

RXファミリ RXv3命令セットアーキテクチャ

ユーザーズマニュアル ソフトウェア編

ルネサス 32ビットマイクロコンピュータ
RXファミリ

本資料に記載の全ての情報は本資料発行時点のものであり、ルネサスエレクトロニクスは、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 エレクトロニクスのホームページなどにより公開される最新情報をご確認ください。

ご注意書き

1. 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・システムの設計において、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て生じた損害（お客様または第三者いずれに生じた損害も含まれます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 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。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

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、産業用ロボット等

高品質水準： 輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、
金融端末基幹システム、各種安全制御装置等

当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。

6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合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。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、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、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等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、お客様の機器・システムとしての出荷保証を行ってください。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、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の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。かかる法令を遵守しないことにより生じた損害に関し、当社は、一切その責任を負いません。
9. 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・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
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禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

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直接的、間接的に支配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。

このマニュアルの使い方

このマニュアルは、RXv3 命令セットアーキテクチャ (RXv3) をユーザに理解していただくためのマニュアルです。このマニュアルでは、CPU の特長や命令体系について詳細に記載しています。本 CPU を用いた応用システムを設計するユーザを対象にしています。このマニュアルを使用するには、マイクロコンピュータに関する基本的な知識が必要です。

略号について

本マニュアルで使用する略号について、以下に説明します。

分類	略号	意味
記号	IMM	即値 (Immediate) を表します。
	SIMM	処理サイズに合わせて、符号拡張される即値 (Signed) を表します。
	UIMM	処理サイズに合わせて、ゼロ拡張される即値 (Unsigned) を表します。
	src, src2	命令オペランドのソース (Source) を表します。
	dest	命令オペランドのデスティネーション (Destination) を表します。
	dsp	相対アドレッシングの変位 (Displacement) を表します。
	pcdsp	プログラムカウンタ相対アドレッシングの変位 (Displacement) を表します。
	[]	間接アドレッシングを表します。
	Rn	汎用レジスタを表します。特に断りがない場合は、R0～R15が指定できます。
	Rs	汎用レジスタ (Source) を表します。特に断りがない場合は、R0～R15が指定できます。
	Rs2	オペランドに2つの汎用レジスタ (Source) が指定可能な命令においては、第1汎用レジスタ (Source) をRs、第2汎用レジスタ (Source) をRs2と表記します。
	Rd	汎用レジスタ (Destination) を表します。特に断りがない場合は、R0～R15が指定できます。
	Rd2	主にPOPM、RTSD命令の説明に使用されます。これらの命令では、オペランドに2つの汎用レジスタ (Destination) が指定可能なため、第1汎用レジスタ (Destination) をRd、第2汎用レジスタ (Destination) をRd2と表記します。
	Rb	汎用レジスタ (Base) を表します。特に断りがない場合は、R0～R15が指定できます。
	Ri	汎用レジスタ (Index) を表します。特に断りがない場合は、R0～R15が指定できます。
	Rx	制御レジスタを表します。PC、ISP、USP、INTB、EXTB、PSW、BPC、BPSW、FINTV、FPSWが指定できます。PCはMVFC、PUSHC命令のsrcにのみ指定できます。
	flag	PSWのビット (U、I)、フラグ (O、S、Z、C) を表します。
	Adest	アキュムレータ (Destination) を表します。A0、A1が指定できます。
	Asrc	アキュムレータ (Source) を表します。A0、A1が指定できます。
	tmp, tmp0、tmp1、tmp2、tmp3など	一時レジスタを表します。
	slsb, dlsb, width	ビットフィールド転送命令での、ビットフィールド情報を表します。
	DRs, DRs2	倍精度浮動小数点データレジスタ (Source) を表します。DR0～DR15が指定できます。
	DRLs	倍精度浮動小数点データレジスタの下位32ビット (Source) を表します。DRL0～DRL15が指定できます。
	DRHs	倍精度浮動小数点データレジスタの上位32ビット (Source) を表します。DRH0～DRH15が指定できます。
	DRd, DRd2	倍精度浮動小数点データレジスタ (Destination) を表します。DR0～DR15が指定できます。
	DRLd	倍精度浮動小数点データレジスタの下位32ビット (Destination) を表します。DRL0～DRL15が指定できます。
	DRHd	倍精度浮動小数点データレジスタの上位32ビット (Destination) を表します。DRH0～DRH15が指定できます。
	DCRs, DCRs2	倍精度浮動小数点制御レジスタ (Source) を表します。DPSW、DCMR、DECNT、DEPCが指定できます。
	DCRd, DCRd2	倍精度浮動小数点制御レジスタ (Destination) を表します。DPSW、DCMR、DECNT、DEPCが指定できます。

分類	略号	意味
数値	000b	2進数を表します。
	0000h	16進数を表します。
ビット長	#IMM: <u>g</u> など	オペランド記号の有効ビット長を表します。
	: <u>1</u>	有効ビット長が1ビットであることを表します。
	: <u>2</u>	有効ビット長が2ビットであることを表します。
	: <u>3</u>	有効ビット長が3ビットであることを表します。
	: <u>4</u>	有効ビット長が4ビットであることを表します。
	: <u>5</u>	有効ビット長が5ビットであることを表します。
	: <u>8</u>	有効ビット長が8ビットであることを表します。
	: <u>16</u>	有効ビット長が16ビットであることを表します。
	: <u>24</u>	有効ビット長が24ビットであることを表します。
: <u>32</u>	有効ビット長が32ビットであることを表します。	
サイズ指定	MOV_ <u>W</u> など	命令の取り扱うサイズを指定する記号です。
	<u>B</u>	バイト（8ビット）を指定します。
	<u>W</u>	ワード（16ビット）を指定します。
	<u>L</u>	ロングワード（32ビット）を指定します。
	<u>D</u>	ダブルロングワード（64ビット）を指定します。
分岐距離指定	BRA_ <u>A</u> など	分岐の相対距離の有効ビット長を指定する記号です。
	<u>S</u>	3ビットのPC前方相対を表します。有効値は、3～10です。
	<u>B</u>	8ビットのPC相対を表します。有効値は、-128～127です。
	<u>W</u>	16ビットのPC相対を表します。有効値は、-32768～32767です。
	<u>A</u>	24ビットのPC相対を表します。 有効値は、-8388608～8388607です。
	<u>L</u>	32ビットのPC相対を表します。 有効値は、-2147483648～2147483647です。
メモリオペランドに付加されるサイズ拡張指定	dsp:16[Rs] <u>UB</u> など	メモリオペランドのサイズと拡張方法を指定する記号です。省略した場合、ロングワードとして扱います。
	<u>B</u>	バイト（8ビット）を指定します。拡張方法は符号拡張です。
	<u>UB</u>	バイト（8ビット）を指定します。拡張方法はゼロ拡張です。
	<u>W</u>	ワード（16ビット）を指定します。拡張方法は符号拡張です。
	<u>UW</u>	ワード（16ビット）を指定します。拡張方法はゼロ拡張です。
	<u>L</u>	ロングワード（32ビット）を指定します。

分類	略号	意味
オペレーション	(原則としてC言語の文法規則に則っています。以下に、本マニュアルで使用している表記について説明します。)	
	=	代入演算子です。右辺の値を左辺に代入します。
	-	単項演算子の負号、または二項演算子の「差」を表します。
	+	二項演算子の「和」を表します。
	*	ポインタ演算子、または二項演算子の「積」を表します。
	/	二項演算子の「商」を表します。
	%	二項演算子の「剰余」を表します。
	~	単項ビット演算子の「NOT」を表します。
	&	二項ビット演算子の「AND」を表します。
		二項ビット演算子の「OR」を表します。
	^	二項ビット演算子の「Exclusive OR」を表します。
	;	文の終了を表します。
	{ }	複文の開始と終了を表します。{ }内には複数の文が記述できます。
	if(式)文1 else 文2	if文を表します。式を評価して、真であれば文1を、偽であれば文2を実行します。
	for(文1;式;文2)文3	for文を表します。文1を実行した後、式を評価して、真であれば文3を実行します。文3の実行後は、文2を実行した後、式を評価します。
	do 文 while(式);	do文を表します。式が真の間、文を実行します。式の真偽にかかわらず、文は最低1回実行されます。
	while(式)文	while文を表します。式が真の間、文を実行します。
	==、!=	比較演算子です。順に「等しい」、「等しくない」を表します。
	>、<	比較演算子です。順に「大なり」、「小なり」を表します。
	>=、<=	比較演算子です。「>」、「<」に「==」の条件が加わります。
	&&	論理演算子です。左側の条件と、右側の条件の「AND」を表します。
	論理演算子です。左側の条件と、右側の条件の「OR」を表します。	
<<、>>	シフト演算子です。「左にシフト」、「右にシフト」を表します。	
!	論理演算子です。変数や式のブール値を反転します。	
浮動小数点数	NaN	非数 (Not a Number)
浮動小数点形式	SNaN	Signaling NaN
	QNaN	Quiet NaN

目次

このマニュアルの使い方	3
RX ファミリ RXv3 命令セット一覧	10
アルファベット順ページ早見表	10
1. CPU プログラミングモデル	14
1.1 特長	14
1.2 CPU レジスタセット	15
1.2.1 汎用レジスタ (R0 ~ R15)	16
1.2.2 制御レジスタ	16
1.2.2.1 割り込みスタックポインタ (ISP) / ユーザスタックポインタ (USP)	17
1.2.2.2 割り込みテーブルレジスタ (INTB)	17
1.2.2.3 プログラムカウンタ (PC)	17
1.2.2.4 プロセッサステータスワード (PSW)	18
1.2.2.5 バックアップ PC (BPC)	20
1.2.2.6 バックアップ PSW (BPSW)	20
1.2.2.7 高速割り込みベクタレジスタ (FINTV)	20
1.2.2.8 単精度浮動小数点ステータスワード (FPSW)	21
1.2.2.9 例外テーブルレジスタ (EXTB)	23
1.2.3 アキュムレータ	24
1.3 単精度浮動小数点例外	25
1.3.1 オーバフロー	25
1.3.2 アンダフロー	25
1.3.3 精度異常	26
1.3.4 ゼロ除算	26
1.3.5 無効演算	27
1.3.6 非実装処理	27
1.4 プロセッサモード	28
1.4.1 スーパーバイザモード	28
1.4.2 ユーザモード	28
1.4.3 特権命令	28
1.4.4 プロセッサモード間の移行	28
1.5 データタイプ	29
1.5.1 整数	29
1.5.2 単精度浮動小数点数	29
1.5.3 ビット	30
1.5.4 スtring	30
1.6 データ配置	31
1.6.1 レジスタのデータ配置	31
1.6.2 メモリ上のデータ配置	31
1.7 ベクタテーブル	32
1.7.1 例外ベクタテーブル	32
1.7.2 割り込みベクタテーブル	33

1.8	アドレス空間	34
1.9	レジスタ一括退避機能	35
1.10	倍精度浮動小数点コプロセッサ	36
1.10.1	特長	36
1.10.2	倍精度浮動小数点レジスタセット	36
1.10.2.1	倍精度浮動小数点データレジスタ (DR0 ~ DR15)	37
1.10.2.2	倍精度浮動小数点制御レジスタ	37
1.10.3	倍精度浮動小数点例外	42
1.10.3.1	オーバフロー	42
1.10.3.2	アンダフロー	42
1.10.3.3	精度異常	43
1.10.3.4	ゼロ除算	43
1.10.3.5	無効演算	44
1.10.3.6	非実装処理	44
1.10.4	データタイプ (倍精度浮動小数点コプロセッサ)	45
1.10.4.1	倍精度浮動小数点数	45
1.10.5	データ配置 (倍精度浮動小数点コプロセッサ)	46
1.10.5.1	倍精度浮動小数点レジスタのデータ配置	46
1.10.5.2	メモリ上の倍精度浮動小数点データ配置	46
2.	アドレッシングモード	47
2.1	本章の見方	48
2.2	アドレッシングモード	49
2.2.1	IMM の範囲	54
3.	命令	55
3.1	命令セット概要	55
	命令一覧表	55
3.2	RXv3 拡張命令一覧	60
3.2.1	RXv3 新規追加命令	60
3.2.2	RXv3 仕様拡張命令	61
3.3	倍精度浮動小数点処理命令一覧	62
3.4	本章の見方	63
3.5	命令詳細説明	69
3.5.1	標準搭載命令	69
3.5.2	レジスタ一括退避機能命令	212
3.5.3	倍精度浮動小数点処理命令	215
4.	命令コード	251
4.1	本章の見方	251
4.2	命令コード詳細説明	254
4.2.1	標準搭載命令	254
4.2.2	レジスタ一括退避機能命令	340
4.2.3	倍精度浮動小数点処理命令	342

5.	例外処理.....	356
5.1	例外事象.....	356
5.1.1	未定義命令例外.....	357
5.1.2	特権命令例外.....	357
5.1.3	アクセス例外.....	357
5.1.4	アドレス例外.....	357
5.1.5	単精度浮動小数点例外.....	357
5.1.6	リセット.....	357
5.1.7	ノンマスカブル割り込み.....	357
5.1.8	割り込み.....	357
5.1.9	無条件トラップ.....	357
5.2	例外の処理手順.....	358
5.3	例外事象の受け付け.....	360
5.3.1	受け付けタイミングと保存される PC 値.....	360
5.3.2	ベクタと PC、PSW の退避場所.....	360
5.4	例外の受け付け / 復帰時のハードウェア処理.....	361
5.5	ハードウェア前処理.....	362
5.5.1	未定義命令例外.....	362
5.5.2	特権命令例外.....	362
5.5.3	アクセス例外.....	362
5.5.4	アドレス例外.....	362
5.5.5	単精度浮動小数点例外.....	363
5.5.6	リセット.....	363
5.5.7	ノンマスカブル割り込み.....	363
5.5.8	割り込み.....	363
5.5.9	無条件トラップ.....	364
5.6	例外処理ルーチンからの復帰.....	365
5.7	例外事象の優先順位.....	365
5.8	コプロセッサの例外事象.....	366
5.8.1	倍精度浮動小数点例外.....	366
索引	367
改訂記録	370

RXファミリRXv3命令セット一覧

アルファベット順ページ早見表 (1/4)

ニーモニック		機能	命令詳細 記載ページ	命令コード 詳細記載ページ
ABS		絶対値	70	255
ADC		キャリ付き加算	71	256
ADD		キャリなし加算	72	257
AND		論理積	74	259
BCLR		ビットクリア	75	261
BCnd	BGEU	相対条件分岐	76	263
	BC		76	263
	BEQ		76	263
	BZ		76	263
	BGTU		76	263
	BPZ		76	263
	BGE		76	263
	BGT		76	263
	BO		76	263
	BLTU		76	263
	BNC		76	263
	BNE		76	263
	BNZ		76	263
	BLEU		76	263
	BN		76	263
	BLE		76	263
BLT	76	263		
BNO	76	263		
BFMOV		ビットフィールド転送	77	264
BFMOVZ		ゼロへのビットフィールド転送	78	264
BMnd	BMGEU	条件ビット転送	79	265
	BMC		79	265
	BMEQ		79	265
	BMZ		79	265
	BMGTU		79	265
	BMPZ		79	265
	BMGE		79	265
	BMGT		79	265
	BMO		79	265
	BMLTU		79	265
	BMNC		79	265
	BMNE		79	265
	BMNZ		79	265
	BMLEU		79	265
	BMN		79	265
	BMLE		79	265
BMLT	79	265		
BMNO	79	265		
BNOT		ビット反転	81	266
BRA		相対無条件分岐	82	267
BRK		無条件トラップ	83	268
BSET		ビットセット	84	268

アルファベット順ページ早見表 (2/4)

ニーモニック		機能	命令詳細 記載ページ	命令コード 詳細記載ページ
BSR		相対サブルーチン分岐	85	270
BTST		ビットテスト	86	271
CLRPSW		PSWのフラグ、ビットのクリア	87	272
CMP		比較	88	273
DABS		倍精度浮動小数点絶対値	216	343
DADD		倍精度浮動小数点加算	217	343
DCMPcm	DCMPUN	倍精度浮動小数点比較	219	344
	DCMPEQ		219	344
	DCMPLT		219	344
	DCMPLE		219	344
DDIV		倍精度浮動小数点除算	221	344
DIV		符号付き除算	89	274
DIVU		符号なし除算	90	276
DMOV		倍精度浮動小数点データ転送	223	345
DMUL		倍精度浮動小数点乗算	225	348
DNEG		倍精度浮動小数点符号反転	227	348
DPOPM		複数の倍精度浮動小数点レジスタの復帰	228	349
DPUSHM		複数の倍精度浮動小数点レジスタの退避	230	350
DROUND		倍精度浮動小数点数→符号付き整数変換	232	351
DSQRT		倍精度浮動小数点平方根	234	351
DSUB		倍精度浮動小数点減算	236	351
DTOF		倍精度浮動小数点数→単精度浮動小数点数変換	238	352
DTOI		倍精度浮動小数点数→符号付き整数変換	240	352
DTOU		倍精度浮動小数点数→符号なし整数変換	242	352
EMACA		32ビット積和演算	91	277
EMSBA		32ビット積差演算	92	277
EMUL		符号付き乗算	93	278
EMULA		32ビット乗算	95	279
EMULU		符号なし乗算	96	279
FADD		単精度浮動小数点加算	98	281
FCMP		単精度浮動小数点比較	101	282
FDIV		単精度浮動小数点除算	103	283
FMUL		単精度浮動小数点乗算	105	284
FSQRT		単精度浮動小数点平方根	108	285
FSUB		単精度浮動小数点減算	110	286
FTOD		単精度浮動小数点数→倍精度浮動小数点数変換	244	353
FTOI		単精度浮動小数点数→符号付き整数変換	113	287
FTOU		単精度浮動小数点数→符号なし整数変換	115	287
INT		ソフトウェア割り込み	117	288
ITOD		符号付き整数→倍精度浮動小数点数変換	246	353
ITOF		符号付き整数→単精度浮動小数点数変換	118	288
JMP		無条件分岐	120	289
JSR		サブルーチン分岐	121	289
MACHI		上位16ビット積和演算	122	290
MACLH		下位16ビット・上位16ビット積和演算	123	290
MACLO		下位16ビット積和演算	124	290
MAX		最大値選択	125	291
MIN		最小値選択	126	292
MOV		転送	127	293
MOVCO		LIフラグクリア付きストア	131	298

アルファベット順ページ早見表 (3 / 4)

ニーモニック	機能	命令詳細 記載ページ	命令コード 詳細記載ページ
MOVLI	LIフラグセット付きロード	132	298
MOVU	符号なしデータ転送	133	299
MSBHI	上位16ビット積差演算	134	300
MSBLH	下位16ビット・上位16ビット積差演算	135	300
MSBLO	下位16ビット積差演算	136	301
MUL	乗算	137	301
MULHI	上位16ビット乗算	139	303
MULLH	下位16ビット・上位16ビット乗算	140	303
MULLO	下位16ビット乗算	141	304
MVFACGU	アキュムレータガードビットからの転送	142	304
MVFACHI	アキュムレータ上位32ビットからの転送	143	305
MVFACLO	アキュムレータ下位32ビットからの転送	144	305
MVFACMI	アキュムレータ中央32ビットからの転送	145	306
MVFC	制御レジスタからの転送	146	306
MVFDC	倍精度浮動小数点制御レジスタからの転送	247	354
MVFDR	倍精度浮動小数点比較結果レジスタからの転送	248	354
MVTACGU	アキュムレータガードビットへの転送	147	307
MVTACHI	アキュムレータ上位32ビットへの転送	148	307
MVTACLO	アキュムレータ下位32ビットへの転送	149	307
MVTC	制御レジスタへの転送	150	308
MVTD	倍精度浮動小数点制御レジスタへの転送	249	355
MVTIPL (特権命令)	割り込み優先レベル設定	151	309
NEG	符号反転	152	310
NOP	ノーオペレーション	153	310
NOT	論理反転	154	311
OR	論理和	155	312
POP	スタックからレジスタへのデータ復帰	156	313
POPC	制御レジスタの復帰	157	314
POPM	複数レジスタの復帰	158	314
PUSH	スタックへデータ退避	159	315
PUSHC	制御レジスタの退避	160	316
PUSHM	複数レジスタの退避	161	316
RACL	符号付きアキュムレータ丸め処理	162	317
RACW	16ビット符号付きアキュムレータ丸め処理	164	317
RDACL	符号付きアキュムレータ丸め処理	166	318
RDACW	16ビット符号付きアキュムレータ丸め処理	168	318
REVL	エンディアン変換	170	319
REWV	エンディアン変換	171	319
RMPA	積和演算	172	320
ROL	キャリ付き左回転	174	320
ROR	キャリ付き右回転	175	320
ROTL	左回転	176	321
ROTR	右回転	177	321
ROUND	単精度浮動小数点数→符号付き整数変換	178	322
RSTR (特権命令)	レジスタの一括復帰	213	341
RTE (特権命令)	例外からの復帰	181	322
RTFI (特権命令)	高速割り込みからの復帰	182	322
RTS	サブルーチンからの復帰	183	323
RTSD	スタックフレームの解放とサブルーチンからの復帰	184	323

アルファベット順ページ早見表 (4 / 4)

ニーモニック		機能	命令詳細 記載ページ	命令コード 詳細記載ページ
SAT		32ビット符号付き飽和处理	186	323
SATR		RMPA命令用64ビット符号付き飽和处理	187	324
SAVE (特権命令)		レジスタの一括退避	214	341
SBB		ポロー付き減算	188	324
SC <i>Cnd</i>	SCGEU	条件設定	189	325
	SCC		189	325
	SCEQ		189	325
	SCZ		189	325
	SCGTU		189	325
	SCPZ		189	325
	SCGE		189	325
	SCGT		189	325
	SCO		189	325
	SCLTU		189	325
	SCNC		189	325
	SCNE		189	325
	SCNZ		189	325
	SCLEU		189	325
	SCN		189	325
	SCLE		189	325
SCLT	189	325		
SCNO	189	325		
SCMPU		ストリング比較	190	325
SETPSW		PSWのフラグ、ビットのセット	191	326
SHAR		算術右シフト	192	327
SHLL		論理/算術左シフト	193	328
SHLR		論理右シフト	194	329
SMOVB		逆方向ストリング転送	195	329
SMOVF		順方向ストリング転送	196	330
SMOVU		ストリング転送	197	330
SSTR		ストリングストア	198	330
STNZ		条件付き転送	199	331
STZ		条件付き転送	200	332
SUB		ポローなし減算	201	333
SUNTIL		ストリングサーチ	202	334
SWHILE		ストリングサーチ	204	334
TST		テスト	206	335
UTOD		符号なし整数→倍精度浮動小数点数変換	250	355
UTOF		符号なし整数→単精度浮動小数点数変換	207	336
WAIT (特権命令)		ウェイト	209	337
XCHG		交換	210	337
XOR		排他的論理和	211	338

1. CPU プログラミングモデル

RXv3 命令セットアーキテクチャ (RXv3) は、RXv2 命令セットアーキテクチャ (RXv2) と上位互換性のある命令セットアーキテクチャです。

- 可変長命令方式の採用
可変長命令形式の採用により、使用頻度の高い命令をより短い命令長に割り付けていますので、コード効率の良いプログラムを開発できます。
- 強力な命令セット
RXv3 は厳選された 113 個の命令をサポートしています。DSP 機能命令や浮動小数点演算命令により、DSP に匹敵するデータ処理能力を発揮します。
- 豊富なアドレッシングモード
11 種類の豊富なアドレッシングモードを持ち、レジスタ-レジスタ間、レジスタ-メモリ間の演算や、ビットを対象とする演算ができます。また、メモリ-メモリ間の転送ができます。

1.1 特長

- 最短命令実行時間：1 サイクルで実行
- アドレス空間：4G バイト・リニアアドレス
- CPU レジスタセット
汎用レジスタ：32 ビット×16 本
制御レジスタ：32 ビット×10 本
アキュムレータ：72 ビット×2 本
- 可変長命令形式（1 バイト長～8 バイト長）
- 113 命令 / 11 種類アドレッシングモード
標準搭載命令：111 命令
基本命令：77 命令
単精度浮動小数点演算命令：11 命令
DSP 機能命令：23 命令
レジスタ括退避機能命令：2 命令（オプション）
- プロセッサモード
スーパーバイザモード、ユーザモード
- ベクタテーブル
例外ベクタテーブル、割り込みベクタテーブル
- メモリプロテクションユニット（オプション機能）
- データ配置
リトルエンディアン / ビッグエンディアン選択可能
- 倍精度浮動小数点コプロセッサ搭載（オプション機能）
倍精度浮動小数点処理命令：21 命令

1.2 CPUレジスタセット

RXv3 CPU のレジスタには、汎用レジスタ（16本）と、制御レジスタ（10本）、およびDSP機能命令で使用するアキュムレータ（2本）があります。



図 1.1 CPUレジスタセット

1.2.1 汎用レジスタ (R0 ~ R15)

汎用レジスタは、32ビット幅で16本 (R0 ~ R15) あります。汎用レジスタ R0 ~ R15 は、データレジスタやアドレスレジスタとして使用します。

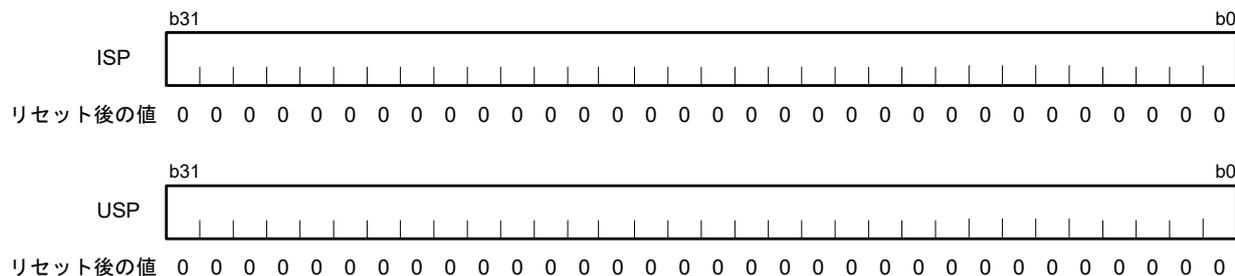
汎用レジスタ R0 には、汎用レジスタとしての機能に加えて、スタックポインタ (SP) としての機能が割り当てられています。SP は、プロセッサステータスワード (PSW) のスタックポインタ指定ビット (U) によって、割り込みスタックポインタ (ISP)、またはユーザスタックポインタ (USP) に切り替わります。

1.2.2 制御レジスタ

制御レジスタには、以下の10本のレジスタがあります。

- 割り込みスタックポインタ (ISP)
- ユーザスタックポインタ (USP)
- 割り込みテーブルレジスタ (INTB)
- プログラムカウンタ (PC)
- プロセッサステータスワード (PSW)
- バックアップ PC (BPC)
- バックアップ PSW (BPSW)
- 高速割り込みベクタレジスタ (FINTV)
- 単精度浮動小数点ステータスワード (FPSW)
- 例外テーブルレジスタ (EXTB)

1.2.2.1 割り込みスタックポインタ (ISP) / ユーザスタックポインタ (USP)



スタックポインタ (SP) には、割り込みスタックポインタ (ISP) と、ユーザスタックポインタ (USP) の2種類があります。使用するスタックポインタ (ISP/USP) は、プロセッサステータスワード (PSW) のスタックポインタ指定ビット (U) によって切り替えられます。

1.2.2.2 割り込みテーブルレジスタ (INTB)



割り込みテーブルレジスタ (INTB) には、割り込みベクタテーブルの先頭番地を設定してください。

1.2.2.3 プログラムカウンタ (PC)



プログラムカウンタ (PC) は、実行中の命令の番地を示します。

1.2.2.4 プロセッサステータスワード (PSW)

	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—		IPL[3:0]			—	—	—	PM	—	—	U	I
リセット後の値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	O	S	Z	C
リセット後の値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ビット	シンボル	ビット名	機能	R/W
b0	C	キャリフラグ	0: キャリの発生なし 1: キャリの発生あり	R/W
b1	Z	ゼロフラグ	0: 演算結果は0でなかった 1: 演算結果は0であった	R/W
b2	S	サインフラグ	0: 演算結果は正または0であった 1: 演算結果は負であった	R/W
b3	O	オーバフローフラグ	0: オーバフローの発生なし 1: オーバフローの発生あり	R/W
b15-b4	—	予約ビット	書く場合、“0”としてください。 読むと“0”が読めます。	R/W
b16	I (注1)	割り込み許可ビット	0: 割り込み禁止 1: 割り込み許可	R/W
b17	U (注1)	スタックポインタ指定ビット	0: 割り込みスタックポインタ (ISP) を指定 1: ユーザスタックポインタ (USP) を指定	R/W
b19-b18	—	予約ビット	書く場合、“0”としてください。 読むと“0”が読めます。	R/W
b20	PM (注1、2、3)	プロセッサモード設定ビット	0: スーパーバイザモードに設定 1: ユーザモードに設定	R/W
b23-b21	—	予約ビット	書く場合、“0”としてください。 読むと“0”が読めます。	R/W
b27-b24	IPL[3:0] (注1)	プロセッサ割り込み優先レベル	b27 b24 0 0 0 0: 優先レベル0 (最低) 0 0 0 1: 優先レベル1 0 0 1 0: 優先レベル2 0 0 1 1: 優先レベル3 0 1 0 0: 優先レベル4 0 1 0 1: 優先レベル5 0 1 1 0: 優先レベル6 0 1 1 1: 優先レベル7 1 0 0 0: 優先レベル8 1 0 0 1: 優先レベル9 1 0 1 0: 優先レベル10 1 0 1 1: 優先レベル11 1 1 0 0: 優先レベル12 1 1 0 1: 優先レベル13 1 1 1 0: 優先レベル14 1 1 1 1: 優先レベル15 (最高)	R/W
b31-b28	—	予約ビット	書く場合、“0”としてください。 読むと“0”が読めます。	R/W

- 注1. ユーザモードのときは、MVTC、POPC 命令による IPL[3:0]、PM、U、I ビットへの書き込みは無視されます。また、MVTIPL 命令で IPL[3:0] ビットへの書き込みを行おうとした場合は、特権命令例外が発生します。
- 注2. スーパーバイザモードのときは、MVTC、POPC 命令による PM ビットへの書き込みは無視されます。それ以外のビットへの書き込みはできません。
- 注3. スーパーバイザモードからユーザモードに切り替える場合は、スタック上の PSW の PM ビットを“1”にした後、RTE 命令を実行するか、バックアップ PSW (BPSW) の PM ビットを“1”にした後、RTFI 命令を実行してください。

プロセッサステータスワード (PSW) は、命令実行の結果や、CPU の状態を示します。

C フラグ (キャリフラグ)

キャリー、ボロー、シフトアウトしたビット等を保持します。

Z フラグ (ゼロフラグ)

演算の結果が 0 のとき “1” になり、それ以外るとき “0” になります。

S フラグ (サインフラグ)

演算の結果が負のとき “1” になり、それ以外るとき “0” になります。

O フラグ (オーバフローフラグ)

演算の結果がオーバフローしたとき “1” になり、それ以外るとき “0” になります。

I ビット (割り込み許可ビット)

割り込み要求の受け付けを許可するビットです。例外を受け付けると、このビットは “0” になります。

U ビット (スタックポインタ指定ビット)

使用するスタックポインタ (ISP/USP) を指定するビットです。例外を受け付けると、このビットは “0” になります。スーパーバイザモードからユーザモードに移行すると、このビットは “1” になります。

PM ビット (プロセッサモード設定ビット)

プロセッサの動作モードを設定するビットです。例外を受け付けると、このビットは “0” になります。

IPL[3:0] ビット (プロセッサ割り込み優先レベル)

IPL[3:0] ビットは、優先レベル 0 (最低) ~ 優先レベル 15 (最高) までの 16 段階のプロセッサ割り込み優先レベルを指定します。要求があった割り込みの優先レベルが、プロセッサ割り込み優先レベルより高い場合、その割り込みが許可されます。IPL[3:0] ビットをレベル 15 (Fh) に設定したとき、すべての割り込みが禁止されます。IPL[3:0] ビットは、ノンマスカブル割り込みが発生したとき、レベル 15 (Fh) になります。割り込みが発生したとき、受け付けた割り込みの優先レベルになります。

1.2.2.5 バックアップ PC (BPC)



リセット後の値 不定

バックアップ PC (BPC) は、割り込み応答を高速化するために設けられたレジスタです。高速割り込みが発生すると、プログラムカウンタ (PC) の内容が BPC に退避させられます。

1.2.2.6 バックアップ PSW (BPSW)



リセット後の値 不定

バックアップ PSW (BPSW) は、割り込み応答を高速化するために設けられたレジスタです。高速割り込みが発生すると、プロセッサステータスワード (PSW) の内容が BPSW に退避させられます。BPSW のビットの割り当ては、PSW に対応しています。

1.2.2.7 高速割り込みベクタレジスタ (FINTV)



リセット後の値 不定

高速割り込みベクタレジスタ (FINTV) は、割り込み応答を高速化するために設けられたレジスタです。高速割り込み発生時の分岐先番地を設定してください。

1.2.2.8 単精度浮動小数点ステータスワード (FPSW)

	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	FS	FX	FU	FZ	FO	FV	—	—	—	—	—	—	—	—	—	—
リセット後の値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	EX	EU	EZ	EO	EV	—	DN	CE	CX	CU	CZ	CO	CV	RM[1:0]	—
リセット後の値	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

ビット	シンボル	ビット名	機能	R/W
b1-b0	RM[1:0]	単精度浮動小数点丸めモード設定ビット	b1 b0 0 0 : 最近値への丸め 0 1 : 0方向への丸め 1 0 : +∞方向への丸め 1 1 : -∞方向への丸め	R/W
b2	CV	無効演算要因フラグ	0 : 無効演算の発生なし 1 : 無効演算の発生あり	R/(W) (注1)
b3	CO	オーバフロー要因フラグ	0 : オーバフローの発生なし 1 : オーバフローの発生あり	R/(W) (注1)
b4	CZ	ゼロ除算要因フラグ	0 : ゼロ除算の発生なし 1 : ゼロ除算の発生あり	R/(W) (注1)
b5	CU	アンダフロー要因フラグ	0 : アンダフローの発生なし 1 : アンダフローの発生あり	R/(W) (注1)
b6	CX	精度異常要因フラグ	0 : 精度異常の発生なし 1 : 精度異常の発生あり	R/(W) (注1)
b7	CE	非実装処理要因フラグ	0 : 非実装処理の発生なし 1 : 非実装処理の発生あり	R/(W) (注1)
b8	DN	非正規化数の0フラッシュビット	0 : 非正規化数を非正規化数として扱う 1 : 非正規化数を0として扱う (注2)	R/W
b9	—	予約ビット	書く場合、“0”としてください。 読むと“0”が読めます。	R/W
b10	EV	無効演算例外処理許可ビット	0 : 無効演算発生による例外処理を禁止 1 : 無効演算発生による例外処理を許可	R/W
b11	EO	オーバフロー例外処理許可ビット	0 : オーバフロー発生による例外処理を禁止 1 : オーバフロー発生による例外処理を許可	R/W
b12	EZ	ゼロ除算例外処理許可ビット	0 : ゼロ除算発生による例外処理を禁止 1 : ゼロ除算発生による例外処理を許可	R/W
b13	EU	アンダフロー例外処理許可ビット	0 : アンダフロー発生による例外処理を禁止 1 : アンダフロー発生による例外処理を許可	R/W
b14	EX	精度異常例外処理許可ビット	0 : 精度異常発生による例外処理を禁止 1 : 精度異常発生による例外処理を許可	R/W
b25-b15	—	予約ビット	書く場合、“0”としてください。 読むと“0”が読めます。	R/W
b26	FV (注3)	無効演算フラグ	0 : 無効演算の発生なし 1 : 無効演算の発生あり (注8)	R/W
b27	FO (注4)	オーバフローフラグ	0 : オーバフローの発生なし 1 : オーバフローの発生あり (注8)	R/W
b28	FZ (注5)	ゼロ除算フラグ	0 : ゼロ除算の発生なし 1 : ゼロ除算の発生あり (注8)	R/W
b29	FU (注6)	アンダフローフラグ	0 : アンダフローの発生なし 1 : アンダフローの発生あり (注8)	R/W

ビット	シンボル	ビット名	機能	R/W
b30	FX (注7)	精度異常フラグ	0: 精度異常の発生なし 1: 精度異常の発生あり (注8)	R/W
b31	FS	単精度浮動小数点エラーサマリフラグ	FU、FZ、FO、FVフラグの論理和を反映	R

- 注1. “0”を書いた場合、“0”になります。“1”を書いた場合、前の値を保持します。
- 注2. 正の非正規化数は+0、負の非正規化数は-0として扱います。
- 注3. EVビットが“0”のときに、FVフラグは有効となります。
- 注4. EOビットが“0”のときに、FOフラグは有効となります。
- 注5. EZビットが“0”のときに、FZフラグは有効となります。
- 注6. EUビットが“0”のときに、FUフラグは有効となります。
- 注7. EXビットが“0”のときに、FXフラグは有効となります。
- 注8. 当該ビットが一度“1”になると、ソフトウェアで“0”にするまで“1”を保持します。

単精度浮動小数点ステータスワード (FPSW) は、単精度浮動小数点演算結果を示します。

例外処理許可ビット (Ej) で例外処理を許可 (Ej = 1) した場合は、例外処理ルーチンで該当する Cj フラグをチェックし、例外発生の要因を判断することができます。例外処理を禁止 (Ej = 0) した場合は、一連の処理の最後に Fj フラグをチェックし、例外発生の有無を確認することができます。Fj フラグは蓄積フラグです (j = X, U, Z, O, V)。

RM[1:0] ビット (単精度浮動小数点丸めモード設定ビット)

単精度浮動小数点丸めモードを設定します。

【単精度浮動小数点丸めモードの説明】

- 最近値への丸め (デフォルト) : 無限の有効桁を持つと仮定して計算した結果に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
 - 0 方向への丸め : 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)
 - +∞ 方向への丸め : 結果の値が大きくなる方向へ丸める
 - ∞ 方向への丸め : 結果の値が小さくなる方向へ丸める
- (1) 「最近値への丸め」はデフォルトのモードであり、最も正確な値を返します。
- (2) 「0 方向への丸め」、「+∞ 方向への丸め」、「-∞ 方向への丸め」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。

CV フラグ (無効演算要因フラグ)、CO フラグ (オーバフロー要因フラグ)、 CZ フラグ (ゼロ除算要因フラグ)、CU フラグ (アンダフロー要因フラグ)、 CX フラグ (精度異常要因フラグ)、CE フラグ (非実装処理要因フラグ)

IEEE754 規格で規定された 5 つの例外 (オーバフロー、アンダフロー、精度異常、ゼロ除算、無効演算) の他に、非実装処理が発生した場合に該当するフラグが“1”になります。

- 単精度浮動小数点演算命令実行時にこれらの例外・非実装処理が発生しなかった場合は該当するフラグが“0”になります。
- MVTC、POPC 命令で“0”を書いた場合、“0”になります。“1”を書いた場合、前の値を保持します。

DN ビット (非正規化数の 0 フラッシュビット)

“0”のとき非正規化数を非正規化数として扱います。

“1”のとき非正規化数を 0 として扱います。

EV ビット (無効演算例外処理許可ビット)、EO ビット (オーバフロー例外処理許可ビット)、EZ ビット (ゼロ除算例外処理許可ビット)、EU ビット (アンダフロー例外処理許可ビット)、EX ビット (精度異常例外処理許可ビット)

単精度浮動小数点演算命令実行により、IEEE754 規格で規定された 5 つの例外が発生したときに、CPU が例外処理に移行するかどうかを制御します。“0” の場合、例外処理は禁止されます。“1” の場合、例外処理が許可されます。

FV フラグ (無効演算フラグ)、FO フラグ (オーバフローフラグ)、FZ フラグ (ゼロ除算フラグ)、FU フラグ (アンダフローフラグ)、FX フラグ (精度異常フラグ)

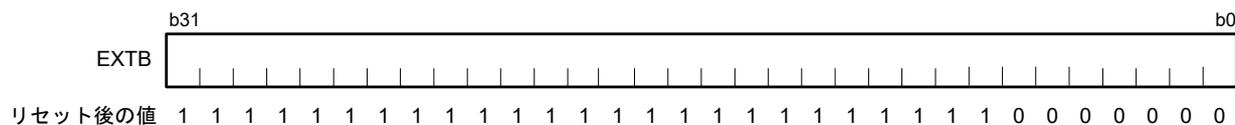
例外処理許可ビット E_j が “0” (例外処理を禁止) の場合、IEEE754 規格で規定された 5 つの例外が発生すると、該当するフラグが “1” になります。

- $E_j = 1$ (例外処理を許可) のときは、このフラグは動きません。
- 当該フラグが “1” になると、ソフトウェアで “0” にするまで “1” を保持します (蓄積フラグ)。

FS フラグ (単精度浮動小数点エラーサマリフラグ)

FU、FZ、FO、FV フラグの論理和を反映します。

1.2.2.9 例外テーブルレジスタ (EXTB)



例外テーブルレジスタ (EXTB) には、例外ベクタテーブルの先頭番地を設定してください。

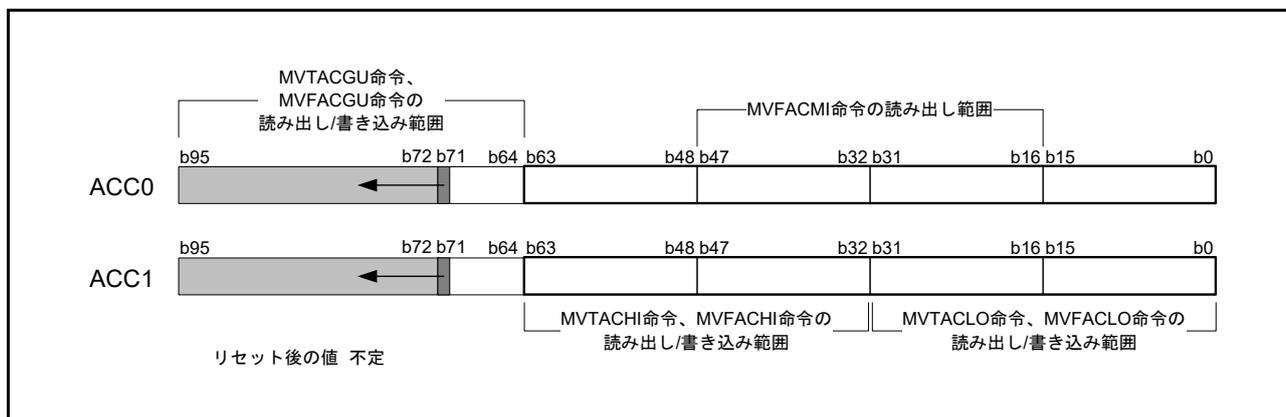
1.2.3 アキュムレータ

アキュムレータ (ACC0, ACC1) は、72 ビットのレジスタです。DSP 機能命令で使用されます。アキュムレータは、読み出し時や書き込み時は 96 ビットのレジスタとして扱われます。このとき、アキュムレータの b95 ~ b72 の扱いは、読み出し時に b71 の値を符号拡張し、書き込み時には無視します。また、ACC0 は乗算命令 (EMUL, EMULU, FMUL, MUL)、積和演算命令 (RMPA) でも使用され、これらの命令実行の際は ACC0 の値が変更されます。

ACC0、ACC1 への書き込みには、MVTACGU 命令、MVTACHI 命令と MVTACLO 命令を使用します。MVTACGU 命令は (b95 ~ b64) に、MVTACHI 命令は上位側 32 ビット (b63 ~ b32) に、MVTACLO 命令は下位側 32 ビット (b31 ~ b0) にデータを転送します。

読み出しには、MVFACGU 命令、MVFACHI 命令、MVFACMI 命令と MVTACLO 命令とを使用します。

MVFACGU 命令でガードビット (b95 ~ b64)、MVFACHI 命令で上位側 32 ビット (b63 ~ b32)、MVFACMI 命令で中央の 32 ビット (b47 ~ b16)、MVTACLO 命令で下位側 32 ビット (b31 ~ b0) のデータをそれぞれ読み出します。



注. b95 ~ b72 は、b71 の値を符号拡張した値が読み出されます。この部分への書き込みは無視されます。

1.3 単精度浮動小数点例外

単精度浮動小数点例外は、単精度浮動小数点演算命令実行時に、IEEE754規格で規定された5つの例外（オーバーフロー、アンダフロー、精度異常、ゼロ除算、無効演算）および、非実装処理を検出した場合に発生します。5つの例外については、対応するFPSWのEX、EU、EZ、EO、EVビットが“1”のときのみCPUの例外処理が行われます。

以下に、単精度浮動小数点例外を発生させる事象の概要を示します。

1.3.1 オーバフロー

オーバーフローは、演算結果の絶対値が単精度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。表1.1にオーバーフロー発生時の演算結果を示します。

表1.1 オーバフロー発生時の演算結果

単精度浮動小数点丸めモード	結果の符号	演算結果（デスティネーションレジスタの値）	
		EO = 0	EO = 1
-∞方向への丸め	+	+MAX	変化なし
	-	-∞	
+∞方向への丸め	+	+∞	
	-	-MAX	
0方向への丸め	+	+MAX	
	-	-MAX	
最近値への丸め	+	+∞	
	-	-∞	

注. EO = 0のとき、オーバーフローが発生すると精度異常が発生します。

1.3.2 アンダフロー

アンダフローは、演算結果の絶対値が単精度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（ただし、0を除く）に発生します。表1.2にアンダフロー発生時の演算結果を示します。

表1.2 アンダフロー発生時の演算結果

演算結果（デスティネーションレジスタの内容）	
EU = 0	EU = 1
DN = 0のとき、変化しない（非実装処理が発生）	変化なし
DN = 1のとき、“0”を返す	

1.3.3 精度異常

精度異常は、無限の有効桁を持つと仮定して計算した結果と、演算結果が異なっていた場合に発生します。表 1.3 に精度異常発生条件と演算結果を示します。

表 1.3 精度異常発生条件と演算結果

発生条件	演算結果（デスティネーションレジスタの内容）	
	EX = 0	EX = 1
オーバーフロー例外禁止状態でのオーバーフロー発生	「表 1.1 オーバーフロー発生時の演算結果」参照	変化なし
丸めの発生	丸め後の値	

注1. アンダフロー発生時、精度異常は発生しません。

注2. オーバーフロー例外許可状態でのオーバーフロー発生時、丸めの発生にかかわらず、精度異常は発生しません。

1.3.4 ゼロ除算

ゼロ除算は、0 でない有限数を 0 で割った場合に発生します。表 1.4 にゼロ除算発生時の演算結果を示します。ただし、被除数が表 1.5 に示す値の場合、ゼロ除算は発生しません。

表 1.4 ゼロ除算発生時の演算結果

被除数	演算結果（デスティネーションレジスタの内容）	
	EZ = 0	EZ = 1
0でない有限数	$\pm\infty$ （符号は除数、被除数の符号の排他的論理和となる）	変化なし

表 1.5 ゼロ除算が発生しないときの動作

被除数	動作
0	無効演算発生
∞	例外は発生しない。結果は ∞
非正規化数 (DN = 0)	非実装処理発生
QNaN	例外は発生しない。結果はQNaN
SNaN	無効演算発生

1.3.5 無効演算

無効演算は、無効な演算が実行された場合に発生します。表 1.6 に無効演算発生条件と演算結果を示します。

表 1.6 無効演算発生条件と演算結果

発生条件	演算結果（デスティネーションレジスタの内容）	
	EV = 0	EV = 1
SNaNオペランドに対する演算	QNaN	変化なし
$+\infty + (-\infty)$ 、 $+\infty - (+\infty)$ 、 $-\infty - (-\infty)$		
$0 \times \infty$		
$0 \div 0$ 、 $\infty \div \infty$		
0より小さい数に対する平方根演算		
FTOI命令、ROUND命令実行時、整数変換がオーバーフローするか、NaN、 ∞ を整数変換したとき	変換前の符号ビットが、“0”のときは7FFFFFFFh、“1”のときは80000000hを返す	
FTOU命令実行時、整数変換がオーバーフローするか、NaN、 ∞ を整数変換したとき	変換前の符号ビットが、“0”のときはFFFFFFFh、“1”のときは00000000hを返す	
SNaNオペランドに対する比較	デスティネーションはなし	

- ・ NaN (Not a Number) : 非数
- ・ SNaN (Signaling NaN) : 仮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、無効演算が発生します。変数の初期値として使用することにより、プログラムのバグの発見に役立ちます。なお、SNaNをハードウェアが生成することはありません。
- ・ QNaN (Quiet NaN) : 仮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、例外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ェアが生成します。

表 1.7 に演算結果が QNaN となる場合の生成規則を示します。

表 1.7 QNaN生成規則

ソースオペランド	演算結果（デスティネーションレジスタの内容）
SNaNとQNaN	QNaN化されたSNaNソースオペランド
ともにSNaN	QNaN化されたdest
ともにQNaN	dest
SNaNと実数	QNaN化されたSNaNソースオペランド
QNaNと実数	QNaNソースオペランド
どちらもNaNでないケースで無効演算発生時	7FFFFFFFh

注. SNaNのQNaN化は、仮数部の最上位ビットを“1”にして行います。

1.3.6 非実装処理

非実装処理は、DN = 0 で非正規化数が演算オペランドとして与えられるか、DN = 0 で演算の結果、アンダフローが発生した場合に発生します。DN = 1 では非実装処理は発生しません。

非実装処理発生による例外処理を禁止することはできません（非実装処理発生による例外処理を禁止する例外処理許可ビットはありません）。デスティネーションレジスタは変化しません。

1.4 プロセッサモード

RXv3 CPUには、スーパーバイザモード、およびユーザモードの2つのプロセッサモードがあります。これらのプロセッサモードとメモリプロテクション機能を使用して、CPUリソースやメモリに対する階層的な保護機構を実現することができます。各プロセッサモードには、メモリアクセスや実行可能な命令に対する権限を規定しており、スーパーバイザモードはユーザモードより高い権限を持っています。リセット後は、スーパーバイザモードで動作します。

1.4.1 スーパーバイザモード

スーパーバイザモードでは、すべてのCPUリソースにアクセスすることができ、また、すべての命令を実行することができます。ただし、MVTC、POPC命令によるプロセッサステータスワード (PSW) のプロセッサモード設定ビット (PM) への書き込みは無視されます。PMビットへの書き込み方法については、「1.2.2.4 プロセッサステータスワード (PSW)」を参照してください。

1.4.2 ユーザモード

ユーザモードでは、一部のCPUリソースへのライトアクセスが制限されます。ライトアクセスが制限されるCPUリソースは以下のとおりです。この制限はすべての命令からのアクセスが対象になります。

- プロセッサステータスワード (PSW) の一部のビット (IPL[3:0], PM, U, I)
- 割り込みスタックポインタ (ISP)
- 割り込みテーブルレジスタ (INTB)
- バックアップ PSW (BPSW)
- バックアップ PC (BPC)
- 高速割り込みベクタレジスタ (FINTV)
- 例外テーブルレジスタ (EXTB)

1.4.3 特権命令

特権命令は、スーパーバイザモードでのみ実行可能な命令です。ユーザモードで特権命令を実行すると、特権命令例外が発生します。特権命令には、RTFI、MVTIPL、RTE、WAIT、SAVE、RSTR命令があります。

1.4.4 プロセッサモード間の移行

プロセッサモードは、プロセッサステータスワード (PSW) のプロセッサモード設定ビット (PM) によって切り替えられます。ただし、MVTC、POPC命令によるPMビットの書き換えは無効です。以下に示す方法で切り替えてください。

(1) ユーザモードからスーパーバイザモードへの移行

例外が発生するとPSWのPMビットが“0”になり、CPUはスーパーバイザモードへ移行します。ハードウェア前処理は、スーパーバイザモードで実行されます。例外が発生する直前のプロセッサモードは、退避させられたPSWのPMビットに保持されます。

(2) スーパーバイザモードからユーザモードへの移行

スタック上に退避させられているPSWのPMビットを“1”にした後RTE命令を実行する、あるいはバックアップPSW (BPSW) に退避させられているPSWのPMビットを“1”にした後RTFI命令を実行することにより、ユーザモードへ移行します。ユーザモードへ移行すると、PSWのスタックポインタ指定ビット (U) が“1”になります。

1.5 データタイプ

RXv3 CPU は、整数、単精度浮動小数点数、ビット、ストリングの4種類のデータを扱うことができます。

1.5.1 整数

整数には、符号付きと、符号なしがあります。符号付き整数の負の値は、2の補数で表現します。

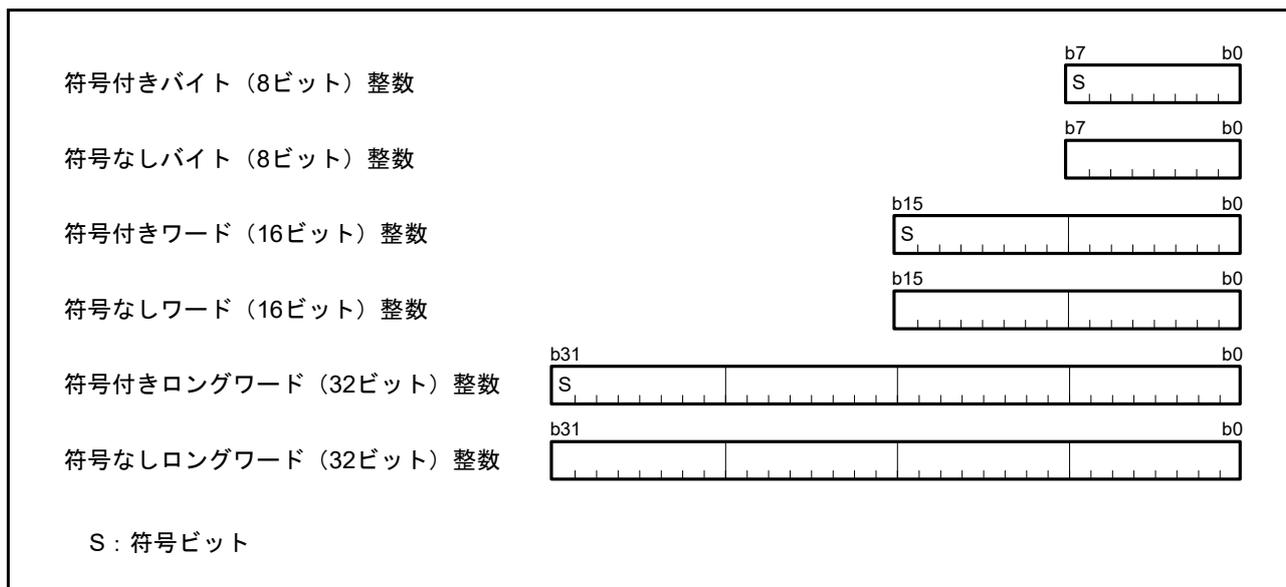


図 1.2 整数

1.5.2 単精度浮動小数点数

単精度浮動小数点数は、IEEE754 規格で規定されている単精度浮動小数点数に対応しています。単精度浮動小数点数は、単精度浮動小数点演算命令 FADD、FCMP、FDIV、FMUL、FSUB、FTOI、ITOF、ROUND、FTOU、UTOF、FSQRT の 11 種類の命令で使用できます。

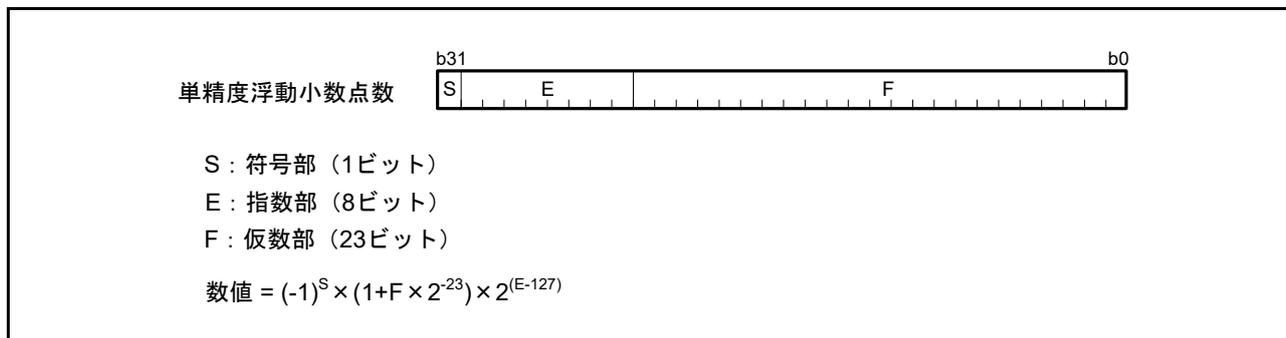


図 1.3 単精度浮動小数点数

単精度浮動小数点数は、以下の数値に対応しています。

- $0 < E < 255$ (正規化数 - Normal Numbers)
- $E = 0$ かつ $F = 0$ (ゼロ - Signed Zero)
- $E = 0$ かつ $F > 0$ (非正規化数 - Subnormal Numbers) (注)

- E = 255 かつ F = 0 (無限大 - Infinity)
- E = 255 かつ F > 0 (非数 - NaN : Not a Number)

注. FPSWのDNビットが“1”のときは、0として扱います。DNビットが“0”のときは、非実装処理が発生します。

1.5.3 ビット

ビットは、ビット操作命令 BCLR、BMCnd、BNOT、BSET、BTST の5種類の命令で使用できます。

レジスタのビットは、対象とするレジスタと、31～0のビット番号で指定します。

メモリのビットは、対象とするアドレスと、7～0のビット番号で指定します。アドレス指定に使用できるアドレッシングモードは、レジスタ間接、レジスタ相対の2種類です。

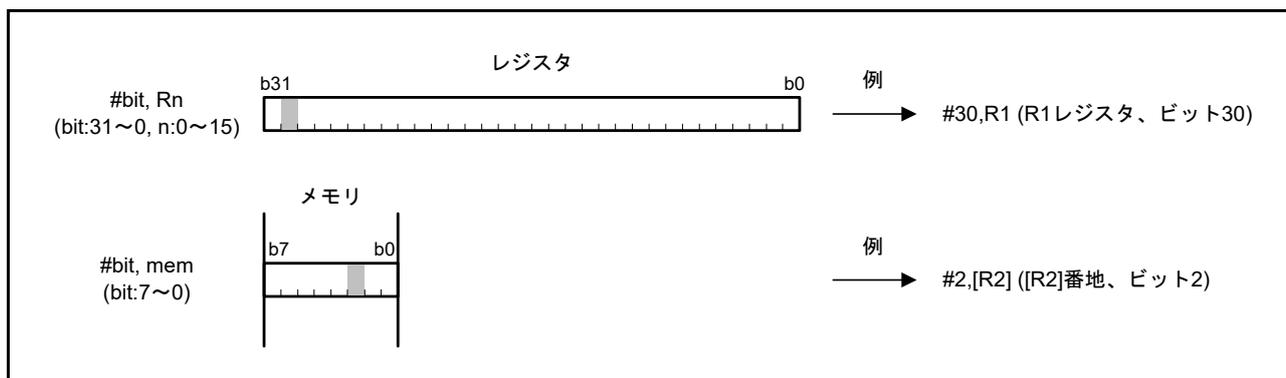


図 1.4 ビット

1.5.4 ストリング

ストリングとは、バイト (8ビット)、ワード (16ビット)、またはロングワード (32ビット) のデータを任意の数だけ連続して並べたデータタイプです。ストリングは、ストリング操作命令 SCMPU、SMOVB、SMOVF、SMOVU、SSTR、SUNTIL、SWHILE の7種類の命令で使用できます。

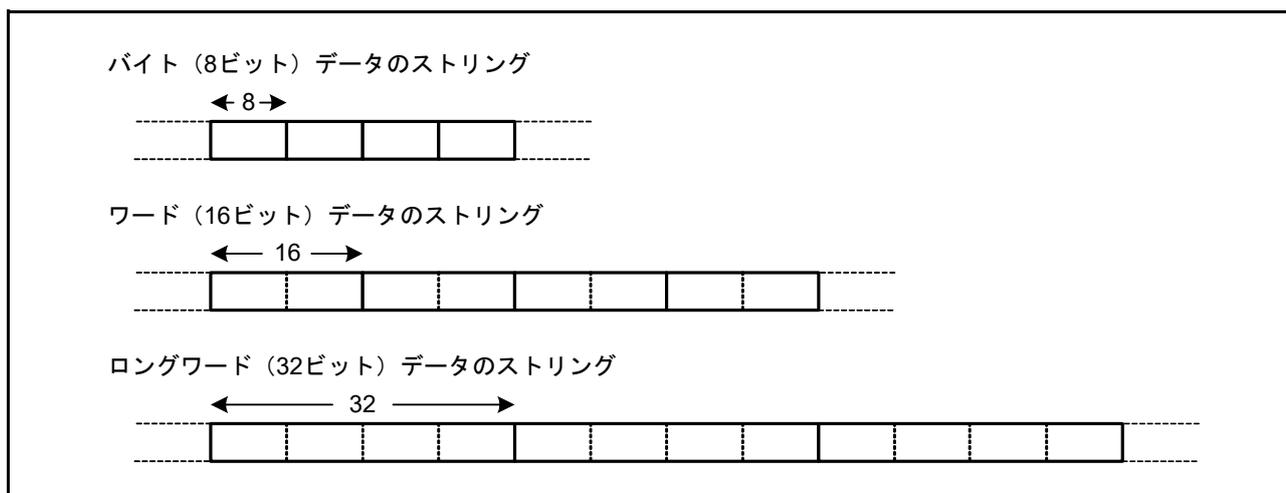


図 1.5 ストリング

1.6 データ配置

1.6.1 レジスタのデータ配置

レジスタのデータサイズと、ビット番号の関係を図 1.6 に示します。

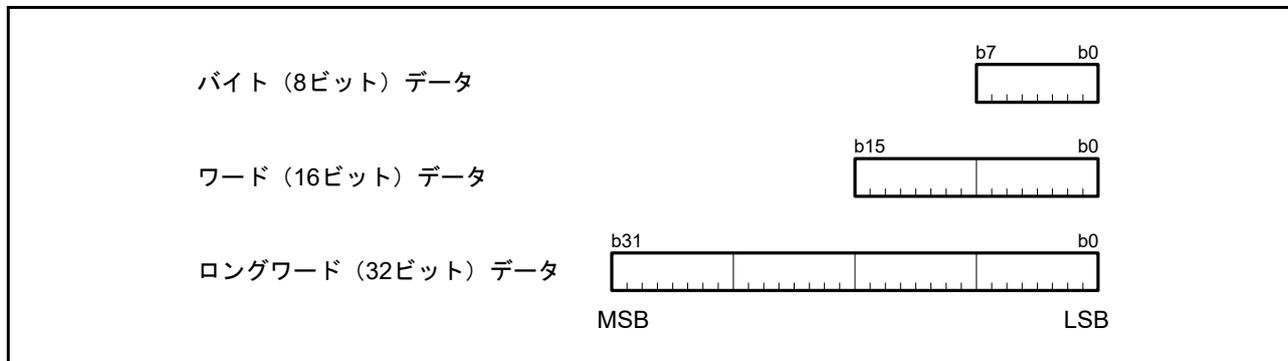


図 1.6 レジスタのデータ配置

1.6.2 メモリ上のデータ配置

メモリ上のデータサイズは、バイト (8 ビット)、ワード (16 ビット)、ロングワード (32 ビット) の 3 種類です。データ配置は、リトルエンディアンか、ビッグエンディアンかを選択することができます。メモリ上のデータ配置を図 1.7 に示します。

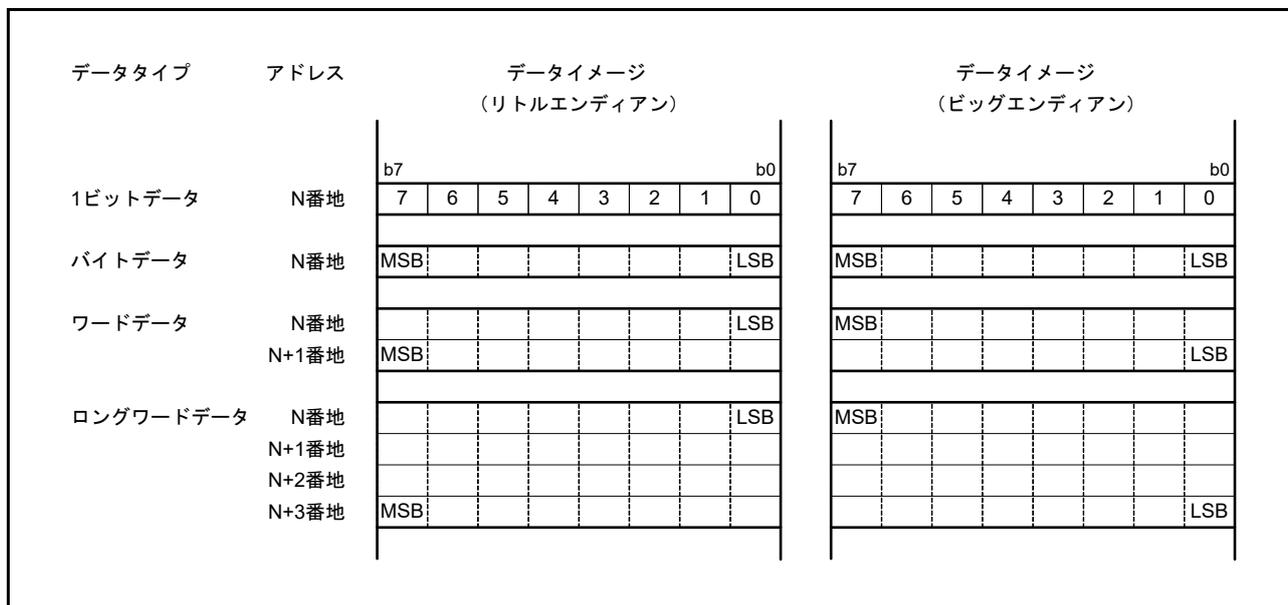


図 1.7 メモリ上のデータ配置

1.7 ベクタテーブル

ベクタテーブルには、例外ベクタテーブルと割り込みベクタテーブルがあります。ベクタテーブルは、1ベクタあたり4バイトで構成されており、各ベクタには対応する例外処理ルーチンの先頭アドレスを設定します。

1.7.1 例外ベクタテーブル

例外ベクタテーブルは、例外テーブルレジスタ (EXTB) の内容で示された値を先頭アドレス (ExtBase) とする 124 バイトの領域に、特権命令例外、アクセス例外、アドレス例外、未定義命令例外、単精度浮動小数点例外、ノンマスカブル割り込みの各ベクタを配置しています。リセットのベクタは、例外ベクタテーブルの値に関係なく常に FFFFFFFCh 番地に配置されます。図 1.8 に例外ベクタテーブルを示します。

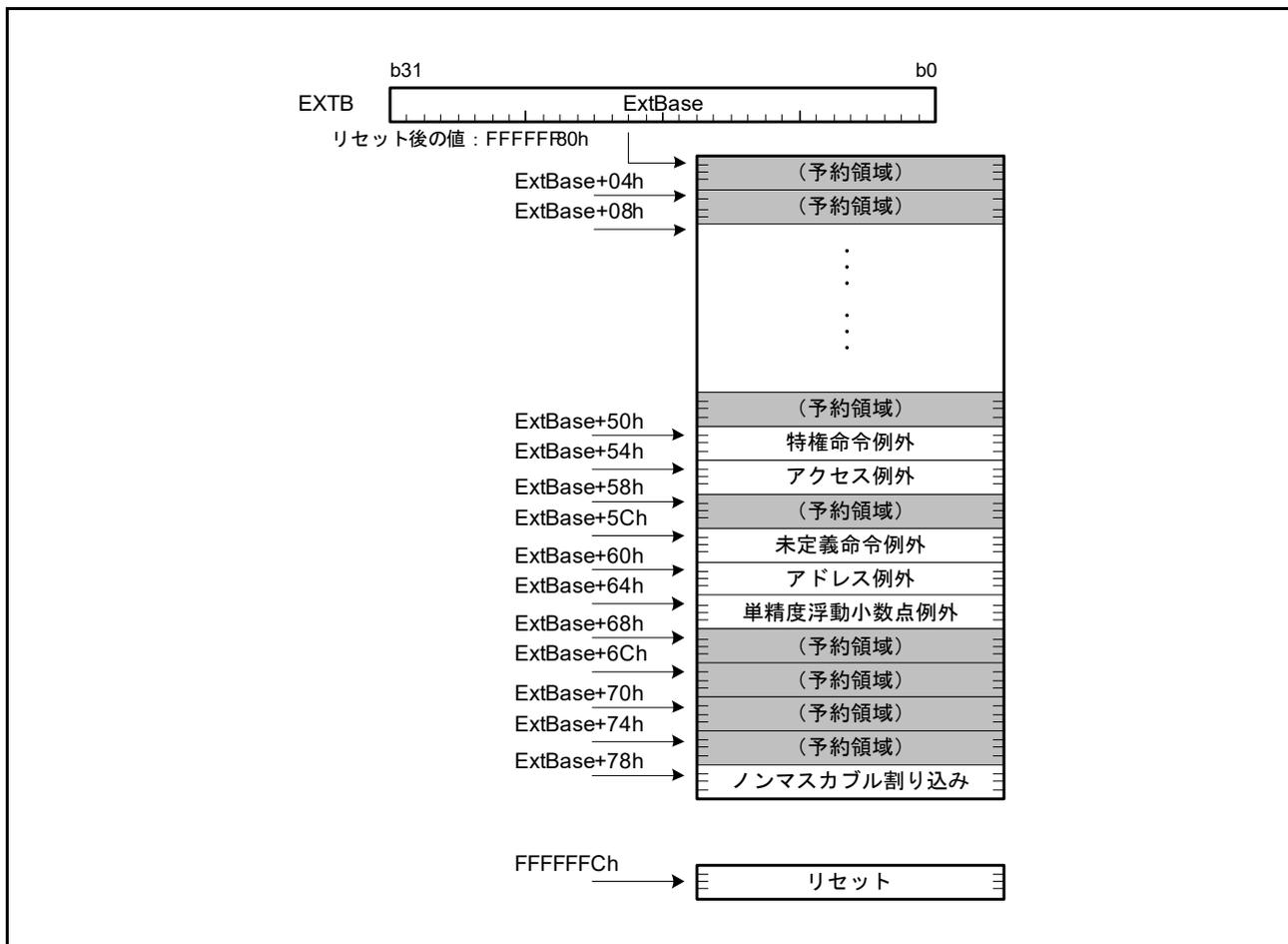


図 1.8 例外ベクタテーブル

1.7.2 割り込みベクタテーブル

割り込みベクタテーブルは、テーブルの配置アドレスを変えることができるベクタテーブルです。割り込みテーブルレジスタ (INTB) の内容で示された値を先頭アドレス (IntBase) とする 1,024 バイトの領域に、無条件トラップ、割り込みの各ベクタを配置しています。図 1.9 に割り込みベクタテーブルを示します。

割り込みベクタテーブルには、ベクタごとに番号 (0 ~ 255) が付けられています。無条件トラップ発生要因の INT 命令では INT 命令番号 (0 ~ 255) に対応したベクタが、BRK 命令では番号 0 のベクタが割り当てられています。また、割り込み要因では、製品ごとに決められた番号 (0 ~ 255) が割り当てられています。

