User's Manual

32

RXファミリ RXv3e命令セットアーキテクチャ

ユーザーズマニュアル　ソフトウェア編

ルネサス32ビットマイクロコンピュータ

RXファミリ

本資料に記載の全ての情報は本資料発行時点のものであり、ルネサス エレクトロニクスは、
予告なしに、本資料に記載した製品または仕様を変更することがあります。
ルネサス エレクトロニクスのホームページなどにより公開される最新情報をご確認ください。

ルネサス エレクトロニクス
www.renesas.com

Rev.1.01　2026.08

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。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、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

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

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

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 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。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

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

(Rev.5.0-1 2020.10)

本社所在地

〒 135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。

すべての商標および登録商標は、それぞれの所有者に帰属します。

© 2026 Renesas Electronics Corporation. All rights reserved.

このマニュアルの使い方

　このマニュアルは、RXv3e 命令セットアーキテクチャ（RXv3e）をユーザに理解していただくためのマニュアルです。このマニュアルでは、CPU の特長や命令体系について詳細に記載しています。本 CPU を用いた応用システムを設計するユーザを対象にしています。このマニュアルを使用するには、マイクロコンピュータに関する基本的な知識が必要です。

略号について

本マニュアルで使用する略号について、以下に説明します。

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

分類	略号	意味
数値	000b	2進数を表します。
	0000h	16進数を表します。
ビット長	#IMM: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。
	.D	ダブルロングワード（64ビット）を指定します。
分岐距離指定	BRA.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。

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

目次

このマニュアルの使い方 3

RX ファミリ RXv3e 命令セット一覧 11

アルファベット順ページ早見表 11

1. CPU プログラミングモデル 16

1.1 特長 16

1.2 Protect Zone 技術 17

1.3 CPU レジスタセット 18

1.3.1 汎用レジスタ（R0 ～ R15） 19

1.3.2 制御レジスタ 19

1.3.2.1 制御レジスタのアクセス制限 20

1.3.2.2 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） 22

1.3.2.3 割り込みテーブルレジスタ（INTB） 22

1.3.2.4 プログラムカウンタ（PC） 22

1.3.2.5 プロセッサステータスワード（PSW） 23

1.3.2.6 バックアップ PC（BPC） 26

1.3.2.7 バックアップ PSW（BPSW） 26

1.3.2.8 高速割り込みベクタレジスタ（FINTV） 26

1.3.2.9 単精度浮動小数点ステータスワード（FPSW） 27

1.3.2.10 例外テーブルレジスタ（EXTB） 29

1.3.2.11 割り込みベクタ制御レジスタ（IVC） 30

1.3.2.12 ワールド ID レジスタ（WID） 32

1.3.2.13 ルートコールベクタテーブルレジスタ（TRCTB） 33

1.3.2.14 ワールド遷移割り込みベクタテーブルレジスタ（WINTB） 33

1.3.2.15 ワールド共通プロセッサ割り込み優先レベルレジスタ（WPRI） 34

1.3.3 アキュムレータ 36

1.4 単精度浮動小数点例外 37

1.4.1 オーバフロー 37

1.4.2 アンダフロー 37

1.4.3 精度異常 38

1.4.4 ゼロ除算 38

1.4.5 無効演算 39

1.4.6 非実装処理 39

1.5 プロセッサモード 40

1.5.1 スーパバイザモード 40

1.5.2 ユーザモード 40

1.5.3 特権命令 40

1.5.4 プロセッサモード間の移行 41

1.6 データタイプ 42

1.6.1 整数 42

1.6.2 単精度浮動小数点数 42

1.6.3 ビット 43

1.6.4 ストリング 43
1.7 データ配置 44
1.7.1 レジスタのデータ配置 44
1.7.2 メモリ上のデータ配置 44
1.8 ベクタテーブル 45
1.8.1 例外ベクタテーブル 45
1.8.2 割り込みベクタテーブル 46
1.8.3 ルートコールベクタテーブル 47
1.8.4 ワールド遷移割り込みベクタテーブル 47
1.9 アドレス空間 48
1.10 レジスタ一括退避機能 49
1.11 倍精度浮動小数点コプロセッサ 51
1.11.1 特長 51
1.11.2 倍精度浮動小数点レジスタセット 51
1.11.2.1 倍精度浮動小数点データレジスタ（DR0 ～ DR15） 52
1.11.2.2 倍精度浮動小数点制御レジスタ 52
1.11.3 倍精度浮動小数点例外 57
1.11.3.1 オーバフロー 57
1.11.3.2 アンダフロー 57
1.11.3.3 精度異常 58
1.11.3.4 ゼロ除算 58
1.11.3.5 無効演算 59
1.11.3.6 非実装処理 59
1.11.4 データタイプ（倍精度浮動小数点コプロセッサ） 60
1.11.4.1 倍精度浮動小数点数 60
1.11.5 データ配置（倍精度浮動小数点コプロセッサ） 61
1.11.5.1 倍精度浮動小数点レジスタのデータ配置 61
1.11.5.2 メモリ上の倍精度浮動小数点データ配置 61
2. アドレッシングモード 62
2.1 本章の見方 63
2.2 アドレッシングモード 64
2.2.1 IMM の範囲 69
3. 命令 70
3.1 命令セット概要 70
命令一覧表 70
3.2 RXv3e 拡張命令一覧 75
3.2.1 RXv3e 新規追加命令 75
3.2.2 RXv3e 仕様拡張命令 76
3.3 本章の見方 77
3.4 命令詳細説明 83
3.4.1 標準搭載命令 83

3.4.2 レジスタ一括退避機能命令 226
3.4.3 倍精度浮動小数点処理命令 229
3.4.4 Protect Zone 命令 265
4. 命令コード 270
4.1 本章の見方 270
4.2 命令コード詳細説明 273
4.2.1 標準搭載命令 273
4.2.2 レジスタ一括退避機能命令 361
4.2.3 倍精度浮動小数点処理命令 363
4.2.4 Protect Zone 命令 377
5. 例外処理 382
5.1 例外事象 382
5.1.1 未定義命令例外 383
5.1.2 特権命令例外 383
5.1.3 アクセス例外 383
5.1.4 セキュア例外 383
5.1.5 アドレス例外 383
5.1.6 単精度浮動小数点例外 383
5.1.7 リセット 383
5.1.8 ノンマスカブル割り込み 384
5.1.9 割り込み 384
5.1.10 無条件トラップ 385
5.2 例外の処理手順 386
5.2.1 ワールド遷移なしの例外の処理手順 387
5.2.2 ワールド遷移ありの例外の処理手順 389
5.3 例外事象の受け付け 391
5.3.1 受け付けタイミングと保存される PC 値 391
5.3.2 ベクタと PC、PSW の退避場所 391
5.4 例外の受け付け / 復帰時のハードウェア処理 393
5.4.1 ワールド遷移なしの例外の受け付け / 復帰時のハードウェア処理 393
5.4.2 ワールド遷移ありの例外の受け付け / 復帰時のハードウェア処理 394
5.5 ハードウェア前処理 395
5.5.1 未定義命令例外 395
5.5.2 特権命令例外 395
5.5.3 アクセス例外 395
5.5.4 セキュア例外 395
5.5.5 アドレス例外 396
5.5.6 単精度浮動小数点例外 396
5.5.7 リセット 396
5.5.8 ノンマスカブル割り込み（通常受付） 396
5.5.9 ノンマスカブル割り込み（ワールド遷移受付） 396

5.5.10 割り込み（通常受付）..... 397
5.5.11 割り込み（ワールド遷移受付）..... 397
5.5.12 無条件トラップ（通常受付）..... 397
5.5.13 無条件トラップ（ワールド遷移受付）..... 398
5.6 例外処理ルーチンからの復帰..... 399
5.7 例外事象の優先順位..... 399
5.8 コプロセッサの例外事象..... 400
5.8.1 倍精度浮動小数点例外..... 400
索引..... 401
改訂記録..... 405

RX ファミリ RXv3e 命令セット一覧

アルファベット順ページ早見表 (1 / 5)

ニーモニック		機能	命令詳細記載ページ	命令コード詳細記載ページ
ABS		絶対値	84	274
ADC		キャリー付き加算	85	275
ADD		キャリーなし加算	86	276
AND		論理積	88	278
BCLR		ビットクリア	89	280
B*Cnd*	BGEU	相対条件分岐	90	282
	BC		90	282
	BEQ		90	282
	BZ		90	282
	BGTU		90	282
	BPZ		90	282
	BGE		90	282
	BGT		90	282
	BO		90	282
	BLTU		90	282
	BNC		90	282
	BNE		90	282
	BNZ		90	282
	BLEU		90	282
	BN		90	282
	BLE		90	282
	BLT		90	282
	BNO		90	282
BFMOV		ビットフィールド転送	91	283
BFMOVZ		ゼロへのビットフィールド転送	92	283
BM*Cnd*	BMGEU	条件ビット転送	93	284
	BMC		93	284
	BMEQ		93	284
	BMZ		93	284
	BMGTU		93	284
	BMPZ		93	284
	BMGE		93	284
	BMGT		93	284
	BMO		93	284
	BMLTU		93	284
	BMNC		93	284
	BMNE		93	284
	BMNZ		93	284
	BMLEU		93	284
	BMN		93	284
	BMLE		93	284
	BMLT		93	284
	BMNO		93	284

アルファベット順ページ早見表 (2 / 5)

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
BNOT		ビット反転	95	285
BRA		相対無条件分岐	96	286
BRK		無条件トラップ	97	287
BSET		ビットセット	98	287
BSR		相対サブルーチン分岐	99	289
BTST		ビットテスト	100	290
CLRPSW		PSWのフラグ、ビットのクリア	101	291
CMP		比較	102	292
DABS		倍精度浮動小数点絶対値	230	364
DADD		倍精度浮動小数点加算	231	364
DCMPcm	DCMPUN	倍精度浮動小数点比較	233	365
	DCMPEQ		233	365
	DCMPLT		233	365
	DCMPLE		233	365
DDIV		倍精度浮動小数点除算	235	365
DIV		符号付き除算	103	293
DIVU		符号なし除算	104	295
DMOV		倍精度浮動小数点データ転送	237	366
DMUL		倍精度浮動小数点乗算	239	369
DNEG		倍精度浮動小数点符号反転	241	369
DPOPM		複数の倍精度浮動小数点レジスタの復帰	242	370
DPUSHM		複数の倍精度浮動小数点レジスタの退避	244	371
DROUND		倍精度浮動小数点数→符号付き整数変換	246	372
DSQRT		倍精度浮動小数点平方根	248	372
DSUB		倍精度浮動小数点減算	250	372
DTOF		倍精度浮動小数点数→単精度浮動小数点数変換	252	373
DTOI		倍精度浮動小数点数→符号付き整数変換	254	373
DTOU		倍精度浮動小数点数→符号なし整数変換	256	373
EMACA		32ビット積和演算	105	296
EMSBA		32ビット積差演算	106	296
EMUL		符号付き乗算	107	297
EMULA		32ビット乗算	109	298
EMULU		符号なし乗算	110	298
FADD		単精度浮動小数点加算	112	300
FCMP		単精度浮動小数点比較	115	301
FDIV		単精度浮動小数点除算	117	302
FMUL		単精度浮動小数点乗算	119	303
FSQRT		単精度浮動小数点平方根	122	304
FSUB		単精度浮動小数点減算	124	305
FTOD		単精度浮動小数点数→倍精度浮動小数点数変換	258	374
FTOI		単精度浮動小数点数→符号付き整数変換	127	306
FTOU		単精度浮動小数点数→符号なし整数変換	129	306
INT		ソフトウェア割り込み	131	307
ITOD		符号付き整数→倍精度浮動小数点数変換	260	374
ITOF		符号付き整数→単精度浮動小数点数変換	132	308
JMP		無条件分岐	134	309

アルファベット順ページ早見表 (3 / 5)

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
JSR	サブルーチン分岐	135	309
MACHI	上位16ビット積和演算	136	310
MACLH	下位16ビット・上位16ビット積和演算	137	310
MACLO	下位16ビット積和演算	138	310
MAX	最大値選択	139	311
MIN	最小値選択	140	312
MOV	転送	141	313
MOVCO	LIフラグクリア付きストア	145	318
MOVLI	LIフラグセット付きロード	146	318
MOVU	符号なしデータ転送	147	319
MSBHI	上位16ビット積差演算	148	320
MSBLH	下位16ビット・上位16ビット積差演算	149	320
MSBLO	下位16ビット積差演算	150	321
MUL	乗算	151	321
MULHI	上位16ビット乗算	153	323
MULLH	下位16ビット・上位16ビット乗算	154	323
MULLO	下位16ビット乗算	155	324
MVFACGU	アキュムレータガードビットからの転送	156	324
MVFACHI	アキュムレータ上位32ビットからの転送	157	325
MVFACLO	アキュムレータ下位32ビットからの転送	158	325
MVFACMI	アキュムレータ中央32ビットからの転送	159	326
MVFC	制御レジスタからの転送	160	327
MVFDC	倍精度浮動小数点制御レジスタからの転送	261	375
MVFDR	倍精度浮動小数点比較結果レジスタからの転送	262	375
MVTACGU	アキュムレータガードビットへの転送	161	328
MVTACHI	アキュムレータ上位32ビットへの転送	162	328
MVTACLO	アキュムレータ下位32ビットへの転送	163	328
MVTC	制御レジスタへの転送	164	329
MVTDC	倍精度浮動小数点制御レジスタへの転送	263	376
MVTIPL（特権命令）	割り込み優先レベル設定	165	330
NEG	符号反転	166	331
NOP	ノーオペレーション	167	331
NOT	論理反転	168	332
OR	論理和	169	333
POP	スタックからレジスタへのデータ復帰	170	334
POPC	制御レジスタの復帰	171	335
POPM	複数レジスタの復帰	172	335
PUSH	スタックへデータ退避	173	336
PUSHC	制御レジスタの退避	174	337
PUSHM	複数レジスタの退避	175	337
RACL	符号付きアキュムレータ丸め処理	176	338
RACW	16ビット符号付きアキュムレータ丸め処理	178	338
RDACL	符号付きアキュムレータ丸め処理	180	339
RDACW	16ビット符号付きアキュムレータ丸め処理	182	339
REVL	エンディアン変換	184	340

アルファベット順ページ早見表 (4 / 5)

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
REVW		エンディアン変換	185	340
RMPA		積和演算	186	341
ROLC		キャリー付き左回転	188	341
RORC		キャリー付き右回転	189	341
ROTL		左回転	190	342
ROTR		右回転	191	342
ROUND		単精度浮動小数点数→符号付き整数変換	192	343
RSTR（特権命令）		レジスタの一括復帰	227	362
RTE（特権命令）		例外からの復帰	195	343
RTFI（特権命令）		高速割り込みからの復帰	196	343
RTS		サブルーチンからの復帰	197	344
RTSD		スタックフレームの解放とサブルーチンからの復帰	198	344
SAT		32ビット符号付き飽和処理	200	344
SATR		RMPA命令用64ビット符号付き飽和処理	201	345
SAVE（特権命令）		レジスタの一括退避	228	362
SBB		ボロー付き減算	202	345
SC*Cnd*	SCGEU	条件設定	203	346
	SCC		203	346
	SCEQ		203	346
	SCZ		203	346
	SCGTU		203	346
	SCPZ		203	346
	SCGE		203	346
	SCGT		203	346
	SCO		203	346
	SCLTU		203	346
	SCNC		203	346
	SCNE		203	346
	SCNZ		203	346
	SCLEU		203	346
	SCN		203	346
	SCLE		203	346
	SCLT		203	346
	SCNO		203	346
SCMPU		ストリング比較	204	346
SETPSW		PSWのフラグ、ビットのセット	205	347
SHAR		算術右シフト	206	348
SHLL		論理/算術左シフト	207	349
SHLR		論理右シフト	208	350
SMOVB		逆方向ストリング転送	209	350
SMOVF		順方向ストリング転送	210	351
SMOVU		ストリング転送	211	351
SSTR		ストリングストア	212	351
STNZ		条件付き転送	213	352
STZ		条件付き転送	214	353
SUB		ボローなし減算	215	354

アルファベット順ページ早見表 (5 / 5)

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
SUNTIL	ストリングサーチ	216	355
SWHILE	ストリングサーチ	218	355
TRCALL（特権命令）	Rootワールドコール	266	378
TRMVFC（特権命令）	Rootワールドでの制御レジスタからの転送	267	378
TRMVTC（特権命令）	Rootワールドでの制御レジスタへの転送	268	379
TRRTE（特権命令）	Rootワールドの例外処理からの復帰	269	381
TST	テスト	220	356
UTOD	符号なし整数→倍精度浮動小数点数変換	264	376
UTOF	符号なし整数→単精度浮動小数点数変換	221	357
WAIT（特権命令）	ウェイト	223	358
XCHG	交換	224	358
XOR	排他的論理和	225	359

1. CPU プログラミングモデル

RXv3e 命令セットアーキテクチャ（RXv3e）は、RXv3 命令セットアーキテクチャ（RXv3）と上位互換性のある命令セットアーキテクチャです。

- Protect Zone
  CPU コアのセキュリティ状態を管理する Protect Zone 技術を搭載し、ソフトウェア実行環境の分離をサポートします。
- 可変長命令方式の採用
  可変長命令形式の採用により、使用頻度の高い命令をより短い命令長に割り付けていますので、コード効率の良いプログラムを開発できます。
- 強力な命令セット
  RXv3e は厳選された 117 個の命令をサポートしています。DSP 機能命令や浮動小数点演算命令により、DSP に匹敵するデータ処理能力を発揮します。
- 豊富なアドレッシングモード
  11 種類の豊富なアドレッシングモードを持ち、レジスタ – レジスタ間、レジスタ – メモリ間の演算や、ビットを対象とする演算ができます。また、メモリ – メモリ間の転送ができます。

1.1 特長

- 最短命令実行時間：1 サイクル
- アドレス空間：4G バイト・リニアアドレス
- Protect Zone：3 ワールド（Non-secure, Secure, Root）
- 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 × 15 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）
- 117 命令 /11 種類のアドレッシングモード
  標準搭載命令：111 命令
  　基本命令：77 命令
  　単精度浮動小数点演算命令：11 命令
  　DSP 機能命令：23 命令
  レジスタ一括退避機能命令：2 命令（オプション）
  Protect Zone 命令：4 命令
- 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、ルートコールベクタテーブル、ワールド遷移割り込みベクタテーブル
- メモリプロテクションユニット（オプション機能）
- データ配置
  リトルエンディアン / ビッグエンディアン選択可能
- 倍精度浮動小数点コプロセッサ搭載（オプション機能）
  倍精度浮動小数点処理命令：21 命令

1.2　Protect Zone 技術

Protect Zone 技術を採用した CPU コアはワールドと呼ぶ 3 つのセキュリティ状態を持ちます。各ワールドはワールド ID で区別されます。ワールド ID 0 を Root ワールド、ワールド ID 1 を Secure ワールド、ワールド ID 2 を Non-secure ワールドと呼びます。

以降、対象が各々のワールドであることが明らかな場合に「ワールド」を省き単に「Root」「Secure」「Non-secure」と表記する場合があります。

Secure ワールドと Non-secure ワールドで実行されるプログラムとアクセスできるリソースは分離することが可能です。Secure ワールドと Non-secure ワールド間は直接遷移することはできず、必ず Root ワールド経由で遷移します。制御レジスタなど CPU コアの一部のリソースは CPU コアのハードウェアによって分離されますが、それ以外のリソースは Root ワールドのプログラムによって分離され得ます。

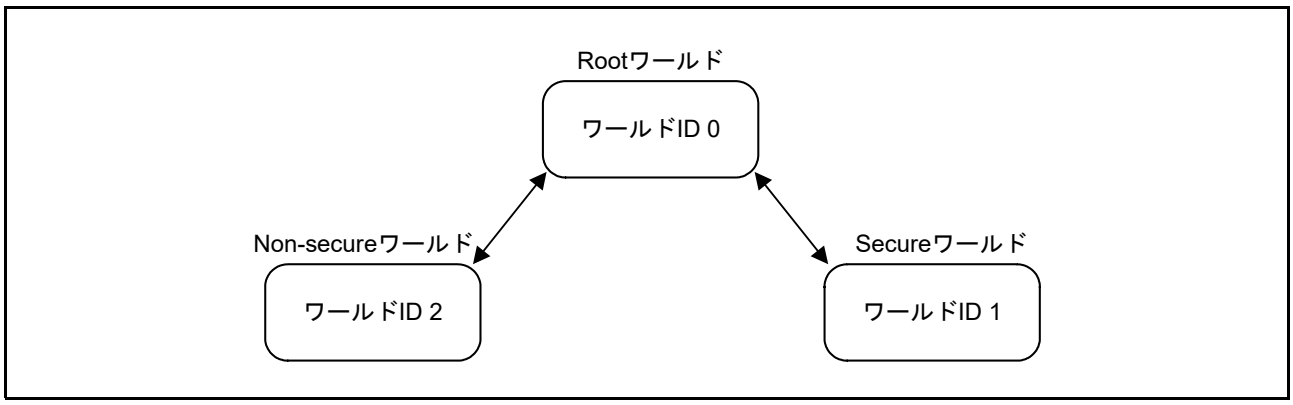


図 1.1　Protect Zone の 3 つのワールド

リセット解除後のワールドは Root ワールドです。

命令セットアーキテクチャとしては、主に CPU コアに関する仕様を規定します。Protect Zone に対応する割り込みやアドレス空間の仕様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ください。

Root・Secure・Non-secure の各ワールドはそれぞれ用途が異なります。

Root ワールドは、Non-secure、Secure へ遷移する起点となるワールドです。他のワールドへ制御を移すには必ず Root ワールドを介す必要があり、ワールド遷移に関するソフトウェア処理はこのワールドで行います。このために、Root ワールドでのみ実行可能な命令を規定しています。Root ワールドでは、Root ワールドと Secure ワールド、Non-secure ワールドのリソースにアクセスできます。

Non-secure ワールドは、Non-secure なアプリケーションが実行されることを想定したワールドです。Non-secure ワールドでは、Non-secure ワールドのリソースにのみアクセスできます。

Secure ワールドは、Secure なアプリケーションが実行されることを想定したワールドです。Secure ワールドでは、Secure ワールドと Non-secure ワールドのリソースにのみアクセスできます。

RXv3 用のプログラムのように Protect Zone のワールドの概念を意識せずにプログラムを動作させることも可能です。そのようなソフトウェアプロジェクトを Flat-Project と呼びます。Flat-Project では、Root ワールドのみを動作させることで、他のワールドを意識せずプログラムを動作させることができます。

CPU に入力される割り込み要求は、どのワールド宛の割り込み要求かの情報（宛先ワールド）を持ちます。Root ワールド、Secure ワールドを宛先とした割り込みの割り込み優先レベルは 1 ～ 31 が使用可能です。Non-secure ワールドを宛先とした割り込みの割り込み優先レベルは 1 ～ 15 が使用可能です。

高速割り込みは、Flat-Project でのみ使用可能です。

1.3 CPU レジスタセット

RXv3e CPUのレジスタには、汎用レジスタ（16本）と、制御レジスタ（15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 1.2　CPU レジスタセット

1.3.1 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。

1.3.2 制御レジスタ

制御レジスタには、以下の 15 本のレジスタがあります。

- 割り込みスタックポインタ（ISP）(注 1)
- 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）(注 1)
- 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）(注 1)
- バックアップ PSW（BPSW）(注 1)
- 高速割り込みベクタレジスタ（FINTV）
- 単精度浮動小数点ステータスワード（FPSW）
- 例外テーブルレジスタ（EXTB）(注 1)
- 割り込みベクタ制御レジスタ（IVC）(注 1)
- ワールド ID レジスタ（WID）
- ルートコールベクタテーブルレジスタ（TRCTB）
- ワールド遷移割り込みベクタテーブルレジスタ（WINTB）
- ワールド共通プロセッサ割り込み優先レベルレジスタ（WPRI）

注 1. 各制御レジスタの実体は、Protect Zone 対応として、Root 用（_TR）、Secure 用（_S）、Non-secure 用（_NS）と表記されるレジスタがあります。_TR、_S、_NS の記載がない場合、特に明記されない限り、現在実行中のワールドの実体を指します。

1.3.2.1　制御レジスタのアクセス制限

各制御レジスタのアクセス制限を**表 1.1** に示します。

ここに記載しているアクセス制限は、制御レジスタ内の少なくとも 1 つのフィールドにアクセス可能な特権・ワールドを記載しています。各制御レジスタはフィールド毎に異なるアクセス制限が設けられているものがあります。詳細は各制御レジスタの説明を参照してください。

Root・Secure・Non-secure でそれぞれ異なる実体を持つ制御レジスタは MVTC/MVFC/PUSHC/POPC 命令で他のワールドの実体にアクセスすることはできません。ただし、Root ワールドの場合のみ TRMVTC/TRMVFC 命令で他のワールドの実体にアクセスが可能です。

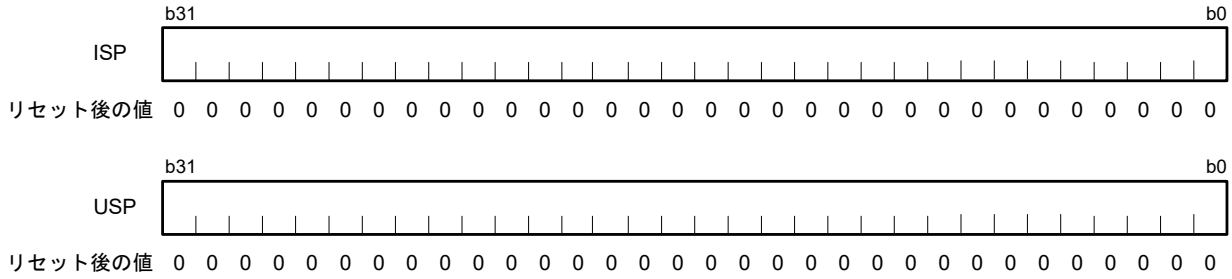
制御レジスタは表の命令・アクセス元ワールド・プロセッサモードのアクセス制限で許可される場合のみアクセスできます。ライトアクセスの条件を満たさない場合、書き込みは行われません。また、リードアクセスの条件を満たさない場合、“0” が読みだされます。ただし、TRMVTC/TRMVFC 命令は現在のワールドが Root 以外の場合やプロセッサモードがユーザモードの場合、例外が発生します。

表 1.1　制御レジスタアクセス制限

制御レジスタ名	CR	命令・アクセス元ワールド・プロセッサモード						
		命令群 1	命令群 2					
		Root	Root		Secure		Non-secure	
		スーパバイザ	スーパバイザ	ユーザ	スーパバイザ	ユーザ	スーパバイザ	ユーザ
PSW	0	—	○	○	○	○	○	○
PC	1	—	R	R	R	R	R	R
USP	2	—	○	○	○	○	○	○
FPSW	3	—	○	○	○	○	○	○
IVC_TR	5	○	○	R	—	—	—	—
IVC_S		○	—	—	R	R	—	—
IVC_NS		○	—	—	—	—	R	R
BPSW_TR	8	○	○	R	—	—	—	—
BPSW_S		○	—	—	○	R	—	—
BPSW_NS		○	—	—	—	—	○	R
BPC_TR	9	○	○	R	—	—	—	—
BPC_S		○	—	—	○	R	—	—
BPC_NS		○	—	—	—	—	○	R
ISP_TR	10	○	○	R	—	—	—	—
ISP_S		○	—	—	○	R	—	—
ISP_NS		○	—	—	—	—	○	R
FINTV	11	—	○	R	—	—	—	—
INTB_TR	12	○	○	R	—	—	—	—
INTB_S		○	—	—	○	R	—	—
INTB_NS		○	—	—	—	—	○	R
EXTB_TR	13	○	○	R	—	—	—	—
EXTB_S		○	—	—	○	R	—	—
EXTB_NS		○	—	—	—	—	○	R
WPRI	14	—	○	R	○	R	—	—
TRCTB	14	○	—	—	—	—	—	—
WID	15	—	○	R	R	R	R	R
WINTB	15	○	—	—	—	—	—	—

CR 欄は MVTC/MVFC/TRMVTC/TRMVFC/PUSHC/POPC 命令でアクセスする場合の cr フィールドの値です。
命令群 1：TRMVTC/TRMVFC 命令
命令群 2：MVTC/MVFC/PUSHC/POPC 命令
○： リード・ライト可能
R： リード可能・ライト不可
—： アクセス不可

1.3.2.2　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）

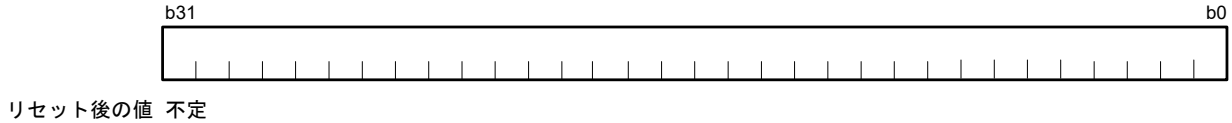


スタックポインタ（SP）には、割り込みスタックポインタ（ISP）と、ユーザスタックポインタ（USP）の 2 種類があります。使用するスタックポインタ（ISP/USP）は、プロセッサステータスワード（PSW）のスタックポインタ指定ビット（U）によって切り替えられます。

ISP には、Protect Zone 対応として以下の実体が存在します。

- Root 用：ISP_TR（Root ワールドでのみアクセスできます）
- Secure 用：ISP_S（Secure ワールド、Root ワールドでのみアクセスできます）
- Non-secure 用：ISP_NS（Non-secure ワールド、Root ワールドでのみアクセスできます）

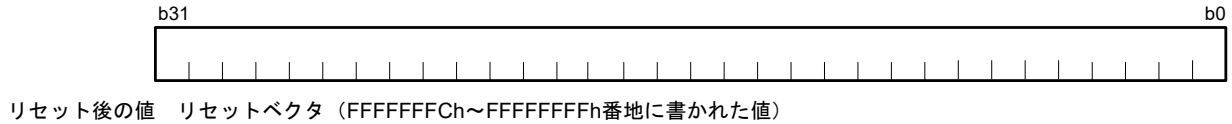
1.3.2.3　割り込みテーブルレジスタ（INTB）



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。このレジスタは、Protect Zone 対応として以下の実体レジスタが存在します。

- Root 用：INTB_TR（Root ワールドでのみアクセスできます）
- Secure 用：INTB_S（Secure ワールド、Root ワールドでのみアクセスできます）
- Non-secure 用：INTB_NS（Non-secure ワールド、Root ワールドでのみアクセスできます）

1.3.2.4　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。

1.3.2.5　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[4:0]					–	–	–	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	キャリーフラグ	0：キャリーの発生なし 1：キャリーの発生あり	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	読むと“0”が読めます。書く場合、“0”としてください	R/W
b16	I (注1)	割り込み許可ビット	0：表1.2を参照 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	読むと“0”が読めます。書く場合、“0”としてください	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	読むと“0”が読めます。書く場合、“0”としてください	R/W

ビット	シンボル	ビット名	機能	R/W
b28–b24	IPL[4:0] (注1、4)	プロセッサ割り込み優先レベル	b28 b24 0 0 0 0 0：優先レベル0（最低） 0 0 0 0 1：優先レベル1 0 0 0 1 0：優先レベル2 0 0 0 1 1：優先レベル3 0 0 1 0 0：優先レベル4 0 0 1 0 1：優先レベル5 0 0 1 1 0：優先レベル6 0 0 1 1 1：優先レベル7 0 1 0 0 0：優先レベル8 0 1 0 0 1：優先レベル9 0 1 0 1 0：優先レベル10 0 1 0 1 1：優先レベル11 0 1 1 0 0：優先レベル12 0 1 1 0 1：優先レベル13 0 1 1 1 0：優先レベル14 0 1 1 1 1：優先レベル15（Non-secure最高） 1 0 0 0 0：優先レベル16 1 0 0 0 1：優先レベル17 1 0 0 1 0：優先レベル18 1 0 0 1 1：優先レベル19 1 0 1 0 0：優先レベル20 1 0 1 0 1：優先レベル21 1 0 1 1 0：優先レベル22 1 0 1 1 1：優先レベル23 1 1 0 0 0：優先レベル24 1 1 0 0 1：優先レベル25 1 1 0 1 0：優先レベル26 1 1 0 1 1：優先レベル27 1 1 1 0 0：優先レベル28 1 1 1 0 1：優先レベル29 1 1 1 1 0：優先レベル30 1 1 1 1 1：優先レベル31（最高）	R/W
b31–b29	—	予約ビット	読むと“0”が読めます。書く場合、“0”としてください	R/W

注1.　ユーザモードのときは、MVTC、POPC命令によるIPL[4:0]、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[4:0]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、スタック上のPSWのPMビットを“1”にした後、RTE命令を実行するか、バックアップPSW（BPSW）のPMビットを“1”にした後、RTFI命令を実行してください。

注4.　Non-secureワールドでMVTC、POPC、MVTIPL命令によりIPL[4:0]へ書き込みを行おうとした場合、IPL[4]ビットへは常に“0”が書き込まれます。
Non-secureワールドでは、RTFI命令を実行した場合、IPL[4]ビットへは常に“0”が書き込まれます。

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（キャリーフラグ）

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。Non-secure ワールド実行中は、このビットが “0” であっても割り込みベクタ制御レジスタ（IVC）の NSIC ビットと割り込み要求の割り込み優先レベルによって、**表 1.2** の条件を満たしたときに割り込みを受け付けます。

表 1.2　　割り込み受付条件

実行中のワールド	I ビット	IVC.NSIC	IPR	割り込み許可/禁止
Root or Secure	0	—	—	禁止
	1	—	—	許可
Non-secure	0	0	—	禁止
		1	16 未満	禁止
			16 以上	許可
	1	—	—	許可

IPR：割り込み要求の割り込み優先レベル

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[4:0] ビット（プロセッサ割り込み優先レベル）

Root ワールド、Secure ワールドにおいて IPL[4:0] ビットは、優先レベル 0（最低）～優先レベル 31（最高）までの 32 段階のプロセッサ割り込み優先レベルを指定します。Non-secure ワールドでは、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。Non-secure ワールドでは優先レベル 16 以上は指定できません。

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。Root ワールド、Secure ワールドにおいて IPL[4:0] ビットをレベル 31（1Fh）に設定したとき、すべての割り込みが禁止されます。Non-secure ワールドでは、IPL[4:0] ビットをレベル 15（0Fh）に設定したときは、すべての Non-secure ワールド宛の割り込みが禁止されます。IPL[4:0] ビットは、Root ワールド、Secure ワールド宛のノンマスカブル割り込みが発生したとき、レベル 31（1Fh）になります。Non-secure ワールドにおいて処理中に Non-secure ワールド宛のノンマスカブル割り込みが発生したとき、レベル 15（0Fh）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。

1.3.2.6 バックアップ PC（BPC）

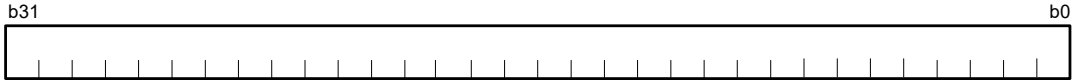


リセット後の値 不定

バックアップ PC（BPC）は、割り込み応答を高速化するために設けられたレジスタです。高速割り込みが発生すると、プログラムカウンタ（PC）の内容が BPC に退避させられます。このレジスタは、Protect Zone 対応として以下の実体レジスタが存在します。

- Root 用：BPC_TR（Root ワールドでのみアクセスできます）
- Secure 用：BPC_S（Secure ワールド、Root ワールドでのみアクセスできます）
- Non-secure 用：BPC_NS（Non-secure ワールド、Root ワールドでのみアクセスできます）

1.3.2.7 バックアップ PSW（BPSW）



リセット後の値 不定

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

- Root 用：BPSW_TR（Root ワールドでのみアクセスできます）
- Secure 用：BPSW_S（Secure ワールド、Root ワールドでのみアクセスできます）
- Non-secure 用：BPSW_NS（Non-secure ワールド、Root ワールドでのみアクセスできます）

BPSW_NS の b28 に値を書き込むと、“0” が書けます。

1.3.2.8 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。このレジスタには、Root ワールドでのみアクセスできます。

1.3.2.9　単精度浮動小数点ステータスワード（FPSW）

	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	FS	FX	FU	FZ	FO	FV	—	—	—	—	—	—	—	—	—	—
リセット後の値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	EX	EU	EZ	EO	EV	—	DN	CE	CX	CU	CZ	CO	CV	RM[1:0]	
リセット後の値	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

ビット	シンボル	ビット名	機能	R/W
b1–b0	RM[1:0]	単精度浮動小数点丸めモード設定ビット	b1 b0 0 0：最近値への丸め 0 1：0方向への丸め 1 0：+∞方向への丸め 1 1：−∞方向への丸め	R/W
b2	CV	無効演算要因フラグ	0：無効演算の発生なし 1：無効演算の発生あり	R/(W) (注1)
b3	CO	オーバフロー要因フラグ	0：オーバフローの発生なし 1：オーバフローの発生あり	R/(W) (注1)
b4	CZ	ゼロ除算要因フラグ	0：ゼロ除算の発生なし 1：ゼロ除算の発生あり	R/(W) (注1)
b5	CU	アンダフロー要因フラグ	0：アンダフローの発生なし 1：アンダフローの発生あり	R/(W) (注1)
b6	CX	精度異常要因フラグ	0：精度異常の発生なし 1：精度異常の発生あり	R/(W) (注1)
b7	CE	非実装処理要因フラグ	0：非実装処理の発生なし 1：非実装処理の発生あり	R/(W) (注1)
b8	DN	非正規化数の0フラッシュビット	0：非正規化数を非正規化数として扱う 1：非正規化数を0として扱う(注2)	R/W
b9	—	予約ビット	読むと“0”が読めます。書く場合、“0”としてください	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	読むと“0”が読めます。書く場合、“0”としてください	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
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

ビット	シンボル	ビット名	機能	R/W
b31	FS	単精度浮動小数点エラーサマリフラグ	FU、FZ、FO、FVフラグの論理和を反映	R

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

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

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

RM[1:0] ビット（単精度浮動小数点丸めモード設定ビット）

単精度浮動小数点丸めモードを設定します。

【単精度浮動小数点丸めモードの説明】

- 最近値への丸め（デフォルト）　：無限の有効桁を持つと仮定して計算した結果に近い方の値へ丸める。中間時は結果が偶数になる方向へ丸める
- 0方向への丸め　：結果の絶対値が小さくなる方向へ丸める（単純な切り捨て）
- +∞方向への丸め　：結果の値が大きくなる方向へ丸める
- −∞方向への丸め　：結果の値が小さくなる方向へ丸める

(1)「最近値への丸め」はデフォルトのモードであり、最も正確な値を返します。
(2)「0方向への丸め」、「+∞方向への丸め」、「−∞方向への丸め」は、区間演算（Interval arithmetic）を使用した精度保証を行うときに使用します。

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

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

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

DN ビット（非正規化数の 0 フラッシュビット）

“0”のとき非正規化数を非正規化数として扱います。
“1”のとき非正規化数を0として扱います。

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

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

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

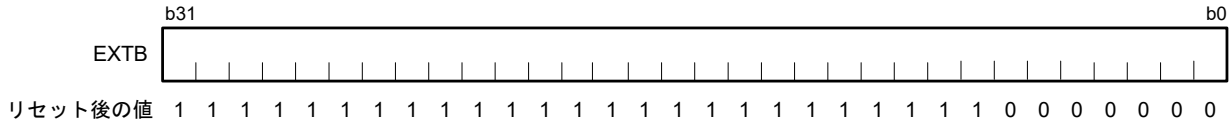
例外処理許可ビット Ej が “0”（例外処理を禁止）の場合、IEEE754 規格で規定された 5 つの例外が発生すると、該当するフラグが “1” になります。

- 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（単精度浮動小数点エラーサマリフラグ）

FU、FZ、FO、FV フラグの論理和を反映します。

1.3.2.10　例外テーブルレジスタ（EXTB）



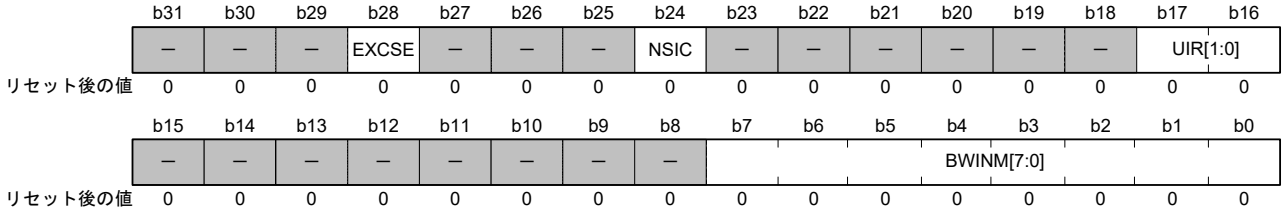
例外テーブルレジスタ（EXTB）には、例外ベクタテーブルの先頭番地を設定してください。このレジスタは、Protect Zone 対応として以下の実体レジスタが存在します。

- Root 用：EXTB_TR（Root ワールドでのみアクセスできます）
- Secure 用：EXTB_S（Secure ワールド、Root ワールドでのみアクセスできます）
- Non-secure 用：EXTB_NS（Non-secure ワールド、Root ワールドでのみアクセスできます）

初期値はワールドによって異なります。

- EXTB_TR, EXTB_S：　FFFF FF80h
- EXTB_NS：　　7FFF FF80h

1.3.2.11　割り込みベクタ制御レジスタ（IVC）



ビット	シンボル	ビット名	機能	R/W
b7-b0	BWINM[7:0]	バックアップワールド遷移割り込み要因番号	ワールド遷移受付された割り込みの要因番号を保持します	R/W
b15-b8	—	予約ビット	読むと“0”が読めます。書く場合、“0”としてください	R/W
b17-b16	UIR[1:0]	ソフトウェア割り込み制限ビット	ユーザモードでINT命令を実行した場合にオペランドに指定できる割り込み番号を制限します。範囲外の値を設定すると特権命令例外が発生します。スーパバイザモードではビットの指定に関わらず、0～255番が指定できます • INT #imm:8の場合 b17 b16 0　0：0～255番が指定可能 0　1：0～15番が指定可能 1　x：設定しないでください • INT Rsの場合 b17 b16 0　0：0～255番が指定可能 0　1：ソフトウェア割り込み実行不可 1　x：設定しないでください	R/W
b23-b18	—	予約ビット	読むと“0”が読めます。書く場合、“0”としてください	R/W
b24	NSIC	ノンセキュア割り込み許可制御ビット	Non-secureワールドにおけるPSW.Iビットの挙動を制御します 0：PSW.Iビット＝0の場合、割り込み禁止 1：PSW.Iビット＝0の場合、優先レベル15までの割り込み禁止	R/W (注1)
b27-b25	—	予約ビット	読むと“0”が読めます。書く場合、“0”としてください	R/W
b28	EXCSE	ハードウェア前処理中セキュア例外検出ビット	ハードウェア前処理中にセキュア例外が発生した場合にセットします 0：ハードウェア前処理中のセキュア例外発生なし 1：ハードウェア前処理中のセキュア例外発生あり	R/W (注1)
b31-b29	—	予約ビット	読むと“0”が読めます。書く場合、“0”としてください	R/W

注1.　IVC_NS、IVC_Sは、読むと“0”が読めます。

このレジスタは、Protect Zone 対応として以下の実体レジスタが存在します。

- Root 用：IVC_TR（Root ワールドでのみアクセスできます）
- Secure 用：IVC_S（Secure ワールド、Root ワールドでのみアクセスできます）
- Non-secure 用：IVC_NS（Non-secure ワールド、Root ワールドでのみアクセスできます）

これらの実体レジスタは、スーパバイザモードかつ Root ワールドで実行中のみ変更可能です。

BWINM[7:0] ビット（バックアップワールド遷移割り込み要因番号）

ワールド遷移受付された割り込みが発生したときに、IVC_TR.BWINM[7:0] ビットと割り込み飛び先ワー

ルドの IVC.BWINM[7:0] ビットに、割り込みベクタテーブルのベクタ番号を保持します。
IVC_TR.BWINM[7:0] ビットと割り込み宛先ワールドの BWINM[7:0] ビット（Secure の場合 IVC_S.BWINM[7:0] ビット、Non-secure の場合 IVC_NS.BWINM[7:0] ビット）に同じ値が書き込まれます。

UIR[1:0] ビット（ソフトウェア割り込み制限ビット）

ユーザモードで INT 命令のオペランドに指定できる割り込み番号を制限します。スーパバイザモードの場合は、本ビットは無視されます。

NSIC ビット（ノンセキュア割り込み許可制御ビット）

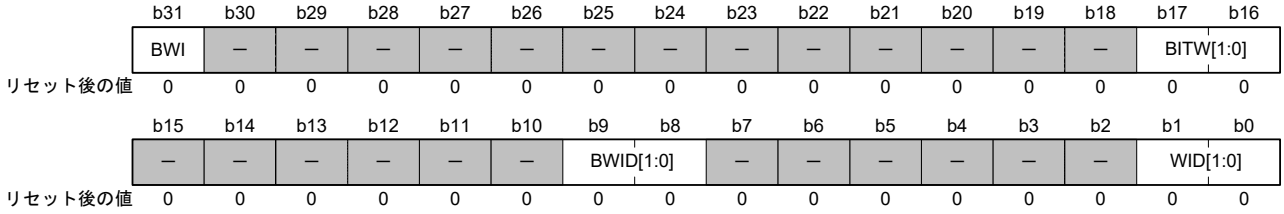
Non-secure ワールドにおける、割り込み優先レベルが 16 以上の割り込みに対する PSW.I ビットの動作を変更します。NSIC ビット = 1 の場合、割り込み優先レベルが 16 以上の割り込みは PSW.I ビットの設定によらず受け付けられます。割り込み優先レベルが 15 以下の割り込みは PSW.I ビットが “1” の場合のみ受け付けられます。NSIC ビット = 0 の場合、すべての優先レベルの割り込みは PSW.I ビットが “1” の場合のみ受け付けられます。

EXCSE ビット（ハードウェア前処理中セキュア例外検出ビット）

ハードウェア前処理中にセキュア例外が発生したことを示すビットです。

ハードウェア前処理中にセキュア例外が発生した場合、ハードウェア前処理が途中で中断されセキュア例外が受け付けられます。この場合、TRRTE 命令でセキュア例外から復帰することはできません。この状態が起こっていることをこのビットで確認することができます。

1.3.2.12　ワールドIDレジスタ（WID）



ビット	シンボル	ビット名	機能	R/W
b1-b0	WID[1:0]	ワールドID	現在実行中のワールドIDを保持します	R
b7-b2	—	予約ビット	読むと“0”が読めます。書く場合、“0”としてください	R/W
b9-b8	BWID[1:0]	バックアップワールドID	Rootワールドへ遷移時にバックアップしたワールドIDを保持します	R/W (注1)
b15-b10	—	予約ビット	読むと“0”が読めます。書く場合、“0”としてください	R/W
b17-b16	BITW[1:0]	バックアップワールド遷移先ワールドID	バックアップした割り込み（ワールド遷移受付）/ノンマスカブル割り込み（ワールド遷移受付）の宛先ワールドIDを保持します	R
b30-b18	—	予約ビット	読むと“0”が読めます。書く場合、“0”としてください	R/W
b31	BWI	バックアップワールド遷移元割り込み許可ビット	無条件トラップ（ワールド遷移受付）の遷移元に設定していた割り込み許可ビット（PSW.I）を保持します	R

注1.　Secure、Non-secureワールドでは、読むと“0”が読めます。

ワールドIDは以下のように定義されます。

00：Rootワールド

01：Secureワールド

10：Non-secureワールド

このレジスタは、スーパバイザモードかつRootワールドで実行中のみ変更可能です。Secureワールド、Non-secureワールドで実行中は参照のみ可能です。

1.3.2.13　ルートコールベクタテーブルレジスタ（TRCTB）



	b31																												b0
リセット後の値	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ルートコールベクタテーブルレジスタ（TRCTB）には、ワールド遷移受付された無条件トラップで使用するルートコールベクタテーブルの先頭番地を設定してください。初期値は 8000 0000h となります。このレジスタは、スーパバイザモードかつ Root ワールドで実行中のみアクセス可能です。

1.3.2.14　ワールド遷移割り込みベクタテーブルレジスタ（WINTB）



	b31																												b0
リセット後の値	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ワールド遷移割り込みベクタテーブルレジスタ（WINTB）には、ワールド遷移受付された割り込みで使用するワールド遷移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。初期値は 8000 0000h となります。このレジスタは、スーパバイザモードかつ Root ワールドで実行中のみアクセス可能です。

1.3.2.15　ワールド共通プロセッサ割り込み優先レベルレジスタ（WPRI）

	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	WIPL[4:0]					—	—	—	—	—	—	—	—
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
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
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
b23-b0	—	予約ビット	読むと“0”が読めます。書く場合、“0”としてください	R
b28-b24	WIPL[4:0]	ワールド共通プロセッサ割り込み優先レベル	b28　b24 0 0 0 0 0：優先レベル0（最低） 0 0 0 0 1：優先レベル1 0 0 0 1 0：優先レベル2 0 0 0 1 1：優先レベル3 0 0 1 0 0：優先レベル4 0 0 1 0 1：優先レベル5 0 0 1 1 0：優先レベル6 0 0 1 1 1：優先レベル7 0 1 0 0 0：優先レベル8 0 1 0 0 1：優先レベル9 0 1 0 1 0：優先レベル10 0 1 0 1 1：優先レベル11 0 1 1 0 0：優先レベル12 0 1 1 0 1：優先レベル13 0 1 1 1 0：優先レベル14 0 1 1 1 1：優先レベル15（Non-secure最高） 1 0 0 0 0：優先レベル16 1 0 0 0 1：優先レベル17 1 0 0 1 0：優先レベル18 1 0 0 1 1：優先レベル19 1 0 1 0 0：優先レベル20 1 0 1 0 1：優先レベル21 1 0 1 1 0：優先レベル22 1 0 1 1 1：優先レベル23 1 1 0 0 0：優先レベル24 1 1 0 0 1：優先レベル25 1 1 0 1 0：優先レベル26 1 1 0 1 1：優先レベル27 1 1 1 0 0：優先レベル28 1 1 1 0 1：優先レベル29 1 1 1 1 0：優先レベル30 1 1 1 1 1：優先レベル31（Secure/Root最高）	R/W
b31-b29	—	予約ビット	読むと“0”が読めます。書く場合、“0”としてください	R

このレジスタは、Secure、Root ワールドでのみ読み書きできます。
このレジスタは、スーパバイザモードでのみ書き込みできます。

WIPL[4:0] ビット（ワールド共通プロセッサ割り込み優先レベル）

WIPL[4:0] ビットは、優先レベル 0（最低）～優先レベル 31（最高）までの 32 段階のワールド共通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、現在のプロセッサ割り込み優先レベル（PWS.IPL[4:0]）より高く、かつワールド共通プロセッサ割り込み優先レベル（WPRI.WIPL[4:0]）よりも高い場合に、その割り込みが許可されます。

1.3.3　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 命令はガードビット（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）にデータを書き込みます。

読み出しには、MVFACGU 命令、MVFACHI 命令、MVFACMI 命令と MVFACLO 命令を使用します。MVFACGU 命令はガードビット（b95 ～ b64）、MVFACHI 命令は上位側 32 ビット（b63 ～ b32）、MVFACMI 命令は中央の 32 ビット（b47 ～ b16）、MVFACLO 命令は下位側 32 ビット（b31 ～ b0）からデータを読み出します。

