EPSON
EXCEED YOUR VISION

CMOS 16-BIT SINGLE CHIP MICROCONTROLLER

S1C17 Family

S1C17コアマニュアル

セイコーエプソン株式会社

Rev. 1.2

本資料のご使用につきましては、次の点にご留意願います。

本資料の内容については、予告なく変更することがあります。

1. 本資料の一部、または全部を弊社に無断で転載、または、複製など他の目的に使用することは堅くお断りいたします。
2. 本資料に掲載される応用回路、プログラム、使用方法等はあくまでも参考情報であり、これらに起因する第三者の知的財産権およびその他の権利侵害あるいは損害の発生に対し、弊社はいかなる保証を行うものではありません。また、本資料によって第三者または弊社の知的財産権およびその他の権利の実施権の許諾を行うものではありません。
3. 特性値の数値の大小は、数直線上の大小関係で表しています。
4. 製品および弊社が提供する技術を輸出等するにあたっては「外国為替および外国貿易法」を遵守し、当該法令の定める手続きが必要です。大量破壊兵器の開発等およびその他の軍事用途に使用する目的をもって製品および弊社が提供する技術を費消、再販売または輸出等しないでください。
5. 本資料に掲載されている製品は、生命維持装置その他、きわめて高い信頼性が要求される用途を前提としていません。よって、弊社は本(当該)製品をこれらの用途に用いた場合のいかなる責任についても負いかねます。
6. 本資料に掲載されている会社名、商品名は、各社の商標または登録商標です。

ー 目 次 ー

1 概要

S1C17コアはセイコーエプソンオリジナルの16ビットRISCプロセッサです。
低消費電力、最大60MHz～90MHzの高速動作、16Mバイトの広いアドレス空間、主要命令の1クロック実行、省ゲート設計を特長とし、上位のS1C33コアほどのデータ処理能力を必要としないような用途、たとえば、8ビットCPUがよく使われるコントローラやシーケンサ等への組み込み用に最適です。
コプロセッサインタフェースも内蔵しており、追加演算機能の実装にも対応しています。

また、S1C33 Familyと同様なIDEワークベンチ、Cコンパイラ、シリアルICE、デバッガなどのソフトウェア開発環境の提供により、アプリケーションソフトウェア開発を支援します。

1.1 特長

プロセッサ形式
- セイコーエプソンオリジナル16ビットRISCプロセッサ
- 0.35～0.15μm低電力CMOSプロセステクノロジ

動作周波数
- 最大90MHz(機種およびプロセス技術により異なります。)

命令セット
- コード長　　16ビット固定長
- 命令数　　基本命令111個(全184命令)
- 実行サイクル　主要命令は1サイクルで実行
- 即値拡張命令　即値を24ビットまで拡張
- Cによる開発用に最適化されたコンパクトかつ高速な命令セット

レジスタセット
- 24ビット汎用レジスタ×8
- 24ビット特殊レジスタ×2
- 8ビット特殊レジスタ×1

メモリ空間, バス
- 最大16Mバイトのメモリ空間(24ビットアドレス)
- 命令バス(16ビット)とデータバス(32ビット)を分離したハーバードアーキテクチャ

割り込み
- リセット、NMI、32種類の外部割り込みに対応
- アドレス不整割り込み
- デバッグ割り込み
- ベクタテーブルからベクタを読み込み、割り込み処理ルーチンへ直接分岐
- ベクタ番号によるソフトウェア割り込みを発生可能(全ベクタ番号を指定可能)

パワーセーブ
- HALT(`halt`命令)
- SLEEP(`slp`命令)

コプロセッサインタフェース
- ALU命令を強化可能

2 レジスタ

S1C17コアは、8本の汎用レジスタおよび3本の特殊レジスタを内蔵しています。

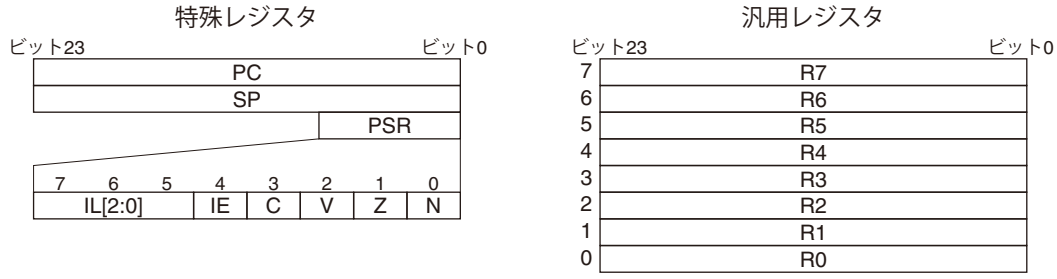


図2.1 レジスタの構成

2.1 汎用レジスタ(R0〜R7)

シンボル	レジスタ名	サイズ	R/W	初期値
R0~R7	汎用レジスタ	24ビット	R/W	0x000000

8本のR0〜R7レジスタは、データの演算、データの転送、メモリのアドレッシング等、使用目的が固定されていない24ビット長の汎用レジスタです。これらのレジスタの内容は、すべて24ビットのデータまたはアドレスとして扱われ、8ビット、16ビットデータはロード命令またはコンバージョン命令で符号拡張またはゼロ拡張可能です。汎用レジスタをアドレス参照に用いる場合は、ひとつのレジスタで24ビットのメモリ空間を直接アクセスすることができます。
イニシャルリセット時、汎用レジスタは0に設定されます。

2.2 プログラムカウンタ(PC)

シンボル	レジスタ名	サイズ	R/W	初期値
PC	プログラムカウンタ	24ビット	R	(リセットベクタ)

プログラムカウンタ(以下、PC)は、実行命令のアドレスを保持する24ビット長のカウンタです。
PCの値は、次に実行されるアドレスを示しています。S1C17コアの命令は16ビット固定長のため、PCの最下位ビット(ビット0)は常に0となります。S1C17コアではPCをプログラムで参照することが可能です。ただし、データ転送命令で変更することはできません。なお、`ld.a %rd,%pc`命令(ディレイドスロット命令として実行可能)を実行した場合、指定レジスタには「この`ld`命令のPC値 + 2」がロードされます。
イニシャルリセット時、PCにはTTBRで示されるベクタテーブルの先頭に書き込まれているリセットベクタ(アドレス)がロードされ、そのアドレスからプログラムが実行されます。

図2.2.1 プログラムカウンタ(PC)

2.3 プロセッサステータスレジスタ(PSR)

シンボル	レジスタ名	サイズ	R/W	初期値
PSR	プロセッサステータスレジスタ	8ビット	R/W	0x00

プロセッサステータスレジスタ(以下、PSR)は、プロセッサ内部の状態を示す8ビット長のレジスタです。PSRには命令の実行によって変化した内部ステータス情報が格納されます。これらの内部ステータスは算術演算や分岐命令などで参照され、プログラムを構成する上で重要な情報として用いられます。PSRの内容は、IEビット以外をプログラムで直接変更することはできません。
PSRはプログラムの実行に影響を与えるため、割り込みが発生したときには(デバッグ割り込みを除く)、PSRをスタックに退避して内容を保存します。また、IEビット(ビット4)は0にクリアされます。割り込み処理からは`reti`命令で復帰します。`reti`命令はPCを割り込みが発生した位置に戻すと同時に、PSRの値もスタックからレジスタに戻します。

	7	6	5	4	3	2	1	0
PSR	IL[2:0]			IE	C	V	Z	N
初期値	0	0	0	0	0	0	0	0

図2.3.1 プロセッサステータスレジスタ(PSR)

IL[2:0](ビット[7:5]) Interrupt Level

プロセッサの割り込みレベルを示します。マスク可能な割り込み要求は、その割り込みレベルがILビットフィールドに設定されたレベルより高い場合にのみ受け付けられます。また、1つの割り込みを受け付けるとILビットフィールドがその割り込みレベルに設定され、それ以降はILビットフィールドを再設定するか、割り込み処理ルーチンを`reti`命令で終了するまで、同じレベルの割り込み要求が再度発生してもマスクされます。

IE(ビット4): Interrupt Enable

マスク可能な割り込みを受け付けるか禁止するかを制御します。IEビットが1のとき、プロセッサはマスク可能な割り込みを許可します。IEビットが0のときはマスク可能な割り込みを禁止します。割り込みを受け付けると、プロセッサはPSRをスタックに退避させた後、このフラグを0にクリアします。ただし、デバッグ割り込みではPSRはスタックには退避されず、このフラグもクリアされません。

C(ビット3): Carry

キャリーまたはボローを示します。加算命令または減算命令において演算結果を符号なし16ビットまたは24ビット整数として扱う場合に、命令の実行結果が符号なし16ビットまたは24ビット整数の範囲を超えると1にセットされます。結果が符号なし16ビットまたは24ビット整数の範囲内の場合は0にクリアされます。
また、シフト命令によっても値が変わります。
Cフラグがセットされる条件は以下のとおりです。

(1) 16ビット整数の加算命令(条件実行除く)で、演算結果が符号なし16ビット整数の最大値0xffffよりも大きい値となる加算を実行した場合

(2) 16ビット整数の減算命令(条件実行除く)で、演算結果が符号なし16ビット整数の最小値0x0000よりも小さい値となる減算を実行した場合

(3) 16ビット整数の比較命令(条件実行除く)で、演算結果が符号なし16ビット整数の最小値0x0000よりも小さい値となる比較(減算)を実行した場合

(4) 24ビット整数の比較命令(条件実行除く)で、演算結果が符号なし24ビット整数の最小値0x000000よりも小さい値となる比較(減算)を実行した場合

(5) 右論理シフト命令で、レジスタのビット0が1のレジスタのシフトを実行した場合

(6) 左論理シフト命令で、レジスタのビット15が1のレジスタのシフトを実行した場合

(7) 右算術シフト命令で、レジスタのビット0が1のレジスタのシフトを実行した場合

V(ビット2): OVerflow

オーバーフローまたはアンダーフローが発生したことを示します。加算命令または減算命令において演算結果を符号付き16ビット整数として扱う場合に、命令の実行によりオーバーフローまたはアンダーフローが発生すると1にセットされます。加算または減算結果が符号付き16ビット範囲内の場合は0にクリアされます。論理演算命令の実行によっても0にクリアされます。
また、Vフラグは16ビット算術演算命令でセットされ、24ビット算術演算命令ではセットされません。
Vフラグがセットされる条件は以下のとおりです。

(1)負の整数と負の整数を加算した場合に、結果の符号ビット(最上位ビット)が0(正)になった場合

(2)正の整数と正の整数を加算した場合に、結果の符号ビット(最上位ビット)が1(負)になった場合

(3)正の整数から負の整数を減算した場合に、結果の符号ビット(最上位ビット)が1(負)になった場合

(4)負の整数から正の整数を減算した場合に、結果の符号ビット(最上位ビット)が0(正)になった場合

Z(ビット1): Zero

結果が0であることを示します。論理演算、算術演算、シフト命令の実行結果がゼロの場合、1にセットされ、ゼロ以外のときは0にクリアされます。
また、Zフラグは16ビット算術演算命令と24ビットの比較命令でセットされ、24ビット加算、減算命令ではセットされません。

N(ビット0): Negative

符号を示します。論理演算、算術演算、シフト命令の実行結果の最上位ビット(ビット15)がNフラグにコピーされます。
また、Nフラグは16ビット算術演算命令でセットされ、24ビット算術演算命令ではセットされません。

2.4 スタックポインタ(SP)

シンボル	レジスタ名	サイズ	R/W	初期値
SP	スタックポインタ	24ビット	R/W	0x000000

スタックポインタ(以下、SP)は、スタックの先頭アドレスを保持する24ビットレジスタです。
スタックはシステムのRAM上に任意に配置可能な領域で、初期設定でSPにスタックの先頭アドレスをセットします。また、SPの下位2ビットは0固定となり書き込みは行えません。したがって、SPで指定できるアドレスは32ビット境界となります。

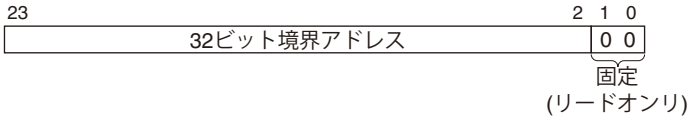


図2.4.1 スタックポインタ(SP)

2.4.1 スタック領域について

スタックとして使用可能な領域サイズは、RAMのサイズと通常のRAMデータが占有する領域サイズによって制限されます。両者が重複しないように注意が必要です。
また、SPはイニシャルリセットにより0x000000に設定されますので、初期化ルーチン内の先頭部分でアドレス(スタック最終アドレス + 4、下位2ビットは0)を書き込んでください。アドレスの書き込みはロード命令で行えます。スタック設定前に割り込みが発生するとPCやPSRが不定の位置にセーブされ、プログラムの正常な動作が保証できません。このためソフトウェア制御が不可能なNMIは、SPが初期化されるまでハードウェアによってマスクされるようになっています。

2.4.2 サブルーチンコール/リターン時の動作

サブルーチンコール命令callまたはcallaはスタックを4バイト使用します。call/calla命令はサブルーチンに分岐する前にPCの内容(リターンアドレス)をスタックにセーブします。セーブされたアドレスはサブルーチンの最後にret命令によってPCに戻され、プログラムはcall/calla命令の次のアドレスに戻ります。

call/calla命令の動作

(1) SP = SP - 4
(2) PC + 2 → [SP]

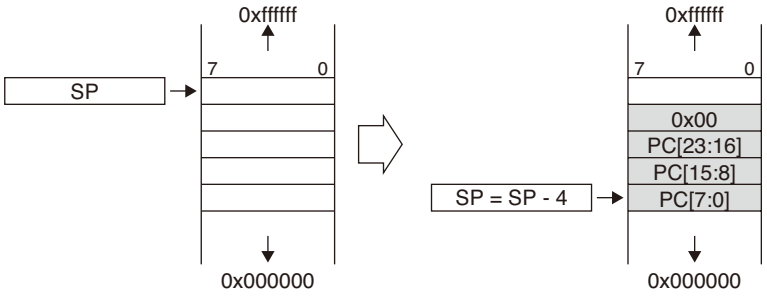


図2.4.2.1 SPとスタック(1)

ret命令の動作

(1) [SP] → PC
(2) SP = SP + 4

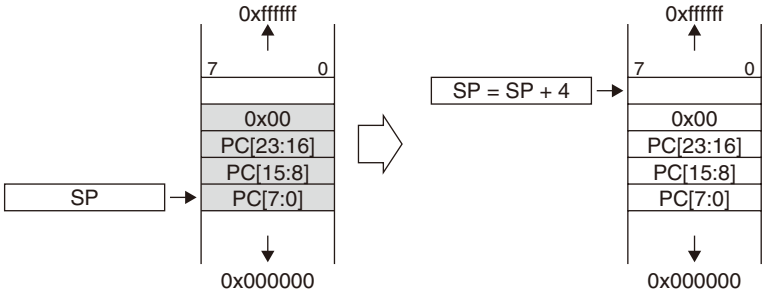


図2.4.2.2 SPとスタック(2)

2.4.3 割り込み発生時の動作

割り込み、int/int1命令によるソフトウェア割り込み等が発生すると、プロセッサは割り込み処理に入ります。
プロセッサはそれぞれの割り込み処理ルーチンに分岐する前にPCとPSRの内容をSPの示すスタックにセーブします。これは割り込みによって変更されるこの2つのレジスタの内容を保護するためです。PCとPSRのデータは図2.4.3.1のようにスタックにセーブされます。
処理ルーチンからのリターンにはPCとPSRの内容を復帰するreti命令を使用します。reti命令ではPC、PSRのセーブ内容が読み出され、SPの内容が図2.4.3.2のように変更されます。

割り込み発生時の動作

(1) SP = SP - 4
(2) PC + 2 → [SP]
(3) PSR → [SP + 3]

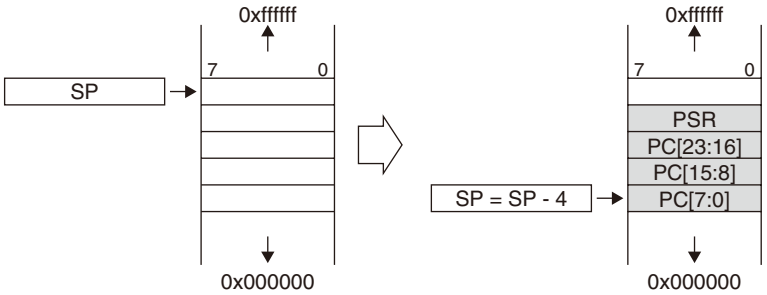


図2.4.3.1 SPとスタック(3)

reti命令実行の動作

(1) [SP] → PC
(2) [SP + 3] → PSR
(3) SP = SP + 4

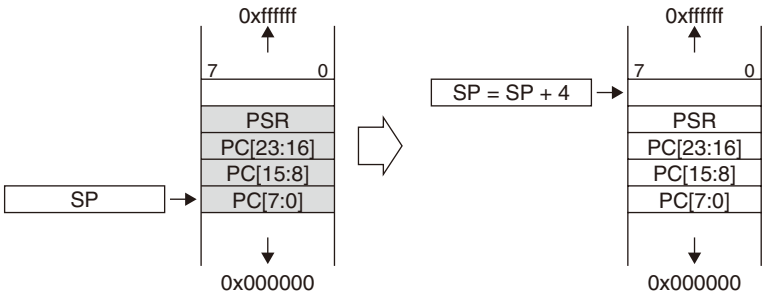


図2.4.3.2 SPとスタック(4)

2.4.4 ロード命令を使用したレジスタ値の退避と復帰

S1C17コアでは、レジスタ値のスタックへの退避と復帰を行うため、プッシュ /ポップ命令に代わるロード命令が用意されています。

スタックへのレジスタ退避

例: `ld.a  -[%sp],%r0`

(1) SP = SP - 4

(2) R0 → [SP]

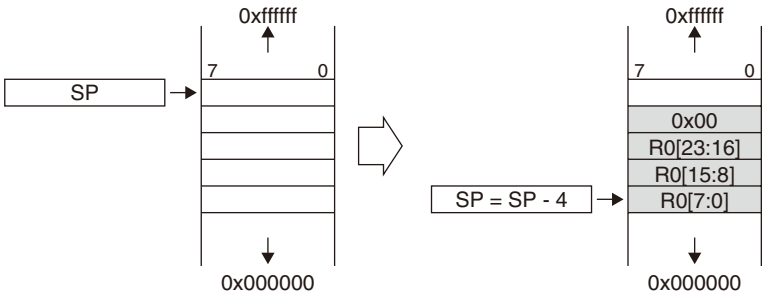


図2.4.4.1 SPとスタック(5)

スタックからのレジスタ復帰

例: `ld.a  %r0,[%sp]+`

(1) [SP] → R0

(2) SP = SP + 4

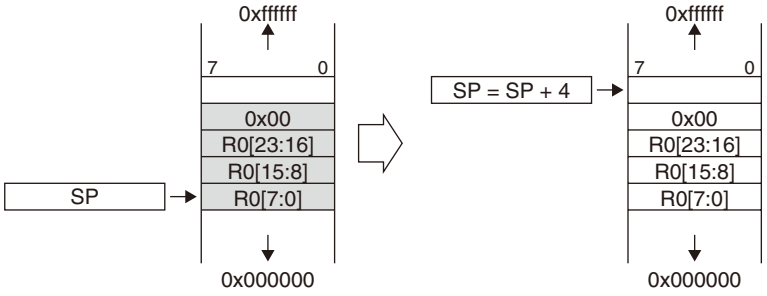


図2.4.4.2 SPとスタック(6)

この他にもスタック操作が可能なロード命令が用意されています。それらの命令については、“7 命令の詳細説明”を参照してください。

2.5 レジスタの表記とレジスタ番号

ここでは、S1C17コア命令セットのレジスタ表記とレジスタ番号について説明します。

2.5.1 汎用レジスタ

命令コード中では汎用レジスタの指定に3ビットのフィールドを使用しており、このフィールドにレジスタ番号が入ります。なお、ニーモニックのレジスタ指定においては、レジスタ名の前に“%”を記述します。

%*rs* *rs*は演算や転送のソースデータを保持している汎用レジスタを示すメタシンボルです。実際には%r0、%r1、...%r7のように記述します。

%*rd* *rd*はデスティネーション(演算結果が格納される、あるいはデータがロードされる)となる汎用レジスタを示すメタシンボルです。実際には%r0、%r1、...%r7のように記述します。

%*rb* *rb*はアクセスするメモリのベースアドレスを保持している汎用レジスタを示すメタシンボルです。この場合の汎用レジスタはインデックスレジスタとして機能します。
実際の表記は、レジスタ間接アドレッシングを示す[]で囲み、[%r0]、[%r1]、...[%r7]のように記述します。
レジスタ間接アドレッシングでは、連続したメモリアドレスをアクセスするためのポストインクリメント/デクリメント機能、プリデクリメント機能を使用することができます。

ポストインクリメント
例: ld %*rd*,[%*rb*]+ ; (1)ld %*rd*,[%*rb*] (2)%*rb* = %*rb* + 2
メモリアクセス後にベースアドレスがアクセスしたサイズに従ってインクリメントされます。

ポストデクリメント
例: ld.a %*rd*,[%*rb*]- ; (1)ld.a %*rd*,[%*rb*] (2)%*rb* = %*rb* - 4
メモリアクセス後にベースアドレスがアクセスしたサイズに従ってデクリメントされます。

プリデクリメント
例: ld.b -[%*rb*],%*rs* ; (1)%*rb* = %*rb* - 1 (2) ld.b [%*rb*],%*rs*
メモリアクセス前にベースアドレスがアクセスするサイズに従ってデクリメントされます。

アドレスのインクリメント/デクリメント値はext命令を使用して任意の値に設定することも可能です。

*rb*はコール命令やジャンプ命令の分岐アドレスを格納しているレジスタを示すシンボルとしても使用します。この場合は[]が不要で、%r0、%r1、...%r7のように記述します。

命令コード中のレジスタを指定するビットフィールドにはレジスタ番号に対応するコードが入ります。レジスタとレジスタ番号の対応は以下のとおりです。

表2.5.1.1 汎用レジスタ

汎用レジスタ	レジスタ番号	レジスタ表記
R0	0	%r0
R1	1	%r1
R2	2	%r2
R3	3	%r3
R4	4	%r4
R5	5	%r5
R6	6	%r6
R7	7	%r7

2.5.2 特殊レジスタ

命令で直接指定可能な特殊レジスタはSP(スタックポインタ)とPC(プログラムカウンタ)のみです。%sp、[%sp]、-[%sp]、[%sp]+、[%sp]-、[%sp+*imm7*]、%pcのように記述します。

3 データ形式

3.1 レジスタ↔レジスタ間オペレーションで扱うデータ形式

S1C17コアのレジスタオペレーションでは、8ビット長、16ビット長、24ビット長のデータを扱うことができます。
本書では、データのサイズを次のように表します。

8ビット　**バイト**, B, b
16ビット　**ワード**, W, w
24ビット　**アドレスデータ**, A, a

データサイズは、汎用レジスタ間のデータ転送(ロード命令)においてのみ選択可能です。
汎用レジスタへの8ビットのデータ転送では、レジスタにロードする際に16ビットへの符号拡張またはゼロ拡張が行われます。符号拡張またはゼロ拡張のどちらが行われるかについては、使用するロード命令によって決まります。
汎用レジスタからの16ビットデータ転送または8ビットデータ転送では、転送元レジスタの下位16ビットまたは下位8ビットが転送データとなります。
データ転送のサイズと種類は以下に示すとおりです。

3.1.1 符号なし8ビット転送(レジスタ→レジスタ)

例: `ld.ub  %rd,%rs`

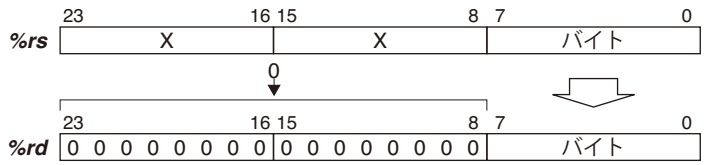


図3.1.1.1 符号なし8ビット転送(レジスタ→レジスタ)

転送先レジスタのビット23～8は0x0000に設定されます。

3.1.2 符号付き8ビット転送(レジスタ→レジスタ)

例: `ld.b  %rd,%rs`

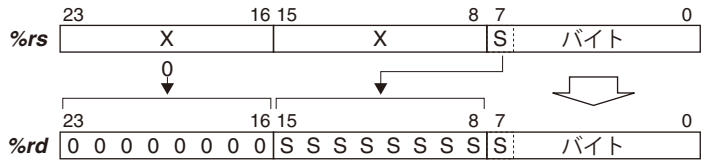


図3.1.2.1 符号付き8ビット転送(レジスタ→レジスタ)

転送先レジスタのビット15～8は符号拡張され、ビット23～16は0x00に設定されます。

3.1.3 16ビット転送(レジスタ→レジスタ)

例: `ld  %rd,%rs`

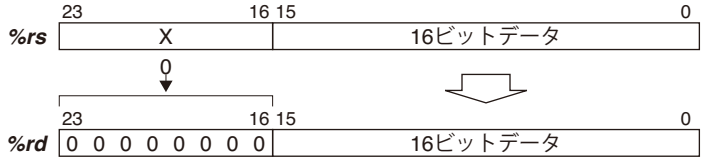


図3.1.3.1 16ビット転送(レジスタ→レジスタ)

転送先レジスタのビット23～16は0x00に設定されます。

3.1.4 24ビット転送(レジスタ→レジスタ)

例: `ld.a  %rd,%rs`

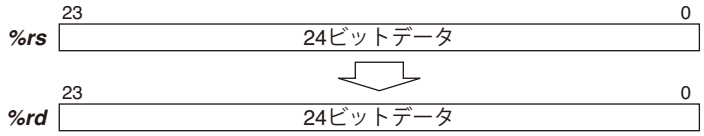


図3.1.4.1 24ビット転送(レジスタ→レジスタ)

3.2 レジスタ↔メモリ間オペレーションで扱うデータ形式

S1C17コアのメモリオペレーションでは、8ビット長、16ビット長、32ビット長のデータを扱うことができます。
本書では、データのサイズを次のように表します。

8ビット　**バイト**, B, b
16ビット　**ワード**, W, w
32ビット　**アドレスデータ**, A, a

データサイズは、メモリと汎用レジスタ間においてのみ選択可能です。
汎用レジスタへの8ビットのデータ転送では、レジスタにロードする際に16ビットへの符号拡張またはゼロ拡張が行われます。符号拡張またはゼロ拡張のどちらが行われるかについては、使用するロード命令によって決まります。
汎用レジスタからの16ビットデータ転送または8ビットデータ転送では、転送元レジスタの下位16ビットまたは下位8ビットが転送データとなります。
メモリはバイト、16ビット、32ビット単位にリトルエンディアン形式でアクセスされます。
なお、16ビット単位および32ビット単位のアクセスは、指定するベースアドレスがそれぞれ16ビット境界(アドレスの最下位ビットが0)、32ビット境界(アドレスの下位2ビットが00)であることが必要で、この条件を満たしていないアクセスに対してはアドレス不整割り込みが発行されます。

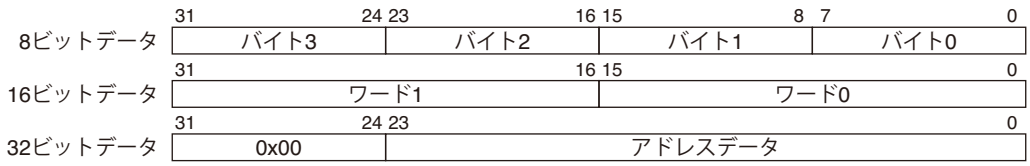


図3.2.1 データ形式(リトルエンディアン)

※ 32ビットアクセス時の上位8ビットデータについて
データは上位8ビットを0として書き込まれます。メモリからの読み出し時は上位8ビットが無視されます。割り込み処理のスタック操作時はPSRの値を上位8ビットとして書き込み/読み出しを行います。

データ転送のサイズと種類は以下に示すとおりです。

3.2.1 符号なし8ビット転送(メモリ→レジスタ)

例: `ld.ub  %rd,[%rb]`

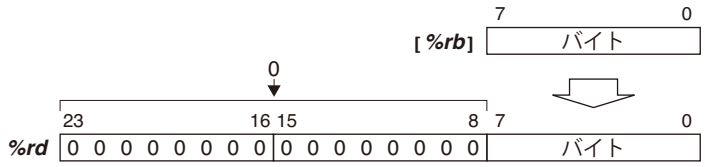


図3.2.1.1 符号なし8ビット転送(メモリ→レジスタ)

転送先レジスタのビット23～8は0x0000に設定されます。

3.2.2 符号付き8ビット転送(メモリ→レジスタ)

例: `ld.b  %rd,[%rb]`

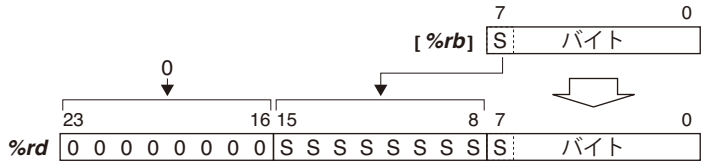


図3.2.2.1 符号付き8ビット転送(メモリ→レジスタ)

転送先レジスタのビット15～8は符号拡張され、ビット23～16は0x00に設定されます。

3.2.3 8ビット転送(レジスタ→メモリ)

例: `ld.b  [%rb],%rs`

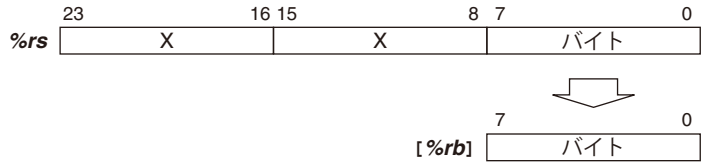


図3.2.3.1 8ビット転送(レジスタ→メモリ)

3.2.4 16ビット転送(メモリ→レジスタ)

例: `ld  %rd,[%rb]`

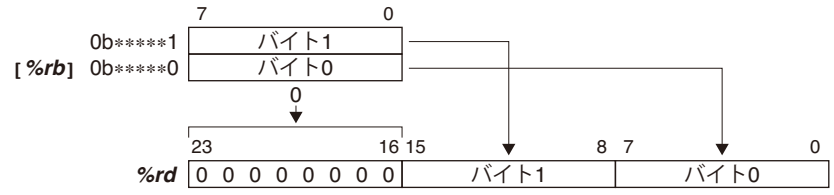


図3.2.4.1 16ビット転送(メモリ→レジスタ)

転送先レジスタのビット23～16は0x00に設定されます。

3.2.5 16ビット転送(レジスタ→メモリ)

例: `ld [%rb],%rs`

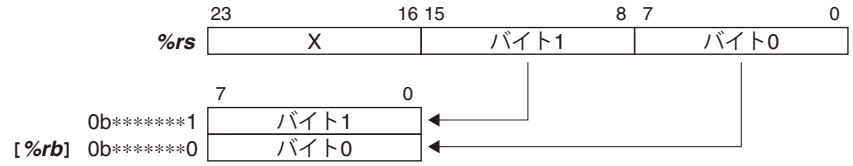


図3.2.5.1 16ビット転送(レジスタ→メモリ)

3.2.6 32ビット転送(メモリ→レジスタ)

例: `ld.a %rd,[%rb]`

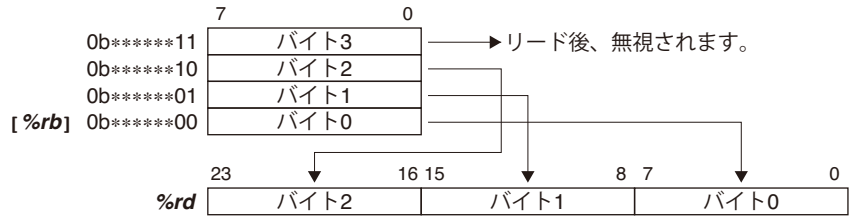


図3.2.6.1 32ビット転送(メモリ→レジスタ)

3.2.7 32ビット転送(レジスタ→メモリ)

例: `ld.a [%rb],%rs`

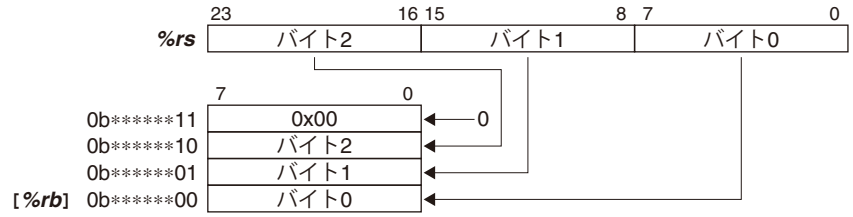


図3.2.7.1 32ビット転送(レジスタ→メモリ)

4 アドレスマップ

4.1 アドレス空間

S1C17コアは最大16Mバイト(24ビット)のアドレス空間をリニアに使用することができます。0xfffc00～0xffffffはコア予約のI/Oエリアです。これ以外にユーザRAM領域内の64バイトをデバッグ用に使用します。図4.1.1にS1C17コアのアドレス空間を示します。

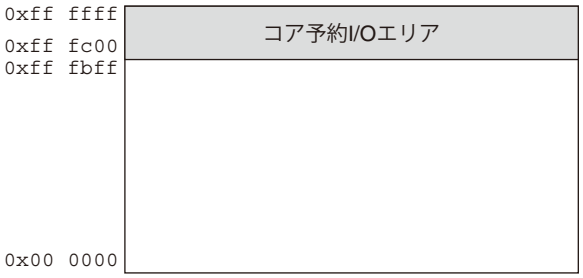


図4.1.1 S1C17コアのアドレス空間

ブートアドレスとデバッグRAMアドレスはS1C17シリーズの機種により異なります。各機種のテクニカルマニュアルを参照してください。

4.2 プロセッサ情報

コア予約I/Oエリアには以下に説明するプロセッサ情報が書き込まれています。

4.2.1 ベクタテーブルベースレジスタ（TTBR, 0xffff80）

Register name	Address	Bit	Name	Function	Setting	Init.	R/W	Remarks
Vector table base register	FFFF80 (L)	D31–24	–	Unused (fixed at 0)	0x0	0x0	R	
		D23	TTBR23	Vector table base address	0x0–0xFFFB00	*	R	Initial value is set by the TTBR pins of the C17 macro.
		|	|	TTBR[7:0] is fixed at 0x0.	(256 byte units)				
		D0	TTBR0					

ベクタテーブルベースアドレスが格納されるリードオンリレジスタです。
ベクタテーブルは各割り込み処理ルーチンへのベクタ（処理ルーチンの先頭アドレス）を記述しておくテーブルで、割り込み発生時にS1C17コアが参照して対応する処理ルーチンを実行します。リセット時に実行を開始するブートアドレスは、このベクタテーブルの先頭に書き込んでおきます。

このレジスタに格納されるアドレス値については、各機種のテクニカルマニュアルを参照してください。

4.2.2 プロセッサIDレジスタ（IDIR, 0xffff84）

Register name	Address	Bit	Name	Function	Setting	Init.	R/W	Remarks
Processor ID register	FFFF84 (B)	D7	IDIR7	Processor ID	0x10	0x10	R	
		|	|	0x10: S1C17 Core				
		D0	IDIR0					

プロセッサの機種を示すIDコードが格納されるリードオンリレジスタです。S1C17コアのIDコードは0x10です。

4.2.3 デバッグRAMベースレジスタ（DBRAM, 0xffff90）

Register name	Address	Bit	Name	Function	Setting	Init.	R/W	Remarks
Debug RAM base register	FFFF90 (L)	D31–24	–	Unused (fixed at 0)	0x0	0x0	R	
		D23	DBRAM23	Debug RAM base address	0x0–0xFFFDC0	*	R	Initial value is set in the C17 RTL-define DBRAM_BASE.
		|	|	DBRAM[5:0] is fixed at 0x0.	(64 byte units)				
		D0	DBRAM0					

デバッグ用ワークエリア（64バイト）の先頭アドレスが格納されるリードオンリレジスタです。

このレジスタに格納されるアドレス値については、各機種のテクニカルマニュアルを参照してください。

※ このほかに、コア予約I/Oエリアにはデバッグ用のレジスタが配置されています。デバッグ用レジスタについては、“6.5 デバッグ回路”を参照してください。

5 命令セット

S1C17コアの命令コードはすべて16ビットの固定長で、パイプライン処理を行うことによって主要な命令を1サイクルで実行します。各命令の詳細については“7 命令の詳細説明”を参照してください。

5.1 命令一覧

表5.1.1 S1C17命令一覧

種　類	ニーモニック		機　能
データ転送	ld.b	%*rd*,%*rs*	汎用レジスタ(バイト) → 汎用レジスタ(符号拡張)
		%*rd*,[%*rb*] %*rd*,[%*rb*]+ %*rd*,[%*rb*]- %*rd*,-[%*rb*]	メモリ(バイト) → 汎用レジスタ(符号拡張) メモリアドレスのポストインクリメント、ポストデクリメント、プリデクリメント機能を使用可能
		%*rd*,[%sp+*imm7*]	スタック(バイト) → 汎用レジスタ(符号拡張)
		%*rd*,[*imm7*]	メモリ(バイト) → 汎用レジスタ(符号拡張)
		[%*rb*],%*rs* [%*rb*]+,%*rs* [%*rb*]-,%*rs* -[%*rb*],%*rs*	汎用レジスタ(バイト) → メモリ メモリアドレスのポストインクリメント、ポストデクリメント、プリデクリメント機能を使用可能
		[%sp+*imm7*],%*rs*	汎用レジスタ(バイト) → スタック
		[*imm7*],%*rs*	汎用レジスタ(バイト) → メモリ
	ld.ub	%*rd*,%*rs*	汎用レジスタ(バイト) → 汎用レジスタ(ゼロ拡張)
		%*rd*,[%*rb*] %*rd*,[%*rb*]+ %*rd*,[%*rb*]- %*rd*,-[%*rb*]	メモリ(バイト) → 汎用レジスタ(ゼロ拡張) メモリアドレスのポストインクリメント、ポストデクリメント、プリデクリメント機能を使用可能
		%*rd*,[%sp+*imm7*]	スタック(バイト) → 汎用レジスタ(ゼロ拡張)
		%*rd*,[*imm7*]	メモリ(バイト) → 汎用レジスタ(ゼロ拡張)
	ld	%*rd*,%*rs*	汎用レジスタ(16ビット) → 汎用レジスタ
		%*rd*,*sign7*	即値 → 汎用レジスタ(符号拡張)
		%*rd*,[%*rb*] %*rd*,[%*rb*]+ %*rd*,[%*rb*]- %*rd*,-[%*rb*]	メモリ(16ビット) → 汎用レジスタ メモリアドレスのポストインクリメント、ポストデクリメント、プリデクリメント機能を使用可能
		%*rd*,[%sp+*imm7*]	スタック(16ビット) → 汎用レジスタ
		%*rd*,[*imm7*]	メモリ(16ビット) → 汎用レジスタ
		[%*rb*],%*rs* [%*rb*]+,%*rs* [%*rb*]-,%*rs* -[%*rb*],%*rs*	汎用レジスタ(16ビット) → メモリ メモリアドレスのポストインクリメント、ポストデクリメント、プリデクリメント機能を使用可能
		[%sp+*imm7*],%*rs*	汎用レジスタ(16ビット) → スタック
		[*imm7*],%*rs*	汎用レジスタ(16ビット) → メモリ
	ld.a	%*rd*,%*rs*	汎用レジスタ(24ビット) → 汎用レジスタ
		%*rd*,*imm7*	即値 → 汎用レジスタ(ゼロ拡張)
		%*rd*,[%*rb*] %*rd*,[%*rb*]+ %*rd*,[%*rb*]- %*rd*,-[%*rb*]	メモリ(32ビット) → 汎用レジスタ * メモリアドレスのポストインクリメント、ポストデクリメント、プリデクリメント機能を使用可能
		%*rd*,[%sp+*imm7*]	スタック(32ビット) → 汎用レジスタ *
		%*rd*,[*imm7*]	メモリ(32ビット) → 汎用レジスタ *
		[%*rb*],%*rs* [%*rb*]+,%*rs* [%*rb*]-,%*rs* -[%*rb*],%*rs*	汎用レジスタ(32ビット、ゼロ拡張) → メモリ * メモリアドレスのポストインクリメント、ポストデクリメント、プリデクリメント機能を使用可能
		[%sp+*imm7*],%*rs*	汎用レジスタ(32ビット、ゼロ拡張) → スタック *
		[*imm7*],%*rs*	汎用レジスタ(32ビット、ゼロ拡張) → メモリ *
		%*rd*,%sp	SP → 汎用レジスタ
		%*rd*,%pc	PC → 汎用レジスタ
		%*rd*,[%sp] %*rd*,[%sp]+ %*rd*,[%sp]- %*rd*,-[%sp]	スタック(32ビット) → 汎用レジスタ * スタックポインタのポストインクリメント、ポストデクリメント、プリデクリメント機能を使用可能

種　類	ニーモニック		機　能
データ転送	ld.a	[%sp],%*rs* [%sp]+,%*rs* [%sp]-,%*rs* -[%sp],%*rs*	汎用レジスタ(32ビット、ゼロ拡張) → スタック * スタックポインタのポストインクリメント、ポストデクリメント、プリデクリメント機能を使用可能
		%sp,%*rs*	汎用レジスタ(24ビット) → SP
		%sp,*imm7*	即値 → SP
整数算術演算	add add/c add/nc	%*rd*,%*rs*	汎用レジスタ間の16ビット加算 条件実行に対応(/c: C = 1の場合に実行, /nc: C = 0の場合に実行)
	add	%*rd*,*imm7*	汎用レジスタと即値の16ビット加算
	add.a add.a/c add.a/nc	%*rd*,%*rs*	汎用レジスタ間の24ビット加算 条件実行に対応(/c: C = 1の場合に実行, /nc: C = 0の場合に実行)
	add.a	%sp,%*rs*	SPと汎用レジスタの24ビット加算
		%*rd*,*imm7*	汎用レジスタと即値の24ビット加算
		%sp,*imm7*	SPと即値の24ビット加算
	adc adc/c adc/nc	%*rd*,%*rs*	汎用レジスタ間のキャリー付き16ビット加算 条件実行に対応(/c: C = 1の場合に実行, /nc: C = 0の場合に実行)
	adc	%*rd*,*imm7*	汎用レジスタと即値のキャリー付き16ビット加算
	sub sub/c sub/nc	%*rd*,%*rs*	汎用レジスタ間の16ビット減算 条件実行に対応(/c: C = 1の場合に実行, /nc: C = 0の場合に実行)
	sub	%*rd*,*imm7*	汎用レジスタと即値の16ビット減算
	sub.a sub.a/c sub.a/nc	%*rd*,%*rs*	汎用レジスタ間の24ビット減算 条件実行に対応(/c: C = 1の場合に実行, /nc: C = 0の場合に実行)
	sub.a	%sp,%*rs*	SPと汎用レジスタの24ビット減算
		%*rd*,*imm7*	汎用レジスタと即値の24ビット減算
		%sp,*imm7*	SPと即値の24ビット減算
	sbc sbc/c sbc/nc	%*rd*,%*rs*	汎用レジスタ間のキャリー付き16ビット減算 条件実行に対応(/c: C = 1の場合に実行, /nc: C = 0の場合に実行)
	sbc	%*rd*,*imm7*	汎用レジスタと即値のキャリー付き16ビット減算
	cmp cmp/c cmp/nc	%*rd*,%*rs*	汎用レジスタ間の16ビット比較 条件実行に対応(/c: C = 1の場合に実行, /nc: C = 0の場合に実行)
	cmp	%*rd*,*sign7*	汎用レジスタと即値の16ビット比較
	cmp.a cmp.a/c cmp.a/nc	%*rd*,%*rs*	汎用レジスタ間の24ビット比較 条件実行に対応(/c: C = 1の場合に実行, /nc: C = 0の場合に実行)
	cmp.a	%*rd*,*imm7*	汎用レジスタと即値の24ビット比較
	cmc cmc/c cmc/nc	%*rd*,%*rs*	汎用レジスタ間のキャリー付き16ビット比較 条件実行に対応(/c: C = 1の場合に実行, /nc: C = 0の場合に実行)
	cmc	%*rd*,*sign7*	汎用レジスタと即値のキャリー付き16ビット比較
論理演算	and and/c and/nc	%*rd*,%*rs*	汎用レジスタ間の論理積 条件実行に対応(/c: C = 1の場合に実行, /nc: C = 0の場合に実行)
	and	%*rd*,*sign7*	汎用レジスタと即値の論理積
	or or/c or/nc	%*rd*,%*rs*	汎用レジスタ間の論理和 条件実行に対応(/c: C = 1の場合に実行, /nc: C = 0の場合に実行)
	or	%*rd*,*sign7*	汎用レジスタと即値の論理和
	xor xor/c xor/nc	%*rd*,%*rs*	汎用レジスタ間の排他的論理和 条件実行に対応(/c: C = 1の場合に実行, /nc: C = 0の場合に実行)
	xor	%*rd*,*sign7*	汎用レジスタと即値の排他的論理和
	not not/c not/nc	%*rd*,%*rs*	汎用レジスタ間の論理否定(1の補数) 条件実行に対応(/c: C = 1の場合に実行, /nc: C = 0の場合に実行)
	not	%*rd*,*sign7*	汎用レジスタと即値の論理否定(1の補数)

種　類	ニーモニック		機　能
シフト&スワップ	sr	%rd,%rs	右論理シフト(レジスタによるシフトビット数指定)
		%rd,imm7	右論理シフト(即値によるシフトビット数指定)
	sa	%rd,%rs	右算術シフト(レジスタによるシフトビット数指定)
		%rd,imm7	右算術シフト(即値によるシフトビット数指定)
	sl	%rd,%rs	左論理シフト(レジスタによるシフトビット数指定)
		%rd,imm7	左論理シフト(即値によるシフトビット数指定)
	swap	%rd,%rs	16ビット境界でバイト単位のスワップ
即値拡張	ext	imm13	直後の命令のオペランドを拡張
コンバージョン	cv.ab	%rd,%rs	符号付き8ビットデータを24ビットに変換
	cv.as	%rd,%rs	符号付き16ビットデータを24ビットに変換
	cv.al	%rd,%rs	32ビットデータを24ビットに変換
	cv.la	%rd,%rs	24ビットデータを32ビットに変換
	cv.ls	%rd,%rs	16ビットデータを32ビットに変換
分岐	jpr	sign10	PC相対ジャンプ
	jpr.d	%rb	ディレイド分岐可
	jpa	imm7	絶対ジャンプ
	ipa.d	%rb	ディレイド分岐可
	jrgt	sign7	PC相対条件ジャンプ　分岐条件: !Z & !(N ^ V)
	jrgt.d		ディレイド分岐可
	jrge	sign7	PC相対条件ジャンプ　分岐条件: !(N ^ V)
	jrge.d		ディレイド分岐可
	jrlt	sign7	PC相対条件ジャンプ　分岐条件: N ^ V
	jrlt.d		ディレイド分岐可
	jrle	sign7	PC相対条件ジャンプ　分岐条件: Z I N ^ V
	jrle.d		ディレイド分岐可
	jrugt	sign7	PC相対条件ジャンプ　分岐条件: !Z & !C
	jrugt.d		ディレイド分岐可
	jruge	sign7	PC相対条件ジャンプ　分岐条件: !C
	jruge.d		ディレイド分岐可
	jrult	sign7	PC相対条件ジャンプ　分岐条件: C
	jrult.d		ディレイド分岐可
	jrule	sign7	PC相対条件ジャンプ　分岐条件: Z I C
	jrule.d		ディレイド分岐可
	jreq	sign7	PC相対条件ジャンプ　分岐条件: Z
	jreq.d		ディレイド分岐可
	jrne	sign7	PC相対条件ジャンプ　分岐条件: !Z
	jrne.d		ディレイド分岐可
	call	sign10	PC相対サブルーチンコール
	call.d	%rb	ディレイド分岐可
	calla	imm7	絶対サブルーチンコール
	calla.d	%rb	ディレイド分岐可
	ret		サブルーチンからのリターン
	ret.d		ディレイド分岐可
	int	imm5	ソフトウェア割り込み
	intl	imm5,imm3	割り込みレベル指定付きソフトウェア割り込み
	reti		割り込み処理からのリターン
	reti.d		ディレイド分岐可
	brk		デバッグ割り込み
	retd		デバッグ処理からのリターン
システム制御	nop		ノーオペレーション
	halt		HALT
	slp		SLEEP
	ei		割り込み許可
	di		割り込み禁止
コプロセッサ制御	ld.cw	%rd,%rs	コプロセッサへのデータ転送
		%rd,imm7	
	ld.ca	%rd,%rs	コプロセッサへのデータ転送、結果とフラグ状態の取得
		%rd,imm7	
	ld.cf	%rd,%rs	コプロセッサへのデータ転送、フラグ状態の取得
		%rd,imm7	

＊ ld.a命令は32ビットのメモリアクセスを行います。レジスタからメモリへのデータ転送では上位8ビットを0とした32ビットデータがメモリに書き込まれます。メモリからの読み出し時は、読み出しデータの上位8ビットが無視されます。

表中の記号の意味は次のとおりです。

表5.1.2 記号の意味

記　号	説　明
%*rs*	汎用ソースレジスタ
%*rd*	汎用デスティネーションレジスタ
[%*rb*]	汎用レジスタで間接指定されるメモリ
[%*rb*]+	汎用レジスタで間接指定されるメモリ(アドレスポストインクリメント付き)
[%*rb*]-	汎用レジスタで間接指定されるメモリ(アドレスポストデクリメント付き)
-[%*rb*]	汎用レジスタで間接指定されるメモリ(アドレスプリデクリメント付き)
%sp	スタックポインタ
[%sp],[%sp+*imm7*]	スタック
[%sp]+	スタック(アドレスポストインクリメント付き)
[%sp]-	スタック(アドレスポストデクリメント付き)
-[%sp]	スタック(アドレスプリデクリメント付き)
imm3,*imm5*,*imm7*,*imm13*	符号なし即値(数値はビット長)
sign7,*sign10*	符号付き即値(数値はビット長)

5.2 アドレッシングモード(ext拡張なし)

