第1章　はじめに

この章では、まず TX19 の特長について説明します。また、当社が提供している 900/L1 などの CISC プロセッサと比較して TX19 RISC アーキテクチャが具体的にどのような特長をもっているのか、ということについて説明します。

1.1　プロセッサの一般的特長

TX19 は Silicon Graphic 社 MIPS グループの 32 ビット RISC プロセッサ R3000A をベースにして当社が開発した TX39 に、高コード効率を追求した MIPS16 ASE を追加した、高性能 32 ビット RISC プロセッサです。TX19 の命令セットには、TX39 の 32 ビット命令がサブセットとして含まれています。そのため、TX19 は、TX39 に対しソフトウエアの上位互換性があります。

TX19 ファミリーは、TX19 プロセッサコア、内部バス、および特定の用途向けにさまざまな周辺回路を組み合わせて、ASSP として、またシステム ASIC のコアとして提供されてます。

■ **16 ビットと 32 ビット ISA モード**

◆ 16 ビット命令は MIPS16 ASE とオブジェクトレベルで互換性があります。

> **注意**:　MIPS16 ASE **の** 64 **ビットデータ処理の命令は** TX19 **にはありません。**

◆ 32 ビット命令は、演算性能の優れた TX39 とオブジェクトレベルで互換性があります。

- 16 ビット ISA モードと 32 ビット ISA モードが命令により切り換えられます。
- コプロセッサ命令と TLB 命令の一部を除き、R3000A に対し上位互換性があります。
- ハードウエアインタロック機構により、ロード命令直後のにロードされたレジスタの内容を参照する命令を配置できるため、NOP 命令を挿入する必要がありません。
- 分岐ライクリ命令により分岐先で実行する命令を、分岐命令の直後に配置できるため、NOP 命令を挿入する必要がありません。

■ **高性能**

◆ ほとんどの命令を、1 クロックサイクルで実行します。

◆ 3 オペランドの演算命令

- ◆ 内部 32 ビット構成
  32 ビット汎用レジスタと 32 ビットプログラムカウンタなど
- ◆ 5 段パイプライン
- ◆ アクセス時間が 1 クロックサイクルの高速メモリを、命令用とデータ用に独立して内蔵可能
- ◆ ライトバッファ内蔵可能
  命令キャッシュとデータキャッシュを独立して内蔵可能
- ◆ ハーバードアーキテクチャを採用
  TX19 は、命令とデータ(オペランド)用に別々のバスを使用します。TX19 は、プロセッサコアにデータの出し入れをするデータバス、データ用のアドレスバス、オペコードを運ぶバス、オペコード用のアドレスの 4 本のバスをもっています。別々のバスで命令とデータを同時にアクセスできるので、命令のスループットが高くなります。
- ◆ ノンブロッキングロード機能
  外部メモリからのロードで大きな遅延を発生した場合でも、ロード遅延スロット中の後続命令を実行できます。
- ◆ 積和演算器 (MAC) を内蔵
  32 ビット×32 ビット＋64 ビットの演算を 1 クロックで実行します。
- ◆ 4G バイトの仮想アドレス空間
- ◆ 4 チャネルのコプロセッサを接続可能
  システム構成、例外処理、メモリ管理などをつかさどるシステム制御コプロセッサ (CP0) を内蔵しています。

■ 低消費電力

- ◆ 消費電力を抑えた最適化設計
- ◆ 動作周波数をプログラムにより変更可能
  fc/2、fc/4、fc/8 から選択できます。 (fc = 原振)
- ◆ プログラムにより設定できる低消費電力モード (HALT モード・DOZE モード)
  外部マスタからのバス制御権の要求に応答できる DOZE モードと、応答しない HALT モードがあります。

■ **リアルタイム制御に向いた高速割り込み応答**

- ◆ 各割り込みサービスルーチンのエントリーアドレスを独立化
- ◆ 各割り込みソースに対するベクタアドレスを自動生成
  例外ベクタを読み出すとき優先順位を判定し、割り込み要求レベルが現在の割り込みマスクレベルより大きい場合のみ、割り込み例外を発生します。これにより、高位の割り込み要求に対する保留を最短にします。
- ◆ 割り込みマスクレベルの自動更新

■ **システム** ASIC **用マイコンコア**

- ◆ ASIC と同一の製品プロセス、開発環境
- ◆ コンパクトなコア
- ◆ TX シリーズの標準バスである G-Bus に直接接続可能

■ **システム開発環境**

- ◆ 言語ツール　C コンパイラ、アセンブラなど
  当社とサードパーティのツールを整備
- ◆ リアルタイムオペレーションシステム
  当社(μITRON)とサードパーティのリアルタイムオペレーションシステムを整備
- ◆ テバック支援システム
  - ソースレベルデバック環境を提供するリアルタイムエミュレータとして、当社とサードパーティのツールを整備
  - デバック支援回路 (DSU) を ASIC に挿入する簡易ツールをサポート

1.2　RISC とは?

1980 年代の初めまでは、すべての CPU は複雑命令セットコンピュータ (CISC) という方式をとっており、既存のソフトウエアとの互換性を維持するために、新しい種類の命令や、より複雑な演算を可能にするための機能を追加しながら進化してきました。通常、CISC はあらゆる状況を想定した何百もの命令を実現した CPU を意味します。何百もの命令を解読・実行することのできる CPU は、当然のことながら回路が非常に複雑になり、設計にも時間、コストがかかります。

ところが、1980 年代の初めになると、プロセッサの資源がソフトウエアにより実際にどのように使用されているかという統計的な分析にもとづいて、新しい概念のコンピュータが提唱されるようになります。これを縮小命令セットコンピュータ (RISC) といいます。RISC の最大の特長は、プログラマやコンパイラによりあまり使用されることのない複雑な命令をなくし、命令セットを簡素化したことです。

■　特長 1

RISC プロセッサは複雑な命令をなくし、基本的な命令に絞り込んでいます。例えば、ブロック転送、ブロックサーチ、ビットスキャンなどの複雑な命令はありません。

また、RISC プロセッサはロード・ストアアーキテクチャを採用しています。CISC プロセッサでは、メモリ中のデータをオペランドとして直接指定できる命令があります。例えば、当社 900/L1 では、ADD A, (1000H) という命令はメモリの 1000H 番地のデータを CPU に取り込んで、A レジスタの内容と加算し、その結果を再び A レジスタに書き戻します。RISC ではこのような命令はなく、メモリに対する操作には、メモリ中のデータを CPU レジスタにロードするか、または CPU レジスタ中のデータをメモリにストアする命令しかありません。すなわち、すべての演算は CPU レジスタ中のオペランドを対象とします。

CISC プロセッサには非常に多くの命令があり、さらに各命令でいくつものアドレッシングモードをサポートしているため、命令を実行するのにマイクロコードが使用されます。RISC に比べるとプログラミングが容易で、コードサイズが縮小されるという利点である一方で、マイクロコードの実現にチップのかなりの面積が割かれるため、プロセッサの性能を改良する上での障害になっています。

■　特長 2

RISC プロセッサの命令は固定長です。一方、CISC プロセッサには、1 バイト命令、2 バイト命令などさまざまな長さの命令があり、なかには 7 バイト命令もあります。このように命令長がばらばらであると、入ってくる命令の長さが分からないので、命令デコーダは極めて複雑になります。これに対して、例えば、TX19 の 32 ビット ISA では、命令長はすべて 32 ビットに固定されています。固定命令長により、命令のデコードが高速になります。

■ **特長 3**

RISC には少数の単純な命令しかないので、大部分の命令は 1 クロックサイクルで実行できます。そのため、パイプライン中の各命令が異なったクロックサイクルを必要とする CISC と比べると、RISC はパイプライン化が簡単です。通常、RISC プロセッサは多段パイプラインを備えています。

1.3　TX19 の特徴

前項では、一般的な RISC プロセッサと CISC プロセッサの違いを説明しました。この項では、TX19 の命令セットアーキテクチャ (ISA)の特徴について、当社の 8 ビット CISC プロセッサ 870/X と 16 ビット CISC プロセッサ 900/L1 を比較しながら説明します。

TX19 には 16 ビットと 32 ビットの 2 つの ISA モードがあります。16 ビット ISA モードと 32 ビット ISA モードは、プログラムの実行中にサブルーサン単位で、命令により切り換えられます。16 ビット ISA (MIPS16) は独立した命令セットではなく、32 ビット MIPS アーキテクチャを拡張したものです。32 ビット ISA には 85 の命令があり、16 ビット ISA には 53 の命令があります。一般的にコードサイズを縮小したい部分は 16 ビット ISA を用い、高速化したい部分は 32 ビット ISA を用います。

一方、870/X や 900/L1 には約 1000 個の命令と多くのアドレッシングモードがあります。一般的に CISC プロセッサはコード効率の点で優れています。

1.3.1　命令セットアーキテクチャ

TX19 の命令セットアーキテクチャの特徴を以下に示します。

◆ **複雑な処理を行う命令がない**

TX19 にはロード、ストア、加算、減算、乗算、除算、AND、OR、XOR、シフト、ジャンプ、分岐などの基本的な命令しかありません。900/L1 にある LDIR (ブロック転送) 、CPIR (ブロック検索) などの複雑な命令はありません。これらの複雑な処理を行う命令を CISC プロセッサがハードウエアで実行するのに対して、TX19 では基本的な命令を組み合わせてソフトウエアルーチンを作成しなければなりません。これはコンパイラ(またはプログラマ)の役目になります。ただし、例外として、高速処理が要求される積和演算命令 (MADD、MADDU) は命令セットの中に用意されています。 (これらの命令は専用の MAC ユニットで実行されます。)

◆ **他の命令で代替できる命令はない**

命令数を減らすために、TX19 は他の命令を使って実行できる命令は除いてあります。例えば、TX19 には、NOP (No Operation) 命令、INC (Increment) 命令、DEC (Decrement) 命令がありません。例えば、TX19 では、NOP 命令と同様の処理は、以下のようにシフト命令を代用します。

```
SLL  r0,r0,0
```

TX19 では、レジスタ r0 はハードウエア的に 0 に固定されています。上記の命令は実際には、レジスタ r0 の内容を 0 ビットシフトし、その結果をレジスタ r0 に格納します。(アセンブラではプログラムを分かりやすくするために疑似命令として NOP を許可しています。)

また、レジスタのインクリメントは以下のように即値加算命令 ADDIU (Add Immediate Unsigned) を用いて実行します。

```
ADDIU  rt,rs,1
```

ここで *rt* と *rs* はそれぞれターゲットレジスタとソースレジスタです。同様にレジスタのデクリメントは以下のように実行します。

```
ADDIU  rt,rs,-1
```

◆ **複数の単純な命令で実現できる命令はない**

TX19 は、2 つ以上の単純な命令から実現できる命令は切り捨て、命令数を減らしています。例えば、TX19 にはスタックに対するポップ命令、プッシュ命令はありません。CISC プロセッサでは、プッシュ命令を実行すると、レジスタの内容がスタックに退避され、スタックポインタレジスタがオペランドサイズ分デクリメントされます。TX19 では、慣例として 32 個ある汎用レジスタのうちの 1 つをスタックポインタレジスタとして使い、プッシュをこのレジスタに対する加算命令とストア命令で実行します。

◆ **ロード・ストアアーキテクチャを採用**

870/X や 900/L1 のような CISC プロセッサでは、ADD A, (1000H) のように演算命令でもメモリをアクセスできます。これに対して、TX19 では、メモリへのアクセスは、ロード・ストア命令により、メモリと CPU の汎用レジスタとの間でデータを動かす命令に限定しています。

◆ **メモリのアドレッシングモードを限定**

900/L1 や 870/X にはメモリをアクセスためのアドレッシングモードが 7 種類以上あります。例えば、レジスタ間接モード、オートインクリメント付きレジスタ間接モード、インデックス相対モード、ベースドインデックス相対モードなどです。これらのアドレッシングモードは、アセンブリ言語のプログラマには便利で、コードサイズを小さくするのに役立ちます。

これに対して、TX19 には、32 ビット ISA モードでは、メモリのアドレッシングは、ベースド相対(オフセット付きレジスタ間接)の 1 種類しかありません。また、16 ビット ISA モードでも、これに加えて PC 相対と SP 相対の 3 種類のアドレッシングモードしかありません(PC 相対モードと SP 相対モードを使えるのは 3 命令に限定されています)。このように、アドレッシングモードが数少ないため、TX19 は 900/L1 や 870/X などの CISC プロセッサに比べると、回路が単純になります。

◆ **3 オペランドの演算命令を採用**

TX19 では、2 つのソースレジスタと 1 つのデスティネーションレジスタを別々に指定できる 3 オペランドの命令があります。例を以下に示します。

```
ADD  rd,rs1,rs2
```

この命令は *rs1* と *rs2* の 2 つのソースレジスタの内容を加算し、その結果をデスティネーションレジスタ *rd* に格納します。これに対して、900/L1 では、以下のように演算命令は 2 つのオペランドしかとれません。この命令は、XWA と XBC の内容を加算して、その結果を XWA に書き戻します。

```
ADD  XWA,XBC
```

◆ **フラグレジスタがない**

TX19 には、キャリー、オーバーフロー、サインなどの演算フラグがありません。例えば、900/L1 ではキャリーフラグは加算、減算でキャリー、ボローが発生したことを記録するのに使われます。キャリーフラグは、多桁の数値を加算する際によく使われます。900/L1 には、そのためにキャリーフラグすなわち、桁上がりのビットを 2 つのレジスタの内容に加算する ADC 命令が用意されています。

一方、TX19 では、32 ビットの演算が 1 つの命令で実行できるので、フラグビットを必要とすることはあまりありません。ただし、キャリーが発生する可能性のある大きな数値の加算をするときには、最初に加算の結果キャリーが発生した場合はそれを汎用レジスタに記録しておきます。したがって、複数ワードサイズ以上の加算をするときには、キャリーが発生する場合のコードと、キャリーが発生しない場合のコードの 2 つのコードが必要になります。

また、900/L1 の比較命令 CP(Compare)命令は、減算でボローが発生したかどうかを示すのにキャリーフラグを使用します。TX19 では、SLT(Set On Less Than)のような比較命令の結果は、汎用レジスタに格納します。

1.3.2　命令フォーマット

TX19 には、16 ビットと 32 ビットの 2 つの ISA モードがあります。32 ビット ISA モードの命令は、すべて 32 ビットの固定長です。16 ビット ISA モードの命令は、一部の例外を除き 16 の固定長です。

各 16 ビット命令は、32 ビット命令に対応しており、単純な伸長回路により、32 ビット命令に伸長されたのち、通常の命令デコーダに送り込まれます。

870/X には、1 バイト命令から 6 バイト命令まであり、命令長は固定されていません。900/L1 は 7 バイト命令まであります。したがって、コードサイズが小さくなるという利点がある反面、入ってくる命令のサイズが分からないため、命令デコーダは複雑になり、処理が遅くなります。

1.3.3　パイプライン処理

TX19 は 5 段パイプラインで命令を処理します。5 段パイプラインは、各命令の実行を 5 段に分けて、5 つの命令を同時に処理します。各ステージは 1 クロックで動作します。

TX19 の大きな特長は、ほとんどの命令は同じクロック数で実行されることです。そのため、TX19 は比較的パイプライン化するのが簡単で、1 命令当たりほぼ 1 クロックで実行できます。

CISC プロセッサのように異なる命令長で命令が構成されている場合、パイプラインの管理が複雑になります。また、コンパイラによるスケジューリングでも、パイプラインのストールをなくすことが難しくなります。例えば、870/X は 3 段パイプラインを備えていますが、命令によって実行時間が大きく違い、4 サイクル～60 サイクルかかります。900/L1 も 3 段階パイプラインをもっていますが、命令の実行時間は 2 サイクル～27 サイクルとばらばらです。

第2章　CPU アーキテクチャ

この章ではデータ形式、プログラミングモデル、ISA モード、コプロセッサ、パイプライン、メモリ管理など、TX19 のアーキテクチャの概要について説明します。

2.1　データ形式

この項ではレジスタおよびメモリにおけるデータの形についてと、オペランドの符号拡張、ゼロ拡張について説明します。

2.1.1　バイト順序

TX19 は 8 ビット、16 ビット、32 ビット、64 ビットのデータ形式をサポートしています。1 バイトは 8 ビット、ハーフワードは 2 バイト(16 ビット)、1 ワードは 4 バイト(32 ビット)、ダブルワードは 2 ワード(64 ビット)と定義されています。

データ形式がダブルワード、ワード、ハーフワードの場合、バイト順序として、ビッグエンディアン方式とリトルエンディアン方式をサポートしています。バイト順序はリセット時に入力端子 ENDIAN の状態により設定できます (派生品によっては、どちらかのエンディアンに固定されています)。

図 2-1 にビッグエンディアン方式とリトルエンディアン方式のバイト順序を示します。TX19 はバイトアドレッシング方式を用いています。ビッグエンディアン方式では、最上位(左端)のバイトから下位のアドレスを割り当てられます。リトルエンディアン方式では、最下位(右端)のバイトから下位のアドレスが割り当てられます。リトルエンディアン方式では、同じ整数値あれば、データ形式がハーフワードなのかワードなどかに関係なく、各アドレスには同じ値が入るという特徴があります。

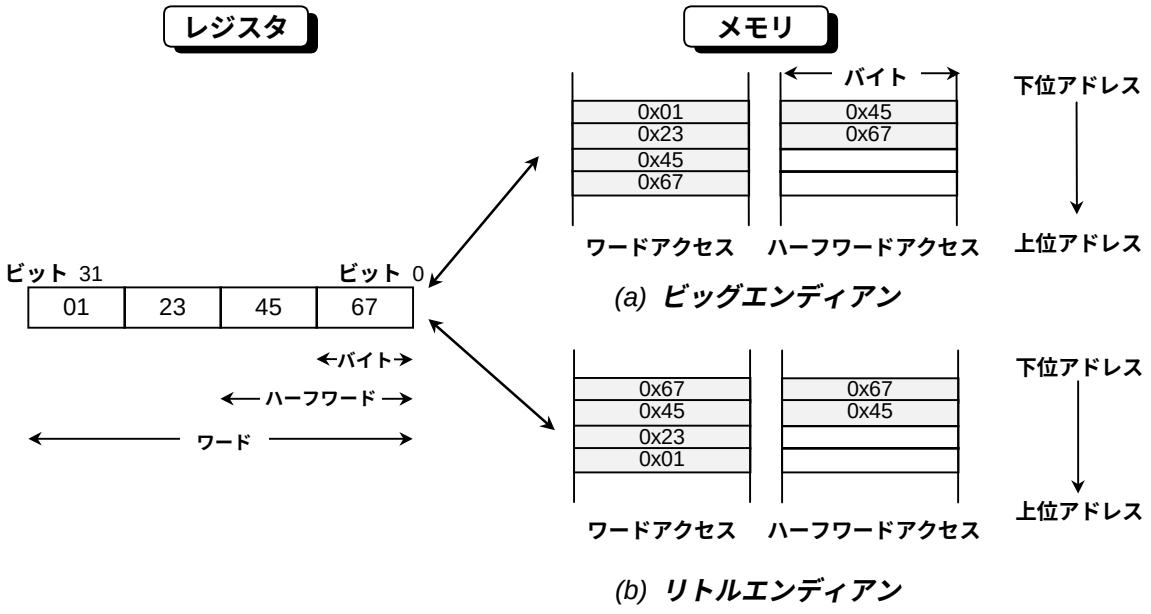
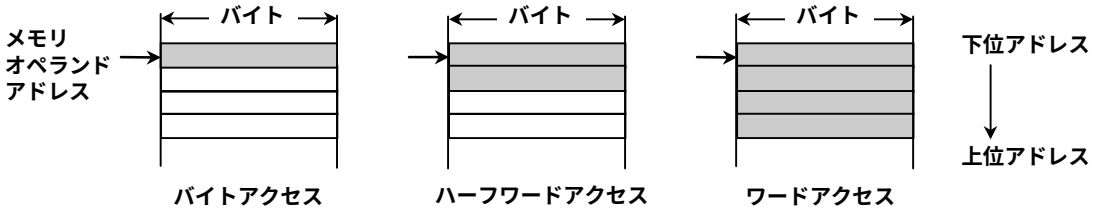


図 2-1　バイト順序

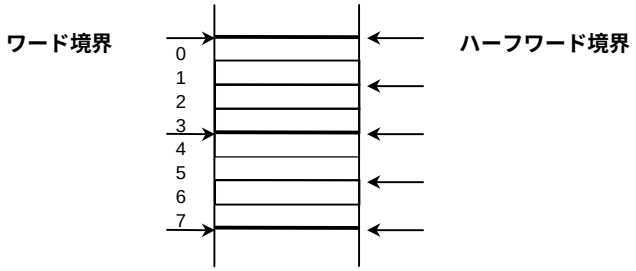
2.1.2　メモリアドレッシング

TX19 のメモリアクセスには、バイトアクセス、ハーフワードアクセス、ワードアクセスがあります。ハーフワード、ワードのアドレス指定には、データが格納されているメモリの最下位アドレスを使用します。これはビッグエンディアンの場合は、データの最上位バイトのアドレス、リトルエンディアンの場合は、データの最下位バイトのアドレスになります。

メモリをアクセスする命令には、位置合わせ (アライメント)境界があります。ハーフワードアクセスの場合は、2 の倍数のバイト境界上に位置合わせしなければならず、ワードアクセスの場合は 4 の倍数のバイト境界上に、位置合わせしなければなりません。



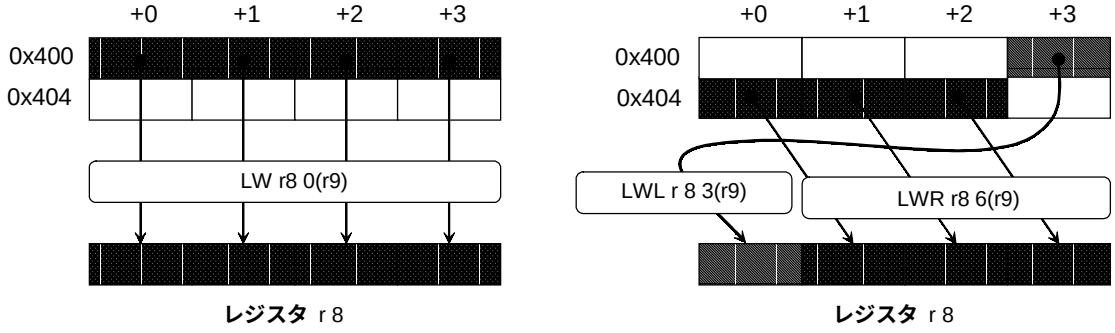
(a) メモリアクセス



(b) ワード境界・ハーフワード境界

図 2-2　位置合わせされたデータ

位置合わせは、処理速度に影響するので、ほとんどの命令では、位置合わせが必要です。ワード境界に整列されていないデータのロード、ストアの場合、LWL、LWR、SWL、SWR という特殊な命令を使います。LWL は LWR と、SWL は SWR とペアで使います。図 2-3にワード境界に整列されているデータと整列されていないデータを示します。



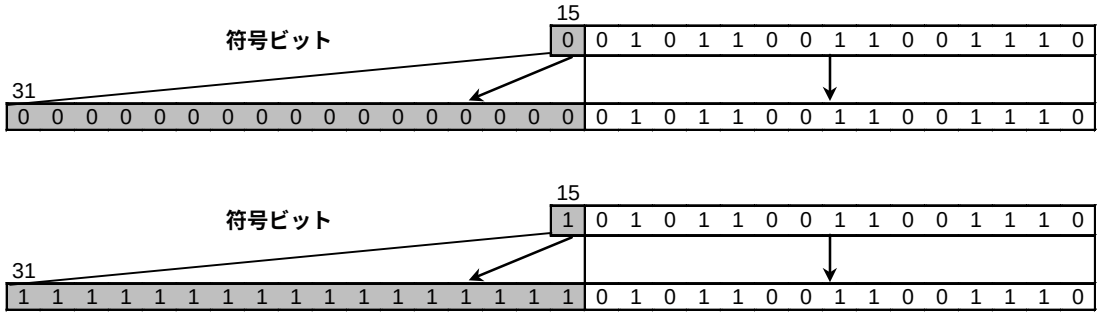
(a) 位置合わせされてるデータのロード (ビッグエンディアン方式)

(b) 位置合わせされていないデータのロード (ビッグエンディアン方式)

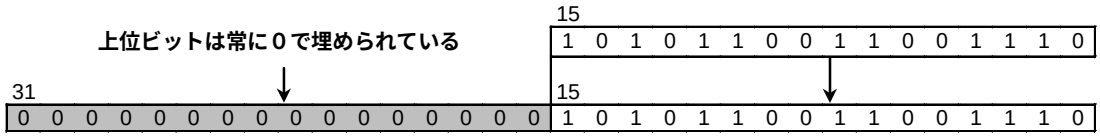
図 2-3　位置合わせされているデータとされていないデータのロード

2.1.3　データ拡張

図 2-4 に符号拡張とゼロ拡張を示します。符号付き数値では、最上位ビットは符号を表し、残りのビットは値の大きさを表します。符号拡張は、16 ビットの即値、またはロードしたバイト、またはハーフワードデータの最上位ビット(符号ビット)を上位のビットにコピーします。ゼロ拡張は、16 ビットの即値、またはロードしたバイト、またはハーフワードデータの最上位ビットの値に関係なく、上位のビットを 0 で埋めます。



(a) 16 ビットから 32 ビットへの符号拡張



(b) 16 ビットから 32 ビットへのゼロ拡張

図 2-4　符号拡張とゼロ拡張

符号拡張は、通常、算術演算で使われます。例えば、ADDI(Add Immediate Signed)命令は ADDI r3，r1，0x1234 のようにオペランドとして 16 ビットの即値をとることができます。この命令は、0x1234 を 32 ビットに符号拡張してからレジスタ r1 の内容に加算し、結果をレジスタ r3 に格納します。

また、符号拡張は LB (Load Byte)、LBU (Load Byte Unsigned)、LH (Load Halfword)、LHU (Load Halfword Unsigned)、LW (Load Word)、SB (Store Byte)、SH (Store Halfword)、SW (Store word) などのロード、ストア命令でも使われます。ロード、ストア命令でサポートされているアドレッシングモードは「ベースレジスタ＋16 ビットオフセット」のみで、LW r9，4(r8) のような書式になります。この命令は、オフセットの 4(0100)を符号拡張し、レジスタ r8 に格納されているベースアドレスに加算することにより、実効アドレスを生成します。そして、実効アドレスにより指定されたワードのデータを r9 にロードします。

LBU 命令が ASCII コードのような符号なし数値を扱うのに対して、LB 命令は、指定されたメモリ位置のデータを符号付き数値として扱います。したがって、LB 命令はロードしたバイトを符号拡張し、LBU 命令はゼロ拡張してから、レジスタに格納します。

また、論理 AND 命令と論理 OR 命令には、AND・ANDI と OR・ORI があります。AND 命令と OR 命令が 2 つのソースレジスタの論理積と論理和をとるのに対して、ANDI(AND Immediate)命令と ORI(OR Immediate)命令はオペランドとして 16 ビットの即値を使用します。ANDI と ORI は 16 ビットの即値をゼロ拡張して、汎用レジスタの内容ビットごとの論理積、論理和 OR をとります。

2.2　プログラミングモデル

TX19 のプログラミングモデルは、CPU レジスタとシステム制御コプロセッサ(CP0)レジスタの 2 つのレジスタグループで構成されています。

2.2.1　CPU レジスタ

図 2-5に CPU レジスタを示します。TX19 には 32 本の汎用レジスタ、1 本のプログラムカウンタ(PC)レジスタ、整数の乗除算結果が格納される 2 本の特殊レジスタ(HI/LO)があります。CPU レジスタはすべて 32 ビットです。

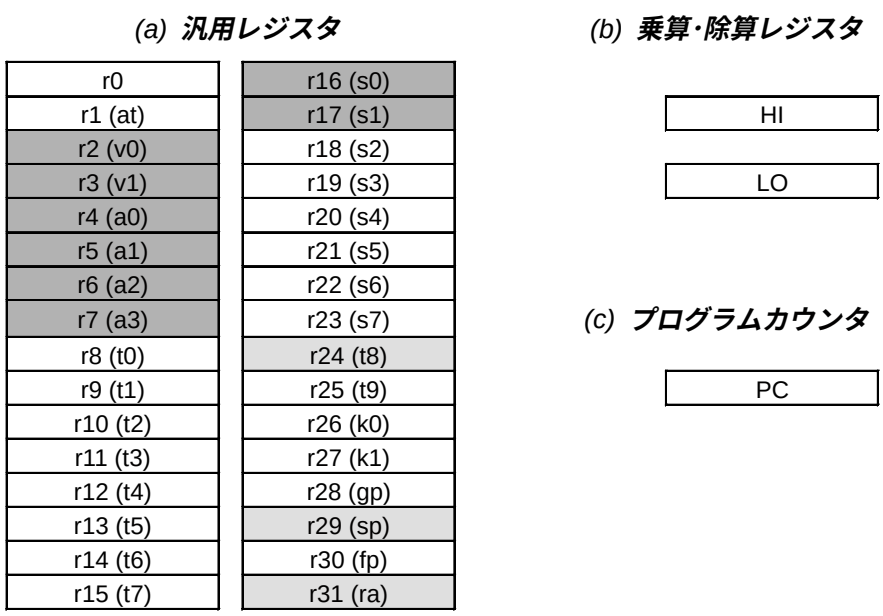


図 2-5　CPU **レジスタ**

■　汎用レジスタ

32 ビット ISA 命令は、図 2-5に示す 32 本の汎用レジスタをすべて使用できます。汎用レジスタには r0～r31 まで番号がつけられています。r0 以外の汎用レジスタは、v0～v1、a0～a3 などのようにアセンブラで使用できるシンボル名(ソフトウエア名)をもっています。32 ビット ISA 命令では、r0 と r31 を除いて、汎用レジスタはすべて同じように扱われます。r0 はハードウエア的に常に値が 0 に固定されています。そのため、r0 は演算結果を破棄したい場合のターゲットレジスタや、値として 0 が必要な場合のソースレジスタとして使用します。r31(ra: リターンアドレス)はリンク機能付きのジャンプ命令、分岐命令、分岐ライクリ命令で使われるリンクレジスタです。これらの命令は、サブルーチンからのリターンアドレスを r31 に格納します。

16 ビット命令では、基本的に、32 本の汎用レジスタのうち r2～r7、r16、r17 の 8 本のレジスタしかアクセスできません。ただし、プロセッサには 32 ビット ISA モードで使用する 32 本のレジスタが存在するため、16 ビット ISA でアクセスできる 8 本のレジスタと残りの 24 本のレジスタ間で値を移送するための move 命令が 16 ビット ISA に用意されています。また、特定の命令では、暗黙的に r24 (t8)、r29 (sp)、r31 (ra)を使用します。r24 は比較結果が格納されるレジスタとして、r29 はスタックポインタとして、r31 はリンクレジスタとして用いられます。

■ HI レジスタ･LO レジスタ

HI レジスタと LO レジスタは整数の乗除算、積和の演算結果が格納されるレジスタです。整数の乗算、積和では、ダブルワード(64 ビット)の結果の上位 32 ビットが HI レジスタに、下位 32 ビットが LO レジスタに格納されます。整数の除算では、商が LO レジスタに、剰余が HI レジスタに格納されます。HI レジスタ、LO レジスタと汎用レジスタのあいだのデータの移送には、MFHI、MFLO、MTHI、MTLO 命令を使います。

■ プログラムカウンタ(PC)

プログラムカウンタの最下位ビットは、ISA モードビットで、0 は 32 ビット ISA モードを、1 は 16 ビット ISA モードを示します。最下位ビットはアドレスの一部とは見なされず、最下位ビットを 0 にクリアしたときの 32 ビット全体の値が現在実行中の命令のアドレスを表します。

2.2.2　システム制御コプロセッサ(CP0)レジスタ

TX19 には、システム制御コプロセッサ CP0 が内蔵されています。CP0 には、図 2-6 に示すユーザーがアクセスできる 9 本のレジスタを含む 32 本のレジスタがあります。これらのレジスタはすべて 32 ビットです。

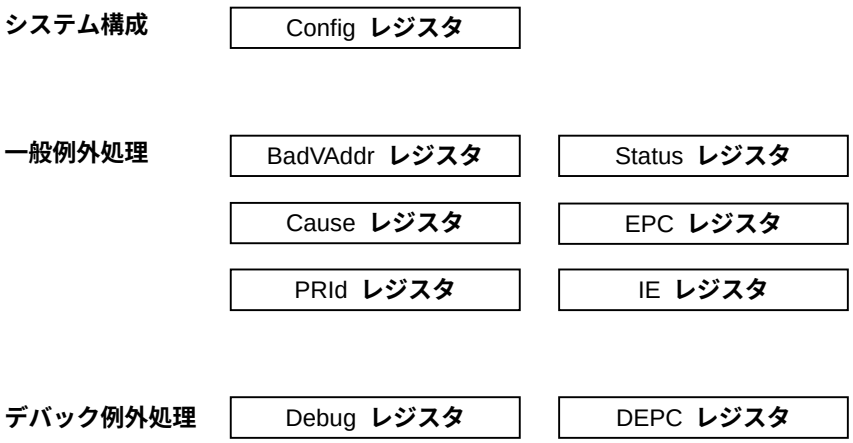


図 2-6　システム制御コプロセッサ (CP0) レジスタ

CP0 レジスタは、システム構成レジスタ、一般例外処理レジスタ、デバック例外処理レジスタの 3 種類に分類されます。プロセッサがカーネルモードのときは、システム制御、コプロセッサ CP0 命令によって、CP0 レジスタに常にアクセスできます。また、プロセッサがユーザーモードのときは、Status レジスタの CU[0]ビットが 1 のときのみ、CP0 レジスタにアクセスできます。プロセッサの動作モードについては「2.6 メモリ管理」で説明します。

表 2-1　システム構成レジスタ

レジスタ名	機能説明
Config	低消費電力モード、キャッシュイネーブル状態などのシステム構成の設定をします。

表 2-2　一般例外処理レジスタ

レジスタ名	機能説明
BadVAddr	仮想アドレスから物理アドレスへの変換で、エラーを起こした仮想アドレスを保持します。読み出し専用です。
Status	動作モード(ユーザーモード･カーネルモード)､割り込みイネーブル状態などのプロセッサの状態を保持します。
Cause	直前に発生した例外の原因を保持します。
EPC	例外プログラムカウンタ。例外を発生した命令のアドレス、ISA モードを保持します。
PRId	TX19 プロセッサコアのリビジョンを示します。読み出し専用です。
IE	Status レジスタの割り込み許可・禁止ビットを操作するレジスタです。

表 2-3　デバック例外処理レジスタ

レジスタ名	機能説明
Debug	デバック例外の原因とデバック時の状態を保持します。
DEPC	デバック例外プログラムカウンタ。デバック例外からのリターンアドレス、ISA モードを保持します。

2.3　32 ビット ISA モード・16 ビット ISA モード

TX19 には、16 ビット ISA モードと 32 ビット ISA モードの 2 つの ISA モードがあります。16 ビット ISA モードと 32 ビット ISA モードは、プログラムの実行中にサブルーチン単位で、命令により切り換えられます。一般的に、コードサイズを縮小したい部分は、16 ビット ISA モードを用い、高速化したい部分は 32 ビット ISA モードを用います。

プログラムカウンタ(PC)の最下位ビットが ISA モードビットで、0 のときは 32 ビット ISA モードになり､1 のときは 16 ビット ISA モードになります｡32 ビット ISA モードと 16 ビット ISA モードは JALX、JR、JALR 命令により切り換えられます。

16 ビット ISA モード中で例外が発生すると､プロセッサは自動的に 32 ビット ISA モードに切り換わります。このとき、リターンアドレスと ISA モードビットが、例外プログラムカウンタ(EPC)またはデバック例外プログラムカウンタ(DEPC)に保存されます。EPC レジスタに保存されているリターンアドレスへ戻るには、JR 命令を使います。また、デバック例外の場合は、DERET 命令により DEPC レジスタに保存されているリターンアドレスへ戻ります。

32 ビット ISA、16 ビット ISA にはそれぞれ図 2-7に示す命令があります。

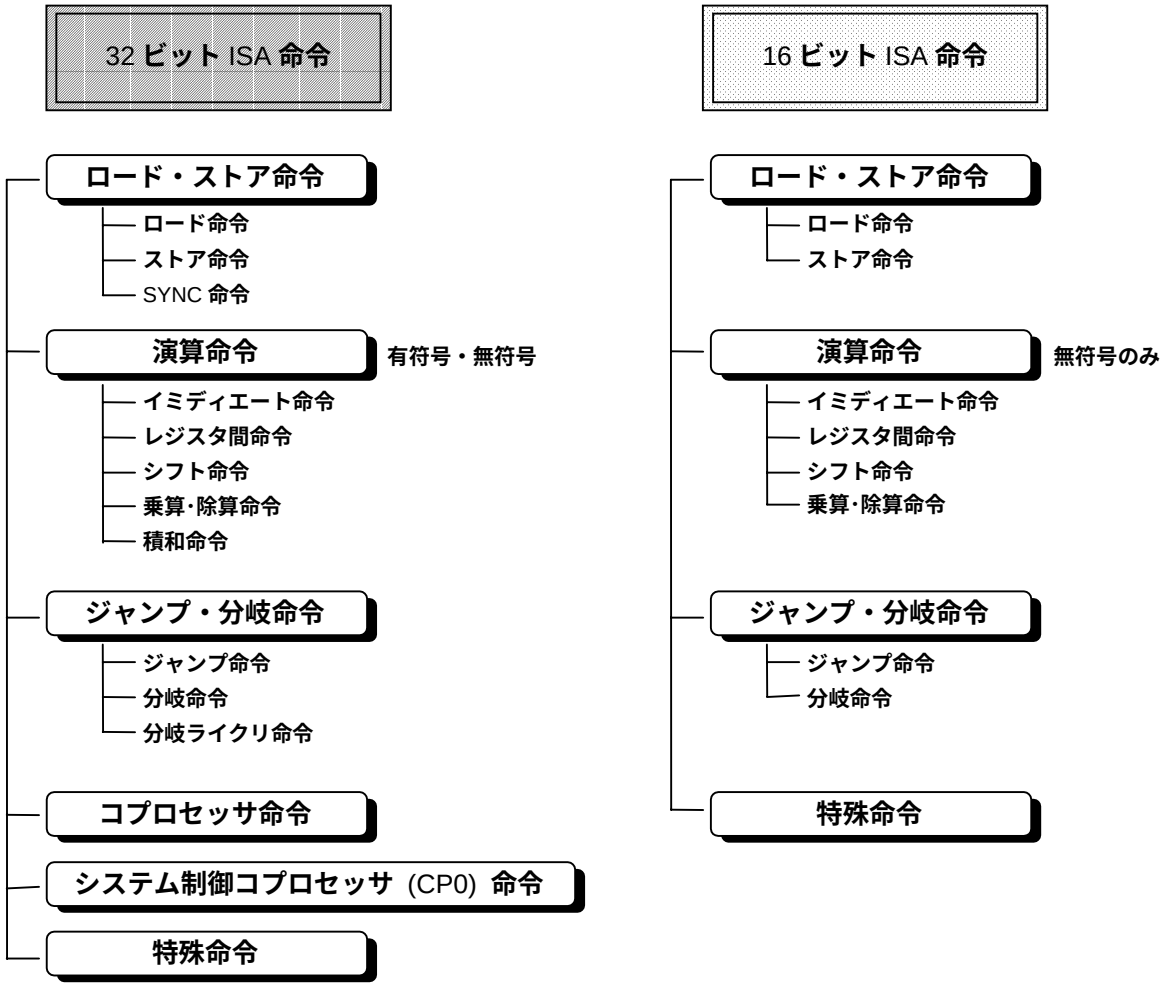


図 2-7　32 ビット ISA と 16 ビット ISA の命令

32 ビット ISA の命令は、すべて 32 ビットです。16 ビット ISA の命令は、32 ビットの JAL、JALX 命令を除き、すべて 16 ビットです。16 ビット ISA モードでは EXTEND という命令が用意されています。EXTEND 命令自体は 16 ビットで、オペコードと 11 ビットの即値のみで構成されます。EXTEND 命令は、それだけでは機械語の命令を生成しませんが、自身の即値を後続命令の即値に連結することにより、16 ビットの即値を使えるようにします。EXTEND 命令により、16 ビットの命令が 32 ビットに拡張されます。これを拡張命令をいいます。

基本的に、各 16 ビット命令は 32 ビット命令に対応しており、メインメモリまたは命令キャッシュからフェッチされた 16 ビット命令は伸長回路により、32 ビットに変換されてから、通常の命令デコーダに送り込まれます。ただし、16 ビット命令と 32 ビット命令で機能が異なる命令はいくつかあります。付録 B に 16 ビットモードと 32 ビットモードの命令フォーマットの対応、機能の違いについての説明をします。

2.4　コプロセッサ

コプロセッサとは、CPU の負荷を減らすことにより、処理スピードを上げるためのユニットです。TX19 には、CP0、CP1、CP2、CP3 の最大 4 つのコプロセッサを接続できます。

CP0 はシステム制御コプロセッサで、CP0 は TX19 に内蔵されています。システム構成、例外処理、メモリ管理などを行います。CP0 の基本機能はプロセッサコアに、拡張機能はメモリ管理ユニット(MMU)に内蔵されています。

CP1、CP2、CP3 はプロセッサコアの外側に置かれ、浮動小数点演算などの複雑で時間のかかるタスクを実行します。CP1、CP2、CP3 は製品ごとに違うので、データーシートを参照してください。

ユーザーモードでは、CP0 命令および CP0 レジスタを使用できるかどうかは、Status レジスタの CU[0]ビットにより制御します。CU[0]ビットがクリアされているときに、ユーザーモードのプログラムで CP0 命令を実行しようとすると、コプロセッサ使用不可例外が発生します。カーネルモードでは、CU[0]ビットの設定に関係なく、すべての CP0 命令が実行できます。

Status レジスタの CU[3:1]ビットはユーザーモード、カーネルモードでの各コプロセッサの使用可能、不能状態を制御します。その対応する CU ビットがクリアされているときに、コプロセッサ命令を実行しようとすると、コプロセッサ使用不可例外が発生します。

4 本のコプロセッサは、それぞれ最大 64 個の 32 ビット長のレジスタをもつことができます。システム制御コプロセッサ(CP0)には、32 本のレジスタがあり、そのうちの 9 本をユーザーがアクセスできます。8 章で詳しく説明します。

2.5　パイプライン

TX19 は 5 段パイプラインをもっています。各命令は 5 つのステージに分けて実行されます。それぞれのステージは、ほぼ 1 クロックで実行されるため、1 命令の実行は最短で 5 クロックかかります (16 ビット ISA モードの JAL、JALX 命令はそれ以上のサイクルが必要です)。ところが 5 段パイプラインは、各命令の実行をフェッチ(F)、デコード(D)、実行(E)、メモリアクセス(M)、レジスタライトバック(W)の 5 つの部分に分割し、図 2-8に示すように最大 5 つの命令を同時に実行します。したがって、1 命令当たり、ほぼ 1 クロックで実行できます。