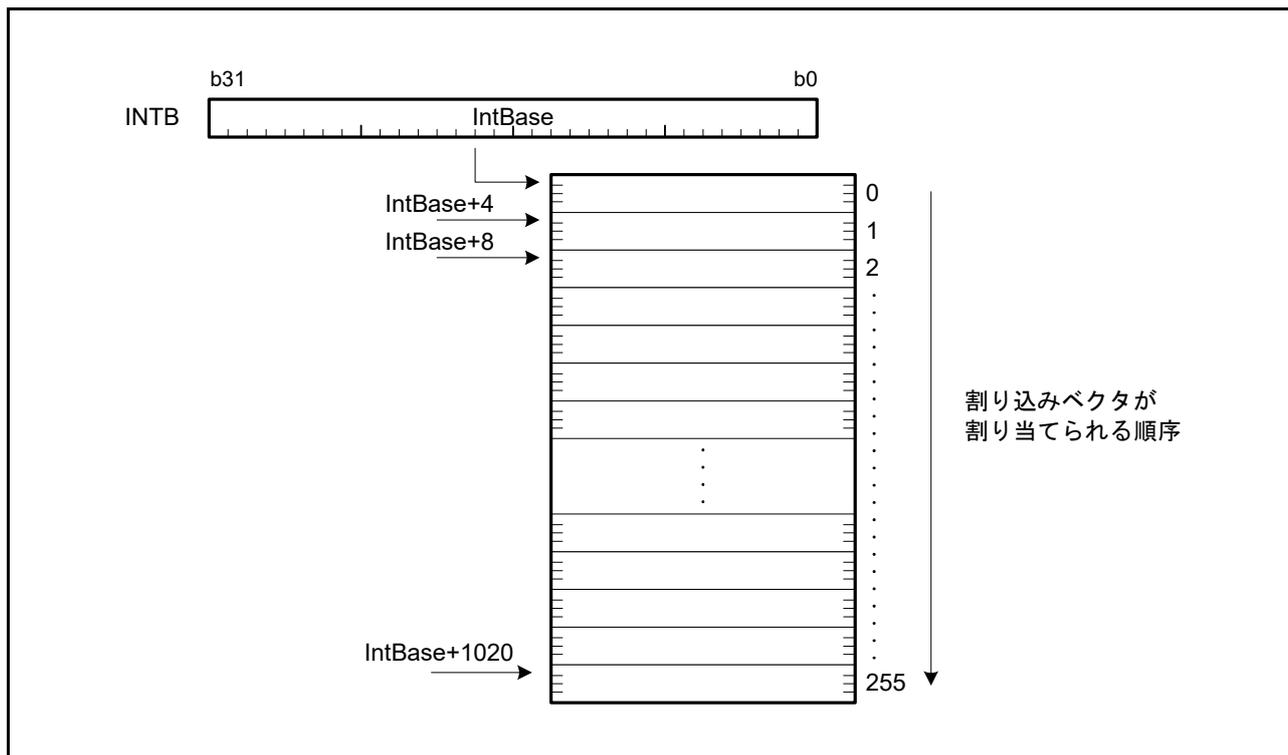


図 1.9 割り込みベクタテーブル

1.8 アドレス空間

RXv3 CPU のアドレス空間は、00000000h 番地から FFFFFFFFh 番地までの 4G バイトあります。プログラム領域およびデータ領域合計最大 4G バイトをリニアにアクセス可能です。RXv3 CPU のアドレス空間を図 1.10 に示します。各領域は、各製品、動作モードによって異なります。詳細は、各製品のユーザーズマニュアルハードウェア編を参照してください。

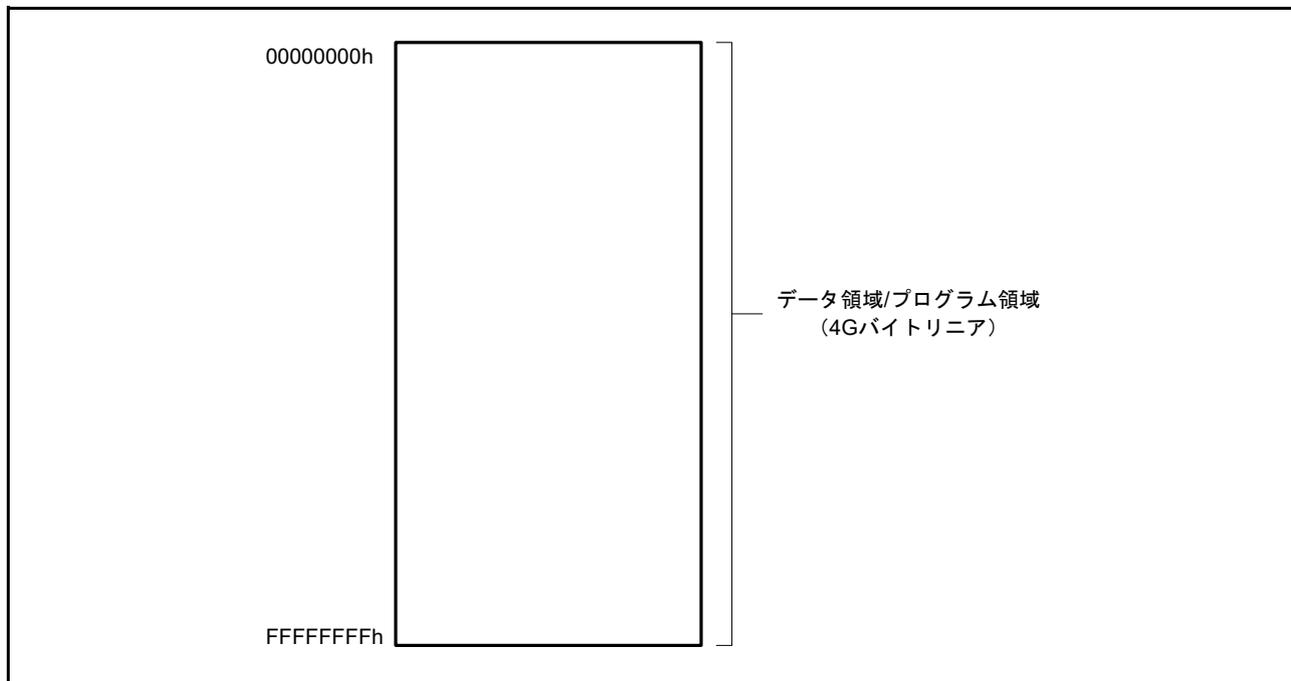


図 1.10 アドレス空間

1.9 レジスタ一括退避機能

RXv3 CPUは、CPUレジスタの退避・復帰を一括して高速に行うために、専用のレジスタ退避バンクとそれを使用するための命令を有しています（図 1.11 参照）。レジスタ退避バンクを使うことで、例外処理ルーチン先頭でのレジスタ退避と、末尾でのレジスタ復帰を一括して高速に行うことが可能です。

レジスタ退避バンクはSAVE命令、RSTR命令のみでアクセス可能な退避用領域であり、4Gバイトのアドレス空間とは独立して存在します。レジスタ退避バンクは複数のバンクで構成されており、1つのバンクに退避・復帰させられるCPUレジスタは、R0を除く汎用レジスタとUSP、FPSW、アキュムレータ（ACC0, ACC1）です。リセット後のレジスタ退避バンクに格納されている値は不定です。

レジスタ退避バンクでは、1つのバンクに対し1つの番号（バンク番号）が割り当てられています。バンク番号の範囲は0～255です。

レジスタ一括退避機能はオプション機能です。レジスタ退避バンクの有無、搭載時に使用できるバンク番号の範囲（搭載容量）は製品ごとに異なります。詳細は、各製品のユーザーズマニュアルハードウェア編を参照してください。

例外処理における利用方法については「5.2 例外の処理手順」を参照してください。

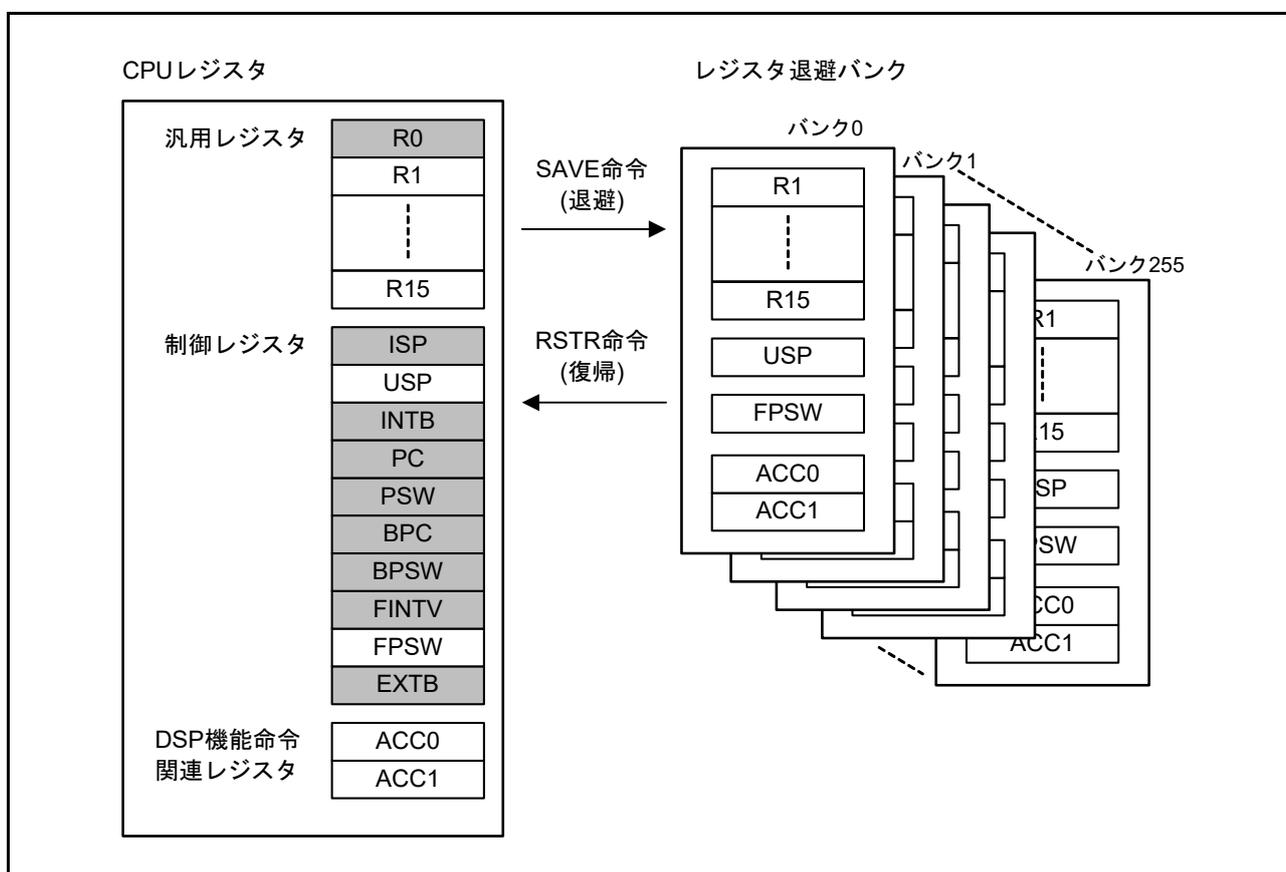


図 1.11 レジスタ退避バンク

1.10 倍精度浮動小数点コプロセッサ

倍精度浮動小数点コプロセッサは、CPUのコプロセッサとして動作し、倍精度浮動小数点処理命令を実行します。倍精度浮動小数点コプロセッサを使用することにより、倍精度浮動小数点演算の処理能力が大きく向上します。

倍精度浮動小数点コプロセッサはオプション機能です。倍精度浮動小数点コプロセッサの搭載有無は製品ごとに異なります。詳細は、各製品のユーザーズマニュアルハードウェア編を参照してください。

1.10.1 特長

- 倍精度浮動小数点レジスタセット
倍精度浮動小数点データレジスタ：64ビット×16本
倍精度浮動小数点制御レジスタ：32ビット×4本
- 倍精度浮動小数点処理命令：21命令
- 倍精度浮動小数点例外の割り込みコントローラへの通知機能

1.10.2 倍精度浮動小数点レジスタセット

倍精度浮動小数点コプロセッサには、倍精度浮動小数点データレジスタ（16本）と倍精度浮動小数点制御レジスタ（4本）があります。

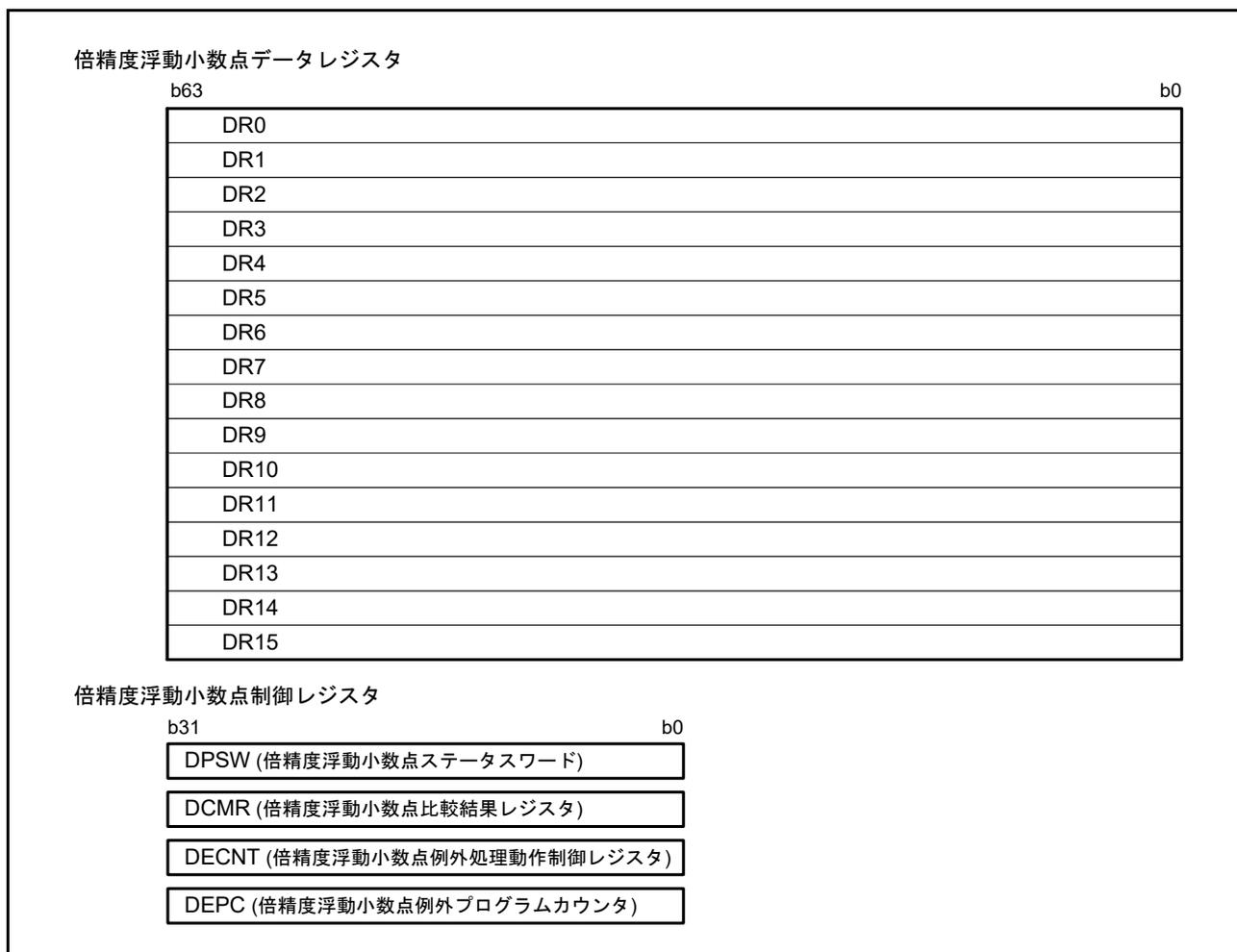


図 1.12 倍精度浮動小数点レジスタセット

1.10.2.1 倍精度浮動小数点データレジスタ (DR0 ~ DR15)

倍精度浮動小数点データレジスタは、64ビット幅で16本 (DR0 ~ DR15) あります。32ビットデータを指定する場合には、上位32ビット (DRH0 ~ DRH15)、下位32ビット (DRL0 ~ DRL15) という単位で扱います。

1.10.2.2 倍精度浮動小数点制御レジスタ

倍精度浮動小数点制御レジスタには、以下の4本のレジスタがあります。

- 倍精度浮動小数点ステータスワード (DPSW)
- 倍精度浮動小数点比較結果レジスタ (DCMR)
- 倍精度浮動小数点例外処理動作制御レジスタ (DECNT)
- 倍精度浮動小数点例外プログラムカウンタ (DEPC)

注. 倍精度浮動小数点制御レジスタを表現するのに何番目の制御レジスタ (DCRn) かを表す別の表記も使用します。

DCR0 : DPSW

DCR1 : DCMR

DCR2 : DECNT

DCR3 : DEPC

(1) 倍精度浮動小数点ステータスワード (DPSW)

	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	DFS	DFX	DFU	DFZ	DFO	DFV	—	—	—	—	—	—	—	—	—	—
リセット後の値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	DEX	DEU	DEZ	DEO	DEV	—	DDN	DCE	DCX	DCU	DCZ	DCO	DCV	DRM[1:0]	
リセット後の値	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

ビット	シンボル	ビット名	機能	R/W
b1-b0	DRM[1:0]	倍精度浮動小数点丸めモード設定ビット	b1 b0 0 0: 最近値への丸め 0 1: 0方向への丸め 1 0: +∞方向への丸め 1 1: -∞方向への丸め	R/W
b2	DCV	無効演算要因フラグ	0: 無効演算の発生なし 1: 無効演算の発生あり	R/(W) (注1)
b3	DCO	オーバフロー要因フラグ	0: オーバフローの発生なし 1: オーバフローの発生あり	R/(W) (注1)
b4	DCZ	ゼロ除算要因フラグ	0: ゼロ除算の発生なし 1: ゼロ除算の発生あり	R/(W) (注1)
b5	DCU	アンダフロー要因フラグ	0: アンダフローの発生なし 1: アンダフローの発生あり	R/(W) (注1)
b6	DCX	精度異常要因フラグ	0: 精度異常の発生なし 1: 精度異常の発生あり	R/(W) (注1)
b7	DCE	非実装処理要因フラグ	0: 非実装処理の発生なし 1: 非実装処理の発生あり	R/(W) (注1)
b8	DDN	非正規化数の0フラッシュビット	0: 非正規化数を非正規化数として扱う 1: 非正規化数を0として扱う (注2)	R/W
b9	—	予約ビット	読むと“0”が読めます。書く場合、“0”としてください	R/W
b10	DEV	無効演算例外処理許可ビット	0: 無効演算の発生による例外処理を禁止 1: 無効演算の発生による例外処理を許可	R/W
b11	DEO	オーバフロー例外処理許可ビット	0: オーバフローによる例外処理を禁止 1: オーバフローによる例外処理を許可	R/W
b12	DEZ	ゼロ除算例外処理許可ビット	0: ゼロ除算の発生による例外処理を禁止 1: ゼロ除算の発生による例外処理を許可	R/W
b13	DEU	アンダフロー例外処理許可ビット	0: アンダフローの発生による例外処理を禁止 1: アンダフローの発生による例外処理を許可	R/W
b14	DEX	精度異常例外処理許可ビット	0: 精度異常の発生による例外処理を禁止 1: 精度異常の発生による例外処理を許可	R/W
b25-b15	—	予約ビット	読むと“0”が読めます。書く場合、“0”としてください	R/W
b26	DFV	無効演算フラグ (注3)	0: 無効演算の発生なし 1: 無効演算の発生あり (注8)	R/W
b27	DFO	オーバフローフラグ (注4)	0: オーバフローの発生なし 1: オーバフローの発生あり (注8)	R/W
b28	DFZ	ゼロ除算フラグ (注5)	0: ゼロ除算の発生なし 1: ゼロ除算の発生あり (注8)	R/W
b29	DFU	アンダフローフラグ (注6)	0: アンダフローの発生なし 1: アンダフローの発生あり (注8)	R/W

ビット	シンボル	ビット名	機能	R/W
b30	DFX	精度異常フラグ(注7)	0: 精度異常の発生なし 1: 精度異常の発生あり(注8)	R/W
b31	DFS	倍精度浮動小数点エラーサマリフラグ	DFU、DFZ、DFO、DFVフラグの論理和を反映	R

- 注1. “0”を書いた場合、“0”になります。“1”を書いた場合、前の値を保持します。
 注2. 正の非正規化数は+0、負の非正規化数は-0として扱います。
 注3. DEVビットが“0”のときに、DFVフラグは有効となります。
 注4. DEOビットが“0”のときに、DFOフラグは有効となります。
 注5. DEZビットが“0”のときに、DFZフラグは有効となります。
 注6. DEUビットが“0”のときに、DFUフラグは有効となります。
 注7. DEXビットが“0”のときに、DFXフラグは有効となります。
 注8. 当該ビットが一度“1”になると、ソフトウェアで“0”にするまで“1”を保持します。

倍精度浮動小数点ステータスワード (DPSW) は、倍精度浮動小数点演算結果を示します。

例外処理許可ビット (DEj) で例外処理を許可 (DEj = 1) した場合は、例外処理ルーチン (倍精度浮動小数点例外を要因とする割り込みの処理ルーチン) で該当する DCj フラグをチェックし、例外発生の要因を判断することができます。例外処理を禁止 (DEj = 0) した場合は、一連の処理の最後に DFj フラグをチェックし、例外発生の有無を確認することができます。DFj フラグは蓄積フラグです (j = X, U, Z, O, V)。

なお、倍精度浮動小数点演算では単精度浮動小数点ステータスワード (FPSW) を参照・更新しません。

DRM[1:0] ビット (倍精度浮動小数点丸めモード設定ビット)

倍精度浮動小数点丸めモードを設定します。

【倍精度浮動小数点丸めモードの説明】

- 最近値への丸め (デフォルト) : 無限の有効桁を持つと仮定して計算した結果に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
 - 0方向への丸め : 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)
 - +∞方向への丸め : 結果の値が大きくなる方向へ丸める
 - -∞方向への丸め : 結果の値が小さくなる方向へ丸める
- (1) 「最近値への丸め」はデフォルトのモードであり、最も正確な値を返します。
 (2) 「0方向への丸め」、「+∞方向への丸め」、「-∞方向への丸め」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。

DCV フラグ (無効演算要因フラグ)、DCO フラグ (オーバフロー要因フラグ)、

DCZ フラグ (ゼロ除算要因フラグ)、DCU フラグ (アンダフロー要因フラグ)、

DCX フラグ (精度異常要因フラグ)、DCE フラグ (非実装処理要因フラグ)

IEEE754 規格で規定された5つの例外 (オーバフロー、アンダフロー、精度異常、ゼロ除算、無効演算) の他に、非実装処理が発生した場合に該当するフラグが“1”になります。

- 倍精度浮動小数点演算命令 (DABS、DNEG を除く) 実行時に、これらの例外・非実装処理が発生しなかった場合、該当するフラグが“0”になります。
- MVTDC 命令で“0”を書いた場合、“0”になります。“1”を書いた場合、前の値を保持します。

DDN ビット (非正規化数の0フラッシュビット)

“0”のとき非正規化数を非正規化数として扱います。

“1”のとき非正規化数を0として扱います。

DEV ビット (無効演算例外処理許可ビット)、DEO ビット (オーバフロー例外処理許可ビット)、DEZ ビット (ゼロ除算例外処理許可ビット)、DEU ビット (アンダフロー例外処理許可ビット)、DEX ビット (精度異常例外処理許可ビット)

倍精度浮動小数点演算命令実行により、IEEE754 規格で規定された 5 つの例外が発生したときに、例外処理 (割り込みコントローラへの割り込み要求) を行うかどうかを制御します。“0” の場合、割り込み要求を行いません。“1” の場合、割り込み要求を行います。

DFV フラグ (無効演算フラグ)、DFO フラグ (オーバフローフラグ)、DFZ フラグ (ゼロ除算フラグ)、DFU フラグ (アンダフローフラグ)、DFX フラグ (精度異常フラグ)

例外処理許可ビット DE_j が “0” (例外処理を禁止) の場合、IEEE754 規格で規定された 5 つの例外が発生すると、該当するフラグが “1” になります。

- DE_j = 1 (例外処理を許可) のときは、このフラグは動きません。
- 当該フラグが “1” になると、ソフトウェアで “0” にするまで “1” を保持します (蓄積フラグ)。

DFS フラグ (倍精度浮動小数点エラーサマリフラグ)

DFU、DFZ、DFO、DFV フラグの論理和を反映します。

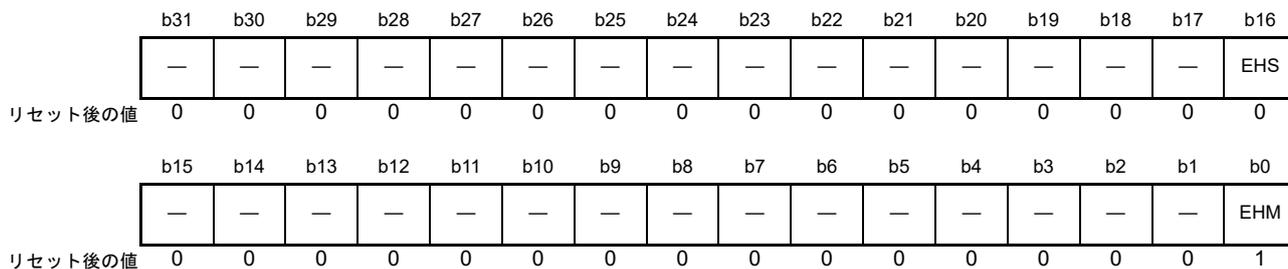
DECNT レジスタの EHM ビットが “1” でかつ EHS ビットが “1” のとき、このレジスタの値は更新されません。

(2) 倍精度浮動小数点比較結果レジスタ (DCMR)

	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
リセット後の値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	RES
リセット後の値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

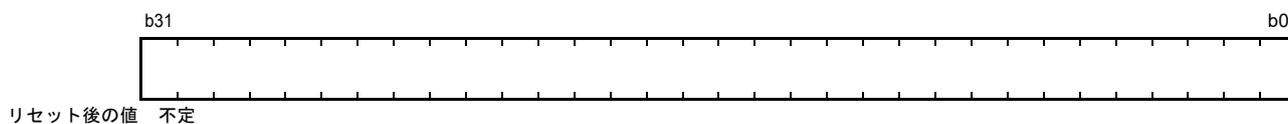
ビット	シンボル	ビット名	機能	R/W
b0	RES	倍精度浮動小数点比較命令結果フラグ	0 : 比較条件が非成立 1 : 比較条件が成立	R/W
b31-b1	—	予約ビット	読むと “0” が読めます。書く場合、“0” としてください	R/W

(3) 倍精度浮動小数点例外処理動作制御レジスタ (DECNT)



ビット	シンボル	ビット名	機能	R/W
b0	EHM	倍精度浮動小数点例外発生情報保持モードビット	0: 例外発生情報非保持モード 倍精度浮動小数点例外発生後、例外発生時の情報を保持しません。 1: 例外発生情報保持モード 倍精度浮動小数点例外が発生し、割り込みコントローラへの割り込み要求を行ったときにEHSビットを“1”として、例外発生時の情報を保持します。	R/W
b15-b1	—	予約ビット	読むと“0”が読めます。書く場合、“0”としてください	R/W
b16	EHS	倍精度浮動小数点例外発生情報保持ステータスビット	0: 例外発生情報保持なし 1: 例外発生情報保持中 EHMビットが“1”でかつEHSビットが“1”のとき、以下のレジスタが更新されなくなり、倍精度浮動小数点例外による割り込みが発生しなくなります。 • 倍精度浮動小数点ステータスワード • 倍精度浮動小数点例外プログラムカウンタ このビットに“0”を書き込むと例外発生時の情報保持が解除されます。	R/W
b31-b17	—	予約ビット	読むと“0”が読めます。書く場合、“0”としてください	R/W

(4) 倍精度浮動小数点例外プログラムカウンタ (DEPC)



倍精度浮動小数点例外が発生し、割り込みコントローラへの割り込み要求を行ったときに、倍精度浮動小数点例外が発生した命令のプログラムカウンタ値がこのレジスタに保持されます。このレジスタはリードオンリです。DECNTレジスタのEHMビットが“1”でかつEHSビットが“1”のとき、このレジスタの値は更新されません。

1.10.3 倍精度浮動小数点例外

倍精度浮動小数点例外は、倍精度浮動小数点演算命令実行時に、IEEE754規格で規定された5つの例外（オーバフロー、アンダフロー、精度異常、ゼロ除算、無効演算）および非実装処理を検出した場合に発生します。倍精度浮動小数点例外が発生すると、CPUでの例外処理は行われず、割り込みコントローラへの割り込み要求が行われます。5つの例外については、対応するDPSWのDEX、DEU、DEZ、DEO、DEVビットが“1”のときのみ割り込み要求が行われます。

以下に、倍精度浮動小数点例外を発生させる事象の概要を示します。

1.10.3.1 オーバフロー

オーバフローは、演算結果の絶対値が倍精度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。DFOF命令の場合は変換結果の絶対値が単精度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。表1.8にオーバフロー発生時の演算結果を示します。

表1.8 オーバフロー発生時の倍精度浮動小数点演算結果

倍精度浮動小数点丸めモード	結果の符号	演算結果（デスティネーションレジスタの値）	
		DEO = 0	DEO = 1
-∞方向への丸め	+	+MAX	変化なし
	-	-∞	
+∞方向への丸め	+	+∞	
	-	-MAX	
0方向への丸め	+	+MAX	
	-	-MAX	
最近値への丸め	+	+∞	
	-	-∞	

注. DEO = 0のとき、倍精度浮動小数点演算のオーバフローが発生すると精度異常が発生します。

1.10.3.2 アンダフロー

アンダフローは、演算結果の絶対値が倍精度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（ただし、0を除く）に発生します。DFOF命令の場合は変換結果の絶対値が単精度浮動小数点フォーマットで表現可能な値よりも小さくなった場合（ただし、0を除く）に発生します。表1.9にアンダフロー発生時の演算結果を示します。

表1.9 アンダフロー発生時の倍精度浮動小数点演算結果

演算結果（デスティネーションレジスタの内容）	
DEU = 0	DEU = 1
DDN = 0のとき、変化しない（非実装処理が発生）	変化なし
DDN = 1のとき、“0”を返す	

1.10.3.3 精度異常

精度異常は、無限の有効桁を持つと仮定して計算した結果と、演算結果が異なっていた場合に発生します。表 1.10 に精度異常発生条件と演算結果を示します。

表 1.10 精度異常発生条件と倍精度浮動小数点演算結果

発生条件	演算結果（デスティネーションレジスタの内容）	
	DEX = 0	DEX = 1
オーバフロー例外禁止状態でのオーバフロー発生	「表 1.8 オーバフロー発生時の倍精度浮動小数点演算結果」参照	変化なし
丸めの発生	丸め後の値	

注. アンダフロー発生時、精度異常は発生しません。

注. オーバフロー例外許可状態でのオーバフロー発生時、丸めの発生にかかわらず、精度異常は発生しません。

1.10.3.4 ゼロ除算

ゼロ除算は、0 でない有限数を 0 で割った場合に発生します。表 1.11 にゼロ除算発生時の演算結果を示します。ただし、被除数が表 1.12 に示す値の場合、ゼロ除算は発生しません。

表 1.11 ゼロ除算発生時の倍精度浮動小数点演算結果

被除数	演算結果（デスティネーションレジスタの内容）	
	DEZ = 0	DEZ = 1
0 でない有限数	$\pm\infty$ （符号は除数、被除数の符号の排他的論理和となる）	変化なし

表 1.12 ゼロ除算が発生しないときの動作

被除数	動作
0	無効演算発生
∞	例外は発生しない。結果は ∞
非正規化数（DDN = 0）	非実装処理発生
QNaN	例外は発生しない。結果はQNaN
SNaN	無効演算発生

1.10.3.5 無効演算

無効演算は、無効な演算が実行された場合に発生します。表 1.13 に無効演算発生条件と演算結果を示します。

表 1.13 無効演算発生条件と倍精度浮動小数点演算結果

発生条件	演算結果（デスティネーションレジスタの内容）	
	DEV = 0	DEV = 1
SNaNオペランドに対する演算	QNaN	変化なし
$+\infty + (-\infty)$ 、 $+\infty - (+\infty)$ 、 $-\infty - (-\infty)$		
$0 \times \infty$		
$0 \div 0$ 、 $\infty \div \infty$		
0より小さい数に対する平方根演算		
DTOI命令、DROUND命令実行時、整数変換がオーバーフローするか、NaN、 ∞ を整数変換したとき	変換前の符号ビットが、“0”のときは7FFFFFFFh、“1”のときは80000000hを返す	
DTOU命令実行時、整数変換がオーバーフローするか、NaN、 ∞ を整数変換したとき	変換前の最上位ビットが、“0”のときはFFFFFFFh、“1”のときは00000000hを返す	
SNaNオペランドに対する比較	デスティネーションはなし	

表 1.14 に演算結果が QNaN となる場合の生成規則を示します。

表 1.14 QNaN生成規則

ソースオペランド	演算結果（デスティネーションレジスタの内容）
SNaNとQNaN	QNaN化されたSNaNソースオペランド
ともにSNaN	QNaN化されたsrc2
ともにQNaN	src2
SNaNと実数	QNaN化されたSNaNソースオペランド
QNaNと実数	QNaNソースオペランド
どちらもNaNでないケースで無効演算発生時	7FFFFFFFFFFFFFFFh

注. SNaNのQNaN化は、仮数部の最上位ビットを“1”にして行います。

1.10.3.6 非実装処理

倍精度浮動小数点演算の非実装処理は、DDN = 0 で非正規化数が演算オペランドとして与えられるか、DDN = 0 で演算の結果、アンダフローが発生した場合に発生します。DDN = 1 では非実装処理は発生しません。

非実装処理発生による例外処理を禁止することはできません（非実装処理発生による例外処理を禁止する例外処理許可ビットはありません）。デスティネーションレジスタは変化しません。

1.10.4 データタイプ（倍精度浮動小数点コプロセッサ）

倍精度浮動小数点コプロセッサは、倍精度浮動小数点数を扱うことができます。

また、DFOB、FTOD 命令では単精度浮動小数点数、DROUND、DFOI、DFOU、ITOD、UTOD 命令では32ビット整数も扱えます。単精度浮動小数点数については、「1.5.2 単精度浮動小数点数」、32ビット整数については、「1.5.1 整数」を参照してください。

1.10.4.1 倍精度浮動小数点数

倍精度浮動小数点数は、IEEE754 で規定されている倍精度浮動小数点数に対応しています。倍精度浮動小数点数は、倍精度浮動小数点演算命令 DABS、DADD、DCMPcm、DDIV、DMUL、DNEG、DROUND、DSUB、DSQRT、DFOB、DFOI、DFOU、FTOD、ITOD、UTOD の15種類の命令で使用できます。

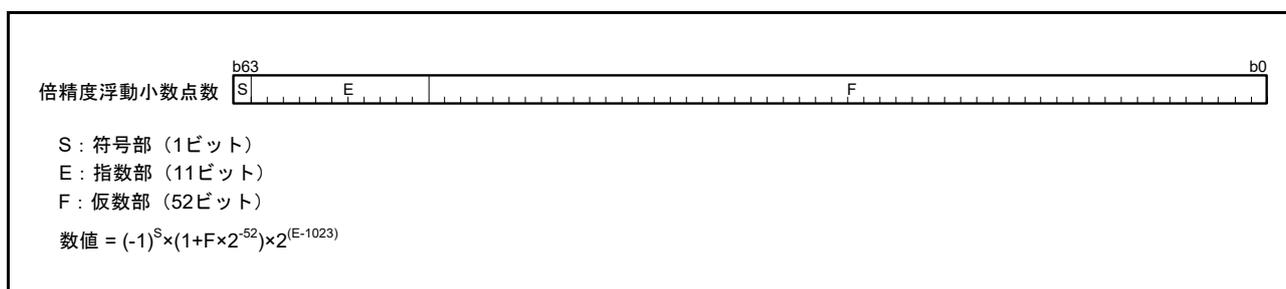


図 1.13 倍精度浮動小数点数

倍精度浮動小数点数は、以下の数値に対応しています。

- $0 < E < 2047$ (正規化数 - Normal Number)
- $E = 0$ かつ $F = 0$ (ゼロ - Signed Zero)
- $E = 0$ かつ $F > 0$ (非正規化数 - Subnormal Number) (注 1)
- $E = 2047$ かつ $F = 0$ (無限大 - Infinity)
- $E = 2047$ かつ $F > 0$ (非数 - NaN : Not a Number)

注 1. DPSW の DDN ビットが“1”のときは、0として扱います。DDN ビットが“0”のときは、非実装処理が発生します。

1.10.5 データ配置（倍精度浮動小数点コプロセッサ）

倍精度浮動小数点コプロセッサは、32ビットデータと64ビットデータを扱うことができます。32ビットデータについては、「1.6 データ配置」を参照してください。

1.10.5.1 倍精度浮動小数点レジスタのデータ配置

図 1.14 にレジスタのデータサイズと、ビット番号の関係を示します。

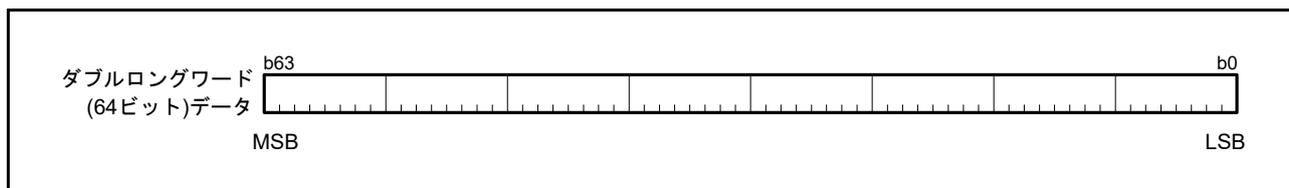


図 1.14 レジスタのデータ配置

1.10.5.2 メモリ上の倍精度浮動小数点データ配置

倍精度浮動小数点のメモリ上のデータサイズは、ダブルロングワード（64ビット）の1種類です。データ配置は、リトルエンディアンか、ビッグエンディアンかを選択することができます。図 1.15 にメモリ上のデータ配置を示します。

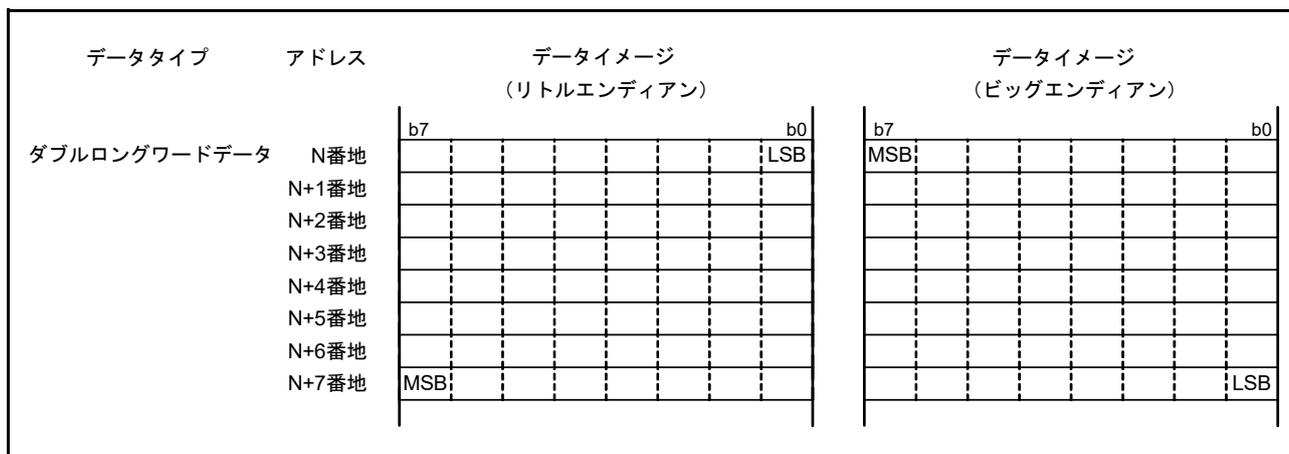


図 1.15 メモリ上のデータ配置

2. アドレッシングモード

アドレッシングモードごとに、アドレッシングモードを示す記号、動作について説明します。
アドレッシングモードは、以下に示す 11 種類があります。

- 即値
- レジスタ直接
- レジスタ間接
- レジスタ相対
- ポストインクリメントレジスタ間接
- プリデクリメントレジスタ間接
- インデックス付きレジスタ間接
- 制御レジスタ直接
- PSW 直接
- プログラムカウンタ相対
- アキュムレータ直接

倍精度浮動小数点コプロセッサを搭載している製品では、以下に示す 2 種類のアドレッシングモードが追加されます。

- 倍精度浮動小数点データレジスタ直接
- 倍精度浮動小数点制御レジスタ直接

2.1 本章の見方

本章の見方を以下に実例をあげて示します。

(1) レジスタ相対		
(2) dsp:5[Rn] (Rn = R0~R7)	ディスプレイースメント (dsp) の値を32ビットにゼロ拡張した後、規則に従い (右図参照)、1/2/4倍した値と、レジスタ値を加算した結果の下位32ビットが演算対象の実効アドレスと	<p>レジスタ Rn → address</p> <p>dsp → (×) → (+) → address</p> <p>メモリ</p> <p>アドレス増加方向</p> <ul style="list-style-type: none"> • サイズ指定子をとる命令 <ul style="list-style-type: none"> .Bのとき : 1倍 .Wのとき : 2倍 .Lのとき : 4倍 .Dのとき : 4倍 • サイズ拡張指定子をとる命令 <ul style="list-style-type: none"> .B/.UBのとき : 1倍 .W/.UWのとき : 2倍 .Lのとき : 4倍
(3) dsp:8[Rn] (Rn = R0~R15)	ディスプレイースメント (dsp) の値を32ビットにゼロ拡張した後、規則に従い (右図参照)、1/2/4倍した値と、レジスタ値を加算した結果の下位32ビットが演算対象の実効アドレスとなります。実効アドレスの範囲は、00000000h~FFFFFFFhです。dsp:nは、nビット長の	
(4) R0~R15)	ディスプレイースメントを表します。	
dsp:16[Rn] (Rn = R0~R15)	dsp:5[Rn] (Rn = R0~R7)、dsp:8[Rn] (Rn = R0~R15)、dsp:16[Rn] (Rn = R0~R15) が指定できます。dsp:5[Rn] (Rn = R0~R7)は、MOV、MOVU命令でのみ使用されます。	

(1) 名称

アドレッシングモードの名称です。

(2) 記号

アドレッシングモードを示す記号です。

“:8”、“:16”は、直前の値の有効ビット数を示します。マニュアルの記載上、有効ビット数を明記する必要があるために付加していますが、プログラムを記述するときは、付加する必要はありません。

(3) 解説

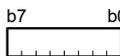
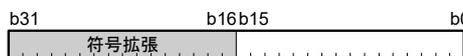
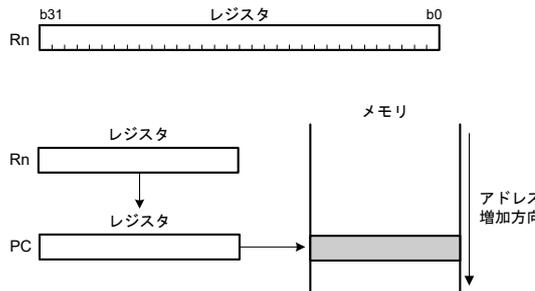
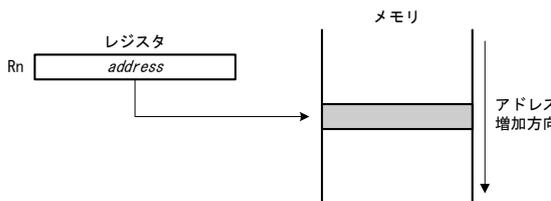
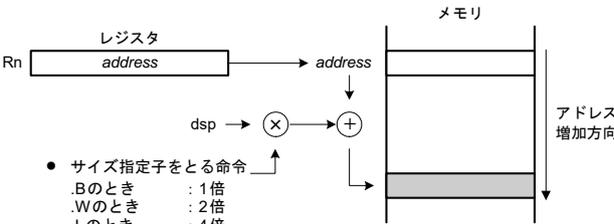
動作、実効アドレスの範囲を説明します。

(4) 動作図

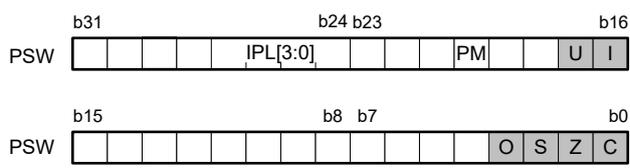
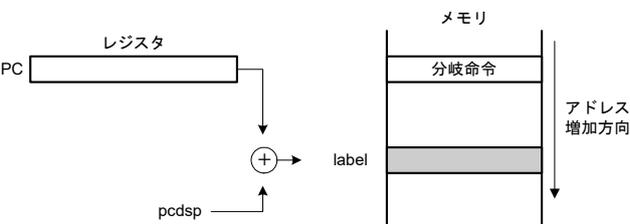
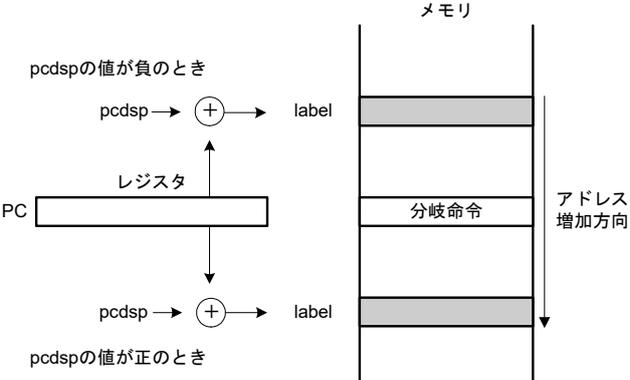
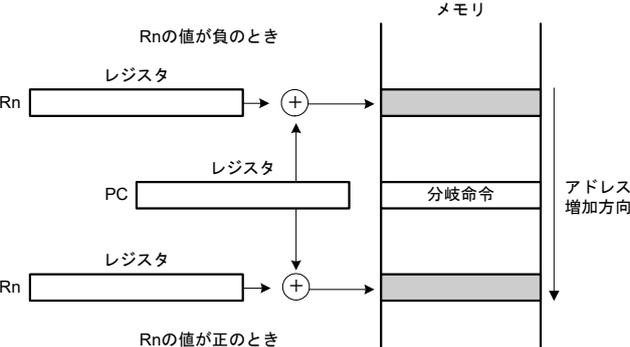
動作を図で説明します。

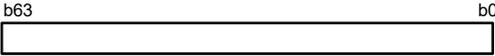
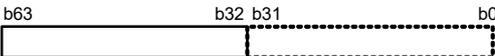
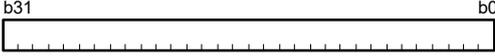
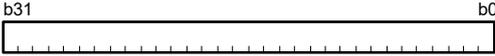
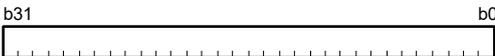
2.2 アドレッシングモード

即値		
#IMM:1	<ul style="list-style-type: none"> • #IMM:1 	#IMM:1 
#IMM:2	<p>#IMMで示した1ビット即値が演算の対象となります。このアドレッシングモードは、RACL、RACW、RDACL、RDACW命令のソースで使用されます。</p> <ul style="list-style-type: none"> • #IMM:2 <p>#IMMで示した2ビット即値が演算の対象となります。このアドレッシングモードは、MVFACGU、MVFACHI、MVFACLO、MVFACMI命令のソースで使用されます。</p> <ul style="list-style-type: none"> • #IMM:3 <p>#IMMで示した3ビット即値が演算の対象となります。このアドレッシングモードは、ビット操作命令（BCLR、BMCnd、BNOT、BSET、BTST）のビット番号指定で使用されます。</p> <ul style="list-style-type: none"> • #IMM:4 <p>#IMMで示した4ビット即値が演算の対象となります。このアドレッシングモードは、MVTIPL命令の割り込み優先レベル指定で使用されます。</p> <ul style="list-style-type: none"> • #UIMM:4 <p>#UIMMで示した4ビット即値を32ビットにゼロ拡張した結果が演算の対象となります。このアドレッシングモードは、ADD、AND、CMP、MOV、MUL、OR、SUB命令のソースで使用されます。</p> <ul style="list-style-type: none"> • #IMM:5 <p>#IMMで示した5ビット即値が演算の対象となります。このアドレッシングモードは、ビット操作命令（BCLR、BMCnd、BNOT、BSET、BTST）のビット番号指定、転送命令（BFMOV、BFMOVZ）のビット番号指定とビット幅指定、算術/論理演算命令（SHAR、SHLL、SHLR）のシフト幅指定、および算術/論理演算命令（ROTL、ROTR）のローテート幅指定で使用されます。</p>	#IMM:2 
#IMM:3		#IMM:3 
#IMM:4		#IMM:4 
#UIMM:4		#UIMM:4 
#IMM:5		#IMM:5 

<p>即値</p> <p>#IMM:8 #SIMM:8 #UIMM:8 #IMM:16 #SIMM:16 #SIMM:24 #IMM:32</p>	<p>即値で指定した値が演算の対象となります。ただし、#UIMMで指定した即値は処理サイズにゼロ拡張した結果が、#SIMMで指定した即値は処理サイズに符号拡張した結果が演算の対象となります。#IMM:n、#UIMM:n、#SIMM:nは、nビット長の即値を表します。IMMの範囲は、「2.2.1 IMMの範囲」を参照してください。</p>	<p>処理サイズがBのとき</p> <p>#IMM:8 </p> <p>処理サイズがWのとき</p> <p>#SIMM:8 </p> <p>#UIMM:8 </p> <p>#IMM:16 </p> <p>処理サイズがLのとき</p> <p>#UIMM:8 </p> <p>#SIMM:8 </p> <p>#SIMM:16 </p> <p>#SIMM:24 </p> <p>#IMM:32 </p>
<p>レジスタ直接</p> <p>Rn (Rn = R0 ~ R15)</p>	<p>指定したレジスタが演算の対象となります。またはJMP、JSR命令の場合、Rnの値をプログラムカウンタ(PC)に転送します。実効アドレスの範囲は、00000000h ~ FFFFFFFFhです。Rn (Rn = R0 ~ R15)が指定できます。</p>	
<p>レジスタ間接</p> <p>[Rn] (Rn = R0 ~ R15)</p>	<p>レジスタの値が演算対象の実効アドレスとなります。実効アドレスの範囲は、00000000h ~ FFFFFFFFhです。[Rn] (Rn = R0 ~ R15)が指定できます。</p>	
<p>レジスタ相対</p> <p>dsp:5[Rn] (Rn = R0 ~ R7)</p> <p>dsp:8[Rn] (Rn = R0 ~ R15)</p> <p>dsp:16[Rn] (Rn = R0 ~ R15)</p>	<p>ディスプレイメント (dsp) の値を32ビットにゼロ拡張した後、規則に従い (右図参照)、1/2/4倍した値と、レジスタ値を加算した結果の下位32ビットが演算対象の実効アドレスとなります。実効アドレスの範囲は、00000000h ~ FFFFFFFFhです。dsp:nは、nビット長のディスプレイメントを表します。dsp:5[Rn] (Rn = R0 ~ R7)、dsp:8[Rn] (Rn = R0 ~ R15)、dsp:16[Rn] (Rn = R0 ~ R15)が指定できます。dsp:5[Rn] (Rn = R0 ~ R7)は、MOV、MOVU命令でのみ使用されます。</p>	 <ul style="list-style-type: none"> ● サイズ指定子をとる命令 <ul style="list-style-type: none"> .Bのとき : 1倍 .Wのとき : 2倍 .Lのとき : 4倍 .Dのとき : 4倍 ● サイズ拡張指定子をとる命令 <ul style="list-style-type: none"> .B/.UBのとき : 1倍 .W/.UWのとき : 2倍 .Lのとき : 4倍

<p>ポストインクリメントレジスタ間接</p> <p>[Rn+] (Rn = R0 ~ R15)</p>	<p>レジスタの値が演算対象の実効アドレスとなります。実効アドレスの範囲は、00000000h ~ FFFFFFFFhです。演算実行後、レジスタの値にサイズ指定子.B/W/Lに応じてそれぞれ1/2/4を加算します。このアドレッシングモードは、MOV、MOVU命令で使用されます。</p>	<p>レジスタ Rn address</p> <p>メモリ</p> <p>① ② ③ +</p> <p>サイズ指定子.Bのとき :1加算 サイズ指定子.Wのとき :2加算 サイズ指定子.Lのとき :4加算</p> <p>アドレス増加方向</p>
<p>プリデクリメントレジスタ間接</p> <p>[-Rn] (Rn = R0 ~ R15)</p>	<p>レジスタの値にサイズ指定子.B/W/Lに応じてそれぞれ1/2/4を減算します。減算後の値が演算対象の実効アドレスとなります。実効アドレスの範囲は、00000000h ~ FFFFFFFFhです。このアドレッシングモードは、MOV、MOVU命令で使用されます。</p>	<p>メモリ</p> <p>サイズ指定子.Bのとき :1減算 サイズ指定子.Wのとき :2減算 サイズ指定子.Lのとき :4減算</p> <p>レジスタ Rn address</p> <p>① ② ③ ④ -</p> <p>アドレス増加方向</p>
<p>インデックス付きレジスタ間接</p> <p>[Ri,Rb] (Ri, Rb = R0 ~ R15)</p>	<p>インデックスレジスタ (Ri) の値をサイズ指定子.B/W/Lに応じてそれぞれ1/2/4倍した値と、ベースレジスタ (Rb) の値を加算した結果の下位32ビットが演算対象の実効アドレスとなります。実効アドレスの範囲は、00000000h ~ FFFFFFFFhです。このアドレッシングモードは、MOV、MOVU命令で使用されます。</p>	<p>ベースレジスタ Rb address</p> <p>インデックスレジスタ Ri</p> <p>メモリ</p> <p>① ② ③ ④ × +</p> <p>サイズ指定子.Bのとき:1倍 サイズ指定子.Wのとき:2倍 サイズ指定子.Lのとき:4倍</p> <p>アドレス増加方向</p>
<p>制御レジスタ直接</p> <p>PC ISP USP INTB PSW BPC BPSW FINTV FPSW EXTB</p>	<p>指定した制御レジスタが演算の対象となります。このアドレッシングモードは、MVFC、MVTC、POPC、PUSHC命令で使用されます。PCはMVFC、PUSHC命令のsrcにのみ指定できます。</p>	<p>レジスタ</p> <p>PC ISP USP INTB PSW BPC BPSW FINTV FPSW EXTB</p> <p>b31 b0</p>

PSW直接		
C Z S O I U	指定したフラグ、またはビットが演算の対象となります。このアドレッシングモードは、CLRPSW、SETPSW命令で使用されます。	 <p>PSW bit fields: b31 to b16 (IPL[3:0], PM, U, I) and b15 to b0 (O, S, Z, C).</p>
プログラムカウンタ相対		
pcdsp:3	分岐距離指定子が“.S”のとき、プログラムカウンタ (PC) にディスプレイメント (pcdsp) の値を符号なしで加算した結果の低位32ビットが実効アドレスとなります。分岐の範囲は、3～10です。実効アドレスの範囲は、00000000h～FFFFFFFhです。このアドレッシングモードは、BCnd (BEQ、BZ、BNE、BNZのみ)、BRA命令で使用されます。	 <p>レジスタ (PC) と pcdsp が加算され、メモリ (分岐命令) の label へジャンプ。アドレス増加方向は下向き。</p>
pcdsp:8 pcdsp:16 pcdsp:24	分岐距離指定子が“.B”、“.W”、または“.A”のとき、プログラムカウンタ (PC) の値と、ディスプレイメント (pcdsp) の値を符号付きで加算した結果が実効アドレスとなります。pcdspの範囲は、 “.B”のとき：-128 ≤ pcdsp:8 ≤ 127 “.W”のとき：-32768 ≤ pcdsp:16 ≤ 32767 “.A”のとき：-8388608 ≤ pcdsp:24 ≤ 8388607 となります。実効アドレスの範囲は、00000000h～FFFFFFFhです。このアドレッシングモードは、“.B”のときBCnd、BRA命令で、“.W”のときBCnd (BEQ、BZ、BNE、BNZのみ)、BRA、BSR命令で、“.A”のときBRA、BSR命令で使用されます。	 <p>レジスタ (PC) と pcdsp が加算され、メモリ (分岐命令) の label へジャンプ。アドレス増加方向は下向き。</p>
Rn (Rn = R0～R15)	プログラムカウンタ (PC) の値と、Rnの値を符号付きで加算した結果が実効アドレスとなります。Rnの値の範囲は、-2147483648～2147483647です。実効アドレスの範囲は、00000000h～FFFFFFFhです。このアドレッシングモードは、BRA(.L)、BSR(.L)命令で使用されます。	 <p>レジスタ (PC) と Rn が加算され、メモリ (分岐命令) の label へジャンプ。アドレス増加方向は下向き。</p>
アキュムレータ直接		
A0、A1 (A0 = ACC0、A1 = ACC1)	指定したアキュムレータ ACC0、ACC1が演算の対象となります。	 <p>An bit fields: b71 to b0.</p>

倍精度浮動小数点データレジスタ直接		
DRn (DRn = DR0 ~ DR15)	指定した倍精度浮動小数点データレジスタが演算の対象となります。DRLnでは下位32ビットが、DRHnでは上位32ビットが対象となります。	DRn 
DRLn (DRLn = DRL0 ~ DRL15)		DRLn 
DRHn (DRHn = DRH0 ~ DRH15)		DRHn 
倍精度浮動小数点制御レジスタ直接		
DPSW DCMR DECNT DEPC	指定した倍精度浮動小数点制御レジスタが演算の対象となります。	DPSW 
		DCMR 
		DECNT 
		DEPC 

2.2.1 IMM の範囲

IMM の範囲を表 2.1 に示します。

「3.5 命令詳細説明」の各命令に特に記述がない場合、IMM の範囲は以下となります。

表 2.1 IMMの範囲

IMM	10進表記	16進表記
IMM:1	1, 2	1h, 2h
IMM:2	0~2	0h~2h
IMM:3	0~7	0h~7h
IMM:4	0~15	0h~0Fh
UIMM:4	0~15	0h~0Fh
IMM:5	0~31	0h~1Fh
IMM:8	-128~255	-80h~0FFh
UIMM:8	0~255	0h~0FFh
SIMM:8	-128~127	-80h~7Fh
IMM:16	-32768~65535	-8000h~0FFFFh
SIMM:16	-32768~32767	-8000h~7FFFh
SIMM:24	-8388608~8388607	-800000h~7FFFFFFh
IMM:32	-2147483648~4294967295	-80000000h~0FFFFFFFFh

- 注.
1. 弊社の「RXファミリ アセンブラ」では、IMMを最適なビット長の命令コードに変換します。
 2. 弊社の「RXファミリ アセンブラ」では、16進表記は32ビット表記も可能です。
例. 10進表記“-127”、16進表記“-7Fh”は、“0FFFFFF81h”とも表記できます。
 3. INT命令とRTSD命令のIMMの範囲は、「3.5 命令詳細説明」の該当命令を参照してください。

3. 命令

3.1 命令セット概要

RXv3アーキテクチャの命令数は113です。1～8バイト可変長命令形式を採用しています。

RXv3命令セットアーキテクチャは、RXv2とRXv1に対して上位互換性のある命令セットアーキテクチャです。

- RXv2はRXv1に対し、主にDSP機能命令と単精度浮動小数点演算命令が拡充され、DSPに匹敵するデータ処理能力を実現しています。
- RXv3はRXv2に対し、主にレジスタ一括退避機能の搭載により割り込み応答性能の向上を実現しています。
- 倍精度浮動小数点コプロセッサを搭載している製品では、さらに21種類の命令（倍精度浮動小数点処理命令）が追加になります。

RXv3命令セットについて以下に示します。

命令一覧表 (1/5)

命令の種類	ニーモニック	機能	命令詳細 記載ページ	命令コード 詳細記載 ページ	備考
標準搭載命令					
算術/論理 演算命令	ABS	絶対値	70	255	
	ADC	キャリ付き加算	71	256	
	ADD	キャリなし加算	72	257	
	AND	論理積	74	259	
	CMP	比較	88	273	
	DIV	符号付き除算	89	274	
	DIVU	符号なし除算	90	276	
	EMUL	符号付き乗算	93	278	
	EMULU	符号なし乗算	96	279	
	MAX	最大値選択	125	291	
	MIN	最小値選択	126	292	
	MUL	乗算	137	301	
	NEG	符号反転	152	310	
	NOP	ノーオペレーション	153	310	
	NOT	論理反転	154	311	
	OR	論理和	155	312	
	RMPA	積和演算	172	320	
	ROLC	キャリ付き左回転	174	320	
	RORC	キャリ付き右回転	175	320	
	ROTL	左回転	176	321	
	ROTR	右回転	177	321	
	SAT	32ビット符号付き飽和処理	186	323	
	SATR	RMPA命令用64ビット符号付き飽和処理	187	324	
	SBB	ボロー付き減算	188	324	
	SHAR	算術右シフト	192	327	
	SHLL	論理/算術左シフト	193	328	
SHLR	論理右シフト	194	329		
SUB	ボローなし減算	201	333		
TST	テスト	206	335		
XOR	排他的論理和	211	338	RXv3で拡張	

命令一覧表 (2 / 5)

命令の種類	ニーモニック	機能	命令詳細 記載ページ	命令コード 詳細記載 ページ	備考	
単精度浮動小数点 演算命令	FADD	単精度浮動小数点加算	98	281	RXv2で拡張	
	FCMP	単精度浮動小数点比較	101	282		
	FDIV	単精度浮動小数点除算	103	283		
	FMUL	単精度浮動小数点乗算	105	284	RXv2で拡張	
	FSUB	単精度浮動小数点減算	110	286	RXv2で拡張	
	FSQRT	単精度浮動小数点平方根	108	285	RXv2以降でサポート	
	FTOI	単精度浮動小数点数→符号付き整数変換	113	287		
	FTOU	単精度浮動小数点数→符号なし整数変換	115	287	RXv2以降でサポート	
	ITOF	符号付き整数→単精度浮動小数点数変換	118	288		
	ROUND	単精度浮動小数点数→符号付き整数変換	178	322		
	UTOF	符号なし整数→単精度浮動小数点数変換	207	336	RXv2以降でサポート	
転送命令	BFMOV	ビットフィールド転送	77	264	RXv3以降でサポート	
	BFMOVZ	ゼロへのビットフィールド転送	78	264	RXv3以降でサポート	
	MOV	転送	127	293		
	MOVCO	LIフラグクリア付きストア	131	298	RXv2以降でサポート	
	MOVLI	LIフラグセット付きロード	132	298	RXv2以降でサポート	
	MOVU	符号なしデータ転送	133	299		
	POP	スタックからレジスタへのデータ復帰	156	313		
	POPC	制御レジスタの復帰	157	314	RXv2で拡張	
	POPM	複数レジスタの復帰	158	314		
	PUSH	スタックヘデータ退避	159	315		
	PUSHC	制御レジスタの退避	160	316	RXv2で拡張	
	PUSHM	複数レジスタの退避	161	316		
	REVL	エンディアン変換	170	319		
	REVV	エンディアン変換	171	319		
	SCCnd	SCGEU	条件設定	189	325	
		SCC		189	325	
		SCEQ		189	325	
		SCZ		189	325	
		SCGTU		189	325	
		SCPZ		189	325	
		SCGE		189	325	
		SCGT		189	325	
		SCO		189	325	
		SCLTU		189	325	
		SCNC		189	325	
		SCNE		189	325	
		SCNZ		189	325	
		SCLEU		189	325	
		SCN		189	325	
		SCLE		189	325	
	SCLT	189	325			
	SCNO	189	325			
STNZ	条件付き転送	199	331	RXv2で拡張		
STZ	条件付き転送	200	332	RXv2で拡張		
XCHG	交換	210	337			

命令一覧表 (3 / 5)

命令の種類	ニーモニック	機能	命令詳細 記載ページ	命令コード 詳細記載 ページ	備考
分岐命令	BCnd	BGEU	相対条件分岐	76	263
		BC		76	263
		BEQ		76	263
		BZ		76	263
		BGTU		76	263
		BPZ		76	263
		BGE		76	263
		BGT		76	263
		BO		76	263
		BLTU		76	263
		BNC		76	263
		BNE		76	263
		BNZ		76	263
		BLEU		76	263
		BN		76	263
		BLE		76	263
	BLT	76	263		
	BNO	76	263		
		BRA	相対無条件分岐	82	267
		BSR	相対サブルーチン分岐	85	270
	JMP	無条件分岐	120	289	
	JSR	サブルーチン分岐	121	289	
	RTS	サブルーチンからの復帰	183	323	
	RTSD	スタックフレームの解放とサブルーチンからの復帰	184	323	

命令一覧表 (4 / 5)

命令の種類	ニーモニック	機能	命令詳細 記載ページ	命令コード 詳細記載 ページ	備考	
ビット操作命令	BCLR	ビットクリア	75	261		
	BMCnd	BMGEU	条件ビット転送	79	265	
		BMC		79	265	
		BMEQ		79	265	
		BMZ		79	265	
		BMGTU		79	265	
		BMPZ		79	265	
		BMGE		79	265	
		BMGT		79	265	
		BMO		79	265	
		BMLTU		79	265	
		BMNC		79	265	
		BMNE		79	265	
		BMNZ		79	265	
		BMLEU		79	265	
		BMN		79	265	
BMLE	79	265				
BMLT	79	265				
BMNO	79	265				
BNOT	ビット反転	81	266			
BSET	ビットセット	84	268			
BTST	ビットテスト	86	271			
ストリング 操作命令	SCMPU	ストリング比較	190	325		
	SMOVb	逆方向ストリング転送	195	329		
	SMOVf	順方向ストリング転送	196	330		
	SMOVu	ストリング転送	197	330		
	SSTR	ストリングストア	198	330		
	SUNTIL	ストリングサーチ	202	334		
	SWHILE	ストリングサーチ	204	334		
システム 操作命令	BRK	無条件トラップ	83	268		
	CLRPSW	PSWのフラグ、ビットのクリア	87	272		
	INT	ソフトウェア割り込み	117	288		
	MVFC	制御レジスタからの転送	146	306	RXv2で拡張	
	MVTC	制御レジスタへの転送	150	308	RXv2で拡張	
	MVTIPL (特権命令)	割り込み優先レベル設定	151	309		
	RTE (特権命令)	例外からの復帰	181	322	RXv2で拡張	
	RTFI (特権命令)	高速割り込みからの復帰	182	322	RXv2で拡張	
	SETPSW	PSWのフラグ、ビットのセット	191	326		
WAIT (特権命令)	ウェイト	209	337			

命令一覧表 (5 / 5)

命令の種類	ニーモニック	機能	命令詳細 記載ページ	命令コード 詳細記載 ページ	備考
DSP機能命令	EMACA	32ビット積和演算	91	277	RXv2以降でサポート
	EMSBA	32ビット積差演算	92	277	RXv2以降でサポート
	EMULA	32ビット乗算	95	279	RXv2以降でサポート
	MACHI	上位16ビット積和演算	122	290	RXv2で拡張
	MACLH	下位16ビット・上位16ビット積和演算	123	290	RXv2以降でサポート
	MACLO	下位16ビット積和演算	124	290	RXv2で拡張
	MSBHI	上位16ビット積差演算	134	300	RXv2以降でサポート
	MSBLH	下位16ビット・上位16ビット積差演算	135	300	RXv2以降でサポート
	MSBLO	下位16ビット積差演算	136	301	RXv2以降でサポート
	MULHI	上位16ビット乗算	139	303	RXv2で拡張
	MULLH	下位16ビット・上位16ビット乗算	140	303	RXv2以降でサポート
	MULLO	下位16ビット乗算	141	304	RXv2で拡張
	MVFACGU	アキュムレータガードビットからの転送	142	304	RXv2以降でサポート
	MVFACHI	アキュムレータ上位32ビットからの転送	143	305	RXv2で拡張
	MVFACLO	アキュムレータ下位32ビットからの転送	144	305	RXv2以降でサポート
	MVFACMI	アキュムレータ中央32ビットからの転送	145	306	RXv2で拡張
	MVTACGU	アキュムレータガードビットへの転送	147	307	RXv2以降でサポート
	MVTACHI	アキュムレータ上位32ビットへの転送	148	307	RXv2で拡張
	MVTACLO	アキュムレータ下位32ビットへの転送	149	307	RXv2で拡張
	RACL	符号付きアキュムレータ丸め処理	162	317	RXv2以降でサポート
RACW	16ビット符号付きアキュムレータ丸め処理	164	317	RXv2で拡張	
RDACL	符号付きアキュムレータ丸め処理	166	318	RXv2以降でサポート	
RDACW	16ビット符号付きアキュムレータ丸め処理	168	318	RXv2以降でサポート	
レジスタ一括退避機能命令 (オプション)					
レジスタ一括退避 機能命令	RSTR (特権命令)	レジスタの一括復帰	213	341	RXv3以降でサポート
	SAVE (特権命令)	レジスタの一括退避	214	341	
倍精度浮動小数点処理命令 (オプション)					
倍精度浮動小数点 転送命令	DMOV	倍精度浮動小数点データ転送	223	345	倍精度浮動小数点 コプロセッサ搭載製品で サポート
	DPOPM	複数の倍精度浮動小数点レジスタの復帰	228	349	
	DPUSHM	複数の倍精度浮動小数点レジスタの退避	230	350	
	MVFDC	倍精度浮動小数点制御レジスタからの転送	247	354	
	MVFDR	倍精度浮動小数点比較結果レジスタからの転送	248	354	
	MVTDC	倍精度浮動小数点制御レジスタへの転送	249	355	
倍精度浮動小数点 演算命令	DABS	倍精度浮動小数点絶対値	216	343	倍精度浮動小数点 コプロセッサ搭載製品で サポート
	DADD	倍精度浮動小数点加算	217	343	
	DCMPcm	倍精度浮動小数点比較	219	344	
	DDIV	倍精度浮動小数点除算	221	344	
	DMUL	倍精度浮動小数点乗算	225	348	
	DNEG	倍精度浮動小数点符号反転	227	348	
	DROUND	倍精度浮動小数点数→符号付き整数変換	232	351	
	DSQRT	倍精度浮動小数点平方根	234	351	
	DSUB	倍精度浮動小数点減算	236	351	
	DTOF	倍精度浮動小数点数→単精度浮動小数点数変換	238	352	
	DTOI	倍精度浮動小数点数→符号付き整数変換	240	352	
	DTOU	倍精度浮動小数点数→符号なし整数変換	242	352	
	FTOD	単精度浮動小数点数→倍精度浮動小数点数変換	244	353	
	ITOD	符号付き整数→倍精度浮動小数点数変換	246	353	
UTOD	符号なし整数→倍精度浮動小数点数変換	250	355		

3.2 RXv3 拡張命令一覧

RXv3アーキテクチャでは、RXv2アーキテクチャから4個の命令を追加（新規追加命令）し、1個の命令について仕様を拡張（仕様拡張命令）しています。

3.2.1 RXv3 新規追加命令

表3.1にRXv3で、RXv2命令セットから新規に追加された命令一覧を示します。

表3.1 新規に追加された命令一覧表

分類	ニーモニック	機能概要
転送命令	BFMOV	ビットフィールド転送
	BFMOVZ	ゼロへのビットフィールド転送
レジスタ一括退避機能命令（オプション）	RSTR（特権命令）	レジスタの一括復帰
	SAVE（特権命令）	レジスタの一括退避

3.2.2 RXv3 仕様拡張命令

表3.2にRXv3で、RXv2命令セットから仕様が拡張された命令一覧を示します。

表3.2 仕様が拡張された命令一覧表

分類	ニーモニック	仕様拡張概要
算術/論理演算命令	XOR	3オペランド(src, src2, dst)を追加、(Rs, Rs2, Rd)を指定可能

3.3 倍精度浮動小数点処理命令一覧

表3.3に倍精度浮動小数点処理命令一覧を示します。

表3.3 倍精度浮動小数点処理命令一覧表

分類	ニーモニック	機能概要
倍精度浮動小数点転送命令 (オプション)	DMOV	倍精度浮動小数点データ転送
	DPOPM	複数の倍精度浮動小数点レジスタの復帰
	DPUSHM	複数の倍精度浮動小数点レジスタの退避
	MVFDC	倍精度浮動小数点制御レジスタからの転送
	MVFDR	倍精度浮動小数点比較結果レジスタからの転送
	MVTDC	倍精度浮動小数点制御レジスタへの転送
倍精度浮動小数点演算命令 (オプション)	DABS	倍精度浮動小数点絶対値
	DADD	倍精度浮動小数点加算
	DCMPcm	倍精度浮動小数点比較
	DDIV	倍精度浮動小数点除算
	DMUL	倍精度浮動小数点乗算
	DNEG	倍精度浮動小数点符号反転
	DROUND	倍精度浮動小数点数→符号付き整数変換
	DSQRT	倍精度浮動小数点平方根
	DSUB	倍精度浮動小数点減算
	DTOF	倍精度浮動小数点数→単精度浮動小数点数変換
	DTOI	倍精度浮動小数点数→符号付き整数変換
	DTOU	倍精度浮動小数点数→符号なし整数変換
	FTOD	単精度浮動小数点数→倍精度浮動小数点数変換
	ITOD	符号付き整数→倍精度浮動小数点数変換
UTOD	符号なし整数→倍精度浮動小数点数変換	

