User's Manual

32

RX ファミリ RXv2 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

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2013.11

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、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 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に関し、当社は、何らの責任を負うものではありません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製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）、交通用信号機器、
   防災・防犯装置、各種安全装置等

   当社製品は、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（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
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、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害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。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 本資料に記載されている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。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
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。

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

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

(2012.4)

はじめに

このマニュアルは、ユーザの皆様に RXv2 命令セットアーキテクチャ（RXv2）のソフトウェアをよく理解していただき、その機能を最大限に生かしていただくために作成しました。

CPU の特長や命令体系について詳細に記載しておりますので、広くご活用ください。

なお、RX ファミリの各ハードウェアおよび開発環境につきましては、各製品のユーザーズマニュアルおよび各種関連ドキュメントをご併用くださいますようお願い申し上げます。

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	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	一時レジスタを表します。
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:8など	オペランド記号の有効ビット長を表します。
	:1	有効ビット長が1ビットであることを表します。
	:2	有効ビット長が2ビットであることを表します。
	:3	有効ビット長が3ビットであることを表します。
	:4	有効ビット長が4ビットであることを表します。
	:5	有効ビット長が5ビットであることを表します。
	:8	有効ビット長が8ビットであることを表します。
	:16	有効ビット長が16ビットであることを表します。
	:24	有効ビット長が24ビットであることを表します。
	:32	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.Wなど	命令の取り扱うサイズを指定する記号です。
	.B	バイト（8ビット）を指定します。
	.W	ワード（16ビット）を指定します。
	.L	ロングワード（32ビット）を指定します。

分類	略号	意味
分岐距離指定	BRA.A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。
	.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。
メモリオペランドに付加されるサイズ拡張指定	dsp:16[Rs].UB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。
	.B	バイト（8ビット）を指定します。拡張方法は符号拡張です。
	.UB	バイト（8ビット）を指定します。拡張方法はゼロ拡張です。
	.W	ワード（16ビット）を指定します。拡張方法は符号拡張です。
	.UW	ワード（16ビット）を指定します。拡張方法はゼロ拡張です。
	.L	ロングワード（32ビット）を指定します。
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

RX ファミリ RXv2 命令セットアーキテクチャ命令一覧

アルファベット順ページ早見表 (1 / 4)

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
CLRPSW	PSWのフラグ、ビットのクリア	71	217
CMP	比較	72	218
DIV	符号付き除算	73	219
DIVU	符号なし除算	74	221
EMACA	32ビット積和演算	75	222
EMSBA	32ビット積差演算	76	222
EMUL	符号付き乗算	77	223
EMULA	32ビット乗算	79	224
EMULU	符号なし乗算	80	224
FADD	浮動小数点加算	82	226
FCMP	浮動小数点比較	85	227
FDIV	浮動小数点除算	87	228
FMUL	浮動小数点乗算	89	229
FSQRT	浮動小数点平方根	92	230
FSUB	浮動小数点減算	94	231
FTOI	浮動小数点数→整数変換	97	232
FTOU	浮動小数点数→整数変換	99	232
INT	ソフトウェア割り込み	101	233
ITOF	整数→浮動小数点数変換	102	233
JMP	無条件分岐	104	234
JSR	サブルーチン分岐	105	234
MACHI	上位16ビット積和演算	106	235
MACLH	下位16ビット・上位16ビット積和演算	107	235
MACLO	下位16ビット積和演算	108	235
MAX	最大値選択	109	236
MIN	最小値選択	110	237
MOV	転送	111	238
MOVCO	LIフラグクリア付きストア	115	243
MOVLI	LIフラグセット付きロード	116	243
MOVU	符号なしデータ転送	117	244
MSBHI	上位16ビット積差演算	118	245
MSBLH	下位16ビット・上位16ビット積差演算	119	245
MSBLO	下位16ビット積差演算	120	246
MUL	乗算	121	246
MULHI	上位16ビット乗算	123	248
MULLH	下位16ビット・上位16ビット乗算	124	248
MULLO	下位16ビット乗算	125	249
MVFACGU	アキュムレータガードビットからの転送	126	249
MVFACHI	アキュムレータ上位32ビットからの転送	127	250
MVFACLO	アキュムレータ下位32ビットからの転送	128	250
MVFACMI	アキュムレータ中央32ビットからの転送	129	251
MVFC	制御レジスタからの転送	130	251
MVTACGU	アキュムレータガードビットへの転送	131	252
MVTACHI	アキュムレータ上位32ビットへの転送	132	252
MVTACLO	アキュムレータ下位32ビットへの転送	133	252
MVTC	制御レジスタへの転送	134	253
MVTIPL（特権命令）	割り込み優先レベル設定	135	254
NEG	符号反転	136	255
NOP	ノーオペレーション	137	255

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
NOT		論理反転	138	256
OR		論理和	139	257
POP		スタックからレジスタへのデータ復帰	141	258
POPC		制御レジスタの復帰	142	259
POPM		複数レジスタの復帰	143	259
PUSH		スタックへデータ退避	144	260
PUSHC		制御レジスタの退避	145	261
PUSHM		複数レジスタの退避	146	261
RACL		符号付きアキュムレータ丸め処理	147	262
RACW		16ビット符号付きアキュムレータ丸め処理	149	262
RDACL		符号付きアキュムレータ丸め処理	151	263
RDACW		16ビット符号付きアキュムレータ丸め処理	153	263
REVL		エンディアン変換	155	264
REVW		エンディアン変換	156	264
RMPA		積和演算	157	265
ROLC		キャリ付き左回転	159	265
RORC		キャリ付き右回転	160	265
ROTL		左回転	161	266
ROTR		右回転	162	266
ROUND		浮動小数点数→整数変換	163	267
RTE（特権命令）		例外からの復帰	166	267
RTFI（特権命令）		高速割り込みからの復帰	167	267
RTS		サブルーチンからの復帰	168	268
RTSD		スタックフレームの解放とサブルーチンからの復帰	169	268
SAT		32ビット符号付き飽和処理	171	268
SATR		RMPA命令用64ビット符号付き飽和処理	172	269
SBB		ボロー付き減算	173	269
SC*Cnd*	SCGEU	条件設定	174	270
	SCC		174	270
	SCEQ		174	270
	SCZ		174	270
	SCGTU		174	270
	SCPZ		174	270
	SCGE		174	270
	SCGT		174	270
	SCO		174	270
	SCLTU		174	270
	SCNC		174	270
	SCNE		174	270
	SCNZ		174	270
	SCLEU		174	270
	SCN		174	270
	SCLE		174	270
	SCLT		174	270
	SCNO		174	270
SCMPU		ストリング比較	175	270
SETPSW		PSWのフラグ、ビットのセット	176	271
SHAR		算術右シフト	177	272
SHLL		論理/算術左シフト	178	273
SHLR		論理右シフト	179	274

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
SMOVB	逆方向ストリング転送	180	274
SMOVF	順方向ストリング転送	181	275
SMOVU	ストリング転送	182	275
SSTR	ストリングストア	183	275
STNZ	条件付き転送	184	276
STZ	条件付き転送	185	277
SUB	ボローなし減算	186	278
SUNTIL	ストリングサーチ	187	279
SWHILE	ストリングサーチ	189	279
TST	テスト	191	280
UTOF	整数→浮動小数点数変換	192	281
WAIT（特権命令）	ウェイト	194	282
XCHG	交換	195	282
XOR	排他的論理和	196	283

1. CPU プログラミングモデル

RXv2 命令セットアーキテクチャ (RXv2) は、RXv1 命令セットアーキテクチャ (RXv1) 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
  RXv1 と同様に、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
  RXv2 は厳選された 109 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の拡充により、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）
- 109 命令 /11 種類アドレッシングモード
  基本命令：75 種類
  浮動小数点演算命令：11 種類
  DSP 機能命令：23 種類
- プロセッサモード
  スーパバイザモード、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

1.2 CPU レジスタセット

RXv2 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。

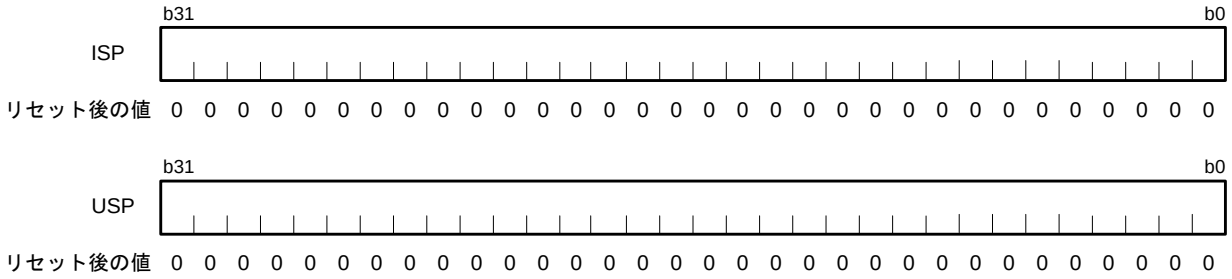
汎用レジスタ R0 には、汎用レジスタとしての機能に加えて、スタックポインタ（SP）としての機能が割り当てられています。SP は、プロセッサステータスワード（PSW）のスタックポインタ指定ビット（U）によって、割り込みスタックポインタ（ISP）、またはユーザスタックポインタ（USP）に切り替わります。

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。

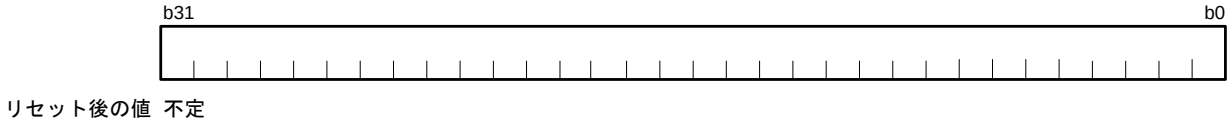
- 割り込みスタックポインタ（ISP）
- ユーザスタックポインタ（USP）
- 割り込みテーブルレジスタ（INTB）
- プログラムカウンタ（PC）
- プロセッサステータスワード（PSW）
- バックアップ PC（BPC）
- バックアップ PSW（BPSW）
- 高速割り込みベクタレジスタ（FINTV）
- 浮動小数点ステータスワード（FPSW）
- 例外テーブルレジスタ (EXTB)

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）



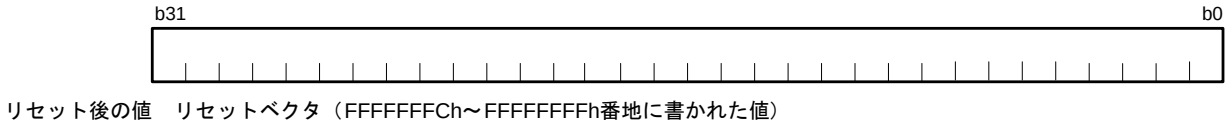
スタックポインタ（SP）には、割り込みスタックポインタ（ISP）と、ユーザスタックポインタ（USP）の 2 種類があります。使用するスタックポインタ（ISP/USP）は、プロセッサステータスワード（PSW）のスタックポインタ指定ビット（U）によって切り替えられます。

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す。

注3.　スーパバイザモードからユーザモードに切り替える場合は、スタックに退避された 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、それ以外のとき “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、それ以外のとき “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、それ以外のとき “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）

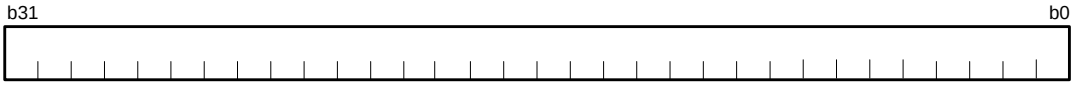
IPL[3:0] ビットは、優先レベル 0（最低）～ 優先レベル 15（最高）までの 16 段階のプロセッサ割り込み優先レベルを指定します。要求があった割り込みの優先レベルが、プロセッサ割り込み優先レベルより高い場合、その割り込みが許可されます。IPL[3:0] ビットをレベル 15（Fh）に設定したとき、すべての割り込みが禁止されます。IPL[3:0] ビットは、ノンマスカブル割り込みが発生したとき、レベル 15（Fh）になります。割り込みが発生したとき、受け付けた割り込みの優先レベルになります。

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。

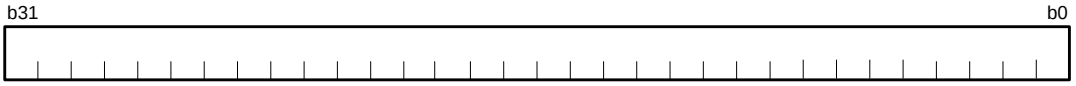
1.2.2.6 バックアップ PSW（BPSW）



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　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]	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	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。

浮動小数点ステータスワード（FPSW）は、浮動小数点演算結果を示します。浮動小数点命令に対応していない製品では、“00000000h” が読め、書き込みは無視されます。

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（浮動小数点丸めモード設定ビット）

浮動小数点丸めモードを設定します。

【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して計算した場合の結果と近い方の値へ丸める。中間時は結果が偶数になる方向へ丸める
- 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。

- “1” の場合、浮動小数点演算命令実行時に “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）

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）

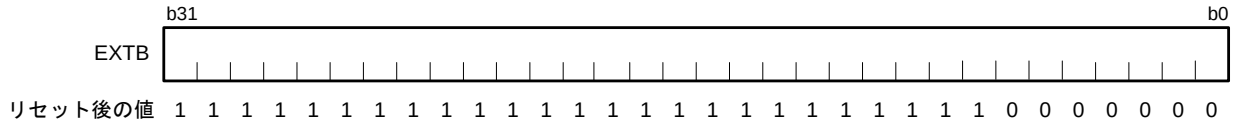
例外処理許可ビット Ej が “0”（例外処理を禁止）の場合、IEEE754 規格で規定された 5 つの例外が発生すると、該当するフラグが “1” になります。

- Ej=“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（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。

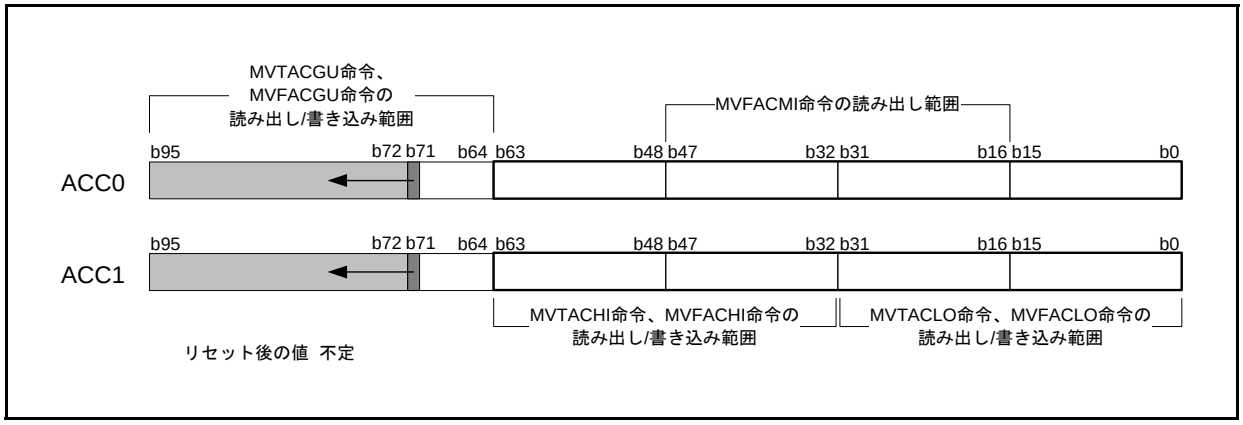
1.2.3　アキュムレータ

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「MVFACLO 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）、「MVFACLO 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 浮動小数点例外

浮動小数点例外は、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、非実装処理を検出した場合に発生します。以下に、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

オーバフローは、演算結果の絶対値が浮動小数点フォーマットで表現可能な値よりも大きくなった場合に発生します。**表 1.1** にオーバ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

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”
オーバフロー例外禁止状態でのオーバフロー発生	「表1.1　オーバ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。

1.3.4　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。

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	±∞（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

注.　以下のときは、ゼロ除算は発生しません。

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 1.5** に無効演算発生条件と演算結果を示します。

表 1.5　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
+∞ +(−∞) 、+∞ −(+∞) 、−∞ −(−∞)		
0×∞		
0÷0、∞÷∞		
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 FFFFFFFFh、“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、無効演算を発生します。変数の初期値として使用することにより、プログラムのバグの発見に役立ちます。なお、SNaN はハードウェアで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 1.6 に演算結果が QNaN となる場合の生成規則を示します。

表 1.6　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

RXv2 CPU には、スーパバイザモード、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、スーパバイザモードで動作します。

1.4.1　スーパバイザモード

スーパバイザモードでは、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 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、退避された PSW の PM ビットに保持されます。

(2) スーパ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

RXv2 CPU は、整数、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。

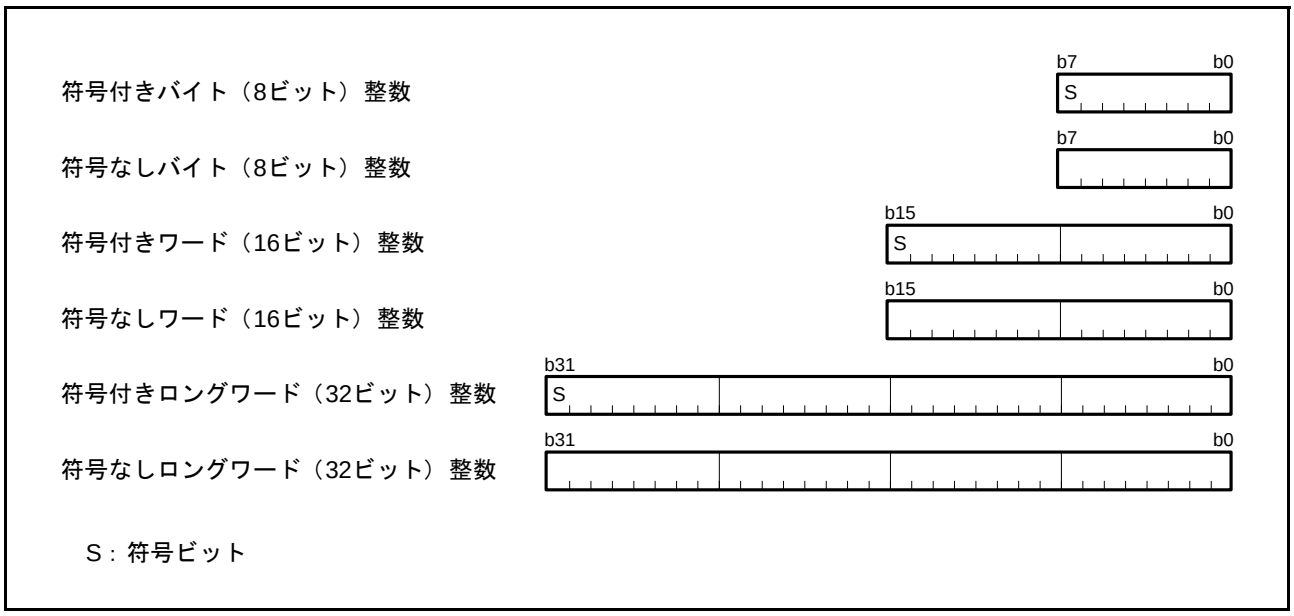


図 1.2　整数

1.5.2　浮動小数点数

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。浮動小数点数は、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。

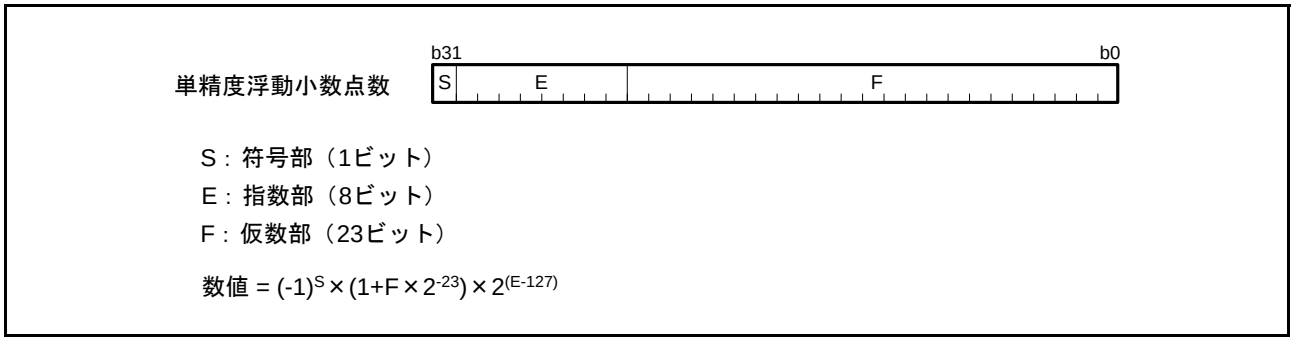


図 1.3　浮動小数点数

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、BM*Cnd* 、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。

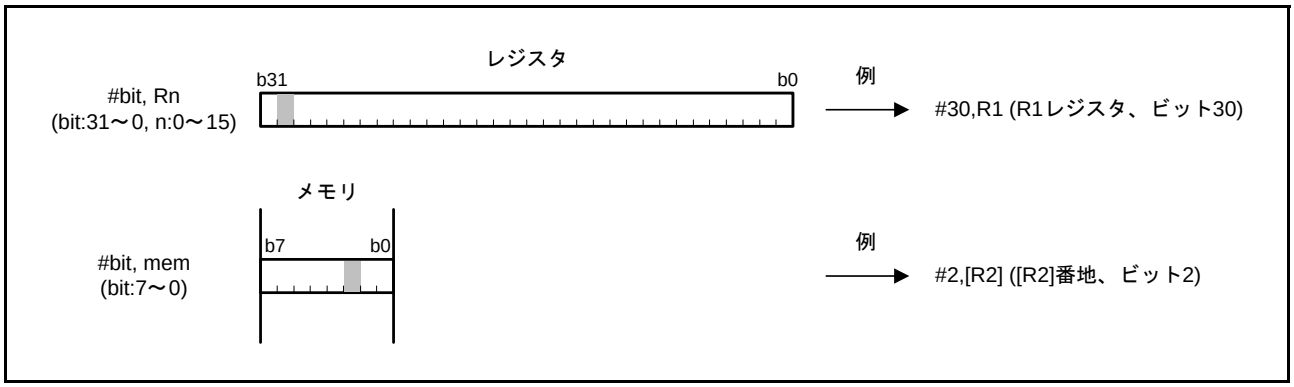


図 1.4　ビット

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。

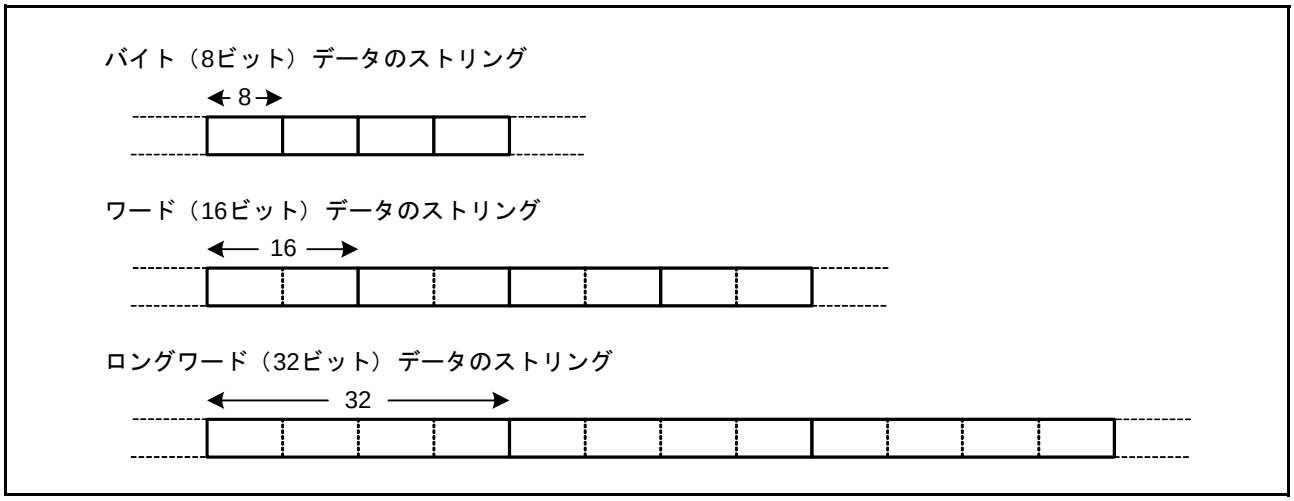


図 1.5　ストリング

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。

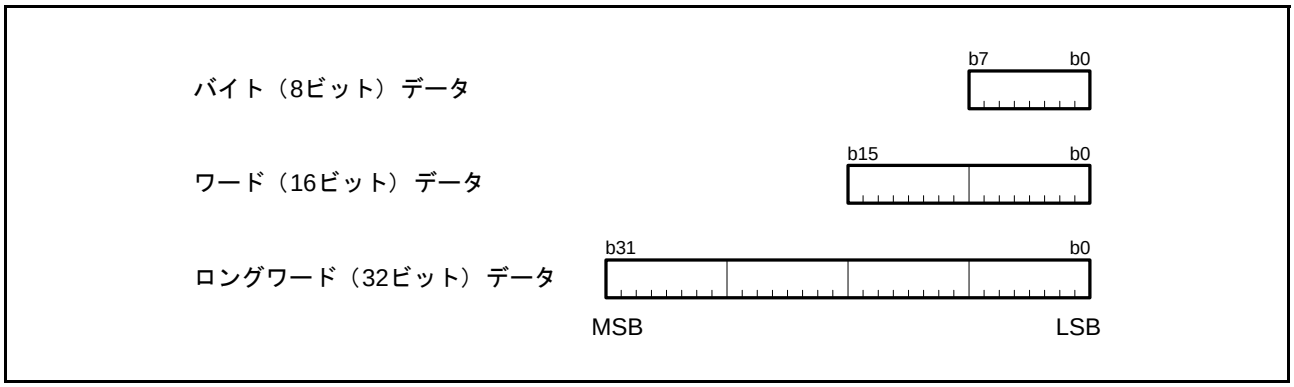


図 1.6　レジスタのデータ配置

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。

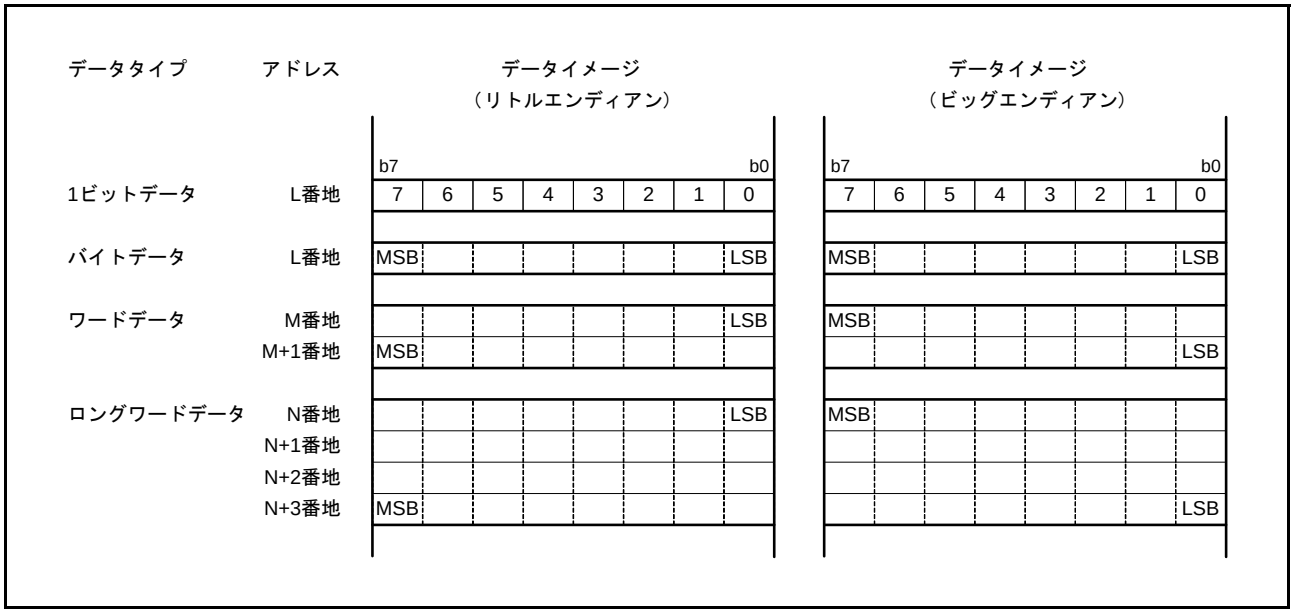


図 1.7　メモリ上のデータ配置

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 128 バイトの領域に、特権命令例外、アクセス例外、未定義命令例外、浮動小数点例外、ノンマスカブル割り込み、リセットの各ベクタを配置しています。ただし、リセットのベクタは、FFFFFFFCh 番地に配置されます。