S1C17コアの命令セットは以下に示す7種類のアドレッシングモードを持ちます。プロセッサは各命令のオペランドによってアドレッシングモードを決定し、データをアクセスします。
(1)即値アドレッシング
(2)レジスタ直接アドレッシング
(3)レジスタ間接アドレッシング
(4)ポストインクリメント/ポストデクリメント/プリデクリメント付きレジスタ間接アドレッシング
(5)ディスプレースメント付きレジスタ間接アドレッシング
(6)符号付きPC相対アドレッシング
(7)PC絶対アドレッシング

5.2.1 即値アドレッシング

命令コード中に含まれる*immX*(符号なし即値)、*signX*(符号付き即値)で示される即値をソースデータとして使用します。各命令で指定可能な即値サイズは、シンボルの数字(例: *imm7* = 符号なし7ビット、*sign7* = 符号付き7ビット)で示されます。*sign7*などの符号付き即値は最上位ビットが符号となり、このビットが命令実行時に16ビットまたは24ビットまで拡張されます。

```
例: ld    %r0,0x70      ; 16ビットデータのロード
```

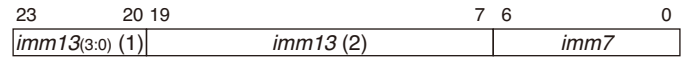
実行前 r0 = 0x*XXXXXX*
実行後 r0 = 0x00fff0

*sign7*は+63～-64(0b0111111～0b1000000)まで表すことができます。

また、シフト系命令を除き、ext命令によって即値を最大24ビットまで拡張することができます。

```
例: ext     imm13       (1)
    ext     imm13       (2)
    ld.a    %r0,imm7    ; 24ビットデータのロード
```

実行後のr0



5.2.2 レジスタ直接アドレッシング

指定のレジスタの内容をそのままソースデータとして使用します。また、結果をレジスタにロードする命令のデスティネーションとして指定した場合は、結果がそのレジスタにロードされます。以下のシンボルをオペランドとして持つ命令がこのアドレッシングモードで実行されます。

%rs *rs*は演算や転送のソースデータを保持している汎用レジスタを示すメタシンボルです。実際には%r0～%r7と記述します。

%rd *rd*はデスティネーションとなる汎用レジスタを示すメタシンボルです。実際には%r0～%r7と記述します。命令によってはソースデータにもなります。

特殊レジスタ名は次のように記述します。

```
スタックポインタ      %sp
プログラムカウンタ    %pc
```

レジスタ名はシンボル名やラベル名等と区別するため、必ず"%"を前置きします。

5.2.3 レジスタ間接アドレッシング

アドレスを保持している汎用レジスタまたはスタックポインタを指定して、間接的にメモリをアクセスするモードです。[%*rb*]または[%sp]をオペランドとして持つロード命令にのみ適用されます。実際の汎用レジスタ名を[]で囲み、[%r0]、[%r1]、...[%r7]、[%sp]のように記述します。
プロセッサは指定レジスタの内容をベースアドレスとして、ロード命令の種類によって決まるデータ形式でデータ転送を行います。
例: メモリ → レジスタ

```
ld.b   %r0,[%r1]          ; 8ビットデータのロード
ld     %r0,[%r1]          ; 16ビットデータのロード
ld.a   %r0,[%r1]          ; 24ビットデータのロード
```

レジスタ → メモリ

```
ld.b   [%r1],%r0          ; 8ビットデータのストア
ld     [%r1],%r0          ; 16ビットデータのストア
ld.a   [%r1],%r0          ; 24ビットデータのストア
```

この例では、r1の示すアドレスがデータを転送する対象のメモリアドレスになります。

16ビット転送、24ビット転送の場合、レジスタに設定するベースアドレスはそれぞれ16ビット境界(最下位ビット = 0)、32ビット境界(下位2ビット = 0)を指している必要があり、それ以外ではアドレス不整割り込みが発生します。

5.2.4 ポストインクリメント/デクリメント, プリデクリメント付きレジスタ間接アドレッシング

レジスタ間接アドレッシングと同様に、汎用レジスタまたはスタックポインタによってアクセスするメモリを間接的に指定します。
このアドレッシングモードでは、データ転送の前または後に、指定のレジスタが保持しているベースアドレスが転送データサイズ分インクリメント/デクリメントされます。これにより、最初に1回先頭アドレスを設定するだけで、メモリ上の連続したデータの読み出し/書き込みが行えます。

* インクリメント/デクリメントサイズ(extなし)
  バイト転送(ld.b, ld.ub): *rb* → *rb* + 1、*rb* → *rb* - 1
  16ビット転送(ld): *rb* → *rb* + 2、*rb* → *rb* - 2
  24ビット転送(ld.a): *rb* → *rb* + 4、*rb* → *rb* - 4

ポストインクリメント付きレジスタ間接アドレッシング

データ転送が終了すると、ベースアドレスをインクリメントします。
このアドレッシングモードは、レジスタ名を[]で囲み"+"を後置きして指定します。
実際には[%r0]+、[%r1]+、...[%r7]+、[%sp]+のように記述します。

ポストデクリメント付きレジスタ間接アドレッシング

データ転送が終了すると、ベースアドレスをデクリメントします。
このアドレッシングモードは、レジスタ名を[]で囲み"-"を後置きして指定します。
実際には[%r0]-、[%r1]-、...[%r7]-、[%sp]-のように記述します。

プリデクリメント付きレジスタ間接アドレッシング

データの転送前に、ベースアドレスをデクリメントします。
このアドレッシングモードは、レジスタ名を[]で囲み"-"を前置きして指定します。
実際には-[%r0]、-[%r1]、...-[%r7]、-[%sp]のように記述します。

5.2.5 ディスプレースメント付きレジスタ間接アドレッシング

レジスタの内容に指定の即値(ディスプレースメント)を加えたアドレスから始まるメモリをアクセスするモードです。ext命令を使用しない場合、このアドレッシングモードは[%sp+*imm7*]をオペランドとして持つロード命令にのみ適用されます。

例: ld.b %r0,[%sp+0x10]
現在のSPの内容に0x10を加算したアドレスのバイトデータをR0レジスタにロードします。

ext命令を使用すると、通常のレジスタ間接アドレッシング([%*rb*])がext命令で指定した即値をディスプレースメントとするアドレッシングモードに変わります(詳細は5.3節を参照)。

例: ext *imm13*
ld.b *%rd*,[*%rb*] アクセスするメモリアドレスは"*%rb*+*imm13*"となります。

5.2.6 符号付きPC相対アドレッシング

符号付き7/10ビット即値(*sign7*/*sign10*)または*%rb*をオペランドに持つjpr、jr*、call命令に適用されるアドレッシングモードです。それらの分岐命令を実行すると、現在のPCに*sign7*/*sign10*の2倍の値(16ビット境界)または*rb*レジスタの値を加算したアドレスに分岐します。

例: PC + 0 jrne 0x04 jrneの分岐条件が成立するとPC + 8番地に分岐します。
: : (PC + 0) + 0x04 * 2 → PC + 8
: :
PC + 8

5.2.7 PC絶対アドレッシング

符号なし7ビット即値(*imm7*)または*%rb*をオペランドに持つjpa、calla命令に適用されるアドレッシングモードです。それらの分岐命令を実行すると、*imm7*または*rb*レジスタの値を直接PCにロードして分岐します。また、ベクタ番号を指定して割り込み処理ルーチンを実行するint、intl命令もこのアドレッシングモードに含まれます。

例: int 0x03 ベクタ番号3(TTBR + 0xc)の割り込み処理を実行

5.3 ext付きアドレッシングモード

16ビット固定長の命令コードで指定できる即値は、命令によって異なりますが7ビットまたは10ビットのビットフィールドで指定します。ext命令はこの即値のサイズを拡張するために使用します。
ext命令は、データ転送命令、演算命令、分岐命令と組み合わせて使用し、即値の拡張を行いたい命令の直前に置きます。命令はext *imm13*の形式で記述します。1個のext命令で拡張可能な即値サイズは13ビットで、さらに即値拡張するために2個までext命令を続けて記述することができます。
ext命令が有効となるのは直後に記述された即値拡張が可能な命令に対してのみで、その他の命令に対しては無効です。3個以上のext命令が連続的に記述された場合は最後の2個が有効となり、それ以外は無視されます。
ext命令の直後の命令がextによる拡張機能に対応していない場合、そのext命令はnop命令として実行されます。

5.3.1 即値アドレッシングの拡張

●*imm7*の拡張

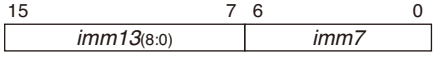
*imm7*を16ビット、20ビット、あるいは24ビット即値に拡張します。

16ビット即値への拡張

即値を16ビットに拡張するには、1個のext命令を対象命令の直前に置きます。

例:
```
ext   imm13
add   %rd,imm7          ; = add   %rd,imm16
```

拡張された即値

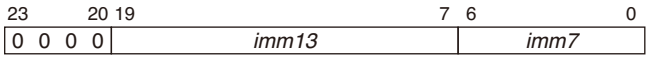


20ビット即値への拡張

即値を20ビットに拡張するには、1個のext命令を対象命令の直前に置きます。

例:
```
ext     imm13
add.a   %rd,imm7        ; = add.a   %rd,imm20
```

拡張された即値



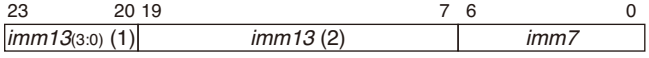
ビット23～20は0で埋まります(ゼロ拡張)。

24ビット即値への拡張

即値を24ビットに拡張するには、2個のext命令を対象命令の直前に置きます。

例:
```
ext   imm13       (1)
ext   imm13       (2)
ld    %rd,[imm7]        ; = ld   %rd,[imm24]
```

拡張された即値



●*sign7*の拡張

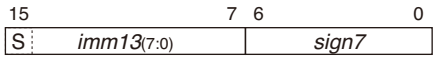
*sign7*を符号拡張された16ビット即値に拡張します。

16ビット即値への拡張

即値を16ビットに拡張するには、1個のext命令を対象命令の直前に置きます。

例: `ext  imm13`
`ld  %rd,sign7`

拡張された即値



ext命令の即値のビット8を符号として、16ビットの符号付きデータに拡張されます。*sign7*の最上位ビットは符号ではなく、7ビットデータの最上位データとして扱われます。

5.3.2 レジスタ直接アドレッシングの拡張

●レジスタ間演算命令を拡張

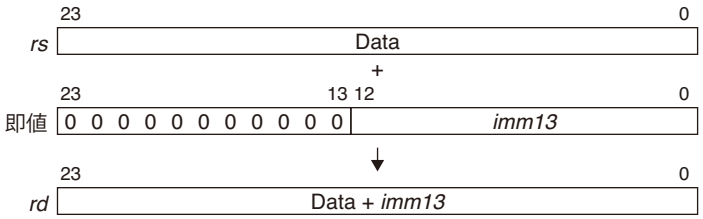
レジスタ間の演算命令をext命令で拡張します。データ転送命令と異なり、この場合は*rs*レジスタの内容とext命令で指定する即値を命令に従って演算し、その結果を*rd*レジスタに格納します。*rd*レジスタの内容は演算には影響を与えません。
加算の場合の拡張例を以下に示します。

rs + *imm13*への拡張(16ビットおよび24ビット演算命令)

rs + *imm13*に拡張するには、1個のext命令を対象命令の直前に置きます。

例: `ext  imm13`
`add  %rd,%rs`

拡張がない場合 → *rd* = *rd* + *rs*
1個のext命令で拡張した場合 → *rd* = *rs* + *imm13*

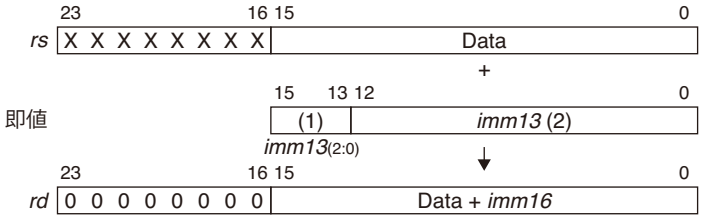


rs + *imm16*への拡張(16ビット演算命令)

rs + *imm16*に拡張するには、2個のext命令を対象命令の直前に置きます。

例: `ext  imm13`　　(1)
`ext  imm13`　　(2)
`add  %rd,%rs`

拡張がない場合 → *rd* = *rd* + *rs*
2個のext命令で拡張した場合 → *rd* = *rs* + *imm16*

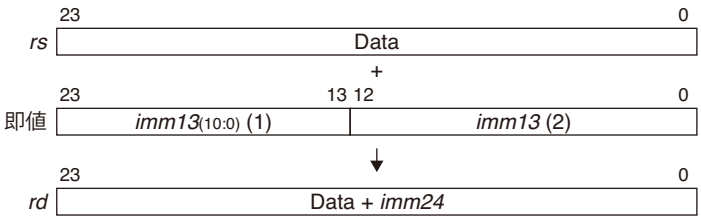


rs + *imm24*への拡張（24ビット演算命令）

rs + *imm24*に拡張するには、2個のext命令を対象命令の直前に置きます。

例: ext *imm13* （1）
ext *imm13* （2）
add *%rd*,*%rs*

拡張がない場合 → *rd* = *rd* + *rs*
2個のext命令で拡張した場合 → *rd* = *rs* + *imm24*



5.3.3 レジスタ間接アドレッシングの拡張

●[%rb]にディスプレースメントを付加

[%*rb*]で間接参照されるアドレスに、ext命令で指定される即値を加算したアドレスをアクセスします。

13ビットの即値を加算

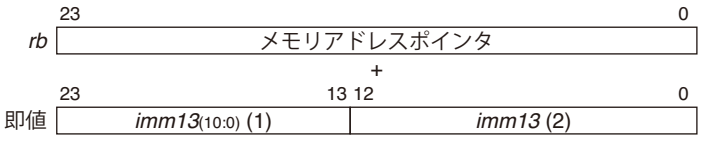
*rb*レジスタで指定されるアドレスに*imm13*で指定される13ビットの即値を加えたアドレスをアクセスします。アドレス演算の際、*imm13*は24ビットにゼロ拡張されます。

例: ext *imm13*
ld.b *%rd*,[*%rb*] ; = ld.b *%rd*,[*%rb*+*imm13*]

24ビットの即値を加算

*rb*レジスタで指定されるアドレスに*imm24*で指定される24ビットの即値を加えたアドレスをアクセスします。

例: ext *imm13* （1）
ext *imm13* （2）
ld.b *%rd*,[*%rb*] ; = ld.b *%rd*,[*%rb*+*imm24*]



5.3.4 ディスプレースメント付きレジスタ間接アドレッシングの拡張

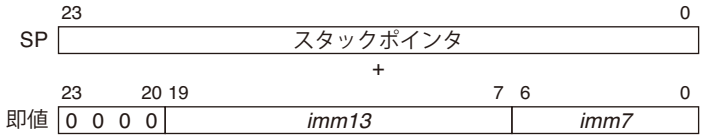
●[%sp+*imm7*]のディスプレースメントを拡張

ディスプレースメント付きレジスタ間接アドレッシング命令の即値(*imm7*)を拡張します。拡張されたデータはSPと加算され、転送先または転送元アドレスとして扱われます。

20ビット即値への拡張

即値を20ビットに拡張するには、1個のext命令を対象命令の直前に置きます。

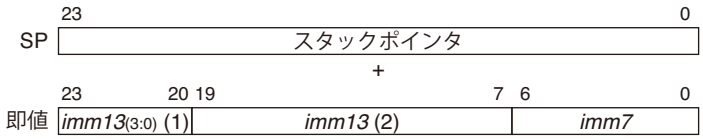
例:
```
ext   imm13
ld    %rd,[%sp+imm7]        ; = ld   %rd,[%sp+imm20]
```



24ビット即値への拡張

即値を24ビットに拡張するには、2個のext命令を対象命令の直前に配置します。

例:
```
ext   imm13       (1)
ext   imm13       (2)
ld    %rd,[%sp+imm7]        ; = ld   %rd,[%sp+imm24]
```



5.3.5 符号付きPC相対アドレッシングの拡張

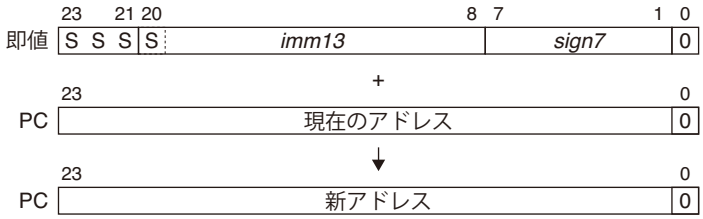
●PC相対分岐命令のディスプレースメントの拡張

PC相対分岐命令の即値*sign7*を、21ビットの符号付き即値または24ビットの符号付き即値に拡張します。*sign7*を2倍にして分岐先アドレスの相対値に変換後、PCと加算して分岐先アドレスを決定します。ext命令はこの分岐相対値を拡張します。

21ビット即値への拡張

*sign7*を21ビットの即値に拡張するには、1個のext命令を対象命令の直前に置きます。

例:
```
ext   imm13
jrgt  sign7                 ; = jrgt  sign21
```

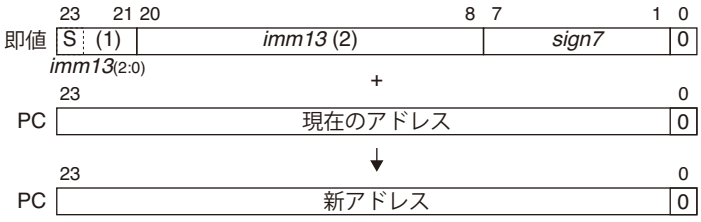


ext命令で拡張された即値の最上位ビット"S"が符号としてビット23～21に拡張され、21ビットの符号付きデータとなります。*sign7*の最上位ビットは符号ではなく、7ビットデータの最上位データとして扱われます。

24ビット即値への拡張

*sign7*を24ビットの即値に拡張するには、2個のext命令を対象命令の直前に置きます。

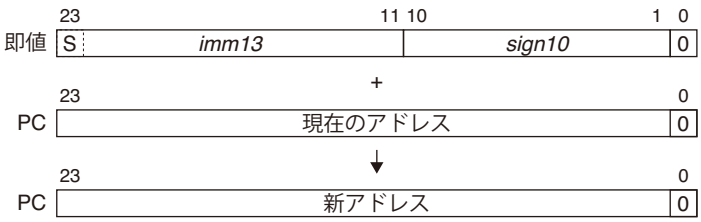
例: ext *imm13* (1)
ext *imm13* (2)
jrgt *sign7* ; = jrgt *sign24*



ext命令で拡張された即値の最上位ビット“S”が符号になります。最初のext命令のビット12～3は使用されません。

jprおよびcall命令のオペランド*sign10*も1個のext命令で24ビットの即値に拡張することができます。

例: ext *imm13*
call *sign10* ; = call *sign24*



5.3.6 PC絶対アドレッシングの拡張

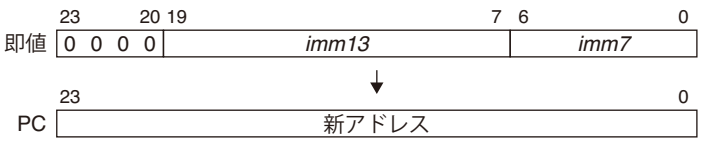
●分岐先アドレスの拡張

*imm7*を20ビットあるいは24ビット即値に拡張します。

20ビット即値への拡張

即値を20ビットに拡張するには、1個のext命令を対象命令の直前に置きます。

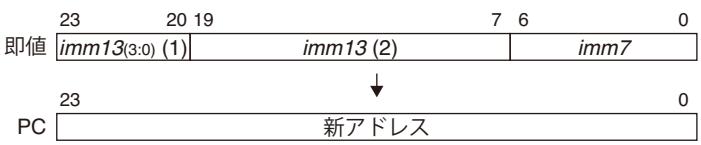
例: ext *imm13*
calla *imm7* ; = calla *imm20*



24ビット即値への拡張

即値を24ビットに拡張するには、2個のext命令を対象命令の直前に置きます。

例: ext *imm13* (1)
ext *imm13* (2)
jpa *imm7* ; = jpa *imm24*



5.4 データ転送命令

S1C17コアの転送命令は、レジスタ～レジスタ間、レジスタ～メモリ間のデータ転送をサポートしています。転送データサイズとデータ拡張形式が命令コードで指定可能です。ニーモニック表記上では次のように分類されます。

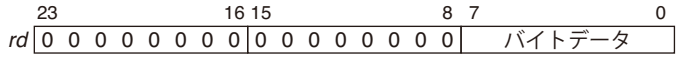
ld.b	符号付きバイトデータ転送
ld.ub	符号なしバイトデータ転送
ld	16ビットデータ転送
ld.a	24/32ビットデータ転送

レジスタへの符号付きバイト転送では、ソースデータが16ビットに符号拡張されます。符号なしバイト転送では、ソースデータが16ビットにゼロ拡張されます。
レジスタをソースとする転送では、レジスタ内下位側の指定サイズ分が転送データとなります。

転送先が汎用レジスタのとき、転送後のレジスタの内容は次のようになります。

●符号付きバイトデータ転送

●符号なしバイトデータ転送



●16ビットデータ転送

●24/32ビットデータ転送

メモリ上のデータ配置については、“3 データ形式”を参照してください。

5.5 論理演算命令

S1C17コアでは、4種類の論理演算命令が使用可能です。

and	論理積命令
or	論理和命令
xor	排他的論理和命令
not	否定命令

すべての論理演算は、指定の汎用レジスタ(R0～R7)に対して行われます。
ソースは汎用レジスタの16ビットデータか、即値データ(7 、13 、16ビット)の2種類です。
論理演算命令を実行するとPSRのVフラグ(ビット2)がクリアされます。

●条件実行

レジスタ間論理演算命令(*op %rd,%rs*)では、Cフラグの状態によって命令を実行するか否かを指定するスイッチが使用可能です。

無条件実行命令

op %rd,%rs (*op* = and, or, xor, not)

スイッチなしの命令は、Cフラグの状態にかかわらず常に実行されます。
例: and *%rd,%rs*

C条件実行命令

op/c *%rd,%rs* (*op* = and, or, xor, not)

/cスイッチ付きの命令は、Cフラグが1にセットされている場合にのみ実行されます。
例: or/c *%rd,%rs*

NC条件実行命令

op/nc *%rd,%rs* (*op* = and, or, xor, not)

/ncスイッチ付きの命令は、Cフラグが0にクリアされている場合にのみ実行されます。
例: xor/nc *%rd,%rs*

5.6 算術演算命令

S1C17コア命令セットでは、算術演算用に加減算、比較命令をサポートしています。

add	16ビット加算命令
add.a	24ビット加算命令
adc	16ビットキャリー付き加算命令
sub	16ビット減算命令
sub.a	24ビット減算命令
sbc	16ビットボロー付き減算命令
cmp	16ビット比較命令
cmp.a	24ビット比較命令
cmc	16ビットボロー付き比較命令

上記算術演算は、汎用レジスタ間(R0～R7)、汎用レジスタ～即値間で行われます。add.a命令、sub.a命令は、さらにSP～汎用レジスタ/即値間の演算にも対応しています。cmp命令を除き、演算単位(16ビットまたは24ビット)未満の即値は演算の際にゼロ拡張されます。
cmp命令は2つのオペランドを比較する命令で、比較結果によりPSR内のフラグのみを変更します。基本的には条件ジャンプ命令の条件設定に使用します。ソースに演算単位(16ビットまたは24ビット)未満の即値を指定した場合は、比較の際に符号拡張されます。

●条件実行

レジスタ間算術演算命令(*op %rd,%rs*)では、Cフラグの状態によって命令を実行するか否かを指定するスイッチが使用可能です。

無条件実行命令

op %rd,%rs (*op* = add, add.a, adc, sub, sub.a, sbc, cmp, cmp.a, cmc)

スイッチなしの命令は、Cフラグの状態にかかわらず常に実行されます。
例: add %rd,%rs

C条件実行命令

op/c %rd,%rs (*op* = add, add.a, adc, sub, sub.a, sbc, cmp, cmp.a, cmc)

/cスイッチ付きの命令は、Cフラグが1にセットされている場合にのみ実行されます。
例: sub/c %rd,%rs

NC条件実行命令

op/nc %rd,%rs (*op* = add, add.a, adc, sub, sub.a, sbc, cmp, cmp.a, cmc)

/ncスイッチ付きの命令は、Cフラグが0にクリアされている場合にのみ実行されます。
例: cmp/nc %rd,%rs

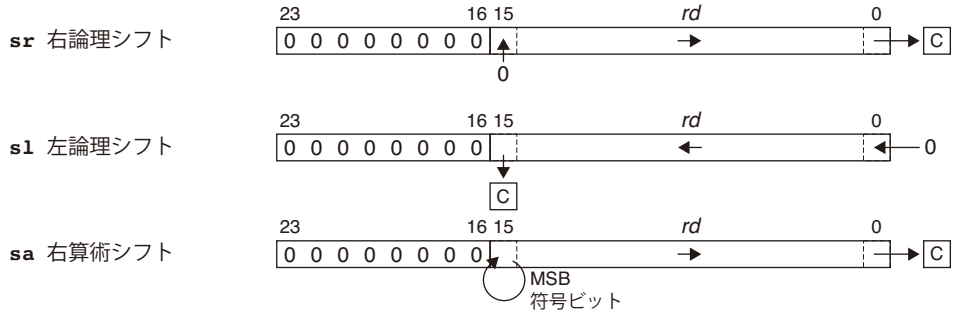
5.7 シフト/スワップ命令

S1C17コアの命令セットは、レジスタデータのシフト、スワップ命令をサポートしています。

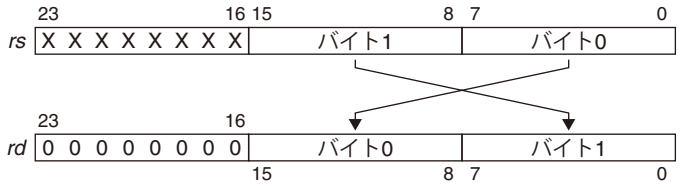
sr	右論理シフト
sl	左論理シフト
sa	右算術シフト
swap	上位と下位バイトの入れ替え

シフト操作は指定レジスタのビット15～0に対して行われ、ビット23～16は0となります。
シフト量は、オペランドの*imm5*または*rs*レジスタで次のように0～3ビット、4ビット、または8ビットの指定が可能です。
%rs/*imm7* = 0～3: 0～3ビットのシフト
%rs/*imm7* = 4～7: 4ビットのシフト(固定)
%rs/*imm7* = 8以上: 8ビットのシフト(固定)

例: `sr %rd,1` *%rd*のビット15～0を1ビット右論理シフト
`sl %rd,7` *%rd*のビット15～0を4ビット左論理シフト
`sa %rd,0xf` *%rd*のビット15～0を8ビット右算術シフト



スワップ命令は汎用レジスタの内容を下図のように入れ替えます。



5.8 分岐命令/ディレイド分岐命令

5.8.1 分岐命令の種類

(1)PC相対ジャンプ命令

PC相対ジャンプ命令には次のものがあります。

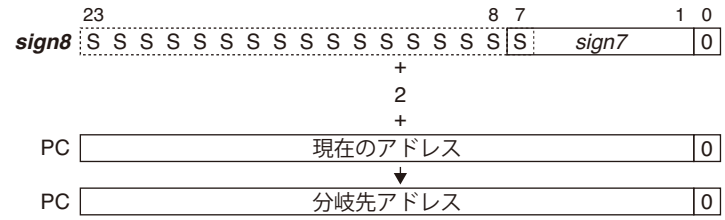
```
jr*  sign7
jpr  sign10
jpr  %rb
```

PC相対ジャンプ命令はリロケータブルなプログラミングに対応した分岐命令で、PC + 2のアドレス(分岐命令の次のアドレス)にオペランドで指定した符号付きのディスプレースメントを加えたアドレスに分岐します。
*sign7/10*または*rb*で分岐先までの命令ステップ数を指定します。しかし、S1C17コア内部では、命令長が16ビット固定のため、*sign7/10*の値を2倍して16ビット単位のワードアドレスとします。したがって、実際にPCに加算されるディスプレースメントは*sign7/10*を2倍にした符号付き8ビット/11ビットとなり(最下位ビットは常に0)、偶数アドレスに分岐します。*rb*レジスタによる指定時は、レジスタ値が2倍されずにPCに加算されます。
指定可能なディスプレースメントはext命令による拡張も可能で、それぞれ次のようになります。

分岐命令単独の場合

`jr*  sign7  ;` "`jr* sign8`"として機能します。(*sign8* = {*sign7*, 0})

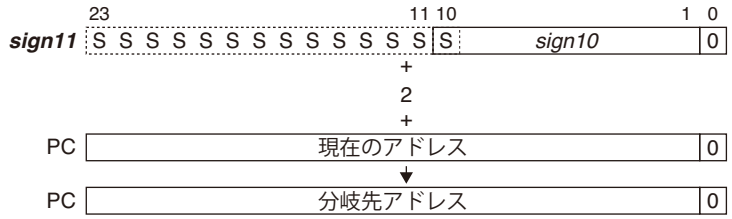
`jr*`命令単独では、符号付き7ビットのディスプレースメント(*sign7*)が指定可能です。



分岐範囲は(PC - 126)～(PC + 128)です。

`jpr  sign10  ;` "`jpr sign11`"として機能します。(*sign11* = {*sign10*, 0})

`jpr`命令単独では、符号付き10ビットのディスプレースメント(*sign10*)が指定可能です。

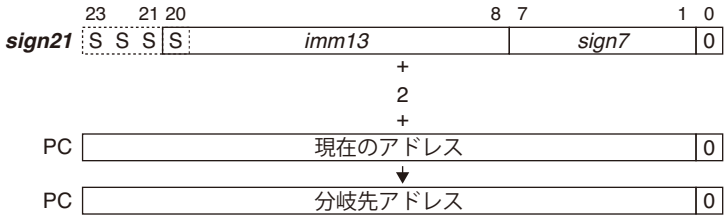


分岐範囲は(PC - 2,046)～(PC + 2,048)です。

ext命令を1個拡張した場合

```
ext   imm13
jr*   sign7    ; "jr* sign21"として機能します。(sign21 = {imm13, sign7, 0})
```

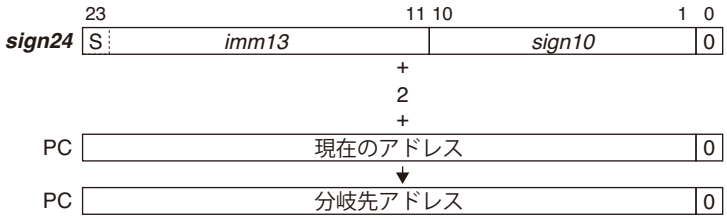
ext命令で指定した*imm13*を*sign21*の上位13ビットとして拡張します。



分岐範囲は(PC - 1,048,574)～(PC + 1,048,576)です。

```
ext   imm13
jpr   sign10   ; "jpr sign24"として機能します。(sign24 = {imm13, sign10, 0})
```

ext命令で指定した*imm13*を*sign24*の上位13ビットとして拡張します。

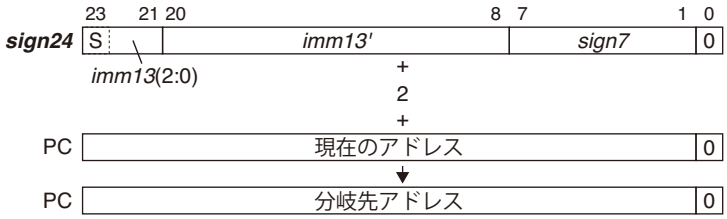


分岐範囲は(PC - 8,388,606)～(PC + 8,388,608)です。

ext命令を2個拡張した場合

```
ext   imm13
ext   imm13'
jr*   sign7    ; "jr* sign24"として機能します。
```

最初のext命令で指定した*imm13*はビット2～0の3ビットのみが有効で(上位10ビットを無視)、*sign24*は次のように構成されます。
sign24 = {*imm13*(2:0), *imm13'*, *sign7*, 0}



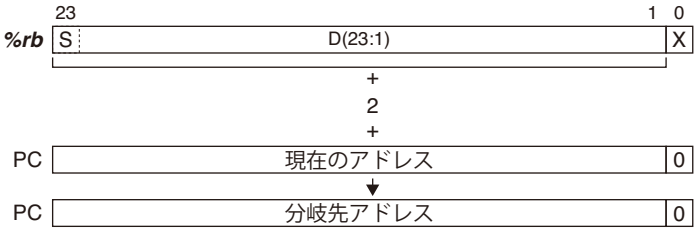
分岐範囲は(PC - 8,388,606)～(PC + 8,388,608)です。

上記の分岐範囲は論理的な値で、実際は使用するメモリ領域の範囲に制限されます。

jpr %rbの場合

jpr　%rb

*rb*で符号付き24ビットの相対値を指定可能です。
分岐アドレスは次のように構成されます。
{*rb*(23:1), 0}



*rb*レジスタの最下位ビットは常に0として扱われます。
分岐範囲は(PC - 8,388,606)～(PC + 8,388,608)です。

上記の分岐範囲は論理的な値で、実際は使用するメモリ領域の範囲に制限されます。

分岐条件

jpr命令は常にプログラムが分岐する無条件ジャンプ命令です。
jrで始まる命令は、それぞれフラグの組み合わせによる分岐条件が設定されており、その条件が満たされている場合にのみ指定アドレスに分岐する条件ジャンプ命令です。条件が合っていない場合は分岐しません。条件ジャンプ命令は、基本的にcmp命令による2つの値の比較結果を判定するために使用します。このため、各命令の名称には大小関係を表す文字が使用されています。
条件ジャンプ命令の種類と分岐条件を表5.8.1.1に示します。

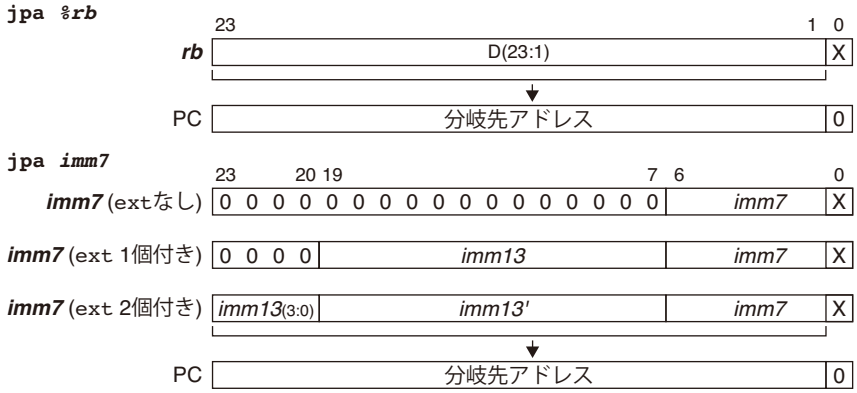
表5.8.1.1 条件ジャンプ命令と分岐条件

命　令		フラグ条件	A:Bの比較	備　考
jrgt	Greater Than	!Z & !(N ^ V)	A > B	符号付きデータ比較用
jrge	Greater or Equal	!(N ^ V)	A ≥ B	
jrlt	Less Than	N ^ V	A < B	
jrle	Less or Equal	Z I (N ^ V)	A ≤ B	
jrugt	Unsigned, Greater Than	!Z & !C	A > B	符号なしデータ比較用
jruge	Unsigned, Greater or Equal	!C	A ≥ B	
jrult	Unsigned, Less Than	C	A < B	
jrule	Unsigned, Less or Equal	Z I C	A ≤ B	
jreq	Equal	Z	A = B	
jrne	Not Equal	!Z	A ≠ B	

A:Bの比較は“cmp A,B”の場合です。

(2)絶対ジャンプ命令

絶対ジャンプ命令jpaは、指定の汎用レジスタ(*rb*)の内容または即値*imm7*(ext命令により*imm20*、*imm24*に拡張可能)を絶対アドレスとして無条件に分岐します。*rb*レジスタの内容または即値がPCにロードされると、その最下位ビットは常に0となります。



(3)PC相対コール命令

PC相対コール命令call *sign10*/%*rb*はリロケータブルなプログラミングに対応したサブルーチンコール命令で、PC + 2のアドレス(分岐命令の次のアドレス)にオペランドで指定した符号付きのディスプレースメントを加えたアドレスから始まるサブルーチンへ無条件に分岐します。分岐時には、callの次の命令のアドレス(ディレイド分岐時は2つ目の命令のアドレス)をリターンアドレスとしてスタックにセーブします。サブルーチンの最後にret命令を実行するとこのアドレスがPCにロードされ、サブルーチンからリターンします。
なお、命令長が16ビット固定のため、ディスプレースメントの最下位ビットは常に0として扱われ(*sign10*は2倍される、*rb*は2倍されない)、偶数アドレスに分岐します。
指定可能なディスプレースメントは、PC相対ジャンプ命令と同様にext命令による拡張も可能です。ディスプレースメントの拡張については前述の"(1)PC相対ジャンプ命令"を参照してください。

(4)絶対コール命令

絶対コール命令callaは、指定の汎用レジスタ(*rb*)の内容または即値*imm7*(ext命令により*imm20*、*imm24*に拡張可能)を絶対アドレスとして、そのアドレスから始まるサブルーチンを無条件にコールします。*rb*レジスタの内容または即値がPCにロードされると、その最下位ビットは常に0となります。("(2)絶対ジャンプ命令"参照)

(5)ソフトウェア割り込み

ソフトウェア割り込みintおよびintlはソフトウェアによって割り込みを発生させ、指定の割り込み処理ルーチンを実行するための命令です。オペランドの*imm5*によって、発生させる割り込みのベクタ番号を指定します。プロセッサはソフトウェア割り込みが発生すると、PSRとint/intlの次の命令アドレスをスタックにセーブし、ベクタテーブルから指定のベクタを読み出して割り込み処理ルーチンを実行します。したがって、割り込み処理ルーチンからのリターンにはPSRも復帰させるreti命令を使用する必要があります。
ソフトウェア割り込みの詳細については、"6.3 割り込み"を参照してください。

(6)リターン命令

ret命令はcallおよびcalla命令に対応するリターン命令で、スタックにセーブされているリターンアドレスをPCにロードしてサブルーチンを終了します。したがって、ret命令実行時のSPの値は、そのサブルーチンの実行開始時の値と同じ(リターンアドレスの位置を示している)でなければなりません。

reti命令は割り込み処理ルーチン用のリターン命令です。割り込み処理ではリターンアドレスとともにPSRもスタックにセーブされますので、reti命令によってPSRの内容を復帰させる必要があります。reti命令ではPC、PSRの順にスタックから読み出されます。ret命令と同様に、reti命令実行時と割り込み処理ルーチンの実行開始時のSPの値は同じでなければなりません。

(7)デバッグ割り込み

brk命令とretd命令はデバッグ割り込み処理ルーチンの呼び出しとリターンに使用します。基本的にはデバッグファームウェア用の命令のため、アプリケーションプログラムでは使用しないでください。
これらの命令の機能については、"6.5 デバッグ回路"を参照してください。

5.8.2 ディレイド分岐命令

S1C17コアは、パイプライン処理により命令の実行とフェッチを同時に行います。分岐命令実行時は続く命令がすでにフェッチされているため、分岐前にその命令を実行することによって分岐命令の実行サイクル数を1サイクル削減することができます。これがディレイド分岐機能で、分岐前に実行される命令(分岐命令の次のアドレスの命令)をディレイドスロット命令と呼びます。
ディレイド分岐機能が使用できる命令は以下のとおりで、ニーモニックでは分岐命令の後ろに".d"を付けて指定します。

ディレイド分岐命令

```
jrgt.d    jrge.d    jrlt.d    jrle.d    jrugt.d   jruge.d   jrult.d
jrule.d   jreq.d    jrne.d    call.d    calla.d   jpr.d     jpa.d
ret.d     reti.d
```

ディレイドスロット命令

以下の命令以外は、すべてディレイドスロット命令として使用可能です。

ディレイドスロット命令として使用できない命令

```
brk call calla ext halt int intl jpa jpr jr* ret retd reti slp
```

ディレイドスロット命令は、ext命令でオペランドを拡張することはできません。
ディレイドスロット命令は、ディレイド分岐命令が条件付きか、あるいは無条件かにかかわらず、また分岐するしないにかかわらず必ず実行されます。
ディレイド分岐でない分岐命令(".d"の付かないもの)では、分岐命令の次のアドレスの命令は、命令フローが分岐する場合は実行されませんが、条件分岐命令で分岐しなかった場合には、次のアドレスの命令が分岐命令に続く命令として実行されます。

call.dまたはcalla.d命令でスタックにセーブされるリターンアドレスはディレイドスロット命令の次の命令のアドレスとなり、サブルーチンからのリターン時にディレイドスロット命令は実行されません。

ディレイド分岐命令とディレイドスロット命令の間は、割り込みなどはハードウェアによってマスクされ発生しません。

リーフサブルーチンへの応用

ディレイド分岐命令を利用した、リーフサブルーチンの高速コール機能への応用例を次に示します。
例:

```
    jpr.d  SUB      ; ディレイド分岐命令でサブルーチンへジャンプ
    ld.a   %r7,%pc  ; ディレイドスロット命令でリターンアドレスを汎用レジスタにロード
    add.a  %r1,%r2  ; リターンアドレス
     :     :
SUB:
     :     :
    jpr    %r7      ; リターン
```

注: • **ld.a *%rd*,%pc命令はディレイドスロット命令として使用してください。ディレイド分岐命令の直後以外の場所に記述した場合、*rd*レジスタにロードされるPC値がld.aの次の命令のアドレスを示すとは限りません。**

• **ld.a *%rd*,%pc命令と組み合わせ可能なディレイド分岐命令は以下のとおりです。**
- jpr.d *%rb/sign10*
- jr*.d *sign7*
- jpa.d *%rb/imm7*

5.9 システム制御命令

以下の5つの命令はシステムを制御します。

nop	PCをインクリメントするのみで、他の動作を行いません。
halt	プロセッサをHALTモードにします。
slp	プロセッサをSLEEPモードにします。
ei	割り込みを許可します。
di	割り込みを禁止します。

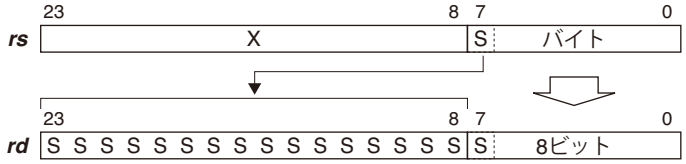
HALTモードとSLEEPモードについては、“6.4 パワーダウンモード”、および各機種のテクニカルマニュアルの説明を参照してください。
割り込み制御の詳細については、“6.3 割り込み”を参照してください。

5.10 コンバージョン命令

Cコンパイラへの対応のため、8/16/24/32ビット間のデータ変換命令が用意されています。

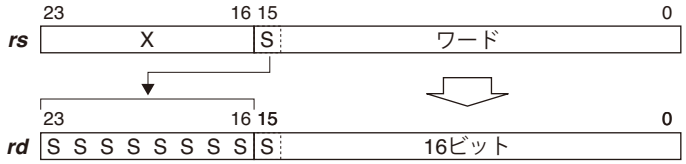
cv.ab %rd,%rs

8ビットを符号拡張して24ビットデータに変換します。



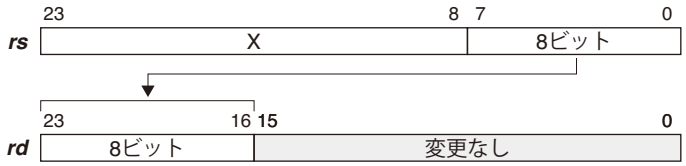
cv.as %rd,%rs

16ビットデータを符号拡張して24ビットデータに変換します。



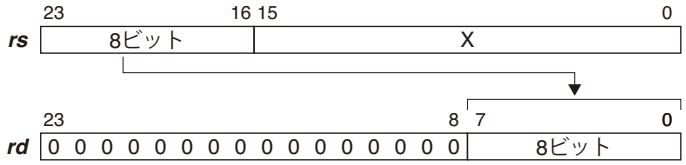
cv.al %rd,%rs

32ビットデータを24ビットデータに変換するため、上位8ビットを取得します。



cv.la %rd,%rs

24ビットデータを32ビットデータに変換するため、上位8ビットを取得します。



cv.ls %rd,%rs

16ビットデータを32ビットデータに変換するため、符号を拡張します。

5.11 コプロセッサ命令

S1C17コアはコプロセッサインタフェースを搭載しており、以下のコプロセッサ専用命令が用意されています。

ld.cw　コプロセッサへのデータ転送
ld.ca　コプロセッサへのデータ転送と結果、フラグ状態の入力
ld.cf　コプロセッサからフラグ状態を入力

ld.cwとld.ca命令はコプロセッサに対して、*rd*(データ0)と*rs*(データ1)レジスタに設定した2個の24ビットデータを転送します。データ1は*imm7*でも指定可能で、この場合はext命令を使用して即値を*imm20*、*imm24*に拡張できます。
ld.ca命令が入力する結果は*rd*レジスタにロードされます。
ld.caとld.cf命令によりコプロセッサから入力したフラグ状態は、PSR(C、V、Z、Nフラグ)に設定されます。
コプロセッサの具体的なコマンドやステータスは、接続するコプロセッサによって異なりますので、コプロセッサの説明書を参照してください。

6 機能

ここでは、S1C17コアのプロセッサ処理状態と動作の概要を説明します。

6.1 プロセッサの状態遷移

S1C17コアの状態遷移を図6.1.1に示します。

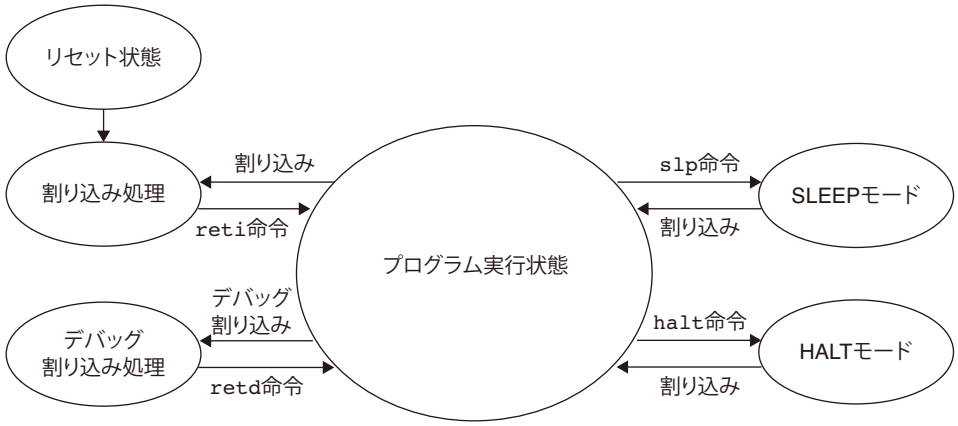


図6.1.1 プロセッサ状態遷移図

6.1.1 リセット状態

リセット信号がアクティブ状態になるとプロセッサは初期化され、リセット信号がインアクティブになるとプロセッサはリセットベクタから処理を開始します。

6.1.2 プログラム実行状態

S1C17コアがユーザプログラムをシーケンシャルに実行している状態です。
割り込みの発生、あるいはslp/halt命令の実行により他のプロセッサ状態に移行します。

6.1.3 割り込み処理

ソフトウェア割り込みなど、割り込みが発生するとプロセッサは割り込み処理状態に移行します。割り込み処理発生要因を以下に示します。

(1) 外部割り込み
(2) ソフトウェア割り込み
(3) アドレス不整割り込み
(4) NMI

6.1.4 デバッグ割り込み

S1C17コアは、アプリケーションの開発効率を向上させるため、デバッグを支援する機能を搭載しています。この機能を使用するためのモードがデバッグモードで、brk命令やデバッグ割り込みによってユーザモードから切り換わるようになっています。通常は、このモードになることはありません。

6.1.5 HALTモードとSLEEPモード

ソフトウェアでhalt命令またはslp命令を実行すると、プロセッサは消費電力を抑えるためのHALTモードまたはSLEEPモードに移行します(6.4節参照)。通常、HALTおよびSLEEPモードは、イニシャルリセット、NMI、外部割り込みの発生によって解除できます。

6.2 プログラムの実行

プロセッサは、イニシャルリセットが解除されるとリセットベクタ(リセット処理ルーチンのアドレス)をPCにロードして、そのアドレスから命令の実行を開始します。S1C17コアの命令は16ビットの固定長となっていますので、以後プロセッサはPCの示すアドレスから命令をフェッチするごとにPCに+2ずつ加算し、命令を順次実行します。
分岐命令が実行されるとプロセッサはPSRのフラグを検査し、分岐条件が成立していれば分岐先のアドレスをPCにロードします。
割り込みが発生すると、プロセッサはベクタテーブルから割り込み処理ルーチンのアドレスをPCにロードします。
ベクタテーブルはリセットベクタを先頭に割り込みベクタが書き込まれたテーブルで、TTBRレジスタ(0xffff80)に設定されたアドレスに配置されます。先頭アドレスはコンフィギュレーションで設定できます。

6.2.1 命令フェッチと実行

S1C17コアは内部で3段のパイプライン処理を行って、分岐やメモリアドレスインクリメント/デクリメント付きデータ転送命令を除く基本命令を1クロックで実行します。
パイプライン処理は、フェッチと実行を同時に行うことで処理時間の高速化を図るもので、3段のパイプラインでは各命令を3つのステージに分けて平行処理します。

基本命令ステージ

命令フェッチ	命令デコード	命令実行/メモリアクセス/レジスタライト

以降、各ステージは次の記号で表します。
F (Fetch): 命令フェッチ
D (Decode): 命令デコード
E(Execute): 命令実行、メモリアクセス、レジスタライト

パイプライン動作

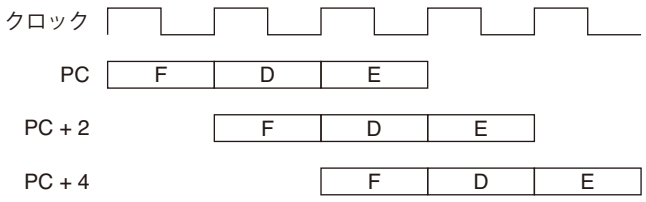


図6.2.1.1 パイプライン動作

注: 上記のパイプライン動作は内部メモリを使用したときのものです。外部メモリや低速外部デバイスの使用時はデバイスによって任意のウェイトサイクルが挿入され、Eステージでウェイトします。

