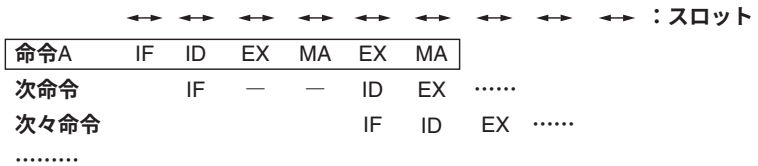
パイプラインは、IF、ID、EX の 3 段で終了します。EX ステージで、ALU を通してデータ演算は完結します。

(2) メモリ論理演算命令

●命令の種類

```
AND.B       #imm,@(R0,GBR)
OR.B        #imm,@(R0,GBR)
TST.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。もちろん、これらの命令の MA は IF と競合します。

(3) TAS 命令

●命令の種類

```
TAS.B       @Rn
```

●パイプライン

	↔	↔	↔	↔	↔	↔	↔	↔	↔ ：スロット
命令A	IF	ID	EX	MA	EX	MA			
次命令		IF	—	—	—	ID	EX	……	
次々命令						IF	ID	EX	……
……									

●動作説明

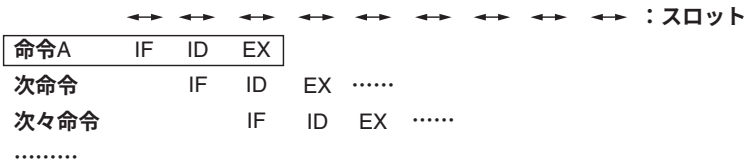
パイプラインは、IF、ID、EX、MA、EX、MA の 6 段で終了します。次命令の ID は 3 スロット分ストールされます。もちろん、TAS 命令の MA は IF と競合します。

8.8.4 シフト命令

●命令の種類

```
ROTL        Rn
ROTR        Rn
ROTCL       Rn
ROTCR       Rn
SHAL        Rn
SHAR        Rn
SHLL        Rn
SHLR        Rn
SHLL2       Rn
SHLR2       Rn
SHLL8       Rn
SHLR8       Rn
SHLL16      Rn
SHLR16      Rn
```

●パイプライン



(a) **●動作説明**

パイプラインは、IF、ID、EX の 3 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。

8.8.5 分岐命令

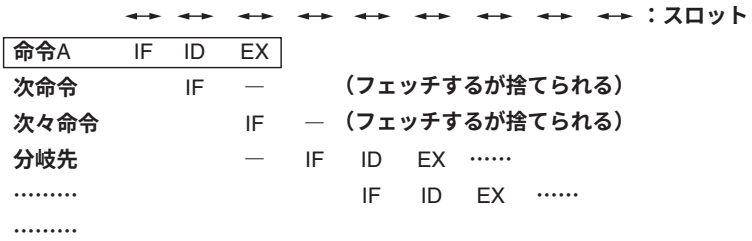
(1) 条件分岐命令

●命令の種類

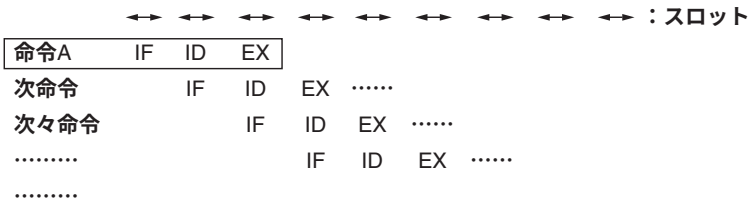
```
BF          label
BT          label
```

●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。

【注】 SH-2E 常にロングワードでフェッチを行います。したがって「（a）条件が成立したとき」の次々命令のフェッチは、その番地が 4n 番地境界にある場合、2 命令分オーバランフェッチします。

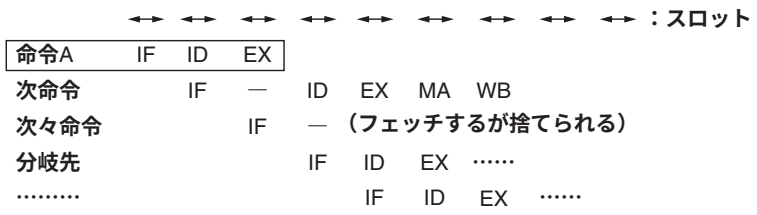
(2) 遅延付き条件分岐命令

●命令の種

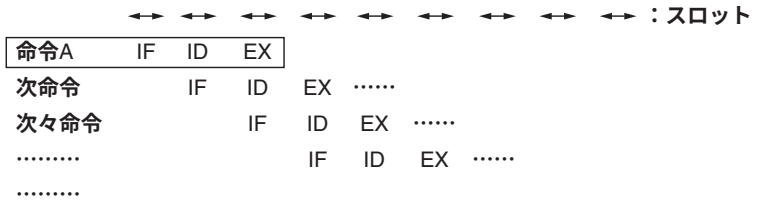
```
BF/S        label
BT/S        label
```

●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。

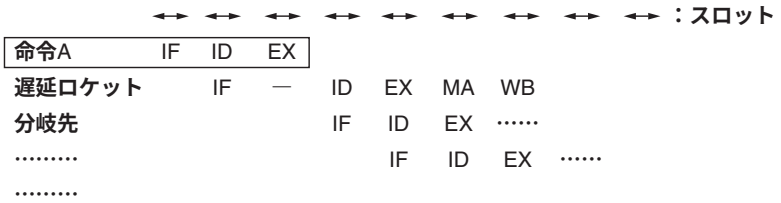
【注】 SH-2E **は常にロングワードでフェッチを行います。したがって「（a）条件が成立したとき」の次々命令のフェッチは、その番地が** 4n **番地境界にある場合、**2 **命令分オーバランフェッチします。**

(3) 無条件分岐命令

●命令の種類

```
BRA         label
BRAF        Rm（SH-2 CPU のみ）
BSR         label
BSRF        Rm（SH-2 CPU のみ）
JMP         @Rm
JSR                         @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、フェッチしてから条件分岐命令のように捨てられず、そのまま実行します。ただし、この遅延スロット命令は ID ステージが 1 スロット分ストールされます。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。

8.8.6 システム制御命令

(1) システム制御 ALU 命令

●命令の種類

```
CLRT
LDC         Rm,SR
LDC         Rm,GBR
LDC         Rm,VBR
LDS         Rm,PR
NOP
SETT
STC         SR,Rn
STC         GBR,Rn
STC         VBR,Rn
STS         PR,Rn
```

●パイプライン

	↔	↔	↔	↔	↔	↔	↔	↔	↔ : スロット
命令A	IF	ID	EX						
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 の 3 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。

(2) LDC.L 命令

●命令の種類

```
LDC.L       @Rm+,SR
LDC.L       @Rm+,GBR
LDC.L       @Rm+,VBR
```

●パイプライン

	↔	↔	↔	↔	↔	↔	↔	↔	↔ : スロット
命令A	IF	ID	EX	MA	EX				
次命令		IF	—	—	ID	EX	……		
次々命令					IF	ID	EX	……	
……									

●動作説明

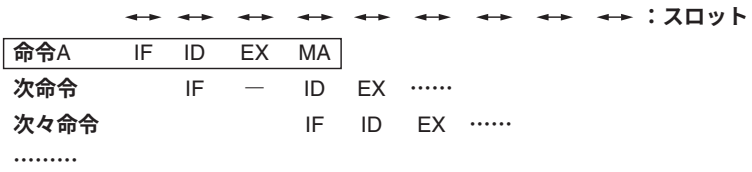
パイプラインは、IF、ID、EX、MA、EX の 5 段で終了します。次命令の ID は 2 スロット分ストールされます。

(3) STC.L 命令

●命令の種類

```
STC.L       SR,@-Rn
STC.L       GBR,@-Rn
STC.L       VBR,@-Rn
```

●パイプライン



●動作説明

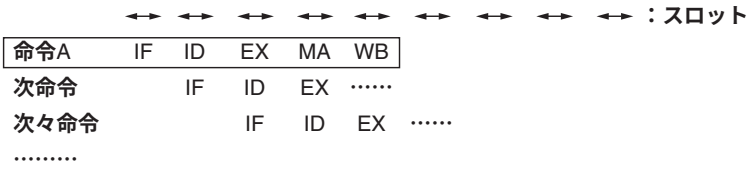
パイプラインは、IF、ID、EX、MA の 4 段で終了します。次命令の ID は 1 スロット分ストールされます。

(4) LDS.L 命令（PR）

●命令の種類

```
LDS.L       @Rm+,PR
```

●パイプライン



●動作説明

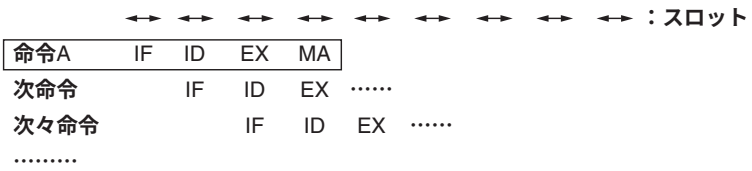
パイプラインは、IF、ID、EX、MA、WB の 5 段で終了します。通常のロード命令と同様です。

(5) STS.L 命令（PR）

●命令の種類

```
STS.L       PR,@-Rn
```

●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。通常のストア命令と同様です。

(6) **レジスタ→MAC 転送命令**

●命令の種類

```
CLRMAC
LDS         Rm,MACH
LDS         Rm,MACL
```

●パイプライン

●動作説明

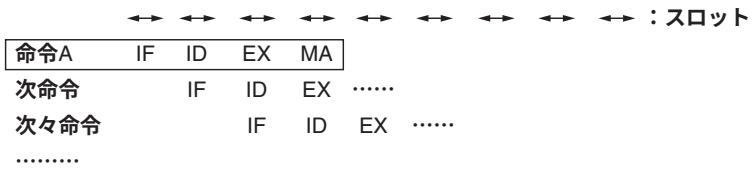
パイプラインは、IF、ID、EX、MA の 4 段で終了します。MA は乗算器アクセスのためのステージです。この MA は IF と競合を起こします。すなわち、通常のストア命令と同様です。ただし、乗算器との競合を起こしますので、積和命令、乗算命令、倍精度乗算命令の項を参照してください。

(7) **メモリ→MAC 転送命令**

●命令の種類

```
LDS.L       @Rm+,MACH
LDS.L       @Rm+,MACL
```

●パイプライン



●動作説明

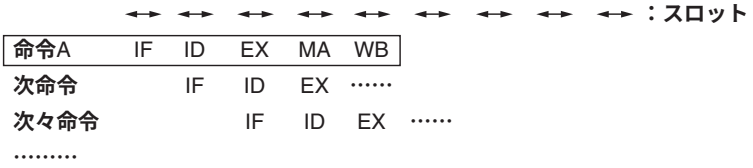
パイプラインは、IF、ID、EX、MA の 4 段で終了します。MA はメモリアクセスと乗算器アクセスのためのステージです。この MA は IF と競合を起こします。すなわち、通常のロード命令と同様です。ただし、乗算器との競合を起こしますので、積和命令、乗算命令、倍精度乗算命令の項を参照してください。

(8) MAC→**レジスタ転送命令**

●命令の種類

```
STS         MACH,Rn
STS         MACL,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 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。すなわち、通常のロード命令と同様です。ただし、乗算器との競合を起こしますので、積和命令、乗算命令、倍精度乗算命令の項を参照してください。

(9) MAC→**メモリ転送命令**

●命令の種類

```
STS.L       MACH,@-Rn
STS.L       MACL,@-Rn
```

●パイプライン

●動作説明

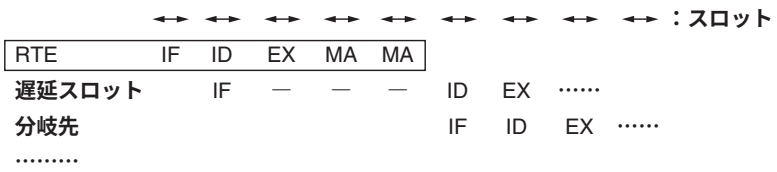
パイプラインは、IF、ID、EX、MA の 4 段で終了します。MA は乗算器アクセスとメモリへのアクセスのためのステージです。この MA は IF と競合を起こします。すなわち、通常のストア命令と同様です。ただし、乗算器との競合を起こしますので、積和命令、乗算命令、倍精度乗算命令の項を参照してください。

(10) RTE 命令

●命令の種類

```
RTE
```

●パイプライン



●動作説明

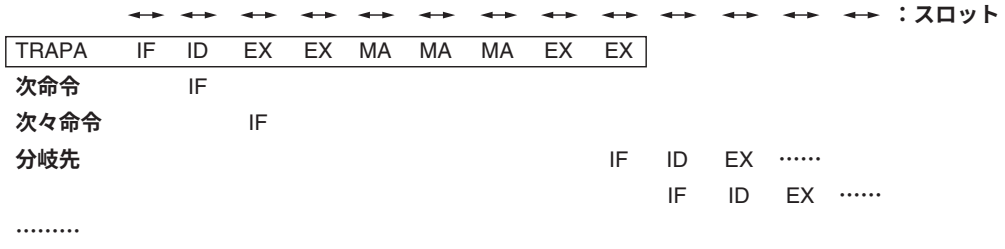
パイプラインは、IF、ID、EX、MA、MA の 5 段で終了します。これらの MA は IF と競合を起こしません。RTE は遅延分岐命令です。遅延スロット命令の ID は 3 スロット分ストールします。分岐先命令の IF は RTE の MA の次のスロットから開始されます。

(11) TRAP 命令

●命令の種類