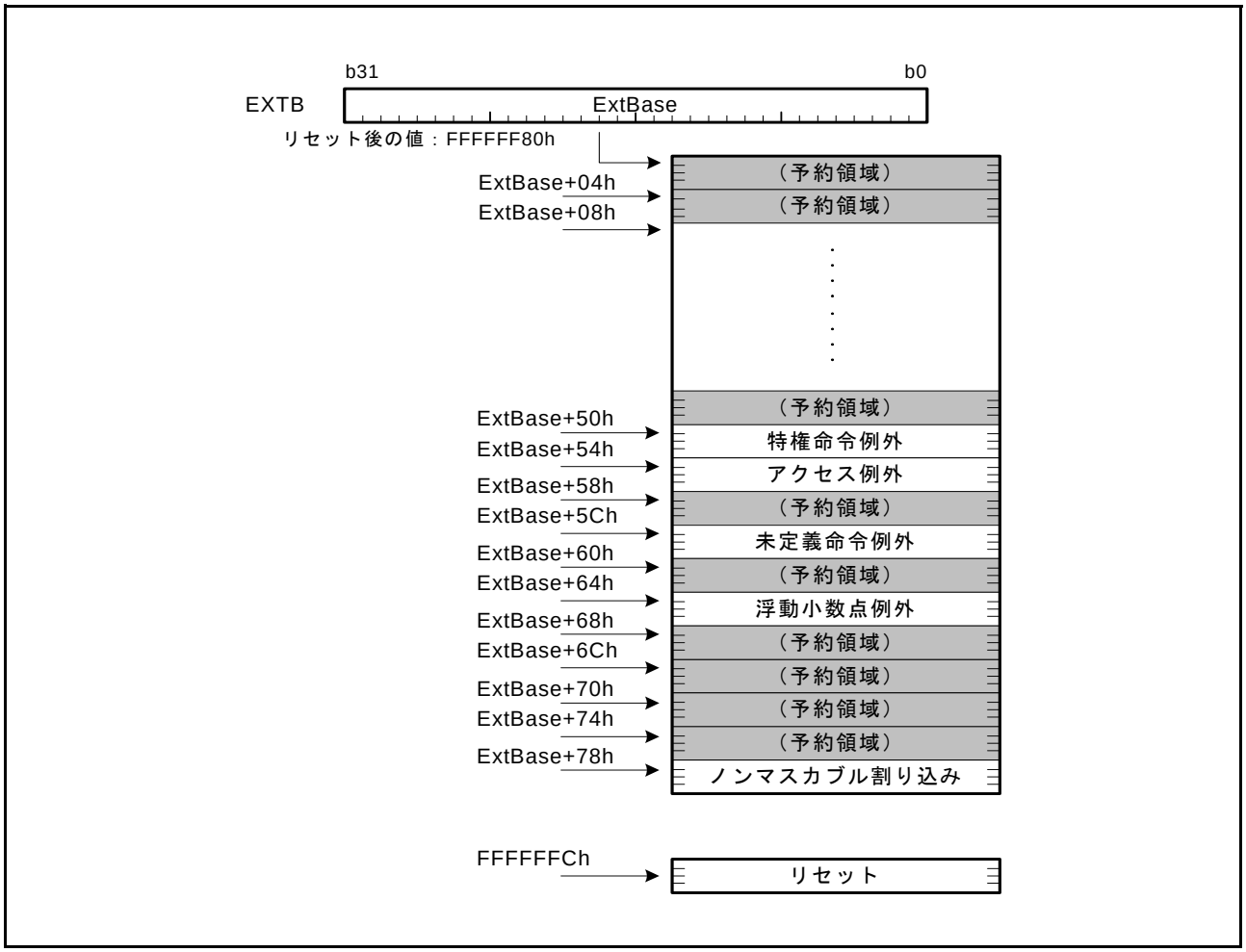


図 1.8　例外ベクタテーブル

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。

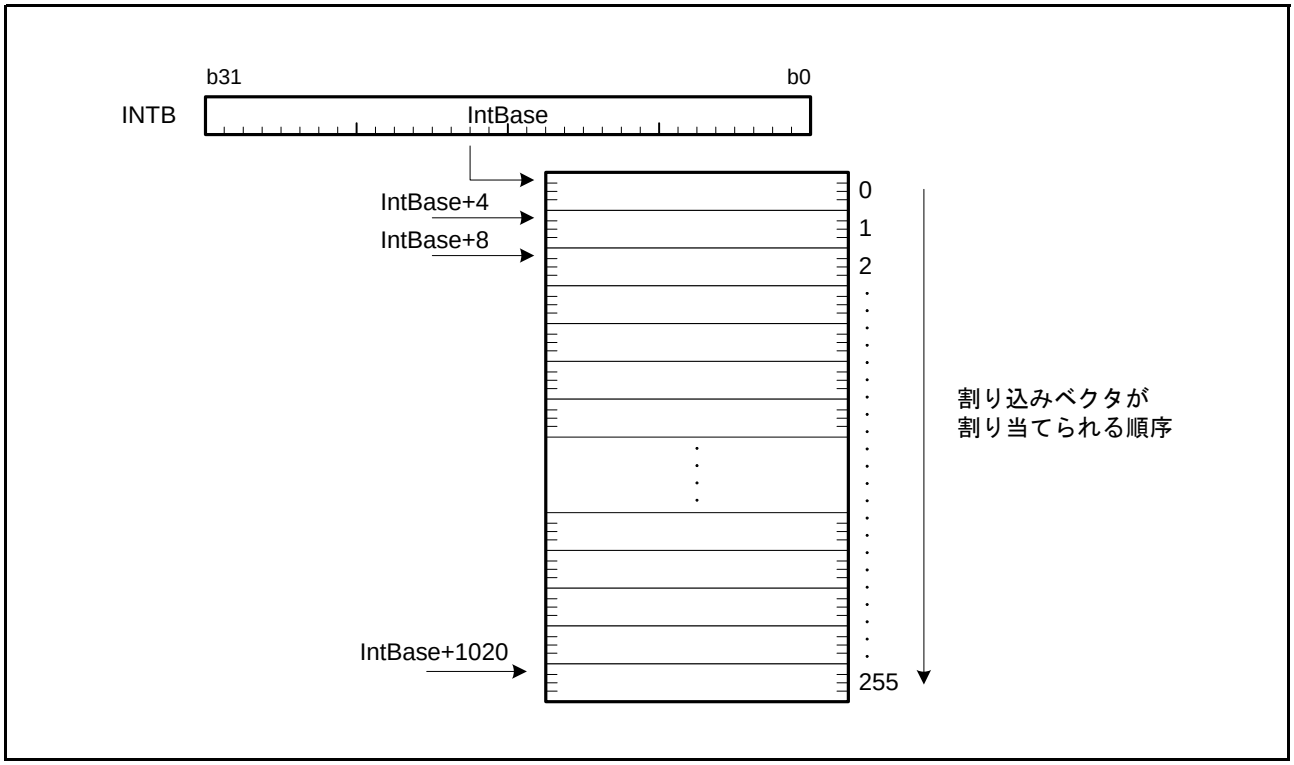


図 1.9　割り込みベクタテーブル

1.8　アドレス空間

RXv2 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。RXv2 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。

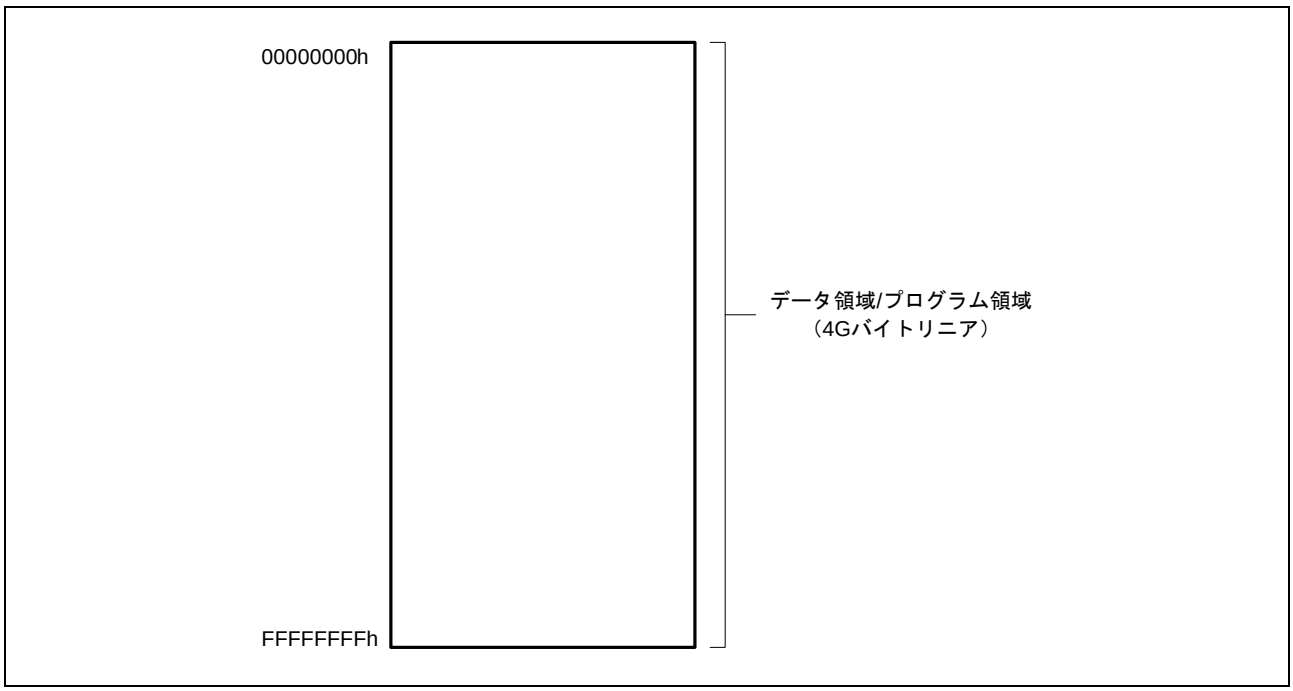


図 1.10　アドレス空間

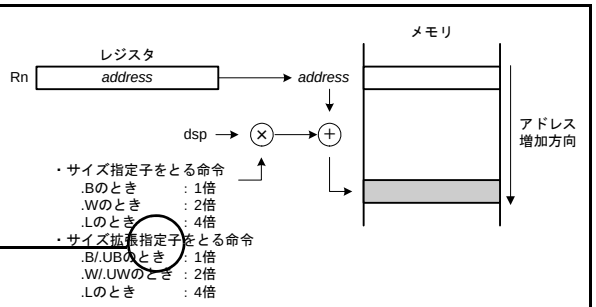
2. アドレッシングモード

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

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～FFFFFFFFhです。dsp:nは、nビット長のディスプレースメントを表します。 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。	
(3) dsp:8[Rn] (Rn=R0~R15)		
(4) dsp:16[Rn] (Rn=R0~R15)		

(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。

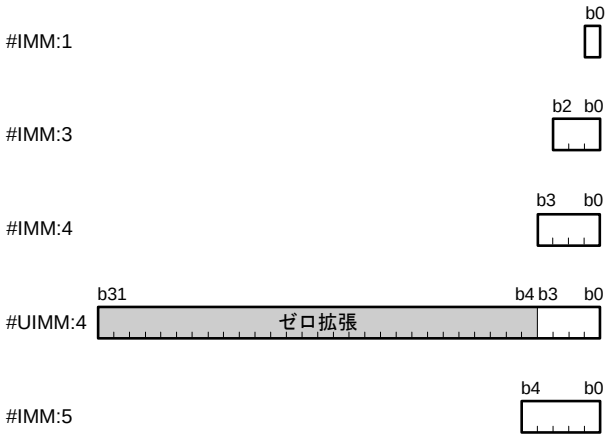
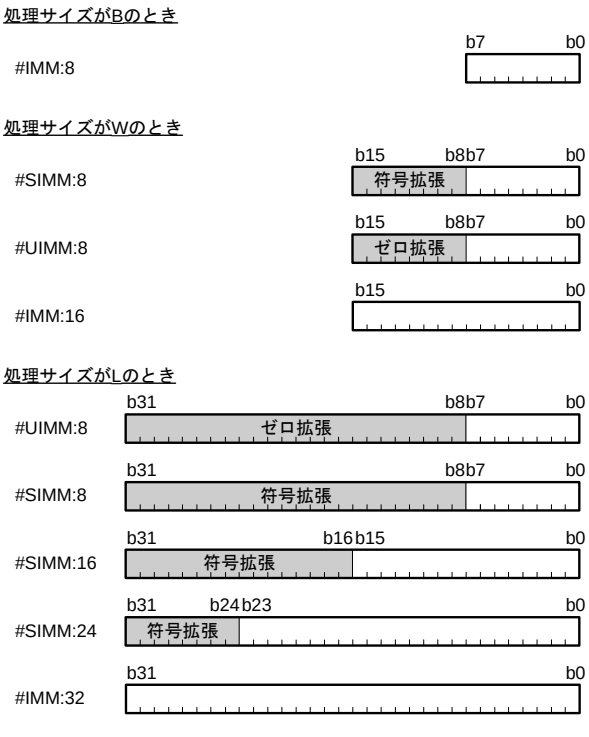
2.2 アドレッシングモード

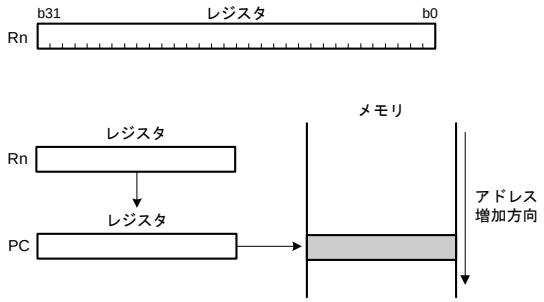
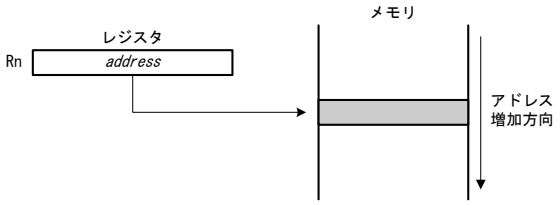
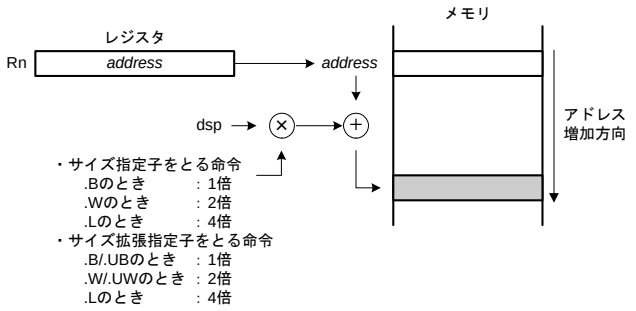
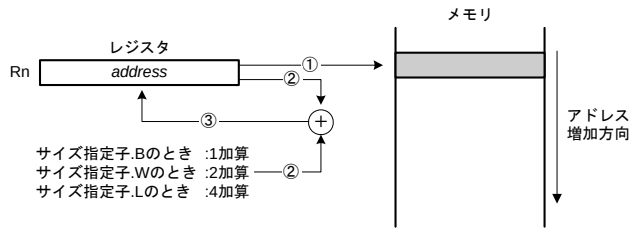
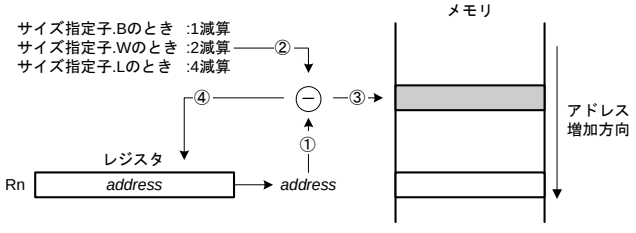
<table>
<tr><td colspan="2">即値</td><td></td></tr>
<tr><td>#IMM:1
#IMM:3
#IMM:4
#UIMM:4
#IMM:5</td><td>・#IMM:1
#IMMで示した1ビット即値が演算の対象となります。このアドレッシングモードは、RACW命令のソースで使用されます。

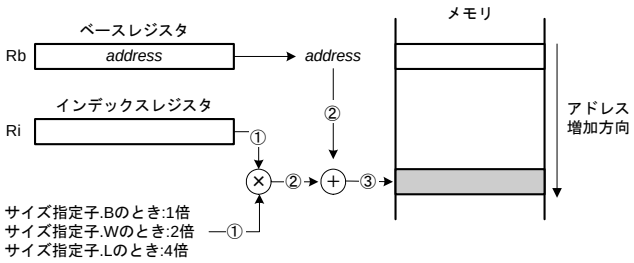
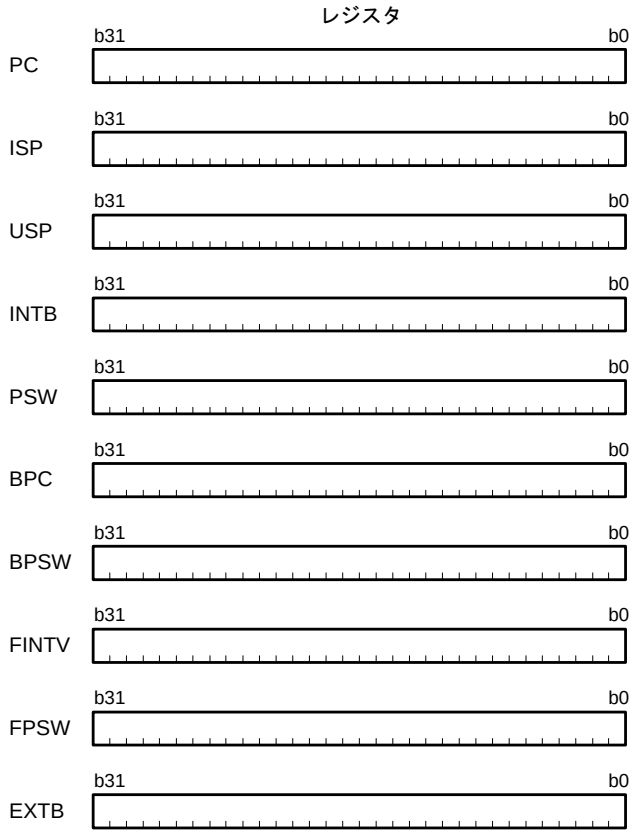
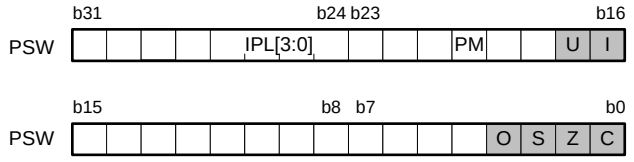
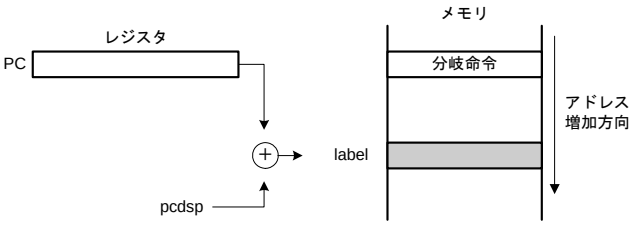
・#IMM:3
#IMMで示した3ビット即値が演算の対象となります。このアドレッシングモードは、ビット操作命令（BCLR、BMCnd、BNOT、BSET、BTST）のビット番号指定で使用されます。

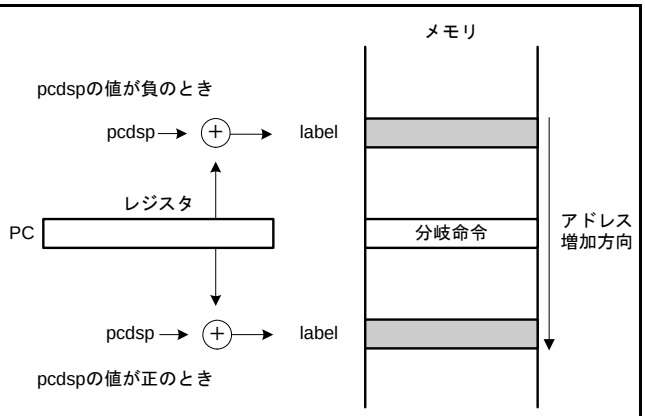
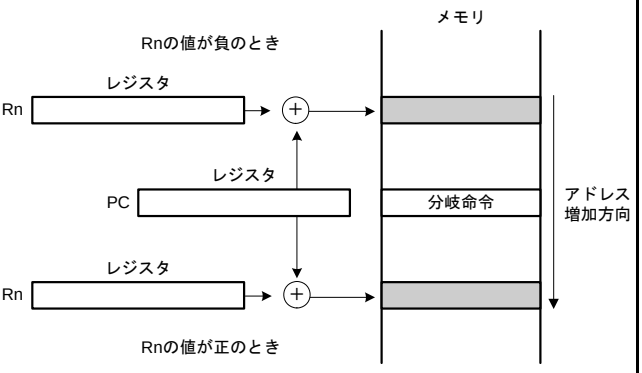
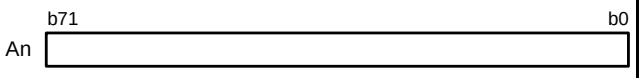
・#IMM:4
#IMMで示した4ビット即値が演算の対象となります。このアドレッシングモードは、MVTIPL命令の割り込み優先レベル指定で使用されます。

・#UIMM:4
#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。

・#IMM:5
#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、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。</td><td> 
</td></tr>
<tr><td>#IMM:8
#SIMM:8
#UIMM:8
#IMM:16
#SIMM:16
#SIMM:24
#IMM:32</td><td>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。</td><td> 
</td></tr>
</table>

レジスタ直接		
Rn (Rn= R0~R15)	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。	
レジスタ間接		
[Rn] (Rn= R0~R15)	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。	
レジスタ相対		
dsp:5[Rn] (Rn= R0~R7) dsp:8[Rn] (Rn= R0~R15) dsp:16[Rn] (Rn= R0~R15)	ディスプレースメント（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～FFFFFFFFhです。dsp:n は、nビット長のディスプレースメントを表します。 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。	
ポストインクリメントレジスタ間接		
[Rn+] (Rn= R0~R15)	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。	
プリデクリメントレジスタ間接		
[-Rn] (Rn= R0~R15)	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。	

<table>
<tr><td colspan="2">インデックス付きレジスタ間接</td><td rowspan="2"> 
</td></tr>
<tr><td>[Ri,Rb]
(Ri,Rb=
R0~R15)</td><td>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。</td></tr>
<tr><td colspan="2">制御レジスタ直接</td><td rowspan="2"> 
</td></tr>
<tr><td>PC
ISP
USP
INTB
PSW
BPC
BPSW
FINTV
FPSW
EXTB</td><td>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。</td></tr>
<tr><td colspan="2">PSW直接</td><td rowspan="2"> 
</td></tr>
<tr><td>C
Z
S
O
I
U</td><td>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。</td></tr>
<tr><td colspan="2">プログラムカウンタ相対</td><td rowspan="2"> 
</td></tr>
<tr><td>pcdsp:3</td><td>分岐距離指定子が“.S”のとき、プログラムカウンタ（PC）にディスプレースメント（pcdsp）の値を符号なしで加算した結果の下位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～FFFFFFFFhです。このアドレッシングモードは、BCnd（Cnd==EQ/Z、NE/NZのみ）、BRA命令で使用されます。</td></tr>
</table>

<table>
<tr><td colspan="2">プログラムカウンタ相対</td><td rowspan="2"> 
</td></tr>
<tr><td>pcdsp:8
pcdsp:16
pcdsp:24</td><td>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、ディスプレースメント（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～FFFFFFFFh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（Cnd==EQ/Z、NE/NZのみ）、BRA、BSR命令で、“.A”のときBRA、BSR命令で使用されます。</td></tr>
<tr><td>Rn
(Rn=
R0~R15)</td><td>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～FFFFFFFFh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。</td><td> 
</td></tr>
<tr><td colspan="2">アキュムレータ直接</td><td rowspan="2"> 
</td></tr>
<tr><td>A0、A1
(A0=ACC0、
A1=ACC1)</td><td>指定したアキュムレータACC0、ACC1が演算の対象となります。</td></tr>
</table>

2.2.1 IMM の範囲

IMM の範囲を**表 2.1** に示します。

「3.4 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～7FFFFFh
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” と表記できます。
3. INT 命令と RTSD 命令の IMM の範囲は、「3.4　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

RXv2 アーキテクチャの命令数は 109 です。1～8 バイト可変長命令形式を採用しています。

RXv2 命令セットについて以下に示します。RXv1 命令セットから新たに追加された命令を「追加」、仕様拡張された命令を「拡張」で示します。

命令一覧表 (1 / 5)

RXv2 での追加 / 拡張	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
	算術/論理演算命令	ABS	絶対値	55	201
		ADC	キャリ付き加算	56	202
		ADD	キャリなし加算	57	203
		AND	論理積	59	205
		CMP	比較	72	218
		DIV	符号付き除算	73	219
		DIVU	符号なし除算	74	221
		EMUL	符号付き乗算	77	223
		EMULU	符号なし乗算	80	224
		MAX	最大値選択	109	236
		MIN	最小値選択	110	237
		MUL	乗算	121	246
		NEG	符号反転	136	255
		NOP	ノーオペレーション	137	255
		NOT	論理反転	138	256
		OR	論理和	139	257
		RMPA	積和演算	157	265
		ROLC	キャリ付き左回転	159	265
		RORC	キャリ付き右回転	160	265
		ROTL	左回転	161	266
		ROTR	右回転	162	266
		SAT	32 ビット符号付き飽和処理	171	268
		SATR	RMPA 命令用 64 ビット符号付き飽和処理	172	269
		SBB	ボロー付き減算	173	269
		SHAR	算術右シフト	177	272
		SHLL	論理/算術左シフト	178	273
		SHLR	論理右シフト	179	274
		SUB	ボローなし減算	186	278
		TST	テスト	191	280
		XOR	排他的論理和	196	283
拡張	浮動小数点演算命令	FADD	浮動小数点加算	82	226
		FCMP	浮動小数点比較	85	227
		FDIV	浮動小数点除算	87	228
拡張		FMUL	浮動小数点乗算	89	229
拡張		FSUB	浮動小数点減算	94	231
追加		FSQRT	浮動小数点平方根	92	230
		FTOI	浮動小数点数→整数変換	97	232
追加		FTOU	浮動小数点数→整数変換	99	232
		ITOF	整数→浮動小数点数変換	102	233
		ROUND	浮動小数点数→整数変換	163	267
追加		UTOF	整数→浮動小数点数変換	192	281

命令一覧表 (2 / 5)

RXv2 での追加 / 拡張	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
	転送命令	MOV		転送	111	238
追加		MOVCO		LIフラグクリア付きストア	115	243
追加		MOVLI		LIフラグセット付きロード	116	243
		MOVU		符号なしデータ転送	117	244
		POP		スタックからレジスタへのデータ復帰	141	258
拡張		POPC		制御レジスタの復帰	142	259
		POPM		複数レジスタの復帰	143	259
		PUSH		スタックへデータ退避	144	260
拡張		PUSHC		制御レジスタの退避	145	261
		PUSHM		複数レジスタの退避	146	261
		REVL		エンディアン変換	155	264
		REVW		エンディアン変換	156	264
		SC*Cnd*	SCGEU	条件設定	174	270
			SCC		174	270
			SCEQ		174	270
			SCZ		174	270
			SCGTU		174	270
			SCPZ		174	270
			SCGE		174	270
			SCGT		174	270
			SCO		174	270
			SCLTU		174	270
			SCNC		174	270
			SCNE		174	270
			SCNZ		174	270
			SCLEU		174	270
			SCN		174	270
			SCLE		174	270
			SCLT		174	270
			SCNO		174	270
拡張		STNZ		条件付き転送	184	276
拡張		STZ		条件付き転送	185	277
		XCHG		交換	195	282

命令一覧表 (3 / 5)

RXv2 での追加 / 拡張	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
	分岐命令	BC*nd*	BGEU	相対条件分岐	62	209
			BC		62	209
			BEQ		62	209
			BZ		62	209
			BGTU		62	209
			BPZ		62	209
			BGE		62	209
			BGT		62	209
			BO		62	209
			BLTU		62	209
			BNC		62	209
			BNE		62	209
			BNZ		62	209
			BLEU		62	209
			BN		62	209
			BLE		62	209
			BLT		62	209
			BNO		62	209
		BRA		相対無条件分岐	66	212
		BSR		相対サブルーチン分岐	69	215
		JMP		無条件分岐	104	234
		JSR		サブルーチン分岐	105	234
		RTS		サブルーチンからの復帰	168	268
		RTSD		スタックフレームの解放とサブルーチンからの復帰	169	268

命令一覧表 (4 / 5)