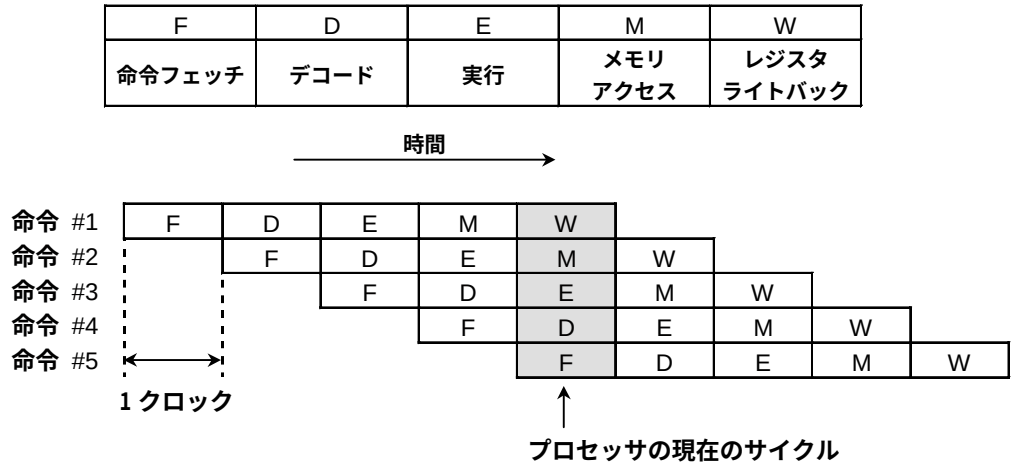


図 2-8　5 段パイプライン

2.6　メモリ管理

TX19 にはユーザーモードとカーネルモードの 2 種類の動作モードがあります。TX19 は例外が発生すると、自動的にカーネルモードに移行します。また、システムリセットがかかると、プロセッサはリセット例外により立ち上がるため、立ち上げ時はカーネルモードになります。カーネルモードからユーザーモードへの復帰は PFE (Restore From Exception) 命令または、DERET (Debug Exception Return) 命令により行います。

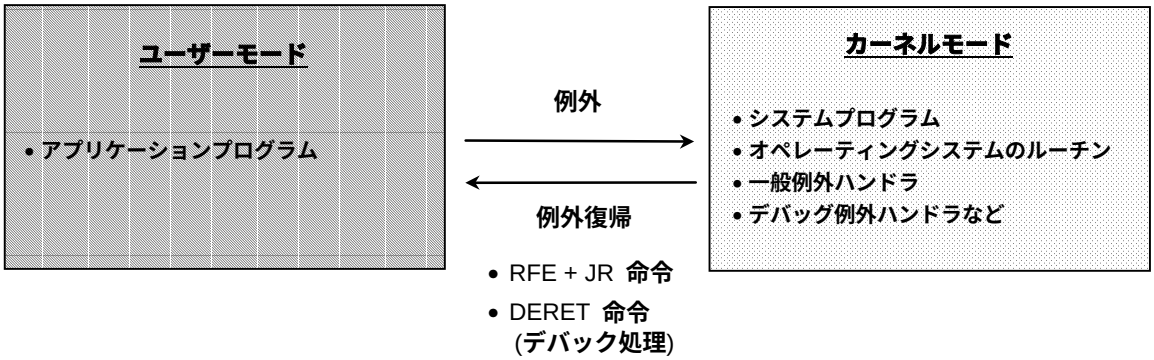


図 2-9　動作モード

動作モードによりプログラムで使用できる仮想アドレス空間、レジスタ、命令が異なります。カーネルモードはユーザーモードより高い特権レベルが与えられ、すべての仮想アドレス空間、レジスタ、命令を使用できます。ユーザーモードではこれらの使用が制限されます。オペレーティングシステムのルーチン、例外ハンドラ、デバックハンドラなどのプログラムは、カーネルモードで実行します。これによりカーネル

モードでのみ使用できるアドレス空間に、ユーザーモードからアクセスできないようにシステムを保護できます。

TX19 のメモリ管理ユニット(MMU)はダイレクトセグメントマッピング方式を使っており、TLB を内蔵していません。仮想アドレスから物理アドレスへのマッピングを図 2-10に示します。仮想アドレス空間は、4 つの領域に分けられています。kuseg はカーネルモード、またはユーザーモードのどちらかでもアクセスできる領域です。他の 3 つの領域 kseg0、kseg1、kseg2 は、カーネルモードでのみ使用できます。メモリ管理については 6 章で詳しく説明します。

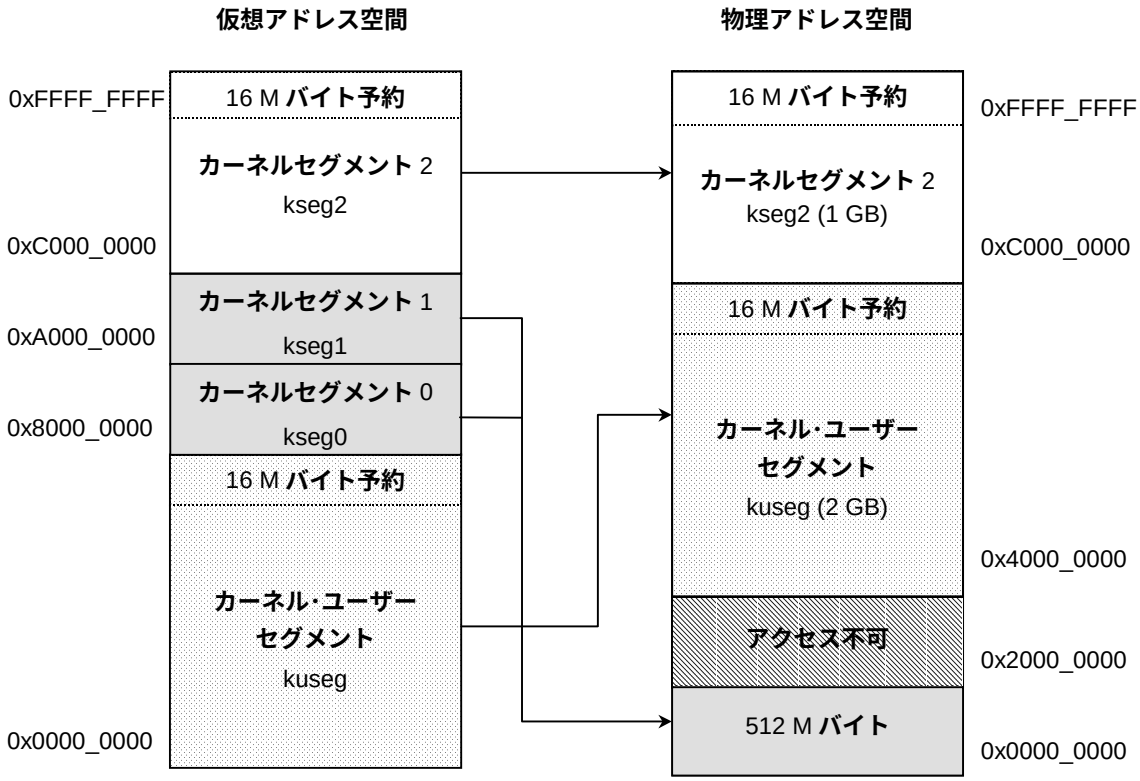


図 2-10　ダイレクトセグメントマッピング方式

第3章　32 ビット ISA の概要

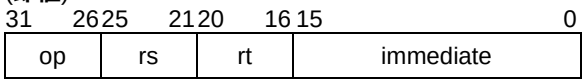
この章では、32 ビット ISA の命令とアドレッシングモードの概要を説明します。また、32 ビット命令を使った基本的なプログラミングについても説明します。32 ビット ISA の命令は以下のように分類されます。

- ロード命令・ストア命令
- 演算命令
- ジャンプ命令・分岐命令・分岐ライクリ命令
- コプロセッサ命令
- 特殊命令

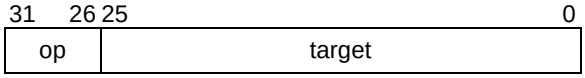
3.1　命令形式

32 ビット ISA の命令は、すべて 32 ビット長で、図 3-1に示す 3 種類の命令形式があります。命令形式を 3 種類に限定することにより、命令のデコードを簡素化しています。複雑で使用頻度の低いオペレーション命令は、コンパイラにより複数の命令を組み合わせることにより実現します。32 ビット命令はすべてワード境界で位置合わせしなければなりません。

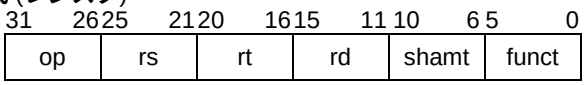
I 形式 (即値)



J 形式 (ジャンプ)



R 形式 (レジスタ)



op	6 ビットの命令コード
rs	5 ビットのソースレジスタ番号
rt	5 ビットのターゲットレジスタ番号または分岐条件
immediate	16 ビットの即値、分岐オフセット値 またはアドレスのオフセット値
target	26 ビットのジャンプターゲットアドレス
rd	5 ビットのデスティネーションレジスタ番号
shamt	5 ビットのシフト量
funct	6 ビットの機能コード

図 3-1　命令形式

3.2　ロード・ストア命令

ロード・ストア命令は、メモリと CPU 汎用レジスタ間でデータを移送するのに使います。ロード・ストア命令はメモリからレジスタへデータをロードするか、またはレジスタからメモリへデータをストアするだけです。レジスタとメモリの内容に対して、算術・論理演算を実行する命令はありません。

3.2.1　アドレッシングモード

32 ビット ISA のロード・ストア命令は、すべて I 形式の命令です。ロード・ストア命令は、図 3-2に示すオフセット付きレジスタ間接アドレッシングモードを使います。実効アドレスは、ベースレジスタとして指定した汎用レジスタの値に 16 ビットの即値を符号拡張した値を加算することにより、計算されます。以下に例を示します。

```
LW r9,4(r8)
```

この命令では、4 (0100) がオフセットで、r8 がベースアドレスが格納されている汎用レジスタ、r9 がロード先のターゲットレジスタです。

このアドレッシングモードでは、r0 をベースレジスタに指定すると、即値アドレッシングモードと同じになり、またオフセット 0 を指定すると、レジスタ直接アドレッシングモードと同じになります。

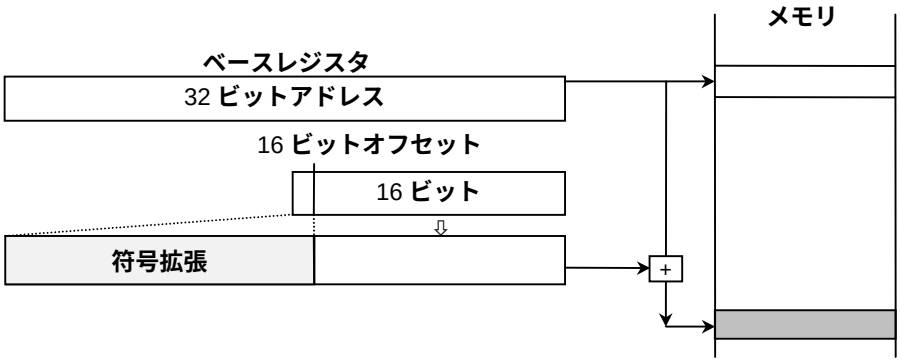


図 3-2　オフセット付きレジスタ間接アドレッシングモード

3.2.2　位置合わせされているデータのロード・ストア

表 3-1には、バイトアクセス、ハーフワードアクセス、ワードアクセス用のロード・ストア命令を示します。LB 命令と LH 命令では、ロードしたバイトまたはハーフワードは符号拡張されて、レジスタに格納されます。それに対して、接尾辞 (unsigned) が付いた LBU 命令と LHU 命令では、ロードしたバイトまたはハーフワードはゼロ拡張されて、レジスタに格納されます。

表 3-1　位置合わせされているデータのロード・ストア

データ形式	符号なしロード	符号付きロード	ストア
バイト	LBU	LB	SB
ハーフワード	LHU	LH	SH
ワード	LW	—	SW

3.2.3　位置合わせされていないデータのロード・ストア

前項に示したロード・ストア命令を使って、ハーフワード境界に位置合わせされていないハーフワードや、ワード境界に位置合わせされていないワードをロードまたはストアしようとすると、アドレスエラー例外が発生します。位置合わせされていないワードのロード・ストアについては、表 3-2に示す特別な命令が用意されています。LWL (Load Word Left) 命令と LWR (Load Word Right) 命令はペアで、SWL (Store Word Left) 命令と SWR (Store Word Right) 命令はペアで使います。これらの命令は、位置合わせされていないワードで、ロード・ストア命令とシフト命令を組み合わせて使うより効率的です。8 ビット、16 ビット CPU 用に作成した古いプログラムを再利用するのに便利です。

表 3-2　位置合わせされていないデータのロード・ストア

データ形式	符号なしロード	ストア
左部分 (上位バイト)	LWL	SWL
右部分 (下位バイト)	LWR	SWR

3.2.4　SYNC 命令

SYNC 命令は、SYNC 命令の直前に実行したロード、ストア、命令フェッチまで、命令パイプラインをインタロックし後続のロード、ストアの実行を遅らせます。これにより、SYNC 命令の前の命令と後続の命令の実行順序を守ることができます。

3.2.5　32 ビットのアドレスの生成

32 ビット ISA のロード・ストア命令は、オフセットとして 16 ビットの符号付き即値しかとることができません。最上位ビットは符号で、残りの 15 ビットがオフセットの大きさを指定するのに使われます。したがって、オフセットの範囲は、–32768 から +32767 となります。オフセットがこの範囲外の場合は、オフセットをいったん汎用レジスタに格納する必要があります。3 つの例を以下に示します。

◆ 例 1　ベースアドレス + 32 ビットオフセット
以下の例では、ADDU (Add Unsigned) 命令によりレジスタ r5 に格納されているオフセットをレジスタ r4 のベースアドレスに加算し、その結果を r4 に書き戻しています。次に LW 命令で、r4 をベースレジスタとして指定しています。

```
ADDU    r4,r4,r5
LW      r6,0(r4)
```

◆ 例 2　ベースアドレス + 32 ビットオフセット

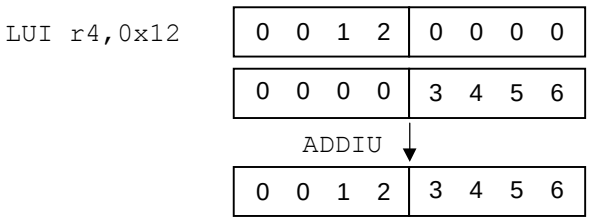
以下の例では、LUI (Load Upper Immediate) 命令を使って、指定された 16 ビットの即値をレジスタ r5 の上位 16 ビットにロードしています。下位 16 ビットはゼロで埋められます。次の ADDU (Add Unsigned) 命令で r4 のベースアドレスに r5 を加算しています。つまり、ベースアドレスにオフセットの上位の 16 ビットが加算されたことになります。そこで、LW 命令では、r4 にオフセットの下位 16 ビットを加算することにより、最終的なターゲットアドレスを生成しています。

```
LUI     r5,0x12
ADDU    r4,r4,r5
LW      r6,0x3454(r4)
```

◆ 例 3　任意の 32 ビットのアドレス

以下の例で、LUI (Load upper Immediate) 命令を使って、16 ビットの即値をレジスタ r4 の上位 16 ビットにロードしています。次に ADDIU (Add Immediate Unsigned) 命令でオフセットの下位 16 ビット 0x3456 を r4 に加算しています。これで、r4 に 32 ビットのオフセットが格納されました。したがって、LW 命令では、r4 をベースレジスタとして使えば、オフセットをゼロとすることができます。

```
LUI     r4,0x12
ADDIU   r4,r4,0x3456
LW      r6,0(r4)
```



3.3　演算命令

この項では、32 ビット ISA の演算命令について説明します。3.3.1項では、演算命令の分類を示します。3.3.2項では、32 ビットの定数を使用する演算についてを説明します。3.3.3項では、64 ビットの加算、減算をどのように実行するか、例を使って説明します。3.3.4項では、整数演算のオーバフローをトラップを使わずに検出する方法を示します。3.3.5項では、64 ビット×64 ビットの乗算を実行する方法について説明します。32 ビット命令にはローテート命令がありません。3.3.6項では、32 ビット ISA の命令を使って、どのようにローテートを実現するか説明します。

3.3.1　演算命令の分類

32 ビット ISA の演算命令は、表 3-3に示す 5 つのグループに分類されます。演算命令は、算術、比較、論理、シフト、乗算、除算、積和演算命令があります。演算命令は、オペランドとして 16 ビットの即値をとる I 形式か、レジスタを 3 つ指定する R 形式になります。

表 3-3　演算命令

分類	命令	オペコード
ALU 即値	加算	ADDI・ADDIU
	大小比較	SLTI・SLTIU
	論理積	ANDI
	論理和	ORI
	排他的論理和	XORI
	上位即値のロード	LUI
3 オペランドレジスタタイプ	加算	ADD・ADDU
	減算	SUB・SUBU
	大小比較	SLT・SLTU
	論理積	AND
	論理和	OR
	排他的論理和	XOR
	否定論理和	NOR
シフト	論理シフト	SLL・SLLV・SRL・SRLV
	算術シフト	SRA・SRAV
乗算・除算	乗算	MULT・MULTU
	除算	DIV・DIVU
	HI・LO レジスタと汎用レジスタ間の転送	MFHI・MFLO・MTHI・MTLO
積和		MADD・MADDU

ALU 即値命令では、ソースオペランドは汎用レジスタと 16 ビットの符号付き即値です。例えば、ADDI (Add Immediate) 命令のシンタックスは「ADDI *rd, rs, immediate*」で、ソースレジスタ (*rs*) と符号拡張した即値 (*immediate*) を加算し、その結果をデスティネーションレジスタ (*rd*) に格納します。

3 オペランドレジスタタイプの命令は、汎用レジスタに格納されている 2 つの値を対象に演算を実行し、その結果を汎用レジスタに格納します。

シフト命令は、汎用レジスタの内容を指定されたビット数だけ左または右へずらします。シフト命令には、論理シフト命令と算術シフト命令の 2 種類があります。また、シフト変数命令 (SLLV、SRLV、SRAV) には、シフト量を指定するフィールド (*shamt*) がなく、代わりにシフト量が格納されている汎用レジスタを指定します。

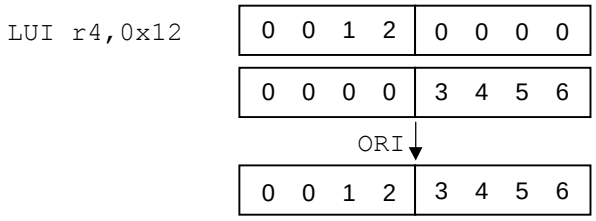
乗除算命令は、汎用レジスタに格納されている 2 つの整数値を対象に乗算または除算を行い、その結果を特殊レジスタの HI レジスタと LO レジスタに格納します。一般の命令では、HI レジスタ・LO レジスタにはアクセスできません。汎用レジスタと HI レジスタ・LO レジスタ間でデータを移送するには、MFHI、MFLO、MTHI、MTLO 命令を使います。TX19 では、MIPS の命令を当社で拡張しており、乗算の場合、積の下位 32 ビットを LO レジスタと汎用レジスタの両方に同時に格納できます。「3.3.5 64 ビット × 64 ビットの乗算」で、使用例を示します。

積和命令は当社で拡張した命令です。積和命令は、32 ビットの 2 つの整数を掛け合わせ、HO・LO レジスタの 64 ビットの値に加算します。また、同時に結果の下位 32 ビットを汎用レジスタに格納することもできます。積和演算は、デジタル信号処理 (DSP) で頻繁に使用されるため、専用の積和演算器 (MAC) で高速に実行されます。

3.3.2　32 ビットの定数

I 形式の命令では、即値フィールドは 16 ビットしかありません。即値が 16 ビットより大きい場合は、32 ビットの定数を生成し、それを一時的に汎用レジスタに格納しておく必要があります。以下の例では、LUI (Load Upper Immediate) 命令は指定された即値をレジスタ r4 の上位 16 ビットに格納し、下位 16 ビットを 0 で埋めます。そして、次の ORI (OR Immediate) 命令で、r4 の内容と ORI 命令自体の即値の論理和をとり、結果を r4 に書き戻しています。ORI 命令の即値はゼロ拡張されます。

```
LUI     r4,0x12
ORI     r4,r4,0x3456
```



次に、汎用レジスタの内容に 32 ビットの定数を加算する例を示します。この例では、LUI 命令で r5 の上位 16 ビットに 0x1234 を格納し、それに、ADDIU (Add Immediate Unsigned) 命令により 0x5678 を加算して、32 ビットの定数 0x12345678 を得ています。最後に ADDU (Add Unsigned) 命令により r4 と r5 を加算し、その結果を r6 に格納しています。

```
LUI     r5,0x1234
ADDIU   r5,r5,0x5678
ADDU    r6,r4,r5
```

注意: **ADDI 命令、SLTI 命令では、即値は 32 ビットに符号拡張されます。また、ADDIU 命令と SLTIU 命令のニモニックは、それぞれ** Add Immediate *Unsigned* (**符号なし**) **と** Set On Less Than Immediate *Unsigned* (**符号なし**) **を表しますが、即値は** ADDI **命令、**SLTI **命令と同様、符号拡張されます。**ADDI **命令と** ADDIU **命令の唯一の違いは、**ADDIU **ではオーバフロー例外が絶対に発生しない点です。したがって、**ADDIU **命令は、オーバフロー例外を発生せずに、負の数値の加算を行いたいときに使うことができます。**TX19 **の命令セットに符号付き即値をともなう減算命令がないので、**ADDIU **命令は便利です。また、**SLTI **命令と** SLTIU **命令の違いは、**SLTI **命令が** 2 **値 (***rs* **と符号拡張した即値) を符号付き整数として比較するのに対して、**SLTIU **命令は** 2 **値 (***rs* **と符号拡張した即値) を符号なし整数として比較する点です。**

注意: **通常、アセンブラは** 16 **ビットより長い即値も受けつけます。例えば、以下のように記述したとします。**

```
ADDI  r3,r2,0x12345678
```

すると、アセンブラはこの命令を次のように自動的に複数の命令に分解してくれます。

```
LUI   r1,0x1234
ORI   r1,r1,0x5678
ADD   r3,r2,r1
```

これにより、プログラミングの作業負荷を軽くすることができます。この例で分かるように、レジスタ r1 **はアゼンブラの予約レジスタになっています。ユーザーのアセンブリ言語プログラム中では** r1 **を使わないでください。**

3.3.3　64 ビットの加算・減数

加算、減算したい数値が 32 ビットより長い場合があります。その場合、汎用レジスタは 32 ビットなので、32 ビットずつに分けて演算を行う必要があります。図 3-3 に 64 ビットの定数の加算・減算を示します。この図では、64 ビットの定数の上位 32 ビットが r3 に、下位 32 ビットが r2 に格納されています。同様に r5 と r4 にも、64 ビットの定数上位 32 ビットと下位 32 ビットが格納されています。

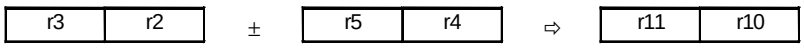


図 3-3　64 **ビット加算・減算**

■　**キャリーが発生する加算**

2 つの 64 ビットの定数を加算するためのコード例を以下に示します。

```
ADDU     r10,r2,r4  # r10 ← r2 + r4
SLTU     r11,r10,r2 # r10 (和)<r2 ならば r11=1
ADD(U)   r11,r11,r3 # r11 ← r11 (キャリー) + r3
ADD(U)   r11,r11,r5 # r11 ← r11 + r5
```

最初の ADDU 命令は、2 つの定数の下位 32 ビットどうしを加算し、結果を r10 に格納しています。TX19 には、算術演算でキャリーが発生したことを記録するフラ

グがありません。そのため、加算でキャリーが発生した場合、何らかの方法でそれを記録しておかなければなりません。ここでは正の数どうしの加算に限定して話をします。この場合、和がどちらかのオペランドよりも小さくなった場合、キャリーが発生したことになります。したがって、次の SLTU (Set On Less Than Unsigned) 命令で、r10 が r2 より小さいかどうかを調べ、小さい場合は、r11 に 1 を設定しています。そして、次の 2 つの ADD (U) 命令で、キャリービット (0 または 1) と 2 つの定数の上位 32 ビットを加算しています。

最後の 2 つの命令は ADD 命令でも ADDU 命令のどちらでもかまいません。ADD 命令と ADDU 命令の唯一の違いは、ADDU (Add Unsigned) 命令ではオーバフロー例外が発生しないという点だけだからです。

ただし、ADDU 命令を使用するときは、オーバフローが発生するかどうかを判断し、オーバフローが発生する場合の処理と、発生しない場合の処理を行うためのコードを別々に用意しておかなければなりません。これについて次の項で説明します。

■　**ボローが発生する減算**

64 ビットの減算を行う場合、上位 32 ビットのオペランドからの下位 32 ビットオペランドへのボローに注意しなければなりません。ボローが発生する減算は、キャリーが発生する加算によく似ています。以下に 64 ビットの定数から 64 ビットの定数を減算する例を示します。

```
SLTU    r8,r2,r4      # r2<r4 ならば r8=1
SUBU    r10,r2,r4     # r10 ← r2 - r4
SUB(U)  r11,r3,r5     # r11 ← r3 - r5
SUB(U)  r11,r11,r8    # r11 ← r11 - r8 (ボロー)
```

最初に、SLTU 命令で r2 (被減数) が r4 (減数) より小さいかどうか調べ、小さければ、r8 を 1 に設定します。これで、下位 32 ビットの減算でボローが発生したとき、ボローが r8 に記録されます。r8 の内容は最後の SUB (U) 命令で減算しています。

SUB 命令と SUBU 命令の唯一の違いは、SUBU 命令ではオーバフロー例外が発生しないという点だけです。

3.3.4　オーバフローが発生するかどうかの判断

前項で説明したように、符号付き加算命令 (ADD)、符号付き減算命令 (SUB) は、加算・減数結果で、2 の補数のオーバフローが発生した場合、オーバフロー例外を発生します。これに対して、符号なし加算・減算命令 (ADDU・SUBU) ではオーバフロー例外は発生しません。符号付き演算で例外を発生させずにオーバフローを検出したい場合、または符号なし演算でオーバフローを検出したい場合には、オーバフロー検出用のルーチンを作成しなければなりません。

加算の場合は、オペランドの符号が同じで、加算結果 (和) の符号が異なると、オーバフローが発生したことになります。以下に、符号付き加算の結果、オーバフローが発生したかどうかを調べるためのコードを示します。

```
        ADDU r2,r3,r4    # r2 ← r3 + r4
        XOR  r5,r3,r4    # r3 と r4 の符号を比較。
                         # 異なる場合、オーバフローなし(r5<0)
        BLTZ r5, No_Ov   # r4<0 ならば、No_0v に分岐
        XOR  r5,r2,r3    # 和(r2)と加数(r3)の符号を比較。
                         # 異なれば、オーバフロー発生 (r5<0)
        BLTZ r4,Ov       # r5<0 ならば、0v に分岐

    No_0v:
```

また、a－b＝c 減算では、2 つの a と b の符号が異なり、減算結果 c の符号が被減数 a の符号と異なる場合、オーバフローが発生したことになります。以下に、符号付き減算の結果、オーバフローが発生したかどうかを調べるためのコードを示します。

```
        SUBU r2,r3,r4    # r2 ← r3 - r4
        XOR  r5,r3,r4    # r3 と r4 の符号を比較。
                         # 同じならば、オーバフローなし (r4=>0)
        BGEZ r5,No_Ov    # r5=>0 ならば、No_0v に分岐
        XOR  r5,r2,r3    # 差(r1)と被減数(r3)の符号を比較。
                         # 異なれば、オーバフロー発生(r5<0)
        BLTZ r5,Ov       # r5<0 ならば、0v に分岐

    No_Ov:
```

3.3.5　64 ビット × 64 ビットの乗算

TX19 で 2 つの整数を乗算する場合、整数は汎用レジスタに格納されていなければなりません。汎用レジスタは 32 ビットなので 64 ビット × 64 ビットの乗算では、被乗数、乗数のオペランドを格納するのに、レジスタを 2 つずつ使用します。

図 3-4に例を示します。この例では被乗数の上位 32 ビットが r3、下位 32 ビットが r2 に格納されているものとします。また、同様に乗数は r5 と r4 に格納されているものとします。

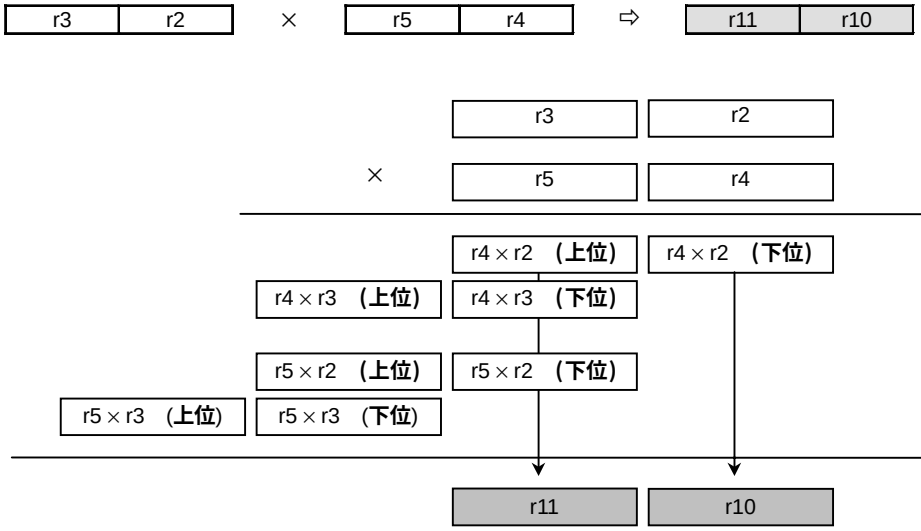


図 3-4　64 ビット × 64 ビットの乗算

以下に 64 ビット×64 ビットの乗算を実行するためのコードの例を示します。積は最大 128 ビットになりますが、説明を簡単にするため、積の下位 64 ビットのみを考えます。

```
MULTU   r10,r2,r4   # r4 × r2, 積の下位 32 ビットを r10 に格納
MFHI    r11         # 積の上位 32 ビットを HI から r11 に転送
MULTU   r9,r3,r4    # r3 × r4, 積の下位 32 ビットを r9 に格納
ADDU    r11,r11,r9  # r11 ← r11 + r9
MULTU   r9,r2,r5    # r5 × r2, 積の下位 32 ビットを r9 に格納
ADDU    r11,r11,r9  # r11 ← r11 + r9
```

TX19 のアーキテクチャでは、MIPS の MULTU (Multiply Unsigned) 命令の機能が拡張されています。MIPS アーキテクチャでは、MULTU 命令は、被乗数と乗数が格納されているソースレジスタを 2 つしか指定できず、積は HI レジスタと LO レジスタに格納されます。それに対して、TX19 では、MULTU 命令は 3 つのオペランドを指定することができます。これにより、TX19 の MULTU 命令は積の下位 32 ビットを LO レジスタのほかに汎用レジスタにも同時に格納することができます。したがって、LO レジスタの内容を汎用レジスタに転送するのに、MFLO (Move From LO) 命令を使う必要がありません。

MFHI (Move From HI) 命令は、HI レジスタの内容、すなわち積の上位 32 ビットを汎用レジスタに転送します。

3.3.6　ローテート命令

TX19 には、機械語命令として、ローテート命令がありません。(ただし、アセンブラではローテートを実行するマクロ命令があります。) 例えば、左方向ローテートでは、各ビットの値が右から左方向へシフトし、左端 (MSB) からあふれたビットは、右端 (LSB) に入ります。それに対し、左シフトでは、シフトにより左端からあふれたビットは廃棄され、右側の空いたビットは 0 で埋められます。

TX19 では、ローテート命令は、シフト命令と論理 OR 命令を組み合わせて実現されます。図 3-5にレジスタの内容を 6 ビット左ローテートする例を示します。

```
SLL   r9,r8,6

SRL   r8,r8,(32-6)

OR    r8,r8,r9
```

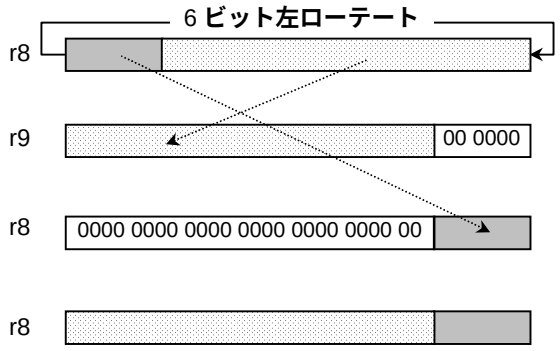


図 3-5　6 ビット左ローテート

図 3-5では、SLL (Shift Left Logical) 命令は r8 の内容を 6 ビット左にシフトし、その結果を r9 に格納しています。空いた下位ビットは 0 で埋められます。次に SRL (Shift Right Logical) 命令で r8 の内容を 26 (32 – 6) ビット右にシフトしています。最後に、OR 命令で r8 と r9 の内容の論理 OR をとり、その結果を r8 に格納しています。その結果、r8 を 6 ビット左ローテートしたのと同じ結果が得られます。

3.4　ジャンプ・分岐・分岐ライクリ命令

プログラムの流れを変える命令には、ジャンプ命令、分岐命令、分岐ライクリ命令があります。3.4.1項でこれらの命令の概要を説明します。3.4.2項では、ジャンプ命令、分岐命令、分岐ライクリ命令でサポートされているアドレッシングモードについて説明します。3.4.3項では、32 ビット ISA モードと 16 ビット ISA モードの切り換え方法について説明します。3.4.4項では、一般分岐命令と分岐ライクリ命令の違いについて説明します。3.4.5項では、大小関係に基づき分岐する方法について説明します。3.4.6項では、32 ビットの絶対アドレスへジャンプする方法について説明します。3.4.7項では、サブルーチンコールとサブルーチンからの復帰について説明します。

3.4.1　ジャンプ・分岐・分岐ライクリ命令の概要

TX19 では、ジャンプ命令は、無条件分岐のための命令です。それに対して、分岐命令と分岐ライクリ命令は、多くのマイクロプロセッサで条件付きジャンプと呼んでいる命令で、条件が成立した場合のみプログラムの流れを変えます。表 3-4と表 3-5 に 32 ビット ISA で用意されているジャンプ、分岐、分岐ライクリ命令を示します。

表 3-4　ジャンプ命令 (32 ビット ISA)

オペコード	命令	アドレッシング	命令形式
J	Jump	ページ内絶対	I 形式
JAL	Jump And Link	ページ内絶対	I 形式
JALX	Jump And Link eXchange	ページ内絶対	I 形式
JR	Jump Register	レジスタ間接	R 形式
JALR	Jump And Link Register	レジスタ間接	R 形式

表 3-5　分岐命令・分岐ライクリ命令 (32 ビット ISA)

オペコード	命令	条件	アドレッシング	命令形式
BEQ(L)	Branch On Equal (Likely)	*rs* = *rt*	PC 相対	I 形式
BNE(L)	Branch On Not Equal (Likely)	*rs* ≠ *rt*	PC 相対	I 形式
BGTZ(L)	Branch On Greater Than Zero (Likely)	*rs* > 0	PC 相対	I 形式
BGEZ(L)	Branch On Greater Than or Equal To Zero (Likely)	*rs* ≥ 0	PC 相対	I 形式
BLTZ(L)	Branch On Less Than Zero (Likely)	*rs* < 0	PC 相対	I 形式
BLEZ(L)	Branch On Less Than or Equal To Zero (Likely)	*rs* ≤ 0	PC 相対	I 形式
BLTZAL(L)	Branch On Less Than Zero And Link (Likely)	*rs* < 0	PC 相対	I 形式
BGEZAL(L)	Branch On Greater Than or Equal To Zero And Link (Likely)	*rs* ≥ 0	PC 相対	I 形式

リンク機能付きジャンプ命令とリンク機能付き分岐命令では、レジスタ r31 に戻りアドレスが格納されます。これらの命令はサブルーチンコールで使われます。

ジャンプ命令と一般分岐命令では、ターゲット命令をメモリからフェッチしているあいだに、その直後に置かれた遅延スロットの命令が先に実行されます。これは、分岐するかしないかにかかわらず、すべての一般分岐命令であてはまります。これに対して、分岐ライクリ命令では、分岐条件が成立したときのみ、遅延スロットの命令が実行され、条件が成立しなかったときは、遅延スロットの命令は廃棄されます。遅延スロットについては、「5 章　CPU パイプライン」を参照してください。分岐ライクリ命令については、3.4.4項で説明します。

3.4.2　アドレッシングモード

表 3-4と表 3-5に示したように、ジャンプ命令、分岐命令、分岐ライクリ命令は、以下のアドレッシングモードを使って、ターゲット命令の実効アドレスを計算します。

- ページ内絶対アドレッシング
- レジスタ間接アドレッシング
- オフセット付き PC 相対アドレッシング

■　ページ内絶対アドレッシングモード

J、JAL、JALX 命令は、絶対アドレッシングモードを使って、ターゲットアドレスへ無条件にジャンプします。これらの命令では、指定された 26 ビットのオフセット値を左に 2 ビットシフトした値に、プログラムカウンタ (PC) の上位 4 ビットと連結した結果がターゲットアドレスになります。これを図 3-6に示します。図 3-6に示すように、ターゲットアドレスは、ジャンプ命令の直後の命令、すなわちジャンプ遅延スロットのアドレスから生成されます。PC の上位 4 ビットは 16 ページアドレス空間の特定のページを示します。

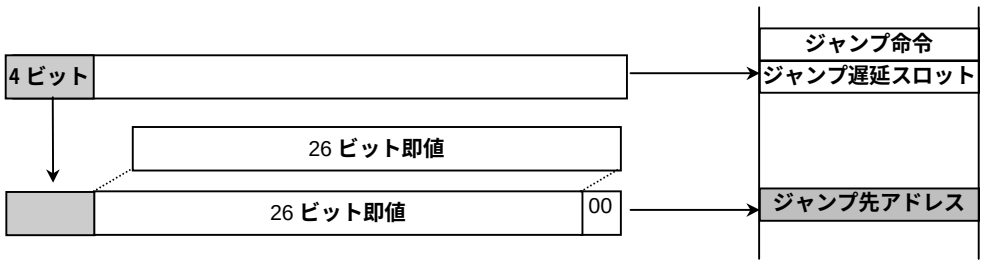


図 3-6　ページ内絶対アドレッシング (32 ビット ISA モード)

■　レジスタ間接アドレッシング

JR、JALR 命令は、汎用レジスタに格納されている 32 ビットの絶対アドレスに無条件にジャンプします。ジャンプ先のアドレスは指定されたターゲットレジスタの最下位ビットを 0 にマスクした値です。32 ビット ISA の命令はワード境界に位置合わせされるので、JR 命令や JALR 命令で指定するレジスタの値は、下位 2 ビットが 0 でなければなりません。

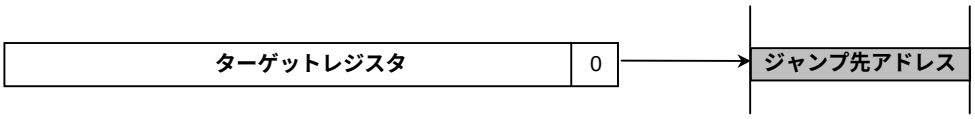


図 3-7　レジスタ間接アドレッシング (32 ビット ISA モード)

■　オフセット付 PC 相対アドレッシングモード

分岐命令、分岐ライクリ命令は、すべて PC 相対アドレッシングモードを使います。これらの命令では、指定された 16 ビットの即値 (オフセット) を 2 ビットシフトして符号拡張した値を、プログラムカウンタ (PC) の値に加算した結果が、ターゲットアドレスになります。図 3-8にこれを示します。図 3-8に示すように、分岐先のアドレスは分岐命令の直後の命令、すなわち分岐遅延スロットのアドレスから生成されます。

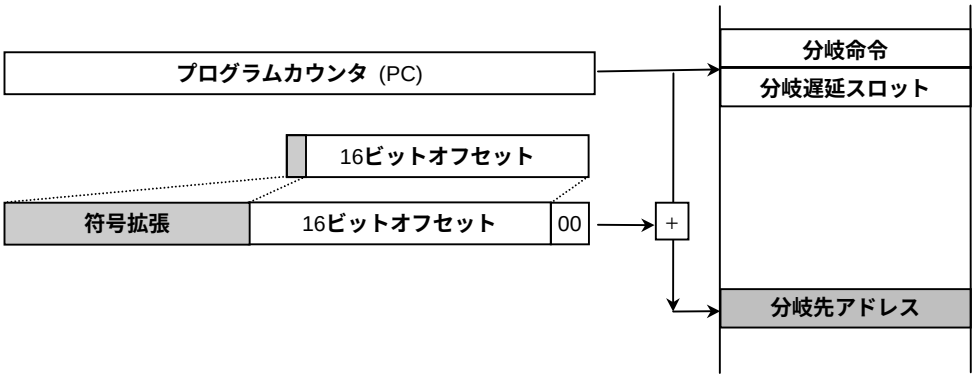


図 3-8　オフセット付き PC 相対アドレッシング (32 ビット ISA モード)

3.4.3　ISA モードの切り換え

TX19 には、16 ビット ISA モードと 32 ビット ISA モードの 2 つの ISA モードがあります。16 ビット ISA モードと 32 ビット ISA モードは、プログラムの実行中に、JALX、JR、JALR 命令により切り換えられます。プログラムカウンタ (PC) の最下位ビットが ISA モードビットで、0 のときは 32 ビット ISA モードになり、1 のときは 16 ビット ISA モードになります。JALX 命令では、ジャンプ後の PC の ISA モードビット (最下位ビット) が他の ISA モードに無条件で切り換えられます。JR 命令と JALR 命令では、ジャンプアドレスが格納されているレジスタの最下位ビットから ISA モードビットが設定されます。ジャンプアドレスは、ISA モードビットをゼロにマスクした値になります。

32 ビット ISA モードでは、命令はワード境界上に位置合わせしなければなりません。そのため、16 ビット ISA モードから 32 ビット ISA モードへ切り換えるときは、JR 命令または JALR 命令で指定するレジスタの値は下位 2 ビットが 0 でなければなりません。下位 2 ビットが 0 でないと、ジャンプ先の命令をフェッチするときに、アドレスエラー例外が発生します。

