お客様各位

カタログ等資料中の旧社名の扱いについて

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

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

2010年4月1日
ルネサスエレクトロニクス株式会社

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

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

ご注意書き

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

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

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

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

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

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

User's Manual

32

M32R-FPU

ソフトウェアマニュアル

ルネサス32ビットRISCシングルチップマイクロコンピュータ

ルネサス エレクトロニクス
www.renesas.com

Rev.1.01　2003.10

安全設計に関するお願い

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

本資料ご利用に際しての留意事項

・本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報についてルネサス テクノロジが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。

・本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、ルネサス テクノロジは責任を負いません。

・本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、ルネサス テクノロジは、予告なしに、本資料に記載した製品または仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前にルネサス テクノロジ、ルネサス販売または特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ（http://www.renesas.com）などを通じて公開される情報に常にご注意ください。

・本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、ルネサス テクノロジはその責任を負いません。

・本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。ルネサス テクノロジは、適用可否に対する責任は負いません。

・本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。

・本資料の転載、複製については、文書によるルネサス テクノロジの事前の承諾が必要です。

・本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたらルネサス テクノロジ、ルネサス販売または特約店までご照会ください。

改訂記録	M32R-FPU ソフトウエアマニュアル

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2002.07.12	－	初版発行
1.01	2003.10.31	1-9	(5) 無効演算例外の表の項目の修正　誤）-∞(-∞)　→　正) -∞-(-∞)
		3-57	用語の修正　誤）トラップ発生時　→　正）EIT発生時
		付録 -2, 3	16 進命令コード対応表の修正（JL, JMP、LDI 命令部分）
		付録 -4	CLRPSW #imm8 のニーモニックの修正
		付録 -8	付図 3.1.1 の修正　誤）※ FDIV 命令時の E1 ステージは 13 サイクルとなります 正）※ FDIV 命令時の E1 ステージは 14 サイクルとなります
		付録 -10	付図3.2.1の修正　誤）LD1　→　正）LDI
		付録 -13	付図3.2.4の修正　誤）ADD　*R1*, R6, R7　→　正）FMADD　*R1*, R6, R7
		付録 -17	付表4.1.1の修正　誤）BEST, BCLR命令　→　正）BSET, BCLR命令

目次

＊空きページです＊

第１章

CPUプログラミングモデル

1.1 CPUレジスタ

M32RファミリのFPU内蔵CPU(以下M32R-FPU)には16本の汎用レジスタ、6本の制御レジスタ、アキュムレータ及びプログラムカウンタがあります。アキュムレータは56ビット、その他のレジスタはすべて32ビット構成になっています。

1.2 汎用レジスタ

汎用レジスタは32ビット幅で16本(R0～R15)あり、データやベースアドレスの保持、整数演算/浮動小数点演算等に使用します。R14はリンクレジスタとして、R15はスタックポインタとして使用されます。リンクレジスタはサブルーチン呼び出し命令実行の際、戻り先番地の格納に使われます。またスタックポインタは、プロセッサ状態語レジスタ(PSW)のスタックモード(SM)ビットの値に応じて割り込み用スタックポインタ(SPI)と、ユーザ用スタックポインタ(SPU)とに切り替わります。

リセット解除時、汎用レジスタの値は不定です。

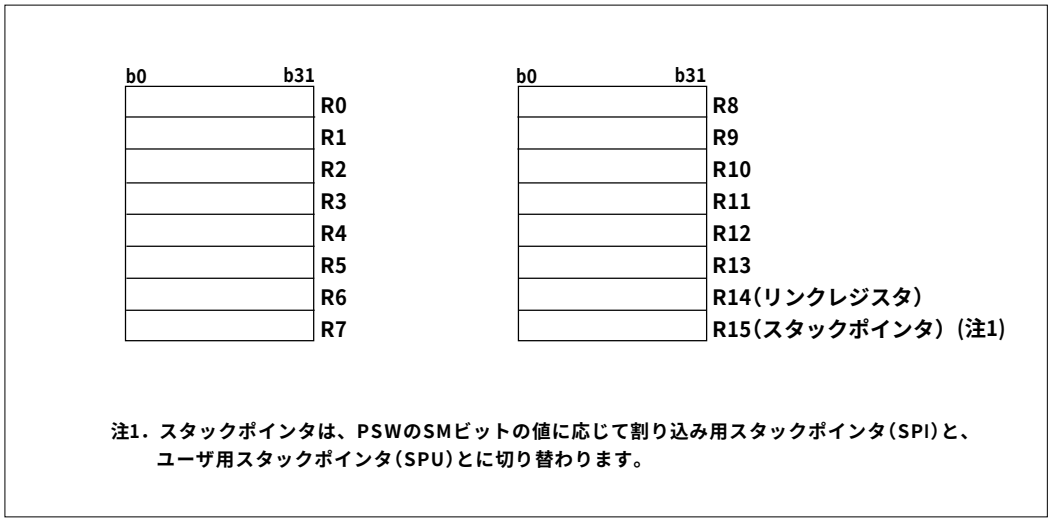


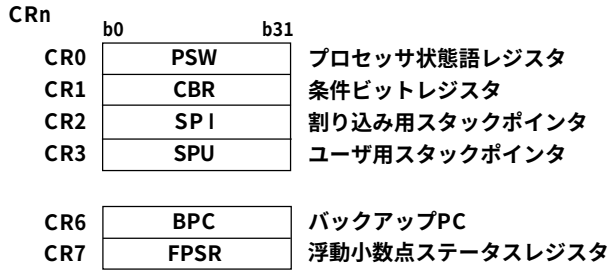
図1.2.1 汎用レジスタ

1.3 制御レジスタ

制御レジスタ(CR)には、プロセッサ状態語レジスタ(PSW)，条件ビットレジスタ(CBR)，割り込み用スタックポインタ(SPI)，ユーザ用スタックポインタ(SPU)，バックアップPC(BPC)，浮動小数点ステータスレジスタ(FPSR)の6つがあります。

これら制御レジスタの設定や読み出しには、専用の「MVTC命令」と「MVFC命令」を使用します。

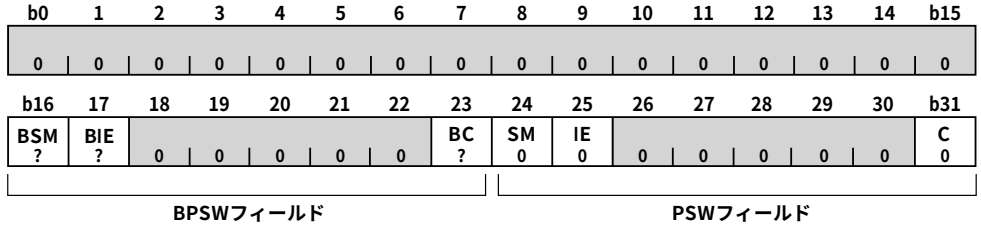
また、プロセッサ状態語レジスタ(PSW)のSMビット、IEビット及びCビットについては、「SETPSW命令」と「CLRPSW命令」でも設定できます。



注．・CRn（n=0～3, 6, 7）は制御レジスタの番号です。
・制御レジスタの設定、読み出しには、専用の「MVTC命令」と「MVFC命令」を使用します。
・PSWのSMビット、IEビット及びCビットについては、「SETPSW命令」と「CLRPSW命令」でも設定できます。

図1.3.1 制御レジスタ

1.3.1 プロセッサ状態語レジスタ：PSW（CR0）



＜リセット解除時："B'0000 0000 0000 0000 ??00 000? 0000 0000＞

b	ビット名	機能	R	W
0～15	何も配置されていません。"0"に固定してください。		0	0
16	BSM バックアップSMビット	EIT受付時に、SMビットの値が保存される	R	W
17	BIE バックアップIEビット	EIT受付時に、IEビットの値が保存される	R	W
18～22	何も配置されていません。"0"に固定してください。		0	0
23	BC バックアップCビット	EIT受付時に、Cビットの値が保存される	R	W
24	SM スタックモードビット	0：割り込み用スタックポインタを使用 1：ユーザ用スタックポインタを使用	R	W
25	IE 割り込みイネーブルビット	0：割り込みを受け付けない 1：割り込みを受け付ける	R	W
26～30	何も配置されていません。"0"に固定してください。		0	0
31	C 条件ビット	命令の実行に応じて演算結果のキャリー、ボロー、 オーバフローの有無を示す	R	W

プロセッサ状態語レジスタ(PSW)は、M32R-FPUのステータスを表示するレジスタで、通常使用するPSWフィールドと、EIT発生時にPSWフィールドを退避するためのBPSWフィールドからなります。

PSWフィールドは、スタックモードビット(SM)，割り込みイネーブルビット(IE)，条件ビット(C)の各ビットで構成されています。

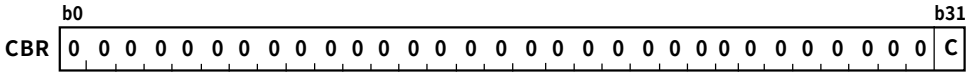
また、BPSWフィールドはバックアップSMビット(BSM)，バックアップIEビット(BIE)，バックアップCビット(BC)で構成されています。

リセット解除時、BSM, BIE, BCは不定、それ以外のビットは"0"です。

1.3.2 条件ビットレジスタ：CBR(CR1)

条件ビットレジスタ(CBR)は、PSWのうち条件ビット(C)を抜き出して別レジスタとしたものです。PSWの条件ビット(C)に書き込まれた値はこのレジスタに反映されます。このレジスタは読み出しのみ可能です(「MVTC命令」で書き込みを行っても無視されます)。

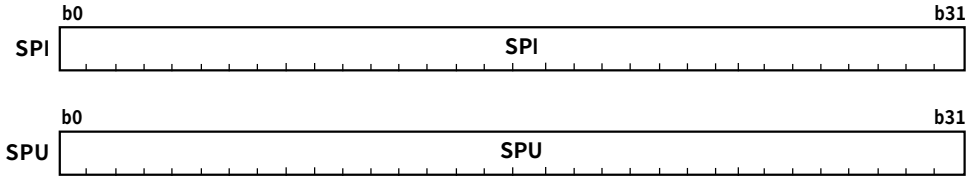
リセット解除時、CBRは"H'0000 0000"です。



1.3.3 割り込み用スタックポインタ：SPI(CR2)
ユーザ用スタックポインタ：SPU(CR3)

割り込み用スタックポインタ(SPI)、ユーザ用スタックポインタ(SPU)は、現在のスタックポインタのアドレスを保持します。これらのレジスタは、汎用レジスタR15としてアクセスできます。このときR15をSPIとして使用するかSPUとして使用するかは、PSWのスタックモードビット(SM)によって切り替わります。

リセット解除時は、SPIとSPUは不定です。

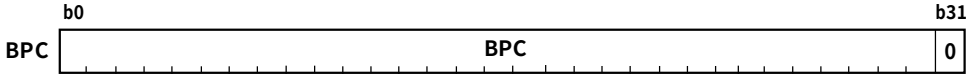


1.3.4 バックアップPC：BPC(CR6)

バックアップPC(BPC)は、EIT発生時にプログラムカウンタ(PC)の値を退避するためのレジスタです。b31は"0"に固定です。

EIT発生時には発生したEITによってEIT発生時又は次命令のPC値がセットされ、「RTE命令」実行時にBPCの値はPCに戻されます。ただし復帰時にPCの下位2ビットは常に"00"になります(常にワード境界に復帰します)。

リセット解除時、BPCは不定です。



1.3.5 浮動小数点ステータスレジスタ：FPSR（CR7）

b0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	b15
FS 0	FX 0	FU 0	FZ 0	FO 0	FV 0	0	0	0	0	0	0	0	0	0	0

b16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	b31
0	EX 0	EU 0	EZ 0	EO 0	EV 0	0	DN 1	CE 0	CX 0	CU 0	CZ 0	CO 0	CV 0	RM 0	0

＜リセット解除時：H0000 0100＞

b	ビット名	機能	R	W
0	FS 浮動小数点例外サマリビット	FU, FZ, FO, FVの論理和を反映します。	R	ー
1	FX 精度異常例外フラグ	精度異常例外の発生（EIT処理実行なしの場合（注1））により、"1"にセットされます。一度"1"にセットされると、ソフトウエアにて"0"クリアするまで、"1"の値を保持します。	R	W
2	FU アンダフロー例外フラグ	アンダフロー例外の発生（EIT処理実行なしの場合（注1））により、"1"にセットされます。一度"1"にセットされると、ソフトウエアにて"0"クリアするまで、"1"の値を保持します。	R	W
3	FZ ゼロ除算例外フラグ	ゼロ除算例外の発生（EIT処理実行なしの場合（注1））により、"1"にセットされます。一度"1"にセットされると、ソフトウエアにて"0"クリアするまで、"1"の値を保持します。	R	W
4	FO オーバフロー例外フラグ	オーバフロー例外の発生（EIT処理実行なしの場合（注1））により、"1"にセットされます。一度"1"にセットされると、ソフトウエアにて"0"クリアするまで、"1"の値を保持します。	R	W
5	FV 無効演算例外フラグ	無効演算例外の発生（EIT処理実行なしの場合（注1））により、"1"にセットされます。一度"1"にセットされると、ソフトウエアにて"0"クリアするまで、"1"の値を保持します。	R	W
6～16	何も配置されていません。"0" に固定してください。		0	0
17	EX 精度異常例外イネーブルビット	0：精度異常例外の発生によるEIT処理をマスク 1：精度異常例外の発生時にEIT処理を実行	R	W
18	EU アンダフロー例外イネーブルビット	0：アンダフロー例外の発生によるEIT処理をマスク 1：アンダフロー例外の発生時にEIT処理を実行	R	W
19	EZ ゼロ除算例外イネーブルビット	0：ゼロ除算例外の発生によるEIT処理をマスク 1：ゼロ除算例外の発生時にEIT処理を実行	R	W
20	EO オーバフロー例外イネーブルビット	0：オーバフロー例外の発生によるEIT処理をマスク 1：オーバフロー例外の発生時にEIT処理を実行	R	W
21	EV 無効演算例外イネーブルビット	0：無効演算例外発生によるEIT処理をマスク 1：無効演算例外の発生時にEIT処理実行	R	W
22	何も配置されていません。"0" に固定してください。		0	0

23	DN 非正規化数の0フラッシュビット (注2)	0：非正規化数を非正規化数として扱います。 1：非正規化数を0として扱います。	R W
24	CE 非実装例外要因ビット	0：非実装例外の発生なし 1：非実装例外の発生あり "1"にセットされている場合、FPU演算命令実行時、 "0"にクリアされます。	R (注3)
25	CX 精度異常例外要因ビット	0：精度異常例外の発生なし 1：精度異常例外の発生あり "1"にセットされている場合、FPU演算命令実行時、 "0"にクリアされます。	R (注3)
26	CU アンダフロー例外要因ビット	0：アンダフロー例外の発生なし 1：アンダフロー例外の発生あり "1"にセットされている場合、FPU演算命令実行時、 "0"にクリアされます。	R (注3)
27	CZ ゼロ除算例外要因ビット	0：ゼロ除算例外の発生なし 1：ゼロ除算例外の発生あり "1"にセットされている場合、FPU演算命令実行時、 "0"にクリアされます。	R (注3)
28	CO オーバフロー例外要因ビット	0：オーバフロー例外の発生なし 1：オーバフロー例外の発生あり "1"にセットされている場合、FPU演算命令実行時、 "0"にクリアされます。	R (注3)
29	CV 無効演算例外要因ビット	0：無効演算例外の発生なし 1：無効演算例外の発生あり "1"にセットされている場合、FPU演算命令実行時、 "0"にクリアされます。	R (注3)
30, 31	RM 丸めモード	00：最近傍への丸め 01：0方向への丸め 10：+∞方向への丸め 11：-∞方向への丸め	R W

注1．「EIT処理実行なし」とは、各例外が発生した場合においても、b17～b21のイネーブルビットを"0"にし、マスクしていたことにより、EIT処理を実行しないことを言います。例えば、2つの例外が同時に発生し、それぞれのイネーブルビットの設定が異なる場合(どちらかがEIT処理の実行を設定)、EIT処理を行います。このときは、イネーブルビットの設定に関係なくこの2つのフラグは変化しません。
注2．DN＝0のときに非正規化数がオペランドに与えられると、非実装例外が発生します。
注3．"0"書き込み"0"クリア、"1"書き込み無効(書き込み前の値を保持)です。

1.3.6 浮動小数点例外(FPE)

浮動小数点例外(FPE：Floating Point Exception)は、IEEE754規格で規定された5つの例外(OVF/UDF/IXCT/DIV0/IVLD)の他に、非実装例外(UIPL)を検出した場合に発生します。以下に、各例外処理の概要を示します。

(1) オーバフロー例外（OVF）

演算結果の絶対値が、浮動小数点フォーマットで表現可能な値よりも大きくなった場合に発生します。以下にOVFが発生したときの演算結果を示します。

丸めモード	結果の符号	演算結果(デスティネーションレジスタの内容)	
		オーバフロー例外による EIT処理マスク時(注1)	オーバフロー例外による EIT処理の実行設定時(注2)
-∞	+	+MAX	変化なし
	-	-∞	
+∞	+	+∞	
	-	-MAX	
0	+	+MAX	
	-	-MAX	
最近傍	+	+∞	
	-	-∞	

注1．オーバフロー例外イネーブル(EO)ビット(FPSRレジスタのb20)＝"0"のとき
注2．オーバフロー例外イネーブル(EO)ビット(FPSRレジスタのb20)＝"1"のとき
注．・オーバフロー例外によるEIT処理マスク時に、オーバフロー例外が発生すると、同時に精度異常例外が発生します。
・+MAX＝H'7F7F FFFF, -MAX＝H'FF7F FFFF

(2) アンダフロー例外（UDF）

演算結果の絶対値が、浮動小数点フォーマットの正規化数で表現可能な値よりも小さくなった場合に発生します。以下にUDFが発生したときの演算結果を示します。

演算結果(デスティネーションレジスタの内容)	
アンダフロー例外によるEIT処理マスク時(注1)	アンダフロー例外によるEIT処理の実行設定時(注2)
DN＝0：非実装例外が発生します。 DN＝1：0を返します。	変化なし

注1．アンダフロー例外イネーブル(EU)ビット(FPSRレジスタのb18)＝"0"のとき
注2．アンダフロー例外イネーブル(EU)ビット(FPSRレジスタのb18)＝"1"のとき

(3) 精度異常例外（IXCT）

無限の有効桁を持つと仮定して計算したときの結果と、演算結果が異なっていたときに発生します。以下にIXCTの発生条件と、演算結果を示します。

発生条件	演算結果(デスティネーションレジスタの内容)	
	精度異常例外による EIT処理マスク時(注1)	精度異常例外による EIT処理の実行設定時(注2)
OVF例外マスク状態での オーバフロー発生	OVF例外の演算結果参照	変化なし
丸めの発生	丸め後の値	変化なし

注1．精度異常例外イネーブル(EX)ビット(FPSRレジスタのb17)＝"0"のとき
注2．精度異常例外イネーブル(EX)ビット(FPSRレジスタのb17)＝"1"のとき

(4) ゼロ除算例外（DIV0）

0でない有限数を0で割ったときに発生します。以下にDIV0が発生したときの演算結果を示します。

被除数	演算結果(デスティネーションレジスタの内容)	
	0除算例外によるEIT処理マスク時（注1）	0除算例外によるEIT処理の実行設定時（注2）
0でない有限数	±∞(符号は除数、被除数の符号の排他的倫理論となる)	変化なし

注1．ゼロ除算例外イネーブル(EZ)ビット(FPSRレジスタのb19)＝"0"のとき
注2．ゼロ除算例外イネーブル(EZ)ビット(FPSRレジスタのb19)＝"1"のとき

なお、次の場合は、DIV0は発生しません。ご注意ください。

被除数	動作
0	無効演算例外発生
∞	例外は発生しない(結果は"∞")

(5) 無効演算例外（IVLD）

無効な演算が実行されたときに発生します。以下にIVLDの発生条件と、演算結果を示します。

発生条件		演算結果(デスティネーションレジスタの内容)	
		無効演算例外によるEIT処理マスク時(注1)	無効演算例外によるEIT処理の実行設定時(注2)
SNaNオペランドに対する演算		QNaN	変化なし
+∞-(∞)、-∞-(-∞)			
0×∞			
0÷0、∞÷∞			
整数変換がオーバフローしたとき NaN、∞を整数変換したとき	FTOI命令実行時	変換前の符号ビットが 0のときはH'7FFF FFFF 1のときはH'8000 0000を返す	
	FTOS命令実行時	変換前の符号ビットが 0のときはH'0000 7FFF 1のときはH'FFF 8000を返す	
<、>の比較をNaNに対して行ったとき		比較結果(比較不能)	

注．・NaN(非数：Not a Number)
SNaN(Signaling NaN)：少数部の最上位ビットが0であるNaNです。SNaNを演算のソースオペランドとして使用すると、無効演算例外を発生します。変数の初期値として使用することにより、プログラムバグの発見に役立ちます。なお、SNaNはハードウエアが生成することはありません。
QNaN(Quiet NaN)：少数部の最上位ビットが1であるNaNです。QNaNを演算のソースオペランドとして使用しても、無効演算例外は発生しません(比較、フォーマット変換を除く)。演算によって伝播するため、EIT処理を実行せずに結果だけを見てデバッグを行うことができます。なお、QNaNは演算によりハードウエアが生成します。
注1．無効演算例外イネーブル(EV)ビット(FPSRレジスタのb21)＝"0"のとき
注2．無効演算例外イネーブル(EV)ビット(FPSRレジスタのb21)＝"1"のとき

(6) 非実装例外（UIPL）

非正規化数の0フラッシュ(DN)ビット(FPSRレジスタのb23)＝"0"であり、非正規化数が演算オペランドとして与えられたときに発生します(注1)。

UIPLにはイネーブルビットがないため、発生時UIPLをマスクすることはできません。

デスティネーションレジスタは変化しません。

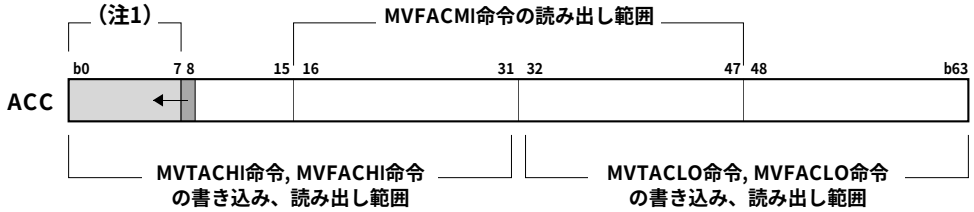
注．・演算の中間結果が非正規化数となったときはUDFが発生し、このときDNビット(FPSRレジスタのb23)＝"0"ならばUIPLが発生します。

1.4 アキュムレータ

アキュムレータ(ACC)は、DSP機能用命令で使用される56ビットのレジスタです。
読み出し時や書き込み時には64ビットのレジスタとして扱われ、読み出し時にはb8の値が符号拡張されます。書き込み時にはb0～b7は無視されます。また、アキュムレータは乗算命令「MUL」でも使用され、この命令実行の際はアキュムレータの値が破壊されるので注意してください。

アキュムレータへの書き込みには「MVTACHI命令」と「MVTACLO命令」を使用します。「MVTACHI命令」は上位側32ビット(b0～b31)に、「MVTACLO命令」は下位側32ビット(b32～b63)にデータを書き込みます。

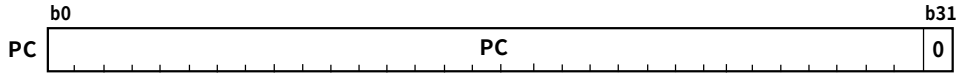
読み出しには「MVFACHI命令」、「MVFACLO命令」および「MVFACMI命令」を使用します。「MVFACHI命令」は上位側32ビット(b0～b31)を、「MVFACLO命令」は下位側32ビット(b32～b63)を、また「MVFACMI命令」は中央の32ビット(b16～b47)のデータをそれぞれ読み出します。
リセット解除時、ACCの値は不定です。



注1. b0～b7は、ビット8の値を符号拡張した値が常に読み出されます。この部分への書き込みは無視されます。

1.5 プログラムカウンタ

プログラムカウンタ(PC)は32ビットのカウンタで、現在実行中の命令アドレスを保持します。M32R-FPUの命令は偶数アドレスから始まるため、LSB(b31)は"0"になります。
リセット解除時、PCは"H'0000 0000"です。



1.6 データフォーマット

1.6.1 データタイプ

M32R-FPUの命令セットで扱えるデータタイプは、符号付き、または符号なしの8, 16, 32ビット整数及び単精度浮動小数点です。符号付き整数の値は2の補数で表現されます。

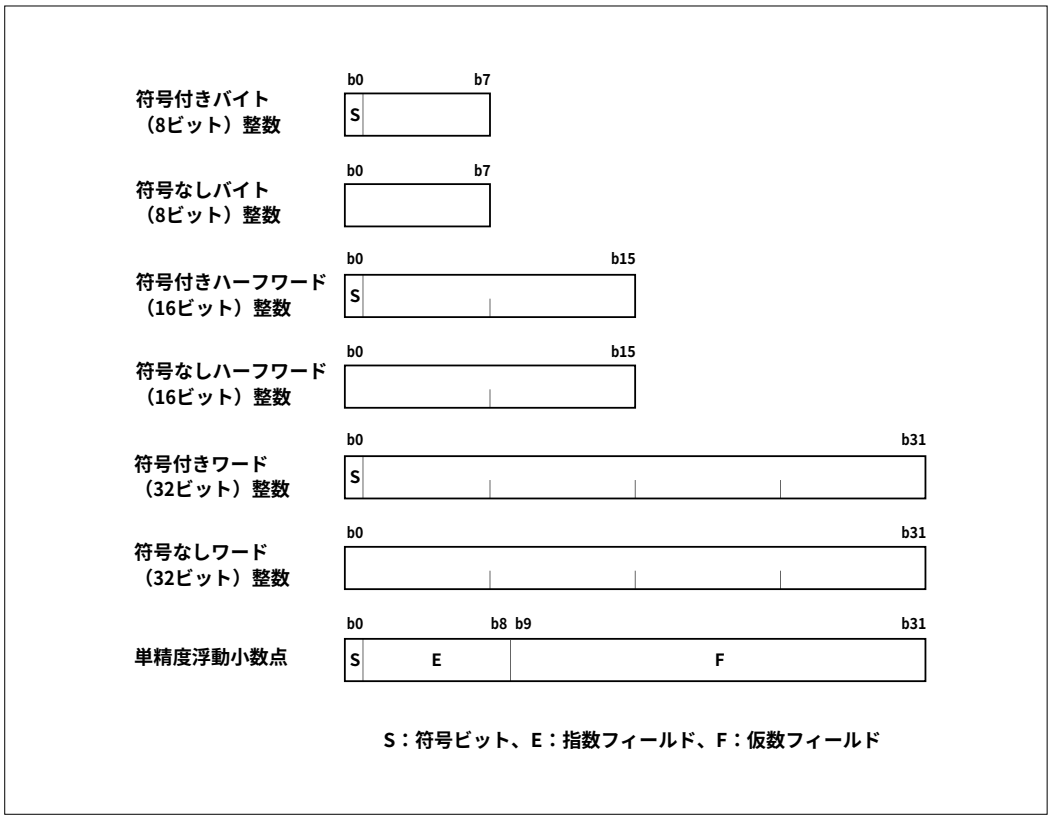


図1.6.1 データタイプ

1.6.2 データフォーマット

(1) M32R-FPU レジスタ上のデータフォーマット

M32R-FPUのレジスタ上でのデータサイズは常にワード(32ビット)です。

メモリ上のバイト(8ビット)、ハーフワード(16ビット)のデータをロードする場合は、ワード(32ビット)データに符号拡張(LDB，LDH命令)又はゼロ拡張(LDUB，LDUH命令)後、レジスタに格納されます。

M32R-FPUのレジスタ上のデータをメモリにストアする場合は、ST命令ではレジスタ上の32ビットデータ、STH命令ではLSB側の16ビットデータ、STB命令ではLSB側8ビットデータをそれぞれメモリにストアします。