3.4 本章の見方

本章では、構文、オペレーション、機能、選択可能なsrc/dest、フラグ変化、記述例について命令ごとに説明しています。

本章の見方を以下に実例をあげて示します。

(1) **ABS**

(4) **【構文】**
 (1) ABS dest
 (2) ABS src, dest

(5) **【オペレーション】**
 (1) if (dest < 0)
 dest = -dest;
 (2) if (src < 0)
 dest = -src;
 else
 dest = src;

(6) **【機能】**
 (1) destの絶対値をとり、その結果をdestに格納します。
 (2) srcの絶対値をとり、その結果をdestに格納します。

(7) **【フラグ変化】**

フラグ	変化	条件
C	—	
Z	○	演算後のdestが0のとき“1”、それ以外のとき“0”になります。
S	○	演算後のdestのMSBが“1”のとき“1”、それ以外のとき“0”になります。
O	○	(1) 演算前のdestが80000000hのとき“1”、それ以外のとき“0”になります。 (2) 演算前のsrcが80000000hのとき“1”、それ以外のとき“0”になります。

(8) **【命令フォーマット】**

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
(1) ABS dest	L	—	Rd	2
(2) ABS src, dest	L	Rs	Rd	3

(9) **【記述例】**
 ABS R2
 ABS R1, R2

絶対値
ABSolute

ABS

(2) **算術/論理演算命令**

(3) **【命令コード】**

記載ページ：260

(1) ニーモニック

本ページで説明するニーモニックを示しています。中央には命令の簡単な動作とフルネームを記載しています。

(2) 命令の種類

命令の種類を示します。

(3) 命令コード

命令コードの記載ページを示しています。
命令コードについては、このページを参照してください。

(4) 構文

命令の構文を記号で示しています。

(a) ニーモニック

ニーモニックを記述します。

(b) サイズ指定子 .size

転送命令、ストリング操作命令の一部、およびRMPA命令では、ニーモニックの語尾にサイズ指定子を指定することができます。以下のとおり、サイズ指定子により、取り扱うデータサイズを指定します。

.B バイト (8 ビット)
.W ワード (16 ビット)
.L ロングワード (32 ビット)
.D ダブルロングワード (64 ビット)

(c) オペランド src、dest

オペランドを記述します。

src ソースオペランド
dest デスティネーションオペランド
Asrc ソースオペランド (アキュムレータ)
Adest デスティネーションオペランド (アキュムレータ)

(5) オペレーション

命令のオペレーションを示します。動作表記はC言語の表記方法に準じています。

(a) データタイプ

signed char	符号付きバイト (8 ビット) 整数
signed short	符号付きワード (16 ビット) 整数
signed long	符号付きロングワード (32 ビット) 整数
signed long long	符号付きロングロングワード (64 ビット) 整数
unsigned char	符号なしバイト (8 ビット) 整数
unsigned short	符号なしワード (16 ビット) 整数
unsigned long	符号なしロングワード (32 ビット) 整数
unsigned long long	符号なしロングロングワード (64 ビット) 整数
float	単精度浮動小数点数
double	倍精度浮動小数点数

(b) 擬似関数

register(n)	: n をレジスタ番号に持つレジスタ Rn を返します。(n : 0 ~ 15)
register_num(Rn)	: Rn または DRn、DCRn のレジスタ番号 n を返します。
sqrt(src)	: src の平方根を返します。
isNaN(src)	: src が NaN のとき 1 を返します。
DR(n)	: n をレジスタ番号にもつ倍精度浮動小数点データレジスタ DRn を表します。(n : 0 ~ 15)
DCR(n)	: n をレジスタ番号にもつ倍精度浮動小数点制御レジスタ DCRn を表します。(n : 0 ~ 3)
bank(n)	: レジスタ退避バンクのバンク番号 n を表します。

(c) 特殊表記

Rn[i+7:i]	: Rn のビット (i+7) からビット i までの符号なしバイト整数を示します。 (n : 0 ~ 15、i : 24、16、8、0)
Rm:Rn	: 2つのレジスタを連結した仮想的な 64 ビットレジスタを示します。 (m、n : 0 ~ 15、Rm がビット 63 ~ 32、Rn がビット 31 ~ 0 に割り当てられます。)
Rl:Rm:Rn	: 3つのレジスタを連結した仮想的な 96 ビットレジスタを示します。 (l、m、n : 0 ~ 15、Rl がビット 95 ~ 64、Rm がビット 63 ~ 32、Rn がビット 31 ~ 0 に割り当てられます。)
{byte3, byte2, byte1, byte0}	: 4つの符号なしバイト整数が連結した符号なしロングワード整数を示します。
{ R1, R2,..., ACC1 }	: {} 内に列挙されたレジスタの集合を表します。

(6) 機能

命令の機能、注意事項を説明しています。

(7) フラグ変化

PSW のフラグ (O, S, Z, C) の変化を示します。

単精度浮動小数点演算命令は、FPSW のフラグ (FX, FU, FZ, FO, FV, CE, CX, CU, CZ, CO, CV) の変化も示します。

倍精度浮動小数点演算命令は、DPSW のフラグ (DFX, DFU, DFZ, DFO, DFV, DCE, DCX, DCU, DCZ, DCO, DCV) の変化も示します。

表中に示す記号の意味は以下のとおりです。

- : 変化しません。
- : 条件によって変化します。

(8) 命令フォーマット

命令フォーマットを示します。

【命令フォーマット】

	構文	処理サイズ	対象			コードサイズ (バイト)
			src	src2	dest	
(a)	(1) AND src, dest	L	#UIMM:4	—	Rd	2
		L	#SIMM:8	—	Rd	3
(d)		L	#SIMM:16	—	Rd	4
		L	#SIMM:24	—	Rd	5
		L	#IMM:32	—	Rd	6
(f)		L	Rs	—	Rd	2
		L	[Rs].memex	—	Rd	2 (memex == "UB") 3 (memex != "UB")
(e)		L	dsp:8[Rs].memex (注)	—	Rd	3 (memex == "UB") 4 (memex != "UB")
		L	dsp:16[Rs].memex (注)	—	Rd	4 (memex == "UB") 5 (memex != "UB")
	(2) AND src, src2, dest	L	Rs	Rs2	Rd	3

【命令フォーマット】

	構文	処理サイズ	対象		コードサイズ (バイト)
			src	dest (注)	
(b)	MVTC src, dest	L	#SIMM:8	Rx	4
		L	#SIMM:16	Rx	5
		L	#SIMM:24	Rx	6
		L	#IMM:32	Rx	7
		L	Rs	Rx	3

【命令フォーマット】

	構文	対象	コードサイズ (バイト)
		dest	
(c)	SETPSW dest	flag	2

(a) レジスタ

特に断りがない場合、Rs、Rs2、Rd、Rd2、Ri、Rbには、R0～R15が指定できます。

DSP機能命令のアクキュムレータの指定のためにA0、A1が指定できます。

倍精度浮動小数点処理命令のDRs、DRs2、DRd、DRd2にはDR0～DR15が指定できます。DRLs、DRLdにはDRL0～DRL15が指定できます。DRHs、DRHdにはDRH0～DRH15が指定できます。

(b) 制御レジスタ

Rxには、PC、ISP、USP、INTB、PSW、BPC、BPSW、FINTV、FPSW、EXTBが指定できます。

PCはMVFC、PUSHC命令のsrcにのみ指定できます。

倍精度浮動小数点処理命令のDCRs、DCRs2、DCRd、DCRd2には、DPSW、DCMR、DECNT、DEPCが指定できます。

(c) フラグ、ビット

flagには、PSWのビット (U, I)、フラグ (O, S, Z, C) が指定できます。

(d) 即値

#IMM:n、#UIMM:n、#SIMM:nは、nビット長の即値を示します。拡張が必要な場合、UIMMはゼロ拡張、SIMMは符号拡張が行われます。

(e) メモリオペランドに付加されるサイズ拡張指定子 (.memex)

.memexは、メモリオペランドのサイズと拡張方法を示します。指定されたサイズ拡張指定子による拡張方法に従って、処理サイズに拡張された後、各命令は処理されます。

memex	サイズ	拡張方法
B	バイト	符号拡張
UB	バイト	ゼロ拡張
W	ワード	符号拡張
UW	ワード	ゼロ拡張
L	ロングワード	なし

省略した場合、ビット操作命令ではバイトとして扱い、それ以外の命令ではロングワードとして扱います。

(f) 処理サイズ

処理サイズは、CPU内部での転送サイズ、または演算サイズを示します。

(9) 記述例

命令の記述例を示します。

BCnd、BRA、BSR命令の構文について、以下にBRA命令を例にあげて示します。



【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	length	対象		コードサイズ (バイト)
		src	pcdsp / Rsの範囲	
BRA(.length) src	S	pcdsp:3	$3 \leq \text{pcdsp} \leq 10$	1
	B	pcdsp:8	$-128 \leq \text{pcdsp} \leq 127$	2
	W	pcdsp:16	$-32768 \leq \text{pcdsp} \leq 32767$	3
	A	pcdsp:24	$-8388608 \leq \text{pcdsp} \leq 8388607$	4
	L	Rs	$-2147483648 \leq \text{Rs} \leq 2147483647$	2

【記述例】

```
BRA label1
BRA.A label2
BRA R1
BRA.L R2
```

注． 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値 (pcdsp:3, pcdsp:8, pcdsp:16, pcdsp:24) は、分岐先のラベルまたは実効アドレスを指定してください。命令コード (pcdsp) には、指定されたアドレスから命令の配置アドレスを引いた値が埋め込まれます。

【記述例】

```
BRA label
BRA 1000h
```

(4) 構文

命令の構文を記号で示しています。

(a) ニーモニック

ニーモニックを記述します。

(b) 分岐距離指定子 .length

分岐・ジャンプ系命令では、ニーモニックの後に分岐距離指定子を指定することができます。以下のとおり、分岐距離指定子により、分岐の相対距離の有効ビット数を指定します。

- .S 3ビットのPC前方相対を表します。有効値は、3～10です。
- .B 8ビットのPC相対を表します。有効値は、-128～127です。
- .W 16ビットのPC相対を表します。有効値は、-32768～32767です。
- .A 24ビットのPC相対を表します。有効値は、-8388608～8388607です。
- .L 32ビットのPC相対を表します。有効値は、-2147483648～2147483647です。

3.5 命令詳細説明

RXv3各命令の詳細説明を示します。

3.5.1 標準搭載命令

次ページより標準搭載命令の詳細説明を示します。

ABS

絶対値
ABSolute

ABS

【構文】

- (1) ABS dest
- (2) ABS src, dest

算術/論理演算命令

【命令コード】

記載ページ : 255

【オペレーション】

- (1) if (dest < 0)
dest = -dest;
- (2) if (src < 0)
dest = -src;
else
dest = src;

【機能】

- (1) destの絶対値をとり、その結果をdestに格納します。
- (2) srcの絶対値をとり、その結果をdestに格納します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	演算後のdestが0のとき“1”、それ以外のとき“0”になります。
S	○	演算後のdestのMSBが“1”のとき“1”、それ以外のとき“0”になります。
O	○	(1) 演算前のdestが80000000hのとき“1”、それ以外のとき“0”になります。 (2) 演算前のsrcが80000000hのとき“1”、それ以外のとき“0”になります。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
(1) ABS dest	L	—	Rd	2
(2) ABS src, dest	L	Rs	Rd	3

【記述例】

```
ABS R2
ABS R1, R2
```

ADC

キャリ付き加算
ADD with Carry

ADC

【構文】

ADC src, dest

算術/論理演算命令

【命令コード】

記載ページ：256

【オペレーション】

dest = dest + src + C;

【機能】

- destとsrcとCフラグを加算し、その結果をdestに格納します。

【フラグ変化】

フラグ	変化	条件
C	○	符号なし演算のオーバーフローが発生したとき“1”、それ以外るとき“0”になります。
Z	○	演算後のdestが0のとき“1”、それ以外るとき“0”になります。
S	○	演算後のdestのMSBが“1”のとき“1”、それ以外るとき“0”になります。
O	○	符号付き演算のオーバーフローが発生したとき“1”、それ以外るとき“0”になります。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
ADC src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].L	Rd	4
	L	dsp:8[Rs].L (注)	Rd	5
	L	dsp:16[Rs].L (注)	Rd	6

注. 弊社の「RXファミリ アセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、4の倍数を指定してください。dsp:8には、0～1020 (255×4) が指定できます。dsp:16には、0～262140 (65535×4) が指定できます。命令コードには、1/4した値が埋め込まれます。

【記述例】

ADC #127, R2

ADC R1, R2

ADC [R1], R2

ADDキャリなし加算
ADD**ADD**

【構文】

- (1) ADD src, dest
 (2) ADD src, src2, dest

算術/論理演算命令

【命令コード】

記載ページ：257

【オペレーション】

- (1) dest = dest + src;
 (2) dest = src2 + src;

【機能】

- (1) destとsrcを加算し、その結果をdestに格納します。
 (2) src2とsrcを加算し、その結果をdestに格納します。

【フラグ変化】

フラグ	変化	条件
C	○	符号なし演算のオーバーフローが発生したとき“1”、それ以外るとき“0”になります。
Z	○	演算後のdestが0のとき“1”、それ以外るとき“0”になります。
S	○	演算後のdestのMSBが“1”のとき“1”、それ以外るとき“0”になります。
O	○	符号付き演算のオーバーフローが発生したとき“1”、それ以外るとき“0”になります。

【命令フォーマット】

構文	処理サイズ	対象			コードサイズ (バイト)
		src	src2	dest	
(1) ADD src, dest	L	#UIMM:4	—	Rd	2
	L	#SIMM:8	—	Rd	3
	L	#SIMM:16	—	Rd	4
	L	#SIMM:24	—	Rd	5
	L	#IMM:32	—	Rd	6
	L	Rs	—	Rd	2
	L	[Rs].memex	—	Rd	2 (memex == “UB”) 3 (memex != “UB”)
	L	dsp:8[Rs].memex (注)	—	Rd	3 (memex == “UB”) 4 (memex != “UB”)
(2) ADD src, src2, dest	L	#SIMM:8	Rs	Rd	3
	L	#SIMM:16	Rs	Rd	4
	L	#SIMM:24	Rs	Rd	5
	L	#IMM:32	Rs	Rd	6
	L	Rs	Rs2	Rd	3

注. 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、サイズ拡張指定子が“.W”または“.UW”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:8には、サイズ拡張指定子が“.W”または“.UW”のとき0～510 (255 × 2) が、“.L”のとき0～1020 (255 × 4) が指定できます。dsp:16には、サイズ拡張指定子が“.W”または“.UW”のとき0～131070 (65535 × 2) が、“.L”のとき0～262140 (65535 × 4) が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【記述例】

```
ADD #15, R2
ADD R1, R2
ADD [R1], R2
ADD [R1].UB, R2
ADD #127, R1, R2
ADD R1, R2, R3
```

AND

論理積 AND

AND

【構文】

- (1) AND src, dest
 (2) AND src, src2, dest

算術/論理演算命令

【命令コード】

記載ページ : 259

【オペレーション】

- (1) dest = dest & src;
 (2) dest = src2 & src;

【機能】

- (1) destとsrcの論理積をとり、その結果をdestに格納します。
 (2) src2とsrcの論理積をとり、その結果をdestに格納します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	演算後のdestが0のとき“1”、それ以外のとき“0”になります。
S	○	演算後のdestのMSBが“1”のとき“1”、それ以外のとき“0”になります。
O	—	

【命令フォーマット】

構文	処理サイズ	対象			コードサイズ (バイト)
		src	src2	dest	
(1) AND src, dest	L	#UIMM:4	—	Rd	2
	L	#SIMM:8	—	Rd	3
	L	#SIMM:16	—	Rd	4
	L	#SIMM:24	—	Rd	5
	L	#IMM:32	—	Rd	6
	L	Rs	—	Rd	2
	L	[Rs].memex	—	Rd	2 (memex == “UB”) 3 (memex != “UB”)
	L	dsp:8[Rs].memex (注)	—	Rd	3 (memex == “UB”) 4 (memex != “UB”)
L	dsp:16[Rs].memex (注)	—	Rd	4 (memex == “UB”) 5 (memex != “UB”)	
(2) AND src, src2, dest	L	Rs	Rs2	Rd	3

注. 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、サイズ拡張指定子が“.W”または“.UW”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:8には、サイズ拡張指定子が“.W”または“.UW”のとき0～510 (255×2) が、“.L”のとき0～1020 (255×4) が指定できます。dsp:16には、サイズ拡張指定子が“.W”または“.UW”のとき0～131070 (65535×2) が、“.L”のとき0～262140 (65535×4) が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【記述例】

AND #15, R2
 AND R1, R2
 AND [R1], R2
 AND [R1].UW, R2
 AND R1, R2, R3

BCLR

ビットクリア
Bit CLear

BCLR

【構文】

BCLR src, dest

ビット操作命令

【命令コード】

記載ページ : 261

【オペレーション】

- (1) destがメモリの場合
unsigned char dest;
dest &= ~(1 << (src & 7));
- (2) destがレジスタの場合
register unsigned long dest;
dest &= ~(1 << (src & 31));

【機能】

- srcで指定されたdestのビットを“0”にします。
- srcのIMMの値はビット番号です。
IMM:3の範囲は、 $0 \leq \text{IMM:3} \leq 7$ です。
IMM:5の範囲は、 $0 \leq \text{IMM:5} \leq 31$ です。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
(1) BCLR src, dest	B	#IMM:3	[Rd].B	2
	B	#IMM:3	dsp:8[Rd].B	3
	B	#IMM:3	dsp:16[Rd].B	4
	B	Rs	[Rd].B	3
	B	Rs	dsp:8[Rd].B	4
	B	Rs	dsp:16[Rd].B	5
(2) BCLR src, dest	L	#IMM:5	Rd	2
	L	Rs	Rd	3

【記述例】

```
BCLR #7, [R2]
BCLR R1, [R2]
BCLR #31, R2
BCLR R1, R2
```

BCnd

相対条件分岐
Branch Conditionally

BCnd

【構文】

```
BCnd(.length) src
```

分岐命令

【命令コード】

記載ページ：263

【オペレーション】

```
if ( Cnd )
    PC = PC + src;
```

【機能】

- Cndで示す条件の真偽値を判断し、srcで示される分岐先へ相対分岐します。真の場合は分岐しますが、偽の場合は分岐しません。
- BCndには以下の種類があります。

BCnd	条件		式	BCnd	条件		式
BGEU, BC	C == 1	等しいまたは大きい/ Cフラグが“1”	≤	BLTU, BNC	C == 0	小さい/ Cフラグが“0”	>
BEQ, BZ	Z == 1	等しい/ Zフラグが“1”	=	BNE, BNZ	Z == 0	等しくない/ Zフラグが“0”	≠
BGTU	(C & ~Z) == 1	大きい	<	BLEU	(C & ~Z) == 0	等しいまたは小さい	≥
BPZ	S == 0	正またゼロ	0 ≤	BN	S == 1	負	0 >
BGE	(S ^ O) == 0	等しい、または符号付き で大きい	≤	BLE	((S ^ O) Z) == 1	等しい、または符号付き で小さい	≥
BGT	((S ^ O) Z) == 0	符号付きで大きい	<	BLT	(S ^ O) == 1	符号付きで小さい	>
BO	O == 1	Oフラグが“1”		BNO	O == 0	Oフラグが“0”	

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	length	対象		コードサイズ (バイト)
		src	pcdspの範囲	
(1) BEQ.S src	S	pcdsp:3	3 ≤ pcdsp ≤ 10	1
(2) BNE.S src	S	pcdsp:3	3 ≤ pcdsp ≤ 10	1
(3) BCnd.B src	B	pcdsp:8	-128 ≤ pcdsp ≤ 127	2
(4) BEQ.W src	W	pcdsp:16	-32768 ≤ pcdsp ≤ 32767	3
(5) BNE.W src	W	pcdsp:16	-32768 ≤ pcdsp ≤ 32767	3

【記述例】

```
BC label1
BC.B label2
```

注． 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値（pcdsp:3, pcdsp:8, pcdsp:16）は、分岐先のラベルまたは実効アドレスを指定してください。命令コード（pcdsp）には、指定されたアドレスから命令の配置アドレスを引いた値が埋め込まれます。

【記述例】

```
BC label
BC 1000h
```

BFMOV

ビットフィールド転送
Bit Field MOVE

BFMOV

転送命令

【命令コード】

記載ページ：264

【構文】

BFMOV slsb, dlsb, width, src, dest

【オペレーション】

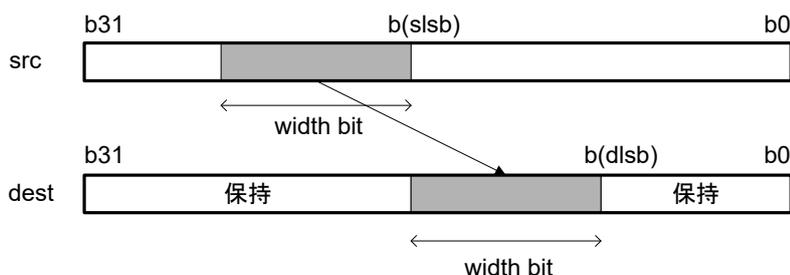
```

unsigned long tmp1, tmp2;
tmp1 = (0xFFFFFFFFh >> (32-width)) << dlsb;
tmp2 = (src >> slsb) << dlsb;
dest = (tmp2 & tmp1) | (dest & ~tmp1);

```

【機能】

- srcのビット位置slsbからwidthビット幅のデータをdestのビット位置dlsbへ転送します。destの残りのビットは保持されます。
- slsbの範囲は $0 \leq \text{slsb} \leq 31$ 、dlsbの範囲は $0 \leq \text{dlsb} \leq 31$ 、widthの範囲は $1 \leq \text{width} \leq 31$ です。
- $(\text{slsb} + \text{width}) > 32$ の場合、および $(\text{dlsb} + \text{width}) > 32$ の場合、destは不定になります。

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象					コードサイズ (バイト)
		slsb	dlsb	width	src	dest	
BFMOV slsb, dlsb, width, src, dest	L	#IMM:5	#IMM:5	#IMM:5	Rs	Rd	5

【記述例】

BFMOV #5, #10, #3, R1, R2

BFMOVZ

ゼロへのビットフィールド転送 Bit Field MOVE to Zero

BFMOVZ

【構文】

```
BFMOVZ slsb, dlsb, width, src, dest
```

転送命令

【命令コード】

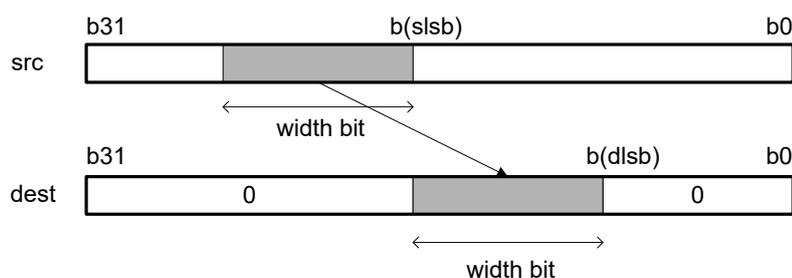
記載ページ：264

【オペレーション】

```
unsigned long tmp1, tmp2;
tmp1 = (0xFFFFFFFFh >> (32-width)) << dlsb;
tmp2 = (src >> slsb) << dlsb;
dest = (tmp2 & tmp1);
```

【機能】

- srcのビット位置slsbからwidthビット幅のデータをdestのビット位置dlsbへ転送します。destの残りのビットは0になります。
- slsbの範囲は $0 \leq \text{slsb} \leq 31$ 、dlsbの範囲は $0 \leq \text{dlsb} \leq 31$ 、widthの範囲は $1 \leq \text{width} \leq 31$ です。
- $(\text{slsb} + \text{width}) > 32$ の場合、および $(\text{dlsb} + \text{width}) > 32$ の場合、destは不定になります。



【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象					コードサイズ (バイト)
		slsb	dlsb	width	src	dest	
BFMOVZ slsb, dlsb, width, src, dest	L	#IMM:5	#IMM:5	#IMM:5	Rs	Rd	5

【記述例】

```
BFMOVZ #5, #10, #3, R1, R2
```

BM*Cnd*

条件ビット転送 Bit Move Conditionally

BM*Cnd*

【構文】

```
BMCnd src, dest
```

ビット操作命令

【命令コード】

記載ページ : 265

【オペレーション】

- (1) destがメモリの場合
- ```
unsigned char dest;
if (Cnd)
 dest |= (1 << (src & 7));
else
 dest &= ~(1 << (src & 7));
```
- (2) destがレジスタの場合
- ```
register unsigned long dest;
if ( Cnd )
    dest |= ( 1 << ( src & 31 ) );
else
    dest &= ~( 1 << ( src & 31 ) );
```

【機能】

- *Cnd*で示す条件の真偽値をsrcで指定されたdestのビットに転送します。真の場合は“1”、偽の場合は“0”が転送されます。
- BM*Cnd*には以下の種類があります。

BM <i>Cnd</i>	条件	式	BM <i>Cnd</i>	条件	式		
BMGEU, BMC	C == 1	等しいまたは大きい/ Cフラグが“1”	≤	BMLTU, BMNC	C == 0	小さい/ Cフラグが“0”	>
BMEQ, BMZ	Z == 1	等しい/ Zフラグが“1”	=	BMNE, BMNZ	Z == 0	等しくない/ Zフラグが“0”	≠
BMGTU	(C & ~Z) == 1	大きい	<	BMLEU	(C & ~Z) == 0	等しいまたは小さい	≥
BMPZ	S == 0	正またゼロ	0 ≤	BMN	S == 1	負	0 >
BMGE	(S ^ O) == 0	等しい、または符号付き で大きい	≤	BMLE	((S ^ O) Z) == 1	等しい、または符号付き で小さい	≥
BMGT	((S ^ O) Z) == 0	符号付きで大きい	<	BMLT	(S ^ O) == 1	符号付きで小さい	>
BMO	O == 1	Oフラグが“1”		BMNO	O == 0	Oフラグが“0”	

- srcのIMMの値はビット番号です。
IMM:3の範囲は、 $0 \leq \text{IMM:3} \leq 7$ です。
IMM:5の範囲は、 $0 \leq \text{IMM:5} \leq 31$ です。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
(1) <i>BMCnd</i> src, dest	B	#IMM:3	[Rd].B	3
	B	#IMM:3	dsp:8[Rd].B	4
	B	#IMM:3	dsp:16[Rd].B	5
(2) <i>BMCnd</i> src, dest	L	#IMM:5	Rd	3

【記述例】

BMC #7, [R2]

BMZ #31, R2

BNOT

ビット反転
Bit NOT

BNOT

【構文】

BNOT src, dest

ビット操作命令

【命令コード】

記載ページ : 266

【オペレーション】

- (1) destがメモリの場合
unsigned char dest;
dest ^= (1 << (src & 7));
- (2) destがレジスタの場合
register unsigned long dest;
dest ^= (1 << (src & 31));

【機能】

- srcで指定されたdestのビットの値を反転し、その結果を元のビットに格納します。
- srcのIMMの値はビット番号です。
IMM:3の範囲は、 $0 \leq \text{IMM:3} \leq 7$ です。
IMM:5の範囲は、 $0 \leq \text{IMM:5} \leq 31$ です。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
(1) BNOT src, dest	B	#IMM:3	[Rd].B	3
	B	#IMM:3	dsp:8[Rd].B	4
	B	#IMM:3	dsp:16[Rd].B	5
	B	Rs	[Rd].B	3
	B	Rs	dsp:8[Rd].B	4
	B	Rs	dsp:16[Rd].B	5
(2) BNOT src, dest	L	#IMM:5	Rd	3
	L	Rs	Rd	3

【記述例】

```
BNOT #7, [R2]
BNOT R1, [R2]
BNOT #31, R2
BNOT R1, R2
```

BRA

相対無条件分岐
BRanch Always

BRA

分岐命令

【命令コード】

記載ページ：267

【構文】

BRA(.length) src

【オペレーション】

PC = PC + src;

【機能】

- srcで示される分岐先に相対分岐します。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	length	対象		コードサイズ (バイト)
		src	pcdsp / Rsの範囲	
BRA(.length) src	S	pcdsp:3	$3 \leq \text{pcdsp} \leq 10$	1
	B	pcdsp:8	$-128 \leq \text{pcdsp} \leq 127$	2
	W	pcdsp:16	$-32768 \leq \text{pcdsp} \leq 32767$	3
	A	pcdsp:24	$-8388608 \leq \text{pcdsp} \leq 8388607$	4
	L	Rs	$-2147483648 \leq \text{Rs} \leq 2147483647$	2

【記述例】

```
BRA label1
BRA.A label2
BRA R1
BRA.L R2
```

注． 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値（pcdsp:3, pcdsp:8, pcdsp:16, pcdsp:24）は、分岐先のラベルまたは実効アドレスを指定してください。命令コード（pcdsp）には、指定されたアドレスから命令の配置アドレスを引いた値が埋め込まれます。

【記述例】

```
BRA label
BRA 1000h
```

BRK

無条件トラップ
BReaK

BRK

【構文】

BRK

システム操作命令

【命令コード】

記載ページ：268

【オペレーション】

```

tmp0 = PSW;
U = 0;
I = 0;
PM = 0;
tmp1 = PC + 1;
PC = *IntBase;
SP = SP - 4;
*SP = tmp0;
SP = SP - 4;
*SP = tmp1;

```

【機能】

- 番号0の無条件トラップが発生します。
- スーパーバイザモードに移行し、PSWのPMビットが“0”になります。
- PSWのU、Iビットが“0”になります。
- 実行したBRK命令の次の命令のアドレスがスタックに退避させられます。

【フラグ変化】

- フラグは変化しません。
- 命令実行前のPSWは、スタックに退避させられます。

【命令フォーマット】

構文	コードサイズ (バイト)
BRK	1

【記述例】

BRK

BSET

ビットセット Bit SET

BSET

【構文】

```
BSET src, dest
```

ビット操作命令

【命令コード】

記載ページ : 268

【オペレーション】

- (1) destがメモリの場合
unsigned char dest;
dest |= (1 << (src & 7));
- (2) destがレジスタの場合
register unsigned long dest;
dest |= (1 << (src & 31));

【機能】

- srcで指定されたdestのビットを“1”にします。
- srcのIMMの値はビット番号です。
IMM:3の範囲は、 $0 \leq \text{IMM:3} \leq 7$ です。
IMM:5の範囲は、 $0 \leq \text{IMM:5} \leq 31$ です。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
(1) BSET src, dest	B	#IMM:3	[Rd].B	2
	B	#IMM:3	dsp:8[Rd].B	3
	B	#IMM:3	dsp:16[Rd].B	4
	B	Rs	[Rd].B	3
	B	Rs	dsp:8[Rd].B	4
	B	Rs	dsp:16[Rd].B	5
(2) BSET src, dest	L	#IMM:5	Rd	2
	L	Rs	Rd	3

【記述例】

```
BSET #7, [R2]
BSET R1, [R2]
BSET #31, R2
BSET R1, R2
```

BSR

相対サブルーチン分岐
Branch to SubRoutine

BSR

【構文】

BSR(.length) src

分岐命令

【命令コード】

記載ページ：270

【オペレーション】

SP = SP - 4;

*SP = (PC + n); (注)

PC = PC + src;

- 注. 1. (PC + n) は、BSR 命令の次の命令の番地です。
2. n は、コードサイズです。コードサイズについては、【命令フォーマット】を参照してください。

【機能】

- src で示される分岐先に相対分岐します。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	length	対象		コードサイズ (バイト)
		src	pcdsp / Rsの範囲	
BSR(.length) src	W	pcdsp:16	$-32768 \leq \text{pcdsp} \leq 32767$	3
	A	pcdsp:24	$-8388608 \leq \text{pcdsp} \leq 8388607$	4
	L	Rs	$-2147483648 \leq \text{Rs} \leq 2147483647$	2

【記述例】

BSR label1

BSR.A label2

BSR R1

BSR.L R2

- 注. 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値 (pcdsp:16, pcdsp:24) は、分岐先のラベルまたは実効アドレスを指定してください。命令コード (pcdsp) には、指定されたアドレスから命令の配置アドレスを引いた値が埋め込まれます。

【記述例】

BSR label

BSR 1000h

BTST

ビットテスト Bit TeST

BTST

【構文】

```
BTST src, src2
```

ビット操作命令

【命令コード】

記載ページ : 271

【オペレーション】

(1) src2がメモリの場合

```
unsigned char src2;
```

```
Z = ~(( src2 >> ( src & 7 )) & 1 );
```

```
C = (( src2 >> ( src & 7 )) & 1 );
```

(2) src2がレジスタの場合

```
register unsigned long src2;
```

```
Z = ~(( src2 >> ( src & 31 )) & 1 );
```

```
C = (( src2 >> ( src & 31 )) & 1 );
```

【機能】

- srcで指定したsrc2のビットの値を反転した結果をZフラグに、srcで指定したsrc2のビットの値をCフラグに転送します。
- srcのIMMの値はビット番号です。
IMM:3の範囲は、 $0 \leq \text{IMM:3} \leq 7$ です。
IMM:5の範囲は、 $0 \leq \text{IMM:5} \leq 31$ です。

【フラグ変化】

フラグ	変化	条件
C	○	指定ビットが“1”のとき“1”、それ以外のとき“0”になります。
Z	○	指定ビットが“0”のとき“1”、それ以外のとき“0”になります。
S	—	
O	—	

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	src2	
(1) BTST src, src2	B	#IMM:3	[Rs].B	2
	B	#IMM:3	dsp:8[Rs].B	3
	B	#IMM:3	dsp:16[Rs].B	4
	B	Rs	[Rs2].B	3
	B	Rs	dsp:8[Rs2].B	4
	B	Rs	dsp:16[Rs2].B	5
(2) BTST src, src2	L	#IMM:5	Rs	2
	L	Rs	Rs2	3

【記述例】

```
BTST #7, [R2]
```

```
BTST R1, [R2]
```

```
BTST #31, R2
```

```
BTST R1, R2
```

CLRPSW

PSWのフラグ、ビットのクリア
CLear flag in PSW

CLRPSW

システム操作命令

【命令コード】

記載ページ：272

【構文】

CLRPSW dest

【オペレーション】

dest = 0;

【機能】

- destで指定されたO、S、Z、Cフラグ、もしくはU、Iビットを“0”にします。
- ユーザモードでは、U、Iビットへの書き込みは無視されます。スーパーバイザモードでは、すべてのフラグとビットへの書き込みが行えます。

【フラグ変化】

フラグ	変化	条件
C	(注)	
Z	(注)	
S	(注)	
O	(注)	

注. 指定されたフラグが“0”になります。

【命令フォーマット】

構文	対象	コードサイズ (バイト)
	dest	
CLRPSW dest	flag	2

【記述例】

CLRPSW C

CLRPSW Z

CMP

比較 CoMPare

CMP

【構文】

CMP src, src2

算術/論理演算命令

【命令コード】

記載ページ：273

【オペレーション】

src2 - src;

【機能】

- src2からsrcを減算した結果に従って、PSWの各フラグが変化します。

【フラグ変化】

フラグ	変化	条件
C	○	符号なし演算のオーバーフローが発生しなかったとき“1”、それ以外るとき“0”になります。
Z	○	演算結果が0のとき“1”、それ以外るとき“0”になります。
S	○	演算結果のMSBが“1”のとき“1”、それ以外るとき“0”になります。
O	○	符号付き演算のオーバーフローが発生したとき“1”、それ以外るとき“0”になります。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	src2	
CMP src, src2	L	#UIMM:4	Rs	2
	L	#UIMM:8 (注1)	Rs	3
	L	#SIMM:8 (注1)	Rs	3
	L	#SIMM:16	Rs	4
	L	#SIMM:24	Rs	5
	L	#IMM:32	Rs	6
	L	Rs	Rs2	2
	L	[Rs].memex	Rs2	2 (memex == “UB”) 3 (memex != “UB”)
	L	dsp:8[Rs].memex (注2)	Rs2	3 (memex == “UB”) 4 (memex != “UB”)
	L	dsp:16[Rs].memex (注2)	Rs2	4 (memex == “UB”) 5 (memex != “UB”)

注1. 0～127の範囲は、ゼロ拡張命令コードになります。

注2. 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、サイズ拡張指定子が“.W”または“.UW”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:8には、サイズ拡張指定子が“.W”または“.UW”のとき0～510 (255×2) が、“.L”のとき0～1020 (255×4) が指定できます。dsp:16には、サイズ拡張指定子が“.W”または“.UW”のとき0～131070 (65535×2) が、“.L”のとき0～262140 (65535×4) が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【記述例】

CMP #7, R2

CMP R1, R2

CMP [R1], R2

DIV

符号付き除算 DIVide

DIV

【構文】

DIV src, dest

算術/論理演算命令

【命令コード】

記載ページ：274

【オペレーション】

dest = dest / src;

【機能】

- destをsrcで符号付き除算し、その商をdestに格納します。商は0方向に丸められます。
- 演算は32ビットで行い、結果は32ビットで格納します。
- 除数（src）が0のとき、または演算の結果、オーバフローが発生したときのdestの値は不定です。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	—	
O	○	除数（src）が0のとき、または演算が $-2147483648 \div (-1)$ のとき“1”、それ以外のとき“0”になります。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
DIV src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == “UB”) 4 (memex != “UB”)
	L	dsp:8[Rs].memex (注)	Rd	4 (memex == “UB”) 5 (memex != “UB”)
	L	dsp:16[Rs].memex (注)	Rd	5 (memex == “UB”) 6 (memex != “UB”)

注. 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値（dsp:8, dsp:16）は、サイズ拡張指定子が“.W”または“.UW”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:8には、サイズ拡張指定子が“.W”または“.UW”のとき0～510（255×2）が、“.L”のとき0～1020（255×4）が指定できます。dsp:16には、サイズ拡張指定子が“.W”または“.UW”のとき0～131070（65535×2）が、“.L”のとき0～262140（65535×4）が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【記述例】

DIV #10, R2

DIV R1, R2

DIV [R1], R2

DIV 3[R1].B, R2

DIVU

符号なし除算
DIVide Unsigned

DIVU

【構文】

DIVU src, dest

算術/論理演算命令

【命令コード】

記載ページ : 276

【オペレーション】

dest = dest / src;

【機能】

- destをsrcで符号なし除算し、その商をdestに格納します。商は0方向に丸められます。
- 演算は32ビットで行い、結果は32ビットで格納します。
- 除数 (src) が0のときのdestの値は不定です。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	—	
O	○	除数 (src) が0のとき“1”、それ以外るとき“0”になります。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
DIVU src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == “UB”) 4 (memex != “UB”)
	L	dsp:8[Rs].memex (注)	Rd	4 (memex == “UB”) 5 (memex != “UB”)
	L	dsp:16[Rs].memex (注)	Rd	5 (memex == “UB”) 6 (memex != “UB”)

注. 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、サイズ拡張指定子が“.W”または“.UW”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:8には、サイズ拡張指定子が“.W”または“.UW”のとき0～510 (255 × 2) が、“.L”のとき0～1020 (255 × 4) が指定できます。dsp:16には、サイズ拡張指定子が“.W”または“.UW”のとき0～131070 (65535 × 2) が、“.L”のとき0～262140 (65535 × 4) が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【記述例】

```

DIVU #10, R2
DIVU R1, R2
DIVU [R1], R2
DIVU 3[R1].UB, R2

```

EMACA

32ビット積和演算
Extend Multiply-ACcumulate to Accumulator

EMACA

【構文】

```
EMACA src, src2, Adest
```

DSP機能命令

【命令コード】

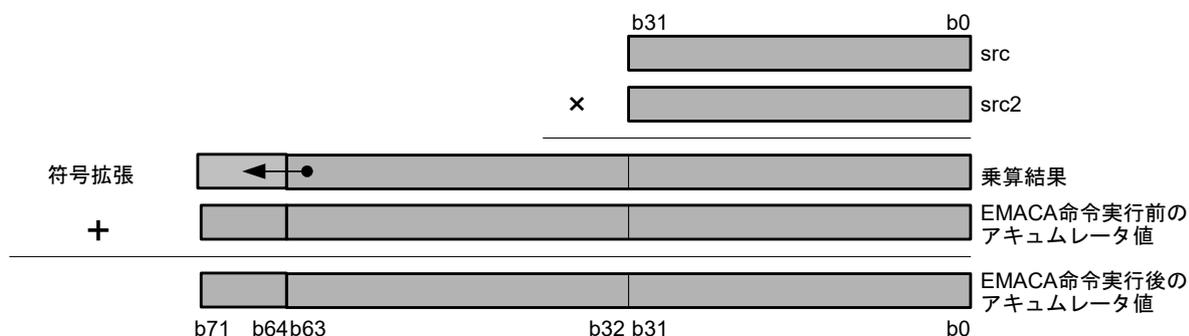
記載ページ：277

【オペレーション】

```
signed 72bit tmp;
tmp = (signed long) src * (signed long) src2;
Adest = Adest + tmp;
```

【機能】

- srcとsrc2の乗算を行い、乗算結果とアキュムレータの加算を行います。加算結果はアキュムレータに格納されます。srcとsrc2は符号付き整数として扱われます。

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	対象			コードサイズ (バイト)
	src	src2	Adest	
EMACA src, src2, Adest	Rs	Rs2	A0, A1	3

【記述例】

```
EMACA R1, R2, A1
```

EMSBA

32ビット積差演算
Extended Multiply-Subtract to Accumulator

EMSBA

【構文】

```
EMSBA src, src2, Adest
```

DSP機能命令

【命令コード】

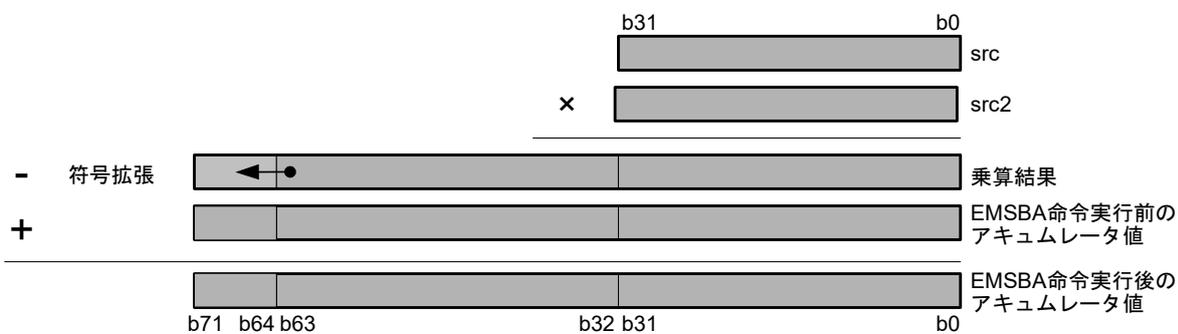
記載ページ：277

【オペレーション】

```
signed 72bit tmp;
tmp = (signed long) src * (signed long) src2;
Adest = Adest - tmp;
```

【機能】

- srcとsrc2の乗算を行い、乗算結果をアキュムレータから減算します。減算結果はアキュムレータに格納されます。srcとsrc2は符号付き整数として扱われます。

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	対象			コードサイズ (バイト)
	src	src2	Adest	
EMSBA src, src2, Adest	Rs	Rs2	A0, A1	3

【記述例】

```
EMSBA R1, R2, A1
```

EMUL

符号付き乗算
Extended MULtiply, signed

EMUL

【構文】

```
EMUL src, dest
```

算術/論理演算命令

【命令コード】

記載ページ : 278

【オペレーション】

```
dest2:dest = dest * src;
```

【機能】

- destをsrcで符号付き乗算します。
- src、destとも32ビットで演算し、結果を64ビットでレジスタペアdest2:dest (R(n+1):Rn)に格納します。
- destにはRn (n = 0~14)の15種類が指定できます。

注. アキュムレータ (ACC0) を使用します。命令実行後のACC0の値は不定です。

destで指定するレジスタ	64ビット拡張で使用されるレジスタ
R0	R1:R0
R1	R2:R1
R2	R3:R2
R3	R4:R3
R4	R5:R4
R5	R6:R5
R6	R7:R6
R7	R8:R7
R8	R9:R8
R9	R10:R9
R10	R11:R10
R11	R12:R11
R12	R13:R12
R13	R14:R13
R14	R15:R14

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
EMUL src, dest	L	#SIMM:8	Rd (Rd = R0~R14)	4
	L	#SIMM:16	Rd (Rd = R0~R14)	5
	L	#SIMM:24	Rd (Rd = R0~R14)	6
	L	#IMM:32	Rd (Rd = R0~R14)	7
	L	Rs	Rd (Rd = R0~R14)	3
	L	[Rs].memex	Rd (Rd = R0~R14)	3 (memex == "UB") 4 (memex != "UB")
	L	dsp:8[Rs].memex (注)	Rd (Rd = R0~R14)	4 (memex == "UB") 5 (memex != "UB")
	L	dsp:16[Rs].memex (注)	Rd (Rd = R0~R14)	5 (memex == "UB") 6 (memex != "UB")

注. 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、サイズ拡張指定子が“.W”または“.UW”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:8には、サイズ拡張指定子が“.W”または“.UW”のとき0~510 (255 × 2) が、“.L”のとき0~1020 (255 × 4) が指定できます。dsp:16には、サイズ拡張指定子が“.W”または“.UW”のとき0~131070 (65535 × 2) が、“.L”のとき0~262140 (65535 × 4) が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【記述例】

```
EMUL #10, R2
EMUL R1, R2
EMUL [R1], R2
EMUL 8[R1].W, R2
```

EMULA

32ビット乗算
Extended MULTIply to Accumulator

EMULA

【構文】

EMULA src, src2, Adest

DSP機能命令

【命令コード】

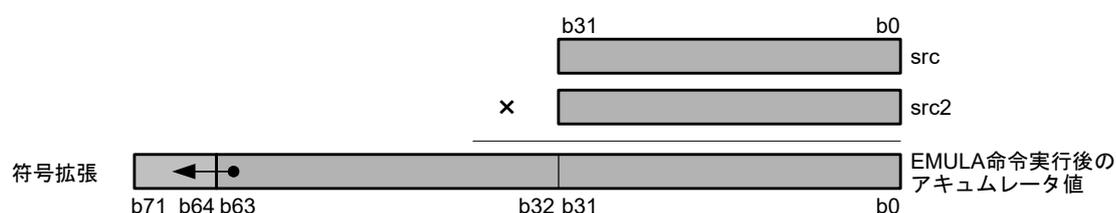
記載ページ：279

【オペレーション】

Adest = (signed long) src * (signed long) src2;

【機能】

- srcとsrc2の乗算を行い、その結果をアキュムレータに格納します。srcとsrc2は符号付き整数として扱われます。

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	対象			コードサイズ (バイト)
	src	src2	Adest	
EMULA src, src2, Adest	Rs	Rs2	A0, A1	3

【記述例】

EMULA R1, R2, A1

EMULU

符号なし乗算
Extended MULTiply, Unsigned

EMULU

算術/論理演算命令

【命令コード】

記載ページ：279

【構文】

EMULU src, dest

【オペレーション】

dest2:dest = dest * src;

【機能】

- destをsrcで符号なし乗算します。
- src, destとも32ビットで演算し、結果を64ビットでレジスタペアdest2:dest (R(n+1):Rn)に格納します。
- destにはRn (n = 0~14)の15種類が指定できます。

注. アキュムレータ (ACC0) を使用します。命令実行後のACC0の値は不定です。

destで指定するレジスタ	64ビット拡張で使用されるレジスタ
R0	R1:R0
R1	R2:R1
R2	R3:R2
R3	R4:R3
R4	R5:R4
R5	R6:R5
R6	R7:R6
R7	R8:R7
R8	R9:R8
R9	R10:R9
R10	R11:R10
R11	R12:R11
R12	R13:R12
R13	R14:R13
R14	R15:R14

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
EMULU src, dest	L	#SIMM:8	Rd (Rd = R0~R14)	4
	L	#SIMM:16	Rd (Rd = R0~R14)	5
	L	#SIMM:24	Rd (Rd = R0~R14)	6
	L	#IMM:32	Rd (Rd = R0~R14)	7
	L	Rs	Rd (Rd = R0~R14)	3
	L	[Rs].memex	Rd (Rd = R0~R14)	3 (memex == "UB") 4 (memex != "UB")
	L	dsp:8[Rs].memex (注)	Rd (Rd = R0~R14)	4 (memex == "UB") 5 (memex != "UB")
	L	dsp:16[Rs].memex (注)	Rd (Rd = R0~R14)	5 (memex == "UB") 6 (memex != "UB")

注. 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、サイズ拡張指定子が“.W”または“.UW”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:8には、サイズ拡張指定子が“.W”または“.UW”のとき0~510 (255 × 2) が、“.L”のとき0~1020 (255 × 4) が指定できます。dsp:16には、サイズ拡張指定子が“.W”または“.UW”のとき0~131070 (65535 × 2) が、“.L”のとき0~262140 (65535 × 4) が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【記述例】

```
EMULU #10, R2
EMULU R1, R2
EMULU [R1], R2
EMULU 8[R1].UW, R2
```

FADD

単精度浮動小数点加算 Floating-point ADD

FADD

【構文】

- (1) FADD src, dest
- (2) FADD src, src2, dest

単精度浮動小数点演算命令

【命令コード】

記載ページ : 281

【オペレーション】

- (1) dest = dest + src;
- (2) dest = src2 + src;

【機能】

- (1) destに格納された単精度浮動小数点数と、srcに格納された単精度浮動小数点数を加算し、その結果をdestに格納します。
- (2) src2に格納された単精度浮動小数点数と、srcに格納された単精度浮動小数点数を加算し、その結果をdestに格納します。
 - 結果はFPSWのRM[1:0]ビットに従って丸められます。
 - 非正規化数の扱いは、FPSWのDNビットによって変化します。
 - 反対の符号を持つ(src, dest)、(src, src2)の和が正確に0であるときは、 $-\infty$ 方向への丸めモードの場合を除いて、結果は+0になります。 $-\infty$ 方向への丸めモードの場合は、結果は-0になります。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	演算の結果が"+0"または"-0"のとき"1"、それ以外るとき"0"になります。
S	○	演算の結果、符号部（ビット31）が"1"のとき"1"、"0"のとき"0"になります。
O	—	
CV	○	無効演算が発生したとき"1"、それ以外るとき"0"になります。
CO	○	オーバフローが発生したとき"1"、それ以外るとき"0"になります。
CZ	○	"0"になります。
CU	○	アンダフローが発生したとき"1"、それ以外るとき"0"になります。
CX	○	精度異常が発生したとき"1"、それ以外るとき"0"になります。
CE	○	非実装処理が発生したとき"1"、それ以外るとき"0"になります。
FV	○	無効演算が発生したとき"1"、それ以外るときは変化しません。
FO	○	オーバフローが発生したとき"1"、それ以外るときは変化しません。
FZ	—	
FU	○	アンダフローが発生したとき"1"、それ以外るときは変化しません。
FX	○	精度異常が発生したとき"1"、それ以外るときは変化しません。

注. FX, FU, FO, FVフラグは、例外処理許可ビットEX, EU, EO, EVが"1"の場合は変化しません。S, Zフラグは、例外処理が発生した場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象			コードサイズ (バイト)
		src	src2	dest	
(1) FADD src, dest	L	#IMM:32	—	Rd	7
	L	Rs	—	Rd	3
	L	[Rs].L	—	Rd	3
	L	dsp:8[Rs].L (注)	—	Rd	4
	L	dsp:16[Rs].L (注)	—	Rd	5
(2) FADD src, src2, dest	L	Rs	Rs2	Rd	3

注. 弊社の「RXファミリ アセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、4の倍数を指定してください。dsp:8には、0~1020 (255 × 4) が指定できます。dsp:16には、0~262140 (65535 × 4) が指定できます。命令コードには、1/4した値が埋め込まれます。

【単精度浮動小数点例外発生要因】

- 非実装処理
- 無効演算
- オーバフロー
- アンダフロー
- 精度異常

【記述例】

FADD R1, R2
 FADD [R1], R2
 FADD R1, R2, R3

【動作補足説明】

- FPSWのDNビットが“0”の場合と“1”の場合について、src、src2、destの値と演算結果の対応を以下に示します。

DN = 0のとき

		src							
		正規化数	+0	-0	+∞	-∞	非正規化数	QNaN	SNaN
dest または src2	正規化数	加算							
	+0		+0	(注)		-∞			
	-0		(注)	-0					
	+∞				+∞	無効演算			
	-∞		-∞		無効演算	-∞			
	非正規化数	非実装処理							
	QNaN							QNaN	
SNaN								無効演算	

DN = 1のとき

		src								
		正規化数	+0、 +非正規化数	-0、 -非正規化数	+∞	-∞	QNaN	SNaN		
dest または src2	正規化数	加算	正規化数		+∞	-∞	QNaN	SNaN		
	+0、+非正規化数	正規化数	+0	(注)						
	-0、-非正規化数		(注)	-0						
	+∞			+∞	無効演算					
	-∞	-∞		無効演算	-∞					
	QNaN								QNaN	
	SNaN									無効演算

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

単精度浮動小数点比較 Floating-point CoMPare

FCMP

【構文】

FCMP src, src2

単精度浮動小数点演算命令

【命令コード】

記載ページ：282

【オペレーション】

src2 - src;

【機能】

- src2に格納された単精度浮動小数点数と、srcに格納された単精度浮動小数点数を比較し、その結果に従ってフラグが変化します。
- 非正規化数の扱いは、FPSWのDNビットによって変化します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	src2 == srcのとき“1”、それ以外のとき“0”になります。
S	○	src2 < srcのとき“1”、それ以外のとき“0”になります。
O	○	比較結果が順序化不能のとき“1”、それ以外のとき“0”になります。
CV	○	無効演算が発生したとき“1”、それ以外のとき“0”になります。
CO	○	“0”になります。
CZ	○	“0”になります。
CU	○	“0”になります。
CX	○	“0”になります。
CE	○	非実装処理が発生したとき“1”、それ以外のとき“0”になります。
FV	○	無効演算が発生したとき“1”、それ以外のときは変化しません。
FO	—	
FZ	—	
FU	—	
FX	—	

注. FVフラグは、例外処理許可ビットEVが“1”の場合は変化しません。O、S、Zフラグは、例外処理が発生した場合は変化しません。

条件 \ フラグ	O	S	Z
src2 > src	“0”	“0”	“0”
src2 < src	“0”	“1”	“0”
src2 == src	“0”	“0”	“1”
順序化不能	“1”	“0”	“0”

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	src2	
FCMP src, src2	L	#IMM:32	Rs	7
	L	Rs	Rs2	3
	L	[Rs].L	Rs2	3
	L	dsp:8[Rs].L (注)	Rs2	4
	L	dsp:16[Rs].L (注)	Rs2	5

注. 弊社の「RXファミリ アセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、4の倍数を指定してください。dsp:8には、0～1020 (255×4) が指定できます。dsp:16には、0～262140 (65535×4) が指定できます。命令コードには、1/4した値が埋め込まれます。

【単精度浮動小数点例外発生要因】

非実装処理
無効演算

【記述例】

FCMP R1, R2
FCMP [R1], R2

【動作補足説明】

- FPSWのDNビットが“0”の場合と“1”の場合について、src、src2の値と演算結果の対応を以下に示します。
(> : src2 > src、< : src2 < src、= : src2 == src)

DN = 0のとき

		src							
		正規化数	+0	-0	+∞	-∞	非正規化数	QNaN	SNaN
src2	正規化数	比較			<	>	非実装処理	順序化不能	無効演算 (順序化不能)
	+0	=							
	-0	=							
	+∞	>		=					
	-∞	<		=					
	非正規化数								
	QNaN								
SNaN									

DN = 1のとき

		src							
		正規化数	+0、 +非正規化数	-0、 -非正規化数	+∞	-∞	QNaN	SNaN	
src2	正規化数	比較			<	>	非実装処理	順序化不能	無効演算 (順序化不能)
	+0、+非正規化数	=							
	-0、-非正規化数	=							
	+∞	>		=					
	-∞	<		=					
	QNaN								
	SNaN								

FDIV

単精度浮動小数点除算 Floating-point Divide

FDIV

【構文】

FDIV src, dest

単精度浮動小数点演算命令

【命令コード】

記載ページ：283

【オペレーション】

dest = dest / src;

【機能】

- destに格納された単精度浮動小数点数を、srcに格納された単精度浮動小数点数で除算し、その結果をdestに格納します。結果はFPSWのRM[1:0]ビットに従って丸められます。
- 非正規化数の扱いは、FPSWのDNビットによって変化します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	演算の結果が“+0”または“-0”のとき“1”、それ以外のとき“0”になります。
S	○	演算の結果、符号部（ビット31）が“1”のとき“1”、“0”のとき“0”になります。
O	—	
CV	○	無効演算が発生したとき“1”、それ以外のとき“0”になります。
CO	○	オーバフローが発生したとき“1”、それ以外のとき“0”になります。
CZ	○	ゼロ除算が発生したとき“1”、それ以外のとき“0”になります。
CU	○	アンダフローが発生したとき“1”、それ以外のとき“0”になります。
CX	○	精度異常が発生したとき“1”、それ以外のとき“0”になります。
CE	○	非実装処理が発生したとき“1”、それ以外のとき“0”になります。
FV	○	無効演算が発生したとき“1”、それ以外のときは変化しません。
FO	○	オーバフローが発生したとき“1”、それ以外のときは変化しません。
FZ	○	ゼロ除算が発生したとき“1”、それ以外のときは変化しません。
FU	○	アンダフローが発生したとき“1”、それ以外のときは変化しません。
FX	○	精度異常が発生したとき“1”、それ以外のときは変化しません。

注. FX、FU、FZ、FO、FVフラグは、例外処理許可ビットEX、EU、EZ、EO、EVが“1”の場合は変化しません。S、Zフラグは、例外処理が発生した場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
FDIV src, dest	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].L	Rd	3
	L	dsp:8[Rs].L (注)	Rd	4
	L	dsp:16[Rs].L (注)	Rd	5

注. 弊社の「RXファミリ アセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、4の倍数を指定してください。dsp:8には、0～1020 (255×4) が指定できます。dsp:16には、0～262140 (65535×4) が指定できます。命令コードには、1/4した値が埋め込まれます。

【単精度浮動小数点例外発生要因】

- 非実装処理
- 無効演算
- オーバフロー
- アンダフロー
- 精度異常
- ゼロ除算

【記述例】

FDIV R1, R2
 FDIV [R1], R2

【動作補足説明】

- FPSWのDNビットが“0”の場合と“1”の場合について、src、destの値と演算結果の対応を以下に示します。

DN = 0のとき

		src									
		正規化数	+0	-0	+∞	-∞	非正規化数	QNaN	SNaN		
dest	正規化数	除算	ゼロ除算		0		非実装処理	QNaN	無効演算		
	+0	0	無効演算		+0	-0					
	-0				-0	+0					
	+∞	∞	+∞	-∞	無効演算						
	-∞		-∞	+∞							
	非正規化数	非実装処理									
	QNaN	QNaN									
SNaN	無効演算										

DN = 1のとき

		src									
		正規化数	+0、 +非正規化数	-0、 -非正規化数	+∞	-∞	QNaN	SNaN			
dest	正規化数	除算	ゼロ除算		0		非実装処理	QNaN	無効演算		
	+0、+非正規化数	0	無効演算		+0	-0					
	-0、-非正規化数				-0	+0					
	+∞	∞	+∞	-∞	無効演算						
	-∞		-∞	+∞							
	QNaN	QNaN									
	SNaN	無効演算									

FMUL

単精度浮動小数点乗算 Floating-point MULtiPLY

FMUL

【構文】

- (1) FMUL src, dest
- (2) FMUL src, src2, dest

単精度浮動小数点演算命令

【命令コード】

記載ページ：284

【オペレーション】

- (1) dest = dest * src;
- (2) dest = src2 * src;

【機能】

- (1) destに格納された単精度浮動小数点数と、srcに格納された単精度浮動小数点数を乗算し、その結果をdestに格納します。
- (2) src2に格納された単精度浮動小数点数と、srcに格納された単精度浮動小数点数を乗算し、その結果をdestに格納します。
 - 結果はFPSWのRM[1:0]ビットに従って丸められます。
 - 非正規化数の扱いは、FPSWのDNビットによって変化します。

注． 単精度浮動小数点例外発生有無にかかわらず、命令実行後のACC0の値は不定です。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	演算の結果が"+0"または"-0"のとき"1"、それ以外のとき"0"になります。
S	○	演算の結果、符号部（ビット31）が"1"のとき"1"、"0"のとき"0"になります。
O	—	
CV	○	無効演算が発生したとき"1"、それ以外のとき"0"になります。
CO	○	オーバフローが発生したとき"1"、それ以外のとき"0"になります。
CZ	○	"0"になります。
CU	○	アンダフローが発生したとき"1"、それ以外のとき"0"になります。
CX	○	精度異常が発生したとき"1"、それ以外のとき"0"になります。
CE	○	非実装処理が発生したとき"1"、それ以外のとき"0"になります。
FV	○	無効演算が発生したとき"1"、それ以外のときは変化しません。
FO	○	オーバフローが発生したとき"1"、それ以外のときは変化しません。
FZ	—	
FU	○	アンダフローが発生したとき"1"、それ以外のときは変化しません。
FX	○	精度異常が発生したとき"1"、それ以外のときは変化しません。

注． FX、FU、FO、FVフラグは、例外処理許可ビットEX、EU、EO、EVが"1"の場合は変化しません。S、Zフラグは、例外処理が発生した場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象			コードサイズ (バイト)
		src	src2	dest	
(1) FMUL src, dest	L	#IMM:32	—	Rd	7
	L	Rs	—	Rd	3
	L	[Rs].L	—	Rd	3
	L	dsp:8[Rs].L (注)	—	Rd	4
	L	dsp:16[Rs].L (注)	—	Rd	5
(2) FMUL src, src2, dest	L	Rs	Rs2	Rd	3

注. 弊社の「RXファミリ アセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、4の倍数を指定してください。dsp:8には、0～1020 (255×4) が指定できます。dsp:16には、0～262140 (65535×4) が指定できます。命令コードには、1/4した値が埋め込まれます。

【単精度浮動小数点例外発生要因】

非実装処理
無効演算
オーバフロー
アンダフロー
精度異常

【記述例】

FMUL R1, R2
FMUL [R1], R2
FMUL R1, R2, R3

【動作補足説明】

- FPSWのDNビットが“0”の場合と“1”の場合について、src、src2、destの値と演算結果の対応を以下に示します。

DN = 0のとき

		src						非正規化数	QNaN	SNaN		
		正規化数	+0	-0	+∞	-∞						
dest または src2	正規化数	乗算	∞		無効演算		非実装処理	QNaN	無効演算			
	+0		+0	-0	無効演算							
	-0		-0	+0	無効演算							
	+∞	∞	無効演算		+∞	-∞						
	-∞		無効演算		-∞	+∞						
	非正規化数	非実装処理								QNaN	無効演算	
	QNaN	非実装処理										
SNaN	非実装処理											

DN = 1のとき

		src								
		正規化数	+0、 +非正規化数	-0、 -非正規化数	+∞	-∞	QNaN	SNaN		
dest または src2	正規化数	乗算	∞		QNaN			無効演算		
	+0、+非正規化数		+0	-0					無効演算	
	-0、-非正規化数		-0	+0						
	+∞	∞	無効演算		+∞	-∞				
	-∞		-∞	+∞						
	QNaN						QNaN			
	SNaN									

FSQRT

単精度浮動小数点平方根
Floating-point Square Root

FSQRT

【構文】

```
FSQRT src, dest
```

単精度浮動小数点演算命令

【命令コード】

記載ページ：285

【オペレーション】

```
dest = sqrt(src);
```

【機能】

- srcに格納された単精度浮動小数点数の平方根を求め、その結果をdestに格納します。結果はFPSWのRM[1:0]ビットに従って丸められます。
- 非正規化数の扱いは、FPSWのDNビットによって変化します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	演算の結果が"+0"または"-0"のとき"1"、それ以外のとき"0"になります。
S	○	演算の結果、符号部（ビット31）が"1"のとき"1"、"0"のとき"0"になります。
O	—	
CV	○	無効演算が発生したとき"1"、それ以外のとき"0"になります。
CO	○	"0"になります。
CZ	○	"0"になります。
CU	○	"0"になります。
CX	○	精度異常が発生したとき"1"、それ以外のとき"0"になります。
CE	○	非実装処理が発生したとき"1"、それ以外のとき"0"になります。
FV	○	無効演算が発生したとき"1"、それ以外のときは変化しません。
FO	—	
FZ	—	
FU	—	
FX	○	精度異常が発生したとき"1"、それ以外のときは変化しません。

注. FX、FVフラグは、例外処理許可ビットEX、EVが"1"の場合は変化しません。S、Zフラグは、例外処理が発生した場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
FSQRT src, dest	L	Rs	Rd	3
	L	[Rs].L	Rd	3
	L	dsp:8[Rs].L (注)	Rd	4
	L	dsp:16[Rs].L (注)	Rd	5

注. 弊社の「RXファミリ アセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、4の倍数を指定してください。dsp:8には、0～1020 (255×4) が指定できます。dsp:16には、0～262140 (65535×4) が指定できます。命令コードには、1/4した値が埋め込まれます。

【単精度浮動小数点例外発生要因】

非実装処理
無効演算
精度異常

【記述例】

FSQRT R1, R2
 FSQRT [R1], R2

【動作補足説明】

- FPSWのDNビットが“0”の場合と“1”の場合について、srcの値と演算結果の対応を以下に示します。

DN = 0のとき

	src								
	+ 正規化数	- 正規化数	+0	-0	+ ∞	- ∞	非正規化数	QNaN	SNaN
結果	平方根	無効演算	+0	-0	+ ∞	無効演算	非実装処理	QNaN	無効演算

DN = 1のとき

	src									
	+ 正規化数	- 正規化数	+0	-0	+ ∞	- ∞	+ 非正規化数	- 非正規化数	QNaN	SNaN
結果	平方根	無効演算	+0	-0	+ ∞	無効演算	+0	-0	QNaN	無効演算

【無効演算発生時のQNaN生成規則】

ソースオペランド	演算結果
SNaN	QNaN化されたSNaNソースオペランド
上記以外	7FFFFFFFh

注. 「表 1.7 QNaN生成規則」に対応

FSUB

単精度浮動小数点減算 Floating-point SUBtract

FSUB

【構文】

- (1) FSUB src, dest
- (2) FSUB src, src2, dest

単精度浮動小数点演算命令

【命令コード】

記載ページ : 286

【オペレーション】

- (1) dest = dest - src;
- (2) dest = src2 - src;

【機能】

- (1) destに格納された単精度浮動小数点数から、srcに格納された単精度浮動小数点数を減算し、その結果をdestに格納します。
 - (2) src2に格納された単精度浮動小数点数から、srcに格納された単精度浮動小数点数を減算し、その結果をdestに格納します。
- 結果はFPSWのRM[1:0]ビットに従って丸められます。
 - 非正規化数の扱いは、FPSWのDNビットによって変化します。
 - 同一の符号を持つsrc、dest（もしくはsrc2）の差が正確に0であるときは、 $-\infty$ 方向への丸めモードの場合を除いて、結果は+0になります。 $-\infty$ 方向への丸めモードの場合は、結果は-0になります。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	演算の結果が"+0"または"-0"のとき"1"、それ以外のとき"0"になります。
S	○	演算の結果、符号部(ビット31)が"1"のとき"1"、"0"のとき"0"になります。
O	—	
CV	○	無効演算が発生したとき"1"、それ以外のとき"0"になります。
CO	○	オーバフローが発生したとき"1"、それ以外のとき"0"になります。
CZ	○	"0"になります。
CU	○	アンダフローが発生したとき"1"、それ以外のとき"0"になります。
CX	○	精度異常が発生したとき"1"、それ以外のとき"0"になります。
CE	○	非実装処理が発生したとき"1"、それ以外のとき"0"になります。
FV	○	無効演算が発生したとき"1"、それ以外のときは変化しません。
FO	○	オーバフローが発生したとき"1"、それ以外のときは変化しません。
FZ	—	
FU	○	アンダフローが発生したとき"1"、それ以外のときは変化しません。
FX	○	精度異常が発生したとき"1"、それ以外のときは変化しません。

注. FX、FU、FO、FVフラグは、例外処理許可ビットEX、EU、EO、EVが"1"の場合は変化しません。S、Zフラグは、例外処理が発生した場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象			コードサイズ (バイト)
		src	src2	dest	
(1) FSUB src, dest	L	#IMM:32	—	Rd	7
	L	Rs	—	Rd	3
	L	[Rs].L	—	Rd	3
	L	dsp:8[Rs].L (注)	—	Rd	4
	L	dsp:16[Rs].L (注)	—	Rd	5
(2) FSUB src, src2, dest	L	Rs	Rs2	Rd	3

注. 弊社の「RXファミリ アセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、4の倍数を指定してください。dsp:8には、0～1020 (255×4) が指定できます。dsp:16には、0～262140 (65535×4) が指定できます。命令コードには、1/4した値が埋め込まれます。

【単精度浮動小数点例外発生要因】

非実装処理
無効演算
オーバフロー
アンダフロー
精度異常

【記述例】

FSUB R1, R2
FSUB [R1], R2
FSUB R1, R2, R3

【動作補足説明】

- FPSWのDNビットが“0”の場合と“1”の場合について、src、destの値と演算結果の対応を以下に示します。

DN = 0のとき

		src							
		正規化数	+0	-0	+∞	-∞	非正規化数	QNaN	SNaN
dest	正規化数	減算			-∞	+∞	非実装処理	QNaN	SNaN
	+0		(注)	+0					
	-0		-0	(注)					
	+∞		+∞		無効演算				
	-∞	-∞		無効演算					
	非正規化数	非実装処理							
QNaN	QNaN								
SNaN	無効演算								

DN = 1のとき

		src							
		正規化数	+0、 +非正規化数	-0、 -非正規化数	+∞	-∞	QNaN	SNaN	
dest	正規化数	減算			-∞	+∞	QNaN	無効演算	
	+0、+非正規化数		(注)	+0					
	-0、-非正規化数		-0	(注)					
	+∞	+∞		無効演算					
	-∞	-∞		無効演算					
	QNaN	QNaN							
	SNaN	無効演算							

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単精度浮動小数点数→符号付き整数変換
Float TO Integer**FTOI**

【構文】

FTOI src, dest

単精度浮動小数点演算命令

【命令コード】

記載ページ：287

【オペレーション】

dest = (signed long) src;

【機能】

- srcに格納された単精度浮動小数点数を符号付きロングワード（32ビット）整数に変換し、その結果をdestに格納します。
- 結果はFPSWのRM[1:0]ビットに関係なく、0方向に丸められます。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	演算の結果が“0”のとき“1”、それ以外のとき“0”になります。
S	○	演算の結果、符号部（ビット31）が“1”のとき“1”、“0”のとき“0”になります。
O	—	
CV	○	無効演算が発生したとき“1”、それ以外のとき“0”になります。
CO	○	“0”になります。
CZ	○	“0”になります。
CU	○	“0”になります。
CX	○	精度異常が発生したとき“1”、それ以外のとき“0”になります。
CE	○	非実装処理が発生したとき“1”、それ以外のとき“0”になります。
FV	○	無効演算が発生したとき“1”、それ以外のときは変化しません。
FO	—	
FZ	—	
FU	—	
FX	○	精度異常が発生したとき“1”、それ以外のときは変化しません。