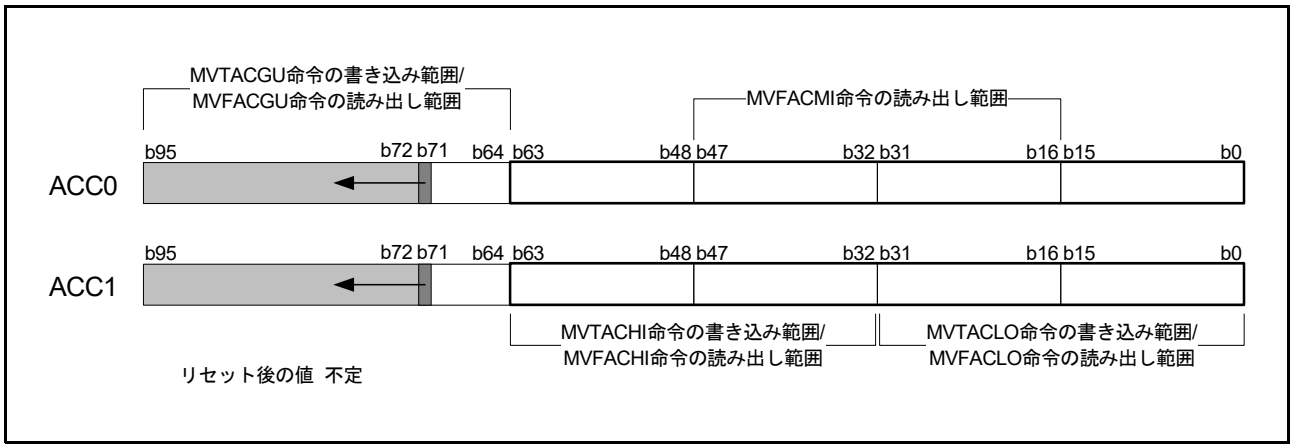


図 1.3　アキュムレータ

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

1.4　単精度浮動小数点例外

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

以下に、単精度浮動小数点例外を発生させる事象の概要を示します。

1.4.1　オーバフロー

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

表 1.1　オーバフロー発生時の演算結果

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

注．　EO = 0のとき、オーバフローが発生すると精度異常が発生します。

1.4.2　アンダフロー

アンダフローは、演算結果の絶対値が単精度浮動小数点フォーマットの正規化数で表現可能な値よりも小さくなった場合（ただし、0 を除く）に発生します。**表 1.2** にアンダフロー発生時の演算結果を示します。

表 1.2　アンダフロー発生時の演算結果

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

1.4.3　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。

1.4.4　ゼロ除算

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

表1.4　ゼロ除算発生時の演算結果

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

表1.5　ゼロ除算が発生しないときの動作

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

1.4.5　無効演算

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

表1.6　無効演算発生条件と演算結果

発生条件	演算結果（デスティネーションレジスタの内容）	
	EV = 0	EV = 1
SNaNオペランドに対する演算	QNaN	変化なし
+∞ + (–∞)、+∞ – (+∞)、–∞ – (–∞)		
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、無効演算が発生します。変数の初期値として使用することにより、プログラムのバグの発見に役立ちます。なお、SNaNをハードウェアが生成することはありません。
- ・QNaN（Quiet NaN）　：　仮数部の最上位ビットが“1”であるNaNです。QNaNを演算のソースオペランドとして使用しても、無効演算は発生しません（比較、フォーマット変換を除く）。演算によって伝播するため、例外処理を実行せずに結果だけを見てデバッグを行うことができます。なお、QNaNは演算によりハードウェアが生成します。

表 1.7 に演算結果が QNaN となる場合の生成規則を示します。

表1.7　QNaN生成規則

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

注.　SNaNのQNaN化は、仮数部の最上位ビットを“1”にして行います。

1.4.6　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。

1.5 プロセッサモード

RXv3e 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。

1.5.1 スーパバイザモード

スーパバイザモードでは、実行中のワールドでアクセス可能なすべての CPU リソースにアクセスすることができ、また、実行中のワールドで実行可能な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、「1.3.2.5 プロセッサステータスワード（PSW）」を参照してください。

1.5.2 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[4:0], 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）
- 割り込みベクタ制御レジスタ（IVC）
- ワールド ID レジスタ（WID）
- ルートコールベクタテーブルレジスタ（TRCTB）
- ワールド遷移割り込みベクタテーブルレジスタ（WINTB）
- ワールド共通プロセッサ割り込み優先レベルレジスタ（WPRI）

1.5.3 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、TRCALL、TRMVFC、TRMVTC、TRRTE、SAVE、RSTR 命令があります。

また、指定可能な範囲外の番号を指定してユーザモードで INT 命令を実行した場合、特権命令例外が発生します。詳細は、IVC.UIR ビットの説明を参照ください。

1.5.4 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、TRMVTC、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、退避させた 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せている PSW の PM ビットを “1” にした後 RTE 命令、または TRRTE 命令を実行する、あるいはバックアップ PSW（BPSW）に退避させている 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。

1.6　データタイプ

RXv3e CPUは、整数、単精度浮動小数点数、ビット、ストリングの4種類のデータを扱うことができます。

1.6.1　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。

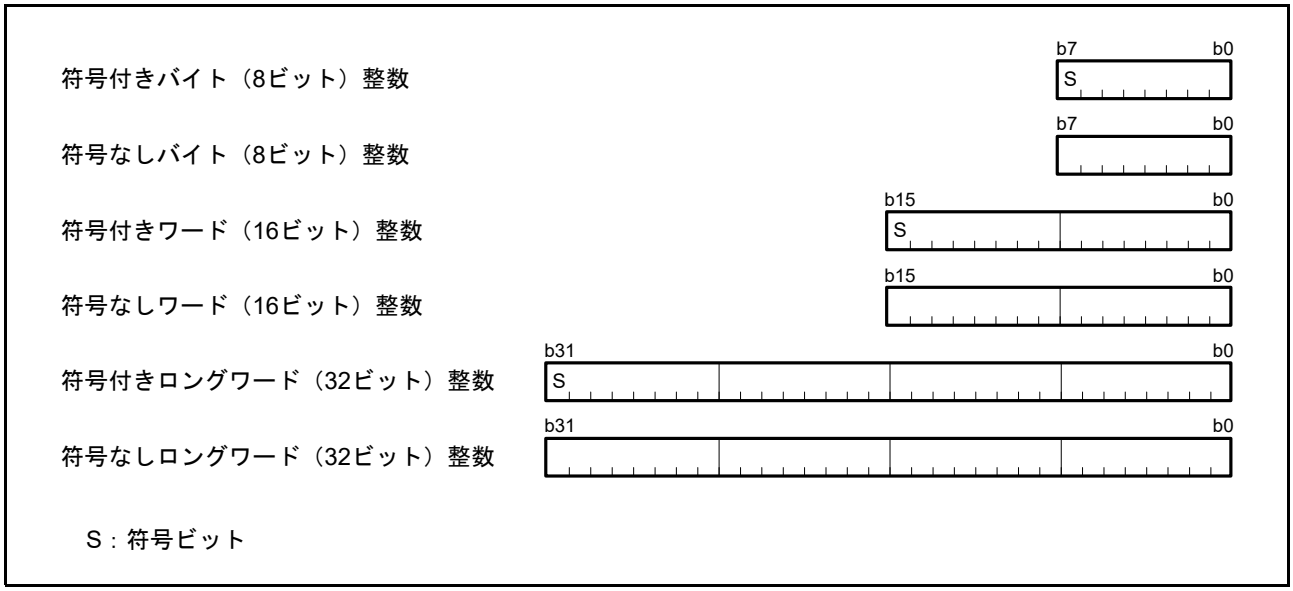


図 1.4　整数

1.6.2　単精度浮動小数点数

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

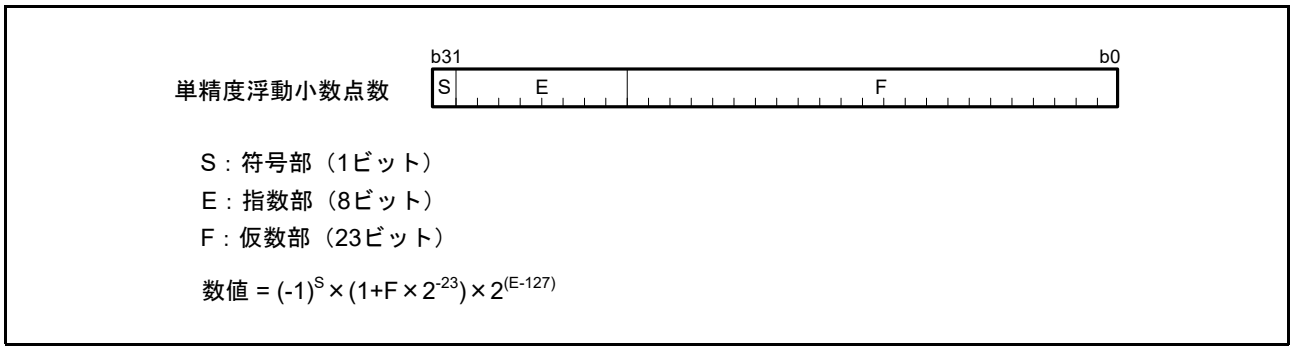


図 1.5　単精度浮動小数点数

単精度浮動小数点数は、以下の数値に対応しています。

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

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

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

1.6.3　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。

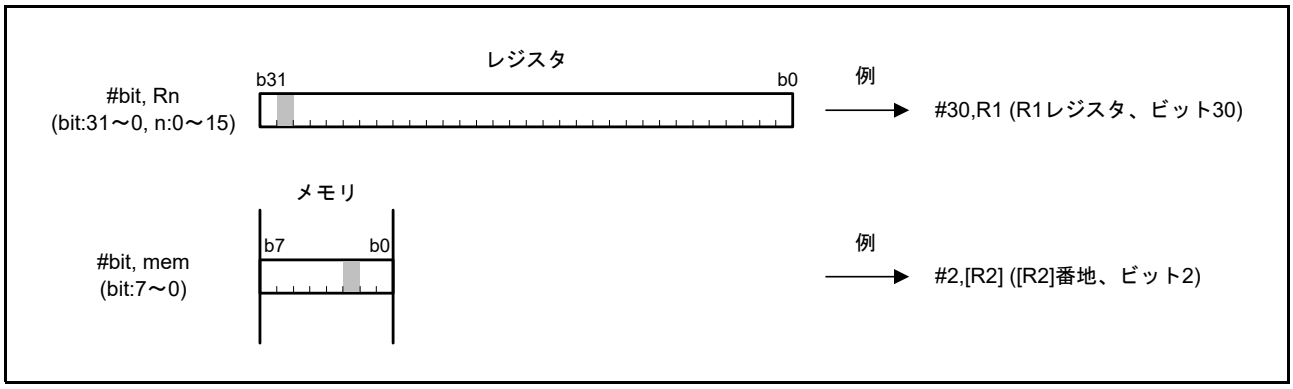


図 1.6　ビット

1.6.4　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。

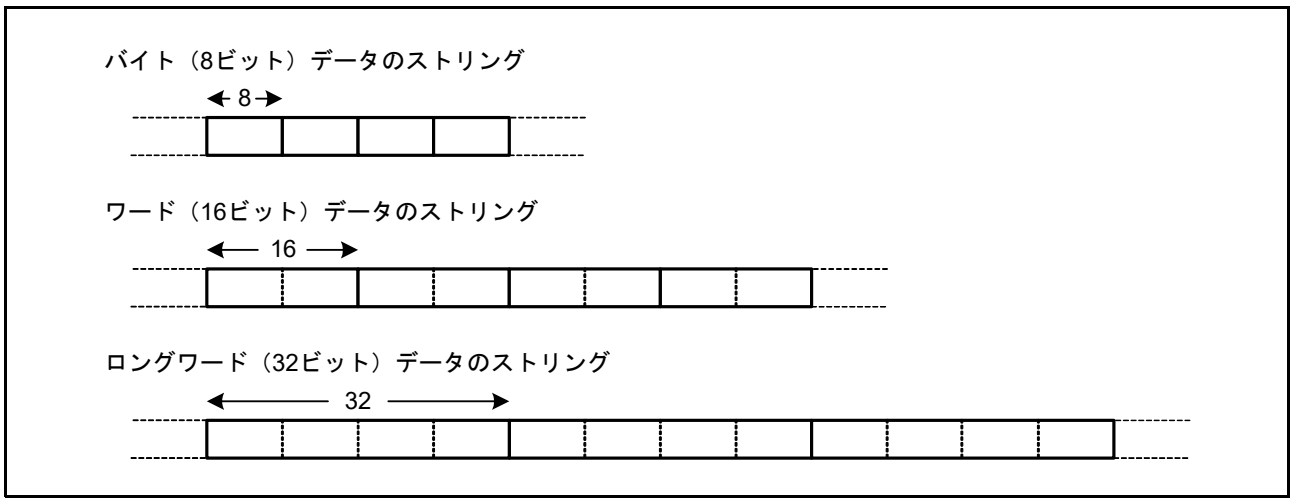


図 1.7　ストリング

1.7 データ配置

1.7.1 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 1.8 に示します。

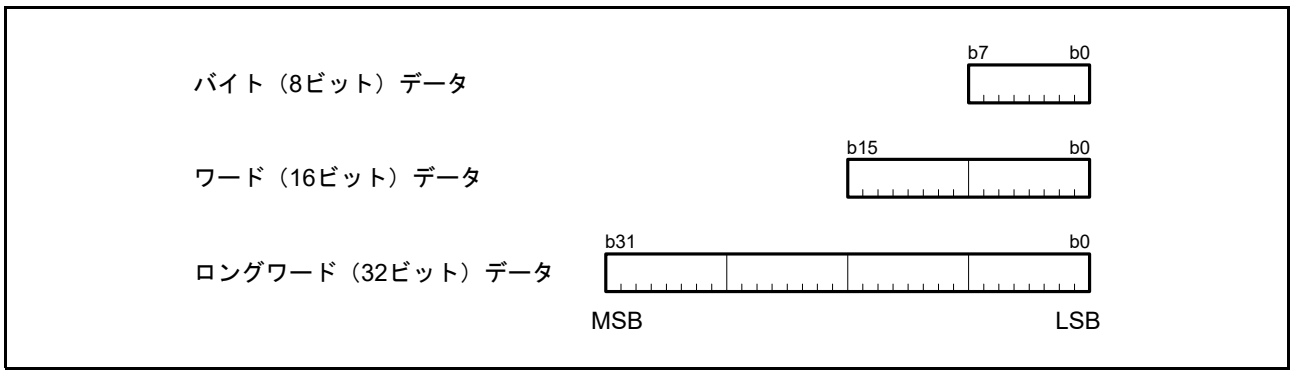


図 1.8　レジスタのデータ配置

1.7.2 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 1.9 に示します。

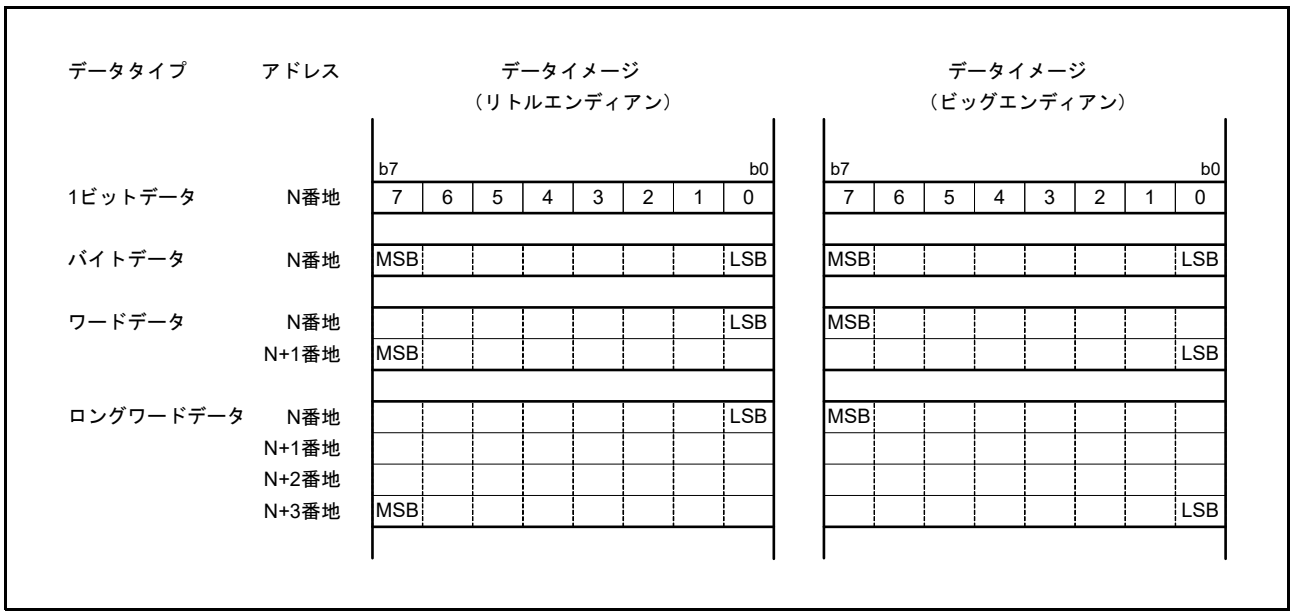


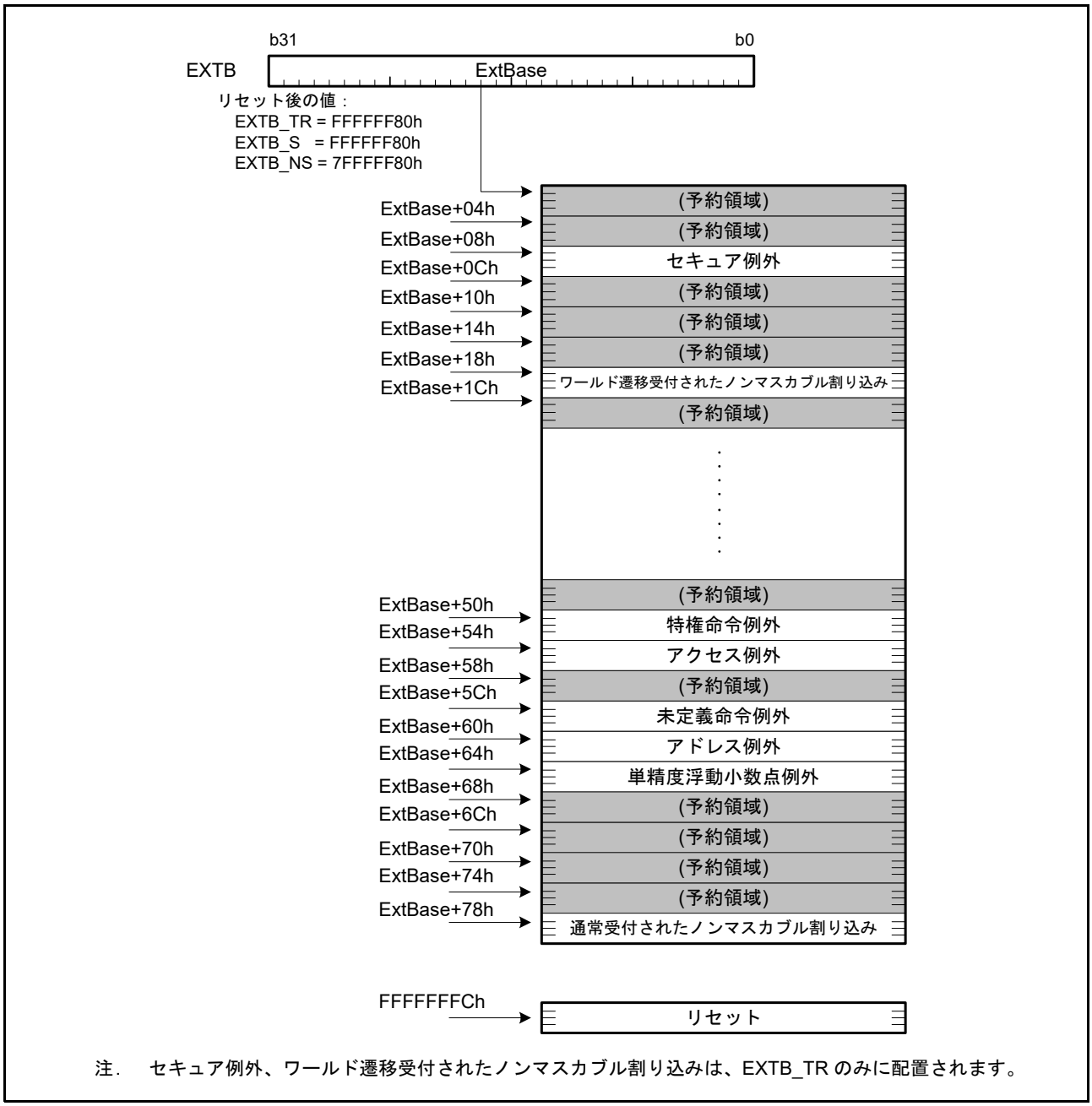
図 1.9　メモリ上のデータ配置

1.8　ベクタテーブル

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

1.8.1　例外ベクタテーブル

例外ベクタテーブルは、例外テーブルレジスタ（EXTB）の内容で示された値を先頭アドレス（ExtBase）とする 124 バイトの領域に、セキュア例外、ワールド遷移受付されたノンマスカブル割り込み、特権命令例外、アクセス例外、アドレス例外、未定義命令例外、単精度浮動小数点例外、通常受付されたノンマスカブル割り込みの各ベクタを配置しています。リセットのベクタは、例外ベクタテーブルの値に関係なく常に FFFFFFFCh 番地に配置されます。**図 1.10** に例外ベクタテーブルを示します。



注．　セキュア例外、ワールド遷移受付されたノンマスカブル割り込みは、EXTB_TR のみに配置されます。

図 1.10　例外ベクタテーブル

1.8.2　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 1.11** 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。

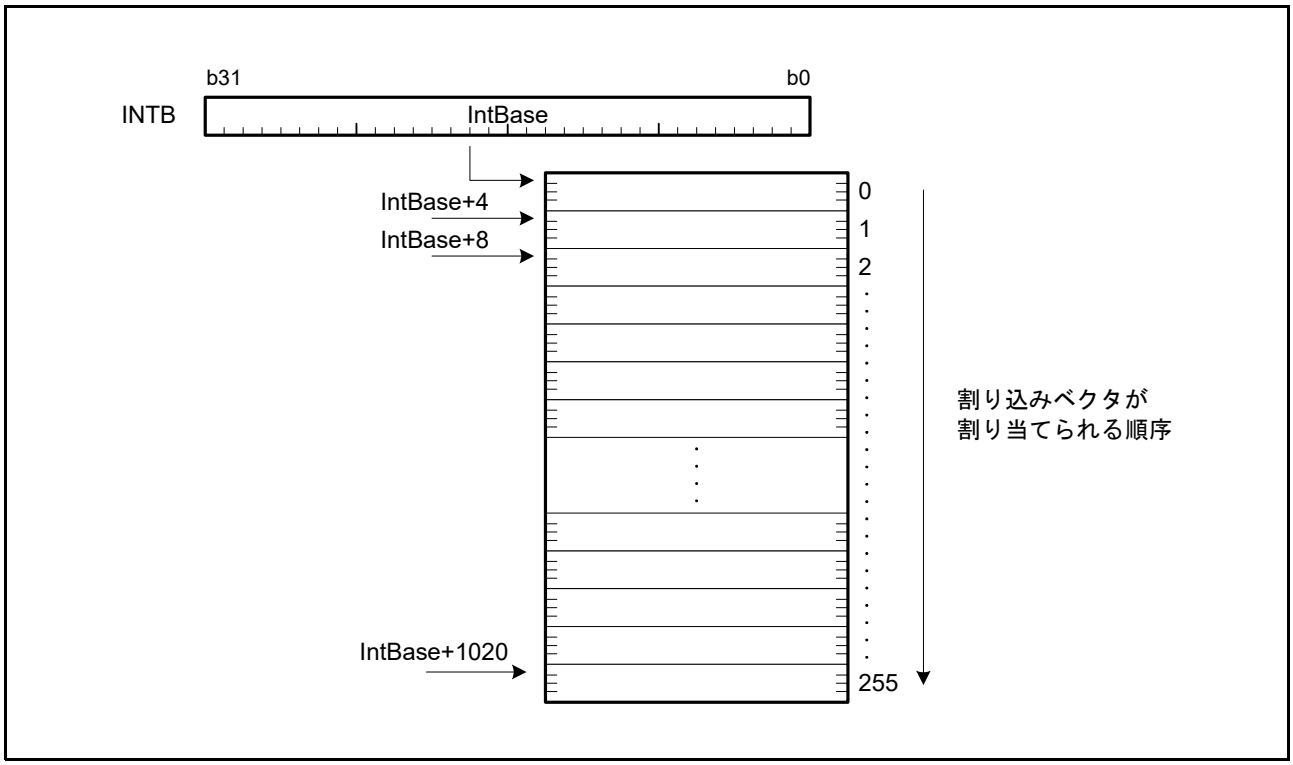


図 1.11　割り込みベクタテーブル

1.8.3　ルートコールベクタテーブル

ルートコールベクタテーブルは、ルートコールベクタテーブルレジスタ（TRCTB）の内容で示された値を先頭アドレス（TRCallBase）とする 1,024 バイトの領域に、TRCALL 命令でコールされる各ベクタを配置しています。**図 1.12** にルートコールベクタテーブルを示します。

ルートコール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。TRCALL 命令にてオペランドで指定した番号（0 ～ 255）に対応したベクタが割り当てられています。

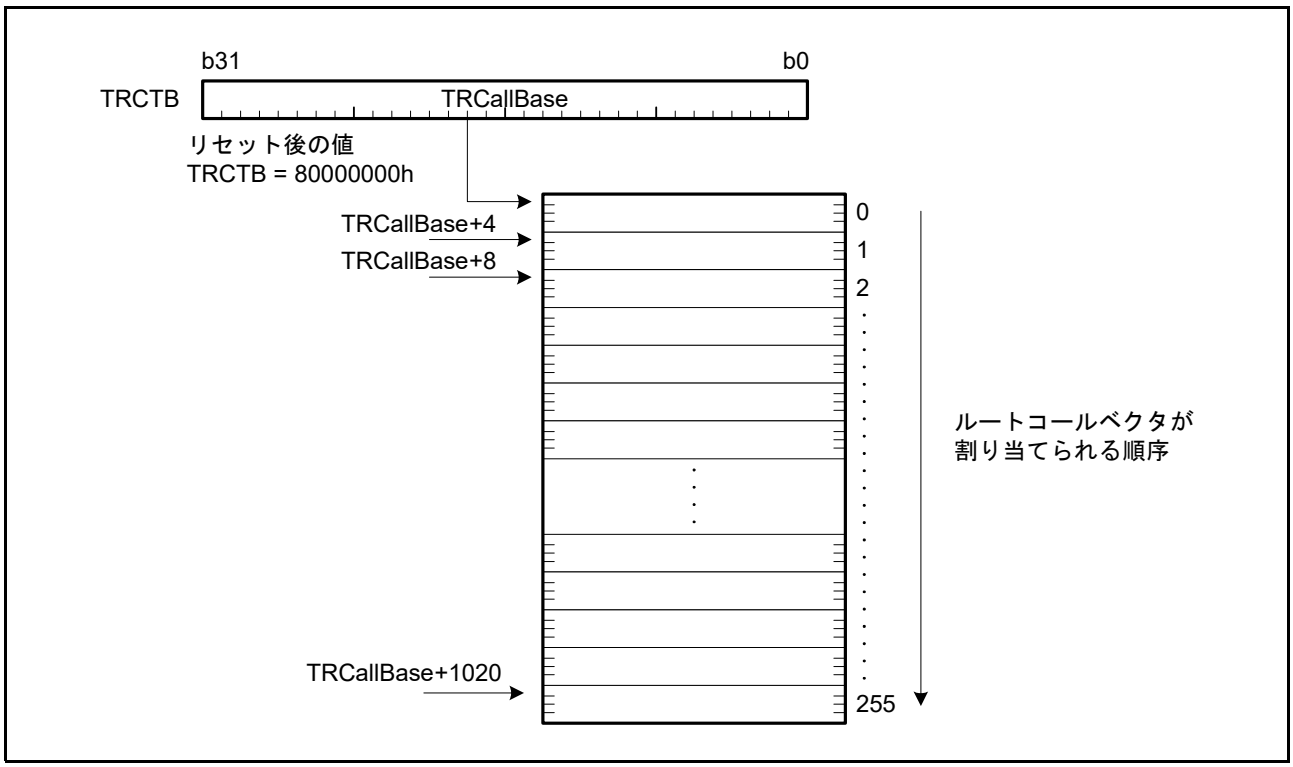


図 1.12　ルートコールベクタテーブル

1.8.4　ワールド遷移割り込みベクタテーブル

ワールド遷移割り込みベクタテーブルは、ワールド遷移割り込みベクタテーブルレジスタ（WINTB）の内容で示された値を先頭アドレス（WINTBase）とする 12 バイトの領域に、ワールド遷移受付された割り込みの各ベクタを配置しています。**図 1.13** にワールド遷移割り込みベクタテーブルを示します。

ワールド遷移割り込みベクタテーブルには、ベクタごとに番号（0 ～ 2）が付けられています。ベクタ番号は受け付けた割り込み（ワールド遷移受付）の宛先ワールド ID に対応します。

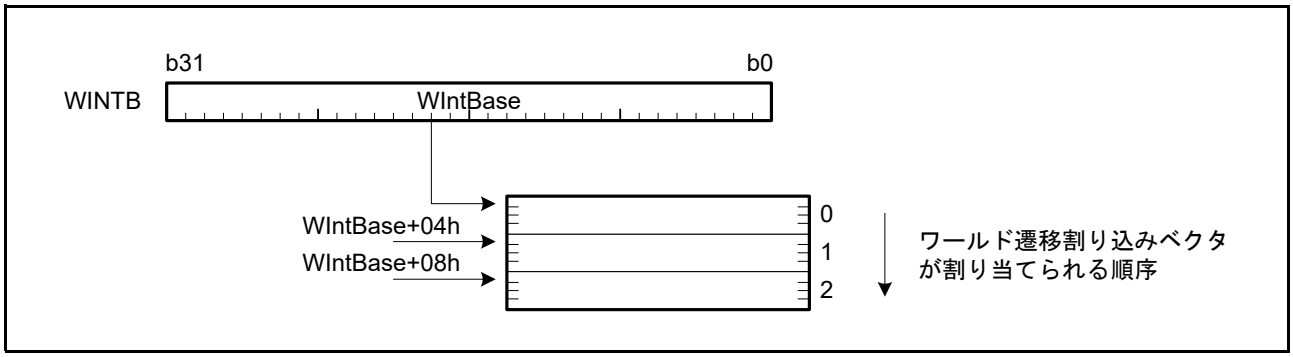


図 1.13　ワールド遷移割り込みベクタテーブル

1.9　アドレス空間

RXv3e 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。RXv3e CPUのアドレス空間を図1.14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。

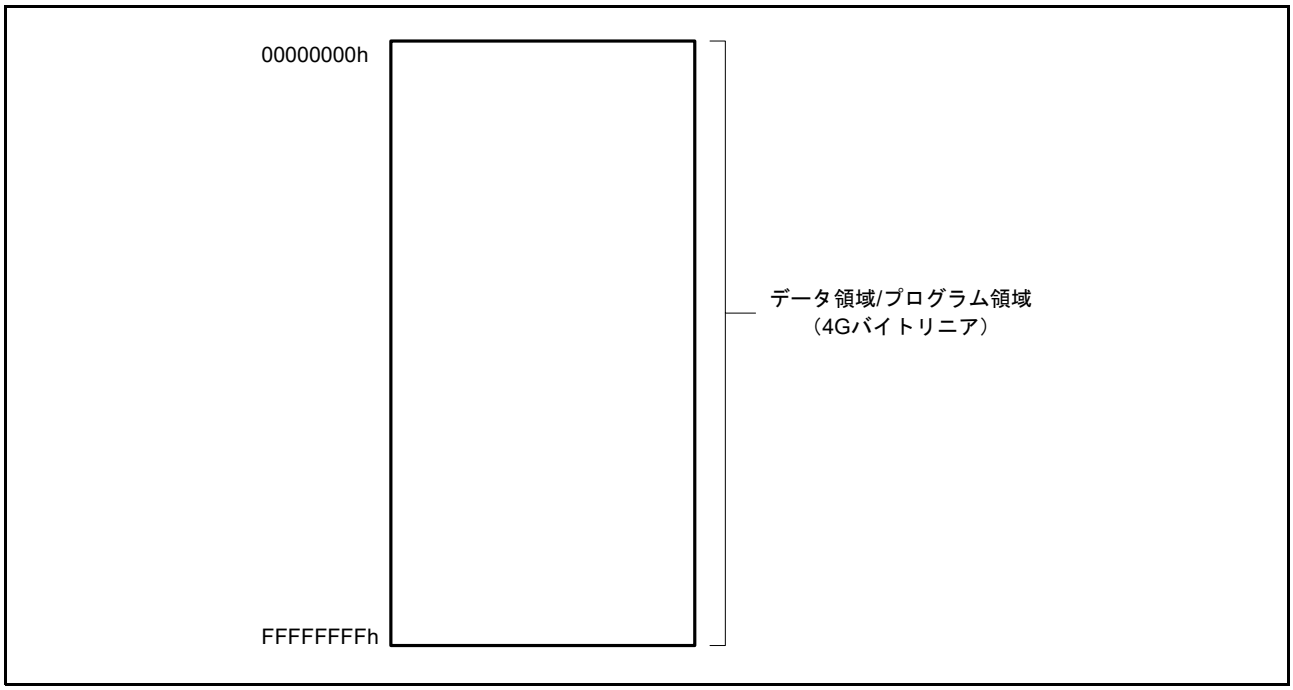


図1.14　アドレス空間

1.10　レジスター括退避機能

RXv3e CPUは、CPUレジスタの退避・復帰を一括して高速に行うために、専用のレジスタ退避バンクとそれを使用するための命令を有しています（図1.15参照）。レジスタ退避バンクを使うことで、例外処理ルーチン先頭でのレジスタ退避と、末尾でのレジスタ復帰を一括して高速に行うことが可能です。

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

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

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

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

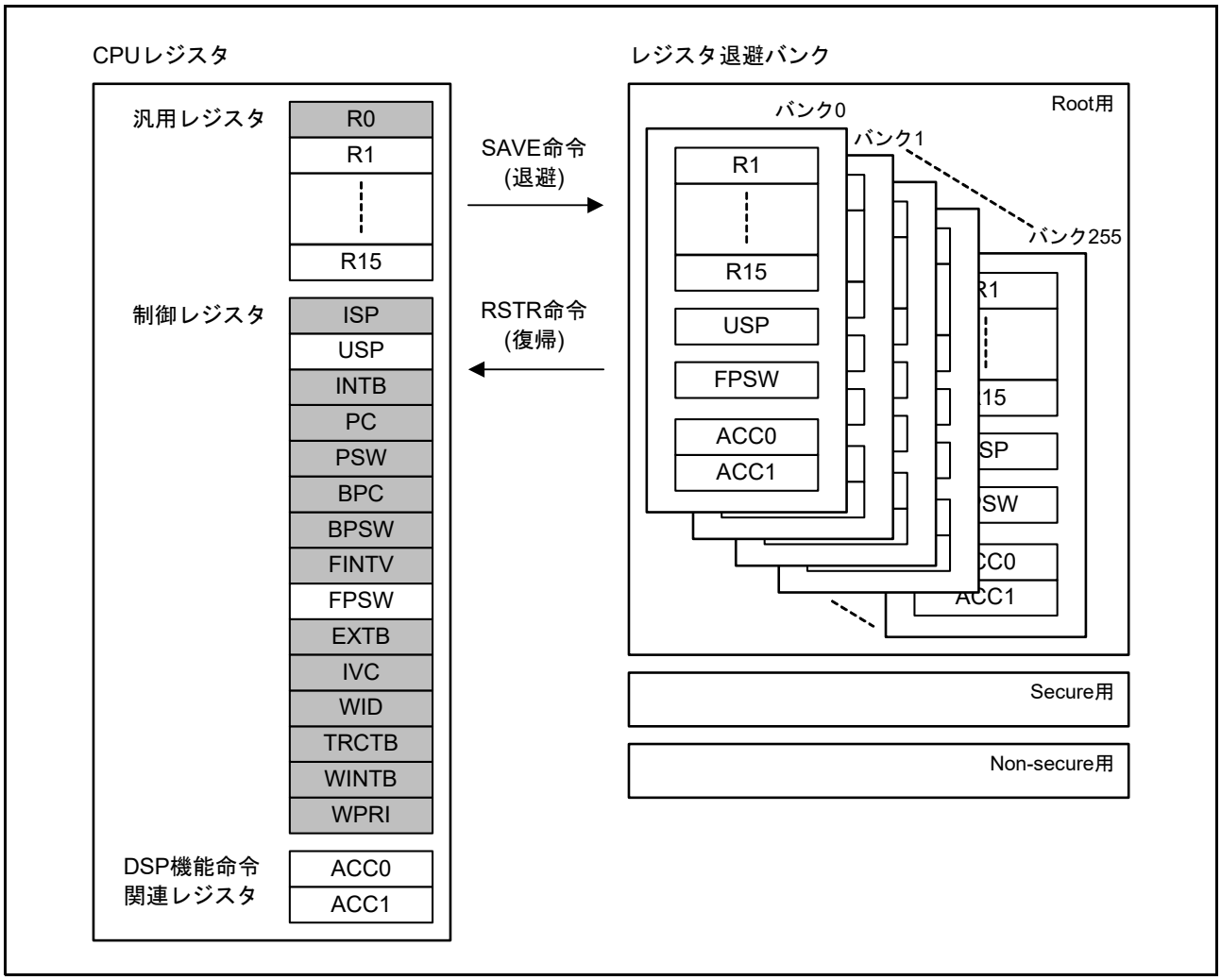


図1.15　レジスタ退避バンク

レジスタ退避バンクは、各ワールドにそれぞれ存在します。Root 用のレジスタ退避バンク TR、Secure 用のレジスタ退避バンク S、Non-secure 用のレジスタ退避バンク NS があります。SAVE 命令、RSTR 命令を実行したワールドに対応するレジスタ退避バンクが使用されます。

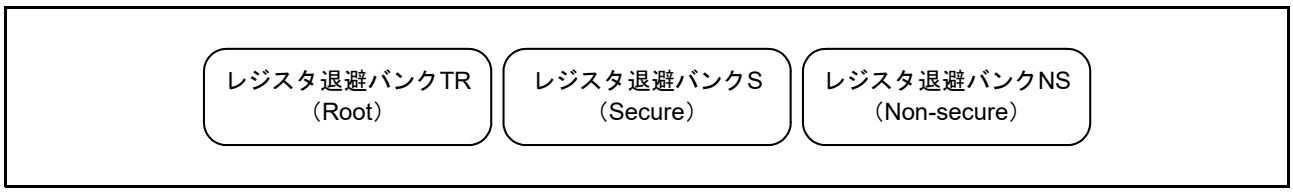


図 1.16　Protect Zone 対応レジスタ退避バンク構成

1.11　倍精度浮動小数点コプロセッサ

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

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

1.11.1　特長

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

1.11.2　倍精度浮動小数点レジスタセット

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

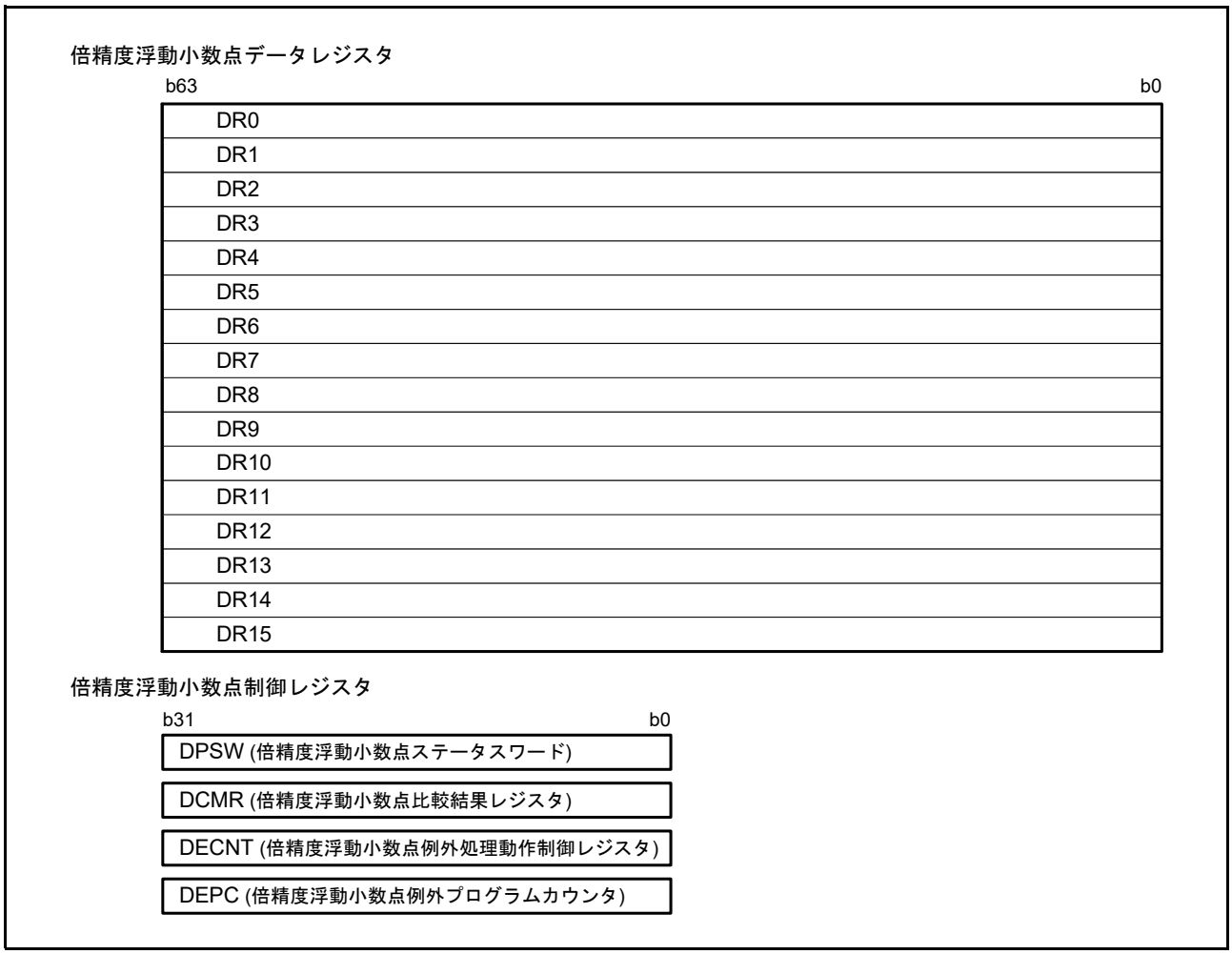


図 1.17　倍精度浮動小数点レジスタセット

1.11.2.1 倍精度浮動小数点データレジスタ（DR0 ～ DR15）

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

1.11.2.2 倍精度浮動小数点制御レジスタ

倍精度浮動小数点制御レジスタには、以下の 4 本のレジスタがあります。

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

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

DCR0：DPSW

DCR1：DCMR

DCR2：DECNT

DCR3：DEPC

(1) 倍精度浮動小数点ステータスワード（DPSW）

	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	DFS	DFX	DFU	DFZ	DFO	DFV	—	—	—	—	—	—	—	—	—	—
リセット後の値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	DEX	DEU	DEZ	DEO	DEV	—	DDN	DCE	DCX	DCU	DCZ	DCO	DCV	DRM[1:0]	
リセット後の値	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

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

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

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

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

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

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

DRM[1:0] ビット（倍精度浮動小数点丸めモード設定ビット）

倍精度浮動小数点丸めモードを設定します。

【倍精度浮動小数点丸めモードの説明】

- 最近値への丸め（デフォルト）　：　無限の有効桁を持つと仮定して計算した結果に近い方の値へ丸める
中間時は結果が偶数になる方向へ丸める
- 0方向への丸め　：　結果の絶対値が小さくなる方向へ丸める（単純な切り捨て）
- +∞方向への丸め　：　結果の値が大きくなる方向へ丸める
- –∞方向への丸め　：　結果の値が小さくなる方向へ丸める

（1）「最近値への丸め」はデフォルトのモードであり、最も正確な値を返します。
（2）「0方向への丸め」、「+∞方向への丸め」、「–∞方向への丸め」は、区間演算（Interval arithmetic）を使用した精度保証を行うときに使用します。

DCV フラグ（無効演算要因フラグ）、DCO フラグ（オーバフロー要因フラグ）、
DCZ フラグ（ゼロ除算要因フラグ）、DCU フラグ（アンダフロー要因フラグ）、
DCX フラグ（精度異常要因フラグ）、DCE フラグ（非実装処理要因フラグ）

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

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

DDN ビット（非正規化数の 0 フラッシュビット）

“0”のとき非正規化数を非正規化数として扱います。
“1”のとき非正規化数を0として扱います。

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

倍精度浮動小数点演算命令実行により、IEEE754 規格で規定された 5 つの例外が発生したときに、例外処理（DECNT.EHS ビットがセットされることによる割り込み情報の保持と割り込みコントローラへの割り込み要求）を行うかどうかを制御します。“0” の場合、DECNT.EHS ビットに “1” をセットしません。“1” の場合、DECNT.EHS ビットに “1” をセットします。ただし、DECNT.EHM ビットが “0” の場合は本ビットの値にかかわらず DECNT.EHS ビットに “1” をセットしません。

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

例外処理許可ビット DEj 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。

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

DFS フラグ（倍精度浮動小数点エラーサマリフラグ）

DFU、DFZ、DFO、DFV フラグの論理和を反映します。

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

(2)　倍精度浮動小数点比較結果レジスタ（DCMR）

	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
リセット後の値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	RES
リセット後の値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

(3)　倍精度浮動小数点例外処理動作制御レジスタ（DECNT）

	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	EHS
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
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	EHM
リセット後の値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

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

(4)　倍精度浮動小数点例外プログラムカウンタ（DEPC）

	b31	b0
リセット後の値	不定	

倍精度浮動小数点例外が発生し、対応するDPSWのDEX、DEU、DEZ、DEO、DEVビットが“1”のときに、倍精度浮動小数点例外が発生した命令のプログラムカウンタ値がこのレジスタに保持されます。このレジスタはリードオンリです。DECNTレジスタのEHMビットが“1”でかつEHSビットが“1”のとき、このレジスタの値は更新されません。

1.11.3　倍精度浮動小数点例外

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

以下に、倍精度浮動小数点例外を発生させる事象の概要を示します。

1.11.3.1　オーバフロー

オーバフローは、演算結果の絶対値が倍精度浮動小数点フォーマットで表現可能な値よりも大きくなった場合に発生します。DTOF 命令の場合は変換結果の絶対値が単精度浮動小数点フォーマットで表現可能な値よりも大きくなった場合に発生します。**表 1.8** にオーバフロー発生時の演算結果を示します。

表 1.8　オーバフロー発生時の倍精度浮動小数点演算結果

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

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

1.11.3.2　アンダフロー

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

表 1.9　アンダフロー発生時の倍精度浮動小数点演算結果

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

1.11.3.3　精度異常

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

表1.10　精度異常発生条件と倍精度浮動小数点演算結果

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

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

1.11.3.4　ゼロ除算

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

表1.11　ゼロ除算発生時の倍精度浮動小数点演算結果

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

表1.12　ゼロ除算が発生しないときの動作

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

1.11.3.5　無効演算

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

表1.13　無効演算発生条件と倍精度浮動小数点演算結果

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

表 1.14 に演算結果が QNaN となる場合の生成規則を示します。

表1.14　QNaN生成規則

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

注.　SNaNのQNaN化は、仮数部の最上位ビットを“1”にして行います。

1.11.3.6　非実装処理

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

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

1.11.4　データタイプ（倍精度浮動小数点コプロセッサ）

倍精度浮動小数点コプロセッサは、倍精度浮動小数点数を扱うことができます。

また、DTOF、FTOD 命令では単精度浮動小数点数、DROUND、DTOI、DTOU、ITOD、UTOD 命令では 32 ビット整数も扱えます。単精度浮動小数点数については、**「1.6.2　単精度浮動小数点数」**、32 ビット整数については、**「1.6.1　整数」**を参照してください。

1.11.4.1　倍精度浮動小数点数

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

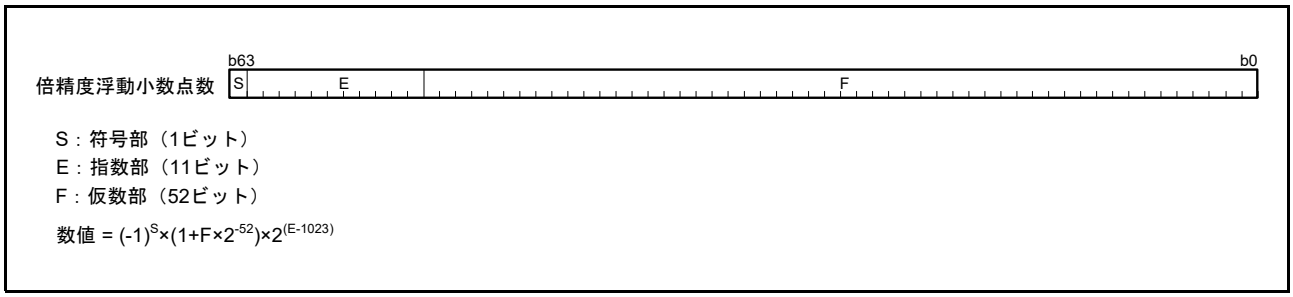


図 1.18　倍精度浮動小数点数

倍精度浮動小数点数は、以下の数値に対応しています。

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

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

1.11.5　データ配置（倍精度浮動小数点コプロセッサ）

倍精度浮動小数点コプロセッサは、32 ビットデータと 64 ビットデータを扱うことができます。
32 ビットデータについては、「1.7　**データ配置**」を参照してください。

1.11.5.1　倍精度浮動小数点レジスタのデータ配置

図 1.19 にレジスタのデータサイズと、ビット番号の関係を示します。

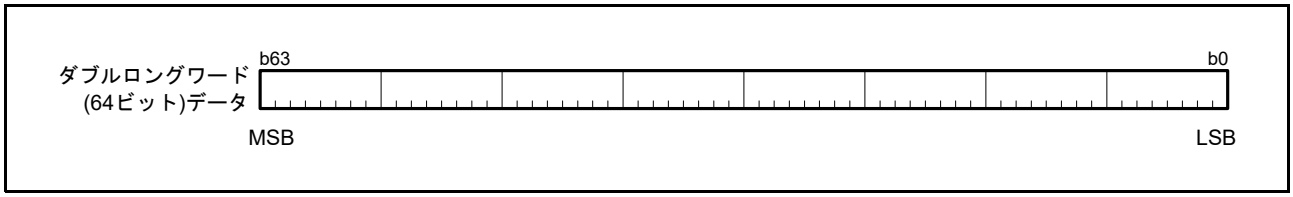


図 1.19　レジスタのデータ配置

1.11.5.2　メモリ上の倍精度浮動小数点データ配置

倍精度浮動小数点数のメモリ上のデータサイズは、ダブルロングワード（64 ビット）の 1 種類です。データ配置は、リトルエンディアンか、ビッグエンディアンかを選択することができます。図 1.20 にメモリ上のデータ配置を示します。