6.2.2 実行サイクルとフラグ

以下に、ハーバードバスに接続した1サイクルアクセスのメモリでの実行サイクル数とフラグの変化の一覧を示します。
機種により、S1C17コア外部のバス調停回路に費やされるクロックサイクルや、接続されているデバイスの固有ウェイトサイクルが付加される場合があります。

表6.2.2.1 命令実行サイクルクロック数とフラグ変化

分 類	ニーモニック		サイクル	フラグ						備 考
				IL	IE	C	V	Z	N	
データ転送	ld.b	%rd,%rs	1	–	–	–	–	–	–	*1 ext未使用時: 1サイクル ext使用時: 2サイクル
		%rd,[%rb]	1–2*1	–	–	–	–	–	–	
		%rd,[%rb]+	2	–	–	–	–	–	–	
		%rd,[%rb]-	2	–	–	–	–	–	–	
		%rd,-[%rb]	2	–	–	–	–	–	–	
		%rd,[%sp+imm7]	2	–	–	–	–	–	–	
		%rd,[imm7]	1	–	–	–	–	–	–	
		[%rb],%rs	1–2*1	–	–	–	–	–	–	
		[%rb]+,%rs	2	–	–	–	–	–	–	
		[%rb]-,%rs	2	–	–	–	–	–	–	
		-[%rb],%rs	2	–	–	–	–	–	–	
		[%sp+imm7],%rs	2	–	–	–	–	–	–	
		[imm7],%rs	1	–	–	–	–	–	–	
	ld.ub	%rd,%rs	1	–	–	–	–	–	–	
		%rd,[%rb]	1–2*1	–	–	–	–	–	–	
		%rd,[%rb]+	2	–	–	–	–	–	–	
		%rd,[%rb]-	2	–	–	–	–	–	–	
		%rd,-[%rb]	2	–	–	–	–	–	–	
		%rd,[%sp+imm7]	2	–	–	–	–	–	–	
		%rd,[imm7]	1	–	–	–	–	–	–	
	ld	%rd,%rs	1	–	–	–	–	–	–	
		%rd,sign7	1	–	–	–	–	–	–	
		%rd,[%rb]	1–2*1	–	–	–	–	–	–	
		%rd,[%rb]+	2	–	–	–	–	–	–	
		%rd,[%rb]-	2	–	–	–	–	–	–	
		%rd,-[%rb]	2	–	–	–	–	–	–	
		%rd,[%sp+imm7]	2	–	–	–	–	–	–	
		%rd,[imm7]	1	–	–	–	–	–	–	
		[%rb],%rs	1–2*1	–	–	–	–	–	–	
		[%rb]+,%rs	2	–	–	–	–	–	–	
		[%rb]-,%rs	2	–	–	–	–	–	–	
		-[%rb],%rs	2	–	–	–	–	–	–	
		[%sp+imm7],%rs	2	–	–	–	–	–	–	
		[imm7],%rs	1	–	–	–	–	–	–	
	ld.a	%rd,%rs	1	–	–	–	–	–	–	
		%rd,imm7	1	–	–	–	–	–	–	
		%rd,[%rb]	1–2*1	–	–	–	–	–	–	
		%rd,[%rb]+	2	–	–	–	–	–	–	
		%rd,[%rb]-	2	–	–	–	–	–	–	
		%rd,-[%rb]	2	–	–	–	–	–	–	
		%rd,[%sp+imm7]	2	–	–	–	–	–	–	
		%rd,[imm7]	1	–	–	–	–	–	–	
		[%rb],%rs	1–2*1	–	–	–	–	–	–	
		[%rb]+,%rs	2	–	–	–	–	–	–	
		[%rb]-,%rs	2	–	–	–	–	–	–	
		-[%rb],%rs	2	–	–	–	–	–	–	
		[%sp+imm7],%rs	2	–	–	–	–	–	–	
		[imm7],%rs	1	–	–	–	–	–	–	
		%rd,%sp	1	–	–	–	–	–	–	
		%rd,%pc	1	–	–	–	–	–	–	
		%rd,[%sp]	1–2*1	–	–	–	–	–	–	
		%rd,[%sp]+	2	–	–	–	–	–	–	
		%rd,[%sp]-	2	–	–	–	–	–	–	
		%rd,-[%sp]	2	–	–	–	–	–	–	

分　類	ニーモニック		サイクル	フラグ						備　考
				IL	IE	C	V	Z	N	
データ転送	ld.a	[%sp],%*rs*	1–2*1	–	–	–	–	–	–	*1 ext未使用時: 1サイクル ext使用時:　2サイクル
		[%sp]+,%*rs*	2	–	–	–	–	–	–	
		[%sp]-,%*rs*	2	–	–	–	–	–	–	
		-[%sp],%*rs*	2	–	–	–	–	–	–	
		%sp,%*rs*	1	–	–	–	–	–	–	
		%sp,*imm7*	1	–	–	–	–	–	–	
整数算術演算	add	%*rd*,%*rs*	1	–	–	↔	↔	↔	↔	
	add/c	%*rd*,%*rs*	1	–	–	–	↔	↔	↔	
	add/nc	%*rd*,%*rs*	1	–	–	–	↔	↔	↔	
	add	%*rd*,*imm7*	1	–	–	↔	↔	↔	↔	
	add.a	%*rd*,%*rs*	1	–	–	–	–	–	–	
	add.a/c	%*rd*,%*rs*	1	–	–	–	–	–	–	
	add.a/nc	%*rd*,%*rs*	1	–	–	–	–	–	–	
	add.a	%sp,%*rs*	1	–	–	–	–	–	–	
		%*rd*,*imm7*	1	–	–	–	–	–	–	
		%sp,*imm7*	1	–	–	–	–	–	–	
	adc	%*rd*,%*rs*	1	–	–	↔	↔	↔	↔	
	adc/c	%*rd*,%*rs*	1	–	–	–	↔	↔	↔	
	adc/nc	%*rd*,%*rs*	1	–	–	–	↔	↔	↔	
	adc	%*rd*,*imm7*	1	–	–	↔	↔	↔	↔	
	sub	%*rd*,%*rs*	1	–	–	↔	↔	↔	↔	
	sub/c	%*rd*,%*rs*	1	–	–	–	↔	↔	↔	
	sub/nc	%*rd*,%*rs*	1	–	–	–	↔	↔	↔	
	sub	%*rd*,*imm7*	1	–	–	↔	↔	↔	↔	
	sub.a	%*rd*,%*rs*	1	–	–	–	–	–	–	
	sub.a/c	%*rd*,%*rs*	1	–	–	–	–	–	–	
	sub.a/nc	%*rd*,%*rs*	1	–	–	–	–	–	–	
	sub.a	%sp,%*rs*	1	–	–	–	–	–	–	
		%*rd*,*imm7*	1	–	–	–	–	–	–	
		%sp,*imm7*	1	–	–	–	–	–	–	
	sbc	%*rd*,%*rs*	1	–	–	↔	↔	↔	↔	
	sbc/c	%*rd*,%*rs*	1	–	–	–	↔	↔	↔	
	sbc/nc	%*rd*,%*rs*	1	–	–	–	↔	↔	↔	
	sbc	%*rd*,*imm7*	1	–	–	↔	↔	↔	↔	
	cmp	%*rd*,%*rs*	1	–	–	↔	↔	↔	↔	
	cmp/c	%*rd*,%*rs*	1	–	–	–	↔	↔	↔	
	cmp/nc	%*rd*,%*rs*	1	–	–	–	↔	↔	↔	
	cmp	%*rd*,*sign7*	1	–	–	↔	↔	↔	↔	
	cmp.a	%*rd*,%*rs*	1	–	–	↔	–	↔	–	
	cmp.a/c	%*rd*,%*rs*	1	–	–	–	–	↔	–	
	cmp.a/nc	%*rd*,%*rs*	1	–	–	–	–	↔	–	
	cmp.a	%*rd*,*imm7*	1	–	–	↔	–	↔	–	
	cmc	%*rd*,%*rs*	1	–	–	↔	↔	↔	↔	
	cmc/c	%*rd*,%*rs*	1	–	–	–	↔	↔	↔	
	cmc/nc	%*rd*,%*rs*	1	–	–	–	↔	↔	↔	
	cmc	%*rd*,*sign7*	1	–	–	↔	↔	↔	↔	
論理演算	and	%*rd*,%*rs*	1	–	–	–	0	↔	↔	
	and/c	%*rd*,%*rs*	1	–	–	–	0	↔	↔	
	and/nc	%*rd*,%*rs*	1	–	–	–	0	↔	↔	
	and	%*rd*,*sign7*	1	–	–	–	0	↔	↔	
	or	%*rd*,%*rs*	1	–	–	–	0	↔	↔	
	or/c	%*rd*,%*rs*	1	–	–	–	0	↔	↔	
	or/nc	%*rd*,%*rs*	1	–	–	–	0	↔	↔	
	or	%*rd*,*sign7*	1	–	–	–	0	↔	↔	
	xor	%*rd*,%*rs*	1	–	–	–	0	↔	↔	
	xor/c	%*rd*,%*rs*	1	–	–	–	0	↔	↔	
	xor/nc	%*rd*,%*rs*	1	–	–	–	0	↔	↔	
	xor	%*rd*,*sign7*	1	–	–	–	0	↔	↔	
	not	%*rd*,%*rs*	1	–	–	–	0	↔	↔	
	not/c	%*rd*,%*rs*	1	–	–	–	0	↔	↔	
	not/nc	%*rd*,%*rs*	1	–	–	–	0	↔	↔	
	not	%*rd*,*sign7*	1	–	–	–	0	↔	↔	

分　類	ニーモニック		サイクル	フラグ						備　考
				IL	IE	C	V	Z	N	
シフト&スワップ	sr	%rd,%rs	1	–	–	↔	–	↔	↔	
		%rd,imm7	1	–	–	↔	–	↔	↔	
	sa	%rd,%rs	1	–	–	↔	–	↔	↔	
		%rd,imm7	1	–	–	↔	–	↔	↔	
	sl	%rd,%rs	1	–	–	↔	–	↔	↔	
		%rd,imm7	1	–	–	↔	–	↔	↔	
	swap	%rd,%rs	1	–	–	–	–	–	–	
即値拡張	ext	imm13	1	–	–	–	–	–	–	
コンバージョン	cv.ab	%rd,%rs	1	–	–	–	–	–	–	
	cv.as	%rd,%rs	1	–	–	–	–	–	–	
	cv.al	%rd,%rs	1	–	–	–	–	–	–	
	cv.la	%rd,%rs	1	–	–	–	–	–	–	
	cv.ls	%rd,%rs	1	–	–	–	–	–	–	
分岐	jpr	sign10	3	–	–	–	–	–	–	*2 非分岐時: 2サイクル 分岐時:　3サイクル *3 1サイクルディレイドスロット命令が続く場合の値 2サイクルディレイドスロット命令が続く場合は.dなしと同様
	jpr.d	%rb	2(.d)*3							
	jpa	imm7	3	–	–	–	–	–	–	
	ipa.d	%rb	2(.d)*3							
	jrgt	sign7	2–3*2	–	–	–	–	–	–	
	jrgt.d		2(.d)*3							
	jrge	sign7	2–3*2	–	–	–	–	–	–	
	jrge.d		2(.d)*3							
	jrlt	sign7	2–3*2	–	–	–	–	–	–	
	jrlt.d		2(.d)*3							
	jrle	sign7	2–3*2	–	–	–	–	–	–	
	jrle.d		2(.d)*3							
	jrugt	sign7	2–3*2	–	–	–	–	–	–	
	jrugt.d		2(.d)*3							
	jruge	sign7	2–3*2	–	–	–	–	–	–	
	jruge.d		2(.d)*3							
	jrult	sign7	2–3*2	–	–	–	–	–	–	
	jrult.d		2(.d)*3							
	jrule	sign7	2–3*2	–	–	–	–	–	–	
	jrule.d		2(.d)*3							
	jreq	sign7	2–3*2	–	–	–	–	–	–	
	jreq.d		2(.d)*3							
	jrne	sign7	2–3*2	–	–	–	–	–	–	
	jrne.d		2(.d)*3							
	call	sign10	4	–	–	–	–	–	–	
	call.d	%rb	3(.d)*3							
	calla	imm7	4	–	–	–	–	–	–	
	calla.d	%rb	3(.d)*3							
	ret		3	–	–	–	–	–	–	
	ret.d		2(.d)*3							
	int	imm5	3	–	0	–	–	–	–	
	intl	imm5,imm3	3	↔	0	–	–	–	–	
	reti		3	↔	↔	↔	↔	↔	↔	
	reti.d		2(.d)*3							
	brk		4	–	0	–	–	–	–	
	retd		4	↔	↔	↔	↔	↔	↔	
システム制御	nop		1	–	–	–	–	–	–	
	halt		6	–	–	–	–	–	–	
	slp		6	–	–	–	–	–	–	
	ei		1	–	1	–	–	–	–	
	di		1	–	0	–	–	–	–	
コプロセッサ制御	ld.cw	%rd,%rs	1	–	–	–	–	–	–	
		%rd,imm7								
	ld.ca	%rd,%rs	1	–	–	↔	↔	↔	↔	
		%rd,imm7								
	ld.cf	%rd,%rs	1	–	–	↔	↔	↔	↔	
		%rd,imm7								

6.3 割り込み

プロセッサはプログラム実行中に、割り込みが発生すると割り込み処理状態となります。割り込み処理状態は、各割り込み要因に対応したユーザの処理ルーチンに分岐するまでのプロセスで、分岐後は再びプログラム実行状態に戻ります。

6.3.1 割り込みの優先順位

S1C17コアがサポートする割り込みとベクタアドレス、優先順位を下表に示します。

表6.3.1.1 割り込みのベクタアドレスと優先順位

割り込み	ベクタアドレス(Hex)	優先順位
リセット	TTBR + 0x00	高い
アドレス不整割り込み	TTBR + 0x04	
デバッグ割り込み	(0xfffc00)	
NMI	TTBR + 0x08	
ソフトウェア割り込み	TTBR + 0x00 ～ TTBR + 0x7c	
マスク可能な外部割り込み	TTBR + 0x00 ～ TTBR + 0x7c	低い

同時に割り込みが発生した場合、より優先順位の高いものから割り込み処理を行います。
割り込みが発生すると、プロセッサはそれ以降の割り込みを禁止し、割り込み処理に移行します。多重割り込みに対応するには、割り込み処理ルーチンの中でPSRのIEフラグを1にして割り込み処理中の割り込み発生を許可します。基本的には多重割り込みを行う場合でも、IL[2:0]ビットによって同レベル以下の割り込みの発生は禁止されます。

デバッグ割り込みでは、ベクタテーブルは参照されません。また、PCとPSRの退避にもスタックは使用されず、R0とともに特定の領域に格納されます。
デバッグ割り込み発生時に参照されるアドレスは、次のとおりです。

表6.3.1.2 デバッグ割り込み処理開始アドレスとレジスタ退避領域

アドレス	内　容
0xfffc00	デバッグ割り込み処理開始アドレス
DBRAM設定値 + 0x00	PC/PSR退避領域
DBRAM設定値 + 0x04	R0退避領域

(DBRAM: 4.2.3節参照)

なお、デバッグ割り込み処理中は、他の割り込みおよび多重のデバッグ割り込みは受け付けられません。デバッグ割り込み処理終了後に受け付けられます。

6.3.2 ベクタテーブル

S1C17コアのベクタテーブル

ベクタテーブルを参照するS1C17コアの割り込みを表6.3.2.1に示します。

表6.3.2.1 ベクター一覧

ベクタNo./ソフトウェア割り込みNo.	割り込み	ベクタアドレス
0 (0x00)	リセット	TTBR + 0x00
1 (0x01)	アドレス不整割り込み	TTBR + 0x04
2 (0x02)	NMI	TTBR + 0x08
3 (0x03)	マスク可能な外部割り込み3	TTBR + 0x0c
:	:	:
31 (0x1f)	マスク可能な外部割り込み31	TTBR + 0x7c

ベクタアドレスは、各割り込みが発生した場合に実行するユーザの割り込み処理ルーチンへのベクタ(分岐先アドレス)を格納しておくアドレスです。アドレス値を格納しておくため、それぞれ16ビット境界に配置されます。このベクタを格納しておくメモリ領域をベクタテーブルと呼び、ベクタアドレス欄に示した“TTBR”はベクタテーブルのベース(先頭)アドレスを表します。TTBR値については、各機種のテクニカルマニュアルを参照してください。この設定値は0xffff80番地のTTBR(ベクタテーブルベースレジスタ)から読み出すことができます。

6.3.3 割り込み処理

割り込みが発生するとプロセッサは割り込み処理を開始します。(ここで説明する割り込み処理は、リセット、デバッグ割り込みには適用されません。)
以下に、割り込み処理の動作を示します。

(1)実行中の命令列を中断します。
割り込みは実行中の命令が終了するシステムクロックの立ち上がりエッジに同期して発生します。

(2)PC、PSRの順にそれぞれの内容をスタック(SP)に退避させます。

(3)PSRのIE(割り込みイネーブル)ビットをクリアし、それ以降のマスク可能な割り込みを禁止します。
発生した割り込みがマスク可能な割り込みの場合は、PSRのIL(割り込みレベル)を発生した割り込みのレベルに変更します。

(4)ベクタテーブルから、発生した割り込みのベクタを読み出しPCにセットします。これにより、ユーザの割り込み処理ルーチンに分岐します。

ユーザの割り込み処理ルーチンでは最後に`reti`命令を実行する必要があります。`reti`命令はPC、PSRの順にスタックからデータを復帰させ、中断していた命令列に処理を戻します。

6.3.4 リセット

プロセッサのrst_n端子にLowパルスを入力することで、プロセッサがリセットされます。これにより内部レジスタが0にクリアされます。
プロセッサはリセットパルスの立ち上がりエッジで動作を開始し、リセット処理を行います。リセット処理ではベクタテーブルの先頭からリセットベクタが読み出され、PCにセットされます。これにより、ユーザの初期化ルーチンに分岐してプログラムの実行を開始します。リセット処理は他のすべての処理に優先します。

6.3.5 アドレス不整割り込み

メモリやI/O領域をアクセスするロード命令は、命令により転送するデータサイズが決まっています。そのアドレスはデータサイズごとの境界でなければなりません。

命令	転送データサイズ	アドレス
ld.b/ld.ub	バイト(8ビット)	バイト境界(全アドレスが対象)
ld	16ビット	16ビット境界(アドレスの最下位ビットが0)
ld.a	32ビット	32ビット境界(アドレスの下位2ビットが00)

ロード命令の指定アドレスがこの条件を満たしていない場合、プロセッサはアドレス不整割り込みとして割り込み処理に移行します。この場合でも、アドレスの最下位ビットまたは下位2ビットを0としてロード命令は実行されます。割り込み処理でスタックにセーブするPC値は、割り込みを発生したロード命令のアドレスとなります。

プログラムの分岐命令では、PCの最下位ビットが常に0に固定されるため、この割り込みは発生しません。割り込み処理のベクタについても同様です。

6.3.6 NMI

プロセッサのnmi_n入力がアクティブになるとNMIが発生します。NMIが発生すると、プロセッサは実行中の命令を終了後、割り込み処理に移行します。

6.3.7 マスク可能な外部割り込み

S1C17コアは32種類までのマスク可能な外部割り込みを受け付けることができます(ただし、最初の3つはリセット割り込み、アドレス不整割り込み、NMIと同一のベクタアドレスを使用します)。
マスク可能な割り込みは、PSRのIE(割り込みイネーブル)フラグがセットされている場合にのみプロセッサが受け付けます。また、PSRのIL(割り込みレベル)フィールドにより受け付け可能な割り込みのレベルが制限されます。ILフィールドの割り込みレベル(0～7)はプロセッサが受け付け可能な割り込みレベルを示し、その値より大きいレベルの割り込みのみを受け付けます。同じ値の割り込みは受け付けません。
IEフラグはソフトウェアで設定可能です。また、割り込み発生時にはPSRをスタックにセーブ後、IEフラグは0(割り込み禁止)にクリアされ、処理ルーチン内でIEフラグをセットするか、PSRを復帰させるreti命令で処理ルーチンを終了させるまで、マスク可能な割り込みを禁止します。ILフィールドも発生した割り込みのレベルに設定されます。
割り込み処理ルーチン内でIEフラグをセットすることによって、現在処理中の割り込みよりも高いレベルの割り込みを受け付ける多重割り込みが容易に実現できます。
プロセッサがリセットされた場合はPSRが0に初期化されるため、マスク可能な割り込みは禁止され、割り込みレベルは0(1～7の割り込みレベルを許可)に設定されます。

マスク可能な割り込みの発生手順とプロセッサの割り込み処理の内容は次のとおりです。

(1)実行中の命令列を中断します。
　割り込みは、実行中の命令が終了するシステムクロックの立ち上がりエッジに同期して受け付けられます。

(2)プロセッサはPC、PSRの順に各レジスタの内容をスタック(SP)に退避させます。

(3)PSRのIEフラグをクリアし、発生した割り込みの割り込みレベルをILフィールドにコピーします。

(4)プロセッサは割り込みに対応したベクタテーブル内のベクタアドレスからベクタを読み出してPCにセットし、割り込み処理ルーチンに分岐します。

割り込み処理ルーチンでは、処理の最後にreti命令を実行する必要があります。reti命令は、PC、PSRの順にスタックからデータを復帰させ、中断していた命令列に処理を戻します。

6.3.8 ソフトウェア割り込み

S1C17コアでは、ソフトウェアによって割り込みを発生させることができます。このための命令が、int *imm5*とintl *imm5*,*imm3*です。オペランドの即値*imm5*でベクタテーブルのベクタ番号(0～31)を指定します。intl命令では、*imm3*でPSRのILフィールドに設定する割り込みレベル(0～7)を指定することもできます。
プロセッサの割り込み処理の内容は、ハードウェアによる割り込み発生時と同様です。

6.3.9 割り込みマスク区間

以下の命令間では、アドレス不整割り込み、NMI、デバッグ割り込み、マスク可能な外部割り込みはマスクされ発生しません(保留状態)。マスク区間が終わると、保留状態の割り込みが受け付けられます。

(1)ext命令と次の命令の間
(2)ディレイド分岐命令(.d)とディレイドスロット命令の間
(3)retdと次の命令(戻り先命令)の間
(4)reti、reti.d*1と次の命令(戻り先命令)の間*2
(5)int、ei、di、slp、haltと次の命令の間*2
(6)条件不成立の条件ジャンプ(jr*)命令と次の命令の間*2

*1 reti.dでは、ディレイドスロット命令と戻り先命令を実行後に割り込みが受け付けられます。

reti.d	
ディレイドスロット命令	割り込みマスク
戻り先命令	まだ割り込みマスク状態のため、割り込みが可能となる前に1命令は実行されます。
次の命令	割り込みマスク解除

*2 (4)～(6)の条件でも、デバッグ割り込みのみは発生します。

6.4 パワーダウンモード

S1C17コアはHALTモードとSLEEPモードの2種類のパワーダウンモードをサポートしています。

HALTモード

S1C17コアがhalt命令を実行すると、その時点でプログラムの実行を中断しHALTモードに移行します。HALTモードではS1C17コアだけが動作を停止するのが一般的ですが、コア外部のクロック制御回路のインプリメンテーションに依存します。詳細については、機種別のテクニカルマニュアルを参照してください。

SLEEPモード

S1C17コアがslp命令を実行すると、その時点でプログラムの実行を中断しSLEEPモードに移行します。SLEEPモードではS1C17コアおよびチップ上の周辺回路も動作を停止するのが一般的で、HALTモードよりも大幅に消費電流を低減することができます。ただし、動作を停止するモジュールはコア外部のクロック制御回路のインプリメンテーションに依存します。詳細については、機種別のテクニカルマニュアルを参照してください。

HALT, SLEEPモードの解除

HALT、SLEEPモードを解除する要因はイニシャルリセット以外、S1C17コア外部のクロック制御回路のインプリメンテーションに依存します。詳細については機種別のテクニカルマニュアルを参照してください。
一般的にはイニシャルリセット、マスク可能な外部割り込み、NMI、デバッグ割り込みによって解除します。

割り込みによるHALT、SLEEPモードの解除には、プロセッサの割り込み許可/禁止の状態は影響しません。PSRのIEフラグや、割り込みコントローラの割り込み許可ビット(インプリメント依存)などが割り込み禁止に設定されている場合でも、割り込み信号によりHALT、SLEEPモードを解除することができます。
プロセッサが割り込み許可の状態で、割り込みによってHALT、SLEEPモードを解除した場合は、haltまたはslp命令の次の命令実行後、対応する割り込み処理ルーチンを実行します。
プロセッサが割り込み禁止状態の場合、HALT、SLEEPモードを解除後はhaltまたはslpの次の命令から実行を開始します。

6.5 デバッグ回路

S1C17コアには、プログラム開発を支援するデバッグ回路が設けられています。

6.5.1 デバッグ機能

デバッグ回路がサポートしている機能は以下のとおりです。

- **命令ブレーク**
  設定した命令のアドレスを実行する前にデバッグ割り込みを発生します。2カ所のアドレスに命令ブレークを設定できます。
- **シングルステップ**
  各命令ごとにデバッグ割り込みを発生します。
- **強制ブレーク**
  外部入力信号でデバッグ割り込みを発生します。
- **ソフトウェアブレーク**
  brk命令の実行によりデバッグ割り込みを発生します。

デバッグ割り込みが発生すると、プロセッサは次の処理を行います。

(1)実行中の命令列を中断します。

(2)プロセッサは、PCとPSR、R0の順にそれぞれの内容を以下のアドレスに格納します。
PCとPSR → DBRAM + 0x0
R0 → DBRAM + 0x4 （DBRAM: ユーザRAM内のデバッグ用ワークエリア先頭アドレス）

(3)プロセッサは、アドレス0xfffc00番地からPCにロードし、デバッグ割り込み処理ルーチンに分岐します。

割り込み処理ルーチンでは、処理の最後にretd命令を実行して中断している命令列に復帰します。
retd命令で復帰する際、プロセッサはR0、PCとPSRの順でデータを復帰します。
デバッグ割り込み中はハードウェア割り込みおよびNMIは受け付けられません。

6.5.2 リソース要件とデバッグツール

デバッグを行うには、64バイトのデバッグ用ワークエリアが必要です。デバッグ用ワークエリアについては、各機種のテクニカルマニュアルを参照してください。

デバッグは、S1C17コアのデバッグ端子にシリアルICEを接続し、パソコン上のデバッガからデバッグコマンドを入力して行います。このため、以下のツールが必要です。
• S1C17 Family シリアルICE（S5U1C17001H）
• S1C17 Family Cコンパイラパッケージ

6.5.3 デバッグ用レジスタ

コア予約I/Oエリアに以下のデバッグ用レジスタが配置されています。

0xFFFF90: Debug RAM Base Register (DBRAM)

Register name	Address	Bit	Name	Function	Setting	Init.	R/W	Remarks
Debug RAM base register	FFFF90 (L)	D31–24	–	Unused (fixed at 0)	0x0	0x0	R	
		D23	DBRAM23	Debug RAM base address	0x0–0xFFFDC0	*	R	Initial value is set in the C17 RTL-define DBRAM_BASE.
		I	I	DBRAM[5:0] is fixed at 0x0.	(64 byte units)			
		D0	DBRAM0					

D[23:0] DBRAM[23:0]: Debug RAM Base Address Bits

デバッグ用ワークエリア(64バイト)の先頭アドレスが格納されるリードオンリレジスタです。

0xFFFFA0: Debug Control Register (DCR)

Register name	Address	Bit	Name	Function	Setting				Init.	R/W	Remarks
Debug control register	FFFFA0 (B)	D7–5	–	Reserved	–				–	–	0 when being read.
		D4	DR	Debug request flag	1	Occurred	0	Not occurred	0	R/W	Reset by writing 1.
		D3	IBE1	Instruction break #1 enable	1	Enable	0	Disable	0	R/W	
		D2	IBE0	Instruction break #0 enable	1	Enable	0	Disable	0	R/W	
		D1	SE	Single step enable	1	Enable	0	Disable	0	R/W	
		D0	DM	Debug mode	1	Debug mode	0	User mode	0	R	

D[7:5] Reserved

D4 DR: Debug Request Flag

外部からのデバッグ要求の有無を示します。
1(R): 発生
0(R): なし(デフォルト)
1(W): フラグをリセット
0(W): 無効

このフラグは、1の書き込みでクリア(0にリセット)されます。デバッグ処理ルーチンを`retd`命令で終了する前にクリアしておく必要があります。

D3 IBE1: Instruction Break #1 Enable Bit

命令ブレーク#1を許可/禁止します。
1(R/W): 許可
0(R/W): 禁止(デフォルト)

このビットを1に設定すると、命令フェッチアドレスとInstruction Break Address Register 1 (0xffffb4)の設定値が比較され、一致すると命令ブレークが発生します。このビットを0に設定すると、比較は行われません。

D2 IBE0: Instruction Break #0 Enable Bit

命令ブレーク#0を許可/禁止します。
1(R/W): 許可
0(R/W): 禁止(デフォルト)

このビットを1に設定すると、命令フェッチアドレスとInstruction Break Address Register 0 (0xffffb0)の設定値が比較され、一致すると命令ブレークが発生します。このビットを0に設定すると、比較は行われません。

D1 SE: Single Step Enable Bit

シングルステップ動作を許可/禁止します。
1(R/W): 許可
0(R/W): 禁止(デフォルト)

D0 DM: Debug Mode Bit

プロセッサの動作モード(デバッグモードまたはユーザモード)を示します。
1(R): デバッグモード
0(R): ユーザモード(デフォルト)

0xFFFFB0: Instruction Break Address Register 0 (IBAR0)

Register name	Address	Bit	Name	Function	Setting	Init.	R/W	Remarks
Instruction break address register 0	FFFFB0 (L)	D31–24	–	Unused (fixed at 0)	0x0	0x0	R	
		D23	IBAR023	Instruction break address #0	0x0–0xFFFDE	0x0	R/W	
		I	I	IBAR00 is fixed at 0.				
		D0	IBAR00					

D[23:0] IBAR0[23:0]: Instruction Break Address #0
命令ブレークアドレス#0を設定します。（デフォルト: 0x000000）

0xFFFFB4: Instruction Break Address Register 1 (IBAR1)

Register name	Address	Bit	Name	Function	Setting	Init.	R/W	Remarks
Instruction break address register 1	FFFFB4 (L)	D31–24	–	Unused (fixed at 0)	0x0	0x0	R	
		D23	IBAR123	Instruction break address #1	0x0–0xFFFDE	0x0	R/W	
		I	I	IBAR10 is fixed at 0.				
		D0	IBAR10					

D[23:0] IBAR1[23:0]: Instruction Break Address #1
命令ブレークアドレス#1を設定します。（デフォルト: 0x000000）

0xFFFFC0: Serial Status Register for Debugging (SSR)

Register name	Address	Bit	Name	Function	Setting				Init.	R/W	Remarks
Serial status register for debugging	FFFFC0 (B)	D7–3	–	Reserved	–				–	–	0 when being read.
		D2	RXDEN	Receive disable	1	Disable	0	Enable	1	R/W	
		D1	TDBE	Transmit data buffer empty flag	1	Empty	0	Not empty	1	R	
		D0	RDBF	Receive data buffer full flag	1	Full	0	Not full	0	R	

D[7:3] Reserved

D2 RXDEN: Receive Disable Bit
オンチップデバッグモニタ用シリアルインタフェースの受信を許可/禁止します。
1(R/W): 禁止(デフォルト)
0(R/W): 許可

D1 TDBE: Transmit Data Buffer Empty Flag
オンチップデバッグモニタ用シリアルインタフェースの送信データバッファの状態を示します。
1(R): 空(デフォルト)
0(R): データあり

D0 RDBF: Receive Data Buffer Full Flag
オンチップデバッグモニタ用シリアルインタフェースの受信データバッファの状態を示します。
1(R): 満杯
0(R): データなし(デフォルト)

0xFFFFC2: Serial Transmit/Receive Data Register (SDR)

Register name	Address	Bit	Name	Function	Setting	Init.	R/W	Remarks
Serial transmit/ receive data register for debugging	FFFFC2 (B)	D7	TXRXD7	Transmit/receive data	0x0–0xFF	0x0	R/W	
		I	I					
		I	I					
		D0	TXRXD0					

D[7:0] TXRXD[7:0]: Transmit/Receive Data
オンチップデバッグモニタ用シリアルインタフェースの送信データを書き込みます。また、受信時は受信データが格納されます。（デフォルト: 0x00）

7 命令の詳細説明

本章では全命令をアルファベット順に解説します。

命令説明中の記号

レジスタ/レジスタデータ

%*rd*, *rd*:	デスティネーションとなる汎用レジスタ(R0~R7)、またはレジスタの内容
%*rs*, *rs*:	ソースとなる汎用レジスタ(R0~R7)、またはレジスタの内容
%*rb*, *rb*:	レジスタ間接アドレッシングのベースレジスタを保持している汎用レジスタ(R0~R7)、またはレジスタの内容
%sp, sp:	スタックポインタ(SP)またはその内容
%pc, pc:	プログラムカウンタ(PC)またはその内容

コード中のレジスタフィールド(*rd*, *rs*)には、汎用レジスタの番号が入ります。
R0 = 0b000, R1 = 0b001 . . . R7 = 0b111

メモリ/アドレス/メモリデータ

[%*rb*], [%sp]:	レジスタ間接アドレッシング指定
[%*rb*]+, [%sp]+:	ポストインクリメント付きレジスタ間接アドレッシング指定
[%*rb*]-, [%sp]-:	ポストデクリメント付きレジスタ間接アドレッシング指定
-[%*rb*], -[%sp]:	プリデクリメント付きレジスタ間接アドレッシング指定
[%sp+*immX*]:	ディスプレースメント付きレジスタ間接アドレッシング指定
[*imm7*]:	即値によるメモリアドレス指定
B[*XXX*]:	*XXX*で指定されるアドレス、またはそのアドレスにストアされているバイトデータ
W[*XXX*]:	*XXX*で指定される16ビット境界アドレス、またはそのアドレスにストアされているワードデータ
A[*XXX*]:	*XXX*で指定される32ビット境界アドレス、またはそのアドレスにストアされている24ビットデータもしくは32ビットデータ

即値

immX:	符号なし*X*ビット即値(*X*は即値のビット長を示す数値)
signX:	符号付き*X*ビット即値(*X*は即値のビット長を示す数値。最上位ビットは符号ビット)

ビットフィールド

(*X*):	データのビット*X*
(*X*:*Y*):	ビット*X*からビット*Y*までのビットフィールド
{*X*, *Y*…}:	ビット構成

コード

rd, *rs*, *rb*:	レジスタ番号(R0 = 0 ... R7 = 7)
d:	ディレイドビット(0: 基本分岐命令, 1: ディレイド分岐命令)

機能

←:	右側の内容が左側の項目にロード/設定されることを示します。
+:	加算
-:	減算
&:	論理積
|:	論理和
^:	排他的論理和
!:	論理否定

フラグ

IL:	割り込みレベル
IE:	割り込みイネーブルフラグ
C:	キャリーフラグ
V:	オーバーフローフラグ
Z:	ゼロフラグ
N:	ネガティブフラグ
–:	変更なし
↔:	セット(1)またはリセット(0)
1:	セット(1)
0:	リセット(0)

adc %rd, %rs
adc/c %rd, %rs
adc/nc %rd, %rs

機 能

16ビットキャリー付き加算

標準） *rd*(15:0) ← *rd*(15:0) + *rs*(15:0) + C, *rd*(23:16) ← 0
拡張1) *rd*(15:0) ← *rs*(15:0) + *imm13*(ゼロ拡張) + C, *rd*(23:16) ← 0
拡張2) *rd*(15:0) ← *rs*(15:0) + *imm16* + C, *rd*(23:16) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	1	1	0	*rd*			1	0	0	1	*rs*			adc
0	0	1	1	1	0	*rd*			0	0	0	1	*rs*			adc/c
0	0	1	1	1	0	*rd*			0	1	0	1	*rs*			adc/nc

フラグ

IL	IE	C	V	Z	N	
–	–	↔	↔	↔	↔	adc
–	–	–	↔	↔	↔	adc/c, adc/nc

モード

Src:レジスタ直接 `%rs` = `%r0`～`%r7`
Dst:レジスタ直接 `%rd` = `%r0`～`%r7`

CLK

1サイクル

説 明

(1)標準

```
adc   %rd,%rs          ; rd ← rd + rs + C
```

*rs*レジスタの内容とC(キャリー)フラグの内容を*rd*レジスタに加えます。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。

(2)拡張1

```
ext   imm13
adc   %rd,%rs          ; rd ← rs + imm13 + C
```

*rs*レジスタの内容に13ビット即値*imm13*とC(キャリー)フラグの内容をゼロ拡張して加え、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。*rs*レジスタの内容は変更されません。

(3)拡張2

```
ext   imm3             ; imm3(2:0) = imm16(15:13)
ext   imm13            ; = imm16(12:0)
adc   %rd,%rs          ; rd ← rs + imm16 + C
```

*rs*レジスタの内容に16ビット即値*imm16*とC(キャリー)フラグの内容を加え、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。*rs*レジスタの内容は変更されません。

(4)条件実行

オペコードに`/c`または`/nc`を付けることにより、条件実行命令になります。
`adc/c`　Cフラグが1の場合に実行、0の場合は`nop`
`adc/nc`　Cフラグが0の場合に実行、1の場合は`nop`

この場合も、`ext`命令による拡張が可能です。

(5)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

(1)
```
adc   %r0,%r1          ; r0 = r0 + r1 + C
```

(2)32ビットデータの加算

データ1 = {r2, r1}, データ2 = {r4, r3}, 結果 = {r2, r1}

```
add   %r1,%r3          ; 下位ワードの加算
adc   %r2,%r4          ; 上位ワードの加算
```

adc *%rd, imm7*

機　能

16ビットキャリー付き加算

標準）*rd*(15:0) ← *rd*(15:0) + *imm7*(ゼロ拡張) + C, *rd*(23:16) ← 0

拡張1）*rd*(15:0) ← *rd*(15:0) + *imm16* + C, *rd*(23:16) ← 0

拡張2）不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	1	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	↔	↔	↔	↔

モード

Src:即値（符号なし）

Dst:レジスタ直接　`%rd` = `%r0`〜`%r7`

CLK

1サイクル

説　明

（1）標準

```
adc   %rd,imm7          ; rd ← rd + imm7 + C
```

7ビット即値*imm7*とC（キャリー）フラグの内容をゼロ拡張して*rd*レジスタに加えます。演算は16ビットで行われ、*rd*レジスタのビット23〜16は0に設定されます。

（2）拡張1

```
ext   imm9              ; imm9(8:0) = imm16(15:7)
adc   %rd,imm7          ; rd ← rd + imm16 + C, imm7 = imm16(6:0)
```

16ビット即値*imm16*とC（キャリー）フラグの内容を*rd*レジスタに加えます。演算は16ビットで行われ、*rd*レジスタのビット23〜16は0に設定されます。

（3）ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

（1）
```
adc   %r0,0x7f          ; r0 = r0 + 0x7f + C
```

（2）
```
ext   0x1ff
adc   %r1,0x7f          ; r1 = r1 + 0xffff + C
```

add %rd, %rs
add/c %rd, %rs
add/nc %rd, %rs

機 能

16ビット加算

標準） *rd*(15:0) ← *rd*(15:0) + *rs*(15:0), *rd*(23:16) ← 0
拡張1）*rd*(15:0) ← *rs*(15:0) + *imm13*(ゼロ拡張), *rd*(23:16) ← 0
拡張2）*rd*(15:0) ← *rs*(15:0) + *imm16*, *rd*(23:16) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	1	1	0	*rd*			1	0	0	0	*rs*			add
0	0	1	1	1	0	*rd*			0	0	0	0	*rs*			add/c
0	0	1	1	1	0	*rd*			0	1	0	0	*rs*			add/nc

フラグ

IL	IE	C	V	Z	N	
–	–	↔	↔	↔	↔	add
–	–	–	↔	↔	↔	add/c, add/nc

モード

Src: レジスタ直接 `%rs` = `%r0`〜`%r7`
Dst: レジスタ直接 `%rd` = `%r0`〜`%r7`

CLK

1サイクル

説 明

(1)標準

```
add   %rd,%rs          ; rd ← rd + rs
```

*rs*レジスタの内容を*rd*レジスタに加えます。演算は16ビットで行われ、*rd*レジスタのビット23〜16は0に設定されます。

(2)拡張1

```
ext   imm13
add   %rd,%rs          ; rd ← rs + imm13
```

*rs*レジスタの内容に13ビット即値*imm13*の内容をゼロ拡張して加え、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23〜16は0に設定されます。*rs*レジスタの内容は変更されません。

(3)拡張2

```
ext   imm3             ; imm3(2:0) = imm16(15:13)
ext   imm13            ; = imm16(12:0)
add   %rd,%rs          ; rd ← rs + imm16
```

*rs*レジスタの内容に16ビット即値*imm16*の内容を加え、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23〜16は0に設定されます。*rs*レジスタの内容は変更されません。

(4)条件実行

オペコードに`/c`または`/nc`を付けることにより、条件実行命令になります。
`add/c` Cフラグが1の場合に実行、0の場合は`nop`
`add/nc` Cフラグが0の場合に実行、1の場合は`nop`

この場合も、`ext`命令による拡張が可能です。

(5)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

(1)
```
add   %r0,%r0          ; r0 = r0 + r0
```

(2)
```
ext   0x1
ext   0x1fff
add   %r1,%r2          ; r1 = r2 + 0x3fff
```

add *%rd, imm7*

機　能

16ビット加算

標準） *rd*(15:0) ← *rd*(15:0) + *imm7*(ゼロ拡張), *rd*(23:16) ← 0

拡張1）*rd*(15:0) ← *rd*(15:0) + *imm16*, *rd*(23:16) ← 0

拡張2）不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	↔	↔	↔	↔

モード

Src:即値（符号なし）

Dst:レジスタ直接 `%rd` = `%r0`〜`%r7`

CLK

1サイクル

説　明

(1)標準

```
add   %rd,imm7          ; rd ← rd + imm7
```

7ビット即値*imm7*をゼロ拡張して*rd*レジスタに加えます。演算は16ビットで行われ、*rd*レジスタのビット23〜16は0に設定されます。

(2)拡張1

```
ext   imm9              ; imm9(8:0) = imm16(15:7)
add   %rd,imm7          ; rd ← rd + imm16, imm7 = imm16(6:0)
```

16ビット即値*imm16*を*rd*レジスタに加えます。演算は16ビットで行われ、*rd*レジスタのビット23〜16は0に設定されます。

(3)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

(1)
```
add   %r0,0x3f          ; r0 = r0 + 0x3f
```

(2)
```
ext   0x1ff
add   %r1,0x7f          ; r1 = r1 + 0xffff
```

add.a %rd, %rs
add.a/c %rd, %rs
add.a/nc %rd, %rs

機 能

24ビット加算

標準） *rd*(23:0) ← *rd*(23:0) + *rs*(23:0)
拡張1）*rd*(23:0) ← *rs*(23:0) + *imm13*(ゼロ拡張)
拡張2）*rd*(23:0) ← *rs*(23:0) + *imm24*

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	1	0	0	*rd*			1	0	0	0	*rs*			add.a
0	0	1	1	0	0	*rd*			0	0	0	0	*rs*			add.a/c
0	0	1	1	0	0	*rd*			0	1	0	0	*rs*			add.a/nc

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接 `%rs` = `%r0`～`%r7`
Dst: レジスタ直接 `%rd` = `%r0`～`%r7`

CLK

1サイクル

説 明

(1)標準

```
add.a   %rd,%rs       ; rd ← rd + rs
```

*rs*レジスタの内容を*rd*レジスタに加えます。

(2)拡張1

```
ext     imm13
add.a   %rd,%rs       ; rd ← rs + imm13
```

*rs*レジスタの内容に13ビット即値*imm13*の内容をゼロ拡張して加え、結果を*rd*レジスタにロードします。*rs*レジスタの内容は変更されません。

(3)拡張2

```
ext     imm11         ; imm11(10:0) = imm24(23:13)
ext     imm13         ; = imm24(12:0)
add.a   %rd,%rs       ; rd ← rs + imm24
```

*rs*レジスタの内容に24ビット即値*imm24*の内容を加え、結果を*rd*レジスタにロードします。*rs*レジスタの内容は変更されません。

(4)条件実行

オペコードに/cまたは/ncを付けることにより、条件実行命令になります。
`add.a/c` Cフラグが1の場合に実行、0の場合はnop
`add.a/nc` Cフラグが0の場合に実行、1の場合はnop

この場合も、`ext`命令による拡張が可能です。

(5)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

(1)
```
add.a   %r0,%r0       ; r0 = r0 + r0
```

(2)
```
ext     0x7ff
ext     0x1fff
add.a   %r1,%r2       ; r1 = r2 + 0xffffff
```

add.a *%rd, imm7*

機　能

24ビット加算

標準） *rd*(23:0) ← *rd*(23:0) + *imm7*(ゼロ拡張)
拡張1）*rd*(23:0) ← *rd*(23:0) + *imm20*(ゼロ拡張)
拡張2）*rd*(23:0) ← *rd*(23:0) + *imm24*

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src:即値（符号なし）
Dst:レジスタ直接 `%rd` = `%r0`〜`%r7`

CLK

1サイクル

説　明

(1)標準

```
add.a   %rd,imm7        ; rd ← rd + imm7
```

7ビット即値*imm7*をゼロ拡張して*rd*レジスタに加えます。

(2)拡張1

```
ext     imm13           ; = imm20(19:7)
add.a   %rd,imm7        ; rd ← rd + imm20, imm7 = imm20(6:0)
```

20ビット即値*imm20*をゼロ拡張して*rd*レジスタに加えます。

(3)拡張2

```
ext     imm4            ; imm4(3:0) = imm24(23:20)
ext     imm13           ; = imm24(19:7)
add.a   %rd,imm7        ; rd ← rd + imm24, imm7 = imm24(6:0)
```

24ビット即値*imm24*を*rd*レジスタに加えます。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

(1)
```
add.a   %r0,0x7f        ; r0 = r0 + 0x7f
```

(2)
```
ext     0xf
ext     0x1fff
add.a   %r1,0x7f        ; r1 = r1 + 0xffffff
```

add.a %sp, *%rs*

機　能

24ビット加算

標準） sp(23:0) ← sp(23:0) + *rs*(23:0)
拡張1) sp(23:0) ← *rs*(23:0) + *imm13*(ゼロ拡張)
拡張2) sp(23:0) ← *rs*(23:0) + *imm24*

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	0	0	0	0	0	0	0	1	*rs*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接 `%rs` = `%r0`〜`%r7`
Dst: レジスタ直接 `%sp`

CLK

1サイクル

説　明

(1)標準

```
add.a   %sp,%rs         ; sp ← sp + rs
```

*rs*レジスタの内容をスタックポインタSPに加えます。

(2)拡張1

```
ext     imm13
add.a   %sp,%rs         ; sp ← rs + imm13
```

*rs*レジスタの内容に13ビット即値*imm13*の内容をゼロ拡張して加え、結果をスタックポインタSPにロードします。*rs*レジスタの内容は変更されません。

(3)拡張2

```
ext     imm11           ; imm11(10:0) = imm24(23:13)
ext     imm13           ; = imm24(12:0)
add.a   %sp,%rs         ; sp ← rs + imm24
```

*rs*レジスタの内容に24ビット即値*imm24*の内容を加え、結果をスタックポインタSPにロードします。*rs*レジスタの内容は変更されません。

(4)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

(1)
```
add.a   %sp,%r0         ; sp = sp + r0
```

(2)
```
ext     0x1
ext     0x1ffc
add.a   %sp,%r2         ; sp = r2 + 0x3ffc
```

注　意

加算結果の下位2ビットは常に0としてSPにロードされます。

add.a %sp, *imm7*

機 能

24ビット加算

標準） sp(23:0) ← sp(23:0) + *imm7*(ゼロ拡張)
拡張1）sp(23:0) ← sp(23:0) + *imm20*(ゼロ拡張)
拡張2）sp(23:0) ← sp(23:0) + *imm24*

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	0	0	0	*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src:即値（符号なし）
Dst:レジスタ直接 `%sp`

CLK

1サイクル

説 明

(1)標準

```
add.a   %sp,imm7        ; sp ← sp + imm7
```

7ビット即値*imm7*をゼロ拡張してスタックポインタSPに加えます。

(2)拡張1

```
ext     imm13           ; = imm20(19:7)
add.a   %sp,imm7        ; sp ← sp + imm20, imm7 = imm20(6:0)
```

20ビット即値*imm20*をゼロ拡張してスタックポインタSPに加えます。

(3)拡張2

```
ext     imm4            ; imm4(3:0) = imm24(23:20)
ext     imm13           ; = imm24(19:7)
add.a   %sp,imm7        ; sp ← sp + imm24, imm7 = imm24(6:0)
```

24ビット即値*imm24*をスタックポインタSPに加えます。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

(1)
```
add.a   %sp,0x7c        ; sp = sp + 0x7c
```

(2)
```
ext     0x1fff
add.a   %sp,0x7c        ; sp = sp + 0xffffc
```

注 意

加算結果の下位2ビットは常に0としてSPにロードされます。

and %rd, %rs
and/c %rd, %rs
and/nc %rd, %rs

機 能

16ビット論理積

標準） *rd*(15:0) ← *rd*(15:0) & *rs*(15:0), *rd*(23:16) ← 0
拡張1）*rd*(15:0) ← *rs*(15:0) & *imm13*(ゼロ拡張), *rd*(23:16) ← 0
拡張2）*rd*(15:0) ← *rs*(15:0) & *imm16*, *rd*(23:16) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	1	1	*r d*			1	0	0	0	*r s*			and
0	0	1	0	1	1	*r d*			0	0	0	0	*r s*			and/c
0	0	1	0	1	1	*r d*			0	1	0	0	*r s*			and/nc

フラグ

IL	IE	C	V	Z	N
–	–	–	0	↔	↔

モード

Src: レジスタ直接 `%rs` = `%r0`〜`%r7`
Dst: レジスタ直接 `%rd` = `%r0`〜`%r7`

CLK

1サイクル

説 明

(1)標準

```
and   %rd,%rs          ; rd ← rd & rs
```

*rs*レジスタの内容と*rd*レジスタの内容の論理積をとり、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23〜16は0に設定されます。