RXv2 での追加 / 拡張	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
	ビット操作命令	BCLR		ビットクリア	61	207
		BMCnd	BMGEU	条件ビット転送	63	210
			BMC		63	210
			BMEQ		63	210
			BMZ		63	210
			BMGTU		63	210
			BMPZ		63	210
			BMGE		63	210
			BMGT		63	210
			BMO		63	210
			BMLTU		63	210
			BMNC		63	210
			BMNE		63	210
			BMNZ		63	210
			BMLEU		63	210
			BMN		63	210
			BMLE		63	210
			BMLT		63	210
			BMNO		63	210
		BNOT		ビット反転	65	211
		BSET		ビットセット	68	213
		BTST		ビットテスト	70	216
	ストリング操作命令	SCMPU		ストリング比較	175	270
		SMOVB		逆方向ストリング転送	180	274
		SMOVF		順方向ストリング転送	181	275
		SMOVU		ストリング転送	182	275
		SSTR		ストリングストア	183	275
		SUNTIL		ストリングサーチ	187	279
		SWHILE		ストリングサーチ	189	279
	システム操作命令	BRK		無条件トラップ	67	213
		CLRPSW		PSWのフラグ、ビットのクリア	71	217
		INT		ソフトウェア割り込み	101	233
拡張		MVFC		制御レジスタからの転送	130	251
拡張		MVTC		制御レジスタへの転送	134	253
		MVTIPL（特権命令）		割り込み優先レベル設定	135	254
拡張		RTE（特権命令）		例外からの復帰	166	267
拡張		RTFI（特権命令）		高速割り込みからの復帰	167	267
		SETPSW		PSWのフラグ、ビットのセット	176	271
		WAIT（特権命令）		ウェイト	194	282

命令一覧表 (5 / 5)

RXv2 での追加 / 拡張	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
追加	DSP機能命令	EMACA	32ビット積和演算	75	222
追加		EMSBA	32ビット積差演算	76	222
追加		EMULA	32ビット乗算	79	224
拡張		MACHI	上位16ビット積和演算	106	235
追加		MACLH	下位16ビット・上位16ビット積和演算	107	235
拡張		MACLO	下位16ビット積和演算	108	235
追加		MSBHI	上位16ビット積和演算	118	235
追加		MSBLH	下位16ビット・上位16ビット積和演算	119	245
追加		MSBLO	下位16ビット積和演算	120	246
拡張		MULHI	上位16ビット乗算	123	248
追加		MULLH	下位16ビット・上位16ビット乗算	124	248
拡張		MULLO	下位16ビット乗算	125	248
追加		MVFACGU	アキュムレータガードビットからの転送	126	249
拡張		MVFACHI	アキュムレータ上位32ビットからの転送	127	250
追加		MVFACLO	アキュムレータ下位32ビットからの転送	128	250
拡張		MVFACMI	アキュムレータ中央32ビットからの転送	129	251
追加		MVTACGU	アキュムレータガードビットへの転送	131	252
拡張		MVTACHI	アキュムレータ上位32ビットへの転送	132	252
拡張		MVTACLO	アキュムレータ下位32ビットへの転送	133	252
追加		RACL	符号付きアキュムレータ丸め処理	147	262
拡張		RACW	16ビット符号付きアキュムレータ丸め処理	149	262
追加		RDACL	符号付きアキュムレータ丸め処理	151	263
追加		RDACW	16ビット符号付きアキュムレータ丸め処理	153	263

3.2 RXv2 拡張命令セット一覧

RXv2 アーキテクチャでは、RXv1 アーキテクチャから 19 の命令を追加（新規追加命令）し、20 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 RXv2 新規追加命令

表 3.1 に RXv2 で、RXv1 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
浮動小数点演算命令	FSQRT	浮動小数点平方根
	FTOU	浮動小数点数→整数変換
	UTOF	整数→浮動小数点数変換
転送命令	MOVCO	LI フラグクリア付きストア
	MOVLI	LI フラグセット付きロード
DSP 機能命令	EMACA	32 ビット積和演算
	EMSBA	32 ビット積差演算
	EMULA	32 ビット乗算
	MACLH	下位 16 ビット・上位 16 ビット積和演算
	MSBHI	上位 16 ビット積差演算
	MSBLH	下位 16 ビット・上位 16 ビット積差演算
	MSBLO	下位 16 ビット積差演算
	MULLH	下位 16 ビット・上位 16 ビット乗算
	MVFACGU	アキュムレータガードビットからの転送
	MVFACLO	アキュムレータ下位 32 ビットからの転送
	MVTACGU	アキュムレータガードビットへの転送
	RACL	符号付きアキュムレータ丸め処理
	RDACL	符号付きアキュムレータ丸め処理
	RDACW	16 ビット符号付きアキュムレータ丸め処理

3.2.2 RXv2 仕様拡張命令

表3.2にRXv2で、RXv1命令セットから仕様が拡張された命令一覧を示します。

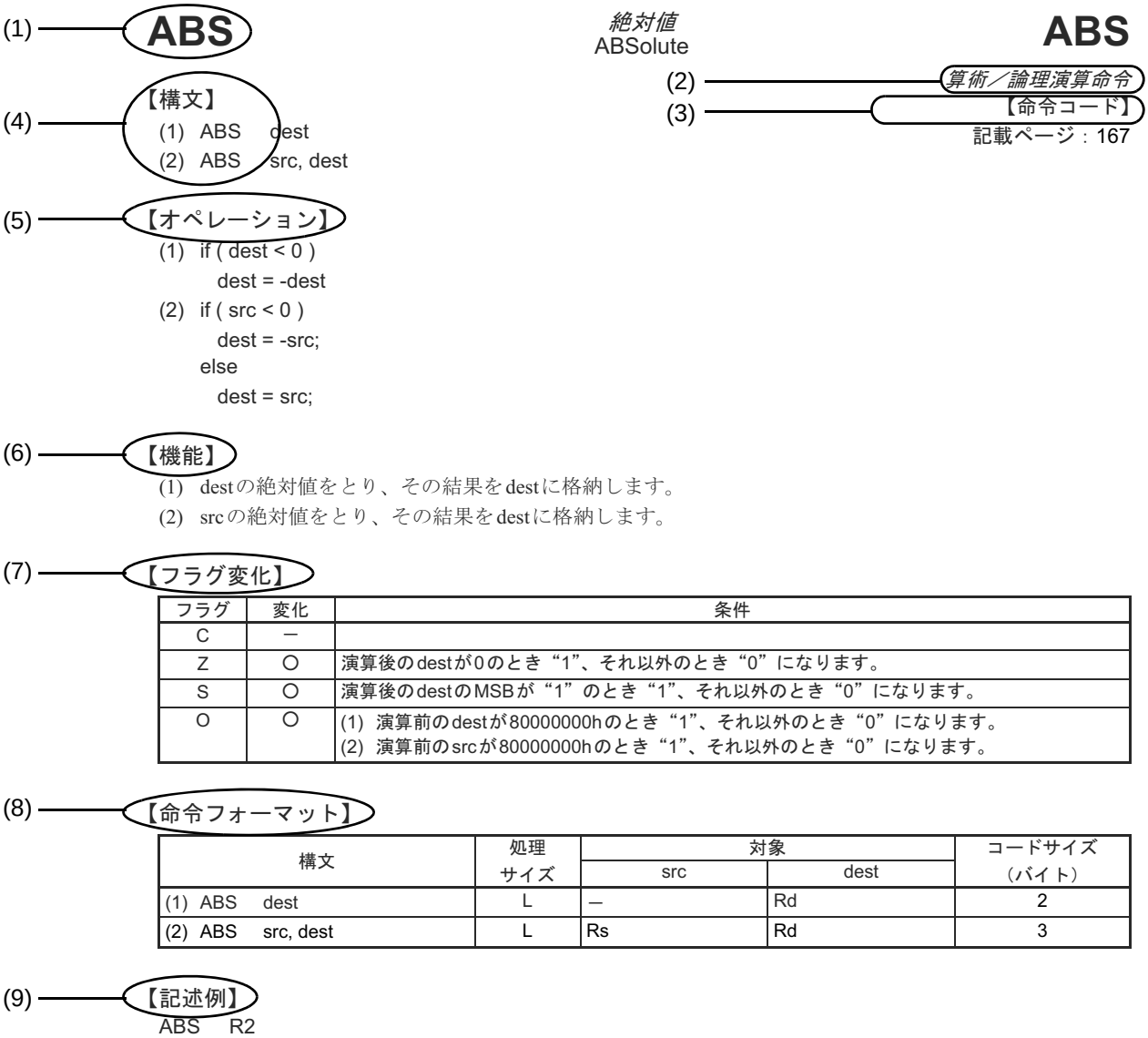
表3.2 仕様が拡張された命令一覧表

分類	ニーモニック	仕様拡張概要
浮動小数点演算命令	FADD	3オペランド (src, src2, dst) を追加し、(Rs, Rs2, Rd) を指定可能
	FMUL	
	FSUB	
転送命令	STNZ	ソースオペランドにレジスタ直接Rnを指定可能
	STZ	
システム操作命令	MVFC	オペランドにEXTBレジスタを指定可能
	MVTC	
	POPC	
	PUSHC	
	RTE	排他制御命令採用に伴い、オペレーションに機能追加 (LIフラグのクリア)
	RTFI	
DSP機能命令	MACHI	オペランドにアキュムレータA0, A1を指定可能 また、アキュムレータは72ビット幅に拡張
	MACLO	
	MULHI	
	MULLO	
	MVFACHI	オペランドにアキュムレータA0, A1を指定可能 アキュムレータは72ビット幅に拡張 また、アキュムレータをイミディエート(IMM:2)で指定したビット左シフト実行後の値がレジスタに読み出される
	MVFACMI	
	MVTACHI	オペランドにアキュムレータA0, A1を指定可能 アキュムレータは72ビット幅に拡張
	MVTACLO	
	RACW	オペランドにアキュムレータA0, A1を指定可能 アキュムレータは72ビット幅に拡張 また、アキュムレータをイミディエート(IMM:1)で指定したビット左シフト実行後の値が丸め操作に反映される

3.3　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。



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。

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

(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）

(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

(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 のレジスタ番号 n を返します。

(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。

(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。

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

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 src	対象 src2	対象 dest	コードサイズ（バイト）
(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 *	—	Rd	3 (memex == UB) 4 (memex != UB)
	L	dsp:16[Rs].memex *	—	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 src	対象 dest *	コードサイズ（バイト）
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 dest	コードサイズ（バイト）
SETPSW dest	flag	2

(a) レジスタ

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

DSP機能命令のアキュムレータの指定のためにA0、A1が指定できます。

(b) 制御レジスタ

RXv2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。

(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

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。

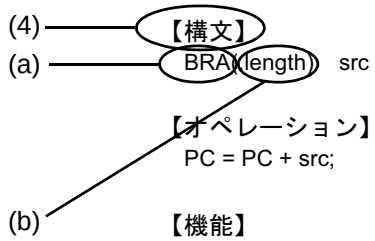
B*Cnd*、BRA、BSR 命令の構文について、以下に BRA 命令を例にあげて示します。

BRA　　*相対無条件分岐*　　**BRA**

BRanch Always　　*分岐命令*

【命令コード】

記載ページ：178



【機能】

- src で示される分岐先に相対分岐します。

【フラグ変化】

- フラグ変化はありません。

【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 ≦ pcdsp ≦ 10	1
	B	pcdsp:8	−128 ≦ pcdsp ≦ 127	2
	W	pcdsp:16	−32768 ≦ pcdsp ≦ 32767	3
	A	pcdsp:24	−8388608 ≦ pcdsp ≦ 8388607	4
	L	Rs	−2147483648 ≦ Rs ≦ 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、ディスプレースメントの値（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。

3.4 命令詳細説明

次ページより RX ファミリの各命令の詳細説明を示します。

ABS

絶対値
ABSolute

ABS

算術/論理演算命令
【命令コード】
記載ページ：201

【構文】
(1) ABS dest
(2) ABS src, dest

【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

算術/論理演算命令
【命令コード】
記載ページ：202

【構文】
ADC src, dest

【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フローが発生したとき“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	○	符号付き演算の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。

【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、ディスプレースメントの値（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

算術/論理演算命令
【命令コード】
記載ページ：203

【構文】

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

【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フローが発生したとき“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	○	符号付き演算の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。

【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)
	L	dsp:16[Rs].memex （注）	—	Rd	4 (memex == UB) 5 (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】

```
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

算術/論理演算命令
【命令コード】
記載ページ：205

【構文】

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

【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】

```
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

ビット操作命令
【命令コード】
記載ページ：207

【構文】
BCLR src, dest

【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≦IMM:3≦7です。
  IMM:5の範囲は、0≦IMM:5≦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

分岐命令
【命令コード】
記載ページ：209

【構文】
BC*nd*(.length) src
o
【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。
- BC*nd* には以下の種類があります。

BC*nd*	条件		式	BC*nd*	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) BC*nd*.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、ディスプレースメントの値（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
```

BM*Cnd*

条件ビット転送
Bit Move Conditionally

BM*Cnd*

ビット操作命令
【命令コード】
記載ページ：210

【構文】
BM*Cnd* src, dest

【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*Cnd*	条件		式	BM*Cnd*	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≦IMM:3≦7です。
  IMM:5の範囲は、0≦IMM:5≦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) BM*Cnd* 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) BM*Cnd* 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

ビット操作命令
【命令コード】
記載ページ：211

【構文】
BNOT　src, dest

【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≦IMM:3≦7です。
  IMM:5の範囲は、0≦IMM:5≦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：212

【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 ≦ pcdsp ≦ 10	1
	B	pcdsp:8	−128 ≦ pcdsp ≦ 127	2
	W	pcdsp:16	−32768 ≦ pcdsp ≦ 32767	3
	A	pcdsp:24	−8388608 ≦ pcdsp ≦ 8388607	4
	L	Rs	−2147483648 ≦ Rs ≦ 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、ディスプレースメントの値（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

システム操作命令
【命令コード】
記載ページ：213

【構文】
BRK

【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

ビット操作命令
【命令コード】
記載ページ：213

【構文】
BSET　src, dest

【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≦IMM:3≦7です。
  IMM:5の範囲は、0≦IMM:5≦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

分岐命令
【命令コード】
記載ページ：215

【構文】
BSR(.length) src

【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 ≦ pcdsp ≦ 32767	3
	A	pcdsp:24	−8388608 ≦ pcdsp ≦ 8388607	4
	L	Rs	−2147483648 ≦ Rs ≦ 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、ディスプレースメントの値（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

ビット操作命令
【命令コード】
記載ページ：216

【構文】
BTST　src, src2

【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≦IMM:3≦7です。
  IMM:5の範囲は、0≦IMM:5≦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：217

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

【記述例】
CLRPSW C
CLRPSW Z

CMP

比較
CoMPare

CMP

算術/論理演算命令
【命令コード】
記載ページ：218

【構文】
CMP src, src2

【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フローが発生しなかったとき“1”、それ以外のとき“0”になります。
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	○	符号付き演算の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。

【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】
```
CMP   #7, R2
CMP   R1, R2
CMP   [R1], R2
```

DIV

符号付き除算
DIVide

DIV

算術/論理演算命令
【命令コード】
記載ページ：219

【構文】
DIV src, dest

【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÷(−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】
```
DIV    #10, R2
DIV    R1, R2
DIV    [R1], R2
DIV    3[R1].B, R2
```

DIVU

符号なし除算
DIVide Unsigned

DIVU

算術/論理演算命令
【命令コード】
記載ページ：221

【構文】
DIVU　src, dest

【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	
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】
```
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

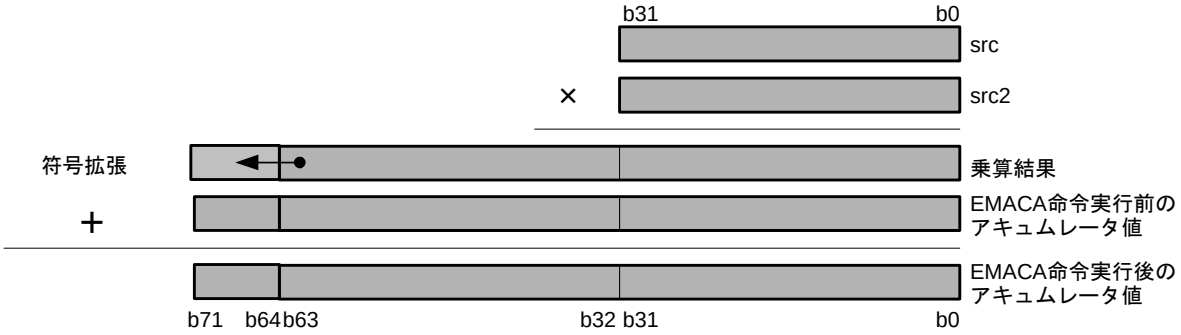
EMACA

DSP 機能命令
【命令コード】
記載ページ：222

【構文】
EMACA　src, src2, Adest

【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

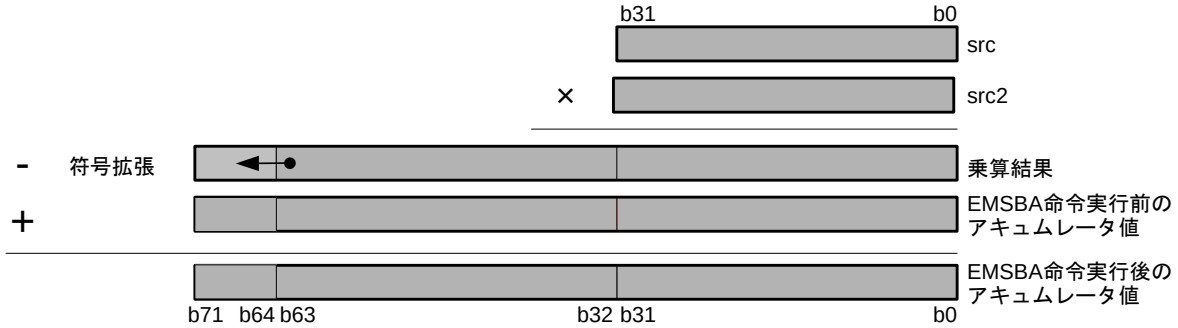
EMSBA

DSP 機能命令
【命令コード】
記載ページ：222

【構文】
EMSBA　src, src2, Adest

【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

算術/論理演算命令
【命令コード】
記載ページ：223

【構文】
EMUL　src, dest

【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】

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

EMULA

32 ビット乗算
Extended MULtiply to Accumulater

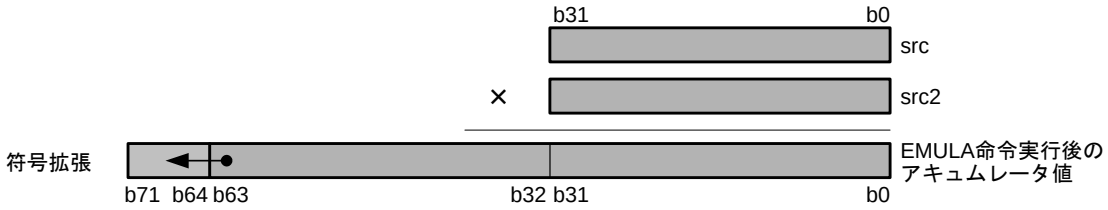
EMULA

DSP 機能命令
【命令コード】
記載ページ：224

【構文】
EMULA src, src2, Adest

【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：224

【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
		src	dest	（バイト）
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】

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

FADD

浮動小数点加算
Floating-point ADD

FADD

浮動小数点演算命令
【命令コード】
記載ページ：226

【構文】

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

【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、-∞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。-∞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) 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、ディスプレースメントの値（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。

【発生例外】

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】

- DN=0 と DN=1 のときについて、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				− ∞	非実装処理	QNaN	無効演算
	+0		+0	（注）					
	−0		（注）	−0					
	+ ∞				+ ∞	無効演算			
	− ∞				無効演算	− ∞			
	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
dest または src2	正規化数	加算	正規化数					
	+0、+ 非正規化数	正規化数	+0	（注）		− ∞		
	−0、− 非正規化数		（注）	−0				
	+ ∞				+ ∞	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

浮動小数点比較
Floating-point CoMPare

FCMP

浮動小数点演算命令
【命令コード】
記載ページ：227

【構文】
FCMP src, src2

【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、ディスプレースメントの値（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。

【発生例外】

非実装処理
無効演算

【記述例】

```
FCMP    R1, R2
FCMP    [R1], R2
```

【動作補足説明】

- DN=0とDN=1のときについて、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			<	>			
	+0		=						
	−0								
	+ ∞	>			=				
	− ∞	<				=			
	非正規化数	非実装処理							
	QNaN	順序化不能							
	SNaN	無効演算（順序化不能）							

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			<	>		
	+0、+ 非正規化数		=					
	−0、− 非正規化数							
	+ ∞	>			=			
	− ∞	<				=		
	QNaN	順序化不能						
	SNaN	無効演算（順序化不能）						

FDIV

浮動小数点除算
Floating-point DIVide

FDIV

浮動小数点演算命令
【命令コード】
記載ページ：228

【構文】
FDIV　src, dest

【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、ディスプレースメントの値（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。

【発生例外】
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】
- DN=0 と DN=1 のときについて、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				
	+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			
	+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

浮動小数点乗算
Floating-point MULtiply

FMUL

浮動小数点演算命令
【命令コード】
記載ページ：229

【構文】

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

【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． 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、ディスプレースメントの値（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。

【発生例外】

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】

- DN=0 と DN=1 のときについて、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	乗算			∞				
	+0		+0	−0	無効演算				
	−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 または src2	正規化数	乗算			∞			
	+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							無効演算

FSQRT

浮動小数点平方根
Floating-point SQuare RooT

FSQRT

浮動小数点演算命令
【命令コード】
記載ページ：230

【構文】
FSQRT　src, dest

【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、ディスプレースメントの値（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。

【発生例外】
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】
- DN=0とDN=1のときについて、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1.6 QNaN生成規則」に対応

FSUB

浮動小数点減算
Floating-point SUBtract

FSUB

浮動小数点演算命令
【命令コード】
記載ページ：231

【構文】

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

【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、−∞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。−∞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、ディスプレースメントの値（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。

【発生例外】

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】

- DN=0 と DN=1 のときについて、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			− ∞	+ ∞			
	+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							
	SNaN							

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

FTOI

浮動小数点数→整数変換
Float TO Integer

FTOI

浮動小数点演算命令
【命令コード】
記載ページ：232

【構文】
FTOI　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、ディスプレースメントの値（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。

【発生例外】
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】
- DN=0とDN=1のときについて、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ビット＝1で無効演算発生時：変化なし 上記以外：7FFFFFFFh	無効演算
	127≧指数部≧31		
	30≧指数部≧−126	00000000h～7FFFFF80h	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
	30≧指数部≧−126	00000000h～80000080h	なし（注1）
	127≧指数部≧31	EVビット＝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ビット＝1で無効演算発生時：変化なし 上記以外： 符号ビット＝0：7FFFFFFFh 符号ビット＝1：80000000h	無効演算
	SNaN		

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ビット＝1で無効演算発生時：変化なし 上記以外：7FFFFFFFh	無効演算
	127≧指数部≧31		
	30≧指数部≧−126	00000000h～7FFFFF80h	なし（注1）
	+0、+非正規化数	00000000h	なし
src＜0	−0、−非正規化数		
	30≧指数部≧−126	00000000h～80000080h	なし（注1）
	127≧指数部≧31	EVビット＝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ビット＝1で無効演算発生時：変化なし 上記以外： 符号ビット＝0：7FFFFFFFh 符号ビット＝1：80000000h	無効演算
	SNaN		

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

浮動小数点数→整数変換
Float TO Unsigned integer

FTOU

浮動小数点演算命令
【命令コード】
記載ページ：232

【構文】
FTOU src, dest

【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”、それ以外のとき“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	
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、ディスプレースメントの値（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。

【発生例外】
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】

- DN=0とDN=1のときについて、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ビット＝1で無効演算発生時：変化なし 上記以外：FFFFFFFFh	無効演算
	127≧指数部≧32		
	31≧指数部≧−126	00000000h～FFFFFF00h	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ビット＝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ビット＝1で無効演算発生時：変化なし 上記以外： 最上位ビット＝0：FFFFFFFFh 最上位ビット＝1：00000000h	無効演算
	SNaN		

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ビット＝1で無効演算発生時：変化なし 上記以外：FFFFFFFFh	無効演算
	127≧指数部≧32		
	31≧指数部≧−126	00000000h～FFFFFF00h	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ビット＝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ビット＝1で無効演算発生時：変化なし 上記以外： 符号ビット＝0：FFFFFFFFh 符号ビット＝1：00000000h	無効演算
	SNaN		

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

システム操作命令
【命令コード】
記載ページ：233

【構文】
INT src

【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≦src≦255です。
- スーパ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、スタックに退避されます。

【命令フォーマット】

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

【記述例】
```
INT #0
```

ITOF

整数→浮動小数点数変換
Integer TO Floating-point

ITOF

浮動小数点演算命令
【命令コード】
記載ページ：233

【構文】
ITOF　src, dest

【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	○	“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	○	“0”になります。
FV	—	
FO	—	
FZ	—	
FU	—	
FX	○	精度異常が発生したとき“1”、それ以外の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。

【発生例外】

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

分岐命令
【命令コード】
記載ページ：234

【構文】
JMP　src

【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
	src	（バイト）
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：234

【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

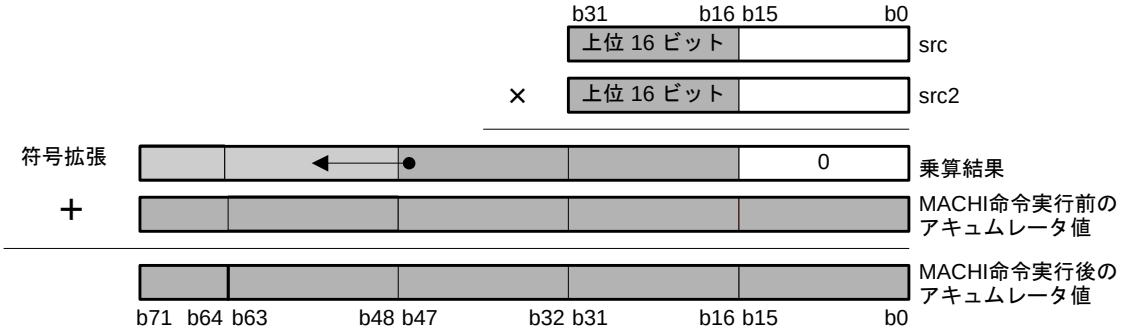
MACHI

DSP 機能命令
【命令コード】
記載ページ：235

【構文】
MACHI src, src2, Adest

【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

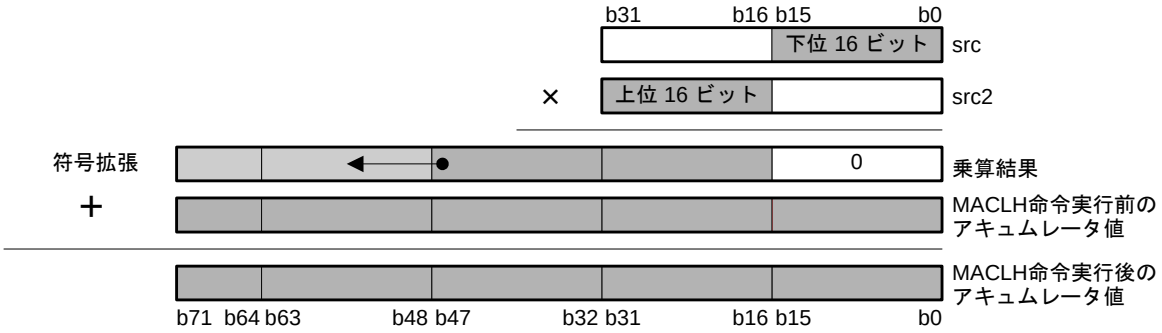
MACLH

DSP 機能命令
【命令コード】
記載ページ：235

【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

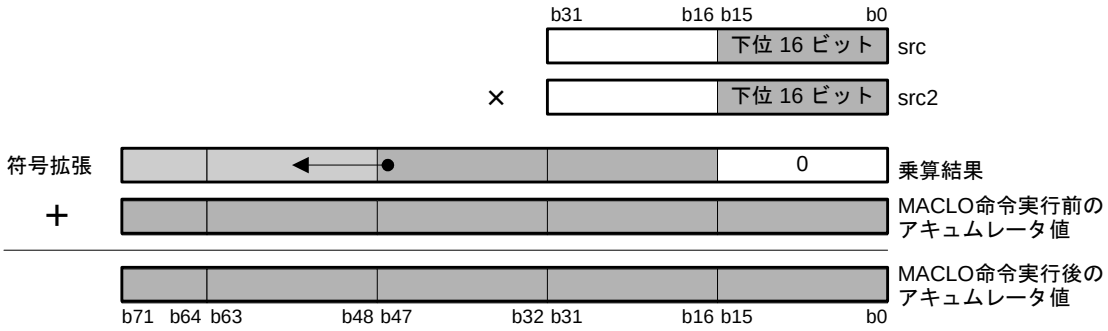
MACLO

DSP 機能命令
【命令コード】
記載ページ：235

【構文】
MACLO　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;
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

算術/論理演算命令
【命令コード】
記載ページ：236

【構文】
MAX　src, dest

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

MIN

最小値選択
MINimum value select

MIN

算術/論理演算命令
【命令コード】
記載ページ：237

【構文】
MIN　src, dest

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

MOV

転送
MOVe

MOV

転送命令
【命令コード】
記載ページ：238

【構文】
MOV.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
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：243