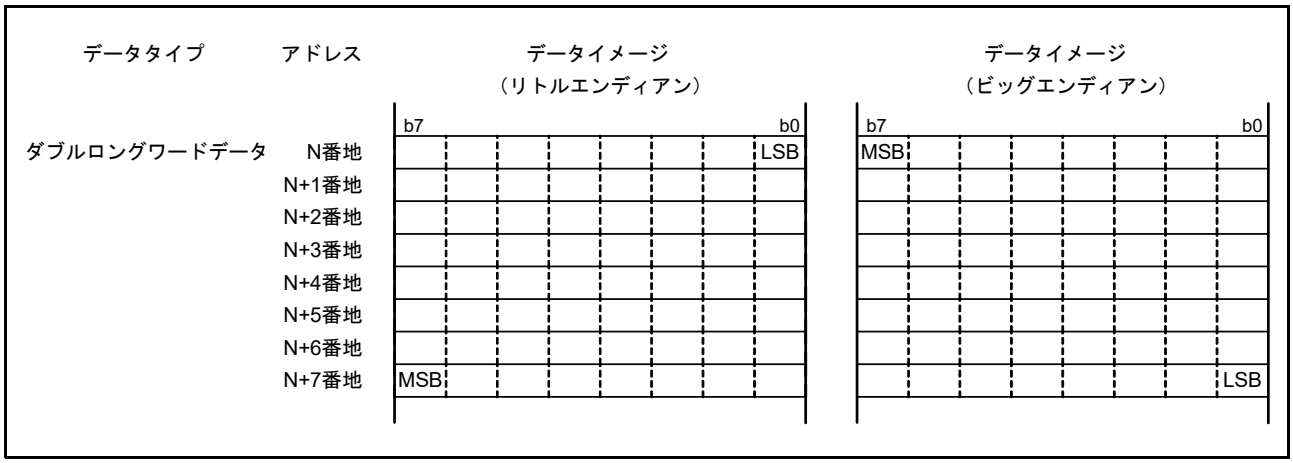


図 1.20　メモリ上のデータ配置

2. アドレッシングモード

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

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

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

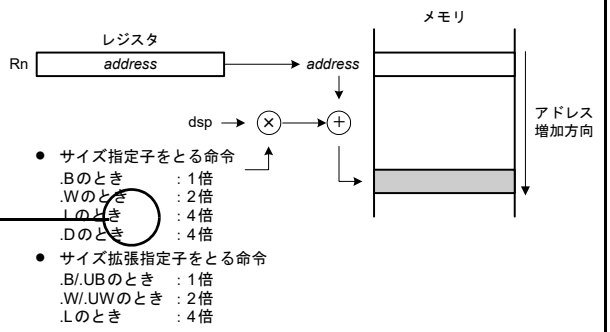
- 倍精度浮動小数点データレジスタ直接
- 倍精度浮動小数点制御レジスタ直接

2.1　本章の見方

本章の見方を以下に実例をあげて示します。

(1)

レジスタ相対	
(2) dsp:5[Rn] (Rn = R0～R7)	(3) 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。
(4) dsp:8[Rn] (Rn = R0～R15) 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。

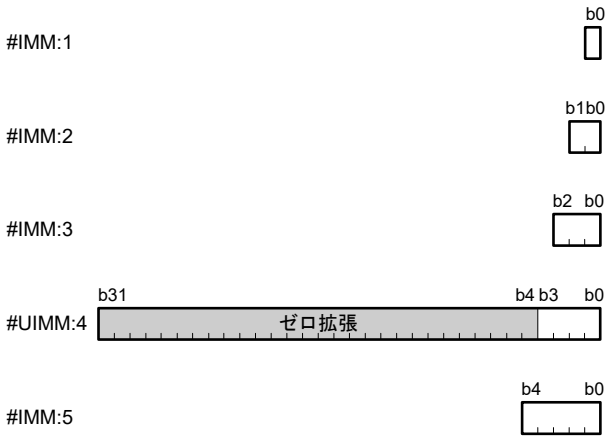
(3)　解説

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

即値		
#IMM:1 #IMM:2 #IMM:3 #UIMM:4 #IMM:5	• #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、RACL、RACW、RDACL、RDACW命令のソースで使用されます。 • #IMM:2 #IMMで示した2ビット即値が演算の対象となります。このアドレッシングモードは、MVFACGU、MVFACHI、MVFACLO、MVFACMI命令のソースで使用されます。 • #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、BM*Cnd*、BNOT、BSET、BTST）のビット番号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、BM*Cnd*、BNOT、BSET、BTST）のビット番号指定、転送命令（BFMOV、BFMOVZ）のビット番号指定とビット幅指定、算術/論理演算命令（SHAR、SHLL、SHLR）のシフト幅指定、および算術/論理演算命令（ROTL、ROTR）のローテート幅指定、MVTIPL命令の割り込み優先レベル指定で使用されます。	

<table>
<tr><td colspan="2">即値</td><td rowspan="2"><u>処理サイズがBのとき</u>
#IMM:8　b7 b0

<u>処理サイズがWのとき</u>
#SIMM:8　b15 b8b7 b0 符号拡張
#UIMM:8　b15 b8b7 b0 ゼロ拡張
#IMM:16　b15 b0

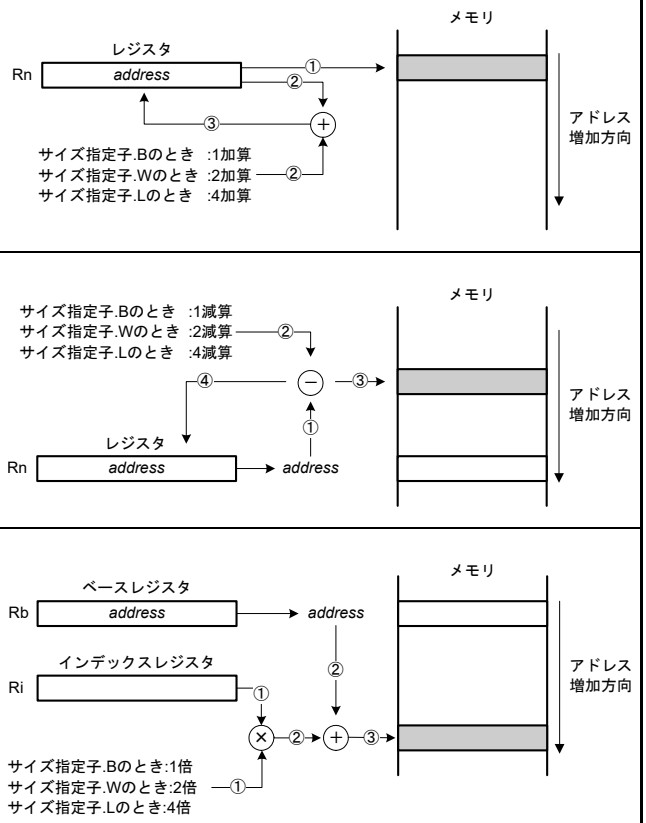
<u>処理サイズがLのとき</u>
#UIMM:8　b31 b8b7 b0 ゼロ拡張
#SIMM:8　b31 b8b7 b0 符号拡張
#SIMM:16　b31 b16b15 b0 符号拡張
#SIMM:24　b31 b24b23 b0 符号拡張
#IMM:32　b31 b0</td></tr>
<tr><td>#IMM:8
#SIMM:8
#UIMM:8
#IMM:16
#SIMM:16
#SIMM:24
#IMM:32</td><td>即値で指定した値が演算の対象となります。ただし、#UIMMで指定した即値は処理サイズにゼロ拡張した結果が、#SIMMで指定した即値は処理サイズに符号拡張した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></tr>
<tr><td colspan="2">レジスタ直接</td><td rowspan="2">b31 レジスタ b0
Rn

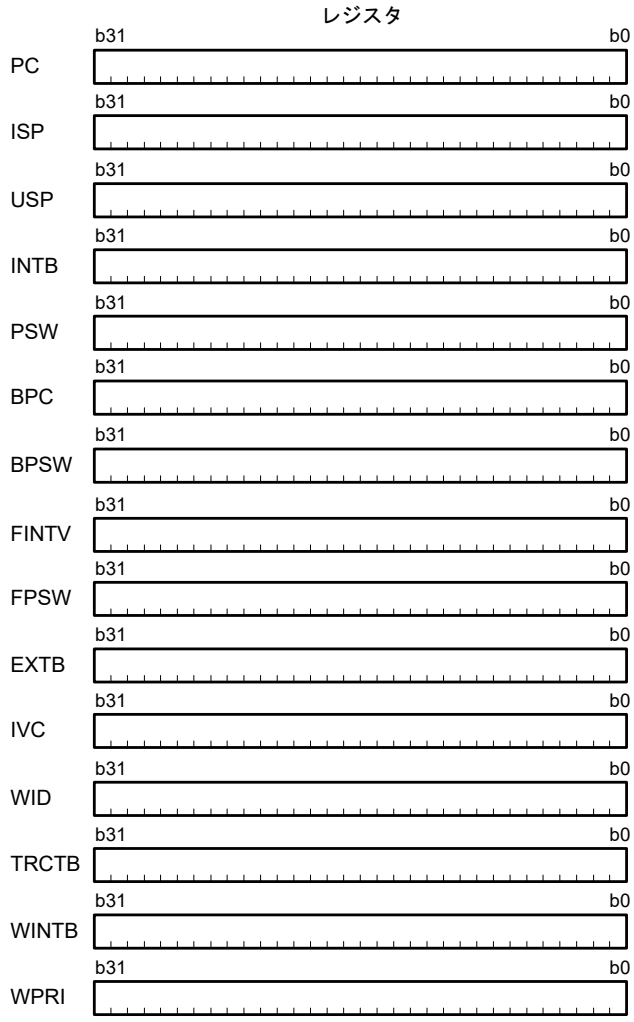
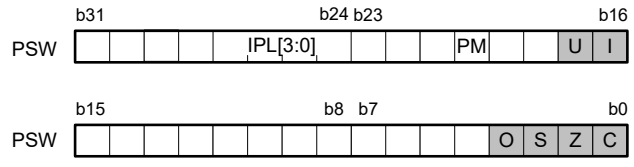
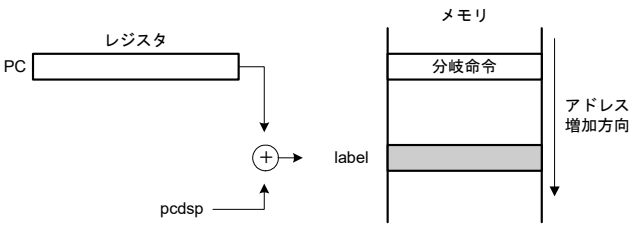
メモリ
レジスタ
Rn
レジスタ
PC
アドレス増加方向</td></tr>
<tr><td>Rn
(Rn =
R0～R15)</td><td>指定したレジスタが演算の対象となります。
またはJMP、JSR命令の場合、Rnの値をプログラムカウンタ（PC）に転送します。
実効アドレスの範囲は、00000000h～FFFFFFFFhです。
Rn (Rn = R0～R15)が指定できます。</td></tr>
<tr><td colspan="2">レジスタ間接</td><td rowspan="2">メモリ
レジスタ
Rn　address
アドレス増加方向</td></tr>
<tr><td>[Rn]
(Rn =
R0～R15)</td><td>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。</td></tr>
<tr><td colspan="2">レジスタ相対</td><td rowspan="2">メモリ
レジスタ
Rn　address　address
dsp　×　+
アドレス増加方向
● サイズ指定子をとる命令
.Bのとき　：1倍
.Wのとき　：2倍
.Lのとき　：4倍
.Dのとき　：4倍
● サイズ拡張指定子をとる命令
.B/.UBのとき　：1倍
.W/.UWのとき　：2倍
.Lのとき　：4倍</td></tr>
<tr><td>dsp:5[Rn]
(Rn =
R0～R7)

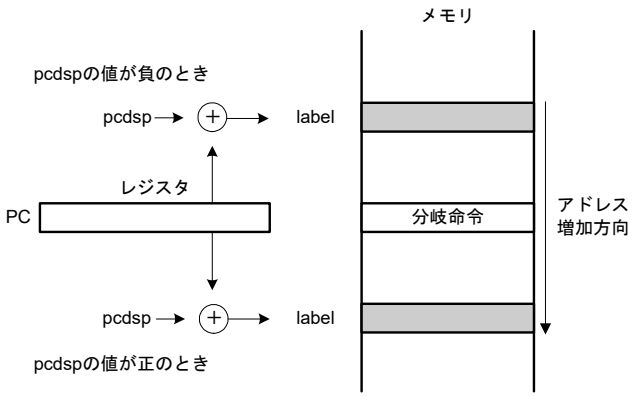
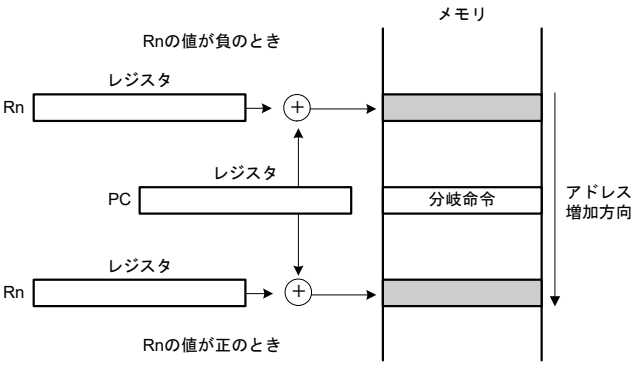
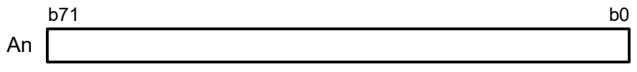
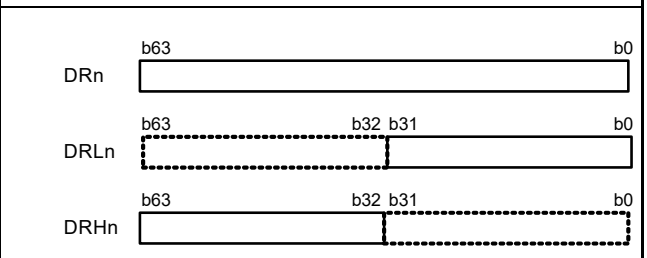
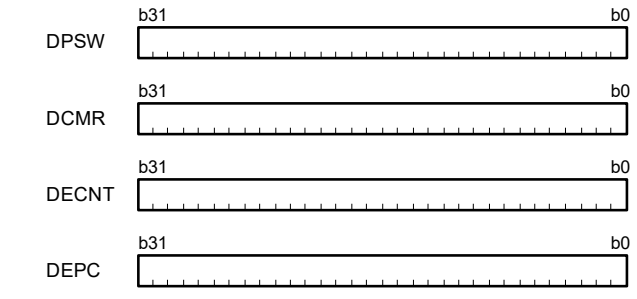
dsp:8[Rn]
(Rn =
R0～R15)

dsp:16[Rn]
(Rn =
R0～R15)</td><td>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。</td></tr>
</table>

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。
インデックス付きレジスタ間接	
[Ri,Rb] (Ri, Rb = R0～R15)	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。



<table>
<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
IVC
WID
TRCTB
WINTB
WPRI</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、TRMVFC、TRMVTC命令で使用されます。
命令により指定可能な制御レジスタは異なります。各命令の説明を参照してください。</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(BEQ、BZ、BNE、BNZのみ)、BRA命令で使用されます。</td></tr>
</table>

<table>
<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(BEQ、BZ、BNE、BNZのみ)、BRA、BSR命令で、“.A”のときBRA、BSR命令で使用されます。</td><td> 
</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>
<tr><td colspan="2">倍精度浮動小数点データレジスタ直接</td><td rowspan="2"> 
</td></tr>
<tr><td>DRn
(DRn = DR0～DR15)
DRLn
(DRLn = DRL0～DRL15)
DRHn
(DRHn = DRH0～DRH15)</td><td>指定した倍精度浮動小数点データレジスタが演算の対象となります。DRLnでは下位32ビットが、DRHnでは上位32ビットが対象となります。</td></tr>
<tr><td colspan="2">倍精度浮動小数点制御レジスタ直接</td><td rowspan="2"> 
</td></tr>
<tr><td>DPSW
DCMR
DECNT
DEPC</td><td>指定した倍精度浮動小数点制御レジスタが演算の対象となります。</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
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” とも表記できます。
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

RXv3eアーキテクチャの命令数は117です。1～8バイト可変長命令形式を採用しています。

RXv3e命令セットアーキテクチャは、RXv3、RXv2とRXv1に対して上位互換性のある命令セットアーキテクチャです。

- RXv2はRXv1に対し、主にDSP機能命令と単精度浮動小数点演算命令が拡充され、DSPに匹敵するデータ処理能力を実現しています。
- RXv3はRXv2に対し、主にレジスタ一括退避機能の搭載により割り込み応答性能の向上を実現しています。
- RXv3eはRXv3に対し、Protect Zoneを実装しセキュリティ機能を拡張しています。
- 倍精度浮動小数点コプロセッサを搭載している製品では、さらに21 種類の命令（倍精度浮動小数点処理命令）が追加になります。

RXv3e命令セットについて以下に示します。

命令一覧表 (1 / 5)

命令の種類	ニーモニック	機能	命令詳細記載ページ	命令コード詳細記載ページ	備考
標準搭載命令					
算術/論理演算命令	ABS	絶対値	84	274	
	ADC	キャリー付き加算	85	275	
	ADD	キャリーなし加算	86	276	
	AND	論理積	88	278	
	CMP	比較	102	292	
	DIV	符号付き除算	103	293	
	DIVU	符号なし除算	104	295	
	EMUL	符号付き乗算	107	297	
	EMULU	符号なし乗算	110	298	
	MAX	最大値選択	139	311	
	MIN	最小値選択	140	312	
	MUL	乗算	151	321	
	NEG	符号反転	166	331	
	NOP	ノーオペレーション	167	331	
	NOT	論理反転	168	332	
	OR	論理和	169	333	
	RMPA	積和演算	186	341	
	ROLC	キャリー付き左回転	188	341	
	RORC	キャリー付き右回転	189	341	
	ROTL	左回転	190	342	
	ROTR	右回転	191	342	
	SAT	32ビット符号付き飽和処理	200	344	
	SATR	RMPA命令用64ビット符号付き飽和処理	201	345	
	SBB	ボロー付き減算	202	345	
	SHAR	算術右シフト	206	348	
	SHLL	論理/算術左シフト	207	349	
	SHLR	論理右シフト	208	350	
	SUB	ボローなし減算	215	354	
	TST	テスト	220	356	
	XOR	排他的論理和	225	359	RXv3で拡張

命令一覧表 (2 / 5)

命令の種類	ニーモニック		機能	命令詳細記載ページ	命令コード詳細記載ページ	備考
転送命令	BFMOV		ビットフィールド転送	91	283	RXv3以降でサポート
	BFMOVZ		ゼロへのビットフィールド転送	92	283	RXv3以降でサポート
	MOV		転送	141	313	
	MOVCO		LIフラグクリア付きストア	145	318	RXv2以降でサポート
	MOVLI		LIフラグセット付きロード	146	318	RXv2以降でサポート
	MOVU		符号なしデータ転送	147	319	
	POP		スタックからレジスタへのデータ復帰	170	334	
	POPC		制御レジスタの復帰	171	335	RXv3eで拡張
	POPM		複数レジスタの復帰	172	335	
	PUSH		スタックへデータ退避	173	336	
	PUSHC		制御レジスタの退避	174	337	RXv3eで拡張
	PUSHM		複数レジスタの退避	175	337	
	REVL		エンディアン変換	184	340	
	REVW		エンディアン変換	185	340	
	SC*Cnd*	SCGEU	条件設定	203	346	
		SCC		203	346	
		SCEQ		203	346	
		SCZ		203	346	
		SCGTU		203	346	
		SCPZ		203	346	
		SCGE		203	346	
		SCGT		203	346	
		SCO		203	346	
		SCLTU		203	346	
		SCNC		203	346	
		SCNE		203	346	
		SCNZ		203	346	
		SCLEU		203	346	
		SCN		203	346	
		SCLE		203	346	
		SCLT		203	346	
		SCNO		203	346	
	STNZ		条件付き転送	213	352	RXv2で拡張
	STZ		条件付き転送	214	353	RXv2で拡張
	XCHG		交換	224	358	

命令一覧表 (3 / 5)

命令の種類	ニーモニック		機能	命令詳細記載ページ	命令コード詳細記載ページ	備考
分岐命令	BC*nd*	BGEU	相対条件分岐	90	282	
		BC		90	282	
		BEQ		90	282	
		BZ		90	282	
		BGTU		90	282	
		BPZ		90	282	
		BGE		90	282	
		BGT		90	282	
		BO		90	282	
		BLTU		90	282	
		BNC		90	282	
		BNE		90	282	
		BNZ		90	282	
		BLEU		90	282	
		BN		90	282	
		BLE		90	282	
		BLT		90	282	
		BNO		90	282	
	BRA		相対無条件分岐	96	286	
	BSR		相対サブルーチン分岐	99	289	
	JMP		無条件分岐	134	309	
	JSR		サブルーチン分岐	135	309	
	RTS		サブルーチンからの復帰	197	344	
	RTSD		スタックフレームの解放とサブルーチンからの復帰	198	344	
ビット操作命令	BCLR		ビットクリア	89	280	
	BM*Cnd*	BMGEU	条件ビット転送	93	284	
		BMC		93	284	
		BMEQ		93	284	
		BMZ		93	284	
		BMGTU		93	284	
		BMPZ		93	284	
		BMGE		93	284	
		BMGT		93	284	
		BMO		93	284	
		BMLTU		93	284	
		BMNC		93	284	
		BMNE		93	284	
		BMNZ		93	284	
		BMLEU		93	284	
		BMN		93	284	
		BMLE		93	284	
		BMLT		93	284	
		BMNO		93	284	

命令一覧表 (4 / 5)

命令の種類	ニーモニック	機能	命令詳細記載ページ	命令コード詳細記載ページ	備考
ビット操作命令	BNOT	ビット反転	95	285	
	BSET	ビットセット	98	287	
	BTST	ビットテスト	100	290	
ストリング操作命令	SCMPU	ストリング比較	204	346	
	SMOVB	逆方向ストリング転送	209	350	
	SMOVF	順方向ストリング転送	210	351	
	SMOVU	ストリング転送	211	351	
	SSTR	ストリングストア	212	351	
	SUNTIL	ストリングサーチ	216	355	
	SWHILE	ストリングサーチ	218	355	
システム操作命令	BRK	無条件トラップ	97	287	
	CLRPSW	PSWのフラグ、ビットのクリア	101	291	
	INT	ソフトウェア割り込み	131	307	RXv3eで拡張
	MVFC	制御レジスタからの転送	160	327	RXv3eで拡張
	MVTC	制御レジスタへの転送	164	329	RXv3eで拡張
	MVTIPL（特権命令）	割り込み優先レベル設定	165	330	RXv3eで拡張
	RTE（特権命令）	例外からの復帰	195	343	RXv3eで拡張
	RTFI（特権命令）	高速割り込みからの復帰	196	343	RXv3eで拡張
	SETPSW	PSWのフラグ、ビットのセット	205	347	
	WAIT（特権命令）	ウェイト	223	358	
単精度浮動小数点演算命令	FADD	単精度浮動小数点加算	112	300	RXv2で拡張
	FCMP	単精度浮動小数点比較	115	301	
	FDIV	単精度浮動小数点除算	117	302	
	FMUL	単精度浮動小数点乗算	119	303	RXv2で拡張
	FSUB	単精度浮動小数点減算	124	305	RXv2で拡張
	FSQRT	単精度浮動小数点平方根	122	304	RXv2以降でサポート
	FTOI	単精度浮動小数点数→符号付き整数変換	127	306	
	FTOU	単精度浮動小数点数→符号なし整数変換	129	306	RXv2以降でサポート
	ITOF	符号付き整数→単精度浮動小数点数変換	132	308	
	ROUND	単精度浮動小数点数→符号付き整数変換	192	343	
	UTOF	符号なし整数→単精度浮動小数点数変換	221	357	RXv2以降でサポート
DSP機能命令	EMACA	32ビット積和演算	105	296	RXv2以降でサポート
	EMSBA	32ビット積差演算	106	296	RXv2以降でサポート
	EMULA	32ビット乗算	109	298	RXv2以降でサポート
	MACHI	上位16ビット積和演算	136	310	RXv2で拡張
	MACLH	下位16ビット・上位16ビット積和演算	137	310	RXv2以降でサポート
	MACLO	下位16ビット積和演算	138	310	RXv2で拡張
	MSBHI	上位16ビット積差演算	148	320	RXv2以降でサポート
	MSBLH	下位16ビット・上位16ビット積差演算	149	320	RXv2以降でサポート
	MSBLO	下位16ビット積差演算	150	321	RXv2以降でサポート
	MULHI	上位16ビット乗算	153	323	RXv2で拡張
	MULLH	下位16ビット・上位16ビット乗算	154	323	RXv2以降でサポート
	MULLO	下位16ビット乗算	155	324	RXv2で拡張
	MVFACGU	アキュムレータガードビットからの転送	156	324	RXv2以降でサポート
	MVFACHI	アキュムレータ上位32ビットからの転送	157	325	RXv2で拡張
	MVFACLO	アキュムレータ下位32ビットからの転送	158	325	RXv2以降でサポート

命令一覧表 (5 / 5)

命令の種類	ニーモニック	機能	命令詳細記載ページ	命令コード詳細記載ページ	備考
DSP機能命令	MVFACMI	アキュムレータ中央32ビットからの転送	159	326	RXv2で拡張
	MVTACGU	アキュムレータガードビットへの転送	161	328	RXv2以降でサポート
	MVTACHI	アキュムレータ上位32ビットへの転送	162	328	RXv2で拡張
	MVTACLO	アキュムレータ下位32ビットへの転送	163	328	RXv2で拡張
	RACL	符号付きアキュムレータ丸め処理	176	338	RXv2以降でサポート
	RACW	16ビット符号付きアキュムレータ丸め処理	178	338	RXv2で拡張
	RDACL	符号付きアキュムレータ丸め処理	180	339	RXv2以降でサポート
	RDACW	16ビット符号付きアキュムレータ丸め処理	182	339	RXv2以降でサポート
レジスター括退避機能命令（オプション）					
レジスター括退避機能命令	RSTR（特権命令）	レジスタの一括復帰	227	362	RXv3以降でサポート
	SAVE（特権命令）	レジスタの一括退避	228	362	
倍精度浮動小数点処理命令（オプション）					
倍精度浮動小数点転送命令	DMOV	倍精度浮動小数点データ転送	237	366	倍精度浮動小数点コプロセッサ搭載製品でサポート
	DPOPM	複数の倍精度浮動小数点レジスタの復帰	242	370	
	DPUSHM	複数の倍精度浮動小数点レジスタの退避	244	371	
	MVFDC	倍精度浮動小数点制御レジスタからの転送	261	375	
	MVFDR	倍精度浮動小数点比較結果レジスタからの転送	262	375	
	MVTDC	倍精度浮動小数点制御レジスタへの転送	263	376	
倍精度浮動小数点演算命令	DABS	倍精度浮動小数点絶対値	230	364	倍精度浮動小数点コプロセッサ搭載製品でサポート
	DADD	倍精度浮動小数点加算	231	364	
	DCMPcm	倍精度浮動小数点比較	233	365	
	DDIV	倍精度浮動小数点除算	235	365	
	DMUL	倍精度浮動小数点乗算	239	369	
	DNEG	倍精度浮動小数点符号反転	241	369	
	DROUND	倍精度浮動小数点数→符号付き整数変換	246	372	
	DSQRT	倍精度浮動小数点平方根	248	372	
	DSUB	倍精度浮動小数点減算	250	372	
	DTOF	倍精度浮動小数点数→単精度浮動小数点数変換	252	373	
	DTOI	倍精度浮動小数点数→符号付き整数変換	254	373	
	DTOU	倍精度浮動小数点数→符号なし整数変換	256	373	
	FTOD	単精度浮動小数点数→倍精度浮動小数点数変換	258	374	
	ITOD	符号付き整数→倍精度浮動小数点数変換	260	374	
	UTOD	符号なし整数→倍精度浮動小数点数変換	264	376	
Protect Zone命令					
Protect Zone命令	TRCALL（特権命令）	Rootワールドコール	266	378	RXv3e以降でサポート
	TRMVFC（特権命令）	Rootワールドでの制御レジスタからの転送	267	378	
	TRMVTC（特権命令）	Rootワールドでの制御レジスタへの転送	268	379	
	TRRTE（特権命令）	Rootワールドの例外処理からの復帰	269	381	

3.2 RXv3e 拡張命令一覧

RXv3eアーキテクチャでは、RXv3アーキテクチャから4個の命令を追加（新規追加命令）し、8個の命令について仕様を拡張（仕様拡張命令）しています。

3.2.1 RXv3e 新規追加命令

表3.1にRXv3eで、RXv3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
Protect Zone命令	TRCALL	Rootワールドコール
	TRMVFC	Rootワールドでの制御レジスタからの転送
	TRMVTC	Rootワールドでの制御レジスタへの転送
	TRRTE	Rootワールドの例外処理からの復帰

3.2.2 RXv3e 仕様拡張命令

表3.2にRXv3eで、RXv3命令セットから仕様が拡張された命令一覧を示します。

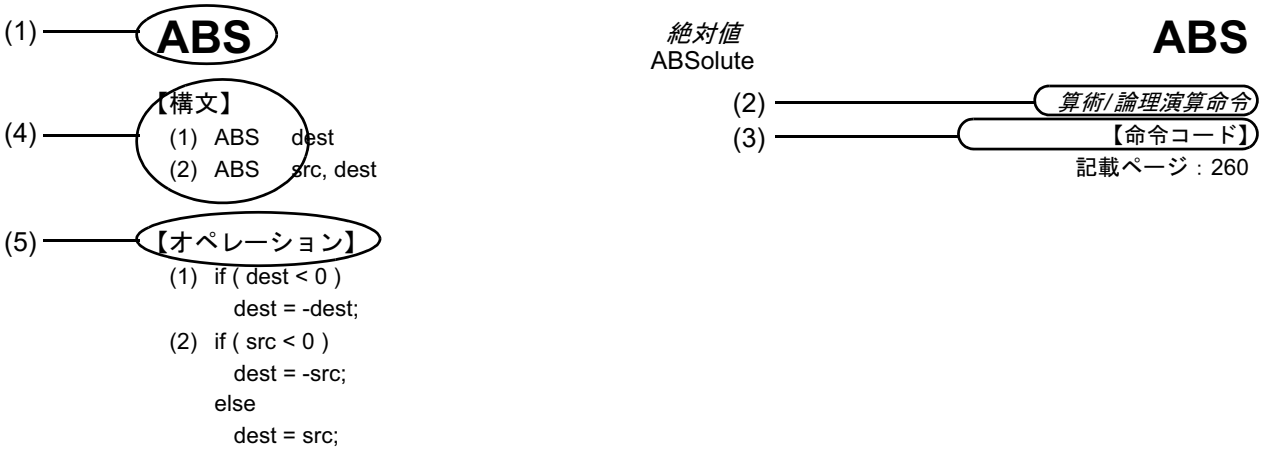
表3.2 仕様が拡張された命令一覧表

分類	ニーモニック	仕様拡張概要
転送命令	POPC	IVC、WID、WPRIレジスタを操作対象に追加
	PUSHC	
システム操作命令	INT	ソースオペランドにレジスタ指定を追加
	MVFC	IVC、WID、WPRIレジスタを操作対象に追加
	MVTC	
	MVTIPL	指定できる割り込みレベルを拡張
	RTE	Non-secure時のPSW.IPL[4]への書き込み制限追加
	RTFI	

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。

(8)【命令フォーマット】

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

(9)【記述例】

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

(1)　ニーモニック

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

(2)　命令の種類

命令の種類を示します。

(3) 命令コード

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

(4) 構文

命令の構文を記号で示しています。

(a) ニーモニック

ニーモニックを記述します。

(b) サイズ指定子 .size

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

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

(c) オペランド src、dest

オペランドを記述します。

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

(5) オペレーション

命令のオペレーションを示します。動作表記はC言語の表記方法に準じています。

(a) データタイプ

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

(b) 擬似関数

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

(c) 特殊表記

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

(6) 機能

命令の機能、注意事項を説明しています。

(7) フラグ変化

PSWのフラグ（O, S, Z, C）の変化を示します。

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

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

表中に示す記号の意味は以下のとおりです。

— ： 変化しません。
○ ： 条件によって変化します。

(8) 命令フォーマット

命令フォーマットを示します。

【命令フォーマット】

構文	処理サイズ	対象			コードサイズ（バイト）
		src	src2	dest	
(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

【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

【命令フォーマット】

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

(a) レジスタ

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

DSP機能命令のアキュムレータの指定のためにA0、A1が指定できます。

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

(b) 制御レジスタ

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

命令により指定可能な制御レジスタは異なります。各命令の説明を参照してください。

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

(c) フラグ、ビット

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

(d) 即値

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

(e) メモリオペランドに付加されるサイズ拡張指定子（.memex）

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

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

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

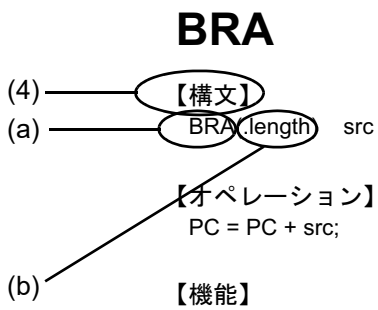
(f) 処理サイズ

処理サイズは、CPU内部での転送サイズ、または演算サイズを示します。

(9) 記述例

命令の記述例を示します。

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。



【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
```

(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

RXv3e各命令の詳細説明を示します。

3.4.1 標準搭載命令

次ページより標準搭載命令の詳細説明を示します。

ABS

絶対値
ABSolute

ABS

算術/論理演算命令

【命令コード】
記載ページ：274

【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

キャリー付き加算
ADd with Carry

ADC

算術/論理演算命令
【命令コード】
記載ページ：275

【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

キャリーなし加算
ADD

ADD

算術/論理演算命令
【命令コード】
記載ページ：276

【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：278

【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：280

【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
```

B*Cnd*

相対条件分岐
Branch Conditionally

B*Cnd*

【構文】
B*Cnd*(.length) src

分岐命令
【命令コード】
記載ページ：282

【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。
- B*Cnd*には以下の種類があります。

B*Cnd*	条件		式	B*Cnd*	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) B*Cnd*.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
```

BFMOV

ビットフィールド転送
Bit Field MOVe

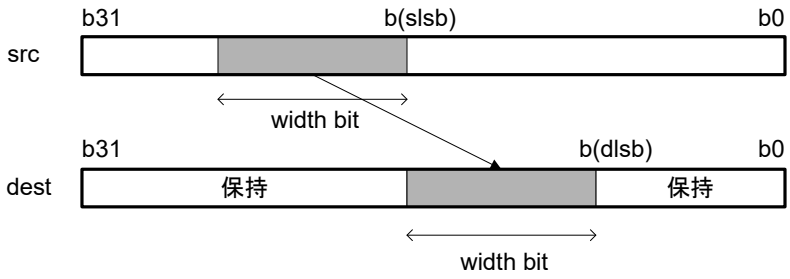
BFMOV

【構文】
BFMOV　slsb, dlsb, width, src, dest

転送命令
【命令コード】
記載ページ：283

【オペレーション】
```
unsigned long tmp1, tmp2;
tmp1 = (0FFFFFFFFh >> (32-width)) << dlsb;
tmp2 = (src >> slsb) << dlsb;
dest = (tmp2 & tmp1) | (dest & ~tmp1);
```

【機能】
- srcのビット位置slsbからwidthビット幅のデータをdestのビット位置dlsbへ転送します。destの残りのビットは保持されます。
- slsbの範囲は0≦slsb≦31、dlsbの範囲は0≦dlsb≦31、widthの範囲は1≦width≦31です。
- (slsb + width) > 32の場合、または(dlsb + width) > 32の場合、destは不定になります。



【フラグ変化】
- フラグは変化しません。

【命令フォーマット】

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

【記述例】
```
BFMOV　#5, #10, #3, R1, R2
```

BFMOVZ

ゼロへのビットフィールド転送
Bit Field MOVe to Zero

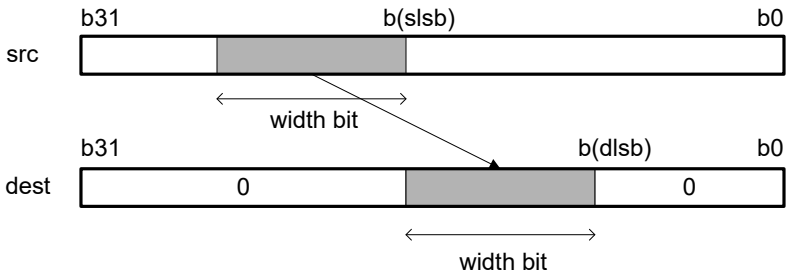
BFMOVZ

【構文】
BFMOVZ slsb, dlsb, width, src, dest

転送命令
【命令コード】
記載ページ：283

【オペレーション】
```
unsigned long tmp1, tmp2;
tmp1 = (0FFFFFFFFh >> (32-width))<<dlsb;
tmp2 = (src >> slsb) << dlsb;
dest = (tmp2 & tmp1);
```

【機能】
- srcのビット位置slsbからwidthビット幅のデータをdestのビット位置dlsbへ転送します。destの残りのビットは0になります。
- slsbの範囲は0≦slsb≦31、dlsbの範囲は0≦dlsb≦31、widthの範囲は1≦width≦31です。
- (slsb + width) > 32の場合、または(dlsb + width) > 32の場合、destは不定になります。



【フラグ変化】
- フラグは変化しません。

【命令フォーマット】

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

【記述例】
```
BFMOVZ  #5, #10, #3, R1, R2
```

BM*Cnd*

条件ビット転送
Bit Move Conditionally

BM*Cnd*

ビット操作命令
【命令コード】
記載ページ：284

【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：285

【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

【構文】
BRA(.length) src

【命令コード】
記載ページ：286

【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：287

【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せられます。

【フラグ変化】

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

【命令フォーマット】

構文	コードサイズ（バイト）
BRK	1

【記述例】

```
BRK
```

BSET

ビットセット
Bit SET

BSET

ビット操作命令
【命令コード】
記載ページ：287

【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：289

【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：290

【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

【構文】
CLRPSW dest

システム操作命令
【命令コード】
記載ページ：291

【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

【構文】
CMP src, src2

算術/論理演算命令
【命令コード】
記載ページ：292

【オペレーション】
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

【構文】
DIV src, dest

算術/論理演算命令
【命令コード】
記載ページ：293

【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

【構文】
DIVU src, dest

【命令コード】
記載ページ：295

【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：296

【構文】

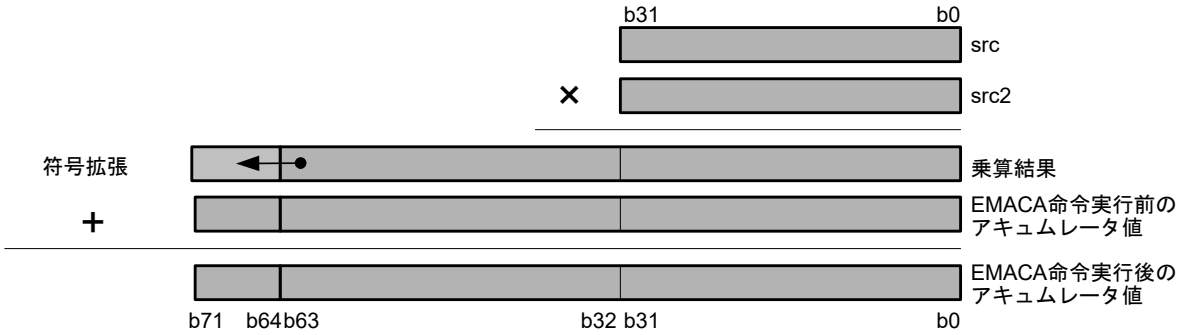
EMACA　src, src2, Adest

【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

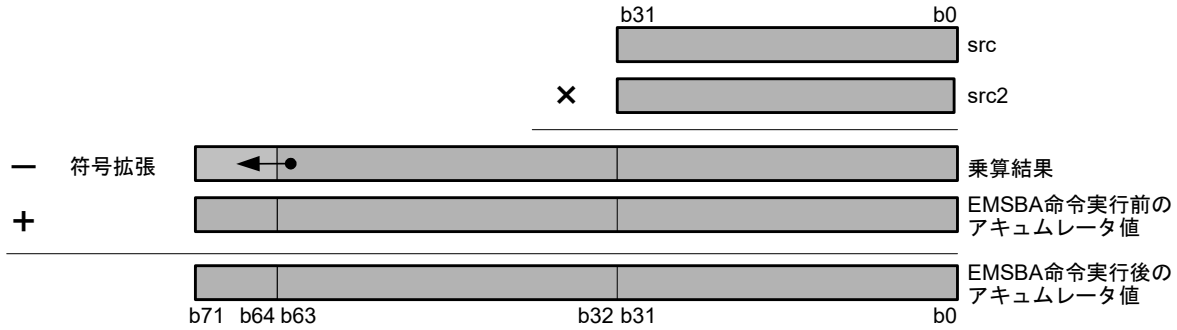
DSP機能命令

【構文】
EMSBA src, src2, Adest

【命令コード】
記載ページ：296

【オペレーション】
```
signed 72bit tmp;
tmp = (signed long) src * (signed long) src2;
Adest = Adest - tmp;
```

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



【フラグ変化】
- フラグは変化しません。

【命令フォーマット】

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

【記述例】
```
EMSBA R1, R2, A1
```

EMUL

符号付き乗算
Extended MULtiply, signed

EMUL

【構文】
EMUL src, dest

算術/論理演算命令
【命令コード】
記載ページ：297

【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

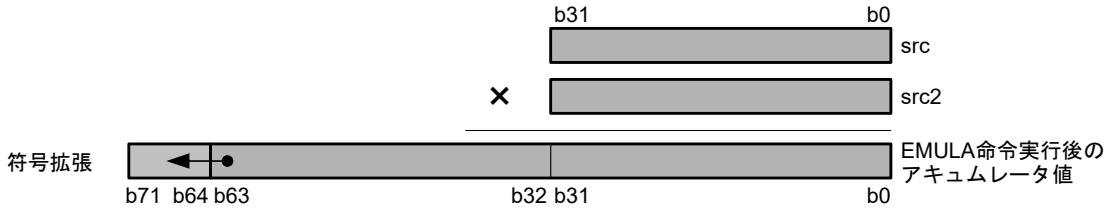
EMULA

【構文】
EMULA src, src2, Adest

DSP機能命令
【命令コード】
記載ページ：298

【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

【構文】
EMULU src, dest

算術/論理演算命令
【命令コード】
記載ページ：298

【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

単精度浮動小数点加算
Floating-point ADD

FADD

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

単精度浮動小数点演算命令
【命令コード】
記載ページ：300

【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。
- srcとdest、またはsrcとsrc2が反対の符号を持ち、その和が正確に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。

【単精度浮動小数点例外発生要因】

非実装処理
無効演算
オーバフロー
アンダフロー
精度異常

【記述例】

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

【動作補足説明】

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

DN = 0のとき

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

注. –∞方向への丸めモードの場合は–0、それ以外の丸めモードの場合は+0。

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. –∞方向への丸めモードの場合は–0、それ以外の丸めモードの場合は+0。

FCMP

単精度浮動小数点比較
Floating-point CoMPare

FCMP

【構文】
FCMP src, src2

単精度浮動小数点演算命令
【命令コード】
記載ページ：301

【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。

【単精度浮動小数点例外発生要因】

非実装処理
無効演算

【記述例】

```
FCMP　R1, R2
FCMP　[R1], R2
```

【動作補足説明】

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

DN = 0のとき

		src							
		正規化数	+0	–0	+ ∞	– ∞	非正規化数	QNaN	SNaN
src2	正規化数	比較							
	+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

単精度浮動小数点除算
Floating-point DIVide

FDIV

【構文】
FDIV src, dest

単精度浮動小数点演算命令
【命令コード】
記載ページ：302

【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。

【単精度浮動小数点例外発生要因】

非実装処理
無効演算
オーバフロー
アンダフロー
精度異常
ゼロ除算

【記述例】

```
FDIV    R1, R2
FDIV    [R1], R2
```

【動作補足説明】

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

DN = 0のとき

		src							
		正規化数	+0	–0	+ ∞	– ∞	非正規化数	QNaN	SNaN
dest	正規化数	除算	ゼロ除算		0				
	+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

単精度浮動小数点乗算
Floating-point MULtiply

FMUL

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

単精度浮動小数点演算命令
【命令コード】
記載ページ：303

【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． アキュムレータ (ACC0) を使用します。単精度浮動小数点例外発生有無にかかわらず、命令実行後の ACC0 の値は不定です。

【フラグ変化】

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

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

【命令フォーマット】

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

注. 弊社の「RXファミリ アセンブラ」では、ディスプレースメントの値（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。

【単精度浮動小数点例外発生要因】

非実装処理
無効演算
オーバフロー
アンダフロー
精度異常

【記述例】

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

【動作補足説明】

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

DN = 0のとき

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="8">src</td></tr>
<tr><td>正規化数</td><td>+0</td><td>–0</td><td>+ ∞</td><td>– ∞</td><td>非正規化数</td><td>QNaN</td><td>SNaN</td></tr>
<tr><td rowspan="8">dest
または
src2</td><td>正規化数</td><td colspan="3">乗算</td><td colspan="2">∞</td><td rowspan="3"></td><td rowspan="6"></td><td rowspan="7"></td></tr>
<tr><td>+0</td><td colspan="2">+0</td><td>–0</td><td colspan="2" rowspan="2">無効演算</td></tr>
<tr><td>–0</td><td colspan="2">–0</td><td>+0</td></tr>
<tr><td>+ ∞</td><td rowspan="2">∞</td><td colspan="2" rowspan="2">無効演算</td><td>+ ∞</td><td>– ∞</td></tr>
<tr><td>– ∞</td><td>– ∞</td><td>+ ∞</td></tr>
<tr><td>非正規化数</td><td colspan="5"></td><td>非実装処理</td></tr>
<tr><td>QNaN</td><td colspan="6"></td><td>QNaN</td></tr>
<tr><td>SNaN</td><td colspan="7"></td><td>無効演算</td></tr>
</table>

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

単精度浮動小数点平方根
Floating-point SQuare RooT

FSQRT

【構文】
FSQRT src, dest

単精度浮動小数点演算命令
【命令コード】
記載ページ：304

【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。

【単精度浮動小数点例外発生要因】
非実装処理
無効演算
精度異常

【記述例】

```
FSQRT   R1, R2
FSQRT   [R1], R2
```

【動作補足説明】

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

DN = 0のとき

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

DN = 1のとき

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

【無効演算発生時のQNaN生成規則】

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

注. 「表1.7 QNaN生成規則」に対応

FSUB

単精度浮動小数点減算
Floating-point SUBtract

FSUB

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

単精度浮動小数点演算命令
【命令コード】
記載ページ：305

【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。
- srcとdest、またはsrcとsrc2が同一の符号を持ち、その差が正確に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。

【単精度浮動小数点例外発生要因】

非実装処理
無効演算
オーバフロー
アンダフロー
精度異常

【記述例】

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

【動作補足説明】

- FPSWのDNビットが“0”の場合と“1”の場合について、src、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

<table>
<tr><th colspan="2" rowspan="2"></th><th colspan="8">src</th></tr>
<tr><th>正規化数</th><th>+0</th><th>–0</th><th>+ ∞</th><th>– ∞</th><th>非正規化数</th><th>QNaN</th><th>SNaN</th></tr>
<tr><td rowspan="8">dest または src2</td><td>正規化数</td><td colspan="3">減算</td><td rowspan="3">– ∞</td><td rowspan="3">+ ∞</td><td rowspan="6"></td><td rowspan="5"></td><td rowspan="4"></td></tr>
<tr><td>+0</td><td colspan="2">（注）</td><td>+0</td></tr>
<tr><td>–0</td><td colspan="2">–0</td><td>（注）</td></tr>
<tr><td>+ ∞</td><td colspan="3">+ ∞</td><td>無効演算</td></tr>
<tr><td>– ∞</td><td colspan="4">– ∞</td><td>無効演算</td></tr>
<tr><td>非正規化数</td><td colspan="6">非実装処理</td></tr>
<tr><td>QNaN</td><td colspan="7">QNaN</td></tr>
<tr><td>SNaN</td><td colspan="8">無効演算</td></tr>
</table>

注. –∞方向への丸めモードの場合は–0、それ以外の丸めモードの場合は+0。

DN = 1のとき

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="7">src</td></tr>
<tr><td>正規化数</td><td>+0、
+ 非正規化数</td><td>–0、
– 非正規化数</td><td>+ ∞</td><td>– ∞</td><td>QNaN</td><td>SNaN</td></tr>
<tr><td rowspan="7">dest
または
src2</td><td>正規化数</td><td rowspan="3">減算</td><td colspan="2" rowspan="3">（注）</td><td rowspan="3">– ∞</td><td rowspan="4">+ ∞</td><td rowspan="5"></td><td rowspan="6"></td></tr>
<tr><td>+0、+ 非正規化数</td><td>+0</td></tr>
<tr><td>–0、– 非正規化数</td><td>–0</td><td>（注）</td></tr>
<tr><td>+ ∞</td><td colspan="3">+ ∞</td><td>無効演算</td></tr>
<tr><td>– ∞</td><td colspan="4">– ∞</td><td>無効演算</td></tr>
<tr><td>QNaN</td><td colspan="5"></td><td>QNaN</td></tr>
<tr><td>SNaN</td><td colspan="6"></td><td>無効演算</td></tr>
</table>

注. –∞方向への丸めモードの場合は–0、それ以外の丸めモードの場合は+0。

FTOI

単精度浮動小数点数→符号付き整数変換
Float TO Integer

FTOI

【構文】
FTOI src, dest

単精度浮動小数点演算命令
【命令コード】
記載ページ：306

【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。

【単精度浮動小数点例外発生要因】
非実装処理
無効演算
精度異常

【記述例】

```
FTOI    R1, R2
FTOI    [R1], R2
```

【動作補足説明】

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

DN = 0のとき

srcの値（指数部はゲタなしの値）		dest	例外
src≧0	+∞	EV = 1で無効演算発生時：変化なし 上記以外：7FFFFFFFh	無効演算
	31≦指数部≦127		
	–126≦指数部≦30	00000000h～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
	–126≦指数部≦30	00000000h～80000080h	なし（注1）
	31≦指数部≦127	EV = 1で無効演算発生時：変化なし 上記以外：80000000h	無効演算（注2）
	–∞		
NaN	QNaN	EV = 1で無効演算発生時：変化なし 上記以外： 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 = 1で無効演算発生時：変化なし 上記以外：7FFFFFFFh	無効演算
	31≦指数部≦127		
	–126≦指数部≦30	00000000h～7FFFFF80h	なし（注1）
	+0、+非正規化数	00000000h	なし
src < 0	–0、–非正規化数		
	–126≦指数部≦30	00000000h～80000080h	なし（注1）
	31≦指数部≦127	EV = 1で無効演算発生時：変化なし 上記以外：80000000h	無効演算（注2）
	–∞		
NaN	QNaN	EV = 1で無効演算発生時：変化なし 上記以外： 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

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

FTOU

【構文】
FTOU src, dest

単精度浮動小数点演算命令
【命令コード】
記載ページ：306

【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。

【単精度浮動小数点例外発生要因】
非実装処理
無効演算
精度異常

【記述例】

