RJJ09B0132-0100Z

32

Software Manual

OPSP
ソフトウエアマニュアル

ルネサス 32 ビットオープンプラットフォームシンセサイザブルプロセッサ

Rev.1.00
発行：2004 年 3 月 1 日

ルネサス テクノロジ
www.renesas.com

安全設計に関するお願い

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

本資料ご利用に際しての留意事項

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

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

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

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

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

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

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

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

改訂記録	OPSPソフトウエアマニュアル

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2004.03.01	-	初版発行

レイアウトの都合上、このページは白紙です。

目次

第1章 CPUプログラミングモデル

第2章 命令セット

第3章 命令

付録

第1章
CPUプログラミングモデル

1.1 プロセッサモード

OPSP-CPUコア（以下OPSP-CPUと略します）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を提供します。プロセッサモードを用いて、リソース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に対してより高い権限を持っています。

EIT事象が発生すると、CPUはスーパバイザモードに移行します。EIT事象が発生する直前のプロセッサモードは、プロセッサ状態語レジスタ(PSW)のバックアップPM(BPM)ビットに記憶されます。RTE命令の実行により、BPMビットに記憶されたプロセッサモードに復帰します。

1.1.1 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「RTE命令」、「MVTC命令」、「SETPSW命令」、「CLRPSW命令」があります。

1.2 CPUレジスタ

OPSP-CPUコアには16本の汎用レジスタ、6本の制御レジスタ、2本のアキュムレータ及びプログラムカウンタがあります。アキュムレータは64ビット、その他のレジスタはすべて32ビット構成になっています。

1.3 汎用レジスタ

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

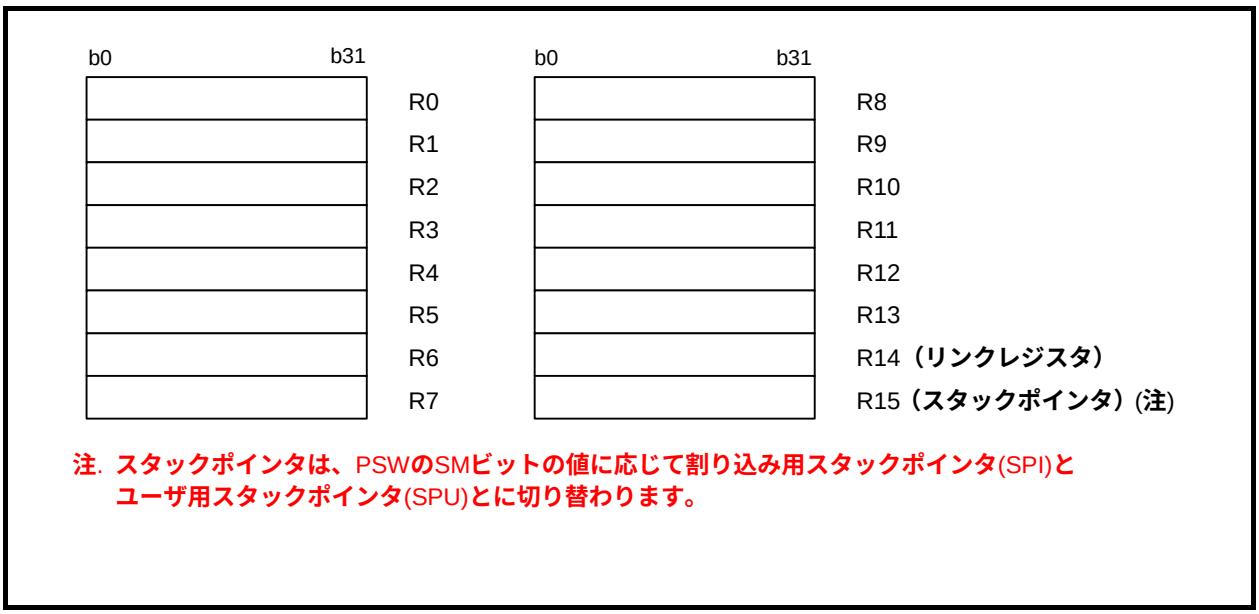


図1.3.1 汎用レジスタ

1.4 制御レジスタ

制御レジスタには、プロセッサ状態語レジスタ（PSW)、条件ビットレジスタ（C)、割り込み用スタックポインタ（SPI)、ユーザー用スタックポインタ（SPU)、EITベクタベースレジスタ(EVB)、バックアップPC（BPC）の６つがあります。

これら制御レジスタの設定や読み出しには、専用の「MVTC命令」と「MVFC命令」を使用します。また、PSWには関しては「SETPSW命令」と「CLRPSW命令」が使用できます。

「MVTC命令」、「SETPSW命令」および「CLRPSW命令」はプロセッサモード(PM)がスーパバイザモードのときのみ使用できる特権命令です。プロセッサモードは、プロセッサ状態語レジスタ(PSW)のプロセッサモード(PM)ビットで設定します。

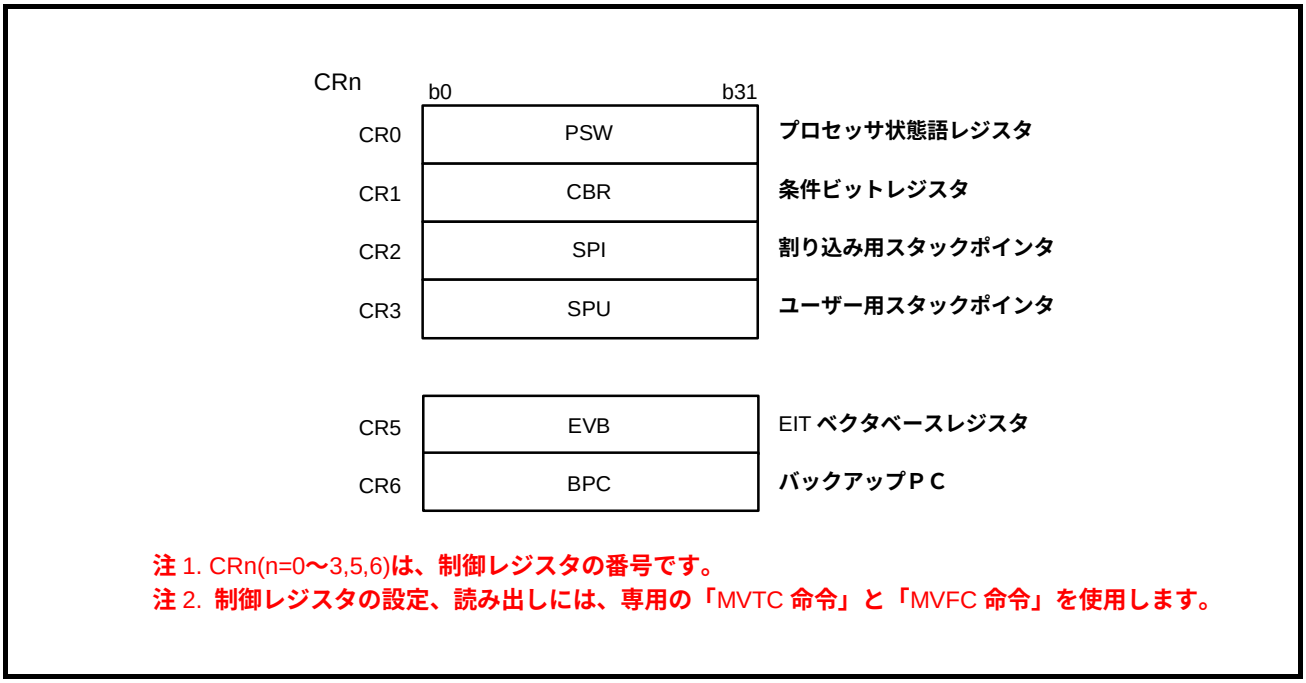


図1.4.1 制御レジスタ

1.4.1 プロセッサ状態語レジスタ：PSW（CR0）

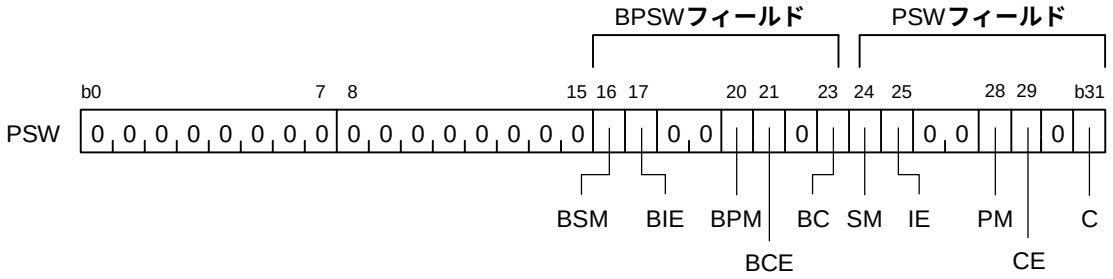
プロセッサ状態語レジスタ（PSW）は、OPSP-CPUのステータスを表示するレジスタで、通常使用するPSWフィールドと、EIT発生時にPSWフィールドを退避するためのBPSWフィールドからなります。

PSWフィールドは、スタックモード(SM)、割り込みイネーブル(IE)、プロセッサモード(PM)、コプロセッサ割り込みイネーブルビット(CE)、条件ビット(C)の各ビットで構成されています。

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

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

プロセッサモードを切り替える場合には、MVTC命令でBPM="1"を設定した後、RTE命令を実行してユーザ空間（H'0000 0000～H'7FFF FFFF領域）に分岐して下さい。なお、MVTC命令でPMビットを直接変更する場合は、必ずユーザ空間で行ってください。



＜リセット解除時："B'0000 0000 0000 0000 ??00 ??0? 0000 0000"**＞**

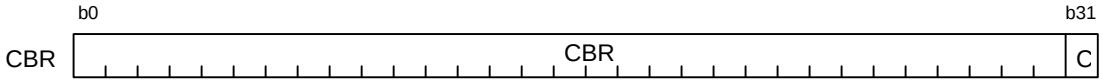
b	ビット名	機能	R	W
0~15	**何も配置されていません。"0"に固定してください。**		0	0
16	BSM **バックアップ**SM**ビット**	EIT**受付時に、**SM**ビットの値を保存**	R	W
17	BIE **バックアップ**IE**ビット**	EIT**受付時に、**IE**ビットの値を保存**	R	W
18,19	**何も配置されていません。"0"に固定してください。**		0	0
20	BPM **バックアップ**PM	EIT**受付時に、**PM**ビットの値を保存**	R	W
21	BCE **バックアップ**CE	EIT**受付時に、**CE**ビットの値を保存**	R	W
22	**何も配置されていません。"0"に固定してください。**		0	0
23	BC **バックアップ**C**ビット**	EIT**受付時に、**C**ビットの値を保存**	R	W
24	SM **スタックモードビット**	0：**割り込み用スタックポインタを使用** 1：**ユーザ用スタックポインタを使用**	R	W
25	IE **割り込みイネーブルビット**	0：**割り込み受付禁止** 1：**割り込み受付許可**	R	W
26,27	**何も配置されていません。"0"に固定してください。**		0	0
28	PM **プロセッサモードビット**	0：**スーパバイザモード** 1：**ユーザモード**	R	W
29	CE **コプロセッサ** **割り込みイネーブルビット**	0：**コプロセッサ割り込みを受け付けない** 1：**コプロセッサ割り込みを受け付ける**	R	W

30	**何も配置されていません。"0"に固定してください。**		0	0
31	C **条件ビット**	**命令の実行に応じて演算結果のキャリー、ボロー、オーバーフローの有無を示す。**	R	W

1.4.2 条件ビットレジスタ：CBR (CR1)

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

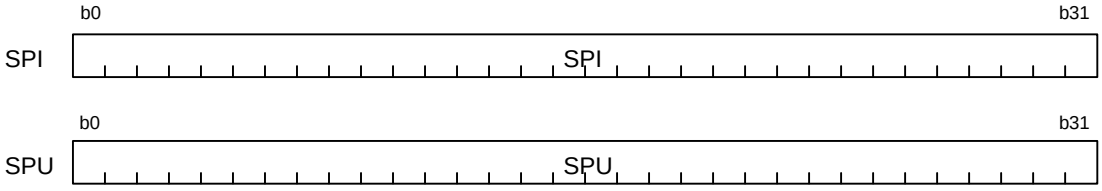
リセット解除時、CBRは"H'0000 0000"です。



1.4.3 割り込み用スタックポインタ：SPI (CR2)、ユーザー用スタックポインタ：SPU (CR3)

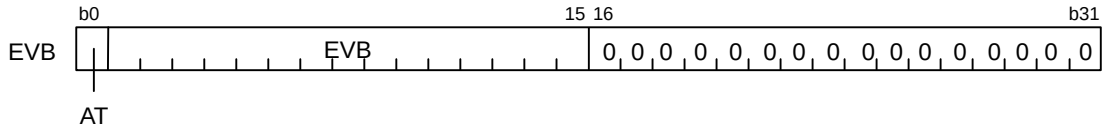
割り込み用スタックポインタ(SPI)、ユーザ用スタックポインタ(SPU)は、現在のスタックポインタのアドレスを保持します。これらのレジスタは、汎用レジスタR15としてアクセスできます。このとき、R15をSPIとして使用するかSPUとして使用するかは、PSWのスタックモードビット(SM)によって切り替わります。

リセット解除時、SPI、SPUは不定です。



1.4.4 EITベクタベースレジスタ :EVB(CR5)

EITベクタベースレジスタ(EVB)は、EITベクタエントリの先頭アドレスを保持します。EITベクタエントリの先頭アドレスの上位16ビットは、このレジスタの上位16ビットの値となります。



＜リセット解除時：H'0000 0000＞

b	ビット名	機能	R	W
0	AT **アドレス変換モードビット**	**アドレス変換モード**	R	N
1~15	EVB **ベクタベースビット**	EIT**ベクタエントリの**A1~A15**を設定してください。**	R	W
16~31	**何も配置されていません。**"0"**に固定してください。**		0	0

（1）AT（アドレス変換モード）ビット(b0)

このビットは、MATMレジスタのアドレス変換モードビット（AT）のコピーで読み出し専用ビットです。

（2）EVB（ベクタベースビット）ビット(b1~b15)

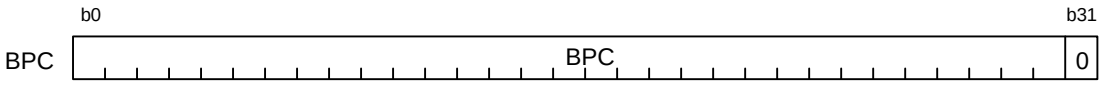
このビットにより、EITベクタエントリの先頭番地A1~A15を設定します。だし、リセット割り込み（RI）ベクタは、EITベクタベースビットの設定に関係なく、"H'0000 0000"に配置されます。

注. EVBレジスタは、リセット直後、1回のみ設定することができます。EVBレジスタへの書き込みは、リセットハンドラの先頭で行ってください。

1.4.5 バックアップPC：BPC (CR6)

バックアップPC（BPC）は、EIT発生時にプログラムカウンタ（PC）の値を退避するためのレジスタです。ビット31は0固定です。

EIT発生時には、発生したEITの種類により「EITの発生したPC値」または「次命令のPC値」がセットされ、「RTE命令」実行時にBPCの値はPCに戻されます。



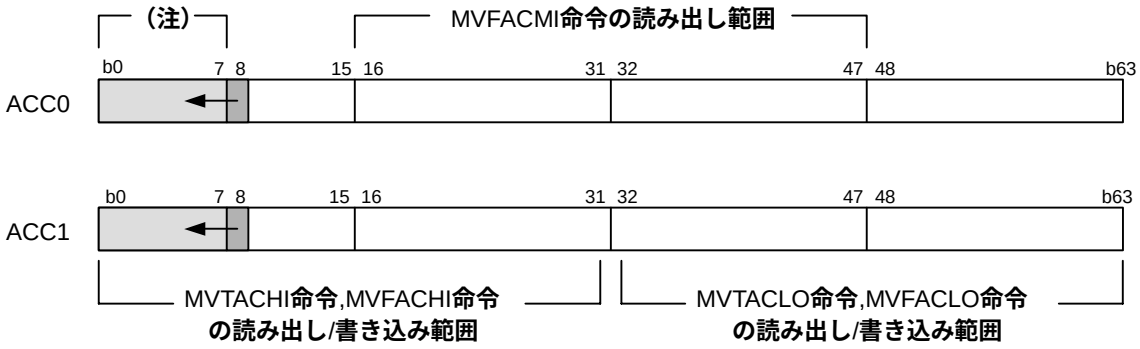
1.5 アキュムレータ

アキュムレータは、DSP機能用命令で使用される56ビットのレジスタでACC0、ACC1の2本あります。アキュムレータは、読み出し時や書き込み時には64ビットのレジスタとして扱われます。この時、アキュムレータのビット0～7の扱いは、読み出し時にはビット8の値を符号拡張し、書き込み時には無視します。また、アキュムレータは乗算命令「MUL」でも使用され、この命令実行の際はアキュムレータACC0,ACC1の値が破壊されるので注意してください。

アキュムレータへの書き込みには「MVTACHI命令」と「MVTACLO命令」を使用します。「MVTACHI命令」は上位側32ビット（ビット0～31）に、「MVTACLO命令」は下位側32ビット（ビット32～63）にデータを書き込みます。

読み出しには「MVFACHI命令」,「MVFACLO命令」,「MVFACMI命令」を使用します。「MVFACHI命令」で上位側32ビット（ビット0～31)、「MVFACLO命令」で下位側32ビット（ビット32～63)、「MVFACMI命令」で中央の32ビット（ビット16～47）のデータをそれぞれ読み出します。

リセット解除時、ACC0、ACC1は不定てす。

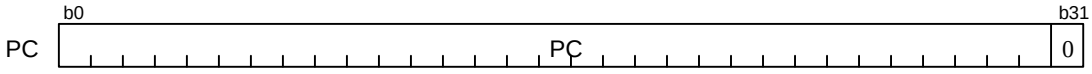


注. ビット0～7は、ビット8の値を符号拡張された値が常に読み出されます。この部分への書き込みは無視されます。

1.6 プログラムカウンタ(PC)

プログラムカウンタ（PC）は32ビットのカウンタで、現在実行中の命令アドレスを保持します。OPSP-CPUの命令は偶数アドレスから始まるため、LSB(ビット31)は"0"になります。

リセット解除時、PCは”H0000 0000”です。



1.7 データフォーマット

1.7.1 バイエンディアン機能

OPSP-CPUはデータフォーマットとして、ビッグエンディアンとリトルエンディアン、いずれかの方式をとることが可能なバイエンディアン機能をサポートしています。

このマニュアルのデータフォーマットはビッグエンディアン方式で記述しています。

1.7.2 データタイプ

OPSP-CPUの命令セットで扱えるデータタイプは、符号付き、または符号なしの 8、16、32ビット整数です。符号付き整数の値は2の補数で表現されます。

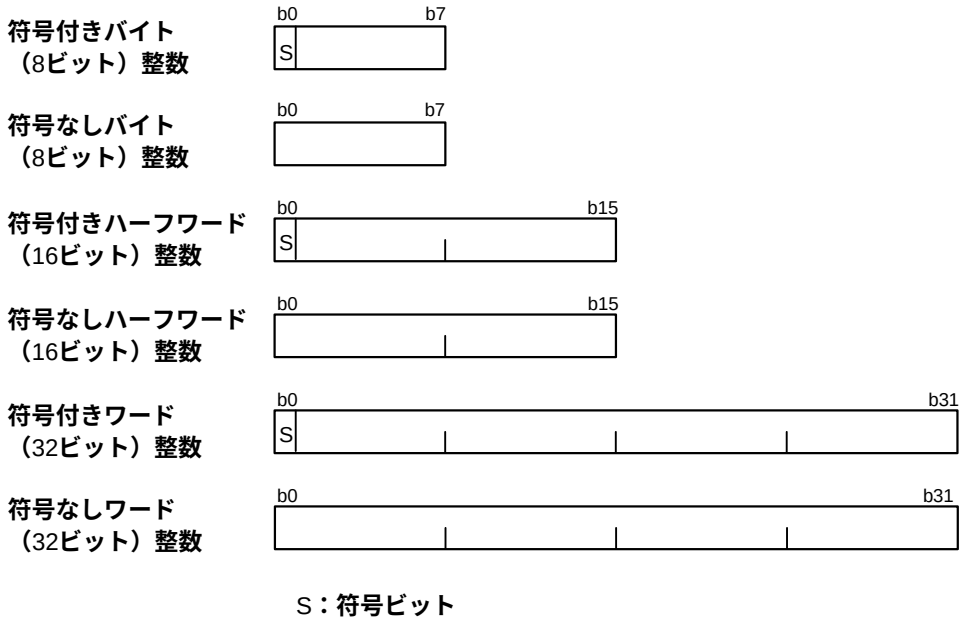


図1.7.1 データタイプ

1.7.3 データのフォーマット

(1) OPSP-CPUレジスタ上のデータフォーマット

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

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

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

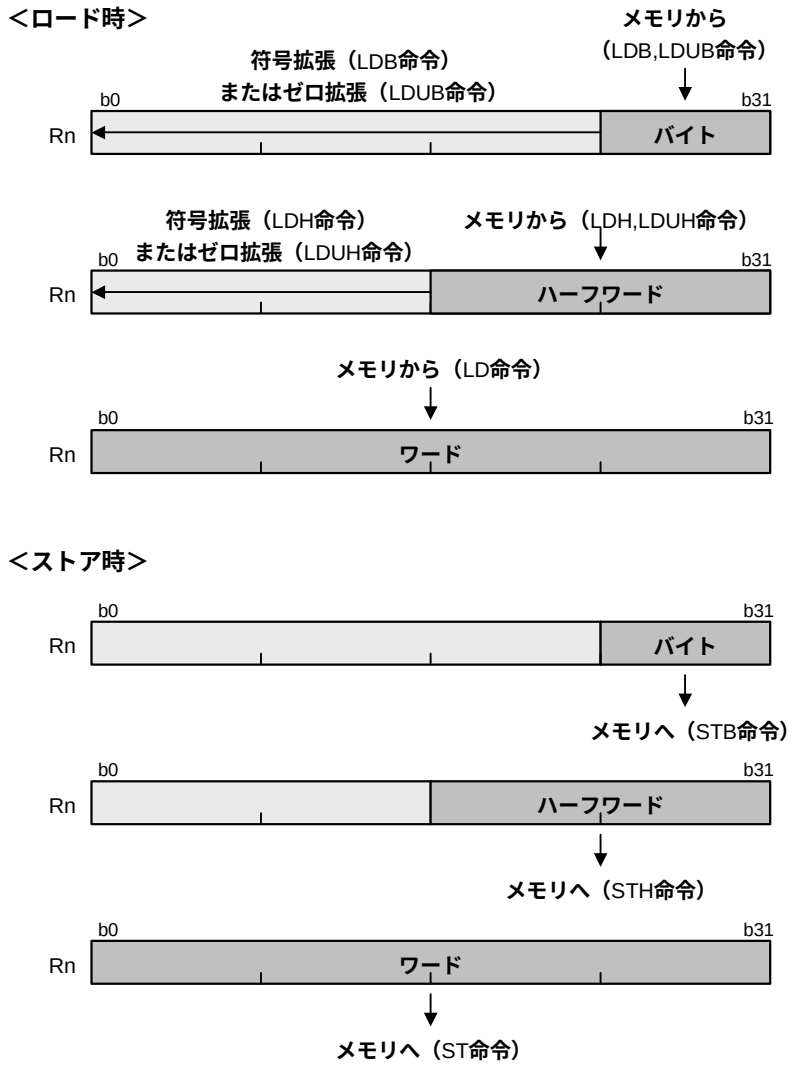


図1.7.2 レジスタ上のデータフォーマット

(2) **メモリ上のデータフォーマット**

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

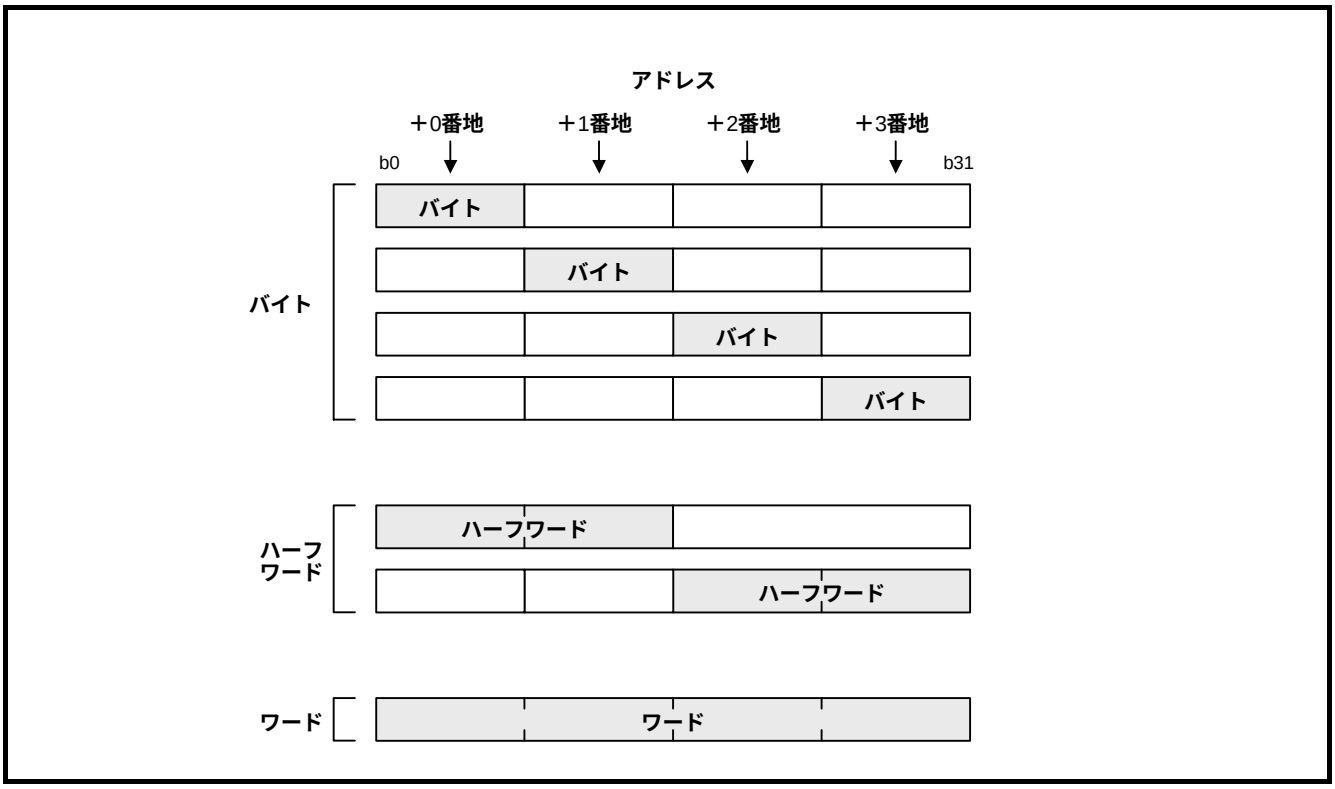


図1.7.3　**メモリ上のデータフォーマット**

1.8 アドレッシングモード

OPSP-CPUのアドレッシングモードには、以下のものがあります。

（1）レジスタ直接［ 表記：R,CRまたは、A(注) ］

操作の対象として汎用レジスタ、制御レジスタまたはアキュムレータを指定するもの。

（2）レジスタ間接［ 表記：@R ］

レジスタの値をアドレスとするもの（すべてのロード・ストア命令で指定可能）。

（3）レジスタ相対間接[表記：@(disp,R)]

