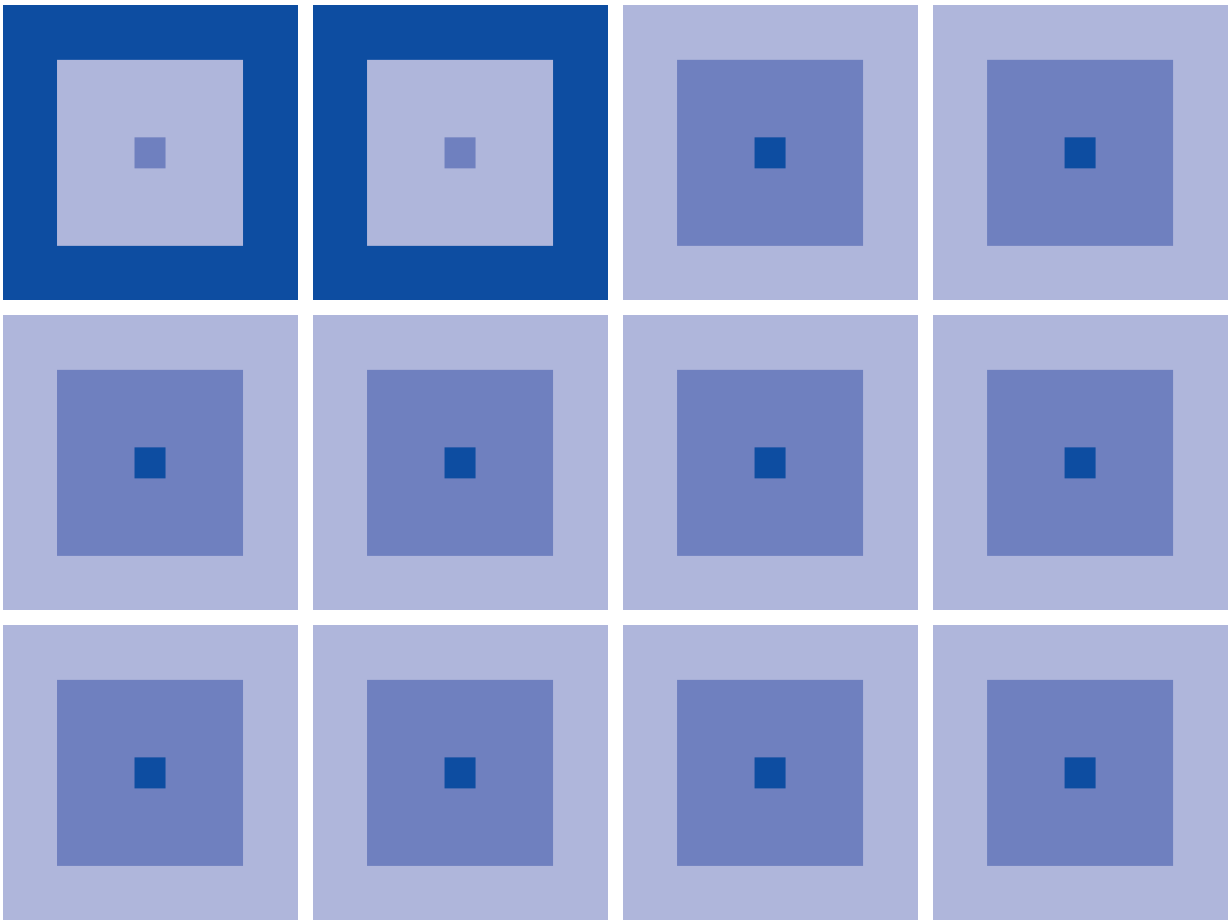
MF1096-02a

EPSON

CMOS 32-BIT SINGLE CHIP MICROCOMPUTER

S1C33000
コアCPUマニュアル



セイコーエプソン株式会社

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。

1. 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。
2. 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します。
3. 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権利(工業所有権を含む)侵害あるいは損害の発生に対し、弊社は如何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工業所有権の実施権の許諾を行うものではありません。
4. 特性表の数値の大小は、数直線上の大小関係で表しています。
5. 本資料に掲載されている製品のうち、「外国為替および外国貿易法」に定める戦略物資に該当するものについては、輸出する場合、同法に基づく輸出許可が必要です。
6. 本資料に掲載されている製品は、一般民生用です。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如何なる責任についても負いかねます。

CMOS 32-BIT SINGLE CHIP MICROCOMPUTER

S1C33000 Core CPU Manual

このマニュアルは、32ビットシングルチップマイクロコンピュータS1C33 Family各機種のコアとして使用されるRISC型32ビットCPU〈S1C33000〉の機能と命令セットについて解説しています。
チップ上の周辺回路を含むハードウェアの詳細については、S1C33 Family各機種のテクニカルマニュアルを参照してください。

データの表記について

本マニュアルでは、データサイズや数値を以下のように記述しています。

データサイズ

8ビット: バイト, Byte, B
16ビット: ハーフワード, Half word, H
32ビット: ワード, Word, W

数値の表記

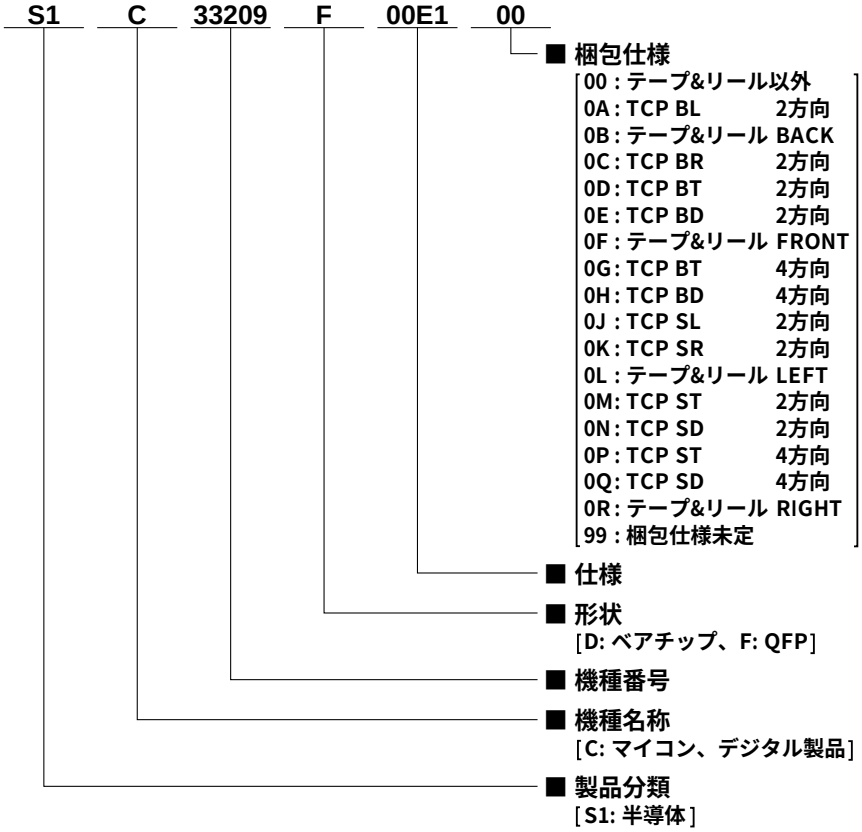
16進数: 0x0000000, 0xFF等
2進数: 0b0000, 0b1111等
その他の数値は基本的に10進数です。ただし、明らかに2進数と判別できるものについては0bを省略している場合もあります。

命　令

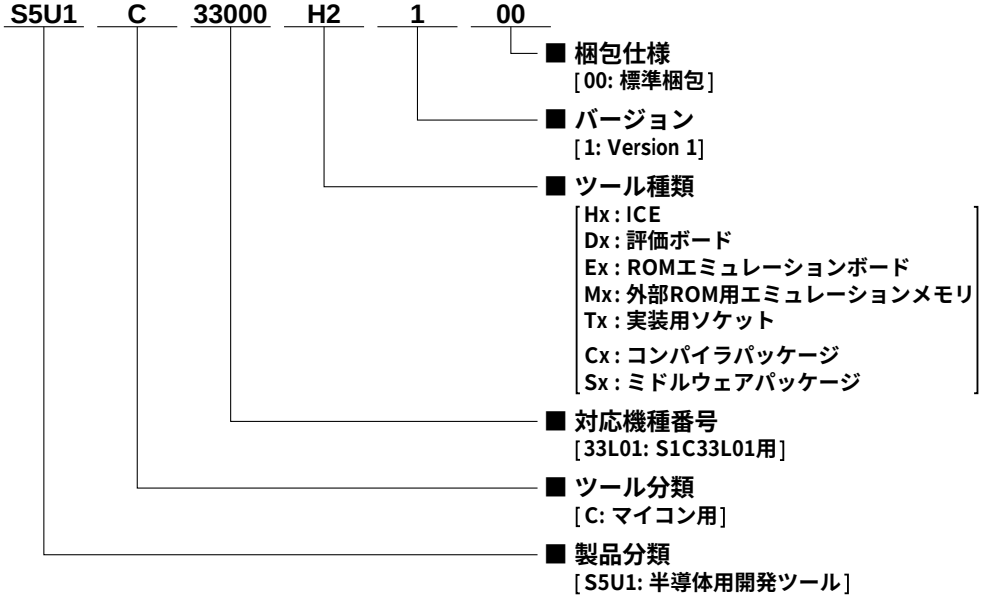
命令や記述例は基本的に小文字(a～z)を使用しています。実際に記述する場合は大文字も使用することができます。命令のオペランドや機能説明に使用するシンボルについては、"4.1 記号の意味"を参照してください。

製品型番体系

●デバイス



●開発ツール



－ 目 次 －

1 概要

S1C33000は32ビットRISC型CPUで、32ビットシングルチップマイクロコンピュータS1C33 Familyの全機種にコアCPUとして使用されます。
S1C33000はセイコーエプソンオリジナルアーキテクチャのもとに、低消費電力、高速動作を要求される組み込み型アプリケーションへの応用を目的として開発され、各種携帯機器からFA/OA機器まで広範囲に対応します。
パイプライン処理とロード・ストア型アーキテクチャにより、動作周波数を上回るMIPS値を達成しました。命令セットはC言語による開発用に最適化されており、Cコンパイラから非常にコンパクトなオブジェクトコードを生成することができます。また、オプションにより搭載可能な乗算器と積和(MAC)命令により、オンチップでDSP機能を実現します。
S1C33000を搭載するS1C33 FamilyマイクロコンピュータはROM、RAM、各種高性能周辺機能ブロックをチップ上に集積し、お客様の必要とするほぼすべての機能を1チップで実現します。

1.1 特長

プロセッサ形式

・セイコーエプソンオリジナル32ビットRISC CPU
・32ビット長の内部データ処理

動作周波数

・DC～33MHz(S1C33 Familyの機種により異なります。)

命令セット

・コード長: 16ビット/インストラクション(固定長)
・命令数: 105命令
・主要な命令は1サイクルで実行
・即値拡張命令によりコード内の即値を32ビットまで拡張可能

積和演算機能

・64ビットの積和演算(MAC命令)が可能(16ビット×16ビット＋64ビット)

レジスタセット

・32ビット汎用レジスタ 16本
・32ビット特殊レジスタ 3本
・32ビット積和演算用レジスタ 2本

メモリ空間、外部バス

・命令、データ、I/O混在型のリニア空間
・最大256MB(28ビット)のメモリ空間をアクセス可能
・8、16ビットの外部デバイスに対応可能
・外部グルーロジックを不要とする、19エリアのセレクト信号を出力可能
・DRAM等を直接駆動可能(S1C33 Familyの機種により異なります。)
・ハーバードアーキテクチャ
・リトルエンディアン形式

割り込み

・リセット、NMI、128本の外部割り込みに対応
・ソフトウェア例外 4本、命令実行例外 2種
・トラップテーブルからベクタを読み込み各処理ルーチンへ直接分岐

リセット

・コールドリセット(すべてをリセット)
・ホットリセット(バスとポートの状態を保持)

パワーダウンモード

・HALTモード(コアCPUのみ停止)
・SLEEPモード(コアCPUと高速発振回路を停止)

1.2 ブロック図

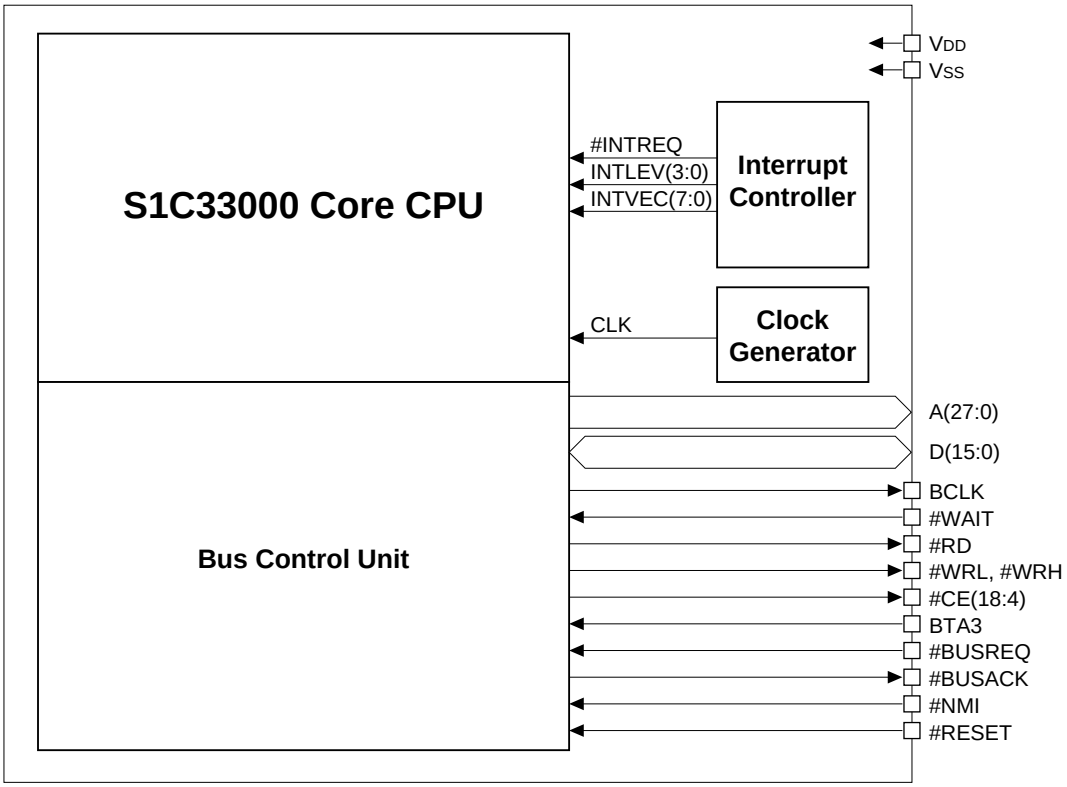


図1.2.1 ブロック図

図は主要なブロックと信号の概要で、実際の回路構成を示すものではありません。
実際のS1C33 Familyの各機種は、上記ブロックを中心に周辺回路ブロックをチップ上に内蔵します。

1.3 入出力信号

表1.3.1にS1C33000コアの動作に関係する主要な入出力信号を示します。

表1.3.1 入出力信号

信号名	I/O	説　　明
VDD	I	＋電源(電源電圧は機種により異なります。)
VSS	I	－電源(GND)
CLK(内部信号)	I	入力クロック(クロック周波数は機種により異なります。)
BCLK	O	バスクロック バスサイクルのクロックが出力されます。
D(15:0)	I/O	データバス 双方向の16ビットデータバスです。
A(27:0)	O	アドレスバス 28ビットのアドレスバスです。
#WAIT	I	ウェイトサイクル要求信号 低速デバイスがCPUに対して出力する信号で、CPUはこの信号がアクティブな間バスサイクルを延長し、デバイスの入出力終了を待ちます。
#RD	O	リード信号 データバスからデータを読み込むときに出力されます。バス上の選択されたデバイスは、この信号がアクティブな間にデータをデータバスに出力します。
#WRL #WRH	O	ライト信号 バス上のデバイスにデータを書き込むときに出力されます。この信号がアクティブな間に、選択されたデバイスはデータバスからデータを入力します。 #WRLは下位バイト書き込み用、#WRHは上位バイト書き込み用です。 S1C33000ではバスストローブ方式の信号(#WR/#BSL/#BSH)にも対応しています。
#CE(18:4)	O	チップイネーブル信号 19エリアに分割されたメモリ領域の外部メモリエリアに個々に対応するチップイネーブル信号です。各エリアのデバイスをアクセスする際にアクティブになります。
#RESET	I	イニシャルリセット信号 Lowレベルの入力でCPUをリセットします。 #RESET=0 & #NMI=1: コールドリセット #RESET=0 & #NMI=0: ホットリセット
BTA3	I	ブートアドレス設定信号 ブートアドレスを指定します。 BTA3=1: 内蔵ROM(エリア3)からブート BTA3=0: 外部ROM(エリア10)からブート
#NMI	I	NMI要求信号 マスクが不可能な割り込み要求信号です。この信号の入力により、CPUはトラップ処理に移行します。また、イニシャルリセットの条件指定にも使用されます。
#INTREQ (内部信号)	I	割り込み要求信号 CPUに対するマスク可能な外部割り込み要求信号です。 S1C33 Familyではオンチップ上の割り込みコントローラがこの信号を出力します。 割り込み条件が整っていれば、この信号によってCPUはトラップ処理に移行します。
INTLEV(3:0) (内部信号)	I	割り込みレベル 外部割り込みを要求した周辺回路の割り込みレベルが入力されます。この内容は、CPUが割り込みを受け付けた時点でプロセッサステータスレジスタ(PSR)のILフィールドにセットされ、それ以降の低いレベルの割り込みを禁止します。
INTVEC(7:0) (内部信号)	I	割り込みベクタ番号 外部割り込みを要求した周辺回路のベクタ番号が入力されます。CPUは割り込みを受け付けると、トラップテーブルの指定ベクタを読み込んで割り込み処理ルーチンに分岐します。
#BUSREQ	I	バスリクエスト信号 外部バスマスタが出力するバス権解放要求信号です。
#BUSACK	O	バスアクノリッジ信号 CPUが外部バスマスタのバス権解放要求を受け付けたことを示す信号です。この信号がアクティブな間、CPUはバスをハイインピーダンスにして外部バスマスタに解放します。バス権は外部バスマスタがバス動作を終了して#BUSREQをインアクティブにした段階でCPUに戻ります。

信号名の#はLowアクティブを示します。

実際の入出力信号/端子についてはS1C33 Family各機種のテクニカルマニュアルを参照してください。

2 アーキテクチャ

2.1 レジスタセット

S1C33000は16本の32ビット汎用レジスタおよび5本の特殊レジスタを内蔵しています。

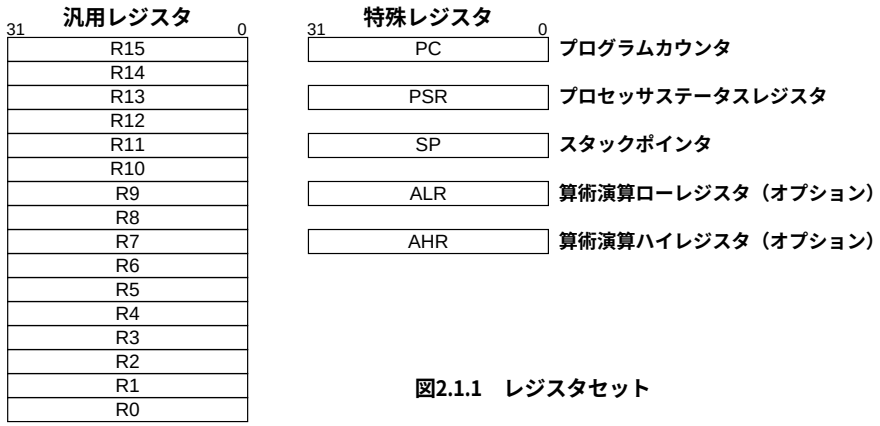


図2.1.1 レジスタセット

2.1.1 汎用レジスタ(R0～R15)

16本のレジスタR0～R15はデータ演算、データ転送、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32ビット長の汎用レジスタです。IC内部では、これらのレジスタの内容はすべて32ビットデータまたはアドレスとして扱われ、それ以下のビット長のデータはレジスタへのロード時に符号拡張またはゼロ拡張されます。また、アドレスとして用いる場合はアドレスバスが28ビットのため、上位4ビットは無効となります。なお、実際の有効アドレスは各機種のメモリ構成によって異なります。
イニシャルリセット時、汎用レジスタの内容は不定となりますので、使用する前にプログラムによる初期化が必要です。

2.1.2 プログラムカウンタ(PC)

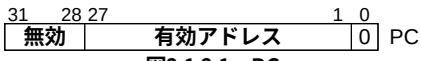


図2.1.2.1 PC

プログラムカウンタ(以降PCと記述)は、実行命令のアドレスを保持する32ビット長のカウンタです。S1C33000命令セットではすべての命令が16ビットの固定長のため、PCの最下位ビット(bit 0)は常に0に固定されます。また、アドレスバスが28ビットのため、上位4ビットは無効となります。なお、実際の有効アドレスは各機種のメモリ構成によって異なります。
PCはプログラムによって直接アクセスすることはできません。PCが変更されるのは以下の場合に限られます。

(1)イニシャルリセット時

イニシャルリセットによりPCにはブートアドレスがロードされ、そのアドレスから命令の実行を開始します。ブートアドレスを格納する番地はBTA3端子の設定により内蔵ROMの0x0080000番地、または外部ROMの0x0C00000番地となります。

(2)各命令の実行時

PCは1命令の実行ごとにインクリメント(+2)され、常に実行中のアドレスを示します。

(3)プログラムの分岐時

ジャンプ、サブルーチンコール、割り込みなどのトラップ処理、あるいはサブルーチンからのリターン等、プログラムの分岐時にはPCに分岐先アドレスがロードされます。
また、サブルーチンコールやトラップ処理等、リターンが必要な処理への分岐時は分岐前にPCの内容がスタックにセーブされ、リターン命令実行時にPCに戻されます。

2.1.3 プロセッサステータスレジスタ(PSR)

プロセッサステータスレジスタ(以降PSRと記述)はCPUの状態を保持する32ビットレジスタで、実行した命令によって変化します。ロード命令による読み出し、書き込みも可能です。
PSRはプログラムの実行にも影響を与えるため、割り込みや例外の発生時は処理ルーチンへの分岐前にPSRの内容がスタックにセーブされ、リターン(RETI)命令実行時にPSRに戻されます。
イニシャルリセット時、PSRの各ビットは0に設定されます。
各ビットの機能を以下に説明します。

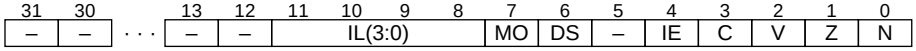


図2.1.3.1 プロセッサステータスレジスタ

"–"は未使用ビットを示します。書き込みは無効で、読み出しは常に0となります。

N(bit 0): ネガティブフラグ

正・負の符号を示します。論理演算・算術演算、シフト命令の実行結果(ディスティネーションレジスタ)の最上位ビット(bit 31)がNフラグにコピーされます。ステップ除算の実行時には除数の符号ビットがNフラグにセットされ、除算の実行に影響を与えます。

Z(bit 1): ゼロフラグ

結果がゼロ(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(ディスティネーションレジスタ)がゼロの場合に1にセットされ、ゼロ以外の場合に0にリセットされます。

V(bit 2): オーバーフローフラグ

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値を符号付き32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・減算結果が符号付き32ビット範囲内の場合は0にリセットされます。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(結果の最上位ビット)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が0(正)になった場合

C(bit 3): キャリーフラグ

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値を符号なし32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32ビット整数の範囲を越えると1にセットされます。結果が符号なし32ビット整数の範囲内の場合は0にリセットされます。Cフラグがセットされる条件は以下のとおりです。
(1) 加算命令で、演算結果が符号なし32ビット整数の最大値0xFFFFFFFFよりも大きい値となる加算を実行した場合
(2) 減算命令で、演算結果が符号なし32ビット整数の最小値0x00000000よりも小さい値となる減算を実行した場合

IE(bit 4): 割り込みイネーブルビット

マスク可能な外部割り込みを受け付けるか禁止するか制御します。IEビットが1のとき、CPUはマスク可能な外部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外部割り込みを禁止します。
IEビットの詳細については"3.3.8 マスク可能な外部割り込み"を参照してください。

DS(bit 6): 被除数符号フラグ

ステップ除算の実行時には被除数の符号ビットがDSフラグにセットされ、除算の実行に影響を与えます。

MO(bit 7): 積和演算オーバーフローフラグ

積和演算によるオーバーフローを示します。積和(mac)演算実行中に途中結果が符号付き64ビットの有効範囲を越えると1にセットされます。演算はオーバーフローの発生にかかわらず最後まで実行されますので、演算終了後にMOフラグを読み出して結果が有効かどうかを判断します。MOフラグは一旦セットされると、イニシャルリセットかプログラムによって明示的にリセットするまで1を保持します。

IL(bit 8 ~ bit 11): 割り込みレベル

CPUの割り込みレベルを示します。マスク可能な外部割り込み要求は、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のレベルに設定され、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。

2.1.4 スタックポインタ

図2.1.4.1 SP

スタックポインタ(以降SPと記述)はスタックの先頭アドレスを保持する32ビットレジスタです。
スタックはRAM上に任意に配置可能な領域で、SPで初期設定したアドレスからセーブ(プッシュ)したデータの数に従って低いアドレスの方へ拡大していきます。データをスタックにプッシュする際、SPは書き込みの前に減算(ワード単位、-4)され、データを書き込む領域を確保します。逆にデータをスタックから取り出す(ポップ)場合は、SPの示すアドレスからワード単位にデータが取り出されます。その後、SPには4が加算され、取り出したデータの領域を解放します。

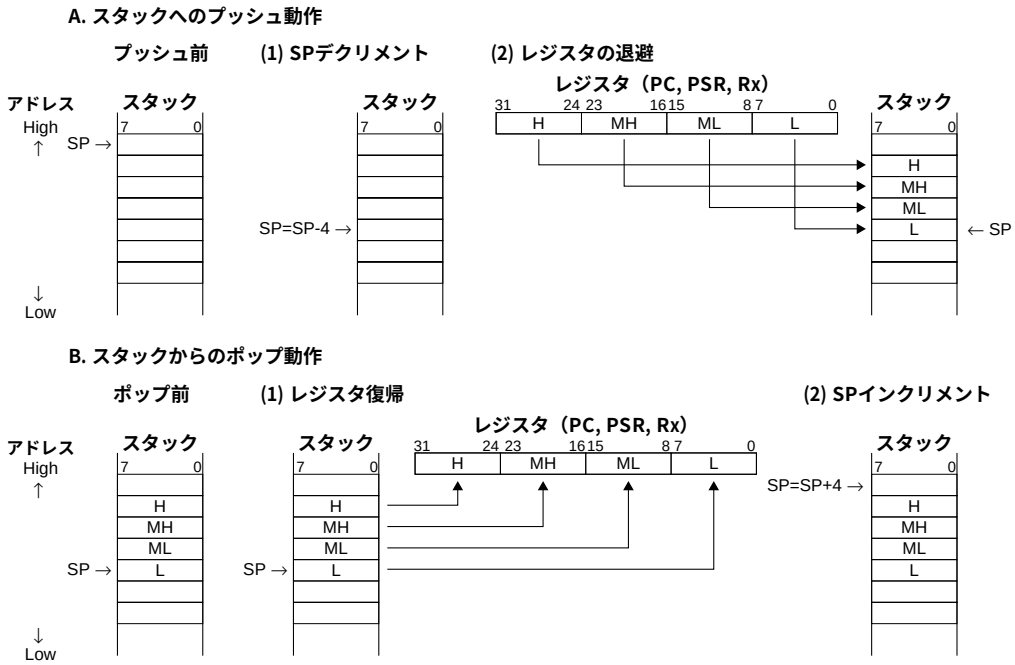


図2.1.4.2 SPとスタック

スタックにプッシュされるのはすべて32ビットの内部レジスタデータのため、SPの下位2ビットはワード境界を示す0に固定されます。また、アドレスバスが28ビットのため、上位4ビットは無効となります。なお、実際の有効アドレスは各機種のメモリ構成によって異なります。
スタックデータのプッシュとポップは以下の場合に行われます。

(1)call命令実行時

callはサブルーチンコール命令で、スタックを1ワード使用します。call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、callの次の命令アドレス)をスタックにプッシュします。プッシュされたアドレスはサブルーチン内のRET(リターン)命令によってPCにポップされ、プログラムの実行は呼び出し元のルーチンに戻ります。

(2)割り込みまたは例外の発生時

割り込み、int命令によるソフトウェア割り込みあるいは例外等、トラップが発生すると、CPUはそれぞれの処理ルーチンに分岐する前にPCとPSRの内容をスタックにプッシュします。これはトラップによって、この2つのレジスタの内容が変更されるためです。PCとPSRのデータは図2.1.4.3のようにスタックにプッシュされます。

処理ルーチンからのリターンにはPCとPSRの内容をポップするreti命令を使用します。

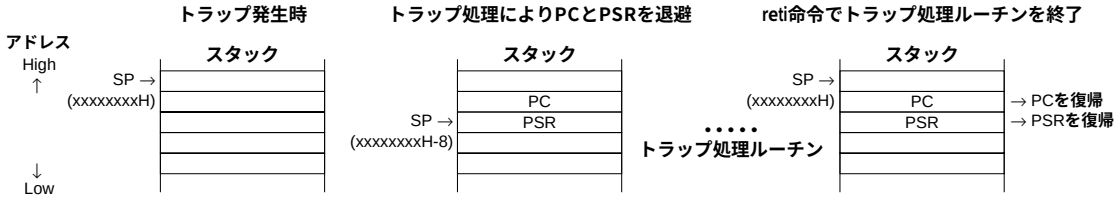


図2.1.4.3　トラップ発生時のスタック操作

(3)pushn命令、popn命令実行時

pushn命令によってr0から指定の汎用レジスタまでをスタックに退避させることができます。プッシュしたデータはpopn命令によって各レジスタに戻ります。

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不定となりますので、初期化ルーチン内の先頭部分でアドレス(スタックの最終アドレス+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や例外が発生すると、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。

2.1.5 算術演算レジスタ(ALR, AHR)

特殊レジスタには、乗除算、積和演算に使用する算術演算ローレジスタ(以降ALRと記述)と算術演算ハイレジスタ(以降AHRと記述)があります。

それぞれ、32ビットのデータ用レジスタで、ロード命令によって汎用レジスタとのデータ転送が行えます。

乗算命令および積和演算命令はALRに演算結果の下位32ビットを、AHRに演算結果の上位32ビットを置きます。

除算命令はALRに商を、AHRに余りを置きます。

イニシャルリセット時、ALRおよびAHRは不定となります。

なお、ALRとAHRはオプションの乗除算回路を内蔵した機種でのみ使用可能です。

2.1.6 レジスタの表記とレジスタ番号

以下にS1C33000命令セットのレジスタ表記とレジスタ番号を示します。命令コード中ではレジスタの指定に4ビットのフィールドを使用しており、そこにレジスタ番号が入ります。なお、ニーモニック中では、レジスタ名の前に"%"を付けて指定します。

(1)汎用レジスタ

%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～%r15と記述します。

%rd　rdはディスティネーションとなる(演算が行われる、あるいはデータがロードされる)汎用レジスタを示すメタシンボルです。実際には%r0～%r15と記述します。

%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]～[%r15]のように記述します。レジスタ間接アドレッシングでは、連続したメモリアクセスのためのポストインクリメント機能を使用することができます。その場合は[%r0]+～[%r15]+のように"+"を後置きします。ポストインクリメントを指定すると、メモリアクセスの後にベースアドレスがアクセスしたデータサイズに従ってインクリメントされます。
rbはcall命令やjp命令の分岐先アドレスを保持しているレジスタを示すシンボルとしても使用しています。この場合は[ ]が不要で、%r0～%r15と記述します。

汎用レジスタのレジスタ番号は、実際のレジスタ名の番号と同じです。コード中のレジスタビットフィールドには指定したレジスタの0～15(0b0000～0b1111)が入ります。

(2)特殊レジスタ

%ss　ssは汎用レジスタに転送するソースデータを保持している特殊レジスタを示すメタシンボルです。特殊レジスタをソースとする命令は"ld.w %rd, %ss"のみです。

%sd　sdは汎用レジスタからデータをロードする特殊レジスタを示すメタシンボルです。特殊レジスタをディスティネーションとする命令は"ld.w %sd, %rs"のみです。

特殊レジスタ番号と、実際の記述方法は表2.1.6.1のとおりです。

表2.1.6.1　特殊レジスタ番号と記述方法

特殊レジスタ名	レジスタ番号	記述方法
プロセッサステータスレジスタ	0	%psr
スタックポインタ	1	%sp
算術演算ローレジスタ	2	%alr
算術演算ハイレジスタ	3	%ahr

レジスタビットフィールドの上位2ビットに0b00、下位2ビットにレジスタ番号0～3(0b00～0b11)が入ります。

2.2　データの形式

S1C33000は8ビット、16ビット、32ビットのデータを扱うことができます。
本書ではそれぞれのデータサイズを次のように記述します。

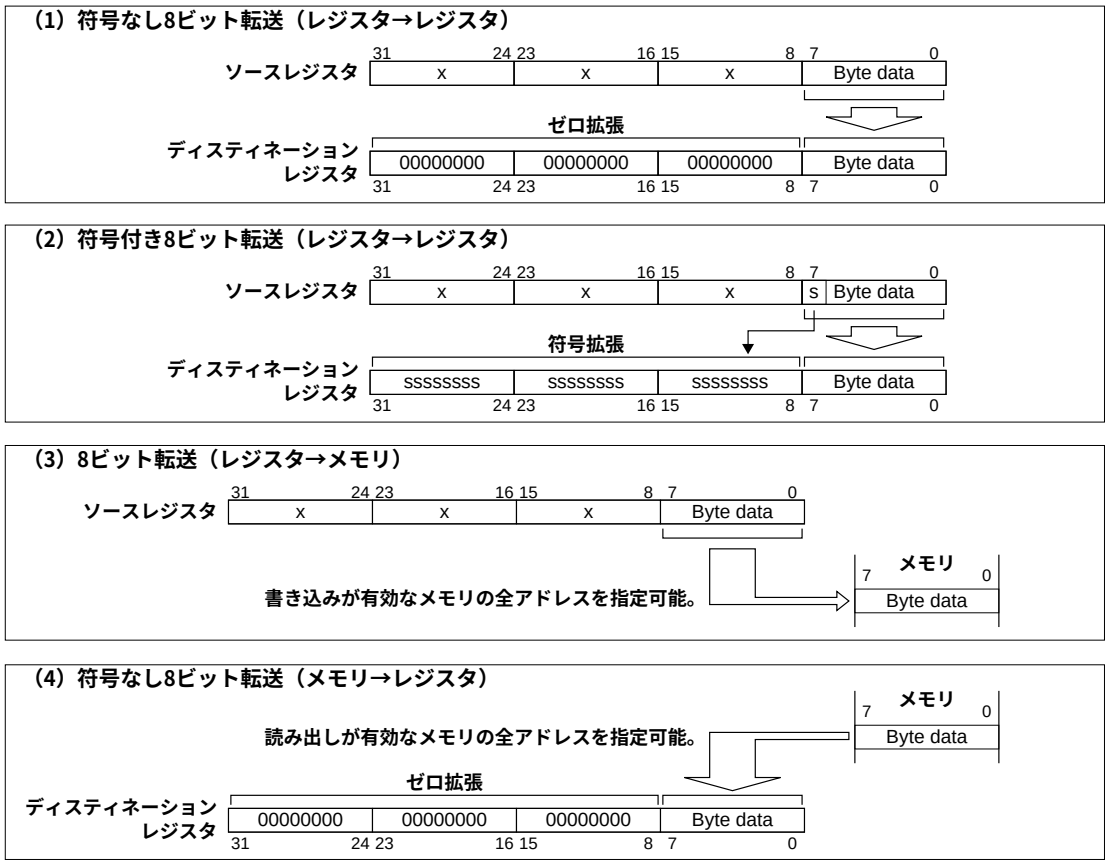
8ビット:　バイト(Byte, B)
16ビット:　ハーフワード(Half word, H)
32ビット:　ワード(Word, W)

16ビットをワード(Word)、32ビットをロングワード(Long word)等と記述している一般の書籍もありますので注意してください。

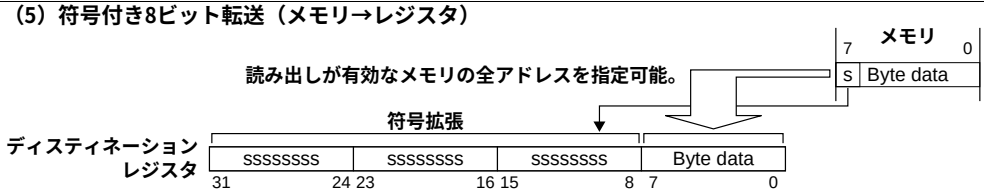
データサイズは、メモリと汎用レジスタ間、および汎用レジスタ間のデータ転送(ロード命令)においてのみ選択可能です。
CPUの内部処理はすべて32ビットで行われますので、汎用レジスタをディスティネーションとする16ビットデータ転送および8ビットデータ転送では、レジスタにロードする際に32ビットに符号拡張またはゼロ拡張されます。符号拡張とゼロ拡張は使用するロード命令によって決まります。
汎用レジスタをソースとする16ビットデータ転送または8ビットデータ転送では、ソースレジスタの下位ハーフワードまたは下位1バイトが転送データとなります。

メモリはバイト、ハーフワード、ワード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、ハーフワード単位およびワード単位のアクセスは、指定するベースアドレスがそれぞれハーフワード境界(アドレスの最下位ビットが0)、ワード境界(アドレスの下位2ビットが0)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例外が発行されます。

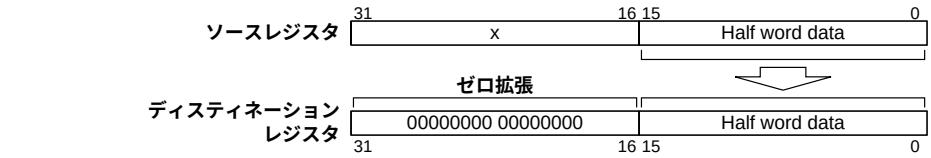
図2.2.1にデータ転送の種類を示します。



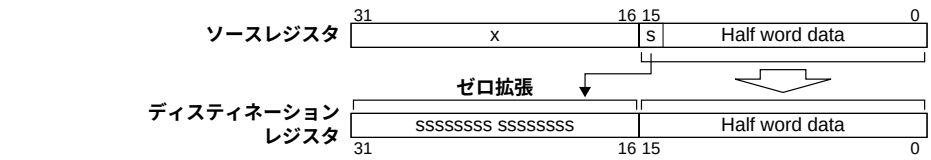
（5）符号付き8ビット転送（メモリ→レジスタ）



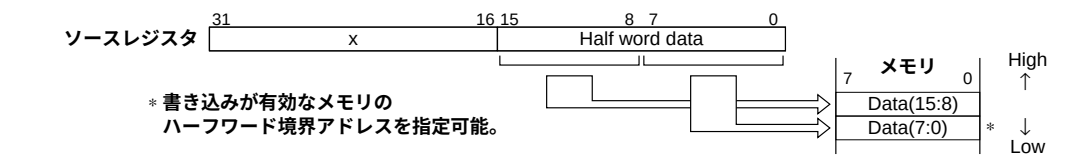
（6）符号なし16ビット転送（レジスタ→レジスタ）



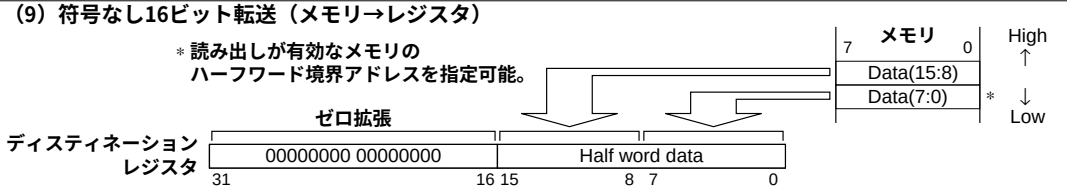
（7）符号付き16ビット転送（レジスタ→レジスタ）



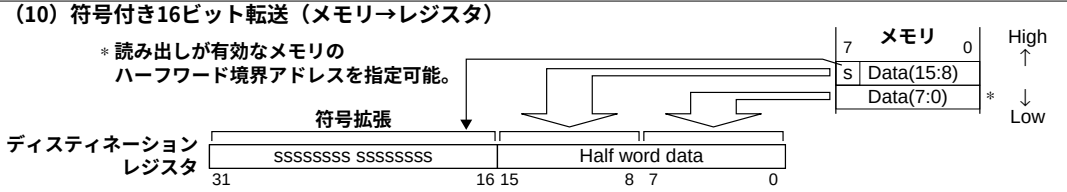
（8）16ビット転送（レジスタ→メモリ）



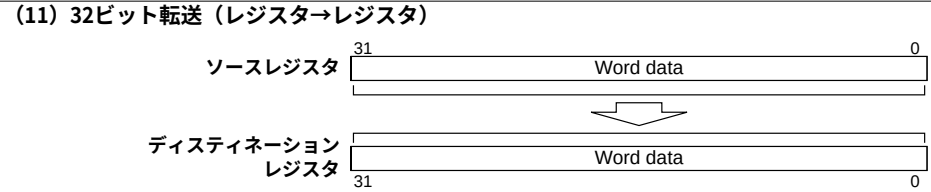
（9）符号なし16ビット転送（メモリ→レジスタ）



（10）符号付き16ビット転送（メモリ→レジスタ）



（11）32ビット転送（レジスタ→レジスタ）



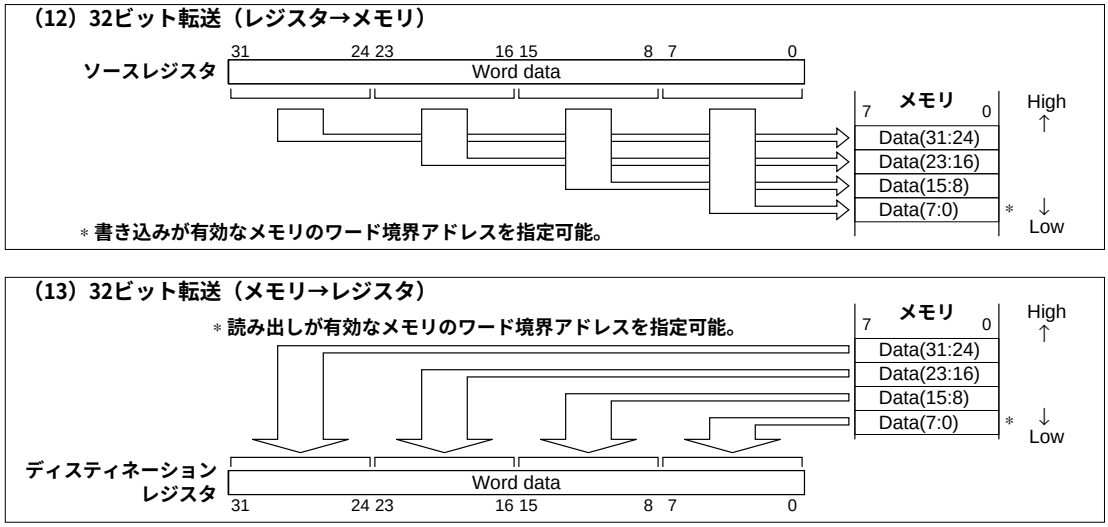


図2.2.1　データ転送の種類

2.3 アドレス空間

S1C33000は28ビット(256Mバイト)のアドレス空間を持ちます。
すべてのメモリはこの空間内に配置されます。また、メモリマップドI/O方式により、I/Oモジュールの制御レジスタ等もこの空間に配置され、通常のメモリと同様にアクセスできます。
図2.3.1に基本的なメモリマップを示します。