```
TRAPA       #imm
```

●パイプライン



●動作説明

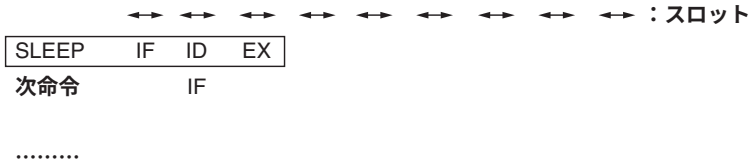
パイプラインは、IF、ID、EX、EX、MA、MA、MA、EX、EX の 9 段で終了します。これらの MA は IF と競合を起こしません。TRAP 命令は遅延分岐命令ではありません。TRAP 命令の次命令と次々命令はフェッチされますが、実行されずに捨てられます。分岐先命令の IF は TRAP 命令の 9 ステージ目の EX のスロットから開始されます。

(12) 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 の 3 段で終了します。次命令の IF までは発行されます。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。

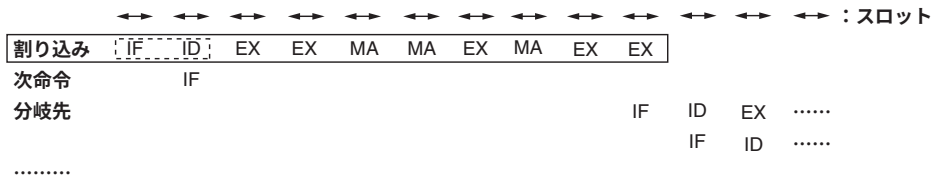
8.8.7 例外処理

(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、MA、MA、EX、MA、EX、EX の 10 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 があるスロットから IF を開始します。

割り込み要因には、NMI などの外部割り込み要求端子、ユーザブレーク、内蔵周辺モジュールによる割り込みがあります。

(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

●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、MA、MA、EX、MA、EX、EX の 10 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。発生要因の詳細については、各ハードウェアマニュアルを参照してください。

(3) 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、MA、MA、MA、EX、EX の 9 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も、オーバランフェッチ（IF）は起こります。IF が次命令だけか、あるいは次々命令でも起こるのかは、実行しようとしていた命令によります。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とスロット不当命令例外処理となります。

また、FPU をモジュールストップ中に FPU 命令および FPU に関する CPU 命令を実行すると、一般不当命令例外処理となります。

(4) FPU 例外処理

●命令の種類

FPU 例外処理

●パイプライン

●動作説明

FPU 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 FPU 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、MA、MA、EX、MA、EX、EX の 10 段で終了します。 FPU 例外処理は遅延分岐ではありません。 FPU 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 FPU 例外処理の最後の EX があるスロットから IF を開始します。

FPU は例外が発生した命令を NOP 化し、さらに例外が発生してから例外を受け付けた命令までの間にある FPU 命令、および FPU に関する CPU 命令を NOP 化します。CPU 命令は NOP 化されず、通常どおりに動作します。

8.8.8 浮動小数点命令および FPU に関する CPU 命令

(1) FPUL ロード命令

●命令の種類

```
LDS         Rm,FPUL
LDS.L       @Rm+,FPUL
```

●パイプライン

	↔	↔	↔	↔	↔	↔	
							：スロット
命令	IF	ID	EX	MA			：CPUパイプライン
	IF	DF	E1	E2	SF		：FPUパイプライン
次命令		IF	ID	EX	……		：CPUパイプライン
		IF	DF	E1	……		：FPUパイプライン（CPU命令の場合なし）
次々命令			IF	ID	EX	……	：CPUパイプライン
			IF	DF	E1	……	：FPUパイプライン（CPU命令の場合なし）
……							

●動作説明

パイプラインは CPU パイプラインが IF、ID、EX、MA の 4 段で、FPU パイプラインが IF、DF、E1、E2、SF の 5 段で終了します。CPU の MA ステージは IF と競合します。この命令の直後に FPUL をリードする命令を置くと競合が発生します。

(2) FPSCR ロード命令

●命令の種類

```
LDS         Rm,FPSCR
LDS.L       @Rm+,FPSCR
```

●パイプライン

	↔	↔	↔	↔	↔	↔	↔	↔	
									：スロット
命令	IF	ID	EX	MA					：CPUパイプライン
	IF	DF	E1	E2	SF				：FPUパイプライン
次命令		IF	ID	—	—	EX	……		：CPUパイプライン
		IF	DF	—	—	E1	……		：FPUパイプライン（CPU命令の場合なし）
次々命令			IF	—	—	ID	EX	……	：CPUパイプライン
			IF	—	—	DF	E1	……	：FPUパイプライン（CPU命令の場合なし）
……									

●動作説明

パイプラインは CPU パイプラインが IF、ID、EX、MA の 4 段で、FPU パイプラインが IF、DF、E1、E2、SF の 5 段で終了します。**図 8.11** に示した競合が発生し、続く命令が 2 スロット期間分実行が遅延します。

(3) FPUL ストア命令（STS）

●命令の種類

```
STS         FPUL,Rn
```

●パイプライン

	←→	←→	←→	←→	←→	←→	：スロット
命令	IF	ID	EX	MA	WB		：CPUパイプライン
	IF	DF	E1	E2			：FPUパイプライン
次命令		IF	ID	EX	……		：CPUパイプライン
		IF	DF	E1	……		：FPUパイプライン（CPU命令の場合なし）
次々命令			IF	ID	EX	……	：CPUパイプライン
			IF	DF	E1	……	：FPUパイプライン（CPU命令の場合なし）
……							

●動作説明

パイプラインは CPU パイプラインが IF、ID、EX、MA、WB の 5 段で、FPU パイプラインが IF、DF、E1、E2 の 4 段で終了します。CPU の MA ステージは IF と競合します。この命令の直後に、この命令のデスティネーションを使う命令を置くと競合が発生します。

(4) FPUL ストア命令（STS.L）

●命令の種類

```
STS.L       FPUL,@-Rn
```

●パイプライン

	←→	←→	←→	←→	←→	←→	：スロット
命令	IF	ID	EX	MA			：CPUパイプライン
	IF	DF	E1	E2			：FPUパイプライン
次命令		IF	ID	EX	……		：CPUパイプライン
		IF	DF	E1	……		：FPUパイプライン（CPU命令の場合なし）
次々命令			IF	ID	EX	……	：CPUパイプライン
			IF	DF	E1	……	：FPUパイプライン（CPU命令の場合なし）
……							

(a) ●動作説明

パイプラインは CPU パイプラインが IF、ID、EX、MA の 4 段で、FPU パイプラインが IF、DF、E1、E2 の 4 段で終了します。CPU の MA ステージは IF と競合します。

(5) FPSCR ストア命令（STS）

●命令の種類

```
STS         FPSCR,Rn
```

●パイプライン

	↔	↔	↔	↔	↔	↔	↔	↔	：スロット
命令	IF	ID	—	—	EX	MA	WB		：CPUパイプライン
	IF	DF	—	—	E1	E2			：FPUパイプライン
次命令		IF	—	—	ID	EX	……		：CPUパイプライン
		IF	—	—	DF	E1	……		：FPUパイプライン（CPU命令の場合なし）
次々命令					IF	ID	EX	……	：CPUパイプライン
					IF	DF	E1	……	：FPUパイプライン（CPU命令の場合なし）
……									

●動作説明

パイプラインは CPU パイプラインが IF、ID、EX、MA、WB の 5 段で、FPU パイプラインが IF、DF、E1、E2 の 4 段で終了します。**図 8.12** に示した競合が発生し、2 スロット期間分実行が遅延します。CPU の MA ステージは IF と競合します。この命令の直後に、この命令のデスティネーションを使う命令を置くと競合が発生します。

(6) FPSCR ストア命令（STS.L）

●命令の種類

```
STS.L       FPSCR,@-Rn
```

●パイプライン

	↔	↔	↔	↔	↔	↔	↔	↔	：スロット
命令	IF	ID	—	—	EX	MA			：CPUパイプライン
	IF	DF	—	—	E1	E2			：FPUパイプライン
次命令		IF	—	—	ID	EX	……		：CPUパイプライン
		IF	—	—	DF	E1	……		：FPUパイプライン（CPU命令の場合なし）
次々命令					IF	ID	EX	……	：CPUパイプライン
					IF	DF	E1	……	：FPUパイプライン（CPU命令の場合なし）
……									

●動作説明

パイプラインは CPU パイプラインが IF、ID、EX、MA の 4 段で、FPU パイプラインが IF、DF、E1、E2 の 4 段で終了します。**図 8.12** に示した競合が発生し、2 スロット期間分実行が遅延します。CPU の MA ステージは IF と競合します。

(7) 浮動小数点レジスタ-レジスタ間転送命令

●命令の種類

```
FLDS        FRm,FPUL
FMOV        FRm,FRn
FSTS        FPUL,FRn
```

●パイプライン

	↔	↔	↔	↔	↔	↔	：スロット
命令	IF	ID	EX				：CPUパイプライン
	IF	DF	E1	E2	SF		：FPUパイプライン
次命令		IF	ID	EX	……		：CPUパイプライン
		IF	DF	E1	……		：FPUパイプライン（CPU命令の場合なし）
次々命令			IF	ID	EX	……	：CPUパイプライン
			IF	DF	E1	……	：FPUパイプライン（CPU命令の場合なし）
……							

●動作説明

パイプラインは CPU パイプラインが IF、ID、EX の 3 段で、FPU パイプラインが IF、DF、E1、E2、SF の 5 段で終了します。この命令の直後に、この命令のデスティネーションをリードする命令を置くと競合が発生します。

(8) 浮動小数点レジスタ-イミディエイト命令

●命令の種類

```
FLDI0       FRn
FMDI1       FRn
```

●パイプライン

	↔	↔	↔	↔	↔	↔	：スロット
命令	IF	ID	EX				：CPUパイプライン
	IF	DF	E1	E2	SF		：FPUパイプライン
次命令		IF	ID	EX	……		：CPUパイプライン
		IF	DF	E1	……		：FPUパイプライン（CPU命令の場合なし）
次々命令			IF	ID	EX	……	：CPUパイプライン
			IF	DF	E1	……	：FPUパイプライン（CPU命令の場合なし）
……							

●動作説明

パイプラインは CPU パイプラインが IF、ID、EX の 3 段で、FPU パイプラインが IF、DF、E1、E2、SF の 5 段で終了します。この命令の直後に、この命令のデスティネーションをリードする命令を置くと競合が発生します。

(9) 浮動小数点レジスタロード命令

●命令の種類

```
FMOV.S      @Rm,FRn
FMOV.S      @Rm+,FRn
FMOV.S      @(R0,Rm),FRn
```

●パイプライン

	↔	↔	↔	↔	↔	↔	：スロット
命令	IF	ID	EX	MA			：CPUパイプライン
	IF	DF	E1	E2	SF		：FPUパイプライン
次命令		IF	ID	EX	……		：CPUパイプライン
		IF	DF	E1	……		：FPUパイプライン（CPU命令の場合なし）
次々命令			IF	ID	EX	……	：CPUパイプライン
			IF	DF	E1	……	：FPUパイプライン（CPU命令の場合なし）
……							

●動作説明

パイプラインは CPU パイプラインが IF、ID、EX、MA の 4 段で、FPU パイプラインが IF、DF、E1、E2、SF の 5 段で終了します。CPU の MA ステージは IF と競合します。この命令の直後に、この命令のデスティネーションをリードする命令を置くと競合が発生します。

(10) 浮動小数点レジスタストア命令

●命令の種類

```
FMOV.S      FRm,@Rn
FMOV.S      FRm,@-Rn
FMOV.S      FRm,@(R0,Rn)
```

●パイプライン

	↔	↔	↔	↔	↔	↔	：スロット
命令	IF	ID	EX	MA			：CPUパイプライン
	IF	DF	E1	E2			：FPUパイプライン
次命令		IF	ID	EX	……		：CPUパイプライン
		IF	DF	E1	……		：FPUパイプライン（CPU命令の場合なし）
次々命令			IF	ID	EX	……	：CPUパイプライン
			IF	DF	E1	……	：FPUパイプライン（CPU命令の場合なし）
……							

●動作説明

パイプラインは CPU パイプラインが IF、ID、EX、MA の 4 段で、FPU パイプラインが IF、DF、E1、E2 の 4 段で終了します。CPU の MA ステージは IF と競合します。

(11) 浮動小数点演算命令（FDIV 以外）

●命令の種類