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

（4）レジスタ間接+レジスタ更新

- レジスタ値を＋１する
  更新前のレジスタをアドレスとするもの（STB命令でのみ指定可能）
- レジスタ値を＋２する
  更新前のレジスタをアドレスとするもの（STH命令でのみ指定可能）
- レジスタ値を＋４する
  更新前のレジスタをアドレスとするもの（LD命令でのみ指定可能）
- レジスタ値を＋４する
  更新後のレジスタ値をアドレスとするもの（ST命令でのみ指定可能）
- レジスタ値を－４する
  更新後のレジスタ値をアドレスとするもの（ST命令でのみ指定可能）

（5）イミディエート[表記：#imm]

1,4,5,8,16または24ビットの即値（値の扱いは各命令の詳細説明参照）。

（6）PC相対[表記：pcdisp]

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

注．アキュムレータACC0,ACC1のニーモニックはA0,A1で表記します。

レイアウトの都合上、このページは白紙です。

第2章
命令セット

2.1 命令セット概要

OPSP-CPUの命令数は115です。OPSP-CPUはRISCアーキテクチャを採用しており、メモリアクセスは基本的にロード命令とストア命令で行います。また、各種の演算はレジスタ間演算で実行します。さらに、ロード＆アドレス更新、ストア＆アドレス更新といった複合命令もサポートしています。

2.2 命令セット

OPSP-CPU命令セットについて以下に示します。
M32Rファミリ命令セットから新たに追加された命令を”**” 、仕様拡張された命令を”*”で示します。

2.2.1ロード・ストア命令　(10命令)

メモリ～レジスタ間のデータ転送を行います。

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

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

（1）レジスタ間接

レジスタの値をアドレスとするもの（すべてのロード・ストア命令で指定可能）

（2）レジスタ相対間接

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

（3）レジスタ間接+レジスタ更新

- レジスタ値を＋１する
  更新前のレジスタをアドレスとするもの（STB命令でのみ指定可能）
- レジスタ値を＋２する
  更新前のレジスタをアドレスとするもの（STH命令でのみ指定可能）
- レジスタ値を＋４する
  更新前のレジスタをアドレスとするもの（LD命令でのみ指定可能）
- レジスタ値を＋４する
  更新後のレジスタ値をアドレスとするもの（ST命令でのみ指定可能）
- レジスタ値を－４する
  更新後のレジスタ値をアドレスとするもの（ST命令でのみ指定可能）

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

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

2.2.2 転送命令 (6命令)

レジスタ～レジスタ間またはレジスタ～イミディエート（即値）の転送を行います。

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

2.2.3 演算命令 (46命令)

レジスタ～レジスタ間で比較、算術論理演算、乗除算、シフトなどを行います。

■ **比較** (7命令)

	CMP	Compare
**	CMPEQ	Compare equal to
	CMPI	Compare immediate
	CMPU	Compare unsigned
	CMPUI	Compare unsigned immediate
**	CMPZ	Compare equal to zero
**	PCMPBZ	Parallel compare byte to zero

■ **算術演算** (10命令)

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

■ **論理演算** (7命令)

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

■ **乗除算** (13命令)

	DIV	Divide
**	DIVB	Divide byte
**	DIVH	Divide halfword
	DIVU	Divide unsigned
**	DIVUB	Divide unsigned byte
**	DIVUH	Divide unsigned halfword
	MUL	Multiply
	REM	Reminder
**	REMB	Reminder byte
**	REMH	Reminder halfword
	REMU	Reminder unsigned
**	REMUB	Reminder unsigned byte
**	REMUH	Reminder unsigned halfword

■ **シフト** (9命令)

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

2.2.4 分岐命令 (21命令)

プログラムの流れを変えるための命令です。

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

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

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

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

SC,SNC命令は、(分岐命令のPC値)+4が分岐先アドレスとなります。

その他の分岐命令は、(分岐命令のPC値)+(符号拡張されたイミディエート値を左に2ビットシフトした値)が分岐先アドレスとなります。ただし、加算が行われる際のPC値の下位2ビットは”00”にクリアされます。たとえば、図2.2.1で「命令A」または「命令B」が分岐命令で、「命令G」に分岐する場合、いずれもイミディエート値は4になります。

サブルーチンコール用のJL,BL,BCL,BNCL命令は、分岐と同時に戻り先PC値がR14に格納されます。R14に格納される値は(分岐命令のPC値+4)で、PC値の下位2ビットは”00”にクリアされます。
たとえば図2.2.1で「命令A」または「命令B」がJL,BL,BCL,BNCL命令あった場合、いずれも戻り先は「命令C」になります。

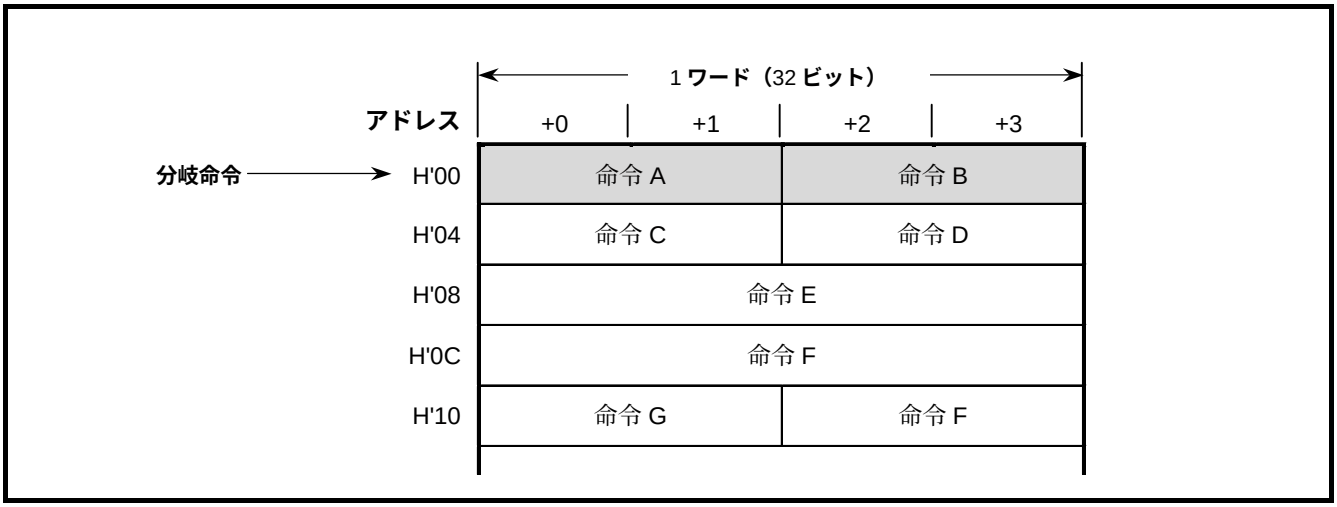


図2.2.1　分岐命令の分岐先

2.2.5 ビット操作命令 (5命令)

ビット操作命令は、メモリ内容、レジスタ、プロセッサ状態後レジスタ(PSW)のビット操作を行う命令です。

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

2.2.6 EIT関連命令 (2命令)

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

*	TRAP	Trap
*	RTE	Return from EIT

2.2.7 DSP機能命令 (22命令)

OPSP-CPUでは、M32Rファミリ命令セットのDSP機能命令に対して以下の拡張がされています。

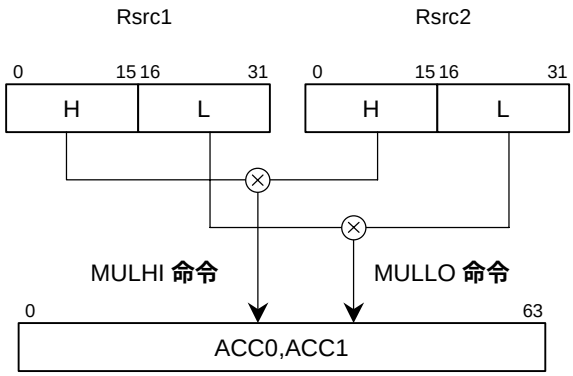
- **アキュムレータが1本から2本に強化**
- **積和演算の強化**
- **汎用レジスタ丸め命令を追加**

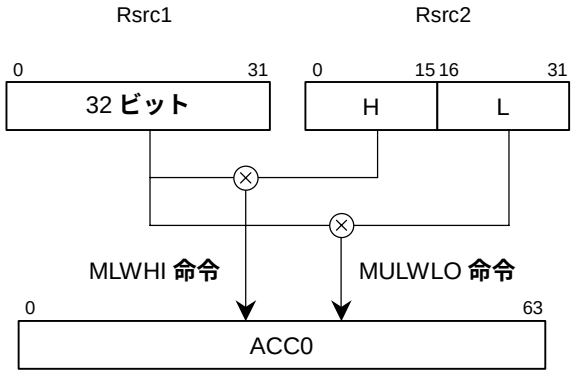
OPSP-CPUのDSP機能命令を示します。
M32Rファミリ命令セットから新たに追加された命令を”**” 、仕様拡張された命令を”*”で示します。

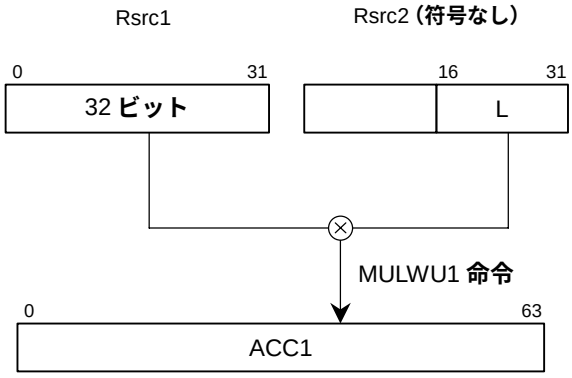
32ビット×16ビット、16ビット×16ビットの乗算や積和演算を行います。また、アキュムレータおよび汎用レジスタ内のデータ丸めやアキュムレータ～汎用レジスタ間の転送を行います。

**	MACHI	Multiply-accumulate high-order halfwords
**	MACLH1	Multiply-accumulate low-order halfword and high-order halfword using accumulator1
**	MACLO	Multiply-accumulate low-order halfwords
	MACWHI	Multiply-accumulate word and high-order halfword
	MACWLO	Multiply-accumulate word and low-order halfword
**	MACWU1	Multiply-accumulate word and unsigned low-order halfword using accumulator1
**	MSBLO	Multiply low-order halfwords and subtract
**	MULHI	Multiply high-order halfwords
**	MULLO	Multiply low-order halfwords
	MULWHI	Multiply word and high-order halfword
	MULWLO	Multiply word and low-order halfword
**	MULWU1	Multiply word and unsigned low-order halfword using accumulator1
	MVFACHI	Move high-order word from accumulator
	MVFACLO	Move low-order word from accumulator
	MVFACMI	Move middle-order word from accumulator
	MVTACHI	Move high-order word to accumulator
	MVTACLO	Move low-order word to accumulator
*	RAC	Round accumulator
*	RACH	Round accumulator halfword
**	SADD	Add accumulators
**	SATB	Saturate word into byte
**	SATH	Saturate word into halfword

次ページにこれらの命令の動作概要を示します。

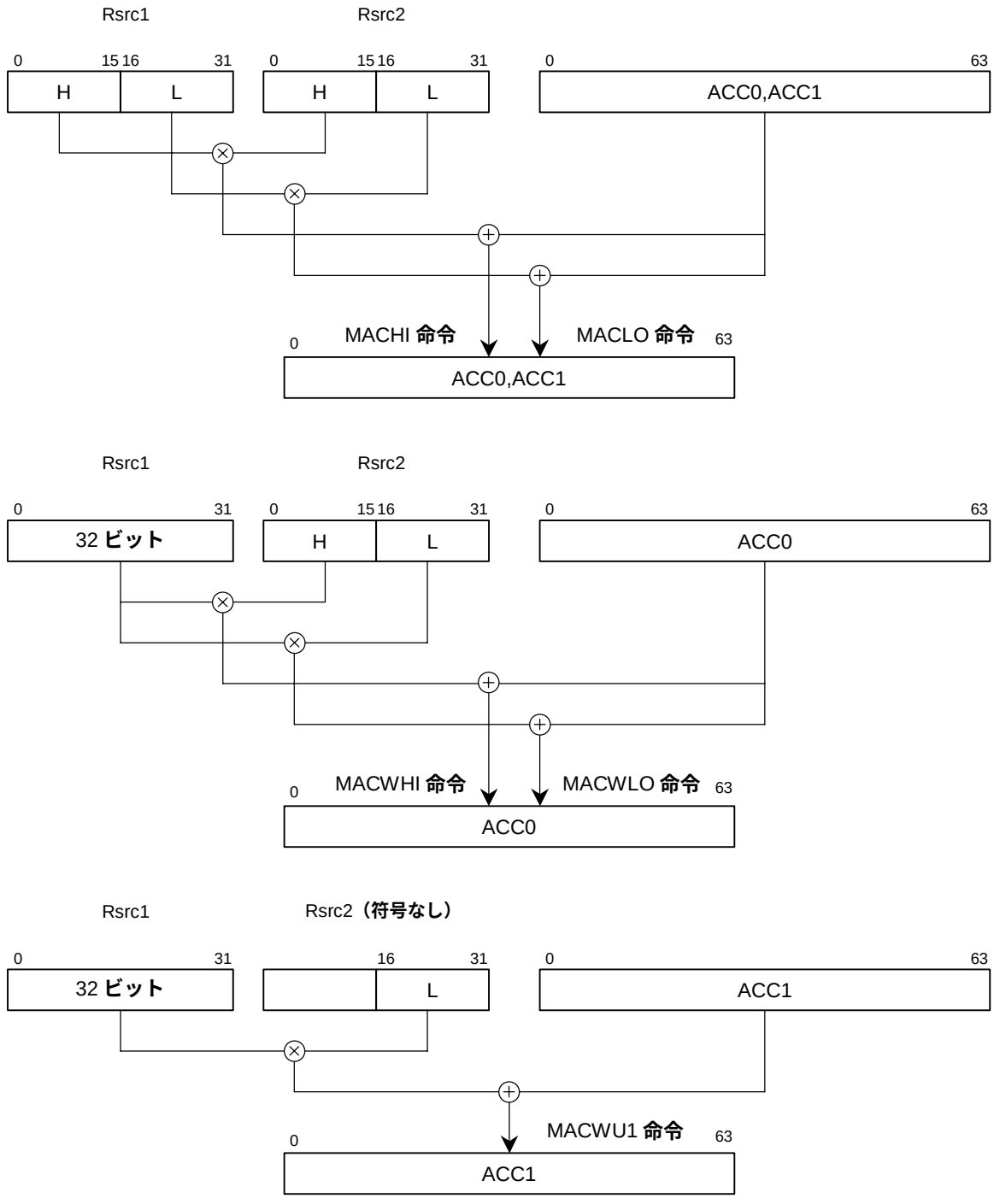






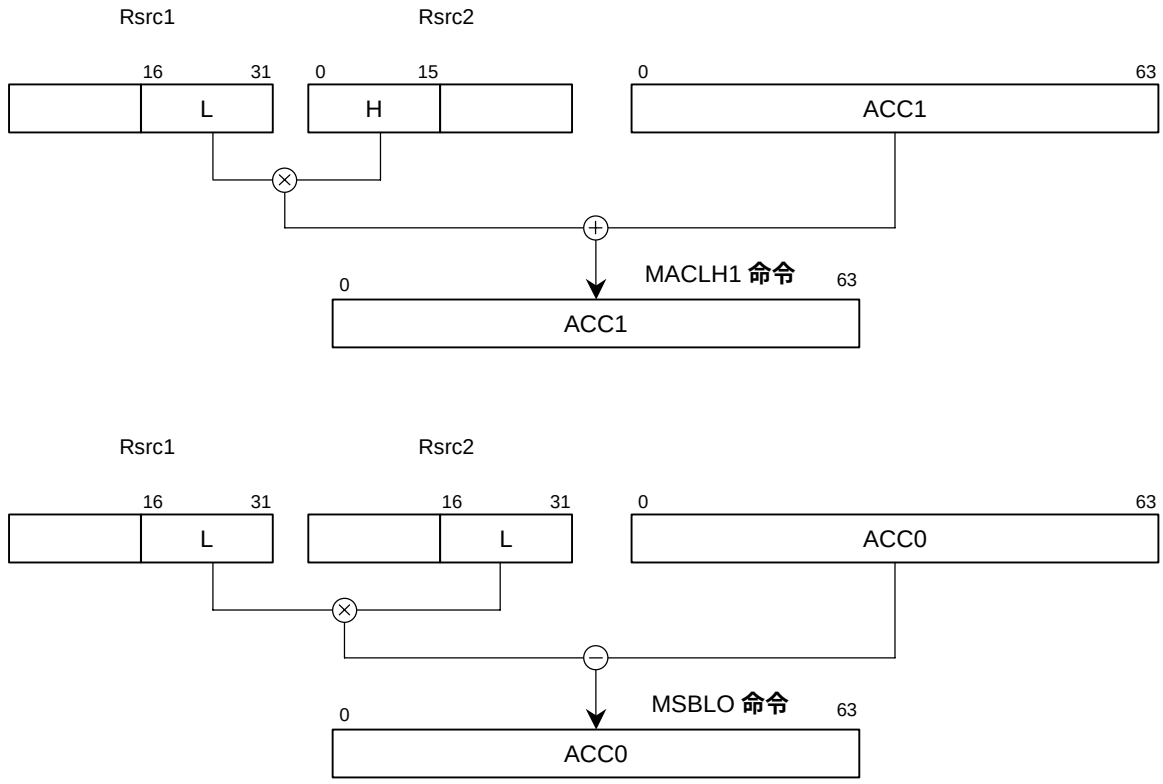
注．実際の DSP 機能命令の演算では結果の格納位置の調節や符号拡張が行われます。
詳しくは第 3 章「命令」を参照ください。

図2.2.2　DSP機能用命令の動作1(乗算)



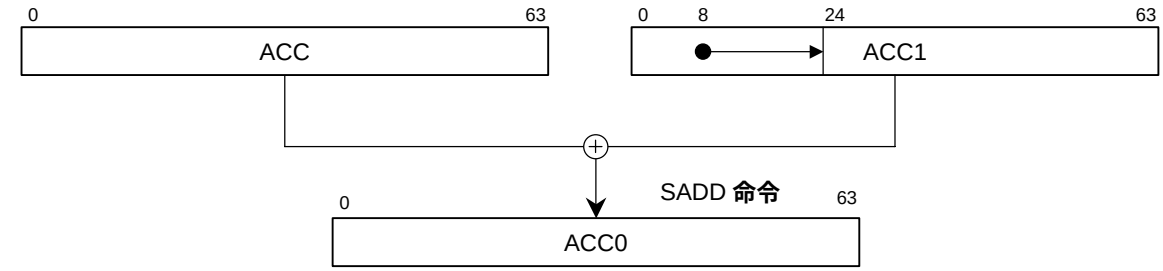
注．実際の DSP 機能命令の演算では結果の格納位置の調節や符号拡張が行われます。
詳しくは第 3 章「命令」を参照ください。

図2.2.3　DSP機能用命令の動作2(積和演算)



注．実際の DSP 機能命令の演算では結果の格納位置の調節や符号拡張が行われます。
詳しくは第 3 章「命令」を参照ください。

図2.2.4　DSP機能用命令の動作3(積和演算)

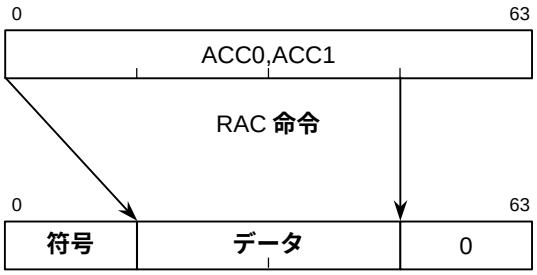


注．実際の DSP 機能命令の演算では結果の格納位置の調節や符号拡張が行われます。
詳しくは第 3 章「命令」を参照ください。

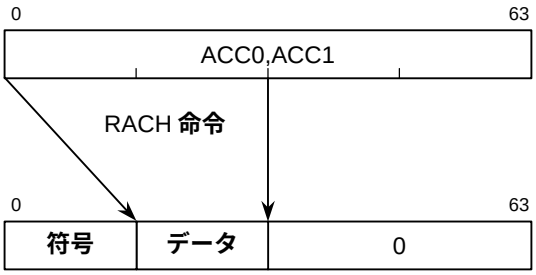
図2.2.5　DSP機能用命令の動作4(加算)

■ アキュムレータに対する丸め命令

＜ワードサイズの丸め＞

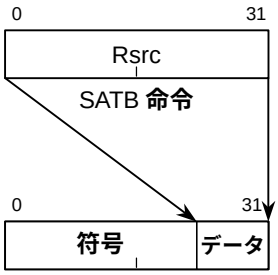


＜ハーフワードサイズの丸め＞



■ 汎用レジスタに対する丸め命令

＜バイトサイズの丸め＞



＜ハーフワードサイズの丸め＞

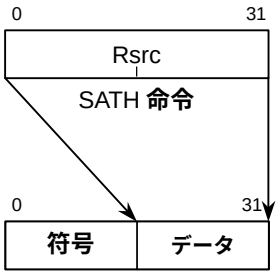


図2.2.6　DSP機能用命令の動作5　(丸め命令操作)

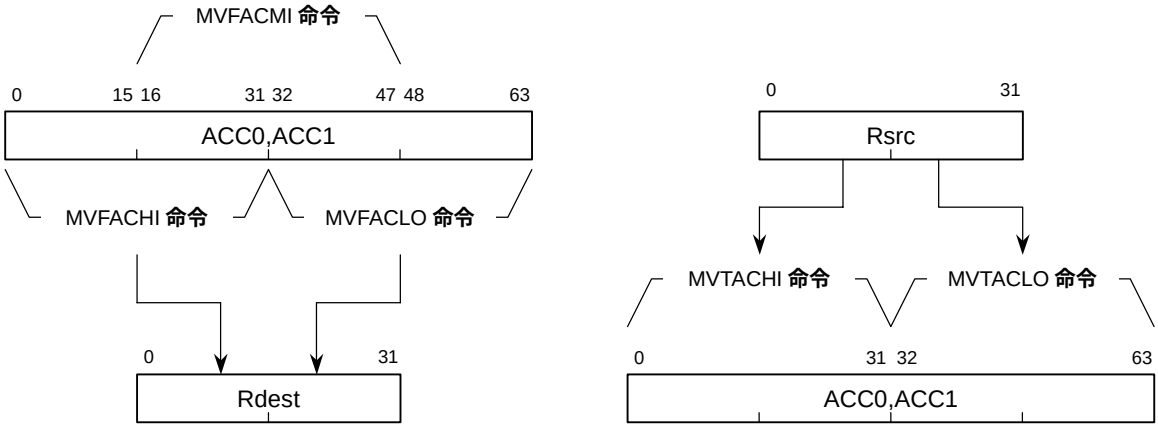


図2.2.7　7　DSP機能用命令の動作6　(アキュムレータ～レジスタ間転送)