エリア番号	アドレス		エリアサイズ
エリア18	0xFFFFFFF 0xC000000	外部メモリ	64Mバイト
エリア17	0xBFFFFFF 0x8000000	外部メモリ	64Mバイト
エリア16	0x7FFFFFF 0x6000000	外部メモリ	32Mバイト
エリア15	0x5FFFFFF 0x4000000	外部メモリ	32Mバイト
エリア14	0x3FFFFFF 0x3000000	外部メモリ	16Mバイト
エリア13	0x2FFFFFF 0x2000000	外部メモリ	16Mバイト
エリア12	0x1FFFFFF 0x1800000	外部メモリ	8Mバイト
エリア11	0x17FFFFF 0x1000000	外部メモリ	8Mバイト
エリア10	0x0FFFFFF 0x0C00000	外部メモリ	4Mバイト
エリア9	0x0BFFFFF 0x0800000	外部メモリ	4Mバイト
エリア8	0x07FFFFF 0x0600000	外部メモリ	2Mバイト
エリア7	0x05FFFFF 0x0400000	外部メモリ	2Mバイト
エリア6	0x03FFFFF 0x0300000	外部I/O	1Mバイト
エリア5	0x02FFFFF 0x0200000	外部メモリ	1Mバイト
エリア4	0x01FFFFF 0x0100000	外部メモリ	1Mバイト
エリア3	0x00FFFFF 0x0080000	内蔵ROM	512Kバイト
エリア2	0x007FFFF 0x0060000	ICE用予約エリア	128Kバイト
エリア1	0x005FFFF 0x0040000	内蔵周辺回路	128Kバイト
エリア0	0x003FFFF 0x0000000	内蔵RAM	256Kバイト

図2.3.1 メモリマップ

図のようにS1C33000はアドレス空間全体を19個のエリアに分割して管理します。それぞれのエリアには接続するモジュールの種類が割り当てられています。特にエリア0はS1C33 Familyの内蔵RAM、エリア1は内蔵周辺回路、エリア3は内蔵ROMに割り当てられています。また、エリア10に外部ROMを接続し、そのROMからブートすることも可能です。
エリア2は内蔵用エリアですが、ICE用に予約されていますので使用しないでください。("3.6 デバッグモード"参照)。

外部モジュール用のエリアは、決められたエリア範囲ごとに使用するデバイスの種類や、ポートのデータ幅、ウェイトサイクルなどを指定することができます。その内容はS1C33 Family各機種の構成によって異なります。

S1C33000はアドレスデコーダを内蔵しており、この19エリアに対応する19本のセレクト信号を出力可能です。このため、メモリマップの構成に沿ったシステムは外部にグルーロジックを設ける必要がなく、直接デバイスを接続することができます。

内蔵メモリ容量やI/Oメモリ、アドレスバスのサイズはS1C33 Familyの機種により異なります。したがって、図2.3.1に示したメモリマップがすべての機種に適用される訳ではありません。実際のメモリマップについてはS1C33 Familyの各機種に用意されたテクニカルマニュアルを参照してください。

2.4　ブートアドレス

S1C33000はBTA3端子の入力レベルによって、トラップテーブルを置くエリアをエリア3(内蔵ROM)またはエリア10(外部ROM)の2種類から選択できるようになっています。トラップテーブルはエリアの先頭から始まり、ブート用のリセットベクタがテーブルの先頭に置かれます。したがって、ブートアドレスは選択されたエリアの先頭アドレスに置かれます。

表2.4.1　ブートアドレスの設定

端子入力	選択されるエリア	ブートアドレスのロケーション
BTA3=1(High)	エリア3(内蔵ROM)	0x0080000
BTA3=0(Low)	エリア10(外部ROM)	0x0C00000

S1C33 Familyの一般的な機種はROMを内蔵しているため、どちらからでもブートできます。
内蔵ROMを持たない機種では、外部ROMからのブートのみとなります。
実際の機種の対応についてはS1C33 Familyの各機種に用意されたテクニカルマニュアルを参照してください。

2.5 命令セット

S1C33000命令セットは61の基本命令(全105命令)を持ちます。命令コードはすべて16ビットの固定長で、CPUはパイプライン処理とロード・ストア型のアーキテクチャにより、主要な命令を1サイクルで実行します。また、C言語による開発でも非常にコンパクトなモジュールが生成可能なコード体系になっています。
ここではS1C33000命令セットと機能の概要を説明します。
命令個別の詳細については"4 命令の詳細説明"を参照してください。

2.5.1 命令の種類

以下に機能別の命令一覧表を示します。

表2.5.1.1 命令一覧表

分類	ニーモニック		機　　能
論理演算	and	%rd, %rs	汎用レジスタ間の論理積
		%rd, sign6	汎用レジスタと即値の論理積(即値は符号拡張)
	or	%rd, %rs	汎用レジスタ間の論理和
		%rd, sign6	汎用レジスタと即値の論理和(即値は符号拡張)
	xor	%rd, %rs	汎用レジスタ間の排他的論理和
		%rd, sign6	汎用レジスタと即値の排他的論理和(即値は符号拡張)
	not	%rd, %rs	汎用レジスタの論理反転
		%rd, sign6	即値の論理反転(即値は符号拡張)
算術演算	add	%rd, %rs	汎用レジスタ間の加算
		%rd, imm6	汎用レジスタと即値の加算(即値はゼロ拡張)
		%sp, imm10	SPと即値の加算(即値はゼロ拡張)
	adc	%rd, %rs	汎用レジスタ間のキャリー付き加算
	sub	%rd, %rs	汎用レジスタ間の減算
		%rd, imm6	汎用レジスタから即値を減算(即値はゼロ拡張)
		%sp, imm10	SPから即値を減算(即値はゼロ拡張)
	sbc	%rd, %rs	汎用レジスタ間のボロー付き減算
	cmp	%rd, %rs	汎用レジスタ間の比較
		%rd, sign6	汎用レジスタと即値の比較(即値は符号拡張)
	mlt.h	%rd, %rs	符号付き整数乗算(16ビット×16ビット=32ビット)　〈オプション〉
	mltu.h	%rd, %rs	符号なし整数乗算(16ビット×16ビット=32ビット)　〈オプション〉
	mlt.w	%rd, %rs	符号付き整数乗算(32ビット×32ビット=64ビット)　〈オプション〉
	mltu.w	%rd, %rs	符号なし整数乗算(32ビット×32ビット=64ビット)　〈オプション〉
	div0s	%rs	符号付き整数除算の第1ステップ　〈オプション〉
	div0u	%rs	符号なし整数除算の第1ステップ　〈オプション〉
	div1	%rs	ステップ除算実行　〈オプション〉
	div2s	%rs	符号付き整数除算結果のデータ補正1　〈オプション〉
	div3s		符号付き整数除算結果のデータ補正2　〈オプション〉
シフト ローテート	srl	%rd, %rs	右方向論理シフト(レジスタによるシフト量指定)
		%rd, imm4	右方向論理シフト(即値によるシフト量指定)
	sll	%rd, %rs	左方向論理シフト(レジスタによるシフト量指定)
		%rd, imm4	左方向論理シフト(即値によるシフト量指定)
	sra	%rd, %rs	右方向算術シフト(レジスタによるシフト量指定)
		%rd, imm4	右方向算術シフト(即値によるシフト量指定)
	sla	%rd, %rs	左方向算術シフト(レジスタによるシフト量指定)
		%rd, imm4	左方向算術シフト(即値によるシフト量指定)
	rr	%rd, %rs	右方向ローテート(レジスタによるシフト量指定)
		%rd, imm4	右方向ローテート(即値によるシフト量指定)
	rl	%rd, %rs	左方向ローテート(レジスタによるシフト量指定)
		%rd, imm4	左方向ローテート(即値によるシフト量指定)
分岐	jrgt jrgt.d	sign8	PC相対条件ジャンプ　分岐条件: !Z & !(N ^ V) (".d"によりディレイド分岐指定可)
	jrge jrge.d	sign8	PC相対条件ジャンプ　分岐条件: !(N ^ V) (".d"によりディレイド分岐指定可)
	jrlt jrlt.d	sign8	PC相対条件ジャンプ　分岐条件: N ^ V (".d"によりディレイド分岐指定可)
	jrle jrle.d	sign8	PC相対条件ジャンプ　分岐条件: Z | N ^ V (".d"によりディレイド分岐指定可)
	jrugt jrugt.d	sign8	PC相対条件ジャンプ　分岐条件: !Z & !C (".d"によりディレイド分岐指定可)
	jruge jruge.d	sign8	PC相対条件ジャンプ　分岐条件: !C (".d"によりディレイド分岐指定可)

分類	ニーモニック		機　　能
分岐	jrult jrult.d	sign8	PC相対条件ジャンプ　分岐条件: C (".d"によりディレイド分岐指定可)
	jrule jrule.d	sign8	PC相対条件ジャンプ　分岐条件: Z | C (".d"によりディレイド分岐指定可)
	jreq jreq.d	sign8	PC相対条件ジャンプ　分岐条件: Z (".d"によりディレイド分岐指定可)
	jrne jrne.d	sign8	PC相対条件ジャンプ　分岐条件: !Z (".d"によりディレイド分岐指定可)
	jp	sign8	PC相対ジャンプ(".d"によりディレイド分岐指定可)
	jp.d	%rb	絶対ジャンプ(".d"によりディレイド分岐指定可)
	call	sign8	PC相対コール(".d"によりディレイド分岐指定可)
	call.d	%rb	絶対コール(".d"によりディレイド分岐指定可)
	ret ret.d		サブルーチンからのリターン (".d"によりディレイド分岐指定可)
	reti		割り込み処理ルーチン、例外処理ルーチンからのリターン
	retd		デバッグ処理ルーチンからのリターン
	int	imm2	ソフトウェア例外
	brk		デバッグ例外
データ転送	ld.b	%rd, %rs	汎用レジスタ(バイト)→汎用レジスタ(データは符号拡張)
		%rd, [%rb]	メモリ(バイト)→汎用レジスタ(データは符号拡張)
		%rd, [%rb]+	"+"はアドレスのポストインクリメント指定
		%rd, [%sp+imm6]	スタック(バイト)→汎用レジスタ(データは符号拡張)
		[%rb], %rs	汎用レジスタ(バイト)→メモリ
		[%rb]+, %rs	"+"はアドレスのポストインクリメント指定
		[%sp+imm6], %rs	汎用レジスタ(バイト)→スタック
	ld.ub	%rd, %rs	汎用レジスタ(バイト)→汎用レジスタ(データはゼロ拡張)
		%rd, [%rb]	メモリ(バイト)→汎用レジスタ(データはゼロ拡張)
		%rd, [%rb]+	"+"はアドレスのポストインクリメント指定
		%rd, [%sp+imm6]	スタック(バイト)→汎用レジスタ(データはゼロ拡張)
	ld.h	%rd, %rs	汎用レジスタ(ハーフワード)→汎用レジスタ(データは符号拡張)
		%rd, [%rb]	メモリ(ハーフワード)→汎用レジスタ(データは符号拡張)
		%rd, [%rb]+	"+"はアドレスのポストインクリメント指定
		%rd, [%sp+imm6]	スタック(ハーフワード)→汎用レジスタ(データは符号拡張)
		[%rb], %rs	汎用レジスタ(ハーフワード)→メモリ
		[%rb]+, %rs	"+"はアドレスのポストインクリメント指定
		[%sp+imm6], %rs	汎用レジスタ(ハーフワード)→スタック
	ld.uh	%rd, %rs	汎用レジスタ(ハーフワード)→汎用レジスタ(データはゼロ拡張)
		%rd, [%rb]	メモリ(ハーフワード)→汎用レジスタ(データはゼロ拡張)
		%rd, [%rb]+	"+"はアドレスのポストインクリメント指定
		%rd, [%sp+imm6]	スタック(ハーフワード)→汎用レジスタ(データはゼロ拡張)
	ld.w	%rd, %rs	汎用レジスタ(ワード)→汎用レジスタ
		%rd, %ss	特殊レジスタ(ワード)→汎用レジスタ
		%sd, %rs	汎用レジスタ(ワード)→特殊レジスタ
		%rd, sign6	即値→汎用レジスタ(即値は符号拡張)
		%rd, [%rb]	メモリ(ワード)→汎用レジスタ
		%rd, [%rb]+	"+"はアドレスのポストインクリメント指定
		%rd, [%sp+imm6]	スタック(ワード)→汎用レジスタ
		[%rb], %rs	汎用レジスタ(ワード)→メモリ
		[%rb]+, %rs	"+"はアドレスのポストインクリメント指定
		[%sp+imm6], %rs	汎用レジスタ(ワード)→スタック
システム制御	nop		ノーオペレーション
	halt		HALTモード設定
	slp		SLEEPモード設定
即値拡張	ext	imm13	直後の命令のオペランド(即値)を拡張
ビット処理	btst	[%rb], imm3	メモリデータ(バイト)の指定ビットをテスト
	bclr	[%rb], imm3	メモリデータ(バイト)の指定ビットをクリア
	bset	[%rb], imm3	メモリデータ(バイト)の指定ビットをセット
	bnot	[%rb], imm3	メモリデータ(バイト)の指定ビットを反転
その他	scan0	%rd, %rs	"0"ビットのサーチ
	scan1	%rd, %rs	"1"ビットのサーチ
	swap	%rd, %rs	ワード中のバイト境界でバイト単位のスワップ(上位↔下位)
	mirror	%rd, %rs	ワード中のバイト毎にビット単位でスワップ(上位↔下位)
	mac	%rs	積和演算　16ビット×16ビット+64ビット→64ビット　〈オプション〉
	pushn	%rs	%rsから%r0までの汎用レジスタをスタックにプッシュ
	popn	%rd	%r0から%rdまでの汎用レジスタのデータをスタックからポップ

2.5.2 アドレッシングモード

S1C33000命令セットは以下に示す6種類のアドレッシングモードを持ちます。CPU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

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。論理演算(and, or, xor, not)、算術演算(add, sub, cmp)、即値ロード("ld.w %rd, sign6")、シフト&ローテート(srl, sll, sra, sla, rr, rl)、ビット操作(btst, bclr, bset, bnot)、即値拡張(ext)の各命令で、このアドレッシングモードを使用することができます。
各命令で指定可能な即値サイズは、シンボルの数字(例: imm4=符号なし4ビット、sign6=符号付き6ビット)で示されます。
シフト・ローテート命令を除き、ext命令による即値拡張が行えます(次節参照)。

(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ィ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～%r15と記述します。

%rd　rdはディ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～%r15と記述します。命令によって、ソースデータにもなります。

%ss　ssは汎用レジスタに転送するソースデータを保持している特殊レジスタを示すメタシンボルです。

%sd　sdは汎用レジスタからデータをロードする特殊レジスタを示すメタシンボルです。
実際の特殊レジスタ名は次のように記述します。

プロセッサステータスレジスタ	%psr
スタックポインタ	%sp
算術演算ローレジスタ	%alr
算術演算ハイレジスタ	%ahr

レジスタ名はシンボル等と区別するため、必ず"%"を前置きします。

(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]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。実際は汎用レジスタ名を[ ]で囲み、[%r0]～[%r15]のように記述します。
CPU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。
ハーフワード転送、ワード転送の場合、レジスタに設定するベースアドレスはそれぞれハーフワード境界(最下位ビット=0)、ワード境界(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例外が発生します。

(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

レジスタ間接アドレッシングと同様に、汎用レジスタによってアクセスするメモリを間接的に指定します。データ転送が終了すると、指定したレジスタが保持しているベースアドレスを、転送したデータサイズ分インクリメント*します。これにより、メモリ上の連続したデータの読み出し/書き込みが、最初に1回先頭アドレスを設定するだけで行えます。

*** インクリメントサイズ**

バイト転送(ld.b, ld.ub):	rb←rb+1
ハーフワード転送(ld.h, ld.uh):	rb←rb+2
ワード転送(ld.w):	rb←rb+4

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]+～[%r15]+のように記述します。

(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+imm6]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。バイトデータ転送の場合、6ビットの即値がそのままディスプレースメントとして加算されます。
ld.h  %r0,[%sp+0x10]   ; 現在のSPの内容に0x20を加算したアドレスのハーフワードデータをR0レジスタにロードします。ハーフワードデータ転送の場合、ハーフワード境界をアクセスするため、指定した6ビット即値を2倍(最下位ビットは常に0)にした値がディスプレースメントとなります。
ld.w  %r0,[%sp+0x10]   ; 現在のSPの内容に0x40を加算したアドレスのワードデータをR0レジスタにロードします。ワードデータ転送の場合、ワード境界をアクセスするため、指定した6ビット即値を4倍(下位2ビットは常に0)にした値がディスプレースメントとなります。
```

次節で説明する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本アドレッシングモードに変わります。

例:
```
ext   imm13
ld.b  %rd,[%rb]        ; "ld.b  %rd, [%rb+imm13]"として機能します。
```

(6)符号付きPC相対アドレッシング

符号付き8ビット即値(sign8)をオペランドに持つ分岐命令(jr*, jp, call)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sign8の2倍の値(ハーフワード境界)を加算したアドレスに分岐します。

ext命令により即値のサイズを拡張することができます(次節参照)。

2.5.3 即値拡張(EXT)命令

16ビット固定長の命令コードでは、その中で指定可能な即値のサイズは限られてしまいます。ext命令は主にこの即値のサイズを拡張するために使用します。
ある命令の即値を拡張したい場合、その命令の直前にext命令を記述します。また、ext命令で指定可能な即値サイズは13ビットのため、さらに即値拡張するために2個までext命令を続けることができます。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。また、3個以上のext命令が連続的に記述された場合は、最初と最後(即値拡張の対象となる命令の直前のext命令)の2個のみが有効で、その間のext命令は無効となります。
以下にext命令の機能を示します。

注: ext命令の拡張例では、1個目のext命令の即値をimm13、2個目のext命令の即値をimm13'と区別して表記しています。

(1)即値データアドレッシング命令の即値サイズの拡張

- **imm6の拡張**
  拡張対象の命令: "add %rd, imm6", "sub %rd, imm6"
  上記命令単独で指定可能な即値サイズは6ビットです。
  これらの命令の直前にext命令を記述することで、即値サイズを19ビット、あるいは32ビットに拡張することができます。

  ext命令を1個使用

```
ext     imm13
add     %rd,imm6        ; "add %rd, imm19"として機能します。
```

  imm6(6ビット)をimm19(19ビット)に拡張します。ext命令のimm13はimm19の上位13ビットとなります。rdレジスタに対する演算は32ビットとして行われますので、imm19は32ビットにゼロ拡張されます。

  ext命令を2個使用

```
ext     imm13
ext     imm13'
sub     %rd,imm6        ; "sub %rd, imm32"として機能します。
```

  imm6(6ビット)をimm32(32ビット)に拡張します。各即値は上位側からimm13、imm13'、imm6の順で32ビットデータを構成します。

- **sign6の拡張**
  拡張対象の命令: "and %rd, sign6", "or %rd, sign6", "xor %rd, sign6", "not %rd, sign6", "cmp %rd, sign6", "ld.w %rd, sign6"
  上記命令単独で指定可能な即値サイズは符号付き6ビットです。
  これらの命令の直前にext命令を記述することで、即値サイズを符号付き19ビット、あるいは符号付き32ビットに拡張することができます。

  ext命令を1個使用

```
ext     imm13
and     %rd,sign6       ; "and %rd, sign19"として機能します。
```

  sign6(符号付き6ビット)をsign19(符号付き19ビット)に拡張します。ext命令のimm13はsign19の上位13ビットとなります。rdレジスタに対する演算/転送は32ビットとして行われますので、sign19はimm13のMSBを符号ビット(0=+、1=-)として32ビットに符号拡張されます。

  ext命令を2個使用

```
ext     imm13
ext     imm13'
cmp     %rd,sign6       ; "cmp %rd, sign32"として機能します。
```

  sign6(符号付き6ビット)をsign32(符号付き32ビット)に拡張します。各即値は上位側からimm13、imm13'、sign6の順で32ビットデータを構成します。最初のimm13のMSBが符号ビットとなります。

(2) レジスタ間接アドレッシングのディスプレースメントを拡張

- **[%rb]にディスプレースメントを付加**

  拡張対象の命令:　"ld. %rd, [%rb]"(ld.*: ld.b, ld.ub, ld.h, ld.uh, ld.w), "ld.* [%rb], %rs"(ld.*: ld.b, ld.h, ld.w),*
  "btst [%rb], imm3", "bclr [%rb], imm3", "bset [%rb], imm3", "bnot [%rb], imm3"

  上記命令単独ではrbレジスタの内容をベースアドレスとしたレジスタ間接アドレッシングが行われます。これらの命令の直前にext命令を記述することで、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更することができます。

  ext命令を1個使用

  ```
  ext     imm13
  ld.b    %rd,[%rb]          ; "ld.b  %rd, [%rb+imm13]"として機能します。
  ```

  rbレジスタで指定されるベースアドレスにimm13で指定される13ビットのディスプレースメント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32ビットにゼロ拡張されます。

  ext命令を2個使用

  ```
  ext     imm13
  ext     imm13'
  btst    [%rb],imm3         ; "btst  [%rb+imm26], imm3"として機能します。
  ```

  rbレジスタで指定されるベースアドレスにimm26で指定される26ビットのディスプレースメントを加えたアドレスをアクセスします。各即値は上位側からimm13、imm13'の順で26ビットディスプレースメントを構成します。アドレス演算の際、imm26は32ビットにゼロ拡張されます。

  この拡張は、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([%rb]+)の命令には適用されません。

- **[%sp+imm6]のディスプレースメントを拡張**

  拡張対象の命令:　"ld. %rd, [%sp+imm6]"(ld.*: ld.b, ld.ub, ld.h, ld.uh, ld.w)*
  "ld. [%sp+imm6], %rs"(ld.*: ld.b, ld.h, ld.w)*

  上記命令単独ではSPの内容をベースアドレス、コード中の即値imm6を6ビット～8ビットのディスプレースメントとして使用する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が行われます。

バイト転送(ld.b, ld.ub):	6ビットディスプレースメント = imm6 = {imm6}
ハーフワード転送(ld.h, ld.uh):	7ビットディスプレースメント = imm6×2 = {imm6, 0}
ワード転送(ld.w):	8ビットディスプレースメント = imm6×4 = {imm6, 00}

  これらの命令の直前にext命令を記述することで、ディスプレースメントのサイズを19ビットあるいは32ビットに拡張することができます。

  ext命令を1個使用

  ```
  ext     imm13
  ld.b    %rd,[%sp+imm6]     ; "ld.b  %rd, [%sp+imm19]"として機能します。
  ```

  SPが示すスタック先頭アドレスにimm19で指定される19ビットのディスプレースメントを加えたアドレスをアクセスします。ext命令で指定したimm13がimm19の上位13ビットとなり、ロード命令のimm6はimm19の下位6ビットとして使用されます。ただし、ハーフワード転送またはワードデータ転送では、アドレス不整例外が起こらないように、imm6が次のように用いられます。

バイト転送(ld.b, ld.ub):	imm19 = {imm13, imm6}
ハーフワード転送(ld.h, ld.uh):	imm19 = {imm13, imm6(5:1), 0}　(imm6の最下位ビットは無効)
ワード転送(ld.w):	imm19 = {imm13, imm6(5:2), 00}（imm6の下位2ビットは無効）

  アドレス演算の際、imm19は32ビットにゼロ拡張されます。

ext命令を2個使用

```
ext     imm13
ext     imm13'
ld.w    [%sp+imm6],%rs  ; "ld.w [%sp+imm32], %rs"として機能します。
```

SPが示すスタック先頭アドレスにimm32で指定される32ビットのディスプレースメントを加えたアドレスをアクセスします。各即値は上位側からimm13、imm13'、imm6の順で32ビットディスプレースメントを構成します。ただし、ハーフワード転送またはワードデータ転送では、アドレス不整例外が起こらないように、imm6が次のように用いられます。

バイト転送(ld.b, ld.ub): imm32 = {imm13, imm13', imm6}
ハーフワード転送(ld.h, ld.uh): imm32 = {imm13, imm13', imm6(5:1), 0}（imm6の最下位ビットは無効）
ワード転送(ld.w): imm32 = {imm13, imm13', imm6(5:2), 00}(imm6の下位2ビットは無効）

アドレス演算の際、imm32は符号なし32ビットデータとして扱われます。ディスプレースメントを加算後の値が有効アドレス範囲(最大28ビット)を越えた場合は、越えた部分が無効となります。

(3)レジスタ間演算命令を3オペランド命令に拡張

拡張対象の命令: "add %rd, %rs", "sub %rd, %rs", "cmp %rd, %rs", "and %rd, %rs", "or %rd, %rs", "xor %rd, %rs"

上記命令単独ではrdレジスタの内容とrsレジスタの内容を演算し、結果をrdレジスタにロードします。これらの命令の直前にext命令を記述すると、rsレジスタとext命令で指定された即値との演算を行い、結果をrdにロードします。rdレジスタの内容は演算に影響を与えません。

ext命令を1個使用

```
ext     imm13
add     %rd,%rs     ; "rd ← rs + imm13"として機能します。
```

演算は32ビットとして行われますので、imm13は32ビットにゼロ拡張されます。

ext命令を2個使用

```
ext     imm13
ext     imm13'
sub     %rd,%rs     ; "rd ← rs - imm26"として機能します。
```

各即値は上位側からimm13、imm13'の順で26ビットデータを構成します。
演算は32ビットとして行われますので、imm26は32ビットにゼロ拡張されます。

(4)PC相対分岐命令のディスプレースメントの拡張

オペランドにsign8(符号付き8ビット即値)を持つPC相対分岐命令は、現在のPCが示すアドレスにsign8を2倍にした9ビットディスプレースメントを加え、そのアドレスに分岐します。ext命令を使用することで、このディスプレースメントを22ビット(ext命令1個)、32ビット(ext命令2個)に拡張することができます。この詳細については"2.5.12 分岐命令・ディレイド命令"を参照してください。

2.5.4 データ転送命令

S1C33000命令セットは、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.h	**符号付きハーフワードデータ転送**
ld.uh	**符号なしハーフワードデータ転送**
ld.w	**ワードデータ転送**

レジスタへの符号付きバイト/ハーフワード転送では、ソースデータが32ビットに符号拡張されます。符号なしバイト/ハーフワード転送では、ソースデータが32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。

2.5.5 論理演算命令

S1C33000命令セットでは、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～R15)に対して行われます。ソースは指定の汎用レジスタの32ビットデータか、符号付き即値データ(6、19、32ビット)の2種類です。

2.5.6 算術演算命令

S1C33000命令セットでは算術演算用に、加減算、比較、乗除算命令をサポートしています(乗除算命令は次節で説明します)。

add	**加算命令**
adc	**キャリー付き加算命令**
sub	**減算命令**
sbc	**ボロー付き減算命令**
cmp	**比較命令**

上記算術演算は、汎用レジスタ間(R0～R15)、汎用レジスタ～即値間で行われます。add命令、sub命令は、さらにSP～即値間の演算にも対応しています。ワードサイズ以外の即値はcmp命令を除き、演算時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。ソースにワードサイズ以外の即値を指定した場合は、比較の際に符号拡張されます。

2.5.7 乗算・除算命令

S1C33000は乗除算機能もサポートします。ただし、オプションの乗算器を内蔵した機種に限られます。各機種のテクニカルマニュアルで確認してください。

(1)乗算命令

S1C33000の命令セットには4つの乗算命令が含まれています。

mlt.h	**16ビット×16ビット→32ビット(符号付き)**
mltu.h	**16ビット×16ビット→32ビット(符号なし)**
mlt.w	**32ビット×32ビット→64ビット(符号付き)**
mltu.w	**32ビット×32ビット→64ビット(符号なし)**

乗数と被乗数はそれぞれ指定の汎用レジスタ(R0～R15)のデータを使用します。16ビット乗算の場合は指定レジスタの下位16ビットが使用されます。符号付き乗算命令は乗数、被乗数のMSBを符号ビットとして扱います。
16ビット×16ビットの演算結果はALRにロードされます。32ビット×32ビットの演算結果は上位32ビットがAHRに、下位32ビットがALRにロードされます。
S1C33000は16ビット×16ビットの乗算を1サイクル、32ビット×32ビットの乗算を5サイクルで実行します。

(2)除算命令

S1C33000は符号付きおよび符号なしのステップ除算機能を持っています。

符号付きステップ除算に使用する命令: div0s, div1, div2s, div3s
符号なしステップ除算に使用する命令: div0u, div1

以下に、ステップ除算の実行手順と各命令の機能を示します。

1. ステップ除算の前処理(div0s, div0u)
除算を開始する前に、被除数をALR、除数をrsレジスタ(汎用レジスタR0～R15)に用意し、div0sまたはdiv0uを実行します。それぞれの命令は次のような動作を行います。

div0s(符号付きステップ除算の前処理)
・AHRにALR(被除数)の符号を拡張
　被除数が正の場合、AHRは0x00000000, 負の場合は0xFFFFFFFFに設定されます。
・PSRのDSフラグに被除数の符号ビットをセット
　被除数が正の場合、DSフラグは0にリセット, 負の場合は1にセットされます。
・PSRのNフラグに除数(rs)の符号ビットをセット
　除数が正の場合、Nフラグは0にリセット, 負の場合は1にセットされます。

div0u(符号なしステップ除算の前処理)
・AHRを0x00000000にクリア
・PSRのDSフラグを0にリセット
・PSRのNフラグを0にリセット

2. ステップ除算を実行
div1命令を必要ステップ数、実行します。たとえば、32ビット÷32ビットの場合はdiv1命令を32回実行します。
div1命令は符号付き除算、符号なし除算に共通です。
div1命令の1回の実行で、以下の処理が行われます。

1) {AHR, ALR}の64ビットを左(上位側)に1ビットシフト(ALR(0) = 0)

2) AHRとrsを加算またはAHRからrsを減算し、結果によりAHR、ALRを再設定
　加減算はAHRの内容にDSフラグを符号ビットとして付加した33ビットと、rsレジスタの内容にNフラグを符号ビットとして付加した33ビットデータによって行います。
　この処理はPSRのDSフラグおよびNフラグによって以下のように異なります。なお、演算結果の33ビット目の値をtmp(32)として説明します。

DS = 0(被除数: 正)、N = 0(除数: 正)の場合

2-1)　tmp = {0, AHR} - {0, rs}を実行
2-2)　tmp(32) = 1の場合: AHR = tmp(31:0)、ALR(0) = 1として終了
　　tmp(32) = 0の場合: AHR、ALRをそのままに終了

DS = 1(被除数: 負)、N = 0(除数: 正)の場合

2-1)　tmp = {1, AHR} + {0, rs}を実行
2-2)　tmp(32) = 0の場合: AHR = tmp(31:0)、ALR(0) = 1として終了
　　tmp(32) = 1の場合: AHR、ALRをそのままに終了

DS = 0(被除数: 正)、N = 1(除数: 負)の場合

2-1)　tmp = {0, AHR} + {1, rs}を実行
2-2)　tmp(32) = 1の場合: AHR = tmp(31:0)、ALR(0) = 1として終了
　　tmp(32) = 0の場合: AHR、ALRをそのままに終了

DS = 1(被除数: 負)、N = 1(除数: 負)の場合

2-1)　tmp = {1, AHR} - {1, rs}を実行
2-2)　tmp(32) = 0の場合: AHR = tmp(31:0)、ALR(0) = 1として終了
　　tmp(32) = 1の場合: AHR、ALRをそのままに終了

符号なし除算の場合は、div1命令を必要数実行することにより結果が次のレジスタから得られます。
AHR　余り　　ALR　商

符号付き除算の場合は、以下に説明する補正が必要です。

3. 符号付き除算の補正

符号付き除算の場合、div1命令を必要ステップ数実行後、div2s命令とdiv3s命令を続けて実行し、演算結果を補正します。
符号なし除算では、div2s命令とdiv3s命令を実行する必要はありません。なお、実行した場合でもnop命令と同様に機能し、演算結果には影響を与えません。

以下に、div2s命令とdiv3s命令の機能を示します。

div2s(符号付きステップ除算結果の補正1)

被除数が負の数の場合に除算のステップ(div1命令の実行)で演算結果がゼロになると、全ステップ終了後の演算結果が、除数と同じ余り(AHR)、および実際の値よりも絶対値で1少ない商(ALR)となる可能性があります。div2s命令はこの結果を補正します。
div2s命令の動作は次のとおりです。

DS = 0(被除数: 正)の場合

被除数が正の場合は上記の問題は発生しません。したがって、div2s命令は何も実行せずに終了します(nop命令と同じ)。

DS = 1(被除数: 負)の場合

1)　N = 0(除数: 正)の場合: tmp = AHR + rsを実行
　　N = 1(除数: 負)の場合: tmp = AHR - rsを実行

2)　1)の演算結果により、
　　tmpがゼロの場合:　　AHR = tmp(31:0)、ALR = ALR + 1として終了
　　tmpがゼロ以外の場合: AHR、ALRをそのままに終了

div3s(符号付きステップ除算結果の補正2)

ステップ除算によりALRから得られる商は常に正の数になります。被除数と除数の符号が異なる場合、結果は負でなければなりません。div3s命令はこの場合の符号補正を行います。

DS = N(被除数と除数が同符号)の場合

この場合は上記の問題は発生しません。したがって、div3s命令は何も実行せずに終了します(nop命令と同じ)。

DS = !N(被除数と除数の符号が異なる)の場合

ALR(商)の符号を反転します。

div2s命令およびdiv3s命令実行後、符号付き除算の最終結果が次のレジスタから得られます。

AHR　　余り　　ALR　商

除算の実行例

(1)符号付き32ビット÷32ビットの実行

(被除数がR0、除数がR1にロードされている場合)

```
ld.w   %alr,%r0  ; ALRに被除数を設定
div0s  %r1       ; 初期化ステップ
div1   %r1       ; ステップ除算
 :      :
div1   %r1       ; div1命令を32回実行
div2s  %r1       ; 補正命令1
div3s            ; 補正命令2
```

AHRに余り、ALRに商がロードされます。
この例の実行にかかる時間は36サイクルです。

符号付き除算の場合、除算結果の余りの符号は被除数と同じになります。
例: (-8) ÷ 5 = -1　余り -3
　8 ÷ (-5) = -1　余り 3

(2)符号なし32ビット÷32ビットの実行

(被除数がR0、除数がR1にロードされている場合)

```
ld.w   %alr,%r0  ; ALRに被除数を設定
div0u  %r1       ; 初期化ステップ
div1   %r1       ; ステップ除算
 :      :
div1   %r1       ; div1命令を32回実行
```

AHRに余り、ALRに商がロードされます。
この例の実行にかかる時間は34サイクルです。

2.5.8 積和演算命令

S1C33000は、"64ビット+16ビット×16ビット"の演算を指定回数実行する積和演算機能をサポートしています。この機能により、専用のDSPを外部に設けることなく、デジタル信号処理をオンチップで実現します。ただし、積和演算機能の乗算器を内蔵した機種に限られます。各機種のテクニカルマニュアルで確認してください。
積和演算はmac命令によって実行します。
"mac %rs"命令は"{AHR, ALR} ← {AHR, ALR} + H[<rs+1>]+ × H[<rs+2>]+"をrsレジスタで指定される回数分実行します。

rsレジスタには、積和演算開始前に繰り返し回数を設定しておきます。rsレジスタは回数カウンタとして使用され、演算ごとにデクリメントされます。rsレジスタが0になると、mac命令は終了します。したがって、2^{32}-1回(4,294,967,295回)までの繰り返しが可能です。rsレジスタに0を設定してmac命令を実行しても、積和演算は行われず、AHR、ALRも変更されません。rsレジスタも0のままデクリメントされません。

<rs+1>と<rs+2>はrsレジスタに続く2つの汎用レジスタです。
例: rsにR0レジスタを指定　　<rs+1>=R1レジスタ、<rs+2>=R2レジスタ
　　rsにR15レジスタを指定　　<rs+1>=R0レジスタ、<rs+2>=R1レジスタ

H[<rs+1>]+、H[<rs+2>]+は、上記のレジスタの内容をベースアドレスとして指定されるメモリのハーフワードデータを表します。積和演算は、それらを符号付き16ビットデータとして乗算し、結果を{AHR, ALR}レジスタペアに加算します。"+"は1回の演算後にそれぞれのベースアドレス(<rs+1>と<rs+2>レジスタの内容)がインクリメント(+2)されることを示します。

例: R0 = 16、R1 = 0x100、R2 = 0x120、AHR = ALR = 0に設定し、"mac %r0"を実行
1) {AHR, ALR} = 0 + H[0x100]×H[0x120]
2) {AHR, ALR} = {AHR, ALR} + H[0x102]×H[0x122]
3) {AHR, ALR} = {AHR, ALR} + H[0x104]×H[0x124]
:　　:
16) {AHR, ALR} = {AHR, ALR} + H[0x11E]×H[0x13E]

演算結果は、AHRを上位32ビット、ALRを下位32ビットとする符号付き64ビットデータとして得られます。レジスタの値は、R0 = 0、R1 = 0x120、R2 = 0x140となります。

積和演算中のオーバーフロー

積和演算中に演算結果が符号付き64ビットの範囲を越えると、オーバーフローとしてPSRのMOフラグが1にセットされます。この場合でも、rsレジスタに設定した回数を終了するまで演算は継続されます。MOフラグは、ソフトウェアによってリセットするまで1を保持します。mac命令の実行終了後にMOフラグを読み出すことで、演算結果が有効かどうかチェックできます。

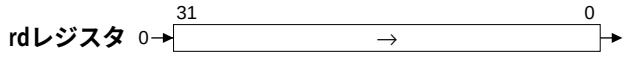
積和演算中の割り込み

mac命令の実行中は、繰り返しの途中であっても割り込みを受け付けます。割り込み処理ルーチンに分岐する際、スタックには実行中のmac命令のアドレスがリターンアドレスとしてセーブされます。したがって、割り込み処理ルーチンをreti命令で終了すると、中断していたmac命令の実行を再開します。ただし、その時点のrsレジスタの内容が残りのカウント数となりますので、割り込み処理ルーチン中でrsレジスタの内容が変更されると、当初設定した回数とは異なる結果となります。同様に、<rs+1>、<rs+2>レジスタの値が割り込み処理ルーチン中で変化すると、再開したmac命令は正しく実行されません。

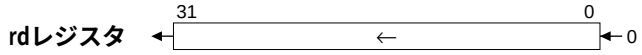
2.5.9 シフト&ローテート命令

S1C33000命令セットは、レジスタデータのシフト、ローテート命令をサポートしています。

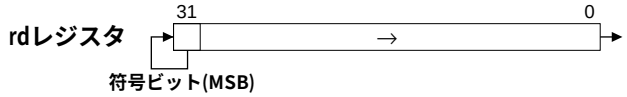
srl 論理右シフト



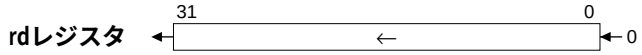
sll 論理左シフト



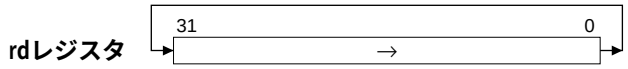
sra 算術右シフト



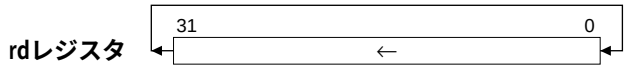
sla 算術左シフト



rr 右ローテート



rl 左ローテート



それぞれ、図のように指定の汎用レジスタのビットをシフトします。
シフト量は、各命令とも任意の汎用レジスタまたは即値により0から8ビットの範囲で指定可能です。

命令　%rd, %rs　　rdレジスタのビットをrsレジスタによる指定量シフト/ローテートします。
シフト量はrsレジスタのビット3からビット0の4ビットが有効となります。

命令　%rd, imm4　　rdレジスタのビットを符号なし4ビット即値imm4による指定量シフト/ローテートします。

rsレジスタまたはimm4で指定するシフト量は次のとおりです。

rs(3:0)/imm4	シフト量	
1xxx	8ビット	(x: 1または0)
0111	7ビット	
0110	6ビット	
0101	5ビット	
0100	4ビット	
0011	3ビット	
0010	2ビット	
0001	1ビット	
0000	0ビット	

2.5.10 ビット操作命令

メモリ上のデータをビット単位で操作するため、以下の4種類の命令が用意されています。これらの命令を使用することで、表示メモリやI/Oマップの制御ビットを直接変更することができます。

btst　[%rb], imm3　　指定ビットが0ならばZフラグをセット
bclr　[%rb], imm3　　指定ビットを0にクリア
bset　[%rb], imm3　　指定ビットを1にセット
bnot　[%rb], imm3　　指定ビットを反転(1↔0)

ビット操作は、rb(汎用)レジスタで指定されるメモリアドレスに対して行われます。imm3はそのアドレスにストアされているバイトデータのビット番号(ビット0～ビット7)を指定します。
これらの命令(btstを除く)により変更される内容は指定のビットのみですが、メモリアクセスはバイト単位のため、指定アドレスは書き換えられます。このため、ビットの書き込み動作により機能が有効となるようなI/O制御ビットが割り付けられているアドレスの操作には注意が必要です。

2.5.11 プッシュ&ポップ

汎用レジスタの内容をスタックに一時待避させるため、また退避させたデータを元のレジスタに復帰させるため、プッシュ命令とポップ命令が用意されています。

プッシュ命令 pushn %rs
この命令は、rsレジスタからR0レジスタまでを連続的にスタックにセーブします。

ポップ命令　popn %rd
この命令は、スタックのデータをR0レジスタからrdレジスタまで連続的にロードします。

例:

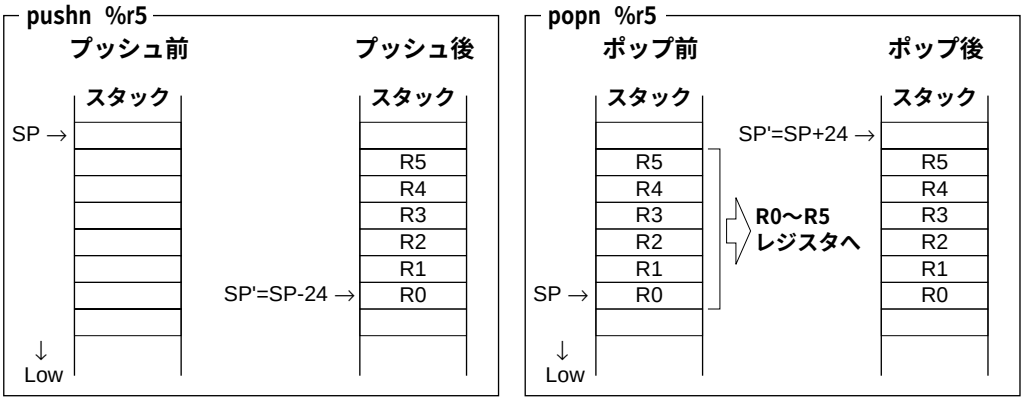


図2.5.11.1　汎用レジスタの退避と復帰

pushn命令とpopn命令は、同じレジスタ指定のものが対になっている必要があります。
これらの命令は退避/復帰するデータ数に従ってSPを変更します。

プッシュ/ポップ命令のほかに、SPをベースアドレスとした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([%sp+imm6])のロード命令も用意されていますので、SPを基準にしたレジスタ個別のストア/ロードも行えます。ただし、この場合はSPが変更されません。

2.5.12 分岐命令・ディレイド命令

分岐命令の種類

(1) PC相対ジャンプ命令("jr* sign8", "jp sign8")

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、現在のPCが示すアドレス(分岐命令のアドレス)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。なお、命令長が16ビット固定のため、sign8は16ビット単位のハーフワードアドレスを指定します。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sign8を2倍にした符号付き9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。また、ディスプレースメントを加えたPC値が28ビットのアドレス空間を越える場合は、越えた分(PCの上位4ビット)が無効となります。
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