```
FTOU   R1, R2
FTOU   [R1], R2
```

【動作補足説明】

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

DN = 0のとき

srcの値（指数部はゲタなしの値）		dest	例外
src≧0	+∞	EV = 1で無効演算発生時：変化なし 上記以外：FFFFFFFFh	無効演算
	32≦指数部≦127		
	–126≦指数部≦31	00000000h～FFFFFF00h	なし（注）
	+非正規化数	変化なし	非実装
	+0	00000000h	なし
src < 0	–0		
	–非正規化数	変化なし	非実装
	–正規化数、–∞	EV = 1で無効演算発生時：変化なし 上記以外：00000000h	無効演算
NaN	QNaN	EV = 1で無効演算発生時：変化なし 上記以外： 最上位ビット= 0：FFFFFFFFh 最上位ビット= 1：00000000h	無効演算
	SNaN		

注. 丸め発生時は、精度異常が発生します。

DN = 1のとき

srcの値（指数部はゲタなしの値）		dest	例外
src≧0	+∞	EV = 1で無効演算発生時：変化なし 上記以外：FFFFFFFFh	無効演算
	32≦指数部≦127		
	–126≦指数部≦31	00000000h～FFFFFF00h	なし（注）
	+0、+非正規化数	00000000h	なし
src < 0	–0、–非正規化数		
	–正規化数、–∞	EV = 1で無効演算発生時：変化なし 上記以外：00000000h	無効演算
NaN	QNaN	EV = 1で無効演算発生時：変化なし 上記以外： 符号ビット= 0：FFFFFFFFh 符号ビット= 1：00000000h	無効演算
	SNaN		

注. 丸め発生時は、精度異常が発生します。

INT

ソフトウェア割り込み
INTerrupt

INT

【構文】
INT src

システム操作命令
【命令コード】
記載ページ：307

【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。
- srcがレジスタのとき、LSB側8ビットのみ有効です。
- IVC.UIRの値、アドレッシングモード、srcの値に応じて特権命令となります。詳細は割り込みベクタ制御レジスタ（IVC）を参照してください。

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

【命令フォーマット】

構文	対象	コードサイズ（バイト）
	src	
INT src	#IMM:8	3
	Rsrc	4

【記述例】
```
INT    #0
INT    R1
```

ITOF

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

ITOF

【構文】
ITOF src, dest

単精度浮動小数点演算命令
【命令コード】
記載ページ：308

【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。

【単精度浮動小数点例外発生要因】

精度異常

【記述例】

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

JMP

無条件分岐
JuMP

JMP

分岐命令
【命令コード】
記載ページ：309

【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（バイト）
	src	
JMP　src	Rs	2

【記述例】
JMP　R1

JSR

サブルーチン分岐
Jump SubRoutine

JSR

分岐命令

【構文】
JSR src

【命令コード】
記載ページ：309

【オペレーション】
SP = SP - 4;
*SP = (PC + 2);（注）
PC = src;

注 . (PC + 2) は JSR 命令の次の命令の番地です。

【機能】
- srcが示すサブルーチンへ分岐します。

【フラグ変化】
- フラグは変化しません。

【命令フォーマット】

構文	対象	コードサイズ（バイト）
	src	
JSR src	Rs	2

【記述例】
JSR R1

MACHI

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

MACHI

【構文】
MACHI　src, src2, Adest

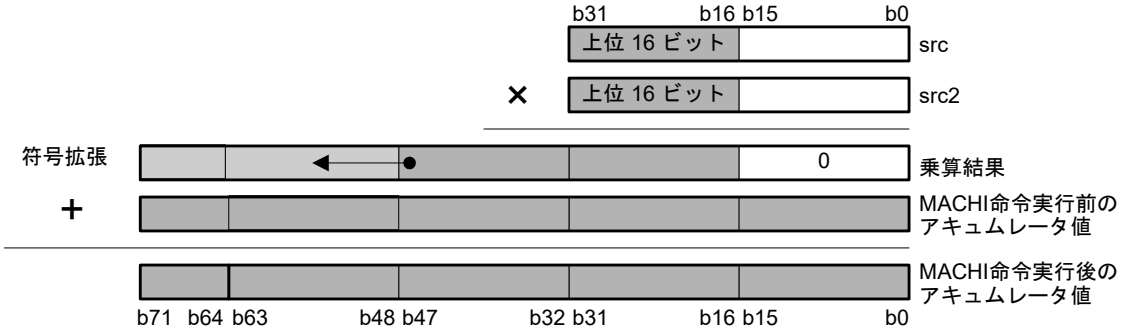
DSP機能命令
【命令コード】
記載ページ：310

【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

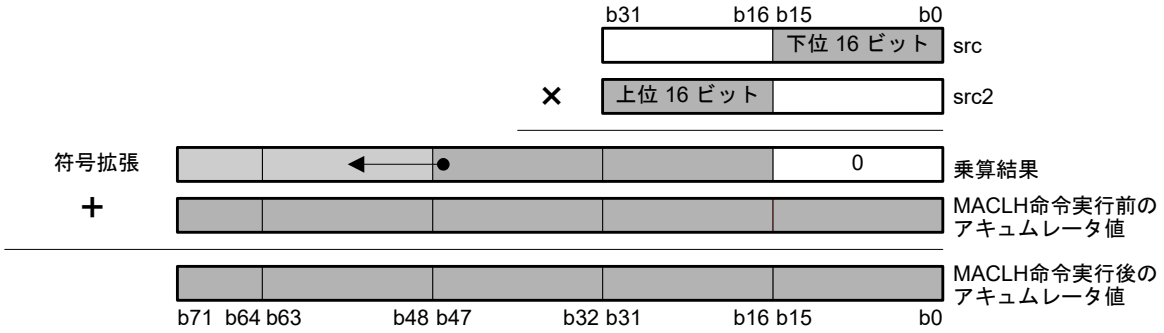
MACLH

DSP機能命令
【命令コード】
記載ページ：310

【構文】
MACLH　src, src2, Adest

【オペレーション】
```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) src;
tmp2 = (signed short) (src2 >> 16);
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = Adest + (tmp3 << 16);
```

【機能】
- srcの下位16ビットとsrc2の上位16ビットの乗算を行い、乗算結果とアキュムレータの加算を行います。ただし、乗算結果の最下位ビットはアキュムレータのb16にあわせて加算します。加算結果はアキュムレータに格納されます。srcの下位16ビットとsrc2の上位16ビットは符号付き整数として扱われます。



【フラグ変化】
- フラグは変化しません。

【命令フォーマット】

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

【記述例】
MACLH　R1, R2, A1

MACLO

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

MACLO

【構文】
MACLO src, src2, Adest

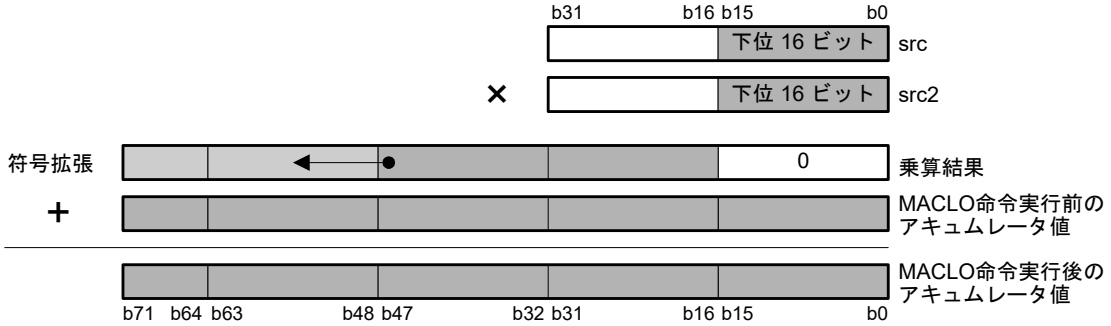
DSP機能命令
【命令コード】
記載ページ：310

【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：311

【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

【構文】
MIN src, dest

【命令コード】
記載ページ：312

【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

【構文】
MOV.size src, dest

転送命令
【命令コード】
記載ページ：313

【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

【構文】
MOVCO src, dest

【命令コード】
記載ページ：318

【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を“1”にします。

注．
1. 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。
2. 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

【構文】
MOVLI src, dest

転送命令
【命令コード】
記載ページ：318

【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、LIフラグを“1”にします。

注 . 1. 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
2. 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

【構文】
MOVU.size src, dest

【命令コード】
記載ページ：319

【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

【構文】
MSBHI src, src2, Adest

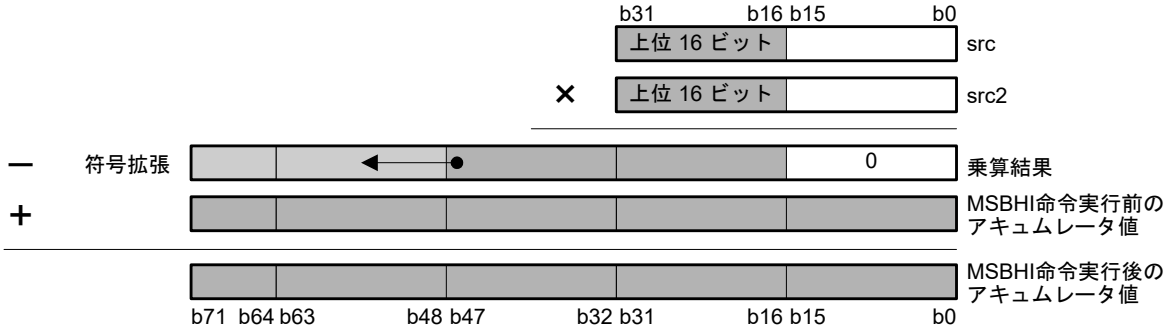
【命令コード】
記載ページ：320

【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

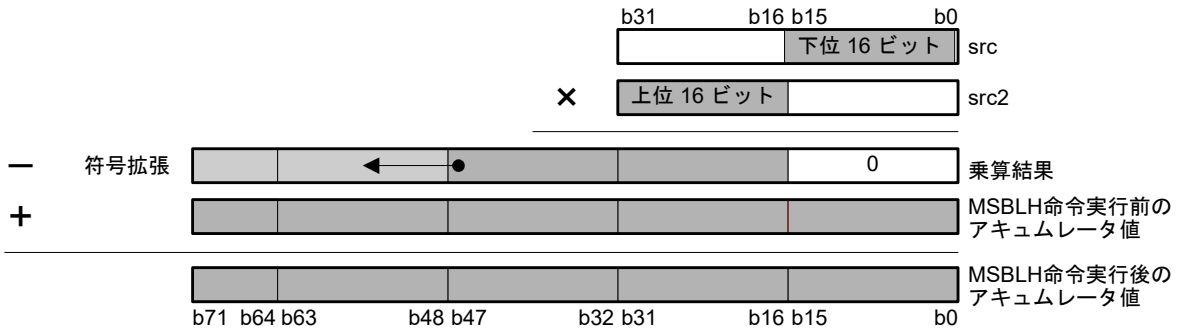
MSBLH

DSP機能命令
【命令コード】
記載ページ：320

【構文】
MSBLH　src, src2, Adest

【オペレーション】
```
signed short tmp1, tmp2;
signed 72bit tmp3;
tmp1 = (signed short) src;
tmp2 = (signed short) (src2 >> 16);
tmp3 = (signed long) tmp1 * (signed long) tmp2;
Adest = Adest - (tmp3 << 16);
```

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



【フラグ変化】
- フラグは変化しません。

【命令フォーマット】

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

【記述例】
```
MSBLH　R1, R2, A1
```

MSBLO

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

MSBLO

【構文】
MSBLO　src, src2, Adest

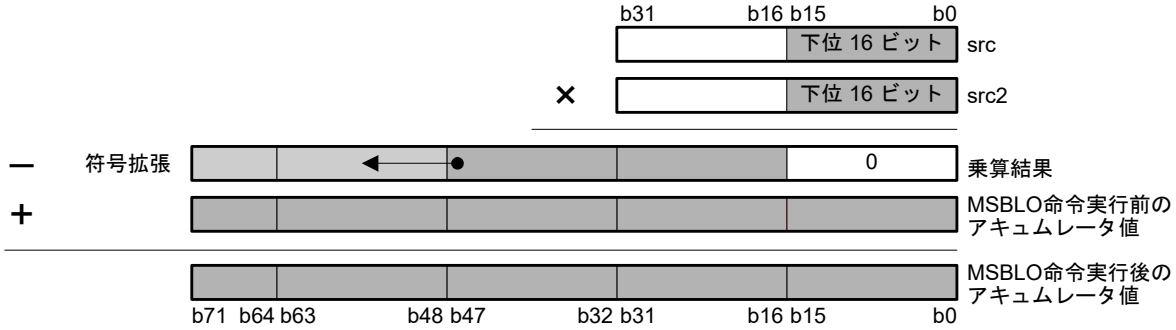
DSP機能命令
【命令コード】
記載ページ：321

【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：321

【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

【構文】
MULHI src, src2, Adest

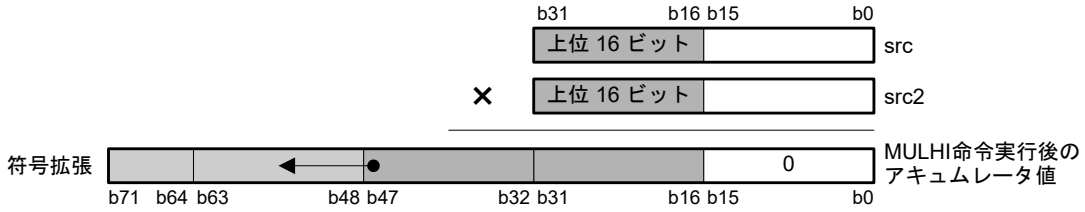
DSP機能命令
【命令コード】
記載ページ：323

【オペレーション】

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

【機能】

- srcの上位16ビットとsrc2の上位16ビットの乗算を行い、その結果をアキュムレータに格納します。ただし、乗算結果の最下位ビットはアキュムレータのb16にあわせ、アキュムレータのb71～b48に対応する部分は、符号拡張されます。また、アキュムレータのb15～b0は、“0”になります。srcの上位16ビットとsrc2の上位16ビットは符号付き整数として扱われます。



【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

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

【記述例】

```
MULHI   R1, R2, A1
```

MULLH

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

MULLH

【構文】
MULLH src, src2, Adest

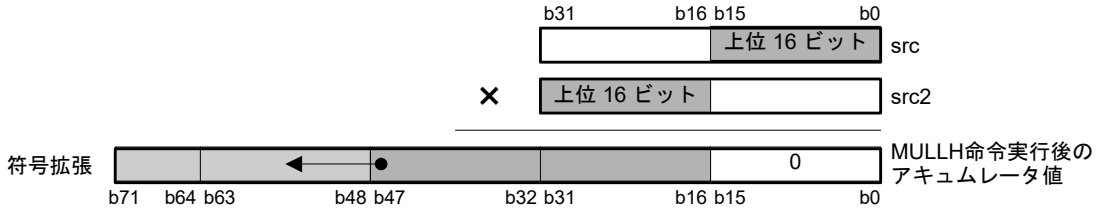
DSP機能命令
【命令コード】
記載ページ：323

【オペレーション】

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

【機能】

- srcの下位16ビットとsrc2の上位16ビットの乗算を行い、その結果をアキュムレータに格納します。ただし、乗算結果の最下位ビットはアキュムレータのb16にあわせ、アキュムレータのb71～b48に対応する部分は、符号拡張されます。また、アキュムレータのb15～b0は、“0”になります。srcの下位16ビットとsrc2の上位16ビットは符号付き整数として扱われます。



【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

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

【記述例】

```
MULLH  R1, R2, A1
```

MULLO

下位16ビット乗算
MULtiply LOw-order word

MULLO

【構文】
MULLO src, src2, Adest

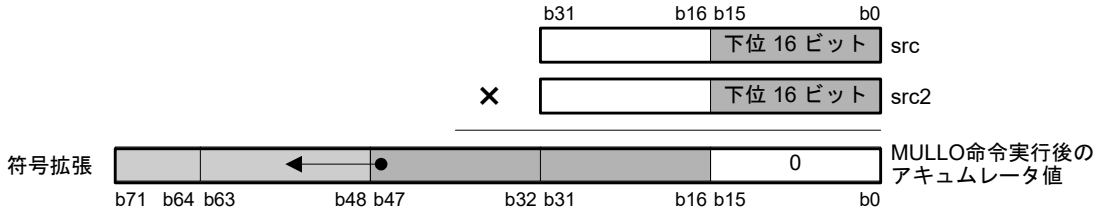
DSP機能命令
【命令コード】
記載ページ：324

【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

【構文】
MVFACGU src, Asrc, dest

DSP機能命令
【命令コード】
記載ページ：324

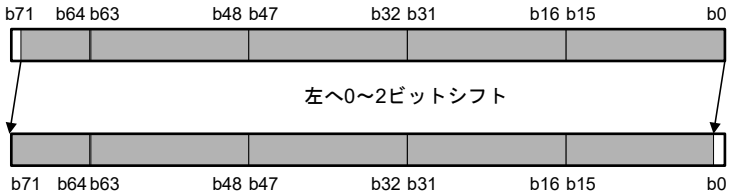
【オペレーション】

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

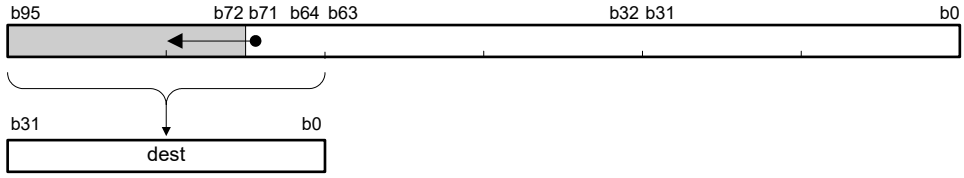
【機能】

- MVFACGU命令は、以下のような手順で実行されます。

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



処理2. シフト後の値の最上位32ビットの内容をdestに転送します。



【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

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

【記述例】

```
MVFACGU  #1, A1, R1
```

MVFACHI

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

MVFACHI

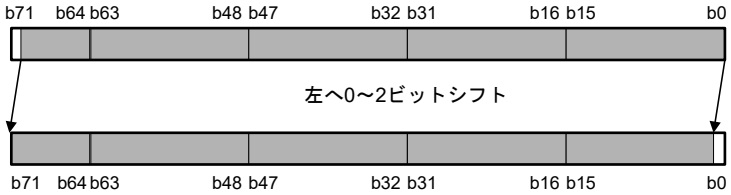
【構文】
MVFACHI src, Asrc, dest

DSP機能命令
【命令コード】
記載ページ：325

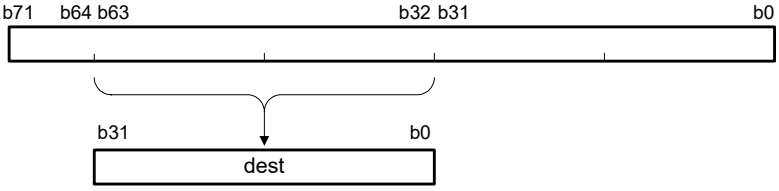
【オペレーション】
```
signed 72bit tmp;
tmp = (signed 72bit) Asrc << src;
dest = (signed long) (tmp >> 32);
```

【機能】
- MVFACHI命令は、以下のような手順で実行されます。

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



処理2. シフト後の値のb63～b32の内容をdestに転送します。



【フラグ変化】
- フラグは変化しません。

【命令フォーマット】

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

【記述例】
```
MVFACHI  #1, A1, R1
```

MVFACLO

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

MVFACLO

【構文】
MVFACLO src, Asrc, dest

DSP機能命令
【命令コード】
記載ページ：325

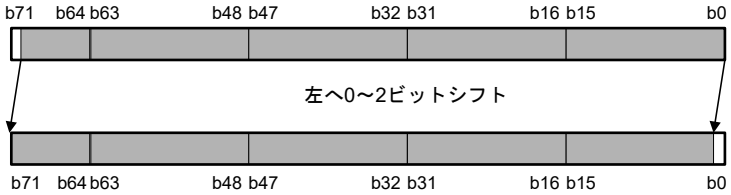
【オペレーション】

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

【機能】

- MVFACLO命令は、以下のような手順で実行されます。

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



処理2. シフト後の値のb31～b0の内容をdestに転送します。



【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

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

【記述例】

```
MVFACLO  #1, A1, R1
```

MVFACMI

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

MVFACMI

【構文】
MVFACMI src, Asrc, dest

DSP機能命令
【命令コード】
記載ページ：326

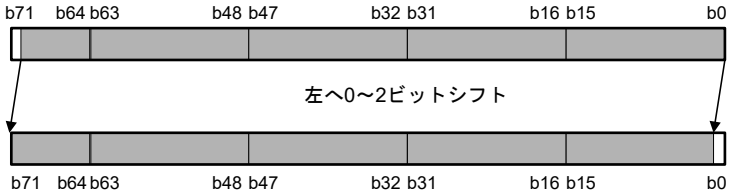
【オペレーション】

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

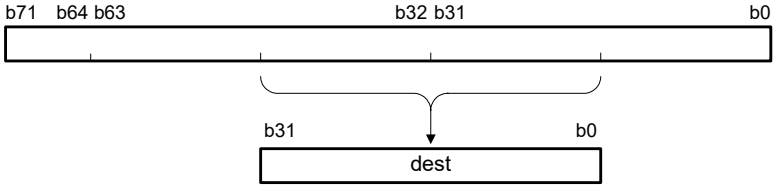
【機能】

- MVFACMI命令は、以下のような手順で実行されます。

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



処理2. シフト後の値のb47～b16の内容をdestに転送します。



【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

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

【記述例】

```
MVFACMI #1, A1, R1
```

MVFC

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

MVFC

【構文】
MVFC src, dest

システム操作命令
【命令コード】
記載ページ：327

【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。
- Secureワールド、Non-secureワールドでの FINTVの読み出し値は“0”です。
- Non-secureワールドでの WPRIの読み出し値は“0”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（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, IVC, WID, WPRI

【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

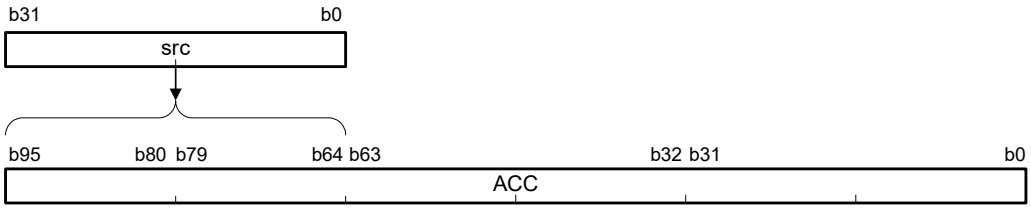
MVTACGU

DSP機能命令
【命令コード】
記載ページ：328

【構文】
MVTACGU src, Adest

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

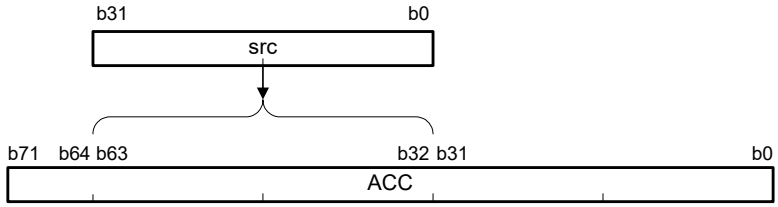
MVTACHI

DSP機能命令
【命令コード】
記載ページ：328

【構文】
MVTACHI src, Adest

【オペレーション】
Adest = (Adest & FF00000000FFFFFFFFh) | ((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

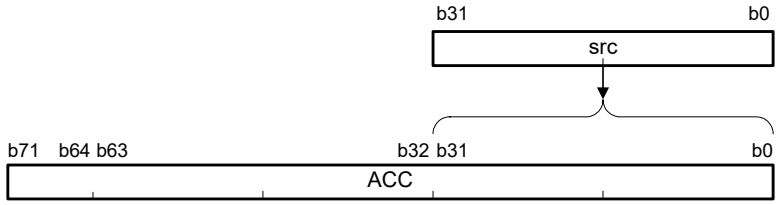
MVTACLO

【構文】
MVTACLO src, Adest

DSP機能命令
【命令コード】
記載ページ：328

【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：329

【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、IVC、WID、WPRIと、PSWのIPL[4:0]、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。
- Non-secureワールドでは PSW、BPSWへの書き込みを行った場合、IPL[4]ビットへは常に“0”が書き込まれます。
- Secureワールド、Non-secureワールドでの FINTV、IVC、WIDへの書き込みは無視されます。
- Non-secureワールドでの WPRI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, IVC, WID, WPRI
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：330

【構文】
MVTIPL src

【オペレーション】
```
if ( WID.WID == 2 )
   IPL = src & 0Fh;
else
   IPL = src;
```

【機能】
- srcをPSWのIPL[4:0]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≦31です。
- Non-secureワールドでは、IPL[4]ビットへは“0”が書き込まれます。

【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	
MVTIPL src	#IMM:5	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：331

【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：331

【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：332

【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：333

【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：334

【構文】
POP dest

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

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

【フラグ変化】
- フラグは変化しません。

【命令フォーマット】

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

【記述例】
```
POP R1
```

POPC

制御レジスタの復帰
POP Control register

POPC

【構文】
POPC dest

転送命令
【命令コード】
記載ページ：335

【オペレーション】

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

【機能】

- スタックから復帰させたデータを、destで示される制御レジスタに転送します。
- 使用されるスタックポインタは、PSWのUビットで示すスタックポインタになります。
- ユーザモードでは、ISP、INTB、EXTB、BPC、BPSW、FINTV、IVC、WID、WPRIと、PSWのIPL[4:0]、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。
- Non-secureワールドでは PSW、BPSWへの書き込みを行った場合、IPL[4]ビットへは常に“0”が書き込まれます。
- Secureワールド、Non-secureワールドでの FINTV、IVC、WIDへの書き込みは無視されます。
- Non-secureワールドでの WPRI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 dest（注）	コードサイズ （バイト）
POPC dest	L	Rx	2

注． 選択可能なdest：ISP, USP, INTB, EXTB, PSW, BPC, BPSW, FINTV, FPSW, IVC, WID, WPRI
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

【構文】
POPM dest-dest2

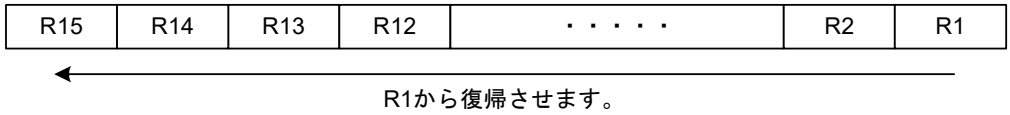
転送命令
【命令コード】
記載ページ：335

【オペレーション】

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

【機能】

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



【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

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

【記述例】

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

PUSH

スタックへデータ退避
PUSH data onto the stack

PUSH

【構文】
PUSH.size src

転送命令
【命令コード】
記載ページ：336

【オペレーション】

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

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

【機能】

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

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

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

注．弊社の「RXファミリ アセンブラ」では、ディスプレースメントの値（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：337

【構文】
PUSHC src

【オペレーション】
```
tmp = src;
SP = SP - 4;
*SP = tmp;
```

【機能】
- srcで示される制御レジスタをスタックに退避させます。
- 使用されるスタックポインタは、PSWのUビットで示すスタックポインタになります。
- srcにPCを指定した場合、本命令の番地をスタックに退避させます。
- Secureワールド、Non-secureワールドでの FINTVの読み出し値は“0”です。
- Non-secureワールドでの WPRIの読み出し値は“0”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（注）	
PUSHC src	L	Rx	2

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

【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：337

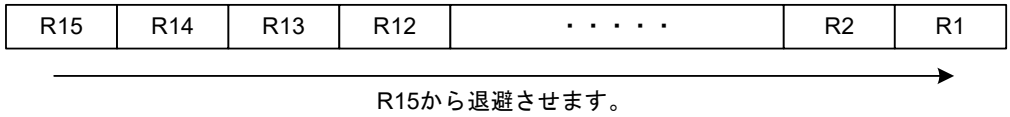
【構文】
PUSHM src-src2

【オペレーション】

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

【機能】

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



【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

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

【記述例】

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

RACL

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

RACL

【構文】
RACL src, Adest

DSP機能命令
【命令コード】
記載ページ：338

【オペレーション】

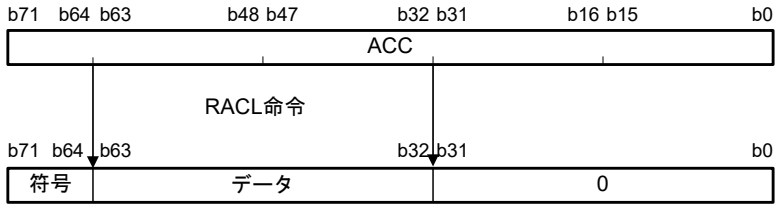
```
signed 72bit tmp;
signed 73bit tmp73;

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 = tmp73 & FFFFFFFFFF00000000h;
```

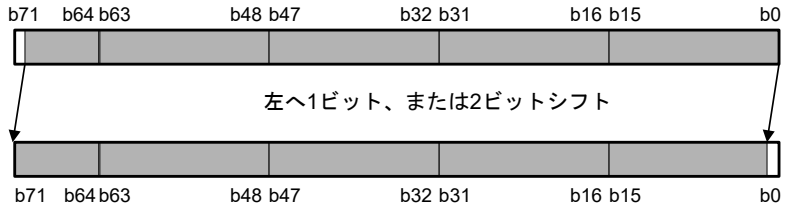
【機能】

- 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。

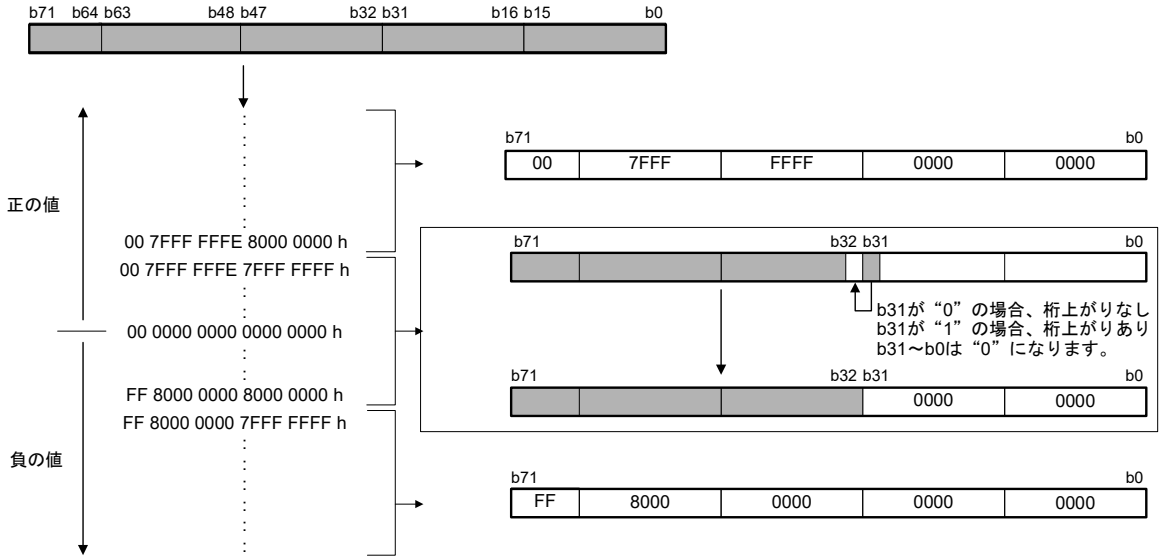


- RACL命令は、以下のような手順で実行されます。

  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	Adest	
RACL src, Adest	#IMM:1 (IMM:1 = 1, 2)	A0, A1	3

【記述例】

```
RACL  #1, A1
RACL  #2, A0
```

RACW

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

RACW

【構文】
RACW src, Adest

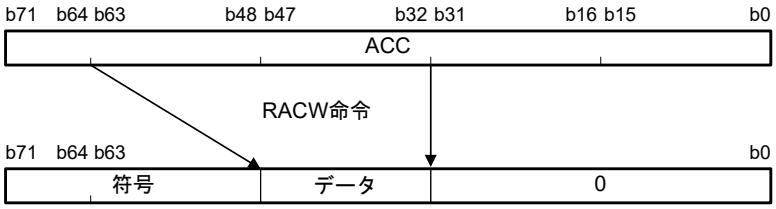
DSP機能命令
【命令コード】
記載ページ：338

【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 = tmp73 & FFFFFFFFFF00000000h;
```

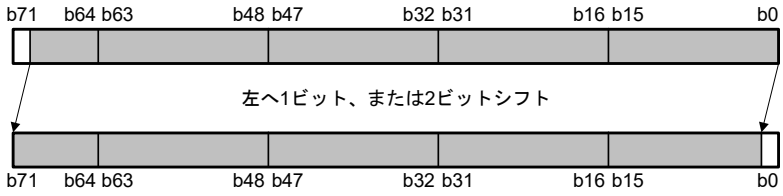
【機能】

- 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。

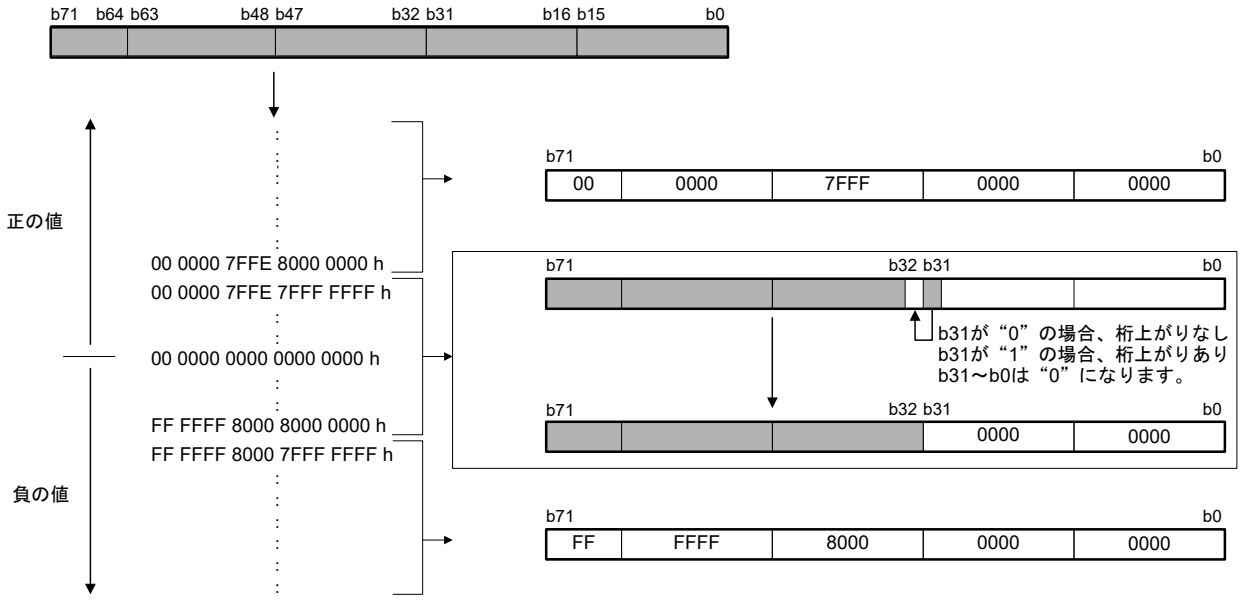


- RACW命令は、以下のような手順で実行されます。

  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

【構文】
RDACL src, Adest

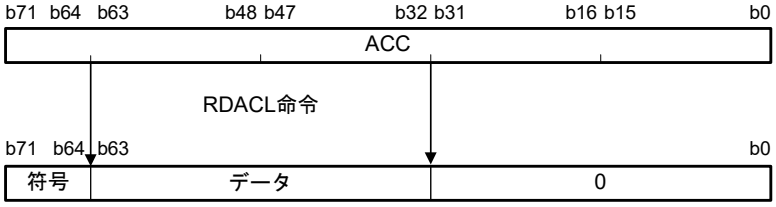
DSP機能命令
【命令コード】
記載ページ：339

【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;
```

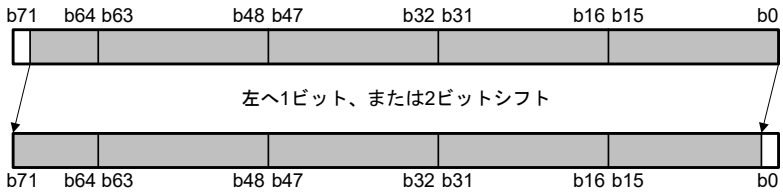
【機能】

- 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。

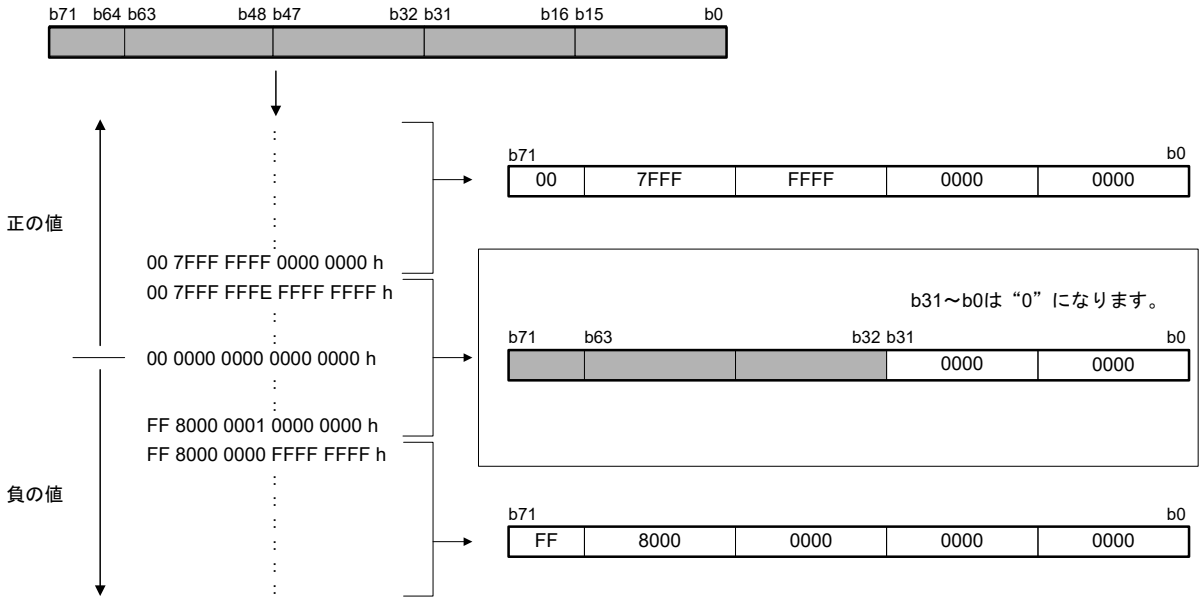


- RDACL命令は、以下のような手順で実行されます。

  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

【構文】
RDACW src, Adest

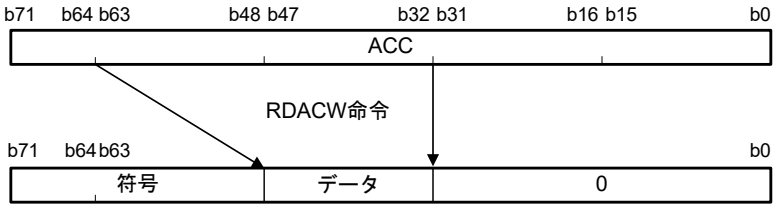
DSP機能命令
【命令コード】
記載ページ：339

【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。

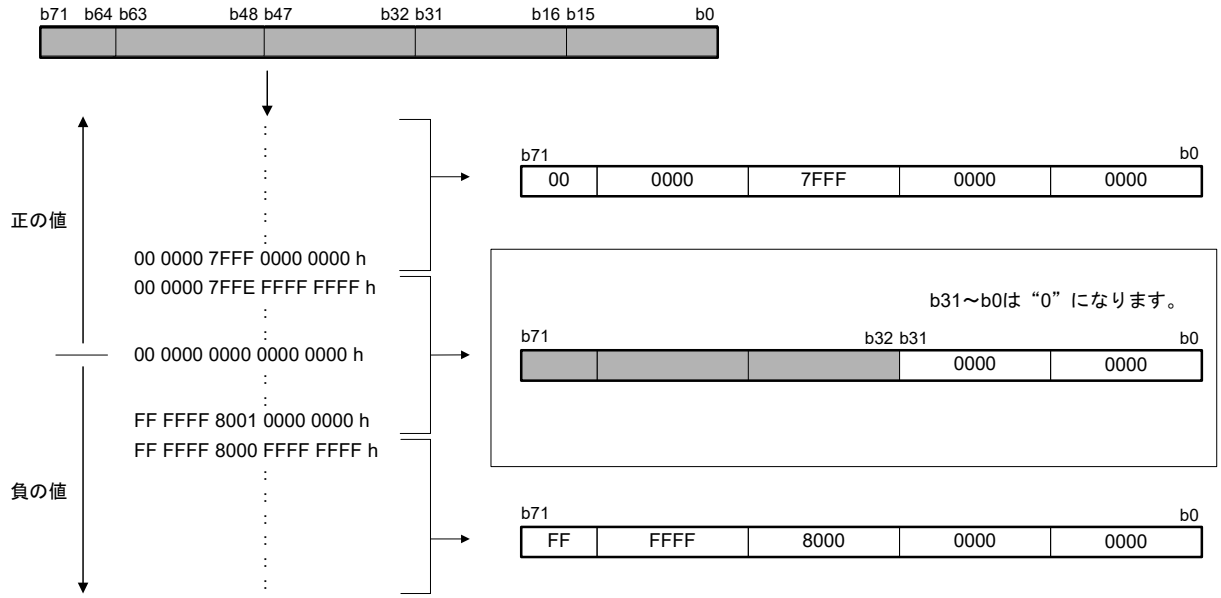


- RDACW命令は、以下のような手順で実行されます。

  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：340

【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：340

【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

【構文】
RMPA.size

算術/論理演算命令
【命令コード】
記載ページ：341

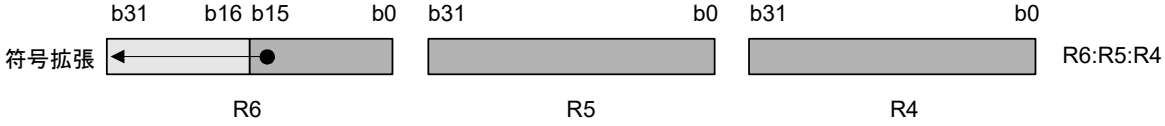
【オペレーション】

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

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

【機能】

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



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

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

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	○	R6のMSBが“1”のとき“1”、それ以外のとき“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

キャリー付き左回転
ROtate Left with Carry

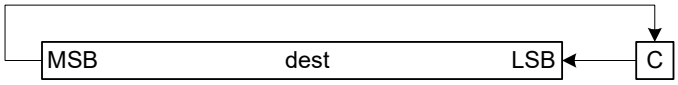
ROLC

算術/論理演算命令
【命令コード】
記載ページ：341

【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
		dest	（バイト）
ROLC dest	L	Rd	2

【記述例】
```
ROLC R1
```

RORC

キャリー付き右回転
ROtate Right with Carry

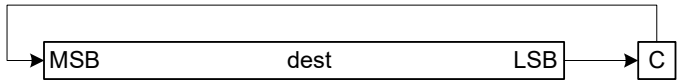
RORC

算術/論理演算命令
【命令コード】
記載ページ：341

【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

【構文】
ROTL src, dest

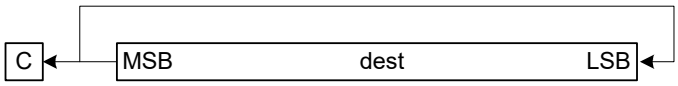
算術/論理演算命令
【命令コード】
記載ページ：342

【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

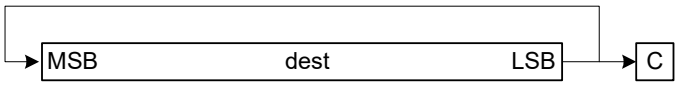
算術/論理演算命令

【命令コード】
記載ページ：342

【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

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

ROUND

【構文】
ROUND src, dest

単精度浮動小数点演算命令
【命令コード】
記載ページ：343

【オペレーション】
dest = (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。

【単精度浮動小数点例外発生要因】

非実装処理
無効演算
精度異常

【記述例】

```
ROUND   R1, R2
ROUND   [R1], R2
```

【動作補足説明】

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

DN = 0のとき

srcの値（指数部はゲタなしの値）		dest	例外
src≧0	+∞	EV = 1で無効演算発生時：変化なし 上記以外：7FFFFFFFh	無効演算
	31≦指数部≦127		
	–126≦指数部≦30	00000000h～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
	–126≦指数部≦30	00000000h～80000080h	なし（注1）
	31≦指数部≦127	EV = 1で無効演算発生時：変化なし 上記以外：80000000h	無効演算（注2）
	–∞		
NaN	QNaN	EV = 1で無効演算発生時：変化なし 上記以外： 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 = 1で無効演算発生時：変化なし 上記以外：7FFFFFFFh	無効演算
	31≦指数部≦127		
	–126≦指数部≦30	00000000h～7FFFFF80h	なし（注1）
	+0、+非正規化数	00000000h	なし
src < 0	–0、–非正規化数		
	–126≦指数部≦30	00000000h～80000080h	なし（注1）
	31≦指数部≦127	EV = 1で無効演算発生時：変化なし 上記以外：80000000h	無効演算（注2）
	–∞		
NaN	QNaN	EV = 1で無効演算発生時：変化なし 上記以外： 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：343

【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させたPCとPSWを復帰させ、例外処理ルーチンから戻ります。
- この命令は特権命令です。ユーザモードで実行すると特権命令例外が発生します。
- ユーザモードに移行する場合、PSWのUビットは“1”になります。
- Non-secureワールドで実行した場合、PSW.IPL[4]ビットへは常に“0”が書き込まれます。

【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：343

【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させたPCとPSWを、それぞれBPC、BPSWから復帰させ、高速割り込み処理ルーチンから戻ります。
- この命令は特権命令です。ユーザモードで実行すると特権命令例外が発生します。
- ユーザモードに移行する場合、PSWのUビットは“1”になります。
- 命令終了時のBPC、BPSWの値は不定になります。
- Non-secureワールドで実行した場合、PSW.IPL[4]ビットへは常に“0”が書き込まれます。

【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：344

【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：344

【構文】

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

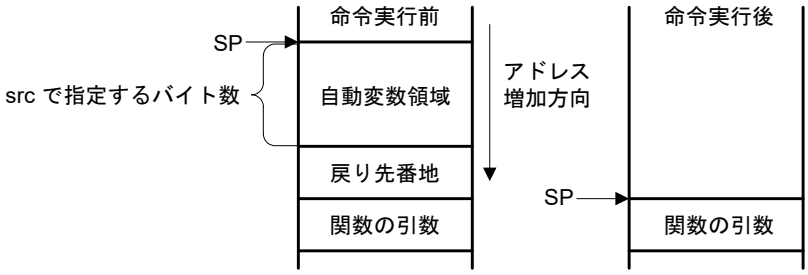
【オペレーション】

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

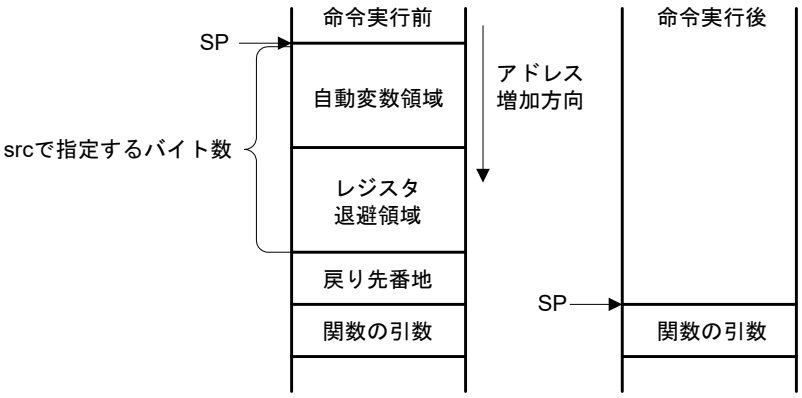
(2)
```
signed char i;
SP = SP + ( src - ( register_num(dest2) - register_num(dest) +1 ) * 4 );
for ( i = register_num(dest); i <= register_num(dest2); i++ ) {
  tmp = *SP;
  SP = SP + 4;
  register(i) = tmp;
}
PC = *SP;
SP = SP + 4;
```

【機能】

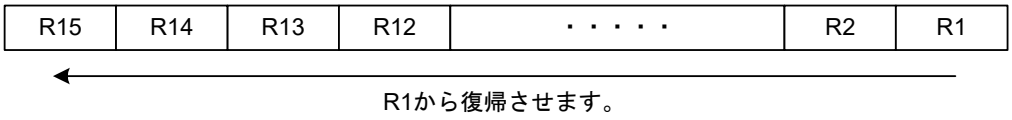
(1) スタックフレームを解放後、サブルーチンから復帰します。
- srcがスタックフレーム（自動変数領域）のサイズになるように指定してください。



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



【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

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

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

【記述例】

```
RTSD    #4
RTSD    #16, R5-R7
```

SAT

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

SAT

【構文】
SAT dest

算術/論理演算命令
【命令コード】
記載ページ：344

【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

【構文】
SATR

算術/論理演算命令
【命令コード】
記載ページ：345

【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

【構文】
SBB src, dest

【命令コード】
記載ページ：345

【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

【構文】
SC*Cnd*.size dest

【命令コード】

記載ページ：346

【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：346

【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/復帰させてください。

【フラグ変化】

フラグ	変化	条件
C	○	（*R1 - *R2）を符号なしで演算した結果、0 以上のとき“1”、それ以外のとき“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：347

【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

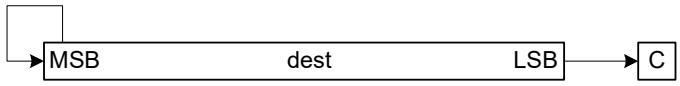
【構文】
(1) SHAR src, dest
(2) SHAR src, src2, dest

【命令コード】
記載ページ：348

【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：349

【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);

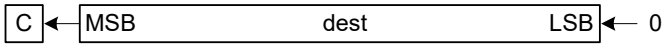
【機能】

(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：350

【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);

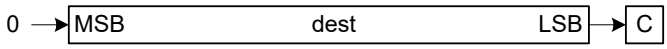
【機能】

(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：350

【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/復帰させてください。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	コードサイズ（バイト）
SMOVB	B	2

【記述例】
SMOVB

SMOVF

順方向ストリング転送
Strings MOVe Forward

SMOVF

ストリング操作命令
【命令コード】
記載ページ：351

【構文】
SMOVF

【オペレーション】

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

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

【機能】

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

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	コードサイズ（バイト）
SMOVF	B	2

【記述例】
SMOVF

SMOVU

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

SMOVU

ストリング操作命令
【命令コード】
記載ページ：351

【構文】
SMOVU

【オペレーション】

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

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

【機能】

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

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	コードサイズ（バイト）
SMOVU	B	2

【記述例】

```
SMOVU
```

SSTR

ストリングストア
Strings SToRe

SSTR

【構文】
SSTR.size

ストリング操作命令
【命令コード】
記載ページ：351

【オペレーション】

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

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

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

【フラグ変化】
- フラグは変化しません。

【命令フォーマット】

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

【記述例】
SSTR.W

STNZ

条件付き転送
STore on Not Zero

STNZ

転送命令

【構文】
STNZ src, dest

【命令コード】

記載ページ：352

【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：353

【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：354

【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：355

【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/復帰させてください。

【フラグ変化】

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

【命令フォーマット】

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

【記述例】

```
SUNTIL.W
```

SWHILE

ストリングサーチ
Search WHILE unequal string

SWHILE

【構文】
SWHILE.size

ストリング操作命令
【命令コード】
記載ページ：355

【オペレーション】

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

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

【機能】

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

【フラグ変化】

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

【命令フォーマット】

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

【記述例】

SWHILE.W

TST

テスト
TeST logical

TST

【構文】
TST src, src2

算術/論理演算命令
【命令コード】
記載ページ：356

【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

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

UTOF

【構文】
UTOF src, dest

単精度浮動小数点演算命令
【命令コード】
記載ページ：357

【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。

【単精度浮動小数点例外発生要因】
精度異常

【記述例】
```
UTOF    R1, R2
UTOF    [R1], R2
UTOF    16[R1].L, R2
```

WAIT

ウェイト
WAIT

WAIT

【構文】
WAIT

システム操作命令
【命令コード】
記載ページ：358

【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せられるPCは、WAIT命令の次のアドレスになります。

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

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

構文	コードサイズ（バイト）
WAIT	2

【記述例】
WAIT

XCHG

交換
eXCHanGe

XCHG

転送命令

【構文】
XCHG　src, dest

【命令コード】
記載ページ：358

【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：359

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

【オペレーション】
(1) dest = dest ^ src;
(2) dest = src2 ^ src;

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

【フラグ変化】

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

【命令フォーマット】

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

注. 弊社の「RXファミリ アセンブラ」では、ディスプレースメントの値（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
XOR    R1, R2, R3
```