2.2.8 コプロセッササポート命令 (3命令)

コプロセッサとのインタフェースに用いる命令を以下に示します。

*	MVTCP	Move to Coprocessor register
*	MVFCP	Move from Coprocessor register
*	OPECP	Operate Coprocessor

2.3 OPSP 拡張命令セット一覧

OPSP-CPU命令セットでは、M32Rファミリ命令セットから27命令を追加（新規拡張命令）し、21の命令について仕様を拡張（仕様拡張命令）しました。

2.3.1 OPSP-CPU新規拡張命令

表2.3.1に、M32Rファミリ命令セットから新規に追加された命令の一覧を示します。

表2.3.1 OPSP-CPU新規拡張命令一覧

分類	ニーモニック	機能概要
比較命令	CMPEQ	レジスタ間の比較
	CMPZ	レジスタと即値0との比較
	PCMPBZ	レジスタのバイト毎に即値0との比較
乗除算命令	DIVB	8ビット符号付き整数の除算
	DIVH	16ビット符号付き整数の除算
	DIVUB	8ビット符号なし整数の除算
	DIVUH	16ビット符号なし整数の除算
	REMB	8ビット符号付き整数の剰余
	REMH	16ビット符号付き整数の剰余
	REMUB	8ビット符号なし整数の剰余
	REMUH	16ビット符号なし整数の剰余
分岐命令	BCL	条件ビット(C)が1のとき分岐し、戻り先をR14に格納
	BNCL	条件ビット(C)が0のとき分岐し、戻り先をR14に格納
	JC	条件ビット(C)が1のとき分岐
	JNC	条件ビット(C)が0のとき分岐
	SC	条件ビット(C)が1のとき、並列実行ペアをスキップ
	SNC	条件ビット(C)が0のとき、並列実行ペアをスキップ
DSP機能命令	MACLH1	積和演算(レジスタ×レジスタ + アキュムレータA1 →アキュムレータA1)
	MACWU1	積和演算(レジスタ×レジスタ + アキュムレータA1 →アキュムレータA1)
	MSBLO	積和演算(アキュムレータA0 - レジスタ×レジスタ →アキュムレータA0)
	MULWU1	乗算(レジスタ×レジスタ→アキュムレータA1)
	SADD	加算(アキュムレータA0 + アキュムレータA1 →アキュムレータA0)
	SATB	レジスタのデータに対するバイトサイズへの丸め
	SATH	レジスタのデータに対するハーフワードサイズへの丸め
コプロセッササポート命令	MVTCP	コプロセッサのレジスタへの転送
	MVFCP	コプロセッサのレジスタからの転送
	OPECP	コプロセッサの操作

注．アキュムレータACC0,ACC1のニーモニックは、表中ではそれぞれA0,A1と表記しています。

2.3.2 OPSP-CPU仕様拡張命令

表2.3.2に、M32Rファミリ命令セットから仕様が拡張された命令の一覧を示します。

表2.3.2 OPSP-CPU仕様拡張命令一覧

分類	ニーモニック	仕様拡張内容
DSP機能命令	MACHI	オペランド表記でアキュムレータA0,A1を指定可能
	MACLO	
	MULHI	
	MULLO	
	MVFACHI	
	MVFACLO	
	MVFACMI	
	MVTACHI	
	MVTACLO	
	RAC	オペランド表記でアキュムレータA0,A1を指定可能 また、アキュムレータをイミディエート(imm1)で指定したビットの左シフト実行後の値に対して、丸め操作を行う
	RACH	
演算命令	SLL	並列実行の命令カテゴリを、左側命令(O－)から両側命令(OS)に変更 （命令カテゴリについては、2.7.3 16ビット命令のカテゴリー覧を参照してください。）
	SLLI	
	SRA	
	SRAI	
	SRL	
	SRLI	
ロードストア命令	STB	アドレッシングモードにレジスタ更新を追加
	STH	
EIT関連命令	TRAP	実行時のBPCの値が、BPC=PC+4からBPC=次命令PCに変更
	RTE	ハーフワード境界への復帰が可能

注．アキュムレータACC0,ACC1のニーモニックは、表中ではそれぞれA0,A1と表記しています。

2.4 命令フォーマット

OPSP-CPUの命令フォーマットは2種類あり、1つは32ビットワード境界内に格納された2つの16ビット命令、もう1つは単一の32ビット命令です。(図2.4.1)。

OPSP-CPUの基本命令フォーマットを図2.4.2に示します。

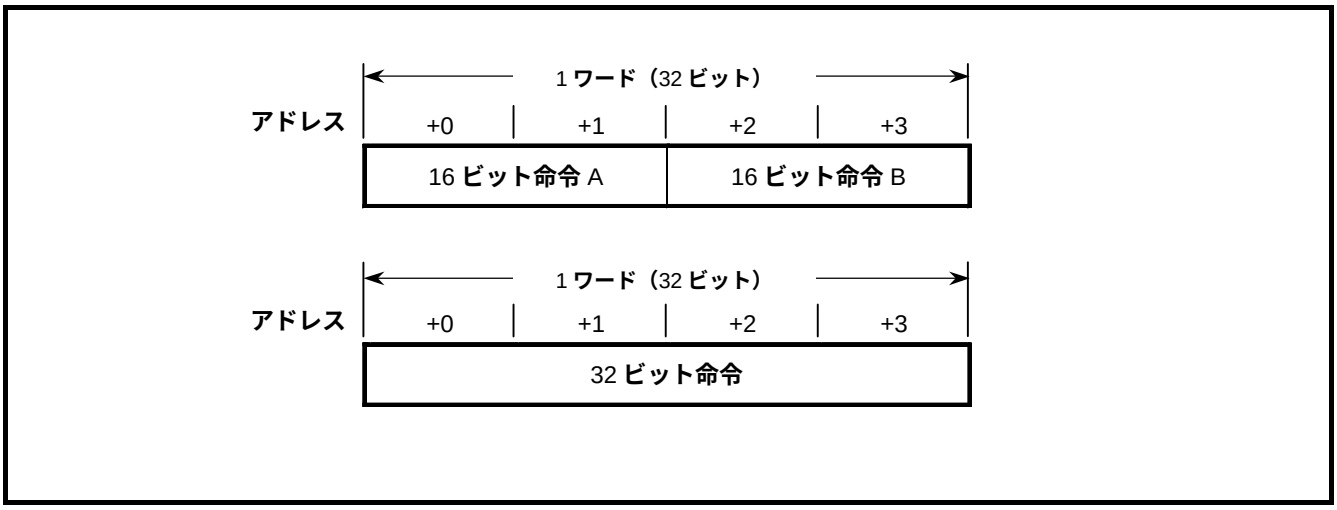


図2.4.1　16ビット命令と32ビット命令

■ 16 ビット命令

＜命令フォーマット＞				＜命令の動作＞	＜命令の例＞	
op1	R1	op2	R2	R1 = R1 op R2	AND	Rdest , Rsrc
op1	R1	c		R1 = R1 op c	ADD	Rdest , #imm8
op1	cond	c		Branch (Short Displacement)	BC	pcdisp8

■ 32 ビット命令

＜命令フォーマット＞					＜命令の動作＞	＜命令の例＞	
op1	R1	op2	R2	c	R1 = R2 op c	SRL3	Rdest , Rsrc , #imm16
op1	R1	op2	R2	c	Compare and Branch	BEQ	Rsrc1 , Rsrc2 ,
op1	R1	c			R1 = R1 op c	LD24	Rdest , #imm24
op1	cond	c			Branch	BC	pcdisp24

図2.4.2　基本命令フォーマット

2.5 並列実行処理

2.5.1 命令フォーマット

OPSP-CPU命令セットアーキテクチャでは、ワード境界内に2命令1組で配置された16ビット命令について、並列実行処理をサポートしています。並列実行処理がおこなわれるかどうかは、各16ビット命令のMSB(Most Significant Bit)の値によって決定されます（各命令のMSBは実行方法を決定するビットであり、命令の機能に影響を及ぼしません）。

16ビット命令の場合、上位ハーフワードに存在する命令のMSBは常に”0”になります。それにつづき16ビット命令のMSBが”0”の場合シーケンシャルに実行され、MSBが”1”の場合パラレルに実行されます。

図2.5.1において「命令B」のMSBが”0”の場合、「命令A」と「命令B」はシーケンシャルに実行されますが、「命令B」のMSBが”1”の場合は、「命令A」と「命令B」はパラレルに実行されます。
並列実行される場合の「命令B」は、アセンブラが自動的にMSBを1にした命令に変えます。ワードアラインメント調整のためのNOP命令は、必ずアセンブラがMSBを1にしたNOP命令に変えます。

32ビット命令のMSBは常に”1”となり、並列実行処理の対象とはなりません。

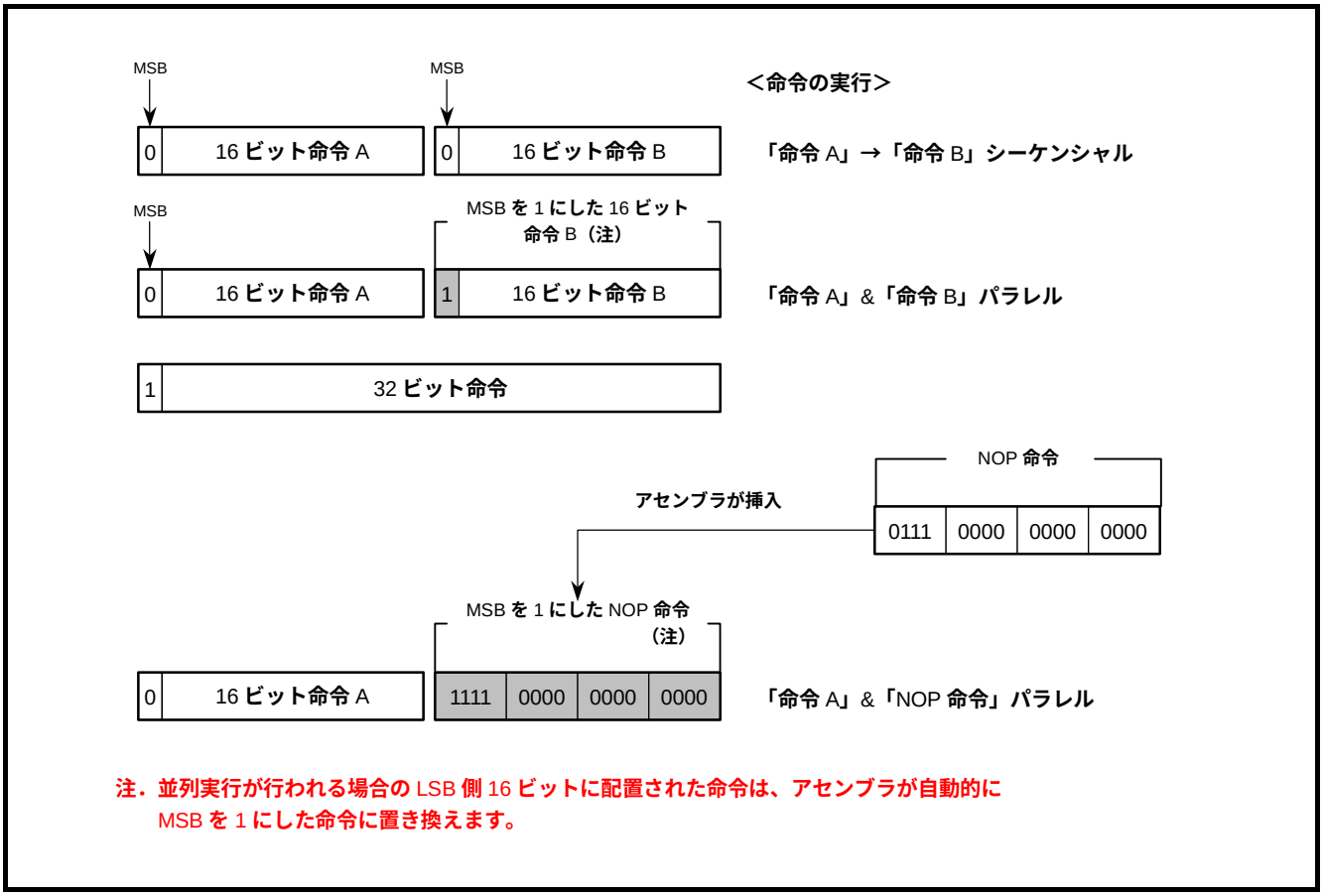


図2.5.1 命令の処理

2.5.2 OPSP 並列実行処理

OPSP-CPU はOパイプ、Sパイプの2本のパイプラインをもっています。この2本のパイプラインにより、2つの16ビット命令の並列実行を行います。

Sパイプでは、DSP機能用命令、乗算、算術演算、論理演算、シフト、比較、転送及びNOPの16ビット命令を、Oパイプでは、DSP機能用命令および乗算命令以外の16ビット命令を実行することができます。

なお、32ビット命令は並列処理の対象とならず、すべてOパイプで実行されます。

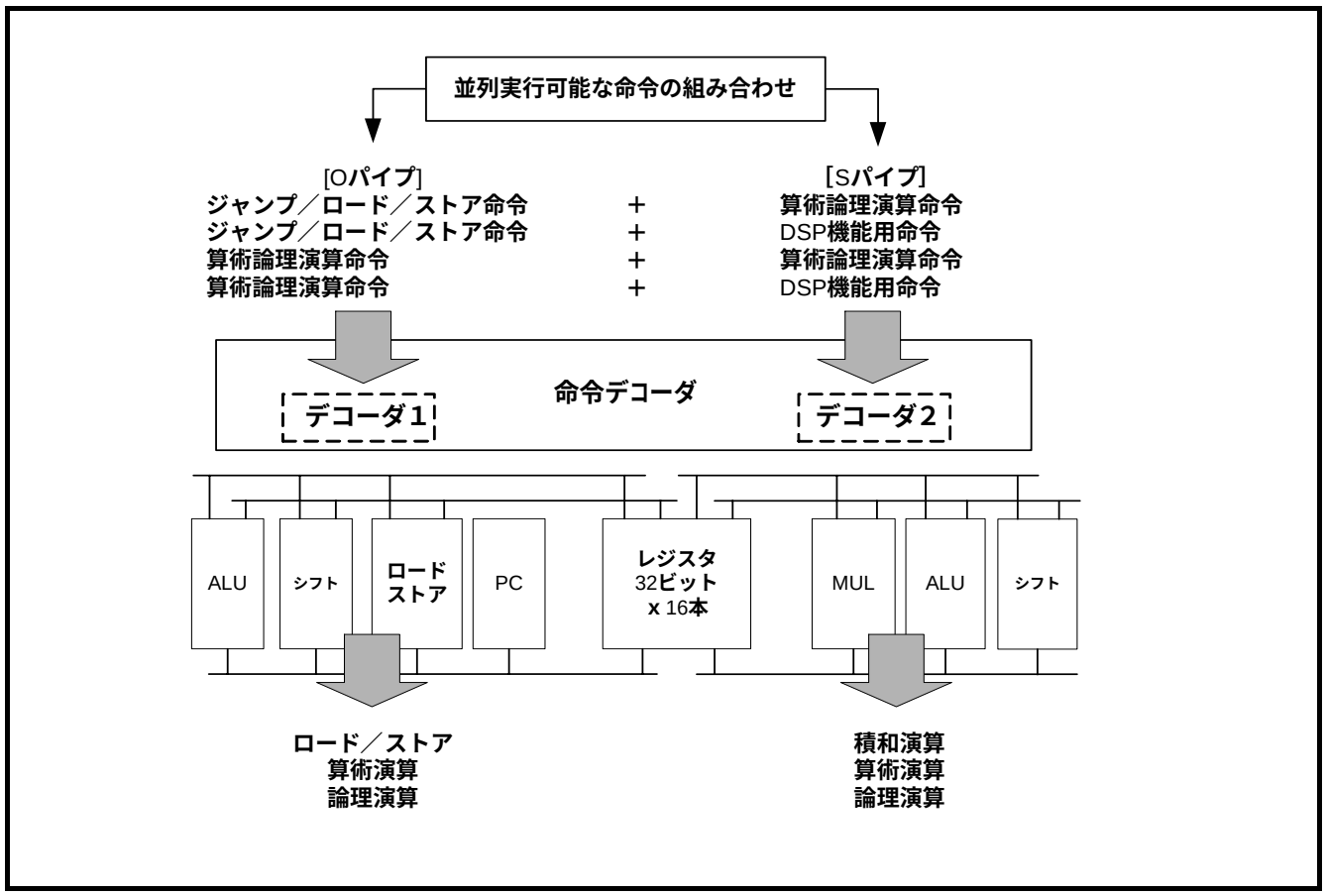


図2.5.2 OPSP-CPUの並列実行機構

2.5.3 16ビット命令のカテゴリ別一覧

並列実行対象となる16ビット命令は、実行できるパイプラインにより3つのカテゴリに分類されます。

- **Oパイプのみで実行できる命令 （左側命令：O－）**
- **Sパイプのみで実行できる命令 （右側命令：－S）**
- **Oパイプ、Sパイプの両方で実行できる命令 （両側命令：OS）**

表2.5.1に16ビット命令のカテゴリ別一覧を示します。

表2.5.1　16ビット命令のカテゴリ別一覧

Oー（左側命令）	OS（両側命令）	ーS（右側命令）
BC	ADD	MACHI
BCL	ADDI	MACLH1
BL	ADDV	MACLO
BNC	ADDX	MACWHI
BNCL	AND	MACQLO
BRA	CMP	MACWU1
BTST	CMPEQ	MSBLO
CLRPSW	CMPU	MUL
JC	CMPZ	MULHI
JL	LDI	MULLO
JMP	MV	MULWHI
JNC	NEG	NULWLO
LD	NOP	NULWU1
LDB	NOT	MVFACHI
LDH	OR	MVFACLO
LDUB	PCMPBZ	MVFACMI
LDUH	SLL	MVTACHI
LOCK	SLLI	MVTACLO
MVFC	SRA	RAC
MVTC	SRAI	RACH
RTE	SRL	SADD
SETPSW	SRLI	
SC	SUB	
SNC	SUBV	
ST	SUBX	
STB	XOR	
STH		
TRAP		
UNLOCK		

2.5.4 並列実行と命令の配置

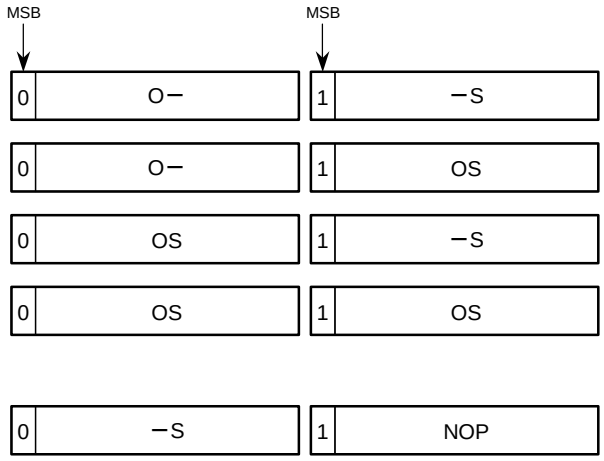
並列実行可能な1ビット命令のペアは、以下の4通りの命令カテゴリの組み合わせに限定されます。

- **左側命令と右側命令（O－と－S）**
- **左側命令と両側命令（O－とOS）**
- **両側命令と右側命令（OSと－S）**
- **両側命令と両側命令（OSとOS）**

16ビット命令ペアが並列実行される場合のカテゴリの配置を以下に示します。

- **左側命令（O－）をMSB側16ビット、右側命令（－S）をLSB側16ビット**
- **左側命令（O－）をMSB側16ビット、両側命令（OS）をLSB側16ビット**
- **両側命令（OS）をMSB側16ビット、右側命令（－S）をLSB側16ビット**
- **両側命令（OS）をMSB側16ビット、LSB側16ビット**

ただし、ワードアラインメント調整のためのNOP命令をLSB側16ビットに配置する場合は、右側命令（－S）をMSB側16ビットに配置して並列実行することができます。



注．並列実行が行われる場合の LSB 側 16 ビットに配置された命令は、アセンブラが自動的に MSB を 1 にした命令に置き換えます。

図2.5.3 並列実行可能な命令カテゴリの配置

2.5.5 オペランドの干渉

並列実行される２つの16ビット命令は、時間的な順序はなく、お互いに独立に実行されます。並列実行の場合は、２つの命令間でオペランドに関する依存関係はないものとして取り扱いますので、インタロック処理は行いません。プログラミング時には注意が必要です。

並列実行の命令ペアで参照するソースオペランドの値は、並列実行が行われる直前の値が使用されます。たとえば、並列実行される命令ペアにおいて、一方の命令が書き込みを行うレジスタを、もう一方の命令で参照した場合は、命令ペアの並列実行直前のレジスタ値を参照します。また、実行後は命令ペアの実行結果をレジスタに書き込みます。

なお、同じレジスタに書き込む2命令を並列実行した場合（レジスタへの書き込み衝突）の動作は保証されません。

（1）汎用レジスタにおけるオペランドの干渉例

2つの転送命令(MV命令)による、汎用レジスタにおけるオペランド干渉の代表的な例を以下に示します。

例1：
```
MV   R1,R0   ||  MV   R2,R1
```

例2：
```
MV   R1,R0   ||  MV   R1,R2
```

注. "||"の表記は、２つの命令が並列実行されることを示します。

例1は、命令ペアの一方で書き込みを行うレジスタ(R1)を、もう一方で参照する例です。この場合、R1にはR0の値が代入されます。また、R2にはR1への代入が行われる前のR1の値（”MV R1,R0 ”が実行される前のR1の値）が代入されます。

例2は、同じレジスタに書き込む命令ペアを実行する例（レジスタへの書き込み衝突）です。この場合、2つのMV命令の書き込み先レジスタが共にR1であり、実行後のR1の値は不定となります。

（2）制御レジスタにおけるオペランドの干渉例

オペランドの干渉は、汎用レジスタに限らず、条件(C)ビットを含むPSW,CBRといった制御レジスタにおいても成り立ちます。

例3：　2つの命令が逐次実行される場合
```
CMP   R1,R0
BC    _label
```

例4：　2つの命令が並列実行される場合
```
CMP   R1,R0   ||  BC   _label
```

注. "||"の表記は、２つの命令が並列実行されることを示します。

例3は、比較命令(CMP)の実行後、条件分岐命令(BC)が実行されます。この場合、CMP命令実行の結果、条件ビット(C)は更新され、BC命令はこの更新された条件ビット(C)の値によって、分岐を決定します。

例4は、CMP命令とBC命令が並列実行されます。BC命令はCMP命令実行前の条件ビット(C)によって分岐を決定します。CMP命令実行後の条件ビット(C)の値ではないことにご注意ください。

CMP命令の実行結果は、並列実行後の条件ビット(C)に反映されます。

また、条件ビット(C)を変化させる2つの命令を並列実行した場合、汎用レジスタにおけるレジスタ書き込み衝突と同様に、実行後の条件ビット(C)の値は不定になります。

命令ペアの並列実行後に条件ビット(C)が不定となる例を以下に示します。

```
例5：    CMP    R1,R2      ||    ADDX   R3,R4
例6：    MVTC   R1,PSW     ||    ADDX   R1,R2
例7：    TRAP   #1         ||    CMP    R3,R4
例8：    RTE               ||    ADDX   R3,R4
```

注. "||"の表記は、２つの命令が並列実行されることを示します。

レイアウトの都合上、このページは白紙です。

第3章
命令

3.1 命令詳細説明の記述方法

命令詳細説明で記述される各項目の概要を以下に示します。

【ニーモニック】

OPSP-CPUのニーモニックは、命令とそれに続く命令のオペランド（操作対象）の記述で構成されています。オペランドには以下のものがあります。

表3.1.1 オペランド一覧

表記(注)	アドレッシングモード	命令の操作対象
R	レジスタ直接	OPSP-CPUの汎用レジスタ（R0～R15）
CR	制御レジスタ	OPSP-CPUの制御レジスタ (CR0=PSW, CR1=CBR, CR2=SPI, CR3=SPU, CR5=EVB, CR6=BPC)
CPR	コプロセッサレジスタ	コプロセッサのレジスタ
A	アキュムレータ	OPSP-CPUのアキュムレータ(A0,A1)の内容
@Rn	レジスタ間接	レジスタの値をアドレスとするメモリの内容
@(disp, Rn)	レジスタ相対間接	(レジスタの値)+(16ビットの定数を32ビットに符号拡張した値)をアドレスとするメモリの内容
@Rn+	レジスタ間接 +レジスタ更新	レジスタ値を+4、+2または+1する (更新前のレジスタ値をアドレスとするメモリの内容)
@+Rn	レジスタ間接 +レジスタ更新	レジスタ値を+4する (更新後のレジスタ値をアドレスとするメモリの内容)
@－Rn	レジスタ間接 +レジスタ更新	レジスタ値を－4する (更新後のレジスタ値をアドレスとするメモリの内容)
#imm	イミディエート	即値(値の扱いは各命令の詳細説明参照)
pcdisp	PC相対	(PCの値)+(8, 16または24ビットのディスプレースメントを32ビットに符号拡張して左へ2ビットシフトした値)をアドレスとするメモリの内容