```
jp      sign8       ; "jp sign9"として機能します。(sign9 = {sign8, 0})
```

分岐命令単独では、符号付き8ビットのディスプレースメント(sign8)が指定可能です。
sign8は16ビット単位の相対値のため、分岐範囲は[PC - 256 ～ PC + 254]です。

ext命令を1個拡張した場合

```
ext     imm13
jp      sign8       ; "jp sign22"として機能します。(sign22 = {imm13, sign8, 0})
```

ext命令で指定したimm13をsign22の上位13ビットとして拡張します。
分岐範囲は[PC - 2,097,152 ～ PC + 2,097,150]です。

ext命令を2個拡張した場合

```
ext     imm13
ext     imm13'
jp      sign8       ; "jp sign32"として機能します。
```

最初のext命令で指定したimm13はビット12～ビット3の10ビットのみが有効(下位3ビットを無視)で、sign32は次のように構成されます。

sign32 = {imm13(12:3), imm13', sign8, 0}

分岐範囲は[PC - 2,147,483,648 ～ PC + 2,147,483,646]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

jp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2.5.12.1に示します。

表2.5.12.1 条件ジャンプ命令と分岐条件

命令		フラグ条件	"cmp A, B"の結果	備考
jrgt	(Grater Than)	!Z & !(N ^ V)	A＞B	符号付きデータ比較用
jrge	(Grater or Equal)	!(N ^ V)	A≧B	
jrlt	(Less Than)	N ^ V	A＜B	
jrle	(Less or Equal)	Z | (N ^ V)	A≦B	
jrugt	(Unsigned, Grater Than)	!Z & !C	A＞B	符号なしデータ比較用
jruge	(Unsigned, Grater or Equal)	!C	A≧B	
jrult	(Unsigned, Less Than)	C	A＜B	
jrule	(Unsigned, Less or Equal)	Z | C	A≦B	
jreq:	(Equal)	Z	A＝B	符号付きおよび
jrne:	(Not equal)	!Z	A≠B	符号なしデータ比較用

フラグ条件の論理式が真(1)の場合に分岐します。(!: NOT, |: OR, &: AND, ^: XOR)

(2)絶対ジャンプ命令("jp %rb")

絶対ジャンプ命令"jp %rb"は、指定の汎用レジスタ(rb)の内容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、アドレス空間を外れる上位4ビットも無効となります。

(3)PC相対コール命令("call sign8")

PC相対コール命令"call sign8"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、現在のPCが示すアドレス(分岐命令のアドレス)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(sign8が2倍され)、偶数アドレスに分岐します。また、ディスプレースメントを加えたPC値が28ビットのアドレス空間を越える場合は、越えた分(PCの上位4ビット)が無効となります。
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ページの"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("call %rb")

絶対コール命令"call %rb"は、指定の汎用レジスタ(rb)の内容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、アドレス空間を外れる上位4ビットも無効となります。

(5)ソフトウェア例外("int imm2")

ソフトウェア例外"int imm2"は、ソフトウェアによって例外を発生させ、指定のトラップ処理ルーチンを実行するための命令です。4種類のトラップ処理ルーチンを作成することができ、imm2によってそれぞれのベクタ番号を指定します。CPUはソフトウェア例外が発生するとPSRと、intの次の命令アドレスをスタックにセーブし、トラップテーブルから指定のベクタを読み出してトラップ処理ルーチンを実行します。したがって、トラップ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。
ソフトウェア例外の詳細については、"3.3 トラップ(割り込みと例外)"を参照してください。

(6)リターン命令("ret", "reti")

ret命令はcall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命令はトラップ処理ルーチン用のリターン命令です。トラップ処理では、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。ret命令と同様にreti命令実行時とトラップ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)デバッグ例外("brk", "retd")

brk命令とretd命令はデバッグ例外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ICEソフト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、"3.6 デバッグモード"を参照してください。

ディレイド分岐機能

S1C33000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 jp.d ret.d

ディレイド命令

ディレイド命令は以下の条件をすべて満たしていることが必要です。

- 1サイクル命令
- メモリをアクセスしない
- ext命令による拡張なし

以下の命令はディレイド命令として使用することができます。

ld.w	%rd, %rs	ld.w	%rd, sign6		
add	%rd, %rs	add	%rd, imm6	add	%sp, imm10
adc	%rd, %rs				
sub	%rd, %rs	sub	%rd, imm6	sub	%sp, imm10
sbc	%rd, %rs				
mlt.h	%rd, %rs				
mltu.h	%rd, %rs				
cmp	%rd, %rs	cmp	%rd, sign6		
and	%rd, %rs	and	%rd, sign6		
or	%rd, %rs	or	%rd, sign6		
xor	%rd, %rs	xor	%rd, sign6		
not	%rd, %rs	not	%rd, sign6		
srl	%rd, %rs	srl	%rd, imm4		
sll	%rd, %rs	sll	%rd, imm4		
sra	%rd, %rs	sra	%rd, imm4		
sla	%rd, %rs	sla	%rd, imm4		
rr	%rd, %rs	rr	%rd, imm4		
rl	%rd, %rs	rl	%rd, imm4		
scan0	%rd, %rs				
scan1	%rd, %rs				
swap	%rd, %rs				
mirror	%rd, %rs				

注: 上記の条件を満たさない命令は動作が不定となるため、ディレイド命令として使用することは禁止します。

ディレイド命令は、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。

ディレイド分岐の使用できない分岐命令(".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。
分岐命令が条件分岐命令であって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命令の間は、割り込みや例外などのトラップはハードウェアによってマスクされ発生しません。

2.5.13 システム制御命令

以下の3つの命令はシステムを制御するもので、レジスタやメモリには影響を与えません。

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 CPUをHALTモードにします。
slp CPUをSLEEPモードにします。

HALTモード、SLEEPモードについては"3.4 パワーダウンモード"を参照してください。

2.5.14 スキャン命令

スキャン命令は、指定の汎用レジスタの上位8ビットをMSBからスキャンして、最初に見つかった1または0のビットの位置を返します。

scan0 %rd, %rs

rsレジスタの上位8ビットをスキャンして、最初の0のビットの位置(MSBからのオフセット)をrdレジスタにロードします。rdレジスタのビット31からビット4はすべて0になります。0が見つからない場合、rdレジスタには"0x00000008"がロードされ、Cフラグがセットされます。

例:

rsの上位8ビット	rdの下位8ビット	PSR			
		C	V	Z	N
0xxx xxxx	0000 0000	0	0	1	0
10xx xxxx	0000 0001	0	0	0	0
110x xxxx	0000 0010	0	0	0	0
1110 xxxx	0000 0011	0	0	0	0
1111 0xxx	0000 0100	0	0	0	0
1111 10xx	0000 0101	0	0	0	0
1111 110x	0000 0110	0	0	0	0
1111 1110	0000 0111	0	0	0	0
1111 1111	0000 1000	1	0	0	0

scan1 %rd, %rs

rsレジスタの上位8ビットをスキャンして、最初の1のビットの位置(MSBからのオフセット)をrdレジスタにロードします。rdレジスタのビット31からビット4はすべて0になります。1が見つからない場合、rdレジスタには"0x00000008"がロードされ、Cフラグがセットされます。

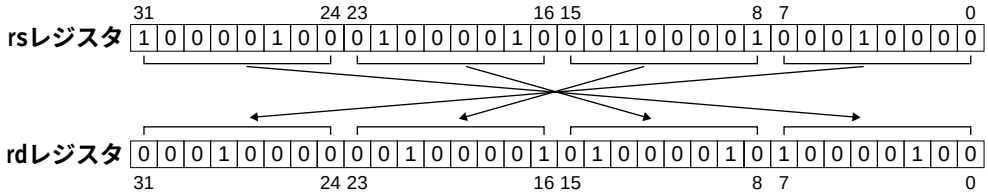
例:

rsの上位8ビット	rdの下位8ビット	PSR			
		C	V	Z	N
1xxx xxxx	0000 0000	0	0	1	0
01xx xxxx	0000 0001	0	0	0	0
001x xxxx	0000 0010	0	0	0	0
0001 xxxx	0000 0011	0	0	0	0
0000 1xxx	0000 0100	0	0	0	0
0000 01xx	0000 0101	0	0	0	0
0000 001x	0000 0110	0	0	0	0
0000 0001	0000 0111	0	0	0	0
0000 0000	0000 1000	1	0	0	0

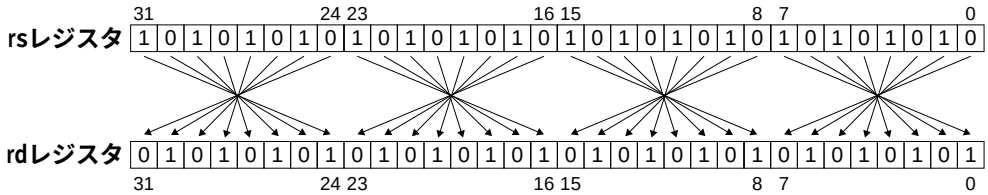
2.5.15 スワップとミラー命令

スワップ命令とミラー命令は汎用レジスタの内容を図のように入れ替えます。

スワップ命令: swap %rd, %rs



ミラー命令: mirror %rd, %rs



3　CPUの動作と処理状態

ここでは、CPUの処理状態と動作の概要を述べます。詳細についてはS1C33 Family各機種のテクニカルマニュアルを参照してください。

3.1　CPUの処理状態

S1C33000の状態遷移図を図3.1.1に示します。

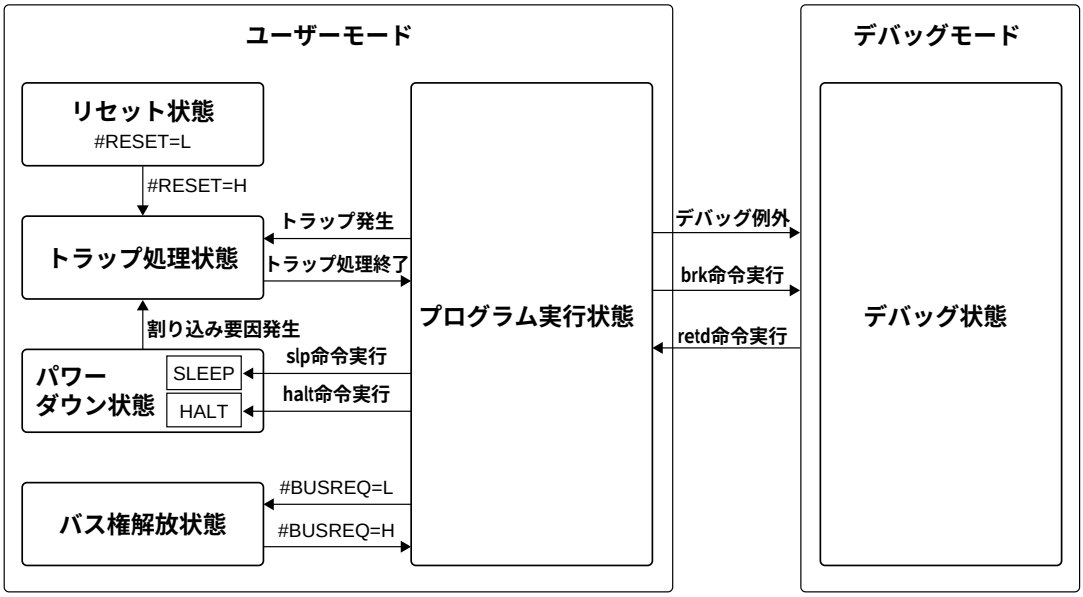


図3.1.1　状態遷移図

ユーザーモード

ユーザーモードはS1C33000がアプリケーションプログラムを実行するモードです。
イニシャルリセット後はこのモードで動作します。この中で、S1C33000は常に以下に示す5つの処理状態のいずれかに置かれています。

(1)リセット状態
　CPUの内部が初期化され、動作を停止している状態です。
(2)プログラム実行状態
　CPUがユーザープログラムをシーケンシャルに実行している状態です。
(3)トラップ処理状態
　CPUが、発生した割り込みや例外の処理ルーチンに分岐するための処理を行っている状態です。
(4)パワーダウン状態
　消費電流を低減するため、CPUが動作を停止している状態です。
(5)バス解放状態
　バス権を外部バスマスタに解放し、外部バスマスタが動作を終了するのを待っている状態です。

デバッグモード

S1C33000は開発効率を上げるため、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命令やデバッグ例外によってユーザー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。

3.2 プログラム実行状態

プログラム実行状態とは、ROMやRAM内のユーザープログラムをシーケンシャルに実行している状態です。実行中のアドレスはPC(プログラムカウンタ)が保持しており、各命令の実行によりインクリメントされます。分岐命令実行時は分岐先アドレスがPCにロードされ、そのアドレスに分岐します。
プログラム実行状態は、トラップの発生、haltまたはslp命令の実行、周辺回路からのバスリクエストによって中断され、CPUはそれぞれの処理状態に移行します。

3.2.1 プログラムのフェッチと実行

S1C33000は処理速度を上げるため、各命令の実行と実行予定の命令のフェッチを同時に行う3段のパイプライン処理を行っています。また、ハーバードアーキテクチャにより、内部ROM(プログラムメモリ)と内部RAM(データメモリ)を同時にアクセスすることができます。

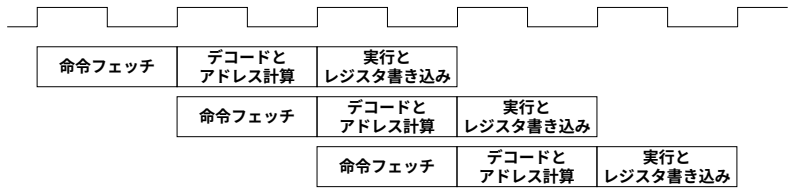


図3.2.1.1 プログラムのフェッチと実行

3.2.2 命令の実行サイクル数

S1C33000は主要な命令を1サイクルで実行することができます。各命令の実行サイクル数についてはAppendixの命令セット一覧表を参照してください。なお、本書に記載の実行サイクル数は内蔵ROMのプログラムと内蔵RAMのデータの場合です。以下に、外部メモリ/デバイスを使用する場合のサイクル数について補足しておきます。実行時間を計算する際には注意してください。なお、以下の計算方法は簡略化されたものです。実際はさらに、命令の組み合わせ、メモリマップの設定内容によって実行サイクルが変わる場合があります。

(1) 命令フェッチが内蔵ROM/内蔵RAM領域以外の場合、その領域の[ウェイトサイクル+1]サイクル分、実行サイクルが延びます。

(2) 内蔵RAM以外の領域に対してld.*命令でデータの書き込み・読み出しを行う場合、その領域の[ウェイトサイクル+1]サイクル分、実行サイクルが延びます。

(3) 内蔵RAMから命令をフェッチしながら内蔵RAMに対するデータアクセスを行う場合、1回のデータアクセスについて1サイクル分、実行サイクルが延びます。

(4) 命令フェッチ、データの書き込み・読み出しでは、転送されるデータサイズと接続されたデバイスサイズの組み合わせによって、バスオペレーションが1回、2回または4回実行されます。これに併せて実行サイクルが延びます。また、ウェイトサイクルも必要サイクル分だけ長くなります。たとえば、ゼロウェイトで8ビット幅の外部ROMから命令をフェッチする場合、2回のバスオペレーションが実行され、実行サイクルが3サイクル延びます。

(5) 上記の他にも、BCU(バスコントロールユニット)に設定された外部バス条件の中で以下の要因が実行サイクルに影響します。
・外部バス上のデバイスに対して設定されている出力ディセーブルサイクル
・DRAMのRASサイクル、プリチャージサイクルおよびリフレッシュサイクル
・外部#WAIT端子入力によるウェイトサイクル

(6) 以下の命令は実行中にデータを複数回アクセスします。したがって、その1回のデータアクセスにつき[ウェイトサイクル+1]サイクル分、実行サイクルが延びます。

・ビット操作命令(btst)	1回	・ソフトウェア例外(int)	3回
・ビット操作命令(bset, bclr, bnot)	2回	・トラップ処理ルーチンからのリターン(reti)	2回
・プッシュ/ポップ命令(pushn, popn)	n回	・デバッグ例外(brk)	3回
・積和演算命令(mac)	2n回	・デバッグ例外処理ルーチンからのリターン(retd)	2回

(7) インターロックによる遅延
メモリからデータを汎用レジスタに転送するロード命令のディスティネーションレジスタ(%rd)を直後の命令で演算のソースとして使用する場合は(%rsまたは%rdが直前の%rdと同じ場合)、インターロックを解消するために実行サイクルが1サイクル延びます。

BCUおよびウェイト等の外部バス条件については、S1C33 Family各機種のテクニカルマニュアルを参照してください。

3.3 トラップ(割り込みと例外)

CPUはプログラム実行中に、トラップ(割り込みや例外)が発生するとトラップ処理状態となります。トラップ処理状態は、各割り込み/例外要因に対応したユーザー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。

3.3.1 トラップテーブル

S1C33000のトラップ一覧を表3.3.1.1に示します。

表3.3.1.1 トラップ一覧

トラップ名称	同期/非同期	分類	ベクタアドレス	優先順位	トラップ発生後の割り込みレベル
リセット	非同期	割り込み	base+0x0	最も高い	レベル0
Reserved			base+0x4~0xC	↑	変更なし
ゼロ除算	同期	例外	base+0x10		変更なし
Reserved			base+0x14		変更なし
アドレス不整例外	同期	例外	base+0x18		変更なし
デバッグ例外(brk、その他)	同期	例外	0x0 or 0x60000		変更なし
NMI	非同期	割り込み	base+0x1C		変更なし
Reserved			base+0x20~0x2C		変更なし
ソフトウェア例外0	同期	例外	base+0x30		変更なし
:	:	:	:		:
ソフトウェア例外3	同期	例外	base+0x3C		変更なし
マスク可能な外部割り込み0	非同期	割り込み	base+0x40		割り込み要求元の
:	:	:	:	↓	割り込みレベル
マスク可能な外部割り込み215	非同期	割り込み	base+0x39C	最も低い	(レベル0~15)

トラップ名称に挙げた7種類が、S1C33000で扱うトラップ要因です(それぞれの詳細は後述します)。

同期/非同期は、そのトラップがプログラムの実行に同期して発生するか、非同期に発生するかを示しています。同期して発生するものを「例外」、非同期に発生するものを「割り込み」として分類しています。トラップの発生によって行うCPUの処理について、本書では一括してトラップ処理と記述しています。

ベクタアドレスは、各トラップが発生した場合に実行するユーザーのトラップ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、それぞれワード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トラップテーブルと呼び、ベクタアドレス欄に示した"base"はトラップテーブルのベース(先頭)アドレスを表します。
S1C33000では、トラップテーブルのベースアドレスをTTBRレジスタによって設定することができます。
TTBR0 = D(9:0)/0x48134: トラップテーブルベースアドレス(9:0) ...0に固定
TTBR1 = D(F:A)/0x48134: トラップテーブルベースアドレス(15:10)
TTBR2 = D(B:0)/0x48136: トラップテーブルベースアドレス(27:16)
TTBR3 = D(F:C)/0x48136: トラップテーブルベースアドレス(31:28) ...0に固定

コールドスタート(3.3.3節参照)によるイニシャルリセット時、TTBRレジスタはBTA3端子の状態で決まるブートアドレスに設定されます。

表3.3.1.2 コールドスタート後のトラップテーブルの位置

BTA3端子	トラップテーブルの位置
High	エリア3(内蔵ROMの先頭、base=0x0080000)
Low	エリア10(外部ROMの先頭、base=0x0C00000)

したがって、トラップテーブルの位置を変更する場合でも、コールドスタート用のリセットベクタのみは上記のアドレスに書き込んでおくことが必要です。ホットスタートによるイニシャルリセット時は、TTBRレジスタは初期化されません。

TTBR0とTTBR3は読み出し専用で"0"に固定されます。このため、トラップテーブルの先頭アドレスは常に1KB境界アドレスから始まります。
なお、TTBRレジスタは誤って書き換えられることのないように、通常は書き込み禁止状態に置かれ、この書き込み保護機能を解除するためにTBRPレジスタ(D(7:0)/0x4812D)が用意されています。TBRPレジスタに0x59を書き込むとTTBRレジスタへの書き込みが許可され、TTBRレジスタの最上位バイト(0x48137)への書き込みにより書き込み禁止状態に戻ります。したがって、TTBRレジスタへの書き込みは下位ハーフワードから先に行うことが必要です。ただし、下位と上位ハーフワードの書き込みの間にNMI等が発生すると誤動作しますので、ワード書き込みを推奨します。

機種によってアクセス可能なメモリ空間は異なります。各ベクタ用にワードサイズの領域が確保されていますが、実際にベクタとして使用されるのは下位側の有効なビット長のみです。また、ベクタはプログラムメモリのアドレスのため、最下位ビットは常に0として扱われます。
トラップテーブルのサイズは機種により設定されるマスク可能な割り込みの数で決まります。

優先順位は、複数のトラップが同時に発生した場合に、どれが先にCPUに受け付けられるかを示しています。複数の例外が同時に発生することはありません。リセット要因は他のすべての処理に優先して受け付けられます。マスク可能な割り込みについては、後述する割り込みレベルによっても優先度が管理されます。したがって、表3.3.1.1に示したマスク可能な割り込みの優先順位は、すべてが同じ割り込みレベルを持つ場合のものです。

トラップ発生後の割り込みレベルについては、"3.3.8 マスク可能な外部割り込み"で説明します。

3.3.2 トラップ処理

リセットおよびデバッグ例外を除いたトラップが発生すると、CPUは以下に示すトラップ処理を実行します。なお、以下の処理はリセット処理には適用されません。リセットについては次節で、デバッグ例外については3.6節で説明します。

(1) 実行中の命令を終了またはキャンセルします。
(2) PC、PSRの順にそれぞれの内容をスタックに待避させます。
(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。発生したトラップ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) トラップテーブルから、発生したトラップのベクタを読み出しPCにセットします。これにより、ユーザーのトラップ処理ルーチンに分岐します。

ここまでが、CPUのトラップ処理です。

ユーザーのトラップ処理ルーチンの最後にreti命令を実行すると、スタックに待避していたPSRとPCの内容がそれぞれのレジスタに戻り、トラップの発生によって中断していた処理を再開します。
トラップ処理ルーチンからのリターンには、PCのみを復帰するret命令は使用できません。

なお、リセットを除くトラップは以下の場合にマスクされ、その要因が解除されるまでCPUには受け付けられません。

(1) ext命令実行時
ext命令があると、続くターゲット命令の実行を終了するまでトラップはマスクされます。
ただし、アドレス不整例外の場合を除きます。

(2) ディレイド分岐命令実行時
ディレイド分岐命令(.d)があると、続くディレイド命令の実行を開始するまでトラップはマスクされます。

(3) SP未設定時のNMI
CPUをリセット後、SPにデータを書き込む(スタックが設定される)まではプログラムの暴走を防ぐためNMIがマスクされます。
例外は発生が予測できるため、マスクの対象とはなりません。また、リセット後はマスク可能な割り込みもPSRのIEビットによりマスクされているため同様にマスクの対象とはなりません。

3.3.3 リセット

CPUの#RESET端子にLowパルスを入力することで、CPUがリセットされます。これによりPSRの全ビットが0にクリアされます。他のレジスタは不定となります。
CPUは#RESETパルスの立ち上りエッジで動作を開始し、リセット処理を行います。リセット処理ではトラップ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、ユーザー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。

S1C33000は、コールドスタートとホットスタートの2種類のリセット方式をサポートしています。この条件設定には#NMI端子を#RESET端子とともに使用します。

コールドスタート(#RESET=L, #NMI=H)

#RESET端子をLow、#NMI端子をHighにしてリセットすると、S1C33 FamilyのMPUはコールドスタートします。コールドスタートの場合、CPUの他にチップ上の周辺回路もすべて初期化されますので、主にパワーオンリセットに用います。

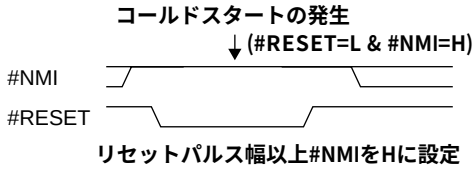


図3.3.3.1　コールドスタートのタイミング

ホットスタート(#RESET=L, #NMI=L)

#RESET端子および#NMI端子をLowにしてリセットすると、S1C33 FamilyのMPUはホットスタートします。ホットスタートの場合、CPUは初期化されますが、周辺回路の中で外部バスコントロールユニットや入出力ポートなどは初期化されません。外部メモリや外部入出力の状態を保持したままリセットをかけたい場合に用います。

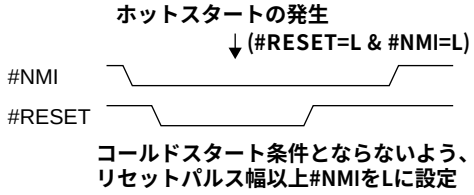


図3.3.3.2　ホットスタートのタイミング

リセットタイミングや周辺回路の初期化については、S1C33 Family各機種のテクニカルマニュアルを参照してください。

3.3.4 ゼロ除算例外

除算命令実行時に除数がゼロであると、このゼロ除算例外が発生します。
この例外が発生するのは、除算の前処理を行うdiv0sまたはdiv0uの2命令です。除数が0の場合、CPUはこの命令の実行を終了後、トラップ処理に移行します。トラップ処理でスタックにセーブするPC値は、次の命令(通常div1命令)のアドレスとなります。
ただし、パイプライン処理の関係で、例外の発生が1命令後ろにずれる場合があります。

3.3.5 アドレス不整例外

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 (使用領域の全アドレスが対象)
ld.h/ld.uh	ハーフワード (16ビット)	ハーフワード境界 (アドレスの最下位ビットが常に0)
ld.w	ワード (32ビット)	ワード境界 (アドレスの下位2ビットが常に0)

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、CPUはアドレス不整例外としてトラップ処理に移行します。この場合、ロード命令は実行されません。トラップ処理でスタックにセーブするPC値は、例外を発生したロード命令のアドレスとなります。
通常のext命令実行時は、次の命令との間でトラップがマスクされるようになっています。ただし、アドレス不整例外のみはマスクされません。したがって、ext命令に続くロード命令(ext命令による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)でアドレス不整例外が発生した場合は、上記のように、そのロード命令を実行前にトラップ処理に移行します。この場合、処理ルーチンから単純にreti命令で戻ると、そのロード命令のみが単独で(ext命令のないレジスタ間接アドレッシングとして)実行されてしまいますので注意が必要です。

積和演算(mac)命令もメモリ上のハーフワードデータを扱うため、アドレス不整例外が発生する可能性があります。この場合も、スタックにセーブされるリターンアドレスはmac命令のアドレスとなり、トラップ処理ルーチンからのリターン後は残りの回数の積和演算を継続します。

ベースアドレスとしてSPを使用するロード命令においては、アドレスがデータサイズに応じてアライメントされるため、アドレス不整例外は発生しません。

プログラムの分岐を伴う命令("call %rb"、"jp %rb")では、PCの最下位ビットが常に0に固定されるため、この例外は発生しません。トラップ処理のベクタについても同様です。

3.3.6 NMI(マスク不可能な割り込み)

CPUの#NMI入力がアクティブ(Low)になるとNMIが発生します。NMIが発生すると、CPUは実行中の命令を終了後、トラップ処理に移行します。トラップ処理でスタックにセーブするPC値は、次の命令のアドレスとなります。
NMIはマスクが不可能な割り込みですが、CPUリセット後(コールドスタート、ホットスタートとも)のSPが不定な期間に発生するとプログラムが暴走するため、"ld.w %sp, %rs"命令によってSPを設定するまではハードウェアによってCPUの#NMI入力がマスクされるようになっています。

3.3.7 ソフトウェア例外

ソフトウェア例外は"int imm2"命令の実行によって発生します。このトラップ処理でスタックにセーブするPC値は、次の命令のアドレスとなります。int命令中のオペランドimm2は、4種類のソフトウェア例外のベクタアドレスを指定します。CPUはbase+48(ソフトウェア例外0ベクタアドレス)に4×imm2を加算したアドレスからベクタを読み出して処理ルーチンに分岐します。

3.3.8 マスク可能な外部割り込み

S1C33000は128種類までのマスク可能な外部割り込み(NMIを除く)を受け付けることができます。
マスク可能な割り込みは、PSRのIE(割り込みイネーブル)ビットがセットされている場合にのみCPU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(0〜15)はCPU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。
IEビットとILフィールドはソフトウェアで設定可能です。また、トラップ発生時にはPSRをスタックにセーブ後、IEビットは0(割り込み禁止)にクリアされ、処理ルーチン内でIEビットをセットするか、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ビット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。
CPU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(1〜15の割り込みレベルを許可)に設定されます。

S1C33 Familyの全機種がチップ上に割り込みコントローラを内蔵しているため、CPUへの割り込み要求は割り込みコントローラが行います。
オンチップ割り込みコントローラによるマスク可能な割り込みの発生手順とCPUのトラップ処理の内容は次のとおりです。

(1) オンチップ割り込みコントローラは、#INTREQ端子をLowにして割り込みを要求します。同時に、INTLEV(3:0)に割り込みレベルを、INTVEC(7:0)にベクタ番号を出力します。
(2) CPUはその割り込み要求を受け付けるとPCとPSRをスタックにセーブ後、PSRのIEビットをクリアし、ILフィールドをINTLEVの状態に従った割り込みレベルに設定します。
(3) CPUはトラップテーブル内のINTVECで指定される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。

オンチップ割り込みコントローラの制御方法については、各機種のテクニカルマニュアルを参照してください。

3.4 パワーダウンモード

システムがキー入力待ちなど、特にプログラムを実行する必要のない待機状態の場合、CPUの動作を停止して消費電流を低減することができます。このために、S1C33000は2種類のパワーダウンモード、HALTモードとSLEEPモードをサポートしています。
なお、パワーダウンモード中もCPU内部のレジスタの状態は保持されます。

3.4.1 HALTモード

CPU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CPUの動作が停止します。チップ上の周辺回路に対してはクロックが供給されますので、周辺回路は動作を継続します。
HALTモードはイニシャルリセットまたはNMIを含む割り込みによって解除され、解除後はそれぞれのトラップ処理を経てプログラム実行状態に移行します。割り込みによって解除した場合、トラップ処理によってhaltの次の命令のアドレスがスタックにセーブされます。したがって、発生した割り込みの処理ルーチンをreti命令で終了すると、haltの次の命令の位置にリターンします。

3.4.2 SLEEPモード

CPU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CPUおよびチップ上の周辺回路も動作を停止します。このため、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。
SLEEPモードはイニシャルリセットまたはNMIを含む割り込みによって解除され、解除後はそれぞれのトラップ処理を経てプログラム実行状態に移行します。割り込みによって解除した場合、トラップ処理によってslpの次の命令のアドレスがスタックにセーブされます。したがって、発生した割り込みの処理ルーチンをreti命令で終了すると、slpの次の命令の位置にリターンします。
なお、SLEEPモードではチップ上の発振回路およびその出力クロックを使用する周辺回路が基本的に停止するため、SLEEPモードの解除はキー入力割り込み等により行われます。
また、SLEEP解除により発振回路が動作を開始するため、CPUの動作開始には発振安定待ち時間が必要となります。

HALTモード、SLEEPモード時の周辺回路の状態、解除方法等の詳細については、各機種のテクニカルマニュアルを参照してください。

3.5 バス権解放状態

周辺回路/デバイスが接続される外部バスは、通常CPUの管理下に置かれています。S1C33000は、DMA(ダイレクト・メモリ・アクセス)を必要とする周辺回路/デバイスやマルチCPUシステムに対応するため、バス権を外部に解放することが可能となっています。
この制御には、CPUの#BUSREQ端子と#BUSACK端子を使用します。
バス権解放のシーケンスは次のとおりです。

(1) バス権を要求する外部デバイスは、CPUの#BUSREQ端子をLowレベルにします。
(2) CPUは常に#BUSREQ端子の状態を監視しており、端子がLowレベルになると実行中のバスサイクルを終了し、その1サイクル後にアドレスバス(A27～A0)、データバス(D15～D0)、バス制御信号(#RD, #WRL, #WRH)をハイインピーダンスにします。
さらに1サイクル後、CPUは#BUSACK端子をLowレベルにしてバス権を解放したことを、外部デバイスに知らせます。
(3) (2)によってバス権を要求した外部デバイスがバスマスタとなり、自らのバスサイクルを実行します。
この間、外部バスマスタはCPUの#BUSREQ端子をLowに固定しておく必要があります。
(4) 外部バスマスタは必要なバスサイクルを終了後、バスをハイインピーダンスにしてから#BUSREQ端子をHighレベルに戻します。
(5) CPUは#BUSREQ端子のHighレベルを確認すると、その1サイクル後に#BUSACK端子をHighレベルにし、中断していた処理を再開します。

機種によっては、バス権解放中にCPUがバス権を取り戻すことが必要な場合もあります(たとえば、DRAMを直結する機種のリフレッシュ要求等)。その場合は、CPUの要求を出力ポートなどの周辺回路を利用して出力するため、外部バスマスタとなるデバイスはその信号への対応も必要となります。詳細については、S1C33 Family各機種のテクニカルマニュアルを参照してください。

3.6 デバッグモード

S1C33000はデバッグモードという特殊な動作モードを持っています。
このデバッグモードはターゲットプログラム開発時のデバッグを支援するために設定されており、量産用のアプリケーションで使用することはできません。ここでは、CPUの機能としての概要を説明します。

3.6.1 デバッグモードの機能

S1C33000は以下のデバッグ機能を内蔵しています。

- **シングルステップ**
  ユーザーターゲットプログラムの各命令の実行前にデバッグ例外を発生させることができます。
- **命令ブレーク**
  設定したアドレスを実行する直前にデバッグ例外を発生させることができます。アドレスは3ヶ所まで設定することができます。
- **データブレーク**
  設定したアドレスに対して読み出し/書き込みが行われると、読み出しまたは書き込みの実行より1～数命令後にデバッグ例外を発生させることができます。アドレスは1ヶ所のみ設定することができます。
- **ソフトウェアブレーク**
  brk命令の実行によりデバッグ例外を発生することができます。デバッグ例外発生時の退避アドレスはbrk命令の次のアドレスとなります。

これらのデバッグ例外が発生すると、CPUはユーザーモードとは異なる例外処理を行いデバッグモードに移行します。
デバッグモードでは、ユーザーが作成したデバッグ処理ルーチン、あるいはセイコーエプソンが提供するデバッグ処理ルーチンを実行することにより、ユーザーターゲットプログラムを任意のアドレスでブレークさせたり、シングルステップなどの実行を行うことができます。

3.6.2 エリア2の構成

S1C33000はアドレス空間の中のエリア2(0x0060000～0x007FFFFの128Kバイト)をICE(In-Circuit Emulator)用の予約エリアとしています。このエリア2にデバッグ機能を制御するレジスタが配置されています。
0x0060010～0x0077FFFがICE用制御プログラム領域として予約され、0x0078000以降の領域はデバッグ機能を制御するレジスタとCPUの専用領域として予約されています。
エリア2のデバッグ機能を制御するレジスタへのデータ書き込みは、ユーザーモードでは行えません。デバッグ例外発生後のデバッグモードで行う必要があります。なお、デバッグモードでは特に各エリアへのアクセス条件はありませんので、すべてのエリアに対してアクセスが可能です。

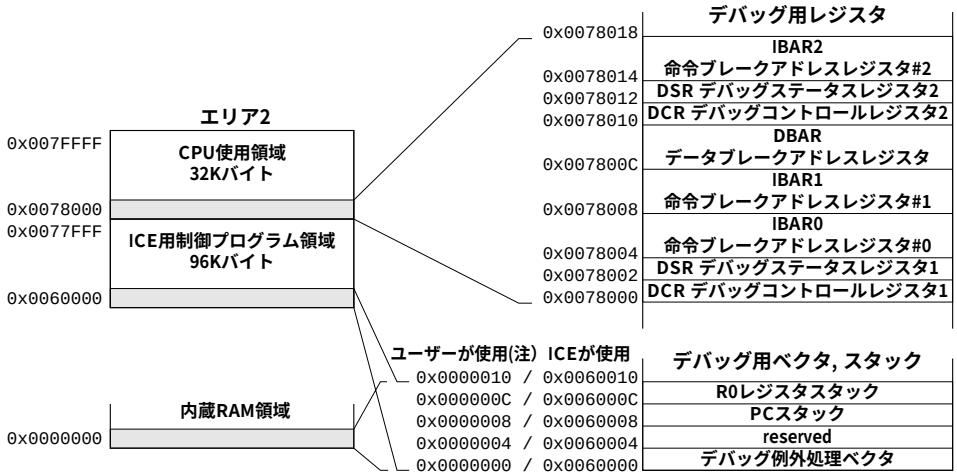


図3.6.2.1 エリア2の構成

注: ユーザーがデバッグモードを使用した場合には、0x0000000のアドレスからデバッグ例外処理ベクタを読み出し、また0x00000008、0x0000000CにそれぞれPCとR0レジスタを退避します。MON33(デバッグモニタ)は、このユーザー用の設定で作成されています。

3.6.3 ユーザーモードからデバッグモードへの移行

デバッグ例外が発生すると(例:brk命令の実行)、CPUはデバッグ例外処理に移行します。
このデバッグ例外処理は通常の例外処理とは異なり、通常のトラップテーブルを使用せずにエリア0の0x0000000またはICEが使用する0x0060000に書き込まれたベクタを読み出してデバッグモードに移行します。同時に動作モードをユーザーモードからデバッグモードに切り換えます。この際に退避されるレジスタもR0レジスタとPCで、通常の例外処理とは異なります。また、この退避領域も通常の例外処理とは異なり、R0レジスタには0x000000CまたはICEが使用する0x006000C、PCには0x0000008またはICEが使用する0x0060008の固定された領域が使用されます。

デバッグモードからユーザーモードへの移行は、retd命令を実行することにより行われます。retd命令は、R0レジスタとPCの内容をデバッグモード移行前の状態に復帰させ、ユーザーモードに戻します。

3.6.4 デバッグ用レジスタ

エリア2にはデバッグ機能を制御するデバッグ用レジスタが設定されており、デバッグモード時に書き込みが可能となります。以下に各レジスタの内容と機能を示します。

DCR(デバッグコントロールレジスタ): 0x0078000/バイト, 0x0078010/バイト

	7	6	5	4	3	2	1	0	
0x0078000	–	MWRBE	MRDBE	DBE	IBE(1:0)		SE	DM	R/W可 (DMはRのみ)
0x0078010	–	–	(注)	(注)	(注)	(注)	(注)	IBE(2)	R/W可

注: 0x0078010のビット5〜1は以下の値以外に設定しないでください。デバッグ機能が正常に動作しなくなります。
ビット5, 4: "0"固定、ビット3〜1: "1"固定

DCRはデバッグ機能を許可/禁止するためのレジスタです。イニシャルリセット時、DCRの全ビットは0に初期化されます。

0x0078000

名称	ビット番号	ビットの状態		機　能
		1	0	
DM	0	デバッグモード	ユーザーモード	Debug Mode: CPUがデバッグモードにあることを示します。 デバッグ例外が発生すると、DMがセット(1)され、CPUがデバッグモードになります。デバッグ処理ルーチンでretd命令を実行するとリセット(0)され、CPUはユーザーモードに戻ります。DMは読み出し専用で、ソフトウェアによる書き込みはできません。
SE	1	許可	禁止	Single Step Enable: シングルステップ機能を許可/禁止します。 SEをセット(1)するとシングルステップ機能が許可され、ユーザーモードのプログラム実行時は各命令実行前にデバッグ例外が発生します。デバッグモード時はシングルステップ動作とはなりません。 SEをリセット(0)するとシングルステップ機能は禁止されます。
IBE(1:0)	2, 3	許可	禁止	Instruction Break Enable: 命令ブレークポイントを許可/禁止します。 IBE(0)(ビット2)、IBE(1)(ビット3)はそれぞれ命令ブレークポイント#0、#1に対応します。IBEビットをセット(1)するとIBAR0/IBAR1レジスタに設定したブレークアドレスが有効となります。ユーザーモードでのプログラム実行時にそのアドレスの命令をフェッチすると、命令の実行前にデバッグ例外が発生します。デバッグモードでのプログラム実行中はブレークしません。 IBEビットをリセット(0)すると命令ブレークポイントは無効となります。
DBE	4	許可	禁止	Data Break Enable: データブレークを許可/禁止します。 DBEをセット(1)するとDBARレジスタに設定したデータブレークアドレスが有効となります。ユーザーモードでのプログラム実行時にそのアドレスをアクセスすると、データアクセス実行後にデバッグ例外が発生します。デバッグモードでのプログラム実行中はブレークしません。データブレークを発生させるアクセス条件(読み出し, 書き込み, 読み出し/書き込み)はMRDBEおよびMWRBEによって指定可能です。 DBEをリセット(0)するとデータブレーク機能は無効となります。また、DBEがセットされている場合でも、MRDBE(読み出し)およびMWRBE(書き込み)が共にリセットされているときには、データブレークは発生しません。

名称	ビット番号	ビットの状態		機　能
		1	0	
MRDBE	5	許可	禁止	Memory Read Break Enable: メモリ読み出しによるデータブレークを許可/禁止します。 DBEがセット(1)され、かつMRDBEがセット(1)されている場合、データブレークは指定アドレスのデータをCPUが読み出した後に発生します。 MRDBEをリセット(0)すると、メモリ読み出しによるデータブレーク機能は無効となります。
MWRBE	6	許可	禁止	Memory Write Break Enable: メモリ書き込みによるデータブレークを許可/禁止します。 DBEがセット(1)され、かつMWRBEがセット(1)されている場合、データブレークは指定アドレスにCPUがデータを書き込んだ後に発生します。 MWRBEをリセット(0)すると、メモリ書き込みによるデータブレーク機能は無効となります。

0x0078010

名称	ビット番号	ビットの状態		機　能
		1	0	
IBE(2)	0	許可	禁止	Instruction Break Enable: 命令ブレークポイントを許可/禁止します。 IBE(2)は命令ブレークポイント#2に対応します。IBE(2)ビットをセット(1)するとIBAR2レジスタに設定したブレークアドレスが有効となります。ユーザーモードでのプログラム実行時にそのアドレスの命令をフェッチすると、命令の実行前にデバッグ例外が発生します。デバッグモードでのプログラム実行中はブレークしません。 IBE(2)ビットをリセット(0)すると命令ブレークポイントは無効となります。

DSR(デバッグステータスレジスタ): 0x0078002/バイト, 0x0078012/バイト

	7	6	5	4	3	2	1	0	
0x0078002	BKF	MWRB	MRDB	DB	IB1	IB0	SS	DR	R/W可
0x0078012	–	–	–	–	–	–	–	IB2	R/W可

DSRは発生したデバッグ例外を示すステータスレジスタです。デバッグ例外が発生した場合、例外の種類にかかわらず同一のベクタによりデバッグ処理処理に移行します。したがって、デバッグ例外を処理するルーチン内でDSRを読み出し、発生したデバッグ例外の種類を判定する必要があります。

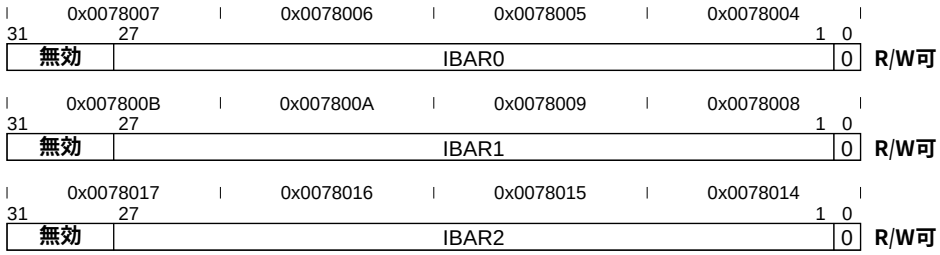
0x0078002

名称	ビット番号	ビットの状態		機　能
		1	0	
DR	0	発生	なし	Debug Request: 外部よりのデバッグ要求の発生を示します。DRは外部デバッグ要求信号#DBGREQの立ち下がりエッジでセット(1)されます。この機能はICE用で、通常のチップに#DBGREQ端子はありません。
SS	1	発生	なし	Single Step: シングルステップブレークの発生を示します。SSはシングルステップによるデバッグ例外が発生するとセット(1)されます。
IB0	2	発生	なし	Instruction Break 0: 命令ブレーク#0の発生を示します。IB0は命令ブレークポイント#0によるデバッグ例外が発生するとセット(1)されます。
IB1	3	発生	なし	Instruction Break 1: 命令ブレーク#1の発生を示します。IB1は命令ブレークポイント#1によるデバッグ例外が発生するとセット(1)されます。
DB	4	発生	なし	Data Break: データブレークの発生を示します。DBはデータブレークポイントによるデバッグ例外が発生するとセット(1)されます。
MRDB	5	発生	なし	Memory Read Break: メモリ読み出しによるデータブレークの発生を示します。MRDBはメモリ読み出しによるデータブレークが発生するとセット(1)されます。
MWRB	6	発生	なし	Memory Write Break: メモリ書き込みによるデータブレークの発生を示します。MWRBはメモリ書き込みによるデータブレークが発生するとセット(1)されます。
BKF	7	発生	なし	Break Flag: brk命令によるデバッグ例外の発生を示します。BKFはbrk命令の実行によるデバッグ例外が発生するとセット(1)されます。

0x0078012

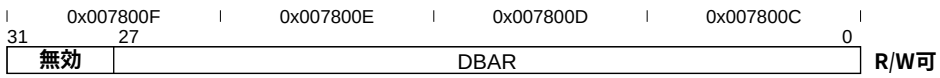
名称	ビット番号	ビットの状態		機　能
		1	0	
IB2	0	発生	なし	Instruction Break 2: 命令ブレーク#2の発生を示します。IB2は命令ブレークポイント#2によるデバッグ例外が発生するとセット(1)されます。

IBAR0(命令ブレークアドレスレジスタ#0): 0x0078006(bit27–16), 0x0078004(bit15–0)
IBAR1(命令ブレークアドレスレジスタ#1): 0x007800A(bit27–16), 0x0078008(bit15–0)
IBAR2(命令ブレークアドレスレジスタ#2): 0x0078016(bit27–16), 0x0078014(bit15–0)



命令ブレークアドレス#0～#2を設定します。最下位ビットは常に"0"として扱われ、ビット27～ビット1の27ビットのみが有効となります。
DCRのIBE(0)/IBE(1)/IBE(2)がセット(1)されていると、ユーザーモードでのプログラム実行時にIBAR0/IBAR1/IBAR2レジスタの値がPCと比較され、一致した場合にデバッグ例外が発生します。これらのレジスタはリード/ライト可能です。

DBAR(データブレークアドレスレジスタ): 0x007800E(bit27–16), 0x007800C(bit15–0)



データブレークアドレスを設定します。DCRのDBEがセット(1)されていると、ユーザーモードでのプログラム実行時にDBARレジスタの値がアクセスしたメモリのアドレスと比較され、読み出し/書き込み条件と共に一致した場合にデバッグ例外が発生します。このレジスタはリード/ライト可能です。
全ビットがアクセスするメモリのベースアドレスに一致しないとデータブレークは発生しません。したがってワードデータの読み出し/書き込みによってデータブレークを発生させたい場合はワード境界アドレス(下位2ビットが0)を設定しておくことが必要です。同様にハーフワードデータの場合はハーフワード境界アドレス(最下位ビットが0)を設定しておく必要があります。

3.6.5 デバッグモード時のトラップ

デバッグモードでは、リセット、アドレス不整例外、ゼロ除算例外、int命令を除くすべての例外および割り込み(NMIも含む)はマスクされ発生しません。アドレス不整例外、ゼロ除算例外、int命令による例外が発生した場合には通常の例外処理が行われます。
また、retd命令によってデバッグモードからユーザーモードに復帰した場合も、復帰アドレスの1命令を実行するまではリセット、アドレス不整例外を除くすべての例外および割り込みがマスクされます。
ただし、命令を実行後に発生する例外および割り込みはマスクされません。

3.6.6 デバッグ例外の同時発生

複数のデバッグ例外要因が同時に発生した場合、その要因すべてに対応するDSR内のビットがセットされ、デバッグ例外は1回のみ発生します。

4 命令の詳細説明

本章はS1C33000命令セットの各命令を、アルファベット順に解説します。

4.1 記号の意味

4.1.1 レジスタ

以下のシンボルはレジスタまたはその内容を表します。

%rd, rd: ディスティネーションとして使用する汎用レジスタ(R0～R15)またはその内容です。
%rs, rs: ソースとして使用する汎用レジスタ(R0～R15)またはその内容です。
%rb, rb: レジスタ間接アドレッシングでアクセスされるベースアドレスを保持している汎用レジスタ(R0～R15)またはその内容です。
%sd, sd: ディスティネーションとして使用する特殊レジスタ(PSR, SP, ALR, AHR)またはその内容です。
%ss, ss: ソースとして使用する特殊レジスタ(PSR, SP, ALR, AHR)またはその内容です。
%sp, sp: スタックポインタ(SP)またはその内容です。

ニーモニック表記上は、シンボルと区別するために、レジスタ名の前に必ず"%"を記述してください。
汎用レジスタ: %r0, %r1, %r2・・・%r15, または%R0, %R1, %R2・・・%R15
特殊レジスタ: PSR ..... %psr, または%PSR
SP ..... %sp, または%SP
ALR ..... %alr, または%ALR
AHR ..... %ahr, または%AHR

コード中のレジスタフィールド(rd, rs, sd, ss)には、レジスタ番号が入ります。
汎用レジスタ(rd, rs): R0 = 0b0000, R1 = 0b0001・・・R15 = 0b1111
特殊レジスタ(sd, ss): PSR = 0b0000, SP = 0b0001, ALR = 0b0010, AHR = 0b0011

4.1.2 即値

以下のシンボルは即値データを表します。

immX: Xビットの符号なし即値です。Xには即値のビット長を示す番号が入ります。
signX: Xビットの符号付き即値です。Xには即値のビット長を示す番号が入ります。また、最上位ビットは符号ビットとして扱われます。

4.1.3 メモリ

以下のシンボルはメモリの指定、メモリの内容を表します。

[%rb]: レジスタ間接アドレッシングの指定形式です。汎用レジスタrbの内容がアクセスされるメモリのベースアドレスとして使用されます。
[%rb]+: 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の指定形式です。汎用レジスタrbの内容がアクセスされるメモリのベースアドレスとして使用されます。命令実行の最後に、rbレジスタの内容がデータサイズに従ってインクリメントされます。
[%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の指定形式で、スタック領域のアドレス指定に使用します。SPの内容に即値immXを加えた内容がアクセスされるメモリのベースアドレスとなります。
B[rb]: 汎用レジスタrbで指定されるメモリアドレス、またはそのアドレスのバイトデータを表します。
B[rb+immX]: 汎用レジスタrbの内容に即値immXを加えて指定されるメモリアドレス、またはそのアドレスのバイトデータを表します。
B[sp+immX]: SPの内容に即値immXを加えて指定されるメモリアドレス、またはそのアドレスのバイトデータを表します。
H[rb]: 汎用レジスタrbの内容をベースアドレスとしたハーフワード(16ビット)領域、またはその領域のハーフワードデータを表します。ベースアドレス上のデータが下位バイトとして扱われます。

H[rb+immX]: 汎用レジスタrbの内容に即値immXを加えた値をベースアドレスとしたハーフワード(16ビット)領域、またはその領域のハーフワードデータを表します。ベースアドレス上のデータが下位バイトとして扱われます。
H[sp+immX]: SPの内容に即値immXを加えた値をベースアドレスとしたハーフワード(16ビット)領域、またはその領域のハーフワードデータを表します。ベースアドレス上のデータが下位バイトとして扱われます。
W[rb]: 汎用レジスタrbの内容をベースアドレスとしたワード(32ビット)領域、またはその領域のワードデータを表します。ベースアドレス上のデータが最下位バイトとして扱われます。
W[rb+immX]: 汎用レジスタrbの内容に即値immXを加えた値をベースアドレスとしたワード(32ビット)領域、またはその領域のワードデータを表します。ベースアドレス上のデータが最下位バイトとして扱われます。
W[sp]: SPの内容をベースアドレスとしたワード(32ビット)領域、またはその領域のワードデータを表します。ベースアドレス上のデータが最下位バイトとして扱われます。
W[sp+immX]: SPの内容に即値immXを加えた値をベースアドレスとしたワード(32ビット)領域、またはその領域のワードデータを表します。ベースアドレス上のデータが最下位バイトとして扱われます。

4.1.4 ビット・ビットフィールド

以下のシンボルはレジスタやメモリのシンボルとともに使用され、レジスタやメモリデータのビット番号やビットフィールドを表します。
(X): データ中のビットXを表します。(0)がLSBです。
(X:Y): ビットXからビットYまでのビットフィールドを表します。
{X, Y・・・}: AHR、ALRを64ビットのレジスタペアとして使用する場合の表記、またはビット構成を示すために使用します。{ }内の左側がデータの上位側となります。

4.1.5 フラグ

以下のシンボルはPSR内のフラグ、およびその変化の状態を表します。
IL[3:0]: 割り込みレベルフィールド
MO: MACオーバーフローフラグ
DS: 被除数符号フラグ
IE: 割り込みイネーブル
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
–: 命令の実行によって変化しないことを示します。
↔: 命令の実行によってセット(1)またはリセット(0)されることを示します。
0: 命令の実行によってリセット(0)されることを示します。

4.1.6 機能・その他

以下のシンボルは命令の機能説明に使用します。
←: 右側の内容が左側の項目にロード、セットされることを表します。
+: 加算
-: 減算
&: AND
|: OR
^: XOR
!: NOT
×: 乗算
÷: 除算

その他、複数のコードやニーモニックを一括して示す場合などに、次のシンボルを使用します。
*: 1または0の数字、あるいはa～zの英文字

4.2 命令コード体系

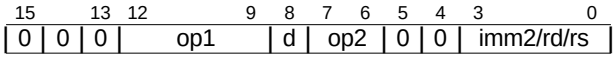
S1C33000命令セットでは、すべての命令が16ビット固定長です。
ビット構成は、機能およびアドレッシングモードにより大きく8種(class 0～7)に分類されています。このクラスをコードの上位3ビットが表します。

乗算・除算関連の命令は、オプションの乗除算回路を内蔵した機種に限り実行可能です。オプションを含まない機種では、以下の命令はnopとして処理されます。また、AHR、ALRレジスタも使用できません。

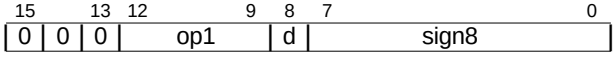
mlt.h	mltu.h	mlt.w	mltu.w	
div0s	div0u	div1	div2s	div3s
mac				
ld.w %rd, %ahr		ld.w %rd, %alr		
ld.w %ahr, %rs		ld.w %alr, %rs		

CLASS 0

このクラスにはオペランドを1つ持つ命令、および分岐命令が含まれます。



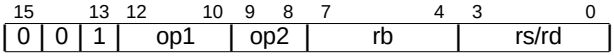
op1	op2	ニーモニック	機　能
0000	00	nop	ノーオペレーション
0000	01	slp	SLEEPモード
0000	10	halt	HALTモード
0000	11	reserved	
0001	00	pushn %rs	汎用レジスタのプッシュ
0001	01	popn %rd	汎用レジスタのポップ
0001	1*	reserved	
0010	00	brk	デバッグ用ソフトウェア例外
0010	01	retd	デバッグルーチンからのリターン
0010	10	int imm2	ソフトウェア例外
0010	11	reti	トラップ処理ルーチンからのリターン
0011	00	call %rb	サブルーチンコール
0011	01	ret	サブルーチンからのリターン
0011	10	jp %rb	無条件ジャンプ
0011	11	reserved	



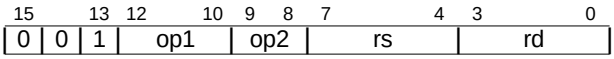
op1	ニーモニック	機　能
0100	jrgt sign8	PC相対条件ジャンプ　分岐条件 = !Z & !(N ^ V)
0101	jrge sign8	PC相対条件ジャンプ　分岐条件 = !(N ^ V)
0110	jrlt sign8	PC相対条件ジャンプ　分岐条件 = N ^ V
0111	jrle sign8	PC相対条件ジャンプ　分岐条件 = Z | (N ^ V)
1000	jrugt sign8	PC相対条件ジャンプ　分岐条件 = !Z & !C
1001	jruge sign8	PC相対条件ジャンプ　分岐条件 = !C
1010	jrult sign8	PC相対条件ジャンプ　分岐条件 = C
1011	jrule sign8	PC相対条件ジャンプ　分岐条件 = Z | C
1100	jreq sign8	PC相対条件ジャンプ　分岐条件 = Z
1101	jrne sign8	PC相対条件ジャンプ　分岐条件 = !Z
1110	call sign8	PC相対サブルーチンコール
1111	jp sign8	PC相対無条件ジャンプ

CLASS 1

このクラスには汎用レジスタとメモリ間のデータ転送命令、および汎用レジスタ間の演算命令が含まれます。



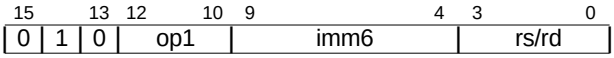
op1	op2	ニーモニック	機　能
000	00	ld.b %rd,[%rb]	メモリから汎用レジスタへのバイトデータ転送(符号拡張)
001	00	ld.ub %rd,[%rb]	メモリから汎用レジスタへのバイトデータ転送(ゼロ拡張)
010	00	ld.h %rd,[%rb]	メモリから汎用レジスタへのハーフワードデータ転送(符号拡張)
011	00	ld.uh %rd,[%rb]	メモリから汎用レジスタへのハーフワードデータ転送(ゼロ拡張)
100	00	ld.w %rd,[%rb]	メモリから汎用レジスタへのワードデータ転送
101	00	ld.b [%rb],%rs	汎用レジスタからメモリへのバイトデータ転送
110	00	ld.h [%rb],%rs	汎用レジスタからメモリへのハーフワードデータ転送
111	00	ld.w [%rb],%rs	汎用レジスタからメモリへのワードデータ転送
000	01	ld.b %rd,[%rb]+	メモリから汎用レジスタへのバイトデータ転送(符号拡張)
001	01	ld.ub %rd,[%rb]+	メモリから汎用レジスタへのバイトデータ転送(ゼロ拡張)
010	01	ld.h %rd,[%rb]+	メモリから汎用レジスタへのハーフワードデータ転送(符号拡張)
011	01	ld.uh %rd,[%rb]+	メモリから汎用レジスタへのハーフワードデータ転送(ゼロ拡張)
100	01	ld.w %rd,[%rb]+	メモリから汎用レジスタへのワードデータ転送
101	01	ld.b [%rb]+,%rs	汎用レジスタからメモリへのバイトデータ転送
110	01	ld.h [%rb]+,%rs	汎用レジスタからメモリへのハーフワードデータ転送
111	01	ld.w [%rb]+,%rs	汎用レジスタからメモリへのワードデータ転送



op1	op2	ニーモニック	機　能
000	10	add %rd,%rs	汎用レジスタ間加算
001	10	sub %rd,%rs	汎用レジスタ間減算
010	10	cmp %rd,%rs	汎用レジスタ間比較
011	10	ld.w %rd,%rs	汎用レジスタ間データ転送
100	10	and %rd,%rs	汎用レジスタ間論理積
101	10	or %rd,%rs	汎用レジスタ間論理和
110	10	xor %rd,%rs	汎用レジスタ間排他的論理和
111	10	not %rd,%rs	汎用レジスタ論理否定
***	11	reserved	

CLASS 2

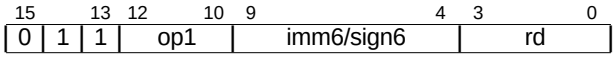
このクラスにはSPによるディスプレースメント付きレジスタ間接アドレッシングのデータ転送命令が含まれます。



op1	ニーモニック	機　能
000	ld.b %rd,[%sp+imm6]	スタックから汎用レジスタへのバイトデータ転送(符号拡張)
001	ld.ub %rd,[%sp+imm6]	スタックから汎用レジスタへのバイトデータ転送(ゼロ拡張)
010	ld.h %rd,[%sp+imm6]	スタックから汎用レジスタへのハーフワードデータ転送(符号拡張)
011	ld.uh %rd,[%sp+imm6]	スタックから汎用レジスタへのハーフワードデータ転送(ゼロ拡張)
100	ld.w %rd,[%sp+imm6]	スタックから汎用レジスタへのワードデータ転送
101	ld.b [%sp+imm6],%rs	汎用レジスタからスタックへのバイトデータ転送
110	ld.h [%sp+imm6],%rs	汎用レジスタからスタックへのハーフワードデータ転送
111	ld.w [%sp+imm6],%rs	汎用レジスタからスタックへのワードデータ転送

CLASS 3

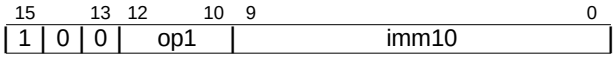
このクラスには6ビット即値を使用するデータ転送命令および演算命令が含まれます。



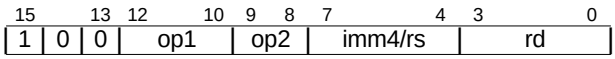
op1	ニーモニック	機　能
000	add %rd,imm6	汎用レジスタへの即値加算
001	sub %rd,imm6	汎用レジスタからの即値減算
010	cmp %rd,sign6	汎用レジスタ～即値間比較
011	ld.w %rd,sign6	汎用レジスタへの即値データ転送
100	and %rd,sign6	汎用レジスタ～即値間論理積
101	or %rd,sign6	汎用レジスタ～即値間論理和
110	xor %rd,sign6	汎用レジスタ～即値間排他的論理和
111	not %rd,sign6	即値論理否定

CLASS 4

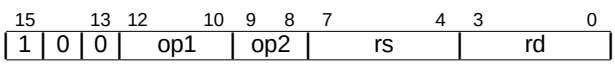
このクラスにはSPに対する演算命令、シフト/ローテート命令、除算命令等が含まれます。



op1	ニーモニック	機　能
000	add %sp,imm10	SPへの即値加算
001	sub %sp,imm10	SPからの即値減算



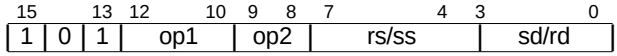
op1	op2	ニーモニック	機　能
010	00	srl %rd,imm4	右方向論理シフト(imm4により最大8ビット)
011	00	sll %rd,imm4	左方向論理シフト(imm4により最大8ビット)
100	00	sra %rd,imm4	右方向算術シフト(imm4により最大8ビット)
101	00	sla %rd,imm4	左方向算術シフト(imm4により最大8ビット)
110	00	rr %rd,imm4	右方向ローテート(imm4により最大8ビット)
111	00	rl %rd,imm4	左方向ローテート(imm4により最大8ビット)
010	01	srl %rd,%rs	右方向論理シフト(rsにより最大8ビット)
011	01	sll %rd,%rs	左方向論理シフト(rsにより最大8ビット)
100	01	sra %rd,%rs	右方向算術シフト(rsにより最大8ビット)
101	01	sla %rd,%rs	左方向算術シフト(rsにより最大8ビット)
110	01	rr %rd,%rs	右方向ローテート(rsにより最大8ビット)
111	01	rl %rd,%rs	左方向ローテート(rsにより最大8ビット)



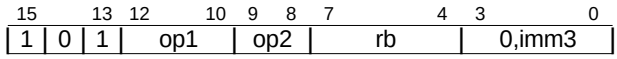
op1	op2	ニーモニック	機　能
010	10	scan0 %rd,%rs	"0"ビットサーチ
011	10	scan1 %rd,%rs	"1"ビットサーチ
100	10	swap %rd,%rs	バイト単位のスワップ
101	10	mirror %rd,%rs	バイト単位のビット順列変更
11*	10	reserved	
010	11	div0s %rs	符号付き除算第1ステップ
011	11	div0u %rs	符号なし除算第1ステップ
100	11	div1 %rs	ステップ除算
101	11	div2s %rs	符号付き除算のデータ補正1
110	11	div3s	符号付き除算のデータ補正2
111	11	reserved	

CLASS 5

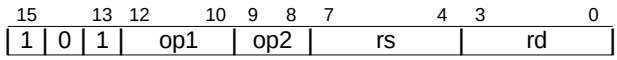
このクラスには汎用レジスタと特殊レジスタ間および汎用レジスタ間のデータ転送命令、ビット操作命令、乗算命令、積和演算等が含まれます。



op1	op2	ニーモニック	機　能
000	00	ld.w %sd,%rs	汎用レジスタから特殊レジスタへのワードデータ転送
001	00	ld.w %rd,%ss	特殊レジスタから汎用レジスタへのワードデータ転送



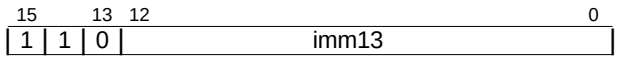
op1	op2	ニーモニック	機　能
010	00	btst [%rb],imm3	メモリデータのビットテスト
011	00	bclr [%rb],imm3	メモリデータのビットクリア
100	00	bset [%rb],imm3	メモリデータのビットセット
101	00	bnot [%rb],imm3	メモリデータのビット反転



op1	op2	ニーモニック	機　能
110	00	adc %rd,%rs	汎用レジスタ間キャリー付き加算
111	00	sbc %rd,%rs	汎用レジスタ間ボロー付き減算
000	01	ld.b %rd,%rs	汎用レジスタ間バイトデータ転送(符号拡張)
001	01	ld.ub %rd,%rs	汎用レジスタ間バイトデータ転送(ゼロ拡張)
010	01	ld.h %rd,%rs	汎用レジスタ間ハーフワードデータ転送(符号拡張)
011	01	ld.uh %rd,%rs	汎用レジスタ間ハーフワードデータ転送(ゼロ拡張)
1**	01	reserved	
000	10	mlt.h %rd,%rs	符号付き16ビット乗算
001	10	mltu.h %rd,%rs	符号なし16ビット乗算
010	10	mlt.w %rd,%rs	符号付き32ビット乗算
011	10	mltu.w %rd,%rs	符号なし32ビット乗算
100	10	mac %rs	積和演算
101	10	reserved	
11*	10	reserved	
***	11	reserved	

CLASS 6

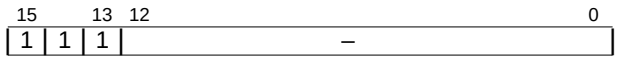
即値拡張命令がこのクラスに設定されています。



ニーモニック	機　能
ext imm13	即値拡張

CLASS 7

このクラスは将来の拡張用に予約されています。



4.3 個別命令リファレンス

次ページより全命令をアルファベット順に解説します。

説明には以下の項目を含んでいます。

機　能　各命令の機能を示します。
「標準」は命令を単独に実行した場合の機能です。
「拡張1」は命令の直前にext命令を1個使用して即値を拡張した場合の機能です。
「拡張2」は命令の直前にext命令を2個使用して即値を拡張した場合の機能です。
「拡張」が不可となっている命令は直前のext命令が無効となります。

コード　各命令のコードを示します。

フラグ　命令実行後の各フラグの状態を示します。

モード　アドレッシングモードを示します。Srcはソース、Dstはディスティネーションのアドレッシングモードです。

ＣＬＫ　各命令の実行に要するサイクル数を示します。記載のサイクル数は内蔵ROMの命令を実行し、データメモリアクセスは内蔵RAMのみを対象とした場合のものです。
外部メモリを使用した場合やインターロックによる遅延等、その他の条件における実行サイクル数については、"3.2.2 命令の実行サイクル数"を参照してください。

説　明　機能を解説します。

例　アセンブラレベルの使用例です。

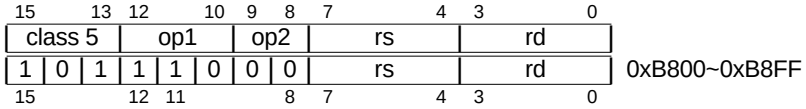
注　意　使用上の注意事項等を示します。

adc %rd, %rs

機 能 **キャリー付き加算**

標準： **rd ← rd + rs + C**
拡張1: **不可**
拡張2: **不可**

コード



0xB800~0xB8FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	↔	↔	↔	↔

モード **Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

CLK 1サイクル

説 明 **(1)標準**

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。

(2)ディレイド命令

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命令として実行されます。

例