3.4.2 レジスター括退避機能命令

次ページよりレジスター括退避機能命令の詳細説明を示します。

RSTR

レジスタの一括復帰
ReSToRe registers

RSTR

【構文】
RSTR src

レジスター括退避機能命令
【命令コード】
記載ページ：362

【オペレーション】
{ R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, USP, FPSW, ACC0, ACC1 } = bank(src);

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

【フラグ変化】
- フラグは変化しません。

【命令フォーマット】

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

【記述例】
```
RSTR  #5
RSTR  R1
```

SAVE

レジスタの一括退避
SAVE registers

SAVE

【構文】
SAVE src

レジスター括退避機能命令
【命令コード】
記載ページ：362

【オペレーション】
bank(src) = { R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, USP, FPSW, ACC0, ACC1};

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

【フラグ変化】
- フラグは変化しません。

【命令フォーマット】

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

【記述例】
```
SAVE  #5
SAVE  R1
```

3.4.3 倍精度浮動小数点処理命令

次ページより倍精度浮動小数点処理命令の各命令の詳細説明を示します。

DABS

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

DABS

【構文】
DABS src, dest

倍精度浮動小数点演算命令
【命令コード】
記載ページ：364

【オペレーション】

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

【機能】

- srcに格納された倍精度浮動小数点数の絶対値を算出し、その結果をdestに格納します。
- 非正規化数は、DPSWのDDNビットによらず同じ処理をします。

【フラグ変化】

- フラグは変化しません。

【命令フォーマット】

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

【倍精度浮動小数点例外発生要因】
なし

【記述例】

```
DABS  DR0, DR1
```

【動作補足説明】

- srcの値と演算結果の対応を以下に示します。

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

DADD

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

DADD

【構文】
DADD src, src2, dest

倍精度浮動小数点演算命令
【命令コード】
記載ページ：364

【オペレーション】
dest = src2 + src;

【機能】
- src2に格納された倍精度浮動小数点数と、srcに格納された倍精度浮動小数点数を加算し、その結果をdestに格納します。
- 結果はDPSWのDRM[1:0]ビットに従って丸められます。
- 非正規化数の扱いは、DPSWのDDNビットによって変化します。
- srcとsrc2が反対の符号を持ち、その和が正確に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	—	
S	—	
O	—	
DCV	○	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。
DCO	○	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。
DCZ	○	“0”になります。
DCU	○	アンダフローが発生したとき“1”、それ以外のとき“0”になります。
DCX	○	精度異常が発生したとき“1”、それ以外のとき“0”になります。
DCE	○	非実装処理が発生したとき“1”、それ以外のとき“0”になります。
DFV	○	無効演算が発生したとき“1”、それ以外のときは変化しません。
DFO	○	オーバフローが発生したとき“1”、それ以外のときは変化しません。
DFZ	—	
DFU	○	アンダフローが発生したとき“1”、それ以外のときは変化しません。
DFX	○	精度異常が発生したとき“1”、それ以外のときは変化しません。

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

【命令フォーマット】

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

【倍精度浮動小数点例外発生要因】

非実装処理
無効演算
オーバフロー
アンダフロー
精度異常

【記述例】

```
DADD  DR0, DR1, DR2
```

【動作補足説明】

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

DDN = 0のとき

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

注. –∞方向への丸めモードの場合は–0、それ以外の丸めモードの場合は+0。

DDN = 1のとき

		src						
		正規化数	+0、 +非正規化数	–0、 –非正規化数	+∞	–∞	QNaN	SNaN
src2	正規化数	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. –∞方向への丸めモードの場合は–0、それ以外の丸めモードの場合は+0。

DCMPcm

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

DCMPcm

【構文】
DCMPcm src, src2

倍精度浮動小数点演算命令
【命令コード】
記載ページ：365

【オペレーション】

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

【機能】

- src2に格納された倍精度浮動小数点数と、srcに格納された倍精度浮動小数点数を比較条件cmで比較し、その結果をDCMRレジスタのRESビットに格納します。
- 比較条件cmが成立する場合にRESビットは“1”となり、それ以外の場合は“0”となります。
- 非正規化数の扱いは、DPSWのDDNビットによって変化します。

【比較条件】

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

【フラグ変化】

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

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

【命令フォーマット】

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

【倍精度浮動小数点例外発生要因】

非実装処理

無効演算

【記述例】

DCMPEQ DR0, DR1

【動作補足説明】

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

DDN = 0のとき

		src							
		正規化数	+0	–0	+∞	–∞	非正規化数	QNaN	SNaN
src2	正規化数	比較				>			
	+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）							

DDN = 1のとき

		src						
		正規化数	+0、+非正規化数	–0、–非正規化数	+∞	–∞	QNaN	SNaN
src2	正規化数	比較				>		
	+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）						

DDIV

倍精度浮動小数点除算
Double precision floating-point DIVide

DDIV

【構文】
DDIV src, src2, dest

倍精度浮動小数点演算命令
【命令コード】
記載ページ：365

【オペレーション】
dest = src2 / src;

【機能】
- src2に格納された倍精度浮動小数点数から、srcに格納された倍精度浮動小数点数を除算し、その結果をdestに格納します。
- 結果はDPSWのDRM[1:0]ビットに従って丸められます。
- 非正規化数の扱いは、DPSWのDDNビットによって変化します。

【フラグ変化】

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

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

【命令フォーマット】

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

【倍精度浮動小数点例外発生要因】
非実装処理
無効演算
オーバフロー
アンダフロー
精度異常
ゼロ除算

【記述例】

DDIV DR0, DR1, DR2

【動作補足説明】

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

DDN = 0のとき

		src							
		正規化数	+0	–0	+∞	–∞	非正規化数	QNaN	SNaN
src2	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							

DDN = 1のとき

		src						
		正規化数	+0、+非正規化数	–0、–非正規化数	+∞	–∞	QNaN	SNaN
src2	正規化数	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						

DMOV

倍精度浮動小数点データ転送
Double precision floating-point data MOVe

DMOV

【構文】
DMOV.size src, dest

倍精度浮動小数点転送命令
【命令コード】
記載ページ：366

【オペレーション】
dest = src;

【機能】
- 以下のとおり、srcをdestに転送します。

src	dest	機能
汎用レジスタ	倍精度浮動小数点データレジスタ	レジスタ（src）のデータを倍精度浮動小数点データレジスタ（dest）に転送します。サイズ指定子は.Lと.Dが選択できます。サイズ指定子.Dは倍精度浮動小数点データレジスタ（dest）の上位32ビットを指定する場合のみ選択できます。倍精度浮動小数点データレジスタ（dest）の上位32ビットを指定する場合、サイズ指定子.Dの場合には下位32ビットに0を転送し、サイズ指定子.Lの場合は下位32ビットはそのまま保持されます。倍精度浮動小数点データレジスタ（dest）の下位32ビットを指定する場合には、上位32ビットはそのまま保持されます。
倍精度浮動小数点データレジスタ	汎用レジスタ	倍精度浮動小数点データレジスタ（src）のデータをレジスタ（dest）に転送します。サイズ指定子は.Lのみとなります。倍精度浮動小数点データレジスタ（src）は、上位32ビット/下位32ビットを指定します。
倍精度浮動小数点データレジスタ	倍精度浮動小数点データレジスタ	倍精度浮動小数点データレジスタ（src）のデータを倍精度浮動小数点データレジスタ（dest）に転送します。サイズ指定子は.Dのみとなります。
倍精度浮動小数点データレジスタ	メモリ	倍精度浮動小数点データレジスタのデータをメモリに転送します。サイズ指定子は.Dのみとなります。
メモリ	倍精度浮動小数点データレジスタ	メモリのデータを倍精度浮動小数点データレジスタに転送します。サイズ指定子は.Dのみとなります。
即値	倍精度浮動小数点データレジスタ	命令で指定した即値のデータを倍精度浮動小数点データレジスタ（dest）に転送します。サイズ指定子は.L と.D が選択できます。サイズ指定子.D は倍精度浮動小数点データレジスタ（dest）の上位32 ビットを指定する場合のみ選択できます。倍精度浮動小数点データレジスタ（dest）の上位32 ビットを指定する場合、サイズ指定子.D の場合には下位32 ビットに0 を転送し、サイズ指定子.L の場合は下位32 ビットはそのまま保持されます。倍精度浮動小数点データレジスタ（dest）の下位32 ビットを指定する場合には、上位32 ビットはそのまま保持されます。

【フラグ変化】
- フラグは変化しません。

【命令フォーマット】

構文	size	処理サイズ	対象		コードサイズ（バイト）
			src	dest	
DMOV.size src, dest	レジスタ間転送				
	D	D	Rs	DRHd	4
	L	L	Rs	DRHd	4
	L	L	Rs	DRLd	4
	L	L	DRHs	Rd	4
	L	L	DRLs	Rd	4
	D	D	DRs	DRd	4
	ストア				
	D	D	DRs	[Rd]	4
	D	D	DRs	dsp:8[Rd]（注）	5
	D	D	DRs	dsp:16[Rd]（注）	6
	ロード				
	D	D	[Rs]	DRd	4
	D	D	dsp:8[Rs]（注）	DRd	5
	D	D	dsp:16[Rs]（注）	DRd	6
	即値				
	D	D	#IMM:32	DRHd	7
	L	L	#IMM:32	DRHd	7
	L	L	#IMM:32	DRLd	7

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“.D”のときは4の倍数を指定してください。dsp:8には、サイズ指定子が“.D”のとき0～1020（255 × 4）が指定できます。dsp:16には、サイズ指定子が“.D”のとき0～262140（65535 × 4）が指定できます。命令コードには、1/4した値が埋め込まれます。

【倍精度浮動小数点例外発生要因】

なし

【記述例】

```
DMOV.D  R1, DRH2
DMOV.L  R1, DRH2
DMOV.L  R1, DRL2
DMOV.L  DRH2, R1
DMOV.L  DRL2, R1
DMOV.D  DR0, DR1
DMOV.D  DR1, [R1]
DMOV.D  [R1], DR1
DMOV.D #3FF00000h, DRH2
DMOV.L #4C000000h, DRH2
DMOV.L #00000000h, DRL2
```

DMUL

倍精度浮動小数点乗算
Double precision floating-point MULtiply

DMUL

【構文】
DMUL src, src2, dest

倍精度浮動小数点演算命令
【命令コード】
記載ページ：369

【オペレーション】
dest = src2 * src;

【機能】
- src2に格納された倍精度浮動小数点数と、srcに格納された倍精度浮動小数点数を乗算し、その結果をdestに格納します。
- 結果はDPSWのDRM[1:0]ビットに従って丸められます。
- 非正規化数の扱いは、DPSWのDDNビットによって変化します。

【フラグ変化】

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

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

【命令フォーマット】

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

【倍精度浮動小数点例外発生要因】
非実装処理
無効演算
オーバフロー
アンダフロー
精度異常

【記述例】

DMUL DR0, DR1, DR2

【動作補足説明】

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

DDN = 0のとき

		src							
		正規化数	+0	–0	+∞	–∞	非正規化数	QNaN	SNaN
src2	正規化数	乗算			∞				
	+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							

DDN = 1のとき

		src						
		正規化数	+0、+非正規化数	–0、–非正規化数	+∞	–∞	QNaN	SNaN
src2	正規化数	乗算			∞			
	+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						

DNEG 倍精度浮動小数点符号反転 Double precision floating-point flag NEGate DNEG

【構文】
DNEG src, dest

倍精度浮動小数点演算命令
【命令コード】
記載ページ：369

【オペレーション】
dest = -src;

【機能】
- srcに格納された倍精度浮動小数点数の符号を反転し、その結果をdestに格納します。
- 非正規化数は、DPSWのDDNビットによらず同じ処理をします。

【フラグ変化】
- フラグは変化しません。

【命令フォーマット】

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

【倍精度浮動小数点例外発生要因】
なし

【記述例】
DNEG DR0, DR1

【動作補足説明】
- srcの値と演算結果の対応を以下に示します。

	src								
	+正規化数	−正規化数	+0	−0	+∞	−∞	非正規化数	QNaN	SNaN
結果	−正規化数	+正規化数	−0	+0	−∞	+∞	符号反転した非正規化数	符号反転したQNaN	符号反転したSNaN

DPOPM

複数の倍精度浮動小数点レジスタの復帰
Double precision POP Multiple register

DPOPM

【構文】
DPOPM.size dest-dest2

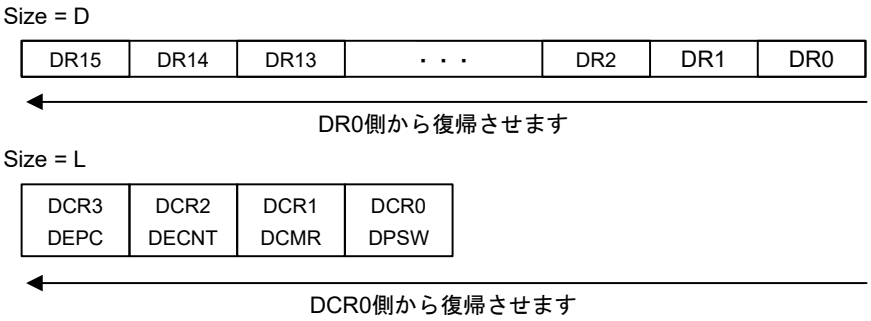
倍精度浮動小数点転送命令
【命令コード】
記載ページ：370

【オペレーション】

```
signed char i;
for ( i = register_num(dest); i <= register_num(dest2); i++ ) {
  tmp = *SP;
  SP = SP + (( size == D ) ? 8 : 4);
  If ( size == D ) {
     DR(i) = tmp;
  } else {
     DCR(i) = tmp;
  }
}
```

【機能】

- destとdest2で範囲指定した倍精度浮動小数点レジスタを一括してスタックから復帰させます。
- 範囲は、destとdest2で指定します。ただし、（destのレジスタ番号≦dest2のレジスタ番号）となっている必要があります。
- 使用されるスタックポインタは、PSWのUビットで示すスタックポインタになります。
- size＝Dのときは転送対象が倍精度浮動小数点データレジスタになります。
- size＝Lのときは転送対象が倍精度浮動小数点制御レジスタになります。
- スタックから復帰させる順序は以下のとおりです。



【フラグ変化】

- フラグ変化はありません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ（バイト）
		dest	dest2	
DPOPM.size dest-dest2	D	DRd	DRd2	3
	L	DCRd	DCRd2	3

【倍精度浮動小数点例外発生要因】
なし

【記述例】
```
DPOPM.D  DR0-DR15
DPOPM.L  DPSW-DCMR
DPOPM.D  DR1-DR1
```

DPUSHM

複数の倍精度浮動小数点レジスタの退避
Double precision PUSH Multiple register

DPUSHM

【構文】
DPUSHM.size src-src2

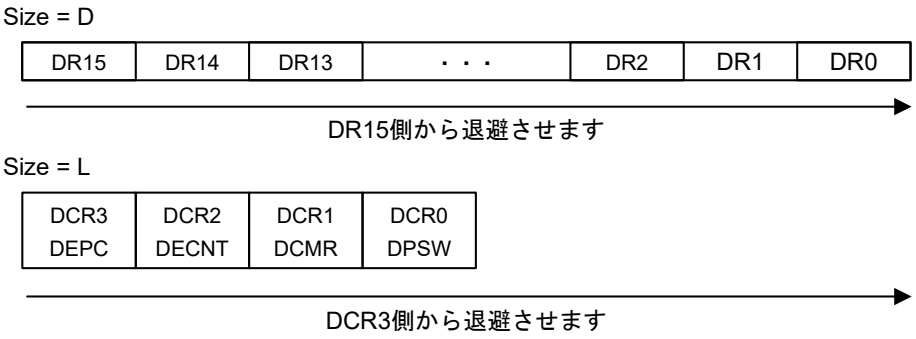
倍精度浮動小数点転送命令
【命令コード】
記載ページ：371

【オペレーション】

```
signed char i;
for ( i = register_num(src2); i >= register_num(src); i-- ) {
  tmp = ( size == D ) ? DR(i) : DCR(i);
  SP = SP - (( size == D ) ? 8 : 4);
  *SP = tmp;
}
```

【機能】

- srcとsrc2で範囲指定した倍精度浮動小数点レジスタを一括してスタックに退避させます。
- 範囲は、srcとsrc2で指定します。ただし、（srcのレジスタ番号≦src2のレジスタ番号）となっている必要があります。
- 使用されるスタックポインタは、PSWのUビットで示すスタックポインタになります。
- size = Dのときは転送対象が倍精度浮動小数点データレジスタになります。
- size = Lのときは転送対象が倍精度浮動小数点制御レジスタになります。
- スタックに退避させる順序は以下のとおりです。



【フラグ変化】

- フラグ変化はありません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ（バイト）
		src	src2	
DPUSHM.size src-src2	D	DRs	DRs2	3
	L	DCRs	DCRs2	3

【倍精度浮動小数点例外発生要因】
なし

【記述例】

```
DPUSHM.D  DR0-DR15
DPUSHM.L  DPSW-DCMR
DPUSHM.D  DR1-DR1
```

DROUND

倍精度浮動小数点数→符号付き整数変換
Double precision ROUND to signed integer

DROUND

【構文】
DROUND src, dest

倍精度浮動小数点演算命令
【命令コード】
記載ページ：372

【オペレーション】
dest = (signed long) src;

【機能】
- srcに格納された倍精度浮動小数点数を符号付きロングワード（32ビット）整数に変換し、その結果をdestの下位32ビットに格納します。
- 結果はDPSWのDRM[1:0]ビットに従って丸められます。
- destの[63:32]ビットは不定になります。

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

【フラグ変化】

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

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

【命令フォーマット】

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

【倍精度浮動小数点例外発生要因】

非実装処理
無効演算
精度異常

【記述例】

```
DROUND  DR0, DR1
```

【動作補足説明】

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

DDN = 0のとき

srcの値（指数部はゲタなしの値）		dest (注1)	例外
src≧0	+∞	7FFFFFFFh	無効演算
	31≦指数部≦1023		
	–1022≦指数部≦30	00000000h～7FFFFFFFh	なし (注2、注3)
	+非正規化数	変化なし	非実装処理
	+0	00000000h	なし
src < 0	–0		
	–非正規化数	変化なし	非実装処理
	–1022≦指数部≦30	00000000h～80000000h	なし (注2)
	31≦指数部≦1023	80000000h (注4)	無効演算 (注2、注5)
	–∞		
NaN	QNaN	符号ビット= 0：7FFFFFFFh 符号ビット= 1：80000000h	無効演算
	SNaN		

注1. 倍精度浮動小数点例外が発生し、対応するDEj（j = X, U, Z, O, V）が“1”の場合には変化しません。
注2. 無効演算が発生しないとき、丸め発生時には精度異常が発生します。
注3. 丸め後の値が7FFFFFFFhを超える場合は、無効演算が発生します。
注4. 丸め後の値によらずdestは80000000hとなります。
注5. 丸め後の値が80000000hとなる場合は、無効演算は発生しません。

DDN = 1のとき

srcの値（指数部はゲタなしの値）		dest (注1)	例外
src≧0	+∞	7FFFFFFFh	無効演算
	31≦指数部≦1023		
	–1022≦指数部≦30	00000000h～7FFFFFFFh	なし (注2、注3)
	+非正規化数	00000000h	なし
	+0		
src < 0	–0		
	–非正規化数		
	–1022≦指数部≦30	00000000h～80000000h	なし (注2)
	31≦指数部≦1023	80000000h (注4)	無効演算 (注2、注5)
	–∞		
NaN	QNaN	符号ビット= 0：7FFFFFFFh 符号ビット= 1：80000000h	無効演算
	SNaN		

注1. 倍精度浮動小数点例外が発生し、対応するDEj（j = X, U, Z, O, V）が“1”の場合には変化しません。
注2. 無効演算が発生しないとき、丸め発生時には精度異常が発生します。
注3. 丸め後の値が7FFFFFFFhを超える場合は、無効演算が発生します。
注4. 丸め後の値によらずdestは80000000hとなります。
注5. 丸め後の値が80000000hとなる場合は、無効演算は発生しません。

DSQRT

倍精度浮動小数点平方根
Double precision floating-point SQuare RooT

DSQRT

倍精度浮動小数点演算命令

【構文】
DSQRT src, dest

【命令コード】
記載ページ：372

【オペレーション】
dest = sqrt (src);

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

【フラグ変化】

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

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

【命令フォーマット】

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

【倍精度浮動小数点例外発生要因】
非実装処理
無効演算
精度異常

【記述例】
DSQRT DR0, DR1

【動作補足説明】

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

DDN = 0のとき

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

DDN = 1のとき

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

【無効演算発生時のQNaN生成規則】

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

DSUB

倍精度浮動小数点減算
Double precision floating-point SUBtract

DSUB

【構文】
DSUB src, src2, dest

倍精度浮動小数点演算命令
【命令コード】
記載ページ：372

【オペレーション】
dest = src2 - src;

【機能】
- src2に格納された倍精度浮動小数点数から、srcに格納された倍精度浮動小数点数を減算し、その結果をdestに格納します。
- 結果はDPSWのDRM[1:0]ビットに従って丸められます。
- 非正規化数の扱いは、DPSWのDDNビットによって変化します。
- srcとsrc2が同一の符号を持ち、その差が正確に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	—	
S	—	
O	—	
DCV	○	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。
DCO	○	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。
DCZ	○	“0”になります。
DCU	○	アンダフローが発生したとき“1”、それ以外のとき“0”になります。
DCX	○	精度異常が発生したとき“1”、それ以外のとき“0”になります。
DCE	○	非実装処理が発生したとき“1”、それ以外のとき“0”になります。
DFV	○	無効演算が発生したとき“1”、それ以外のときは変化しません。
DFO	○	オーバフローが発生したとき“1”、それ以外のときは変化しません。
DFZ	—	
DFU	○	アンダフローが発生したとき“1”、それ以外のときは変化しません。
DFX	○	精度異常が発生したとき“1”、それ以外のときは変化しません。

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

【命令フォーマット】

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

【倍精度浮動小数点例外発生要因】
非実装処理
無効演算
オーバフロー
アンダフロー
精度異常

【記述例】
DSUB DR0, DR1, DR2

【動作補足説明】

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

DDN = 0のとき

		src							
		正規化数	+0	–0	+∞	–∞	非正規化数	QNaN	SNaN
src2	正規化数	減算				+∞			
	+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							

注. –∞方向への丸めモードの場合は–0、それ以外の丸めモードの場合は+0。

DDN = 1のとき

		src						
		正規化数	+0、+非正規化数	–0、–非正規化数	+∞	–∞	QNaN	SNaN
src2	正規化数	減算				+∞		
	+0、+非正規化数		(注)	+0	–∞			
	–0、–非正規化数		–0	(注)				
	+∞	+∞			無効演算			
	–∞	–∞				無効演算		
	QNaN	QNaN						
	SNaN	無効演算						

注. –∞方向への丸めモードの場合は–0、それ以外の丸めモードの場合は+0。

DTOF

倍精度浮動小数点数→単精度浮動小数点数変換
Double TO Float

DTOF

【構文】
DTOF src, dest

倍精度浮動小数点演算命令
【命令コード】
記載ページ：373

【オペレーション】
dest = (float) src;

【機能】
- srcに格納された倍精度浮動小数点数を単精度浮動小数点数に変換し、その結果をdestの下位32ビットに格納します。
- 結果はDPSWのDRM[1:0]ビットに従って丸められます。
- destの[63:32]ビットは不定になります。

【フラグ変化】

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

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

【命令フォーマット】

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

【倍精度浮動小数点例外発生要因】
非実装処理
無効演算
オーバフロー
アンダフロー
精度異常

【記述例】

```
DTOF  DR0, DR1
```

【動作補足説明】

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

DDN = 0のとき

srcの値（指数部はゲタなしの値）		dest (注1)	例外
src≧0	+∞	+∞	なし
	128≦指数部≦1023		オーバフロー
	–126≦指数部≦127	変換結果	なし (注2、注3)
	–1022≦指数部≦–127	変化なし	非実装処理 アンダフロー
	+非正規化数	変化なし	非実装処理
	+0	+0	なし
src < 0	–0	–0	
	–非正規化数	変化なし	非実装処理
	–1022≦指数部≦–127	変化なし	非実装処理 アンダフロー
	–126≦指数部≦127	変換結果	なし (注2、注3)
	128≦指数部≦1023	–∞	オーバフロー
	–∞		なし
NaN	QNaN	QNaN (注4)	なし
	SNaN	SNaN (注5)	無効演算

注1. 倍精度浮動小数点例外が発生し、対応するDEj（j = X, U, Z, O, V）が“1”の場合には変化しません。
注2. 丸め発生時は、精度異常が発生します。
注3. 丸め後の値の指数部が128となる場合には、オーバフローとなります。
注4. destは入力のQNaNと同符号で、入力値の[51:29]ビットを仮数部にもつQNaNとなります。
注5. destは入力のSNaNと同符号で、入力値をQNaN化した値の[51:29]ビットを仮数部にもつQNaNとなります。

DDN = 1のとき

srcの値（指数部はゲタなしの値）		dest (注1)	例外
src≧0	+∞	+∞	なし
	128≦指数部≦1023		オーバフロー
	–126≦指数部≦127	変換結果	なし (注2、注3)
	–1022≦指数部≦–127	+0	アンダフロー
	+非正規化数	+0	なし
	+0	+0	なし
src < 0	–0	–0	
	–非正規化数	–0	なし
	–1022≦指数部≦–127	–0	アンダフロー
	–126≦指数部≦127	変換結果	なし (注2、注3)
	128≦指数部≦1023	–∞	オーバフロー
	–∞		なし
NaN	QNaN	QNaN (注4)	なし
	SNaN	SNaN (注5)	無効演算

注1. 倍精度浮動小数点例外が発生し、対応するDEj（j = X, U, Z, O, V）が“1”の場合には変化しません。
注2. 丸め発生時は、精度異常が発生します。
注3. 丸め後の値の指数部が128となる場合には、オーバフローとなります。
注4. destは入力のQNaNと同符号で、入力値の[51:29]ビットを仮数部にもつQNaNとなります。
注5. destは入力のSNaNと同符号で、入力値をQNaN化した値の[51:29]ビットを仮数部にもつQNaNとなります。

DTOI

倍精度浮動小数点数→符号付き整数変換
Double TO signed Intereger

DTOI

【構文】
DTOI src, dest

倍精度浮動小数点演算命令
【命令コード】
記載ページ：373

【オペレーション】
dest = (long) src;

【機能】
- srcに格納された倍精度浮動小数点数を符号付きロングワード（32ビット）整数に変換し、その結果をdestの下位32ビットに格納します。
- 結果はDPSWのDRM[1:0]ビットに関係なく0方向に丸められます。
- destの[63:32]ビットは不定になります。

【フラグ変化】

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

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

【命令フォーマット】

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

【倍精度浮動小数点例外発生要因】
非実装処理
無効演算
精度異常

【記述例】
DTOI DR0, DR1

【動作補足説明】

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

DDN = 0のとき

srcの値（指数部はゲタなしの値）		dest (注1)	例外
src≧0	+∞	7FFFFFFFh	無効演算
	31≦指数部≦1023		
	–1022≦指数部≦30	00000000h～7FFFFFFFh	なし (注2、注3)
	+非正規化数	変化なし	非実装処理
	+0	00000000h	なし
src < 0	–0		
	–非正規化数	変化なし	非実装処理
	–1022≦指数部≦30	00000000h～80000000h	なし (注2)
	31≦指数部≦1023	80000000h (注4)	無効演算 (注2、注5)
	–∞		
NaN	QNaN	符号ビット= 0：7FFFFFFFh	無効演算
	SNaN	符号ビット= 1：80000000h	

注1. 倍精度浮動小数点例外が発生し、対応するDEj（j = X, U, Z, O, V）が“1”の場合には変化しません。
注2. 無効演算が発生しないとき、丸め発生時には精度異常が発生します。
注3. 丸め後の値が7FFFFFFFhを超える場合には、無効演算が発生します。
注4. 丸め後の値によらずdestは80000000hとなります。
注5. 丸め後の値が80000000hとなる場合には、無効演算が発生しません。

DDN = 1のとき

srcの値（指数部はゲタなしの値）		dest (注1)	例外
src≧0	+∞	7FFFFFFFh	無効演算
	31≦指数部≦1023		
	–1022≦指数部≦30	00000000h～7FFFFFFFh	なし (注2、注3)
	+非正規化数	00000000h	なし
	+0		
src < 0	–0		
	–非正規化数		
	–1022≦指数部≦30	00000000h～80000000h	なし (注2)
	31≦指数部≦1023	80000000h (注4)	無効演算 (注2、注5)
	–∞		
NaN	QNaN	符号ビット= 0：7FFFFFFFh	無効演算
	SNaN	符号ビット= 1：80000000h	

注1. 倍精度浮動小数点例外が発生し、対応するDEj（j = X, U, Z, O, V）が“1”の場合には変化しません。
注2. 無効演算が発生しないとき、丸め発生時には精度異常が発生します。
注3. 丸め後の値が7FFFFFFFhを超える場合は、無効演算が発生します。
注4. 丸め後の値によらずdestは80000000hとなります。
注5. 丸め後の値が80000000hとなる場合には、無効演算が発生しません。

DTOU

倍精度浮動小数点数→符号なし整数変換
Double TO Unsigned intereger

DTOU

【構文】
DTOU src, dest

倍精度浮動小数点演算命令
【命令コード】
記載ページ：373

【オペレーション】
dest = (unsigned long) src;

【機能】
- srcに格納された倍精度浮動小数点数を符号なしロングワード（32ビット）整数に変換し、その結果をdestの下位32ビットに格納します。
- 結果はDPSWのDRM[1:0]ビットに関係なく0方向に丸められます。
- destの[63:32]ビットは不定になります。

【フラグ変化】

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

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

【命令フォーマット】

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

【倍精度浮動小数点例外発生要因】
非実装処理
無効演算
精度異常

【記述例】
DTOU DR0, DR1

【動作補足説明】

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

DDN = 0のとき

srcの値（指数部はゲタなしの値）		dest (注1)	例外
src≧0	+∞	FFFFFFFFh	無効演算
	32≦指数部≦1023		
	–1022≦指数部≦31	00000000h～FFFFFFFFh	なし (注2、注3)
	+非正規化数	変化なし	非実装処理
	+0	00000000h	なし
src < 0	–0		
	–非正規化数	変化なし	非実装処理
	–正規化数、–∞	00000000h	無効演算
NaN	QNaN	最上位ビット= 0:FFFFFFFFh 最上位ビット= 1:00000000h	無効演算
	SNaN		

注1. 倍精度浮動小数点例外が発生し、対応するDEj（j = X, U, Z, O, V）が“1”の場合には変化しません。
注2. 丸め発生時は、精度異常が発生します。
注3. 丸め後の値がFFFFFFFFhを超える場合は、無効演算が発生します。

DDN = 1のとき

srcの値（指数部はゲタなしの値）		dest (注1)	例外
src≧0	+∞	FFFFFFFFh	無効演算
	32≦指数部≦1023		
	–1022≦指数部≦31	00000000h～FFFFFFFFh	なし (注2、注3)
	+0、+非正規化数	00000000h	なし
src < 0	–0、–非正規化数		
	–正規化数、–∞	00000000h	無効演算
NaN	QNaN	符号ビット= 0：FFFFFFFFh 符号ビット= 1：00000000h	無効演算
	SNaN		

注1. 倍精度浮動小数点例外が発生し、対応するDEj（j = X, U, Z, O, V）が“1”の場合には変化しません。
注2. 丸め発生時は、精度異常が発生します。
注3. 丸め後の値がFFFFFFFFhを超える場合は、無効演算が発生します。

FTOD

単精度浮動小数点数→倍精度浮動小数点数変換
Float TO Double

FTOD

【構文】
FTOD src, dest

倍精度浮動小数点演算命令
【命令コード】
記載ページ：374

【オペレーション】
dest = (double) src;

【機能】
- srcに格納された単精度浮動小数点数を倍精度浮動小数点数に変換し、その結果をdestに格納します。

【フラグ変化】

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

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

【命令フォーマット】

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

【倍精度浮動小数点例外発生要因】
非実装処理
無効演算

【記述例】
FTOD R1, DR1

【動作補足説明】

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

DDN = 0のとき

srcの値（指数部はゲタなしの値）		dest (注1)	例外
src≧0	+∞	+∞	なし
	–126≦指数部≦127	変換結果	なし
	+非正規化数	変化なし	非実装処理
	+0	+0	なし
src < 0	–0	–0	
	–非正規化数	変化なし	非実装処理
	–126≦指数部≦127	変換結果	なし
	–∞	–∞	なし
NaN	QNaN	QNaN (注2)	なし
	SNaN	SNaN (注3)	無効演算

注1. 倍精度浮動小数点例外が発生し、対応するDEj（j = X, U, Z, O, V）が“1”の場合には変化しません。
注2. destは入力のQNaNと同符号で、入力値の[22:0]ビットを仮数部の[51:29]にもち、仮数部の[28:0]が0のQNaNとなります。
注3. destは入力のSNaNと同符号で、入力値をQNaN化した値の[22:0]ビットを仮数部の[51:29]にもち、仮数部の[28:0]が0のQNaNとなります。

DDN = 1のとき

srcの値（指数部はゲタなしの値）		dest (注1)	例外
src≧0	+∞	+∞	なし
	–126≦指数部≦127	変換結果	なし
	+非正規化数	+0	なし
	+0	+0	なし
src < 0	–0	–0	
	–非正規化数	–0	なし
	–126≦指数部≦127	変換結果	なし
	–∞	–∞	なし
NaN	QNaN	QNaN (注2)	なし
	SNaN	SNaN (注3)	無効演算

注1. 倍精度浮動小数点例外が発生し、対応するDEj（j = X, U, Z, O, V）が“1”の場合には変化しません。
注2. destは入力のQNaNと同符号で、入力値の[22:0]ビットを仮数部の[51:29]にもち、仮数部の[28:0]が“0”のQNaNとなります。
注3. destは入力のSNaNと同符号で、入力値をQNaN化した値の[22:0]ビットを仮数部の[51:29]にもち、仮数部の[28:0]が“0”のQNaNとなります。

ITOD

符号付き整数→倍精度浮動小数点数変換
signed Integer TO Double

ITOD

倍精度浮動小数点演算命令

【構文】
ITOD src, dest

【命令コード】
記載ページ：374

【オペレーション】
dest = (double) src;

【機能】
- srcに格納された符号付きロングワード（32ビット）整数を倍精度浮動小数点数に変換し、その結果をdestに格納します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	—	
O	—	
DCV	○	“0”になります。
DCO	○	“0”になります。
DCZ	○	“0”になります。
DCU	○	“0”になります。
DCX	○	“0”になります。
DCE	○	“0”になります。
DFV	—	
DFO	—	
DFZ	—	
DFU	—	
DFX	—	

【命令フォーマット】

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

【倍精度浮動小数点例外発生要因】
なし

【記述例】
ITOD R1, DR0

MVFDC

倍精度浮動小数点制御レジスタからの転送
MoVe From Double precision floating-point Control register

MVFDC

倍精度浮動小数点転送命令
【命令コード】
記載ページ：375

【構文】
MVFDC src, dest

【オペレーション】
dest = src;

【機能】
- srcをdestに転送します。

【フラグ変化】
- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ（バイト）
		src	dest	
MVFDC src, dest	L	DCRs	Rd	4

注． 選択可能な src：DPSW, DCMR, DECNT, DEPC

【倍精度浮動小数点例外発生要因】
なし

【記述例】
MVFDC DPSW, R2

MVFDR

倍精度浮動小数点比較結果レジスタからの転送
MoVe From Double precision floating-point compare Result

MVFDR

倍精度浮動小数点転送命令
【命令コード】
記載ページ：375

【構文】
MVFDR

【オペレーション】
Z = DCMR.RES;

【機能】
- DCMRのRESビットをPWSのZフラグに転送します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	○	DCMR.RESの内容が書き込まれます。
S	—	
O	—	

注. DPSWレジスタは変化しません。

【命令フォーマット】

構文	コードサイズ（バイト）
MVFDR	3

【倍精度浮動小数点例外発生要因】
なし

【記述例】
MVFDR

MVTDC

倍精度浮動小数点制御レジスタへの転送
MoVe To Double precision floating-point Control register

MVTDC

【構文】
MVTDC src, dest

倍精度浮動小数点転送命令
【命令コード】
記載ページ：376

【オペレーション】
dest = src;

【機能】
- srcをdestに転送します。

【フラグ変化】
- フラグは変化しません。

【命令フォーマット】

構文	処理サイズ	対象		コードサイズ（バイト）
		src	dest	
MVTDC src, dest	L	Rs	DCRd	4

注. 選択可能なdest：DPSW, DCMR, DECNT, DEPC

【倍精度浮動小数点例外発生要因】
なし

【記述例】
MVTDC R1,DPSW

UTOD

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

UTOD

【構文】
UTOD src, dest

倍精度浮動小数点演算命令
【命令コード】
記載ページ：376

【オペレーション】
dest = (double) src;

【機能】
- srcに格納された符号なしロングワード（32ビット）整数を倍精度浮動小数点数に変換し、その結果をdestに格納します。

【フラグ変化】

フラグ	変化	条件
C	—	
Z	—	
S	—	
O	—	
DCV	○	“0”になります。
DCO	○	“0”になります。
DCZ	○	“0”になります。
DCU	○	“0”になります。
DCX	○	“0”になります。
DCE	○	“0”になります。
DFV	—	
DFO	—	
DFZ	—	
DFU	—	
DFX	—	

【命令フォーマット】

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

【倍精度浮動小数点例外発生要因】
なし

【記述例】
UTOD R1, DR0

3.4.4 Protect Zone 命令

次ページより Protect Zone命令の各命令の詳細説明を示します。

TRCALL

Root ワールドコール
Transition Root world CALL

TRCALL

Protect Zone 命令
【命令コード】
記載ページ：378

【構文】
TRCALL　src

【オペレーション】
```
WID.BWI = PSW.I;
WID.BWID = WID.WID;
WID.WID = 0;
tmp0 = PSW;
I = 0
U=0;
tmp1 = PC + 4;
PC = *(TRCTB+src*4);
ISP_TR = ISP_TR - 4;
*ISP_TR = tmp0;
ISP_TR = ISP_TR - 4;
*ISP_TR = tmp1;
```

【機能】
- srcで指定した番号の無条件トラップ（ワールド遷移受付）が発生します。
- srcの範囲は、0≦src≦255です。
- Rootワールドのスーパバイザモードに移行し、WIDレジスタのWIDビットが“00b”になります。
- 命令実行前のワールドIDはWIDレジスタのBWIDビットに保持されます。
- 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。

【フラグ変化】
- フラグは変化しません。
- 命令実行前のPSWはRootワールドの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（バイト）
	src	
TRCALL src	#IMM:8	4

【記述例】
```
TRCALL  #0
```

TRMVFC

Root ワールドでの制御レジスタからの転送
Transition Root world MoVe From Control register

TRMVFC

【構文】
TRMVFC src, dest

Protect Zone 命令
【命令コード】
記載ページ：378

【オペレーション】
dest = src;

【機能】
- srcをdestに転送します。
- srcにRoot用、Secure用、Non-secure用が存在するレジスタを指定した場合は、WIDレジスタのBWIDビットが示すワールド用レジスタから転送します。
  WID.BWID==2：**_NS
  WID.BWID==1：**_S
  WID.BWID==0：**_TR
- この命令はRootワールド専用命令です。その他のワールドで実行した場合は、未定義命令例外が発生します。
- この命令は特権命令です。Rootワールドで、ユーザモードで実行すると特権命令例外が発生します。

【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	
TRMVFC src, dest	L	Rx	Rd	3

注． 選択可能なsrc：ISP, INTB, BPC, BPSW, EXTB, IVC, TRCTB, WINTB

【記述例】
```
TRMVFC  ISP, R1
TRMVFC  TRCTB, R1
```

TRMVTC

Rootワールドでの制御レジスタへの転送
Transition Root world MoVe To Control register

TRMVTC

【構文】
TRMVTC src, dest

Protect Zone命令
【命令コード】
記載ページ：379

【オペレーション】
dest = src;

【機能】
- srcをdestに転送します。
- destにRoot用、Secure用、Non-secure用が存在するレジスタを指定した場合は、WIDレジスタのBWIDビットが示すワールド用レジスタに転送します。
  WID.BWID==2：**_NS
  WID.BWID==1：**_S
  WID.BWID==0：**_TR
- BPSW_NSのb28に“1”を書き込んでも“0”を書き込んでも、“0”が書き込まれます。
- この命令はRootワールド専用命令です。その他のワールドで実行した場合は、未定義命令例外が発生します。
- この命令は特権命令です。Rootワールドで、ユーザモードで実行すると特権命令例外が発生します。

【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	
TRMVTC 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, INTB, BPC, BPSW, EXTB, IVC, TRCTB, WINTB

【記述例】
```
TRMVTC  #0FFFF0000h, TRCTB
TRMVTC  R1, ISP
TRMVTC  R1,WINTB
```

TRRTE

Root ワールドの例外処理からの復帰
Transition Root world ReTurn from Exception

TRRTE

Protect Zone 命令
【命令コード】
記載ページ：381

【構文】
TRRTE

【オペレーション】

```
PC = *SP_TR;
SP_TR = SP_TR + 4;
tmp1 = *SP_TR;
SP_TR = SP_TR + 4;
if ( WID.BWID == 2 ) {
 tmp1 = 0effffffh & tmp1;
}
PSW = tmp1;
WID.WID = WID.BWID;
WID.BWID = 0;
WID.BITW = 0;
WID.BWI = 0;
LI = 0;
```

注 . SP_TR : PSW.U==0 の場合 ISP_TR、PSW.U==1 の場合 USP

【機能】

- 例外が受け付けられたときに退避したPCとPSWを復帰させ、RootワールドからSecureワールドもしくはNon-secureワールドに遷移します。
- この命令はRootワールド専用命令です。その他のワールドで実行した場合は、未定義命令例外が発生します。
- この命令は特権命令です。Rootワールドで、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には、PSWのUビットは“1”になります。

【フラグ変化】

- C、Z、S、OはRootのスタック上の値になります。

【命令フォーマット】

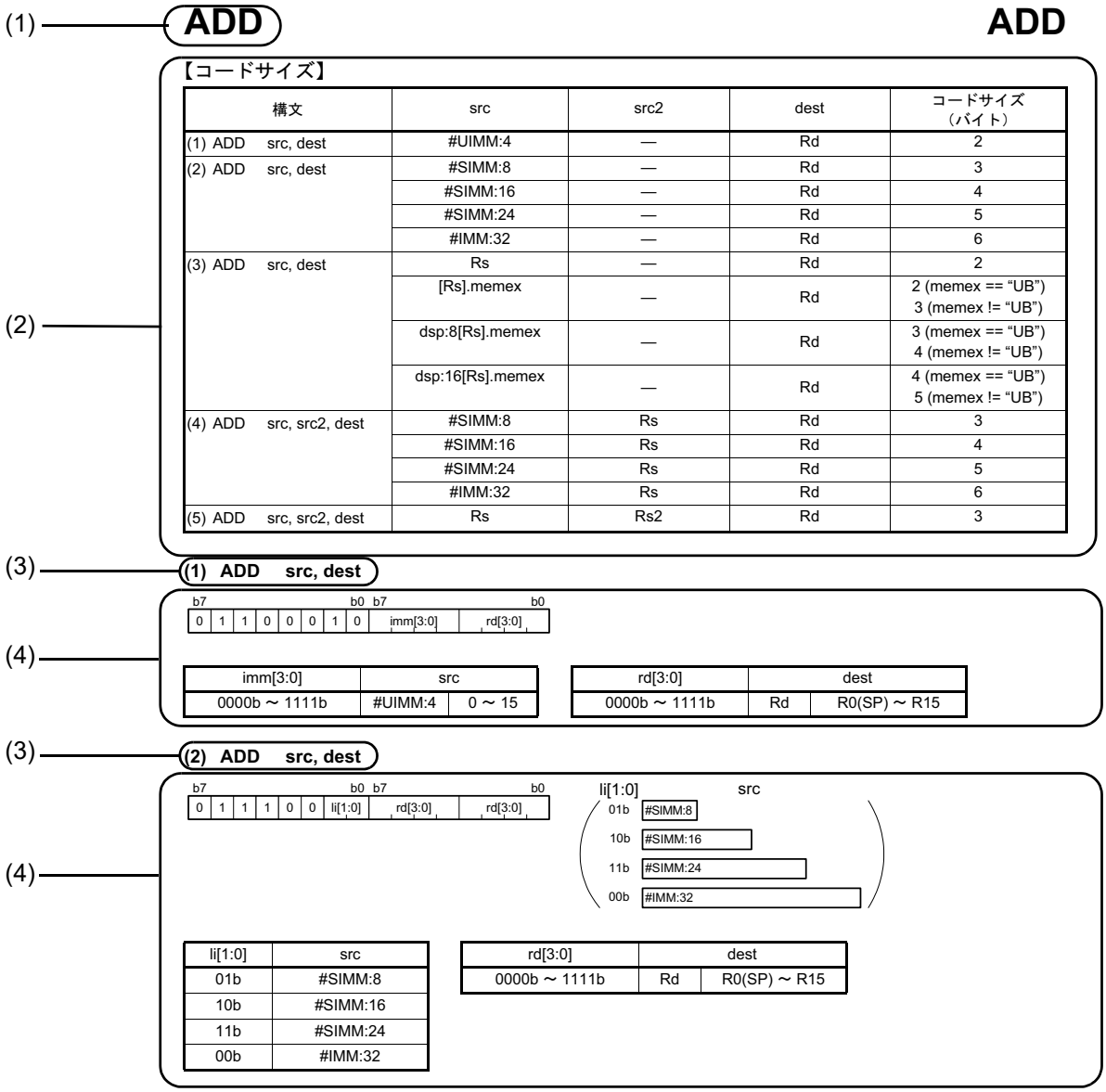
構文	コードサイズ（バイト）
TRRTE	2

【記述例】
TRRTE

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。



構文	src	src2	dest	コードサイズ（バイト）
(1) ADD src, dest	#UIMM:4	—	Rd	2
(2) ADD src, dest	#SIMM:8	—	Rd	3
	#SIMM:16	—	Rd	4
	#SIMM:24	—	Rd	5
	#IMM:32	—	Rd	6
(3) 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”)
(4) ADD src, src2, dest	#SIMM:8	Rs	Rd	3
	#SIMM:16	Rs	Rd	4
	#SIMM:24	Rs	Rd	5
	#IMM:32	Rs	Rd	6
(5) ADD src, src2, dest	Rs	Rs2	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。

(2) コードサイズ表

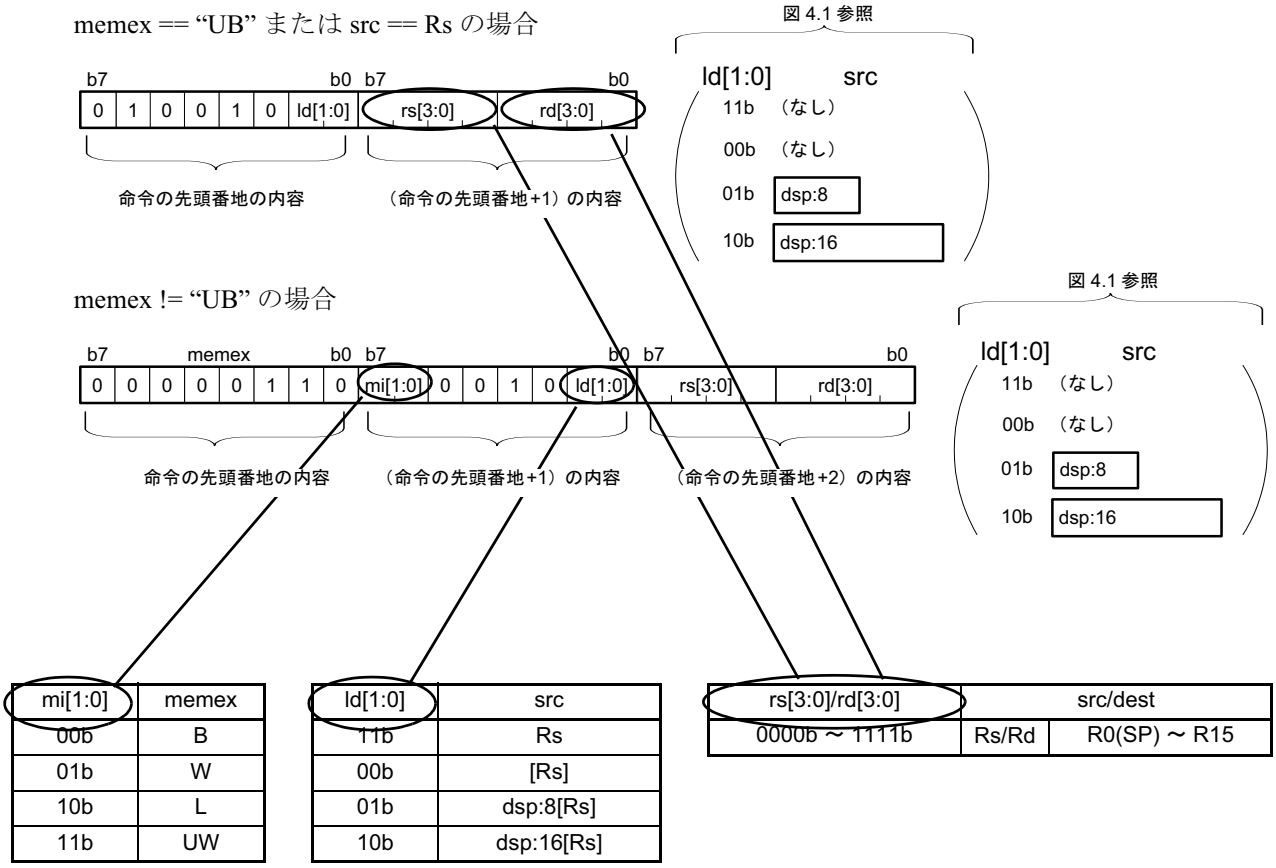
命令のバイト数を示しています。RXv3e CPUの命令のバイト数は、1バイト～8バイトです。

(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。

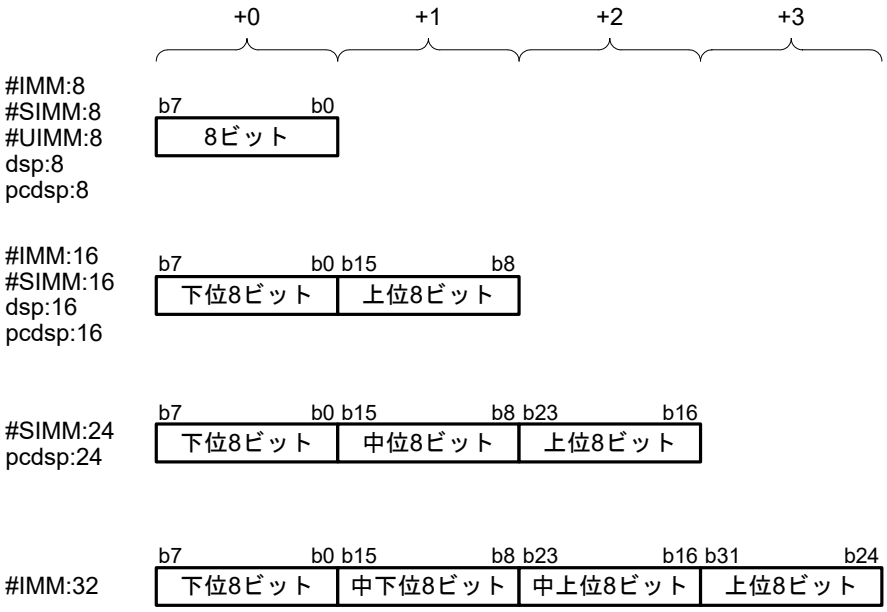


図 4.1　即値（IMM）とディスプレースメント（dsp）

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
nm : Number of registers
cm : Compare condition