注．オペランド表記において、Rsrc, Rdestと表記した場合のsrc, destは任意の汎用レジスタ番号(0～15)が入ります。また、CRsrc, CRdestと表記した場合のsrc, destには任意の制御レジスタ番号(0～3,5,6)が、Asrc, Adestと表記した場合のsrc, destには任意のアキュムレータ番号(0,1)が入ります。

【動作】

命令の動作説明では、命令の動作概要とC言語に準じた表記での動作説明を行っています。動作の表記法の概要を以下に示します。

表3.1.2 動作表記法（演算子）

演算子	意味
+	加算(二項演算子)
−	減算(二項演算子)
*	乗算(二項演算子)
／	除算(二項演算子)
%	剰余算(二項演算子)
++	インクリメント(単項演算子)
−−	デクリメント(単項演算子)
−	符号の反転(単項演算子)
=	右辺から左辺への代入(代入演算子)
+=	左辺と右辺の変数を加算し、左辺に代入(代入演算子)
−=	左辺の変数から右辺の変数を減算し、左辺に代入(代入演算子)
>	より大(関係演算子)
<	より小(関係演算子)
>=	より大か等しい(関係演算子)
<=	より小か等しい(関係演算子)
==	等しい(関係演算子)
! =	等しくない(関係演算子)
&&	AND(論理演算子)
｜｜	OR(論理演算子)
!	NOT(論理演算子)
?:	条件式をつくる(条件演算子)

表3.1.3 動作表記法（ビット演算子）

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

表3.1.4 データタイプ

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

【機能】

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

【発生EIT】

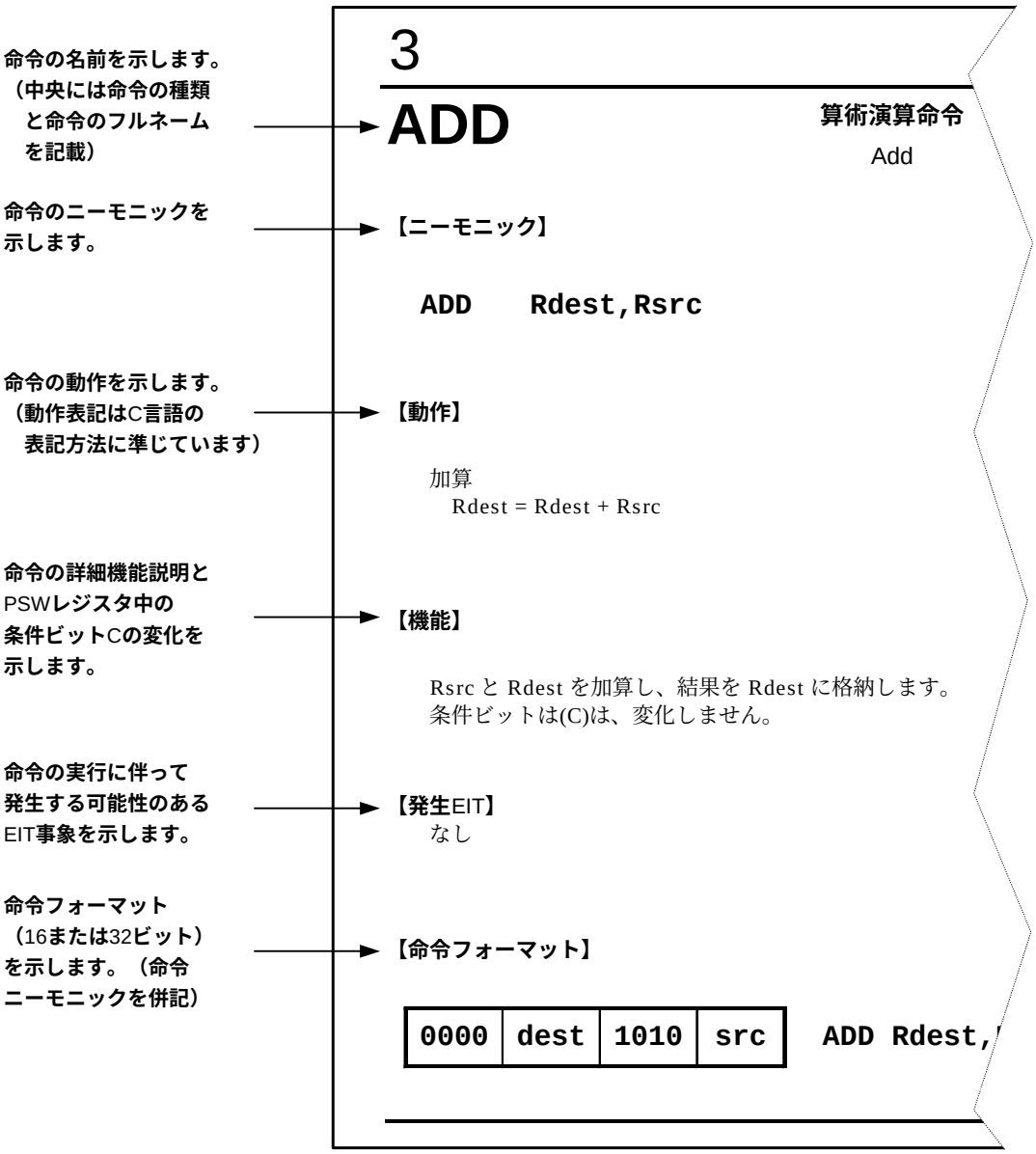
発生EITには、その命令実行で発生する可能性のあるEIT事象（例外、割り込み、トラップ）を示しています。命令実行で発生する可能性があるのはアドレス例外、トラップ、特権命令例外です。

【命令フォーマット】

16ビットまたは32ビットの命令ビットパターンを示しています。なお、src, destフィールドには対応するレジスタ番号が、また imm, dispにはそれぞれイミディエート（即値）、ディスプレースメントの値が入ります（代入可能な数値の大きさはフィールドの幅で決まります）。命令フォーマットについては、「第2章　2.3　命令フォーマット」を参照下さい。

3.2 命令詳細説明

次ページよりOPSP-CPUの各命令の詳細説明を示します。各命令はアルファベット順に掲載しています。なお各ページの構成は下記のとおりです。



ADD

算術演算命令

Add

ADD

【ニーモニック】

```
ADD  Rdest,Rsrc
```

【動作】

加算

Rdest = Rdest + Rsrc

【機能】

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

【発生EIT】

なし

【命令フォーマット】

0000	dest	1010	src	ADD Rdest,Rsrc

ADD3

算術演算命令

Add 3-operand

ADD3

【ニーモニック】

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

【動作】

加算

Rdest = Rsrc + (signed short) imm16;

【機能】

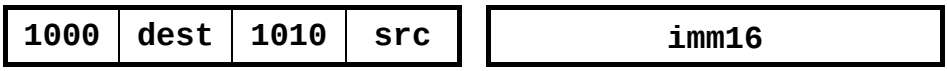
Rsrcに16ビット即値を加算し、結果をRdestに格納します。16ビット即値は、演算前に32ビットに符号拡張されます。

条件ビット(C)は変化しません

【発生EIT】

なし

【命令フォーマット】




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

ADDI

算術演算命令

Add immediate

ADDI

【ニーモニック】

ADDI Rdest,#imm8

【動作】

加算

Rdest = Rdest + (signed char) imm8;

【機能】

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

条件ビット(C)は変化しません

【発生EIT】

なし

【命令フォーマット】

0100	dest	imm8

ADDI Rdest,#imm8

ADDV

算術演算命令

Add with overflow checking

ADDV

【ニーモニック】

```
ADDV  Rdest,Rsrc
```

【動作】

加算

Rdest = (signed) Rdest + (signed) Rsrc;

C = overflow ? 1 : 0

【機能】

RdestとRsrcを加算し、結果をRdestに格納します。

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

【発生EIT】

なし

【命令フォーマット】

0000	dest	1000	src

ADDV Rdest,Rsrc

ADDV3

算術演算命令

Add 3-operand with overflow checking

ADDV3

【ニーモニック】

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

【動作】

加算

Rdest = (signed) Rsrc+ (signed)((signed short)imm16);

C = overflow ? 1 : 0

【機能】

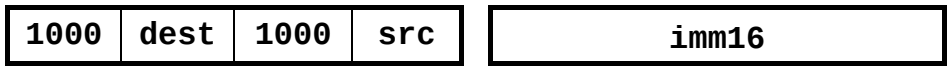
Rsrcと16ビット即値を加算し、結果をRdestに格納します。16ビット即値は演算前に32ビットに符号拡張されます。

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

【発生EIT】

なし

【命令フォーマット】




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

ADDX

算術演算命令

Add with carry

ADDX

【ニーモニック】

```
ADDX  Rdest,Rsrc
```

【動作】

加算

Rdest = (unsigned) Rdest + (unsigned) Rsrc + C;

C = carry_out ? 1 : 0;

【機能】

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

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

【発生EIT】

なし

【命令フォーマット】

0000	dest	1001	src

ADDX Rdest,Rsrc

AND

算術演算命令

AND

AND

【ニーモニック】

```
AND  Rdest,Rsrc
```

【動作】

論理積

Rdest = Rdest & Rsrc;

【機能】

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

【発生EIT】

なし

【命令フォーマット】

0000	dest	1100	src

AND Rdest,Rsrc

AND3

算術演算命令

AND 3-operand

AND3

【ニーモニック】

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

【動作】

論理積

Rdest = Rsrc & (unsigned short) imm16;

【機能】

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

【発生EIT】

なし

【命令フォーマット】


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

BC 分岐命令 BC

Branch on C-bit

【ニーモニック】

① BC pcdisp8
② BC pcdisp24

【動作】

条件付き分岐

① if (C= =1) PC = (PC & 0xfffffffc) + (((signed char) pcdisp8) << 2);

② if (C= =1) PC = (PC & 0xfffffffc) + (sign_extend (pcdisp24) << 2);

where

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

【機能】

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

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0111	1100	pcdisp8

BC pcdisp8

1111	1100	

pcdisp24

BC pcdisp24

BCL　分岐命令　BCL

Branch and link on C-bit

【ニーモニック】

① BCL　pcdisp8
② BCL　pcdisp24

【動作】

条件付きサブルーチン呼び出し(PC相対間接)

```
①  if ( C = = 1 ) {
       R14 = ( PC & 0xfffffffc ) + 4 ;
       PC = ( PC & 0xfffffffc ) + ( ( ( signed char ) pcdisp8 ) << 2 ) ;
    }
②  if ( C = = 1 ) {
       R14 = ( PC & 0xfffffffc ) + 4 ;
       PC = ( PC & 0xfffffffc ) + ( sign_extend ( pcdisp24 ) << 2 ) ;
    }
where
#define sign_extend(x)   ( ( ( signed ) ( ( x ) << 8 ) ) >> 8 )
```

【機能】

条件ビット(C)が1のとき、指定されたラベルへ分岐し、戻り先をR14に格納します。命令フォーマットには2種類ありますが、アセンブラレベルで記述する場合はオペランドに分岐先のラベルを書けば、通常最短のフォーマットが選ばれます。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0111	1000	pcdisp8

BCL pcdisp8

1111	1000	
pcdisp24		

BCL pcdisp24

BCLR

ビット操作命令

Bit clear

BCLR

【ニーモニック】

BCLR #bitpos,@(disp16,Rsrc)

【動作】

指定ビットのクリア

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

【機能】

Rsrcと16ビットのディスプレースメントで指定された番地のメモリのバイトデータを読み込み、bitposで指定されたビットを0に変更した値をストアします。ディスプレースメントは、アドレス計算の前に符号拡張されます。bitposは0～7で、MSBが0、LSBが7となります。メモリのアクセスはバイトサイズで行われます。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

BCLR #bitpos,@(disp16,Rsrc)

BEQ

分岐命令

Branch on equal

BEQ

【ニーモニック】

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

【動作】

条件付き分岐

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

【機能】

Rsrc1とRsrc2が等しいとき、指定されたラベルへ分岐します。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】


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

BEQZ

分岐命令

Branch on equal to zero

BEQZ

【ニーモニック】

```
BEQZ  Rsrc,pcdisp16
```

【動作】

条件付き分岐

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

【機能】

Rsrcが0のとき、指定されたラベルへ分岐します。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

1011	0000	1000	src	pcdisp16

```
BEQZ Rsrc,pcdisp16
```

BGEZ

分岐命令

Branch on greater than or equal to zero

BGEZ

【ニーモニック】

```
BGEZ  Rsrc,pcdisp16
```

【動作】

条件付き分岐

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

【機能】

Rsrcの内容を符号付き32ビット値としてみたとき、0または0より大きい場合に指定されたラベルへ分岐します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

1011	0000	1011	src	pcdisp16

```
BGEZ Rsrc,pcdisp16
```

BGTZ

分岐命令

Branch on greater than zero

BGTZ

【ニーモニック】

```
BGTZ  Rsrc,pcdisp16
```

【動作】

条件付き分岐

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

【機能】

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

【発生EIT】

なし

【命令フォーマット】

1011	0000	1101	src	pcdisp16

```
BGTZ Rsrc,pcdisp16
```

BL　分岐命令　BL

Branch and link

【ニーモニック】

① BL pcdisp8
② BL pcdisp24

【動作】

サブルーチン呼び出し(PC相対)

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

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

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

【機能】

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

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0111	1110	pcdisp8

BL pcdisp8

1111	1110	

pcdisp24

BL pcdisp24

BLEZ

分岐命令

Branch on less than or equal to zero

BLEZ

【ニーモニック】

```
BLEZ  Rsrc,pcdisp16
```

【動作】

条件付き分岐

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

【機能】

Rsrcの内容を符号付き32ビット値としてみたとき、0または0より小さい場合に指定されたラベルへ分岐します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

1011	0000	1100	src	pcdisp16

BLEZ Rsrc,pcdisp16

BLTZ

分岐命令

Branch on less than zero

BLTZ

【ニーモニック】

```
BLTZ  Rsrc,pcdisp16
```

【動作】

条件付き分岐

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

【機能】

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

【発生EIT】

なし

【命令フォーマット】

1011	0000	1010	src	pcdisp16

BLTZ Rsrc,pcdisp16

BNC 分岐命令 BNC

Branch on not C-bit

【ニーモニック】

① **BNC pcdisp8**
② **BNC pcdisp24**

【動作】

条件付き分岐

① if (C= =0) PC = (PC & 0xfffffffc) + (((signed char) pcdisp8) << 2);

② if (C= =0) PC = (PC & 0xfffffffc) + (sign_extend (pcdisp24) << 2);

where

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

【機能】

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

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0111	1101	pcdisp8

BNC pcdisp8

1111	1101	

pcdisp24

BNC pcdisp24

BNCL 分岐命令 BNCL

Branch and link on not C-bit

【ニーモニック】

① **BNCL pcdisp8**
② **BNCL pcdisp24**

【動作】

条件付きサブルーチン呼び出し(PC相対間接)

```
①  if ( C = = 0 ) {
       R14 = ( PC & 0xfffffffc ) + 4 ;
       PC = ( PC & 0xfffffffc ) + ( ( ( signed char ) pcdisp8 ) << 2 ) ;
    }
②  if ( C = = 0 ) {
       R14 = ( PC & 0xfffffffc ) + 4 ;
       PC = ( PC & 0xfffffffc ) + ( sign_extend ( pcdisp24 ) << 2 ) ;
    }
where
#define sign_extend(x)   ( ( ( signed ) ( ( x ) << 8 ) ) >> 8 )
```

【機能】

条件ビット(C)が0のとき、指定されたラベルへ分岐し、戻り先をR14に格納します。命令フォーマットには2種類ありますが、アセンブラレベルで記述する場合はオペランドに分岐先のラベルを書けば、通常最短のフォーマットが選ばれます。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0111	1001	pcdisp8

BNCL pcdisp8

1111	1001	

pcdisp24

BNCL pcdisp24

BNE

分岐命令

Branch on not equal to

BNE

【ニーモニック】

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

【動作】

条件付き分岐

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

【機能】

Rsrc1とRsrc2が等しくないとき、指定されたラベルへ分岐します。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

1011	src1	0001	src2	pcdisp16

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

BNEZ

分岐命令

Branch on not equal to zero

BNEZ

【ニーモニック】

```
BNEZ  Rsrc,pcdisp16
```

【動作】

条件付き分岐

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

【機能】

Rsrcが0でないとき、指定されたラベルへ分岐します。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

1011	0000	1001	src	pcdisp16

```
BNEZ Rsrc,pcdisp16
```

BRA

分岐命令
Branch

BRA

【ニーモニック】

① BRA pcdisp8
② BRA pcdisp24

【動作】

無条件分岐

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

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

where

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

【機能】

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

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0111	1111	pcdisp8

BRA pcdisp8

1111	1111	

pcdisp24

BRA pcdisp24

BSET

ビット操作命令

Bit set

BSET

【ニーモニック】

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

【動作】

指定ビットのセット。

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

【機能】

Rsrcと16ビットのディスプレースメントで指定された番地のメモリのバイトデータを読み込み、bitposで指定されたビットを1に変更した値をストアします。ディスプレースメントは、アドレス計算の前に符号拡張されます。bitposは0～7で、MSBが0、LSBが7となります。メモリアクセスはバイトサイズで行われます。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

1010	0	bitpos	0110	src

disp16

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

BTST

ビット操作命令

Bit test

BTST

【ニーモニック】

```
BTST  #bitpos,Rsrc
```

【動作】

レジスタの指定ビットを取り出す

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

【機能】

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

【発生EIT】

なし

【命令フォーマット】

BTST #bitpos, Rsrc

CLRPSW

ビット操作命令

Clear PSW

CLRPSW

【ニーモニック】

CLRPSW #imm8

【動作】

PSWレジスタのSM,IE,PM,CE,Cの任意のビットをクリア

PSW &= ~(unsigned char) imm8 | 0x0000ff00

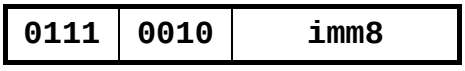
【機能】

imm8で指定された8ビット値の各ビットの反転値とPSWの下位8ビット(ビット24～31)の各ビットとの論理積をとり、その結果をPSWの下位8ビットにそれぞれ書き込みます。

【発生EIT】

特権命令例外（PIE）

【命令フォーマット】



CLRPSW #imm8

CMP

比較命令

Compare

CMP

【ニーモニック】

```
CMP  Rsrc1,Rsrc2
```

【動作】

比較

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

【機能】

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

【発生EIT】

なし

【命令フォーマット】

0000	src1	0100	src2	CMP Rsrc1,Rsrc2

CMPEQ

比較命令

Compare equal to

CMPEQ

【ニーモニック】

```
CMPEQ  Rsrc1,Rsrc2
```

【動作】

比較

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

【機能】

Rsrc1とRsrc2を比較し、Rsrc1とRsrc2が等しいとき条件ビット(C)が１にセットされます。

【発生EIT】

なし

【命令フォーマット】

0000	src1	0110	src2

CMPEQ Rsrc1,Rsrc2

CMPI 比較命令 CMPI

Compare immediate

【ニーモニック】

```
CMPI   Rsrc,#imm16
```

【動作】

比較

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

【機能】

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

【発生EIT】

なし

【命令フォーマット】

1000	0000	0100	src

imm16

```
CMPI Rsrc,#imm16
```

CMPU 比較命令 CMPU

Compare unsigned

【ニーモニック】

```
CMPU  Rsrc1,Rsrc2
```

【動作】

比較

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

【機能】

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

【発生EIT】

なし

【命令フォーマット】

0000	src1	0101	src2

CMPU Rsrc1,Rsrc2

CMPUI 比較命令 CMPUI

Compare unsigned immediate

【ニーモニック】

```
CMPUI  Rsrc,#imm16
```

【動作】

比較

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

【機能】

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

【発生EIT】

なし

【命令フォーマット】

1000	0000	0101	src	imm16

```
CMPUI Rsrc,#imm16
```

CMPZ

比較命令

Compare equal to zero

CMPZ

【ニーモニック】

```
CMPZ  Rsrc
```

【動作】

比較

C = (Rsrc == 0)　? 1 : 0;

【機能】

Rsrcが0のとき条件ビット(C)が１にセットされます。

【発生EIT】

なし

【命令フォーマット】

0000	0000	0111	src	CMPZ Rsrc

DIV　乗除算命令　DIV

Divide

【ニーモニック】

```
DIV  Rdest,Rsrc
```

【動作】

符号付き除算

Rdest = (signed) Rdest / (signed) Rsrc;

【機能】

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

条件ビット(C)は変化しません。

Rsrcが0のとき、結果は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	0000	src

0000	0000	0000	0000

```
DIV Rdest,Rsrc
```

DIVB

乗除算命令
Divide byte

DIVB

【ニーモニック】

```
DIVB  Rdest,Rsrc
```

【動作】

符号付き除算

Rdest =(signed char)Rdest / (signed)Rsrc ;

【機能】

RdestをRsrcで割り算し、商をRdestに格納します。

オペランドは,被除数が符号付き8ビット値として扱われ、上位24ビット(ビット0～23)は無視されます。除数は符号付き32ビット値として扱われ、商は0方向に丸められます。

条件ビット(C)は変化しません。

Rsrcが0のとき、Rdestの値は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	0000	src

0000	0000	0001	1000

DIVB Rdest,Rsrc

DIVH

乗除算命令

Divide Half-word

DIVH

【ニーモニック】

```
DIVH  Rdest,Rsrc
```

【動作】

符号付き除算

Rdest = (signed short) Rdest / (signed) Rsrc ;

【機能】

RdestをRsrcで割り算し、商をRdestに格納します。オペランドは、被除数が符号付き16ビット値として扱われ、上位16ビット(ビット0～15)は無視されます。除数は符号付き32ビット値として扱われ、商は0方向に丸められます。

条件ビット(C)は変化しません。

Rsrcが0のとき、Rdestの値は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	0000	src

0000	0000	0001	0000

```
DIVH Rdest,Rsrc
```

DIVU

乗除算命令

Divide unsigned

DIVU

【ニーモニック】

```
DIVU  Rdest,Rsrc
```

【動作】

符号なし除算

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

【機能】

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

条件ビット(C)は変化しません。

Rsrcが0のとき、結果は変化しません。

【発生EIT】

なし

【命令フォーマット】


```
DIVU Rdest,Rsrc
```

DIVUB

乗除算命令

Divide unsigned byte

DIVUB

【ニーモニック】

```
DIVUB  Rdest,Rsrc
```

【動作】

符号なし除算

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

【機能】

RdestをRsrcで割り算し、商をRdestに格納します。

オペランドは,被除数が符号なし8ビット値として扱われ、上位24ビット(ビット0～23)は無視されます。除数は符号なし32ビット値として扱われ、商は0方向に丸められます。

条件ビット(C)は変化しません。

Rsrcが0のとき、Rdestの値は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	0001	src

0000	0000	0001	1000

```
DIVUB Rdest,Rsrc
```

DIVUH

乗除算命令

Divide unsigned halfword

DIVUH

【ニーモニック】

```
DIVUH  Rdest,Rsrc
```

【動作】

符号なし除算

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

【機能】

RdestをRsrcで割り算し、商をRdestに格納します。

オペランドは,被除数が符号なし16ビット値として扱われ、上位16ビット(ビット0～15)は無視されます。除数は符号なし32ビット値として扱われ、商は0方向に丸められます。

条件ビット(C)は変化しません。

Rsrcが0のとき、Rdestの値は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	0001	src

0000	0000	0001	0000

```
DIVUH Rdest,Rsrc
```

JC

分岐命令

Jump on C-bit

JC

【ニーモニック】

```
JC   Rsrc
```

【動作】

条件付き分岐

if (C = =1)PC =Rsrc & 0xfffffffc ;

【機能】

条件ビット(C)が1のとき、Rsrcで指定された番地へ分岐します。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	1100	1100	src	JC Rsrc

JL　分岐命令　JL

Jump and link

【ニーモニック】

```
JL   Rsrc
```

【動作】

サブルーチン呼び出し(レジスタ直接)

R14 = (PC & 0xfffffffc) + 4;

PC = Rsrc & 0xfffffffc;

【機能】

Rsrcで指定された番地へ無条件分岐を行い、戻り先をR14に格納します。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	1110	1100	src

