User's Manual

32

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-2A、SH2A-FPU

ユーザーズマニュアル　ソフトウェア編

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.4.00　2011.02

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

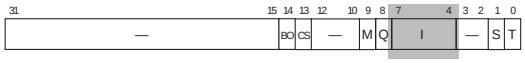
   高品質水準：輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   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。

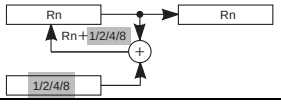
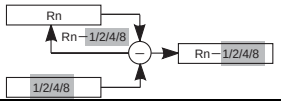
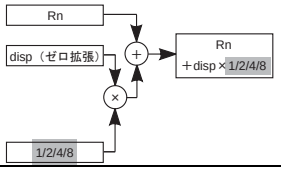
本版で修正または追加された箇所

<table>
<tr><th>項目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>2.2.1 汎用レジスタ</td><td>2-2</td><td>修正
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、復帰は、R15 を用いてスタックを参照し行います。</td></tr>
<tr><td>（1）ステータスレジスタ SR</td><td>2-3</td><td>修正
（32 ビット、初期値 = 0000 0000 0000 0000 0000 00XX 1111 00XX（X＝不定））
 

・I：割り込みマスクレベル</td></tr>
<tr><td>2.2.3 システムレジスタ</td><td>2-4</td><td>修正
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 は現在実行中の命令の 4 バイト先を示します。</td></tr>
<tr><td>（2）プロシージャレジスタ PR</td><td>2-4</td><td>修正
（2）プロシージャレジスタ PR（32 ビット、初期値=不定）</td></tr>
<tr><td>（3）プログラムカウンタ PC</td><td>2-4</td><td>修正
PC は現在実行中の命令の 4 バイト先を示します。</td></tr>
<tr><td>（2）浮動小数点ステータス／コントロールレジスタ FPSCR</td><td>2-7</td><td>修正
【注】SH-2A/SH2A-FPU では、FPU エラーは発生しません。</td></tr>
<tr><td>2.3.2 メモリ上でのデータ形式</td><td>2-8</td><td>修正
データフォーマットは、ビッグエンディアンのバイト順のみ選択できます。</td></tr>
<tr><td>図 2.4 メモリ上のデータ形式</td><td>2-8</td><td>修正
 
</td></tr>
<tr><td>表 3.2 例外要因検出と例外処理開始タイミング</td><td>3-2</td><td>修正
<table>
<tr><th colspan="2">例外処理</th><th>要因検出および処理開始タイミング</th></tr>
<tr><td rowspan="2">命令</td><td>整数除算例外</td><td>ゼロによる除算例外、または負の最大値（H'80000000）を－1 で除算することによるオーバフロー例外が検出されると開始される。</td></tr>
<tr><td>FPU 例外</td><td>浮動小数点演算命令の無効演算例外（IEEE754 規定）、ゼロによる除算例外、オーバフロー、アンダフローまたは不正確例外により開始される。また、FPSCR の QIS ビットがセットされているとき、qNaN もしくは±∞を浮動小数点演算命令のソースに入力すると開始される。</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>項目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>3.2.3 マニュアルリセット</td><td>3-5</td><td>削除
バス権解放中や DMAC バースト転送中にマニュアルリセットが発生すると、CPU がバス権を獲得するまでマニュアルリセット例外処理は保留されます。~~ただし、マニュアルリセットが発生してからバスサイクルの終了までの期間が内部マニュアルリセット期間の一定サイクル以上であると、内部マニュアルリセット要因は保留されずに無視され、マニュアルリセット例外処理は発生しません。~~詳細は、各製品のハードウェアマニュアルの「例外処理」に記載されている「マニュアルリセット」を参照してください。</td></tr>
<tr><td>3.3.2 アドレスエラー例外処理</td><td>3-6</td><td>追加
アドレスエラーが発生すると、アドレスエラーを起こしたバスサイクルが終了し*、実行中の命令が完了してから…。
【注】* データ読み出し／書き込みによるアドレスエラー時。
命令フェッチによるアドレスエラーは、上記動作（3）の終了までにアドレスエラーを起こしたバスサイクルが終了しない場合、該当バスサイクル終了まで、CPU は再度アドレスエラー例外処理を開始します。</td></tr>
<tr><td>3.6.3 割り込み例外処理</td><td>3-9</td><td>修正
アドレスエラー、NMI 割り込み、UBC 割り込み、命令による例外処理の場合、レジスタバンクへの退避は行われません。</td></tr>
<tr><td>3.7.1 命令による例外の種類</td><td>3-9</td><td>修正
例外処理を起動する命令には、表 3.8 に示すように、トラップ命令、スロット不当命令、一般不当命令、整数除算例外、FPU 例外があります。</td></tr>
<tr><td>表 3.8 命令による例外の種類</td><td>3-10</td><td>修正
<table>
<tr><th>種類</th><th>要因となる命令</th><th>備考</th></tr>
<tr><td>FPU 例外</td><td>IEEE754 規格で定義された無効演算例外またはゼロによる除算例外を引き起こす命令、オーバフロー、アンダフローおよび不正確例外を引き起こす可能性のある命令</td><td>FADD、FSUB、FMUL、FDIV、FMAC、FCMP/EQ 、 FCMP/GT 、 FLOAT 、 FTRC、FCNVDS、FCNVSD、FSQRT</td></tr>
</table></td></tr>
<tr><td>3.7.4 一般不当命令</td><td>3-11</td><td>修正
また、レジスタバンク非搭載品のレジスタバンク関連命令も未定義コードとして扱われ、遅延分岐命令の直後（遅延スロット）以外に配置された場合、この命令がデコードされると一般不当命令例外処理が開始されます。</td></tr>
<tr><td>3.7.5 整数除算例外</td><td>3-11</td><td>修正
3.7.5 整数除算例外
整数除算命令がゼロによる除算を実行した場合…。
（1）発生した整数除算~~命令~~例外に対応する例外サービスルーチン開始アドレスを、例外処理ベクタテーブルから取り出します。</td></tr>
</table>

項目	ページ	修正箇所
3.7.6 FPU 例外	3-11、 3-12	修正 3.7.6 FPU 例外 浮動小数点ステータスレジスタ（FPSCR）の FPU 例外イネーブルフィールド（Enable）中の V、Z、O、U または I ビットがセットされているとき、FPU 例外処理が発生します。これは浮動小数点演算命令が IEEE754 規格で定義された無効演算例外、ゼロによる除算例外、オーバフロー（可能性のある命令）、アンダフロー（可能性のある命令）および不正確例外（可能性のある命令）を引き起こしたことを示します。 FPU 例外処理の発生要因となる浮動小数点演算命令には以下があります。 FADD、FSUB、FMUL、FDIV、FMAC、FCMP/EQ、FCMP/GT、FLOAT、FTRC、FCNVDS、FCNVSD、FSQRT 該当する FPU 例外イネーブルビット（Enable）がセットされているときのみ、FPU 例外処理が発生します。FPU が浮動小数点演算による例外要因を検出すると、FPU の動作は中断されて CPU に FPU 例外処理の発生を通知します。CPU は例外処理を開始すると次のように動作します。 （1）発生した FPU 例外処理に対応する例外サービスルーチン開始アドレスを例外処理ベクタテーブルから取り出します。 （2）ステータスレジスタ（SR）をスタックに退避します。 （3）プログラムカウンタ（PC）をスタックに退避します。退避する PC の値は最後に実行した命令の次の命令の先頭アドレスです。 （4）例外処理ベクタテーブルから取り出したアドレスへジャンプして、プログラムの実行を開始します。このときのジャンプは遅延分岐ではありません。 FPSCR の例外フラグフィールド（Flag）は、FPU 例外処理が受け付けられたか否かにかかわらず常に更新され、ユーザが明示的に命令でクリアするまでセットされたままです。FPSCR の要因フィールド（Cause）は浮動小数点演算命令が実行されるごとに変化します。 また、FPSCR レジスタの FPU 例外イネーブルフィールド（Enable）中の V ビットがセットされ、かつ FPSCR の QIS ビットがセットされているとき、qNaN もしくは±∞を浮動小数点演算命令のソースに入力すると FPU 例外処理が発生します。
3.10.4 割り込みマスクビット変更による割り込み制御	3-14	追加

<table>
<tr><th>項目</th><th>ページ</th><th>修正箇所</th></tr>
<tr><td>表 4.7 アドレッシングモードと実効アドレス</td><td>4-4、
4-5</td><td>修正
<table>
<tr><th>命令フォーマット</th><th>実効アドレスの計算方法</th><th>計算式</th></tr>
<tr><td>@Rn+</td><td>実効アドレスはレジスタ Rn の内容です。命令実行後 Rn に定数を加算します。定数はオペランドサイズがバイトのとき 1、ワードのとき 2、ロングワードのとき 4、クワッドワードのとき 8 です。
 
</td><td>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
クワッドワード：
Rn＋8→Rn</td></tr>
<tr><td>@-Rn</td><td>実効アドレスは、あらかじめ定数を減算したレジスタ Rn の内容です。定数はバイトのとき 1、ワードのとき 2、ロングワードのとき 4、クワッドワードのとき 8 です。
 
</td><td>バイト：Rn－1→Rn
ワード：Rn－2→Rn
ロングワード：Rn－4→Rn
クワッドワード：
Rn－8→Rn
（計算後の Rn で命令実行）</td></tr>
<tr><td>@(disp:12,Rn)</td><td>実効アドレスはレジスタ Rn に 12 ビットディスプレースメント disp を加算した内容です。disp はゼロ拡張します。
 
</td><td>バイト：Rn＋disp
ワード：Rn＋disp×2
ロングワード：Rn＋disp×4
クワッドワード：
Rn＋disp×8</td></tr>
</table></td></tr>
<tr><td>表 5.3 算術演算命令</td><td>5-10</td><td>修正
命令 MULR R0, Rn の T ビット：－
命令 SUBV Rm, Rn の T ビット：アンダフロー</td></tr>
<tr><td>表 5.9 FPU に関する CPU 命令</td><td>5-17</td><td>修正
Rn-=4, FPSCR→(Rn)</td></tr>
<tr><td>（4）使用例</td><td>6-50</td><td>追加
<pre>MOV.L @(2*, R0), R1   ;実行前 @(R0+8)=H'12345670
                        ;実行後 R1=H'12345670</pre>【注】*「6.3.19（2）注意」を参照してください。</td></tr>
<tr><td>（4）使用例</td><td>6-54</td><td>修正
<pre>MOVI20S H'7FFFF00, R0   ;実行前 R0=H'00000000
                              ;実行後 R0=H'07FFFF00</pre></td></tr>
<tr><td>（2）動作内容</td><td>6-122</td><td>修正
<pre>if (MACH&0x00008000);
~~else~~   Res2+=MACH|0xFFFF0000;
else   Res2+=MACH&0x00007FFF;</pre></td></tr>
</table>

項目	ページ	修正箇所
(4) 使用例	6-135	修正 100A BRA NEXT ;遅延分岐命令 100C MOV.L @(4,PC),R3 ;R3=H'12345678 100E IMM.data.w H'9ABC ;
(2) 動作内容	6-205	修正 case NZERO: zero(n,(sign_of(0)^sign_of(m))& sign_of(n)); break;

	6-206、
	6-207
		修正
		case NZERO: zero(n,(sign_of(0)^ sign_of(m))& sign_of(n));
		break;
		default: break;
		} break;
		} break;
		case PINF :
		case NINF : switch (data_type_of(m)){
		case PZERO:
		case NZERO:invalid(n); break;
		default: switch (data_type_of(n)){
		case qNaN: gnan(n); break;
		case PINF:
		case NINF: if(sign_of(0)^ sign_of(m)^ sign_of(n))
		invalid(n);
		else inf(n,sign_of(0)^ sign_of(m)); break;
		default: inf(n,sign_of(0)^ sign_of(m)); break;

(2) バンク番号レジスタ IBNR	7-3	修正 次に退避されるバンク番号を示します。
(3) システム制御 ALU 命令	8-53	修正 典型的な、CS ビット読み出し時のパイプラインを示しておきます。
A. SH-2A/SH2A-FPU 並列実行性	付録-4	修正 ・マルチサイクル命令は最初のサイクルと最後のサイクルで並列実行を行います。 ・FPU 命令は、SH4 の分類を踏襲する（①LS タイプ②FE タイプ③CO タイプ）。 新規命令である 32 ビット FMOV 命令は①LS タイプに分類します。 ・32 ビット命令は、原則、前の命令がマルチサイクル命令であれば並列実行可能。 次の命令との並列実行は不可。ただし、メモリ-Tbit 間ビット操作命令同士の組み合わせは並列実行可能。 ・MOVMU.L,MOVML.L 命令は、前の命令との並列実行は不可。 ・遅延分岐命令と遅延スロットとの並列実行は不可。 ・マルチサイクル命令：
(1) MOVI20 の使用方法	付録-5	修正 MOVI20~~S~~では符号拡張を行います。

目次

1. 概要

1.1 SH-2A/SH2A-FPU の特長

SH-2A/SH2A-FPU は SH-1、SH-2、SH-2E マイクロコンピュータとのオブジェクトコードレベルでの上位互換性を特長とする 32 ビット RISC（縮小命令セットコンピュータ）マイコンで、FPU なしの SH-2A と、FPU ありの SH2A-FPU があります。基本命令を 16 ビット長とすることにより、コード効率、性能、使い勝手を向上させることができます。

SH-2A/SH2A-FPU の特長を**表 1.1** に示します。

表 1.1 SH-2A/SH2A-FPU の特長

項目	特長
CPU	• ルネサスオリジナルアーキテクチャ • 32 ビット内部データバス • 汎用レジスタアーキテクチャ： – 16 本の 32 ビット汎用レジスタ – 4 本の 32 ビットコントロールレジスタ – 4 本の 32 ビットシステムレジスタ – 高速割り込み応答のためのレジスタバンク • RISC タイプ命令セット（SH シリーズと上位互換性）： – 命令長： コードの効率改善のための 16 ビット基本命令と性能、使い勝手向上のための 32 ビット命令 – ロードストアアーキテクチャ – 遅延分岐命令 – C 言語に基づく命令セット • FPU を含む 2 命令同時実行型スーパスカラ • 命令実行時間： 最大 2 命令／サイクル • アドレス空間： 4G バイト • 乗算器内蔵 • 5 段パイプライン • ハーバードアーキテクチャ

項目	特長
浮動小数点ユニット（FPU）	• 浮動小数点コプロセッサ内蔵 • 単精度（32 ビット）および倍精度（64 ビット）をサポート • IEEE754 に準拠したデータタイプおよび例外をサポート • 丸めモード： 近傍および 0 方向への丸め • 非正規化数の扱い： 0 への切り捨て • 浮動小数点レジスタ – 16 本の 32 ビット浮動小数点レジスタ （単精度 x16 ワードまたは倍精度 x8 ワード） – 2 本の 32 ビット浮動小数点システムレジスタ • FMAC（乗算およびアキュムレート）命令をサポート • FDIV（除算）／FSQRT（平方根）命令をサポート • FLDI0／FLDI1（ロード定数 0/1）命令をサポート • 命令実行時間 – レイテンシ（FMAC/FADD/FSUB/FMUL）：3 サイクル（単精度）、 8 サイクル（倍精度） – ピッチ（FMAC/FADD/FSUB/FMUL）： 1 サイクル（単精度）、6 サイクル（倍精度） 【注】FMAC は単精度に対してのみサポートしています。 • 5 段パイプライン

2. プログラミングモデル

2.1 データフォーマット

SH-2A/SH2A-FPU でサポートしているデータフォーマットを図 2.1 に示します。

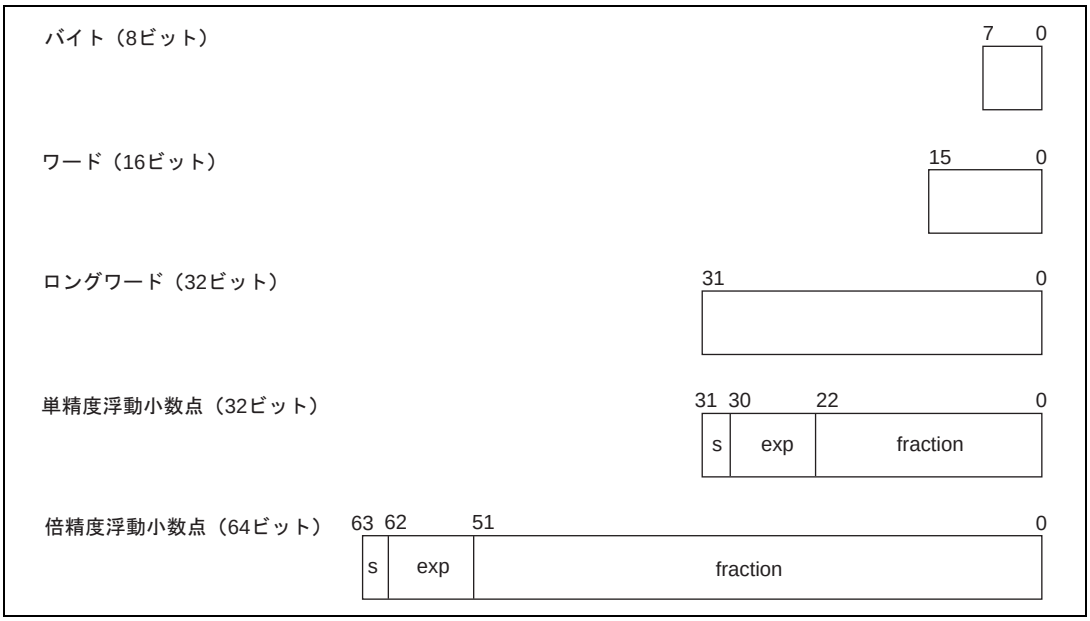


図 2.1　データフォーマット

2.2 レジスタの構成

2.2.1 汎用レジスタ

図 2.2 に汎用レジスタを示します。汎用レジスタは、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、復帰は、R15 を用いてスタックを参照し行います。

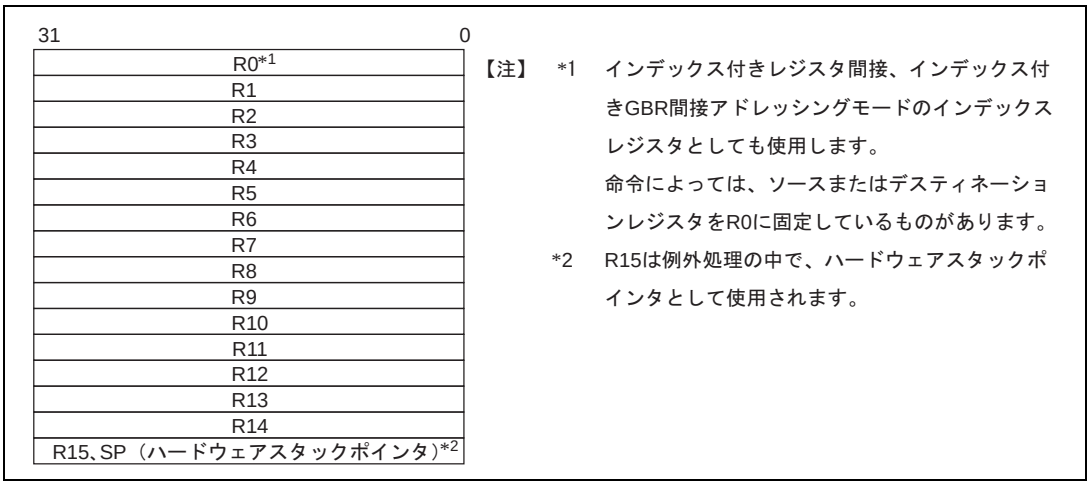


図 2.2 汎用レジスタ

2.2.2　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：Status register）、グローバルベースレジスタ（GBR : Global base register）、ベクタベースレジスタ（VBR : Vector base register）、ジャンプテーブルベースレジスタ（TBR：Jump table base register）の 4 本があります。

SR レジスタは各種命令の処理の状態を表します。

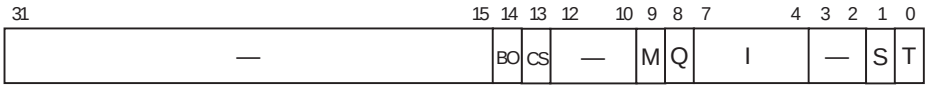
GBR レジスタは GBR 間接アドレッシングモードのベースアドレスとして使用し、内蔵周辺モジュールのレジスタのデータ転送などに使用します。

VBR レジスタ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。

TBR レジスタは関数テーブル領域のベースアドレスとして使用します。

(1)　**ステータスレジスタ　SR**

（32 ビット、初期値 =　0000 0000 0000 0000 0000 00XX 1111 00XX(X＝不定)）



【注】—：予約ビット。読み出すと常に 0 が読み出されます。書き込む値も常に 0 にしてください。

- BO：レジスタバンクがオーバフローしていることを示します。
- CS： CLIP 命令の実行で、飽和上限値を上回った、または飽和下限値を下回ったことを示します。
- M、Q：DIV0S、DIV0U、DIV1 命令で使用します。
- I：割り込みマスクレベル
- S：MAC 命令の飽和動作を指定します。
- T：真／偽条件、またはキャリ／ボロービット

(2)　**グローバルベースレジスタ　GBR（32 ビット、初期値=不定）**

GBR は GBR 参照 MOV 命令のベースアドレスとして参照されます。

(3)　**ベクタベースレジスタ　VBR（32 ビット、初期値=H'0000 0000）**

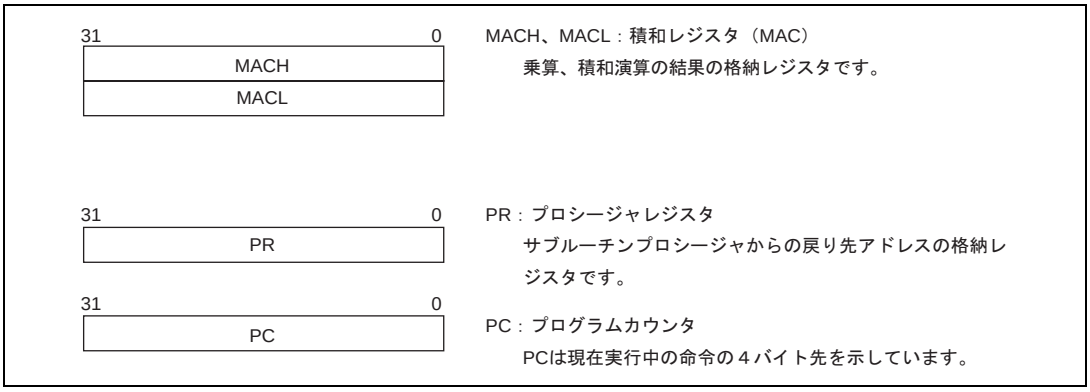
VBR は例外および割り込み発生時、分岐先のベースアドレスとして参照されます。

(4)　**ジャンプテーブルベースレジスタ　TBR（32 ビット、初期値=不定）**

テーブル参照サブルーチンコール命令 JSR/N @@(disp8,TBR)で、メモリに配置された関数テーブルの先頭アドレスとして参照します。

2.2.3 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 は現在実行中の命令の 4 バイト先を示します。



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

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

(2) プロシージャレジスタ PR（32 ビット、初期値=不定）

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

(3) プログラムカウンタ PC（32 ビット、初期値はベクタテーブル中の PC の値）

PC は現在実行中の命令の 4 バイト先を示します。

2.2.4　浮動小数点レジスタ

図 2.3 に浮動小数点レジスタを示します。16 本の 32 ビット浮動小数点レジスタ FPR0～FPR15 があります。この 16 本のレジスタは FR0～FR15、DR0/2/4/6/8/l0/12/14 として参照されます。FPRn と参照名の対応は FPSCR の PR ビットと SZ ビットによって決まります。図 2.3 を参照してください。

(1)　浮動小数点レジスタ　FPRn (16 レジスタ)

FPR0, FPR l, FPR2, FPR3, FPR4, FPR5, FPR6, FPR7,
FPR8, FPR9, FPR10, FPR11, FPR12, FPR13, FPR14, FPR15

(2)　単精度浮動小数点レジスタ　FRi (16 レジスタ)

FR0～FR15 は FPR0～FPR15 に割り当てられます。

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

DR レジスタは、2 つの FR レジスタから構成されます。
DR0 = {FPR0, FPR1} ,DR2 = {FPR2, FPR3 },
DR4 = {FPR4, FPR5} , DR6 = {FPR6, FPR7} ,
DR8 = {FPR8, FPR9}, DR10 = {FPR10, FPR11},
DR12 = {FPR12, FPR13}, DR14 = {FPR14, FPR15}

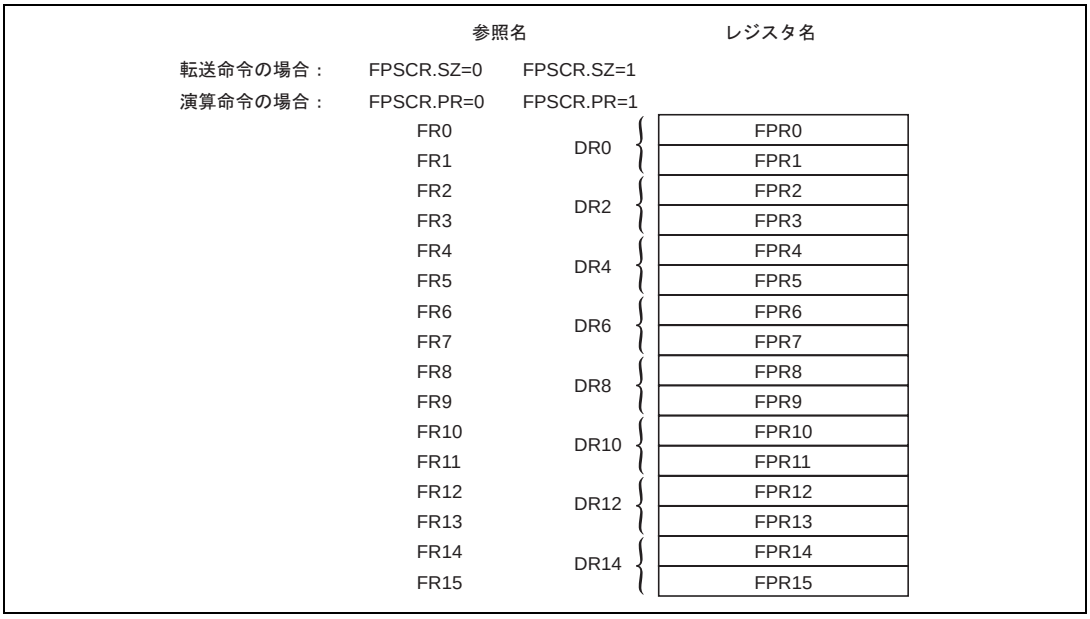


図 2.3　浮動小数点レジスタ

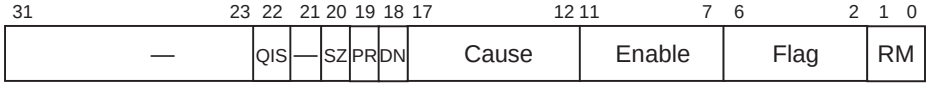
【プログラミング上の注意】
リセット後の FPR0～FPR15 の値は不定です。

2.2.5 浮動小数点システムレジスタ

(1) 浮動小数点通信レジスタ FPUL（32 ビット、初期値=不定）

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

(2) 浮動小数点ステータス／コントロールレジスタ FPSCR
（32 ビット、初期値=H'0004 0001）



QIS：qNaNあるいは±∞をsNaNとして扱う。FPSCRのイネーブルV=1のときのみ有効。
QIS=0：qNaNあるいは±∞として処理。
QIS=1：例外発生（sNaNと同様に処理）。
SZ：転送サイズモード
SZ=0：FMOV命令のデータサイズは32ビットです。
SZ=1：FMOV命令のデータサイズは32ビットペア（64ビット）です。
PR：精度モード
PR=0：浮動小数点命令を単精度で実行します。
PR=1：浮動小数点命令を倍精度で実行します（倍精度がサポートされていない命令の結果は未定義です。）
DN：非正規化モード（常に1です）
DN=1：非正規化数を0として扱います。

Cause：FPU例外要因フィールド
Enable：FPU例外イネーブルフィールド
Flag：FPU例外フラグフィールド

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

FPU 演算命令を実行すると、FPU 例外要因フィールドは最初に 0 に設定されます。次に FPU 例外が発生すると、FPU 例外要因フィールドと FPU 例外フラグフィールドの該当ビットが 1 にセットされます。

FPU 例外フラグフィールドは、FPU 例外フラグフィールドが最後にクリアされたそれ以降に発生した例外のステータスを保持します。

RM：丸めモード
RM=00：近傍への丸め
RM=01：0方向への丸め
RM=10：予約
RM=11：予約

ビット21、23～31：予約

【注】 SH-2A/SH2A-FPU では、FPU エラーは発生しません。

2.2.6　レジスタバンク

汎用レジスタ R0～R14、コントロールレジスタ GBR、システムレジスタ MACH、MACL、PR の 19 本の 32 ビットレジスタは、レジスタバンクを使って、高速なレジスタ退避、復帰を行うことが可能です。バンクへの退避は、CPU がレジスタバンクを使用する割り込みを受け付けた後、自動的に行われます。バンクからの復帰は、割り込み処理ルーチンで RESBANK 命令を発行することで実行されます。

詳細については**「第 7 章　レジスタバンク」**を参照してください。

2.2.7　レジスタの初期値

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）、BO、CS は 0、予約ビットは 0、その他は不定
	GBR、TBR	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	FPR0～FPR15	不定
浮動小数点システムレジスタ	FPUL	不定
	FPSCR	H'00040001

2.3 データ形式

2.3.1 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。

2.3.2 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。メモリは 8 ビットのバイト、16 ビットのワード、32 ビットのロングワードいずれの形でもアクセスすることができます。32 ビットに満たないメモリオペランドは符号拡張もしくはゼロ拡張されてレジスタに格納されます。

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

データフォーマットは、ビッグエンディアンのバイト順のみ選択できます。

メモリ上のデータ形式を**図 2.4** に示します。

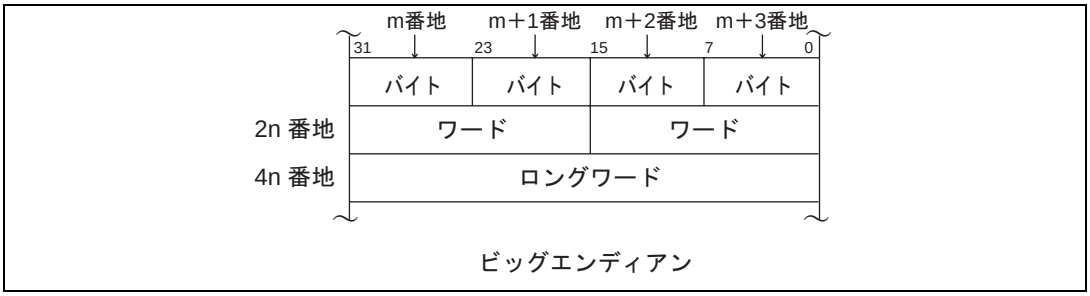


図 2.4 メモリ上のデータ形式

2.3.3 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。

20 ビットのイミディエイトデータは 32 ビット長の転送命令 MOVI20 および MOVI20S のコードの中に配置します。MOVI20 命令は、イミディエイトを符号拡張してデスティネーションレジスタに格納します。MOVI20S 命令は、イミディエイトを上位に 8 ビットシフトし、符号拡張してデスティネーションレジスタに格納します。

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。

具体例については、**「第 4 章　命令の特長」**の**「4.1（10）イミディエイトデータ」**を参照してください。

2.4　処理状態

CPU の処理状態には、リセット状態、例外処理状態、バス権解放状態、プログラム実行状態、低消費電力状態の 5 種類があります。状態間の遷移を**図 2.5** に示します。

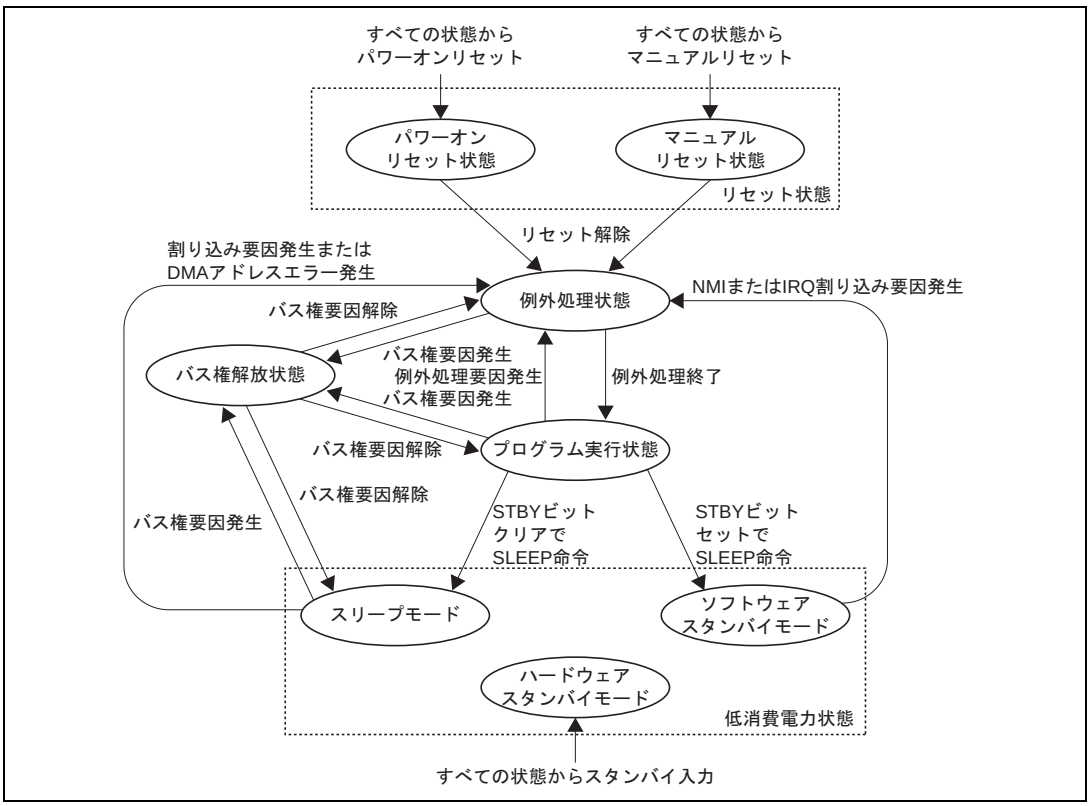


図 2.5　処理状態の状態遷移図

(1)　リセット状態

CPU がリセットされている状態です。リセットには、パワーオンリセットとマニュアルリセットの 2 種類があります。詳細は、ハードウェアマニュアルを参照してください。

(2)　例外処理状態

リセットや割り込みなどの例外処理要因によって、CPU が処理状態の流れを変えるときの過渡的な状態です。

リセットの場合は、例外処理ベクタテーブルからプログラムカウンタ（PC）の初期値としての実行開始アドレスとスタックポインタ（SP）の初期値を取り出しそれぞれ格納し、スタートアドレスに分岐してプログラムの実行を開始します。

割り込みなどの場合は、SP を参照して、PC とステータスレジスタ（SR）をスタック領域に退避します。例外処理ベクタテーブルから例外サービスルーチンの開始アドレスを取り出し、そのアドレスに分岐してプログラムの実行を開始します。

その後処理状態はプログラム実行状態となります。

(3) プログラム実行状態

CPU が順次プログラムを実行している状態です。

(4) 低消費電力状態

CPU の動作が停止し消費電力が低い状態です。スリープ命令でスリープモード、またはソフトウェアスタンバイモードになります。ハードウェアスタンバイ入力が入るとハードウェアスタンバイモードになります。

(5) バス権解放状態

CPU がバス権を要求したデバイスにバスを解放している状態です。

【注】 処理状態の詳細は各製品のハードウェアマニュアルを参照してください。

3. 例外処理

3.1 概要

3.1.1 例外処理の種類と優先順位

例外処理は、**表 3.1** に示すようにリセット、アドレスエラー、RAM エラー、レジスタバンクエラー、割り込み、および命令の各要因によって起動されます。例外要因には、**表 3.1** に示すように優先順位が設けられており、複数の例外要因が同時に発生した場合は、この優先順位に従って受け付けられ、処理されます。

表 3.1 例外要因の種類と優先順位

	例外処理	優先順位
リセット	パワーオンリセット	高
	マニュアルリセット	
アドレスエラー	CPU アドレスエラー	
	DMAC アドレスエラー	
RAM エラー	RAM エラー	
命令	FPU 例外	
	整数除算例外（0 除算）	
	整数除算例外（オーバフロー）	
レジスタバンクエラー	バンクアンダフロー	
	バンクオーバフロー	
割り込み	NMI	
	ユーザブレーク	
	H-UDI	
	外部割り込み（IRQ）	
	内蔵周辺モジュール	
命令	トラップ命令（TRAPA 命令）	
	一般不当命令（未定義コード）	
	スロット不当命令（遅延分岐命令*1 直後に配置された未定義コード（含 FPU モジュールスタンバイ時もしくは FPU 非搭載品における FPU 命令および FPU に関する CPU 命令、レジスタバンク非搭載品のレジスタバンク関連命令*2）、PC を書き換える命令*3、32 ビット命令*4、RESBANK 命令、DIVS 命令または DIVU 命令）	低

【注】 *1 遅延分岐命令 : JMP、JSR、BRA、BSR、RTS、RTE、BF/S、BT/S、BSRF、BRAF

*2 レジスタバンク関連命令: RESBANK、LDBANK、STBANK

*3 PC を書き換える命令 : JMP、JSR、BRA、BSR、RTS、RTE、BT、BF、TRAPA、BF/S、BT/S、BSRF、BRAF、JSR/N、RTV/N

*4 32 ビット命令 : BAND.B、BANDNOT.B、BCLR.B、BLD.B、BLDNOT.B、BOR.B、BORNOT.B、BSET.B、BST.B、BXOR.B、FMOV.S@disp12、FMOV.D@disp12、MOV.B@disp12、MOV.W@disp12、MOV.L@disp12、MOVI20、MOVI20S、MOVU.B、MOVU.W

3.1.2 例外処理の動作

各例外要因は**表 3.2** に示すタイミングで検出され、処理が開始されます。

表 3.2 例外要因検出と例外処理開始タイミング

例外処理		要因検出および処理開始タイミング
リセット	パワーオンリセット	パワーオンリセット条件の検出で開始される。
	マニュアルリセット	マニュアルリセット条件の検出で開始される。
アドレスエラー		命令のデコード時に検出され、この前までに実行中の命令が完了後開始される。
RAM エラー		
割り込み		
レジスタバンクエラー	バンクアンダフロー	レジスタバンクに退避が行われていないときに、RESBANK 命令を実行しようとすると開始される。
	バンクオーバフロー	割り込みコントローラでレジスタバンクオーバフロー例外を受け付けるように設定されており、レジスタバンクを使用する割り込みが発生し、CPU に受け付けられたとき、レジスタバンクのすべての領域にすでに退避が行われていたときに開始される。
命令	トラップ命令	TRAPA 命令の実行により開始される。
	一般不当命令	遅延分岐命令直後（遅延スロット）以外にある未定義コード（含 FPU モジュールスタンバイ時もしくは FPU 非搭載品における、FPU 命令および FPU に関する CPU 命令およびレジスタバンク非搭載品のレジスタバンク関連命令）がデコードされると開始される。
	スロット不当命令	遅延分岐命令直後（遅延スロット）に配置された未定義コード（含 FPU モジュールスタンバイ時もしくは FPU 非搭載品における、FPU 命令および FPU に関する CPU 命令およびレジスタバンク非搭載品のレジスタバンク関連命令）、PC を書き換える命令、32 ビット命令、RESBANK 命令、DIVS 命令または DIVU 命令がデコードされると開始される。
	整数除算例外	ゼロによる除算例外、または負の最大値（H'80000000）を－1 で除算することによるオーバフロー例外が検出されると開始される。
	FPU 例外	浮動小数点演算命令の無効演算例外（IEEE754 規定）、ゼロによる除算例外、オーバフロー、アンダフローまたは不正確例外により開始される。また、FPSCR の QIS ビットがセットされているとき、qNaN もしくは±∞を浮動小数点演算命令のソースに入力すると開始される。

例外処理が起動されると、CPU は次のように動作します。

(1) リセットによる例外処理

プログラムカウンタ（PC）とスタックポインタ（SP）の初期値を例外処理ベクタテーブル（PC、SP をそれぞれ、パワーオンリセット時に H'00000000 番地、H'00000004 番地、マニュアルリセット時に H'00000008 番地、H'0000000C 番地）から取り出します。例外処理ベクタテーブルについては、**「3.1.3 例外処理ベクタテーブル」**を参照してください。次にベクタベースレジスタ（VBR）を H'00000000 に、ステータスレジスタ（SR）の割り込みマスクビット（I3～I0）を H'F（1111）に、BO ビットおよび CS ビットを 0 に初期化します。また INTC の IBNR の BN ビットを 0 に初期化します。さらにパワーオンリセット時には、FPU 搭載品では FPSCR を H'00040001 に初期化します。例外処理ベクタテーブルから取り出した PC のアドレスからプログラムの実行を開始します。

(2) アドレスエラー、RAM エラー、レジスタバンクエラー、割り込み、命令による例外処理

SR と PC を R15 で示すスタック上に退避します。NMI および UBC 以外の割り込み例外処理で、レジスタバンクを使用する設定が行われている場合、汎用レジスタ R0～R14、コントロールレジスタ GBR、システムレジスタ MACH、MACL、PR および実行される割り込み例外処理のベクタテーブルアドレスオフセットを、レジスタバンクに退避します。アドレスエラー、RAM エラー、レジスタバンクエラー、NMI 割り込み、UBC 割り込み、命令による例外処理の場合、レジスタバンクへの退避は行われません。また、レジスタバンクのすべてのバンクに退避が行われていた場合には、レジスタバンクの代わりにスタックへの自動退避が行われます。この場合、割り込みコントローラにおいて、レジスタバンクオーバフロー例外を受け付けないように設定されている必要があります。レジスタバンクオーバフロー例外を受け付けるように設定されている場合には、レジスタバンクオーバフロー例外が発生します。割り込み例外処理の場合、割り込み優先レベルを SR の I3～I0 に書き込みます。アドレスエラー、RAM エラー、命令による例外処理の場合、I3～I0 ビットは影響を受けません。次に例外処理ベクタテーブルからスタートアドレスを取り出し、そのアドレスからプログラムの実行を開始します。

3.1.3 例外処理ベクタテーブル

例外処理実行前には、あらかじめ例外処理ベクタテーブルが、メモリ上に設定されている必要があります。例外処理ベクタテーブルには、例外サービスルーチンの開始アドレスを格納しておきます（リセット例外処理のテーブルには、PC と SP の初期値を格納しておきます）。

各例外要因には、それぞれ異なるベクタ番号とベクタテーブルアドレスオフセットが割り当てられています。ベクタテーブルアドレスは、対応するベクタ番号やベクタテーブルアドレスオフセットから算出されます。例外処理では、このベクタテーブルアドレスが示す例外処理ベクタテーブルから、例外サービスルーチンのスタートアドレスが取り出されます。

ベクタ番号とベクタテーブルアドレスオフセットを**表 3.3** に、ベクタテーブルアドレスの算出法を**表 3.4** に示します。

表 3.3 例外処理ベクタテーブル

例外要因		ベクタ番号	ベクタテーブルアドレスオフセット
パワーオンリセット	PC	0	H'00000000 ～ H'00000003
	SP	1	H'00000004 ～ H'00000007
マニュアルリセット	PC	2	H'00000008 ～ H'0000000B
	SP	3	H'0000000C ～ H'0000000F
一般不当命令		4	H'00000010 ～ H'00000013
RAM エラー		5	H'00000014 ～ H'00000017
スロット不当命令		6	H'00000018 ～ H'0000001B
（システム予約）		7	H'0000001C ～ H'0000001F
		8	H'00000020 ～ H'00000023
CPU アドレスエラー		9	H'00000024 ～ H'00000027
DMAC アドレスエラー		10	H'00000028 ～ H'0000002B
割り込み	NMI	11	H'0000002C ～ H'0000002F
	ユーザブレーク	12	H'00000030 ～ H'00000033
FPU 例外		13	H'00000034 ～ H'00000037
H-UDI		14	H'00000038～ H'0000003B
バンクオーバフロー		15	H'0000003C ～ H'0000003F
バンクアンダフロー		16	H'00000040 ～ H'00000043

例外要因	ベクタ番号	ベクタテーブルアドレスオフセット
整数除算例外（0除算）	17	H'00000044 ～ H'00000047
整数除算例外（オーバフロー）	18	H'00000048 ～ H'0000004B
（システム予約）	19 ・ 31	H'0000004C ～ H'0000004F ・ H'0000007C ～ H'0000007F
トラップ命令（ユーザベクタ）	32 ・ 63	H'00000080 ～ H'00000083 ・ H'000000FC ～ H'000000FF
外部割り込み(IRQ)、内蔵周辺モジュール*	64 ・ 511	H'00000100 ～ H'00000103 ・ H'000007FC ～ H'000007FF

【注】 * 外部割り込み、各内蔵周辺モジュール割り込みのベクタ番号とベクタテーブルオフセットは各製品のハードウェアマニュアルの「割り込みコントローラ」に記載されている「割り込み例外処理ベクタと優先順位」を参照してください。

表 3.4 例外処理ベクタテーブルアドレスの算出法

例外要因	ベクタテーブルアドレス算出法
リセット	ベクタテーブルアドレス ＝ （ベクタテーブルアドレスオフセット） ＝ （ベクタ番号）× 4
アドレスエラー、RAMエラー、レジスタバンクエラー、割り込み、命令	ベクタテーブルアドレス ＝ VBR＋（ベクタテーブルアドレスオフセット） ＝ VBR＋（ベクタ番号）× 4

【注】 VBR：ベクタベースレジスタ
ベクタテーブルアドレスオフセット：表 3.3 を参照
ベクタ番号：表 3.3 を参照

3.2 リセット

3.2.1 リセットの種類

リセットは最も優先順位の高い例外処理要因です。リセットには、パワーオンリセットとマニュアルリセットの 2 種類があります。パワーオンリセット、マニュアルリセットのどちらでも CPU 状態は初期化されます。FPU 状態はパワーオンリセットでは初期化され、マニュアルリセットでは初期化されません。内蔵周辺モジュール、PFC、IO ポートの状態については、各製品のハードウェアマニュアルを参照してください。

3.2.2 パワーオンリセット

パワーオンリセット条件が検出されるとパワーオンリセット状態になります。パワーオンリセット条件については、各製品のハードウェアマニュアルの**「例外処理」**に記載されている**「パワーオンリセット」**を参照してください。

パワーオンリセット状態を解除すると、パワーオンリセット例外処理が開始されます。このとき、CPU は次のように動作します。

(1) プログラムカウンタ（PC）の初期値（実行開始アドレス）を、例外処理ベクタテーブルから取り出します。
(2) スタックポインタ（SP）の初期値を、例外処理ベクタテーブルから取り出します。

(3) ベクタベースレジスタ(VBR)をH'00000000にクリアし、ステータスレジスタ(SR)の割り込みマスクビット（I3～I0）をH'F（1111）に、BOビットおよびCSビットを0に初期化します。またINTCのIBNRのBNビットを0に初期化します。さらに、FPU搭載品ではFPSCRをH'00040001に初期化します。

(4) 例外処理ベクタテーブルから取り出した値をそれぞれPCとSPに設定し、プログラムの実行を開始します。

なお、パワーオンリセット処理は、システムの電源投入時、必ず行うようにしてください。

3.2.3 マニュアルリセット

マニュアルリセット条件が検出されるとマニュアルリセット状態になります。マニュアルリセット条件については、各製品のハードウェアマニュアルの**「例外処理」**に記載されている**「マニュアルリセット」**を参照してください。

マニュアルリセット処理が開始されると、CPU は次のように動作します。

(1) プログラムカウンタ（PC）の初期値（実行開始アドレス）を、例外処理ベクタテーブルから取り出します。

(2) スタックポインタ（SP）の初期値を、例外処理ベクタテーブルから取り出します。

(3) ベクタベースレジスタ(VBR)をH'00000000にクリアし、ステータスレジスタ(SR)の割り込みマスクビット（I3～I0）をH'F（1111）に、BOビットおよびCSビットを0に初期化します。またINTCのIBNRのBNビットを0に初期化します。

(4) 例外処理ベクタテーブルから取り出した値をそれぞれPCとSPに設定し、プログラムの実行を開始します。

マニュアルリセット発生時、バスサイクルは保持されます。バス権解放中や DMAC バースト転送中にマニュアルリセットが発生すると、CPU がバス権を獲得するまでマニュアルリセット例外処理は保留されます。詳細は、各製品のハードウェアマニュアルの**「例外処理」**に記載されている**「マニュアルリセット」**を参照してください。

マニュアルリセットでは CPU および INTC の IBNR の BN ビットを初期化します。FPU やその他のモジュールは初期化されません。

3.3 アドレスエラー

3.3.1 アドレスエラー発生要因

アドレスエラーは、表 3.5 に示すように命令フェッチ、データ読み出し／書き込み時に発生します。

表 3.5 バスサイクルとアドレスエラー

バスサイクル		バスサイクルの内容	アドレスエラーの発生
種類	バスマスタ		
命令フェッチ	CPU	偶数アドレスから命令をフェッチ	なし（正常）
		奇数アドレスから命令をフェッチ	アドレスエラー発生
		内蔵周辺モジュール空間*以外から命令をフェッチ	なし（正常）
		内蔵周辺モジュール空間*から命令をフェッチ	アドレスエラー発生
		シングルチップモード時に外部メモリ空間から命令をフェッチ	アドレスエラー発生
データ読み出し／書き込み	CPU または DMAC	ワードデータを偶数アドレスからアクセス	なし（正常）
		ワードデータを奇数アドレスからアクセス	アドレスエラー発生
		ロングワードデータをロングワード境界からアクセス	なし（正常）
		ダブルロングワードデータをダブルロングワード境界からアクセス	なし（正常）
		ダブルロングワードデータをダブルロングワード境界以外からアクセス	アドレスエラー発生
		ロングワードデータを 16 ビットの内蔵周辺モジュール空間*でアクセス	なし（正常）
		ロングワードデータを 8 ビットの内蔵周辺モジュール空間*でアクセス	なし（正常）
		シングルチップモード時に外部メモリ空間をアクセス	アドレスエラー発生

【注】 * 内蔵周辺モジュール空間については、各製品のハードウェアマニュアルの「バスステートコントローラ」を参照してください。

3.3.2 アドレスエラー例外処理

アドレスエラーが発生すると、アドレスエラーを起こしたバスサイクルが終了し*、実行中の命令が完了してからアドレスエラー例外処理が開始されます。このとき、CPU は次のように動作します。

(1) 発生したアドレスエラーに対応する例外サービスルーチン開始アドレスを、例外処理ベクタテーブルから取り出します。
(2) ステータスレジスタ（SR）をスタックに退避します。
(3) プログラムカウンタ（PC）をスタックに退避します。退避するPCの値は、最後に実行した命令の次命令の先頭アドレスです。
(4) 例外処理ベクタテーブルから取り出したアドレスへジャンプして、プログラムの実行を開始します。このときのジャンプは遅延分岐ではありません。

【注】 *データ読み出し／書き込みによるアドレスエラー時。
命令フェッチによるアドレスエラーは、上記動作（3）の終了までにアドレスエラーを起こしたバスサイクルが終了しない場合、該当バスサイクル終了まで、CPU は再度アドレスエラー例外処理を開始します。

3.4 RAM エラー

3.4.1 RAM エラー発生要因

RAM エラーは内蔵 RAM のリードアクセス時に、ソフトエラーが生じると発生します。詳細は、各製品のハードウェアマニュアルの「**例外処理**」に記載されている「**RAM エラー**」を参照してください。

3.4.2 RAM エラー例外処理

RAM エラーが発生すると、RAM エラーを起こしたバスサイクルが終了し、実行中の命令が完了してから RAM エラー例外処理が開始されます。このとき、CPU は次のように動作します。

(1) 発生したRAMエラーに対応する例外サービスルーチン開始アドレスを、例外処理ベクタテーブルから取り出します。
(2) ステータスレジスタ（SR）をスタックに退避します。
(3) プログラムカウンタ（PC）をスタックに退避します。退避するPCの値は、最後に実行した命令の次命令の先頭アドレスです。
(4) 例外処理ベクタテーブルから取り出したアドレスへジャンプして、プログラムの実行を開始します。このときのジャンプは遅延分岐ではありません。

3.5 レジスタバンクエラー

3.5.1 レジスタバンクエラー発生要因

(1) バンクオーバフロー
割り込みコントローラにおいて、レジスタバンクオーバフロー例外を受け付けるように設定されており、レジスタバンクを使用する割り込みが発生し、CPUに受け付けられたとき、レジスタバンクのすべての領域に退避がすでに行われていた場合
(2) バンクアンダフロー
レジスタバンクに退避が行われていないときに、RESBANK命令を実行しようとした場合

3.5.2 レジスタバンクエラー例外処理

レジスタバンクエラーが発生すると、レジスタバンクエラー例外処理が開始されます。このとき、CPU は次のように動作します。

(1) 発生したレジスタバンクエラーに対応する例外サービスルーチン開始アドレスを、例外処理ベクタテーブルから取り出します。
(2) ステータスレジスタ（SR）をスタックに退避します。
(3) プログラムカウンタ（PC）をスタックに退避します。退避するPCの値は、バンクオーバフロー時は最後に実行した命令の次命令の先頭アドレス、アンダフロー時は実行したRESBANK命令の先頭アドレスです。
バンクオーバフロー時は多重割り込みを防止するために、バンクオーバフローの要因となった割り込みのレベルをステータスレジスタ（SR）の割り込みマスクレベルビット（I3～I0）に書き込みます。
(4) 例外処理ベクタテーブルから取り出したアドレスへジャンプして、プログラムの実行を開始します。このときのジャンプは遅延分岐ではありません。