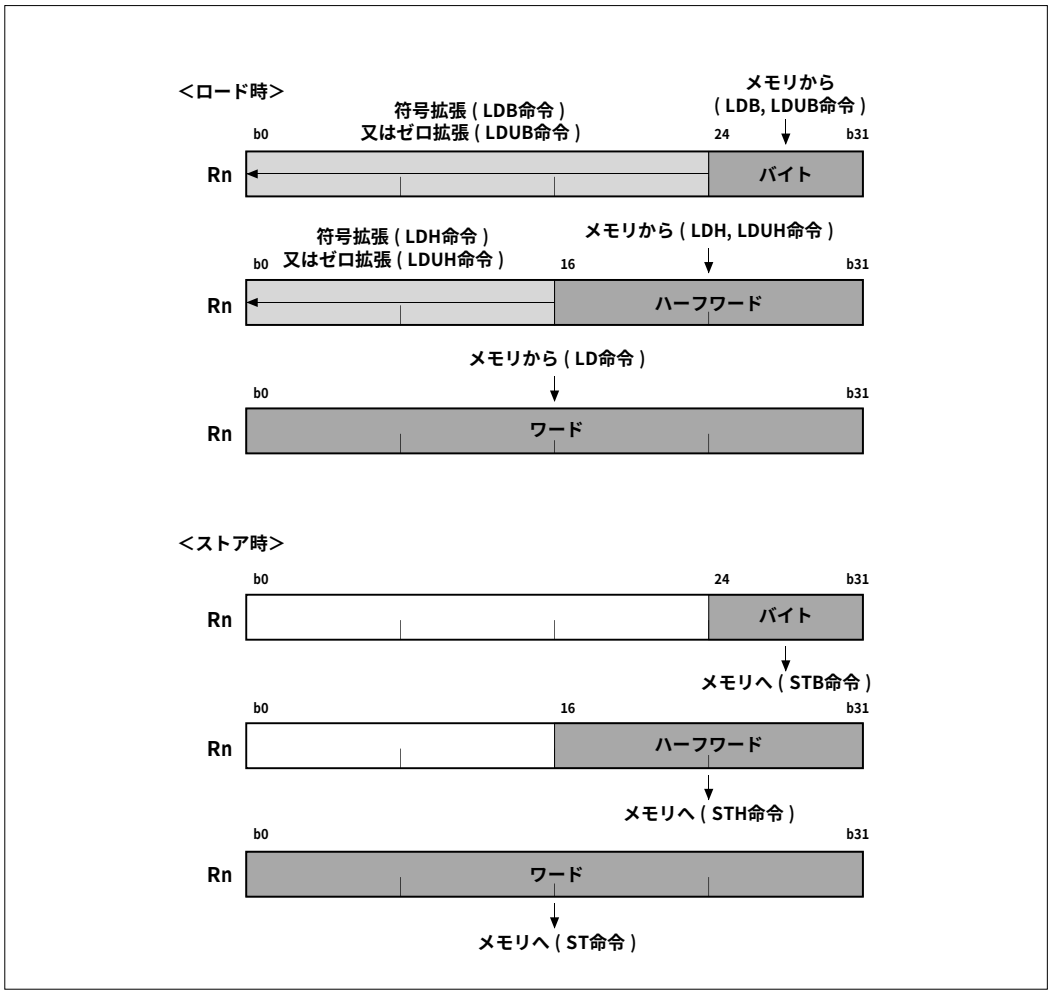


図1.6.2 レジスタ上のデータフォーマット

(2) メモリ上のデータフォーマット

メモリ上でのデータサイズはバイト(8ビット)、ハーフワード(16ビット)、ワード(32ビット)の3種類です。バイトデータは任意のアドレスに配置できますが、ハーフワードデータはハーフワード境界(アドレスの最下位ビットが"0"の番地)、またワードデータはワード境界(アドレスの下位2ビットが"00"の番地)に配置されなければなりません。この境界をまたぐメモリデータをアクセスしようとするとアドレス例外が発生します。

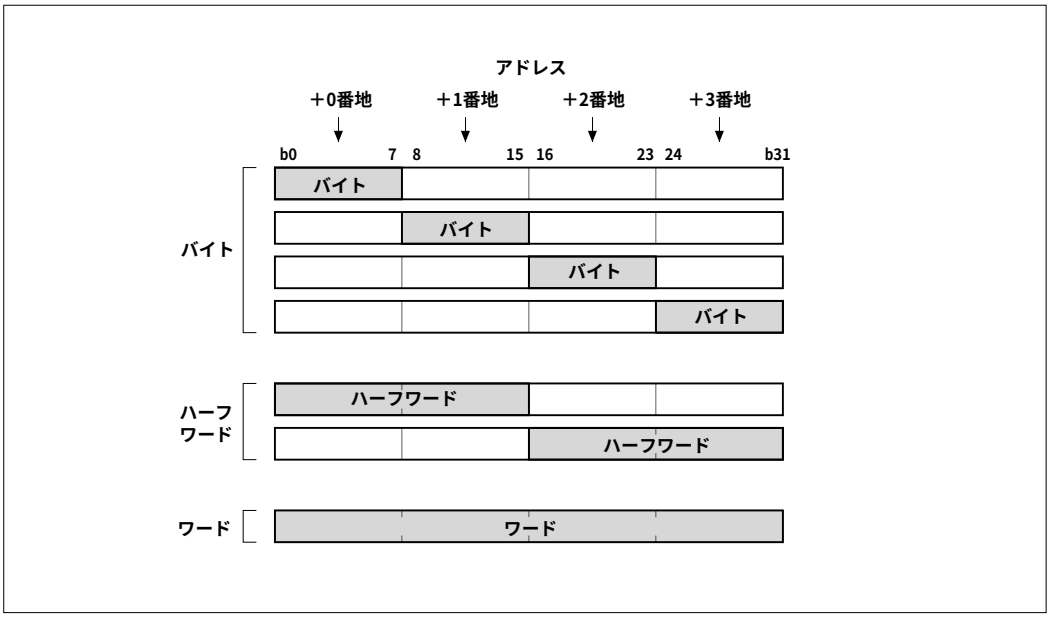


図1.6.3 メモリ上のデータフォーマット

1.7 アドレッシングモード

M32R-FPUのアドレッシングモードには、以下のものがあります。

(1) レジスタ直接〔表記：RまたはCR〕

操作の対象として汎用レジスタ又は制御レジスタを指定するもの。

(2) レジスタ間接〔表記：@R〕

レジスタ値をアドレスとするもの(すべてのロード／ストア命令で指定可能)。

(3) レジスタ相対間接〔表記：@(disp,R)〕

(レジスタ値)＋(16ビットのディスプレースメントを32ビットに符号拡張した値)をアドレスとするもの。

(4) レジスタ間接＋レジスタ更新

- レジスタ値を＋4する**〔表記：@R+〕**
  レジスタ値をアドレスとするもの。その後、レジスタ値を＋4する(LD命令でのみ指定可能)。
- レジスタ値を＋2する**〔表記：@R+〕(M32R-FPU拡張アドレッシングモード)**
  レジスタ値をアドレスとするもの。その後、レジスタ値を＋2する(STH命令でのみ指定可能)。
- レジスタ値を＋4する**〔表記：@+R〕**
  レジスタ値を＋4する。その後、レジスタ値をアドレスとするもの(ST命令でのみ指定可能)。
- レジスタ値を－4する**〔表記：@-R〕**
  レジスタ値を－4する。その後、レジスタ値をアドレスとするもの(ST命令でのみ指定可能)。

(5) イミディエート〔表記：#imm 〕

4, 5, 8, 16 又は 24ビットの即値(値の扱いは各命令の詳細説明参照)。

(6) PC相対〔表記：pcdisp 〕

(PCの値)＋(8ビット, 16ビット又は24ビットのディスプレースメントを32ビットに符号拡張して左へ2ビットシフトした値)をアドレスとするもの。

第２章

命令セット

2.1 命令セット概要

M32R-FPUの命令数は、100です。RISCアーキテクチャを採用しており、メモリアクセスは基本的にロード命令とストア命令で行います。また、各種の演算はレジスタ間演算で実行します。さらに、ロード＆アドレス更新、ストア＆アドレス更新といった複合命令もサポートしています。

2.1.1 ロード／ストア命令

メモリ～レジスタ間のデータ転送を行います。

LD	Load
LDB	Load byte
LDUB	Load unsigned byte
LDH	Load halfword
LDUH	Load unsigned halfword
LOCK	Load locked
ST	Store
STB	Store byte
STH	Store halfword
UNLOCK	Store unlocked

ロード／ストア命令におけるアドレッシングモードは、次の3種類を指定可能です。

(1) レジスタ間接

レジスタ値をアドレスとするもの(すべてのロード／ストア命令で指定可能)。

(2) レジスタ相対間接

(レジスタ値)＋(16ビットのディスプレースメントを32ビットに符号拡張した値)をアドレスとするもの(LOCK,UNLOCK命令以外の命令で指定可能)。

(3) レジスタ間接＋レジスタ更新

- レジスタ値を＋4する**〔表記：@R+〕**
  レジスタ値をアドレスとするもの。その後、レジスタ値を＋4する
  (LD命令でのみ指定可能)。
- レジスタ値を＋2する**〔表記：@R+〕(M32R-FPU拡張アドレッシングモード)**
  レジスタ値をアドレスとするもの。その後、レジスタ値を＋2する
  (STH命令でのみ指定可能)。
- レジスタ値を＋4.する**〔表記：@+R〕**
  レジスタ値を＋4する。その後、レジスタ値をアドレスとするもの
  (ST命令でのみ指定可能)。
- レジスタ値を－4する**〔表記：@-R〕**
  レジスタ値を－4する。その後、レジスタ値をアドレスとするもの
  (ST命令でのみ指定可能)。

いずれのアドレッシングモードを使用した場合でも、メモリ上のデータフォーマットの規則を守る必要があります。ハーフワード、ワードサイズのデータをアクセスする場合には、それぞれハーフワードアライメント、ワードアライメントのとれたアドレスを指定します(ハーフワードサイズの場合にはアドレスの下位２ビットは"00"か"10"、ワードサイズの場合にはアドレスの下位２ビットは"00")。アライメントのとれていないアドレスを指定するとアドレス例外が発生します。

ロード命令でバイトデータ、ハーフワードデータをアクセスした場合には、上位ビットが符号拡張又はゼロ拡張された32ビットデータとしてレジスタに格納されます。

2.1.2 転送命令

レジスタ～レジスタ間又はレジスタ～イミディエート(即値)の転送を行います。

LD24	Load 24-bit immediate
LDI	Load immediate
MV	Move register
MVFC	Move from control register
MVTC	Move to control register
SETH	Set high-order 16-bit

2.1.3 演算命令

レジスタ～レジスタ間で比較、算術論理演算、乗除算、シフトなどを行います。

■比較

CMP	Compare
CMPI	Compare immediate
CMPU	Compare unsigned
CMPUI	Compare unsigned immediate

■算術演算

ADD	Add
ADD3	Add 3-operand
ADDI	Add immediate
ADDV	Add with overflow checking
ADDV3	Add 3-operand with overflow checking
ADDX	Add with carry
NEG	Negate
SUB	Subtract
SUBV	Subtract with overflow checking
SUBX	Subtract with borrow

■論理演算

AND	AND
AND3	AND 3-operand
NOT	Logical NOT
OR	OR
OR3	OR 3-operand
XOR	Exclusive OR
XOR3	Exclusive OR 3-operand

■乗除算

DIV	Divide
DIVU	Divide unsigned
MUL	Multiply
REM	Remainder
REMU	Remainder unsigned

■シフト

SLL	Shift left logical
SLL3	Shift left logical 3-operand
SLLI	Shift left logical immediate
SRA	Shift right arithmetic
SRA3	Shift right arithmetic 3-operand
SRAI	Shift right arithmetic immediate
SRL	Shift right logical
SRL3	Shift right logical 3-operand
SRLI	Shift right logical immediate

2.1.4 分岐命令

プログラムの流れを変えるための命令です。

BC	Branch on C-bit
BEQ	Branch on equal to
BEQZ	Branch on equal to zero
BGEZ	Branch on greater than or equal to zero
BGTZ	Branch on greater than zero
BL	Branch and link
BLEZ	Branch on less than or equal to zero
BLTZ	Branch on less than zero
BNC	Branch on not C-bit
BNE	Branch on not equal to
BNEZ	Branch on not equal to zero
BRA	Branch
JL	Jump and link
JMP	Jump
NOP	No operation

分岐先として指定できるのはワードアライメントのとれたアドレス(ワード境界)だけです。

BRA, BL, BC, BNC命令のアドレッシングモードは、8ビットか24ビットのイミディエート値を指定できます。またBEQ, BNE, BEQZ, BNEZ, BGTZ, BLTZ, BGEZ, BLEZ命令のアドレッシングモードは、16ビットのイミディエート値です。

JMP, JL命令は、レジスタの値が分岐先アドレスとなります。ただし、レジスタの下位2ビットの値は無視されます。

その他の分岐命令は、(分岐命令のPC値)＋(符号拡張されたイミディエート値を左に2ビットシフトした値)が分岐先アドレスとなります。ただし、加算が行われる際のPC値の下位2ビットは"00"にクリアされます。たとえば図2.1.1で「命令A」または「命令B」が分岐命令で、「命令G」に分岐する場合、いずれもイミディエート値は4になります。

サブルーチンコール用のJL, BL命令は、分岐と同時に戻り先PC値がR14に格納されます。R14に格納される値は(分岐命令のPC値＋4)で、PC値の下位2ビットは"00"にクリアされます。

例えば図2.1.1で「命令A」又は「命令B」がJL, BL命令であった場合、いずれも戻り先は「命令C」となります。

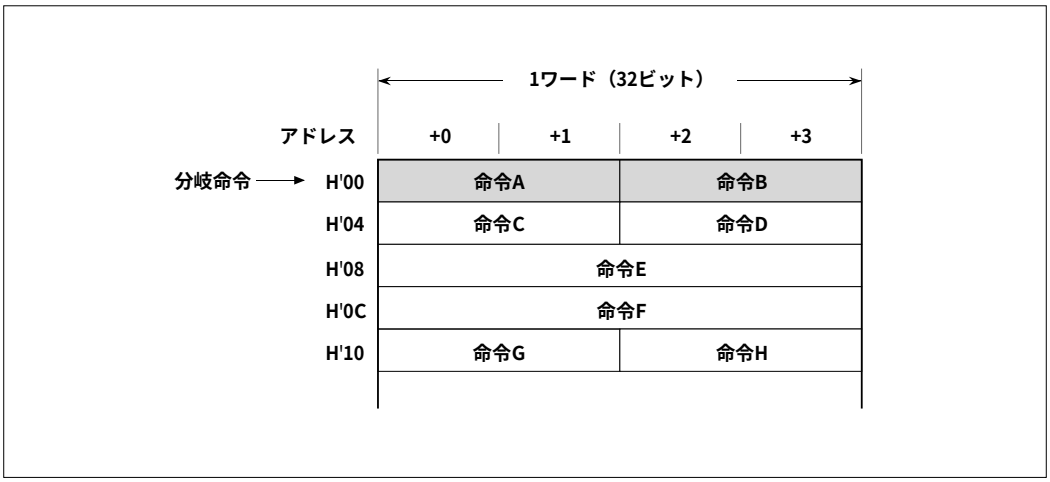


図2.1.1 分岐命令の分岐先

2.1.5 EIT関連命令

EIT関連命令は、M32RのEIT事象(Exception：例外, Interrupt：割り込み, Trap：トラップ)のための命令です。EIT関連命令にはトラップの起動命令とEIT処理からの復帰命令があります。

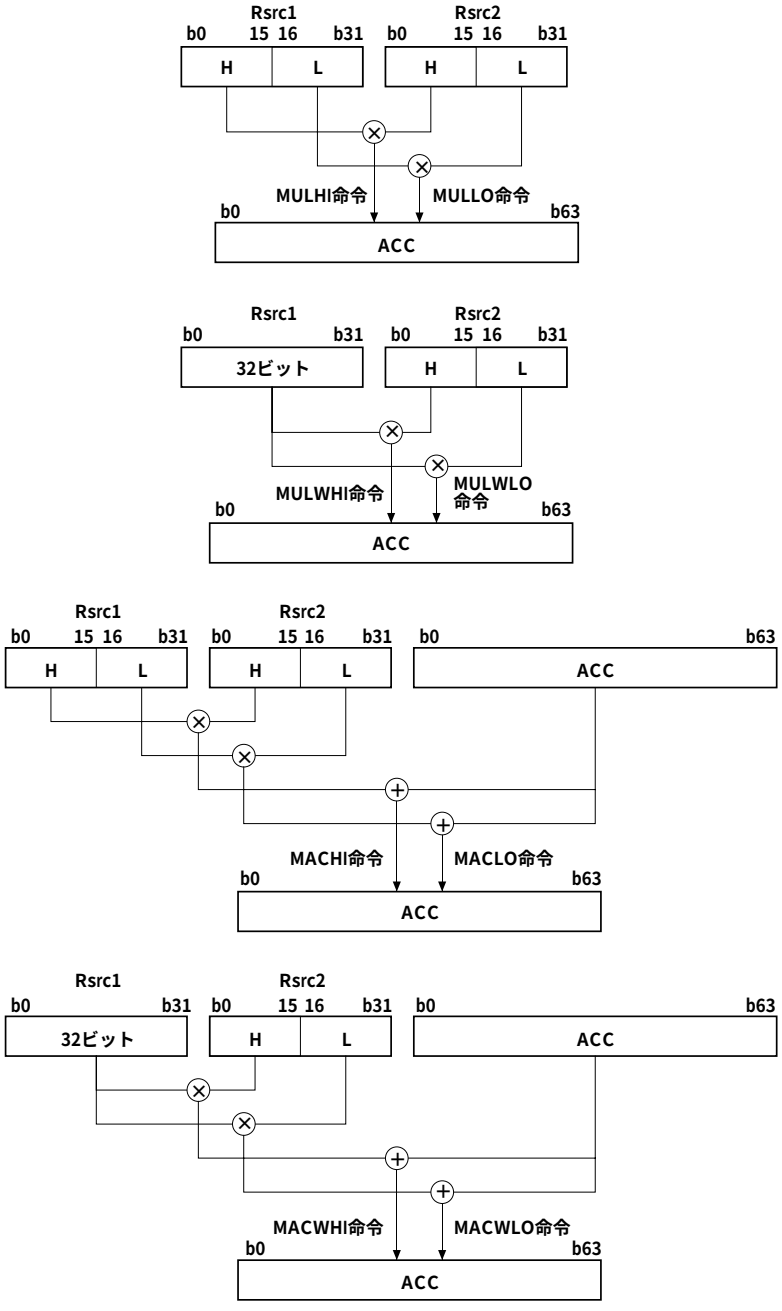
TRAP	Trap
RTE	Return from EIT

2.1.6 DSP機能用命令

32ビット×16ビット、16ビット×16ビットの乗算や積和演算を行います。また、アキュムレータ内のデータの丸めやアキュムレータ～汎用レジスタ間の転送を行います。

MACHI	Multiply-accumulate high-order halfwords
MACLO	Multiply-accumulate low-order halfwords
MACWHI	Multiply-accumulate word and high-order halfword
MACWLO	Multiply-accumulate word and low-order halfword
MULHI	Multiply high-order halfwords
MULLO	Multiply low-order halfwords
MULWHI	Multiply word and high-order halfword
MULWLO	Multiply word and low-order halfword
MVFACHI	Move high-order word from accumulator
MVFACLO	Move low-order word from accumulator
MVFACMI	Move middle-order word from accumulator
MVTACHI	Move high-order word to accumulator
MVTACLO	Move low-order word to accumulator
RAC	Round accumulator
RACH	Round accumulator halfword

次ページにこれらの命令の動作概要を示します。



注. 実際のDSP機能用命令の演算では結果の格納位置の調整や符号拡張が行われます。
詳細については「第3章 命令」を参照してください。

図2.1.2 DSP機能用命令の動作 1（乗算、積和演算）

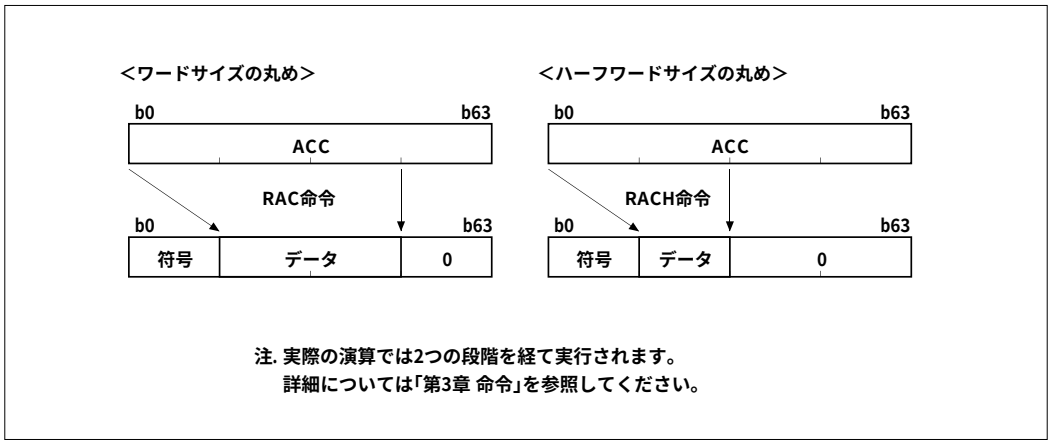


図2.1.3 DSP機能用命令の動作 2（丸め操作）

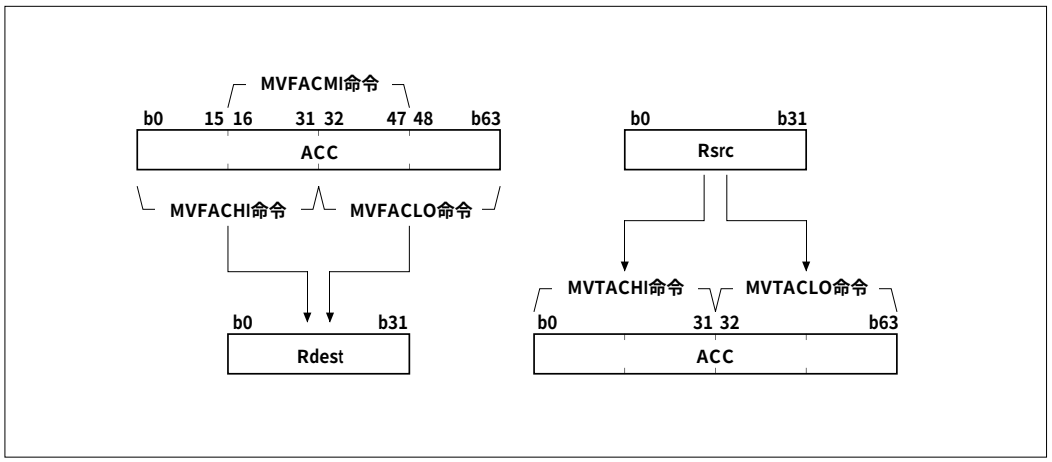


図2.1.4 DSP機能用命令の動作 3（アキュムレータ～レジスタ間転送）

2.1.7 浮動小数点命令

浮動小数点演算を行います。

FADD	Floating-point add
FSUB	Floating-point subtract
FMUL	Floating-point multiply
FDIV	Floating-point divide
FMADD	Floating-point multiply and add
FMSUB	Floating-point multiply and subtract
ITOF	Integer to float
UTOF	Unsigned integer to float
FTOI	Float to integer
FTOS	Float to short
FCMP	Floating-point compare
FCMPE	Floating-point compare with exception if unordered

2.1.8 ビット操作命令

レジスタ又はメモリの指定されたビットを操作します。

BSET	Bit set
BCLR	Bit clear
BTST	Bit test
SETPSW	Set PSW
CLRPSW	Clear PSW

2.2 命令フォーマット

M32R-FPUの命令フォーマットは2種類あり、1つは32ビットワード境界内に格納された2つの16ビット命令、もう1つは単一の32ビット命令です(図2.2.1参照)。

M32R CPUの基本命令フォーマットを図2.2.2に示します。

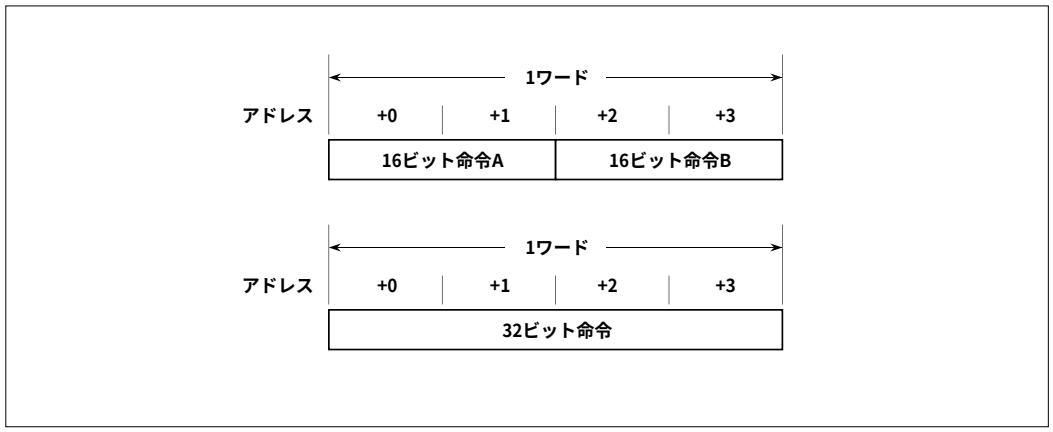


図2.2.1 16ビット命令と32ビット命令

■16ビット命令

<命令フォーマット>				<命令の動作>	<命令の例>	
op1	R_1	op2	R_2	$R_1 = R_1$ op R_2	AND	Rdest, Rsrc
op1	R_1	c		$R_1 = R_1$ op c	ADDI	Rdest, #imm8
op1	cond	c		Branch (Short Displacement)	BC	pcdisp8

■32ビット命令

<命令フォーマット>								<命令の動作>	<命令の例>	
op1	R_1	op2	R_2	c				$R_1 = R_2$ op c	SRL3	Rdest, Rsrc, #imm16
op1	R_1	op2	R_2	c				Compare and Branch	BEQ	Rsrc1, Rsrc2, pcdisp16
op1	R_1	c						$R_1 = R_1$ op c	LD24	Rdest, #imm24
op1	cond	c						Branch	BC	pcdisp24
op1	R_s	op2	0000	op3	Rd	op4	0000	Floating-point 2-operand (Rd=op(Rs))	FMADD	Rdest, Rsrc1, Rsrc2
op1	Rs1	op2	Rs2	op3	Rd	op4	0000	Floating-point 3-operand (Rd=Rs1 op Rs2)	FTOI	Rdest, Rsrc

図2.2.2 M32R CPUの基本命令フォーマット

32ビット命令のMSB(Most Significant Bit)は常に"1"です。16ビット命令の場合、上位のハーフワード境界に存在する命令のMSBは常に"0"ですが、それにつづく16ビット命令は、その命令のMSBの値によって処理が異なります。

図2.2.3において「命令B」のMSBが"0"の場合、「命令A」と「命令B」はシーケンシャルに実行されますが、「命令B」のMSBが"1"の場合は、「命令A」と「命令B」はパラレルに実行されます。

ただし現状のインプリメンテーションレベルでは「NOP命令」のみ、このパラレル実行をサポートします(例としてワードアライメント調整のためのNOP命令は、アセンブラが自動的にMSBを1にしたNOP命令に変えるため、M32R-FPUは見かけ上クロックを消費しない命令としてこれを処理します)。

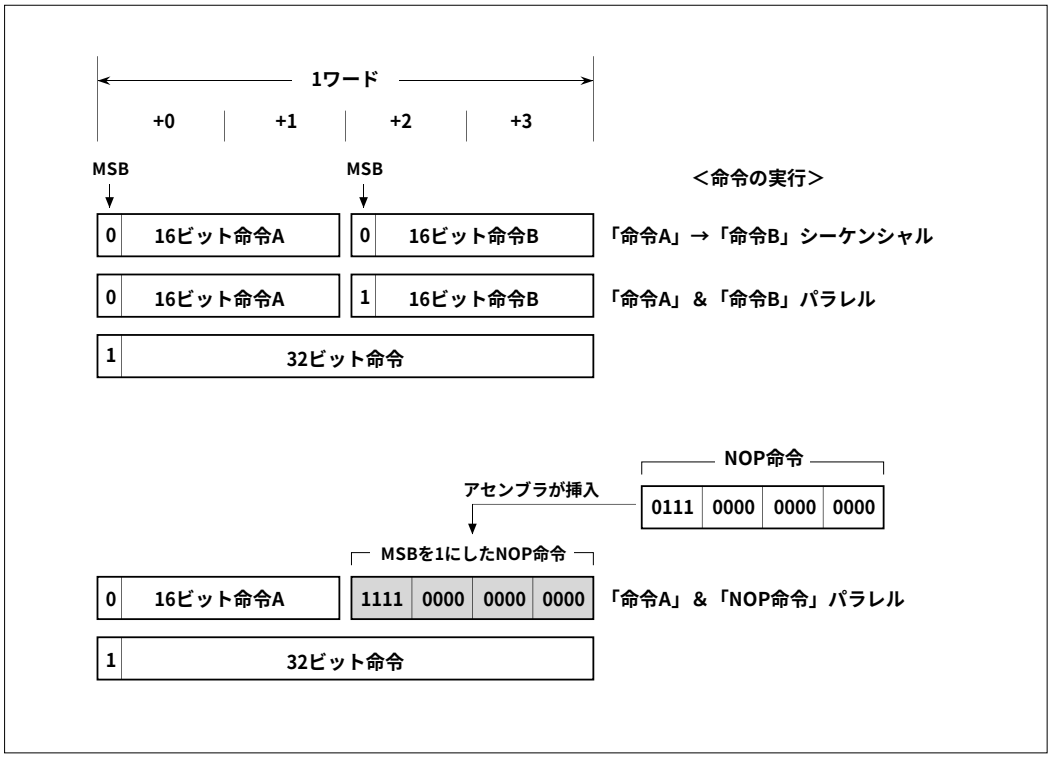


図2.2.3 16ビット命令の処理

＊空きページです＊

第３章

命令

3.1 命令の記述方法

命令詳細説明で記述される各項目は以下のとおりです。

【ニーモニック】

ニーモニックは、命令とそれに続く命令のオペランド(操作対象)の記述で構成されます。

表3.1.1 オペランド一覧

表記(注)	アドレッシングモード	命令の操作対象
R	レジスタ直接	M32Rの汎用レジスタ(R0～R15)の内容
CR	制御レジスタ	M32Rの制御レジスタ(CR0=PSW, CR1=CBR, CR2=SPI, CR3=SPU, CR6=BPC, CR7=FPSR)の内容
@R	レジスタ間接	レジスタ値をアドレスとするメモリの内容
@(disp,R)	レジスタ相対間接	(レジスタ値)＋(16ビットのディスプレースメントを32ビットに符号拡張した値)をアドレスとするメモリの内容
@R+	レジスタ間接 ＋レジスタ更新	レジスタ値を＋4する(レジスタ値をアドレスとするメモリの内容。その後、レジスタ値を＋4する。)
@+R	レジスタ間接 ＋レジスタ更新	レジスタ値を＋4する(レジスタ値を＋4する。その後、レジスタ値をアドレスとするメモリの内容。)
@-R	レジスタ間接 ＋レジスタ更新	レジスタ値を－4する(レジスタ値を－4する。その後、レジスタ値をアドレスとするメモリの内容。)
#imm	イミディエート	即値(値の扱いは各命令の詳細説明参照)
#bitpos	ビットポディション	バイトデータのビット位置の内容
pcdisp	PC相対	(PCの値)＋(8ビット, 16ビット又は24ビットのディスプレースメントを32ビットに符号拡張して左へ2ビットシフトした値)をアドレスとするメモリの内容