```
FABS        FRn
FADD        FRm,FRn
FLOAT       FPUL,FRn
FMAC        FR0,FRm,FRn
FMUL        FRm,FRn
FNEG        FRn
FSUB        FRm,FRn
FTRC        FRm,FPUL
```

●パイプライン

	↔	↔	↔	↔	↔	↔	：スロット
命令	IF	ID	EX				：CPUパイプライン
	IF	DF	E1	E2	SF		：FPUパイプライン
次命令		IF	ID	EX	……		：CPUパイプライン
		IF	DF	E1	……		：FPUパイプライン（CPU命令の場合なし）
次々命令			IF	ID	EX	……	：CPUパイプライン
			IF	DF	E1	……	：FPUパイプライン（CPU命令の場合なし）
……							

●動作説明

パイプラインは CPU パイプラインが IF、ID、EX の 3 段で、FPU パイプラインが IF、DF、E1、E2、SF の 5 段で終了します。この命令の直後に、この命令のデスティネーションをリードする命令を置くと競合が発生します。

(12) 浮動小数点演算命令（FDIV）

●命令の種類

```
FDIV        FRm,FRn
```

●パイプライン

ケース1：次命令が浮動小数点命令もしくはFPUに関するCPU命令のとき

	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	スロット
命令	IF	ID	EX		·····							：CPUパイプライン
	IF	DF	E1	E1	·····	E1	E2	SF				：FPUパイプライン
次命令		IF	ID	ー	·····	ー	ー	ー	EX	·····		：CPUパイプライン
		IF	DF	ー	·····	ー	ー	ー	E1	·····		：FPUパイプライン
次々命令			IF	ー	·····	ー	ー	ー	ID	EX	·····	：CPUパイプライン
			IF	ー	·····	ー	ー	ー	DF	E1	·····	：FPUパイプライン
·····												（CPU命令の場合なし）

ケース2：次命令がCPU命令で、次々命令が浮動小数点命令もしくはFPUに関するCPU命令のとき

	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	スロット
命令	IF	ID	EX									：CPUパイプライン
	IF	DF	E1	E1	·····	E1	E2	SF				：FPUパイプライン
次命令		IF	ID	EX	·····							：CPUパイプライン
次々命令			IF	ID	·····	ー	ー	ー	EX	·····		：CPUパイプライン
			IF	DF	·····	ー	ー	ー	E1	·····		：FPUパイプライン
·····												

●動作説明

パイプラインは CPU パイプラインが IF、ID、EX の 3 段で、FPU パイプラインが IF、DF、E1、E1、E1、E1、E1、E1、E1、E1、E1、E1、E1、E1、E1、E2、SF の 17 段で終了します。すなわち、E1 ステージを 13 回繰り返します。

図 8.13 に示した競合が発生します。FDIV のパイプライン中に浮動小数点命令もしくは FPU に関する CPU 命令のパイプが重なると、E1 以降のステージが FDIV 実行終了後までストールされ、以降の命令もストールを受けます。したがって、FDIV 命令の直後から 14 命令以内は浮動小数点命令もしくは FPU に関する CPU 命令を配置しないとその期間に CPU 命令が実行でき、性能が向上します。

(13) 浮動小数点比較命令

●命令の種類