注. FX、FVフラグは、例外処理許可ビットEX、EVが“1”の場合は変化しません。S、Zフラグは、例外処理が発生した場合は変化しません。

【命令フォーマット】

構文	処理 サイズ	対象		コードサイズ (バイト)
		src	dest	
FTOI src, dest	L	Rs	Rd	3
	L	[Rs].L	Rd	3
	L	dsp:8[Rs].L (注)	Rd	4
	L	dsp:16[Rs].L (注)	Rd	5

注. 弊社の「RXファミリ アセンブラ」では、ディスプレイメントの値（dsp:8, dsp:16）は、4の倍数を指定してください。dsp:8には、0～1020（255×4）が指定できます。dsp:16には、0～262140（65535×4）が指定できます。命令コードには、1/4した値が埋め込まれます。

【単精度浮動小数点例外発生要因】

非実装処理
無効演算
精度異常

【記述例】

FTOI R1, R2

FTOI [R1], R2

【動作補足説明】

- FPSWのDNビットが“0”の場合と“1”の場合について、src、destの値と演算結果の対応を以下に示します。

DN = 0のとき

srcの値 (指数部はゲタなしの値)		dest	例外
src ≥ 0	+∞	EV = 1で無効演算発生時：変化なし 上記以外：7FFFFFFFh	無効演算
	31 ≤ 指数部 ≤ 127		
	-126 ≤ 指数部 ≤ 30	00000000h ~ 7FFFFFF80h	なし (注1)
	+非正規化数	変化なし	非実装
	+0	00000000h	なし
src < 0	-0		
	-非正規化数	変化なし	非実装
	-126 ≤ 指数部 ≤ 30	00000000h ~ 80000080h	なし (注1)
	31 ≤ 指数部 ≤ 127	EV = 1で無効演算発生時：変化なし 上記以外：80000000h	無効演算 (注2)
	-∞		
NaN	QNaN	EV = 1で無効演算発生時：変化なし 上記以外：	無効演算
	SNaN	符号ビット = 0 : 7FFFFFFFh 符号ビット = 1 : 80000000h	

注1. 丸め発生時は、精度異常が発生します。

注2. src = CF000000hのとき、無効演算は発生しません。

DN = 1のとき

srcの値 (指数部はゲタなしの値)		dest	例外
src ≥ 0	+∞	EV = 1で無効演算発生時：変化なし 上記以外：7FFFFFFFh	無効演算
	31 ≤ 指数部 ≤ 127		
	-126 ≤ 指数部 ≤ 30	00000000h ~ 7FFFFFF80h	
	+0、+非正規化数	00000000h	
src < 0	-0、-非正規化数		なし (注1)
	-126 ≤ 指数部 ≤ 30	00000000h ~ 80000080h	
	31 ≤ 指数部 ≤ 127	EV = 1で無効演算発生時：変化なし 上記以外：80000000h	
	-∞		
NaN	QNaN	EV = 1で無効演算発生時：変化なし 上記以外：	無効演算
	SNaN	符号ビット = 0 : 7FFFFFFFh 符号ビット = 1 : 80000000h	

注1. 丸め発生時は、精度異常が発生します。

注2. src = CF000000hのとき、無効演算は発生しません。

FTOU

単精度浮動小数点数→符号なし整数変換
Float TO Unsigned integer

FTOU

【構文】

```
FTOU src, dest
```

単精度浮動小数点演算命令

【命令コード】

記載ページ：287

【オペレーション】

```
dest = ( unsigned long ) src;
```

【機能】

- srcに格納された単精度浮動小数点数を符号なしロングワード（32ビット）整数に変換し、その結果をdestに格納します。
- 結果はFPSWのRM[1:0]ビットに関係なく、0方向に丸められます。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	演算の結果が“0”のとき“1”、それ以外の場合“0”になります。
S	○	演算の結果、ビット31が“1”のとき“1”、“0”のとき“0”になります。
O	—	
CV	○	無効演算が発生したとき“1”、それ以外の場合“0”になります。
CO	○	“0”になります。
CZ	○	“0”になります。
CU	○	“0”になります。
CX	○	精度異常が発生したとき“1”、それ以外の場合“0”になります。
CE	○	非実装処理が発生したとき“1”、それ以外の場合“0”になります。
FV	○	無効演算が発生したとき“1”、それ以外の場合は変化しません。
FO	—	
FZ	—	
FU	—	
FX	○	精度異常が発生したとき“1”、それ以外の場合は変化しません。

注. FX、FVフラグは、例外処理許可ビットEX、EVが“1”の場合は変化しません。S、Zフラグは、例外処理が発生した場合は変化しません。

【命令フォーマット】

構文	処理 サイズ	対象		コードサイズ (バイト)
		src	dest	
FTOU src, dest	L	Rs	Rd	3
	L	[Rs].L	Rd	3
	L	dsp:8[Rs].L (注)	Rd	4
	L	dsp:16[Rs].L (注)	Rd	5

注. 弊社の「RXファミリ アセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、4の倍数を指定してください。dsp:8には、0～1020 (255×4) が指定できます。dsp:16には、0～262140 (65535×4) が指定できます。命令コードには、1/4した値が埋め込まれます。

【単精度浮動小数点例外発生要因】

非実装処理
無効演算
精度異常

【記述例】

FTOU R1, R2
FTOU [R1], R2

【動作補足説明】

- FPSWのDNビットが“0”の場合と“1”の場合について、src、destの値と演算結果の対応を以下に示します。

DN = 0のとき

srcの値 (指数部はゲタなしの値)	dest	例外	
src ≥ 0	+∞	EV = 1で無効演算発生時：変化なし 上記以外：FFFFFFFFh	無効演算
	32 ≤ 指数部 ≤ 127		
	-126 ≤ 指数部 ≤ 31	00000000h ~ FFFFFFF0h	なし (注1)
	+非正規化数	変化なし	非実装
	+0	00000000h	なし
src < 0	-0		
	-非正規化数	変化なし	非実装
	-正規化数、-∞	EV = 1で無効演算発生時：変化なし 上記以外：00000000h	無効演算
NaN	QNaN	EV = 1で無効演算発生時：変化なし 上記以外：	無効演算
	SNaN	最上位ビット = 0 : FFFFFFFFh 最上位ビット = 1 : 00000000h	

注1. 丸め発生時は、精度異常が発生します。

DN = 1のとき

srcの値 (指数部はゲタなしの値)	dest	例外	
src ≥ 0	+∞	EV = 1で無効演算発生時：変化なし 上記以外：FFFFFFFFh	無効演算
	32 ≤ 指数部 ≤ 127		
	-126 ≤ 指数部 ≤ 31	00000000h ~ FFFFFFF0h	なし (注1)
	+0、+非正規化数	00000000h	なし
src < 0	-0、-非正規化数		
	-正規化数、-∞	EV = 1で無効演算発生時：変化なし 上記以外：00000000h	無効演算
NaN	QNaN	EV = 1で無効演算発生時：変化なし 上記以外：	無効演算
	SNaN	符号ビット = 0 : FFFFFFFFh 符号ビット = 1 : 00000000h	

注1. 丸め発生時は、精度異常が発生します。

INT

ソフトウェア割り込み INTerrupt

INT

【構文】

INT src

システム操作命令

【命令コード】

記載ページ：288

【オペレーション】

```

tmp0 = PSW;
U = 0;
I = 0;
PM = 0;
tmp1 = PC + 3;
PC = *(IntBase + src * 4);
SP = SP - 4;
*SP = tmp0;
SP = SP - 4;
*SP = tmp1;

```

【機能】

- srcで指定した番号の無条件トラップが発生します。
- srcの範囲は、 $0 \leq \text{src} \leq 255$ です。
- スーパーバイザモードに移行し、PSWのPMビットが“0”になります。
- PSWのU、Iビットが“0”になります。

【フラグ変化】

- フラグは変化しません。
- 命令実行前のPSWは、スタックに退避させられます。

【命令フォーマット】

構文	対象	コードサイズ (バイト)
	src	
INT src	#IMM:8	3

【記述例】

INT #0

ITOF符号付き整数→単精度浮動小数点数変換
Integer TO Floating-point**ITOF****【構文】**

ITOF src, dest

単精度浮動小数点演算命令

【命令コード】

記載ページ：288

【オペレーション】

dest = (float) src;

【機能】

- srcに格納された符号付きロングワード（32ビット）整数を単精度浮動小数点数に変換し、その結果をdestに格納します。結果はFPSWのRM[1:0]ビットに従って丸められます。00000000hは丸めモードに関係なく、“+0”として扱われます。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	演算の結果が“+0”のとき“1”、それ以外るとき“0”になります。
S	○	演算の結果、符号部（ビット31）が“1”のとき“1”、“0”のとき“0”になります。
O	—	
CV	○	“0”になります。
CO	○	“0”になります。
CZ	○	“0”になります。
CU	○	“0”になります。
CX	○	精度異常が発生したとき“1”、それ以外るとき“0”になります。
CE	○	“0”になります。
FV	—	
FO	—	
FZ	—	
FU	—	
FX	○	精度異常が発生したとき“1”、それ以外ときは変化しません。

注. FXフラグは、例外処理許可ビットEXが“1”の場合は変化しません。S、Zフラグは、例外処理が発生した場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
ITOF src, dest	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == "UB") 4 (memex != "UB")
	L	dsp:8[Rs].memex (注)	Rd	4 (memex == "UB") 5 (memex != "UB")
	L	dsp:16[Rs].memex (注)	Rd	5 (memex == "UB") 6 (memex != "UB")

注. 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、サイズ拡張指定子が“.W”または“.UW”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:8には、サイズ拡張指定子が“.W”または“.UW”のとき0～510 (255 × 2) が、“.L”のとき0～1020 (255 × 4) が指定できます。dsp:16には、サイズ拡張指定子が“.W”または“.UW”のとき0～131070 (65535 × 2) が、“.L”のとき0～262140 (65535 × 4) が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【単精度浮動小数点例外発生要因】

精度異常

【記述例】

ITOF R1, R2
ITOF [R1], R2
ITOF 16[R1].L, R2

JMP

無条件分岐
JuMP

JMP

【構文】

JMP src

分岐命令

【命令コード】

記載ページ：289

【オペレーション】

PC = src;

【機能】

- srcへ分岐します。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	対象	コードサイズ (バイト)
	src	
JMP src	Rs	2

【記述例】

JMP R1

JSR

サブルーチン分岐
Jump SubRoutine

JSR

分岐命令

【命令コード】

記載ページ：289

【構文】

JSR src

【オペレーション】

SP = SP - 4;
*SP = (PC + 2); (注)
PC = src;

注. (PC + 2) は JSR 命令の次の命令の番地です。

【機能】

- src が示すサブルーチンへ分岐します。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	対象	コードサイズ (バイト)
	src	
JSR src	Rs	2

【記述例】

JSR R1

MACHI

上位16ビット積和演算 Multiply-ACcumulate High-order word

MACHI

【構文】

MACHI src, src2, Adest

DSP機能命令

【命令コード】

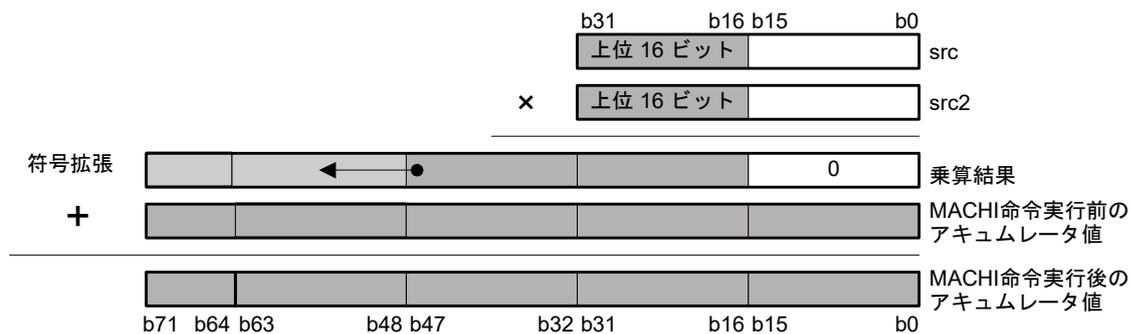
記載ページ：290

【オペレーション】

```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) (src >> 16);
tmp2 = (signed short) (src2 >> 16);
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = Adest + (tmp3 << 16);
```

【機能】

- srcの上位16ビットとsrc2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b16にあわせて加算します。加算結果はアキュムレータに格納されます。srcの上位16ビットとsrc2の上位16ビットは符号付き整数として扱われます。



【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	対象			コードサイズ (バイト)
	src	src2	Adest	
MACHI src, src2, Adest	Rs	Rs2	A0, A1	3

【記述例】

MACHI R1, R2, A1

MACLH

下位16ビット・上位16ビット積和演算
Multiply-ACcumulate Low-order word and
High-order word

MACLH

DSP機能命令
【命令コード】
記載ページ：290

【構文】

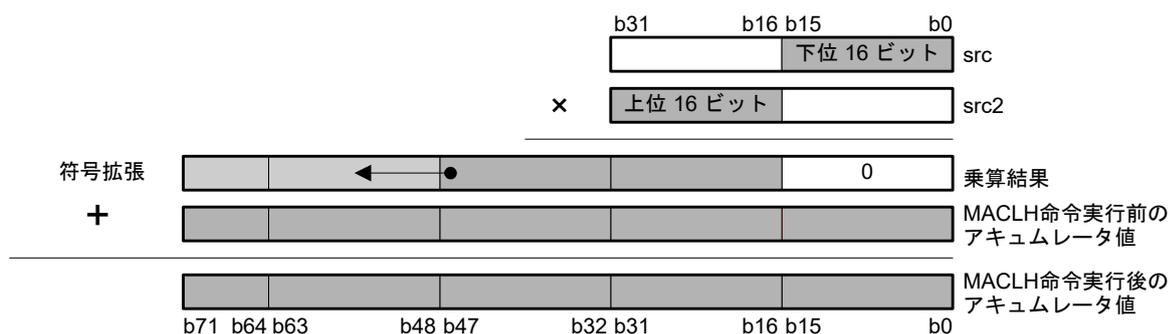
MACLH src, src2, Adest

【オペレーション】

```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) src;
tmp2 = (signed short) (src2 >> 16);
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = Adest + (tmp3 << 16);
```

【機能】

- srcの下位16ビットとsrc2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b16にあわせて加算します。加算結果はアキュムレータに格納されます。srcの下位16ビットとsrc2の上位16ビットは符号付き整数として扱われます。



【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	対象			コードサイズ (バイト)
	src	src2	Adest	
MACLH src, src2, Adest	Rs	Rs2	A0, A1	3

【記述例】

MACLH R1, R2, A1

MACLO

下位16ビット積和演算
Multiply-ACcumulate LOW-order word

MACLO

【構文】

```
MACLO src, src2, Adest
```

DSP 機能命令

【命令コード】

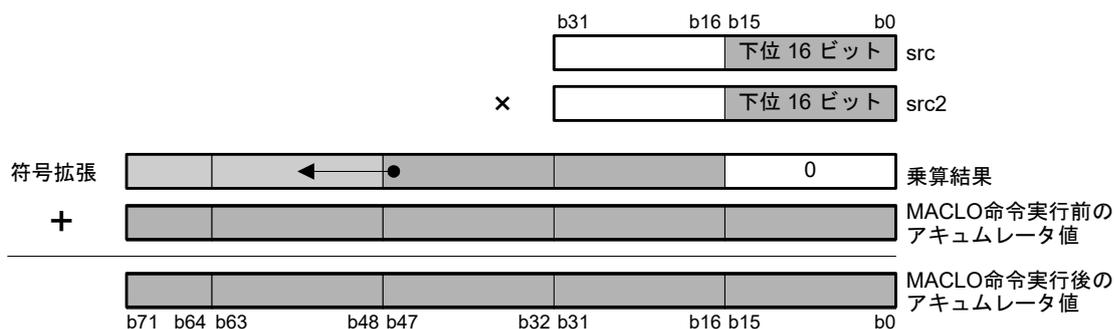
記載ページ：290

【オペレーション】

```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) src;
tmp2 = (signed short) src2;
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = Adest + (tmp3 << 16);
```

【機能】

- srcの下位16ビットとsrc2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b16にあわせて加算します。加算結果はアキュムレータに格納されます。srcの下位16ビットとsrc2の下位16ビットは符号付き整数として扱われます。

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	対象			コードサイズ (バイト)
	src	src2	Adest	
MACLO src, src2, Adest	Rs	Rs2	A0, A1	3

【記述例】

```
MACLO R1, R2, A1
```

MAX

最大値選択
MAXimum value select

MAX**【構文】**

```
MAX src, dest
```

算術/論理演算命令

【命令コード】

記載ページ：291

【オペレーション】

```
if ( src > dest )
    dest = src;
```

【機能】

- srcとdestを符号付きで比較し、大きい方の値をdestに格納します。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
MAX src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == "UB") 4 (memex != "UB")
	L	dsp:8[Rs].memex (注)	Rd	4 (memex == "UB") 5 (memex != "UB")
	L	dsp:16[Rs].memex (注)	Rd	5 (memex == "UB") 6 (memex != "UB")

注． 弊社の「RXファミリ アセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、サイズ拡張指定子が“.W”または“.UW”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:8には、サイズ拡張指定子が“.W”または“.UW”のとき0～510 (255×2) が、“.L”のとき0～1020 (255×4) が指定できます。dsp:16には、サイズ拡張指定子が“.W”または“.UW”のとき0～131070 (65535×2) が、“.L”のとき0～262140 (65535×4) が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【記述例】

```
MAX #10, R2
MAX R1, R2
MAX [R1], R2
MAX 3[R1].B, R2
```

MIN

最小値選択 MINimum value select

MIN

【構文】

MIN src, dest

算術/論理演算命令

【命令コード】

記載ページ：292

【オペレーション】if (src < dest)
dest = src;**【機能】**

- srcとdestを符号付きで比較し、小さい方の値をdestに格納します。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
MIN src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == "UB") 4 (memex != "UB")
	L	dsp:8[Rs].memex (注)	Rd	4 (memex == "UB") 5 (memex != "UB")
	L	dsp:16[Rs].memex (注)	Rd	5 (memex == "UB") 6 (memex != "UB")

注. 弊社の「RXファミリ アセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、サイズ拡張指定子が“.W”または“.UW”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:8には、サイズ拡張指定子が“.W”または“.UW”のとき0～510 (255×2) が、“.L”のとき0～1020 (255×4) が指定できます。dsp:16には、サイズ拡張指定子が“.W”または“.UW”のとき0～131070 (65535×2) が、“.L”のとき0～262140 (65535×4) が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【記述例】

```
MIN #10, R2
MIN R1, R2
MIN [R1], R2
MIN 3[R1].B, R2
```

MOV

転送
MOVE

MOV

【構文】

MOV.size src, dest

転送命令

【命令コード】

記載ページ : 293

【オペレーション】

dest = src;

【機能】

- 以下のとおり、srcをdestに転送します。

src	dest	機能
即値	レジスタ	即値をレジスタに転送します。32ビット未満の即値が指定された場合、#UIMMはゼロ拡張を、#SIMMは符号拡張を行いレジスタに転送します。
即値	メモリ	即値を指定したサイズでメモリに転送します。指定したサイズよりもビット幅の小さい即値が指定された場合、#UIMMはゼロ拡張を、#SIMMは符号拡張を行いメモリに転送します。
レジスタ	レジスタ	レジスタ (src) のデータをレジスタ (dest) に転送します。サイズ指定子が.Bのときは、レジスタ (src) のLSB側のバイトデータをロングワードデータに符号拡張し、レジスタ (dest) に転送します。サイズ指定子が.Wのときは、レジスタ (src) のLSB側のワードデータをロングワードデータに符号拡張し、レジスタ (dest) に転送します。
レジスタ	メモリ	レジスタのデータをメモリに転送します。サイズ指定子が.Bのときは、レジスタのLSB側のバイトデータを転送します。サイズ指定子が.Wのときは、レジスタのLSB側のワードデータを転送します。
メモリ	レジスタ	メモリのデータをレジスタに転送します。サイズ指定子が.Bまたは.Wのときは、メモリのデータをロングワードデータに符号拡張し、レジスタに転送します。
メモリ	メモリ	指定したサイズでメモリ (src) のデータをメモリ (dest) に転送します。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	size	処理サイズ	対象		コードサイズ (バイト)
			src	dest	
MOV.size src, dest	ストア (短縮命令)				
	B/W/L	size	Rs (Rs = R0 ~ R7)	dsp:5[Rd] (注1) (Rd = R0 ~ R7)	2
	ロード (短縮命令)				
	B/W/L	L	dsp:5[Rs] (注1) (Rs = R0 ~ R7)	Rd (Rd = R0 ~ R7)	2
	レジスタへの即値設定 (短縮命令)				
	L	L	#UIMM:4	Rd	2
	メモリへの即値設定 (短縮命令)				
	B	B	#IMM:8	dsp:5[Rd] (注1) (Rd = R0 ~ R7)	3
	W/L	size	#UIMM:8	dsp:5[Rd] (注1) (Rd = R0 ~ R7)	3
	レジスタへの即値設定				
	L	L	#UIMM:8 (注2)	Rd	3
	L	L	#SIMM:8 (注2)	Rd	3
	L	L	#SIMM:16	Rd	4
	L	L	#SIMM:24	Rd	5
	L	L	#IMM:32	Rd	6
	レジスタ間転送 (符号拡張あり)				
	B/W	L	Rs	Rd	2
	レジスタ間転送 (符号拡張なし)				
	L	L	Rs	Rd	2
	メモリへの即値設定				
	B	B	#IMM:8	[Rd]	3
	B	B	#IMM:8	dsp:8[Rd] (注1)	4
	B	B	#IMM:8	dsp:16[Rd] (注1)	5
	W	W	#SIMM:8	[Rd]	3
	W	W	#SIMM:8	dsp:8[Rd] (注1)	4
	W	W	#SIMM:8	dsp:16[Rd] (注1)	5
	W	W	#IMM:16	[Rd]	4
	W	W	#IMM:16	dsp:8[Rd] (注1)	5
	W	W	#IMM:16	dsp:16[Rd] (注1)	6
	L	L	#SIMM:8	[Rd]	3
	L	L	#SIMM:8	dsp:8[Rd] (注1)	4
	L	L	#SIMM:8	dsp:16[Rd] (注1)	5
	L	L	#SIMM:16	[Rd]	4
L	L	#SIMM:16	dsp:8[Rd] (注1)	5	
L	L	#SIMM:16	dsp:16[Rd] (注1)	6	
L	L	#SIMM:24	[Rd]	5	
L	L	#SIMM:24	dsp:8[Rd] (注1)	6	
L	L	#SIMM:24	dsp:16[Rd] (注1)	7	
L	L	#IMM:32	[Rd]	6	
L	L	#IMM:32	dsp:8[Rd] (注1)	7	
L	L	#IMM:32	dsp:16[Rd] (注1)	8	

構文	size	処理サイズ	対象		コードサイズ (バイト)
			src	dest	
MOV.size src, dest	ロード				
	B/W/L	L	[Rs]	Rd	2
	B/W/L	L	dsp:8[Rs] (注1)	Rd	3
	B/W/L	L	dsp:16[Rs] (注1)	Rd	4
	B/W/L	L	[Ri, Rb]	Rd	3
	ストア				
	B/W/L	size	Rs	[Rd]	2
	B/W/L	size	Rs	dsp:8[Rd] (注1)	3
	B/W/L	size	Rs	dsp:16[Rd] (注1)	4
	B/W/L	size	Rs	[Ri, Rb]	3
	メモリ間転送				
	B/W/L	size	[Rs]	[Rd]	2
	B/W/L	size	[Rs]	dsp:8[Rd] (注1)	3
	B/W/L	size	[Rs]	dsp:16[Rd] (注1)	4
	B/W/L	size	dsp:8[Rs] (注1)	[Rd]	3
	B/W/L	size	dsp:8[Rs] (注1)	dsp:8[Rd] (注1)	4
	B/W/L	size	dsp:8[Rs] (注1)	dsp:16[Rd] (注1)	5
	B/W/L	size	dsp:16[Rs] (注1)	[Rd]	4
	B/W/L	size	dsp:16[Rs] (注1)	dsp:8[Rd] (注1)	5
	B/W/L	size	dsp:16[Rs] (注1)	dsp:16[Rd] (注1)	6
	ポストインクリメント付きストア (注3)				
	B/W/L	size	Rs	[Rd+]	3
	プリデクリメント付きストア (注3)				
	B/W/L	size	Rs	[-Rd]	3
	ポストインクリメント付きロード (注4)				
	B/W/L	L	[Rs+]	Rd	3
	プリデクリメント付きロード (注4)				
	B/W/L	L	[-Rs]	Rd	3

- 注1. 弊社の「RXファミリ アセンブラ」では、ディスプレースメントの値 (dsp:5, dsp:8, dsp:16) は、サイズ指定子が“.W”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:5には、サイズ指定子が“.W”のとき0～62 (31×2) が、“.L”のとき0～124 (31×4) が指定できます。dsp:8には、サイズ指定子が“.W”のとき0～510 (255×2) が、“.L”のとき0～1020 (255×4) が指定できます。dsp:16には、サイズ指定子が“.W”のとき0～131070 (65535×2) が、“.L”のとき0～262140 (65535×4) が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。
- 注2. 0～127の範囲は、ゼロ拡張命令コードになります。
- 注3. ポストインクリメント付きストア、プリデクリメント付きストアで、RsとRdに同じレジスタを指定した場合、アドレス更新前の値がソースとして転送されます。
- 注4. ポストインクリメント付きロード、プリデクリメント付きロードで、RsとRdに同じレジスタを指定した場合、メモリから転送されてきたデータがRdに格納されます。

【記述例】

```
MOV.L #0, R2
MOV.L #128:8, R2
MOV.L #-128:8, R2
MOV.L R1, R2
MOV.L #0, [R2]
MOV.W [R1], R2
MOV.W R1, [R2]
MOV.W [R1, R2], R3
MOV.W R1, [R2, R3]
MOV.W [R1], [R2]
MOV.B R1, [R2+]
MOV.B [R1+], R2
MOV.B R1, [-R2]
MOV.B [-R1], R2
```

MOVCO

LIフラグクリア付きストア
MOVE-COnditional

MOVCO

転送命令

【命令コード】

記載ページ：298

【構文】

MOVCO src, dest

【オペレーション】

```
if (LI == 1) {
    dest = src;
    src = 0;
} else {
    src = 1;
}
LI = 0;
```

【機能】

LIフラグが1のとき、src（レジスタ）のデータをdest（メモリ）にストアし、LIフラグおよびsrcを0にします。

LIフラグが0のとき、ストア動作は行わず、srcに1をセットします。

LIフラグはCPUの内部にあり、MOVCO命令、MOVLI命令、RTE命令、RTFI命令による操作を除き、ユーザがこのフラグを直接アクセスすることはできません。

MOVCO命令実行前には、同じアドレスに対するMOVLI命令を実行してください。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
MOVCO src, dest	L	Rs	[Rd]	3

【記述例】

MOVCO R1, [R2]

MOVLI

LIフラグセット付きロード
MOVe LInked

MOVLI

転送命令

【命令コード】

記載ページ：298

【構文】

MOVLI src, dest

【オペレーション】

LI = 1;
dest = src;

【機能】

src（メモリ）のロングワードデータを、dest（レジスタ）に転送します。
この命令は、通常のロード動作を行うと共に、LIフラグのセットも行います。

LIフラグがクリアされるのは、以下の条件が成立したときです。

MOVCO命令を実行したとき
RTE, RTFI命令を実行したとき

LIフラグはCPUの内部にあり、MOVCO命令、MOVLI命令、RTE命令、RTFI命令による操作を除き、ユーザがこのフラグを直接アクセスすることはできません。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
MOVLI src, dest	L	[Rs]	Rd	3

【記述例】

MOVLI [R1], R2

MOVU

符号なしデータ転送
MOVE Unsigned data

MOVU

転送命令

【命令コード】

記載ページ：299

【構文】

MOVU.size src, dest

【オペレーション】

dest = src;

【機能】

- 以下のとおり、srcをdestに転送します。

src	dest	機能
レジスタ	レジスタ	レジスタ (src) のLSB側のバイトデータまたはワードデータをロングワードデータにゼロ拡張し、レジスタ (dest) に転送します。
メモリ	レジスタ	メモリのバイトデータまたはワードデータをロングワードデータにゼロ拡張し、レジスタに転送します。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	size	処理サイズ	対象		コードサイズ (バイト)
			src	dest	
MOVU.size src, dest	ロード (短縮命令)				
	B/W	L	dsp:5[Rs] (注1) (Rs = R0 ~ R7)	Rd (Rd = R0 ~ R7)	2
	レジスタ間転送 (ゼロ拡張あり)				
	B/W	L	Rs	Rd	2
	ロード				
	B/W	L	[Rs]	Rd	2
	B/W	L	dsp:8[Rs] (注1)	Rd	3
	B/W	L	dsp:16[Rs] (注1)	Rd	4
	B/W	L	[Ri, Rb]	Rd	3
	ポストインクリメント付きロード (注2)				
	B/W	L	[Rs+]	Rd	3
	プリデクリメント付きロード (注2)				
	B/W	L	[-Rs]	Rd	3

注1. 弊社の「RXファミリ アセンブラ」では、ディスプレイメントの値 (dsp:5, dsp:8, dsp:16) は、サイズ指定子が“.W"のときは2の倍数を指定してください。dsp:5には、サイズ指定子が“.W"のとき0~62 (31×2) が指定できます。dsp:8には、サイズ指定子が“.W"のとき0~510 (255×2) が指定できます。dsp:16には、サイズ指定子が“.W"のとき0~131070 (65535×2) が指定できます。命令コードには、1/2した値が埋め込まれます。

注2. ポストインクリメント付きロード、プリデクリメント付きロードで、RsとRdに同じレジスタを指定した場合、メモリから転送されてきたデータがRdに格納されます。

【記述例】

MOVU.W 2[R1], R2

MOVU.W R1, R2

MOVU.B [R1+], R2

MOVU.B [-R1], R2

MSBHI

上位16ビット積差演算
Multiply-SuBtract High-order word

MSBHI

【構文】

```
MSBHI src, src2, Adest
```

DSP 機能命令

【命令コード】

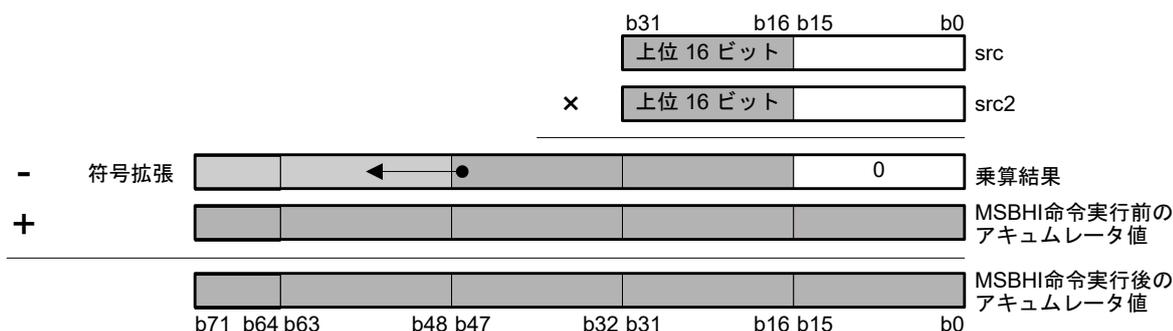
記載ページ : 300

【オペレーション】

```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) (src >> 16);
tmp2 = (signed short) (src2 >> 16);
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = Adest - (tmp3 << 16);
```

【機能】

- srcの上位16ビットとsrc2の上位16ビットの乗算を行い、乗算結果をアキュムレータから減算します。ただし、乗算結果の最下位ビットはアキュムレータのb16にあわせて減算します。減算結果はアキュムレータに格納されます。srcの上位16ビットとsrc2の上位16ビットは符号付き整数として扱われます。

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	対象			コードサイズ (バイト)
	src	src2	Adest	
MSBHI src, src2, Adest	Rs	Rs2	A0, A1	3

【記述例】

```
MSBHI R1, R2, A1
```

MSBLH

下位16ビット・上位16ビット積差演算
Multiply-SuBtract Low-order word and
High-order word

MSBLH

DSP 機能命令

【命令コード】

記載ページ：300

【構文】

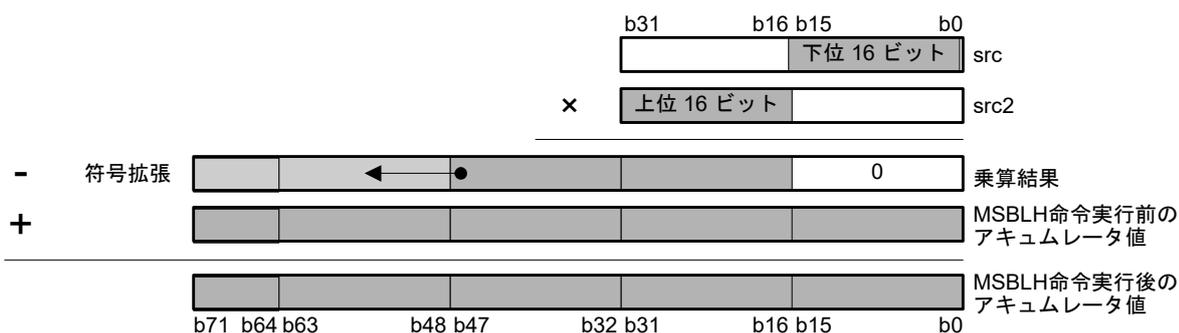
MSBLH src, src2, Adest

【オペレーション】

```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) src;
tmp2 = (signed short) (src2 >> 16);
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = Adest - (tmp3 << 16);
```

【機能】

- srcの下位16ビットとsrc2の上位16ビットの乗算を行い、乗算結果をアキュムレータから減算します。ただし、乗算結果の最下位ビットはアキュムレータのb16にあわせて減算します。減算結果はアキュムレータに格納されます。srcの下位16ビットとsrc2の上位16ビットは符号付き整数として扱われます。

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	対象			コードサイズ (バイト)
	src	src2	Adest	
MSBLH src, src2, Adest	Rs	Rs2	A0, A1	3

【記述例】

MSBLH R1, R2, A1

MSBLO

下位16ビット積差演算
Multiply-SuBtract LOw-order word

MSBLO

【構文】

```
MSBLO src, src2, Adest
```

DSP機能命令

【命令コード】

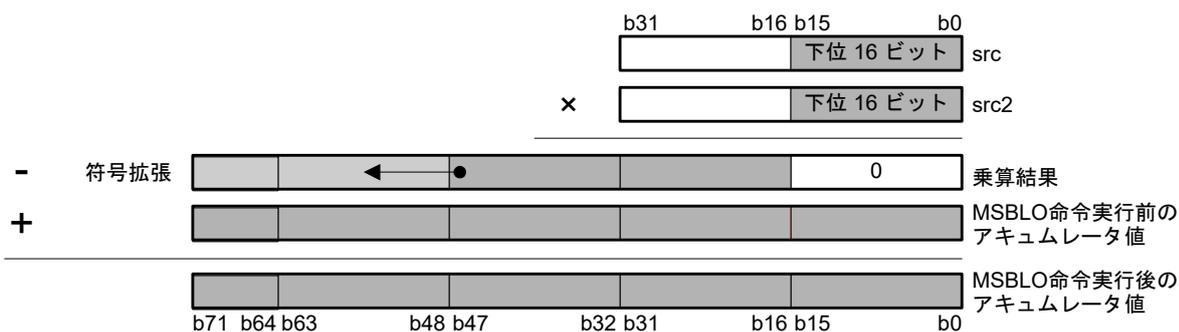
記載ページ：301

【オペレーション】

```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) src;
tmp2 = (signed short) src2;
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = Adest - (tmp3 << 16);
```

【機能】

- srcの下位16ビットとsrc2の下位16ビットの乗算を行い、乗算結果をアキュムレータから減算します。ただし、乗算結果の最下位ビットはアキュムレータのb16にあわせて減算します。減算結果はアキュムレータに格納されます。srcの下位16ビットとsrc2の下位16ビットは符号付き整数として扱われま

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	対象			コードサイズ (バイト)
	src	src2	Adest	
MSBLO src, src2, Adest	Rs	Rs2	A0, A1	3

【記述例】

```
MSBLO R1, R2, A1
```

MUL

乗算
MULTIPLY

MUL

【構文】

- (1) MUL src, dest
- (2) MUL src, src2, dest

算術/論理演算命令

【命令コード】

記載ページ：301

【オペレーション】

- (1) dest = src * dest;
- (2) dest = src * src2;

【機能】

- (1) srcとdestを乗算し、その結果をdestに格納します。
 - 演算は32ビットで行い、結果の下位32ビットを格納します。
 - 演算結果は、符号付き乗算、符号なし乗算に関係なく同じになります。
- (2) srcとsrc2を乗算し、その結果をdestに格納します。
 - 演算は32ビットで行い、結果の下位32ビットを格納します。
 - 演算結果は、符号付き乗算、符号なし乗算に関係なく同じになります。

注. アキュムレータ (ACC0) を使用します。命令実行後のACC0の値は不定です。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象			コードサイズ (バイト)
		src	src2	dest	
(1) MUL src, dest	L	#UIMM:4	—	Rd	2
	L	#SIMM:8	—	Rd	3
	L	#SIMM:16	—	Rd	4
	L	#SIMM:24	—	Rd	5
	L	#IMM:32	—	Rd	6
	L	Rs	—	Rd	2
	L	[Rs].memex	—	Rd	2 (memex == "UB") 3 (memex != "UB")
	L	dsp:8[Rs].memex (注)	—	Rd	3 (memex == "UB") 4 (memex != "UB")
	L	dsp:16[Rs].memex (注)	—	Rd	4 (memex == "UB") 5 (memex != "UB")
(2) MUL src, src2, dest	L	Rs	Rs2	Rd	3

注. 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、サイズ拡張指定子が“.W”または“.UW”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:8には、サイズ拡張指定子が“.W”または“.UW”のとき0～510 (255×2) が、“.L”のとき0～1020 (255×4) が指定できます。dsp:16には、サイズ拡張指定子が“.W”または“.UW”のとき0～131070 (65535×2) が、“.L”のとき0～262140 (65535×4) が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【記述例】

MUL #10, R2

MUL R1, R2

MUL [R1], R2

MUL 4[R1].W, R2

MUL R1, R2, R3

MULHI

上位16ビット乗算
MULTIply High-order word

MULHI

【構文】

MULHI src, src2, Adest

DSP機能命令

【命令コード】

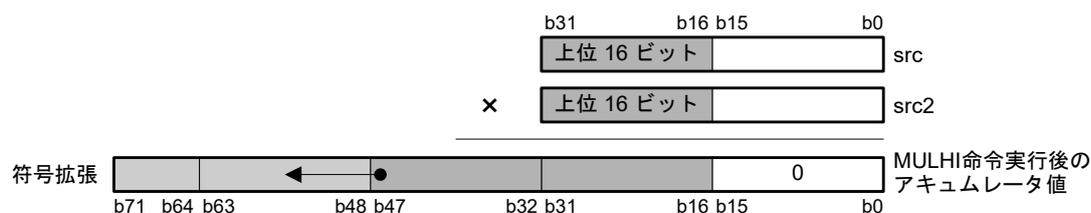
記載ページ：303

【オペレーション】

```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) (src >> 16);
tmp2 = (signed short) (src2 >> 16);
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = (tmp3 << 16);
```

【機能】

- srcの上位16ビットとsrc2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b16にあわせ、アキュムレータのb71～b48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b15～b0は、“0”になります。srcの上位16ビットとsrc2の上位16ビットは符号付き整数として扱われます。

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	対象			コードサイズ (バイト)
	src	src2	Adest	
MULHI src, src2, Adest	Rs	Rs2	A0, A1	3

【記述例】

MULHI R1, R2, A1

MULLH

下位16ビット・上位16ビット乗算
Multiply Low-order word and High-order word

MULLH

【構文】

MULLH src, src2, Adest

DSP機能命令

【命令コード】

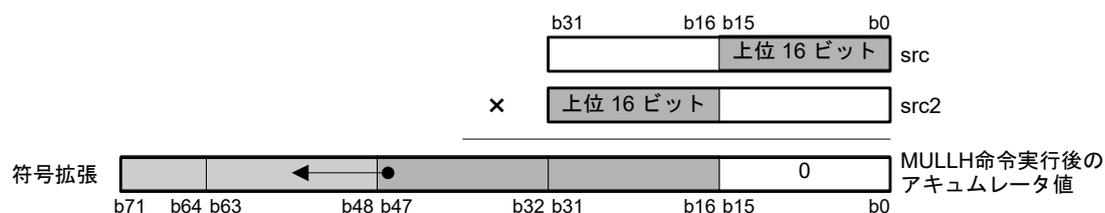
記載ページ：303

【オペレーション】

```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) src;
tmp2 = (signed short) (src2 >> 16);
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = (tmp3 << 16);
```

【機能】

- srcの下位16ビットとsrc2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b16にあわせ、アキュムレータのb71～b48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b15～b0は、“0”になります。srcの下位16ビットとsrc2の上位16ビットは符号付き整数として扱われます。

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	対象			コードサイズ (バイト)
	src	src2	Adest	
MULLH src, src2, Adest	Rs	Rs2	A0, A1	3

【記述例】

MULLH R1, R2, A1

MULLO

下位16ビット乗算
MULTIply LOw-order word

MULLO

【構文】

MULLO src, src2, Adest

DSP機能命令

【命令コード】

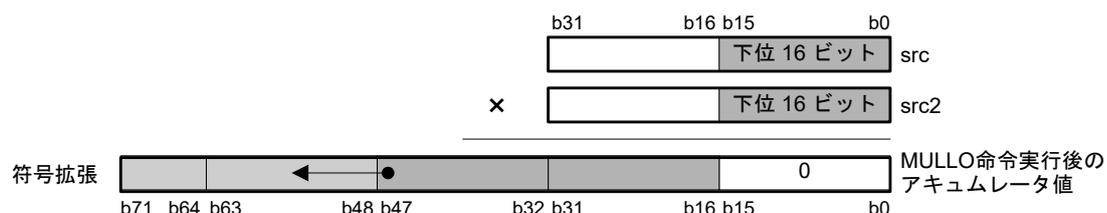
記載ページ：304

【オペレーション】

```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) src;
tmp2 = (signed short) src2;
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = (tmp3 << 16);
```

【機能】

- srcの下位16ビットとsrc2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b16にあわせ、アキュムレータのb71～b48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b15～b0は、“0”になります。srcの下位16ビットとsrc2の下位16ビットは符号付き整数として扱われます。

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	対象			コードサイズ (バイト)
	src	src2	Adest	
MULLO src, src2, Adest	Rs	Rs2	A0, A1	3

【記述例】

MULLO R1, R2, A1

MVFACGU

アキュムレータガードビットからの転送
MoVe From ACcumulator GUard longword

MVFACGU

DSP機能命令

【命令コード】

記載ページ：304

【構文】

MVFACGU src, Asrc, dest

【オペレーション】

```
signed 72bit tmp;
tmp = (signed 72bit) Asrc << src;
dest = (signed long) (tmp >> 64);
```

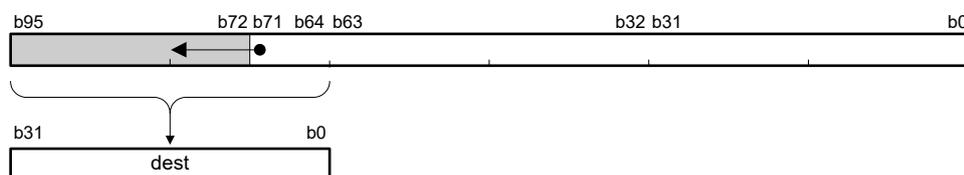
【機能】

- MVFACGU命令は、以下のような手順で実行されます。

処理1. アキュムレータの値を、srcで指定したビット数分（0～2ビット）、左シフトします。



処理2. シフト後の値の最上位32ビットの内容をdestに転送します。

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	対象			コードサイズ (バイト)
	src	Asrc	dest	
MVFACGU src, Asrc, dest	#IMM:2 (IMM:2 = 0～2)	A0, A1	Rd	3

【記述例】

MVFACGU #1, A1, R1

MVFACHI

アキュムレータ上位32ビットからの転送
MoVe From ACcumulator High-order longword

MVFACHI

DSP 機能命令

【命令コード】

記載ページ : 305

【構文】

MVFACHI src, Asrc, dest

【オペレーション】

```
signed 72bit tmp;
tmp = (signed 72bit) Asrc << src;
dest = (signed long) (tmp >> 32);
```

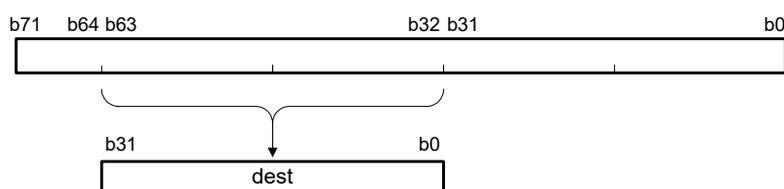
【機能】

- MVFACHI命令は、以下のような手順で実行されます。

処理1. アキュムレータの値を、srcで指定したビット数分 (0～2ビット)、左シフトします。



処理2. シフト後の値のb63～b32の内容をdestに転送します。

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	対象			コードサイズ (バイト)
	src	Asrc	dest	
MVFACHI src, Asrc, dest	#IMM:2 (IMM:2 = 0～2)	A0, A1	Rd	3

【記述例】

MVFACHI #1, A1, R1

MVFACLO

アキュムレータ下位32ビットからの転送
MoVe From ACcumulator LOw-order longword

MVFACLO

DSP 機能命令

【命令コード】

記載ページ : 305

【構文】

MVFACLO src, Asrc, dest

【オペレーション】

signed 72bit tmp;

tmp = (signed 72bit) Asrc << src;

dest = (signed long) tmp;

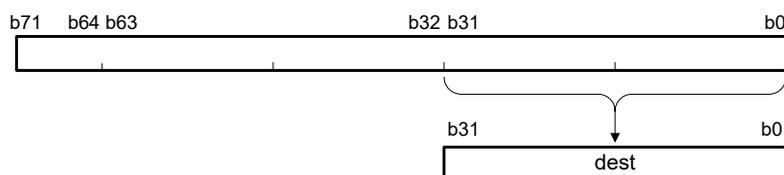
【機能】

- MVFACLO命令は、以下のような手順で実行されます。

処理1. アキュムレータの値を、srcで指定したビット数分 (0～2ビット)、左シフトします。



処理2. シフト後の値のb31～b0の内容をdestに転送します。

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	対象			コードサイズ (バイト)
	src	Asrc	dest	
MVFACLO src, Asrc, dest	#IMM:2 (IMM:2 = 0～2)	A0, A1	Rd	3

【記述例】

MVFACLO #1, A1, R1

MVFACMI

アキュムレータ中央32ビットからの転送
MoVe From ACcumulator MIddle-order longword

MVFACMI

DSP 機能命令

【命令コード】

記載ページ：306

【構文】

MVFACMI src, Asrc, dest

【オペレーション】

```
signed 72bit tmp;
tmp = (signed 72bit) Asrc << src;
dest = (signed long) (tmp >> 16);
```

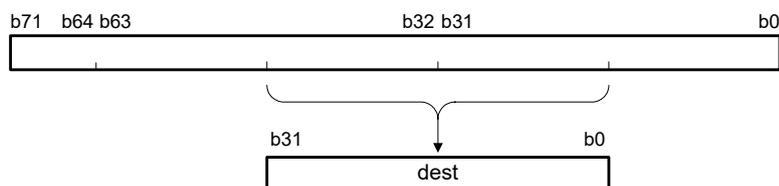
【機能】

- MVFACMI命令は、以下のような手順で実行されます。

処理1. アキュムレータの値を、srcで指定したビット数分（0～2ビット）、左シフトします。



処理2. シフト後の値のb47～b16の内容をdestに転送します。

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	対象			コードサイズ (バイト)
	src	Asrc	dest	
MVFACMI src, Asrc, dest	#IMM:2 (IMM:2 = 0～2)	A0, A1	Rd	3

【記述例】

MVFACMI #1, A1, R1

MVFC

制御レジスタからの転送
MoVe From Control register

MVFC

システム操作命令

【命令コード】

記載ページ：306

【構文】

MVFC src, dest

【オペレーション】

dest = src;

【機能】

- srcをdestに転送します。
- srcにPCを指定した場合、本命令の番地をdestに転送します。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src (注)	dest	
MVFC src, dest	L	Rx	Rd	3

注. 選択可能な src : PC, ISP, USP, INTB, EXTB, PSW, BPC, BPSW, FINTV, FPSW

【記述例】

MVFC USP, R1

MVTACGU

アキュムレータガードビットへの転送
MoVe To ACcumulator GUard longword

MVTACGU

DSP機能命令

【命令コード】

記載ページ：307

【構文】

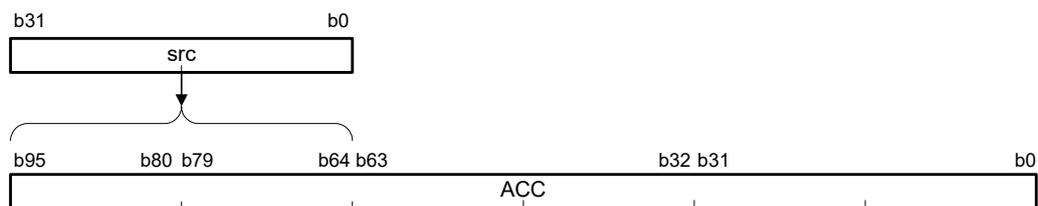
MVTACGU src, Adest

【オペレーション】

Adest = (Adest & 00FFFFFFFFFFFFFFFh) | ((signed 72bit) src << 64);

【機能】

- srcの内容をアキュムレータの最上位32ビット（b95～b64）に転送します。

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	対象		コードサイズ (バイト)
	src	Adest	
MVTACGU src, Adest	Rs	A0, A1	3

【記述例】

MVTACGU R1, A1

MVTACHI

アキュムレータ上位32ビットへの転送
MoVe To ACcumulator High-order longword

MVTACHI

DSP 機能命令

【命令コード】

記載ページ : 307

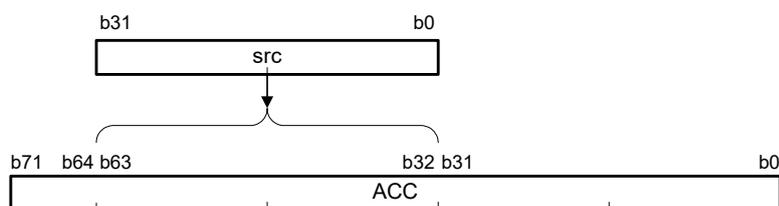
【構文】

MVTACHI src, Adest

【オペレーション】

$$Adest = (Adest \& \text{FF00000000FFFFFFFFh}) | ((\text{signed 72bit } src \ll 32);$$
【機能】

- srcの内容をアキュムレータの上位32ビット（b63～b32）に転送します。

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	対象		コードサイズ (バイト)
	src	Adest	
MVTACHI src, Adest	Rs	A0, A1	3

【記述例】

MVTACHI R1, A1

MVTACLO

アキュムレータ下位32ビットへの転送
MoVe To ACcumulator LOw-order longword

MVTACLO

DSP 機能命令

【命令コード】

記載ページ : 307

【構文】

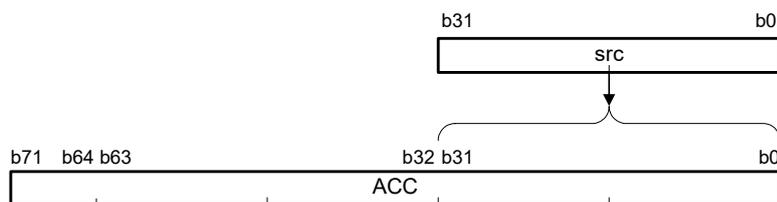
MVTACLO src, Adest

【オペレーション】

Adest = (Adest & FFFFFFFF00000000h) | (unsigned 72bit) src;

【機能】

- srcの内容をアキュムレータの下位32ビット (b31～b0) に転送します。

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	対象		コードサイズ (バイト)
	src	Adest	
MVTACLO src, Adest	Rs	A0, A1	3

【記述例】

MVTACLO R1, A1

MVTC

制御レジスタへの転送
MoVe To Control register

MVTC

【構文】

```
MVTC src, dest
```

システム操作命令

【命令コード】

記載ページ：308

【オペレーション】

```
dest = src;
```

【機能】

- srcをdestに転送します。
- ユーザモードでは、ISP、INTB、EXTB、BPC、BPSW、FINTVと、PSWのIPL[3:0]、PM、U、Iビットへの書き込みは無視されます。スーパーバイザモードでは、PSWのPMビットへの書き込みは無視されます。

【フラグ変化】

フラグ	変化	条件
C	(注)	
Z	(注)	
S	(注)	
O	(注)	

注. destがPSWのときだけ変化します。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest (注)	
MVTC src, dest	L	#SIMM:8	Rx	4
	L	#SIMM:16	Rx	5
	L	#SIMM:24	Rx	6
	L	#IMM:32	Rx	7
	L	Rs	Rx	3

注. 選択可能な dest : ISP, USP, INTB, EXTB, PSW, BPC, BPSW, FINTV, FPSW
destにPCを指定することはできません。

【記述例】

```
MVTC #0FFFFFF00h, INTB
```

```
MVTC R1, USP
```

MVTIPL

割り込み優先レベル設定
MoVe To Interrupt Priority Level

MVTIPL

システム操作命令

【命令コード】

記載ページ : 309

【構文】

MVTIPL src

【オペレーション】

IPL = src;

【機能】

- srcをPSWのIPL[3:0]ビットに転送します。
- この命令は特権命令です。ユーザモードで実行すると特権命令例外が発生します。
- srcの値は符号なし整数です。srcの範囲は、 $0 \leq \text{src} \leq 15$ です。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	対象	コードサイズ (バイト)
	src	
MVTIPL src	#IMM:4	3

【記述例】

MVTIPL #2

NEG

符号反転
NEGate

NEG

【構文】

- (1) NEG dest
- (2) NEG src, dest

算術/論理演算命令

【命令コード】

記載ページ : 310

【オペレーション】

- (1) dest = -dest;
- (2) dest = -src;

【機能】

- (1) destを符号反転し（2の補数を取り）、その結果をdestに格納します。
- (2) srcを符号反転し（2の補数を取り）、その結果をdestに格納します。

【フラグ変化】

フラグ	変化	条件
C	○	演算後のdestが0のとき“1”、それ以外のとき“0”になります。
Z	○	演算後のdestが0のとき“1”、それ以外のとき“0”になります。
S	○	演算後のdestのMSBが“1”のとき“1”、それ以外のとき“0”になります。
O	○	(1) 演算前のdestが80000000hのとき“1”、それ以外のとき“0”になります。 (2) 演算前のsrcが80000000hのとき“1”、それ以外のとき“0”になります。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
(1) NEG dest	L	—	Rd	2
(2) NEG src, dest	L	Rs	Rd	3

【記述例】

```
NEG R1
NEG R1, R2
```

NOP

ノーオペレーション
No OPeration

NOP

【構文】

NOP

算術/論理演算命令

【命令コード】

記載ページ：310

【オペレーション】

/* ノーオペレーション */

【機能】

- 処理は何も行いません。次の命令から継続して実行されます。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	コードサイズ (バイト)
NOP	1

【記述例】

NOP

NOT

論理反転
NOT

NOT

【構文】

- (1) NOT dest
- (2) NOT src, dest

算術/論理演算命令

【命令コード】

記載ページ : 311

【オペレーション】

- (1) dest = ~dest;
- (2) dest = ~src;

【機能】

- (1) destを論理反転し、その結果をdestに格納します。
- (2) srcを論理反転し、その結果をdestに格納します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	演算後のdestが0のとき“1”、それ以外のとき“0”になります。
S	○	演算後のdestのMSBが“1”のとき“1”、それ以外のとき“0”になります。
O	—	

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
(1) NOT dest	L	—	Rd	2
(2) NOT src, dest	L	Rs	Rd	3

【記述例】

```
NOT R1
NOT R1, R2
```

OR

論理和 OR

OR

【構文】

- (1) OR src, dest
 (2) OR src, src2, dest

算術/論理演算命令

【命令コード】

記載ページ：312

【オペレーション】

- (1) dest = dest | src;
 (2) dest = src2 | src;

【機能】

- (1) destとsrcの論理和をとり、その結果をdestに格納します。
 (2) src2とsrcの論理和をとり、その結果をdestに格納します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	演算後のdestが0のとき“1”、それ以外のとき“0”になります。
S	○	演算後のdestのMSBが“1”のとき“1”、それ以外のとき“0”になります。
O	—	

【命令フォーマット】

構文	処理サイズ	対象			コードサイズ (バイト)
		src	src2	dest	
(1) OR src, dest	L	#UIMM:4	—	Rd	2
	L	#SIMM:8	—	Rd	3
	L	#SIMM:16	—	Rd	4
	L	#SIMM:24	—	Rd	5
	L	#IMM:32	—	Rd	6
	L	Rs	—	Rd	2
	L	[Rs].memex	—	Rd	2 (memex == “UB”) 3 (memex != “UB”)
	L	dsp:8[Rs].memex (注)	—	Rd	3 (memex == “UB”) 4 (memex != “UB”)
	L	dsp:16[Rs].memex (注)	—	Rd	4 (memex == “UB”) 5 (memex != “UB”)
(2) OR src, src2, dest	L	Rs	Rs2	Rd	3

注. 弊社の「RXファミリアセンブラ」では、ディスプレースメントの値 (dsp:8, dsp:16) は、サイズ拡張指定子が“.W”または“.UW”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:8には、サイズ拡張指定子が“.W”または“.UW”のとき0～510 (255 × 2) が、“.L”のとき0～1020 (255 × 4) が指定できます。dsp:16には、サイズ拡張指定子が“.W”または“.UW”のとき0～131070 (65535 × 2) が、“.L”のとき0～262140 (65535 × 4) が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【記述例】

- OR #8, R1
 OR R1, R2
 OR [R1], R2
 OR 8[R1].L, R2
 OR R1, R2, R3

POP

スタックからレジスタへのデータ復帰
POP data from the stack

POP

転送命令

【命令コード】

記載ページ : 313

【構文】

POP dest

【オペレーション】

tmp = *SP;

SP = SP + 4;

dest = tmp;

【機能】

- スタックから復帰させたデータをdestに転送します。
- 使用されるスタックポインタは、PSWのUビットで示すスタックポインタになります。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象	コードサイズ (バイト)
		dest	
POP dest	L	Rd	2

【記述例】

POP R1

POPC

制御レジスタの復帰
POP Control register

POPC

転送命令

【命令コード】

記載ページ：314

【構文】

POPC dest

【オペレーション】

```
tmp = *SP;
SP = SP + 4;
dest = tmp;
```

【機能】

- スタックから復帰させたデータを、destで示される制御レジスタに転送します。
- 使用されるスタックポインタは、PSWのUビットで示すスタックポインタになります。
- ユーザモードでは、ISP、INTB、EXTB、BPC、BPSW、FINTVと、PSWのIPL[3:0]、PM、U、Iビットへの書き込みは無視されます。スーパーバイザモードでは、PSWのPMビットへの書き込みは無視されます。

【フラグ変化】

フラグ	変化	条件
C	(注)	
Z	(注)	
S	(注)	
O	(注)	

注. destがPSWのときだけ変化します。

【命令フォーマット】

構文	処理サイズ	対象	コードサイズ (バイト)
		dest (注)	
POPC dest	L	Rx	2

注. 選択可能なdest : ISP, USP, INTB, EXTB, PSW, BPC, BPSW, FINTV, FPSW
destにPCを指定することはできません。

【記述例】

POPC PSW

POPM

複数レジスタの復帰 POP Multiple registers

POPM

転送命令

【命令コード】

記載ページ : 314

【構文】

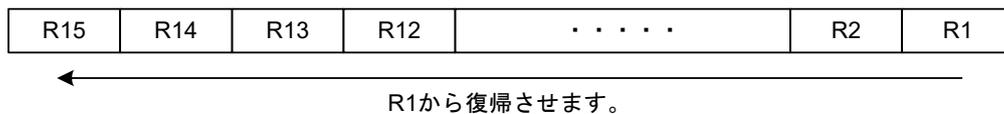
POPM dest-dest2

【オペレーション】

```
signed char i;
for ( i = register_num(dest); i <= register_num(dest2); i++ ) {
    tmp = *SP;
    SP = SP + 4;
    register(i) = tmp;
}
```

【機能】

- destとdest2で範囲指定したレジスタを一括してスタックから復帰させます。
- 範囲は、先頭レジスタ番号と最終レジスタ番号で指定します。ただし、(先頭レジスタのレジスタ番号<最終レジスタのレジスタ番号) となっている必要があります。
- R0を指定することはできません。
- 使用されるスタックポインタは、PSWのUビットで示すスタックポインタになります。
- スタックから復帰させる順序は以下のとおりです。



【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		dest	dest2	
POPM dest-dest2	L	Rd (Rd = R1 ~ R14)	Rd2 (Rd2 = R2 ~ R15)	2

【記述例】

```
POPM R1-R3
POPM R4-R8
```

PUSH

スタックヘデータ退避
PUSH data onto the stack

PUSH

【構文】

```
PUSH.size src
```

転送命令

【命令コード】

記載ページ : 315

【オペレーション】

```
tmp = src;
SP = SP - 4; (注)
*SP = tmp;
```

注. サイズ指定子 (.size) が“.B”、“.W”でもSPは4減算されます。“B”のときの上位24ビット、“W”のときの上位16ビットは不定になります。

【機能】

- srcをスタックに退避させます。
- srcがレジスタ、かつサイズ指定子が“.B”または“.W”のとき、それぞれレジスタのLSB側のバイトデータ、またはワードデータを退避させます。
- スタックへの転送サイズは、ロングワードで行います。サイズ指定子が.Bのときの上位24ビット、.Wのときの上位16ビットは不定になります。
- 使用されるスタックポインタは、PSWのUビットで示すスタックポインタになります。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	size	処理サイズ	対象	コードサイズ (バイト)
			src	
PUSH.size src	B/W/L	L	Rs	2
	B/W/L	L	[Rs]	2
	B/W/L	L	dsp:8[Rs] (注)	3
	B/W/L	L	dsp:16[Rs] (注)	4

注. 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、サイズ指定子が“.W”のときは2の倍数、“L”のときは4の倍数を指定してください。dsp:8には、サイズ指定子が“.W”のとき0～510 (255×2) が、“L”のとき0～1020 (255×4) が指定できます。dsp:16には、サイズ指定子が“.W”のとき0～131070 (65535×2) が、“L”のとき0～262140 (65535×4) が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【記述例】

```
PUSH.B R1
PUSH.L [R1]
```

PUSHC

制御レジスタの退避
PUSH Control register

PUSHC

転送命令

【命令コード】

記載ページ : 316

【構文】

PUSHC src

【オペレーション】

```
tmp = src;
SP = SP - 4;
*SP = tmp;
```

【機能】

- srcで示される制御レジスタをスタックに退避させます。
- 使用されるスタックポインタは、PSWのUビットで示すスタックポインタになります。
- srcにPCを指定した場合、本命令の番地をスタックに退避させます。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象	コードサイズ (バイト)
		src (注)	
PUSHC src	L	Rx	2

注. 選択可能なsrc : PC, ISP, USP, INTB, EXTB, PSW, BPC, BPSW, FINTV, FPSW

【記述例】

PUSHC PSW

PUSHM

複数レジスタの退避 PUSH Multiple registers

PUSHM

転送命令

【命令コード】

記載ページ：316

【構文】

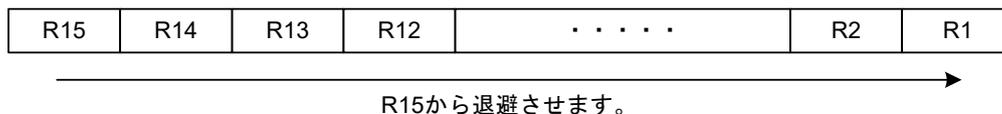
PUSHM src-src2

【オペレーション】

```
signed char i;
for ( i = register_num(src2); i >= register_num(src); i-- ) {
    tmp = register(i);
    SP = SP - 4;
    *SP = tmp;
}
```

【機能】

- srcとsrc2で範囲指定したレジスタを一括してスタックに退避させます。
- 範囲は、先頭レジスタ番号と最終レジスタ番号で指定します。ただし、(先頭レジスタのレジスタ番号<最終レジスタのレジスタ番号) となっている必要があります。
- R0を指定することはできません。
- 使用されるスタックポインタは、PSWのUビットで示すスタックポインタになります。
- スタックに退避させる順序は以下のとおりです。

**【フラグ変化】**

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	src2	
PUSHM src-src2	L	Rs (Rs = R1 ~ R14)	Rs2 (Rs2 = R2 ~ R15)	2

【記述例】

```
PUSHM R1-R3
PUSHM R4-R8
```

RACL

符号付きアキュムレータ丸め処理
Round ACcumulator Long Word

RACL

DSP機能命令

【命令コード】

記載ページ : 317

【構文】

RACL src, Adest

【オペレーション】

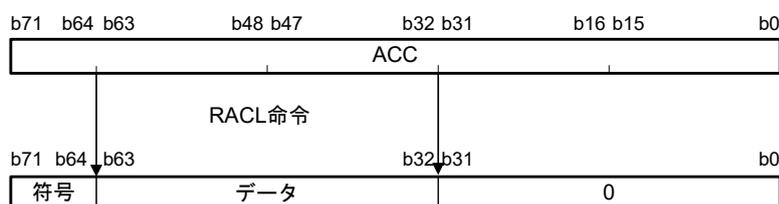
signed 72bit tmp;
signed 73bit tmp73;

tmp = (signed 72bit) Adest << src;
tmp73 = (signed 73bit) tmp + 0000000000080000000h;

if (tmp73 > (signed 73bit) 0007FFFFFFFF00000000h)
 Adest = 007FFFFFFFF00000000h;
else if (tmp73 < (signed 73bit) 1FF800000000000000h)
 Adest = FF8000000000000000h;
else
 Adest = tmp & FFFFFFFF00000000h;

【機能】

- アキュムレータの値に対してロング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。以下に動作概要図を示します。

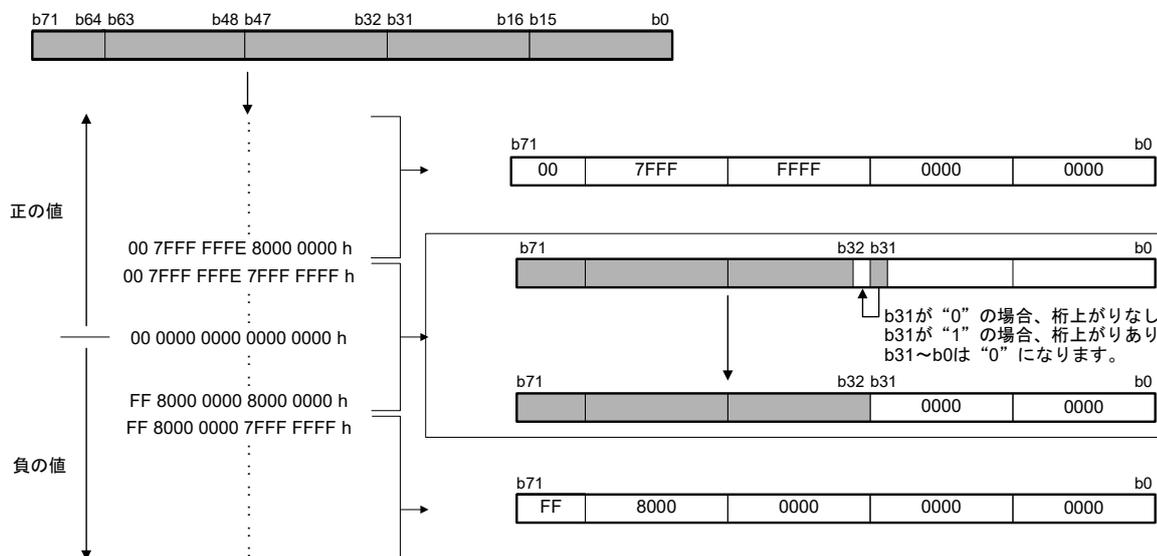


- RACL命令は、以下のような手順で実行されます。

処理1. アキュムレータの値を、srcで指定したビット数分（1ビットまたは2ビット）、左シフトします。



処理2. 1ビットまたは2ビットの左シフトを行った64ビットの値に従って、アキュムレータの値が変化します。



【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	対象		コードサイズ (バイト)
	src	Adest	
RACL src, Adest	#IMM:1 (注) (IMM:1 = 1, 2)	A0, A1	3

注. 弊社の「RXファミリアセンブラ」では、即値 (IMM:1) は、1または2を指定してください。命令コードには、-1した値が埋め込まれます。

【記述例】

RACL #1, A1
RACL #2, A0

RACW

16ビット符号付きアキュムレータ丸め処理
Round ACcumulator Word

RACW

【構文】

RACW src, Adest

DSP機能命令

【命令コード】

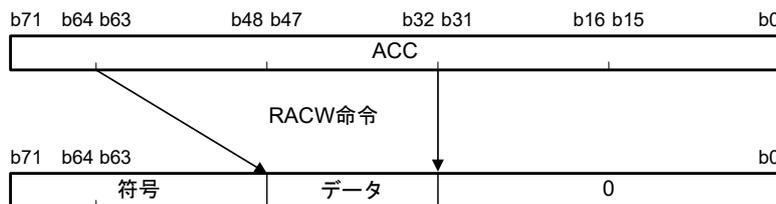
記載ページ：317

【オペレーション】

```
signed 72bit tmp;
signed 73bit tmp73;
tmp = (signed 72bit) Adest << src;
tmp73 = (signed 73bit) tmp + 000000000080000000h;
if (tmp73 > (signed 73bit) 00000007FFF0000000h)
  Adest = 0000007FFF00000000h;
else if (tmp73 < (signed 73bit) 1FFFFFF8000000000000h)
  Adest = FFFFFFF8000000000000h;
else
  Adest = tmp & FFFFFFFF00000000h;
```

【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。以下に動作概要図を示します。

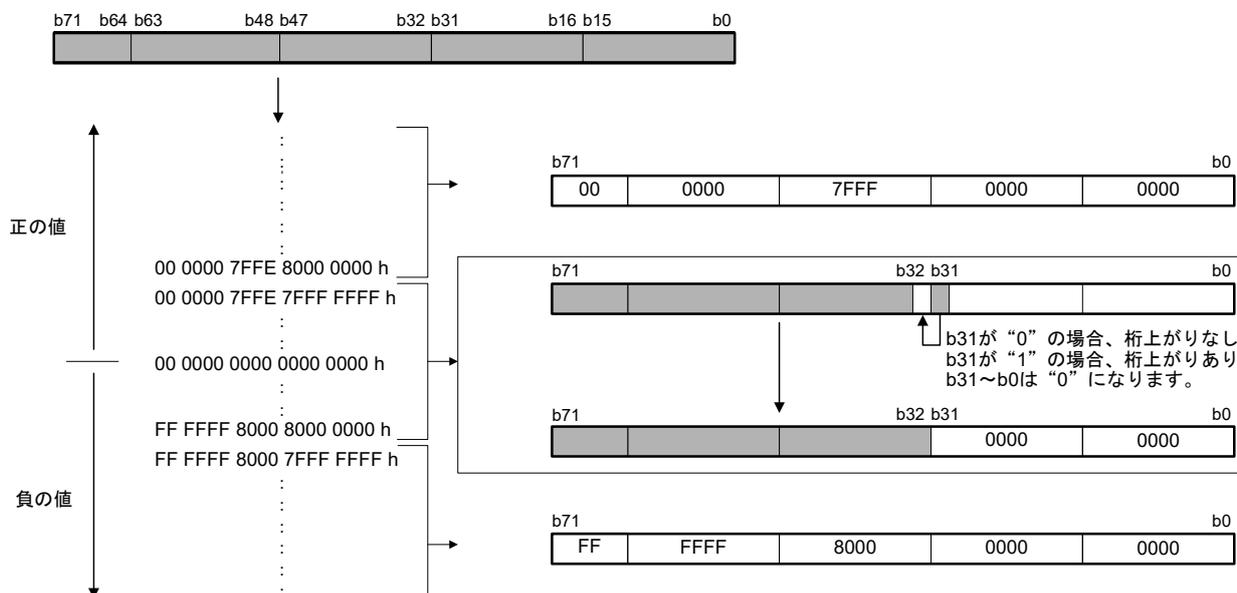


- RACW命令は、以下のような手順で実行されます。

処理1. アキュムレータの値を、srcで指定したビット数分（1ビットまたは2ビット）、左シフトします。



処理2. 1ビットまたは2ビットの左シフトを行った64ビットの値に従って、アキュムレータの値が変化します。



【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	対象		コードサイズ (バイト)
	src	Adest	
RACW src, Adest	#IMM:1 (IMM:1 = 1, 2)	A0, A1	3