3.6 割り込み

3.6.1 割り込み要因

割り込み例外処理を起動させる要因には、**表 3.6** に示すように NMI、ユーザブレーク、H-UDI、外部割り込み、内蔵周辺モジュールがあります。

表 3.6 割り込み要因

種類	要求元	要因数
NMI	NMI 端子（外部からの入力）	1
ユーザブレーク	ユーザブレークコントローラ	1
H-UDI	ユーザデバッグインタフェース	1
外部割り込み（IRQ）、内蔵周辺モジュール	外部割り込み端子、内蔵周辺モジュール*	*

各割り込み要因には、それぞれ異なるベクタ番号とベクタテーブルオフセットが割り当てられています。ベクタ番号とベクタテーブルアドレスオフセットについては各製品のハードウェアマニュアルの**「割り込みコントローラ」**に記載されている**「割り込み例外ベクタと優先順位」**を参照してください。

【注】* **外部割り込み（IRQ）、内蔵周辺モジュールの要求元と要因数については、各製品のハードウェアマニュアルの「割り込みコントローラ」に記載されている「割り込み要因」を参照してください。**

3.6.2 割り込み優先順位

割り込み要因には優先順位が設けられており、複数の割り込みが同時に発生した場合（多重割り込み）、割り込みコントローラ（INTC）によって優先順位が判定され、その判定結果に従って例外処理が起動されます。

割り込み要因の優先順位は、優先レベル 0～16 の値で表され、優先レベル 0 が最低で、優先レベル 16 が最高です。NMI 割り込みは、優先レベル 16 のマスクできない最優先の割り込みで、常に受け付けられます。ユーザブレーク割り込み、および H-UDI の優先レベルは 15 です。IRQ 割り込みと内蔵周辺モジュール割り込みの優先レベルは、INTC の割り込み優先レベル設定レジスタで自由に設定することができます（**表 3.7**）。設定できる優先レベルは 0～15 で、優先レベル 16 は設定できません。割り込み優先レベル設定レジスタについては各製品のハードウェアマニュアルの**「割り込みコントローラ」**を参照してください。

表 3.7 割り込み優先順位

種類	優先レベル	備考
NMI	16	優先レベル固定、マスク不可能
ユーザブレーク	15	優先レベル固定
H-UDI	15	優先レベル固定
外部割り込み（IRQ）、内蔵周辺モジュール	0～15	割り込み優先レベル設定レジスタにより設定

3.6.3 割り込み例外処理

割り込みが発生すると、割り込みコントローラ（INTC）によって優先順位が判定されます。NMI は常に受け付けられますが、それ以外の割り込みは、その優先レベルがステータスレジスタ（SR）の割り込みマスクビット（I3～I0）に設定されている優先レベルより高い場合だけ受け付けられます。

割り込みが受け付けられると割り込み例外処理が開始されます。割り込み例外処理では、CPU は SR とプログラムカウンタ（PC）をスタックに退避します。NMI、UBC 以外の割り込み例外処理で、レジスタバンクを使用する設定が行われている場合、汎用レジスタ R0～R14、コントロールレジスタ GBR、システムレジスタ MACH、MACL、PR および実行される例外処理のベクタテーブルアドレスオフセットを、レジスタバンクに退避します。アドレスエラー、NMI 割り込み、UBC 割り込み、命令による例外処理の場合、レジスタバンクへの退避は行われません。また、レジスタバンクのすべてのバンクに退避が行われていた場合には、レジスタバンクの代わりにスタックへの自動退避が行われます。この場合、割り込みコントローラにおいて、レジスタバンクオーバフロー例外を受け付けないように設定されている必要があります。レジスタバンクオーバフロー例外を受け付けるように設定されている場合には、レジスタバンクオーバフロー例外が発生します。次に、受け付けた割り込みの優先レベル値を SR の I3～I0 ビットに書き込みます。ただし、NMI の場合優先レベルは 16 ですが、I3～I0 ビットに設定される値は H'F（レベル 15）です。次に、受け付けた割り込みに対応する例外処理ベクタテーブルから例外サービスルーチン開始アドレスを取り出し、そのアドレスにジャンプして実行を開始します。割り込み例外処理の詳細については各製品のハードウェアマニュアルの**「割り込みコントローラ」**に記載されている**「動作説明」**を参照してください。

3.7 命令による例外

3.7.1 命令による例外の種類

例外処理を起動する命令には、**表 3.8** に示すように、トラップ命令、スロット不当命令、一般不当命令、整数除算例外、FPU 例外があります。

表 3.8 命令による例外の種類

種類	要因となる命令	備考
トラップ命令	TRAPA	
スロット不当命令	遅延分岐命令直後（遅延スロット）に配置された未定義コード（含 FPU モジュールスタンバイ時もしくは FPU 非搭載品における FPU 命令および FPU に関する CPU 命令およびレジスタバンク非搭載品のレジスタバンク関連命令）、PC を書き換える命令、32 ビット命令、RESBANK 命令、DIVS 命令または DIVU 命令	遅延分岐命令 : JMP、JSR、BRA、BSR、RTS、RTE、BF/S、BT/S、BSRF、BRAF レジスタバンク関連命令: RESBANK、LDBANK、STBANK PC を書き換える命令 : JMP、JSR、BRA、BSR、RTS、RTE、BT、BF、TRAPA、BF/S、BT/S、BSRF、BRAF、JSR/N、RTV/N 32 ビット命令 : BAND.B、BANDNOT.B、BCLR.B、BLD.B、BLDNOT.B、BOR.B、BORNOT.B、BSET.B、BST.B、BXOR.B、FMOV.S@disp12、FMOV.D@disp12、MOV.B@disp12、MOV.W@disp12、MOV.L@disp12、MOVI20、MOVI20S、MOVU.B、MOVU.W

種類	要因となる命令	備考
一般不当命令	遅延スロット以外にある未定義コード（含 FPU モジュールスタンバイ時もしくは FPU 非搭載品における FPU 命令および FPU に関する CPU 命令およびレジスタバンク非搭載品のレジスタバンク関連命令）	
整数除算例外	ゼロ除算	DIVU、DIVS
	負の最大値÷(-1)	DIVS
FPU 例外	IEEE754 規格で定義された無効演算例外またはゼロによる除算例外を引き起こす命令、オーバフロー、アンダフローおよび不正確例外を引き起こす可能性のある命令	FADD、FSUB、FMUL、FDIV、FMAC、FCMP/EQ、FCMP/GT、FLOAT、FTRC、FCNVDS、FCNVSD、FSQRT

3.7.2 トラップ命令

TRAPA 命令を実行すると、トラップ命令例外処理が開始されます。このとき、CPU は次のように動作します。

(1) TRAPA命令で指定したベクタ番号に対応する例外サービスルーチン開始アドレスを、例外処理ベクタテーブルから取り出します。
(2) ステータスレジスタ（SR）をスタックに退避します。
(3) プログラムカウンタ（PC）をスタックに退避します。退避するPCの値は、TRAPA命令で次命令の先頭アドレスです。
(4) 例外処理ベクタテーブルから取り出したアドレスへジャンプして、プログラムの実行を開始します。このときのジャンプは遅延分岐ではありません。

3.7.3 スロット不当命令

遅延分岐命令の直後に配置された命令のことを「遅延スロットに配置された命令」とよびます。遅延スロットに配置された命令が未定義コードのとき、この未定義コードがデコードされるとスロット不当命令例外処理が開始されます。また、遅延スロットに配置された命令が PC を書き換える命令のときも、この PC を書き換える命令がデコードされるとスロット不当命令例外処理が開始されます。さらに、FPU 非搭載製品および FPU 搭載製品で FPU をモジュールスタンバイ状態にしたときは、浮動小数点命令および FPU に関する CPU 命令は未定義コードとして扱われ、遅延スロットに配置された場合、この命令がデコードされるとスロット不当命令例外処理が開始されます。

また、レジスタバンク非搭載品のレジスタバンク関連命令も未定義コードとして扱われ、遅延スロットに配置された場合、この命令がデコードされるとスロット不当命令例外処理が開始されます。

さらに、遅延スロットに配置された命令が 32 ビット命令、RESBANK 命令、DIVS 命令および DIVU 命令のときも、この命令がデコードされるとスロット不当命令例外処理が開始されます。

スロット不当命令例外処理のとき、CPU は次のように動作します。

(1) 例外サービスルーチン開始アドレスを例外処理ベクタテーブルから取り出します。
(2) ステータスレジスタ（SR）をスタックに退避します。
(3) プログラムカウンタ（PC）をスタックに退避します。退避するPCの値は、未定義コード、PCを書き換える命令、32ビット命令、RESBANK命令、DIVS命令、またはDIVU命令の直前にある遅延分岐命令の飛び先アドレスです。
(4) 例外処理ベクタテーブルから取り出したアドレスへジャンプして、プログラムの実行を開始します。このときのジャンプは遅延分岐ではありません。

3.7.4 一般不当命令

遅延分岐命令の直後（遅延スロット）以外に配置された未定義コードをデコードすると、一般不当命令例外処理が開始されます。また、FPU 非搭載製品および FPU 搭載製品で FPU をモジュールスタンバイ状態にしたときは、浮動小数点命令および FPU に関する CPU 命令は未定義コードとして扱われ、遅延分岐命令の直後（遅延スロット）以外に配置された場合、この命令がデコードされると一般不当命令例外処理が開始されます。

また、レジスタバンク非搭載品のレジスタバンク関連命令も未定義コードとして扱われ、遅延分岐命令の直後（遅延スロット）以外に配置された場合、この命令がデコードされると一般不当命令例外処理が開始されます。

一般不当命令例外処理時、CPU はスロット不当命令例外処理と同じ手順で動作します。ただし、退避する PC の値は、スロット不当命令例外処理と異なり、この未定義コードの先頭アドレスになります。

3.7.5 整数除算例外

整数除算命令がゼロによる除算を実行した場合、または整数除算の結果がオーバフローしたとき、整数除算例外が発生します。ゼロによる除算例外の要因となる命令は、DIVU と DIVS です。オーバフロー例外の要因となる命令は DIVS のみで、負の最大値を-1 で除算する場合にのみ発生します。整数除算例外が発生すると CPU は次のように動作します。

(1) 発生した整数除算例外に対応する例外サービスルーチン開始アドレスを、例外処理ベクタテーブルから取り出します。
(2) ステータスレジスタ（SR）をスタックに退避します。
(3) プログラムカウンタ（PC）をスタックに退避します。退避するPCの値は、例外を発生した整数除算命令の先頭アドレスです。
(4) 例外処理ベクタテーブルから取り出したアドレスへジャンプして、プログラムの実行を開始します。このときのジャンプは遅延分岐ではありません。

3.7.6 FPU 例外

浮動小数点ステータスレジスタ（FPSCR）の FPU 例外イネーブルフィールド（Enable）中の V、Z、O、U または I ビットがセットされているとき、FPU 例外処理が発生します。これは浮動小数点演算命令が IEEE754 規格で定義された無効演算例外、ゼロによる除算例外、オーバフロー（可能性のある命令）、アンダフロー（可能性のある命令）および不正確例外（可能性のある命令）を引き起こしたことを示します。

FPU 例外処理の発生要因となる浮動小数点演算命令には以下があります。

FADD、FSUB、FMUL、FDIV、FMAC、FCMP/EQ、FCMP/GT、FLOAT、FTRC、FCNVDS、FCNVSD、FSQRT

該当する FPU 例外イネーブルビット（Enable）がセットされているときのみ、FPU 例外処理が発生します。FPU が浮動小数点演算による例外要因を検出すると、FPU の動作は中断されて CPU に FPU 例外処理の発生を通知します。CPU は例外処理を開始すると次のように動作します。

(1) 発生したFPU例外処理に対応する例外サービスルーチン開始アドレスを例外処理ベクタテーブルから取り出します。
(2) ステータスレジスタ（SR）をスタックに退避します。
(3) プログラムカウンタ（PC）をスタックに退避します。退避するPCの値は最後に実行した命令の次の命令の先頭アドレスです。
(4) 例外処理ベクタテーブルから取り出したアドレスへジャンプして、プログラムの実行を開始します。このときのジャンプは遅延分岐ではありません。

FPSCRの例外フラグフィールド（Flag）は、FPU例外処理が受け付けられたか否かにかかわらず常

に更新され、ユーザが明示的に命令でクリアするまでセットされたままです。FPSCRの要因フィールド（Cause）は浮動小数点演算命令が実行されるごとに変化します。

また、FPSCRレジスタのFPU例外イネーブルフィールド（Enable）中のVビットがセットされ、かつFPSCRのQISビットがセットされているとき、qNaNもしくは±∞を浮動小数点演算命令のソースに入力するとFPU例外処理が発生します。

3.8 例外処理が受け付けられない場合

アドレスエラー、RAM エラー、FPU 例外、レジスタバンクエラー（オーバフロー）および割り込みは、**表 3.9** に示すように、遅延分岐命令の直後に発生すると、すぐに受け付けられず保留される場合があります。この場合、例外を受け付けられる命令がデコードされたときに受け付けられます。

表 3.9 遅延分岐命令の直後の例外要因発生

発生した時点	例外要因				
	アドレスエラー	RAM エラー	FPU 例外	レジスタバンクエラー（オーバフロー）	割り込み
遅延分岐命令*の直後	×	×	×	×	×

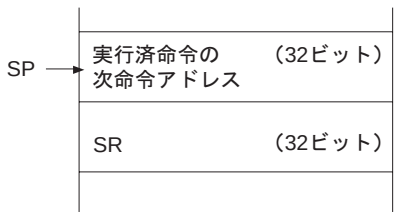
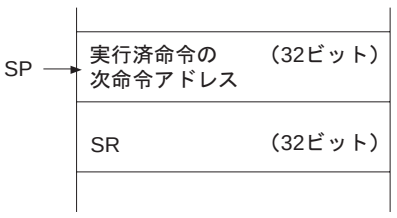
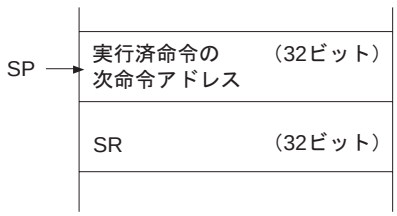
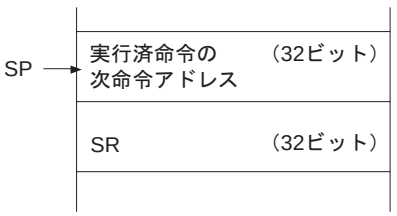
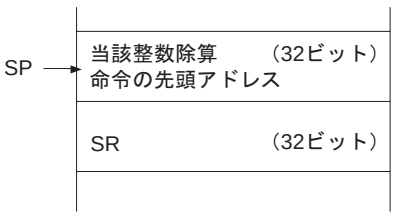
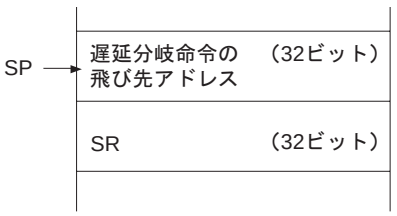
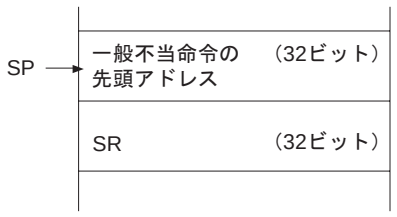
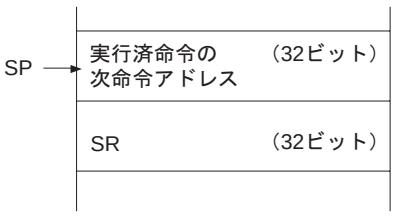
【注】 ×：受け付けられない

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

3.9　例外処理後のスタックの状態

例外処理終了後のスタックの状態は、**表 3.10** に示すようになります。

表 3.10　例外処理終了後のスタックの状態

種類	スタックの状態	種類	スタックの状態
アドレスエラー		割り込み	
RAMエラー		レジスタバンクエラー（オーバフロー）	
レジスタバンクエラー（アンダフロー）		整数除算命令（0除算、オーバフロー）	
トラップ命令		スロット不当命令	
一般不当命令		FPU 例外	

3.10 使用上の注意

3.10.1 スタックポインタ（SP）の値

SP の値は必ず 4 の倍数になるようにしてください。SP が 4 の倍数以外のとき、例外処理でスタックがアクセスされるとアドレスエラーが発生します。

3.10.2 ベクタベースレジスタ（VBR）の値

VBR の値は必ず 4 の倍数になるようにしてください。VBR が 4 の倍数以外のとき、例外処理でベクタがアクセスされるとアドレスエラーが発生します。

3.10.3 アドレスエラー例外処理のスタッキングで発生するアドレスエラー

SP が 4 の倍数になっていないと、例外処理（割り込みなど）のスタッキングでアドレスエラーが発生し、その例外処理終了後、アドレスエラー例外処理に移ります。アドレスエラー例外処理でのスタッキングでもアドレスエラーが発生しますが、無限にアドレスエラー例外処理によるスタッキングが続かないように、そのときのアドレスエラーは受け付けないようになっています。これにより、プログラムの制御をアドレスエラー例外サービスルーチンに移すことができ、エラー処理を行うことができます。

なお、例外処理のスタッキングでアドレスエラーが発生した場合、スタッキングのバスサイクル（ライト）は実行されます。SR と PC のスタッキングでは、SP がそれぞれ−4 されるので、スタッキング終了後も SP の値は 4 の倍数になっていません。また、スタッキング時に出力されるアドレスの値は SP の値で、エラーの発生したアドレスそのものが出力されます。このとき、スタッキングされたライトデータは不定です。

3.10.4 割り込みマスクビット変更による割り込み制御

LDC、LDC.L 命令でステータスレジスタ（SR）の割り込みマスクビット（I3～I0）の値を操作して、割り込みを禁止から許可に変更する場合、割り込みを許可する命令に続く 5 命令を実行する間は割り込みを受け付けない場合があります。

したがって、LDC、LDC.L 命令でステータスレジスタ（SR）の割り込みマスクビット（I3～I0）の値を操作して、割り込みの許可／禁止を制御する場合は、割り込みを許可する命令と割り込みを禁止する命令の間に 5 命令以上配置してください。

4. 命令の特長

4.1 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
基本命令は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) 32ビット固定長命令の追加
SH-2A/SH2A-FPUでは、32ビット固定長の命令が追加されています。これにより、性能および使い勝手が向上します。

(3) 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。

(4) 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 4.1 ワードデータの符号拡張

SH-2A/SH2A-FPU 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。

(5) 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。

(6) 遅延分岐
無条件分岐命令などは、一部の命令を除き遅延分岐命令です。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 4.2 遅延分岐命令

SH-2A/SH2A-FPU 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

(7) 遅延スロットなし無条件分岐命令の追加
SH-2A/SH2A-FPUでは、遅延スロット命令を実行しない無条件分岐命令を追加しました。これにより、不要なNOP命令の削減が可能となり、コードサイズを削減できます。

(8) 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。

(9) 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 4.3 T ビット

SH-2A/SH2A-FPU 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

(10) 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。
またSH-2A/SH2A-FPUでは、17～28ビットのイミディエイトデータを命令コードの中に配置することも可能です。ただし、21～28ビットのイミディエイトデータについては、レジスタ転送後、OR命令を実行する必要があります。

表 4.4 イミディエイトデータによる参照

区分	SH-2A/SH2A-FPU 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	MOVI20 #H'1234, R0	MOV.W #H'1234,R0
20 ビットイミディエイト	MOVI20 #H'12345, R0	MOV.L #H'12345,R0
28 ビットイミディエイト	MOVI20S #H'12345, R0 OR #H'67, R0	MOV.L #H'1234567,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。

（11） 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。

また、SH-2A/SH2A-FPUでは、28ビット以下の絶対アドレスでデータを参照するとき、命令コード中に配置したイミディエイトデータ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することも可能です。ただし、21～28ビットの絶対アドレスでデータを参照するときは、レジスタ転送後、OR命令を使用する必要があります。

表 4.5　絶対アドレスによる参照

区分	SH-2A/SH2A-FPU CPU	他の CPU の例
20 ビット以下	MOVI20　#H'12345, R1 MOV.B　@R1, R0	MOV.B　@H'12345,R0
21～28 ビット	MOVI20S　#H'12345, R1 OR　#H'67, R1 MOV.B　@R1, R0	MOV.B　@H'1234567,R0
29 ビット以上	MOV.L　@(disp,PC),R1 MOV.B　@R1,R0 ・・・・・・・・ .DATA.L　H'12345678	MOV.B　@H'12345678,R0

（12） 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。

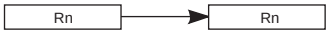
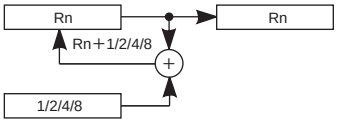
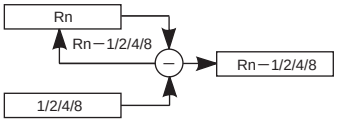
表 4.6　ディスプレースメントによる参照

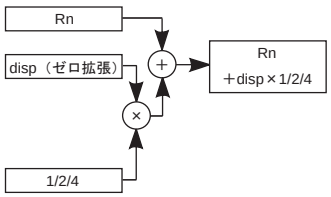
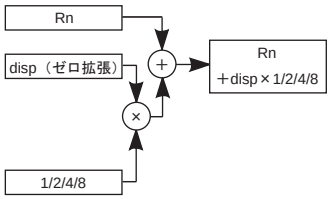
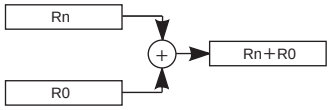
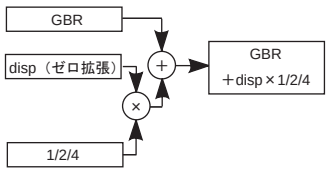
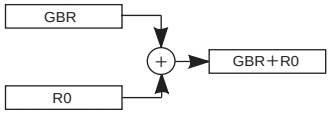
区分	SH-2A/SH2A-FPU CPU	他の CPU の例
16 ビットディスプレースメント	MOV.W　@(disp,PC),R0 MOV.W　@(R0,R1),R2 ・・・・・・・・ .DATA.W　H'1234	MOV.W　@(H'1234,R1),R2

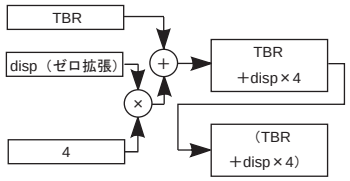
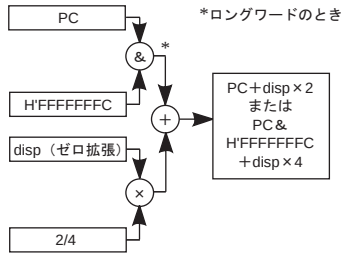
4.2 アドレッシングモード

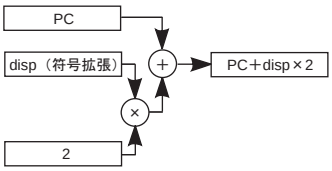
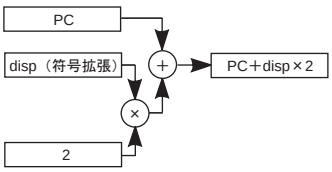
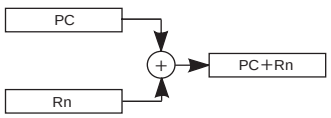
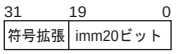
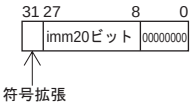
アドレッシングモードと実効アドレスの計算方法は次のとおりです。

表 4.7 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、クワッドワードのとき 8 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 クワッドワード： Rn＋8→Rn
プリデクリメントレジスタ間接	@-Rn	実効アドレスは、あらかじめ定数を減算したレジスタ Rn の内容です。定数はバイトのとき 1、ワードのとき 2、ロングワードのとき 4、クワッドワードのとき 8 です。 	バイト：Rn－1→Rn ワード：Rn－2→Rn ロングワード：Rn－4→Rn クワッドワード： Rn－8→Rn （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
	@(disp:12,Rn)	実効アドレスはレジスタ Rn に 12 ビットディスプレースメント disp を加算した内容です。disp はゼロ拡張します。 	バイト：Rn＋disp ワード：Rn＋disp×2 ロングワード：Rn＋disp×4 クワッドワード： Rn＋disp×8
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 TBR 二重間接	@@ (disp:8,TBR)	実効アドレスはレジスタ TBR に 8 ビットディスプレースメント disp を加算したアドレスの内容です。disp はゼロ拡張後 4 倍します。 	（TBR＋disp×4）アドレスの内容
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

アドレッシングモード	命令フォーマット	実効アドレスの計算方法	計算式
PC 相対	disp:8	実効アドレスはレジスタ 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:20	MOVI20 命令の 20 ビットイミディエイト imm は符号拡張します。 	—
		MOVI20S 命令の 20 ビットイミディエイト imm は 8 ビット左にシフトし、上位側は符号拡張、下位側はゼロ詰めを行います。 	—
	#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。	—
	#imm:3	BAND、BOR、BXOR、BST、BLD、BSET、BCLR 命令の 3 ビットイミディエイト imm はビット位置を表します。	—

4.3 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

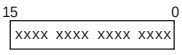
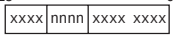
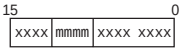
mmmm ：ソースレジスタ

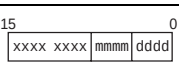
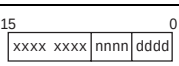
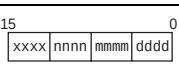
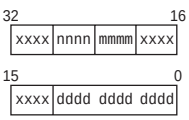
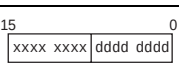
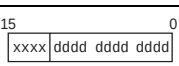
nnnn ：デスティネーションレジスタ

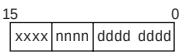
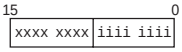
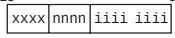
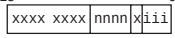
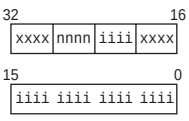
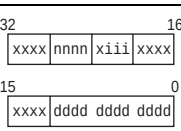
iiii ：イミディエイトデータ

dddd ：ディスプレースメント

表 4.8 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
		R0（レジスタ直接）	nnnn：レジスタ直接	DIVU R0, 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
		mmmm：レジスタ直接	R15(プリデクリメントレジスタ間接)	MOVMU.L Rm, @-R15
		R15(ポストインクリメントレジスタ間接)	nnnn：レジスタ直接	MOVMU.L @R15+, Rn
		R0（レジスタ直接）	nnnn：ポストインクリメントレジスタ間接	MOV.L R0,@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：プリデクリメントレジスタ間接	R0（レジスタ直接）	MOV.L @-Rm, R0
		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+

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
nm 形式		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)
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
nmd12 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, @(disp12, 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 @(disp12, 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：ディスプレースメント付き TBR 二重間接	—	JSR/N @@(disp8,TBR)
		dddddddd：PC 相対	—	BF label
d12 形式		ddddddddddddd：PC 相対	—	BRA label (label=disp+PC)

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
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
ni3 形式		nnnn: レジスタ直接 iii:イミディエイト	―	BLD #imm3,Rn
		―	nnnn: レジスタ直接 iii:イミディエイト	BST #imm3,Rn
ni20 形式		iiiiiiiiiiiiiiiii：イミディエイト	nnnn: レジスタ直接	MOVI20 #imm20, Rn
nid 形式		nnnndddddddddddd: ディスプレイスメント付きレジスタ間接 iii:イミディエイト	―	BLD.B #imm3,@(disp12,Rn)
		―	nnnndddddddddddd: ディスプレイスメント付きレジスタ間接 iii:イミディエイト	BST.B #imm3,@(disp12,Rn)

【注】 * 積和命令では nnnn は、ソースレジスタです。

5. 命令セット

5.1 分類順命令セット

命令を分類順に表 5.1 に示します。

表 5.1 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	13	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 逆スタック転送	62
		MOVA	実行アドレスの転送	
		MOVI20	20 ビットイミディエイトデータの転送	
		MOVI20S	20 ビットイミディエイトデータの転送 左 8 ビットシフト	
		MOVML	R0～Rn のレジスタ退避･回復	
		MOVMU	Rn～R14,PR のレジスタ退避･回復	
		MOVRT	Ｔビット反転 Rn への転送	
		MOVT	Ｔビットの転送	
		MOVU	無符号データの転送	
		NOTT	Ｔビット反転	
		PREF	オペランドキャッシュへのプリフェッチ	
		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	26	ADD	2 進加算	40
		ADDC	キャリ付き 2 進加算	
		ADDV	オーバフロー付き 2 進加算	
		CMP/cond	比較	
		CLIPS	符号付き飽和値比較	
		CLIPU	符号なし飽和値比較	
		DIVS	符号付き除算(32÷32)	
		DIVU	符号なし除算(32÷32)	
		DIV1	1 ステップ除算	
		DIV0S	符号付き 1 ステップ除算の初期化	

分　類	命令の種類	オペコード	機　能	命令数
算術演算命令	26	DIV0U	符号なし 1 ステップ除算の初期化	40
		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	
		MULR	Rn 結果格納符号付き乗算	
		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 進減算	
		SUBV	アンダフロー付き 2 進減算	
論理演算命令	6	AND	論理積演算	14
		NOT	ビット反転	
		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	12	ROTL	1 ビット左回転	16
		ROTR	1 ビット右回転	
		ROTCL	Ｔビット付き 1 ビット左回転	
		ROTCR	Ｔビット付き 1 ビット右回転	
		SHAD	ダイナミック算術的シフト	
		SHAL	算術的 1 ビット左シフト	
		SHAR	算術的 1 ビット右シフト	
		SHLD	ダイナミック論理的シフト	
		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	10	BF	条件分岐、遅延付き条件分岐 （Ｔ=0 で分岐）	15
		BT	条件分岐、遅延付き条件分岐 （Ｔ=1 で分岐）	
		BRA	遅延付き無条件分岐	
		BRAF	遅延付き無条件分岐	
		BSR	遅延付きサブルーチンプロシージャへの分岐	
		BSRF	遅延付きサブルーチンプロシージャへの分岐	
		JMP	遅延付き無条件分岐	

分　類	命令の種類	オペコード	機　能	命令数
分岐命令	10	JSR	サブルーチンプロシージャへの分岐、 遅延付きサブルーチンプロシージャへの分岐	15
		RTS	サブルーチンプロシージャからの復帰 遅延付きサブルーチンプロシージャからの復帰	
		RTV/N	Rm→R0 転送付き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	14	CLRT	Ｔビットのクリア	36
		CLRMAC	MAC レジスタのクリア	
		LDBANK	指定レジスタバンクエントリからのレジスタ復帰	
		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	
		RESBANK	レジスタバンクからのレジスタ復帰	
		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	
		STBANK	指定レジスタバンクエントリへのレジスタ退避	
		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	
浮動小数点演算命令	19	FABS	浮動小数点数絶対値	48
		FADD	浮動小数点数加算	
		FCMP	浮動小数点数比較	
		FCNVDS	倍精度から単精度への変換	
		FCNVSD	単精度から倍精度への変換	
		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	
		FSCHG	SZ ビット反転	
		FSQRT	浮動小数点平方根	
		FSTS	システムレジスタ FPUL からの浮動小数点数ストア	
		FSUB	浮動小数点数減算	
		FTRC	浮動小数点数の整数への切り捨て変換	
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	

分　類	命令の種類	オペコード	機　能	命令数
ビット操作命令	10	BAND	ビット論理積	14
		BCLR	ビットクリア	
		BLD	ビットロード	
		BOR	ビット論理和	
		BSET	ビットセット	
		BST	ビットストア	
		BXOR	ビット排他的論理和	
		BANDNOT	ビットノット論理積	
		BORNOT	ビットノット論理和	
		BLDNOT	ビットノットロード	
	計　112			253

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。 記号の説明 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。 記号の説明 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。 記号の説明 →、← : 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。 記号の説明 — :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 6 章　各命令の説明」を参照してください。

5.1.1 データ転送命令

表 5.2 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	互換性		
					SH2E	SH4	新規 SH-2A/ SH2A-FPU
MOV #imm, Rn	1110nnnniiiiiiii	imm→符号拡張→Rn	1	—	○	○	
MOV.W @(disp, PC), Rn	1001nnnnddddddd	(disp×2+PC)→符号拡張→Rn	1	—	○	○	
MOV.L @(disp, PC), Rn	1101nnnnddddddd	(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), Rn	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	—	○	○	

命　令	命令コード	動　作	実行ステート	Tビット	互換性		
					SH2E	SH4	新規 SH-2A/ SH2A-FPU
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	—	○	○	
MOV.B R0, @Rn+	0100nnnn10001011	R0→(Rn), Rn+1→Rn	1	—			○
MOV.W R0, @Rn+	0100nnnn10011011	R0→(Rn), Rn+2→Rn	1	—			○
MOV.L R0, @Rn+	0100nnnn10101011	R0→(Rn), Rn+4→Rn	1	—			○
MOV.B @-Rm, R0	0100mmmm11001011	Rm-1→Rm, (Rm)→符号拡張→R0	1	—			○
MOV.W @-Rm, R0	0100mmmm11011011	Rm-2→Rm, (Rm)→符号拡張→R0	1	—			○
MOV.L @-Rm, R0	0100mmmm11101011	Rm-4→Rm, (Rm)→R0	1	—			○
MOV.B Rm, @(disp12, Rn)	0011nnnnmmmm0001 0000dddddddddddd	Rm→(disp+Rn)	1	—			○
MOV.W Rm, @(disp12, Rn)	0011nnnnmmmm0001 0001dddddddddddd	Rm→(disp×2+Rn)	1	—			○
MOV.L Rm, @(disp12, Rn)	0011nnnnmmmm0001 0010dddddddddddd	Rm→(disp×4+Rn)	1	—			○
MOV.B @(disp12, Rm), Rn	0011nnnnmmmm0001 0100dddddddddddd	(disp+Rm)→符号拡張→Rn	1	—			○
MOV.W @(disp12, Rm), Rn	0011nnnnmmmm0001 0101dddddddddddd	(disp×2+Rm)→符号拡張→Rn	1	—			○
MOV.L @(disp12, Rm), Rn	0011nnnnmmmm0001 0110dddddddddddd	(disp×4+Rm)→Rn	1	—			○
MOVA @(disp, PC), R0	11000111dddddddd	disp×4+PC→R0	1	—	○	○	
MOVI20 #imm20, Rn	0000nnnniiii0000 iiiiiiiiiiiiiiii	imm→符号拡張→Rn	1	—			○
MOVI20S #imm20, Rn	0000nnnniiii0001 iiiiiiiiiiiiiiii	imm<<8→符号拡張→Rn	1	—			○
MOVML.L Rm, @-R15	0100mmmm11110001	R15-4→R15, Rm→(R15) R15-4→R15, Rm-1→(R15) : R15-4→R15, R0→(R15) ※Rm=R15 のとき、Rm を PR に読み替え	1～16	—			○
MOVML.L @R15+, Rn	0100nnnn11110101	(R15)→R0, R15+4→R15 (R15)→R1, R15+4→R15 : (R15)→Rn ※Rn=R15 のとき、Rn を PR に読み替え	1～16	—			○

命　令	命令コード	動　作	実行ステート	Tビット	互換性		
					SH2E	SH4	新規 SH-2A/ SH2A-FPU
MOVMU.L Rm, @-R15	0100mmmm11110000	R15-4→R15, PR→(R15) R15-4→R15, R14→(R15) : R15-4→R15, Rm→(R15) ※Rm=R15 のとき、Rm を PR に読み替え	1～16	—			○
MOVMU.L @R15+, Rn	0100nnnn11110100	(R15)→Rn, R15+4→R15 (R15)→Rn+1, R15+4→R15 : (R15)→R14, R15+4→R15 (R15)→PR ※Rn=R15 のとき、Rn を PR に読み替え	1～16	—			○
MOVRT Rn	0000nnnn00111001	~T→Rn	1	—			○
MOVT Rn	0000nnnn00101001	T→Rn	1	—	○	○	
MOVU.B @(disp12,Rm), Rn	0011nnnnmmmm0001 1000dddddddddddd	(disp+Rm)→ゼロ拡張→Rn	1	—			○
MOVU.W @(disp12,Rm),Rn	0011nnnnmmmm0001 1001dddddddddddd	(disp×2+Rm)→ゼロ拡張→Rn	1	—			○
NOTT	0000000001101000	~T→T	1	演算結果			○
PREF @Rn	0000nnnn10000011	(Rn)→オペランドキャッシュ	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	—	○	○	

5.1.2 算術演算命令

表 5.3 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	互換性		
					SH2E	SH4	新規 SH-2A/ SH2A-FPU
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 それ以外のとき 0→T	1	比較結果	○	○	
CMP/EQ Rm, Rn	0011nnnnmmmm0000	Rn=Rm のとき 1→T それ以外のとき 0→T	1	比較結果	○	○	
CMP/HS Rm, Rn	0011nnnnmmmm0010	無符号で Rn≧Rm のとき 1→T それ以外のとき 0→T	1	比較結果	○	○	
CMP/GE Rm, Rn	0011nnnnmmmm0011	有符号で Rn≧Rm のとき 1→T それ以外のとき 0→T	1	比較結果	○	○	
CMP/HI Rm, Rn	0011nnnnmmmm0110	無符号で Rn>Rm のとき 1→T それ以外のとき 0→T	1	比較結果	○	○	
CMP/GT Rm, Rn	0011nnnnmmmm0111	有符号で Rn>Rm のとき 1→T それ以外のとき 0→T	1	比較結果	○	○	
CMP/PL Rn	0100nnnn00010101	Rn>0 のとき 1→T それ以外のとき 0→T	1	比較結果	○	○	
CMP/PZ Rn	0100nnnn00010001	Rn≧0 のとき 1→T それ以外のとき 0→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 それ以外のとき 0→T	1	比較結果	○	○	
CLIPS.B Rn	0100nnnn10010001	Rn>(H'0000007F)のとき、 (H'0000007F) →Rn, 1→CS Rn<(H'FFFFFF80)のとき、 (H'FFFFFF80) →Rn, 1→CS	1	—			○
CLIPS.W Rn	0100nnnn10010101	Rn>(H'00007FFF)のとき、 (H'00007FFF) →Rn, 1→CS Rn<(H'FFFF8000)のとき、 (H'FFFF8000) →Rn, 1→CS	1	—			○
CLIPU.B Rn	0100nnnn10000001	Rn>(H'000000FF)のとき、 (H'000000FF) →Rn, 1→CS	1	—			○
CLIPU.W Rn	0100nnnn10000101	Rn>(H'0000FFFF)のとき、 (H'0000FFFF) →Rn, 1→CS	1	—			○
DIV1 Rm, Rn	0011nnnnmmmm0100	1 ステップ除算(Rn÷Rm)	1	計算結果	○	○	

命　令	命令コード	動　作	実行ステート	T ビット	互換性		
					SH 2E	SH4	新規 SH-2A/ SH2A -FPU
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	○	○	
DIVS R0, Rn	0100nnnn10010100	符号付きで Rn÷R0→Rn　32÷32→32 ビット	36	—			○
DIVU R0, Rn	0100nnnn10000100	符号なしで Rn÷R0→Rn　32÷32→32 ビット	34	—			○
DMULS.L Rm, Rn	0011nnnnmmmm1101	符号付きで Rn×Rm→MACH, MACL 32×32→64 ビット	2	—	○	○	
DMULU.L Rm, Rn	0011nnnnmmmm0101	符号なしで Rn×Rm→MACH, MACL 32×32→64 ビット	2	—	○	○	
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	4	—	○	○	
MAC.W @Rm+, @Rn+	0100nnnnmmmm1111	符号付きで(Rn)×(Rm)+MAC→MAC 16×16+64→64 ビット	3	—	○	○	
MUL.L Rm, Rn	0000nnnnmmmm0111	Rn×Rm→MACL　32×32→32 ビット	2	—	○	○	
MULR R0, Rn	0100nnnn10000000	R0×Rn→Rn　32×32→32 ビット	2	—			○
MULS.W Rm, Rn	0010nnnnmmmm1111	符号付きで Rn×Rm→MACL 16×16→32 ビット	1	—	○	○	
MULU.W Rm, Rn	0010nnnnmmmm1110	符号なしで Rn×Rm→MACL 16×16→32 ビット	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	アンダフロー	○	○	

5.1.3 論理演算命令

表 5.4 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	互換性		
						SH2E	SH4	新規 SH-2A/ SH2A-FPU
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, それ以外のとき 0→T, 1→MSB of(Rn)	3	テスト結果	○	○	
TST	Rm, Rn	0010nnnnmmmm1000	Rn & Rm, 結果が 0 のとき 1→T, その他 0→T	1	テスト結果	○	○	
TST	#imm, R0	11001000iiiiiiii	R0 & imm, 結果が 0 のとき 1→T その他 0→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 その他 0→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	—	○	○	

5.1.4 シフト命令

表 5.5 シフト命令

命令	命令コード	動作	実行ステート	Tビット	互換性		
					SH2E	SH4	新規 SH-2A/ SH2A-FPU
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	○	○	
SHAD Rm, Rn	0100nnnnmmmm1100	Rm≧0 のとき Rn<<Rm→Rn Rm<0 のとき Rn>>|Rm|→[MSB→Rn]	1	—		○	
SHAL Rn	0100nnnn00100000	T←Rn←0	1	MSB	○	○	
SHAR Rn	0100nnnn00100001	MSB→Rn→T	1	LSB	○	○	
SHLD Rm, Rn	0100nnnnmmmm1101	Rm≧0 のとき Rn<<Rm→Rn Rm<0 のとき Rn>>|Rm|→[0→Rn]	1	—		○	
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	—	○	○	

5.1.5 分岐命令

表 5.6 分岐命令

命令		命令コード	動作	実行ステート	Tビット	互換性		
						SH2E	SH4	新規 SH-2A/ SH2A-FPU
BF	label	10001011dddddddd	T=0 のとき disp×2+PC→PC, T=1 のとき nop	3/1*	—	○	○	
BF/S	label	10001111dddddddd	遅延分岐、T=0 のとき disp×2+PC→PC, T=1 のとき nop	2/1*	—	○	○	
BT	label	10001001dddddddd	T=1 のとき disp×2+PC→PC, T=0 のとき nop	3/1*	—	○	○	
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*	—	○	○	
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	—	○	○	
JSR/N	@Rm	0100mmmm01001011	PC-2→PR, Rm→PC	3	—			○
JSR/N	@@(disp8, TBR)	10000011dddddddd	PC-2→PR, (disp×4+TBR)→PC	5	—			○
RTS		0000000000001011	遅延分岐、PR→PC	2	—	○	○	
RTS/N		0000000001101011	PR→PC	3	—			○
RTV/N	Rm	0000mmmm01111011	Rm→R0, PR→PC	3	—			○

【注】* 分岐しないときは 1 ステートになります。

5.1.6 システム制御命令

表 5.7 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	互換性		
					SH2E	SH4	新規 SH-2A/ SH2A-FPU
CLRT	0000000000001000	0→T	1	0	○	○	
CLRMAC	0000000000101000	0→MACH,MACL	1	—	○	○	
LDBANK @Rm, R0	0100mmmm11100101	（指定レジスタバンクエントリ）→R0	6	—			○
LDC Rm, SR	0100mmmm00001110	Rm→SR	3	LSB	○	○	
LDC Rm, TBR	0100mmmm01001010	Rm→TBR	1	—			○
LDC Rm, GBR	0100mmmm00011110	Rm→GBR	1	—	○	○	
LDC Rm, VBR	0100mmmm00101110	Rm→VBR	1	—	○	○	
LDC.L @Rm+, SR	0100mmmm00000111	(Rm)→SR, Rm+4→Rm	5	LSB	○	○	
LDC.L @Rm+, GBR	0100mmmm00010111	(Rm)→GBR, Rm+4→Rm	1	—	○	○	
LDC.L @Rm+, VBR	0100mmmm00100111	(Rm)→VBR, Rm+4→Rm	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+, 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	—	○	○	
RESBANK	0000000001011011	バンク→R0～R14, GBR, MACH, MACL, PR	9*	—			○
RTE	0000000000101011	遅延分岐、スタック領域→PC/SR	6	—	○	○	
SETT	0000000000011000	1→T	1	1	○	○	
SLEEP	0000000000011011	スリープ	5	—	○	○	
STBANK R0, @Rn	0100nnnn11100001	R0→（指定レジスタバンクエントリ）	7	—			○
STC SR, Rn	0000nnnn00000010	SR→Rn	2	—	○	○	
STC TBR, Rn	0000nnnn01001010	TBR→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)	1	—	○	○	
STC.L VBR, @- Rn	0100nnnn00100011	Rn-4→Rn, VBR→(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 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	—	○	○	

命　令	命令コード	動　作	実行ステート	Tビット	互換性		
					SH2E	SH4	新規SH-2A/SH2A-FPU
TRAPA　#imm	11000011iiiiiiii	PC/SR→スタック領域、(imm×4+VBR)→PC	5	—	○	○	

【注】 命令の実行ステートについて

表に示した実行ステートは最少値です。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。

＊　バンクのオーバフロー時は、ステート数が 19 です。

5.1.7 浮動小数点命令

表 5.8 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	互換性		
					SH2E	SH4	新規SH-2A/SH2A-FPU
FABS FRn	1111nnnn01011101	|FRn|→FRn	1	—	○	○	
FABS DRn	1111nnn001011101	|DRn|→DRn	1	—		○	
FADD FRm, FRn	1111nnnnmmmm0000	FRn+FRm→FRn	1	—	○	○	
FADD DRm, DRn	1111nnn0mmm00000	DRn+DRm→DRn	6	—		○	
FCMP/EQ FRm, FRn	1111nnnnmmmm0100	(FRn=FRm)? 1:0→T	1	比較結果	○	○	
FCMP/EQ DRm, DRn	1111nnn0mmm00100	(DRn=DRm)? 1:0→T	2	比較結果		○	
FCMP/GT FRm, FRn	1111nnnnmmmm0101	(FRn>FRm)? 1:0→T	1	比較結果	○	○	
FCMP/GT DRm, DRn	1111nnn0mmm00101	(DRn>DRm)? 1:0→T	2	比較結果		○	
FCNVDS DRm, FPUL	1111mmm010111101	(float)DRm→FPUL	2	—		○	
FCNVSD FPUL, DRn	1111nnn010101101	(double)FPUL→DRn	2	—		○	
FDIV FRm, FRn	1111nnnnmmmm0011	FRn/FRm→FRn	10	—	○	○	
FDIV DRm, DRn	1111nnn0mmm00011	DRn/DRm→DRn	23	—		○	
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	—	○	○	
FLOAT FPUL,DRn	1111nnn000101101	(double)FPUL→DRn	2	—		○	
FMAC FR0,FRm,FRn	1111nnnnmmmm1110	FR0×FRm+FRn→FRn	1	—	○	○	
FMOV FRm, FRn	1111nnnnmmmm1100	FRm→FRn	1	—	○	○	
FMOV DRm, DRn	1111nnn0mmm01100	DRm→DRn	2	—		○	
FMOV.S @(R0, Rm), FRn	1111nnnnmmmm0110	(R0+Rm)→FRn	1	—	○	○	
FMOV.D @(R0, Rm), DRn	1111nnn0mmmm0110	(R0+Rm)→DRn	2	—		○	
FMOV.S @Rm+, FRn	1111nnnnmmmm1001	(Rm)→FRn, Rm+=4	1	—	○	○	
FMOV.D @Rm+, DRn	1111nnn0mmmm1001	(Rm)→DRn, Rm+=8	2	—		○	
FMOV.S @Rm, FRn	1111nnnnmmmm1000	(Rm)→FRn	1	—	○	○	
FMOV.D @Rm, DRn	1111nnn0mmmm1000	(Rm)→DRn	2	—		○	
FMOV.S @(disp12,Rm),FRn	0011nnnnmmmm0001 0111dddddddddddd	(disp×4+Rm)→FRn	1	—			○
FMOV.D @(disp12,Rm),DRn	0011nnn0mmmm0001 0111dddddddddddd	(disp×8+Rm)→DRn	2	—			○
FMOV.S FRm, @(R0,Rn)	1111nnnnmmmm0111	FRm→(R0+Rn)	1	—	○	○	
FMOV.D DRm, @(R0,Rn)	1111nnnnmmm00111	DRm→(R0+Rn)	2	—		○	

命　令	命令コード	動　作	実行ステート	Tビット	互換性		
					SH2E	SH4	新規SH-2A/SH2A-FPU
FMOV.S FRm, @-Rn	1111nnnnmmmm1011	Rn-=4, FRm→(Rn)	1	—	○	○	
FMOV.D DRm, @-Rn	1111nnnnmmm01011	Rn-=8, DRm→(Rn)	2	—		○	
FMOV.S FRm, @Rn	1111nnnnmmmm1010	FRm→(Rn)	1	—	○	○	
FMOV.D DRm, @Rn	1111nnnnmmm01010	DRm→(Rn)	2	—		○	
FMOV.S FRm, @(disp12,Rn)	0011nnnnmmmm0001 0011ddddddddddd	FRm→(disp×4+Rn)	1	—			○
FMOV.D DRm, @(disp12,Rn)	0011nnnnmmm00001 0011ddddddddddd	DRm→(disp×8+Rn)	2	—			○
FMUL FRm, FRn	1111nnnnmmmm0010	FRn×FRm→FRn	1	—	○	○	
FMUL DRm, DRn	1111nnn0mmm00010	DRn×DRm→DRn	6	—		○	
FNEG FRn	1111nnnn01001101	-FRn→FRn	1	—	○	○	
FNEG DRn	1111nnn001001101	-DRn→DRn	1	—		○	
FSCHG	1111001111111101	FPSCR.SZ=~FPSCR.SZ	1	—		○	
FSQRT FRn	1111nnnn01101101	√FRn→FRn	9	—		○	
FSQRT DRn	1111nnn001101101	√DRn→DRn	22	—		○	
FSTS FPUL,FRn	1111nnnn00001101	FPUL→FRn	1	—	○	○	
FSUB FRm, FRn	1111nnnnmmmm0001	FRn-FRm→FRn	1	—	○	○	
FSUB DRm, DRn	1111nnn0mmm00001	DRn-DRm→DRn	6	—		○	
FTRC FRm, FPUL	1111mmmm00111101	(long)FRm→FPUL	1	—	○	○	
FTRC DRm, FPUL	1111mmm000111101	(long)DRm→FPUL	2	—		○	

5.1.8 FPU に関する CPU 命令

表 5.9 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	互換性		
					SH2E	SH4	新規SH-2A/SH2A-FPU
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, FPSCR→(Rn)	1	—	○	○	
STS.L FPUL,@-Rn	0100nnnn01010010	Rn-=4, FPUL→(Rn)	1	—	○	○	

5.1.9 ビット操作命令

表 5.10 ビット操作命令

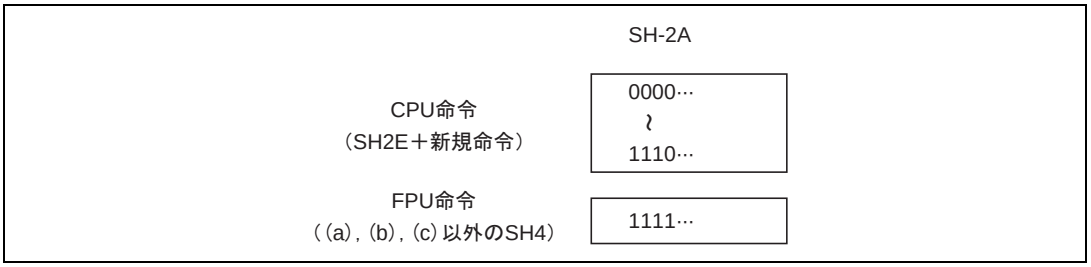
命　令	命令コード	動　作	実行ステート	Tビット	互換性		
					SH2E	SH4	新規 SH-2A/ SH2A-FPU
BAND.B #imm3,@(disp12,Rn)	0011nnnn0iii1001 0100dddddddddddd	(imm of (disp+ Rn))&T → T	3	演算結果			○
BANDNOT.B #imm3,@(disp12,Rn)	0011nnnn0iii1001 1100dddddddddddd	~(imm of (disp+ Rn))&T → T	3	演算結果			○
BCLR.B #imm3,@(disp12,Rn)	0011nnnn0iii1001 0000dddddddddddd	0→ (imm of (disp+ Rn))	3	—			○
BCLR #imm3, Rn	10000110nnnn0iii	0→ imm of Rn	1	—			○
BLD.B #imm3,@(disp12,Rn)	0011nnnn0iii1001 0011dddddddddddd	(imm of (disp+Rn)) → T	3	演算結果			○
BLD #imm3, Rn	10000111nnnn1iii	imm of Rn →T	1	演算結果			○
BLDNOT.B #imm3,@(disp12,Rn)	0011nnnn0iii1001 1011dddddddddddd	~(imm of (disp+Rn)) → T	3	演算結果			○
BOR.B #imm3,@(disp12,Rn)	0011nnnn0iii1001 0101dddddddddddd	(imm of (disp+ Rn)) | T → T	3	演算結果			○
BORNOT.B #imm3,@(disp12,Rn)	0011nnnn0iii1001 1101dddddddddddd	~(imm of (disp+ Rn)) | T → T	3	演算結果			○
BSET.B #imm3,@(disp12,Rn)	0011nnnn0iii1001 0001dddddddddddd	1→ (imm of (disp+Rn))	3	—			○
BSET #imm3, Rn	10000110nnnn1iii	1→ imm of Rn	1	—			○
BST.B #imm3,@(disp12,Rn)	0011nnnn0iii1001 0010dddddddddddd	T→(imm of (disp+Rn))	3	—			○
BST #imm3, Rn	10000111nnnn0iii	T→ imm of Rn	1	—			○
BXOR.B #imm3, @(disp12, Rn)	0011nnnn0iii1001 0110dddddddddddd	(imm of (disp+ Rn)) ^ T → T	3	演算結果			○