```
adc   %r0,%r1        ; r0 = r0 + r1 + C
```

64ビットデータの加算
データ1 = {r2, r1} データ2 = {r4, r3} 加算結果 = {r2, r1}

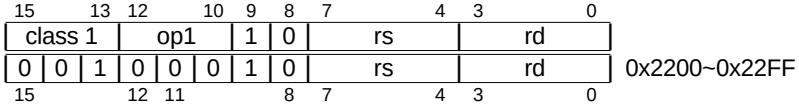
```
add   %r1,%r3        ; 下位ワードの加算
adc   %r2,%r4        ; 上位ワードの加算
```

add %rd, %rs

機 能 **加算**

標準 : **rd ← rd + rs**
拡張1: **rd ← rs + imm13**
拡張2: **rd ← rs + imm26**

コード



フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	↔	↔	↔	↔

モード **Src:** レジスタ直接(%rs = %r0~%r15)
Dst: レジスタ直接(%rd = %r0~%r15)

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。

(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 imm13 ; = imm26(25:13)
ext imm13' ; = imm26(12:0)
add %rd, %rs ; rd ← rs + imm26

rsレジスタの内容に26ビット即値imm26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。

(4)ディレイド命令

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

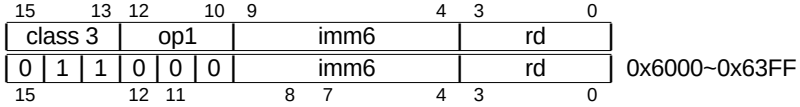
```
add  %r0,%r0       ; r0 = r0 + r0

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

add %rd, imm6

機 能 **加算**

標準： **rd ← rd + imm6**
拡張1: **rd ← rd + imm19**
拡張2: **rd ← rd + imm32**

コード



0x6000~0x63FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	↔	↔	↔	↔

モード **Src:** 即値（符号なし）
Dst: レジスタ直接（%rd = %r0~%r15）

CLK 1サイクル

説 明 **(1)標準**

```
add   %rd, imm6    ; rd ← rd + imm6
```

6ビット即値imm6をゼロ拡張してrdレジスタに加えます。

(2)拡張1

```
ext   imm13        ; = imm19(18:6)
add   %rd, imm6    ; rd ← rd + imm19, imm6 = imm19(5:0)
```

ext命令により拡張した19ビット即値imm19をゼロ拡張してrdレジスタに加えます。

(3)拡張2

```
ext   imm13        ; = imm32(31:19)
ext   imm13'       ; = imm32(18:6)
add   %rd, imm6    ; rd ← rd + imm32, imm6 = imm32(5:0)
```

ext命令により拡張した32ビット即値imm32をrdレジスタに加えます。

(4)ディレイド命令

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

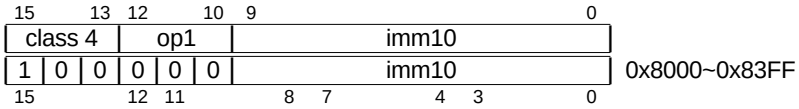
```
add   %r0,0x3f      ; r0 = r0 + 0x3f

ext   0x1fff
ext   0x1fff
add   %r1,0x3f      ; r1 = r1 + 0xffffffff
```

add %sp, imm10

機 能 **加算**

標準 : **sp ← sp + imm10 x 4**
拡張1: **不可**
拡張2: **不可**

コード



0x8000~0x83FF

フラグ

IL(3:0)	MO	DS	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**

10ビット即値imm10を4倍し、ゼロ拡張してスタックポインタSPに加えます。

(2)ディレイド命令

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命令として実行されます。

例

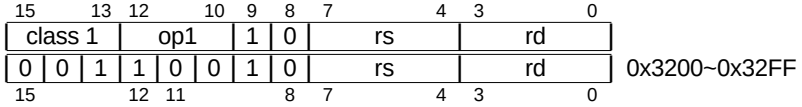
```
add  %sp,0x100    ; sp = sp + 0x400
```

and %rd, %rs

機 能 **論理積**

標準： **rd ← rd & rs**
拡張1: **rd ← rs & imm13**
拡張2: **rd ← rs & imm26**

コード



0x3200~0x32FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

モード **Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

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。

(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。rsレジスタの内容は変更されません。

(3)拡張2

```
ext    imm13         ; = imm26(25:13)
ext    imm13'        ; = imm26(12:0)
and    %rd, %rs      ; rd ← rs & imm26
```

rsレジスタの内容とゼロ拡張した26ビット即値imm26の論理積をとり、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)ディレイド命令

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

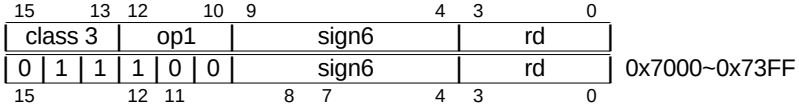
```
and   %r0,%r0       ; r0 = r0 & r0

ext   0x1
ext   0x1fff
and   %r1,%r2       ; r1 = r2 & 0x00003fff
```

and %rd, sign6

機 能 **論理積**

標準 : **rd ← rd & sign6**
拡張1: **rd ← rd & sign19**
拡張2: **rd ← rd & sign32**

コード



フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

モード **Src:** 即値(符号付き)
Dst: レジスタ直接(%rd = %r0~%r15)

ＣＬＫ 1サイクル

説 明 **(1)標準**

and %rd, sign6 ; rd ← rd & sign6

rdレジスタの内容と符号拡張した6ビット即値sign6の論理積をとり、結果をrdレジスタにロードします。

(2)拡張1

ext imm13 ; = sign19(18:6)
and %rd, sign6 ; rd ← rd & sign19, sign6 = sign19(5:0)

rdレジスタの内容と符号拡張した19ビット即値sign19の論理積をとり、結果をrdレジスタにロードします。

(3)拡張2

ext imm13 ; = sign32(31:19)
ext imm13' ; = sign32(18:6)
and %rd, sign6 ; rd ← rd & sign32, sign6 = sign32(5:0)

rdレジスタの内容とext命令により拡張した32ビット即値sign32の論理積をとり、結果をrdレジスタにロードします。

(4)ディレイド命令

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

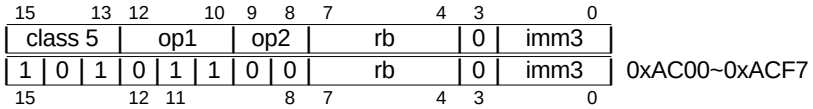
```
and  %r0,0x3e      ; r0 = r0 & 0xfffffffe

ext  0x7ff
and  %r1,0x3f      ; r1 = r1 & 0x0001ffff
```

bclr [%rb], imm3

機 能 **ビットクリア**

標準： **B[rb](imm3) ← 0**
拡張1: **B[rb + imm13](imm3) ← 0**
拡張2: **B[rb + imm26](imm3) ← 0**

コード



0xAC00~0xACF7

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード **Src:** 即値（符号なし）
Dst: レジスタ間接（%rb = %r0~%r15）

ＣＬＫ 3サイクル

説 明 **(1)標準**

bclr [%rb], imm3 ; B[rb](imm3) ← 0

指定メモリのデータビットをクリアします。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。3ビット即値imm3はクリアするビット番号（バイトデータのビット7～0の1つ）を指定します。

(2)拡張1

ext imm13
bclr [%rb], imm3 ; B[rb + imm13](imm3) ← 0

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を加えたアドレス上の、imm3で指定されたデータビットをクリアします。rbレジスタの内容は変更されません。

(3)拡張2

ext imm13 ; = imm26(25:13)
ext imm13' ; = imm26(12:0)
bclr [%rb], imm3 ; B[rb + imm26](imm3) ← 0

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26ビット即値imm26を加えたアドレス上の、imm3で指定されたデータビットをクリアします。rbレジスタの内容は変更されません。

例

```
ld.w %r0,[%sp+0x10]  ; アクセスするメモリアドレスをR0レジスタに設定
bclr [%r0],0x0       ; 指定アドレス上のデータのビット0をクリア

ext  0x1
bclr [%r0],0x7       ; 上記アドレスの次のアドレスのデータビット7をクリア
```

bnot [%rb], imm3

機　能　**ビット反転**

標準 :　**B[rb](imm3)** ← **!B[rb](imm3)**
拡張1:　**B[rb + imm13](imm3)** ← **!B[rb + imm13](imm3)**
拡張2:　**B[rb + imm26](imm3)** ← **!B[rb + imm26](imm3)**

コード

15 13	12 10	9 8	7 4	3	0
class 5	op1	op2	rb	0	imm3

15	14	13	12	11	10	9	8	7 4	3	0
1	0	1	1	0	1	0	0	rb	0	imm3

0xB400~0xB4F7

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** 即値（符号なし）
Dst: レジスタ間接（%rb = %r0~%r15）

ＣＬＫ　3サイクル

説　明　**(1)標準**

bnot　[%rb], imm3　; B[rb](imm3) ← **!B[rb](imm3)**
指定メモリのデータビットを反転（1↔0）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。3ビット即値imm3は反転するビット番号（バイトデータのビット7～0の1つ）を指定します。

(2)拡張1

ext　　imm13
bnot　[%rb], imm3　; B[rb + imm13](imm3) ← **!B[rb + imm13](imm3)**
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を加えたアドレス上の、imm3で指定されたデータビットを反転します。rbレジスタの内容は変更されません。

(3)拡張2

ext　　imm13　　; = imm26(25:13)
ext　　imm13'　　; = imm26(12:0)
bnot　[%rb], imm3　; B[rb + imm26](imm3) ← **!B[rb + imm26](imm3)**
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26ビット即値imm26を加えたアドレス上の、imm3で指定されたデータビットを反転します。rbレジスタの内容は変更されません。

例

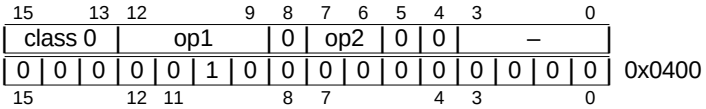
```
ld.w %r0,[%sp+0x10]   ; アクセスするメモリアドレスをR0レジスタに設定
bnot [%r0],0x0        ; 指定アドレス上のデータのビット0を反転

ext  0x1
bnot [%r0],0x7        ; 上記アドレスの次のアドレスのデータビット7を反転
```

brk

機　能　**デバッグ用ソフトウェア例外**

標準：　**W[0x8(or 0x60008)] ← pc + 2, W[0xC(or 0x6000C)] ← r0, pc ← W[0x0(or 0x60000)]**
拡張1:　**不可**
拡張2:　**不可**

コード



フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	0	–	–	–	–

ＣＬＫ　10サイクル

説　明　デバッグ処理ルーチンを呼び出すソフトウェア例外です。
次の命令のアドレスと汎用レジスタR0をデバッグ用スタックにセーブ後、デバッグ用ベクタアドレス0x0000000(もしくは0x0060000)からデバッグルーチンへのベクタを読み出して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、CPU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ICE制御ソフト専用です。一般のプログラムでは使用しません。

例

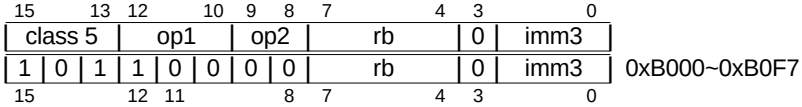
```
brk                     ; デバッグ処理ルーチンを実行
```

bset [%rb], imm3

機　能　**ビットセット**

標準 :　**B[rb](imm3) ← 1**
拡張1:　**B[rb + imm13](imm3) ← 1**
拡張2:　**B[rb + imm26](imm3) ← 1**

コード



0xB000~0xB0F7

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** 即値（符号なし）
Dst: レジスタ間接（%rb = %r0~%r15）

ＣＬＫ　3サイクル

説　明　**(1)標準**

bset　[%rb], imm3　; B[rb](imm3) ← 1

指定メモリのデータビットをセットします。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。3ビット即値imm3はセットするビット番号（バイトデータのビット7～0の1つ）を指定します。

(2)拡張1

ext　　imm13
bset　[%rb], imm3　; B[rb + imm13](imm3) ← 1

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を加えたアドレス上の、imm3で指定されたデータビットをセットします。rbレジスタの内容は変更されません。

(3)拡張2

ext　　imm13　　; = imm26(25:13)
ext　　imm13'　　; = imm26(12:0)
bset　[%rb], imm3　; B[rb + imm26](imm3) ← 1

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26ビット即値imm26を加えたアドレス上の、imm3で指定されたデータビットをセットします。rbレジスタの内容は変更されません。

例

```
ld.w %r0,[%sp+0x10]  ; アクセスするメモリアドレスをR0レジスタに設定
bset [%r0],0x0       ; 指定アドレス上のデータのビット0をセット

ext  0x1
bset [%r0],0x7       ; 上記アドレスの次のアドレスのデータビット7をセット
```

btst [%rb], imm3

機 能 **ビットテスト**

標準： **Z flag ← 1 if B[rb](imm3) = 0 else Z flag ← 0**
拡張1: **Z flag ← 1 if B[rb + imm13](imm3) = 0 else Z flag ← 0**
拡張2: **Z flag ← 1 if B[rb + imm26](imm3) = 0 else Z flag ← 0**

コード

15–13	12–10	9	8	7–4	3	3–0	
class 5	op1		op2	rb	0	imm3	
1 0 1	0 1 0	0	0	rb	0	imm3	0xA800~0xA8F7
15–12	11–8			7–4	3	3–0	

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	–

モード **Src:** 即値(符号なし)
Dst: レジスタ間接(%rb = %r0~%r15)

CLK 3サイクル

説 明 **(1)標準**

btst [%rb], imm3 ; Z flag ← 1 if B[rb](imm3) = 0 else Z flag ← 0
指定メモリのデータビットをテストし、それが0ならばZ(ゼロ)フラグをセットします。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。3ビット即値imm3はテストするビット番号(バイトデータのビット7～0の1つ)を指定します。

(2)拡張1

ext imm13
btst [%rb], imm3 ; Z flag ← 1 if B[rb + imm13](imm3) = 0 else Z flag ← 0
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を加えたアドレス上の、imm3で指定されたデータビットをテストします。rbレジスタの内容は変更されません。

(3)拡張2

ext imm13 ; = imm26(25:13)
ext imm13' ; = imm26(12:0)
btst [%rb], imm3 ; Z flag ← 1 if B[rb + imm26](imm3) = 0 else Z flag ← 0
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26ビット即値imm26を加えたアドレス上の、imm3で指定されたデータビットをテストします。rbレジスタの内容は変更されません。

例

```
ld.w %r0,[%sp+0x10]  ; アクセスするメモリアドレスをR0レジスタに設定
btst [%r0],0x7       ; 指定アドレス上のデータのビット7をテスト
jreq POSITIVE        ; ビットが0ならばジャンプ
```

call %rb / call.d %rb

機 能 **サブルーチンコール**

標準 : **sp ← sp - 4, W[sp] ← pc + 2, 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
class 0			op1				d	op2		0	0	rb			
0	0	0	0	0	1	1	d	0	0	0	0	rb			

0x0600~0x060F, 0x070F~0x070F

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード レジスタ直接(%rb = %r0~%r15)

ＣＬＫ **call:** 3サイクル
call.d: 2サイクル

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、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、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=1)

call.d %rb

call.d命令では命令コード中のdビットがセットされ、次の命令がディレイド命令となります。ディレイド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命令の間はトラップがマスクされ、割り込みや例外は発生しません。

例

```
call %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
```

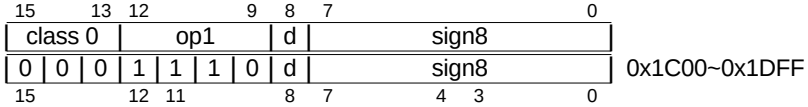
注 意 call.d命令(ディレイド分岐)を使用する場合、次の命令はディレイド命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

call sign8 / call.d sign8

機 能 **サブルーチンコール**

標準： **sp ← sp - 4, W[sp] ← pc + 2, pc ← pc + sign8 x 2**
拡張1: **sp ← sp - 4, W[sp] ← pc + 2, pc ← pc + sign22**
拡張2: **sp ← sp - 4, W[sp] ← pc + 2, pc ← pc + sign32**

コード



0x1C00~0x1DFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード 符号付きPC相対

ＣＬＫ **call:** 3サイクル
call.d: 2サイクル

説 明 **(1)標準**

```
call    sign8          ; = "call  sign9", sign8 = sign9(8:1), sign9(0) = 0
```

次の命令のアドレスをスタックにセーブ後、PCに符号付き8ビット即値sign8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。sign8は16ビット単位のハーフ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。sign8(×2)による分岐可能範囲はPC-0x100～PC+0xFEです。

(2)拡張1

```
ext     imm13          ; = sign22(21:9)
call    sign8          ; = "call  sign22", sign8 = sign22(8:1), sign22(0) = 0
```

ext命令の13ビット即値imm13が符号拡張され、PCに加算するディスプレースメントが符号付き22ビットとなります。sign22による分岐可能範囲はPC-0x200000～PC+0x1FFFFEです。

(3)拡張2

```
ext     imm13          ; imm13(12:3)= sign32(31:22)
ext     imm13'         ; = sign(21:9)
call    sign8          ; = "call  sign32", sign9 = sign32(8:1), sign32(0) = 0
```

ext命令の2つの13ビット即値imm13、imm13'が符号拡張され、PCに加算するディスプレースメントが符号付き32ビットとなります。これにより、S1C33000の全アドレス空間をカバーします。

(4)ディレイド分岐(dビット=1)

```
call.d  sign8
```

call.d命令では命令コード中のdビットがセットされ、次の命令がディレイド命令となります。ディレイド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命令の間はトラップがマスクされ、割り込みや例外は発生しません。

例

```
ext   0x1fff
call  0x0      ; PC-0x200番地から始まるサブルーチンをコール
```

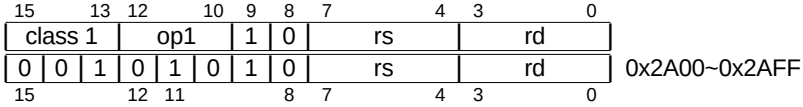
注 意 call.d命令(ディレイド分岐)を使用する場合、次の命令はディレイド命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

cmp %rd, %rs

機 能 **比較**

標準 : **rd - rs**
拡張1: **rs - imm13**
拡張2: **rs - imm26**

コード



フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	↔	↔	↔	↔

モード **Src:** レジスタ直接(%rs = %r0~%r15)
Dst: レジスタ直接(%rd = %r0~%r15)

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。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。rdおよびrsレジスタは変更されません。

(3)拡張2

ext imm13 ; = imm26(25:13)
ext imm13' ; = imm26(12:0)
cmp %rd, %rs ; rs - imm26

rsレジスタの内容から26ビット即値imm26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。rdおよびrsレジスタは変更されません。