注. ・オペランド表記において、Rsrc, Rdestと表記した場合のsrc, destには任意の汎用レジスタ番号(0～15)が入ります。
また、CRsrc, CRdestと表記した場合のsrc, destには任意の制御レジスタ番号(0～3, 6, 7)が入ります。

【動作】

命令の動作概要とC言語に準じた表記での動作説明です。

表3.1.2 動作表記法１(演算子)

演算子	意味
+	加算(二項演算子)
-	減算(二項演算子)
*	乗算(二項演算子)
/	除算(二項演算子)
%	剰余算(二項演算子)
++	インクリメント(単項演算子)
--	デクリメント(単項演算子)

表3.1.3　動作表記法２（演算子）

演算子	意味
-	符号の反転（単項演算子）
=	右辺から左辺への代入（代入演算子）
+=	左辺と右辺の変数を加算し、左辺に代入（代入演算子）
-=	左辺の変数から右辺の変数を減算し、左辺に代入（代入演算子）
>	より大（関係演算子）
<	より小（関係演算子）
>=	より大か等しい（関係演算子）
<=	より小か等しい（関係演算子）
==	等しい（関係演算子）
!=	等しくない（関係演算子）
&&	AND（論理演算子）
||	OR（論理演算子）
!	NOT（論理演算子）
?:	条件式をつくる（条件演算子）

表3.1.4　動作表記法3（ビット演算子）

演算子	意味
<<	ビットを左にシフト
>>	ビットを右にシフト
&	ビット積（AND）
|	ビット和（OR）
^	ビット排他的論理和（EXOR）
~	ビットの反転

表3.1.5　データタイプ（整数）

表現	符号の有無	ビット長	数値の範囲
signed char	あり	8	－128～＋127
signed short	あり	16	－32,768～＋32,767
signed int	あり	32	－2,147,483,648～＋2,147,483,647
unsigned char	なし	8	0～255
unsigned short	なし	16	0～655,535
unsigned int	なし	32	0～4,294,967,295
signed64bit	あり	64	符号付き64ビット整数（アキュムレータ操作時）

表3.1.6　データタイプ（浮動小数点）

表現	浮動小数点フォーマット
float	単精度フォーマット

【機能】

各命令の機能の詳細を説明しています。また、その命令の実行で起こるPSWレジスタ中の条件ビット(C)の変化を記述しています。

【発生EIT】

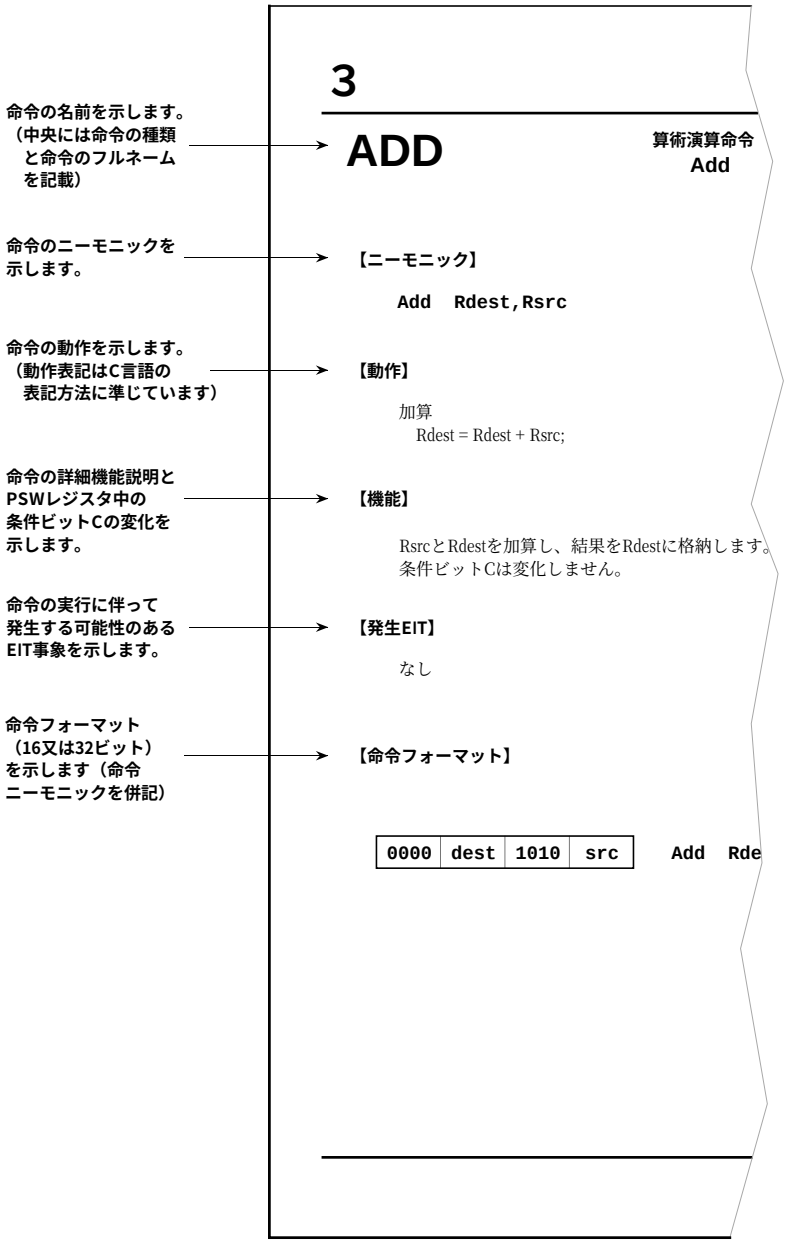
各命令の実行で発生する可能性のあるEIT事象(例外、割り込み、トラップ)を示しています。命令実行で発生する可能性があるのはアドレス例外、浮動小数点例外及びトラップです。

【命令フォーマット】

16ビット又は32ビットの命令ビットパターンを示します。src, destフィールドには対応するレジスタ番号が、また imm, dispにはそれぞれイミディエート(即値)、ディスプレースメントが入ります(代入可能な数値の大きさはフィールドの幅で決まります)。命令フォーマットについては、「2.2　命令フォーマット」を参照ください。

3.2 命令詳細説明

次ページよりM32R-FPUの各命令の詳細説明を示します。各命令はアルファベット順に掲載されています。各ページの構成は、下記のとおりです。



ADD

算術演算命令

Add

ADD

【ニーモニック】

```
ADD  Rdest,Rsrc
```

【動作】

加算

Rdest = Rdest + Rsrc;

【機能】

RsrcとRdestを加算し、結果をRdestに格納します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0000	dest	1010	src

ADD Rdest,Rsrc

ADD3 算術演算命令 Add 3-operand ADD3

【ニーモニック】

```
ADD3  Rdest,Rsrc,#imm16
```

【動作】

加算

Rdest = Rsrc + (signed short) imm16;

【機能】

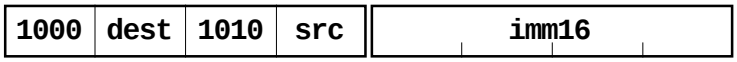
Rsrcに16ビット即値を加算し、結果をRdestに格納します。16ビット即値は、演算前に32ビットに符号拡張されます。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
ADD3 Rdest,Rsrc,#imm16
```

ADDI

算術演算命令

Add immediate

ADDI

【ニーモニック】

```
ADDI  Rdest,#imm8
```

【動作】

加算

Rdest = Rdest + (signed char) imm8;

【機能】

Rdestに8ビット即値を加算し、結果をRdestに格納します。8ビット即値は、演算前に32ビットに符号拡張されます。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0100	dest	imm8

ADDI Rdest,#imm8

ADDV

算術演算命令

Add with overflow checking

ADDV

【ニーモニック】

```
ADDV   Rdest,Rsrc
```

【動作】

加算

```
Rdest = ( signed ) Rdest + ( signed ) Rsrc;
C = overflow ? 1 : 0;
```

【機能】

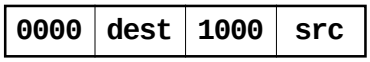
RdestとRsrcを加算し、結果をRdestに格納します。

条件ビット(C)は加算結果がオーバフローした場合にセットされ、それ以外の場合はクリアされます。

【発生EIT】

なし

【命令フォーマット】



ADDV Rdest,Rsrc

ADDV3 *算術演算命令* Add 3-operand with overflow checking ADDV3

【ニーモニック】

```
ADDV3  Rdest,Rsrc,#imm16
```

【動作】

加算

```
Rdest = ( signed ) Rsrc + ( signed ) ( ( signed short ) imm16 );
C = overflow ? 1 : 0;
```

【機能】

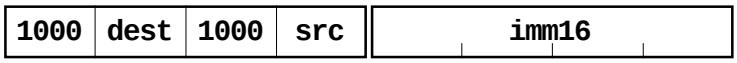
Rsrcと16ビット即値を加算し、結果をRdestに格納します。16ビット即値は演算前に32ビットに符号拡張されます。

条件ビット(C)は加算結果がオーバフローした場合にセットされ、それ以外の場合はクリアされます。

【発生EIT】

なし

【命令フォーマット】




```
ADDV3 Rdest,Rsrc,#imm16
```

ADDX

算術演算命令

Add with carry

ADDX

【ニーモニック】

```
ADDX  Rdest,Rsrc
```

【動作】

加算

```
Rdest = ( unsigned ) Rdest + ( unsigned ) Rsrc + C;
C = carry_out ? 1 : 0;
```

【機能】

Rdestに(Rsrc＋条件ビット(C)の値)を加算し、結果をRdestに格納します。

条件ビット(C)は加算結果が32ビット符号なし整数で表せないときにセットされ、それ以外の場合はクリアされます。

【発生EIT】

なし

【命令フォーマット】

0000	dest	1001	src

ADDX Rdest,Rsrc

AND

論理演算命令

AND

AND

【ニーモニック】

```
AND  Rdest,Rsrc
```

【動作】

論理積

Rdest = Rdest & Rsrc;

【機能】

RdestとRsrcの対応するビットの論理積をとり、結果をRdestに格納します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0000	dest	1100	src

AND Rdest,Rsrc

AND3

論理演算命令
AND 3-operand

AND3

【ニーモニック】

```
AND3  Rdest,Rsrc,#imm16
```

【動作】

論理積
Rdest = Rsrc & (unsigned short) imm16;

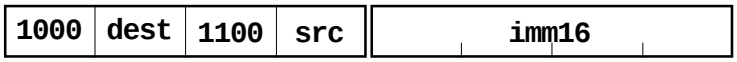
【機能】

Rsrcと32ビットにゼロ拡張された16ビット即値の対応するビットの論理積をとり、結果をRdestに格納します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
AND3  Rdest,Rsrc,#imm16
```

BC 分岐命令 Branch on C-bit BC

【ニーモニック】

① **BC pcdisp8**
② **BC pcdisp24**

【動作】

条件付き分岐
① if (C==1) PC = (PC & 0xfffffffc) + (((signed char) pcdisp8) << 2);
② if (C==1) PC = (PC & 0xfffffffc) + (sign_extend (pcdisp24) << 2);
where
#define sign_extend(x) (((signed) ((x)<< 8)) >>8)

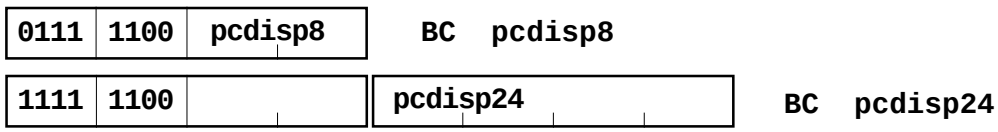
【機能】

条件ビット(C)が1のとき、指定されたラベルへ分岐します。命令フォーマットには2種類ありますが、アセンブラレベルで記述する場合はオペランドに分岐先のラベルを書けば、通常最短のフォーマットが選ばれます。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】



BCLR

ビット操作命令

Bit clear

[M32R-FPU拡張命令]

BCLR

【ニーモニック】

```
BCLR  #bitpos,@(disp16,Rsrc)
```

【動作】

メモリ内容のビット操作　指定ビットに0をセット

```
* ( signed char* ) ( Rsrc + ( signed short ) disp16 ) & = ~ ( 1<< ( 7-bitpos ) ) ;
```

【機能】

Rsrcと16ビットのディスプレースメントで指定された番地のメモリのバイトデータを読み込み、bitposで指定されたビットを0に変更した値をストアします。ディスプレースメントは、アドレス計算の前に符号拡張されます。bitposは0～7で、MSBが0、LSBが7となります。メモリアクセスはバイトサイズで行われます。BCLR命令実行中はLOCKビットがセットされ、BCLR命令が終了すれば、LOCKビットがクリアされます。LOCKビットはCPU内部にあり、ユーザがこのビットを直接リード/ライトすることはできません。

条件ビット(C)は変化しません。

LOCKビットとはCPU内部のビットでCPU以外のバス権を要求する回路に対してバス権の受付を制御するビットです。

CPU以外のバス権の要求は、各種マイコンにより異なりますので、各ユーザーズマニュアルを参照してください。

【発生EIT】

なし

【命令フォーマット】

1010	0	bitpos	0111	src	disp16

```
BCLR  #bitpos,@(disp16,Rsrc)
```

BEQ

分岐命令

Branch on equal to

BEQ

【ニーモニック】

```
BEQ  Rsrc1,Rsrc2,pcdisp16
```

【動作】

条件付き分岐

```
if ( Rsrc1 == Rsrc2 )  PC = ( PC & 0xfffffffc ) + ( ( ( signed short ) pcdisp16 ) << 2 );
```

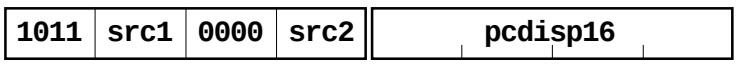
【機能】

Rsrc1とRsrc2が等しいとき、指定されたラベルへ分岐します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
BEQ  Rsrc1,Rsrc2,pcdisp16
```

BEQZ

分岐命令

Branch on equal to zero

BEQZ

【ニーモニック】

```
BEQZ  Rsrc,pcdisp16
```

【動作】

条件付き分岐

```
if ( Rsrc == 0 )  PC = ( PC & 0xfffffffc ) + ( ( ( signed short ) pcdisp16 ) << 2 );
```

【機能】

Rsrcが0のとき、指定されたラベルへ分岐します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】


```
BEQZ  Rsrc,pcdisp16
```

BGEZ

分岐命令

Branch on greater than or equal to zero

BGEZ

【ニーモニック】

```
BGEZ  Rsrc,pcdisp16
```

【動作】

条件付き分岐

if ((signed) Rsrc >= 0) PC = (PC & 0xfffffffc) + (((signed short) pcdisp16) << 2);

【機能】

Rsrcの内容を符号付き32ビット値としてみたとき、0又は0より大きい場合に指定されたラベルへ分岐します。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】


```
BGEZ  Rsrc,pcdisp16
```

BGTZ

分岐命令

Branch on greater than zero

BGTZ

【ニーモニック】

```
BGTZ  Rsrc,pcdisp16
```

【動作】

条件付き分岐

```
if ( ( signed ) Rsrc > 0 )  PC = ( PC & 0xfffffffc ) + ( ( ( signed short ) pcdisp16 ) << 2 );
```

【機能】

Rsrcの内容を符号付き32ビット値としてみたとき、0より大きい場合に指定されたラベルへ分岐します。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】


```
BGTZ  Rsrc,pcdisp16
```

BL 分岐命令 Branch and link BL

【ニーモニック】

① **BL pcdisp8**
② **BL pcdisp24**

【動作】

サブルーチン呼び出し(PC相対)

① R14 = (PC & 0xfffffffc) + 4;
PC = (PC & 0xfffffffc) + (((signed char) pcdisp8) << 2);

② R14 = (PC & 0xfffffffc) + 4;
PC = (PC & 0xfffffffc) + (sign_extend (pcdisp24) << 2);

where
#define sign_extend(x) (((signed) ((x)<< 8)) >>8)

【機能】

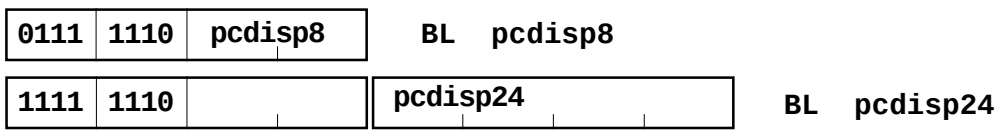
無条件分岐を行い、戻り先をR14に格納します。命令フォーマットには2種類ありますが、アセンブラレベルで記述する場合はオペランドに分岐先のラベルを書けば、通常最短のフォーマットが選ばれます。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】



BLEZ

分岐命令

Branch on less than or equal to zero

BLEZ

【ニーモニック】

```
BLEZ  Rsrc,pcdisp16
```

【動作】

条件付き分岐

```
if ( ( signed ) Rsrc <= 0 )  PC = ( PC & 0xfffffffc ) + ( ( ( signed short ) pcdisp16 ) << 2 );
```

【機能】

Rsrcの内容を符号付き32ビット値としてみたとき、0又は0より小さい場合に指定されたラベルへ分岐します。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】


```
BLEZ  Rsrc,pcdisp16
```

BLTZ

分岐命令

Branch on less than zero

BLTZ

【ニーモニック】

```
BLTZ  Rsrc,pcdisp16
```

【動作】

条件付き分岐

if ((signed) Rsrc < 0) PC = (PC & 0xfffffffc) + (((signed short) pcdisp16) << 2);

【機能】

Rsrcの内容を符号付き32ビット値としてみたとき、0より小さい場合に指定されたラベルへ分岐します。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

1011	0000	1010	src	pcdisp16

```
BLTZ  Rsrc,pcdisp16
```

BNC

分岐命令

Branch on not C-bit

BNC

【ニーモニック】

① **BNC pcdisp8**
② **BNC pcdisp24**

【動作】

条件付き分岐
① if (C==0) PC = (PC & 0xfffffffc) + (((signed char) pcdisp8) << 2);
② if (C==0) PC = (PC & 0xfffffffc) + (sign_extend (pcdisp24) << 2);
where
#define sign_extend(x) (((signed) ((x)<< 8)) >>8)

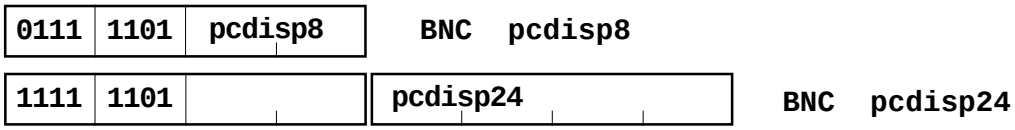
【機能】

条件ビット(C)が0のとき、指定されたラベルへ分岐します。命令フォーマットには2種類ありますが、アセンブラレベルで記述する場合はオペランドに分岐先のラベルを書けば、通常最短のフォーマットが選ばれます。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】



BNE

分岐命令

Branch on not equal to

BNE

【ニーモニック】

```
BNE  Rsrc1,Rsrc2,pcdisp16
```

【動作】

条件付き分岐

```
if ( Rsrc1 != Rsrc2 )  PC = ( PC & 0xfffffffc ) + ( ( ( signed short ) pcdisp16 ) << 2 );
```

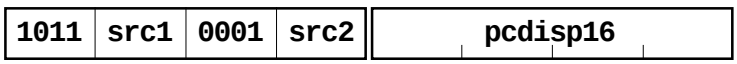
【機能】

Rsrc1とRsrc2が等しくないとき、指定されたラベルへ分岐します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
BNE  Rsrc1,Rsrc2,pcdisp16
```

BNEZ

分岐命令

Branch on not equal to zero

BNEZ

【ニーモニック】

```
BNEZ  Rsrc,pcdisp16
```

【動作】

条件付き分岐

if (Rsrc != 0) PC = (PC & 0xfffffffc) + (((signed short) pcdisp16) << 2);

【機能】

Rsrcが0でないとき、指定されたラベルへ分岐します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】


```
BNEZ Rsrc,pcdisp16
```

BRA 分岐命令 Branch BRA

【ニーモニック】

① **BRA pcdisp8**
② **BRA pcdisp24**

【動作】

無条件分岐
① PC = (PC & 0xfffffffc) + (((signed char) pcdisp8) << 2);
② PC = (PC & 0xfffffffc) + (sign_extend (pcdisp24) << 2);
where
#define sign_extend(x) (((signed) ((x) << 8)) >> 8)

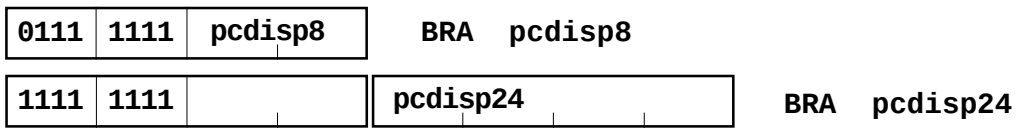
【機能】

指定されたラベルへ無条件分岐します。命令フォーマットには2種類ありますが、アセンブラレベルで記述する場合はオペランドに分岐先のラベルを書けば、通常最短のフォーマットが選ばれます。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】



BSET

ビット操作命令

Bit set

[M32R-FPU拡張命令]

BSET

【ニーモニック】

```
BSET  #bitpos,@(disp16,Rsrc)
```

【動作】

メモリ内容のビット操作　指定ビットに1をセット

```
* ( signed char* ) ( Rsrc + ( signed short ) disp16 ) |= ( 1<< ( 7-bitpos ) ) ;
```

【機能】

Rsrcと16ビットのディスプレースメントで指定された番地のメモリのバイトデータを読み込み、bitposで指定されたビットを1に変更した値をストアします。ディスプレースメントは、アドレス計算の前に符号拡張されます。bitposは0～7で、MSBが0、LSBが7となります。メモリアクセスはバイトサイズで行われます。BSET命令実行中はLOCKビットがセットされ、BSET命令が終了すれば、LOCKビットがクリアされます。LOCKビットはCPU内部にあり、ユーザがこのビットを直接リード/ライトすることはできません。

条件ビット(C)は変化しません。

LOCKビットとはCPU内部のビットでCPU以外のバス権を要求する回路に対してバス権の受付を制御するビットです。

CPU以外のバス権の要求は、各種マイコンにより異なりますので、各ユーザーズマニュアルを参照してください。

【発生EIT】

なし

【命令フォーマット】

1010	0	bitpos	0110	src	disp16

BSET #bitpos,@(disp16,Rsrc)

BTST

ビット操作命令

Bit test

[M32R-FPU拡張命令]

BTST

【ニーモニック】

```
BTST  #bitpos,Rsrc
```

【動作】

レジスタの指定ビットを取り出すビット操作

```
C = ( Rsrc >> ( 7-bitpos ) ) &1;
```

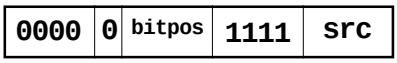
【機能】

Rsrcの下位8ビット中で、bitposで指定されたビットを取り出し、条件ビット(C)にセットします。bitposは0～7で、MSBが0、LSBが7となります。

【発生EIT】

なし

【命令フォーマット】




```
BTST  #bitpos,Rsrc
```

CLRPSW

ビット操作命令

Clear PSW

[M32R-FPU拡張命令]

CLRPSW

【ニーモニック】

```
CLRPSW  #imm8
```

【動作】

PSWのSM, IE, Cの任意のビットに0をセット

PSW& = ~imm8 | 0xffffff00

【機能】

8ビット即値のb0(MSB), b1, b7(LSB)の各ビットの反転値と、PSWのSM, IE, Cの各ビットとの論理積を、PSWのSM, IE, Cにそれぞれセットします。#imm8のb7(LSB)が1のとき条件ビット(C)は0になり、それ以外は変化しません。

【発生EIT】

なし

【命令フォーマット】

0111	0010	imm8

CLRPSW #imm8

CMP 比較命令 Compare CMP

【ニーモニック】

```
CMP  Rsrc1,Rsrc2
```

【動作】

比較

C = ((signed) Rsrc1 < (signed) Rsrc2) ? 1:0;

【機能】

Rsrc1とRsrc2を比較し、Rsrc1がRsrc2よりも小さいとき条件ビット(C)が1にセットされます。オペランドは符号付き32ビット値として扱われます。

【発生EIT】

なし

【命令フォーマット】

0000	src1	0100	src2

CMP Rsrc1,Rsrc2

CMPI 比較命令 Compare immediate CMPI

【ニーモニック】

```
CMPI  Rsrc,#imm16
```

【動作】

比較

C = ((signed) Rsrc < (signed) ((signed short) imm16)) ? 1:0;

【機能】

Rsrcと16ビット即値を比較し、Rsrcが16ビット即値よりも小さいとき条件ビット(C)が1にセットされます。オペランドは符号付き32ビット値として扱われます。16ビット即値は比較の前に32ビットデータに符号付き拡張されます。

【発生EIT】

なし

【命令フォーマット】


```
CMPI  Rsrc,#imm16
```

CMPU

比較命令
Compare unsigned

CMPU

【ニーモニック】

```
CMPU  Rsrc1,Rsrc2
```

【動作】

比較
C = ((unsigned) Rsrc1 < (unsigned) Rsrc2) ? 1:0;

【機能】

Rsrc1とRsrc2を比較し、Rsrc1がRsrc2よりも小さいとき条件ビット(C)が1にセットされます。
オペランドは符号なし32ビット値として扱われます。

【発生EIT】

なし

【命令フォーマット】

0000	src1	0101	src2

CMPU Rsrc1,Rsrc2

CMPUI 比較命令 Compare unsigned immediate CMPUI

【ニーモニック】

```
CMPUI  Rsrc,#imm16
```

【動作】

比較

C = ((unsigned) Rsrc < (unsigned) ((signed short) imm16)) ? 1:0;

【機能】

Rsrcと16ビット即値を比較し、Rsrcが16ビット即値よりも小さいとき条件ビット(C)が1にセットされます。オペランドは符号なし32ビット値として扱われます。16ビット即値は比較の前に32ビットデータに符号付き拡張されます。

【発生EIT】

なし

【命令フォーマット】


```
CMPUI Rsrc,#imm16
```

DIV

乗除算命令
Divide

DIV

【ニーモニック】

```
DIV  Rdest,Rsrc
```

【動作】

符号付き除算
Rdest = (signed) Rdest / (signed) Rsrc;

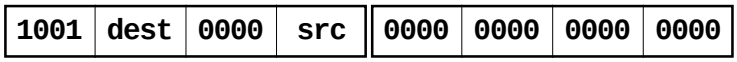
【機能】

RdestをRsrcで割り算し、商をRdestに格納します。オペランドは符号付き32ビット値として扱われ、商は0方向に丸められます。
条件ビット(C)は変化しません。
Rsrcが0のとき、結果は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
DIV  Rdest,Rsrc
```

DIVU

乗除算命令

Divide unsigned

DIVU

【ニーモニック】

```
DIVU  Rdest,Rsrc
```

【動作】

符号なし除算

Rdest = (unsigned) Rdest / (unsigned) Rsrc;

【機能】

RdestをRsrcで割り算し、商をRdestに格納します。オペランドは符号なし32ビット値として扱われ、商は0方向に丸められます。

条件ビット(C)は変化しません。

Rsrcが0のとき、結果は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	0001	src	0000	0000	0000	0000

```
DIVU  Rdest,Rsrc
```

FADD

浮動小数点命令

Floating-point add

[M32R-FPU拡張命令]

FADD

【ニーモニック】

```
FADD  Rdest,Rsrc1,Rsrc2
```

【動作】

浮動小数点加算

Rdest = Rsrc1 + Rsrc2 ;

【機能】

Rsrc1に格納された単精度浮動小数点数とRsrc2に格納された単精度浮動小数点数を加算し、結果をRdestに格納します。結果はFPSRのRMフィールドに従って丸められます。非正規化数の扱いはFPSRのDNビットによって変化します。条件ビット(C)は変化しません。

【発生EIT】

浮動小数点例外(FPE)

- 非実装例外(UIPL)
- 無効演算例外(IVLD)
- オーバフロー(OVF)
- アンダフロー(UDF)
- 精度異常例外(IXCT)

【命令フォーマット】

1101	src1	0000	src2

0000	dest	0000	0000

```
FADD  Rdest,Rsrc1,Rsrc2
```

【動作補足説明】

DN＝0とDN＝1のときについて、Rsrc1及びRsrc2の値と演算結果の対応を示します。

DN＝0のとき

		Rsrc2							
		正規化数	+0	-0	+∞	-∞	**非正規化数**	QNaN	SNaN
Rsrc1	**正規化数**	**加算**							
	+0		+0	**（注）**		-∞			
	-0		**（注）**	-0					
	+∞				+∞	IVLD			
	-∞		-∞		IVLD	-∞			
	非正規化数						UIPL		
	QNaN							QNaN	
	SNaN								IVLD

DN＝1のとき

		Rsrc2						
		正規化数	+0, +非正規化数	-0, -非正規化数	+∞	-∞	QNaN	SNaN
Rsrc1	**正規化数**	**加算**	**正規化数**					
	+0, +非正規化数	**正規化数**	+0	**（注）**		-∞		
	-0, -非正規化数		**（注）**	-0				
	+∞				+∞	IVLD		
	-∞		-∞		IVLD	-∞		
	QNaN						QNaN	
	SNaN							IVLD

IVLD：無効演算例外
UIPL：非実装例外
NaN：非数(Not a Number)
SNaN：Signaling NaN
QNaN：Quiet NaN

注．丸めモードが-∞方向への丸めのときは-0、それ以外の丸めモードのときは+0。

FCMP

浮動小数点命令

Floating-point compare

[M32R-FPU拡張命令]

FCMP

【ニーモニック】