4.2 命令コード詳細説明

RXv3e 各命令の命令コードの詳細説明を示します。

4.2.1 標準搭載命令

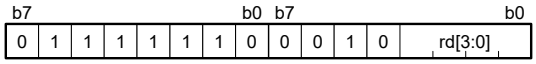
次ページより標準搭載命令の命令コードの詳細説明を示します。

ABS ABS

【コードサイズ】

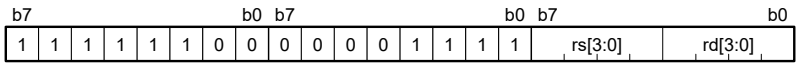
構文	src	dest	コードサイズ（バイト）
(1) ABS dest	—	Rd	2
(2) ABS src, dest	Rs	Rd	3

(1) ABS dest



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

(2) ABS src, dest



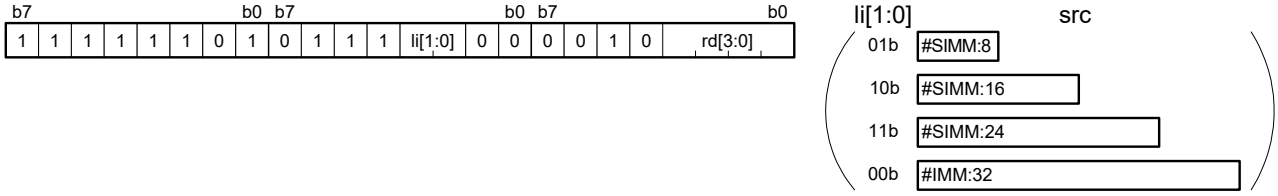
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

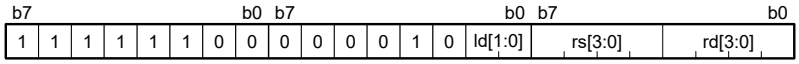
(1) ADC src, dest



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

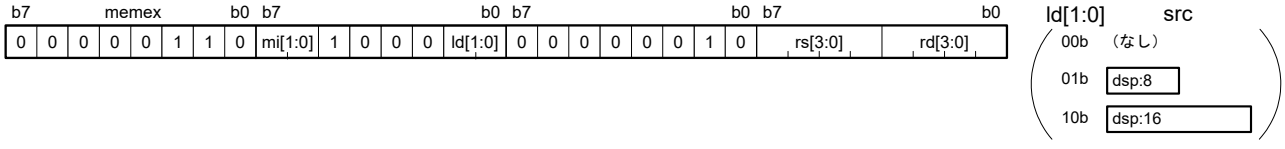
(2) ADC src, dest



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]

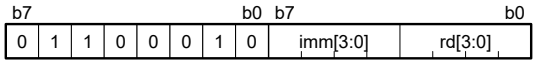
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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
(2) ADD src, dest	#SIMM:8	—	Rd	3
	#SIMM:16	—	Rd	4
	#SIMM:24	—	Rd	5
	#IMM:32	—	Rd	6
(3) 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”)
(4) ADD src, src2, dest	#SIMM:8	Rs	Rd	3
	#SIMM:16	Rs	Rd	4
	#SIMM:24	Rs	Rd	5
	#IMM:32	Rs	Rd	6
(5) ADD src, src2, dest	Rs	Rs2	Rd	3

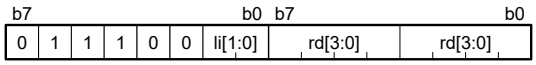
(1) ADD src, dest

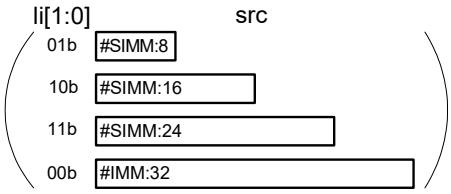


imm[3:0]	src	
0000b ～ 1111b	#UIMM:4	0 ～ 15

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

(2) ADD src, dest





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

注 . (4) の src2 を dest に一致させたコードと等価です。

(3) ADD src, dest

memex == “UB” または src == Rs の場合



memex != “UB” の場合

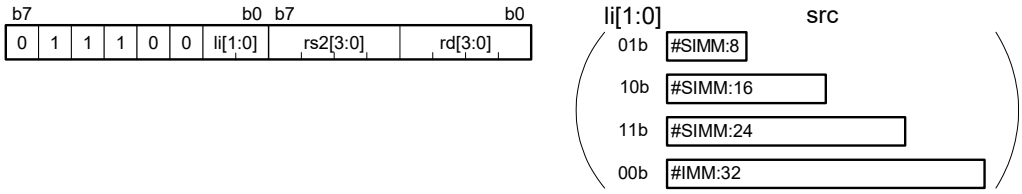


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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

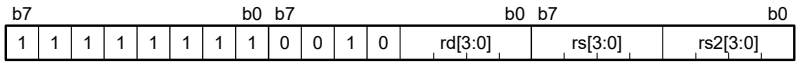
(4) ADD src, src2, dest



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

(5) ADD src, src2, dest



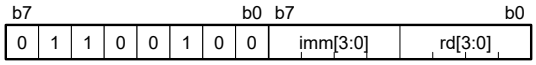
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ～ 1111b	Rs/Rs2/Rd	R0(SP) ～ R15

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

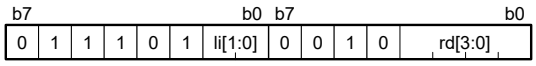
(1) AND src, dest

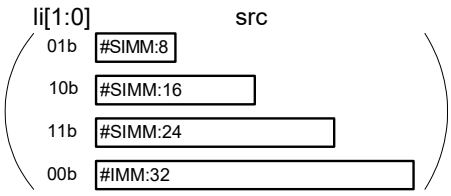


imm[3:0]	src	
0000b ～ 1111b	#UIMM:4	0 ～ 15

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

(2) AND src, dest



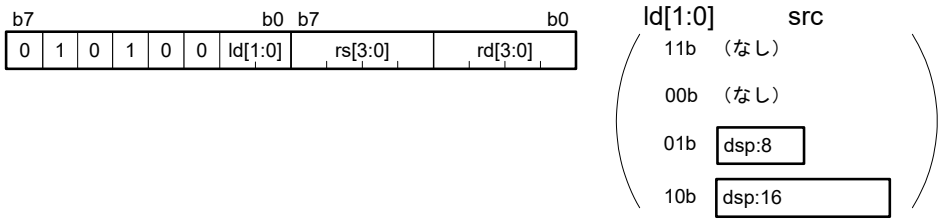


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

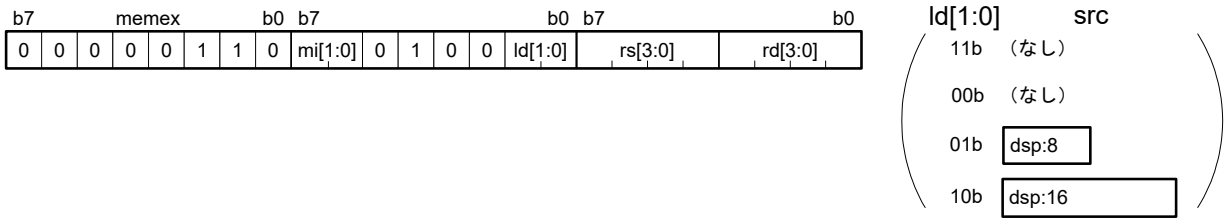
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(3) AND src, dest

memex == “UB” または src == Rs の場合



memex != “UB” の場合

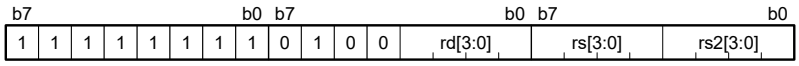


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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



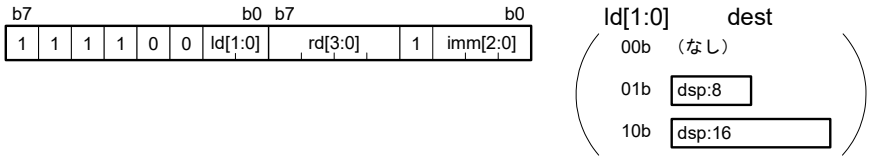
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ～ 1111b	Rs/Rs2/Rd	R0(SP) ～ R15

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

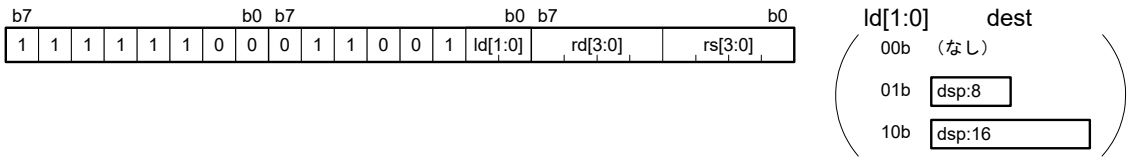


ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

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

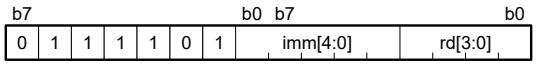
(2) BCLR src, dest



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

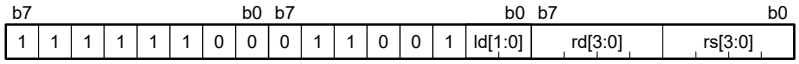
(3) BCLR src, dest



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

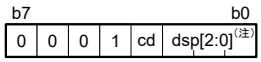
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

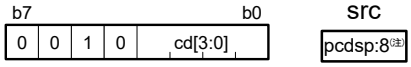


注 . dsp[2:0] で pcdsp:3 = src を指定します。

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

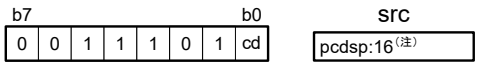
(2) B*Cnd*.B src



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

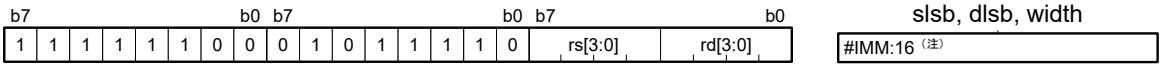
BFMOV

BFMOV

【コードサイズ】

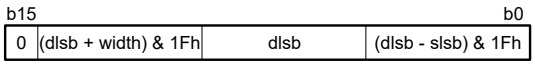
構文	対象					コードサイズ（バイト）
	slsb	dlsb	width	src	dest	
(1) BFMOV　slsb, dlsb, width, src, dest	#IMM:5	#IMM:5	#IMM:5	Rs	Rd	5

(1) BFMOV slsb, dlsb, width, src, dest



注 .　命令オペランドの slsb, dlsb, width の値に従って、次の規則で生成してください。
#IMM:16 = (((dlsb + width) & 1Fh) << 10) | (dlsb << 5) | ((dlsb - slsb) & 1Fh)

イメージ



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

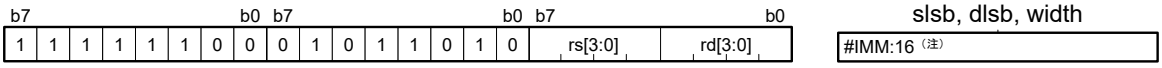
BFMOVZ

BFMOVZ

【コードサイズ】

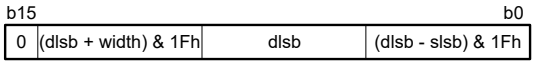
構文	対象					コードサイズ（バイト）
	slsb	dlsb	width	src	dest	
(1) BFMOVZ　slsb, dlsb, width, src, dest	#IMM:5	#IMM:5	#IMM:5	Rs	Rd	5

(1) BFMOVZ slsb, dlsb, width, src, dest



注 .　命令オペランドの slsb, dlsb, width の値に従って、次の規則で生成してください。
#IMM:16 = (((dlsb + width) & 1Fh) << 10) | (dlsb << 5) | ((dlsb - slsb) & 1Fh)

イメージ



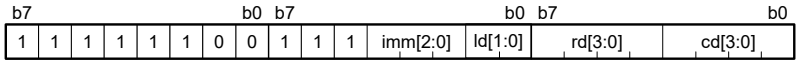
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

BM*Cnd* BM*Cnd*

【コードサイズ】

構文	src	dest	コードサイズ（バイト）
(1) 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





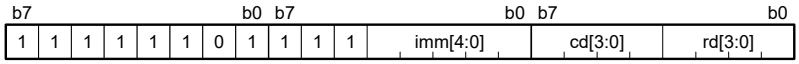
imm[2:0]	src	
000b ～ 111b	#IMM:3	0 ～ 7

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）

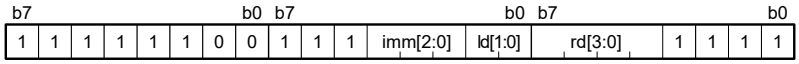
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

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



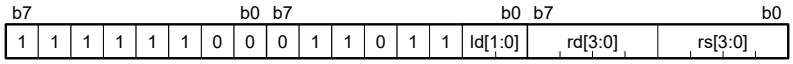


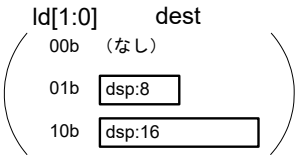
imm[2:0]	src	
000b ～ 111b	#IMM:3	0 ～ 7

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

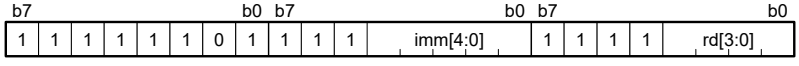




ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

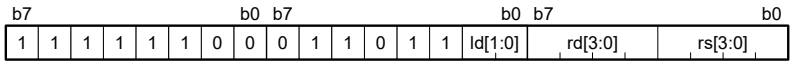
(3) BNOT src, dest



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

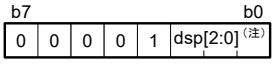
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

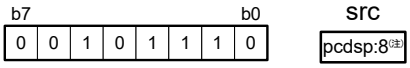
(1) BRA.S src



注 . dsp[2:0] で pcdsp:3 = src を指定します。

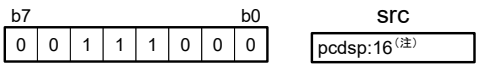
dsp[2:0]	分岐距離
011b	3
100b	4
101b	5
110b	6
111b	7
000b	8
001b	9
010b	10

(2) BRA.B src



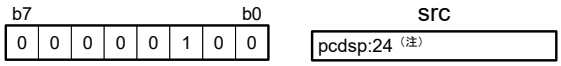
注 . pcdsp:8 = src が示す番地 – 命令の先頭番地

(3) BRA.W src



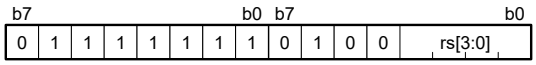
注 . pcdsp:16 = src が示す番地 – 命令の先頭番地

(4) BRA.A src



注 . pcdsp:24 = src が示す番地 – 命令の先頭番地

(5) BRA.L src



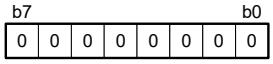
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

BRK BRK

【コードサイズ】

構文	コードサイズ（バイト）
(1) BRK	1

(1) BRK

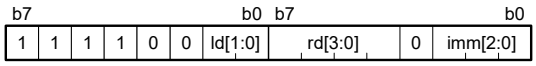


BSET BSET

【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



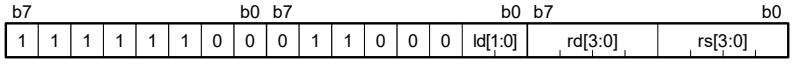


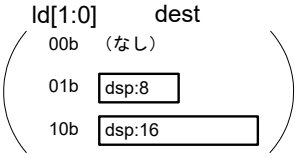
ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

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

(2) BSET src, dest

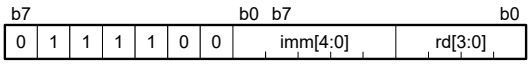




ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

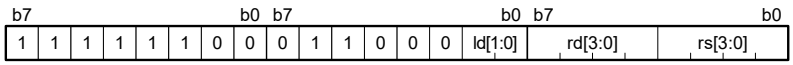
(3) BSET src, dest



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

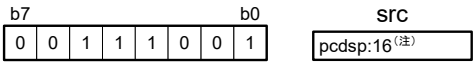
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

BSR BSR

【コードサイズ】

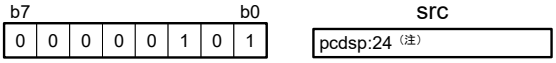
構文	src	コードサイズ（バイト）
(1) BSR.W src	pcdsp:16	3
(2) BSR.A src	pcdsp:24	4
(3) BSR.L src	Rs	2

(1) BSR.W src



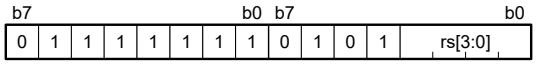
注 . pcdsp:16 = src が示す番地 – 命令の先頭番地

(2) BSR.A src



注 . pcdsp:24 = src が示す番地 – 命令の先頭番地

(3) BSR.L src



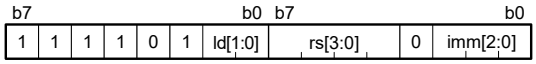
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

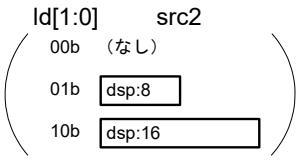
BTST BTST

【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



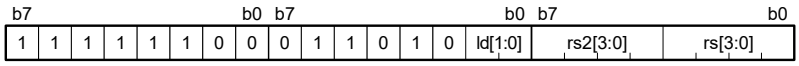


ld[1:0]	src2
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

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

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

(2) BTST src, src2

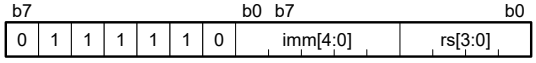




ld[1:0]	src2
00b	[Rs2]
01b	dsp:8[Rs2]
10b	dsp:16[Rs2]

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

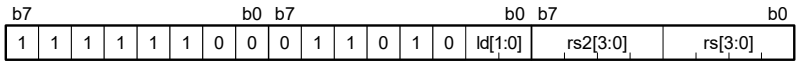
(3) BTST src, src2



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

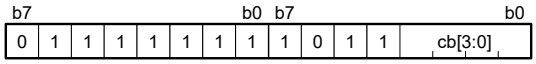
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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



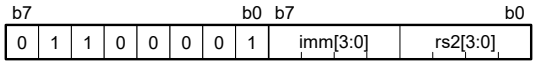
cb[3:0]	dest	
0000b	flag	C
0001b		Z
0010b		S
0011b		O
0100b		（予約）
0101b		（予約）
0110b		（予約）
0111b		（予約）
1000b		I
1001b		U
1010b		（予約）
1011b		（予約）
1100b		（予約）
1101b		（予約）
1110b		（予約）
1111b		（予約）

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”)

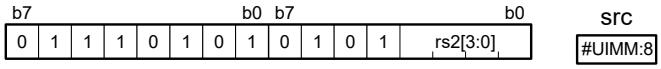
(1) CMP src, src2



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

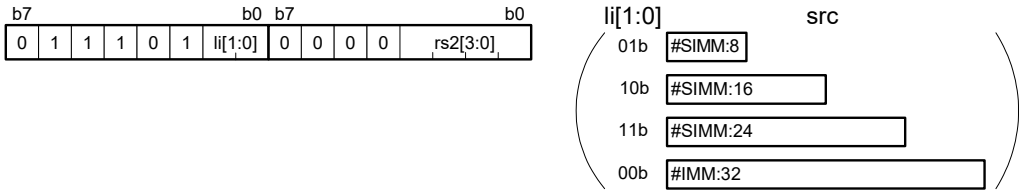
rs2[3:0]	src2	
0000b ～ 1111b	Rs	R0(SP) ～ R15

(2) CMP src, src2



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

(3) CMP src, src2

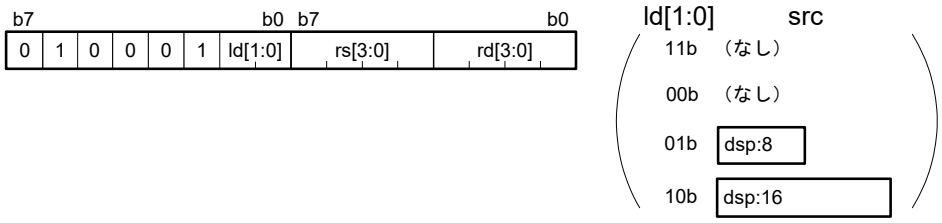


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

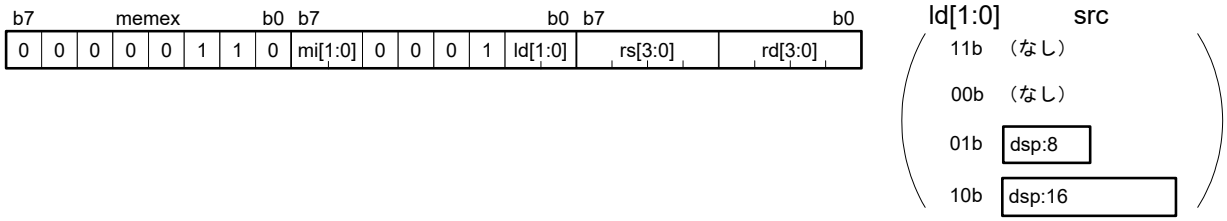
rs2[3:0]	src2	
0000b ～ 1111b	Rs	R0(SP) ～ R15

(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]

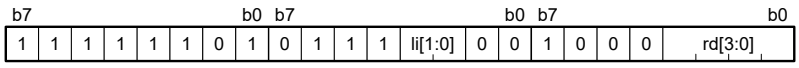
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

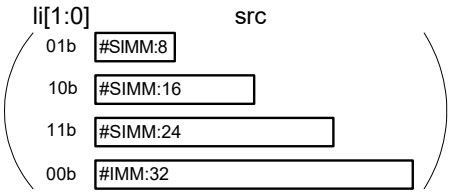
DIV DIV

【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



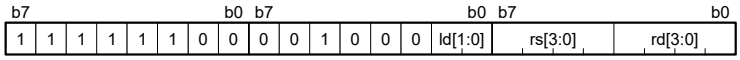


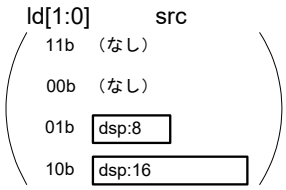
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

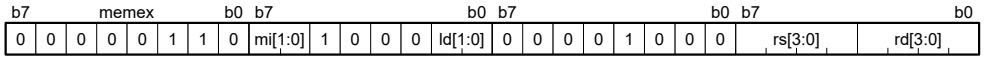
(2) DIV src, dest

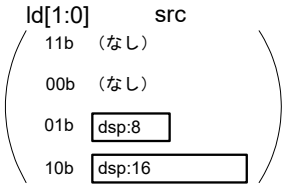
memex == “UB” または src == Rs の場合





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]

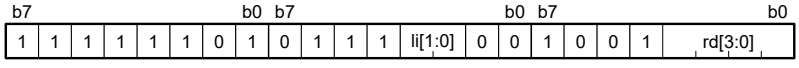
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

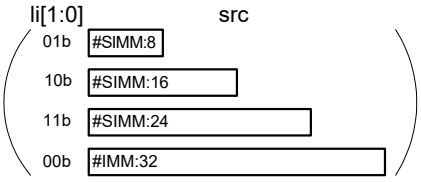
DIVU DIVU

【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



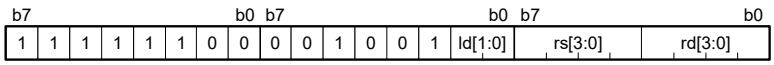


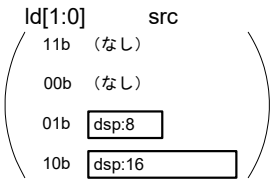
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

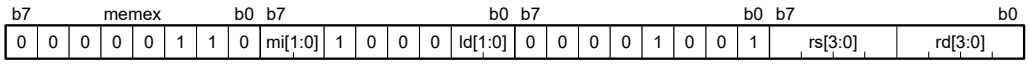
(2) DIVU src, dest

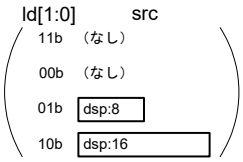
memex == “UB” または src == Rs の場合





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]

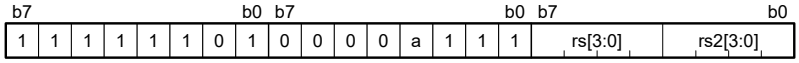
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

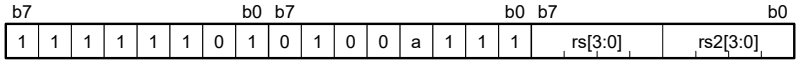
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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

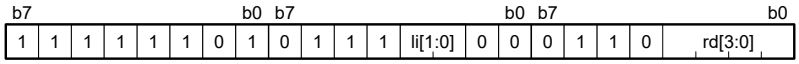
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

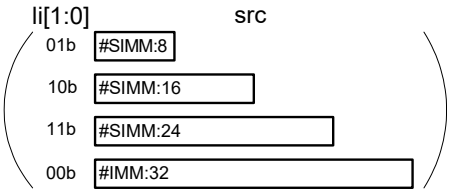
EMUL EMUL

【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



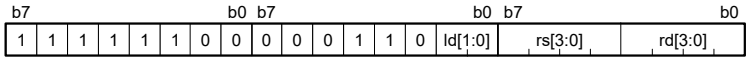


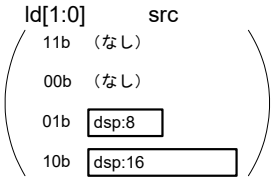
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

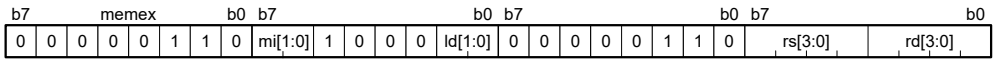
(2) EMUL src, dest

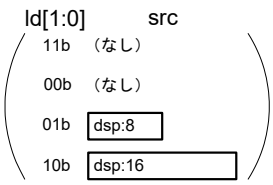
memex == “UB” または src == Rs の場合





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

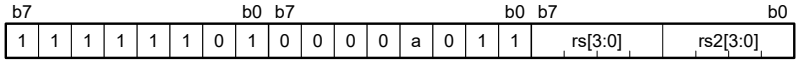
rd[3:0]	dest	
0000b ～ 1110b	Rd	R0(SP) ～ R14

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

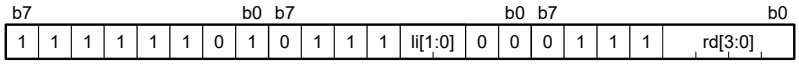
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

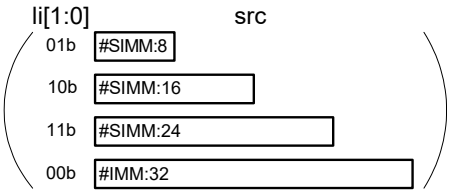
EMULU EMULU

【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



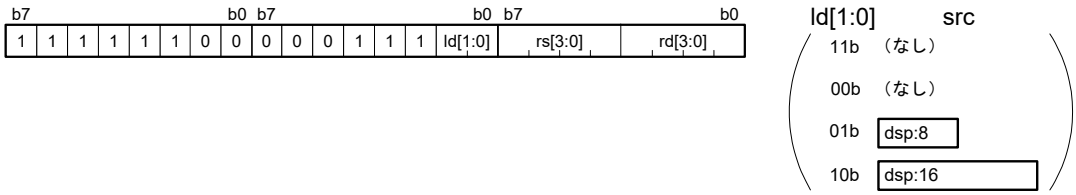


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

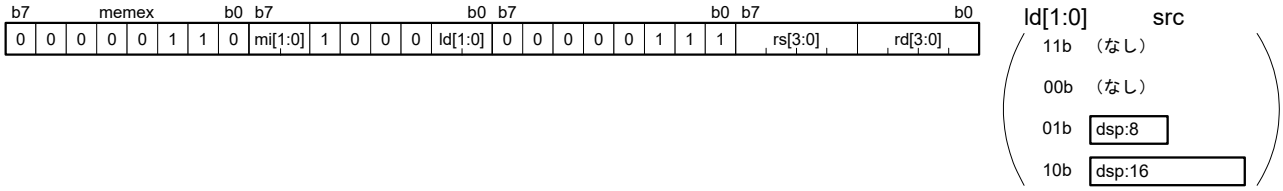
rd[3:0]	dest	
0000b ～ 1110b	Rd	R0(SP) ～ R14

(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

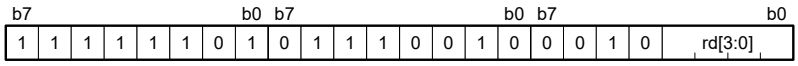
rd[3:0]	dest	
0000b ～ 1110b	Rd	R0(SP) ～ R14

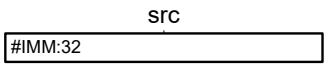
FADD FADD

【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

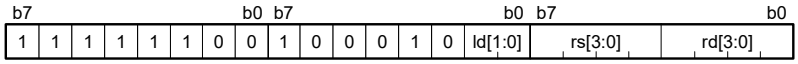
(1) FADD src, dest

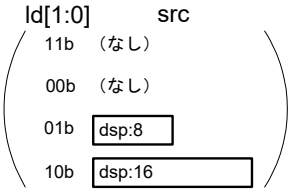




rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) FADD src, dest

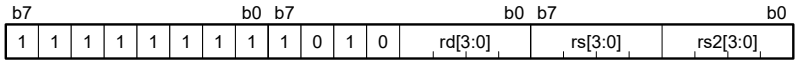




ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

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

(3) FADD src, src2, dest



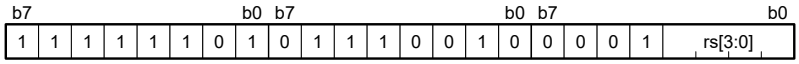
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ～ 1111b	Rs/Rs2/Rd	R0(SP) ～ R15

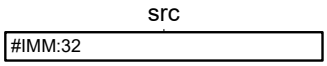
FCMP FCMP

【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

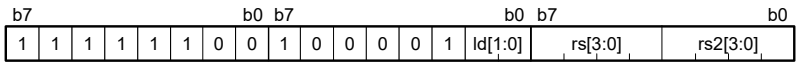
(1) FCMP src, src2

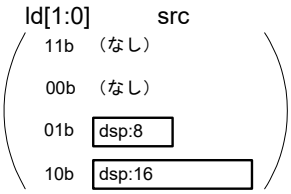




rs[3:0]	src2	
0000b ～ 1111b	Rs	R0(SP) ～ R15

(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

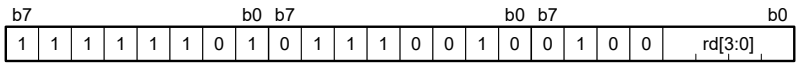
FDIV

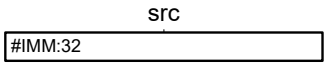
FDIV

【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

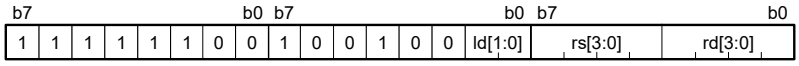
(1) FDIV src, dest

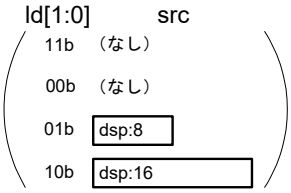




rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) FDIV src, dest





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

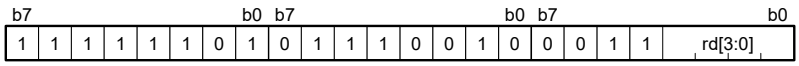
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

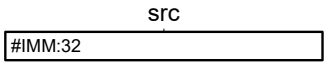
FMUL FMUL

【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

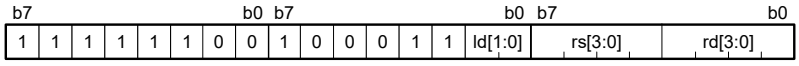
(1) FMUL src, dest

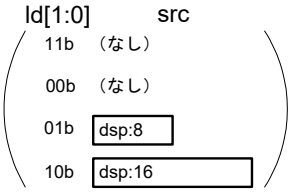




rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) FMUL src, dest

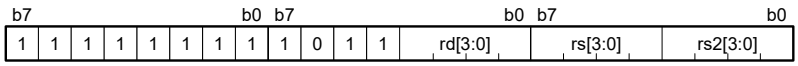




ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

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]

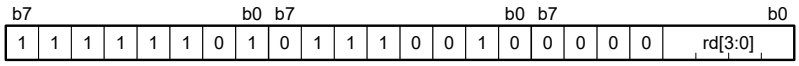
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

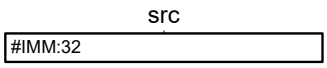
FSUB　FSUB

【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

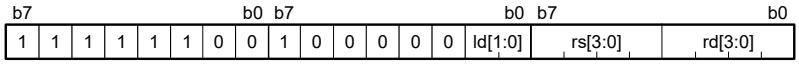
(1) FSUB　src, dest

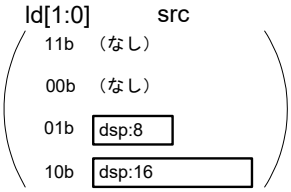




rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) FSUB　src, dest

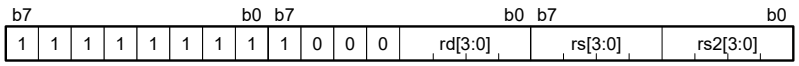




ld[1:0]	src
11b	Rs
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

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

(3) FSUB　src, src2, dest



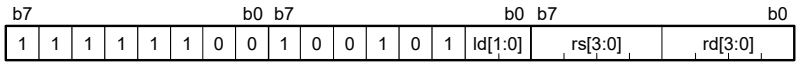
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ～ 1111b	Rs/Rs2/Rd	R0(SP) ～ R15

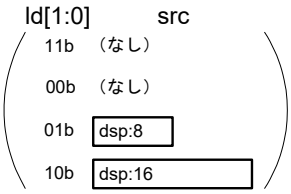
FTOI FTOI

【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]

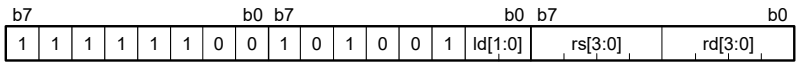
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

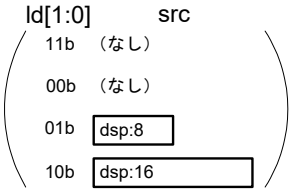
FTOU FTOU

【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]

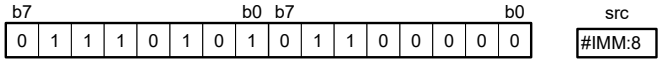
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

INT INT

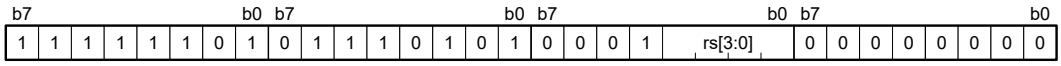
【コードサイズ】

構文	src	コードサイズ（バイト）
(1) INT src	#IMM:8	3
(2) INT src	Rs	4

(1) INT src



(2) INT src



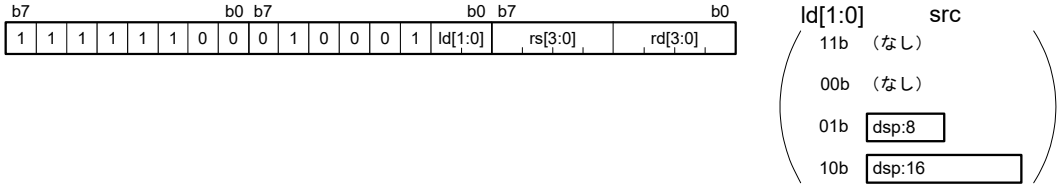
ITOF ITOF

【コードサイズ】

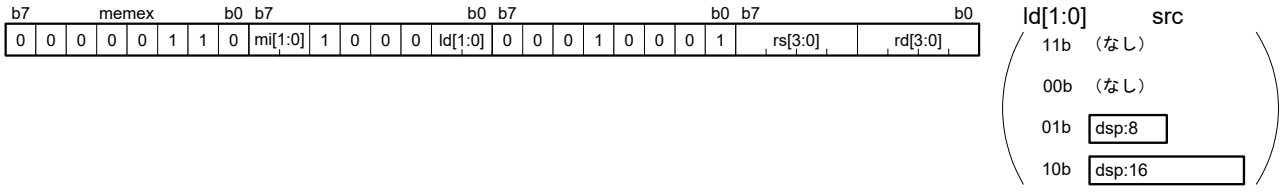
構文	src	dest	コードサイズ（バイト）
(1) ITOF src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == “UB”) 4 (memex != “UB”)
	dsp:8[Rs].memex	Rd	4 (memex == “UB”) 5 (memex != “UB”)
	dsp:16[Rs].memex	Rd	5 (memex == “UB”) 6 (memex != “UB”)

(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]

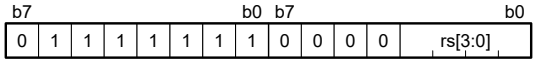
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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



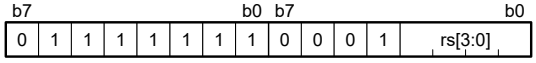
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

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



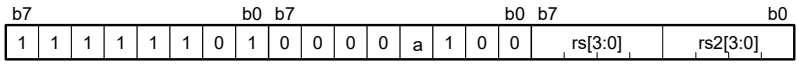
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

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

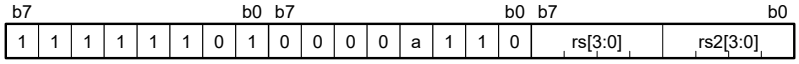
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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

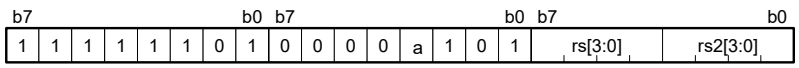
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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

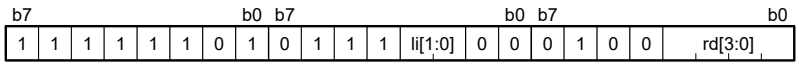
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

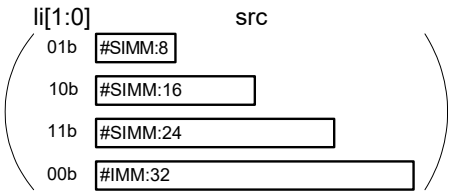
MAX MAX

【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



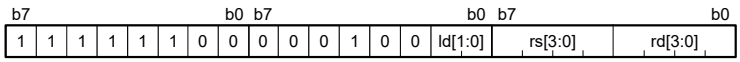


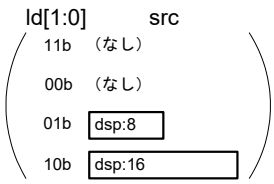
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

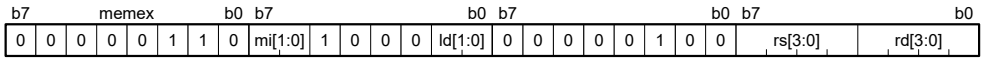
(2) MAX src, dest

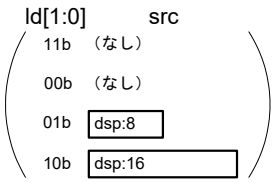
memex == “UB” または src == Rs の場合





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]

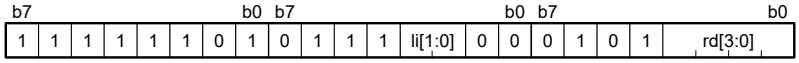
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

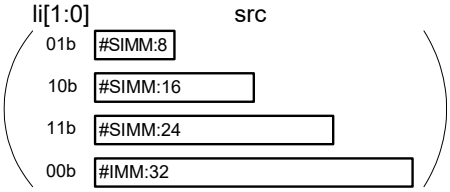
MIN MIN

【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



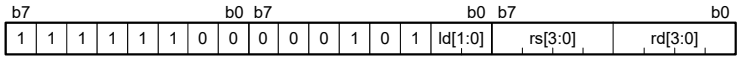


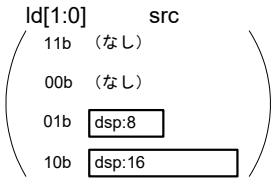
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

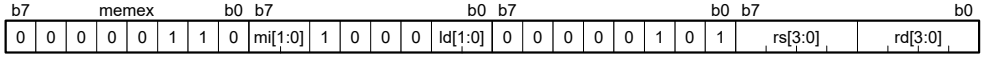
(2) MIN src, dest

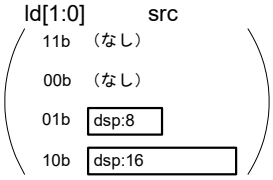
memex == “UB” または src == Rs の場合





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

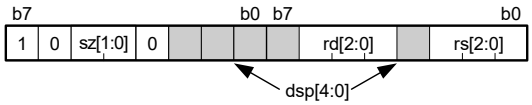
MOV MOV

【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
(12) MOV.size src, dest	B/W/L	size	Rs	[Ri, Rb]	3

構文	size	処理サイズ	src	dest	コードサイズ（バイト）
(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

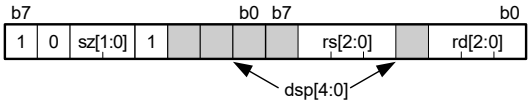


sz[1:0]	size
00b	B
01b	W
10b	L

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

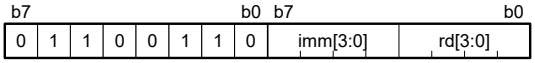


sz[1:0]	size
00b	B
01b	W
10b	L

dsp[4:0]	dsp:5
00000b ～ 11111b	0 ～ 31

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

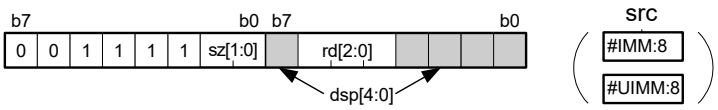
(3) MOV.size src, dest



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

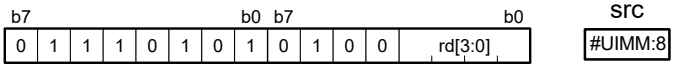


sz[1:0]	size
00b	B
01b	W
10b	L

dsp[4:0]	dsp:5
00000b～11111b	0～31

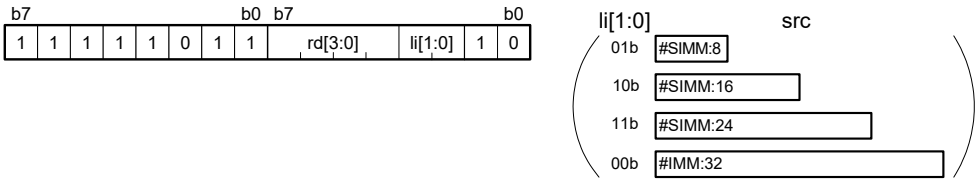
rd[2:0]	dest	
000b ～ 111b	Rd	R0(SP) ～ R7

(5) MOV.size src, dest



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

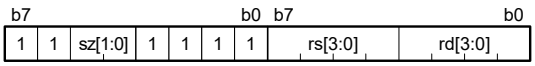
(6) MOV.size src, dest



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



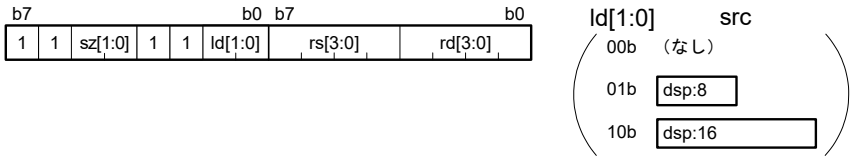
ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

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

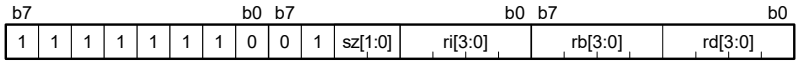


sz[1:0]	size
00b	B
01b	W
10b	L

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

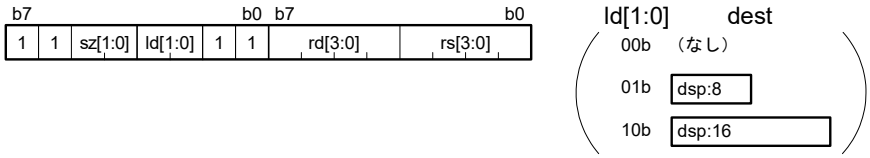
(10) MOV.size src, dest



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

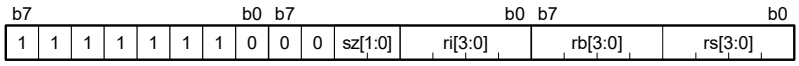


sz[1:0]	size
00b	B
01b	W
10b	L

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

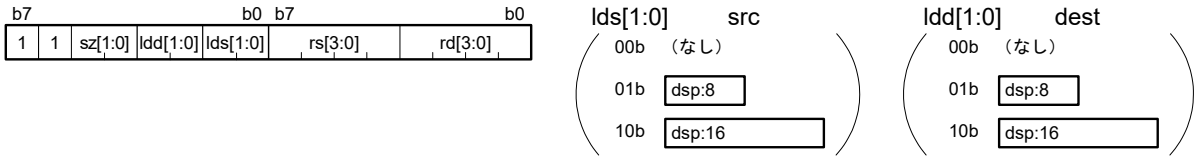
(12) MOV.size src, dest



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

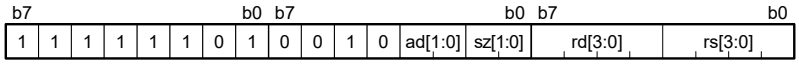


sz[1:0]	size
00b	B
01b	W
10b	L

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

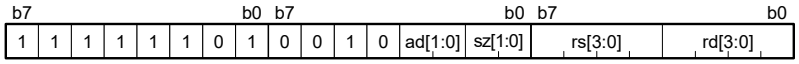


ad[1:0]	addressing
00b	Rs, [Rd+]
01b	Rs, [−Rd]

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

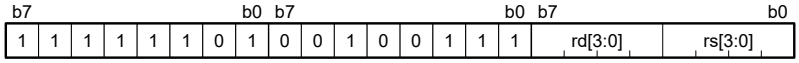
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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



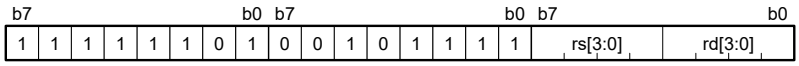
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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



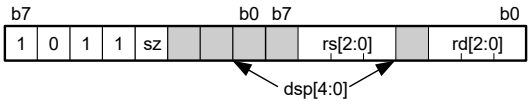
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

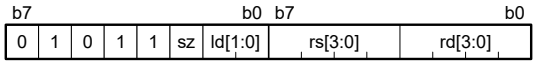


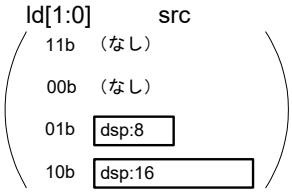
sz	size
0b	B
1b	W

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



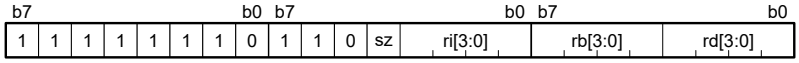


sz	size
0b	B
1b	W

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

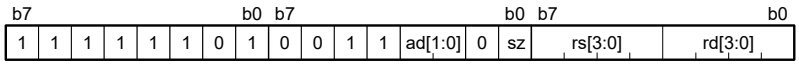
(3) MOVU.size src, dest



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

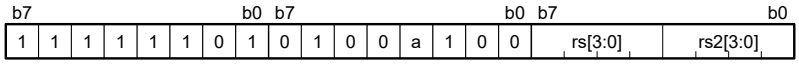
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

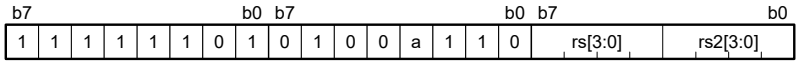
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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

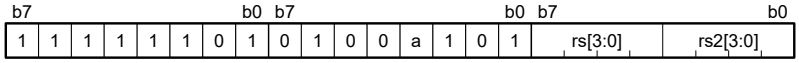
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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

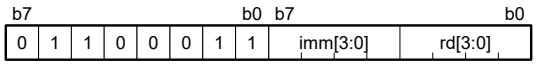
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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

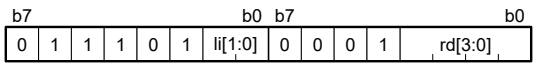
(1) MUL src, dest

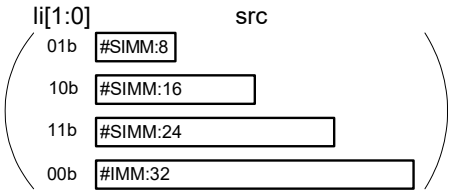


imm[3:0]	src	
0000b ～ 1111b	#UIMM:4	0 ～ 15

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

(2) MUL src, dest



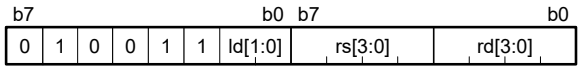


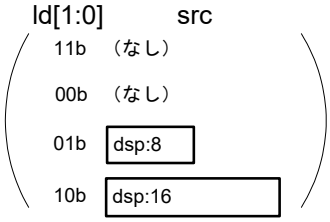
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

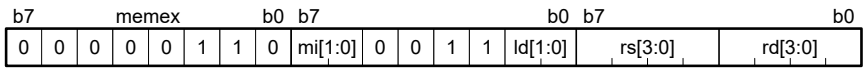
(3) MUL src, dest

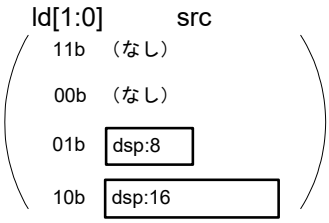
memex == “UB” または src == Rs の場合





memex != “UB” の場合



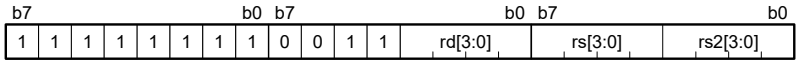


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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



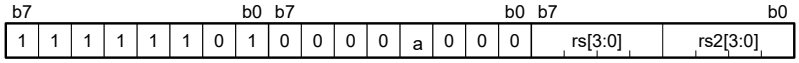
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ～ 1111b	Rs/Rs2/Rd	R0(SP) ～ R15

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

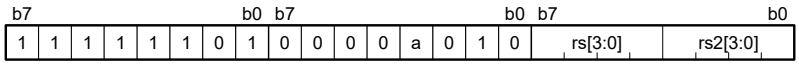
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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

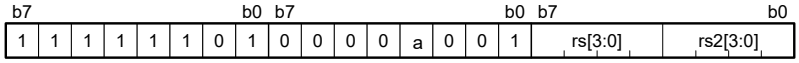
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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

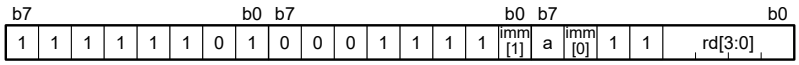
rs[3:0]/rs2[3:0]	src/src2	
0000b ～ 1111b	Rs/Rs2	R0(SP) ～ R15

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

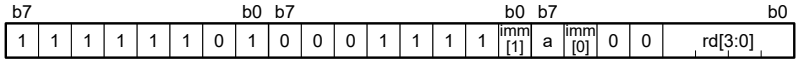
rd[3:0]	dest	
0000b～1111b	Rd	R0(SP) ～ R15

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

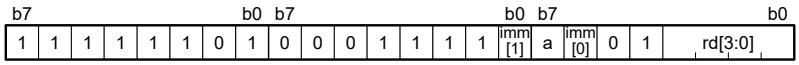
rd[3:0]	dest	
0000b～1111b	Rd	R0(SP) ～ R15