6. 各命令の説明

6.1 新規命令の概要

SH-2A/SH2A-FPU では、SH2E の CPU 命令（最上位 4 ビットが 0000～1110 の命令コード）と SH4 の FPU 命令（最上位 4 ビットが 1111 の命令コード）に割り当てた命令コード以外の空いている箇所に新規命令を追加しています。ただし、SH4 の FPU 命令である、(a) XDm/XDn を指定する FMOV 命令と(b)FRCHG 命令と(c) FIPR、FTRV 命令については、サポートしていません。

本章では、その新規命令の各々について、詳しく説明します。



新規命令は下記(1)～(14)です。(1)～(3)は、32 ビット固定長命令、(4)～(14)は、16 ビット固定長命令です。

(1) イミディエイト転送命令

MOVI20, MOVI20S

命令コード内の 20 ビットイミディエイトデータをレジスタに転送する命令です。本命令との組み合わせで、28 ビットアドレスを容易に生成できるため、最大 256M バイトまでオンチップメモリのアドレスを指定することが可能となります。

(2) 構造体アクセス用命令

MOV.B/W/L Rm,@(disp12, Rn), MOV.B/W/L @(disp12, Rm), Rn
MOVU.B/W @(disp12, Rm), Rn
FMOV.S FRm, @(disp12, Rn), FMOV.S @(disp12, Rm), FRn
FMOV.D DRm, @(disp12, Rn), FMOV.D @(disp12, Rm), DRn

命令コードの中に配置した 12 ビットのディスプレースメントを指定してメモリを参照する命令です。また、ゼロ拡張の実行を自動的に行う、符号なしロード命令 MOVU を追加しています。

(3) ビット操作命令（メモリ対象）

BAND.B #imm3, @(disp12, Rn), BOR.B #imm3, @(disp12,Rn)
BCLR.B #imm3, @(disp12, Rn), BSET.B #imm3, @(disp12, Rn)
BST.B #imm3, @(disp12, Rn), BLD.B #imm3, @(disp12, Rn)
BXOR.B #imm3, @(disp12, Rn)
BANDNOT.B #imm3, @(disp12, Rn), BORNOT.B #imm3, @(disp12,Rn)
BLDNOT.B #imm3, @(disp12, Rn)

BAND.B 命令と BOR.B 命令と BXOR.B 命令は、メモリ中の 1 ビットと T ビットとの論理演算を行

い、その結果を T ビットに格納する命令です。BCLR.B 命令と BSET.B 命令は、メモリ中の 1 ビットを操作する命令です。BST.B 命令と BLD.B 命令は、メモリ中の 1 ビットと T ビット間の転送を実行する命令です。

また、BANDNOT.B 命令と BORNOT.B 命令は、メモリ中の 1 ビットを反転した値と T ビットとの論理演算を行い、その結果を T ビットに格納する命令です。

BLDNOT.B 命令は、メモリ中の 1 ビットを反転し T ビットに格納する命令です。

指定されたビット以外は影響を受けません。

(4) ビット操作命令（汎用レジスタ対象）

BCLR #imm3, Rn, BSET #imm3, Rn

BST #imm3, Rn , BLD #imm3, Rn

BCLR 命令と BSET 命令は､汎用レジスタ Rn の LSB 側 8 ビット中の 1 ビットを操作する命令です。BST 命令と BLD 命令は、汎用レジスタ Rn の LSB 側 8 ビット中の 1 ビットと T ビット間の転送を実行する命令です。指定されたビット以外は影響を受けません。

(5) 乗算結果 Rn 格納命令

MULR

32 ビット×32 ビットの乗算を行い、乗算結果の下位 32 ビットを汎用レジスタ Rn に格納する命令です。

(6) 一括除算命令

DIVS, DIVU

32 ビット÷32 ビットの除算を一括で行う命令です｡DIVU 命令は無符号のデータを除算する命令、DIVS 命令は符号付きのデータを除算する命令です。

(7) 飽和値比較命令

CLIPS, CLIPU

飽和値との比較を行い、汎用レジスタ Rn が飽和上限値を上回るときは飽和上限値を、飽和下限値を下回るときは飽和下限値を、汎用レジスタ Rn に格納します。なお、バイト、ワードの飽和値のみをサポートしています。

(8) バレルシフト命令

SHAD, SHLD

任意のビットをシフトする命令です。算術シフトと論理シフトの 2 種類の命令を用意しています。

(9) 複数レジスタ退避・復帰命令

MOVML, MOVMU

複数の連続したレジスタをメモリへ退避する、あるいはメモリから復帰する命令です。汎用レジスタ Rn を指定し、指定した Rn より上位もしくは下位の連続した汎用レジスタの退避や復帰をすることが可能となります。

(10) T ビット反転転送命令

MOVRT, NOTT

T ビットを反転して汎用レジスタ Rn あるいは T ビットに転送する命令です。

(11) レジスタバンク関連命令

RESBANK, STBANK, LDBANK

割り込み処理の高速化のために搭載した、レジスタバンクに関する命令です。

(12) 逆スタック転送命令

MOV.B/W/L

スタックの伸長する方向が逆である転送命令です。

(13) 遅延スロットなし無条件分岐命令

JSR/N, RTS/N

不要な NOP 命令の削減によりコードサイズを削減するため、遅延スロットを持たない命令を提供しています。

(14) キャッシュ関連命令

PREF

SH3-DSP のキャッシュ関連命令を提供しています。

6.2 命令説明のフォーム

この章の見方
以下の形式で説明します。

命令の名称　命令の機能（英文）　命令の分類

命令の機能　命令セット互換

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。動作を理解するための参考として記述してあります。
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  int               Read_Int  (unsigned  long  Addr) ;
unsigned  long              Read_Long (unsigned  long  Addr) ;
unsigned  double            Read_Quad (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  long              Read_Bank_Long (unsigned  long  Addr) ;
```

アドレス Addr の内容が指すレジスタバンク内のエントリの内容を返します。

```
unsigned  char              Write_Byte (unsigned long Addr, unsigned long Data) ;
unsigned  short             Write_Word (unsigned long Addr, unsigned long Data) ;
unsigned  int               Write_Int (unsigned long Addr, unsigned long Data) ;
unsigned  long              Write_Long (unsigned long Addr, unsigned long Data) ;
unsigned  double            Write_Quad (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 番地へから

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。

```
unsigned  long  Write_Bank_Long (unsigned  long Add, unsigned long Data);
```

アドレス Addr の内容が指すレジスタバンク内のエントリに、データ Data を書き込みます。

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

各レジスタの本体

```
struct  BANK {
            unsigned  long  Rn_BANK[15];
            unsigned  long  GBR_BANK;
            unsigned  long  MACH_BANK;
            unsigned  long  MACL_BANK;
            unsigned  long  PR_BANK;
            unsigned  long  IVN;
} ;
BANK  Register_Bank[512];
```

レジスタバンクの構造の定義

（VTO：割り込みベクタテーブルアドレスオフセット）

```
struct SR0 {
            unsigned  long  dummy0:17 ;
            unsigned  long  BO0:1
            unsigned  long  CS0:1;
            unsigned  long  dummy1:3;
            unsigned  long  M0:1 ;
            unsigned  long  Q0:1 ;
            unsigned  long  I0:4 ;
            unsigned  long  dummy2:2;
            unsigned  long  S0:1;
            unsigned  long  T0:1;
} ;
```

SR の構造の定義

```
#define BO ( ( * (struct SR0 * ) ( &SR) ).BO0)
#define CS ( ( * (struct SR0 * ) ( &SR) ).CS0)
#define M ( ( * (struct SR0 * ) ( &SR) ).M0)
#define Q ( ( * (struct SR0 * ) ( &SR) ).Q0)
#define I ( ( * (struct SR0 * ) ( &SR) ).I0)
#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

浮動小数点用の定義文です。

```
#define PZERO        0
#define NZERO        1
#define DENORM       2
#define NORM         3
#define PINF         4
#define NINF         5
#define qNaN         6
#define sNaN         7
#define EQ           0
#define GT           1
#define LT           2
#define UO           3
#define INVALID      4
#define FADD         0
#define FSUB         1

#define CAUSE        0x0003f000/* FPSCR(bit17-12) */
#define SET_E        0x00020000/* FPSCR(bit17) */
#define SET_V        0x00010040/* FPSCR(bit16,6) */
#define SET_Z        0x00008020/* FPSCR(bit15,5) */
#define SET_O        0x00004010/* FPSCR(bit14,4) */
#define SET_U        0x00002008/* FPSCR(bit13,3) */
#define SET_I        0x00001004/* FPSCR(bit12,2) */
#define ENABLE_VOUI  0x00000b80/* FPSCR(bit11,9-7) */
#define ENABLE_V     0x00000800/* FPSCR(bit11) */
#define ENABLE_Z     0x00000400/* FPSCR(bit10) */
```

```
#define ENABLE_OUI   0x00000380/* FPSCR(bit9-7) */
#define ENABLE_I     0x00000080/* FPSCR(bit7) */
#define FLAG         0x0000007C/* FPSCR(bit6-2) */

#define FPSCR_FR     FPSCR>>21&1
#define FPSCR_PR     FPSCR>>19&1
#define FPSCR_DN     FPSCR>>18&1
#define FPSCR_I      FPSCR>>12&1
#define FPSCR_RM     FPSCR&1
#define FR_HEX       frf.l[ FPSCR_FR]
#define FR           frf.f[ FPSCR_FR]
#define DR_HEX       frf.l[ FPSCR_FR]
#define DR           frf.d[ FPSCR_FR]

union {
      int  l[2][16];
      float  f[2][16];
      double d[2][8];
} frf;
int FPSCR;

int sign_of(int n)
{
      return(FR_HEX[n]>>31);
}
int data_type_of(int n) {
int abs;
      abs = FR_HEX[n] & 0x7fffffff;
      if(FPSCR_PR == 0) {/* 単精度 */
         if(abs < 0x00800000){
            if((FPSCR_DN == 1) || (abs == 0x00000000)){
               if(sign_of(n) == 0)   {zero(n, 0); return(PZERO);}
               else                  {zero(n, 1); return(NZERO);}
            }
            else                     return(DENORM);
         }
         else if(abs < 0x7f800000)   return(NORM);
         else if(abs == 0x7f800000) {
            if(sign_of(n) == 0)      return(PINF);
            else                     return(NINF);
```

```
         }
         else if(abs < 0x7fc00000)   return(qNaN);
         else                        return(sNaN);
      }
      else {   /* 倍精度 */
         if(abs < 0x00100000){
            if((FPSCR_DN == 1) || ((abs == 0x00000000) && (FR_HEX[n+1] == 0x00000000){
                if(sign_of(n) == 0)  {zero(n, 0); return(PZERO);}
                else  {zero(n, 1); return(NZERO);}
            }
            else               return(DENORM);
         }
}
         else if(abs < 0x7ff00000)             return(NORM);
         else if((abs == 0x7ff00000) &&
                (FR_HEX[n+1] == 0x00000000)) {
            if(sign_of(n) == 0)return(PINF);
            else                return(NINF);
         }
         else if(abs < 0x7ff80000)return(qNaN);
         else                       return(sNaN);
      }
}
void register_copy(int m,n)
{
                        FR[n]   = FR[m];
      if(FPSCR_PR == 1)  FR[n+1] = FR[m+1];
}
void normal_faddsub(int m,n,type)
{
union {
      float f;
      int l;
}     dstf,srcf;
union {
      double d;
      int l[2];
}     dstd,srcd;
union {             /* “long double” のフォーマット:*/
```

```
        long double x; /*1-bit 符号*/
        int l[4];      /*15-bit 指数*/
}       dstx;          /*112-bit 小数*/
        if(FPSCR_PR == 0) {
           if(type == FADD)srcf.f =  FR[m];
           else           srcf.f = -FR[m];
           dstd.d = FR[n]; /* 単精度から倍精度への変換*/
           dstd.d += srcf.f;
           if(((dstd.d == FR[n]) && (srcf.f != 0.0)) ||
              ((dstd.d == srcf.f) && (FR[n] != 0.0))) {
              set_I();
              if(sign_of(m)^ sign_of(n)) {
                  dstd.l[1] -= 1;
              if(dstd.l[1] == 0xffffffff) dstd.l[0] -= 1;
              }
           }
           if(dstd.l[1] & 0x1fffffff) set_I();
           dstf.f += srcf.f; /* 近傍への丸め */
           if(FPSCR_RM == 1) {
              dstd.l[1] &= 0xe0000000; /* 0への丸め */
              dstf.f = dstd.d;
           }
           check_single_exception(&FR[n],dstf.f);
        } else {
           if(type == FADD)   srcd.d =  DR[m>>1];
           else               srcd.d = -DR[m>>1];
           dstx.x = DR[n>>1]; /* 倍精度から拡張倍精度への変換 */
           dstx.x += srcd.d;
           if(((dstx.x == DR[n>>1]) && (srcd.d != 0.0)) ||
              ((dstx.x == srcd.d) && (DR[n>>1] != 0.0)) ) {
              set_I();
              if(sign_of(m)^ sign_of(n)) {
                 dstx.l[3] -= 1;
                 if(dstx.l[3] == 0xffffffff) {dstx.l[2] -= 1;
                 if(dstx.l[2] == 0xffffffff) {dstx.l[1] -= 1;
                 if(dstx.l[1] == 0xffffffff) {dstx.l[0] -= 1;}}}
              }
           }
           if((dstx.l[2] & 0x0fffffff) || dstx.l[3]) set_I();
```

```
            dst.d += srcd.d; /*近傍への丸め */
            if(FPSCR_RM == 1) {
               dstx.l[2] &= 0xf0000000; /* 0への丸め */
               dstx.l[3]  = 0x00000000;
               dst.d = dstx.x;
            }
            check_double_exception(&DR[n>>1] ,dst.d);
         }
   }
   void normal_fmul(int m,n)
   {
   union {
         float f;
               int l;
   }     tmpf;
   union {
         double d;
         int l[2];
   }     tmpd;
   union {
         long double x;
         int l[4];
   }     tmpx;
         if(FPSCR_PR == 0) {
            tmpd.d = FR[n]; /* 単精度から倍精度 */
            tmpd.d *= FR[m]; /* 正確に作成 */
            tmpf.f *= FR[m]; /* 近傍への丸め */
            if(tmpf.f != tmpd.d) set_I();
            if((tmpf.f > tmpd.d) && (FPSCR_RM == 1)) {
               tmpf.l -= 1; /* 0への丸め */
            }
            check_single_exception(&FR[n],tmpf.f);
         } else {
            tmpx.x = DR[n>>1]; /* 単精度から倍精度 */
            tmpx.x *= DR[m>>1]; /* 正確に作成 */
            tmpd.d *= DR[m>>1]; /* 近傍への丸め */
            if(tmpd.d != tmpx.x) set_I();
            if(tmpd.d > tmpx.x) && (FPSCR_RM == 1)) {
               tmpd.l[1] -= 1; /* 0への丸め */
```

```
            if(tmpd.l[1] == 0xffffffff) tmpd.l[0] -= 1;
         }
         check_double_exception(&DR[n>>1], tmpd.d);
      }
}
void check_single_exception(float *dst,result)
{
union {
      float f;
      int l;
}     tmp;
float abs;
      if(result < 0.0)  tmp.l = 0xff800000; /* -無限大 */
      else              tmp.l = 0x7f800000; /* +無限大 */
      if(result == tmp.f) {
         set_O(); set_I();
         if(FPSCR_RM == 1){
            tmp.l -= 1; /* 正規化数の最大値 */
            result = tmp.f;
         }
      }
      if(result < 0.0)  abs = -result;
      else              abs =  result;
      tmp.l = 0x00800000; /* 正規化数の最小値 */
      if(abs < tmp.f) {
         if((FPSCR_DN == 1) && (abs != 0.0)) {
            set_I();
            if(result < 0.0)  result = -0.0; /* 非正規化数を0にする。 */
            else              result =  0.0;
         }
         if(FPSCR_I == 1) set_U();
      }
      if(FPSCR & ENABLE_OUI) fpu_exception_trap();
      else                   *dst = result;
}
void check_double_exception(double *dst,result)
{
union {
      double d;
```

```
        int l[2];
    }     tmp;
    double abs;
        if(result < 0.0)  tmp.l[0] = 0xfff00000; /* -無限大 */
        else              tmp.l[0] = 0x7ff00000; /* +無限大 */
                    tmp.l[1] = 0x00000000;
        if(result == tmp.d)
           set_O(); set_I();
           if(FPSCR_RM == 1) {
              tmp.l[0] -= 1;
              tmp.l[1] = 0xffffffff;
              result = tmp.d; /* 正規化数の最大値 */
           }
        }
        if(result < 0.0)abs = -result;
        else        abs =  result;
        tmp.l[0] = 0x00100000; /* 正規化数の最小値 */
        tmp.l[1] = 0x00000000;
        if(abs < tmp.d) {
           if((FPSCR_DN == 1) && (abs != 0.0)) {
              set_I();
              if(result < 0.0)  result = -0.0; /* 非正規化数を０にする。 */
                 else           result =  0.0;
           }
           if(FPSCR_I == 1) set_U();
        }
        if(FPSCR & ENABLE_OUI)fpu_exception_trap();
        else                      *dst = result;
    }
    int check_product_invalid(int m,n)
    {
        return(check_product_infinity(m,n)  &&
              ((data_type_of(m) == PZERO) || (data_type_of(n) == PZERO) ||
               (data_type_of(m) == NZERO) || (data_type_of(n) == NZERO)));
    }
    int check_product_infinity(int m,n)
    {
        return((data_type_of(m) == PINF) || (data_type_of(n) == PINF) ||
               (data_type_of(m) == NINF) || (data_type_of(n) == NINF));
```

```
}
int check_positive_infinity(int m,n)
{
  return(((check_product_infinity(m,n) && (~sign_of(m)^ sign_of(n))) ||
       ((check_product_infinity(m+1,n+1) && (~sign_of(m+1)^ sign_of(n+1))) ||
       ((check_product_infinity(m+2,n+2) && (~sign_of(m+2)^ sign_of(n+2))) ||
       ((check_product_infinity(m+3,n+3) && (~sign_of(m+3)^ sign_of(n+3))));
}
int check_negative_infinity(int m,n)
{
  return(((check_product_infinity(m,n) && (sign_of(m)^ sign_of(n))) ||
       ((check_product_infinity(m+1,n+1) && (sign_of(m+1)^ sign_of(n+1))) ||
       ((check_product_infinity(m+2,n+2) && (sign_of(m+2)^ sign_of(n+2))) ||
       ((check_product_infinity(m+3,n+3) && (sign_of(m+3)^ sign_of(n+3))));
}
void clear_cause () {FPSCR &= ~CAUSE;}
void set_E() {FPSCR |= SET_E; fpu_exception_trap();}
void set_V() {FPSCR |= SET_V;}
void set_Z() {FPSCR |= SET_Z;}
void set_O() {FPSCR |= SET_O;}
void set_U() {FPSCR |= SET_U;}
void set_I() {FPSCR |= SET_I;}
void invalid(int n)
{
      set_V();
      if((FPSCR & ENABLE_V) == 0 qnan(n);
      else    fpu_exception_trap();
}

void dz(int n,sign)
{
      set_Z();
      if((FPSCR & ENABLE_Z) == 0 inf(n,sign);
      else    fpu_exception_trap();
}
void zero(int n,sign)
{
      if(sign == 0)        FR_HEX [n]   = 0x00000000;
      else                 FR_HEX [n]   = 0x80000000;
      if (FPSCR_PR==1)     FR_HEX [n+1] = 0x00000000;
```

```
}
void inf(int n,sign) {
      if (FPSCR_PR==0) {
         if(sign == 0)     FR_HEX [n]   = 0x7f800000;
         else              FR_HEX [n]   = 0xff800000;
      } else {
         if(sign == 0)     FR_HEX [n]   = 0x7ff00000;
         else              FR_HEX [n]   = 0xfff00000;
                           FR_HEX [n+1] = 0x00000000;
      }
}
void qnan(int n)
{
      if (FPSCR_PR==0)     FR[n]   = 0x7fbfffff;
      else {               FR[n]   = 0x7ff7ffff;
                           FR[n+1] = 0xffffffff;
      }
}
```

(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
.align 32	32 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。

6.3　新規命令

6.3.1　BAND　　Bit AND　　ビット操作命令

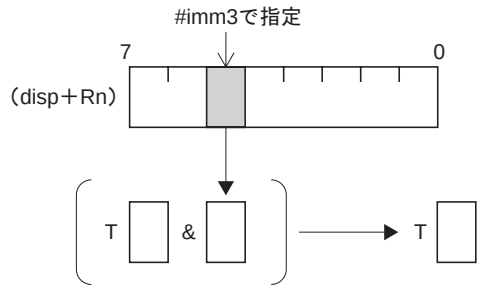
ビット論理積　　SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
BAND.B #imm3,@(disp12,Rn)	(<imm>of（disp+Rn）)&T→T	0011nnnn0iii1001 0100dddddddddddd	3	演算結果

(1)　説明

(disp＋Rn)が示すアドレスのメモリ中の指定された 1 ビットと、T ビットとの論理積をとり、その結果を T ビットに格納します。ビット番号は、3 ビットのイミディエイトデータで指定します。本命令では、バイト単位でメモリからデータを読み出します。

```
BAND.B #imm3, @(disp12, Rn)
```



(2)　動作内容

```
BANDM (long d, long i, long n)  /*BAND.B #imm3, @(disp12, Rn) */
 {
   long disp, imm, temp, assignbit;

     disp = (0x00000FFF & (long)d );
     imm= (0x00000007&(long)i);
     temp= (long) Read_Byte ( R[n]+disp);
     assignbit =(0x00000001<<imm)&temp;
     if((T==0)||(assignbit==0)) T=0;
     else T=1;
     PC+=4;
}
```

(3) 使用例

```
BAND.B#H'5, @(2, R0)          ;実行前  @(R0+2)=H'DF, T=1
                              ;実行後  @(R0+2)=H'DF, T=0
```

6.3.2　BANDNOT　Bit ANDNOT　　ビット操作命令

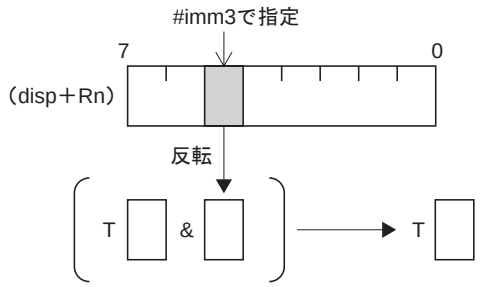
ビットノット論理積　　SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
BANDNOT.B #imm3,@(disp12,Rn)	~(<imm>of（disp+Rn）)&T→T	0011nnnn0iii1001 1100dddddddddddd	3	演算結果

(1)　説明

(disp＋Rn)が示すアドレスのメモリ中の指定された 1 ビットを反転した値と、T ビットとの論理積をとり、その結果を T ビットに格納します。ビット番号は、3 ビットのイミディエイトデータで指定します。本命令では、バイト単位でメモリからデータを読み出します。

```
BANDNOT.B #imm3, @(disp12, Rn)
```



(2)　動作内容

```
BANDNOTM (long d, long i, long n)  /*BANDNOT.B #imm3, @(disp12, Rn) */
 {
   long disp, imm, temp, assignbit;

     disp = (0x00000FFF & (long)d );
     imm= (0x00000007&(long)i);
     temp= (long) Read_Byte ( R[n]+disp);
     assignbit =(0x00000001<<imm)&temp;
     if((T==1)&&(assignbit==0)) T=1;
     else T=0;
     PC+=4;
}
```

(3) 使用例

```
BANDNOT.B#H'5, @(2, R0)        ;実行前  @(R0+2)=H'20, T=1
                               ;実行後  @(R0+2)=H'20, T=0
```

6.3.3　BCLR　　Bit CLeaR　　ビット操作命令

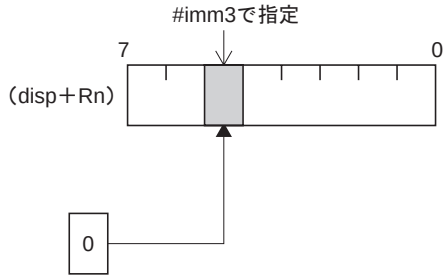
ビットクリア　　SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
BCLR.B #imm3,@(disp12,Rn)	0→(<imm>of（disp+Rn）)	0011nnnn0iii1001 0000dddddddddddd	3	—
BCLR　#imm3, Rn	0→ <imm>of Rn	10000110nnnn0iii	1	—

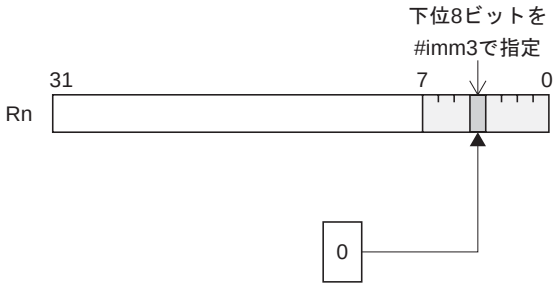
(1)　説明

(disp＋Rn)が示すアドレスのメモリあるいは汎用レジスタ Rn の LSB 側 8 ビット中の、指定された 1 ビットをクリアします。ビット番号は、3 ビットのイミディエイトデータで指定します。BCLR.B 命令では、バイト単位でメモリからデータを読み出した後、指定ビットのクリアを実行し、実行後のデータをバイト単位でメモリへ書き込みます。

```
BCLR.B #imm3, @(disp12, Rn)
```




```
BCLR #imm3, Rn
```



(2) 動作内容

```
BCLRM (long d, long i, long n)  /*BCLR.B #imm3, @(disp12, Rn) */
 {
   long disp, imm, temp;

     disp = (0x00000FFF & (long)d );
     imm= (0x00000007&(long)i);
     temp= (long) Read_Byte ( R[n]+disp);
     temp&=(~(0x00000001<<imm));
     Write_Byte (R[n]+disp, temp);
     PC+=4;
}

BCLR (long i, long n)  /*BCLR #imm3, Rn */
 {
   long imm, temp;

   imm= (0x00000007 &(long)i);
   R[n]&=(~(0x00000001<<imm))
   PC+=2;
  }
```

(3) 使用例

```
BCLR.B#H'5, @(2, R0)          ;実行前  @(R0+2)=H'FF
                              ;実行後  @(R0+2)=H'DF
BCLR#H'4, R0                  ;実行前  R0=H'FFFFFFFF
                              ;実行後  R0=H'FFFFFFFF
```

6.3.4　BLD　　Bit LoaD　　ビット操作命令

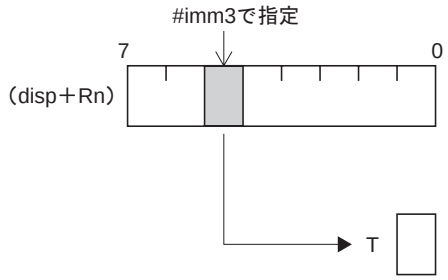
ビットロード　　SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
BLD.B #imm3,@(disp12,Rn)	(<imm>of（disp+Rn）) →T	0011nnnn0iii1001 0011dddddddddddd	3	演算結果
BLD　#imm3, Rn	<imm>of Rn→ T	10000111nnnn1iii	1	演算結果

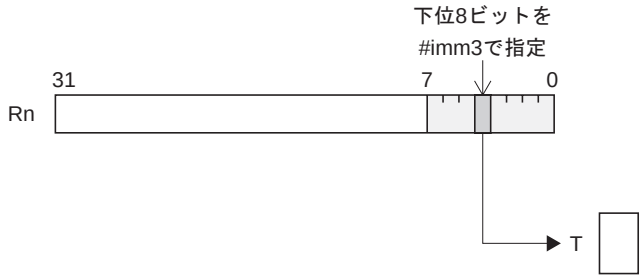
(1)　説明

(disp＋Rn)が示すアドレスのメモリあるいは汎用レジスタ Rn の LSB 側８ビット中の、指定された 1 ビットをＴビットに格納します。ビット番号は、3 ビットのイミディエイトデータで指定します。BLD.B 命令では、バイト単位でメモリからデータを読み出します。

```
BLD.B #imm3, @(disp12, Rn)
```




```
BLD #imm3, Rn
```



(2) 動作内容

```
BLDM (long d, long i, long n)  /*BLD.B #imm3, @(disp12, Rn) */
 {
   long disp, imm, temp,assignbit;

  disp = (0x00000FFF & (long)d );
  imm= (0x00000007&(long)i);
  temp = (long) Read_Byte ( R[n]+disp);
  assignbit=(0x00000001<<imm)&temp;
  if(assignbit==0) T=0;
  else T=1;
  PC+=4;
}
BLD (long i, long n)  /*BLD #imm3, Rn */
 {
   long imm, assignbit;

   imm= (0x00000007&(long)i);
   assignbit=(0x00000001<<imm)&R[n];
   if(assignbit ==0) T=0;
   else T=1;
   PC+=2;
  }
```

(3) 使用例

```
BLD.B#H'5, @(2, R0)          ;実行前  @(R0+2)=H'20, T=0
                             ;実行後  @(R0+2)=H'20, T=1
BLD#H'4, R0                  ;実行前  R0=H'000000EF, T=1
                             ;実行後  R0=H'000000EF, T=0
```

6.3.5　BLDNOT　Bit LoaD　NOT ビット操作命令

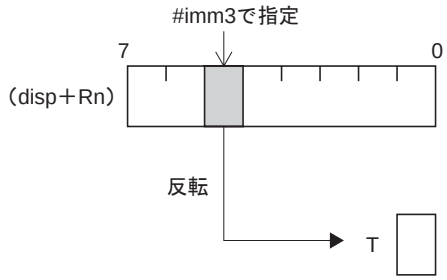
ビットノットロード　　SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
BLDNOT.B #imm3,@(disp12,Rn)	~(<imm>of（disp+Rn）) →T	0011nnnn0iii1001 1011dddddddddddd	3	演算結果

(1)　説明

(disp＋Rn)が示すアドレスのメモリ中の指定された 1 ビットを反転しＴビットに格納します。ビット番号は、3 ビットのイミディエイトデータで指定します。BLDNOT.B 命令では、バイト単位でメモリからデータを読み出します。

```
BLDNOT.B #imm3, @(disp12, Rn)
```



(2)　動作内容

```
BLDNOTM (long d, long i, long n)  /*BLDNOT.B #imm3, @(disp12, Rn) */
 {
   long disp, imm, temp,assignbit;

  disp = (0x00000FFF & (long)d );
  imm= (0x00000007&(long)i);
  temp = (long) Read_Byte ( R[n]+disp);
  assignbit=(0x00000001<<imm)&temp;
  if(assignbit==0) T=1;
  else T=0;
  PC+=4;
}
```

(3)　使用例

```
BLDNOT.B#H'5, @(2, R0)          ;実行前  @(R0+2)=H'20, T=1
                                ;実行後  @(R0+2)=H'20, T=0
```

6.3.6 BOR Bit OR ビット操作命令

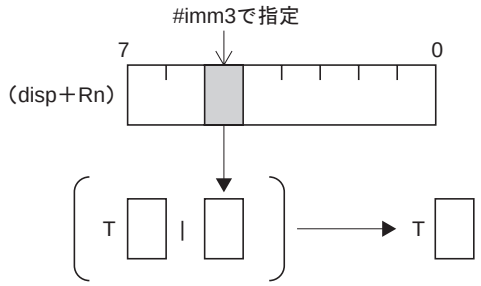
ビット論理和 SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
BOR.B #imm3,@(disp12,Rn)	(<imm>of（disp+Rn）) | T→T	0011nnnn0iii1001 0101dddddddddddd	3	演算結果

(1) 説明

(disp＋Rn)が示すアドレスのメモリ中の指定された 1 ビットと、T ビットとの論理和をとり、その結果を T ビットに格納します。ビット番号は、3 ビットのイミディエイトデータで指定します。本命令では、バイト単位でメモリからデータを読み出します。

```
BOR.B #imm3, @(disp12, Rn)
```



(2) 動作内容

```
BORM (long d, long i, long n)  /*BOR.B #imm3, @(disp12, Rn) */
 {
   long disp, imm, temp, assignbit;

     disp = (0x00000FFF & (long)d );
     imm= (0x00000007&(long)i);
     temp= (long) Read_Byte ( R[n]+disp);
     assignbit =(0x00000001<<imm)&temp;
     if((T==0)&&(assignbit==0)) T=0;
     else T=1;

     PC+=4;
}
```

(3) 使用例

```
BOR.B#H'5, @(2, R0)          ;実行前  @(R0, 2)=H'20, T=0
                             ;実行後  @(R0, 2)=H'20, T=1
```

6.3.7 BORNOT Bit ORNOT ビット操作命令

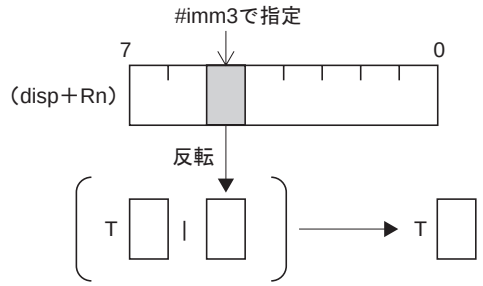
ビットノット論理和 SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
BORNOT.B #imm3,@(disp12,Rn)	~(<imm>of（disp+Rn）) | T→T	0011nnnn0iii1001 1101dddddddddddd	3	演算結果

(1) 説明

(disp＋Rn)が示すアドレスのメモリ中の指定された 1 ビットを反転した値と、T ビットとの論理和をとり、その結果を T ビットに格納します。ビット番号は、3 ビットのイミディエイトデータで指定します。本命令では、バイト単位でメモリからデータを読み出します。

```
BORNOT.B #imm3, @(disp12, Rn)
```



(2) 動作内容

```
BORNOTM (long d, long i, long n)  /*BORNOT.B #imm3, @(disp12, Rn) */
 {
   long disp, imm, temp, assignbit;

     disp = (0x00000FFF & (long)d );
     imm= (0x00000007&(long)i);
     temp= (long) Read_Byte ( R[n]+disp);
     assignbit =(0x00000001<<imm)&temp;
     if((T==1)||(assignbit==0)) T=1;
     else T=0;

     PC+=4;
}
```

(3) 使用例

```
BORNOT.B#H'5, @(2, R0)        ;実行前  @(R0+2)=H'DF, T=0
                              ;実行後  @(R0+2)=H'DF, T=1
```

6.3.8 BSET Bit SET ビット操作命令

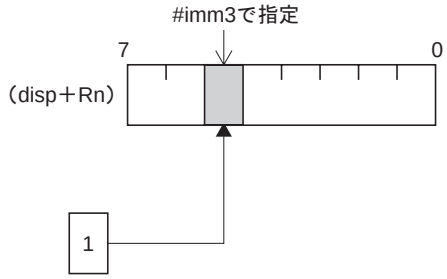
ビットセット SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
BSET.B #imm3,@(disp12,Rn)	1→(<imm>of（disp+Rn）)	0011nnnn0iii1001 0001dddddddddddd	3	—
BSET #imm3, Rn	1→<imm>of Rn	10000110nnnn1iii	1	—

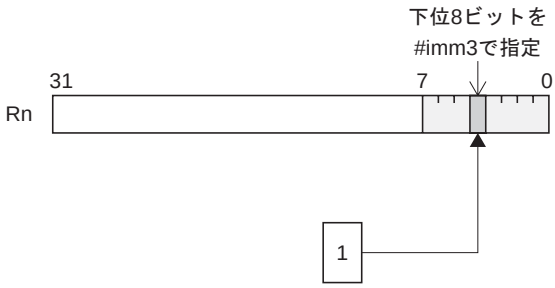
(1) 説明

(disp＋Rn)が示すアドレスのメモリあるいは汎用レジスタ Rn の LSB 側８ビット中の、指定された 1 ビットを 1 にセットします。ビット番号は、3 ビットのイミディエイトデータで指定します。BSET.B 命令では、バイト単位でメモリからデータを読み出した後、指定ビットを 1 にセットし、実行後のデータをバイト単位でメモリへ書き込みます。

```
BSET.B #imm3, @(disp12, Rn)
```




```
BSET #imm3, Rn
```



(2) 動作内容

```
BSETM (long d, long i, long n)  /*BSET.B #imm3, @(disp12, Rn) */
 {
   long disp, imm, temp;

     disp = (0x00000FFF & (long)d );
     imm= (0x00000007&(long)i);
     temp= (long) Read_Byte ( R[n]+disp);
     temp|=(0x00000001<<imm);
     Write_Byte (R[n]+disp, temp);
     PC+=4;
}


BSET (long i, long n)  /*BSET #imm3, Rn */
 {
   long imm, temp;

   imm= (0x00000007 &(long)i);
   R[n]|=(0x00000001<<imm);
   PC+=2;
  }
```

(3) 使用例

```
BSET.B#H'5, @(2, R0)          ;実行前  @(R0+2)=H'00
                              ;実行後  @(R0+2)=H'20
BSET#H'4, R0                  ;実行前  R0=H'00000000
                              ;実行後  R0=H'00000010
```

6.3.9 BST Bit STore ビット操作命令

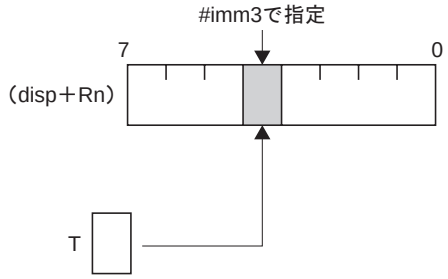
ビットストア SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
BST.B #imm3,@(disp12,Rn)	T →(<imm>of（disp+Rn）)	0011nnnn0iii1001 0010dddddddddddd	3	—
BST #imm3, Rn	T →<imm>of Rn	10000111nnnn0iii	1	—

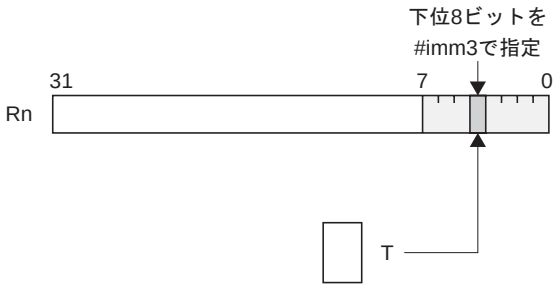
(1) 説明

(disp＋Rn)が示すアドレスのメモリあるいは汎用レジスタ Rn の LSB 側 8 ビット中の、指定された 1 ビットのロケーションに T ビットの内容を転送します。ビット番号は、3 ビットのイミディエイトデータで指定します。BST.B 命令では、バイト単位でメモリからデータを読み出した後、T ビットから指定ビットへの転送を実行し、実行後のデータをバイト単位でメモリへ書き込みます。

```
BST.B #imm3, @(disp12, Rn)
```




```
BST #imm3, Rn
```



(2) 動作内容

```
BSTM (long d, long i, long n)  /*BST.B #imm3, @(disp12, Rn) */
 {
   long disp, imm, temp;

  disp = (0x00000FFF & (long)d );
  imm= (0x00000007&(long)i);
  temp = (long) Read_Byte ( R[n]+disp);
if(T==0) temp&＝(~(0x00000001<<imm));
else  temp|=(0x00000001<<imm);
  Write_Byte (R[n]+disp, temp);

  PC+=4;

}

BST (long i, long n)  /*BST  #imm3, Rn */
 {
   long disp, imm;

  disp = (0x00000FFF & (long)d );
  imm= (0x00000007&(long)i);
  if (T==0) R[n]&＝(~(0x00000001<<imm));
else  R[n]|=(0x00000001<<imm);

PC+=2;
}
```

(3) 使用例

```
BST.B#H'4, @(2, R0)          ;実行前  @(R0+2)=H'FF, T=0
                             ;実行後  @(R0+2)=H'EF, T=0
BST#H'4, R0                  ;実行前  R0=H'00000000, T=1
                             ;実行後  R0=H'00000010, T=1
```

6.3.10 BXOR Bit exclusive OR ビット操作命令

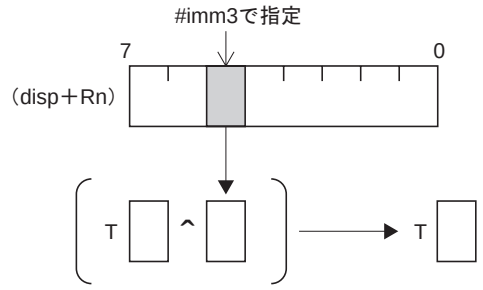
ビット排他的論理和 SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
BXOR.B #imm3,@(disp12,Rn)	(<imm>of（disp+Rn）)^T→T	0011nnnn0iii1001 0110dddddddddddd	3	演算結果

(1) 説明

(disp＋Rn)が示すアドレスのメモリの指定された 1 ビットと、T ビットとの排他的論理和をとり、その結果を T ビットに格納します。ビット番号は、3 ビットのイミディエイトデータで指定します。本命令では、バイト単位でメモリからデータを読み出します。

```
BXOR.B #imm3, @(disp12, Rn)
```



(2) 動作内容

```
BXORM (long d, long i, long n)  /*BXOR.B #imm3, @(disp12, Rn) */
 {
   long disp, imm, temp, assignbit;

     disp = (0x00000FFF & (long)d );
     imm= (0x00000007&(long)i);
     temp= (long) Read_Byte ( R[n]+disp);
     assignbit =(0x00000001<<imm)&temp;
     if (assignbit==0)
     {
     if(T==0) T=0;
     else  T=1;
     }
      else
      {
```

```
        if(T==0) T=1;
       else    T=0;
    }
       PC+=4;
   }
```

(3) 使用例

```
BXOR.B#H'5, @(2, R0)          ;実行前  @(R0+2)=H'FF, T=1
                              ;実行後  @(R0+2)=H'EF, T=0
```

6.3.11 CLIPS CLIP as Signed 算術演算命令

符号付き飽和値比較命令 SH-2A/SH2A-FPU（新規）

No.	書式	動作概略	命令コード	実行ステート	T ビット
1	CLIPS.B Rn	Rn＞(飽和上限値)のとき、	0100nnnn10010001	1	—
2	CLIPS.W Rn	(飽和上限値)→Rn, 1→CS Rn＜(飽和下限値)のとき、 (飽和下限値)→Rn, 1→CS	0100nnnn10010101	1	—

(1) 説明

飽和を判定する命令です。本命令では符号付きのデータを用います。汎用レジスタ Rn が飽和上限値を上回るときは飽和上限値を、飽和下限値を下回るときは飽和下限値を Rn に格納し、CS ビットを１にセットします。各命令に対する飽和上限値、飽和下限値は、以下の表のとおりです。

No.	命令	飽和下限値	飽和上限値
1	CLIPS.B Rn	H’FFFFFF80	H’0000007F
2	CLIPS.W Rn	H’FFFF8000	H’00007FFF

(2) 注意

汎用レジスタ Rn が飽和上限値を上回らないとき、あるいは飽和下限値を下回らないときは、CS ビットの値は変わりません。

(3) 動作内容

```
CLIPSB(long n)  /* CLIPS.B Rn*/
 {
  if ( R[n] > 0x0000007F)
   {
     R[n]=0x0000007F;
     CS=1;
   }
  else if (R[n] < 0xFFFFFF80)
   {
     R[n]=0xFFFFFF80;
     CS=1;
    }
  PC+2;
 }
```

```
CLIPSW(long n)  /* CLIPS.W Rn*/
 {
  if ( R[n] > 0x00007FFF)
    {
     R[n]=0x00007FFF;
     CS=1;
     }
  else  if (R[n] < 0xFFFF8000)
     {
       R[n]=0xFFFF8000;
       CS=1;
  PC+2;
 }
```

(4) 使用例

```
CLIPS.B R0                    ;実行前  R0=H'0000000F, CS=0
                              ;実行後  R0=H'0000000F, CS=0
CLIPS.B R1                    ;実行前  R1=H'00000080, CS=0
                              ;実行後  R1=H'0000007F, CS=1
CLIPS.W R0                    ;実行前  R0=H'FFFFFFF0, CS=0
                              ;実行後  R0=H'FFFFFFF0, CS=0
CLIPS.W R1                    ;実行前  R1=H'FFFF7000, CS=0
                              ;実行後  R1=H'FFFF8000, CS=1
```

6.3.12 CLIPU CLIP as Unsigned 算術演算命令

符号なし飽和値比較命令 SH-2A/SH2A-FPU（新規）

No.	書式	動作概略	命令コード	実行ステート	T ビット
1	CLIPU.B Rn	Rn＞(飽和値)のとき、	0100nnnn10000001	1	—
2	CLIPU.W Rn	(飽和値)→Rn, 1→CS	0100nnnn10000101	1	—

(1) 説明

飽和を判定する命令です。本命令では符号なしのデータを用います。汎用レジスタ Rn が飽和値を上回るときは飽和値を Rn に格納し、CS ビットを 1 にセットします。各命令に対する飽和値は、以下の表のとおりです。

No.	命令	飽和値
1	CLIPU.B Rn	H'000000FF
2	CLIPU.W Rn	H'0000FFFF

(2) 注意

汎用レジスタ Rn が飽和上限値を上回らないときは、CS ビットの値は変わりません。

(3) 動作内容

```
CLIPUB(long n)  /* CLIPU.B Rn*/
  {
   if ( R[n] > 0x000000FF)
     {
      R[n]=0x000000FF;
      CS=1;
     }
   PC+2;
  }

CLIPUW(long n)  /* CLIPU.W Rn*/
  {
   if ( R[n] > 0x0000FFFF)
     {
      R[n]=0x0000FFFF;
      CS=1;
     }
   PC+2;
```

```
    }
```

(4) 使用例

```
   CLIPU.B R0                    ;実行前  R0=H'0000000F, CS=0
                                 ;実行後  R0=H'0000000F, CS=0
   CLIPU.B R1                    ;実行前  R1=H'00000100, CS=0
                                 ;実行後  R1=H'000000FF, CS=1
   CLIPU.W R0                    ;実行前  R0=H'00000FFF, CS=0
                                 ;実行後  R0=H'00000FFF, CS=0
   CLIPU.W R1                    ;実行前  R1=H'00010000, CS=0
                                 ;実行後  R1=H'0000FFFF, CS=1
```

6.3.13 DIVS DIVide as Signed 算術演算命令

符号付き除算 SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
DIVS R0,Rn	符号付きで Rn÷R0→Rn	0100nnnn10010100	36	—

(1) 説明

汎用レジスタ Rn の 32 ビットの内容（被除数）を R0 の内容（除数）で除算を実行する命令です。本命令は、符号付き除算を実行し、単独で商を求めます。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。そのときに求めた乗余の符号は、被除数の符号と同じになります。

(2) 注意

オーバフロー例外は負の最大値(H'80000000)を－1 で割ったときに発生します。ゼロ除算例外は、ゼロによる除算のときに発生します。

本命令実行中に割り込みが発生すると実行を中止します。戻り番地は本命令の先頭アドレスとなり、本命令は再実行されます。

(3) 動作内容

```
DIVS (long n) /* DIVS R0, Rn */
{
  R[n]=R[n] / R[0];
   PC+=2;
 }
```

(4) 使用例

```
DIVS R0, R1                    ;R1(32 ビット)÷R0(32 ビット)=R1(32 ビット)：符号付き
```

6.3.14 DIVU　DIVide as Unsigned　算術演算命令

符号なし除算　　SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
DIVU R0, Rn	符号なしで Rn÷R0→Rn	0100nnnn10000100	34	—

(1) 説明

汎用レジスタ Rn の 32 ビットの内容（被除数）を R0 の内容（除数）で除算を実行する命令です。本命令は符号なし除算を実行し、単独で商を求めます。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。

(2) 注意

ゼロ除算例外はゼロによる除算のときに発生します。

本命令実行中に割り込みが発生すると実行を中止します。戻り番地は本命令の先頭アドレスとなり、本命令は再実行されます。

(3) 動作内容

```
DIVU  (long n)  /*  DIVU  R0, Rn */
{
 (unsigned long) R[n]= ( unsigned long )R[n] / (unsigned long )R[0];
 PC+=2;
}
```

(4) 使用例

```
DIVU R0, R1                    ;R1(32 ビット)÷R0(32 ビット)=R1(32 ビット)：符号なし
```

6.3.15 FMOV Floating-point MOVe 浮動小数点命令

浮動小数点転送 SH-2A/SH2A-FPU（新規）

No.	SZ	書式	動作概略	命令コード	実行ステート	T ビット
1	0	FMOV.S FRm, @(disp12,Rn)	FRm→(disp×4+Rn)	0011nnnnmmmm0001 0011dddddddddddd	1	—
2	1	FMOV.D DRm, @(disp12,Rn)	DRm→(disp×8+Rn)	0011nnnnmmm00001 0011dddddddddddd	2	—
3	0	FMOV.S @(disp12,Rm),FRn	(disp×4+Rm)→FRn	0011nnnnmmmm0001 0111dddddddddddd	1	—
4	1	FMOV.D @(disp12,Rm),DRn	(disp×8+Rm)→DRn	0011nnn0mmmm0001 0111dddddddddddd	2	—

(1) 説明

1. FRmの内容を(disp+Rn)が示すアドレスのメモリに転送します。
2. DRmの内容を(disp+Rn)が示すアドレスのメモリに転送します。
3. (disp+Rn)が示すアドレスのメモリの内容をFRnに転送します。
4. (disp+Rn)が示すアドレスのメモリの内容をDRnに転送します。

(2) 注意

ルネサスの「SuperH RISC engine アセンブラ」では、ディスプレースメントの値としてスケーリング（×4、×8）された値を用いて記述してください。

(3) 動作内容

```
void FMOV_INDEX_DISP12_STORE(int  m,n)  /*FMOV.S FRm, @(disp12,Rn) */
{
    long disp;

    disp = (0x00000FFF & (long)d);
     Write_Int ( R[n]+(disp<<2), FR[m]);
     PC +=4;
}

void FMOV_INDEX_DISP12_STORE_DR(int  m,n)
/*FMOV.D DRm, @(disp12,Rn) */
{
    long disp;

    disp = (0x00000FFF & (long)d);
```

```
    Write_Quad ( R[n]+(disp<<3), DR[m>>1]);
    PC +=4;
}



void FMOV_INDEX_DISP12_LOAD(int  m,n)  /*FMOV.S @(disp12,Rm), FRn */
{
   long disp;

   disp = (0x00000FFF & (long)d);
    FR[n] = Read_Int ( R[m]+(disp<<2));
    PC +=4;
}

void FMOV_INDEX_DISP12_LOAD_DR(int m,n)
                                        /*FMOV.D @(disp12,Rm), DRn */
{
   long disp;

   disp = (0x00000FFF & (long)d);
    DR[n>>1] = Read_Quad ( R[m]+(disp<<3));
    PC +=4;
}
```

(4) 使用例

```
FMOV.S FR0, @(2, R2)          ;実行前  FR0=H'12345670
                              ;実行後  @(R2+8)=H'12345670
FMOV.D DR0, @(2, R2)          ;実行前  FR0=H'01234567
                              ;        FR1=H'89ABCDEF
                              ;実行後  @(R2+16)=H'01234567
                              ;        @(R2+20)=H'89ABCDEF
FMOV.S @(2, R2), FR0          ;実行前  @(R2+8)=H'12345670
                              ;実行後  FR0=H'12345670
FMOV.D @(2, R2), DR0          ;実行前  @(R2+16)=H'01234567
                              ;        @(R2+20)=H'89ABCDEF
                              ;実行後  FR0=H'01234567
```

```
;       FR1=H'89ABCDEF
```

6.3.16　JSR/N　　Jump to SubRoutine with No delay slot　　分岐命令

遅延スロットなしサブルーチンプロシージャへの分岐　　SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
JSR/N @Rm	PC-2→PR,Rm→PC	0100mmmm01001011	3	—
JSR/N @@(disp8, TBR)	PC-2→PR, (disp×4+TBR)→PC	10000011dddddddd	5	—

(1)　説明

指定したアドレスのサブルーチンプロシージャへ分岐します。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、もしくは（disp×4 + TBR）番地のメモリから読み出されたアドレスへ分岐します。退避される PC は、本命令の 2 番地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

本命令は遅延分岐ではありません。

ルネサスの「SuperH RISC engine アセンブラ」では、ディスプレースメントの値としてスケーリング（×4）された値を用いて記述してください。

(3)　動作内容

```
JSRN (long m)  /* JSR/N  @Rm, */
{
   unsigned  long  temp;

   temp=PC;
   PR=PC-2;
   PC=R[m]+4;
}

JSRNM (long d )  /* JSR/N  @@(disp8, TBR) */
{
   unsigned  long  temp;
   long disp;

   temp=PC;
   PR=PC-2;
   disp=(0x000000FF & d);
```

```
    PC=Read_Long(TBR+(disp<<2))+4;
}
```

(4) 使用例

```
                MOV.L JSRN_TABLE, R0;R0=TRGET のアドレス
                JSR/N @R0               ;TRGET へ分岐します。
                ADD    R0, R1           ;←プロシージャからの戻り先
                                                （PR の内容）です。
                . . . . . . . . .
                .align 4
JSRN_TABLE:     .data.1 TRGET           ;ジャンプテーブル
TRGET:          NOP                     ;←プロシージャの入口
                MOV   R2, R3            ;
                RTS/N                   ;上記 ADD 命令に戻ります。