```
FCMP/EQ     FRm,FRn
FCMP/GT     FRm,FRn
```

●パイプライン

	↔	↔	↔	↔	↔	↔	
命令	IF	ID	EX				**：CPUパイプライン**
	IF	DF	E1				**：FPUパイプライン**
次命令		IF	ID	EX	……		**：CPUパイプライン**
		IF	DF	E1	……		**：FPUパイプライン（CPU命令の場合なし）**
次々命令			IF	ID	EX	……	**：CPUパイプライン**
			IF	DF	E1	……	**：FPUパイプライン（CPU命令の場合なし）**
……							

●動作説明

パイプラインは CPU パイプラインが IF、ID、EX の 3 段で、FPU パイプラインが IF、DF、E1 の 3 段で終了します

付　録

A.　命令コード

詳細については「**第 7 章　各命令の説明**」を参照してください。

A.1　アドレッシングモード別命令セット

命令の命令コードと実行ステートを、アドレッシングモード別に示します。

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	8
レジスタ直接	**デスティネーションオペランドのみ**	MOVT　Rn	22
	ソースとデスティネーションオペランド	ADD　Rm,Rn	42
	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とストア	LDC　Rm,SR STS　MACH,Rn	18
レジスタ間接	**ソースオペランドのみ**	JMP　@Rm	2
	デスティネーションオペランドのみ	TAS.B　@Rn	1
	レジスタ直接とのデータ転送	MOV.L　Rm,@Rn	8
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
	レジスタ直接からのデータ転送	MOV.L　@Rm+,Rn	4
	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	8
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
	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	8
ディスプレ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
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
ディスプレースメント付き GBR 間接	**レジスタ直接とのデータ転送**	MOV.L　R0,@(disp,GBR)	6
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
ディスプレースメント付き PC 相対	**レジスタ直接へのデータ転送**	MOV.L　@(disp,PC),Rn	3
Rm を用いた PC 相対	**分岐命令**	BRAF　Rm	2
PC 相対	**分岐命令**	BRA　label	6

アドレッシングモード	区分	命令の例	種類
イミディエイト	レジスタへのロード	FLDI0　FRn	2
	レジスタ直接との算術論理演算	ADD　#imm,Rn	7
	例外処理ベクタの指定	TRAPA　#imm	1
			計　172

(1)　**オペランドなし**

表 A.2　**オペランドなし**

命令	命令コード	動作	実行ステート	T ビット
CLRT	0000000000001000	0→T	1	0
CLRMAC	0000000000101000	0→MACH,MACL	1	—
DIV0U	0000000000011001	0→M/Q/T	1	0
NOP	0000000000001001	**無操作**	1	—
RTE	0000000000101011	**遅延分岐、スタック領域**→PC/SR	4	LSB
RTS	0000000000001011	**遅延分岐、**PR→PC	2	—
SETT	0000000000011000	1→T	1	1
SLEEP	0000000000011011	**スリープ**	3	—

(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		命令コード	動作	実行ステート	T ビット
CMP/PL	Rn	0100nnnn00010101	Rn>0 **とき** 1→T	1	**比較結果**
CMP/PZ	Rn	0100nnnn00010001	Rn≧0 **とき** 1→T	1	**比較結果**
DT	Rn	0100nnnn00010000	Rn-1→Rn, Rn **が** 0 **のとき** 1→T Rn **が** 0 **以外のとき** 0→T	1	**比較結果**
FABS	FRn	1111nnnn01011101	abs(FRn)→FRn	1	—
FLOAT	FPUL, FRn	1111nnnn00101101	(float)FPUL→FRn	1	—
FNEG	FRn	1111nnnn01001101	-1.0×FRn→FRn	1	—
FTRC	FRm, FPUL	1111mmmm00111101	(int)FRm→FPUL	1	—
MOVT	Rn	0000nnnn00101001	T→Rn	1	—
ROTL	Rn	0100nnnn00000100	T←Rn←MSB	1	MSB
ROTR	Rn	0100nnnn00000101	LSB→Rn→T	1	LSB
ROTCL	Rn	0100nnnn00100100	T←Rn←T	1	MSB
ROTCR	Rn	0100nnnn00100101	T→Rn→T	1	LSB
SHAL	Rn	0100nnnn00100000	T←Rn←0	1	MSB
SHAR	Rn	0100nnnn00100001	MSB→Rn→T	1	LSB
SHLL	Rn	0100nnnn00000000	T←Rn←0	1	MSB
SHLR	Rn	0100nnnn00000001	0→Rn→T	1	LSB
SHLL2	Rn	0100nnnn00001000	Rn<<2→Rn	1	—
SHLR2	Rn	0100nnnn00001001	Rn>>2→Rn	1	—

命令		命令コード	動作	実行ステート	T ビット
SHLL8	Rn	0100nnnn00011000	Rn<<8→Rn	1	—
SHLR8	Rn	0100nnnn00011001	Rn>>8→Rn	1	—
SHLL16	Rn	0100nnnn00101000	Rn<<16→Rn	1	—
SHLR16	Rn	0100nnnn00101001	Rn>>16→Rn	1	—

表 A.4　ソースとデスティネーションオペランド

命令		命令コード	動作	実行ステート	T ビット
ADD	Rm,Rn	0011nnnnmmmm1100	Rn+Rm→Rn	1	—
ADDC	Rm,Rn	0011nnnnmmmm1110	Rn+Rm+T→Rn,**キャリ**→T	1	**キャリ**
ADDV	Rm,Rn	0011nnnnmmmm1111	Rn+Rm→Rn, **オーバフロー**→T	1	**オーバフロー**
AND	Rm,Rn	0010nnnnmmmm1001	Rn & Rm→Rn	1	—
CMP/EQ	Rm,Rn	0011nnnnmmmm0000	Rn=Rm **のとき** 1→T	1	**比較結果**
CMP/HS	Rm,Rn	0011nnnnmmmm0010	**無符号で** Rn≧Rm **のとき** 1→T	1	**比較結果**
CMP/GE	Rm,Rn	0011nnnnmmmm0011	**有符号で** Rn≧Rm **のとき** 1→T	1	**比較結果**
CMP/HI	Rm,Rn	0011nnnnmmmm0110	**無符号で** Rn>Rm **のとき** 1→T	1	**比較結果**
CMP/GT	Rm,Rn	0011nnnnmmmm0111	**有符号で** Rn>Rm **のとき** 1→T	1	**比較結果**
CMP/STR	Rm,Rn	0010nnnnmmmm1100	**いずれかのバイトが等しいとき** 1→T	1	**比較結果**
DIV1	Rm,Rn	0011nnnnmmmm0100	1 **ステップ除算**(Rn÷Rm)	1	**計算結果**
DIV0S	Rm,Rn	0010nnnnmmmm0111	Rn **の** MSB→Q, Rm **の** MSB→M, M ^ Q→T	1	**計算結果**
DMULS.L	Rm,Rn	0011nnnnmmmm1101	**符号付きで** Rn×Rm→MACH, MACL	2〜4*1	—
DMULU.L	Rm,Rn	0011nnnnmmmm0101	**符号なしで** Rn×Rm→MACH, MACL	2〜4*1	—
EXTS.B	Rm,Rn	0110nnnnmmmm1110	Rm **をバイトから符号拡張**→Rn	1	—
EXTS.W	Rm,Rn	0110nnnnmmmm1111	Rm **をワードから符号拡張**→Rn	1	—
EXTU.B	Rm,Rn	0110nnnnmmmm1100	Rm **をバイトからゼロ拡張**→Rn	1	—
EXTU.W	Rm,Rn	0110nnnnmmmm1101	Rm **をワードからゼロ拡張**→Rn	1	—
FADD	FRm,FRn	1111nnnnmmmm0000	FRm+FRn→FRn	1	—
FCMP/EQ	FRm,FRn	1111nnnnmmmm0100	(FRn=FRm)?1:0→T	1	**比較結果**
FCMP/GT	FRm,FRn	1111nnnnmmmm0101	(Rn>FRm)?1:0→T	1	**比較結果**
FDIV	FRm,FRn	1111nnnnmmmm0011	FRn/FRm→FRn	13	—
FMAC	FR0,FRm FRn	1111nnnnmmmm1110	(FR0×FRm)+FRn→FRn	1	—
FMOV	FRm,FRn	1111nnnnmmmm1100	FRm→FRn	1	—
FMUL	FRm,FRn	1111nnnnmmmm0010	FRn×FRm→FRn	1	—
FSUB	FRm,FRn	1111nnnnmmmm0001	FRn-FRm→FRn	1	—
MOV	Rm,Rn	0110nnnnmmmm0011	Rm→Rn	1	—
MUL.L	Rm,Rn	0000nnnnmmmm0111	Rn×Rm→MACL	2〜4*1	—
MULS.W	Rm,Rn	0010nnnnmmmm1111	**符号付きで** Rn×Rm→MAC	1〜3*1	—

命令		命令コード	動作	実行ステート	T ビット
MULU.W	Rm,Rn	0010nnnnmmmm1110	符号なしで Rn×Rm→MAC	1〜3*1	—
NEG	Rm,Rn	0110nnnnmmmm1011	0-Rm→Rn	1	—
NEGC	Rm,Rn	0110nnnnmmmm1010	0-Rm-T→Rn, ボロー→T	1	ボロー
NOT	Rm,Rn	0110nnnnmmmm0111	~Rm→Rn	1	—
OR	Rm,Rn	0010nnnnmmmm1011	Rn | Rm→Rn	1	—
SUB	Rm,Rn	0011nnnnmmmm1000	Rn-Rm→Rn	1	—
SUBC	Rm,Rn	0011nnnnmmmm1010	Rn-Rm-T→Rn, ボロー→T	1	ボロー
SUBV	Rm,Rn	0011nnnnmmmm1011	Rn-Rm→Rn, アンダフロー→T	1	アンダフロー
SWAP.B	Rm,Rn	0110nnnnmmmm1000	Rm→下位 2 バイトの上下バイト交換→Rn	1	—
SWAP.W	Rm,Rn	0110nnnnmmmm1001	Rm→上下ワード交換→Rn	1	—
TST	Rm,Rn	0010nnnnmmmm1000	Rn & Rm, 結果が 0 のとき 1→T	1	テスト結果
XOR	Rm,Rn	0010nnnnmmmm1010	Rn ^ Rm→Rn	1	—
XTRCT	Rm,Rn	0010nnnnmmmm1101	Rm と Rn の中央 32 ビット→Rn	1	—

【注】 *1　通常実行ステートを示します。

表 A.5　コントロールレジスタまたはシステムレジスタとのロードとストア

命令		命令コード	動作	実行ステート	T ビット
FLDS	FRm, FPUL	1111mmmm00011101	FRm→FPUL	1	—
FSTS	FPUL,FRn	1111nnnn00001101	FPUL→FRn	1	—
LDC	Rm,SR	0100mmmm00001110	Rm→SR	1	LSB
LDC	Rm,GBR	0100mmmm00011110	Rm→GBR	1	—
LDC	Rm,VBR	0100mmmm00101110	Rm→VBR	1	—
LDS	Rm, FPSCR	0100mmmm01101010	Rm→FPSCR	1	—
LDS	Rm, FPUL	0100mmmm01011010	Rm→FPUL	1	—
LDS	Rm,MACH	0100mmmm00001010	Rm→MACH	1	—
LDS	Rm,MACL	0100mmmm00011010	Rm→MACL	1	—
LDS	Rm,PR	0100mmmm00101010	Rm→PR	1	—
STC	SR,Rn	0000nnnn00000010	SR→Rn	1	—
STC	GBR,Rn	0000nnnn00010010	GBR→Rn	1	—
STC	VBR,Rn	0000nnnn00100010	VBR→Rn	1	—
STS	FPSCR, Rn	0000nnnn01101010	FPSCR→Rn	1	—
STS	FPUL, Rn	0000nnnn01011010	FPUL→Rn	1	—
STS	MACH,Rn	0000nnnn00001010	MACH→Rn	1	—
STS	MACL,Rn	0000nnnn00011010	MACL→Rn	1	—
STS	PR,Rn	0000nnnn00101010	PR→Rn	1	—

(3)　レジスタ間接

表 A.6　ソースオペランドのみ

命令		命令コード	動作	実行ステート	T ビット
JMP	@Rm	0100mmmm00101011	遅延分岐、Rm→PC	2	—
JSR	@Rm	0100mmmm00001011	遅延分岐、PC→PR, Rm→PC	2	—

表 A.7　デスティネーションオペランドのみ

命令		命令コード	動作	実行ステート	T ビット
TAS.B	@Rn	0100nnnn00011011	(Rn)が 0 のとき 1→T, 1→MSB of (Rn)	4	テスト結果

表 A.8　レジスタ直接とのデータ転送

命令		命令コード	動作	実行ステート	T ビット
FMOV.S	FRm, @Rn	1111nnnnmmmm1010	FRm→(Rn)	1	—
FMOV.S	@Rm, FRn	1111nnnnmmmm1000	(Rm)→FRn	1	—
MOV.B	Rm,@Rn	0010nnnnmmmm0000	Rm→(Rn)	1	—
MOV.W	Rm,@Rn	0010nnnnmmmm0001	Rm→(Rn)	1	—
MOV.L	Rm,@Rn	0010nnnnmmmm0010	Rm→(Rn)	1	—
MOV.B	@Rm,Rn	0110nnnnmmmm0000	(Rm)→符号拡張→Rn	1	—
MOV.W	@Rm,Rn	0110nnnnmmmm0001	(Rm)→符号拡張→Rn	1	—
MOV.L	@Rm,Rn	0110nnnnmmmm0010	(Rm)→Rn	1	—

(4)　ポストインクリメントレジスタ間接

表 A.9　積和演算

命令		命令コード	動作	実行ステート	T ビット
MAC.L	@Rm+,@Rn+	0000nnnnmmmm1111	符号付きで(Rn)×(Rm)+MAC→MAC	3/ (2〜4)*1	—
MAC.W	@Rm+,@Rn+	0100nnnnmmmm1111	符号付きで(Rn)×(Rm)+MAC→MAC	3/(2)*1	—

【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。

表 A.10　レジスタ直接からのデータ転送

命令		命令コード	動作	実行ステート	T ビット
FMOV.S	@Rm+,FRn	1111nnnnmmmm1001	(Rm)→FRn, Rm+4→Rm	1	—
MOV.B	@Rm+,Rn	0110nnnnmmmm0100	(Rm)→符号拡張→Rn, Rm+1→Rm	1	—
MOV.W	@Rm+,Rn	0110nnnnmmmm0101	(Rm)→符号拡張→Rn, Rm+2→Rm	1	—
MOV.L	@Rm+,Rn	0110nnnnmmmm0110	(Rm)→Rn, Rm+4→Rm	1	—

表 A.11　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		命令コード	動作	実行ステート	T ビット
LDC.L	@Rm+,SR	0100mmmm00000111	(Rm)→SR, Rm+4→Rm	3	LSB
LDC.L	@Rm+,GBR	0100mmmm00010111	(Rm)→GBR, Rm+4→Rm	3	—
LDC.L	@Rm+,VBR	0100mmmm00100111	(Rm)→VBR, Rm+4→Rm	3	—
LDS.L	@Rm+,FPSCR	0100mmmm01100110	(Rm)→FPSCR, Rm+4→Rm	1	—