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

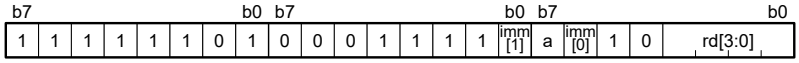
rd[3:0]	dest	
0000b～1111b	Rd	R0(SP) ～ R15

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

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

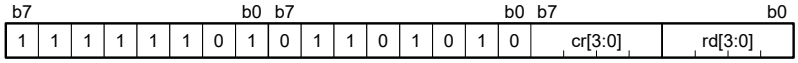
MVFC

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		IVC
0110b		（予約）
0111b		（予約）
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b		EXTB
1110b		WPRI
1111b		WID

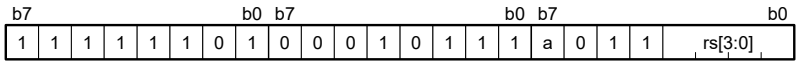
rd[3:0]	dest	
0000b～1111b	Rd	R0(SP)～R15

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

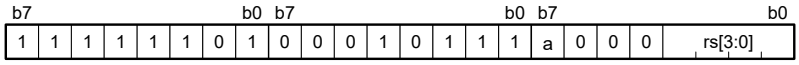
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

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

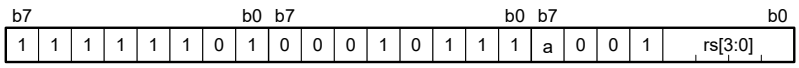
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

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

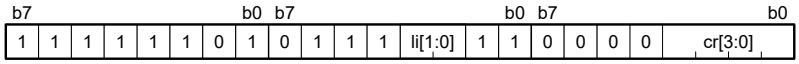
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

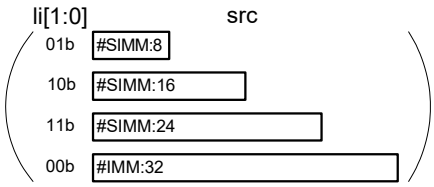
MVTC MVTC

【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

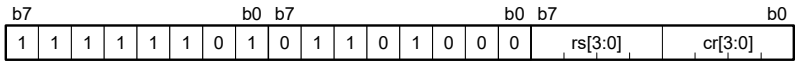




li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

(2) MVTC src, dest



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

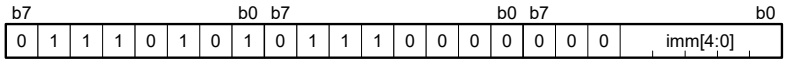
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

MVTIPL MVTIPL

【コードサイズ】

構文	src	コードサイズ（バイト）
(1) MVTIPL src	#IMM:5	3

(1) MVTIPL src



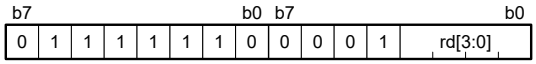
imm[4:0]	#IMM:5
00000b ～ 11111b	0 ～ 31

NEG NEG

【コードサイズ】

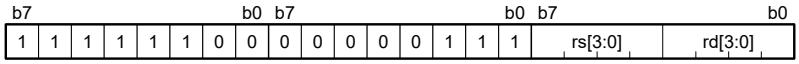
構文	src	dest	コードサイズ（バイト）
(1) NEG dest	—	Rd	2
(2) NEG src, dest	Rs	Rd	3

(1) NEG dest



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

(2) NEG src, dest



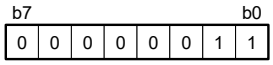
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

NOP NOP

【コードサイズ】

構文	コードサイズ（バイト）
(1) NOP	1

(1) NOP

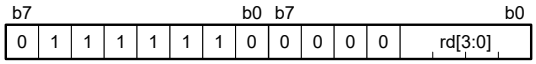


NOT NOT

【コードサイズ】

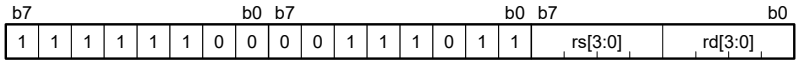
構文	src	dest	コードサイズ（バイト）
(1) NOT dest	—	Rd	2
(2) NOT src, dest	Rs	Rd	3

(1) NOT dest



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

(2) NOT src, dest



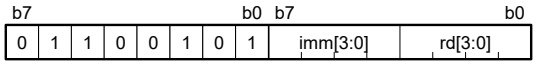
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

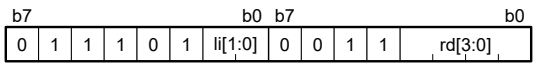
(1) OR src, dest

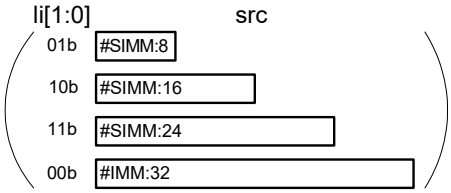


imm[3:0]	src	
0000b ～ 1111b	#UIMM:4	0 ～ 15

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

(2) OR src, dest



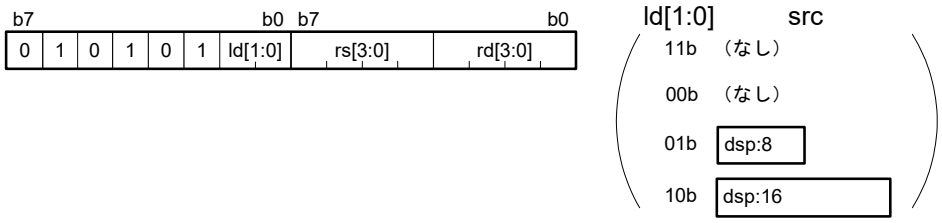


li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

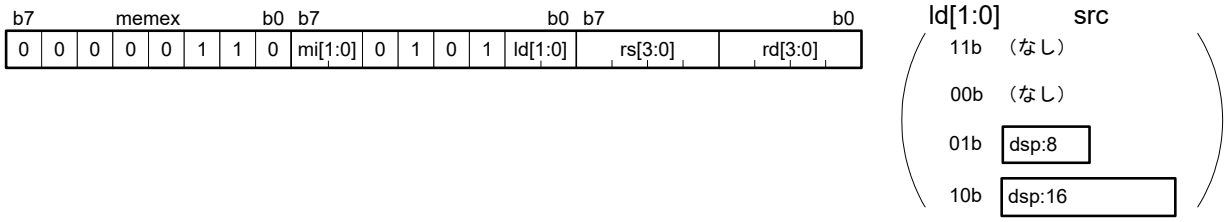
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(3) OR src, dest

memex == “UB” または src == Rs の場合



memex != “UB” の場合

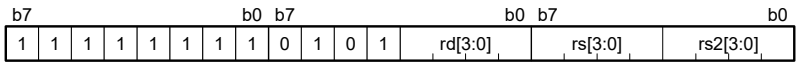


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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

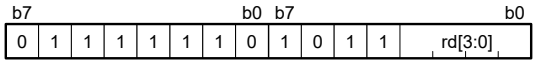
POP

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

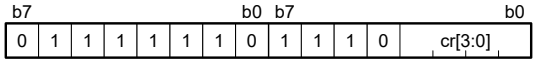
POPC

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		IVC
0110b		（予約）
0111b		（予約）
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b		EXTB
1110b		WPRI
1111b		WID

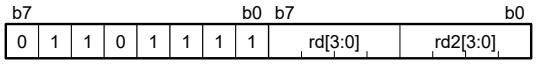
POPM

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

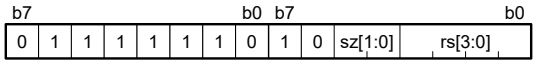
rd2[3:0]	dest2	
0010b ～ 1111b	Rd2	R2 ～ R15

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

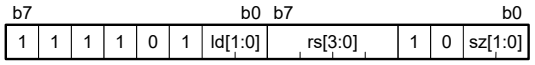
(1) PUSH.size src

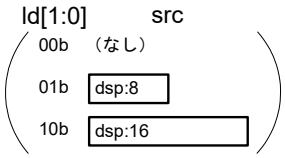


sz[1:0]	size
00b	B
01b	W
10b	L

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

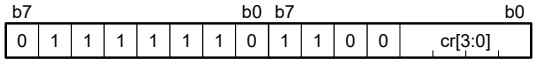
sz[1:0]	size
00b	B
01b	W
10b	L

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



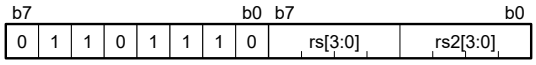
cr[3:0]	src	
0000b	Rx	PSW
0001b		PC
0010b		USP
0011b		FPSW
0100b		（予約）
0101b		IVC
0110b		（予約）
0111b		（予約）
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		FINTV
1100b		INTB
1101b		EXTB
1110b		WPRI
1111b		WID

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

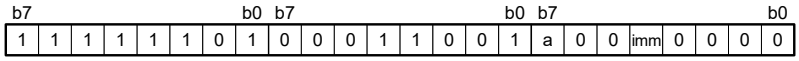
rs2[3:0]	src2	
0010b ～ 1111b	Rs2	R2 ～ R15

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

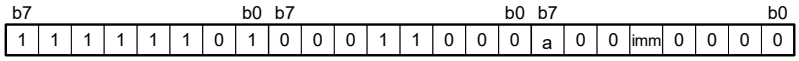
imm	src	
0b, 1b	#IMM:1	1, 2

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 (IMM:1 = 1, 2)	A0, A1	3

(1) RACW src, Adest



a	Adest
0b	A0
1b	A1

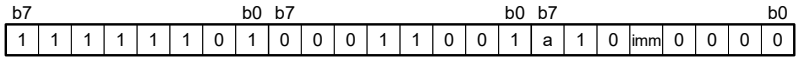
imm	src	
0b, 1b	#IMM:1	1, 2

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

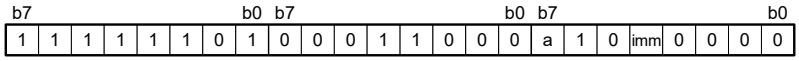
imm	src	
0b, 1b	#IMM:1	1, 2

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

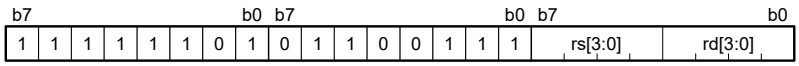
imm	src	
0b, 1b	#IMM:1	1, 2

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



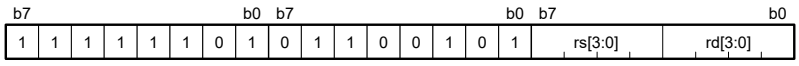
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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



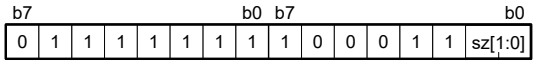
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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



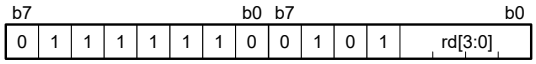
sz[1:0]	size
00b	B
01b	W
10b	L

ROLC ROLC

【コードサイズ】

構文	dest	コードサイズ（バイト）
(1) ROLC dest	Rd	2

(1) ROLC dest



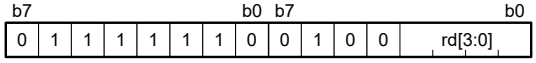
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

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



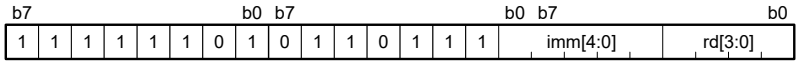
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

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

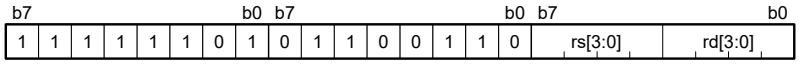
(1) ROTL src, dest



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

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

(2) ROTL src, dest



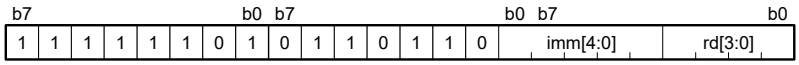
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

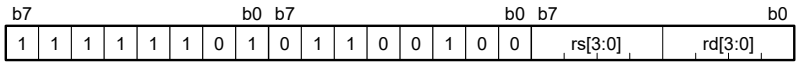
(1) ROTR src, dest



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

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

(2) ROTR src, dest



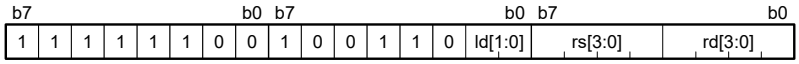
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

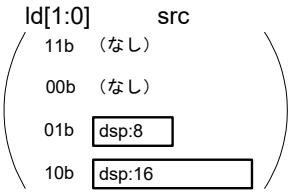
ROUND ROUND

【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]

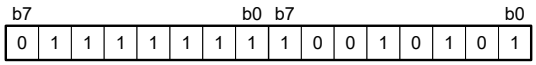
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

RTE RTE

【コードサイズ】

構文	コードサイズ（バイト）
(1) RTE	2

(1) RTE

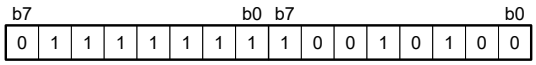


RTFI RTFI

【コードサイズ】

構文	コードサイズ（バイト）
(1) RTFI	2

(1) RTFI

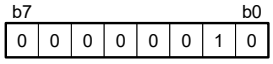


RTS RTS

【コードサイズ】

構文	コードサイズ（バイト）
(1) RTS	1

(1) RTS

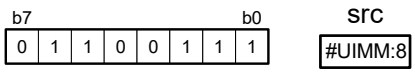


RTSD RTSD

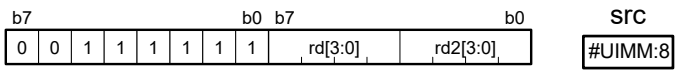
【コードサイズ】

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



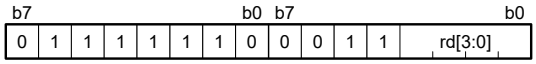
rd[3:0]/rd2[3:0]	dest/dest2	
0001b ～ 1111b	Rd/Rd2	R1 ～ R15

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



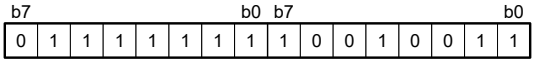
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

SATR SATR

【コードサイズ】

構文	コードサイズ（バイト）
(1) SATR	2

(1) SATR

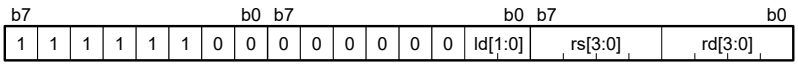


SBB SBB

【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

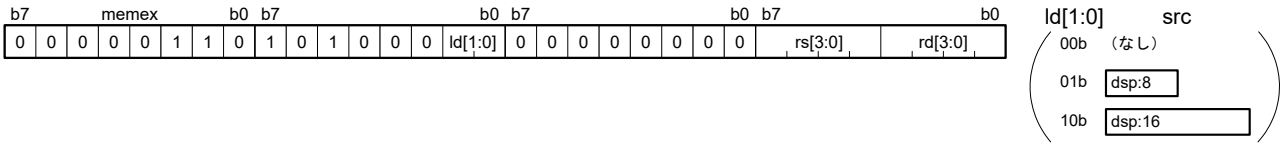
(1) SBB src, dest



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]

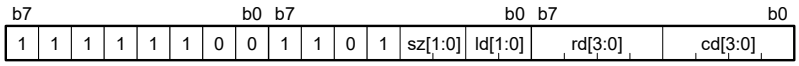
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

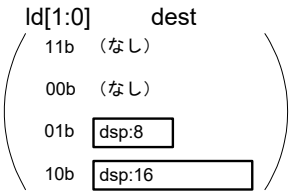
SC*Cnd* SC*Cnd*

【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

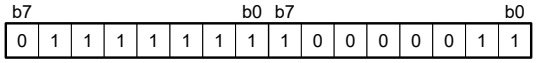
cd[3:0]	SC*Cnd*	cd[3:0]	SC*Cnd*
0000b	SCEQ, SCZ	1000b	SCGE
0001b	SCNE, SCNZ	1001b	SCLT
0010b	SCGEU, SCC	1010b	SCGT
0011b	SCLTU, SCNC	1011b	SCLE
0100b	SCGTU	1100b	SCO
0101b	SCLEU	1101b	SCNO
0110b	SCPZ	1110b	(予約)
0111b	SCN	1111b	(予約)

SCMPU SCMPU

【コードサイズ】

構文	コードサイズ（バイト）
(1) SCMPU	2

(1) SCMPU

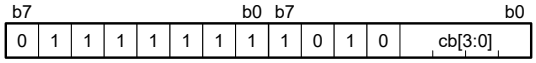


SETPSW SETPSW

【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



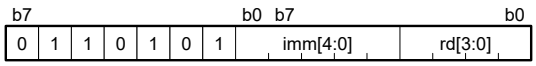
cb[3:0]	dest	
0000b	flag	C
0001b		Z
0010b		S
0011b		O
0100b		（予約）
0101b		（予約）
0110b		（予約）
0111b		（予約）
1000b		I
1001b		U
1010b		（予約）
1011b		（予約）
1100b		（予約）
1101b		（予約）
1110b		（予約）
1111b		（予約）

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

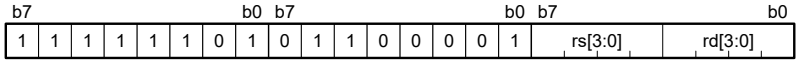
(1) SHAR src, dest



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

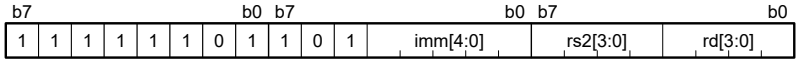
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(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

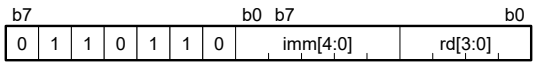
rs2[3:0]/rd[3:0]	src2/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

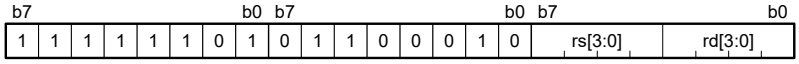
(1) SHLL src, dest



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

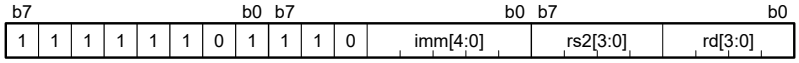
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(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

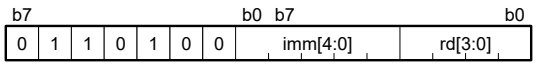
rs2[3:0]/rd[3:0]	src2/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

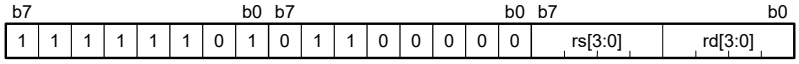
(1) SHLR src, dest



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

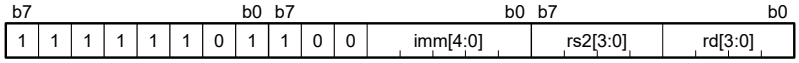
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(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

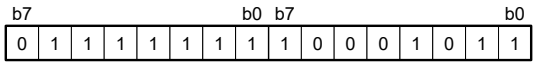
rs2[3:0]/rd[3:0]	src2/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

SMOVB SMOVB

【コードサイズ】

構文	コードサイズ（バイト）
(1) SMOVB	2

(1) SMOVB

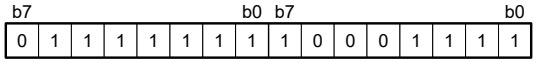


SMOVF　SMOVF

【コードサイズ】

構文	コードサイズ（バイト）
(1) SMOVF	2

(1) SMOVF

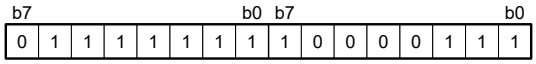


SMOVU　SMOVU

【コードサイズ】

構文	コードサイズ（バイト）
(1) SMOVU	2

(1) SMOVU

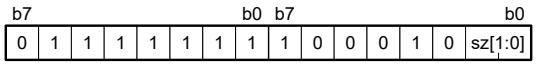


SSTR　SSTR

【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



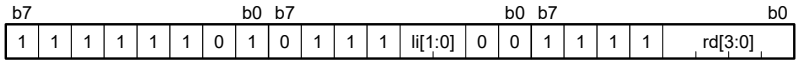
sz[1:0]	size
00b	B
01b	W
10b	L

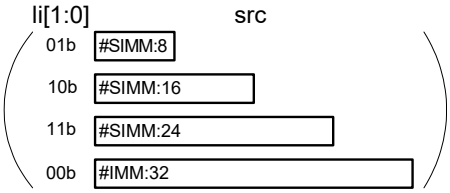
STNZ STNZ

【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

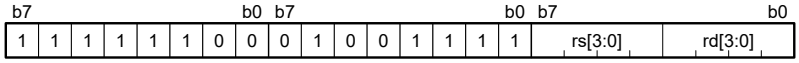




li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

(2) STNZ src, dest



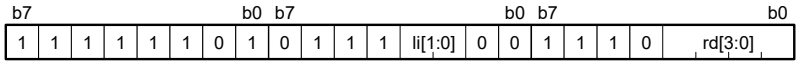
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

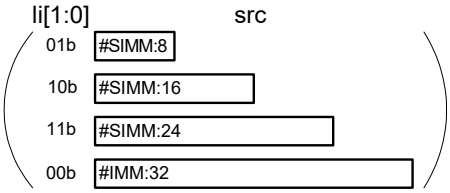
STZ STZ

【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

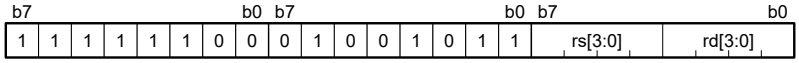




li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

(2) STZ src, dest



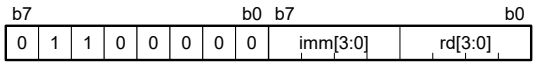
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

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

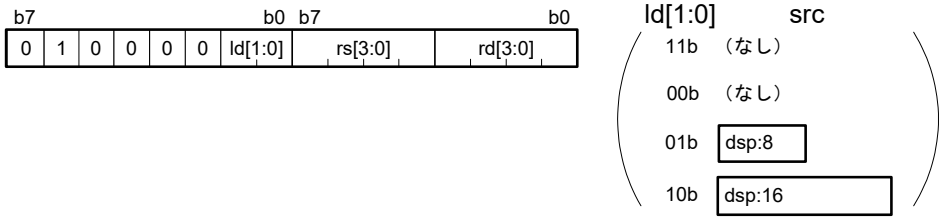


imm[3:0]	src	
0000b ～ 1111b	#UIMM:4	0 ～ 15

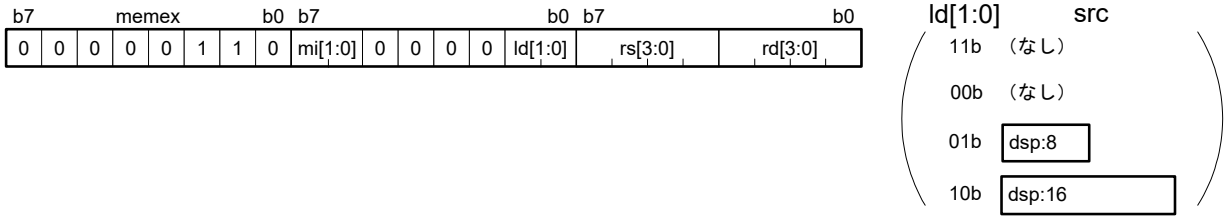
rd[3:0]	dest	
0000b ～ 1111b	Rd	R0(SP) ～ R15

(2) SUB src, dest

memex == “UB” または src == Rs の場合



memex != “UB” の場合

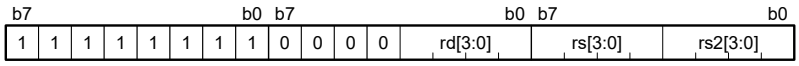


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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



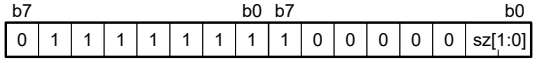
rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b ～ 1111b	Rs/Rs2/Rd	R0(SP) ～ R15

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



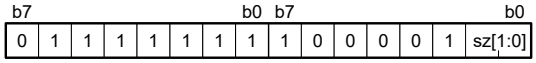
sz[1:0]	size
00b	B
01b	W
10b	L

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



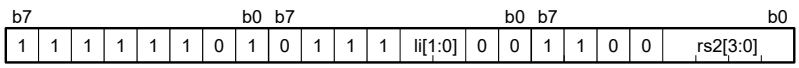
sz[1:0]	size
00b	B
01b	W
10b	L

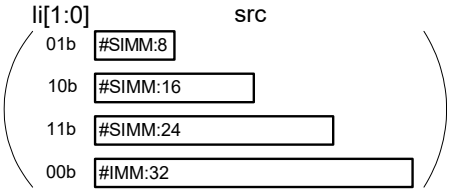
TST TST

【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



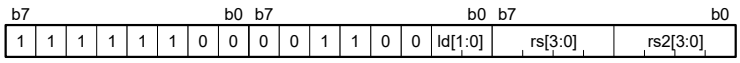


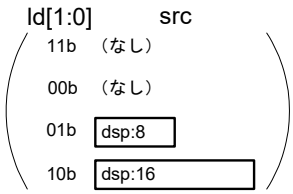
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

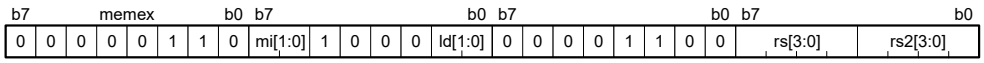
(2) TST src, src2

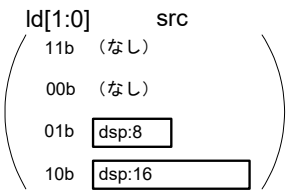
memex == “UB” または src == Rs の場合





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

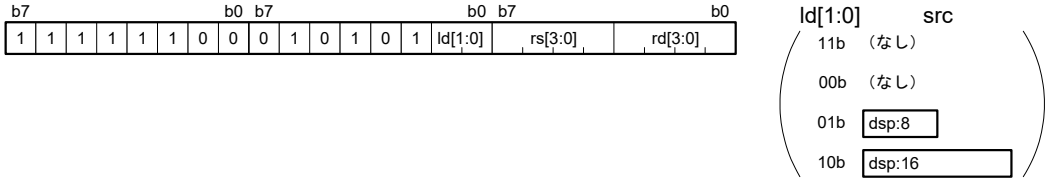
UTOF UTOF

【コードサイズ】

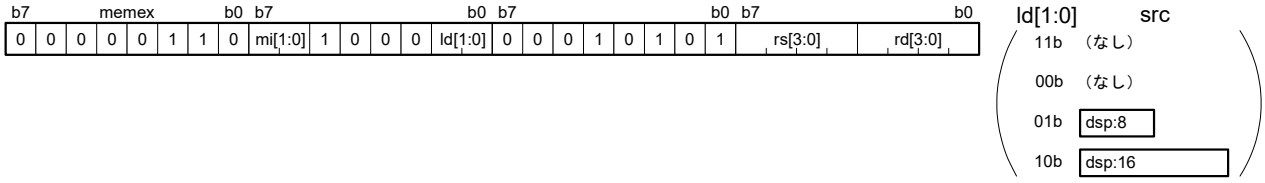
構文	src	dest	コードサイズ（バイト）
(1) UTOF src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == “UB”) 4 (memex != “UB”)
	dsp:8[Rs].memex	Rd	4 (memex == “UB”) 5 (memex != “UB”)
	dsp:16[Rs].memex	Rd	5 (memex == “UB”) 6 (memex != “UB”)

(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

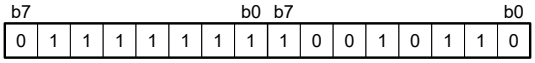
WAIT

WAIT

【コードサイズ】

構文	コードサイズ（バイト）
(1) WAIT	2

(1) WAIT



XCHG

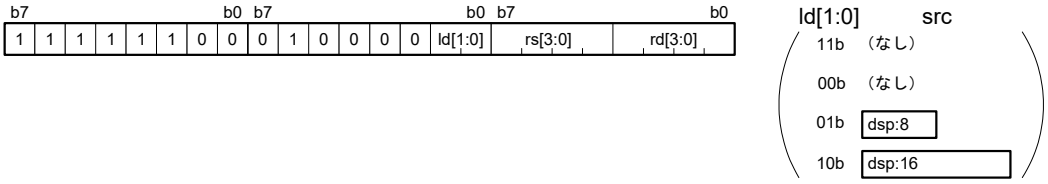
XCHG

【コードサイズ】

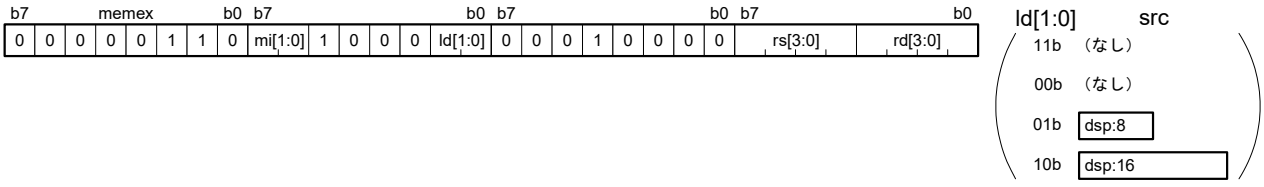
構文	src	dest	コードサイズ（バイト）
(1) XCHG src, dest	Rs	Rd	3
	[Rs].memex	Rd	3 (memex == “UB”) 4 (memex != “UB”)
	dsp:8[Rs].memex	Rd	4 (memex == “UB”) 5 (memex != “UB”)
	dsp:16[Rs].memex	Rd	5 (memex == “UB”) 6 (memex != “UB”)

(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]

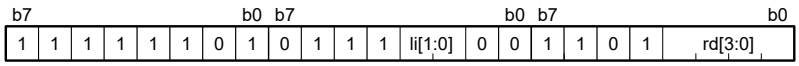
rs[3:0]/rd[3:0]	src/dest	
0000b ～ 1111b	Rs/Rd	R0(SP) ～ R15

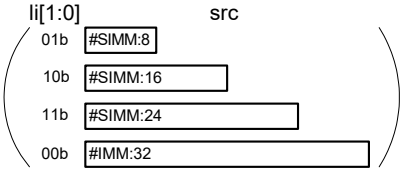
XOR XOR

【コードサイズ】

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

(1) XOR src, dest



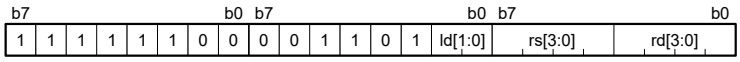


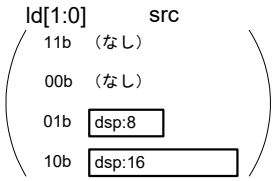
li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

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

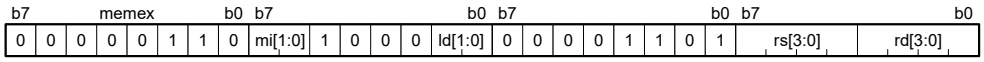
(2) XOR src, dest

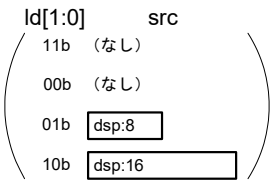
memex == “UB” または src == Rs の場合





memex != “UB” の場合



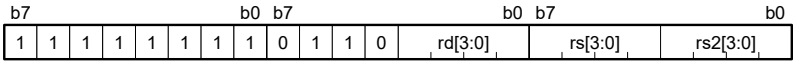


mi[1:0]	memex
00b	B
01b	W
10b	L
11b	UW

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

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

(3) XOR src, src2, dest



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

4.2.2 レジスター括退避機能命令

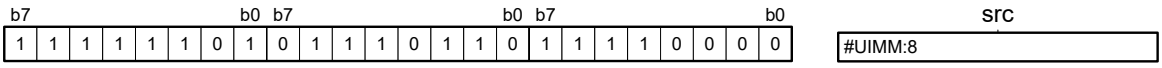
次ページよりレジスター括退避機能命令の命令コードの詳細説明を示します。

RSTR RSTR

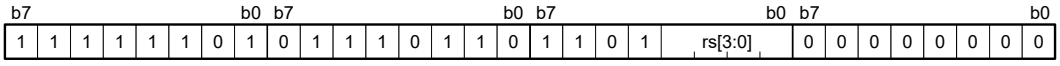
【コードサイズ】

構文	src	コードサイズ（バイト）
(1) RSTR src	#UIMM:8	4
(2) RSTR src	Rs	4

(1) RSTR src



(2) RSTR src



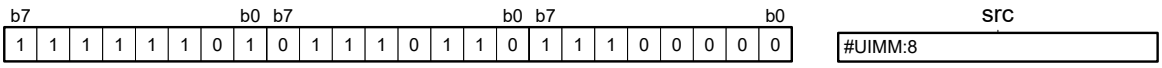
rs[3:0]	src	
0000b ～ 1111b	Rs	R0(SP) ～ R15

SAVE SAVE

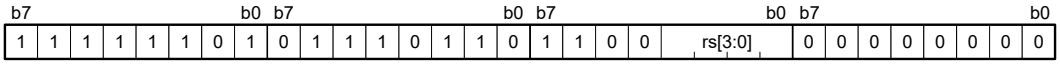
【コードサイズ】

構文	src	コードサイズ（バイト）
(1) SAVE src	#UIMM:8	4
(2) SAVE src	Rs	4

(1) SAVE src



(2) SAVE src



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

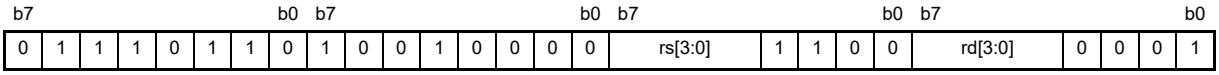
4.2.3 倍精度浮動小数点処理命令

次ページより倍精度浮動小数点処理命令の命令コードの詳細説明を示します。

DABS　DABS

【コードサイズ】

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

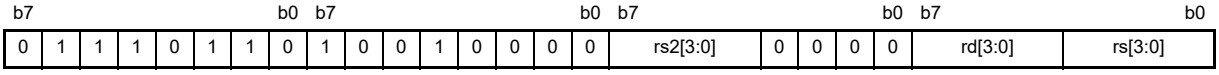


rs[3:0]/rd[3:0]	src/dest	
0000b～1111b	DRs/DRd	DR0～DR15

DADD　DADD

【コードサイズ】

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

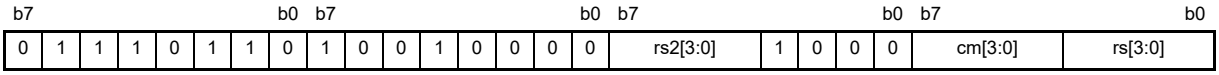


rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b～1111b	DRs/DRs2/DRd	DR0～DR15

DCMPcm DCMPcm

【コードサイズ】

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



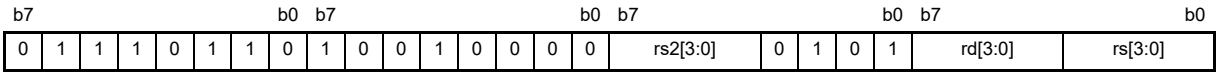
rs[3:0]/rs2[3:0]	src/src2	
0000b～1111b	DRs/DRs2	DR0～DR15

cm	条件
0001b	UN
0010b	EQ
0100b	LT
0110b	LE

DDIV DDIV

【コードサイズ】

構文	対象			コードサイズ（バイト）
	src	src2	dest	
DDIV src, src2, dest	DRs	DRs2	DRd	4



rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b～1111b	DRs/DRs2/DRd	DR0～DR15

DMOV DMOV

【コードサイズ】

構文	フォーマット	size	処理サイズ	対象		コードサイズ（バイト）
				src	dest	
DMOV.size src, dest	レジスタ間転送					
	(1)	D	D	Rs	DRHd	4
	(2)	L	L	Rs	DRHd	4
	(3)	L	L	Rs	DRLd	4
	(4)	L	L	DRHs	Rd	4
	(5)	L	L	DRLs	Rd	4
	(6)	D	D	DRs	DRd	4
	ストア					
	(7)	D	D	DRs	[Rd]	4
		D	D	DRs	dsp:8[Rd]	5
		D	D	DRs	dsp:16[Rd]	6
	ロード					
	(8)	D	D	[Rs]	DRd	4
		D	D	dsp:8[Rs]	DRd	5
		D	D	dsp:16[Rs]	DRd	6
	即値					
	(9)	D	D	#IMM:32	DRHd	7
	(10)	L	L	#IMM:32	DRHd	7
	(11)	L	L	#IMM:32	DRLd	7

(1) DMOV.size src, dest

b7							b0	b7							b0	b7							b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	0	0	0	rs[3:0]				rd[3:0]				0	0	1	1

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

rd[3:0]	dest	
0000b～1111b	DRHd	DRH0～DRH15

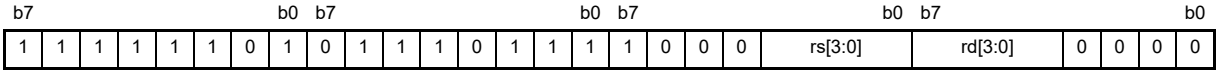
(2) DMOV.size src, dest

b7							b0	b7							b0	b7							b0	b7				b0
1	1	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	0	0	0	rs[3:0]				rd[3:0]				0	0	1	0

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

rd[3:0]	dest	
0000b～1111b	DRHd	DRH0～DRH15

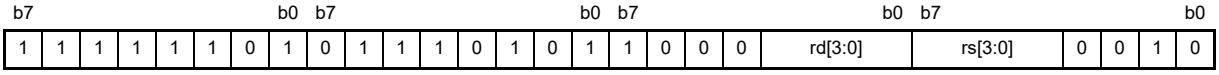
(3) DMOV.size src, dest



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

rd[3:0]	dest	
0000b～1111b	DRLd	DRL0～DRL15

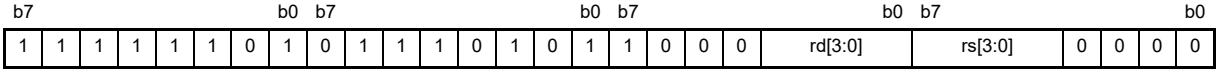
(4) DMOV.size src, dest



rs[3:0]	src	
0000b～1111b	DRHs	DRH0～DRH15

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

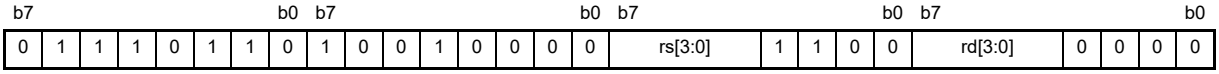
(5) DMOV.size src, dest



rs[3:0]	src	
0000b～1111b	DRLs	DRL0～DRL15

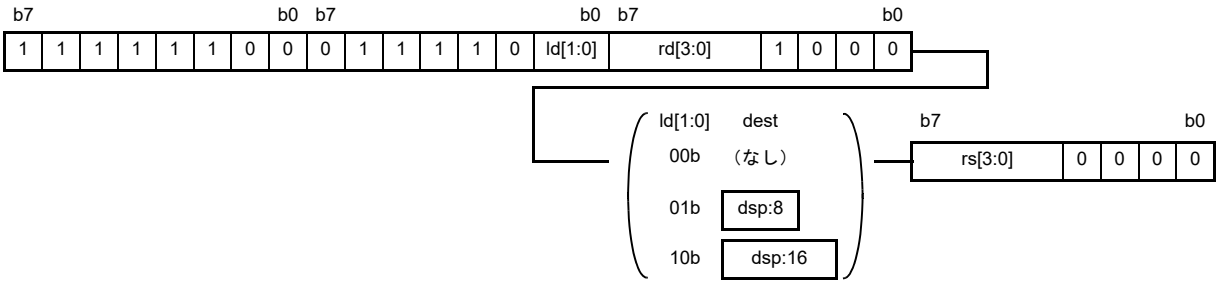
rd[3:0]	dest	
0000b～1111b	Rd	R0(SP)～R15

(6) DMOV.size src, dest



rs[3:0]/rd[3:0]	src/dest	
0000b～1111b	DRs/DRd	DR0～DR15

(7) DMOV.size src, dest

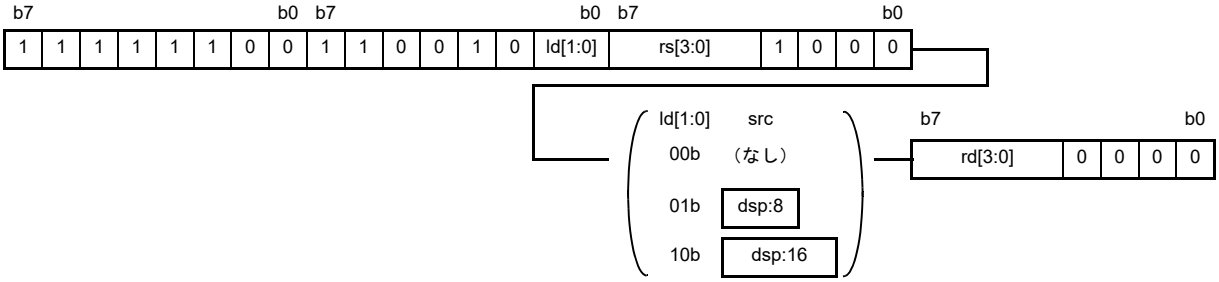


ld[1:0]	dest
00b	[Rd]
01b	dsp:8[Rd]
10b	dsp:16[Rd]

rs[3:0]	src	
0000b～1111b	DRs	DR0～DR15

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

(8) DMOV.size src, dest

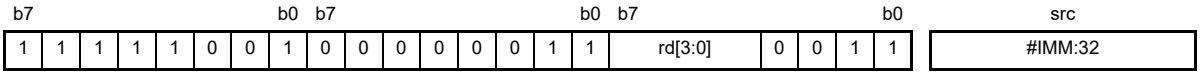


ld[1:0]	src
00b	[Rs]
01b	dsp:8[Rs]
10b	dsp:16[Rs]

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

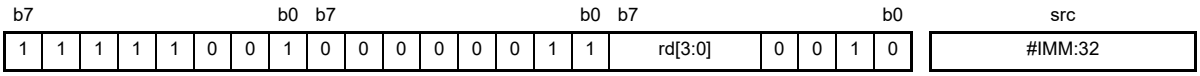
rd[3:0]	dest	
0000b～1111b	Rd	DR0～DR15

(9) DMOV.size src, dest



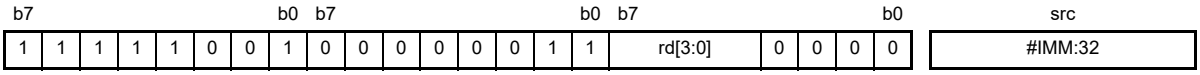
rd[3:0]	dest	
0000b～1111b	DRHd	DRH0～DRH15

(10) DMOV.size src, dest



rd[3:0]	dest	
0000b～1111b	DRHd	DRH0～DRH15

(11) DMOV.size src, dest

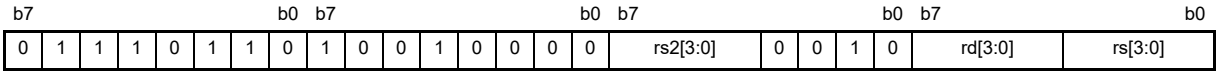


rd[3:0]	dest	
0000b～1111b	DRLd	DRL0～DRL15

DMUL　DMUL

【コードサイズ】

構文	対象			コードサイズ（バイト）
	src	src2	dest	
DMUL src, src2, dest	DRs	DRs2	DRd	4

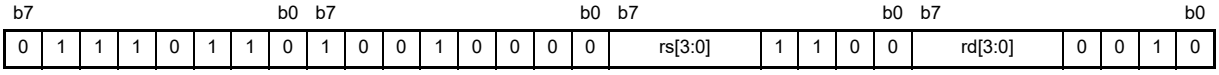


rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b～1111b	DRs/DRs2/DRd	DR0～DR15

DNEG　DNEG

【コードサイズ】

構文	対象		コードサイズ（バイト）
	dest	dest	
DNEG src, dest	DRd	DRd	4



rs[3:0]/rd[3:0]	src/dest	
0000b～1111b	DRs/DRd	DR0～DR15

DPOPM DPOPM

【コードサイズ】

構文	フォーマット	処理サイズ	対象		コードサイズ（バイト）
			dest	dest2	
DPOPM.size dest-dest2	(1)	D	DRd	DRd2	3
	(2)	L	DCRd	DCRd2	3

(1) DPOPM.size dest-dest2

b7							b0	b7							b0	b7								b0
0	1	1	1	0	1	0	1	1	0	1	1	1	0	0	0	rd[3:0]				nm[3:0]				

rd[3:0]	dest	
0000b～1111b	DRd	DR0～DR15

nm[3:0]	num(dest2 – dest)
0000b～1111b	d2 – d（注）

注． d2 – d は DRd – DRd2 間にある（DRd, DRd2 含む）すべてのレジスタの個数 – 1 を示します。

(2) DPOPM.size dest-dest2

b7							b0	b7							b0	b7								b0
0	1	1	1	0	1	0	1	1	0	1	0	1	0	0	0	rd[3:0]				nm[3:0]				

rd[3:0]	dest	
0000b	DCRd	DCR0/DPSW
0001b		DCR1/DCMR
0010b		DCR2/DECNT
0011b		DCR3/DEPC

nm[3:0]	num(dest2 – dest)
0000b～0011b	d2 – d（注）

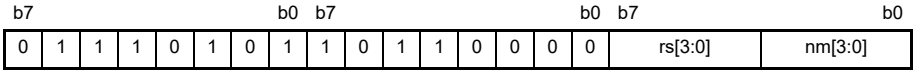
注． d2 – d は DCRd – DCRd2 間にある（DCRd, DCRd2 含む）すべてのレジスタの個数 – 1 を示します。

DPUSHM DPUSHM

【コードサイズ】

構文	フォーマット	処理サイズ	対象		コードサイズ（バイト）
			src	src2	
DPUSHM.size src-src2	(1)	D	DRs	DRs2	3
	(2)	L	DCRs	DCRs2	3

(1) DPUSHM.size src-src2

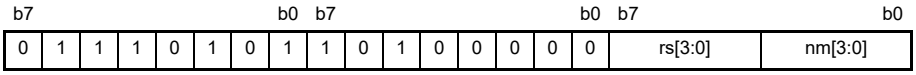


rs[3:0]	src	
0000b～1111b	DRs	DR0～DR15

nm[3:0]	num(src2 – src)
0000b～1111b	s2 – s（注）

注． s2 – s は DRs – DRs2 間にある（DRs, DRs2 含む）すべてのレジスタの個数 – 1 を示します。

(2) DPUSHM.size src-src2



rs[3:0]	src	
0000b	DCRs	DCR0/DPSW
0001b		DCR1/DCMR
0010b		DCR2/DECNT
0011b		DCR3/DEPC

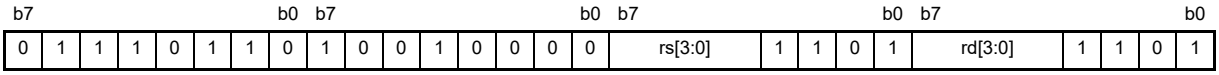
nm[3:0]	num(src2 – src)
0000b～0011b	s2 – s（注）

注． s2 – s は DCRs – DCRs2 間にある（DCRs, DCRs2 含む）すべてのレジスタの個数 – 1 を示します。

DROUND DROUND

【コードサイズ】

構文	対象		コードサイズ（バイト）
	src	dest	
DROUND src, dest	DRs	DRd	4

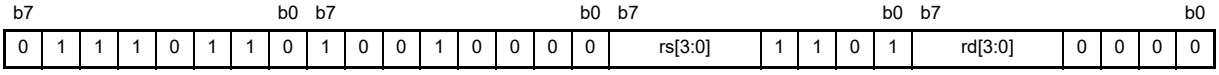


rs[3:0]/rd[3:0]	src/dest	
0000b～1111b	DRs/DRd	DR0～DR15

DSQRT DSQRT

【コードサイズ】

構文	対象		コードサイズ（バイト）
	src	dest	
DSQRT src, dest	DRs	DRd	4

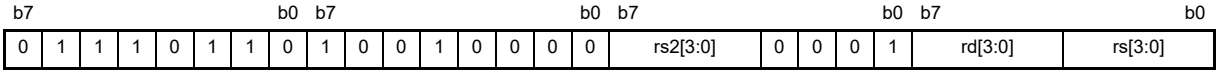


rs[3:0]/rd[3:0]	src/dest	
0000b～1111b	DRs/DRd	DR0～DR15

DSUB DSUB

【コードサイズ】

構文	対象			コードサイズ（バイト）
	src	src2	dest	
DSUB src, src2, dest	DRs	DRs2	DRd	4

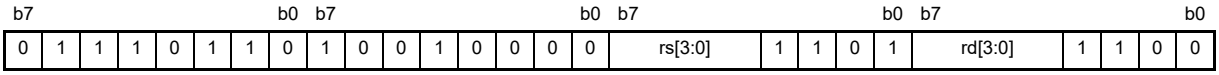


rs[3:0]/rs2[3:0]/rd[3:0]	src/src2/dest	
0000b～1111b	DRs/DRs2/DRd	DR0～DR15

DTOF DTOF

【コードサイズ】

構文	対象		コードサイズ（バイト）
	src	dest	
DTOF src, dest	DRs	DRd	4

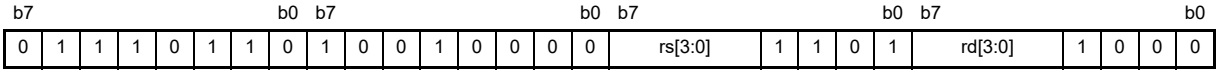


rs[3:0]/rd[3:0]	src/dest	
0000b～1111b	DRs/DRd	DR0～DR15

DTOI DTOI

【コードサイズ】

構文	対象		コードサイズ（バイト）
	src	dest	
DTOI src, dest	DRs	DRd	4

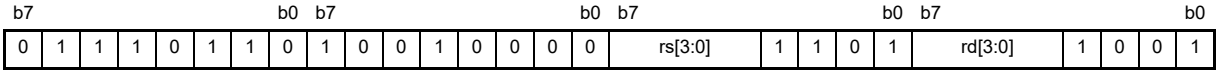


rs[3:0]/rd[3:0]	src/dest	
0000b～1111b	DRs/DRd	DR0～DR15

DTOU DTOU

【コードサイズ】

構文	対象		コードサイズ（バイト）
	src	dest	
DTOU src, dest	DRs	DRd	4

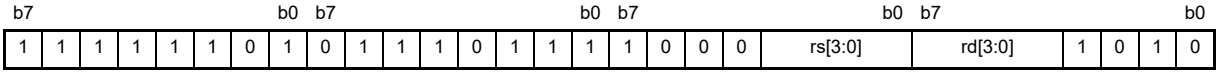


rs[3:0]/rd[3:0]	src/dest	
0000b～1111b	DRs/DRd	DR0～DR15

FTOD FTOD

【コードサイズ】

構文	対象		コードサイズ（バイト）
	src	dest	
FTOD src, dest	Rs	DRd	4



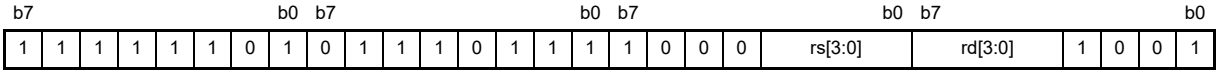
rs[3:0]	src	
0000b～1111b	Rs	R0(SP)～R15

rd[3:0]	dest	
0000b～1111b	DRd	DR0～DR15

ITOD ITOD

【コードサイズ】

構文	対象		コードサイズ（バイト）
	src	dest	
ITOD src, dest	Rs	DRd	4



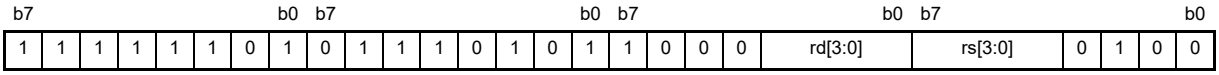
rs[3:0]	src	
0000b～1111b	Rs	R0(SP)～R15

rd[3:0]	dest	
0000b～1111b	DRd	DR0～DR15

MVFDC MVFDC

【コードサイズ】

構文	対象		コードサイズ（バイト）
	src	dest	
MVFDC src, dest	DCRs	Rd	4



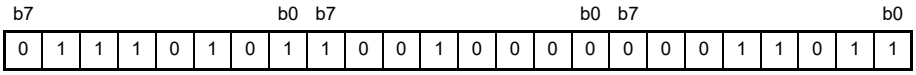
rs[3:0]	src	
0000b	DCRs	DCR0/DPSW
0001b		DCR1/DCMR
0010b		DCR2/DECNT
0011b		DCR3/DEPC