TBR+H'08        .data.1  FFFF7F80       ;
                . . . . . . . . .
                JSR/N  @@(2, TBR)       ;TBR+H'08 番地の内容のアドレスに分岐します。
                ADD    R0, R1           ; ←プロシージャからの戻り先
                                                （PR の内容）です。
                . . . . . . . . .
FFFF7F80        NOP                     ;←プロシージャの入口
FFFF7F82        MOV   R2, R3            ;
FFFF7F84        RTS/N                   ;上記 ADD 命令に戻ります。
```

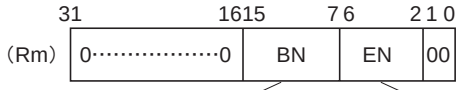
6.3.17 LDBANK LoaD register BANK システム制御命令

指定レジスタバンクエントリへの転送　　SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
LDBANK @Rm, R0	（指定レジスタバンクエントリ）→ R0	0100mmmm11100101	6	—

(1) 説明

汎用レジスタ Rm の内容が指すレジスタバンク中の 1 エントリを汎用レジスタ R0 に転送します。レジスタバンクの番号とバンク内に格納されているレジスタを、汎用レジスタ Rm で指定します。



BN：バンク番号フィールド
EN：エントリ番号フィールド

BN	レジスタバンク
000000000	バンク0
000000001	バンク1
000000010	バンク2
000000011	バンク3
000000100	バンク4
000000101	バンク5
000000110	バンク6
000000111	バンク7
000001000	バンク8
000001001	バンク9
000001010	バンク10
000001011	バンク11
000001100	バンク12
000001101	バンク13
000001110	バンク14

EN	レジスタバンク内エントリ
00000	R0
00001	R1
00010	R2
00011	R3
00100	R4
00101	R5
00110	R6
00111	R7
01000	R8
01001	R9
01010	R10
01011	R11
01100	R12
01101	R13
01110	R14
01111	MACH
10000	割り込みベクタオフセット
10001	PR
10010	GBR
10011	MACL

(2) 注意

アーキテクチャ上では、最大 512 個のバンクを持つことができます。
ただし、バンクの数は製品により異なります。

(3) 動作内容

```
LDBANK ( long m) /*LDBANK @Rm, R0 */
{
R[0]=Read_Bank_Long( R[m] );
PC+=2;
}
```

(4) 使用例

```
LDBANK@R1, R0                ;実行前  R1=H'00000108
                             ;実行後  R0=バンク 2 に退避した R2 の内容
```

6.3.18 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　　SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
LDC Rm, TBR	Rm→TBR	0100mmmm01001010	1	—

(1) 説明

ソースオペランドをコントロールレジスタ TBR に格納します。

(2) 動作内容

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

(3) 使用例

```
LDC R0, TBR                    ;実行前  R0=H'12345678, TBR=H'00000000
                               ;実行後  TBR=H'12345678
```

6.3.19 MOV MOVe structure data データ転送命令

構造体データの転送 SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
MOV.B Rm, @(disp12,Rn)	Rm→(disp+Rn)	0011nnnnmmmm0001 0000dddddddddddd	1	—
MOV.W Rm, @(disp12,Rn)	Rm→(disp×2+Rn)	0011nnnnmmmm0001 0001dddddddddddd	1	—
MOV.L Rm, @(disp12,Rn)	Rm→(disp×4+Rn)	0011nnnnmmmm0001 0010dddddddddddd	1	—
MOV.B @(disp12,Rm), Rn	(disp+Rm)→符号拡張→Rn	0011nnnnmmmm0001 0100dddddddddddd	1	—
MOV.W @(disp12,Rm), Rn	(disp×2+Rm)→符号拡張→Rn	0011nnnnmmmm0001 0101dddddddddddd	1	—
MOV.L @(disp12,Rm), Rn	(disp×4+Rm)→Rn	0011nnnnmmmm0001 0110dddddddddddd	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。

(2) 注意

ルネサスの「SuperH RISC engine アセンブラ」では、ディスプレースメントの値としてスケーリング（×1、×2、×4）された値を記述してください。

(3) 動作内容

```
MOVBS12 (long d, long m, long n)  /* MOV.B Rm, @(disp12,Rn) */
{
   long disp;

   disp = (0x00000FFF & (long)d);
   Write_Byte(R[n]+disp,R[m]);
   PC+=4;
}

MOVWS12 (long d, long m, long n)  /* MOV.W Rm, @(disp12,Rn) */
{
   long disp;
```

```
   disp = (0x00000FFF & (long)d);
   Write_Word(R[n]+(disp<<1),R[m]);
   PC+=4;
}



MOVLS12 (long d, long m, long n)  /* MOV.L Rm, @(disp12,Rn) */
{
   long disp;

   disp = (0x00000FFF & (long)d);
   Write_Long(R[n]+(disp<<2), R[m]);
   PC+=4;
}

MOVBL12 (long d, long m, long n)  /* MOV.B @(disp12,Rm), Rn */
{
   long disp;

   disp = (0x00000FFF & (long)d);
   R[n]=Read_Byte(R[m]+disp);
   if ( ( R[n]&0x80 ) ==0) R[n] &=0x000000FF;
   else R[0] |=0xFFFFFF00;
   PC+=4;
}

MOVWL12 (long d, long m, long n)  /* MOV.W @(disp12,Rm), Rn */
{
   long disp;

   disp = (0x00000FFF & (long)d);
   R[n]=Read_Word(R[m]+(disp<<1));
   if ( ( R[n]&0x8000 ) ==0) R[n] &=0x0000FFFF;
   else R[n] |=0xFFFF0000;
    PC+=4;
}
```

```
MOVLL12 (long d, long m, long n)  /* MOV.L @(disp12,Rm), Rn */
{
   long disp;

   disp = (0x00000FFF & (long)d);
   R[n]=Read_Long(R[m]+(disp<<2));
   PC+=4;
}
```

(4) 使用例

```
MOV.B R0, @(1, R1)            ;実行前  R0=H'FFFF7F80
                              ;実行後  @(R1+1)=H'80
MOV.L @(2*, R0), R1           ;実行前  @(R0+8)=H'12345670
                              ;実行後  R1=H'12345670
```

【注】*「6.3.19（2）注意」を参照してください。

6.3.20 MOV MOVe reverse stack データ転送命令

逆スタック転送　　SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
MOV.B R0, @Rn+	R0→(Rn), Rn+1→Rn	0100nnnn10001011	1	—
MOV.W R0, @Rn+	R0→(Rn), Rn+2→Rn	0100nnnn10011011	1	—
MOV.L R0, @Rn+	R0→(Rn), Rn+4→Rn	0100nnnn10101011	1	—
MOV.B @-Rm, R0	Rm-1→Rm (Rm)→符号拡張→R0	0100mmmm11001011	1	—
MOV.W @-Rm, R0	Rm-2→Rm (Rm)→符号拡張→R0	0100mmmm11011011	1	—
MOV.L @-Rm, R0	Rm-4→Rm (Rm)→R0	0100mmmm11101011	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。

(2) 動作内容

```
MOVRSBP (long n)  /* MOV.B R0, @Rn+*/
{
  Write_Byte(R[n], R[0]);
  R[n]+=1;
  PC+=2;
}

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

MOVRSLP (long n)  /* MOV.L R0, @Rn+*/
{
  Write_Long(R[n], R[0]);
  R[n]+=4;
  PC+=2;
```

```
}
MOVRSBM (long m)   /* MOV.B @-Rm, R0*/
{
R[m]－=1;
   R[0]=(long) Read_Word (R[m]);
   if ( (R[0]&0x80)==0)  R[0]&=0x000000FF;
   else R[0] |=0xFFFFFF00;

   PC+=2;
}

MOVRSWM (long m)   /* MOV.W @-Rm, R0*/
{
R[m]－=2;
   R[0]=(long) Read_Word (R[m]);
   if ( (R[0]&0x8000)==0)  R[0]&=0x0000FFFF;
   else R[0] |=0xFFFF0000;

   PC+=2;
}

MOVRSLM(long m)   /* MOV.L @-Rm, R0*/
{
R[m]－=4;
R[0]=Read_Long (R[m]);

    PC+=2;
}
```

(3) 使用例

```
MOV.B R0, @R1+              ;実行前  R0=H'AAAAAAAA, R1=H'FFFF7F80
                            ;実行後  R1=H'FFFF7F81, @(H'FFFF7F80)=H'AA
MOV.L @-R1, R0              ;実行前  R1=H'12345678
                            ;実行後  R1=H'12345674, R0=@(H'12345674)
```

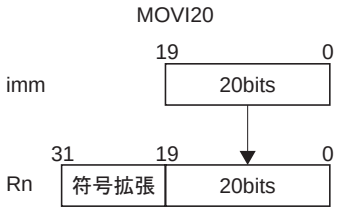
6.3.21　MOVI20　　MOVe　Immediate 20bits data　　データ転送命令

20 ビットイミディエイトデータの転送　　SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
MOVI20　#imm20, Rn	imm→符号拡張→Rn	0000nnnniiii0000 iiiiiiiiiiiiiiii	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。



(2)　動作内容

```
MOVI20 ( long i, long n)   /* MOVI20 #imm, Rn */
 {
       if ( i&0x00080000 ) ==0) R[n]= (0x000FFFFF & (long) i );
       else R[n]=(0xFFF00000 | (long) i );

    PC+=4;
   }
```

(3)　使用例

```
MOVI20 H'7FFFF, R0          ;実行前  R0=H'00000000
                            ;実行後  R0=H'0007FFFF
```

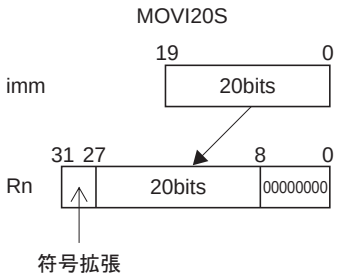
6.3.22 MOVI20S MOVe Immediate 20bits data and 8bits Shift left データ転送命令

20 ビットイミディエイトデータの転送・8 ビット左シフト SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
MOVI20S #imm20, Rn	imm<<8 →符号拡張→Rn	0000nnnniiii0001 iiiiiiiiiiiiiiii	1	—

(1) 説明

イミディエイトデータを左に 8 ビットシフトし、ロングワードに符号拡張後、汎用レジスタ Rn に格納します。後続命令として ADD 命令もしくは OR 命令を用いることで、28 ビットの絶対アドレスを生成することが可能です。詳細は**「付録 B. プログラミングの指針（MOVI20、MOVI20S の使用方法について）」**を参照してください。



(2) 注意

ルネサスの「SuperH RISC engine アセンブラ」では、イミディエイトデータとして 8 ビット左シフトされた値を記述してください。

(3) 動作内容

```
MOVI20S (long i, long n)   /* MOVI20S #imm, Rn */
 {
   if ( i&0x00080000 ) ==0) R[n]= (0x000FFFFF & (long) i );
   else R[n]=(0xFFF00000 | (long) i );
   R[n]<<=8;
   PC+=4;
 }
```

(4) 使用例

```
MOVI20S H'7FFFF00, R0          ;実行前  R0=H'00000000
                               ;実行後  R0=H'07FFFF00
```

6.3.23 MOVML.L MOVe Multi-register Lower part データ転送命令

R0～Rn のレジスタ退避・回復命令　　SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
MOVML.L Rm, @-R15	R15-4→R15, Rm→(R15) R15-4→R15, Rm-1→(R15) ・ ・ R15-4→R15, R0→(R15) ※ Rm=R15 のとき、 Rm を PR に読み替え	0100mmmm11110001	1～16	—
MOVML.L @R15+, Rn	(R15)→R0, R15+4→R15 (R15)→R1, R15+4→R15 ・ ・ (R15)→Rn, R15+4→R15 ※ Rn=R15 のとき、 Rn を PR に読み替え	0100nnnn11110101	1～16	—

(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。指定したレジスタ番号以下の複数の汎用レジスタ（R0～Rn/Rm）と、R15 の内容をアドレスとするメモリとの転送を行います。

R15 を指定した場合、R15 の代わりに PR を転送します。つまり、nnnn(mmmm)=1111 を指定したときは、R0～R14, PR が転送対象の汎用レジスタとなります。

(2) 動作内容

```
MOVLMML (long m) /*MOVML.L Rm, @-R15*/
{
   long i;

  for (i=m; i≧0; i－－)
   {
     if (i==15)
      {
        Write_Long (R[15]－4, PR);
        R[15]－＝4;
      }
    else
```

```
      {
        Write_Long (R[15]－4, R[i]);
        R[15]－＝4;
       }
     }

   PC+=2;
   }

  MOVLPML (long n )  /*MOVML.L @R15+, Rn */
  {
    int  i;

    for ( i=0; i≦n; i++ )
     {
      if (i==15)
        {
          PR=Read_Long (R[15]);
         }
      else
        {
        R[i] = Read_Long (R[15]);
         }
       R[15]+=4;
     }
    PC+=2;
  }
```

(3) 使用例

```
MOVML.L R7, @-R15              ;実行前  R15=H'FFFF7F80
                                        R0=H'00000000, R1=H'11111111
                                        R2=H'22222222, R3=H'33333333
                                        R4=H'44444444, R5=H'55555555
                                        R6=H'66666666, R7=H'77777777
                                ;実行後  R15=H'FFFF7F60
                                       @(H'FFFF7F7C)=H'77777777
                                       @(H'FFFF7F78)=H'66666666
```

```
                                @(H'FFFF7F74)=H'55555555
                                @(H'FFFF7F70)=H'44444444
                                @(H'FFFF7F6C)=H'33333333
                                @(H'FFFF7F68)=H'22222222
                                @(H'FFFF7F64)=H'11111111
                                @(H'FFFF7F60)=H'00000000
MOVML.L @R15, R7                ;実行前  R15=H'FFFF7F60
                                @(H'FFFF7F60)=H'00000000
                                @(H'FFFF7F64)=H'11111111
                                @(H'FFFF7F68)=H'22222222
                                @(H'FFFF7F6C)=H'33333333
                                @(H'FFFF7F70)=H'44444444
                                @(H'FFFF7F74)=H'55555555
                                @(H'FFFF7F78)=H'66666666
                                @(H'FFFF7F7C)=H'77777777
                                ;実行後  R15=H'FFFF7F80
                                 R0=H'00000000, R1=H'11111111
                                 R2=H'22222222, R3=H'33333333
                                 R4=H'44444444, R5=H'55555555
                                 R6=H'66666666, R7=H'77777777
```

6.3.24 MOVMU.L MOVe Multi-register Upper part データ転送命令

Rn～R14, PR のレジスタ退避・回復命令 SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
MOVMU.L Rm, @-R15	R15-4→R15, PR→(R15) R15-4→R15, R14→(R15) ・ ・ R15-4→R15, Rm→(R15) ※ Rm=R15 のとき、 Rm を PR に読み替え	0100mmmm11110000	1～16	—
MOVMU.L @R15+, Rn	(R15)→Rn,R15+4→R15 (R15)→Rn+1,R15+4→R15 ・ (R15)→R14, R15+4→R15 (R15)→PR, R15+4→R15 ※ Rn=R15 のとき、 Rn を PR に読み替え	0100nnnn11110100	1～16	—

(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。指定したレジスタ番号以上の複数の汎用レジスタ（Rn/Rm～R14, PR）と、R15 の内容をアドレスとするメモリとの転送を行います。

R15 を指定した場合、R15 の代わりに PR を転送します。

(2) 動作内容

```
 MOVLMMU (long m) /*MOVMU.L Rm, @－R15 */
{
  int i;

  Write_Long (R[15]－4, PR);
  R[15]－＝4;

  for ( i = 14; i≧m; i－－)
   {
   Write_Long (R[15]－4, R[i]);
   R[15]－＝4;
   }
PC+=2;
}


MOVLPMU (long n)  /*MOVMU.L @R15+, Rn*/
  {
    int i;

    for ( i=n; i≦14; i++ )
      {
       R[i] = Read_Long (R[15]);
       R[15]+=4;
      }
    PR=Read_Long (R[15]);
    R[15]+=4;
    PC+=2;
  }
```

(3) 使用例

```
MOVMU.L R8, @-R15             ;実行前  R15=H'FFFF7F80
                                       R8=H'88888888, R9=H'999999999
                                       R10=H'AAAAAAAA, R11=H'BBBBBBBB
                                       R12=H'CCCCCCCC, R13=H'DDDDDDDD
                                       R14=H'EEEEEEEE, PR=H'FFFFFFF0
```

```
                                ;実行後  R15=H'FFFF7F60
                                        @(H'FFFF7F7C)=H'FFFFFFF0
                                        @(H'FFFF7F78)=H'EEEEEEEE
                                        @(H'FFFF7F74)=H'DDDDDDDD
                                        @(H'FFFF7F70)=H'CCCCCCCC
                                        @(H'FFFF7F6C)=H'BBBBBBBB
                                        @(H'FFFF7F68)=H'AAAAAAAA
                                        @(H'FFFF7F64)=H'99999999
                                        @(H'FFFF7F60)=H'88888888

MOVMU.L @R15, R8                ;実行前  R15=H'FFFF7F60
                                        @(H'FFFF7F60)=H'88888888
                                        @(H'FFFF7F64)=H'99999999
                                        @(H'FFFF7F68)=H'AAAAAAAA
                                        @(H'FFFF7F6C)=H'BBBBBBBB
                                        @(H'FFFF7F70)=H'CCCCCCCC
                                        @(H'FFFF7F74)=H'DDDDDDDD
                                        @(H'FFFF7F78)=H'EEEEEEEE
                                        @(H'FFFF7F7C)=H'FFFFFFF0
                                ;実行後  R15=H'FFFF7F80
                                        R8=H'88888888, R9=H'99999999
                                        R10=H'AAAAAAAA, R11=H'BBBBBBBB
                                        R12=H'CCCCCCCC, R13=H'DDDDDDDD
                                        R14=H'EEEEEEEE, PR=H'FFFFFFF0
```

6.3.25　MOVRT　MOVe Reverse Tbit　　データ転送命令

T ビット反転 Rn 転送　　SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
MOVRT　Rn	~T→Rn	0000nnnn00111001	1	—

(1)　説明

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

(2)　動作内容

```
MOVRT ( long n )  /*MOVRT Rn */
{
 if (T ==1) R[n]=0x00000000;
 else R[n] = 0x00000001;
PC+=2;
}
```

(3)　使用例

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

6.3.26 MOVU MOVe structure data as Unsigned データ転送命令

構造体データの無符号転送 SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
MOVU.B @(disp12,Rm), Rn	(disp+Rm)→ゼロ拡張→Rn	0011nnnnmmmm0001 1000dddddddddddd	1	—
MOVU.W @(disp12,Rm), Rn	(disp×2+Rm)→ゼロ拡張→Rn	0011nnnnmmmm0001 1001dddddddddddd	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。本命令は、符号なしのデータの転送を行います。構造体、スタック内のデータアクセスに最適です。

(2) 注意

ルネサスの「SuperH RISC engine アセンブラ」では、ディスプレースメントの値としてスケーリング（×1、×2）された値を記述してください。

(3) 動作内容

```
MOVBUL12 (long d, long m, long n)  /* MOVU.B @(disp12,Rm), Rn */
{
   long disp;

   disp = (0x00000FFF & (long)d);
   R[n]=Read_Byte(R[m]+disp);
   R[n] &=0x000000FF;
   PC+=4;
}

MOVWUL12 (long d, long m, long n)  /* MOVU.W @(disp12,Rm), Rn */
{
   long disp;

   disp = (0x00000FFF & (long)d);
   R[n]=Read_Word(R[m]+(disp<<1));
   R[n] &=0x0000FFFF;
   PC+=4;
}
```

(4) 使用例

```
MOVU.B@(2, R0), R1          ;実行前  @(R0+2)=H'FF
                            ;実行後  R1=H'000000FF
MOVU.W@(2, R0), R1          ;実行前  @(R0+4)=H'FFFF
                            ;実行後  R1=H'0000FFFF
```

6.3.27 MULR MULtiply to Register 算術演算命令

Rn 結果格納符号付き乗算 SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
MULR R0,Rn	R0 × Rn→Rn	0100nnnn10000000	2	—

(1) 説明

汎用レジスタ R0 の内容と Rn を 32 ビットで乗算し、結果の下位側 32 ビットを汎用レジスタ Rn に格納します。

(2) 動作内容

```
MULR ( long n)  /* MULR  R0, Rn */
{
   R[n] = R[0]*R[n];
   PC+=2;
}
```

(3) 使用例

```
MULR R0, R1                ;実行前  R0=H'FFFFFFFE, R1=H'00005555
                           ;実行後  R1=H'FFFF5556
```

6.3.28　NOTT　　NOT　Tbit　　データ転送命令

T ビット反転転送　　SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
NOTT	~T→T	0000000001101000	1	演算結果

(1)　説明

T ビットを反転した後、T ビットに格納します。

(2)　動作内容

```
NOTT ( long n )  /*NOTT Rn */
 {
  if (T ==1) T=0;
  else T=1;
 PC+=2;
 }
```

(3)　使用例

```
SETT                    ;T=1
NOTT                    ;T=0
NOTT                    ;T=1
```

6.3.29 PREF PREFetch data to cache データ転送命令

データキャッシュへのプリフェッチ SH4

書式	動作概略	命令コード	実行ステート	T ビット
PREF @Rn	prefetch cache block	0000nnnn10000011	1	—

(1) 説明

16 バイト境界で始まる 16 バイトのデータブロックをオペランドキャッシュに読み込みます。

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

(2) 注意

キャッシュ非搭載の製品では、NOP 命令として取り扱われます。

(3) 動作内容

```
PREF ( long n ) /* PREF @Rn */
 {
 PC+=2;
 }
```

(4) 使用例

```
            MOV.L SOFT_PF, R1  ;R1 のアドレスは SOFT_PF
            PREF @R1           ; SOFT_PF のデータを内蔵データキャッシュへロード

            .align  16
SOFT_PF:    .data.w H'1234
            .data.w H'5678
            .data.w H'9ABC
            .data.w H'DEF0
```

6.3.30　RESBANK　REStore from registerBANK　　システム制御命令

レジスタバンクからのレジスタ復帰　　SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
RESBANK	レジスタバンクからの復帰	0000000001011011	9*	—

【注】 *　　バンクオーバフローが起きており、レジスタをスタックから復帰するときは 19

(1)　説明

最後に退避したレジスタバンクからレジスタを復帰します。

(2)　動作内容

```
RESBANK( )   /*RESBANK */
             /*m=(最後に退避を行ったレジスタバンクの番号)*/
{
 int  m;

if(BO==0)
{
    PR = Register_Bank[m].PR_BANK;
    GBR = Register_Bank[m].GBR_BANK;
    MACL = Register_Bank[m].MACL_BANK;
    MACH = Register_Bank[m].MACH_BANK;
    for (i=14; i≧0; i--)
    {
     R[i] = Register_Bank[m].R_BANK[i];
    }
 }
else
 {
    for (i=0; i≦14; i++)
    {
     R[i] = Read_Long(R[15]);
     R[15]+=4;
    }
    PR=Read_Long(R[15]);
```

```
        R[15]+=4;
        GBR=Read_Long(R[15]);
        R[15]+=4;
        MACH=Read_Long(R[15]);
        R[15]+=4;
        MACL =Read_Long(R[15]);
        R[15]+=4;
    }

    PC+=2;

    }
```

(3) 使用例

```
RESBANK                         ;レジスタバンクからレジスタを復帰します。
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。
```

6.3.31 RTS/N ReTurn from Subroutine with No delay slot 分岐命令

遅延スロットなしサブルーチンプロシージャからの復帰　SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
RTS/N	PR→PC	0000000001101011	3	—

(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

本命令は遅延分岐ではありません。

(3) 動作内容

```
RTSN (  )   /* RTS/N  */
{
   PC=PR+4;
}
```

(4) 使用例

```
             MOV.L TABLE, R3        ;R0=TRGET のアドレス
             JSR/N @R3              ;TRGET へ分岐します。
             ADD    R0, R1          ;←プロシージャからの戻り先
                                    （PR の内容）です。
             . . . . . . . . .
TABLE:       .data.1 TRGET          ;ジャンプテーブル
             . . . . . . . . .
TRGET:       NOP                    ;←プロシージャの入口
             MOV   R2, R3           ;
             RTS/N                  ;上記 ADD 命令に戻ります。
```

6.3.32 RTV/N ReTurn to Value and from subroutine with No delay slot 分岐命令

レジスタ値転送付き遅延スロットなしサブルーチンプロシージャからの復帰

SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
RTV/N Rm	Rm→R0, PR→PC	0000mmmm01111011	3	—

(1) 説明

指定した汎用レジスタ Rm から R0 へ転送後、サブルーチンプロシージャから復帰します。すなわち、Rm の値を R0 に格納後、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

本命令は遅延分岐ではありません。

(3) 動作内容

```
RTVN ( int m)   /* RTV/N  Rm */
{
   R[0]=R[m];
   PC=PR+4;
 }
```

(4) 使用例

```
             MOV.L TABLE, R3        ;R0=TRGET のアドレス
             JSR/N @R3              ;TRGET へ分岐します。
             ADD    R0, R1          ;←プロシージャからの戻り先
                                    （PR の内容）です。
             . . . . . . . . .
TABLE:       .data.1 TRGET          ;ジャンプテーブル
             . . . . . . . . .
TRGET:       NOP                    ;←プロシージャの入口
             MOV   #12, R3          ;R3=H'00000012
             RTV/N R3               ;上記 ADD 命令に戻ります。
                                    ;R0=H'00000012
```

6.3.33 SHAD SHift Arithemetic Dynamically シフト命令

ダイナミック算術シフト

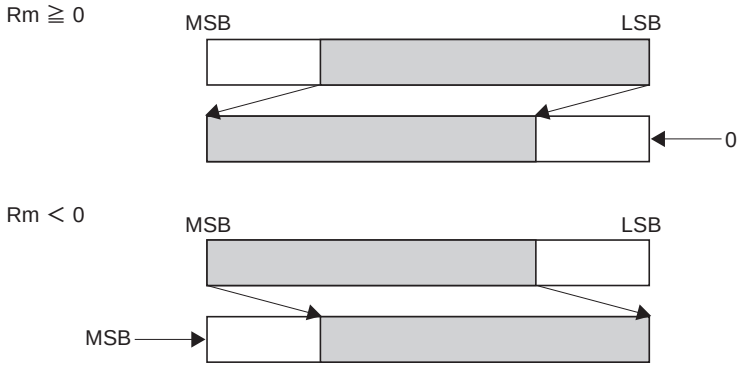
書式	動作概略	命令コード	実行ステート	T ビット
SHAD Rm, Rn	Rm≧0 のとき、 Rn<<Rm→Rn Rm<0 のとき、 Rn>>|Rm|→[MSB→Rn]	0100nnnnmmmm1100	1	—

(1) 説明

汎用レジスタ Rn の内容を算術的にシフトします。汎用レジスタ Rm がシフトの方向とシフトするビット数を指定します。

Rm レジスタの値が正のとき左へシフトし、負のとき右へシフトします。右にシフトするときは上位に MSB が追加されます。

シフトするビット数は Rm レジスタの下位 5 ビット(ビット 4～0)で指定されます。値が負(MSB=1)のときは Rm レジスタは 2 の補数で表されています。したがって、右シフトのシフト量は、Rm レジスタの下位 5 ビットの反転に 1 を加えた数になります。左シフトのシフト量は 0～31 で、右シフトのシフト量は 1～32 です。



(2) 動作内容

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

(3) 使用例

```
SHAD R1, R2                          ;実行前  R1=H'FFFFFFEC, R2=H'80180000
                                     ;実行後  R1=H'FFFFFFEC, R2=H'FFFFF801
SHAD R3, R4                          ;実行前  R3=H'00000014, R4=H'FFFFF801
                                     ;実行後  R3=H'00000014, R4=H'80100000
```

6.3.34 SHLD SHift Logical Dynamically シフト命令

ダイナミック論理シフト

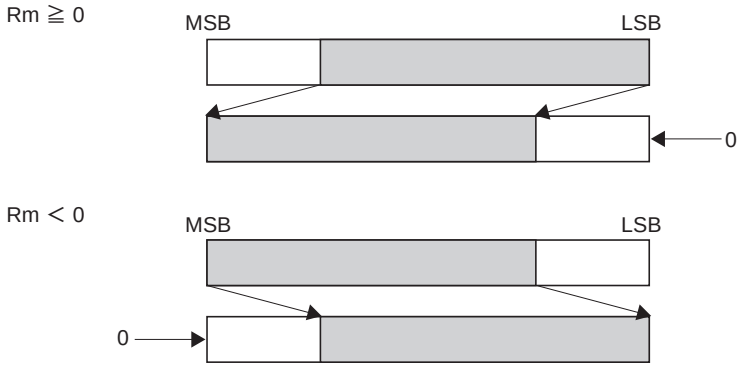
書式	動作概略	命令コード	実行ステート	T ビット
SHLD Rm, Rn	Rm≧0 のとき、 Rn<<Rm→Rn Rm<0 のとき、 Rn>>|Rm|→[0→Rn]	0100nnnnmmmm1101	1	—

(1) 説明

汎用レジスタ Rn の内容を論理的にシフトします。汎用レジスタ Rm がシフトの方向とシフトするビット数を指定します。

Rm レジスタの値が正のとき左へシフトし、負のとき右へシフトします。右にシフトするときは上位に 0 が追加されます。

シフトするビット数は Rm レジスタの下位 5 ビット(ビット 4～0)で指定されます。値が負(MSB=1)のときは Rm レジスタは 2 の補数で表されています。したがって、右シフトのシフト量は、Rm レジスタの下位 5 ビットの反転に 1 を加えた数になります。左シフトのシフト量は 0～31 で、右シフトのシフト量は 1～32 です。



(2) 動作内容

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

(3) 使用例

```
SHLD R1, R2                          ;実行前  R1=H'FFFFFFEC, R2=H'80180000
                                     ;実行後  R1=H'FFFFFFEC, R2=H'00000801
SHLD R3, R4                          ;実行前  R3=H'00000014, R4=H'FFFFF801
                                     ;実行後  R3=H'00000014, R4=H'80100000
```

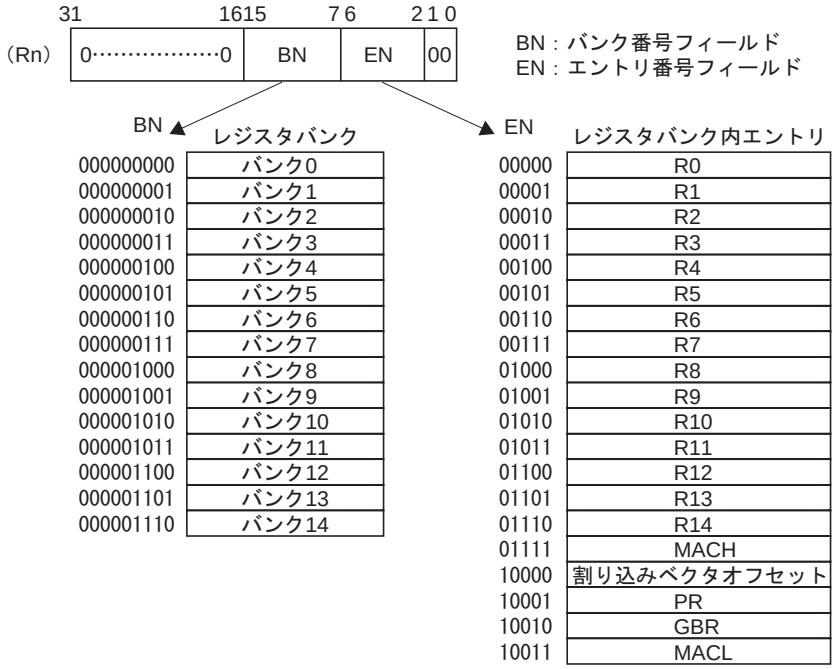
6.3.35 STBANK STore register BANK システム制御命令

指定バンクエントリへのレジスタ退避　　SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
STBANK R0, @Rn	R0 →（指定レジスタバンクエントリ）	0100nnnn11100001	7	—

(1) 説明

R0 を汎用レジスタ Rn の内容が指すレジスタバンク中の 1 エントリに転送します。レジスタバンクの番号とバンク内に格納されているレジスタを、汎用レジスタ Rn で指定します。



BN	レジスタバンク
000000000	バンク0
000000001	バンク1
000000010	バンク2
000000011	バンク3
000000100	バンク4
000000101	バンク5
000000110	バンク6
000000111	バンク7
000001000	バンク8
000001001	バンク9
000001010	バンク10
000001011	バンク11
000001100	バンク12
000001101	バンク13
000001110	バンク14

EN	レジスタバンク内エントリ
00000	R0
00001	R1
00010	R2
00011	R3
00100	R4
00101	R5
00110	R6
00111	R7
01000	R8
01001	R9
01010	R10
01011	R11
01100	R12
01101	R13
01110	R14
01111	MACH
10000	割り込みベクタオフセット
10001	PR
10010	GBR
10011	MACL

(2) 注意

アーキテクチャ上では、最大 512 個のバンクを持つことができます。
ただし、バンクの数は製品により異なります。

(3) 動作内容

```
STBANK ( long n)  /*STBANK  R0, @Rn */
  {
   Write_Bank_Long (R[n], R[0])
 PC+=2;
}
```

(4) 使用例

```
STBANK R0,@R1                    ;実行前  R1=H'00000108, R0=H'FFFFFFFF
                                 ;実行後  バンク 2 に退避した R2=H'FFFFFFFF
```

6.3.36 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　　SH-2A/SH2A-FPU（新規）

書式	動作概略	命令コード	実行ステート	T ビット
STC TBR, Rn	TBR→Rn	0000nnnn01001010	1	—

(1) 説明

コントロールレジスタ TBR のデータをディスティネーションに格納します。

(2) 動作内容

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

(3) 使用例

```
STC TBR, R0                    ;実行前  R0=H'12345678, TBR=H'00000000
                               ;実行後  R0=H'00000000
```

6.4 SH-2E の CPU 命令

6.4.1 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
```

6.4.2 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
```

6.4.3 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
```

6.4.4　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
```

6.4.5 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 4 番地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);
    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
```

6.4.6 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 4 番地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);
        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。

6.4.7 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	1010dddddddddddd	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 4 番地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);
    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。

6.4.8 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 4 番地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=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。

6.4.9 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	1011ddddddddddddd	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 4 番地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);
    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。

6.4.10 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 4 番地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=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。

6.4.11 BT Branch if True：分岐命令

条件分岐

書式	動作概略	命令コード	実行ステート	T ビット
BT label	T=1 のとき disp×2+PC→PC, T=0 のとき nop	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 4 番地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);
    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
```

6.4.12 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 4 番地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);
        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。

6.4.13　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+                ;
```

6.4.14 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
```

6.4.15 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>>24)&0x000000FF;
    HL=(temp>>16)&0x000000FF;
    LH=(temp>>8)&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。
```

6.4.16 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 命令などを組み合わせて、繰り返し除算を行い商を求めます。詳しくは DIV1 の説明を参照してください。

(2) 動作内容

```
DIV0S(long m, long n)   /* DIV0S Rm,Rn */
{
    if ((R[n] & 0x80000000)==0) Q=0;
    else Q=1;
    if ((R[m] & 0x80000000)==0) M=0;
    else M=1;
    T=!(M==Q);
    PC+=2;
}
```

(3) 使用例

DIV1 の使用例を参照してください。

6.4.17 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 命令などを組み合わせて、繰り返し除算を行い商を求めます。詳しくは DIV1 の説明を参照してください。

(2) 動作内容

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

(3) 使用例

DIV1 の使用例を参照してください。

6.4.18 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わせて繰り返し実行し商を求めます。この繰り返し中は、指定したレジスタと M、Q、T ビットを書き換えないでください。

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)
```

6.4.19 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	—

(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
```

6.4.20 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	—

(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
```

6.4.21 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 R5                   ;R5 の値をデクリメントし、0 になったかどうか判定します。
    BF LOOP                 ;T=0 なら LOOP へ分岐します(この例では 4 回ループします)。
```

6.4.22 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
```

6.4.23 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
```

6.4.24 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。

6.4.25 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 4 番地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。

6.4.26　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

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	3	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	5	LSB
LDC.L	@Rm+,GBR	(Rm)→GBR, Rm+4→Rm	0100mmmm00010111	1	—
LDC.L	@Rm+,VBR	(Rm)→VBR, Rm+4→Rm	0100mmmm00100111	1	—

(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)　動作内容

```
LDCSR(long m)    /* LDC Rm,SR */
{
    SR=R[m]&0x000063F3;
    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])&0x000063F3;
    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;
}
```

(3) 使用例

```
LDC    R0,SR                  ;実行前  R0=H'FFFFFFFF,SR=H'00000000
                              ;実行後  SR=H'000063F3
LDC.L  @R15+,GBR              ;実行前 R15=H'10000000
                              ;実行後 R15=H'10000004,GBR=@H'10000000
```

6.4.27　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

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)　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;
}
```

(3) 使用例

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

6.4.28 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	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(fnLmL<0){
   Res2=~Res2;
   if (Res0==0) Res2++;
   else Res0=(~Res0)+1;
}
if(S==1){
   Res0=MACL+Res0;
   if (MACL>Res0) Res2++;
   if (MACH&0x00008000);
          Res2+=MACH|0xFFFF0000;
   else   Res2+=MACH&0x00007FFF;
```

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

        MACH=(Res2&0x0000FFFF)|(MACH&0xFFFF0000);
        MACL=Res0;
    }
    else {
        Res0=MACL+Res0;
        if (MACL>Res0) Res2++;
        Res2+=MACH;

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

(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             ;
```

6.4.29 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	—

(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 レジスタを H'00000001 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) {
           MACH=0x00000001;
           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   ;
```

6.4.30 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
```

6.4.31 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 4 番地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 4 番地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。

ルネサスの「SuperH RISC engine アセンブラ」では、ディスプレースメントの値としてスケーリング（×2、×4）された値を記述してください。

(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  @(4,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 ;
```

6.4.32 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

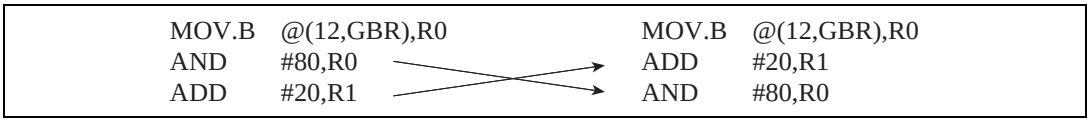
ソースオペランドをデスティネーションへ転送します。内蔵周辺モジュール領域内のデータアクセスに最適です。データサイズをバイト、ワード、またはロングワードの範囲で指定できますが、レジスタが R0 固定になります。GBR には、内蔵周辺モジュールのベースアドレスを設定します。

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。



ルネサスの「SuperH RISC engine アセンブラ」では、ディスプレースメントの値としてスケーリング（×1、×2、×4）された値を記述してください。

(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
```

6.4.33 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
```

ルネサスの「SuperH RISC engine アセンブラ」では、ディスプレースメントの値としてスケーリング（×1、×2、×4）された値を記述してください。

(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
```

6.4.34 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 4 番地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。

ルネサスの「SuperH RISC engine アセンブラ」では、ディスプレースメントの値としてスケーリング（×4）された値を記述してください。

(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                  ;
```

6.4.35 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
```

6.4.36 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	—

(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
```

6.4.37 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	—

(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
```

6.4.38 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	—

(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
```

6.4.39 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
```

6.4.40 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
```

6.4.41 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。
```

6.4.42 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
```

6.4.43 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
```

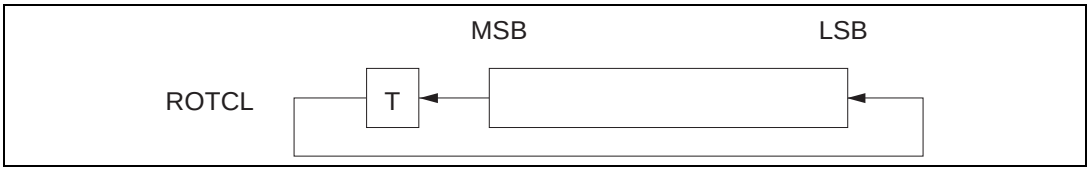
6.4.44 ROTCL ROTate with Carry Left：シフト命令

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
```

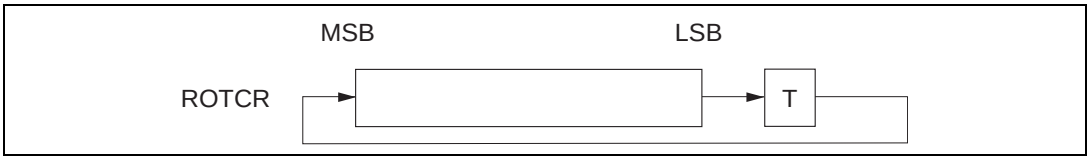
6.4.45 ROTCR ROTate with Carry Right：シフト命令

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
```

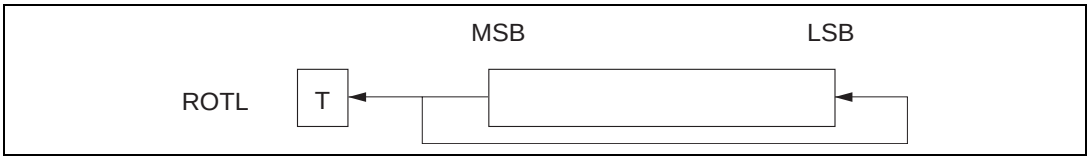
6.4.46　ROTL　ROTate Left：シフト命令

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
```

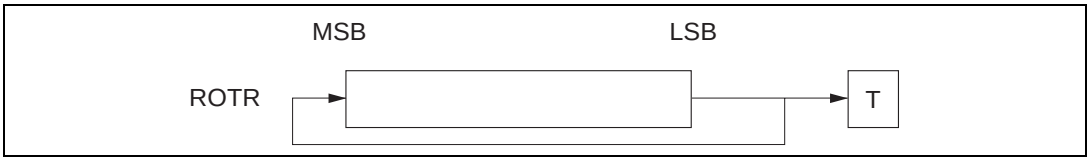
6.4.47 ROTR ROTate Right：シフト命令

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
```

6.4.48 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	6	—

(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])&0x000063F3;
    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。

6.4.49 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。

6.4.50 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
```

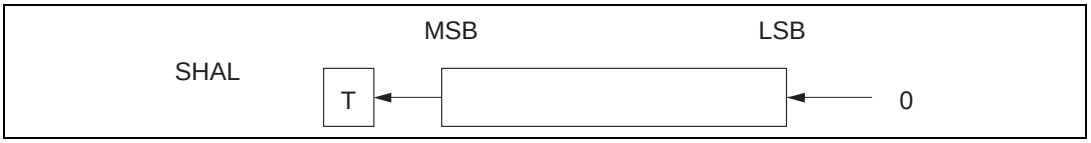
6.4.51 SHAL SHift Arithmetic Left：シフト命令

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
```

6.4.52　SHAR　SHift Arithmetic 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
```

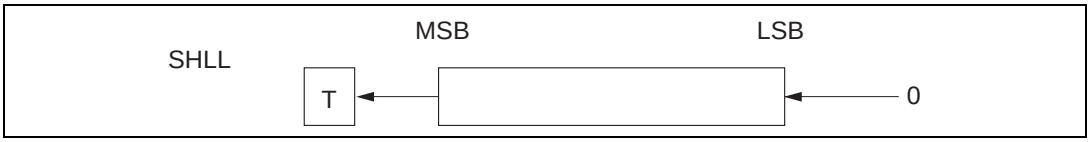
6.4.53 SHLL SHift Logical Left：シフト命令

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
```

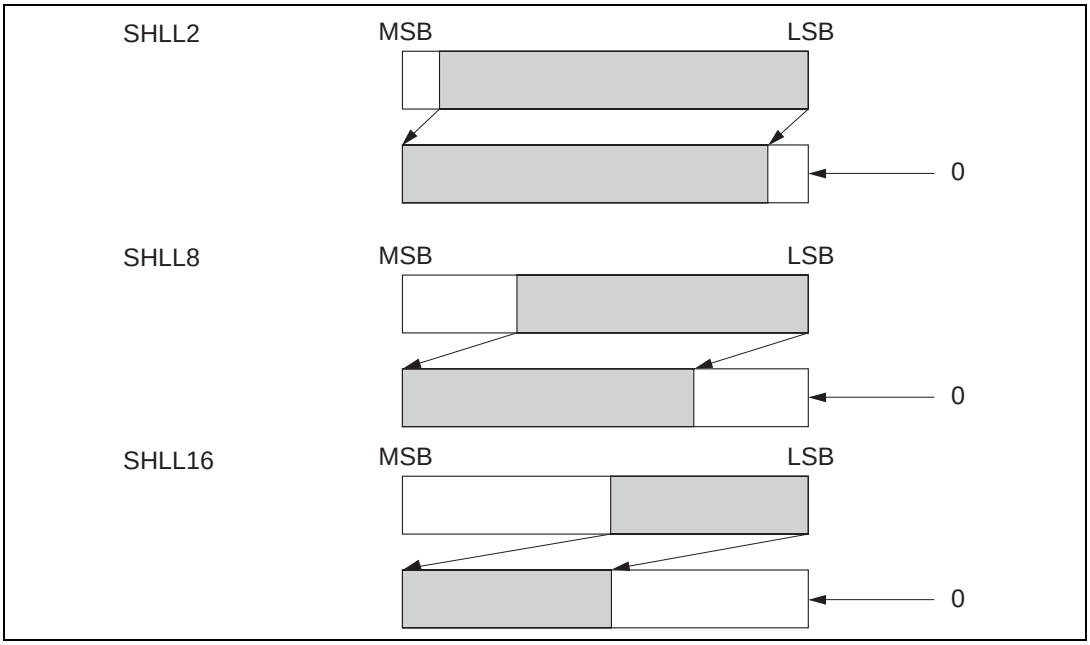
6.4.54　SHLLn　n bits SHift Logical Left：シフト命令

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
```

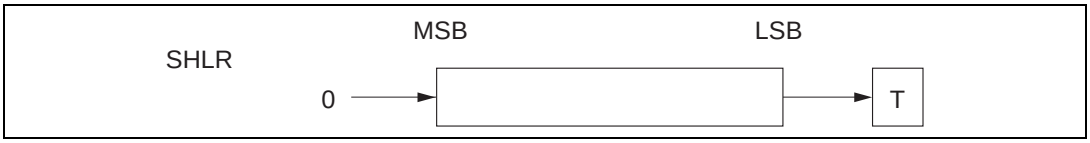
6.4.55　SHLR　SHift Logical Right：シフト命令

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
```

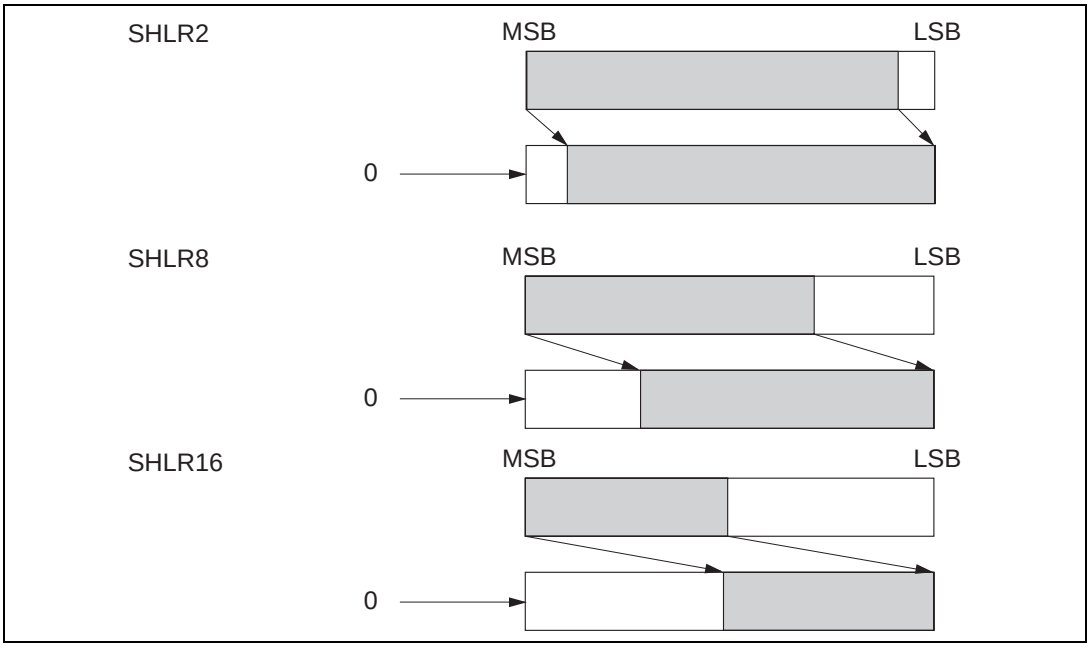
6.4.56 SHLRn n bits SHift Logical Right：シフト命令

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
```

6.4.57 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	5	—

(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 5 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 */
{
   wait_for_exception;
}
```

(4) 使用例

```
SLEEP              ;低消費電力状態への遷移
```

6.4.58　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

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	2	—
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	1	—
STC.L	VBR,@-Rn	Rn-4→Rn, VBR→(Rn)	0100nnnn00100011	1	—

(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)　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;
}
```

(3) 使用例

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

6.4.59　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

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) 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;
}
```

(3) 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
```

6.4.60 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
```

6.4.61 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
```

6.4.62 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
```

6.4.63 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
```

6.4.64 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	3	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。
```

6.4.65 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	5	—

(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 と組み合わせて、システムコールに使います。

(2) 注意

ルネサスの「SuperH RISC engine アセンブラ」では、ディスプレースメントの値としてスケーリング（×4）された値を記述してください。

(3) 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;
}
```

(4) 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。
```

6.4.66　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
```

6.4.67 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
```

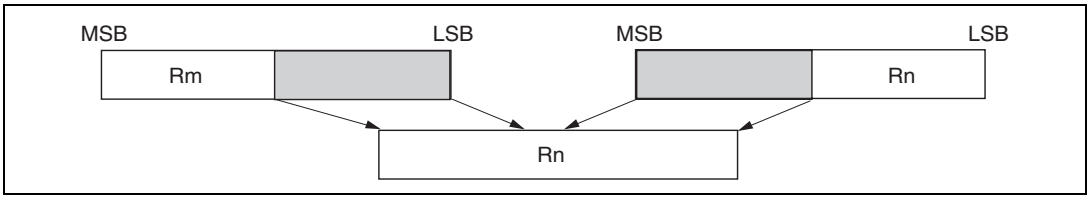
6.4.68 XTRCT eXTRaCT：データ転送命令

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
```

6.5 浮動小数点命令と FPU に関する CPU 命令

6.5.1 FABS Floating - point ABSolute value 浮動小数点命令

浮動小数点絶対値

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FABS FRn	|FRn|→FRn	1111nnnn01011101	1	—
1	FABS DRn	|DRn|→DRn	1111nnn001011101	1	—

(1) 説明

浮動小数点レジスタ FRn/DRn の内容の最上位ビットを 0 にクリアして、結果を FRn/DRn に格納します。

FPSCR の cause/flag 部分は更新されません。

(2) 動作内容

```
void FABS (int n){
    FR[n] = FR[n] & 0x7fffffff;
    pc += 2;
}
/* 精度に依存せず、同じ動作を行います。 */
```

(3) 発生する可能性がある例外

なし

6.5.2　FADD　　Floating - point ADD　　浮動小数点命令

浮動小数点加算

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FADD　FRm,FRn	FRn＋FRm→FRn	1111nnnnmmmm0000	1	—
1	FADD　DRm,DRn	DRn＋DRm→DRn	1111nnn0mmm00000	6	—

(1)　説明

FPSCR.PR=0 の場合：FRn と FRm の内容の２つの単精度浮動小数点数を算術加算し、結果を FRn に格納します。

FPSCR.PR=1 の場合：DRn と DRm の内容の２つの倍精度浮動小数点数を算術加算し、結果を DRn に格納します。

FPSCR.enable.O/U/I がセットされている場合、FPU 例外トラップが、例外の発生如何にかかわらず発生します。例外発生時は、FPSCR.cause　FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

(2)　動作内容

```
void FADD (int m,n)
{
      pc += 2;
      clear_cause();
      if((data_type_of(m) == sNaN) ||
         (data_type_of(n) == sNaN)) invalid(n);
      else if((data_type_of(m) == qNaN) ||
            (data_type_of(n) == qNaN)) qnan(n);
      else if((data_type_of(m) == DENORM) ||
            (data_type_of(n) == DENORM)) set_E();
      else switch (data_type_of(m)){
         case NORM: switch (data_type_of(n)){
            case NORM:     normal_faddsub(m,n,ADD); break;
            case PZERO:
            case NZERO:register_copy(m,n); break;
            default:       break;
         }         break;
         case PZERO: switch (data_type_of(n)){
            case NZERO:zero(n,0); break;
            default:       break;
         }         break;
```

```
            case NZERO:break;
            case PINF: switch (data_type_of(n)){
               case NINF:     invalid(n);     break;
               default:       inf(n,0);       break;
            }          break;
            case NINF: switch (data_type_of(n)){
               case PINF:     invalid(n);     break;
               default:       inf(n,1);       break;
            }          break;
         }
   }
```

FADD 特殊ケース

FRm,DRm	FRn,DRn						
	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 0 として扱われます。

(3) 発生する可能性がある例外

無効演算（Invalid operation）

オーバフロー（Overflow）

アンダフロー（Underflow）

不正確例外（Inexact）

6.5.3 FCMP Floating - point CoMPare 浮動小数点命令

浮動小数点比較

No.	PR	書式	動作概略	命令コード	実行ステート	Tビット
1	0	FCMP/EQ FRm,FRn	(FRn==FRm)?1:0→T	1111nnnnmmmm0100	1	1/0
2	1	FCMP/EQ DRm,DRn	(DRn==DRm)?1:0→T	1111nnn0mmm00100	2	1/0
3	0	FCMP/GT FRm,FRn	(FRn>FRm)?1:0→T	1111nnnnmmmm0101	1	1/0
4	1	FCMP/GT DRm,DRn	(DRn>DRm)?1:0→T	1111nnn0mmm00101	2	1/0

(1) 説明

1. FPSCR.PR=0の場合 : FRnとFRmの内容の2つの単精度浮動小数点数を算術比較し、等しい場合にTビットに1を、他の場合に0を格納します。
2. FPSCR.PR=1の場合 : DRnとDRmの内容の2つの倍精度浮動小数点数を算術比較し、等しい場合にTビットに1を、他の場合に0を格納します。
3. FPSCR.PR=0の場合 : FRnとFRmの内容の2つの単精度浮動小数点数を算術比較し、FRn>FRmの場合にTビットに1を、他の場合に0を格納します。
4. FPSCR.PR=1の場合 : DRnとDRmの内容の2つの倍精度浮動小数点数を算術比較し、DRn>DRmの場合にTビットに1を、他の場合に0を格納します。