(4)ディレイド命令

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

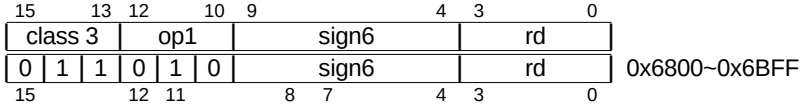
```
cmp  %r0,%r1       ; r0 - r1、結果によりフラグを変更

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

cmp %rd, sign6

機 能 **比較**

標準： **rd - sign6**
拡張1: **rd - sign19**
拡張2: **rd - sign32**

コード



フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	↔	↔	↔	↔

モード **Src:** 即値（符号付き）
Dst: レジスタ直接（%rd = %r0~%r15）

ＣＬＫ 1サイクル

説 明 **(1)標準**

cmp %rd, sign6 ; rd - sign6

rdレジスタの内容から6ビット即値sign6を符号拡張して減算し、結果によりフラグ(C、V、Z、N)をセット/リセットします。rdレジスタは変更されません。

(2)拡張1

ext imm13 ; = sign19(18:6)
cmp %rd, sign6 ; rd - sign19, sign6 = sign19(5:0)

rdレジスタの内容から19ビット即値sign19を符号拡張して減算し、結果によりフラグ(C、V、Z、N)をセット/リセットします。rdレジスタは変更されません。

(3)拡張2

ext imm13 ; = sign32(31:19)
ext imm13' ; = sign32(18:6)
cmp %rd, sign6 ; rd - sign32, imm6 = sign32(5:0)

rdレジスタの内容からext命令により拡張した符号付き32ビット即値sign32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。rdレジスタは変更されません。

(4)ディレイド命令

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

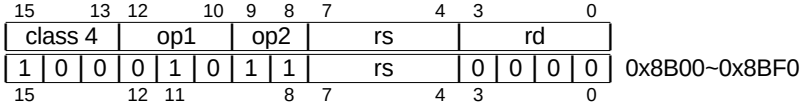
```
cmp   %r0,0x3f      ; r0 - 0x3f、結果によりフラグを変更

ext   0x1fff
ext   0x1fff
cmp   %r1,0x3f      ; r1 - 0xffffffff、結果によりフラグを変更
```

div0s %rs （オプション）

機　能　**符号付き除算第1ステップ**

標準 :　**除算実行用の初期化処理**
拡張1:　**不可**
拡張2:　**不可**

コード



フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	↔	–	–	–	–	↔

モード　レジスタ直接（%rs = %r0~%r15）

ＣＬＫ　1サイクル

説　明　符号付き除算を行う場合は、ALRに被除数、rsレジスタに除数を設定し、最初にdiv0s命令を実行します。div0s命令は以下の初期化処理を行います。

1）ALRの被除数を{AHR, ALR}の64ビットに符号拡張します。

2）PSRのDSフラグに、被除数の符号ビット（ALRの最上位ビット）をセットします。

3）PSRのNフラグに、除数の符号ビット（rsレジスタの最上位ビット）をセットします。

このため、ALRとrsレジスタに用意しておく被除数と除数は32ビットに符号拡張されていることが必要です。
div0s命令実行後はdiv1命令による除算を実行します。符号付き除算では、その後にdiv2s命令およびdiv3s命令による補正が必要です。

例　**符号付き除算例（32ビット÷32ビット）**

R0レジスタに被除数、R1レジスタに除数が設定されている場合

```
ld.w    %alr,%r0   ; ALRに被除数を設定
div0s   %r1        ; 符号付き除算の初期化ステップ
div1    %r1        ; div1を32回実行
 :       :
div1    %r1
div2s   %r1        ; 補正命令1
div3s              ; 補正命令2
```

上記命令実行後、商がALRから、余りがAHRから得られます。

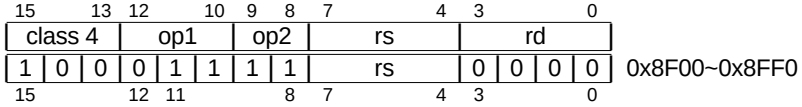
注　意　rsレジスタに0を設定してdiv0s命令を実行すると、ゼロ除算例外が発生します。
被除数、除数とも演算可能なデータサイズは32ビットまでです。
本命令はオプションの乗除算回路を内蔵した機種に限り実行可能です。オプションを含まない機種ではnopとして処理されます。

div0u %rs （オプション）

機 能 **符号なし除算第1ステップ**

標準： **除算実行用の初期化処理**
拡張1: **不可**
拡張2: **不可**

コード



0x8F00~0x8FF0

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	0	–	–	–	–	0

モード レジスタ直接（%rs = %r0~%r15）

CLK 1サイクル

説 明 符号なし除算を行う場合は、ALRに被除数、rsレジスタに除数を設定し、最初にdiv0u命令を実行します。div0u命令は以下の初期化処理を行います。

1）AHRをオール0にクリアします。

2）PSRのDSフラグをリセット(0)します。

3）PSRのNフラグをリセット(0)します。

div0u命令実行後はdiv1命令による除算を実行します。符号なし除算では、div1命令によって除算結果が得られ、その後の補正は不要です。

例 符号なし除算例(32ビット÷32ビット)

R0レジスタに被除数、R1レジスタに除数が設定されている場合

```
ld.w    %alr,%r0   ; ALRに被除数を設定
div0u   %r1        ; 符号なし除算の初期化ステップ
div1    %r1        ; div1を32回実行
 :       :
div1    %r1
```

上記命令実行後、商がALRから、余りがAHRから得られます。

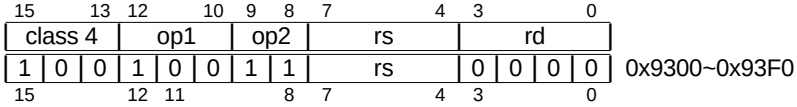
注 意 rsレジスタに0を設定してdiv0u命令を実行すると、ゼロ除算例外が発生します。
被除数、除数とも演算可能なデータサイズは32ビットまでです。
本命令はオプションの乗除算回路を内蔵した機種に限り実行可能です。オプションを含まない機種ではnopとして処理されます。

div1 %rs （オプション）

機　能　**除算**

標準：　**除算実行**
拡張1:　**不可**
拡張2:　**不可**

コード



0x9300~0x93F0

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　レジスタ直接（%rs = %r0~%r15）

ＣＬＫ　1サイクル

説　明　div1は符号付き除算、符号なし除算に共通なステップ除算実行命令です。本命令は、div0s命令（符号付き除算）またはdiv0u（符号なし除算）の初期化処理を実行後、被除数のデータサイズに従った回数を実行する必要があります。たとえば32ビット÷32ビットの場合は32回、16ビット÷16ビットの場合は16回実行します。
div1命令は1ステップで以下の処理を行います。

1）{AHR, ALR}の64ビットを左（上位側）に1ビットシフト（ALR(0) = 0）
2）AHRとrsを加算またはAHRからrsを減算し、結果によりAHR、ALRを再設定
加減算はAHRの内容にDSフラグを符号ビットとして付加した33ビットと、rsレジスタの内容にNフラグを符号ビットとして付加した33ビットデータによって行います。
この処理はPSRのDSフラグおよびNフラグによって以下のように異なります。なお、演算結果の33ビット目の値をtmp(32)として説明します。

DS = 0（被除数: 正）、N = 0（除数: 正）の場合
　2-1）　tmp = {0, AHR} - {0, rs}を実行
　2-2）　tmp(32) = 0の場合: AHR = tmp(31:0)、ALR(0) = 1として終了
　　tmp(32) = 1の場合: AHR、ALRをそのままに終了

DS = 1（被除数: 負）、N = 0（除数: 正）の場合
　2-1）　tmp = {1, AHR} + {0, rs}を実行
　2-2）　tmp(32) = 1の場合: AHR = tmp(31:0)、ALR(0) = 1として終了
　　tmp(32) = 0の場合: AHR、ALRをそのままに終了

DS = 0（被除数: 正）、N = 1（除数: 負）の場合
　2-1）　tmp = {0, AHR} + {1, rs}を実行
　2-2）　tmp(32) = 0の場合: AHR = tmp(31:0)、ALR(0) = 1として終了
　　tmp(32) = 1の場合: AHR、ALRをそのままに終了

DS = 1（被除数: 負）、N = 1（除数: 負）の場合
　2-1）　tmp = {1, AHR} - {1, rs}を実行
　2-2）　tmp(32) = 1の場合: AHR = tmp(31:0)、ALR(0) = 1として終了
　　tmp(32) = 0の場合: AHR、ALRをそのままに終了

符号なし除算の場合は、div1命令を必要数実行することにより結果が次のレジスタから得られます。
　　符号なし除算結果: ALR = 商　AHR = 余り

符号付き除算では、この後にdiv2s命令およびdiv3s命令による補正が必要です。

例 **符号なし除算例(32ビット÷32ビット)**

R0レジスタに被除数、R1レジスタに除数が設定されている場合

```
ld.w    %alr,%r0   ; ALRに被除数を設定
div0u   %r1        ; 符号なし除算の初期化ステップ
div1    %r1        ; div1を32回実行
 :       :
div1    %r1
```

上記命令実行後、商がALRから、余りがAHRから得られます。

符号付き除算例(32ビット÷32ビット)

R0レジスタに被除数、R1レジスタに除数が設定されている場合

```
ld.w    %alr,%r0   ; ALRに被除数を設定
div0s   %r1
div1    %r1        ; div1を32回実行
 :       :
div1    %r1
div2s   %r1        ; 補正命令1
div3s              ; 補正命令2
```

上記命令実行後、商がALRから、余りがAHRから得られます。

注 意 本命令はオプションの乗除算回路を内蔵した機種に限り実行可能です。オプションを含まない機種ではnopとして処理されます。

div2s %rs （オプション）

機　能　**符号付き除算結果補正1**

標準 :　**符号付き除算実行結果の補正処理**
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	
class 4			op1			op2		rs				rd				
1	0	0	1	0	1	1	1	rs				0	0	0	0	0x9700~0x97F0

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　レジスタ直接（%rs = %r0~%r15）

ＣＬＫ　1サイクル

説　明　div2s命令は符号付き除算結果の補正を行います。符号なし除算の場合はdiv2s命令を実行する必要はありません。

被除数が負の数の場合に除算のステップ（div1命令の実行）で演算結果がゼロになると、全ステップ終了後の演算結果が、除数と同じ余り（AHR）、および実際の値よりも絶対値で1少ない商（ALR）となる可能性があります。div2s命令はこの結果を補正します。
div2s命令の動作は次のとおりです。

DS = 0（被除数: 正）の場合
　被除数が正の場合は上記の問題は発生しません。したがって、div2s命令は何も実行せずに終了します（nop命令と同じ）。

DS = 1（被除数: 負）の場合
　1）N = 0（除数: 正）の場合、tmp = AHR + rsを実行
　　N = 1（除数: 負）の場合、tmp = AHR - rsを実行
　2）1)の演算結果により、
　　tmpがゼロの場合:　　AHR = tmp(31:0)、ALR = ALR + 1として終了
　　tmpがゼロ以外の場合: AHR、ALRをそのままに終了

例　**符号付き除算例（32ビット÷32ビット）**

R0レジスタに被除数、R1レジスタに除数が設定されている場合

```
ld.w    %alr,%r0    ; ALRに被除数を設定
div0s   %r1         ; 符号付き除算の初期化ステップ
div1    %r1         ; div1を32回実行
 :       :
div1    %r1
div2s   %r1         ; 補正命令1
div3s               ; 補正命令2
```

上記命令実行後、商がALRから、余りがAHRから得られます。

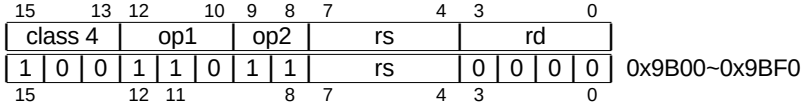
注　意　本命令はオプションの乗除算回路を内蔵した機種に限り実行可能です。オプションを含まない機種ではnopとして処理されます。

div3s （オプション）

機　能　**符号付き除算結果補正2**

標準：　**符号付き除算実行結果の補正処理**
拡張1:　**不可**
拡張2:　**不可**

コード



0x9B00~0x9BF0

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

ＣＬＫ　1サイクル

説　明　div3s命令は符号付き除算結果の補正を行います。符号なし除算の場合はdiv3s命令を実行する必要はありません。

ステップ除算によりALRから得られる商は常に正の数になります。被除数と除数の符号が異なる場合、結果は負でなければなりません。div3s命令はこの場合の符号補正を行います。
div3s命令の動作は次のとおりです。

DS = N（被除数と除数が同符号）の場合
　この場合は上記の問題は発生しません。したがって、div3s命令は何も実行せずに終了します（nop命令と同じ）。

DS = ! N（被除数と除数の符号が異なる）の場合
　ALR（商）の符号を反転します。

div2s命令およびdiv3s命令実行後、符号付き除算の最終結果が次のレジスタから得られます。
　ALR = 商　AHR = 余り

例　**符号付き除算例（32ビット÷32ビット）**

R0レジスタに被除数、R1レジスタに除数が設定されている場合

```
ld.w    %alr,%r0    ; ALRに被除数を設定
div0s   %r1         ; 符号付き除算の初期化ステップ
div1    %r1         ; div1を32回実行
 :       :
div1    %r1
div2s   %r1         ; 補正命令1
div3s               ; 補正命令2
```

上記命令実行後、商がALRから、余りがAHRから得られます。

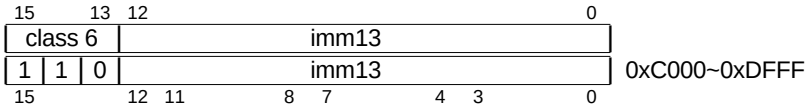
注　意　本命令はオプションの乗除算回路を内蔵した機種に限り実行可能です。オプションを含まない機種ではnopとして処理されます。

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:　**ext命令を2個続けて使用可能**
拡張2:　**不可**

コード



フラグ

IL(3:0)	MO	DS	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個以上連続して記述した場合は、最初のext命令とターゲット命令の直前にあるext命令の2個が有効となり、途中のext命令は無効となります。
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命令と拡張対象の命令の間は、リセットとアドレス不整例外を除くトラップはハードウェアによりマスクされ、発生しません。

例

```
ext   0x1000        ; 有効
ext   0x1           ; 無効
ext   0x1fff        ; 有効
add   %r1,0x3f      ; r1 = r1 + 0x8007ffff
```

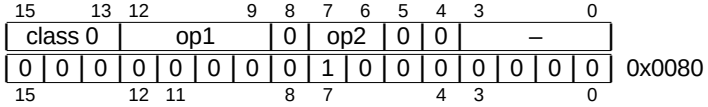
注　意　ext命令に続けてメモリとレジスタ間のロード命令を実行する場合、そのロード命令の実行前にアドレス不整例外が発生する可能性があります(ext命令で指定した即値をディスプレースメントとする指定アドレスが、転送データサイズのアドレス境界を指していない場合)。ここでアドレス不整例外が発生し、それによって実行されたトラップ処理ルーチンを単純にreti命令で終了させると、そのトラップ処理ルーチンからは例外を発生したロード命令のアドレスに戻り、直前のext命令が無効となります。したがって、リターンアドレスの操作等が必要になりますので注意してください。

halt

機 能 **HALT**

標準： **CPUをHALTモードに設定**
拡張1: **不可**
拡張2: **不可**

コード



フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

CLK 1サイクル

説 明 CPUをHALTモードにします。
これにより、CPUは動作を停止し、消費電流を抑えることができます。
チップ上の周辺回路はHALTモードでも動作します。
HALTモードは割り込みによって解除され、その割り込み処理ルーチンを実行後、halt命令の次の命令位置に戻ります。

例

```
halt                    ; CPUをHALTモードに設定
```

int imm2

機　能　**ソフトウェア例外**

標準：　**sp ← sp - 4, W[sp] ← pc + 2, sp ← sp - 4, W[sp] ← psr, pc ← Software exception vector**
拡張1:　**不可**
拡張2:　**不可**

コード

15 13	12 9	8	7 6	5	4	3 0
class 0	op1	0	op2	0	0	imm2

15 12	11 8	7 4	3 0	
0 0 0 0	0 1 0 0	1 0 0 0	0 0 imm2	0x0480~0x0483

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	1	–	–	–	–

モード　即値(符号なし)

ＣＬＫ　10サイクル

説　明　ソフトウェア例外を発生させます。
次の命令のアドレスとPSRをスタックにセーブ後、トラップテーブルからソフトウェア例外ベクタを読み出してPCにロードします。これによりソフトウェア例外処理ルーチンに分岐します。
S1C33000では4種類のソフトウェア例外に対応しており、オペランドの2ビット即値imm2によってその番号(0～3)を指定します。この番号によりトラップテーブル内のベクタアドレスが決定します。

	imm2	ベクタアドレス
ソフトウェア例外0	0	Base + 48
ソフトウェア例外1	1	Base + 52
ソフトウェア例外2	2	Base + 56
ソフトウェア例外3	3	Base + 60

Baseはトラップテーブルの先頭アドレスで、内蔵ROMよりブートするシステム(CPUのBTA3端子がHigh)では0x0080000、外部ROMよりブートするシステム(CPUのBTA3端子がLow)では0x0C00000番地です。

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              ; ソフトウェア例外2の処理ルーチンを実行
```

jp %rb / jp.d %rb

機 能 **無条件ジャンプ**

標準： **pc ← rb**
拡張1: **不可**
拡張2: **不可**

コード

15	13	12	9	8	7	6	5	4	3	0
class 0		op1		d	op2		0	0	rb	

15			12	11			8	7			4	3	0
0	0	0	0	0	1	1	d	1	0	0	0	rb	

0x0680~0x068F, 0x0780~0x078F

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード レジスタ直接（%rb = %r0~%r15）

ＣＬＫ **jp:** 2サイクル
jp.d: 1サイクル

説 明 **(1)標準**

jp %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=1）

jp.d %rb

jp.d命令では命令コード中のdビットがセットされ、次の命令がディレイド命令となります。ディレイド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命令の間はトラップがマスクされ、割り込みや例外は発生しません。

例

```
jp    %r0       ; R0レジスタが示すアドレスにジャンプ
```

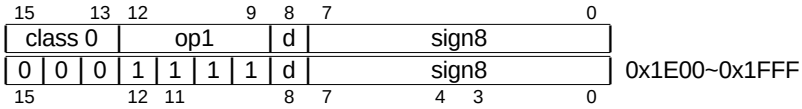
注 意 jp.d命令（ディレイド分岐）を使用する場合、次の命令はディレイド命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

jp sign8 / jp.d sign8

機 能 **無条件PC相対ジャンプ**

標準 : **pc ← pc + sign8 x 2**
拡張1: **pc ← pc + sign22**
拡張2: **pc ← pc + sign32**

コード



フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード 符号付きPC相対

ＣＬＫ **jp:** 2サイクル
jp.d: 1サイクル

説 明 **(1)標準**

```
jp      sign8           ; = "jp  sign9", sign8 = sign9(8:1), sign9(0)=0
```

PCに符号付き8ビット即値sign8を2倍して加算し、そのアドレスに分岐します。sign8は16ビット単位のハーフワードアドレスを指定します。
sign8(×2)による分岐可能範囲はPC-0x100～PC+0xFEです。

(2)拡張1

```
ext     imm13           ; = sign22(21:9)
jp      sign8           ; = "jp  sign22", sign8 = sign22(8:1), sign22(0)=0
```

ext命令の13ビット即値imm13が符号拡張され、PCに加算するディスプレースメントが符号付き22ビットとなります。sign22による分岐可能範囲はPC-0x200000～PC+0x1FFFFEです。

(3)拡張2

```
ext     imm13           ; imm13(12:3)= sign32(31:22)
ext     imm13'          ; = sign32(21:9)
jp      sign8           ; = "jp  sign32", sign8 = sign32(8:1), sign32(0)=0
```

ext命令の2つの13ビット即値imm13、imm13'が符号拡張され、PCに加算するディスプレースメントが符号付き32ビットとなります。これにより、S1C33000の全アドレス空間をカバーします。最初のimm13の下位3ビットは無視されますので注意が必要です。

(4)ディレイド分岐(dビット=1)

```
jp.d    sign8
```

jp.d命令では命令コード中のdビットがセットされ、次の命令がディレイド命令となります。ディレイド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命令の間はトラップがマスクされ、割り込みや例外は発生しません。

例

```
ext   0x8
ext   0x0
jp    0x80      ; PC+0x400100番地へジャンプ
```

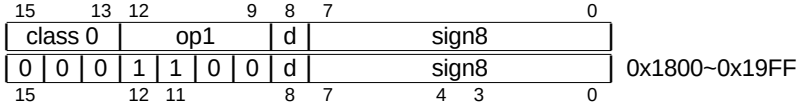
注 意 jp.d命令(ディレイド分岐)を使用する場合、次の命令はディレイド命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

jreq sign8 / jreq.d sign8

機 能 **条件付きPC相対ジャンプ**

標準： **pc ← pc + sign8 x 2 if Z is true**
拡張1: **pc ← pc + sign22 if Z is true**
拡張2: **pc ← pc + sign32 if Z is true**

コード



0x1800~0x19FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード 符号付きPC相対

ＣＬＫ **jreq:** 1サイクル(分岐しない場合), 2サイクル(分岐する場合)
jreq.d: 1サイクル

説 明 **(1)標準**

```
jreq    sign8          ; = "jreq  sign9", sign8 = sign9(8:1), sign9(0) = 0
```

次の条件が成立している場合、PCに符号付き8ビット即値sign8を2倍して加算し、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)
sign8は16ビット単位のハーフワードアドレスを指定します。sign8(×2)による分岐可能範囲はPC-0x100～PC+0xFEです。

(2)拡張1

```
ext     imm13          ; = sign22(21:9)
jreq    sign8          ; = "jreq  sign22", sign8 = sign22(8:1), sign22(0) = 0
```

ext命令の13ビット即値imm13が符号拡張され、PCに加算するディスプレースメントが符号付き22ビットとなります。sign22による分岐可能範囲はPC-0x200000～PC+0x1FFFFEです。

(3)拡張2

```
ext     imm13          ; imm13(12:3)= sign32(31:22)
ext     imm13'         ; = sign32(21:9)
jreq    sign8          ; = "jreq  sign32", sign8 = sign32(8:1), sign32(0) = 0
```

ext命令の2つの13ビット即値imm13、imm13'が符号拡張され、PCに加算するディスプレースメントが符号付き32ビットとなります。これにより、S1C33000の全アドレス空間をカバーします。最初のimm13の下位3ビットは無視されますので注意が必要です。

(4)ディレイド分岐(dビット=1)

```
jreq.d  sign8
```

jreq.d命令では命令コード中のdビットがセットされ、次の命令がディレイド命令となります。ディレイド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命令の間はトラップがマスクされ、割り込みや例外は発生しません。

例

```
cmp   %r0,%r1
jreq  0x2       ; r1＝r0ならば次の命令をスキップ
```

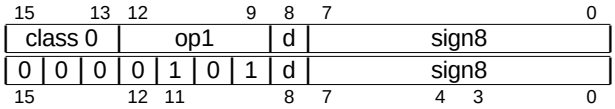
注 意 jreq.d命令(ディレイド分岐)を使用する場合、次の命令はディレイド命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

jrge sign8 / jrge.d sign8

機　能　**条件付きPC相対ジャンプ(符号付き演算結果判定)**

標準 :　**pc ← pc + sign8 x 2 if !(N^V) is true**
拡張1:　**pc ← pc + sign22 if !(N^V) is true**
拡張2:　**pc ← pc + sign32 if !(N^V) is true**

コード



0x0A00~0x0BFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　符号付きPC相対

ＣＬＫ　**jrge:**　1サイクル(分岐しない場合), 2サイクル(分岐する場合)
jrge.d: 1サイクル

説　明　**(1)標準**

```
jrge    sign8          ; = "jrge  sign9", sign8 = sign9(8:1), sign9(0) = 0
```

符号付き演算結果により次の条件が成立している場合、PCに符号付き8ビット即値sign8を2倍して加算し、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)
sign8は16ビット単位のハーフワードアドレスを指定します。sign8(×2)による分岐可能範囲はPC-0x100～PC+0xFEです。

(2)拡張1

```
ext     imm13          ; = sign22(21:9)
jrge    sign8          ; = "jrge  sign22", sign8 = sign22(8:1), sign22(0) = 0
```

ext命令の13ビット即値imm13が符号拡張され、PCに加算するディスプレースメントが符号付き22ビットとなります。sign22による分岐可能範囲はPC-0x200000～PC+0x1FFFFEです。

(3)拡張2

```
ext     imm13          ; imm13(12:3)= sign32(31:22)
ext     imm13'         ; = sign32(21:9)
jrge    sign8          ; = "jrge  sign32", sign8 = sign32(8:1), sign32(0) = 0
```

ext命令の2つの13ビット即値imm13、imm13'が符号拡張され、PCに加算するディスプレースメントが符号付き32ビットとなります。これにより、S1C33000の全アドレス空間をカバーします。最初のimm13の下位3ビットは無視されますので注意が必要です。

(4)ディレイド分岐(dビット=1)

```
jrge.d  sign8
```

jrge.d命令では命令コード中のdビットがセットされ、次の命令がディレイド命令となります。ディレイド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命令の間はトラップがマスクされ、割り込みや例外は発生しません。

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  0x2     ; r0≧r1ならば次の命令をスキップ
```

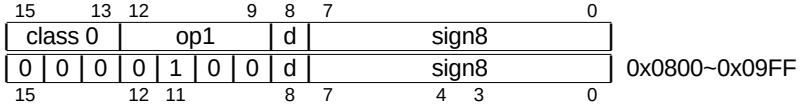
注　意　jrge.d命令(ディレイド分岐)を使用する場合、次の命令はディレイド命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

jrgt sign8 / jrgt.d sign8

機 能 **条件付きPC相対ジャンプ(符号付き演算結果判定)**

標準 : **pc ← pc + sign8 x 2 if !Z&!(N^V) is true**
拡張1: **pc ← pc + sign22 if !Z&!(N^V) is true**
拡張2: **pc ← pc + sign32 if !Z&!(N^V) is true**

コード



0x0800~0x09FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード 符号付きPC相対

CLK **jrgt:** 1サイクル(分岐しない場合), 2サイクル(分岐する場合)
jrgt.d: 1サイクル

説 明 **(1)標準**

```
jrgt    sign8          ; = "jrgt  sign9", sign8 = sign9(8:1), sign9(0) = 0
```

符号付き演算結果により次の条件が成立している場合、PCに符号付き8ビット即値sign8を2倍して加算し、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)
sign8は16ビット単位のハーフワードアドレスを指定します。sign8(×2)による分岐可能範囲はPC-0x100～PC+0xFEです。

(2)拡張1

```
ext     imm13          ; = sign22(21:9)
jrgt    sign8          ; = "jrgt  sign22", sign8 = sign22(8:1), sign22(0) = 0
```

ext命令の13ビット即値imm13が符号拡張され、PCに加算するディスプレースメントが符号付き22ビットとなります。sign22による分岐可能範囲はPC-0x200000～PC+0x1FFFFEです。

(3)拡張2

```
ext     imm13          ; imm13(12:3)= sign32(31:22)
ext     imm13'         ; = sign32(21:9)
jrgt    sign8          ; = "jrgt  sign32", sign8 = sign32(8:1), sign32(0) = 0
```

ext命令の2つの13ビット即値imm13、imm13'が符号拡張され、PCに加算するディスプレースメントが符号付き32ビットとなります。これにより、S1C33000の全アドレス空間をカバーします。最初のimm13の下位3ビットは無視されますので注意が必要です。

(4)ディレイド分岐(dビット=1)

```
jrgt.d  sign8
```

jrgt.d命令では命令コード中のdビットがセットされ、次の命令がディレイド命令となります。ディレイド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命令の間はトラップがマスクされ、割り込みや例外は発生しません。

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  0x2     ; 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 sign8 / jrle.d sign8

機　能　**条件付きPC相対ジャンプ(符号付き演算結果判定)**

標準 :　**pc ← pc + sign8 x 2 if Z | (N^V) is true**
拡張1:　**pc ← pc + sign22 if Z | (N^V) is true**
拡張2:　**pc ← pc + sign32 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	
class 0			op1				d	sign8								
0	0	0	0	1	1	1	d	sign8								0x0E00~0x0FFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　符号付きPC相対

ＣＬＫ　**jrle:**　1サイクル(分岐しない場合), 2サイクル(分岐する場合)
jrle.d:　1サイクル

説　明　**(1)標準**

```
jrle    sign8           ; = "jrle  sign9", sign8 = sign9(8:1), sign9(0) = 0
```

符号付き演算結果により次の条件が成立している場合、PCに符号付き8ビット即値sign8を2倍して加算し、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)
sign8は16ビット単位のハーフワードアドレスを指定します。sign8(×2)による分岐可能範囲はPC-0x100～PC+0xFEです。

(2)拡張1

```
ext     imm13           ; = sign22(21:9)
jrle    sign8           ; = "jrle  sign22", sign8 = sign22(8:1), sign22(0) = 0
```

ext命令の13ビット即値imm13が符号拡張され、PCに加算するディスプレースメントが符号付き22ビットとなります。sign22による分岐可能範囲はPC-0x200000～PC+0x1FFFFEです。

(3)拡張2

```
ext     imm13           ; imm13(12:3)= sign32(31:22)
ext     imm13'          ; = sign32(21:9)
jrle    sign8           ; = "jrle  sign32", sign8 = sign32(8:1), sign32(0) = 0
```

ext命令の2つの13ビット即値imm13、imm13'が符号拡張され、PCに加算するディスプレースメントが符号付き32ビットとなります。これにより、S1C33000の全アドレス空間をカバーします。最初のimm13の下位3ビットは無視されますので注意が必要です。

(4)ディレイド分岐(dビット=1)

```
jrle.d  sign8
```

jrle.d命令では命令コード中のdビットがセットされ、次の命令がディレイド命令となります。ディレイド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命令の間はトラップがマスクされ、割り込みや例外は発生しません。

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  0x2     ; r0≦r1ならば次の命令をスキップ
```

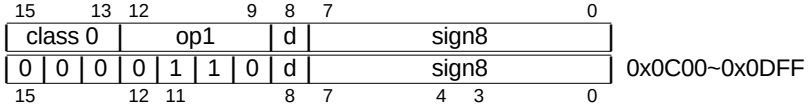
注　意　jrle.d命令(ディレイド分岐)を使用する場合、次の命令はディレイド命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

jrlt sign8 / jrlt.d sign8

機 能 **条件付きPC相対ジャンプ(符号付き演算結果判定)**

標準 : **pc ← pc + sign8 x 2 if N^V is true**
拡張1: **pc ← pc + sign22 if N^V is true**
拡張2: **pc ← pc + sign32 if N^V is true**

コード



0x0C00~0x0DFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード 符号付きPC相対

CLK **jrlt:** 1サイクル(分岐しない場合), 2サイクル(分岐する場合)
jrlt.d: 1サイクル

説 明 **(1)標準**

```
jrlt    sign8          ; = "jrlt  sign9", sign8 = sign9(8:1), sign9(0) = 0
```

符号付き演算結果により次の条件が成立している場合、PCに符号付き8ビット即値sign8を2倍して加算し、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)
sign8は16ビット単位のハーフワードアドレスを指定します。sign8(×2)による分岐可能範囲はPC-0x100～PC+0xFEです。

(2)拡張1

```
ext     imm13          ; = sign22(21:9)
jrlt    sign8          ; = "jrlt  sign22", sign8 = sign22(8:1), sign22(0) = 0
```

ext命令の13ビット即値imm13が符号拡張され、PCに加算するディスプレースメントが符号付き22ビットとなります。sign22による分岐可能範囲はPC-0x200000～PC+0x1FFFFEです。

(3)拡張2

```
ext     imm13          ; imm13(12:3)= sign32(31:22)
ext     imm13'         ; = sign32(21:9)
jrlt    sign8          ; = "jrlt  sign32", sign8 = sign32(8:1), sign32(0) = 0
```

ext命令の2つの13ビット即値imm13、imm13'が符号拡張され、PCに加算するディスプレースメントが符号付き32ビットとなります。これにより、S1C33000の全アドレス空間をカバーします。最初のimm13の下位3ビットは無視されますので注意が必要です。

(4)ディレイド分岐(dビット=1)

```
jrlt.d  sign8
```

jrlt.d命令では命令コード中のdビットがセットされ、次の命令がディレイド命令となります。ディレイド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命令の間はトラップがマスクされ、割り込みや例外は発生しません。

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  0x2     ; r0<r1ならば次の命令をスキップ
```

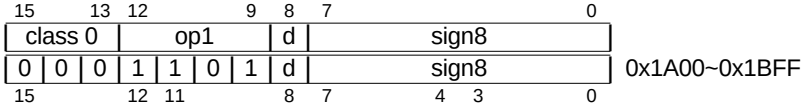
注 意 jrlt.d命令(ディレイド分岐)を使用する場合、次の命令はディレイド命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

jrne sign8 / jrne.d sign8

機 能 **条件付きPC相対ジャンプ**

標準 : **pc ← pc + sign8 x 2 if !Z is true**
拡張1: **pc ← pc + sign22 if !Z is true**
拡張2: **pc ← pc + sign32 if !Z is true**

コード



0x1A00~0x1BFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード 符号付きPC相対

ＣＬＫ **jrne:** 1サイクル(分岐しない場合), 2サイクル(分岐する場合)
jrne.d: 1サイクル

説 明 **(1)標準**

```
jrne    sign8           ; = "jrne  sign9", sign8 = sign9(8:1), sign9(0) = 0
```

次の条件が成立している場合、PCに符号付き8ビット即値sign8を2倍して加算し、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)
sign8は16ビット単位のハーフワードアドレスを指定します。sign8(×2)による分岐可能範囲はPC-0x100～PC+0xFEです。

(2)拡張1

```
ext     imm13           ; = sign22(21:9)
jrne    sign8           ; = "jrne  sign22", sign8 = sign22(8:1), sign22(0) = 0
```

ext命令の13ビット即値imm13が符号拡張され、PCに加算するディスプレースメントが符号付き22ビットとなります。sign22による分岐可能範囲はPC-0x200000～PC+0x1FFFFEです。

(3)拡張2

```
ext     imm13           ; imm13(12:3)= sign32(31:22)
ext     imm13'          ; = sign32(21:9)
jrne    sign8           ; = "jrne  sign32", sign8 = sign32(8:1), sign32(0) = 0
```

ext命令の2つの13ビット即値imm13、imm13'が符号拡張され、PCに加算するディスプレースメントが符号付き32ビットとなります。これにより、S1C33000の全アドレス空間をカバーします。最初のimm13の下位3ビットは無視されますので注意が必要です。

(4)ディレイド分岐(dビット=1)

```
jrne.d  sign8
```

jrne.d命令では命令コード中のdビットがセットされ、次の命令がディレイド命令となります。ディレイド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命令の間はトラップがマスクされ、割り込みや例外は発生しません。

例

```
cmp   %r0,%r1
jrne  0x2       ; r1≠r0ならば次の命令をスキップ
```

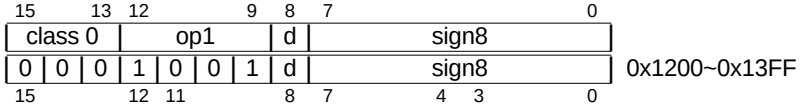
注 意 jrne.d命令(ディレイド分岐)を使用する場合、次の命令はディレイド命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

jruge sign8 / jruge.d sign8

機 能 **条件付きPC相対ジャンプ(符号なし演算結果判定)**

標準 : **pc ← pc + sign8 x 2 if !C is true**
拡張1: **pc ← pc + sign22 if !C is true**
拡張2: **pc ← pc + sign32 if !C is true**

コード



0x1200~0x13FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード 符号付きPC相対

CLK **jruge:** 1サイクル(分岐しない場合), 2サイクル(分岐する場合)
jruge.d: 1サイクル

説 明 **(1)標準**

```
jruge   sign8          ; = "jruge  sign9", sign8 = sign9(8:1), sign9(0) = 0
```

符号なし演算結果により次の条件が成立している場合、PCに符号付き8ビット即値sign8を2倍して加算し、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)
sign8は16ビット単位のハーフワードアドレスを指定します。sign8(×2)による分岐可能範囲はPC-0x100～PC+0xFEです。

(2)拡張1

```
ext     imm13          ; = sign22(21:9)
jruge   sign8          ; = "jruge  sign22", sign8 = sign22(8:1), sign22(0) = 0
```

ext命令の13ビット即値imm13が符号拡張され、PCに加算するディスプレースメントが符号付き22ビットとなります。sign22による分岐可能範囲はPC-0x200000～PC+0x1FFFFEです。

(3)拡張2

```
ext     imm13          ; imm13(12:3)= sign32(31:22)
ext     imm13'         ; = sign32(21:9)
jruge   sign8          ; = "jruge  sign32", sign8 = sign32(8:1), sign32(0) = 0
```

ext命令の2つの13ビット即値imm13、imm13'が符号拡張され、PCに加算するディスプレースメントが符号付き32ビットとなります。これにより、S1C33000の全アドレス空間をカバーします。最初のimm13の下位3ビットは無視されますので注意が必要です。

(4)ディレイド分岐(dビット=1)

```
jruge.d sign8
```

jruge.d命令では命令コード中のdビットがセットされ、次の命令がディレイド命令となります。ディレイド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命令の間はトラップがマスクされ、割り込みや例外は発生しません。

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 0x2     ; r0≧r1ならば次の命令をスキップ
```

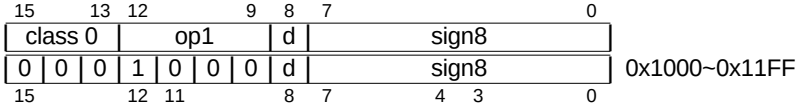
注 意 jruge.d命令(ディレイド分岐)を使用する場合、次の命令はディレイド命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

jrugt sign8 / jrugt.d sign8

機　能　**条件付きPC相対ジャンプ(符号なし演算結果判定)**

標準 :　**pc ← pc + sign8 x 2 if !Z&!C is true**
拡張1:　**pc ← pc + sign22 if !Z&!C is true**
拡張2:　**pc ← pc + sign32 if !Z&!C is true**

コード



フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　符号付きPC相対

ＣＬＫ　**jrugt:**　1サイクル(分岐しない場合), 2サイクル(分岐する場合)
jrugt.d:　1サイクル

説　明　**(1)標準**

jrugt　sign8　　; = "jrugt sign9", sign8 = sign9(8:1), sign9(0) = 0

符号なし演算結果により次の条件が成立している場合、PCに符号付き8ビット即値sign8を2倍して加算し、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)
sign8は16ビット単位のハーフワードアドレスを指定します。sign8(×2)による分岐可能範囲はPC-0x100～PC+0xFEです。

(2)拡張1

ext　imm13　　; = sign22(21:9)
jrugt　sign8　　; = "jrugt sign22", sign8 = sign22(8:1), sign22(0) = 0

ext命令の13ビット即値imm13が符号拡張され、PCに加算するディスプレースメントが符号付き22ビットとなります。sign22による分岐可能範囲はPC-0x200000～PC+0x1FFFFEです。

(3)拡張2

ext　imm13　　; imm13(12:3)= sign32(31:22)
ext　imm13'　　; = sign32(21:9)
jrugt　sign8　　; = "jrugt sign32", sign8 = sign32(8:1), sign32(0) = 0

ext命令の2つの13ビット即値imm13、imm13'が符号拡張され、PCに加算するディスプレースメントが符号付き32ビットとなります。これにより、S1C33000の全アドレス空間をカバーします。最初のimm13の下位3ビットは無視されますので注意が必要です。

(4)ディレイド分岐(dビット=1)

jrugt.d sign8

jrugt.d命令では命令コード中のdビットがセットされ、次の命令がディレイド命令となります。ディレイド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命令の間はトラップがマスクされ、割り込みや例外は発生しません。

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 0x2     ; r0＞r1ならば次の命令をスキップ
```

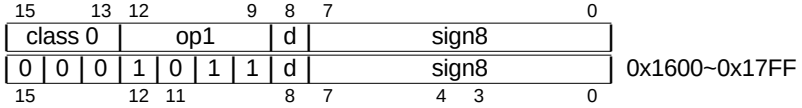
注　意　jrugt.d命令(ディレイド分岐)を使用する場合、次の命令はディレイド命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

jrule sign8 / jrule.d sign8

機 能 **条件付きPC相対ジャンプ(符号なし演算結果判定)**

標準 : **pc ← pc + sign8 x 2 if Z | C is true**
拡張1: **pc ← pc + sign22 if Z | C is true**
拡張2: **pc ← pc + sign32 if Z | C is true**

コード



0x1600~0x17FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード 符号付きPC相対

CLK **jrule:** 1サイクル(分岐しない場合), 2サイクル(分岐する場合)
jrule.d: 1サイクル

説 明 **(1)標準**

```
jrule   sign8          ; = "jrule  sign9", sign8 = sign9(8:1), sign9(0) = 0
```

符号なし演算結果により次の条件が成立している場合、PCに符号付き8ビット即値sign8を2倍して加算し、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)
sign8は16ビット単位のハーフワードアドレスを指定します。sign8(×2)による分岐可能範囲はPC-0x100～PC+0xFEです。

(2)拡張1

```
ext     imm13          ; = sign22(21:9)
jrule   sign8          ; = "jrule  sign22", sign8 = sign22(8:1), sign22(0) = 0
```

ext命令の13ビット即値imm13が符号拡張され、PCに加算するディスプレースメントが符号付き22ビットとなります。sign22による分岐可能範囲はPC-0x200000～PC+0x1FFFFEです。

(3)拡張2

```
ext     imm13          ; imm13(12:3)= sign32(31:22)
ext     imm13'         ; = sign32(21:9)
jrule   sign8          ; = "jrule  sign32", sign8 = sign32(8:1), sign32(0) = 0
```

ext命令の2つの13ビット即値imm13、imm13'が符号拡張され、PCに加算するディスプレースメントが符号付き32ビットとなります。これにより、S1C33000の全アドレス空間をカバーします。最初のimm13の下位3ビットは無視されますので注意が必要です。

(4)ディレイド分岐(dビット=1)

```
jrule.d  sign8
```

jrule.d命令では命令コード中のdビットがセットされ、次の命令がディレイド命令となります。ディレイド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命令の間はトラップがマスクされ、割り込みや例外は発生しません。

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 0x2     ; r0≦r1ならば次の命令をスキップ
```

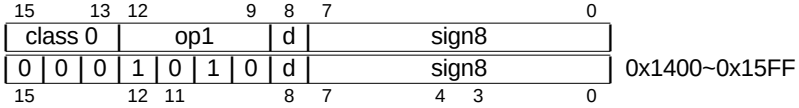
注 意 jrule.d命令(ディレイド分岐)を使用する場合、次の命令はディレイド命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

jrult sign8 / jrult.d sign8

機　能　**条件付きPC相対ジャンプ(符号なし演算結果判定)**

標準 :　**pc ← pc + sign8 x 2 if C is true**
拡張1:　**pc ← pc + sign22 if C is true**
拡張2:　**pc ← pc + sign32 if C is true**

コード



フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　符号付きPC相対

ＣＬＫ　**jrult:**　1サイクル(分岐しない場合), 2サイクル(分岐する場合)
jrult.d:　1サイクル

説　明　**(1)標準**

jrult　sign8　　; = "jrult sign9", sign8 = sign9(8:1), sign9(0) = 0

符号なし演算結果により次の条件が成立している場合、PCに符号付き8ビット即値sign8を2倍して加算し、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)
sign8は16ビット単位のハーフワードアドレスを指定します。sign8(×2)による分岐可能範囲はPC-0x100～PC+0xFEです。

(2)拡張1

ext　imm13　　; = sign22(21:9)
jrult　sign8　　; = "jrult sign22", sign8 = sign22(8:1), sign22(0) = 0

ext命令の13ビット即値imm13が符号拡張され、PCに加算するディスプレースメントが符号付き22ビットとなります。sign22による分岐可能範囲はPC-0x200000～PC+0x1FFFFEです。