【記述例】

RACW #1, A1
RACW #2, A0

RDACL

符号付きアキュムレータ丸め処理
Round Down ACcumulator Long Word

RDACL

【構文】

```
RDACL src, Adest
```

DSP機能命令

【命令コード】

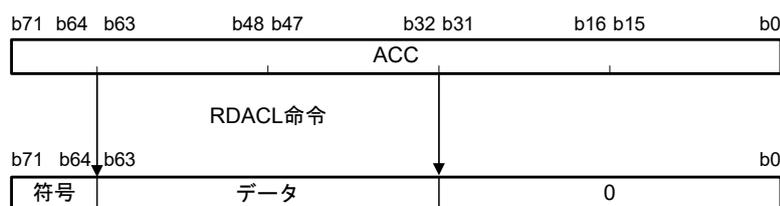
記載ページ : 318

【オペレーション】

```
signed 72bit tmp;
tmp = (signed 72bit) Adest << src;
if (tmp > (signed 72bit) 007FFFFFFFF00000000h)
    Adest = 007FFFFFFFF00000000h;
else if (tmp < (signed 72bit) FF800000000000000h)
    Adest = FF800000000000000h;
else
    Adest = tmp & FFFFFFFF00000000h;
```

【機能】

- アキュムレータの値に対してロング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。以下に動作概要図を示します。

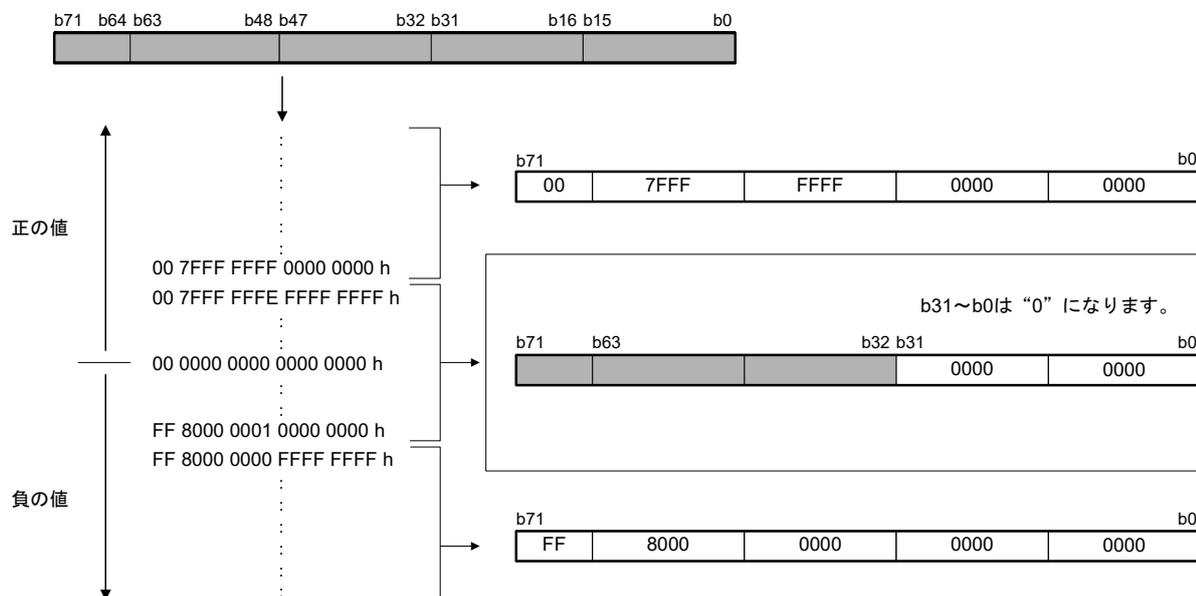


- RDACL命令は、以下のような手順で実行されます。

処理1. アキュムレータの値を、srcで指定したビット数分（1ビットまたは2ビット）、左シフトします。



処理2. 1ビットまたは2ビットの左シフトを行った64ビットの値に従って、アキュムレータの値が変化します。



【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	対象		コードサイズ (バイト)
	src	Adest	
RDACL src, Adest	#IMM:1 (IMM:1 = 1, 2)	A0, A1	3

【記述例】

- RDACL #1, A1
- RDACL #2, A0

RDACW

16ビット符号付きアキュムレータ丸め処理
Round Down ACcumulator Word

RDACW

DSP 機能命令

【命令コード】

記載ページ : 318

【構文】

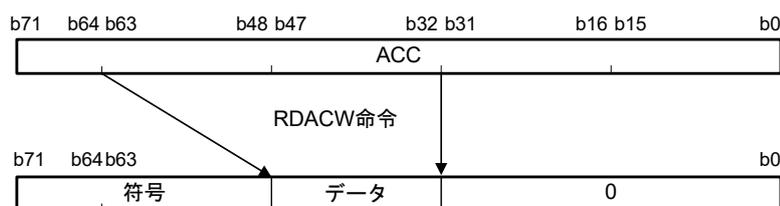
RDACW src, Adest

【オペレーション】

```
signed 72bit tmp;
tmp = (signed 72bit) Adest << src;
if (tmp > (signed 72bit) 0000007FFF00000000h)
    Adest = 0000007FFF00000000h;
else if (tmp < (signed 72bit) FFFFFFFF80000000000000h)
    Adest = FFFFFFFF80000000000000h;
else
    Adest = tmp & FFFFFFFF00000000h;
```

【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。以下に動作概要図を示します。

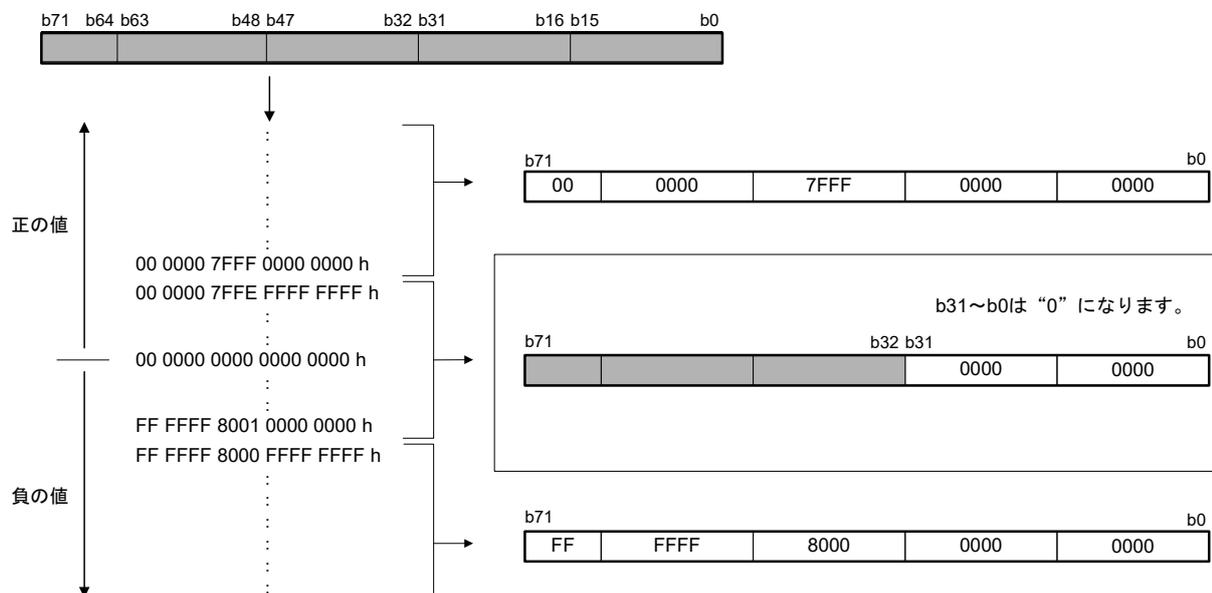


- RDACW 命令は、以下のような手順で実行されます。

処理1. アキュムレータの値を、srcで指定したビット数分（1ビットまたは2ビット）、左シフトします。



処理2. 1ビットまたは2ビットの左シフトを行った64ビットの値に従って、アキュムレータの値が変化します。



【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	対象		コードサイズ (バイト)
	src	Adest	
RDACW src, Adest	#IMM:1 (IMM:1 = 1, 2)	A0, A1	3

【記述例】

RDACW #1, A1
RDACW #2, A0

REVL

エンディアン変換
REVerse Longword data

REVL

【構文】

REVL src, dest

転送命令

【命令コード】

記載ページ : 319

【オペレーション】

Rd = { Rs[7:0], Rs[15:8], Rs[23:16], Rs[31:24] }

【機能】

- srcで指定した32ビットデータをバイト単位でエンディアン変換します。その結果をdestに格納します。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	対象		コードサイズ (バイト)
	src	dest	
REVL src, dest	Rs	Rd	3

【記述例】

REVL R1, R2

REVV

エンディアン変換
REVerse Word data

REVV

【構文】

REVV src, dest

転送命令

【命令コード】

記載ページ : 319

【オペレーション】

Rd = { Rs[23:16], Rs[31:24], Rs[7:0], Rs[15:8] }

【機能】

- src で指定した上位16ビットデータと下位16ビットデータそれぞれで、バイト単位でエンディアン変換します。その結果を dest に格納します。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	対象		コードサイズ (バイト)
	src	dest	
REVV src, dest	Rs	Rd	3

【記述例】

REVV R1, R2

RMPA

積和演算 Repeated MultiPly and Accumulate

RMPA

【構文】

RMPA.size

算術/論理演算命令

【命令コード】

記載ページ：320

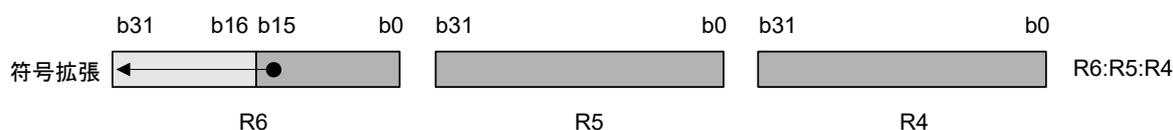
【オペレーション】

```
while ( R3 != 0 ) {
  R6:R5:R4 = R6:R5:R4 + *R1 * *R2;
  R1 = R1 + n;
  R2 = R2 + n;
  R3 = R3 - 1;
}
```

- 注.
1. R3に0を設定して実行したとき、本命令は無視され、レジスタおよびフラグは変化しません。
 2. n : サイズ指定子 (.size) が “B” のとき 1、“W” のとき 2、“L” のとき 4 になります。

【機能】

- R1を被乗数番地、R2を乗数番地、R3を回数とする積和演算を行います。演算は符号付きで行い、その結果をR6:R5:R4の80ビットに格納します。ただし、R6の上位16ビットには、下位16ビットを符号拡張した値が格納されます。
- R3に設定可能な最大値は00010000hです。



- 命令終了時のR1、R2の内容は不定となります。
- 命令実行前にR6:R5:R4には初期値を設定してください。また、R6にはR5:R4が負のときは“FFFFFFFFh”を、正のときは“00000000h”を設定してください。
- 命令実行中に割り込み要求があった場合は、演算を中断して割り込みを受け付けます。割り込みルーチンからの復帰後、中断されていた処理を継続して実行します。本命令を使用する際には、割り込み時、R1、R2、R3、R4、R5、R6とPSWを退避/復帰させてください。
- 命令実行時は、R1で示される被乗数番地とR2で示される乗数番地から、それぞれデータプリフェッチが行われる場合があります。ただし、R3で指定された範囲を超えるデータプリフェッチは行いません。プリフェッチされるデータサイズについては、各製品のユーザーズマニュアルハードウェア編を参照してください。

注. アキュムレータ (ACC0) を使用します。命令実行後のACC0の値は不定です。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	○	R6のMSBが“1”のとき“1”、それ以外るとき“0”になります。
O	○	R6:R5:R4の内容が $2^{63} - 1$ または -2^{63} を超えると“1”、それ以外るとき“0”になります。

【命令フォーマット】

構文	size	処理サイズ	コードサイズ (バイト)
RMPA.size	B/W/L	size	2

【記述例】

RMPA.W

ROLC

キャリ付き左回転
ROtate Left with Carry

ROLC

【構文】

```
ROLC dest
```

算術/論理演算命令

【命令コード】

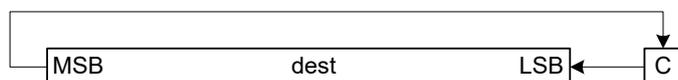
記載ページ：320

【オペレーション】

```
dest <<= 1;
if ( C == 0 )
    dest &= FFFFFFFEh;
else
    dest |= 00000001h;
```

【機能】

- Cフラグを含めて、destを1ビット左へ回転します。

**【フラグ変化】**

フラグ	変化	条件
C	○	シフトアウトしたビットが“1”のとき“1”、それ以外のとき“0”になります。
Z	○	演算後のdestが0のとき“1”、それ以外のとき“0”になります。
S	○	演算後のdestのMSBが“1”のとき“1”、それ以外のとき“0”になります。
O	—	

【命令フォーマット】

構文	処理サイズ	対象	コードサイズ (バイト)
		dest	
ROLC dest	L	Rd	2

【記述例】

```
ROLC R1
```

RORC

キャリ付き右回転
ROtate Right with Carry

RORC

【構文】

```
RORC dest
```

算術/論理演算命令

【命令コード】

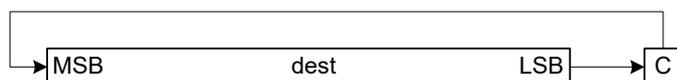
記載ページ : 320

【オペレーション】

```
dest >>= 1;
if ( C == 0 )
    dest &= 7FFFFFFFh;
else
    dest |= 80000000h;
```

【機能】

- Cフラグを含めて、destを1ビット右へ回転します。

**【フラグ変化】**

フラグ	変化	条件
C	○	シフトアウトしたビットが“1”のとき“1”、それ以外のとき“0”になります。
Z	○	演算後のdestが0のとき“1”、それ以外のとき“0”になります。
S	○	演算後のdestのMSBが“1”のとき“1”、それ以外のとき“0”になります。
O	—	

【命令フォーマット】

構文	処理サイズ	対象	コードサイズ (バイト)
		dest	
RORC dest	L	Rd	2

【記述例】

```
RORC R1
```

ROTL

左回転
ROTate Left

ROTL

【構文】

ROTL src, dest

算術/論理演算命令

【命令コード】

記載ページ : 321

【オペレーション】

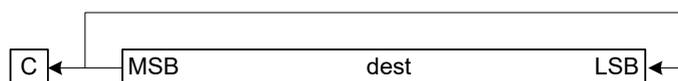
```

unsigned long tmp0, tmp1;
tmp0 = src & 31;
tmp1 = dest << tmp0;
dest = (( unsigned long ) dest >> ( 32 - tmp0 )) | tmp1;

```

【機能】

- destをsrcで指定されたビット数だけ左回転します。MSBから溢れたビットはLSBとCフラグに転送します。
- srcの値は符号なし整数です。srcの範囲は、 $0 \leq \text{src} \leq 31$ です。
- srcがレジスタのとき、LSB側5ビットのみ有効です。

**【フラグ変化】**

フラグ	変化	条件
C	○	演算後のdestのLSBと同じになります。srcが0のときも、演算後のdestのLSBと同じになります。
Z	○	演算後のdestが0のとき“1”、それ以外の場合“0”になります。
S	○	演算後のdestのMSBが“1”のとき“1”、それ以外の場合“0”になります。
O	—	

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
ROTL src, dest	L	#IMM:5	Rd	3
	L	Rs	Rd	3

【記述例】

```

ROTL #1, R1
ROTL R1, R2

```

ROTR

右回転
ROTate Right

ROTR

【構文】

```
ROTR src, dest
```

算術/論理演算命令

【命令コード】

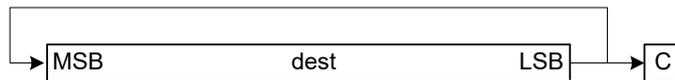
記載ページ : 321

【オペレーション】

```
unsigned long tmp0, tmp1;
tmp0 = src & 31;
tmp1 = ( unsigned long ) dest >> tmp0;
dest = ( dest << ( 32 - tmp0 ) ) | tmp1;
```

【機能】

- destをsrcで指定されたビット数だけ右回転します。LSBから溢れたビットはMSBとCフラグに転送します。
- srcの値は符号なし整数です。srcの範囲は、 $0 \leq \text{src} \leq 31$ です。
- srcがレジスタのとき、LSB側5ビットのみ有効です。

**【フラグ変化】**

フラグ	変化	条件
C	○	演算後のdestのMSBと同じになります。srcが0のときも、演算後のdestのMSBと同じになります。
Z	○	演算後のdestが0のとき“1”、それ以外の場合“0”になります。
S	○	演算後のdestのMSBが“1”のとき“1”、それ以外の場合“0”になります。
O	—	

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
ROTR src, dest	L	#IMM:5	Rd	3
	L	Rs	Rd	3

【記述例】

```
ROTR #1, R1
ROTR R1, R2
```

ROUND

単精度浮動小数点数→符号付き整数変換
ROUND floating-point to integer

ROUND

単精度浮動小数点演算命令

【命令コード】

記載ページ：322

【構文】

ROUND src, dest

【オペレーション】

dest = (signed long) src;

【機能】

- srcに格納された単精度浮動小数点数を符号付きロングワード（32ビット）整数に変換し、その結果をdestに格納します。結果はFPSWのRM[1:0]ビットに従って丸められます。

RM[1:0]ビットの値	丸めモード
00b	最近値への丸め
01b	0方向への丸め
10b	+∞方向への丸め
11b	-∞方向への丸め

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	演算の結果が“0”のとき“1”、それ以外のとき“0”になります。
S	○	演算の結果、符号部（ビット31）が“1”のとき“1”、“0”のとき“0”になります。
O	—	
CV	○	無効演算が発生したとき“1”、それ以外のとき“0”になります。
CO	○	“0”になります。
CZ	○	“0”になります。
CU	○	“0”になります。
CX	○	精度異常が発生したとき“1”、それ以外のとき“0”になります。
CE	○	非実装処理が発生したとき“1”、それ以外のとき“0”になります。
FV	○	無効演算が発生したとき“1”、それ以外のときは変化しません。
FO	—	
FZ	—	
FU	—	
FX	○	精度異常が発生したとき“1”、それ以外のときは変化しません。

注． FX、FVフラグは、例外処理許可ビットEX、EVが“1”の場合は変化しません。S、Zフラグは、例外処理が発生した場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
ROUND src, dest	L	Rs	Rd	3
	L	[Rs].L	Rd	3
	L	dsp:8[Rs].L (注)	Rd	4
	L	dsp:16[Rs].L (注)	Rd	5

注. 弊社の「RXファミリ アセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、4の倍数を指定してください。dsp:8には、0～1020 (255×4) が指定できます。dsp:16には、0～262140 (65535×4) が指定できます。命令コードには、1/4した値が埋め込まれます。

【単精度浮動小数点例外発生要因】

非実装処理
無効演算
精度異常

【記述例】

ROUND R1, R2
ROUND [R1], R2

【動作補足説明】

- FPSWのDNビットが“0”の場合と“1”の場合について、src、destの値と演算結果の対応を以下に示します。

DN = 0のとき

srcの値 (指数部はゲタなしの値)		dest	例外
src ≥ 0	+∞	EV = 1で無効演算発生時: 変化なし 上記以外: 7FFFFFFFh	無効演算
	31 ≤ 指数部 ≤ 127		
	-126 ≤ 指数部 ≤ 30	00000000h ~ 7FFFFFFF80h	なし (注1)
	+非正規化数	変化なし	非実装
	+0	00000000h	なし
src < 0	-0		
	-非正規化数	変化なし	非実装
	-126 ≤ 指数部 ≤ 30	00000000h ~ 80000080h	なし (注1)
	31 ≤ 指数部 ≤ 127	EV = 1で無効演算発生時: 変化なし 上記以外: 80000000h	無効演算 (注2)
	-∞		
NaN	QNaN	EV = 1で無効演算発生時: 変化なし 上記以外:	無効演算
	SNaN	符号ビット = 0: 7FFFFFFFh 符号ビット = 1: 80000000h	

注1. 丸め発生時は、精度異常が発生します。

注2. src = CF000000hのとき、無効演算は発生しません。

DN = 1のとき

srcの値 (指数部はゲタなしの値)		dest	例外
src ≥ 0	+∞	EV = 1で無効演算発生時: 変化なし 上記以外: 7FFFFFFFh	無効演算
	31 ≤ 指数部 ≤ 127		
	-126 ≤ 指数部 ≤ 30	00000000h ~ 7FFFFFF80h	なし (注1)
	+0, +非正規化数	00000000h	なし
src < 0	-0, -非正規化数		
	-126 ≤ 指数部 ≤ 30	00000000h ~ 80000080h	なし (注1)
	31 ≤ 指数部 ≤ 127	EV = 1で無効演算発生時: 変化なし 上記以外: 80000000h	無効演算 (注2)
	-∞		
NaN	QNaN	EV = 1で無効演算発生時: 変化なし 上記以外:	無効演算
	SNaN	符号ビット = 0: 7FFFFFFFh 符号ビット = 1: 80000000h	

注1. 丸め発生時は、精度異常が発生します。

注2. src = CF000000hのとき、無効演算は発生しません。

RTE例外からの復帰
ReTurn from Exception**RTE**

【構文】

RTE

システム操作命令

【命令コード】

記載ページ：322

【オペレーション】

```
PC = *SP;
SP = SP + 4;
tmp = *SP;
SP = SP + 4;
PSW = tmp;
LI = 0
```

【機能】

- 例外が受け付けられたときに退避させたPCとPSWを復帰させ、例外処理ルーチンから戻ります。
- この命令は特権命令です。ユーザモードで実行すると特権命令例外が発生します。
- ユーザモードに移行する場合、PSWのUビットは“1”になります。

【フラグ変化】

フラグ	変化	条件
C	(注)	
Z	(注)	
S	(注)	
O	(注)	

注. スタック上の値になります。

【命令フォーマット】

構文	コードサイズ (バイト)
RTE	2

【記述例】

RTE

RTFI

高速割り込みからの復帰 ReTurn from Fast Interrupt

RTFI

【構文】

RTFI

システム操作命令

【命令コード】

記載ページ：322

【オペレーション】

PSW = BPSW;

PC = BPC;

LI = 0

【機能】

- 高速割り込み要求を受け付けたときに退避させたPCとPSWを、それぞれBPC、BPSWから復帰させ、高速割り込み処理ルーチンから戻ります。
- この命令は特権命令です。ユーザモードで実行すると特権命令例外が発生します。
- ユーザモードに移行する場合、PSWのUビットは“1”になります。
- 命令終了時のBPC、BPSWの値は不定になります。

【フラグ変化】

フラグ	変化	条件
C	(注)	
Z	(注)	
S	(注)	
O	(注)	

注. BPSWの値になります。

【命令フォーマット】

構文	コードサイズ (バイト)
RTFI	2

【記述例】

RTFI

RTS

サブルーチンからの復帰
ReTurn from Subroutine

RTS

【構文】

RTS

分岐命令

【命令コード】

記載ページ：323

【オペレーション】

PC = *SP;

SP = SP + 4;

【機能】

- サブルーチンから復帰します。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	コードサイズ (バイト)
RTS	1

【記述例】

RTS

RTSD

スタックフレームの解放とサブルーチンからの復帰

ReTurn from Subroutine and
Deallocate stack frame

RTSD

分岐命令

【命令コード】

記載ページ : 323

【構文】

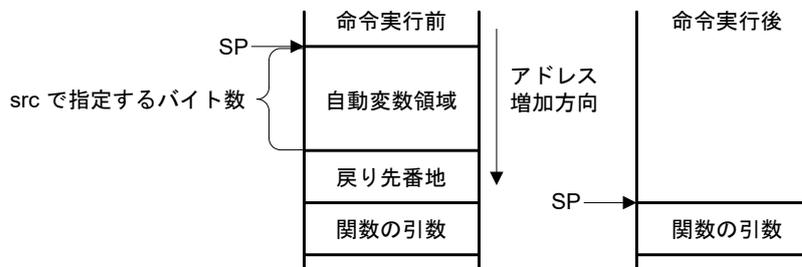
- (1) RTS D  src
- (2) RTS D  src, dest-dest2

【オペレーション】

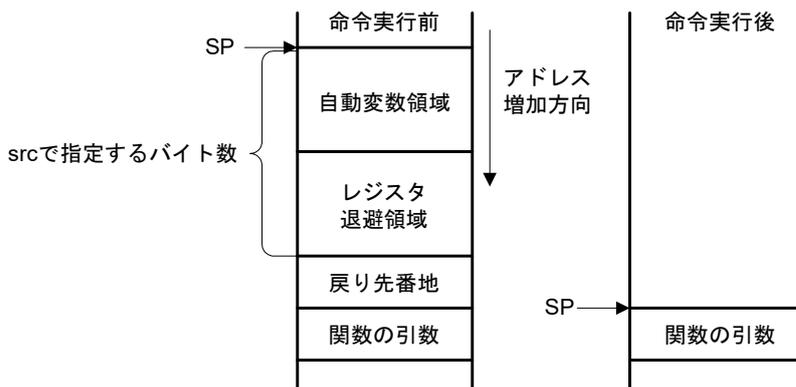
- (1) $SP = SP + src;$
 $PC = *SP;$
 $SP = SP + 4;$
- (2) signed char i;
 $SP = SP + (src - (register_num(dest2) - register_num(dest) + 1) * 4);$
for (i = register_num(dest); i <= register_num(dest2); i++) {
 tmp = *SP;
 SP = SP + 4;
 register(i) = tmp;
}
 $PC = *SP;$
 $SP = SP + 4;$

【機能】

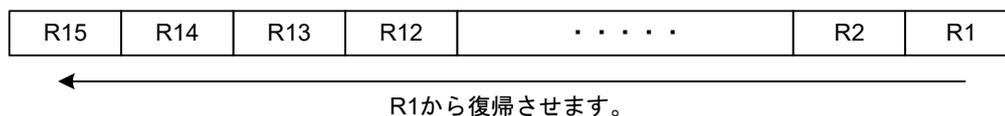
- (1) スタックフレームを解放後、サブルーチンから復帰します。
 - srcがスタックフレーム（自動変数領域）のサイズになるように指定してください。



- (2) スタックフレームの解放とレジスタの復帰を行った後、サブルーチンから復帰します。
 - srcがスタックフレーム（自動変数領域とレジスタ退避領域）のサイズになるように指定してください。



- destとdest2で範囲指定したレジスタを一括してスタックから復帰させます。
- 範囲は先頭レジスタ番号と最終レジスタ番号で指定します。ただし、(先頭レジスタのレジスタ番号 ≤ 最終レジスタのレジスタ番号) となっている必要があります。
- R0を指定することはできません。
- 使用されるスタックポインタは、PSWのUビットで示すスタックポインタになります。
- スタックから復帰させる順序は以下のとおりです。



【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	対象			コードサイズ (バイト)
	src	dest	dest2	
(1) RTSD src	#UIMM:8 (注)	—	—	2
(2) RTSD src, dest-dest2	#UIMM:8 (注)	Rd (Rd = R1 ~ R15)	Rd2 (Rd2 = R1 ~ R15)	3

注. 弊社の「RXファミリ アセンブラ」では、即値は、4の倍数を指定してください。UIMM:8には、0～1020 (255×4) が指定できます。命令コードには、1/4した値が埋め込まれます。

【記述例】

RTSD #4

RTSD #16, R5-R7

SAT

32ビット符号付き飽和処理
SATurate signed 32-bit data

SAT

【構文】

```
SAT dest
```

算術/論理演算命令

【命令コード】

記載ページ：323

【オペレーション】

```
if ( O == 1 && S == 1 )
    dest = 7FFFFFFFh;
else if ( O == 1 && S == 0 )
    dest = 80000000h;
```

【機能】

- 32ビット符号付きで飽和処理を行います。
- Oフラグが“1”かつSフラグが“1”のとき、演算結果が7FFFFFFFhになり、その結果をdestに格納します。Oフラグが“1”かつSフラグが“0”のとき、演算結果が80000000hになり、その結果をdestに格納します。それ以外のとき、destは変化しません。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象	コードサイズ (バイト)
		dest	
SAT dest	L	Rd	2

【記述例】

```
SAT R1
```

SATR

RMPA 命令用64ビット符号付き飽和处理
SATuRate signed 64-bit data for RMPA

SATR

【構文】

SATR

算術/論理演算命令

【命令コード】

記載ページ : 324

【オペレーション】

```
if ( O == 1 && S == 0 )
    R6:R5:R4 = 000000007FFFFFFFFFFFFFFFh;
else if ( O == 1 && S == 1 )
    R6:R5:R4 = FFFFFFFF8000000000000000h;
```

【機能】

- 64ビット符号付きで飽和处理を行います。
- Oフラグが“1”かつSフラグが“0”のとき、演算結果が000000007FFFFFFFFFFFFFFFhになり、その結果をR6:R5:R4に格納します。Oフラグが“1”かつSフラグが“1”のとき、演算結果がFFFFFFFF8000000000000000hになり、その結果をR6:R5:R4に格納します。それ以外のとき、R6:R5:R4は変化しません。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	コードサイズ (バイト)
SATR	2

【記述例】

SATR

SBBボロー付き減算
SuBtract with Borrow**SBB**

【構文】

SBB src, dest

算術/論理演算命令

【命令コード】

記載ページ：324

【オペレーション】

dest = dest - src - !C;

【機能】

- destからsrcとCフラグの反転（ボロー）を減算し、その結果をdestに格納します。

【フラグ変化】

フラグ	変化	条件
C	○	符号なし演算のオーバーフローが発生しなかったとき“1”、それ以外るとき“0”になります。
Z	○	演算後のdestが0のとき“1”、それ以外るとき“0”になります。
S	○	演算後のdestのMSBが“1”のとき“1”、それ以外るとき“0”になります。
O	○	符号付き演算のオーバーフローが発生したとき“1”、それ以外るとき“0”になります。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
SBB src, dest	L	Rs	Rd	3
	L	[Rs].L	Rd	4
	L	dsp:8[Rs].L (注)	Rd	5
	L	dsp:16[Rs].L (注)	Rd	6

注. 弊社の「RXファミリ アセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、4の倍数を指定してください。dsp:8には、0～1020 (255×4) が指定できます。dsp:16には、0～262140 (65535×4) が指定できます。命令コードには、1/4した値が埋め込まれます。

【記述例】

SBB R1, R2

SBB [R1], R2

SCCnd

条件設定 Store Condition Conditionally

SCCnd

【構文】

```
SCCnd.size dest
```

転送命令

【命令コード】

記載ページ：325

【オペレーション】

```
if ( Cnd )
    dest = 1;
else
    dest = 0;
```

【機能】

- Cndで示す条件の真偽値をdestに設定します。真の場合は“1”を、偽の場合は“0”を設定します。
- SCCndには次の種類があります。

SCCnd	条件		式	SCCnd	条件		式
SCGEU, SCC	C == 1	等しいまたは大きい/ Cフラグが“1”	≤	SCLTU, SCNC	C == 0	小さい/ Cフラグが“0”	>
SCEQ, SCZ	Z == 1	等しい/ Zフラグが“1”	=	SCNE, SCNZ	Z == 0	等しくない/ Zフラグが“0”	≠
SCGTU	(C & ~Z) == 1	大きい	<	SCLEU	(C & ~Z) == 0	等しいまたは小さい	≥
SCPZ	S == 0	正またゼロ	0 ≤	SCN	S == 1	負	0 >
SCGE	(S ^ O) == 0	等しい、または符号付き で大きい	≤	SCLE	((S ^ O) Z) == 1	等しい、または符号付き で小さい	≥
SCGT	((S ^ O) Z) == 0	符号付きで大きい	<	SCLT	(S ^ O) == 1	符号付きで小さい	>
SCO	O == 1	Oフラグが“1”		SCNO	O == 0	Oフラグが“0”	

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	size	処理サイズ	対象	コードサイズ (バイト)
			dest	
SCCnd.size dest	L	L	Rd	3
	B/W/L	size	[Rd]	3
	B/W/L	size	dsp:8[Rd] (注)	4
	B/W/L	size	dsp:16[Rd] (注)	5

注. 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、サイズ指定子が“.W”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:8には、サイズ指定子が“.W”のとき0～510 (255×2) が、“.L”のとき0～1020 (255×4) が指定できます。dsp:16には、サイズ指定子が“.W”のとき0～131070 (65535×2) が、“.L”のとき0～262140 (65535×4) が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【記述例】

```
SCC.L R2
SCNE.W [R2]
```

SCMPU

ストリング比較
String CoMPare Until not equal

SCMPU

【構文】

SCMPU

ストリング操作命令

【命令コード】

記載ページ：325

【オペレーション】

```
unsigned char *R2, *R1, tmp0, tmp1;
unsigned long R3;
while ( R3 != 0 ) {
    tmp0 = *R1++;
    tmp1 = *R2++;
    R3--;
    if ( tmp0 != tmp1 || tmp0 == '\0' ) {
        break;
    }
}
```

注． R3に0を設定して実行したとき、本命令は無視され、レジスタおよびフラグは変化しません。

【機能】

- R1で示される比較元番地とR2で示される比較先番地のデータを、比較の結果が不一致になるか、Nullキャラクタ‘\0’ (= 00h) が検出されるまで、R3で指定されたバイト数を上限として、アドレスの加算方向にストリング比較を行います。
- 命令実行時は、R1で示される比較元番地とR2で示される比較先番地から、それぞれデータプリフェッチが行われる場合があります。ただし、R3で指定された範囲を超えるデータプリフェッチは行いません。プリフェッチされるデータサイズについては、各製品のユーザーズマニュアルハードウェア編を参照してください。
- 命令終了時のR1、R2は不定になります。
- 命令実行中に割り込み要求があった場合は、演算を中断して割り込みを受け付けます。割り込みルーチンからの復帰後、中断されていた処理を継続して実行します。本命令を使用する際には、割り込み時、R1、R2、R3とPSWを退避/復帰させてください。

【フラグ変化】

フラグ	変化	条件
C	○	(*R1 - *R2) を符号なしで演算した結果、0以上のとき“1”、それ以外るとき“0”になります。
Z	○	双方のストリングが一致していたとき“1”、それ以外るとき“0”になります。
S	—	
O	—	

【命令フォーマット】

構文	処理サイズ	コードサイズ (バイト)
SCMPU	B	2

【記述例】

SCMPU

SETPSW

PSWのフラグ、ビットのセット
SET flag of PSW

SETPSW

システム操作命令

【命令コード】

記載ページ : 326

【構文】

SETPSW dest

【オペレーション】

dest = 1;

【機能】

- destで指定されたO、S、Z、Cフラグ、もしくはU、Iビットを“1”にします。
- ユーザモードでは、PSWのU、Iビットへの書き込みは無視されます。スーパーバイザモードでは、すべてのフラグとビットへの書き込みが行えます。

【フラグ変化】

フラグ	変化	条件
C	(注)	
Z	(注)	
S	(注)	
O	(注)	

注. 指定されたフラグが“1”になります。

【命令フォーマット】

構文	対象	コードサイズ (バイト)
	dest	
SETPSW dest	flag	2

【記述例】

SETPSW C

SETPSW Z

SHAR

算術右シフト
SHift Arithmetic Right

SHAR

【構文】

- (1) SHAR src, dest
- (2) SHAR src, src2, dest

算術/論理演算命令

【命令コード】

記載ページ：327

【オペレーション】

- (1) $dest = (\text{signed long}) dest \gg (\text{src} \& 31);$
- (2) $dest = (\text{signed long}) src2 \gg (\text{src} \& 31);$

【機能】

- (1) destをsrcで指定されたビット数分、算術右シフトし、その結果をdestに格納します。
 - LSBから溢れたビットはCフラグに転送します。
 - srcの値は符号なし整数です。srcの範囲は、 $0 \leq \text{src} \leq 31$ です。
 - srcがレジスタのとき、LSB側5ビットのみ有効です。
- (2) src2をdestに転送後、destをsrcで指定されたビット数分、算術右シフトし、その結果をdestに格納します。
 - LSBから溢れたビットはCフラグに転送します。
 - srcの値は符号なし整数です。srcの範囲は、 $0 \leq \text{src} \leq 31$ です。

**【フラグ変化】**

フラグ	変化	条件
C	○	シフトアウトしたビットが“1”のとき“1”、それ以外の場合“0”になります。ただし、srcが0のときは“0”になります。
Z	○	演算後のdestが0のとき“1”、それ以外の場合“0”になります。
S	○	演算後のdestのMSBが“1”のとき“1”、それ以外の場合“0”になります。
O	○	“0”になります。

【命令フォーマット】

構文	処理サイズ	対象			コードサイズ (バイト)
		src	src2	dest	
(1) SHAR src, dest	L	#IMM:5	—	Rd	2
	L	Rs	—	Rd	3
(2) SHAR src, src2, dest	L	#IMM:5	Rs	Rd	3

【記述例】

```
SHAR #3, R2
SHAR R1, R2
SHAR #3, R1, R2
```

SHLL

論理/算術左シフト
SHift Logical and arithmetic Left

SHLL

【構文】

- (1) SHLL src, dest
- (2) SHLL src, src2, dest

算術/論理演算命令

【命令コード】

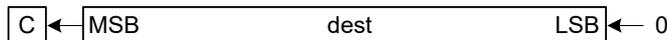
記載ページ：328

【オペレーション】

- (1) dest = dest << (src & 31);
- (2) dest = src2 << (src & 31);

【機能】

- (1) destをsrcで指定されたビット数分、論理左シフトし、その結果をdestに格納します。
 - MSBから溢れたビットはCフラグに転送します。
 - srcがレジスタのとき、LSB側5ビットのみ有効です。
 - srcの値は符号なし整数です。srcの範囲は、 $0 \leq \text{src} \leq 31$ です。
- (2) src2をdestに転送後、destをsrcで指定されたビット数分、論理左シフトし、その結果をdestに格納します。
 - MSBから溢れたビットはCフラグに転送します。
 - srcの値は符号なし整数です。srcの範囲は、 $0 \leq \text{src} \leq 31$ です。

**【フラグ変化】**

フラグ	変化	条件
C	○	シフトアウトしたビットが“1”のとき“1”、それ以外のとき“0”になります。ただし、srcが0のときは“0”になります。
Z	○	演算後のdestが0のとき“1”、それ以外のとき“0”になります。
S	○	演算後のdestのMSBが“1”のとき“1”、それ以外のとき“0”になります。
O	○	演算結果のMSBとシフトアウトしたビットがすべて同じ値のとき（シフト中に符号が変化しなかったとき）“0”、それ以外のとき“1”になります。ただし、srcが0のときは“0”になります。

【命令フォーマット】

構文	処理サイズ	対象			コードサイズ (バイト)
		src	src2	dest	
(1) SHLL src, dest	L	#IMM:5	—	Rd	2
	L	Rs	—	Rd	3
(2) SHLL src, src2, dest	L	#IMM:5	Rs	Rd	3

【記述例】

```
SHLL #3, R2
SHLL R1, R2
SHLL #3, R1, R2
```

SHLR

論理右シフト
SHift Logical Right

SHLR

【構文】

- (1) SHLR src, dest
- (2) SHLR src, src2, dest

算術/論理演算命令

【命令コード】

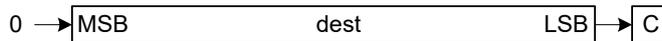
記載ページ : 329

【オペレーション】

- (1) $dest = (unsigned\ long)\ dest \gg (src \& 31);$
- (2) $dest = (unsigned\ long)\ src2 \gg (src \& 31);$

【機能】

- (1) destをsrcで指定されたビット数分、論理右シフトし、その結果をdestに格納します。
 - LSBから溢れたビットはCフラグに転送します。
 - srcの値は符号なし整数です。srcの範囲は、 $0 \leq src \leq 31$ です。
 - srcがレジスタのとき、LSB側5ビットのみ有効です。
- (2) src2をdestに転送後、destをsrcで指定されたビット数分、論理右シフトし、その結果をdestに格納します。
 - LSBから溢れたビットはCフラグに転送します。
 - srcの値は符号なし整数です。srcの範囲は、 $0 \leq src \leq 31$ です。

**【フラグ変化】**

フラグ	変化	条件
C	○	シフトアウトしたビットが“1”のとき“1”、それ以外のとき“0”になります。ただし、srcが0のときは“0”になります。
Z	○	演算後のdestが0のとき“1”、それ以外のとき“0”になります。
S	○	演算後のdestのMSBが“1”のとき“1”、それ以外のとき“0”になります。
O	—	

【命令フォーマット】

構文	処理サイズ	対象			コードサイズ (バイト)
		src	src2	dest	
(1) SHLR src, dest	L	#IMM:5	—	Rd	2
	L	Rs	—	Rd	3
(2) SHLR src, src2, dest	L	#IMM:5	Rs	Rd	3

【記述例】

```
SHLR #3, R2
SHLR R1, R2
SHLR #3, R1, R2
```

SMOVB

逆方向ストリング転送
Strings MOVE Backward

SMOVB

【構文】

SMOVB

ストリング操作命令

【命令コード】

記載ページ：329

【オペレーション】

```

unsigned char *R1, *R2;
unsigned long R3;
while ( R3 != 0 ) {
    *R1-- = *R2--;
    R3 = R3 - 1;
}

```

注． R3に0を設定して実行したとき、本命令は無視され、レジスタおよびフラグは変化しません。

【機能】

- R3で指定されたバイト数分、R2で示される転送元番地からR1で示される転送先番地へ、アドレス減算方向にストリング転送を行います。
- 命令実行時は、R2で示される転送元番地からデータプリフェッチが行われる場合があります。ただし、R3で指定された範囲を超えるデータプリフェッチは行いません。プリフェッチされるデータサイズについては、各製品のユーザーズマニュアルハードウェア編を参照してください。
- R2で示される転送元番地からデータプリフェッチされる範囲にR1で示される転送先番地が含まれない条件で使用してください。
- 命令終了時のR1、R2は、最後に転送したデータの次の番地を示します。
- 命令実行中に割り込み要求があった場合は、命令途中で転送を中断して割り込みを受け付けます。割り込みルーチンからの復帰後、中断されていた処理を継続して実行します。本命令を使用する際には、割り込み時、R1、R2、R3とPSWを退避/復帰させてください。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	コードサイズ (バイト)
SMOVB	B	2

【記述例】

SMOVB

SMOVF

順方向ストリング転送
Strings MOVE Forward

SMOVF

ストリング操作命令

【命令コード】

記載ページ：330

【構文】

SMOVF

【オペレーション】

```
unsigned char *R1, *R2;
unsigned long R3;
while ( R3 != 0 ) {
    *R1++ = *R2++;
    R3 = R3 - 1;
}
```

注. R3に0を設定して実行したとき、本命令は無視され、レジスタおよびフラグは変化しません。

【機能】

- R3で指定されたバイト数分、R2で示される転送元番地からR1で示される転送先番地へ、アドレス加算方向にストリング転送を行います。
- 命令実行時は、R2で示される転送元番地からデータプリフェッチが行われる場合があります。ただし、R3で指定された範囲を超えるデータプリフェッチは行いません。プリフェッチされるデータサイズについては、各製品のユーザーズマニュアルハードウェア編を参照してください。
- R2で示される転送元番地からデータプリフェッチされる範囲にR1で示される転送先番地が含まれない条件で使用してください。
- 命令終了時のR1、R2は、最後に転送したデータの次の番地を示します。
- 命令実行中に割り込み要求があった場合は、命令途中で転送を中断して割り込みを受け付けます。割り込みルーチンからの復帰後、中断されていた処理を継続して実行します。本命令を使用する際には、割り込み時、R1、R2、R3とPSWを退避/復帰させてください。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	コードサイズ (バイト)
SMOVF	B	2

【記述例】

SMOVF

SMOVU

ストリング転送
Strings MOVE while Unequal to zero

SMOVU

ストリング操作命令

【命令コード】

記載ページ：330

【構文】

SMOVU

【オペレーション】

```
unsigned char *R1, *R2, tmp;
unsigned long R3;
while ( R3 != 0 ) {
    tmp = *R2++;
    *R1++ = tmp;
    R3--;
    if ( tmp == '\0' ) {
        break;
    }
}
```

注． R3に0を設定して実行したとき、本命令は無視され、レジスタおよびフラグは変化しません。

【機能】

- R2で示される転送元番地からR1で示される転送先番地へNullキャラクタ‘\0’ (=00h) が検出されるまで、R3で指定されたバイト数を上限として、アドレス加算方向にストリング転送を行います。転送はNullキャラクタ転送後に終了します。
- 命令実行時は、R2で示される転送元番地からデータプリフェッチが行われる場合があります。ただし、R3で指定された範囲を超えるデータプリフェッチは行いません。プリフェッチされるデータサイズについては、各製品のユーザーズマニュアルハードウェア編を参照してください。
- R2で示される転送元番地からデータプリフェッチされる範囲にR1で示される転送先番地が含まれない条件で使用してください。
- 命令終了時のR1、R2は、不定となります。
- 命令実行中に割り込み要求があった場合は、命令途中で転送を中断して割り込みを受け付けます。割り込みルーチンからの復帰後、中断されていた処理を継続して実行します。本命令を使用する際には、割り込み時、R1、R2、R3とPSWを退避/復帰させてください。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	コードサイズ (バイト)
SMOVU	B	2

【記述例】

SMOVU

SSTR

ストリングストア
Strings SToRe

SSTR

【構文】

SSTR.size

ストリング操作命令

【命令コード】

記載ページ：330

【オペレーション】

```
unsigned { char | short | long } *R1, R2;
unsigned long R3;
while ( R3 != 0 ) {
    *R1++ = R2;
    R3 = R3 - 1;
}
```

- 注 .
1. R3に0を設定して実行したとき、本命令は無視され、レジスタおよびフラグは変化しません。
 2. R1++ : サイズ指定子 (.size) が “.B” のとき 1、“.W” のとき 2、“.L” のとき 4 が加算されます。
 3. R2 : サイズ指定子 (.size) が “.B” のとき R2 の LSB 側バイトデータ、“.W” のとき R2 の LSB 側ワードデータ、“.L” のとき R2 のロングワードデータがストアされます。

【機能】

- R3 で示される回数分、R2 の内容を R1 で示される転送先番地へ、アドレス加算方向にストリングストアを行います。
- 命令終了時の R1 は、最後に転送したデータの次の番地を示します。
- 命令実行中に割り込み要求があった場合は、命令途中で転送を中断して割り込みを受け付けます。割り込みルーチンからの復帰後、中断されていた処理を継続して実行します。本命令を使用する際には、割り込み時、R1、R2、R3 と PSW を退避/復帰させてください。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	size	処理サイズ	コードサイズ (バイト)
SSTR.size	B/W/L	size	2

【記述例】

SSTR.W

STNZ

条件付き転送
STore on Not Zero

STNZ

【構文】

```
STNZ src, dest
```

転送命令

【命令コード】

記載ページ : 331

【オペレーション】

```
if ( Z == 0 )
    dest = src;
```

【機能】

- Zフラグが“0”のとき、srcをdestに転送します。“1”のときdestは変化しません。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
STNZ src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3

【記述例】

```
STNZ #1, R2
STNZ R1, R2
```

STZ

条件付き転送
STore on Zero

STZ

転送命令

【命令コード】

記載ページ：332

【構文】

STZ src, dest

【オペレーション】

```
if ( Z == 1 )
    dest = src;
```

【機能】

- Zフラグが“1”のとき、srcをdestに転送します。“0”のとき、destは変化しません。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
STZ src, dest	L	#SIMM:8	Rd	4
	L	#SIMM:16	Rd	5
	L	#SIMM:24	Rd	6
	L	#IMM:32	Rd	7
	L	Rs	Rd	3

【記述例】

STZ #1, R2

STZ R1, R2

SUB

ボローなし減算
SUBtract

SUB

【構文】

- (1) SUB src, dest
 (2) SUB src, src2, dest

算術/論理演算命令

【命令コード】

記載ページ : 333

【オペレーション】

- (1) dest = dest - src;
 (2) dest = src2 - src;

【機能】

- (1) destからsrcを減算し、その結果をdestに格納します。
 (2) src2からsrcを減算し、その結果をdestに格納します。

【フラグ変化】

フラグ	変化	条件
C	○	符号なし演算のオーバーフローが発生しなかったとき“1”、それ以外るとき“0”になります。
Z	○	演算後のdestが0のとき“1”、それ以外るとき“0”になります。
S	○	演算後のdestのMSBが“1”のとき“1”、それ以外るとき“0”になります。
O	○	符号付き演算のオーバーフローが発生したとき“1”、それ以外るとき“0”になります。

【命令フォーマット】

構文	処理サイズ	対象			コードサイズ (バイト)
		src	src2	dest	
(1) SUB src, dest	L	#UIMM:4	—	Rd	2
	L	Rs	—	Rd	2
	L	[Rs].memex	—	Rd	2 (memex == “UB”) 3 (memex != “UB”)
	L	dsp:8[Rs].memex (注)	—	Rd	3 (memex == “UB”) 4 (memex != “UB”)
	L	dsp:16[Rs].memex (注)	—	Rd	4 (memex == “UB”) 5 (memex != “UB”)
(2) SUB src, src2, dest	L	Rs	Rs2	Rd	3

注. 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、サイズ拡張指定子が“.W”または“.UW”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:8には、サイズ拡張指定子が“.W”または“.UW”のとき0～510 (255 × 2) が、“.L”のとき0～1020 (255 × 4) が指定できます。dsp:16には、サイズ拡張指定子が“.W”または“.UW”のとき0～131070 (65535 × 2) が、“.L”のとき0～262140 (65535 × 4) が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【記述例】

SUB #15, R2
 SUB R1, R2
 SUB [R1], R2
 SUB 1[R1].B, R2
 SUB R1, R2, R3

SUNTIL

ストリングサーチ Search UNTIL equal string

SUNTIL

【構文】

SUNTIL.size

ストリング操作命令

【命令コード】

記載ページ：334

【オペレーション】

```

unsigned { char | short | long } *R1;
unsigned long R2, R3, tmp;
while ( R3 != 0 ) {
    tmp = ( unsigned long ) *R1++;
    R3--;
    if ( tmp == R2 ) {
        break;
    }
}

```

- 注．
1. R3に0を設定して実行したとき、本命令は無視され、レジスタおよびフラグは変化しません。
 2. R1++：サイズ指定子（.size）が“.B”のとき1、“.W”のとき2、“.L”のとき4が加算されます。

【機能】

- R1で示される比較先番地からアドレスの加算方向に、R2の内容と一致するデータが現れるまで、R3で指定される回数を上限として検索を行います。サイズ指定子（.size）が“.B”または“.W”のときは、メモリのバイトデータまたはワードデータをロングワードデータにゼロ拡張し、R2の内容と比較を行います。
- 命令実行時は、R1で示される比較先番地からデータプリフェッチが行われる場合があります。ただし、R3で指定された範囲を超えるデータプリフェッチは行いません。プリフェッチされるデータサイズについては、各製品のユーザーズマニュアルハードウェア編を参照してください。
- フラグは「*R1 - R2」の演算結果に従って変化します。
- 命令終了時のR1は、一致したデータの次の番地を示します。すべて一致しなかったときは、最後に転送したデータの次の番地を示します。
- 命令終了後のR3は、「初期値 - 比較回数」となります。
- 命令実行中に割り込み要求があった場合は、命令途中で転送を中断して割り込みを受け付けます。割り込みルーチンからの復帰後、中断されていた処理を継続して実行します。本命令を使用する際には、割り込み時、R1、R2、R3とPSWを退避/復帰させてください。

【フラグ変化】

フラグ	変化	条件
C	○	符号なし比較の結果、0以上のとき“1”、それ以外のとき“0”になります。
Z	○	一致したとき“1”、それ以外のとき“0”になります。
S	—	
O	—	

【命令フォーマット】

構文	size	処理サイズ	コードサイズ (バイト)
SUNTIL.size	B/W/L	L	2

【記述例】

SUNTIL.W

SWHILE

ストリングサーチ
Search WHILE unequal string

SWHILE

ストリング操作命令

【命令コード】

記載ページ：334

【構文】

SWHILE.size

【オペレーション】

```
unsigned { char | short | long } *R1;
unsigned long R2, R3, tmp;
while ( R3 != 0 ) {
    tmp = ( unsigned long ) *R1++;
    R3--;
    if ( tmp != R2 ) {
        break;
    }
}
```

- 注． 1. R3に0を設定して実行したとき、本命令は無視され、レジスタおよびフラグは変化しません。
2. R1++：サイズ指定子 (.size) が“.B”のとき1、“.W”のとき2、“.L”のとき4が加算されます。

【機能】

- R1で示される比較先番地からアドレスの加算方向に、R2の内容と一致しないデータが現れるまで、R3で指定される回数を上限として検索を行います。サイズ指定子 (.size) が“.B”または“.W”のときは、メモリのバイトデータまたはワードデータをロングワードデータにゼロ拡張し、R2の内容と比較を行います。
- 命令実行時は、R1で示される比較先番地からデータプリフェッチが行われる場合があります。ただし、R3で指定された範囲を超えるデータプリフェッチは行いません。プリフェッチされるデータサイズについては、各製品のユーザーズマニュアルハードウェア編を参照してください。
- フラグは「*R1 - R2」の演算結果に従って変化します。
- 命令終了時のR1は、一致しなかったデータの次の番地を示します。すべて一致したときは、最後に転送したデータの次の番地を示します。
- 命令終了後のR3は、「初期値 - 比較回数」となります。
- 命令実行中に割り込み要求があった場合は、命令途中で転送を中断して割り込みを受け付けます。割り込みルーチンからの復帰後、中断されていた処理を継続して実行します。本命令を使用する際には、割り込み時、R1、R2、R3とPSWを退避/復帰させてください。

【フラグ変化】

フラグ	変化	条件
C	○	符号なし比較の結果、0以上のとき“1”、それ以外のとき“0”になります。
Z	○	すべて一致したとき“1”、それ以外のとき“0”になります。
S	—	
O	—	

【命令フォーマット】

構文	size	処理サイズ	コードサイズ (バイト)
SWHILE.size	B/W/L	L	2

【記述例】

SWHILE.W

TST

テスト
TeST logical

TST

【構文】

TST src, src2

算術/論理演算命令

【命令コード】

記載ページ：335

【オペレーション】

src2 & src;

【機能】

- src2とsrcの論理積をとった結果に従って、PSWの各フラグが変化します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	演算結果が0のとき“1”、それ以外のとき“0”になります。
S	○	演算結果のMSBが“1”のとき“1”、それ以外のとき“0”になります。
O	—	

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	src2	
TST src, src2	L	#SIMM:8	Rs	4
	L	#SIMM:16	Rs	5
	L	#SIMM:24	Rs	6
	L	#IMM:32	Rs	7
	L	Rs	Rs2	3
	L	[Rs].memex	Rs2	3 (memex == “UB”) 4 (memex != “UB”)
	L	dsp:8[Rs].memex (注)	Rs2	4 (memex == “UB”) 5 (memex != “UB”)
	L	dsp:16[Rs].memex (注)	Rs2	5 (memex == “UB”) 6 (memex != “UB”)

注. 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、サイズ拡張指定子が“.W”または“.UW”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:8には、サイズ拡張指定子が“.W”または“.UW”のとき0～510 (255 × 2) が、“.L”のとき0～1020 (255 × 4) が指定できます。dsp:16には、サイズ拡張指定子が“.W”または“.UW”のとき0～131070 (65535 × 2) が、“.L”のとき0～262140 (65535 × 4) が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【記述例】

```
TST #7, R2
TST R1, R2
TST [R1], R2
TST 1[R1].UB, R2
```

UTOF

符号なし整数→単精度浮動小数点数変換
Integer TO Floating-point

UTOF

【構文】

UTOF src, dest

単精度浮動小数点演算命令

【命令コード】

記載ページ：336

【オペレーション】

dest = (float) (unsigned long) src;

【機能】

- srcに格納された符号なしロングワード（32ビット）整数を単精度浮動小数点数に変換し、その結果をdestに格納します。結果はFPSWのRM[1:0]ビットに従って丸められます。0000000hは丸めモードに関係なく、“+0”として扱われます。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	演算の結果が“+0”のとき“1”、それ以外るとき“0”になります。
S	○	“0”になります。
O	—	
CV	○	“0”になります。
CO	○	“0”になります。
CZ	○	“0”になります。
CU	○	“0”になります。
CX	○	精度異常が発生したとき“1”、それ以外るとき“0”になります。
CE	○	“0”になります。
FV	—	
FO	—	
FZ	—	
FU	—	
FX	○	精度異常が発生したとき“1”、それ以外るときは変化しません。

注. FXフラグは、例外処理許可ビットEXが“1”の場合は変化しません。S、Zフラグは、例外処理が発生した場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	src2	
UTOF src, dest	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == “UB”) 4 (memex != “UB”)
	L	dsp:8[Rs].memex (注)	Rd	4 (memex == “UB”) 5 (memex != “UB”)
	L	dsp:16[Rs].memex (注)	Rd	5 (memex == “UB”) 6 (memex != “UB”)

注. 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値（dsp:8, dsp:16）は、サイズ拡張指定子が“.W”または“.UW”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:8には、サイズ拡張指定子が“.W”または“.UW”のとき0～510（255×2）が、“.L”のとき0～1020（255×4）が指定できます。dsp:16には、サイズ拡張指定子が“.W”または“.UW”のとき0～131070（65535×2）が、“.L”のとき0～262140（65535×4）が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【単精度浮動小数点例外発生要因】

精度異常

【記述例】

UTOF R1, R2

UTOF [R1], R2

UTOF 16[R1].L, R2

WAIT

ウェイト
WAIT

WAIT

【構文】

WAIT

システム操作命令

【命令コード】

記載ページ：337

【オペレーション】**【機能】**

- プログラムの実行を停止します。ノンマスクブル割り込み、割り込み、またはリセットが発生するとプログラムの実行を開始します。
- この命令は特権命令です。ユーザモードで実行すると特権命令例外が発生します。
- PSWのIビットが“1”になります。
- 割り込み発生時に退避させられるPCは、WAIT命令の次のアドレスになります。

注． プログラムの実行を停止した状態での低消費電力状態については、各製品のユーザーズマニュアル ハードウェア編を参照してください。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	コードサイズ (バイト)
WAIT	2

【記述例】

WAIT

XCHG

交換
eXCHanGe

XCHG

【構文】

XCHG src, dest

転送命令

【命令コード】

記載ページ：337

【オペレーション】tmp = src;
src = dest;
dest = tmp;**【機能】**

- 以下のとおり、srcとdestの内容を交換します。

src	dest	機能
レジスタ	レジスタ	レジスタ (src) のデータとレジスタ (dest) のデータを交換します。
メモリ	レジスタ	メモリのデータとレジスタのデータを交換します。サイズ拡張指定子が.Bおよび.UBのときは、レジスタのLSB側のバイトデータとメモリのデータを交換します。サイズ拡張指定子が.Wおよび.UWのときは、レジスタのLSB側のワードデータとメモリのデータを交換します。サイズ拡張指定子が.L以外のときは、指定した拡張方法でメモリのデータをロングワードデータに拡張し、レジスタに転送します。

- 実装により、排他制御に使える場合があります。詳細については、各製品のユーザーズマニュアルハードウェア編を参照してください。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
XCHG src, dest	L	Rs	Rd	3
	L	[Rs].memex	Rd	3 (memex == "UB") 4 (memex != "UB")
	L	dsp:8[Rs].memex ^(注)	Rd	4 (memex == "UB") 5 (memex != "UB")
	L	dsp:16[Rs].memex ^(注)	Rd	5 (memex == "UB") 6 (memex != "UB")

注. 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、サイズ拡張指定子が“.W”または“.UW”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:8には、サイズ拡張指定子が“.W”または“.UW”のとき0～510 (255 × 2) が、“.L”のとき0～1020 (255 × 4) が指定できます。dsp:16には、サイズ拡張指定子が“.W”または“.UW”のとき0～131070 (65535 × 2) が、“.L”のとき0～262140 (65535 × 4) が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【記述例】XCHG R1, R2
XCHG [R1].W, R2

XOR

排他的論理和
eXclusive OR logical

XOR

【構文】

- (1) XOR src, dest
(2) XOR src, src2, dest

算術/論理演算命令

【命令コード】

記載ページ : 338

【オペレーション】

- (1) dest = dest ^ src;
(2) dest = src2 ^ src;

【機能】

- (1) destとsrcの排他的論理和をとり、その結果をdestに格納します。
(2) src2とsrcの排他的論理和をとり、その結果をdestに格納します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	演算後のdestが0のとき“1”、それ以外のとき“0”になります。
S	○	演算後のdestのMSBが“1”のとき“1”、それ以外のとき“0”になります。
O	—	

【命令フォーマット】

構文	処理サイズ	対象			コードサイズ (バイト)
		src	src2	dest	
(1) XOR src, dest	L	#SIMM:8	—	Rd	4
	L	#SIMM:16	—	Rd	5
	L	#SIMM:24	—	Rd	6
	L	#IMM:32	—	Rd	7
	L	Rs	—	Rd	3
	L	[Rs].memex	—	Rd	3 (memex == “UB”) 4 (memex != “UB”)
	L	dsp:8[Rs].memex (注)	—	Rd	4 (memex == “UB”) 5 (memex != “UB”)
	L	dsp:16[Rs].memex (注)	—	Rd	5 (memex == “UB”) 6 (memex != “UB”)
(2) XOR src, src2, dest	L	Rs	Rs2	Rd	3

注. 弊社の「RXファミリアセンブラ」では、ディスプレイメントの値 (dsp:8, dsp:16) は、サイズ拡張指定子が“.W”または“.UW”のときは2の倍数、“.L”のときは4の倍数を指定してください。dsp:8には、サイズ拡張指定子が“.W”または“.UW”のとき0～510 (255 × 2) が、“.L”のとき0～1020 (255 × 4) が指定できます。dsp:16には、サイズ拡張指定子が“.W”または“.UW”のとき0～131070 (65535 × 2) が、“.L”のとき0～262140 (65535 × 4) が指定できます。命令コードには、1/2、1/4した値が埋め込まれます。

【記述例】

```
XOR #8, R1
XOR R1, R2
XOR [R1], R2
XOR 16[R1].L, R2
XOR R1, R2, R3
```

3.5.2 レジスタ括退避機能命令

次ページよりレジスタ括退避機能命令の詳細説明を示します。

RSTR

レジスタの一括復帰
ReSToRe registers

RSTR

【構文】

RSTR src

レジスタ一括退避機能命令

【命令コード】

記載ページ：341

【オペレーション】

```
{ R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, USP, FPSW, ACC0, ACC1 } =
bank(src);
```

【機能】

- レジスタ退避バンクからCPUレジスタ(R1～R15, USP, FPSW, ACC0, ACC1)へ値を一括復帰させます。
- 復帰させたいバンク番号は、srcによって指定します。
- この命令は特権命令です。ユーザモードで実行すると特権命令例外が発生します。
- レジスタ退避バンクの有無、ならびに使用できるバンク番号の範囲(搭載容量)については各製品のユーザーズマニュアルハードウェア編を参照してください。
- 存在しないバンク番号を指定した場合の動作は不定です。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象	コードサイズ (バイト)
		src	
RSTR src	—	#UIMM:8	4
	—	Rs	4

【記述例】

RSTR #5

RSTR R1

SAVE

レジスタの一括退避 SAVE registers

SAVE

【構文】

SAVE src

レジスタ一括退避機能命令

【命令コード】

記載ページ：341

【オペレーション】

bank(src) = { R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, USP,
FPSW, ACC0, ACC1};

【機能】

- レジスタ退避バンクにCPUレジスタ (R1～R15, USP, FPSW, ACC0, ACC1)の値を一括退避させます。
- 退避先のバンク番号は、srcによって指定します。
- この命令は特権命令です。ユーザモードで実行すると特権命令例外が発生します。
- レジスタ退避バンクの有無、ならびに使用できるバンク番号の範囲(搭載容量)については各製品のユーザーズマニュアルハードウェア編を参照してください。
- 存在しないバンク番号を指定した場合の動作は不定です。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象	コードサイズ (バイト)
		src	
SAVE src	—	#UIMM:8	4
	—	Rs	4

【記述例】

SAVE #5

SAVE R1

3.5.3 倍精度浮動小数点処理命令

次ページより倍精度浮動小数点処理命令の各命令の詳細説明を示します。

DABS

倍精度浮動小数点絶対値
Double precision floating-point ABSolute

DABS

【構文】

```
DABS src, dest
```

倍精度浮動小数点演算命令

【命令コード】

記載ページ：343

【オペレーション】

```
if ( src < 0 )
    dest = -src;
else
    dest = src;
```

【機能】

src に格納された倍精度浮動小数点数の絶対値を算出し、その結果を dest に格納します。

- 非正規化数は、DPSWのDDNビットによらず同じ処理をします。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
DABS src, dest	D	DRs	DRd	4

【倍精度浮動小数点例外発生要因】

なし

【記述例】

```
DABS DR0, DR1
```

【動作補足説明】

- src の値と演算結果の対応を以下に示します。

	src								
	+正規化数	-正規化数	+0	-0	+∞	-∞	非正規化数	QNaN	SNaN
結果	+正規化数	+正規化数	+0	+0	+∞	+∞	正の 非正規化数	正の QNaN	正の SNaN

DADD

倍精度浮動小数点加算
Double precision floating-point ADD

DADD

【構文】

```
DADD src, src2, dest
```

倍精度浮動小数点演算命令

【命令コード】

記載ページ：343

【オペレーション】

```
dest = src2 + src;
```

【機能】

src2 に格納された倍精度浮動小数点数と、src に格納された倍精度浮動小数点数を加算し、その結果を dest に格納します。

- 結果はDPSWのDRM[1:0]ビットに従って丸められます。
- 非正規化数の扱いは、DPSWのDDNビットによって変化します。
- 反対の符号を持つ (src, src2) の和が正確に0であるときは、 $-\infty$ 方向への丸めモードの場合を除いて、結果は+0になります。 $-\infty$ 方向への丸めモードの場合は、結果は-0になります。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	—	
O	—	
DCV	○	無効演算が発生したとき“1”、それ以外るとき“0”になります。
DCO	○	オーバフローが発生したとき“1”、それ以外るとき“0”になります。
DCZ	○	“0”になります。
DCU	○	アンダフローが発生したとき“1”、それ以外るとき“0”になります。
DCX	○	精度異常が発生したとき“1”、それ以外るとき“0”になります。
DCE	○	非実装処理が発生したとき“1”、それ以外るとき“0”になります。
DFV	○	無効演算が発生したとき“1”、それ以外るときは変化しません。
DFO	○	オーバフローが発生したとき“1”、それ以外るときは変化しません。
DFZ	—	
DFU	○	アンダフローが発生したとき“1”、それ以外るときは変化しません。
DFX	○	精度異常が発生したとき“1”、それ以外るときは変化しません。

注. DFX、DFU、DFZ、DFO、DFVフラグは、DPSWレジスタの例外処理許可ビットDEX、DEU、DEZ、DEO、DEVが“1”の場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象			コードサイズ (バイト)
		src	src2	dest	
DADD src, src2, dest	D	DRs	DRs2	DRd	4

【倍精度浮動小数点例外発生要因】

- 非実装処理
- 無効演算
- オーバフロー
- アンダフロー
- 精度異常

【記述例】

DADD DR0, DR1, DR2

【動作補足説明】

- DPSWのDDNビットが“0”の場合と“1”の場合について、src、src2の値と演算結果の対応を以下に示します。

DDN = 0のとき

		src							
		正規化数	+0	-0	+∞	-∞	非正規化数	QNaN	SNaN
src2	正規化数	加算							
	+0		+0	(注1)		-∞			
	-0		(注1)	-0					
	+∞				+∞	無効演算			
	-∞				-∞	無効演算	-∞		
	非正規化数	非実装処理							
	QNaN							QNaN	
	SNaN								無効演算

注1. 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になります。

DDN = 1のとき

		src						QNaN	SNaN
		正規化数	+0、 +非正規化数	-0、 -非正規化数	+∞	-∞			
src2	正規化数	加算		正規化数					
	+0、+非正規化数	正規化数	+0	(注1)		-∞			
	-0、-非正規化数		(注1)	-0					
	+∞				+∞	無効演算			
	-∞				-∞	無効演算	-∞		
	QNaN							QNaN	
SNaN								無効演算	

注1. 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になります。

DCMPcm

倍精度浮動小数点比較
Double precision floating-point CoMPare

DCMPcm

【構文】

DCMPcm src, src2

倍精度浮動小数点演算命令

【命令コード】

記載ページ：344

【オペレーション】

```

If ( cm == UN )
    RES = isNaN(src) || isNaN(src2);
else if ( cm == EQ )
    RES = ( src2 == src );
else if ( cm == LT )
    RES = ( src2 < src );
else if ( cm == LE )
    RES = ( src2 <= src );

```

【機能】

src2 に格納された倍精度浮動小数点数と、src に格納された倍精度浮動小数点数を比較条件 cm で比較し、その結果を DCMR レジスタの RES ビットに格納します。比較条件 cm が成立する場合に RES ビットは“1”となり、それ以外の場合は“0”となります。

【比較条件】

cm	定義	説明
UN	Unordered	順序化不能を検出する条件です。
EQ	src2 == src	src2, src が等しいことを検出する条件です。
LT	src2 < src	src2 が src より小さいことを検出する条件です。
LE	src2 ≤ src	src2 が src より小さいか等しいことを検出する条件です。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	—	
O	—	
DCV	○	無効演算が発生したとき“1”、それ以外るとき“0”になります。
DCO	○	“0”になります。
DCZ	○	“0”になります。
DCU	○	“0”になります。
DCX	○	“0”になります。
DCE	○	非実装処理が発生したとき“1”、それ以外るとき“0”になります。
DFV	○	無効演算が発生したとき“1”、それ以外るときは変化しません。
DFO	—	
DFZ	—	
DFU	—	
DFX	—	

注. DFV フラグは、DPSW レジスタの例外処理許可ビット DEV が“1”の場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	src2	
DCMPcm src, src2	D	DRs	DRs2	4

【倍精度浮動小数点例外発生要因】

非実装処理
無効演算

【記述例】

DCMPEQ DR0, DR1

【動作補足説明】

- DPSWのDDNビットが“0”の場合と“1”の場合について、src、src2の値と比較結果の対応を以下に示します。
(> : src2 > src、< : src2 < src、= : src2 == src)

DDN = 0のとき

		src								
		正規化数	+0	-0	+∞	-∞	非正規化数	QNaN	SNaN	
src2	正規化数	比較			<	>	非実装処理	順序化不能	無効演算 (順序化不能)	
	+0	=								
	-0									
	+∞	>		=						
	-∞	<		=						
	非正規化数									
	QNaN									
	SNaN									

DDN = 1のとき

		src							
		正規化数	+0、 +非正規化数	-0、 -非正規化数	+∞	-∞	QNaN	SNaN	
src2	正規化数	比較			<	>	順序化不能	無効演算 (順序化不能)	
	+0、+非正規化数	=							
	-0、-非正規化数								
	+∞	>		=					
	-∞	<		=					
	QNaN								
	SNaN								

DDIV

倍精度浮動小数点除算 Double precision floating-point DIVide

DDIV

【構文】

DDIV src, src2, dest

倍精度浮動小数点演算命令

【命令コード】

記載ページ：344

【オペレーション】

dest = src2 / src;