JALX 命令、JR 命令、JALR 命令のジャンプ遅延スロットにある命令は、ジャンプ前の ISA モードで実行されます。

リンク機能付きジャンプ命令、分岐命令、分岐ライクリ命令では、レジスタ r31 (ra) または指定されたデスティネーションレジスタ (*rd*) に、サブルーチンからの戻りアドレスが自動的に格納されます。レジスタの最下位ビットには、サブルーチンが実行された後の ISA モードが格納されます。

3.4.4　分岐ライクリ命令

ジャンプ命令、分岐命令では、ジャンプまたは分岐先の実効アドレスを計算し、命令をフェッチしなければならないので、プログラムのフローを変えるのに 2 命令の遅延が発生します。この遅延をジャンプ遅延、分岐遅延といいます。TX19 では、遅延スロットの処理をソフトウエアに任せており、ターゲット命令をメモリからフェッチしているあいだに、ジャンプまたは分岐直後の命令を実行するようにコンパイラまたはアセンブラにより命令の順序が再編成されます。

ジャンプ命令は、プログラムの流れを無条件に変更するので何の問題もありません。つまり、ジャンプ直後の命令を、常に遅延スロットに置くことができます。しかし、分岐命令では、プロセッサは分岐条件が成立するかどうか、あらかじめ知ることはできません。したがって、遅延スロットの命令は、プログラムの論理に影響を与えない命令でなければなりません。そのような命令がない場合は、NOP (No Operation) 命令で命令パイプラインを埋める必要があります。(NOP はアセンブラで受け付けられる疑似命令で、1 章で説明したように NOP はアセンブラにより 0 ビットのシフト命令に変換されます。)

図 3-9に、r8 の値が 0 か 1 かという判断に基づいて、レジスタ r2 を 0 または 1 に

設定するプログラムを示します。この例では一般分岐命令を使っています。ADDI 命令はプログラムの論理の上から BEQ 命令より先に実行できないので、BEQ 命令の直後に NOP が必要です。

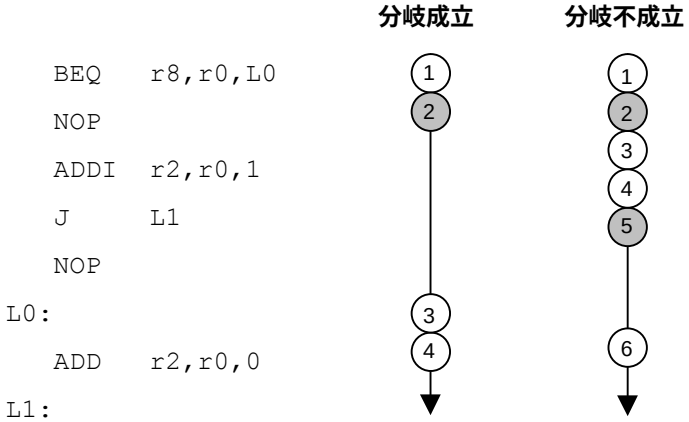


図 3-9　一般分岐命令

次に同様の処理を分岐ライクリ命令を使って実現した例を示します。図 3-10では、BEQ 命令の代わりに分岐ライクリ命令 BEQL (Branch On Equal Likely) を使っています。分岐ライクリ命令では、分岐条件が成立したとき、遅延スロットの命令が実行されます。分岐条件が成立しなかったときは、遅延スロットの命令は廃棄されます。このため、遅延スロットに NOP 命令を挿入する必要がなく、コードサイズの縮小、分岐処理の高速化に役立ちます。

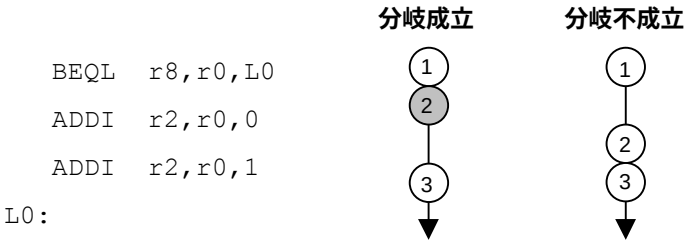


図 3-10　分岐ライクリ命令

3.4.5　大小関係に基づく分岐

2 つのレジスタの値を比較して、その結果に基づいて分岐する命令には、BEQ (Branch On Equal) 命令、BNE (Branch On Not Equal) 命令、およびそれらの分岐ライクリ命令 (BEQL・BNEL) しかありません。以下に例を示します。

```
BEQ r2,r3,Equal
```

上記の例では、レジスタ r2 の内容とレジスタ r3 の内容を比較し、両者が等しい場合、Equal に分岐します。しかし、r2 の内容が r3 の内容より大きいかどうかという判断に基づいて分岐する命令はありません。2 つのレジスタ、またはレジスタと即値

の比較を行うには、2 つの命令を組み合わせて使わなくてはなりません。以下に 3 つの例を示します。(アセンブラには、マクロ命令が用意されているものがあります。そのようなアセンブラを使うとマクロ命令を機械語命令に自動的に変換してくれるので、プログラミングの負荷を軽くすることができます。)

◆ 例 1　r6 ≥ r7 の場合の分岐
以下に r6 の値が r7 の値以上の場合に、分岐する例を示します。r6 が r7 より小さいと、SLT (Set On Less Than) 命令により、r24 が 1 に設定されます。r6 が r7 以上の場合は、r24 は 0 に設定されます。r24 の値が 0 の場合、BEQ 命令で Label に分岐します。(r0 はハードウエア的に 0 に固定されています。)

```
SLT     r24,r6,r7
BEQ     r24,r0,Label
```

◆ 例 2　r7 ≥ 0x1234 の場合の分岐
以下に、r7 の値が 0x1234 以上の場合に分岐する例を示します。この例では、SLTI (Set On Less Than Immediate) 命令により、r7 の値と 0x1234 が比較され、r7 のほうが大きい場合は、r24 が 0 に設定されます。

```
SLTI    r24,r7,0x1234
BEQ     r24,r0,Label
```

◆ 例 3　r7 ≠ 0x1234 の場合の分岐
以下に、レジスタの値と即値が等しいかどうか調べて、それに基づき分岐する例を示します。この例では、ORI (OR Immediate) 命令を使って、0x1234 を r10 に一時的に格納しています。次に BEQ 命令で r10 の値が r7 の値と等しいか調べています。

```
ORI     r10,r0,0x1234
BEQ     r10,r7,Label
```

3.4.6　32 ビットのアドレスへのジャンプ

3.4.2項で説明したように、ページ内絶対アドレッシングモードを使う J 命令、JAL 命令、JALX 命令は、最大 26 ビットです。26 ビットの即値は 2 ビット左シフトされるので、ターゲットアドレスは、2^{28} バイトセグメント内でなければなりません。任意の 32 ビットのアドレスへジャンプするには、LUI 命令と ORI 命令を使って、希望するアドレスをいったんレジスタに格納し、次に JR (Jump Register) 命令を使う必要があります。以下に、アドレス 0x76543210 へジャンプするための例を示します。

```
LUI     r8,0x7654
ORI     r8,0x3210
JR      r8
```

3.4.7　サブルーチンコール

32 ビット ISA モードには、リンク機能付きジャンプ命令 (JAL、JALX、JALR)、リンク機能付き分岐命令 (BLTZAL、BGEZAL)、リンク機能付き分岐ライクリ命令 (BLTZALL、BGEZALL) があります。これらの命令は、通常サブルーチンコールに使われ、サブルーチンの戻りアドレスがレジスタ r31 (ra) に格納されます。JALR (Jump-And-Link Register) 命令では、r31 以外の汎用レジスタ (*rd*) を使うこともできます。この戻りアドレスを格納するレジスタをリンクレジスタといいます。

戻りアドレスは、遅延スロットの直後の命令のアドレスになります。また、リンク機能付きのジャンプ命令では、リンクレジスタの最下位ビットに ISA モードが保存されます。

サブルーチンから復帰するには、JR 命令を使います。ISA モードビット (PC の最下位ビット) は、リンクレジスタの最下位ビットから復帰されます。

サブルーチンをネスティングする場合は、次のサブルーチンをコールするまえにリンクレジスタの戻りアドレスをスタック領域に退避させなければなりません。

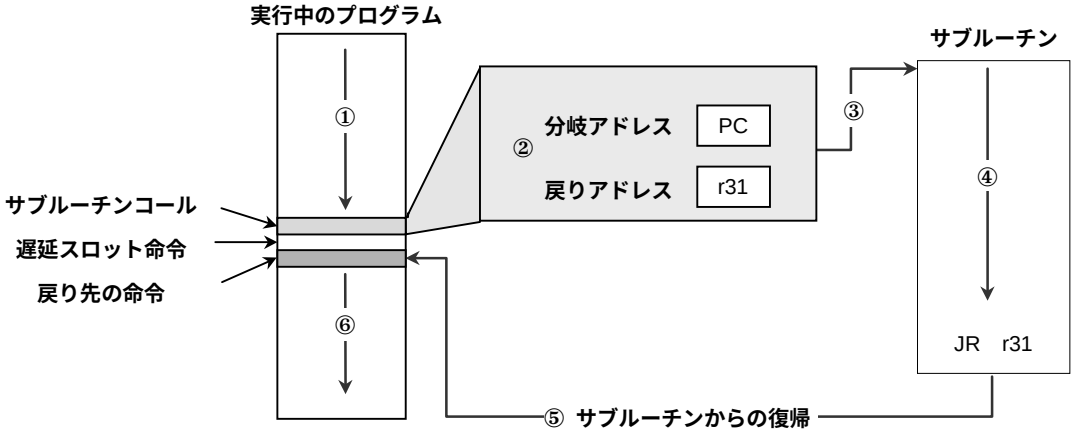


図 3-11　サブルーチンとコール

JAL 命令、JALX 命令を除くリンク機能付きジャンプ命令、分岐命令、分岐ライクリ命令は、ソースレジスタ (*rs*) フィールドをもっています。以下に例を示します。

```
BGEZAL r8,PSUB
```

この命令で、r8 はソースレジスタです。BGEZAL 命令は r8 の値が 0 以上かどうか調べ、その結果に基づき分岐します。

ジャンプ遅延スロットまたは分岐遅延スロットの命令を実行している最中に、例外や割り込みが発生すると、その命令の終了を待たずに例外処理が始まります。この場合、例外または割り込みが発生した命令の直前にあるジャンプ分岐、分岐命令、または分岐ライクリ命令のアドレスが例外プログラムカウンタ (EPC) に設定されます。

例外ハンドラの実行後は、ジャンプ命令、分岐命令、または分岐ライクリ命令から処理を再開しなければなりません。そのため、r31 (ra) をソースレジスタとして使用してはいけません。例外処理の手順については、「9 章　例外処理」を参照してください。

3.5　コプロセッサ命令

TX19 には、最大 4 つのコプロセッサ (CP0、CP1、CP2、CP3) を接続できます。「コプロセッサ命令に分類される命令」は、このうちコプロセッサ CP1~CP3 に対して使います。コプロセッサのロード・ストア命令は I 形式で、コプロセッサの演算命令の形式はコプロセッサごとに異なります。

CP1~CP3 に対するコプロセッサ命令を使えるかどうかは、Status レジスタの CU[3:1]ビットで制御します。動作モード (ユーザーモード・カーネルモード) にかかわらず、対応する CU ビットが 0 のときに、コプロセッサ命令を実行すると、コプロセッサ使用不可例外が発生します。

表 3-6に CP0 命令を除くコプロセッサ命令を示します。(表で *z* はコプロセッサ番号を示します。)

表 3-6　コプロセッサ命令 (32 ビット ISA)

命令	オペコード
Move To/From Coprocessor	MTC*z* • MFC*z*
Move Control To/From Coprocessor	CTC*z* • CFC*z*
Coprocessor Operation	COP*z*
Branch on Coprocessor *z* True/False	BC*z*T • BC*z*F
Branch on Coprocessor *z* True/False Likely	BC*z*TL • BC*z*FL

MIPS の R3000A にある LWC*z* (Load Word To Coprocessor) 命令と SWC*z* (Store Word From Coprocessor) 命令は、TX19 ではサポートされていません。これらのロード・ストア命令を実行しようとすると、予約命令例外が発生します。

システム制御コプロセッサ (CP0) 命令は、システム構成、メモリ管理、例外処理などの操作を行うための命令で、CP0 レジスタを操作します。そのため、CP0 にはある程度の特権保護が与えられています。ユーザーモードのときは、Stauts レジスタの CU[0]ビットが 1 に設定されていないと、CP0 レジスタにはアクセスできません。CU[0]ビットが 0 のときに CP0 命令を実行しようとすると、コプロセッサ使用不可例外が発生します。ただし、カーネルモードのときは、CU[0]ビットの設定に関係なく、すべての CP0 命令を実行できます。表 3-7に CP0 命令を示します。

表 3-7　システム制御コプロセッサ (CP0) 命令

命令名	オペコード
Move To/From CP0	MTC0・MFC0
Restore From Exception	RFE
Debug Exception Return	DERET
Cache Operation	CACHE

TX19 は、仮想アドレスから物理アドレスへの変換にダイレクトセグメントマッピング方式を採用しており、TLB (table lookaside buffer) をサポートしていません。

3.6　特殊命令

32 ビット ISA には 3 つ特殊命令が用意されていて、ソフトウエアにより例外を発生させることができます。特殊命令には SYSCALL (System Call)、BREAK (Breakpoint)、SDBBP (Software Debug Breakpoint) があり、すべて R 形式です。SDBBP は TX19 用に当社で拡張した命令で、MIPS の R3000A にはありません。特殊命令を実行すると、プログラムの処理は無条件に対応する例外ハンドラに移ります。例外処理の詳細については、6 章を参照してください。

3.7　命令の概要

この項では、32 ビット ISA の命令の概要を分類ごとに示します。

■　表記規則

この項では、命令中 *rt*、*rs*、*rd*、*immediate*、*sa* (シフト量: shift amount)などの小文字の斜体で示してある部分には、ユーザーが任意のレジスタや値などを指定できます。また、オペランドの意味がより明確になるように、例えば、ロード命令とストア命令では、*rs*、*immediate* と書かずに *base*、*offset* と記述してあります。HI と LO は、整数の乗算・除算の結果を格納する特殊レジスタです。

■　TX19 で拡張された命令

当社の TX39 や MIPS の R3000A アーキテクチャにはなく、TX19 で拡張されている命令があります。それらの命令には、この項でただし書きを添えてあります。詳細は付録 D を参照してください。

表 3-8　ロード・ストア命令 (32 ビット ISA)

命令	形式	説明
Load Byte	LB *rt*, *offset*(*base*)	実効アドレス (EA) は *base* + *offset* です。16 ビット *offset* を符号拡張します。EA でアドレス指定されたバイトデータを符号拡張し、*rt* にロードします。
Load Byte Unsigned	LBU *rt*, *offset*(*base*)	実効アドレス (EA) は *base* + *offset* です。16 ビット *offset* を符号拡張します。EA でアドレス指定されたバイトデータをゼロ拡張し、*rt* にロードします。
Load Halfword	LH *rt*, *offset*(*base*)	実効アドレス (EA) は *base* + *offset* です。16 ビット *offset* を符号拡張します。EA でアドレス指定されたハーフワードデータを符号拡張し、*rt* にロードします。
Load Halfword Unsigned	LHU *rt*, *offset*(*base*)	実効アドレス (EA) は *base* + *offset* です。16 ビット *offset* を符号拡張します。EA でアドレス指定されたハーフワードデータをゼロ拡張し、*rt* にロードします。
Load Word	LW *rt*, *offset*(*base*)	実効アドレス (EA) は *base* + *offset* です。16 ビット *offset* を符号拡張します。EA でアドレス指定されたワードデータを、*rt* にロードします。
Load Word Left	LWL *rt*, *offset*(*base*)	実効アドレス (EA) は *base* + *offset* です。16 ビット *offset* を符号拡張します。*rt* の左側に、EA でアドレス指定されたメモリワードの上位部分を格納します。
Load Word Right	LWR *rt*, *offset*(*base*)	実効アドレス (EA) は *base* + *offset* です。16 ビット *offset* を符号拡張します。*rt* の右側に、EA でアドレス指定されたメモリワードの下位部分を格納します。
Store Byte	SB *rt*, *offset*(*base*)	実効アドレス (EA) は *base* + *offset* です。16 ビット *offset* を符号拡張します。*rt* の最下位バイトを、このアドレスにストアします。
Store Halfword	SH *rt*, *offset*(*base*)	実効アドレス (EA) は *base* + *offset* です。16 ビット *offset* を符号拡張します。*rt* の下位のハーフワードを、このアドレスにストアします。
Store Word	SW *rt*, *offset*(*base*)	実効アドレス (EA) は *base* + *offset* です。16 ビット *offset* を符号拡張します。*rt* の内容を、このアドレスにストアします。
Store Word Left	SWL *rt*, *offset*(*base*)	実効アドレス (EA) は *base* + *offset* です。16 ビット *offset* を符号拡張します。*rt* の左側の内容を、EA でアドレス指定されたメモリワードの上位の部分にストアします。
Store Word Right	SWR *rt*, *offset*(*base*)	実効アドレス (EA) は *base* + *offset* です。16 ビット *offset* を符号拡張します。*rt* の右側の内容を、EA でアドレス指定されたメモリワードの下位部分にストアします。
Sync	SYNC	**この命令は R3000A にはありません。** 直前に実行したロードまたはストア命令が完了するまで、パイプラインをインタロックします。

表 3-9　ALU 即値命令 (32 ビット ISA)

命令	形式	説明
Add Immediate	ADDI *rt*, *rs*, *immediate*	*rs* + *immediate* を *rt* に格納します。16 ビット *immediate* を符号拡張します。2 の補数のオーバフローで例外が発生します。
Add Immediate Unsigned	ADDIU *rt*, *rs*, *immediate*	*rs* + *immediate* を *rt* に格納します。16 ビット *immediate* を符号拡張します。2 の補数のオーバフローでも例外が発生しません。
Set On Less Than Immediate	SLTI *rt*, *rs*, *immediate*	*rs* が *immediate* より小さい場合は、*rt* に 1 を、大きい場合は 0 を格納します。16 ビット *immediate* を符号拡張します。*rs* と *immediate* を符号付き整数として比較します。
Set On Less Than Immediate Unsigned	SLTIU *rt*, *rs*, *immediate*	*rs* が *immediate* より小さい場合は、*rt* に 1 を、大きい場合は 0 を格納します。16 ビット *immediate* を符号拡張します。*rs* と *immediate* を符号なし整数として比較します。
AND Immediate	ANDI *rt*, *rs*, *immediate*	*rs* の内容と *immediate* の AND をとり、結果を *rt* に格納します。16 ビット *immediate* をゼロ拡張します。
OR Immediate	ORI *rt*, *rs*, *immediate*	*rs* の内容と *immediate* の OR をとり、結果を *rt* に格納します。16 ビット *immediate* をゼロ拡張します。
Exclusive-OR Immediate	XORI *rt*, *rs*, *immediate*	*rs* の内容と *immediate* の排他的 OR をとり、結果を *rt* に格納します。16 ビット *immediate* をゼロ拡張します。
Load Upper Immediate	LUI *rt*, *immediate*	16 ビット *immediate* を 16 ビット左にシフトし、下位 16 ビットの 0 と連結して、結果を *rt* に格納します。

表 3-10　3 オペランドレジスタタイプ命令 (32 ビット ISA)

命令	形式	説明
Add	ADD *rd*, *rs*, *rt*	*rs* + *rt* の和を *rd* に格納します。2 の補数のオーバフローで例外が発生します。
Add Unsigned	ADDU *rd*, *rs*, *rt*	*rs* + *rt* の和を *rd* に格納します。2 の補数のオーバフローでも例外が発生しません。
Subtract	SUB *rd*, *rs*, *rt*	*rs* – *rt* の差を *rd* に格納します。2 の補数のオーバフローで例外が発生します。
Subtract Unsigned	SUBU *rd*, *rs*, *rt*	*rs* – *rt* の差を *rd* に格納します。2 の補数のオーバフローでも例外が発生しません。
Set On Less Than	SLT *rd*, *rs*, *rt*	*rs* が *rt* より小さい場合は、*rd* に 1 を、そうでない場合は、*rd* に 0 を格納します。*rs* と *rt* を符号付き整数として比較します。
Set On Less Than Unsigned	SLTU *rd*, *rs*, *rt*	*rs* が *rt* より小さい場合は、*rd* に 1 を、そうでない場合は、*rd* に 0 を格納します。*rs* と *rt* を符号付なし整数として比較します。
AND	AND *rd*, *rs*, *rt*	*rs* の内容と *rt* の内容の AND をとり、結果を *rd* に格納します。
OR	OR *rd*, *rs*, *rt*	*rs* の内容と *rt* の内容の OR をとり、結果を *rd* に格納します。
Exclusive-R	XOR *rd*, *rs*, *rt*	*rs* の内容と *rt* の内容の排他的 OR をとり、結果を *rd* に格納します。
NOR	NOR *rd*, *rs*, *rt*	*rs* の内容と *rt* の内容の NOR をとり、結果を *rd* に格納します。

表 3-11　シフト命令 (32 ビット ISA)

命令	形式		説明
Shift Left Logical	SLL	*rd*, *rt*, *sa*	*rt* の内容を *sa* ビット左へシフトし、右端の空いたビットを 0 で埋めます。結果を *rd* に格納します。
Shift Left Logical Variable	SLLV	*rd*, *rt*, *rs*	*rt* の内容を *rs* の下位 5 ビットで指定されたビット数、左にシフトし、右端の空いたビットを 0 で埋めます。結果を *rd* に格納します。
Shift Right Logical	SRL	*rd*, *rt*, *sa*	*rt* の内容を *sa* ビット右にシフトし、左端の空いたビットを 0 で埋めます。結果を *rd* に格納します。
Shift Right Logical Variable	SRLV	*rd*, *rt*, *rs*	*rt* の内容を *rs* の最下位 5 ビットで指定されたビット数、右にシフトし、左端の空いたビットを 0 で埋めます。結果を *rd* に格納します。
Shift Right Arithmetic	SRA	*rd*, *rt*, *sa*	*rt* の内容を *sa* ビット右にシフトし、左端の空いたビットを符号ビットで埋めます。結果を *rd* に格納します。
Shift Right Arithmetic Variable	SRAV	*rd*, *rt*, *rs*	*rt* の内容を *rs* の下位 5 ビットで指定されたビット数、右にシフトし、左端の空いたビットを符号ビットで埋めます。結果を *rd* に格納します。

表 3-12　乗算・除算命令 (32 ビット ISA)

命令	形式		説明
Multiply	MULT	(*rd*,) *rs*, *rt*	***rd* は当社で拡張した部分です。R3000A にはありません。** 被乗数は *rs* の符号付き整数です。乗数は *rt* の符号付き整数です。64 ビットの積 *rs* * *rt* を HI レジスタと LO レジスタに格納します。また、下位 32 ビットを任意で *rd* に格納します。
Multiply Unsigned	MULTU	(*rd*,) *rs*, *rt*	***rd* は当社で拡張した部分です。R3000A にはありません。** 被乗数は *rs* の符号なし整数です。乗数は *rt* の符号なし整数です。64 ビットの積 *rs* * *rt* を HI レジスタと LO レジスタに格納します。また、下位 32 ビットを任意で *rd* にコピーします。
Divide	DIV	*rs*, *rt*	被除数は *rs* の符号付き整数です。除数は *rt* の符号付き整数です。商を LO レジスタに、剰余を HI レジスタに格納します。
Divide Unsigned	DIVU	*rs*, *rt*	被除数は *rs* の符号なし整数です。除数は *rt* の符号なし整数です。商を LO レジスタに、剰余を HI レジスタに格納します。
Move From HI	MFHI	*rd*	HI レジスタの内容を *rd* にロードします。
Move From LO	MFLO	*rd*	LO レジスタの内容を *rd* にロードします。
Move To HI	MTHI	*rs*	*rs* の内容を HI レジスタにロードします。
Move To LO	MTLO	*rs*	*rs* の内容を LO レジスタにロードします

表 3-13　積和命令 (32 ビット ISA)

命令	形式	説明
Multiply-and-Add	MADD (*rd*,) *rs*, *rt*	**この命令は R3000A にはありません。** 被乗数は *rs* の符号付き整数です。乗数は *rt* の符号付き整数です。64 ビットの積 *rs* * *rt* を HI レジスタ・LO レジスタに格納されている 64 ビットの値に加算し、その結果を HI レジスタと LO レジスタに格納します。また、結果の下位 32 ビットを *rd* に格納します。
Multiply-and-Add Unsigned	MADDU (*rd*,) *rs*, *rt*	**この命令は R3000A にはありません。** 被乗数は *rs* の符号なし整数です。乗数は *rt* の符号なし整数です。64 ビットの積 *rs* * *rt* を HI レジスタ・LO レジスタに格納されている 64 ビットの値に加算し、その結果を HI レジスタと LO レジスタに格納します。また、結果の下位 32 ビットを *rd* に格納します。

表 3-14　ジャンプ命令 (32 ビット ISA)

命令	形式	説明
Jump	J *target*	ページ内絶対アドレッシングモードを使ってジャンプします。つまり、26 ビット *target* を 2 ビット左へシフトし、PC + 4 の上位 4 ビットと連結した結果がターゲットアドレスになります。
Jump And Link	JAL *target*	ページ内絶対アドレッシングモードを使ってジャンプします。つまり、26 ビット *target* を 2 ビット左へシフトし、PC + 4 の上位 4 ビットと連結した結果がターゲットアドレスになります。また、遅延スロットの後続命令のアドレスを、r31 に格納します。
Jump And Link eXchange	JALX *target*	**この命令は TX39、R3000A にはありません。** ページ内絶対アドレッシングモードを使ってジャンプします。つまり、26 ビット *target* を 2 ビット左へシフトし、PC + 4 の上位 4 ビットと連結した結果がターゲットアドレスになります。また、遅延スロットの後続命令のアドレスを、r31 に保存します。PC の ISA モードビットが切り換わります。
Jump Register	JR *rs*	*rs* の上位 31 ビットで指定されたアドレスへジャンプします。*rs* の最下位のビットの値によって、ISA モードが切り換わります。
Jump And Link Register	JALR (*rd*,) *rs*	*rs* の上位 31 ビットで指定されたアドレスへジャンプします。*rs* の最下位のビットの値によって、ISA モードが切り換わります。また、遅延スロットの後続命令のアドレスを *rd* に保存します。*rd* を省略すると、デフォルトで r31 が使用されます。

表 3-15　分岐・分岐ライクラリ命令 (32 ビット ISA)

命令	形式	説明
Branch On Equal (Likely)	BEQ(L)　*rs*, *rt*, *offset*	**BEQL 命令は R3000A にはありません。** *rs* = *rt* ならば、PC + 4 (遅延スロットのアドレス) に対して相対的に計算された 16 ビット *offset* に分岐します。
Branch On Not Equal (Likely)	BNE(L)　*rs*, *rt*, *offset*	**BNEL 命令は R3000A にはありません。** *rs* ≠ *rt* ならば、PC + 4 (遅延スロットのアドレス) に対して相対的に計算された 16 ビット *offset* に分岐します。
Branch On Greater Than Zero (Likely)	BGTZ(L)　*rs*, *offset*	**BGTZL 命令は R3000A にはありません。** *rs* > 0 ならば、PC + 4 (遅延スロットのアドレス) に対して相対的に計算された 16 ビット *offset* に分岐します。
Branch On Greater Than or Equal to Zero (Likely)	BGEZ(L)　*rs*, *offset*	**BGEZL 命令は R3000A にはありません。** *rs* ≥ 0 ならば、PC + 4 (遅延スロットのアドレス) に対して相対的に計算された 16 ビット *offset* に分岐します。
Branch On Less Than Zero (Likely)	BLTZ(L)　*rs*, *offset*	**BLTZL 命令は R3000A にはありません。** *rs* < 0 ならば、PC + 4 (遅延スロットのアドレス) に対して相対的に計算された 16 ビット *offset* に分岐します。
Branch On Less Than or Equal to Zero (Likely)	BLEZ(L)　*rs*, *offset*	**BLEZL 命令は R3000A にはありません。** *rs* ≤ 0 ならば、PC + 4 (遅延スロットのアドレス) に対して相対的に計算された 16 ビット *offset* に分岐します。
Branch On Less Than Zero And Link (Likely)	BLTZAL(L)　*rs*, *offset*	**BLTZALL 命令は R3000A にはありません。** *rs* < 0 ならば、PC + 4 (遅延スロットのアドレス) に対して相対的に計算された 16 ビット *offset* に分岐します。また、遅延スロットの後続命令のアドレスを r31 に格納します。
Branch On Greater Than or Equal to Zero And Link (Likely)	BGEZAL(L)　*rs*, *offset*	**BGEZALL 命令は R3000A にはありません。** *rs* ≥ 0 ならば、PC + 4 (遅延スロットのアドレス) に対して相対的に計算された 16 ビット *offset* に分岐します。また、遅延スロットの後続命令のアドレスを r31 に格納します。

† カッコ内に示したオペコードの接尾辞「L」は分岐ライクリ命令を表します。

表 3-16　コプロセッサ命令 (32 ビット ISA)

命令	形式	説明
Move To Coprocessor	MTC*z*　*rt*, *rd*	汎用レジスタ *rt* の内容を、コプロセッサユニット *z* のコプロセッサレジスタ *rd* にロードします。
Move From Coprocessor	MFC*z*　*rt*, *rd*	コプロセッサユニット *z* のコプロセッサレジスタ *rd* の内容を、汎用レジスタ *rt* にロードします。
Move Control To Coprocessor	CTC*z*　*rt*, *rd*	汎用レジスタ *rt* の内容を、コプロセッサユニット *z* のコプロセッサ制御レジスタ *rd* にロードします。
Move Control From Coprocessor	CFC*z*　*rt*, *rd*	コプロセッサ *z* のコプロセッサ制御レジスタ *rd* の内容を、汎用レジスタ *rt* にロードします。
Coprocessor Operation	COP*z*　*cofun*	コプロセッサユニット *z* は *cofun* で指定された操作を実行します。
Branch On Coprocessor *z* True (Likely)	BC*z*T(L)　*offset*	コプロセッサ *z* の条件信号が真ならば、PC + 4 (分岐遅延スロットのアドレス) に対して相対的に計算された 16 ビット *offset* に分岐します。
Branch On Coprocessor *z* False (Likely)	BC*z*F(L)　*offset*	コプロセッサ *z* の条件信号が偽ならば、PC + 4 (分岐遅延スロットのアドレス) に対して相対的に計算された 16 ビット *offset* に分岐します。

† カッコ内に示したオペコードの接尾辞「L」は分岐ライクリ命令を表します。

表 3-17　システム制御プロセッサ (CP0) 命令 (32 ビット ISA)

命令	形式	説明
Move To CP0	MTC0　*rt*, *rd*	**この命令は R3000A にはありません。** 汎用レジスタ *rt* の内容を CP0 レジスタ *rd* にロードします。
Move From CP0	MFC0　*rt*, *rd*	**この命令は R3000A にはありません。** CP0 レジスタ *rd* の内容を汎用レジスタ *rt* にロードします。
Restore From Exception	RFE	**この命令は R3000A にはありません。** Status レジスタの旧ステータスビット (割り込みイネーブルビット・動作モードビット) を前ステータスビットに復元し、前ステータスビットを現ステータスビットに復元します。また、前割り込みマスクレベルフィールドの値を現フィールドに復元します。
Debug Exception Return	DERET	**この命令は R3000A にはありません。** プログラム制御は、デバッグ例外処理プログラムよりユーザープログラムに戻ります。DEPC レジスタの戻りアドレスを PC に復元しま
Cache	CACHE *op*, *offset*(*base*)	**この命令は R3000A にはありません。** 仮想アドレスは *offset* + *base* で、仮想アドレスを物理アドレスに変換し、*op* はこの物理アドレスに対するキャッシュ動作を指定します。

表 3-18　特殊命令 (32 ビット ISA)

命令	形式	説明
System Call	SYSCALL *code*	システムコール例外が発生し、無条件で例外ハンドラの処理に移ります。
Breakpoint	BREAK　*code*	ブレークポイント例外が発生し、無条件で例外ハンドラの処理に移ります。
Software Debug Breakpoint Exception	SDBBP　*code*	**この命令は R3000A アーキテクチャにはありません。** デバッグブレークポイント例外が発生し、無条件で例外ハンドラの処理に移ります。

第4章　16 ビット ISA の概要

この章では、16 ビット ISA モードの命令とアドレッシングモードの概要を説明します。また、16 ビット命令を使った基本的なプログラミングについても説明します。16 ビット ISA の命令は以下のように分類されます。分岐ライクリ命令とコプロセッサ命令は、16 ビット ISA ではサポートされていません。

- ロード命令・ストア命令
- 演算命令
- ジャンプ命令・分岐命令
- 特殊命令

MIPS16 ASE のダブルワード命令は、TX19 では使用できません。

16 ビット ISA 命令では、32 本の汎用レジスタのうち、通常、r2~r7、r16、r17 の 8 本のレジスタしか使用できません。ただし、CPU 自体には、32 本の汎用レジスタが含まれているので、16 ビット ISA には、16 ビット ISA レジスタで通常アクセスできる 8 本のレジスタと 32 本のレジスタ間でデータを転送するための MOVE 命令があります。また、命令によっては、r24 (t8)、r29 (sp)、r31 (ra) を暗黙的に使用しています。r24 は比較結果を格納するレジスタで、16 ビット ISA では t8 (コンディションコードレジスタ) と呼ばれています。r29 はスタックポインタレジスタとして、r31 はリンクレジスタとして使われます。また、乗算・除算命令は特殊レジスタの HI レジスタと LO レジスタを使います。

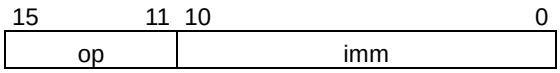
4.1　命令形式

16 ビット ISA の命令は、JAL、JALX 命令を例外として、そのほかの命令はすべて 16 ビット長です。16 ビットの命令には基本的に図 4-1に示す 10 種類の命令形式があります。32 ビットの JAL、JALX 命令は、図 4-2に示す JAL・JALX 形式になります。

16 ビット命令では、命令長が 16 ビットしかないため、即値フィールドは 4~11 ビットに制約されます。ただし、16 ビット ISA には、これを補うための拡張命令があります。各命令形式で割り当てられているフィールドを超える即値を指定すると、アセンブラにより、その命令のまえに EXTEND という命令が自動的に付加されます。EXTEND 命令はオペコードと即値のみで構成される命令で、それ自体だけでは機械語命令は生成されませんが、自身の即値フィールドを直後の命令の即値フィールドに連結することにより、32 ビット ISA と同様の最大 16 ビットの即値を扱うことができるようになります。この場合、EXTEND が付加された命令 (拡張命令) は全体で 32 ビットになり、図 4-2に示す命令形式があります。例えば、I 形式の命令を拡張した命令は EXT-I 形式になります。

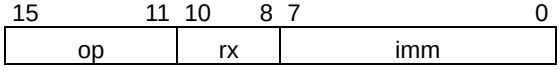
16 ビット命令

I 形式



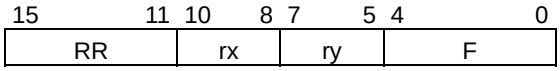
op:　B

RI 形式

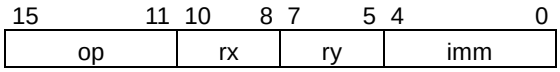


op:　ADDIU8 • ADDIUPC • ADDIUSP • BEQZ • BNEZ • CMPI • LI • LWPC • LWSP • SLTI • SLTIU SWSP

RR 形式

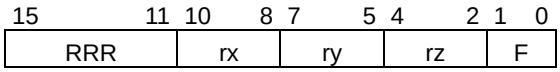


RRI 形式

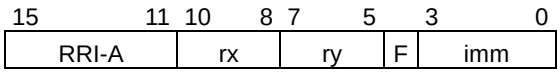


op:　LB • LBU • LH • LHU • LW • SB • SH • SW

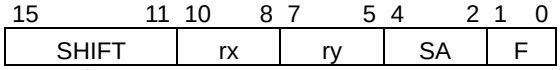
RRR 形式



RRI-A 形式

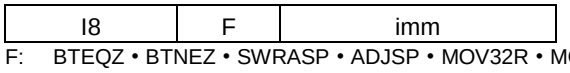


SHIFT 形式



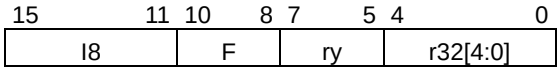
SA:　3 ビットの *sa* フィールドは、1 ~ 8 のシフト量を示します。16 ビット ISA では、
8 ビットのシフトのとき 0 を設定します。

I8 形式

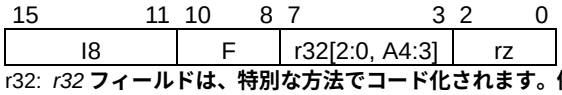


F:　BTEQZ • BTNEZ • SWRASP • ADJSP • MOV32R • MOVR32

I8_MOVR32 形式



I8_MOV32R 形式



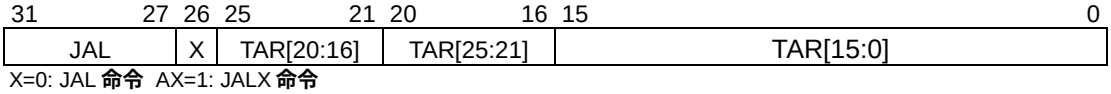
r32:　*r32* フィールドは、特別な方法でコード化されます。例えば、レジスタ r7 (00111)をコード化すると、
r32 フィールドでは 11100 になります。

op	5 ビットの命令コード
rx	3 ビットのソースレジスタ番号・デスティネーションレジスタ番号
ry	3 ビットのソースレジスタ番号・デスティネーションレジスタ番号
immediate または imm	4、5、8、11 ビットの即値、分岐オフセット値またはアドレスのオフセット値
rz	3 ビットのソースレジスタ番号・デスティネーションレジスタ番号
F	1、2、3、5 ビットの機能コード
r32	32 ビットの ISA 汎用レジスタ番号

図 4-1　16 ビット命令形式

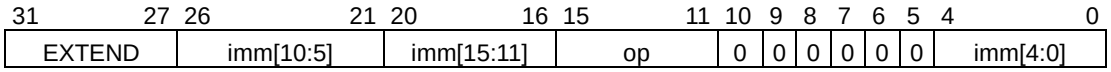
16 ビット命令

JAL・JALX 形式

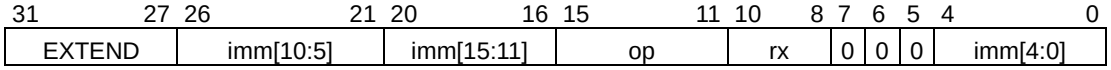


X=0: JAL 命令　AX=1: JALX 命令

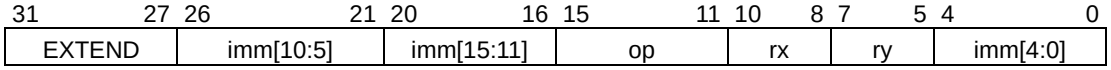
EXT-I 形式



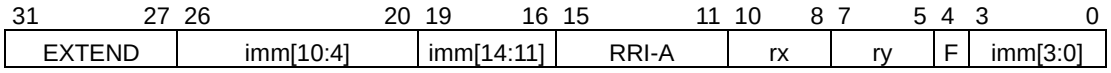
EXT-RI 形式



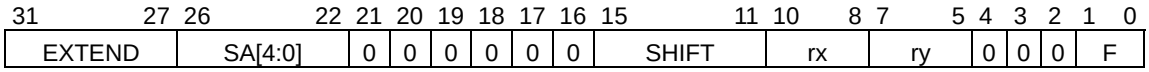
EXT-RRI 形式



EXT-RRI-A 形式



EXT-SHIFT 形式



EXT-I8 形式

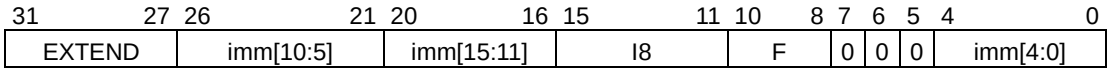


図 4-2　32 ビット命令形式

4.2　ロード・ストア命令

16 ビット ISA には、位置合わせされていないデータ用のロード・ストア命令、および SYNC 命令がありません。16 ビット ISA のロード・ストア命令では、表現できる即値 (オフセット) の大きさが、符号なしの 5 ビットまたは 8 ビットに制限されます。この制限を越えるオフセットを指定すると、オフセットフィールドは EXTEND 命令により、符号付きの 16 ビットに拡張されます。EXTEND 命令については、「4.5 特殊命令」を参照してください。また、16 ビット ISA には、コード中に埋め込まれた 32 ビットの定数をロードするためのアドレッシングモードが追加されています。

4.2.1項では、16 ビットロード・ストア命令でサポートされているアドレッシングモードについて説明します。4.2.2項では、ロード・ストア命令の概要を説明します。4.2.3項では、新しく追加されたアドレッシングモードを使って、32 ビットのアドレスを取得する方法について説明します。

4.2.1　アドレッシングモード

16 ビット ISA のロード・ストア命令では、以下の 3 つのアドレッシングモードがサポートされています。

- ◆ オフセット付きレジスタ間接アドレッシングモード
- ◆ オフセット付き SP 相対アドレッシングモード
- ◆ オフセット付き PC 相対アドレッシングモード

■ オフセット付きレジスタ間接アドレッシングモード

16 ビット ISA では、ほとんどのロード・ストア命令は、オフセット付きレジスタ間接アドレッシングモードを使います。命令形式は RRI (レジスタ・レジスタ・即値) 形式になります。このアドレッシングモードを使う命令は、ベースレジスタと符号なし 5 ビットオフセットフィールドをもっています。実効アドレスは、ベースレジスタとして指定した汎用レジスタの値に、5 ビットのオフセット値をゼロ拡張した値を加算することにより生成されます。ベースレジスタには、16 ビット ISA で通常アクセス可能な汎用レジスタ (r2～r7、r16、r17) ならどれでも使用できます。16 ビット ISA では、少しでもオフセットの範囲を広くとれるように、オフセット値は、ロードまたはストアするデータ形式にあわせて、左にシフトされます。すなわち、ワードアクセスの場合は、オフセットは 2 ビット左にシフトされ、ハーフワードアクセスの場合は、オフセットは 1 ビット左にシフトされます。