(3)拡張2

ext　imm13　　; imm13(12:3)= sign32(31:22)
ext　imm13'　　; = sign32(21:9)
jrultsign8　　; = "jrult sign32", sign8 = sign32(8:1), sign32(0) = 0

ext命令の2つの13ビット即値imm13、imm13'が符号拡張され、PCに加算するディスプレースメントが符号付き32ビットとなります。これにより、S1C33000の全アドレス空間をカバーします。最初のimm13の下位3ビットは無視されますので注意が必要です。

(4)ディレイド分岐(dビット=1)

jrult.d　sign8

jrult.d命令では命令コード中のdビットがセットされ、次の命令がディレイド命令となります。ディレイド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命令の間はトラップがマスクされ、割り込みや例外は発生しません。

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 0x2     ; r0＜r1ならば次の命令をスキップ
```

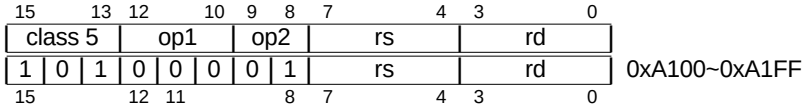
注　意　jrult.d命令(ディレイド分岐)を使用する場合、次の命令はディレイド命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

ld.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(31:8)** ← **rs(7)**
拡張1: **不可**
拡張2: **不可**

コード



0xA100~0xA1FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード **Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

ＣＬＫ 1サイクル

説 明 rsレジスタの下位8ビット（バイトデータ）を32ビットに符号拡張してrdレジスタに転送します。

例

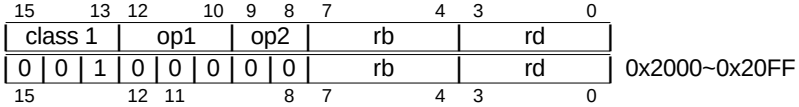
```
ld.b %r0,%r1        ; r0←r1の下位8ビットを符号拡張
```

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(31:8) ← B[rb](7)**
拡張1:　**rd(7:0) ← B[rb + imm13], rd(31:8) ← B[rb + imm13](7)**
拡張2:　**rd(7:0) ← B[rb + imm26], rd(31:8) ← B[rb + imm26](7)**

コード



フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** レジスタ間接（%rb = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

ＣＬＫ　1～2サイクル
（注）本命令は通常1サイクルで実行されます。ただし、本命令で使用したrdレジスタを直後の命令のオペランド（%rd、%rsまたは%rbとして）でも使用している場合は、もう1サイクル余分にかかります。

説　明

(1)標準

ld.b　%rd, [%rb]　; メモリアドレス = rb

指定メモリのバイトデータを32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。

(2)拡張1

ext　imm13
ld.b　%rd, [%rb]　; メモリアドレス = 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　imm13　; = imm26(25:13)
ext　imm13'　; = imm26(12:0)
ld.b　%rd, [%rb]　; メモリアドレス = rb + imm26

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26ビット即値imm26を加えたアドレスのバイトデータを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。

例

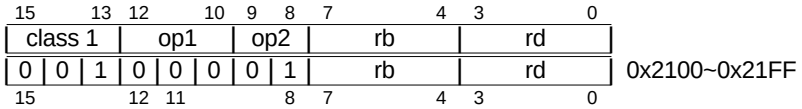
```
ext  0x10
ld.b %r0,[%r1]     ; r0←B[r1+0x10]を符号拡張
```

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(31:8) ← B[rb](7), rb ← rb + 1**
拡張1:　**不可**
拡張2:　**不可**

コード



0x2100~0x21FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** ポストインクリメント付きレジスタ間接（%rb = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

ＣＬＫ　2サイクル

説　明　指定メモリのバイトデータを32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。データ転送後、rbレジスタ内のアドレスをインクリメント(+1)します。

例

```
ld.b %r0,[%r1]+    ; r0←B[r1]を符号拡張, r1←r1+1
```

注　意　rdとrbに同一のレジスタを指定すると、rdレジスタにはデータ転送後のインクリメントされたアドレスがロードされます。

ld.b %rd, [%sp + imm6]

機　能　**符号付きバイトデータ転送**

標準 :　**rd(7:0) ← B[sp + imm6], rd(31:8) ← B[sp + imm6](7)**
拡張1:　**rd(7:0) ← B[sp + imm19], rd(31:8) ← B[sp + imm19](7)**
拡張2:　**rd(7:0) ← B[sp + imm32], rd(31:8) ← B[sp + imm32](7)**

コード

15–13	12–10	9–4	3–0	
class 2	op1	imm6	rd	
0 1 0	0 0 0	imm6	rd	0x4000~0x43FF

フラグ

IL(3:0)	MO	DS	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~%r15）

ＣＬＫ　1～2サイクル
（注）本命令は通常1サイクルで実行されます。ただし、本命令で使用したrdレジスタを直後の命令のオペランド（%rd、%rsまたは%rbとして）でも使用している場合は、もう1サイクル余分にかかります。

説　明　**(1)標準**

```
ld.b    %rd, [%sp + imm6]   ; メモリアドレス = sp + imm6
```

指定メモリのバイトデータを32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6ビット即値imm6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               ; = imm19(18:6)
ld.b    %rd, [%sp + imm6]   ; メモリアドレス = sp + imm19, imm6 = imm19(5:0)
```

ext命令により、ディスプレースメントが19ビットに拡張されます。これにより、SPの内容に19ビット即値imm19を加えたアドレスのバイトデータをrdレジスタに転送します。

(3)拡張2

```
ext     imm13               ; = imm32(31:19)
ext     imm13'              ; = imm32(18:6)
ld.b    %rd, [%sp + imm6]   ; メモリアドレス = sp + imm32, imm6 = imm32(5:0)
```

2つのext命令により、ディスプレースメントが32ビットに拡張されます。これにより、SPの内容に32ビット即値imm32を加えたアドレスのバイトデータをrdレジスタに転送します。

例

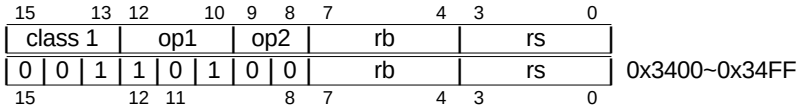
```
ext   0x1
ld.b %r0,[%sp+0x1]      ; r0←B[sp+0x41]を符号拡張
```

ld.b [%rb], %rs

機 能 **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 + imm26]** ← **rs(7:0)**

コード



0x3400~0x34FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード **Src:** レジスタ直接(**%**rs = %r0~%r15)
Dst: レジスタ間接(%rb = %r0~%r15)

CLK 1サイクル

説 明 **(1)標準**

ld.b [%rb], %rs ; メモリアドレス = 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 ; メモリアドレス = 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、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 imm13 ; = imm26(25:13)
ext imm13' ; = imm26(12:0)
ld.b [%rb], %rs ; メモリアドレス = rb + imm26

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26ビット即値imm26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。

例

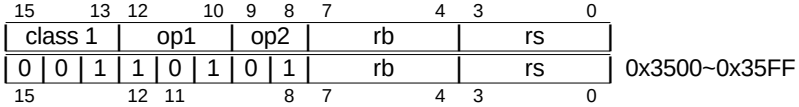
```
ext  0x10
ld.b [%r1],%r0     ; B[r1+0x10]←r0の下位8ビット
```

ld.b [%rb]+, %rs

機　能　**バイトデータ転送**

標準 :　**B[rb]** ← **rs(7:0), rb** ← **rb + 1**
拡張1:　**不可**
拡張2:　**不可**

コード



0x3500~0x35FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** レジスタ直接（%rs = %r0~%r15）
Dst: ポストインクリメント付きレジスタ間接（%rb = %r0~%r15）

ＣＬＫ　1サイクル

説　明　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。データ転送後、rbレジスタ内のアドレスをインクリメント(+1)します。

例

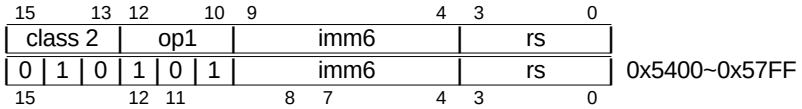
```
ld.b [%r1]+,%r0    ; B[r1]←r0の下位8ビット, r1←r1+1
```

ld.b [%sp + imm6], %rs

機 能 **バイトデータ転送**

標準： **B[sp + imm6]** ← **rs(7:0)**
拡張1: **B[sp + imm19]** ← **rs(7:0)**
拡張2: **B[sp + imm32]** ← **rs(7:0)**

コード



0x5400~0x57FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード **Src:** レジスタ直接（%rd = %r0~%r15）
Dst: ディスプレースメント付きレジスタ間接

ＣＬＫ 1サイクル

説 明 **(1)標準**

```
ld.b    [%sp + imm6], %rs    ; メモリアドレス = sp + imm6
```

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6ビット即値imm6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                ; = imm19(18:6)
ld.b    [%sp + imm6], %rs    ; メモリアドレス = sp + imm19, imm6 = imm19(5:0)
```

ext命令により、ディスプレースメントが19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19ビット即値imm19を加えたアドレスに転送します。

(3)拡張2

```
ext     imm13                ; = imm32(31:19)
ext     imm13'               ; = imm32(18:6)
ld.b    [%sp + imm6], %rs    ; メモリアドレス = sp + imm32, imm6 = imm32(5:0)
```

2つのext命令により、ディスプレースメントが32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32ビット即値imm32を加えたアドレスに転送します。

例

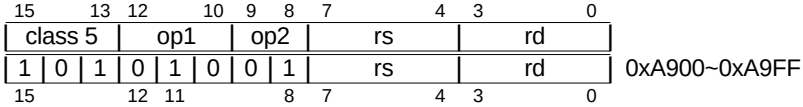
```
ext   0x1
ld.b [%sp+0x1],%r0       ; B[sp+0x41]←r0の下位8ビット
```

ld.h %rd, %rs

機　能　**符号付きハーフワードデータ転送**

標準 :　**rd(15:0)** ← **rs(15:0), rd(31:16)** ← **rs(15)**
拡張1:　**不可**
拡張2:　**不可**

コード



0xA900~0xA9FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

ＣＬＫ　1サイクル

説　明　rsレジスタの下位16ビット（ハーフワードデータ）を32ビットに符号拡張してrdレジスタに転送します。

例

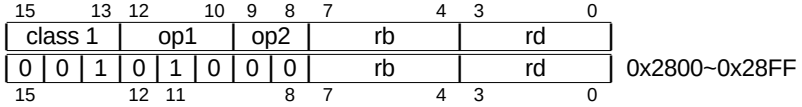
```
ld.h %r0,%r1        ; r0←r1の下位16ビットを符号拡張
```

ld.h %rd, [%rb]

機 能 **符号付きハーフワードデータ転送**

標準： **rd(15:0) ← H[rb], rd(31:16) ← H[rb](15)**
拡張1: **rd(15:0) ← H[rb + imm13], rd(31:16) ← H[rb + imm13](15)**
拡張2: **rd(15:0) ← H[rb + imm26], rd(31:16) ← H[rb + imm26](15)**

コード



0x2800~0x28FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード **Src:** レジスタ間接（%rb = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

CLK 1～2サイクル
（注）本命令は通常1サイクルで実行されます。ただし、本命令で使用したrdレジスタを直後の命令のオペランド（%rd、%rsまたは%rbとして）でも使用している場合は、もう1サイクル余分にかかります。

説 明 **(1)標準**

```
ld.h    %rd, [%rb]    ; メモリアドレス = rb
```

指定メモリのハーフワードデータを32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。

(2)拡張1

```
ext     imm13
ld.h    %rd, [%rb]    ; メモリアドレス = 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ハーフワードデータを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。

(3)拡張2

```
ext     imm13         ; = imm26(25:13)
ext     imm13'        ; = imm26(12:0)
ld.h    %rd, [%rb]    ; メモリアドレス = rb + imm26
```

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26ビット即値imm26を加えたアドレスのハーフワードデータを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。

例

```
ext   0x10
ld.h %r0,[%r1]      ; r0←H[r1+0x10]を符号拡張
```

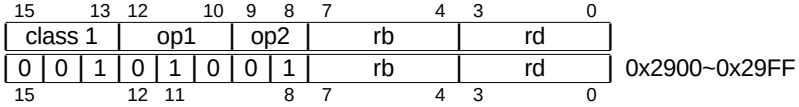
注 意 rbレジスタおよびディスプレースメントで指定されるメモリアドレスは、ハーフワード境界（最下位ビット=0）を示していることが必要です。奇数アドレスが指定されると、アドレス不整例外が発生します。
なお、データ転送は指定アドレスのデータを下位8ビット、次のアドレスのデータを上位8ビットとして行われます。

ld.h %rd, [%rb]+

機　能　**符号付きハーフワードデータ転送**

標準 :　**rd(15:0)** ← **H[rb], rd(31:16)** ← **H[rb](15), rb** ← **rb + 2**
拡張1:　**不可**
拡張2:　**不可**

コード



0x2900~0x29FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** ポストインクリメント付きレジスタ間接(%rb = %r0~%r15)
Dst: レジスタ直接(%rd = %r0~%r15)

ＣＬＫ　2サイクル

説　明　指定メモリのハーフワードデータを32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。データ転送後、rbレジスタ内のアドレスをインクリメント(+2)します。

例

```
ld.h %r0,[%r1]+    ; r0←H[r1]を符号拡張, r1←r1+2
```

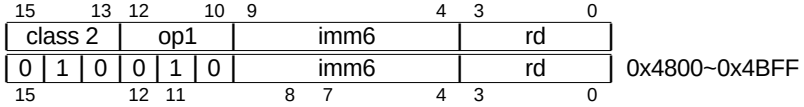
注　意
- rbレジスタで指定されるメモリアドレスは、ハーフワード境界(最下位ビット=0)を示していることが必要です。奇数アドレスが指定されると、アドレス不整例外が発生します。
  なお、データ転送は指定アドレスのデータを下位8ビット、次のアドレスのデータを上位8ビットとして行われます。
- rdとrbに同一のレジスタを指定すると、rdレジスタにはデータ転送後のインクリメントされたアドレスがロードされます。

ld.h %rd, [%sp + imm6]

機 能 **符号付きハーフワードデータ転送**

標準： **rd(15:0) ← H[sp + imm6 x 2], rd(31:16) ← H[sp + imm6 x 2](15)**
拡張1: **rd(15:0) ← H[sp + imm19], rd(31:16) ← H[sp + imm19](15)**
拡張2: **rd(15:0) ← H[sp + imm32], rd(31:16) ← H[sp + imm32](15)**

コード



0x4800~0x4BFF

フラグ

IL(3:0)	MO	DS	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~%r15)

CLK 1～2サイクル
(注) 本命令は通常1サイクルで実行されます。ただし、本命令で使用したrdレジスタを直後の命令のオペランド(%rd、%rsまたは%rbとして)でも使用している場合は、もう1サイクル余分にかかります。

説 明 **(1)標準**

```
ld.h    %rd, [%sp + imm6]   ; メモリアドレス = sp + imm6 x 2
```

指定メモリのハーフワードデータを32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、6ビット即値imm6を2倍した値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。imm6は16ビット単位のハーフワードアドレスを指定します。ディスプレースメントの最下位ビットは常に0となります。

(2)拡張1

```
ext     imm13               ; = imm19(18:6)
ld.h    %rd, [%sp + imm6]   ; メモリアドレス = sp + imm19, imm6 = imm19(5:0)
```

ext命令により、ディスプレースメントが19ビットに拡張されます。これにより、SPの内容に19ビット即値imm19を加えたアドレスのハーフワードデータをrdレジスタに転送します。
なお、imm6はハーフワード境界(最下位ビット=0)を指定してください。

(3)拡張2

```
ext     imm13               ; = imm32(31:19)
ext     imm13'              ; = imm32(18:6)
ld.h    %rd, [%sp + imm6]   ; メモリアドレス = sp + imm32, imm6 = imm32(5:0)
```

2つのext命令により、ディスプレースメントが32ビットに拡張されます。これにより、SPの内容に32ビット即値imm32を加えたアドレスのハーフワードデータをrdレジスタに転送します。
なお、imm6はハーフワード境界(最下位ビット=0)を指定してください。

例

```
ext   0x1
ext   0x0
ld.h %r1,[%sp+0x2] ; r1←H[SP+0x80002]を符号拡張
```

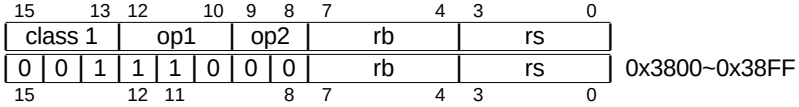
注 意 ディスプレースメントを拡張する場合、imm6の最下位ビットは0として扱われ、常にハーフワード境界をアドレスします。したがって、アドレス不整例外は発生しません。
なお、データ転送は指定アドレスのデータを下位8ビット、次のアドレスのデータを上位8ビットとして行われます。

ld.h [%rb], %rs

機 能 **ハーフワードデータ転送**

標準 : **H[rb]** ← **rs(15:0)**
拡張1: **H[rb + imm13]** ← **rs(15:0)**
拡張2: **H[rb + imm26]** ← **rs(15:0)**

コード



0x3800~0x38FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード **Src:** レジスタ直接(%rs = %r0~%r15)
Dst: レジスタ間接(%rb = %r0~%r15)

ＣＬＫ 1サイクル

説 明 **(1)標準**

ld.h [%rb], %rs ; メモリアドレス = rb

rsレジスタの下位16ビットを指定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.h [%rb], %rs ; メモリアドレス = 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、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 imm13 ; = imm26(25:13)
ext imm13' ; = imm26(12:0)
ld.h [%rb], %rs ; メモリアドレス = rb + imm26

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26ビット即値imm26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。

例

```
ext  0x10
ld.h [%r1],%r0     ; H[r1+0x10]←r0の下位16ビット
```

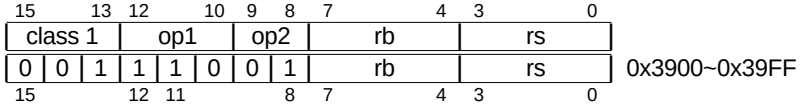
注 意 rbレジスタおよびディスプレースメントで指定されるメモリアドレスは、ハーフワード境界(最下位ビット=0)を示していることが必要です。奇数アドレスが指定されると、アドレス不整例外が発生します。
なお、指定アドレスに下位8ビットデータが、次のアドレスに上位8ビットデータが転送されます。

ld.h [%rb]+, %rs

機　能　**ハーフワードデータ転送**

標準：　**H[rb] ← rs(15:0), rb ← rb + 2**
拡張1:　**不可**
拡張2:　**不可**

コード



0x3900~0x39FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** レジスタ直接（%rs = %r0~%r15）
Dst: ポストインクリメント付きレジスタ間接（%rb = %r0~%r15）

ＣＬＫ　1サイクル

説　明　rsレジスタの下位16ビットを指定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。データ転送後、rbレジスタ内のアドレスをインクリメント(+2)します。

例

```
ld.h [%r1]+,%r0    ; H[r1]←r0の下位16ビット, r1←r1+2
```

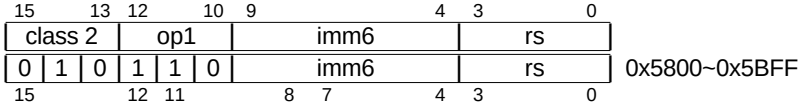
注　意　rbレジスタで指定されるメモリアドレスは、ハーフワード境界(最下位ビット=0)を示していることが必要です。奇数アドレスが指定されると、アドレス不整例外が発生します。
なお、指定アドレスに下位8ビットデータが、次のアドレスに上位8ビットデータが転送されます。

ld.h [%sp + imm6], %rs

機 能 **ハーフワードデータ転送**

標準 : **H[sp + imm6 x 2] ← rs(15:0)**
拡張1: **H[sp + imm19] ← rs(15:0)**
拡張2: **H[sp + imm32] ← rs(15:0)**

コード



0x5800~0x5BFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード **Src:** レジスタ直接(%rs = %r0~%r15)
Dst: ディスプレースメント付きレジスタ間接

ＣＬＫ 1サイクル

説 明 **(1)標準**

ld.h [%sp + imm6], %rs ; メモリアドレス = sp + imm6 x 2

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6ビット即値imm6を2倍した値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。imm6は16ビット単位のハーフワードアドレスを指定します。ディスプレースメントの最下位ビットは常に0となります。

(2)拡張1

ext imm13 ; = imm19(18:6)
ld.h [%sp + imm6], %rs ; メモリアドレス = sp + imm19, imm6 = imm19(5:0)

ext命令により、ディスプレースメントが19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19ビット即値imm19を加えたアドレスに転送します。
なお、imm6はハーフワード境界(最下位ビット=0)を指定してください。

(3)拡張2

ext imm13 ; = imm32(31:19)
ext imm13' ; = imm32(18:6)
ld.h [%sp + imm6], %rs ; メモリアドレス = sp + imm32, imm6 = imm32(5:0)

2つのext命令により、ディスプレースメントが32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32ビット即値imm32を加えたアドレスに転送します。
なお、imm6はハーフワード境界(最下位ビット=0)を指定してください。

例

```
ext  0x1
ext  0x0
ld.h [%sp+0x2],%r1 ; H[SP+0x80002]← r1の下位16ビット
```

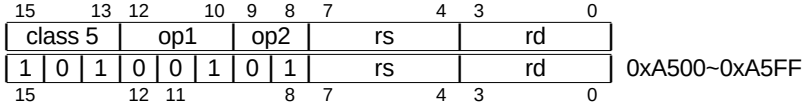
注 意 ディスプレースメントを拡張する場合、imm6の最下位ビットは0として扱われ、常にハーフワード境界をアドレスします。したがって、アドレス不整例外は発生しません。
なお、データ転送は指定アドレスのデータを下位8ビット、次のアドレスのデータを上位8ビットとして行われます。

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(31:8)** ← **0**
拡張1:　**不可**
拡張2:　**不可**

コード



0xA500~0xA5FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

ＣＬＫ　1サイクル

説　明　rsレジスタの下位8ビット（バイトデータ）を32ビットにゼロ拡張してrdレジスタに転送します。

例

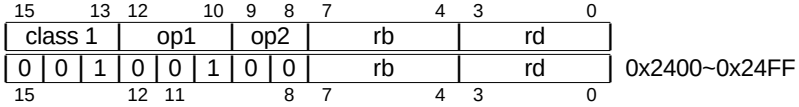
```
ld.ub %r0,%r1       ; r0←r1の下位8ビットをゼロ拡張
```

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(31:8) ← 0**
拡張1:　**rd(7:0) ← B[rb + imm13], rd(31:8) ← 0**
拡張2:　**rd(7:0) ← B[rb + imm26], rd(31:8) ← 0**

コード



0x2400~0x24FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** レジスタ間接（%rb = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

ＣＬＫ　1～2サイクル
（注）本命令は通常1サイクルで実行されます。ただし、本命令で使用したrdレジスタを直後の命令のオペランド（%rd、%rsまたは%rbとして）でも使用している場合は、もう1サイクル余分にかかります。

説　明

(1)標準

```
ld.ub   %rd, [%rb]     ; メモリアドレス = rb
```

指定メモリのバイトデータを32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。

(2)拡張1

```
ext     imm13
ld.ub   %rd, [%rb]     ; メモリアドレス = 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     imm13          ; = imm26(25:13)
ext     imm13'         ; = imm26(12:0)
ld.ub   %rd, [%rb]     ; メモリアドレス = rb + imm26
```

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26ビット即値imm26を加えたアドレスのバイトデータを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。

例

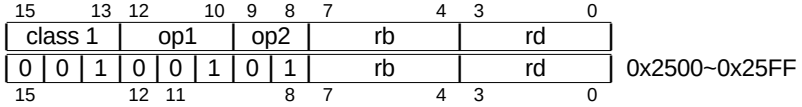
```
ext    0x10
ld.ub  %r0,[%r1]    ; r0←B[r1+0x10]をゼロ拡張
```

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(31:8) ← 0, rb ← rb + 1**
拡張1:　**不可**
拡張2:　**不可**

コード



0x2500~0x25FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** ポストインクリメント付きレジスタ間接（%rb = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

ＣＬＫ　2サイクル

説　明　指定メモリのバイトデータを32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。データ転送後、rbレジスタ内のアドレスをインクリメント（+1）します。

例

```
ld.ub %r0,[%r1]+  ; r0←B[r1]をゼロ拡張, r1←r1+1
```

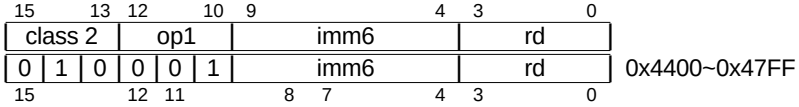
注　意　rdとrbに同一のレジスタを指定すると、rdレジスタにはデータ転送後のインクリメントされたアドレスがロードされます。

ld.ub %rd, [%sp + imm6]

機　能　**符号なしバイトデータ転送**

標準 :　**rd(7:0) ← B[sp + imm6], rd(31:8) ← 0**
拡張1:　**rd(7:0) ← B[sp + imm19], rd(31:8) ← 0**
拡張2:　**rd(7:0) ← B[sp + imm32], rd(31:8) ← 0**

コード



0x4400~0x47FF

フラグ

IL(3:0)	MO	DS	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~%r15）

ＣＬＫ　1～2サイクル
（注）本命令は通常1サイクルで実行されます。ただし、本命令で使用したrdレジスタを直後の命令のオペランド（%rd、%rsまたは%rbとして）でも使用している場合は、もう1サイクル余分にかかります。

説　明　**(1)標準**

```
ld.ub   %rd, [%sp + imm6]   ; メモリアドレス = sp + imm6
```

指定メモリのバイトデータを32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6ビット即値imm6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               ; = imm19(18:6)
ld.ub   %rd, [%sp + imm6]   ; メモリアドレス = sp + imm19, imm6 = imm19(5:0)
```

ext命令により、ディスプレースメントが19ビットに拡張されます。これにより、SPの内容に19ビット即値imm19を加えたアドレスのバイトデータをrdレジスタに転送します。

(3)拡張2

```
ext     imm13               ; = imm32(31:19)
ext     imm13'              ; = imm32(18:6)
ld.ub   %rd, [%sp + imm6]   ; メモリアドレス = sp + imm32, imm6 = imm32(5:0)
```

2つのext命令により、ディスプレースメントが32ビットに拡張されます。これにより、SPの内容に32ビット即値imm32を加えたアドレスのバイトデータをrdレジスタに転送します。

例

```
ext    0x1
ld.ub  %r0,[%sp+0x1]      ; r0←B[sp+0x41]をゼロ拡張
```

ld.uh %rd, %rs

機 能 **符号なしハーフワードデータ転送**

標準： **rd(15:0)** ← **rs(15:0), rd(31:16)** ← **0**
拡張1: **不可**
拡張2: **不可**

コード

15–13	12–10	9–8	7–4	3–0
class 5	op1	op2	rs	rd

15	14	13	12	11	10	9	8	7–4	3–0	
1	0	1	0	1	1	0	1	rs	rd	0xAD00~0xADFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード **Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

ＣＬＫ 1サイクル

説 明 rsレジスタの下位16ビット（ハーフワードデータ）を32ビットにゼロ拡張してrdレジスタに転送します。

例

```
ld.uh %r0,%r1       ; r0←r1の下位16ビットをゼロ拡張
```

ld.uh %rd, [%rb]

機　能　**符号なしハーフワードデータ転送**

標準 :　**rd(15:0) ← H[rb], rd(31:16) ← 0**
拡張1:　**rd(15:0) ← H[rb + imm13], rd(31:16) ← 0**
拡張2:　**rd(15:0) ← H[rb + imm26], rd(31:16) ← 0**

コード

15 13	12 10	9 8	7 4	3 0
class 1	op1	op2	rb	rd

15			12	11			8	7			4	3			0	
0	0	1	0	1	1	0	0	rb				rd				0x2C00~0x2CFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** レジスタ間接（%rb = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

ＣＬＫ　1～2サイクル
（注）本命令は通常1サイクルで実行されます。ただし、本命令で使用したrdレジスタを直後の命令のオペランド（%rd、%rsまたは%rbとして）でも使用している場合は、もう1サイクル余分にかかります。

説　明

(1)標準

```
ld.uh   %rd, [%rb]     ; メモリアドレス = rb
```

指定メモリのハーフワードデータを32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。

(2)拡張1

```
ext     imm13
ld.uh   %rd, [%rb]     ; メモリアドレス = 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ハーフワードデータを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。

(3)拡張2

```
ext     imm13          ; = imm26(25:13)
ext     imm13'         ; = imm26(12:0)
ld.uh   %rd, [%rb]     ; メモリアドレス = rb + imm26
```

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26ビット即値imm26を加えたアドレスのハーフデータを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。

例

```
ext    0x10
ld.uh  %r0,[%r1]    ; r0←H[r1+0x10]をゼロ拡張
```

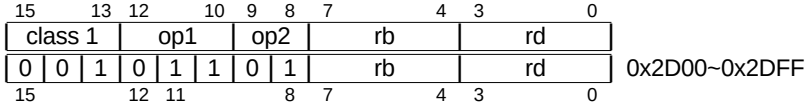
注　意　rbレジスタおよびディスプレースメントで指定されるメモリアドレスは、ハーフワード境界（最下位ビット=0）を示していることが必要です。奇数アドレスが指定されると、アドレス不整例外が発生します。
なお、データ転送は指定アドレスのデータを下位8ビット、次のアドレスのデータを上位8ビットとして行われます。

ld.uh %rd, [%rb]+

機　能　**符号なしハーフワードデータ転送**

標準：　**rd(15:0)** ← **H[rb], rd(31:16)** ← **0, rb** ← **rb + 2**
拡張1:　**不可**
拡張2:　**不可**

コード



0x2D00~0x2DFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** ポストインクリメント付きレジスタ間接（%rb = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

ＣＬＫ　2サイクル

説　明　指定メモリのハーフワードデータを32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。データ転送後、rbレジスタ内のアドレスをインクリメント(+2)します。

例

```
ld.uh %r0,[%r1]+   ; r0←H[r1]をゼロ拡張, r1←r1+2
```

注　意

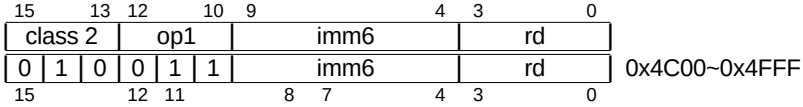
- rbレジスタで指定されるメモリアドレスは、ハーフワード境界(最下位ビット=0)を示していることが必要です。奇数アドレスが指定されると、アドレス不整例外が発生します。
  なお、データ転送は指定アドレスのデータを下位8ビット、次のアドレスのデータを上位8ビットとして行われます。
- rdとrbに同一のレジスタを指定すると、rdレジスタにはデータ転送後のインクリメントされたアドレスがロードされます。

ld.uh %rd, [%sp + imm6]

機　能　**符号なしハーフワードデータ転送**

標準 :　**rd(15:0) ← H[sp + imm6 x 2], rd(31:16) ← 0**
拡張1:　**rd(15:0) ← H[sp + imm19], rd(31:16) ← 0**
拡張2:　**rd(15:0) ← H[sp + imm32], rd(31:16) ← 0**

コード



フラグ

IL(3:0)	MO	DS	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~%r15）

ＣＬＫ　1～2サイクル
（注）本命令は通常1サイクルで実行されます。ただし、本命令で使用したrdレジスタを直後の命令のオペランド（%rd、%rsまたは%rbとして）でも使用している場合は、もう1サイクル余分にかかります。

説　明　**(1)標準**

ld.uh　%rd, [%sp + imm6]　; メモリアドレス = sp + imm6 x 2

指定メモリのハーフワードデータを32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、6ビット即値imm6を2倍した値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。imm6は16ビット単位のハーフワードアドレスを指定します。ディスプレースメントの最下位ビットは常に0となります。

(2)拡張1

```
ext     imm13                ; = imm19(18:6)
ld.uh   %rd, [%sp + imm6]    ; メモリアドレス = sp + imm19, imm6 = imm19(5:0)
```

ext命令により、ディスプレースメントが19ビットに拡張されます。これにより、SPの内容に19ビット即値imm19を加えたアドレスのハーフワードデータをrdレジスタに転送します。
なお、imm6はハーフワード境界（最下位ビット=0）を指定してください。

(3)拡張2

```
ext     imm13                ; = imm32(31:19)
ext     imm13'               ; = imm32(18:6)
ld.uh   %rd, [%sp + imm6]    ; メモリアドレス = sp + imm32, imm6 = imm32(5:0)
```

2つのext命令により、ディスプレースメントが32ビットに拡張されます。これにより、SPの内容に32ビット即値imm32を加えたアドレスのハーフワードデータをrdレジスタに転送します。
なお、imm6はハーフワード境界（最下位ビット=0）を指定してください。

例

```
ext    0x1
ext    0x0
ld.uh  %r1,[%sp+0x2]      ; r1←H[SP+0x80002]をゼロ拡張
```

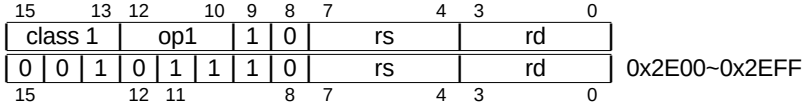
注　意　ディスプレースメントを拡張する場合、imm6の最下位ビットは0として扱われ、常にハーフワード境界をアドレスします。したがって、アドレス不整例外は発生しません。
なお、データ転送は指定アドレスのデータを下位8ビット、次のアドレスのデータを上位8ビットとして行われます。

ld.w %rd, %rs

機 能 **ワードデータ転送**

標準： **rd ← rs**
拡張1: **不可**
拡張2: **不可**

コード



0x2E00~0x2EFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード **Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

CLK 1サイクル

説 明 **(1)標準**

rsレジスタの内容（ワードデータ）をrdレジスタに転送します。

(2)ディレイド命令

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命令として実行されます。

例

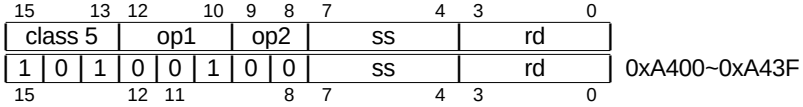
```
ld.w %r0,%r1        ; r0←r1
```

ld.w %rd, %ss

機　能　**ワードデータ転送**

標準 :　**rd ← ss**
拡張1:　**不可**
拡張2:　**不可**

コード



0xA400~0xA43F

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** レジスタ直接（%ss = %sp, %psr, %alr, %ahr）
Dst: レジスタ直接（%rd = %r0~%r15）

ＣＬＫ　1サイクル

説　明　特殊レジスタ（SP、PSR、ALR、AHR）の内容（ワードデータ）をrdレジスタに転送します。

例

```
ld.w %r0,%psr      ; r0←psr
```

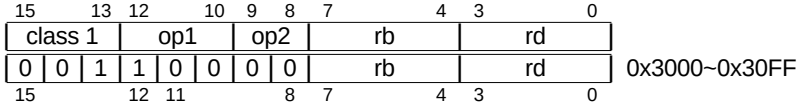
注　意　ALRおよびAHRレジスタはオプションの乗除算回路を内蔵した機種に限り使用可能です。オプションを含まない機種でソースレジスタにALRまたはAHRを指定した場合、本命令はnopとして処理されます。

ld.w %rd, [%rb]

機 能 **ワードデータ転送**

標準 : **rd** ← **W[rb]**
拡張1: **rd** ← **W[rb + imm13]**
拡張2: **rd** ← **W[rb + imm26]**

コード



0x3000~0x30FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード **Src:** レジスタ間接(%rb = %r0~%r15)
Dst: レジスタ直接(%rd = %r0~%r15)

CLK 1～2サイクル
(注) 本命令は通常1サイクルで実行されます。ただし、本命令で使用したrdレジスタを直後の命令のオペランド(%rd、%rsまたは%rbとして)でも使用している場合は、もう1サイクル余分にかかります。

説 明 **(1)標準**

```
ld.w    %rd, [%rb]    ; メモリアドレス = rb
```

指定メモリのワードデータを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。

(2)拡張1

```
ext     imm13
ld.w    %rd, [%rb]    ; メモリアドレス = 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ワードデータを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。

(3)拡張2

```
ext     imm13         ; = imm26(25:13)
ext     imm13'        ; = imm26(12:0)
ld.w    %rd, [%rb]    ; メモリアドレス = rb + imm26
```

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26ビット即値imm26を加えたアドレスのワードデータを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。

例

```
ext   0x10
ld.w %r0,[%r1]      ; r0←W[r1+0x10]
```

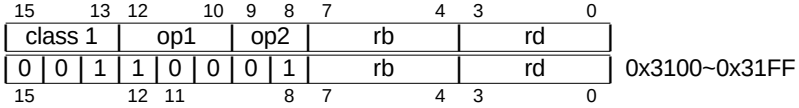
注 意 rbレジスタおよびディスプレースメントで指定されるメモリアドレスは、ワード境界(下位2ビット=0)を示していることが必要です。それ以外のアドレスが指定されると、アドレス不整例外が発生します。
なお、データ転送は指定アドレスを下位8ビットとして1ワード(4アドレス)分行われます。

ld.w %rd, [%rb]+

機　能　**ワードデータ転送**

標準 :　**rd ← W[rb], rb ← rb + 4**
拡張1:　**不可**
拡張2:　**不可**

コード



0x3100~0x31FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** ポストインクリメント付きレジスタ間接(%rb = %r0~%r15)
Dst: レジスタ直接(%rd = %r0~%r15)

ＣＬＫ　2サイクル

説　明　指定メモリのワードデータを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。データ転送後、rbレジスタ内のアドレスをインクリメント(+4)します。

例

```
ld.w %r0,[%r1]+    ; r0←W[r1], r1←r1+4
```

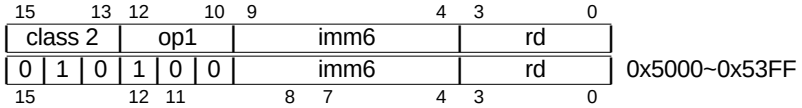
注　意
- rbレジスタで指定されるメモリアドレスは、ワード境界(下位2ビット=0)を示していることが必要です。それ以外のアドレスが指定されると、アドレス不整例外が発生します。
  なお、データ転送は指定アドレスを下位8ビットとして1ワード(4アドレス)分行われます。
- rdとrbに同一のレジスタを指定すると、rdレジスタにはデータ転送後のインクリメントされたアドレスがロードされます。

ld.w %rd, [%sp + imm6]

機 能 **ワードデータ転送**

標準： **rd ← W[sp + imm6 x 4]**
拡張1: **rd ← W[sp + imm19]**
拡張2: **rd ← W[sp + imm32]**

コード



0x5000~0x53FF

フラグ

IL(3:0)	MO	DS	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~%r15)

CLK 1～2サイクル
(注) 本命令は通常1サイクルで実行されます。ただし、本命令で使用したrdレジスタを直後の命令のオペランド(%rd、%rsまたは%rbとして)でも使用している場合は、もう1サイクル余分にかかります。

説 明 **(1)標準**

```
ld.w    %rd, [%sp + imm6]   ; メモリアドレス = sp + imm6 x 4
```

指定メモリのワードデータをrdレジスタに転送します。現在のSPの内容に6ビット即値imm6を4倍した値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。imm6は32ビット単位のワードアドレスを指定します。ディスプレースメントの下位2ビットは0に固定されます。

(2)拡張1

```
ext     imm13               ; = imm19(18:6)
ld.w    %rd, [%sp + imm6]   ; メモリアドレス = sp + imm19, imm6 = imm19(5:0)
```

ext命令により、ディスプレースメントが19ビットに拡張されます。これにより、SPの内容に19ビット即値imm19を加えたアドレスのワードデータをrdレジスタに転送します。
なお、imm6はワード境界(下位2ビット=0)を指定してください。

(3)拡張2

```
ext     imm13               ; = imm32(31:19)
ext     imm13'              ; = imm32(18:6)
ld.w    %rd, [%sp + imm6]   ; メモリアドレス = sp + imm32, imm6 = imm32(5:0)
```

2つのext命令により、ディスプレースメントが32ビットに拡張されます。これにより、SPの内容に32ビット即値imm32を加えたアドレスのワードデータをrdレジスタに転送します。
なお、imm6はワード境界(下位2ビット=0)を指定してください。

例

```
ext   0x1
ext   0x0
ld.w %r1,[%sp+0x4] ; r1←W[SP+0x80004]
```

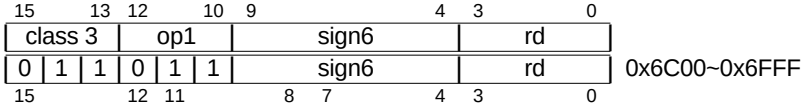
注 意 ディスプレースメントを拡張する場合、imm6の下位2ビットは0として扱われ、常にワード境界をアドレスします。したがって、アドレス不整例外は発生しません。
なお、データ転送は指定アドレスを下位8ビットとして1ワード(4アドレス)分行われます。

ld.w %rd, sign6

機　能　**ワードデータ転送**

標準 :　**rd(5:0) ← sign6(5:0), rd(31:6) ← sign6(5)**
拡張1:　**rd(18:0) ← sign19(18:0), rd(31:19) ← sign19(18)**
拡張2:　**rd ← sign32**

コード



0x6C00~0x6FFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** 即値(符号付き)
Dst: レジスタ直接(%rd = %r0~%r15)

ＣＬＫ　1サイクル

説　明　**(1)標準**

ld.w　%rd, sign6　; rd ← 符号拡張 ← sign6
6ビット即値データsign6を32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　; = sign19(18:6)
ld.w　%rd, sign6　; rd ← 符号拡張 ← sign19, sign6 = sign19(5:0)
ext命令で拡張した19ビット即値データsign19を32ビットに符号拡張してrdレジスタにロードします。

(3)拡張2

ext　imm13　; = sign32(31:19)
ext　imm13'　; = sign32(18:6)
ld.w　%rd, sign6　; rd ← sign32, sign6 = sign32(5:0)
ext命令で拡張した32ビット即値データsign32をrdレジスタにロードします。

(4)ディレイド命令

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命令として実行されます。ただし、その場合はext命令による拡張はできません。

例

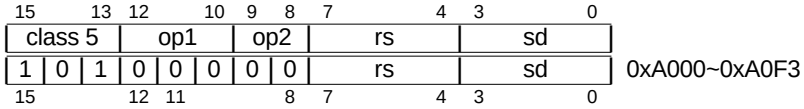
```
ld.w %r0,0x3f      ; r0←0xffffffff
```

ld.w %sd, %rs

機　能　**ワードデータ転送**

標準：　**sd ← rs**
拡張1:　**不可**
拡張2:　**不可**

コード



0xA000~0xA0F3

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

（%sd=%psrの場合は、すべて変更）

モード　**Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ直接（%sd = %sp, %psr, %alr, %ahr）

ＣＬＫ　1サイクル

説　明　rsレジスタの内容（ワードデータ）を特殊レジスタ（SP、PSR、ALR、AHR）に転送します。

例