【機能】

src2 に格納された倍精度浮動小数点数から、src に格納された倍精度浮動小数点数を除算し、その結果を dest に格納します。

- 結果はDPSWのDRM[1:0]ビットに従って丸められます。
- 非正規化数の扱いは、DPSWのDDNビットによって変化します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	—	
O	—	
DCV	○	無効演算が発生したとき“1”、それ以外るとき“0”になります。
DCO	○	オーバフローが発生したとき“1”、それ以外るとき“0”になります。
DCZ	○	ゼロ除算が発生したとき“1”、それ以外るとき“0”になります。
DCU	○	アンダフローが発生したとき“1”、それ以外るとき“0”になります。
DCX	○	精度異常が発生したとき“1”、それ以外るとき“0”になります。
DCE	○	非実装処理が発生したとき“1”、それ以外るとき“0”になります。
DFV	○	無効演算が発生したとき“1”、それ以外るときは変化しません。
DFO	○	オーバフローが発生したとき“1”、それ以外るときは変化しません。
DFZ	○	ゼロ除算が発生したとき“1”、それ以外るときは変化しません。
DFU	○	アンダフローが発生したとき“1”、それ以外るときは変化しません。
DFX	○	精度異常が発生したとき“1”、それ以外るときは変化しません。

注. DFX、DFU、DFZ、DFO、DFVフラグは、DPSWレジスタの例外処理許可ビットDEX、DEU、DEZ、DEO、DEVが“1”の場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象			コードサイズ (バイト)
		src	src2	dest	
DDIV src, src2, dest	D	DRs	DRs2	DRd	4

【倍精度浮動小数点例外発生要因】

非実装処理
無効演算
オーバフロー
アンダフロー
精度異常
ゼロ除算

【記述例】

DDIV DR0, DR1, DR2

【動作補足説明】

- DPSWのDDNビットが“0”の場合と“1”の場合について、src、src2の値と演算結果の対応を以下に示します。

DDN = 0のとき

		src									
		正規化数	+0	-0	$+\infty$	$-\infty$	非正規化数	QNaN	SNaN		
src2	正規化数	除算	ゼロ除算		0		非実装処理	QNaN	無効演算		
	+0	0	無効演算		+0	-0					
	-0				-0	+0					
	$+\infty$	∞	$+\infty$	$-\infty$	無効演算						
	$-\infty$		$-\infty$	$+\infty$							
	非正規化数										
	QNaN										
SNaN											

DDN = 1のとき

		src							
		正規化数	+0、 +非正規化数	-0、 -非正規化数	$+\infty$	$-\infty$	QNaN	SNaN	
src2	正規化数	除算	ゼロ除算		0		QNaN	無効演算	
	+0、+非正規化数	0	無効演算		+0	-0			
	-0、-非正規化数				-0	+0			
	$+\infty$	∞	$+\infty$	$-\infty$	無効演算				
	$-\infty$		$-\infty$	$+\infty$					
	QNaN								
	SNaN								

DMOV

倍精度浮動小数点データ転送 Double precision floating-point data MOVE

DMOV

【構文】

DMOV.size src, dest

倍精度浮動小数点転送命令

【命令コード】

記載ページ : 345

【オペレーション】

dest = src;

【機能】

- 以下のとおり、srcをdestに転送します。

src	dest	機能
汎用レジスタ	倍精度浮動小数点データレジスタ	レジスタ (src) のデータを倍精度浮動小数点データレジスタ (dest) に転送します。サイズ指定子は.Lと.Dが選択できます。サイズ指定子.Dは倍精度浮動小数点データレジスタ (dest) の上位32ビットを指定する場合のみ選択できます。倍精度浮動小数点データレジスタ (dest) の上位32ビットを指定する場合、サイズ指定子.Dの場合には下位32ビットに0を転送し、サイズ指定子.Lの場合には下位32ビットはそのまま保持されます。倍精度浮動小数点データレジスタ (dest) の下位32ビットを指定する場合には、上位32ビットはそのまま保持されます。
倍精度浮動小数点データレジスタ	汎用レジスタ	倍精度浮動小数点データレジスタ (src) のデータをレジスタ (dest) に転送します。サイズ指定子は.Lのみとなります。倍精度浮動小数点データレジスタ (src) は、上位32ビット/下位32ビットを指定します。
倍精度浮動小数点データレジスタ	倍精度浮動小数点データレジスタ	倍精度浮動小数点データレジスタ (src) のデータを倍精度浮動小数点データレジスタ (dest) に転送します。サイズ指定子は.Dのみとなります。
倍精度浮動小数点データレジスタ	メモリ	倍精度浮動小数点データレジスタのデータをメモリに転送します。サイズ指定子は.Dのみとなります。
メモリ	倍精度浮動小数点データレジスタ	メモリのデータを倍精度浮動小数点データレジスタに転送します。サイズ指定子は.Dのみとなります。
即値	倍精度浮動小数点データレジスタ	命令で指定した即値のデータを倍精度浮動小数点データレジスタ (dest) に転送します。サイズ指定子は.Lと.Dが選択できます。サイズ指定子.Dは倍精度浮動小数点データレジスタ (dest) の上位32ビットを指定する場合のみ選択できます。倍精度浮動小数点データレジスタ (dest) の上位32ビットを指定する場合、サイズ指定子.Dの場合には下位32ビットに0を転送し、サイズ指定子.Lの場合には下位32ビットはそのまま保持されます。倍精度浮動小数点データレジスタ (dest) の下位32ビットを指定する場合には、上位32ビットはそのまま保持されます。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	size	処理サイズ	対象		コードサイズ (バイト)
			src	dest	
DMOV.size src, dest	レジスタ間転送				
	D	D	Rs	DRHd	4
	L	L	Rs	DRHd	4
	L	L	Rs	DRLd	4
	L	L	DRHs	Rd	4
	L	L	DRLs	Rd	4
	D	D	DRs	DRd	4
	ストア				
	D	D	DRs	[Rd]	4
	D	D	DRs	dsp:8[Rd] (注1)	5
	D	D	DRs	dsp:16[Rd] (注1)	6
	ロード				
	D	D	[Rs]	DRd	4
	D	D	dsp:8[Rs] (注1)	DRd	5
	D	D	dsp:16[Rs] (注1)	DRd	6
	即値				
	D	D	#IMM:32	DRHd	7
	L	L	#IMM:32	DRHd	7
	L	L	#IMM:32	DRLd	7

注1. 弊社の「RXファミリアセンブラ」では、ディスプレイacementsの値 (dsp:8, dsp:16) は、サイズ指定子が“.D”のときは4の倍数を指定してください。dsp:8には、サイズ指定子が“.D”のとき0～1020 (255×4) が指定できます。dsp:16には、サイズ指定子が“.D”のとき0～262140 (65535×4) が指定できます。命令コードには、1/4した値が埋め込まれます。

【倍精度浮動小数点例外発生要因】

なし

【記述例】

DMOV.D R1, DRH2
 DMOV.L R1, DRH2
 DMOV.L R1, DRL2
 DMOV.L DRH2, R1
 DMOV.L DRL2, R1
 DMOV.D DR0, DR1
 DMOV.D DR1, [R1]
 DMOV.D [R1], DR1
 DMOV.D #3FF00000h, DRH2
 DMOV.L #4C000000h, DRH2
 DMOV.L #00000000h, DRL2

DMUL倍精度浮動小数点乗算
Double precision floating-point MULTiPLY**DMUL**

【構文】

DMUL src, src2, dest

倍精度浮動小数点演算命令

【命令コード】

記載ページ：348

【オペレーション】

dest = src2 * src;

【機能】

src2 に格納された倍精度浮動小数点数と、src に格納された倍精度浮動小数点数を乗算し、その結果を dest に格納します。

- 結果はDPSWのDRM[1:0]ビットに従って丸められます。
- 非正規化数の扱いは、DPSWのDDNビットによって変化します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	—	
O	—	
DCV	○	無効演算が発生したとき“1”、それ以外るとき“0”になります。
DCO	○	オーバフローが発生したとき“1”、それ以外るとき“0”になります。
DCZ	○	“0”になります。
DCU	○	アンダフローが発生したとき“1”、それ以外るとき“0”になります。
DCX	○	精度異常が発生したとき“1”、それ以外るとき“0”になります。
DCE	○	非実装処理が発生したとき“1”、それ以外るとき“0”になります。
DFV	○	無効演算が発生したとき“1”、それ以外るときは変化しません。
DFO	○	オーバフローが発生したとき“1”、それ以外るときは変化しません。
DFZ	—	
DFU	○	アンダフローが発生したとき“1”、それ以外るときは変化しません。
DFX	○	精度異常が発生したとき“1”、それ以外るときは変化しません。

注. DFX、DFU、DFZ、DFO、DFVフラグは、DPSWレジスタの例外処理許可ビットDEX、DEU、DEZ、DEO、DEVが“1”の場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象			コードサイズ (バイト)
		src	src2	dest	
DMUL src, src2, dest	D	DRs	DRs2	DRd	4

【倍精度浮動小数点例外発生要因】

非実装処理
無効演算
オーバフロー
アンダフロー
精度異常

【記述例】

DMUL DR0, DR1, DR2

【動作補足説明】

- DPSWのDDNビットが“0”の場合と“1”の場合について、src、src2の値と演算結果の対応を以下に示します。

DDN = 0のとき

		src									
		正規化数	+0	-0	+∞	-∞	非正規化数	QNaN	SNaN		
src2	正規化数	乗算			∞		非実装処理	QNaN	無効演算		
	+0		+0	-0	無効演算						
	-0		-0	+0	無効演算						
	+∞		無効演算		+∞	-∞					
	-∞	∞	無効演算		-∞	+∞					
	非正規化数	非実装処理									
	QNaN	QNaN									
	SNaN	無効演算									

DDN = 1のとき

		src							
		正規化数	+0、 +非正規化数	-0、 -非正規化数	+∞	-∞	QNaN	SNaN	
src2	正規化数	乗算			∞		QNaN	無効演算	
	+0、+非正規化数		+0	-0	無効演算				
	-0、-非正規化数		-0	+0	無効演算				
	+∞		無効演算		+∞	-∞			
	-∞	∞	無効演算		-∞	+∞			
	QNaN	QNaN							
	SNaN	無効演算							

DNEG

倍精度浮動小数点符号反転
Double precision floating-point flag NEGate

DNEG

【構文】

DNEG src, dest

倍精度浮動小数点演算命令

【命令コード】

記載ページ：348

【オペレーション】

dest = -src;

【機能】

src に格納された倍精度浮動小数点数の符号を反転し、その結果を dest に格納します。

- 非正規化数は、DPSWのDDNビットによらず同じ処理をします。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
DNEG src, dest	D	DRs	DRd	4

【倍精度浮動小数点例外発生要因】

なし

【記述例】

DNEG DR0, DR1

【動作補足説明】

- src の値と演算結果の対応を以下に示します。

	src								
	+正規化数	-正規化数	+0	-0	+∞	-∞	非正規化数	QNaN	SNaN
結果	-正規化数	+正規化数	-0	+0	-∞	+∞	符号反転した 非正規化数	符号反転した QNaN	符号反転した SNaN

DPOPM

複数の倍精度浮動小数点レジスタの復帰
Double precision POP Multiple register

DPOPM

倍精度浮動小数点転送命令

【命令コード】

記載ページ：349

【構文】

DPOPM.size dest-dest2

【オペレーション】

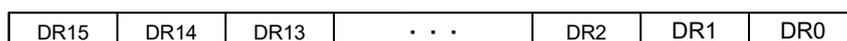
```
signed char i;
for ( i = register_num(dest); i <= register_num(dest2); i++ ) {
    tmp = *SP;
    SP = SP + (( size == D ) ? 8 : 4);
    If ( size == D ) {
        DR(i) = tmp;
    } else {
        DCR(i) = tmp;
    }
}
```

【機能】

dest と dest2 で範囲指定した倍精度浮動小数点レジスタを一括してスタックから復帰します。

- 範囲は、dest と dest2 で指定します。ただし、(dest のレジスタ番号 ≤ dest2 のレジスタ番号) となっている必要があります。
- 使用されるスタックポインタは、PSW の U ビットで示すスタックポインタになります。
- size = D のときは転送対象が倍精度浮動小数点データレジスタになります。
- size = L のときは転送対象が倍精度浮動小数点制御レジスタになります。
- スタックから復帰する順序は以下のとおりです。

Size = D



DR0側から復帰します

Size = L



DCR0側から復帰します

【フラグ変化】

- フラグ変化はありません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		dest	dest2	
DPOPM.size dest-dest2	D	DRd	DRd2	3
	L	DCRd	DCRd2	3

【倍精度浮動小数点例外発生要因】

なし

【記述例】

DPOPM.D DR0-DR15
DPOPM.L DPSW-DCMR
DPOPM.D DR1-DR1

DPUSHM

複数の倍精度浮動小数点レジスタの退避
Double precision PUSH Multiple register

DPUSHM

倍精度浮動小数点転送命令

【命令コード】

記載ページ：350

【構文】

DPUSHM.size src-src2

【オペレーション】

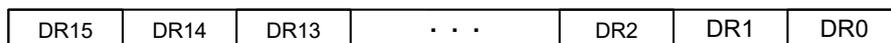
```
signed char i;
for ( i = register_num(src2); i >= register_num(src); i-- ) {
    tmp = ( size == D ) ? DR(i) : DCR(i);
    SP = SP - (( size == D ) ? 8 : 4);
    *SP = tmp;
}
```

【機能】

src と src2 で範囲指定した倍精度浮動小数点レジスタを一括してスタックに退避します。

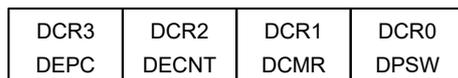
- 範囲は、src と src2 で指定します。ただし、(src のレジスタ番号 ≤ src2 のレジスタ番号) となっている必要があります。
- 使用されるスタックポインタは、PSW の U ビットで示すスタックポインタになります。
- size = D のときは転送対象が倍精度浮動小数点データレジスタになります。
- size = L のときは転送対象が倍精度浮動小数点制御レジスタになります。
- スタックに退避する順序は以下のとおりです。

Size = D



DR15側から退避します

Size = L



DCR3側から退避します

【フラグ変化】

- フラグ変化はありません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	src2	
DPUSHM.size src-src2	D	DRs	DRs2	3
	L	DCRs	DCRs2	3

【倍精度浮動小数点例外発生要因】

なし

【記述例】

DPUSHM.D DR0-DR15

DPUSHM.L DPSW-DCMR

DPUSHM.D DR1-DR1

DROUND

倍精度浮動小数点数→符号付き整数変換
Double precision ROUND to signed integer

DROUND

倍精度浮動小数点演算命令

【命令コード】

記載ページ：351

【構文】

DROUND src, dest

【オペレーション】

dest = (signed long) src;

【機能】

src に格納された倍精度浮動小数点数を符号付きロングワード（32 ビット）整数に変換し、その結果を dest の下位 32 ビットに格納します。

- 結果はDPSWのDRM[1:0]ビットに従って丸められます。
- destの[63:32]ビットは不定になります。

DRM[1:0]ビットの値	丸めモード
00b	最近値への丸め
01b	0方向への丸め
10b	+∞方向への丸め
11b	-∞方向への丸め

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	—	
O	—	
DCV	○	無効演算が発生したとき“1”、それ以外るとき“0”になります。
DCO	○	“0”になります。
DCZ	○	“0”になります。
DCU	○	“0”になります。
DCX	○	精度異常が発生したとき“1”、それ以外るとき“0”になります。
DCE	○	非実装処理が発生したとき“1”、それ以外るとき“0”になります。
DFV	○	無効演算が発生したとき“1”、それ以外るときは変化しません。
DFO	—	
DFZ	—	
DFU	—	
DFX	○	精度異常が発生したとき“1”、それ以外るときは変化しません。

注. DFX、DFVフラグは、DPSWレジスタの例外処理許可ビットDEX、DEVが“1”の場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
DROUND src, dest	D	DRs	DRd	4

【倍精度浮動小数点例外発生要因】

非実装処理
無効演算
精度異常

【記述例】

DROUND DR0, DR1

【動作補足説明】

- DPSWのDDNビットが“0”の場合と“1”の場合について、srcの値と変換結果の対応を以下に示します。

DDN = 0のとき

srcの値 (指数部はゲタなしの値)		dest (注1)	例外
src ≥ 0	+∞	7FFFFFFFh	無効演算
	31 ≤ 指数部 ≤ 1023		
	-1022 ≤ 指数部 ≤ 30	00000000h ~ 7FFFFFFFh	なし (注2、注3)
	+非正規化数	変化なし	非実装処理
src < 0	+0	00000000h	なし
	-0		
	-非正規化数	変化なし	非実装処理
	-1022 ≤ 指数部 ≤ 30	00000000h ~ 80000000h	なし (注2)
NaN	31 ≤ 指数部 ≤ 1023	80000000h (注4)	無効演算 (注2、注5)
	-∞		
	QNaN	符号ビット = 0 : 7FFFFFFFh	無効演算
SNaN	符号ビット = 1 : 80000000h		

- 注1. 倍精度浮動小数点例外が発生し、対応するDEj (j = X, U, Z, O, V) が“1”の場合には変化しません。
 注2. 無効演算が発生しないとき、丸め発生時には精度異常が発生します。
 注3. 丸め後の値が7FFFFFFFhを超える場合は、無効演算が発生します。
 注4. 丸め後の値によらずdestは80000000hとなります。
 注5. 丸め後の値が80000000hとなる場合は、無効演算は発生しません。

DDN = 1のとき

srcの値 (指数部はゲタなしの値)		dest (注1)	例外
src ≥ 0	+∞	7FFFFFFFh	無効演算
	31 ≤ 指数部 ≤ 1023		
	-1022 ≤ 指数部 ≤ 30	00000000h ~ 7FFFFFFFh	なし (注2、注3)
	+非正規化数	00000000h	なし
+0			
src < 0	-0		
	-非正規化数		
	-1022 ≤ 指数部 ≤ 30	00000000h ~ 80000000h	なし (注2)
	31 ≤ 指数部 ≤ 1023	80000000h (注4)	無効演算 (注2、注5)
NaN	-∞		
	QNaN	符号ビット = 0 : 7FFFFFFFh	無効演算
SNaN	符号ビット = 1 : 80000000h		

- 注1. 倍精度浮動小数点例外が発生し、対応するDEj (j = X, U, Z, O, V) が“1”の場合には変化しません。
 注2. 無効演算が発生しないとき、丸め発生時には精度異常が発生します。
 注3. 丸め後の値が7FFFFFFFhを超える場合は、無効演算が発生します。
 注4. 丸め後の値によらずdestは80000000hとなります。
 注5. 丸め後の値が80000000hとなる場合は、無効演算は発生しません。

DSQRT

倍精度浮動小数点平方根
Double precision floating-point Square Root

DSQRT

【構文】

DSQRT src, dest

倍精度浮動小数点演算命令

【命令コード】

記載ページ：351

【オペレーション】

dest = sqrt (src);

【機能】

src に格納された倍精度浮動小数点数の平方根を求め、その結果を dest に格納します。

- 結果はDPSWのDRM[1:0]ビットに従って丸められます。
- 非正規化数の扱いは、DPSWのDDNビットによって変化します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	—	
O	—	
DCV	○	無効演算が発生したとき“1”、それ以外るとき“0”になります。
DCO	○	“0”になります。
DCZ	○	“0”になります。
DCU	○	“0”になります。
DCX	○	精度異常が発生したとき“1”、それ以外るとき“0”になります。
DCE	○	非実装処理が発生したとき“1”、それ以外るとき“0”になります。
DFV	○	無効演算が発生したとき“1”、それ以外るときは変化しません。
DFO	—	
DFZ	—	
DFU	—	
DFX	○	精度異常が発生したとき“1”、それ以外るときは変化しません。

注. DFX、DFVフラグは、DPSWレジスタの例外処理許可ビットDEX、DEVが“1”の場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
DSQRT src, dest	D	DRs	DRd	4

【倍精度浮動小数点例外発生要因】

非実装処理
無効演算
精度異常

【記述例】

DSQRT DR0, DR1

【動作補足説明】

- DPSWのDDNビットが“0”の場合と“1”の場合について、srcの値と演算結果の対応を以下に示します。

DDN = 0のとき

	src								
	+正規化数	-正規化数	+0	-0	+∞	-∞	非正規化数	QNaN	SNaN
結果	平方根	無効演算	+0	-0	+∞	無効演算	非実装処理	QNaN	無効演算

DDN = 1のとき

	src									
	+正規化数	-正規化数	+0	-0	+∞	-∞	+非正規化数	-非正規化数	QNaN	SNaN
結果	平方根	無効演算	+0	-0	+∞	無効演算	+0	-0	QNaN	無効演算

【無効演算発生時のQNaN生成規則】

ソースオペランド	演算結果
SNaN	QNaN化されたSNaNソースオペランド
上記以外	7FFFFFFFFFFFFFFFh

DSUB

倍精度浮動小数点減算
Double precision floating-point SUBtract

DSUB

【構文】

```
DSUB src, src2, dest
```

倍精度浮動小数点演算命令

【命令コード】

記載ページ：351

【オペレーション】

```
dest = src2 - src;
```

【機能】

src2 に格納された倍精度浮動小数点数から、src に格納された倍精度浮動小数点数を減算し、その結果を dest に格納します。

- 結果はDPSWのDRM[1:0]ビットに従って丸められます。
- 非正規化数の扱いは、DPSWのDDNビットによって変化します。
- 反対の符号を持つ (src、src2) の差が正確に0であるときは、 $-\infty$ 方向への丸めモードの場合を除いて、結果は+0になります。 $-\infty$ 方向への丸めモードの場合は、結果は-0になります。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	—	
O	—	
DCV	○	無効演算が発生したとき“1”、それ以外るとき“0”になります。
DCO	○	オーバフローが発生したとき“1”、それ以外るとき“0”になります。
DCZ	○	“0”になります。
DCU	○	アンダフローが発生したとき“1”、それ以外るとき“0”になります。
DCX	○	精度異常が発生したとき“1”、それ以外るとき“0”になります。
DCE	○	非実装処理が発生したとき“1”、それ以外るとき“0”になります。
DFV	○	無効演算が発生したとき“1”、それ以外るときは変化しません。
DFO	○	オーバフローが発生したとき“1”、それ以外るときは変化しません。
DFZ	—	
DFU	○	アンダフローが発生したとき“1”、それ以外るときは変化しません。
DFX	○	精度異常が発生したとき“1”、それ以外るときは変化しません。

注. DFX、DFU、DFZ、DFO、DFVフラグは、DPSWレジスタの例外処理許可ビットDEX、DEU、DEZ、DEO、DEVが“1”の場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象			コードサイズ (バイト)
		src	src2	dest	
DSUB src, src2, dest	D	DRs	DRs2	DRd	4

【倍精度浮動小数点例外発生要因】

非実装処理
無効演算
オーバフロー
アンダフロー

精度異常

【記述例】

DSUB DR0, DR1, DR2

【動作補足説明】

- DPSWのDDNビットが“0”の場合と“1”の場合について、src、src2の値と演算結果の対応を以下に示します。

DDN = 0のとき

		SRC									
		正規化数	+0	-0	+∞	-∞	非正規化数	QNaN	SNaN		
src2	正規化数	減算		-∞	+∞	非実装処理	QNaN	無効演算			
	+0	(注1)	+0								
	-0	-0	(注1)								
	+∞	+∞		無効演算							
	-∞	-∞		無効演算							
	非正規化数	非実装処理									
	QNaN	QNaN									
	SNaN	無効演算									

注1. 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になります。

DDN = 1のとき

		SRC								
		正規化数	+0、 +非正規化数	-0、 -非正規化数	+∞	-∞	QNaN	SNaN		
src2	正規化数	減算		-∞	+∞	非実装処理	QNaN	無効演算		
	+0、+非正規化数	(注1)	+0							
	-0、-非正規化数	-0	(注1)							
	+∞	+∞		無効演算						
	-∞	-∞		無効演算						
	QNaN	非実装処理								
	SNaN	無効演算								

注1. 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になります。

DTOF倍精度浮動小数点数→単精度浮動小数点数変換
Double TO Float**DTOF****【構文】**

DTOF src, dest

倍精度浮動小数点演算命令

【命令コード】

記載ページ：352

【オペレーション】

dest = (float) src;

【機能】

srcに格納された倍精度浮動小数点数を単精度浮動小数点数に変換し、その結果をdestの下位32ビットに格納します。

- 結果はDPSWのDRM[1:0]ビットに従って丸められます。
- destの[63:32]ビットは不定になります。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	—	
O	—	
DCV	○	無効演算が発生したとき“1”、それ以外るとき“0”になります。
DCO	○	オーバフローが発生したとき“1”、それ以外るとき“0”になります。
DCZ	○	“0”になります。
DCU	○	アンダフローが発生したとき“1”、それ以外るとき“0”になります。
DCX	○	精度異常が発生したとき“1”、それ以外るとき“0”になります。
DCE	○	非実装処理が発生したとき“1”、それ以外るとき“0”になります。
DFV	○	無効演算が発生したとき“1”、それ以外るときは変化しません。
DFO	—	
DFZ	—	
DFU	—	
DFX	○	精度異常が発生したとき“1”、それ以外るときは変化しません。

注： DFX、DFVフラグは、DPSWレジスタの例外処理許可ビットDEX、DEVが“1”の場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
DTOF src, dest	D	DRs	DRd	4

【倍精度浮動小数点例外発生要因】

非実装処理

無効演算

オーバフロー

アンダフロー

精度異常

【記述例】

DTON DR0, DR1

【動作補足説明】

- DPSWのDDNビットが“0”の場合と“1”の場合について、srcの値と変換結果の対応を以下に示します。

DDN = 0のとき

srcの値 (指数部はゲタなしの値)		dest (注1)	例外
src ≥ 0	+∞	+∞	なし
	128 ≤ 指数部 ≤ 1023		オーバフロー
	-126 ≤ 指数部 ≤ 127	変換結果	なし (注2、注3)
	-1022 ≤ 指数部 ≤ -127	変化なし	非実装処理 アンダフロー
	+非正規化数	変化なし	非実装処理
	+0	+0	なし
src < 0	-0	-0	
	-非正規化数	変化なし	非実装処理
	-1022 ≤ 指数部 ≤ -127	変化なし	非実装処理 アンダフロー
	-126 ≤ 指数部 ≤ 127	変換結果	なし (注2、注3)
	128 ≤ 指数部 ≤ 1023	-∞	オーバフロー
	-∞		なし
NaN	QNaN	QNaN (注4)	なし
	SNaN	SNaN (注5)	無効演算

注1. 倍精度浮動小数点例外が発生し、対応するDEj (j = X, U, Z, O, V) が“1”の場合には変化しません。

注2. 丸め発生時は、精度異常が発生します。

注3. 丸め後の値の指数部が128となる場合には、オーバフローとなります。

注4. destは入力のQNaNと同符号で、入力値の[51:29]ビットを仮数部にもつQNaNとなります。

注5. destは入力のSNaNと同符号で、入力値をQNaN化した値の[51:29]ビットを仮数部にもつQNaNとなります。

DDN = 1のとき

srcの値 (指数部はゲタなしの値)		dest (注1)	例外
src ≥ 0	+∞	+∞	なし
	128 ≤ 指数部 ≤ 1023		オーバフロー
	-126 ≤ 指数部 ≤ 127	変換結果	なし (注2、注3)
	-1022 ≤ 指数部 ≤ -127	+0	アンダフロー
	+非正規化数	+0	なし
	+0	+0	なし
src < 0	-0	-0	
	-非正規化数	-0	なし
	-1022 ≤ 指数部 ≤ -127	-0	アンダフロー
	-126 ≤ 指数部 ≤ 127	変換結果	なし (注2、注3)
	128 ≤ 指数部 ≤ 1023	-∞	オーバフロー
	-∞		なし
NaN	QNaN	QNaN (注4)	なし
	SNaN	SNaN (注5)	無効演算

注1. 倍精度浮動小数点例外が発生し、対応するDEj (j = X, U, Z, O, V) が“1”の場合には変化しません。

注2. 丸め発生時は、精度異常が発生します。

注3. 丸め後の値の指数部が128となる場合には、オーバフローとなります。

注4. destは入力のQNaNと同符号で、入力値の[51:29]ビットを仮数部にもつQNaNとなります。

注5. destは入力のSNaNと同符号で、入力値をQNaN化した値の[51:29]ビットを仮数部にもつQNaNとなります。

DTOI倍精度浮動小数点数→符号付き整数変換
Double TO signed Integer**DTOI****【構文】**

DTOI src, dest

倍精度浮動小数点演算命令

【命令コード】

記載ページ：352

【オペレーション】

dest = (long) src;

【機能】

src に格納された倍精度浮動小数点数を符号付きロングワード（32 ビット）整数に変換し、その結果を dest の下位 32 ビットに格納します。

- 結果はDPSWのDRM[1:0]ビットに関係なく0方向に丸められます。
- destの[63:32]ビットは不定になります。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	—	
O	—	
DCV	○	無効演算が発生したとき“1”、それ以外るとき“0”になります。
DCO	○	“0”になります。
DCZ	○	“0”になります。
DCU	○	“0”になります。
DCX	○	精度異常が発生したとき“1”、それ以外るとき“0”になります。
DCE	○	非実装処理が発生したとき“1”、それ以外るとき“0”になります。
DFV	○	無効演算が発生したとき“1”、それ以外るときは変化しません。
DFO	—	
DFZ	—	
DFU	—	
DFX	○	精度異常が発生したとき“1”、それ以外るときは変化しません。

注. DFX、DFVフラグは、DPSWレジスタの例外処理許可ビットDEX、DEVが“1”の場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
DTOI src, dest	D	DRs	DRd	4

【倍精度浮動小数点例外発生要因】

非実装処理
無効演算
精度異常

【記述例】

DTOI DR0, DR1

【動作補足説明】

- DPSWのDDNビットが“0”の場合と“1”の場合について、srcの値と変換結果の対応を以下に示します。

DDN = 0のとき

srcの値 (指数部はゲタなしの値)		dest (注1)	例外
src ≥ 0	+∞	7FFFFFFFh	無効演算
	31 ≤ 指数部 ≤ 1023		
	-1022 ≤ 指数部 ≤ 30	00000000h ~ 7FFFFFFFh	なし (注2、注3)
	+非正規化数	変化なし	非実装処理
	+0	00000000h	なし
src < 0	-0		非実装処理
	-非正規化数	変化なし	
	-1022 ≤ 指数部 ≤ 30	00000000h ~ 80000000h	なし (注2)
	31 ≤ 指数部 ≤ 1023	80000000h (注4)	無効演算 (注2、注5)
	-∞		
NaN	QNaN	符号ビット = 0 : 7FFFFFFFh	無効演算
	SNaN	符号ビット = 1 : 80000000h	

- 注1. 倍精度浮動小数点例外が発生し、対応するDEj (j = X, U, Z, O, V) が“1”の場合には変化しません。
 注2. 無効演算が発生しないとき、丸め発生時には精度異常が発生します。
 注3. 丸め後の値が7FFFFFFFhを超える場合には、無効演算が発生します。
 注4. 丸め後の値によらずdestは80000000hとなります。
 注5. 丸め後の値が80000000hとなる場合には、無効演算が発生しません。

DDN = 1のとき

srcの値 (指数部はゲタなしの値)		dest (注1)	例外
src ≥ 0	+∞	7FFFFFFFh	無効演算
	31 ≤ 指数部 ≤ 1023		
	-1022 ≤ 指数部 ≤ 30	00000000h ~ 7FFFFFFFh	なし (注2、注3)
	+非正規化数	00000000h	なし
	+0		
src < 0	-0		なし (注2)
	-非正規化数		
	-1022 ≤ 指数部 ≤ 30	00000000h ~ 80000000h	
	31 ≤ 指数部 ≤ 1023	80000000h (注4)	無効演算 (注2、注5)
	-∞		
NaN	QNaN	符号ビット = 0 : 7FFFFFFFh	無効演算
	SNaN	符号ビット = 1 : 80000000h	

- 注1. 倍精度浮動小数点例外が発生し、対応するDEj (j = X, U, Z, O, V) が“1”の場合には変化しません。
 注2. 無効演算が発生しないとき、丸め発生時には精度異常が発生します。
 注3. 丸め後の値が7FFFFFFFhを超える場合は、無効演算が発生します。
 注4. 丸め後の値によらずdestは80000000hとなります。
 注5. 丸め後の値が80000000hとなる場合には、無効演算が発生しません。

DTOU倍精度浮動小数点数→符号なし整数変換
Double TO Unsigned integer**DTOU****【構文】**

DTOU src, dest

倍精度浮動小数点演算命令

【命令コード】

記載ページ：352

【オペレーション】

dest = (unsigned long) src;

【機能】

src に格納された倍精度浮動小数点数を符号なしロングワード（32 ビット）整数に変換し、その結果を dest の下位 32 ビットに格納します。

- 結果はDPSWのDRM[1:0]ビットに関係なく0方向に丸められます。
- destの[63:32]ビットは不定になります。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	—	
O	—	
DCV	○	無効演算が発生したとき“1”、それ以外るとき“0”になります。
DCO	○	“0”になります。
DCZ	○	“0”になります。
DCU	○	“0”になります。
DCX	○	精度異常が発生したとき“1”、それ以外るとき“0”になります。
DCE	○	非実装処理が発生したとき“1”、それ以外るとき“0”になります。
DFV	○	無効演算が発生したとき“1”、それ以外るときは変化しません。
DFO	—	
DFZ	—	
DFU	—	
DFX	○	精度異常が発生したとき“1”、それ以外るときは変化しません。

注. DFX、DFVフラグは、DPSWレジスタの例外処理許可ビットDEX、DEVが“1”の場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
DTOU src, dest	D	DRs	DRd	4

【倍精度浮動小数点例外発生要因】

非実装処理
無効演算
精度異常

【記述例】

DTOU DR0, DR1

【動作補足説明】

- DPSWのDDNビットが“0”の場合と“1”の場合について、srcの値と変換結果の対応を以下に示します。

DDN = 0のとき

srcの値 (指数部はゲタなしの値)		dest (注1)	例外
src ≥ 0	+∞	FFFFFFFFh	無効演算
	32 ≤ 指数部 ≤ 1023		
	-1022 ≤ 指数部 ≤ 31	00000000h ~ FFFFFFFFh	なし (注2、注3)
	+非正規化数	変化なし	非実装処理
	+0	00000000h	なし
src < 0	-0		
	-非正規化数	変化なし	非実装処理
	-正規化数、-∞	00000000h	無効演算
NaN	QNaN	最上位ビット = 0: FFFFFFFFh	無効演算
	SNaN	最上位ビット = 1: 00000000h	

注1. 倍精度浮動小数点例外が発生し、対応するDEj (j = X, U, Z, O, V) が“1”の場合には変化しません。

注2. 丸め発生時は、精度異常が発生します。

注3. 丸め後の値がFFFFFFFFhを超える場合は、無効演算が発生します。

DDN = 1のとき

srcの値 (指数部はゲタなしの値)		dest (注1)	例外
src ≥ 0	+∞	FFFFFFFFh	無効演算
	32 ≤ 指数部 ≤ 1023		
	-1022 ≤ 指数部 ≤ 31	00000000h ~ FFFFFFFFh	なし (注2、注3)
	+0、+非正規化数	00000000h	なし
src < 0	-0、-非正規化数		
	-正規化数、-∞	00000000h	無効演算
NaN	QNaN	符号ビット = 0 : FFFFFFFFh	無効演算
	SNaN	符号ビット = 1 : 00000000h	

注1. 倍精度浮動小数点例外が発生し、対応するDEj (j = X, U, Z, O, V) が“1”の場合には変化しません。

注2. 丸め発生時は、精度異常が発生します。

注3. 丸め後の値がFFFFFFFFhを超える場合は、無効演算が発生します。

FTOD

単精度浮動小数点数→倍精度浮動小数点数変換
Float TO Double

FTOD**【構文】**

```
FTOD src, dest
```

倍精度浮動小数点演算命令

【命令コード】

記載ページ：353

【オペレーション】

```
dest = (double) src;
```

【機能】

srcに格納された単精度浮動小数点数を倍精度浮動小数点数に変換し、その結果をdestに格納します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	—	
O	—	
DCV	○	無効演算が発生したとき“1”、それ以外るとき“0”になります。
DCO	○	“0”になります。
DCZ	○	“0”になります。
DCU	○	“0”になります。
DCX	○	“0”になります。
DCE	○	非実装処理が発生したとき“1”、それ以外るとき“0”になります。
DFV	○	無効演算が発生したとき“1”、それ以外るときは変化しません。
DFO	—	
DFZ	—	
DFU	—	
DFX	—	

注. DFVフラグは、DPSWレジスタの例外処理許可ビットDEVが“1”の場合は変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
FTOD src, dest	D	Rs	DRd	4

【倍精度浮動小数点例外発生要因】

非実装処理
無効演算

【記述例】

```
FTOD R1, DR1
```

【動作補足説明】

- DPSWのDDNビットが“0”の場合と“1”の場合について、srcの値と変換結果の対応を以下に示します。

DDN = 0のとき

srcの値 (指数部はゲタなしの値)		dest (注1)	例外
src ≥ 0	+∞	+∞	なし
	-126 ≤ 指数部 ≤ 127	変換結果	なし
	+非正規化数	変化なし	非実装処理
	+0	+0	なし
src < 0	-0	-0	
	-非正規化数	変化なし	非実装処理
	-126 ≤ 指数部 ≤ 127	変換結果	なし
	-∞	-∞	なし
NaN	QNaN	QNaN (注2)	なし
	SNaN	SNaN (注3)	無効演算

注1. 倍精度浮動小数点例外が発生し、対応するDEj (j = X, U, Z, O, V) が“1”の場合には変化しません。

注2. destは入力のQNaNと同符号で、入力値の[22:0]ビットを仮数部の[51:29]にもち、仮数部の[28:0]が0のQNaNとなります。

注3. destは入力のSNaNと同符号で、入力値をQNaN化した値の[22:0]ビットを仮数部の[51:29]にもち、仮数部の[28:0]が0のQNaNとなります。

DDN = 1のとき

srcの値 (指数部はゲタなしの値)		dest (注1)	例外
src ≥ 0	+∞	+∞	なし
	-126 ≤ 指数部 ≤ 127	変換結果	なし
	+非正規化数	+0	なし
	+0	+0	なし
src < 0	-0	-0	
	-非正規化数	-0	なし
	-126 ≤ 指数部 ≤ 127	変換結果	なし
	-∞	-∞	なし
NaN	QNaN	QNaN (注2)	なし
	SNaN	SNaN (注3)	無効演算

注1. 倍精度浮動小数点例外が発生し、対応するDEj (j = X, U, Z, O, V) が“1”の場合には変化しません。

注2. destは入力のQNaNと同符号で、入力値の[22:0]ビットを仮数部の[51:29]にもち、仮数部の[28:0]が“0”のQNaNとなります。

注3. destは入力のSNaNと同符号で、入力値をQNaN化した値の[22:0]ビットを仮数部の[51:29]にもち、仮数部の[28:0]が“0”のQNaNとなります。

ITOD

符号付き整数→倍精度浮動小数点数変換
signed Integer TO Double

ITOD

【構文】

```
ITOD src, dest
```

倍精度浮動小数点演算命令

【命令コード】

記載ページ：353

【オペレーション】

```
dest = (double) src;
```

【機能】

srcに格納された符号付きロングワード（32ビット）整数を倍精度浮動小数点数に変換し、その結果をdestに格納します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	—	
O	—	
DCV	○	“0”になります。
DCO	○	“0”になります。
DCZ	○	“0”になります。
DCU	○	“0”になります。
DCX	○	“0”になります。
DCE	○	“0”になります。
DFV	—	
DFO	—	
DFZ	—	
DFU	—	
DFX	—	

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
ITOD src, dest	D	Rs	DRd	4

【倍精度浮動小数点例外発生要因】

なし

【記述例】

```
ITOD R1, DR0
```

MVFDC

倍精度浮動小数点制御レジスタからの転送
MoVe From Double precision floating-point
Control register

MVFDC

倍精度浮動小数点転送命令

【命令コード】

記載ページ：354

【構文】

MVFDC src, dest

【オペレーション】

dest = src;

【機能】

srcをdestに転送します。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
MVFDC src, dest	L	DCRs	Rd	4

注. 選択可能な src : DPSW, DCMR, DECNT, DEPC

【倍精度浮動小数点例外発生要因】

なし

【記述例】

MVFDC DPSW, R2

MVFDR

倍精度浮動小数点比較結果レジスタからの転送
MoVe From Double precision floating-point
compare Result

MVFDR

倍精度浮動小数点転送命令

【命令コード】

記載ページ：354

【構文】

MVFDR

【オペレーション】

Z = DCMR.RES;

【機能】

DCMR の RES ビットを PWS の Z フラグに転送します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	DCMR.RESの内容が書き込まれます。
S	—	
O	—	

注. DPSWレジスタは変化しません。

【命令フォーマット】

構文	コードサイズ (バイト)
MVFDR	3

【倍精度浮動小数点例外発生要因】

なし

【記述例】

MVFDR

MVTDC

倍精度浮動小数点制御レジスタへの転送
MoVe To Double precision floating-point Control register

MVTDC**【構文】**

MVTDC src, dest

倍精度浮動小数点転送命令

【命令コード】

記載ページ : 355

【オペレーション】

dest = src;

【機能】

srcをdestに転送します。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
MVTDC src, dest	L	Rs	DCRd	4

注. 選択可能なdest : DPSW, DCMR, DECNT, DEPC

【倍精度浮動小数点例外発生要因】

なし

【記述例】

MVTDC R1,DPSW

UTOD

符号なし整数→倍精度浮動小数点数変換
Unsigned integer TO Double

UTOD

【構文】

UTOD src, dest

倍精度浮動小数点演算命令

【命令コード】

記載ページ：355

【オペレーション】

dest = (double) src;

【機能】

srcに格納された符号なしロングワード（32ビット）整数を倍精度浮動小数点数に変換し、その結果をdestに格納します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	—	
O	—	
DCV	○	“0”になります。
DCO	○	“0”になります。
DCZ	○	“0”になります。
DCU	○	“0”になります。
DCX	○	“0”になります。
DCE	○	“0”になります。
DFV	—	
DFO	—	
DFZ	—	
DFU	—	
DFX	—	

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ (バイト)
		src	dest	
UTOD src, dest	D	Rs	DRd	4

【倍精度浮動小数点例外発生要因】

なし

【記述例】

UTOD R1, DR0

4. 命令コード

4.1 本章の見方

本章は、命令コードをオペコードごとに説明しています。
本章の見方を以下に実例をあげて示します。

(1) **ADD** **ADD**

【コードサイズ】

構文	src	src2	dest	コードサイズ (バイト)
(1) ADD src, dest (3オペランドの命令コードになります。)	#UIMM:4	—	Rd	2
	#SIMM:8	—	Rd	3
	#SIMM:16	—	Rd	4
	#SIMM:24	—	Rd	5
	#IMM:32	—	Rd	6
(2) ADD src, dest	Rs	—	Rd	2
	[Rs].memex	—	Rd	2 (memex == "UB") 3 (memex != "UB")
	dsp:8[Rs].memex	—	Rd	3 (memex == "UB") 4 (memex != "UB")
	dsp:16[Rs].memex	—	Rd	4 (memex == "UB") 5 (memex != "UB")
(3) ADD src, src2, dest	#SIMM:8	Rs	Rd	3
	#SIMM:16	Rs	Rd	4
	#SIMM:24	Rs	Rd	5
	#IMM:32	Rs	Rd	6
(4) ADD src, src2, dest	Rs	Rs2	Rd	3

(2) (1) **ADD src, dest**

b7	b0 b7	b0
0 1 1 0 0 0 1 0	imm[3:0]	rd[3:0]

imm[3:0] 0000b ~ 1111b	src #UIMM:4 0 ~ 15	rd[3:0] 0000b ~ 1111b	dest Rd R0(SP) ~ R15
---------------------------	-----------------------	--------------------------	-------------------------

(3) (2) **ADD src, dest**

memex == "UB" または src == Rs の場合

b7	b0 b7	b0		ld[1:0]	src
0 1 0 0 0 1 0	ld[1:0]	rs[3:0]	rd[3:0]	11b	(なし)
				00b	(なし)
				01b	dsp:8
				10b	dsp:16

memex != "UB" の場合

b7	memex	b0 b7	b0 b7	b0 b7	b0		ld[1:0]	src
0 0 0 0 0 1 1 0	mi[1:0]	0 0 1 0	ld[1:0]	rs[3:0]	rd[3:0]	11b	(なし)	(なし)
						00b	(なし)	(なし)
						01b	dsp:8	(なし)
						10b	dsp:16	(なし)

mi[1:0]	memex	ld[1:0]	src	rs[3:0]/rd[3:0]	src/dest
00b	B	11b	Rs	0000b ~ 1111b	Rs/Rd R0(SP) ~ R15
01b	W	00b	[Rs]		
10b	L	01b	dsp:8[Rs]		
11b	UW	10b	dsp:16[Rs]		

(1) ニーモニック

本ページで説明するニーモニックを示しています。

(2) コードサイズ表

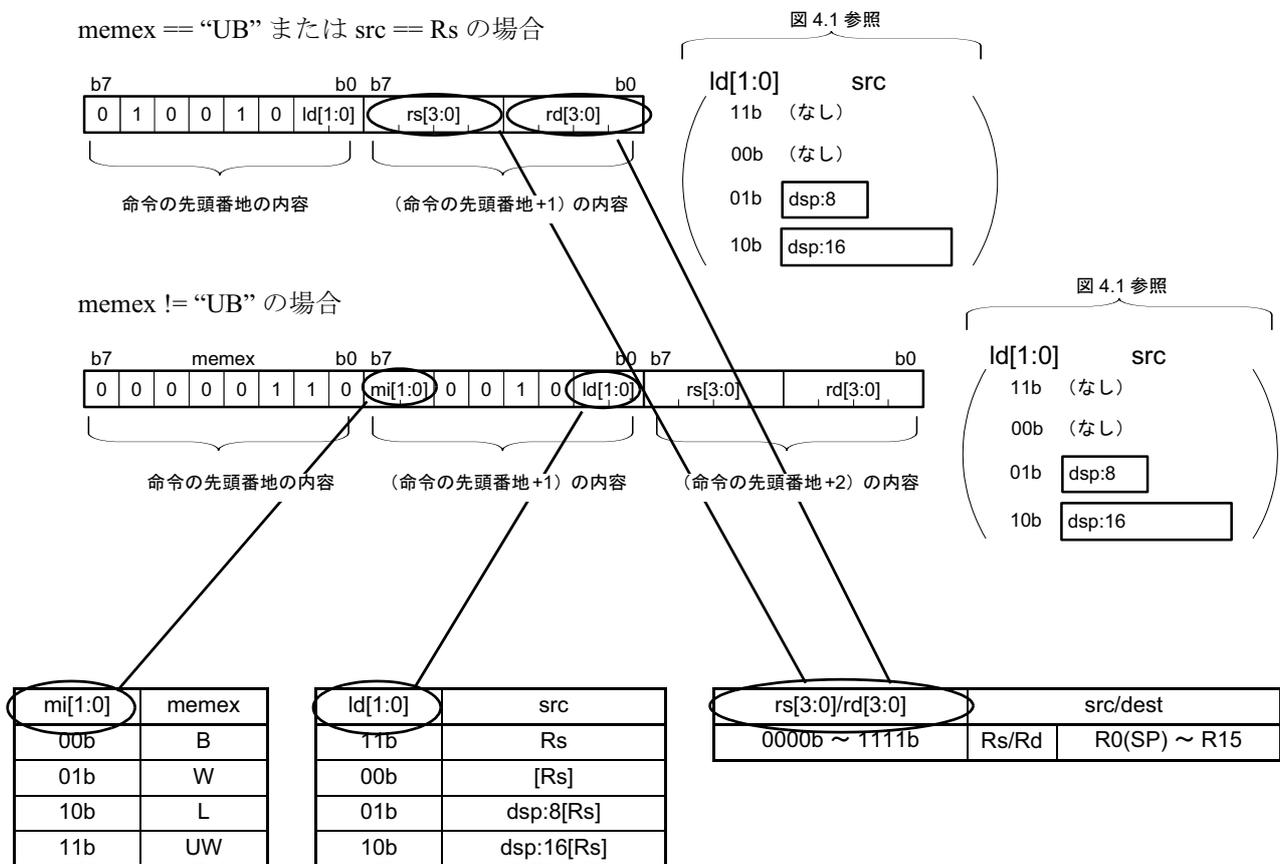
命令のバイト数を示しています。RXv3 CPUの命令のバイト数は、1バイト～8バイトです。

(3) 構文

命令の構文を記号で示しています。

(4) 命令コード

命令コードを示しています。() 内は選択するsrc/destによって選択、または省略されます。



src/destの内容（前ページの例では（命令の先頭番地+2）または（命令の先頭番地+3）以降）は、
図4.1に示すように配置されます。

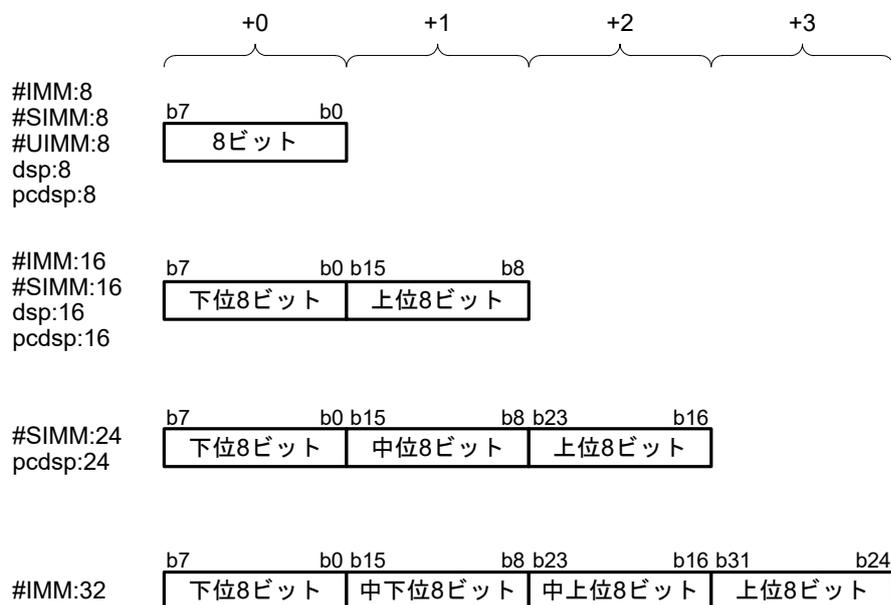


図 4.1 即値（IMM）とディスプレイースメント（dsp）

rs、rd、ld、miなどの略号は、以下を意味します。

- rs : Source register
- rs2 : Second source register
- rd : Destination register
- rd2 : Second destination register
- ri : Index register
- rb : Base register
- li : Length of immediate
- ld : Length of displacement
- lds : Length of source displacement
- ldd : Length of destination displacement
- mi : Memory extension size infix
- imm : Immediate
- dsp : Displacement
- cd : Condition code
- cr : Control register
- cb : Control bit
- sz : Size specifier
- ad : Addressing
- nm : Number of registers
- cm : Compare condition

4.2 命令コード詳細説明

RXv3 各命令の命令コードの詳細説明を示します。

4.2.1 標準搭載命令

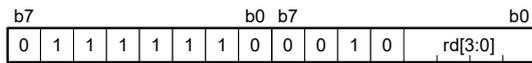
次ページより標準搭載命令の命令コードの詳細説明を示します。

ABS

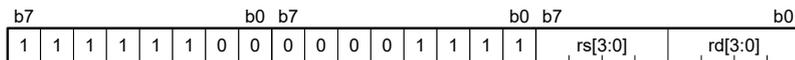
ABS

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) ABS dest	—	Rd	2
(2) ABS src, dest	Rs	Rd	3

(1) ABS dest

rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

(2) ABS src, dest

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

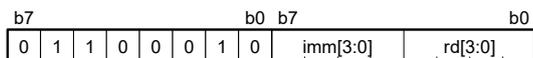
ADD

ADD

【コードサイズ】

構文	src	src2	dest	コードサイズ (バイト)
(1) ADD src, dest	#UIMM:4	—	Rd	2
(3オペランドの 命令コードになります。)	#SIMM:8	—	Rd	3
	#SIMM:16	—	Rd	4
	#SIMM:24	—	Rd	5
	#IMM:32	—	Rd	6
(2) ADD src, dest	Rs	—	Rd	2
	[Rs].memex	—	Rd	2 (memex == "UB") 3 (memex != "UB")
	dsp:8[Rs].memex	—	Rd	3 (memex == "UB") 4 (memex != "UB")
	dsp:16[Rs].memex	—	Rd	4 (memex == "UB") 5 (memex != "UB")
(3) ADD src, src2, dest	#SIMM:8	Rs	Rd	3
	#SIMM:16	Rs	Rd	4
	#SIMM:24	Rs	Rd	5
	#IMM:32	Rs	Rd	6
(4) ADD src, src2, dest	Rs	Rs2	Rd	3

(1) ADD src, dest

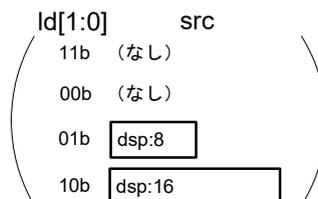
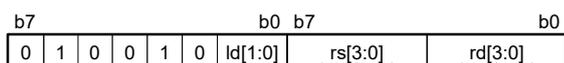


imm[3:0]	src
0000b ~ 1111b	#UIMM:4 0 ~ 15

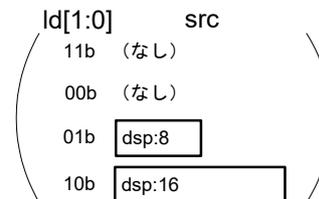
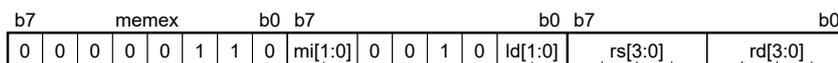
rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

(2) ADD src, dest

memex == "UB" または src == Rs の場合



memex != "UB" の場合

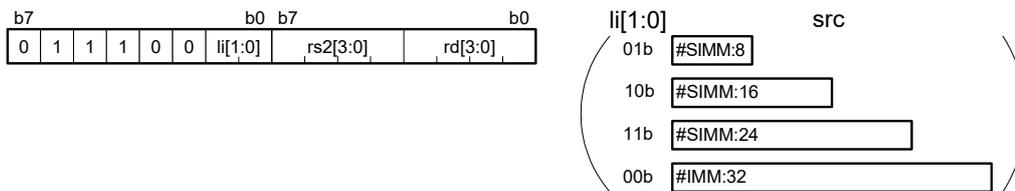


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

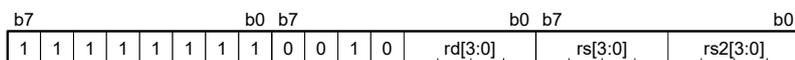
(3) ADD src, src2, dest



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rs2[3:0]/rd[3:0]	src2/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

(4) ADD src, src2, dest



rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ~ 1111b	Rs/Rs2/Rd	R0(SP) ~ R15

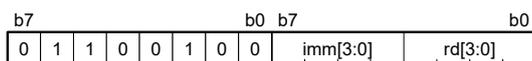
AND

AND

【コードサイズ】

構文	src	src2	dest	コードサイズ (バイト)
(1) AND src, dest	#UIMM:4	—	Rd	2
(2) AND src, dest	#SIMM:8	—	Rd	3
	#SIMM:16	—	Rd	4
	#SIMM:24	—	Rd	5
	#IMM:32	—	Rd	6
(3) AND src, dest	Rs	—	Rd	2
	[Rs].memex	—	Rd	2 (memex == "UB") 3 (memex != "UB")
	dsp:8[Rs].memex	—	Rd	3 (memex == "UB") 4 (memex != "UB")
	dsp:16[Rs].memex	—	Rd	4 (memex == "UB") 5 (memex != "UB")
(4) AND src, src2, dest	Rs	Rs2	Rd	3

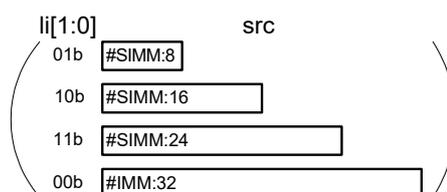
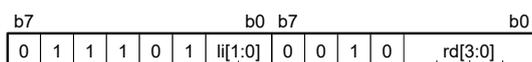
(1) AND src, dest



imm[3:0]	src
0000b ~ 1111b	#UIMM:4 0 ~ 15

rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

(2) AND src, dest



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

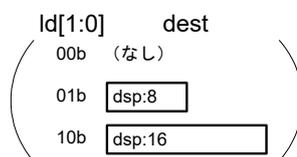
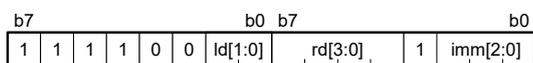
BCLR

BCLR

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) BCLR src, dest	#IMM:3	[Rd].B	2
	#IMM:3	dsp:8[Rd].B	3
	#IMM:3	dsp:16[Rd].B	4
(2) BCLR src, dest	Rs	[Rd].B	3
	Rs	dsp:8[Rd].B	4
	Rs	dsp:16[Rd].B	5
(3) BCLR src, dest	#IMM:5	Rd	2
(4) BCLR src, dest	Rs	Rd	3

(1) BCLR src, dest

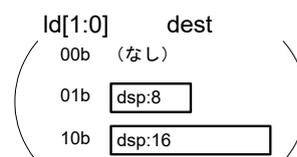
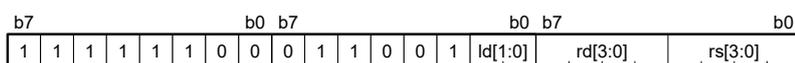


ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

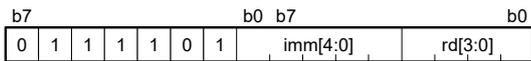
imm[2:0]	src
000b ~ 111b	#IMM:3 0 ~ 7

(2) BCLR src, dest



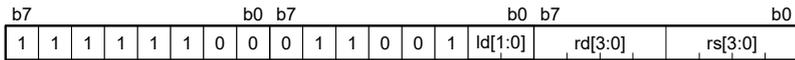
ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

(3) BCLR src, dest

imm[4:0]				src			
00000b ~ 11111b				#IMM:5	0 ~ 31		

rd[3:0]				dest			
0000b ~ 1111b				Rd	R0(SP) ~ R15		

(4) BCLR src, dest

ld[1:0]		dest	
11b		Rd	

rs[3:0]/rd[3:0]				src/dest			
0000b ~ 1111b				Rs/Rd	R0(SP) ~ R15		

BCnd

BCnd

【コードサイズ】

構文	src	コードサイズ (バイト)
(1) BCnd.S src	pcdsp:3	1
(2) BCnd.B src	pcdsp:8	2
(3) BCnd.W src	pcdsp:16	3

(1) BCnd.S src

b7	b0
0	0
0	0
1	cd
dsp[2:0] ^(注)	

注. dsp[2:0] で pcdsp:3 = src を指定します。

cd	BCnd
0b	BEQ, BZ
1b	BNE, BNZ

dsp[2:0]	分岐距離
011b	3
100b	4
101b	5
110b	6
111b	7
000b	8
001b	9
010b	10

(2) BCnd.B src

b7	b0	src
0	0	pcdsp:8 ^(注)
1	0	
cd[3:0]		

注. pcdsp:8 = src が示す番地 — 命令の先頭番地

cd[3:0]	BCnd	cd[3:0]	BCnd
0000b	BEQ, BZ	1000b	BGE
0001b	BNE, BNZ	1001b	BLT
0010b	BGEU, BC	1010b	BGT
0011b	BLTU, BNC	1011b	BLE
0100b	BGTU	1100b	BO
0101b	BLEU	1101b	BNO
0110b	BPZ	1110b	BRA.B
0111b	BN	1111b	(予約)

(3) BCnd.W src

b7	b0	src
0	0	pcdsp:16 ^(注)
1	1	
1	0	
1	1	
cd		

注. pcdsp:16 = src が示す番地 — 命令の先頭番地

cd	BCnd
0b	BEQ, BZ
1b	BNE, BNZ

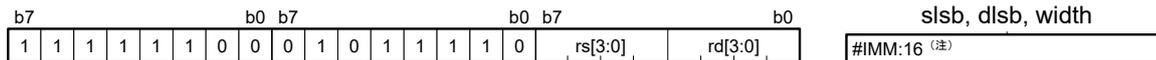
BFMOV

BFMOV

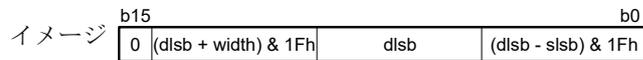
【コードサイズ】

構文	対象					コードサイズ (バイト)
	slsb	dlsb	width	src	dest	
(1) BFMOV slsb, dlsb, width, src, dest	#IMM:5	#IMM:5	#IMM:5	Rs	Rd	5

(1) BFMOV slsb, dlsb, width, src, dest



注. 命令オペランドの slsb, dlsb, width の値に従って、次の規則で生成してください。
 $\#IMM:16 = (((dlsb + width) \& 1Fh) \ll 10) \mid (dlsb \ll 5) \mid ((dlsb - slsb) \& 1Fh)$



rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

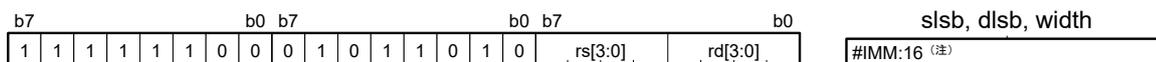
BFMOVZ

BFMOVZ

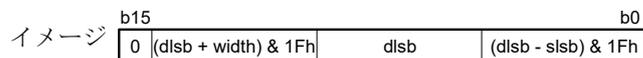
【コードサイズ】

構文	対象					コードサイズ (バイト)
	slsb	dlsb	width	src	dest	
(1) BFMOVZ slsb, dlsb, width, src, dest	#IMM:5	#IMM:5	#IMM:5	Rs	Rd	5

(1) BFMOVZ slsb, dlsb, width, src, dest



注. 命令オペランドの slsb, dlsb, width の値に従って、次の規則で生成してください。
 $\#IMM:16 = (((dlsb + width) \& 1Fh) \ll 10) \mid (dlsb \ll 5) \mid ((dlsb - slsb) \& 1Fh)$



rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

BM*Cnd***BM*Cnd***

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) <i>BM<i>Cnd</i></i> src, dest	#IMM:3	[Rd].B	3
	#IMM:3	dsp:8[Rd].B	4
	#IMM:3	dsp:16[Rd].B	5
(2) <i>BM<i>Cnd</i></i> src, dest	#IMM:5	Rd	3

(1) *BM*Cnd** src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0 0 1 1 1	imm[2:0]	ld[1:0]	rd[3:0]
			cd[3:0]

ld[1:0]	dest
00b	(なし)
01b	dsp:8
10b	dsp:16

imm[2:0]	src
000b ~ 111b	#IMM:3 0 ~ 7

ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

cd[3:0]	<i>BM<i>Cnd</i></i>	cd[3:0]	<i>BM<i>Cnd</i></i>
0000b	BMEQ, BMZ	1000b	BMGE
0001b	BMNE, BMNZ	1001b	BMLT
0010b	BMGEU, BMC	1010b	BMGT
0011b	BMLTU, BMNC	1011b	BMLE
0100b	BMGTU	1100b	BMO
0101b	BMLEU	1101b	BMNO
0110b	BMPZ	1110b	(予約)
0111b	BMN	1111b	(予約)

(2) *BM*Cnd** src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0 1 1 1 1	imm[4:0]	cd[3:0]	rd[3:0]

imm[4:0]	src
00000b ~ 11111b	#IMM:5 0 ~ 31

cd[3:0]	<i>BM<i>Cnd</i></i>	cd[3:0]	<i>BM<i>Cnd</i></i>
0000b	BMEQ, BMZ	1000b	BMGE
0001b	BMNE, BMNZ	1001b	BMLT
0010b	BMGEU, BMC	1010b	BMGT
0011b	BMLTU, BMNC	1011b	BMLE
0100b	BMGTU	1100b	BMO
0101b	BMLEU	1101b	BMNO
0110b	BMPZ	1110b	(予約)
0111b	BMN	1111b	(予約)

rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

BNOT

BNOT

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) BNOT src, dest	#IMM:3	[Rd].B	3
	#IMM:3	dsp:8[Rd].B	4
	#IMM:3	dsp:16[Rd].B	5
(2) BNOT src, dest	Rs	[Rd].B	3
	Rs	dsp:8[Rd].B	4
	Rs	dsp:16[Rd].B	5
(3) BNOT src, dest	#IMM:5	Rd	3
(4) BNOT src, dest	Rs	Rd	3

(1) BNOT src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0	0 0 1 1 1	imm[2:0]	ld[1:0]
		rd[3:0]	1 1 1 1

ld[1:0]	dest
00b	(なし)
01b	dsp:8
10b	dsp:16

imm[2:0]	src
000b ~ 111b	#IMM:3 0 ~ 7

ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

(2) BNOT src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0	0 0 0 1 1 0 1 1	ld[1:0]	rd[3:0]
		rs[3:0]	

ld[1:0]	dest
00b	(なし)
01b	dsp:8
10b	dsp:16

ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

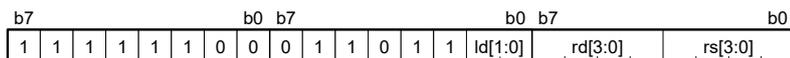
rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

(3) BNOT src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0	1 1 1 1 1	imm[4:0]	rd[3:0]

imm[4:0]	src
00000b ~ 11111b	#IMM:5 0 ~ 31

rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

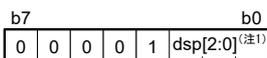
(4) BNOT src, dest

ld[1:0]	dest
11b	Rd

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

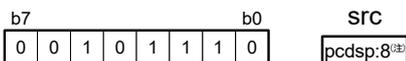
BRA**BRA****【コードサイズ】**

構文	src	コードサイズ (バイト)
(1) BRA.S src	pcdsp:3	1
(2) BRA.B src	pcdsp:8	2
(3) BRA.W src	pcdsp:16	3
(4) BRA.A src	pcdsp:24	4
(5) BRA.L src	Rs	2

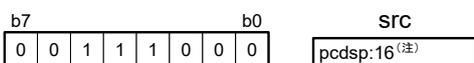
(1) BRA.S src

注. dsp[2:0] で pcdsp:3 = src を指定します。

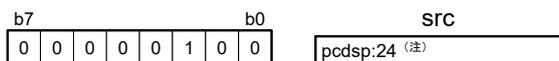
dsp[2:0]	分岐距離
011b	3
100b	4
101b	5
110b	6
111b	7
000b	8
001b	9
010b	10

(2) BRA.B src

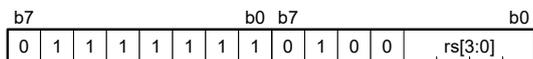
注. pcdsp:8 = src が示す番地 - 命令の先頭番地

(3) BRA.W src

注. pcdsp:16 = src が示す番地 - 命令の先頭番地

(4) BRA.A src

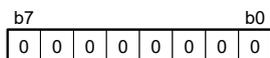
注 . pcdsp:24 = src が示す番地 - 命令の先頭番地

(5) BRA.L src

rs[3:0]	src	
0000b ~ 1111b	Rs	R0(SP) ~ R15

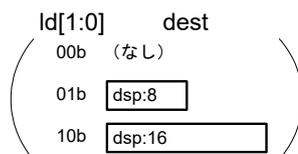
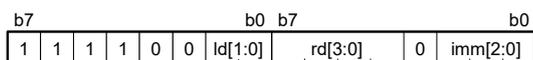
BRK**BRK****【コードサイズ】**

構文	コードサイズ (バイト)
(1) BRK	1

(1) BRK**BSET****BSET****【コードサイズ】**

構文	src	dest	コードサイズ (バイト)
(1) BSET src, dest	#IMM:3	[Rd].B	2
	#IMM:3	dsp:8[Rd].B	3
	#IMM:3	dsp:16[Rd].B	4
(2) BSET src, des	Rs	[Rd].B	3
	Rs	dsp:8[Rd].B	4
	Rs	dsp:16[Rd].B	5
(3) BSET src, dest	#IMM:5	Rd	2
(4) BSET src, dest	Rs	Rd	3

(1) BSET src, dest

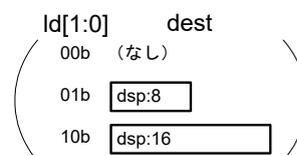
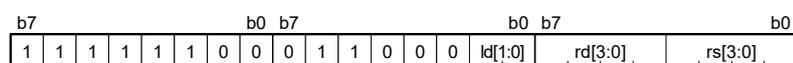


ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

imm[2:0]	src
000b ~ 111b	#IMM:3 0 ~ 7

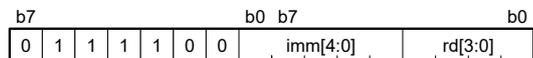
(2) BSET src, dest



ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

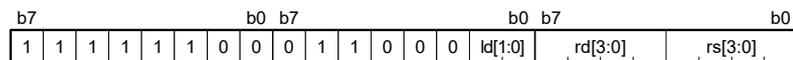
(3) BSET src, dest



imm[4:0]	src
00000b ~ 11111b	#IMM:5 0 ~ 31

rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

(4) BSET src, dest



ld[1:0]	dest
11b	Rd

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

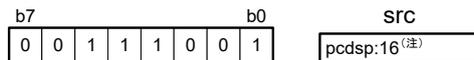
BSR

BSR

【コードサイズ】

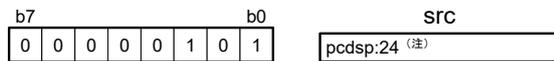
構文	src	コードサイズ (バイト)
(1) BSR.W src	pcdsp:16	3
(2) BSR.A src	pcdsp:24	4
(3) BSR.L src	Rs	2

(1) BSR.W src



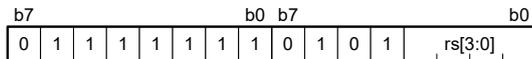
注 . pcdsp:16 = src が示す番地 - 命令の先頭番地

(2) BSR.A src



注 . pcdsp:24 = src が示す番地 - 命令の先頭番地

(3) BSR.L src



rs[3:0]	src	
0000b ~ 1111b	Rs	R0(SP) ~ R15

BTST

BTST

【コードサイズ】

構文	src	src2	コードサイズ (バイト)
(1) BTST src, src2	#IMM:3	[Rs].B	2
	#IMM:3	dsp:8[Rs].B	3
	#IMM:3	dsp:16[Rs].B	4
(2) BTST src, src2	Rs	[Rs2].B	3
	Rs	dsp:8[Rs2].B	4
	Rs	dsp:16[Rs2].B	5
(3) BTST src, src2	#IMM:5	Rs	2
(4) BTST src, src2	Rs	Rs2	3

(1) BTST src, src2

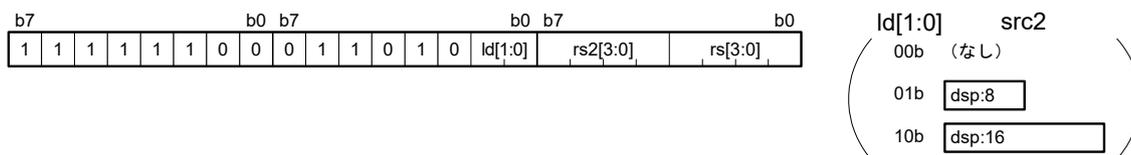


ld[1:0]	src2
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]	src2
0000b ~ 1111b	Rs R0(SP) ~ R15

imm[2:0]	src
000b ~ 111b	#IMM:3 0 ~ 7

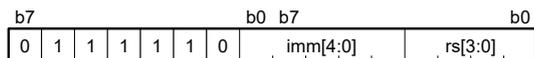
(2) BTST src, src2



ld[1:0]	src2
00b	[Rs2]
01b	dsp:8[Rs2]
10b	dsp:16[Rs2]

rs[3:0]/rs2[3:0]	src/src2
0000b ~ 1111b	Rs/Rs2 R0(SP) ~ R15

(3) BTST src, src2



imm[4:0]	src
00000b ~ 11111b	#IMM:5 0 ~ 31

rs[3:0]	src2
0000b ~ 1111b	Rs R0(SP) ~ R15

(4) BTST src, src2

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0 0 0 1 1 0 1 0	ld[1:0]	rs2[3:0]	rs[3:0]

ld[1:0]	src2
11b	Rs2

rs[3:0]/rs2[3:0]	src/src2
0000b ~ 1111b	Rs/Rs2 R0(SP) ~ R15

CLRPSW**CLRPSW****【コードサイズ】**

構文	dest	コードサイズ (バイト)
(1) CLRPSW dest	flag	2

(1) CLRPSW dest

b7	b0 b7	b0
0 1 1 1 1 1 1 1 1 1 0 1 1	cb[3:0]	

cb[3:0]	dest	
0000b	flag	C
0001b		Z
0010b		S
0011b		O
0100b		(予約)
0101b		(予約)
0110b		(予約)
0111b		(予約)
1000b		I
1001b		U
1010b		(予約)
1011b		(予約)
1100b		(予約)
1101b		(予約)
1110b		(予約)
1111b		(予約)