(2) 動作内容

```
void FCMP_EQ(int m,n) /* FCMP/EQ  FRm,FRn */
{
    pc += 2;
    clear_cause();
    if(fcmp_chk (m,n) == INVALID) fcmp_invalid();
    else if(fcmp_chk (m,n) == EQ)     T = 1;
    else                              T = 0;
}
void FCMP_GT(int m,n) /* FCMP/GT  FRm,FRn */
{
    pc += 2;
    clear_cause();
    if ((fcmp_chk (m,n) == INVALID) ||
        (fcmp_chk (m,n) == UO)) fcmp_invalid();
    else if(fcmp_chk (m,n) == GT)  T = 1;
    else                       T = 0;
}
int fcmp_chk (int m,n)
{
```

```
    if((data_type_of(m) == sNaN) ||
       (data_type_of(n) == sNaN))       return(INVALID);
    else if((data_type_of(m) == qNaN) ||
            (data_type_of(n) == qNaN))     return(UO);
    else switch(data_type_of(m)){
         case NORM:    switch(data_type_of(n)){
              case PINF   :return(GT); break;
              case NINF   :return(LT); break;
              default:                 break;
              }     break;
         case PZERO:
         case NZERO:   switch(data_type_of(n)){
              case PZERO  :
              case NZERO  :return(EQ); break;
              default:                 break;
              }     break;
         case PINF :   switch(data_type_of(n)){
              case PINF   :return(EQ); break;
              default:return(LT);      break;
              }     break;
         case NINF :   switch(data_type_of(n)){
              case NINF   :return(EQ); break;
              default:return(GT);      break;
              }     break;
    }
    if(FPSCR_PR == 0) {
            if(FR[n] == FR[m])         return(EQ);
            else if(FR[n] > FR[m])     return(GT);
            else                       return(LT);
    }else {
            if(DR[n>>1] == DR[m>>1])     return(EQ);
            else if(DR[n>>1] > DR[m>>1]) return(GT);
            else                         return(LT);
    }
}
void fcmp_invalid()
{
```

```
            set_V(); T = 0; if((FPSCR & ENABLE_V) == 1) fpu_exception_trap();
}
```

FCMP 特殊ケース

FCMP/EQ	FRn,DRn						
FRm,DRm	NORM	+0	–0	+INF	–INF	qNaN	sNaN
NORM	CMP						
+0		EQ					
–0							
+INF				EQ			
–INF					EQ		
qNaN						!EQ	
sNaN							Invalid

【注】 非正規化数の値は 0 として扱われます。

FCMP/GT	FRn,DRn						
FRm,DRm	NORM	+0	–0	+INF	–INF	qNaN	sNaN
NORM	CMP			GT	!GT		
+0		!GT					
–0							
+INF	!GT			!GT			
–INF	GT				!GT		
qNaN						UO	
sNaN							Invalid

【注】 非正規化数の値は 0 として扱われます。
UO はアンオーダードです。アンオーダードは!GT と扱います。

(3) 発生する可能性がある例外

無効演算（Invalid operation）

6.5.4 FCNVDS Floating - point CoNVert Double to Single precision 浮動小数点命令

倍精度単精度変換

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	—	—	—	—	—
1	FCNVDS DRm,FPUL	(float)DRm →FPUL	1111mmm010111101	2	—

(1) 説明

FPSCR.PR=1 の場合、DRm 内の倍精度浮動小数点数を単精度浮動小数点数に変換し FPUL に格納します。

FPSCR.enable.O/U/I がセットされている場合、FPU 例外トラップが、例外の発生如何にかかわらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FPUL は更新されません。ソフトウェアで適切な処理を行ってください。

FPSCR.PR=0 の場合、不当命令となります。

(2) 動作内容

```
void FCNVDS(int m, float *FPUL){
    case((FPSCR.PR){
        0:  undefined_operation();   /* reserved */
        1:  fcnvds(m, *FPUL);  break;  /* FCNVDS */
    }
}
void fcnvds(int m, float *FPUL)
{
    pc += 2;
    clear_cause();
    case(data_type_of(m)){
        NORM  :
        PZERO :
        NZERO :    normal_ fcnvds(m, *FPUL);  break;
        PINF  :    *FPUL = 0x7f800000; break;
        NINF  :    *FPUL = 0xff800000; break;
        qNaN  :    *FPUL = 0x7fbfffff; break;
        sNaN  :     set_V();
                      if((FPSCR & ENABLE_V) == 0) *FPUL = 0x7fbfffff;
                      else fpu_exception_trap();   break;
    }
```

```
}
void normal_fcnvds(int m,float *FPUL)
{
int sign;
float abs;
union {
      float f;
      int l;
}     dstf,tmpf;
union {
      double d;
      int l[2];
}     dstd;
      dstd.d = DR[m>>1];
      if(dstd.l[1] & 0x1fffffff)) set_I();
      if(FPSCR_RM == 1) dstd.l[1] &= 0xe0000000; /* round toward zero */
      dstf.f = dstd.d;
      check_single_exception(FPUL, dstf.f);
}
```

FCNVDS 特殊ケース

FRn	+NORM	–NORM	+0	–0	+INF	–INF	qNaN	sNaN
FCNVDS(FRn FPUL)	FCNVDS	FCNVDS	+0	–0	+INF	–INF	qNaN	Invalid

【注】 非正規化数の値は 0 として扱われます。

(3) 発生する可能性がある例外

無効演算（Invalid operation）

オーバフロー（Overflow）

アンダフロー（Underflow）

不正確例外（Inexact）

6.5.5 FCNVSD Floating - point CoNVert Single to Double precision 浮動小数点命令

単精度倍精度変換

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	—	—	—	—	—
1	FCNVSD FPUL, DRn	(double)FPUL→DRn	1111nnn010101101	2	—

(1) 説明

FPSCR.PR=1 の場合、FPUL の内容を単精度浮動小数点数と解釈し、倍精度浮動小数点数に変換し、結果を DRn に格納します。

FPSCR.PR=0 の場合、不当命令となります。

(2) 動作内容

```
void FCNVSD(int n, float *FPUL){
    pc += 2;
    clear_cause();
    case((FPSCR_PR){
        0:  undefined_operation();   /* reserved */
        1:  fcnvsd (n, *FPUL);  break;  /* FCNVSD */
    }
}
void fcnvsd(int n, float *FPUL)
{
    case(fpul_type(*FPUL)){
        PZERO :
        NZERO :
        PINF  :
        NINF  :       DR[n>>1] = *FPUL; break;
        qNaN  :       qnan(n);     break;
        sNaN  :       invalid(n);     break;
    }
}
int fpul_type(int *FPUL)
{
int abs;
      abs = *FPUL & 0x7fffffff;
```

```
        if(abs < 0x00800000){
           if((FPSCR_DN == 1) || (abs == 0x00000000)){
              if(sign_of(src) == 0)   return(PZERO);
              else                    return(NZERO);
           }
           else                       return(DENORM);
        }
        else if(abs < 0x7f800000)  return(NORM);
        else if(abs == 0x7f800000) {
              if(sign_of(src) == 0)   return(PINF);
              else                    return(NINF);
        }
        else if(abs < 0x7fc00000)  return(qNaN);
        else                       return(sNaN);
   }
```

FCNVSD 特殊ケース

FRn	+NORM	–NORM	+0	–0	+INF	–INF	qNaN	sNaN
FCNVSD(FPUL FRn)	+NORM	–NORM	+0	–0	+INF	–INF	qNaN	Invalid

【注】 非正規化数の値は 0 として扱われます。

(3) 発生する可能性がある例外

無効演算（Invalid operation）

6.5.6 FDIV Floating - point DIVide 浮動小数点命令

浮動小数点除算

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FDIV FRm,FRn	FRn/FRm→FRn	1111nnnnmmmm0011	10	—
1	FDIV DRm,DRn	DRn/DRm→DRn	1111nnn0mmm00011	23	—

(1) 説明

FPSCR.PR=0 の場合：FRn と FRm の内容の 2 つの単精度浮動小数点数を算術除算し、結果を FRn に格納します。

FPSCR.PR=1 の場合：DRn と DRm の内容の 2 つの倍精度浮動小数点数を算術除算し、結果を DRn に格納します。

FPSCR.enable.O/U/I がセットされている場合、FPU 例外トラップが、例外の発生如何にかかわらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

(2) 動作内容

```
void FDIV(int m,n)    /* FDIV FRm,FRn */
{
    pc += 2;
    clear_cause();
    if((data_type_of(m) == sNaN) ||
      (data_type_of(n) == sNaN)) invalid(n);
    else if((data_type_of(m) == qNaN) ||
          (data_type_of(n) == qNaN)) qnan(n);
    else switch (data_type_of(m)){
        case NORM: switch (data_type_of(n)){
            case PINF:
            case NINF:     inf(n,sign_of(m)^sign_of(n));break;
            case PZERO:
            case NZERO:zero(n,sign_of(m)^sign_of(n));break;
            case DENORM:   set_E();      break;
            default:       normal_fdiv(m,n);  break;
           }     break;
        case PZERO: switch (data_type_of(n)){
            case PZERO:
            case NZERO:invalid(n);break;
            case PINF:
```

```
                case NINF:     break;
                default:       dz(n,sign_of(m)^sign_of(n));break;
                          }    break;
           case NZERO: switch (data_type_of(n)){
                case PZERO:
                case NZERO:invalid(n);break;
                case PINF:     inf(n,1);break;
                case NINF:     inf(n,0);break;
                default:       dz(FR[n],sign_of(m)^sign_of(n)); break;
               }     break;
           case PINF :
           case NINF : switch (data_type_of(n)){
                case PINF:
                case NINF:  invalid(n);    break;
                default:    zero(n,sign_of(m)^sign_of(n));break
               }     break;
           }
}
void normal_fdiv(int m,n)
{
union {
      float f;
      int l;
}     dstf,tmpf;
union {
      double d;
      int l[2];
}     dstd,tmpd;
union {
      int double x;
      int l[4];
}     tmpx;
      if(FPSCR_PR == 0) {
           tmpf.f = FR[n]; /* save destination value */
           dstf.f /= FR[m]; /* round toward nearest or even */
           tmpd.d = dstf.f; /* convert single to double */
           tmpd.d *= FR[m];
```

```
            if(tmpf.f != tmpd.d) set_I();
            if((tmpf.f < tmpd.d) && (SPSCR_RM == 1))
                dstf.l -= 1; /* round toward zero */
            check_single_exception(&FR[n], dstf.f);
        } else {
            tmpd.d = DR[n>>1]; /* save destination value */
            dstd.d /= DR[m>>1]; /* round toward nearest or even */
            tmpx.x = dstd.d; /* convert double to int double */
            tmpx.x *= DR[m>>1];
            if(tmpd.d != tmpx.x) set_I();
            if((tmpd.d < tmpx.x) && (SPSCR_RM == 1)) {
                dstd.l[1] -= 1; /* round toward zero */
                if(dstd.l[1] == 0xffffffff) dstd.l[0] -= 1;
            }
            check_double_exception(&DR[n>>1], dstd.d);
        }
    }
```

FDIV 特殊ケース

FRm,DRm	FRn,DRn						
	NORM	+0	–0	+INF	–INF	qNaN	sNaN
NORM	DIV	0		INF			
+0	DZ	Invalid		+INF	-INF		
–0				-INF	+INF		
+INF	0	+0	–0	Invalid			
–INF		–0	+0				
qNaN						qNaN	
sNaN							Invalid

【注】 非正規化数の値は 0 として扱われます。

(3) 発生する可能性がある例外

無効演算（Invalid operation）

ゼロ割り（Divide by zero）

オーバフロー（Overflow）

アンダフロー（Underflow）

不正確例外（Inexact）

6.5.7 FLDI0 Floating - point LoaD Immediate 0.0

浮動小数点命令

0.0 ロード

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FLDI0 FRn	0x00000000→FRn	1111nnnn10001101	1	—
1	—	—	—	—	—

(1) 説明

FPSCR.PR=0 の場合、浮動小数点数の 0.0（0x00000000)を FRn に格納します。
FPSCR.PR=1 の場合、不当命令となります。

(2) 動作内容

```
void FLDI0(int n)
{
    FR[n] = 0x00000000;
    pc += 2;
}
```

(3) 発生する可能性がある例外

なし

6.5.8 FLDI1 Floating - point LoaD Immediate 1.0

浮動小数点命令

1.0 ロード

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FLDI1 FRn	0x3F800000→FRn	1111nnnn10011101	1	—
1	—	—	—	—	—

(1) 説明

FPSCR.PR=0 の場合、浮動小数点数の 1.0（0x3F800000)を FRn に格納します。
FPSCR.PR=1 の場合、不当命令となります。

(2) 動作内容

```
void FLDI1(int n)
{
    FR[n] = 0x3F800000;
    pc += 2;
}
```

(3) 発生する可能性がある例外

なし

6.5.9 FLDS Floating - point LoaD to System register 浮動小数点命令

システムレジスタへの転送

書式	動作概略	命令コード	実行ステート	T ビット
FLDS FRm,FPUL	FRm → FPUL	1111mmmm00011101	1	—

(1) 説明

浮動小数点レジスタ FRm の内容をシステムレジスタである FPUL に格納します。

(2) 動作内容

```
void FLDS(int m,float *FPUL)
{
    *FPUL = FR[m];
    pc += 2;
}
```

(3) 発生する可能性がある例外

なし

6.5.10 FLOAT Floating - point convert from integer
浮動小数点命令

整数浮動小数点数変換

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FLOAT FPUL,FRn	(float)FPUL→FRn	1111nnnn00101101	1	—
1	FLOAT FPUL,DRn	(double)FPUL→DRn	1111nnn000101101	2	—

(1) 説明

FPSCR.PR=0 の場合 : FPUL の内容を 32 ビット整数とみなして、単精度浮動小数点数に変換し、 結果を FRn に格納します。

FPSCR.PR=1 の場合: FPUL の内容を 32 ビット整数とみなして、倍精度浮動小数点数に変換し、結果を DRn に格納します。

FPSCR.enable.I=1 と、FPCSR.PR=0 の場合、FPU 例外トラップが、例外の発生如何にかかわらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

(2) 動作内容

```
void FLOAT(int n,float *FPUL)
{
union {
      double d;
      int l[2];
}     tmp;
      pc += 2;
      clear_cause();
      if(FPSCR.PR==0){
         FR[n] = *FPUL; /* convert from integer to float */
         tmp.d = *FPUL;
         if(tmp.l[1] & 0x1fffffff) inexact();
      } else {
         DR[n>>1] = *FPUL; /* convert from integer to double */
      }
}
```

(3) 発生する可能性がある例外

不正確例外（Inexact）：FPSCR.PR = 1 の場合、発生しません。

6.5.11 FMAC　Floating - point Multiply and Accumulate　浮動小数点命令

浮動小数点積和

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FMAC FR0,FRm,FRn	FR0*FRm＋FRn→FRn	1111nnnnmmmm1110	1	—
1	—	—	—	—	—

(1) 説明

FPSCR.PR=0 の場合、FR0 と FRm の内容の２つの単精度浮動小数点数を算術乗算し、さらに、FRn の内容を算術加算し、結果を FRn に格納します。

FPSCR.enable.O/U/I がセットされている場合、FPU 例外トラップが、例外の発生如何にかかわらず発生します。例外発生時は、FPSCR.cause　FPSCR.flag には、例外の正しい情報が反映され、FRn は更新されません。ソフトウェアで適切な処理を行ってください。

FPSCR.PR=1 の場合、不当命令となります。

(2) 動作内容

```
void FMAC(int m,n)
{
      pc += 2;
      clear_cause();
      if(FPSCR_PR == 1) undefined_operation();
      else if((data_type_of(0) == sNaN) ||
              (data_type_of(m) == sNaN) ||
              (data_type_of(n) == sNaN)) invalid(n);
      else if((data_type_of(0) == qNaN) ||
              (data_type_of(m) == qNaN)) qnan(n);
      else if((data_type_of(0) == DENORM) ||
              (data_type_of(m) == DENORM)) set_E();
      else switch (data_type_of(0){
              case NORM: switch (data_type_of(m)){
              case PZERO:
              case NZERO: switch (data_type_of(n)){
                  case qNaN:      qnan(n);          break;
                  case PZERO:
                  case NZERO:zero(n,(sign_of(0)^ sign_of(m))& sign_of(n));  break;
                  default:       break;
                  }
```

```
        case PINF:
case NINF: switch (data_type_of(n)){
     case qNaN: qnan(n);    break;
     case PINF:
     case NINF: if(sign_of(0)^ sign_of(m)^sign_of(n))invalid(n);
                else inf(n,sign_of(0)^ sign_of(m));    break;
     default:   inf(n,sign_of(0)^ sign_of(m));         break;
     }
case NORM: switch (data_type_of(n)){
     case qNaN: qnan(n);    break;
     case PINF:
     case NINF:     inf(n,sign_of(n));    break;
     case PZERO:
     case NZERO:
     case NORM:     normal_fmac(m,n);     break;
}       break;
case PZERO:
case NZERO: switch (data_type_of(m)){
     case PINF:
     case NINF:     invalid(n);           break;
     case PZERO:
     case NZERO:
     case NORM: switch (data_type_of(n)){
     case qNaN:     qnan(n);              break;
     case PZERO:
     case NZERO:    zero(n,(sign_of(0)^ sign_of(m))& sign_of(n));  break;
     default:       break;
     }           break;
}       break;
case PINF :
case NINF : switch (data_type_of(m)){
     case PZERO:
     case NZERO:    invalid(n);           break;
     default: switch (data_type_of(n)){
     case qNaN: gnan(n);                  break;
     case PINF:
     case NINF: if(sign_of(0)^ sign_of(m)^ sign_of(n))invalid(n);
```

```
                    else inf(n,sign_of(0)^ sign_of(m));   break;
            default:  inf(n,sign_of(0)^ sign_of(m));break;
        }     break;
    }
}
void normal_fmac(int m,n)
{
union {
      int double x;
      int l[4];
}     dstx,tmpx;
float dstf,srcf;
      if((data_type_of(n) == PZERO)|| (data_type_of(n) == NZERO))
             srcf = 0.0; /* flush denormalized value */
      else   srcf = FR[n];
      tmpx.x = FR[0]; /* convert single to int double */
      tmpx.x *= FR[m]; /* exact product */
      dstx.x = tmpx.x + srcf;
      if(((dstx.x == srcf) && (tmpx.x != 0.0)) ||
          ((dstx.x == tmpx.x) && (srcf != 0.0))) {
          set_I();
          if(sign_of(0)^ sign_of(m)^ sign_of(n))  {
              dstx.l[3] -= 1; /* correct result */
              if(dstx.l[3] == 0xffffffff) dstx.l[2] -= 1;
              if(dstx.l[2] == 0xffffffff) dstx.l[1] -= 1;
              if(dstx.l[1] == 0xffffffff) dstx.l[0] -= 1;
          }
          else   dstx.l[3] |= 1;
      }
      if((dstx.l[1] & 0x01ffffff) || dstx.l[2] || dstx.l[3]) set_I();
      if(FPSCR_RM == 1) {
             dstx.l[1] &= 0xfe000000; /* round toward zero */
             dstx.l[2]  = 0x00000000;
             dstx.l[3]  = 0x00000000;
      }
      dstf = dstx.x;
      check_single_exception(&FR[n],dstf);
```

```
}
```

FMAC 特殊ケース

FRn	FR0	FRm							
		+Norm	-Norm	+0	–0	+INF	–INF	qNaN	sNaN
Norm	Norm	MAC				INF			
	0					Invalid			
	INF	INF		Invalid		INF			
+0	Norm	MAC							
	0				+0	Invalid			
	INF	INF		Invalid		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		Invalid		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					
	Norm								
!sNaN	qNaN							qNaN	
All types	sNaN								
SNaN	all types								Invalid

【注】 非正規化数の値は 0 として扱われます。

(3) 発生する可能性がある例外

無効演算（Invalid operation）

オーバフロー（Overflow）

アンダフロー（Underflow）

不正確例外（Inexact）

6.5.12 FMOV Floating - point MOVe 浮動小数点命令

浮動小数点転送

No.	SZ	書式	動作概略	命令コード	実行ステート	T ビット
1	0	FMOV FRm,FRn	FRm→FRn	1111nnnnmmmm1100	1	—
2	1	FMOV DRm,DRn	DRm→DRn	1111nnn0mmm01100	2	—
3	0	FMOV.S FRm,@Rn	FRm→(Rn)	1111nnnnmmmm1010	1	—
4	1	FMOV.D DRm,@Rn	DRm→(Rn)	1111nnnnmmm01010	2	—
5	0	FMOV.S @Rm,FRn	(Rm)→FRn	1111nnnnmmmm1000	1	—
6	1	FMOV.D @Rm,DRn	(Rm)→DRn	1111nnn0mmmm1000	2	—
7	0	FMOV.S @Rm+,FRn	(Rm)→FRn,Rm+=4	1111nnnnmmmm1001	1	—
8	1	FMOV.D @Rm+,DRn	(Rm)→DRn,Rm+=8	1111nnn0mmmm1001	2	—
9	0	FMOV.S FRm,@-Rn	Rn-=4,FRm→(Rn)	1111nnnnmmmm1011	1	—
10	1	FMOV.D DRm,@-Rn	Rn-=8,DRm→(Rn)	1111nnnnmmm01011	2	—
11	0	FMOV.S @(R0,Rm),FRn	(R0+Rm)→FRn	1111nnnnmmmm0110	1	—
12	1	FMOV.D @(R0,Rm),DRn	(R0+Rm)→DRn	1111nnn0mmmm0110	2	—
13	0	FMOV.S FRm, @(R0,Rn)	FRm→(R0+Rn)	1111nnnnmmmm0111	1	—
14	1	FMOV.D DRm, @(R0,Rn)	DRm→(R0+Rn)	1111nnnnmmm00111	2	—

(1) 説明

1. FRmの内容をFRnに転送します。
2. DRmの内容をDRnに転送します。
3. FRmの内容をRnが示すアドレスのメモリに転送します。
4. DRmの内容をRnが示すアドレスのメモリに転送します。
5. Rmが示すアドレスのメモリの内容をFRnに転送します。
6. Rmが示すアドレスのメモリの内容をDRnに転送します。
7. Rmが示すアドレスのメモリの内容をFRnに転送し、Rmに４を加算します。
8. Rmが示すアドレスのメモリの内容をDRnに転送し、Rmに８を加算します。
9. Rnから４を減算し、 FRmの内容をそのRnが示すアドレスのメモリに転送します。
10. Rnから８を減算し、 DRmの内容をそのRnが示すアドレスのメモリに転送します。
11. (R0+Rm)が示すアドレスのメモリの内容をFRnに転送します。
12. (R0+Rm)が示すアドレスのメモリの内容をDRnに転送します。
13. FRmの内容を(R0+Rn)が示すアドレスのメモリに転送します。
14. DRmの内容を(R0+Rn)が示すアドレスのメモリに転送します。

(2) 動作内容

```
void FMOV(int m,n)              /* FMOV FRm,FRn */
{
    FR[n] = FR[m];
    pc += 2;
}
void FMOV_DR(int m,n)          /* FMOV DRm,DRn */
```

```
{
    DR[n>>1] = DR[m>>1];
    pc += 2;
}
void FMOV_STORE(int m,n)      /* FMOV.S FRm,@Rn */
{
    store_int(FR[m],R[n]);
    pc += 2;
}
void FMOV_STORE_DR(int m,n)   /* FMOV.D DRm,@Rn */
{
    store_quad(DR[m>>1],R[n]);
    pc += 2;
}
 void FMOV_LOAD(int m,n)      /* FMOV.S @Rm,FRn */
{
    load_int(R[m],FR[n]);
    pc += 2;
}
void FMOV_LOAD_DR(int m,n)    /* FMOV.D @Rm,DRn */
{
    load_quad(R[m],DR[n>>1]);
    pc += 2;
}
void FMOV_RESTORE(int m,n)     /* FMOV.S @Rm+,FRn */
{
    load_int(R[m],FR[n]);
    R[m] += 4;
    pc += 2;
}
void FMOV_RESTORE_DR(int m,n) /* FMOV.D @Rm+,DRn */
{
    load_quad(R[m],DR[n>>1]) ;
    R[m] += 8;
    pc += 2;
}
void FMOV_SAVE(int m,n)       /* FMOV.S FRm,@-Rn */
```

```
{
    store_int(FR[m],R[n]-4);
    R[n] -= 4;
    pc += 2;
}
void FMOV_SAVE_DR(int m,n)   /* FMOV.D DRm,@–Rn */
{
    store_quad(DR[m>>1],R[n]-8);
    R[n] -= 8;
    pc += 2;
}
void FMOV_INDEX_LOAD(int m,n)  /* FMOV.S @(R0,Rm),FRn */
{
    load_int(R[0] + R[m],FR[n]);
    pc += 2;
}
void FMOV_INDEX_LOAD_DR(int m,n) /*FMOV.D @(R0,Rm),DRn */
{
    load_quad(R[0] + R[m],DR[n>>1]);
    pc += 2;
}
void FMOV_INDEX_STORE(int m,n)  /*FMOV.S FRm,@(R0,Rn)*/
{
    store_int(FR[m], R[0] + R[n]);
    pc += 2;
}
void FMOV_INDEX_STORE_DR(int m,n)/*FMOV.D DRm,@(R0,Rn)*/
{
    store_quad(DR[m>>1], R[0] + R[n]);
    pc += 2;
}
```

(3) 発生する可能性がある例外

アドレスエラー

6.5.13 FMUL Floating - point MULtiply 浮動小数点命令

浮動小数点乗算

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FMUL FRm,FRn	FRn*FRm→FRn	1111nnnnmmmm0010	1	—
1	FMUL DRm,DRn	DRn*DRm→DRn	1111nnn0mmm00010	6	—

(1) 説明

FPSCR.PR=0 の場合：FRn と FRm の内容の 2 つの単精度浮動小数点数を算術乗算し、結果を FRn に格納します。

FPSCR.PR=1 の場合：DRn と DRm の内容の 2 つの倍精度浮動小数点数を算術乗算し、結果を DRn に格納します。

FPSCR.enable.O/U/I がセットされている場合、FPU 例外トラップが、例外の発生如何にかかわらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

(2) 動作内容

```
void FMUL(int m,n)
{
   pc += 2;
      clear_cause();
      if((data_type_of(m) == sNaN) ||
          (data_type_of(n) == sNaN)) invalid(n);
      else if((data_type_of(m) == qNaN) ||
              (data_type_of(n) == qNaN)) qnan(n);
      else switch (data_type_of(m)){
         case NORM: switch (data_type_of(n)){
              case PZERO:
              case NZERO:zero(n,sign_of(m)^sign_of(n));  break;
              case PINF:
              case NINF      inf(n,sign_of(m)^sign_of(n));  break;
              default:       normal_fmul(m,n);break;
      }      break;
         case PZERO:
         case NZERO: switch (data_type_of(n)){
            case PINF:
            case NINF:        invalid(n);break;
            default:          zero(n,sign_of(m)^sign_of(n));break;
```

```
        }     break;
           case PINF :
           case NINF : switch (data_type_of(n)){
              case PZERO:
              case NZERO:invalid(n);break;
               default:
        inf(n,sign_of(m)^sign_of(n));break
        }     break;
        }
   }
```

FMUL 特殊ケース

FRm,DRm	FRn,DRn						
	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 0 として扱われます。

(3) 発生する可能性がある例外

無効演算（Invalid operation）

オーバフロー（Overflow）

アンダフロー（Underflow）

不正確例外（Inexact）

6.5.14 FNEG Floating - point NEGate value 浮動小数点命令

浮動小数点符号反転

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FNEG FRn	-FRn→FRn	1111nnnn01001101	1	—
1	FNEG DRn	-DRn→DRn	1111nnn001001101	1	—

(1) 説明

浮動小数点レジスタ FRn/DRn の内容の最上位ビット(符号ビット）を反転して、結果を FRn/DRn に格納します。

FPSCR の cause/flag 部分は更新されません。

(2) 動作内容

```
void FNEG (int n){
    FR[n] = -FR[n];
    pc += 2;
}

/* 精度に依存せず、同じ動作を行います。 */
```

(3) 発生する可能性がある例外

なし

6.5.15 FSCHG Sz-bit CHanGe 浮動小数点命令

SZ ビット反転

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FSCHG	FPSCR.SZ=~FPSCR.SZ	1111001111111101	1	—
1	—	—	—	—	—

(1) 説明

FPSCR.PR=0 の場合、浮動小数点状態レジスタ FPSCR の SZ ビットを反転します。FPSCR の SZ ビットを換えると、FMOV 命令のデータ転送が、単精度データ 1 つか、ペアかが切り替わります。FPSCR.SZ=0 のとき、FMOV 命令は単精度データを 1 つ転送します。FPSCR.SZ=1 のとき、FMOV 命令は単精度データをペアで 2 つ転送します。

FPSCR.PR=1 の場合、不当命令となります。

(2) 動作内容

```
void FSCHG()   /* FSCHG */
{
     if(FPSCR_PR == 0){
          FPSCR ^= 0x00100000; /* bit 20 */
          PC += 2;
     }
     else undefined_operation();
}
```

(3) 発生する可能性がある例外

なし

6.5.16 FSQRT Floating - point SQuare RooT 浮動小数点命令

浮動小数点平方根

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FSQRT FRn	√FRn→FRn	1111nnnn01101101	9	—
1	FSQRT DRn	√DRn→DRn	1111nnnn01101101	22	—

(1) 説明

FPSCR.PR=0 の場合 : FRn の内容の単精度浮動小数点数の算術平方根を求め、結果を FRn に格納します。

FPSCR.PR=1 の場合： DRn の内容の倍精度浮動小数点数の算術平方根を求め、結果を DRn に格納します。

FPSCR.enable.I がセットされている場合、FPU 例外トラップが、例外の発生如何にかかわらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

(2) 動作内容

```
void FSQRT(int n){
      pc += 2;
      clear_cause();
      switch(data_type_of(n)){
         case NORM  :     if(sign_of(n) == 0)normal_ fsqrt(n);
                    else       invalid(n);break;
         case PZERO :
         case NZERO :
         case PINF  :    break;
         case NINF  :    invalid(n);  break;
         case qNaN  :    qnan(n);     break;
         case sNaN  :    invalid(n);  break;
      }
}
void normal_fsqrt(int n)
{
union {
      float f;
      int l;
}     dstf,tmpf;
```

```
union {
      double d;
      int l[2];
}     dstd,tmpd;
union {
      int double x;
      int l[4];
}     tmpx;
      if(FPSCR_PR == 0) {
         tmpf.f = FR[n]; /* save destination value */
         dstf.f = sqrt(FR[n]); /* round toward nearest or even */
         tmpd.d = dstf.f; /* convert single to double */
         tmpd.d *= dstf.f;
         if(tmpf.f != tmpd.d) set_I();
         if((tmpf.f < tmpd.d) && (SPSCR_RM == 1))
            dstf.l -= 1; /* round toward zero */
         if(FPSCR & ENABLE_I)fpu_exception_trap();
         else                   FR[n] = dstf.f;
      } else {
         tmpd.d = DR[n>>1]; /* save destination value */
         dstd.d = sqrt(DR[n>>1]); /* round toward nearest or even */
         tmpx.x = dstd.d; /* convert double to int double */
         tmpx.x *= dstd.d;
         if(tmpd.d != tmpx.x) set_I();
         if((tmpd.d < tmpx.x) && (SPSCR_RM == 1)) {
            dstd.l[1] -= 1; /* round toward zero */
            if(dstd.l[1] == 0xffffffff) dstd.l[0] -= 1;
         }
         if(FPSCR & ENABLE_I)fpu_exception_trap();
         else                            DR[n>>1] = dstd.d;
      }
}
```

FSQRT 特殊ケース

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

【注】 非正規化数の値は 0 として扱われます。

(3) 発生する可能性がある例外

無効演算（Invalid operation）

不正確例外（Inexact）

6.5.17 FSTS Floating - point Store System register 浮動小数点命令

システムレジスタからの転送

書式	動作概略	命令コード	実行ステート	T ビット
FSTS FPUL,FRn	FPUL→FRn	1111nnnn00001101	1	—

(1) 説明

システムレジスタ FPUL の内容を浮動小数点レジスタ FRn に転送します。

(2) 動作内容

```
void FSTS(int n, float *FPUL)
{
    FR[n] = *FPUL;
    pc += 2;
}
```

(3) 発生する可能性がある例外

なし

6.5.18 FSUB Floating - point SUBtract 浮動小数点命令

浮動小数点減算

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FSUB FRm,FRn	FRn-FRm→FRn	1111nnnnmmmm0001	1	—
1	FSUB DRm,DRn	DRn-DRm→DRn	1111nnn0mmm00001	6	—

(1) 説明

FPSCR.PR=0 の場合：FRn と FRm の内容の 2 つの単精度浮動小数点数を算術減算し、結果を FRn に格納します。

FPSCR.PR=1 の場合：DRn と DRm の内容の 2 つの倍精度浮動小数点数を算術減算し、結果を DRn に格納します。

FPSCR.enable.O/U/I がセットされている場合、FPU 例外トラップが、例外の発生如何にかかわらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

(2) 動作内容

```
void FSUB (int m,n)
{
     pc += 2;
     clear_cause();
     if((data_type_of(m) == sNaN) ||
        (data_type_of(n) == sNaN)) invalid(n);
     else if((data_type_of(m) == qNaN) ||
             (data_type_of(n) == qNaN)) qnan(n);
     else switch (data_type_of(m)){
        case NORM: switch          pe_of(n)){
            case NORM:     normal_faddsub(m,n,SUB); break;
            case PZERO:
            case NZERO:register_copy(m,n); FR[n] = -FR[n];break;
            default:       break;
          }          break;
          case PZERO:    break;
          case NZERO: switch (data_type_of(n)){
             case NZERO:zero(n,0); break;
             default:      break;
          }          break;
          case PINF: switch (data_type_of(n)){
```

```
            case PINF:     invalid(n);     break;
            default:       inf(n,1);       break;
         }          break;
         case NINF: switch (data_type_of(n)){
            case NINF:     invalid(n);     break;
            default:       inf(n,0);       break;
         }          break;
      }
   }
```

FSUB 特殊ケース

FRm,DRm	FRn,DRn						
	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 0 として扱われます。

(3) 発生する可能性がある例外

無効演算（Invalid operation）

オーバフロー（Overflow）

アンダフロー（Underflow）

不正確例外（Inexact）

6.5.19 FTRC Floating - point Truncate and Convert to integer 浮動小数点命令

整数への変換

PR	書式	動作概略	命令コード	実行ステート	T ビット
0	FTRC FRm,FPUL	(long)FRm→FPUL	1111mmmm00111101	1	—
1	FTRC DRm,FPUL	(long)DRm→FPUL	1111mmm000111101	2	—

(1) 説明

FPSCR.PR=0 の場合：FRm の内容の単精度浮動小数点数を 32 ビット整数に変換し、結果を FPUL に格納します。

FPSCR.PR=1 の場合： DRm の内容の倍精度浮動小数点数を 32 ビット整数に変換し、結果を FPUL に格納します。

丸めモードは常に切り捨てになります。

(2) 動作内容

```
#define N_INT_SINGLE_RANGE 0xcf000000 & 0x7fffffff  /* -1.000000 * 2^31 */
#define P_INT_SINGLE_RANGE 0x4effffff  /* 1.fffffe * 2^30 */
#define N_INT_DOUBLE_RANGE 0xc1e0000000200000 & 0x7fffffffffffffff
#define P_INT_DOUBLE_RANGE 0x41e0000000000000

void FTRC(int m,int *FPUL)
{
    pc += 2;
    clear_cause();
    if(FPSCR.PR==0){
        case(ftrc_single_ type_of(m)){
        NORM:    *FPUL = FR[m];    break;
        PINF:    ftrc_invalid(0, *FPUL); break;
        NINF:    ftrc_invalid(1, *FPUL); break;
        }
    }
    else{               /* case FPSCR.PR=1 */
        case(ftrc_double_type_of(m)){
        NORM:    *FPUL = DR[m>>1]; break;
        PINF:    ftrc_invalid(0, *FPUL); break;
        NINF:    ftrc_invalid(1, *FPUL); break;
        }
    }
```

```
}
int ftrc_single_type_of(int m)
{
       if(sign_of(m) == 0){
            if(FR_HEX[m] > 0x7f800000)     return(NINF);      /* NaN */
            else if(FR_HEX[m] > P_INT_SINGLE_RANGE)
                       return(PINF);       /* out of range,+INF */
            else       return(NORM);       /* +0,+NORM        */
       } else {
            if((FR_HEX[m] & 0x7fffffff) > N_INT_SINGLE_RANGE)
                       return(NINF);       /* out of range ,+INF,NaN*/
            else       return(NORM);       /* -0,-NORM           */
    }
}
int ftrc_double_type_of(int m)
{
       if(sign_of(m) == 0){
            if((FR_HEX[m] > 0x7ff00000) ||
              ((FR_HEX[m] == 0x7ff00000) &&
               (FR_HEX[m+1] != 0x00000000)))  return(NINF);     /* NaN */
            else if(DR_HEX[m>>1] >= P_INT_DOUBLE_RANGE)
                        return(PINF);      /* out of range,+INF */
            else        return(NORM);      /* +0,+NORM        */
       } else   {
            if((DR_HEX[m>>1] & 0x7fffffffffffffff) >= N_INT_DOUBLE_RANGE)
                        return(NINF);      /* out of range ,+INF,NaN*/
            else        return(NORM);      /* -0,-NORM           */
       }
}
void ftrc_invalid(int sign,int *FPUL)
{
    set_V();
     if((FPSCR & ENABLE_V) == 0){
           if(sign == 0)    *FPUL = 0x7fffffff;
           else           *FPUL = 0x80000000;
    }
```

```
        else fpu_exception_trap();
    }
```

FTRC 特殊ケース

FRn,DRn	NORM	+0	–0	Positive Out of Range	Negative Out of Range	+INF	–INF	qNaN	sNaN
FTRC (FRn,DRn)	TRC	0	0	Invalid +MAX	Invalid –MAX	Invalid +MAX	Invalid –MAX	Invalid –MAX	Invalid –MAX

（注） 非正規化数の値は 0 として扱われます。

(3) 発生する可能性がある例外

無効演算（Invalid operation）

6.5.20 LDS LoaD to FPU System register システム制御命令

FPU システム

レジスタへのロード

書式		動作概略	命令コード	実行ステート	T ビット
LDS	Rm,FPUL	Rm→FPUL	0100mmmm01011010	1	—
LDS.L	@Rm+,FPUL	(Rm)→FPUL, Rm＋4→Rm	0100mmmm01010110	1	—
LDS	Rm,FPSCR	Rm→FPSCR	0100mmmm01101010	1	—
LDS.L	@Rm+,FPSCR	(Rm)→FPSCR, Rm＋4→Rm	0100mmmm01100110	1	—

(1) 説明

ソースオペランドを FPU システムレジスタ FPUL、FPSCR に格納します。

(2) 動作内容

```
#define FPSCR_MASK 0x003FFFFF

LDSFPUL(int  m,int *FPUL)     /* LDS Rm,FPUL  */
{
      *FPUL=R[m];
      PC+=2;
}
LDSMFPUL(int  m,int *FPUL)    /* LDS.L @Rm+,FPUL  */
{
      *FPUL=Read_Long(R[m]);
      R[m]+=4;
      PC+=2;
}
LDSFPSCR(int  m)       /* LDS Rm,FPSCR  */
{
      FPSCR=R[m] & FPSCR_MASK;
      PC+=2;
}
LDSMFPSCR(int  m)      /* LDS.L @Rm+,FPSCR  */
{
      FPSCR=Read_Long(R[m]) & FPSCR_MASK;
      R[m]+=4;
```

```
        PC+=2;
    }
```

(3) 発生する可能性がある例外

アドレスエラー

6.5.21 STS Store from FPU System register システム制御命令

FPU システムレジスタからのストア

書式	動作概略	命令コード	実行ステート	T ビット
STS FPUL,Rn	FPUL→Rn	0000nnnn01011010	1	—
STS FPSCR,Rn	FPSCR→Rn	0000nnnn01101010	1	—
STS.L FPUL,@-Rn	Rn-4→Rn, FPUL→(Rn)	0100nnnn01010010	1	—
STS.L FPSCR,@-Rn	Rn-4→Rn, FPSCR→(Rn)	0100nnnn01100010	1	—

(1) 説明

FPU システムレジスタ FPUL、FPSCR をデスティネーションに格納します。

(2) 動作内容

```
STSFPUL(int  n, int *FPUL)       /* STS FPUL,Rn  */
{
      R[n]= *FPUL;
      PC+=2;
}
STSMFPUL(int  n, int *FPUL)      /* STS.L FPUL,@-Rn  */
{
      R[n]-=4;
      Write_Long(R[n],*FPUL) ;
      PC+=2;
}
STSFPSCR(int  n)                 /* STS FPSCR,Rn  */
{
      R[n]=FPSCR&0x003FFFFF;
      PC+=2;
}
STSMFPSCR(int  n)                /* STS.L FPSCR,@-Rn  */
{
      R[n]-=4;
      Write_Long(R[n],FPSCR&0x003FFFFF)
      PC+=2;
}
```

(3) 発生する可能性のある例外

アドレスエラー

(4) 使用例

```
STS
Example 1:
        MOV.L     #H'12ABCDEF, R12
        LDS       R12, FPUL
        STS       FPUL, R13
                                ; After executing the STS instruction:
                                ; R13 = 12ABCDEF
  Example 2:
        STS       FPSCR, R2
                                ; After executing the STS instruction:
                                ; The current content of FPSCR is stored in register R2
STS.L
Example 1:
MOV.L   #H'0C700148, R7
STS.L   FPUL, @-R7
                                ; Before executing the STS.L instruction:
                                ; R7 = 0C700148
                                ; After executing the STS.L instruction:
                                ; R7 = 0C700144, and the content of FPUL is saved at memory
                                ; location 0C700144.
  Example 2:
MOV.L   #H'0C700154, R8
STS.L   FPSCR, @-R8
                                ; After executing the STS.L instruction:
                                ; The content of FPSCR is saved at memory location
                                  0C700150.
```

7. レジスタバンク

7.1 概要

SH-2A/SH2A-FPU は、割り込み処理に伴うレジスタの退避、復帰を高速に行うためのレジスタバンクを内蔵しています。レジスタバンクの構成を図 7.1 に示します。

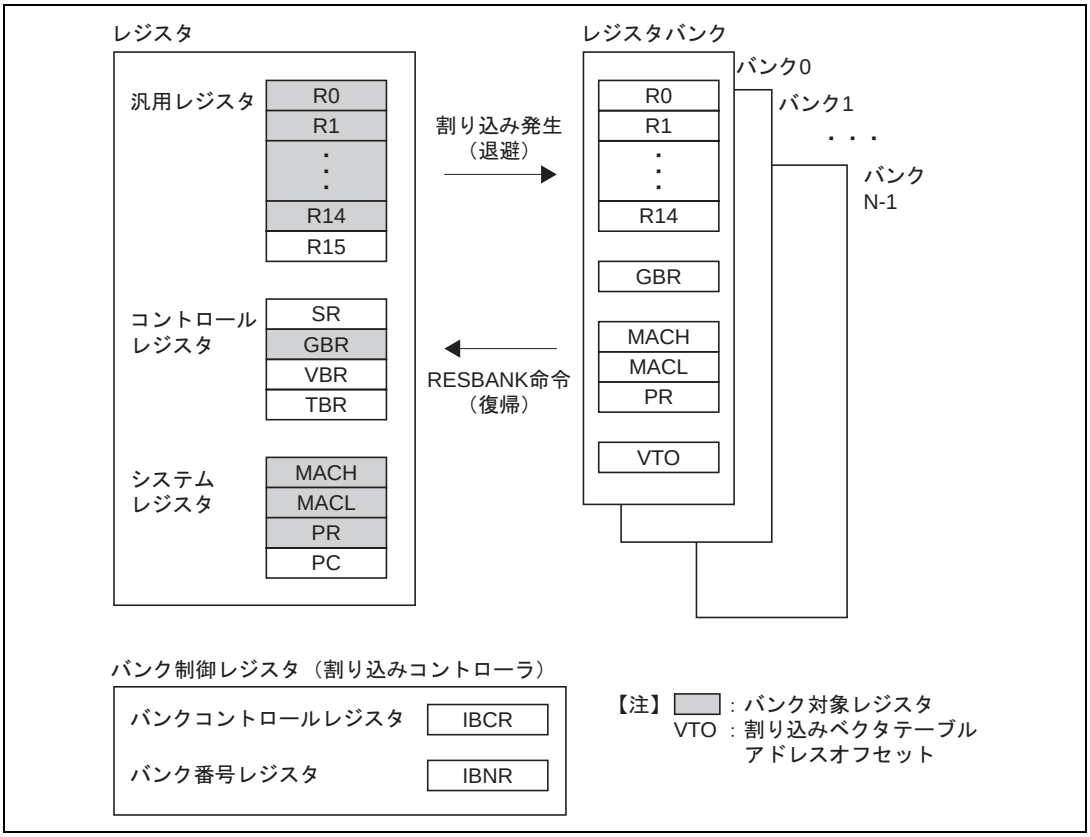


図 7.1 レジスタバンクの構成の概要

7.2 レジスタバンクとバンク制御レジスタ

7.2.1 バンクの対象

汎用レジスタ R0～R14、グローバルベースレジスタ（GBR）、積和レジスタ（MACH、MACL）、プロシージャレジスタ（PR）と、割り込みベクタテーブルアドレスオフセット（VTO）をバンクの対象とします。

7.2.2 レジスタバンク

レジスタバンクは、バンク 0 からバンク N－1 までの N 個のバンクを持ちます（最大 512 バンク）。レジスタバンクは先入れ後出し（FILO）式のスタックになっており、退避はバンク 0 から順番に行い、復帰は最後に退避したバンクから行います。バンクの数 N は、製品により異なります。詳しくは、ハードウェアマニュアルの**「レジスタバンク」**を参照してください。

7.2.3 バンク制御レジスタ

(1) バンクコントロールレジスタ IBCR

割り込みの優先レベル、あるいは割り込み要因に対して、レジスタバンク使用の許可／禁止を設定するレジスタです。レジスタの仕様、初期値は製品により異なります。詳しくは、ハードウェアマニュアルの**「割り込みコントローラ」**を参照してください。

ここでは、割り込みの優先レベルが 1～15 のそれぞれに対して、レジスタバンク使用の許可／禁止を設定する IBCR の例を示します。

IBCR（16 ビット、初期値=H'0000）

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	E15	E14	E13	E12	E11	E10	E9	E8	E7	E6	E5	E4	E3	E2	E1	—

ビット 15～1：E15～E1

割り込みの優先レベル 15～1 に対してレジスタバンクの使用の許可／禁止を設定します。

ビット 15～1 E15～E1	説　　明
0	レジスタバンクの使用を禁止します。（初期値）
1	レジスタバンクの使用を許可します。

ビット 0：予約ビット

読み出すと常に 0 が読み出されます。書き込む値も常に 0 にしてください。

(2)　バンク番号レジスタ　IBNR（16 ビット、初期値=H'0000）

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	BE1	BE0	BOVE	—	—	—	—	—	—	—	—	—	BN3	BN2	BN1	BN0

バンク番号レジスタ（IBNR）はレジスタバンク使用の許可／禁止、およびレジスタバンクオーバフロー例外の許可／禁止を設定します。また、BN3～0 により次に退避されるバンク番号を示します。IBNR はパワーオンリセットで H'0000 に初期化されます。

ビット 15、14：BE1、BE0

レジスタバンクの使用の許可／禁止を設定します。

ビット 15、14 BE1、BE0	説　　明
00	すべての割り込みでバンクの使用を禁止します。IBCR の設定は無視します。（初期値）
01	NMI、UBC 以外のすべての割り込みでバンクの使用を許可します。IBCR の設定は無視されます。
10	予約（設定しないでください）
11	レジスタバンクの使用は、IBCR の設定に従います。

ビット 13：BOVE

レジスタバンクオーバフロー例外の許可／禁止を設定します。

ビット 13 BOVE	説　　明
0	レジスタバンクオーバフロー例外の発生を禁止します。（初期値）
1	レジスタバンクオーバフロー例外の発生を許可します。

ビット 12～4：予約ビット

読み出すと常に 0 が読み出されます。書き込む値も常に 0 にしてください。

ビット 3～0：BN3～BN0

次に退避されるバンク番号を示します。レジスタバンクを使用した割り込みが受け付けられたとき、BN3～0 が示すレジスタバンクに退避を行い、BN を＋1 します。レジスタバンク復帰命令の実行により、BN を－1 した後、レジスタバンクから復帰を行います。

読み出しのみ有効です。書き込みは無効です。

7.3 バンク退避、復帰の動作

7.3.1 バンクへの退避

図 7.2 にレジスタバンクへの退避の動作を示します。割り込みが発生し、CPU で受け付けられた割り込みのレジスタバンク使用が IBCR で許可されている場合、次のように動作します。

(a) 割り込み発生前のバンク番号レジスタIBNRのバンク番号BNの値をiとします。

(b) BNの示すバンクiに、レジスタR0～R14、GBR、MACH、MACL、PRと、受け付けられた割り込みのベクタテーブルアドレスオフセット（VTO）を退避します。

(c) BNの値を+1します。

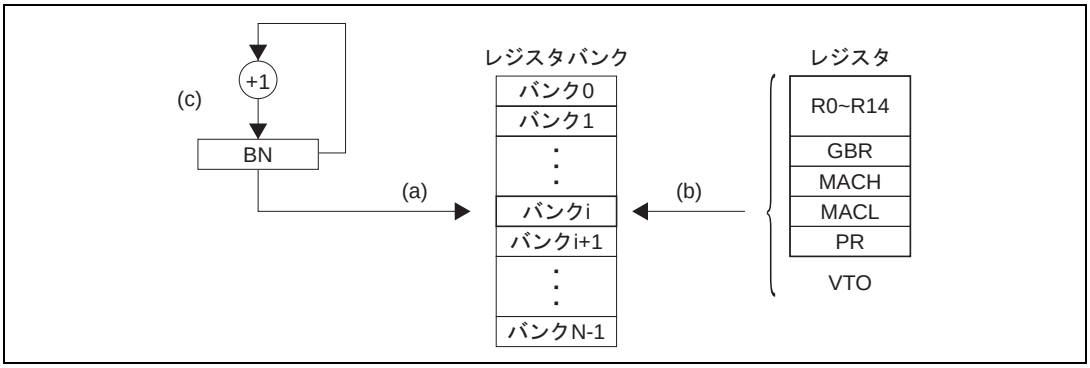


図 7.2 バンク退避の動作

図 7.3 にレジスタバンク退避のタイミングを示します。レジスタバンクへの退避は、割り込み例外処理開始から例外サービスルーチンの先頭命令のフェッチを開始するまでの間に実行されます。

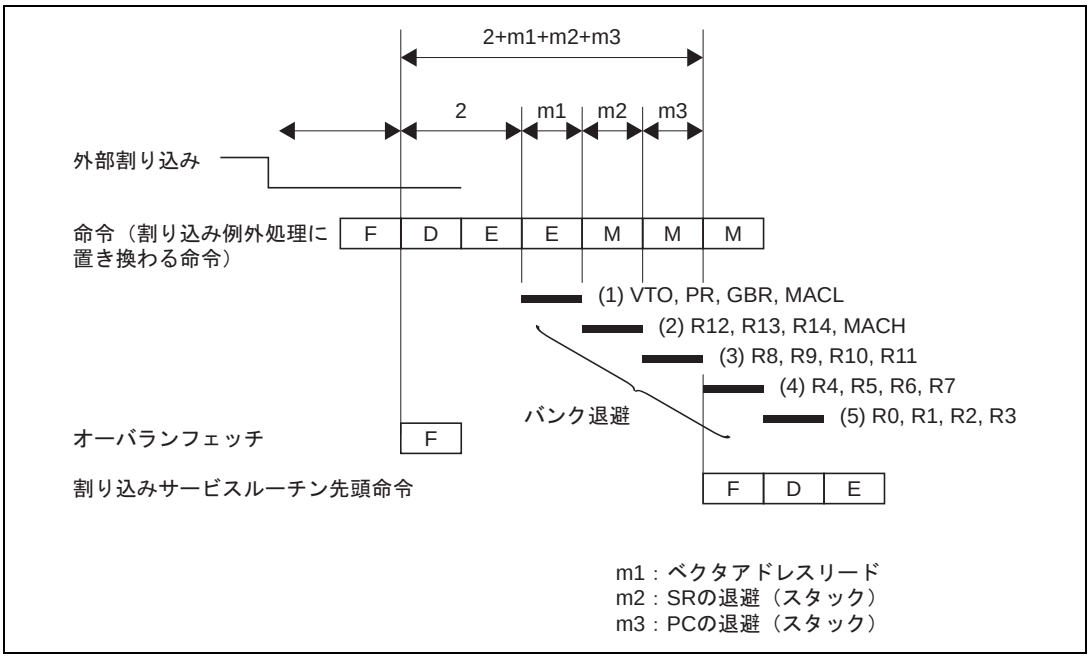


図 7.3 バンク退避のタイミング

7.3.2　バンクからの復帰

バンクに退避したデータを復帰するには、バンク復帰命令 RESBANK を使います。割り込みサービスルーチンの最後に、RESBANK 命令でバンク復帰を行った後、RTE 命令で例外処理からの復帰を行ってください。

7.3.3　すべてのバンクに退避が行われた状態での退避、復帰

レジスタバンクのすべてのバンクに退避が行われている状態で、割り込みが発生し、CPU で受け付けられた割り込みがレジスタバンクの使用を許可されている場合、レジスタバンクの代わりにスタックに自動的に退避を行わせることができます。これは、割り込みコントローラでレジスタバンクオーバフロー例外をマスクすることで可能になります。レジスタバンクオーバフロー例外を発生させた場合には、スタックへの退避は行われません。詳しくは、ハードウェアマニュアルの**「割り込みコントローラ」**を参照してください。スタックへの自動退避、復帰の動作は次のようになります。