```
ld.w %sp,%r0        ; sp←r0
```

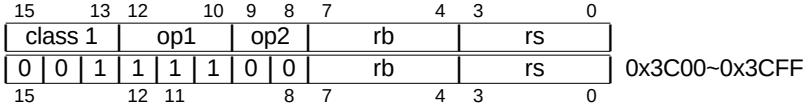
注　意　ALRおよびAHRレジスタはオプションの乗除算回路を内蔵した機種に限り使用可能です。オプションを含まない機種でディスティネーションレジスタにALRまたはAHRを指定した場合、本命令はnopとして処理されます。

ld.w [%rb], %rs

機　能　**ワードデータ転送**

標準 :　**W[rb]** ← **rs**
拡張1:　**W[rb + imm13]** ← **rs**
拡張2:　**W[rb + imm26]** ← **rs**

コード



0x3C00~0x3CFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ間接（%rb = %r0~%r15）

ＣＬＫ　1サイクル

説　明　**(1)標準**

ld.w　[%rb], %rs　; メモリアドレス = rb

rsレジスタの内容（ワードデータ）を指定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.w　[%rb], %rs　; メモリアドレス = 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、rsレジスタの内容（ワードデータ）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　imm13　; = imm26(25:13)
ext　imm13'　; = imm26(12:0)
ld.w　[%rb], %rs　; メモリアドレス = rb + imm26

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（ワードデータ）を、rbレジスタの内容に26ビット即値imm26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。

例

```
ext  0x10
ld.w [%r1],%r0     ; W[r1+0x10]←r0
```

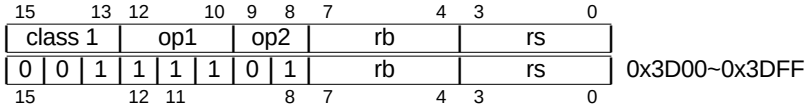
注　意　rbレジスタおよびディスプレースメントで指定されるメモリアドレスは、ワード境界（下位2ビット=0）を示していることが必要です。それ以外アドレスが指定されると、アドレス不整例外が発生します。
なお、データ転送は指定アドレスを下位8ビットとして1ワード（4アドレス）分行われます。

ld.w [%rb]+, %rs

機 能 **ワードデータ転送**

標準： **W[rb] ← rs, rb ← rb + 4**
拡張1: **不可**
拡張2: **不可**

コード



0x3D00~0x3DFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード **Src:** レジスタ直接（%rs = %r0~%r15）
Dst: ポストインクリメント付きレジスタ間接（%rb = %r0~%r15）

ＣＬＫ 1サイクル

説 明 rsレジスタの内容（ワードデータ）を指定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。データ転送後、rbレジスタ内のアドレスをインクリメント（+4）します。

例

```
ld.w [%r1]+,%r0    ; W[r1]←r0, r1←r1+4
```

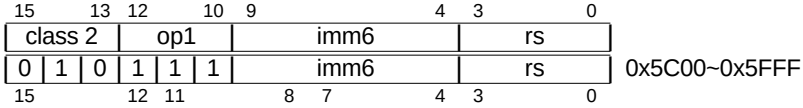
注 意 rbレジスタで指定されるメモリアドレスは、ワード境界（下位2ビット=0）を示していることが必要です。それ以外アドレスが指定されると、アドレス不整例外が発生します。
なお、データ転送は指定アドレスを下位8ビットとして1ワード（4アドレス）分行われます。

ld.w [%sp + imm6], %rs

機　能　**ワードデータ転送**

標準 :　**W[sp + imm6 x 4]** ← **rs**
拡張1:　**W[sp + imm19]** ← **rs**
拡張2:　**W[sp + imm32]** ← **rs**

コード



0x5C00~0x5FFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** レジスタ直接（%rs = %r0~%r15）
Dst: ディスプレースメント付きレジスタ間接

ＣＬＫ　1サイクル

説　明　**(1)標準**

ld.w　[%sp + imm6], %rs　; メモリアドレス = sp + imm6 x 4

rsレジスタの内容（ワードデータ）を指定メモリに転送します。現在のSPの内容に、6ビット即値imm6を4倍した値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。imm6は32ビット単位のワードアドレスを指定します。ディスプレースメントの下位2ビットは常に0となります。

(2)拡張1

ext　imm13　; = imm19(18:6)
ld.w　[%sp + imm6], %rs　; メモリアドレス = sp + imm19, imm6 = imm19(5:0)

ext命令により、ディスプレースメントが19ビットに拡張されます。これにより、rsレジスタの内容（ワードデータ）を、SPの内容に19ビット即値imm19を加えたアドレスに転送します。なお、imm6はワード境界（下位2ビット=0）を指定してください。

(3)拡張2

ext　imm13　; = imm32(31:19)
ext　imm13'　; = imm32(18:6)
ld.w　[%sp + imm6], %rs　; メモリアドレス = sp + imm32, imm6 = imm32(5:0)

2つのext命令により、ディスプレースメントが32ビットに拡張されます。これにより、rsレジスタの内容（ワードデータ）を、SPの内容に32ビット即値imm32を加えたアドレスに転送します。なお、imm6はワード境界（下位2ビット=0）を指定してください。

例

```
ext   0x1
ext   0x0
ld.w [%sp+0x4],%r1      ; H[SP+0x80004]← r1
```

注　意　ディスプレースメントを拡張する場合、imm6の下位2ビットは0として扱われ、常にワード境界をアドレスします。したがって、アドレス不整例外は発生しません。
なお、データ転送は指定アドレスを下位8ビットとして1ワード（4アドレス）分行われます。

mac %rs （オプション）

機 能 **積和演算**

標準： **"{ahr, alr}←{ahr, alr} + H[<rs+1>] × H[<rs+2>], <rs+1>←<rs+1> + 2, <rs+2>←<rs+2> + 2" × rs回**
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	
class 5			op1			op2		rs				–				
1	0	1	1	0	0	1	0	rs				0	0	0	0	0xB200~0xB2F0

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	↔	–	–	–	–	–	–

モード レジスタ直接（%rs = %r0~%r15）

CLK 2×N+4サイクル（N: rsレジスタに設定する繰り返し回数）

説 明 "mac %rs"命令は"{AHR, ALR} ← {AHR, ALR} + H[<rs+1>]+ × H[<rs+2>]+"（64ビット+16ビット×16ビット）をrsレジスタで指定される回数分実行します。
rsレジスタは回数カウンタとして使用され、演算ごとにデクリメントされます。rsレジスタが0になると、mac命令は終了します。したがって、2^{32}-1回までの繰り返しが可能です。rsレジスタに0を設定してmac命令を実行しても、積和演算は行われず、AHR、ALRレジスタも変更されません。rsレジスタも0のままデクリメントされません。
<rs+1>と<rs+2>はrsレジスタに続く2つの汎用レジスタです。
例: rsにR0レジスタを指定　　<rs+1>=R1レジスタ、<rs+2>=R2レジスタ
　rsにR15レジスタを指定　　<rs+1>=R0レジスタ、<rs+2>=R1レジスタ
積和演算は、これらのレジスタをベースアドレスとして指定されるメモリのハーフワードを符号付き16ビットデータとして演算します。ベースアドレスは1回の演算ごとにポストインクリメント（+2）されます。
演算結果は、AHRを上位32ビット、ALRを下位32ビットとする符号付き64ビットデータとして得られます。

積和演算中に演算結果が符号付き64ビットの範囲を越えると、オーバーフローとしてPSRのMOフラグが1にセットされます。この場合でも、rsレジスタに設定した回数を終了するまで演算は継続されます。MOフラグは、ソフトウェアによってリセットするまで1を保持します。mac命令の実行終了後にMOフラグを読み出すことで、演算結果が有効かどうかチェックできます。

mac命令の実行中は、繰り返しの途中であっても割り込みを受け付けます。割り込み処理ルーチンに分岐する際、スタックには実行中のmac命令のアドレスがリターンアドレスとしてセーブされます。したがって、割り込み処理ルーチンをreti命令で終了すると、中断していたmac命令の実行を再開します。ただし、その時点のrsレジスタの内容が残りのカウント数となりますので、割り込み処理ルーチン中でrsレジスタの内容が変更されると、当初設定した回数とは異なる結果となります。同様に、<rs+1>、<rs+2>レジスタの値が割り込み処理ルーチン中で変化すると、再開したmac命令は正しく実行されません。

例
```
mac   %r1              ; {ahr, alr}← {ahr,alr} + H[r2]+ × H[r3]+をr1回繰り返し実行
```

注 意
- <rs+1>および<rs+2>レジスタで指定されるメモリアドレスは、ハーフワード境界（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、アドレス不整例外が発生します。
- 本命令はオプションの乗除算回路を内蔵した機種に限り実行可能です。オプションを含まない機種ではnopとして処理されます。

mirror %rd, %rs

機 能 ミラー

標準 : **rd(31:24)← rs(24:31), rd(23:16)← rs(16:23), rd(15:8)← rs(8:15), rd(7:0)← rs(0:7)**
拡張1: **不可**
拡張2: **不可**

コード

15–13	12–10	9–8	7–4	3–0
class 4	op1	op2	rs	rd
1 0 0	1 0 1	1 0	rs	rd

0x9600~0x96FF

フラグ

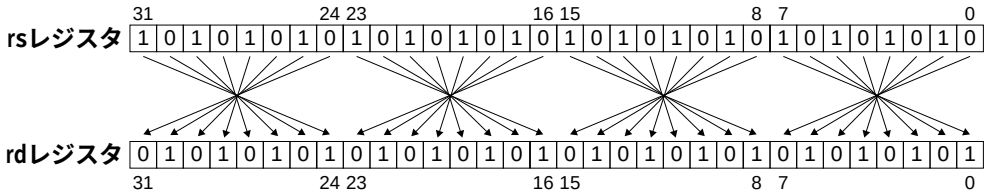
IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード **Src:** レジスタ直接(%rs = %r0~%r15)
Dst: レジスタ直接(%rd = %r0~%r15)

ＣＬＫ 1サイクル

説 明 **(1)標準**

rsレジスタのバイトデータごとにビットの上位と下位を入れ替え、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命令として実行されます。

例 r1が0x88442211の場合

```
mirror  %r0,%r1    ; r0 ← 0x11224488
```

32ビットデータのミラー(r1が0x44332211の場合)

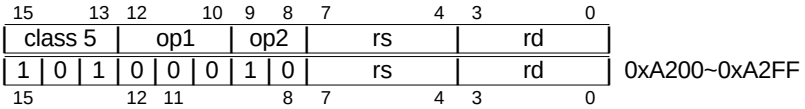
```
swap    %r1,%r1    ; r1 ← 0x11223344
mirror  %r1,%r1    ; r1 ← 0x8844CC22
```

mlt.h %rd, %rs （オプション）

機　能　**符号付き16ビット乗算**

標準：　**alr ← rd(15:0) × rs(15:0)**
拡張1:　**不可**
拡張2:　**不可**

コード



0xA200~0xA2FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

ＣＬＫ　1サイクル

説　明　**(1)標準**

rdレジスタの下位16ビットとrsレジスタの下位16ビットを符号付きで乗算し、32ビットの演算結果をALRにロードします。

(2)ディレイド命令

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命令として実行されます。

例

```
mlt.h %r0,%r1      ; alr ← r0(15:0)) × r1(15:0)  符号付き乗算
```

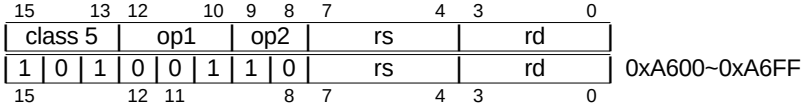
注　意　本命令はオプションの乗除算回路を内蔵した機種に限り実行可能です。オプションを含まない機種ではnopとして処理されます。

mltu.h %rd, %rs （オプション）

機 能 **符号なし16ビット乗算**

標準 : **alr ← rd(15:0) × rs(15:0)**
拡張1: **不可**
拡張2: **不可**

コード



0xA600~0xA6FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード **Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

ＣＬＫ 1サイクル

説 明 **(1)標準**

rdレジスタの下位16ビットとrsレジスタの下位16ビットを符号なしで乗算し、32ビットの演算結果をALRにロードします。

(2)ディレイド命令

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命令として実行されます。

例

```
mltu.h  %r0,%r1    ; alr ← r0(15:0)×r1(15:0)  符号なし乗算
```

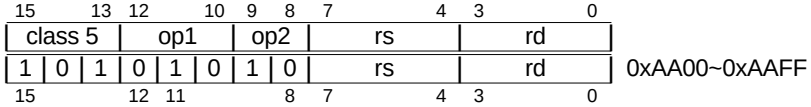
注 意 本命令はオプションの乗除算回路を内蔵した機種に限り実行可能です。オプションを含まない機種ではnopとして処理されます。

mlt.w %rd, %rs （オプション）

機 能 **符号付き32ビット乗算**

標準： **{ahr, alr}** ← **rd** × **rs**
拡張1: **不可**
拡張2: **不可**

コード



0xAA00~0xAAFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード **Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

ＣＬＫ 5サイクル

説 明 rdレジスタの32ビットデータとrsレジスタの32ビットデータを符号付きで乗算し、64ビットの演算結果をAHR（上位32ビット）、ALR（下位32ビット）にロードします。

例
```
mlt.w   %r0,%r1    ; {ahr, alr} ← r0×r1  符号付き乗算
```

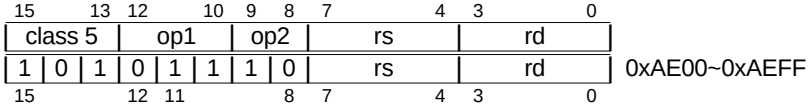
注 意 本命令はオプションの乗除算回路を内蔵した機種に限り実行可能です。オプションを含まない機種ではnopとして処理されます。

mltu.w %rd, %rs （オプション）

機　能　**符号なし32ビット乗算**

標準 :　**{ahr, alr}** ← **rd** × **rs**
拡張1:　**不可**
拡張2:　**不可**

コード



0xAE00~0xAEFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

ＣＬＫ　5サイクル

説　明　rdレジスタの32ビットデータとrsレジスタの32ビットデータを符号なしで乗算し、64ビットの演算結果をAHR（上位32ビット）、ALR（下位32ビット）にロードします。

例

```
mltu.w  %r0,%r1    ; {ahr, alr} ← r0×r1  符号なし乗算
```

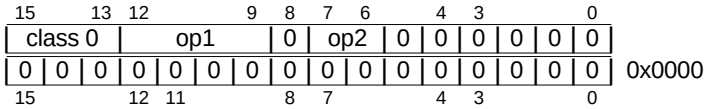
注　意　本命令はオプションの乗除算回路を内蔵した機種に限り実行可能です。オプションを含まない機種ではnopとして処理されます。

nop

機　能　**No Operation**

標準：　**なし**
拡張1:　**不可**
拡張2:　**不可**

コード



フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

ＣＬＫ　1サイクル

説　明　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。

例

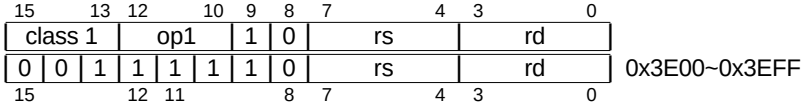
```
nop
nop          ; 2サイクルのウェイト
```

not %rd, %rs

機 能 **論理否定**

標準 : **rd** ← ! **rs**
拡張1: **不可**
拡張2: **不可**

コード



0x3E00~0x3EFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

モード **Src:** レジスタ直接(%rs = %r0~%r15)
Dst: レジスタ直接(%rd = %r0~%r15)

ＣＬＫ 1サイクル

説 明 **(1)標準**

rsレジスタの全ビットを反転しrdレジスタにロードします。

(2)ディレイド命令

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命令として実行されます。

例 r1レジスタ=0x55555555の場合

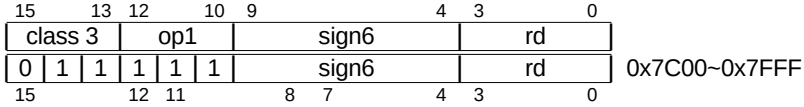
```
not   %r0,%r1       ; r0 = 0xAAAAAAAA
```

not %rd, sign6

機 能 **論理否定**

標準： **rd ← ! sign6**
拡張1: **rd ← ! sign19**
拡張2: **rd ← ! sign32**

コード



0x7C00~0x7FFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

モード **Src:** 即値（符号付き）
Dst: レジスタ直接（%rd = %r0~%r15）

CLK 1サイクル

説 明 **(1)標準**

not %rd, sign6 ; rd ← ! sign6

符号拡張した6ビット即値sign6の全ビットを反転し、結果をrdレジスタにロードします。

(2)拡張1

ext imm13 ; = sign19(18:6)
not %rd, sign6 ; rd ← ! sign19, sign6 = sign19(5:0)

符号拡張した19ビット即値sign19の全ビットを反転し、結果をrdレジスタにロードします。

(3)拡張2

ext imm13 ; = sign32(31:19)
ext imm13' ; = sign32(18:6)
not %rd, sign6 ; rd ← ! sign32, sign6 = sign32(5:0)

ext命令により拡張した32ビット即値sign32の全ビットを反転し、結果をrdレジスタにロードします。

(4)ディレイド命令

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

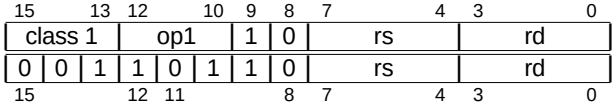
```
not  %r0,0x1f      ; r0 = 0xffffffe0

ext  0x7ff
not  %r1,0x3f      ; r1 = 0xfffe0000
```

or %rd, %rs

機　能　**論理和**

標準 :　**rd** ← **rd** | **rs**
拡張1:　**rd** ← **rs** | **imm13**
拡張2:　**rd** ← **rs** | **imm26**

コード



フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

モード　**Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

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。

(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。rsレジスタの内容は変更されません。

(3)拡張2

ext　imm13　; = imm26(25:13)
ext　imm13'　; = imm26(12:0)
or　%rd, %rs　; rd ← **rs | imm26**
rsレジスタの内容とゼロ拡張した26ビット即値imm26の論理和をとり、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)ディレイド命令

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

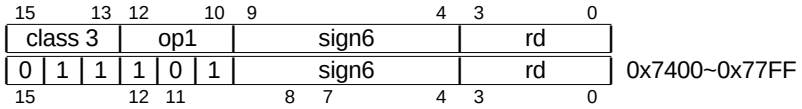
```
or   %r0,%r0       ; r0 = r0 | r0

ext  0x1
ext  0x1fff
or   %r1,%r2       ; r1 = r2 | 0x00003fff
```

or %rd, sign6

機 能 **論理和**

標準 : **rd ← rd | sign6**
拡張1: **rd ← rd | sign19**
拡張2: **rd ← rd | sign32**

コード



0x7400~0x77FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

モード **Src:** 即値(符号付き)
Dst: レジスタ直接(%rd = %r0~%r15)

CLK 1サイクル

説 明 **(1)標準**

```
or      %rd, sign6      ; rd ← rd | sign6
```

rdレジスタの内容と符号拡張した6ビット即値sign6の論理和をとり、結果をrdレジスタにロードします。

(2)拡張1

```
ext     imm13           ; = sign19(18:6)
or      %rd, sign6      ; rd ← rd | sign19, sign6 = sign19(5:0)
```

rdレジスタの内容と符号拡張した19ビット即値sign19の論理和をとり、結果をrdレジスタにロードします。

(3)拡張2

```
ext     imm13           ; = sign32(31:19)
ext     imm13'          ; = sign32(18:6)
or      %rd, sign6      ; rd ← rd | sign32, sign6 = sign32(5:0)
```

rdレジスタの内容とext命令により拡張した32ビット即値sign32の論理和をとり、結果をrdレジスタにロードします。

(4)ディレイド命令

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

```
or    %r0,0x3e      ; r0 = r0 | 0xfffffffe

ext   0x7ff
or    %r1,0x3f      ; r1 = r1 | 0x0001ffff
```

popn %rd

機 能 **ポップ**

標準： **rN ← W[sp], sp ← sp + 4, rN = r0～rdまで繰り返し**
拡張1: **不可**
拡張2: **不可**

コード

15 13	12 9	8	7 6	4	3 0
class 0	op1	0	op2	0	rd

15 12	11 8	7 4	3 0	
0 0 0 0 0 0 1 0	0 1 0 0		rd	0x0240~0x024F

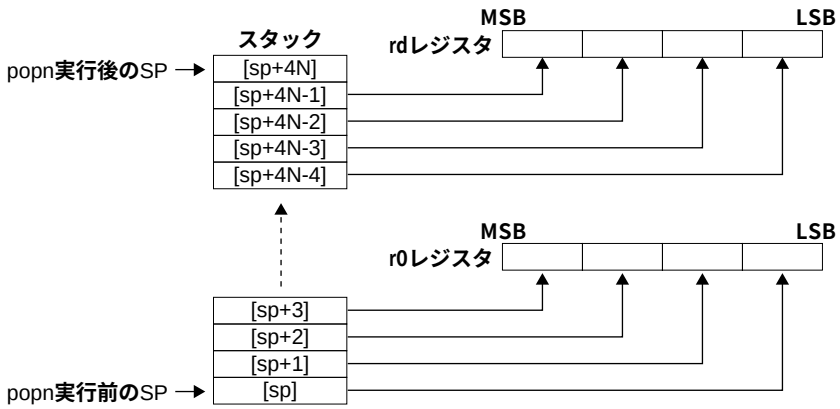
フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード レジスタ直接（%rd = %r0~%r15）

ＣＬＫ Nサイクル（N=復帰するレジスタ数）

説 明 pushn命令でスタックに待避させた汎用レジスタのデータを各レジスタに復帰させます。
popn命令は、最初に現在のSPが示すアドレスのワードデータをr0レジスタに復帰し、SPを1ワード（4バイト）分インクリメントします。この動作をrdレジスタまで連続して行います。rdは対応するpushn命令で指定したレジスタと同一である必要があります。



例

```
popn  %r3              ; r0, r1, r2, r3を復帰
```

pushn %rs

機 能 **プッシュ**

標準： **sp ← sp - 4, W[sp] ← rN, rN = rs～r0まで繰り返し**
拡張1: **不可**
拡張2: **不可**

コード

15	13	12	9	8	7	6	4	3	0	
class 0		op1		0	op2		0	rs		
0	0	0	0	0	0	1	0	0	0	0	0	rs		0x0200~0x020F
15			12	11			8	7			4	3	0	

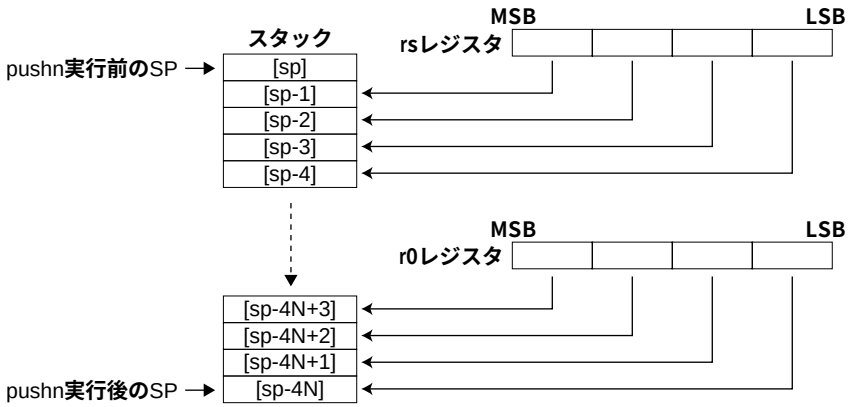
フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード レジスタ直接(%rd = %r0~%r15)

ＣＬＫ Nサイクル(N=待避させるレジスタ数)

説 明 汎用レジスタのデータをスタックに待避させます。
pushn命令は、最初に現在のSPを1ワード(4バイト)分デクリメントし、rsレジスタの内容をそのアドレスに待避させます。この動作をr0レジスタまで連続して行います。



例

```
pushn %r3          ; r3, r2, r1, r0を待避
```

ret / ret.d

機　能　**サブルーチンからのリターン**

標準 :　**pc ← W[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
class 0			op1				d	op2		0	0	–			
0	0	0	0	0	1	1	d	0	1	0	0	0	0	0	0

0x0640, 0x0740

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

ＣＬＫ　**ret:**　4サイクル
ret.d:　3サイクル

説　明　**(1)標準**

ret

call命令実行時にスタックに待避させた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1ワード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ことが必要です。

(2)ディレイド分岐(dビット=1)

ret.d

ret.d命令では命令コード中のdビットがセットされ、次の命令がディレイド命令となります。ディレイド命令はリターン前に実行されます。
ret.d命令と次のディレイド命令の間はトラップがマスクされ、割り込みや例外は発生しません。

例

```
ret.d
add  %r0,%r1 ; リターン前に実行されます。
```

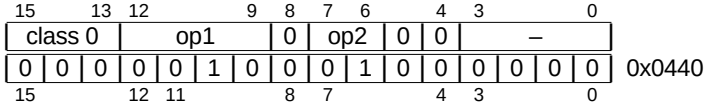
注　意　ret.d命令(ディレイド分岐)を使用する場合、次の命令はディレイド命令として使用可能な命令に限られます。それ以外の命令を実行した場合、動作は不定となりますので注意してください。使用可能な命令については、Appendixの命令一覧表を参照してください。

retd

機　能　**デバッグ処理ルーチンからのリターン**

標準：　**r0 ← W[0xC (or 0x6000C)], pc ← W[0x8 (or 0x60008)]**
拡張1:　**不可**
拡張2:　**不可**

コード



フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

ＣＬＫ　5サイクル

説　明　brk命令実行時にデバッグ用スタックに待避させたR0とPCの内容をそれぞれに戻して、デバッグ処理ルーチン(デバッグモード)からリターンします。
本命令はICE制御ソフト専用です。一般のプログラムでは使用しません。

例

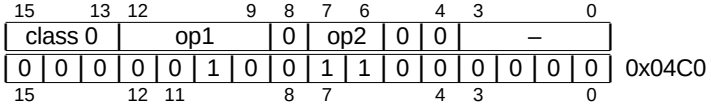
```
retd                    ; デバッグモードからのリターン
```

reti

機　能　**トラップ処理ルーチンからのリターン**

標準 :　**psr ← W[sp], sp ← sp + 4, pc ← W[sp], sp ← sp + 4**
拡張1:　**不可**
拡張2:　**不可**

コード



フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
↔	↔	↔	↔	↔	↔	↔	↔

ＣＬＫ　5サイクル

説　明　例外や割り込み発生時にスタックに待避させたPSRとPCの内容をそれぞれに戻して、トラップ処理ルーチンからリターンします。SPは2ワード分インクリメントされます。

例

```
reti                    ; トラップ処理ルーチンからのリターン
```

rl %rd, %rs

機 能 **左方向ローテート**

標準： **rdの内容をrsの指定ビット(0～8)分、左にローテート, LSB ← MSB**
拡張1: **不可**
拡張2: **不可**

コード

15 13	12 10	9 8	7 4	3 0
class 4	op1	op2	rs	rd

1	0	0	1	1	1	0	1	rs	rd

15 12 11 8 7 4 3 0

0x9D00~0x9DFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

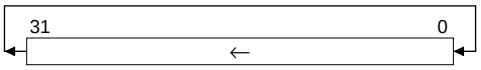
モード **Src:** レジスタ直接(%rs = %r0~%r15)
Dst: レジスタ直接(%rd = %r0~%r15)

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回転させます。ビットのシフト量はrsレジスタの下位4ビットによって0～8まで指定できます。

rdレジスタ



rs(3:0)	**1xxx**	**0111**	**0110**	**0101**	**0100**	**0011**	**0010**	**0001**	**0000**
シフト量	8	7	6	5	4	3	2	1	0

(2)ディレイド命令

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命令として実行されます。

例 r1レジスタ=0x55555555、r0レジスタ=1の場合

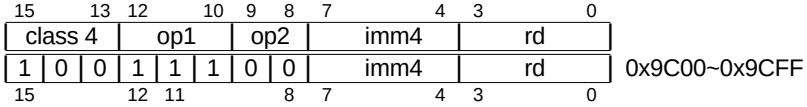
```
rl   %r1,%r0        ; r1 = 0xAAAAAAAA
```

rl %rd, imm4

機 能 **左方向ローテート**

標準 : **rdの内容をimm4の指定ビット(0～8)分、左にローテート, LSB ← MSB**
拡張1: **不可**
拡張2: **不可**

コード



0x9C00~0x9CFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

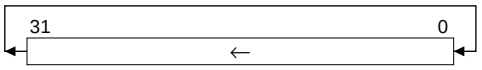
モード **Src:** 即値(符号なし)
Dst: レジスタ直接(%rd = %r0~%r15)

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回転させます。ビットのシフト量は4ビット即値imm4によって0～8まで指定できます。

rdレジスタ



imm4	**1xxx**	**0111**	**0110**	**0101**	**0100**	**0011**	**0010**	**0001**	**0000**
シフト量	8	7	6	5	4	3	2	1	0

(2)ディレイド命令

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命令として実行されます。

例 r1レジスタ=0x01010101の場合

```
rl   %r1,0x4       ; r1 = 0x10101010
```

rr %rd, %rs

機 能 **右方向ローテート**

標準： **rdの内容をrsの指定ビット(0～8)分、右にローテート, MSB ← LSB**
拡張1: **不可**
拡張2: **不可**

コード

15 13	12 10	9 8	7 4	3 0
class 4	op1	op2	rs	rd

1	0	0	1	1	0	0	1	rs	rd	0x9900~0x99FF
15 12			11 8				7 4	3 0		

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

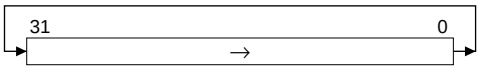
モード **Src:** レジスタ直接(%rs = %r0~%r15)
Dst: レジスタ直接(%rd = %r0~%r15)

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回転させます。ビットのシフト量はrsレジスタの下位4ビットによって0～8まで指定できます。

rdレジスタ



rs(3:0)	**1xxx**	**0111**	**0110**	**0101**	**0100**	**0011**	**0010**	**0001**	**0000**
シフト量	8	7	6	5	4	3	2	1	0

(2)ディレイド命令

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命令として実行されます。

例 r1レジスタ=0x55555555、r0レジスタ=1の場合

```
rr    %r1,%r0        ; r1 = 0xAAAAAAAA
```

rr %rd, imm4

機 能 **右方向ローテート**

標準 : **rdの内容をimm4の指定ビット(0～8)分、右にローテート, MSB ← LSB**
拡張1: **不可**
拡張2: **不可**

コード

15 13	12 10	9 8	7 4	3 0
class 4	op1	op2	imm4	rd

15	14	13	12	11	10	9	8	7 4	3 0	
1	0	0	1	1	0	0	0	imm4	rd	0x9800~0x98FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

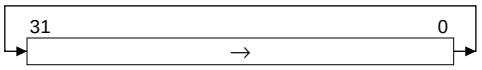
モード **Src:** 即値(符号なし)
Dst: レジスタ直接(%rd = %r0~%r15)

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回転させます。ビットのシフト量は4ビット即値imm4によって0～8まで指定できます。

rdレジスタ



imm4	**1xxx**	**0111**	**0110**	**0101**	**0100**	**0011**	**0010**	**0001**	**0000**
シフト量	8	7	6	5	4	3	2	1	0

(2)ディレイド命令

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命令として実行されます。

例 r1レジスタ=0x01010101の場合

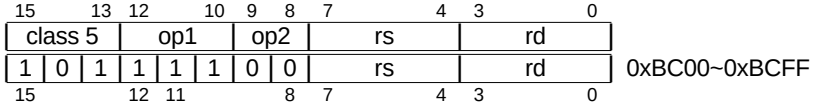
```
rr   %r1,0x4       ; r1 = 0x10101010
```

sbc %rd, %rs

機 能 **ボロー付き減算**

標準： **rd ← rd - rs - C**
拡張1: **不可**
拡張2: **不可**

コード



0xBC00~0xBCFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	↔	↔	↔	↔

モード **Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

CLK 1サイクル

説 明 **(1)標準**

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。

(2)ディレイド命令

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命令として実行されます。

例

```
sbc   %r0,%r1        ; r0 = r0 - r1 - C
```

64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}

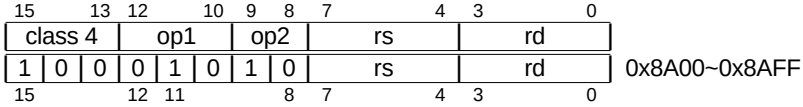
```
sub   %r1,%r3        ; 下位ワードの減算
sbc   %r2,%r4        ; 上位ワードの減算 {r2, r1} ← {r2, r1} - {r4, r3}
```

scan0 %rd, %rs

機　能　**0のビットスキャン**

標準 :　**rd ← rs(31:24)の0のビットオフセット**
拡張1:　**不可**
拡張2:　**不可**

コード



フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	↔	0	↔	0

モード　**Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

ＣＬＫ　1サイクル

説　明　**(1)標準**

rsレジスタの最上位バイト（ビット31～24）をスキャンし、0のビットが見つかると、その位置（MSBからのオフセット）をrdレジスタにロードします。MSBが0の場合、rdレジスタには0がロードされ、Zフラグがセットされます。rsレジスタの最上位バイト中に0が見つからなかった場合、rdレジスタには0x00000008がロードされ、Cフラグがセットされます。
rdレジスタのビット31～4はすべて0となります。

rsの上位8ビット	rdの下位8ビット	C	V	Z	N
0xxx xxxx	0000 0000	0	0	1	0
10xx xxxx	0000 0001	0	0	0	0
110x xxxx	0000 0010	0	0	0	0
1110 xxxx	0000 0011	0	0	0	0
1111 0xxx	0000 0100	0	0	0	0
1111 10xx	0000 0101	0	0	0	0
1111 110x	0000 0110	0	0	0	0
1111 1110	0000 0111	0	0	0	0
1111 1111	0000 1000	1	0	0	0

(2)ディレイド命令

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命令として実行されます。

例　32ビットデータのビットスキャン
r0 = テンポラリ, r1 = ビットスキャンソースデータ, r2 = ビットスキャン結果

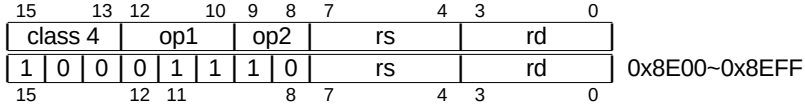
```
scan0   %r0,%r1     ; 1回目のビットスキャン
sll     %r1,%r0
ld.w    %r2,%r0
scan0   %r0,%r1     ; 2回目のビットスキャン
sll     %r1,%r0
add     %r2,%r0
scan0   %r0,%r1     ; 3回目のビットスキャン
sll     %r1,%r0
add     %r2,%r0
scan0   %r0,%r1     ; 4回目のビットスキャン
sll     %r1,%r0
add     %r2,%r0
```

scan1 %rd, %rs

機 能 **1のビットスキャン**

標準： **rd ← rs(31:24)の1のビットオフセット**
拡張1: **不可**
拡張2: **不可**

コード



0x8E00~0x8EFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	↔	0	↔	0

モード **Src:** レジスタ直接(%rs = %r0~%r15)
Dst: レジスタ直接(%rd = %r0~%r15)

CLK 1サイクル

説 明 **(1)標準**

rsレジスタの最上位バイト(ビット31～24)をスキャンし、1のビットが見つかると、その位置(MSBからのオフセット)をrdレジスタにロードします。MSBが1の場合、rdレジスタには0がロードされ、Zフラグがセットされます。rsレジスタの最上位バイト中に1が見つからなかった場合、rdレジスタには0x00000008がロードされ、Cフラグがセットされます。
rdレジスタのビット31～4はすべて0となります。

rsの上位8ビット	rdの下位8ビット	C	V	Z	N
1xxx xxxx	0000 0000	0	0	1	0
01xx xxxx	0000 0001	0	0	0	0
001x xxxx	0000 0010	0	0	0	0
0001 xxxx	0000 0011	0	0	0	0
0000 1xxx	0000 0100	0	0	0	0
0000 01xx	0000 0101	0	0	0	0
0000 001x	0000 0110	0	0	0	0
0000 0001	0000 0111	0	0	0	0
0000 0000	0000 1000	1	0	0	0

(2)ディレイド命令

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命令として実行されます。

例 32ビットデータのビットスキャン
r0 = テンポラリ, r1 = ビットスキャンソースデータ, r2 = ビットスキャン結果

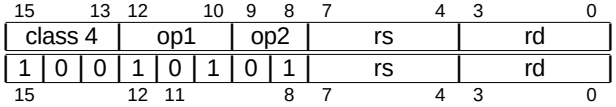
```
scan1   %r0,%r1     ; 1回目のビットスキャン
sll     %r1,%r0
ld.w    %r2,%r0
scan1   %r0,%r1     ; 2回目のビットスキャン
sll     %r1,%r0
add     %r2,%r0
scan1   %r0,%r1     ; 3回目のビットスキャン
sll     %r1,%r0
add     %r2,%r0
scan1   %r0,%r1     ; 4回目のビットスキャン
sll     %r1,%r0
add     %r2,%r0
```

sla %rd, %rs

機　能　**左方向算術シフト**

標準 :　**rdの内容をrsの指定ビット(0～8)分、左にシフト, LSB ← 0**
拡張1:　**不可**
拡張2:　**不可**

コード



0x9500~0x95FF

フラグ

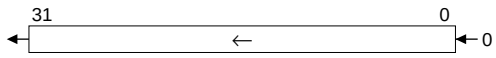
IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

モード　**Src:** レジスタ直接(%rs = %r0~%r15)
Dst: レジスタ直接(%rd = %r0~%r15)

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レジスタの下位4ビットによって0～8まで指定できます。最下位ビットには0が入ります。



rs(3:0)	**1xxx**	**0111**	**0110**	**0101**	**0100**	**0011**	**0010**	**0001**	**0000**
シフト量	8	7	6	5	4	3	2	1	0

(2)ディレイド命令

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命令として実行されます。

例　r1レジスタ=0x55555555、r0レジスタ=1の場合

```
sla   %r1,%r0       ; r1 = 0xAAAAAAAA
```

sla %rd, imm4

機 能 **左方向算術シフト**

標準： **rdの内容をimm4の指定ビット(0～8)分、左にシフト, LSB ← 0**
拡張1: **不可**
拡張2: **不可**

コード

15 13	12 10	9 8	7 4	3 0
class 4	op1	op2	imm4	rd

1	0	0	1	0	1	0	0	imm4	rd
15 12			11 8					7 4	3 0

0x9400~0x94FF

フラグ

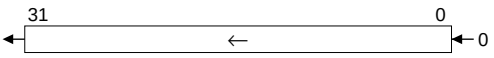
IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

モード **Src:** 即値(符号なし)
Dst: レジスタ直接(%rd = %r0~%r15)

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4ビット即値imm4によって0～8まで指定できます。最下位ビットには0が入ります。



rdレジスタ

imm4	**1xxx**	**0111**	**0110**	**0101**	**0100**	**0011**	**0010**	**0001**	**0000**
シフト量	8	7	6	5	4	3	2	1	0

(2)ディレイド命令

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命令として実行されます。

例 r1レジスタ=0x01010101の場合

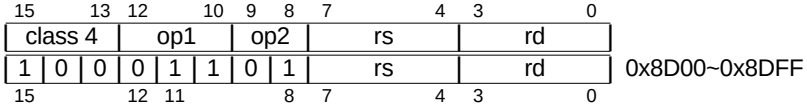
```
sla  %r1,0x4       ; r1 = 0x10101010
```

sll %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(0～8)分、左にシフト, LSB ← 0**
拡張1:　**不可**
拡張2:　**不可**

コード



0x8D00~0x8DFF

フラグ

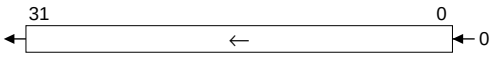
IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

モード　**Src:** レジスタ直接(%rs = %r0~%r15)
Dst: レジスタ直接(%rd = %r0~%r15)

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レジスタの下位4ビットによって0～8まで指定できます。最下位ビットには0が入ります。



rs(3:0)	**1xxx**	**0111**	**0110**	**0101**	**0100**	**0011**	**0010**	**0001**	**0000**
シフト量	8	7	6	5	4	3	2	1	0

(2)ディレイド命令

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命令として実行されます。

例　r1レジスタ=0x55555555、r0レジスタ=1の場合

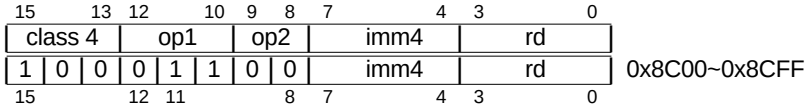
```
sll  %r1,%r0       ; r1 = 0xAAAAAAAA
```

sll %rd, imm4

機 能 **左方向論理シフト**

標準： **rdの内容をimm4の指定ビット(0～8)分、左にシフト, LSB ← 0**
拡張1: **不可**
拡張2: **不可**

コード



0x8C00~0x8CFF

フラグ

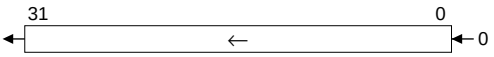
IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

モード **Src:** 即値(符号なし)
Dst: レジスタ直接(%rd = %r0~%r15)

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4ビット即値imm4によって0～8まで指定できます。最下位ビットには0が入ります。



rdレジスタ

imm4	**1xxx**	**0111**	**0110**	**0101**	**0100**	**0011**	**0010**	**0001**	**0000**
シフト量	8	7	6	5	4	3	2	1	0

(2)ディレイド命令

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命令として実行されます。

例 r1レジスタ=0x01010101の場合

```
sll  %r1,0x4       ; r1 = 0x10101010
```

slp

機 能 **SLEEP**

標準 : **CPU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	
class 0			op1				0	op2		0	0	–				
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0x0040

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

ＣＬＫ 1サイクル

説 明 CPUをSLEEPモードにします。
これにより、CPUおよびチップ上の周辺回路は動作を停止し、消費電流を大幅に抑えることができます。
SLEEPモードは割り込みによって解除され、その割り込み処理ルーチンを実行後、slp命令の次の命令位置に戻ります。

例

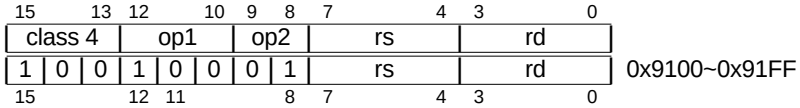
```
slp                    ; CPUをSLEEPモードに設定
```

sra %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(0～8)分、右にシフト, MSB ← MSB**
拡張1: **不可**
拡張2: **不可**

コード



0x9100~0x91FF

フラグ

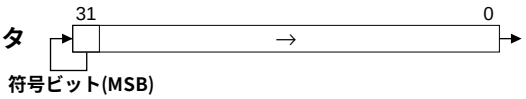
IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

モード **Src:** レジスタ直接(%rs = %r0~%r15)
Dst: レジスタ直接(%rd = %r0~%r15)

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レジスタの下位4ビットによって0～8まで指定できます。最上位ビットには符号ビットがコピーされます。



rs(3:0)	1xxx	0111	0110	0101	0100	0011	0010	0001	0000
シフト量	8	7	6	5	4	3	2	1	0

(2)ディレイド命令

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命令として実行されます。

例 r1レジスタ=0x55555555、r0レジスタ=1の場合