LDS.L	@Rm+, FPUL	0100mmmm01010110	(Rm)→FPUL, Rm+4→Rm	1	—
LDS.L	@Rm+,MACH	0100mmmm00000110	(Rm)→MACH, Rm+4→Rm	1	—
LDS.L	@Rm+,MACL	0100mmmm00010110	(Rm)→MACL, Rm+4→Rm	1	—
LDS.L	@Rm+,PR	0100mmmm00100110	(Rm)→PR, Rm+4→Rm	1	—

(5)　**プリデクリメントレジスタ間接**

表 A.12　レジスタ直接からのデータ転送

命令		命令コード	動作	実行ステート	T ビット
FMOV.S	FRm,@-Rn	1111nnnnmmmm1011	Rn-4→Rn, FRm→(Rn)	1	—
MOV.B	Rm,@-Rn	0010nnnnmmmm0100	Rn-1→Rn, Rm→(Rn)	1	—
MOV.W	Rm,@-Rn	0010nnnnmmmm0101	Rn-2→Rn, Rm→(Rn)	1	—
MOV.L	Rm,@-Rn	0010nnnnmmmm0110	Rn-4→Rn, Rm→(Rn)	1	—

表 A.13　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		命令コード	動作	実行ステート	T ビット
STC.L	SR,@-Rn	0100nnnn00000011	Rn-4→Rn, SR→(Rn)	2	—
STC.L	GBR,@-Rn	0100nnnn00010011	Rn-4→Rn, GBR→(Rn)	2	—
STC.L	VBR,@-Rn	0100nnnn00100011	Rn-4→Rn, VBR→(Rn)	2	—
STS.L	FPSCR,@-Rn	0100nnnn01100010	Rn-4→Rn, FPSCR→(Rn)	1	—
STS.L	FPUL,@-Rn	0100nnnn01010010	Rn-4→Rn, FPUL→(Rn)	1	—
STS.L	MACH,@-Rn	0100nnnn00000010	Rn-4→Rn, MACH→(Rn)	1	—
STS.L	MACL,@-Rn	0100nnnn00010010	Rn-4→Rn, MACL→(Rn)	1	—
STS.L	PR,@-Rn	0100nnnn00100010	Rn-4→Rn, PR→(Rn)	1	—

(6)　ディスプレースメント付きレジスタ間接

表 A.14　**ディスプレースメント付きレジスタ間接**

命令	命令コード	動作	実行ステート	T ビット
MOV.B R0,@(disp,Rn)	10000000nnnndddd	R0→(disp+Rn)	1	—
MOV.W R0,@(disp,Rn)	10000001nnnndddd	R0→(disp×2+Rn)	1	—
MOV.L Rm,@(disp,Rn)	0001nnnnmmmmdddd	Rm→(disp×4+Rn)	1	—
MOV.B @(disp,Rm),R0	10000100mmmmdddd	(disp+Rm)→**符号拡張**→R0	1	—
MOV.W @(disp,Rm),R0	10000101mmmmdddd	(disp×2+Rm)→**符号拡張**→R0	1	—
MOV.L @(disp,Rm),Rn	0101nnnnmmmmdddd	(disp×4+Rm)→Rn	1	—

(7)　インデックス付きレジスタ間接

表 A.15　**インデックス付きレジスタ間接**

命令	命令コード	動作	実行ステート	T ビット
MOV.B Rm,@(R0,Rn)	0000nnnnmmmm0100	Rm→(R0+Rn)	1	—
MOV.W Rm,@(R0,Rn)	0000nnnnmmmm0101	Rm→(R0+Rn)	1	—
MOV.L Rm,@(R0,Rn)	0000nnnnmmmm0110	Rm→(R0+Rn)	1	—
FMOV.S FRm,@(R0,Rn)	1111nnnnmmmm0111	FRm→(R0+Rn)	1	—
MOV.B @(R0,Rm),Rn	0000nnnnmmmm1100	(R0+Rm)→**符号拡張**→Rn	1	—
MOV.W @(R0,Rm),Rn	0000nnnnmmmm1101	(R0+Rm)→**符号拡張**→Rn	1	—
MOV.L @(R0,Rm),Rn	0000nnnnmmmm1110	(R0+Rm)→Rn	1	—
FMOV.S @(R0,FRm),FRn	1111nnnnmmmm0110	(R0+Rn)→FRn	1	—

(8)　ディスプレースメント付き GBR 間接

表 A.16　**ディスプレースメント付き** GBR **間接**

命令	命令コード	動作	実行ステート	T ビット
MOV.B R0,@(disp,GBR)	11000000dddddddd	R0→(disp+GBR)	1	—
MOV.W R0,@(disp,GBR)	11000001dddddddd	R0→(disp×2+GBR)	1	—
MOV.L R0,@(disp,GBR)	11000010dddddddd	R0→(disp×4+GBR)	1	—
MOV.B @(disp,GBR),R0	11000100dddddddd	(disp+GBR)→**符号拡張**→R0	1	—
MOV.W @(disp,GBR),R0	11000101dddddddd	(disp×2+GBR)→**符号拡張**→R0	1	—
MOV.L @(disp,GBR),R0	11000110dddddddd	(disp×4+GBR)→R0	1	—

(9)　インデックス付き GBR 間接

表 A.17　インデックス付き GBR 間接

命令		命令コード	動作	実行ステート	T ビット
AND.B	#imm,@(R0,GBR)	11001101iiiiiiii	(R0+GBR) & imm→(R0+GBR)	3	—
OR.B	#imm,@(R0,GBR)	11001111iiiiiiii	(R0+GBR) | imm→(R0+GBR)	3	—
TST.B	#imm,@(R0,GBR)	11001100iiiiiiii	(R0+GBR) & imm, 結果が 0 のとき 1→T	3	テスト結果
XOR.B	#imm,@(R0,GBR)	11001110iiiiiiii	(R0+GBR) ^ imm→(R0+GBR)	3	—

(10)　ディスプレースメント付き PC 相対

表 A.18　ディスプレースメント付き PC 相対

命令		命令コード	動作	実行ステート	T ビット
MOV.W	@(disp,PC),Rn	1001nnnndddddddd	(disp×2+PC)→符号拡張→Rn	1	—
MOV.L,Rn	@(disp,PC)	1101nnnndddddddd	(disp×4+PC)→Rn	1	—
MOVA	@(disp,PC),R0	11000111dddddddd	disp×4+PC→R0	1	—

(11)　Rm を用いた PC 相対

表 A.19　Rm を用いた PC 相対

命令		命令コード	動作	実行ステート	T ビット
BRA	FRm	0000mmmm00100011	遅延分岐、Rm+PC→PC	2	—
BSRF	Rm	0000mmmm00000011	遅延分岐、PC→PR, Rm+PC→PC	2	—

(12)　PC 相対

表 A.20　PC 相対

命令		命令コード	動作	実行ステート	T ビット
BF	label	10001011dddddddd	T=0 のとき disp×2+PC→PC, T=1 のとき nop	3/1*2	—
BF/S	label	10001111dddddddd	T=0 のとき disp×2+PC→PC, T=1 のとき nop	2/1*2	—
BT	label	10001001dddddddd	T=1 のとき disp×2+PC→PC, T=0 のとき nop	3/1*2	—
BT/S	label	10001101dddddddd	T=1 のとき disp×2+PC→PC, T=0 のとき nop	2/1*2	—
BRA	label	1010dddddddddddd	遅延分岐、disp×2+PC→PC	2	—
BSR	label	1011dddddddddddd	遅延分岐、PC→PR, disp×2+PC→PC	2	—

【注】 *2　分岐しないときは 1 ステートになります。

(13) イミディエイト

表 A.21　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	1111nnnn10001101	0×00000000→FRn	1	—
FLDI1	FRn	1111nnnn10011101	0×3F800000→FRn	1	—

表 A.22　レジスタ直接との算術論理演算

命令		命令コード	動作	実行ステート	T ビット
ADD	#imm,Rn	0111nnnniiiiiiii	Rn+imm→Rn	1	—
AND	#imm,R0	11001001iiiiiiii	R0 & imm→R0	1	—
CMP/EQ	#imm,R0	10001000iiiiiiii	R0=imm **のとき** 1→T	1	**比較結果**
MOV	#imm,Rn	1110nnnniiiiiiii	imm→**符号拡張**→Rn	1	—
OR	#imm,R0	11001011iiiiiiii	R0 | imm→R0	1	—
TST	#imm,R0	11001000iiiiiiii	R0 & imm, **結果が** 0 **のとき** 1→T	1	**テスト結果**
XOR	#imm,R0	11001010iiiiiiii	R0 ^ imm→R0	1	—

表 A.23　例外処理ベクタの指定

命令		命令コード	動作	実行ステート	T ビット
TRAPA	#imm	11000011iiiiiiii	PC/SR→**スタック領域**, (imm×4+VBR)→PC	8	—

A.2　命令形式別命令セット

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.24　**命令形式別命令セット**

命令形式	区分	命令の例	種類
0 形式	—	NOP	8
n 形式	レジスタ直接	MOVT Rn	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	8
	レジスタ間接	TAS.B @Rn	1
	プリデクリメントレジスタ間接	STC.L SR,@-Rn	8
	浮動小数点命令	FABS FRn	6
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	8
	Rm を用いた PC 相対	BRAF Rm	2
	レジスタ間接	JMP @Rm	2
	ポストインクリメントレジスタ間接	LDC.L @Rm+,SR	8
	浮動小数点命令	FLDS FRm,FPUL	2
nm 形式	レジスタ直接	ADD Rm,Rn	34
	レジスタ間接	MOV.L Rm,@Rn	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
	ポストインクリメントレジスタ間接	MOV.L @Rm+,Rn	3
	プリデクリメントレジスタ間接	MOV.L Rm,@-Rn	3
	インデックス付きレジスタ間接	MOV.L Rm,@(R0,Rn)	6
	浮動小数点命令	FADD FRm,FRn	14
md 形式	ディスプレースメント付きレジスタ間接	MOV.B @(disp,Rm),R0	2
nd4 形式	ディスプレースメント付きレジスタ間接	MOV.B R0,@(disp,Rn)	2
nmd 形式	ディスプレースメント付きレジスタ間接	MOV.L Rm,@(disp,Rn)	2
d 形式	ディスプレースメント付き GBR 間接	MOV.L R0,@(disp,GBR)	6
	ディスプレースメント付き PC 相対	MOVA @(disp,PC),R0	1
	PC 相対	BF label	4
d12 形式	PC 相対	BRA label	2
nd8 形式	ディスプレースメント付き PC 相対	MOV.L @(disp,PC),Rn	2
i 形式	インデックス付き GBR 間接	AND.B #imm,@(R0,GBR)	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
	イミディエイト（例外処理ベクタの指定）	TRAPA #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
			計 172

(1)　0 形式

表 A.25　0 形式

命令	命令コード	動作	実行ステート	T ビット
CLRT	0000000000001000	0→T	1	0
CLRMAC	0000000000101000	0→MACH, MACL	1	—
DIV0U	0000000000011001	0→M/Q/T	1	0
NOP	0000000000001001	無操作	1	—
RTE	0000000000101011	遅延分岐、 スタック領域→PC/SR	4	LSB
RTS	0000000000001011	遅延分岐、PR→PC	2	—
SETT	0000000000011000	1→T	1	1
SLEEP	0000000000011011	スリープ	3*1	—

【注】 *1　スリープ状態に遷移するまでのステート数です。

(2)　n 形式

表 A.26　レジスタ直接

命令		命令コード	動作	実行ステート	T ビット
CMP/PL	Rn	0100nnnn00010101	Rn>0 とき 1→T	1	比較結果
CMP/PZ	Rn	0100nnnn00010001	Rn≧0 とき 1→T	1	比較結果
DT	Rn	0100nnnn00010000	Rn-1→Rn, Rn が 0 のとき 1→T Rn が 0 以外のとき 0→T	1	比較結果
MOVT	Rn	0000nnnn00101001	T→Rn	1	—
ROTL	Rn	0100nnnn00000100	T←Rn←MSB	1	MSB
ROTR	Rn	0100nnnn00000101	LSB→Rn→T	1	LSB
ROTCL	Rn	0100nnnn00100100	T←Rn←T	1	MSB
ROTCR	Rn	0100nnnn00100101	T→Rn→T	1	LSB
SHAL	Rn	0100nnnn00100000	T←Rn←0	1	MSB
SHAR	Rn	0100nnnn00100001	MSB→Rn→T	1	LSB
SHLL	Rn	0100nnnn00000000	T←Rn←0	1	MSB
SHLR	Rn	0100nnnn00000001	0→Rn→T	1	LSB
SHLL2	Rn	0100nnnn00001000	Rn<<2→Rn	1	—
SHLR2	Rn	0100nnnn00001001	Rn>>2→Rn	1	—
SHLL8	Rn	0100nnnn00011000	Rn<<8→Rn	1	—
SHLR8	Rn	0100nnnn00011001	Rn>>8→Rn	1	—
SHLL16	Rn	0100nnnn00101000	Rn<<16→Rn	1	—
SHLR16	Rn	0100nnnn00101001	Rn>>16→Rn	1	—

表 A.27　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		命令コード	動作	実行ステート	T ビット
STC	SR,Rn	0000nnnn00000010	SR→Rn	1	—
STC	GBR,Rn	0000nnnn00010010	GBR→Rn	1	—
STC	VBR,Rn	0000nnnn00100010	VBR→Rn	1	—
STS	FPSCR,Rn	0000nnnn01101010	FPSCR→Rn	1	—
STS	FPUL,Rn	0000nnnn01011010	FPUL→Rn	1	—
STS	MACH,Rn	0000nnnn00001010	MACH→Rn	1	—
STS	MACL,Rn	0000nnnn00011010	MACL→Rn	1	—
STS	PR,Rn	0000nnnn00101010	PR→Rn	1	—

表 A.28　レジスタ間接

命令		命令コード	動作	実行ステート	T ビット
TAS.B	@Rn	0100nnnn00011011	(Rn)が 0 のとき 1→T, 1→MSBof(Rn)	4	テスト結果

表 A.29　プリデクリメントレジスタ間接

命令		命令コード	動作	実行ステート	T ビット
STC.L	SR,@-Rn	0100nnnn00000011	Rn-4→Rn, SR→(Rn)	2	—
STC.L	GBR,@-Rn	0100nnnn00010011	Rn-4→Rn, GBR→(Rn)	2	—
STC.L	VBR,@-Rn	0100nnnn00100011	Rn-4→Rn, VBR→(Rn)	2	—
STS.L	FPSCR,@-Rn	0100nnnn01100010	Rn-4→Rn, FPSCR→Rn	1	—
STS.L	FPUL,@-Rn	0100nnnn01010010	Rn-4→Rn, FPUL→Rn	1	—
STS.L	MACH,@-Rn	0100nnnn00000010	Rn-4→Rn, MACH→(Rn)	1	—
STS.L	MACL,@-Rn	0100nnnn00010010	Rn-4→Rn, MACL→(Rn)	1	—
STS.L	PR,@-Rn	0100nnnn00100010	Rn-4→Rn, PR→(Rn)	1	—