```
FCMP   Rdest,Rsrc1,Rsrc2
```

【動作】

浮動小数点比較

Rdest = (Rsrc1とRsrc2の比較結果) ;

Rsrc1, Rsrc2の少なくとも一方がSNaNのとき、浮動小数点例外(無効演算例外)を発生します。

【機能】

Rsrc1に格納された単精度浮動小数点数とRsrc2に格納された単精度浮動小数点数を比較し、結果をRdestに格納します。比較結果は以下の方法で判定することができます。

Rdest		比較結果	比較結果を判定するために使用する命令例
b0＝0	b1～b31＝**すべてのビットが**0	Rsrc1＝Rsrc2	beqz　Rdest, LABEL
	b1～b9＝111 1111 11, b10～b31＝**任意の値**	**比較不能**	bgtz　Rdest, LABEL
	上記以外	Rsrc1＞Rsrc2	
b0＝1	b1～b31＝**任意の値**	Rsrc1＜Rsrc2	bltz　Rdest, LABEL

非正規化数の扱いはFPSRのDNビットによって変化します。条件ビット(C)は変化しません。

【発生EIT】

浮動小数点例外(FPE)

- 非実装例外(UIPL)
- 無効演算例外(IVLD)

【命令フォーマット】

1101	src1	0000	src2	0000	dest	1100	0000

```
FCMP   Rdest,Rsrc1,Rsrc2
```

【動作補足説明】

DN＝0とDN＝1のときについて、Rsrc1及びRsrc2の値と演算結果の対応を示します。

DN＝0のとき

		Rsrc2							
		正規化数	+0	-0	+∞	-∞	**非正規化数**	QNaN	SNaN
Rsrc1	**正規化数**	**比較**			-∞	+∞			
	+0		00000000						
	-0								
	+∞	+∞			00000000				
	-∞	-∞				00000000			
	非正規化数						UIPL		
	QNaN							**比較不能**	
	SNaN								IVLD

DN＝1のとき

		Rsrc2						
		正規化数	+0, +**非正規化数**	-0, -**非正規化数**	+∞	-∞	QNaN	SNaN
Rsrc1	**正規化数**	**比較**			-∞	+∞		
	+0, +**非正規化数**		00000000					
	-0, -**非正規化数**							
	+∞	+∞			00000000			
	-∞	-∞				00000000		
	QNaN						**比較不能**	
	SNaN							IVLD

IVLD：無効演算例外
UIPL：非実装例外
NaN：非数(Not a Number)
SNaN：Signaling NaN
QNaN：Quiet NaN

FCMPE 浮動小数点命令 Floating-point compare with exception if unordered FCMPE

[M32R-FPU拡張命令]

【ニーモニック】

```
FCMPE   Rdest,Rsrc1,Rsrc2
```

【動作】

浮動小数点比較

Rdest = (Rsrc1とRsrc2の比較結果) ;

Rsrc1、Rsrc2の少なくとも一方がQNaN又はSNaNのとき、浮動小数点例外(無効演算例外)を発生します。

【機能】

Rsrc1に格納された単精度浮動小数点数とRsrc2に格納された単精度浮動小数点数を比較し、結果をRdestに格納します。比較結果は以下の方法で判定することができます。

Rdest		比較結果	比較結果を判定するために使用する命令例
b0＝0	b1～b31＝**すべてのビットが**0	Rsrc1＝Rsrc2	beqz　Rdest, LABEL
	b1～b9＝111 1111 11, b10～b31＝**任意の値** **（注）**	**比較不能**	bgtz　Rdest, LABEL
	上記以外	Rsrc1＞Rsrc2	
b0＝1	b1～b31＝**任意の値**	Rsrc1＜Rsrc2	bltz　Rdest, LABEL

注．EVビット(FPSRレジスタのb21)＝"0"のときのみ

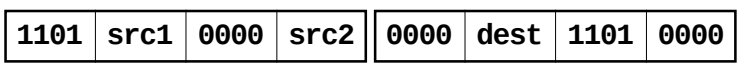
非正規化数の扱いはFPSRのDNビットによって変化します。条件ビット(C)は変化しません。

【発生EIT】

浮動小数点例外(FPE)

- 非実装例外(UIPL)
- 無効演算例外(IVLD)

【命令フォーマット】




```
FCMPE   Rdest,Rsrc1,Rsrc2
```

【動作補足説明】

DN＝0とDN＝1のときについて、Rsrc1及びRsrc2の値と演算結果の対応を示します。

DN＝0のとき

		Rsrc2							
		正規化数	+0	-0	+∞	-∞	**非正規化数**	QNaN	SNaN
Rsrc1	**正規化数**	**比較**			-∞	+∞	UIPL	IVLD	
	+0		00000000						
	-0								
	+∞	+∞			00000000				
	-∞	-∞				00000000			
	非正規化数								
	QNaN								
	SNaN								

DN＝1のとき

		Rsrc2						
		正規化数	**+0, +非正規化数**	**-0, -非正規化数**	+∞	-∞	QNaN	SNaN
Rsrc1	**正規化数**	**比較**			-∞	+∞	IVLD	
	+0, +非正規化数		00000000					
	-0, -非正規化数							
	+∞	+∞			00000000			
	-∞	-∞				00000000		
	QNaN							
	SNaN							

IVLD：無効演算例外
UIPL：非実装例外
NaN：非数(Not a Number)
SNaN：Signaling NaN
QNaN：Quiet NaN

FDIV

浮動小数点命令

Floating-point divide

[M32R-FPU拡張命令]

FDIV

【ニーモニック】

```
FDIV  Rdest,Rsrc1,Rsrc2
```

【動作】

浮動小数点除算

Rdest = Rsrc1 / Rsrc2 ;

【機能】

Rsrc1に格納された単精度浮動小数点数をRsrc2に格納された単精度浮動小数点数で除算し、結果をRdestに格納します。結果はFPSRのRMフィールドに従って丸められます。非正規化数の扱いはFPSRのDNビットによって変化します。条件ビット(C)は変化しません。

【発生EIT】

浮動小数点例外(FPE)

- 非実装例外(UIPL)
- 無効演算例外(IVLD)
- オーバフロー(OVF)
- アンダフロー(UDF)
- 精度異常例外(IXCT)
- ゼロ除算例外(DIV0)

【命令フォーマット】

1101	src1	0000	src2	0010	dest	0000	0000

```
FDIV  Rdest,Rsrc1,Rsrc2
```

【動作補足説明】

DN＝0とDN＝1のときについて、Rsrc1及びRsrc2の値と演算結果の対応を示します。

DN＝0のとき

		Rsrc2							
		正規化数	+0	-0	+∞	-∞	**非正規化数**	QNaN	SNaN
Rsrc1	**正規化数**	**除算**	DIV0		0				
	+0	0	IVLD		+0	-0			
	-0				-0	+0			
	+∞	∞	+∞	-∞	IVLD				
	-∞		-∞	+∞					
	非正規化数	UIPL							
	QNaN	QNaN							
	SNaN	IVLD							

DN＝1のとき

		Rsrc2						
		正規化数	+0, +**非正規化数**	-0, -**非正規化数**	+∞	-∞	QNaN	SNaN
Rsrc1	**正規化数**	**除算**	DIV0		0			
	+0, +**非正規化数**	0	IVLD		+0	-0		
	-0, -**非正規化数**				-0	+0		
	+∞	∞	+∞	-∞	IVLD			
	-∞		-∞	+∞				
	QNaN	QNaN						
	SNaN	IVLD						

IVLD：無効演算例外
UIPL：非実装例外
DIV0：ゼロ除算例外
NaN：非数(Not a Number)
SNaN：Signaling NaN
QNaN：Quiet NaN

FMADD

浮動小数点命令

Floating-point multiply and add

[M32R-FPU拡張命令]

FMADD

【ニーモニック】

```
FMADD  Rdest,Rsrc1,Rsrc2
```

【動作】

浮動小数点積和演算

Rdest = Rdest + Rsrc1 * Rsrc2 ;

【機能】

本命令は次の2ステップで実行されます。

● ステップ1

Rsrc1に格納された単精度浮動小数点数とRsrc2に格納された単精度浮動小数点数を乗算します。乗算結果はFPSRのRMフィールドの設定に関係なく、0方向に丸められます。

● ステップ2

ステップ1の結果(丸め後の値)とRdestに格納された単精度浮動小数点数を加算します。加算結果はFPSRのRMフィールドに従って丸められます。

演算結果はRdestに格納されます。例外の判定はステップ1、ステップ2の両方で行います。非正規化数の扱いはFPSRのDNビットによって変化します。条件ビット(C)は変化しません。

【発生EIT】

浮動小数点例外(FPE)

- 非実装例外(UIPL)
- 無効演算例外(IVLD)
- オーバフロー(OVF)
- アンダフロー(UDF)
- 精度異常例外(IXCT)

【命令フォーマット】

1101	src1	0000	src2	0011	dest	0000	0000

FMADD Rdest,Rsrc1,Rsrc2

【動作補足説明】

DN＝0とDN＝1のときについて、Rsrc1、Rsrc2及びRdestの値と演算結果の対応を示します。

DN＝0のとき

乗算終了時点の値

		Rsrc2							
		正規化数	+0	-0	+∞	-∞	**非正規化数**	QNaN	SNaN
Rsrc1	**正規化数**	**乗算**			∞				
	+0		+0	-0	IVLD				
	-0		-0	+0					
	+∞	∞	IVLD		+∞	-∞			
	-∞				-∞	+∞			
	非正規化数	UIPL							
	QNaN	QNaN							
	SNaN	IVLD							

加算後の値

		乗算終了時点の値					
		正規化数	+0	-0	+∞	-∞	QNaN
Rdest	**正規化数**	**加算**					
	+0		+0	**（注）**		-∞	
	-0		**（注）**	-0			
	+∞	+∞				IVLD	
	-∞	-∞			IVLD	-∞	
	非正規化数	UIPL					
	QNaN	QNaN					
	SNaN	IVLD					

IVLD：無効演算例外
UIPL：非実装例外
NaN：非数（Not a Number）
SNaN：Signaling NaN
QNaN：Quiet NaN

注．丸めモードが-∞方向への丸めのときは-0、それ以外の丸めモードのときは+0。

DN＝1のとき

乗算終了時点の値

		Rsrc2						
		正規化数	+0, +非正規化数	-0, -非正規化数	+∞	-∞	QNaN	SNaN
Rsrc1	正規化数	乗算			∞			
	+0, +非正規化数		+0	-0	IVLD			
	-0, -非正規化数		-0	+0				
	+∞	∞	IVLD		+∞	-∞		
	-∞				-∞	+∞		
	QNaN	QNaN						
	SNaN	IVLD						

加算後の値

		乗算終了時点の値					
		正規化数	+0	-0	+∞	-∞	QNaN
Rdest	正規化数	加算					
	+0		+0	（注）		-∞	
	-0		（注）	-0			
	+∞	+∞				IVLD	
	-∞	-∞			IVLD	-∞	
	QNaN	QNaN					
	SNaN	IVLD					

IVLD：無効演算例外
UIPL：非実装例外
NaN：非数（Not a Number）
SNaN：Signaling NaN
QNaN：Quiet NaN

注．丸めモードが-∞方向への丸めのときは-0、それ以外の丸めモードのときは+0。

FMSUB 浮動小数点命令 Floating-point multiply and subtract FMSUB

[M32R-FPU拡張命令]

【ニーモニック】

```
FMSUB  Rdest,Rsrc1,Rsrc2
```

【動作】

浮動小数点積差演算

Rdest = Rdest - Rsrc1 * Rsrc2 ;

【機能】

本命令は次の2ステップで実行されます。

● ステップ1

Rsrc1に格納された単精度浮動小数点数とRsrc2に格納された単精度浮動小数点数を乗算します。乗算結果はFPSRのRMフィールドの設定に関係なく、0方向に丸められます。

● ステップ2

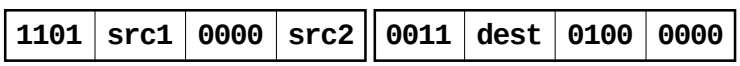
Rdestに格納された単精度浮動小数点数からステップ1の結果(丸め後の値)を減算します。減算結果はFPSRのRMフィールドに従って丸められます。

演算結果はRdestに格納されます。例外の判定はステップ1、ステップ2の両方で行います。非正規化数の扱いはFPSRのDNビットによって変化します。条件ビット(C)は変化しません。

【発生EIT】

浮動小数点例外(FPE)
- ・非実装例外(UIPL)
- ・無効演算例外(IVLD)
- ・オーバフロー(OVF)
- ・アンダフロー(UDF)
- ・精度異常例外(IXCT)

【命令フォーマット】




```
FMSUB  Rdest,Rsrc1,Rsrc2
```

【動作補足説明】

DN＝0とDN＝1のときについて、Rsrc1、Rsrc2及びRdestの値と演算結果の対応を示します。

DN＝0のとき

乗算終了時点の値

		Rsrc2							
		正規化数	+0	-0	+∞	-∞	**非正規化数**	QNaN	SNaN
Rsrc1	**正規化数**	**乗算**			∞				
	+0		+0	-0	IVLD				
	-0		-0	+0					
	+∞	∞	IVLD		+∞	-∞			
	-∞				-∞	+∞			
	非正規化数	UIPL							
	QNaN	QNaN							
	SNaN	IVLD							

減算後の値

		乗算終了時点の値					
		正規化数	+0	-0	+∞	-∞	QNaN
Rdest	**正規化数**	**減算**			-∞	+∞	
	+0		**（注）**	+0			
	-0		-0	**（注）**			
	+∞	+∞			IVLD		
	-∞	-∞				IVLD	
	非正規化数	UIPL					
	QNaN	QNaN					
	SNaN	IVLD					

IVLD：無効演算例外
UIPL：非実装例外
NaN：非数（Not a Number）
SNaN：Signaling NaN
QNaN：Quiet NaN

注．丸めモードが-∞方向への丸めのときは-0、それ以外の丸めモードのときは+0。

DN＝1のとき

乗算終了時点の値

Rsrc1 \ Rsrc2		正規化数	+0, +非正規化数	-0, -非正規化数	+∞	-∞	QNaN	SNaN
	正規化数	乗算			∞			
	+0, +非正規化数		+0	-0	IVLD			
	-0, -非正規化数		-0	+0				
Rsrc1	+∞	∞	IVLD		+∞	-∞		
	-∞				-∞	+∞		
	QNaN	QNaN						
	SNaN	IVLD						

減算後の値

Rdest \ 乗算終了時点の値		正規化数	+0	-0	+∞	-∞	QNaN
	正規化数	減算			-∞	+∞	
	+0		（注）	+0			
	-0		-0	（注）			
Rdest	+∞	+∞			IVLD		
	-∞	-∞				IVLD	
	QNaN	QNaN					
	SNaN	IVLD					

IVLD：無効演算例外
UIPL：非実装例外
NaN：非数（Not a Number）
SNaN：Signaling NaN
QNaN：Quiet NaN

注．丸めモードが-∞方向への丸めのときは-0、それ以外の丸めモードのときは+0。

FMUL

浮動小数点命令

Floating-point multiply

[M32R-FPU拡張命令]

FMUL

【ニーモニック】

```
FMUL  Rdest,Rsrc1,Rsrc2
```

【動作】

浮動小数点乗算

Rdest = Rsrc1 * Rsrc2 ;

【機能】

Rsrc1に格納された単精度浮動小数点数とRsrc2に格納された単精度浮動小数点数を乗算し、結果をRdestに格納します。結果はFPSRのRMフィールドに従って丸められます。非正規化数の扱いはFPSRのDNビットによって変化します。条件ビット(C)は変化しません。

【発生EIT】

浮動小数点例外(FPE)

- 非実装例外(UIPL)
- 無効演算例外(IVLD)
- オーバフロー(OVF)
- アンダフロー(UDF)
- 精度異常例外(IXCT)

【命令フォーマット】

1101	src1	0000	src2

0001	dest	0000	0000

```
FMUL  Rdest,Rsrc1,Rsrc2
```

【動作補足説明】

DN＝0とDN＝1のときについて、Rsrc1及びRsrc2の値と演算結果の対応を示します。

DN＝0のとき

		Rsrc2							
		正規化数	+0	-0	+∞	-∞	**非正規化数**	QNaN	SNaN
Rsrc1	**正規化数**	**乗算**			∞				
	+0		+0	-0	IVLD				
	-0		-0	+0					
	+∞	∞	IVLD		+∞	-∞			
	-∞				-∞	+∞			
	非正規化数	UIPL							
	QNaN	QNaN							
	SNaN	IVLD							

DN＝1のとき

		Rsrc2						
		正規化数	+0, +非正規化数	-0, -非正規化数	+∞	-∞	QNaN	SNaN
Rsrc1	**正規化数**	**乗算**			∞			
	+0, +非正規化数		+0	-0	IVLD			
	-0, -非正規化数		-0	+0				
	+∞	∞	IVLD		+∞	-∞		
	-∞				-∞	+∞		
	QNaN	QNaN						
	SNaN	IVLD						

IVLD：無効演算例外
UIPL：非実装例外
NaN：非数(Not a Number)
SNaN：Signaling NaN
QNaN：Quiet NaN

FSUB

浮動小数点命令

Floating-point subtract

[M32R-FPU拡張命令]

FSUB

【ニーモニック】

```
FSUB  Rdest,Rsrc1,Rsrc2
```

【動作】

浮動小数点減算

Rdest = Rsrc1 - Rsrc2 ;

【機能】

Rsrc1に格納された単精度浮動小数点数からRsrc2に格納された単精度浮動小数点数を減算し、結果をRdestに格納します。結果はFPSRのRMフィールドに従って丸められます。非正規化数の扱いはFPSRのDNビットによって変化します。条件ビット(C)は変化しません。

【発生EIT】

浮動小数点例外(FPE)

- 非実装例外(UIPL)
- 無効演算例外(IVLD)
- オーバフロー(OVF)
- アンダフロー(UDF)
- 精度異常例外(IXCT)

【命令フォーマット】

1101	src1	0000	src2

0000	dest	0100	0000

```
FSUB  Rdest,Rsrc1,Rsrc2
```

【動作補足説明】

DN＝0とDN＝1のときについて、Rsrc1及びRsrc2の値と演算結果の対応を示します。

DN＝0のとき

		Rsrc2							
		正規化数	+0	-0	+∞	-∞	**非正規化数**	QNaN	SNaN
Rsrc1	**正規化数**	**減算**			-∞	+∞	UIPL	QNaN	IVLD
	+0		**（注）**	+0					
	-0		-0	**（注）**					
	+∞	+∞			IVLD				
	-∞	-∞				IVLD			
	非正規化数								
	QNaN							QNaN	
	SNaN								IVLD

DN＝1のとき

		Rsrc2						
		正規化数	**+0, +非正規化数**	**-0, -非正規化数**	+∞	-∞	QNaN	SNaN
Rsrc1	**正規化数**	**減算**			-∞	+∞	QNaN	IVLD
	+0, +非正規化数		**（注）**	+0				
	-0, -非正規化数		-0	**（注）**				
	+∞	+∞			IVLD			
	-∞	-∞				IVLD		
	QNaN						QNaN	
	SNaN							IVLD

IVLD：無効演算例外
UIPL：非実装例外
NaN：非数（Not a Number）
SNaN：Signaling NaN
QNaN：Quiet NaN

注．丸めモードが-∞方向への丸めのときは-0、それ以外の丸めモードのときは+0。

FTOI

浮動小数点命令

Float to Integer

[M32R-FPU拡張命令]

FTOI

【ニーモニック】

```
FTOI  Rdest,Rsrc
```

【動作】

単精度浮動小数点数から32ビット整数への変換

Rdest = (signed int) Rsrc ;

【機能】

Rsrcに格納された単精度浮動小数点数を32ビット整数に変換し、結果をRdestに格納します。結果はFPSRのRMに関係なく、常に0方向に丸められます。条件ビット(C)は変化しません。

【発生EIT】

浮動小数点例外(FPE)

- 非実装例外(UIPL)
- 無効演算例外(IVLD)
- 精度異常例外(IXCT)

【命令フォーマット】

1101	src	0000	0000	0100	dest	1000	0000

```
FTOI  Rdest,Rsrc
```

【動作補足説明】

DN＝0とDN＝1のときについて、Rsrcの値によるFTOI命令の実行結果を以下に示します。

DN＝0のとき

Rsrcの値（指数部はゲタなしの値）		Rdest	例外
Rsrc≧0	+∞	EIT発生時：変化なし 上記以外：H'7FFF FFFF	無効演算例外
	127≧指数部≧31		
	30≧指数部≧-126	H'0000 0000～H'7FFF FF80	なし（注1）
	+非正規化数	変化なし	非実装例外
	+0	H'0000 0000	なし
Rsrc＜0	-0		
	-非正規化数	変化なし	非実装例外
	30≧指数部≧-126	H'0000 0000～H'8000 0080	なし（注1）
	127≧指数部≧31	EIT発生時：変化なし 上記以外：H'8000 0000	無効演算例外（注2）
	-∞		
NaN	QNaN	EIT発生時：変化なし 上記以外： 符号ビット＝0:H'7FFF FFFF 符号ビット＝1:H'8000 0000	無効演算例外
	SNaN		

注1．丸め発生時は精度異常例外が発生します。
2．Rsrc＝H'CF00 0000のときは例外は発生しません。

DN＝1のとき

Rsrcの値（指数部はゲタなしの値）		Rdest	例外
Rsrc≧0	+∞	EIT発生時：変化なし 上記以外：H'7FFF FFFF	無効演算例外
	127≧指数部≧31		
	30≧指数部≧-126	H'0000 0000～H'7FFF FF80	なし（注1）
	+0, +非正規化数	H'0000 0000	なし
Rsrc＜0	-0, -非正規化数		
	30≧指数部≧-126	H'0000 0000～H'8000 0080	なし（注1）
	127≧指数部≧31	EIT発生時：変化なし 上記以外：H'8000 0000	無効演算例外（注2）
	-∞		
NaN	QNaN	EIT発生時：変化なし 上記以外： 符号ビット＝0:H'7FFF FFFF 符号ビット＝1:H'8000 0000	無効演算例外
	SNaN		

注1．丸め発生時は精度異常例外が発生します。
2．Rsrc＝H'CF00 0000のときは例外は発生しません。

FTOS

浮動小数点命令

Float to short

[M32R-FPU拡張命令]

FTOS

【ニーモニック】

```
FTOS  Rdest,Rsrc
```

【動作】

単精度浮動小数点数から16ビット整数への変換

Rdest = (signed short) Rsrc ;

【機能】

Rsrcに格納された単精度浮動小数点数を16ビット整数に変換し、結果をRdestに格納します。結果はFPSRのRMに関係なく、常に0方向に丸められます。条件ビット(C)は変化しません。

【発生EIT】

浮動小数点例外(FPE)

- 非実装例外(UIPL)
- 無効演算例外(IVLD)
- 精度異常例外(IXCT)

【命令フォーマット】

1101	src	0000	0000	0100	dest	1100	0000

```
FTOS  Rdest,Rsrc
```

【動作補足説明】

DN＝0とDN＝1のときについて、Rsrcの値によるFTOS命令の実行結果を以下に示します。

DN＝0のとき

Rsrcの値（指数部はゲタなしの値）		Rdest	例外
Rsrc≧0	+∞	EIT発生時：変化なし 上記以外：H'0000 7FFF	無効演算例外
	127≧指数部≧15		
	14≧指数部≧-126	H'0000 0000～H'0000 7FFF	なし（注1）
	+非正規化数	変化なし	非実装例外
	+0	H'0000 0000	なし
Rsrc＜0	-0		
	-非正規化数	変化なし	非実装例外
	14≧指数部≧-126	H'0000 0000～H'FFFF 8001	なし（注1）
	127≧指数部≧15	EIT発生時：変化なし 上記以外：H'FFFF 8000	無効演算例外（注2）
	-∞		
NaN	QNaN	EIT発生時：変化なし 上記以外： 符号ビット＝0:H'0000 7FFF 符号ビット＝1:H'FFFF 8000	無効演算例外
	SNaN		

注1．丸め発生時は精度異常例外が発生します。
2．Rsrc＝H'C700 0000のときは例外は発生しません。

DN＝1のとき

Rsrcの値（指数部はゲタなしの値）		Rdest	例外
Rsrc≧0	+∞	EIT発生時：変化なし 上記以外：H'0000 7FFF	無効演算例外
	127≧指数部≧15		
	14≧指数部≧-126	H'0000 0000～H'0000 7FFF	なし（注1）
	+0, +非正規化数	H'0000 0000	なし
Rsrc＜0	-0, -非正規化数		
	14≧指数部≧-126	H'0000 0000～H'FFFF 8001	なし（注1）
	127≧指数部≧15	EIT発生時：変化なし 上記以外：H'FFFF 8000	無効演算例外（注2）
	-∞		
NaN	QNaN	EIT発生時：変化なし 上記以外： 符号ビット＝0:H'0000 7FFF 符号ビット＝1:H'FFFF 8000	無効演算例外
	SNaN		

注1．丸め発生時は精度異常例外が発生します。
2．Rsrc＝H'C700 0000のときは例外は発生しません。Rsrc＝H'C700 0001～H'C700 00FFのときは精度異常例外が発生し、無効演算例外は発生しません。

ITOF

浮動小数点命令

Integer to float

[M32R-FPU拡張命令]

ITOF

【ニーモニック】

```
ITOF  Rdest,Rsrc
```

【動作】

整数から単精度浮動小数点数への変換

Rdes = (float) Rsrc ;

【機能】

Rsrcに格納された32ビット整数を単精度浮動小数点数に変換し、結果をRdestに格納します。結果はFPSRのRMフィールドに従って丸められます。条件ビット(C)は変化しません。H'0000 0000は丸めモードに関係なく、“ +0 ”として扱われます。

【発生EIT】

浮動小数点例外(FPE)

・精度異常例外(IXCT)

【命令フォーマット】

1101	src	0000	0000	0100	dest	0000	0000

```
ITOF  Rdest,Rsrc
```

JL

分岐命令

Jump and link

JL

【ニーモニック】

```
JL  Rsrc
```

【動作】

サブルーチン呼び出し(レジスタ直接)

```
R14 = ( PC & 0xfffffffc ) + 4;
PC = Rsrc  & 0xfffffffc;
```

【機能】

Rsrcで指定された番地へ無条件分岐を行い、戻り先をR14に格納します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	1110	1100	src

JL Rsrc

JMP

分岐命令
Jump

JMP

【ニーモニック】

```
JMP  Rsrc
```

【動作】

無条件分岐
PC = Rsrc & 0xfffffffc;

【機能】

Rsrcで指定された番地へ無条件分岐を行います。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	1111	1100	src

JMP Rsrc

LD ロード／ストア命令 Load LD

【ニーモニック】

① **LD Rdest,@Rsrc**
② **LD Rdest,@Rsrc+**
③ **LD Rdest,@(disp16,Rsrc)**

【動作】

メモリからレジスタへのロード
① Rdest = *(signed int *) Rsrc;
② Rdest = *(signed int *) Rsrc, Rsrc + = 4;
③ Rdest = *(signed int *) (Rsrc + (signed short) disp16);

【機能】

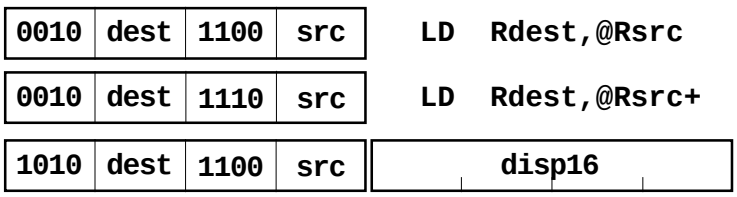
① Rsrcで指定された番地のメモリ内容をRdestに格納します。
② Rsrcで指定された番地のメモリ内容をRdestに格納し、その後でRsrcを4インクリメントします。
③ Rsrcと16ビットのディスプレースメントで指定された番地のメモリ内容を、Rdestに格納します。16ビットのディスプレースメントは、アドレス計算の前に32ビットに符号拡張されます。

条件ビット(C)は変化しません。

【発生EIT】

アドレス例外(AE)

【命令フォーマット】



LD Rdest,@(disp16,Rsrc)

LD24 転送命令 Load 24-bit immediate LD24

【ニーモニック】

```
LD24  Rdest,#imm24
```

【動作】

24ビット即値データの転送
Rdest = imm24 & 0x00ffffff;

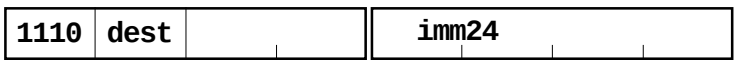
【機能】

24ビットの即値をRdestに格納します。24ビットの即値は32ビットにゼロ拡張されます。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
LD24  Rdest,#imm24
```

LDB

ロード／ストア命令
Load byte

LDB

【ニーモニック】

① **LDB Rdest,@Rsrc**
② **LDB Rdest,@(disp16,Rsrc)**

【動作】

メモリからレジスタへのロード
① Rdest = *(signed char *) Rsrc;
② Rdest = *(signed char *) (Rsrc + (signed short) disp16);