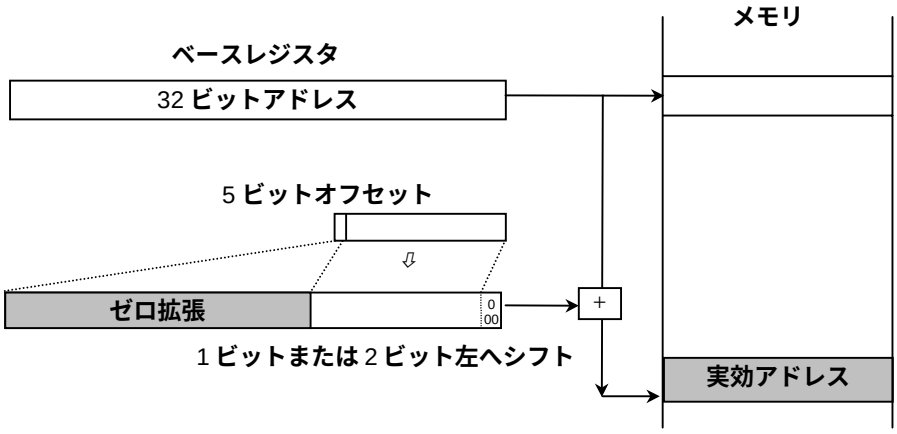


図 4-3　オフセット付きレジスタ間接アドレッシングモード (16 ビット ISA)

■ オフセット付き SP 相対アドレス

32 ビット ISA では、慣例的に r29 をスタックポインタレジスタとして使いますが、ハードウエア的には r0 以外の汎用レジスタならどのレジスタでも使うことができます。それに対して、16 ビット ISA では、r29 が常にスタックポインタとして使われます。r29 は別名で sp と呼ばれています。16 ビット ISA では、r29 を特殊な機能コ

ードを使って暗黙的に参照しているため、命令コード中にベースレジスタフィールドがありません。このため、オフセットフィールドを 8 ビットとることができ、命令形式は RI (レジスタ・即値) 形式になります。実効アドレスは、スタックポインタレジスタ sp の内容に、8 ビットのオフセットを 2 ビット左へシフトし、ゼロ拡張した値を加算することにより生成されます。SP 相対アドレッシングモードは LW (Load Word) 命令と SW (Store Word) 命令で使用できます。これらの命令は、EXTEND 命令で拡張しなくても 1K バイト (2^{10}) の範囲までアドレッシングできます。

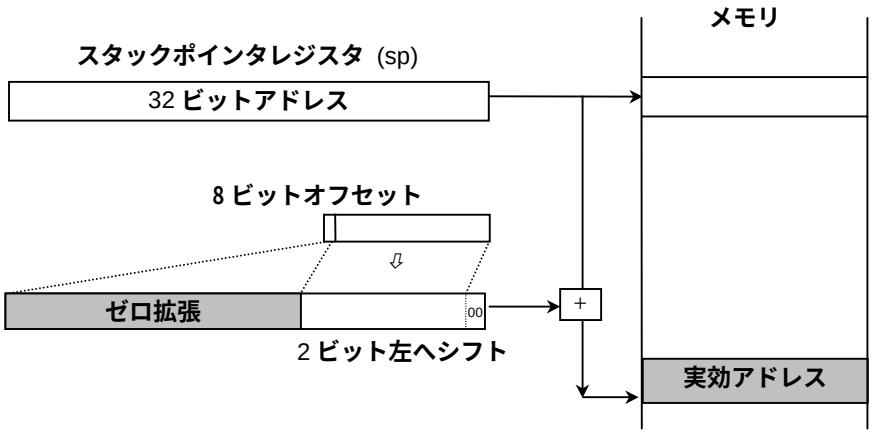


図 4-4　SP 相対アドレッシングモード (16 ビット ISA)

■　オフセット付き PC 相対アドレッシングモード

オフセット付き PC 相対は、LW (Load Word) 命令でサポートされているアドレッシングモードです。実効アドレスは、8 ビットのオフセットを 2 ビット左へシフトし、ゼロ拡張した値を、下位 2 ビットを 0 にマスクした PC の値に加算することにより生成されます。実効アドレスのメモリ位置に格納されている 32 ビットの定数が、レジスタにロードされます。これにより、32 ビットの定数を、コード中に埋めこむことができます。

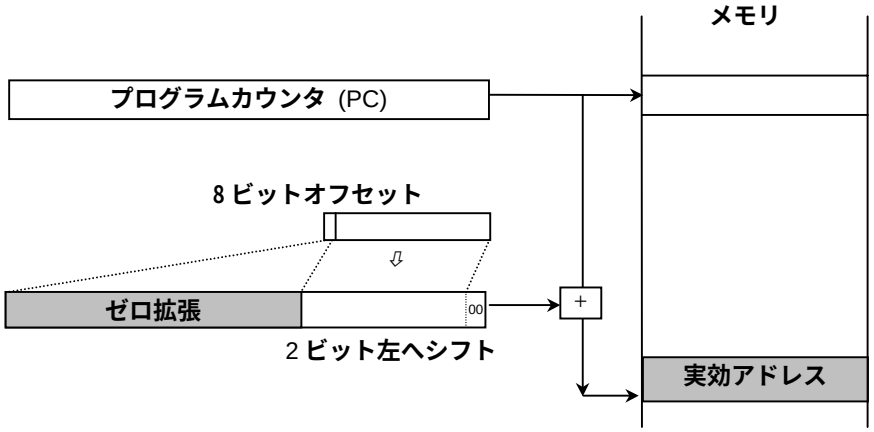


図 4-5　オフセット付き PC 相対アドレッシングモード (16 ビット ISA)

4.2.2　ロード・ストア命令の概要

表 4-1と表 4-2にバイトアクセス、ハーフワードアクセス、ワードアクセス用のロード・ストア命令を示します。LB、LH 命令では、ロードしたバイトまたはハーフワードは符号拡張されて、レジスタに格納されます。LBU、LHU 命令では、ロードしたバイトまたはハーフワードはゼロ拡張されて、レジスタに格納されます。

バイトアクセス、ハーフワードアクセス用のロード・ストア命令は、オフセット付きレジスタ間接アドレッシングモードを使います。ワードアクセス用のロード・ストア命令では、SW 命令で PC 相対アドレッシングがないという点を除いて、前項で説明したすべてのアドレッシングモードを使えます。

表 4-1　ロード命令

データ形式	符号なしロード	符号付きロード	アドレッシング
バイト	LBU	LB	レジスタ間接
ハーフワード	LHU	LH	レジスタ間接
ワード	LW	—	レジスタ間接 SP 相対 PC 相対

表 4-2　ストア命令

データ形式	符号なしロード	アドレッシング
バイト	SB	レジスタ間接
ハーフワード	SH	レジスタ間接
ワード	SW	レジスタ間接 SP 相対

4.2.3　32 ビットのアドレスの生成

16 ビット ISA のロード・ストア命令では、オフセットフィールドは 5 ビットまたは 8 ビットしかありません。EXTEND 命令により拡張すると、32 ビット ISA と同様の符号付きの 16 ビットのオフセット値 (–32768 ~ +32767) が扱えるようになります。ただし、オフセットがこの範囲外の場合は、オフセットをいったん汎用レジスタに格納する必要があります。ワードロードの場合は、オフセット付き PC 相対アドレッシングモードを使用できます。3 つの例を以下に示します。

◆ 例 1　ベースアドレス + 32 ビットオフセット
以下の命令では、ADDU (Add Unsigned) 命令によりレジスタ r5 に格納されているオフセットをレジスタ r4 のベースアドレスに加算し、その結果を r4 に書き戻しています。次に LW 命令で、r4 をベースレジスタとして指定しています。

```
ADDU    r4,r4,r5
LW      r6,0(r4)
```

◆ 例 2　ベースアドレス+ 32 ビットオフセット

3 章で説明したように、オフセット値が 16 ビット以上の場合、32 ビット ISA では、LUI (Load Upper Immediate) 命令を使って、指定した 16 ビットの即値をレジスタの上位 16 ビットにロードし、次に加算命令により下位 16 ビットを連結します。ところが、16 ビット ISA には LUI 命令がありません。その代わりに、16 ビット ISA には、PC 相対アドレッシングモードがあります。次の例で、最初の LW 命令により指定されるメモリ位置には 32 ビットのオフセット値が格納されています。この命令で r5 にオフセットがロードされます。次に、ADDU 命令で、r4 に格納されているベースアドレスにオフセットを加算することにより、実効アドレスを生成しています。最後の LW 命令では、r4 をベースレジスタとして使えば、オフセットをゼロとすることができます。

```
LW      r5,16(pc)
ADDU    r4,r4,r5
LW      r6,0(r4)
```

◆ 例 3　任意の 32 ビットの絶対アドレス指定

以下の例で、最初の LW 命令は、PC 相対アドレッシングモードを使ってメモリから 32 ビットの絶対アドレスをロードしています。次の LW 命令では、r4 をベースレジスタとして使えば、オフセットをゼロとすることができます。

```
LW      r4,16(pc)
LW      r6,0(r4)
```

4.3　演算命令

この項では、16 ビット ISA の演算命令について説明します。4.3.1項では演算命令の分類と、16 ビット ISA と 32 ビット ISA の違いについて説明します。4.3.2項では 32 ビット定数を使用する演算について説明します。64 ビットの加算、減算、ローテートについては､32 ビット ISA と 16 ビット ISA で同じ手法を使えるので、「3 章 32 ビット ISA 概要」を参照してください。

4.3.1　演算命令の分類

16 ビット ISA の演算命令は、表 4-3に示す 4 つのグループに分類されます。演算命令には、算術、比較、論理、シフト、乗算、除算命令があります。積和演算命令は、16 ビット ISA にはありません。また、TX19 の 16 ビット ISA は、MIPS16 のダブルワード命令をサポートしていません。

表 4-3　演算命令

分類	命令	オペコード
ALU 即値	加算	ADDIU
	大小比較	SLTI・SLTIU
	一致比較	CMPI
	即値のロード	LI
レジスタタイプ	加算	ADDU
	減算	SUBU
	大小比較	SLT・SLTU
	一致比較	CMP
	否定	NEG
	論理積	AND
	論理和	OR
	排他的論理和	XOR
	反転	NOT
	移動	MOVE
シフト	論理シフト	SLL・SLLV・SRL・SRLV
	算術シフト	SRA・SRAV
乗算・除算	乗算	MULT・MULTU
	除算	DIV・DIVU
	HI・LO レジスタと汎用レジスタ間の転送	MFHI・MFLO

ALU 即値命令では、ソースオペランドは、汎用レジスタと 5 ビットまたは 8 ビットの即値です。16 ビット ISA には、ANDI、ORI、XORI のような即値をとる論理演算命令がありません。また、新しく追加された命令として、一致比較命令 CMPI があります。CMPI 命令は、汎用レジスタ *(rs)* の内容とゼロ拡張された即値の排他的 OR をとり、その結果をレジスタ t8 (r24) に格納します。また、LI 命令は、指定された即値をゼロ拡張して、レジスタにロードします。

ADDIU 命令を除き、ALU 即値命令の 5 ビットまたは 8 ビットの即値はゼロ拡張されます。ただし、EXTEND 命令により拡張された場合は、32 ビット ISA と同様、符号付き 16 ビットの即値として扱われます。

レジスタタイプ命令は、2 つの汎用レジスタに格納されている値を対象に演算をし、その結果を汎用レジスタに格納します。オペランドとして 2 つのレジスタをとる RR 形式の命令と、3 つのレジスタをとる RRR 形式の命令があります。16 ビット ISA では、オペコードフィールドが 32 ビット ISA の 6 ビットから 5 ビットに縮められているため、例外を発生する算術命令が削られています。その代わりに、16 ビット ISA には、CMP、NEG、NOT 命令があります。CMP 命令は 2 つのレジスタの値を比較する命令です。NEG 命令はレジスタ値の 2 の補数をとる命令です。NOT 命令はレジスタ値の 1 の補数をとる命令です。また、16 ビット ISA には、16 ビット ISA で通常使用できる 8 本のレジスタと、32 本のレジスタの間で値を移送するための MOVE 命令が用意されています。

上記の LI (Load Immediate)、NEG (Negate)、NOT (Not) 命令のオペレーションは、32 ビット ISA では、ソースレジスタに r0 を使うことで、他の命令で実現できます。ところが、16 ビット ISA では r0 を使えないため、個別の命令として追加されています。一致比較命令 (CMP、CMPI) と大小比較命令 (SLTI、SLTIU、SLT、SLTU) ではデスティネーションレジスタとして t8 (r24) が暗黙的に使われます。

16 ビット ISA には、32 ビット ISA と同じ種類のシフト命令がありますが、16 ビット ISA では *sa* フィールドは 3 ビットで、シフト量は 1 ~ 8 しか指定できません。(000 は 8 ビットのシフトとして定義されています)。ただし、EXTEND 命令により拡張された場合は、*sa* フィールドは、32 ビット ISA と同様の 5 ビットになります。

16 ビット ISA には、32 ビット ISA と同様の乗除算命令がありますが、16 ビット ISA では、MULT、MULTU 命令は積の下位 32 ビットを汎用レジスタにロードする機能がありません。また、16 ビット ISA には、積和演算命令、および MTHI・MTLO (Move To HI/LO) 命令がありません。

4.3.2　32 ビットの定数

EXTEND 命令を使っても、演算命令の即値フィールドは 16 ビットまでしか拡張できません。32 ビット ISA では、LUI 命令と ORI 命令の組み合わせで 32 ビットの定数を作ることができましたが、16 ビット ISA にはこれらの命令がありません。16 ビット ISA モードでは、32 ビットの定数は、コード中、通常はサブルーチンとサブルーチンの間に埋め込み、LW 命令で PC 相対アドレッシングモードを使って参照します。定数を格納しておくエリアを考慮しても、32 ビット ISA の LUI 命令と ORI 命令で必要とされるコードサイズ (32 ビット × 2) よりもコードサイズが小さくなります。

汎用レジスタの内容に 32 ビットの定数を加算する例を以下に示します。この例では、LW 命令で、32 ビットの定数をメモリから r5 にロードしています。そして、ADDU 命令で r4 と r5 の内容を加算して、その結果を r6 に格納しています。

```
LW      r5,offset(pc)
ADDU    r6,r4,r5
```

■ **ゼロ値**

通常、16 ビット ISA の命令は、r0 に直接アクセスすることはできません。そこで、ゼロ値が必要なときは、LI (Load Immediate) 命令を以下のように使います。この命令は即値 (0) をゼロ拡張し、*rx* に格納します。

```
LI rx,0
```

また、LI 命令の代わりに、MOVE 命令を使ってゼロ値を得ることができます。MOVE 命令は、16 ビット ISA で通常アクセスできる 8 本のレジスタと 32 本のレジスタの間でデータを移送できるので、以下の命令でゼロ値を得ることができます。

```
MOVE ry,r0
```

4.4　ジャンプ・分岐命令

この項では、16 ビット ISA で使用できるジャンプ・分岐命令について、32 ビット命令との違いを中心に説明します。4.4.1項では、ジャンプ・分岐命令の概要を説明します。4.4.2項では、大小関係に基づき分岐する方法について説明します。4.4.3項では、32 ビットの絶対アドレスへジャンプする方法について説明します。

4.4.1　ジャンプ・分岐命令の概要

16 ビット ISA には、BEQ、BNE、BGEZ、BGTZ、BLEZ、BLTZ のような 2 値を比較し、分岐する命令がありません。これらの命令がない代わりに、16 ビット ISA には、2 つのレジスタ値、またはレジスタ値と即値が等しいかどうかを調べる一致比較命令 (CMP、CMPI) があります。これらの比較命令は 2 値の排他的 OR をとって、その結果をレジスタ t8 に格納します。つまり、2 値が等しい場合、t8 は 0 に設定されます。また、同様に大小比較命令 (SLT・SLTI・SLTIU・SLTU) でも、比較結果が t8 に設定されます。そこで、16 ビット ISA には、t8 の内容を調べ t8 が 0 かどうかによって分岐する分岐命令が用意されています。16 ビット ISA には、リンク機能付きの分岐命令はありません。

16 ビット ISA でも、ジャンプ先のアドレスの範囲を広くとれるように、JAL・JALX 命令だけは例外として 32 ビットの命令になっています。

表 4-4と表 4-5に 16 ビット ISA のジャンプ、分岐命令を示します。

表 4-4　ジャンプ命令 (16 ビット ISA)

オペコード	命令	アドレッシング
JAL	Jump And Link	ページ内絶対
JALX	Jump And Link Exchange	ページ内絶対
JR	Jump Register	レジスタ間接
JALR	Jump And Link Register	レジスタ間接

表 4-5　分岐・分岐ライクリ命令 (16 ビット ISA)

オペコード	命令	条件	アドレッシング
BEQZ	Branch On Equal to Zero	$rx = 0$	PC 相対
BNEZ	Branch On Not Equal Zero	$rx \neq 0$	PC 相対
BTEQZ	Branch On T8 Equal To Zero	$t8 > 0$	PC 相対
BTNEZ	Branch On T8 Not Equal To Zero	$t8 \neq 0$	PC 相対
B	Branch Unconditional	—	PC 相対

リンク機能付きジャンプ命令では、レジスタ r31 に戻りアドレスが格納されます。これらの命令は、サブルーチンコールで使われます。

16 ビット ISA の分岐命令は、32 ビット ISA と同じアドレッシングモードを使います。ただし、16 ビット命令はハーフワード境界にそろえられるため、アドレスのオフセット値は 2 ビットではなく、1 ビットのみのシフトになります。また、オフセット値は 8 ビットになります。

B 命令は 32 ビットに伸長後、r0 と r0 を比較する BEQ 命令として実現されているため、分岐命令に分類されていますが、無条件分岐命令です。

■ **分岐遅延スロット**

16 ビット ISA モードには、分岐遅延スロットはありません。分岐命令は常に後続の命令の前に実行されます。条件が成立して分岐した場合は、直後に置かれた命令は廃棄され、実行されません。そのため、分岐命令の直後に置く命令に制限がありません。

ジャンプ命令は、32 ビット ISA 同様、16 ビット ISA でも遅延スロットがあります。

■ **ISA モードの実行中の切り換え**

表 4-1に示したように、16 ビット ISA には、32 ビット ISA と同様に JALX、JR、JALR 命令があります。これらの命令によりプログラムカウンタ (PC) の ISA モードビットを操作し、ISA モードを切り換えることができます。詳しくは、「3.4.3 ISA モードの切り換え」を参照してください。

■ **サブルーチンコール**

16 ビット ISA モードには、リンク機能付きジャンプ命令 (JALX・JALR) はありますが、リンク機能付き分岐命令、リンク機能付き分岐ライクリ命令はありません。サブルーチンコールついては、「3.4.7 サブルーチンコール」を参照してください。

4.4.2 大小関係に基づく分岐

前項で説明したように、16 ビット ISA には「BEQ r10, r7, Equal」のように 2 つのレジスタの値を比較して、分岐する命令がありません。また、16 ビット ISA の大小比較命令 (SLT・SLTU) では、オペランドとしてレジスタを 2 つしか指定できません。16 ビット ISA の、SLT、SLTU 命令は、2 つのレジスタの値が等しいがどうかによって、暗黙的にレジスタ t8 を設定します。そのため 16 ビット ISA には、t8 レジスタが 0 かどうかを調べる命令 (BTEQZ・BTNEZ) があります。

「3.4.5 大小関係に基づく分岐」で説明したように、32 ビット ISA モードでは、ORI 命令と BEQ 命令 (または BNE 命令) を組み合わせて使うことにより、レジスタの内容と即値を比較しました。

```
ORI     r10,r0,0x1234
BEQ     r10,r7,Label
```

ところが、16 ビット ISA には ORI のような即値をとる論理演算命令はなく、また r0 を使うこともできません。その代わりに 16 ビット ISA には、CMPI 命令が用意されています。この命令は、レジスタの内容と即値を比較し、等しいかどうかに基づき t8 を設定します。

16 ビット ISA モードでの比較・分岐の 3 つの例を以下に示します。

◆ 例 1　r6 ≥ r7 の場合の分岐
以下に、r6 の値が r7 の値以上の場合に、分岐する例を示します。r6 が r7 より小さいと、SLT (Set On Less Than) 命令により t8 が 1 に設定されます。r6 が r7 以上の場合は、r8 は 0 に設定されます。t8 の値が 0 の場合、BTEQZ 命令で Label に分岐します。(r0 はハードウエア的に 0 に固定されています。)

```
SLT     r6,r7
BTEQZ   Label
```

◆ 例 2　r7 ≥ 0x1234 の場合の分岐
以下に、r7 の値が 0x1234 以上の場合に分岐する例を示します。例 1 と同様、SLTI (Set On Less Than Immediate) 命令により、r7 の内容と 0x1234 の大小関係に基づき、t8 が暗黙的に設定されます。t8 が 0 の場合、BTEQZ 命令は Label に分岐します。

```
SLTI    r7,0x1234
BTEQZ   Label
```

◆ 例 3　r7 = 0x1234 の場合の分岐
以下に、レジスタの値と即値が等しいかどうか調べて、それに基づき分岐する例を示します。この例では、CMPI (Compare Immediate) 命令は、r7 の値と 0x1234 を比較し、等しければ t8 を 0 に設定します。(CMPI は、実際には 2 値の排他的 OR をとっています。)

```
CMPI    r7,0x1234
BTEQZ   Label
```

4.4.3　32 ビットのアドレスへのジャンプ

16 ビット ISA では LUI (Load Upper Immediate) 命令および ORI (OR Immediate) 命令がないため、LUI 命令と ORI 命令の組み合わせで、32 ビットのアドレスを生成できません。、16 ビット ISA では、32 ビットの定数をコード中に埋め込むことができます。PC 相対アドレッシングモードにより、LW 命令でメモリから 32 ビットの定数をロードできます。以下に例を示します。

```
LW      r4,0(pc)
JR      r4
```

また、PC 相対アドレスを計算し、その結果をレジスタに格納するための命令 (ADDIU, *rx*, pc, *immediate*) も用意されています。

4.5　特殊命令

特殊命令には、BREAK (Breakpoint) 命令と SDBBP (Software Debug Breakpoint) 命令があります。16 ビット ISA には、SYSCALL (System Call) 命令がありません。

また、16 ビット ISA には、EXTEND という命令があります。EXTEND 命令は、それだけでは機械語の命令を生成しません。EXTEND は、5 ビットのオペコードと 11 ビットの即値だけで構成されており、自身の即値を後続の命令の即値に連結することにより、表 4-6に示すように、即値フィールドを 32 ビット ISA で扱える大きさにまで拡張します。

表 4-6　拡張命令

16 ビット命令		即値ビットサイズ	
		拡張前	拡張後
ロード・ストア	LB・LBU	5	16
	LH・LHU	5	16
	LW	5 (または 8)	16
	SB	5	16
	SH	5	16
	SW	5 (または 8)	16
演算	ADDIU	4	15
		8	16
	SLTI・SLTIU	8	16
	CMPI	8	16
	LI	8	16
	SLL	3	5
	SRL	3	5
	SRA	3	5
分岐	BEQZ	8	16
	BNEZ	8	16
	BTEQZ	8	16
	BTNEZ	8	16
	B	11	16

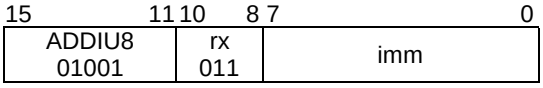
EXTEND 命令は、ワード境界から開始する必要はありません。EXTEND 命令には使用上 1 つだけ制約があり、ジャンプ遅延スロットに置いてはいけません。ジャンプ遅延スロットに置いた場合の動作は確定しません。

EXTEND 命令は、16 ビット命令の前に明示的に記述する必要はありません。即値フィールドをもつ 16 ビット ISA の命令で、即値フィールドで扱える範囲を超える値を指定すると、アセンブラにより自動的に EXTEND 命令を使って、即値が分解されます。例えば、以下のように記述したとします。

```
ADDIU r3,0x1234
```

この命令は RI 形式で、扱える即値は、本来表 4-6に示すように 8 ビットです。したがって、上記の ADDIU 命令は EXTEND 命令により拡張され、32 ビット EXT-RI 形式になります。

RI 形式 (即値)



EXT-RI 形式

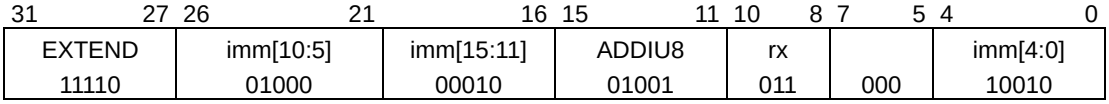


図 4-6　RI 形式と EXT-RI 形式

また、「ADDIU, *ry*, *rx*, *immediate*」という命令では即値フィールドは 4 ビットしかありません。EXTEND 命令の即値フィールドは 11 ビットしかないので、この命令だけは拡張しても、扱える即値は 32 ビット ISA と同様の 16 ビットにはならず、15 ビットまでです。

4.6　命令の概要

この項では、16 ビット ISA モードでの命令の概要を分類ごとに示します。

■　命令表記規則

この項では、命令中 *rx*、*ry*、*rz*、*immediate*、*sa* (シフト量: shift amount) などの小文字の斜体で示してある部分には、ユーザーが任意のレジスタや値などを指定できます。オペランドの意味がより明確になるように、例えば、ロード命令とストア命令では、*rx*、*immediate* と書かずに *base*、*offset* と記述してあります。命令によっては、特定の目的のために r24 (t8)、r29 (sp)、r31 (ra) を使用します。これらのレジスタは、t8、sp、ra と示します。HI と LO は、整数の乗算・除算の結果を格納する特殊レジスタです。

■　TX19 で実現されていない命令

TX19 は、MIPS16 のダブルワード命令をサポートしていません。TX19 と MIPS16 の比較は付録 D に示します。

表 4-7　ロード・ストア命令 (16 ビット ISA)

命令	形式		説明
Load Byte	LB	*ry*, *offset*(*base*)	5 ビット *offset* をゼロ拡張し、*base* に加算した結果が実効アドレス (EA) になります。EA でアドレス指定されたバイトデータを符号拡張し、*ry* にロードします。
Load Byte Unsigned	LBU	*ry*, *offset*(*base*)	5 ビット *offset* をゼロ拡張し、*base* に加算した結果が実効アドレス (EA) になります。EA でアドレス指定されたバイトデータをゼロ拡張し、*ry* にロードします。
Load Halfword	LH	*ry*, *offset*(*base*)	5 ビット *offset* を 1 ビット左へシフトして、ゼロ拡張し、*base* に加算した結果が実効アドレス (EA) になります。EA でアドレス指定されたハーフワードデータを符号拡張し、*ry* にロードします。
Load Halfword Unsigned	LHU	*ry*, *offset*(*base*)	5 ビット *offset* を 1 ビット左へシフトして、ゼロ拡張し、*base* に加算した結果が実効アドレス (EA) になります。EA でアドレス指定されたハーフワードデータをゼロ拡張し、*ry* にロードします。
Load Word	LW	*ry*, *offset*(*base*)	5 ビット *offset* を 2 ビット左へシフトして、ゼロ拡張し、*base* に加算した結果が実効アドレス (EA) になります。EA でアドレス指定されたワードデータを *ry* にロードします。
	LW	*ry*, *offset*(pc)	8 ビット *offset* を 2 ビット左へシフトして、ゼロ拡張し、マスクベース PC 値 (下位 2 ビットを 0 にマスクした PC 値) に加算した結果が実効アドレス (EA) になります。EA でアドレス指定されたワードデータを *ry* にロードします。
	LW	*ry*, *offset*(sp)	8 ビット *offset* を 2 ビット左へシフトして、ゼロ拡張し、sp に加算した結果が実効アドレス (EA) になります。EA でアドレス指定されたワードデータを *ry* にロードします。

命令	形式	説明
Store Byte	SB　*ry*, *offset*(*base*)	5 ビット *offset* をゼロ拡張し、*base* に加算した結果が実効アドレス (EA) になります。*ry* の最下位バイトをこのアドレスにストアしま
Store Halfword	SH　*ry*, *offset*(*base*)	5 ビット *offset* を 1 ビット左へシフトして、ゼロ拡張し、*base* に加算した結果がアドレス (EA) になります。*ry* の下位のハーフワードをこのアドレスに格納します。
Store Word	SW　*ry*, *offset*(*base*)	5 ビット *offset* を 2 ビット左へシフトして、ゼロ拡張し、*base* に加算した結果が実効アドレス (EA) になります。*ry* の内容をこのアドレスに格納します。
	SW　*rx*, *offset*(sp)	8 ビット *offset* を 2 ビット左へシフトして、ゼロ拡張し、sp に加算した結果が実効アドレス (EA) になります。*ry* の内容をこのアドレスに格納します。
	SW　ra, *offset*(sp)	8 ビット *offset* を 2 ビット左へシフトして、ゼロ拡張し、sp に加算した結果が実効アドレス (EA) になります。ra の内容をこのアドレスに格納します。

表 4-8　ALU 即値命令 (16 ビット ISA)

命令	形式	説明
Add Immediate	ADDIU　*ry*, *rx*, *immediate*	4 ビット *immediate* を符号拡張して、*rx* に加算し、その結果を *ry* に格納します。2 の補数のオーバフローでも例外を発生しません。
	ADDIU　*rx*, *immediate*	8 ビット *immediate* を符号拡張して、*rx* に加算し、その結果を *rx* に格納します。2 の補数のオーバフローでも例外を発生しません。
	ADDIU　sp, *immediate*	8 ビット *immediate* を左へ 3 ビットシフトし、符号拡張します。その結果を sp に加算した和を sp に格納します。2 の補数のオーバフローでも例外を発生しません。
	ADDIU　*rx*, pc, *immediate*	8 ビット *immediate* を 2 ビット左へシフトし、ゼロ拡張します。その結果をマスクベース PC 値 (下位 2 ビットを 0 にマスクした PC 値) に加算した和を *rx* に格納します。2 の補数のオーバフローでも例外を発生しません。
	ADDIU　*rx*, sp, *immediate*	8 ビット *immediate* を左へ 2 ビットシフトし、ゼロ拡張します。その結果を sp に加算した和を *rx* に格納します。2 の補数のオーバフローでも例外を発生しません。
Set On Less Than Immediate	SLTI　*rx*, *immediate*	*rx* が *immediate* より小さい場合は、t8 に 1 を格納し、そうでない場合は 0 を格納します。8 ビット *immediate* をゼロ拡張した値と *rx* を符号付き整数として比較します。
Set On Less Than Immediate Unsigned	SLTIU　*rx*, *immediate*	*rx* が *immediate* より小さい場合は、t8 に 1 を設定し、そうでない場合は 0 を格納します。8 ビット *immediate* をゼロ拡張した値と *rx* を符号なし整数として比較します。
Compare Immediate	CMPI　*rx*, *immediate*	*rx* = *immediate* ならば、t8 に 0 を、rx ≠ *immediate* ならば、t8 に 0 以外の値を格納します。8 ビット *immediate* を符号拡張します。
Load Immediate	LI　*rx*, *immediate*	8 ビット *immediate* をゼロ拡張し、*rx* に格納します。

表 4-9　レジスタタイプ命令 (16 ビット ISA)

命令	形式	説明
Add Unsigned	ADDU *rz*, *rx*, *ry*	*rx* + *ry* の和を *rz* に格納します。2 の補数のオーバフローでも例外を発生しません。
Subtract Unsigned	SUBU *rz*, *rx*, *ry*	*rx* – *ry* の差を *rz* に格納します。2 の補数のオーバフローでも例外を発生しません。
Set On Less Than	SLT *rx*, *ry*	*rx* が *ry* より小さい場合、t8 に 1 を、そうでない場合は、t8 に 0 を格納します。2 値を符号付き整数として比較します。
Set On Less Than Unsigned	SLTU *rx*, *ry*	*rx* が *ry* より小さい場合、t8 に 1 を、そうでない場合は、t8 に 0 を格納します。2 値を符号なし整数として比較します。
Compare	CMP *rx*, *ry*	*rx* が *ry* と等しい場合、t8 に 0 を格納します。そうでない場合、t8 に 0 以外の値を格納します。
Negate	NEG *rx*, *ry*	*rx* = 0 – *ry* (2 の補数)
AND	AND *rx*, *ry*	*rx* の内容と *ry* の内容の AND をとり、結果を *rx* に格納します。
OR	OR *rx*, *ry*	*rx* の内容と *ry* の内容の OR をとり、結果を *rx* に格納します。
Exclusive-R	XOR *rx*, *ry*	*rx* の内容と *ry* の内容の排他的 OR をとり、結果を *rx* に格納します。
Not	NOT *rx*, *ry*	*ry* をビットごとに反転させ、結果を *rx* に格納します。(1 の補数)
Move	MOVE *ry, r32*	*r32* の内容を *ry* に格納します。
	MOVE *r32, rz*	*rz* の内容を *r32* に格納します。

表 4-10　シフト命令 (16 ビット ISA)

命令	形式	説明
Shift Left Logical	SLL *rx*, *ry*, *sa*	*ry* の内容を *sa* ビット左へシフトし、右端の空いたビットをゼロで埋め、結果を *rx* に格納します。
Shift Left Logical Variable	SLLV *ry*, *rx*	*ry* の内容を *rx* の下位 5 ビットで指定されたビット数、左へシフトし、右端の空いたビットをゼロで埋めます。結果を *ry* に格納します。
Shift Right Logical	SRL *rx*, *ry*	*ry* の内容を *sa* ビット右へシフトし、左端の空いたビットを 0 で埋め、結果を *rx* に格納します。
Shift Right Logical Variable	SRLV *ry*, *rx*	*ry* の内容を *rx* の下位 5 ビットで指定されたビット数、右へシフトし、左端の空いたビットを 0 で埋めます。結果を *ry* に格納します。
Shift Right Arithmetic	SRA *rx*, *ry*, *sa*	*ry* の内容を *sa* ビット右へシフトし、左端の空いたビットを符号ビットで埋めます。結果を *rx* に格納します。
Shift Right Arithmetic Variable	SRAV *ry*, *rx*	*ry* の内容を *rx* の下位 5 ビットで指定されたビット数、右へシフトし、右端の空いたビットを符号ビットで埋めます。結果を *ry* に格納します。

表 4-11　乗算・除算命令 (16 ビット ISA)

命令	形式		説明
Multiply	MULT	*rx*, *ry*	被乗数は *rx* の符号付きの値です。乗数は *ry* の符号付きの値です。64 ビットの積 *rx * ry* を HI レジスタと LO レジスタに格納します。
Multiply Unsigned	MULTU	*rx*, *ry*	被乗数は *rx* の符号なしの値です。乗数は *ry* の符号なしの値です。64 ビットの積 *rx * ry* を HI レジスタと LO レジスタに格納します。
Divide	DIV	*rx*, *ry*	被除数は *rx* の符号付きの値です。除数は *ry* の符号付きの値です。商を LO レジスタに、剰余を HI レジスタに格納します。
Divide Unsigned	DIVU	*rx*, *ry*	被除数は *rx* の符号なしの値です。除数は *ry* の符号なしの値です。商を LO レジスタに、剰余を HI レジスタに格納します。
Move From HI	MFHI	*rx*	HI レジスタの内容を *rx* にロードします。
Move From LO	MFLO	*rx*	LO レジスタの内容を *rx* にロードします。

表 4-12　ジャンプ命令 (16 ビット ISA)

命令	形式		説明
Jump And Link	JAL	*target*	ページ内絶対アドレッシングモードを使ってジャンプします。つまり、26 ビット *target* を 2 ビット左へシフトし、PC + 2 の上位 4 ビットと連結した結果がターゲットアドレスになります。遅延スロットの後続命令のアドレスを r31 に格納します。
Jump And Link eXchange	JALX	*target*	ページ内絶対アドレッシングモードを使ってジャンプします。つまり、26 ビット *target* を 2 ビット左へシフトし、PC + 2 の上位 4 ビットと連結した結果がターゲットアドレスになります。遅延スロットの後続命令のアドレスを r31 に格納します。PC 内の ISA モードビットはトグルします。
Jump Register	JR	*rx*	*rx* の上位 31 ビットで指定されたアドレスへジャンプします。*rx* の最下位のビットの値によって、ISA モードが切り換わります。
	JR	ra	ra の上位 31 ビットで指定されたアドレスへジャンプします。ra の最下位のビットの値によって、ISA モードが切り換わります。
Jump And Link Register	JALR	(ra,) *rx*	*rx* の上位 31 ビットで指定されたアドレスへジャンプします。*rx* の最下位のビットの値によって、ISA モードが切り換わります。また、遅延スロットの後続命令のアドレスを ra に格納します。

表 4-13　分岐命令 (16 ビット ISA)

命令	形式		説明
Branch On Equal To Zero	BEQZ	*rx*, *offset*	*rx* = 0 ならば、PC + 2 に対して相対的に指定された 8 ビット *offset* に分岐します。
Branch On Not Equal To Zero	BNEZ	*rx*, *offset*	*rx* ≠ 0 ならば、PC + 2 に対して相対的に指定された 8 ビット *offset* に分岐します。
Branch On T8 Equal To Zero	BTEQZ	*offset*	t8 = 0 ならば、PC + 2 に対して相対的に指定された 16 ビット *offset* に分岐します。
Branch On T8 Not Equal To Zero	BTNEZ	*offset*	t8 ≠ 0 ならば、PC + 2 に対して相対的に指定された 16 ビット *offset* に分岐します。
Branch Unconditional	B	*offset*	PC + 2 に対して相対的に指定された 16 ビット *offset* に、無条件に分岐します。

表 4-14　特殊命令 (16 ビット ISA)

Instruction	Format	Operation
Breakpoint	BREAK *code*	ブレークポイント例外が発生し、無条件で例外ハンドラの処理に移ります。
Software Debug Breakpoint Exception	SDBBP *code*	デバッグポイント例外が発生し、無条件で例外ハンドラの処理に移ります。
Extend	EXTEND *immediate*	*immediate* が後続命令の即値フィールドに連結されます。

第5章　CPU パイプライン

5.1　概要

CPU の実行ユニットはステージといういくつかの部分で構成されているものと見ることができます。「2.5 パイプライン」で説明したように、各命令の処理は単純な処理に細分化され、それぞれの処理は各ステージで実行されます。あるステージでの処理結果は次のステージに渡され、パイプラインの各ステージは左から右へと流れていきます。TX19 のパイプラインは、命令フェッチ (F)、デコード (D)、実行 (E)、メモリアクセス (M)、レジスタ書き込み (W) の 5 ステージで構成されます。例えばある命令が D ステージを終了し、E ステージに進むと、後続の命令が D ステージへと進んできます。各ステージは、約 1 クロックで実行されます。いったんパイプラインが満たされると、図 5-1に示すように 5 つの命令が同時に実行されます。

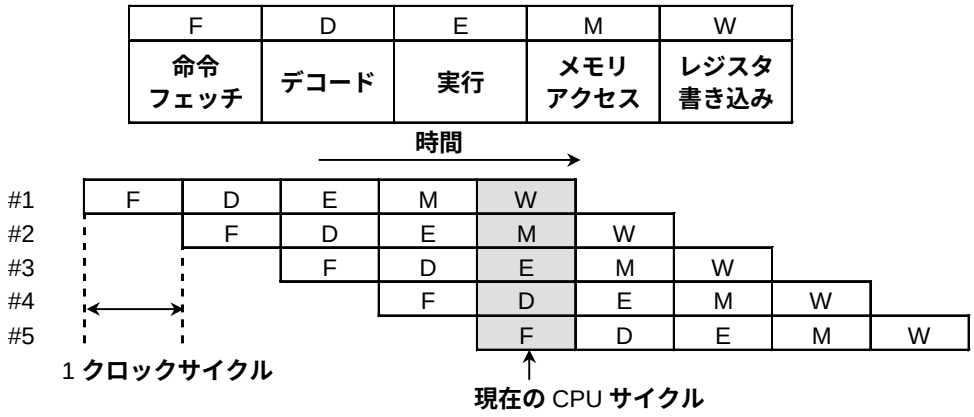


図 5-1　5 つの CPU パイプラインステージ

以下に代表的な命令が、各ステージでどのように処理されるかを示します。

命令フェッチ (F)　　命令メモリサブシステム(命令 ROM、命令 RAM)から命令がフェッチされます。命令は ISA モードに関らず、1 ワード単位でフェッチされます。

デコード (D)　　命令がデコードされ、必要なオペランドがレジスタファイルから読み出されます。

実行 (E)　　命令の種類により、以下の処理が ALU により実行されます。

- 整数の算術演算、論理演算、またはシフト処理を開始します。
- ロード、ストア命令の場合は、ベースレジスタの内容にオフセット値を加算して、実効アドレスを生成します。
- ジャンプ命令の場合は、ターゲットアドレスを計算します。
- 分岐、分岐ライクリ命令の場合は、分岐条件が成立するかどうかを判定するとともに、ターゲットアドレスを計算します。

メモリアクセス (M)　ロード、ストア命令の場合、データメモリがアクセスされます。

レジスタ書き込み (W)　命令の種類により、以下の処理が実行されます。
- 算術論理演算命令の場合は、E ステージでの ALU 操作の結果が、レジスタファイルに書き込まれます。
- リンク機能付きジャンプ、リンク機能付き分岐、リンク機能付き分岐ライクリ命令の場合は、戻りアドレスがレジスタ r31(ra)に書き込まれます。

TX19 のようなパイプラインを備えた CPU では、パイプラインのスムーズな流れを乱す命令があります。パイプラインの乱れをパイプラインハザードといいます。以下の項では、パイプラインハザードの原因、およびそれに対するハードウエア、ソフトウエア処理について説明します。

5.2　ロード・ストア・SYNC 命令

ソフトウエアの性能は、ソフトウエアの設計者、特にアセンブリ言語のプログラマが、プロセッサの基本的なハードウエアをどの程度理解しているかにより大きく左右されます。この項では、ロード遅延、ノンブロッキングロード、SYNC 命令などを CPU のパイプラインの観点から説明します。

5.2.1　ロード遅延

ロード命令は、図 5-2に示す順序で処理されます。

F	D	E	M	W
命令フェッチ	**デコード**	**実効アドレス生成**	**メモリアクセス**	**レジスタ書き込み**

図 5-2　ロード命令

ロード命令は、データ (オペランド) をメモリから CPU レジスタに読み込みます。高速内蔵メモリからのロードの場合は、ロード命令の M ステージが終了した時点で、後続の命令はロードされたデータを使用できるようになります。W ステージでレジスタに書き込まれるまで待つ必要がありません。それでも、図 5-3に示すように、ロードされたデータは、ロード命令の直後の命令では使えません。なぜならば、直後の命令の E ステージには間に合わないからです。この場合、ADD 命令は LW 命令に対してデータ依存関係があるといいます。図 5-3では、TX19 が後続命令の E ステージにウエイト (ストール) サイクルを挿入することにより、データの依存関係をなくしています。図 5-3では、1 サイクルの遅延 (レイテンシ) が発生します。ロード命令の直後の命令位置を「ロード遅延スロット」といいます。外部メモリからロードする必要がある場合は、さらに多くのストールサイクルが発生します。