(1)　スタックへの退避

（a）　割り込み例外処理時に、ステータスレジスタ（SR）、プログラムカウンタ（PC）をスタックに退避します。
（b）　バンク対象レジスタ（R0～R14、GBR、MACH、MACL、PR）をスタックに退避します。スタックに退避するレジスタの順番は、MACL、MACH、GBR、PR、R14、R13、‥‥、R1、R0の順となります。
（c）　SRのレジスタバンクオーバフロービットを1にセットします。
（d）　バンク番号レジスタIBNRのバンク番号BNは最大値Nのまま変化しません。

(2)　スタックからの復帰

SR のレジスタバンクオーバフロービットが 1 にセットされている状態で、バンク復帰命令 RESBANK を実行すると、次のように動作します。

（a）　バンク対象レジスタ（R0～R14、GBR、MACH、MACL、PR）をスタックから復帰します。スタックから復帰するレジスタの順番はR0、R1、‥‥、R13、R14、PR、GBR、MACH、MACLの順となります。
（b）　バンク番号レジスタIBNRのバンク番号BNは最大値Nのまま変化しません。

7.4　レジスタバンクデータ転送命令

デバッグのために、レジスタバンクの任意のデータを汎用レジスタ R0 との間で転送する、LDBANK 命令、STBANK 命令があります。

7.4.1　命令の説明

(1)　LDBANK（レジスタバンクから R0 へのロード）

[書式]LDBANK @Rm,R0
[動作]Rm が示すレジスタバンクアドレスからの 4B データを R0 に転送します。

(2)　STBANK（R0 からレジスタバンクへのストア）

[書式]STBANK R0,@Rn
[動作]R0 を Rn が示すレジスタバンクアドレスへ転送します。

7.4.2 レジスタバンクのアドレッシング

図 7.4 は、レジスタバンク転送命令のアドレス（LDBANK は Rm、STBANK は Rn の値）と、レジスタバンクのエントリの対応を示します。アドレスの 15～7 ビット（BN）でバンク番号を指定し、アドレスの 6～2 ビット（EN）でバンク内のエントリ（R0～R14、GBR、MACH、MACL、PR、VTO）を指定します。アドレスの 31～16 ビットと 1～0 ビットはすべて 0 にしてください。アドレスの 31～16 ビットと 1～0 ビットがすべて 0 でない場合、アドレスの 15～7 ビットで存在しないバンクを指定した場合、アドレスの 6～2 ビットで存在しないエントリを指定した場合は、動作を保証しません。

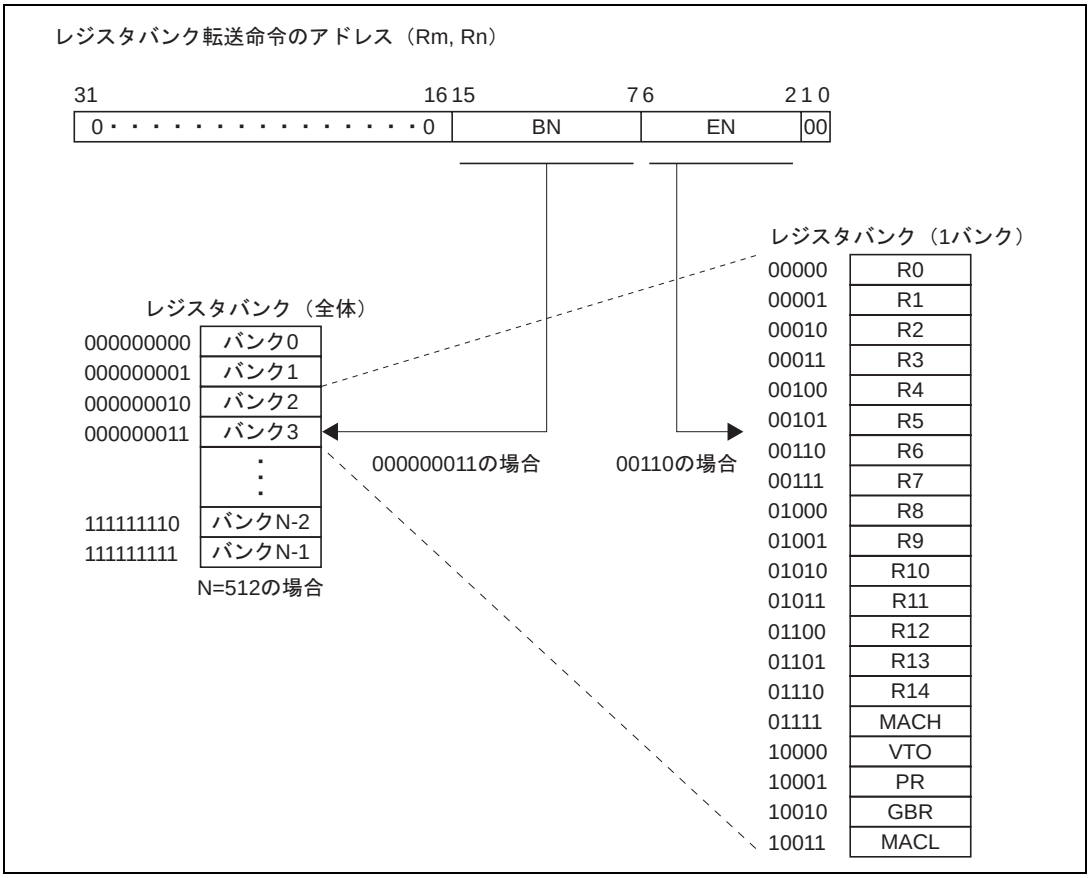


図 7.4 レジスタバンクのアドレッシング

7.5　レジスタバンクの例外

レジスタバンクの例外（レジスタバンクエラー）には、レジスタバンクオーバフローとレジスタバンクアンダフローの 2 種類があります。

7.5.1　レジスタバンクエラー発生要因

(1)　レジスタバンクオーバフロー

レジスタバンクのすべてのバンクに退避が行われている状態で、割り込みが発生し、CPU で受け付けられた割り込みがレジスタバンクの使用を許可されている場合、割り込みコントローラでレジスタバンクオーバフロー例外がマスクされていない場合に発生します。このとき、バンク番号レジスタ IBNR のバンク番号 BN はバンク数 N のまま変化せず、レジスタバンクへの退避は行われません。

(2)　レジスタバンクアンダフロー

レジスタバンクに退避が全く行われていない状態で、RESBANK 命令を実行した場合に発生します。このとき、R0～R14、GBR、MACH、MACL、PR の値は変化しません。また、バンク番号レジスタ IBNR のバンク番号 BN は 0 のまま変化しません。

7.5.2　レジスタバンクエラー例外処理

レジスタバンクエラーが発生すると、レジスタバンクエラー例外処理が開始されます。このとき、CPU は次のように動作します。

(1)　ステータスレジスタ（SR）をスタックに退避します。
(2)　プログラムカウンタ（PC）をスタックに退避します。レジスタバンクオーバフロー時の退避するPCの値は、最後に実行した命令の次命令の先頭アドレスです。レジスタバンクアンダフロー時の退避するPCの値は、当該のRESBANK命令の先頭アドレスです。
バンクオーバフロー時は多重割り込みを防止するために、バンクオーバフローの要因となった割り込みのレベルを、ステータスレジスタ（SR）の割り込みマスクビット（I3～I0）に書き込みます。
(3)　発生したレジスタバンクエラーに対応する例外処理ベクタテーブルから例外サービスルーチンスタートアドレスを取り出し、そのアドレスからプログラムを実行します。

7.6　SR のレジスタバンクオーバフロービット（BO ビット）

BO ビットは、RTE 命令による SR の復帰により書き換わります。BO ビットは、RESBANK 命令を実行しても書き換わりません。BO ビットは、割り込み時のバンクオーバフローで割り込みコントローラ内の例外発生イネーブルを OFF したときに、1 にセットされます。BO ビットは、割り込み時のバンクオーバフローで割り込みコントローラ内の例外発生イネーブルを ON したときには、書き換わりません。BO ビットは、LDC Rm, SR, LDC.L@Rm＋, SR 命令により書き換わります。

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。

SH-2A/SH2A-FPU は 2 命令並列型（2-ILP, Instruction-Level-Parallelism）のスーパスカラパイプライン処理マイクロプロセッサです。命令実行はパイプライン化され、2 つの命令を並行して実行できます。また、ハーバードアーキテクチャを採用しており、メモリアクセスと命令フェッチは競合しません。さらに、命令フェッチのユニットを持つため、命令フェッチ中でも CPU コアが停止することはありません。

8.1 パイプラインの基本構成

SH-2A/SH2A-FPU では、以下のパイプラインを持っています。（図 8.1）

- 整数パイプライン 1，2　　：整数演算を処理します。
- メモリアクセスパイプライン　：メモリアクセスや FPU へのデータのロードを処理します。
- 乗算器パイプライン　　：乗算命令や FPU からのデータのストアを処理します。
- 分岐パイプライン　　：分岐命令を処理します。
- シフトパイプライン　　：シフト命令を処理します。
- FPU ロードストアパイプライン：FPU のロードストア命令を処理します。
- FPU 算術演算パイプライン　：FPU の算術演算を処理します。
- FPU 除開平算パイプライン　：FPU の除算、開平算を処理します。

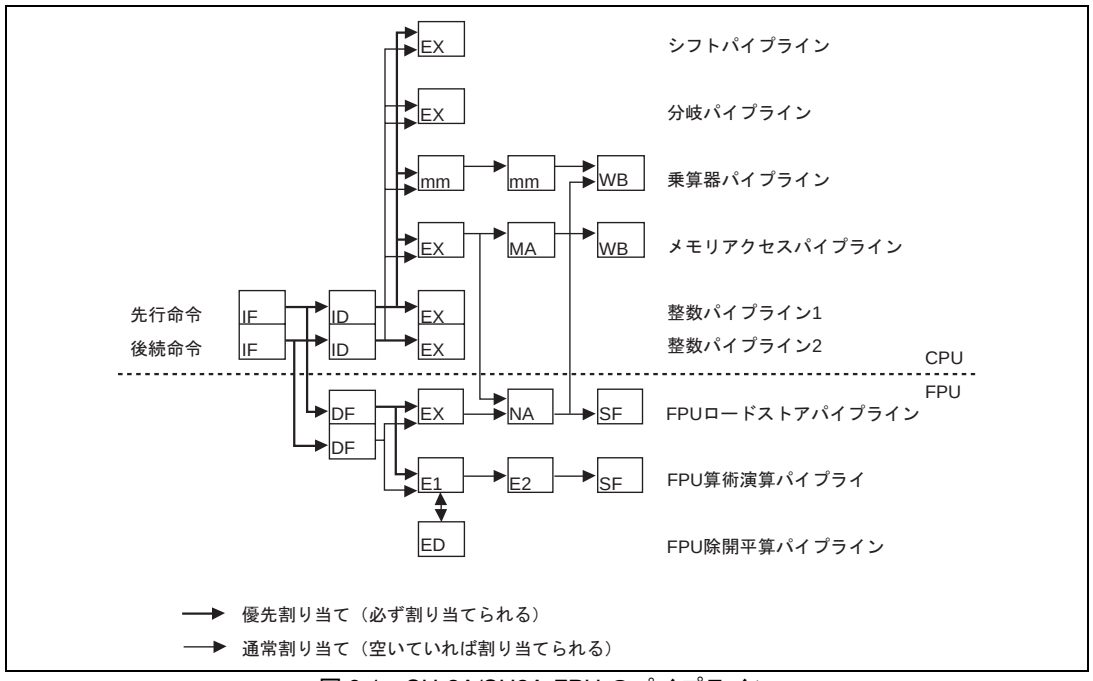


図 8.1　SH-2A/SH2A-FPU のパイプライン

すべての命令はまず整数パイプラインによって処理され、必要であれば他のパイプラインにも渡されます。これらのパイプラインはすべて独立に動作することができます。したがって、競合がなければ、常に 2 つの命令を発行しつづけることができます。

メモリアクセスを行う命令、CPU から FPU へのデータのロードを行う命令はメモリアクセスパイプラインを使用します。

乗算命令、乗算結果レジスタアクセス命令は、乗算器パイプラインを使用します。また、FPU からのデータのストアを行う命令は、乗算器パイプラインの WB ステージを使用します。

分岐命令は、分岐パイプラインを使用します。

シフト命令は、シフトパイプラインを使用します。

FPU 内部でのレジスタの移動、FPU からメモリ、CPU へのデータのやり取りを行う命令は、FPU ロードストアパイプラインを使用します。

FPU 算術演算を行う命令は、FPU 算術演算パイプラインを使用します。

FPU 算術演算のうち、FDIV、FSQRT に関しては、FPU 算術演算パイプラインと FPU 除開平算パイプラインを使用します。

詳しくは**「8.9 各命令のパイプライン動作」**を参照してください。

各 CPU パイプラインのステージの詳細を以下に説明します。

- 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やCPUからFPUへのデータのロードを行う命令で発生します。
- mm ：乗算器アクセス　乗算器のアクセスを行います。乗算命令や、乗算結果レジスタへのアクセスを行う命令で発生します。
- WB ：ライトバック　メモリアクセス、乗算器アクセスした結果（データ）をレジスタに戻します。

各 FPU パイプラインのステージの詳細を以下に説明します。CPU と FPU のパイプラインは第 1 ステージの命令フェッチ（IF）を共有します。

- 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　浮動小数点演算を実行します。
- SF ：FPU ストア　浮動小数点演算を完了、FPU レジスタに結果を書き込みます。
- ED ：FPU 除開平算　FDIV、FSQRT のみに用います。
- EX ：FPU ロードストア第 1 ステージ　浮動小数点ロードストア命令のデータ準備を行います。
- NA ：FPU ロードストア第 2 ステージ　浮動小数点ロードストア命令のデータ交換を行います。

ID、DF 以降の各ステージの長さはすべて同じです。IF のみデータ待ちで延びることがありますが、命令フェッチユニットとパイプラインは独立に動作するため、この場合においても、すでにフェッチを完了している命令のパイプラインは動作しつづけます。

図 8.2 に示すように、命令の各ステージは、命令の実行と共に流れていき、パイプラインを構成します。基本的なパイプラインの流れを図 8.2 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よび“←→”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（整数パイプライン）はすべての命令に存在します。その後、必要なパイプラインも同時に動作し、命令の処理を行います。

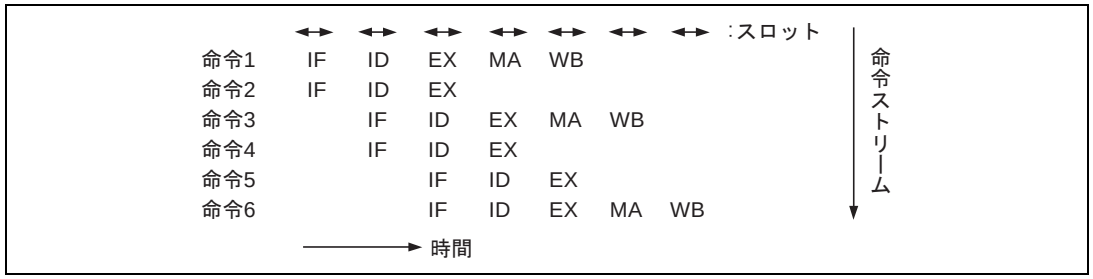


図 8.2　パイプラインの基本構成

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よびます。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、mm、E1、E2、DF、ED、SF、NA）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8.3）。なお、EDステージはスロットにかかわらず動作します。

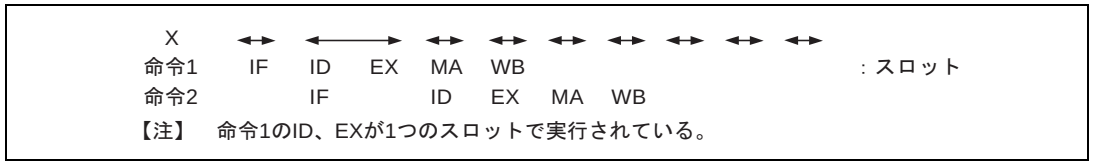


図 8.3　ありえないパイプラインの流れ（1）

(2)　1スロットには別の命令の異なるステージが整数パイプラインは最大2つ、その他のパイプラインは最大1つまで設定されます。これ以上の数のパイプラインが同時に実行されることはありません。（図8.4）。

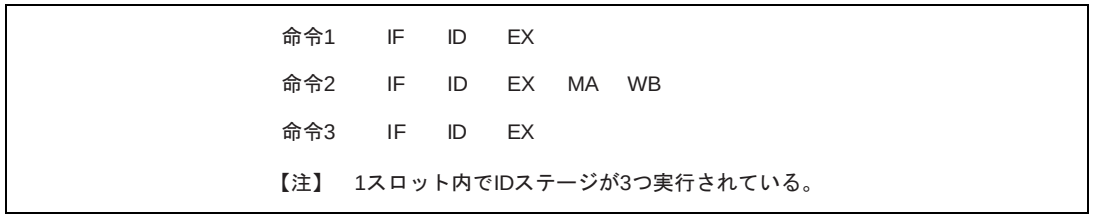


図 8.4　ありえないパイプラインの流れ（2）

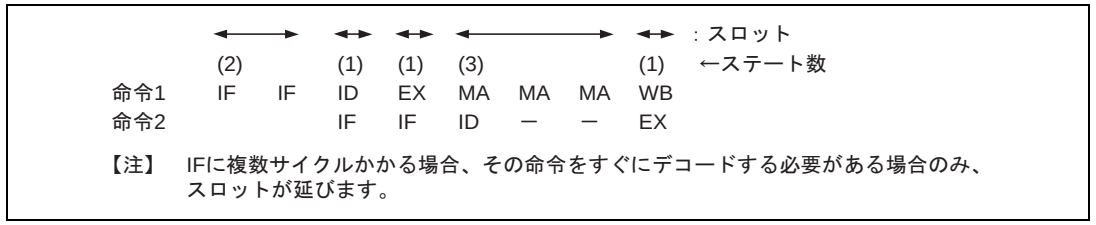
(3)　1スロットの実行にかかるステート数（システムクロックサイクル数）Sは次の条件で算出します。

(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
  （フェッチバッファを持ち、あらかじめ命令フェッチを行っていますので、フェッチした命令をすぐにデコードする必要がある場合を除き、パイプラインをストールさせることはありません。）
- 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
- mm ：常に1ステート
- DF ：常に1ステート
- E1 ：常に1ステート
- E2 ：常に1ステート
- SF ：常に1ステート
- ED ：常に1ステート、ただしスロットにかかわらず動作します。
- NA ：常に1ステート

たとえば、図8.5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、ほかは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。"ー"はストールを表しています。なお、この図では簡単のため、スーパスカラを考慮しておりません。



【注】 IFに複数サイクルかかる場合、その命令をすぐにデコードする必要がある場合のみ、スロットが延びます。

図 8.5 複数サイクルかかるスロット

8.3 命令実行、並列実行性

SH-2A/SH2A-FPU は 2 命令並列型（2-ILP, Instruction-Level-Parallelism）のスーパスカラパイプライン処理マイクロプロセッサです。命令が 2 個 ID ステージに入っているとき、2 命令を同時に実行することができます。（図 8.6）

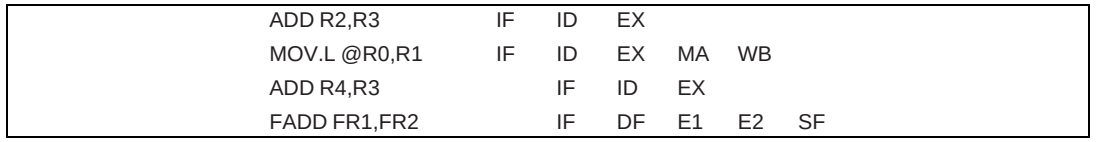


図 8.6 並列実行の例

ただし、次の場合は並列実行できません。

- リソース競合が起きているとき（8.3.1 にて説明）
- 既発行命令の結果待ちのとき（8.3.2 にて説明）
- レジスタ競合・フラグ競合が起きているとき（8.3.3 にて説明）
- マルチサイクル命令が先行命令として実行されたとき（8.3.4 にて説明）
- 32 ビット命令が先行命令として実行されたとき（8.3.5 にて説明）
- FPSCR を使用する命令と、FPU 命令および FPU に関係する CPU 命令（8.3.6 にて説明）

- 分岐が起きた遅延付き無条件分岐命令と、遅延スロット（8.3.7 にて説明）

これらの競合が発生せず、IF が 2 命令分完了しているとき、SH-2A/SH2A-FPU は 2 命令を並列実行することができます。

以下の項にて、それぞれについて説明します。説明時には、以下の呼称を用います。

- 先行命令 … 同一スロット内で、前側の命令
- 後行命令 … 同一スロット内で、後ろ側の命令
- 既発行命令 … すでに発行された命令の総称

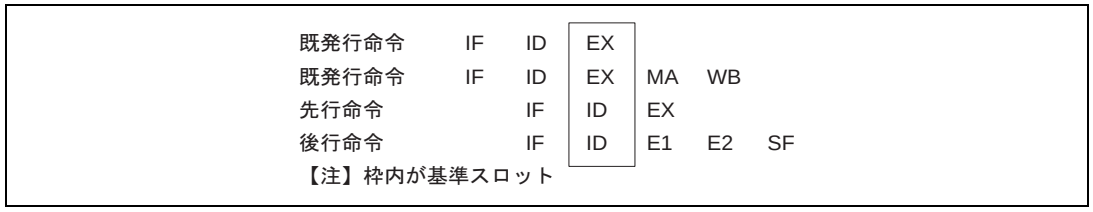


図 8.7　先行、後行、既発行命令の定義

8.3.1　リソース競合の詳細

整数パイプライン以外のパイプラインは各 1 本ずつしか持っていないため、先行命令と後行命令が同時に使用しようとすると、競合が発生し、後行命令は実行を待たされます。競合が発生する場合は、以下のとおりです。

(1)　先行命令と後行命令が、ともにメモリアクセスを伴う命令であるとき。（図8.8）
　　または、CPU→FPUへのデータ転送命令と、メモリライト命令組み合わせ。（図8.9）
　　あるいは共にCPU→FPUへのデータ転送命令の組み合わせ。
　　このとき、メモリアクセスパイプラインの競合が発生します。

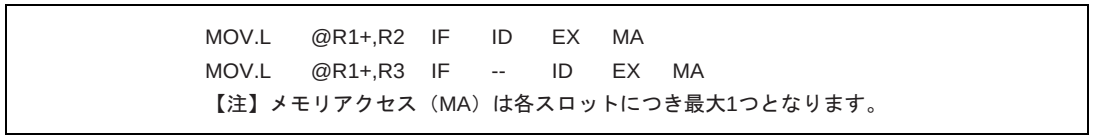


図 8.8　メモリアクセス競合の例

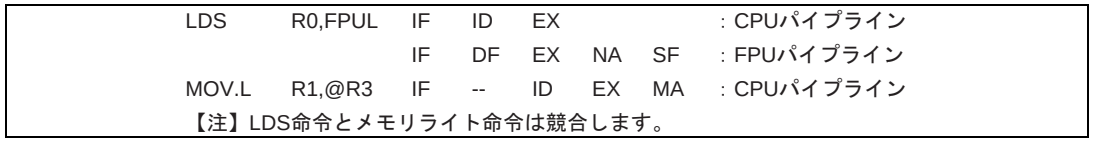


図 8.9　LDS 命令とメモリライト命令の競合の例

なお、FPU→CPUへのデータ転送命令と、メモリアクセス命令は競合しません。（図8.10）
また、CPU→FPUへのデータ転送命令と、メモリリード命令も競合しません。（図8.11）

```
STS      FPUL,R0   IF  ID  EX  WB              : CPUパイプライン
                   IF  DF  EX  NA  SF          : FPUパイプライン
MOV.L    @R1+,R3   IF  ID  EX  MA  WB  WB      : CPUパイプライン
```

【注】STS命令とメモリアクセス命令は競合しません。

図 8.10 STS 命令とメモリアクセス命令の例

```
LDS      R0,FPUL   IF  ID  EX                  : CPUパイプライン
                   IF  DF  EX  NA  SF          : FPUパイプライン
MOV.L    @R1+,R3   IF  ID  EX  MA  WB          : CPUパイプライン
```

【注】LDS命令とメモリリード命令は競合しません。

図 8.11 LDS 命令とメモリリード命令の例

(2) 先行命令と後行命令が、共に乗算器を使用する命令であるとき。（図8.12）
ただし、乗算器の場合は既発行命令がロックしているときも、競合が起きます。（図8.13）
また、MACH、MACLを読み出す命令、MULR命令、FPUL、FPSCRの値をCPUに転送する命令は、読み出しバスを共有しているため、競合します。（図8.14）

```
MULS.W   R2,R1     IF  ID  mm  mm
MULR     R0,R3     IF  --  ID  mm  mm  mm  WB
```

図 8.12 乗算器競合の例

```
乗算器ロック                    ←→
LDS.L    @R1+,MACH IF  ID  EX  MA  WB
MULR     R0,R3     IF  --  --  ID  mm  mm  mm  WB
```

図 8.13 既発行命令によって競合する例

```
STS      MACH,R0   IF  ID  EX  MA  WB
STS      FPUL,R1   IF  --  ID  mm  mm  mm  WB
```

【注】乗算結果読み出しバスを使用する命令どうしは競合します。

図 8.14 乗算結果読み出しバスを使用する命令の競合

(3) 先行命令と後行命令が、ともにシフト命令またはローテート命令であるとき。（図8.15）

```
SHAD R0,R1       IF  ID  EX
SHAD R2,R3       IF  --  ID  EX
```

図 8.15 シフト命令での競合の例

(4) 先行命令と後行命令がともにFPU算術演算命令であるとき。（図8.16）
また、FPU算術演算命令については、倍精度であるときや、FDIV、FSQRT命令では複雑にリソース競合を起こします。これについては、「8.6 FPUによる競合」を参照してください。

```
FADD FR0,FR1    IF  DF  E1  E2  SF
FADD FR2,FR3    IF  --  DF  E1  E2  SF
```

図 8.16 FPU 算術演算命令での競合の例

(5) 先行命令と後行命令がともにFPUロードストア命令であるとき。（図8.17）

```
FNEG FR0        IF  DF  EX  NA  SF
FMOV FR1,FR3    IF  --  DF  EX  NA  SF
```

図 8.17 FPU ロードストア命令での競合の例

8.3.2 既発行命令の結果待ちによる競合の詳細

既発行命令の結果をソースとして使用する場合、その命令のレイテンシだけ待たされてから実行されます。これには、以下のケースなどがあります。

- メモリアクセスの結果を待つ場合（詳しくは、「8.5 メモリロード命令によるパイプラインへの影響」を参照してください）
- FPU 演算結果を待つ場合（詳しくは、「8.6 FPU による競合」を参照してください）
- 乗算結果を待つ場合。（詳しくは、「8.7 乗算器による競合」を参照してください）

このとき、先行命令が競合を起こした場合、後行命令も実行を待たされます。
後行命令が競合を起こした場合、先行命令はほかに競合がなければ実行されます。

8.3.3 レジスタ競合・フラグ競合の詳細

以下の場合に、同一スロット内でのレジスタ競合・フラグ競合が発生します。

(1) 先行命令のディスティネーションレジスタ、フラグをソースレジスタ、フラグとして後行命令が使用するとき。（ただし、先行命令がレイテンシ0の命令であった場合を除きます）（図8.18、図8.19）

```
CMP/EQ R2,R3    IF  ID  EX
BF              IF  --  ID  EX
```

図 8.18 先行ディスティネーション、後行ソースでのフラグ競合の例

```
MOV  R3,R4      IF  ID  EX
ADD  R4,R5      IF  ID  EX
```

図 8.19 レイテンシ 0 命令と後行命令で競合しない例

(2) 先行命令のディスティネーションレジスタ、フラグに対して後行命令が書き込む場合。（ただし、競合となるのは、FPUレジスタ、CSビット以外のレジスタ、フラグに対して、乗算命令、除算命令、LDBANK命令、RESBANK命令、MOVMU命令、MOVML命令以外の命令で書き込む場合のみとなります。乗算命令、除算命令、LDBANK命令、RESBANK命令においては、この競合は検出されません。MOVMU命令においてはRn、MOVML命令においてはR0のみに対してこの競合が検出され、これらの命令でのこれ以外のレジスタに対しては、この競合は発生しません。）（図8.20～図8.25）

```
ADD R3,R4      IF   ID   EX
MOV R5,R4      IF   --   ID   EX
```

図 8.20 先行命令のディスティネーションに対して上書きする命令が競合する例 1

```
MOV.L @R0,R1   IF   ID   EX   MA
MOV.L @R2,R1   IF   --   --   ID   EX
```

図 8.21 先行命令のディスティネーションに対して上書きする命令が競合する例 2

```
CLIPS.B R3     IF   ID   EX
CLIPS.B R4     IF   ID   EX
```

図 8.22 CS ビットが競合しない例

```
MOV   R5,R6    IF   ID   EX
MULR  R0,R6    IF   ID   mm   mm   mm   WB
```

図 8.23 MULR が競合しない例

```
MOV      R5,R6         IF   ID   EX
MOVMU.L  @R15+,R13     IF   ID   EX   MA   MA   MA   WB
```

図 8.24 MOVMU.L が競合しない例

```
MOV      R5,R13        IF   ID   EX
MOVMU.L  @R15+,R13     IF   --   ID   EX   MA   MA   MA   WB
```

図 8.25 MOVMU.L が競合する例

8.3.4 マルチサイクル命令による競合の詳細

実行ステートが 1 でない命令を「マルチサイクル命令」とよびます。この命令においては、以下に示すルールがあります。

(1) マルチサイクル命令が先行命令として実行されるとき、後行命令と並列実行できません。

(2) マルチサイクル命令の実行中、最後のスロットでないときは、新たに次の命令を実行することはできません。ここにおいて、実行中とは、命令のIDステージから数えて実行ステートサイクル以下のスロットを指しています。

(3) マルチサイクル命令の実行ステートの最後（最終スロット：実行ステートサイクル目）においては、次の命令と並列実行することができます。次の命令が32ビット命令であっても並列実行できます。

(4) マルチサイクル命令は、先行命令のシングルサイクル命令（実行ステート=1の命令）と並列実行できます。
このときの例を、図8.26に示します。

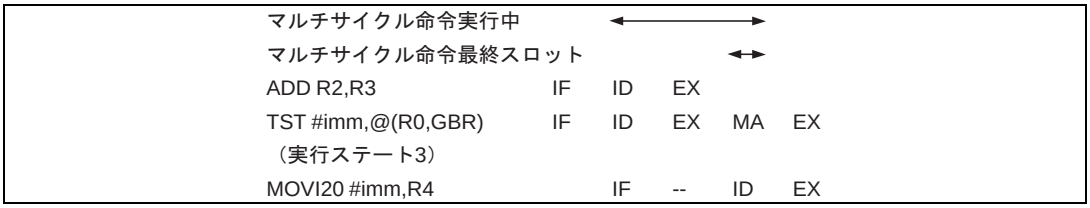


図 8.26　マルチサイクル命令の実行の例

(5)　32ビット命令のマルチサイクル命令であるBAND.B、BANDNOT.B、BLD.B、BLDNOT.B、BOR.B、BORNOT.B、BXOR命令は、後行命令がBAND.B、BANDNOT.B、BLD.B、BLDNOT.B、BOR.B、BORNOT.B、BXOR命令の場合に、後行命令と並列実行できます。（図8.27）

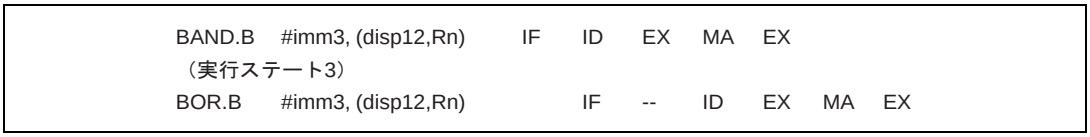


図 8.27　32 ビットのビット操作命令連続時の実行の例

(6)　32ビット命令のマルチサイクル命令は、(5) の場合を除き、後行命令とは並列実行できません。（図8.28）

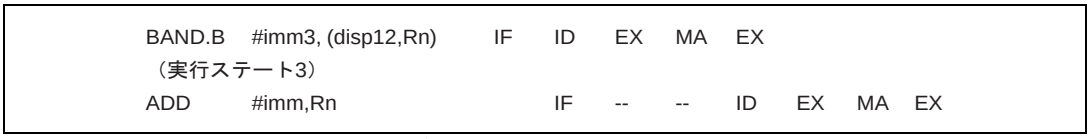


図 8.28　32 ビットマルチサイクル命令の実行の例

8.3.5　32 ビット命令による競合の詳細

32 ビット命令の実行に関しては、以下のルールがあります。

(1)　先行命令が32ビット命令であるとき、並列実行できません。（図8.29）
(2)　後行命令が32ビット命令であるとき、先行命令は実行できますが、後行命令は実行できません。（図8.29）
(3)　マルチサイクル命令の最終スロットと、32ビット命令は並列実行できます。（図8.26）
(4)　先行命令が32ビットのマルチサイクル命令であるとき、BAND.B、BANDNOT.B、BLD.B、BLDNOT.B、BOR.B、BORNOT.B、BXOR命令の最終スロットと、後行命令がBAND.B、BANDNOT.B、BLD.B、BLDNOT.B、BOR.B、BORNOT.B、BXOR命令の場合のみ、並列実行できますが、それ以外の命令の組み合わせでは、並列実行できません。（図8.27、図8.28）
(5)　32ビット命令は、上側16ビットと下側16ビットがIFを完了していないと、実行できません。（図8.30）

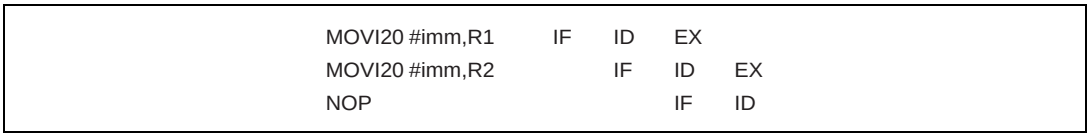


図 8.29　32 ビット命令での競合の例

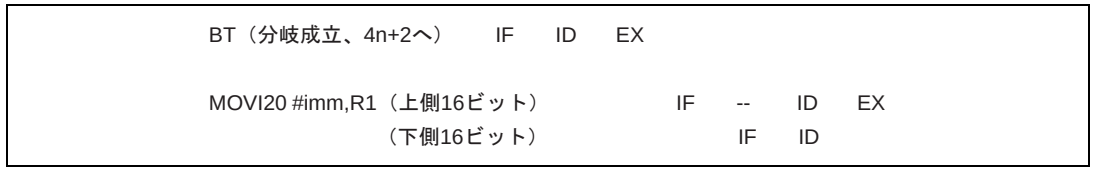


図 8.30　32 ビット命令の内部ストールの例

8.3.6　FPSCR 使用命令による競合の詳細

FPSCR を使用する命令は、この命令が先行命令として供給された場合、すべての命令と並列実行できません。この命令が後行命令として供給された場合、FPU 命令および FPU に関する CPU 命令と並列実行することができません。（図 8.31）

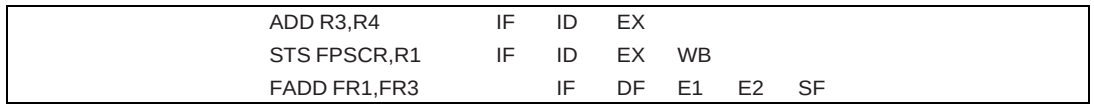


図 8.31　FPSCR を使用する命令の競合例

8.3.7　分岐命令による競合の詳細

分岐命令による競合には、以下のルールがあります。

(1)　分岐命令が分岐しないとき、並列実行できます。
(2)　分岐命令が後行命令として供給された場合、分岐のいかにかかわらず先行命令と並列実行できます。
(3)　分岐命令が先行命令として供給され、分岐が発生する場合、後行命令と並列実行できません。すでに遅延スロットがIFを完了していても並列実行できません。（図8.32）
(4)　遅延スロットは、分岐命令のEXステージのある次のスロットでIDを行います。
(5)　遅延分岐命令は、遅延スロットに対してのフェッチが行われていない場合、実行を待たされます。
(6)　分岐命令は、分岐命令と競合します。

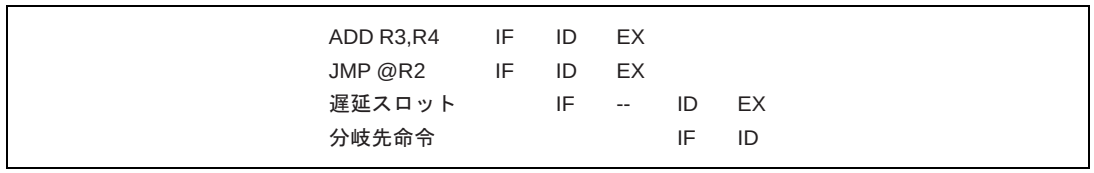


図 8.32　分岐命令の競合例

8.4　命令実行ステート数

各命令の実行ステート数は、EX ステージの実行間隔で数えます。命令 1 の EX ステージ実行開始から、次に続く命令 2 の EX ステージ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 8.33 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 4 ステージですから、命令 1 の実行時間は 4 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 8.33 の場合、命令 3 の実行時間は 1 ステートになります）。

図 8.33 の命令 1～命令 3 までの実行時間は合計 4＋1＋1＝6 ステートとなります。
なお、この図では簡単のため、スーパスカラを考慮しておりません。

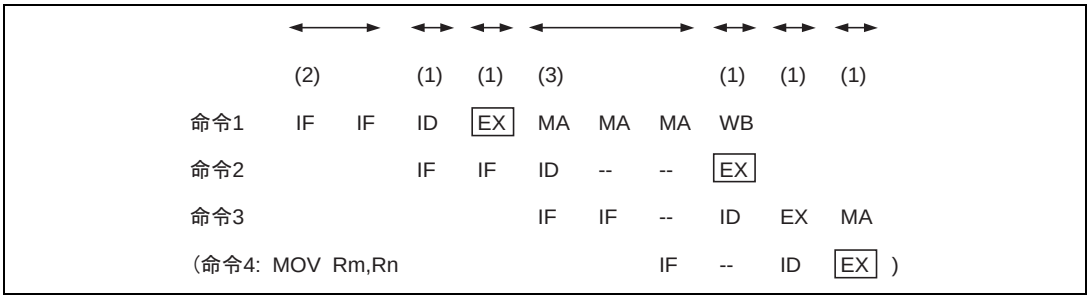


図 8.33　命令実行ステート数の数え方の例

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 ステージがくる前に、命令 2 の EX ステージがくることになります。

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 2 の ID は命令 1 のレイテンシだけ実行を待たされます。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同様です。

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）はさらに 1 スロットストールします。

このようなレジスタ競合が起こった場合、このディスティネーションレジスタを使用することができるスロットは、命令 1 の MA 完了後のサイクルとなります。その様子を図 8.34 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。一般的な場合、ロード命令のレイテンシは 2 となっておりますので、ロード結果を使う命令は、ロード命令の 3～4 命令以後に置くと速度が低下しません。メモリアクセス命令が先行命令として実行された場合 4 命令以降、後行命令として実行された場合、3 命令以降となります。

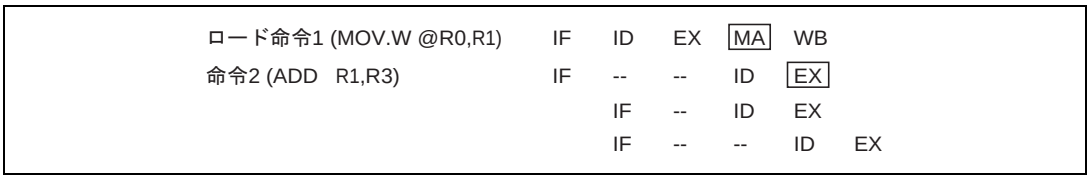


図 8.34　メモリロード命令によるパイプラインへの影響

8.6　FPU による競合

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、次に続く浮動小数点数算術演算命令または FMOV FRm,FRn 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、演算が完了してから次の命令が発行されます。この結果、その命令は先の演算命令のレイテンシサイクルだけ待たされます。（図 8.35）レイテンシ 0 の命令は、後行の命令が結果レジスタをソースする場合でも、後行命令と並列実行できます。（図 8.36）

図 8.35 FPU 演算結果を後続命令が使用する場合の例

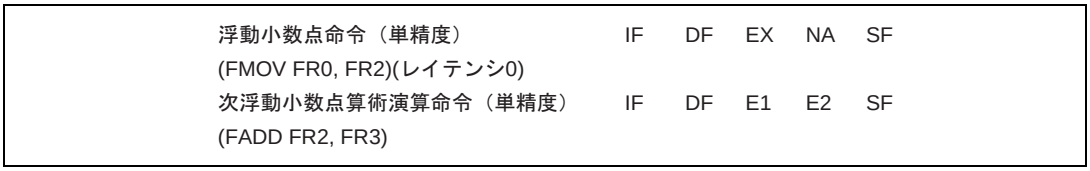


図 8.36 レイテンシ 0 命令の結果をソースとして使用する場合の例

浮動小数点数算術演算命令の結果を格納するレジスタ（FR0～FR15）が、次に続く FMOV、STS.L 命令によってリード（ソースレジスタとして使用）され、メモリに値が出力される場合には、レイテンシが 1 サイクル短縮されます。（図 8.37）

図 8.37 FPU 演算直後にその結果をメモリライトする場合の例

浮動小数点数算術演算命令の結果を格納するレジスタ（FPUL）が、次に続く STS 命令によってリード（ソースレジスタとして使用）され、CPU に値が出力される場合には、レイテンシが 2 サイクル短縮されます。（図 8.38）

図 8.38 FPU 演算直後にその結果を CPU に転送する場合の例

FCMP 命令の結果が Tbit に反映されるまで、単精度では 2 サイクル、倍精度では 3 サイクルを要します。この結果、その命令（次に続く命令）はが T ビットを参照する場合、上記スロット期間分実行が遅延します。（図 8.39）

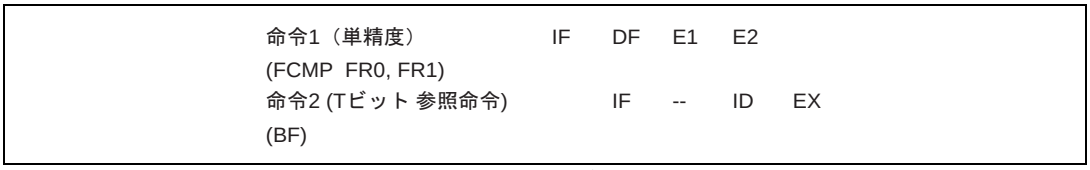


図 8.39 FCMP 命令の直後に T ビットを参照する場合の例

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、3 スロット期間分実行が遅延します。（図 8.40）

命令1	IF	DF	EX	NA	SF				
(LDS R2, FPSCR)									
命令2		IF	--	--	--	DF	E1	E2	SF
(FADD FR4, FR5)									

図 8.40　FPSCR のロード直後に FPU 演算をする場合の例

STS または STS.L 命令を使用して FPSCR の値をリードすると、先に発行した演算が完了してから FPSCR がリードされます。この結果、先の演算のレイテンシ+1 スロット期間分実行が遅延します（**図 8.41**）

命令1（単精度）	IF	DF	E1	E2	SF			
(FADD FR6, FR9)								
命令2		IF	--	--	--	DF	EX	NA	SF
(STS FPSCR, R3)									

図 8.41　FPSCR を読み出す場合の例

倍精度浮動小数点数算術演算命令（FADD、FSUB、FMUL）は、E1 ステージに 6 サイクルを要します。この期間中は、他の浮動小数点算術演算命令が E1 ステージに入ることはありません。倍精度浮動小数点数算術演算命令が E1 ステージを終了する前に、他の浮動小数点算術演算命令が出現した場合、その浮動小数点算術演算命令は所定のスロット期間実行を遅延し、倍精度浮動小数点数算術演算命令が E1 ステージを終了した後に E1 ステージに入ります。なお、この間に次に来る浮動小数点ロードストア命令は、実行することができます。（**図 8.42**）

FADD DR4, DR6	IF	DF	E1	E1	E1	E1	E1	E1	E2	SF
FABS DR0	IF	DF	EX	NA	SF					
STS FPUL, R0		IF	DF	EX	NA					
FMUL DR2, DR0		IF	--	--	--	--	--	DF	E1	E2	SF

図 8.42　倍精度 FPU 演算と次の FPU 命令の動作の例

FDIV、FSQRT 命令は初期化に E1 ステージを使用した後、独立した計算機（ED ステージ）で演算が行われ、その後演算結果が書き戻されます。これらの命令の後の浮動小数点算術演算命令は、以下のように動作します。なお、それぞれの命令がどのようなパイプラインになるかについては、**「8.9 各命令のパイプライン動作」**を参照してください。

(1)　初期化のE1ステージ使用中には、他の浮動小数点算術命令がE1ステージに入ることはありません。これらの命令はFDIV、FSQRTの初期化が終了した後、E1ステージに入ります。

(2)　FDIV、FSQRT命令がEDステージに進んだ後のFPU命令は、FDIV、FSQRT命令の結果レジスタを使用する場合を除き、待たされずに実行されます。（**図8.43**）

(3)　FDIV、FSQRT命令の最後において、演算の書き戻しが発生します。ここでは再びE1ステージを使用するため、ここにちょうど以降の命令がE1ステージでの動作を要求している場合、以降の命令はFDIV、FSQRT命令がE1ステージを使用し終えるまで待たされます。（**図8.44**）

(4)　FDIV、FSQRT直後のFDIV、FSQRTは、先行のFDIV、FSQRTがEDステージを使用している限り、EDステージには入れません。

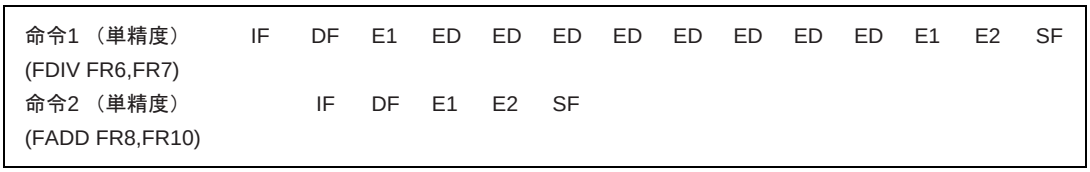


図 8.43 FDIV による E1 ステージ競合の例

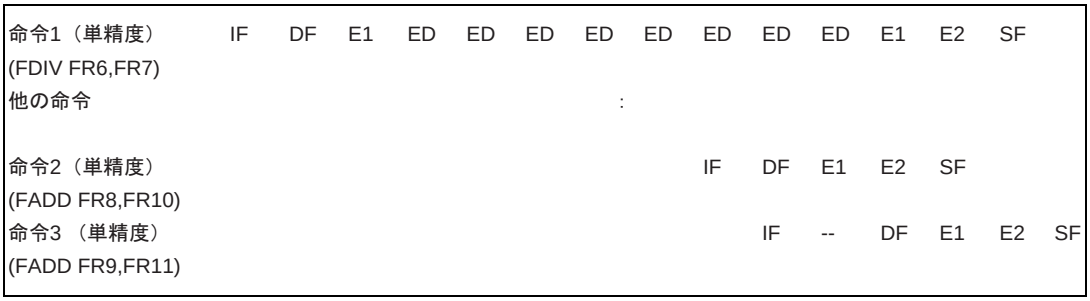


図 8.44 FDIV による E1 ステージ競合の例 2

倍精度算術演算命令でソースレジスタとして使用するレジスタに対して、以前の命令で書き込みが行われており、かつ以前の命令のレイテンシが 2 サイクル以下の場合、それらの命令のレイテンシは 2 となります。（図 8.45）

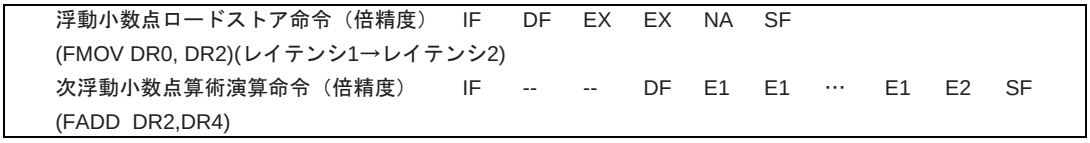


図 8.45 倍精度算術演算直前のレイテンシ 1 の命令の例

倍精度算術演算命令のディスティネーションレジスタを、次の命令がソースレジスタとして使用する場合、FRn の n が奇数であれば 1 サイクルレイテンシが減少します。（図 8.46）ただし、FRn の n が偶数である場合、レイテンシは減少しません。（図 8.47）

図 8.46 倍精度算術演算命令でレイテンシが減少する例

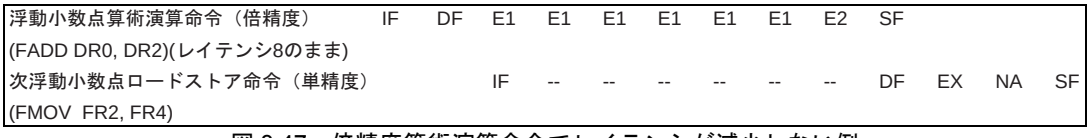


図 8.47 倍精度算術演算命令でレイテンシが減少しない例

浮動小数点数算術演算命令の結果を格納するレジスタ（FR0～FR15、FPUL）が、次に続く浮動小数点数算術演算命令または浮動小数点ロードストア命令によってライト（ディスティネーションレジスタとして使用）された場合、次の命令は待たされてから実行されます。待たされるサイクル数は、先の演算が FDIV、FSQRT だった場合はレイテンシ-1 サイクル、それ以外だった場合はレイテンシ-2 サイクルとなります。（図 8.48）（図 8.49）

浮動小数点数算術演算命令（単精度）	…	ED	E1	E2	SF	
(FDIV FR1, FR2)(レイテンシ12→レイテンシ11)						
次浮動小数点ロードストア命令（単精度）	--	--	DF	EX	NA	SF
(FMOV FR3, FR2)						

図 8.48　上書きによる競合の例（FDIV、FSQRT）

浮動小数点数算術演算命令（単精度）	…	DF	E1	E2	SF
(FADD FR1, FR2)(レイテンシ3→レイテンシ1)					
次浮動小数点ロードストア命令（単精度）	--	DF	EX	NA	SF
(FMOV FR3, FR2)					

図 8.49　上書きによる競合の例（FDIV、FSQRT 以外）

倍精度 FADD、FSUB、FMUL でソースとして使用するレジスタに対して、後続の命令で書き込もうとするとき、後続の命令は 2 サイクル待たされます。（図 8.50）

浮動小数点算術演算命令（倍精度）	IF	DF	E1	E1	E1	E1	E1	E1	E2	SF
(FADD DR0, DR2)(レイテンシ0→レイテンシ2)										
次浮動小数点ロードストア命令（単精度）	IF	--	--	DF	EX	NA	SF			
(FMOV FR4, FR1)										

図 8.50　倍精度演算直後に倍精度命令ソースに書き込む例

8.7　乗算器による競合

乗算命令、積和命令、およびこれらの対象レジスタ（MACH、MACL）を操作する命令は、乗算器を使用します。また、STS FPUL,Rn、STS FPSCR,Rn 命令は乗算器の結果レジスタ読み出しバスを使用します。ここでは、以下のように命令を区分、それぞれのパイプラインと競合の詳細を説明します。命令直後の（A/B/C）は（実行スロット数／レイテンシ／ロックスロット数）を示します。

○積和命令

MAC.L (4/6/5)	IF	ID	EX	MA	MA	mm	mm	mm
MAC.W (3/5/4)	IF	ID	EX	MA	MA	mm	mm	

○乗算命令（I）

DMUL.S、DMUL.U、MUL.L (2/3/2)	IF	ID	mm	mm	mm
MULS.W、MULU.W(1/2/1)	IF	ID	mm	mm	

○乗算命令（II）（レジスタ戻し）

MULR (2/4/2)	IF	ID	mm	mm	mm	WB

○レジスタ書き込み命令（I）

CLRMAC、LDS (1/2/1)	IF	ID	mm	mm

○レジスタ書き込み命令（II）

LDS.L (1/3/2)	IF	ID	EX	MA	WB

○レジスタ読み出し命令（STS FPUL,Rn、STS FPSCR,Rn も含む）

STS (1/2/0)	IF	ID	EX	WB
STS.L (1/2/0)	IF	ID	EX	MA

競合の実際

マルチサイクル命令においては、一般命令と同様、競合が発生します。（図 8.51）詳しくは、「8.3.4

マルチサイクル命令による競合の詳細」を参照してください。

```
MAC.L   @R1+, @R2+      IF  ID  EX  MA  MA  mm  mm  mm
MAC.L   @R3+, @R4+      IF  --  --  --  ID  EX  MA  MA  mm  mm  mm
```

【注】MAC.Lは実行レート4の命令である。

図 8.51　乗算器使用のマルチサイクル命令の例

乗算器を使用する命令に関しては、以下のルールがあります。

(1)　乗算結果をソースとして使用する命令は、その命令のレイテンシだけ実行を待たされます。（**図8.52**）ただし、後続命令がMACH、MACL読み出し命令であった場合、レイテンシ-1サイクルだけ待たされます。（**図8.53**）また、後続命令が積和命令であった場合、待たされることはありません。（**図8.54**）

```
MULR R0,R4      IF  ID  mm  mm  mm  WB
ADD  R4,R       IF  --  --  --  --  ID  EX  WB
```

図 8.52　乗算直後に結果レジスタを参照する例 1

```
MUL.L   R2, R3      IF  ID  mm  mm  mm
STS     MACH, R4    IF  --  --  ID  EX  WB
```

図 8.53　乗算直後に結果レジスタを参照する例 2

```
MAC.W   @R1+, @R2+      IF  ID  EX  MA  MA  mm  mm
MAC.W   @R3+, @R4+      IF  --  --  ID  EX  MA  MA  mm  mm
```