【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：243

【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：244

【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

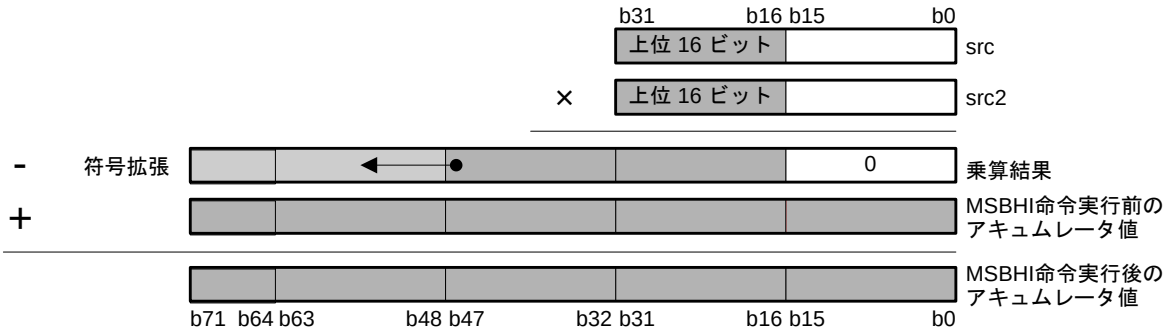
MSBHI

DSP 機能命令
【命令コード】
記載ページ：245

【構文】
MSBHI src, src2, Adest

【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

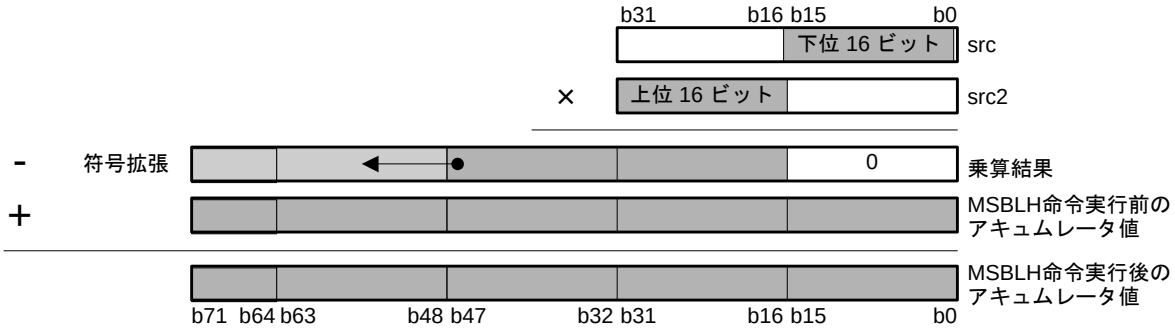
MSBLH

DSP 機能命令
【命令コード】
記載ページ：245

【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

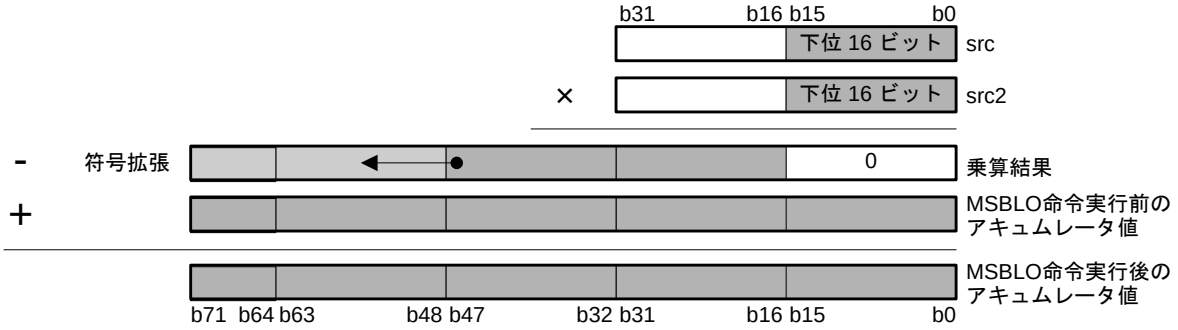
MSBLO

DSP 機能命令
【命令コード】
記載ページ：246

【構文】
MSBLO　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;
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	
MSBLO　src, src2, Adest	Rs	Rs2	A0, A1	3

【記述例】
```
MSBLO　R1, R2, A1
```

MUL

乗算
MULtiply

MUL

算術/論理演算命令
【命令コード】
記載ページ：246

【構文】
(1) MUL src, dest
(2) MUL src, src2, dest

【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】

```
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

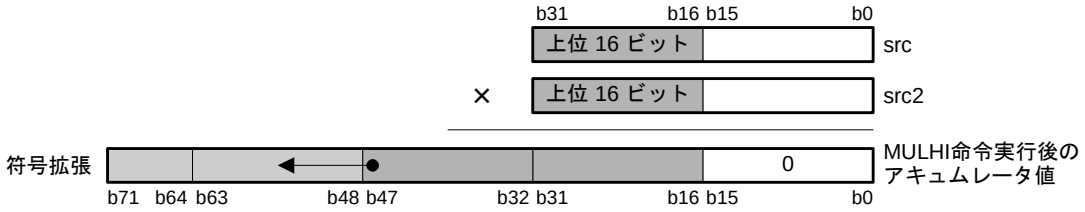
MULHI

DSP 機能命令
【命令コード】
記載ページ：248

【構文】
MULHI　src, src2, Adest

【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

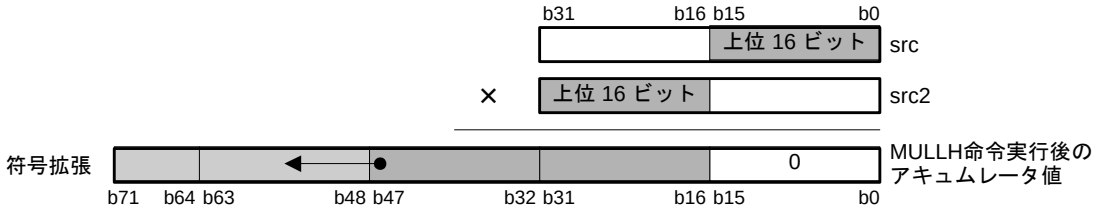
MULLH

DSP 機能命令
【命令コード】
記載ページ：248

【構文】
MULLH 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 = (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

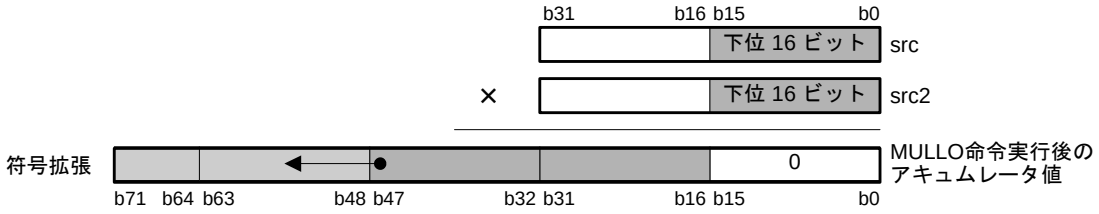
MULLO

DSP機能命令
【命令コード】
記載ページ：249

【構文】
MULLO　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;
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

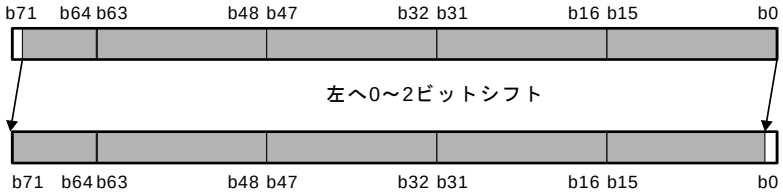
DSP 機能命令
【命令コード】
記載ページ：249

【構文】
MVFACGU src, Asrc, dest

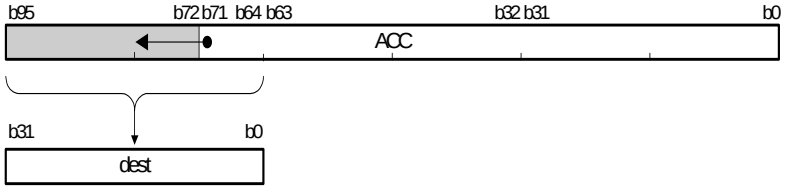
【オペレーション】
```
signed 72bit tmp;
tmp = (signed 72bit) Asrc << src;
dest = (signed long) (tmp >> 64);
```

【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. アキュムレータの最上位 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

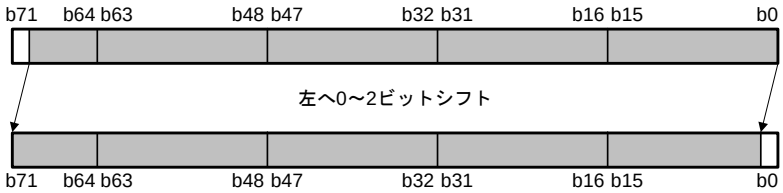
DSP機能命令
【命令コード】
記載ページ：250

【構文】
MVFACHI src, Asrc, dest

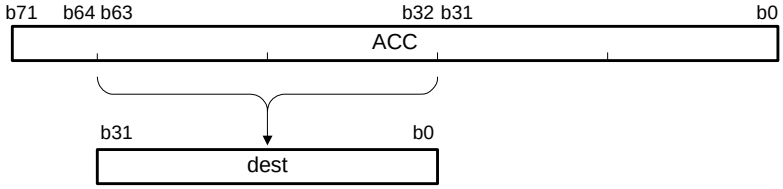
【オペレーション】
```
signed 72bit tmp;
tmp = (signed 72bit) Asrc << src;
dest = (signed long) (tmp >> 32);
```

【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. アキュムレータの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

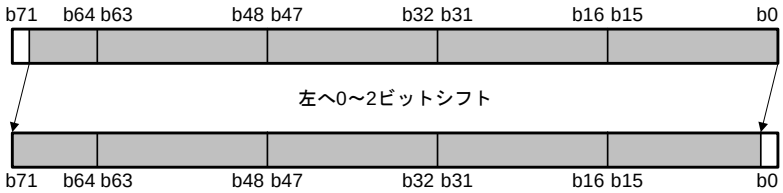
DSP機能命令
【命令コード】
記載ページ：250

【構文】
MVFACLO src, Asrc, dest

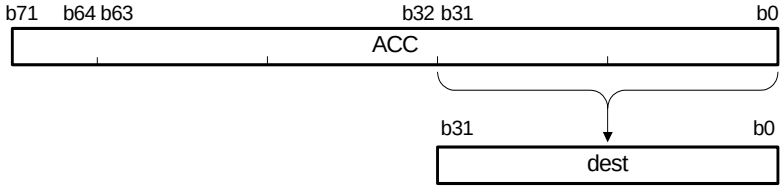
【オペレーション】
```
signed 72bit tmp;
tmp = (signed 72bit) Asrc << src;
dest = (signed long) tmp;
```

【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. アキュムレータの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

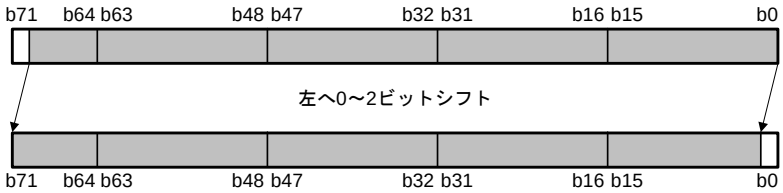
DSP機能命令
【命令コード】
記載ページ：251

【構文】
MVFACMI src, Asrc, dest

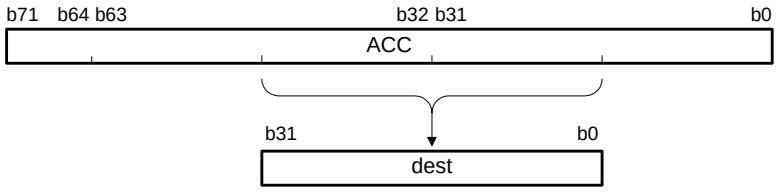
【オペレーション】
```
signed 72bit tmp;
tmp = (signed 72bit) Asrc << src;
dest = (signed long) (Asrc >> 16);
```

【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. アキュムレータの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：251

【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

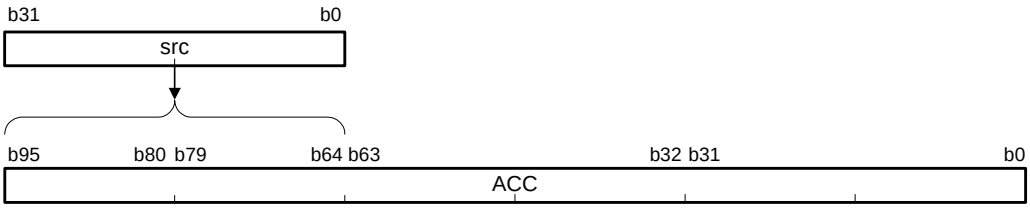
MVTACGU

DSP機能命令
【命令コード】
記載ページ：252

【構文】
MVTACGU src, Adest

【オペレーション】
Adest = (Adest & 00FFFFFFFFFFFFFFh) | ((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

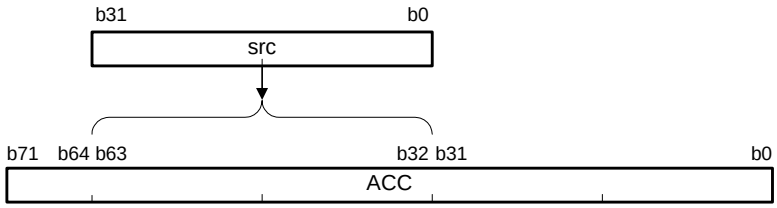
MVTACHI

DSP機能命令
【命令コード】
記載ページ：252

【構文】
MVTACHI src, Adest

【オペレーション】
Adest = (Adest & FF 0000 0000 FFFF FFFFh) | ((signed 72bit) src << 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

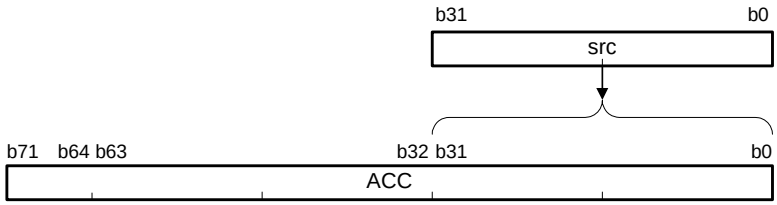
MVTACLO

DSP機能命令
【命令コード】
記載ページ：252

【構文】
MVTACLO src, Adest

【オペレーション】
Adest = (Adest & FFFFFFFFFF00000000h) | (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

システム操作命令
【命令コード】
記載ページ：253

【構文】
MVTC　src, dest

【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　#0FFFFF000h, 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：254

【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≦src≦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

算術/論理演算命令
【命令コード】
記載ページ：255

【構文】
(1) NEG dest
(2) NEG src, dest

【オペレーション】
(1) dest = -dest;
(2) dest = -src;

【機能】
(1) destを符号反転し（2の補数をとり）、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	○	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

算術/論理演算命令
【命令コード】
記載ページ：255

【構文】
NOP

【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

算術/論理演算命令
【命令コード】
記載ページ：256

【構文】
(1) NOT dest
(2) NOT src, dest

【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

算術/論理演算命令
【命令コード】
記載ページ：257

【構文】

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

【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：258

【構文】
POP dest

【オペレーション】
```
tmp = *SP;
SP = SP + 4;
dest = tmp;
```

【機能】
- スタックから復帰したデータを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：259

【構文】
POPC dest

【オペレーション】
```
tmp = *SP;
SP = SP + 4;
dest = tmp;
```

【機能】
- スタックから復帰したデータを、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

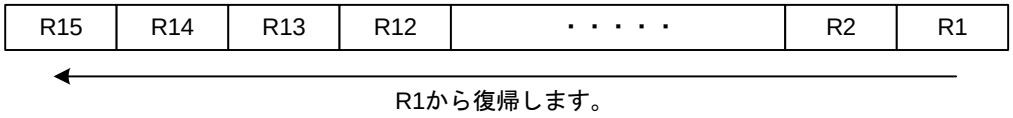
POPM

転送命令
【命令コード】
記載ページ：259

【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します。
- 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。
- スタックから復帰す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

スタックへデータ退避
PUSH data onto the stack

PUSH

転送命令
【命令コード】
記載ページ：260

【構文】
PUSH.size src

【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をスタックに退避します。
- 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します。
- 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、ディスプレースメントの値（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：261

【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。
- srcにPCを指定した場合、本命令の番地をスタックに退避します。

【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）	
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

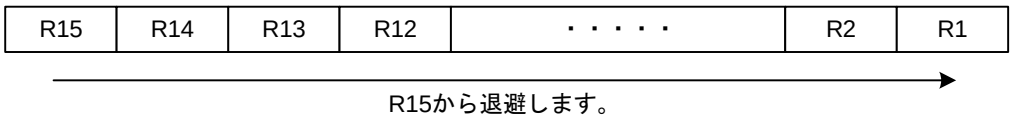
PUSHM

転送命令
【命令コード】
記載ページ：261

【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します。
- 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。
- スタックに退避す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：262

【構文】
RACL　src, Adest

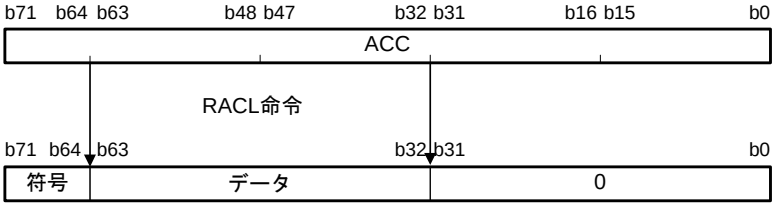
【オペレーション】
```
signed 72bit tmp;
signed 73bit tmp73;

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

if (tmp73 > (signed 73bit) 0007FFFFFFF00000000h)
   Adest = 007FFFFFFF00000000h;
else if (tmp73 < (signed 73bit) 1FF8000000000000000h)
   Adest = FF8000000000000000h;
else
   Adest = tmp & FFFFFFFFFF00000000h;
```

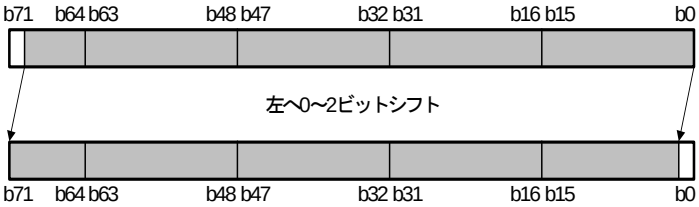
【機能】

- 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。

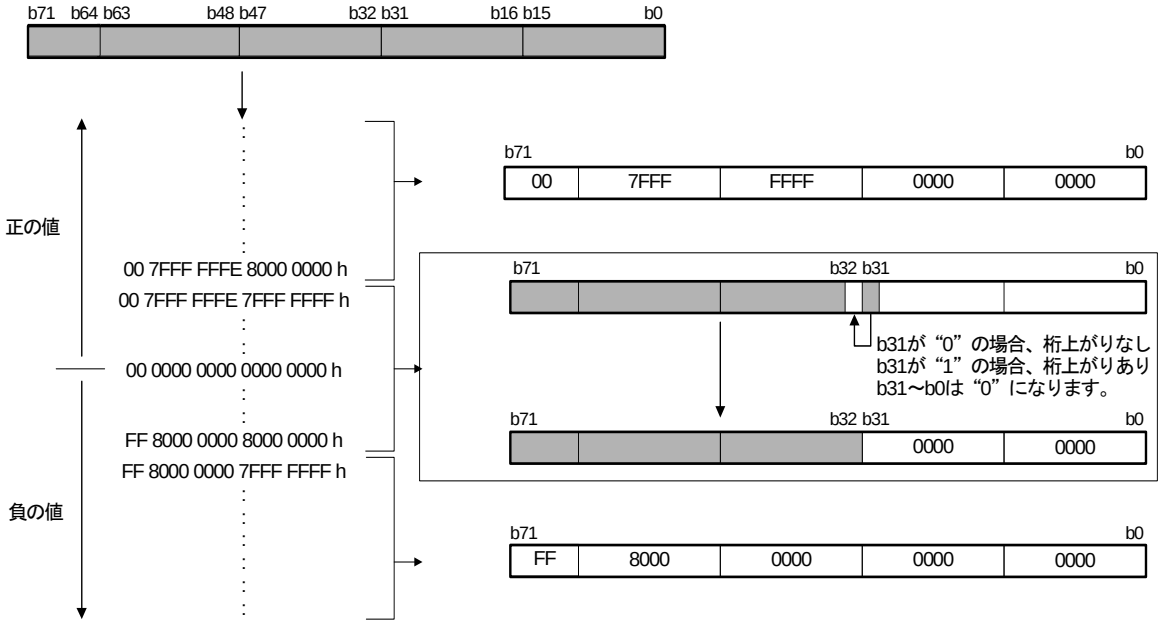


- RACL命令は、以下のような手順で実行されます。

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	
RACL src	#IMM:1 (注) (IMM:1 = 1～2)	3

注. 弊社の「RX ファミリ アセンブラ」では、即値（IMM:1）は、1～2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
RACL   #2
```

RACW

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

RACW

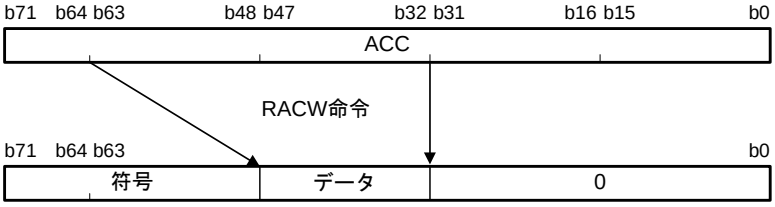
DSP機能命令
【命令コード】
記載ページ：262

【構文】
RACW　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) 00000007FFF00000000h)
   Adest = 0000007FFF00000000h;
else if (tmp73 < (signed 73bit) 1FFFFFF800000000000h)
   Adest = FFFFFF800000000000h;
else
   Adest = tmp & FFFFFFFFFF00000000h;
```

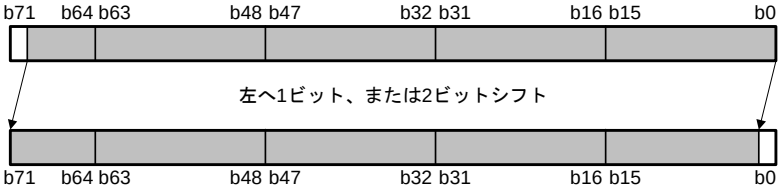
【機能】

- 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。

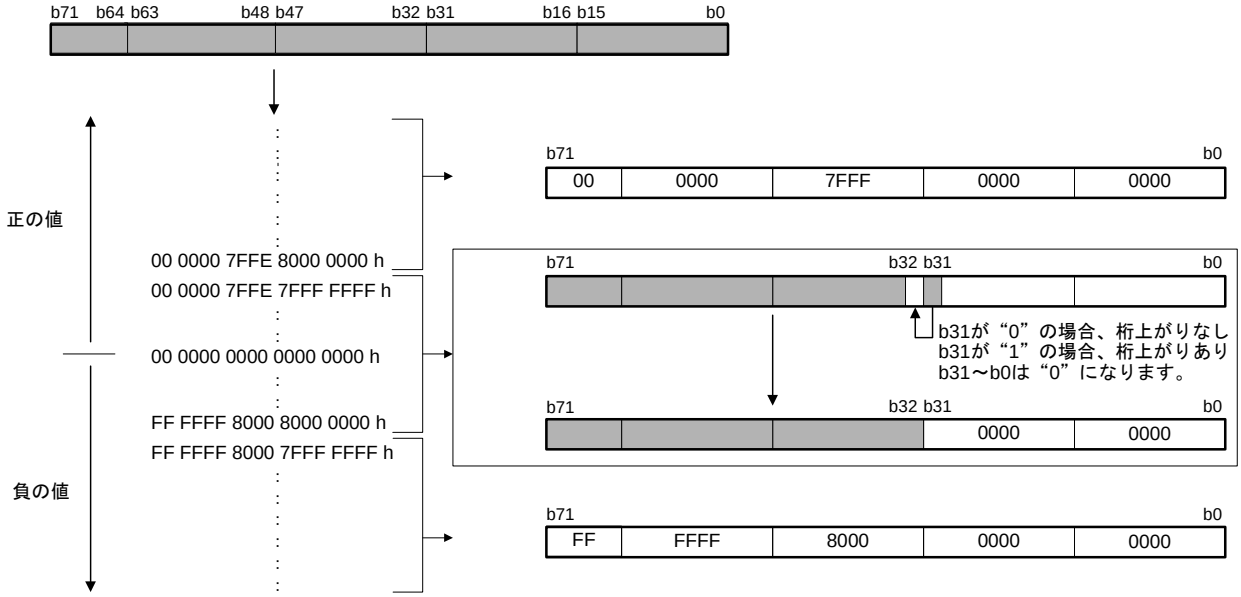


- RACW命令は、以下のような手順で実行されます。

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

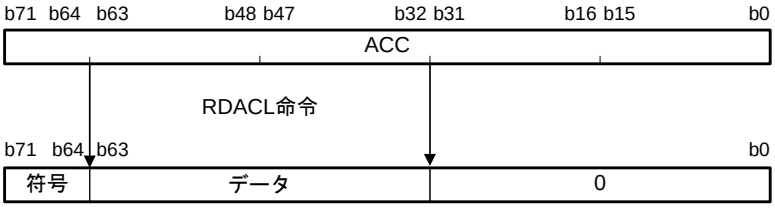
DSP機能命令
【命令コード】
記載ページ：263

【構文】
RDACL　src, Adest

【オペレーション】
```
signed 72bit tmp;
tmp = (signed 72bit) Adest << src;
if (tmp > (signed 72bit) 007FFFFFFF00000000h)
   Adest = 007FFFFFFF00000000h;
else if (tmp < (signed 72bit) FF8000000000000000h)
   Adest = FF8000000000000000h;
else
   Adest = tmp & FFFFFFFFFF00000000h;
```

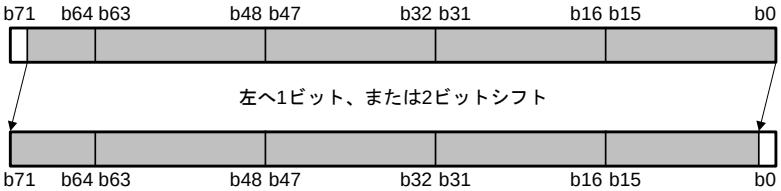
【機能】

- 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。

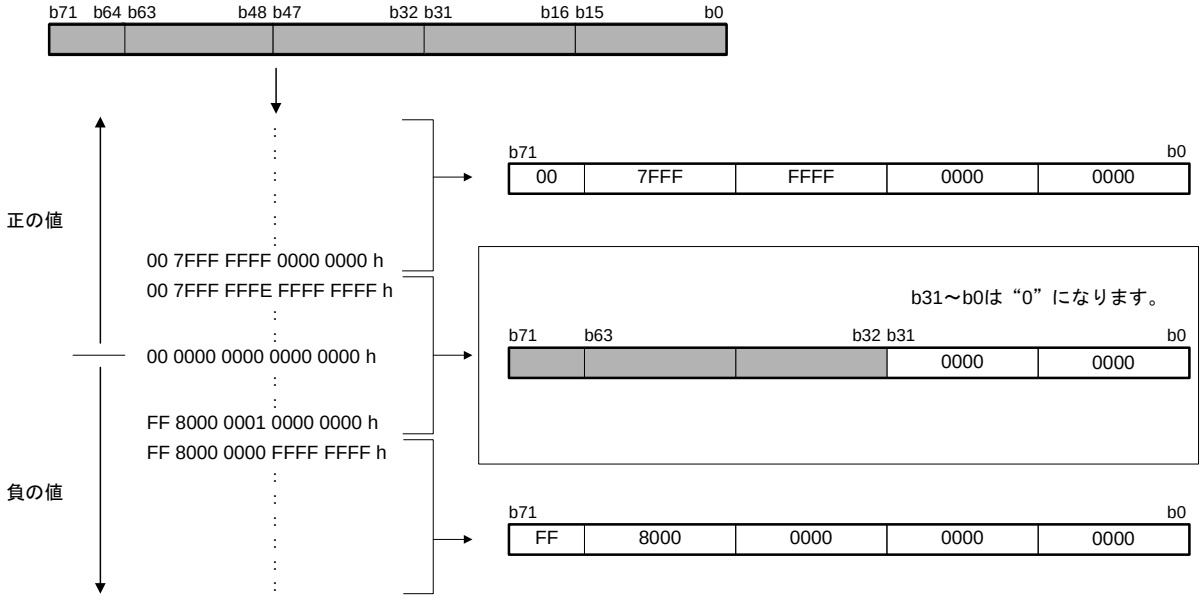


- RDACL命令は、以下のような手順で実行されます。

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

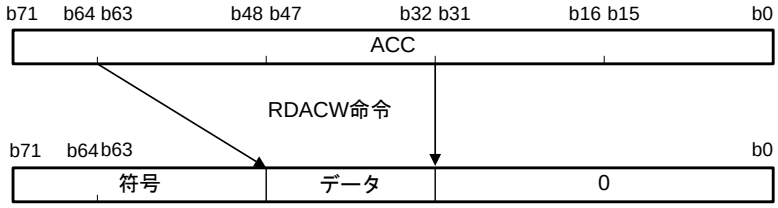
DSP機能命令
【命令コード】
記載ページ：263

【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) FFFFFF800000000000h)
   Adest = FFFFFF800000000000h;
else
   Adest = tmp & FFFFFFFFFF00000000h;
```