```
sra   %r1,%r0       ; r1 = 0x2AAAAAAA
```

sra %rd, imm4

機 能 **右方向算術シフト**

標準 : **rdの内容をimm4の指定ビット(0～8)分、右にシフト, MSB ← MSB**
拡張1: **不可**
拡張2: **不可**

コード

15 13	12 10	9 8	7 4	3 0
class 4	op1	op2	imm4	rd
1 0 0	1 0 0	0 0	imm4	rd

0x9000~0x90FF

フラグ

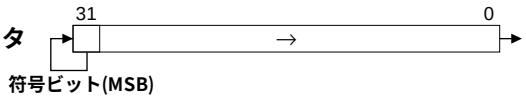
IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

モード **Src:** 即値(符号なし)
Dst: レジスタ直接(%rd = %r0~%r15)

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4ビット即値imm4によって0～8まで指定できます。最上位ビットには符号ビットがコピーされます。



imm4	**1xxx**	**0111**	**0110**	**0101**	**0100**	**0011**	**0010**	**0001**	**0000**
シフト量	8	7	6	5	4	3	2	1	0

(2)ディレイド命令

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命令として実行されます。

例 r1レジスタ=0x81010101の場合

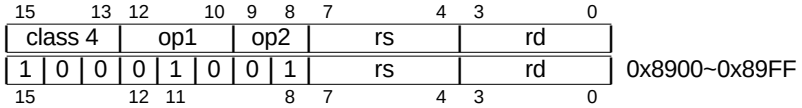
```
sra   %r1,0x4       ; r1 = 0xF8101010
```

srl %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(0～8)分、右にシフト, MSB ← 0**
拡張1: **不可**
拡張2: **不可**

コード



0x8900~0x89FF

フラグ

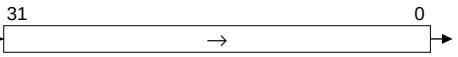
IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

モード **Src:** レジスタ直接(%rs = %r0~%r15)
Dst: レジスタ直接(%rd = %r0~%r15)

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レジスタの下位4ビットによって0～8まで指定できます。最上位ビットには0が入ります。



rs(3:0)	1xxx	0111	0110	0101	0100	0011	0010	0001	0000
シフト量	8	7	6	5	4	3	2	1	0

(2)ディレイド命令

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命令として実行されます。

例 r1レジスタ=0x55555555、r0レジスタ=1の場合

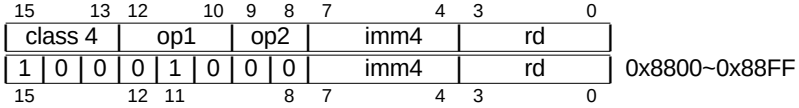
```
srl   %r1,%r0        ; r1 = 0x2AAAAAAA
```

srl %rd, imm4

機　能　**右方向論理シフト**

標準 :　**rdの内容をimm4の指定ビット(0〜8)分、右にシフト, MSB ← 0**
拡張1:　**不可**
拡張2:　**不可**

コード



0x8800~0x88FF

フラグ

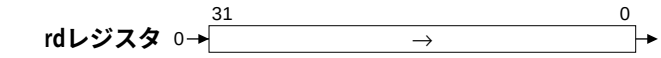
IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

モード　**Src:** 即値(符号なし)
Dst: レジスタ直接(%rd = %r0~%r15)

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4ビット即値imm4によって0〜8まで指定できます。最上位ビットには0が入ります。



imm4	**1xxx**	**0111**	**0110**	**0101**	**0100**	**0011**	**0010**	**0001**	**0000**
シフト量	8	7	6	5	4	3	2	1	0

(2)ディレイド命令

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命令として実行されます。

例　r1レジスタ=0x01010101の場合

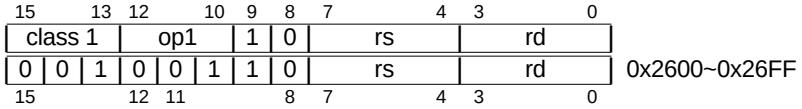
```
srl  %r1,0x4       ; r1 = 0x00101010
```

sub %rd, %rs

機 能 **減算**

標準： **rd ← rd - rs**
拡張1: **rd ← rs - imm13**
拡張2: **rd ← rs - imm26**

コード



0x2600~0x26FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	↔	↔	↔	↔

モード **Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

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。

(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 imm13 ; = imm26(25:13)
ext imm13' ; = imm26(12:0)
sub %rd, %rs ; rd ← rs - imm26

rsレジスタの内容から26ビット即値imm26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)ディレイド命令

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

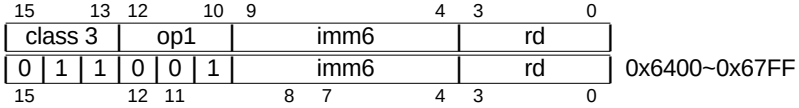
```
sub  %r0,%r0       ; r0 = r0 - r0

ext  0x1
ext  0x1fff
sub  %r1,%r2       ; r1 = r2 - 0x3fff
```

sub %rd, imm6

機 能 **減算**

標準 : **rd ← rd - imm6**
拡張1: **rd ← rd - imm19**
拡張2: **rd ← rd - imm32**

コード



0x6400~0x67FF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	↔	↔	↔	↔

モード **Src:** 即値(符号なし)
Dst: レジスタ直接(%rd = %r0~%r15)

ＣＬＫ 1サイクル

説 明 **(1)標準**

sub %rd, imm6 ; rd ← rd - imm6
rdレジスタから6ビット即値imm6を減算します。

(2)拡張1

ext imm13 ; = imm19(18:6)
sub %rd, imm6 ; rd ← rd - imm19, imm6 = imm19(5:0)
rdレジスタからext命令により拡張した19ビット即値imm19を減算します。

(3)拡張2

ext imm13 ; = imm32(31:19)
ext imm13' ; = imm32(18:6)
sub %rd, imm6 ; rd ← rd - imm32, imm6 = imm32(5:0)
rdレジスタからext命令により拡張した32ビット即値imm32を減算します。

(4)ディレイド命令

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

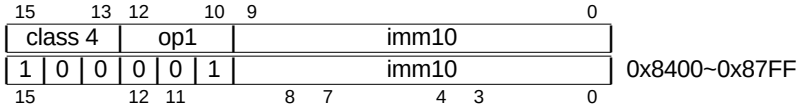
```
sub  %r0,0x3f      ; r0 = r0 - 0x3f

ext  0x1fff
ext  0x1fff
sub  %r1,0x3f      ; r1 = r1 - 0xffffffff
```

sub %sp, imm10

機 能 **減算**

標準： **sp ← sp - imm10 x 4**
拡張1: **不可**
拡張2: **不可**

コード



0x8400~0x87FF

フラグ

IL(3:0)	MO	DS	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**

10ビット即値imm10を4倍し、スタックポインタSPから減算します。

(2)ディレイド命令

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命令として実行されます。

例

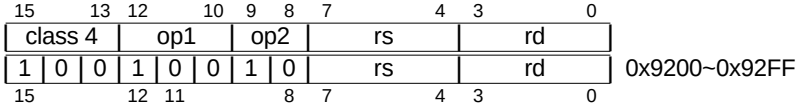
```
sub  %sp,0x1       ; sp = sp - 0x4
```

swap %rd, %rs

機　能　**スワップ**

標準 :　**rd(31:24)← rs(7:0), rd(23:16)← rs(15:8), rd(15:8)← rs(23:16), rd(7:0)← rs(31:24)**
拡張1:　**不可**
拡張2:　**不可**

コード



0x9200~0x92FF

フラグ

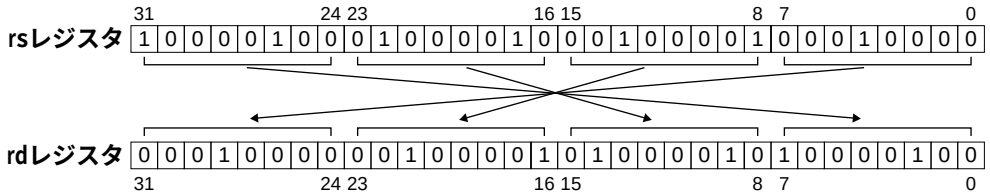
IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	–	–

モード　**Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

ＣＬＫ　1サイクル

説　明　**(1)標準**

rsレジスタのデータをバイト単位で上位と下位を入れ替え、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命令として実行されます。

例　r1が0x87654321の場合

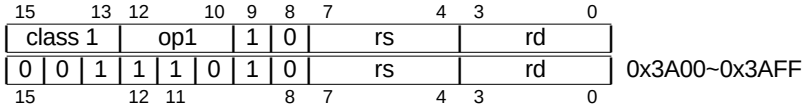
```
swap  %r0,%r1      ; r0 ← 0x21436587
```

xor %rd, %rs

機 能 **排他的論理和**

標準： **rd ← rd ^ rs**
拡張1: **rd ← rs ^ imm13**
拡張2: **rd ← rs ^ imm26**

コード



0x3A00~0x3AFF

フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

モード **Src:** レジスタ直接（%rs = %r0~%r15）
Dst: レジスタ直接（%rd = %r0~%r15）

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。

(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。rsレジスタの内容は変更されません。

(3)拡張2

ext imm13 ; = imm26(25:13)
ext imm13' ; = imm26(12:0)
xor %rd, %rs ; rd ← rs ^ imm26

rsレジスタの内容とゼロ拡張した26ビット即値imm26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。rsレジスタの内容は変更されません。

(4)ディレイド命令

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

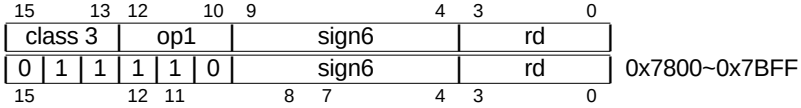
```
xor  %r0,%r0       ; r0 = r0 ^ r0

ext  0x1
ext  0x1fff
xor  %r1,%r2       ; r1 = r2 ^ 0x00003fff
```

xor %rd, sign6

機　能　**排他的論理和**

標準 :　**rd ← rd ^ sign6**
拡張1:　**rd ← rd ^ sign19**
拡張2:　**rd ← rd ^ sign32**

コード



フラグ

IL(3:0)	MO	DS	IE	C	V	Z	N
–	–	–	–	–	–	↔	↔

モード　**Src:** 即値(符号付き)
Dst: レジスタ直接(%rd = %r0~%r15)

ＣＬＫ　1サイクル

説　明　**(1)標準**

xor　%rd, sign6　; rd ← rd ^ sign6

rdレジスタの内容と符号拡張した6ビット即値sign6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。

(2)拡張1

```
ext     imm13          ; = sign19(18:6)
xor     %rd, sign6     ; rd ← rd ^ sign19, sign6 = sign19(5:0)
```

rdレジスタの内容と符号拡張した19ビット即値sign19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。

(3)拡張2

```
ext     imm13          ; = sign32(31:19)
ext     imm13'         ; = sign32(18:6)
xor     %rd, sign6     ; rd ← rd ^ sign32, sign6 = sign32(5:0)
```

rdレジスタの内容とext命令により拡張した32ビット即値sign32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。

(4)ディレイド命令

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

```
xor   %r0,0x3e      ; r0 = r0 ^ 0xfffffffe

ext   0x7ff
xor   %r1,0x3f      ; r1 = r1 ^ 0x0001ffff
```

Appendix

EPSON

CMOS 32-bit Single Chip Microcomputer

S1C33000 Quick Reference

メモリマップとトラップテーブル

メモリマップ

			サイズ
0xFFFFFFF	エリア18	外部メモリ	64MB
	エリア17	外部メモリ	64MB
	エリア16	外部メモリ	32MB
	エリア15	外部メモリ	32MB
	エリア14	外部メモリ	16MB
	エリア13	外部メモリ	16MB
	エリア12	外部メモリ	8MB
0x1000000	エリア11	外部メモリ	8MB
0x0C00000	エリア10	外部メモリ	4MB
	エリア9	外部メモリ	4MB
	エリア8	外部メモリ	2MB
	エリア7	外部メモリ	2MB
	エリア6	外部I/O	1MB
	エリア5	外部メモリ	1MB
0x0100000	エリア4	外部メモリ	1MB
0x0080000	エリア3	内蔵ROM	512KB
0x0060000	エリア2	Reserved	128KB
0x0040000	エリア1	内蔵I/O	128KB
0x0000000	エリア0	内蔵RAM	256KB

トラップテーブル

	ベクタアドレス
リセット	base + 0
Reserved	base + 4~12
ゼロ除算	base + 16
Reserved	base + 20
アドレス不整例外	base + 24
NMI	base + 28
Reserved	base + 32~44
ソフトウェア例外0	base + 48
:	:
ソフトウェア例外3	base + 60
マスク可能な外部割り込み0	base + 64
:	:
マスク可能な外部割り込み215	base + 924

base: トラップテーブル先頭アドレス
= 0x0080000 (内蔵ROMよりのブート時)
= 0x0C00000 (外部ROMよりのブート時)

レジスタ

汎用レジスタ (16)

31	0
R15	
R14	
R13	
:	
R4	
R3	
R2	
R1	
R0	

特殊レジスタ (5)

31	0	
PC		プログラムカウンタ
PSR		プロセッサステータスレジスタ
SP		スタックポインタ
ALR		算術演算ローレジスタ
AHR		算術演算ハイレジスタ

(AHR, ALR: 積和演算, 乗算, 除算用オプション)

PSR

31–12	11–8	7	6	5	4	3	2	1	0
Reserved	IL	MO	DS	–	IE	C	V	Z	N

IL: 割り込みレベル (0~15: 割り込み許可レベル)
MO: MACオーバーフローフラグ (1: オーバーフローあり, 0: なし)
DS: 被除数符号フラグ (1: 負, 0: 正)
IE: 割り込み許可 (1: 許可, 0: 禁止)
Z: ゼロフラグ (1: ゼロ, 0: ゼロ以外)
N: ネガティブフラグ (1: 負, 0: 正)
C: キャリーフラグ (1: キャリー/ボローあり, 0: なし)
V: オーバーフローフラグ (1: オーバーフローあり, 0: なし)

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~R15)、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~R15)、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~R15)、またはベースアドレス
%sd, sd: ディスティネーションとなる特殊レジスタ(PSR, SP, ALR, AHR)、またはレジスタの内容
%ss, ss: ソースとなる特殊レジスタ(PSR, SP, ALR, AHR)、またはレジスタの内容
%sp, sp: スタックポインタまたはスタックポインタの内容
＊ コード中のレジスタビットフィールドには指定されたレジスタの番号が入ります(R0~R15=0~15, PSR=0, SP=1, ALR=2, AHR=3)。

メモリ/アドレス/メモリデータ

[%rb]: レジスタ間接アドレッシング指定
[%rb]+: 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
[%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
B[rb]: rbレジスタ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
H[rb]: rbレジスタでベースアドレスが指定されるハーフワード領域、またはそこにストアされているハーフワードデータ
W[rb]: rbレジスタでベースアドレスが指定されるワード領域、またはそこにストアされているワードデータ
W[sp]: SPでベースアドレスが指定されるワード領域、またはそこにストアされているワードデータ
B[sp+imm6]: SPとディスプレースメントimm6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
H[sp+imm7]: SPとディスプレースメントimm6x2でベースアドレスが指定されるハーフワード領域、またはそこにストアされているハーフワードデータ
W[sp+imm8]: SPとディスプレースメントimm6x4でベースアドレスが指定されるワード領域、またはそこにストアされているワードデータ

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

フラグ

MO: MACオーバーフローフラグ
DS: 被除数符号フラグ
Z: ゼロフラグ
N: ネガティブフラグ
C: キャリーフラグ
V: オーバーフローフラグ
–: 変更なし
↔: セット(1)またはリセット(0)
0: リセット(0)

機能

←: 右側の内容が左側の項目にロード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
×: 乗算

サイクル

内蔵ROMの命令を実行し、内蔵RAMのみをアクセスする場合の実行サイクル数を示します。

EXT

–: ext命令による拡張ができないことを示します。

D

○: ディレイド命令として使用可能なことを示します。
–: ディレイド命令として使用できないことを示します。

データ転送

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			MO	DS	C	V	Z	N		
ld.b	%rd, %rs	1	0	1	0	0	0	0	1	rs				rd			rd(7:0)←rs(7:0), rd(31:8)←rs(7)	1	–	–	–	–	–	–	–	–
	%rd, [%rb]	0	0	1	0	0	0	0	0	rb				rd			rd(7:0)←B[rb], rd(31:8)←B[rb](7)	1–2 *4	–	–	–	–	–	–	*1	–
	%rd, [%rb]+	0	0	1	0	0	0	0	1	rb				rd			rd(7:0)←B[rb], rd(31:8)←B[rb](7), rb←rb+1	2	–	–	–	–	–	–	–	–
	%rd, [%sp+imm6]	0	1	0	0	0	0	imm6						rd			rd(7:0)←B[sp+imm6], rd(31:8)←B[sp+imm6](7)	1–2 *4	–	–	–	–	–	–	*2	–
	[%rb], %rs	0	0	1	1	0	1	0	0	rb				rs			B[rb]←rs(7:0)	1	–	–	–	–	–	–	*1	–
	[%rb]+, %rs	0	0	1	1	0	1	0	1	rb				rs			B[rb]←rs(7:0), rb←rb+1	1	–	–	–	–	–	–	–	–
	[%sp+imm6], %rs	0	1	0	1	0	1	imm6						rs			B[sp+imm6]←rs(7:0)	1	–	–	–	–	–	–	*2	–
ld.ub	%rd, %rs	1	0	1	0	0	1	0	1	rs				rd			rd(7:0)←rs(7:0), rd(31:8)←0	1	–	–	–	–	–	–	–	–
	%rd, [%rb]	0	0	1	0	0	1	0	0	rb				rd			rd(7:0)←B[rb], rd(31:8)←0	1–2 *4	–	–	–	–	–	–	*1	–
	%rd, [%rb]+	0	0	1	0	0	1	0	1	rb				rd			rd(7:0)←B[rb], rd(31:8)←0, rb←rb+1	2	–	–	–	–	–	–	–	–
	%rd, [%sp+imm6]	0	1	0	0	0	1	imm6						rd			rd(7:0)←B[sp+imm6], rd(31:8)←0	1–2 *4	–	–	–	–	–	–	*2	–
ld.h	%rd, %rs	1	0	1	0	1	0	0	1	rs				rd			rd(15:0)←rs(15:0), rd(31:16)←rs(15)	1	–	–	–	–	–	–	–	–
	%rd, [%rb]	0	0	1	0	1	0	0	0	rb				rd			rd(15:0)←H[rb], rd(31:16)←H[rb](15)	1–2 *4	–	–	–	–	–	–	*1	–
	%rd, [%rb]+	0	0	1	0	1	0	0	1	rb				rd			rd(15:0)←H[rb], rd(31:16)←H[rb](15), rb←rb+2	2	–	–	–	–	–	–	–	–
	%rd, [%sp+imm6]	0	1	0	0	1	0	imm6						rd			rd(15:0)←H[sp+imm7], rd(31:16)←H[sp+imm7](15); imm7={imm6, 0}	1–2 *4	–	–	–	–	–	–	*2	–
	[%rb], %rs	0	0	1	1	1	0	0	0	rb				rs			H[rb]←rs(15:0)	1	–	–	–	–	–	–	*1	–
	[%rb]+, %rs	0	0	1	1	1	0	0	1	rb				rs			H[rb]←rs(15:0), rb←rb+2	1	–	–	–	–	–	–	–	–
	[%sp+imm6], %rs	0	1	0	1	1	0	imm6						rs			H[sp+imm7]←rs(15:0); imm7={imm6, 0}	1	–	–	–	–	–	–	*2	–
ld.uh	%rd, %rs	1	0	1	0	1	1	0	1	rs				rd			rd(15:0)←rs(15:0), rd(31:16)←0	1	–	–	–	–	–	–	–	–
	%rd, [%rb]	0	0	1	0	1	1	0	0	rb				rd			rd(15:0)←H[rb], rd(31:16)←0	1–2 *4	–	–	–	–	–	–	*1	–
	%rd, [%rb]+	0	0	1	0	1	1	0	1	rb				rd			rd(15:0)←H[rb], rd(31:16)←0, rb←rb+2	2	–	–	–	–	–	–	–	–
	%rd, [%sp+imm6]	0	1	0	0	1	1	imm6						rd			rd(15:0)←H[sp+imm7], rd(31:16)←0; imm7={imm6, 0}	1–2 *4	–	–	–	–	–	–	*2	–
ld.w	%rd, %rs	0	0	1	0	1	1	1	0	rs				rd			rd←rs	1	–	–	–	–	–	–	–	○
	%sd, %rs	1	0	1	0	0	0	0	0	rs				sd			sd←rs	1	–	–	–	–	–	–	–	–
	%rd, %ss	1	0	1	0	0	1	0	0	ss				rd			rd←ss	1	–	–	–	–	–	–	–	–
	%rd, sign6	0	1	1	0	1	1	sign6						rd			rd(5:0)←sign6(5:0), rd(31:6)←sign6(5)	1	–	–	–	–	–	–	*3	
	%rd, [%rb]	0	0	1	1	0	0	0	0	rb				rd			rd←W[rb]	1–2 *4	–	–	–	–	–	–	*1	
	%rd, [%rb]+	0	0	1	1	0	0	0	1	rb				rd			rd←W[rb], rb←rb+4	2	–	–	–	–	–	–	–	
	%rd, [%sp+imm6]	0	1	0	1	0	0	imm6						rd			rd←W[sp+imm8]; imm8={imm6, 00}	1–2 *4	–	–	–	–	–	–	*2	
	[%rb], %rs	0	0	1	1	1	1	0	0	rb				rs			W[rb]←rs	1	–	–	–	–	–	–	*1	
	[%rb]+, %rs	0	0	1	1	1	1	0	1	rb				rs			W[rb]←rs, rb←rb+4	1	–	–	–	–	–	–	–	
	[%sp+imm6], %rs	0	1	0	1	1	1	imm6						rs			W[sp+imm8]←rs; imm8={imm6, 00}	1	–	–	–	–	–	–	*2	

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+imm26

*2) EXT 1**個使用**: **ベースアドレス** = sp+imm19, EXT 2**個使用**: **ベースアドレス** = sp+imm32
(転送データサイズにかかわらず、imm19 = {imm13, imm6}, imm32 = {imm13, imm13, imm6})

*3) EXT 1**個使用**: **データ** = sign19, EXT 2**個使用**: **データ** = sign32

4) "ld. %rd,[%rb]"**および**"ld.* %rd,[%sp+imm6]"**命令は通常1サイクルで実行されます。**
ただし、このrdレジスタを次の命令がソースレジスタ、ディスティネーションレジスタ、ベースレジスタとして使用している場合は2サイクルとなります。

論理演算

S1C33000 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			MO	DS	C	V	Z	N		
and	%rd, %rs	0	0	1	1	0	0	1	0	rs				rd				rd←rd & rs	1	–	–	–	–	↔	↔	∗1	○
	%rd, sign6	0	1	1	1	0	0	sign6						rd				rd←rd & sign6(**符号拡張**)	1	–	–	–	–	↔	↔	∗2	
or	%rd, %rs	0	0	1	1	0	1	1	0	rs				rd				rd←rd | rs	1	–	–	–	–	↔	↔	∗1	
	%rd, sign6	0	1	1	1	0	1	sign6						rd				rd←rd | sign6(**符号拡張**)	1	–	–	–	–	↔	↔	∗2	
xor	%rd, %rs	0	0	1	1	1	0	1	0	rs				rd				rd←rd ^ rs	1	–	–	–	–	↔	↔	∗1	
	%rd, sign6	0	1	1	1	1	0	sign6						rd				rd←rd ^ sign6(**符号拡張**)	1	–	–	–	–	↔	↔	∗2	
not	%rd, %rs	0	0	1	1	1	1	1	0	rs				rd				rd←!rs	1	–	–	–	–	↔	↔	–	
	%rd, sign6	0	1	1	1	1	1	sign6						rd				rd←!sign6(**符号拡張**)	1	–	–	–	–	↔	↔	∗2	

備　考

∗1) EXT 1**個使用**: rd←rs <op> imm13, EXT 2**個使用**: rd←rs <op> imm26　　∗2) EXT 1**個使用**: **データ** = sign19, EXT 2**個使用**: **データ** = sign32

算術演算

S1C33000 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			MO	DS	C	V	Z	N		
add	%rd, %rs	0	0	1	0	0	0	1	0	rs				rd				rd←rd + rs	1	–	–	↔	↔	↔	↔	∗1	○
	%rd, imm6	0	1	1	0	0	0	imm6						rd				rd←rd + imm6(**ゼロ拡張**)	1	–	–	↔	↔	↔	↔	∗2	○
	%sp, imm10	1	0	0	0	0	0	imm10										sp←sp + imm12(**ゼロ拡張**); imm12={imm10, 00}	1	–	–	–	–	–	–	–	○
adc	%rd, %rs	1	0	1	1	1	0	0	0	rs				rd				rd←rd + rs + C	1	–	–	↔	↔	↔	↔	–	○
sub	%rd, %rs	0	0	1	0	0	1	1	0	rs				rd				rd←rd - rs	1	–	–	↔	↔	↔	↔	∗1	○
	%rd, imm6	0	1	1	0	0	1	imm6						rd				rd←rd - imm6(**ゼロ拡張**)	1	–	–	↔	↔	↔	↔	∗2	○
	%sp, imm10	1	0	0	0	0	1	imm10										sp←sp - imm12(**ゼロ拡張**); imm12={imm10, 00}	1	–	–	–	–	–	–	–	○
sbc	%rd, %rs	1	0	1	1	1	1	0	0	rs				rd				rd←rd - rs - C	1	–	–	↔	↔	↔	↔	–	○
cmp	%rd, %rs	0	0	1	0	1	0	1	0	rs				rd				rd - rs	1	–	–	↔	↔	↔	↔	∗1	○
	%rd, sign6	0	1	1	0	1	0	sign6						rd				rd - sign6(**符号拡張**)	1	–	–	↔	↔	↔	↔	∗3	○
mlt.h	%rd, %rs	1	0	1	0	0	0	1	0	rs				rd				alr←rd(15:0) × rs(15:0); **符号付き乗算** (∗6)	1	–	–	–	–	–	–	–	○
mltu.h	%rd, %rs	1	0	1	0	0	1	1	0	rs				rd				alr←rd(15:0) × rs(15:0); **符号なし乗算** (∗6)	1	–	–	–	–	–	–	–	○
mlt.w	%rd, %rs	1	0	1	0	1	0	1	0	rs				rd				{ahr, alr}←rd × rs; **符号付き乗算** (∗6)	5	–	–	–	–	–	–	–	–
mltu.w	%rd, %rs	1	0	1	0	1	1	1	0	rs				rd				{ahr, alr}←rd × rs; **符号なし乗算** (∗6)	5	–	–	–	–	–	–	–	–
div0s	%rs	1	0	0	0	1	0	1	1	rs				0	0	0	0	**符号付き除算第1ステップ** (∗6); alr = **被除数**, rs = **除数**	1	–	↔	–	–	–	↔	–	–
div0u	%rs	1	0	0	0	1	1	1	1	rs				0	0	0	0	**符号なし除算第1ステップ** (∗6); alr = **被除数**, rs = **除数**	1	–	0	–	–	–	0	–	–
div1	%rs	1	0	0	1	0	0	1	1	rs				0	0	0	0	**ステップ除算** (∗4,∗6); alr←**商**, ahr←**余り** (**符号なし**)	1	–	–	–	–	–	–	–	–
div2s	%rs	1	0	0	1	0	1	1	1	rs				0	0	0	0	**符号付き除算データ補正1** (∗5,∗6)	1	–	–	–	–	–	–	–	–
div3s		1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	**符号付き除算データ補正2** (∗5,∗6); alr←**商**, ahr←**余り**	1	–	–	–	–	–	–	–	–

備　考

∗1) EXT 1**個使用**: rd←rs <op> imm13, EXT 2**個使用**: rd←rs <op> imm26

∗2) EXT 1**個使用**: **データ** = imm19, EXT 2**個使用**: **データ** = imm32　　∗3) EXT 1**個使用**: **データ** = sign19, EXT 2**個使用**: **データ** = sign32

∗4) 32**ビット** ÷ 32**ビットの場合、**div1**命令を32回実行する必要があります。符号なし除算の場合はdiv1命令によって除算結果がalrおよびahrにロードされます。**

∗5) div2s**命令と**div3s**命令は符号なし除算では不要です。**　　∗6) **オプションの乗除算回路を内蔵した機種に限り実行可能です。**

シフト&ローテート

S1C33000 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			MO	DS	C	V	Z	N		
srl	%rd, imm4	1	0	0	0	1	0	0	0	imm4				rd				**右方向論理シフト>>**imm4**ビット**; imm4=0–8, MSB**には0をセット**	1	–	–	–	–	↔	↔	–	○
	%rd, %rs	1	0	0	0	1	0	0	1	rs				rd				**右方向論理シフト>>**rs**ビット**; rs=0–8, MSB**には0をセット**	1	–	–	–	–	↔	↔	–	○
sll	%rd, imm4	1	0	0	0	1	1	0	0	imm4				rd				**左方向論理シフト>>**imm4**ビット**; imm4=0–8, LSB**には0をセット**	1	–	–	–	–	↔	↔	–	○
	%rd, %rs	1	0	0	0	1	1	0	1	rs				rd				**左方向論理シフト>>**rs**ビット**; rs=0–8, LSBLSB**には0をセット**	1	–	–	–	–	↔	↔	–	○
sra	%rd, imm4	1	0	0	1	0	0	0	0	imm4				rd				**右方向算術シフト>>**imm4**ビット**; imm4=0–8, MSB**には符号をコピー**	1	–	–	–	–	↔	↔	–	○
	%rd, %rs	1	0	0	1	0	0	0	1	rs				rd				**右方向算術シフト>>**rs**ビット**; rs=0–8, MSB**には符号をコピー**	1	–	–	–	–	↔	↔	–	○
sla	%rd, imm4	1	0	0	1	0	1	0	0	imm4				rd				**左方向算術シフト>>**imm4**ビット**; imm4=0–8, LSB**には0をセット**	1	–	–	–	–	↔	↔	–	○
	%rd, %rs	1	0	0	1	0	1	0	1	rs				rd				**左方向算術シフト>>**rs**ビット**; rs=0–8, LSB**には0をセット**	1	–	–	–	–	↔	↔	–	○
rr	%rd, imm4	1	0	0	1	1	0	0	0	imm4				rd				**右方向ローテート>>**imm4**ビット**; imm4=0–8, LSB → MSB	1	–	–	–	–	↔	↔	–	○
	%rd, %rs	1	0	0	1	1	0	0	1	rs				rd				**右方向ローテート>>**rs**ビット**; rs=0–8, LSB → MSB	1	–	–	–	–	↔	↔	–	○
rl	%rd, imm4	1	0	0	1	1	1	0	0	imm4				rd				**左方向ローテート>>**imm4**ビット**; imm4=0–8, MSB → LSB	1	–	–	–	–	↔	↔	–	○
	%rd, %rs	1	0	0	1	1	1	0	1	rs				rd				**左方向ローテート>>**rs**ビット**; rs=0–8, MSB → LSB	1	–	–	–	–	↔	↔	–	○

ビット操作

S1C33000 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			MO	DS	C	V	Z	N		
btst	[%rb], imm3	1	0	1	0	1	0	0	0	rb				0	imm3			B[rb](imm3)=0**ならば** Z**フラグ**←1	3	–	–	–	–	↔	–	∗1	–
bclr	[%rb], imm3	1	0	1	0	1	1	0	0	rb				0	imm3			B[rb](imm3)←0	3	–	–	–	–	–	–	∗1	–
bset	[%rb], imm3	1	0	1	1	0	0	0	0	rb				0	imm3			B[rb](imm3)←1	3	–	–	–	–	–	–	∗1	–
bnot	[%rb], imm3	1	0	1	1	0	1	0	0	rb				0	imm3			B[rb](imm3)←!B[rb](imm3)	3	–	–	–	–	–	–	∗1	–

備　考

∗1) EXT 1**個使用**: **アドレス** = rb+imm13, EXT 2**個使用**: **アドレス** = rb+imm26

即値拡張

S1C33000 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			MO	DS	C	V	Z	N		
ext	imm13	1	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ext命令を置くことができます。**

プッシュ&ポップ

S1C33000 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			MO	DS	C	V	Z	N		
pushn	%rs	0	0	0	0	0	0	1	0	0	0	0	0	rs				"sp←sp-4, W[sp]←rn"**を**rn=rs~r0**まで繰り返し**	1xn	–	–	–	–	–	–	–	–
popn	%rd	0	0	0	0	0	0	1	0	0	1	0	0	rd				"rn←W[sp], sp←sp+4"**を**rn=r0~rd**まで繰り返し**	1xn	–	–	–	–	–	–	–	–

分 岐

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			MO	DS	C	V	Z	N		
jrgt jrgt.d	sign8	0	0	0	0	1	0	0	d	sign8										!Z&!(N^V)**が真ならば** pc←pc+sign9; sign9={sign8, 0} (∗2)	1–2 ∗3, 1(.d)	–	–	–	–	–	–	∗1	–
jrge irge.d	sign8	0	0	0	0	1	0	1	d	sign8										!(N^V)**が真ならば** pc←pc+sign9; sign9={sign8, 0} (∗2)	1–2 ∗3, 1(.d)	–	–	–	–	–	–	∗1	–
jrlt jrlt.d	sign8	0	0	0	0	1	1	0	d	sign8										N^V**が真ならば** pc←pc+sign9; sign9={sign8, 0} (∗2)	1–2 ∗3, 1(.d)	–	–	–	–	–	–	∗1	–
jrle jrle.d	sign8	0	0	0	0	1	1	1	d	sign8										Z | (N^V)**が真ならば** pc←pc+sign9; sign9={sign8, 0} (∗2)	1–2 ∗3, 1(.d)	–	–	–	–	–	–	∗1	–
jrugt jrugt.d	sign8	0	0	0	1	0	0	0	d	sign8										!Z&!C**が真ならば** pc←pc+sign9; sign9={sign8, 0} (∗2)	1–2 ∗3, 1(.d)	–	–	–	–	–	–	∗1	–
jruge jruge.d	sign8	0	0	0	1	0	0	1	d	sign8										!C**が真ならば** pc←pc+sign9; sign9={sign8, 0} (∗2)	1–2 ∗3, 1(.d)	–	–	–	–	–	–	∗1	–
jrult jrult.d	sign8	0	0	0	1	0	1	0	d	sign8										C**が真ならば** pc←pc+sign9; sign9={sign8, 0} (∗2)	1–2 ∗3, 1(.d)	–	–	–	–	–	–	∗1	–
jrule jrule.d	sign8	0	0	0	1	0	1	1	d	sign8										Z | C**が真ならば** pc←pc+sign9; sign9={sign8, 0} (∗2)	1–2 ∗3, 1(.d)	–	–	–	–	–	–	∗1	–
jreq jreq.d	sign8	0	0	0	1	1	0	0	d	sign8										Z**が真ならば** pc←pc+sign9; sign9={sign8, 0} (∗2)	1–2 ∗3, 1(.d)	–	–	–	–	–	–	∗1	–
jrne jrne.d	sign8	0	0	0	1	1	0	1	d	sign8										!Z**が真ならば** pc←pc+sign9; sign9={sign8, 0} (∗2)	1–2 ∗3, 1(.d)	–	–	–	–	–	–	∗1	–
call	sign8	0	0	0	1	1	1	0	d	sign8										sp←sp-4, W[sp]←pc+2, pc←pc+sign9; sign9={sign8, 0} (∗2)	3,2(.d)	–	–	–	–	–	–	∗1	–
call.d	%rb	0	0	0	0	0	1	1	d	0	0	0	0	rb						sp←sp-4, W[sp]←pc+2, pc←rb (∗2)	3,2(.d)	–	–	–	–	–	–	–	–
jp	sign8	0	0	0	1	1	1	1	d	sign8										pc←pc+sign9; sign9={sign8, 0} (∗2)	2,1(.d)	–	–	–	–	–	–	∗1	–
jp.d	%rb	0	0	0	0	0	1	1	d	1	0	0	0	rb						pc←rb (∗2)	2,1(.d)	–	–	–	–	–	–	–	–
ret ret.d		0	0	0	0	0	1	1	d	0	1	0	0	0	0	0	0			pc←W[sp], sp←sp+4 (∗2)	4, 3(.d)	–	–	–	–	–	–	–	–
reti		0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0			psr←W[sp], sp←sp+4, pc←W[sp], sp←sp+4	5	↔	↔	↔	↔	↔	↔	–	–
retd		0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0			**デバッグ例外処理ルーチンからのリターン** (ICE**ソフトウェア用**)	5	–	–	–	–	–	–	–	–
int	imm2	0	0	0	0	0	1	0	0	1	0	0	0	0	0	imm2				sp←sp-4, W[sp]←pc+2, sp←sp-4, W[sp]←psr, pc←**ソフトウェア例外ベクタ**	10	–	–	–	–	–	–	–	–
brk		0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0			**デバッグ例外** (ICE**ソフトウェア用**)	10	–	–	–	–	–	–	–	–

備　考

∗1) EXT 1**個使用**: **ディスプレースメント** = sign22 (= {imm13, sign8, 0}), EXT 2**個使用**: **ディスプレースメント** = sign32 (= {1st imm13(12: 3), 2nd imm13, sign8, 0})

∗2) **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。**
ディレイド分岐命令を使用すると、分岐する前に直後の命令が実行されます。ディレイドコール(call.d)はpc+4**をリターンアドレスとしてスタックにセーブします。**

∗3) **ディレイド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1サイクル、分岐するときは2サイクルで実行されます。**

積和演算

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			MO	DS	C	V	Z	N		
mac	%rs	1	0	1	1	0	0	1	0	rs				0	0	0	0	"{ahr, alr}←{ahr, alr} + H[<rs+1>]+ × H[<rs+2>]+"をrs回繰り返し	2xn+4	↔	–	–	–	–	–	–	–

備　考

<rs+1>, <rs+2>: rsレジスタに続く2つのレジスタの内容 (例: rs=r0: <rs+1>=r1, <rs+2>=r2; rs=r15: <rs+1>=r0, <rs+2>=r1), 1回の演算ごとにポストインクリメント(+2)

mac命令はオプションの乗除算回路を内蔵した機種に限り実行可能です。

システム制御

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			MO	DS	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; pc←pc+2	1	–	–	–	–	–	–	–	–
halt		0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	Haltモードに設定	1	–	–	–	–	–	–	–	–
slp		0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	Sleepモードに設定	1	–	–	–	–	–	–	–	–

その他

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			MO	DS	C	V	Z	N		
scan0	%rd, %rs	1	0	0	0	1	0	1	0	rs				rd				rsの最上位バイト中の"0"ビットをスキャン, rd←MSBからのオフセット	1	–	–	↔	0	↔	0	–	○
scan1	%rd, %rs	1	0	0	0	1	1	1	0	rs				rd				rsの最上位バイト中の"1"ビットをスキャン, rd←MSBからのオフセット	1	–	–	↔	0	↔	0	–	○
swap	%rd, %rs	1	0	0	1	0	0	1	0	rs				rd				rd(31:24)←rs(7:0), rd(23:16)←rs(15:8), rd(15:8)←rs(23:16), rd(7:0)←rs(31:24)	1	–	–	–	–	–	–	–	○
mirror	%rd, %rs	1	0	0	1	0	1	1	0	rs				rd				rd(31:24)←rs(24:31), rd(23:16)←rs(16:23), rd(15:8)←rs(8:15), rd(7:0)←rs(0:7)	1	–	–	–	–	–	–	–	○

即値拡張（1）

分　類	ターゲット命令		ext命令1個による拡張 使用方法: ext imm13 ターゲット命令			ext命令2個による拡張 使用方法: ext imm13 ext imm13' ターゲット命令		
	オペコード	オペランド						
レジスタ間接 データ転送 (rbレジスタ使用)	ld.b	%rd, [%rb]	ld.b	%rd, [%rb+imm13]		ld.b	%rd, [%rb+imm26]	imm26={imm13,imm13'}
	ld.ub		ld.ub			ld.ub		
	ld.h		ld.h			ld.h		
	ld.uh		ld.uh			ld.uh		
	ld.w		ld.w			ld.w		
	ld.b	[%rb], %rs	ld.b	[%rb+imm13], %rs		ld.b	[%rb+imm26], %rs	imm26={imm13,imm13'}
	ld.h		ld.h			ld.h		
	ld.w		ld.w			ld.w		
ディスプレース メント付き レジスタ間接 データ転送 (SP使用)	ld.b	%rd, [%sp+imm6]	ld.b	%rd, [%sp+imm19]	imm19={imm13,imm6}	ld.b	%rd, [%sp+imm32]	imm32={imm13,imm13',imm6}
	ld.ub		ld.ub			ld.ub		
	ld.h		ld.h			ld.h		
	ld.uh		ld.uh			ld.uh		
	ld.w		ld.w			ld.w		
	ld.b	[%sp+imm6], %rs	ld.b	[%sp+imm19], %rs	imm19={imm13,imm6}	ld.b	[%sp+imm32], %rs	imm32={imm13,imm13',imm6}
	ld.h		ld.h			ld.h		
	ld.w		ld.w			ld.w		
即値ロード	ld.w	%rd, sign6	ld.w	%rd, sign19	sign19={1mm13, sign6}	ld.w	%rd, sign32	sign32={imm13,imm13',sign6}
レジスタ間 算術・論理演算	add	%rd, %rs	add	%rd, %rs, imm13	rd ← rs <op> imm13	add	%rd, %rs, imm26	rd ← rs <op> imm26 imm26={imm13,imm13'}
	sub		sub			sub		
	and		and			and		
	or		or			or		
	xor		xor			xor		
	cmp		cmp			cmp		
即値による 算術・論理演算	add	%rd, imm6	add	%rd, imm19	imm19={imm13,imm6}	add	%rd, imm32	imm32={imm13,imm13'imm6}
	sub		sub			sub		
	and	%rd, sign6	and	%rd, sign19	sign19={imm13,sign6}	and	%rd, sign32	sign32={imm13,imm13',sign6}
	or		or			or		
	xor		xor			xor		
	not		not			not		
	cmp		cmp			cmp		
ビット操作	btst	[%rb], imm3	btst	[%rb+imm13], imm3		btst	[%rb+imm26], imm3	imm26={imm13,imm13'}
	bset		bset			bset		
	bclr		bclr			bclr		
	bnot		bnot			bnot		

即値拡張（2）

分　類	ターゲット命令		ext命令1個による拡張 使用方法: ext imm13 ターゲット命令			ext命令2個による拡張 使用方法: ext imm13 ext imm13' ターゲット命令		
	オペコード	オペランド						
PC相対分岐	jrgt jrgt.d	sign8	jrgt jrgt.d	sign22	sign22={imm13,sign8,0}	jrgt jrgt.d	sign32	sign32={imm13(12:3),imm13',sign8,0}
	jrge irge.d		jrge irge.d			jrge irge.d		
	jrlt jrlt.d		jrlt jrlt.d			jrlt jrlt.d		
	jrle jrle.d		jrle jrle.d			jrle jrle.d		
	jrugt jrugt.d		jrugt jrugt.d			jrugt jrugt.d		
	jruge jruge.d		jruge jruge.d			jruge jruge.d		
	jrult jrult.d		jrult jrult.d			jrult jrult.d		
	jrule jrule.d		jrule jrule.d			jrule jrule.d		
	jreq jreq.d		jreq jreq.d			jreq jreq.d		
	jrne jrne.d		jrne jrne.d			jrne jrne.d		
	call call.d		call call.d			call call.d		
	jp jp.d		jp jp.d			jp jp.d		

命令索引

セイコーエプソン株式会社
電子デバイス営業本部

IC営業推進部　〒191-8501　東京都日野市日野421-8
IC営業技術G　TEL（042）587-5816（直通）　FAX（042）587-5624
〈東日本〉
ED東京営業部　〒191-8501　東京都日野市日野421-8
東京IC営業G　TEL（042）587-5313（直通）　FAX（042）587-5116
〈西日本〉
ED大阪営業部　〒541-0059　大阪市中央区博労町3-5-1　エプソン大阪ビル15F
TEL（06）6120-6000（代表）　FAX（06）6120-6100
〈東海・北陸〉
ED名古屋営業部　〒461-0005　名古屋市東区東桜1-10-24　栄大野ビル4F
TEL（052）953-8031（代表）　FAX（052）953-8041
〈長野〉
ED長野営業部　〒392-8502　長野県諏訪市大和3-3-5
TEL（0266）58-8171（直通）　FAX（0266）58-9917
〈東北〉
ED仙台営業所　〒980-0013　宮城県仙台市青葉区花京院1-1-20　花京院スクエア19F
TEL（022）263-7975（代表）　FAX（022）263-7990

インターネットによる電子デバイスのご紹介　http://www.epsondevice.com/domcfg.nsf

2001年3月改訂
1998年5月作成 ⓂA