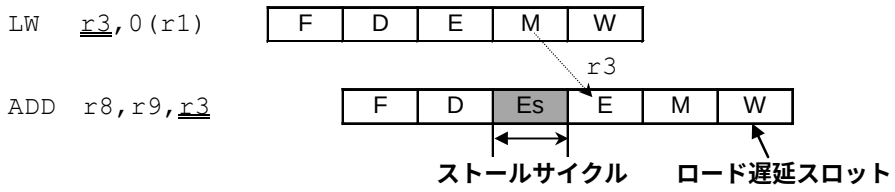


図 5-3　ロード命令のデータ依存

データの依存関係に対して、ハードウエアでストールサイクルを挿入するのは、効率的ではありません。コンパイラまたはアセンブラの最適化処理で、命令を並び替えることにより、ロード遅延スロットの命令が、直前のロード命令でロードされるデータに依存しないようにできます。図 5-4に例を示します。この例は 2 つのメモリ位置の内容を入れ替えるプログラムの一部です。

- データ依存関係があるプログラム

```
LW r9,0(r8)
LW r10,1(r8)
SW r10,0(r8) ← ロード遅延スロット
SW r9,1(r8)
```

- データ依存関係がないプログラム

```
LW r9,0(r8)
LW r10,1(r8)
SW r9,0(r8) ← ロード遅延スロット
SW r10,1(r8)
```

図 5-4　データ依存関係をなくすための命令の並び替え

命令を並び替えたあとのプログラムでは、最初の SW 命令は、直前の LW 命令のデータと依存関係がありません。このため、「LW r10,1(r8)」のロード遅延スロットの命令「SW r9,0(r8)」は、パイプラインをストールさせません。

5.2.2　ノンブロッキングロード

ロード命令の直後の命令が、ロード命令のターゲットレジスタ(*rt*)をアクセスしない場合、データの依存関係は発生しません。TX19 はデータの依存関係があるかどうか判断し、データの依存関係がない場合、パイプラインをストールさせることなく後続の命令を実行します。これを「ノンブロッキングロード」といいます。ノンブロッキングロードにより、外部メモリアクセス中も、パイプラインはストールしません。外部メモリアクセス中、パイプラインの他のステージでは、データ依存関係のない命令が継続して実行されます。

図 5-5の例では、LW 命令で外部メモリアクセスが発生してもストールせずに、データ依存関係のない命令「ADD r6,r4,r2」と「ADD r7,r5,r2」を継続して実行しています。ただし、LW 命令とデータ依存関係のある命令「ADD r8,r9,r3」は、最初の LW 命令のデータがロードされるまでパイプラインをストールします。

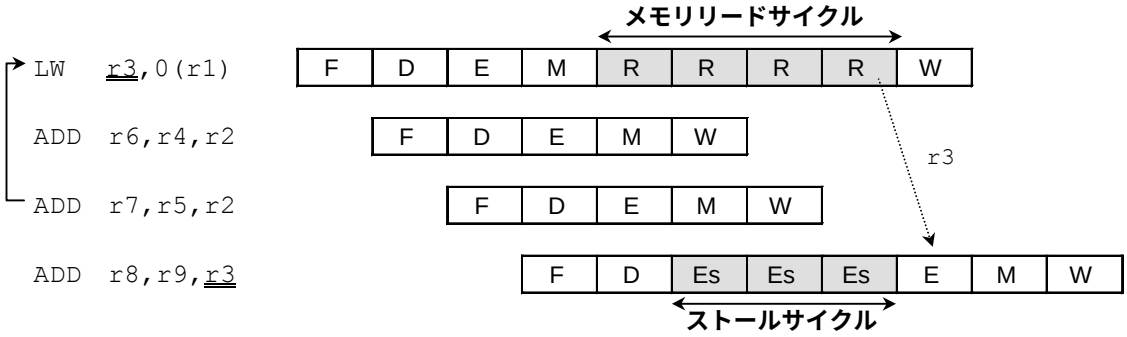


図 5-5　ノンブロッキングロード

ノンブロッキングロード機能を活用して、ロード命令を先行して実行し、ロードされたデータを使う命令のまえに、データ依存関係のない命令を実行させることにより、実行時間を短縮できます。これを「プリフェッチ動作」といい、コンパイラによる最適化でプリフェッチ動作ができるように、命令の並び替えが試みられます。

ノンブロッキングロードが行われると、ロード命令の W ステージと後続の命令の W ステージが、衝突することがあります。この場合、TX19 は後続命令の W ステージにストールサイクルを挿入します。

5.2.3　ストア命令

ストア命令は、図 5-6に示す順序で実行されます。

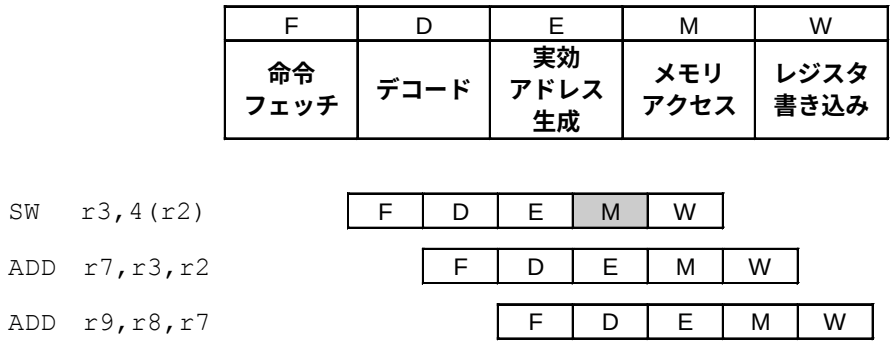


図 5-6　ストア命令

高速内蔵メモリへのストアは、M ステージで実行されます。W ステージでは何も実行されません。外部メモリへのストアは 1 サイクル以上かかります。

5.2.4　SYNC 命令 (32 ビット ISA)

ロード・ストア命令では、メモリにアクセスされるのは M ステージですが、それまでの間、TX19 は並行して他の命令を実行できます。

ロード・ストア命令のあとに SYNC 命令を置くと、SYNC 命令は、SYNC 命令の直前に実行したロード・ストア命令が完了するまで、M ステージでストールし、次の命令の実行を遅らせます。これにより、SYNC 命令のまえのロード・ストア命令と SYNC 命令の後の命令の実行順序を守ることができます。

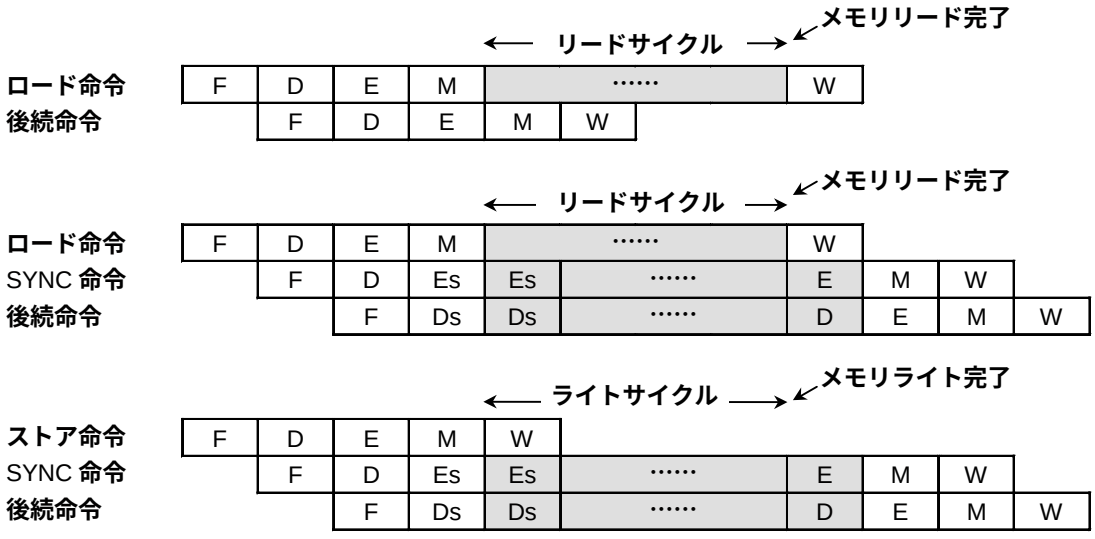


図 5-7　SYNC 命令 (32 ビット ISA)

5.3　ジャンプ・分岐・分岐ライクリ命令

ジャンプ、分岐命令では、通常 2 命令サイクルの遅延 (レイテンシ) が発生します。ただし、コンパイラ、アセンブラは遅延スロットを利用することによって、これを 1 サイクルにすることができます。この項では、ジャンプ、分岐遅延スロットについて、また、分岐ライクリ命令の処理について説明します。

5.3.1　ジャンプ・一般分岐命令 (32 ビット ISA)

ジャンプ命令、一般分岐命令は、図 5-8に示す順序で実行されます。

F	D	E	M	W
命令フェッチ	**デコード**	**ターゲットアドレスの計算** **分岐条件判定** PC **更新**	**なし**	**レジスタ** **書き込み**

図 5-8　**ジャンプ・分岐命令**

ジャンプ・分岐命令では、以下の処理が E ステージで実行されます。

- ジャンプ命令の場合は、ALU によりジャンプターゲットアドレスを計算します。
- 一般分岐命令、分岐ライクリ命令の場合は、ALU により分岐条件が成立するかどうか判定するとともに、分岐ターゲットアドレスを計算します。

M ステージでは何も実行されません。また、リンク機能付きジャンプ命令またはリンク機能付き分岐命令の場合は、W ステージで、戻りアドレスがレジスタ r31(ra) に書き込まれます。

ジャンプまたは分岐ターゲットアドレスは、E ステージで計算されます。そのため、パイプラインをストールさせないと、ジャンプまたは分岐先の命令をフェッチすることができません。図 5-9に、ジャンプまたは分岐命令で 2 命令サイクルの遅延が発生することを示します。図 5-9のように、遅延スロットを、有用な命令で埋めることができれば、パイプラインのストールは 1 サイクルになります。通常、32 ビット ISA のジャンプ命令、一般分岐命令の場合、遅延スロットの命令は (分岐の成立、不成立にかかわらず) 常にジャンプ、分岐先の命令より先に実行されます。遅延スロットはコンパイラの最適化処理により、有用な命令で埋められます。有用な命令がない場合は、コンパイラは遅延スロットに NOP を挿入します。

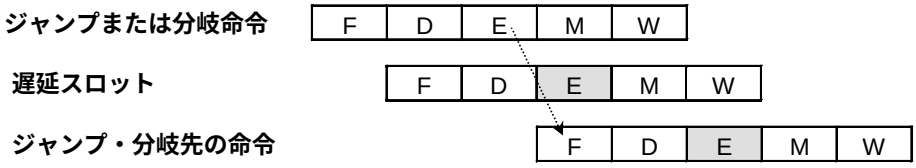


図 5-9　**ジャンプ・分岐遅延スロット**

ジャンプまたは分岐遅延スロットに、ジャンプ命令または分岐命令を置いてはいけません。これらの遅延スロットにジャンプ命令、分岐命令を置いた場合のハードウエアの動作は不定です。

5.3.2　分岐ライクリ命令 (32 ビット ISA)

一般分岐命令では、分岐が成立するかしないかにかかわらず、TX19 は常に遅延スロット内の命令を実行します。そのため、分岐遅延スロット内の命令は、プログラムの論理に影響を与えるものであってはいけません。

これに対して、分岐ライクリ命令では、分岐が成立しない場合は、遅延スロット内の命令を E ステージで無効にし、分岐が成立した場合のみ遅延スロット内の命令を実行します。このため、コンパイラは分岐遅延スロット内に分岐先の命令を置くことができます (図 5-10参照)。

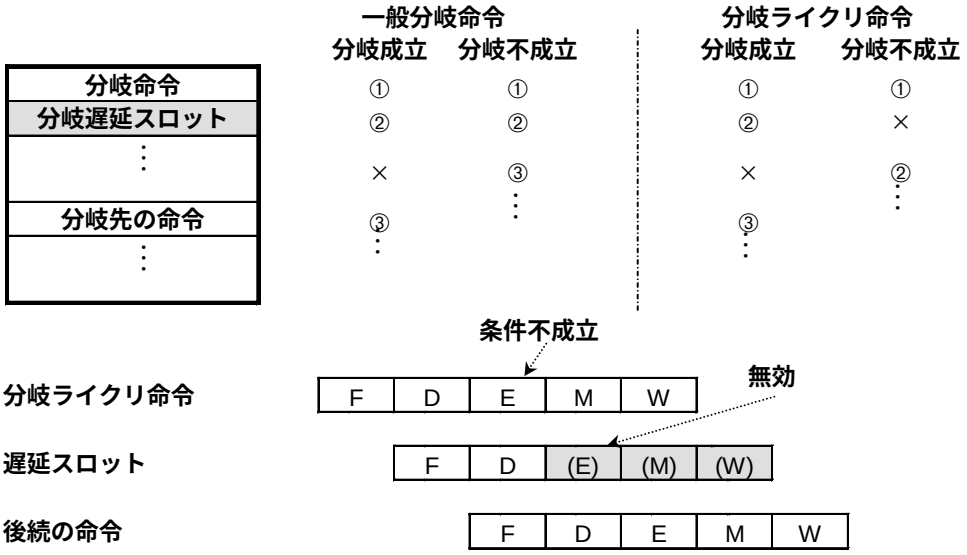


図 5-10　分岐ライクリ命令 (分岐不成立の場合)

5.3.3　ジャンプ命令 (16 ビット ISA)

16 ビット ISA の JAL・JALX 命令は、例外的に 32 ビット長です。そのため、16 ビット ISA では、以下に示すように、これらのジャンプ命令を 2 段階に分けて実行しなければなりません。TX19 は最初の D、E ステージでは何も実行せず、後半のコードがフェッチされるのを待ってから、ジャンプ先のターゲットアドレスを計算します。アドレスの計算は、ジャンプ命令の後半の E ステージで実行されます。その結果、16 ビット ISA の JAL・JALX 命令では 2 命令サイクルの遅延が発生します。

ジャンプ遅延スロット内に、ジャンプ命令、分岐命令、拡張命令を置くことは禁止されています。

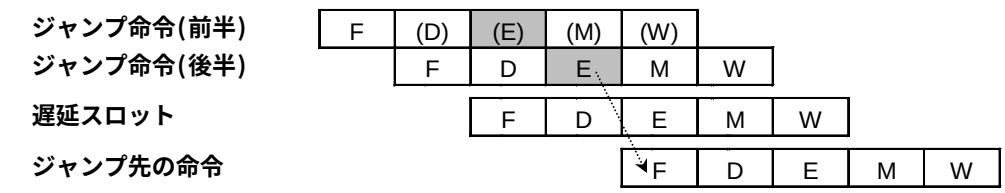


図 5-11　ジャンプ命令 (16 ビット ISA)

5.3.4　分岐命令 (16 ビット ISA)

32 ビット ISA とは異なり、16 ビット ISA の分岐命令には遅延スロットがありません (図 5-12参照)。分岐はその直後の命令の前で有効になります。したがって分岐が成立した場合は、分岐命令の直後に置かれた命令は実行されません。そのため、分岐命令の直後に置く命令に制限はありません。

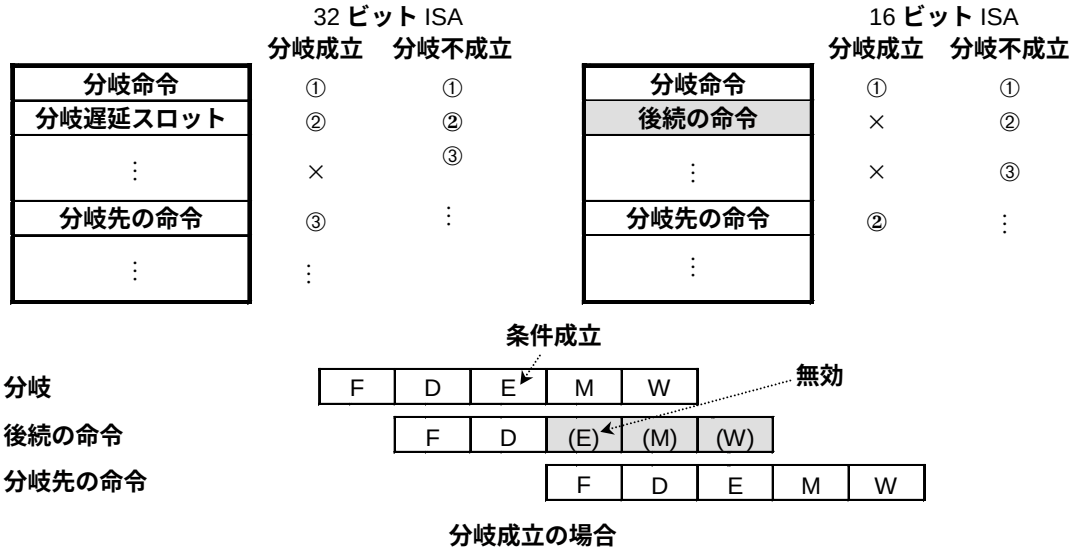


図 5-12　分岐命令 (16 ビット ISA)

5.4　除算命令

整数除算命令は、専用の除算ユニットで実行されるため、他の命令と並行して継続できます。除算ユニットは、遅延サイクル、例外が発生しても、命令の実行を継続します。除算命令の商と剰余は、LO レジスタと HI レジスタに格納されます。

除算は E ステージで実行を開始します。オペランドの値の大きさ、符号にかかわらず、命令の実行時間は 35 命令サイクルです。除算が完了するまえに、MFHI、MFLO、MADD、MADDU 命令を実行すると、LO、HI レジスタが確定するまでパイプラインがストールします。

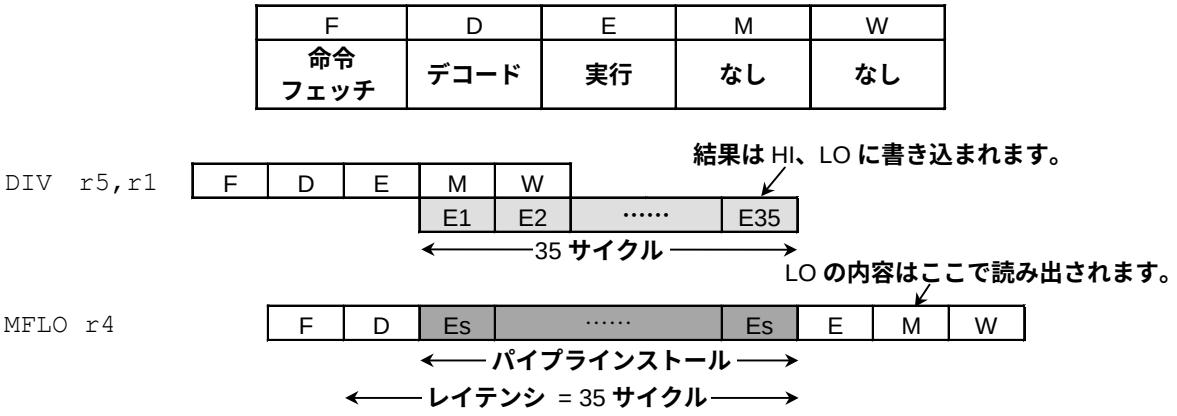


図 5-13　除算命令

5.5　乗算・積和命令

整数の乗算、積和命令は、専用 MAC ユニットで実行されるため、他の命令と並行して継続できます。乗算命令、積和命令は、1 クロックで完了します。

乗算命令、積和命令は E ステージを 1 クロックで完了するので、複数の乗算命令、積和命令を、パイプラインをストールすることなしに連続して実行できます。(図 5-14参照)

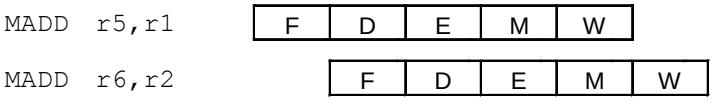


図 5-14　連続した積和命令

HI、LO レジスタの内容を読み出すには、MFHI、MFLO 命令を使います。図 5-15 のように乗算命令、積和命令の直後に MFHI または MFLO 命令を置いても、パイプラインはストールしません。

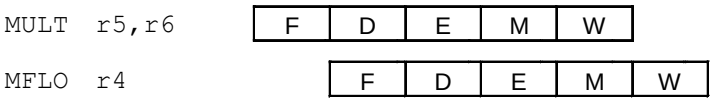


図 5-15　乗算命令と後続の MFLO 命令

乗算命令、積和命令の結果は、E ステージではなく M ステージ完了後に使用可能になります。乗算命令または積和命令で、汎用レジスタをデスティネーションレジスタ(*rd*)として指定する場合、後続の命令は、結果が *rd* に格納されるまで、*rd* にアクセスしてはいけません。後続の命令が *rd* を指定すると、パイプラインは *rd* に結果が格納されるまで、D ステージでストールします。

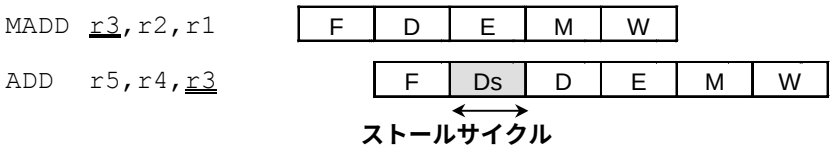


図 5-16　乗算命令での構造ハザード

5.6　拡張命令 (16 ビット ISA)

EXTEND が付加されると、16 ビット ISA の命令は 32 ビットになります。拡張された命令のコードは、16 ビットの EXTEND コードと拡張される 16 ビット命令コードで構成されます。図 5-17に示すように、TX19 は拡張された命令を 2 段階で実行します。

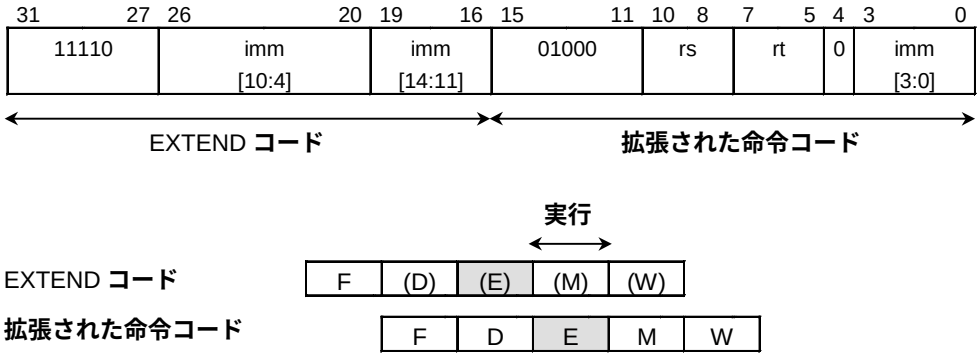


図 5-17　拡張命令 (16 ビット ISA)

第6章　メモリ管理

この章では、TX19 の動作モード、仮想アドレス空間、物理アドレス空間、アドレス変換について説明します。

6.1　動作モード

TX19 にはユーザーモードとカーネルモードの 2 種類の動作モードがあります。TX19 は例外が発生すると、自動的にカーネルモードに移行します。また、システムリセットがかかると、プロセッサはリセット例外により立ち上がるため、立ち上げ時はカーネルモードになります。RFE (Restore From Exception) 命令、または DERET (Debug Exception Return) 命令により、カーネルモードからユーザーモードへ復帰します。

■　ユーザーモード

動作モードにより、プログラムで使用できる仮想アドレス空間、レジスタ、命令が異なります。ユーザーモードではこれらの使用が制限されます。ユーザーモードで使用できる仮想アドレス空間は、0x0000_000 から 2G バイトのリニアアドレス空間 (kuseg) に制限されます。また、CP0 レジスタへのアクセスは、Status レジスタの CU[0]ビットが 1 にセットされているときに制限されます。

■　カーネルモード

カーネルモードはユーザーモードより高い特権レベルが与えられ、4G バイト仮想アドレス空間、すべてのレジスタ、命令を使用できます。オペレーティングシステムのルーチン、例外ハンドラ、デバックハンドラなどのプログラムは、カーネルモードで実行します。

6.2　仮想アドレス空間

図 6-1にユーザーモード、カーネルモードでアクセスできる仮想アドレス空間を示します。ユーザーモードでは、2G バイトの仮想アドレス空間 (kuseg) のみアクセスできます。これに対し、カーネルモードでは、4 つのセグメントの仮想アドレス空間 kuseg、kseg0、kseg1、kseg2 をすべてアクセスできます。

セグメントによりキャッシュ可/不可が決められていますが、TX19 ではキャッシュを実装しないので、この区別に意味はありません。

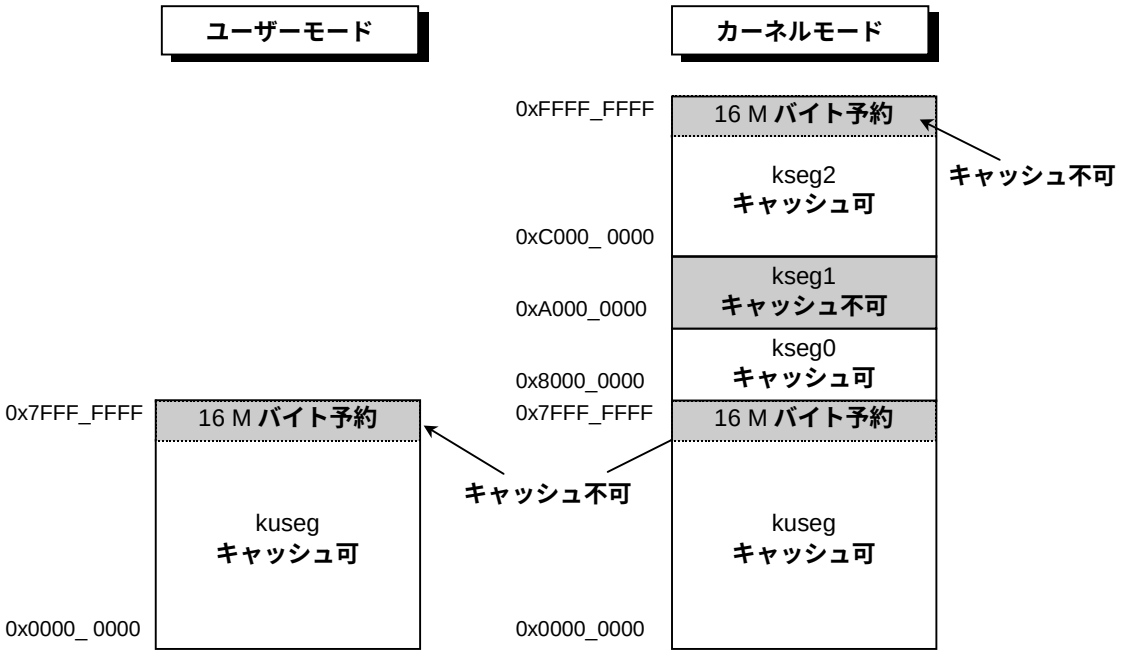


図 6-1　仮想アドレス空間

■ kuseg (**カーネル・ユーザーセグメント**)

カーネルモード、ユーザーモードでアクセスできる 2G バイトのセグメントです。アドレス 0x0000_0000 から 0x7FFF_FFFF までの仮想アドレス空間で、ユーザーモードの仮想アドレスの最上位ビットはかならず 0 になります。ユーザーモードのプログラムが、仮想アドレスの最上位ビットが 1 のアドレスをアクセスしようとすると、アドレスエラー例外が発生します。kuseg の上位 16 M バイトは、チップ上に内蔵された周辺回路用に予約されており、使用できません。

■ kseg0・kseg1・kseg2 (**カーネルセグメント**)

カーネルモードだけでアクセスできる仮想アドレス空間は、仮想アドレス 0x8000_0000 から 0xFFFF_FFFF までの 2G バイトで、kseg0、kseg1、kseg2 の 3 つのセグメントに分割されています。

- kseg0 は、仮想アドレス 0x8000_0000 から始まる 512 M バイトのセグメントで、キャッシュ可能領域です。
- kseg1 は、仮想アドレス 0xA000_0000 から始まる 512 M バイトのセグメントです。kseg0 と異なり、非キャッシュ領域です。
- kseg2 は、仮想アドレス 0xC000_0000 から始まる 1G バイトのリニアアドレス空間です。kseg2 の上位 16 M バイトはチップ上に内蔵された周辺回路用に予約されており、使用できません。0xFF20_0000 から 0xFF3F_FFFF の 2 M バイトのアドレス空間は、デバッグ用に予約されています。上位 16 M バイトは非キャッシュ領域で、それ以外の領域はキャッシュ領域です。

6.3　アドレス変換

仮想アドレスは、ダイレクトセグメントマッピング方式で物理アドレスに変換されます。この方式では、カーネルモードのソフトウエアは、仮想ページの管理を必要とせずに、ユーザーモードアクセスから保護されます。図 6-2に、仮想アドレスと物理アドレスのマッピングを示します。

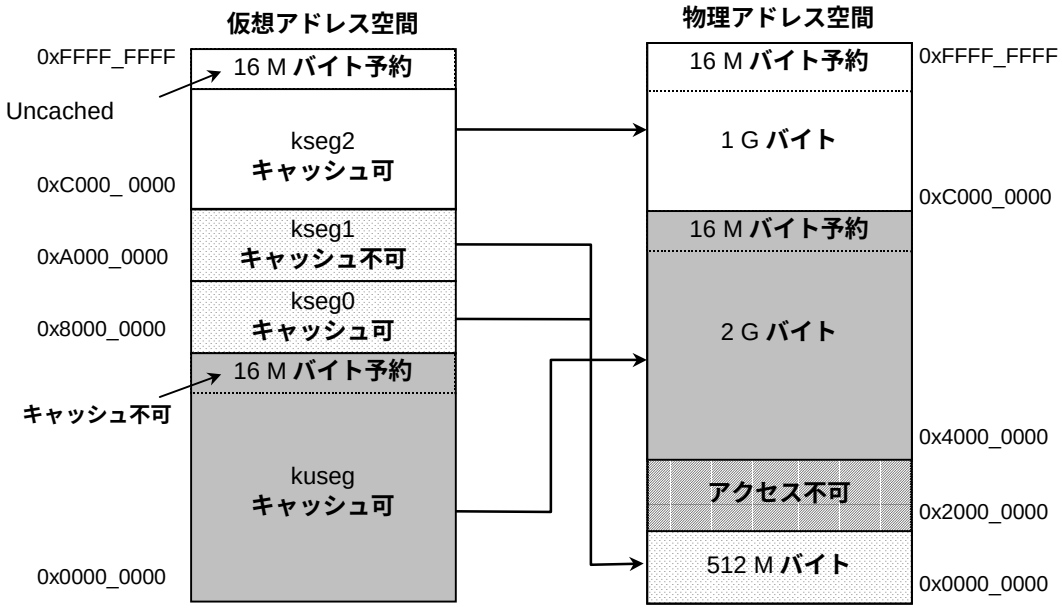


図 6-2　仮想アドレスから物理アドレスへの変換

図 6-3に、TX19 で使用される仮想アドレスの構成を示します。上位 3 ビットによりセグメント番号を表し、この 3 ビットだけで仮想アドレスから物理アドレスへ変換されます。

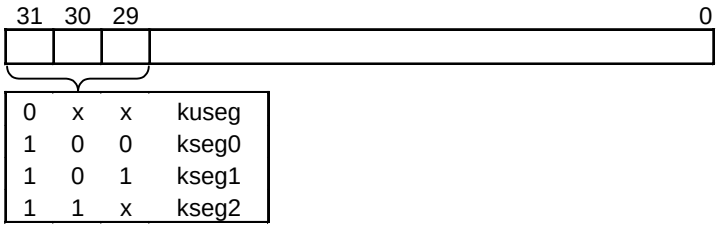


図 6-3　仮想アドレス構成

- kuseg は、アドレス 0x4000_0000 から始まる 2 G バイトの物理アドレス空間にマッピングされます。仮想アドレスの上位 2 ビット 0x を 01 に置き換えると物理アドレスになります。

- kseg0 と kseg1 の仮想アドレスは、アドレス 0x0000_0000 から始まる 512 Mバイトの物理アドレス空間にマッピングされます。仮想アドレスの上位 3 ビットが 100 のとき、仮想アドレスは kseg0 にあります。仮想アドレスの上位 3 ビットが 101 のとき、仮想アドレスは kseg1 にあります。仮想アドレスの上位 3 ビットを 000 に置き換えると物理アドレスになります。

- kseg2 の仮想アドレスは、物理アドレスとしてそのまま出力されます。

表 6-1　仮想アドレスと物理アドレスのマッピング

セグメント		仮想アドレス	物理アドレス	キャッシュ	動作モード
kseg2	予約	0xFF20_0000~0xFFFF_FFFF	0xFF00_0000~0xFFFF_FFFF	不可	カーネル
	自由	0xC000_0000~0xFEFF_FFFF	0xC000_0000~0xFEFF_FFFF	可	カーネル
kseg1		0xA000_0000~0xBFFF_FFFF	0x0000_0000~0x1FFF_FFFF	不可	カーネル
kseg0		0x8000_0000~0x9FFF_FFFF	0x0000_0000~0x1FFF_FFFF	可	カーネル
kuseg	予約	0x7F00_0000~0x7FFF_FFFF	0xBF00_0000~0xBFFF_FFFF	不可	カーネル・ユーザー
	自由	0x0000_0000~0x7EFF_FFFF	0x4000_0000~0xBEFF_FFFF	可	カーネル・ユーザー

セグメントをまたいで、プログラムを置くことは禁止されています。また、ジャンプ命令、分岐命令で、現在のセグメントの外に分岐することもできません。

第7章　内部 I/O バスオペレーション

7.1　内部メモリインタフェース

TX19 コアのバスインタフェースの一例を図 7-1に示します。TX19 コアのメモリインタフェースは、基本的には、命令バスとオペランドバスとが独立したハーヴァードアーキテクチャです。それぞれのバスは互いに独立しており、しかも 1 クロック当たり 1 ワードのデータへアクセスできます。これらのバス仕様により TX19 コアは、1 クロック当たり 1 命令の実行を保証しています。

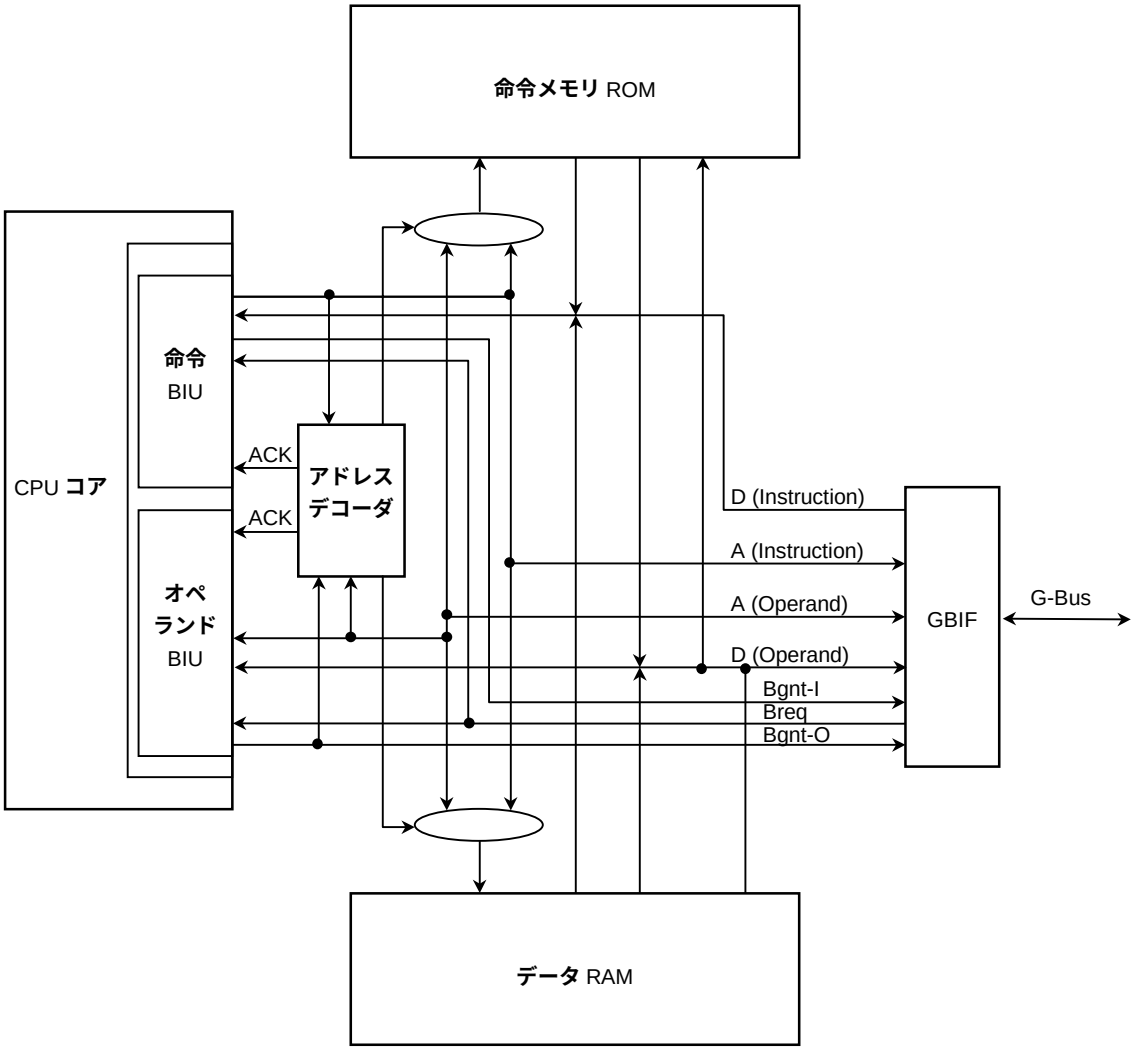


図 7-1　内部メモリインタフェースの概要

7.2　オペランドリード・命令フェッチオペレーション

TX19 コアのバスサイクルの特徴は、先行のバスサイクルからデータを読み込む動作が終了しないうちに後続する次のバスサイクルのアドレスを出力してしまう、パイプラインバスサイクルになっていることです。これにより、フラッシュメモリのような動作が比較的遅いメモリを用いた場合でも、ゼロウェイトの動作が可能です。

オペランドリードおよび命令フェッチの際のバスサイクルのタイミングチャートを図 7-2、図 7-3に示します。

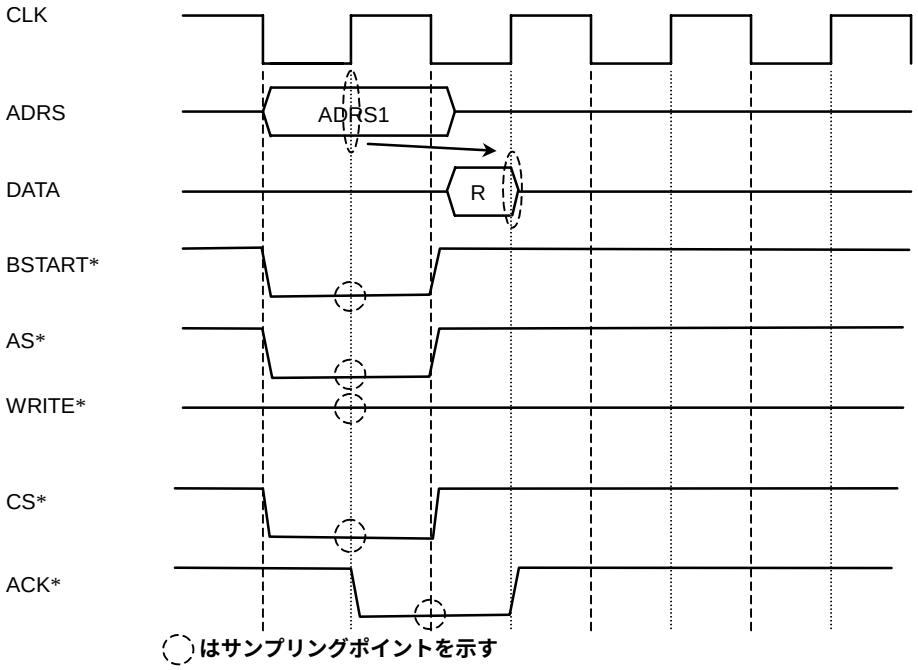


図 7-2　メモリリードのタイミング (ゼロウエイト)

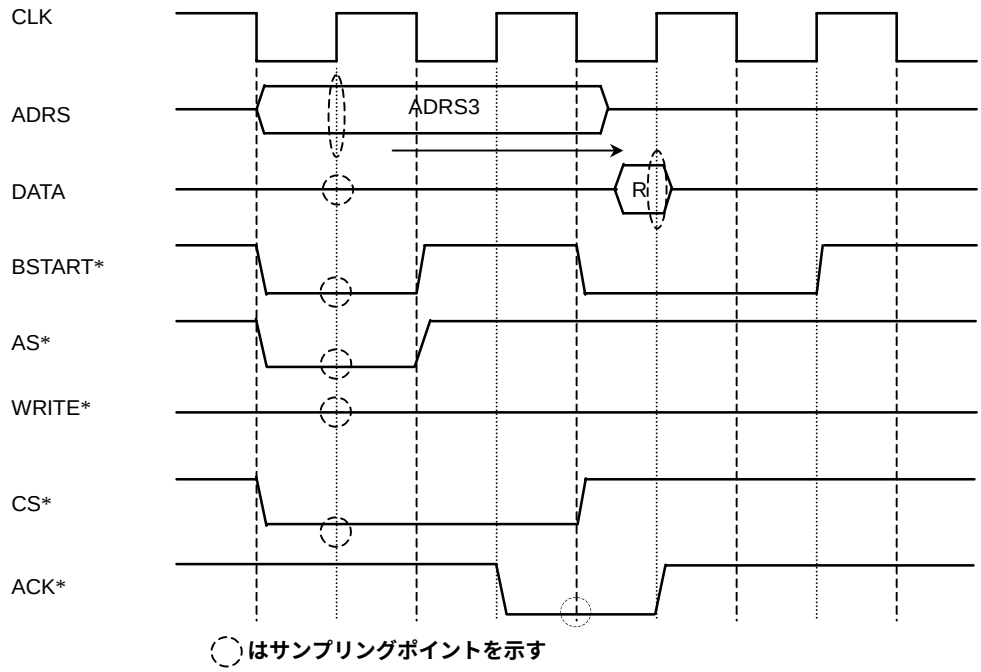


図 7-3　メモリリードタイミング (ADRS3 に対してウエイトを設定した場合)

7.3　ライトオペレーション

ライトオペレーションの動作は、基本的には、リードと同じです。

システムクロックの立ち下がりに同期して、アドレスは出力します。これと同時にアサートする信号は、バイトイネーブル、バススタート(BSTART*), アドレスストローブ(AS*), ライト(WRITE*) などです。

アドレスの出力開始から 1 クロック後の立ち下がりで ACK*をサンプリングし、ACK のアサートを確認します。この確認時点でアドレスを切り替えて次のバスサイクルを開始します。もし、ACK*のアサートが確認できない場合は同じバスサイクルを継続します。