図 8.54　乗算直後に結果レジスタを参照する例 3

(2)　乗算器を使用する命令の後の命令は、前の命令が乗算器をロックしている場合、ロックが外れるまで実行を待たされます。（**図8.55**）

```
MULR1のロック期間      ←――→
MULR1  R0, R1       IF  ID  mm  mm  mm  WB
MULR2  R0, R2       IF  --  --  ID  mm  mm  mm  WB
```

図 8.55　乗算器ロック競合の例

ただし、後続命令が「積和命令」であった場合、ロック解除を待つことはなく、通常のマルチサイクル命令と同様のステート期間だけ待たされて実行されます。（**図 8.56**）

```
MULR1のロック期間      ←――→
MULR1  R0, R1       IF  ID  mm  mm  mm  WB
MAC.L   @R3+, @R4+  IF  --  ID  EX  MA  MA  mm  mm  mm
```

図 8.56　後続が積和命令であり、乗算器ロック競合しない例

また、後続命令が「レジスタ書き込み命令（II）」であったとき、ロック期間が残り 1 スロットであれば実行されます。（**図 8.57**）

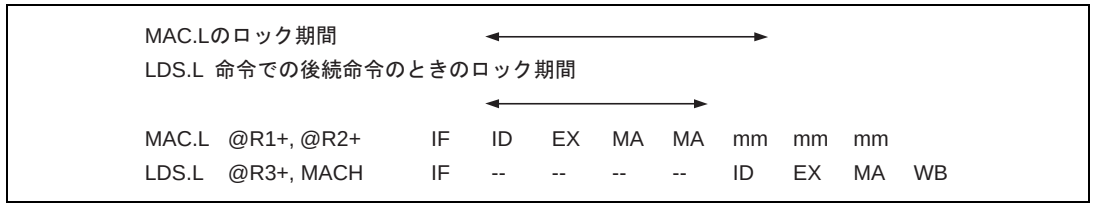


図 8.57　LDS.L 命令がロック期間残り 1 で実行される例

なお、STS、STS.L 命令は乗算器をロックしません。このため、STS 命令と MUL.L 命令などは並列実行できます。（図 8.58）

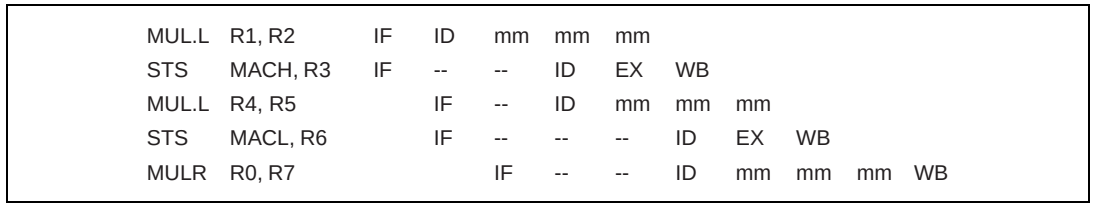


図 8.58　STS 命令と MUL.L 命令の並列実行の例

(3)　MULR命令、MACH、MACL、FPUL、FPSCRに対するSTS命令、MACH、MACLに対するSTS.L命令は結果レジスタ読み出しバスを共有しており、リソース競合を起こします。（MA、WBステージ）したがって、これらのSTS、STS.L命令は並列実行できません。（図8.59）
また、MULR命令の直後にSTS、STS.L命令を配置すると、同様にWBステージの競合が起き、STS、STS.L命令は3サイクル実行を待たされます。（図8.60）

図 8.59　STS、STS.L での競合の例

図 8.60　MULR と STS の競合の例

8.8　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)　分岐先アドレスが、メモリのロングワード境界になるようにします。これにより、分岐直後の並列実行が効率的に行われます。

(2)　メモリからのロード命令の直後の1~3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。ディスティネーションレジスタを使用する命令は、できる限り4命令目以降に配置するようにします。

(3)　32ビット乗算命令の直後の1~3命令には、結果レジスタと同じレジスタを使わない命令を配置します。

(4)　浮動小数点数算術演算命令の直後の1～（その命令のレイテンシ×2）までの命令は浮動小数点数算術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。

8.9 各命令のパイプライン動作

以下に、各命令のパイプラインの動作を説明します。これに、前述のルールや並列実行可能性を加味することにより、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。

以下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。

命令発行の説明においては、リソース競合を考える際に、特にこの命令がどう扱われるべきかを示します。

並列実行の説明においては、並列実行性を考える際に、特にこの命令がどう扱われるべきかを示しています。ここでは、レジスタ競合はないものとして記述されています。

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。なお、この表では、この命令がレジスタの依存性なく実行されたときのステート数を表しています。

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	レイテンシ	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。	実行結果が確定するまでの実行ステート数です。	発生するリソース競合を表します。	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	レイテンシ	競合	命令	
データ転送命令	レジスタ－レジスタ間転送命令	3	1	1	—	MOV	#imm,Rn
			1	0		MOV	Rm,Rn
			1	1		MOVA	@(disp,PC),R0
						MOVT	Rn
						MOVRT	Rn
						NOTT	
						SWAP.B	Rm,Rn
						SWAP.W	Rm,Rn
						XTRCT	Rm,Rn
					・この命令は 32 ビット命令です。	MOVI20	#imm,Rn
						MOVI20S	#imm20,Rn
	メモリロード命令	5	1	2	・この命令はメモリアクセスパイプラインを使用します。	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

分類	区分	ステージ段数	実行ステート	レイテンシ	競合	命令
データ転送命令	メモリロード命令	5	1	2	・この命令はメモリアクセスパイプラインを使用します。	MOV.B @-Rm,R0
						MOV.W @-Rm,R0
						MOV.L @-Rm,R0
						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
		5～20	1～16	2～17		MOVML.L @R15+,Rn
						MOVMU.L @R15+,Rn
		5	1	2	・この命令は 32 ビット命令です。 ・この命令はメモリアクセスパイプラインを使用します。	MOV.B @(disp12,Rm),Rn
						MOV.W @(disp12,Rm),Rn
						MOV.L @(disp12,Rm),Rn
						MOVU.B @(disp12,Rm),Rn
						MOVU.W @(disp12,Rm),Rn
	メモリストア命令	4	1	0	・この命令はメモリアクセスパイプラインを使用します。	MOV.B Rm,@Rn
						MOV.W Rm,@Rn
						MOV.L Rm,@Rn
				1		MOV.B Rm,@-Rn
						MOV.W Rm,@-Rn
						MOV.L Rm,@-Rn
						MOV.B R0,@Rn+
						MOV.W R0,@Rn+
						MOV.L R0,@Rn+
				0		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)
		4～19	1～16	1～16		MOVML.L Rm,@-R15
						MOVMU.L Rm,@-R15

分類	区分	ステージ段数	実行ステート	レイテンシ	競合	命令
データ転送命令	メモリストア命令	4	1	0	・この命令は 32 ビット命令です。 ・この命令はメモリアクセスパイプラインを使用します。	MOV.B Rm,@(disp12,Rn)
						MOV.W Rm,@(disp12,Rn)
						MOV.L Rm,@(disp12,Rn)
	PREF 命令	4	1	0	・この命令はメモリアクセスパイプラインを使用します。	PREF @Rm
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	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
	レジスタ間算術演算命令（乗算系命令 DIVU、DIVS 命令を除く）	3	1	1	—	NEGC Rm,Rn
						SUB Rm,Rn
						SUBC Rm,Rn
						SUBV Rm,Rn
	CLIP 命令	3	1	1	—	CLIPU.B Rn
						CLIPU.W Rn
						CLIPS.B Rn
						CLIPS.W Rn
	積和命令	7	3	4	・この命令は乗算器を 4 スロット期間ロックします。	MAC.W @Rm+,@Rn+

分類	区分	ステージ段数	実行ステート	レイテンシ	競合	命令
算術演算命令	倍精度積和命令	8	4	5	・この命令は乗算器を 5 スロット期間ロックします。	MAC.L @Rm+,@Rn+
	乗算命令	4	1	2	・この命令は乗算器を 1 スロット期間ロックします。	MULS.W Rm,Rn
						MULU.W Rm,Rn
	倍精度乗算命令	5	2	3	・この命令は乗算器を 2 スロット期間ロックします。	DMULS.L Rm,Rn
						DMULU.L Rm,Rn
						MUL.L Rm,Rn
		6	2	4		MULR R0,Rn
	DIVU、DIVS 命令	36	34	34	・この命令はシフトパイプラインを使用します	DIVU R0,Rn
		38	36	36	※DIVS は"ー"を書いてください。	DIVS R0,Rn
論理演算命令	レジスターレジスタ間論理演算命令	3	1	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	2	・この命令はメモリアクセスパイプラインを使用します。	AND.B #imm,@(R0,GBR)
						OR.B #imm,@(R0,GBR)
		5		3		TST.B #imm,@(R0,GBR)
		6		2		XOR.B #imm,@(R0,GBR)
	TAS 命令	6	3	3	・この命令はメモリアクセスパイプラインを使用します。	TAS.B @Rn
ビット操作命令	レジスターレジスタ間ビット演算命令	3	1	1	—	BLD #imm3,Rn
						BSET #imm3,Rn
						BCLR #imm3,Rn
						BST #imm3,Rn
	メモリーTビット間ビット演算命令	5	3	3	・この命令は 32 ビット命令です。 ・この命令はメモリアクセスパイプラインを使用します。	BAND.B #imm3,@(disp12,Rn)
						BANDNOT.B #imm3,@(disp12,Rn)
						BOR.B #imm3,@(disp12,Rn)
						BORNOT.B #imm3,@(disp12,Rn)

分類	区分	ステージ段数	実行ステート	レイテンシ	競合	命令
ビット操作命令	メモリ－Tビット間ビット演算命令	5	3	3	・この命令は 32 ビット命令です。 ・この命令はメモリアクセスパイプラインを使用します。	BLD.B #imm3,@(disp12,Rn)
						BLDNOT.B #imm3,@(disp12,Rn)
						BXOR.B #imm3,@(disp12,Rn)
	メモリビット操作命令	6	3	2		BST.B #imm3,@(disp12,Rn)
						BCLR.B #imm3,@(disp12,Rn)
						BSET.B #imm3,@(disp12,Rn)
シフト命令	シフト命令	3	1	1	・この命令はシフトパイプラインを使用します。	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
						SHAD Rm,Rn
						SHLD Rm,Rn
分岐命令	条件分岐命令	3	3/1*[1]	3/1*[1]	・この命令は分岐パイプラインを使用します。	BF label
						BT label
	遅延付き条件分岐命令	3	2/1*[1]	2/1*[1]	・この命令は分岐パイプラインを使用します。	BF/S label
						BT/S label
	無条件分岐命令	3	2	2	・この命令は分岐パイプラインを使用します。	BRA label
						BRAF Rm
						BSR label
						BSRF Rm
						JMP @Rm
						JSR @Rm
						RTS
	無条件遅延なし分岐命令	3	3	3	・この命令は分岐パイプラインを使用します。	JSR/N @Rm
						RTS/N
						RTV/N Rm

分類	区分	ステージ段数	実行ステート	レイテンシ	競合	命令	
分岐命令	無条件遅延なし分岐命令	5	5	5	・この命令は分岐パイプラインを使用します。 ・この命令はメモリアクセスパイプラインを使用します。	JSR/N	@@(disp,TBR)
システム制御命令	システム制御 ALU 命令	3	1	1	—	CLRT	
		5	3	2		LDC	Rm,SR
		3	1	1	—	LDC	Rm,GBR
						LDC	Rm,TBR
						LDC	Rm,VBR
						LDS	Rm,PR
				0		NOP	
						SETT	
		4	2	2		STC	SR,Rn
		3	1	1		STC	GBR,Rn
						STC	TBR,Rn
						STC	VBR,Rn
						STS	PR,Rn
	LDC.L 命令	7	5	4	・この命令はメモリアクセスパイプラインを使用します。	LDC.L	@Rm+,SR
		5	1	2		LDC.L	@Rm+,GBR
						LDC.L	@Rm+,VBR
	STC.L 命令	5	2	2	・この命令はメモリアクセスパイプラインを使用します。	STC.L	SR,@-Rn
		4	1	1		STC.L	GBR,@-Rn
						STC.L	VBR,@-Rn
	LDS.L 命令（PR）	5	1	2	・この命令はメモリアクセスパイプラインを使用します。	LDS.L	@Rm+,PR
	STS.L 命令（PR）	4	1	1		STS.L	PR,@-Rn
	レジスタ→ MAC 転送命令	4	1	1	・この命令は乗算器を 1 スロット期間ロックします。	CLRMAC	
						LDS	Rm,MACH
						LDS	Rm,MACL
	メモリ→ MAC 転送命令	5	1	2	・この命令は乗算器を 2 スロット期間ロックします。	LDS.L	@Rm+,MACH
						LDS.L	@Rm+,MACL
	MAC→ レジスタ転送命令	4	1	2	・この命令は乗算結果の読み出しバスを使用します。	STS	MACH,Rn
						STS	MACL,Rn
	MAC→ メモリ転送命令	4	1	1	・この命令は乗算結果の読み出しバスを使用します。	STS.L	MACH,@-Rn
						STS.L	MACL,@-Rn
	RTE 命令	8	6	5	—	RTE	
	RESBANK 命令	11/23*2	9/19*2	8/20*2	・BO ビットが 1 のとき、メモリアクセスパイプラインを使用します。	RESBANK	

分類	区分	ステージ段数	実行ステート	レイテンシ	競合	命令	
システム制御命令	LDBANK 命令	8	6	5	—	LDBANK	@Rm,R0
	STBANK 命令	9	7	6	—	STBANK	R0,@Rn
	TRAP 命令	8	5	6	—	TRAPA	#imm
	SLEEP 命令	7	5	0	—	SLEEP	
FPU ロードストア命令	FPUL ロード命令	5	1	1	—	LDS	Rm,FPUL
				2	・この命令はメモリアクセスパイプラインを使用します。	LDS.L	@Rm+,FPUL
	FPSCR ロード命令	5	1	3	—	LDS	Rm,FPSCR
				3	・この命令はメモリアクセスパイプラインを使用します。	LDS.L	@Rm+,FPSCR
	FPUL ストア命令（STS）	4	1	2	・この命令は乗算結果の読み出しバスを使用します。	STS	FPUL,Rn
	FPUL ストア命令（STS.L）	4	1	1	・この命令はメモリアクセスパイプラインを使用します。	STS.L	FPUL,@-Rn
	FPSCR ストア命令（STS）	4	1	2	・この命令は乗算結果の読み出しバスを使用します。	STS	FPSCR,Rn
	FPSCR ストア命令（STS.L）	4	1	1	・この命令はメモリアクセスパイプラインを使用します。	STS.L	FPSCR,@-Rn
単精度浮動小数点命令	浮動小数点レジスタ—レジスタ間転送命令	5	1	0	・この命令は FPU ロードストアパイプラインを使用します。	FLDS	FRm,FPUL
						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	1	0	・この命令は FPU ロードストアパイプラインを使用します。	FLDI0 FLDI1	FRn FRn
	FSCHG 命令	5	1	1	・この命令は、FPU 算術演算パイプラインを使用します。	FSCHG	
	浮動小数点レジスタロード命令	5	1	0/2*[3]	・この命令は FPU ロードストアパイプラインと、メモリアクセスパイプラインを使用します。	FMOV.S	@Rm,FRn
			1	1/2*[3]		FMOV.S	@Rm+,FRn
				0/2*[3]		FMOV.S	@(R0,Rm),FRn

分類	区分	ステージ段数	実行ステート	レイテンシ	競合	命令
単精度浮動小数点命令	浮動小数点レジスタストア命令	4	1	0/2*[3]	・この命令は 32 ビット命令です。 ・この命令は FPU ロードストアパイプラインと、メモリアクセスパイプラインを使用します。	FMOV.S @(disp12,Rm),FRn
				0	・この命令は FPU ロードストアパイプラインと、メモリアクセスパイプラインを使用します。	FMOV.S FRm,@Rn
				1/0*[3]		FMOV.S FRm,@-Rn
				0		FMOV.S FRm,@ (R0,Rn)
					・この命令は 32 ビット命令です。 ・この命令は FPU ロードストアパイプラインと、メモリアクセスパイプラインを使用します。	FMOV.S FRm,@(disp12,Rn)
	浮動小数点演算命令（FDIV 以外）	5	1	3	・この命令は、FPU 算術演算パイプラインを使用します。	FADD FRm,FRn
						FLOAT FPUL,FRn
						FMAC FR0,FRm,FRn
						FMUL FRm,FRn
						FSUB FRm,FRn
						FTRC FRm,FPUL
		5	1	0	・この命令は、FPU ロードストアパイプラインを使用します。	FABS FRn
						FNEG FRn
	浮動小数点演算命令（FDIV、FSQRT）	14	1	12	・この命令は、FPU 算術演算パイプライン、FPU 除開平算パイプラインを使用します。	FDIV FRm,FRn
		13	1	11		FSQRT FRn
	浮動小数点比較命令	4	1	2	・この命令は、FPU 算術演算パイプラインを使用します。	FCMP/EQ FRm,FRn
						FCMP/GT FRm,FRn
倍精度浮動小数点命令	浮動小数点レジスター―レジスタ間転送命令	6	2	1	・この命令は FPU ロードストアパイプラインを使用します。	FMOV DRm,DRn
	FCNVDS、FCNVSD 命令	5	1	4	・この命令は FPU 算術演算パイプラインを使用します。	FCNVSD FPUL,DRn FCNVDS DRm,FPUL

分類	区分	ステージ段数	実行ステート	レイテンシ	競合	命令
倍精度浮動小数点命令	浮動小数点レジスタロード命令	6	2	0/2/3/4*4	・この命令は FPU ロードストアパイプラインと、メモリアクセスパイプラインを使用します。	FMOV.D @Rm,DRn
				1/2/3/4*4		FMOV.D @Rm+,DRn
				0/2/3/4*4		FMOV.D @(R0,Rm),DRn
				0/2/3/4	・この命令は 32 ビット命令です。 ・この命令は FPU ロードストアパイプラインと、メモリアクセスパイプラインを使用します。	FMOV.D @(disp12,Rm),DRn
	浮動小数点レジスタストア命令	5	2	0	・この命令は FPU ロードストアパイプラインと、メモリアクセスパイプラインを使用します。	FMOV.D DRm,@Rn
				1/0*3		FMOV.D DRm,@-Rn
				0		FMOV.D DRm,@ (R0,Rn)
					・この命令は 32 ビット命令です。 ・この命令は FPU ロードストアパイプラインと、メモリアクセスパイプラインを使用します。	FMOV.D DRm,@(disp12,Rn)
	浮動小数点演算命令（FDIV 以外）	10	1	0/8/7/8*4	・この命令は、FPU 算術演算パイプラインを使用します。	FADD DRm,DRn
						FMUL DRm,DRn
						FSUB DRm,DRn
		6	1	0/4*3		FTRC DRm,FPUL
		6	1	0/4/3/4*4		FLOAT FPUL,DRn
		5	1	0	・この命令は、FPU ロードストアパイプラインを使用します。	FABS DRn
						FNEG DRn
	浮動小数点演算命令（FDIV、FSQRT）	27	1	0/25/24/25*4	・この命令は、FPU 算術演算パイプライン、FPU 除開平算パイプラインを使用します。	FDIV DRm,DRn
		26	1	0/24/23/24*4		FSQRT DRn
	浮動小数点比較命令	5	2	3	・この命令は、FPU 算術演算パイプラインを使用します。	FCMP/EQ DRm,DRn
						FCMP/GT DRm,DRn

【注】 *1 分岐しないときは 1 ステートになります。

*2 ステージ段数、実行ステート、レイテンシは、BO ビットが 0 のとき／BO ビットが 1 のときの順で表記しています。

*3 レイテンシは、CPU レジスタ／FPU レジスタの順で表記しています。

*4 レイテンシは、
CPU レジスタ／単精度レジスタとして使用するとき、FRn の偶数側
／単精度レジスタとして使用するとき、FRn の奇数側
／倍精度レジスタとして使用するとき　の順で表記しています。

8.9.1　データ転送命令

(1)　レジスタ-レジスタ間転送命令（MOV Rm, Rn）

●命令の種類

```
MOV        Rm,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 をとおしてデータ転送を行います。

●命令発行

この命令がリソース競合を起こすことはありません。

●並列可能性

この命令はレイテンシ 0 の命令です。この命令が先行命令として実行され、後行命令が Rn を使用するときも、並列実行できます。

(2) レジスタ-レジスタ間転送命令（20 ビット即値）

●命令の種類

```
MOVI20      #imm20,Rn
MOVI20S     #imm20,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 をとおしてデータ転送を行います。

●命令発行

この命令がリソース競合を起こすことはありません。

●並列可能性

この命令は 32 ビット命令であり、並列実行できません（「8.3.5 32 ビット命令による競合の詳細」参照）。

(3) レジスタ-レジスタ間転送命令（MOV Rm,Rn、MOVI20、MOVI20S 以外）

●命令の種類

```
MOV         #imm,Rn
MOVA        @(disp,PC),R0
MOVT        Rn
MOVRT       Rn
SWAP.B      Rm,Rn
SWAP.W      Rm,Rn
XTRCT       Rm,Rn
NOTT
```

●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 をとおしてデータ転送を行います。

●命令発行

SWAP.B、SWAP.W、XTRCT 命令は、シフタを使用します。
それ以外の命令は、リソース競合を起こすことはありません。

●並列可能性

特記事項はありません。

(4) 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       @-Rm,R0
MOV.W       @-Rm,R0
MOV.L       @-Rm,R0
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

	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	MA	WB	
次命令	IF	ID	EX	…		
次々命令		IF	ID	EX	…	

●動作説明

パイプラインは、IF、ID、EX、MA、WB の 5 段です。この命令の 1~3 命令後に、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、競合が発生することがあります（**「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）。

●命令発行

この命令はメモリアクセスパイプラインを使用します。

●並列可能性

特記事項はありません。

(5) メモリロード命令（12 ビットディスプレースメント）

●命令の種類

```
MOV.B       @(disp12,Rm),Rn
MOV.W       @(disp12,Rm),Rn
MOV.L       @(disp12,Rm),Rn
MOVU.B      @(disp12,Rm),Rn
MOVU.W      @(disp12,Rm),Rn
```

●パイプライン

	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	MA	WB	
次命令		IF	ID	EX	…	
次々命令			IF	ID	EX	…

●動作説明

パイプラインは、IF、ID、EX、MA、WB の 5 段です。この命令の 1~2 命令後に、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、競合が発生することがあります。（**「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）。

●命令発行

この命令はメモリアクセスパイプラインを使用します。

●並列可能性

この命令は 32 ビット命令であり、並列実行できません（**「8.3.5 32 ビット命令による競合の詳細」**参照）。

(6) メモリロード命令（MOVMU.L、MOVML.L）

●命令の種類

```
MOVMU.L     @R15+,Rn
MOVML.L     @R15+,Rn
```

●パイプライン

	←→	←→	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	MA	…	MA	MA	MA	WB	
次命令	IF	--	--	--	…	ID	EX	…		
次々命令		IF	--	--	…	--	ID	EX	…	

●動作説明

この命令は、スタックからの復帰を行う命令です。パイプラインは、IF、ID、EX、MA、MA、MA、… MA、WB と必要なだけ MA を繰り返します。この命令の 1~3 命令後に、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、競合が発生することがあります。（「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）。

●命令発行

この命令がデコードされたときに、CPU パイプ内に未完了の命令があった場合、この命令は実行を待たされます。

この命令はメモリアクセスパイプラインを使用します。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません（「8.3.4 マルチサイクル命令による競合の詳細」参照）。

(7) 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,@Rn+
MOV.W       R0,@Rn+
MOV.L       R0,@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

	←→	←→	←→	←→	←→ スロット
命令 A	IF	ID	EX	MA	
次命令	IF	ID	EX	…	
次々命令		IF	ID	EX	…

●動作説明

パイプラインは、IF、ID、EX、MA の 4 段で終了します。レジスタへのデータの戻しがないので WB ステージはありません。

●命令発行

この命令はメモリアクセスパイプラインを使用します。

●並列可能性

特記事項はありません。

(8) メモリストア命令（12 ビットディスプレースメント）

●命令の種類

```
MOV.B       Rm,@(disp12,Rn)
MOV.W       Rm,@(disp12,Rn)
MOV.L       Rm,@(disp12,Rn)
```

●パイプライン

	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	MA		
次命令		IF	ID	EX	…	
次々命令			IF	ID	EX	…

●動作説明

パイプラインは、IF、ID、EX、MA の 4 段で終了します。レジスタへのデータの戻しがないので WB ステージはありません。

●命令発行

この命令はメモリアクセスパイプラインを使用します。

●並列可能性

この命令は 32 ビット命令であり、並列実行できません（「8.3.5 32 ビット命令による競合の詳細」参照）。

(9) メモリストア命令（MOVMU.L、MOVML.L）

●命令の種類

```
MOVMU.L Rm,@-R15
MOVML.L Rm,@-R15
```

●パイプライン

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	MA	…	MA	MA	MA	
次命令	IF	--	--	--	…	ID	EX	…	
次々命令		IF	--	--	…	--	ID	EX	…

●動作説明

この命令は、スタックへの退避を行う命令です。パイプラインは、IF、ID、EX、MA、MA、MA、… MA と必要なだけ MA を繰り返します。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。

●命令発行

この命令がデコードされたときに、CPU パイプ内に未完了の命令があった場合、この命令は実行を待たされます。

この命令はメモリアクセスパイプラインを使用します。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません。

(10) PREF 命令

●命令の種類

PREF　　@Rm

●パイプライン

	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	MA		
次命令	IF	ID	EX	…		
次々命令		IF	ID	EX	…	

●動作説明

パイプラインは、IF、ID、EX、MA の 4 段で終了します。レジスタへのデータの戻しがないので WB ステージはありません。

●命令発行

この命令はメモリアクセスパイプラインを使用します。

●並列可能性

特記事項はありません。

8.9.2 算術演算命令

(1) レジスタ間算術演算命令（乗算系、DIVU、DIVS 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
CLIPU.B     Rn
CLIPU.W     Rn
CLIPS.B     Rn
CLIPS.W     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 をとおしてデータ演算は完結します。

●命令発行

EXTS.B、EXTS.W、EXTU.B、EXTU.W 命令は、シフタを使用します。
それ以外の命令は、リソース競合を起こすことはありません。

●並列可能性

CLIP 命令同士において、CS ビットの上書きの競合は発生せず、並列実行できます。

(2) 積和命令

●命令の種類

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

●パイプライン

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	MA	MA	mm	mm		
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、MA、mm、mm の 7 段で終了します。mm は乗算器が動作している状態を表しています。

一般的なパイプラインの詳細は、**「8.7 乗算器による競合」**を参照してください。この命令は、実行スロット数 3、レイテンシ 5、ロックスロット数 4 の命令です。この命令のパイプラインに関し、乗算器に関する命令が連続したときの詳細の例を、以下に示します。

(a) MAC.W命令の直後に、MAC.W、MAC.L命令が連続してくる場合
乗算器は競合しません。

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
MAC.W @Rm+, @Rn+	IF	ID	EX	MA	MA	mm	mm		
MAC.W @Rm+, @Rn+	IF	--	--	ID	EX	MA	MA	mm	mm
次々命令		IF	--	--	--	ID	EX	…	

(b) MAC.W命令の直後に、MULS.W、MULU.W、DMULS.L、DMULU.L、MUL.L、MULR、 STS（レジスタ）、STS.L（メモリ）、LDS（レジスタ）命令が連続してくる場合
MAC.W命令が乗算器をロックするためさらに2スロット期間分後ろにストールされます。

	←→	←→	←→	←→	←→	←→	←→	←→ スロット
MAC.W @Rm+, @Rn+	IF	ID	EX	MA	MA	mm	mm	
STS MACL, Rn	IF	--	--	--	--	ID	EX	WB
次々命令		IF	--	--	--	ID	EX	…

(c) MAC.W命令の直後に、LDS.L（メモリ）命令がくる場合
MAC命令の実行ステート（3スロット）期間分実行を待たされます。

	←→	←→	←→	←→	←→	←→	←→	←→ スロット
MAC.W @Rm+, @Rn+	IF	ID	EX	MA	MA	mm	mm	
LDS.L @Rn+,MACL	IF	--	--	--	ID	EX	MA	WB
次々命令		IF	--	--	ID	EX	…	

●命令発行

この命令はメモリアクセスパイプラインを使用します。
この命令は乗算器を使用します。
この命令は乗算器がロックされていても実行されます。
この命令は乗算器を 4 スロット期間ロックします。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません（**「8.3.4 マルチサイクル命令による競合の詳細」**参照）。

(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
命令 A	IF	ID	EX	MA	MA	mm	mm	mm
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、MA、mm、mm、mm の 8 段で終了します。mm は乗算器が動作している状態を表しています。

一般的なパイプラインの詳細は、**「8.7 乗算器による競合」**を参照してください。この命令は、実行スロット数 4、レイテンシ 6、ロックスロット数 5 の命令です。この命令のパイプラインに関し、乗算器に関する命令が連続したときの詳細の例を、以下に示します。

(a) MAC.L命令の直後に、MAC.L、MAC.W命令が連続してくる場合
乗算器は競合しません。

	←→	←→	←→	←→	←→	←→	←→	←→	スロット		
MAC.L @Rm+, @Rn+	IF	ID	EX	MA	MA	mm	mm	mm			
MAC.L @Rm+, @Rn+	IF	--	--	--	ID	EX	MA	MA	mm	mm	mm
次々命令		IF	--	--	--	--	--	ID	EX	…	

(b) MAC.W命令の直後に、MULS.W、MULU.W、DMULS.L、DMULU.L、MUL.L、MULR、 STS（レジスタ）、STS.L（メモリ）、LDS（レジスタ）命令が連続してくる場合
MAC.L命令が乗算器をロックするため、さらに2ステート分後ろにストールされます。

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
MAC.L @Rm+, @Rn+	IF	ID	EX	MA	MA	mm	mm	mm	
STS MACH, Rn	IF	--	--	--	--	--	ID	EX	WB
次々命令		IF	--	--	--	--	ID	EX	…

(c) MAC.W命令の直後に、LDS.L（メモリ）命令がくる場合
MAC命令の実行ステート（4スロット）期間分実行を待たされます。

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
MAC.L @Rm+, @Rn+	IF	ID	EX	MA	MA	mm	mm	mm	
LDS.L @Rn+,MACL	IF	--	--	--	--	ID	EX	MA	WB
次々命令		IF	--	--	--	ID	EX	…	

●命令発行

この命令はメモリアクセスパイプラインを使用します。
この命令は乗算器を使用します。
この命令は乗算器がロックされていても実行されます。
この命令は乗算器を 5 スロット期間ロックします。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません（**「8.3.4 マルチサイクル命令による競合の詳細」**参照）。

(4) 乗算命令

●命令の種類

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

●パイプライン

	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	mm	mm		
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、mm、mm の 4 段で終了します。mm は乗算器が動作している状態を表しています。

一般的なパイプラインの詳細は、**「8.7 乗算器による競合」**を参照してください。この命令は、実行スロット数 1、レイテンシ 2、ロックスロット数 1 の命令です。この命令のパイプラインに関し、乗算器に関する命令が連続したときの詳細の例を、以下に示します。

(a) MULS.W命令の直後に、MAC.W、MAC.L命令が連続してくる場合
乗算器は競合しません。

	←→	←→	←→	←→	←→	←→	←→	←→ スロット
MULS.W Rm, Rn	IF	ID	mm	mm				
MAC.W @Rm+, @Rn+	IF	ID	EX	MA	MA	mm	mm	
次々命令		IF	--	ID	EX	…		

(b) MULS.W命令の直後に、MULS.W、MULU.W、DMULS.L、DMULU.L、MUL.L、MULR、 STS（レジスタ）、STS.L（メモリ）、LDS（レジスタ）命令が連続してくる場合
MULS.W命令が乗算器をロックするため、並列実行はできません。

	←→	←→	←→	←→	←→	←→ スロット
MULS.W Rm, Rn	IF	ID	mm	mm		
STS MACL,Rn	IF	--	ID	EX	WB	
次々命令		IF	ID	EX	…	

(c) MULS.W命令の直後に、LDS.L（メモリ）命令がくる場合
MULS.W命令が乗算器をロックするため、並列実行はできません。

	←→	←→	←→	←→	←→	←→ スロット
MULS.W Rm, Rn	IF	ID	mm	mm		
LDS.L @Rn+,MACL	IF	--	ID	EX	MA	WB
次々命令		IF	ID	EX	…	

●命令発行

この命令は乗算器を使用します
この命令は乗算器を 1 スロット期間ロックします。

●並列可能性

特記事項はありません。

(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

	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	mm	mm	mm	
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、mm、mm、mm の 5 段で終了します。mm は乗算器が動作している状態を表しています。

一般的なパイプラインの詳細は、**「8.7　乗算器による競合」**を参照してください。この命令は、実行スロット数 2、レイテンシ 3、ロックスロット数 2 の命令です。この命令のパイプラインに関し、乗算器に関する命令が連続したときの詳細の例を、以下に示します。

(a)　MUL.Lの直後に、MAC.W、MAC.L命令が連続してくる場合
乗算器は競合しません。

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
MUL.L Rm, Rn	IF	ID	mm	mm	mm				
MAC.L @Rm+, @Rn+	IF	--	ID	EX	MA	MA	mm	mm	mm
次々命令		IF	--	--	--	ID	EX	…	

(b)　MUL.L命令の直後に、MULS.W、MULU.W、DMULS.L、DMULU.L、MUL.L、MULR、 STS（レジスタ）、STS.L（メモリ）、LDS（レジスタ）命令が連続してくる場合
MUL.L命令が乗算器をロックするため、さらに2スロット期間分後ろにストールされます。

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
MUL.L Rm, Rn	IF	ID	mm	mm	mm				
STS MACL,Rn	IF	--	--	ID	EX	WB			
次々命令		IF	--	ID	EX	…			

(c)　MUL.L命令の直後に、LDS.L（メモリ）命令がくる場合
MUL.L命令の実行ステート（2サイクル）期間分実行を待たされます。

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
MUL.L Rm, Rn	IF	ID	mm	mm	mm				
LDS.L @Rn+,MACL	IF	--	--	ID	EX	MA	WB		
次々命令		IF	--	ID	EX	…			

●命令発行

この命令は乗算器を使用します。
この命令は乗算器を 2 スロット期間ロックします。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません（**「8.3.4　マルチサイクル命令による競合の詳細」**参照）。

(6) 倍精度の乗算命令（一般レジスタ戻し）

●命令の種類

```
MULR        R0,Rn
```

●パイプライン

	←→	←→	←→	←→	←→	←→	←→ スロット
命令 A	IF	ID	mm	mm	mm	WB	
次命令	IF	--	ID	EX	…		
次々命令		IF	ID	EX	…		

●動作説明

パイプラインは、IF、ID、mm、mm、mm、WB の 6 段で終了します。mm は乗算器が動作している状態を表しています。

一般的なパイプラインの詳細は、**「8.7 乗算器による競合」**を参照してください。この命令は、実行スロット数 2、レイテンシ 4、ロックスロット数の命令です。この命令のパイプラインに関し、乗算器に関する命令が連続したときの詳細を、以下に示します。

(a) MULR命令の直後に、MAC.W、MAC.L命令が連続してくる場合
乗算器は競合しません。

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
MULR R0, Rn	IF	ID	mm	mm	mm	WB			
MAC.L @Rm+, @Rn+	IF	--	ID	EX	MA	MA	mm	mm	mm
次々命令		IF	--	--	--	ID	EX	…	

(b) MULR命令の直後に、命令の直後に、MULS.W、MULU.W、DMULS.L、DMULU.L、MUL.L、MULR、STS（レジスタ）、STS.L（メモリ）、LDS（レジスタ）命令が連続してくる場合
MULR命令が乗算器をロックするため、さらに1スロット期間分後ろにストールされます。

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
MULR R0, Rn	IF	ID	mm	mm	mm	WB			
MULR R0, Rn	IF	--	--	ID	mm	mm	mm	WB	
次々命令		IF	--	ID	EX	…			

(c) MULR命令の直後に、STS（レジスタ）、STS.L（メモリ）命令が連続してくる場合
MULR命令が乗算器をロックし、また乗算結果読み出しバスの競合を起こすため、さらに2スロット期間分後ろにストールされます。

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
MULR R0, Rn	IF	ID	mm	mm	mm	WB			
STS MACL,Rn	IF	--	--	--	ID	EX	WB		
次々命令		IF	--	--	ID	EX	…		

(d) MULR命令の直後に、LDS.L（メモリ）命令がくる場合
MULR命令の実行ステート（2スロット）期間分実行を待たされます。

	←→	←→	←→	←→	←→	←→	←→	←→ スロット
MULR R0, Rn	IF	ID	mm	mm	mm	WB		
LDS.L @Rn+,MACL	IF	--	--	ID	EX	MA	WB	
次々命令		IF	--	ID	EX	…		

●命令発行

この命令は乗算器を使用します。
この命令は乗算器を 2 スロット期間ロックします。
この命令は乗算結果の読み出しバスを使用します。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません（「8.3.4 マルチサイクル命令による競合の詳細」参照）。

(7) DIVU 命令

●命令の種類

```
DIVU R0,Rn
```

●パイプライン

	←→	←→	←→	←→	←→	←→	←→ スロット
命令 A	IF	ID	EX	…	EX	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、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、の 36 段で終了します。EX ステージで、ALU をとおしてデータ演算は完結します。

●命令発行

この命令は、シフトパイプラインを使用します。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません。

(8) DIVS 命令

●命令の種類

```
DIVS R0,Rn
```

●パイプライン

	←→	←→	←→	←→	←→	←→	←→ スロット
命令 A	IF	ID	EX	…	EX	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、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX、EX の 38 段で終了します。EX ステージで、ALU をとおしてデータ演算は完結します。

●命令発行

この命令がリソース競合を起こすことはありません。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません。

8.9.3　　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

	←→	←→	←→	←→	←→	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 をとおしてデータ演算は完結します。

●命令発行

この命令がリソース競合を起こすことはありません。

●並列可能性

特記事項はありません。

(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)
XOR.B       #imm,@(R0,GBR)
```

●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。

●命令発行

この命令はメモリアクセスパイプラインを使用します。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません（**「8.3.4　マルチサイクル命令による競合の詳細」**参照）。

(3)　メモリ論理演算命令

●命令の種類

```
TST.B       #imm,@(R0,GBR)
```

●パイプライン

	←→	←→	←→	←→	←→	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。

●命令発行

この命令はメモリアクセスパイプラインを使用します。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません（**「8.3.4　マルチサイクル命令による競合の詳細」**参照）。

(4) 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。

●命令発行

この命令はメモリアクセスパイプラインを使用します。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません。

(5) レジスターレジスタ間ビット演算命令

●命令の種類

```
BLD         #imm3,Rn
BSET        #imm3,Rn
BCLR        #imm3,Rn
BST         #imm3,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 をとおしてデータ演算は完結します。

●命令発行

この命令がリソース競合を起こすことはありません。

●並列可能性

特記事項はありません。

(6) メモリーT ビット間ビット演算命令

●命令の種類

```
BAND.B      #imm3,@(disp12,Rn)
BANDNOT.B   #imm3,@(disp12,Rn)
BLD.B       #imm3,@(disp12,Rn)
BLDNOT.B    #imm3,@(disp12,Rn)
BOR.B       #imm3,@(disp12,Rn)
BORNOT.B    #imm3,@(disp12,Rn)
BXOR.B      #imm3,@(disp12,Rn)
```

●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。

●命令発行

この命令はメモリアクセスパイプラインを使用します。

●並列可能性

この命令は 32 ビット命令であり、並列実行できません。この命令の後続命令が、BAND.B、BANDNOT.B、BLD.B、BLDNOT.B、BOR.B、BORNOT.B、BXOR.B 命令のうちのいずれかの場合、最終ステップは後続命令と並列実行できます。これらの命令ではない場合、最終ステップは後続命令と並列実行することはできません（「8.3.5　32 ビット命令による競合の詳細」参照）。

	←→	←→	←→	←→	←→	←→		スロット
BAND.B #imm, @(disp12, Rn)	IF	ID	EX	MA	EX			
BOR.B　#imm, @(disp12, Rn)		IF	--	ID	EX	…		
BANDNOT.B #imm, @(disp12, Rn)			IF	--	--	ID	EX	…

	←→	←→	←→	←→	←→	←→	スロット
BAND.B #imm, @(disp12, Rn)	IF	ID	EX	MA	EX		
ADD Rm, Rn		IF	--	--	ID	EX	…
次々命令		IF	--	--	ID	EX	…

	←→	←→	←→	←→	←→	←→	スロット
BAND.B #imm, @(disp12, Rn)	IF	ID	EX	MA	EX		
ROTCL		IF	--	--	ID	EX	
BAND.B #imm, @(disp12, Rn)			IF	--	--	ID	EX
次々命令			IF	--	--	--	--

(7) メモリビット操作命令

●命令の種類

```
BCLR.B      #imm3,@(disp12,Rn)
BSET.B      #imm3,@(disp12,Rn)
BST.B       #imm3,@(disp12,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。

●命令発行

この命令はメモリアクセスパイプラインを使用します。

●並列可能性

この命令は 32 ビット命令であり、並列実行できません（「8.3.5 32 ビット命令による競合の詳細」参照）。

8.9.4 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
SHAD        Rm,Rn
SHLD        Rm,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、シフタをとおしてデータ演算は完結します。

●命令発行

この命令はシフトパイプラインを使用します。

●並列可能性

特記事項はありません。

8.9.5　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

	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX			
次命令	IF		…	(フェッチするが捨てられる)		
次々命令		IF	…	(フェッチするが捨てられる)		
次々々命令		IF	…	(フェッチするが捨てられる)		
分岐先命令			--	IF	ID	EX …

(b)　条件が成立しないとき

	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX			
次命令	IF	ID	EX			
次々命令		IF	ID	EX	…	
次々々命令		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。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。それまでにオーバランフェッチされていた命

令はすべて捨てられます。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。

以下に、典型的なパイプラインを示します。

先行命令がCMP命令だった場合、1サイクル実行を待たされます。

	←→	←→	←→	←→	←→	スロット
CMP	IF	ID	EX			
BF	IF	--	ID	EX		
分岐先					IF	

先行命令が単精度FCMPだった場合、2サイクル実行を待たされます。

	←→	←→	←→	←→	←→	←→	スロット
FCMP/single	IF	DF	E1	E2			
BF	IF	--	--	ID	EX		
分岐先						IF	

先行命令が倍精度FCMPだった場合、3サイクル実行を待たされます。

	←→	←→	←→	←→	←→	←→	←→	スロット
FCMP/double	IF	DF	E1	E1	E2			
BF	IF	--	--	--	ID	EX		
分岐先							IF	

●命令発行

この命令は分岐パイプラインを使用します。

●並列可能性

特記事項はありません。

(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

	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX			
遅延スロット	IF	--	--	ID	EX	…
次命令		IF	…	(フェッチするが捨てられる)		
次々命令		IF	…	(フェッチするが捨てられる)		
分岐先命令			--	IF	ID	EX …

(b) 条件が成立しないとき

	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX			
次命令	IF	ID	EX			
次々命令		IF	ID	EX	…	
次々々命令		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。ID ステージで条件判断を行います。
遅延スロットでは、割り込みを受け付けません。

(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。それまでにオーバランフェッチされていた命令はすべて捨てられます。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。
以下に、典型的なパイプラインを示します。

先行命令がCMP命令だった場合、1サイクル実行を待たされます。

	←→	←→	←→	←→	←→	スロット
CMP	IF	ID	EX			
BF/S	IF	--	ID	EX		
遅延スロット		IF	--	--	ID	

先行命令が単精度FCMPだった場合、2サイクル実行を待たされます。

	←→	←→	←→	←→	←→	←→	スロット
FCMP/single	IF	DF	E1	E2			
BF/S	IF	--	--	ID	EX		
遅延スロット		IF	--	--	--	ID	

先行命令が倍精度FCMPだった場合、3サイクル実行を待たされます。

	←→	←→	←→	←→	←→	←→	←→	スロット
FCMP/double	IF	DF	E1	E1	E2			
BF/S	IF	--	--	--	ID	EX		
遅延スロット		IF	--	--	--	--	ID	

●命令発行

この命令は分岐パイプラインを使用します。
この命令の直後の命令（遅延スロット）に対していまだ命令フェッチを発行していないとき
その命令は実行を待たされます。

●並列可能性

特記事項はありません。

(3) 無条件分岐命令

●命令の種類

```
BRA     label
BRAF    Rm
BSR     label
BSRF    Rm
JMP     @Rm
JSR     @Rm
RTS
```

●パイプライン

	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX			
遅延スロット	IF	--	--	ID	EX	…
次々命令		IF	…	(フェッチするが捨てられる)		
次々々命令		IF	…	(フェッチするが捨てられる)		
分岐先命令			--	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。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 2 スロット期間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。

遅延スロットでは、割り込みを受け付けません。

●命令発行

この命令は分岐パイプラインを使用します。

この命令の直後の命令（遅延スロット）に対していまだ命令フェッチを発行していないとき

この命令は実行を待たされます。

●並列可能性

特記事項はありません。

(4) 遅延なし無条件分岐命令

●命令の種類

```
JSR/N   @Rm
RTS/N
RTV/N   Rm
```

●パイプライン

	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX			
次命令	IF	--	…	(フェッチするが捨てられる)		
次々命令		IF	…	(フェッチするが捨てられる)		
次々々命令		IF	…	(フェッチするが捨てられる)		
分岐先命令			--	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。ID ステージで条件判断を行います。この分岐命令は遅延分岐ではありません。EX ステージで、分岐先アドレスを計算します。それまでにオーバランフェッチされていた命令はすべて捨てられます。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。

●命令発行

この命令は分岐パイプラインを使用します。

●並列可能性

特記事項はありません。

(5)　無条件遅延なし分岐命令（JSR/N @@（disp,TBR））

●命令の種類

JSR/N　　@@（disp,TBR）

●パイプライン

	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	MA	EX	
次命令	IF	--	…	(フェッチするが捨てられる)		
次々命令		IF	…	(フェッチするが捨てられる)		
次々々命令		IF	…	(フェッチするが捨てられる)		
分岐先命令			--	--	--	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。ID ステージで条件判断を行います。この分岐命令は遅延分岐ではありません。2 番目の EX ステージで、分岐先アドレスを計算します。それまでにオーバランフェッチされていた命令はすべて捨てられます。分岐先命令は、命令 A の 2 番目の 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。

●命令発行

この命令は分岐パイプラインを使用します。
この命令はメモリアクセスパイプラインを使用します。

●並列可能性

特記事項はありません。

8.9.6 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,GBR
LDC     Rm,TBR
LDC     Rm,VBR
LDS     Rm,PR
NOP
SETT
STC     GBR,Rn
STC     TBR,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 をとおしてデータ演算は完結します。

●命令発行

この命令がリソース競合を起こすことはありません。

●並列可能性

特記事項はありません。

(2) システム制御 ALU 命令

●命令の種類

```
LDC     Rm,SR
```

●パイプライン

	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	EX	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、EX、EX の 5 段で終了します。1 つめの EX ステージで、ALU をとおしてデータ演算は完結します。

●命令発行

この命令がリソース競合を起こすことはありません。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません。

(3) システム制御 ALU 命令

●命令の種類

```
STC     SR,Rn
```

●パイプライン

	←→	←→	←→	←→	←→	←→	←→ スロット
命令 A	IF	ID	EX	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、ＥＸの 4 段で終了します。2 つめの EX ステージで、ALU をとおしてデータ演算は完結します。

●命令発行

特記事項はありません。

典型的な、CS ビット読み出し時のパイプラインを示しておきます。

	←→	←→	←→	←→	←→	←→	←→ スロット
CLIP	IF	ID	EX				
STC	IF	--	ID	EX	EX		
次命令		IF	--	ID	EX	…	
次々命令		IF	--	ID	EX	…	

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません。

(4) LDC.L、LDS.L 命令

●命令の種類

```
LDC.L     @Rm+,GBR
LDC.L     @Rm+,VBR
LDS.L     @Rm+,PR
```

●パイプライン

	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	MA	WB			
次命令	IF	ID	EX	…				
次々命令		IF	ID	EX	…			

●動作説明

パイプラインは、IF、ID、EX、MA、WB の 5 段で終了します。

●命令発行

この命令はメモリアクセスパイプラインを使用します。

●並列可能性

特記事項はありません。

(5) LDC.L 命令

●命令の種類

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

●パイプライン

	←→	←→	←→	←→	←→	←→	←→	スロット	
命令 A	IF	ID	EX	MA	EX	EX	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、EX、EX の 7 段で終了します。

●命令発行

この命令はメモリアクセスパイプラインを使用します。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません。

(6) STC.L 命令

●命令の種類