JL Rsrc

JMP

分岐命令

Jump

JMP

【ニーモニック】

```
JMP  Rsrc
```

【動作】

無条件分岐

PC = Rsrc & 0xfffffffc;

【機能】

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

【発生EIT】

なし

【命令フォーマット】

0001	1111	1100	src	JMP Rsrc

JNC

分岐命令

Jump on not C-bit

JNC

【ニーモニック】

```
JNC  Rsrc
```

【動作】

条件付き分岐

if (C==0)PC =Rsrc & 0xfffffffc ;

【機能】

条件ビット(C)が0のとき、Rsrcで指定された番地へ分岐します。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	1101	1100	src	JNC Rsrc

LD

ロード／ストア命令

Load

LD

【ニーモニック】

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

【動作】

メモリからレジスタへのロード

① Rdest = *(signed int *) Rsrc;
② Rdest = *(signed int *) Rsrc, Rsrc += 4;
③ Rdest = *(signed int *) (Rsrc + (signed short) disp16);

【機能】

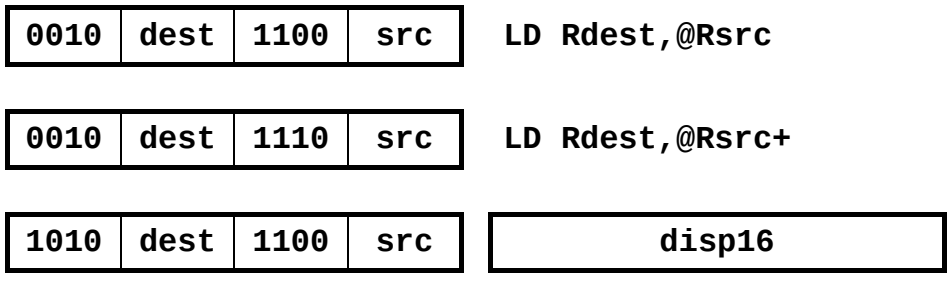
① Rsrcで指定された番地のメモリ内容をRdestに格納します。
② Rsrcで指定された番地のメモリ内容をRdestに格納し、その後でRsrcを4インクリメントします。
③ Rsrcと16ビットのディスプレースメントで指定された番地のメモリ内容を、Rdestに格納します。16ビットのディスプレースメントは、アドレス計算の前に32ビットに符号拡張されます。

条件ビット(C)は変化しません。

【発生EIT】

アドレス例外(AE)

【命令フォーマット】



LD24

転送命令

Load 24-bit immediate

LD24

【ニーモニック】

```
LD24   Rdest,#imm24
```

【動作】

24ビット即値データの転送

Rdest = imm24 & 0x00ffffff;

【機能】

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

【発生EIT】

なし

【命令フォーマット】

1110	dest		imm24

```
LD Rdest,#imm24
```

LDB

ロード／ストア命令

Load byte

LDB

【ニーモニック】

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

【動作】

メモリからレジスタへのロード

① Rdest = *(signed char *) Rsrc;
② Rdest = *(signed char *) (Rsrc + (signed short) disp16);

【機能】

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

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0010	dest	1000	src

LDB Rdest,@Rsrc

1010	dest	1000	src

disp16

LDB Rdest,@(disp16,Rsrc)

LDH

ロード／ストア命令

Load halfword

LDH

【ニーモニック】

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

【動作】

メモリからレジスタへのロード

① Rdest = *(signed short *) Rsrc;
② Rdest = *(signed short *) (Rsrc + (signed short) disp16);

【機能】

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

条件ビット(C)は変化しません。

【発生EIT】

アドレス例外(AE)

【命令フォーマット】

0010	dest	1010	src

LDH Rdest,@Rsrc

1010	dest	1010	src	disp16

LDH Rdest,@(disp16,Rsrc)

LDI

転送命令

Load immediate

LDI

【ニーモニック】

① LDI Rdest,#imm8
② LDI Rdest,#imm16

【動作】

即値データの転送

① Rdest = (signed char) imm8;
② Rdest = (signed short) imm16;

【機能】

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

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0110	dest	imm8

LDI Rdest,#imm8

1001	dest	1111	0000

imm16

LDI Rdest,#imm16

LDUB

ロード／ストア命令

Load unsigned byte

LDUB

【ニーモニック】

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

【動作】

メモリからレジスタへのロード

① Rdest = *(unsigned char *) Rsrc;
② Rdest = *(unsigned char *) (Rsrc + (signed short) disp16);

【機能】

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

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0010	dest	1001	src

LDUB Rdest,@Rsrc

1010	dest	1001	src

disp16

LDUB Rdest,@(disp16,Rsrc)

LDUH

ロード／ストア命令

Load unsigned halfword

LDUH

【ニーモニック】

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

【動作】

メモリからレジスタへのロード

① Rdest = *(unsigned short *) Rsrc;
② Rdest = *(unsigned short *) (Rsrc + (signed short) disp16);

【機能】

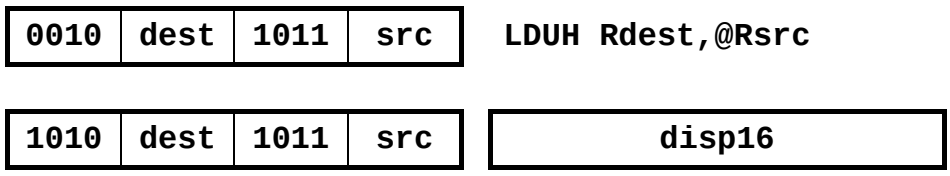
① Rsrcで指定された番地のメモリのハーフワードデータを、ゼロ拡張してRdestに格納します。
② Rsrcと16ビットのディスプレースメントで指定された番地のメモリのハーフワードデータを、ゼロ拡張してRdestに格納します。16ビットのディスプレースメントは、アドレス計算の前に32ビットに符号拡張されます。

条件ビット(C)は変化しません。

【発生EIT】

アドレス例外(AE)

【命令フォーマット】



LDUH Rdest,@(disp16,Rsrc)

LOCK

ロード／ストア命令

Load locked

LOCK

【ニーモニック】

```
LOCK   Rdest,@Rsrc
```

【動作】

ロック付きロード

LOCK = 1, Rdest = *(signed int *) Rsrc;

【機能】

Rsrcで指定された番地のメモリのワードデータを、Rdestに格納します。

条件ビット(C)は変化しません。

この命令は、通常のロードを行う以外にLOCKビットのセットも行います。

LOCKビットが1の場合、OPSP-CPUは内蔵DMACからの要求またはHOLD要求を受け付けません。

LOCKビットのクリアは、UNLOCK命令によって行われます。

LOCKビットはCPUの内部にあり、LOCK命令とUNLOCK命令による操作を除き、ユーザがこのビットを直接アクセスすることはできません。

【発生EIT】

アドレス例外(AE)

【命令フォーマット】

0010	dest	1101	src

LOCK Rdest,@Rsrc

MACHI

DSP機能用命令

Multiply-accumulate high-order halfword

MACHI

【ニーモニック】

```
MACHI  Rsrc1,Rsrc2,Adest
```

【動作】

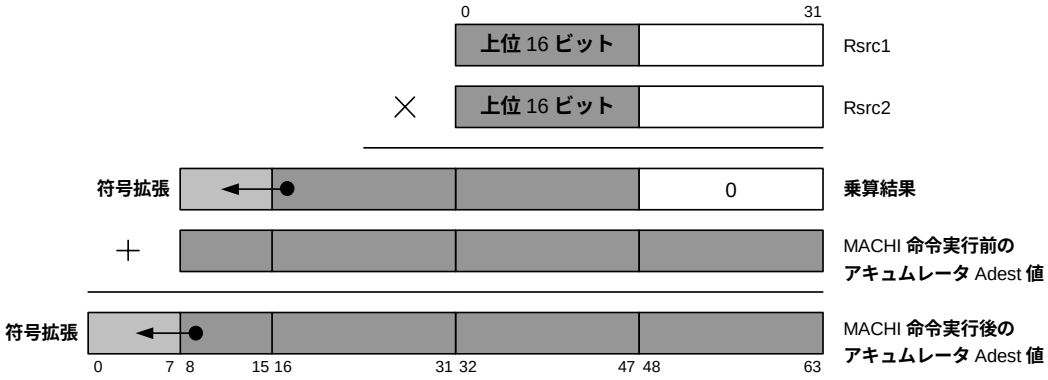
積和演算

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

【機能】

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

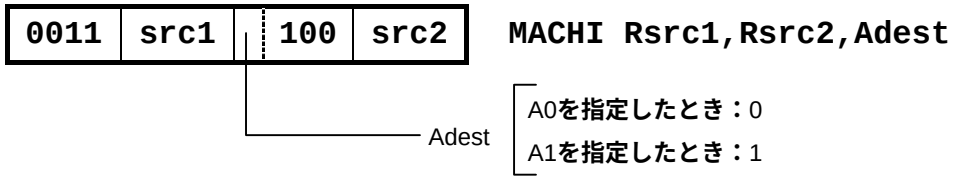
条件ビット(C) は変化しません。



【発生EIT】

なし

【命令フォーマット】



MACLH1

DSP機能用命令

Multiply-accumulate low-order halfword and high-order halfword using accumulator 1

MACLH1

【ニーモニック】

```
MACLH1   Rsrc1,Rsrc2
```

【動作】

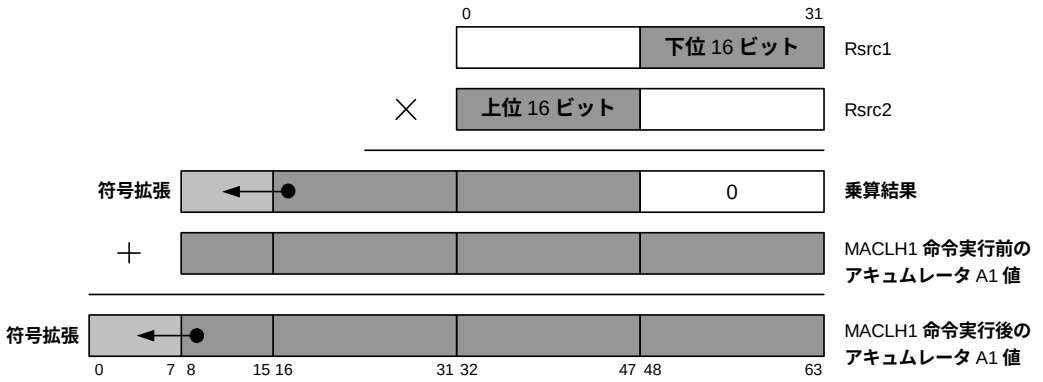
積和演算

A1 += ((signed) (Rsrc1 << 16) * (signed short) (Rsrc2 >> 16));

【機能】

Rsrc1の下位16ビットと、Rsrc2の上位16ビットの乗算を行い、乗算結果とアキュムレータA1の下位56ビットとの加算を行います。ただし、乗算結果の最下位ビットはA1のビット47にあわせ、A1のビット8～15に対応する部分は符号拡張して加算します。加算結果はA1に格納されます。Rsrc1の下位16ビットと、Rsrc2の上位16ビットは符号付き整数として扱われます。

この命令の実行でA0は変化しません。
条件ビット(C) は変化しません。



【発生EIT】

なし

【命令フォーマット】

0101	src1	1100	src2

MACLH1 Rsrc1,Rsrc2

MACLO

DSP機能用命令

Multiply-accumulate low-order halfword

MACLO

【ニーモニック】

```
MACLO  Rsrc1,Rsrc2,Adest
```

【動作】

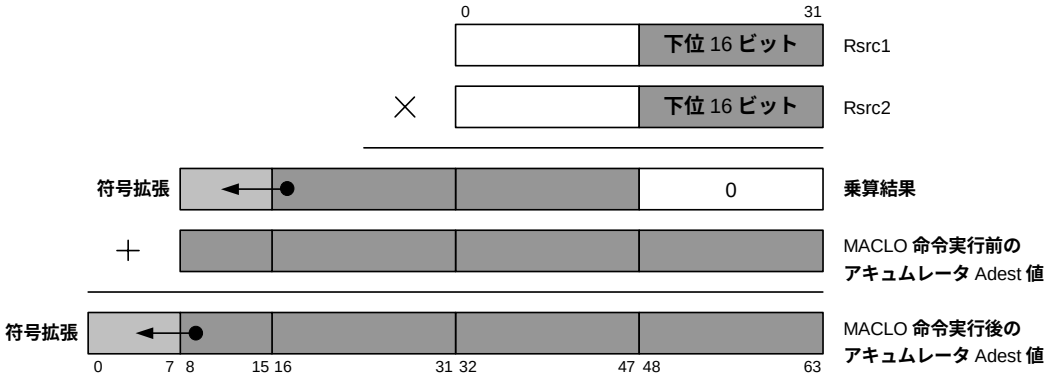
積和演算

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

【機能】

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

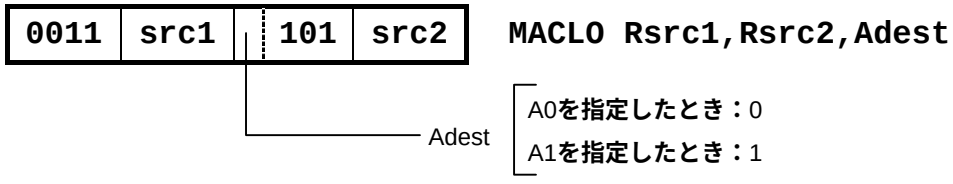
条件ビット(C) は変化しません。



【発生EIT】

なし

【命令フォーマット】



MACWHI

DSP機能用命令

Multiply-accumulate word and high-order halfword

MACWHI

【ニーモニック】

```
MACWHI   Rsrc1,Rsrc2
```

【動作】

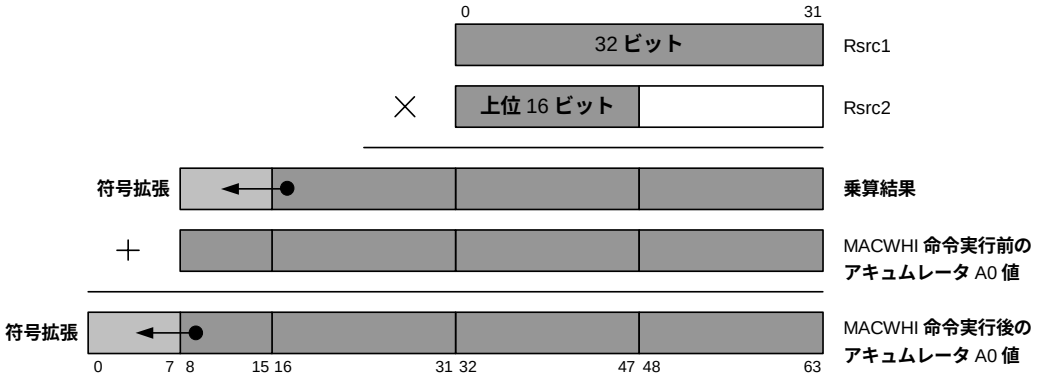
積和演算

A0 += ((signed) Rsrc1 * (signed short) (Rsrc2 >> 16));

【機能】

Rsrc1(32ビット)と、Rsrc2の上位16ビットの乗算を行い、乗算結果とアキュムレータA0の下位56ビットとの加算を行います。ただし、乗算結果の最下位ビットはA0の最下位ビットにあわせ、A0のビット8～15に対応する部分は符号拡張して加算します。加算結果はA0に格納されます。Rsrc1(32ビット)と、Rsrc2の上位16ビットは符号付き整数として扱われます。

この命令の実行でA1は変化しません。
条件ビット(C) は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0110	src2

MACWHI Rsrc1,Rsrc2

MACWLO

DSP機能用命令

Multiply-accumulate word and low-order halfword

MACWLO

【ニーモニック】

```
MACWLO   Rsrc1,Rsrc2
```

【動作】

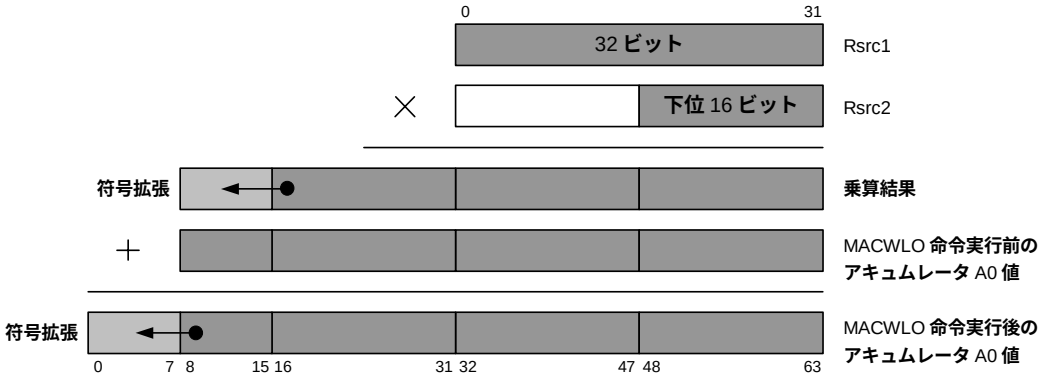
積和演算

A0 += ((signed) Rsrc1 * (signed short) Rsrc2) ;

【機能】

Rsrc1(32ビット)と、Rsrc2の下位16ビットの乗算を行い、乗算結果とアキュムレータA0の下位56ビットとの加算を行います。ただし、乗算結果の最下位ビットはA0の最下位ビットにあわせ、A0のビット8～15に対応する部分は符号拡張して加算します。加算結果はA0に格納されます。Rsrc1(32ビット)と、Rsrc2の下位16ビットは符号付き整数として扱われます。

この命令の実行でA1は変化しません。
条件ビット(C) は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0111	src2

MACWLO Rsrc1,Rsrc2

MACWU1

DSP機能用命令

Multiply-accumulate word and unsigned low-order halfword using accumulator 1

MACWU1

【ニーモニック】

```
MACWU1    Rsrc1,Rsrc2
```

【動作】

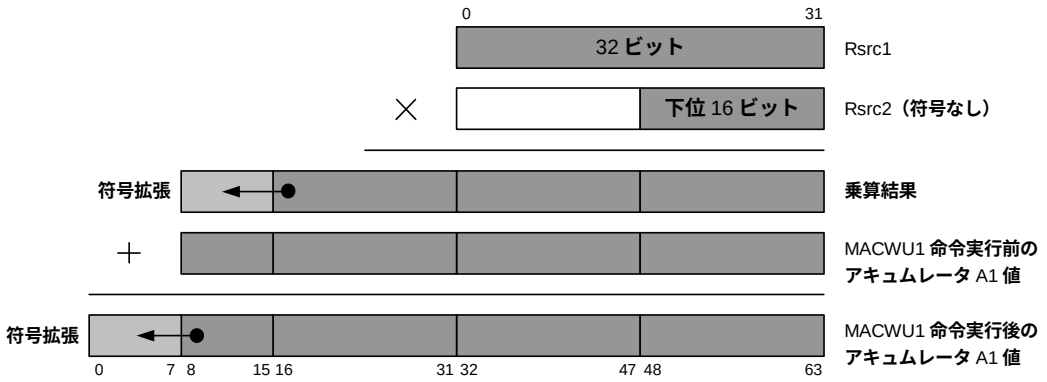
積和演算

A1 += ((signed) Rsrc1 * (unsigned short) Rsrc2);

【機能】

Rsrc1 (32ビット) と、Rsrc2の下位16ビットの乗算を行い、乗算結果とアキュムレータA1の下位56ビットとの加算を行います。ただし、乗算結果の最下位ビットはA1の最下位ビットにあわせ、A1のビット8～15に対応する部分は符号拡張して加算します。加算結果はA1に格納されます。Rsrc1 (32ビット) は符号付き整数、Rsrc2の下位16ビットは符号なし整数として扱われます。

この命令の実行でA0は変化しません。
条件ビット(C) は変化しません。



【発生EIT】

なし

【命令フォーマット】

0101	src1	1011	src2

MACWU1 Rsrc1,Rsrc2

MSBLO

DSP機能用命令

Multiply low-order halfwords and subtract

MSBLO

【ニーモニック】

```
MSBLO  Rsrc1,Rsrc2
```

【動作】

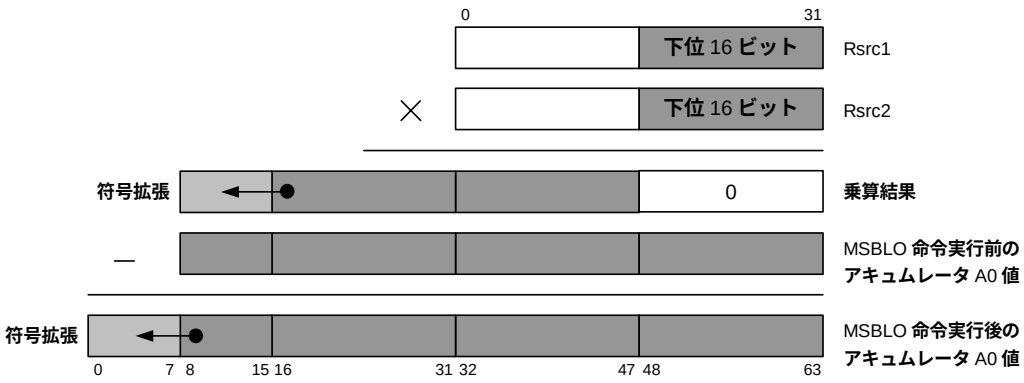
積和演算

A0 -= ((signed) (Rsrc1 << 16) * (signed short) Rsrc2);

【機能】

Rsrc1の下位16ビットと、Rsrc2の下位16ビットの乗算を行い、乗算結果をアキュムレータA0の下位56ビットから減算します。ただし、乗算結果の最下位ビットはA0のビット47にあわせ、A0のビット8～15に対応する部分は符号拡張して減算します。減算結果はA0 に格納されます。Rsrc1の下位16ビットと、Rsrc2の下位16ビットは符号付き整数として扱われます。

この命令の実行でA1は変化しません。
条件ビット(C) は変化しません。



【発生EIT】

なし

【命令フォーマット】

0101	src1	1101	src2

MSBLO Rsrc1,Rsrc2

MUL

乗除算命令

Multiply

MUL

【ニーモニック】

```
MUL  Rdest,Rsrc
```

【動作】

乗算

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

【機能】

RdestとRsrcの乗算を行い、結果をRdestに格納します。オペランドは符号付き整数として扱われます。この命令の実行によりアキュムレータA0,A1の内容はともに破壊されます。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	dest	0110	src	MUL Rdest,Rsrc

MULHI

DSP機能用命令

Multiply high-order halfwords

MULHI

【ニーモニック】

```
MULHI  Rsrc1,Rsrc2,Adest
```

【動作】

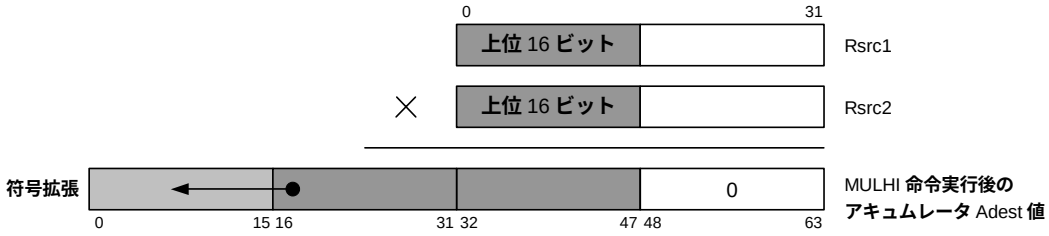
乗算

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

【機能】

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

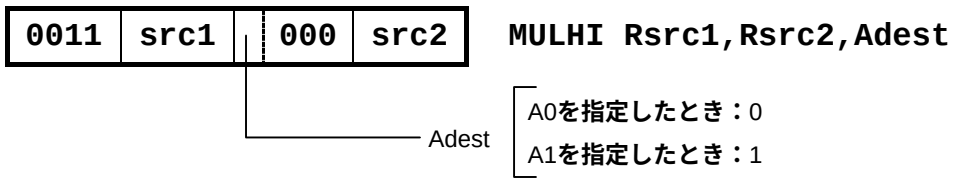
条件ビット(C) は変化しません。



【発生EIT】

なし

【命令フォーマット】



MULLO

DSP機能用命令

Multiply low-order halfwords

MULLO

【ニーモニック】

```
MULLO  Rsrc1,Rsrc2,Adest
```

【動作】

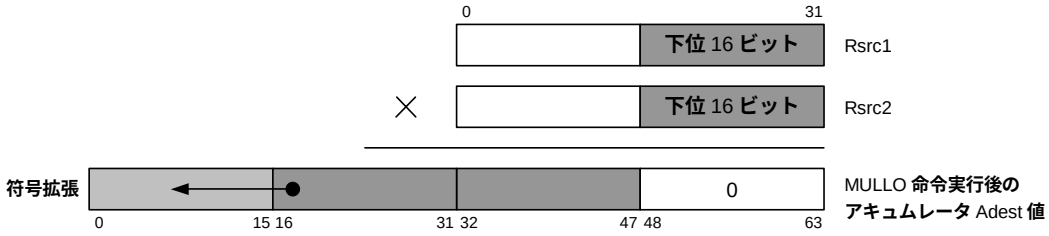
乗算

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

【機能】

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

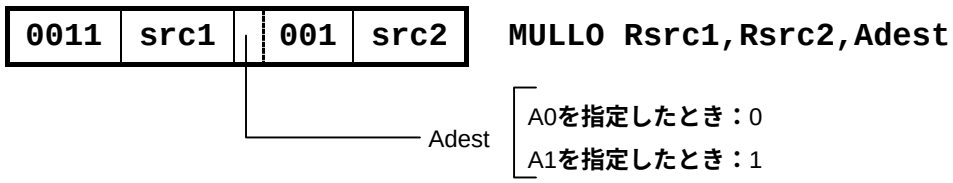
条件ビット(C) は変化しません。



【発生EIT】

なし

【命令フォーマット】



MULWHI

DSP機能用命令

Multiply word and high-order halfword

MULWHI

【ニーモニック】

```
MULWHI  Rsrc1,Rsrc2
```

【動作】

乗算

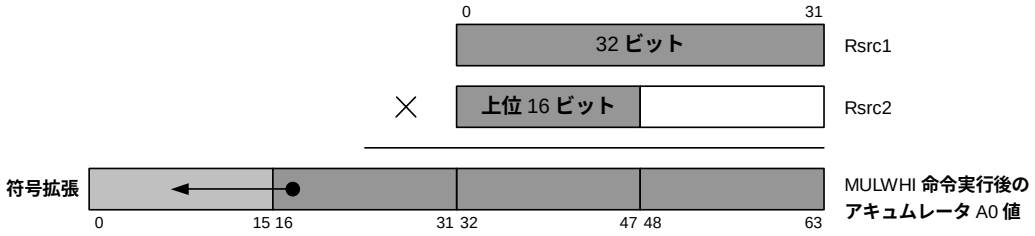
A0 = ((signed) Rsrc1 * (signed short) (Rsrc2 >> 16));