メモリモジュールや I/O モジュールなどが行うリードデータのラッチは、ACK*のサンプリングから半クロック後の立ち上がりで、実施してください。

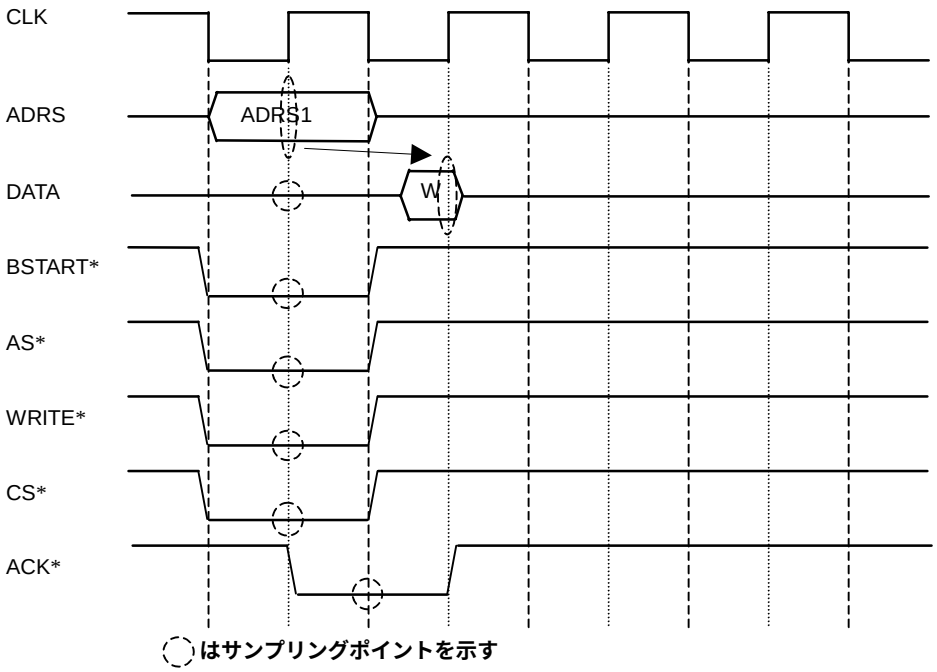


図 7-4　ライトタイミング (ゼロウエイト)

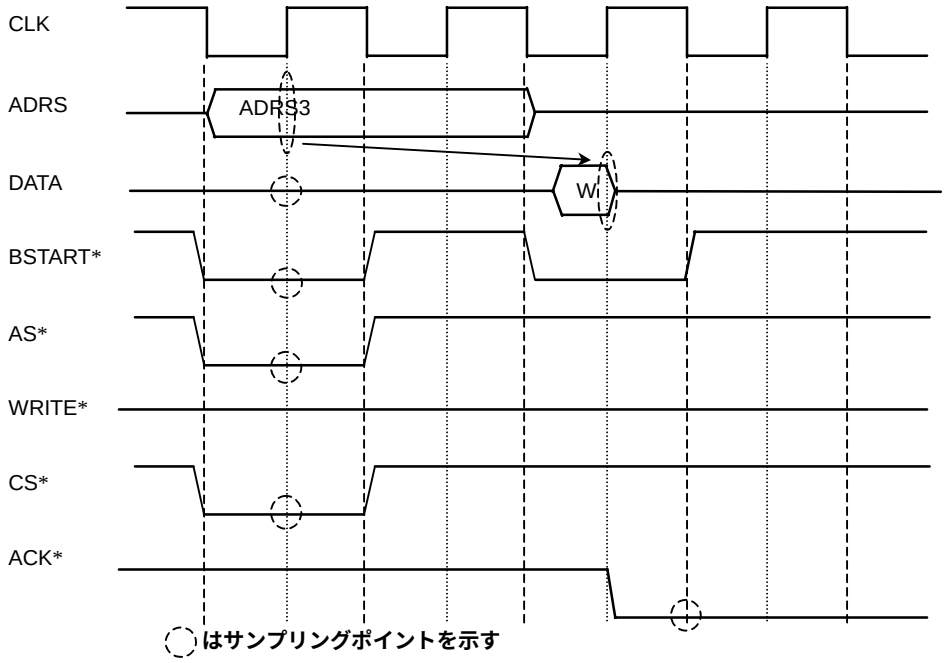


図 7-5　ライトオペレーションのタイミング (ADR3 **に対してウェイトを設定した場合**)

第8章　システム制御コプロセッサ (CP0) レジスタ

この章では、システム構成、メモリ管理、例外処理で使用するシステム制御コプロセッサ (CP0) について説明します。

プロセッサがカーネルモードのときは、システム制御コプロセッサ命令は、常に CP0 レジスタを使用できます。プロセッサがユーザーモードのときは、Status レジスタの CU(0) ビットが 1 に設定されているときのみ、CP0 レジスタを使用できます。

8.1　概要

表 8-1に CP0 レジスタの一覧を示します。レジスタ番号は、MFC0 (Move From CP0) 命令と MTC0 (Move To CP0) 命令で使われます。

表 8-1　CP0 レジスタ

分類	レジスタ名	レジスタ番号	説　明
システム構成	Config	3	TX19 プロセッサのさまざまなコンフィグレーションを設定します。
一般例外処理	BadVAddr	8	仮想アドレスから物理アドレスへの変換でエラーを起こした仮想アドレスを示します。読み出し専用です。
	Status	12	動作モード (ユーザー・カーネル)、割り込み許可状態などのプロセッサの状態を保持します。
	Cause	13	直前に発生した例外の原因を示します。
	EPC	14	例外の原因となった命令のアドレス、すなわち、例外処理後のプログラムの戻りアドレスと、例外発生時の ISA モードを保持します。
	PRId	15	TX19 プロセッサのリビジョンを示します。読み出し専用です。
	IE	31	Status レジスタの割り込み許可・禁止ビットを操作します。
デバッグ例外処理	Debug	16	デバッグ例外の原因と現在の状態を示します。
	DEPC	17	デバッグ例外の原因となった命令のアドレス、すなわち、デバッグ例外処理後のプログラムの戻りアドレスと、例外発生時の ISA モードを保持します。

以下の項では、CP0 レジスタの構成とレジスタの各ビットの意味について説明します。見出し中で、「8.2.1　Config レジスタ (3)」などレジスタ名の後の番号は、レジスタ番号を表します。

8.2　システム構成レジスタ

Config レジスタには、TX19 プロセッサのさまざまなコンフィグレーションを設定します。低消費電力モード (Halt・Doze)、分周モード、データキャッシュイネーブル、命令キャッシュイネーブルなどの設定ができます。TX19 ではキャッシュを実装しないので、これらの設定は意味をもちません。Config レジスタについて、以下の項で説明します。

8.2.1　Config レジスタ (3)

図 8-1に Config レジスタの構成を示します。また、表 8-1に Config レジスタの各ビットの機能を示します。

	31 … 16	15 … 12	11 … 10	9	8
ニモニック	0	0	RF	Doze	Halt
リード・ライト	R	R	RW	RW	RW
リセット後	0	0	00	0	0

Halt Halt モード	
0	Halt モード解除
1	Halt モード

Doze Doze モード	
0	Doze モード解除
1	Doze モード

RF[1:0] プロセッサクロックの分周	
00	分周なし
01	2 分周
10	4 分周
11	8 分周

	7 … 5	4	3 … 2	1	0
ニモニック	0	DCE	0	ICE	Lock
リード・ライト	R	RW	R	RW	RW
リセット後	0	0	0	0	0

DCE データキャッシュイネーブル	
0	データキャッシュ無効
1	データキャッシュ有効

ICE 命令キャッシュイネーブル	
0	命令キャッシュ無効
1	命令キャッシュ有効

Lock Config レジスタの書き込みロック	
0	書き込みロック解除
1	書き込みロック

図 8-1　Config レジスタ

表 8-2　Config レジスタ

ニモニック	名前	リセット後	説　明	リード・ライト
RF (Reduced Frequency)	プロセッサクロックの分周	00	TX19 プロセッサコアの外部のクロックジェネレータへ、クロックの分周比を指示するために用意されています。このフィールドに設定された値は、プロセッサコアの出力ピンに出力されます。RF モードについては、第 10 章を参照してください。	RW
Doze (Doze Mode)	Doze モード	0	TX19 の Doze モードを起動、解除します。 1　Doze モード起動 0　Doze モード解除 このビットを 1 に設定すると、CPU はパイプラインを停止します。Doze モードは、リセット信号 (リセット例外を起動する)、ノンマスカブル割り込み信号、または通常の割り込み信号により解除されます。(割り込みがマスクされている状態でも、割り込み信号を認識します。) Doze モードについては、第 10 章を参照してください。	RW
Halt (Halt Mode)	Halt モード	0	TX19 の Halt モードを起動、解除します。 1　Halt モード起動 0　Halt モード解除 このビットを 1 に設定すると、CPU はパイプラインを停止します。Halt モードは、リセット信号 (リセット例外を起動する)、ノンマスカブル割り込み信号、または通常割り込み信号により解除されます。(割り込みがマスクされている状態でも、割り込み信号を認識します。) Halt モードについては、第 10 章を参照してください。	RW
Lock (Config Register Locking)	Config レジスタロック	0	このビットが 1 のとき、Config レジスタがロックされ、書き込みができません。このビットは、リセット例外により 0 にクリアされます。デバッグ例外ハンドラ (Debug レジスタの DM ビットが 1 に設定されている場合) は、Lock ビットの設定に関係なく、MTC0 命令により、Config レジスタを変更することができます。	RW
ICE (Instruction Cache Enable)	命令キャッシュイネーブル	0	内蔵されている命令キャッシュを、イネーブル・ディセーブルします。 1　イネーブル 0　ディセーブル	RW
DCE (Data Cache Enable)	データキャッシュイネーブル	0	内蔵されているデータキャッシュを、イネーブル・ディセーブルします。 1　イネーブル 0　ディセーブル	RW
0 (Reserved)	予約	–	予約ビットは、書き込みを実行しても無視されます。読み出すと 0 が返されます。	R

†† Doze ビットと Halt ビットを同時に 1 に設定した場合の動作は不定です。

8.3　一般例外処理レジスタ

この項では、一般例外処理で使用される CP0 レジスタについて説明します。デバッグ例外処理で使用される CP0 レジスタは、「8.4　デバッグ例外処理レジスタ」で説明します。

8.3.1　BadVAddr レジスタ (8)

BadVaddr (Bad Virtual Address) レジスタは、読み出し専用のレジスタで、直前に仮想アドレスから物理アドレスへの変換でエラーを起こした仮想アドレスを表示します。アドレス変換エラーが起こると、アドレスエラー例外 (AdEL または AdES) が発生します。図 8-2に BadVAddr レジスタの構成を示します。

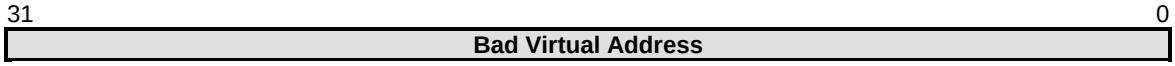


図 8-2　BadVAddr レジスタ

8.3.2　Status レジスタ (12)

Status レジスタの動作モードビットと割り込み許可ビットは、3 レベルのスタック (現、前、旧) を、割り込みマスクレベルフィールドは、2 レベルのスタック (現、前) を構成しています。スタックは、例外が発生するたびにプッシュされ、RFE (Restore From Exception) 命令を実行するとポップされます。詳細は「第 9 章　例外処理」で説明します。また、Status レジスタには、コプロセッサ使用可能ビット、ソフトウエア割り込みマスクビットなどのビットがあります。図 8-3に Status レジスタの構成を示します。また、表 8-3に Status レジスタの各ビットの機能を示します。

	31 28	27 26	25	24 23	22	21	20	19	
ニモニック	CU	0	RE	0	BEV	0	NmI	0	
リード・ライト	RW	R	RW	R	RW	R	RW	R	
リセット後	X	0	X	0	1	0	0	0	

注: X は不定を表します。

NmI ノンマスカブル割り込み	
0	割り込み解除
1	割り込み発生

BEV ブートストラップ例外ベクタ	
0	通常モード: 外部拡張メモリモード
1	代替モード: 内蔵 ROM モード

RE 予約	
0	常に 0 をライトしてください

CU[3:0] コプロセッサ使用可能	
0	使用不可
1	使用可

		18 16	15 13	12	11 8	7 6	5	4	3	2	1	0
ニモニック		PMask	CMask	0	SwiMask	0	KUo	IEo	KUp	IEp	KUc	IEc
リード・ライト		RW	RW	R	RW	R	RW	RW	RW	RW	RW	RW
リセット後		X	7	0	X	0	X	X	X	X	0	0

IEc 割り込み許可状態 (現)	
0	割り込み禁止
1	割り込み許可

KUc 動作モード (現)	
0	カーネルモード
1	ユーザーモード

IEp 割り込み許可状態 (前)	
0	割り込み禁止
1	割り込み許可

KUp 動作モード (前)	
0	カーネルモード
1	ユーザーモード

KUo 割り込み許可状態 (旧)	
0	割り込み禁止
1	割り込み許可

KUo 動作モード (旧)	
0	カーネルモード
1	ユーザーモード

PMask[2:0] 割り込みマスクレベル (前)	
0~7	割り込みマスクレベル

CMask[2:0] 割り込みマスクレベル (現)	
0~7	割り込みマスクレベル

SwiMask[3:0] ソフトウエア割り込みマスク	
0	割り込み禁止
1	割り込み許可

図 8-3　Status レジスタ

表 8-3　Status レジスタ

ニモニック	名前	リセット後	説　明	リード・ライト
CU[3:0] (Coprocessor Usability)	コプロセッサ使用許可	X	コプロセッサ CP3~CP0 の使用許可状態を制御します。 CU[3:1]ビットは、ユーザーモード、カーネルモードにかかわらず、対応するコプロセッサの使用を許可するかどうか制御します。CU ビットがクリアされているときに、そのコプロセッサに対する命令を実行すると、コプロセッサ使用不可例外が発生します。 CU[0]ビットは、ユーザーモードにおける CP0 命令の使用許可状態を制御します。CU[0]ビットがクリアされているときに、ユーザーモードのプログラムで CP0 命令を実行しようとすると、コプロセッサ使用不可例外が発生します。カーネルモードでは、CU[0]ビットの設定に関係なく、CP0 命令を実行できます。	RW
RE (Reserved)	予約	X	このレジスタに書き込むときは、このビットは 0 にしてください。読み出すと、0 が返されます。	R
BEV (Bootstrap Exception Vector)	ブートストラップ例外ベクタ	1	このビットは、リセット時に 1 にセットされます。BEV = 1 のとき、ブートストラップ代替ベクタには、例外ベクタが使用されます。ブートストラップ代替ベクタは、キャッシュ不可な kseg1 を参照するため、通常、キャッシュ機能が有効になるまえに、診断テストを実行するのに使用されます。BEV ビットは、ソフトウエアによって設定またはクリアできます。BEV = 0 のとき、リセット、ノンマスカブル割り込み、デバッグ例外ベクタはキャッシュ不可な kseg1 を参照し、それ以外の例外ベクタはキャッシュ可能な kseg0 を参照します。詳細は、「第 9 章　例外処理」を参照してください。	RW
Nml (Nonmaskable Interrupt)	ノンマスカブル割り込み	0	ノンマスカブル割り込み信号が L にアサートされると、このビットは 1 にセットされます。このビットは 1 を書き込むことで、0 にクリアできます。	RW
PMask[2:0] CMask[2:0] (Interrupt Mask Level) (Previous / Current)	割り込みマスクレベル	X7	プロセッサの割り込み要求信号のレベルが、現在の割り込みマスクレベル CMask[2:0] より高く、IEc (割り込み許可状態) ビットが 1 のとき、割り込みを受け付け、割り込み例外が発生します。割り込み要求信号のレベルが CMask [2:0] の値以下の場合は、その要求は無視されます。 ハードウエアリセットにより、リセット例外が発生すると、CMask[2:0] は最高レベルの 7 に設定されます。CMask[2:0] と PMask[2:0] は 2 レベルのスタックを構成しており、割り込み例外が発生すると、CMask[2:0]の内容が PMask[2:0]フィールドに保存され、RFE (Restore From Exception) 命令を実行すると、PMask[2:0] が CMask[2:0]に復元されます。詳細は、「第 9 章　例外処理」を参照してください。	RW
SwiMask[3:0] (Software Interrupt Mask)	ソフトウエア割り込みマスク	X	4 つのソフトウエア割り込みを、個別にマスクします。Cause レジスタに、各ソフトウエア割り込みを発生させるためのビットがあります。	RW

ニモニック	名前	リセット後	説　明	リード・ライト
KUo / KUp / KUc (Kernel/User Mode (Old / Previous / Current))	動作モード	XX0	プロセッサの動作モードを示します。0 はカーネルモード、1 はユーザーモードです。KUo、KUp、KUc ビットは 3 レベルのスタックを構成しており、それぞれ旧、直前、現在の動作モード状態を保持します。KUc ビットは、ハードウエアリセットおよび、例外の発生により 0 にクリアされ、カーネルモードに移行します。 例外が発生すると、KUc ビットの内容は KUp ビットにプッシュされ、KUp ビットは KUo ビットにプッシュされます。RFT (Restore From Exception) 命令を実行すると、KUo ビットは KUp ビットに、KUp ビットは KUc ビットにポップされます。KUo ビットは変更されません。詳細は「第 9 章　例外処理」を参照してください。	RW
IEo / IEp / IEc (Interrupt Enable) (Old / Previous / Current)	割り込み許可	X0	IEc ビットは、マスカブル割り込み (ハードウエア、ソフトウエア) の、現在の割り込み許可状態を示します。1 でイネーブル状態、0 でディセーブル状態です。IEo、IEp、IEc ビットは、3 レベルのスタックを構成しており、それぞれ、旧、直前、現在の割り込み許可状態を保持しています。IEc ビットは、ハードウエアリセットおよび例外の発生により 0 にクリアされます。このビットを直接操作するほかに、IE レジスタを使うことにより、割り込みを一括してイネーブルまたはディセーブルできます。 例外が発生すると、IEc の内容は IEp ビットにプッシュされ、IEp ビットは IEo ビットにプッシュされます。RFT (Restore From Exception) 命令を実行すると、IEo ビットは IEp ビットに、IEp ビットは IEc ビットにポップされます。IEc ビットは変化しません。詳細は「第 9 章　例外処理」を参照してください。	RW
0 (Reserved)	予約	0	書き込みで予約ビットは無視され、読み出しで 0 が返されます。	R

8.3.3　Cause レジスタ (13)

Cause レジスタは、直前に発生した例外の原因を表示します。TX19 は、4 つのソフトウエア割り込みを認識します。Cause レジスタの Sw[3:0]ビットは、各ソフトウエア割り込みを発生させるために、ソフトウエアで使用できます。4 つのソフトウエア割り込みには、別々の例外ベクタが割り当てられています (「9.1.3　例外ベクタアドレス」を参照)。

図 8-4に、Cause レジスタの構成を示します。また表 8-4に、Cause レジスタの各ビットの機能を示します。

	31	30	29 28	27 16	15 13	12	11 8	7	6 2	1 0
ニモニック	BD	0	CE	0	IL	0	Sw	0	ExcCode	0
リード・ライト	R	R	R	R	R	R	RW	R	R	R
リセット後	X	0	X	0	X	0	X	0	X	0

注:
X **は不定を表します。**

ExcCode 例外コード		
0	Int	マスカブル割り込み (ソフトウエア・ハードウエア)
4	AdEL	アドレスエラー例外 (ロード・命令フェッチ)
5	AdES	アドレスエラー例外 (ストア)
6	IBE	バスエラー例外 (命令フェッチ)
7	DBE	バスエラー例外 (ロード・ストア)
8	Sys	システムコール例外
9	Bp	ブレークポイント例外
10	RI	予約命令例外
11	CpU	コプロセッサ使用不可例外
12	Ov	整数オーバフロー例外
10-31	(予約)	

Sw[3:0] マスカブルソフトウエア割り込み	
0	割り込み解除
1	割り込み

IL[2:0] マスカブルハードウエア割り込みレベル	
0~7	割り込みレベル

CE[1:0] コプロセッサエラー	
00	コプロセッサ 0
01	コプロセッサ 1
10	コプロセッサ 2
11	コプロセッサ 3

BD 分岐遅延	
1	ジャンプまたは分岐遅延スロットで例外発生

図 8-4　Cause **レジスタ**

表 8-4　Cause レジスタ

ニモニック	名前	リセット後	説　明	リード・ライト
BD (Branch Delay)	分岐遅延	X	ジャンプ、または分岐遅延スロット内の命令を実行しているときに、例外が発生すると、1 にセットされます。	R
CE[1:0] (Coprocessor Error)	コプロセッサエラー	X	コプロセッサ使用不可例外が発生したときに参照されたコプロセッサを示します。	R
IL[2:0] (Interrupt Level)	割り込みレベル	X	マスカブルハードウエア割り込みの割り込み要求レベルを示します。プロセッサに入力される 3 ビットの割り込み要求信号は、割り込みの優先度を表し、Status レジスタの割り込みマスクレベルとは無関係に、Cause レジスタの IL[2:0] フィールドに取り込まれます。割り込み要求レベルがマスクレベルより高く、Status レジスタの IEc (割り込み許可状態) ビットが 1 にセットされている場合、割り込みを受け付け、割り込み例外が発生します。割り込みが何も発生していないときは、IL[2:0] ビットは 0 になります。	R
Sw[3:0] (Maskable Software Interrupt)	マスカブルソフトウエア割り込み	X	ソフトウエア割り込みを発生させるのに、ソフトウエアで使用されます。TX19 は、4 つのソフトウエア割り込みを認識します。また、これらの割り込みに対応して、Status レジスタには割り込みマスクビットがあります。	R
ExcCode (Exception Code)	例外コード	X	直前に発生した例外の原因を示します。図 8-4を参照してください。	RW
0 (Reserved)	予約	—	書き込みを実行しても無視されます。読み出すと 0 が返されます。	R

8.3.4　EPC レジスタ (14)

EPC (Exception Program Counter) レジスタには、例外が発生した時点におけるプログラムカウンタ (PC) の内容が保存されます。ただし、例外発生時に実行されていた命令がジャンプまたは分岐遅延スロット内にある場合は、EPC レジスタには、その直前のジャンプまたは分岐命令のアドレスが格納されます。この場合、Cause レジスタの BD ビットが 1 にセットされます。PC と同様、EPC レジスタの最下位ビットは、例外が発生したときの ISA モードを表します。

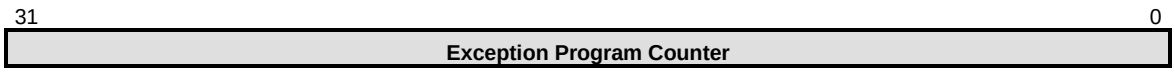


図 8-5　EPC レジスタ

8.3.5　PRId レジスタ (15)

PRId (Product Revision Identifier) レジスタは読み出し専用レジスタで、プロセッサのリビジョンを保持します。図 8-6に、PRId レジスタの構成を示します。また、表 8-5に PRId レジスタの各ビットの意味を示します。

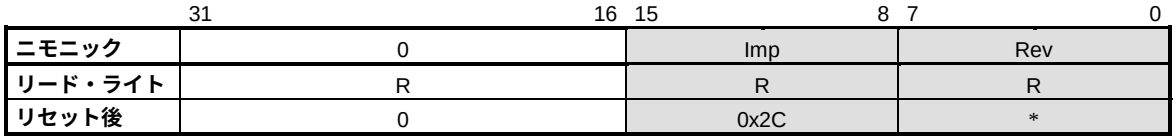


図 8-6　PRId レジスタ

表 8-5　PRId レジスタ

ニモニック	名前	リセット後	説　明	リード・ライト
Imp[7:0] (Implementation Number)	構成要素識別子	0x2C	TX19 プロセッサコアであることを示す 0x2C が入ります。	R
Rev[7:0] (Revision Number)	リビジョン識別子	—	TX19 プロセッサコアのリビジョンを示します。リビジョン番号については、各製品のハードウエアユーザーマニュアルを参照してください。	R
0 (Reserved)	予約	—	書き込みを実行しても無視されます。読み出すと 0 が返されます。	R

8.3.6　IE レジスタ (31)

IE (Interrupt Enable) レジスタは、Status レジスタの IEc (割り込み許可状態) ビットを、セットまたはクリアするのに使用します。IE レジスタに 0 を書き込むと、Status レジスタの IEc ビットがクリアされます。IE レジスタに 0 以外の値を書き込むと、IEc ビットがセットされます。割り込みをディセーブルするには、「MTC0 r0, IE」命令を使用します。割り込みをイネーブルするには、「MTC0 $sp, IE」のようにターゲットレジスタ (*rt*) に 0 以外の値のレジスタを使用します。図 8-7に IE レジスタの構成を示します。

図 8-7　IE レジスタ

IE レジスタを使わずに、直接 Status レジスタの IEc ビットを操作してもかまいませんが、これには以下のようにいくつかの命令を組み合わせて使う必要があります。これに対して、IE レジスタを使えば、上記のように 1 命令で割り込みをイネーブルしたり、ディセーブルしたりできるという利点があります。

```
MFC0    r26,C0_STAUTS
NOP
OR      r26,r26,IEC
MTC0    r26,C0_STATUS
```

ここで C0_STATUS は、Status レジスタを、IEC は定数 Ox0000_0001 を表します。(通常、これらはアセンブラのヘッダーファイルで定義されます。)

8.4　デバッグ例外処理レジスタ

TX19 では、デバッグのため、プログラムの実行を任意に停止させることができます。TX19 には、プログラムのデバッグを容易にするためにレジスタが用意されています。この項では、それらのレジスタについて説明します。

8.4.1　Debug レジスタ (16)

Debug レジスタは、デバッグ例外が発生したときの状態を保持するとともに、デバッグ処理の設定を行うことができます。プログラム中に SDBBP (Software Debug Breakpoint) 命令を埋め込むことにより、SDBBP 命令が実行された時点で、デバッグブレークポイント例外を発生させることができます。また、Debug レジスタの SSt ビットをセットすることで、シングルステップ実行を有効にできます。このとき、1 命令実行するごとにシングルステップ例外が発生します。図 8-8に、Debug レジスタの構成を示します。また、表 8-6に Debug レジスタのビットの機能を示します。

	31	30	29 … 15	14	13	12	11	10	9	8	7 … 6	5 … 2	1	0
ニモニック	DBD	DM	0	NIS	-	OES	TLF	BsF	0	SSt	0	-	DBp	DSS
リード・ライト	R	R	R	R	-	R	R	RW	R	RW	R	-	R	R
リセット後	X	0	0	X	-	X	X	X	0	0	0	-	X	X

注:
X は不定を表します。

DSS デバッグシングルステップ例外	
1	シングルステップ例外発生

DBp デバッグブレークポイント例外	
1	デバッグブレークポイント例外発生

SSt シングルステップ	
0	シングルステップ無効
1	シングルステップ有効

BsF バスエラー例外フラグ	
0	クリア
1	バスエラー例外発生

OES 一般例外発生情態	
1	一般例外発生

NIS ノンマスカブル割り込み例外発生状態	
1	ノンマスカブル割り込み例外発生

DM デバッグモード	
1	デバッグ例外処理中

DBD デバッグ分岐遅延	
1	ジャンプまたは分岐遅延スロットで例外発生

図 8-8　Debug レジスタ

表 8-6　Debug レジスタ

ニモニック	名前	リセット後	説　明	リード・ライト
DBD (Debug Branch Delay)	デバッグ分岐遅延	X	ジャンプまたは分岐遅延スロット内の命令を実行しているときにデバッグ例外が発生すると、1 にセットされます。	R
DM (Debug Mode)	デバッグモード	0	デバッグ例外処理中は 1 にセットされます。DERET (Debug Exception Return) 命令でクリアされます。	R
NIS (Nonmaskable Interrupt Status)	ノンマスカブル割り込み例外	X	デバッグ例外と、ノンマスカブル割り込み例外が同時に発生すると、1 にセットされます。このとき、Status、Cause、EPC、BadVAddr レジスタは、ノンマスカブル割り込み発生後の状態になっていますが、DEPC レジスタは、ノンマスカブル割り込みの例外ベクタアドレス (0xBFC0_0000) になっていません。	R
OES (Other Exception Status)	一般例外	X	デバッグ例外と、リセット・ノンマスカブル割り込み例外以外の一般例外が同時に発生した場合、1 にセットされます。このとき、Status、Cause、EPC、BadVAddr レジスタは、一般例外発生後の状態になっていますが、DEPC レジスタは、同時に発生した例外のベクタアドレスになっていません。	R
BsF (Bus Error Exception Flag)	バスエラー例外発生フラグ	X	デバッグ例外処理中、バスエラー例外が発生したとき 1 にセットされます。このビットは 1 を書き込むことにより、クリアできます。	RW
SSt (Single-step)	シングルステップ例外有効	0	シングルステップ実行の有効・無効を示します。このビットが 1 にセットされていると、1 命令実行するごとに、シングルステップ例外が発生します。ただし、デバッグ例外処理中 (DM = 1) は無効です。	RW
DBp (Debug Breakpoint)	デバッグブレークポイント例外発生	X	デバッグブレークポイント例外が発生したとき、1 にセットされます。	R
DDS (Debug Single-step)	デバッグシングルステップ例外発生	X	シングルステップ例外が発生したとき、1 にセットされます。	R
0 (Reserved)	予約	0	書き込みを実行しても無視されます。読み出すと 0 が返されます。	R
— (Reserved)	予約	X	予約ビットです。	R
TLF (Reserved)	予約	X	予約ビットです。	R

8.4.2　DEPC レジスタ (17)

DEPC (Debug Exception Program Counter) レジスタは、デバッグ例外発生時のプログラムカウンタ (PC) の内容を保存しておくレジスタです。ただし、例外発生時に実行されていた命令が、ジャンプまたは分岐遅延スロット内にある場合は、DEPC レジスタには、その直前のジャンプまたは分岐命令のアドレスが格納されます。この場合、Debug レジスタの DBD ビットが 1 にセットされます。PC と同様、DEPC レジスタの最下位ビットは、例外が発生したときの ISA モードを表します。図 8-9に DEPC レジスタの構成を示します。

図 8-9　DEPC レジスタ

第9章　例外処理

この章では、TX19 アーキテクチャで規定されている例外処理について説明します。この章は以下の項で構成されます。

- 一般例外
- 割り込み
- デバッグ例外

9.1　一般例外

TX19 の例外は、一般例外とデバッグ例外に分類されます。この項では、プログラムのデバッグで使うデバッグ例外以外の例外について、原因、処理などを説明します。

9.1.1　一般例外処理

例外とは、外部割り込み信号、エラー、または命令実行中に生じた異常状態の結果として、通常の命令シーケンスを変更する状態を示します。例外が発生すると、その時点におけるプロセッサの状態を保存し、カーネルモードに切り換えて、あらかじめ決められたアドレスに制御を移します。このアドレスを、例外ベクタといい、例外ハンドラの開始アドレスを示します。

リセット例外以外の一般例外における、TX19 プロセッサの処理を図 9-1に示します。

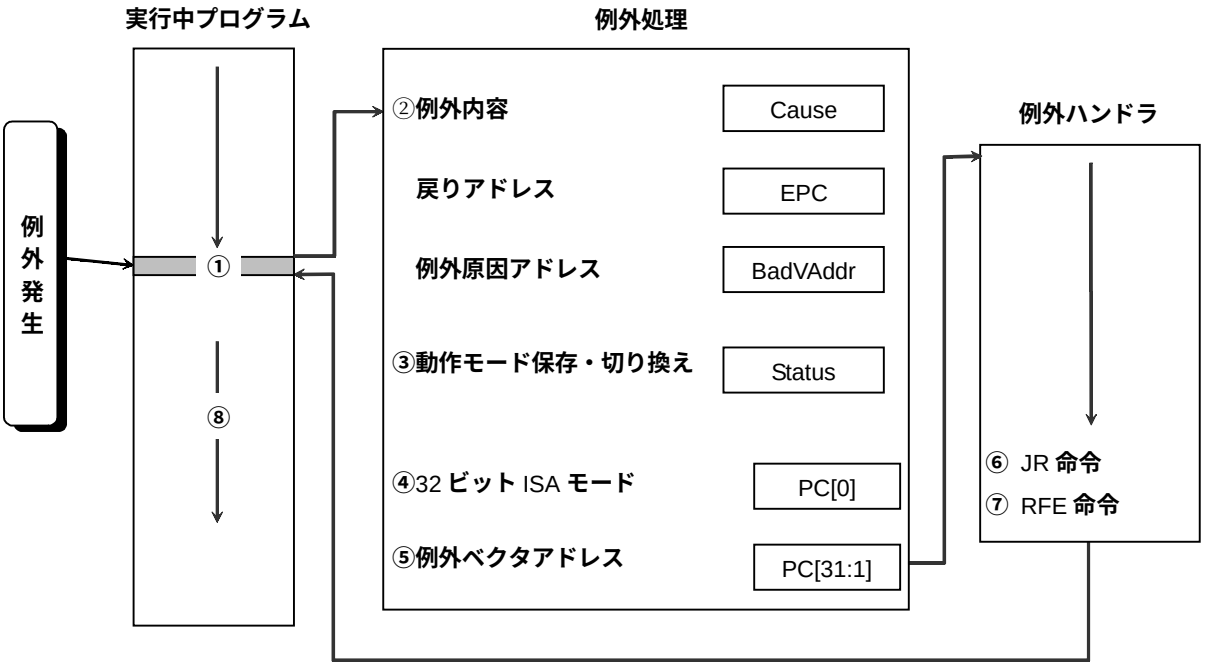


図 9-1　例外処理プロセス

以下の番号は、図 9-1の番号と対応しています。

1. 現在実行中の命令、およびすでに実行が開始されているパイプライン内の命令を中断します。

2. Cause レジスタに、例外の原因についての情報を格納します。複数の例外で共通の例外ベクタを用いているため、例外ハンドラは Cause レジスタを調べて、例外の発生要因を確認します。

   EPC レジスタには、例外が発生した時点におけるプログラムカウンタ (PC) の内容が保存されます。ただし、例外を起こした命令が、ジャンプまたは分岐遅延スロット内の命令である場合は、EPC レジスタには、その直前のジャンプまたは分岐命令のアドレスが格納されます。この場合、Cause レジスタの BD ビットは 1 にセットされます。また、EPC レジスタの最下位ビットには、例外発生時の ISA モードが保存されます。

   アドレスエラー例外の場合は、BadVAddr レジスタに、仮想アドレスから物理アドレスへの変換エラーの原因となった仮想アドレスを格納します。

3. Status レジスタに、プロセッサの現在の動作状態を保存します。また、例外処理用のカーネルモードに切り換え、割り込みを禁止します。

4. 16 ビット ISA モードで例外が発生した場合は、プログラムカウンタ (PC) の最下位ビット (ISA モードビット) を 0 にクリアし、32 ビット ISA モードに切り換えます。

5. 例外ベクタアドレスを PC に設定して、例外ハンドラにジャンプします。

6. 例外処理後、JR (Jump Register) 命令で戻りアドレスへジャンプします。

7. 例外ハンドラの最後で、RFE (Restore From Exception) 命令により、例外発生前の状態を復元します。RFE 命令はジャンプ遅延スロットに置かれます。したがって、実際には、RFE 命令は JR 命令の前に実行されます。

8. 例外が発生した命令から、実行が再開されます。

9.1.2　例外の種類

表 9-1に、TX19 で発生する一般例外の種類を示します。直前に発生した例外の原因は、Cause レジスタの ExcCode フィールドに表示されます。例外の詳細については、9.1.6 ~ 9.1.15項で表 9-1の順序で説明します。

表 9-1　一般例外の種類

ExcCode	例　外	ニモニック	説　明
0	マスカブル割り込み例外	Int	Status レジスタの CMask[2:0]フィールドの値より優先度の高い割り込み要求信号が入力されたとき、または Cause レジスタの Sw[3:0]ビットのいずれかが、ソフトウエアによりセットされたとき、発生します。
—	ノンマスカブル割り込み例外	Nml	NMI*信号の立ち下がりエッジで発生します。
4	アドレスエラー例外 (ロード)	AdEL	以下の場合に発生します。 ・ 境界に位置合わせされていないアドレスからのロード ・ 境界に位置合わせされていないアドレスへのストア ・ ワード境界に位置合わせされていないアドレスからの 32 ビット ISA 命令のフェッチ ・ ユーザーモードで特権領域であるカーネルセグメントへアクセスしたとき
5	アドレスエラー例外 (ストア)	AdES	
6	バスエラー例外 (命令フェッチ)	IBE	バスサイクル中に、バスエラー信号がアサートされたときに、発生します。
7	バスエラー例外 (データ)	DBE	
8	システムコール例外	Sys	SYSCALL 命令を実行すると、発生します。
9	ブレークポイント例外	Bp	BREAK 命令を実行すると、発生します。
10	予約命令例外	RI	主オペコードまたは副オペコードが未定義の命令、または予約命令を実行すると、発生します。
11	コプロセッサ使用不可例外	CpU	Status レジスタの CU ビットがセットされていないときに、対応するコプロセッサに対してコプロセッサ命令を実行しようとすると発生します。
12	整数オーバフロー例外	Ov	加算または減算命令で、2 の補数のオーバフローが発生したときに、発生します。
—	リセット例外	Reset	リセット信号がアサートされ、デアサートされたときに、発生します。
—	デバッグ例外	—	「9.3　デバッグ例外」を参照してください。

† デバッグ例外のニモニックは定義されていません。

9.1.3　例外ベクタアドレス

例外ベクタは、例外ハンドラの開始アドレスです。リセット例外、ノンマスカブル割り込み例外の例外ベクタアドレスは 0xBFC0_0000 です。デバッグ例外での例外ベクタアドレスは 0xBFC0_200 です。その他の例外ベクタアドレスは、Status レジスタの BEV (Bootstrap Exception Vector) ビットにより異なります。表 9-2に例外ベクタアドレスを示します。

表 9-2　例外ベクタアドレス

例外の種類		例外ベクタアドレス			
		BEV = 0		BEV = 1	
		仮想アドレス	物理アドレス	仮想アドレス	物理アドレス
リセット例外 ノンマスカブル割り込み例外		0xBFC0_0000	0x1FC0_0000	0xBFC0_0000	0x1FC0_0000
デバッグ例外		0xBFC0_0200	0x1FC0_0200	0xBFC0_0200	0x1FC0_0200
マスカブル割り込み	ソフトウエア割り込み Swi0	0x8000_0110	0x0000_0110	0xBFC0_0210	0x1FC0_0210
	ソフトウエア割り込み Swi1	0x8000_0120	0x0000_0120	0xBFC0_0220	0x1FC0_0220
	ソフトウエア割り込み Swi2	0x8000_0130	0x0000_0130	0xBFC0_0230	0x1FC0_0230
	ソフトウエア割り込み Swi3	0x8000_0140	0x0000_0140	0xBFC0_0240	0x1FC0_0240
	ハードウエア割り込み	0x8000_0160	0x0000_0160	0xBFC0_0260	0x1FC0_0260
その他の例外		0x8000_0080	0x0000_0080	0xBFC0_0180	0x1FC0_0180

プロセッサがリセットされると、Status レジスタの BEV ビットは 1 に初期化されます。BEV が 1 のとき、例外ベクタはすべてキャッシュ不可な kseg1 領域にあります。通常、この状態は、キャッシュ機能が有効になる前に、診断テストを実行するときに使用されます。BEV ビットはソフトウエアで設定、クリアできます。BEV ビットが 0 のとき、ノンマスカブル割り込み例外、デバッグ例外のベクタアドレスのみキャッシュ不可な kseg1 領域にあり、その他の例外ベクタアドレスはすべてキャッシュ可能な kseg0 領域にあります。

9.1.4　例外の優先順位

1 つの命令に対して、複数の例外が同時に発生する可能性があります。この場合、表 9-3に示す優先順位に基づき、優先度の最も高い例外が 1 つだけ受け付けられます。

表 9-3　例外の優先順位

優先度	例　外	ニモニック
高	リセット例外	Reset
	バスエラー例外 (命令フェッチ)	IBE
	バスエラー例外 (データアクセス)	DBE
	ノンマスカブル割り込み	Nml
	アドレスエラー例外 (命令フェッチ)	AdEL
	コプロセッサ使用不可例外	CpU
	整数オーバフロー例外、システムコール例外、ブレークポイント例外、予約命令例外	Ov, Sys, Bp, RI
	アドレスエラー例外 (ロード)	AdEL
	アドレスエラー例外 (ストア)	AdES
低	マスカブル割り込み	Int

9.1.5　プロセッサコンテキストの保存と復元

Status レジスタのカーネル・ユーザーモード、割り込み許可ビットは 3 レベルのスタック (現モード、前モード、旧モード) を構成しています。KUc ビットはプロセッサの現在の動作モードを示します (0 = カーネルモード、1 = ユーザーモード)。IEc ビットは、マスカブル割り込み (ハードウエア・ソフトウエア) の現在の許可状態を示します (1 = 許可、0 = 禁止)。

例外が発生すると、KUc ビットと IEc ビットは前ビット (KUp・IEp) にプッシュされ、Kup ビットと IEp ビットは旧ビット (KUo・KEo) にプッシュされます。そして、現ビット (KUc・IEc) はクリアされて、カーネルモードの割り込み禁止状態になります。

動作モード・割り込み許可状態は 3 レベルのスタックとして保存されるので、Status レジスタの内容を汎用レジスタ、またはメモリのスタック領域に保存しなくても、2 レベルの例外発生に対応できます。

例外処理後、プロセッサコンテキストを例外が発生する前の状態に復元します。これには、RFE 命令を使います。RFE 命令を実行すると、旧ビット (KUo・IEo) の内容は前ビット (KUp・IEp) にポップされ、前ビット (KUp・IEp) は現ビット (KUc・IEc) にポップされます。旧ビット (KUo・IEo) は変化しません。

さらに、Status レジスタの割り込みマスクレベルフィールドは、2 レベルのスタック (前レベルと現レベル) を構成しています。割り込み要求があると、そのレベルは CMask[2:0] フィールドの値と比べられ、割り込み要求レベルがマスクレベルより大きい場合のみ、割り込み例外が発生します。例外が発生すると、CMask[2:0] フィールドの内容は前フィールド PMask[2:0] にプッシュされ、RFE 命令を実行すると、PMask[2:0] フィールドの値が CMask[2:0]に復元されます。