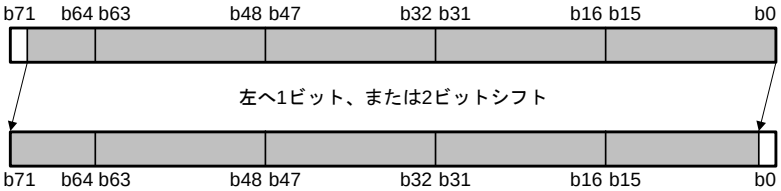
【機能】

- 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。

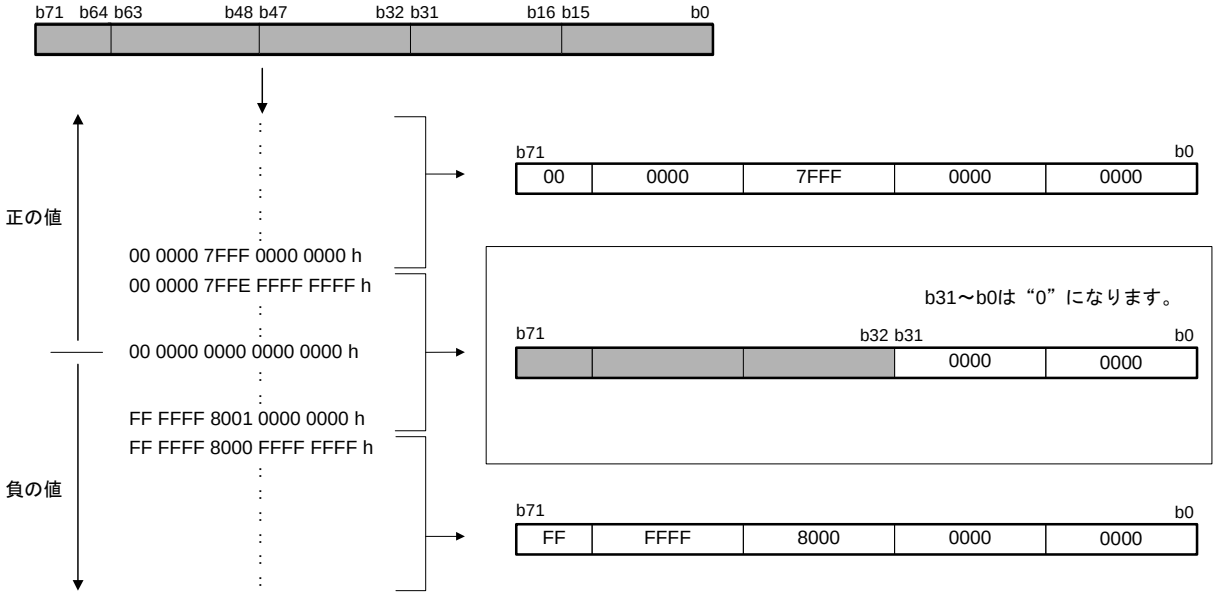


- RDACW命令は、以下のような手順で実行されます。

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

転送命令
【命令コード】
記載ページ：264

【構文】
REVL　src, dest

【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

REVW

エンディアン変換
REVerse Word data

REVW

転送命令
【命令コード】
記載ページ：264

【構文】
REVW　src, dest

【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	
REVW　src, dest	Rs	Rd	3

【記述例】
REVW　R1, R2

RMPA

積和演算
Repeated MultiPly and Accumulate

RMPA

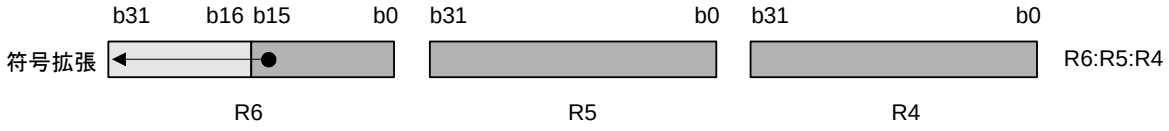
算術/論理演算命令
【命令コード】
記載ページ：265

【構文】
RMPA.size

【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/復帰してください。
- 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”、それ以外のとき“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”、それ以外のとき“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

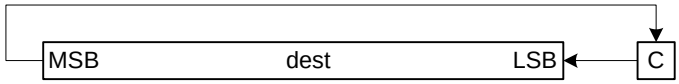
ROLC

算術/論理演算命令
【命令コード】
記載ページ：265

【構文】
ROLC　dest

【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

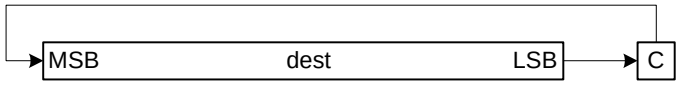
RORC

算術/論理演算命令
【命令コード】
記載ページ：265

【構文】
RORC dest

【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 dest	コードサイズ （バイト）
RORC dest	L	Rd	2

【記述例】
```
RORC R1
```

ROTL

左回転
ROTate Left

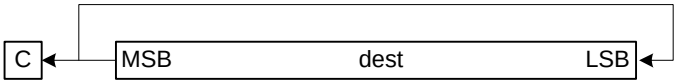
ROTL

算術/論理演算命令
【命令コード】
記載ページ：266

【構文】
ROTL　src, dest

【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分だけ左回転します。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≦src≦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	
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

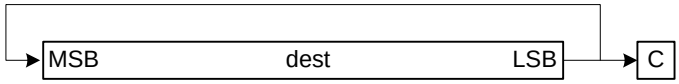
ROTR

算術/論理演算命令
【命令コード】
記載ページ：266

【構文】
ROTR　src, dest

【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分だけ右回転します。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≦src≦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	
ROTR　src, dest	L	#IMM:5	Rd	3
	L	Rs	Rd	3

【記述例】
```
ROTR　#1, R1
ROTR　R1, R2
```

ROUND

浮動小数点数→整数変換
ROUND floating-point to integer

ROUND

浮動小数点演算命令
【命令コード】
記載ページ：267

【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、ディスプレースメントの値（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。

【発生例外】

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】

- DN=0 と DN=1 のときについて、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 ビット=1 で無効演算発生時：変化なし 上記以外：7FFFFFFFh	無効演算
	127 ≧ 指数部 ≧ 31		
	30 ≧ 指数部 ≧ −126	00000000h ～ 7FFFFF80h	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
	30 ≧ 指数部 ≧ −126	00000000h ～ 80000080h	なし（注 1）
	127 ≧ 指数部 ≧ 31	EV ビット=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 ビット=1 で無効演算発生時：変化なし 上記以外： 符号ビット= 0：7FFFFFFFh 符号ビット= 1：80000000h	無効演算
	SNaN		

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ビット=1で無効演算発生時：変化なし 上記以外：7FFFFFFFh	無効演算
	127≧指数部≧31		
	30≧指数部≧−126	00000000h～7FFFFF80h	なし（注1）
	+0、+非正規化数	00000000h	なし
src < 0	−0、−非正規化数		
	30≧指数部≧−126	00000000h～80000080h	なし（注1）
	127≧指数部≧31	EVビット=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ビット=1で無効演算発生時：変化なし 上記以外： 符号ビット=0：7FFFFFFFh 符号ビット=1：80000000h	無効演算
	SNaN		

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

システム操作命令
【命令コード】
記載ページ：267

【構文】
RTE

【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したPCとPSWを復帰し、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

システム操作命令
【命令コード】
記載ページ：267

【構文】
RTFI

【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したPCとPSWを、それぞれBPC、BPSWから復帰し、高速割り込みハンドラから戻ります。
- 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

分岐命令
【命令コード】
記載ページ：268

【構文】
RTS

【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：268

【構文】

(1) RTSD src
(2) RTSD src, dest-dest2

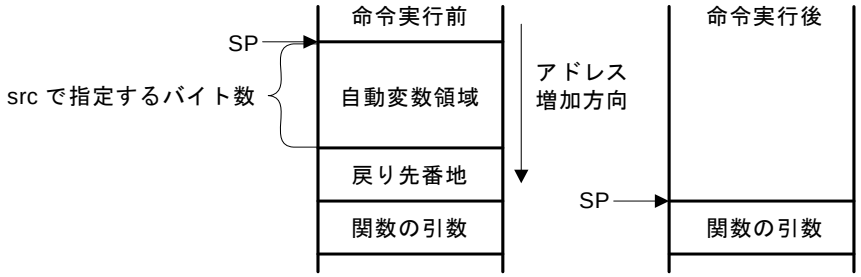
【オペレーション】

(1)
```
SP = SP + src;
PC = *SP;
SP = SP + 4;
```

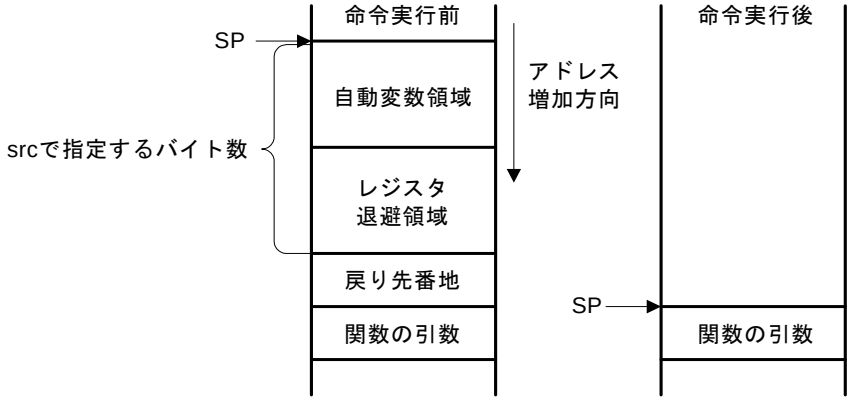
(2)
```
signed char i;
SP = SP + ( src - ( register_num(dest2) - register_num(dest) +1 ) * 4 );
for ( i = register_num(dest); i <= register_num(dest2); i++ ) {
  tmp = *SP;
  SP = SP + 4;
  register(i) = tmp;
}
PC = *SP;
SP = SP + 4;
```

【機能】

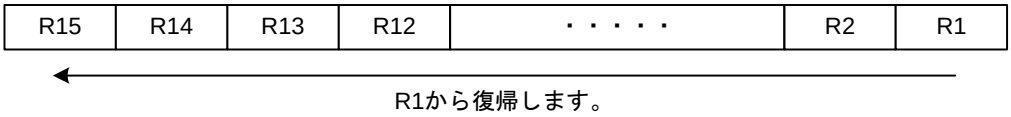
(1) スタックフレームを解放後、サブルーチンから復帰します。
- srcがスタックフレーム（自動変数領域）のサイズになるように指定してください。



(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します。
- 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。
- スタックから復帰す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

算術/論理演算命令
【命令コード】
記載ページ：268

【構文】
SAT dest

【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処理
SATuRate signed 64-bit data for RMPA

SATR

算術/論理演算命令
【命令コード】
記載ページ：269

【構文】
SATR

【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処理を行います。
- 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

算術/論理演算命令
【命令コード】
記載ページ：269

【構文】
SBB src, dest

【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フローが発生しなかったとき“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	○	符号付き演算の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。

【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、ディスプレースメントの値（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

SC*Cnd*

条件設定
Store Condition Conditionally

SC*Cnd*

転送命令
【命令コード】
記載ページ：270

【構文】
SC*Cnd*.size dest

【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。
- SC*Cnd*には次の種類があります。

SC*Cnd*	条件		式	SC*Cnd*	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 dest	コードサイズ（バイト）
SC*Cnd*.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、ディスプレースメントの値（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

ストリング操作命令
【命令コード】
記載ページ：270

【構文】
SCMPU

【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/復帰してください。

【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”、それ以外のとき“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	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：271

【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 dest	コードサイズ（バイト）
SETPSW dest	flag	2

【記述例】
SETPSW C
SETPSW Z

SHAR

算術右シフト
SHift Arithmetic Right

SHAR

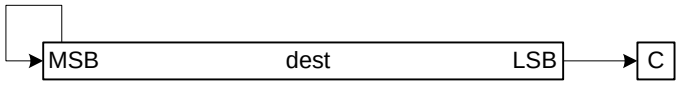
算術/論理演算命令
【命令コード】
記載ページ：272

【構文】
(1) SHAR src, dest
(2) SHAR src, src2, dest

【オペレーション】
(1) dest = (signed long) dest >> (src & 31);
(2) dest = (signed long) src2 >> (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≦src≦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≦src≦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	○	“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

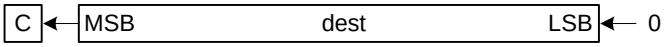
算術/論理演算命令
【命令コード】
記載ページ：273

【構文】
(1) SHLL src, dest
(2) SHLL src, src2, dest

【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≦src≦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≦src≦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

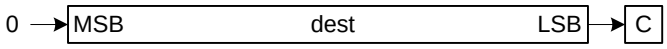
算術/論理演算命令
【命令コード】
記載ページ：274

【構文】
(1) SHLR src, dest
(2) SHLR src, src2, dest

【オペレーション】
(1) dest = (unsigned long) dest >> (src & 31);
(2) dest = (unsigned long) src2 >> (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≦src≦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≦src≦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

ストリング操作命令
【命令コード】
記載ページ：274

【構文】
SMOVB

【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	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：275

【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	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：275

【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	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

ストリング操作命令
【命令コード】
記載ページ：275

【構文】
SSTR.size

【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	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

転送命令
【命令コード】
記載ページ：276

【構文】
STNZ src, dest

【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：277

【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

算術/論理演算命令
【命令コード】
記載ページ：278

【構文】
(1) SUB src, dest
(2) SUB src, src2, dest

【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フローが発生しなかったとき“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	○	符号付き演算の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。