【機能】

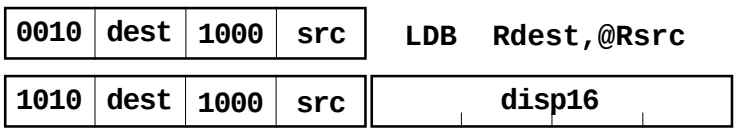
① Rsrcで指定された番地のメモリのバイトデータを、符号拡張してRdestに格納します。
② Rsrcと16ビットのディスプレースメントで指定された番地のメモリのバイトデータを、符号拡張してRdestに格納します。16ビットのディスプレースメントは、アドレス計算の前に32ビットに符号拡張されます。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】



LDB Rdest,@(disp16,Rsrc)

LDH ロード／ストア命令 Load halfword LDH

【ニーモニック】

① **LDH Rdest,@Rsrc**
② **LDH Rdest,@(disp16,Rsrc)**

【動作】

メモリからレジスタへのロード
① Rdest = *(signed short *) Rsrc;
② Rdest = *(signed short *) (Rsrc + (signed short) disp16);

【機能】

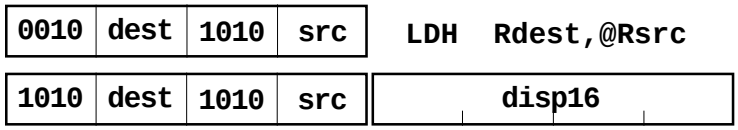
① Rsrcで指定された番地のメモリのハーフワードデータを、符号拡張してRdestに格納します。
② Rsrcと16ビットのディスプレースメントで指定された番地のメモリのハーフワードデータを、符号拡張してRdestに格納します。16ビットのディスプレースメントは、アドレス計算の前に32ビットに符号拡張されます。

条件ビット(C)は変化しません。

【発生EIT】

アドレス例外(AE)

【命令フォーマット】



LDH Rdest,@(disp16,Rsrc)

LDI

転送命令

Load immediate

LDI

【ニーモニック】

① **LDI Rdest,#imm8**
② **LDI Rdest,#imm16**

【動作】

即値データの転送
① Rdest = (signed char) imm8;
② Rdest = (signed short) imm16;

【機能】

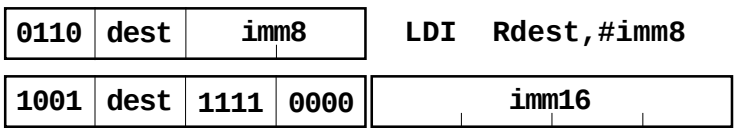
① 8ビットの即値をRdestに格納します。8ビットの即値は32ビットに符号拡張されます。
② 16ビットの即値をRdestに格納します。16ビットの即値は32ビットに符号拡張されます。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】



LDI Rdest,#imm16

LDUB

ロード／ストア命令
Load unsigned byte

LDUB

【ニーモニック】

① **LDUB Rdest,@Rsrc**
② **LDUB Rdest,@(disp16,Rsrc)**

【動作】

メモリからレジスタへのロード
① Rdest = *(unsigned char *) Rsrc;
② Rdest = *(unsigned char *) (Rsrc + (signed short) disp16);

【機能】

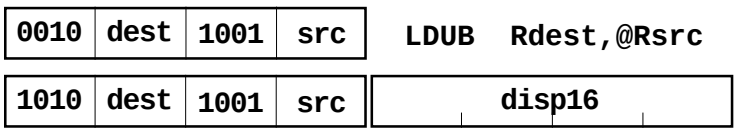
① Rsrcで指定された番地のメモリのバイトデータを、ゼロ拡張してRdestに格納します。
② Rsrcと16ビットのディスプレースメントで指定された番地のメモリのバイトデータを、ゼロ拡張してRdestに格納します。16ビットのディスプレースメントは、アドレス計算の前に32ビットに符号拡張されます。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】



LDUB Rdest,@(disp16,Rsrc)

LDUH

ロード／ストア命令

Load unsigned halfword

LDUH

【ニーモニック】

① **LDUH Rdest,@Rsrc**
② **LDUH Rdest,@(disp16,Rsrc)**

【動作】

メモリからレジスタへのロード
① Rdest = *(unsigned short *) Rsrc;
② Rdest = *(unsigned short *) (Rsrc + (signed short) disp16);

【機能】

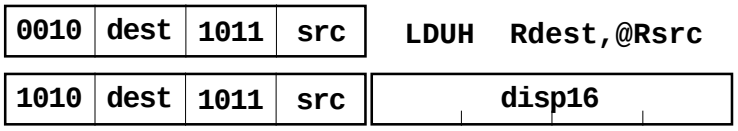
① Rsrcで指定された番地のメモリのハーフワードデータを、ゼロ拡張してRdestに格納します。
② Rsrcと16ビットのディスプレースメントで指定された番地のメモリのハーフワードデータを、ゼロ拡張してRdestに格納します。16ビットのディスプレースメントは、アドレス計算の前に32ビットに符号拡張されます。

条件ビット(C)は変化しません。

【発生EIT】

アドレス例外(AE)

【命令フォーマット】



LDUH Rdest,@(disp16,Rsrc)

LOCK　ロード／ストア命令 Load locked　LOCK

【ニーモニック】

```
LOCK  Rdest,@Rsrc
```

【動作】

ロック付きロード
LOCK = 1, Rdest = *(signed int *) Rsrc;

【機能】

Rsrcで指定された番地のメモリのワードデータを、Rdestに格納します。
条件ビット(C)は変化しません。
この命令は、通常のロードを行う以外にLOCKビットのセットも行います。
LOCKビットのクリアは、UNLOCK命令によって行われます。
LOCKビットはCPU内部にあり、LOCK命令とUNLOCK命令により操作できます。ユーザがこのビットを直接リード/ライトすることはできません。

LOCKビットとはCPU内部のビットでCPU以外のバス権を要求する回路に対してバス権の受付を制御するビットです。
CPU以外のバス権の要求は、各種マイコンにより異なりますので。各ユーザーズマニュアルを参照してください。

【発生EIT】

アドレス例外(AE)

【命令フォーマット】

0010	dest	1101	src

LOCK Rdest,@Rsrc

MACHI *DSP機能用命令* Multiply-accumulate high-order halfwords MACHI

【ニーモニック】

```
MACHI  Rsrc1,Rsrc2
```

【動作】

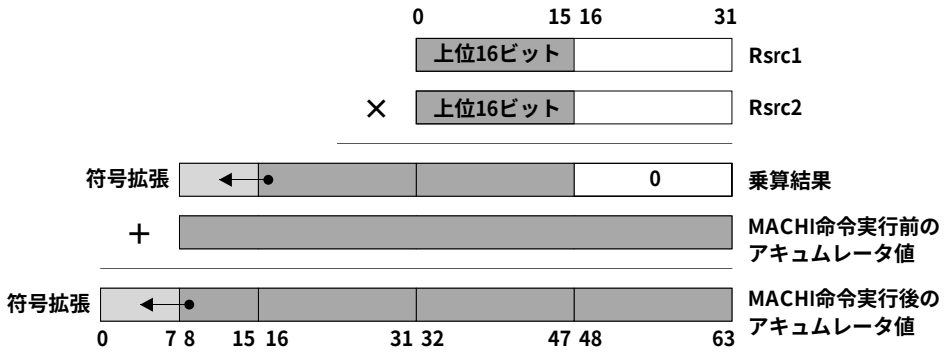
積和演算

accumulator + = ((signed) (Rsrc1 & 0xffff0000) * (signed short) (Rsrc2 >> 16));

【機能】

Rsrc1の上位16ビットと、Rsrc2の上位16ビットの乗算を行い、乗算結果とアキュムレータの下位56ビットとの加算を行います。ただし、乗算結果の最下位ビットはアキュムレータのビット47にあわせ、アキュムレータのビット8～15に対応する部分は符号拡張して加算します。加算結果はアキュムレータに格納されます。Rsrc1の上位16ビットと、Rsrc2の上位16ビットは符号付き整数として扱われます。

条件ビット(C)は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0100	src2

MACHI Rsrc1,Rsrc2

MACLO *DSP機能用命令* Multiply-accumulate low-order halfwords MACLO

【ニーモニック】

```
MACLO  Rsrc1,Rsrc2
```

【動作】

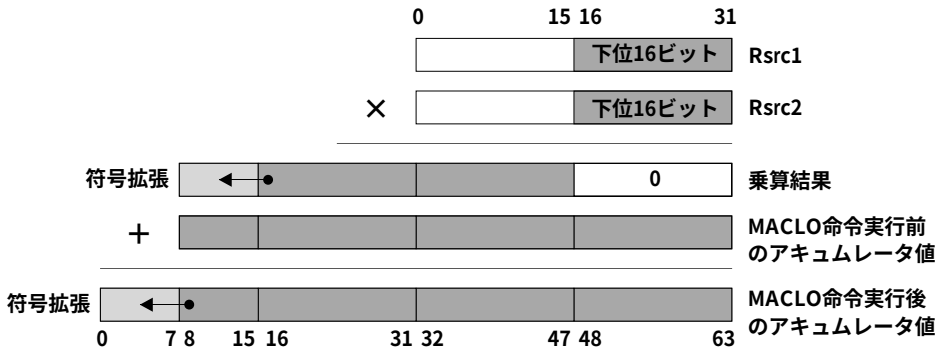
積和演算

accumulator + = ((signed) (Rsrc1 << 16) * (signed short) Rsrc2) ;

【機能】

Rsrc1の下位16ビットと、Rsrc2の下位16ビットの乗算を行い、乗算結果とアキュムレータの下位56ビットとの加算を行います。ただし、乗算結果の最下位ビットはアキュムレータのビット47にあわせ、アキュムレータのビット8～15に対応する部分は符号拡張して加算します。加算結果はアキュムレータに格納されます。Rsrc1の下位16ビットと、Rsrc2の下位16ビットは符号付き整数として扱われます。

条件ビット(C)は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0101	src2

MACLO Rsrc1,Rsrc2

MACWHI *DSP機能用命令* MACWHI

Multiply-accumulate word and high-order halfword

【ニーモニック】

```
MACWHI  Rsrc1,Rsrc2
```

【動作】

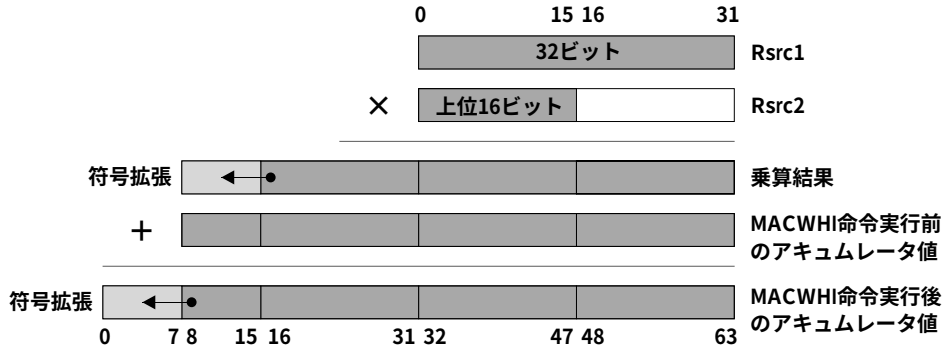
積和演算

accumulator + = ((signed) Rsrc1 * (signed short) (Rsrc2 >> 16));

【機能】

Rsrc1(32ビット)と、Rsrc2の上位16ビットの乗算を行い、乗算結果とアキュムレータの下位56ビットとの加算を行います。ただし、乗算結果の最下位ビットはアキュムレータの最下位ビットにあわせ、アキュムレータのビット8～15に対応する部分は符号拡張して加算します。加算結果はアキュムレータに格納されます。Rsrc1(32ビット)と、Rsrc2の上位16ビットは符号付き整数として扱われます。

条件ビット(C)は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0110	src2

MACWHI Rsrc1,Rsrc2

MACWLO

DSP機能用命令

Multiply-accumulate word and low-order halfword

MACWLO

【ニーモニック】

```
MACWLO  Rsrc1,Rsrc2
```

【動作】

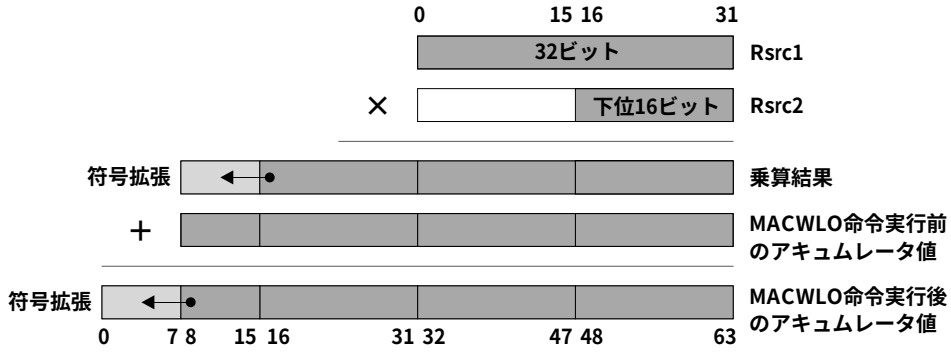
積和演算

accumulator + = ((signed) Rsrc1 * (signed short) Rsrc2) ;

【機能】

Rsrc1(32ビット)と、Rsrc2の下位16ビットの乗算を行い、乗算結果とアキュムレータの下位56ビットとの加算を行います。ただし、乗算結果の最下位ビットはアキュムレータの最下位ビットにあわせ、アキュムレータのビット8～15に対応する部分は符号拡張して加算します。加算結果はアキュムレータに格納されます。Rsrc1(32ビット)と、Rsrc2の下位16ビットは符号付き整数として扱われます。

条件ビット(C)は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0111	src2

MACWLO Rsrc1,Rsrc2

MUL 乗除算命令 Multiply MUL

【ニーモニック】

```
MUL  Rdest,Rsrc
```

【動作】

乗算

```
{ signed64bit tmp;
  tmp = ( signed64bit ) Rdest * ( signed64bit ) Rsrc;
  Rdest = ( signed int ) tmp; }
```

【機能】

RdestとRsrcの乗算を行い、結果をRdestに格納します。オペランドは符号付き整数として扱われます。

この命令の実行によりアキュムレータの内容は破壊されます。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	dest	0110	src

MUL Rdest,Rsrc

MULHI

DSP機能用命令

Multiply high-order halfwords

MULHI

【ニーモニック】

```
MULHI  Rsrc1,Rsrc2
```

【動作】

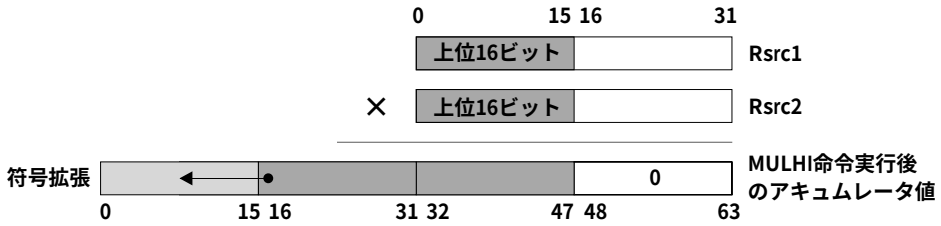
乗算

accumulator = ((signed) (Rsrc1 & 0xffff0000) * (signed short) (Rsrc2 >> 16));

【機能】

Rsrc1の上位16ビットと、Rsrc2の上位16ビットの乗算を行い、その結果をアキュムレータに格納します。ただし、乗算結果の最下位ビットはアキュムレータのビット47にあわせ、アキュムレータのビット0～15に対応する部分は符号拡張されます。また、アキュムレータのビット48～63は、ゼロクリアされます。Rsrc1の上位16ビットと、Rsrc2の上位16ビットは符号付き整数として扱われます。

条件ビット(C)は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0000	src2

MULHI Rsrc1,Rsrc2

MULLO

DSP機能用命令

Multiply low-order halfwords

MULLO

【ニーモニック】

```
MULLO  Rsrc1,Rsrc2
```

【動作】

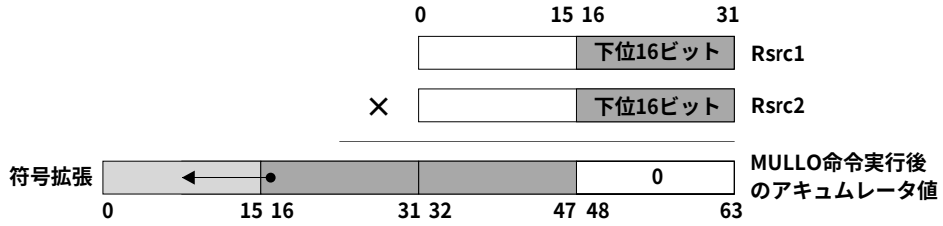
乗算

accumulator = ((signed) (Rsrc1 << 16) * (signed short) Rsrc2);

【機能】

Rsrc1の下位16ビットと、Rsrc2の下位16ビットの乗算を行い、その結果をアキュムレータに格納します。ただし、乗算結果の最下位ビットはアキュムレータのビット47にあわせ、アキュムレータのビット0～15に対応する部分は符号拡張されます。また、アキュムレータのビット48～63は、ゼロクリアされます。Rsrc1の下位16ビットと、Rsrc2の下位16ビットは符号付き整数として扱われます。

条件ビット(C)は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0001	src2

```
MULLO  Rsrc1,Rsrc2
```

MULWHI *DSP機能用命令* MULWHI

Multiply word and high-order halfword

【ニーモニック】

```
MULWHI  Rsrc1,Rsrc2
```

【動作】

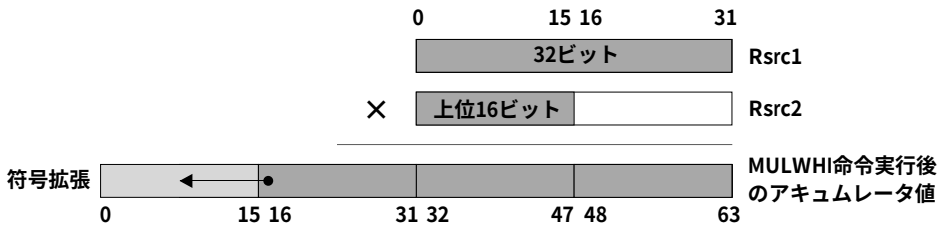
乗算

accumulator = ((signed) Rsrc1 * (signed short) (Rsrc2 >> 16));

【機能】

Rsrc1(32ビット)と、Rsrc2の上位16ビットの乗算を行い、結果をアキュムレータに格納します。ただし、乗算結果の最下位ビットはアキュムレータの最下位ビットにあわせ、アキュムレータのビット0～15に対応する部分は符号拡張されます。Rsrc1(32ビット)と、Rsrc2の上位16ビットは符号付き整数として扱われます。

条件ビット(C)は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0010	src2

MULWHI Rsrc1,Rsrc2

MULWLO *DSP機能用命令* MULWLO

Multiply word and low-order halfword

【ニーモニック】

```
MULWLO  Rsrc1,Rsrc2
```

【動作】

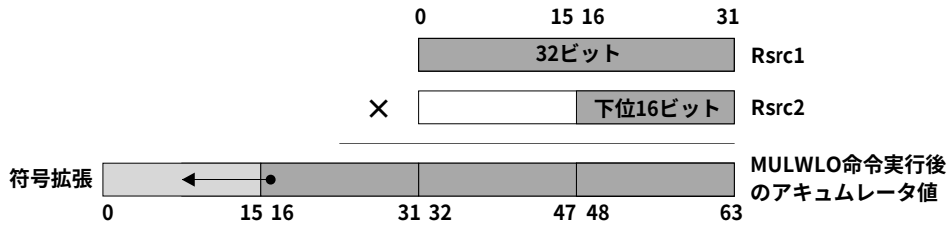
乗算

accumulator = ((signed) Rsrc1 * (signed short) Rsrc2);

【機能】

Rsrc1(32ビット)と、Rsrc2の下位16ビットの乗算を行い、結果をアキュムレータに格納します。ただし、乗算結果の最下位ビットはアキュムレータの最下位ビットにあわせ、アキュムレータのビット0～15に対応する部分は符号拡張されます。Rsrc1(32ビット)と、Rsrc2の下位16ビットは符号付き整数として扱われます。

条件ビット(C)は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0011	src2

MULWLO Rsrc1,Rsrc2

MV 転送命令 Move register MV

【ニーモニック】

```
MV  Rdest,Rsrc
```

【動作】

レジスタ間転送
Rdest = Rsrc;

【機能】

Rsrcの内容をRdestに転送します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	dest	1000	src

MV Rdest,Rsrc

MVFACHI *DSP機能用命令* Move high-order word from accumulator MVFACHI

【ニーモニック】

```
MVFACHI  Rdest
```

【動作】

アキュムレータ～レジスタ間転送
Rdest = (signed int) (accumulator >> 32) ;

【機能】

アキュムレータの上位32ビットの内容をRdestに転送します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	dest	1111	0000

MVFACHI Rdest

MVFACLO

DSP機能用命令

Move low-order word from accumulator

MVFACLO

【ニーモニック】

```
MVFACLO  Rdest
```

【動作】

アキュムレータ～レジスタ間転送

```
Rdest = ( signed int ) accumulator ;
```

【機能】

アキュムレータの下位32ビットの内容をRdestに転送します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	dest	1111	0001

MVFACLO Rdest

MVFACMI *DSP機能用命令* MVFACMI

Move middle-order word from accumulator

【ニーモニック】

```
MVFACMI  Rdest
```

【動作】

アキュムレータ～レジスタ間転送

```
Rdest = ( signed int ) ( accumulator >> 16 ) ;
```

【機能】

アキュムレータのビット16～47の内容をRdestに転送します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	dest	1111	0010

MVFACMI Rdest

MVFC 転送命令 Move from control register MVFC

【ニーモニック】

```
MVFC  Rdest,CRsrc
```

【動作】

レジスタ～制御レジスタ間転送
Rdest = CRsrc ;

【機能】

制御レジスタCRsrcの内容をRdestに転送します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	dest	1001	src

MVFC Rdest,CRsrc

MVTACHI *DSP機能用命令* Move high-order word to accumulator MVTACHI

【ニーモニック】

```
MVTACHI  Rsrc
```

【動作】

アキュムレータ～レジスタ間転送
accumulator [0 : 31] = Rsrc ;

【機能】

Rsrcの内容をアキュムレータの上位32ビット(ビット0～31)に転送します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	src	0111	0000

MVTACHI Rsrc

MVTACLO *DSP機能用命令* MVTACLO

Move low-order word to accumulator

【ニーモニック】

```
MVTACLO  Rsrc
```

【動作】

アキュムレータ～レジスタ間転送
accumulator [32 : 63] = Rsrc ;

【機能】

Rsrcの内容をアキュムレータの下位32ビット(ビット32～63)に転送します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	src	0111	0001

MVTACLO Rsrc

MVTC

転送命令

Move to control register

MVTC

【ニーモニック】

```
MVTC  Rsrc,CRdest
```

【動作】

レジスタ～制御レジスタ間転送
CRdest = Rsrc ;

【機能】

Rsrcの内容を制御レジスタCRdestに転送します。CRdestがPSW(CR0)のとき、条件ビット(C)はその値に依存し、それ以外の場合は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	dest	1010	src

MVTC Rsrc,CRdest

NEG

算術演算命令

Negate

NEG

【ニーモニック】

```
NEG  Rdest,Rsrc
```

【動作】

符号反転

Rdest = 0 - (signed) Rsrc ;

【機能】

Rsrcの内容を符号付き32ビット値として扱い、符号反転して結果をRdestに格納します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0000	dest	0011	src

NEG Rdest,Rsrc

NOP

分岐命令

No operation

NOP

【ニーモニック】

NOP

【動作】

ノーオペレーション
/* */

【機能】

処理はなにも行いません。次の命令から継続して実行されます。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0111	0000	0000	0000	NOP

NOT

論理演算命令

Logical NOT

NOT

【ニーモニック】

```
NOT  Rdest,Rsrc
```

【動作】

論理否定

Rdest = ~ Rsrc ;

【機能】

Rsrcの各ビットの内容を反転して、結果をRdestに格納します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0000	dest	1011	src

NOT Rdest,Rsrc

OR

論理演算命令

OR

OR

【ニーモニック】

```
OR  Rdest,Rsrc
```

【動作】

論理和

Rdest = Rdest ¦ Rsrc ;

【機能】

RdestとRsrcの対応する各ビットの論理和を計算し、結果をRdestに格納します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0000	dest	1110	src

```
OR  Rdest,Rsrc
```

OR3 *論理演算命令* OR3

OR 3-operand

【ニーモニック】

```
OR3  Rdest,Rsrc,#imm16
```

【動作】

論理和
Rdest = Rsrc ¦ (unsigned short) imm16 ;

【機能】

Rsrcと16ビット即値の対応する各ビットの論理和を計算し、結果をRdestに格納します。実行にあたって16ビット即値は32ビットにゼロ拡張されます。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】


```
OR3  Rdest,Rsrc,#imm16
```

RAC

DSP機能用命令

Round accumulator

RAC

【ニーモニック】

RAC

【動作】

飽和処理
```
{ signed64bit tmp;
tmp = ( signed64bit ) accumulator << 1;
tmp = tmp + 0x0000 0000 0000 8000;
if( 0x0000 7fff ffff 0000 < tmp )
   accumulator = 0x0000 7fff ffff 0000;
else if( tmp < 0xffff 8000 0000 0000 )
   accumulator = 0xffff 8000 0000 0000;
else
   accumulator = tmp & 0xffff ffff ffff 0000; }
```

【機能】

アキュムレータの値に対してワードサイズで丸めを行い、その結果をアキュムレータに格納します。この命令は2つの処理から構成されています。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	0000	1001	0000

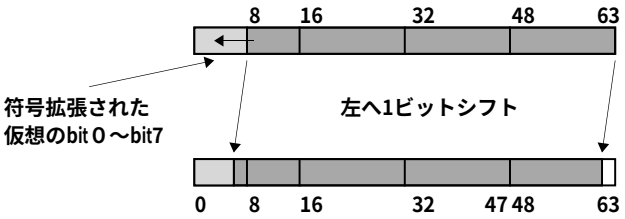
RAC

【機能補足説明】

RAC命令は以下のような処理で構成されています。

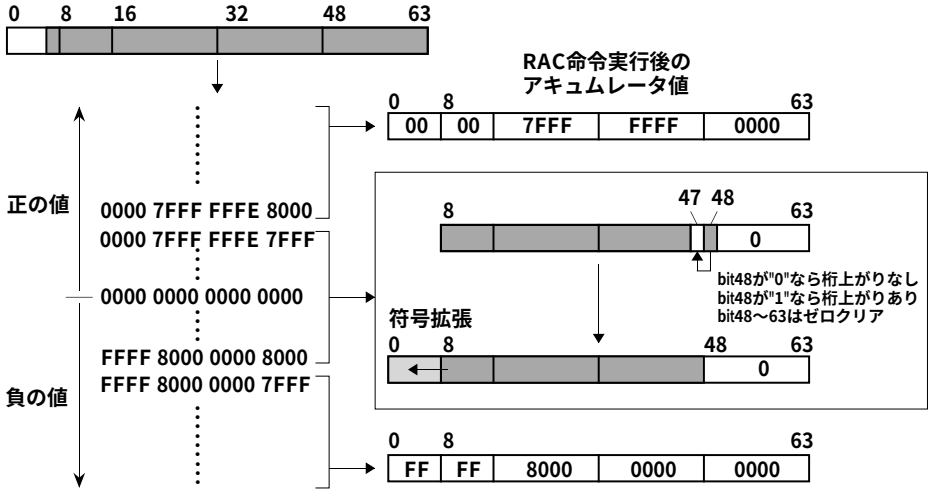
■処理1

アキュムレータの値を、1ビット左シフトします。



■処理2

1ビットの左シフトが反映された仮想のbit0～bit7と、シフト後のbit8～bit63を合わせた64ビットの値にしたがってアキュムレータの値が変化します。



RACH

DSP機能用命令

Round accumulator halfword

RACH

【ニーモニック】

RACH

【動作】

飽和処理

```
{ signed64bit tmp;
tmp = ( signed64bit ) accumulator << 1;
tmp = tmp + 0x0000 0000 8000 0000;
if( 0x0000 7fff 0000 0000 < tmp )
   accumulator = 0x0000 7fff 0000 0000;
else if( tmp < 0xffff 8000 0000 0000 )
   accumulator = 0xffff 8000 0000 0000;
else
   accumulator = tmp & 0xffff ffff 0000 0000; }
```

【機能】

アキュムレータの値に対してハーフワードサイズで丸めを行い、その結果をアキュムレータに格納します。この命令は2つの処理から構成されています。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	0000	1000	0000