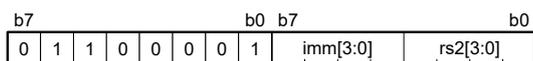
CMP

CMP

【コードサイズ】

構文	src	src2	コードサイズ (バイト)
(1) CMP src, src2	#UIMM:4	Rs	2
(2) CMP src, src2	#UIMM:8	Rs	3
(3) CMP src, src2	#SIMM:8	Rs	3
	#SIMM:16	Rs	4
	#SIMM:24	Rs	5
	#IMM:32	Rs	6
(4) CMP src, src2	Rs	Rs2	2
	[Rs].memex	Rs2	2 (memex == "UB") 3 (memex != "UB")
	dsp:8[Rs].memex	Rs2	3 (memex == "UB") 4 (memex != "UB")
	dsp:16[Rs].memex	Rs2	4 (memex == "UB") 5 (memex != "UB")

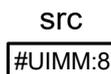
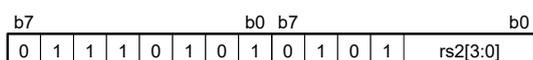
(1) CMP src, src2



imm[3:0]	src
0000b ~ 1111b	#UIMM:4 0 ~ 15

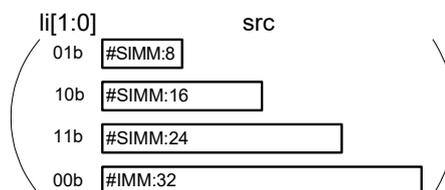
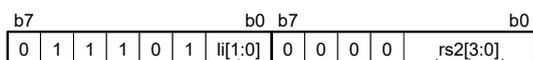
rs2[3:0]	src2
0000b ~ 1111b	Rs R0(SP) ~ R15

(2) CMP src, src2



rs2[3:0]	src2
0000b ~ 1111b	Rs R0(SP) ~ R15

(3) CMP src, src2

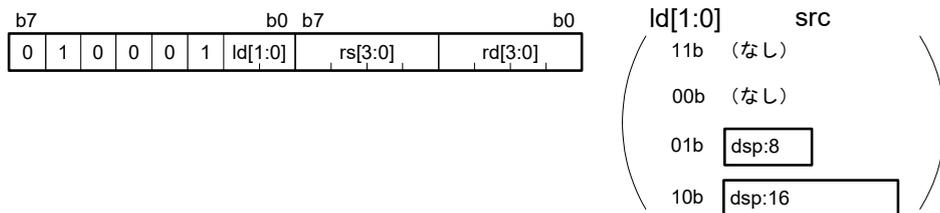


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

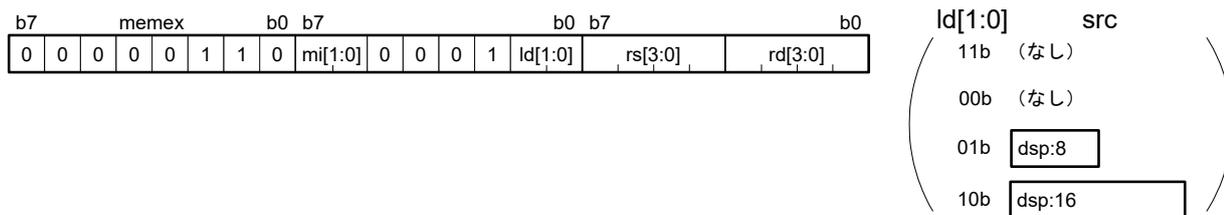
rs2[3:0]	src2
0000b ~ 1111b	Rs R0(SP) ~ R15

(4) CMP src, src2

memex == “UB” または src == Rs の場合



memex != “UB” の場合



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rs2[3:0]	src/src2	
0000b ~ 1111b	Rs/Rs2	R0(SP) ~ R15

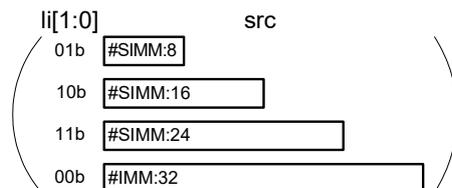
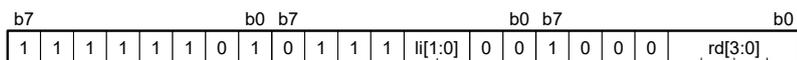
DIV

DIV

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) DIV src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) DIV src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == “UB”) 4 (memex != “UB”)
	dsp:8[Rs].memex	Rd	4 (memex == “UB”) 5 (memex != “UB”)
	dsp:16[Rs].memex	Rd	5 (memex == “UB”) 6 (memex != “UB”)

(1) DIV src, dest

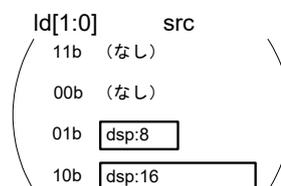
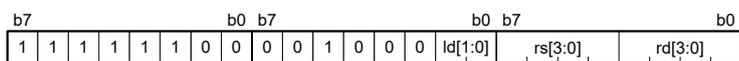


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

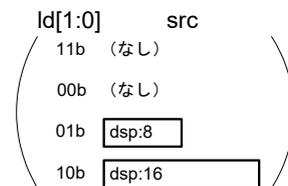
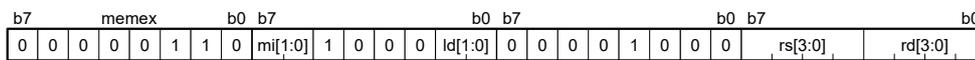
rd[3:0]	dest	
0000b ~ 1111b	Rd	R0(SP) ~ R15

(2) DIV src, dest

memex == “UB” または src == Rs の場合



memex != “UB” の場合



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

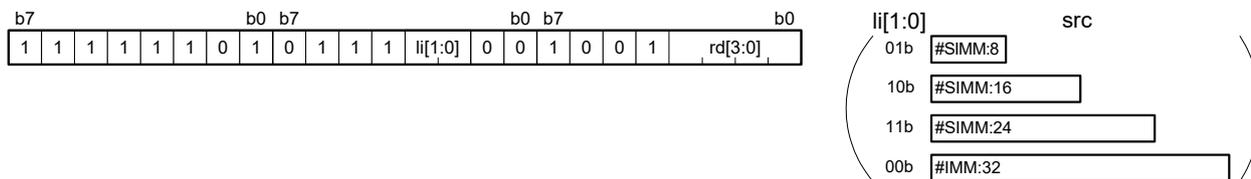
DIVU

DIVU

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) DIVU src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) DIVU src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == "UB") 4 (memex != "UB")
	dsp:8[Rs].memex	Rd	4 (memex == "UB") 5 (memex != "UB")
	dsp:16[Rs].memex	Rd	5 (memex == "UB") 6 (memex != "UB")

(1) DIVU src, dest

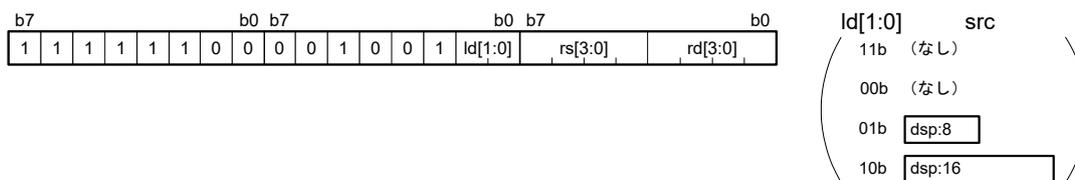


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

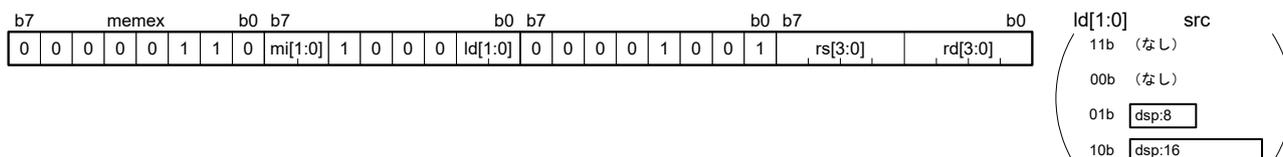
rd[3:0]	dest	
0000b ~ 1111b	Rd	R0(SP) ~ R15

(2) DIVU src, dest

memex == "UB" または src == Rs の場合



memex != "UB" の場合



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

EMACA

EMACA

【コードサイズ】

構文	src	dest2	Adest	コードサイズ (バイト)
(1) EMACA src, src2, Adest	Rs	Rs2	A0, A1	3

(1) EMACA src, src2, Adest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0	1 0 0 0 0 a 1 1 1	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b ~ 1111b	Rs/Rs2	R0(SP) ~ R15

EMSBA

EMSBA

【コードサイズ】

構文	src	dest2	Adest	コードサイズ (バイト)
(1) EMSBA src, src2, Adest	Rs	Rs2	A0, A1	3

(1) EMSBA src, src2, Adest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0	1 0 1 0 0 a 1 1 1	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b ~ 1111b	Rs/Rs2	R0(SP) ~ R15

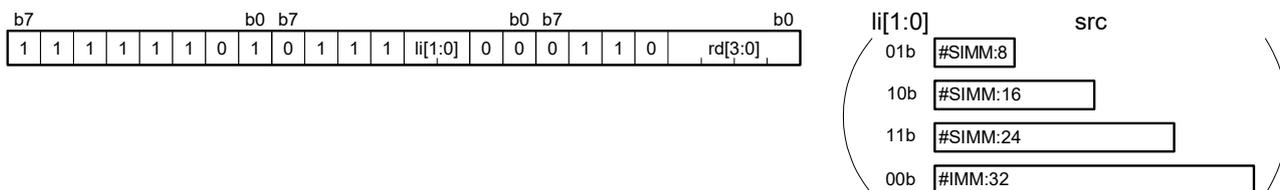
EMUL

EMUL

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) EMUL src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) EMUL src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == "UB") 4 (memex != "UB")
	dsp:8[Rs].memex	Rd	4 (memex == "UB") 5 (memex != "UB")
	dsp:16[Rs].memex	Rd	5 (memex == "UB") 6 (memex != "UB")

(1) EMUL src, dest

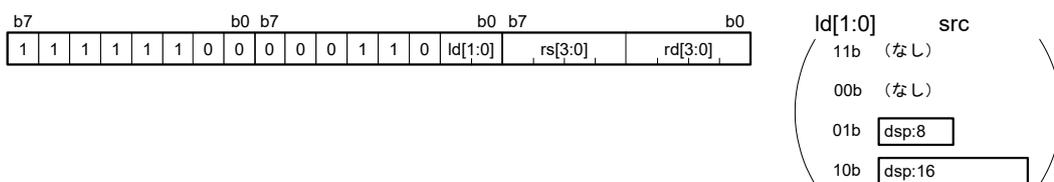


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

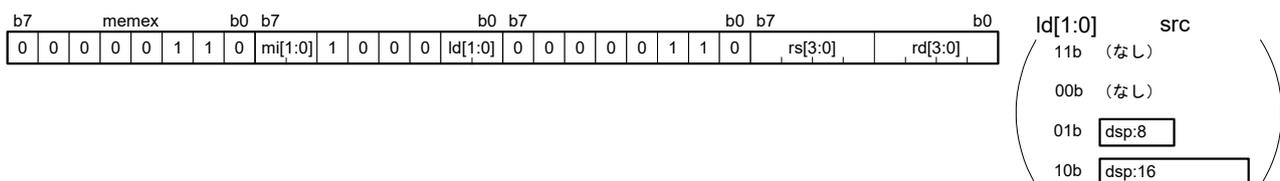
rd[3:0]	dest
0000b ~ 1110b	Rd R0(SP) ~ R14

(2) EMUL src, dest

memex == "UB" または src == Rs の場合



memex != "UB" の場合



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]	src
0000b ~ 1111b	Rs R0(SP) ~ R15
rd[3:0]	dest
0000b ~ 1110b	Rd R0(SP) ~ R14

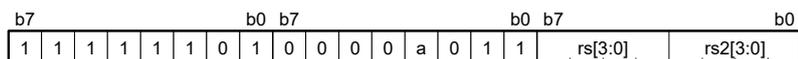
EMULA

EMULA

【コードサイズ】

構文	src	dest2	Adest	コードサイズ (バイト)
(1) EMULA src, src2, Adest	Rs	Rs2	A0, A1	3

(1) EMULA src, src2, Adest



a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b ~ 1111b	Rs/Rs2	R0(SP) ~ R15

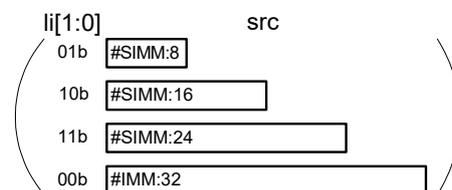
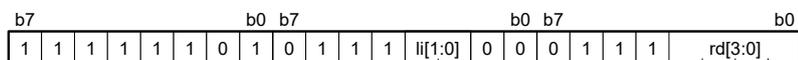
EMULU

EMULU

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) EMULU src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) EMULU src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == "UB") 4 (memex != "UB")
	dsp:8[Rs].memex	Rd	4 (memex == "UB") 5 (memex != "UB")
	dsp:16[Rs].memex	Rd	5 (memex == "UB") 6 (memex != "UB")

(1) EMULU src, dest



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

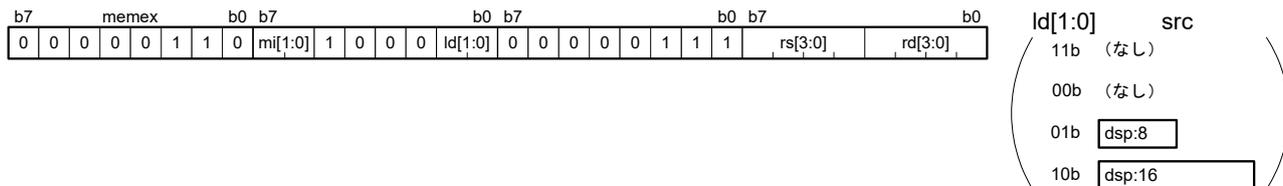
rd[3:0]	dest	
0000b ~ 1110b	Rd	R0(SP) ~ R14

(2) EMULU src, dest

memex == “UB” または src == Rs の場合



memex != “UB” の場合



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]	src	
0000b ~ 1111b	Rs	R0(SP) ~ R15
rd[3:0]	dest	
0000b ~ 1110b	Rd	R0(SP) ~ R14

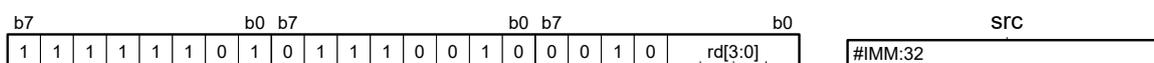
FADD

FADD

【コードサイズ】

構文	src	src2	dest	コードサイズ (バイト)
(1) FADD src, dest	#IMM:32	—	Rd	7
(2) FADD src, dest	Rs	—	Rd	3
	[Rs].L	—	Rd	3
	dsp:8[Rs].L	—	Rd	4
	dsp:16[Rs].L	—	Rd	5
(3) FADD src, src2, dest	Rs	Rs2	Rd	3

(1) FADD src, dest



rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

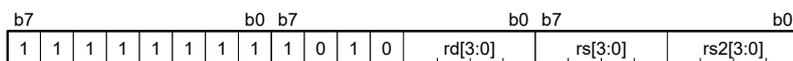
(2) FADD src, dest



ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

(3) FADD src, src2, dest



rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest
0000b ~ 1111b	Rs/Rs2/Rd R0(SP) ~ R15

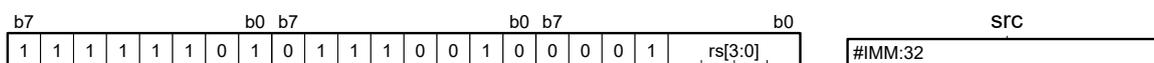
FCMP

FCMP

【コードサイズ】

構文	src	src2	コードサイズ (バイト)
(1) FCMP src, src2	#IMM:32	Rs	7
(2) FCMP src, src2	Rs	Rs2	3
	[Rs].L	Rs2	3
	dsp:8[Rs].L	Rs2	4
	dsp:16[Rs].L	Rs2	5

(1) FCMP src, src2



rs[3:0]	Rs	src2
0000b ~ 1111b	Rs	R0(SP) ~ R15

(2) FCMP src, src2



ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rs2[3:0]	Rs/Rs2	src/src2
0000b ~ 1111b	Rs/Rs2	R0(SP) ~ R15

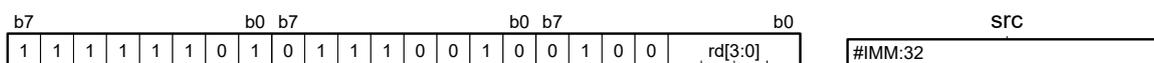
FDIV

FDIV

【コードサイズ】

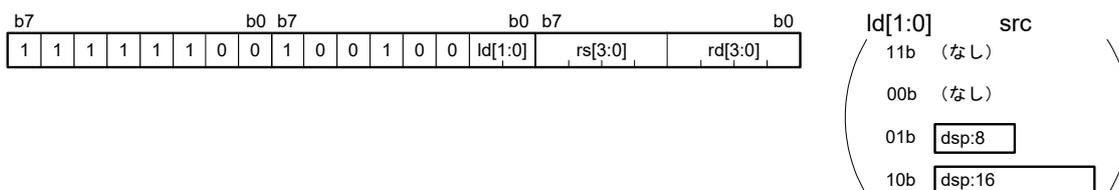
構文	src	dest	コードサイズ (バイト)
(1) FDIV src, dest	#IMM:32	Rd	7
(2) FDIV src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(1) FDIV src, dest



rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

(2) FDIV src, dest



ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

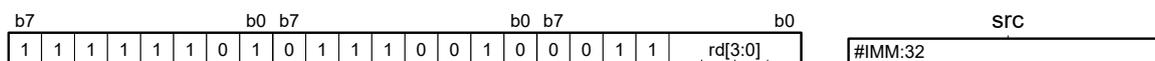
FMUL

FMUL

【コードサイズ】

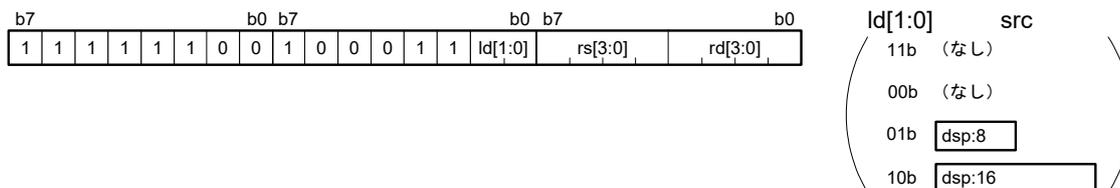
構文	src	src2	dest	コードサイズ (バイト)
(1) FMUL src, dest	#IMM:32	—	Rd	7
(2) FMUL src, dest	Rs	—	Rd	3
	[Rs].L	—	Rd	3
	dsp:8[Rs].L	—	Rd	4
	dsp:16[Rs].L	—	Rd	5
(3) FMUL src, src2, dest	Rs	Rs2	Rd	3

(1) FMUL src, dest



rd[3:0]	dest	
0000b ~ 1111b	Rd	R0(SP) ~ R15

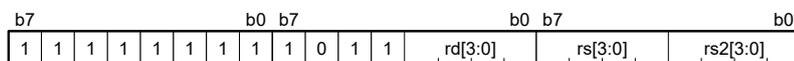
(2) FMUL src, dest



ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

(3) FMUL src, src2, dest



rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ~ 1111b	Rs/Rs2/Rd	R0(SP) ~ R15

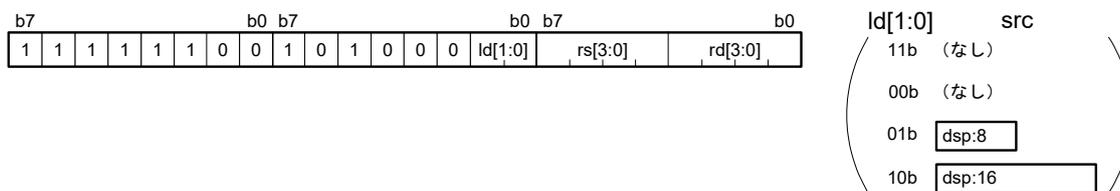
FSQRT

FSQRT

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) FSQRT src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(1) FSQRT src, dest



ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

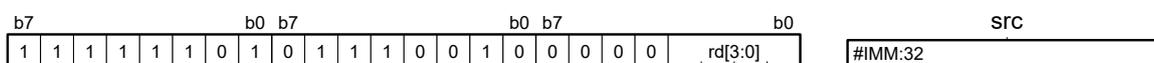
FSUB

FSUB

【コードサイズ】

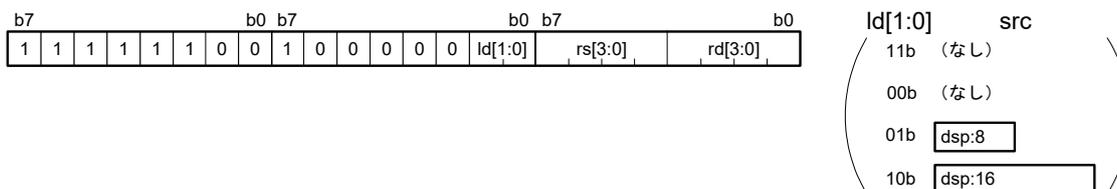
構文	src	src2	dest	コードサイズ (バイト)
(1) FSUB src, dest	#IMM:32	—	Rd	7
(2) FSUB src, dest	Rs	—	Rd	3
	[Rs].L	—	Rd	3
	dsp:8[Rs].L	—	Rd	4
	dsp:16[Rs].L	—	Rd	5
(3) FSUB src, src2, dest	Rs	Rs2	Rd	3

(1) FSUB src, dest



rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

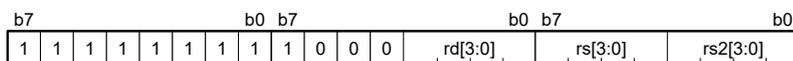
(2) FSUB src, dest



ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

(3) FSUB src, src2, dest



rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest
0000b ~ 1111b	Rs/Rs2/Rd R0(SP) ~ R15

FTOI

FTOI

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) FTOI src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(1) FTOI src, dest



ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

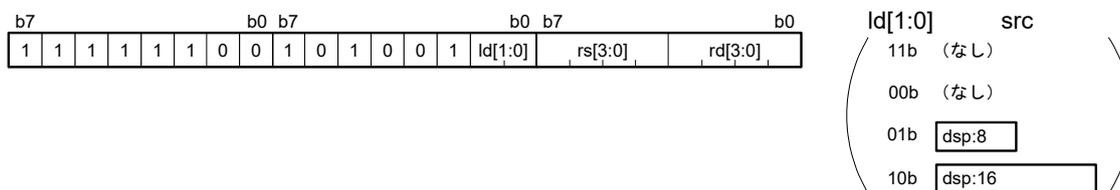
FTOU

FTOU

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) FTOU src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(1) FTOU src, dest



ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

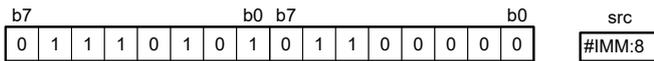
INT

INT

【コードサイズ】

構文	src	コードサイズ (バイト)
(1) INT src	#IMM:8	3

(1) INT src



ITOF

ITOF

【コードサイズ】

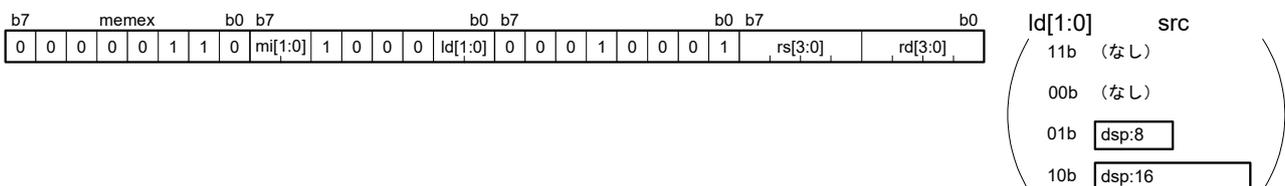
構文	src	dest	コードサイズ (バイト)
(1) ITOF src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == "UB") 4 (memex != "UB")
	dsp:8[Rs].memex	Rd	4 (memex == "UB") 5 (memex != "UB")
	dsp:16[Rs].memex	Rd	5 (memex == "UB") 6 (memex != "UB")

(1) ITOF src, dest

memex == "UB" または src == Rs の場合



memex != "UB" の場合



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

JMP

JMP

【コードサイズ】

構文	src	コードサイズ (バイト)
(1) JMP src	Rs	2

(1) JMP src

b7	b0 b7	b0
0 1 1 1 1 1 1 1	1 0 0 0 0	rs[3:0]

rs[3:0]	Rs	src
0000b ~ 1111b	Rs	R0(SP) ~ R15

JSR

JSR

【コードサイズ】

構文	src	コードサイズ (バイト)
(1) JSR src	Rs	2

(1) JSR src

b7	b0 b7	b0
0 1 1 1 1 1 1 1	1 0 0 0 1	rs[3:0]

rs[3:0]	Rs	src
0000b ~ 1111b	Rs	R0(SP) ~ R15

MACHI

MACHI

【コードサイズ】

構文	src	src2	Adest	コードサイズ (バイト)
(1) MACHI src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MACHI src, src2, Adest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 1 0	1 0 0 0 0 0 a 1 0 0	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b ~ 1111b	Rs/Rs2	R0(SP) ~ R15

MACLH

MACLH

【コードサイズ】

構文	src	src2	Adest	コードサイズ (バイト)
(1) MACLH src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MACLH src, src2, Adest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 1 0	1 0 0 0 0 0 a 1 1 0	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b ~ 1111b	Rs/Rs2	R0(SP) ~ R15

MACLO

MACLO

【コードサイズ】

構文	src	src2	Adest	コードサイズ (バイト)
(1) MACLO src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MACLO src, src2, Adest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 1 0	1 0 0 0 0 0 a 1 0 1	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b ~ 1111b	Rs/Rs2	R0(SP) ~ R15

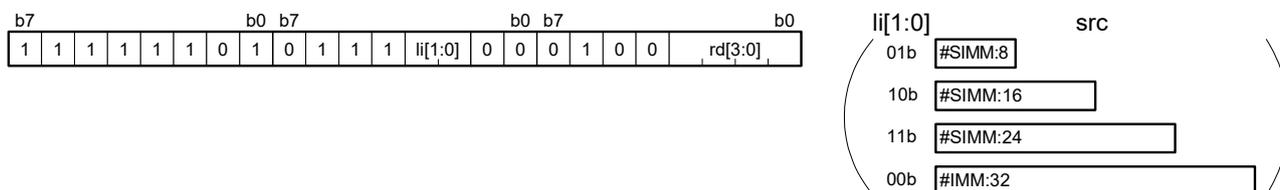
MAX

MAX

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) MAX src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) MAX src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == "UB") 4 (memex != "UB")
	dsp:8[Rs].memex	Rd	4 (memex == "UB") 5 (memex != "UB")
	dsp:16[Rs].memex	Rd	5 (memex == "UB") 6 (memex != "UB")

(1) MAX src, dest

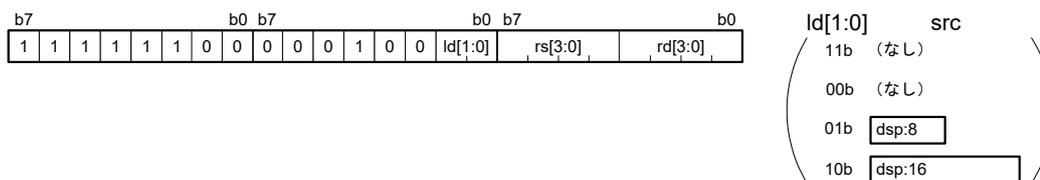


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

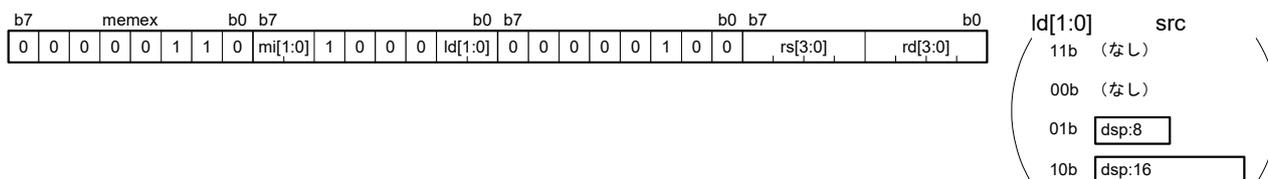
rd[3:0]	dest	
0000b ~ 1111b	Rd	R0(SP) ~ R15

(2) MAX src, dest

memex == "UB" または src == Rs の場合



memex != "UB" の場合



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

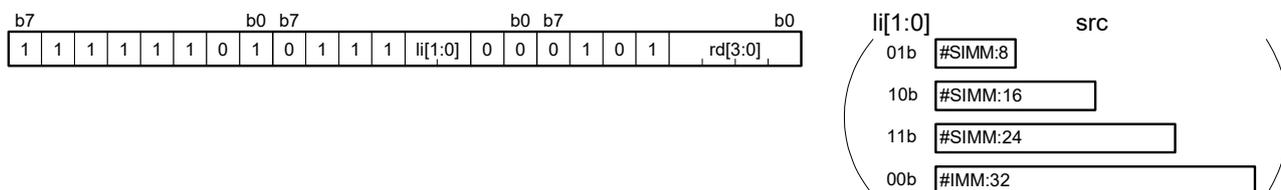
MIN

MIN

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) MIN src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) MIN src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == "UB") 4 (memex != "UB")
	dsp:8[Rs].memex	Rd	4 (memex == "UB") 5 (memex != "UB")
	dsp:16[Rs].memex	Rd	5 (memex == "UB") 6 (memex != "UB")

(1) MIN src, dest

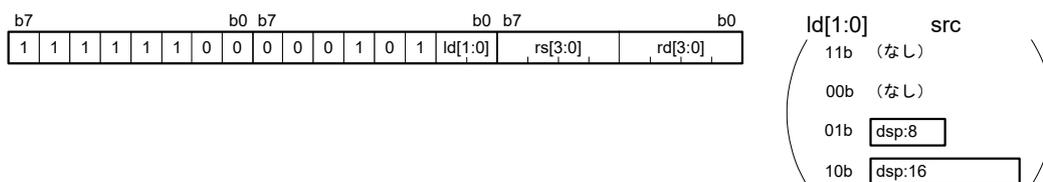


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

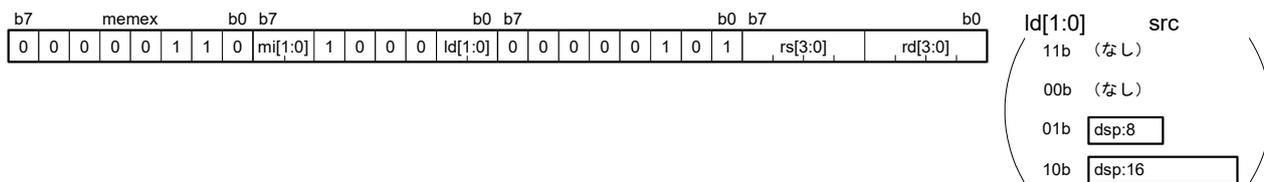
rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

(2) MIN src, dest

memex == "UB" または src == Rs の場合



memex != "UB" の場合



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

MOV

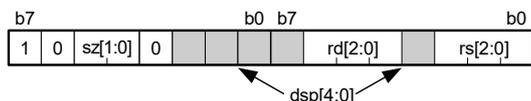
MOV

【コードサイズ】

構文	size	処理サイズ	src	dest	コードサイズ (バイト)
(1) MOV.size src, dest	B/W/L	size	Rs (Rs = R0 ~ R7)	dsp:5[Rd] (Rd = R0 ~ R7)	2
(2) MOV.size src, dest	B/W/L	L	dsp:5[Rs] (Rs = R0 ~ R7)	Rd (Rd = R0 ~ R7)	2
(3) MOV.size src, dest	L	L	#UIMM:4	Rd	2
(4) MOV.size src, dest	B	B	#IMM:8	dsp:5[Rd] (Rd = R0 ~ R7)	3
	W/L	size	#UIMM:8	dsp:5[Rd] (Rd = R0 ~ R7)	3
(5) MOV.size src, dest	L	L	#UIMM:8	Rd	3
(6) MOV.size src, dest	L	L	#SIMM:8	Rd	3
	L	L	#SIMM:16	Rd	4
	L	L	#SIMM:24	Rd	5
	L	L	#IMM:32	Rd	6
(7) MOV.size src, dest	B/W	L	Rs	Rd	2
	L	L	Rs	Rd	2
(8) MOV.size src, dest	B	B	#IMM:8	[Rd]	3
	B	B	#IMM:8	dsp:8[Rd]	4
	B	B	#IMM:8	dsp:16[Rd]	5
	W	W	#SIMM:8	[Rd]	3
	W	W	#SIMM:8	dsp:8[Rd]	4
	W	W	#SIMM:8	dsp:16[Rd]	5
	W	W	#IMM:16	[Rd]	4
	W	W	#IMM:16	dsp:8[Rd]	5
	W	W	#IMM:16	dsp:16[Rd]	6
	L	L	#SIMM:8	[Rd]	3
	L	L	#SIMM:8	dsp:8[Rd]	4
	L	L	#SIMM:8	dsp:16[Rd]	5
	L	L	#SIMM:16	[Rd]	4
	L	L	#SIMM:16	dsp:8[Rd]	5
	L	L	#SIMM:16	dsp:16[Rd]	6
	L	L	#SIMM:24	[Rd]	5
	L	L	#SIMM:24	dsp:8[Rd]	6
	L	L	#SIMM:24	dsp:16[Rd]	7
L	L	#IMM:32	[Rd]	6	
L	L	#IMM:32	dsp:8[Rd]	7	
L	L	#IMM:32	dsp:16[Rd]	8	
(9) MOV.size src, dest	B/W/L	L	[Rs]	Rd	2
	B/W/L	L	dsp:8[Rs]	Rd	3
	B/W/L	L	dsp:16[Rs]	Rd	4
(10) MOV.size src, dest	B/W/L	L	[Ri, Rb]	Rd	3
(11) MOV.size src, dest	B/W/L	size	Rs	[Rd]	2
	B/W/L	size	Rs	dsp:8[Rd]	3
	B/W/L	size	Rs	dsp:16[Rd]	4

構文	size	処理サイズ	src	dest	コードサイズ (バイト)
(12) MOV.size src, dest	B/W/L	size	Rs	[Ri, Rb]	3
(13) MOV.size src, dest	B/W/L	size	[Rs]	[Rd]	2
	B/W/L	size	[Rs]	dsp:8[Rd]	3
	B/W/L	size	[Rs]	dsp:16[Rd]	4
	B/W/L	size	dsp:8[Rs]	[Rd]	3
	B/W/L	size	dsp:8[Rs]	dsp:8[Rd]	4
	B/W/L	size	dsp:8[Rs]	dsp:16[Rd]	5
	B/W/L	size	dsp:16[Rs]	[Rd]	4
	B/W/L	size	dsp:16[Rs]	dsp:8[Rd]	5
(14) MOV.size src, dest	B/W/L	size	Rs	[Rd+]	3
	B/W/L	size	Rs	[-Rd]	3
(15) MOV.size src, dest	B/W/L	L	[Rs+]	Rd	3
	B/W/L	L	[-Rs]	Rd	3

(1) MOV.size src, dest

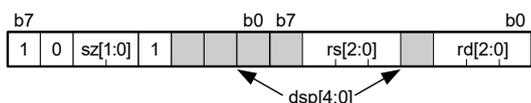


sz[1:0]	size
00b	B
01b	W
10b	L

dsp[4:0]	dsp:5
00000b ~ 11111b	0 ~ 31

rs[2:0]/rd[2:0]	src/dest	
000b ~ 111b	Rs/Rd	R0(SP) ~ R7

(2) MOV.size src, dest

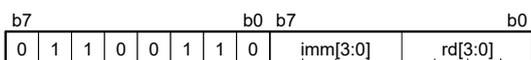


sz[1:0]	size
00b	B
01b	W
10b	L

dsp[4:0]	dsp:5
00000b ~ 11111b	0 ~ 31

rs[2:0]/rd[2:0]	src/dest	
000b ~ 111b	Rs/Rd	R0(SP) ~ R7

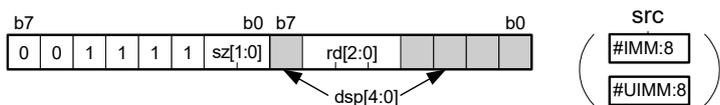
(3) MOV.size src, dest



imm[3:0]	src	
0000b ~ 1111b	#UIMM:4	0 ~ 15

rd[3:0]	dest	
0000b ~ 1111b	Rd	R0(SP) ~ R15

(4) MOV.size src, dest

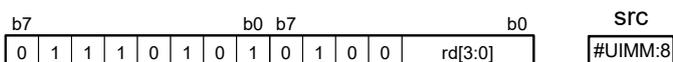


sz[1:0]	size
00b	B
01b	W
10b	L

dsp[4:0]	dsp:5
00000b ~ 11111b	0 ~ 31

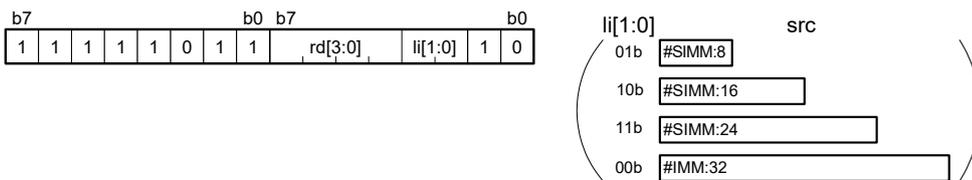
rd[2:0]	dest	
000b ~ 111b	Rd	R0(SP) ~ R7

(5) MOV.size src, dest



rd[3:0]	dest	
0000b ~ 1111b	Rd	R0(SP) ~ R15

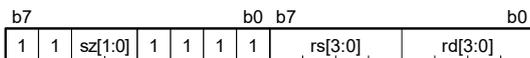
(6) MOV.size src, dest



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b ~ 1111b	Rd	R0(SP) ~ R15

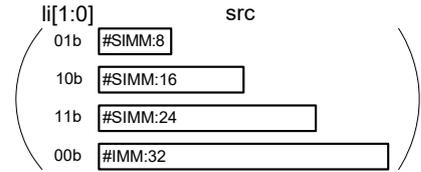
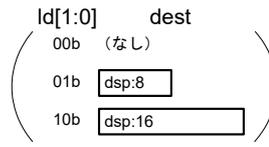
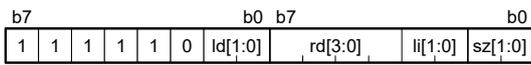
(7) MOV.size src, dest



sz[1:0]	size
00b	B
01b	W
10b	L

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

(8) MOV.size src, dest



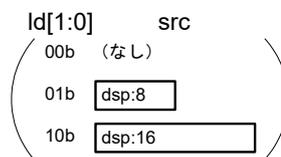
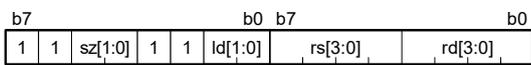
ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

sz[1:0]	size
00b	B
01b	W
10b	L

(9) MOV.size src, dest

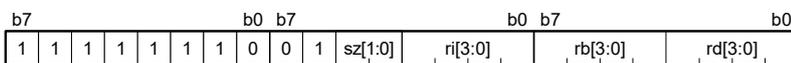


sz[1:0]	size
00b	B
01b	W
10b	L

ld[1:0]	src
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

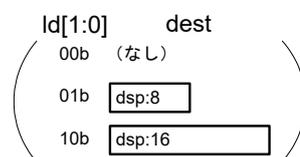
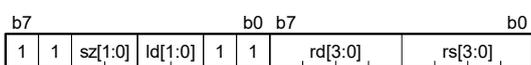
(10) MOV.size src, dest



sz[1:0]	size
00b	B
01b	W
10b	L

ri[3:0]/rb[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Ri/Rb/Rd R0(SP) ~ R15

(11) MOV.size src, dest

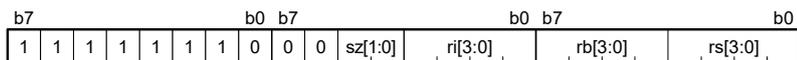


sz[1:0]	size
00b	B
01b	W
10b	L

ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

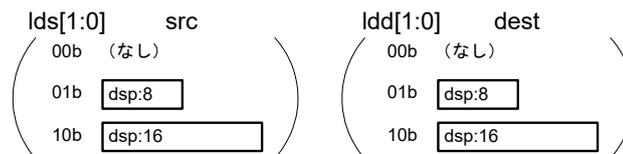
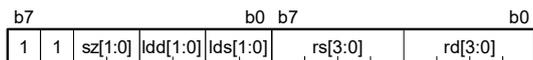
(12) MOV.size src, dest



sz[1:0]	size
00b	B
01b	W
10b	L

rs[3:0]/ri[3:0]/rb[3:0]	src/dest	
0000b ~ 1111b	Rs/Ri/Rb	R0(SP) ~ R15

(13) MOV.size src, dest

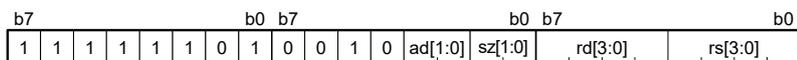


sz[1:0]	size
00b	B
01b	W
10b	L

lds[1:0]/ldd[1:0]	src/dest
00b	[Rs]/[Rd]
01b	dsp:8[Rs]/dsp:8[Rd]
10b	dsp:16[Rs]/dsp:16[Rd]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

(14) MOV.size src, dest

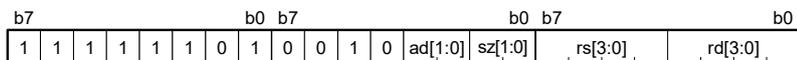


ad[1:0]	addressing
00b	Rs, [Rd+]
01b	Rs, [-Rd]

sz[1:0]	size
00b	B
01b	W
10b	L

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

(15) MOV.size src, dest



ad[1:0]	addressing
10b	[Rs+], Rd
11b	[-Rs], Rd

sz[1:0]	size
00b	B
01b	W
10b	L

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

MOVCO

MOVCO

【コードサイズ】

構文	size	処理サイズ	src	dest	コードサイズ (バイト)
(1) MOVCO src, dest	L	L	Rs	[Rd]	3

(1) MOVCO src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0 1 0	0 1 0 0 1 1 1	rd[3:0]	rs[3:0]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

MOVLI

MOVLI

【コードサイズ】

構文	size	処理サイズ	src	dest	コードサイズ (バイト)
(1) MOVLI src, dest	L	L	[Rs]	Rd	3

(1) MOVLI src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0 1 0	0 1 0 1 1 1 1	rs[3:0]	rd[3:0]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

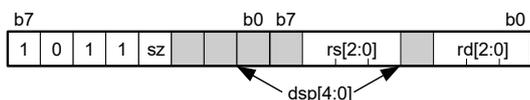
MOVU

MOVU

【コードサイズ】

構文	size	処理サイズ	src	dest	コードサイズ (バイト)
(1) MOVU.size src, dest	B/W	L	dsp:5[Rs] (Rs = R0 ~ R7)	Rd (Rd = R0 ~ R7)	2
(2) MOVU.size src, dest	B/W	L	Rs	Rd	2
	B/W	L	[Rs]	Rd	2
	B/W	L	dsp:8[Rs]	Rd	3
	B/W	L	dsp:16[Rs]	Rd	4
(3) MOVU.size src, dest	B/W	L	[Ri, Rb]	Rd	3
(4) MOVU.size src, dest	B/W	L	[Rs+]	Rd	3
	B/W	L	[-Rs]	Rd	3

(1) MOVU.size src, dest

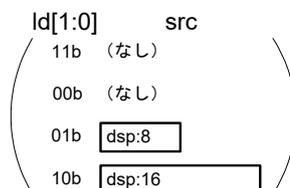
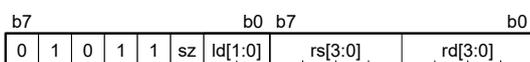


sz	size
0b	B
1b	W

dsp[4:0]	dsp:5
00000b ~ 11111b	0 ~ 31

rs[2:0]/rd[2:0]	src/dest	
000b ~ 111b	Rs/Rd	R0(SP) ~ R7

(2) MOVU.size src, dest



sz	size
0b	B
1b	W

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

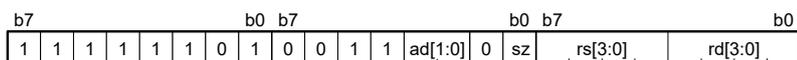
rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

(3) MOVU.size src, dest



sz	size
0b	B
1b	W

ri[3:0]/rb[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Ri/Rb/Rd	R0(SP) ~ R15

(4) MOVU.size src, dest

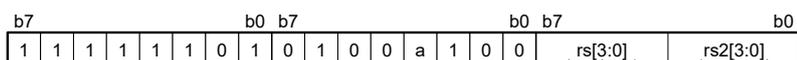
ad[1:0]	addressing
10b	[Rs+], Rd
11b	[-Rs], Rd

sz	size
0b	B
1b	W

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

MSBHI**MSBHI****【コードサイズ】**

構文	src	src2	Adest	コードサイズ (バイト)
(1) MSBHI src, src2, Adest	Rs	Rs2	A0, A1	3

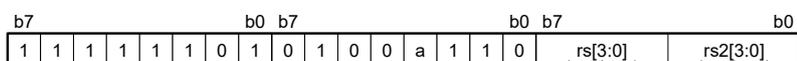
(1) MSBHI src, src2, Adest

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b ~ 1111b	Rs/Rs2	R0(SP) ~ R15

MSBLH**MSBLH****【コードサイズ】**

構文	src	src2	Adest	コードサイズ (バイト)
(1) MSBLH src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MSBLH src, src2, Adest

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b ~ 1111b	Rs/Rs2	R0(SP) ~ R15

MSBLO

MSBLO

【コードサイズ】

構文	src	src2	Adest	コードサイズ (バイト)
(1) MSBLO src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MSBLO src, src2, Adest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0	1 0 1 0 0 a 1 0 1	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b ~ 1111b	Rs/Rs2	R0(SP) ~ R15

MUL

MUL

【コードサイズ】

構文	src	src2	dest	コードサイズ (バイト)
(1) MUL src, dest	#UIMM:4	—	Rd	2
(2) MUL src, dest	#SIMM:8	—	Rd	3
	#SIMM:16	—	Rd	4
	#SIMM:24	—	Rd	5
	#IMM:32	—	Rd	6
(3) MUL src, dest	Rs	—	Rd	2
	[Rs].memex	—	Rd	2 (memex == "UB") 3 (memex != "UB")
	dsp:8[Rs].memex	—	Rd	3 (memex == "UB") 4 (memex != "UB")
	dsp:16[Rs].memex	—	Rd	4 (memex == "UB") 5 (memex != "UB")
(4) MUL src, src2, dest	Rs	Rs2	Rd	3

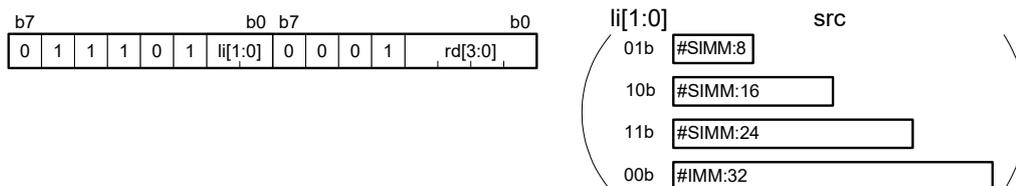
(1) MUL src, dest

b7	b0 b7	b0
0 1 1 0 0 0 1 1	imm[3:0]	rd[3:0]

imm[3:0]	src	
0000b ~ 1111b	#UIMM:4	0 ~ 15

rd[3:0]	dest	
0000b ~ 1111b	Rd	R0(SP) ~ R15

(2) MUL src, dest



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

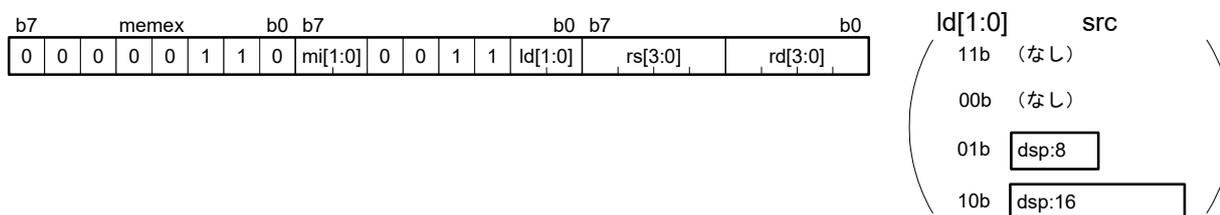
rd[3:0]	dest	
0000b ~ 1111b	Rd	R0(SP) ~ R15

(3) MUL src, dest

memex == “UB” または src == Rs の場合



memex != “UB” の場合

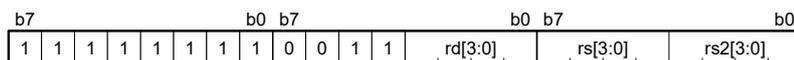


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

(4) MUL src, src2, dest



rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ~ 1111b	Rs/Rs2/Rd	R0(SP) ~ R15

MULHI

MULHI

【コードサイズ】

構文	src	src2	Adest	コードサイズ (バイト)
(1) MULHI src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MULHI src, src2, Adest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0	1 0 0 0 0 a 0 0 0	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b ~ 1111b	Rs/Rs2	R0(SP) ~ R15

MULLH

MULLH

【コードサイズ】

構文	src	src2	Adest	コードサイズ (バイト)
(1) MULLH src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MULLH src, src2, Adest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0	1 0 0 0 0 0 a 0 1 0	rs[3:0]	rs2[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b ~ 1111b	Rs/Rs2	R0(SP) ~ R15

MULLO

MULLO

【コードサイズ】

構文	src	src2	Adest	コードサイズ (バイト)
(1) MULLO src, src2, Adest	Rs	Rs2	A0, A1	3

(1) MULLO src, src2, Adest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0 1 0 0 0 0 a 0 0 1	rs[3:0]	rs2[3:0]	

a	Adest
0b	A0
1b	A1

rs[3:0]/rs2[3:0]	src/src2	
0000b ~ 1111b	Rs/Rs2	R0(SP) ~ R15

MVFACGU

MVFACGU

【コードサイズ】

構文	src	Asrc	dest	コードサイズ (バイト)
(1) MVFACGU src, Asrc, dest	#IMM:2	A0, A1	Rd	3

(1) MVFACGU src, Asrc, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0 1 0 0 0 1 1 1 1	imm[1]	a imm[0]	1 1 rd[3:0]

a	Asrc
0b	A0
1b	A1

imm[1:0]	src	
00	#IMM:2	2
01		—
10		0
11		1

dest	
Rd	R0(SP) ~ R15

MVFACHI

MVFACHI

【コードサイズ】

構文	src	Asrc	dest	コードサイズ (バイト)
(1) MVFACHI src, Asrc, dest	#IMM:2	A0, A1	Rd	3

(1) MVFACHI src, Asrc, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 1 0	1 0 0 0 1 1 1 1	imm[1] a imm[0] 0 0	rd[3:0]

a	Asrc
0b	A0
1b	A1

imm[1:0]	src
00	#IMM:2 2
01	—
10	0
11	1

dest	
Rd	R0(SP) ~ R15

MVFACLO

MVFACLO

【コードサイズ】

構文	src	Asrc	dest	コードサイズ (バイト)
(1) MVFACLO src, Asrc, dest	#IMM:2	A0, A1	Rd	3

(1) MVFACLO src, Asrc, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 1 0	1 0 0 0 1 1 1 1	imm[1] a imm[0] 0 1	rd[3:0]

a	Asrc
0b	A0
1b	A1

imm[1:0]	src
00	#IMM:2 2
01	—
10	0
11	1

dest	
Rd	R0(SP) ~ R15

MVFACMI

MVFACMI

【コードサイズ】

構文	src	Asrc	dest	コードサイズ (バイト)
(1) MVFACMI src, Asrc, dest	#IMM:2	A0, A1	Rd	3

(1) MVFACMI src, Asrc, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 1 0	1 0 0 0 1 1 1 1	imm[1] a imm[0] 1 0	rd[3:0]

a	Asrc
0b	A0
1b	A1

imm[1:0]	src
00	#IMM:2 2
01	—
10	0
11	1

dest	
Rd	R0(SP) ~ R15

MVFC

MVFC

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) MVFC src, dest	Rx	Rd	3

(1) MVFC src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 1 0	1 0 1 1 0 1 0 1	0 cr[3:0]	rd[3:0]

cr[3:0]	src
0000b	Rx PSW
0001b	PC
0010b	USP
0011b	FPSW
0100b	(予約)
0101b	(予約)
0110b	(予約)
0111b	(予約)
1000b	BPSW
1001b	BPC
1010b	ISP
1011b	FINTV
1100b	INTB
1101b	EXTB
1110b ~ 1111b	(予約)

rd[3:0]	dest	
0000b ~ 1111b	Rd	R0(SP) ~ R15

MVTACGU

MVTACGU

【コードサイズ】

構文	src	Adest	コードサイズ (バイト)
(1) MVTACGU src, Adest	Rs	A0, A1	3

(1) MVTACGU src, Adest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 1 0	1 0 0 0 1 0 1 1 1 1	a	0 1 1 , rs[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]	src
0000b ~ 1111b	Rs R0(SP) ~ R15

MVTACHI

MVTACHI

【コードサイズ】

構文	src	Adest	コードサイズ (バイト)
(1) MVTACHI src, Adest	Rs	A0, A1	3

(1) MVTACHI src, Adest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 1 0	1 0 0 0 1 0 1 1 1 1	a	0 0 0 , rs[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]	src
0000b ~ 1111b	Rs R0(SP) ~ R15

MVTACLO

MVTACLO

【コードサイズ】

構文	src	Adest	コードサイズ (バイト)
(1) MVTACLO src, Adest	Rs	A0, A1	3

(1) MVTACLO src, Adest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 1 0	1 0 0 0 1 0 1 1 1 1	a	0 0 1 , rs[3:0]

a	Adest
0b	A0
1b	A1

rs[3:0]	src
0000b ~ 1111b	Rs R0(SP) ~ R15

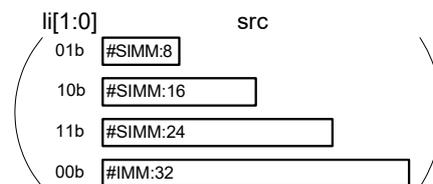
MVTC

MVTC

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) MVTC src, dest	#SIMM:8	Rx	4
	#SIMM:16	Rx	5
	#SIMM:24	Rx	6
	#IMM:32	Rx	7
(2) MVTC src, dest	Rs	Rx	3

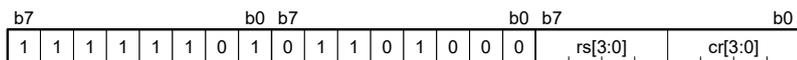
(1) MVTC src, dest



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

cr[3:0]	dest	
0000b	Rx	PSW
0001b		(予約)
0010b		USP
0011b		FPSW
0100b		(予約)
0101b		(予約)
0110b		(予約)
0111b		(予約)
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b		EXTB
1110b ~ 1111b		(予約)

(2) MVTC src, dest



cr[3:0]	dest	
0000b	Rx	PSW
0001b		(予約)
0010b		USP
0011b		FPSW
0100b		(予約)
0101b		(予約)
0110b		(予約)
0111b		(予約)
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b		EXTB
1110b ~ 1111b		(予約)

rs[3:0]	src	
0000b ~ 1111b	Rs	R0(SP) ~ R15

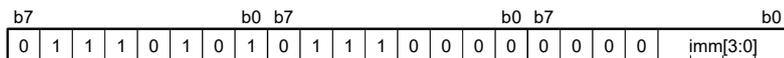
MVTIPL

MVTIPL

【コードサイズ】

構文	src	コードサイズ (バイト)
(1) MVTIPL src	#IMM:4	3

(1) MVTIPL src



imm[3:0]	#IMM:4
0000b ~ 1111b	0 ~ 15

NEG

NEG

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) NEG dest	—	Rd	2
(2) NEG src, dest	Rs	Rd	3

(1) NEG dest

b7	b0 b7	b0
0 1 1 1 1 1 1 0 0 0 0 1	rd[3:0]	

rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

(2) NEG src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1	rs[3:0]	rd[3:0]	

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

NOP

NOP

【コードサイズ】

構文	コードサイズ (バイト)
(1) NOP	1

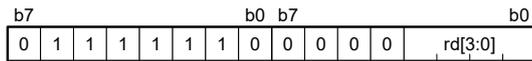
(1) NOP

b7	b0
0 0 0 0 0 0 1 1	

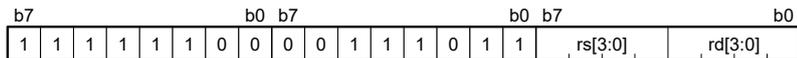
NOT**NOT**

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) NOT dest	—	Rd	2
(2) NOT src, dest	Rs	Rd	3

(1) NOT dest

rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

(2) NOT src, dest

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

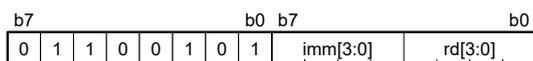
OR

OR

【コードサイズ】

構文	src	src2	dest	コードサイズ (バイト)
(1) OR src, dest	#UIMM:4	—	Rd	2
(2) OR src, dest	#SIMM:8	—	Rd	3
	#SIMM:16	—	Rd	4
	#SIMM:24	—	Rd	5
	#IMM:32	—	Rd	6
(3) OR src, dest	Rs	—	Rd	2
	[Rs].memex	—	Rd	2 (memex == "UB") 3 (memex != "UB")
	dsp:8[Rs].memex	—	Rd	3 (memex == "UB") 4 (memex != "UB")
	dsp:16[Rs].memex	—	Rd	4 (memex == "UB") 5 (memex != "UB")
(4) OR src, src2, dest	Rs	Rs2	Rd	3

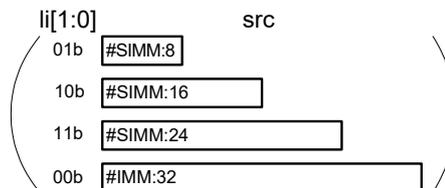
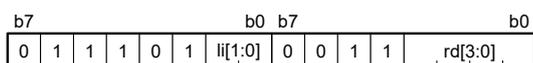
(1) OR src, dest



imm[3:0]	src	
0000b ~ 1111b	#UIMM:4	0 ~ 15

rd[3:0]	dest	
0000b ~ 1111b	Rd	R0(SP) ~ R15

(2) OR src, dest



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b ~ 1111b	Rd	R0(SP) ~ R15

POPC

POPC

【コードサイズ】

構文	dest	コードサイズ (バイト)
(1) POPC dest	Rx	2

(1) POPC dest

b7	b0 b7	b0
0 1 1 1 1 1 1 1	0 1 1 1 0	cr[3:0]

cr[3:0]	dest	
0000b	Rx	PSW
0001b		(予約)
0010b		USP
0011b		FPSW
0100b		(予約)
0101b		(予約)
0110b		(予約)
0111b		(予約)
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b		EXTB
1110b ~ 1111b		(予約)

POPM

POPM

【コードサイズ】

構文	dest	dest2	コードサイズ (バイト)
(1) POPM dest-dest2	Rd	Rd2	2

(1) POPM dest-dest2

b7	b0 b7	b0
0 1 1 0 1 1 1 1	rd[3:0]	rd2[3:0]

rd[3:0]	dest	
0001b ~ 1110b	Rd	R1 ~ R14

rd2[3:0]	dest2	
0010b ~ 1111b	Rd2	R2 ~ R15

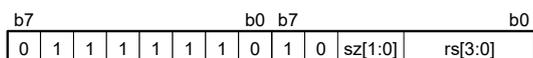
PUSH

PUSH

【コードサイズ】

構文	src	コードサイズ (バイト)
(1) PUSH.size src	Rs	2
(2) PUSH.size src	[Rs]	2
	dsp:8[Rs]	3
	dsp:16[Rs]	4

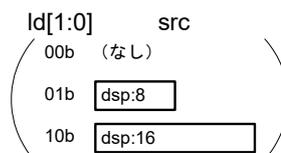
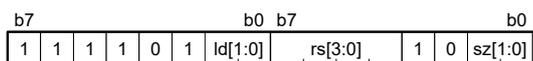
(1) PUSH.size src



sz[1:0]	size
00b	B
01b	W
10b	L

rs[3:0]	src	
0000b ~ 1111b	Rs	R0(SP) ~ R15

(2) PUSH.size src



ld[1:0]	src
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]	src	
0000b ~ 1111b	Rs	R0(SP) ~ R15

sz[1:0]	size
00b	B
01b	W
10b	L

PUSHC

PUSHC

【コードサイズ】

構文	src	コードサイズ (バイト)
(1) PUSHC src	Rx	2

(1) PUSHC src

b7	b0 b7	b0
0 1 1 1 1 1 1 1	0 1 1 0 0	cr[3:0]

cr[3:0]	src	
0000b	Rx	PSW
0001b		PC
0010b		USP
0011b		FPSW
0100b		(予約)
0101b		(予約)
0110b		(予約)
0111b		(予約)
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b		EXTB
1110b ~ 1111b		(予約)

PUSHM

PUSHM

【コードサイズ】

構文	src	src2	コードサイズ (バイト)
(1) PUSHM src-src2	Rs	Rs2	2

(1) PUSHM src-src2

b7	b0 b7	b0
0 1 1 0 1 1 1 0	rs[3:0]	rs2[3:0]

rs[3:0]	src	
0001b ~ 1110b	Rs	R1 ~ R14

rs2[3:0]	src2	
0010b ~ 1111b	Rs2	R2 ~ R15

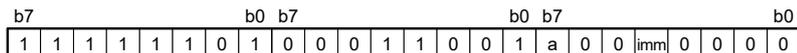
RACL

RACL

【コードサイズ】

構文	src	Adest	コードサイズ (バイト)
(1) RACL src, Adest	#IMM:1 (IMM:1 = 1, 2)	A0, A1	3

(1) RACL src, Adest



a	Adest
0b	A0
1b	A1

imm	src
0b, 1b	#IMM:1 1, 2

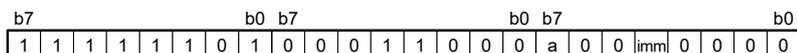
RACW

RACW

【コードサイズ】

構文	src	Adest	コードサイズ (バイト)
(1) RACW src, Adest	#IMM:1	A0, A1	3

(1) RACW src, Adest



a	Adest
0b	A0
1b	A1

imm	src
0b, 1b	#IMM:1 1, 2

RDACL

RDACL

【コードサイズ】

構文	src	Adest	コードサイズ (バイト)
(1) RDACL src, Adest	#IMM:1 (IMM:1 = 1, 2)	A0, A1	3

(1) RDACL src, Adest

b7		b0 b7		b0 b7		b0																			
1	1	1	1	1	1	0	1	0	0	0	0	1	1	0	0	0	1	a	1	0	imm	0	0	0	0

a	Adest
0b	A0
1b	A1

imm	src
0b, 1b	#IMM:1 1, 2

RDACW

RDACW

【コードサイズ】

構文	src	Adest	コードサイズ (バイト)
(1) RDACW src, Adest	#IMM:1 (IMM:1 = 1, 2)	A0, A1	3

(1) RDACW src, Adest

b7		b0 b7		b0 b7		b0																		
1	1	1	1	1	1	0	1	0	0	0	0	1	1	0	0	0	a	1	0	imm	0	0	0	0

a	Adest
0b	A0
1b	A1

imm	src
0b, 1b	#IMM:1 1, 2

REVL

REVL

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) REVL src, dest	Rs	Rd	3

(1) REVL src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0 1	0 1 1 0 0 1 1 1	rs[3:0]	rd[3:0]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

REVV

REVV

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) REVW src, dest	Rs	Rd	3

(1) REVW src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0 1	0 1 1 0 0 1 0 1	rs[3:0]	rd[3:0]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

RMPA

RMPA

【コードサイズ】

構文	size	コードサイズ (バイト)
(1) RMPA.size	B	2
	W	2
	L	2

(1) RMPA.size

b7	b0 b7	b0
0 1 1 1 1 1 1 1	1 1 0 0 0 1 1	sz[1:0]

sz[1:0]	size
00b	B
01b	W
10b	L

ROLC

ROLC

【コードサイズ】

構文	dest	コードサイズ (バイト)
(1) ROLC src, Adest	Rd	2

(1) ROLC dest

b7	b0 b7	b0
0 1 1 1 1 1 1 1	0 0 1 0 1	rd[3:0]

rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

RORC

RORC

【コードサイズ】

構文	dest	コードサイズ (バイト)
(1) RORC dest	Rd	2

(1) RORC dest

b7	b0 b7	b0
0 1 1 1 1 1 1 1	0 0 1 0 0	rd[3:0]

rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

ROTL

ROTL

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) ROTL src, dest	#IMM:5	Rd	3
(2) ROTL src, dest	Rs	Rd	3

(1) ROTL src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0	1 0 1 1 0 1 1 1	imm[4:0]	rd[3:0]

imm[4:0]	src		rd[3:0]	dest	
00000b ~ 11111b	#IMM:5	0 ~ 31	0000b ~ 1111b	Rd	R0(SP) ~ R15

(2) ROTL src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0	1 0 1 1 0 0 1 1 0	rs[3:0]	rd[3:0]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

ROTR

ROTR

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) ROTR src, dest	#IMM:5	Rd	3
(2) ROTR src, dest	Rs	Rd	3

(1) ROTR src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0	1 0 1 1 0 1 1 0	imm[4:0]	rd[3:0]

imm[4:0]	src		rd[3:0]	dest	
00000b ~ 11111b	#IMM:5	0 ~ 31	0000b ~ 1111b	Rd	R0(SP) ~ R15

(2) ROTR src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0	1 0 1 1 0 0 1 0 0	rs[3:0]	rd[3:0]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

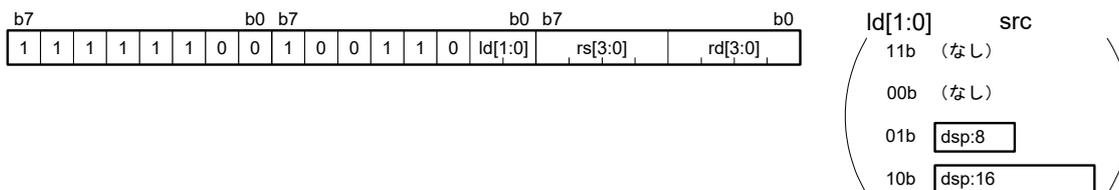
ROUND

ROUND

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) ROUND src, dest	Rs	Rd	3
	[Rs].L	Rd	3
	dsp:8[Rs].L	Rd	4
	dsp:16[Rs].L	Rd	5

(1) ROUND src, dest



ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

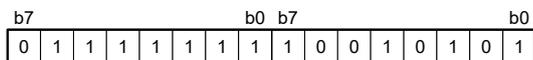
RTE

RTE

【コードサイズ】

構文	コードサイズ (バイト)
(1) RTE	2

(1) RTE



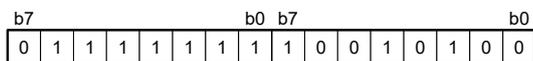
RTFI

RTFI

【コードサイズ】

構文	コードサイズ (バイト)
(1) RTFI	2

(1) RTFI



RTS

RTS

【コードサイズ】

構文	コードサイズ (バイト)
(1) RTS	1

(1) RTS

b7	b0
0 0 0 0 0 0 1 0	

RTSD

RTSD

【コードサイズ】

構文	src	dest	dest2	コードサイズ (バイト)
(1) RTSD src	#UIMM:8	—	—	2
(2) RTSD src, dest-dest2	#UIMM:8	Rd	Rd2	3

(1) RTSD src

b7	b0	SRC
0 1 1 0 0 1 1 1		#UIMM:8

(2) RTSD src, dest-dest2

b7	b0 b7	b0	SRC
0 0 1 1 1 1 1 1	rd[3:0]	rd2[3:0]	#UIMM:8

rd[3:0]/rd2[3:0]	dest/dest2	
0001b ~ 1111b	Rd/Rd2	R1 ~ R15

SAT

SAT

【コードサイズ】

構文	dest	コードサイズ (バイト)
(1) SAT dest	Rd	2

(1) SAT dest

b7	b0 b7	b0
0 1 1 1 1 1 1 1	0 0 0 1 1	rd[3:0]

rd[3:0]	dest	
0000b ~ 1111b	Rd	R0(SP) ~ R15

SATR

SATR

【コードサイズ】

構文	コードサイズ (バイト)
(1) SATR	2

(1) SATR

b7	b0 b7	b0
0 1 1 1 1 1 1 1	1 1 0 0 1 0 0 1	1 1

SBB

SBB

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) SBB src, dest	Rs	Rd	3
(2) SBB src, dest	[Rs].L	Rd	4
	dsp:8[Rs].L	Rd	5
	dsp:16[Rs].L	Rd	6

(1) SBB src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 1 0	0 0 0 0 0 0 0 0	ld[1:0]	rs[3:0]
		rd[3:0]	

ld[1:0]	src
11b	Rs

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

(2) SBB src, dest

b7	memex	b0 b7	b0 b7	b0 b7	b0
0 0 0 0 0 0 1 1	0 1	0 1 0 1 0 0 0 0	ld[1:0]	0 0 0 0 0 0 0 0	rs[3:0]
			rd[3:0]		

ld[1:0]	src
00b (なし)	
01b	dsp:8
10b	dsp:16

ld[1:0]	src
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

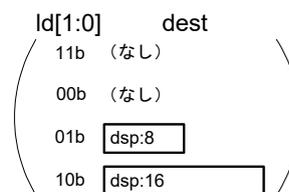
SCCnd

SCCnd

【コードサイズ】

構文	size	dest	コードサイズ (バイト)
(1) SCCnd.size dest	L	Rd	3
	B/W/L	[Rd]	3
	B/W/L	dsp:8[Rd]	4
	B/W/L	dsp:16[Rd]	5

(1) SCCnd.size dest



sz[1:0]	size
00b	B
01b	W
10b	L

ld[1:0]	dest
11b	Rd
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rd[3:0]	dest	
0000b ~ 1111b	Rd	R0(SP) ~ R15

cd[3:0]	SCCnd	cd[3:0]	SCCnd
0000b	SCEQ, SCZ	1000b	SCGE
0001b	SCNE, SCNZ	1001b	SCLT
0010b	SCGEU, SCC	1010b	SCGT
0011b	SCLTU, SCNC	1011b	SCLE
0100b	SCGTU	1100b	SCO
0101b	SCLEU	1101b	SCNO
0110b	SCPZ	1110b	(予約)
0111b	SCN	1111b	(予約)