```
STC.L       GBR,@-Rn
STC.L       VBR,@-Rn
STS.L       PR,@-Rn
```

●パイプライン

	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	MA				
次命令	IF	ID	EX	…				
次々命令		IF	ID	EX	…			

●動作説明

パイプラインは、IF、ID、EX、MA の 4 段で終了します。

●命令発行

この命令はメモリアクセスパイプラインを使用します。

●並列可能性

特記事項はありません。

(7) STC.L 命令

●命令の種類

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

●パイプライン

	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	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、EX、MA の 5 段で終了します。

●命令発行

この命令はメモリアクセスパイプラインを使用します。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません。

また、この命令はメモリアクセスパイプラインを使用しますが、シングルサイクルのメモリアクセス命令が先行命令だった場合、並列実行することができます。

(8) 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

	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	mm	mm				
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、mm、mm の 4 段で終了します。mm は乗算器が動作している状態を表しています。

一般的なパイプラインの詳細は、**「8.7 乗算器による競合」**を参照してください。この命令は、実行スロット数 1、レイテンシ 2、ロックスロット数 1 の命令です。この命令のパイプラインに関し、乗算器に関する命令が連続したときの詳細の例を、以下に示します。

(a) CLRMAC命令の直後に、MAC.W、MAC.L命令が連続してくる場合
乗算器は競合しません。

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
CLRMAC	IF	ID	mm	mm					
MAC.W @Rm+, @Rn+	IF	ID	EX	MA	MA	mm	mm		
次々命令		IF	--	--	ID	EX	…		

(b) CLRMAC命令の直後に、MULS.W、MULU.W、DMULS.L、DMULU.L、MUL.L、MULR、STS（レジスタ）、STS.L（メモリ）、LDS（レジスタ）命令が連続してくる場合
CLRMAC命令が乗算器をロックするため、並列実行はできません。

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
CLRMAC	IF	ID	mm	mm					
STS MACL,Rn	IF	--	ID	EX	WB				
次々命令		IF	ID	EX	…				

(c) CLRMAC命令の直後に、LDS.L（メモリ）命令がくる場合
CLRMAC命令の実行ステート（1スロット）期間分実行を待たされます。

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
CLRMAC	IF	ID	mm	mm					
LDS.L @Rn+,MACL	IF	--	ID	EX	MA	WB			
次々命令		IF	ID	EX	…				

●命令発行

この命令は乗算器を使用します。
この命令は乗算器を 1 スロット期間ロックします。

●並列可能性

特記事項はありません。

(9) 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

	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	MA	WB	
次命令	IF	ID	EX	…		
次々命令		IF	ID	EX	…	

●動作説明

パイプラインは、IF、ID、EX、MA、WBの5段で終了します。

一般的なパイプラインの詳細は、**「8.7 乗算器による競合」**を参照してください。この命令は、実行スロット数1、レイテンシ3、ロックスロット数2の命令です。この命令のパイプラインに関し、乗算器に関する命令が連続したときの詳細を、以下に示します。

(a) LDS.L命令の直後に、MAC.W、MAC.L命令が連続してくる場合
乗算器は競合しませんが、メモリアクセスが競合し、1サイクルストールします。

	←→	←→	←→	←→	←→	←→	←→	スロット
LDS.L @Rm+, MACH	IF	ID	EX	MA	WB			
MAC.W @Rm+, @Rn+	IF	--	ID	EX	MA	MA	mm	mm
次々命令		IF	--	--	ID	EX	…	

(b) LDS.L命令の直後に、MULS.W、MULU.W、DMULS.L、DMULU.L、MUL.L、MULR、 STS（レジスタ）、STS.L（メモリ）、LDS（レジスタ）命令が連続してくる場合
LDS.L命令が乗算器をロックするため、さらに1スロット期間分後ろにストールされます。

	←→	←→	←→	←→	←→	スロット
LDS.L @Rm+, MACH	IF	ID	EX	MA	WB	
STS MACL,Rn	IF	--	--	ID	EX	WB
次々命令		IF	--	ID	EX	…

(c) LDS.L命令の直後に、LDS.L（メモリ）命令がくる場合
LDS.L命令の実行ステート（1スロット）期間分実行を待たされます。

	←→	←→	←→	←→	←→	スロット
LDS.L @Rn+, MACH	IF	ID	EX	MA	WB	
LDS.L @Rn+,MACL	IF	--	ID	EX	MA	WB
次々命令		IF	ID	EX	…	

●命令発行

この命令はメモリアクセスパイプラインを使用します。
この命令は乗算器を使用します。
この命令は乗算器のロック期間が残り1であれば実行されます。
この命令は乗算器を2スロット期間ロックします。

●並列可能性

特記事項はありません。

(10) 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

	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	WB		
次命令	IF	ID	EX	…		
次々命令		IF	ID	EX	…	

●動作説明

パイプラインは、IF、ID、EX、WB の 4 段で終了します。

一般的なパイプラインの詳細は、**「8.7 乗算器による競合」**を参照してください。この命令は、実行スロット数 1、レイテンシ 2、ロックスロット数 0 の命令です。この命令のパイプラインに関し、乗算器に関する命令が連続したときの詳細を、以下に示します。

(a) STS命令の直後に、MAC.W、MAC.L命令が連続してくる場合
乗算器は競合しません。

	←→	←→	←→	←→	←→	←→	←→	スロット
STS MACH, Rn	IF	ID	EX	WB				
MAC.W @Rm+, @Rn+	IF	ID	EX	MA	MA	mm	mm	
次々命令		IF	--	ID	EX	…		

(b) STS命令の直後に、MULS.W、MULU.W、DMULS.L、DMULU.L、MUL.L、MULR、 STS（レジスタ）、STS.L（メモリ）、LDS（レジスタ）命令が連続してくる場合
STS命令が乗算器をロックしないため、並列実行されます。

	←→	←→	←→	←→	←→	←→	←→	スロット
STS MACH, Rn	IF	ID	EX	WB				
MUL.L Rm, Rn	IF	ID	mm	mm	mm			
次々命令		IF	ID	EX	…			

(c) STS命令の直後に、STS（レジスタ）、STS.L（メモリ）命令が連続してくる場合。
乗算結果の読み出しバスが競合するため、並列実行はできません。

	←→	←→	←→	←→	←→	←→	←→	スロット
STS MACH, Rn	IF	ID	EX	WB				
STS MACL, Rn	IF	--	ID	EX	WB			
次々命令		IF	ID	EX	…			

(d) STS命令の直後に、LDS.L（メモリ）命令がくる場合
並列実行されます。

	←→	←→	←→	←→	←→	←→	←→	スロット
STS MACH, Rn	IF	ID	EX	WB				
LDS.L @Rn+,MACL	IF	ID	EX	MA	WB			
次々命令		IF	ID	EX	…			

●命令発行

この命令は乗算器を使用しますが、ロックしません。
この命令は、乗算結果の読み出しバスを使用します。

●並列可能性

特記事項はありません。

(11) 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

	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	MA		
次命令	IF	ID	EX	…		
次々命令		IF	ID	EX	…	

●動作説明

パイプラインは、IF、ID、EX、MA の 4 段で終了します。
一般的なパイプラインの詳細は、**「8.7 乗算器による競合」**を参照してください。この命令は、実行スロット数 1、レイテンシ 2、ロックスロット数 0 の命令です。この命令のパイプラインに関し、乗算器に関する命令が連続したときの詳細を、以下に示します。

(a) STS.L命令の直後に、MAC.W、MAC.L命令が連続してくる場合
乗算器は競合しませんが、メモリアクセスが競合し、1サイクルストールします。

	←→	←→	←→	←→	←→	←→	←→	スロット
STS.L MACH, @-Rn	IF	ID	EX	MA				
MAC.W @Rm+, @Rn+	IF	--	ID	EX	MA	MA	mm	mm
次々命令		IF	--	--	ID	EX	…	

(b) STS.L命令の直後に、MULS.W、MULU.W、DMULS.L、DMULU.L、MUL.L、MULR、 STS（レジスタ）、STS.L（メモリ）、LDS（レジスタ）命令が連続してくる場合
STS.L命令が乗算器をロックしないため、並列実行されます。

	←→	←→	←→	←→	←→	←→	←→	スロット
STS.L MACL, @-Rn	IF	ID	EX	MA				
MUL.L Rm, Rn	IF	ID	mm	mm				
次々命令		IF	ID	EX	…			

(c) STS.L命令の直後に、STS（レジスタ）、STS.L（メモリ）が連続してくる場合。
乗算結果の読み出しバスが競合するため、並列実行はできません。

	←→	←→	←→	←→	←→	←→	←→	スロット
STS.L MACH, @-Rn	IF	ID	EX	MA				
STS.L MACL, @-Rn	IF	--	ID	EX	MA			
次々命令		IF	ID	EX	…			

(d) STS.L命令の直後に、LDS.L（メモリ）命令がくる場合
メモリアクセスパイプラインの競合が発生し、並列実行はできません。

	←→	←→	←→	←→	←→	←→	←→	スロット
STS.L MACH, @-Rn	IF	ID	EX	MA				
LDS.L @Rn+,MACL	IF	--	ID	EX	MA	WB		
次々命令		IF	ID	EX	…			

●命令発行

この命令はメモリアクセスパイプラインを使用します。
この命令は乗算器を使用しますが、ロックしません。
この命令は乗算結果の読み出しバスを使用します。

●並列可能性

特記事項はありません。

(12) RTE 命令

●命令の種類

RTE

●パイプライン

	←→	←→	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	MA	MA	EX	EX	EX		
遅延スロット	IF	--	--	--	--	--	ID	EX	…	
分岐先						IF	--	ID	EX	…

●動作説明

パイプラインは、IF、ID、EX、MA、MA、EX、EX、EX の 8 段で終了します。RTE は遅延分岐命令です。遅延スロット命令の ID は 5 スロット期間分ストールします。分岐先命令の IF は RTE の 2 回目の MA の次のスロットから開始されます。

●命令発行

この命令がリソース競合を起こすことはありません。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません。

(13) RESBANK 命令

●命令の種類

RESBANK

●パイプライン

○BO==0 のとき

	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	EX	EX	EX	EX	EX	EX	EX	EX	
次命令	IF	--	--	--	--	--	--	--	--	ID	EX	…

○BO==1 のとき

	←→	←→	←→	←→	←→	←→	…	←→	←→	←→	スロット
命令 A	IF	ID	EX	MA	MA	MA	…	MA	MA	MA	WB
次命令	IF	--	--	--	--	--	…	ID	EX	…	

●動作説明

BO ビット＝0 のときと 1 のときで動作が異なります。

BO ビット＝0 のとき、バンクからの復帰を行います。パイプラインは、IF、ID、EX、EX、EX、EX、EX、EX、EX、EX、EX と EX を 9 回繰り返し、11 段で終了します。この間に、バンクからレジスタが復帰されます。

BO ビット＝1 のとき、スタックからの復帰を行います。パイプラインは、IF、ID、EX、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、WB と MA を 19 回繰り返し、23 段で終了します。

●命令発行

BO ビット＝0 のとき、この命令がリソース競合を起こすことはありません。

BO ビット＝1 のとき、この命令はメモリアクセスパイプラインを使用します。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません（**「8.3.4　マルチサイクル命令による競合の詳細」**参照）。

(14) LDBANK 命令

●命令の種類

```
LDBANK @Rm,R0
```

●パイプライン

	←→	←→	←→	←→	←→	←→	…	←→	スロット	
命令 A	IF	ID	EX	EX	EX	EX	EX	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、EX、EX、EX、EX、EX の 8 段で終了します。

●命令発行

この命令が、リソース競合をおこすことはありません。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません。（「**8.3.4 マルチサイクル命令による競合の詳細**」参照）。

(15) STBANK 命令

●命令の種類

```
STBANK R0,@Rn
```

●パイプライン

	←→	←→	←→	←→	←→	←→	←→	←→	←→	スロット	
命令 A	IF	ID	EX	EX	EX	EX	EX	EX	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、EX、EX、EX、EX、EX、EX の 9 段で終了します。

●命令発行

この命令が、リソース競合を起こすことはありません。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません。（「**8.3.4 マルチサイクル命令による競合の詳細**」参照）。

(16) TRAP 命令

●命令の種類

```
TRAPA       #imm
```

●パイプライン

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	EX	EX	MA	MA	MA	
次命令	IF	--	…						
次々命令		IF	--	…					
分岐先								IF	

●動作説明

パイプラインは、IF、ID、EX、EX、EX、MA、MA、MA の 8 段で終了します。TRAP 命令は遅延分岐命令ではありません。分岐先命令の IF は TRAP 命令の 3 番目の MA があるスロットから IF を開始します。

●命令発行

この命令はメモリアクセスパイプラインを使用します。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません。

(17) SLEEP 命令

●命令の種類

SLEEP

●パイプライン

	←→	←→	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	MA	EX	EX	EX			
次命令	IF	--	…							
次々命令		IF	--	…						

●動作説明

パイプラインは、IF、ID、EX、MA、EX、EX、EX、の 7 段で終了します。
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。

●命令発行

この命令はメモリアクセスパイプラインを使用します。

●並列可能性

この命令はマルチサイクル命令であり、後続命令とは並列実行できません。

8.9.7 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

○バンキングなしのとき

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
割り込み	IF	ID	EX	EX	MA	MA	MA		
次命令	IF	…							
次々命令		IF	…						
分岐先							IF	ID	

○バンキングありかつオーバフローなしのとき

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
割り込み	IF	ID	EX	EX	MA	MA	MA	EX	
次命令	IF	…							
分岐先							IF	ID	

○バンキングありかつオーバフローのとき

	←→	←→	←→	←→	←→	←→	←→	←→	←→	スロット
割り込み	IF	ID	EX	EX	MA	MA	MA	…	MA	
次命令	IF	…								
分岐先							IF			… ID

●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。

割り込み処理は、バンキングなし、バンキングあり、バンキングありかつオーバフローのときで動作が異なります。

バンキングなしのとき、パイプラインは IF、ID、EX、EX、MA、MA、MA の 7 段で終了します。

バンキングありかつオーバーフローなしのとき、バンクへの退避動作を自動的に行います。パイプラインは、IF、ID、EX、EX、MA、MA、MA、EX の 8 段となります。

バンキングありかつオーバーフローのときは、バンクへ退避するレジスタをスタックへと自動退避し、BO ビットを 1 にします。このとき、パイプラインは IF、ID、EX、EX、MA、MA、MA、EX、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA、MA と EX を 2 回、MA を 3 回、EX を 1 回、MA を 19 回繰り返し、27 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。分岐先命令は割り込み例外処理の 3 番目の MA 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。

●割り込み受け付け

割り込み例外処理は、遅延スロットでは受け付けられません。

現在実行中のマルチサイクル命令があるとき、割り込み例外処理は受け付けられず、その命令の実行が完了してから受け付けられます。ただし、DIVU,DIVS 命令は実行中にキャンセルして割り込みを受け付けることができます。

(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

	←→	←→	←→	←→	←→	←→	←→	スロット
アドレスエラー例外処理	IF	ID	EX	EX	MA	MA	MA	
次命令	IF	…						
次々命令		IF	…					
分岐先							IF	ID

●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、MA の 7 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。分岐先命令はアドレスエラー例外処理の最後の MA 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。

●アドレス例外処理受け付け

アドレス例外処理は、遅延スロットでは受け付けられません。

現在実行中のマルチサイクル命令があるとき、アドレス例外処理は受け付けられず、その命令の実行が完了してから受け付けられます。ただし、DIVU,DIVS 命令は実行中にキャンセルしてアドレス例外処理を受け付けることができます。

(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

	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	スロット
不当命令	IF	ID	EX	EX	MA	MA	MA				
次命令	IF	--	…								
次々命令		IF	--	…							
分岐先							IF	ID			

●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 の 7 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（遅延スロットとよぶ）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および 32 ビット命令および RESBANK 命令および、DIVU、DIVS 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。

分岐先命令の IF は不当命令例外処理の最後の MA があるスロットから IF を開始します。

(4) FPU 例外処理

●命令の種類

FPU 例外処理

●パイプライン

	←→	←→	←→	←→	←→	←→	←→	スロット
FPU 例外処理	IF	ID	EX	EX	MA	MA	MA	
次命令	IF	…						
次々命令		IF	…					
分岐先							IF	ID

●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、MA の 7 段で終了します。FPU 例外処理は遅延分岐ではありません。分岐先命令は FPU 例外処理の最後の MA があるスロットから IF を開始します。

●FPU 例外発生～FPU 例外受け付けまでにパイプライン処理された命令について

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（FCMP 命令を除く）を NOP 化します。したがって、この区間の命令による FPU のレジスタの更新はありません。

FPU に関する CPU 命令は、上記同様に FPU のレジスタは更新しません（NOP 化します）が、CPU のレジスタは更新します。

CPU 命令は NOP 化されず、通常どおりに動作します。

8.9.8 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 A	IF	ID	EX	MA				: CPU パイプライン
	IF	DF	EX	NA	SF			: FPU パイプライン
次命令	IF	ID	EX	…				: CPU パイプライン
	IF	DF	…					: FPU パイプライン
次々命令		IF	ID	EX	…			: CPU パイプライン
		IF	DF	…				: 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、MA の 4 段で、FPU パイプラインが IF、DF、EX、NA、SF の 5 段で終了します。この命令の 1~3 命令後に FPUL をリードする命令を置くと競合が発生することがあります。

●命令発行

この命令は FPU ロードストアパイプラインとメモリアクセスパイプラインを使用します。
ただし、LDS 命令は CPU メモリリード命令と競合しません。

●並列可能性

特記事項はありません。

(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 A	IF	ID	EX	MA				: CPU パイプライン
	IF	DF	EX	NA	SF			: FPU パイプライン
次命令	IF	--	ID	EX	…			: CPU パイプライン
	IF	--	DF	…				: FPU パイプライン
次々命令		IF	ID	EX	…			: CPU パイプライン
		IF	DF	…				: 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、 MA の 4 段で、FPU パイプラインが IF、DF、EX、NA、SF の 5 段で終了します。後続の FPU に関係のある命令は、以降 3 サイクルストールします。

●命令発行

この命令は FPU ロードストアパイプラインを使用します。
また、LDS.L 命令はメモリアクセスパイプラインも使用します。
まだ計算中の FPU 算術演算命令があるとき、終了するまで待たされます。

●並列可能性

この命令は、FPU 命令および FPU に関する CPU 命令とは並列実行できません。

(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 A	IF	ID	EX	WB				: CPU パイプライン
	IF	DF	EX	NA				: FPU パイプライン
次命令	IF	ID	EX	…				: CPU パイプライン
	IF	DF	…					: FPU パイプライン
次々命令		IF	ID	EX	…			: CPU パイプライン
		IF	DF	…				: 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、WB の 4 段で、FPU パイプラインが IF、DF、EX、NA の 4 段で終了します。この命令の 1~3 命令後に、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することがあります。

●命令発行

この命令は乗算結果の読み出しバスを使用します。
この命令は FPU ロードストアパイプラインを使用します。
FPUL が FPU 算術演算の結果待ちとなっている場合、先の命令のレイテンシは 2 減少します。
詳細は「8.6 FPU による競合」を参照してください。

●並列可能性

特記事項はありません。

(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 A	IF	ID	EX	MA				: CPU パイプライン
	IF	DF	EX	NA				: FPU パイプライン
次命令	IF	ID	EX	…				: CPU パイプライン
	IF	DF	…					: FPU パイプライン
次々命令		IF	ID	EX	…			: CPU パイプライン
		IF	DF	…				: 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、MA の 4 段で、FPU パイプラインが IF、DF、EX、NA の 4 段で終了します。

●命令発行

この命令は FPU ロードストアパイプラインとメモリアクセスパイプラインを使用します。
FPUL が FPU 算術演算の結果待ちとなっている場合、先の命令のレイテンシは 1 減少します。
詳細は「8.6　FPU による競合」を参照してください。

●並列可能性

特記事項はありません。

(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 A	IF	ID	EX	WB				: CPU パイプライン
	IF	DF	EX	NA				: FPU パイプライン
次命令	IF	--	ID	EX	…			: CPU パイプライン
	IF	--	DF	…				: FPU パイプライン
次々命令		IF	ID	EX				: CPU パイプライン
		IF	DF	…				: 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、WB の 4 段で、FPU パイプラインが IF、DF、EX、NA の 4 段で終了します。
この命令の 1~3 命令後に、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することがあります。

●命令発行

この命令は乗算結果の読み出しバスを使用します。
この命令は FPU ロードストアパイプラインとメモリアクセスパイプラインを使用します。
まだ計算中の FPU 算術演算命令があるとき、終了するまで待たされます。

●並列可能性

この命令は、FPU 命令および FPU に関する CPU 命令とは並列実行できません。

(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 A	IF	ID	EX	MA			: CPU パイプライン
	IF	DF	EX	NA			: FPU パイプライン
次命令	IF	--	ID	EX	…		: CPU パイプライン
	IF	--	DF	…			: FPU パイプライン
次々命令		IF	ID	EX	…		: CPU パイプライン
		IF	DF	…			: 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、MA の 4 段で、FPU パイプラインが IF、DF、EX、NA の 4 段で終了します。

●命令発行

この命令は FPU ロードストアパイプラインとメモリアクセスパイプラインを使用します。
まだ計算中の FPU 算術演算命令があるとき、終了するまで待たされます。

●並列可能性

この命令は、FPU 命令および FPU に関する CPU 命令とは並列実行できません。

(7) 浮動小数点レジスタ-レジスタ間転送命令・浮動小数点レジスタ-イミディエイト命令・浮動小数点演算命令の一部

●命令の種類

```
FLDS        FRm,FPUL
FMOV        FRm,FRn
FSTS        FPUL,FRn
FLDI0       FRn
FLDI1       FRn
FABS        FRn
FNEG        FRn
FABS        DRn
FNEG        DRn
```

●パイプライン

	←→	←→	←→	←→	←→	←→	←→ スロット
命令 A	IF	ID	EX				: CPU パイプライン
	IF	DF	EX	NA	SF		: FPU パイプライン
次命令	IF	ID	EX	…			: CPU パイプライン
	IF	DF	E1	E2	SF		: FPU パイプライン
次々命令		IF	ID	EX	…		: CPU パイプライン
		IF	DF	E1	E2	E3	: 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、EX、NA、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いても競合が発生しません。

●命令発行

この命令は FPU ロードストアパイプラインを使用します。

●並列可能性

この命令はレイテンシ 0 の命令です。この命令が先行命令として実行され、後行命令が FRn、FPUL を使用するときも、並列実行できます。

(8)　倍精度浮動小数点レジスタ-レジスタ間転送命令

●命令の種類

```
FMOV        DRm,DRn
```

●パイプライン

	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	EX				: CPU パイプライン
	IF	DF	EX	EX	NA	SF		: FPU パイプライン
次命令	IF	…	ID	EX	…			: CPU パイプライン
	IF	…	DF	E1	E2	SF		: FPU パイプライン
次々命令		IF	…	ID	EX	…		: CPU パイプライン
		IF	…	DF	E1	E2	SF	: 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、EX の 4 段で、FPU パイプラインが IF、DF、EX、EX、NA、SF の 6 段で終了します。

●命令発行

この命令は FPU ロードストアパイプラインを使用します。

●並列可能性

特記事項はありません。

(9)　FSCHG 命令

●命令の種類

```
FSCHG
```

●パイプライン

	←→	←→	←→	←→	←→	←→	←→ スロット
命令 A	IF	ID	EX				: CPU パイプライン
	IF	DF	EX	NA	SF		: FPU パイプライン
次命令	IF	ID	EX	…			: CPU パイプライン
	IF	DF	E1	E2	SF		: FPU パイプライン
次々命令		IF	ID	EX	…		: CPU パイプライン
		IF	DF	E1	E2	SF	: 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、EX、NA、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いても競合が発生しません。

●命令発行

この命令は FPU ロードストアパイプラインを使用します。

●並列可能性

特記事項はありません。

(10) 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
FMOV.D      @Rm,DRn
FMOV.D      @Rm+,DRn
FMOV.D      @(R0,Rm),DRn
```

●パイプライン（単精度）

	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	MA				: CPU パイプライン
	IF	DF	EX	NA	SF			: FPU パイプライン
次命令	IF	ID	EX	…				: CPU パイプライン
	IF	DF	E1	E2	SF			: FPU パイプライン
次々命令		IF	ID	EX	…			: CPU パイプライン
		IF	DF	E1	E2	SF		: FPU パイプライン

●パイプライン（倍精度）

	←→	←→	←→	←→	←→	←→	←→ スロット	
命令 A	IF	ID	EX	MA	MA			: CPU パイプライン
	IF	DF	EX	EX	NA	SF		: FPU パイプライン
次命令	IF	--	ID	EX	…			: CPU パイプライン
	IF	--	DF	E1	E2	SF		: FPU パイプライン
次々命令		IF	--	ID	EX	…		: CPU パイプライン
		IF	--	DF	E1	E2	SF	: FPU パイプライン

●動作説明（単精度）

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、EX、NA、SF の 5 段で終了します。この命令の 1~3 命令後に、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することがあります。

●動作説明（倍精度）

パイプラインは CPU パイプラインが IF、ID、EX、MA、MA の 5 段で、FPU パイプラインが IF、DF、EX、EX、NA、SF の 6 段で終了します。この命令の 1~5 命令後に、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することがあります。

●命令発行

この命令は FPU ロードストアパイプラインとメモリアクセスパイプラインを使用します。

●並列可能性

FMOV.D 命令は、マルチサイクル命令であり、後続命令とは並列実行できません（「8.3.4　マルチサイクル命令による競合の詳細」参照）。

(11) 浮動小数点レジスタロード命令（12 ビットディスプレースメント）

●命令の種類

```
FMOV.S      @(disp12,Rm),FRn
FMOV.D      @(disp12,Rm),DRn
```

●パイプライン（単精度）

	←→	←→	←→	←→	←→	←→	←→ スロット	
命令 A	IF	ID	EX	MA				: CPU パイプライン
	IF	DF	EX	NA	SF			: FPU パイプライン
次命令		IF	ID	EX	…			: CPU パイプライン
		IF	DF	EX	NA	SF		: FPU パイプライン
次々命令		IF	ID	EX	…			: CPU パイプライン
		IF	DF	E1	E2	SF		: FPU パイプライン

●パイプライン（倍精度）

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	MA	MA				: CPU パイプライン
	IF	DF	EX	EX	NA	SF			: FPU パイプライン
次命令		IF	--	ID	EX	…	…		: CPU パイプライン
		IF	--	DF	E1	E2	SF		: FPU パイプライン
次々命令			IF	--	ID	EX	…		: CPU パイプライン
			IF	--	DF	E1	E2	SF	: FPU パイプライン

●動作説明（単精度）

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、EX、NA、SF の 5 段で終了します。この命令の 1~3 命令後に、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することがあります。

●動作説明（倍精度）

パイプラインは CPU パイプラインが IF、ID、EX、MA、MA の 5 段で、FPU パイプラインが IF、DF、EX、EX、NA、SF の 6 段で終了します。この命令の 1~3 命令後に、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することがあります。

●命令発行

この命令は FPU ロードストアパイプラインとメモリアクセスパイプラインを使用します。

●並列可能性

この命令は 32 ビット命令であり、並列実行できません（「8.3.5　32 ビット命令による競合の詳細」参照）。

(12) 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)
FMOV.D      DRm,@Rn
FMOV.D      DRm,@-Rn
FMOV.D      DRm,@(R0,Rn)
```

●パイプライン（単精度）

	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	MA				: CPU パイプライン
	IF	DF	EX	NA				: FPU パイプライン
次命令	IF	ID	EX	…				: CPU パイプライン
	IF	DF	E1	E2	SF			: FPU パイプライン
次々命令		IF	ID	EX	…			: CPU パイプライン
		IF	DF	E1	E2	SF		: FPU パイプライン

●パイプライン（倍精度）

	←→	←→	←→	←→	←→	←→	←→ スロット	
命令 A	IF	ID	EX	MA	MA			: CPU パイプライン
	IF	DF	EX	EX	NA			: FPU パイプライン
次命令	IF	--	ID	EX	…			: CPU パイプライン
	IF	--	DF	E1	E2	SF		: FPU パイプライン
次々命令		IF	--	ID	EX	…		: CPU パイプライン
		IF	--	DF	E1	E2	SF	: FPU パイプライン

●動作説明（単精度）

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、EX、NA の 4 段で終了します。

●動作説明（倍精度）

パイプラインは CPU パイプラインが IF、ID、EX、MA、MA の 5 段で、FPU パイプラインが IF、DF、EX、EX、NA の 5 段で終了します。

●命令発行

この命令は FPU ロードストアパイプラインとメモリアクセスパイプラインを使用します。

●並列可能性

FMOV.D 命令は、マルチサイクル命令であり、後続命令とは並列実行できません（**「8.3.4　マルチサイクル命令による競合の詳細」**参照）。

(13) 浮動小数点レジスタストア命令（12 ビットディスプレースメント）

●命令の種類

```
FMOV.S      FRm,@(disp12,Rn)
FMOV.D      DRm,@(disp12,Rn)
```

●パイプライン（単精度）

	←→	←→	←→	←→	←→	←→	←→ スロット	
命令 A	IF	ID	EX	MA				: CPU パイプライン
	IF	DF	EX	NA				: FPU パイプライン
次命令		IF	ID	EX	…			: CPU パイプライン
		IF	DF	E1	E2	SF		: FPU パイプライン
次々命令		IF	ID	EX	…			: CPU パイプライン
		IF	DF	EX	NA	SF		: FPU パイプライン

●パイプライン（倍精度）

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A		IF	ID	EX	MA	MA			: CPU パイプライン
		IF	DF	EX	EX	NA			: FPU パイプライン
次命令		IF	--	ID	EX	…			: CPU パイプライン
		IF	--	DF	E1	E2	SF		: FPU パイプライン
次々命令			IF	--	ID	EX	…		: CPU パイプライン
			IF	--	DF	E1	E2	SF	: FPU パイプライン

●動作説明（単精度）

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、EX、NA の 4 段で終了します。

●動作説明（倍精度）

パイプラインは CPU パイプラインが IF、ID、EX、MA、MA の 5 段で、FPU パイプラインが IF、DF、EX、EX、NA の 5 段で終了します。

●命令発行

この命令は FPU ロードストアパイプラインとメモリアクセスパイプラインを使用します。

●並列可能性

この命令は 32 ビット命令であり、並列実行できません（「8.3.5　32 ビット命令による競合の詳細」参照）。

(14) 浮動小数点演算命令（FDIV、FSQRT、FLOAT、FTRC 以外）

●命令の種類

```
FADD        FRm,FRn
FMAC    FR0,FRm,FRn
FMUL        FRm,FRn
FSUB        FRm,FRn
FADD        DRm,DRn
FMUL        DRm,DRn
FSUB        DRm,DRn
```

●パイプライン（単精度）

	←→	←→	←→	←→	←→	←→	←→ スロット
命令 A	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	EX	NA	SF		: FPU パイプライン
次々命令		IF	ID	EX	…		: CPU パイプライン
		IF	DF	EX	NA	SF	: FPU パイプライン

●パイプライン（倍精度）

	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→ スロット
命令 A	IF	ID	EX								: CPU パイプライン
	IF	DF	E1	E1	E1	E1	E1	E1	E2	SF	: FPU パイプライン
次命令	IF	ID	EX	MA	WB						: CPU パイプライン
	IF	DF	EX	NA	SF						: FPU パイプライン
次々命令		IF	ID	EX	…						: CPU パイプライン
		IF	DF	EX	NA	SF					: FPU パイプライン

●動作説明（単精度）

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 1~5 命令後に、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することがあります。

●動作説明（倍精度）

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、E2、SF の 10 段で終了します。この命令の 1~15 命令後に、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することがあります。

●命令発行

この命令は FPU 算術演算パイプラインを使用します。競合の詳細については「8.6　FPU による競合」を参照してください。

●並列可能性

特記事項はありません。

(15) 浮動小数点演算命令（FLOAT、FTRC）と FCNVSD、FCNVDS 命令

●命令の種類

FLOAT	FPUL,FRn
FTRC	DRm,FPUL
FLOAT	FPUL,DRn
FTRC	DRm,FPUL
FCNVSD	
FCNVDS	

●パイプライン（単精度）

	←→	←→	←→	←→	←→	←→	←→ スロット
命令 A	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	EX	NA	SF		: FPU パイプライン
次々命令		IF	ID	EX	…		: CPU パイプライン
		IF	DF	E1	E2	SF	: FPU パイプライン

●パイプライン（倍精度）

	←→	←→	←→	←→	←→	←→	←→ スロット
命令 A	IF	ID	EX				: CPU パイプライン
	IF	DF	E1	E1	E2	SF	: FPU パイプライン
次命令	IF	ID	EX	MA	WB		: CPU パイプライン
	IF	DF	EX	NA	SF		: FPU パイプライン
次々命令		IF	ID	EX	…		: CPU パイプライン
		IF	DF	EX	NA	SF	: FPU パイプライン

●動作説明（単精度）

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 1~5 命令後に、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することがあります。

●動作説明（倍精度）

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、E2、SF の 6 段で終了します。この命令の 1~7 命令後に、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することがあります。

●命令発行

この命令は FPU 算術演算パイプラインを使用します。競合の詳細については「8.6 FPU による競合」を参照してください。

●並列可能性

特記事項はありません。

(16) 浮動小数点演算命令（FDIV）

●命令の種類

```
FDIV        FRm,FRn
FDIV        DRm,DRn
```

●パイプライン（単精度）

	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	スロット	
命令 A	IF	ID	EX													: CPU パイプライン
	IF	DF	E1	ED	ED	ED	ED	ED	ED	ED	ED	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	EX	NA	SF											: FPU パイプライン
次々命令		IF	ID	EX	…											: CPU パイプライン
		IF	DF	EX	NA	SF										: FPU パイプライン

●パイプライン（倍精度）

	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	スロット	
命令 A	IF	ID	EX													: CPU パイプライン
	IF	DF	E1	E1	ED	…	ED	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
	IF	DF	EX	NA	SF											: FPU パイプライン
次々命令		IF	ID	EX	…											: CPU パイプライン
		IF	DF	EX	NA	SF										: FPU パイプライン

●動作説明（単精度）

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、ED、ED、ED、ED、ED、ED、ED、ED、E1、E2、SF の 14 段で終了します。すなわち、E1 ステージを 1 回行った後、ED ステージを 8 回繰り返し、最後に E1、E2、SF と進みます。

●動作説明（倍精度）

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、ED、ED、ED、ED、ED、ED、ED、ED、ED、ED、ED、ED、ED、ED、ED、ED、E1、E1、E1、E2、SF の 27 段で終了します。すなわち、E1 ステージを 2 回行った後、ED ステージを 18 回繰り返し、最後に E1、E1、E1、E2、SF と進みます。

「8.6　FPU による競合」に示した競合が発生します。FDIV のパイプライン中に FDIV の結果レジスタにアクセスする命令が重なると、その命令は FDIV 命令が実行を終えるまで待たされます。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 命令の直後から単精度の場合は 21、倍精度の場合は 49 命令以内は FDIV の結果レジスタに対しての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 命令または他の FPU 命令が実行でき、性能が向上します。

●命令発行

この命令は FPU 算術演算パイプラインを使用します。競合の詳細については「8.6　FPU による競合」を参照してください。

この命令の ED ステージは、スロットにかかわりなく、ステートで動作します。

●並列可能性

特記事項はありません。

(17) 浮動小数点演算命令（FSQRT）

●命令の種類

FSQRT FRn

FSQRT DRn

●パイプライン（単精度）

	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→ スロット	
命令 A	IF	ID	EX												: CPU パイプライン
	IF	DF	E1	ED	ED	ED	ED	ED	ED	ED	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	EX	NA	SF										: FPU パイプライン
次々命令		IF	ID	EX	…										: CPU パイプライン
		IF	DF	EX	NA	SF									: FPU パイプライン

●パイプライン（倍精度）

	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→	←→ スロット	
命令 A	IF	ID	EX												: CPU パイプライン
	IF	DF	E1	E1	ED	…	ED	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
	IF	DF	EX	NA	SF										: FPU パイプライン
次々命令		IF	ID	EX	…										: CPU パイプライン
		IF	DF	EX	NA	SF									: FPU パイプライン

●動作説明（単精度）

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、ED、ED、ED、ED、ED、ED、ED、E1、E2、SF の 13 段で終了します。すなわち、E1 ステージを 1 回行った後、ED ステージを 7 回繰り返し、最後に E1、E2、SF と進みます。

●動作説明（倍精度）

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、ED、ED、ED、ED、ED、ED、ED、ED、ED、ED、ED、ED、ED、ED、ED、ED、ED、E1、E1、E1、E2、SF の 26 段で終了します。すなわち、E1 ステージを 2 回行った後、ED ステージを 17 回繰り返し、最後に E1、E1、E1、E2、SF と進みます。

「8.6 FPU による競合」に示した競合が発生します。FSQRT のパイプライン中に FSQRT の結果レジスタにアクセスする命令が重なると、その命令は FSQRT 命令が実行を終えるまで待たされます。E1 以降のステージが FSQRT 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、FSQRT 命令の直後から単精度の場合は 19、倍精度の場合は 47 命令以内は FSQRT の結果レジスタに対しての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 命令または他の FPU 命令が実行でき、性能が向上します。

●命令発行

この命令は FPU 算術演算パイプラインを使用します。競合の詳細については「8.6 FPU による競合」を参照してください。

この命令の ED ステージは、スロットにかかわりなく、ステートで動作します。

●並列可能性

特記事項はありません。

(18) 浮動小数点比較命令

●命令の種類

FCMP/EQ　FRm,FRn

FCMP/GT　FRm,FRn

FCMP/EQ　DRm,DRn

FCMP/GT　DRm,DRn

●パイプライン（単精度）

	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX					: 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	EX	NA	SF			: FPU パイプライン
次々命令		IF	ID	EX	…			: CPU パイプライン
		IF	DF	E1	E2	SF		: FPU パイプライン

●パイプライン（倍精度）

	←→	←→	←→	←→	←→	←→	←→	←→	スロット
命令 A	IF	ID	EX	EX					: CPU パイプライン
	IF	DF	E1	E1	E2				: FPU パイプライン
次命令	IF	--	ID	EX	…				: CPU パイプライン
	IF	--	DF	EX	NA	SF			: FPU パイプライン
次々命令		IF	--	ID	EX	…			: CPU パイプライン
		IF	--	DF	EX	NA	SF		: FPU パイプライン

●動作説明（単精度）

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 の 4 段で終了します。T ビットは E2 で確定しますので、直後の T ビットを参照する命令は 2 サイクルストールします。

FCMP	IF	ID	EX			: CPU パイプライン
	IF	DF	E1	E2		: FPU パイプライン
BT	IF	--	--	ID	EX	: CPU パイプライン
	IF	--	--	DF	…	: FPU パイプライン

●動作説明（倍精度）

パイプラインは CPU パイプラインが IF、ID、EX、EX の 4 段で、FPU パイプラインが IF、DF、E1、E1、E2 の 5 段で終了します。T ビットは E2 で確定しますので、直後の T ビットを参照する命令は 3 サイクルストールします。

FCMP	IF	ID	EX				: CPU パイプライン
	IF	DF	E1	E1	E2		: FPU パイプライン
BT	IF	--	--	--	ID	EX	: CPU パイプライン
	IF	--	--	--	DF	…	: FPU パイプライン

●命令発行

この命令は FPU 算術演算パイプラインを使用します。

●並列可能性

倍精度の FCMP 命令は、後続命令とは並列実行できません。

8.10 簡易的な必要サイクル数の算出

以下に、簡易的な必要サイクル数の算出方法を示します。大まかではありますが、この方法により、ターゲット命令列を実行するのに必要なサイクル数を算出することができます。

このとき、以下のルールで算出します。

(1) 命令は、既にフェッチ出来ているとし、命令フェッチは考えません。
(2) 32bit命令は、「実行ステート」サイクルで動作します。
(3) リソース競合がある場合、先に発行された命令は「実行ステート」サイクルで動作します。後続の命令と並列実行はできません。
(4) 先に発行された命令の結果を直後の命令が使用するとき、先に発行された命令は「レイテンシ」サイクルかかると計上します。
(5) 先に発行された命令の結果を直後の命令が使用しないとき、先に発行された命令は「実行ステート」サイクルかかると計上します。
(6) 並列実効の効果は、単純に補正項で修正します。

以上、これだけでは多々例外があり、100%の精度は出ませんが、大まかなサイクル数を計上することができます。以下、例を示します。

1. レイテンシで数える場合

		←→	←→	←→	←→	←→	←→	←→	←→	サイクル数
MOV.L	@R1, R0		ID	EX	MA	WB				2
ADD	#imm, R0		--	--	ID	EX				1
MOV.L	R0, @R1			--		ID	EX	MA		

ADD は先行の MOV.L の結果を使用しますので、MOV.L は「レイテンシ」サイクル数（2Cyc）実行に必要と計上します。その次の MOV.L は ADD の結果を使用しますので、ADD 命令は「レイテンシ」（1Cyc）実行に必要と計上します。

2. 実行ステートで数える場合

		←→	←→	←→	←→	←→	←→	←→	←→	サイクル数
MOV.L	@R1, R0		ID	EX	MA	WB				1→0
ADD	#imm, R2		ID	EX						1
MOV.L	R3, @R4			ID	EX	MA				

この場合、先に発行された命令の結果を後続命令が使用していないため、各命令はリソース競合のない限り並列実行されます。各命令の実行に必要なサイクル数は「実行ステート」で計上し、先行命令側の実行ステートが 1 の場合、後行命令を並列実行することができます。並列実行された場合、先行命令の実行に必要なサイクル数は「実行ステート」－1 と計上しますが、これは補正として簡略に扱います。（この補正は、後述の式の最後の項として出現します）

3. リソース競合がある場合

		←→	←→	←→	←→	←→	←→	←→	←→	サイクル数
MOV.L	@R1, R0		ID	EX	MA	WB				1
MOV.L	@R3, R2			ID	EX	MA	WB			1
MOV.L	@R5, R4				ID	EX	MA	WB		

リソース競合がある場合、並列実行できません。各命令の実行には「実行ステート」必要となります。

4. 実行ステートが 1 より大きい命令

		←→	←→	←→	←→	←→	←→	←→	←→	サイクル数
AND.B	#imm, @(R0, GBR)		ID	EX	MA	EX				3→2
ADD	#imm, R1				ID	EX				1
BAND.B	#imm, @(disp12, R2)					ID	EX	MA	EX	3
ROTCL									ID	

実行ステートが 1 より大きい命令は、この実行ステートがだんだん減っていくと考え、1 になると後続命令と並列実行できます。このとき、後続命令と並列実行されると、実効に必要なサイクル数は「実効ステート」－1、並列実行されなかった場合、「実行ステート」となります。ここは補正として簡略に扱います。（この補正は、後述の式の最後の項として出現します）

これらを元に、非常に単純化して考えると、命令列を実行するのに必要なサイクル数は、以下となります。依存関係がある区間とない区間が存在する場合は、それぞれ分けて計上し、足し合わせてください。

- 各命令間にすべて依存関係がある場合。
  必要サイクル数＝すべての命令の「レイテンシ」を足し合わせたサイクル数。

- 各命令間にまったく依存関係がない場合。
  必要サイクル数＝すべての命令の「実行ステート」を足し合わせたもの
  －（全命令の数－並列実行できない命令の数）÷2
  ここで、「並列実行できない命令の数」とは、リソース競合にて並列実行できない命令の数（特に、直前の命令がメモリアクセス命令である、メモリアクセス命令）と、「実行ステート」が1でない命令と、32bit命令を足した数です。

この最後の項により、並列実行した場合における先行命令の必要サイクル減少分を補正します。

例：各命令間にすべて依存関係がある場合

```
BAND.B
ROTCL
BAND.B
ROTCL
```

すべての命令の「レイテンシ」を足し合わせ、8Cyc となります。

例：各命令間にまったく依存関係がない場合

```
ADD     #imm, R0
BAND.B  #imm, @(disp12, R2)
MULR    R4, R0
ROTCL   R5
```

必要なサイクル数＝1＋3＋2＋1－（4－2）÷2
＝7－1＝6Cyc

付録

A. SH-2A/SH2A-FPU 並列実行性

- 使用している演算器の種類に応じて、下表の並列実行可否判定を行います。分類が複数ある命令については、各分類の欄を確認し、すべての交差の欄が○である場合、並列実行します。

		第 2 命令										
		(1) BR	(2) MR	(3) MW	(4) MF	(5) ML	(6) MU	(7) SF	(8) FL	(9) FP	(10) FC	(11) EX
第 1 命令	(1) BR	×	○	○	○	○	○	○	○	○	○	○
	(2) MR	○	×	×	○	○	○	○	○	○	○	○
	(3) MW	○	×	×	×	○	○	○	○	○	○	○
	(4) MF	○	○	×	×	○	○	○	○	○	○	○
	(5) ML	○	○	○	○	×	○	○	○	○	○	○
	(6) MU	○	○	○	○	○	×	○	○	○	○	○
	(7) SF	○	○	○	○	○	○	×	○	○	○	○
	(8) FL	○	○	○	○	○	○	○	×	○	×	○
	(9) FP	○	○	○	○	○	○	○	○	×	×	○
	(10) FC	×	×	×	×	×	×	×	×	×	×	×
	(11) EX	○	○	○	○	○	○	○	○	○	○	○

第 1 命令時の分類	第 2 命令時の分類	命令					
BR	BR	BF	disp	BF/S	disp	BT	disp
		BT/S	disp	BSR	disp	BSRF	Rm
		BRA	disp	BRAF	Rm	JMP	@Rm
		JSR	@Rm	JSR/N	@Rm	RTS	
		RTS/N		RTV/N	Rm	TRAPA	#imm
MR	MR	LDC.L	@Rm+,GBR	LDC.L	@Rm+,VBR	LDS.L	@Rm+,PR
		MOV.B	@(disp,GBR),R0	MOV.B	@(disp,Rm),R0	MOV.B	@(R0,Rm),Rn
		MOV.B	@Rm,Rn	MOV.B	@Rm+,Rn	MOV.B	@-Rm,R0
		MOV.B	@(disp12,Rm),Rn	MOV.W	@(disp,GBR),R0	MOV.W	@(disp,Rm),R0
		MOV.W	@(R0,Rm),Rn	MOV.W	@Rm,Rn	MOV.W	@Rm+,Rn
		MOV.W	@-Rm,R0	MOV.W	@(disp12,Rm),Rn	MOV.W	@(disp,PC),Rn
		MOV.L	@(disp,GBR),R0	MOV.L	@(disp,Rm),Rn	MOV.L	@(R0,Rm),Rn
		MOV.L	@Rm,Rn	MOV.L	@Rm+,Rn	MOV.L	@-Rm,R0
		MOV.L	@(disp12,Rm),Rn	MOV.L	@(disp,PC),Rn	MOVU.B	@(disp12,Rm),Rn
		MOVU.W	@(disp12,Rm),Rn	MOVML.L	@R15+,Rn	MOVMU.L	@R15+,Rn
		PREF	@Rn				

第 1 命令時の分類	第 2 命令時の分類	命令					
MW	MR	AND.B	#imm,@(R0,GBR)	BCLR.B	#imm3,@(disp12,Rn)	BSET.B	#imm3,@(disp12,Rn)
		BST.B	#imm3,@(disp12,Rn)	OR.B	#imm,@(R0,GBR)	STC.L	SR,@-Rn
		TAS.B	@Rn	XOR.B	#imm,@(R0,GBR)		
MW	MW	MOV.B	R0,@(disp,GBR)	MOV.B	R0,@(disp,Rn)	MOV.B	Rm,@(R0,Rn)
		MOV.B	Rm,@Rn	MOV.B	Rm,@-Rn	MOV.B	R0,@Rn+
		MOV.B	Rm,@(disp12,Rn)	MOV.W	R0,@(disp,GBR)	MOV.W	R0,@(disp,Rn)
		MOV.W	Rm,@(R0,Rn)	MOV.W	Rm,@Rn	MOV.W	Rm,@-Rn
		MOV.W	R0,@Rn+	MOV.W	Rm,@(disp12,Rn)	MOV.L	R0,@(disp,GBR)
		MOV.L	Rm,@(disp,Rn)	MOV.L	Rm,@(R0,Rn)	MOV.L	Rm,@Rn
		MOV.L	Rm,@-Rn	MOV.L	R0,@Rn+	MOV.L	Rm,@(disp12,Rn)
		MOVML.L	Rm,@-R15	MOVMU.L	Rm,@-R15	STC.L	GBR,@-Rn
		STC.L	VBR,@-Rn	STS.L	PR,@-Rn		
ML	ML	STS	MACH,Rn	STS	MACL,Rn		
MU	MU	CLRMAC		DMULS.L	Rm,Rn	DMULU.L	Rm,Rn
		MUL.L	Rm,Rn	MULS.W	Rm,Rn	MULU.W	Rm,Rn
		LDS	Rm,MACL	LDS	Rm,MACH		
ML,MU	ML	MULR	R0,Rn				
SF	SF	DIVU	R0,Rn	EXTS.B	Rm,Rn	EXTS.W	Rm,Rn
		EXTU.B	Rm,Rn	EXTU.W	Rm,Rn	ROTCL	Rn
		ROTCR	Rn	ROTL	Rn	ROTR	Rn
		SHAD	Rm,Rn	SHAL	Rn	SHAR	Rn
		SHLD	Rm,Rn	SHLL	Rn	SHLL16	Rn
		SHLL2	Rn	SHLL8	Rn	SHLR	Rn
		SHLR16	Rn	SHLR2	Rn	SHLR8	Rn
		SWAP.B	Rm,Rn	SWAP.W	Rm,Rn	XTRCT	Rm,Rn
FL	FL	FABS	DRn	FABS	FRn	FLDI0	FRn
		FLDI1	FRn	FLDS	FRm,FPUL	FMOV	DRm,DRn
		FMOV	FRm,FRn	FNEG	DRn	FNEG	FRn
		FSTS	FPUL,FRn				
ML,FL	ML,FL	STS	FPUL,Rn				
FP	FP	FADD	DRm,DRn	FADD	FRm,FRn	FCMP/EQ	FRm,FRn
		FCMP/GT	FRm,FRn	FCNVDS	DRm,FPUL	FCNVSD	FPUL,DRn
		FDIV	DRm,DRn	FDIV	FRm,FRn	FLOAT	FPUL,DRn
		FLOAT	FPUL,FRn	FMAC	FR0,FRm,FRn	FMUL	DRm,DRn
		FMUL	FRm,FRn	FSCHG		FSQRT	DRn
		FSQRT	FRn	FSUB	DRm,DRn	FSUB	FRm,FRn
		FTRC	DRm,FPUL	FTRC	FRm,FPUL		
FC	FC	FCMP/EQ	DRm,DRn	FCMP/GT	DRm,DRn		
ML,FC	ML,FC	STS	FPSCR,Rn				

第 1 命令時の分類	第 2 命令時の分類	命令		
EX	EX	ADD #imm,Rn	ADD Rm,Rn	ADDC Rm,Rn
		ADDV Rm,Rn	AND #imm,R0	AND Rm,Rn
		BCLR #imm3,Rn	BLD #imm3,Rn	BSET #imm3,Rn
		BST #imm3,Rn	CLRT	CMP/EQ #imm,R0
		CMP/EQ Rm,Rn	CMP/GE Rm,Rn	CMP/GT Rm,Rn
		CMP/HI Rm,Rn	CMP/HS Rm,Rn	CMP/PL Rn
		CMP/PZ Rn	CMP/STR Rm,Rn	CLIPS.B Rn
		CLIPS.W Rn	CLIPU.B Rn	CLIPU.W Rn
		DIV0S Rm,Rn	DIV0U	DIVS R0,Rn
		DIV1 Rm,Rn	DT Rn	LDC Rm,GBR
		LDC Rm,SR	LDC Rm,TBR	LDC Rm,VBR
		LDS Rm,PR	LDBANK @Rm,R0	MOV #imm,Rn
		MOV Rm,Rn	MOVA @(disp,PC),R0	MOVI20 #imm20,Rn
		MOVI20S #imm20,Rn	MOVT Rn	MOVRT Rn
		NEG Rm,Rn	NEGC Rm,Rn	NOP
		NOT Rm,Rn	NOTT	OR #imm,R0
		OR Rm,Rn	SETT	STC GBR,Rn
		STC SR,Rn	STC TBR,Rn	STC VBR,Rn
		STS PR,Rn	STBANK R0,@Rn	SUB Rm,Rn
		SUBC Rm,Rn	SUBV Rm,Rn	TST #imm,R0
		TST Rm,Rn	XOR #imm,R0	XOR Rm,Rn
		RESBANK(BO==0)		
MR,MU	MR,MU	LDS.L @Rm+,MACH	LDS.L @Rm+,MACL	
MW.ML	MW,ML	STS.L MACH,@-Rn	STS.L MACL,@-Rn	
MW,FL	MW,FL	FMOV.S @(R0,Rm),FRn	FMOV.S @Rm,FRn	FMOV.S @Rm+,FRn
		FMOV.S @(disp12,Rm),FRn	FMOV.S FRm,@(R0,Rn)	FMOV.S FRm,@-Rn
		FMOV.S FRm,@Rn	FMOV.S FRm,@(disp12,Rn)	FMOV.D @(R0,Rm),DRn
		FMOV.D @Rm,DRn	FMOV.D @Rm+,DRn	FMOV.D @(disp12,Rm),DRn
		FMOV.D DRm,@(R0,Rn)	FMOV.D DRm,@-Rn	FMOV.D DRm,@Rn
		FMOV.D DRm,@(disp12,Rn)		
MF,FL	MF,FL	LDS Rm,FPUL		
MF,FC	MF,FC	LDS Rm,FPSCR		
MR,FC	MR,FC	LDS.L @Rm+,FPSCR	LDS.L @Rm+,FPUL	
MW,ML,FC	MW,ML,FC	STS.L FPSCR,@-Rn	STS.L FPUL,@-Rn	
BR	MR	JSR/N @@(disp8,TBR)		
MR,MU	MR	RESBANK(BO==1)		
EX	MR	BAND.B #imm3,@(disp12,Rn)	BANDNOT.B #imm3,@(disp12,Rn)	BLD.B #imm3,@(disp12,Rn)
		BLDNOT.B #imm3,@(disp12,Rn)	BOR.B #imm3,@(disp12,Rn)	BORNOT.B #imm3,@(disp12,Rn)
		BXOR.B #imm3,@(disp12,Rn)	LDC.L @Rm+,SR	RTE
		SLEEP	TST.B #imm,@(R0,GBR)	
MU	MR	MAC.W @Rm+,@Rn+	MAC.L @Rm+,@Rn+	

- マルチサイクル命令は最初のサイクルと最後のサイクルで並列実行を行います。
- FPU 命令は、SH4 の分類を踏襲する（①LS タイプ②FE タイプ③CO タイプ）。新規命令である 32 ビット FMOV 命令は①LS タイプに分類します。
- 32 ビット命令は、原則、前の命令がマルチサイクル命令であれば並列実行可能。次の命令との並列実行は不可。ただし、メモリ-Tbit 間ビット操作命令同士の組み合わせは並列実行可能。
- MOVMU.L,MOVML.L 命令は、前の命令との並列実行は不可。
- 遅延分岐命令と遅延スロットとの並列実行は不可。

・マルチサイクル命令：

TRAPA, MOVMU.L, MOVML.L, AND.B, OR.B, TST.B, XOR.B, TAS.B, BCLR.B, BSET.B, BST.B, BAND.B, BANDNOT.B, BLD.B, BLDNOT.B, BOR.B, BORNOT.B, BXOR.B, MUL.L, DMULS.L, DMULU.L, MULR, DIVU, DIVS, FCMP/EQ DRm,DRn, FCMP/GT DRm,DRn, LDC Rm,SR, STC SR,Rn, LDC.L @Rm+,SR, STC.L SR,@-Rn, LDBANK, STBANK, RESBANK, FMOV.D, FMOV DRm,DRn, JSR/N @@(disp,TBR), SLEEP, RTE, MAC.W, MAC.L

・32 ビット命令：

MOVI20, MOVI20S, MOV.B @(disp12,Rm),Rn, MOV.W @(disp12,Rm),Rn, MOV.L @(disp12,Rm),Rn, MOV.B Rm,@(disp12,Rn), MOV.W Rm,@(disp12,Rn), MOV.L Rm,@(disp12,Rn),MOVU.B, MOVU.W, FMOV.S @(disp12,Rm),FRn, FMOV.D @(disp12,Rm),DRn, FMOV.S FRm,@(disp12,Rn), FMOV.D DRm,@(disp12,Rn), BCLR.B, BSET.B, BST.B, BAND.B, BANDNOT.B, BLD.B, BLDNOT.B, BOR.B, BORNOT.B, BXOR.B

・32 ビット FMOV 命令：

FMOV.S @(disp12,Rm),FRn, FMOV.D @(disp12,Rm),DRn, FMOV.S FRm,@(disp12,Rn), FMOV.D DRm,@(disp12,Rn),

・メモリ-Tbit 間ビット操作命令：

BAND.B, BANDNOT.B, BLD.B, BLDNOT.B, BOR.B, BORNOT.B, BXOR.B

・遅延分岐命令：

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

B. プログラミングの指針（MOVI20, MOVI20S の使用方法について）

SH2A、SH2A-FPU において、MOVI20＃imm20,Rn および MOVI20S ＃imm20,Rn 命令は PC 相対命令によるリテラルアクセスを減らし、サイクル性能を向上させる効果があります。これらの命令の効果を得るために、アセンブラでは以下のように記述することを推奨します。

(1) MOVI20 の使用方法

MOVI20 では符号拡張を行います。この命令により、H'00000000～H'0007FFFF, H'FFF80000～H'FFFFFFFF の範囲を表現できます。

下記の命令列を連続して配置してください。

```
MOVI20 #imm20, Rn
無条件分岐命令*1
```

例）

```
MOVI20 #imm20, Rn
JMP @Rm
```

(2) MOVI20S の使用方法

MOVI20S では符号拡張を行います。これと ADD#imm, Rn 命令を組み合わせることにより、H'00000000～H'07FFFF7F, H'F7FFFF80～H'FFFFFFFF の範囲を表現できます。

下記の命令列を連続して配置してください。

```
MOVI20S #imm20, Rn
ADD #imm, Rn
無条件分岐命令*1
```

例）

```
MOVI20S #imm20, Rn
ADD #imm, Rn
JMP @Rm
```

【注】1. H'07FF FF80～H'07FF FFFF の範囲の指定は

```
MOVI20S #imm20, R0
OR #imm, R0
無条件分岐命令*1
```

または、以下のように 32bit アドレスリードで指定してください。

```
MOV.L @(disp, PC), Rn
無条件分岐命令*1
```

*1 無条件分岐命令：BRAF Rm, BSRF Rm, JMP @Rm, JSR @Rm, JSR/N @Rm

SH-2A、SH2A-FPU ユーザーズマニュアル
ソフトウェア編

発行年月日　2004年3月8日　Rev.1.00
2011年2月22日　Rev.4.00

発行　ルネサス エレクトロニクス株式会社
〒211-8668　神奈川県川崎市中原区下沼部1753

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所・電話番号は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス販売株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル）

(03)5201-5307

■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：http://japan.renesas.com/inquiry

SH-2A、SH2A-FPU

RENESAS

ルネサス エレクトロニクス株式会社

R01US0031JJ0400
(旧番号：RJJ09B0086-0300)