【機能】

Rsrc1(32ビット)と、Rsrc2の上位16ビットの乗算を行い、結果をアキュムレータA0に格納します。ただし、乗算結果の最下位ビットはA0の最下位ビットにあわせ、A0のビット0～15に対応する部分は符号拡張されます。Rsrc1(32ビット)と、Rsrc2の上位16ビットは符号付き整数として扱われます。

この命令の実行でアキュムレータA1は変化しません。

条件ビット(C) は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0010	src2

MULWHI Rsrc1,Rsrc2

MULWLO

DSP機能用命令

Multiply word and low-order halfword

MULWLO

【ニーモニック】

```
MULWLO   Rsrc1,Rsrc2
```

【動作】

乗算

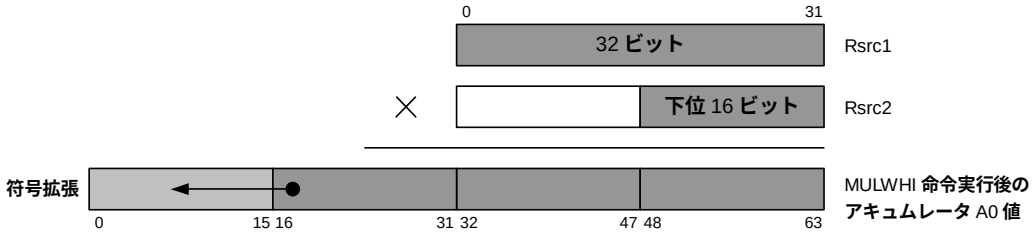
A0 = ((signed) Rsrc1 * (signed short) Rsrc2);

【機能】

Rsrc1(32ビット)と、Rsrc2の下位16ビットの乗算を行い、結果をアキュムレータA0に格納します。ただし、乗算結果の最下位ビットはA0の最下位ビットにあわせ、A0のビット0～15に対応する部分は符号拡張されます。Rsrc1(32ビット)と、Rsrc2の下位16ビットは符号付き整数として扱われます。

この命令の実行でアキュムレータA1は変化しません。

条件ビット(C) は変化しません。



【発生EIT】

なし

【命令フォーマット】

0011	src1	0011	src2

MULWLO Rsrc1,Rsrc2

MULWU1

DSP機能用命令

Multiply word and unsigned low-order halfword unsigned accumulator 1

MULWU1

【ニーモニック】

```
MULWU1   Rsrc1,Rsrc2
```

【動作】

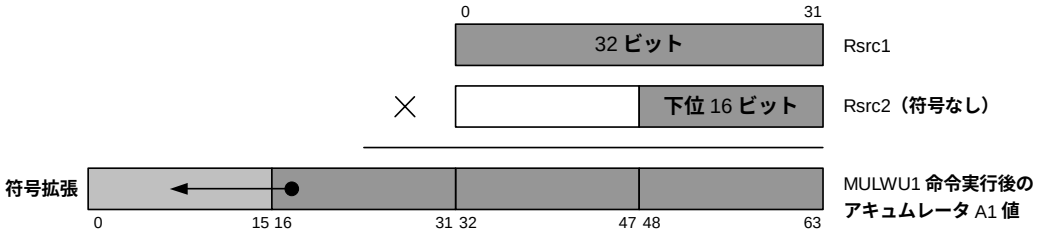
乗算

A1 = ((signed) Rsrc1 * (unsigned short) Rsrc2);

【機能】

Rsrc1 (32ビット) と、Rsrc2の下位16ビットの乗算を行い、結果をアキュムレータA1に格納します。ただし、乗算結果の最下位ビットはA1の最下位ビットにあわせ、A1のビット0～15に対応する部分は符号拡張されます。Rsrc1 (32ビット) は符号付き整数、Rsrc2の下位16ビットは符号なし整数として扱われます。

この命令の実行でアキュムレータA0は変化しません。
条件ビット(C) は変化しません。



【発生EIT】

なし

【命令フォーマット】

0101	src1	1010	src2

MULWU1 Rsrc1,Rsrc2

MV

転送命令

Move register

MV

【ニーモニック】

```
MV  Rdest,Rsrc
```

【動作】

レジスタ間転送

Rdest = Rsrc;

【機能】

Rsrcの内容をRdestに転送します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	dest	1000	src

MV Rdest,Rsrc

MVFACHI

DSP機能用命令

Move from accumulator high-order word

MVFACHI

【ニーモニック】

```
MVFACHI  Rdest,Asrc
```

【動作】

アキュムレータ～レジスタ間転送

Rdest = (signed) (Asrc >> 32);

【機能】

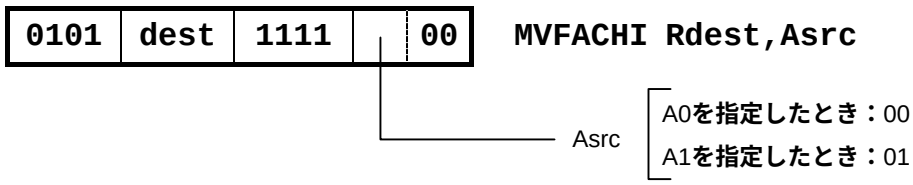
アキュムレータAsrcの上位32ビットの内容をRdestに転送します。

条件ビット(C) は変化しません。

【発生EIT】

なし

【命令フォーマット】



MVFACLO

DSP機能用命令

Move from accumulator low-order word

MVFACLO

【ニーモニック】

```
MVFACLO  Rdest,Asrc
```

【動作】

アキュムレータ～レジスタ間転送

Rdest = (signed) Asrc;

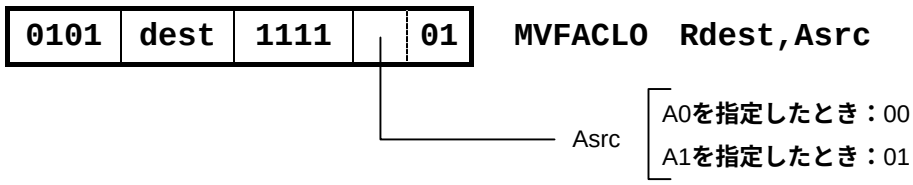
【機能】

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

【発生EIT】

なし

【命令フォーマット】



MVFACMI

DSP機能用命令

Move middle-order word from accumulator

MVFACMI

【ニーモニック】

```
MVFACMI   Rdest,Asrc
```

【動作】

アキュムレータ～レジスタ間転送

```
Rdest = (signed) (Asrc    >> 16);
```

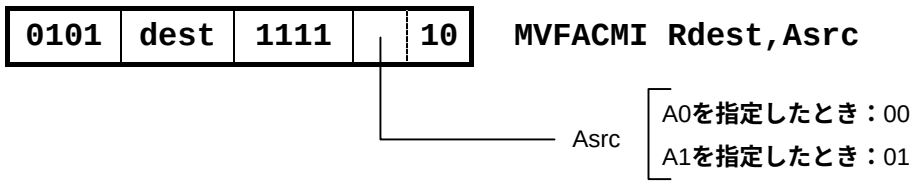
【機能】

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

【発生EIT】

なし

【命令フォーマット】



MVFC

転送命令

Move from control register

MVFC

【ニーモニック】

```
MVFC   Rdest,CRsrc
```

【動作】

レジスタ～制御レジスタ間転送

Rdest = CRsrc ;

【機能】

制御レジスタCRsrcの内容をRdestに転送します。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	dest	1001	src	MVFC Rdest,CRsrc

MVFCP

コプロセッササポート命令

Move from Coprocessor register

MVFCP

【ニーモニック】

```
MVFCP   Rdest,CPRsrc,inst,cpid
```

【動作】

レジスタ間転送

Rdest = CPRsrc(num);

【機能】

コプロセッサID(cpid)で指定したコプロセッサのレジスタCPRsrcの内容をRdestへ転送します。

instはコプロセッサに引き渡す操作ビットです。コプロセッサへのデータ転送中に付加処理を加える場合にはこのフィールドを設定することでコプロセッサに指示を与えることが可能です。

条件ビット(C)は変化しません。

【発生EIT】

コプロセッサ割り込み(CPI)、コプロセッサディスエーブル例外(CDE)

【命令フォーマット】


```
MVFCP Rdest,CPRsrc,inst,cpid
```

MVTACHI

DSP機能用命令

Move high-order word to accumulator

MVTACHI

【ニーモニック】

```
MVTACHI   Rsrc,Adest
```

【動作】

アキュムレータ～レジスタ間転送

Adest[0 : 31] = Rsrc ;

【機能】

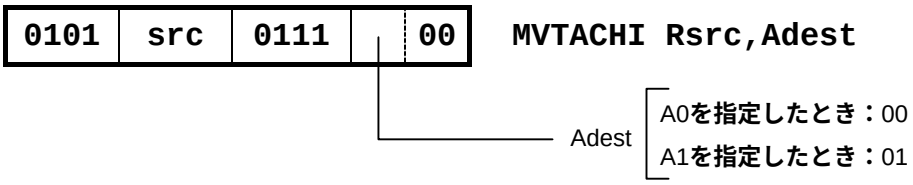
Rsrcの内容をアキュムレータAdestの上位32ビット(ビット0～31)に転送します。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】



MVTACLO

DSP機能用命令

Move low-order word to accumulator

MVTACLO

【ニーモニック】

```
MVTACLO   Rsrc,Adest
```

【動作】

アキュムレータ～レジスタ間転送

Adest [32 : 63] = Rsrc ;

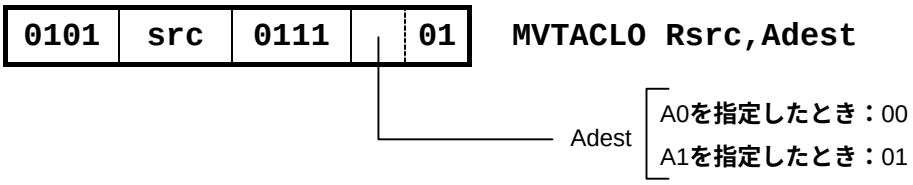
【機能】

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

【発生EIT】

なし

【命令フォーマット】



MVTC

転送命令

Move to control register

MVTC

【ニーモニック】

```
MVTC  Rsrc,CRdest
```

【動作】

レジスタ～制御レジスタ間転送

CRdest = Rsrc ;

【機能】

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

【発生EIT】

特権命令例外（PIE）

【命令フォーマット】

0001	dest	1010	src

MVTC Rsrc,CRdest

MVTCP

コプロセッササポート命令

Move to Coprocessor register

MVTCP

【ニーモニック】

```
MVTCP   Rsrc,CPRdest,inst,cpid
```

【動作】

レジスタ間転送

CPRdest(num) = Rsrc;

【機能】

Rsrcの内容をコプロセッサID(cpid)で指定したコプロセッサのレジスタCPRdestへ転送します。

instはコプロセッサに引き渡す操作ビットです。コプロセッサへのデータ転送中に付加処理を加える場合にはこのフィールドを設定することでコプロセッサに指示を与えることが可能です。

条件ビット(C)は変化しません。

【発生EIT】

コプロセッサ割り込み(CPI)、コプロセッサディスエーブル例外(CDE)

【命令フォーマット】


```
MVTCP Rsrc,CPRdest,inst,cpid
```

NEG

算術演算命令

Negate

NEG

【ニーモニック】

```
NEG  Rdest,Rsrc
```

【動作】

符号反転

Rdest = 0 －(signed) Rsrc ;

【機能】

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

【発生EIT】

なし

【命令フォーマット】

0000	dest	0011	src	NEG Rdest,Rsrc

NOP

分岐命令

No operation

NOP

【ニーモニック】

NOP

【動作】

ノーオペレーション

/* */

【機能】

処理はなにも行いません。次の命令から継続して実行されます。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0111	0000	0000	0000	NOP

NOT

論理演算命令

Logical NOT

NOT

【ニーモニック】

```
NOT  Rdest,Rsrc
```

【動作】

論理否定

Rdest = ~Rsrc ;

【機能】

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

【発生EIT】

なし

【命令フォーマット】

0000	dest	1011	src	NOT Rdest,Rsrc

OR

論理演算命令

OR

OR

【ニーモニック】

```
OR   Rdest,Rsrc
```

【動作】

論理和

Rdest = Rdest | Rsrc ;

【機能】

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

【発生EIT】

なし

【命令フォーマット】

0000	dest	1110	src	OR Rdest,Rsrc

OR3

論理演算命令

OR 3-operand

OR3

【ニーモニック】

OR3 Rdest,Rsrc、#imm16

【動作】

論理和

Rdest = Rsrc | (unsigned short) imm16 ;

【機能】

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

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

1000	dest	1110	src	imm16

OR3 Rdest,Rsrc,#imm16

OPECP　コプロセッササポート命令　OPECP

Operate Coprocessor

【ニーモニック】

```
OPECP   CPRdest,CPRsrc,inst,cpid
```

【動作】

コプロセッサ操作

CPRdest = inst(CPRdest,CPRsrc)(cpid);

【機能】

コプロセッサID(cpid)で指定したコプロセッサに対しコプロセッサ命令(inst)を実行し、その結果をCPdestに格納します。

条件ビット(C)は変化しません。

【発生EIT】

コプロセッサ割り込み(CPI)、コプロセッサディスエーブル例外(CDE)

【命令フォーマット】

1101	CPdest	0111	CPsrc

cpid	0000	inst

```
OPECP CPRdest,CPRsrc,inst,cpid
```

PCMPBZ

比較命令

Parallel compare byte to zero

PCMPBZ

【ニーモニック】

PCMPBZ Rsrc

【動作】

比較

C =(((Rsrc[0:7] ==0) | | (Rsrc[8:15] ==0) | |(Rsrc[16:23] ==0) | | (Rsrc[24:31] ==0)) ? 1 :0)

【機能】

Rsrcをパックされた4つの8ビットとして、4つのうちいずれかの8ビットが0のとき条件ビット(C)が１にセットされます。

【発生EIT】

なし

【命令フォーマット】

0000	0011	0111	src	PCMPBZ Rsrc

RAC

DSP機能用命令

Round accumulator

RAC

【ニーモニック】

```
RAC   Adest,Asrc,#imm1
```

【動作】

飽和処理

```
{ signed64bit tmp;
   tmp = ( signed64bit )Asrc << imm1;
   tmp = tmp + 0x0000 0000 0000 8000;
   if ( tmp > ( signed64bit ) 0x0000 7fff ffff 0000 )
      Adest = 0x0000 7fff ffff 0000;
   else if ( tmp < ( signed64bit ) 0xffff 8000 0000 0000 )
      Adest = 0xffff 8000 0000 0000;
   else
      Adest = tmp & 0xffff ffff ffff 0000;
}
( imm1 = 1, 2; )
```

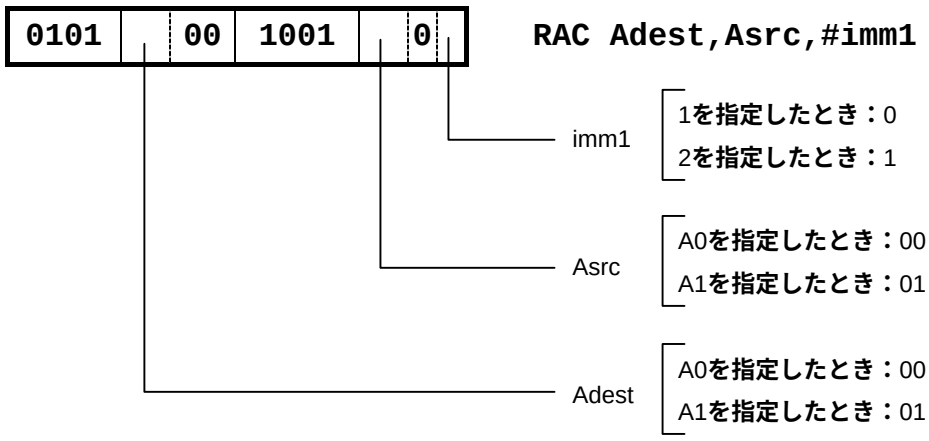
【機能】

アキュムレータAsrc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Adestに格納します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

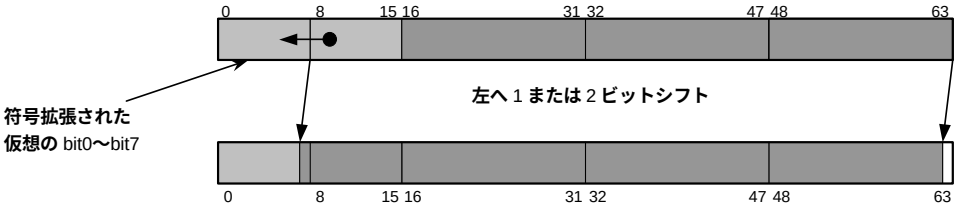


【機能補足説明】

RAC命令は以下のような手順で実行されます。

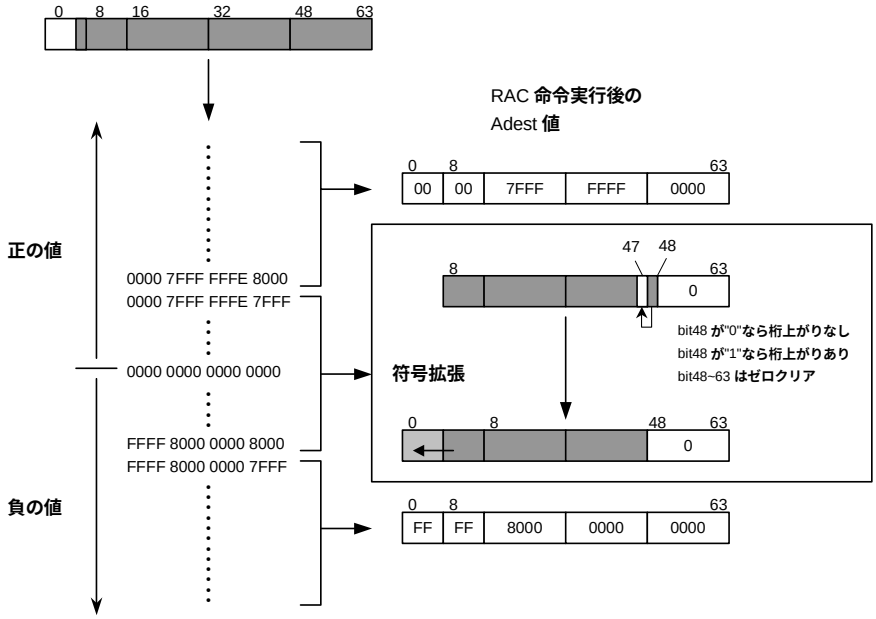
■ 処理1

アキュムレータの値を、imm1で指定したビット(1または2ビット)について左シフトします。



■ 処理2

1ビットまたは2ビットの左シフトが反映された仮想のbit0〜bit7と、シフト後のbit8〜bit63をあわせた64ビットの値にしたがってアキュムレータの値が変化します。



RACH

DSP機能用命令

Round accumulator halfword

RACH

【ニーモニック】

```
RACH  Adest,Asrc,#imm1
```

【動作】

飽和処理

```
{ signed64bit tmp;
   tmp = ( signed64bit )Asrc << imm1;
   tmp = tmp + 0x0000 0000 8000 0000;
   if( tmp > ( signed64bit ) 0x0000 7fff 0000 0000 )
      Adest = 0x0000 7fff 0000 0000;
   else if ( tmp < ( signed64bit) 0xffff 8000 0000 0000 )
      Adest = 0xffff 8000 0000 0000;
   else
      Adest = tmp & 0xffff ffff 0000 0000;
}
( imm1 = 1, 2; )
```

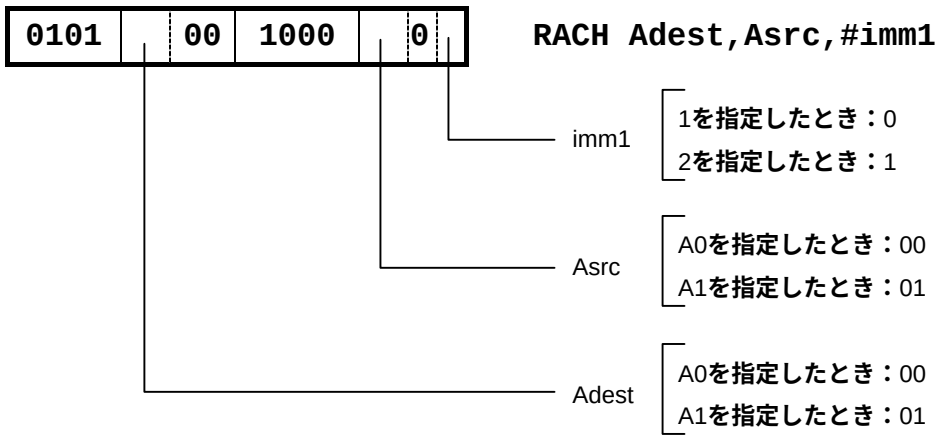
【機能】

アキュムレータの値に対してハーフワードサイズで丸めを行い、その結果をアキュムレータに格納します。
条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

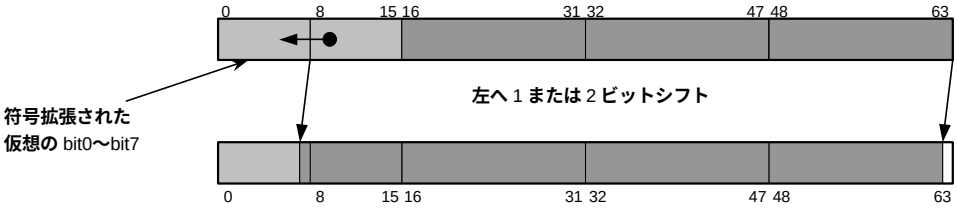


【機能補足説明】

RAC命令は以下のような手順で実行されます。

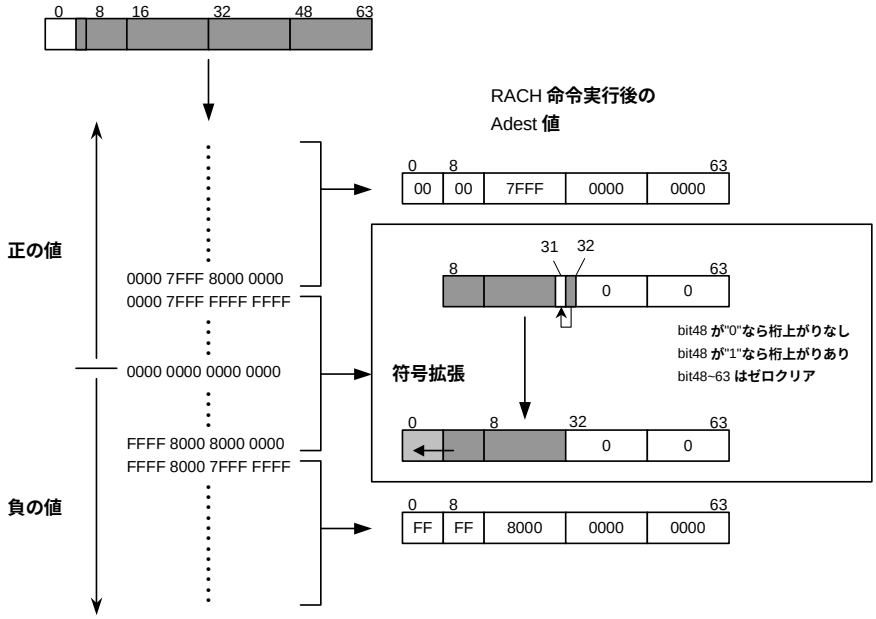
■ 処理1

アキュムレータの値を、imm1で指定したビット(1または2ビット)について左シフトします。



■ 処理2

1ビットまたは2ビットの左シフトが反映された仮想のbit0〜bit7と、シフト後のbit8〜bit63をあわせた64ビットの値にしたがってアキュムレータの値が変化します。