(2)拡張1

```
ext   imm13
and   %rd,%rs          ; rd ← rs & imm13
```

*rs*レジスタの内容とゼロ拡張した13ビット即値*imm13*の論理積をとり、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23〜16は0に設定されます。*rs*レジスタの内容は変更されません。

(3)拡張2

```
ext   imm3             ; imm3(2:0) = imm16(15:13)
ext   imm13            ; = imm16(12:0)
and   %rd,%rs          ; rd ← rs & imm16
```

*rs*レジスタの内容と16ビット即値*imm16*の論理積をとり、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23〜16は0に設定されます。*rs*レジスタの内容は変更されません。

(4)条件実行

オペコードに`/c`または`/nc`を付けることにより、条件実行命令になります。

```
and/c    Cフラグが1の場合に実行、0の場合はnop
and/nc   Cフラグが0の場合に実行、1の場合はnop
```

この場合も、`ext`命令による拡張が可能です。

(5)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

(1)
```
and   %r0,%r0          ; r0 = r0 & r0
```

(2)
```
ext   0x1
ext   0x1fff
and   %r1,%r2          ; r1 = r2 & 0x3fff
```

and *%rd, sign7*

機　能

16ビット論理積

標準） *rd*(15:0) ← *rd*(15:0) & *sign7*(符号拡張), *rd*(23:16) ← 0
拡張1）*rd*(15:0) ← *rd*(15:0) & *sign16*, *rd*(23:16) ← 0
拡張2）不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	*r d*			*sign7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	0	↔	↔

モード

Src:即値（符号付き）
Dst:レジスタ直接 `%rd` = `%r0`〜`%r7`

CLK

1サイクル

説　明

(1)標準

```
and   %rd,sign7        ; rd ← rd & sign7
```

*rd*レジスタの内容と符号拡張した7ビット即値*sign7*の論理積をとり、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23〜16は0に設定されます。

(2)拡張1

```
ext   imm9             ; imm9(8:0) = sign16(15:7)
and   %rd,sign7        ; rd ← rd & sign16, sign7 = sign16(6:0)
```

*rd*レジスタの内容と16ビット即値*sign16*の論理積をとり、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23〜16は0に設定されます。

(3)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

(1)
```
and   %r0,0x7e         ; r0 = r0 & 0xfffe
```

(2)
```
ext   0x3f
and   %r1,0x7f         ; r1 = r1 & 0x1fff
```

brk

機　能

デバッグ割り込み

標準） A[DBRAM] ← {psr, pc + 2}, A[DBRAM + 0x4] ← r0, pc ← 0xfffc00
拡張1）不可
拡張2）不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0

フラグ

IL	IE	C	V	Z	N
–	0	–	–	–	–

モード

–

CLK

4サイクル

説　明

デバッグ処理ルーチンを呼び出します。
次の命令のアドレス（PC + 2）とPSRの内容、および汎用レジスタR0の内容をデバッグ用ワークエリア（DBRAM）にセーブ後、ミニモニタの開始アドレス（0xfffc00）をPCにロードします。これによりデバッグ処理ルーチンに分岐します。また、プロセッサはデバッグモードに移行します。
デバッグ処理ルーチンよりのリターンには`retd`命令を使用します。
本命令はデバッグファームウェア用です。ユーザプログラム内では使用しません。

例

```
brk                     ; デバッグ処理ルーチンを実行
```

call %rb
call.d %rb

機 能

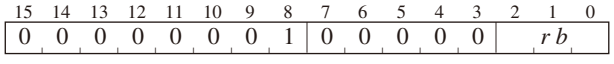
PC相対サブルーチンコール

標準） `call`: sp ← sp - 4, A[sp] ← pc + 2, pc ← pc + 2 + *rb*
`call.d`: sp ← sp - 4, A[sp] ← pc + 4, pc ← pc + 2 + *rb*

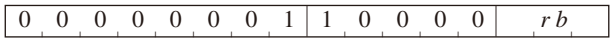
拡張1）不可

拡張2）不可

コード



`call`



`call.d`

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

レジスタ直接 `%rb` = `%r0`〜`%r7`

CLK

`call` 4サイクル
`call.d` 3サイクル（ディレイドスロット命令 = 1サイクルの場合）、4サイクル（その他）

説 明

(1)標準

`call %rb`

次の命令のアドレスをスタックにセーブ後、PC（PC + 2）に*rb*レジスタの内容を加算し、そのアドレスから始まるサブルーチンをコールします。*rb*レジスタの最下位ビットは無効となり、常に0として扱われます。サブルーチンで`ret`命令を実行すると、`call`の次の命令にリターンします。

(2)ディレイド分岐（dビット（ビット7）= 1）

`call.d %rb`

`call.d %rb`命令では命令コード中のdビット（ビット7）がセットされ、次の命令がディレイドスロット命令となります。ディレイドスロット命令はサブルーチンへの分岐前に実行されます。このため、スタックにセーブされるリターンアドレスは、ディレイドスロット命令の次の命令のアドレス（PC + 4）となります。`call.d`命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
call   %r0      ; pc + 2 + r0番地から始まるサブルーチンをコール
```

注 意

`call.d`命令（ディレイド分岐）を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

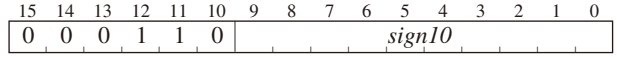
call *sign10*
call.d *sign10*

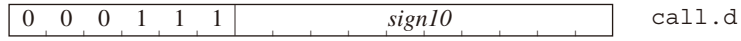
機 能

PC相対サブルーチンコール

標準）`call`: sp ← sp - 4, A[sp] ← pc + 2, pc ← pc + 2 + *sign10* × 2
`call.d`: sp ← sp - 4, A[sp] ← pc + 4, pc ← pc + 2 + *sign10* × 2
拡張1）`call`: sp ← sp - 4, A[sp] ← pc + 2, pc ← pc + 2 + *sign24*
`call.d`: sp ← sp - 4, A[sp] ← pc + 4, pc ← pc + 2 + *sign24*
拡張2）不可

コード





フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

符号付きPC相対

CLK

`call` 4サイクル
`call.d` 3サイクル(ディレイドスロット命令 = 1サイクルの場合)、4サイクル(その他)

説 明

(1)標準

```
call  sign10  ; = "call sign11"
              ; sign10 = sign11(10:1), sign11(0) = 0
```

次の命令のアドレスをスタックにセーブ後、PC(PC + 2)に符号付き10ビット即値*sign10*を2倍して加算し、そのアドレスから始まるサブルーチンをコールします。*sign10*は16ビット単位のワードアドレスを指定します。サブルーチンで`ret`命令を実行すると、`call`の次の命令にリターンします。*sign10*(×2)による分岐可能範囲はPC - 1,022～PC + 1,024です。

(2)拡張1

```
ext   imm13   ; = sign24(23:11)
call  sign10  ; = "call sign24"
              ; sign10 = sign24(10:1), sign24(0) = 0
```

`ext`命令の13ビット即値*imm13*により、PCに加算するディスプレースメントが符号付き24ビットとなります。*sign24*による分岐可能範囲はPC - 8,388,606～PC + 8,388,608です。

(3)ディレイド分岐(dビット(ビット10) = 1)

```
call.d  sign10
```

`call.d` *sign10*命令では命令コード中のdビット(ビット10)がセットされ、次の命令がディレイドスロット命令となります。ディレイドスロット命令はサブルーチンへの分岐前に実行されます。このため、スタックにセーブされるリターンアドレスは、ディレイドスロット命令の次の命令のアドレス(PC + 4)となります。`call.d`命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
ext   0x1fff
call  0x0         ; pc + 2 - 0x800番地から始まるサブルーチンをコール
```

注 意

`call.d`命令(ディレイド分岐)を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

calla %rb
calla.d %rb

機 能

PC絶対サブルーチンコール

標準） `calla`: sp ← sp - 4, A[sp] ← pc + 2, pc ← *rb*
`calla.d`: sp ← sp - 4, A[sp] ← pc + 4, pc ← *rb*

拡張1）不可

拡張2）不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	1	0	0	0	0	1	*r*	*b*		`calla`
0	0	0	0	0	0	0	1	1	0	0	0	1	*r*	*b*		`calla.d`

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

PC絶対

CLK

`calla` 4サイクル
`calla.d` 3サイクル（ディレイドスロット命令 = 1サイクルの場合）、4サイクル（その他）

説 明

（1）標準

```
calla  %rb
```

次の命令のアドレスをスタックにセーブ後、*rb*レジスタの内容をPCにロードして、そのアドレスから始まるサブルーチンをコールします。*rb*レジスタの最下位ビットは無視され、常に0として扱われます。サブルーチンで`ret`命令を実行すると、`calla`の次の命令にリターンします。

（2）ディレイド分岐（dビット（ビット7）= 1）

```
calla.d  %rb
```

`calla.d`命令では命令コード中のdビット（ビット7）がセットされ、次の命令がディレイドスロット命令となります。ディレイドスロット命令はサブルーチンへの分岐前に実行されます。このため、スタックにセーブされるリターンアドレスは、ディレイドスロット命令の次の命令のアドレス（PC + 4）となります。`calla.d`命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
calla  %r0        ; r0レジスタの内容を先頭アドレスとするサブルーチンをコール
```

注 意

`calla.d`命令（ディレイド分岐）を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

calla *imm7*
calla.d *imm7*

機 能

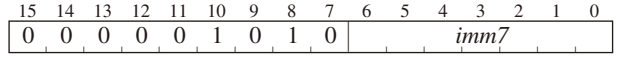
PC絶対サブルーチンコール

標準）`calla`: sp ← sp - 4, A[sp] ← pc + 2, pc ← *imm7*
`calla.d`: sp ← sp - 4, A[sp] ← pc + 4, pc ← *imm7*

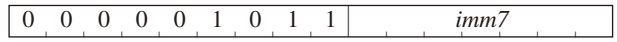
拡張1）`calla`: sp ← sp - 4, A[sp] ← pc + 2, pc ← *imm20*
`calla.d`: sp ← sp - 4, A[sp] ← pc + 4, pc ← *imm20*

拡張2）`calla`: sp ← sp - 4, A[sp] ← pc + 2, pc ← *imm24*
`calla.d`: sp ← sp - 4, A[sp] ← pc + 4, pc ← *imm24*

コード



`calla`



`calla.d`

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

PC絶対

CLK

`calla` 4サイクル
`calla.d` 3サイクル(ディレイドスロット命令 = 1サイクルの場合)、4サイクル(その他)

説 明

(1)標準

```
calla  imm7
```

次の命令のアドレスをスタックにセーブ後、7ビット即値*imm7*をPCにロードして、そのアドレスから始まるサブルーチンをコールします。*imm7*の最下位ビットは無視され、常に0として扱われます。サブルーチンで`ret`命令を実行すると、`calla`の次の命令にリターンします。

(2)拡張1

```
ext    imm13   ; = imm20(19:7)
call   imm7    ; = "call imm20", imm7 = imm20(6:0)
```

`ext`命令の13ビット即値*imm13*により、PCに設定する分岐先アドレスが20ビットに拡張されます。

(3)拡張2

```
ext    imm4    ; imm4(3:0) = imm24(23:20)
ext    imm13   ; = imm24(19:7)
call   imm7    ; = "call imm24", imm7 = imm24(6:0)
```

PCに設定する分岐先アドレスが24ビットに拡張されます。

(4)ディレイド分岐(dビット(ビット7) = 1)

```
calla.d  imm7
```

`calla.d`命令では命令コード中のdビット(ビット7)がセットされ、次の命令がディレイドスロット命令となります。ディレイドスロット命令はサブルーチンへの分岐前に実行されます。このため、スタックにセーブされるリターンアドレスは、ディレイドスロット命令の次の命令のアドレス(PC + 4)となります。`calla.d`命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
ext     0x1fff
calla   0x0       ; アドレス0xfff80から始まるサブルーチンをコール
```

注 意

`calla.d`命令(ディレイド分岐)を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

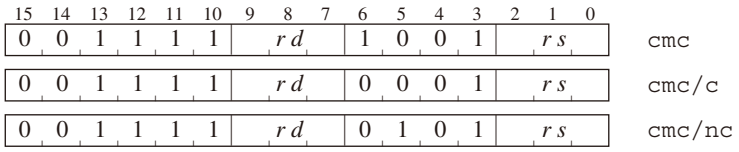
cmc %rd, %rs
cmc/c %rd, %rs
cmc/nc %rd, %rs

機 能

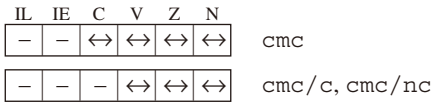
キャリー付き16ビット比較

標準） *rd*(15:0) - *rs*(15:0) - C
拡張1）*rs*(15:0) - *imm13*(ゼロ拡張) - C
拡張2）*rs*(15:0) - *imm16* - C

コード



フラグ



モード

Src: レジスタ直接 `%rs` = `%r0`～`%r7`
Dst: レジスタ直接 `%rd` = `%r0`～`%r7`

CLK

1サイクル

説 明

(1)標準

```
cmc   %rd,%rs          ; rd - rs - C
```

*rd*レジスタの内容から*rs*レジスタとC(キャリー)フラグの内容を減算し、結果によりフラグ(C、V、Z、N)をセット/リセットします。演算は16ビットで行われます。*rd*レジスタは変更されません。

(2)拡張1

```
ext   imm13
cmc   %rd,%rs          ; rs - imm13 - C
```

*rs*レジスタの内容からゼロ拡張した13ビット即値*imm13*とC(キャリー)フラグの内容を減算し、結果によりフラグ(C、V、Z、N)をセット/リセットします。演算は16ビットで行われます。*rd*および*rs*レジスタは変更されません。

※ この組み合わせのとき、*rd*の値は演算に使用されません。

(3)拡張2

```
ext   imm3             ; imm3(2:0) = imm16(15:13)
ext   imm13            ; = imm16(12:0)
cmc   %rd,%rs          ; rs - imm16 - C
```

*rs*レジスタの内容から16ビット即値*imm16*とC(キャリー)フラグの内容を減算し、結果によりフラグ(C、V、Z、N)をセット/リセットします。演算は16ビットで行われます。*rd*および*rs*レジスタは変更されません。

※ この組み合わせのとき、*rd*の値は演算に使用されません。

(4)条件実行

オペコードに/cまたは/ncを付けることにより、条件実行命令になります。
cmc/c　Cフラグが1の場合に実行、0の場合はnop
cmc/nc　Cフラグが0の場合に実行、1の場合はnop

この場合も、ext命令による拡張が可能です。
実行された場合は、実行結果によりフラグ(V、Z、N)をセット/リセットします。

(5)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合はext命令による拡張は行えません。

例

(1) `cmc  %r0,%r1 ; r0 - r1 - C`、結果によりフラグを変更

(2)
```
ext  0x1fff
cmc  %r1,%r2 ; r2 - 0x1fff - C
```
、結果によりフラグを変更

cmc %rd, sign7

機　能

キャリー付き16ビット比較

標準） *rd*(15:0) - *sign7*(符号拡張) - C

拡張1）*rd*(15:0) - *sign16* - C

拡張2）不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	1	*r d*			*sign7*						

フラグ

IL	IE	C	V	Z	N
–	–	↔	↔	↔	↔

モード

Src:即値（符号付き）

Dst:レジスタ直接 `%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準

```
cmc   %rd,sign7        ; rd - sign7 - C
```

*rd*レジスタの内容から符号拡張した7ビット即値*sign7*とC（キャリー）フラグの内容を減算し、結果によりフラグ（C、V、Z、N）をセット/リセットします。演算は16ビットで行われます。*rd*レジスタは変更されません。

(2)拡張1

```
ext   imm9             ; imm9(8:0) = sign16(15:7)
cmc   %rd,sign7        ; rd - sign16 - C, sign7 = sign16(6:0)
```

*rd*レジスタの内容から16ビット即値*sign16*とC（キャリー）フラグの内容を減算し、結果によりフラグ（C、V、Z、N）をセット/リセットします。演算は16ビットで行われます。*rd*レジスタは変更されません。

(3)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

(1) `cmc  %r0,0x7f ; r0 - 0x7f - C`、結果によりフラグを変更

(2)
```
ext  0x1ff
cmc  %r1,0x7f ; r1 - 0xffff - C、結果によりフラグを変更
```

cmp %rd, %rs
cmp/c %rd, %rs
cmp/nc %rd, %rs

機 能

16ビット比較

標準） *rd*(15:0) - *rs*(15:0)
拡張1） *rs*(15:0) - *imm13*(ゼロ拡張)
拡張2） *rs*(15:0) - *imm16*

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	1	1	1	*r d*			1	0	0	0	*r s*			cmp
0	0	1	1	1	1	*r d*			0	0	0	0	*r s*			cmp/c
0	0	1	1	1	1	*r d*			0	1	0	0	*r s*			cmp/nc

フラグ

IL	IE	C	V	Z	N	
–	–	↔	↔	↔	↔	cmp
–	–	–	↔	↔	↔	cmp/c, cmp/nc

モード

Src: レジスタ直接 `%rs` = `%r0`～`%r7`
Dst: レジスタ直接 `%rd` = `%r0`～`%r7`

CLK

1サイクル

説 明

(1)標準

```
cmp   %rd,%rs          ; rd - rs
```

*rd*レジスタの内容から*rs*レジスタの内容を減算し、結果によりフラグ(C、V、Z、N)をセット/リセットします。演算は16ビットで行われます。*rd*レジスタは変更されません。

(2)拡張1

```
ext   imm13
cmp   %rd,%rs          ; rs - imm13
```

*rs*レジスタの内容からゼロ拡張した13ビット即値*imm13*を減算し、結果によりフラグ(C、V、Z、N)をセット/リセットします。演算は16ビットで行われます。*rd*および*rs*レジスタは変更されません。

※ この組み合わせのとき、*rd*の値は演算に使用されません。

(3)拡張2

```
ext   imm3             ; imm3(2:0) = imm16(15:13)
ext   imm13            ; = imm16(12:0)
cmp   %rd,%rs          ; rs - imm16
```

*rs*レジスタの内容から16ビット即値*imm16*を減算し、結果によりフラグ(C、V、Z、N)をセット/リセットします。演算は16ビットで行われます。*rd*および*rs*レジスタは変更されません。

※ この組み合わせのとき、*rd*の値は演算に使用されません。

(4)条件実行

オペコードに/cまたは/ncを付けることにより、条件実行命令になります。
cmp/c　Cフラグが1の場合に実行、0の場合はnop
cmp/nc　Cフラグが0の場合に実行、1の場合はnop

この場合も、ext命令による拡張が可能です。
実行された場合は、実行結果によりフラグ(V、Z、N)をセット/リセットします。

(5)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合はext命令による拡張は行えません。

例

(1) `cmp  %r0,%r1  ; r0 - r1、結果によりフラグを変更`

(2)
```
ext  0x1
ext  0x1fff
cmp  %r1,%r2  ; r2 - 0x3fff、結果によりフラグを変更
```

cmp *%rd, sign7*

機　能

16ビット比較

標準） *rd*(15:0) - *sign7*(符号拡張)

拡張1) *rd*(15:0) - *sign16*

拡張2) 不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	0	*r d*			*sign7*						

フラグ

IL	IE	C	V	Z	N
–	–	↔	↔	↔	↔

モード

Src:即値(符号付き)

Dst:レジスタ直接 `%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準

```
cmp   %rd,sign7       ; rd - sign7
```

*rd*レジスタの内容から符号拡張した7ビット即値*sign7*を減算し、結果によりフラグ(C、V、Z、N)をセット/リセットします。演算は16ビットで行われます。*rd*レジスタは変更されません。

(2)拡張1

```
ext   imm9            ; imm9(8:0) = sign16(15:7)
cmp   %rd,sign7       ; rd - sign16, sign7 = sign16(6:0)
```

*rd*レジスタの内容から16ビット即値*sign16*を減算し、結果によりフラグ(C、V、Z、N)をセット/リセットします。演算は16ビットで行われます。*rd*レジスタは変更されません。

(3)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

(1) `cmp  %r0,0x3f ; r0 - 0x3f`、結果によりフラグを変更

(2)
```
ext   0x1ff
cmp   %r1,0x7f ; r1 - 0xffff、結果によりフラグを変更
```

cmp.a %rd, %rs
cmp.a/c %rd, %rs
cmp.a/nc %rd, %rs

機　能

24ビット比較

標準） *rd*(23:0) - *rs*(23:0)
拡張1）*rs*(23:0) - *imm13*(ゼロ拡張)
拡張2）*rs*(23:0) - *imm24*

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	1	0	1	*rd*			1	0	0	0	*rs*			cmp.a
0	0	1	1	0	1	*rd*			0	0	0	0	*rs*			cmp.a/c
0	0	1	1	0	1	*rd*			0	1	0	0	*rs*			cmp.a/nc

フラグ

IL	IE	C	V	Z	N	
–	–	↔	–	↔	–	cmp.a
–	–	–	–	↔	–	cmp.a/c, cmp.a/nc

モード

Src:レジスタ直接 `%rs` = `%r0`〜`%r7`
Dst:レジスタ直接 `%rd` = `%r0`〜`%r7`

CLK

1サイクル

説　明

(1)標準

```
cmp.a   %rd,%rs        ; rd - rs
```

*rd*レジスタの内容から*rs*レジスタの内容を減算し、結果によりフラグ(C、Z)をセット/リセットします。*rd*レジスタは変更されません。

(2)拡張1

```
ext     imm13
cmp.a   %rd,%rs        ; rs - imm13
```

*rs*レジスタの内容からゼロ拡張した13ビット即値*imm13*を減算し、結果によりフラグ(C、Z)をセット/リセットします。*rd*および*rs*レジスタは変更されません。

※ この組み合わせのとき、*rd*の値は演算に使用されません。

(3)拡張2

```
ext     imm11          ; imm11(10:0) = imm24(23:13)
ext     imm13          ; = imm24(12:0)
cmp.a   %rd,%rs        ; rs - imm24
```

*rs*レジスタの内容から24ビット即値*imm24*を減算し、結果によりフラグ(C、Z)をセット/リセットします。*rd*および*rs*レジスタは変更されません。

※ この組み合わせのとき、*rd*の値は演算に使用されません。

(4)条件実行

オペコードに`/c`または`/nc`を付けることにより、条件実行命令になります。
`cmp.a/c`　Cフラグが1の場合に実行、0の場合はnop
`cmp.a/nc`　Cフラグが0の場合に実行、1の場合はnop

この場合も、`ext`命令による拡張が可能です。
実行された場合は、実行結果によりフラグ(Z)をセット/リセットします。

(5)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

(1)
```
cmp.a   %r0,%r1   ; r0 - r1、結果によりフラグを変更
```

(2)
```
ext     0x1
ext     0x1fff
cmp.a   %r1,%r2   ; r2 - 0x3fff、結果によりフラグを変更
```

cmp.a %rd, imm7

機　能

24ビット比較
標準） *rd*(23:0) - *imm7*(ゼロ拡張)
拡張1）*rd*(23:0) - *imm20*(ゼロ拡張)
拡張2）*rd*(23:0) - *imm24*

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	*rd*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	↔	–	↔	–

モード

Src:即値（符号なし）
Dst:レジスタ直接 `%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準

```
cmp.a   %rd,imm7        ; rd - imm7
```

*rd*レジスタの内容からゼロ拡張した7ビット即値*imm7*を減算し、結果によりフラグ(C、Z)をセット/リセットします。*rd*レジスタは変更されません。

(2)拡張1

```
ext     imm13           ; = imm20(19:7)
cmp.a   %rd,imm7        ; rd - imm20, imm7 = imm20(6:0)
```

*rd*レジスタの内容からゼロ拡張した20ビット即値*imm20*を減算し、結果によりフラグ(C、Z)をセット/リセットします。*rd*レジスタは変更されません。

(3)拡張2

```
ext     imm4            ; imm4(3:0) = imm24(23:20)
ext     imm13           ; = imm24(19:7)
cmp.a   %rd,imm7        ; rd - imm24, imm7 = imm24(6:0)
```

*rd*レジスタの内容から24ビット即値*imm24*を減算し、結果によりフラグ(C、Z)をセット/リセットします。*rd*レジスタは変更されません。

(4)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

(1)
```
cmp.a   %r0,0x7f  ; r0 - 0x7f、結果によりフラグを変更
```

(2)
```
ext     0xf
ext     0x1fff
cmp.a   %r1,0x7f  ; r1 - 0xffffff、結果によりフラグを変更
```

cv.ab *%rd, %rs*

機　能

バイト → 24ビットデータ変換

標準） *rd*(23:8) ← *rs*(7), *rd*(7:0) ← *rs*(7:0)

拡張1）不可

拡張2）不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	0	*r d*			0	1	1	1	*r s*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接 `%rs` = `%r0`～`%r7`

Dst: レジスタ直接 `%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準

*rs*レジスタの下位8ビット（バイトデータ）を24ビットに符号拡張して*rd*レジスタに転送します。

(2)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

例

r1レジスタが0x80の場合

```
cv.ab  %r0,%r1  ; r0 = 0xffff80
```

cv.al *%rd, %rs*

機　能

32ビット → 24ビットデータ変換
標準） *rd*(23:16) ← *rs*(7:0), *rd*(15:0) ← *rd*(15:0)
拡張1）不可
拡張2）不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	0	*r d*			1	1	1	1	*r s*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

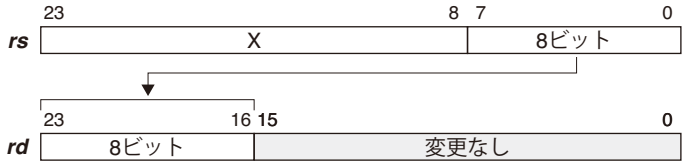
Src: レジスタ直接 `%rs` = `%r0`〜`%r7`
Dst: レジスタ直接 `%rd` = `%r0`〜`%r7`

CLK

1サイクル

説　明

（1）標準
*rs*レジスタの下位8ビットを*rd*レジスタの上位8ビットに転送します。



（2）ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

例

r1レジスタが0xff、r0レジスタが0x0の場合

```
cv.al  %r0,%r1  ; r0 = 0xff0000
```

cv.as %rd, %rs

機　能

16ビット → 24ビットデータ変換
標準） *rd*(23:16) ← *rs*(15), *rd*(15:0) ← *rs*(15:0)
拡張1）不可
拡張2）不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	0	*r d*			1	0	1	1	*r s*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

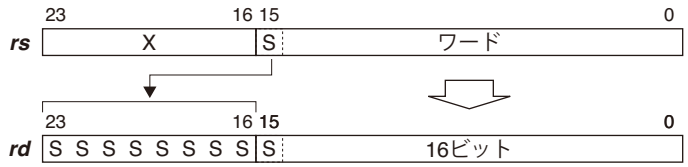
Src: レジスタ直接 `%rs` = `%r0`〜`%r7`
Dst: レジスタ直接 `%rd` = `%r0`〜`%r7`

CLK

1サイクル

説　明

(1)標準
*rs*レジスタの下位16ビットを24ビットに符号拡張して*rd*レジスタに転送します。



(2)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

例

r1レジスタが0x8000の場合

```
cv.as  %r0,%r1  ; r0 = 0xff8000
```

cv.la *%rd, %rs*

機　能

24ビット → 32ビットデータ変換
標準） *rd*(23:8) ← 0, *rd*(7:0) ← *rs*(23:16)
拡張1) 不可
拡張2) 不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	0	*rd*			0	1	1	0	*rs*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

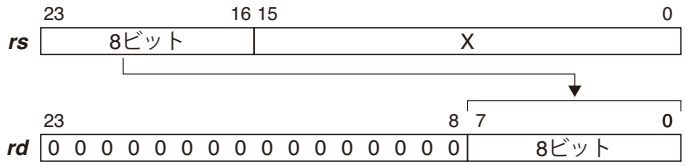
Src: レジスタ直接 `%rs` = `%r0`〜`%r7`
Dst: レジスタ直接 `%rd` = `%r0`〜`%r7`

CLK

1サイクル

説　明

(1)標準
*rs*レジスタの上位8ビットを*rd*レジスタの下位8ビットに転送します。*rd*レジスタの上位16ビットは0に設定されます。



(2)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

例

r1レジスタが0x800000の場合

```
cv.la  %r0,%r1  ; r0 = 0x000080
```

cv.ls *%rd, %rs*

機　能

16ビット → 32ビットデータ変換
標準） *rd*(23:16) ← 0, *rd*(15:0) ← *rs*(15)
拡張1）不可
拡張2）不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	0	*r d*			1	0	1	0	*r s*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

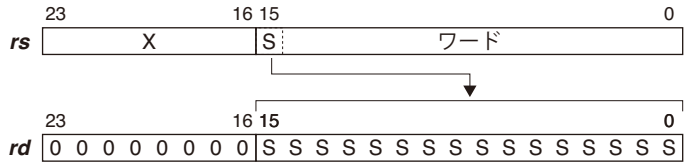
Src: レジスタ直接 `%rs` = `%r0`〜`%r7`
Dst: レジスタ直接 `%rd` = `%r0`〜`%r7`

CLK

1サイクル

説　明

(1)標準
*rs*レジスタのビット15(16ビットデータの符号ビット)を*rd*レジスタの下位16ビットに転送します。*rd*レジスタの上位8ビットは0に設定されます。



(2)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

例

r1レジスタが0x008000の場合

```
cv.ls  %r0,%r1   ; r0 = 0x00ffff
```

di

機　能　**割り込み禁止**

標準）psr(IE) ← 0
拡張1）不可
拡張2）不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

フラグ

IL	IE	C	V	Z	N
–	0	–	–	–	–

モード　–

CLK　1サイクル

説　明　(1)標準

PSRのIEビットを0にリセットし、マスク可能な外部割り込みを禁止します。
リセット割り込み、アドレス不整割り込み、NMIはIEビットが0の場合でも受け付けられます。

(2)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

例

```
di          ; マスク可能な外部割り込みを禁止します。
```

注　意　マスク可能な外部割り込みはdi命令の3サイクル後から禁止されます。

```
di
命令1                ← 1サイクル命令
命令2                ← 1サイクル命令
命令3                ← この命令から割り込みが禁止される
```

di、ei命令を組み合わせた割り込み禁止区間の例

```
ld %r2,%r3           ← 割り込み許可
di                   ← 割り込み許可
ld.a %r0,%r1         ← 割り込み許可
ld.b %r2,%r3         ← 割り込み許可
ld %r4,%r5           ← 割り込み禁止
ei                   ← 割り込み禁止
add %r4,%r5          ← 割り込み禁止
sub %r6,%r7          ← 割り込み禁止
cmp %r0,%r1          ← 割り込み許可
```

ei

機　能

割り込み許可

標準） psr(IE) ← 1
拡張1）不可
拡張2）不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

フラグ

IL	IE	C	V	Z	N
–	1	–	–	–	–

モード

–

CLK

1サイクル

説　明

(1)標準
PSRのIEビットを1にセットし、マスク可能な外部割り込みを許可します。

(2)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

例

```
ei          ; マスク可能な外部割り込みを許可します。
```

注　意

マスク可能な外部割り込みはei命令の3サイクル後から許可されます。

```
ei
命令1                ← 1サイクル命令
命令2                ← 1サイクル命令
命令3                ← この命令から割り込みが許可される
```

di、ei命令を組み合わせた割り込み禁止区間の例

```
ld %r2,%r3           ← 割り込み許可
di                   ← 割り込み許可
ld.a %r0,%r1         ← 割り込み許可
ld.b %r2,%r3         ← 割り込み許可
ld %r4,%r5           ← 割り込み禁止
ei                   ← 割り込み禁止
add %r4,%r5          ← 割り込み禁止
sub %r6,%r7          ← 割り込み禁止
cmp %r0,%r1          ← 割り込み許可
```

ext *imm13*

機　能

即値拡張

標準） 次の命令の即値/オペランドを拡張
拡張1）不可
拡張2）不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	*imm13*												

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

即値（符号なし）

CLK

1サイクル

説　明

直後の命令の即値あるいはオペランドを拡張します。
即値拡張の場合、ext命令で指定した即値が上位側、ターゲット命令（拡張対象命令）が持つ即値が下位側となります。

ext命令は連続2個まで使用可能です。その場合は、最初のext命令で指定した即値が最も上位側となります。3個以上のext命令が連続的に記述された場合は最後の2個が有効となり、それ以外は無視されます。
ext命令による拡張内容および使用例については各命令の説明を参照してください。

ext命令に対する割り込み（リセット、デバッグブレークを除く）はハードウェアによりマスクされ、拡張対象命令実行時に割り込み処理が受け付けられます。ただし、この場合の割り込み処理からのリターンアドレスはext命令の先頭になります。

例

```
ext     0x7ff
ext     0x1fff
add.a   %r1,%r2   ; r1 = r2 + 0xffffff
```

注　意

ext命令に続けてメモリとレジスタ間のロード命令を実行する場合、そのロード命令の実行前にアドレス不整割り込みが発生する可能性があります（ext命令で指定した即値をディスプレースメントとする指定アドレスが、転送データサイズのアドレス境界を指していない場合）。ここでアドレス不整割り込みが発生し、それによって実行された割り込み処理ルーチンを単純にreti命令で終了させると、その割り込み処理ルーチンからは割り込みを発生したロード命令のアドレスに戻り、直前のext命令が無効となります。したがって、リターンアドレスの操作等が必要になりますので注意してください。

halt

機　能

HALT
標準）プロセッサをHALTモードに設定
拡張1）不可
拡張2）不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

–

CLK

6サイクル

説　明

プロセッサをHALTモードにします。これにより、消費電流を抑えることができます。
S1C17コアが`halt`命令を実行すると、その時点でプログラムの実行を中断しHALTモードに移行します。
HALTモードではS1C17コアだけが動作を停止するのが一般的ですが、コア外部のクロック制御回路のインプリメンテーションに依存します。

HALTモードを解除する要因はイニシャルリセット以外、S1C17コア外部のクロック制御回路のインプリメンテーションに依存します。一般的にはイニシャルリセット、マスク可能な外部割り込み、NMI、デバッグ割り込みによって解除します。
割り込みによるHALTモードの解除には、プロセッサの割り込み許可/禁止の状態は影響しません。PSRのIEフラグや、割り込みコントローラの割り込み許可ビット（インプリメント依存）などが割り込み禁止に設定されている場合でも、割り込み信号によりHALTモードを解除することができます。
プロセッサが割り込み許可の状態で、割り込みによってHALTモードを解除した場合は、対応する割り込み処理ルーチンを実行します。したがって、発生した割り込みの処理ルーチンを`reti`で終了すると、`halt`の次の命令の位置にリターンします。
プロセッサが割り込み禁止状態の場合、HALTモードを解除後は`halt`の次の命令から実行を開始します。

HALTモードの詳細については、機種別のテクニカルマニュアルを参照してください。

例

```
halt                    ; プロセッサをHALTモードに設定
```

int *imm5*

機　能

ソフトウェア割り込み

標準） sp ← sp - 4, A[sp] ← {psr, pc + 2}, pc ← TTBR + (ベクタNo. = *imm5*) × 4
拡張1) 不可
拡張2) 不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	0	0	0	*imm5*					0	1

フラグ

IL	IE	C	V	Z	N
–	0	–	–	–	–

モード

即値(符号なし)

CLK

3サイクル

説　明

*imm5*で指定されたベクタ番号の割り込みを発生させます。
次の命令のアドレスとPSRをスタックにセーブ後、ベクタテーブルから指定のベクタを読み出してPCにロードします。これにより対応する割り込み処理ルーチンに分岐します。

imm5	ベクタNo.	ベクタアドレス	割り込み要因
0x00	0	TTBR + 0x00	リセット割り込み
0x01	1	TTBR + 0x04	アドレス不整割り込み
0x02	2	TTBR + 0x08	NMI
0x03	3	TTBR + 0x0c	マスク可能な外部割り込み0x03
:	:	:	:
0x1f	31	TTBR + 0x7c	マスク可能な外部割り込み0x1f

TTBRはベクタテーブルの先頭アドレスです。
処理ルーチンよりのリターンには`reti`命令を使用します。

例

```
int  2    ; NMIを発生
```

intl *imm5, imm3*

機　能

割り込みレベル指定付きソフトウェア割り込み

標準）sp ← sp - 4, A[sp] ← {psr, pc + 2}, pc ← TTBR + (ベクタNo. = *imm5*) × 4,
psr(IL) ← *imm3*

拡張1）不可

拡張2）不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	*imm3*			*imm5*					1	1

フラグ

IL	IE	C	V	Z	N
↔	0	–	–	–	–

モード　即値（符号なし）

CLK　3サイクル

説　明

*imm5*で指定されたベクタ番号の割り込みを発生させます。
次の命令のアドレスとPSRをスタックにセーブ後、ベクタテーブルから指定のベクタを読み出してPCにロードします。これにより対応する割り込み処理ルーチンに分岐します。さらに、PSRのILビット（割り込みレベル）を*imm3*の値に設定します。割り込み処理ルーチン内では*imm3*以下のレベルの割り込みは禁止されます。
変更されたILビットは、割り込み処理ルーチンを`reti`で終了する際に`intl`実行前の状態に戻ります。

例

```
intl   0x3,0x2     ; マスク可能な外部割り込み0x3を発生させ、ILを0x2に設定
```

jpa %rb
jpa.d %rb

機 能

無条件PC絶対ジャンプ

標準） pc ← *rb*

拡張1）不可

拡張2）不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	1	0	1	0	0	1	*r*	*b*		jpa
0	0	0	0	0	0	0	1	1	1	0	0	1	*r*	*b*		jpa.d

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

PC絶対

CLK

`jpa` 3サイクル

`jpa.d` 2サイクル（ディレイドスロット命令 = 1サイクルの場合）、3サイクル（その他）

説 明

（1）標準

`jpa %rb`

*rb*レジスタの内容をPCにロードして、そのアドレスに分岐します。*rb*レジスタの最下位ビットは無視され、常に0として扱われます。

（2）ディレイド分岐（dビット（ビット7）= 1）

`jpa.d %rb`

`jpa.d`命令では次の命令がディレイドスロット命令となります。ディレイドスロット命令は分岐前に実行されます。`jpa.d`命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
jpa  %r0          ; r0レジスタが示すアドレスにジャンプ
```

注 意

`jpa.d`命令（ディレイド分岐）を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

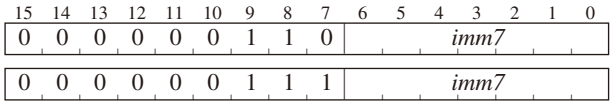
jpa *imm7*
jpa.d *imm7*

機 能

無条件PC絶対ジャンプ

標準） pc ← *imm7*
拡張1) pc ← *imm20*
拡張2) pc ← *imm24*

コード



フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

PC絶対

CLK

jpa	3サイクル
jpa.d	2サイクル(ディレイドスロット命令 = 1サイクルの場合)、3サイクル(その他)

説 明

(1)標準

```
jpa  imm7
```

7ビット即値*imm7*をPCにロードして、そのアドレスに分岐します。*imm7*の最下位ビットは無視され、常に0として扱われます。

(2)拡張1

```
ext  imm13    ; = imm20(19:7)
jpa  imm7     ; = "jpa imm20", imm7 = imm20(6:0)
```

ext命令の13ビット即値*imm13*により、PCに設定する分岐先アドレスが20ビットに拡張されます。

(3)拡張2

```
ext  imm4     ; imm4(3:0) = imm24(23:20)
ext  imm13    ; = imm24(19:7)
jpa  imm7     ; = "jpa imm24", imm7 = imm24(6:0)
```

PCに設定する分岐先アドレスが24ビットに拡張されます。

(4)ディレイド分岐(dビット(ビット7) = 1)

```
jpa.d  imm7
```

jpa.d命令では次の命令がディレイドスロット命令となります。ディレイドスロット命令は分岐前に実行されます。jpa.d命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
ext  0x300
jpa  0x00        ; アドレス0x18000にジャンプ
```

注 意

jpa.d命令(ディレイド分岐)を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

jpr %rb
jpr.d %rb

機 能

無条件PC相対ジャンプ

標準） pc ← pc + 2 + *rb*
拡張1）不可
拡張2）不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	1	0	1	0	0	0	*r b*			jpr
0	0	0	0	0	0	0	1	1	1	0	0	0	*r b*			jpr.d

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

符号付きPC相対

CLK

jpr　　3サイクル
jpr.d　　2サイクル（ディレイドスロット命令 = 1サイクルの場合）、3サイクル（その他）

説 明

（1）標準

```
jpr  %rb
```

*rb*レジスタの内容をPC（PC + 2）に加算して、そのアドレスに分岐します。*rb*レジスタの最下位ビットは無視され、常に0として扱われます。

（2）ディレイド分岐（dビット（ビット7）= 1）

```
jpr.d  %rb
```

jpr.d命令では次の命令がディレイドスロット命令となります。ディレイドスロット命令は分岐前に実行されます。jpr.d命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
jpr  %r0          ; pc ← pc + 2 + r0
```

注 意

jpr.d命令（ディレイド分岐）を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

jpr *sign10*
jpr.d *sign10*

機 能

無条件PC相対ジャンプ

標準） pc ← pc + 2 + *sign10* × 2
拡張1) pc ← pc + 2 + *sign24*
拡張2) 不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	1	0	0	*sign10*										jpr
0	0	0	1	0	1	*sign10*										jpr.d

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

符号付きPC相対

CLK

jpr　　3サイクル
jpr.d　　2サイクル(ディレイドスロット命令 = 1サイクルの場合)、3サイクル(その他)

説 明

(1)標準

```
jpr  sign10   ; = "jp sign11", sign10 = sign11(10:1), sign11(0)=0
```

符号付き10ビット即値*sign10*を2倍してPC(PC + 2)に加算し、そのアドレスに分岐します。*sign10*は16ビット単位のワードアドレスを指定します。
sign10(×2)による分岐可能範囲はPC - 1,022～PC + 1,024です。

(2)拡張1

```
ext  imm13    ; = sign24(23:11)
jpr  sign10   ; = "jpr sign24", sign10 = sign24(10:1), sign24(0)=0
```

ext命令の13ビット即値*imm13*により、PCに加算するディスプレースメントが符号付き24ビットに拡張されます。
*sign24*による分岐可能範囲はPC - 8,388,606～PC + 8,388,608です。

(3)ディレイド分岐(dビット(ビット10) = 1)

```
jpr.d  sign10
```

jp.d命令では次の命令がディレイドスロット命令となります。ディレイドスロット命令は分岐前に実行されます。jp.d命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
ext   0x20
jpr   0x00          ; pc + 2 + 0x10000番地へジャンプ
```

注 意

jpr.d命令(ディレイド分岐)を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

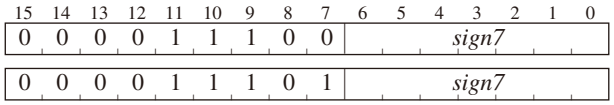
jreq *sign7*
jreq.d *sign7*

機 能

条件PC相対ジャンプ

標準） pc ← pc + 2 + *sign7* × 2 if Z is true
拡張1) pc ← pc + 2 + *sign21* if Z is true
拡張2) pc ← pc + 2 + *sign24* if Z is true

コード



フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

符号付きPC相対

CLK

`jreq` 2サイクル(分岐しない場合)、3サイクル(分岐する場合)
`jreq.d` 2サイクル(ディレイドスロット命令 = 1サイクルの場合)、3サイクル(その他)

説 明

(1)標準

```
jreq  sign7   ; = "jreq sign8", sign7 = sign8(7:1), sign8(0)=0
```

次の条件が成立している場合、符号付き7ビット即値*sign7*を2倍してPC(PC + 2)に加算し、そのアドレスに分岐します。条件が不成立の場合は、分岐しません。
• Zフラグ＝1(例: `cmp A,B`の実行結果が“A＝B”)
*sign7*は16ビット単位のワードアドレスを指定します。*sign7*(×2)による分岐可能範囲はPC - 126～PC + 128です。

(2)拡張1

```
ext   imm13   ; = sign21(20:8)
jreq  sign7   ; = "jreq sign21", sign7 = sign21(7:1), sign21(0)=0
```

`ext`命令の13ビット即値*imm13*により、PC(PC + 2)に加算するディスプレースメントが符号付き21ビットに拡張されます。*sign21*による分岐可能範囲はPC - 1,048,574～PC + 1,048,576です。

(3)拡張2

```
ext   imm3    ; imm3(2:0)= sign24(23:21)
ext   imm13   ; = sign24(20:8)
jreq  sign7   ; = "jreq sign24", sign7 = sign24(7:1), sign24(0)=0
```

`ext`命令の2つの即値(*imm3*、*imm13*)により、PC(PC + 2)に加算するディスプレースメントが符号付き24ビットに拡張されます。*sign24*による分岐可能範囲はPC - 8,388,606～PC + 8,388,608です。

(4)ディレイド分岐(dビット(ビット7) = 1)

```
jreq.d  sign7
```

`jreq.d`命令では次の命令がディレイドスロット命令となります。ディレイドスロット命令は分岐前に実行されます。`jreq.d`命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
cmp   %r0,%r1
jreq  0x1       ; r0 = r1ならば次の命令をスキップ
```

注 意

`jreq.d`命令(ディレイド分岐)を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

jrge *sign7*
jrge.d *sign7*

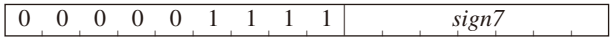
機 能

条件PC相対ジャンプ(符号付き演算結果判定)

標準) pc ← pc + 2 + *sign7* × 2 if !(N^V) is true
拡張1) pc ← pc + 2 + *sign21* if !(N^V) is true
拡張2) pc ← pc + 2 + *sign24* if !(N^V) is true

コード

jrge



jrge.d

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

符号付きPC相対

CLK

jrge　　2サイクル(分岐しない場合)、3サイクル(分岐する場合)
jrge.d　2サイクル(ディレイドスロット命令 = 1サイクルの場合)、3サイクル(その他)

説 明

(1)標準

```
jrge  sign7   ; = "jrge sign8", sign7 = sign8(7:1), sign8(0)=0
```

次の条件が成立している場合、符号付き7ビット即値*sign7*を2倍してPC(PC + 2)に加算し、そのアドレスに分岐します。条件が不成立の場合は、分岐しません。
• Nフラグ=Vフラグ(例: cmp A,Bの実行結果が"A≧B")
*sign7*は16ビット単位のワードアドレスを指定します。*sign7*(×2)による分岐可能範囲はPC - 126～PC + 128です。

(2)拡張1

```
ext   imm13   ; = sign21(20:8)
jrge  sign7   ; = "jrge sign21", sign7 = sign21(7:1), sign21(0)=0
```

ext命令の13ビット即値*imm13*により、PC(PC + 2)に加算するディスプレースメントが符号付き21ビットに拡張されます。*sign21*による分岐可能範囲はPC - 1,048,574～PC + 1,048,576です。

(3)拡張2

```
ext   imm3    ; imm3(2:0)= sign24(23:21)
ext   imm13   ; = sign24(20:8)
jrge  sign7   ; = "jrge sign24", sign7 = sign24(7:1), sign24(0)=0
```

ext命令の2つの即値(*imm3*、*imm13*)により、PC(PC + 2)に加算するディスプレースメントが符号付き24ビットに拡張されます。*sign24*による分岐可能範囲はPC - 8,388,606～PC + 8,388,608です。

(4)ディレイド分岐(dビット(ビット7) = 1)

```
jrge.d  sign7
```

jrge.d命令では次の命令がディレイドスロット命令となります。ディレイドスロット命令は分岐前に実行されます。jrge.d命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
cmp   %r0,%r1    ; r0、r1に符号付きデータがロードされている場合
jrge  0x1        ; r0≧r1ならば次の命令をスキップ
```

注 意

jrge.d命令(ディレイド分岐)を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

jrgt *sign7*
jrgt.d *sign7*

機　能

条件PC相対ジャンプ(符号付き演算結果判定)

標準) pc ← pc + 2 + *sign7* × 2 if !Z&!(N^V) is true
拡張1) pc ← pc + 2 + *sign21* if !Z&!(N^V) is true
拡張2) pc ← pc + 2 + *sign24* if !Z&!(N^V) is true

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	1	1	0	0	*sign7*							jrgt
0	0	0	0	0	1	1	0	1	*sign7*							jrgt.d

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

符号付きPC相対

CLK

`jrgt`　　2サイクル(分岐しない場合)、3サイクル(分岐する場合)
`jrgt.d`　2サイクル(ディレイドスロット命令 = 1サイクルの場合)、3サイクル(その他)

説　明

(1)標準

```
jrgt  sign7   ; = "jrgt sign8", sign7 = sign8(7:1), sign8(0)=0
```

次の条件が成立している場合、符号付き7ビット即値*sign7*を2倍してPC(PC + 2)に加算し、そのアドレスに分岐します。条件が不成立の場合は、分岐しません。
• Zフラグ＝0 かつ Nフラグ＝Vフラグ(例: `cmp A,B`の実行結果が"A＞B")
*sign7*は16ビット単位のワードアドレスを指定します。*sign7*(×2)による分岐可能範囲はPC - 126～PC + 128です。

(2)拡張1

```
ext   imm13   ; = sign21(20:8)
jrgt  sign7   ; = "jrgt sign21", sign7 = sign21(7:1), sign21(0)=0
```

`ext`命令の13ビット即値*imm13*により、PC(PC + 2)に加算するディスプレースメントが符号付き21ビットに拡張されます。*sign21*による分岐可能範囲はPC - 1,048,574～PC + 1,048,576です。

(3)拡張2

```
ext   imm3    ; imm3(2:0)= sign24(23:21)
ext   imm13   ; = sign24(20:8)
jrgt  sign7   ; = "jrgt sign24", sign7 = sign24(7:1), sign24(0)=0
```

`ext`命令の2つの即値(*imm3*、*imm13*)により、PC(PC + 2)に加算するディスプレースメントが符号付き24ビットに拡張されます。*sign24*による分岐可能範囲はPC - 8,388,606～PC + 8,388,608です。