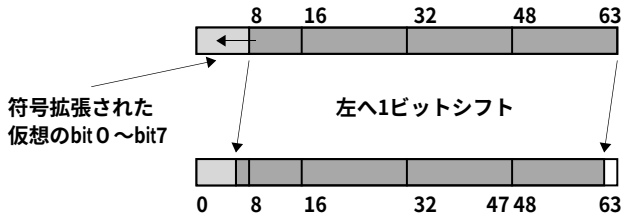
RACH

【機能補足説明】

RACH命令は以下のような処理で構成されています。

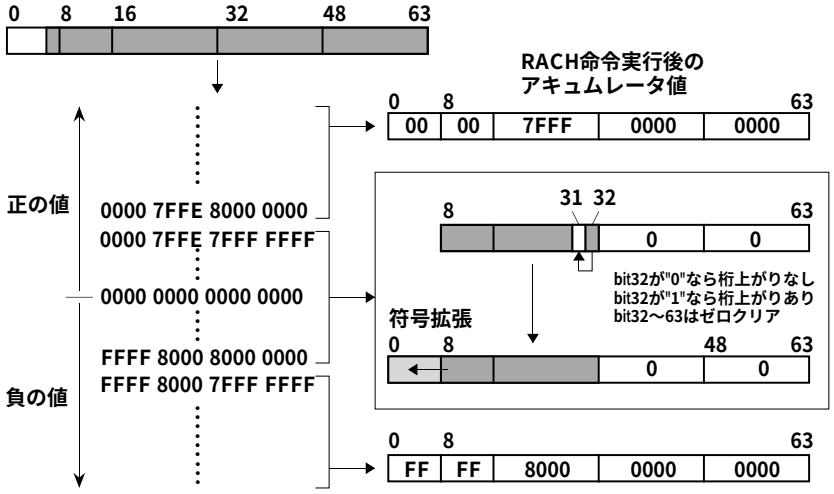
■処理1

アキュムレータの値を、1ビット左シフトします。



■処理2

1ビットの左シフトが反映された仮想のbit0～bit7と、シフト後のbit8～bit63を合わせた64ビットの値にしたがってアキュムレータの値が変化します。



REM

乗除算命令

Remainder

REM

【ニーモニック】

```
REM  Rdest,Rsrc
```

【動作】

符号付き剰余

Rdest = (signed) Rdest % (signed) Rsrc ;

【機能】

RdestをRsrcで除算し、剰余をRdestに格納します。オペランドは符号付き32ビット値として扱われます。商は0方向に丸められ、剰余の符号は被除数と等しくなります。

条件ビット(C)は変化しません。

Rsrcが0のとき、結果は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	0010	src	0000	0000	0000	0000

```
REM  Rdest,Rsrc
```

REMU 乗除算命令 Remainder unsigned REMU

【ニーモニック】

```
REMU  Rdest,Rsrc
```

【動作】

符号なし剰余

Rdest = (unsigned) Rdest % (unsigned) Rsrc ;

【機能】

RdestをRsrcで除算し、剰余をRdestに格納します。オペランドは符号なし32ビット値として扱われます。

条件ビット(C)は変化しません。

Rsrcが0のとき、結果は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	0011	src	0000	0000	0000	0000

```
REMU  Rdest,Rsrc
```

RTE

EIT関連命令

Return from EIT

RTE

【ニーモニック】

RTE

【動作】

EITハンドラからの復帰
SM = BSM ;
IE = BIE ;
C = BC ;
PC = BPC & 0xfffffffc ;

【機能】

PSWレジスタ中のBSM, BIE 及び BC ビットの値をそれぞれSM, IE 及び Cビットに復帰し、BPCで示される番地へ分岐します。
このとき、PSWレジスタ中のBSM, BIE, BCビットは不定になり、BPCも不定になります。

【発生EIT】

なし

【命令フォーマット】

0001	0000	1101	0110

RTE

SETH

転送命令

Set high-order 16-bit

SETH

【ニーモニック】

```
SETH  Rdest,#imm16
```

【動作】

転送命令

Rdest = (signed short) imm16 << 16 ;

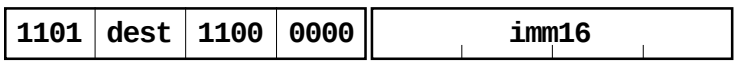
【機能】

16ビット即値をRdestのMSB側16ビットに転送します。このときLSB側16ビットは0になります。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
SETH  Rdest,#imm16
```

SETPSW

ビット操作命令
Set PSW
[M32R-FPU拡張命令]

SETPSW

【ニーモニック】

```
SETPSW  #imm8
```

【動作】

PSWのSM, IE, Cの任意のビットに1をセット。

PSW ¦= imm8&0x000000ff

【機能】

8ビット即値のb0(MSB), b1, b7(LSB)の各ビットと、PSWのSM, IE, Cの各ビットとの論理和を、PSWのSM, IE, Cにそれぞれセットします。8ビット即値のb7(LSB)が1のとき条件ビット(C)は1になり、それ以外の場合は変化しません。

【発生EIT】

なし

【命令フォーマット】

0111	0001	imm8

SETPSW #imm8

【注意事項】

8ビット即値のb2～b6については、"0"を設定してください。

SLL

シフト命令

Shift left logical

SLL

【ニーモニック】

```
SLL   Rdest,Rsrc
```

【動作】

論理左シフト

```
Rdest = Rdest  <<  ( Rsrc & 31 ) ;
```

【機能】

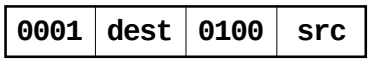
Rsrcで指定された値だけRdestの内容を左に論理シフトします。シフトしたLSB側には0が入ります。RsrcのLSB側5ビットのみ有効です。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
SLL   Rdest,Rsrc
```

SLL3 シフト命令 Shift left logical 3-operand SLL3

【ニーモニック】

```
SLL3  Rdest,Rsrc,#imm16
```

【動作】

論理左シフト

Rdest = Rsrc << (imm16 & 31) ;

【機能】

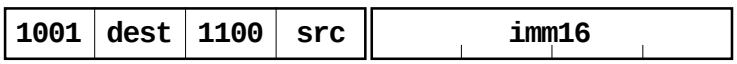
16ビット即値で指定された値だけRsrcの内容を左に論理シフトします。シフトしたLSB側には0が入ります。16ビット即値のLSB側5ビットのみ有効です。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
SLL3  Rdest,Rsrc,#imm16
```

SLLI シフト命令 Shift left logical immediate SLLI

【ニーモニック】

```
SLLI  Rdest,#imm5
```

【動作】

論理左シフト
Rdest = Rdest << imm5 ;

【機能】

5ビット即値で指定された値だけRdestの内容を左に論理シフトます。シフトしたLSB側には0が入ります。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	dest	010	imm5

SLLI Rdest,#imm5

SRA

シフト命令

Shift right arithmetic

SRA

【ニーモニック】

```
SRA  Rdest,Rsrc
```

【動作】

算術右シフト

```
Rdest = ( signed ) Rdest  >>  ( Rsrc & 31 ) ;
```

【機能】

Rsrcで指定された値だけRdestの内容を右に算術シフトします。シフトして空いた側にはMSBにある符号ビットがコピーされて入り、結果はRdestに格納されます。RsrcはLSB側5ビットのみ有効です。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	dest	0010	src

SRA Rdest,Rsrc

SRA3

シフト命令

Shift right arithmetic 3-operand

SRA3

【ニーモニック】

```
SRA3  Rdest,Rsrc,#imm16
```

【動作】

算術右シフト

```
Rdest = ( signed ) Rsrc  >>  ( imm16 & 31 ) ;
```

【機能】

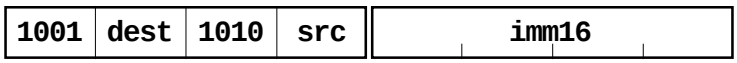
16ビット即値で指定された値だけRsrcの内容を右に算術シフトします。シフトして空いた側にはMSBにある符号ビットがコピーされて入り、結果はRdestに格納されます。16ビット即値のLSB側5ビットのみ有効です。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
SRA3  Rdest,Rsrc,#imm16
```

SRAI

シフト命令

Shift right arithmetic immediate

SRAI

【ニーモニック】

```
SRAI  Rdest,#imm5
```

【動作】

算術右シフト

Rdest = (signed) Rdest >> imm5 ;

【機能】

5ビット即値で指定された値だけRdestの内容を右に算術シフトします。シフトして空いた側にはMSBにある符号ビットがコピーされて入り、結果はRdestに格納されます。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	dest	001	imm5

SRAI Rdest,#imm5

SRL

シフト命令
Shift right logical

SRL

【ニーモニック】

```
SRL   Rdest,Rsrc
```

【動作】

論理右シフト

```
Rdest = ( unsigned ) Rdest  >>  ( Rsrc & 31 ) ;
```

【機能】

Rsrcで指定された値だけRdestの内容を右に論理シフトします。シフトしたMSB側には0が入ります。RsrcのLSB側5ビットのみ有効です。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	dest	0000	src

SRL Rdest,Rsrc

SRL3 シフト命令 Shift right logical 3-operand SRL3

【ニーモニック】

```
SRL3  Rdest,Rsrc,#imm16
```

【動作】

論理右シフト

Rdest = (unsigned) Rsrc >> (imm16 & 31) ;

【機能】

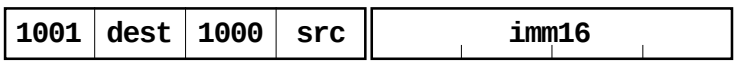
16ビット即値で指定された値だけRsrcの内容を右に論理シフトします。シフトしたMSB側には0が入ります。16ビット即値のLSB側5ビットのみ有効です。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
SRL3  Rdest,Rsrc,#imm16
```

SRLI シフト命令 Shift right logical immediate SRLI

【ニーモニック】

```
SRLI  Rdest,#imm5
```

【動作】

論理右シフト

```
Rdest = ( unsigned ) Rdest  >>  ( imm5 & 31 ) ;
```

【機能】

5ビット即値で指定された値だけRdestの内容を右にシフトします。シフトしたMSB側には0が入り、結果はRdestに格納されます。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	dest	000	imm5

SRLI Rdest,#imm5

ST

ロード／ストア命令
Store

ST

【ニーモニック】

① **ST Rsrc1,@Rsrc2**
② **ST Rsrc1,@+Rsrc2**
③ **ST Rsrc1,@-Rsrc2**
④ **ST Rsrc1,@(disp16,Rsrc2)**

【動作】

レジスタからメモリへのストア
① * (signed int *) Rsrc2 = Rsrc1;
② Rsrc2 += 4, * (signed int *) Rsrc2 = Rsrc1;
③ Rsrc2 -= 4, * (signed int *) Rsrc2 = Rsrc1;
④ * (signed int *) (Rsrc2 + (signed short) disp16) = Rsrc1;

【機能】

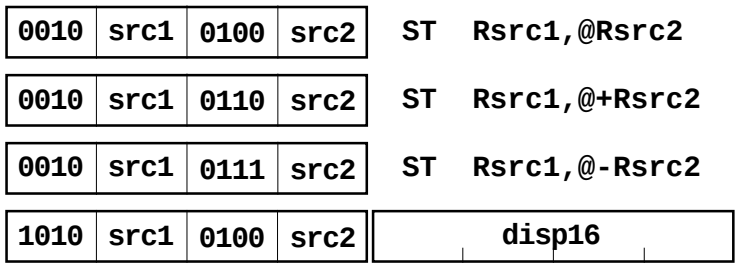
① Rsrc1の内容を、Rsrc2で指定する番地のメモリにストアします。
② Rsrc2を4インクリメントした後、Rsrc1の内容を、インクリメントしたRsrc2で指定する番地のメモリにストアします。
③ Rsrc2を4デクリメントした後、Rsrc1の内容を、デクリメントしたRsrc2で指定する番地のメモリにストアします。
④ Rsrc1の内容を、Rsrc2と16ビットのディスプレースメントで指定する番地のメモリにストアします。ディスプレースメントの値は、アドレス計算の前に符号拡張されます。

条件ビット(C)は変化しません。

【発生EIT】

アドレス例外(AE)

【命令フォーマット】



ST Rsrc1,@(disp16,Rsrc2)

STB

ロード／ストア命令
Store byte

STB

【ニーモニック】

① **STB Rsrc1,@Rsrc2**
② **STB Rsrc1,@(disp16,Rsrc2)**

【動作】

レジスタからメモリへのストア
① * (signed char *) Rsrc2 = Rsrc1;
② * (signed char *) (Rsrc2 + (signed short) disp16) = Rsrc1;

【機能】

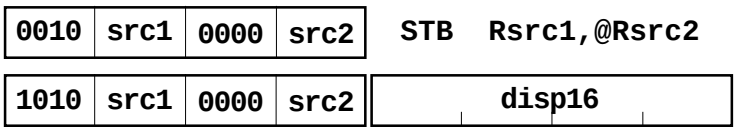
① Rsrc1のLSB側バイトデータを、Rsrc2で指定する番地のメモリにストアします。
② Rsrc1のLSB側バイトデータを、Rsrc2と16ビットディスプレースメントで指定する番地のメモリにストアします。ディスプレースメントは、アドレス計算の前に符号拡張されます。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】



STB Rsrc1,@(disp16,Rsrc2)

STH

ロード／ストア命令

Store halfword

[M32R-FPU拡張ニーモニック]

STH

【ニーモニック】

① **STH Rsrc1,@Rsrc2**
② **STH Rsrc1,@Rsrc2+** [M32R-FPU拡張ニーモニック]
③ **STH Rsrc1,@(disp16,Rsrc2)**

【動作】

レジスタからメモリへのストア
① * (signed short *) Rsrc2 = Rsrc1 ;
② * (signed short *) Rsrc2 = Rsrc1, Rsrc2 += 2 ;
③ * (signed short *) (Rsrc2 + (signed short) disp16) = Rsrc1 ;

【機能】

① Rsrc1のLSB側のハーフワードデータを、Rsrc2で指定する番地のメモリにストアします。
② Rsrc1のLSB側のハーフワードデータを、Rsrc2で指定する番地のメモリにストアし、その後でRsrc2を2インクリメントします。
③ Rsrc1のLSB側のハーフワードデータを、Rsrc2と16ビットディスプレースメントで指定する番地のメモリにストアします。ディスプレースメントは、アドレス計算の前に符号拡張されます。

条件ビット(C)は変化しません。

【発生EIT】

アドレス例外(AE)

【命令フォーマット】

0010	src1	0010	src2		STH Rsrc1,@Rsrc2

0010	src1	0011	src2		STH Rsrc1,@Rsrc2+

1010	src1	0010	src2	disp16

STH Rsrc1,@(disp16,Rsrc2)

SUB 算術演算命令 Subtract SUB

【ニーモニック】

```
SUB  Rdest,Rsrc
```

【動作】

減算
Rdest = Rdest - Rsrc;

【機能】

Rdestの内容からRsrcの内容を引き算し、結果をRdestに格納します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0000	dest	0010	src

SUB Rdest,Rsrc

SUBV

算術演算命令

Subtract with overflow checking

SUBV

【ニーモニック】

```
SUBV  Rdest,Rsrc
```

【動作】

減算(オーバフローチェック付き)

```
Rdest = ( signed ) Rdest - ( signed ) Rsrc;
C = overflow ? 1 : 0;
```

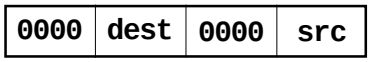
【機能】

Rdestの内容からRsrcの内容を引き算し、結果をRdestに格納します。
条件ビット(C)は引き算の結果がオーバフローしたときセットされ、それ以外はクリアされます。

【発生EIT】

なし

【命令フォーマット】



SUBV Rdest,Rsrc

SUBX 算術演算命令 Subtract with borrow SUBX

【ニーモニック】

```
SUBX  Rdest,Rsrc
```

【動作】

減算(ボロー付き)

```
Rdest = ( unsigned ) Rdest - ( unsigned ) Rsrc - C;
C = borrow ? 1 : 0;
```

【機能】

Rdestの内容から(Rsrcの内容＋条件ビット(C))の値を引き算し、結果をRdestに格納します。
条件ビット(C)は引き算の結果が符号なし32ビット整数として表現できないときセットされ、それ以外はクリアされます。

【発生EIT】

なし

【命令フォーマット】

0000	dest	0001	src

SUBX Rdest,Rsrc

TRAP

EIT関連命令
Trap

TRAP

【ニーモニック】

```
TRAP  #imm4
```

【動作】

トラップの発生

```
BPC = PC + 4;
BSM = SM;
BIE = IE;
BC = C;
IE = 0;
C = 0;
call_trap_handler ( imm4 );
```

【機能】

指定された番号のトラップを発生します。

PSWレジスタ中のSM, IE及びCビットの値をそれぞれBSM, BIE及びBCビットに退避し、IE及びCビットをそれぞれ"0"に更新します。

【発生EIT】

トラップ(TRAP)

【命令フォーマット】

0001	0000	1111	imm4

```
TRAP  #imm4;
```

UNLOCK　ロード／ストア命令 Store unlocked　UNLOCK

【ニーモニック】

```
UNLOCK  Rsrc1,@Rsrc2
```

【動作】

ロック解除付きストア
if (LOCK == 1) { * (signed int *) Rsrc2 = Rsrc1; }
LOCK = 0;

【機能】

LOCKビットが1のとき、Rsrc1をRsrc2で指定された番地のメモリにストアして、LOCKビットをクリアします。
条件ビット(C)は変化しません。
LOCKビットが0のときは、ストア動作を行いません。
LOCKビットはCPU内部にあり、LOCK命令とUNLOCK命令により操作できます。ユーザがこのビットを直接リード/ライトすることはできません。

LOCKビットとはCPU内部のビットでCPU以外のバス権を要求する回路に対してバス権の受付を制御するビットです。
CPU以外のバス権の要求は、各種マイコンにより異なりますが、各ユーザーズマニュアルを参照してください。

【発生EIT】

アドレス例外(AE)

【命令フォーマット】

0010	src1	0101	src2

UNLOCK Rsrc1,@Rsrc2

UTOF

浮動小数点命令

Unsigned integer to float

[M32R-FPU拡張命令]

UTOF

【ニーモニック】

```
UTOF  Rdest,Rsrc
```

【動作】

符号なし整数から単精度浮動小数点数への変換

Rdest = (float) (unsigned int) Rsrc ;

【機能】

Rsrcに格納された32ビット符号なし整数を単精度浮動小数点数に変換し、結果をRdestに格納します。結果はFPSRのRMフィールドに従って丸められます。条件ビット(C)は変化しません。H'0000 0000は丸めモードに関係なく、“ +0 ”として扱われます。

【発生EIT】

浮動小数点例外(FPE)

・精度異常例外(IXCT)

【命令フォーマット】

1101	src	0000	0000	0100	dest	0100	0000

```
UTOF  Rdest,Rsrc
```

XOR 論理演算命令 Exclusive OR XOR

【ニーモニック】

```
XOR  Rdest,Rsrc
```

【動作】

排他的論理和
Rdest = Rdest ^ Rsrc;

【機能】

RdestとRsrcの対応するビットの排他的論理和を計算し、Rdestに格納します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0000	dest	1101	src

XOR Rdest,Rsrc

XOR3 論理演算命令 Exclusive OR 3-operand XOR3

【ニーモニック】

```
XOR3  Rdest,Rsrc,#imm16
```

【動作】

排他的論理和
Rdest = Rsrc ^ (unsigned short) imm16;

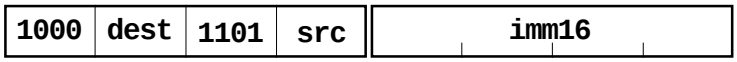
【機能】

Rsrcと16ビット即値の対応するビットの排他的論理和を計算し、Rdestに格納します。16ビット即値は演算前に32ビットにゼロ拡張されます。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】




```
XOR3  Rdest,Rsrc,#imm16
```

付録

付録1 16進命令コード対応表
付録2 命令セット一覧
付録3 パイプライン処理機構
付録4 命令処理時間
付録5 IEEE754規格の概要
付録6 M32R-FPU仕様補足説明
付録7 注意事項

付録1　16進命令コード対応表

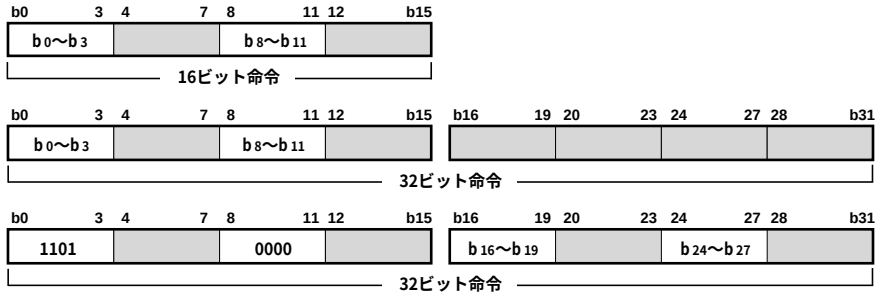
各命令のビットパターンとニーモニックの対応を以下に示します。
太枠で囲んでいる命令は、M32R-FPUで拡張された命令です。

付表1.1.1　M32R-FPU　命令コード対応表

	b8～b11	0000	0001	0010	0011	0100	0101	0110	0111
	b0～b3 16進表記	0	1	2	3	4	5	6	7
16ビット命令	0000 0	**SUBV** Rdest,Rsrc	**SUBX** Rdest,Rsrc	**SUB** Rdest,Rsrc	**NEG** Rdest,Rsrc	**CMP** Rsrc1,Rsrc2	**CMPU** Rsrc1,Rsrc2		
	0001 1	**SRL** Rdest,Rsrc		**SRA** Rdest,Rsrc		**SLL** Rdest,Rsrc		**MUL** Rdest,Rsrc	
	0010 2	**STB** Rsrc1,@Rsrc2		**STH** Rsrc1,@Rsrc2	**STH** Rsrc1,@Rsrc2+	**ST** Rsrc1,@Rsrc2	**UNLOCK** Rsrc1,@Rsrc2	**ST** Rsrc1,@+Rsrc2	**ST** Rsrc1,@-Rsrc2
	0011 3	**MULHI** Rsrc1,Rsrc2	**MULLO** Rsrc1,Rsrc2	**MULWHI** Rsrc1,Rsrc2	**MULWLO** Rsrc1,Rsrc2	**MACHI** Rsrc1,Rsrc2	**MACLO** Rsrc1,Rsrc2	**MACWHI** Rsrc1,Rsrc2	**MACWLO** Rsrc1,Rsrc2
	0100 4	**ADDI** Rdest,#imm8							
	0101 5	**SRLI** Rdest,#imm5		**SRAI** Rdest,#imm5		**SLLI** Rdest,#imm5			**MVTACHI, MVTACLO**（★2）
	0110 6	**LDI** Rdest,#imm8							
	0111 7	**NOP**（★1）		**BC, BNC, BL, BRA,**		**SETPSW, CLRPSW**	（★1）		
32ビット命令	1000 8					**CMPI** Rsrc,#imm16	**CMPUI** Rsrc,#imm16		
	1001 9	**DIV** Rdest,Rsrc	**DIVU** Rdest,Rsrc	**REM** Rdest,Rsrc	**REMU** Rdest,Rsrc				
	1010 A	**STB** Rsrc1,@(disp16,Rsrc2)		**STH** Rsrc1,@(disp16,Rsrc2)		**ST** Rsrc1,@(disp16,Rsrc2)		**BSET** #bitpos,@(disp16,Rsrc)	**BCLR** #bitpos,@(disp16,Rsrc)
	1011 B	**BEQ** Rsrc1,Rsrc2,pcdisp16	**BNE** Rsrc1,Rsrc2,pcdisp16						
	1100 C								
	1101 D	**FPU拡張命令**							
	1110 E	**LD24** Rdest,#imm24							
	1111 F	**BC, BNC, BL, BRA**（★1）							

FPU拡張命令（b0～b3＝1101, b8～b11＝0000のFPU拡張命令の場合）

	b24～b27	0000	0001	0010	0011	0100	0101	0110	0111
	b16～b19 16進表記	0	1	2	3	4	5	6	7
32ビット命令	0000 0	**FADD**				**FSUB**			
	0001 1	**FMUL**							
	0010 2	**FDIV**							
	0011 3	**FMADO**				**FMSUB**			
	0100 4	**ITOF**				**UTOF**			
	0101 5								
	0110 6								
	0111 7								



1000	1001	1010	1011	1100	1101	1110	1111	b8～b11 / 16進表記	b0～b3	
8	9	A	B	C	D	E	F			
ADDV Rdest,Rsrc	**ADDX** Rdest,Rsrc	**ADD** Rdest,Rsrc	**NOT** Rdest,Rsrc	**AND** Rdest,Rsrc	**XOR** Rdest,Rsrc	**OR** Rdest,Rsrc	**BTST** #bitpos,Rsrc	0	0000	16ビット命令
MV Rdest,Rsrc	**MVFC** Rdest,CRsrc	**MVTC** Rsrc,CRdest		**JL, JMP** (★1)	**RTE**		**TRAP** #imm4	1	0001	
LDB Rdest,@Rsrc	**LDUB** Rdest,@Rsrc	**LDH** Rdest,@Rsrc	**LDUH** Rdest,@Rsrc	**LD** Rdest,@Rsrc	**LOCK** Rdest,@Rsrc	**LD** Rdest,@Rsrc+		2	0010	
								3	0011	
ADDI Rdest,#imm8								4	0100	
RACH	**RAC**						**MVFACHI, MVFACLO, MVFACMI** (★2)	5	0101	
LDI Rdest,#imm8								6	0110	
BC, BNC, BL, BRA (★1)								7	0111	
ADDV3 Rdest,Rsrc,#imm16		**ADD3** Rdest,Rsrc,#imm16		**AND3** Rdest,Rsrc,#imm16	**XOR3** Rdest,Rsrc,#imm16	**OR3** Rdest,Rsrc,#imm16		8	1000	32ビット命令
SRL3 Rdest,Rsrc,#imm16		**SRA3** Rdest,Rsrc,#imm16		**SLL3** Rdest,Rsrc,#imm16			**LDI** Rdest,#imm16	9	1001	
LDB Rdest,@(disp16,Rsrc)	**LDUB** Rdest,@(disp16,Rsrc)	**LDH** Rdest,@(disp16,Rsrc)	**LDUH** Rdest,@(disp16,Rsrc)	**LD** Rdest,@(disp16,Rsrc)				A	1010	
BEQZ Rsrc,pcdisp16	**BNEZ** Rsrc,pcdisp16	**BLTZ** Rsrc,pcdisp16	**BGEZ** Rsrc,pcdisp16	**BLEZ** Rsrc,pcdisp16	**BGTZ** Rsrc,pcdisp16			B	1011	
								C	1100	
				SETH Rdest,#imm16				D	1101	
LD24 Rdest,#imm24								E	1110	
BC, BNC, BL, BRA (★1)								F	1111	

1000	1001	1010	1011	1100	1101	1110	1111	b24～b27 / 16進表記	b16～b19	
0	1	2	3	4	5	6	7			
				FCMP	**FCMPE**			0	0000	32ビット命令
								1	0001	
								2	0010	
								3	0011	
FTOI				**FTOS**				4	0100	
								5	0101	
								6	0110	
								7	0111	

注. 表中の★1～2で示された命令は、b0～b3、及びb8～b11以外に、下記に示すビットのパターンにより命令が決定されます。各命令のビットパターンの詳細については、「3.2 命令詳細説明」の該当命令の ページを参照してください。

★1. b4～b7、★2. b12～b15

付録2　命令セット一覧

M32R-FPUの命令セット一覧を以下に示します(アルファベット順)。

ニーモニック		動作	条件ビット(C)
ADD	Rdest,Rsrc	Rdest = Rdest + Rsrc	—
ADD3	Rdest,Rsrc,#imm16	Rdest = Rsrc + (sh)imm16	—
ADDI	Rdest,#imm8	Rdest = Rdest + (sb)imm8	—
ADDV	Rdest,Rsrc	Rdest = (s)Rdest + (s)Rsrc	変化
ADDV3	Rdest,Rsrc,#imm16	Rdest = (s)Rsrc + (sh)imm16	変化
ADDX	Rdest,Rsrc	Rdest = (u)Rdest + (u)Rsrc + C	変化
AND	Rdest,Rsrc	Rdest = Rdest & Rsrc	—
AND3	Rdest,Rsrc,#imm16	Rdest = Rsrc & (uh)imm16	—
BC	pcdisp8	if(C) PC=PC+((sb)pcdisp8<<2)	—
BC	pcdisp24	if(C) PC=PC+((s24)pcdisp24<<2)	—
BCLR	#bitpos,@(disp16,Rsrc)	*(sb *)(Rsrc + (sh)disp16) & = ~(1<<(7-bitpos))	—
BEQ	Rsrc1,Rsrc2,pcdisp16	if(Rsrc1 == Rsrc2) PC=PC+((sh)pcdisp16<<2)	—
BEQZ	Rsrc,pcdisp16	if(Rsrc == 0) PC=PC+((sh)pcdisp16<<2)	—
BGEZ	Rsrc,pcdisp16	if((s)Rsrc >= 0) PC=PC+((sh)pcdisp16<<2)	—
BGTZ	Rsrc,pcdisp16	if((s)Rsrc > 0) PC=PC+((sh)pcdisp16<<2)	—
BL	pcdisp8	R14=PC+4,PC=PC+((sb)pcdisp8<<2)	—
BL	pcdisp24	R14=PC+4,PC=PC+((s24)pcdisp24<<2)	—
BLEZ	Rsrc,pcdisp16	if((s)Rsrc <= 0) PC=PC+((sh)pcdisp16<<2)	—
BLTZ	Rsrc,pcdisp16	if((s)Rsrc < 0) PC=PC+((sh)pcdisp16<<2)	—
BNC	pcdisp8	if(!C) PC=PC+((sb)pcdisp8<<2)	—
BNC	pcdisp24	if(!C) PC=PC+((s24)pcdisp24<<2)	—
BNE	Rsrc1,Rsrc2,pcdisp16	if(Rsrc1 != Rsrc2) PC=PC+((sh)pcdisp16<<2)	—
BNEZ	Rsrc,pcdisp16	if(Rsrc != 0) PC=PC+((sh)pcdisp16<<2)	—
BRA	pcdisp8	PC=PC+((sb)pcdisp8<<2)	—
BRA	pcdisp24	PC=PC+((s24)pcdisp24<<2)	—
BSET	#bitpos,@(disp16,Rsrc)	*(sb *)(Rsrc + (sh)disp16) | = (1<<(7-bitpos))	—
BTST	#bitpos,Rsrc	(Rsrc>>(7-bitpos))&1	変化
CLRPSW	#imm8	PSW & = ~imm8 | 0xffffff00	変化
CMP	Rsrc1,Rsrc2	(s)Rsrc1 < (s)Rsrc2	変化
CMPI	Rsrc,#imm16	(s)Rsrc < (sh)imm16	変化
CMPU	Rsrc1,Rsrc2	(u)Rsrc1 < (u)Rsrc2	変化
CMPUI	Rsrc,#imm16	(u)Rsrc < (u)((sh)imm16)	変化
DIV	Rdest,Rsrc	Rdest = (s)Rdest / (s)Rsrc	—
DIVU	Rdest,Rsrc	Rdest = (u)Rdest / (u)Rsrc	—
FADD	Rdest,Rsrc1,Rsrc2	Rdest = Rsrc1 + Rsrc2	—
FCMP	Rdest,Rsrc1,Rsrc2	Rdest = (Rsrc1 == Rsrc2)?32'h00000000:((Rsrc1< Rsrc2)?{1.31'bx}:{0.31'bx}	—
FCMPE	Rdest,Rsrc1,Rsrc2	FCMP with Exception when unordered	—
FDIV	Rdest,Rsrc1,Rsrc2	Rdest = Rsrc1 / Rsrc2	—

ニーモニック		動作	条件ビット(C)
FMADD	Rdest,Rsrc1,Rsrc2	Rdest = Rdest + Rsrc1 * Rsrc2	—
FMSUB	Rdest,Rsrc1,Rsrc2	Rdest = Rdest - Rsrc1 * Rsrc2	—
FMUL	Rdest,Rsrc1,Rsrc2	Rdest = Rdest * Rsrc2	—
FSUB	Rdest,Rsrc1,Rsrc2	Rdest = Rsrc1 - Rsrc2	—
FTOI	Rdest,Rsrc	Rdest = (s)Rsrc2	—
FTOS	Rdest,Rsrc	Rdest = (sh)Rsrc	—
ITOF	Rdest,Rsrc	Rdest = (float)Rsrc	—
JL	Rsrc	R14 = PC+4, PC = Rsrc	—
JMP	Rsrc	PC = Rsrc	—
LD	Rdest,@(disp16,Rsrc)	Rdest = *(s *)(Rsrc+(sh)disp16)	—
LD	Rdest,@Rsrc	Rdest = *(s *)Rsrc	—
LD	Rdest,@Rsrc+	Rdest = *(s *)Rsrc, Rsrc += 4	—
LD24	Rdest,#imm24	Rdest = imm24 & 0x00ffffff	—
LDB	Rdest,@(disp16,Rsrc)	Rdest = *(sb *)(Rsrc+(sh)disp16)	—
LDB	Rdest,@Rsrc	Rdest = *(sb *)Rsrc	—
LDH	Rdest,@(disp16,Rsrc)	Rdest = *(sh *)(Rsrc+(sh)disp16)	—
LDH	Rdest,@Rsrc	Rdest = *(sh *)Rsrc	—
LDI	Rdest,#imm16	Rdest = (sh)imm16	—
LDI	Rdest,#imm8	Rdest = (sb)imm8	—
LDUB	Rdest,@(disp16,Rsrc)	Rdest = *(ub *)(Rsrc+(sh)disp16)	—
LDUB	Rdest,@Rsrc	Rdest = *(ub *)Rsrc	—
LDUH	Rdest,@(disp16,Rsrc)	Rdest = *(uh *)(Rsrc+(sh)disp16)	—
LDUH	Rdest,@Rsrc	Rdest = *(ub *)Rsrc	—
LOCK	Rdest,@Rsrc	LOCK = 1, Rdest = *(s *)Rsrc	—
MACHI	Rsrc1,Rsrc2	accumulator += (s)(Rsrc1 & 0xffff0000) * (s)((s)Rsrc2>>16)	—
MACLO	Rsrc1,Rsrc2	accumulator += (s)(Rsrc1<<16) * (sh)Rsrc2	—
MACWHI	Rsrc1,Rsrc2	accumulator += (s)Rsrc1 * (s)((s)Rsrc2>>16)	—
MACWLO	Rsrc1,Rsrc2	accumulator += (s)Rsrc1 * (sh)Rsrc2	—
MUL	Rdest,Rsrc	Rdest = (s)Rdest * (s)Rsrc	—
MULHI	Rsrc1,Rsrc2	accumulator = (s)(Rsrc1 & 0xffff0000) * (s)((s)Rsrc2>>16)	—
MULLO	Rsrc1,Rsrc2	accumulator = (s)(Rsrc1<<16) * (sh)Rsrc2	—
MULWHI	Rsrc1,Rsrc2	accumulator = (s)Rsrc1 * (s)((s)Rsrc2>>16)	—
MULWLO	Rsrc1,Rsrc2	accumulator = (s)Rsrc1 * (sh)Rsrc2	—
MV	Rdest,Rsrc	Rdest = Rsrc	—
MVFACHI	Rdest	Rdest = accumulater >> 32	—
MVFACLO	Rdest	Rdest = accumulator	—
MVFACMI	Rdest	Rdest = accumulator >> 16	—
MVFC	Rdest,CRsrc	Rdest = CRsrc	—
MVTACHI	Rsrc	accumulator[0:31] = Rsrc	—
MVTACLO	Rsrc	accumulator[32:63] = Rsrc	—
MVTC	Rsrc,CRdest	CRdest = Rsrc	変化

ニーモニック	動作	条件ビット(C)
NEG Rdest,Rsrc	Rdest = 0 - Rsrc	—
NOP	/*no-operation*/	—
NOT Rdest,Rsrc	Rdest = ~Rsrc	—
OR Rdest,Rsrc	Rdest = Rdest | Rsrc	—
OR3 Rdest,Rsrc,#imm16	Rdest = Rsrc | (uh)imm16	—
RAC	Round the 32-bit value in the accumulator	—
RACH	Round the 16-bit value in the accumulator	—
REM Rdest,Rsrc	Rdest = (s)Rdest % (s)Rsrc	—
REMU Rdest,Rsrc	Rdest = (u)Rdest % (u)Rsrc	—
RTE	PC = BPC & 0xfffffffc, PSW[SM,IE,C] = PSW[BSM,BIE,BC]	変化
SETH Rdest,#imm16	Rdest = imm16 << 16	—
SETPSW #imm8	PSW | = imm8&0x000000ff	変化
SLL Rdest,Rsrc	Rdest = Rdest << (Rsrc & 31)	—
SLL3 Rdest,Rsrc,#imm16	Rdest = Rsrc << (imm16 & 31)	—
SLLI Rdest,#imm5	Rdest = Rdest << imm5	—
SRA Rdest,Rsrc	Rdest = (s)Rdest >> (Rsrc & 31)	—
SRA3 Rdest,Rsrc,#imm16	Rdest = (s)Rsrc >> (imm16 & 31)	—
SRAI Rdest,#imm5	Rdest = (s)Rdest >> imm5	—
SRL Rdest,Rsrc	Rdest = (u)Rdest >> (Rsrc & 31)	—
SRL3 Rdest,Rsrc,#imm16	Rdest = (u)Rsrc >> (imm16 & 31)	—
SRLI Rdest,#imm5	Rdest = (u)Rdest >> imm5	—
ST Rsrc1,@(disp16,Rsrc2)	*(s *)(Rsrc2+(sh)disp16) = Rsrc1	—
ST Rsrc1,@+Rsrc2	Rsrc2 += 4, *(s *)Rsrc2 = Rsrc1	—
ST Rsrc1,@-Rsrc2	Rsrc2 -= 4, *(s *)Rsrc2 = Rsrc1	—
ST Rsrc1,@Rsrc2	*(s *)Rsrc2 = Rsrc1	—
STB Rsrc1,@(disp16,Rsrc2)	*(sb *)(Rsrc2+(sh)disp16) = Rsrc1	—
STB Rsrc1,@Rsrc2	*(sb *)Rsrc2 = Rsrc1	—
STH Rsrc1,@(disp16,Rsrc2)	*(sh *)(Rsrc2+(sh)disp16) = Rsrc1	—
STH Rsrc1,@Rsrc2	*(sh *)Rsrc2 = Rsrc1	—
STH Rsrc1,@Rsrc2+	*(sh *)Rsrc2 = Rsrc1, Rsrc2 += 2	—
SUB Rdest,Rsrc	Rdest = Rdest - Rsrc	—
SUBV Rdest,Rsrc	Rdest = Rdest - Rsrc	変化
SUBX Rdest,Rsrc	Rdest = Rdest - Rsrc - C	変化
TRAP #n	PSW[BSM,BIE,BC] = PSW[SM,IE,C] PSW[SM,IE,C] = PSW[SM,0,0] Call trap-handler number-n	変化
UNLOCK Rsrc1,@Rsrc2	if(LOCK) { *(s *)Rsrc2 = Rsrc1; } LOCK=0	—
UTOF Rdest,Rsrc	Rdest = (float)(unsigned int) Rsrc;	—
XOR Rdest,Rsrc	Rdest = Rdest ^ Rsrc	—
XOR3 Rdest,Rsrc,#imm16	Rdest = Rsrc ^ (uh)imm16	—