図 9-1にコプロセッサコンテキストの保存、復帰処理を示します。

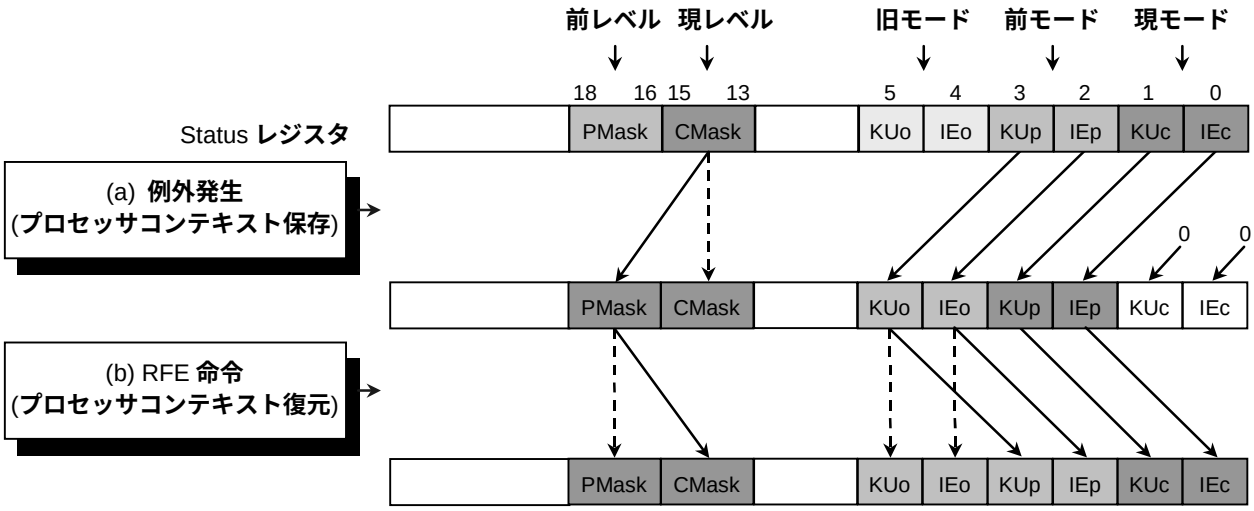


図 9-2　プロセッサコンテキストの保存・復元

例外が発生すると、EPC レジスタには、例外の原因となった命令の仮想アドレスが格納されます。ただし、例外を起こした命令が、ジャンプまたは分岐遅延スロット内の命令である場合は、命令を再実行できるように EPC レジスタには、直前のジャンプまたは分岐命令のアドレスが格納されます。また、EPC レジスタの最下位ビットには、例外発生時の ISA モードビットが保存されます。

通常の例外処理では、Cause、Status、BadVAddr、EPC レジスタは汎用レジスタ、またはメモリのスタック領域に退避します。これにより、最初の例外ハンドラ中で割り込みを再び許可状態にでき、割り込み優先順位をつけることができます。プロセッサが例外を受け付けると、それ以降、割り込みは、自動的に禁止状態になります。そのため、CP0 レジスタを退避させなくても、例外ハンドラを実行することが可能です。しかし、この場合、例外ハンドラを実行することにより、他の例外が発生していないように注意しなければなりません。

例外処理後、JR 命令により、例外が発生したアドレスに戻ります。JR 命令はオペランドとして汎用レジスタしか指定できないので、JR 命令の実行前に、戻りアドレスを汎用レジスタに設定しておかなければなりません。JR 命令は、戻りアドレスと ISA モードビットを PC に復元します。

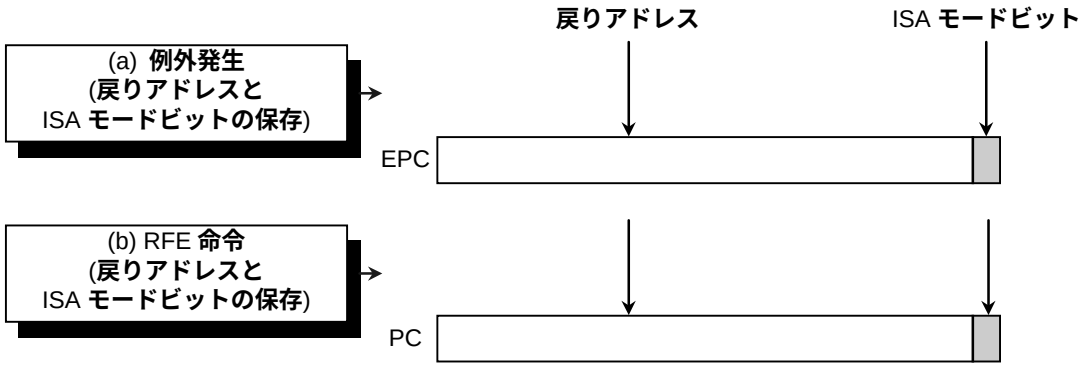


図 9-3　ISA モードの保存と復元

9.1.6　マスカブル割り込み例外

■　**原因**

マスカブル割り込み例外は、マスカブル割り込み (ソフトウエアまたはハードウエア) が発生した場合に発生します。割り込みの受け付けについては、「9.2 割り込み」を参照してください。

■　**処理**

図 9-4に、この例外の処理で使われる CP0 レジスタのフィールドを示します。

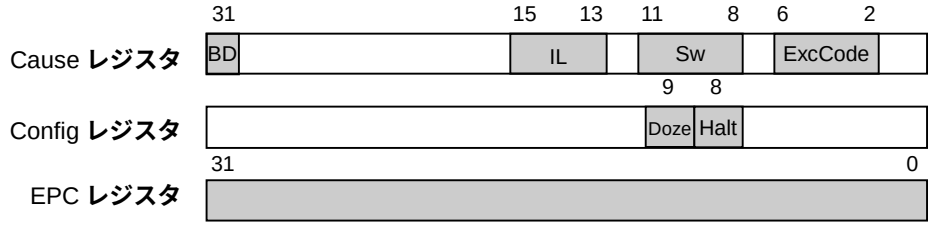


図 9-4　マスカブル割り込み例外

1. Cause レジスタの ExcCode フィールドに Int (0) が設定されます。

2. ハードウエア割り込みを発生した場合は、Cause レジスタの IL フィールドにその割り込み要求レベルが格納されます。ソフトウエアが割り込みを発生した場合は、Sw フィールドに発生状態がセットされます。複数の割り込みが同時にセットされる場合もあります。

3. ハードウエア割り込みの場合、Config レジスタの Halt ビットと Doze ビットがクリアされます。

4. EPC レジスタに、割り込みを発生した時点のプログラムカウンタ (PC) の内容を保存します。ただし、割り込みを発生した命令が、ジャンプまたは分岐遅延スロット内の命令である場合は、EPC レジスタには、直前のジャンプまたは分岐命令のアドレスが格納されます。この場合、Cause レジスタの BD ビットが 1 にセットされます。また、EPC レジスタの最下位ビットには、例外が発生したときの ISA モードが保存されます。

5. Status レジスタ内でプロセッサコンテキストが退避されます。KUc ビットと IEc ビットがクリアされ、これによりカーネルモード割り込み禁止状態になります (「9.1.5　プロセッサコンテキストの保存と復元」を参照してください)。

6. 16 ビット ISA モードで例外が発生した場合は、32 ビット ISA モードに切り換えられます。

7. ハードウエア割り込みの場合は、例外ベクタアドレス 0x8000_0160 に、ソフトウエア割り込みの場合は、対応する例外ベクタアドレスにジャンプし、例外ハンドラに制御が移ります。

8. ハードウエア割り込みの場合は、例外ハンドラは周辺の割り込みコントローラ内の割り込みベクタレジスタを読み出すことにより、割り込みの発生原因を調べ、割り込みハンドラに制御を移します。このとき、割り込み要求レベルの値が Status レジスタの Cmask フィールドにセットされます。
   また、割り込みベクタレジスタを読み出す前に割り込み要求レベルを低い方向に変化させると正常な割り込みができない場合があります。

ソフトウエア割り込みの場合は、割り込み状態は、Cause レジスタの対応する Sw ビットをクリアすることで解除します。ハードウエア割り込みの場合は、プロセッサの割り込み要求信号がアサートされた原因を取り除くことにより、割り込み状態を解除します。

9.1.7　ノンマスカブル割り込み例外

■　**原因**

ノンマスカブル割り込み例外は、プロセッサのノンマスカブル割り込み信号がアサートされたときに発生します。

■　**処理**

図 9-5に、この例外の処理で使われる CP0 レジスタのフィールドを示します。

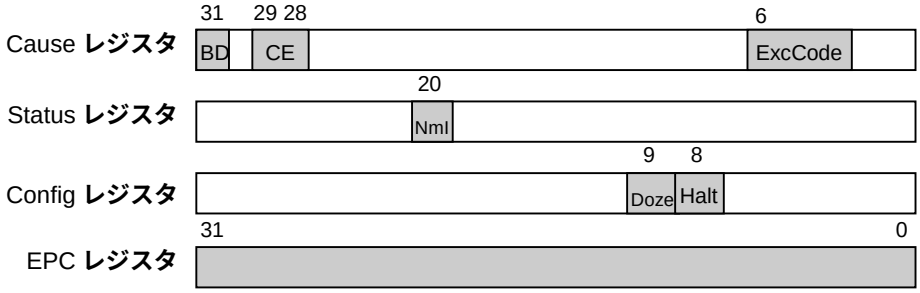


図 9-5　ノンマスカブル割り込み例外

1. Cause レジスタの ExcCode (Exception Code) ビットと CE (Coprocessor Error) ビットは不定なります。
2. Cause レジスタの NmI (Nonmaskable Interrupt) ビットが 1 にセットされます。
3. Config レジスタの Halt ビットと Doze ビットがクリアされます。
4. EPC レジスタに、割り込みを発生した時点のプログラムカウンタ (PC) の内容を保存します。ただし、割り込みを発生した命令が、ジャンプまたは分岐遅延スロット内の命令である場合は、EPC レジスタには、直前のジャンプまたは分岐命令のアドレスが格納されます。この場合、Cause レジスタの BD ビットが 1 にセットされます。また、EPC レジスタの最下位ビットには、例外が発生したときの ISA モードが保存されます。
5. Status レジスタ内でプロセッサコンテキストが退避されます。KUc ビットと IEc ビットがクリアされ、これによりカーネルモード割り込み禁止状態になります (「9.1.5　プロセッサコンテキストの保存と復元」を参照してください)。
6. 16 ビット ISA モードで例外が発生した場合は、32 ビット ISA モードに切り換えられます。
7. 例外ベクタアドレス 0xBFC0_0000 にジャンプし、例外ハンドラに制御が移ります。

バスサイクル中にノンマスカブル割り込み例外が発生した場合は、リセット例外を除く例外と同様、現在のバスサイクルの終端で割り込み要求を認識します。

9.1.8　アドレスエラー例外

■　**原因**

アドレスエラー例外は、以下の場合に発生します。

- ワード境界に位置合わせされていない ISA32 ビット命令をフェッチしようとしたとき
- ハーフワード境界に位置合わせされていない ISA16 ビット命令をフェッチしようとしたとき
- ワード境界に位置合わせされていないワードをロード、またはストアしようとしたとき
- ハーフワード境界に位置合わせされていないハーフワードをロード、またはストアしようとしたとき
- ユーザーモードにおいてカーネルセグメント (kseg0、kseg1、kseg2) を参照しようとしたとき

命令フェッチでは、命令の種類にかかわらずアドレスエラー例外が発生する可能性があります。また、LB、LBU、LH、LHU、LW、LWL、LWR、SB、SH、SW、SWL、SWR 命令では、それ以外の原因でもアドレスエラー例外が発生する可能性があります。

■　**処理**

図 9-6に、この例外の処理で使われる CP0 レジスタのフィールドを示します。

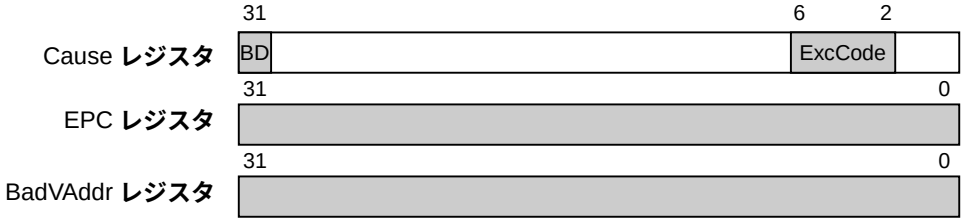


図 9-6　アドレスエラー例外

1. 例外が、命令フェッチまたはロード (AdEL) により発生したのか、ストア (AdES) によって発生したのかにより、Cause レジスタの ExcCode フィールドに、AdEL (4) または AdES (5) が設定されます。
2. EPC レジスタに、例外が発生した時点のプログラムカウンタ (PC) の内容を保存します。ただし、割り込みを発生した命令が、ジャンプまたは分岐遅延スロット内の命令である場合は、EPC レジスタには、直前のジャンプまたは分岐命令のアドレスが格納されます。この場合、Cause レジスタの BD ビットが 1 にセットされます。また、EPC レジスタの最下位ビットには、例外が発生したときの ISA モードが保存されます。

3. BadVAddr レジスタには、例外の原因となった仮想アドレス、または不正アクセスしたカーネルセグメントの仮想アドレスが保存されます。

4. Status レジスタ内でプロセッサコンテキストが退避されます。KUc ビットと IEc ビットがクリアされ、これによりカーネルモード割り込み禁止状態になります(「9.1.5　プロセッサコンテキストの保存と復元」を参照してください)。

5. 16 ビット ISA モードで例外が発生した場合、32 ビット ISA モードに切り換えられます。

6. 例外ベクタアドレス 0x8000_0080 にジャンプし、例外ハンドラに制御が移ります。

9.1.9　バスエラー例外

■　**原因**

バスエラー例外は、メモリバスサイクル中に、バスエラー信号が入力されたとき発生します。

命令フェッチでは、命令の種類にかかわらず、バスエラー例外が発生する可能性があります。また、LB、LBU、LH、LHU、LW、LWL、LWR、SB、SH、SW、SWL、SWR 命令では、ロードまたはストア中にバスエラー例外が発生する可能性があります。

■　**処理**

図 9-7に、この例外の処理で使われる CP0 レジスタのフィールドを示します。

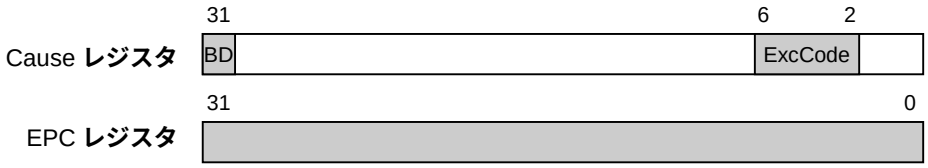


図 9-7　バスエラー例外

1. 例外が、命令フェッチ (IBE) により発生したのか、データのロードまたはストア (DBE) により発生したのかにより、Cause レジスタの ExcCode フィールドには、IBE (6) または DBE (7) が設定されます。

2. 以下の場合、EPC レジスタに、例外が発生した時点のプログラムカウンタ (PC) の内容が保存されます。
   ・ロード命令の直後に SYNC 命令を置いた場合
   ・ロード直後の命令がロードされたデータに依存する場合

   これらの場合、ロード命令が終了するまで、直後の命令はストールします。EPC

レジスタには、ロード命令直後の命令のアドレスが格納されます。

バスタイムアウト、バックプレーンバスパリティエラーなど上記以外の場合は、EPC レジスタは不定になります。例外の原因となった命令のアドレスを知る必要がある場合、外部のハードウエアでアドレスを保存する必要があります。

3. Status レジスタ内でプロセッサコンテキストが退避されます。KUc ビットと IEc ビットがクリアされ、これによりカーネルモード割り込み禁止状態になります(「9.1.5　プロセッサコンテキストの保存と復元」を参照してください)。
4. 16 ビット ISA モードで例外が発生した場合は、32 ビット ISA モードに切り換えられます。
5. 例外ベクタアドレス 0x8000_0080 にジャンプし、例外ハンドラに制御が移ります。

バスエラーが通知されると、メモリバスサイクルはただちにアボートされます。バーストリフィル中にバスエラーが発生した場合、それ以降のキャッシュブロックのリフィルは実行されません。

TX19 プロセッサコアは、自分自身が実行するバスサイクル中でしか、バスエラーを認識しません。したがって、ライトバッファユニットを内蔵している製品では、ライトオペレーション中は、バスエラー例外は発生しません。この場合、一般的な割り込み要求信号を使って、バスオペレーションを中止しなければなりません。

ロード命令でバスエラーが発生すると、デスティネーションレジスタの内容は、不定になります。

9.1.10 システムコール例外

■ **理由**

システムコール例外は、SYSCALL 命令を実行すると発生します。

■ **処理**

図 9-8に、この例外の処理で使われる CP0 レジスタのフィールドを示します。

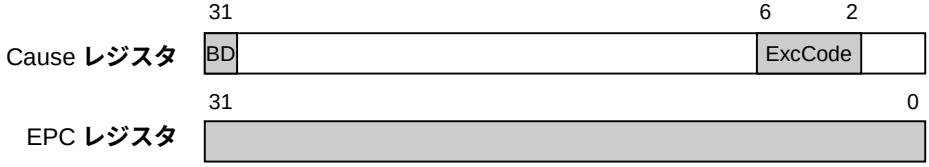


図 9-8　システムコール例外

1. Cause レジスタの ExcCode フィールドに Sys (8) が設定されます。

2. EPC レジスタに、例外が発生した時点のプログラムカウンタ (PC) の内容を保存します。ただし、割り込みを発生した命令が、ジャンプまたは分岐遅延スロット内の命令である場合は、EPC レジスタには、直前のジャンプまたは分岐命令のアドレスが格納されます。この場合、Cause レジスタの BD ビットが 1 にセットされます。また、EPC レジスタの最下位ビットには、例外が発生したときの ISA モードが保存されます。

3. Status レジスタ内でプロセッサコンテキストが退避されます。KUc ビットと IEc ビットがクリアされ、これによりカーネルモード割り込み禁止状態になります (「9.1.5　プロセッサコンテキストの保存と復元」を参照してください)。

4. 例外ベクタアドレス 0x8000_0080 にジャンプし、例外ハンドラに制御が移ります。

システムコール例外が発生すると、例外ハンドラに制御が移ります。SYSCALL 命令の未使用ビット (ビット 25~6) を使って、例外ハンドラに情報を渡すことができます。これらのビットを調べるには、EPC レジスタが指している命令をデータとしてロードします。ただし、例外を発生した命令が、ジャンプまたは分岐遅延スロット内にある (Cause レジスタの BD ビットが 1 にセットされている) 場合は、EPC レジスタの値に 4 を加えなければなりません。

例外ハンドラから戻る際に、SYSCALL 命令が再び実行されないように、EPC レジスタ内のアドレスに 4 を加えなければなりません。SYSCALL 命令が、ジャンプまたは分岐遅延スロット内にある (Cause レジスタの BD ビットが 1 にセットされている) 場合は、戻りアドレスの命令は、直前のジャンプまたは分岐命令になります。この場合、ジャンプまたは分岐命令を解釈し、EPC レジスタを設定し直さなければなりません。

9.1.11 ブレークポイント例外

■ **原因**

ブレークポイント例外は、BREAK 命令を実行すると発生します。

■ **処理**

図 9-9に、この例外の処理で使われる CP0 レジスタのフィールドを示します。

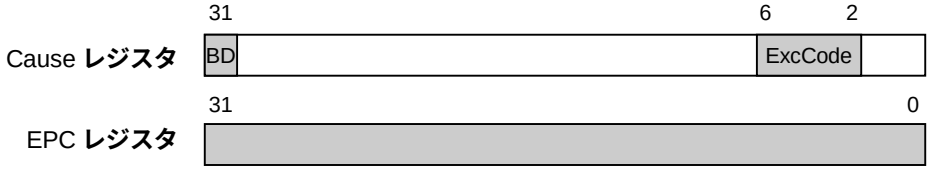


図 9-9　ブレークポイント例外

1. Cause レジスタの ExcCode フィールドに Bp (9) が設定されます。

2. EPC レジスタに、例外が発生した時点のプログラムカウンタ (PC) の内容を保存します。ただし、割り込みを発生した命令が、ジャンプまたは分岐遅延スロット内の命令である場合は、EPC レジスタには、直前のジャンプまたは分岐命令のアドレスが格納されます。この場合、Cause レジスタの BD ビットが 1 にセットされます。また、EPC レジスタの最下位ビットには、例外が発生したときの ISA モードが保存されます。

3. Status レジスタ内でプロセッサコンテキストが退避されます。KUc ビットと IEc ビットがクリアされ、これによりカーネルモード割り込み禁止状態になります (「9.1.5　プロセッサコンテキストの保存と復元」を参照してください)。

4. 16 ビット ISA モードで例外が発生した場合、32 ビット ISA モードに切り換えられます。

5. 例外ベクタアドレス 0x8000_0080 にジャンプし、例外ハンドラに制御が移ります。

ブレークポイント例外が発生すると、例外ハンドラに制御が移ります。BREAK 命令の未使用ビット (32 ビット ISA の場合はビット 25~6、16 ビット ISA の場合はビット 10~5) を使って、例外ハンドラに情報を渡すことができます。これらのビットを調べるには、EPC レジスタが指している命令をデータとしてロードします。ただし、例外を発生した命令が、ジャンプまたは分岐遅延スロット内にある (Cause レジスタの BD ビットが 1 にセットされている) 場合は、EPC レジスタの値に 4 を加えなければなりません。

例外ハンドラから戻る際に、BREAK 命令が再び実行されないように、EPC レジスタ内のアドレスに 4 (32 ビット ISA) または 2 (16 ビット ISA) を加えなければなりません。BREAK 命令が、ジャンプまたは分岐遅延スロット内にある (Cause レジスタの BD ビットが 1 にセットされている) 場合は、戻りアドレスの命令は、直前のジャンプまたは分岐命令になります。この場合、ジャンプまたは命令を解釈し、EPC レジスタを設定し直さなければなりません。

9.1.12　予約命令例外

■　**原因**

32 ビット ISA モードでは、以下の場合、予約命令例外が発生します。

- 主オペコード (ビット 31~26) が未定義の命令、または副オペコード (ビット 5~0) が未定義の SPECIAL 命令を実行したとき
- 予約命令 (LWC*z*、SWC*z*) を実行したとき

16 ビット ISA モードでは、以下の場合、予約命令例外が発生します。

- コードが 1110_1xxx_yyy0_1001、1110_1xxx_yyy1_0001、

1110_1xxx_yyy1_0101、1100_100i_iiii_iiii、0110_0110_iiii_iiii である未定義命令を実行したとき

- 予約命令 (LWU、LD、SD、DADDU、DSUBU、DADDIU、DMULT、DMULTU、DDIV、DDIVU、DSLL、DSRL、DSRA、DSLLV、DSRLV、DSRAV) を実行したとき
- 拡張できない命令に対して EXTEND 命令を実行したとき

■　**処理**

図 9-10に、この例外の処理で使われる CP0 レジスタのフィールドを示します。

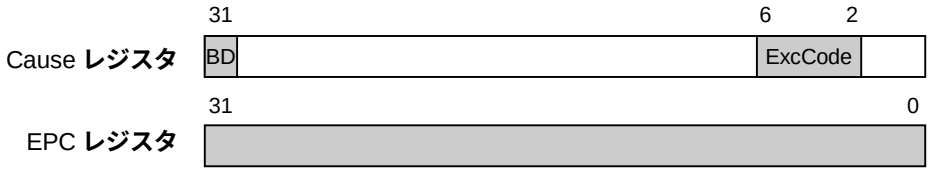


図 9-10　予約命令例外

1. Cause レジスタの ExcCode フィールドに RI (10) が設定されます。
2. EPC レジスタに、例外が発生した時点のプログラムカウンタ (PC) の内容を保存します。ただし、割り込みを発生した命令が、ジャンプまたは分岐遅延スロット内の命令である場合は、EPC レジスタには、直前のジャンプまたは分岐命令のアドレスが格納されます。この場合、Cause レジスタの BD ビットが 1 にセットされます。また、EPC レジスタの最下位ビットには、例外が発生したときの ISA モードが保存されます。
3. Status レジスタ内でプロセッサコンテキストが退避されます。KUc ビットと IEc ビットがクリアされ、これによりカーネルモード割り込み禁止状態になります (「9.1.5　プロセッサコンテキストの保存と復元」を参照してください)。
4. 16 ビット ISA モードで例外が発生した場合、32 ビット ISA モードに切り換えられます。
5. 例外ベクタアドレス 0x8000_0080 にジャンプし、例外ハンドラに制御が移ります。

TX19 は、アドレス変換にダイレクトセグメントマッピング方式を用いています。TLB を内蔵していません。TLB 関連の命令を検出すると、NOP (No Operation) 動作が行われ、予約命令例外は発生しません。

9.1.13 コプロセッサ使用不可例外

■ **理由**

コプロセッサ使用不可例外は、以下の場合発生します。

- Status レジスタ内の CU[*z*] ビットが 0 のとき、対応するコプロセッサユニット CPz に対して、コプロセッサ命令を実行したとき (*z* はコプロセッサ番号 0~3)
- CU[0] ビットが 0 のとき、ユーザーモードで CP0 命令を実行したとき

コプロセッサ使用不可例外は、コプロセッサ命令 LWC*z*、SWC*z*、MTC*z*、MFC*z*、CTC*z*、CFC*z*、COP*z*、BC*z*T、BC*z*F、BC*z*TL、BC*z*FL、システム制御コプロセッサ (CP0) 命令 MTC0、MFC0、RFE、COP0 で発生します。

カーネルモードでは、Status レジスタの CU[0]ビットの設定に関係なく、CP0 命令を実行してもコプロセッサ使用不可例外は発生しません。

■ **処理**

図 9-11に、この例外の処理で使われる CP0 レジスタのフィールドを示します。

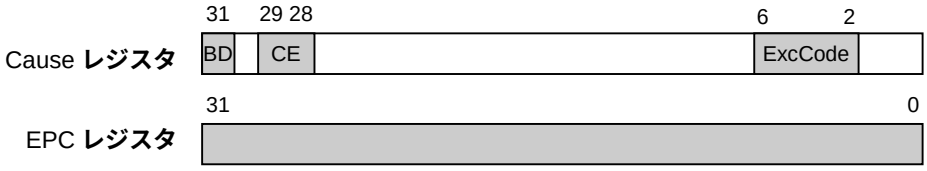


図 9-11　コプロセッサ使用不可例外

1. Cause レジスタの ExcCode フィールドに CpU (11) が設定されます。
2. Cause レジスタの CE フィールドに、例外発生時に参照されたコプロセッサ番号が格納されます。
3. EPC レジスタに、例外が発生した時点のプログラムカウンタ (PC) の内容を保存します。ただし、割り込みを発生した命令が、ジャンプまたは分岐遅延スロット内の命令である場合は、EPC レジスタには、直前のジャンプまたは分岐命令のアドレスが格納されます。この場合、Cause レジスタの BD ビットが 1 にセットされます。また、EPC レジスタの最下位ビットには、例外が発生したときの ISA モードが保存されます。
4. Status レジスタ内でプロセッサコンテキストが退避されます。KUc ビットと IEc ビットがクリアされ、これによりカーネルモード割り込み禁止状態になります (「9.1.5　プロセッサコンテキストの保存と復元」を参照してください)。
5. 16 ビット ISA モードで例外が発生した場合、32 ビット ISA モードに切り換えら

れます。

6. 例外ベクタアドレス 0x8000_0080 にジャンプし、例外ハンドラに制御が移ります。

9.1.14 整数オーバフロー例外

■ **原因**

整数オーバフロー例外は、ADD、ADDI、SUB 命令の結果が 2 の補数のオーバフローになると発生します。

■ **処理**

図 9-12に、この例外の処理で使われる CP0 レジスタのフィールドを示します。

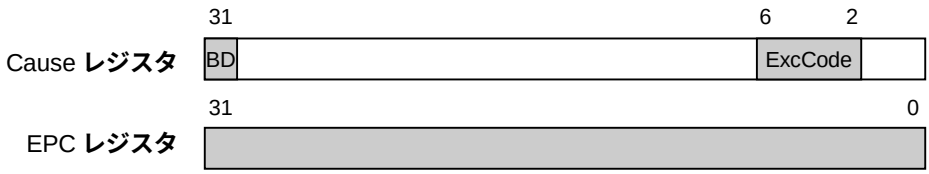


図 9-12　整数オーバフロー例外

1. Cause レジスタの ExcCode フィールドに Ov コード (12) が設定される。
2. EPC レジスタに、例外が発生した時点のプログラムカウンタ (PC) の内容を保存します。ただし、割り込みを発生した命令が、ジャンプまたは分岐遅延スロット内の命令である場合は、EPC レジスタには、直前のジャンプまたは分岐命令のアドレスが格納されます。この場合、Cause レジスタの BD ビットが 1 にセットされます。また、EPC レジスタの最下位ビットには、例外が発生したときの ISA モードが保存されます。
3. Status レジスタ内でプロセッサコンテキストが退避されます。KUc ビットと IEc ビットがクリアされ、これによりカーネルモード割り込み禁止状態になります (「9.1.5　プロセッサコンテキストの保存と復元」を参照してください)。
4. 16 ビット ISA モードで例外が発生した場合、32 ビット ISA モードに切り換えられます。
5. 例外ベクタアドレス 0x8000_0080 にジャンプし、例外ハンドラに制御が移ります。

9.1.15 リセット例外

■ **原因**

プロセッサのリセット信号がアサートされ、デアサートされると、リセット例外が発生します。

■ **処理**

1. すべての CP0 レジスタが、8 章で示したように初期化されます。
2. 例外ベクタアドレス 0xBFC0_0000 にジャンプして、例外ハンドラへ制御を移します。

バスサイクル中にリセット例外が発生すると、プロセッサはバスサイクルをただちに終了して、リセット例外を発生します。

9.2 割り込み

TX19 には、ノンマスカブル割り込み、マスカブルハードウエア割り込み、マスカブルソフトウエア割り込みがあります。この項では、割り込みの種類、優先度、割り込みの受け付けについて説明します。

9.2.1 割り込み種類

TX19 は、ノンマスカブル割り込み、7 レベルのマスカブルハードウエア割り込み、4 本のマスカブルソフトウエア割り込みを認識します。割り込み例外の処理は、ハードウエアで行われ、その後、割り込みハンドラに制御が移されます。割り込み例外の処理については、「9.1.6　マスカブル割り込み例外」と「9.1.7　ノンマスカブル割り込み例外」を参照してください。

ノンマスカブル割り込みは、入力端子 NMI*の立ち下がり、またはウォッチドッグタイマなどの周辺回路からの要求により発生します。ノンマスカブル割り込みの発生原因については、お使いの製品のマニュアルを参照してください。ノンマスカブル割り込みは、緊急性の高い処理を行うための割り込みで、ソフトウエアでマスクできません。ノンマスカブル割り込みは、かならず受け付けられ、強制的に制御がアドレス 0xBFC0_0000 に移ります。

マスカブルハードウエア割り込みは、プロセッサの 3 ビット割り込み要求端子からの要求により発生します。割り込み要求は、外部またはチップ上の周辺回路から出され、割り込みコントローラに入力されます。割り込みコントローラは、割り込み要求を優先度を表す 3 ビットの値に変換し、TX19 プロセッサコアに入力します。プロセッサは、入力された割り込み要求のレベルを現在の割り込みマスクレベル (Status レジスタの CMask[2:0] フィールド) と比較し、割り込み要求レベルがマスクレベルより大きい場合、割り込みを受け付けます。割り込みを認識すると、マスクレベルが更

新されます。

ソフトウエア割り込みには、Swi0~Swi3 の 4 本があります。ソフトウエア割り込みは、Cause レジスタの対応するビットを 1 にセットすることにより発生します。アプリケーションプログラムは、これらのビットを使って割り込みを要求することができます。また、Status レジスタには、各ソフトウエア割り込みを個別にマスクするビットが用意されています。

Status レジスタの割り込み許可ビット IEc により、すべてのマスカブル割り込みを一括して制御できます。

9.2.2　マスカブル割り込みの優先順位

TX19 は、マスカブル割り込みに対して優先順位をつけます。ハードウエア割り込みは、3 ビットの信号により 0 (2 進数 000) から 7 (2 進数 111) までの 8 段階のレベルを設定できます。値が大きいほど優先度が高くなります。ただし、0 レベルの割り込みでは、割り込みは発生しないので、使われることはありません。さまざまな割り込み要因の優先順位は、割り込みコントローラ内の割り込みモードコントロールレジスタで設定します。

ソフトウエア割り込み Swi0~Swi2 はレベル 1 で、Swi3 はレベル 4 です。Swi0、Swi1、Swi2 は同レベルですが、複数のソフトウエア割り込み要求が同時に発生した場合は、Swi2 は Swi1 より、また Swi1 は Swi0 より優先順位が高くなります。

9.2.3　マスカブル割り込みベクタ

4 本のソフトウエア割り込みは、表 9-2に示したように、別々の例外ベクタアドレスをもっています。ハードウエア割り込みの場合は、例外ベクタアドレスは 0x8000_0160 に固定されています。割り込み例外ハンドラは、割り込みコントローラを調べ、割り込み要因に対応した割り込みベクタアドレスを読み出し、制御をそのアドレスに移します。

9.2.4　マスカブル割り込みの受け付け

マスカブル割り込みは、以下の場合に発生します。

- 割り込みが許可状態になっている (Status レジスタの IEc ビットが 1 にセットされている)。
- 割り込み要求レベルが、Status レジスタの CMask[2:0] フィールドで設定されているマスクレベルより大きい。
- ソフトウエア割り込みの場合、Status レジスタの対応するマスクビット (SwiMask [3:0]) が 0 にクリアされている。

同じレベルのハードウエア割り込みとソフトウエア割り込みが発生した場合、ハードウエア割り込みが優先されます。

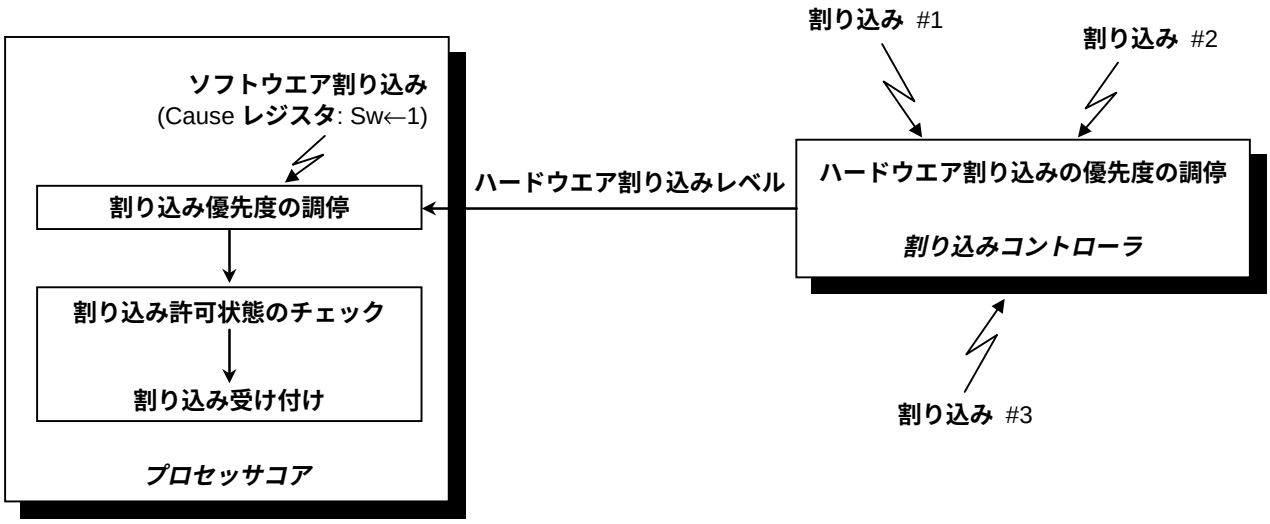


図 9-13　マスカブル割り込みの受け付け

9.2.5　割り込みマスクレベル

プロセッサは割り込みを受け付けたのち、その割り込み要求レベルを Status レジスタの割り込みマスクフィールド CMask[2:0] に自動的に設定します。これにより、割り込みを処理しているあいだ、それよりも高位の割り込みでなければ受け付けなくなります。ソフトウエア割り込みの場合は、割り込みを受け付けると同時に、マスクされます。ハードウエア割り込みの場合は、プロセッサが割り込みベクタを読み出すと同時に、割り込みマスクレベルが設定されます。

したがって、プロセッサコアが割り込みを受け付けてから CMask[2:0] フィールドに書き込むまでのあいだに、より高位の割り込みが発生すれば、その割り込みが優先して受け付けられます。

Status レジスタには、現・前の割り込みマスクレベルフィールドがあります。プロセッサが割り込みを受け付けると、CMask[2:0] の内容が、PMask[2:0] に保存されます。割り込みルーチンから復帰する前に、RFE (Restore From Exception) 命令を実行することにより、PMask[2:0] の値を CMask[2:0] に復元することにより、割り込みマスクレベルが、割り込みが受け付けられる前のレベルに戻ります。

プロセッサは割り込みを受け付けると、IEc ビットを自動的にクリアして、割り込みを禁止状態にします。割り込みマスクレベルが設定されたら、IEc ビットを再度 1 にセットして、より高位の割り込みの受け付けを許可することができます。

9.3　デバッグ例外

TX19 のデバッグ例外には、シングルステップ例外とデバッグブレークポイント例外があります。この項では、デバッグ例外の要因と処理について説明します。

9.3.1　デバッグ例外処理プロセス

TX19 では、デバッグのため、プログラムの実行を任意に停止させることができます。ブレークポイント例外は、SDBBP (Software Debug Breakpoint) 命令を実行すると発生します。シングルステップ例外は、Debug レジスタの SSt ビットをセットすると発生します。

デバッグ例外処理は、図 9-14に示す順序で実行されます。

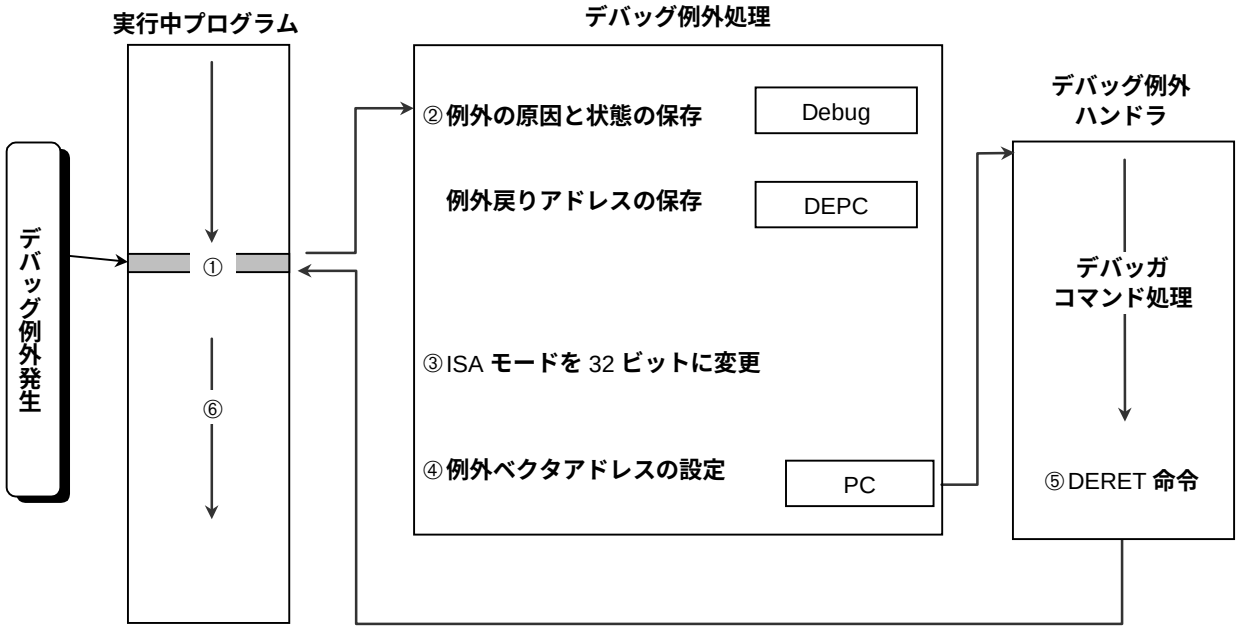


図 9-14　例外処理

1. 現在実行中の命令、およびすでに実行が開始されているパイプライン内の命令を中断します。

2. デバッグ例外用のレジスタは、デバッグに関する情報を保存します。

   - Debug 例外レジスタは、デバッグ例外の原因と、デバッグ例外が現在処理中であるかどうかを示します。

   - DEPC レジスタには、デバッグ例外の原因となった命令の仮想アドレスが格納されます。ただし、例外を発生した命令が、ジャンプまたは分岐遅延スロット内の命令である場合は、DEPC レジスタには直前のジャンプまたは分岐命令のアドレスが格納されます。この場合、Debug レジスタの DBD ビットが 1 にセットされます。また、DEPC レジスタの最下位ビットは、例外が発生したときの ISA モードが保存されます。

3. プロセッサは例外処理用のカーネルモードに切り換えられ、Status レジスタの設定とは無関係に、割り込みが禁止されます。16 ビット ISA モードで例外が発生した場合は、PC の最下位ビット (ISA モード) は 0 にクリアされ、32 ビット ISA モードに切り換えられます。

4. デバッグ例外ベクタアドレスを PC に設定して、デバッグ例外ハンドラの開始アドレスにジャンプします。

5. デバッグ例外ハンドラの処理を終了したら、DERET 命令を実行して、DEPC レジスタに格納されている戻りアドレスへジャンプします。

6. 例外が発生したときプロセッサが実行を中断したアドレスから処理が再開されます。

9.3.2　デバッグ例外の種類

表 9-4に、TX19 で発生するデバッグ例外の種類を示します。

表 9-4　デバッグ例外の種類

例　外	説　明
シングルステップ例外	Debug レジスタの SSt ビットが 1 にセットされているとき、次の命令の開始前に発生します。
デバッグブレークポイント例外	プログラムに埋め込まれた SDBBP 命令を実行すると発生します。ただし、Debug レジスタの SSt ビットが 1 にセットされている場合は、シングルステップ例外が優先されます。デバッグ例外の処理中 (Debug レジスタの DM ビットが 1 にセットされている) は、SDBBP 命令の動作は未定義です。

9.3.3　デバッグ例外の優先順位

シングルステップ例外とデバッグブレークポイント例外は、同時には発生しません。シングルステップ例外は、デバッグブレークポイント例外より優先順位が高くなっています。

デバッグ例外と一般例外は、同時に発生することがあります。この場合、プロセッサは最初にデバッグ例外を処理しますが、この時点で、Status、Cause、EPC、BadVAddr レジスタは、同時に発生した一般例外の情報に更新されています。また、Debug レジスタの NIS ビットまたは OES ビットが 1 にセットされ、ノンマスカブル割り込み例外、またはその他の一般例外が同時に発生したことを示します。