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

MVFDR MVFDR

【コードサイズ】

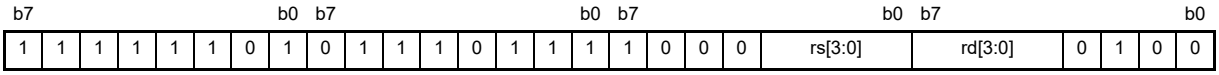
構文	コードサイズ（バイト）
MVFDR	3



MVTDC MVTDC

【コードサイズ】

構文	対象		コードサイズ（バイト）
	src	dest	
MVTDC src, dest	Rs	DCRd	4



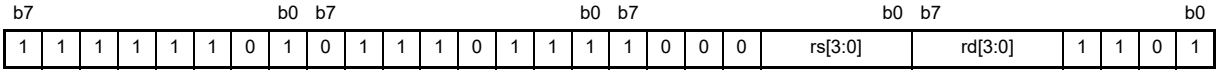
rs[3:0]	src	
0000b～1111b	Rs	R0(SP)～R15

rd[3:0]	dest	
0000b	DCRd	DCR0/DPSW
0001b		DCR1/DCMR
0010b		DCR2/DECNT
0011b		DCR3/DEPC

UTOD UTOD

【コードサイズ】

構文	対象		コードサイズ（バイト）
	src	dest	
UTOD src, dest	Rs	DRd	4



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

rd[3:0]	dest	
0000b～1111b	DRd	DR0～DR15

4.2.4 Protect Zone 命令

次ページより Protect Zone 命令の命令コードの詳細説明を示します。

TRCALL TRCALL

【コードサイズ】

構文	対象	コードサイズ（バイト）
	src	
TRCALL src	#IMM:8	4

b7							b0	b7							b0	b7							b0	src
1	1	1	1	1	1	0	1	0	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	#IMM:8

TRMVFC TRMVFC

【コードサイズ】

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

(1) TRMVFC src, dest

b7							b0	b7							b0	b7								b0
1	1	1	1	1	1	0	1	0	1	0	1	1	0	1	0	cr[3:0]				rd[3:0]				

cr[3:0]	src	
0000b	Rx	(予約)
0001b		(予約)
0010b		(予約)
0011b		(予約)
0100b		(予約)
0101b		IVC
0110b		(予約)
0111b		(予約)
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		(予約)
1100b		INTB
1101b		EXTB
1110b		TRCTB
1111b		WINTB

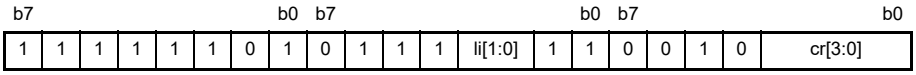
rd[3:0]	dest	
0000b～1111b	Rd	R0(SP)～R15

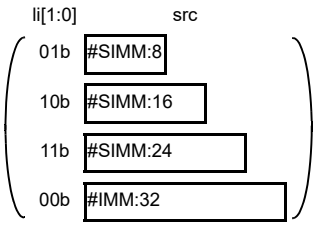
TRMVTC TRMVTC

【コードサイズ】

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

(1) TRMVTC src, dest

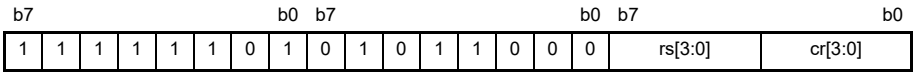




li[1:0]	src
01b	#SIMM:8
10b	#SIMM:16
11b	#SIMM:24
00b	#IMM:32

cr[3:0]	dest	
0000b	Rx	(予約)
0001b		(予約)
0010b		(予約)
0011b		(予約)
0100b		(予約)
0101b		IVC
0110b		(予約)
0111b		(予約)
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		(予約)
1100b		INTB
1101b		EXTB
1110b		TRCTB
1111b		WINTB

(2) TRMVTC src, dest



b7							b0	b7							b0	b7				b0
1	1	1	1	1	1	0	1	0	1	0	1	1	0	0	0	rs[3:0]				cr[3:0]

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

cr[3:0]	dest	
0000b	Rx	(予約)
0001b		(予約)
0010b		(予約)
0011b		(予約)
0100b		(予約)
0101b		IVC
0110b		(予約)
0111b		(予約)
1000b		BPSW
1001b		BPC
1010b		ISP
1011b		(予約)
1100b		INTB
1101b		EXTB
1110b		TRCTB
1111b		WINTB

TRRTE

TRRTE

【コードサイズ】

構文	コードサイズ（バイト）
TRRTE	2

b7							b0	b7							b0
0	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1

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。

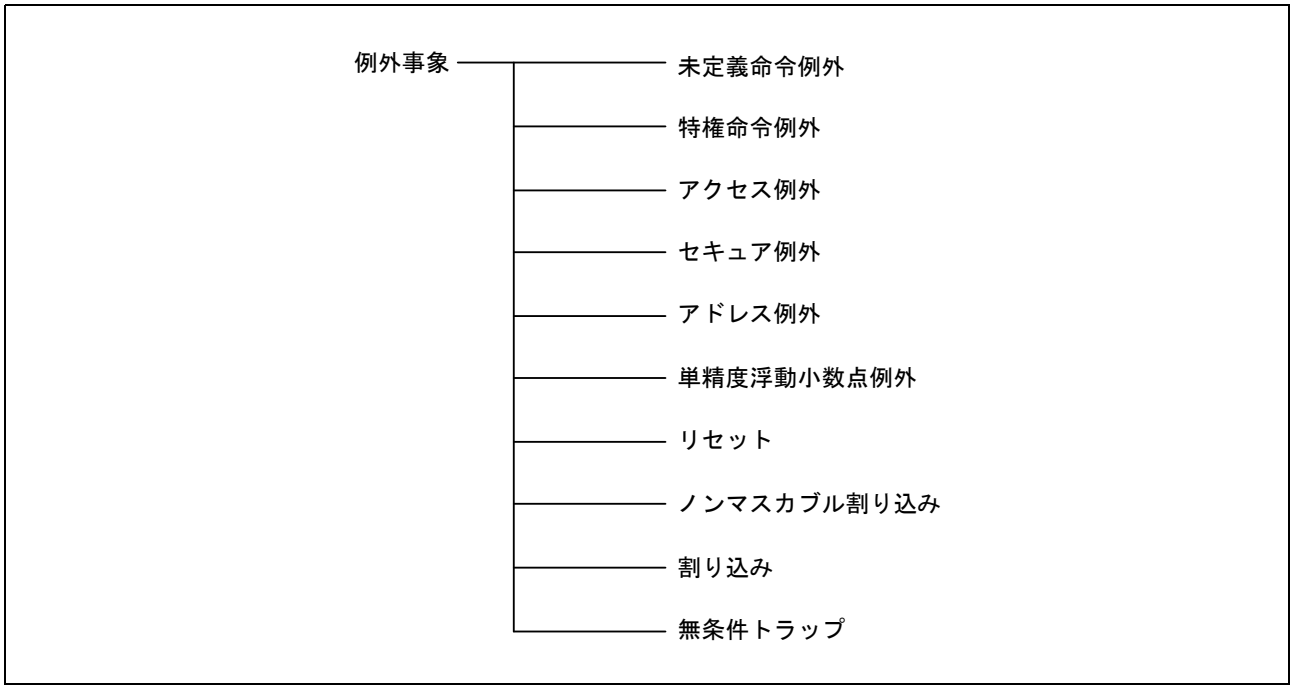


図 5.1　例外事象の種類

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。
Root ワールド専用命令を Root ワールド以外で実行した場合にも、本例外が発生します。

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おいてメモリプロテクションユニットがエラーを検出した場合に発生します。命令フェッチによって発生した場合は命令アクセス例外、オペランドアクセスによって発生した場合はオペランドアクセス例外がそれぞれ発生します。

5.1.4 セキュア例外

セキュア例外は、CPU からのメモリアクセスにおいてセキュリティ違反を検出した場合に発生します。命令フェッチによって発生した場合は命令セキュア例外、オペランドアクセスによって発生した場合オペランドセキュア例外がそれぞれ発生します。
セキュア例外が発生した場合、Root ワールドへ遷移します。
メモリアクセスにおいてセキュリティ違反を検出した場合でも、検出箇所、タイミングによっては CPU がセキュア例外を発生させない場合もあります。

5.1.5 アドレス例外

アドレス例外は、4 の倍数ではないアドレスに対して 64 ビットのオペランドアクセスを実行した場合に発生します。

5.1.6 単精度浮動小数点例外

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

5.1.7 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。

5.1.8 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使用することを想定しています。その場合、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。また、Flat-Project ではない場合、例外発生時に実行していた元のプログラムに復帰することはできません。

ノンマスカブル割り込みは、現在実行中のワールドによらずに受け付けます。

ノンマスカブル割り込みを受け付けた時の状況によって、受け付け時の動作は以下の 2 つに分かれます。

	受け付けた時の状況	ワールド遷移
通常受付	ノンマスカブル割り込みの宛先ワールドIDが現在実行中のワールドIDと同じ	なし
ワールド遷移受付	ノンマスカブル割り込みの宛先ワールドIDが現在実行中のワールドIDと異なる	Rootワールドへ遷移

通常受付の場合、実行中のワールドから他のワールドへは遷移しません。

ワールド遷移受付の場合、Root ワールドへ遷移します。Root ワールドのソフトウェアによりノンマスカブル割り込みの宛先ワールドへ遷移させてください。

5.1.9 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。

以下の表の条件を満たしたときのみ割り込みを受け付けられます。受け付け条件は現在実行中のワールドによって異なります。

表5.1　割り込み受付条件

現在実行中のワールド		
Root	Secure	Non-secure
• ITWがRoot かつ • PSW.Iが1 かつ • (PSW.IPL < IPR) かつ • (WPRI.WIPL < IPR)	• PSW.Iが1 かつ • (PSW.IPL < IPR) かつ • (WPRI.WIPL < IPR)	• PSW.Iが1 かつ • (PSW.IPL < IPR) かつ • (WPRI.WIPL < IPR) ただしIVC.NSIC = 1の場合、上記条件に加えて以下の条件でも受け付けます • (15 < IPR) かつ • (WPRI.WIPL < IPR)

IPR：割り込み要求の割り込み優先レベル
ITW：割り込み要求の宛先ワールド ID

割り込みを受け付けた時の状況によって、受け付け時の動作は以下の 2 つに分かれます。

	受け付けた時の状況	ワールド遷移
通常受付	割り込みの宛先ワールドIDが現在実行中のワールドIDと同じ	なし
ワールド遷移受付	割り込みの宛先ワールドIDが現在実行中のワールドIDと異なる	Rootワールドへ遷移

通常受付の場合、実行中のワールドから他のワールドへは遷移しません。

ワールド遷移受付の場合、Root ワールドへ遷移します。Root ワールドのソフトウェアにより割り込みの宛先ワールドへ遷移させてください。

Root ワールド、Secure ワールドを宛先とした割り込みの割り込み優先レベルは 1 ～ 31 が使用可能です。Non-secure ワールドを宛先とした割り込みの割り込み優先レベルは 1 ～ 15 が使用可能です。

高速割り込みは Flat-Project でのみ使用可能です。

5.1.10 無条件トラップ

INT 命令、BRK 命令、および TRCALL 命令を実行すると無条件トラップが発生します。

命令によって、受け付け時の動作は以下の 2 つに分かれます。

	命令	ワールド遷移
通常受付	INT命令、BRK命令	なし
ワールド遷移受付	TRCALL命令	Rootワールドへ遷移

5.2 例外の処理手順

例外の処理手順には、ワールド遷移なしの例外事象を処理する場合と、ワールド遷移ありの例外事象を処理する場合の 2 通りがあります。

各例外事象のワールド遷移有無を表 5.2 に示します。

表5.2 例外事象のワールド遷移有無

例外事象	ワールド遷移有無
未定義命令例外	なし
特権命令例外	なし
アクセス例外	なし
セキュア例外	あり
アドレス例外	なし
単精度浮動小数点例外	なし
リセット	—（常にRootワールドへ遷移）
ノンマスカブル割り込み（通常受付）	なし
ノンマスカブル割り込み（ワールド遷移受付）	あり
割り込み（通常受付）	なし
割り込み（ワールド遷移受付）	あり
無条件トラップ（通常受付）	なし
無条件トラップ（ワールド遷移受付）	あり

5.2.1　ワールド遷移なしの例外の処理手順

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。

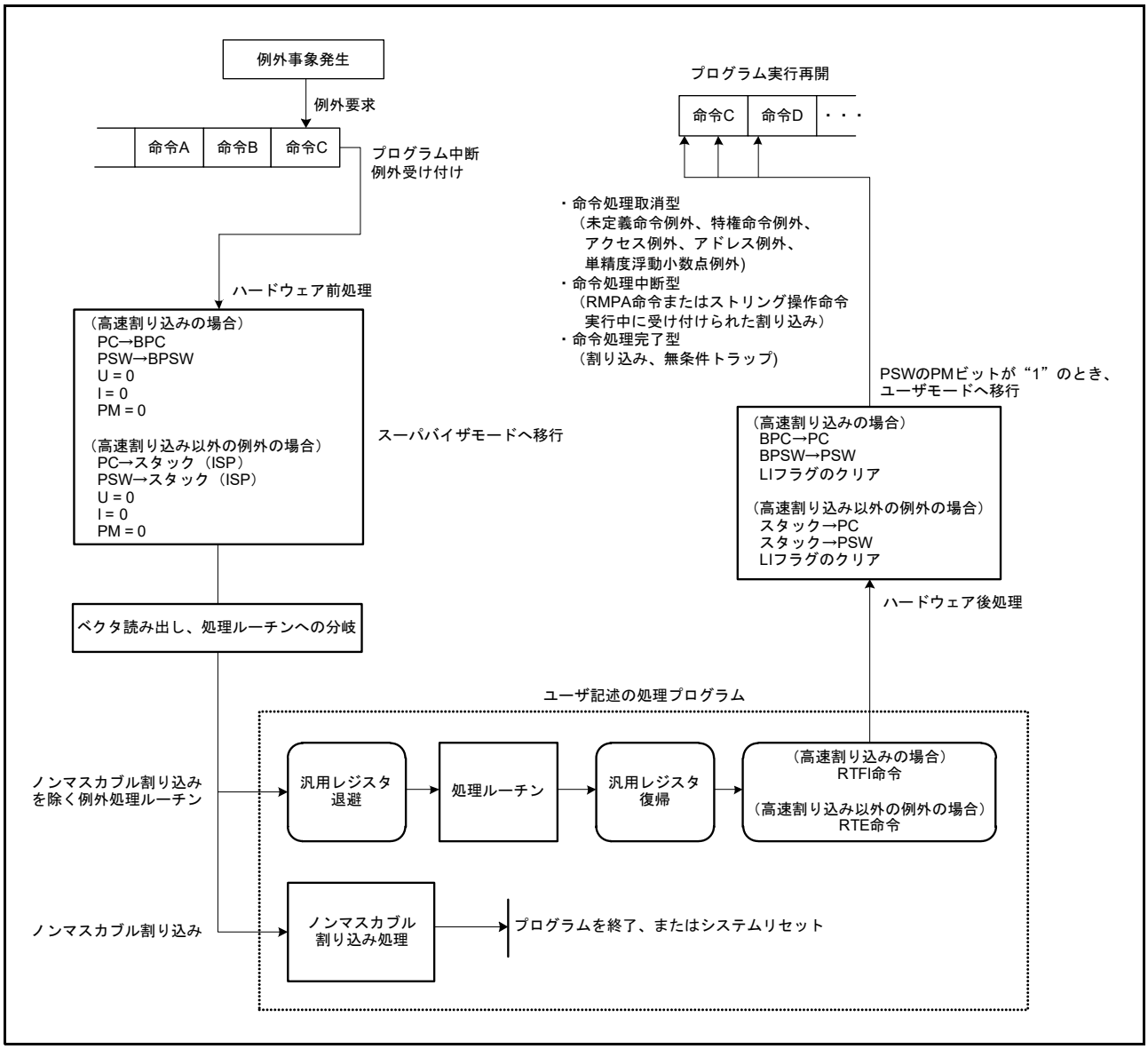


図 5.2　ワールド遷移なしの例外処理手順の概要

例外が受け付けられると、CPU はハードウェア処理を行った後、ベクタテーブルにアクセスし、分岐先アドレスを取得します。ベクタには例外ごとにベクタアドレスが割り当てられており、そこに例外処理ルーチンへの分岐先アドレスを書きます。

CPU のハードウェア前処理では、高速割り込みの場合は、プログラムカウンタ（PC）の内容をバックアップ PC（BPC）に、プロセッサステータスワード（PSW）の内容をバックアップ PSW（BPSW）へ退避させます。高速割り込み以外の例外では、PC、PSW をスタックに退避させます。例外処理ルーチン内で使用する汎用レジスタ、および PC、PSW 以外の制御レジスタについては、例外処理ルーチンの先頭でユーザプログラムによって退避させてください。

例外処理ルーチンの末尾で、退避させたレジスタを復帰させてから RTE 命令を実行することで、例外処理から元のプログラムに復帰します。高速割り込みの場合のみ、RTFI 命令を実行します。ノンマスカブル

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。

CPU のハードウェア後処理では、高速割り込みの場合は BPC を PC に、また、BPSW の値を PSW に戻します。高速割り込み以外の例外では、スタックから PC、PSW の値を復帰させます。

例外処理ルーチンの先頭・末尾で行う汎用レジスタ等の退避・復帰には、スタックまたはレジスタ退避バンクが利用できます。

(1) レジスタ退避バンクを搭載する製品では、SAVE・RSTR 命令を使用することでレジスタ退避バンクへの退避・復帰ができます。

(2) レジスタ退避バンクを搭載しない製品、またはレジスタ退避バンクを搭載していても SAVE・RSTR 命令の退避・復帰対象になっていないレジスタを退避・復帰させる場合には、PUSH・POP 等の命令を使用してスタックへの退避・復帰を行ってください。

なお、例外処理ルーチンで退避させるレジスタが極端に少ない場合を除き、レジスタ退避バンクを使用した方がスタックを使用する場合よりも高速に動作します。

5.2.2　ワールド遷移ありの例外の処理手順

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（Root ワールドの処理プログラム、例外事象宛先ワールドの処理プログラム、例外処理ルーチン）によって処理される部分があります。ワールド遷移ありの例外処理手順を図 5.3 に示します。

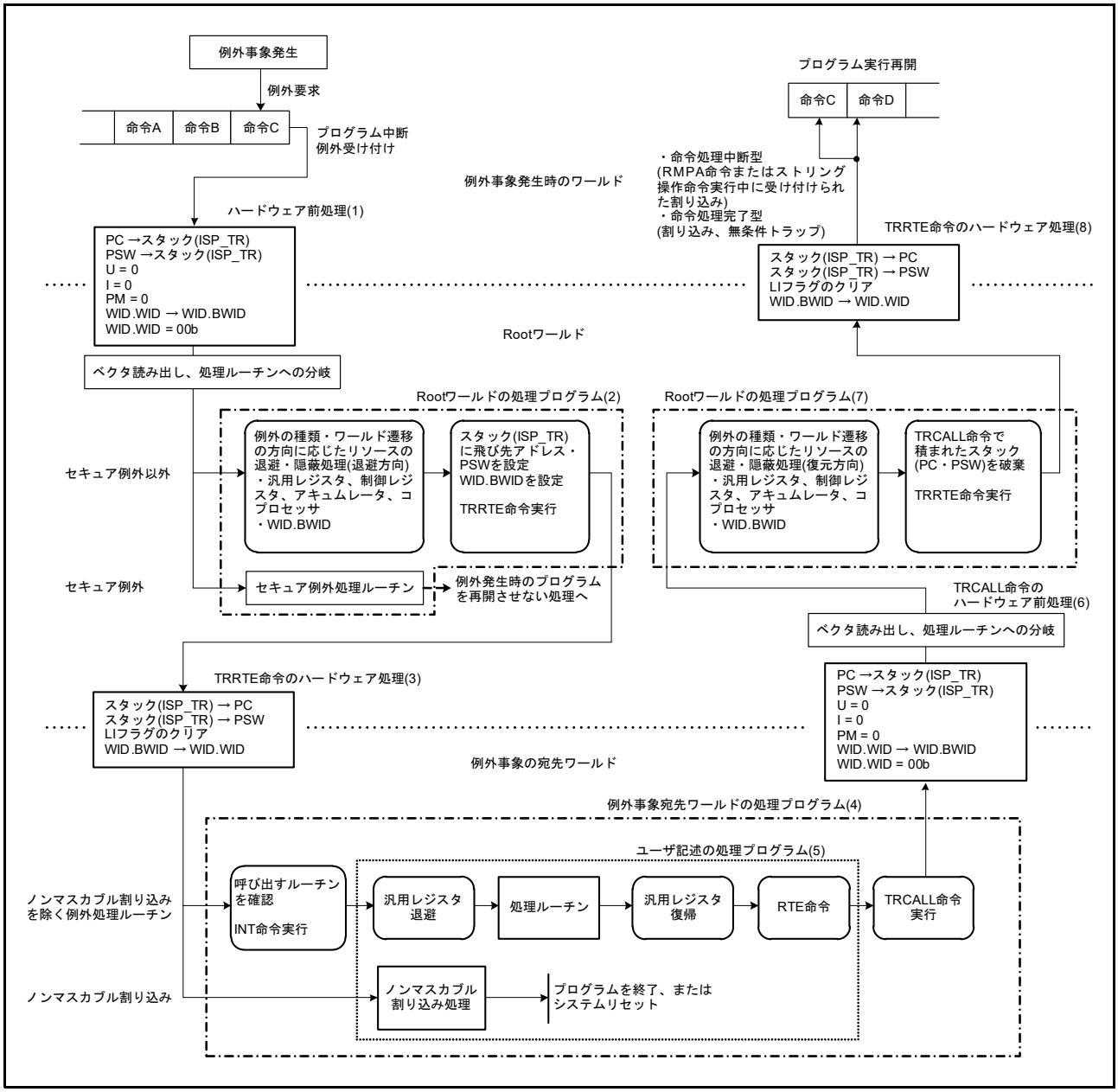


図 5.3　ワールド遷移ありの例外処理手順の概要

ワールド遷移ありの例外が受け付けられると、CPU はハードウェア前処理（1）を行い、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 Root ワールドの処理プログラム（2）への分岐先アドレスを書きます。

CPU のハードウェア前処理（1）では、PC、PSW をスタック（ISP_TR）に退避させます。また WID.WID が WID.BWID に退避され、WID.WID を “0” として Root ワールドへ遷移します。

Root ワールドの処理プログラム（2）では、セキュア例外の場合（言い換えると、元のプログラムを再開させない場合）とそれ以外の場合（言い換えると、Root ワールドを経由して別ワールドの例外処理ルーチンを実行した後、元のワールドの元のプログラムを再開させる場合）で処理の流れが異なります。セキュア

例外の場合、Root ワールドのセキュア例外処理ルーチンによって適切な処理を行ったあと、プログラムを終了させるか、例外発生時のプログラムを再開させない処理へ移って下さい。セキュア例外以外の場合、例外の種類とその例外によるワールド遷移の方向を考慮し、システムとして必要なリソース（汎用レジスタ・制御レジスタ・アキュムレータ・コプロセッサのレジスタ）の退避や隠蔽処理を行ってください。その後、WID.BWID の値を遷移先ワールドに合わせて設定、スタック（ISP_TR）に例外事象宛先ワールドの処理プログラム（4）の先頭アドレスと PSW を設定し TRRTE 命令を実行してください。

TRRTE 命令のハードウェア処理（3）では、例外事象宛先ワールドの処理プログラム（4）への遷移が行われます。

例外事象宛先ワールドの処理プログラム（4）では、ノンマスカブル割り込み以外の場合、INT 命令を使ってユーザ記述の処理プログラム（5）へ遷移させます。割り込みを処理する場合、IVC レジスタの値を INT 命令の引数とします。例外事象宛先ワールドでこのようにプログラムすることで（5）をワールド遷移無しの場合と共通にできます（ここでは処理手順の一例としてこの方法を示していますが、汎用レジスタ退避・復帰の処理を除いて共通化するようなプログラムとすることも可能です）。その後、Root ワールドの処理プログラム（7）を実行するための TRCALL 命令を実行してください。

TRCALL 命令のハードウェア前処理（6）では、PC、PSW をスタック（ISP_TR）に退避させます。また WID.WID が WID.BWID に退避され、WID.WID を “0” として Root ワールドへ遷移します。

Root ワールドの処理プログラム（7）では、（2）で退避した WID.BWID 含むリソースを復帰させ、直前の TRCALL 命令でスタック（ISP_TR）に積まれた PC・PSW を破棄した後、TRRTE 命令を実行してください。

TRRTE 命令のハードウェア処理（8）では、例外受け付け時点のプログラムを再開するポイントへの遷移が行われます。

上記では処理手順の流れについて説明しましたが、Root ワールドの処理プログラム（2）や（7）においては、例えば TRCALL 命令の呼び出し元ワールドなどが適切かどうかや、ワールド遷移の行き帰りの数に矛盾がないかどうかなど、必要とされるセキュリティ要件に応じたチェックを行ってください。

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 5.3** に示します。

割り込み、ノンマスカブル割り込み、無条件トラップについては、ワールド遷移受付か通常受付かにかかわらず以下の表の動作は変わりません。

表5.3　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
セキュア例外		命令処理取消型	命令実行中	例外が発生した命令のPC値
アドレス例外		命令処理取消型	命令実行中	例外が発生した命令のPC値
単精度浮動小数点例外		命令処理取消型	命令実行中	例外が発生した命令のPC値
リセット		命令処理放棄型	各マシンサイクル	なし
ノンマスカブル割り込み	RMPA、SCMPU、SMOVB、SMOVF、SMOVU、SSTR、SUNTIL、SWHILE命令実行中	命令処理中断型	命令実行中	実行中の命令のPC値
	上記以外の状態	命令処理完了型	命令の区切り	次の命令のPC値
割り込み	RMPA、SCMPU、SMOVB、SMOVF、SMOVU、SSTR、SUNTIL、SWHILE命令実行中	命令処理中断型	命令実行中	実行中の命令のPC値
	上記以外の状態	命令処理完了型	命令の区切り	次の命令のPC値
無条件トラップ		命令処理完了型	命令の区切り	次の命令のPC値

5.3.2 ベクタと PC、PSW の退避場所

各例外事象のベクタとプログラムカウンタ（PC）、プロセッサステータスワード（PSW）の退避場所を**表 5.4**、**表 5.5** に示します。

表5.4　ベクタとPC、PSWの退避場所（ワールド遷移なしの例外）

例外事象		ベクタ	PC、PSWの退避場所
未定義命令例外		実行中のワールドの例外ベクタテーブル	実行中のワールドのスタック
特権命令例外		実行中のワールドの例外ベクタテーブル	実行中のワールドのスタック
アクセス例外		実行中のワールドの例外ベクタテーブル	実行中のワールドのスタック
アドレス例外		実行中のワールドの例外ベクタテーブル	実行中のワールドのスタック
単精度浮動小数点例外		実行中のワールドの例外ベクタテーブル	実行中のワールドのスタック
リセット		FFFF FFFCh	なし
ノンマスカブル割り込み		実行中のワールドの例外ベクタテーブル	実行中のワールドのスタック
割り込み	高速割り込み	FINTV	RootワールドのBPC、BPSW
	高速割り込み以外	実行中のワールドの割り込みベクタテーブル	実行中のワールドのスタック
無条件トラップ		実行中のワールドの割り込みベクタテーブル	実行中のワールドのスタック

表5.5 ベクタとPC、PSWの退避場所（ワールド遷移ありの例外）

例外事象		ベクタ	PC、PSWの退避場所
ノンマスカブル割り込み		Rootの例外ベクタテーブル(EXTB_TR)	Rootのスタック(ISP_TR)
割り込み	高速割り込み	—	—
	高速割り込み以外	ワールド遷移割り込みベクタテーブル(WINTB)	Rootのスタック(ISP_TR)
セキュア例外		Rootの例外ベクタテーブル(EXTB_TR)	Rootのスタック(ISP_TR)
無条件トラップ		ルートコールベクタテーブル(TRCTB)	Rootのスタック(ISP_TR)

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。

5.4.1 ワールド遷移なしの例外の受け付け / 復帰時のハードウェア処理

(1) 例外受け付け時のハードウェア前処理

(a) PSW の退避

（高速割り込みの場合）
PSW → BPSW
（高速割り込み以外の例外の場合）
PSW →スタック

注 . FPSW の退避は、ハードウェア前処理では行われません。単精度浮動小数点演算命令を例外処理ルーチン内で使用する場合は、例外処理ルーチン内でスタックに退避させてください。

(b) PSW の PM、U、I ビットの更新

I： 0 にする
U： 0 にする
PM：0 にする

(c) PC の退避

（高速割り込みの場合）
PC → BPC
（高速割り込み以外の例外の場合）
PC →スタック

(d) PC に例外処理ルーチン分岐先アドレスをセット

各例外に対応したベクタを取得し分岐することにより、例外処理へ移行します。

(2) RTE 命令、RTFI 命令実行時のハードウェア後処理

(a) PSW の復帰

（高速割り込みの場合）
BPSW → PSW
（高速割り込み以外の例外の場合）
スタック→ PSW

(b) PC の復帰

（高速割り込みの場合）
BPC → PC
（高速割り込み以外の例外の場合）
スタック→ PC

(c) LI フラグのクリア処理

5.4.2 ワールド遷移ありの例外の受け付け / 復帰時のハードウェア処理

(1) 例外受け付け時のハードウェア前処理

(a) WID の退避

WID.WID → WID.BWID

(b) WID の更新

WID.WID：00b にする

(c) PSW の退避

PSW →スタック（ISP_TR）

(d) 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

(e) PC の退避

PC →スタック（ISP_TR）

(f) PC に例外処理ルーチン分岐先アドレスをセット

各例外に対応したベクタを取得し分岐することにより、例外処理へ移行します。

(2) TRRTE 命令実行時のハードウェア後処理

(a) WID の復帰

WID.BWID → WID.WID

(b) PSW の復帰

スタック（ISP_TR）→ PSW

(c) PC の復帰

スタック（ISP_TR）→ PC

(d) LI フラグのクリア処理

5.5 ハードウェア前処理

例外要求が受け付けられてから例外処理ルーチンが実行されるまでのハードウェア前処理について説明します。

5.5.1 未定義命令例外

(1) プロセッサステータスワード（PSW）の内容をスタック（ISP）に退避させます。
(2) PSW のプロセッサモード設定ビット（PM）、スタックポインタ指定ビット（U）、割り込み許可ビット（I）を “0” にします。
(3) プログラムカウンタ（PC）の内容をスタック（ISP）に退避させます。
(4) EXTB の値 + 0000005Ch 番地からベクタを取得します。
(5) 取得したベクタを PC にセットし、例外処理ルーチンへ分岐します。

5.5.2 特権命令例外

(1) プロセッサステータスワード（PSW）の内容をスタック（ISP）に退避させます。
(2) PSW のプロセッサモード設定ビット（PM）、スタックポインタ指定ビット（U）、割り込み許可ビット（I）を “0” にします。
(3) プログラムカウンタ（PC）の内容をスタック（ISP）に退避させます。
(4) EXTB の値 + 00000050h 番地からベクタを取得します。
(5) 取得したベクタを PC にセットし、例外処理ルーチンへ分岐します。

5.5.3 アクセス例外

(1) プロセッサステータスワード（PSW）の内容をスタック（ISP）に退避させます。
(2) PSW のプロセッサモード設定ビット（PM）、スタックポインタ指定ビット（U）、割り込み許可ビット（I）を “0” にします。
(3) プログラムカウンタ（PC）の内容をスタック（ISP）に退避させます。
(4) EXTB の値 + 00000054h 番地からベクタを取得します。
(5) 取得したベクタを PC にセットし、例外処理ルーチンへ分岐します。

5.5.4 セキュア例外

(1) WID.WID を WID.BWID に退避させます。
(2) WID.WID を “0” にし、Root ワールドに遷移します。
(3) プロセッサステータスワード（PSW）の内容をスタック（ISP_TR）に退避させます。
(4) 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。
(5) プログラムカウンタ（PC）の内容をスタック（ISP_TR）に退避させます。
(6) EXTB の値 + 00000008h 番地からベクタを取得します。
(7) 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 アドレス例外

(1) プロセッサステータスワード（PSW）の内容をスタック（ISP）に退避させます。
(2) PSW のプロセッサモード設定ビット（PM）、スタックポインタ指定ビット（U）、割り込み許可ビット（I）を “0” にします。
(3) プログラムカウンタ（PC）の内容をスタック（ISP）に退避させます。
(4) EXTB の値 + 00000060h 番地からベクタを取得します。
(5) 取得したベクタを PC にセットし、例外処理ルーチンへ分岐します。

5.5.6 単精度浮動小数点例外

(1) プロセッサステータスワード（PSW）の内容をスタック（ISP）に退避させます。
(2) PSW のプロセッサモード設定ビット（PM）、スタックポインタ指定ビット（U）、割り込み許可ビット（I）を “0” にします。
(3) プログラムカウンタ（PC）の内容をスタック（ISP）に退避させます。
(4) EXTB の値 + 00000064h 番地からベクタを取得します。
(5) 取得したベクタを PC にセットし、例外処理ルーチンへ分岐します。

5.5.7 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。

5.5.8 ノンマスカブル割り込み（通常受付）

(1) プロセッサステータスワード（PSW）の内容をスタック（ISP）に退避させます。
(2) PSW のプロセッサモード設定ビット（PM）、スタックポインタ指定ビット（U）、割り込み許可ビット（I）を “0” にします。
(3) RMPA、SCMPU、SMOVB、SMOVF、SMOVU、SSTR、SUNTIL、SWHILE 命令の実行中は、実行中の命令のプログラムカウンタ（PC）の内容を、それ以外の状態では次の命令の PC の内容をスタック（ISP）に退避させます。
(4) PSW のプロセッサ割り込み優先レベル（IPL[4:0]）を、Non-secure ワールドで命令を実行中の場合は “0Fh” に、それ以外の場合は “1Fh” 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。

5.5.9 ノンマスカブル割り込み（ワールド遷移受付）

(1) WID.WID を WID.BWID に退避させます。
(2) WID.WID を “0” にし、Root ワールドに遷移します。
(3) ノンマスカブル割り込みの宛先ワールドを WID.BITW に格納します。
(4) プロセッサステータスワード（PSW）の内容をスタック（ISP_TR）に退避させます。
(5) 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。

(6) RMPA、SCMPU、SMOVB、SMOVF、SMOVU、SSTR、SUNTIL、SWHILE 命令の実行中は、実行中の命令のプログラムカウンタ（PC）の内容を、それ以外の状態では次の命令の PC の内容をスタック（ISP_TR）に退避させます。

(7) PSW のプロセッサ割り込み優先レベル（IPL[4:0]）を “1Fh” にします。

(8) EXTB の値 + 00000018h 番地からベクタを取得します。

(9) 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。

5.5.10 割り込み（通常受付）

(1) プロセッサステータスワード（PSW）の内容をスタック（ISP）に退避させます。高速割り込みの場合は、バックアップ PSW（BPSW）に退避させます。

(2) PSW のプロセッサモード設定ビット（PM）、スタックポインタ指定ビット（U）、割り込み許可ビット（I）を “0” にします。

(3) RMPA、SCMPU、SMOVB、SMOVF、SMOVU、SSTR、SUNTIL、SWHILE 命令の実行中は、実行中の命令のプログラムカウンタ（PC）の内容を、それ以外の状態では次の命令の PC の内容をスタック（ISP）に退避させます。高速割り込みの場合は、バックアップ PC（BPC）に退避させます。

(4) PSW のプロセッサ割り込み優先レベル（IPL[4:0]）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。

5.5.11 割り込み（ワールド遷移受付）

(1) WID.WID を WID.BWID に退避させます。

(2) WID.WID を “0” にし、Root ワールドに遷移します。

(3) 割り込みの宛先ワールドを WID.BITW に格納します。

(4) Root ワールドと割り込み宛先ワールドの IVC.BWINM に割り込み要因番号を格納します。

(5) プロセッサステータスワード（PSW）の内容をスタック（ISP_TR）に退避させます。

(6) 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。

(7) RMPA、SCMPU、SMOVB、SMOVF、SMOVU、SSTR、SUNTIL、SWHILE 命令の実行中は、実行中の命令のプログラムカウンタ（PC）の内容を、それ以外の状態では次の命令の PC の内容をスタック（ISP_TR）に退避させます。

(8) PSW のプロセッサ割り込み優先レベル（IPL[4:0]）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。

(9) ワールド遷移割り込みベクタテーブルから割り込み宛先ワールドに対応するベクタを取得します。

(10)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。

5.5.12 無条件トラップ（通常受付）

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

(2) PSW のプロセッサモード設定ビット（PM）、スタックポインタ指定ビット（U）、割り込み許可ビット

（I）を “0” にします。

(3) 次の命令のプログラムカウンタ（PC）の内容をスタック（ISP）に退避させます。

(4) INT 命令の場合は、割り込みベクタテーブルから INT 命令番号に対応したベクタを取得します。
BRK 命令の場合は、割り込みベクタテーブルの先頭番地からベクタを取得します。

(5) 取得したベクタを PC にセットし、例外処理ルーチンへ分岐します。

5.5.13 無条件トラップ（ワールド遷移受付）

(1) WID.WID を WID.BWID に退避させます。

(2) WID.WID を “0” にし、Root ワールドに遷移します。

(3) PSW.I の値を WID.BWI に格納します。

(4) プロセッサステータスワード（PSW）の内容をスタック（ISP_TR）に退避させます。

(5) 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。

(6) 次の命令のプログラムカウンタ（PC）の内容をスタック（ISP_TR）に退避させます。

(7) ルートコールベクタテーブルから TRCALL 命令番号に対応したベクタを取得します。

(8) 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 5.6** に示す命令を実行すると、ハードウェア前処理でスタックまたはバックアップ PC（BPC）、バックアップ PSW（BPSW）に退避させ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）の内容が復帰します。

表5.6　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
セキュア例外		TRRTE (注1)
アドレス例外		RTE
単精度浮動小数点例外		RTE
リセット		復帰不可能
ノンマスカブル割り込み（通常受付）		RTE (注2)
ノンマスカブル割り込み（ワールド遷移受付）		TRRTE (注2)
割り込み（通常受付）	高速割り込み	RTFI
	高速割り込み以外	RTE
割り込み（ワールド遷移受付）	高速割り込み	—
	高速割り込み以外	TRRTE
無条件トラップ（通常受付）		RTE
無条件トラップ（ワールド遷移受付）		TRRTE

注1.　元のプログラムに復帰することはできません。
注2.　元のプログラムへの復帰は推奨しません。また、Flat-Projectではない場合、例外発生時に実行していた元のプログラムに復帰することはできません。

5.7 例外事象の優先順位

例外事象の優先順位を**表 5.7** 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5.7　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	命令セキュア例外
	6	未定義命令例外 特権命令例外
	7	無条件トラップ
	8	アドレス例外
	9	オペランドアクセス例外
	10	オペランドセキュア例外
低い	11	単精度浮動小数点例外

5.8 コプロセッサの例外事象

コプロセッサで発生する例外事象とその例外処理について説明します。

5.8.1 倍精度浮動小数点例外

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

割り込みコントローラのベクタ番号の割り当ては、各製品のユーザーズマニュアル ハードウェア編を参照してください。

索引

記号

+∞方向への丸め 28
–∞方向への丸め 28

数字

0方向への丸め 28

A

ACC0, ACC1 （アキュムレータ） 36

B

BITW[1:0]ビット（バックアップワールド遷移先
ワールドID） 32
BPC（バックアップPC） 26
BPSW（バックアップPSW） 26
BWID[1:0]ビット（バックアップワールドID） 32
BWINM[7:0]ビット
（バックアップワールド遷移割り込み要因番号） 30
BWIビット（バックアップワールド遷移元割り込み
許可ビット） 32

C

CEフラグ（非実装処理要因フラグ） 28
COフラグ（オーバフロー要因フラグ） 28
CUフラグ（アンダフロー要因フラグ） 28
CVフラグ（無効演算要因フラグ） 28
CXフラグ（精度異常要因フラグ） 28
CZフラグ（ゼロ除算要因フラグ） 28
Cフラグ（キャリーフラグ） 24

D

DCEフラグ（非実装処理要因フラグ） 54
DCMR（倍精度浮動小数点比較結果レジスタ） 55
DCOフラグ（オーバフロー要因フラグ） 54
DCUフラグ（アンダフロー要因フラグ） 54
DCVフラグ（無効演算要因フラグ） 54
DCXフラグ（精度異常要因フラグ） 54
DCZフラグ（ゼロ除算要因フラグ） 54
DDNビット（非正規化数の0フラッシュビット） 54
DECNT
（倍精度浮動小数点例外処理動作制御レジスタ） 56
DEOビット（オーバフロー例外処理許可ビット） 55
DEPC（倍精度浮動小数点例外プログラムカウンタ） 56
DEUビット（アンダフロー例外処理許可ビット） 55
DEVビット（無効演算例外処理許可ビット） 55
DEXビット（精度異常例外処理許可ビット） 55
DEZビット（ゼロ除算例外処理許可ビット） 55
DFOフラグ（オーバフローフラグ） 55
DFSフラグ（倍精度浮動小数点エラーサマリフラグ） 55
DFUフラグ（アンダフローフラグ） 55
DFVフラグ（無効演算フラグ） 55
DFXフラグ（精度異常フラグ） 55
DFZフラグ（ゼロ除算フラグ） 55
DNビット（非正規化数の0フラッシュビット） 28
DPSW（倍精度浮動小数点ステータスワード） 53
DRM[1:0]ビット
（倍精度浮動小数点丸めモード設定ビット） 54

E

EHMビット（倍精度浮動小数点例外発生情報保持
モードビット） 56
EHSビット（倍精度浮動小数点例外発生情報保持
ステータスビット） 56
EOビット（オーバフロー例外処理許可ビット） 29
EUビット（アンダフロー例外処理許可ビット） 29
EVビット（無効演算例外処理許可ビット） 29
EXCSEビット
（ハードウェア前処理中セキュア例外検出ビット） 31
EXTB（例外テーブルレジスタ） 29
EXビット（精度異常例外処理許可ビット） 29
EZビット（ゼロ除算例外処理許可ビット） 29

F

FINTV（高速割り込みベクタレジスタ） 26
FOフラグ（オーバフローフラグ） 29
FPSW（単精度浮動小数点ステータスワード） 27
FSフラグ（単精度浮動小数点エラーサマリフラグ） 29
FUフラグ（アンダフローフラグ） 29
FVフラグ（無効演算フラグ） 29
FXフラグ（精度異常フラグ） 29
FZフラグ（ゼロ除算フラグ） 29

I

INTB（割り込みテーブルレジスタ） 22
IPL[4:0]ビット（プロセッサ割り込み優先レベル） 25
ISP（割り込みスタックポインタ） 22
IVC（割り込みベクタ制御レジスタ） 30
Iビット（割り込み許可ビット） 25

N

NaN （Not a Number） 39
NSICビット
（ノンセキュア割り込み許可制御ビット） 31

O

Oフラグ（オーバフローフラグ）..... 25

P

PC（プログラムカウンタ）..... 22
PMビット（プロセッサモード設定ビット）..... 25
Protect Zone技術 17
PSW（プロセッサステータスワード）..... 23
PSW直接 67

Q

QNaN（Quiet NaN）..... 39

R

R0(SP)～R15（汎用レジスタ）..... 19
register(n) 79
register_num(Rn) 79
RESフラグ（倍精度浮動小数点比較命令結果フラグ）... 55
RM[1:0]ビット
（単精度浮動小数点丸めモード設定ビット）..... 28

S

SNaN（Signaling NaN）..... 39
Sフラグ（サインフラグ）..... 24

T

TRCTB（ルートコールベクタテーブルレジスタ）..... 33

U

UIR[1:0]ビット（ソフトウェア割り込み制限ビット）.... 31
USP（ユーザスタックポインタ）..... 22
Uビット（スタックポインタ指定ビット）..... 25

W

WID（ワールドIDレジスタ）..... 32
WID[1:0]ビット（ワールドID）..... 32
WINTB
（ワールド遷移割り込みベクタテーブルレジスタ）..... 33
WIPL[4:0]]ビット（ワールド共通プロセッサ
割り込み優先レベル）..... 34
WPRI（ワールド共通プロセッサ割り込み優先レベル
レジスタ）..... 34

Z

Zフラグ（ゼロフラグ）.....24

あ

アキュムレータ（ACC0, ACC1）.....36
アキュムレータ直接 68
アクセス例外 383
アドレス例外 383
アンダフローフラグ（DFUフラグ）.....55
アンダフローフラグ（FUフラグ）.....29
アンダフロー要因フラグ（CUフラグ）.....28
アンダフロー要因フラグ（DCUフラグ）.....54
アンダフロー例外処理許可ビット（DEUビット）.....55
アンダフロー例外処理許可ビット（EUビット）.....29

い

インデックス付きレジスタ間接 66

お

オーバフローフラグ（DFOフラグ）.....55
オーバフローフラグ（FOフラグ）.....29
オーバフローフラグ（Oフラグ）.....25
オーバフロー要因フラグ（COフラグ）.....28
オーバフロー要因フラグ（DCOフラグ）.....54
オーバフロー例外処理許可ビット（DEOビット）.....55
オーバフロー例外処理許可ビット（EOビット）.....29

き

キャリーフラグ（Cフラグ）.....24

こ

高速割り込みベクタレジスタ（FINTV）.....26

さ

最近値への丸め 28
サイズ拡張指定子 81
サイズ指定子 78
サインフラグ（Sフラグ）.....24

す

スーパバイザモード 40
スタックポインタ（R0(SP)）.....19
スタックポインタ指定ビット（Uビット）.....25
ストリング 43

せ

制御レジスタ 19
制御レジスタ直接 67
整数 42
精度異常フラグ（DFXフラグ） 55
精度異常フラグ（FXフラグ） 29
精度異常要因フラグ（CXフラグ） 28
精度異常要因フラグ（DCXフラグ） 54
精度異常例外処理許可ビット（DEXビット） 55
精度異常例外処理許可ビット（EXビット） 29
セキュア例外 383
ゼロ除算フラグ（DFZフラグ） 55
ゼロ除算フラグ（FZフラグ） 29
ゼロ除算要因フラグ（CZフラグ） 28
ゼロ除算要因フラグ（DCZフラグ） 54
ゼロ除算例外処理許可ビット（DEZビット） 55
ゼロ除算例外処理許可ビット（EZビット） 29
ゼロフラグ（Zフラグ） 24

そ

即値 64
ソフトウェア割り込み制限ビット（UIR[1:0]ビット） 31

た

単精度浮動小数点エラーサマリフラグ（FSフラグ） 29
単精度浮動小数点数 42
単精度浮動小数点ステータスワード（FPSW） 27
単精度浮動小数点丸めモード設定ビット
（RM[1:0]ビット） 28
単精度浮動小数点例外 37, 383

と

特権命令 40
特権命令例外 383

の

ノンセキュア割り込み許可制御ビット
（NSICビット） 31
ノンマスカブル割り込み 384

は

ハードウェア前処理中セキュア例外検出ビット
（EXCSEビット） 31
倍精度浮動小数点エラーサマリフラグ（DFSフラグ） 55
倍精度浮動小数点コプロセッサ 51
倍精度浮動小数点数 60
倍精度浮動小数点ステータスワード（DPSW） 53
倍精度浮動小数点制御レジスタ直接 68
倍精度浮動小数点データレジスタ直接 68
倍精度浮動小数点比較結果レジスタ（DCMR） 55
倍精度浮動小数点比較命令結果フラグ（RESフラグ） 55
倍精度浮動小数点丸めモード設定ビット
（DRM[1:0]ビット） 54
倍精度浮動小数点例外 57, 400
倍精度浮動小数点例外処理動作制御レジスタ
（DECNT） 56
倍精度浮動小数点例外発生情報保持ステータスビット
（EHSビット） 56
倍精度浮動小数点例外発生情報保持モードビット
（EHMビット） 56
倍精度浮動小数点例外プログラムカウンタ（DEPC） 56
バックアップPC （BPC） 26
バックアップPSW （BPSW） 26
バックアップワールドID （BWID[1:0]ビット） 32
バックアップワールド遷移先ワールドID
（BITW[1:0]ビット） 32
バックアップワールド遷移元割り込み許可ビット
（BWIビット） 32
バックアップワールド遷移割り込み要因番号
（BWINM[7:0]ビット） 30
汎用レジスタ（R0(SP)～R15） 19

ひ

非実装処理要因フラグ（CEフラグ） 28
非実装処理要因フラグ（DCEフラグ） 54
非正規化数の0フラッシュビット（DNビット） 28
非正規化数の0フラッシュビット（DDNビット） 54
ビット 43

ふ

プリデクリメントレジスタ間接 66
プログラムカウンタ（PC） 22
プログラムカウンタ相対 67
プロセッサステータスワード（PSW） 23
プロセッサモード 40
プロセッサモード設定ビット（PMビット） 25
プロセッサ割り込み優先レベル（IPL[4:0]ビット） 25

へ

ベクタテーブル 45

ほ

ポストインクリメントレジスタ間接 66

ま

丸め 28

み

未定義命令例外 383

む

無効演算フラグ（DFVフラグ） 55
無効演算フラグ（FVフラグ） 29
無効演算要因フラグ（CVフラグ） 28
無効演算要因フラグ（DCVフラグ） 54
無効演算例外処理許可ビット（DEVビット） 55
無効演算例外処理許可ビット（EVビット） 29
無条件トラップ 385

ゆ

ユーザスタックポインタ（USP） 22
ユーザモード 40

り

リセット 383

る

ルートコールベクタテーブル 47
ルートコールベクタテーブルレジスタ（TRCTB） 33

れ

例外事象の優先順位 399
例外テーブルレジスタ（EXTB） 29
例外ベクタテーブル 45
レジスタ一括退避機能 49
レジスタ間接 65
レジスタ相対 65
レジスタ直接 65

わ

ワールドID （WID[1:0]ビット） 32
ワールドIDレジスタ（WID） 32
ワールド共通プロセッサ割り込み優先レベル（WIPL[4:0]ビット） 34
ワールド共通プロセッサ割り込み優先レベルレジスタ（WPRI） 34
ワールド遷移割り込みベクタテーブル 47
ワールド遷移割り込みベクタテーブルレジスタ（WINTB） 33
割り込み 384
割り込み許可ビット（Iビット） 25
割り込みスタックポインタ（ISP） 22
割り込みテーブルレジスタ（INTB） 22
割り込みベクタ制御レジスタ（IVC） 30
割り込みベクタテーブル 46

改訂記録	RX ファミリ RXv3e 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
1.01	2026.08.06	—	初版発行

RXファミリ RXv3e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　2026年8月6日　　Rev.1.01

発行　　ルネサス エレクトロニクス株式会社
　　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

RXファミリ RXv3e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

R01US0745JJ0101