REM

乗除算命令

Remainder

REM

【ニーモニック】

REM Rdest,Rsrc

【動作】

符号付き剰余

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

【機能】

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

条件ビット(C)は変化しません。

Rsrcが0のとき、結果は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	0010	src	0000	0000	0000	0000

REM Rdest,Rsrc

REMB

乗除算命令

Remainder byte

REMB

【ニーモニック】

REMB Rdest,Rsrc

【動作】

符号付き剰余

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

【機能】

RdestをRsrcで割り算し、剰余をRdestに格納します。オペランドは、被除数が符号付き8ビット値として扱われ、上位24ビット(ビット0～23)は無視されます。除数は符号付き32ビット値として扱われます。商は0方向に丸められ、剰余の符号は被除数と等しくなります。

条件ビット(C)は変化しません。

Rsrcが0のとき、Rdestの値は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	0010	src

0000	0000	0001	1000

REMB Rdest,Rsrc

REMH

乗除算命令

Remainder halfword

REMH

【ニーモニック】

```
REMH      Rdest,Rsrc
```

【動作】

符号付き剰余

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

【機能】

RdestをRsrcで割り算し、剰余をRdestに格納します。オペランドは,被除数が符号付き16ビット値として扱われ、上位16ビット(ビット0～15)は無視されます。除数は符号付き32ビット値として扱われます。商は0方向に丸められ、剰余の符号は被除数と等しくなります。

条件ビット(C)は変化しません。

Rsrcが0のとき、Rdestの値は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	0010	src

0000	0000	0001	0000

```
REMH Rdest,Rsrc
```

REMU

乗除算命令

Remainder unsigned

REMU

【ニーモニック】

```
REMU  Rdest,Rsrc
```

【動作】

符号なし剰余

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

【機能】

RdestをRsrcで除算し、剰余をRdestに格納します。オペランドは符号なし32ビット値として扱われます。
条件ビット(C)は変化しません。
Rsrcが0のとき、結果は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	0011	src

0000	0000	0000	0000

```
REMU Rdest,Rsrc
```

REMUB

乗除算命令

Remainder unsigned byte

REMUB

【ニーモニック】

```
REMUB  Rdest,Rsrc
```

【動作】

符号なし剰余

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

【機能】

RdestをRsrcで割り算し、剰余をRdestに格納します。オペランドは被除数が符号なし8ビット値として扱われ、上位24ビット(ビット0～23)は無視されます。除数は符号なし32ビット値として扱われます。

条件ビット(C)は変化しません。

Rsrcが0のとき、Rdestの値は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	0011	src

0000	0000	0001	1000

```
REMUB Rdest,Rsrc
```

REMUH

乗除算命令

Remainder unsigned halfword

REMUH

【ニーモニック】

```
REMUH  Rdest,Rsrc
```

【動作】

符号なし剰余

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

【機能】

RdestをRsrcで割り算し、剰余をRdestに格納します。オペランドは被除数が符号なし16ビット値として扱われ、上位16ビット(ビット0～15)は無視されます。除数は符号なし32ビット値として扱われます。

条件ビット(C)は変化しません。

Rsrcが0のとき、Rdestの値は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	0011	src

0000	0000	0001	0000

```
REMUH Rdest,Rsrc
```

RTE

EIT関連命令

Return from EIT

RTE

【ニーモニック】

RTE

【動作】

EITハンドラからの復帰

SM = BSM ;

IE = BIE ;

PM = BPM

CE = BCE

C = BC ;

PC = BPC & 0xfffffffe ;

【機能】

PSWレジスタ中のBSM, BIE, BPM, BCE およびBC ビットの値をそれぞれSM, IE,,PM,BCE およびCビットに復帰し、BPCで示される番地へ分岐します。

【発生EIT】

特権命令例外（PIE）

【命令フォーマット】

0001	0000	1101	0110

RTE

SADD

DSP機能用命令
Add accumulators

SADD

【ニーモニック】

SADD

【動作】

加算

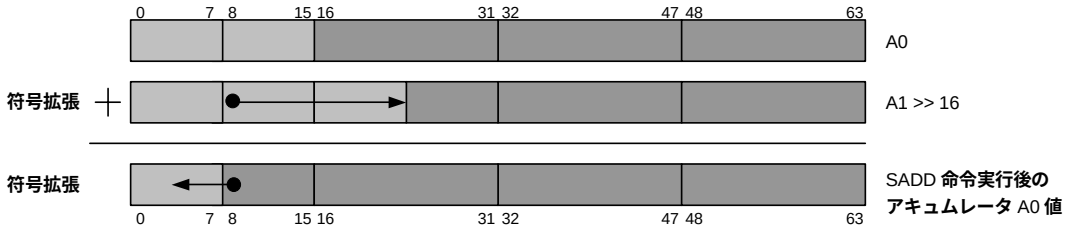
A0 = ((signed) A0 + (signed) ((signed) A1 >> 16));

【機能】

アキュムレータA0とアキュムレータA1を16ビット算術右シフトした値を加算し、結果をA0に格納します。A0と、A1を16ビット右シフトした値は符合付き整数として扱われます。

この命令の実行でアキュムレータA1は変化しません。

条件ビット(C) は変化しません。



【発生EIT】

なし

【命令フォーマット】

0101	0000	1110	0100	SADD

SATB

DSP機能用命令
Saturate word into Byte

SATB

【ニーモニック】

```
SATB     Rdest,Rsrc
```

【動作】

飽和処理

```
{
if ( ( signed char ) 0x7f <= ( signed ) Rsrc )
    Rdest = 0x0000007f;
else if ( ( signed ) Rsrc <= ( signed char ) 0x80; )
    Rdest = 0xffffff80;
else
    Rdest = Rsrc;
};
```

【機能】

Rsrcの値に対してバイトサイズに丸めを行い(飽和処理)、結果をRdestに格納します。
条件ビット(C) は変化しません。

【発生EIT】

なし

【命令フォーマット】

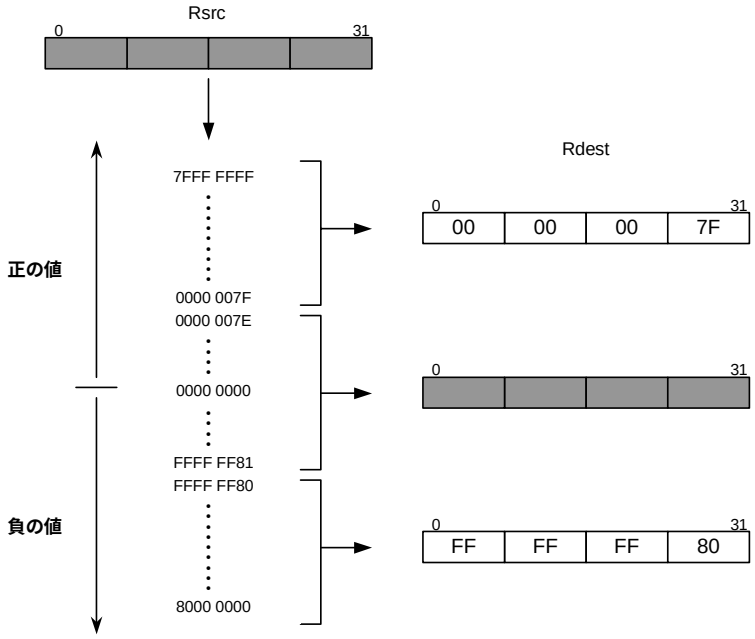
1000	dest	0110	src

0000	0011	0000	0000

SATB Rdest,Rsrc

【機能補足説明】

SATB命令はRsrcの内容にしたがって、Rdestに格納される値が変化します。



SATH

DSP機能用命令

Saturate word into Half-word

SATH

【ニーモニック】

```
SATH    Rdest,Rsrc
```

【動作】

飽和処理

```
{
if ( ( signed short ) 0x7fff <= ( signed ) Rsrc )
    Rdest = 0x00007fff;
else if ( ( signed ) Rsrc <= ( signed short ) 0x8000 )
    Rdest = 0xffff8000;
else
    Rdest = Rsrc;
}
```

【機能】

Rsrcの値に対してハーフワードサイズに丸めを行い（飽和処理）、結果をRdestに格納します。
条件ビット(C) は変化しません。

【発生EIT】

なし

【命令フォーマット】

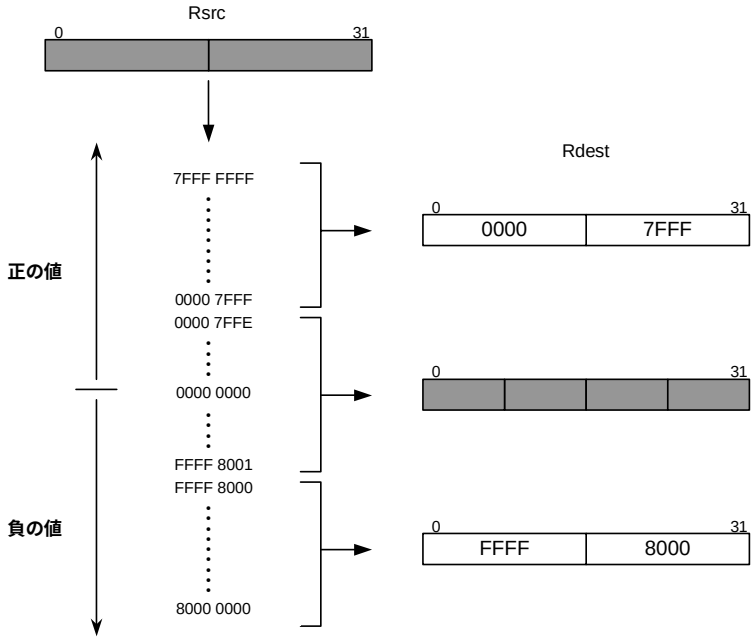
1000	dest	0110	src

0000	0010	0000	0000

```
SATH Rdest,Rsrc
```

【機能補足説明】

SATH命令はRsrcの内容にしたがって、Rdestに格納される値が変化します。



SC 分岐命令 SC

Skip on C-bit

【ニーモニック】

SC

【動作】

条件付きスキップ

if (C ==1)Cancel parallel execution of the next 16-bit instruction ;

【機能】

条件ビット(C)が１の時、同時に実行される予定の16ビット命令をキャンセルし、その次の命令にスキップします。この命令は、並列実行される命令の条件実行(Cが０のとき実行)のための命令であり、並列実行の時にのみ有効です。

ただし、SC命令と同時に実行できる16ビット命令は、両側命令(OS)と右側命令(-S)です。

【発生EIT】

なし

【命令フォーマット】

0111	0100	0000	0001	SC

SETPSW

ビット操作命令

Set PSW

SETPSW

【ニーモニック】

SETPSW #imm8

【動作】

PSWレジスタのSM,IE,PM,CE,Cの任意のビットをセット。

PSW |= (unsigned char) imm8;

【機能】

imm8で指定された8ビット値の各ビットとPSWの下位8ビット(ビット24～31)の各ビットとの論理和をとり、その結果をPSWの下位8ビットにそれぞれ書き込みます。

PSWレジスタがサポートしていないビットには"1"をセットしないでください。

【発生EIT】

特権命令例外（PIE）

【命令フォーマット】

0111	0001	imm8	SETPSW #imm8

SETH

転送命令

Set high-order 16-bit

SETH

【ニーモニック】

```
SETH   Rdest,#imm16
```

【動作】

転送命令

Rdest = (signed short) imm16 << 16 ;

【機能】

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

【発生EIT】

なし

【命令フォーマット】


```
SETH Rdest,#imm16
```

SLL

シフト命令

Shift left logical

SLL

【ニーモニック】

```
SLL  Rdest,Rsrc
```

【動作】

論理左シフト

Rdest = Rdest << (Rsec & 31);

【機能】

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

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	dest	0100	src

SLL Rdest,Rsrc

SLL3

シフト命令
Shift left logical 3-operand

SLL3

【ニーモニック】

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

【動作】

論理左シフト

```
Rdest = Rsrc << ( imm16 & 31 ) ;
```

【機能】

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

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	1100	src	imm16

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

SLLI

シフト命令

Shift left logical immediate

SLLI

【ニーモニック】

```
SLLI   Rdest,#imm5
```

【動作】

論理左シフト

Rdest = Rdest << imm5 ;

【機能】

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

【発生EIT】

なし

【命令フォーマット】

0101	dest	010	imm5

SLLI Rdest,#imm5

SNC

分岐命令

Skip on not C- bit

SNC

【ニーモニック】

SNC

【動作】

条件付きスキップ

if (C ==0)Cancel parallel execution of the next 16- bit instruction ;

【機能】

条件ビット(C)が０の時、同時に実行される予定の16ビット命令をキャンセルし、その次の命令にスキップします。この命令は、並列実行される命令の条件実行(Cが１のとき実行)のための命令であり、並列実行の時にのみ有効です。

ただし、SNC命令と同時に実行できる16ビット命令は、両側命令(OS)と右側命令(-S)です。

【発生EIT】

なし

【命令フォーマット】

0111	0101	0000	0001	SNC

SRA

シフト命令

Shift right arithmetic

SRA

【ニーモニック】

```
SRA  Rdest,Rsrc
```

【動作】

算術右シフト

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

【機能】

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

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	dest	0010	src

SRA Rdest,Rsrc

SRA3

シフト命令

Shift right arithmetic 3-operand

SRA3

【ニーモニック】

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

【動作】

算術右シフト

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

【機能】

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

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	1010	src	imm16

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

SRAI

シフト命令

Shift right arithmetic immediate

SRAI

【ニーモニック】

```
SRAI  Rdest,#imm5
```

【動作】

算術右シフト

Rdest = (signed) Rdest >> imm5 ;

【機能】

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

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	dest	001	imm5	SRAI Rdest,#imm5

SRL

シフト命令

Shift right logical

SRL

【ニーモニック】

```
SRA   Rdest,Rsrc
```

【動作】

論理右シフト

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

【機能】

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

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0001	dest	0000	src

SRL Rdest,Rsrc

SRL3

シフト命令
Shift right logical 3-operand

SRL3

【ニーモニック】

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

【動作】

論理右シフト

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

【機能】

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

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

1001	dest	1000	src	imm16

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

SRLI

シフト命令

Shift right logical immediate

SRLI

【ニーモニック】

```
SRLI    Rdest,#imm5
```

【動作】

論理右シフト

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

【機能】

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

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0101	dest	000	imm5

SRLI Rdest,#imm5

ST　ロード／ストア命令　ST

Store

【ニーモニック】

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

【動作】

レジスタからメモリへのストア

① * (signed int *) Rsrc2 = Rsrc1;
② Rsrc2 += 4, * (signed int *) Rsrc2 = Rsrc1;
③ Rsrc2 -= 4, * (signed int *) Rsrc2 = Rsrc1;
④ * (signed int *) (Rsrc2 + (signed short) disp16) = Rsrc1;

【機能】

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

条件ビット(C)は変化しません。

【発生EIT】

アドレス例外(AE)

【命令フォーマット】

0010	src1	0100	src2	ST Rsrc1,@Rsrc2
0010	src1	0110	src2	ST Rsrc1,@+Rsrc2
0010	src1	0111	src2	ST Rsrc1,@-Rsrc2
1010	src1	0100	src2	disp16

ST Rsrc1,@(disp16,Rsrc2)

STB

ロード／ストア命令

Store byte

STB

【ニーモニック】

① STB Rsrc1,@Rsrc2
② STB Rsrc1,@Rsrc2+
③ STB Rsrc1,@(disp16,Rsrc2)

【動作】

レジスタからメモリへのストア

① *(signed char *)Rsrc2 = Rsrc1;
② *(signed char *)Rsrc2 = Rsrc1, Rsrc2 += 1;
③ *(signed char *)(Rsrc2 + (signed short)disp16) = Rsrc1;

【機能】

① Rsr1cのLSB側のバイトデータを、Rsrc2で指定する番地のメモリにストアします。
② Rsrc1のLSB側のバイトデータを、Rsrc2で指定する番地のメモリにストアし、その後でRsrc2を１インクリメントします。
③ Rsrc1のLSB側のバイトデータを、Rdsrc2と16ビットディスプレースメントで指定する番地のメモリにストアします。ディスプレースメントは、アドレス計算の前に符号拡張されます。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】

0010	src1	0000	src2	STB Rsrc1,@Rsrc2

0010	src1	0001	src2	STB Rsrc1,@Rsrc2+

1010	src1	0000	src2	disp16

STB Rsrc1,@(disp16,Rsrc2)

STH

ロード／ストア命令

Store halfword

STH

【ニーモニック】

① STH Rsrc1,@Rsrc2
② STH Rsrc1,@Rsrc2+
③ STH Rsrc1,@(disp16,Rsrc2)

【動作】

レジスタからメモリへのストア

① *(signed short *)Rsrc2 = Rsrc1;

② *(signed short *)Rsrc2 = Rsrc1, Rsrc2 += 2;

③ *(signed short *)(Rsrc2 + (signed short)disp16) = Rsrc1;

【機能】

① Rsrc1のLSB側のハーフワードデータを、Rsrc2で指定する番地のメモリにストアします。

② Rsrc1のLSB側のハーフワードデータを、Rsrc2で指定する番地のメモリにストアし、その後でRsrc2を2インクリメントします。

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

条件ビット(C)は変化しません。

【発生EIT】

アドレス例外(AE)

【命令フォーマット】

0010	src1	0010	src2	STH Rsrc1,@Rsrc2

0010	src1	0011	src2	STH Rsrc1,@Rsrc2+

1010	src1	0010	src2	disp16

STH Rsrc1,@(disp16,Rsrc2)

SUB

算術演算命令

Subtract

SUB

【ニーモニック】

```
SUB   Rdest,Rsrc
```

【動作】

減算

Rdest = Rdest - Rsrc;

【機能】

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

【発生EIT】

なし

【命令フォーマット】

0000	dest	0010	src

SUB Rdest,Rsrc

SUBV

算術演算命令

Subtract with overflow checking

SUBV

【ニーモニック】

```
SUBV  Rdest,Rsrc
```

【動作】

減算(オーバフローチェック付き)

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

【機能】

Rdestの内容からRsrcの内容を引き算し、結果をRdestに格納します。

条件ビット(C)は引き算の結果がオーバフローしたときセットされ、それ以外はクリアされます。

【発生EIT】

なし

【命令フォーマット】

0000	dest	0000	src

SUBV Rdest,Rsrc

SUBX

算術演算命令

Subtract with borrow

SUBX

【ニーモニック】

```
SUBX  Rdest,Rsrc
```

【動作】

減算(ボロー付き)

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

【機能】

Rdestの内容から(Rsrcの内容+条件ビット(C))の値を引き算し、結果をRdestに格納します。

条件ビット(C)は引き算の結果が符号なし32ビット整数として表現できないときセットされ、それ以外はクリアされます。

【発生EIT】

なし

【命令フォーマット】

0000	dest	0001	src

SUBX Rdest,Rsrc

TRAP

EIT関連命令

Trap

TRAP

【ニーモニック】

TRAP #imm4

【動作】

TRAPの発生

BPC = NextPC;　(NextPCは次命令PCを指します)

BSM = SM;

BIE = IE;

BPM = PM

BCE = CE

BC = C;

IE = 0;

PM =0;

CE = 0

C = 0;

call_trap_handler(imm4);

【機能】

指定された番号のトラップを発生します。

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

【発生EIT】

トラップ(TRAP)

【命令フォーマット】

0001	0000	1111	imm4	TRAP #imm4

UNLOCK　ロード／ストア命令　UNLOCK

Store unlocked

【ニーモニック】

```
UNLOCK  Rsrc1,@Rsrc2
```

【動作】

ロック解除付きストア

```
if ( LOCK == 1 ) { * ( signed int *) Rsrc2 = Rsrc1; }
LOCK = 0;
```

【機能】

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

【発生EIT】

アドレス例外(AE)

【命令フォーマット】

0010	src1	0101	src2

UNLOCK Rsrc1,@Rsrc2

XOR

論理演算命令

Exclusive OR

XOR

【ニーモニック】

```
XOR  Rdest,Rsrc
```

【動作】

排他的論理和

Rdest = Rdest ^ Rsrc;

【機能】

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

【発生EIT】

なし

【命令フォーマット】

0000	dest	1101	src

XOR Rdest,Rsrc

XOR3

論理演算命令

Exclusive OR 3-operand

XOR3

【ニーモニック】

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

【動作】

排他的論理和

Rdest = Rsrc ^ (unsigned short) imm16;

【機能】

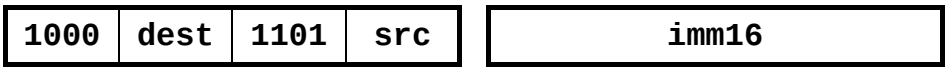
Rsrcと16ビット即値の対応するビットの排他的論理和を計算し、Rdestに格納します。16ビット即値は演算前に32ビットにゼロ拡張されます。

条件ビット(C)は変化しません。

【発生EIT】

なし

【命令フォーマット】




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

3.3 BCL,BNCL命令の注意事項

BCL、BNCL命令をワード境界に配置し、それに続く後半16ビット命令が逐次実行命令だった場合、後半16ビット命令が実行されるか否かは、Cビットの値によります。

BCL、BNCL命令をワード境界に配置し、それに続く後半16ビットに命令を配置する場合は、充分注意してください。

下記のような命令コード配置の場合、Cビット=1でBCL(またはBNCL)が分岐した場合、後半16ビット命令は実行されません。OPSP-CPUの分岐先はワード境界であるためです。(RTE命令実行時を除く)

Cビット=0でBCL(またはBNCL)が分岐しなかった場合は、後半16ビット命令を実行します。

0	BCLまたはBNCL	0	xxxx命令

任意の命令が実行されるか否かは、Cビットの値によります。

なお、下記のように後半16ビット命令(yyyy命令)が並列実行命令だった場合には、yyyy命令はBCL(またはBNCL)と同時に実行されます。

0	BCLまたはBNCL	1	yyyy命令

yyyy命令は並列実行されます。

3.4 並列実行時の例外・トラップ処理

並列実行時における例外・トラップ処理を以下に示します。

表3.4.1 並列実行時の例外・トラップ処理

Oパイプ（左側）命令	Sパイプ（右側）命令	動作
RIE	任意の命令	RIE発生、O側、S側共に実行されない
RIE	RIE	RIE発生、O側、S側共に実行されない
任意の命令	RIE	RIE発生、O側、S側共に実行されない
PIE	任意の命令	PIE発生、O側、S側共に実行されない
PIE	RIE	RIE発生、O側、S側共に実行されない
AE	任意の命令	AE発生、O側、S側共に実行されない
AE	RIE	RIE発生、O側、S側共に実行されない
TRAP	任意の命令	TRAP発生、O側、S側共に実行される
TRAP	RIE	RIE発生、O側、S側共に実行されない

レイアウトの都合上、このページは白紙です。

付録

付録1　パイプライン処理機構

付録1.1　パイプライン処理機構の概要

OPSP CPUコアは2本のパイプライン（OパイプとSパイプ）をもっています。2本のパイプラインはともに5段のパイプラインステージで構成されています。OPSP CPUコアの並列実行は、この2本のパイプラインを同時に使って行います。（同時に実行できる命令の組み合わせは「2.5　並列実行処理」をご覧ください。）

■ Oパイプラインの動作と各ステージの概要

(1) IFステージ（命令フェッチステージ）

命令フェッチを行うステージです。メモリ（キャッシュ）から命令をフェッチします。

OPSP CPUは命令キューを備えており、D（デコード）ステージのデコード処理完了とは無関係に、命令キューが一杯になるまでフェッチを続けます。

(2)Dステージ（デコードステージ）

Dステージは、命令のデコード処理（DEC）を行います。このとき、レジスタの読み出し（RF）を行い、直前の命令の演算結果を参照する処理の場合は、バイパス処理（BYP）を行います。ただし、バイパス処理は直前の命令がレジスタ間の転送、演算、DSP機能用命令の場合に行います。