表 A.30　浮動小数点命令

命令		命令コード	動作	実行ステート	T ビット
FABS	FRn	1111nnnn01011101	| FRn |→FRn	1	—
FLDI0	FRn	1111nnnn10001101	H'00000000→FRn	1	—
FLDI1	FRn	1111nnnn10011101	H'3F800000→FRn	1	—
FLOAT	FPUL,FRn	1111nnnn00101101	(float)FPUL→FRn	1	—
FNEG	FRn	1111nnnn01001101	-FRn→FRn	1	—
FSTS	FPUL,FRn	1111nnnn00001101	FPUL→FRn	1	—

(3)　m 形式

表 A.31　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）

命令		命令コード	動作	実行ステート	T ビット
LDC	Rm,SR	0100mmmm00001110	Rm→SR	1	LSB
LDC	Rm,GBR	0100mmmm00011110	Rm→GBR	1	—
LDC	Rm,VBR	0100mmmm00101110	Rm→VBR	1	—
LDS	Rm,FPSCR	0100mmmm01101010	Rm→FPSCR	1	—
LDS	Rm,FPUL	0100mmmm01011010	Rm→FPUL	1	—
LDS	Rm,MACH	0100mmmm00001010	Rm→MACH	1	—
LDS	Rm,MACL	0100mmmm00011010	Rm→MACL	1	—
LDS	Rm,PR	0100mmmm00101010	Rm→PR	1	—

表 A.32　レジスタ間接

命令		命令コード	動作	実行ステート	T ビット
JMP	@Rm	0100mmmm00101011	遅延分岐、Rm→PC	2	—
JSR	@Rm	0100mmmm00001011	遅延分岐、PC→PR, Rm→PC	2	—

表 A.33　ポストインクリメントレジスタ間接

命令		命令コード	動作	実行ステート	T ビット
LDC.L	@Rm+,SR	0100mmmm00000111	(Rm)→SR, Rm+4→Rm	3	LSB
LDC.L	@Rm+,GBR	0100mmmm00010111	(Rm)→GBR, Rm+4→Rm	3	—
LDC.L	@Rm+,VBR	0100mmmm00100111	(Rm)→VBR, Rm+4→Rm	3	—
LDS.L	@Rm+,FPSCR	0100mmmm01100110	(Rm)→FPSCR, Rm+4→Rm	1	—
LDS.L	@Rm+,FPUL	0100mmmm01010110	(Rm)→FPUL, Rm+4→Rm	1	—
LDS.L	@Rm+,MACH	0100mmmm00000110	(Rm)→MACH, Rm+4→Rm	1	—
LDS.L	@Rm+,MACL	0100mmmm00010110	(Rm)→MACL, Rm+4→Rm	1	—
LDS.L	@Rm+,PR	0100mmmm00100110	(Rm)→PR, Rm+4→Rm	1	—

表 A.34　Rm を用いた PC 相対

命令		命令コード	動作	実行ステート	T ビット
BRAF	Rm	0000mmmm00100011	遅延分岐、Rm+PC→PC	2	—
BSRF	Rm	0000mmmm00000011	遅延分岐、PC→PR, Rm+PC→PC	2	—

表 A.35　浮動小数点命令

命令		命令コード	動作	実行ステート	T ビット
FLDS	FRm,FPUL	1111mmmm00011101	FRm→FPUL	1	—
FTRC	FRm,FPUL	1111mmmm00111101	(long)FRm→FPUL	1	—

(4)　nm 形式

表 A.36　レジスタ直接

命令		命令コード	動作	実行ステート	T ビット
ADD	Rm,Rn	0011nnnnmmmm1100	Rn+Rm→Rn	1	—
ADDC	Rm,Rn	0011nnnnmmmm1110	Rn+Rm+T→Rn, キャリ→T	1	キャリ
ADDV	Rm,Rn	0011nnnnmmmm1111	Rn+Rm→Rn, オーバフロー→T	1	オーバフロー
AND	Rm,Rn	0010nnnnmmmm1001	Rn & Rm→Rn	1	—
CMP/EQ	Rm,Rn	0011nnnnmmmm0000	Rn=Rm のとき 1→T	1	比較結果
CMP/HS	Rm,Rn	0011nnnnmmmm0010	無符号で Rn≧Rm のとき 1→T	1	比較結果
CMP/GE	Rm,Rn	0011nnnnmmmm0011	有符号で Rn≧Rm のとき 1→T	1	比較結果
CMP/HI	Rm,Rn	0011nnnnmmmm0110	無符号で Rn>Rm のとき 1→T	1	比較結果
CMP/GT	Rm,Rn	0011nnnnmmmm0111	有符号で Rn>Rm のとき 1→T	1	比較結果
CMP/STR	Rm,Rn	0010nnnnmmmm1100	いずれかのバイトが等しいとき 1→T	1	比較結果
DIV1	Rm,Rn	0011nnnnmmmm0100	1 ステップ除算(Rn÷Rm)	1	計算結果
DIV0S	Rm,Rn	0010nnnnmmmm0111	Rn の MSB→Q, Rm の MSB→M, M ^ Q→T	1	計算結果
DMULS.L	Rm,Rn	0011nnnnmmmm1101	符号付きで Rn×Rm→MACH, MACL	2〜4※2	—
DMULU.L	Rm,Rn	0011nnnnmmmm0101	符号なしで Rn×Rm→MACH, MACL	2〜4※2	—
EXTS.B	Rm,Rn	0110nnnnmmmm1110	Rm をバイトから符号拡張→Rn	1	—
EXTS.W	Rm,Rn	0110nnnnmmmm1111	Rm をワードから符号拡張→Rn	1	—
EXTU.B	Rm,Rn	0110nnnnmmmm1100	Rm をバイトからゼロ拡張→Rn	1	—
EXTU.W	Rm,Rn	0110nnnnmmmm1101	Rm をワードからゼロ拡張→Rn	1	—
MOV	Rm,Rn	0110nnnnmmmm0011	Rm→Rn	1	—
MUL.L	Rm,Rn	0000nnnnmmmm0111	Rn×Rm→MACL	2〜4※2	—
MULS.W	Rm,Rn	0010nnnnmmmm1111	符号付きで Rn×Rm→MAC	1〜3※2	—
MULU.W	Rm,Rn	0010nnnnmmmm1110	符号なしで Rn×Rm→MAC	1〜3※2	—
NEG	Rm,Rn	0110nnnnmmmm1011	0-Rm→Rn	1	—
NEGC	Rm,Rn	0110nnnnmmmm1010	0-Rm-T→Rn, ボロー→T	1	ボロー
NOT	Rm,Rn	0110nnnnmmmm0111	~Rm→Rn	1	—
OR	Rm,Rn	0010nnnnmmmm1011	Rn | Rm→Rn	1	—
SUB	Rm,Rn	0011nnnnmmmm1000	Rn-Rm→Rn	1	—
SUBC	Rm,Rn	0011nnnnmmmm1010	Rn-Rm-T→Rn, ボロー→T	1	ボロー

命令	命令コード	動作	実行ステート	T ビット
SUBV Rm,Rn	0011nnnnmmmm1011	Rn-Rm→Rn, アンダフロー→T	1	アンダフロー
SWAP.B Rm,Rn	0110nnnnmmmm1000	Rm→下位 2 バイトの上下バイト交換→Rn	1	—
SWAP.W Rm,Rn	0110nnnnmmmm1001	Rm→上下ワード交換→Rn	1	—
TST Rm,Rn	0010nnnnmmmm1000	Rn & Rm, 結果が 0 のとき 1→T	1	テスト結果
XOR Rm,Rn	0010nnnnmmmm1010	Rn ^ Rm→Rn	1	—
XTRCT Rm,Rn	0010nnnnmmmm1101	Rm:Rn の中央 32 ビット→Rn	1	—

【注】 *2　通常実行ステートを示します。

表 A.37　レジスタ間接

命令	命令コード	動作	実行ステート	T ビット
MOV.B Rm,@Rn	0010nnnnmmmm0000	Rm→(Rn)	1	—
MOV.W Rm,@Rn	0010nnnnmmmm0001	Rm→(Rn)	1	—
MOV.L Rm,@Rn	0010nnnnmmmm0010	Rm→(Rn)	1	—
MOV.B @Rm,Rn	0110nnnnmmmm0000	(Rm)→符号拡張→Rn	1	—
MOV.W @Rm,Rn	0110nnnnmmmm0001	(Rm)→符号拡張→Rn	1	—
MOV.L @Rm,Rn	0110nnnnmmmm0010	(Rm)→Rn	1	—

表 A.38　ポストインクリメントレジスタ間接（積和演算）

命令	命令コード	動作	実行ステート	T ビット
MAC.L @Rm+,@Rn+	0000nnnnmmmm1111	符号付きで(Rn)×(Rm)+MAC→MAC	3/(2〜4)*2	—
MAC.W @Rm+,@Rn+	0100nnnnmmmm1111	符号付きで(Rn)×(Rm)+MAC→MAC	3/(2)*2	—

【注】 *2　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。

表 A.39　ポストインクリメントレジスタ間接

命令	命令コード	動作	実行ステート	T ビット
MOV.B @Rm+,Rn	0110nnnnmmmm0100	(Rm)→符号拡張→Rn, Rm+1→Rm	1	—
MOV.W @Rm+,Rn	0110nnnnmmmm0101	(Rm)→符号拡張→Rn, Rm+2→Rm	1	—
MOV.L @Rm+,Rn	0110nnnnmmmm0110	(Rm)→Rn, Rm+4→Rm	1	—

表 A.40　プリデクリメントレジスタ間接

命令	命令コード	動作	実行ステート	T ビット
MOV.B Rm,@-Rn	0010nnnnmmmm0100	Rn-1→Rn, Rm→(Rn)	1	—
MOV.W Rm,@-Rn	0010nnnnmmmm0101	Rn-2→Rn, Rm→(Rn)	1	—
MOV.L Rm,@-Rn	0010nnnnmmmm0110	Rn-4→Rn, Rm→(Rn)	1	—

表 A.41　インデックス付きレジスタ間接

命令	命令コード	動作	実行ステート	T ビット
MOV.B Rm,@(R0,Rn)	0000nnnnmmmm0100	Rm→(R0+Rn)	1	—
MOV.W Rm,@(R0,Rn)	0000nnnnmmmm0101	Rm→(R0+Rn)	1	—
MOV.L Rm,@(R0,Rn)	0000nnnnmmmm0110	Rm→(R0+Rn)	1	—
MOV.B @(R0,Rm),Rn	0000nnnnmmmm1100	(R0+Rm)→符号拡張→Rn	1	—
MOV.W @(R0,Rm),Rn	0000nnnnmmmm1101	(R0+Rm)→符号拡張→Rn	1	—
MOV.L @(R0,Rm),Rn	0000nnnnmmmm1110	(R0+Rm)→Rn	1	—

表 A.42　浮動小数点命令

命令	命令コード	動作	実行ステート	T ビット
FADD FRm,FRn	1111nnnnmmmm0000	FRn+FRm→FRn	1	—
FCMP/EQ FRm,FRn	1111nnnnmmmm0100	(FRn=FRm)? 1:0→T	1	比較結果
FCMP/GT FRm,FRn	1111nnnnmmmm0101	(FRn>FRm)? 1:0→T	1	比較結果
FDIV FRm,FRn	1111nnnnmmmm0011	FRn/FRm→FRn	13	—
FMAC FR0,FRm,FRn	1111nnnnmmmm1110	FR0×FRm+FRn→FRn	1	—
FMOV FRm,FRn	1111nnnnmmmm1100	FRm→FRn	1	—
FMOV.S @(R0,Rm),FRn	1111nnnnmmmm0110	(R0+Rm)→FRn	1	—
FMOV.S @Rm+,FRn	1111nnnnmmmm1001	(Rm)→FRn, Rm+4→Rm	1	—
FMOV.S @Rm,FRn	1111nnnnmmmm1000	(Rm)→FRn	1	—
FMOV.S FRm,@(R0,Rn)	1111nnnnmmmm0111	FRm→(R0+Rn)	1	—
FMOV.S FRm,@-Rn	1111nnnnmmmm1011	Rn-4→Rn, FRm→(Rn)	1	—
FMOV.S FRm,@Rn	1111nnnnmmmm1010	FRm→(Rn)	1	—
FMUL FRm,FRn	1111nnnnmmmm0010	FRn×FRm→FRn	1	—
FSUB FRm,FRn	1111nnnnmmmm0001	FRn-FRm→FRn	1	—

(5)　md 形式

表 A.43　md 形式

命令	命令コード	動作	実行ステート	T ビット
MOV.B @(disp,Rm),R0	10000100mmmmdddd	(disp+Rm)→符号拡張→R0	1	—
MOV.W @(disp,Rm),R0	10000101mmmmdddd	(disp×2+Rm)→符号拡張→R0	1	—

(6)　nd4 形式

表 A.44　nd4 形式

命令		命令コード	動作	実行ステート	T ビット
MOV.B	R0,@(disp,Rn)	10000000nnnndddd	R0→(disp+Rn)	1	—
MOV.W	R0,@(disp,Rn)	10000001nnnndddd	R0→(disp×2+Rn)	1	—

(7)　nmd 形式

表 A.45　nmd 形式

命令		命令コード	動作	実行ステート	T ビット
MOV.L	Rm,@(disp,Rn)	0001nnnnmmmmdddd	Rm→(disp×4+Rn)	1	—
MOV.L	@(disp,Rm),Rn	0101nnnnmmmmdddd	(disp×4+Rm)→Rn	1	—

(8)　d 形式

表 A.46　ディスプレースメント付き GBR 間接

命令		命令コード	動作	実行ステート	T ビット
MOV.B	R0,@(disp,GBR)	11000000dddddddd	R0→(disp+GBR)	1	—
MOV.W	R0,@(disp,GBR)	11000001dddddddd	R0→(disp×2+GBR)	1	—
MOV.L	R0,@(disp,GBR)	11000010dddddddd	R0→(disp×4+GBR)	1	—
MOV.B	@(disp,GBR),R0	11000100dddddddd	(disp+GBR)→符号拡張→R0	1	—
MOV.W	@(disp,GBR),R0	11000101dddddddd	(disp×2+GBR)→符号拡張→R0	1	—
MOV.L	@(disp,GBR),R0	11000110dddddddd	(disp×4+GBR)→R0	1	—

表 A.47　ディスプレースメント付き PC 相対

命令		命令コード	動作	実行ステート	T ビット
MOVA	@(disp,PC),R0	11000111dddddddd	disp×4+PC→R0	1	—

表 A.48　PC 相対

命令		命令コード	動作	実行ステート	T ビット
BF	label	10001011dddddddd	T=0 のとき disp×2+PC→PC, T=1 のとき nop	3/1*3	—
BF/S	label	10001111dddddddd	T=0 のとき disp×2+PC→PC, T=1 のとき nop	2/1*3	—
BT	label	10001001dddddddd	T=1 のとき disp×2+PC→PC, T=0 のとき nop	3/1*3	—
BT/S	label	10001101dddddddd	T=1 のとき disp×2+PC→PC, T=0 のとき nop	2/1*3	—

【注】 *3　分岐しないときは 1 ステートになります。

(9)　d12 形式

表 A.49　d12 形式

命令		命令コード	動作	実行ステート	T ビット
BRA	label	1010ddddddddddddd	遅延分岐、disp×2+PC→PC	2	—
BSR	label	1011ddddddddddddd	遅延分岐、PC→PR, disp×2+PC →PC	2	—

(10)　nd8 形式

表 A.50　nd8 形式

命令		命令コード	動作	実行ステート	T ビット
MOV.W	@(disp,PC),Rn	1001nnnnddddddddd	(disp×2+PC)→符号拡張→Rn	1	—
MOV.L	@(disp,PC),Rn	1101nnnnddddddddd	(disp×4+PC)→Rn	1	—

(11)　i 形式

表 A.51　インデックス付き GBR 間接

命令		命令コード	動作	実行ステート	T ビット
AND.B	#imm,@(R0,GBR)	11001101iiiiiiii	(R0+GBR) & imm→(R0+GBR)	3	—
OR.B	#imm,@(R0,GBR)	11001111iiiiiiii	(R0+GBR) | imm→(R0+GBR)	3	—
TST.B	#imm,@(R0,GBR)	11001100iiiiiiii	(R0+GBR) & imm, 結果が 0 のとき 1→T	3	テスト結果
XOR.B	#imm,@(R0,GBR)	11001110iiiiiiii	(R0+GBR) ^ imm→(R0+GBR)	3	—