【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】
```
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

ストリング操作命令
【命令コード】
記載ページ：279

【構文】
SUNTIL.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/復帰してください。

【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：279

【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/復帰してください。

【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

算術/論理演算命令
【命令コード】
記載ページ：280

【構文】
TST src, src2

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

UTOF

整数→浮動小数点数変換
Integer TO Floating-point

UTOF

浮動小数点演算命令

【命令コード】
記載ページ：281

【構文】
UTOF src, dest

【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。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”、それ以外のとき“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”、それ以外のとき“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”、それ以外の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。

【発生例外】
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

システム操作命令
【命令コード】
記載ページ：282

【構文】
WAIT

【オペレーション】

【機能】

- プログラムの実行を停止します。ノンマスカブル割り込み、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

転送命令
【命令コード】
記載ページ：282

【構文】
XCHG　src, dest

【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】
```
XCHG　R1, R2
XCHG　[R1].W, R2
```

XOR

排他的論理和
eXclusive OR logical

XOR

算術/論理演算命令
【命令コード】
記載ページ：283

【構文】
XOR　src, dest

【オペレーション】
dest = dest ^ src;

【機能】
- 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。

【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	
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)

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

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

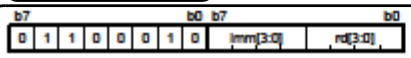
ADD

【コードサイズ】

構文	src	src2	dest	コードサイズ（バイト）
(1) ADD　src, dest	#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

(2)

(3) (1) ADD　src, dest

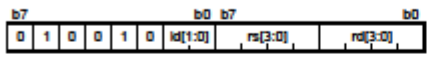


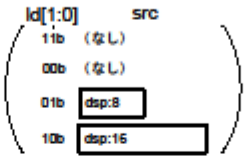
imm[3:0]	src	
0000b ～ 1111b	#UIMM:4	0 ～ 15

rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) ADD　src, dest

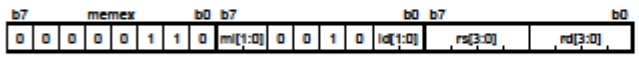
memex == UB または src==Rs の場合

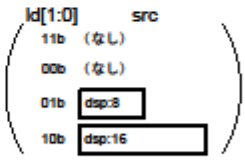




(4)

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

(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

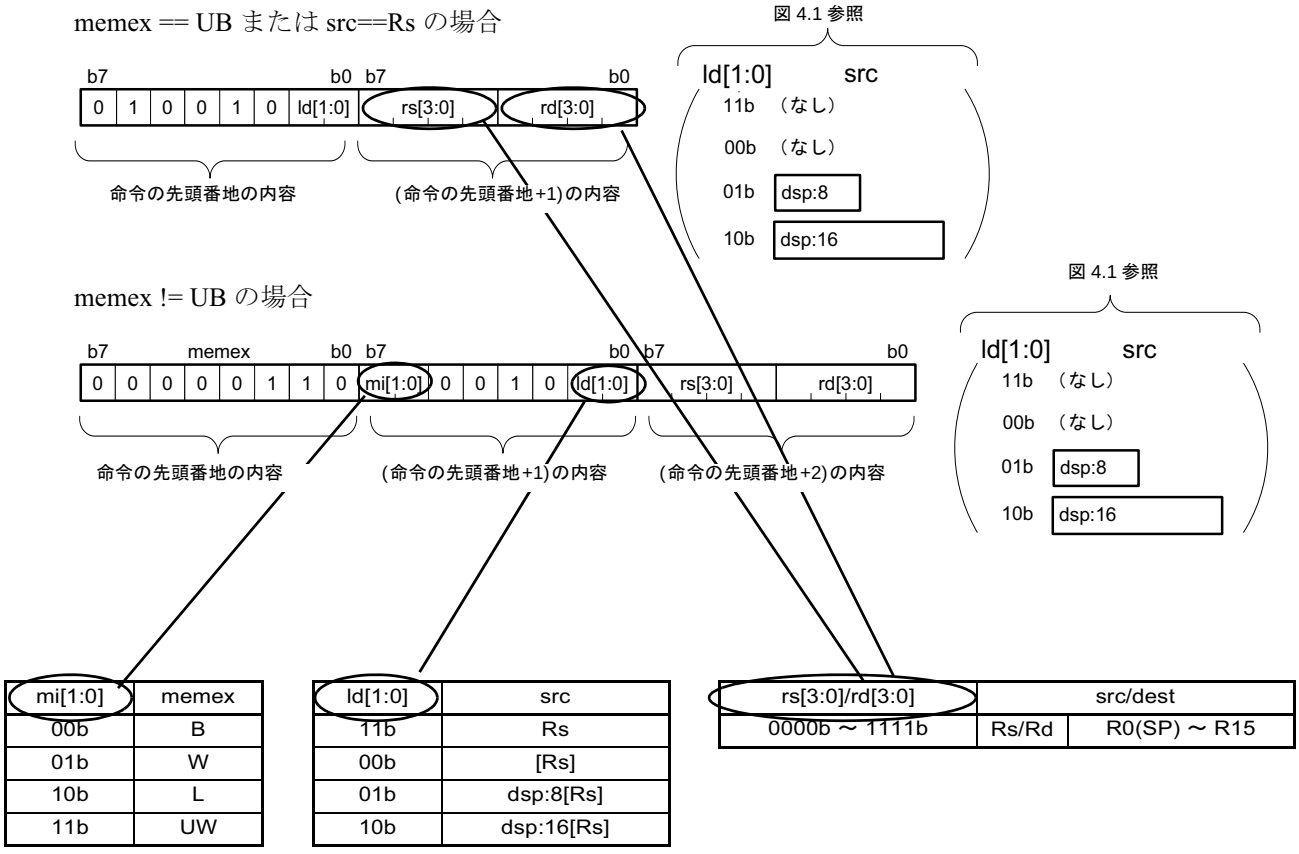
命令のバイト数を示しています。RXv2 CPU の命令のバイト数は、1 バイト～8 バイトです。

(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。



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

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。

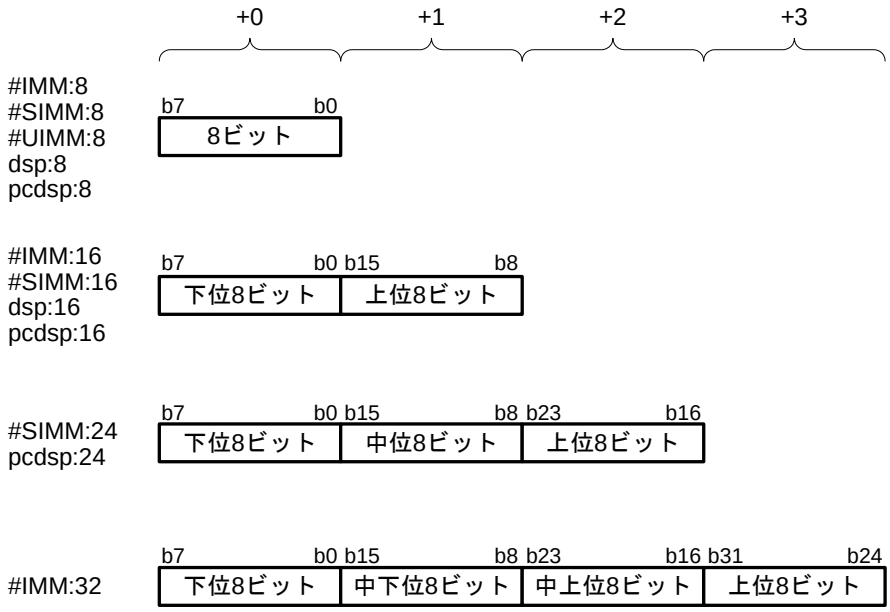


図 4.1　　即値（IMM）とディスプレースメント（dsp）

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

4.2 命令コード詳細説明

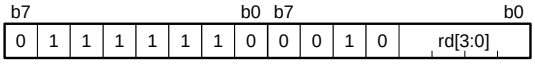
次ページより RXv2 CPU の命令コードの詳細説明を示します。

ABS ABS

【コードサイズ】

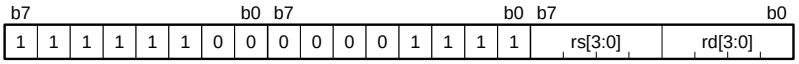
構文	src	dest	コードサイズ（バイト）
(1) ABS dest	—	Rd	2
(2) ABS src, dest	Rs	Rd	3

(1) ABS dest



rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) ABS src, dest



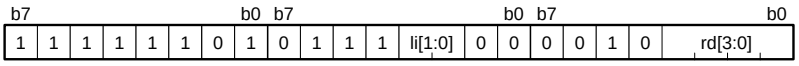
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

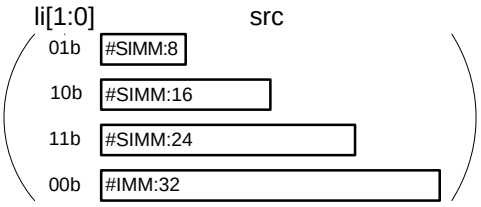
ADC　ADC

【コードサイズ】

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

(1) ADC　src, dest

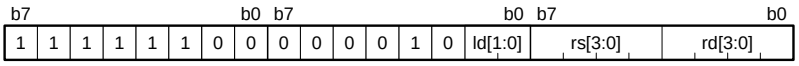




li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

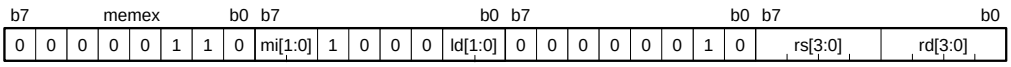
(2) ADC　src, dest

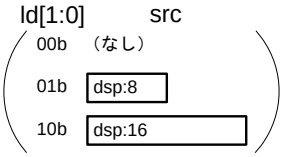


ld[1:0]	src
11b	Rs

rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

(3) ADC　src, dest





mi[1:0]	memex
10b	L

ld[1:0]	src
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

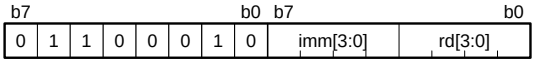
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

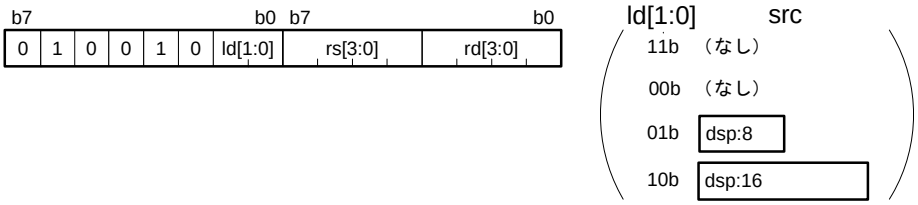


imm[3:0]	src	
0000b ～ 1111b	#UIMM:4	0 ～ 15

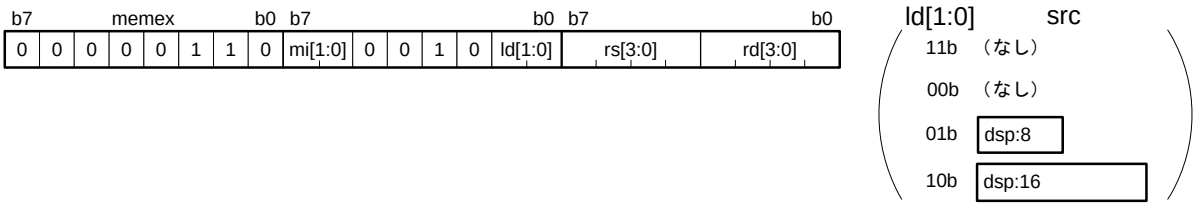
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) ADD src, dest

memex == UB または src==Rs の場合



memex != UB の場合

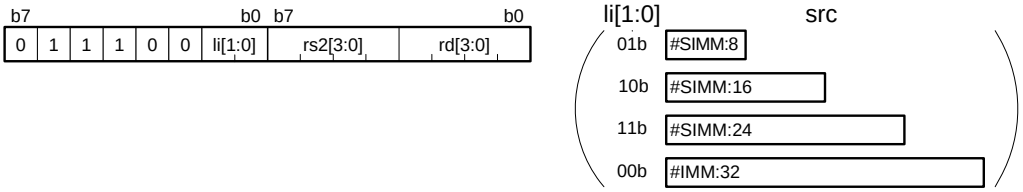


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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

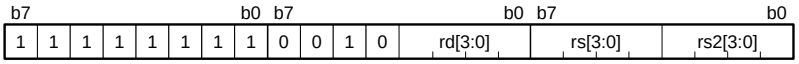
(3) ADD src, src2, dest



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



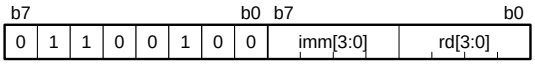
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ～ 1111b	Rs/Rs2/Rd	R0(SP) ～ R15

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

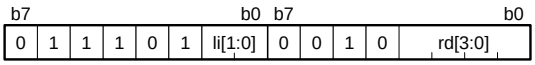
(1) AND src, dest

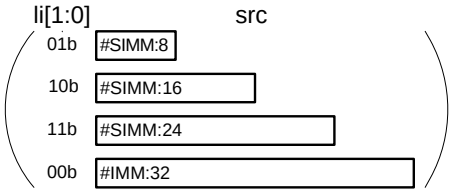


imm[3:0]	src	
0000b ～ 1111b	#UIMM:4	0 ～ 15

rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) AND src, dest



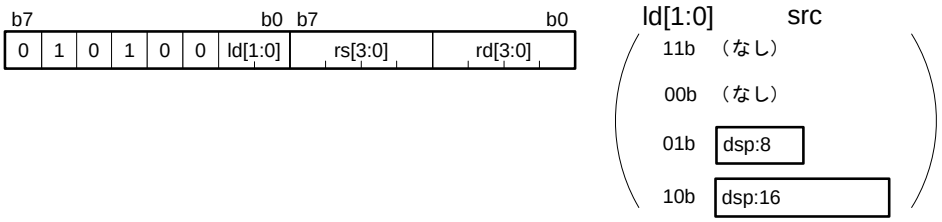


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

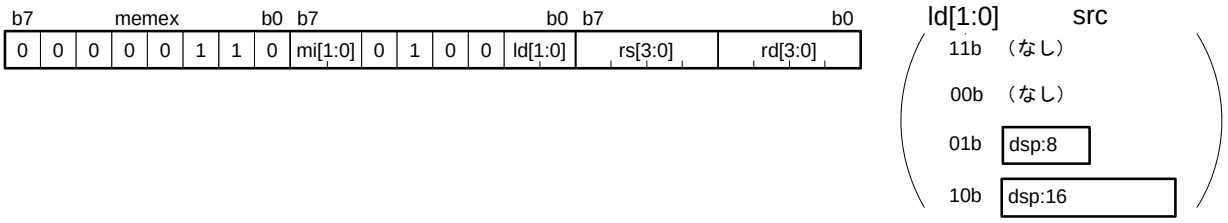
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(3) AND　src, dest

memex == UB または src==Rs の場合



memex != UB の場合

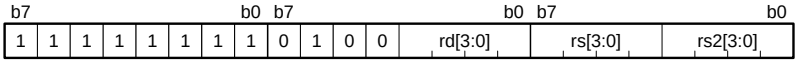


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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) AND　src, src2, dest



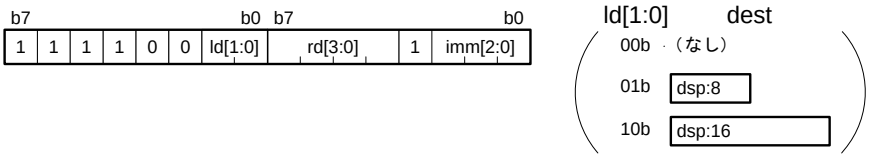
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ～ 1111b	Rs/Rs2/Rd	R0(SP) ～ R15

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

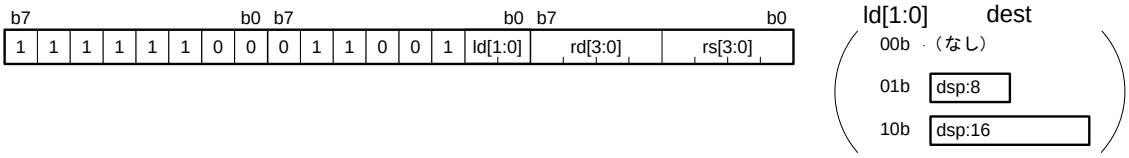


ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

imm[2:0]	src	
000b ～ 111b	#IMM:3	0 ～ 7

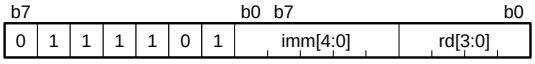
(2) BCLR src, dest



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

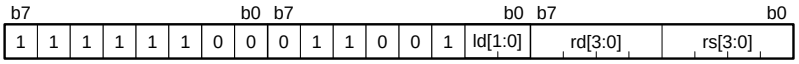
(3) BCLR src, dest



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

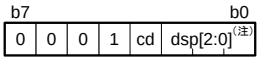
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

B*Cnd* B*Cnd*

【コードサイズ】

構文	src	コードサイズ（バイト）
(1) B*Cnd*.S src	pcdsp:3	1
(2) B*Cnd*.B src	pcdsp:8	2
(3) B*Cnd*.W src	pcdsp:16	3

(1) B*Cnd*.S src

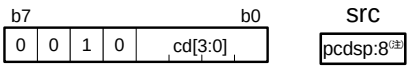


注 . dsp[2:0] で pcdsp:3 = src を指定します。

cd	B*Cnd*
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

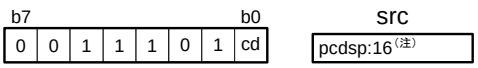
(2) B*Cnd*.B src



注 . pcdsp:8 = src が示す番地 -— 命令の先頭番地

cd[3:0]	B*Cnd*	cd[3:0]	B*Cnd*
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) B*Cnd*.W src



注 . pcdsp:16 = src が示す番地 -— 命令の先頭番地

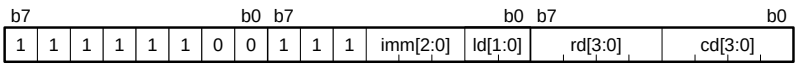
cd	B*Cnd*
0b	BEQ, BZ
1b	BNE, BNZ

BM*Cnd*　BM*Cnd*

【コードサイズ】

構文	src	dest	コードサイズ（バイト）
(1) BM*Cnd* src, dest	#IMM:3	[Rd].B	3
	#IMM:3	dsp:8[Rd].B	4
	#IMM:3	dsp:16[Rd].B	5
(2) BM*Cnd* src, dest	#IMM:5	Rd	3

(1) BM*Cnd* src, dest



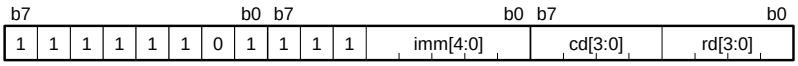
imm[2:0]	src	
000b ～ 111b	#IMM:3	0 ～ 7

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]	BM*Cnd*	cd[3:0]	BM*Cnd*
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



imm[4:0]	src	
00000b ～ 11111b	#IMM:5	0 ～ 31

cd[3:0]	BM*Cnd*	cd[3:0]	BM*Cnd*
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）

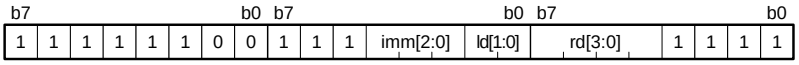
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

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

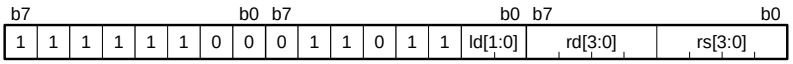


imm[2:0]	src	
000b ～ 111b	#IMM:3	0 ～ 7

ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

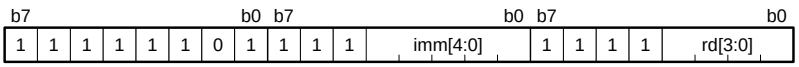
(2) BNOT　src, dest



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

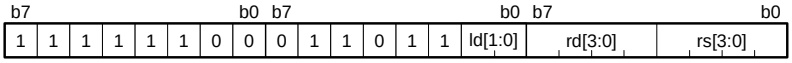
(3) BNOT　src, dest



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

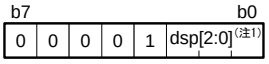
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

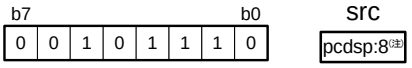
(1) BRA.S src



注 . dsp[2:0] で pcdsp:3 = src を指定します。

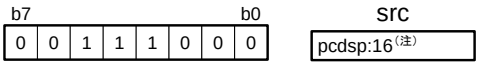
dsp[2:0]	分岐距離
011b	3
100b	4
101b	5
110b	6
111b	7
000b	8
001b	9
010b	10

(2) BRA.B src



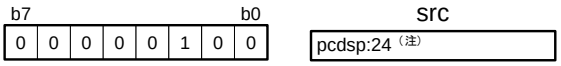
注 . pcdsp:8 = src が示す番地 - 命令の先頭番地

(3) BRA.W src



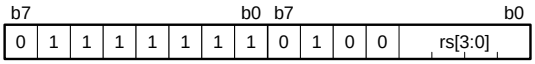
注 . pcdsp:16 = src が示す番地 - 命令の先頭番地

(4) BRA.A src



注 . pcdsp:24 = src が示す番地 - 命令の先頭番地

(5) BRA.L src



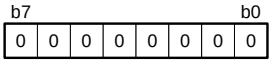
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

BRK BRK

【コードサイズ】

構文	コードサイズ（バイト）
(1) BRK	1

(1) BRK

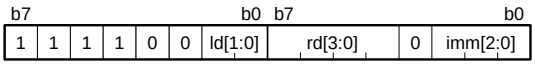


BSET BSET

【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

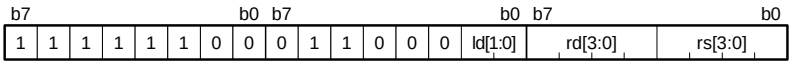


ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

imm[2:0]	src	
000b ～ 111b	#IMM:3	0 ～ 7

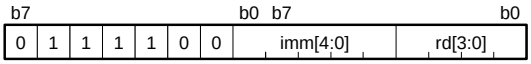
(2) BSET src, dest



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

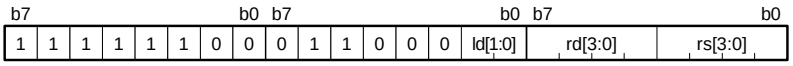
(3) BSET src, dest



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

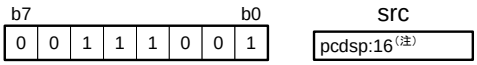
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

BSR BSR

【コードサイズ】

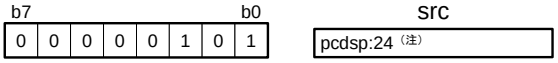
構文	src	コードサイズ（バイト）
(1) BSR.W src	pcdsp:16	3
(2) BSR.A src	pcdsp:24	4
(3) BSR.L src	Rs	2

(1) BSR.W src



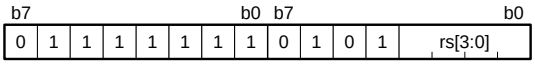
注 . pcdsp:16 = src が示す番地 - 命令の先頭番地

(2) BSR.A src



注 . pcdsp:24 = src が示す番地 - 命令の先頭番地

(3) BSR.L src



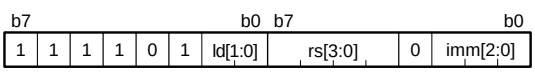
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

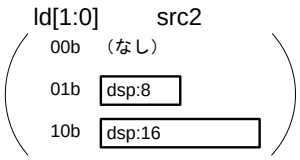
BTST BTST

【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



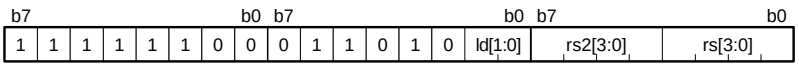


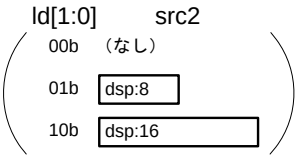
ld[1:0]	src2
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]	src2	
0000b ～ 1111b	Rs	R0(SP) ～ R15

imm[2:0]	src	
000b ～ 111b	#IMM:3	0 ～ 7

(2) BTST src, src2

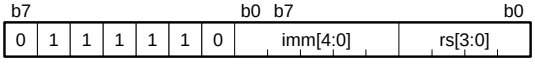




ld[1:0]	src2
00b	[Rs2]
01b	dsp:8[Rs2]
10b	dsp:16[Rs2]

rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

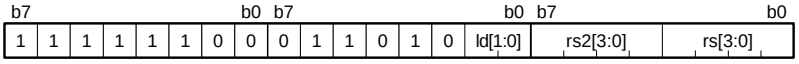
(3) BTST src, src2



imm[4:0]	src	
00000b ～ 11111b	#IMM:5	0 ～ 31

rs[3:0]	src2	
0000b ～ 1111b	Rs	R0(SP) ～ R15

(4) BTST src, src2



ld[1:0]	src2
11b	Rs2

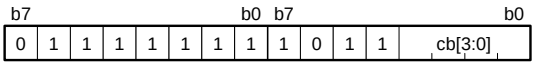
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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



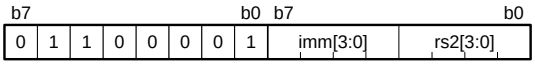
cb[3:0]	dest	
0000b	flag	C
0001b		Z
0010b		S
0011b		O
0100b		（予約）
0101b		（予約）
0110b		（予約）
0111b		（予約）
1000b		I
1001b		U
1010b		（予約）
1011b		（予約）
1100b		（予約）
1101b		（予約）
1110b		（予約）
1111b		（予約）

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)

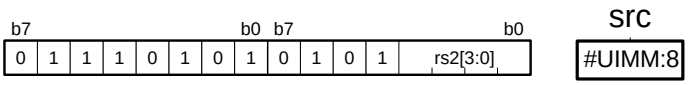
(1) CMP src, src2



imm[3:0]	src	
0000b ～ 1111b	#UIMM:4	0 ～ 15

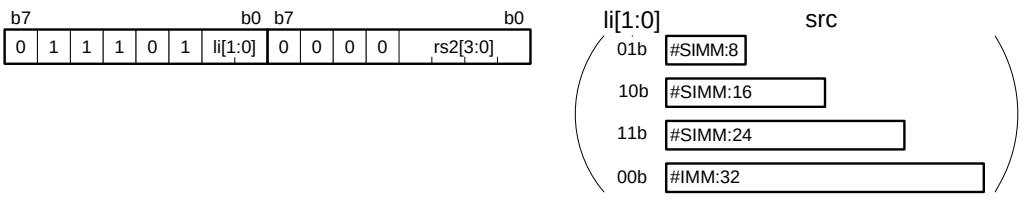
rs2[3:0]	src2	
0000b ～ 1111b	Rs	R0(SP) ～ R15

(2) CMP src, src2



rs2[3:0]	src2	
0000b ～ 1111b	Rs	R0(SP) ～ R15

(3) CMP src, src2

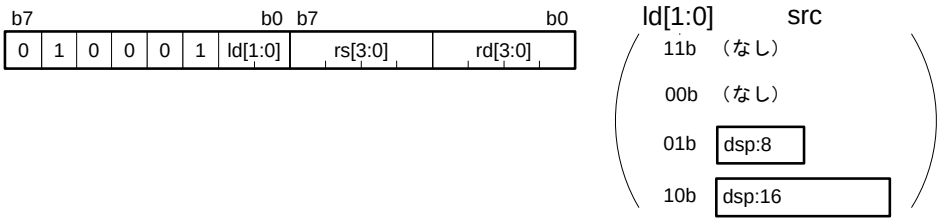


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

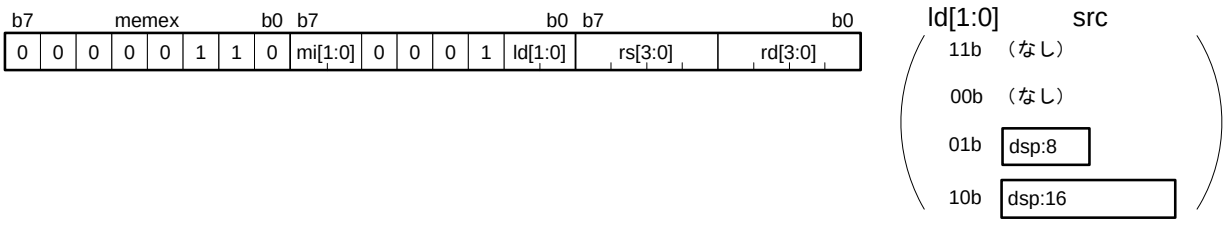
rs2[3:0]	src2	
0000b ～ 1111b	Rs	R0(SP) ～ R15

(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]

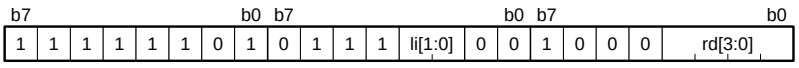
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

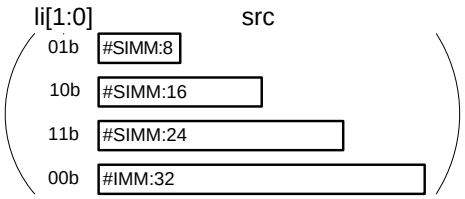
DIV DIV

【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



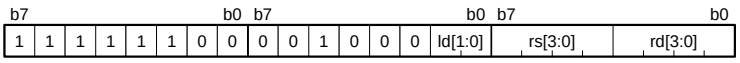


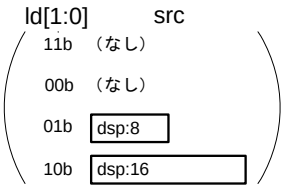
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

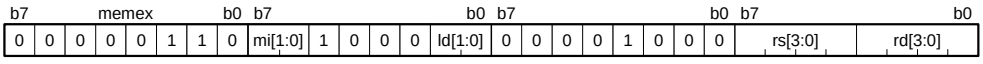
(2) DIV src, dest

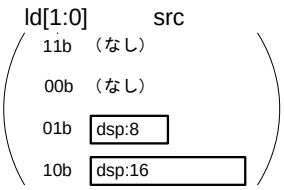
memex == UB または src==Rs の場合





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]

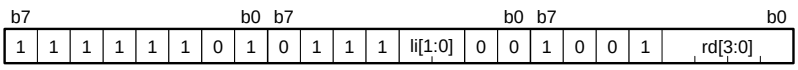
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

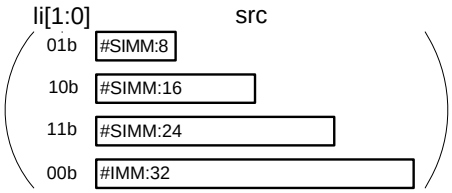
DIVU DIVU

【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



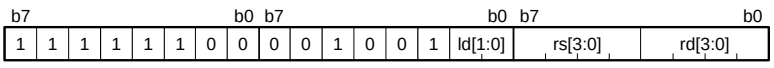


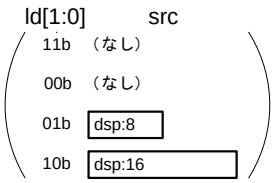
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

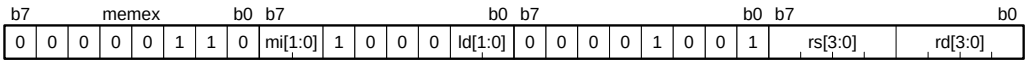
(2) DIVU src, dest

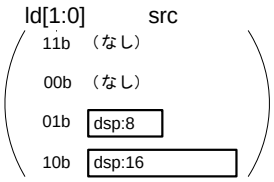
memex == UB または src==Rs の場合





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]

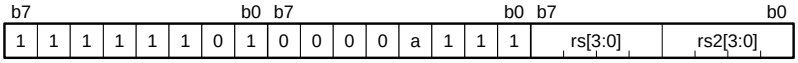
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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



a	Adest
0b	A0
1b	A1

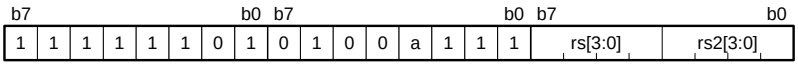
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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



a	Adest
0b	A0
1b	A1

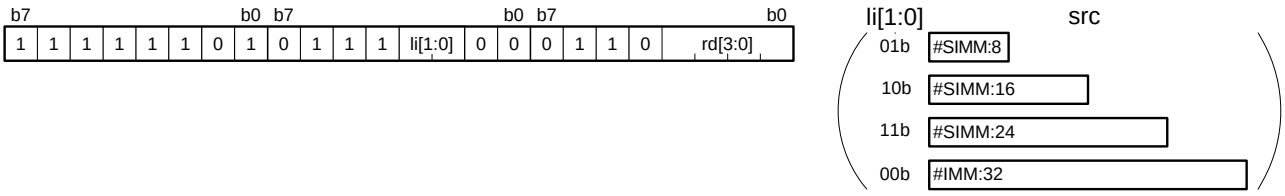
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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

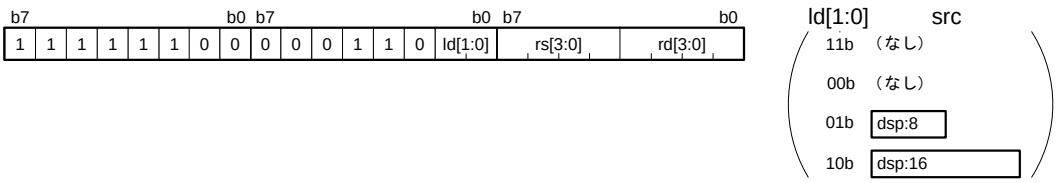


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

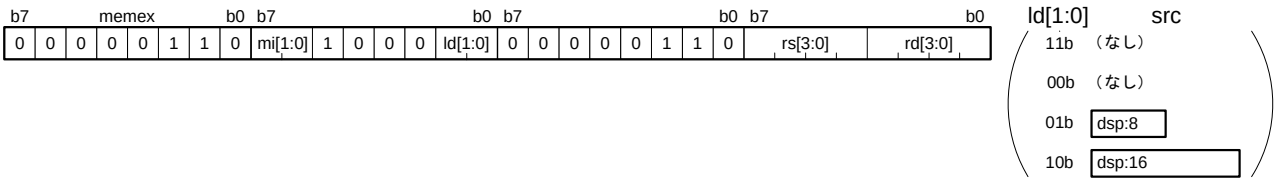
rd[3:0]	dest	
0000b～1110b	Rd	R0(SP)～R14

(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]

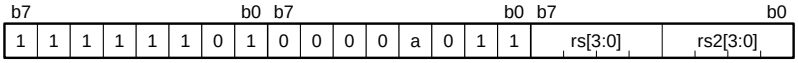
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15
rd[3:0]	dest	
0000b ～ 1110b	Rd	R0(SP) ～ R14

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

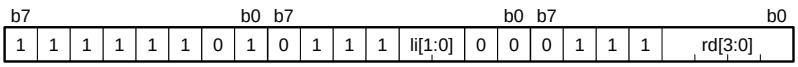
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

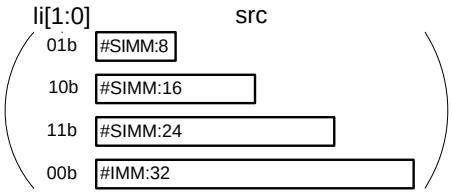
EMULU　EMULU

【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



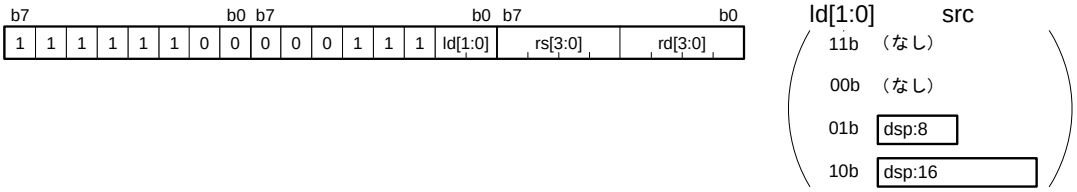


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

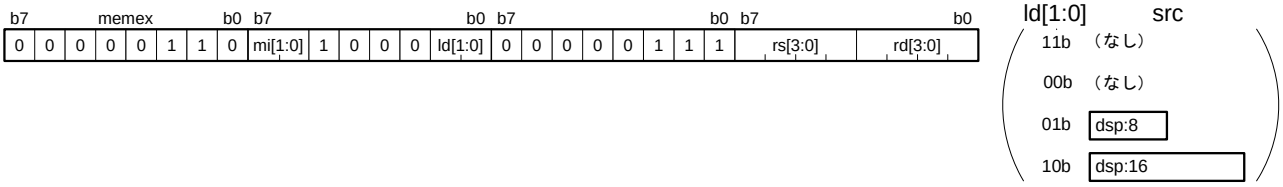
rd[3:0]	dest	
0000b ～ 1110b	Rd	R0(SP) ～ R14

(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]

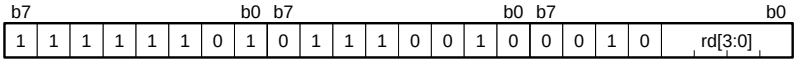
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15
rd[3:0]	dest	
0000b ～ 1110b	Rd	R0(SP) ～ R14

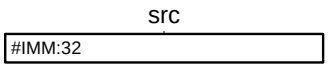
FADD　FADD

【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

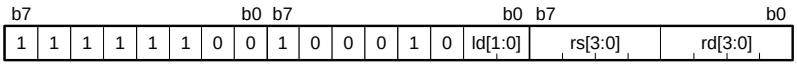
(1) FADD　src, dest

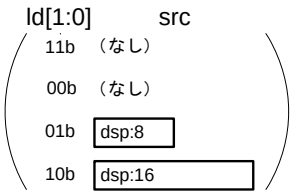




rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) FADD　src, dest

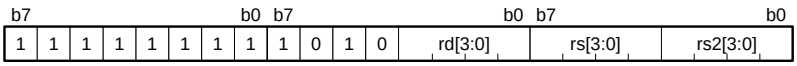




ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

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

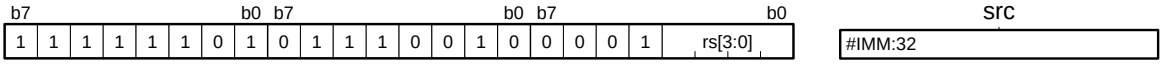
FCMP

FCMP

【コードサイズ】

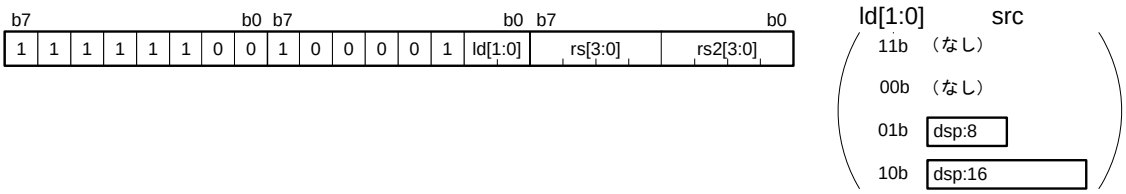
構文	src	src2	コードサイズ（バイト）
(1) FCMP src, src2	#IMM:32	Rs	7
(2) FCMP src, src2	Rs	Rs2	3
	[Rs].L	Rs2	3
	dsp:8[Rs].L	Rs2	4
	dsp:16[Rs].L	Rs2	5

(1) FCMP src, src2



rs[3:0]	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]

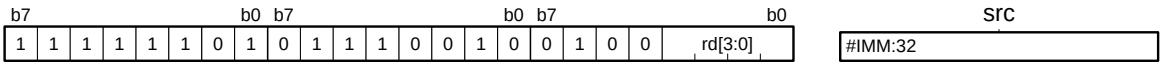
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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]

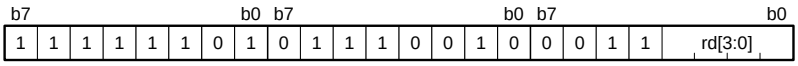
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

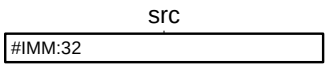
FMUL　FMUL

【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

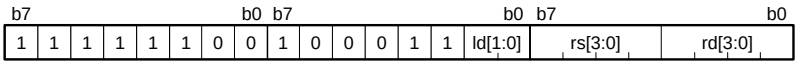
(1) FMUL　src, dest

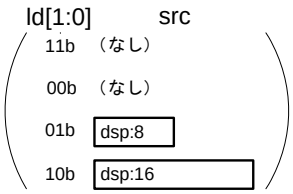




rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) FMUL　src, dest

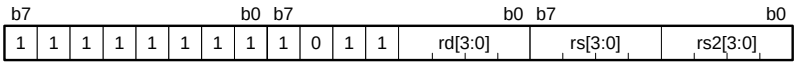




ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

(3) FMUL　src, src2, dest



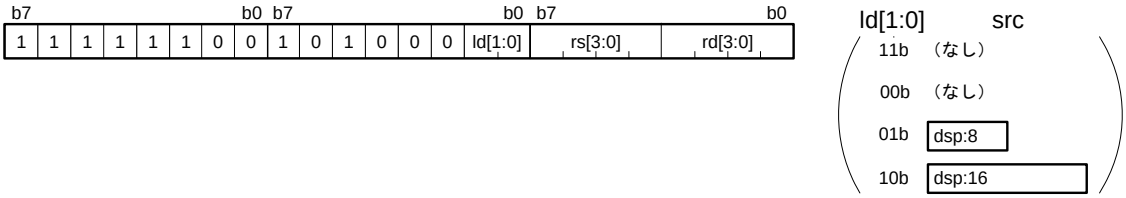
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ～ 1111b	Rs/Rs2/Rd	R0(SP) ～ R15

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]

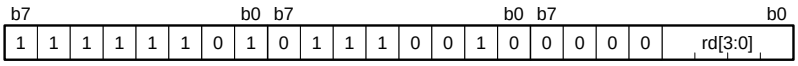
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

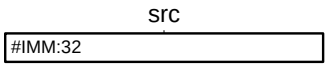
FSUB　FSUB

【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

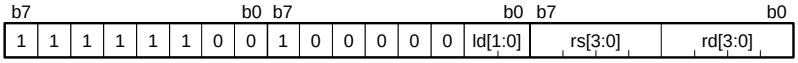
(1) FSUB src, dest

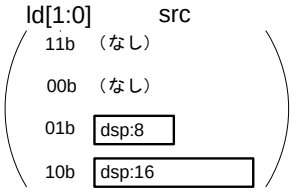




rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) FSUB src, dest

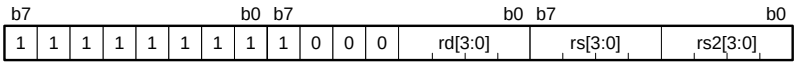




ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

(3) FSUB src, src2, dest



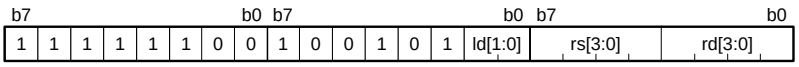
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ～ 1111b	Rs/Rs2/Rd	R0(SP) ～ R15

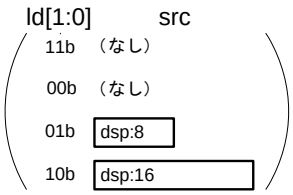
FTOI　FTOI

【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]

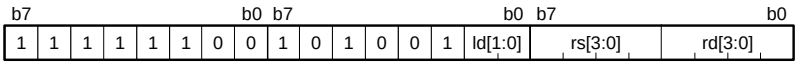
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

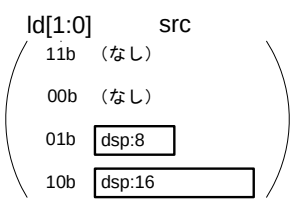
FTOU　FTOU

【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

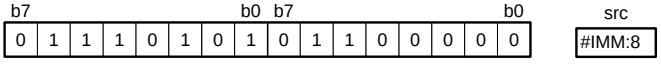
INT

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

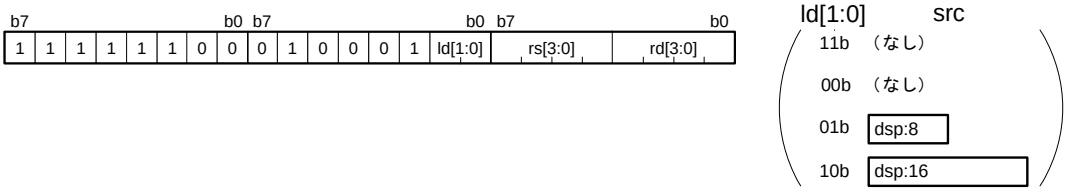
ITOF

【コードサイズ】

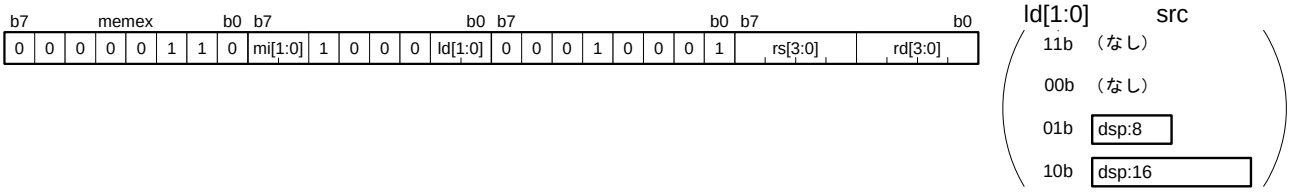
構文	src	dest	コードサイズ（バイト）
(1) ITOF src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(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]

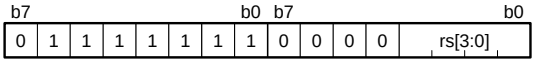
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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



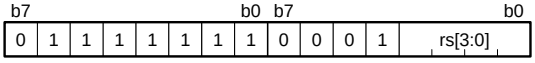
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

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



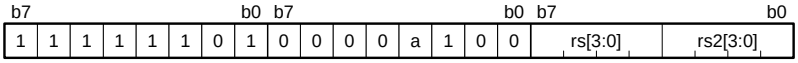
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

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



a	Adest
0b	A0
1b	A1

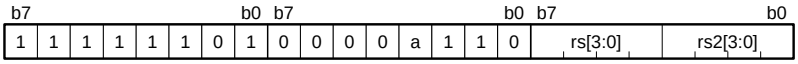
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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



a	Adest
0b	A0
1b	A1

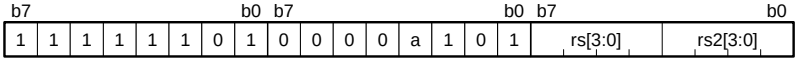
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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



a	Adest
0b	A0
1b	A1

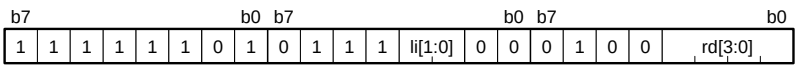
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

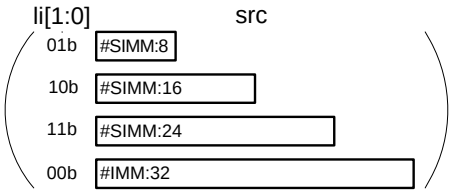
MAX　MAX

【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



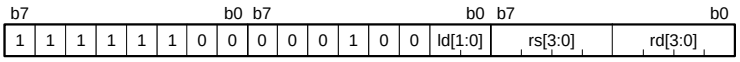


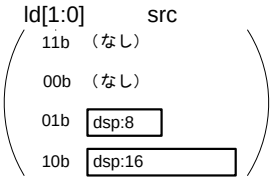
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

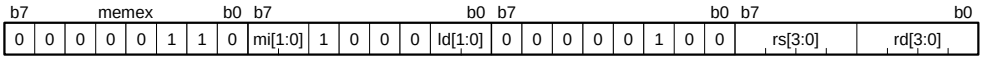
(2) MAX　src, dest

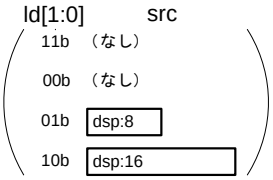
memex == UB または src==Rs の場合





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]

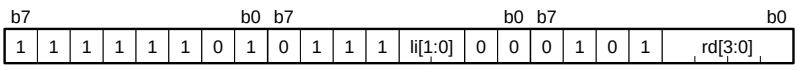
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

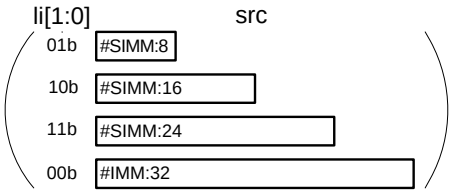
MIN　MIN

【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



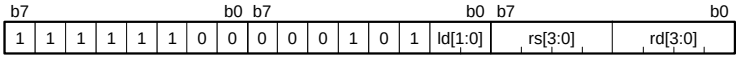


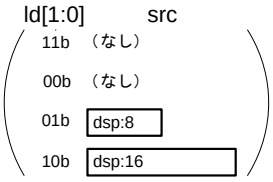
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

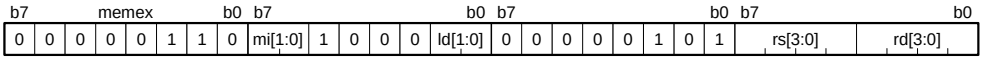
(2) MIN　src, dest

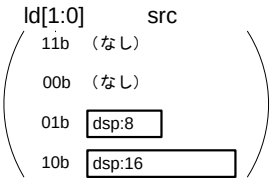
memex == UB または src==Rs の場合





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

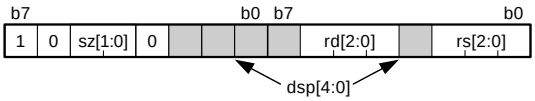
MOV MOV

【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
	B/W/L	size	dsp:16[Rs]	dsp:16[Rd]	6
(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

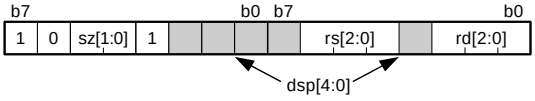


sz[1:0]	size
00b	B
01b	W
10b	L

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

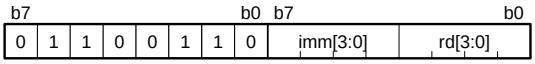


sz[1:0]	size
00b	B
01b	W
10b	L

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

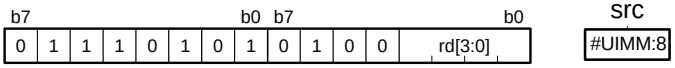
(4) MOV.size src, dest

sz[1:0]	size
00b	B
01b	W
10b	L

dsp[4:0]	dsp:5
00000b～11111b	0～31

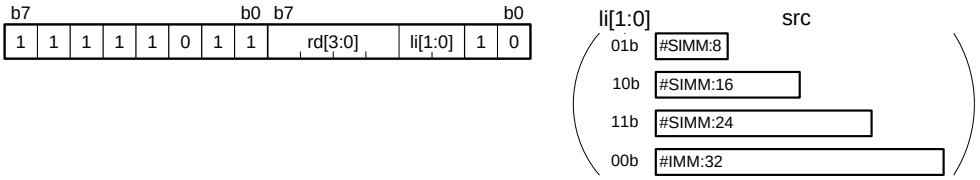
rd[2:0]	dest	
000b ～ 111b	Rd	R0(SP) ～ R7

(5) MOV.size src, dest



rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

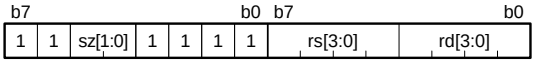
(6) MOV.size src, dest



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

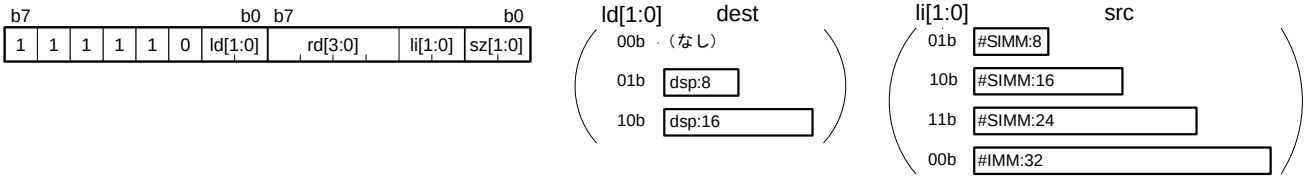
(7) MOV.size src, dest



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



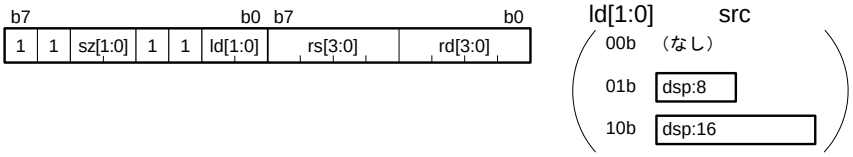
ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

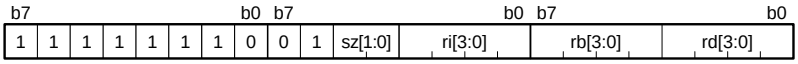


sz[1:0]	size
00b	B
01b	W
10b	L

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

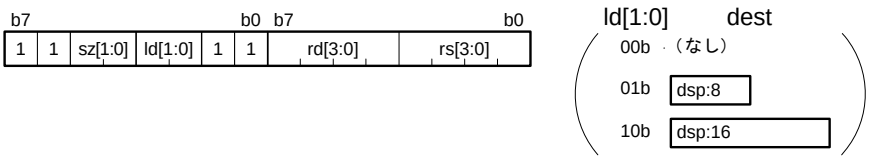
(10) MOV.size src, dest



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

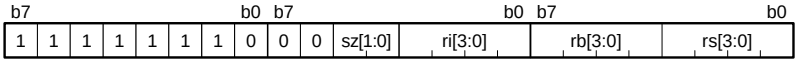


sz[1:0]	size
00b	B
01b	W
10b	L

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

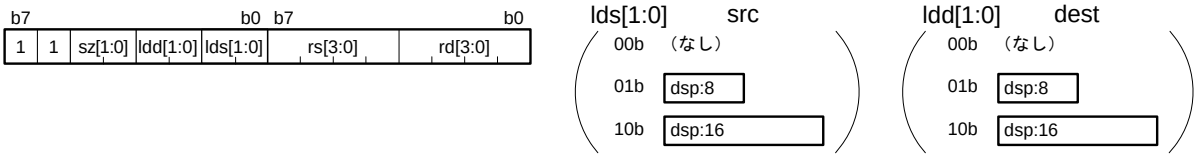
(12) MOV.size　src, dest



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

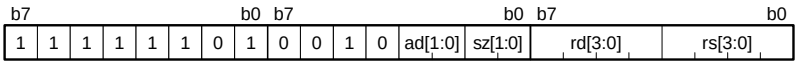


sz[1:0]	size
00b	B
01b	W
10b	L

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

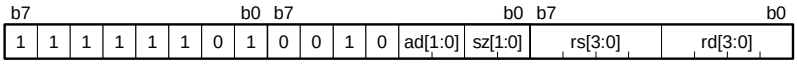


ad[1:0]	addressing
00b	Rs, [Rd+]
01b	Rs, [−Rd]

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

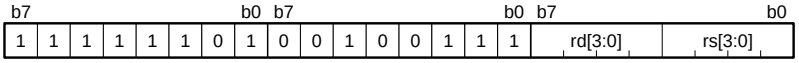
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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



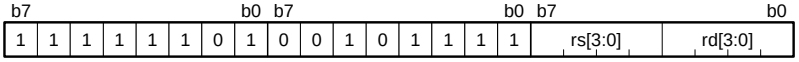
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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



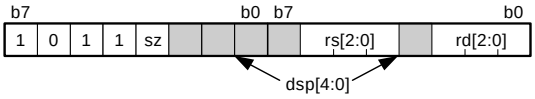
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

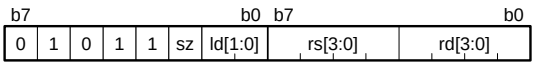


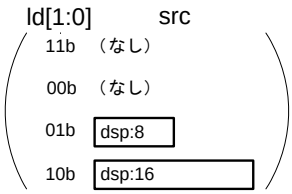
sz	size
0b	B
1b	W

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



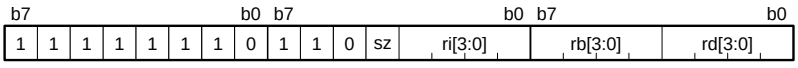


sz	size
0b	B
1b	W

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

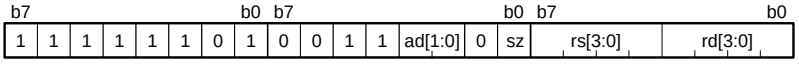
(3) MOVU.size src, dest



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

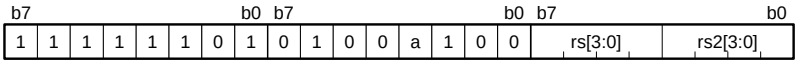
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

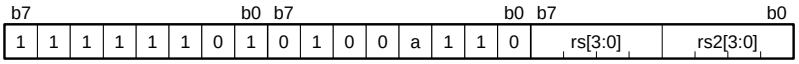
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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

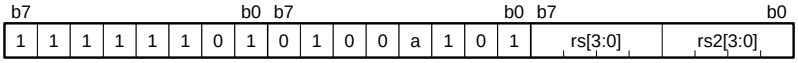
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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



a	Adest
0b	A0
1b	A1

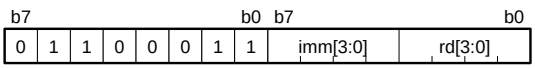
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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

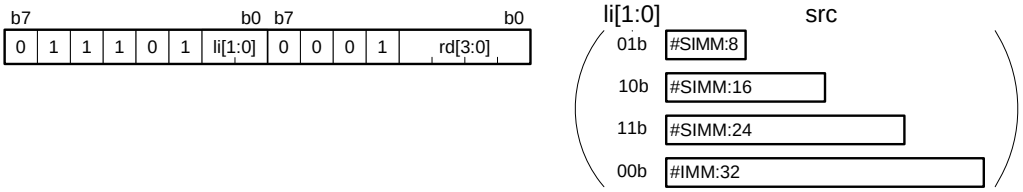
(1) MUL src, dest



imm[3:0]	src	
0000b ～ 1111b	#UIMM:4	0 ～ 15

rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) MUL src, dest

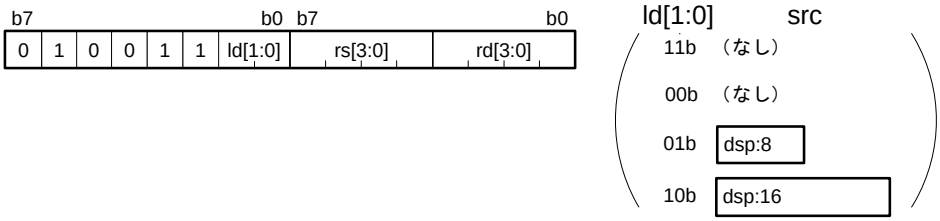


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

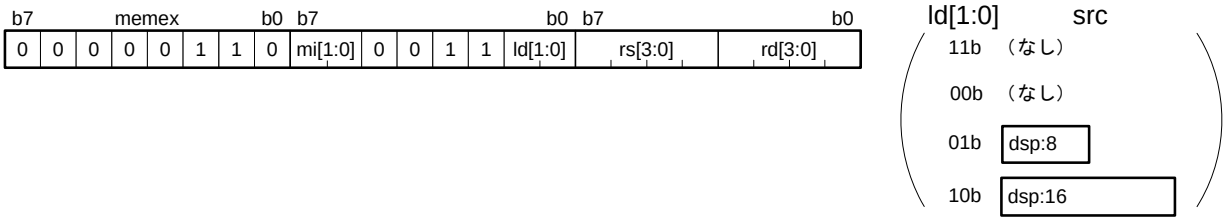
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(3) MUL src, dest

memex == UB または src==Rs の場合



memex != UB の場合

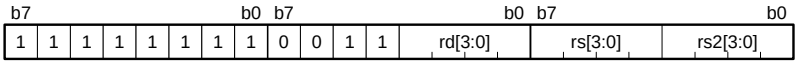


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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



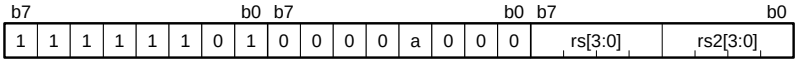
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ～ 1111b	Rs/Rs2/Rd	R0(SP) ～ R15

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



a	Adest
0b	A0
1b	A1

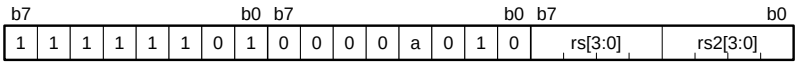
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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



a	Adest
0b	A0
1b	A1

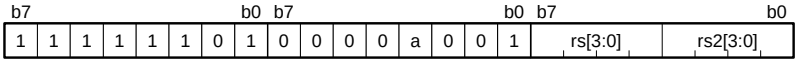
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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



a	Adest
0b	A0
1b	A1

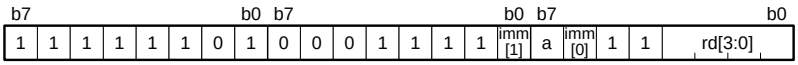
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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



a	Asrc
0b	A0
1b	A1

imm[1:0]	src	
00	#IMM:2	2
01		—
10		0
11		1

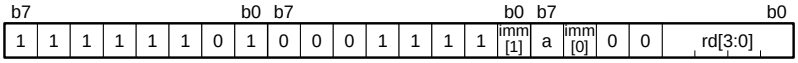
dest	
Rd	R0(SP) ～ R15

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



a	Asrc
0b	A0
1b	A1

imm[1:0]	src	
00	#IMM:2	2
01		—
10		0
11		1

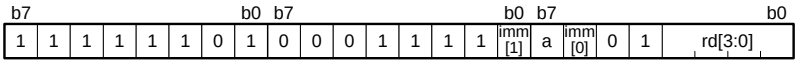
dest	
Rd	R0(SP) ～ R15

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



a	Asrc
0b	A0
1b	A1

imm[1:0]	src	
00	#IMM:2	2
01		—
10		0
11		1

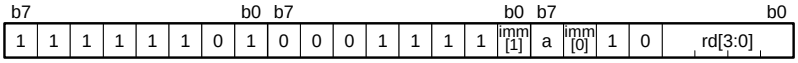
dest	
Rd	R0(SP) ～ R15

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



a	Asrc
0b	A0
1b	A1

imm[1:0]	src	
00	#IMM:2	2
01		—
10		0
11		1

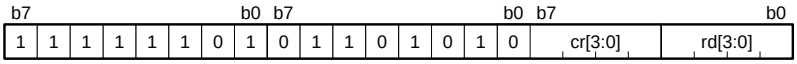
dest	
Rd	R0(SP) ～ R15

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



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）

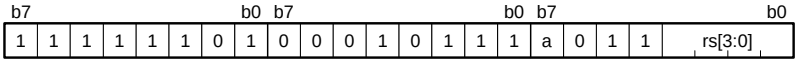
rd[3:0]	dest	
0000b～1111b	Rd	R0(SP)～R15

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



a	Adest
0b	A0
1b	A1

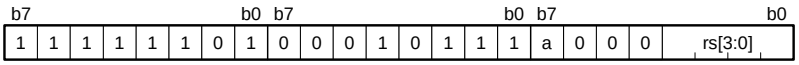
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

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



a	Adest
0b	A0
1b	A1

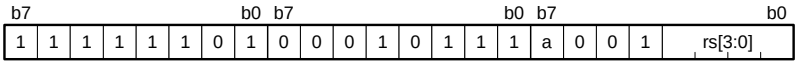
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

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



a	Adest
0b	A0
1b	A1

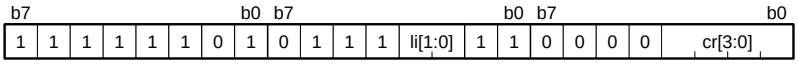
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

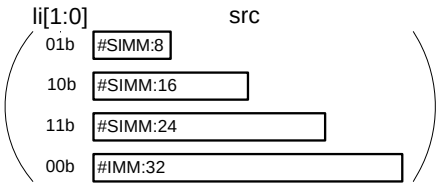
MVTC　MVTC

【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

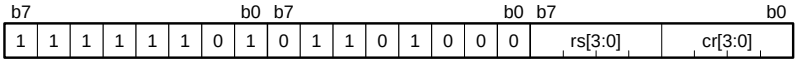




li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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）

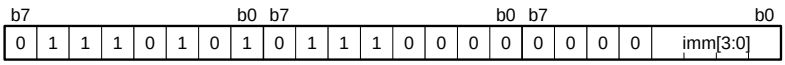
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

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



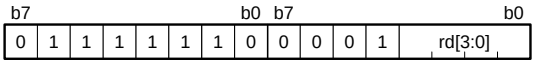
imm[3:0]	#IMM:4
0000b ～ 1111b	0 ～ 15

NEG NEG

【コードサイズ】

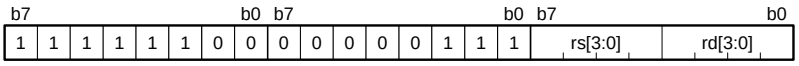
構文	src	dest	コードサイズ（バイト）
(1) NEG dest	—	Rd	2
(2) NEG src, dest	Rs	Rd	3

(1) NEG dest



rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) NEG src, dest



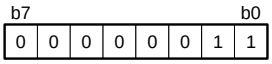
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

NOP NOP

【コードサイズ】

構文	コードサイズ（バイト）
(1) NOP	1

(1) NOP

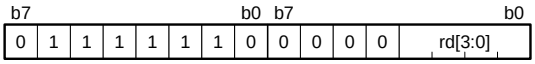


NOT NOT

【コードサイズ】

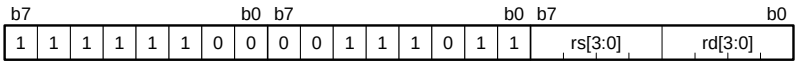
構文	src	dest	コードサイズ（バイト）
(1) NOT dest	—	Rd	2
(2) NOT src, dest	Rs	Rd	3

(1) NOT dest



rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) NOT src, dest



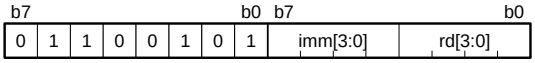
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

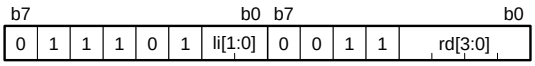
(1) OR src, dest

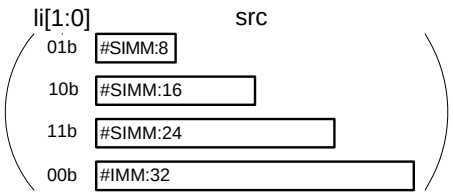


imm[3:0]	src	
0000b ～ 1111b	#UIMM:4	0 ～ 15

rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) OR src, dest



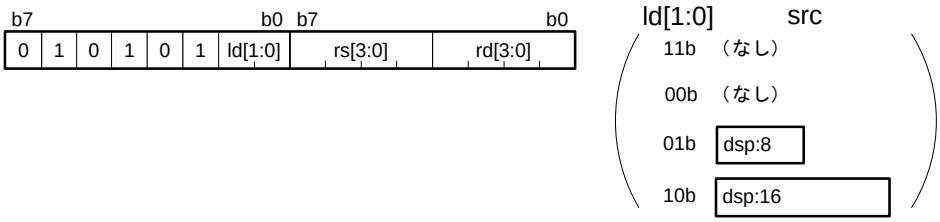


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

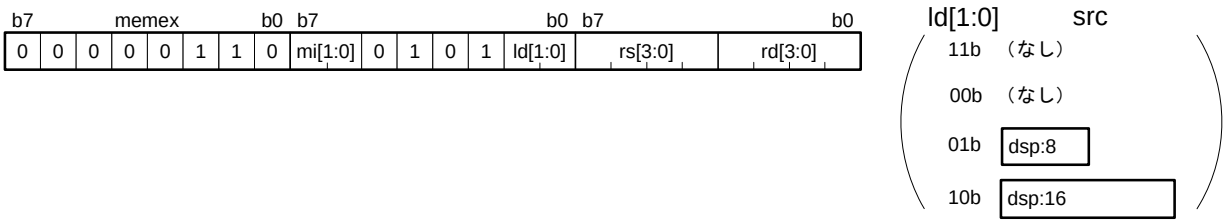
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(3) OR src, dest

memex == UB または src==Rs の場合



memex != UB の場合

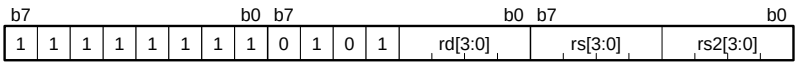


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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) OR src, src2, dest



rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ～ 1111b	Rs/Rs2/Rd	R0(SP) ～ R15

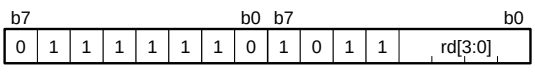
POP

POP

【コードサイズ】

構文	dest	コードサイズ（バイト）
(1) POP dest	Rd	2

(1) POP dest



rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

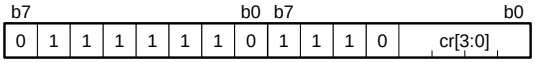
POPC

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



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）

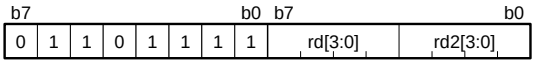
POPM

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



rd[3:0]	dest	
0001b ～ 1110b	Rd	R1 ～ R14

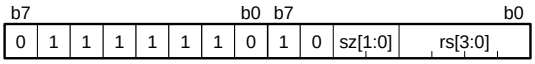
rd2[3:0]	dest2	
0010b ～ 1111b	Rd2	R2 ～ R15

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

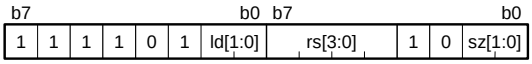
(1) PUSH.size src

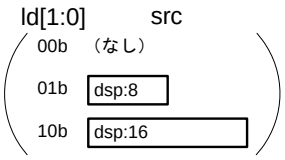


sz[1:0]	size
00b	B
01b	W
10b	L

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

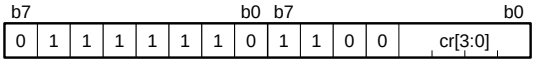
sz[1:0]	size
00b	B
01b	W
10b	L

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



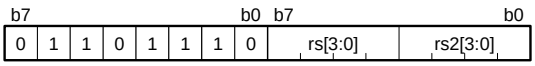
cr[3:0]	src	
0000b	Rx	PSW
0001b		PC
0010b		USP
0011b		FPSW
0100b		（予約）
0101b		（予約）
0110b		（予約）
0111b		（予約）
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b		EXTB
1110b～1111b		（予約）

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



rs[3:0]	src	
0001b ～ 1110b	Rs	R1 ～ R14

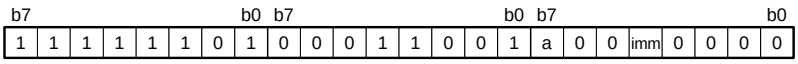
rs2[3:0]	src2	