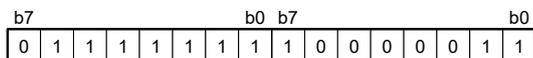
SCMPU

SCMPU

【コードサイズ】

構文	コードサイズ (バイト)
(1) SCMPU	2

(1) SCMPU



SETPSW

SETPSW

【コードサイズ】

構文	dest	コードサイズ (バイト)
(1) SETPSW dest	flag	2

(1) SETPSW dest

b7	b0 b7	b0
0 1 1 1 1 1 1 1	1 1 0 1 0	cb[3:0]

cb[3:0]	dest	
0000b	flag	C
0001b		Z
0010b		S
0011b		O
0100b		(予約)
0101b		(予約)
0110b		(予約)
0111b		(予約)
1000b		I
1001b		U
1010b		(予約)
1011b		(予約)
1100b		(予約)
1101b		(予約)
1110b		(予約)
1111b		(予約)

SHAR

SHAR

【コードサイズ】

構文	src	src2	dest	コードサイズ (バイト)
(1) SHAR src, dest	#IMM:5	—	Rd	2
(2) SHAR src, dest	Rs	—	Rd	3
(3) SHAR src, src2, dest	#IMM:5	Rs	Rd	3

(1) SHAR src, dest

b7	b0 b7	b0
0 1 1 0 1 0 1	imm[4:0]	rd[3:0]

imm[4:0]	src
00000b ~ 11111b	#IMM:5 0 ~ 31

rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

(2) SHAR src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0 1 0 1 1 0 0 0 0 0 1	rs[3:0]	rd[3:0]	

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

(3) SHAR src, src2, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0 1 1 0 1	imm[4:0]	rs2[3:0]	rd[3:0]

imm[4:0]	src
00000b ~ 11111b	#IMM:5 0 ~ 31

rs2[3:0]/rd[3:0]	src2/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

SHLL

SHLL

【コードサイズ】

構文	src	src2	dest	コードサイズ (バイト)
(1) SHLL src, dest	#IMM:5	—	Rd	2
(2) SHLL src, dest	Rs	—	Rd	3
(3) SHLL src, src2, dest	#IMM:5	Rs	Rd	3

(1) SHLL src, dest

b7	b0 b7	b0
0 1 1 0 1 1 0	imm[4:0]	rd[3:0]

imm[4:0]	src
00000b ~ 11111b	#IMM:5 0 ~ 31

rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

(2) SHLL src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0 1 0 1 1 0 0 0 1 0	rs[3:0]	rd[3:0]	

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

(3) SHLL src, src2, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0 1 1 1 0	imm[4:0]	rs2[3:0]	rd[3:0]

imm[4:0]	src
00000b ~ 11111b	#IMM:5 0 ~ 31

rs2[3:0]/rd[3:0]	src2/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

SHLR

SHLR

【コードサイズ】

構文	src	src2	dest	コードサイズ (バイト)
(1) SHLR src, dest	#IMM:5	—	Rd	2
(2) SHLR src, dest	Rs	—	Rd	3
(3) SHLR src, src2, dest	#IMM:5	Rs	Rd	3

(1) SHLR src, dest

b7	b0 b7	b0
0 1 1 0 1 0 0	imm[4:0]	rd[3:0]

imm[4:0]	src
00000b ~ 11111b	#IMM:5 0 ~ 31

rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

(2) SHLR src, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0 1 0 1 1 0 0 0 0 0	rs[3:0]	rd[3:0]	

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

(3) SHLR src, src2, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0 1 1 0 0	imm[4:0]	rs2[3:0]	rd[3:0]

imm[4:0]	src
00000b ~ 11111b	#IMM:5 0 ~ 31

rs2[3:0]/rd[3:0]	src2/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

SMOVB

SMOVB

【コードサイズ】

構文	コードサイズ (バイト)
(1) SMOVB	2

(1) SMOVB

b7	b0 b7	b0
0 1 1 1 1 1 1 1 1 0 0 0 1 0 1 1		

SMOVF

SMOVF

【コードサイズ】

構文	コードサイズ (バイト)
(1) SMOVF	2

(1) SMOVF

b7	b0 b7	b0
0 1 1 1 1 1 1 1	1 1 0 0 0 1 1 1	1

SMOVU

SMOVU

【コードサイズ】

構文	コードサイズ (バイト)
(1) SMOVU	2

(1) SMOVU

b7	b0 b7	b0
0 1 1 1 1 1 1 1	1 1 0 0 0 0 1 1	1

SSTR

SSTR

【コードサイズ】

構文	size	処理サイズ	コードサイズ (バイト)
(1) SSTR.size	B	B	2
	W	W	2
	L	L	2

(1) SSTR.size

b7	b0 b7	b0
0 1 1 1 1 1 1 1	1 1 0 0 0 1 0	sz[1:0]

sz[1:0]	size
00b	B
01b	W
10b	L

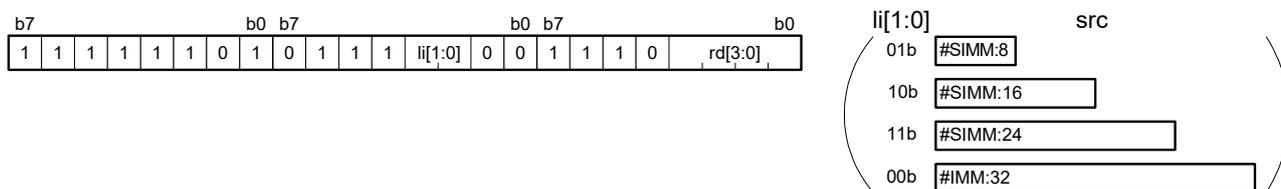
STZ

STZ

【コードサイズ】

構文	src	dest	コードサイズ (バイト)
(1) STZ src, dest	#SIMM:8	Rd	4
	#SIMM:16	Rd	5
	#SIMM:24	Rd	6
	#IMM:32	Rd	7
(2) STZ src, dest	Rs	Rd	3

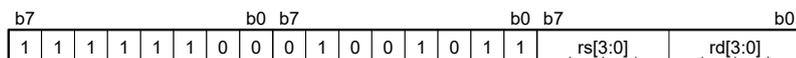
(1) STZ src, dest



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b ~ 1111b	Rd	R0(SP) ~ R15

(2) STZ src, dest



rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

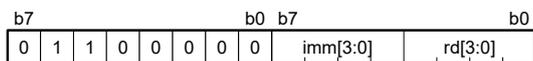
SUB

SUB

【コードサイズ】

構文	src	src2	dest	コードサイズ (バイト)
(1) SUB src, dest	#UIMM:4	—	Rd	2
(2) SUB src, dest	Rs	—	Rd	2
	[Rs].memex	—	Rd	2 (memex == "UB") 3 (memex != "UB")
	dsp:8[Rs].memex	—	Rd	3 (memex == "UB") 4 (memex != "UB")
	dsp:16[Rs].memex	—	Rd	4 (memex == "UB") 5 (memex != "UB")
(3) SUB src, src2, dest	Rs	Rs2	Rd	3

(1) SUB src, dest

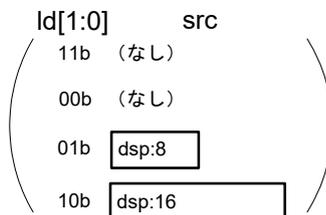
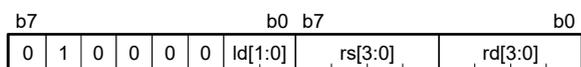


imm[3:0]	src
0000b ~ 1111b	#UIMM:4 0 ~ 15

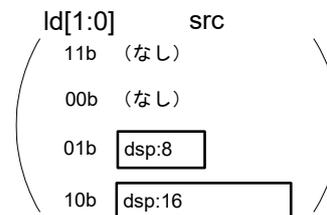
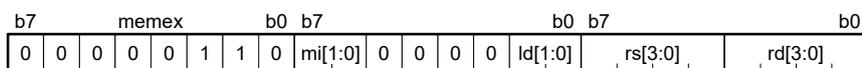
rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

(2) SUB src, dest

memex == "UB" または src == Rs の場合



memex != "UB" の場合



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

(3) SUB src, src2, dest

b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 1 1 1 0 0 0 0	rd[3:0]	rs[3:0]	rs2[3:0]

rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ~ 1111b	Rs/Rs2/Rd	R0(SP) ~ R15

SUNTIL**SUNTIL****【コードサイズ】**

構文	size	処理サイズ	コードサイズ (バイト)
(1) SUNTIL.size	B	B	2
	W	W	2
	L	L	2

(1) SUNTIL.size

b7	b0 b7	b0
0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0	sz[1:0]	

sz[1:0]	size
00b	B
01b	W
10b	L

SWHILE**SWHILE****【コードサイズ】**

構文	size	処理サイズ	コードサイズ (バイト)
(1) SWHILE.size	B	B	2
	W	W	2
	L	L	2

(1) SWHILE.size

b7	b0 b7	b0
0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1	sz[1:0]	

sz[1:0]	size
00b	B
01b	W
10b	L

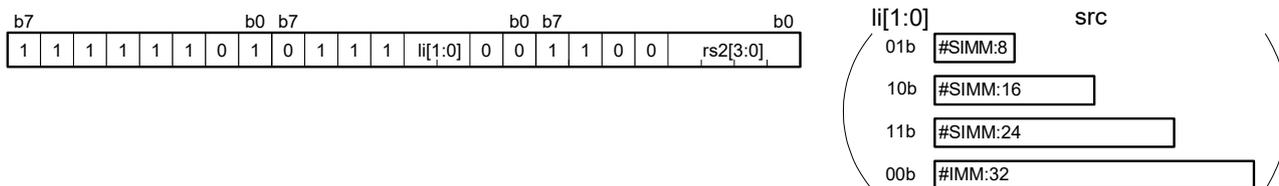
TST

TST

【コードサイズ】

構文	src	src2	コードサイズ (バイト)
(1) TST src, src2	#SIMM:8	Rs	4
	#SIMM:16	Rs	5
	#SIMM:24	Rs	6
	#IMM:32	Rs	7
(2) TST src, src2	Rs	Rs2	3
	[Rs].memex	Rs2	3 (memex == "UB") 4 (memex != "UB")
	dsp:8[Rs].memex	Rs2	4 (memex == "UB") 5 (memex != "UB")
	dsp:16[Rs].memex	Rs2	5 (memex == "UB") 6 (memex != "UB")

(1) TST src, src2

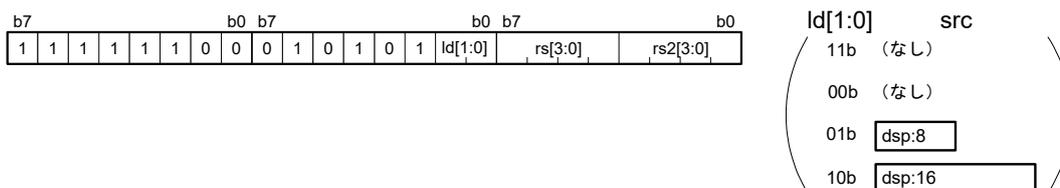


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

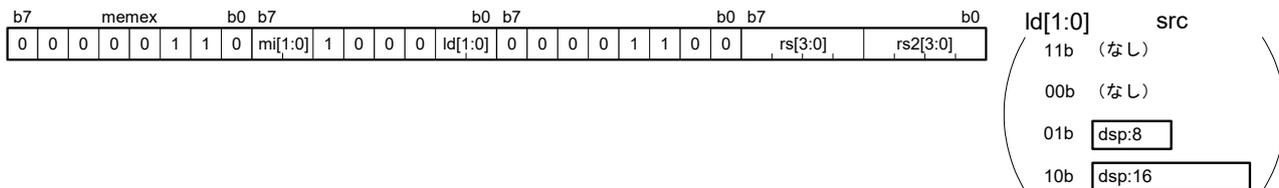
rs2[3:0]	src2	
0000b ~ 1111b	Rs	R0(SP) ~ R15

(2) TST src, src2

memex == "UB" または src == Rs の場合



memex != "UB" の場合



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rs2[3:0]	src/src2	
0000b ~ 1111b	Rs/Rs2	R0(SP) ~ R15

UTOF

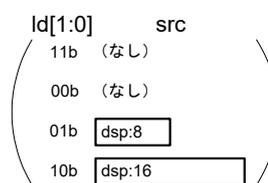
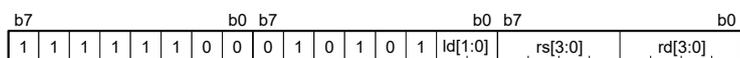
UTOF

【コードサイズ】

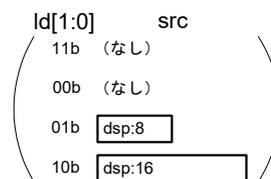
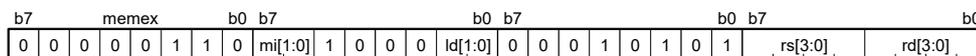
構文	src	dest	コードサイズ (バイト)
(1) UTOF src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == "UB") 4 (memex != "UB")
	dsp:8[Rs].memex	Rd	4 (memex == "UB") 5 (memex != "UB")
	dsp:16[Rs].memex	Rd	5 (memex == "UB") 6 (memex != "UB")

(1) UTOF src, dest

memex == "UB" または src == Rs の場合



memex != "UB" の場合



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

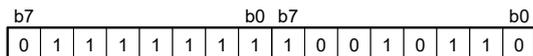
WAIT

WAIT

【コードサイズ】

構文	コードサイズ (バイト)
(1) WAIT	2

(1) WAIT



XCHG

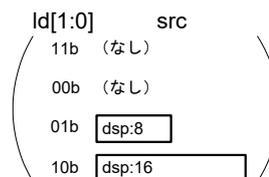
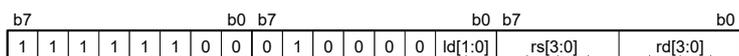
XCHG

【コードサイズ】

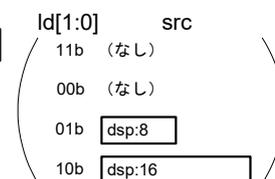
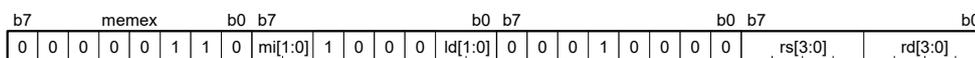
構文	src	dest	コードサイズ (バイト)
(1) XCHG src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == "UB") 4 (memex != "UB")
	dsp:8[Rs].memex	Rd	4 (memex == "UB") 5 (memex != "UB")
	dsp:16[Rs].memex	Rd	5 (memex == "UB") 6 (memex != "UB")

(1) XCHG src, dest

memex == "UB" または src == Rs の場合



memex != "UB" の場合



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	Rs/Rd	R0(SP) ~ R15

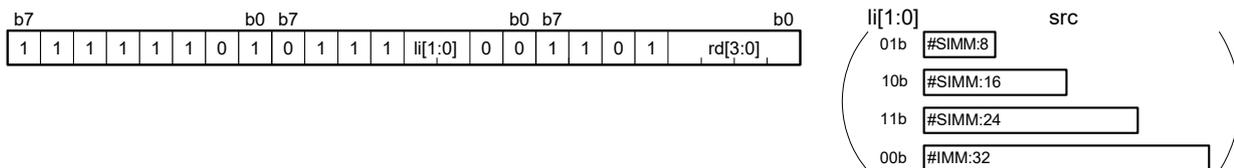
XOR

XOR

【コードサイズ】

構文	src	src2	dest	コードサイズ (バイト)
(1) XOR src, dest	#SIMM:8	—	Rd	4
	#SIMM:16	—	Rd	5
	#SIMM:24	—	Rd	6
	#IMM:32	—	Rd	7
(2) XOR src, dest	Rs	—	Rd	3
	[Rs].memex	—	Rd	3 (memex == "UB") 4 (memex != "UB")
	dsp:8[Rs].memex	—	Rd	4 (memex == "UB") 5 (memex != "UB")
	dsp:16[Rs].memex	—	Rd	5 (memex == "UB") 6 (memex != "UB")
(3) XOR src, src2, dest	Rs	Rs2	Rd	3

(1) XOR src, dest



li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

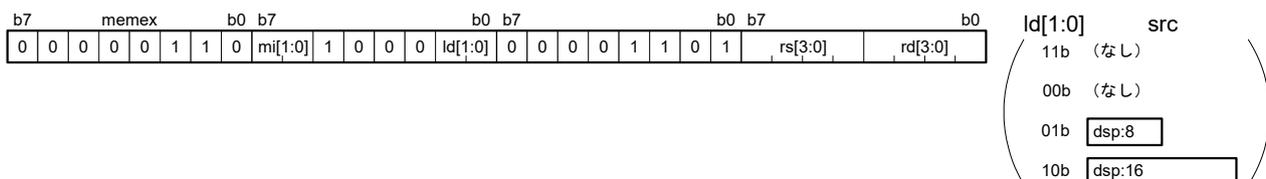
rd[3:0]	dest
0000b ~ 1111b	Rd R0(SP) ~ R15

(2) XOR src, dest

memex == "UB" または src == Rs の場合



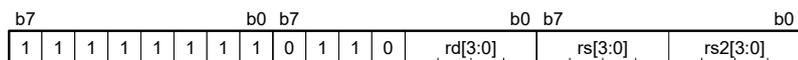
memex != "UB" の場合



mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest
0000b ~ 1111b	Rs/Rd R0(SP) ~ R15

(3) XOR src, src2, dest

rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ~ 1111b	Rs/Rs2/Rd	R0(SP) ~ R15

4.2.2 レジスタ括退避機能命令

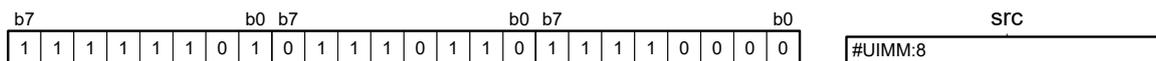
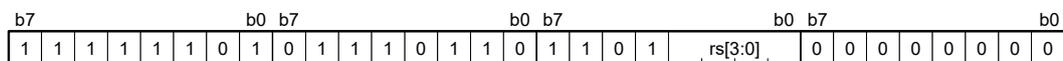
次ページよりレジスタ括退避機能命令の命令コードの詳細説明を示します。

RSTR

RSTR

【コードサイズ】

構文	src	コードサイズ (バイト)
(1) RSTR src	#UIMM:8	4
(2) RSTR src	Rs	4

(1) RSTR src**(2) RSTR src**

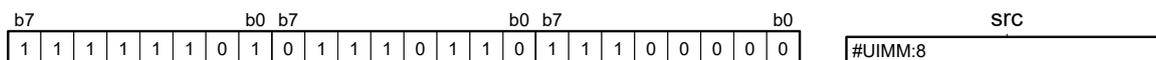
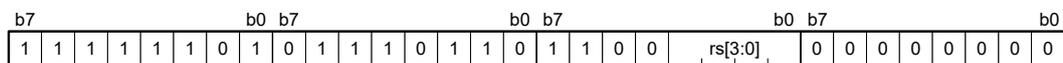
rs[3:0]	src
0000b ~ 1111b	Rs R0(SP) ~ R15

SAVE

SAVE

【コードサイズ】

構文	src	コードサイズ (バイト)
(1) SAVE src	#UIMM:8	4
(2) SAVE src	Rs	4

(1) SAVE src**(2) SAVE src**

rs[3:0]	src
0000b ~ 1111b	Rs R0(SP) ~ R15

4.2.3 倍精度浮動小数点処理命令

次ページより倍精度浮動小数点処理命令の命令コードの詳細説明を示します。

DABS

DABS

【コードサイズ】

構文	対象		コードサイズ (バイト)
	src	dest	
DABS src, dest	DRs	DRd	4

b7	b0 b7	b0 b7	b0 b7	b0
0 1 1 1 0 1 1 0 1 0 0 1 0 0 0 0	rs[3:0]	1 1 0 0	rd[3:0]	0 0 0 1

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	DRs/DRd	DR0 ~ DR15

DADD

DADD

【コードサイズ】

構文	対象			コードサイズ (バイト)
	src	src2	dest	
DADD src, src2, dest	DRs	DRs2	DRd	4

b7	b0 b7	b0 b7	b0 b7	b0
0 1 1 1 0 1 1 0 1 0 0 1 0 0 0 0	rs2[3:0]	0 0 0 0	rd[3:0]	rs[3:0]

rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ~ 1111b	DRs/DRs2/DRd	DR0 ~ DR15

DCMPcm

DCMPcm

【コードサイズ】

構文	対象		コードサイズ (バイト)
	src	src2	
DCMPcm src, src2	DRs	DRs2	4

b7	b0 b7	b0 b7	b0 b7	b0
0 1 1 1 0 1 1 0 1 0 0 1 0 0 0 0	rs2[3:0]	1 0 0 0	cm[3:0]	rs[3:0]

rs[3:0]/rs2[3:0]	src/src2	
0000b ~ 1111b	DRs/DRs2	DR0 ~ DR15

cm	条件
0001b	UN
0010b	EQ
0100b	LT
0110b	LE

DDIV

DDIV

【コードサイズ】

構文	対象			コードサイズ (バイト)
	src	src2	dest	
DDIV src, src2, dest	DRs	DRs2	DRd	4

b7	b0 b7	b0 b7	b0 b7	b0
0 1 1 1 0 1 1 0 1 0 0 1 0 0 0 0	rs2[3:0]	0 1 0 1	rd[3:0]	rs[3:0]

rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ~ 1111b	DRs/DRs2/DRd	DR0 ~ DR15

DMOV

DMOV

【コードサイズ】

構文	フォーマット	size	処理サイズ	対象		コードサイズ (バイト)
				src	dest	
DMOV.size src, dest	レジスタ間転送					
	(1)	D	D	Rs	DRHd	4
	(2)	L	L	Rs	DRHd	4
	(3)	L	L	Rs	DRLd	4
	(4)	L	L	DRHs	Rd	4
	(5)	L	L	DRLs	Rd	4
	(6)	D	D	DRs	DRd	4
	ストア					
	(7)	D	D	DRs	[Rd]	4
		D	D	DRs	dsp:8[Rd] (注1)	5
		D	D	DRs	dsp:16[Rd] (注1)	6
	ロード					
	(8)	D	D	[Rs]	DRd	4
		D	D	dsp:8[Rs] (注1)	DRd	5
		D	D	dsp:16[Rs] (注1)	DRd	6
即値						
(9)	D	D	#IMM:32	DRHd	7	
(10)	L	L	#IMM:32	DRHd	7	
(11)	L	L	#IMM:32	DRLd	7	

注1. 弊社の「RXファミリアセンブラ」では、ディスプレイースメントの値 (dsp:8、dsp:16) は、サイズ指定子が“.D”のときは8の倍数を指定してください。dsp:8には、サイズ指定子が“.D”のとき0~2040 (255×8) が指定できます。dsp:16には、サイズ指定子が“.D”のとき0~524280 (65535×8) が指定できます。命令コードには、1/8した値が埋め込まれます。

(1) DMOV.size src, dest

b7	b0	b7	b0	b7	b0																				
1	1	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	0	0	0	rs[3:0]	rd[3:0]	0	0	1	1

rs[3:0]	src	
0000b~1111b	Rs	R0(SP)~R15

rd[3:0]	dest	
0000b~1111b	DRHd	DRH0~DRH15

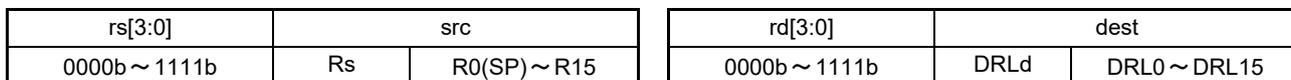
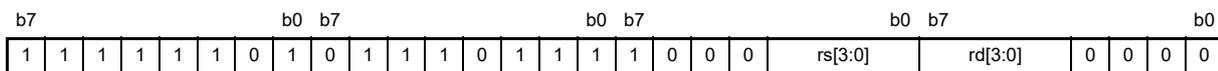
(2) DMOV.size src, dest

b7	b0	b7	b0	b7	b0																				
1	1	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	0	0	0	rs[3:0]	rd[3:0]	0	0	1	0

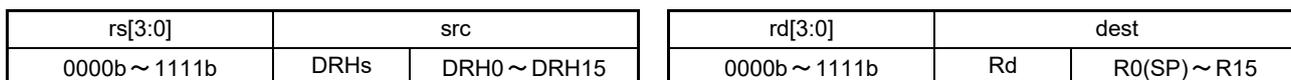
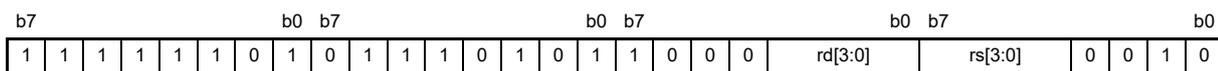
rs[3:0]	src	
0000b~1111b	Rs	R0(SP)~R15

rd[3:0]	dest	
0000b~1111b	DRHd	DRH0~DRH15

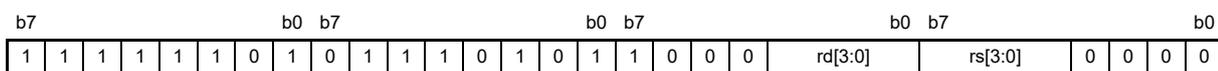
(3) DMOV.size src, dest



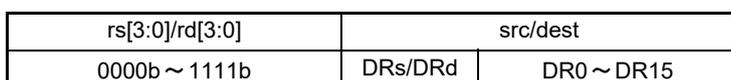
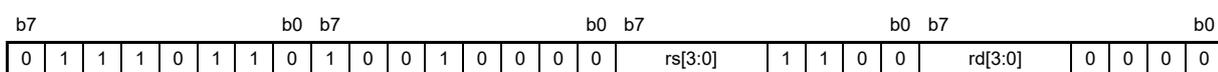
(4) DMOV.size src, dest



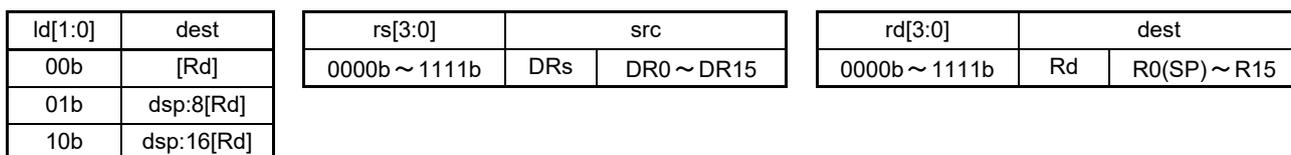
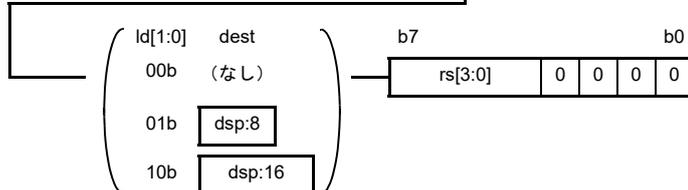
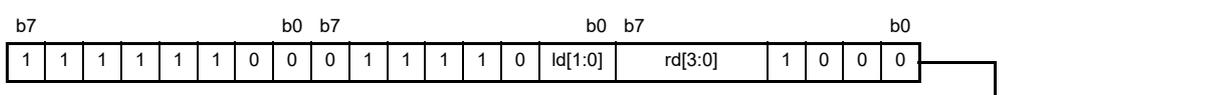
(5) DMOV.size src, dest



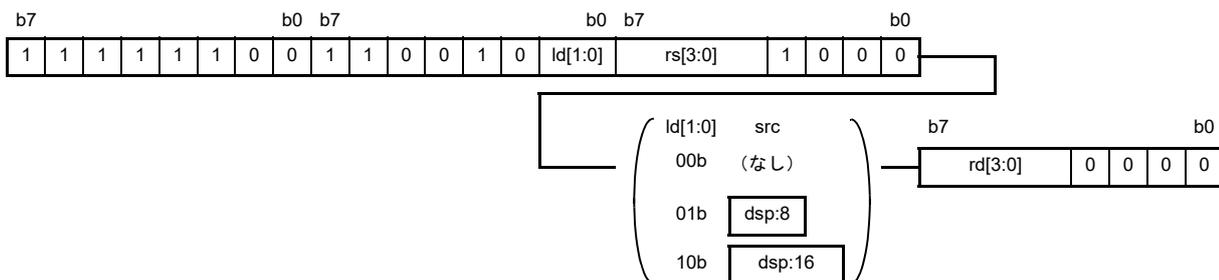
(6) DMOV.size src, dest



(7) DMOV.size src, dest

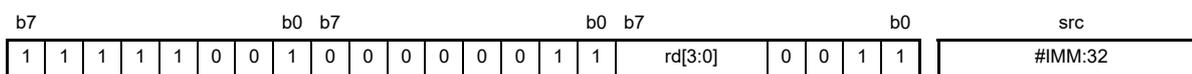


(8) DMOV.size src, dest



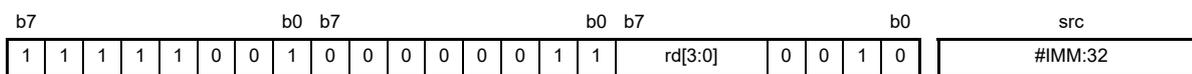
ld[1:0]	src	rs[3:0]	src	rd[3:0]	dest
00b	[Rs]	0000b ~ 1111b	Rs R0(SP) ~ R15	0000b ~ 1111b	Rd DR0 ~ DR15
01b	dsp:8[Rs]				
10b	dsp:16[Rs]				

(9) DMOV.size src, dest



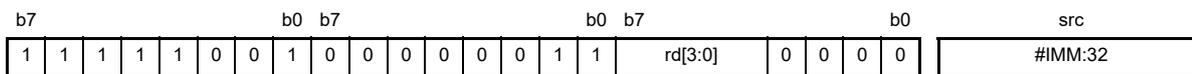
rd[3:0]	dest
0000b ~ 1111b	DRHd DRH0 ~ DRH15

(10) DMOV.size src, dest



rd[3:0]	dest
0000b ~ 1111b	DRHd DRH0 ~ DRH15

(11) DMOV.size src, dest



rd[3:0]	dest
0000b ~ 1111b	DRLd DRL0 ~ DRL15

DMUL

DMUL

【コードサイズ】

構文	対象			コードサイズ (バイト)
	src	src2	dest	
DMUL src, src2, dest	DRs	DRs2	DRd	4

b7	b0 b7	b0 b7	b0 b7	b0																		
0	1	1	1	0	1	1	0	1	0	0	1	0	0	0	0	rs[3:0]	0	0	1	0	rd[3:0]	rs[3:0]

rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ~ 1111b	DRs/DRs2/DRd	DR0 ~ DR15

DNEG

DNEG

【コードサイズ】

構文	対象		コードサイズ (バイト)
	dest	dest	
DNEG src, dest	DRd	DRd	4

b7	b0 b7	b0 b7	b0 b7	b0																					
0	1	1	1	0	1	1	0	1	0	0	1	0	0	0	0	rs[3:0]	1	1	0	0	rd[3:0]	0	0	1	0

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	DRs/DRd	DR0 ~ DR15

DPOPM

DPOPM

【コードサイズ】

構文	フォーマット	処理サイズ	対象		コードサイズ (バイト)
			dest	dest2	
DPOPM.size dest-dest2	(1)	D	DRd	DRd2	3
	(2)	L	DCRd	DCRd2	3

(1) DPOPM.size dest-dest2

b7	b0	b7	b0	b7	b0												
0	1	1	1	0	1	0	1	1	0	1	1	1	0	0	0	rd[3:0]	nm[3:0]

rd[3:0]	dest	
0000b ~ 1111b	DRd	DR0 ~ DR15

nm[3:0]	num(dest2 - dest)
0000b ~ 1111b	d2 - d (注1)

注1. d2 - d はDRd - DRd2 間にある (DRd, DRd2 含む) すべてのレジスタの個数 - 1 を示します。

(2) DPOPM.size dest-dest2

b7	b0	b7	b0	b7	b0												
0	1	1	1	0	1	0	1	1	0	1	0	1	0	0	0	rd[3:0]	nm[3:0]

rd[3:0]	dest	
0000b	DCRd	DCR0/DPSW
0001b		DCR1/DCMR
0010b		DCR2/DECNT
0011b		DCR3/DEPC

nm[3:0]	num(dest2 - dest)
0000b ~ 0011b	d2 - d (注1)

注1. d2 - d はDCRd - DCRd2 間にある (DCRd, DCRd2 含む) すべてのレジスタの個数 - 1 を示します。

DPUSHM

DPUSHM

【コードサイズ】

構文	フォーマット	処理サイズ	対象		コードサイズ (バイト)
			src	src2	
DPUSHM.size src-src2	(1)	D	DRs	DRs2	3
	(2)	L	DCRs	DCRs2	3

(1) DPUSHM.size src-src2

b7	b0	b7	b0	b7	b0												
0	1	1	1	0	1	0	1	1	0	1	1	0	0	0	0	rs[3:0]	nm[3:0]

rs[3:0]	src	
0000b~1111b	DRs	DR0~DR15

nm[3:0]	num(src2 - src)
0000b~1111b	s2 - s (注1)

注1. s2 - s はDRs - DRs2 間にある (DRs, DRs2含む) すべてのレジスタの個数 - 1 を示します。

(2) DPUSHM.size src-src2

b7	b0	b7	b0	b7	b0												
0	1	1	1	0	1	0	1	1	0	1	0	0	0	0	0	rs[3:0]	nm[3:0]

rs[3:0]	src	
0000b	DCRs	DCR0/DPSW
0001b		DCR1/DCMR
0010b		DCR2/DECNT
0011b		DCR3/DEPC

nm[3:0]	num(src2 - src)
0000b~0011b	s2 - s (注1)

注1. s2 - s はDCRs - DCRs2 間にある (DCRs, DCRs2含む) すべてのレジスタの個数 - 1 を示します。

DROUND

DROUND

【コードサイズ】

構文	対象		コードサイズ (バイト)
	src	dest	
DROUND src, dest	DRs	DRd	4

b7	b0 b7	b0 b7	b0 b7	b0																					
0	1	1	1	0	1	1	0	1	0	1	0	0	0	0	0	rs[3:0]	1	1	0	1	rd[3:0]	1	1	0	1

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	DRs/DRd	DR0 ~ DR15

DSQRT

DSQRT

【コードサイズ】

構文	対象		コードサイズ (バイト)
	src	dest	
DSQRT src, dest	DRs	DRd	4

b7	b0 b7	b0 b7	b0 b7	b0																					
0	1	1	1	0	1	1	0	1	0	0	1	0	0	0	0	rs[3:0]	1	1	0	1	rd[3:0]	0	0	0	0

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	DRs/DRd	DR0 ~ DR15

DSUB

DSUB

【コードサイズ】

構文	対象			コードサイズ (バイト)
	src	src2	dest	
DSUB src, src2, dest	DRs	DRs2	DRd	4

b7	b0 b7	b0 b7	b0 b7	b0																		
0	1	1	1	0	1	1	0	1	0	0	1	0	0	0	0	rs2[3:0]	0	0	0	1	rd[3:0]	rs[3:0]

rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ~ 1111b	DRs/DRs2/DRd	DR0 ~ DR15

DTOF**DTOF**

【コードサイズ】

構文	対象		コードサイズ (バイト)
	src	dest	
DTOF src, dest	DRs	DRd	4

b7	b0 b7	b0 b7	b0 b7	b0
0 1 1 1 0 1 1 0 1 0 0 1 0 0 0 0	rs[3:0]	1 1 0 1	rd[3:0]	1 1 0 0

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	DRs/DRd	DR0 ~ DR15

DTOI**DTOI**

【コードサイズ】

構文	対象		コードサイズ (バイト)
	src	dest	
DTOI src, dest	DRs	DRd	4

b7	b0 b7	b0 b7	b0 b7	b0
0 1 1 1 0 1 1 0 1 0 0 1 0 0 0 0	rs[3:0]	1 1 0 1	rd[3:0]	1 0 0 0

rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	DRs/DRd	DR0 ~ DR15

DTOU**DTOU**

【コードサイズ】

構文	対象		コードサイズ (バイト)
	src	dest	
DTOU src, dest	DRs	DRd	4

b7	b0 b7	b0 b7	b0 b7	b0
0 1 1 1 0 1 1 0 1 0 0 1 0 0 0 0	rs[3:0]	1 1 0 1	rd[3:0]	1 0 0 1

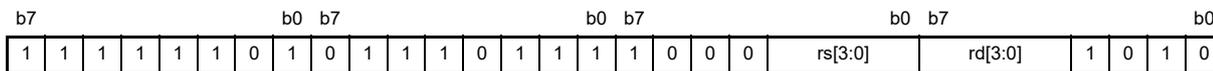
rs[3:0]/rd[3:0]	src/dest	
0000b ~ 1111b	DRs/DRd	DR0 ~ DR15

FTOD

FTOD

【コードサイズ】

構文	対象		コードサイズ (バイト)
	src	dest	
FTOD src, dest	Rs	DRd	4



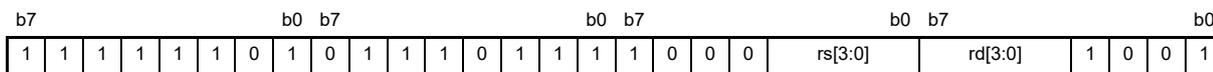
rs[3:0]	src		rd[3:0]	dest	
0000b ~ 1111b	Rs	R0(SP) ~ R15	0000b ~ 1111b	DRd	DR0 ~ DR15

ITOD

ITOD

【コードサイズ】

構文	対象		コードサイズ (バイト)
	src	dest	
ITOD src, dest	Rs	DRd	4



rs[3:0]	src		rd[3:0]	dest	
0000b ~ 1111b	Rs	R0(SP) ~ R15	0000b ~ 1111b	DRd	DR0 ~ DR15

MVFDC

MVFDC

【コードサイズ】

構文	対象		コードサイズ (バイト)
	src	dest	
MVFDC src, dest	DCRs	Rd	4

b7	b0 b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0	1 0 1 1 1 0 1 0	1 1 0 0 0	rd[3:0]	rs[3:0]
			0 1 0 0	

rs[3:0]	src	rd[3:0]	dest	
0000b	DCRs	0000b ~ 1111b	Rd	
0001b				DCR0/DPSW
0010b				DCR1/DCMR
0011b				DCR2/DECNT
			R0(SP) ~ R15	
0011b	DCR3/DEPC			

MVFDR

MVFDR

【コードサイズ】

構文	コードサイズ (バイト)
MVFDR	3

b7	b0 b7	b0 b7	b0
0 1 1 1 0 1 0	1 1 0 0 1 0 0 0	0 0 0 0 0 0 1 1	0 1 1
			1 1

MVTDC

MVTDC

【コードサイズ】

構文	対象		コードサイズ (バイト)
	src	dest	
MVTDC src, dest	Rs	DCRd	4

b7	b0 b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0	1 0 1 1 1 0 1 1 1 1 0 0 0	rs[3:0]	rd[3:0]	0 1 0 0

rs[3:0]	src	
0000b~1111b	Rs	R0(SP)~R15

rd[3:0]	dest	
0000b	DCRd	DCR0/DPSW
0001b		DCR1/DCMR
0010b		DCR2/DECNT
0011b		DCR3/DEPC

UTOD

UTOD

【コードサイズ】

構文	対象		コードサイズ (バイト)
	src	dest	
UTOD src, dest	Rs	DRd	4

b7	b0 b7	b0 b7	b0 b7	b0
1 1 1 1 1 1 0	1 0 1 1 1 0 1 1 1 1 0 0 0	rs[3:0]	rd[3:0]	1 1 0 1

rs[3:0]	src	
0000b~1111b	Rs	R0(SP)~R15

rd[3:0]	dest	
0000b~1111b	DRd	DR0~DR15

5. 例外処理

5.1 例外事象

CPU が通常プログラムを実行している途中で、ある事象の発生によってそのプログラムの実行を中断し、別のプログラムを実行する必要がある場合があります。このような事象を総称して例外事象と呼びます。

図 5.1 に例外事象の種類を示します。

例外が発生すると、プロセッサモードはスーパーバイザモードになります。

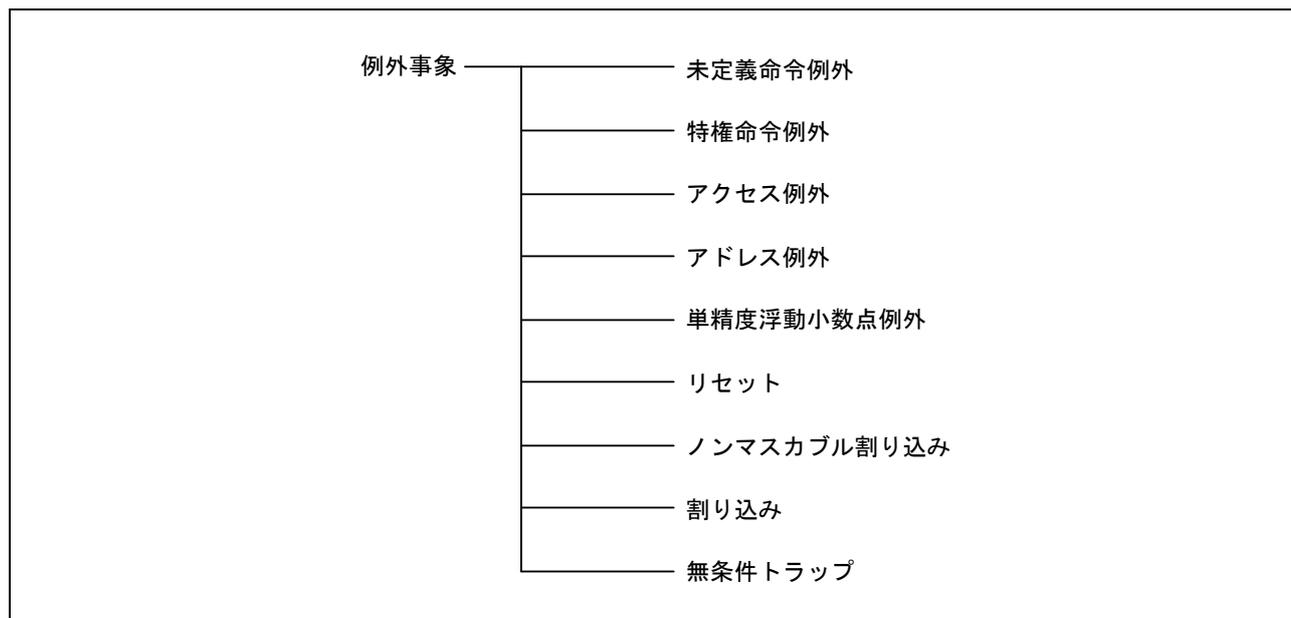


図 5.1 例外事象の種類

5.1.1 未定義命令例外

未定義命令例外は、未定義命令（実装されていない命令）の実行を検出した場合に発生します。

5.1.2 特権命令例外

特権命令例外は、ユーザモードで特権命令の実行を検出した場合に発生します。特権命令はスーパーバイザモードでのみ実行可能です。

5.1.3 アクセス例外

アクセス例外は、CPUからのメモリアクセスによるエラーが検出された場合に発生します。メモリプロテクションユニットによるメモリプロテクションエラーが検出された場合、命令アクセス例外とオペランドアクセス例外が発生します。

5.1.4 アドレス例外

アドレス例外は、4の倍数ではないアドレスに対して64ビットのオペランドアクセスを実行した場合に発生します。

5.1.5 単精度浮動小数点例外

単精度浮動小数点例外は、単精度浮動小数点演算命令実行時に、IEEE754規格で規定された5つの例外（オーバフロー、アンダフロー、精度異常、ゼロ除算、無効演算）および、非実装処理を検出した場合に発生します。5つの例外については、対応するFPSWのEX、EU、EZ、EO、EVビットが“1”のときのみCPUの例外処理が行われます。

5.1.6 リセット

CPUにリセット信号を入力することによって発生します。リセットは最高度の優先順位を持ち、常に受け付けられます。

5.1.7 ノンマスカブル割り込み

CPUにノンマスカブル割り込み信号を入力することによって発生します。システムに致命的な障害が発生したと考えられる場合のみ使用します。例外処理ルーチン処理後、例外発生時に実行していた元のプログラムに復帰しない条件で使用してください。

5.1.8 割り込み

CPUに割り込み信号を入力することによって発生します。割り込みのうち1つの要因を、高速割り込みとして割り当てることが可能です。高速割り込みは、通常の割り込みに比べ、ハードウェア前処理とハードウェア後処理が高速です。高速割り込みの優先レベルは15（最高）です。

PSWのIビットが“0”のとき、割り込みの受け付けは禁止されます。

5.1.9 無条件トラップ

INT命令、およびBRK命令を実行すると無条件トラップが発生します。

5.2 例外の処理手順

例外処理には、ハードウェアが自動的に処理する部分と、ユーザが記述したプログラム（例外処理ルーチン）によって処理される部分があります。リセットを除く、例外受け付け時の処理手順を図 5.2 に示します。

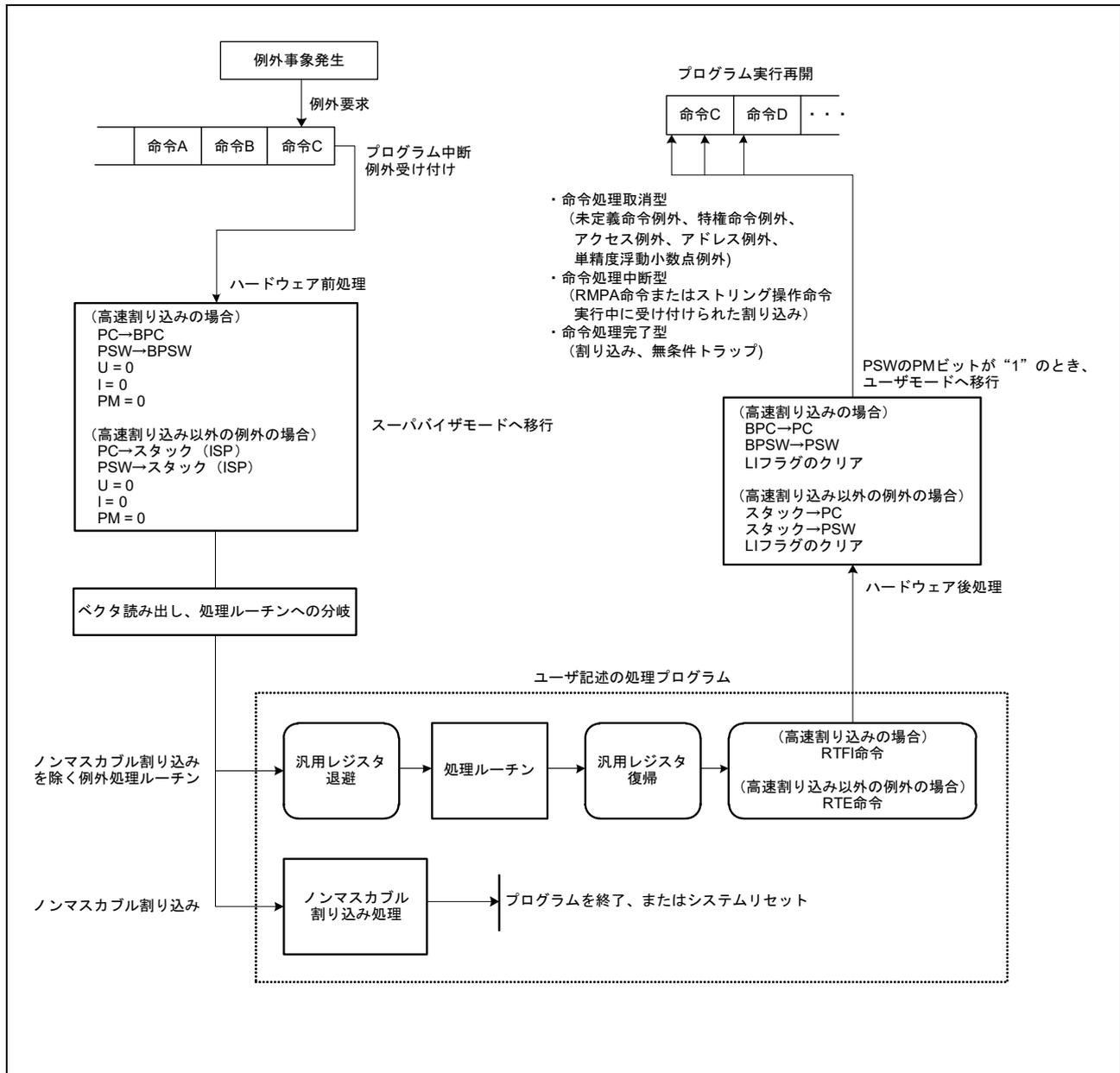


図 5.2 例外処理手順の概要

例外が受け付けられると、CPU はハードウェア処理を行った後、ベクタテーブルにアクセスし、分岐先アドレスを取得します。ベクタには例外ごとにベクタアドレスが割り当てられており、そこに例外処理ルーチンへの分岐先アドレスを書きます。

CPU のハードウェア前処理では、高速割り込みの場合は、プログラムカウンタ (PC) の内容をバックアップ PC (BPC) に、プロセッサステータスワード (PSW) の内容をバックアップ PSW (BPSW) へ退避させます。高速割り込み以外の例外では、PC、PSW をスタックに退避させます。例外処理ルーチン内で使用する汎用レジスタ、および PC、PSW 以外の制御レジスタについては、例外処理ルーチンの先頭でユーザープログラムによって退避させてください。

例外処理ルーチンの末尾で、退避させたレジスタを復帰させてから RTE 命令を実行することで、例外処理から元のプログラムに復帰します。高速割り込みの場合のみ、RTFI 命令を実行します。ノンマスクブル割り込みの場合には、元のプログラムに復帰せず、プログラムを終了、またはシステムリセットを行ってください。

CPU のハードウェア後処理では、高速割り込みの場合は BPC を PC に、また、BPSW の値を PSW に戻します。高速割り込み以外の例外では、スタックから PC、PSW の値を復帰させます。

例外処理ルーチンの先頭・末尾で行う汎用レジスタ等の退避・復帰には、スタックまたはレジスタ退避バンクが利用できます。

- (1) レジスタ退避バンクを搭載する製品では、SAVE・RSTR 命令を使用することでレジスタ退避バンクへの退避・復帰ができます。
- (2) レジスタ退避バンクを搭載しない製品、またはレジスタ退避バンクを搭載していても SAVE・RSTR 命令の退避・復帰対象になっていないレジスタを退避・復帰させる場合には、PUSH・POP 等の命令を使用してスタックへの退避・復帰を行ってください。

なお、例外処理ルーチンで退避させるレジスタが極端に少ない場合を除き、レジスタ退避バンクを使用した方がスタックを使用する場合よりも高速に動作します。

5.3 例外事象の受け付け

例外事象が発生すると、それまで実行していたプログラムを中断して、例外処理ルーチンに分岐します。

5.3.1 受け付けタイミングと保存されるPC値

各例外事象の受け付けタイミングと保存されるプログラムカウンタ（PC）の値を表5.1に示します。

表5.1 受け付けタイミングと保存されるPC値

例外事象	処理型	受け付け タイミング	BPC/スタックに保存されるPC値	
未定義命令例外	命令処理取消型	命令実行中	例外が発生した命令のPC値	
特権命令例外	命令処理取消型	命令実行中	例外が発生した命令のPC値	
アクセス例外	命令処理取消型	命令実行中	例外が発生した命令のPC値	
アドレス例外	命令処理取消型	命令実行中	例外が発生した命令のPC値	
単精度浮動小数点例外	命令処理取消型	命令実行中	例外が発生した命令のPC値	
リセット	命令処理放棄型	各マシンサイクル	なし	
ノンマスカブル 割り込み	RMPA、SCMPU、SMOVB、 SMOVF、SMOVU、SSTR、 SUNTIL、SWHILE命令実行中	命令処理中断型	命令実行中	実行中の命令のPC値
	上記以外の状態	命令処理完了型	命令の区切り	次の命令のPC値
割り込み	RMPA、SCMPU、SMOVB、 SMOVF、SMOVU、SSTR、 SUNTIL、SWHILE命令実行中	命令処理中断型	命令実行中	実行中の命令のPC値
	上記以外の状態	命令処理完了型	命令の区切り	次の命令のPC値
無条件トラップ	命令処理完了型	命令の区切り	次の命令のPC値	

5.3.2 ベクタとPC、PSWの退避場所

各例外事象のベクタとプログラムカウンタ（PC）、プロセッサステータスワード（PSW）の退避場所を表5.2に示します。

表5.2 ベクタとPC、PSWの退避場所

例外事象	ベクタ	PC、PSWの退避場所	
未定義命令例外	例外ベクタテーブル	スタック	
特権命令例外	例外ベクタテーブル	スタック	
アクセス例外	例外ベクタテーブル	スタック	
アドレス例外	例外ベクタテーブル	スタック	
単精度浮動小数点例外	例外ベクタテーブル	スタック	
リセット	例外ベクタテーブル	なし	
ノンマスカブル割り込み	例外ベクタテーブル	スタック	
割り込み	高速割り込み	FINTV	BPC、BPSW
	高速割り込み以外	割り込みベクタテーブル	スタック
無条件トラップ	割り込みベクタテーブル	スタック	

5.4 例外の受け付け / 復帰時のハードウェア処理

リセットを除く、例外の受け付けおよび復帰時のハードウェア処理について説明します。

(1) 例外受け付け時のハードウェア前処理

(a) PSW の退避

(高速割り込みの場合)

PSW → BPSW

(高速割り込み以外の例外の場合)

PSW → スタック

注. FPSW の退避は、ハードウェア前処理では行われません。浮動小数点演算命令を例外処理ルーチン内で使用する場合は、例外処理ルーチン内でスタックに退避させてください。

(b) PSW の PM、U、I ビットの更新

I: 0 にする

U: 0 にする

PM: 0 にする

(c) PC の退避

(高速割り込みの場合)

PC → BPC

(高速割り込み以外の例外の場合)

PC → スタック

(d) PC に例外処理ルーチン分岐先アドレスをセット

各例外に対応したベクタを取得し分岐することにより、例外処理へ移行します。

(2) RTE 命令、RTFI 命令実行時のハードウェア後処理

(a) PSW の復帰

(高速割り込みの場合)

BPSW → PSW

(高速割り込み以外の例外の場合)

スタック → PSW

(b) PC の復帰

(高速割り込みの場合)

BPC → PC

(高速割り込み以外の例外の場合)

スタック → PC

(c) LI フラグのクリア処理

5.5 ハードウェア前処理

例外要求が受け付けられてから例外処理ルーチンが実行されるまでのハードウェア前処理について説明します。

5.5.1 未定義命令例外

- (1) プロセッサステータスワード (PSW) の内容をスタック (ISP) に退避させます。
- (2) PSW のプロセッサモード設定ビット (PM)、スタックポインタ指定ビット (U)、割り込み許可ビット (I) を“0”にします。
- (3) プログラムカウンタ (PC) の内容をスタック (ISP) に退避させます。
- (4) EXTB の値 +0000005Ch 番地からベクタを取得します。
- (5) 取得したベクタを PC にセットし、例外処理ルーチンへ分岐します。

5.5.2 特権命令例外

- (1) プロセッサステータスワード (PSW) の内容をスタック (ISP) に退避させます。
- (2) PSW のプロセッサモード設定ビット (PM)、スタックポインタ指定ビット (U)、割り込み許可ビット (I) を“0”にします。
- (3) プログラムカウンタ (PC) の内容をスタック (ISP) に退避させます。
- (4) EXTB の値 +00000050h 番地からベクタを取得します。
- (5) 取得したベクタを PC にセットし、例外処理ルーチンへ分岐します。

5.5.3 アクセス例外

- (1) プロセッサステータスワード (PSW) の内容をスタック (ISP) に退避させます。
- (2) PSW のプロセッサモード設定ビット (PM)、スタックポインタ指定ビット (U)、割り込み許可ビット (I) を“0”にします。
- (3) プログラムカウンタ (PC) の内容をスタック (ISP) に退避させます。
- (4) EXTB の値 +00000054h 番地からベクタを取得します。
- (5) 取得したベクタを PC にセットし、例外処理ルーチンへ分岐します。

5.5.4 アドレス例外

- (1) プロセッサステータスワード (PSW) の内容をスタック (ISP) に退避させます。
- (2) PSW のプロセッサモード設定ビット (PM)、スタックポインタ指定ビット (U)、割り込み許可ビット (I) を“0”にします。
- (3) プログラムカウンタ (PC) の内容をスタック (ISP) に退避させます。
- (4) EXTB の値 +00000060h 番地からベクタを取得します。
- (5) 取得したベクタを PC にセットし、例外処理ルーチンへ分岐します。

5.5.5 単精度浮動小数点例外

- (1) プロセッサステータスワード (PSW) の内容をスタック (ISP) に退避させます。
- (2) PSW のプロセッサモード設定ビット (PM)、スタックポインタ指定ビット (U)、割り込み許可ビット (I) を“0”にします。
- (3) プログラムカウンタ (PC) の内容をスタック (ISP) に退避させます。
- (4) EXTB の値 +00000064h 番地からベクタを取得します。
- (5) 取得したベクタを PC にセットし、例外処理ルーチンへ分岐します。

5.5.6 リセット

- (1) 制御レジスタを初期化します。
- (2) FFFFFFFCh 番地からベクタを取得します。
- (3) 取得したベクタをプログラムカウンタ (PC) にセットします。

5.5.7 ノンマスカブル割り込み

- (1) プロセッサステータスワード (PSW) の内容をスタック (ISP) に退避させます。
- (2) PSW のプロセッサモード設定ビット (PM)、スタックポインタ指定ビット (U)、割り込み許可ビット (I) を“0”にします。
- (3) RMPA、SCMPU、SMOVB、SMOVF、SMOVU、SSTR、SUNTIL、SWHILE 命令の実行中は、実行中の命令のプログラムカウンタ (PC) の内容を、それ以外の状態では次の命令の PC の内容をスタック (ISP) に退避させます。
- (4) PSW のプロセッサ割り込み優先レベル (IPL[3:0]) を“Fh”にします。
- (5) EXTB の値 +00000078h 番地からベクタを取得します。
- (6) 取得したベクタを PC にセットし、例外処理ルーチンへ分岐します。

5.5.8 割り込み

- (1) プロセッサステータスワード (PSW) の内容をスタック (ISP) に退避させます。高速割り込みの場合は、バックアップ PSW (BPSW) に退避させます。
- (2) PSW のプロセッサモード設定ビット (PM)、スタックポインタ指定ビット (U)、割り込み許可ビット (I) を“0”にします。
- (3) RMPA、SCMPU、SMOVB、SMOVF、SMOVU、SSTR、SUNTIL、SWHILE 命令の実行中は、実行中の命令のプログラムカウンタ (PC) の内容を、それ以外の状態では次の命令の PC の内容をスタック (ISP) に退避させます。高速割り込みの場合は、バックアップ PC (BPC) に退避させます。
- (4) PSW のプロセッサ割り込み優先レベル (IPL[3:0]) に、受け付けた割り込みの割り込み優先レベルを設定します。
- (5) 割り込みベクタテーブルから受け付けた割り込み要因のベクタを取得します。高速割り込みの場合は、高速割り込みベクタレジスタ (FINTV) からベクタを取得します。
- (6) 取得したベクタを PC にセットし、例外処理ルーチンへ分岐します。

5.5.9 無条件トラップ

- (1) プロセッサステータスワード (PSW) の内容をスタック (ISP) に退避させます。
- (2) PSW のプロセッサモード設定ビット (PM)、スタックポインタ指定ビット (U)、割り込み許可ビット (I) を“0”にします。
- (3) 次の命令のプログラムカウンタ (PC) の内容をスタック (ISP) に退避させます。
- (4) INT 命令の場合は、割り込みベクタテーブルから INT 命令番号に対応したベクタを取得します。
BRK 命令の場合は、割り込みベクタテーブルの先頭番地からベクタを取得します。
- (5) 取得したベクタを PC にセットし、例外処理ルーチンへ分岐します。

5.6 例外処理ルーチンからの復帰

例外処理ルーチンの最後で表 5.3 に示す命令を実行すると、ハードウェア前処理でスタックまたはバックアップ PC (BPC)、バックアップ PSW (BPSW) に退避させられていたプログラムカウンタ (PC) とプロセッサステータスワード (PSW) の内容が復帰させられます。

表 5.3 例外処理ルーチンからの復帰命令

例外事象		復帰命令
未定義命令例外		RTE
特権命令例外		RTE
アクセス例外		RTE
アドレス例外		RTE
単精度浮動小数点例外		RTE
リセット		復帰不可能
ノンマスカブル割り込み		復帰禁止
割り込み	高速割り込み	RTFI
	高速割り込み以外	RTE
無条件トラップ		RTE

5.7 例外事象の優先順位

例外事象の優先順位を表 5.4 に示します。複数の例外が同時に発生した場合は、より優先度の高い事象が先に受け付けられます。

表 5.4 例外事象の優先順位

優先順位	例外事象
高い  低い	1 リセット
	2 ノンマスカブル割り込み
	3 割り込み
	4 命令アクセス例外
	5 未定義命令例外 特権命令例外
	6 無条件トラップ
	7 アドレス例外
	8 オペランドアクセス例外
	9 単精度浮動小数点例外

5.8 コプロセッサの例外事象

コプロセッサで発生する例外事象とその例外処理について説明します。

5.8.1 倍精度浮動小数点例外

倍精度浮動小数点例外は、倍精度浮動小数点演算命令実行時に、IEEE754 規格で規定された 5 つの例外（オーバフロー、アンダフロー、精度異常、ゼロ除算、無効演算）および非実装処理を検出した場合に発生します。倍精度浮動小数点例外が発生すると、CPU での例外処理は行われず、割り込みコントローラへの割り込み要求が行われます。5 つの例外については、対応する DPSW の DEX、DEU、DEZ、DEO、DEV ビットが“1”のときのみ割り込み要求が行われます。

割り込みコントローラのベクタ番号の割り当ては、各製品のユーザーズマニュアル ハードウェア編を参照してください。

索引

記号

+∞方向への丸め	22
-∞方向への丸め	22
0方向への丸め	22

B

BPC (バックアップPC)	20
BPSW (バックアップPSW)	20

C

CEフラグ (非実装処理要因フラグ)	22
COフラグ (オーバフロー要因フラグ)	22
CUフラグ (アンダフロー要因フラグ)	22
CVフラグ (無効演算要因フラグ)	22
CXフラグ (精度異常要因フラグ)	22
CZフラグ (ゼロ除算要因フラグ)	22
Cフラグ (キャリフラグ)	19

D

DCEフラグ (非実装処理要因フラグ)	39
DCMR (倍精度浮動小数点比較結果レジスタ)	40
DCOフラグ (オーバフロー要因フラグ)	39
DCUフラグ (アンダフロー要因フラグ)	39
DCVフラグ (無効演算要因フラグ)	39
DCXフラグ (精度異常要因フラグ)	39
DCZフラグ (ゼロ除算要因フラグ)	39
DDN ビット (非正規化数の0フラッシュビット)	39
DECNT (倍精度浮動小数点例外処理動作制御レジスタ)	41
DEO ビット (オーバフロー例外処理許可ビット)	40
DEPC (倍精度浮動小数点例外プログラムカウンタ)	41
DEU ビット (アンダフロー例外処理許可ビット)	40
DEV ビット (無効演算例外処理許可ビット)	40
DEX ビット (精度異常例外処理許可ビット)	40
DEZ ビット (ゼロ除算例外処理許可ビット)	40
DFOフラグ (オーバフローフラグ)	40
DFSフラグ (倍精度浮動小数点エラーサマリフラグ)	40
DFUフラグ (アンダフローフラグ)	40
DFVフラグ (無効演算フラグ)	40
DFXフラグ (精度異常フラグ)	40
DFZフラグ (ゼロ除算フラグ)	40
DN ビット (非正規化数の0フラッシュビット)	22
DPSW (倍精度浮動小数点ステータスワード)	38
DRM[1:0] ビット (倍精度浮動小数点丸めモード設定ビット)	39

E

EHM ビット (倍精度浮動小数点例外発生情報保持 モードビット)	41
EHS ビット (倍精度浮動小数点例外発生情報保持 ステータスビット)	41
EO ビット (オーバフロー例外処理許可ビット)	23
EU ビット (アンダフロー例外処理許可ビット)	23
EV ビット (無効演算例外処理許可ビット)	23
EXTB (例外テーブルレジスタ)	23
EX ビット (精度異常例外処理許可ビット)	23
EZ ビット (ゼロ除算例外処理許可ビット)	23

F

FINTV (高速割り込みベクタレジスタ)	20
FOフラグ (オーバフローフラグ)	23
FPSW (単精度浮動小数点ステータスワード)	21
FSフラグ (単精度浮動小数点エラーサマリフラグ)	23
FUフラグ (アンダフローフラグ)	23
FVフラグ (無効演算フラグ)	23
FXフラグ (精度異常フラグ)	23
FZフラグ (ゼロ除算フラグ)	23

I

INTB (割り込みテーブルレジスタ)	17
IPL[3:0] ビット (プロセッサ割り込み優先レベル)	19
ISP (割り込みスタックポインタ)	17
I ビット (割り込み許可ビット)	19

N

NaN (Not a Number)	27
--------------------------	----

O

Oフラグ (オーバフローフラグ)	19
------------------------	----

P

PC (プログラムカウンタ)	17
PM ビット (プロセッサモード設定ビット)	19
PSW (プロセッサステータスワード)	18
PSW直接	52

Q

QNaN (Quiet NaN)	27
------------------------	----

R

R0(SP)～R15 (汎用レジスタ)	16
register(n)	65
register_num(Rn)	65
RES フラグ (倍精度浮動小数点比較命令結果フラグ) ...	40
RM[1:0] ビット (単精度浮動小数点丸めモード設定ビット)	22

S

SNaN (Signaling NaN)	27
S フラグ (サインフラグ)	19

U

USP (ユーザスタックポインタ)	17
U ビット (スタックポインタ指定ビット)	19

Z

Z フラグ (ゼロフラグ)	19
---------------------	----

あ

アキュムレータ (ACC)	24
アキュムレータ直接	52
アクセス例外	357
アドレス例外	357
アンダフローフラグ (DFU フラグ)	40
アンダフローフラグ (FU フラグ)	23
アンダフロー要因フラグ (CU フラグ)	22
アンダフロー要因フラグ (DCU フラグ)	39
アンダフロー例外処理許可ビット (DEU ビット)	40
アンダフロー例外処理許可ビット (EU ビット)	23

い

インデックス付きレジスタ間接	51
----------------------	----

お

オーバフローフラグ (DFO フラグ)	40
オーバフローフラグ (FO フラグ)	23
オーバフローフラグ (O フラグ)	19
オーバフロー要因フラグ (CO フラグ)	22
オーバフロー要因フラグ (DCO フラグ)	39
オーバフロー例外処理許可ビット (DEO ビット)	40
オーバフロー例外処理許可ビット (EO ビット)	23

き

キャリフラグ (C フラグ)	19
----------------------	----

こ

高速割り込みベクタレジスタ (FINTV)	20
-----------------------------	----

さ

最近値への丸め	22
サイズ拡張指定子	67
サイズ指定子	64
サインフラグ (S フラグ)	19

す

スーパーバイザモード	28
スタックポインタ (R0(SP))	16
スタックポインタ指定ビット (U ビット)	19
ストリング	30

せ

制御レジスタ	16
制御レジスタ直接	51
整数	29
精度異常フラグ (DFX フラグ)	40
精度異常フラグ (FX フラグ)	23
精度異常要因フラグ (CX フラグ)	22
精度異常要因フラグ (DCX フラグ)	39
精度異常例外処理許可ビット (DEX ビット)	40
精度異常例外処理許可ビット (EX ビット)	23
ゼロ除算フラグ (DFZ フラグ)	40
ゼロ除算フラグ (FZ フラグ)	23
ゼロ除算要因フラグ (CZ フラグ)	22
ゼロ除算要因フラグ (DCZ フラグ)	39
ゼロ除算例外処理許可ビット (DEZ ビット)	40
ゼロ除算例外処理許可ビット (EZ ビット)	23
ゼロフラグ (Z フラグ)	19

そ

即値	49
----------	----

た

単精度浮動小数点エラーサマリフラグ (FS フラグ)	23
単精度浮動小数点数	29
単精度浮動小数点ステータスワード (FPSW)	21
単精度浮動小数点丸めモード設定ビット (RM[1:0] ビット)	22
単精度浮動小数点例外	25, 357

と

特権命令	28
------------	----

特権命令例外 357

の

ノンマスカブル割り込み 357

は

倍精度浮動小数点エラーサマリフラグ (DFSフラグ) ... 40
 倍精度浮動小数点コプロセッサ 36
 倍精度浮動小数点数 45
 倍精度浮動小数点ステータスワード (DPSW) 38
 倍精度浮動小数点制御レジスタ直接 53
 倍精度浮動小数点データレジスタ直接 53
 倍精度浮動小数点比較結果レジスタ (DCMR) 40
 倍精度浮動小数点比較命令結果フラグ (RESフラグ) ... 40
 倍精度浮動小数点丸めモード設定ビット
 (DRM[1:0]ビット) 39
 倍精度浮動小数点例外 42, 366
 倍精度浮動小数点例外処理動作制御レジスタ
 (DECNT) 41
 倍精度浮動小数点例外発生情報保持ステータスビット
 (EHSビット) 41
 倍精度浮動小数点例外発生情報保持モードビット
 (EHMビット) 41
 倍精度浮動小数点例外プログラムカウンタ (DEPC) 41
 バックアップPC (BPC) 20
 バックアップPSW (BPSW) 20
 汎用レジスタ (R0(SP)~R15) 16

ひ

非実装処理要因フラグ (CEフラグ) 22
 非実装処理要因フラグ (DCEフラグ) 39
 非正規化数の0フラッシュビット (DNビット) 22
 非正規化数の0フラッシュビット (DDNビット) 39
 ビット 30

ふ

プリデクリメントレジスタ間接 51
 プログラムカウンタ (PC) 17
 プログラムカウンタ相対 52
 プロセッサステータスワード (PSW) 18
 プロセッサモード 28
 プロセッサモード設定ビット (PMビット) 19
 プロセッサ割り込み優先レベル (IPL[3:0]ビット) 19

へ

ベクタテーブル 32

ほ

ポストインクリメントレジスタ間接 51

ま

丸め 22

み

未定義命令例外 357

む

無効演算フラグ (DFVフラグ) 40
 無効演算フラグ (FVフラグ) 23
 無効演算要因フラグ (CVフラグ) 22
 無効演算要因フラグ (DCVフラグ) 39
 無効演算例外処理許可ビット (DEVビット) 40
 無効演算例外処理許可ビット (EVビット) 23
 無条件トラップ 357

ゆ

ユーザスタックポインタ (USP) 17
 ユーザモード 28

り

リセット 357

れ

例外事象の優先順位 365
 例外テーブルレジスタ (EXTB) 23
 例外ベクタテーブル 32
 レジスタ一括退避機能 35
 レジスタ間接 50
 レジスタ相対 50
 レジスタ直接 50

わ

割り込み 357
 割り込み許可ビット (Iビット) 19
 割り込みスタックポインタ (ISP) 17
 割り込みテーブルレジスタ (INTB) 17
 割り込みベクタテーブル 33

改訂記録	RXファミリ RXv3 命令セットアーキテクチャ ユーザーズマニュアル ソフトウェア編
------	---

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2018.11.20	—	初版発行

RXファミリ RXv3命令セットアーキテクチャ ユーザーズマニュアル
ソフトウェア編

発行年月日 2018年11月20日 Rev.1.00

発行 ルネサス エレクトロニクス株式会社
〒135-0061 東京都江東区豊洲3-2-24 (豊洲フォレシア)



ルネサスエレクトロニクス株式会社

営業お問合せ窓口

<http://www.renesas.com>

営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社 〒135-0061 東京都江東区豊洲3-2-24 (豊洲フォレシア)

技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：<https://www.renesas.com/contact/>

RXファミリ RXv3命令セットアーキテクチャ



ルネサスエレクトロニクス株式会社