表 A.52　イミディエイト（レジスタ直接との算術論理演算）

命令		命令コード	動作	実行ステート	T ビット
AND	#imm,R0	11001001iiiiiiii	R0 & imm→R0	1	—
CMP/EQ	#imm,R0	10001000iiiiiiii	R0=imm のとき 1→T	1	比較結果
OR	#imm,R0	11001011iiiiiiii	R0 | imm→R0	1	—
TST	#imm,R0	11001000iiiiiiii	R0 & imm, 結果が 0 のとき 1→T	1	テスト結果
XOR	#imm,R0	11001010iiiiiiii	R0 ^ imm→R0	1	—

表 A.53　イミディエイト（例外処理ベクタの指定）

命令		命令コード	動作	実行ステート	T ビット
TRAPA	#imm	11000011iiiiiiii	PC/SR→スタック領域, (imm×4+VBR)→PC	8	—

(12) ni 形式

表 A.54　ni 形式

命令		命令コード	動作	実行ステート	T ビット
ADD	#imm,Rn	0111nnnniiiiiiii	Rn+imm→Rn	1	—
MOV	#imm,Rn	1110nnnniiiiiiii	imm→符号拡張→Rn	1	—

A.3　命令コード順命令セット

命令の命令コードと実行ステートを、命令コード順に示します。

表 A.55　命令コード順命令セット

命令		命令コード	動作	実行ステート	T ビット
CLRT		0000000000001000	0→T	1	0
NOP		0000000000001001	無操作	1	—
RTS		0000000000001011	遅延分岐、PR→PC	2	—
SETT		0000000000011000	1→T	1	1
DIV0U		0000000000011001	0→M/Q/T	1	0
SLEEP		0000000000011011	スリープ	3	—
CLRMAC		0000000000101000	0→MACH, MACL	1	—
RTE		0000000000101011	遅延分岐、 スタック領域→PC/SR	4	LSB
STC	SR,Rn	0000nnnn00000010	SR→Rn	1	—
BSRF	Rm	0000mmmm00000011	遅延分岐、PC→PR, Rm+PC→PC	2	—
STS	MACH,Rn	0000nnnn00001010	MACH→Rn	1	—
STC	GBR,Rn	0000nnnn00010010	GBR→Rn	1	—
STS	MACL,Rn	0000nnnn00011010	MACL→Rn	1	—
STC	VBR,Rn	0000nnnn00100010	VBR→Rn	1	—
BRAF	Rm	0000mmmm00100011	遅延分岐、Rm+PC→PC	2	—
MOVT	Rn	0000nnnn00101001	T→Rn	1	—
STS	PR,Rn	0000nnnn00101010	PR→Rn	1	—
STS	FPUL,Rn	0000nnnn01011010	FPUL→Rn	1	—
STS	FPSCR,Rn	0000nnnn01101010	FPSCR→Rn	1	—
MOV.B	Rm,@(R0,Rn)	0000nnnnmmmm0100	Rm→(R0+Rn)	1	—
MOV.W	Rm,@(R0,Rn)	0000nnnnmmmm0101	Rm→(R0+Rn)	1	—
MOV.L	Rm,@(R0,Rn)	0000nnnnmmmm0110	Rm→(R0+Rn)	1	—
MUL.L	Rm,Rn	0000nnnnmmmm0111	Rn×Rm→MACL	2〜4*1	—
MOV.B	@(R0,Rm),Rn	0000nnnnmmmm1100	(R0+Rm)→符号拡張→Rn	1	—
MOV.W	@(R0,Rm),Rn	0000nnnnmmmm1101	(R0+Rm)→符号拡張→Rn	1	—
MOV.L	@(R0,Rm),Rn	0000nnnnmmmm1110	(R0+Rm)→Rn	1	—
MAC.L	@Rm+,@Rn+	0000nnnnmmmm1111	符号付きで(Rn)×(Rm)+MAC→MAC	3/ (2〜4)*1	—

命令		命令コード	動作	実行ステート	T ビット
MOV.L	Rm,@(disp,Rn)	0001nnnnmmmmdddd	Rm→(disp×4+Rn)	1	—
MOV.B	Rm,@Rn	0010nnnnmmmm0000	Rm→(Rn)	1	—
MOV.W	Rm,@Rn	0010nnnnmmmm0001	Rm→(Rn)	1	—
MOV.L	Rm,@Rn	0010nnnnmmmm0010	Rm→(Rn)	1	—
MOV.B	Rm,@-Rn	0010nnnnmmmm0100	Rn-1→Rn, Rm→(Rn)	1	—
MOV.W	Rm,@-Rn	0010nnnnmmmm0101	Rn-2→Rn, Rm→(Rn)	1	—
MOV.L	Rm,@-Rn	0010nnnnmmmm0110	Rn-4→Rn, Rm→(Rn)	1	—
DIV0S	Rm,Rn	0010nnnnmmmm0111	Rn の MSB→Q, Rm の MSB→M, M ^ Q→T	1	計算結果
TST	Rm,Rn	0010nnnnmmmm1000	Rn & Rm, 結果が 0 のとき 1→T	1	テスト結果
AND	Rm,Rn	0010nnnnmmmm1001	Rn & Rm→Rn	1	—
XOR	Rm,Rn	0010nnnnmmmm1010	Rn ^ Rm→Rn	1	—
OR	Rm,Rn	0010nnnnmmmm1011	Rn ｜ Rm→Rn	1	—
CMP/STR	Rm,Rn	0010nnnnmmmm1100	いずれかのバイトが等しいとき 1→T	1	比較結果
XTRCT	Rm,Rn	0010nnnnmmmm1101	Rm:Rn の中央 32 ビット→Rn	1	—
MULU.W	Rm,Rn	0010nnnnmmmm1110	符号なしで Rn×Rm→MAC	1〜3*1	—
MULS.W	Rm,Rn	0010nnnnmmmm1111	符号付きで Rn×Rm→MAC	1〜3*1	—
CMP/EQ	Rm,Rn	0011nnnnmmmm0000	Rn=Rm のとき 1→T	1	比較結果
CMP/HS	Rm,Rn	0011nnnnmmmm0010	無符号で Rn≧Rm のとき 1→T	1	比較結果
CMP/GE	Rm,Rn	0011nnnnmmmm0011	有符号で Rn≧Rm のとき 1→T	1	比較結果
DIV1	Rm,Rn	0011nnnnmmmm0100	1 ステップ除算(Rn÷Rm)	1	計算結果
DMULU.L	Rm,Rn	0011nnnnmmmm0101	符号なしで Rn×Rm→MACH, MACL	2〜4*1	—
CMP/HI	Rm,Rn	0011nnnnmmmm0110	無符号で Rn>Rm のとき 1→T	1	比較結果
CMP/GT	Rm,Rn	0011nnnnmmmm0111	有符号で Rn>Rm のとき 1→T	1	比較結果
SUB	Rm,Rn	0011nnnnmmmm1000	Rn-Rm→Rn	1	—
SUBC	Rm,Rn	0011nnnnmmmm1010	Rn-Rm-T→Rn, ボロー→T	1	ボロー
SUBV	Rm,Rn	0011nnnnmmmm1011	Rn-Rm→Rn, アンダフロー→T	1	アンダフロー
ADD	Rm,Rn	0011nnnnmmmm1100	Rn+Rm→Rn	1	—
DUMLS.L	Rm,Rn	0011nnnnmmmm1101	符号付きで Rn×Rm→MACH, MACL	2〜4*1	—
ADDC	Rm,Rn	0011nnnnmmmm1110	Rn+Rm+T→Rn, キャリ→T	1	キャリ
ADDV	Rm,Rn	0011nnnnmmmm1111	Rn+Rm→Rn, オーバフロー→T	1	オーバフロー
SHLL	Rn	0100nnnn00000000	T←Rn←0	1	MSB
SHLR	Rn	0100nnnn00000001	0→Rn→T	1	LSB
STS.L	MACH,@-Rn	0100nnnn00000010	Rn-4→Rn, MACH→(Rn)	1	—
STC.L	SR,@-Rn	0100nnnn00000011	Rn-4→Rn, SR→(Rn)	2	—
ROTL	Rn	0100nnnn00000100	T←Rn←MSB	1	MSB
ROTR	Rn	0100nnnn00000101	LSB→Rn→T	1	LSB

命令		命令コード	動作	実行ステート	T ビット
LDS.L	@Rm+,MACH	0100mmmm00000110	(Rm)→MACH, Rm+4→Rm	1	—
LDC.L	@Rm+,SR	0100mmmm00000111	(Rm)→SR, Rm+4→Rm	3	LSB
SHLL2	Rn	0100nnnn00001000	Rn<<2→Rn	1	—
SHLR2	Rn	0100nnnn00001001	Rn>>2→Rn	1	—
LDS	Rm,MACH	0100mmmm00001010	Rm→MACH	1	—
JSR	@Rm	0100mmmm00001011	遅延分岐、PC→PR, Rm→PC	2	—
LDC	Rm,SR	0100mmmm00001110	Rm→SR	1	LSB
DT	Rn	0100nnnn00010000	Rn-1→Rn, Rn が 0 のとき 1→T Rn が 0 以外のとき 0→T	1	比較結果
CMP/PZ	Rn	0100nnnn00010001	Rn≧0 とき 1→T	1	比較結果
STS.L	MACL,@-Rn	0100nnnn00010010	Rn-4→Rn, MACL→(Rn)	1	—
STC.L	GBR,@-Rn	0100nnnn00010011	Rn-4→Rn, GBR→(Rn)	2	—
CMP/PL	Rn	0100nnnn00010101	Rn>0 とき 1→T	1	比較結果
LDS.L	@Rm+,MACL	0100mmmm00010110	(Rm)→MACL, Rm+4→Rm	1	—
LDC.L	@Rm+,GBR	0100mmmm00010111	(Rm)→GBR, Rm+4→Rm	3	—
SHLL8	Rn	0100nnnn00011000	Rn<<8→Rn	1	—
SHLR8	Rn	0100nnnn00011001	Rn>>8→Rn	1	—
LDS	Rm,MACL	0100mmmm00011010	Rm→MACL	1	—
TAS.B	@Rn	0100nnnn00011011	(Rn)が 0 のとき 1→T, 1→MSB of (Rn)	4	テスト結果
LDC	Rm,GBR	0100mmmm00011110	Rm→GBR	1	—
SHAL	Rn	0100nnnn00100000	T←Rn←0	1	MSB
SHAR	Rn	0100nnnn00100001	MSB→Rn→T	1	LSB
STS.L	PR,@-Rn	0100nnnn00100010	Rn-4→Rn, PR→(Rn)	1	—
STC.L	VBR,@-Rn	0100nnnn00100011	Rn-4→Rn, VBR→(Rn)	2	—
ROTCL	Rn	0100nnnn00100100	T←Rn←T	1	MSB
ROTCR	Rn	0100nnnn00100101	T→Rn→T	1	LSB
LDS.L	@Rm+,PR	0100mmmm00100110	(Rm)→PR, Rm+4→Rm	1	—
LDC.L	@Rm+,VBR	0100mmmm00100111	(Rm)→VBR, Rm+4→Rm	3	—
SHLL16	Rn	0100nnnn00101000	Rn<<16→Rn	1	—
SHLR16	Rn	0100nnnn00101001	Rn>>16→Rn	1	—
LDS	Rm,PR	0100mmmm00101010	Rm→PR	1	—
JMP	@Rm	0100mmmm00101011	遅延分岐、Rm→PC	2	—
LDC	Rm,VBR	0100mmmm00101110	Rm→VBR	1	—
STS.L	FPUL,@-Rn	0100nnnn01010010	Rn-4→Rn, FPUL→ (Rn)	1	—
LDS.L	@Rm+,FPUL	0100nmmm01010110	(Rm)→FPUL, Rm+4→Rm	1	—
LDS	Rm,FPUL	0100mmmm01011010	Rm→FPUL	1	—
STS.L	FPSCR,@-Rn	0100nnnn01100010	Rn-4→Rn, FPSCR→(Rn)	1	—
LDS.L	@Rm,FPSCR	0100mmmm01100110	(Rm)→FPSCR, Rm+4→Rm	1	—
LDS	Rm,FPSCR	0100nmmm01101010	Rm→FPSCR	1	—

命令		命令コード	動作	実行ステート	T ビット
MAC.W	@Rm+,@Rn+	0100nnnnmmmm1111	**符号付きで**(Rn)×(Rm)+MAC→MAC	3/(2)*1	—
MOV.L	@(disp,Rm),Rn	0101nnnnmmmmdddd	(disp×4+Rm)→Rn	1	—
MOV.B	@Rm,Rn	0110nnnnmmmm0000	(Rm)→**符号拡張**→Rn	1	—
MOV.W	@Rm,Rn	0110nnnnmmmm0001	(Rm)→**符号拡張**→Rn	1	—
MOV.L	@Rm,Rn	0110nnnnmmmm0010	(Rm)→Rn	1	—
MOV	Rm,Rn	0110nnnnmmmm0011	Rm→Rn	1	—
MOV.B	@Rm+,Rn	0110nnnnmmmm0100	(Rm)→**符号拡張**→Rn, Rm+1→Rm	1	—
MOV.W	@Rm+,Rn	0110nnnnmmmm0101	(Rm)→**符号拡張**→Rn, Rm+2→Rm	1	—
MOV.L	@Rm+,Rn	0110nnnnmmmm0110	(Rm)→Rn, Rm+4→Rm	1	—
NOT	Rm,Rn	0110nnnnmmmm0111	~Rm→Rn	1	—
SWAP.B	Rm,Rn	0110nnnnmmmm1000	Rm→**下位 2 バイトの上下バイト交換**→Rn	1	—
SWAP.W	Rm,Rn	0110nnnnmmmm1001	Rm→**上下ワード交換**→Rn	1	—
NEGC	Rm,Rn	0110nnnnmmmm1010	0-Rm-T→Rn, **ボロー**→T	1	**ボロー**
NEG	Rm,Rn	0110nnnnmmmm1011	0-Rm→Rn	1	—
EXTU.B	Rm,Rn	0110nnnnmmmm1100	Rm **をバイトからゼロ拡張**→Rn	1	—
EXTU.W	Rm,Rn	0110nnnnmmmm1101	Rm **をワードからゼロ拡張**→Rn	1	—
EXTS.B	Rm,Rn	0110nnnnmmmm1110	Rm **をバイトから符号拡張**→Rn	1	—
EXTS.W	Rm,Rn	0110nnnnmmmm1111	Rm **をワードから符号拡張**→Rn	1	—
ADD	#imm,Rn	0111nnnniiiiiiii	Rn+imm→Rn	1	—
MOV.B	R0,@(disp,Rn)	10000000nnnndddd	R0→(disp+Rn)	1	—
MOV.W	R0,@(disp,Rn)	10000001nnnndddd	R0→(disp×2+Rn)	1	—
MOV.B	@(disp,Rm),R0	10000100mmmmdddd	(disp+Rm)→**符号拡張**→R0	1	—
MOV.W	@(disp,Rm),R0	10000101mmmmdddd	(disp×2+Rm)→**符号拡張**→R0	1	—
CMP/EQ	#imm,R0	10001000iiiiiiii	R0=imm **のとき** 1→T	1	**比較結果**
BT	label	10001001dddddddd	T=1 **のとき** disp×2+PC→PC, T=0 **のとき** nop	3/1*2	—
BT/S	label	10001101dddddddd	T=1 **のとき** disp×2+PC→PC, T=0 **のとき** nop	2/1*2	—
BF	label	10001011dddddddd	T=0 **のとき** disp×2+PC→PC, T=1 **のとき** nop	3/1*2	—
BF/S	label	10001111dddddddd	T=0 **のとき** disp×2+PC→PC, T=1 **のとき** nop	2/1*2	—
MOV.W	@(disp,PC),Rn	1001nnnndddddddd	(disp×2+PC)→**符号拡張**→Rn	1	—
BRA	label	1010dddddddddddd	**遅延分岐、**disp×2+PC→PC	2	—
BSR	label	1011dddddddddddd	**遅延分岐、**PC→PR, disp×2+PC→PC	2	—
MOV.B	R0,@(disp,GBR)	11000000dddddddd	R0→(disp+GBR)	1	—
MOV.W	R0,@(disp,GBR)	11000001dddddddd	R0→(disp×2+GBR)	1	—
MOV.L	R0,@(disp,GBR)	11000010dddddddd	R0→(disp×4+GBR)	1	—

命令		命令コード	動作	実行ステート	T ビット
TRAPA	#imm	11000011iiiiiiii	PC/SR→**スタック領域**, (imm×4+VBR)→PC	8	—