(4)ディレイド分岐(dビット(ビット7) = 1)

```
jrgt.d  sign7
```

`jrgt.d`命令では次の命令がディレイドスロット命令となります。ディレイドスロット命令は分岐前に実行されます。`jrgt.d`命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
cmp   %r0,%r1    ; r0、r1に符号付きデータがロードされている場合
jrgt  0x1        ; r0＞r1ならば次の命令をスキップ
```

注　意

`jrgt.d`命令(ディレイド分岐)を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

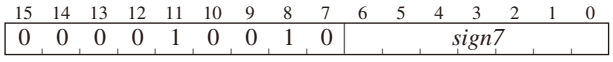
jrle *sign7*
jrle.d *sign7*

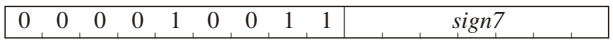
機 能

条件PC相対ジャンプ(符号付き演算結果判定)

標準) pc ← pc + 2 + *sign7* × 2 if Z | (N^V) is true
拡張1) pc ← pc + 2 + *sign21* if Z | (N^V) is true
拡張2) pc ← pc + 2 + *sign24* if Z | (N^V) is true

コード





フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

符号付きPC相対

CLK

`jrle` 2サイクル(分岐しない場合)、3サイクル(分岐する場合)
`jrle.d` 2サイクル(ディレイドスロット命令 = 1サイクルの場合)、3サイクル(その他)

説 明

(1)標準

```
jrle  sign7   ; = "jrle sign8", sign7 = sign8(7:1), sign8(0)=0
```

次の条件が成立している場合、符号付き7ビット即値*sign7*を2倍してPC(PC + 2)に加算し、そのアドレスに分岐します。条件が不成立の場合は、分岐しません。
• Zフラグ=1 または Nフラグ≠Vフラグ(例: `cmp A,B`の実行結果が"A≦B")
*sign7*は16ビット単位のワードアドレスを指定します。*sign7*(×2)による分岐可能範囲はPC - 126～PC + 128です。

(2)拡張1

```
ext   imm13   ; = sign21(20:8)
jrle  sign7   ; = "jrle sign21", sign7 = sign21(7:1), sign21(0)=0
```

`ext`命令の13ビット即値*imm13*により、PC(PC + 2)に加算するディスプレースメントが符号付き21ビットに拡張されます。*sign21*による分岐可能範囲はPC - 1,048,574～PC + 1,048,576です。

(3)拡張2

```
ext   imm3    ; imm3(2:0)= sign24(23:21)
ext   imm13   ; = sign24(20:8)
jrle  sign7   ; = "jrle sign24", sign7 = sign24(7:1), sign24(0)=0
```

`ext`命令の2つの即値(*imm3*、*imm13*)により、PC(PC + 2)に加算するディスプレースメントが符号付き24ビットに拡張されます。*sign24*による分岐可能範囲はPC - 8,388,606～PC + 8,388,608です。

(4)ディレイド分岐(dビット(ビット7) = 1)

```
jrle.d  sign7
```

`jrle.d`命令では次の命令がディレイドスロット命令となります。ディレイドスロット命令は分岐前に実行されます。`jrle.d`命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
cmp   %r0,%r1    ; r0、r1に符号付きデータがロードされている場合
jrle  0x1        ; r0≦r1ならば次の命令をスキップ
```

注 意

`jrle.d`命令(ディレイド分岐)を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

jrlt *sign7*
jrlt.d *sign7*

機　能

条件PC相対ジャンプ(符号付き演算結果判定)

標準) pc ← pc + 2 + *sign7* × 2 if N^V is true
拡張1) pc ← pc + 2 + *sign21* if N^V is true
拡張2) pc ← pc + 2 + *sign24* if N^V is true

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	1	0	0	0	0	*sign7*							jrlt
0	0	0	0	1	0	0	0	1	*sign7*							jrlt.d

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

符号付きPC相対

CLK

`jrlt`　　2サイクル(分岐しない場合)、3サイクル(分岐する場合)
`jrlt.d`　2サイクル(ディレイドスロット命令 = 1サイクルの場合)、3サイクル(その他)

説　明

(1)標準

```
jrlt  sign7   ; = "jrlt sign8", sign7 = sign8(7:1), sign8(0)=0
```

次の条件が成立している場合、符号付き7ビット即値*sign7*を2倍してPC(PC + 2)に加算し、そのアドレスに分岐します。条件が不成立の場合は、分岐しません。
• Nフラグ≠Vフラグ(例: `cmp A,B`の実行結果が"A＜B")
*sign7*は16ビット単位のワードアドレスを指定します。*sign7*(×2)による分岐可能範囲はPC - 126～PC + 128です。

(2)拡張1

```
ext   imm13   ; = sign21(20:8)
jrlt  sign7   ; = "jrlt sign21", sign7 = sign21(7:1), sign21(0)=0
```

`ext`命令の13ビット即値*imm13*により、PC(PC + 2)に加算するディスプレースメントが符号付き21ビットに拡張されます。*sign21*による分岐可能範囲はPC - 1,048,574～PC + 1,048,576です。

(3)拡張2

```
ext   imm3    ; imm3(2:0)= sign24(23:21)
ext   imm13   ; = sign24(20:8)
jrlt  sign7   ; = "jrlt sign24", sign7 = sign24(7:1), sign24(0)=0
```

`ext`命令の2つの即値(*imm3*、*imm13*)により、PC(PC + 2)に加算するディスプレースメントが符号付き24ビットに拡張されます。*sign24*による分岐可能範囲はPC - 8,388,606～PC + 8,388,608です。

(4)ディレイド分岐(dビット(ビット7) = 1)

```
jrlt.d  sign7
```

`jrlt.d`命令では次の命令がディレイドスロット命令となります。ディレイドスロット命令は分岐前に実行されます。`jrlt.d`命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
cmp   %r0,%r1    ; r0、r1に符号付きデータがロードされている場合
jrlt  0x1        ; r0＜r1ならば次の命令をスキップ
```

注　意

`jrlt.d`命令(ディレイド分岐)を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

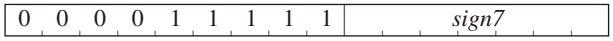
jrne *sign7*
jrne.d *sign7*

機 能

条件PC相対ジャンプ

標準） pc ← pc + 2 + *sign7* × 2 if !Z is true
拡張1) pc ← pc + 2 + *sign21* if !Z is true
拡張2) pc ← pc + 2 + *sign24* if !Z is true

コード



フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

符号付きPC相対

CLK

jrne　　2サイクル(分岐しない場合)、3サイクル(分岐する場合)
jrne.d　2サイクル(ディレイドスロット命令 = 1サイクルの場合)、3サイクル(その他)

説 明

(1)標準

```
jrne  sign7   ; = "jrne sign8", sign7 = sign8(7:1), sign8(0)=0
```

次の条件が成立している場合、符号付き7ビット即値*sign7*を2倍してPC(PC + 2)に加算し、そのアドレスに分岐します。条件が不成立の場合は、分岐しません。
• Zフラグ＝0(例: cmp A,Bの実行結果が“A≠B”)
*sign7*は16ビット単位のワードアドレスを指定します。*sign7*(×2)による分岐可能範囲はPC - 126～PC + 128です。

(2)拡張1

```
ext   imm13   ; = sign21(20:8)
jrne  sign7   ; = "jrne sign21", sign7 = sign21(7:1), sign21(0)=0
```

ext命令の13ビット即値*imm13*により、PC(PC + 2)に加算するディスプレースメントが符号付き21ビットに拡張されます。*sign21*による分岐可能範囲はPC - 1,048,574～PC + 1,048,576です。

(3)拡張2

```
ext   imm3    ; imm3(2:0)= sign24(23:21)
ext   imm13   ; = sign24(20:8)
jrne  sign7   ; = "jrne sign24", sign7 = sign24(7:1), sign24(0)=0
```

ext命令の2つの即値(*imm3*、*imm13*)により、PC(PC + 2)に加算するディスプレースメントが符号付き24ビットに拡張されます。*sign24*による分岐可能範囲はPC - 8,388,606～PC + 8,388,608です。

(4)ディレイド分岐(dビット(ビット7) = 1)

```
jrne.d  sign7
```

jrne.d命令では次の命令がディレイドスロット命令となります。ディレイドスロット命令は分岐前に実行されます。jrne.d命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
cmp   %r0,%r1
jrne  0x1         ; r0≠r1ならば次の命令をスキップ
```

注 意

jrne.d命令(ディレイド分岐)を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

jruge *sign7*
jruge.d *sign7*

機　能

条件PC相対ジャンプ(符号なし演算結果判定)

標準) pc ← pc + 2 + *sign7* × 2 if !C is true
拡張1) pc ← pc + 2 + *sign21* if !C is true
拡張2) pc ← pc + 2 + *sign24* if !C is true

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	1	0	1	1	0	*sign7*							jruge
0	0	0	0	1	0	1	1	1	*sign7*							jruge.d

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

符号付きPC相対

CLK

jruge　　2サイクル(分岐しない場合)、3サイクル(分岐する場合)
jruge.d　2サイクル(ディレイドスロット命令 = 1サイクルの場合)、3サイクル(その他)

説　明

(1)標準

```
jruge  sign7  ; = "jruge sign8", sign7 = sign8(7:1), sign8(0)=0
```

次の条件が成立している場合、符号付き7ビット即値*sign7*を2倍してPC(PC + 2)に加算し、そのアドレスに分岐します。条件が不成立の場合は、分岐しません。
• Cフラグ＝0(例: cmp A,Bの実行結果が"A≧B")
*sign7*は16ビット単位のワードアドレスを指定します。*sign7*(×2)による分岐可能範囲はPC - 126～PC + 128です。

(2)拡張1

```
ext    imm13  ; = sign21(20:8)
jruge  sign7  ; = "jruge sign21", sign7 = sign21(7:1), sign21(0)=0
```

ext命令の13ビット即値*imm13*により、PC(PC + 2)に加算するディスプレースメントが符号付き21ビットに拡張されます。*sign21*による分岐可能範囲はPC - 1,048,574～PC + 1,048,576です。

(3)拡張2

```
ext    imm3   ; imm3(2:0)= sign24(23:21)
ext    imm13  ; = sign24(20:8)
jruge  sign7  ; = "jruge sign24", sign7 = sign24(7:1), sign24(0)=0
```

ext命令の2つの即値(*imm3*、*imm13*)により、PC(PC + 2)に加算するディスプレースメントが符号付き24ビットに拡張されます。*sign24*による分岐可能範囲はPC - 8,388,606～PC + 8,388,608です。

(4)ディレイド分岐(dビット(ビット7) = 1)

```
jruge.d  sign7
```

jruge.d命令では次の命令がディレイドスロット命令となります。ディレイドスロット命令は分岐前に実行されます。jruge.d命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
cmp     %r0,%r1   ; r0、r1に符号なしデータがロードされている場合
jruge   0x1       ; r0≧r1ならば次の命令をスキップ
```

注　意

jruge.d命令(ディレイド分岐)を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

jrugt *sign7*
jrugt.d *sign7*

機 能

条件PC相対ジャンプ(符号なし演算結果判定)

標準) pc ← pc + 2 + *sign7* × 2 if !Z&!C is true
拡張1) pc ← pc + 2 + *sign21* if !Z&!C is true
拡張2) pc ← pc + 2 + *sign24* if !Z&!C is true

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	1	0	1	0	0	*sign7*							jrugt
0	0	0	0	1	0	1	0	1	*sign7*							jrugt.d

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード 符号付きPC相対

CLK

`jrugt` 2サイクル(分岐しない場合)、3サイクル(分岐する場合)
`jrugt.d` 2サイクル(ディレイドスロット命令 = 1サイクルの場合)、3サイクル(その他)

説 明

(1)標準

```
jrugt  sign7  ; = "jrugt sign8", sign7 = sign8(7:1), sign8(0)=0
```

次の条件が成立している場合、符号付き7ビット即値*sign7*を2倍してPC(PC + 2)に加算し、そのアドレスに分岐します。条件が不成立の場合は、分岐しません。
• Zフラグ=0 かつ Cフラグ=0(例: `cmp A,B`の実行結果が"A>B")
*sign7*は16ビット単位のワードアドレスを指定します。*sign7*(×2)による分岐可能範囲はPC - 126～PC + 128です。

(2)拡張1

```
ext    imm13  ; = sign21(20:8)
jrugt  sign7  ; = "jrugt sign21", sign7 = sign21(7:1), sign21(0)=0
```

`ext`命令の13ビット即値*imm13*により、PC(PC + 2)に加算するディスプレースメントが符号付き21ビットに拡張されます。*sign21*による分岐可能範囲はPC - 1,048,574～PC + 1,048,576です。

(3)拡張2

```
ext    imm3   ; imm3(2:0)= sign24(23:21)
ext    imm13  ; = sign24(20:8)
jrugt  sign7  ; = "jrugt sign24", sign7 = sign24(7:1), sign24(0)=0
```

`ext`命令の2つの即値(*imm3*、*imm13*)により、PC(PC + 2)に加算するディスプレースメントが符号付き24ビットに拡張されます。*sign24*による分岐可能範囲はPC - 8,388,606～PC + 8,388,608です。

(4)ディレイド分岐(dビット(ビット7) = 1)

```
jrugt.d  sign7
```

`jrugt.d`命令では次の命令がディレイドスロット命令となります。ディレイドスロット命令は分岐前に実行されます。`jrugt.d`命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
cmp     %r0,%r1   ; r0、r1に符号なしデータがロードされている場合
jrugt   0x1       ; r0>r1ならば次の命令をスキップ
```

注 意

`jrugt.d`命令(ディレイド分岐)を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

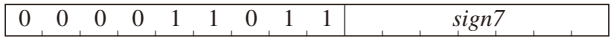
jrule *sign7*
jrule.d *sign7*

機 能 条件PC相対ジャンプ(符号なし演算結果判定)

標準) pc ← pc + 2 + *sign7* × 2 if Z | C is true
拡張1) pc ← pc + 2 + *sign21* if Z | C is true
拡張2) pc ← pc + 2 + *sign24* if Z | C is true

コード

jrule



jrule.d

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード 符号付きPC相対

CLK

jrule 2サイクル(分岐しない場合)、3サイクル(分岐する場合)
jrule.d 2サイクル(ディレイドスロット命令 = 1サイクルの場合)、3サイクル(その他)

説 明

(1)標準

```
jrule  sign7  ; = "jrule sign8", sign7 = sign8(7:1), sign8(0)=0
```

次の条件が成立している場合、符号付き7ビット即値*sign7*を2倍してPC(PC + 2)に加算し、そのアドレスに分岐します。条件が不成立の場合は、分岐しません。
• Zフラグ=1 または Cフラグ=1(例: cmp A,Bの実行結果が"A≦B")
*sign7*は16ビット単位のワードアドレスを指定します。*sign7*(×2)による分岐可能範囲はPC - 126～PC + 128です。

(2)拡張1

```
ext    imm13  ; = sign21(20:8)
jrule  sign7  ; = "jrule sign21", sign7 = sign21(7:1), sign21(0)=0
```

ext命令の13ビット即値*imm13*により、PC(PC + 2)に加算するディスプレースメントが符号付き21ビットに拡張されます。*sign21*による分岐可能範囲はPC - 1,048,574～PC + 1,048,576です。

(3)拡張2

```
ext    imm3   ; imm3(2:0)= sign24(23:21)
ext    imm13  ; = sign24(20:8)
jrule  sign7  ; = "jrule sign24", sign7 = sign24(7:1), sign24(0)=0
```

ext命令の2つの即値(*imm3*、*imm13*)により、PC(PC + 2)に加算するディスプレースメントが符号付き24ビットに拡張されます。*sign24*による分岐可能範囲はPC - 8,388,606～PC + 8,388,608です。

(4)ディレイド分岐(dビット(ビット7) = 1)

```
jrule.d  sign7
```

jrule.d命令では次の命令がディレイドスロット命令となります。ディレイドスロット命令は分岐前に実行されます。jrule.d命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
cmp    %r0,%r1   ; r0、r1に符号なしデータがロードされている場合
jrule  0x1       ; r0≦r1ならば次の命令をスキップ
```

注 意 jrule.d命令(ディレイド分岐)を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

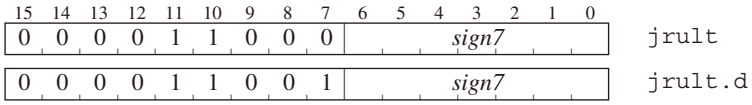
jrult *sign7*
jrult.d *sign7*

機 能

条件PC相対ジャンプ(符号なし演算結果判定)

標準) pc ← pc + 2 + *sign7* × 2 if C is true
拡張1) pc ← pc + 2 + *sign21* if C is true
拡張2) pc ← pc + 2 + *sign24* if C is true

コード



フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

符号付きPC相対

CLK

jrult　2サイクル(分岐しない場合)、3サイクル(分岐する場合)
jrult.d　2サイクル(ディレイドスロット命令 = 1サイクルの場合)、3サイクル(その他)

説 明

(1)標準

```
jrult  sign7  ; = "jrult sign8", sign7 = sign8(7:1), sign8(0)=0
```

次の条件が成立している場合、符号付き7ビット即値*sign7*を2倍してPC(PC + 2)に加算し、そのアドレスに分岐します。条件が不成立の場合は、分岐しません。
• Cフラグ=1(例: cmp A,Bの実行結果が"A<B")
*sign7*は16ビット単位のワードアドレスを指定します。*sign7*(×2)による分岐可能範囲はPC - 126～PC + 128です。

(2)拡張1

```
ext    imm13  ; = sign21(20:8)
jrult  sign7  ; = "jrult sign21", sign7 = sign21(7:1), sign21(0)=0
```

ext命令の13ビット即値*imm13*により、PC(PC + 2)に加算するディスプレースメントが符号付き21ビットに拡張されます。*sign21*による分岐可能範囲はPC - 1,048,574～PC + 1,048,576です。

(3)拡張2

```
ext    imm3   ; imm3(2:0)= sign24(23:21)
ext    imm13  ; = sign24(20:8)
jrult  sign7  ; = "jrult sign24", sign7 = sign24(7:1), sign24(0)=0
```

ext命令の2つの即値(*imm3*、*imm13*)により、PC(PC + 2)に加算するディスプレースメントが符号付き24ビットに拡張されます。*sign24*による分岐可能範囲はPC - 8,388,606～PC + 8,388,608です。

(4)ディレイド分岐(dビット(ビット7) = 1)

```
jrult.d  sign7
```

jrult.d命令では次の命令がディレイドスロット命令となります。ディレイドスロット命令は分岐前に実行されます。jrult.d命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
cmp     %r0,%r1   ; r0、r1に符号なしデータがロードされている場合
jrult   0x1       ; r0<r1ならば次の命令をスキップ
```

注 意

jrult.d命令(ディレイド分岐)を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

ld *%rd, %rs*

機　能

16ビットデータ転送

標準）　*rd*(15:0) ← *rs*(15:0), *rd*(23:16) ← 0
拡張1）　不可
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	0	*r d*			0	0	1	0	*r s*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接　`%rs` = `%r0`〜`%r7`
Dst: レジスタ直接　`%rd` = `%r0`〜`%r7`

CLK

1サイクル

説　明

(1)標準
*rs*レジスタの下位16ビットを*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。

(2)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

例

```
ld  %r0,%r1      ; r0 ← r1(15:0)
```

ld %rd, [%rb]

機　能

16ビットデータ転送

標準）　*rd*(15:0) ← W[*rb*], *rd*(23:16) ← 0
拡張1）　*rd*(15:0) ← W[*rb* + *imm13*], *rd*(23:16) ← 0
拡張2）　*rd*(15:0) ← W[*rb* + *imm24*], *rd*(23:16) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	*r d*			0	0	1	0	*r b*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ間接　`%rb` = `%r0`～`%r7`
Dst: レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

1サイクル（`ext`命令使用時は2サイクル）

説　明

(1)標準

```
ld   %rd,[%rb]          ; memory address = rb
```

指定メモリの16ビットデータを*rd*レジスタに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。*rd*レジスタの上位8ビットは0に設定されます。

(2)拡張1

```
ext   imm13
ld    %rd,[%rb]         ; memory address = rb + imm13
```

`ext`命令により、アドレッシングモードがディスプレースメント付きレジスタ間接アドレッシングに変わります。これにより、*rb*レジスタの内容に13ビット即値*imm13*を加えたアドレスの16ビットデータを*rd*レジスタに転送します。*rb*レジスタの内容は変更されません。*rd*レジスタの上位8ビットは0に設定されます。

(3)拡張2

```
ext   imm11             ; imm11(10:0) = imm24(23:13)
ext   imm13             ; = imm24(12:0)
ld    %rd,[%rb]         ; memory address = rb + imm24
```

アドレッシングモードがディスプレースメント付きレジスタ間接アドレッシングに変わり、*rb*レジスタの内容に24ビット即値*imm24*を加えたアドレスの16ビットデータを*rd*レジスタに転送します。*rb*レジスタの内容は変更されません。*rd*レジスタの上位8ビットは0に設定されます。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。 この場合は`ext`命令による拡張は行えません。

注　意

*rb*レジスタおよびディスプレースメントで指定されるメモリアドレスは、16ビット境界（最下位ビット = 0）を示していることが必要です。奇数アドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの最下位ビットを0としてデータ転送は実行されます。

ld *%rd*, [*%rb*]+
ld *%rd*, [*%rb*]-
ld *%rd*, -[*%rb*]

機　能　アドレスインクリメント/デクリメントオプション付き16ビットデータ転送

ld *%rd*, [*%rb*]+（ポストインクリメントオプション付き）

標準）　*rd*(15:0) ← W[*rb*], *rd*(23:16) ← 0, *rb*(23:0) ← *rb*(23:0) + 2
拡張1）　*rd*(15:0) ← W[*rb*], *rd*(23:16) ← 0, *rb*(23:0) ← *rb*(23:0) + *imm13*
拡張2）　*rd*(15:0) ← W[*rb*], *rd*(23:16) ← 0, *rb*(23:0) ← *rb*(23:0) + *imm24*

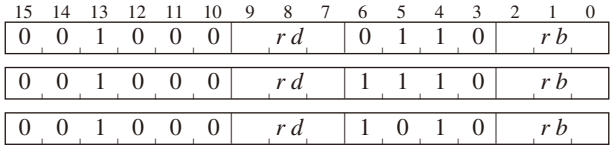
ld *%rd*, [*%rb*]-（ポストデクリメントオプション付き）

標準）　*rd*(15:0) ← W[*rb*], *rd*(23:16) ← 0, *rb*(23:0) ← *rb*(23:0) - 2
拡張1）　*rd*(15:0) ← W[*rb*], *rd*(23:16) ← 0, *rb*(23:0) ← *rb*(23:0) - *imm13*
拡張2）　*rd*(15:0) ← W[*rb*], *rd*(23:16) ← 0, *rb*(23:0) ← *rb*(23:0) - *imm24*

ld *%rd*, -[*%rb*]（プリデクリメントオプション付き）

標準）　*rb*(23:0) ← *rb*(23:0) - 2, *rd*(15:0) ← W[*rb*], *rd*(23:16) ← 0
拡張1）　*rb*(23:0) ← *rb*(23:0) - *imm13*, *rd*(15:0) ← W[*rb*], *rd*(23:16) ← 0
拡張2）　*rb*(23:0) ← *rb*(23:0) - *imm24*, *rd*(15:0) ← W[*rb*], *rd*(23:16) ← 0

コード



フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード　Src: レジスタ間接　`%rb` = `%r0`～`%r7`
Dst: レジスタ直接　`%rd` = `%r0`～`%r7`

CLK　2サイクル

説　明　(1)アドレスインクリメント/デクリメントオプション

`[]+`、`[]-`、`-[]`のオプション指定により、メモリアドレスが自動的にインクリメント/デクリメントされ、連続したデータ転送が簡単にプログラムできます。

`ld  %rd,[%rb]+`　ポストインクリメント付きロード命令
データ転送の後、メモリアドレスがインクリメントされます。

`ld  %rd,[%rb]-`　ポストデクリメント付きロード命令
データ転送の後、メモリアドレスがデクリメントされます。

`ld  %rd,-[%rb]`　プリデクリメント付きロード命令
データ転送の前にメモリアドレスがデクリメントされます。

インクリメント/デクリメントサイズは次のとおりです。
`ext`命令なし(標準): 2(16ビットサイズ)
`ext` 1個付き(拡張1): *imm13*
`ext` 2個付き(拡張2): *imm24*

(2)標準(ポストインクリメントの例)

```
ld  %rd,[%rb]+    ; source memory address = rb
                  ; post increment: rb + 2
```

指定メモリの16ビットデータを*rd*レジスタに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。*rd*レジスタの上位8ビットは0に設定されます。データ転送の後、メモリアドレスが2バイト分インクリメントされます。

(3)拡張1(ポストデクリメントの例)

```
ext   imm13
ld    %rd,[%rb]-   ; source memory address = rb
                   ; post decrement: rb - imm13
```

指定メモリの16ビットデータを*rd*レジスタに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。*rd*レジスタの上位8ビットは0に設定されます。データ転送の後、メモリアドレスが*imm13*バイト分デクリメントされます。

(4)拡張2(プリデクリメントの例)

```
ext   imm11        ; imm11(10:0) = imm24(23:13)
ext   imm13        ; = imm24(12:0)
ld    %rd,-[%rb]   ; source memory address = rb - imm24
```

*rb*レジスタで指定されるメモリアドレスを*imm24*バイト分デクリメントした後、そのアドレスの16ビットデータを*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。

(5)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

注 意 *rb*レジスタおよび即値で指定されるメモリアドレスは、16ビット境界(最下位ビット = 0)を示していることが必要です。奇数アドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの最下位ビットを0としてデータ転送は実行されます。

ld *%rd*, [%sp + *imm7*]

機　能

16ビットデータ転送

標準）　*rd*(15:0) ← W[sp + *imm7*], *rd*(23:16) ← 0
拡張1）　*rd*(15:0) ← W[sp + *imm20*], *rd*(23:16) ← 0
拡張2）　*rd*(15:0) ← W[sp + *imm24*], *rd*(23:16) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: ディスプレースメント付きレジスタ間接
Dst: レジスタ直接　`%rd` = `%r0`〜`%r7`

CLK

2サイクル

説　明

(1)標準

```
ld   %rd,[%sp + imm7]          ; memory address = sp + imm7
```

指定メモリの16ビットデータを*rd*レジスタに転送します。現在のSPの内容に7ビット即値*imm7*をディスプレースメントとして加算した値がアクセスされるメモリアドレスとなります。*rd*レジスタの上位8ビットは0に設定されます。

(2)拡張1

```
ext   imm13                    ; = imm20(19:7)
ld    %rd,[%sp + imm7]         ; memory address = sp + imm20,
                               ; imm7 = imm20(6:0)
```

`ext`命令により、ディスプレースメントが20ビットに拡張されます。これにより、SPの内容に20ビット即値*imm20*を加えたアドレスの16ビットデータを*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。

(3)拡張2

```
ext   imm4                     ; imm4(3:0) = imm24(23:20)
ext   imm13                    ; = imm24(19:7)
ld    %rd,[%sp + imm7]         ; memory address = sp + imm24,
                               ; imm7 = imm24(6:0)
```

2つの`ext`命令により、ディスプレースメントが24ビットに拡張されます。これにより、SPの内容に24ビット即値*imm24*を加えたアドレスの16ビットデータを*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

```
ext   0x1
ld    %r0,[%sp + 0x2]    ; r0 ← [sp + 0x82]
```

注　意

SPおよびディスプレースメントで指定されるメモリアドレスは、16ビット境界(最下位ビット = 0)を示していることが必要です。奇数アドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの最下位ビットを0としてデータ転送は実行されます。

ld %rd, [imm7]

機　能

16ビットデータ転送

標準）　*rd*(15:0) ← W[*imm7*], *rd*(23:16) ← 0
拡張1）　*rd*(15:0) ← W[*imm20*], *rd*(23:16) ← 0
拡張2）　*rd*(15:0) ← W[*imm24*], *rd*(23:16) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	0	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src:即値（符号なし）
Dst:レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準

```
ld   %rd,[imm7]        ; memory address = imm7
```

指定メモリの16ビットデータを*rd*レジスタに転送します。7ビット即値*imm7*がアクセスされるメモリアドレスとなります。*rd*レジスタの上位8ビットは0に設定されます。

(2)拡張1

```
ext   imm13            ; = imm20(19:7)
ld    %rd,[imm7]       ; memory address = imm20,
                       ; imm7 = imm20(6:0)
```

`ext`命令により、メモリアドレスが20ビットに拡張されます。これにより、20ビット即値*imm20*で指定されるアドレスの16ビットデータを*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。

(3)拡張2

```
ext   imm4             ; imm4(3:0) = imm24(23:20)
ext   imm13            ; = imm24(19:7)
ld    %rd,[imm7]       ; memory address = sp + imm24,
                       ; imm7 = imm24(6:0)
```

2つの`ext`命令により、メモリアドレスが24ビットに拡張されます。これにより、24ビット即値*imm24*で指定されるアドレスの16ビットデータを*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。 この場合は`ext`命令による拡張は行えません。

例

```
ext   0x1
ld    %r0,[0x2]   ; r0 ← [0x82]
```

注　意

*imm7*で指定されるメモリアドレスは、16ビット境界（最下位ビット = 0）を示していることが必要です。奇数アドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの最下位ビットを0としてデータ転送は実行されます。

ld *%rd, sign7*

機　能

16ビットデータ転送

標準）　*rd*(6:0) ← *sign7*(6:0), *rd*(15:7) ← *sign7*(6), *rd*(23:16) ← 0

拡張1）　*rd*(15:0) ← *sign16*(15:0), *rd*(23:16) ← 0

拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	*r d*			*sign7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src:即値（符号付き）

Dst:レジスタ直接　`%rd` = `%r0`〜`%r7`

CLK

1サイクル

説　明

(1)標準

```
ld   %rd,sign7          ; rd ← sign7 (sign-extended)
```

7ビット即値データ*sign7*を16ビットに符号拡張して*rd*レジスタにロードします。

(2)拡張1

```
ext  imm13              ; = sign16(15:7)
ld   %rd,sign7          ; rd ← sign16, sign7 = sign16(6:0)
```

`ext`命令で拡張した16ビット即値データ*sign16*を*rd*レジスタにロードします。

(3)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

```
ld  %r0,0x7f    ; r0 ← 0xffff (r0 = 0x00ffff)
```

ld [*%rb*], *%rs*

機　能

16ビットデータ転送

標準）　W[*rb*] ← *rs*(15:0)
拡張1）　W[*rb* + *imm13*] ← *rs*(15:0)
拡張2）　W[*rb* + *imm24*] ← *rs*(15:0)

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1	*rs*			0	0	1	0	*rb*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接　`%rs` = `%r0`～`%r7`
Dst: レジスタ間接　`%rb` = `%r0`～`%r7`

CLK

1サイクル（`ext`命令使用時は2サイクル）

説　明

(1)標準

```
ld   [%rb],%rs        ; memory address = rb
```

*rs*レジスタの下位16ビットを指定のメモリに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。

(2)拡張1

```
ext   imm13
ld    [%rb],%rs       ; memory address = rb + imm13
```

`ext`命令により、アドレッシングモードがディスプレースメント付きレジスタ間接アドレッシングに変わります。*rs*レジスタの下位16ビットを、*rb*レジスタの内容に13ビット即値*imm13*を加えたアドレスに転送します。*rb*レジスタの内容は変更されません。

(3)拡張2

```
ext   imm11           ; imm11(10:0) = imm24(23:13)
ext   imm13           ; = imm24(12:0)
ld    [%rb],%rs       ; memory address = rb + imm24
```

アドレッシングモードがディスプレースメント付きレジスタ間接アドレッシングに変わり、*rs*レジスタの下位16ビットを、*rb*レジスタの内容に24ビット即値*imm24*を加えたアドレスに転送します。*rb*レジスタの内容は変更されません。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。 この場合は`ext`命令による拡張は行えません。

注　意

*rb*レジスタおよびディスプレースメントで指定されるメモリアドレスは、16ビット境界（最下位ビット = 0）を示していることが必要です。奇数アドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの最下位ビットを0としてデータ転送は実行されます。

ld [*%rb*]+, *%rs*
ld [*%rb*]-, *%rs*
ld -[*%rb*], *%rs*

機　能　アドレスインクリメント/デクリメントオプション付き16ビットデータ転送

ld [*%rb*]+, *%rs*（ポストインクリメントオプション付き）

標準）　W[*rb*] ← *rs*(15:0), *rb*(23:0) ← *rb*(23:0) + 2
拡張1）　W[*rb*] ← *rs*(15:0), *rb*(23:0) ← *rb*(23:0) + *imm13*
拡張2）　W[*rb*] ← *rs*(15:0), *rb*(23:0) ← *rb*(23:0) + *imm24*

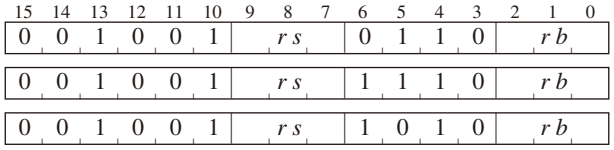
ld [*%rb*]-, *%rs*（ポストデクリメントオプション付き）

標準）　W[*rb*] ← *rs*(15:0), *rb*(23:0) ← *rb*(23:0) - 2
拡張1）　W[*rb*] ← *rs*(15:0), *rb*(23:0) ← *rb*(23:0) - *imm13*
拡張2）　W[*rb*] ← *rs*(15:0), *rb*(23:0) ← *rb*(23:0) - *imm24*

ld -[*%rb*], *%rs*（プリデクリメントオプション付き）

標準）　*rb*(23:0) ← *rb*(23:0) - 2, W[*rb*] ← *rs*(15:0)
拡張1）　*rb*(23:0) ← *rb*(23:0) - *imm13*, W[*rb*] ← *rs*(15:0)
拡張2）　*rb*(23:0) ← *rb*(23:0) - *imm24*, W[*rb*] ← *rs*(15:0)

コード



フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード　Src: レジスタ直接　`%rs` = `%r0`～`%r7`
Dst: レジスタ間接　`%rb` = `%r0`～`%r7`

CLK　2サイクル

説　明　(1)アドレスインクリメント/デクリメントオプション

`[]+`、`[]-`、`-[]`のオプション指定により、メモリアドレスが自動的にインクリメント/デクリメントされ、連続したデータ転送が簡単にプログラムできます。

`ld [%rb]+,%rs`　ポストインクリメント付きロード命令
データ転送の後、メモリアドレスがインクリメントされます。

`ld [%rb]-,%rs`　ポストデクリメント付きロード命令
データ転送の後、メモリアドレスがデクリメントされます。

`ld -[%rb],%rs`　プリデクリメント付きロード命令
データ転送の前にメモリアドレスがデクリメントされます。

インクリメント/デクリメントサイズは次のとおりです。
`ext`命令なし(標準): 2(16ビットサイズ)
`ext` 1個付き(拡張1): *imm13*
`ext` 2個付き(拡張2): *imm24*

(2)標準(ポストインクリメントの例)

```
ld  [%rb]+,%rs      ; Destination memory address = rb
                    ; post increment: rb + 2
```

*rs*レジスタの下位16ビットを指定のメモリに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。
データ転送の後、メモリアドレスが2バイト分インクリメントされます。

(3)拡張1(ポストデクリメントの例)

```
ext   imm13
ld    [%rb]-,%rs      ; Destination memory address = rb
                      ; post decrement: rb - imm13
```

*rs*レジスタの下位16ビットを指定のメモリに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。
データ転送の後、メモリアドレスが*imm13*バイト分デクリメントされます。

(4)拡張2(プリデクリメントの例)

```
ext   imm11           ; imm11(10:0) = imm24(23:13)
ext   imm13           ; = imm24(12:0)
ld    -[%rb],%rs      ; Destination memory address = rb - imm24
```

*rb*レジスタで指定されるメモリアドレスを*imm24*バイト分デクリメントした後、*rs*レジスタの下位16ビットをそのアドレスに転送します。

(5)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。 この場合はext命令による拡張は行えません。

注 意 *rb*レジスタおよび即値で指定されるメモリアドレスは、16ビット境界(最下位ビット = 0)を示していることが必要です。奇数アドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの最下位ビットを0としてデータ転送は実行されます。

ld [%sp + *imm7*], *%rs*

機　能

16ビットデータ転送

標準）　W[sp + *imm7*] ← *rs*(15:0)
拡張1）　W[sp + *imm20*] ← *rs*(15:0)
拡張2）　W[sp + *imm24*] ← *rs*(15:0)

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	*r s*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src:レジスタ直接 `%rs` = `%r0`〜`%r7`
Dst:ディスプレースメント付きレジスタ間接

CLK

2サイクル

説　明

(1)標準

```
ld   [%sp + imm7],%rs        ; memory address = sp + imm7
```

*rs*レジスタの下位16ビットを指定メモリに転送します。現在のSPの内容に7ビット即値*imm7*をディスプレースメントとして加算した値がアクセスされるメモリアドレスとなります。

(2)拡張1

```
ext   imm13                  ; = imm20(19:7)
ld    [%sp + imm7],%rs       ; memory address = sp + imm20,
                             ; imm7 = imm20(6:0)
```

`ext`命令により、ディスプレースメントが20ビットに拡張されます。これにより、*rs*レジスタの下位16ビットを、SPの内容に20ビット即値*imm20*を加えたアドレスに転送します。

(3)拡張2

```
ext   imm4                   ; imm4(3:0) = imm24(23:20)
ext   imm13                  ; = imm24(19:7)
ld    [%sp + imm7],%rs       ; memory address = sp + imm24,
                             ; imm7 = imm24(6:0)
```

2つの`ext`命令により、ディスプレースメントが24ビットに拡張されます。これにより、*rs*レジスタの下位16ビットを、SPの内容に24ビット即値*imm24*を加えたアドレスに転送します。

(4)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

```
ext   0x1
ld    [%sp + 0x2],%r0    ; W[sp + 0x82] ← r0の下位16ビット
```

注　意

SPおよびディスプレースメントで指定されるメモリアドレスは、16ビット境界(最下位ビット = 0)を示していることが必要です。奇数アドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの最下位ビットを0としてデータ転送は実行されます。

ld [*imm7*], *%rs*

機　能

16ビットデータ転送

標準）　W[*imm7*] ← *rs*(15:0)
拡張1）　W[*imm20*] ← *rs*(15:0)
拡張2）　W[*imm24*] ← *rs*(15:0)

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	1	0	*r s*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接　*%rs* = %r0～%r7
Dst:即値（符号なし）

CLK

1サイクル

説　明

(1)標準

```
ld   [imm7],%rs        ; memory address = imm7
```

*rs*レジスタの下位16ビットを指定メモリに転送します。7ビット即値*imm7*がアクセスされるメモリアドレスとなります。

(2)拡張1

```
ext   imm13            ; = imm20(19:7)
ld    [imm7],%rs       ; memory address = imm20, imm7 = imm20(6:0)
```

ext命令により、メモリアドレスが20ビットに拡張されます。これにより、*rs*レジスタの下位16ビットを20ビット即値*imm20*で指定されるアドレスに転送します。

(3)拡張2

```
ext   imm4             ; imm4(3:0) = imm24(23:20)
ext   imm13            ; = imm24(19:7)
ld    [imm7],%rs       ; memory address = imm24, imm7 = imm24(6:0)
```

2つのext命令により、メモリアドレスが24ビットに拡張されます。これにより、*rs*レジスタの下位16ビットを24ビット即値*imm24*で指定されるアドレスに転送します。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。 この場合はext命令による拡張は行えません。

例

```
ext   0x1
ld    [0x2],%r0        ; W[0x82] ← r0の下位16ビット
```

注　意

*imm7*で指定されるメモリアドレスは、16ビット境界(最下位ビット = 0)を示していることが必要です。奇数アドレスが指定されると､アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの最下位ビットを0としてデータ転送は実行されます。

ld.a *%rd*, %pc

機　能

24ビットデータ転送

標準）　*rd*(23:0) ← pc(23:0) + 2
拡張1）　不可
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	1	*rd*			0	1	1	0	0	0	0

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接 `%pc`
Dst: レジスタ直接 `%rd` = `%r0`〜`%r7`

CLK

1サイクル

説　明

PC（PC + 2）の内容を*rd*レジスタに転送します。

例

```
ld.a  %r0,%pc    ; r0 ← pc + 2
```

注　意

- 本命令を実行すると、レジスタに読み込まれる値はこの命令のPC値 + 2となります。この命令はディレイドスロット命令として使用してください。ディレイド分岐命令の直後以外の場所に記述した場合、*rd*レジスタにロードされるPC値が`ld.a`の次の命令のアドレスを示すとは限りません。
- この命令は、`jr*.d`、`jpr.d`または`jpa.d`のディレイドスロット命令として使用してください。

ld.a *%rd, %rs*

機　能

24ビットデータ転送

標準）　*rd*(23:0) ← *rs*(23:0)
拡張1）　不可
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	0	*rd*			0	0	1	1	*rs*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接　`%rs` = `%r0`～`%r7`
Dst: レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準
*rs*レジスタの内容(24ビットデータ)を*rd*レジスタに転送します。

(2)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

例

```
ld.a  %r0,%r1    ; r0 ← r1
```

ld.a *%rd*, %sp

機　能　24ビットデータ転送

標準）　*rd*(23:2) ← sp(23:2), *rd*(1:0) ← 0
拡張1）　不可
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	1	*rd*			0	0	1	0	0	0	0

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード　Src: レジスタ直接 `%sp`
Dst: レジスタ直接 `%rd` = `%r0`〜`%r7`

CLK　1サイクル

説　明　SPの内容（24ビットデータ）を*rd*レジスタに転送します。

例

```
ld.a  %r0,%sp    ; r0 ← sp
```

ld.a *%rd*, [*%rb*]

機　能

32ビットデータ転送

標準）　*rd*(23:0) ← A[*rb*](23:0), 無視 ← A[*rb*](31:24)
拡張1）　*rd*(23:0) ← A[*rb* + *imm13*](23:0), 無視 ← A[*rb* + *imm13*](31:24)
拡張2）　*rd*(23:0) ← A[*rb* + *imm24*](23:0), 無視 ← A[*rb* + *imm24*](31:24)

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	*r d*			0	0	1	1	*r b*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ間接　`%rb` = `%r0`〜`%r7`
Dst: レジスタ直接　`%rd` = `%r0`〜`%r7`

CLK

1サイクル（`ext`命令使用時は2サイクル）

説　明

(1)標準

```
ld.a   %rd,[%rb]       ; memory address = rb
```

指定メモリの32ビットデータ（上位8ビットは無視）を*rd*レジスタに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。

(2)拡張1

```
ext    imm13
ld.a   %rd,[%rb]       ; memory address = rb + imm13
```

`ext`命令により、アドレッシングモードがディスプレースメント付きレジスタ間接アドレッシングに変わります。これにより、*rb*レジスタの内容に13ビット即値*imm13*を加えたアドレスの32ビットデータ（上位8ビットは無視）を*rd*レジスタに転送します。*rb*レジスタの内容は変更されません。

(3)拡張2

```
ext    imm11           ; imm11(10:0) = imm24(23:13)
ext    imm13           ; = imm24(12:0)
ld.a   %rd,[%rb]       ; memory address = rb + imm24
```

アドレッシングモードがディスプレースメント付きレジスタ間接アドレッシングに変わり、*rb*レジスタの内容に24ビット即値*imm24*を加えたアドレスの32ビットデータ（上位8ビットは無視）を*rd*レジスタに転送します。*rb*レジスタの内容は変更されません。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。 この場合は`ext`命令による拡張は行えません。

注　意

*rb*レジスタおよびディスプレースメントで指定されるメモリアドレスは、32ビット境界（下位2ビット = 0）を示していることが必要です。それ以外のアドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの下位2ビットを0としてデータ転送は実行されます。

ld.a %rd, [%rb]+
ld.a %rd, [%rb]-
ld.a %rd, -[%rb]

機　能　アドレスインクリメント/デクリメントオプション付き32ビットデータ転送

ld.a *%rd*, [*%rb*]+（ポストインクリメントオプション付き）

標準）　*rd*(23:0) ← A[*rb*](23:0), 無視 ← A[*rb*](31:24), *rb*(23:0) ← *rb*(23:0) + 4
拡張1）　*rd*(23:0) ← A[*rb*](23:0), 無視 ← A[*rb*](31:24), *rb*(23:0) ← *rb*(23:0) + *imm13*
拡張2）　*rd*(23:0) ← A[*rb*](23:0), 無視 ← A[*rb*](31:24), *rb*(23:0) ← *rb*(23:0) + *imm24*

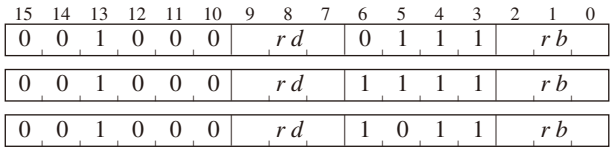
ld.a *%rd*, [*%rb*]-（ポストデクリメントオプション付き）

標準）　*rd*(23:0) ← A[*rb*](23:0), 無視 ← A[*rb*](31:24), *rb*(23:0) ← *rb*(23:0) - 4
拡張1）　*rd*(23:0) ← A[*rb*](23:0), 無視 ← A[*rb*](31:24), *rb*(23:0) ← *rb*(23:0) - *imm13*
拡張2）　*rd*(23:0) ← A[*rb*](23:0), 無視 ← A[*rb*](31:24), *rb*(23:0) ← *rb*(23:0) - *imm24*

ld.a *%rd*, -[*%rb*]（プリデクリメントオプション付き）

標準）　*rb*(23:0) ← *rb*(23:0) - 4, *rd*(23:0) ← A[*rb*](23:0), 無視 ← A[*rb*](31:24)
拡張1）　*rb*(23:0) ← *rb*(23:0) - *imm13*, *rd*(23:0) ← A[*rb*](23:0), 無視 ← A[*rb*](31:24)
拡張2）　*rb*(23:0) ← *rb*(23:0) - *imm24*, *rd*(23:0) ← A[*rb*](23:0), 無視 ← A[*rb*](31:24)

コード



フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード　Src: レジスタ間接　`%rb` = `%r0`～`%r7`
Dst: レジスタ直接　`%rd` = `%r0`～`%r7`

CLK　2サイクル

説　明　(1)アドレスインクリメント/デクリメントオプション

`[]+`、`[]-`、`-[]`のオプション指定により、メモリアドレスが自動的にインクリメント/デクリメントされ、連続したデータ転送が簡単にプログラムできます。

`ld.a %rd,[%rb]+`　ポストインクリメント付きロード命令
データ転送の後、メモリアドレスがインクリメントされます。

`ld.a %rd,[%rb]-`　ポストデクリメント付きロード命令
データ転送の後、メモリアドレスがデクリメントされます。

`ld.a %rd,-[%rb]`　プリデクリメント付きロード命令
データ転送の前にメモリアドレスがデクリメントされます。

インクリメント/デクリメントサイズは次のとおりです。
`ext`命令なし(標準): 4(32ビットサイズ)
`ext` 1個付き(拡張1): *imm13*
`ext` 2個付き(拡張2): *imm24*

(2)標準(ポストインクリメントの例)

```
ld.a  %rd,[%rb]+  ; source memory address = rb
                  ; post increment: rb + 4
```

指定メモリの32ビットデータ(上位8ビットは無視)を*rd*レジスタに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。
データ転送の後、メモリアドレスが4バイト分インクリメントされます。

(3)拡張1(ポストデクリメントの例)

```
ext    imm13
ld.a   %rd,[%rb]-  ; source memory address = rb
                   ; post decrement: rb - imm13
```

指定メモリの32ビットデータ(上位8ビットは無視)を*rd*レジスタに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。
データ転送の後、メモリアドレスが*imm13*バイト分デクリメントされます。

(4)拡張2(プリデクリメントの例)

```
ext    imm11       ; imm11(10:0) = imm24(23:13)
ext    imm13       ; = imm24(12:0)
ld.a   %rd,-[%rb]  ; source memory address = rb - imm24
```

*rb*レジスタで指定されるメモリアドレスを*imm24*バイト分デクリメントした後、そのアドレスの32ビットデータ(上位8ビットは無視)を*rd*レジスタに転送します。

(5)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。 この場合はext命令による拡張は行えません。

注 意 *rb*レジスタおよび即値で指定されるメモリアドレスは、32ビット境界(下位2ビット = 0)を示していることが必要です。それ以外のアドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの下位2ビットを0としてデータ転送は実行されます。

ld.a *%rd*, [%sp]

機　能

32ビットデータ転送

標準）　*rd*(23:0) ← A[sp](23:0), 無視 ← A[sp](31:24)
拡張1）　*rd*(23:0) ← A[sp + *imm13*](23:0), 無視 ← A[sp + *imm13*](31:24)
拡張2）　*rd*(23:0) ← A[sp + *imm24*](23:0), 無視 ← A[sp + *imm24*](31:24)

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	1	*rd*			0	0	1	1	0	0	0

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ間接 `%sp`
Dst: レジスタ直接 `%rd` = `%r0`〜`%r7`

CLK

1サイクル（`ext`命令使用時は2サイクル）

説　明

(1)標準

```
ld.a   %rd,[%sp]       ; memory address = sp
```

指定メモリの32ビットデータ（上位8ビットは無視）を*rd*レジスタに転送します。SPの内容がアクセスされるメモリアドレスとなります。

(2)拡張1

```
ext    imm13
ld.a   %rd,[%sp]       ; memory address = sp + imm13
```

`ext`命令により、アドレッシングモードがディスプレースメント付きレジスタ間接アドレッシングに変わります。これにより、SPの内容に13ビット即値*imm13*を加えたアドレスの32ビットデータ（上位8ビットは無視）を*rd*レジスタに転送します。SPの内容は変更されません。

(3)拡張2

```
ext    imm11           ; imm11(10:0) = imm24(23:13)
ext    imm13           ; = imm24(12:0)
ld.a   %rd,[%sp]       ; memory address = sp + imm24
```

アドレッシングモードがディスプレースメント付きレジスタ間接アドレッシングに変わり、SPの内容に24ビット即値*imm24*を加えたアドレスの32ビットデータ（上位8ビットは無視）を*rd*レジスタに転送します。SPの内容は変更されません。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

注　意

ディスプレースメントは、32ビット境界（下位2ビット = 0）を示していることが必要です。それ以外のアドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの下位2ビットを0としてデータ転送は実行されます。

ld.a %rd, [%sp]+
ld.a %rd, [%sp]-
ld.a %rd, -[%sp]