```
where:
typedef singed int      s;  /* 32 bit signed integer (word)*/
typedef unsigned int    u;  /* 32 bit unsigned integer (word)*/
typedef signed short    sh; /* 16 bit signed integer (halfword)*/
typedef unsigned short  uh; /* 16 bit unsigned integer (halfword)*/
typedef signed char     sb; /*  8 bit signed integer (byte)*/
typedef unsigned char   ub; /*  8 bit unsigned integer (byte)*/
```

付録3　パイプライン処理機構

付録3.1　命令とパイプライン処理

付図3.1.1に命令とパイプライン処理について示します。

■ ロード／ストア命令の場合

※MEM1ステージのサイクル数はアクセス対象によって変化しますが、
MEM2ステージは常に1サイクルで実行されます。

■ その他の整数命令の場合

※乗除算命令のようなマルチサイクル命令はEステージが複数サイクル実行
されます。

■ FPU命令の場合（FMADD、FMSUB以外）

※E1, E2ステージはEステージと同時に実行することはできません。
※FDIV命令時のE1ステージは14サイクルとなります。

■ FPU命令の場合（FMADD、FMSUB時）

※EM, EAステージはE, E1ステージと同時に実行することはできません。

※同じ名前の演算ステージは同時に実行することはできません。異なる名前の演算ステージは
基本的に並列に実行が可能ですが、以下の組み合わせは並列実行できません。

・EステージとE1, E2, EM, EAステージ
・E1ステージとEM, EAステージ

※命令の実行結果を後続命令が使用する場合、前方命令の実行結果をレジスタファイルを経由せず
後続命令の実行ステージに供給することができるケースがあります（バイパス処理）。
バイパス処理が発生するケースを次に示します。

・Eステージに続くWBステージ→　E, E1, EMステージ
・MEM2ステージに続くWBステージ　→　E, E1, EM, EAステージ

付図3.1.1　命令とパイプライン処理

以下にM32R-FPUのパイプラインの各ステージの概要を示します。

● **IFステージ(命令フェッチステージ)**

メモリから命令をフェッチ(IF)します。M32R-FPUは命令キューを備えており、D(デコード)ステージのデコード処理完了とは無関係に、命令キューがいっぱいになるまでフェッチを続けます。

あらかじめ命令キューに命令がある場合は、命令キューから読み出した命令を命令デコーダに渡します。

● **Dステージ(デコードステージ)**

Dステージ前半は、命令のデコード処理(DEC1)を行います。後半は命令デコード処理の続き(DEC2)と、レジスタのフェッチ(RF)を行います。

● **Eステージ(実行ステージ)**

演算やアドレス計算など(OP)を行います。

直前の命令の演算結果を必要とする処理の場合は、Eステージの前半でバイパス(BYP)を行います。

● **E1, EM, EAステージ(実行ステージ)**

FPU命令実行の第一ステージです。EM, EAステージはFMADD, FMSUB命令のみ使用されます。他のFPU命令ではE1ステージが使用されます。

● **E2ステージ(実行ステージ)**

FPU命令実行の第二ステージです。主に丸めを実行します。

● **MEMステージ(メモリアクセスステージ)**

オペランドのアクセス(OA)を行います。ロード／ストア命令実行時のみこのステージが使われます。

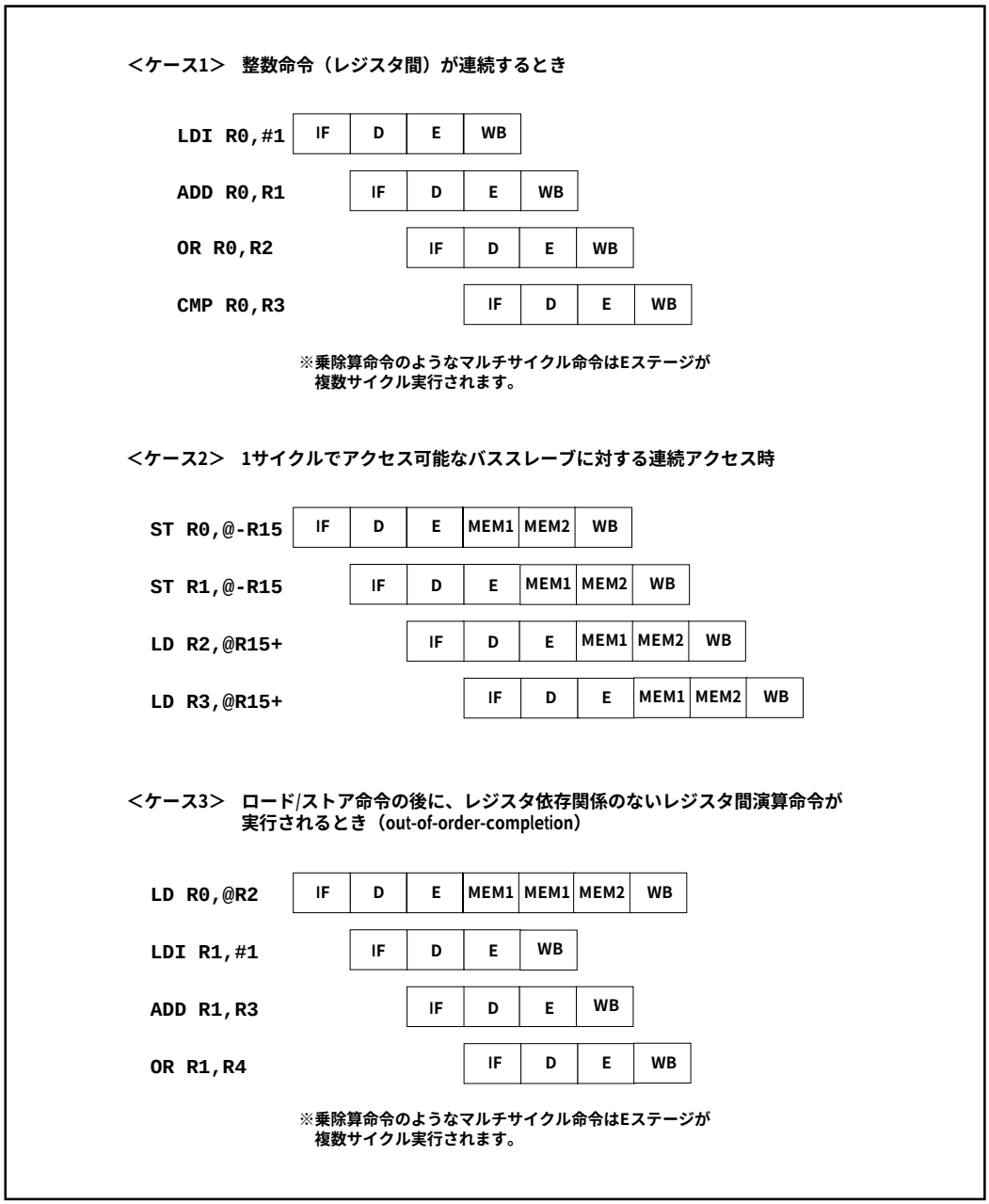
● **WBステージ(ライトバックステージ)**

演算結果やフェッチしてきたデータをレジスタ、またはアキュムレータに書き込みます(WB)。

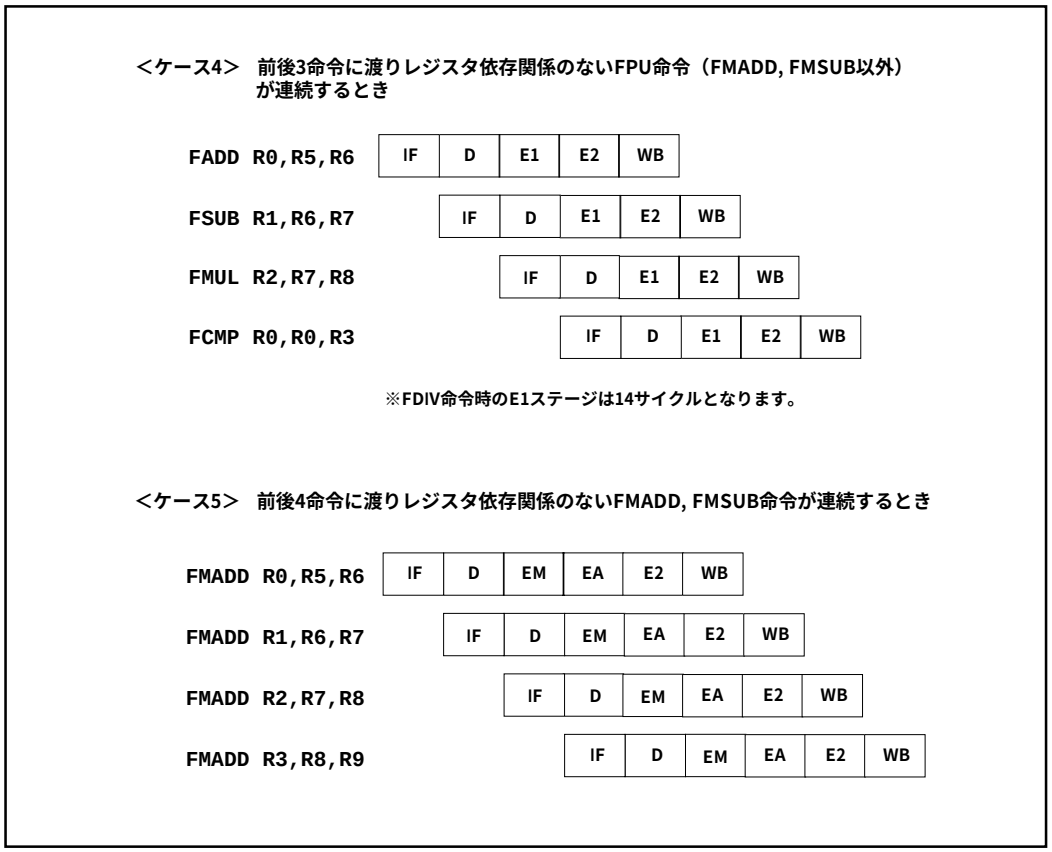
付録3.2　パイプライン基本動作

(1) パイプラインがスムーズに流れる場合

M32R-FPUのパイプラインが乱れることなく流れ、1命令が1サイクルで連続実行可能となるケースの例を示します(本ケースはあくまで例であり、全てのケースを示しているわけではありません)。



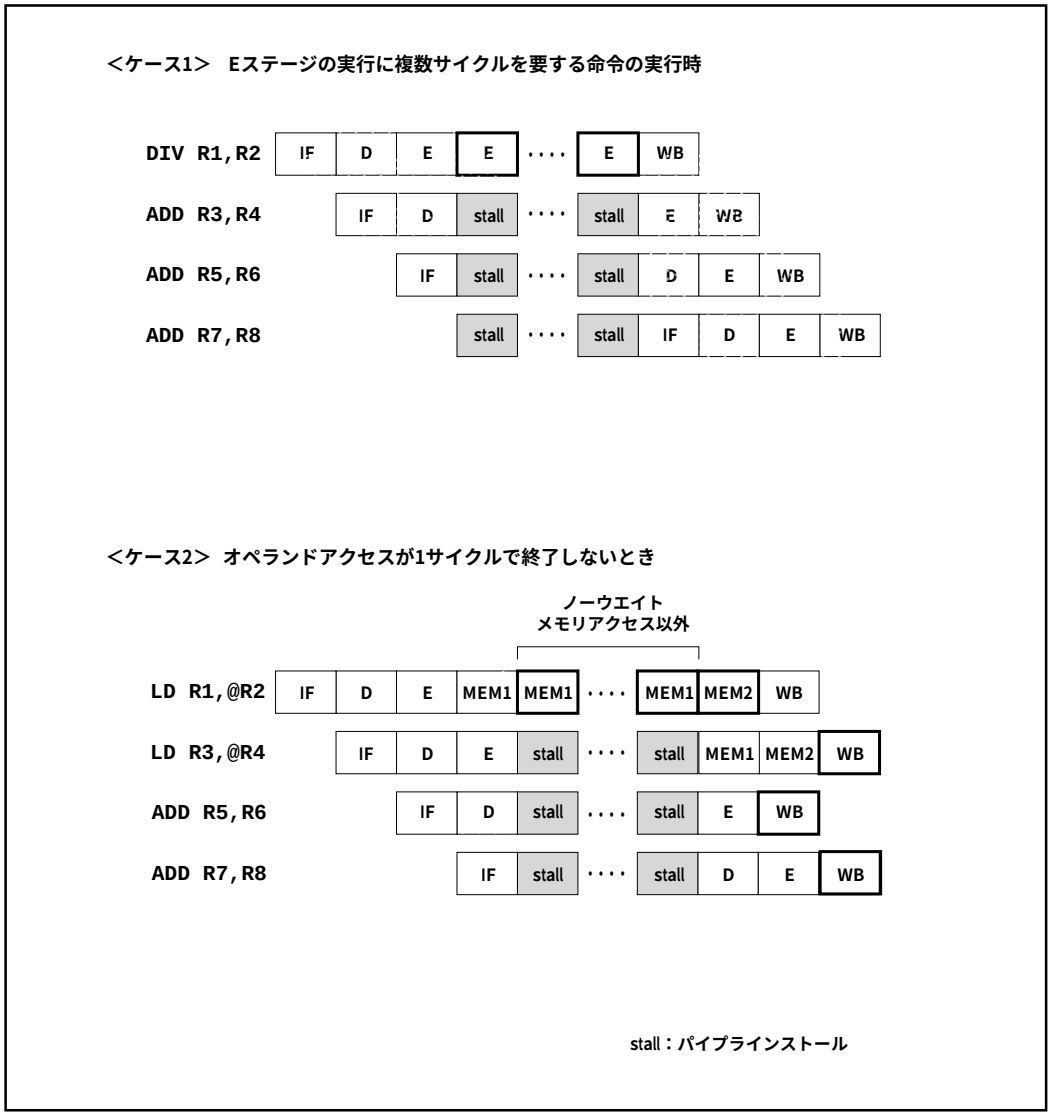
付図3.2.1　パイプライン動作がスムーズに流れる場合(1)



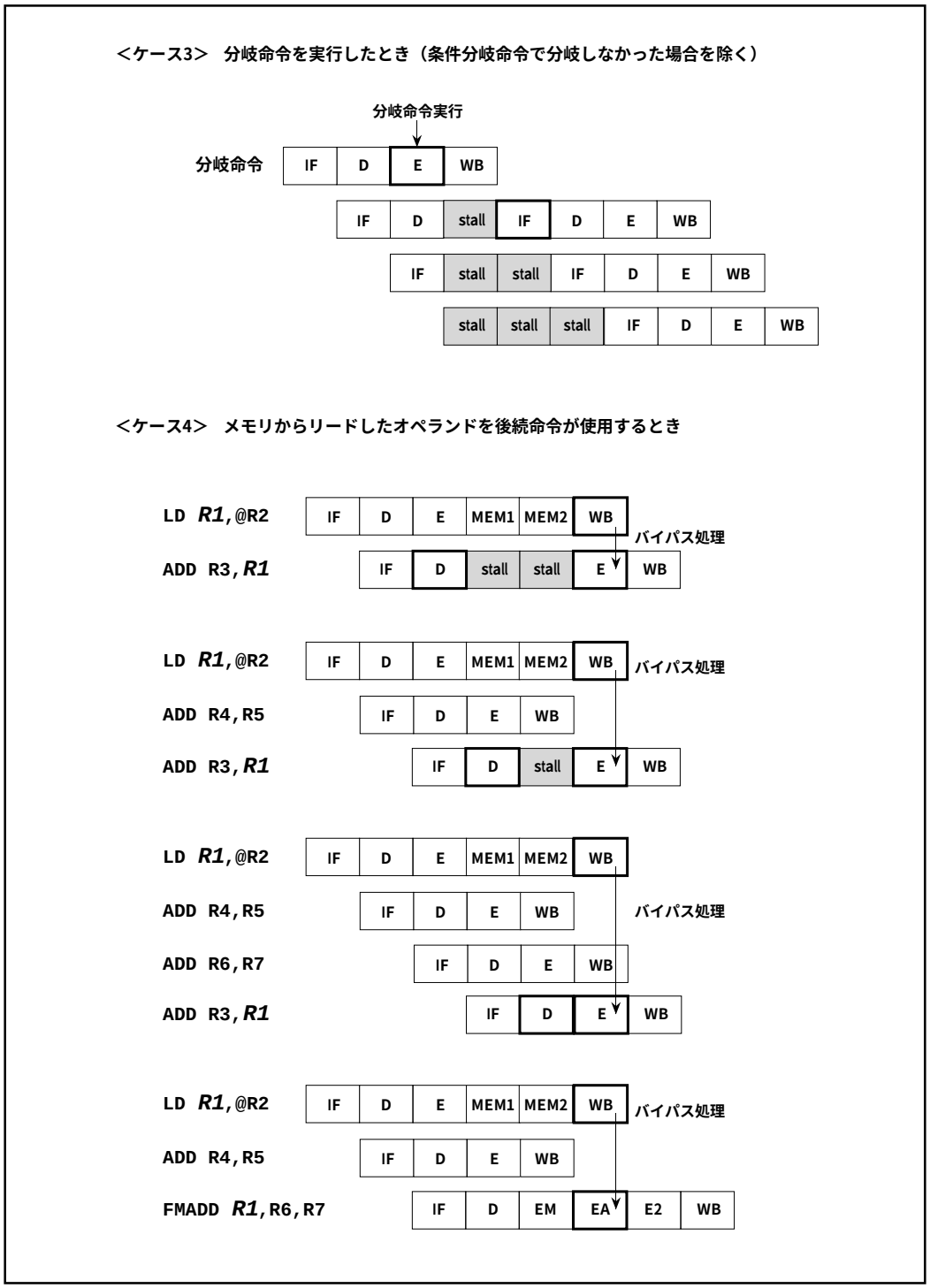
付図3.2.2　パイプライン動作がスムーズに流れる場合(2)

(2) パイプライン動作が乱れる場合

各ステージでの処理や分岐命令の実行などによりパイプライン動作が乱れることがあります。以下にそれらの基本的な動作をケース別に示します。

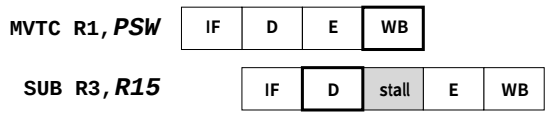


付図3.2.3 パイプライン動作が乱れる場合(1)

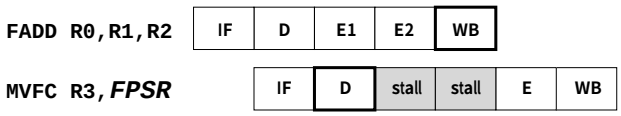


付図3.2.4　パイプライン動作が乱れる場合(2)

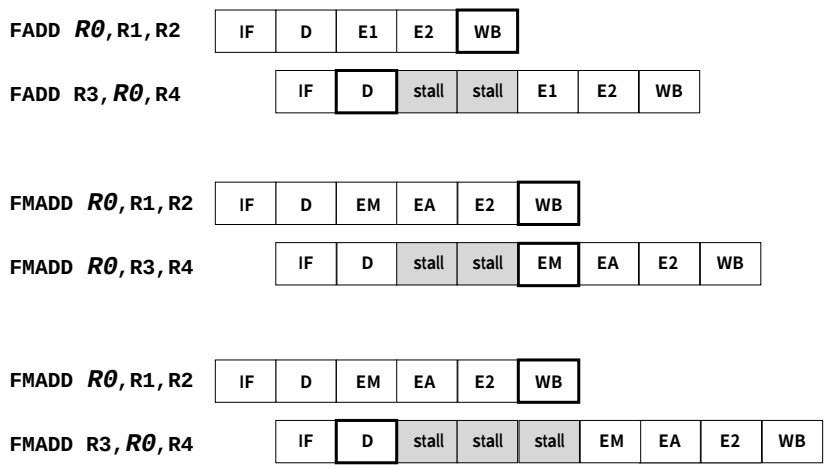
<ケース5> MVTC, SETPSW, CLRPSW命令でPSWに書き込み後、後続命令がR15を読み出す場合



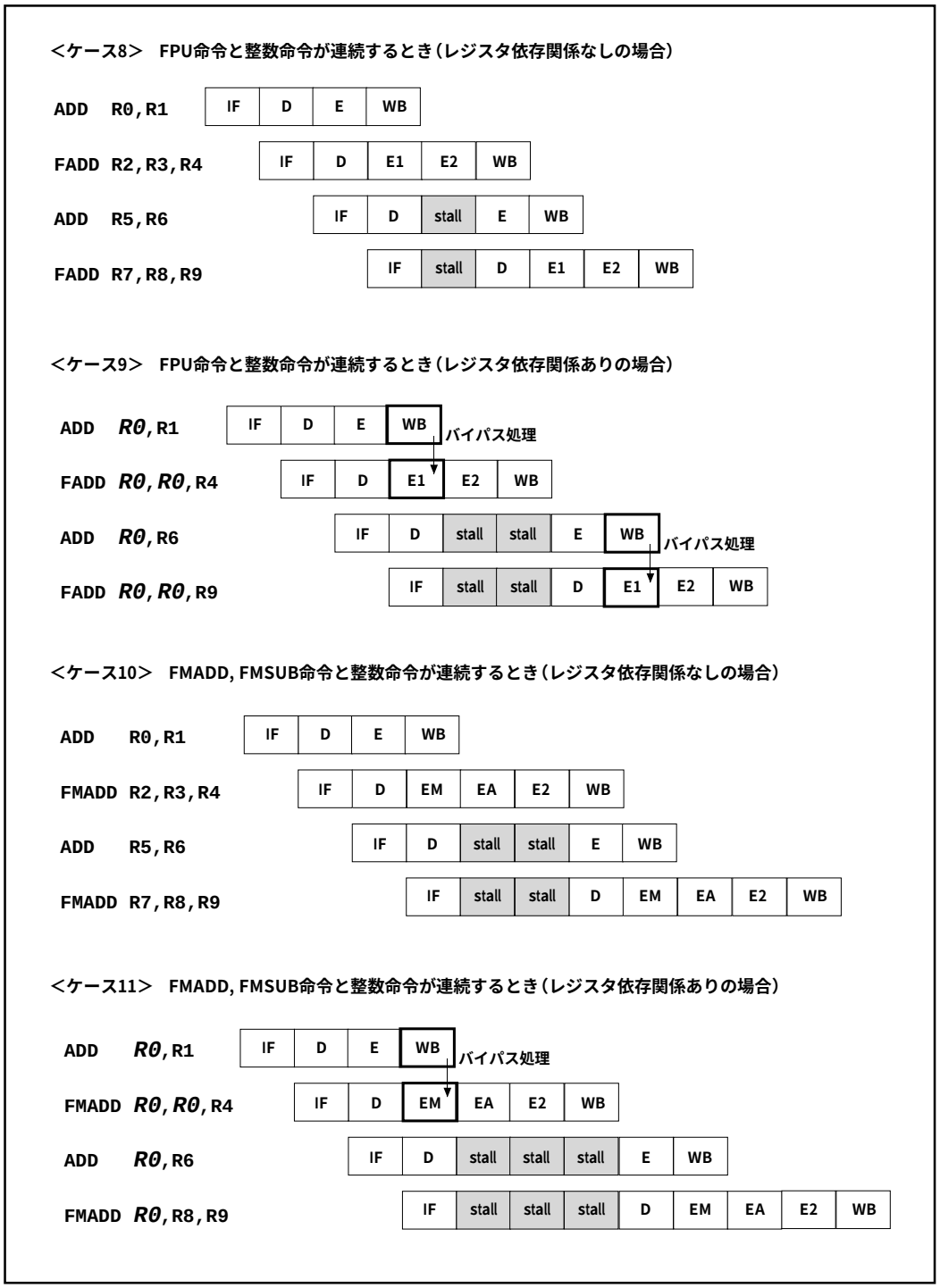
<ケース6> FPU命令実行後にMVFC命令でFPSRをアクセスする場合



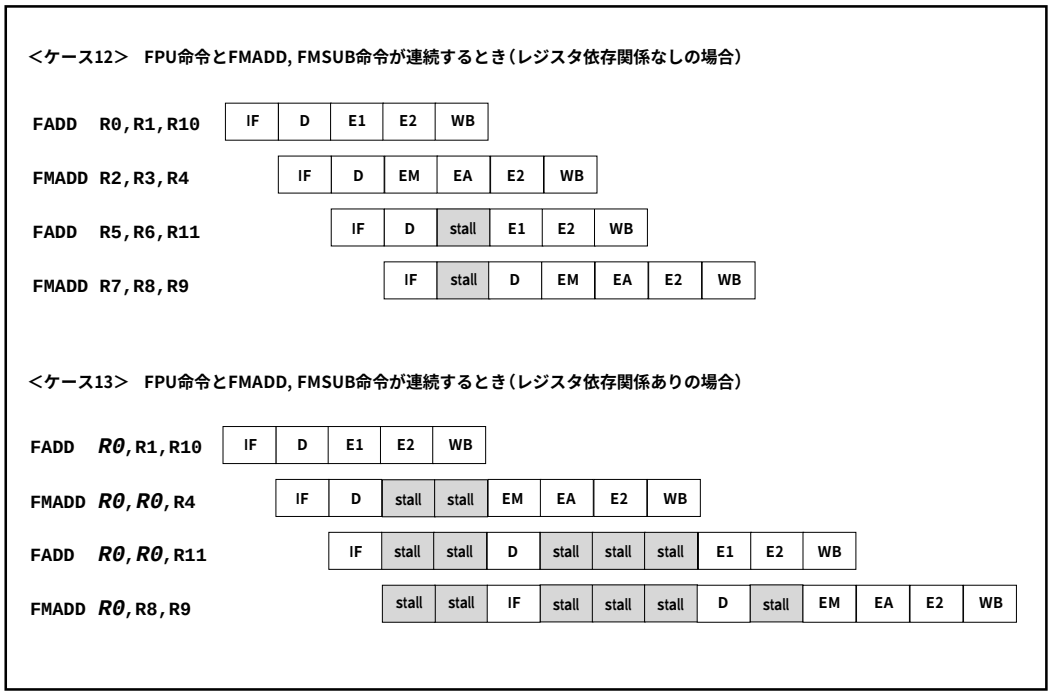
<ケース7> FPU命令の演算結果を後続命令が使用する場合



付図3.2.5 パイプライン動作が乱れる場合(3)



付図3.2.6 パイプライン動作が乱れる場合（4）



付図3.2.7　パイプライン動作が乱れる場合（5）

付録4　命令処理時間

M32R-FPUは、通常Eステージ(注1)における命令実行サイクル数を命令処理時間として代表しますが、パイプラインの動作によっては、それ以外のステージが処理時間に影響を与えることがあります。特に分岐命令を実行した場合の次命令においては、IF(命令フェッチ)、D(デコード)、E(実行)の各ステージの処理時間を考慮に入れることが必要です。

以下にM32R-FPUの各パイプラインステージごとの命令処理時間を示します。なお、IF(命令フェッチステージ)及びMEM(メモリアクセスステージ)の処理時間は、M32Rファミリ各機種のインプリメンテーション(外部バスインタフェースの構成等)に依存するため、これらのステージの実行時間については、M32Rファミリ各機種のユーザーズマニュアルを参照ください。

注1．FPU命令ではE1及びEMステージを使用します。

付表4.1.1　各パイプラインステージにおける命令処理時間[FPU命令以外]

	各ステージにおける実行サイクル数					
命令	IF	D	E	MEM1	MEM2	WB
ロード命令(LD, LDB, LDUB, LDH, LDUH, LOCK)	R(注1)	1	1	R(注1)	1	1
ストア命令(ST, STB, STH, UNLOCK)	R(注1)	1	1	W(注1)	1	(1)(注2)
BSET, BCLR命令	R(注1)	1	R(注1) +3	W(注1)	1	–
乗算命令(MUL)	R(注1)	1	3	–	–	1
除算／剰余命令(DIV, DIVU, REM, REMU)	R(注1)	1	37	–	–	1
上記以外の命令(DSP機能用命令, BTST, SETPSW, CLRPSWを含む)	R(注1)	1	1	–	–	1

注1. R, Wで示される実行サイクル数は、M32Rファミリ各機種ユーザーズマニュアルを参照してください。
注2. ストア命令のうち、レジスタ間接＋レジスタ更新アドレッシングモードを持つもののみWBステージに1サイクル必要です(それ以上は必要ありません)。

付表4.1.2　各パイプラインステージにおける命令処理時間[FPU命令]

	各ステージにおける実行サイクル数						
命令	IF	D	E1	EM	EA	E2	WB
FMADD, FMSUB命令	R(注1)	1	–	1	1	1	1
FDIV命令	R(注1)	1	14	–	–	1	1
上記以外のFPU命令	R(注1)	1	1	–	–	1	1

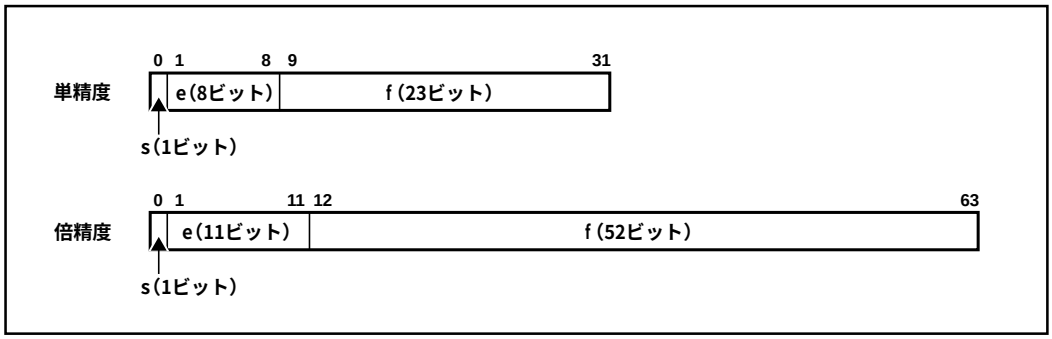
注1. R, Wで示される実行サイクル数は、M32Rファミリ各機種ユーザーズマニュアルを参照してください。

付録5　IEEE754規格の概要

IEEE754規格の概要について説明します。M32R-FPUはソフトとハードの組み合わせにより IEEE754規格の内容を実現することができます。

付録5.1　浮動小数点フォーマット

以下に浮動小数点フォーマットについて説明します。



付図5.1.1　浮動小数点フォーマット

s(sign)：符号ビットです。0のとき正の数を、1のとき負の数を表します。
e(exponent)：指数部です。単精度は127、倍精度は1023を加えて正の数にした値で表現します(ゲタ履き表現)。
f(fraction)：少数部です。少数点部分を表します。

上記の記号を用いることにより、浮動少数点数(正規化数)は次の式で表現されます。

単精度：(-1)＾s×1.f×2＾(e-127)
倍精度：(-1)＾s×1.f×2＾(e-1023)

・特殊な数、上記の表現に従わない数として、±∞、±0、NaN(非数：Not a Number)、非正規化数などがあります。
・上記以外に拡張倍精度などのフォーマットもあります。
☆M32R-FPUでは単精度フォーマットのみをサポートしており、倍精度フォーマットはソフトウエアライブラリの形でサポートします。

付表5.1.1　単精度浮動少数点のビット表現

指数部		表現される数
ゲタ履き前	ゲタ履き後 （ゲタ＝0111 1111）	
0111 1111（＋127）	1111 1110	正規化数 （1．0…0×2＾-126～1．1…1×2＾127 の範囲の絶対値を表現可能）
・・・	・・・	
1000 0010（-126）	0000 0001	
（1000 0001（-127））	0000 0000	少数部＝オール0：±0 少数部≠オール0：非正規化数
（1000 0000（-128））	1111 1111	少数部＝オール0：±∞ 少数部≠オール0：NaN（少数部の最上位ビットの値によりさらにSNaNとQNaNに分かれる）

（1）非正規化数

1．0…0×2＾-126よりも小さな絶対値を持つ数を表す数です。単精度の非正規化数は次の式で表現される。

（-1）＾s×0.f×2＾-126

（2）NaN（非数：Not a Number）

SNaN（Signaling NaN）：少数部の最上位ビットが0であるNaNです。SNaNを演算のソースオペランドとして使用すると、無効演算例外を発生します。変数の初期値として使用することにより、プログラムバグの発見に役立ちます。なお、SNaNはハードウエアが生成することはありません。

QNaN（Quiet NaN）：少数部の最上位ビットが1であるNaNです。QNaNを演算のソースオペランドとして使用しても、無効演算例外は発生しません（比較、フォーマット変換を除く）。演算によって伝播するため、EIT処理を実行せずに結果だけを見てデバッグを行うことができます。なお、QNaNは演算によりハードウエアが生成します。

付録5.2　丸め

IEEE754では以下の4つの丸めモードを規定しています。

付表5.2.1　4つの丸めモード

丸めモード	動作
最近傍への丸め（デフォルト）	無限の有効桁を持つとして計算した場合の結果と近い方の値へ丸める。中間時は結果が偶数になる方向へ丸める。
-∞方向への丸め	結果の値が小さくなる方向へ丸める。
+∞方向への丸め	結果の値が大きくなる方向へ丸める。
0への丸め	結果の絶対値が小さくなる方向へ丸める（単純な切り捨て）。

・「最近傍への丸め」はデフォルトのモードであり、最も正確な値を返します。
・「-∞方向への丸め」、「+∞方向への丸め」、「0への丸め」は区間演算（Interval arithmetic）を使用した、精度保証を行うときに使用します。

付録5.3　例外

IEEE754では以下の5つの例外を規定しています。例外が発生したときEIT処理を実行させるかどうかは、浮動小数点ステータスレジスタで選択することができます。

（1）オーバフロー例外（OVF）

演算結果の絶対値が、浮動小数点フォーマットで表現可能な値よりも大きくなった場合に発生します。付表5.3.1にOVFが発生したときの演算結果を示します。

付表5.3.1　オーバフロー例外発生時の演算結果

丸めモード	結果の符号	演算結果	
		オーバフロー例外によるEIT処理マスク時	オーバフロー例外によるEIT処理の実行設定時
-∞	+	+MAX	round（×2 ^ -a） a＝192（単精度）、 1536（倍精度）
	-	-∞	
+∞	+	+∞	
	-	-MAX	
0	+	+MAX	
	-	-MAX	
最近傍	+	+∞	
	-	-∞	

注．・オーバフロー例外によるEIT処理マスク時に、オーバフロー例外が発生すると、同時に精度異常例外が発生します。
+MAX＝H'7F7F FFFF, -MAX＝H'7F7F FFFF

(2) アンダフロー例外 (UDF)

演算結果の絶対値が、浮動小数点フォーマットの正規化数で表現可能な値より小さくなった場合に発生します。付表5.3.2にUDFが発生したときの演算結果を示します。

付表5.3.2 アンダフロー例外発生時の演算結果

演算結果	
アンダフロー例外による EIT処理マスク時	アンダフロー例外による EIT処理の実行設定時
非正規化数 (例外フラグのセットは丸め発生時のみ)	round(×2 ^ a) a=192(単精度)、1536(倍精度)

注．・演算結果に丸めが生じていたときは、同時に精度異常例外が発生します。

(3) 精度異常 (IXCT)

演算結果が、無限の有効桁を持つと仮定して計算したときの結果と異なっていたときに発生します。付表5.3.3にIXCTの発生条件と演算結果を示します。

付表5.3.3 精度異常例外発生時の演算結果

発生条件	演算結果	
	精度異常例外による EIT処理マスク時	精度異常例外による EIT処理の実行設定時
オーバフロー例外によるEITマスク状態での オーバフローの発生	オーバフローの表参照	同左
丸めの発生	丸め後の値	同左

(4) ゼロ除算例外 (DIV0)

0でない有限数を0で割ったときに発生します。付表5.3.4にDIV0が発生したときの演算結果を示します。

付表5.3.4 ゼロ除算例外発生時の演算結果

被除数	演算結果	
	ゼロ除算例外による EIT処理マスク時	ゼロ除算例外による EIT処理の実行設定時
0でない有限数	±∞(符号は除数、被除数の符号の排他的倫理和となる)	(デスティネーション変化なし)

なお次の場合はDIV0は発生しません。ご注意ください。

被除数	動作
0	無効演算例外発生
∞	例外は発生しない(結果は"∞")

（5）無効演算例外（IVLD）

無効な演算が実行された際に発生します。付表5.3.5にIVLDの発生条件と演算結果を示します。

付表5.3.5　無効演算例外発生時の演算結果

発生条件	演算結果	
	無効演算例外による EIT処理マスク時	無効演算例外による EIT処理の実行設定時
SNaNオペランドに対する演算	QNaN	（デスティネーション変化なし）
+∞-(+∞), -∞-(-∞)		
0×∞		
0÷0、∞÷∞		
0より小さい数に対するルート演算		
整数変換がオーバフローしたとき NaN、∞を整数変換したとき	未定義	
<、>の比較をNaNに対して行ったとき	（変化なし）	

なお、次の演算はいかなる例外も発生しないので注意が必要である。

√(-0)：-0を返す

∞÷0：∞を返す(符号は除数と非除数の符号の排他的倫理和)

■用語説明

・例外

浮動小数点命令の実行により発生する特別な事象。例外の発生によりEIT処理を実行させるかどうかは、浮動小数点ステータスレジスタの各イネーブルビットで選択することができる。例外の発生自体をマスクすることはできません。

・EIT 処理

例外の発生によって引き起こされる動作のことで、浮動小数点例外ベクタアドレスへのジャンプや、それに附随する一連の例外処理シーケンスの起動を意味します。例外発生時にEIT処理を実行させるかどうかは、浮動小数点ステータスレジスタの各イネーブルビットで選択することができます。

・演算の中間結果

指数と仮数のビット数が無限にあると仮定して計算を行ったときの値です。実際のインプリメンテーションにおいては指数と仮数のビット数は有限であり、中間結果に丸めを施すことによって最終的な演算結果を得ることができます。

付録6　M32R-FPU仕様補足説明

付録6.1　FMADD、FMSUB補足説明：FMUL＋FADDの2命令で演算したときとの比較

FMADD、FMSUBの1命令で演算したときと、FMUL＋FADDの2命令で演算したときとの違いについて説明します。

付録 6.1.1　丸めモード

FMUL＋FADDの2命令で演算した時の丸めモードは、FMUL, FADDともにFPSRのRMフィールドの設定に従います。一方FMADD, FMSUB命令のステップ1(乗算ステージ)の結果はFPSRのRMフィールドの設定によらず、0方向に丸められます。

付録 6.1.2　ステップ１で例外発生時の動作

例として次の2つの命令例を比較します。

● **FMUL ＋ FADD：**