MOV.B	@(disp,GBR),R0	11000100dddddddd	(disp+GBR)→**符号拡張**→R0	1	—
MOV.W	@(disp,GBR),R0	11000101dddddddd	(disp×2+GBR)→**符号拡張**→R0	1	—
MOV.L	@(disp,GBR),R0	11000110dddddddd	(disp×4+GBR)→R0	1	—
MOVA	@(disp,PC),R0	11000111dddddddd	disp×4+PC→R0	1	—
TST	#imm,R0	11001000iiiiiiii	R0 & imm, **結果が** 0 **のとき** 1→T	1	**テスト結果**
AND	#imm,R0	11001001iiiiiiii	R0 & imm→R0	1	—
XOR	#imm,R0	11001010iiiiiiii	R0 ^ imm→R0	1	—
OR	#imm,R0	11001011iiiiiiii	R0 | imm→R0	1	—
TST.B	#imm,@(R0,GBR)	11001100iiiiiiii	(R0+GBR) & imm, **結果が** 0 **のとき** 1→T	3	**テスト結果**
AND.B	#imm,@(R0,GBR)	11001101iiiiiiii	(R0+GBR) & imm→(R0+GBR)	3	—
XOR.B	#imm,@(R0,GBR)	11001110iiiiiiii	(R0+GBR) ^ imm→(R0+GBR)	3	—
OR.B	#imm,@(R0,GBR)	11001111iiiiiiii	(R0+GBR) | imm→ (R0+GBR)	3	—
MOV.L	@(disp,PC),Rn	1101nnnndddddddd	(disp×4+PC)→Rn	1	—
MOV	#imm,Rn	1110nnnniiiiiiii	imm→**符号拡張**→Rn	1	—
FSTS	FPUL,FRn	1111nnnn00001101	FPUL→FRn	1	—
FLDS	FRm,FPUL	1111nnnn00011101	FRm→FPUL	1	—
FLOAT	FPUL,FRn	1111nnnn00101101	(float)FPUL→FRn	1	—
FTRC	FRm,FPUL	1111nnnn00111101	(long)FRm→FPUL	1	—
FNEG	FRn	1111nnnn01001101	-FRn→FRn	1	—
FABS	FRn	1111nnnn01011101	| FRn |→FRn	1	—
FLDI0	FRn	1111nnnn10001101	H'00000000→FRn	1	—
FLDI1	FRn	1111nnnn10011101	H'3F800000→FRn	1	—
FADD	FRm,FRn	1111nnnnmmmm0000	FRn+FRm→FRn	1	—
FSUB	FRm,FRn	1111nnnnmmmm0001	FRn-FRm→FRn	1	—
FMUL	FRm,FRn	1111nnnnmmmm0010	FRn×FRm→FRn	1	—
FDIV	FRm,FRn	1111nnnnmmmm0011	FRn/FRm→FRn	13	—
FCMP/EQ	FRm,FRn	1111nnnnmmmm0100	(FRn=FRm)?1:0→T	1	**比較結果**
FCMP/GT	FRm,FRn	1111nnnnmmmm0101	(FRn>FRm)?1:0→T	1	**比較結果**
FMOV.S	@(R0,Rm),FRn	1111nnnnmmmm0110	(R0+Rm)→FRn	1	—
FMOV.S	FRm,@(R0,Rn)	1111nnnnmmmm0111	(FRm)→(R0+Rn)	1	—
FMOV.S	@Rm,FRn	1111nnnnmmmm1000	(Rm)→FRn	1	—
FMOV.S	@Rm+,FRn	1111nnnnmmmm1001	(Rm)→FRn, Rm+4→Rm	1	—
FMOV.S	FRm,@Rn	1111nnnnmmmm1010	FRm→(Rn)	1	—
FMOV.S	FRm,@-Rn	1111nnnnmmmm1011	Rn-4→Rn, FRm→(Rn)	1	—
FMOV	FRm,FRn	1111nnnnmmmm1100	FRm→FRn	1	—
FMAC	FR0,FRm,FRn	1111nnnnmmmm1110	FR0×FRm+FRn→FRn	1	—

【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　**分岐しないときは** 1 **ステートになります。**

A.4　オペレーションコードマップ

オペレーションコードマップを示します。

表 A.56　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	Fx	0010	STC SR,Rn	STC GBR,Rn	STC VBR,Rn	
0000	Rm	Fx	0011	BSRF Rm		BRAF 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	Fx	1000	CLRT	SETT	CLRMAC	
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	Fx	1010	STS MACH,Rn	STS MACL,Rn	STS PR,Rn	STS FPUL,Rn/ STS FPSCR,Rn
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.LRm,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	DMULS.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	Fx	0010	STS.L MACH,@-Rn	STS.L MACL,@-Rn	STS.L PR,@-Rn	STS.L FPSCR,@-Rn STS.L FPUL,@-Rn
0100	Rn	Fx	0011	STC.L SR,@-Rn	STC.L GBR,@-Rn	STC.L VBR,@-Rn	
0100	Rn	Fx	0100	ROTL Rn		ROTCL Rn	

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	Fx	0101	ROTR Rn	CMP/PL Rn	ROTCR Rn	
0100	Rm	Fx	0110	LDS.L @Rm+,MACH	LDS.L @Rm+,MACL	LDS.L @Rm+,PR	LDS.L @Rm+,FPSCR LDS.L @Rm+,FPUL
0100	Rm	Fx	0111	LDC.L @Rm+,SR	LDC.L @Rm+,GBR	LDC.L @Rm+,VBR	
0100	Rn	Fx	1000	SHLL2 Rn	SHLL8 Rn	SHLL16 Rn	
0100	Rn	Fx	1001	SHLR2 Rn	SHLR8 Rn	SHLR16 Rn	
0100	Rm	Fx	1010	LDS Rm,MACH	LDS Rm,MACL	LDS Rm,PR	LDS Rm,FPSCR LDS Rm,FPUL
0100	Rm/Rn	Fx	1011	JSR @Rm	TAS.B @Rn	JMP @Rm	
0100	Rm	Fx	1100				
0100	Rm	Fx	1101				
0100	Rm	Fx	1110	LDC Rm,SR	LDC Rm,GBR	LDC Rm,VBR	
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.WRm,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)		
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 label:8		BF label:8
1000	11MD	imm/disp			BT/S label:8		BF/S lable:8
1001	Rn	disp		MOV.W @(disp:8,PC),Rn			
1010	disp			BRA label:12			
1011	disp			BSA label:12			
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

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	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	**********		**浮動小数点命令**			

B. パイプラインの動作と競合

SH-2E CPU では、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 命令が 2 ステート以上必要になるのは、分岐命令などで分岐先アドレスを変更する場合、MA と IF との競合を発生しステート数が増加する場合、レジスタ間競合が行う場合などです。競合の種類と命令との対応について、実行ステートとステージ段数を**表** B.1 および**表** B.2 に示します。競合がない命令、競合がなく 2 ステート以上必要とする命令もあわせて示します。

競合についてまとめると次のようになります。

(a) CPU 命令の場合

（1）レジスタ間の演算、転送は1ステートで実行し競合は発生しません。
（2） 競合は発生しませんが2ステート以上必要とする命令があります。
（3） 競合が発生し、ステートが増加します。競合の組み合せは次のようになります。
（a）MAはIFと競合することがあります。
（b） MAはIFと競合し、かつメモリロードの競合を起こすことがあります。
（c） MAはIFと競合し、かつ乗算器の競合を起こすことがあります。
（d） MAはIFと競合し、メモリロードの競合を起こし、かつ乗算器の競合を起こすことがあります。

(b) 浮動小数点命令および FPU に関する CPU 命令の場合

（1）FCMP命令では競合は発生しません。
（2）FR0～FR15、FPULのストア命令ではMAはIFと競合します。
（3）FDIVを除く浮動小数点演算命令、浮動小数点レジスタ-レジスタ間転送命令、および浮動小数点レジスタ-イミディエイト命令では、次命令がデスティネーションレジスタをリードする命令のとき競合が発生します。
（4）FR0～FR15、FPULのロード命令ではMAはIFと競合し、次命令がデスティネーションレジスタをリードする命令のとき競合が発生します。
（5）STS FPUL,Rn命令、およびSTS FPSCR,Rn命令では次命令にRnを使う命令を置くと競合が発生します。
（6）FPSCRのロード命令では**図**8.11に示した競合が発生します。
（7）FPSCRのストア命令では**図**8.12に示した競合が発生し、MAはIFと競合します。
（8）FDIV命令では**図**8.13に示した競合が発生します。

表 B.1　競合の種類と命令との対応（CPU 命令）

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
			システム制御 ALU 命令
	2	3	無条件分岐
	3/1	3	条件分岐
	3	3	SLEEP 命令
	4	5	RTE 命令
	8	9	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）
	2	4	STC.L 命令
	3	6	メモリ論理演算
	4	6	TAS 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	メモリロード命令 LDS.L 命令（PR）
	3	5	LDC.L 命令
• 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*1	6	乗算命令
	3/(2)*1	7	積和命令
	3/(2〜4)*1	9	倍精度積和命令（SH-2 CPU のみ）
	2〜4*1	9	倍精度乗算命令（SH-2 CPU のみ）
• 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	1	5	MAC→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。

表 B.2 競合の種類と命令との対応（浮動小数点命令および FPU に関する CPU 命令）

競合	実行ステート	ステージ段数	命令	
なし	1	3（FPU パインプライン） 3（CPU パインプライン）	FCMP/EQ FCMP/GT	FRm,FRn FRm,FRn
• CPU パイプラインの MA は IF と競合します	1	4（FPU パインプライン） 4（CPU パインプライン）	STS.L FMOV.S FMOV.S FMOV.S	FPUL,@-Rn FRm,@Rn FRm,@-Rn FRm,@(R0,Rn)
• 次命令がデスティネーションレジスタをリードする命令のとき、競合が発生します	1	5（FPU パインプライン） 3（CPU パインプライン）	FLDS FMOV FSTS FLDI0 FLDI1 FABS FADD FLOAT FMAC FMUL FNEG FSUB FTRC	FRm,FPUL FRm,FRn FPUL,FRn FRn FRn FRn FRm,FRn FPUL,FRn FR0,FRm,FRn FRm,FRn FRn FRm,FRn FRm,FPUL
• 次命令がデスティネーションレジスタをリードする命令のとき、競合が発生します • CPU パイプラインの MA は IF と競合します	1	5（FPU パインプライン） 4（CPU パインプライン）	LDS LDS.L FMOV.S FMOV.S FMOV.S	Rm,FPUL @Rm+,FPUL @Rm,FRn @Rm+,FRn @(R0,Rm),FRn
• 次命令に Rn を使う命令を置くと競合します • CPU パイプラインの MA は IF と競合します	1	4（FPU パインプライン） 5（CPU パインプライン）	STS	FPUL,Rn
• 図 8.11 に示したように競合が発生します	1	5（FPU パインプライン） 4（CPU パインプライン）	LDS LDS.L	Rm,FPSCR @Rm+,FPSCR
• 図 8.12 に示したように競合が発生します • 次命令に Rn を使う命令を置くと競合します • CPU パイプラインの MA は IF と競合します	1	4（FPU パインプライン） 5（CPU パインプライン）	STS	FPSCR,Rn
• 図 8.12 に示したように競合が発生します • CPU パイプラインの MA は IF と競合します	1	4（FPU パインプライン） 4（CPU パインプライン）	STS.L	FPSCR,@-Rn
• 図 8.13 に示したように競合が発生します	13	17（FPU パインプライン） 3（CPU パインプライン）	FDIV	FRm,FRn

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-2E

発行年月日　1998 年 12 月　　第 1 版
　　2006 年 5 月 30 日　Rev.2.00
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

© 2006. Renesas Technology Corp., All rights reserved. Printed in Japan.

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

http://www.renesas.com

本社	〒100-0004	千代田区大手町2-6-2 (日本ビル)	(03) 5201-5350
京浜支社	〒212-0058	川崎市幸区鹿島田890-12 (新川崎三井ビル)	(044) 549-1662
西東京支社	〒190-0023	立川市柴崎町2-2-23 (第二高島ビル2F)	(042) 524-8701
東北支社	〒980-0013	仙台市青葉区花京院1-1-20 (花京院スクエア13F)	(022) 221-1351
いわき支店	〒970-8026	いわき市平小太郎町4-9 (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	大阪市中央区伏見町4-1-1 (明治安田生命大阪御堂筋ビル)	(06) 6233-9500
北陸支社	〒920-0031	金沢市広岡3-1-1 (金沢パークビル8F)	(076) 233-5980
広島支店	〒730-0036	広島市中区袋町5-25 (広島袋町ビルディング8F)	(082) 244-2570
鳥取支店	〒680-0822	鳥取市今町2-251 (日本生命鳥取駅前ビル)	(0857) 21-1915
九州支社	〒812-0011	福岡市博多区博多駅前2-17-1 (ヒロカネビル本館5F)	(092) 481-7695

■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　E-Mail: csc@renesas.com

SH-2E
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

RJJ09B0344-0200