(3)Eステージ（実行ステージ）

演算やアドレス計算など（OP）を行います。

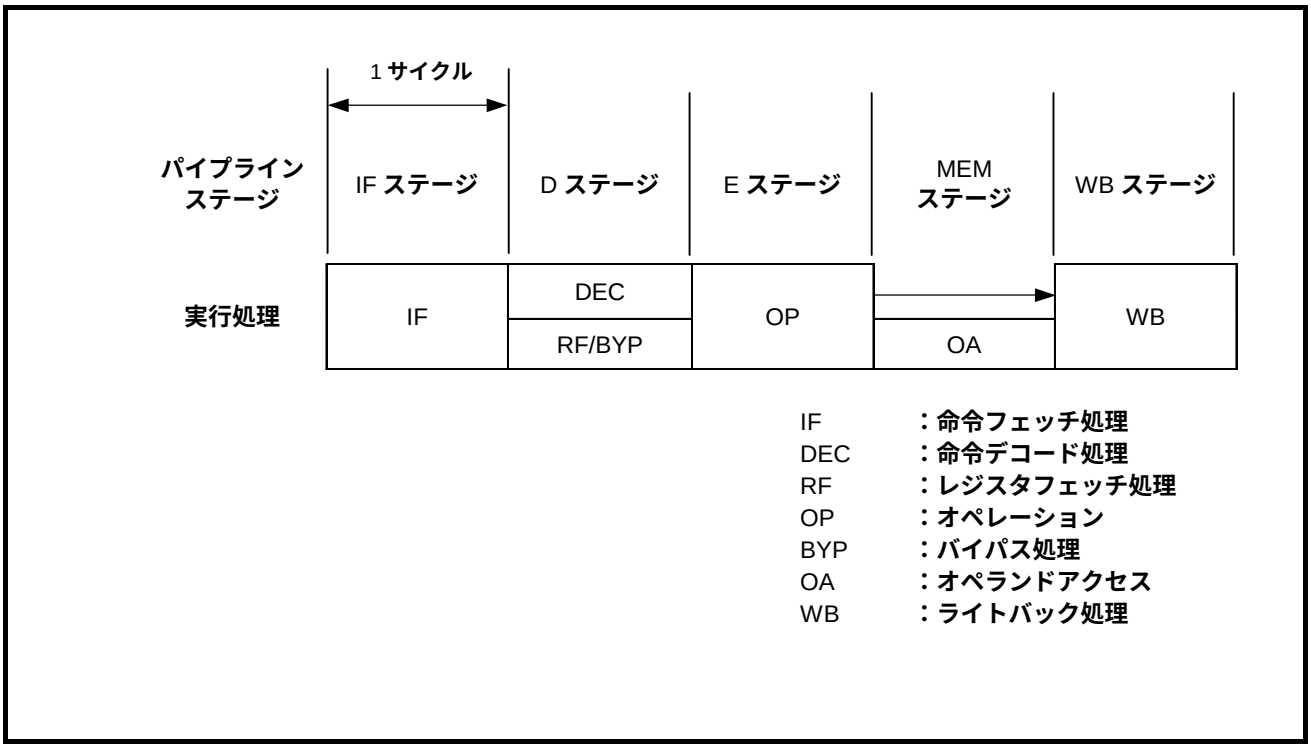
(4)MEMステージ（メモリアクセスステージ）

オペランドのアクセス（OA）を行います。ロード／ストア命令実行時のみ、このステージを使用します。

(5)WBステージ（ライトバックステージ）

演算結果やフェッチしてきたデータをレジスタに書き込みます。

付図1.1にOパイプラインの構成とその動作を示します。



付図1.1 Oパイプライン構成と動作

■ Sパイプラインの動作と各ステージの概要

(1) IFステージ（命令フェッチステージ）

命令フェッチを行うステージです。メモリ（キャッシュ）から命令をフェッチします。

OPSP CPUは命令キューを備えており、D（デコード）ステージのデコード処理完了とは無関係に、命令キューが一杯になるまでフェッチを続けます。

(2)Dステージ（デコードステージ）

Dステージは、命令のデコード処理（DEC）を行います。このとき、レジスタの読み出し（RF）を行い、直前の命令の演算結果を参照する処理の場合は、バイパス処理（BYP）を行います。ただし、バイパス処理は直前の命令がレジスタ間の転送、演算、DSP機能用命令の場合に行います。

(3)E1ステージ（実行ステージ1）

演算やレジスタ、アキュムレータへのデータ転送（OP1）を行います。

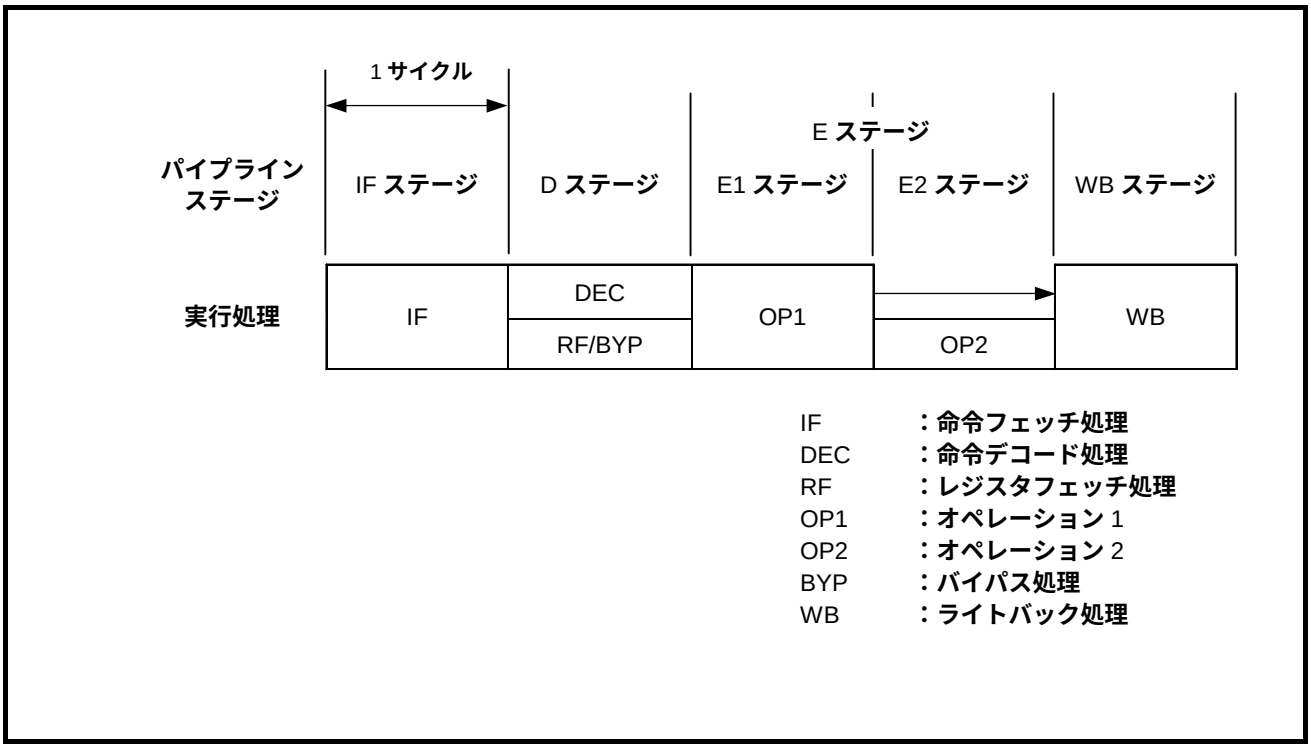
(4)E2ステージ（実行ステージ2）

DPS機能用命令でアキュムレータにデータを書き込む命令は、このステージを使用します（OP2）。このときE1及びE2の2つのステージを使用するため、実行ステージとして2サイクルかかります。演算結果をアキュムレータに書き込まない命令は、このステージは使用されず次のWBステージに進みます。

(5)WBステージ（ライトバックステージ）

演算結果をレジスタまたはアキュムレータに書き込みます。

付図1.2にSパイプラインの構成とその動作を示します。



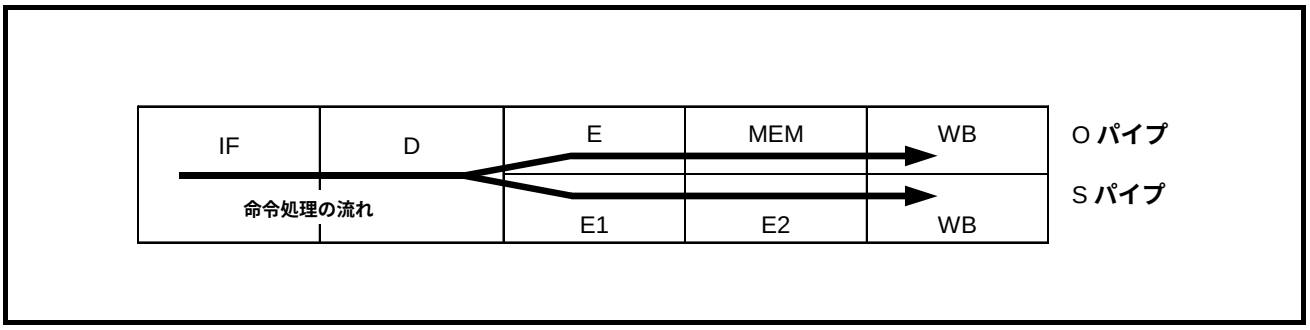
付図1.2 Sパイプライン構成と動作

付録1.2 OパイプおよびSパイプの命令処理の流れ

Oパイプ、Sパイプの2つのパイプラインにおけるIFステージ、Dステージはともに共有しており、どの命令もDステージまでは区別なく処理されます。Dステージで命令をどちらのパイプラインに発行するかが決定されます。

並列実行の場合は、Oパイプ、Sパイプの両方に対して命令が発行されます。並列実行でない場合は、いずれかのパイプラインでEステージ以降の処理がされます。

以下にパイプラインの命令処理の流れを示します。

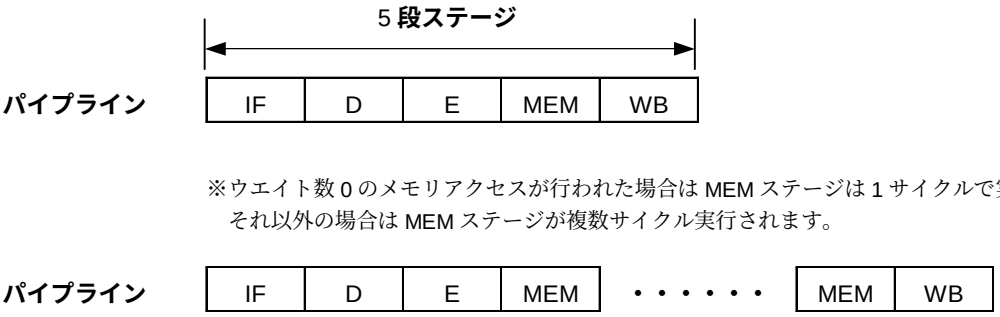


付図1.3 Oパイプおよびパイプの命令処理の流れ

付録1.3 命令とパイプライン処理

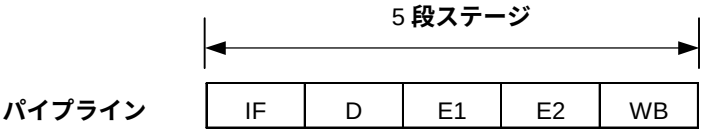
OPSP CPUのパイプラインは5段のステージで構成されます、MEMステージはロード・ストア命令でのみ、またE2ステージはDSP機能命令でアキュムレータにデータを書き込む命令のみでしか使用されないため、それ以外の命令では5段のパイプライン処理となります。

■ロード／ストア命令の場合

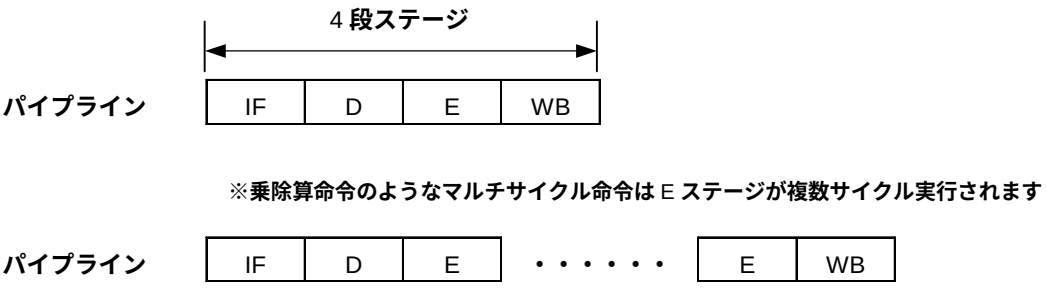


■DSP 機能命令の場合

（アキュムレータにデータを書き込む命令）



■その他の命令の場合



付図1.4 命令とパイプライン処理

付録1.4 並列実行のパイプライン処理

OPSP CPUコアは２本のパイプライン(Oパイプ、Sパイプ)を同時に使用することで、並列実行処理を行います。

並列実行対象となる16ビット命令ペアはIFステージからDステージまでは、同時にパイプラインステージを移動します。Eステージに命令が発行された後は、OパイプとSパイプは独立したパイプライン動作を行います。並列実行時のパイプライン動作の例を示します。

＜ケース 1＞ 左側命令（O−）と右側命令（−S）の並列実行時

LD	**R1,@R2**	IF	D	E	MEM	WB	O **パイプ**
MULHI	**R3,R4**	IF	D	E1	E2	WB	S **パイプ**

＜ケース 2＞ 左側命令(O−)と両側命令(OS)の並列実行

LD	**R1,@R2**	IF	D	E	MEM	WB	O **パイプ**
ADD	**R3,R4**	IF	D	E1	WB		S **パイプ**

＜ケース 3＞ 両側命令(OS)と右側命令(−S)の並列実行

ADD	**R1,R2**	IF	D	E	WB		O **パイプ**
MULHI	**R3,R4**	IF	D	E1	E2	WB	S **パイプ**

＜ケース 4＞ 両側命令(OS)と両側命令(OS)の並列実行

ADD	**R1,R2**	IF	D	E	WB		O **パイプ**
ADD	**R3,R4**	IF	D	E1	WB		S **パイプ**

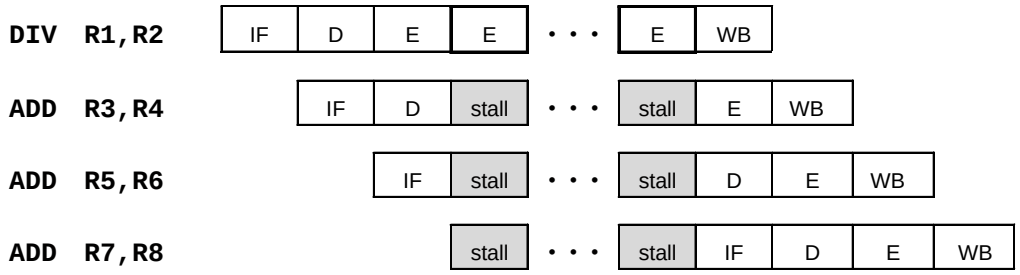
付図1.5 並列実行のパイプライン処理

付録1.5 パイプラインの基本動作

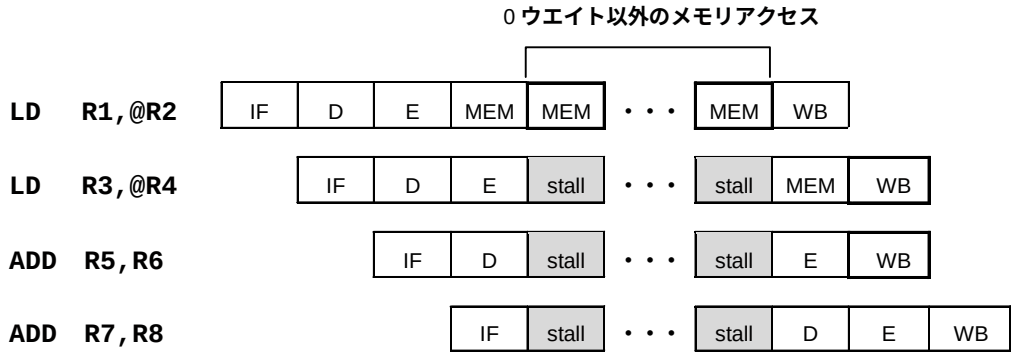
理想的なパイプライン処理では、各ステージの実行サイクル数は１ですが、各ステージでの処理や分岐命令の実行などによりパイプライン動作が乱れることがあります。

以下に基本的な動作をケース別に示します。

<ケース 1>：E ステージの実行に複数サイクルを要する命令の実行時



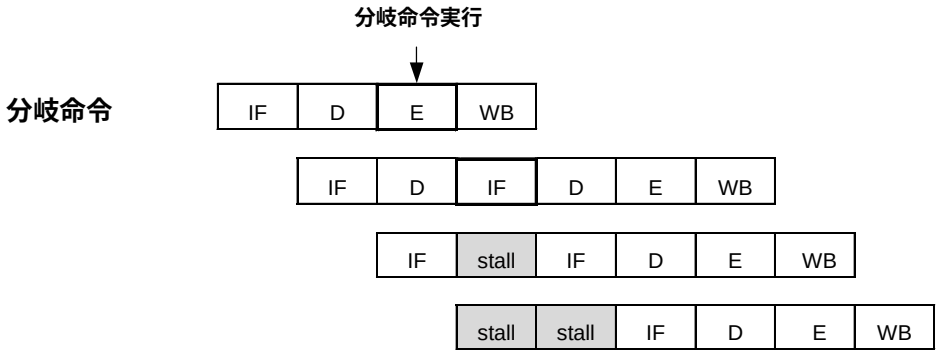
<ケース 2>：オペランドアクセスが 1 サイクルで終了しない場合



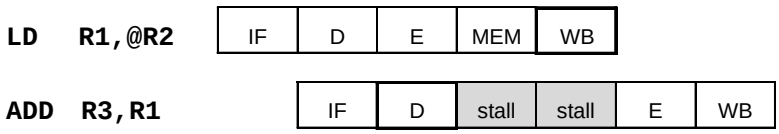
stall：パイプラインストール

付図1.6 パイプライン動作が乱れるケース-1

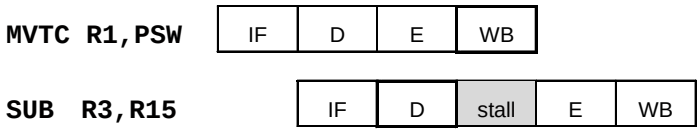
<ケース 3>：分岐命令を実行した時　（条件分岐命令で分岐しなかった場合を除く）



<ケース 4>：メモリからリードしたオペランドを後続命令が使用する時



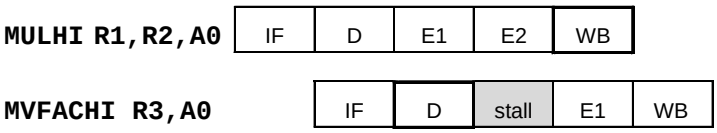
<ケース 5>：MVTC 命令で PSW の SM ビットに書き込み後、後続命令が R15 を読み出す場合



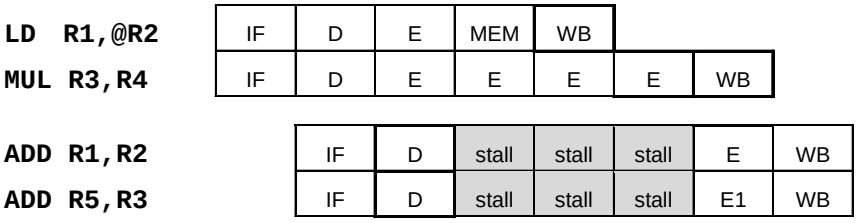
stall：パイプラインストール

付図1.7　パイプライン動作が乱れるケース-2

<ケース 6>：アキュムレータにデータを書き込む命令(DSP 機能用命令の MULHI など)の実行後、
同じアキュムレータを MVFAC 命令で読み出す場合



<ケース 7>：並列実行処理において、ケース 1 とケース 4 が同時に発生した場合



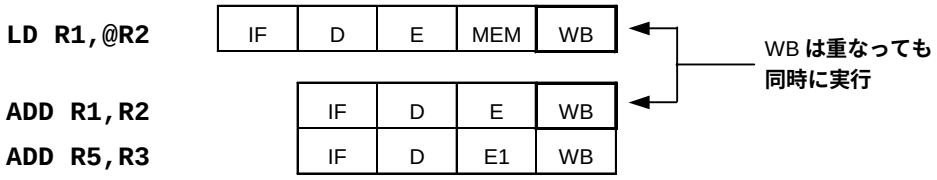
stall：パイプラインストール

付図1.8 パイプライン動作が乱れるケース-3

下記の場合については、特殊なケースとして、パイプライン動作は乱れません。

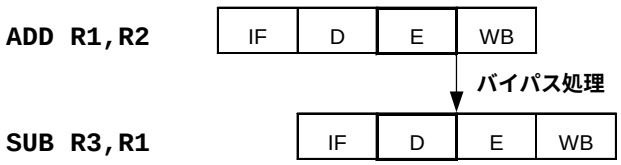
■ ロード命令とロード命令以外の WB ステージが重なっている場合

(同時に書き込めるので、パイプライン処理は乱れません)



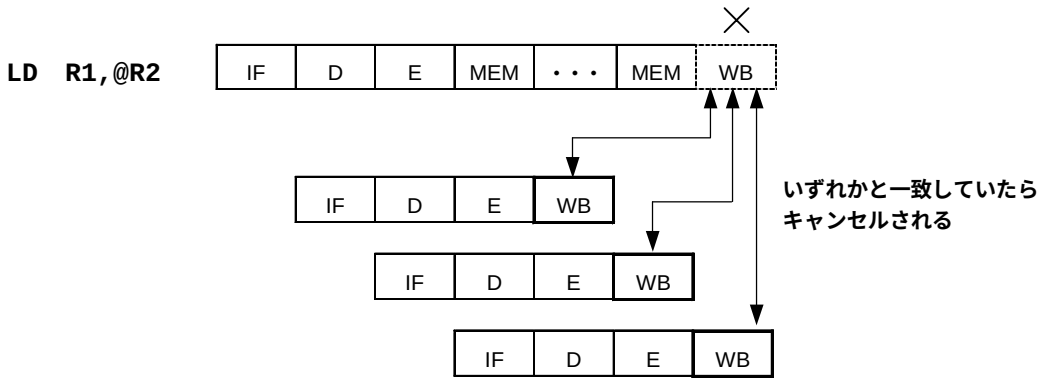
■ 先行命令が書き込んだレジスタを後続命令が使用する場合

(レジスタ間演算の場合は、バイパス機構によりパイプライン処理は乱れません)



■ ロード命令を終了する前に、後続命令が同じレジスタへ書き込みを行った場合

(ロード命令の WB ステージ実行はキャンセル)



付図1.9 パイプライン動作が乱れない特殊なケース

付録2　命令処理時間

OPSP CPUは通常Eステージにおける命令実行サイクル数を命令処理時間として代表しますが、パイプライン動作によっては、それ以外のステージの影響によって、命令処理時間が変動することがあります。

以下に、OPSP CPUの各パイプラインステージの命令処理時間を示します。

命令	各ステージにおける実行サイクル数				
	IF	D	E（注3）	MEM	WB
ロード命令（LD,LDB,LDUB,LDH,LDUH,LOCK）	R（注2）	1	1	R（注2）	1
ストア命令（ST,STB,STH,UNLOCK）	R（注2）	1	1	W（注2）	-
BSET、BCLR命令	R（注2）	1	1	R+W（注2）	-
乗算命令（MUL）	R（注2）	1	4	-	1
除算／剰余命令（DIVB,DIVUB,REMB,REMUB）	R（注2）	1	13	-	1
除算／剰余命令（DIVH,DIVUH,REMH,REMUH）	R（注2）	1	21	-	1
除算／剰余命令（DIV,DIVU,REM,REMU）	R（注2）	1	37	-	1
DSP機能用命令[1]（アキュムレータにデータを書き込む場合）	R（注2）	1	2（1）（注4）	-	1
DSP機能用命令[2]（アキュムレータにデータを書き込まない場合）（注1）	R（注2）	1	1	-	1
上記以外の命令（DSP機能用命令、BTST、SETPSW、CLRPSW命令を含む）	R（注2）	1	1	-	1

注1．アキュムレータにデータを書き込まないDSP機能用命令には、MVFACHI、MVFACLO、MVFACMI、SATB、SATHがあります。

注2．R,Wで示される実行サイクル数は、機種依存です。

注3．Oパイプで実行される命令の場合Eステージの実行サイクル数を示します。また、Sパイプで実行される命令の場合、E1とE2ステージを加算した実行サイクル数を示します。

注4．DSP機能用命令[1]のEステージの実行サイクル数は2サイクル必要ですが、パイプライン処理が行われますので、実際のスループットは1になります。

ルネサス 32 **ビットオープンプラットフォームシンセサイザブルプロセッサ**
ソフトウエアマニュアル
OPSP

発行年月日 2004 **年** 3 **月** 1 **日** Rev.1.00

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

OPSP
ソフトウエアマニュアル

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