機 能

アドレスインクリメント/デクリメントオプション付き32ビットデータ転送

ld.a *%rd*, [%sp]+（ポストインクリメントオプション付き）

標準） *rd*(23:0) ← A[sp](23:0), 無視 ← A[sp](31:24), sp(23:0) ← sp(23:0) + 4
拡張1） *rd*(23:0) ← A[sp](23:0), 無視 ← A[sp](31:24), sp(23:0) ← sp(23:0) + *imm13*
拡張2） *rd*(23:0) ← A[sp](23:0), 無視 ← A[sp](31:24), sp(23:0) ← sp(23:0) + *imm24*

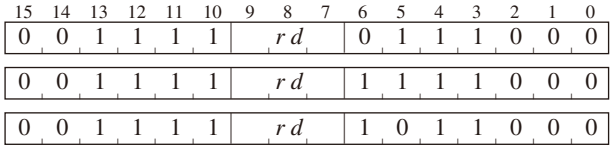
ld.a *%rd*, [%sp]-（ポストデクリメントオプション付き）

標準） *rd*(23:0) ← A[sp](23:0), 無視 ← A[sp](31:24), sp(23:0) ← sp(23:0) - 4
拡張1） *rd*(23:0) ← A[sp](23:0), 無視 ← A[sp](31:24), sp(23:0) ← sp(23:0) - *imm13*
拡張2） *rd*(23:0) ← A[sp](23:0), 無視 ← A[sp](31:24), sp(23:0) ← sp(23:0) - *imm24*

ld.a *%rd*, -[%sp]（プリデクリメントオプション付き）

標準） sp(23:0) ← sp(23:0) - 4, *rd*(23:0) ← A[sp](23:0), 無視 ← A[sp](31:24)
拡張1） sp(23:0) ← sp(23:0) - *imm13, rd*(23:0) ← A[sp](23:0), 無視 ← A[sp](31:24)
拡張2） sp(23:0) ← sp(23:0) - *imm24, rd*(23:0) ← A[sp](23:0), 無視 ← A[sp](31:24)

コード



フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ間接 `%sp`
Dst: レジスタ直接 `%rd` = `%r0`〜`%r7`

CLK

2サイクル

説 明

(1)アドレスインクリメント/デクリメントオプション

`[]+`、`[]-`、`-[]`のオプション指定により、メモリアドレスが自動的にインクリメント/デクリメントされ、連続したデータ転送が簡単にプログラムできます。

`ld.a %rd,[%sp]+` ポストインクリメント付きロード命令
データ転送の後、メモリアドレスがインクリメントされます。

`ld.a %rd,[%sp]-` ポストデクリメント付きロード命令
データ転送の後、メモリアドレスがデクリメントされます。

`ld.a %rd,-[%sp]` プリデクリメント付きロード命令
データ転送の前にメモリアドレスがデクリメントされます。

インクリメント/デクリメントサイズは次のとおりです。
`ext`命令なし(標準): 4(32ビットサイズ)
`ext` 1個付き(拡張1): *imm13*
`ext` 2個付き(拡張2): *imm24*

(2)標準(ポストインクリメントの例)

```
ld.a  %rd,[%sp]+  ; source memory address = sp
                  ; post increment: sp + 4
```

指定メモリの32ビットデータ(上位8ビットは無視)を*rd*レジスタに転送します。SPの内容がアクセスされるメモリアドレスとなります。
データ転送の後、メモリアドレスが4バイト分インクリメントされます。

(3)拡張1(ポストデクリメントの例)

```
ext    imm13
ld.a   %rd,[%sp]-  ; source memory address = sp
                   ; post decrement: sp - imm13
```

指定メモリの32ビットデータ(上位8ビットは無視)を*rd*レジスタに転送します。SPの内容がアクセスされるメモリアドレスとなります。
データ転送の後、メモリアドレスが*imm13*バイト分デクリメントされます。

(4)拡張2(プリデクリメントの例)

```
ext    imm11       ; imm11(10:0) = imm24(23:13)
ext    imm13       ; = imm24(12:0)
ld.a   %rd,-[%sp]  ; source memory address = sp - imm24
```

SPで指定されるメモリアドレスを*imm24*バイト分デクリメントした後、そのアドレスの32ビットデータ(上位8ビットは無視)を*rd*レジスタに転送します。

(5)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合はext命令による拡張は行えません。

注 意 即値は、32ビット境界(下位2ビット = 0)を示していることが必要です。それ以外のアドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの下位2ビットを0としてデータ転送は実行されます。

ld.a *%rd*, [%sp + *imm7*]

機　能

32ビットデータ転送

標準）　*rd*(23:0) ← A[sp + *imm7*](23:0), 無視 ← A[sp + *imm7*](31:24)
拡張1）　*rd*(23:0) ← A[sp + *imm20*](23:0), 無視 ← A[sp + *imm20*](31:24)
拡張2）　*rd*(23:0) ← A[sp + *imm24*](23:0), 無視 ← A[sp + *imm24*](31:24)

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src:ディスプレースメント付きレジスタ間接
Dst:レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

2サイクル

説　明

(1)標準

```
ld.a   %rd,[%sp + imm7]      ; memory address = sp + imm7
```

指定メモリの32ビットデータ(上位8ビットは無視)を*rd*レジスタに転送します。現在のSPの内容に7ビット即値*imm7*をディスプレースメントとして加算した値がアクセスされるメモリアドレスとなります。

(2)拡張1

```
ext    imm13                 ; = imm20(19:7)
ld.a   %rd,[%sp + imm7]      ; memory address = sp + imm20,
                             ; imm7 = imm20(6:0)
```

`ext`命令により、ディスプレースメントが20ビットに拡張されます。これにより、SPの内容に20ビット即値*imm20*を加えたアドレスの32ビットデータ(上位8ビットは無視)を*rd*レジスタに転送します。

(3)拡張2

```
ext    imm4                  ; imm4(3:0) = imm24(23:20)
ext    imm13                 ; = imm24(19:7)
ld.a   %rd,[%sp + imm7]      ; memory address = sp + imm24,
                             ; imm7 = imm24(6:0)
```

2つの`ext`命令により、ディスプレースメントが24ビットに拡張されます。これにより、SPの内容に24ビット即値*imm24*を加えたアドレスの32ビットデータ(上位8ビットは無視)を*rd*レジスタに転送します。

(4)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

```
ext    0x1
ld.a   %r0,[%sp + 0x4]   ; r0 ← [sp + 0x84]
```

注　意

SPおよびディスプレースメントで指定されるメモリアドレスは、32ビット境界(下位2ビット = 0)を示していることが必要です。それ以外のアドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの下位2ビットを0としてデータ転送は実行されます。

ld.a %rd, [imm7]

機　能

32ビットデータ転送

標準）　*rd*(23:0) ← A[*imm7*](23:0), 無視 ← A[*imm7*](31:24)
拡張1）　*rd*(23:0) ← A[*imm20*](23:0), 無視 ← A[*imm20*](31:24)
拡張2）　*rd*(23:0) ← A[*imm24*](23:0), 無視 ← A[*imm24*](31:24)

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src:即値（符号なし）
Dst:レジスタ直接　`%rd` = `%r0`〜`%r7`

CLK

1サイクル

説　明

(1)標準

```
ld.a   %rd,[imm7]      ; memory address = imm7
```

指定メモリの32ビットデータ（上位8ビットは無視）を*rd*レジスタに転送します。7ビット即値*imm7*がアクセスされるメモリアドレスとなります。

(2)拡張1

```
ext    imm13           ; = imm20(19:7)
ld.a   %rd,[imm7]      ; memory address = imm20, imm7 = imm20(6:0)
```

`ext`命令により、メモリアドレスが20ビットに拡張されます。これにより、20ビット即値*imm20*で指定されるアドレスの32ビットデータ（上位8ビットは無視）を*rd*レジスタに転送します。

(3)拡張2

```
ext    imm4            ; imm4(3:0) = imm24(23:20)
ext    imm13           ; = imm24(19:7)
ld.a   %rd,[imm7]      ; memory address = imm24, imm7 = imm24(6:0)
```

2つの`ext`命令により、メモリアドレスが24ビットに拡張されます。これにより、24ビット即値*imm24*で指定されるアドレスの32ビットデータ（上位8ビットは無視）を*rd*レジスタに転送します。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

```
ext    0x1
ld.a   %r0,[0x4]  ; r0 ← [0x84]
```

注　意

*imm7*で指定されるメモリアドレスは、32ビット境界（下位2ビット = 0）を示していることが必要です。それ以外のアドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの下位2ビットを0としてデータ転送は実行されます。

ld.a *%rd, imm7*

機　能

24ビットデータ転送

標準）　*rd*(6:0) ← *imm7*, *rd*(23:7) ← 0
拡張1）　*rd*(19:0) ← *imm20*, *rd*(23:20) ← 0
拡張2）　*rd*(23:0) ← *imm24*

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	1	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src:即値（符号なし）
Dst:レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準

```
ld.a   %rd,imm7        ; rd ← imm7 (ゼロ拡張)
```

7ビット即値データ*imm7*をゼロ拡張して*rd*レジスタにロードします。

(2)拡張1

```
ext    imm13           ; = sign20(19:7)
ld.a   %rd,imm7        ; rd ← imm20 (ゼロ拡張), imm7 = imm20(6:0)
```

ext命令で拡張した20ビット即値データ*imm20*を、ゼロ拡張して*rd*レジスタにロードします。

(3)拡張2

```
ext    imm4            ; imm4(3:0) = imm24(23:20)
ext    imm13           ; = imm24(19:7)
ld.a   %rd,imm7        ; rd ← imm24, imm7 = imm24(6:0)
```

ext命令で拡張した24ビット即値データ*imm24*を*rd*レジスタにロードします。

(4)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合はext命令による拡張は行えません。

例

```
ld.a   %r0,0x3f   ; r0 ← 0x00003f
```

ld.a %sp, *%rs*

機　能

24ビットデータ転送

標準）　sp(23:2) ← *rs*(23:2), sp(1:0) ← 0
拡張1）　不可
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	1	*r s*			1	0	1	0	0	0	0

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接 `%rs` = `%r0`〜`%r7`
Dst: レジスタ直接 `%sp`

CLK

1サイクル

説　明

*rs*レジスタの内容をSPに転送します。

例

```
ld.a  %sp,%r0    ; sp ← r0
```

注　意

SPへのデータ転送では、転送元データの下位2ビットは常に0として扱われます。

ld.a %sp, *imm7*

機　能

24ビットデータ転送

標準）　sp(6:2) ← *imm7*(6:2), sp(23:7) ← 0, sp(1:0) ← 0
拡張1）　sp(19:2) ← *imm20*(19:2), sp(23:20) ← 0, sp(1:0) ← 0
拡張2）　sp(23:2) ← *imm24*(23:2), sp(1:0) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	0	0	0	*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src:即値（符号なし）
Dst:レジスタ直接 %sp

CLK

1サイクル

説　明

(1)標準

```
ld.a  %sp,imm7        ; sp ← imm7 (ゼロ拡張)
```

7ビット即値データ*imm7*をゼロ拡張してSPにロードします。

(2)拡張1

```
ext   imm13           ; = sign20(19:7)
ld.a  %sp,imm7        ; sp ← imm20 (ゼロ拡張), imm7 = imm20(6:0)
```

ext命令で拡張した20ビット即値データ*imm20*を、ゼロ拡張してSPにロードします。

(3)拡張2

```
ext   imm4            ; imm4(3:0) = imm24(23:20)
ext   imm13           ; = imm24(19:7)
ld.a  %sp,imm7        ; sp ← imm24, imm7 = imm24(6:0)
```

ext命令で拡張した24ビット即値データ*imm24*をSPにロードします。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合はext命令による拡張は行えません。

例

```
ext   0x8
ld.a  %sp,0x0    ; sp ← 0x400
```

注　意

SPへのデータ転送では、転送元データの下位2ビットは常に0として扱われます。

ld.a [*%rb*], *%rs*

機　能

32ビットデータ転送

標準）　A[*rb*](23:0) ← *rs*(23:0), A[*rb*](31:24) ← 0
拡張1）　A[*rb* + *imm13*](23:0) ← *rs*(23:0), A[*rb* + *imm13*](31:24) ← 0
拡張2）　A[*rb* + *imm24*](23:0) ← *rs*(23:0), A[*rb* + *imm24*](31:24) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1	*r s*			0	0	1	1	*r b*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src:レジスタ直接 `%rs` = `%r0`〜`%r7`
Dst:レジスタ間接 `%rb` = `%r0`〜`%r7`

CLK

1サイクル(`ext`命令使用時は2サイクル)

説　明

(1)標準

```
ld.a   [%rb],%rs       ; memory address = rb
```

*rs*レジスタの内容(24ビットデータ)を指定のメモリに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。メモリには上位8ビットを0とした32ビットデータが書き込まれます。

(2)拡張1

```
ext    imm13
ld.a   [%rb],%rs       ; memory address = rb + imm13
```

`ext`命令により、アドレッシングモードがディスプレースメント付きレジスタ間接アドレッシングに変わります。*rs*レジスタの内容を、*rb*レジスタの内容に13ビット即値*imm13*を加えたアドレスに転送します。*rb*レジスタの内容は変更されません。

(3)拡張2

```
ext    imm11           ; imm11(10:0) = imm24(23:13)
ext    imm13           ; = imm24(12:0)
ld.a   [%rb],%rs       ; memory address = rb + imm24
```

アドレッシングモードがディスプレースメント付きレジスタ間接アドレッシングに変わり、*rs*レジスタの内容を、*rb*レジスタの内容に24ビット即値*imm24*を加えたアドレスに転送します。*rb*レジスタの内容は変更されません。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

注　意

*rb*レジスタおよびディスプレースメントで指定されるメモリアドレスは、32ビット境界(下位2ビット = 0)を示していることが必要です。それ以外のアドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの下位2ビットを0としてデータ転送は実行されます。

ld.a [*%rb*]+, *%rs*
ld.a [*%rb*]-, *%rs*
ld.a -[*%rb*], *%rs*

機 能

アドレスインクリメント/デクリメントオプション付き32ビットデータ転送

ld.a [*%rb*]+, *%rs*（ポストインクリメントオプション付き）

標準） A[*rb*](23:0) ← *rs*(23:0), A[*rb*](31:24) ← 0, *rb*(23:0) ← *rb*(23:0) + 4
拡張1） A[*rb*](23:0) ← *rs*(23:0), A[*rb*](31:24) ← 0, *rb*(23:0) ← *rb*(23:0) + *imm13*
拡張2） A[*rb*](23:0) ← *rs*(23:0), A[*rb*](31:24) ← 0, *rb*(23:0) ← *rb*(23:0) + *imm24*

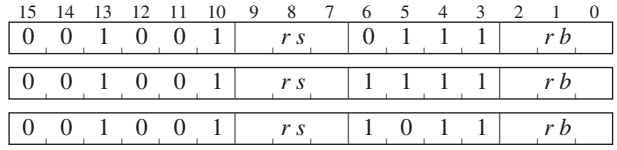
ld.a [*%rb*]-, *%rs*（ポストデクリメントオプション付き）

標準） A[*rb*](23:0) ← *rs*(23:0), A[*rb*](31:24) ← 0, *rb*(23:0) ← *rb*(23:0) - 4
拡張1） A[*rb*](23:0) ← *rs*(23:0), A[*rb*](31:24) ← 0, *rb*(23:0) ← *rb*(23:0) - *imm13*
拡張2） A[*rb*](23:0) ← *rs*(23:0), A[*rb*](31:24) ← 0, *rb*(23:0) ← *rb*(23:0) - *imm24*

ld.a -[*%rb*], *%rs*（プリデクリメントオプション付き）

標準） *rb*(23:0) ← *rb*(23:0) - 4, A[*rb*](23:0) ← *rs*(23:0), A[*rb*](31:24) ← 0
拡張1） *rb*(23:0) ← *rb*(23:0) - *imm13,* A[*rb*](23:0) ← *rs*(23:0), A[*rb*](31:24) ← 0
拡張2） *rb*(23:0) ← *rb*(23:0) - *imm24,* A[*rb*](23:0) ← *rs*(23:0), A[*rb*](31:24) ← 0

コード




```
ld.a   [%rb]+,%rs
ld.a   [%rb]-,%rs
ld.a   -[%rb],%rs
```

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接 *%rs* = %r0～%r7
Dst: レジスタ間接 *%rb* = %r0～%r7

CLK

2サイクル

説 明

(1)アドレスインクリメント/デクリメントオプション

[]+、[]-、-[]のオプション指定により、メモリアドレスが自動的にインクリメント/デクリメントされ、連続したデータ転送が簡単にプログラムできます。

`ld.a [%rb]+,%rs` ポストインクリメント付きロード命令
データ転送の後、メモリアドレスがインクリメントされます。

`ld.a [%rb]-,%rs` ポストデクリメント付きロード命令
データ転送の後、メモリアドレスがデクリメントされます。

`ld.a -[%rb],%rs` プリデクリメント付きロード命令
データ転送の前にメモリアドレスがデクリメントされます。

インクリメント/デクリメントサイズは次のとおりです。
ext命令なし(標準): 4(32ビットサイズ)
ext 1個付き(拡張1): *imm13*
ext 2個付き(拡張2): *imm24*

(2)標準(ポストインクリメントの例)

```
ld.a   [%rb]+,%rs     ; Destination memory address = rb
                      ; post increment: rb + 4
```

*rs*レジスタの内容(24ビットデータ)を指定のメモリに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。メモリには上位8ビットを0とした32ビットデータが書き込まれます。
データ転送の後、メモリアドレスが4バイト分インクリメントされます。

(3)拡張1(ポストデクリメントの例)

```
ext    imm13
ld.a   [%rb],%rs      ; Destination memory address = rb
                      ; post decrement: rb - imm13
```

*rs*レジスタの内容(24ビットデータ)を指定のメモリに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。メモリには上位8ビットを0とした32ビットデータが書き込まれます。
データ転送の後、メモリアドレスが*imm13*バイト分デクリメントされます。

(4)拡張2(プリデクリメントの例)

```
ext    imm11          ; imm11(10:0) = imm24(23:13)
ext    imm13          ; = imm24(12:0)
ld.a   [%rb],%rs      ; Destination memory address = rb - imm24
```

*rb*レジスタで指定されるメモリアドレスを*imm24*バイト分デクリメントした後、*rs*レジスタの24ビットデータをそのアドレスに転送します。メモリには上位8ビットを0とした32ビットデータが書き込まれます。

(5)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合はext命令による拡張は行えません。

注 意 *rb*レジスタおよび即値で指定されるメモリアドレスは、32ビット境界(下位2ビット = 0)を示していることが必要です。それ以外のアドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの下位2ビットを0としてデータ転送は実行されます。

ld.a [%sp], *%rs*

機　能

32ビットデータ転送

標準）　A[sp](23:0) ← *rs*(23:0), A[sp](31:24) ← 0
拡張1）　A[sp + *imm13*](23:0) ← *rs*(23:0), A[sp + *imm13*](31:24) ← 0
拡張2）　A[sp + *imm24*](23:0) ← *rs*(23:0), A[sp + *imm24*](31:24) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	1	*r s*			0	0	1	1	1	0	0

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接　`%rs` = `%r0`〜`%r7`
Dst: レジスタ間接　`%sp`

CLK

1サイクル（`ext`命令使用時は2サイクル）

説　明

(1)標準

```
ld.a   [%sp],%rs      ; memory address = sp
```

*rs*レジスタの内容(24ビットデータ)を指定のメモリに転送します。SPの内容がアクセスされるメモリアドレスとなります。メモリには上位8ビットを0とした32ビットデータが書き込まれます。

(2)拡張1

```
ext    imm13
ld.a   [%sp],%rs      ; memory address = sp + imm13
```

`ext`命令により、アドレッシングモードがディスプレースメント付きレジスタ間接アドレッシングに変わります。*rs*レジスタの内容を、SPの内容に13ビット即値*imm13*を加えたアドレスに転送します。SPの内容は変更されません。

(3)拡張2

```
ext    imm11          ; imm11(10:0) = imm24(23:13)
ext    imm13          ; = imm24(12:0)
ld.a   [%sp],%rs      ; memory address = sp + imm24
```

`ext`命令により、アドレッシングモードがディスプレースメント付きレジスタ間接アドレッシングに変わります。*rs*レジスタの内容を、SPの内容に24ビット即値*imm24*を加えたアドレスに転送します。SPの内容は変更されません。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。 この場合は`ext`命令による拡張は行えません。

注　意

SPおよびディスプレースメントは、32ビット境界(下位2ビット = 0)を示していることが必要です。それ以外のアドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの下位2ビットを0としてデータ転送は実行されます。

ld.a [%sp]+, *%rs*
ld.a [%sp]-, *%rs*
ld.a -[%sp], *%rs*

機　能　アドレスインクリメント/デクリメントオプション付き32ビットデータ転送

ld.a [%sp]+, *%rs*（ポストインクリメントオプション付き）

標準）　A[sp](23:0) ← *rs*(23:0), A[sp](31:24) ← 0, sp(23:0) ← sp(23:0) + 4
拡張1）　A[sp](23:0) ← *rs*(23:0), A[sp](31:24) ← 0, sp(23:0) ← sp(23:0) + *imm13*
拡張2）　A[sp](23:0) ← *rs*(23:0), A[sp](31:24) ← 0, sp(23:0) ← sp(23:0) + *imm24*

ld.a [%sp]-, *%rs*（ポストデクリメントオプション付き）

標準）　A[sp](23:0) ← *rs*(23:0), A[sp](31:24) ← 0, sp(23:0) ← sp(23:0) - 4
拡張1）　A[sp](23:0) ← *rs*(23:0), A[sp](31:24) ← 0, sp(23:0) ← sp(23:0) - *imm13*
拡張2）　A[sp](23:0) ← *rs*(23:0), A[sp](31:24) ← 0, sp(23:0) ← sp(23:0) - *imm24*

ld.a -[%sp], *%rs*（プリデクリメントオプション付き）

標準）　sp(23:0) ← sp(23:0) - 4, A[sp](23:0) ← *rs*(23:0), A[sp](31:24) ← 0
拡張1）　sp(23:0) ← sp(23:0) - *imm13*, A[sp](23:0) ← *rs*(23:0), A[sp](31:24) ← 0
拡張2）　sp(23:0) ← sp(23:0) - *imm24*, A[sp](23:0) ← *rs*(23:0), A[sp](31:24) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	1	1	1	*rs*			0	1	1	1	1	0	0	`ld.a [%sp]+,%rs`
0	0	1	1	1	1	*rs*			1	1	1	1	1	0	0	`ld.a [%sp]-,%rs`
0	0	1	1	1	1	*rs*			1	0	1	1	1	0	0	`ld.a -[%sp],%rs`

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード　Src: レジスタ直接　`%rs` = `%r0`～`%r7`
Dst: レジスタ間接　`%sp`

CLK　2サイクル

説　明　(1)アドレスインクリメント/デクリメントオプション

`[]+`、`[]-`、`-[]`のオプション指定により、メモリアドレスが自動的にインクリメント/デクリメントされ、連続したデータ転送が簡単にプログラムできます。

`ld.a [%sp]+,%rs`　ポストインクリメント付きロード命令
データ転送の後、メモリアドレスがインクリメントされます。

`ld.a [%sp]-,%rs`　ポストデクリメント付きロード命令
データ転送の後、メモリアドレスがデクリメントされます。

`ld.a -[%sp],%rs`　プリデクリメント付きロード命令
データ転送の前にメモリアドレスがデクリメントされます。

インクリメント/デクリメントサイズは次のとおりです。
`ext`命令なし(標準): 4(32ビットサイズ)
`ext` 1個付き(拡張1): *imm13*
`ext` 2個付き(拡張2): *imm24*

(2)標準(ポストインクリメントの例)

```
ld.a  [%sp]+,%rs     ; Destination memory address = sp
                     ; post increment: sp + 4
```

*rs*レジスタの内容(24ビットデータ)を指定のメモリに転送します。SPの内容がアクセスされるメモリアドレスとなります。メモリには上位8ビットを0とした32ビットデータが書き込まれます。
データ転送の後、メモリアドレスが4バイト分インクリメントされます。

(3)拡張1(ポストデクリメントの例)

```
ext    imm13
ld.a   [%sp]-,%rs      ; Destination memory address = sp
                       ; post decrement: sp - imm13
```

*rs*レジスタの内容(24ビットデータ)を指定のメモリに転送します。SPの内容がアクセスされるメモリアドレスとなります。メモリには上位8ビットを0とした32ビットデータが書き込まれます。
データ転送の後、メモリアドレスが*imm13*バイト分デクリメントされます。

(4)拡張2(プリデクリメントの例)

```
ext    imm11           ; imm11(10:0) = imm24(23:13)
ext    imm13           ; = imm24(12:0)
ld.a   -[%sp],%rs      ; Destination memory address = sp - imm24
```

SPで指定されるメモリアドレスを*imm24*バイト分デクリメントした後、*rs*レジスタの24ビットデータをそのアドレスに転送します。メモリには上位8ビットを0とした32ビットデータが書き込まれます。

(5)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。 この場合はext命令による拡張は行えません。

注 意 SPおよび即値は、32ビット境界(下位2ビット = 0)を示していることが必要です。それ以外のアドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの下位2ビットを0としてデータ転送は実行されます。

ld.a [%sp + *imm7*], *%rs*

機　能

32ビットデータ転送

標準）　A[sp + *imm7*](23:0) ← *rs*(23:0), A[sp + *imm7*](31:24) ← 0
拡張1）　A[sp + *imm20*](23:0) ← *rs*(23:0), A[sp + *imm20*](31:24) ← 0
拡張2）　A[sp + *imm24*](23:0) ← *rs*(23:0), A[sp + *imm24*](31:24) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	*r s*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接　*%rs* = %r0～%r7
Dst: ディスプレースメント付きレジスタ間接

CLK

2サイクル

説　明

(1)標準

```
ld.a  [%sp + imm7],%rs      ; memory address = sp + imm7
```

*rs*レジスタの内容を指定メモリに転送します。現在のSPの内容に7ビット即値*imm7*をディスプレースメントとして加算した値がアクセスされるメモリアドレスとなります。メモリには上位8ビットを0とした32ビットデータが書き込まれます。

(2)拡張1

```
ext   imm13                 ; = imm20(19:7)
ld.a  [%sp + imm7],%rs      ; memory address = sp + imm20,
                            ; imm7 = imm20(6:0)
```

ext命令により、ディスプレースメントが20ビットに拡張されます。これにより、*rs*レジスタの内容を、SPの内容に20ビット即値*imm20*を加えたアドレスに転送します。

(3)拡張2

```
ext   imm4                  ; imm4(3:0) = imm24(23:20)
ext   imm13                 ; = imm24(19:7)
ld.a  [%sp + imm7],%rs      ; memory address = sp + imm24,
                            ; imm7 = imm24(6:0)
```

2つのext命令により、ディスプレースメントが24ビットに拡張されます。これにより、*rs*レジスタの内容を、SPの内容に24ビット即値*imm24*を加えたアドレスに転送します。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合はext命令による拡張は行えません。

例

```
ext   0x1
ld.a  [%sp + 0x4],%r0  ; [sp + 0x84] ← r0
```

注　意

SPおよびディスプレースメントで指定されるメモリアドレスは、32ビット境界(下位2ビット = 0)を示していることが必要です。それ以外のアドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの下位2ビットを0としてデータ転送は実行されます。

ld.a [*imm7*], *%rs*

機　能

32ビットデータ転送

標準）　A[*imm7*](23:0) ← *rs*(23:0), A[*imm7*](31:24) ← 0
拡張1）　A[*imm20*](23:0) ← *rs*(23:0), A[*imm20*](31:24) ← 0
拡張2）　A[*imm24*](23:0) ← *rs*(23:0), A[*imm24*](31:24) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	1	1	*r s*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接　*%rs* = %r0～%r7
Dst: 即値（符号なし）

CLK

1サイクル

説　明

(1)標準

```
ld.a  [imm7],%rs     ; memory address = imm7
```

*rs*レジスタの内容を指定メモリに転送します。7ビット即値*imm7*がアクセスされるメモリアドレスとなります。メモリには上位8ビットを0とした32ビットデータが書き込まれます。

(2)拡張1

```
ext   imm13          ; = imm20(19:7)
ld.a  [imm7],%rs     ; memory address = imm20, imm7 = imm20(6:0)
```

ext命令により、メモリアドレスが20ビットに拡張されます。これにより、*rs*レジスタの内容を20ビット即値*imm20*で指定されるアドレスに転送します。

(3)拡張2

```
ext   imm4           ; imm4(3:0) = imm24(23:20)
ext   imm13          ; = imm24(19:7)
ld.a  [imm7],%rs     ; memory address = imm24, imm7 = imm24(6:0)
```

2つのext命令により、メモリアドレスが24ビットに拡張されます。これにより、*rs*レジスタの内容を24ビット即値*imm24*で指定されるアドレスに転送します。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合はext命令による拡張は行えません。

例

```
ext   0x1
ld.a  [0x4],%r0        ; [0x84] ← r0
```

注　意

*imm7*で指定されるメモリアドレスは、32ビット境界（下位2ビット = 0）を示していることが必要です。それ以外のアドレスが指定されると、アドレス不整割り込みが発生します。ただし、この場合でも、アドレスの下位2ビットを0としてデータ転送は実行されます。

ld.b *%rd, %rs*

機　能

符号付きバイトデータ転送

標準）　*rd*(7:0) ← *rs*(7:0), *rd*(15:8) ← *rs*(7), *rd*(23:16) ← 0
拡張1）　不可
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	0	*rd*			0	0	0	0	*rs*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接　`%rs` = `%r0`〜`%r7`
Dst: レジスタ直接　`%rd` = `%r0`〜`%r7`

CLK

1サイクル

説　明

(1)標準
*rs*レジスタの下位8ビットを16ビットに符号拡張して*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。

(2)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

例

```
ld.b  %r0,%r1    ; r0 ← r1(7:0)を符号拡張
```

ld.b %rd, [%rb]

機　能

符号付きバイトデータ転送

標準）　*rd*(7:0) ← B[*rb*], *rd*(15:8) ← B[*rb*](7), *rd*(23:16) ← 0
拡張1）　*rd*(7:0) ← B[*rb* + *imm13*], *rd*(15:8) ← B[*rb* + *imm13*](7), *rd*(24:16) ← 0
拡張2）　*rd*(7:0) ← B[*rb* + *imm24*], *rd*(15:8) ← B[*rb* + *imm24*](7), *rd*(24:16) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	*r d*			0	0	0	0	*r b*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ間接　`%rb` = `%r0`〜`%r7`
Dst: レジスタ直接　`%rd` = `%r0`〜`%r7`

CLK

1サイクル（`ext`命令使用時は2サイクル）

説　明

(1)標準

```
ld.b   %rd,[%rb]       ; memory address = rb
```

指定メモリのバイトデータを16ビットに符号拡張して*rd*レジスタに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。*rd*レジスタの上位8ビットは0に設定されます。

(2)拡張1

```
ext    imm13
ld.b   %rd,[%rb]       ; memory address = rb + imm13
```

`ext`命令により、アドレッシングモードがディスプレースメント付きレジスタ間接アドレッシングに変わります。これにより、*rb*レジスタの内容に13ビット即値*imm13*を加えたアドレスのバイトデータを16ビットに符号拡張して*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。*rb*レジスタの内容は変更されません。

(3)拡張2

```
ext    imm11           ; imm11(10:0) = imm24(23:13)
ext    imm13           ; = imm24(12:0)
ld.b   %rd,[%rb]       ; memory address = rb + imm24
```

アドレッシングモードがディスプレースメント付きレジスタ間接アドレッシングに変わり、*rb*レジスタの内容に24ビット即値*imm24*を加えたアドレスのバイトドデータを16ビットに符号拡張して*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。*rb*レジスタの内容は変更されません。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。 この場合は`ext`命令による拡張は行えません。

ld.b %rd, [%rb]+
ld.b %rd, [%rb]-
ld.b %rd, -[%rb]

機 能 符号付きバイトデータ転送(アドレスインクリメント/デクリメントオプション付き)

ld.b *%rd*, [*%rb*]+(ポストインクリメントオプション付き)

標準) *rd*(7:0) ← B[*rb*], *rd*(15:8) ← B[*rb*](7), *rd*(23:16) ← 0, *rb*(23:0) ← *rb*(23:0) + 1
拡張1) *rd*(7:0) ← B[*rb*], *rd*(15:8) ← B[*rb*](7), *rd*(24:16) ← 0, *rb*(23:0) ← *rb*(23:0) + *imm13*
拡張2) *rd*(7:0) ← B[*rb*], *rd*(15:8) ← B[*rb*](7), *rd*(24:16) ← 0, *rb*(23:0) ← *rb*(23:0) + *imm24*

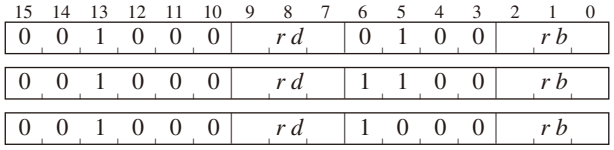
ld.b *%rd*, [*%rb*]-(ポストデクリメントオプション付き)

標準) *rd*(7:0) ← B[*rb*], *rd*(15:8) ← B[*rb*](7), *rd*(23:16) ← 0, *rb*(23:0) ← *rb*(23:0) - 1
拡張1) *rd*(7:0) ← B[*rb*], *rd*(15:8) ← B[*rb*](7), *rd*(24:16) ← 0, *rb*(23:0) ← *rb*(23:0) - *imm13*
拡張2) *rd*(7:0) ← B[*rb*], *rd*(15:8) ← B[*rb*](7), *rd*(24:16) ← 0, *rb*(23:0) ← *rb*(23:0) - *imm24*

ld.b *%rd*, -[*%rb*](プリデクリメントオプション付き)

標準) *rb*(23:0) ← *rb*(23:0) - 1, *rd*(7:0) ← B[*rb*], *rd*(15:8) ← B[*rb*](7), *rd*(23:16) ← 0
拡張1) *rb*(23:0) ← *rb*(23:0) - *imm13*, *rd*(7:0) ← B[*rb*], *rd*(15:8) ← B[*rb*](7), *rd*(24:16) ← 0
拡張2) *rb*(23:0) ← *rb*(23:0) - *imm24*, *rd*(7:0) ← B[*rb*], *rd*(15:8) ← B[*rb*](7), *rd*(24:16) ← 0

コード



フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード Src:レジスタ間接 %rb = %r0〜%r7
Dst:レジスタ直接 %rd = %r0〜%r7

CLK 2サイクル

説 明 (1)アドレスインクリメント/デクリメントオプション

[]+、[]-、-[]のオプション指定により、メモリアドレスが自動的にインクリメント/デクリメントされ、連続したデータ転送が簡単にプログラムできます。

`ld.b %rd,[%rb]+` ポストインクリメント付きロード命令
データ転送の後、メモリアドレスがインクリメントされます。

`ld.b %rd,[%rb]-` ポストデクリメント付きロード命令
データ転送の後、メモリアドレスがデクリメントされます。

`ld.b %rd,-[%rb]` プリデクリメント付きロード命令
データ転送の前にメモリアドレスがデクリメントされます。

インクリメント/デクリメントサイズは次のとおりです。
ext命令なし(標準): 1(バイトサイズ)
ext 1個付き(拡張1): *imm13*
ext 2個付き(拡張2): *imm24*

(2)標準(ポストインクリメントの例)

```
ld.b  %rd,[%rb]+  ; source memory address = rb
                  ; post increment: rb + 1
```

指定メモリのバイトデータを16ビットに符号拡張して*rd*レジスタに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。*rd*レジスタの上位8ビットは0に設定されます。
データ転送の後、メモリアドレスが1バイト分インクリメントされます。

(3)拡張1(ポストデクリメントの例)

```
ext    imm13
ld.b   %rd,[%rb]-  ; source memory address = rb
                   ; post decrement: rb - imm13
```

指定メモリのバイトデータを16ビットに符号拡張して*rd*レジスタに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。*rd*レジスタの上位8ビットは0に設定されます。
データ転送の後、メモリアドレスが*imm13*バイト分デクリメントされます。

(4)拡張2(プリデクリメントの例)

```
ext    imm11       ; imm11(10:0) = imm24(23:13)
ext    imm13       ; = imm24(12:0)
ld.b   %rd,-[%rb]  ; source memory address = rb - imm24
```

*rb*レジスタで指定されるメモリアドレスを*imm24*バイト分デクリメントした後、そのアドレスのバイトデータを16ビットに符号拡張して*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。

(5)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。 この場合はext命令による拡張は行えません。

ld.b *%rd*, [%sp + *imm7*]

機　能

符号付きバイトデータ転送

標準）　*rd*(7:0) ← B[sp + *imm7*], *rd*(15:8) ← B[sp + *imm7*](7), *rd*(23:16) ← 0
拡張1）　*rd*(7:0) ← B[sp + *imm20*], *rd*(15:8) ← B[sp + *imm20*](7), *rd*(23:16) ← 0
拡張2）　*rd*(7:0) ← B[sp + *imm24*], *rd*(15:8) ← B[sp + *imm24*](7), *rd*(23:16) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: ディスプレースメント付きレジスタ間接
Dst: レジスタ直接　`%rd` = `%r0`〜`%r7`

CLK

2サイクル

説　明

(1)標準

```
ld.b   %rd,[%sp + imm7]      ; memory address = sp + imm7
```

指定メモリのバイトデータを16ビットに符号拡張して*rd*レジスタに転送します。現在のSPの内容に7ビット即値*imm7*をディスプレースメントとして加算した値がアクセスされるメモリアドレスとなります。*rd*レジスタの上位8ビットは0に設定されます。

(2)拡張1

```
ext    imm13                 ; = imm20(19:7)
ld.b   %rd,[%sp + imm7]      ; memory address = sp + imm20,
                             ; imm7 = imm20(6:0)
```

`ext`命令により、ディスプレースメントが20ビットに拡張されます。これにより、SPの内容に20ビット即値*imm20*を加えたアドレスのバイトデータを16ビットに符号拡張して*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。

(3)拡張2

```
ext    imm4                  ; imm4(3:0)= imm24(23:20)
ext    imm13                 ; = imm24(19:7)
ld.b   %rd,[%sp + imm7]      ; memory address = sp + imm24,
                             ; imm7 ← imm24(6:0)
```

2つの`ext`命令により、ディスプレースメントが24ビットに拡張されます。これにより、SPの内容に24ビット即値*imm24*を加えたアドレスのバイトデータを16ビットに符号拡張して*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

```
ext    0x1
ld.b   %r0,[%sp + 0x1]   ; r0 ← [sp + 0x81]を符号拡張
```

ld.b %rd, [imm7]

機　能

符号付きバイトデータ転送

標準）　*rd*(7:0) ← B[*imm7*], *rd*(15:8) ← B[*imm7*](7), *rd*(23:16) ← 0
拡張1）　*rd*(7:0) ← B[*imm20*], *rd*(15:8) ← B[*imm20*](7), *rd*(23:16) ← 0
拡張2）　*rd*(7:0) ← B[*imm24*], *rd*(15:8) ← B[*imm24*](7), *rd*(23:16) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src:即値（符号なし）
Dst:レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準

```
ld.b   %rd,[imm7]      ; memory address = imm7
```

指定メモリのバイトデータを16ビットに符号拡張して*rd*レジスタに転送します。7ビット即値*imm7*がアクセスされるメモリアドレスとなります。*rd*レジスタの上位8ビットは0に設定されます。

(2)拡張1

```
ext    imm13           ; = imm20(19:7)
ld.b   %rd,[imm7]      ; memory address = imm20, imm7 = imm20(6:0)
```

ext命令により、メモリアドレスが20ビットに拡張されます。これにより、20ビット即値*imm20*で指定されるアドレスのバイトデータを16ビットに符号拡張して*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。

(3)拡張2

```
ext    imm4            ; imm4(3:0)= imm24(23:20)
ext    imm13           ; = imm24(19:7)
ld.b   %rd,[imm7]      ; memory address = imm24, imm7 ← imm24(6:0)
```

2つのext命令により、メモリアドレスが24ビットに拡張されます。これにより、24ビット即値*imm24*で指定されるアドレスのバイトデータを16ビットに符号拡張して*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。 この場合はext命令による拡張は行えません。

例

```
ext    0x1
ld.b   %r0,[0x1]  ; r0 ← [0x81]を符号拡張
```

ld.b [*%rb*], *%rs*

機　能

符号付きバイトデータ転送

標準）　B[*rb*] ← *rs*(7:0)
拡張1）　B[*rb* + *imm13*] ← *rs*(7:0)
拡張2）　B[*rb* + *imm24*] ← *rs*(7:0)

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1	*rs*			0	0	0	0	*rb*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接　`%rs` = `%r0`〜`%r7`
Dst: レジスタ間接　`%rb` = `%r0`〜`%r7`

CLK

1サイクル（`ext`命令使用時は2サイクル）

説　明

(1)標準

```
ld.b  [%rb],%rs      ; memory address = rb
```

*rs*レジスタの下位8ビットを指定のメモリに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。

(2)拡張1

```
ext   imm13
ld.b  [%rb],%rs      ; memory address = rb + imm13
```

`ext`命令により、アドレッシングモードがディスプレースメント付きレジスタ間接アドレッシングに変わります。*rs*レジスタの下位8ビットを、*rb*レジスタの内容に13ビット即値*imm13*を加えたアドレスに転送します。*rb*レジスタの内容は変更されません。

(3)拡張2

```
ext   imm11          ; imm11(10:0) = imm24(23:13)
ext   imm13          ; = imm24(12:0)
ld.b  [%rb],%rs      ; memory address = rb + imm24
```

アドレッシングモードがディスプレースメント付きレジスタ間接アドレッシングに変わり、*rs*レジスタの下位8ビットを、*rb*レジスタの内容に24ビット即値*imm24*を加えたアドレスに転送します。*rb*レジスタの内容は変更されません。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。 この場合は`ext`命令による拡張は行えません。

ld.b [*%rb*]+, *%rs*
ld.b [*%rb*]-, *%rs*
ld.b -[*%rb*], *%rs*

機 能

符号付きバイトデータ転送(アドレスインクリメント/デクリメントオプション付き)

ld.b [*%rb*]+, *%rs*(ポストインクリメントオプション付き)

標準) B[*rb*] ← *rs*(7:0), *rb*(23:0) ← *rb*(23:0) + 1
拡張1) B[*rb*] ← *rs*(7:0), *rb*(23:0) ← *rb*(23:0) + *imm13*
拡張2) B[*rb*] ← *rs*(7:0), *rb*(23:0) ← *rb*(23:0) + *imm24*

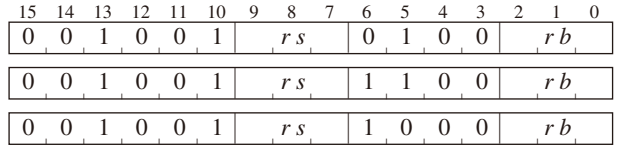
ld.b [*%rb*]-, *%rs*(ポストデクリメントオプション付き)

標準) B[*rb*] ← *rs*(7:0), *rb*(23:0) ← *rb*(23:0) - 1
拡張1) B[*rb*] ← *rs*(7:0), *rb*(23:0) ← *rb*(23:0) - *imm13*
拡張2) B[*rb*] ← *rs*(7:0), *rb*(23:0) ← *rb*(23:0) - *imm24*

ld.b -[*%rb*], *%rs*(プリデクリメントオプション付き)

標準) *rb*(23:0) ← *rb*(23:0) - 1, B[*rb*] ← *rs*(7:0)
拡張1) *rb*(23:0) ← *rb*(23:0) - *imm13*, B[*rb*] ← *rs*(7:0)
拡張2) *rb*(23:0) ← *rb*(23:0) - *imm24*, B[*rb*] ← *rs*(7:0)

コード



フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src:レジスタ直接 `%rs` = `%r0`〜`%r7`
Dst:レジスタ間接 `%rb` = `%r0`〜`%r7`

CLK

2サイクル

説 明

(1)アドレスインクリメント/デクリメントオプション

`[]+`、`[]-`、`-[]`のオプション指定により、メモリアドレスが自動的にインクリメント/デクリメントされ、連続したデータ転送が簡単にプログラムできます。

`ld.b [%rb]+,%rs` ポストインクリメント付きロード命令
データ転送の後、メモリアドレスがインクリメントされます。

`ld.b [%rb]-,%rs` ポストデクリメント付きロード命令
データ転送の後、メモリアドレスがデクリメントされます。

`ld.b -[%rb],%rs` プリデクリメント付きロード命令
データ転送の前にメモリアドレスがデクリメントされます。

インクリメント/デクリメントサイズは次のとおりです。
`ext`命令なし(標準): 1(バイトサイズ)
`ext` 1個付き(拡張1): *imm13*
`ext` 2個付き(拡張2): *imm24*

(2)標準(ポストインクリメントの例)

```
ld.b  [%rb]+,%rs    ; Destination memory address = rb
                    ; post increment: rb + 1
```

*rs*レジスタの下位8ビットを指定のメモリに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。
データ転送の後、メモリアドレスが1バイト分インクリメントされます。

(3)拡張1(ポストデクリメントの例)

```
ext    imm13
ld.b   [%rb]-,%rs      ; Destination memory address = rb
                       ; post decrement: rb - imm13
```

*rs*レジスタの下位8ビットを指定のメモリに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。
データ転送の後、メモリアドレスが*imm13*バイト分デクリメントされます。

(4)拡張2(プリデクリメントの例)

```
ext    imm11           ; imm11(10:0) = imm24(23:13)
ext    imm13           ; = imm24(12:0)
ld.b   -[%rb],%rs      ; Destination memory address = rb - imm24
```

*rb*レジスタで指定されるメモリアドレスを*imm24*バイト分デクリメントした後、*rs*レジスタの下位8ビットをそのアドレスに転送します。

(5)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。 この場合はext命令による拡張は行えません。

ld.b [%sp + *imm7*], *%rs*

機　能

符号付きバイトデータ転送

標準）　B[sp + *imm7*] ← *rs*(7:0)
拡張1）　B[sp + *imm20*] ← *rs*(7:0)
拡張2）　B[sp + *imm24*] ← *rs*(7:0)

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	*r s*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接　`%rs` = `%r0`～`%r7`
Dst: ディスプレースメント付きレジスタ間接

CLK

2サイクル

説　明

(1)標準

```
ld.b  [%sp + imm7],%rs      ; memory address = sp + imm7
```

*rs*レジスタの下位8ビットを指定メモリに転送します。現在のSPの内容に7ビット即値*imm7*をディスプレースメントとして加算した値がアクセスされるメモリアドレスとなります。

(2)拡張1

```
ext   imm13                 ; = imm20(19:7)
ld.b  [%sp + imm7],%rs      ; memory address = sp + imm20,
                            ; imm7 = imm20(6:0)
```

ext命令により、ディスプレースメントが20ビットに拡張されます。これにより、*rs*レジスタの下位8ビットを、SPの内容に20ビット即値*imm20*を加えたアドレスに転送します。

(3)拡張2

```
ext   imm4                  ; imm4(3:0) = imm24(23:20)
ext   imm13                 ; = imm24(19:7)
ld.b  [%sp + imm7],%rs      ; memory address = sp + imm24,
                            ; imm7 = imm24(6:0)
```

2つのext命令により、ディスプレースメントが24ビットに拡張されます。これにより、*rs*レジスタの下位8ビットを、SPの内容に24ビット即値*imm24*を加えたアドレスに転送します。

(4)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合はext命令による拡張は行えません。

例

```
ext   0x1
ld.b  [%sp + 0x1],%r0  ; B[sp + 0x81] ← r0の下位8ビット
```

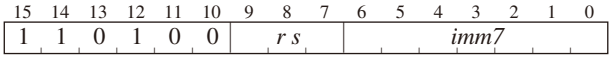
ld.b [*imm7*], *%rs*

機　能

符号付きバイトデータ転送

標準）　B[*imm7*] ← *rs*(7:0)
拡張1）　B[*imm20*] ← *rs*(7:0)
拡張2）　B[*imm24*] ← *rs*(7:0)

コード



フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src:レジスタ直接 `%rs` = `%r0`～`%r7`
Dst:即値（符号なし）

CLK

1サイクル

説　明

(1)標準

```
ld.b  [imm7],%rs     ; memory address = sp + imm7
```

*rs*レジスタの下位8ビットを指定メモリに転送します。7ビット即値*imm7*がアクセスされるメモリアドレスとなります。

(2)拡張1

```
ext   imm13          ; = imm20(19:7)
ld.b  [imm7],%rs     ; memory address = imm20, imm7 = imm20(6:0)
```

`ext`命令により、メモリアドレスが20ビットに拡張されます。これにより、*rs*レジスタの下位8ビットを20ビット即値*imm20*で指定されるアドレスに転送します。

(3)拡張2

```
ext   imm4           ; imm4(3:0) = imm24(23:20)
ext   imm13          ; = imm24(19:7)
ld.b  [imm7],%rs     ; memory address = imm24, imm7 = imm24(6:0)
```

2つの`ext`命令により、メモリアドレスが24ビットに拡張されます。これにより、*rs*レジスタの下位8ビットを24ビット即値*imm24*で指定されるアドレスに転送します。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

```
ext   0x1
ld.b  [0x1],%r0          ; B[0x81] ← r0の下位8ビット
```

ld.ca *%rd, %rs*

機　能

コプロセッサへのデータ転送および結果の取得

標準）　co_dout0 ← *rd*, co_dout1 ← *rs*, *rd* ← co_din, psr(C, V, Z, N) ← co_cvzn
拡張1）　不可
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	*rd*			0	0	1	1	*rs*		

フラグ

IL	IE	C	V	Z	N
–	–	↔	↔	↔	↔

モード

Src:レジスタ直接 `%rs` = `%r0`～`%r7`
Dst:レジスタ直接 `%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準

```
ld.ca  %rd,%rs       ; co_dout0 data = rd, co_dout1 data = rs
```