```
FMUL    R3, R1, R2   (R3 = R1 * R2)
FADD    R0, R3, R0   (R0 = R3 + R0)
```

● **FMADD, FMSUB：**

```
FMADD   R0, R1, R2   (R0 = R0 + R1 * R2)
```

注．レジスタの対応関係が上記以外のケースでは、以下に示す動作と異なる部分が出てきます。

(1) ステップ１でオーバフロー発生時

＜EO＝0, EX＝0時：OVFとIXCT発生＞

R0の種類	条件		FMUL＋FADDの動作	FMADDの動作
正規化数, 0	－		R0＝OVF即値(注1)＋R0	R0＝OVF即値(注2)
∞	OVF即値がR0と	EV＝0	IVLD発生　R0＝H'7FFF FFFF	同左
	反対符号の∞の時	EV＝1	IVLD発生　EIT発生 R0＝保持	同左
	上記以外	－	R0＝∞(元と同じ値)	同左
非正規化数	DN＝0		UIPL発生　EIT発生 R0＝保持	同左
	DN＝1		R0＝OVF即値(注1)	同左
QNaN	－		R0＝保持(QNaN)	同左
SNaN	EV＝0		IVLD発生 R0＝QNaN化されたR0	同左
	EV＝1		IVLD発生　EIT発生 R0＝保持(SNaN)	同左

注1．オーバフロー例外によるEIT処理マスク時のOVF発生時の即値については「付表5.3.1　オーバフロー例外時の演算結果」を参照してください。

注2．ステップ1は「0方向への丸め」固定ですが、OVF時の即値は丸めモードに従います。値については「付表5.3.1　オーバフロー例外時の演算結果」を参照してください。なお、丸めモードが「最近傍への丸め」ときはOVF即値＝∞となるため、R0の値はFMUL＋FADD時と同じになります。

＜EO＝1時：OVF発生＞

R0の種類	条件	FMUL＋FADDの動作	FMADDの動作
正規化数, 0, ∞	－	FMUL終了時点でEIT発生 R0＝保持	EIT発生　R0＝保持
非正規化数	DN＝0	同上	UIPL発生　EIT発生 R0＝保持
	DN＝1	同上	EIT発生　R0＝保持
QNaN	－	同上	同上
SNaN	EV＝0	同上	IVLD発生　EIT発生 R0＝保持
	EV＝1	同上	同上

(2) ステップ1でアンダフロー発生時

＜EU＝0, DN＝1時：UDF発生＞

R0の種類	条件	FMUL＋FADDの動作	FMADDの動作
正規化数, 0, ∞	－	R0＝R0＋0	同左
非正規化数	－	R0＝0	同左
QNaN	－	R0＝保持(QNaN)	同左
SNaN	EV＝0	R0＝QNaN化されたR0 IVLD発生	同左
	EV＝1	R0＝保持(SNaN) IVLD発生　EIT発生	同左

＜EU＝0, DN＝0時：UDFとUIPL発生＞

R0の種類	条件	FMUL＋FADDの動作	FMADDの動作
正規化数, 0, ∞	－	FMUL終了時点でEIT発生 R0＝保持	EIT発生　R0＝保持
非正規化数	－	同上	同上
QNaN	－	同上	同上
SNaN	EV＝0	同上	IVLD発生　EIT発生 R0＝保持
	EV＝1	同上	同上

＜EU＝1時：UDF発生＞

R0の種類	条件	FMUL＋FADDの動作	FMADDの動作
正規化数, 0, ∞	－	FMUL終了時点でEIT発生 R0＝保持	EIT発生　R0＝保持
非正規化数	DN＝0	同上	UIPL発生　EIT発生 R0＝保持
	DN＝1	同上	EIT発生　R0＝保持
QNaN	－	同上	同上
SNaN	EV＝0	同上	IVLD発生　EIT発生 R0＝保持
	EV＝1	同上	同上

(3) ステップ 1 で無効演算例外発生時

■ R1, R2 の内少なくとも一方が SNaN のとき

＜EV＝0時：IVLD発生＞

R0の種類	条件	FMUL＋FADDの動作	FMADDの動作
正規化数, 0, ∞	－	R0＝R3（QNaN化されたSNaN）	同左
非正規化数	DN＝0	R0＝R3（QNaN化されたSNaN）	同左
	DN＝1	R0＝R3（QNaN化されたSNaN）	同左
QNaN	－	R0＝保持（QNaN）	同左
SNaN	－	R0＝QNaN化されたR0	同左

＜EV＝1時：IVLD発生＞

R0の種類	条件	FMUL＋FADDの動作	FMADDの動作
正規化数, 0, ∞	－	FMUL終了時点でEIT発生 R0＝保持	EIT発生　R0＝保持
非正規化数	DN＝0	同上	UIPL発生　EIT発生 R0＝保持
	DN＝1	同上	EIT発生　R0＝保持
QNaN	－	同上	同上
SNaN	－	同上	同上

■ R1, R2 で×∞を行ったとき：

＜EV＝0時：IVLD発生＞

R0の種類	条件	FMUL＋FADDの動作	FMADDの動作
正規化数, 0, ∞	－	R0＝H'7FFF FFFF	同左
非正規化数	DN＝0	R0＝H'7FFF FFFF	同左
	DN＝1	R0＝H'7FFF FFFF	同左
QNaN	－	R0＝保持（QNaN）	同左
SNaN	－	R0＝QNaN化されたR0	同左

＜EV＝1時：IVLD発生＞

『R1, R2の内少なくとも一方がSNaNのとき』と同じです。

(4) ステップ1で精度異常例外発生時

■丸めの発生による精度異常発生時：

＜EX＝0時：IXCT発生＞

R0の種類	条件	FMUL＋FADDの動作	FMADDの動作
正規化数, 0, ∞	－	R0＝R1*R2の丸め後の値＋R0	同左
非正規化数	DN＝0	UIPL発生　EIT発生 R0＝保持	同左
	DN＝1	R0＝R1*R2の丸め後の値	同左
QNaN	－	R0＝保持(QNaN)	同左
SNaN	EV＝0	IVLD発生 R0＝QNaN化されたR0	同左
	EV＝1	IVLD発生　EIT発生 R0＝保持(SNaN)	同左

＜EX＝1時：IXCT発生＞

R0の種類	条件	FMUL＋FADDの動作	FMADDの動作
正規化数, 0, ∞	－	FMUL終了時点でEIT発生 R0＝保持	EIT発生。R0＝保持
非正規化数	DN＝0	同上	UIPL発生　EIT発生 R0＝保持
	DN＝1	同上	EIT発生　R0＝保持
QNaN	－	同上	同上
SNaN	EV＝0	同上	IVLD発生　EIT発生 R0＝保持
	EV＝1	同上	同上

■ EO ＝ 0 での OVF 発生による精度異常発生時：

＜EX＝0時：IXCT発生＞

『(1)ステップ1でオーバフロー発生時』の『＜EO＝0, EX＝0時：OVFとIXCT発生＞』を参照してください。

＜EX＝1時：IXCT発生＞

『■丸めの発生による精度異常発生時』の『＜EX＝1時：IXCT発生＞』と同じです。

付録6.2　M32R-FPUにおけるQNaN生成規則

演算結果がQNaNとなる場合の生成規則を示します。なお演算結果としてNaNを生成する命令はFADD, FSUB, FMUL, FDIV, FMADD, FMSUBの6命令です。

【注意事項】

FCMP, FCMPEの「比較結果」(Rdestに入るデータ)がNaNのビットパターンを構成する場合については、本項の生成規則には従いません。

＜FADD, FSUB, FMUL, FDIVの場合＞

ソースオペランド(Rsrc1, Rsrc2)	Rdest
SNaNとQNaN	QNaN化されたSNaN(注1)
ともにSNaN	QNaN化されたRsrc2(注1)
ともにQNaN	Rsrc2
SNaNと実数	QNaN化されたSNaN(注1)
QNaNと実数	QNaN
どちらもNaNでないケースで、IVLD発生時	H'7FFF FFFF

注1．SNaNのb9に"1"をセットしてQNaN化したものです。

＜FMADD, FMSUBの場合＞

ソースオペランド		Rdest
Rdest	Rsrc1, Rsrc2	
実数	SNaNとQNaN	QNaN化されたSNaN(注1)
	ともにSNaN	QNaN化されたRsrc2(注1)
	ともにQNaN	Rsrc2
	SNaNと実数	QNaN化されたSNaN(注1)
	QNaNと実数	QNaN
	どちらもNaNでないケースで、IVLD発生時	H'7FFF FFFF
QNaN	Don't care	Rdest(保持)
SNaN	Don't care	QNaN化されたRdest(注1)

注1．SNaNのb9に"1"をセットしてQNaN化したものです。

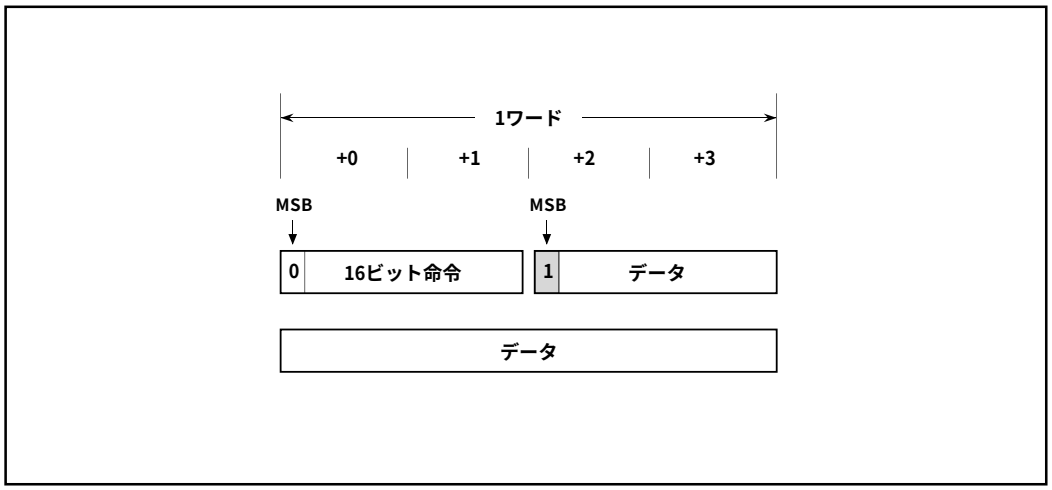
付録7　注意事項

■データ配置時の注意

プログラムのコード領域につづいてデータ領域の配置または確保をする場合は、ワードアライメント調整を行ったアドレスから配置してください。

ワードアライメント調整を行わずにデータ領域の配置または確保をした場合、1ワードの上位のハーフワードに16ビット命令が存在し、それにつづくハーフワードにMSBが"1"であるデータが配置されることがあります。このとき、M32Rファミリ上位互換のCPUでは、16ビット命令とデータを並列実行可能な命令のペアであると認識し並列実行処理を行います。

ソフトウエアの上位互換性を考慮して、上位のハーフワードに16ビット命令が存在する場合、それにつづくデータ領域はワードアライメント調整を行ったアドレスから配置または確保するように、プログラミング時にはご注意ください。



索引

索引

記号

A

B

C

D

E

F

I

J

L

M

N

O

P

R

索引

索引

メ

ユ

レ

ロ

ワ

ルネサス32ビットRISCシングルチップマイクロコンピュータ
ソフトウエアマニュアル
M32R-FPU

発行年月日　2002年7月12日　Rev. 1.00
　　2003年10月31日　Rev. 1.01

発行　　株式会社 ルネサス テクノロジ　営業企画統括部
　　〒100-0004　東京都千代田区大手町2-6-2

M32R-FPU
ソフトウエアマニュアル

株式会社ルネサス テクノロジ
東京都千代田区大手町2-6-2 日本ビル 〒100-0004

M32R-FPU
ソフトウェアマニュアル

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668

RJJ09B0107-0101Z