0010b ～ 1111b	Rs2	R2 ～ R15

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

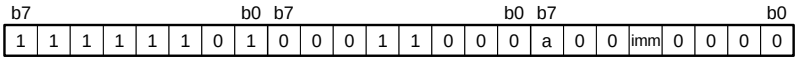
imm	src	
0b ～ 1b	#IMM:1	1 ～ 2

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

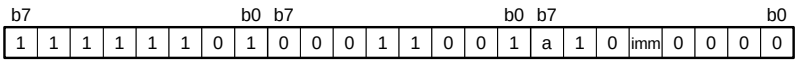
imm	src	
0b ～ 1b	#IMM:1	1 ～ 2

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



a	Adest
0b	A0
1b	A1

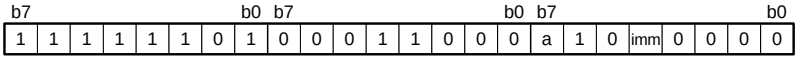
imm	src	
0b ～ 1b	#IMM:1	1 ～ 2

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



a	Adest
0b	A0
1b	A1

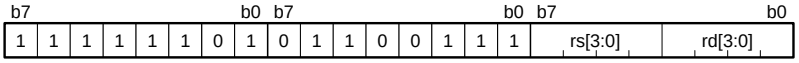
imm	src	
0b ～ 1b	#IMM:1	1 ～ 2

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



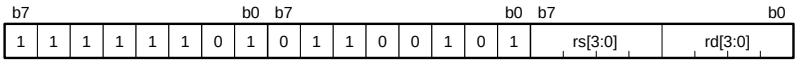
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

REVW REVW

【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



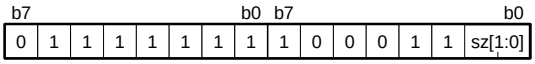
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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



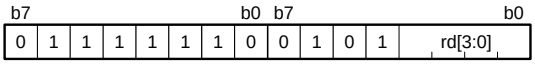
sz[1:0]	size
00b	B
01b	W
10b	L

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



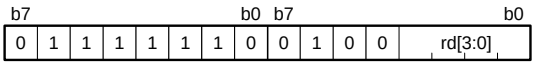
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

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



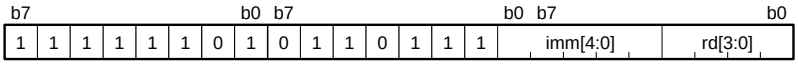
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

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

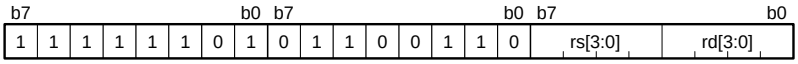
(1) ROTL src, dest



imm[4:0]	src	
00000b ～ 11111b	#IMM:5	0 ～ 31

rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) ROTL src, dest



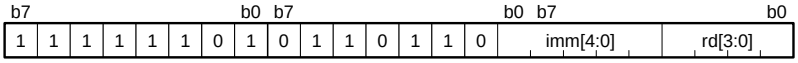
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

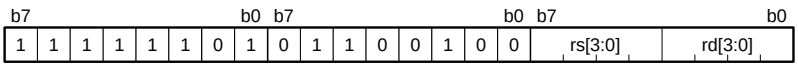
(1) ROTR src, dest



imm[4:0]	src	
00000b ～ 11111b	#IMM:5	0 ～ 31

rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) ROTR src, dest



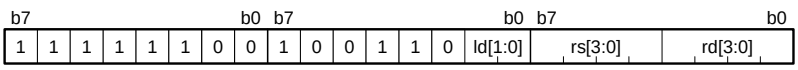
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

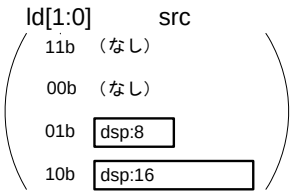
ROUND　ROUND

【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]

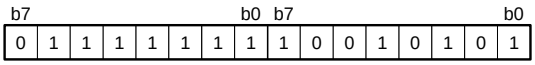
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

RTE　RTE

【コードサイズ】

構文	コードサイズ（バイト）
(1) RTE	2

(1) RTE

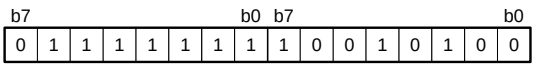


RTFI　RTFI

【コードサイズ】

構文	コードサイズ（バイト）
(1) RTFI	2

(1) RTFI

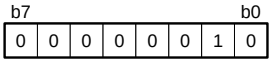


RTS　RTS

【コードサイズ】

構文	コードサイズ（バイト）
(1) RTS	1

(1) RTS

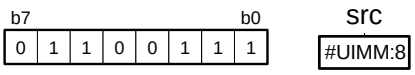


RTSD　RTSD

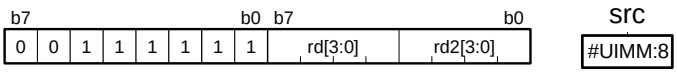
【コードサイズ】

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



(2) RTSD　src, dest-dest2



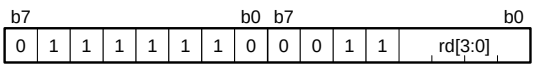
rd[3:0]/rd2[3:0]	dest/dest2	
0001b ～ 1111b	Rd/Rd2	R1 ～ R15

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



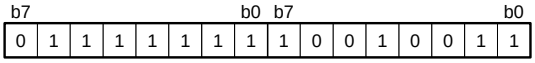
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

SATR　SATR

【コードサイズ】

構文	コードサイズ（バイト）
(1) SATR	2

(1) SATR

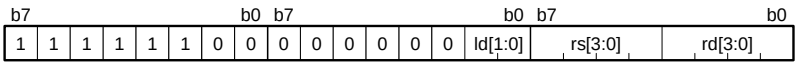


SBB　SBB

【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

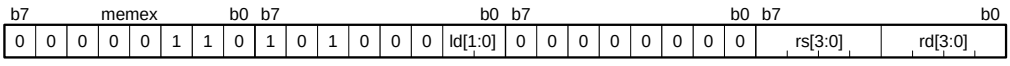
(1) SBB　src, dest



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



ld[1:0]	src
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

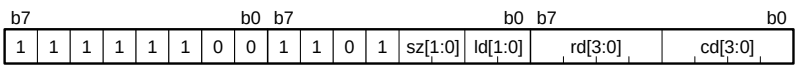
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

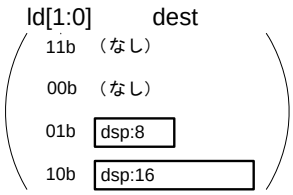
SC*Cnd* SC*Cnd*

【コードサイズ】

構文	size	dest	コードサイズ（バイト）
(1) SC*Cnd*.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) SC*Cnd*.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

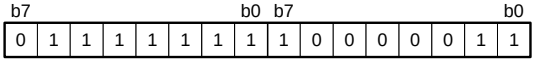
cd[3:0]	SC*Cnd*	cd[3:0]	SC*Cnd*
0000b	SCEQ, SCZ	1000b	SCGE
0001b	SCNE, SCNZ	1001b	SCLT
0010b	SCGEU, SCC	1010b	SCGT
0011b	SCLTU, SCNC	1011b	SCLE
0100b	SCGTU	1100b	SCO
0101b	SCLEU	1101b	SCNO
0110b	SCPZ	1110b	(予約)
0111b	SCN	1111b	(予約)

SCMPU SCMPU

【コードサイズ】

構文	コードサイズ（バイト）
(1) SCMPU	2

(1) SCMPU

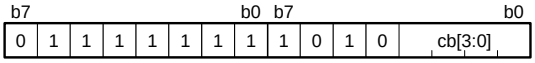


SETPSW SETPSW

【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



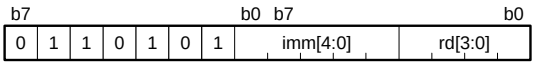
cb[3:0]	dest	
0000b	flag	C
0001b		Z
0010b		S
0011b		O
0100b		（予約）
0101b		（予約）
0110b		（予約）
0111b		（予約）
1000b		I
1001b		U
1010b		（予約）
1011b		（予約）
1100b		（予約）
1101b		（予約）
1110b		（予約）
1111b		（予約）

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

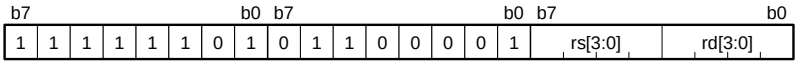
(1) SHAR src, dest



imm[4:0]	src	
00000b ～ 11111b	#IMM:5	0 ～ 31

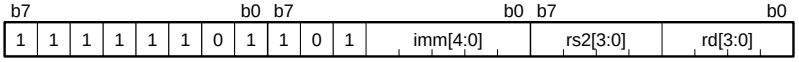
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) SHAR src, dest



rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

(3) SHAR src, src2, dest



imm[4:0]	src	
00000b ～ 11111b	#IMM:5	0 ～ 31

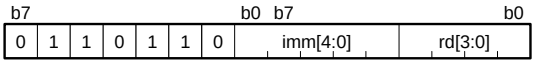
rs2[3:0]/rd[3:0]	src2/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

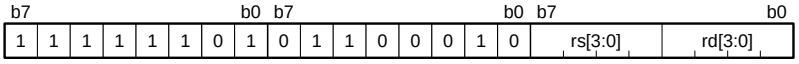
(1) SHLL src, dest



imm[4:0]	src	
00000b ～ 11111b	#IMM:5	0 ～ 31

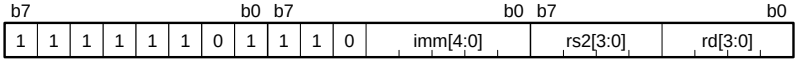
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) SHLL src, dest



rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

(3) SHLL src, src2, dest



imm[4:0]	src	
00000b ～ 11111b	#IMM:5	0 ～ 31

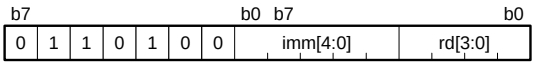
rs2[3:0]/rd[3:0]	src2/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

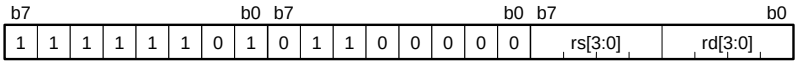
(1) SHLR src, dest



imm[4:0]	src	
00000b ～ 11111b	#IMM:5	0 ～ 31

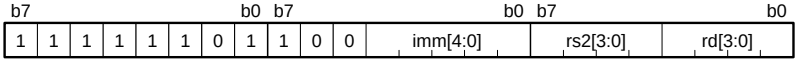
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) SHLR src, dest



rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

(3) SHLR src, src2, dest



imm[4:0]	src	
00000b ～ 11111b	#IMM:5	0 ～ 31

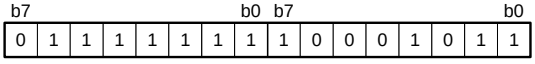
rs2[3:0]/rd[3:0]	src2/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

SMOVB SMOVB

【コードサイズ】

構文	コードサイズ（バイト）
(1) SMOVB	2

(1) SMOVB

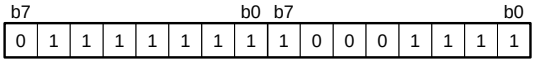


SMOVF　SMOVF

【コードサイズ】

構文	コードサイズ（バイト）
(1) SMOVF	2

(1) SMOVF

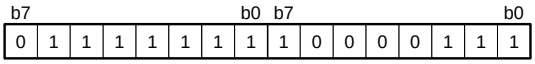


SMOVU　SMOVU

【コードサイズ】

構文	コードサイズ（バイト）
(1) SMOVU	2

(1) SMOVU

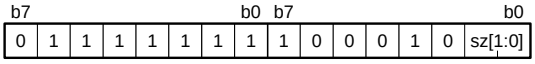


SSTR　SSTR

【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



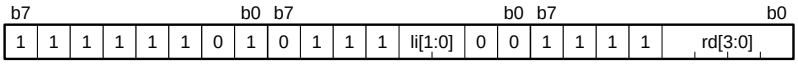
sz[1:0]	size
00b	B
01b	W
10b	L

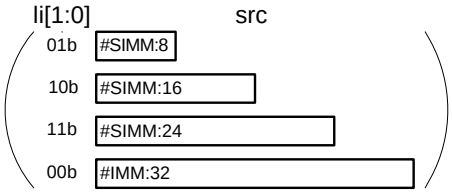
STNZ STNZ

【コードサイズ】

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

(1) STNZ src, dest

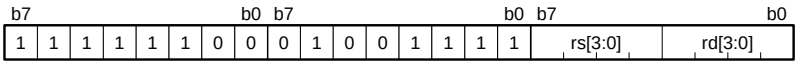




li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) STNZ src, dest



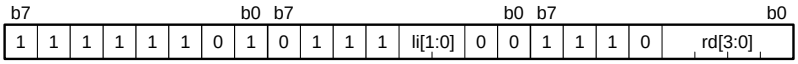
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

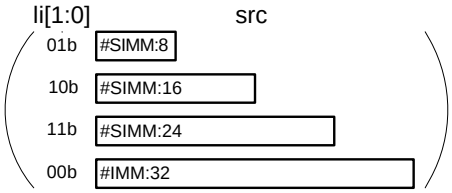
STZ STZ

【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

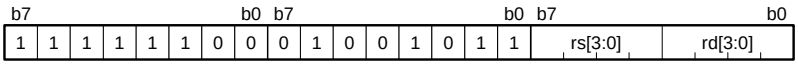




li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) STZ src, dest



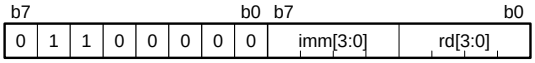
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

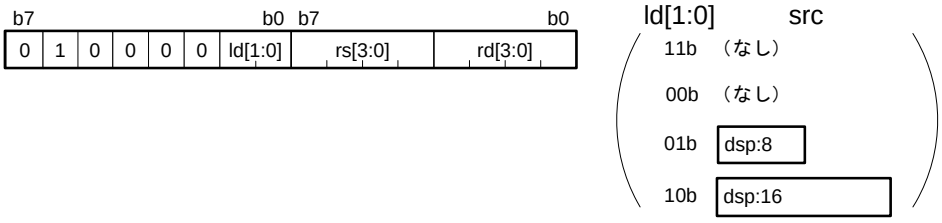


imm[3:0]	src	
0000b ～ 1111b	#UIMM:4	0 ～ 15

rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) SUB src, dest

memex == UB または src==Rs の場合



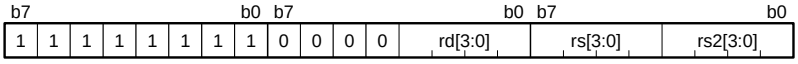
memex != UB の場合

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



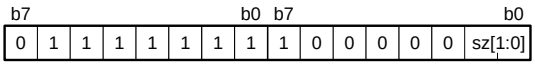
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ～ 1111b	Rs/Rs2/Rd	R0(SP) ～ R15

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



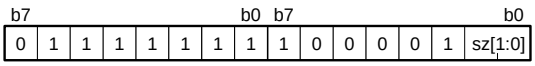
sz[1:0]	size
00b	B
01b	W
10b	L

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



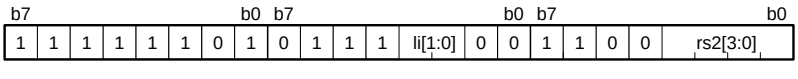
sz[1:0]	size
00b	B
01b	W
10b	L

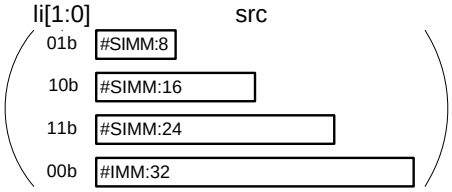
TST　TST

【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



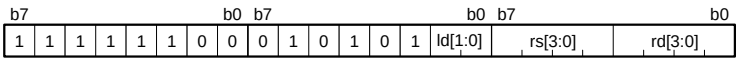


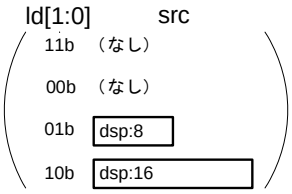
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rs2[3:0]	src2	
0000b ～ 1111b	Rs	R0(SP) ～ R15

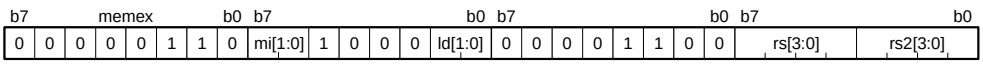
(2) TST src, src2

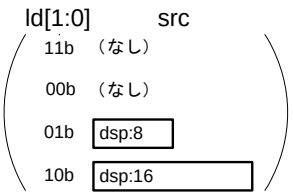
memex == UB または src==Rs の場合





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

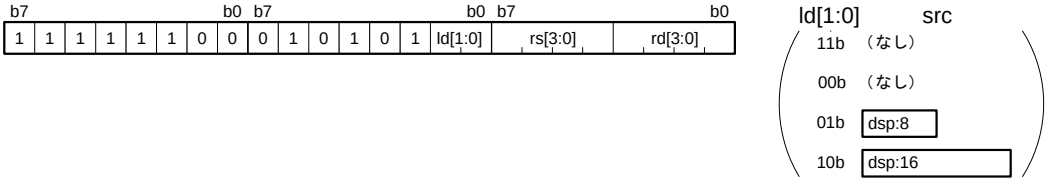
UTOF　UTOF

【コードサイズ】

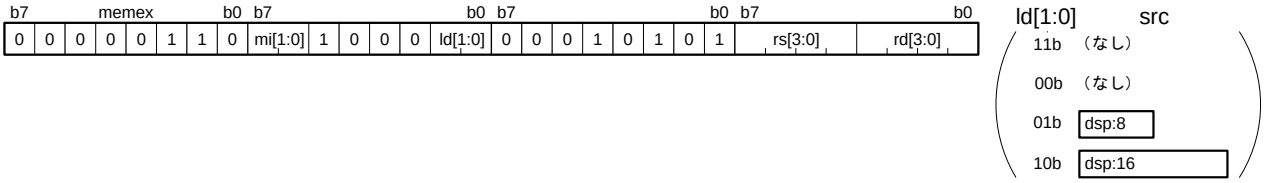
構文	src	dest	コードサイズ（バイト）
(1) UTOF　src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(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]

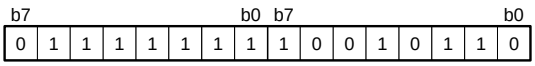
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

WAIT　WAIT

【コードサイズ】

構文	コードサイズ（バイト）
(1) WAIT	2

(1) WAIT



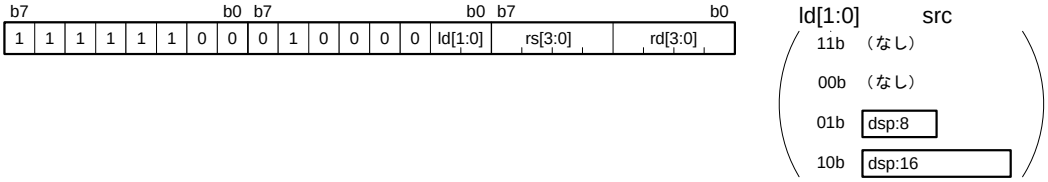
XCHG　XCHG

【コードサイズ】

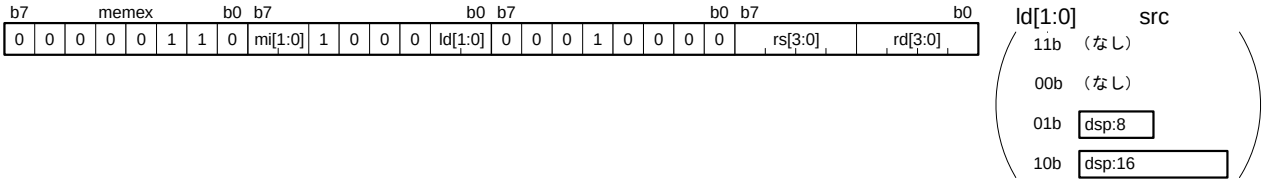
構文	src	dest	コードサイズ（バイト）
(1) XCHG　src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == UB) 4 (memex != UB)
	dsp:8[Rs].memex	Rd	4 (memex == UB) 5 (memex != UB)
	dsp:16[Rs].memex	Rd	5 (memex == UB) 6 (memex != UB)

(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]

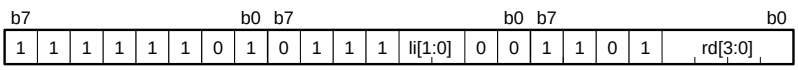
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

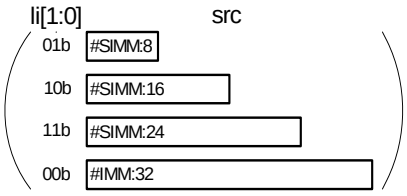
XOR　XOR

【コードサイズ】

構文	src	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)

(1) XOR src, dest



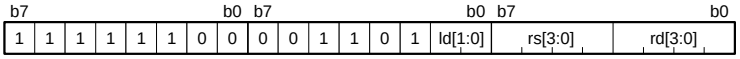


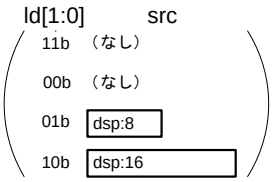
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

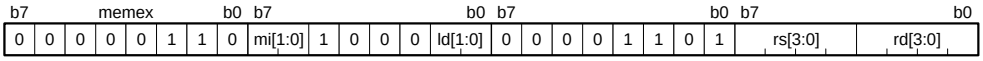
(2) XOR src, dest

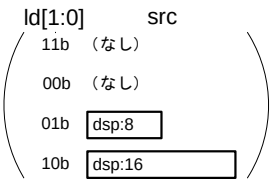
memex == UB または src==Rs の場合





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

5. 例外処理

5.1 例外事象

CPU が通常の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生じる場合があります。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。

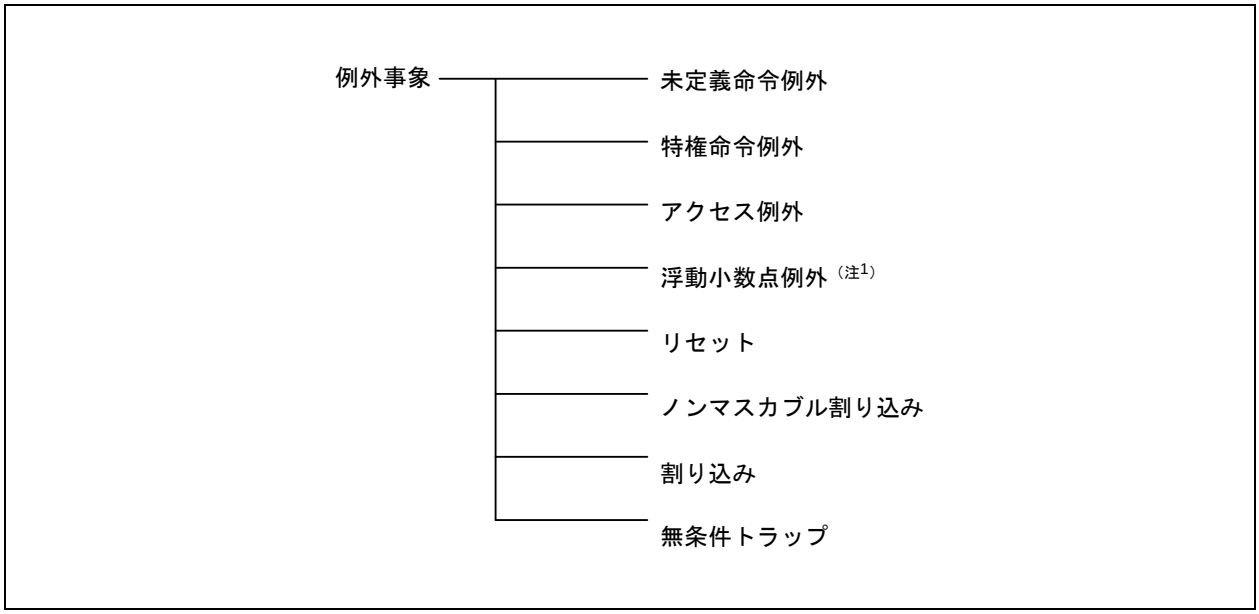


図 5.1　例外事象の種類

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 浮動小数点例外

浮動小数点例外は、IEEE754 規格で規定された 5 つの例外事象（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、非実装処理を検出した場合に発生します。浮動小数点例外は、FPSW の EX、EU、EZ、EO、EV ビットが “0” のとき、例外処理が禁止されます。

5.1.5 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。

5.1.6 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にのみ使用します。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。

5.1.7 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。

5.1.8 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。

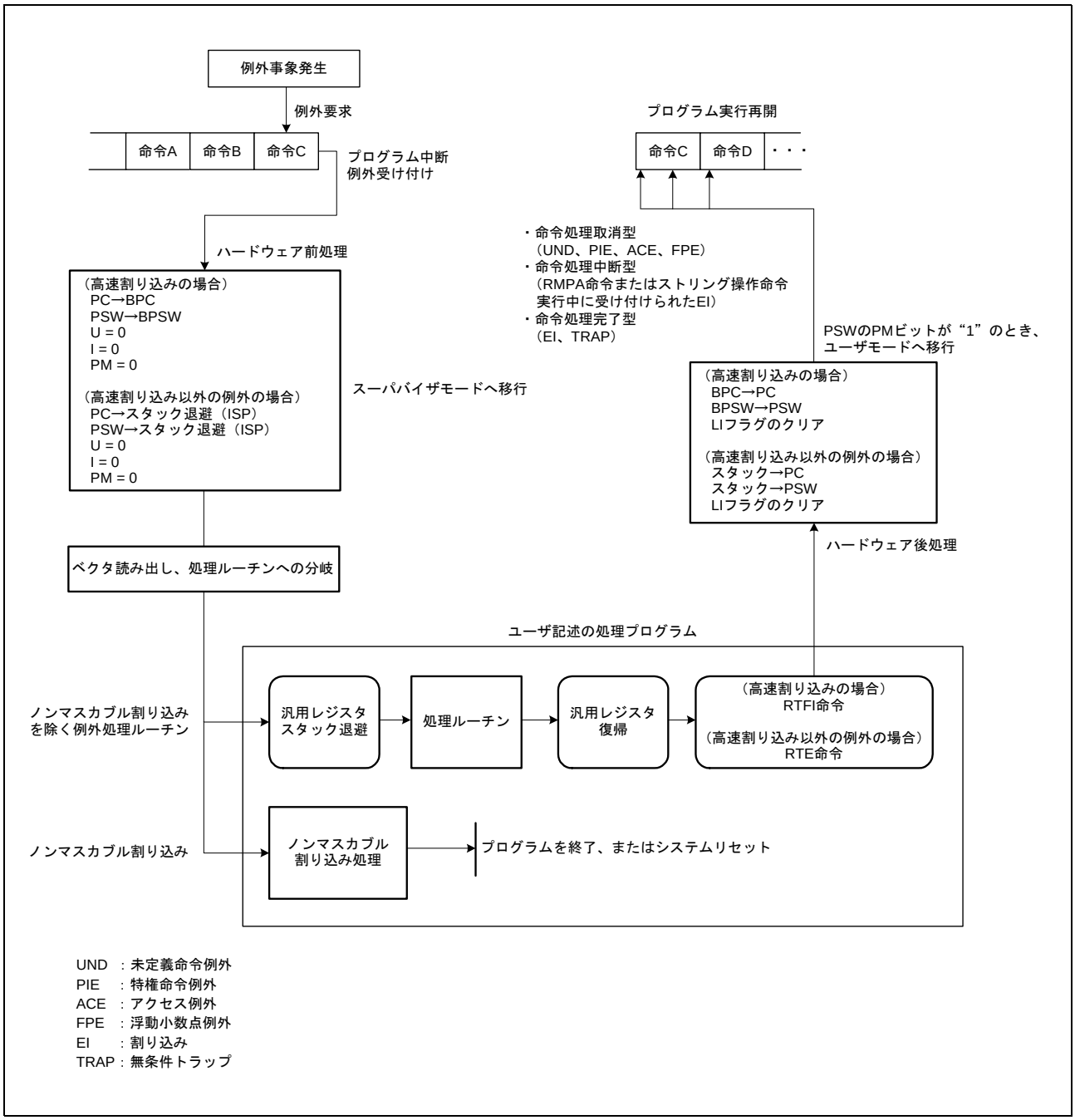


図 5.2　例外処理手順の概要

例外が受け付けられると、RXv2 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。

RXv2 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）へ退避します。高速割り込み以外の例外では、PC、PSW をスタック領域に退避します。例外処理ルーチン中で

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プログラムによってスタックに退避してください。

例外処理ルーチン処理完了後、スタックに退避したレジスタを復帰して 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。ただし、ノンマスカ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。

RXv2 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、スタック領域から PC、PSW の値を復帰します。

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
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
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 →スタック領域

注 . FPSW は、ハードウェア前処理では退避されません。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ユーザがスタックへ退避してください。

(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 →スタック領域

(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、例外処理ルーチン処理へ移行します。

(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）
スタック領域→ 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）
スタック領域→ 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）の内容をスタック領域（ISP）に退避します。
(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）の内容をスタック領域（ISP）に退避します。
(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）の内容をスタック領域（ISP）に退避します。
(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）の内容をスタック領域（ISP）に退避します。
(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）の内容をスタック領域（ISP）に退避します。
(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）の内容をスタック領域（ISP）に退避します。
(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 浮動小数点例外

(1) プロセッサステータスワード（PSW）の内容をスタック領域（ISP）に退避します。
(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）の内容をスタック領域（ISP）に退避します。
(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。

5.5.5 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。

5.5.6 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）の内容をスタック領域（ISP）に退避します。
(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 命令を実行中は、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 の内容をスタック領域（ISP）に退避します。
(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。

5.5.7 割り込み

(1) プロセッサステータスワード（PSW）の内容をスタック領域（ISP）に退避します。高速割り込みの場合は、バックアップ PSW（BPSW）に退避します。
(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 命令を実行中は、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 の内容をスタック領域（ISP）に退避します。高速割り込みの場合は、バックアップ PC（BPC）に退避します。
(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。

5.5.8 無条件トラップ

(1) プロセッサステータスワード（PSW）の内容をスタック領域（ISP）に退避します。
(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）の内容をスタック領域（ISP）に退避します。
(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、例外処理シーケンス直前にスタック領域または制御レジスタ（BPC、BPSW）に退避さ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
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		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 ↑	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	オペランドアクセス例外
低い	8	浮動小数点例外

索引

記号

B

C

D

E

F

I

N

O

P

Q

R

S

U

Z

あ

い

お

き

こ

さ

す

せ

そ

と

の

は

ひ

ふ

へ

ほ

ま

み

む

ゆ

り

れ

わ

改訂記録	RX ファミリ RXv2 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	2013.11.12	—	初版発行

RX ファミリ RXv2 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　2013 年 11 月 12 日　Rev.1.00

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

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所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 〒100-0004 千代田区大手町2-6-2（日本ビル）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/contact/

RX ファミリ RXv2 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

R01US0071JJ0100