*rd*レジスタと*rs*レジスタに設定したデータをコプロセッサに転送し、コプロセッサの演算結果を*rd*レジスタとPSRのC、V、Z、Nフラグに保存します。

(2)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

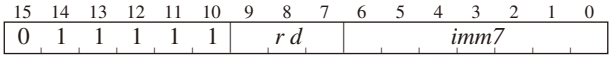
ld.ca *%rd, imm7*

機　能

コプロセッサへのデータ転送および結果の取得

標準）　co_dout0 ← *rd*, co_dout1 ← *imm7*, *rd* ← co_din, psr(C, V, Z, N) ← co_cvzn
拡張1）　co_dout0 ← *rd*, co_dout1 ← *imm20*, *rd* ← co_din, psr(C, V, Z, N) ← co_cvzn
拡張2）　co_dout0 ← *rd*, co_dout1 ← *imm24*, *rd* ← co_din, psr(C, V, Z, N) ← co_cvzn

コード



フラグ

IL	IE	C	V	Z	N
–	–	↔	↔	↔	↔

モード

Src:即値（符号なし）
Dst:レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準

```
ld.ca   %rd,imm7      ; co_dout0 data = rd, co_dout1 data = imm7
```

*rd*レジスタに設定したデータと7ビット即値*imm7*をコプロセッサに転送し、コプロセッサの演算結果を*rd*レジスタとPSRのC、V、Z、Nフラグに保存します。

(2)拡張1

```
ext     imm13         ; = imm20(19:7)
ld.ca   %rd,imm7      ; co_dout0 data = rd
                      ; co_dout1 data = imm20, imm7 = imm20(6:0)
```

`ext`命令により、即値が20ビットに拡張されます。これにより、*rd*レジスタに設定したデータと20ビット即値*imm20*をコプロセッサに転送し、コプロセッサの演算結果を*rd*レジスタとPSRのC、V、Z、Nフラグに保存します。

(3)拡張2

```
ext     imm4          ; imm4(3:0) = imm24(23:20)
ext     imm13         ; = imm24(19:7)
ld.ca   %rd,imm7      ; co_dout0 data = rd
                      ; co_dout1 data = imm24, imm7 ← imm24(6:0)
```

2つの`ext`命令により、即値が24ビットに拡張されます。これにより、*rd*レジスタに設定したデータと24ビット即値*imm24*をコプロセッサに転送し、コプロセッサの演算結果を*rd*レジスタとPSRのC、V、Z、Nフラグに保存します。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

ld.cf *%rd, %rs*

機　能

コプロセッサへのデータ転送およびフラグ状態の取得

標準）　co_dout0 ← *rd*, co_dout1 ← *rs*, psr(C, V, Z, N) ← co_cvzn
拡張1）　不可
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	*rd*			0	0	0	1	*rs*		

フラグ

IL	IE	C	V	Z	N
–	–	↔	↔	↔	↔

モード

Src:レジスタ直接 `%rs` = `%r0`～`%r7`
Dst:レジスタ直接 `%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準

```
ld.cf   %rd,%rs        ; co_dout0 data = rd, co_dout1 data = rs
```

*rd*レジスタと*rs*レジスタに設定したデータをコプロセッサに転送し、コプロセッサのフラグの状態をPSRのC、V、Z、Nフラグに取得します。

(2)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

ld.cf *%rd, imm7*

機　能

コプロセッサへのデータ転送およびフラグ状態の取得

標準）　co_dout0 ← *rd*, co_dout1 ← *imm7*, psr(C, V, Z, N) ← co_cvzn
拡張1）　co_dout0 ← *rd*, co_dout1 ← *imm20*, psr(C, V, Z, N) ← co_cvzn
拡張2）　co_dout0 ← *rd*, co_dout1 ← *imm24*, psr(C, V, Z, N) ← co_cvzn

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	1	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	↔	↔	↔	↔

モード

Src:即値（符号なし）
Dst:レジスタ直接　`%rd` = `%r0`〜`%r7`

CLK

1サイクル

説　明

(1)標準

```
ld.cf   %rd,imm7        ; co_dout0 data = rd, co_dout1 data = imm7
```

*rd*レジスタに設定したデータと7ビット即値*imm7*をコプロセッサに転送し、コプロセッサのフラグ状態をPSRのC、V、Z、Nフラグに取得します。

(2)拡張1

```
ext     imm13           ; = imm20(19:7)
ld.cf   %rd,imm7        ; co_dout0 data = rd
                        ; co_dout1 data = imm20, imm7 = imm20(6:0)
```

`ext`命令により、即値が20ビットに拡張されます。これにより、*rd*レジスタに設定したデータと20ビット即値*imm20*をコプロセッサに転送し、コプロセッサのフラグ状態をPSRのC、V、Z、Nフラグに取得します。

(3)拡張2

```
ext     imm4            ; imm4(3:0) = imm24(23:20)
ext     imm13           ; = imm24(19:7)
ld.cf   %rd,imm7        ; co_dout0 data = rd
                        ; co_dout1 data = imm24, imm7 ← imm24(6:0)
```

2つの`ext`命令により、即値が24ビットに拡張されます。これにより、*rd*レジスタに設定したデータと24ビット即値*imm24*をコプロセッサに転送し、コプロセッサのフラグ状態をPSRのC、V、Z、Nフラグに取得します。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

ld.cw *%rd, %rs*

機　能

コプロセッサへのデータ転送

標準）　co_dout0 ← *rd*, co_dout1 ← *rs*
拡張1）　不可
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	*r d*			0	0	1	0	*r s*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接 `%rs` = `%r0`～`%r7`
Dst: レジスタ直接 `%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準

```
ld.cw  %rd,%rs       ; co_dout0 data = rd, co_dout1 data = rs
```

*rd*レジスタと*rs*レジスタに設定したデータをコプロセッサに転送します。*rd*レジスタおよびPSRのC、V、Z、Nフラグは変更されません。

(2)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

ld.cw *%rd, imm7*

機　能

コプロセッサへのデータ転送

標準）　co_dout0 ← *rd*, co_dout1 ← *imm7*
拡張1）　co_dout0 ← *rd*, co_dout1 ← *imm20*
拡張2）　co_dout0 ← *rd*, co_dout1 ← *imm24*

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	0	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src:即値（符号なし）
Dst:レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準

```
ld.cw   %rd,imm7        ; co_dout0 data = rd, co_dout1 data = imm7
```

*rd*レジスタに設定したデータと7ビット即値*imm7*をコプロセッサに転送します。*rd*レジスタおよびPSRのC、V、Z、Nフラグは変更されません。

(2)拡張1

```
ext     imm13           ; = imm20(19:7)
ld.cw   %rd,imm7        ; co_dout0 data = rd
                        ; co_dout1 data = imm20, imm7 = imm20(6:0)
```

`ext`命令により、即値が20ビットに拡張されます。これにより、*rd*レジスタに設定したデータと20ビット即値*imm20*をコプロセッサに転送します。*rd*レジスタおよびPSRのC、V、Z、Nフラグは変更されません。

(3)拡張2

```
ext     imm4            ; imm4(3:0) = imm24(23:20)
ext     imm13           ; = imm24(19:7)
ld.cw   %rd,imm7        ; co_dout0 data = rd
                        ; co_dout1 data = imm24, imm7 ← imm24(6:0)
```

2つの`ext`命令により、即値が24ビットに拡張されます。これにより、*rd*レジスタに設定したデータと24ビット即値*imm24*をコプロセッサに転送します。*rd*レジスタおよびPSRのC、V、Z、Nフラグは変更されません。

(4)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

ld.ub *%rd, %rs*

機　能

符号なしバイトデータ転送

標準）　*rd*(7:0) ← *rs*(7:0), *rd*(15:8) ← 0, *rd*(23:16) ← 0
拡張1）　不可
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	0	*r d*			0	0	0	1	*r s*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接　`%rs` = `%r0`～`%r7`
Dst: レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準
*rs*レジスタの下位8ビットを16ビットにゼロ拡張して*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。

(2)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

例

```
ld.ub  %r0,%r1  ; r0 ← r1(7:0)をゼロ拡張
```

ld.ub %rd, [%rb]

機　能

符号なしバイトデータ転送

標準）　*rd*(7:0) ← B[*rb*], *rd*(15:8) ← 0, *rd*(23:16) ← 0
拡張1）　*rd*(7:0) ← B[*rb* + *imm13*], *rd*(15:8) ← 0, *rd*(24:16) ← 0
拡張2）　*rd*(7:0) ← B[*rb* + *imm24*], *rd*(15:8) ← 0, *rd*(24:16) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	*r d*			0	0	0	1	*r b*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ間接　`%rb` = `%r0`〜`%r7`
Dst: レジスタ直接　`%rd` = `%r0`〜`%r7`

CLK

1サイクル（`ext`命令使用時は2サイクル）

説　明

(1)標準

```
ld.ub   %rd,[%rb]      ; memory address = rb
```

指定メモリのバイトデータを16ビットにゼロ拡張して*rd*レジスタに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。*rd*レジスタの上位8ビットは0に設定されます。

(2)拡張1

```
ext     imm13
ld.ub   %rd,[%rb]      ; memory address = rb + imm13
```

`ext`命令により、アドレッシングモードがディスプレースメント付きレジスタ間接アドレッシングに変わります。これにより、*rb*レジスタの内容に13ビット即値*imm13*を加えたアドレスのバイトデータを16ビットにゼロ拡張して*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。*rb*レジスタの内容は変更されません。

(3)拡張2

```
ext     imm11          ; imm11(10:0) = imm24(23:13)
ext     imm13          ; = imm24(12:0)
ld.ub   %rd,[%rb]      ; memory address = rb + imm24
```

アドレッシングモードがディスプレースメント付きレジスタ間接アドレッシングに変わり、*rb*レジスタの内容に24ビット即値*imm24*を加えたアドレスのバイトドデータを16ビットにゼロ拡張して*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。*rb*レジスタの内容は変更されません。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

ld.ub %rd, [%rb]+
ld.ub %rd, [%rb]-
ld.ub %rd, -[%rb]

機 能

符号なしバイトデータ転送(アドレスインクリメント/デクリメントオプション付き)

ld.ub *%rd*, [*%rb*]+（ポストインクリメントオプション付き）

標準） *rd*(7:0) ← B[*rb*], *rd*(15:8) ← 0, *rd*(23:16) ← 0, *rb*(23:0) ← *rb*(23:0) + 1
拡張1） *rd*(7:0) ← B[*rb*], *rd*(15:8) ← 0, *rd*(24:16) ← 0, *rb*(23:0) ← *rb*(23:0) + *imm13*
拡張2） *rd*(7:0) ← B[*rb*], *rd*(15:8) ← 0, *rd*(24:16) ← 0, *rb*(23:0) ← *rb*(23:0) + *imm24*

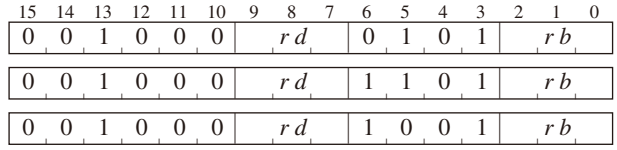
ld.ub *%rd*, [*%rb*]-（ポストデクリメントオプション付き）

標準） *rd*(7:0) ← B[*rb*], *rd*(15:8) ← 0, *rd*(23:16) ← 0, *rb*(23:0) ← *rb*(23:0) - 1
拡張1） *rd*(7:0) ← B[*rb*], *rd*(15:8) ← 0, *rd*(24:16) ← 0, *rb*(23:0) ← *rb*(23:0) - *imm13*
拡張2） *rd*(7:0) ← B[*rb*], *rd*(15:8) ← 0, *rd*(24:16) ← 0, *rb*(23:0) ← *rb*(23:0) - *imm24*

ld.ub *%rd*, -[*%rb*]（プリデクリメントオプション付き）

標準） *rb*(23:0) ← *rb*(23:0) - 1, *rd*(7:0) ← B[*rb*], *rd*(15:8) ← 0, *rd*(23:16) ← 0
拡張1） *rb*(23:0) ← *rb*(23:0) - *imm13*, *rd*(7:0) ← B[*rb*], *rd*(15:8) ← 0, *rd*(24:16) ← 0
拡張2） *rb*(23:0) ← *rb*(23:0) - *imm24*, *rd*(7:0) ← B[*rb*], *rd*(15:8) ← 0, *rd*(24:16) ← 0

コード



フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src:レジスタ間接 `%rb` = `%r0`～`%r7`
Dst:レジスタ直接 `%rd` = `%r0`～`%r7`

CLK

2サイクル

説 明

(1)アドレスインクリメント/デクリメントオプション

`[]+`、`[]-`、`-[]`のオプション指定により、メモリアドレスが自動的にインクリメント/デクリメントされ、連続したデータ転送が簡単にプログラムできます。

`ld.ub %rd,[%rb]+` ポストインクリメント付きロード命令
データ転送の後、メモリアドレスがインクリメントされます。

`ld.ub %rd,[%rb]-` ポストデクリメント付きロード命令
データ転送の後、メモリアドレスがデクリメントされます。

`ld.ub %rd,-[%rb]` プリデクリメント付きロード命令
データ転送の前にメモリアドレスがデクリメントされます。

インクリメント/デクリメントサイズは次のとおりです。
`ext`命令なし(標準): 1(バイトサイズ)
`ext` 1個付き(拡張1): *imm13*
`ext` 2個付き(拡張2): *imm24*

(2)標準(ポストインクリメントの例)

```
ld.ub  %rd,[%rb]+ ; source memory address = rb
                  ; post increment: rb + 1
```

指定メモリのバイトデータを16ビットにゼロ拡張して*rd*レジスタに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。*rd*レジスタの上位8ビットは0に設定されます。
データ転送の後、メモリアドレスが1バイト分インクリメントされます。

(3)拡張1(ポストデクリメントの例)

```
ext     imm13
ld.ub   %rd,[%rb]- ; source memory address = rb
                   ; post decrement: rb - imm13
```

指定メモリのバイトデータを16ビットにゼロ拡張して*rd*レジスタに転送します。*rb*レジスタの内容がアクセスされるメモリアドレスとなります。*rd*レジスタの上位8ビットは0に設定されます。
データ転送の後、メモリアドレスが*imm13*バイト分デクリメントされます。

(4)拡張2(プリデクリメントの例)

```
ext     imm11       ; imm11(10:0) = imm24(23:13)
ext     imm13       ; = imm24(12:0)
ld.ub   %rd,-[%rb] ; source memory address = rb - imm24
```

*rb*レジスタで指定されるメモリアドレスを*imm24*バイト分デクリメントした後、そのアドレスのバイトデータを16ビットにゼロ拡張して*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。

(5)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。 この場合はext命令による拡張は行えません。

ld.ub *%rd*, [%sp + *imm7*]

機　能

符号なしバイトデータ転送

標準）　*rd*(7:0) ← B[sp + *imm7*], *rd*(15:8) ← 0, *rd*(23:16) ← 0
拡張1）　*rd*(7:0) ← B[sp + *imm20*], *rd*(15:8) ← 0, *rd*(23:16) ← 0
拡張2）　*rd*(7:0) ← B[sp + *imm24*], *rd*(15:8) ← 0, *rd*(23:16) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	1	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: ディスプレースメント付きレジスタ間接
Dst: レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

2サイクル

説　明

(1)標準

```
ld.ub   %rd,[%sp + imm7]     ; memory address = sp + imm7
```

指定メモリのバイトデータを16ビットにゼロ拡張して*rd*レジスタに転送します。現在のSPの内容に7ビット即値*imm7*をディスプレースメントとして加算した値がアクセスされるメモリアドレスとなります。*rd*レジスタの上位8ビットは0に設定されます。

(2)拡張1

```
ext     imm13                ; = imm20(19:7)
ld.ub   %rd,[%sp + imm7]     ; memory address = sp + imm20,
                             ; imm7 = imm20(6:0)
```

`ext`命令により、ディスプレースメントが20ビットに拡張されます。これにより、SPの内容に20ビット即値*imm20*を加えたアドレスのバイトデータを16ビットにゼロ拡張して*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。

(3)拡張2

```
ext     imm4                 ; imm4(3:0) = imm24(23:20)
ext     imm13                ; = imm24(19:7)
ld.ub   %rd,[%sp + imm7]     ; memory address = sp + imm24,
                             ; imm7 ← imm24(6:0)
```

2つの`ext`命令により、ディスプレースメントが24ビットに拡張されます。これにより、SPの内容に24ビット即値*imm24*を加えたアドレスのバイトデータを16ビットにゼロ拡張して*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。

(4)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

```
ext     0x1
ld.ub   %r0,[%sp + 0x1] ; r0 ← [sp + 0x81]をゼロ拡張
```

ld.ub %rd, [imm7]

機　能

符号なしバイトデータ転送

標準）　*rd*(7:0) ← B[*imm7*], *rd*(15:8) ← 0, *rd*(23:16) ← 0
拡張1）　*rd*(7:0) ← B[*imm20*], *rd*(15:8) ← 0, *rd*(23:16) ← 0
拡張2）　*rd*(7:0) ← B[*imm24*], *rd*(15:8) ← 0, *rd*(23:16) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	1	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src:即値（符号なし）
Dst:レジスタ直接　`%rd` = `%r0`〜`%r7`

CLK

1サイクル

説　明

(1)標準

```
ld.ub   %rd,[imm7]     ; memory address = imm7
```

指定メモリのバイトデータを16ビットにゼロ拡張して*rd*レジスタに転送します。7ビット即値*imm7*がアクセスされるメモリアドレスとなります。*rd*レジスタの上位8ビットは0に設定されます。

(2)拡張1

```
ext     imm13          ; = imm20(19:7)
ld.ub   %rd,[imm7]     ; memory address = imm20, imm7 = imm20(6:0)
```

ext命令により、メモリアドレスが20ビットに拡張されます。これにより、20ビット即値*imm20*で指定されるアドレスのバイトデータを16ビットにゼロ拡張して*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。

(3)拡張2

```
ext     imm4           ; imm4(3:0) = imm24(23:20)
ext     imm13          ; = imm24(19:7)
ld.ub   %rd,[imm7]     ; memory address = imm24, imm7 ← imm24(6:0)
```

2つのext命令により、メモリアドレスが24ビットに拡張されます。これにより、24ビット即値*imm24*で指定されるアドレスのバイトデータを16ビットにゼロ拡張して*rd*レジスタに転送します。*rd*レジスタの上位8ビットは0に設定されます。

(4)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合はext命令による拡張は行えません。

例

```
ext     0x1
ld.ub   %r0,[0x1] ; r0 ← [0x81]をゼロ拡張
```

nop

機　能

No operation

標準）　No operation
拡張1）　不可
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

–

CLK

1サイクル

説　明

(1)標準
　何の動作もせずに1サイクルの時間を費やします。PCはインクリメント(+2)されます。

(2)ディレイドスロット命令
　本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

例

```
nop
nop                    ; 2サイクルのウェイト
```

not %rd, %rs
not/c %rd, %rs
not/nc %rd, %rs

機 能

16ビット論理否定

標準） *rd*(15:0) ← !*rs*(15:0), *rd*(23:16) ← 0
拡張1） *rd*(15:0) ← !*imm13*(ゼロ拡張), *rd*(23:16) ← 0
拡張2） *rd*(15:0) ← !*imm16*, *rd*(23:16) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	1	1	*rd*			1	0	1	1	*rs*			not
0	0	1	0	1	1	*rd*			0	0	1	1	*rs*			not/c
0	0	1	0	1	1	*rd*			0	1	1	1	*rs*			not/nc

フラグ

IL	IE	C	V	Z	N
–	–	–	0	↔	↔

モード

Src:レジスタ直接 %*rs* = %r0～%r7
Dst:レジスタ直接 %*rd* = %r0～%r7

CLK

1サイクル

説 明

(1)標準

```
not   %rd,%rs          ; rd ← !rs
```

*rs*レジスタの下位16ビットを反転し、*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。

(2)拡張1

```
ext   imm13
not   %rd,%rs          ; rd ← !imm13
```

13ビット即値*imm13*を16ビットにゼロ拡張した後にビットを反転し、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。

(3)拡張2

```
ext   imm3             ; imm3(2:0) = imm16(15:13)
ext   imm13            ; = imm16(12:0)
not   %rd,%rs          ; rd ← !imm16
```

16ビット即値*imm16*の全ビットを反転し、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。

(4)条件実行

オペコードに/cまたは/ncを付けることにより、条件実行命令になります。
not/c　Cフラグが1の場合に実行、0の場合はnop
not/nc　Cフラグが0の場合に実行、1の場合はnop

この場合も、ext命令による拡張が可能です。

(5)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合はext命令による拡張は行えません。

例

r1 = 0x555555の場合

```
not   %r0,%r1     ; r0 = 0x00aaaa
```

not *%rd, sign7*

機　能

16ビット論理否定

標準）　*rd*(15:0) ← !*sign7*(符号拡張), *rd*(23:16) ← 0
拡張1）　*rd*(15:0) ← !*sign16*, *rd*(23:16) ← 0
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	*r d*			*sign7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	0	↔	↔

モード

Src:即値（符号付き）
Dst:レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準

```
not   %rd,sign7       ; rd ← !sign7
```

7ビット即値*sign7*を16ビットに符号拡張した後にビットを反転し、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。

(2)拡張1

```
ext   imm9            ; imm9(8:0) = sign16(15:7)
not   %rd,sign7       ; rd ← !sign16, sign7 = sign16(6:0)
```

16ビット即値*sign16*の全ビットを反転し、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。

(3)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

(1)
```
not   %r0,0x3f        ; r0 = 0x00ffc0
```

(2)
```
ext   0x1ff
not   %r1,0x7f        ; r1 = 0x000000
```

or %rd, %rs
or/c %rd, %rs
or/nc %rd, %rs

機 能

16ビット論理和

標準） *rd*(15:0) ← *rd*(15:0) | *rs*(15:0), *rd*(23:16) ← 0
拡張1） *rd*(15:0) ← *rs*(15:0) | *imm13*(ゼロ拡張), *rd*(23:16) ← 0
拡張2） *rd*(15:0) ← *rs*(15:0) | *imm16*, *rd*(23:16) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	1	1	*rd*			1	0	0	1	*rs*			or
0	0	1	0	1	1	*rd*			0	0	0	1	*rs*			or/c
0	0	1	0	1	1	*rd*			0	1	0	1	*rs*			or/nc

フラグ

IL	IE	C	V	Z	N
–	–	–	0	↔	↔

モード

Src: レジスタ直接 `%rs` = `%r0`〜`%r7`
Dst: レジスタ直接 `%rd` = `%r0`〜`%r7`

CLK

1サイクル

説 明

(1)標準

```
or    %rd,%rs          ; rd ← rd | rs
```

*rs*レジスタの内容と*rd*レジスタの内容の論理和をとり、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23〜16は0に設定されます。

(2)拡張1

```
ext   imm13
or    %rd,%rs          ; rd ← rs | imm13
```

*rs*レジスタの内容とゼロ拡張した13ビット即値*imm13*の論理和をとり、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23〜16は0に設定されます。*rs*レジスタの内容は変更されません。

(3)拡張2

```
ext   imm3             ; imm3(2:0) = imm16(15:13)
ext   imm13            ; = imm16(12:0)
or    %rd,%rs          ; rd ← rs | imm16
```

*rs*レジスタの内容と16ビット即値*imm16*の論理和をとり、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23〜16は0に設定されます。*rs*レジスタの内容は変更されません。

(4)条件実行

オペコードに/cまたは/ncを付けることにより、条件実行命令になります。
or/c　Cフラグが1の場合に実行、0の場合はnop
or/nc　Cフラグが0の場合に実行、1の場合はnop

この場合も、ext命令による拡張が可能です。

(5)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合はext命令による拡張は行えません。

例

(1)
```
or    %r0,%r0          ; r0 = r0 | r0
```

(2)
```
ext   0x1
ext   0x1fff
or    %r1,%r2          ; r1 = r2 | 0x3fff
```

or %rd, sign7

機　能

16ビット論理和

標準）　*rd*(15:0) ← *rd*(15:0) | *sign7*(符号拡張), *rd*(23:16) ← 0
拡張1）　*rd*(15:0) ← *rd*(15:0) | *sign16*, *rd*(23:16) ← 0
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	*r d*			*sign7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	0	↔	↔

モード

Src:即値（符号付き）
Dst:レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準

```
or    %rd,sign7       ; rd ← rd | sign7
```

*rd*レジスタの内容と符号拡張した7ビット即値*sign7*の論理和をとり、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。

(2)拡張1

```
ext   imm9            ; imm9(8:0) = sign16(15:7)
or    %rd,sign7       ; rd ← rd | sign16, sign7 = sign16(6:0)
```

*rd*レジスタの内容と16ビット即値*sign16*の論理和をとり、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。

(3)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

```
(1) or    %r0,0x7e        ; r0 = r0 | 0xfffe

(2) ext   0xff
    or    %r1,0x7f        ; r1 = r1 | 0x7fff
```

ret
ret.d

機　能

サブルーチンからのリターン

標準）　pc ← A[sp](23:0), sp ← sp + 4
拡張1）　不可
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	ret
0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	ret.d

フラグ

IE	C	V	Z	N
–	–	–	–	–

モード

–

CLK

ret	3サイクル
ret.d	2サイクル(ディレイドスロット命令 = 1サイクルの場合)、3サイクル(その他)

説　明

(1)標準

```
ret
```

call/calla命令実行時にスタックに退避させたPC値(リターンアドレス)をPCに戻して、サブルーチンから呼び出し元のルーチンにリターンします。SPは4バイト分インクリメントされます。
サブルーチン内でスタック操作を行った場合は、ret命令実行前にSPがリターンアドレスを示すように戻しておく必要があります。

(2)ディレイド分岐(dビット(ビット7) = 1)

```
ret.d
```

ret.d命令では次の命令がディレイドスロット命令となります。ディレイドスロット命令はサブルーチンからのリターン前に実行されます。ret.d命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
ret.d
add   %r0,%r1    ; リターン前に実行
```

注　意

ret.d命令(ディレイド分岐)を使用する場合、次の命令はディレイドスロット命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

retd

機　能　デバッグ処理ルーチンからのリターン

標準）　r0 ← A[DBRAM + 0x4](23:0), {psr, pc} ← A[DBRAM + 0x0]
拡張1）　不可
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	1	1	0	1	0	0	0

フラグ

IL	IE	C	V	Z	N
↔	↔	↔	↔	↔	↔

モード　–

CLK　4サイクル

説　明　デバッグ割り込み発生時にデバッグ用ワークエリア(DBRAM)に退避させたR0、PCとPSRの内容をそれぞれに戻して、デバッグ処理ルーチンからリターンします。
本命令はデバッグファームウェア用です。ユーザプログラム内では使用しません。

例
```
retd                ; デバッグ処理ルーチンからのリターン
```

reti
reti.d

機　能

割り込み処理ルーチンからのリターン

標準）　{psr, pc} ← A[sp], sp ← sp + 4
拡張1）　不可
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	reti
0	0	0	0	0	0	0	1	1	0	1	0	1	0	0	0	reti.d

フラグ

IL	IE	C	V	Z	N
↔	↔	↔	↔	↔	↔

モード

–

CLK

`reti`　　3サイクル
`reti.d`　2サイクル（ディレイドスロット命令 = 1サイクルの場合）、3サイクル（その他）

説　明

(1)標準

`reti`

割り込み発生時にスタックに退避させたPCとPSRの内容をそれぞれに戻して、割り込み処理ルーチンからリターンします。SPは4バイト分インクリメントされます。

(2)ディレイド分岐（dビット（ビット7）= 1）

`reti.d`

`reti.d`命令では次の命令がディレイドスロット命令となります。ディレイドスロット命令は割り込み処理ルーチンからのリターン前に実行されます。`reti.d`命令と次のディレイドスロット命令の間は割り込みがマスクされ、発生しません。

例

```
reti                ; 割り込み処理ルーチンからリターン
```

sa %rd, %rs

機　能

右方向算術シフト

標準）　*rd*の内容を*rs*の指定ビット（0～3、4または8ビット）分、右にシフト
MSB ← MSB（符号ビット）

拡張1）　不可

拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	*rd*			1	1	0	1	*rs*		

フラグ

IL	IE	C	V	Z	N
–	–	↔	–	↔	↔

モード

Src: レジスタ直接　`%rs` = `%r0`～`%r7`
Dst: レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

1サイクル

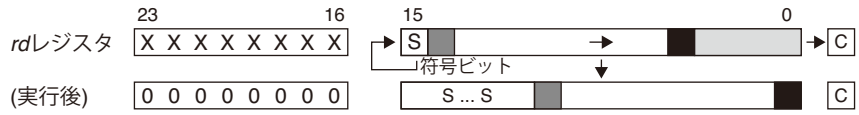
説　明

(1)標準

*rd*レジスタのビットを図のようにシフトさせます。
ビットのシフト量は*rs*レジスタの値により以下のとおり設定されます。

rs = 0～3:　0～3ビット
rs = 4～7:　4ビット
rs = 8以上: 8ビット

*rd*レジスタのビット15には符号ビットがコピーされます。処理は下位16ビットに対して行われ、*rd*レジスタのビット23～16は0に設定されます。



(2)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

sa %rd, imm7

機　能

右方向算術シフト

標準）　*rd*の内容を*imm7*の指定ビット（0～3、4または8ビット）分、右にシフト
MSB ← MSB（符号ビット）

拡張1）　*imm7*を*imm20*に拡張

拡張2）　*imm7*を*imm24*に拡張

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	1	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	↔	–	↔	↔

モード

Src: 即値（符号なし）
Dst: レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

1サイクル

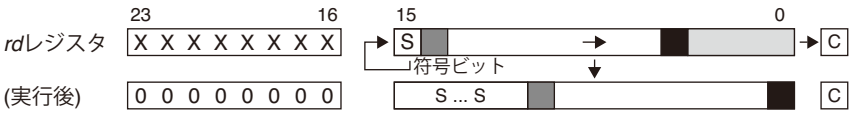
説　明

（1）標準

*rd*レジスタのビットを図のようにシフトさせます。
ビットのシフト量は7ビット即値*imm7*により以下のとおり設定されます。

imm7 = 0～3:　0～3ビット
imm7 = 4～7:　4ビット
imm7 = 8以上: 8ビット

*rd*レジスタのビット15には符号ビットがコピーされます。処理は下位16ビットに対して行われ、*rd*レジスタのビット23～16は0に設定されます。



（2）拡張

`ext`命令により7ビット即値*imm7*が20ビット即値*imm20*、または24ビット即値*imm24*に拡張されます。ただし、標準命令と動作に違いはありません。

（3）ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

sbc %rd, %rs
sbc/c %rd, %rs
sbc/nc %rd, %rs

機 能

ボロー付き16ビット減算

標準） *rd*(15:0) ← *rd*(15:0) - *rs*(15:0) - C, *rd*(23:16) ← 0
拡張1） *rd*(15:0) ← *rs*(15:0) - *imm13*(ゼロ拡張) - C, *rd*(23:16) ← 0
拡張2） *rd*(15:0) ← *rs*(15:0) - *imm16* - C, *rd*(23:16) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	1	1	0	*r d*			1	0	1	1	*r s*			sbc
0	0	1	1	1	0	*r d*			0	0	1	1	*r s*			sbc/c
0	0	1	1	1	0	*r d*			0	1	1	1	*r s*			sbc/nc

フラグ

IL	IE	C	V	Z	N	
–	–	↔	↔	↔	↔	sbc
–	–	–	↔	↔	↔	sbc/c, sbc/nc

モード

Src:レジスタ直接 %*rs* = %r0～%r7
Dst:レジスタ直接 %*rd* = %r0～%r7

CLK

1サイクル

説 明

(1)標準

```
sbc  %rd,%rs          ; rd ← rd - rs - C
```

*rd*レジスタから*rs*レジスタの内容とC(キャリー)フラグの内容を減算します。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。

(2)拡張1

```
ext  imm13
sbc  %rd,%rs          ; rd ← rs - imm13 - C
```

*rs*レジスタの内容から13ビット即値*imm13*とC(キャリー)フラグの内容をゼロ拡張して減算し、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。*rs*レジスタの内容は変更されません。

(3)拡張2

```
ext  imm3             ; imm3(2:0) = imm16(15:13)
ext  imm13            ; = imm16(12:0)
sbc  %rd,%rs          ; rd ← rs - imm16 - C
```

*rs*レジスタの内容から16ビット即値*imm16*とC(キャリー)フラグの内容を減算し、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。*rs*レジスタの内容は変更されません。

(4)条件実行

オペコードに/cまたは/ncを付けることにより、条件実行命令になります。
sbc/c　Cフラグが1の場合に実行、0の場合はnop
sbc/nc　Cフラグが0の場合に実行、1の場合はnop

この場合も、ext命令による拡張が可能です。

(5)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合はext命令による拡張は行えません。

例

(1)
```
sbc  %r0,%r1          ; r0 = r0 - r1 - C
```

(2)32ビットデータの減算

データ1 = {r2, r1}, データ2 = {r4, r3}, 減算結果 = {r2, r1}

```
sub  %r1,%r3          ; 下位ワードの減算
sbc  %r2,%r4          ; 上位ワードの減算
```

sbc *%rd, imm7*

機　能

ボロー付き16ビット減算

標準）　*rd*(15:0) ← *rd*(15:0) - *imm7*(ゼロ拡張) - C, *rd*(23:16) ← 0
拡張1）　*rd*(15:0) ← *rd*(15:0) - *imm16* - C, *rd*(23:16) ← 0
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	↔	↔	↔	↔

モード

Src:即値（符号なし）
Dst:レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準

```
sbc   %rd,imm7          ; rd ← rd - imm7 - C
```

*rd*レジスタから7ビット即値*imm7*とC(キャリー)フラグの内容をゼロ拡張して減算します。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。

(2)拡張1

```
ext   imm9              ; imm9(8:0) = imm16(15:7)
sbc   %rd,imm7          ; rd ← rd - imm16 - C, imm7 = imm16(6:0)
```

*rd*レジスタから16ビット即値*imm16*とC(キャリー)フラグの内容を減算します。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。

(3)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

(1)
```
sbc   %r0,0x7f          ; r0 = r0 - 0x7f - C
```

(2)
```
ext   0x1ff
sbc   %r1,0x7f          ; r1 = r1 - 0xffff - C
```

sl %rd, %rs

機　能

左方向論理シフト

標準）　*rd*の内容を*rs*の指定ビット(0〜3、4または8ビット)分、左にシフト
LSB ← 0

拡張1）　不可

拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	*r d*			1	1	1	0	*r s*		

フラグ

IL	IE	C	V	Z	N
–	–	↔	–	↔	↔

モード

Src: レジスタ直接　`%rs` = `%r0`〜`%r7`
Dst: レジスタ直接　`%rd` = `%r0`〜`%r7`

CLK

1サイクル

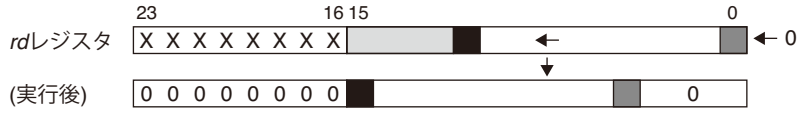
説　明

(1)標準

*rd*レジスタのビットを図のようにシフトさせます。
ビットのシフト量は*rs*レジスタの値により以下のとおり設定されます。

rs = 0〜3:　0〜3ビット
rs = 4〜7:　4ビット
rs = 8以上: 8ビット

*rd*レジスタの最下位ビットには0が入ります。処理は下位16ビットに対して行われ、*rd*レジスタのビット23〜16は0に設定されます。



(2)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

sl *%rd, imm7*

機　能

左方向論理シフト

標準）　*rd*の内容を*imm7*の指定ビット（0～3、4または8ビット）分、左にシフト
LSB ← 0

拡張1）　*imm7*を*imm20*に拡張

拡張2）　*imm7*を*imm24*に拡張

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	0	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	↔	–	↔	↔

モード

Src:即値（符号なし）
Dst:レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

1サイクル

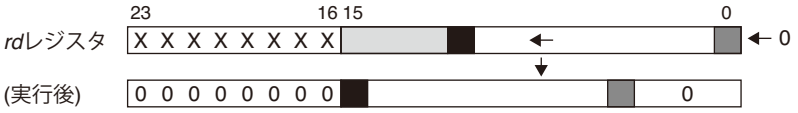
説　明

（1）標準

*rd*レジスタのビットを図のようにシフトさせます。
ビットのシフト量は7ビット即値*imm7*により以下のとおり設定されます。

imm7 = 0～3:　0～3ビット
imm7 = 4～7:　4ビット
imm7 = 8以上: 8ビット

*rd*レジスタの最下位ビットには0が入ります。処理は下位16ビットに対して行われ、*rd*レジスタのビット23～16は0に設定されます。



（2）拡張

`ext`命令により7ビット即値*imm7*が20ビット即値*imm20*、または24ビット即値*imm24*に拡張されます。ただし、標準命令と動作に違いはありません。

（3）ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

slp

機　能

SLEEP
標準）　プロセッサをSLEEPモードに設定
拡張1）　不可
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

–

CLK

6サイクル

説　明

プロセッサをSLEEPモードにします。これにより、消費電流を抑えることができます。
S1C17コアがslp命令を実行すると、その時点でプログラムの実行を中断しSLEEPモードに移行します。SLEEPモードではS1C17コアおよびチップ上の周辺回路も動作を停止するのが一般的で、HALTモードよりも大幅に消費電流を低減することができます。ただし、動作を停止するモジュールはコア外部のクロック制御回路のインプリメンテーションに依存します。

SLEEPモードを解除する要因はイニシャルリセット以外、S1C17コア外部のクロック制御回路のインプリメンテーションに依存します。一般的にはイニシャルリセット、マスク可能な外部割り込み、NMI、デバッグ割り込みによって解除します。
割り込みによるSLEEPモードの解除には、プロセッサの割り込み許可/禁止の状態は影響しません。PSRのIEフラグや、割り込みコントローラの割り込み許可ビット（インプリメント依存）などが割り込み禁止に設定されている場合でも、割り込み信号によりSLEEPモードを解除することができます。
プロセッサが割り込み許可の状態で、割り込みによってSLEEPモードを解除した場合は、対応する割り込み処理ルーチンを実行します。したがって、発生した割り込みの処理ルーチンをretiで終了すると、slpの次の命令の位置にリターンします。
プロセッサが割り込み禁止状態の場合、SLEEPモードを解除後はslpの次の命令から実行を開始します。

SLEEPモードの詳細については、機種別のテクニカルマニュアルを参照してください。

例

```
slp          ; プロセッサをSLEEPモードに設定
```

sr %rd, %rs

機　能

右方向論理シフト

標準）　*rd*の内容を*rs*の指定ビット(0～3、4または8ビット)分、右にシフト
MSB ← 0

拡張1）　不可

拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	*rd*			1	1	0	0	*rs*		

フラグ

IL	IE	C	V	Z	N
–	–	↔	–	↔	↔

モード

Src: レジスタ直接　`%rs` = `%r0`～`%r7`
Dst: レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

1サイクル

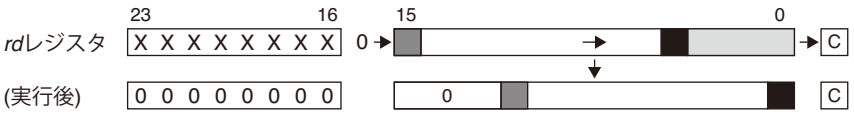
説　明

(1)標準

*rd*レジスタのビットを図のようにシフトさせます。
ビットのシフト量は*rs*レジスタの値により以下のとおり設定されます。

rs = 0～3:　0～3ビット
rs = 4～7:　4ビット
rs = 8以上: 8ビット

*rd*レジスタのビット15には0が入ります。処理は下位16ビットに対して行われ、*rd*レジスタのビット23～16は0に設定されます。



(2)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

sr %rd, imm7

機　能

右方向論理シフト

標準）　*rd*の内容を*imm7*の指定ビット（0～3、4または8ビット）分、右にシフト
MSB ← 0

拡張1）　*imm7*を*imm20*に拡張

拡張2）　*imm7*を*imm24*に拡張

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	↔	–	↔	↔

モード

Src:即値（符号なし）
Dst:レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

1サイクル

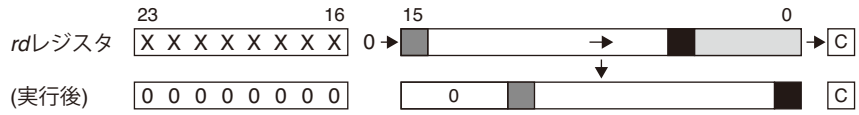
説　明

（1）標準

*rd*レジスタのビットを図のようにシフトさせます。
ビットのシフト量は7ビット即値*imm7*により以下のとおり設定されます。

imm7 = 0～3:　0～3ビット
imm7 = 4～7:　4ビット
imm7 = 8以上: 8ビット

*rd*レジスタのビット15には0が入ります。処理は下位16ビットに対して行われ、*rd*レジスタのビット23～16は0に設定されます。



（2）拡張

`ext`命令により7ビット即値*imm7*が20ビット即値*imm20*、または24ビット即値*imm24*に拡張されます。ただし、標準命令と動作に違いはありません。

（3）ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

sub %rd, %rs
sub/c %rd, %rs
sub/nc %rd, %rs

機 能

16ビット減算

標準） *rd*(15:0) ← *rd*(15:0) - *rs*(15:0), *rd*(23:16) ← 0

拡張1） *rd*(15:0) ← *rs*(15:0) - *imm13*(ゼロ拡張), *rd*(23:16) ← 0

拡張2） *rd*(15:0) ← *rs*(15:0) - *imm16*, *rd*(23:16) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	1	1	0	*rd*			1	0	1	0	*rs*			sub
0	0	1	1	1	0	*rd*			0	0	1	0	*rs*			sub/c
0	0	1	1	1	0	*rd*			0	1	1	0	*rs*			sub/nc

フラグ

IL	IE	C	V	Z	N	
–	–	↔	↔	↔	↔	sub
–	–	–	↔	↔	↔	sub/c, sub/nc

モード

Src: レジスタ直接 `%rs` = `%r0`～`%r7`
Dst: レジスタ直接 `%rd` = `%r0`～`%r7`

CLK

1サイクル

説 明

(1)標準

```
sub   %rd,%rs          ; rd ← rd - rs
```

*rd*レジスタから*rs*レジスタの内容を減算します。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。

(2)拡張1

```
ext   imm13
sub   %rd,%rs          ; rd ← rs - imm13
```

*rs*レジスタの内容から13ビット即値*imm13*をゼロ拡張して減算し、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。*rs*レジスタの内容は変更されません。

(3)拡張2

```
ext   imm3             ; imm3(2:0) = imm16(15:13)
ext   imm13            ; = imm16(12:0)
sub   %rd,%rs          ; rd ← rs - imm16
```

*rs*レジスタの内容から16ビット即値*imm16*を減算し、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。*rs*レジスタの内容は変更されません。

(4)条件実行

オペコードに/cまたは/ncを付けることにより、条件実行命令になります。

sub/c　Cフラグが1の場合に実行、0の場合はnop

sub/nc　Cフラグが0の場合に実行、1の場合はnop

この場合も、ext命令による拡張が可能です。

(5)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合はext命令による拡張は行えません。

例

(1)
```
sub   %r0,%r0          ; r0 = r0 - r0
```

(2)
```
ext   0x1
ext   0x1fff
sub   %r1,%r2          ; r1 = r2 - 0x3fff
```

sub %rd, imm7

機　能

16ビット減算

標準）　*rd*(15:0) ← *rd*(15:0) - *imm7*(ゼロ拡張), *rd*(23:16) ← 0
拡張1）　*rd*(15:0) ← *rd*(15:0) - *imm16*, *rd*(23:16) ← 0
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	↔	↔	↔	↔

モード

Src:即値（符号なし）
Dst:レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準

```
sub   %rd,imm7          ; rd ← rd - imm7
```

*rd*レジスタから7ビット即値*imm7*をゼロ拡張して減算します。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。

(2)拡張1

```
ext   imm9              ; imm9(8:0) = imm16(15:7)
sub   %rd,imm7          ; rd ← rd - imm16, imm7 = imm16(6:0)
```

*rd*レジスタから16ビット即値*imm16*を減算します。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。

(3)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

```
(1) sub   %r0,0x3f          ; r0 = r0 - 0x3f

(2) ext   0x1ff
    sub   %r1,0x7f          ; r1 = r1 - 0xffff
```

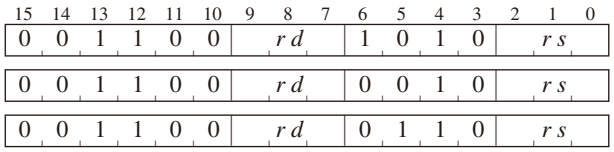
sub.a %rd, %rs
sub.a/c %rd, %rs
sub.a/nc %rd, %rs

機 能

24ビット減算

標準） *rd*(23:0) ← *rd*(23:0) - *rs*(23:0)
拡張1） *rd*(23:0) ← *rs*(23:0) - *imm13*(ゼロ拡張)
拡張2） *rd*(23:0) ← *rs*(23:0) - *imm24*

コード



フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接 %*rs* = %r0～%r7
Dst: レジスタ直接 %*rd* = %r0～%r7

CLK

1サイクル

説 明

(1)標準

```
sub.a   %rd,%rs         ; rd ← rd - rs
```

*rd*レジスタから*rs*レジスタの内容を減算します。

(2)拡張1

```
ext     imm13
sub.a   %rd,%rs         ; rd ← rs - imm13
```

*rs*レジスタの内容から13ビット即値*imm13*をゼロ拡張して減算し、結果を*rd*レジスタにロードします。*rs*レジスタの内容は変更されません。

(3)拡張2

```
ext     imm11           ; imm11(10:0) = imm24(23:13)
ext     imm13           ; = imm24(12:0)
sub.a   %rd,%rs         ; rd ← rs - imm24
```

*rs*レジスタの内容から24ビット即値*imm24*を減算し、結果を*rd*レジスタにロードします。*rs*レジスタの内容は変更されません。

(4)条件実行

オペコードに/cまたは/ncを付けることにより、条件実行命令になります。

```
sub.a/c    Cフラグが1の場合に実行、0の場合はnop
sub.a/nc   Cフラグが0の場合に実行、1の場合はnop
```

この場合も、ext命令による拡張が可能です。

(5)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合はext命令による拡張は行えません。

例

(1)
```
sub.a   %r0,%r0         ; r0 = r0 - r0
```

(2)
```
ext     0x7ff
ext     0x1fff
sub.a   %r1,%r2         ; r1 = r2 - 0xffffff
```

sub.a *%rd, imm7*

機　能

24ビット減算

標準）　*rd*(23:0) ← *rd*(23:0) - *imm7*(ゼロ拡張)
拡張1）　*rd*(23:0) ← *rd*(23:0) - *imm20*(ゼロ拡張)
拡張2）　*rd*(23:0) ← *rd*(23:0) - *imm24*

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	*r d*			*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src:即値(符号なし)
Dst:レジスタ直接　`%rd` = `%r0`～`%r7`

CLK

1サイクル

説　明

(1)標準

```
sub.a   %rd,imm7        ; rd ← rd - imm7
```

*rd*レジスタから7ビット即値*imm7*をゼロ拡張して減算します。

(2)拡張1

```
ext     imm13           ; = imm20(19:7)
sub.a   %rd,imm7        ; rd ← rd - imm20, imm7 = imm20(6:0)
```

*rd*レジスタから20ビット即値*imm20*をゼロ拡張して減算します。

(3)拡張2

```
ext     imm4            ; imm4(3:0) = imm24(23:20)
ext     imm13           ; = imm24(19:7)
sub.a   %rd,imm7        ; rd ← rd - imm24, imm7 = imm24(6:0)
```

*rd*レジスタから24ビット即値*imm24*を減算します。

(4)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

(1)
```
sub.a   %r0,0x7f        ; r0 = r0 - 0x7f
```

(2)
```
ext     0xf
ext     0x1fff
sub.a   %r1,0x7f        ; r1 = r1 - 0xffffff
```

sub.a %sp, %*rs*

機　能

24ビット減算

標準）　sp(23:0) ← sp(23:0) - *rs*(23:0)
拡張1）　sp(23:0) ← *rs*(23:0) - *imm13*(ゼロ拡張)
拡張2）　sp(23:0) ← *rs*(23:0) - *imm24*

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	0	0	0	0	0	0	1	1	*r s*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接 `%rs` = `%r0`〜`%r7`
Dst: レジスタ直接 `%sp`

CLK

1サイクル

説　明

(1)標準

```
sub.a   %sp,%rs         ; sp ← sp - rs
```

スタックポインタSPから*rs*レジスタの内容を減算します。

(2)拡張1

```
ext     imm13
sub.a   %sp,%rs         ; sp ← rs - imm13
```

*rs*レジスタの内容から13ビット即値*imm13*をゼロ拡張して減算し、結果をスタックポインタSPにロードします。*rs*レジスタの内容は変更されません。

(3)拡張2

```
ext     imm11           ; imm11(10:0) = imm24(23:13)
ext     imm13           ; = imm24(12:0)
sub.a   %sp,%rs         ; sp ← rs - imm24
```

*rs*レジスタの内容から24ビット即値*imm24*を減算し、結果をスタックポインタSPにロードします。*rs*レジスタの内容は変更されません。

(4)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合は`ext`命令による拡張は行えません。

例

(1)
```
sub.a   %sp,%r0         ; sp = sp - r0
```

(2)
```
ext     0x1
ext     0x1ffc
sub.a   %sp,%r2         ; sp = r2 - 0x3ffc
```

注　意

減算結果の下位2ビットは常に0としてSPにロードされます。

sub.a %sp, *imm7*

機　能

24ビット減算

標準）　sp(23:0) ← sp(23:0) - *imm7*(ゼロ拡張)
拡張1）　sp(23:0) ← sp(23:0) - *imm20*(ゼロ拡張)
拡張2）　sp(23:0) ← sp(23:0) - *imm24*

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	0	0	0	*imm7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src:即値(符号なし)
Dst:レジスタ直接 %sp

CLK

1サイクル

説　明

(1)標準

```
sub.a  %sp,imm7      ; sp ← sp - imm7
```

スタックポインタSPから7ビット即値*imm7*をゼロ拡張して減算します。

(2)拡張1

```
ext    imm13         ; = imm20(19:7)
sub.a  %sp,imm7      ; sp ← sp - imm20, imm7 = imm20(6:0)
```

スタックポインタSPから20ビット即値*imm20*をゼロ拡張して減算します。

(3)拡張2

```
ext    imm4          ; imm4(3:0) = imm24(23:20)
ext    imm13         ; = imm24(19:7)
sub.a  %sp,imm7      ; sp ← sp - imm24, imm7 = imm24(6:0)
```

スタックポインタSPから24ビット即値*imm24*を減算します。

(4)ディレイドスロット命令
本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合はext命令による拡張は行えません。

例

(1)
```
sub.a  %sp,0x7c      ; sp = sp - 0x7c
```

(2)
```
ext    0x1fff
sub.a  %sp,0x7c      ; sp = sp - 0xffffc
```

注　意

減算結果の下位2ビットは常に0としてSPにロードされます。

swap %rd, %rs

機　能

スワップ

標準）　*rd*(15:8) ← *rs*(7:0), *rd*(7:0) ← *rs*(15:8), *rd*(23:16) ← 0

拡張1）　不可

拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	*rd*			1	1	1	1	*rs*		

フラグ

IL	IE	C	V	Z	N
–	–	–	–	–	–

モード

Src: レジスタ直接　`%rs` = `%r0`〜`%r7`

Dst: レジスタ直接　`%rd` = `%r0`〜`%r7`

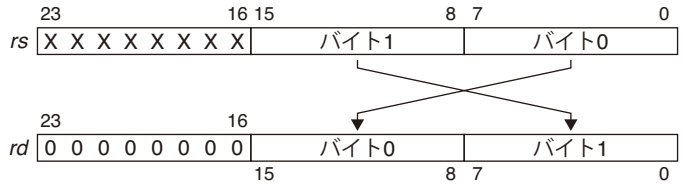
CLK

1サイクル

説　明

(1)標準

*rs*レジスタの下位16ビットの上位/下位バイトデータを入れ替え、結果を*rd*レジスタにロードします。



(2)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。

例

r1 = 0x123456の場合

```
swap  %r2,%r1          ; 0x005634 → r2
```

xor %rd, %rs
xor/c %rd, %rs
xor/nc %rd, %rs

機 能

16ビット排他的論理和

標準） *rd*(15:0) ← *rd*(15:0) ^ *rs*(15:0), *rd*(23:16) ← 0
拡張1） *rd*(15:0) ← *rs*(15:0) ^ *imm13*(ゼロ拡張), *rd*(23:16) ← 0
拡張2） *rd*(15:0) ← *rs*(15:0) ^ *imm16*, *rd*(23:16) ← 0

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	1	1	*r d*			1	0	1	0	*r s*			xor
0	0	1	0	1	1	*r d*			0	0	1	0	*r s*			xor/c
0	0	1	0	1	1	*r d*			0	1	1	0	*r s*			xor/nc

フラグ

IL	IE	C	V	Z	N
–	–	–	0	↔	↔

モード

Src: レジスタ直接 `%rs` = `%r0`～`%r7`
Dst: レジスタ直接 `%rd` = `%r0`～`%r7`

CLK

1サイクル

説 明

(1)標準

```
xor   %rd,%rs        ; rd ← rd ^ rs
```

*rs*レジスタの内容と*rd*レジスタの内容の排他的論理和をとり、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。

(2)拡張1

```
ext   imm13
xor   %rd,%rs        ; rd ← rs ^ imm13
```

*rs*レジスタの内容とゼロ拡張した13ビット即値*imm13*の排他的論理和をとり、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。*rs*レジスタの内容は変更されません。

(3)拡張2

```
ext   imm3           ; imm3(2:0) = imm16(15:13)
ext   imm13          ; = imm16(12:0)
xor   %rd,%rs        ; rd ← rs ^ imm16
```

*rs*レジスタの内容と16ビット即値*imm16*の排他的論理和をとり、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23～16は0に設定されます。*rs*レジスタの内容は変更されません。

(4)条件実行

オペコードに/cまたは/ncを付けることにより、条件実行命令になります。
xor/c　Cフラグが1の場合に実行、0の場合はnop
xor/nc　Cフラグが0の場合に実行、1の場合はnop

この場合も、ext命令による拡張が可能です。

(5)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合はext命令による拡張は行えません。

例

(1)
```
xor   %r0,%r0        ; r0 = r0 ^ r0
```

(2)
```
ext   0x1
ext   0x1fff
xor   %r1,%r2        ; r1 = r2 ^ 0x3fff
```

xor %rd, sign7

機　能

16ビット排他的論理和

標準）　*rd*(15:0) ← *rd*(15:0) ^ *sign7*(符号拡張), *rd*(23:16) ← 0
拡張1）　*rd*(15:0) ← *rd*(15:0) ^ *sign16*, *rd*(23:16) ← 0
拡張2）　不可

コード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	*r d*			*sign7*						

フラグ

IL	IE	C	V	Z	N
–	–	–	0	↔	↔

モード

Src:即値（符号付き）
Dst:レジスタ直接　`%rd` = `%r0`〜`%r7`

CLK

1サイクル

説　明

(1)標準

```
xor   %rd,sign7        ; rd ← rd ^ sign7
```

*rd*レジスタの内容と符号拡張した7ビット即値*sign7*の排他的論理和をとり、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23〜16は0に設定されます。

(2)拡張1

```
ext   imm9             ; imm9(8:0) = sign16(15:7)
xor   %rd,sign7        ; rd ← rd ^ sign16, sign7 = sign16(6:0)
```

*rd*レジスタの内容と16ビット即値*sign16*の排他的論理和をとり、結果を*rd*レジスタにロードします。演算は16ビットで行われ、*rd*レジスタのビット23〜16は0に設定されます。

(3)ディレイドスロット命令

本命令は、dビット付きの分岐命令の直後に記述することによってディレイドスロット命令として実行されます。この場合はext命令による拡張は行えません。

例

```
(1) xor   %r0,0x7e         ; r0 = r0 ^ 0xfffe

(2) ext   0x1ff
    xor   %r1,0x7f         ; r1 = r1 ^ 0xffff
```

Appendix S1C17コア命令一覧

命令一覧中のシンボル

S1C17 Core Instruction Set

レジスタ/レジスタデータ

%rd, rd:	ディスティネーションとなる汎用レジスタ(R0~R7)、またはレジスタの内容
%rs, rs:	ソースとなる汎用レジスタ(R0~R7)、またはレジスタの内容
%rb, rb:	レジスタ間接アドレッシングのベースレジスタを保持している汎用レジスタ(R0~R7)、またはレジスタの内容
%sp, sp:	スタックポインタ(SP)またはその内容
%pc, pc:	プログラムカウンタ(PC)またはその内容

メモリ/アドレス/メモリデータ

[%rb], [%sp]:	レジスタ間接アドレッシング指定
[%rb]+, [%sp]+:	ポストインクリメント付きレジスタ間接アドレッシング指定
[%rb]-, [%sp]-:	ポストデクリメント付きレジスタ間接アドレッシング指定
-[%rb], -[%sp]:	プリデクリメント付きレジスタ間接アドレッシング指定
[%sp+immX]:	ディスプレースメント付きレジスタ間接アドレッシング指定
[imm7]:	即値によるメモリアドレス指定
B[XXX]:	XXXで指定されるアドレス、またはそのアドレスにストアされているバイトデータ
W[XXX]:	XXXで指定される16ビット境界アドレス、またはそのアドレスにストアされているワードデータ
A[XXX]:	XXXで指定される32ビット境界アドレス、またはそのアドレスにストアされている24ビットデータもしくは32ビットデータ

即値

immX:	符号なしXビット即値
signX:	符号付きXビット即値

ビットフィールド

(X):	データのビットX
(X:Y):	ビットXからビットYまでのビットフィールド
{X, Y…}:	ビット構成

コード

rd, rs, rb:	レジスタ番号(R0 = 0 ... R7 = 7)
d:	ディレイドビット(0: 基本分岐命令, 1: ディレイド分岐命令)

機能

←:	右側の内容が左側の項目にロード/設定されることを示します。
+:	加算
-:	減算
&:	論理積
|:	論理和
^:	排他的論理和
!:	論理否定

フラグ

IL:	割り込みレベル
IE:	割り込みイネーブルフラグ
C:	キャリーフラグ
V:	オーバーフローフラグ
Z:	ゼロフラグ
N:	ネガティブフラグ
–:	変更なし
↔:	セット(1)、リセット(0)
1:	セット(1)
0:	リセット(0)

EXT

*X:	ext命令でオペランドが拡張できることを示します(拡張されたオペランドについては、各ページの備考欄を参照してください)。
–:	ext命令でオペランドが拡張できないことを示します。

D

○:	ディレイドスロット命令として使用可能なことを示します。
–:	ディレイドスロット命令として使用できないことを示します。

データ転送命令 (1)

ニーモニック		コード															機　能	サイクル	フラグ						EXT	D
オペコード	オペランド	MSB														LSB			IL	IE	C	V	Z	N		
ld.b	%rd, %rs	0	0	1	0	1	0	rd				0	0	0	0	rs	rd(7:0)←rs(7:0), rd(15:8)←rs(7), rd(23:16)←0	1	–	–	–	–	–	–	–	○
	%rd, [%rb]	0	0	1	0	0	0	rd				0	0	0	0	rb	rd(7:0)←B[rb], rd(15:8)←B[rb](7), rd(23:16)←0	1, 2*7	–	–	–	–	–	–	*1	○
	%rd, [%rb]+	0	0	1	0	0	0	rd				0	1	0	0	rb	rd(7:0)←B[rb], rd(15:8)←B[rb](7), rd(23:16)←0, rb(23:0)←rb(23:0)+1	2	–	–	–	–	–	–	*6	○
	%rd, [%rb]-	0	0	1	0	0	0	rd				1	1	0	0	rb	rd(7:0)←B[rb], rd(15:8)←B[rb](7), rd(23:16)←0, rb(23:0)←rb(23:0)-1	2	–	–	–	–	–	–	*6	○
	%rd, -[%rb]	0	0	1	0	0	0	rd				1	0	0	0	rb	rb(23:0)←rb(23:0)-1, rd(7:0)←B[rb], rd(15:8)←B[rb](7), rd(23:16)←0	2	–	–	–	–	–	–	*6	○
	%rd, [%sp+imm7]	1	1	1	0	0	0	rd						imm7			rd(7:0)←B[sp+imm7], rd(15:8)←B[sp+imm7](7), rd(23:16)←0	2	–	–	–	–	–	–	*5	○
	%rd, [imm7]	1	1	0	0	0	0	rd						imm7			rd(7:0)←B[imm7], rd(15:8)←B[imm7](7), rd(23:16)←0	1	–	–	–	–	–	–	*4	○
	[%rb], %rs	0	0	1	0	0	1	rs				0	0	0	0	rb	B[rb]←rs(7:0)	1, 2*7	–	–	–	–	–	–	*1	○
	[%rb]+, %rs	0	0	1	0	0	1	rs				0	1	0	0	rb	B[rb]←rs(7:0), rb(23:0)←rb(23:0)+1	2	–	–	–	–	–	–	*6	○
	[%rb]-, %rs	0	0	1	0	0	1	rs				1	1	0	0	rb	B[rb]←rs(7:0), rb(23:0)←rb(23:0)-1	2	–	–	–	–	–	–	*6	○
	-[%rb], %rs	0	0	1	0	0	1	rs				1	0	0	0	rb	rb(23:0)←rb(23:0)-1, B[rb]←rs(7:0)	2	–	–	–	–	–	–	*6	○
	[%sp+imm7], %rs	1	1	1	1	0	0	rs						imm7			B[sp+imm7]←rs(7:0)	2	–	–	–	–	–	–	*5	○
	[imm7], %rs	1	1	0	1	0	0	rs						imm7			B[imm7]←rs(7:0)	1	–	–	–	–	–	–	*4	○
ld.ub	%rd, %rs	0	0	1	0	1	0	rd				0	0	0	1	rs	rd(7:0)←rs(7:0), rd(15:8)←0, rd(23:16)←0	1	–	–	–	–	–	–	–	○
	%rd, [%rb]	0	0	1	0	0	0	rd				0	0	0	1	rb	rd(7:0)←B[rb], rd(15:8)←0, rd(23:16)←0	1, 2*7	–	–	–	–	–	–	*1	○
	%rd, [%rb]+	0	0	1	0	0	0	rd				0	1	0	1	rb	rd(7:0)←B[rb], rd(15:8)←0, rd(23:16)←0, rb(23:0)←rb(23:0)+1	2	–	–	–	–	–	–	*6	○
	%rd, [%rb]-	0	0	1	0	0	0	rd				1	1	0	1	rb	rd(7:0)←B[rb], rd(15:8)←0, rd(23:16)←0, rb(23:0)←b(23:0)-1	2	–	–	–	–	–	–	*6	○
	%rd, -[%rb]	0	0	1	0	0	0	rd				1	0	0	1	rb	rb(23:0)←rb(23:0)-1, rd(7:0)←B[rb], rd(15:8)←0, rd(23:16)←0	2	–	–	–	–	–	–	*6	○
	%rd, [%sp+imm7]	1	1	1	0	0	1	rd						imm7			rd(7:0)←B[sp+imm7], rd(15:8)←0, rd(23:16)←0	2	–	–	–	–	–	–	*5	○
	%rd, [imm7]	1	1	0	0	0	1	rd						imm7			rd(7:0)←B[imm7], rd(15:8)←0, rd(23:16)←0	1	–	–	–	–	–	–	*4	○
ld	%rd, %rs	0	0	1	0	1	0	rd				0	0	1	0	rs	rd(15:0)←rs(15:0), rd(23:16)←0	1	–	–	–	–	–	–	–	○
	%rd, sign7	1	0	0	1	1	0	rd						sign7			rd(6:0)←sign7(6:0), rd(15:7)←sign7(6), rd(23:16)←0	1	–	–	–	–	–	–	*2	○
	%rd, [%rb]	0	0	1	0	0	0	rd				0	0	1	0	rb	rd(15:0)←W[rb], rd(23:16)←0	1, 2*7	–	–	–	–	–	–	*1	○
	%rd, [%rb]+	0	0	1	0	0	0	rd				0	1	1	0	rb	rd(15:0)←W[rb], rd(23:16)←0, rb(23:0)←rb(23:0)+2	2	–	–	–	–	–	–	*6	○
	%rd, [%rb]-	0	0	1	0	0	0	rd				1	1	1	0	rb	rd(15:0)←W[rb], rd(23:16)←0, rb(23:0)←rb(23:0)-2	2	–	–	–	–	–	–	*6	○
	%rd, -[%rb]	0	0	1	0	0	0	rd				1	0	1	0	rb	rb(23:0)←rb(23:0)-2, rd(15:0)←W[rb], rd(23:16)←0	2	–	–	–	–	–	–	*6	○
	%rd, [%sp+imm7]	1	1	1	0	1	0	rd						imm7			rd(15:0)←W[sp+imm7], rd(23:16)←0	2	–	–	–	–	–	–	*5	○
	%rd, [imm7]	1	1	0	0	1	0	rd						imm7			rd(15:0)←W[imm7], rd(23:16)←0	1	–	–	–	–	–	–	*4	○
	[%rb], %rs	0	0	1	0	0	1	rs				0	0	1	0	rb	W[rb]←rs(15:0)	1, 2*7	–	–	–	–	–	–	*1	○
	[%rb]+, %rs	0	0	1	0	0	1	rs				0	1	1	0	rb	W[rb]←rs(15:0), rb(23:0)←rb(23:0)+2	2	–	–	–	–	–	–	*6	○
	[%rb]-, %rs	0	0	1	0	0	1	rs				1	1	1	0	rb	W[rb]←rs(15:0), rb(23:0)←rb(23:0)-2	2	–	–	–	–	–	–	*6	○
	-[%rb], %rs	0	0	1	0	0	1	rs				1	0	1	0	rb	rb(23:0)←rb(23:0)-2, W[rb]←rs(15:0)	2	–	–	–	–	–	–	*6	○

備　考

*1) EXT 1個使用: ベースアドレス = rb+imm13, EXT 2個使用: ベースアドレス = rb+imm24
*2) EXT 1個使用: データ = sign16
*3) EXT 1個使用: データ = imm20, EXT 2個使用: データ = imm24
*4) EXT 1個使用: ベースアドレス = imm20, EXT 2個使用: ベースアドレス = imm24
*5) EXT 1個使用: ベースアドレス = sp+imm20, EXT 2個使用: ベースアドレス = sp+imm24
*6) EXT 1個使用: ベースアドレス = rb, アドレスインクリメント/デクリメント rb/sp ← rb/sp±imm13, EXT 2個使用: ベースアドレス = rb, アドレスインクリメント/デクリメント rb/sp ← rb/sp±imm24
*7) EXT未使用時: 1サイクル, EXT使用時: 2サイクル

データ転送命令 (2)

ニーモニック		コード																機　能	サイクル	フラグ						EXT	D
オペコード	オペランド	MSB															LSB			IL	IE	C	V	Z	N		
ld	[%sp+imm7], %rs	1	1	1	1	1	0	rs						imm7				W[sp+imm7]←rs(15:0)	2	–	–	–	–	–	–	∗5	○
	[imm7], %rs	1	1	0	1	1	0	rs						imm7				W[imm7]←rs(15:0)	1	–	–	–	–	–	–	∗4	○
ld.a	%rd, %rs	0	0	1	0	1	0	rd		0	0	1	1				rs	rd(23:0)←rs(23:0)	1	–	–	–	–	–	–	–	○
	%rd, imm7	1	0	0	1	1	1	rd						imm7				rd(6:0)←imm7(6:0), rd(23:7)←0	1	–	–	–	–	–	–	∗3	○
	%rd, [%rb]	0	0	1	0	0	0	rd		0	0	1	1				rb	rd(23:0)←A[rb](23:0), 無視←A[rb](31:24)	1, 2∗8	–	–	–	–	–	–	∗1	○
	%rd, [%rb]+	0	0	1	0	0	0	rd		0	1	1	1				rb	rd(23:0)←A[rb](23:0), 無視←A[rb](31:24), rb(23:0)←rb(23:0)+4	2	–	–	–	–	–	–	∗6	○
	%rd, [%rb]-	0	0	1	0	0	0	rd		1	1	1	1				rb	rd(23:0)←A[rb](23:0), 無視←A[rb](31:24), rb(23:0)←rb(23:0)-4	2	–	–	–	–	–	–	∗6	○
	%rd, -[%rb]	0	0	1	0	0	0	rd		1	0	1	1				rb	rb(23:0)←rb(23:0)-4, rd(23:0)←A[rb](23:0), 無視←A[rb](31:24)	2	–	–	–	–	–	–	∗6	○
	%rd, [%sp+imm7]	1	1	1	0	1	1	rd						imm7				rd(23:0)←A[sp+imm7](23:0), 無視←A[sp+imm7](31:24)	2	–	–	–	–	–	–	∗5	○
	%rd, [imm7]	1	1	0	0	1	1	rd						imm7				rd(23:0)←A[imm7](23:0), 無視←A[imm7](31:24)	1	–	–	–	–	–	–	∗4	○
	[%rb], %rs	0	0	1	0	0	1	rs		0	0	1	1				rb	A[rb](23:0)←rs(23:0), A[rb](31:24)←0	1, 2∗8	–	–	–	–	–	–	∗1	○
	[%rb]+, %rs	0	0	1	0	0	1	rs		0	1	1	1				rb	A[rb](23:0)←rs(23:0), A[rb](31:24)←0, rb(23:0)←rb(23:0)+4	2	–	–	–	–	–	–	∗6	○
	[%rb]-, %rs	0	0	1	0	0	1	rs		1	1	1	1				rb	A[rb](23:0)←rs(23:0), A[rb](31:24)←0, rb(23:0)←rb(23:0)-4	2	–	–	–	–	–	–	∗6	○
	-[%rb], %rs	0	0	1	0	0	1	rs		1	0	1	1				rb	rb(23:0)←rb(23:0)-4, A[rb](23:0)←rs(23:0), A[rb](31:24)←0	2	–	–	–	–	–	–	∗6	○
	[%sp+imm7], %rs	1	1	1	1	1	1	rs						imm7				A[sp+imm7](23:0)←rs(23:0), A[sp+imm7](31:24)←0	2	–	–	–	–	–	–	∗5	○
	[imm7], %rs	1	1	0	1	1	1	rs						imm7				A[imm7](23:0)←rs(23:0), A[imm7](31:24)←0	1	–	–	–	–	–	–	∗4	○
	%rd, %sp	0	0	1	1	1	1	rd		0	0	1	0	0	0	0		rd(23:2)←sp(23:2), rd(1:0)←0	1	–	–	–	–	–	–	–	○
	%rd, %pc (∗7)	0	0	1	1	1	1	rd		0	1	1	0	0	0	0		rd(23:0)←pc(23:0)+2	1	–	–	–	–	–	–	–	○
	%rd, [%sp]	0	0	1	1	1	1	rd		0	0	1	1	0	0	0		rd(23:0)←A[sp](23:0), 無視←A[sp](31:24)	1, 2∗8	–	–	–	–	–	–	∗1	○
	%rd, [%sp]+	0	0	1	1	1	1	rd		0	1	1	1	0	0	0		rd(23:0)←A[sp](23:0), 無視←A[sp](31:24), sp(23:0)←sp(23:0)+4	2	–	–	–	–	–	–	∗6	○
	%rd, [%sp]-	0	0	1	1	1	1	rd		1	1	1	1	0	0	0		rd(23:0)←A[sp](23:0), 無視←A[sp](31:24), sp(23:0)←sp(23:0)-4	2	–	–	–	–	–	–	∗6	○
	%rd, -[%sp]	0	0	1	1	1	1	rd		1	0	1	1	0	0	0		sp(23:0)←sp(23:0)-4, rd(23:0)←A[sp](23:0), 無視←A[sp](31:24)	2	–	–	–	–	–	–	∗6	○
	[%sp], %rs	0	0	1	1	1	1	rs		0	0	1	1	1	0	0		A[sp](23:0)←rs(23:0), A[sp](31:24)←0	1, 2∗8	–	–	–	–	–	–	∗1	○
	[%sp]+, %rs	0	0	1	1	1	1	rs		0	1	1	1	1	0	0		A[sp](23:0)←rs(23:0), A[sp](31:24)←0, sp(23:0)←sp(23:0)+4	2	–	–	–	–	–	–	∗6	○
	[%sp]-, %rs	0	0	1	1	1	1	rs		1	1	1	1	1	0	0		A[sp](23:0)←rs(23:0), A[sp](31:24)←0, sp(23:0)←sp(23:0)-4	2	–	–	–	–	–	–	∗6	○
	-[%sp], %rs	0	0	1	1	1	1	rs		1	0	1	1	1	0	0		sp(23:0)←sp(23:0)-4, A[sp](23:0)←rs(23:0), A[sp](31:24)←0	2	–	–	–	–	–	–	∗6	○
	%sp, %rs	0	0	1	1	1	1	rs		1	0	1	0	0	0	0		sp(23:2)←rs(23:2), sp(1:0)←0	1	–	–	–	–	–	–	–	○
	%sp, imm7	1	0	1	1	1	1	0	0	0				imm7				sp(6:2)←imm7(6:2), sp(23:7)←0, sp(1:0)←0	1	–	–	–	–	–	–	∗3	○

備　考

∗1)　EXT 1個使用: ベースアドレス = rb+imm13, EXT 2個使用: ベースアドレス = rb+imm24

∗2)　EXT 1個使用: データ = sign16

∗3)　EXT 1個使用: データ = imm20, EXT 2個使用: データ = imm24

∗4)　EXT 1個使用: ベースアドレス = imm20, EXT 2個使用: ベースアドレス = imm24

∗5)　EXT 1個使用: ベースアドレス = sp+imm20, EXT 2個使用: ベースアドレス = sp+imm24

∗6)　EXT 1個使用: ベースアドレス = rb, アドレスインクリメント/デクリメント rb/sp ← rb/sp±imm13, EXT 2個使用: ベースアドレス = rb, アドレスインクリメント/デクリメント rb/sp ← rb/sp±imm24

∗7)　"ld.a %rd,%pc"命令はjr*.d、 jpr.d、またはjpa.dディレイド分岐命令用のディレイドスロット命令として使用してください。

∗8)　EXT未使用時: 1サイクル, EXT使用時: 2サイクル

整数算術演算命令 (1)

ニーモニック		コード																機　能	サイクル	フラグ						EXT	D
オペコード	オペランド	MSB															LSB			IL	IE	C	V	Z	N		
add	%rd, %rs	0	0	1	1	1	0	rd			1	0	0	0	rs			rd(15:0)←rd(15:0)+rs(15:0), rd(23:16)←0	1	–	–	↔	↔	↔	↔	*1	○
add/c	%rd, %rs	0	0	1	1	1	0	rd			0	0	0	0	rs			rd(15:0)←rd(15:0)+rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	1	–	–	–	↔	↔	↔	*1	○
add/nc	%rd, %rs	0	0	1	1	1	0	rd			0	1	0	0	rs			rd(15:0)←rd(15:0)+rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	1	–	–	–	↔	↔	↔	*1	○
add	%rd, imm7	1	0	0	0	0	0	rd			imm7							rd(15:0)←rd(15:0)+imm7(ゼロ拡張), rd(23:16)←0	1	–	–	↔	↔	↔	↔	*3	○
add.a	%rd, %rs	0	0	1	1	0	0	rd			1	0	0	0	rs			rd(23:0)←rd(23:0)+rs(23:0)	1	–	–	–	–	–	–	*2	○
add.a/c	%rd, %rs	0	0	1	1	0	0	rd			0	0	0	0	rs			rd(23:0)←rd(23:0)+rs(23:0) if C = 1 (nop if C = 0)	1	–	–	–	–	–	–	*2	○
add.a/nc	%rd, %rs	0	0	1	1	0	0	rd			0	1	0	0	rs			rd(23:0)←rd(23:0)+rs(23:0) if C = 0 (nop if C = 1)	1	–	–	–	–	–	–	*2	○
add.a	%sp, %rs	0	0	1	1	0	0	0	0	0	0	0	0	1	rs			sp(23:0)←sp(23:0)+rs(23:0)	1	–	–	–	–	–	–	*2	○
	%rd, imm7	0	1	1	0	0	0	rd			imm7							rd(23:0)←rd(23:0)+imm7(ゼロ拡張)	1	–	–	–	–	–	–	*4	○
	%sp, imm7	0	1	1	0	0	1	0	0	0	imm7							sp(23:0)←sp(23:0)+imm7(ゼロ拡張)	1	–	–	–	–	–	–	*4	○
adc	%rd, %rs	0	0	1	1	1	0	rd			1	0	0	1	rs			rd(15:0)←rd(15:0)+rs(15:0)+C, rd(23:16)←0	1	–	–	↔	↔	↔	↔	*1	○
adc/c	%rd, %rs	0	0	1	1	1	0	rd			0	0	0	1	rs			rd(15:0)←rd(15:0)+rs(15:0)+C, rd(23:16)←0 if C = 1 (nop if C = 0)	1	–	–	–	↔	↔	↔	*1	○
adc/nc	%rd, %rs	0	0	1	1	1	0	rd			0	1	0	1	rs			rd(15:0)←rd(15:0)+rs(15:0)+C, rd(23:16)←0 if C = 0 (nop if C = 1)	1	–	–	–	↔	↔	↔	*1	○
adc	%rd, imm7	1	0	0	0	0	1	rd			imm7							rd(15:0)←rd(15:0)+imm7(ゼロ拡張)+C, rd(23:16)←0	1	–	–	↔	↔	↔	↔	*3	○
sub	%rd, %rs	0	0	1	1	1	0	rd			1	0	1	0	rs			rd(15:0)←rd(15:0)-rs(15:0), rd(23:16)←0	1	–	–	↔	↔	↔	↔	*1	○
sub/c	%rd, %rs	0	0	1	1	1	0	rd			0	0	1	0	rs			rd(15:0)←rd(15:0)-rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	1	–	–	–	↔	↔	↔	*1	○
sub/nc	%rd, %rs	0	0	1	1	1	0	rd			0	1	1	0	rs			rd(15:0)←rd(15:0)-rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	1	–	–	–	↔	↔	↔	*1	○
sub	%rd, imm7	1	0	0	0	1	0	rd			imm7							rd(15:0)←rd(15:0)-imm7(ゼロ拡張), rd(23:16)←0	1	–	–	↔	↔	↔	↔	*3	○
sub.a	%rd, %rs	0	0	1	1	0	0	rd			1	0	1	0	rs			rd(23:0)←rd(23:0)-rs(23:0)	1	–	–	–	–	–	–	*2	○
sub.a/c	%rd, %rs	0	0	1	1	0	0	rd			0	0	1	0	rs			rd(23:0)←rd(23:0)-rs(23:0) if C = 1 (nop if C = 0)	1	–	–	–	–	–	–	*2	○
sub.a/nc	%rd, %rs	0	0	1	1	0	0	rd			0	1	1	0	rs			rd(23:0)←rd(23:0)-rs(23:0) if C = 0 (nop if C = 1)	1	–	–	–	–	–	–	*2	○
sub.a	%sp, %rs	0	0	1	1	0	0	0	0	0	0	0	1	1	rs			sp(23:0)←sp(23:0)-rs(23:0)	1	–	–	–	–	–	–	*2	○
	%rd, imm7	0	1	1	0	1	0	rd			imm7							rd(23:0)←rd(23:0)-imm7(ゼロ拡張)	1	–	–	–	–	–	–	*4	○
	%sp, imm7	0	1	1	0	1	1	0	0	0	imm7							sp(23:0)←sp(23:0)-imm7(ゼロ拡張)	1	–	–	–	–	–	–	*4	○
sbc	%rd, %rs	0	0	1	1	1	0	rd			1	0	1	1	rs			rd(15:0)←rd(15:0)-rs(15:0)-C, rd(23:16)←0	1	–	–	↔	↔	↔	↔	*1	○
sbc/c	%rd, %rs	0	0	1	1	1	0	rd			0	0	1	1	rs			rd(15:0)←rd(15:0)-rs(15:0)-C, rd(23:16)←0 if C = 1 (nop if C = 0)	1	–	–	–	↔	↔	↔	*1	○
sbc/nc	%rd, %rs	0	0	1	1	1	0	rd			0	1	1	1	rs			rd(15:0)←rd(15:0)-rs(15:0)-C, rd(23:16)←0 if C = 0 (nop if C = 1)	1	–	–	–	↔	↔	↔	*1	○
sbc	%rd, imm7	1	0	0	0	1	1	rd			imm7							rd(15:0)←rd(15:0)-imm7(ゼロ拡張)-C, rd(23:16)←0	1	–	–	↔	↔	↔	↔	*3	○
cmp	%rd, %rs	0	0	1	1	1	1	rd			1	0	0	0	rs			rd(15:0)-rs(15:0)	1	–	–	↔	↔	↔	↔	*1	○
cmp/c	%rd, %rs	0	0	1	1	1	1	rd			0	0	0	0	rs			rd(15:0)-rs(15:0) if C = 1 (nop if C = 0)	1	–	–	–	↔	↔	↔	*1	○
cmp/nc	%rd, %rs	0	0	1	1	1	1	rd			0	1	0	0	rs			rd(15:0)-rs(15:0) if C = 0 (nop if C = 1)	1	–	–	–	↔	↔	↔	*1	○
cmp	%rd, sign7	1	0	0	1	0	0	rd			sign7							rd(15:0)-sign7(符号拡張)	1	–	–	↔	↔	↔	↔	*3	○

備　考

*1)　EXT 1個使用: rd ← rs <op> imm13, EXT 2個使用: rd ← rs <op> imm16

*2)　EXT 1個使用: rd ← rs <op> imm13, EXT 2個使用: rd ← rs <op> imm24

*3)　EXT 1個使用: データ = imm16/sign16

*4)　EXT 1個使用: データ = imm20, EXT 2個使用: データ = imm24

整数算術演算命令 (2)

ニーモニック		コード														機　能	サイクル	フラグ						EXT	D
オペコード	オペランド	MSB													LSB			IL	IE	C	V	Z	N		
cmp.a	%rd, %rs	0	0	1	1	0	1	rd			1	0	0	0	rs	rd(23:0)-rs(23:0)	1	–	–	↔	–	↔	–	∗2	○
cmp.a/c	%rd, %rs	0	0	1	1	0	1	rd			0	0	0	0	rs	rd(23:0)-rs(23:0) if C = 1 (nop if C = 0)	1	–	–	–	–	↔	–	∗2	○
cmp.a/nc	%rd, %rs	0	0	1	1	0	1	rd			0	1	0	0	rs	rd(23:0)-rs(23:0) if C = 0 (nop if C = 1)	1	–	–	–	–	↔	–	∗2	○
cmp.a	%rd, imm7	0	1	1	1	0	0	rd					imm7			rd(23:0)-imm7(ゼロ拡張)	1	–	–	↔	–	↔	–	∗4	○
cmc	%rd, %rs	0	0	1	1	1	1	rd			1	0	0	1	rs	rd(15:0)-rs(15:0)-C	1	–	–	↔	↔	↔	↔	∗1	○
cmc/c	%rd, %rs	0	0	1	1	1	1	rd			0	0	0	1	rs	rd(15:0)-rs(15:0)-C if C = 1 (nop if C = 0)	1	–	–	–	↔	↔	↔	∗1	○
cmc/nc	%rd, %rs	0	0	1	1	1	1	rd			0	1	0	1	rs	rd(15:0)-rs(15:0)-C if C = 0 (nop if C = 1)	1	–	–	–	↔	↔	↔	∗1	○
cmc	%rd, sign7	1	0	0	1	0	1	rd					sign7			rd(15:0)-sign7(符号拡張)-C	1	–	–	↔	↔	↔	↔	∗3	○

備　考

∗1)　EXT 1個使用: rd ← rs <op> imm13, EXT 2個使用: rd ← rs <op> imm16

∗2)　EXT 1個使用: rd ← rs <op> imm13, EXT 2個使用: rd ← rs <op> imm24

∗3)　EXT 1個使用: データ = imm16/sign16

∗4)　EXT 1個使用: データ = imm20, EXT 2個使用: データ = imm24

論理演算命令

ニーモニック		コード														機　能	サイクル	フラグ						EXT	D
オペコード	オペランド	MSB													LSB			IL	IE	C	V	Z	N		
and	%rd, %rs	0	0	1	0	1	1	rd			1	0	0	0	rs	rd(15:0)←rd(15:0)&rs(15:0), rd(23:16)←0	1	–	–	–	0	↔	↔	∗1	○
and/c	%rd, %rs	0	0	1	0	1	1	rd			0	0	0	0	rs	rd(15:0)←rd(15:0)&rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	1	–	–	–	0	↔	↔	∗1	○
and/nc	%rd, %rs	0	0	1	0	1	1	rd			0	1	0	0	rs	rd(15:0)←rd(15:0)&rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	1	–	–	–	0	↔	↔	∗1	○
and	%rd, sign7	1	0	1	0	0	0	rd					sign7			rd(15:0)←rd(15:0)&sign7(符号拡張), rd(23:16)←0	1	–	–	–	0	↔	↔	∗2	○
or	%rd, %rs	0	0	1	0	1	1	rd			1	0	0	1	rs	rd(15:0)←rd(15:0) I rs(15:0), rd(23:16)←0	1	–	–	–	0	↔	↔	∗1	○
or/c	%rd, %rs	0	0	1	0	1	1	rd			0	0	0	1	rs	rd(15:0)←rd(15:0) I rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	1	–	–	–	0	↔	↔	∗1	○
or/nc	%rd, %rs	0	0	1	0	1	1	rd			0	1	0	1	rs	rd(15:0)←rd(15:0) I rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	1	–	–	–	0	↔	↔	∗1	○
or	%rd, sign7	1	0	1	0	0	1	rd					sign7			rd(15:0)←rd(15:0) I sign7(符号拡張), rd(23:16)←0	1	–	–	–	0	↔	↔	∗2	○
xor	%rd, %rs	0	0	1	0	1	1	rd			1	0	1	0	rs	rd(15:0)←rd(15:0)^rs(15:0), rd(23:16)←0	1	–	–	–	0	↔	↔	∗1	○
xor/c	%rd, %rs	0	0	1	0	1	1	rd			0	0	1	0	rs	rd(15:0)←rd(15:0)^rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	1	–	–	–	0	↔	↔	∗1	○
xor/nc	%rd, %rs	0	0	1	0	1	1	rd			0	1	1	0	rs	rd(15:0)←rd(15:0)^rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	1	–	–	–	0	↔	↔	∗1	○
xor	%rd, sign7	1	0	1	0	1	0	rd					sign7			rd(15:0)←rd(15:0)^sign7(符号拡張), rd(23:16)←0	1	–	–	–	0	↔	↔	∗2	○
not	%rd, %rs	0	0	1	0	1	1	rd			1	0	1	1	rs	rd(15:0)←!rs(15:0), rd(23:16)←0	1	–	–	–	0	↔	↔	∗3	○
not/c	%rd, %rs	0	0	1	0	1	1	rd			0	0	1	1	rs	rd(15:0)←!rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	1	–	–	–	0	↔	↔	∗3	○
not/nc	%rd, %rs	0	0	1	0	1	1	rd			0	1	1	1	rs	rd(15:0)←!rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	1	–	–	–	0	↔	↔	∗3	○
not	%rd, sign7	1	0	1	0	1	1	rd					sign7			rd(15:0)←!sign7(符号拡張), rd(23:16)←0	1	–	–	–	0	↔	↔	∗2	○

備　考

∗1)　EXT 1個使用: rd ← rs <op> imm13, EXT 2個使用: rd ← rs <op> imm16

∗2)　EXT 1個使用: データ = sign16

∗3)　EXT 1個使用: rd ← !imm13, EXT 2個使用: rd ← !imm16

分岐命令

S1C17 Core Instruction Set

ニーモニック		コード																機　能	サイクル	フラグ						EXT	D
オペコード	オペランド	MSB															LSB		∗6	IL	IE	C	V	Z	N		
jpr / jpr.d	sign10	0	0	0	1	0	d	sign10										pc←pc+2+sign11; sign11={sign10,0} (∗3)	3	–	–	–	–	–	–	∗4	–
	%rb	0	0	0	0	0	0	0	1	d	1	0	0	0	rb			pc←pc+2+rb (∗3)	2(.d)	–	–	–	–	–	–	–	–
jpa / jpa.d	imm7	0	0	0	0	0	0	1	1	d	imm7							pc←imm7 (∗3)	3	–	–	–	–	–	–	∗2	–
	%rb	0	0	0	0	0	0	0	1	d	1	0	0	1	rb			pc←rb (∗3)	2(.d)	–	–	–	–	–	–	–	–
jrgt / jrgt.d	sign7	0	0	0	0	0	1	1	0	d	sign7							pc←pc+2+sign8 if !Z&!(N^V) is true; sign8={sign7,0} (∗3)		–	–	–	–	–	–	∗1	–
jrge / jrge.d	sign7	0	0	0	0	0	1	1	1	d	sign7							pc←pc+2+sign8 if !(N^V) is true; sign8={sign7,0} (∗3)	2	–	–	–	–	–	–	∗1	–
jrlt / jrlt.d	sign7	0	0	0	0	1	0	0	0	d	sign7							pc←pc+2+sign8 if N^V is true; sign8={sign7,0} (∗3)	(false)	–	–	–	–	–	–	∗1	–
jrle / jrle.d	sign7	0	0	0	0	1	0	0	1	d	sign7							pc←pc+2+sign8 if Z ! (N^V) is true; sign8={sign7,0} (∗3)	or	–	–	–	–	–	–	∗1	–
jrugt / jrugt.d	sign7	0	0	0	0	1	0	1	0	d	sign7							pc←pc+2+sign8 if !Z&!C is true; sign8={sign7,0} (∗3)	3	–	–	–	–	–	–	∗1	–
jruge / jruge.d	sign7	0	0	0	0	1	0	1	1	d	sign7							pc←pc+2+sign8 if !C is true; sign8={sign7,0} (∗3)	(true)	–	–	–	–	–	–	∗1	–
jrult / jrult.d	sign7	0	0	0	0	1	1	0	0	d	sign7							pc←pc+2+sign8 if C is true; sign8={sign7,0} (∗3)	∗5	–	–	–	–	–	–	∗1	–
jrule / jrule.d	sign7	0	0	0	0	1	1	0	1	d	sign7							pc←pc+2+sign8 if Z ! C is true; sign8={sign7,0} (∗3)		–	–	–	–	–	–	∗1	–
jreq / jreq.d	sign7	0	0	0	0	1	1	1	0	d	sign7							pc←pc+2+sign8 if Z is true; sign8={sign7,0} (∗3)	2(.d)	–	–	–	–	–	–	∗1	–
jrne / jrne.d	sign7	0	0	0	0	1	1	1	1	d	sign7							pc←pc+2+sign8 if !Z is true; sign8={sign7,0} (∗3)		–	–	–	–	–	–	∗1	–
call / call.d	sign10	0	0	0	1	1	d	sign10										sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←pc+2+sign11; sign11={sign10,0} (∗3)	4	–	–	–	–	–	–	∗4	–
	%rb	0	0	0	0	0	0	0	1	d	0	0	0	0	rb			sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←pc+2+rb (∗3)	3(.d)	–	–	–	–	–	–	–	–
calla / calla.d	imm7	0	0	0	0	0	1	0	1	d	imm7							sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←imm7 (∗3)	4	–	–	–	–	–	–	∗2	–
	%rb	0	0	0	0	0	0	0	1	d	0	0	0	1	rb			sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←rb (∗3)	3(.d)	–	–	–	–	–	–	–	–
ret / ret.d		0	0	0	0	0	0	0	1	d	0	1	0	0	0	0	0	pc←A[sp](23:0), sp←sp+4 (∗3)	3, 2(.d)	–	–	–	–	–	–	–	–
int	imm5	0	1	1	1	0	1	0	0	0	imm5					0	1	sp←sp-4, A[sp]←{psr, pc+2}, pc←ベクタ(TTBR+imm5×4)	3	–	0	–	–	–	–	–	–
intl	imm5, imm3	0	1	1	1	0	1	imm3			imm5					1	1	sp←sp-4, A[sp]←{psr, pc+2}, pc←ベクタ(TTBR+imm5×4), psr(IL)←imm3	3	↔	0	–	–	–	–	–	–
reti / reti.d		0	0	0	0	0	0	0	1	d	0	1	0	1	0	0	0	{psr, pc}←A[sp], sp←sp+4	3, 2(.d)	↔	↔	↔	↔	↔	↔	–	–
brk		0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	A[DBRAM]←{psr, pc+2}, A[DBRAM+4]←r0, pc←0xfffc00	4	–	0	–	–	–	–	–	–
retd		0	0	0	0	0	0	0	1	0	1	1	0	1	0	0	0	r0←A[DBRAM+4](23:0), {psr, pc}←A[DBRAM]	4	↔	↔	↔	↔	↔	↔	–	–

備　考

∗1) EXT 1個使用: ディスプレースメント = sign21 (= {imm13, sign7, 0}), EXT 2個使用: ディスプレースメント = sign24 (= {1st imm13(2:0), 2nd imm13, sign7, 0})
∗2) EXT 1個使用: 絶対アドレス= sign20 (= {imm13, imm7}), EXT 2個使用: 絶対アドレス = sign24 (= {1st imm13(3:0), 2nd imm13, imm7})
∗3) これらの命令は".d"付きのオペコード(jrgt.d, call.d等)を指定することによりコード内のdビットが1にセットされ、ディレイド分岐命令となります。
∗4) EXT 1個使用: ディスプレースメント = sign24 (= {imm13, sign10, 0})
∗5) ディレイドスロット命令を伴わない条件分岐命令(".d"なし)は、分岐しないときは2サイクル、分岐するときは3サイクルで実行されます。
∗6) (.d)付きのサイクル数はディレイドスロット命令が1サイクルの場合の値です。それ以外の場合は、(.d)なしと同じサイクル数となります。

即値拡張命令

S1C17 Core Instruction Set

ニーモニック		コード																機　能	サイクル	フラグ						EXT	D
オペコード	オペランド	MSB															LSB			IL	IE	C	V	Z	N		
ext	imm13	0	1	0	imm13													次の命令の即値またはオペランドを拡張	1	–	–	–	–	–	–	∗1	–

備　考

∗1) 拡張可能な命令の前に1個または2個のext命令を置くことができます。

シフト&スワップ命令

S1C17 Core Instruction Set

ニーモニック		コード																機　能	サイクル	フラグ						EXT	D
オペコード	オペランド	MSB															LSB			IL	IE	C	V	Z	N		
sr	%rd, %rs	0	0	1	0	1	1	rd				1	1	0	0	rs		右論理シフト: rd(15:0)←rd(15:0)>>rs(15:0), rd(23:16)←0, MSB←0 (*1)	1	–	–	↔	–	↔	↔	–	○
	%rd, imm7	1	0	1	1	0	0	rd				imm7						右論理シフト: rd(15:0)←rd(15:0)>>imm7, rd(23:16)←0, MSB←0 (*1)	1	–	–	↔	–	↔	↔	*2	○
sa	%rd, %rs	0	0	1	0	1	1	rd				1	1	0	1	rs		右算術シフト: rd(15:0)←rd(15:0)>>rs(15:0), rd(23:16)←0, MSB←符号 (*1)	1	–	–	↔	–	↔	↔	–	○
	%rd, imm7	1	0	1	1	0	1	rd				imm7						右算術シフト: rd(15:0)←rd(15:0)>>imm7, rd(23:16)←0, MSB←符号 (*1)	1	–	–	↔	–	↔	↔	*2	○
sl	%rd, %rs	0	0	1	0	1	1	rd				1	1	1	0	rs		左論理シフト: rd(15:0)←rd(15:0)<<rs(15:0), rd(23:16)←0, LSB←0 (*1)	1	–	–	↔	–	↔	↔	–	○
	%rd, imm7	1	0	1	1	1	0	rd				imm7						左論理シフト: rd(15:0)←rd(15:0)<<imm7, rd(23:16)←0, LSB←0 (*1)	1	–	–	↔	–	↔	↔	*2	○
swap	%rd, %rs	0	0	1	0	1	1	rd				1	1	1	1	rs		rd(15:8)←rs(7:0), rd(7:0)←rs(15:8), rd(23:16)←0	1	–	–	–	–	–	–	–	○

備　考

*1) シフト量: rs/imm7 = 0〜3の場合は0〜3ビット, rs/imm7 = 4〜7の場合は4ビット, rs/imm7 = 8以上の場合は8ビット

*2) EXT 1個使用: 即値 = imm20, EXT 2個使用: 即値 = imm24

コンバージョン命令

S1C17 Core Instruction Set

ニーモニック		コード																機　能	サイクル	フラグ						EXT	D
オペコード	オペランド	MSB															LSB			IL	IE	C	V	Z	N		
cv.ab	%rd, %rs	0	0	1	0	1	0	rd				0	1	1	1	rs		rd(23:8)←rs(7), rd(7:0)←rs(7:0)	1	–	–	–	–	–	–	–	○
cv.as	%rd, %rs	0	0	1	0	1	0	rd				1	0	1	1	rs		rd(23:16)←rs(15), rd(15:0)←rs(15:0)	1	–	–	–	–	–	–	–	○
cv.al	%rd, %rs	0	0	1	0	1	0	rd				1	1	1	1	rs		rd(23:16)←rs(7:0), rd(15:0)←rd(15:0)	1	–	–	–	–	–	–	–	○
cv.la	%rd, %rs	0	0	1	0	1	0	rd				0	1	1	0	rs		rd(23:8)←0, rd(7:0)←rs(23:16)	1	–	–	–	–	–	–	–	○
cv.ls	%rd, %rs	0	0	1	0	1	0	rd				1	0	1	0	rs		rd(23:16)←0, rd(15:0)←rs(15)	1	–	–	–	–	–	–	–	○

システム制御命令

S1C17 Core Instruction Set

ニーモニック		コード																機　能	サイクル	フラグ						EXT	D
オペコード	オペランド	MSB															LSB			IL	IE	C	V	Z	N		
nop		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ノーオペレーション	1	–	–	–	–	–	–	–	○
halt		0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	HALT	6	–	–	–	–	–	–	–	–
slp		0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	SLEEP	6	–	–	–	–	–	–	–	–
ei		0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	psr(IE)←1	1	–	1	–	–	–	–	–	○
di		0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	psr(IE)←0	1	–	0	–	–	–	–	–	○

コプロセッサインタフェース命令

ニーモニック		コード																機　能	サイクル	フラグ						EXT	D
オペコード	オペランド	MSB															LSB			IL	IE	C	V	Z	N		
ld.cw	%rd, %rs	0	0	1	1	0	1	rd			0	0	1	0	rs			co_dout0←rd, co_dout1←rs	1	–	–	–	–	–	–	–	○
	%rd, imm7	0	1	1	1	1	0	rd			imm7							co_dout0←rd, co_dout1←imm7	1	–	–	–	–	–	–	∗1	○
ld.ca	%rd, %rs	0	0	1	1	0	1	rd			0	0	1	1	rs			co_dout0←rd, co_dout1←rs, rd←co_din, psr(C, V, Z, N)←co_cvzn	1	–	–	↔	↔	↔	↔	–	○
	%rd, imm7	0	1	1	1	1	1	rd			imm7							co_dout0←rd, co_dout1←imm7, rd←co_din, psr(C, V, Z, N)←co_cvzn	1	–	–	↔	↔	↔	↔	∗1	○
ld.cf	%rd, %rs	0	0	1	1	0	1	rd			0	0	0	1	rs			co_dout0←rd, co_dout1←rs, psr(C, V, Z, N)←co_cvzn	1	–	–	↔	↔	↔	↔	–	○
	%rd, imm7	1	1	0	1	0	1	rd			imm7							co_dout0←rd, co_dout1←imm7, psr(C, V, Z, N)←co_cvzn	1	–	–	↔	↔	↔	↔	∗1	○

備　考

∗1)　EXT 1個使用: co_dout1出力 = imm20, EXT 2個使用: co_dout1出力 = imm24

改訂履歴表

コードNo.	ページ	改訂内容（旧内容を含む） および改訂理由
410905700	全ページ	新規制定
410905701	全ページ	全面改訂
410905702	6-10	“HALT, SLEEPモードの解除”の説明修正
	7-32, 7-33	di、ei命令説明に注意を追加

セイコーエプソン株式会社
営業本部　デバイス営業部

東京　〒191-8501　東京都日野市日野421-8
TEL（042）587-5313（直通）　FAX（042）587-5116

大阪　〒541-0059　大阪市中央区博労町3-5-1　御堂筋グランタワー 15F
TEL（06）6120-6000（代表）　FAX（06）6120-6100

ドキュメントコード：410905702
2007年 4月 作成
2018年 1月 改訂 Ⓛ