デバッグ例外と一般例外は、図 9-15に示す順序で処理されます。

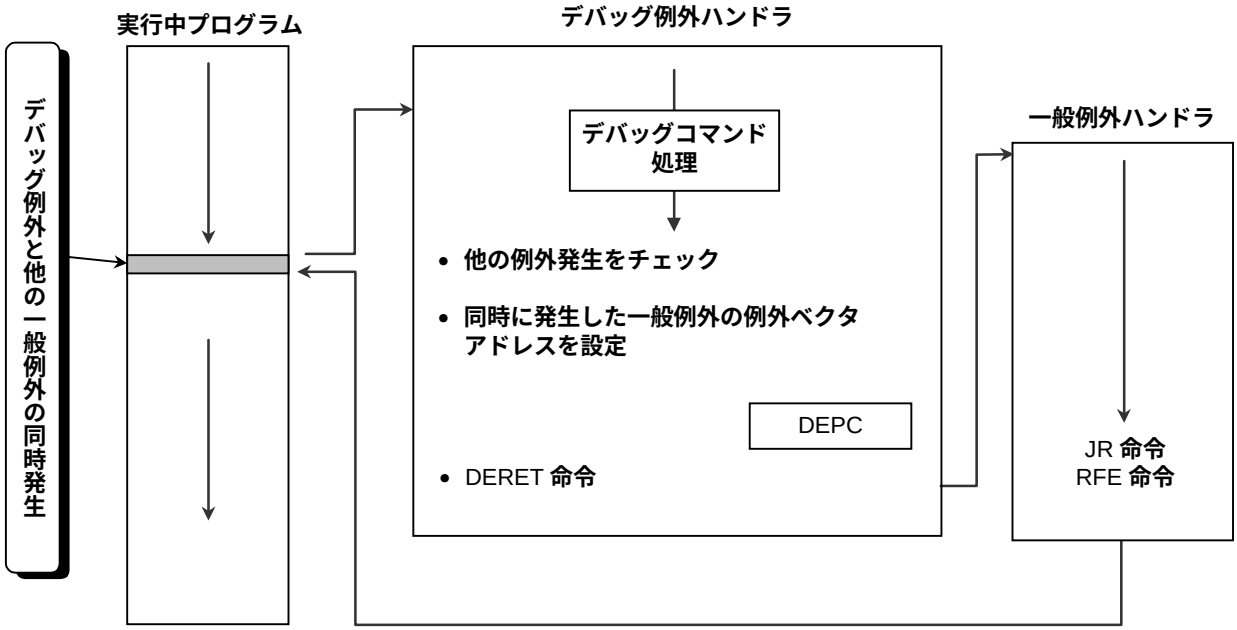


図 9-15　デバッグ例外の優先順位

デバッグ例外が発生すると、DEPC レジスタには、例外の原因となった命令のアドレスが保存されます。デバッグ例外の処理が終わったら、同時に発生した一般例外の処理に移るように、デバッグ例外ハンドラは、Debug レジスタと Cause レジスタを調べ、発生した例外の種類に対応した例外ベクタアドレスを DEPC レジスタに格納します。そして、デバッグ例外ハンドラの終わりで DERET 命令を実行すると、制御を一般ハンドラに移すことができます。

シングルステップ例外は、命令フェッチでのアドレスエラー例外と同時に発生することがありますが、その他の一般例外と同時に発生することはありません。命令フェッチでのアドレスエラー例外を発生した命令は実行されないので、デバッグブレークポイント例外が同時に発生することはありません。

表 9-5に、デバッグ例外ハンドラにより DEPC レジスタにロードする一般例外ベクタアドレスを示します。

表 9-5　一般例外ベクタアドレス

Debug レジスタ		Cause レジスタ			同時に発生した一般例外	例外ベクタ (DEPC に設定する値)
NIS	OES	ExcCode	IL[2:0]	Sw[3:0]		
1	0	x	x	x	ノンマスマブル割り込み	0xBFC0_0000
0	1	≠0	x	x	リセット、 ノンマスカブルまたは マスカブル割り込み以外	0x8000_0080 (BEV=0) 0xBFC0_0180 (BEV=1)
			≥4	x	ハードウエア割り込み	0x8000_0160 (BEV=0) 0xBFC0_0260 (BEV=1)
		=0	1-3	1xxx	ソフトウエア割り込み Swi3	0x8000_0140 (BEV=0) 0xBFC0_0240 (BEV=1)
				0xxx	ハードウエア割り込み	0x8000_0160 (BEV=0) 0xBFC0_0260 (BEV=1)
				1xxx	ソフトウエア割り込み Swi3	0x8000_0140 (BEV=0) 0xBFC0_0240 (BEV=1)
			0	01xx	ソフトウエア割り込み Swi2	0x8000_0130 (BEV=0) 0xBFC0_0230 (BEV=1)
				001x	ソフトウエア割り込み Swi1	0x8000_0120 (BEV=0) 0xBFC0_0220 (BEV=1)
				0001	ソフトウエア割り込み Swi0	0x8000_0110 (BEV=0) 0xBFC0_0210 (BEV=1)

注: x は不定を表します。

9.3.4　例外マスク

デバッグ例外処理中、プロセッサは他の例外をすべてマスクします。

- バスエラー例外が発生すると、Debug レジスタの BsF ビットが 1 にセットされます。
- デバッグ例外が処理されているあいだ、マスカブル割り込みは禁止状態になります (マスカブル割り込みは、DERET 命令の実行でマスクが解除されます)。
- ノンマスカブル割り込みは保留にされ、DERET 命令によるデバッグ例外からの復帰後に発生します。
- デバッグ例外処理中に他の例外が発生したときのプロセッサの動作は未定義です。

9.3.5　デバッグ例外ハンドラの実行

デバッグ例外ハンドラは、プログラムデバッグ用に制御された条件の下で、プロセッサを操作します。Debug レジスタの DSS ビット、DBp ビットにより、シングルステップ例外かデバッグブレークポイント例外かを判断して、対応する処理を行います。

9.3.6　デバッグ例外からの復帰

デバッグ例外ハンドラの処理が終わってから、元のプログラムに復帰するには、DERET 命令を実行します。DERET 命令は、次の処理を実行します。

1. DEPC レジスタの戻りアドレスをプログラムカウンタ (PC) に復元します。これにより、プロセッサは、デバッグ例外が発生したアドレスから処理を再開します。ただし、例外を発生した命令が、ジャンプまたは分岐遅延スロット内の命令であった場合、PC には直前のジャンプまたは分岐命令のアドレスが格納されます。また、PC の ISA モードビットは、DEPC レジスタのビット 0 から復元され、例外が発生したときの ISA モードに戻ります。
2. Debug レジスタの DM (Debug Mode) ビットをクリアします。
3. 強制的な動作モード「カーネルモード、割り込み禁止」が解除され、Status レジスタの KUc ビットと IEc ビットの値が再び有効になります。

9.3.7　シングルステップ例外

■　**原因**

シングルステップ例外は、Debug レジスタの SSt ビットを 1 にセットすると発生します。

■　**処理**

シングルステップ例外は、次の命令を実行する前に実行します。図 9-16に、この例外の処理で使われる CP0 レジスタのフィールドを示します。

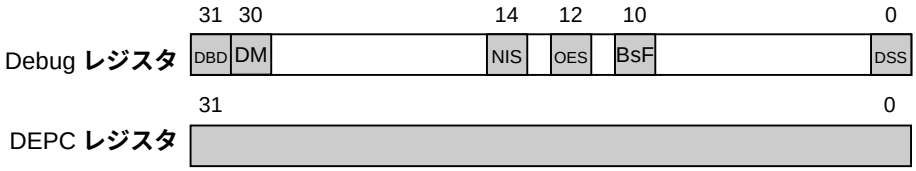


図 9-16　シングルステップ例外

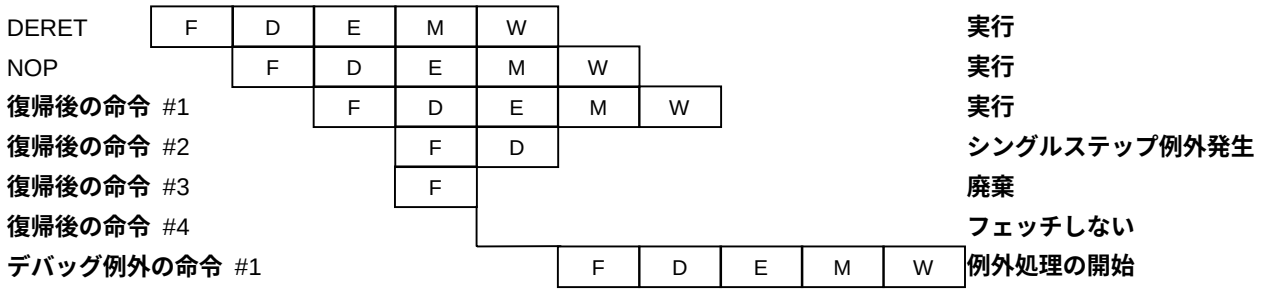
1. Debug レジスタの DM ビットと DSS ビットが 1 にセットされます。一般例外が同時に発生した場合は、NIS ビットまたは OES ビットが 1 にセットされます。シングルステップ例外が発生したということは、SSt ビットが 1 にセットされています。
2. DEPC レジスタに、例外発生時のプログラムカウンタ (PC) の内容が保存されます。また、DEPC レジスタの最下位ビットは、例外発生時の ISA モードが保存されます。
3. Status レジスタの設定に関係なく、プロセッサはカーネルモードに切り換えられ、

割り込みがすべて禁止されます。

4. 例外ベクタアドレス 0xBFC0_0200 にジャンプし、例外ハンドラに制御が移ります。

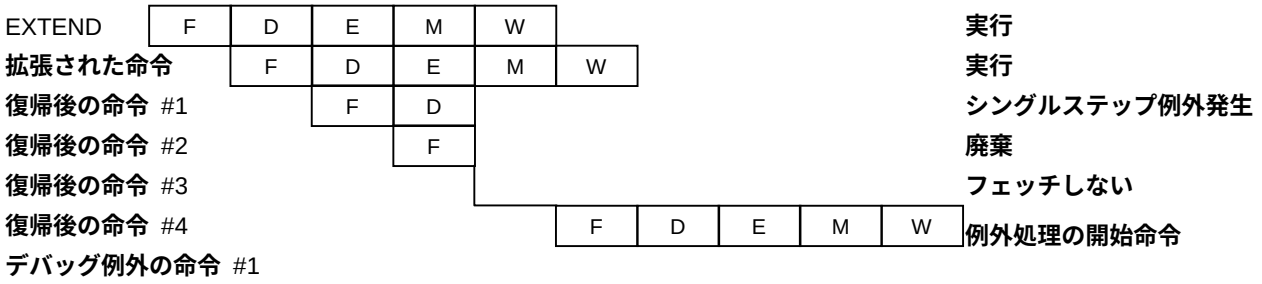
シングルステップ例外は以下の場合、発生しません。

- ジャンプまたは分岐遅延スロット内の命令
- デバッグ命令から、DERET 命令で復帰した最初の命令 (図 9-17参照)
- デバッグ例外処理中 (Debug レジスタの DM ビットが 1 にセットされているとき)
- EXTEND 命令の直後の命令 (図 9-18参照)



DEPC レジスタは、復帰後の命令#2 のアドレスになります。

図 9-17　DERET 命令後の CPU パイプライン動作



DEPC レジスタは、復帰後の命令#1 のアドレスになります。

図 9-18　拡張された命令後の CPU パイプライン動作

9.3.8　デバッグブレークポイント例外

■　**原因**

デバッグポイント例外は、SDBBP 命令を実行すると発生します。

■　**処理**

図 9-19に、この例外の処理で使われる CP0 レジスタのフィールドを示します。

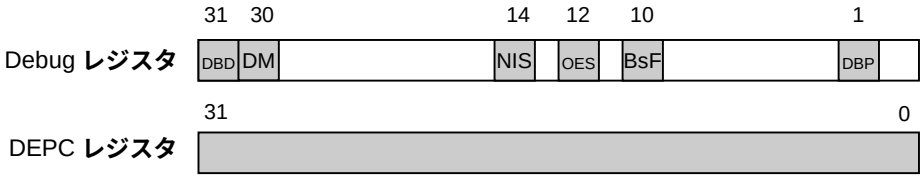


図 9-19　**デバッグブレークポイント例外**

1. Debug レジスタの DM ビットと DBP ビットが 1 にセットされます。一般例外が同時に発生した場合、NIS ビットまたは OES ビットが 1 にセットされます。デバッグブレークポイント例外が発生したということは、SSt ビットは 0 にクリアされています。

2. DEPC レジスタに、割り込みを発生した時点のプログラムカウンタ (PC) の内容を保存します。ただし、割り込みを発生した命令が、ジャンプまたは分岐遅延スロット内の命令である場合は、DEPC レジスタには、直前のジャンプまたは分岐命令のアドレスが格納されます。この場合、Debug レジスタの DBD ビットが 1 にセットされます。また、DEPC レジスタの最下位ビットには、例外が発生したときの ISA モードが保存されます。

3. Status レジスタの設定に関係なく、プロセッサはカーネルモードに切り換えられ、割り込みがすべて禁止されます。

4. 16 ビット ISA モードで例外が発生すると、32 ビット ISA モードに切り換えられます。

5. 例外ベクタアドレス 0xBFC0_0200 にジャンプし、例外ハンドラに制御が移ります。

SDBBP 命令の未使用ビット (32 ビット ISA の場合はビット 25~6、16 ビット ISA の場合はビット 10~5) を使って、例外ハンドラに情報を渡すことができます。これらのビットを調べるには、DEPC レジスタが指している命令をデータとしてロードします。ただし、例外を発生した命令が、ジャンプまたは分岐遅延スロット内にある (Debug レジスタの DBD ビットが 1 にセットされている) 場合は、DEPC レジスタの値に 4 を加えなければなりません。

例外ハンドラから戻る際に、SDBBP 命令が再び実行されないように、DEPC レジ

スタ内のアドレスに 4 (32 ビット ISA) または 2 (16 ビット ISA) を加えなければなりません。SDBBP 命令が、ジャンプまたは分岐遅延スロット内にある (Debug レジスタの DBD ビットが 1 にセットされている) 場合は、戻りアドレスの命令は、ジャンプまたは分岐命令になります。この場合、ジャンプまたは分岐命令を解釈し、DEPC レジスタに設定し直さなければなりません。

第10章 低消費電力モード

TX19 には、いくつかの低消費電力モードがあります。Halt モード、Doze モードへは、CP0 レジスタを設定することにより移行でき、また RF (Reduced Frequency) モードは、レジスタの設定とクロックジェネレータが協調して動作するモードです。この章では、TX19 の低消費電力モードについて説明します。

10.1 各低消費電力モードの特徴

図 10-1に TX19 の低消費電力モードを示します。

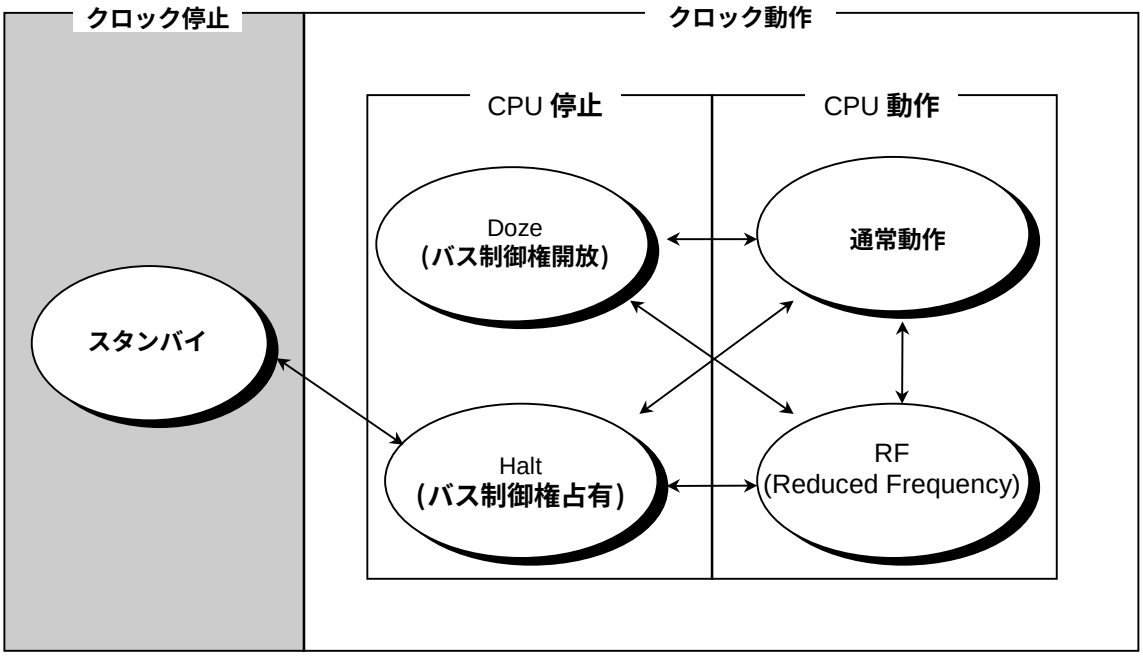


図 10-1　低消費電力モード

TX19 には、動作中消費電力をダイナミックに管理するモードとして、表 10-1に示すモードがあります。

表 10-1　低消費電力モード

モード	説　明
スタンバイモード	消費電力を最小限にするために、プロセッサへのクロック供給を停止できます。スタンバイモードには、2 つのレベルのモードがあります。 1.　プロセッサと発振回路の両方を停止します。 2.　発振回路は継続して動作しますが、プロセッサへのクロック供給を停止します。 スタンバイモードの詳細については、各製品のマニュアルを参照してください。
Halt モード	Halt モードでは、プロセッサの動作はすべて停止し、外部からのバス制御権要求は受け付けません。TX19 は、バス制御権を占有した状態となります。Config レジスタを設定することにより、Halt モードに移行できます。
Doze モード	Doze モードでは、プロセッサの動作は停止しますが、外部からのバス制御権の要求は受け付け、バス制御権は開放した状態となります。Config レジスタを設定することにより、Doze モードに移行できます。
RF (Reduced Frequency) モード	消費電力を低減するために、プロセッサのクロックを fc/2、fc/4、fc/8 の周波数 (fc＝原振) で動作させるモードです。Config レジスタを設定することにより、RF モードに移行できます。
通常動作モード	TX19 のデフォルトの動作モードで、プロセッサが最大のクロック動作周波数で動作しているモードです。
その他のモード	例えば、時計用水晶 32.768 kHz で動作させる超低速モード、上記以外の低消費電力モードをもっている製品があります。各製品のマニュアルを参照してください。

10.2 Halt モード

図 10-2に、Halt モードに移行する経路を示します。

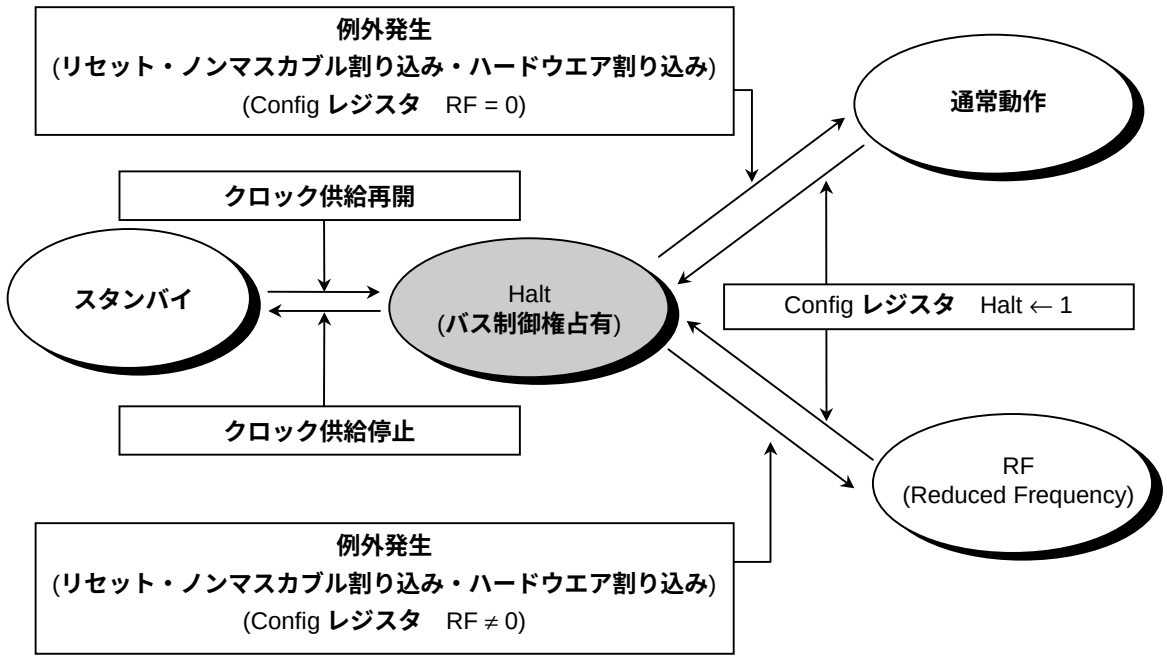


図 10-2　Halt **モード**

Halt モードでは、TX19 プロセッサコアは、パイプラインの状態を保持したままプロセッサ動作を停止します。Halt モードでは、プロセッサは外部からのバス要求信号を無視し、バス制御権を占有したままの状態になります。

Halt モード中でも、ライトバッファユニットのデータを外部メモリに書き終わるまで、ライトバッファユニットは動作しつづけます。

通常動作モードまたは RF モードのときに、Config レジスタの Halt ビットを 1 に設定すると、Halt モードに移行します。また、リセット例外、ノンマスカブル割り込み例外、またはマスカブルハードウエア割り込み例外が発生すると、Halt モードが解除されて、例外処理を実行します。

Status レジスタでマスカブル割り込みがマスクされている状態でも、マスカブル割り込みは認識されます。この場合、Halt モードに移行する前の状態から通常の処理が再開されます。

Halt モードの状態でスタンバイモードに移行することにより、クロックの供給を停止することができます。オシレータの発振停止、クロックの供給停止により、プロセッサはスタンバイモードに移行します。クロックが再開されると、スタンバイモードから Halt モードに復帰します。

10.3 Doze モード

図 10-3に、Doze モードに移行する経路を示します。

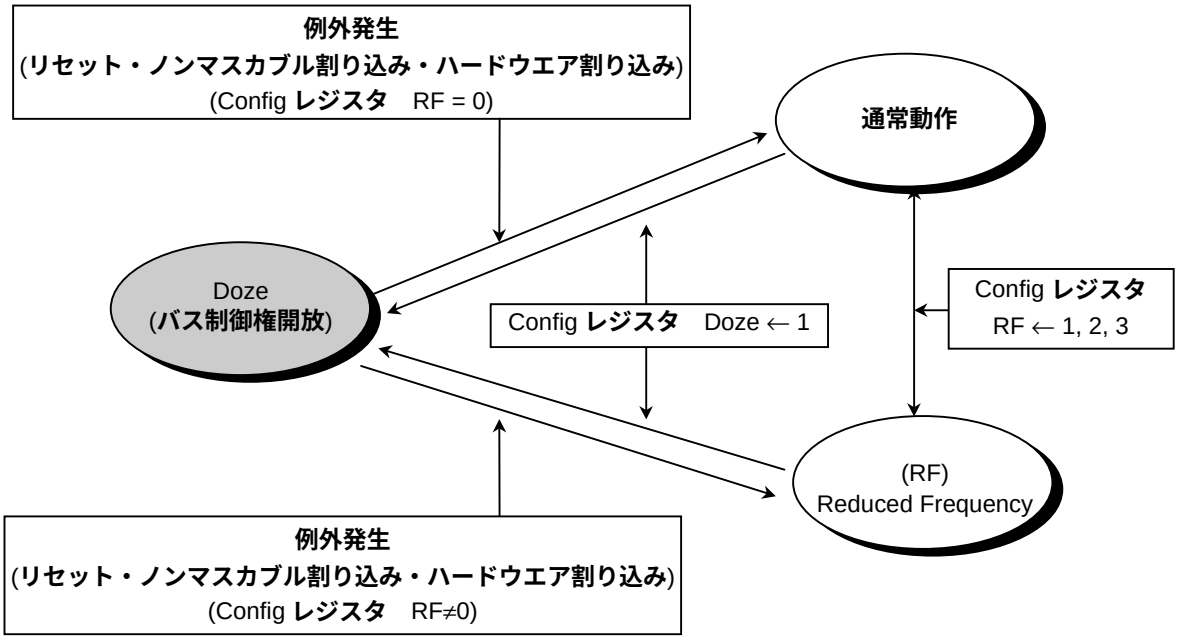


図 10-3　Doze モード

Halt モードと同様、Doze モードでは、TX19 プロセッサコアは、パイプラインの状態を保持したままプロセッサ動作を停止します。ただし、Doze モードでは、プロセッサは外部からのバス要求権を認識します。

Doze モード中でも、ライトバッファユニットのデータを外部メモリに書き終わるまで、ライトバッファユニットは動作しつづけます。

通常動作モードまたは RF モードのときに、Config レジスタの Doze ビットを 1 に設定すると、Doze モードに移行します。リセット例外、ノンマスカブル割り込み例外、またはハードウエアマスカブル割り込み例外が発生すると、Doze モードが解除され、例外処理を実行します。

Status レジスタでマスカブル割り込みがマスクされている状態でも、マスカブル割り込みは認識されます。この場合、Doze モードに移行する前の状態から通常の処理が再開されます。

10.4 プロセッサクロックの分周

RF (Reduced frequency) モードは、消費電力を低減するために、プロセッサクロックを fc/2、fc/4、fc/8 の周波数 (fc = 原振) で動作させるモードです。周波数は、Config レジスタの RF[1:0]フィールドで設定された 2 の累乗で分周されます。Config レジスタの RF[1:0] フィールドの値は、プロセッサコアから出力されて、内蔵のクロックジェネレータに入力されます。RF[1:0]ビットを 0 にセットすると、通常の周波数に戻ります。

プロセッサが RF モードのとき、Config レジスタの Halt ビットまたは Doze ビットを 1 にセットすると、Halt モードまたは Doze モードに移行します。また、リセット例外、ノンマスカブル割り込み例外、または通常の割り込み例外が発生すると RF モードは解除されます。

付録A　32 ビット ISA の詳細

この章では 32 ビット ISA モードの命令について、シンタックス、命令形式、動作、命令の実行によって発生する可能性のある例外などを詳しく説明しています。命令はアルファベット順に記述されています。命令形式については「3.1　命令形式」を参照してください。

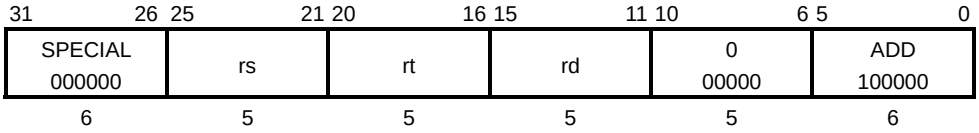
ADD *rd*, *rs*, *rt*

Add

動作

rd ⇐ *rs* + *rt*

コード



説明

汎用レジスタ *rs* の内容と汎用レジスタ *rt* の内容を加算し、その結果を汎用レジスタ *rd* に格納します。

c ⇐ a + b の計算において、a と b の符号が同一で、かつそれら符号と c の符号が異なる場合、計算結果がオーバフローしています。この場合、整数オーバフロー例外が発生します。整数オーバフロー例外が発生するとデスティネーションレジスタ (*rd*) の内容は変更されません。

例外

整数オーバフロー例外

使用例

1. レジスタ r2 の値が 0x0200_0000 で、レジスタ r3 の値が 0x0123_4567 の場合、以下の命令を実行すると、レジスタ r4 に和 (0x0323_4567) が格納されます。

```
ADD r4,r2,r3
```

2. レジスタ r2 の値が 0x7FFF_FFFF で、レジスタ r3 の値が 0x0000_0001 の場合、r2 と r3 を加算すると演算結果は、0x8000_0000 と負の値になり、2 の補数のオーバフローが発生します。この場合、以下の命令を実行すると、整数オーバフロー例外が発生し、レジスタ r4 の内容は変更されません。

```
ADD r4,r2,r3
```

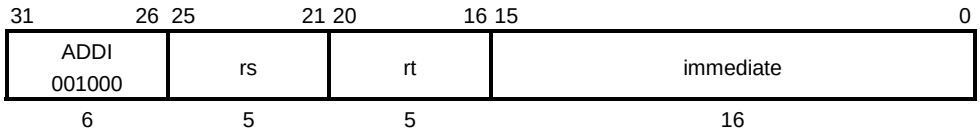
ADDI *rt, rs, immediate*

Add Immediate

動作

rt ⇐ *rs* + *immediate*

コード



説明

16 ビット *immediate* を符号拡張して、汎用レジスタ *rs* の内容と加算し、その結果を汎用レジスタ *rt* に格納します。

2 の補数オーバフローで整数オーバフロー例外が発生します。整数オーバフロー例外が発生すると、デスティネーションレジスタ (*rt*) の内容は変更されません。

immediate フィールドは 16 ビットです。したがって、*immediate* で指定できる数値の範囲は、–32768 ~ +32767 です。この範囲外の値を扱いたい場合は、いったん汎用レジスタに格納してから、ADD または ADDU 命令を使います (「3.3.2　32 ビットの定数」を参照)。

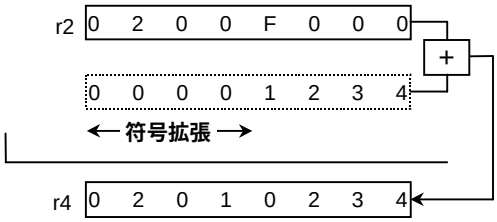
例外

整数オーバフロー例外

使用例

レジスタ r2 の値が 0x0200_F000 の場合、以下の命令を実行すると、和 (0x0201_0234) がレジスタ r3 に格納されます。

```
ADDI r3,r2,0x1234
```



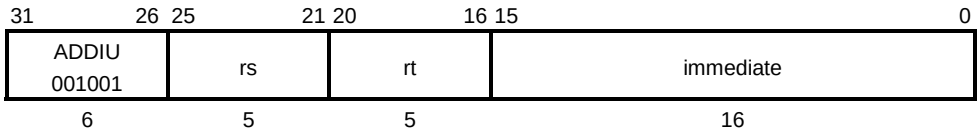
ADDIU *rt, rs, immediate*

Add Immediate Unsigned

動作

rt ⇐ *rs* + *immediate*

コード



説明

ADDIU は Add Immediate Unsigned を表しますが、16 ビット *immediate* を「符号拡張」して、汎用レジスタ *rs* の内容に加算し、その結果を汎用レジスタ *rt* に格納します。

ADDI 命令と ADDIU 命令の唯一の違いは、ADDUI 命令では整数オーバフロー例外が絶対に発生しないという点だけです。

例外

なし

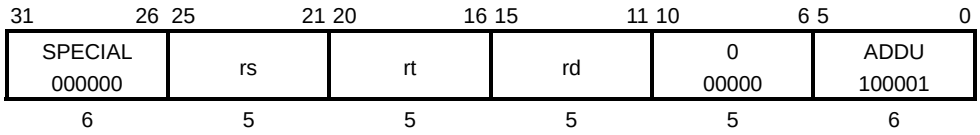
ADDU *rd*, *rs*, *rt*

Add Unsigned

動作

$rd \Leftarrow rs + rt$

コード



説明

汎用レジスタ *rs* の内容と汎用レジスタ *rt* の内容を加算し、その結果を汎用レジスタ *rd* に格納します。

ADDU 命令と ADD 命令の唯一の違いは、ADDU 命令では整数オーバフロー例外が絶対に発生しないという点だけです。

例外

なし

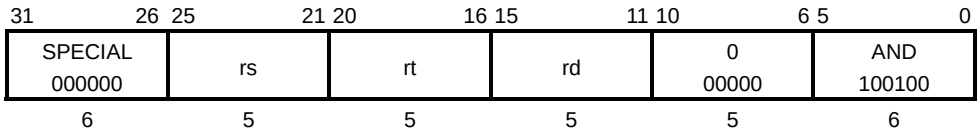
AND *rd*, *rs*, *rt*

AND

動作

rd ⇐ *rs* AND *rt*

コード



説明

汎用レジスタ *rs* の内容と汎用レジスタ *rt* の内容の論理積演算を行い、その結果を汎用レジスタ *rd* に格納します。

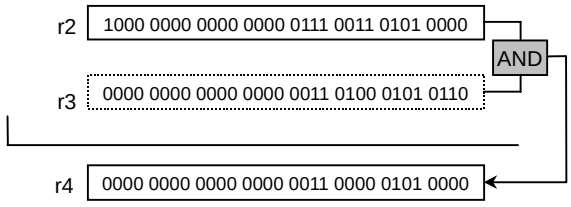
例外

なし

使用例

レジスタ r2 の値が 0x8000_7350 で、レジスタ r3 の値が 0x0000_3456 の場合、以下の命令を実行すると、図示したように、r2 と r3 の論理積 (0x0000_3050) がレジスタ r4 に格納されます。

```
AND r4,r2,r3
```



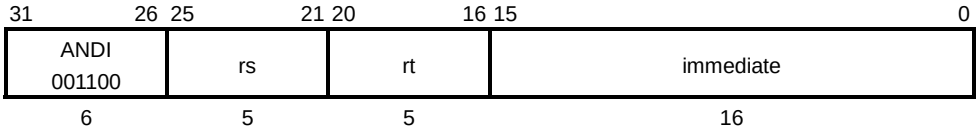
ANDI *rt*, *rs*, *immediate*

Logical AND Immediate

動作

rt ⇐ *rs* AND *immediate*

コード



説明

16 ビット *immediate* をゼロ拡張し、汎用レジスタ *rs* の内容と論理積演算を行い、その結果を汎用レジスタ *rt* に格納します。

immediate フィールドは 16 ビットです。これを超える値を扱いたい場合は、いったん汎用レジスタに格納してから、AND 命令を使います (「3.3.2　32 ビットの定数」を参照)。

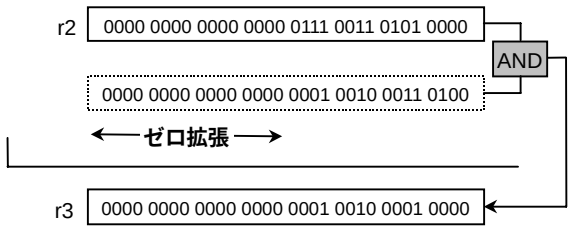
例外

なし

使用例

レジスタ r2 の値が 0x0000_7350 の場合、以下の命令を実行すると、図示したように、0x0000_7350 と 0x0000_1234 の論理積 (0x0000_1210) がレジスタ r3 に格納されます。

```
ANDI r3,r2,0x1234
```



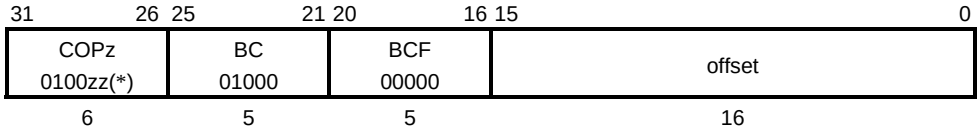
BC*z*F *offset*

Branch On Coprocessor z False

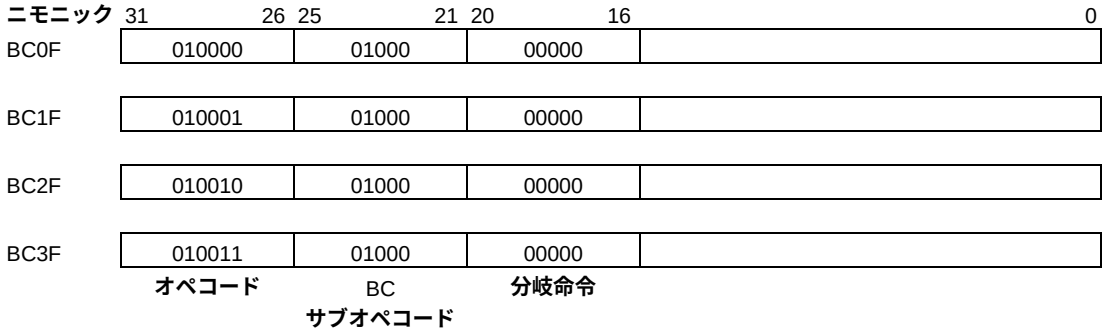
動作

if コプロセッサ *z* の条件信号が偽
　then pc ⇐ pc + *offset*

コード



オペコードのビットコードを以下に示します。オペコードフィールドの下位 2 ビットはコプロセッサユニット番号を表します。



説明

BC*z*F 命令の直前の命令を実行中にサンプリングされたコプロセッサ *z* の条件信号 (CPCOND) が偽の場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスへ分岐します。ターゲットアドレスは、分岐遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。コプロセッサユニット *z* の条件信号 (CPCOND) が真の場合は、分岐しません。

例外

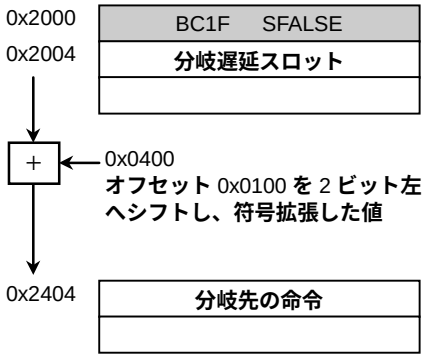
コプロセッサ使用不可例外

使用例

```
BC1F SFALSE
```

上記の分岐命令がアドレス 0x2000 に置かれていて、ラベル SFALSE が 0x2404 に絶対アドレス化される場合、以下に図示するように、SFALSE はアセンブラ・リンカによりオフセット 0x0100 に変換されます。

コプロセッサ 1 の条件信号 (CPCOND) が偽の場合、上記の命令を実行すると、プログラムの処理は、ターゲットアドレス 0x2404 に分岐します。この場合、遅延スロット内の命令は、分岐のまえに実行されます。



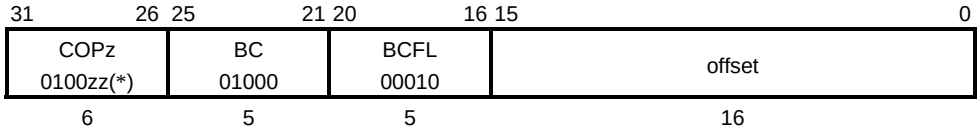
BC*z*FL *offset*

Branch On Coprocessor z False Likely

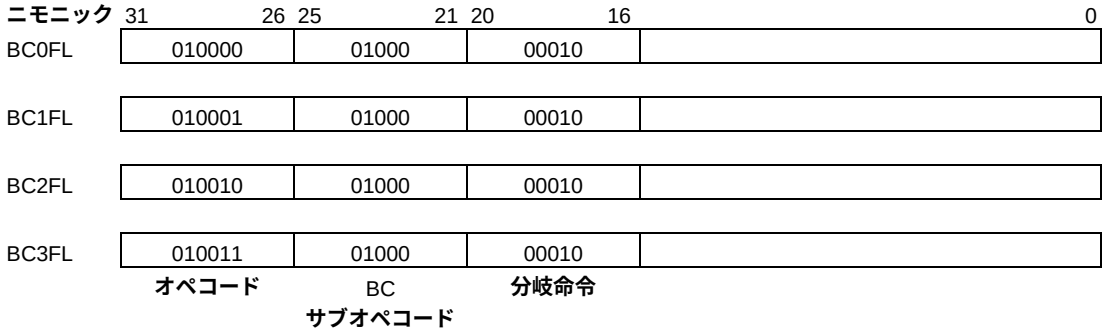
動作

if コプロセッサ *z* の条件信号が偽
　then pc ⇐ pc + *offset*

コード



オペコードのビットコードを以下に示します。オペコードフィールドの下位 2 ビットはコプロセッサユニット番号を表します。



説明

BC*z*FL 命令の直前の命令を実行中にサンプリングされたコプロセッサ *z* の条件信号 (CPCOND) が偽の場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスへ分岐します。ターゲットアドレスは、分岐遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。コプロセッサユニット *z* の条件信号 (CPCOND) が真の場合は、分岐遅延スロット内の命令は無効になります。

例外

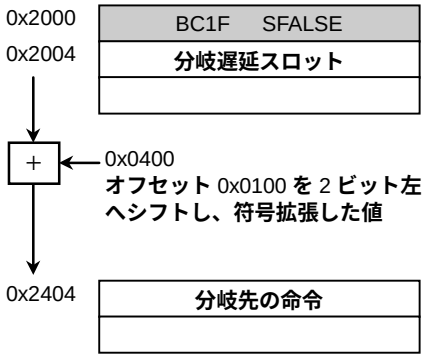
コプロセッサ使用不可例外

使用例

```
BC1FL SFALSE
```

上記の分岐命令がアドレス 0x2000 に置かれていて、ラベル SFALSE が 0x2404 に絶対アドレス化される場合、以下に図示するように、SFALSE はアセンブラ・リンカによりオフセット 0x0100 に変換されます。

コプロセッサ 1 の条件信号 (CPCOND) が偽の場合、上記の命令を実行すると、プログラムの処理はターゲットアドレス 0x2404 に分岐します。この場合、遅延スロット内の命令は、分岐のまえに実行されます。分岐条件が成立しないときは、分岐遅延スロット内の命令は、無効になります。



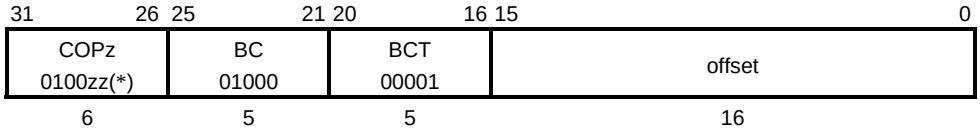
BC*z*T *offset*

Branch On Coprocessor z True

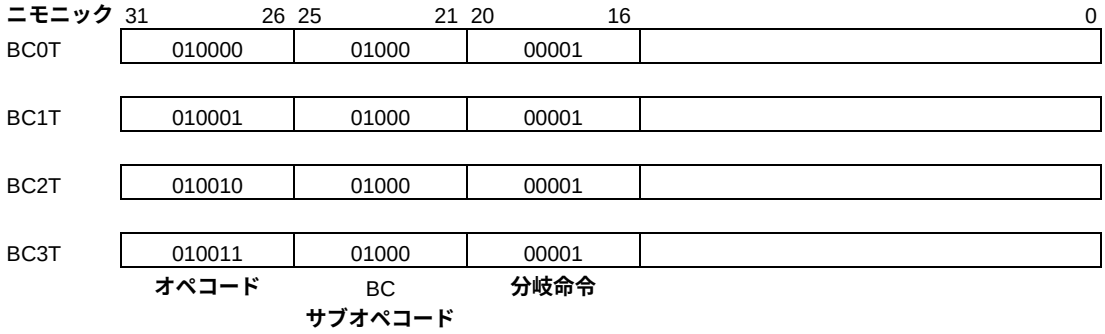
動作

if コプロセッサ *z* の条件信号が真
　then pc ⇐ pc + *offset*

コード



オペコードのビットコードを以下に示します。オペコードフィールドの下位 2 ビットはコプロセッサユニット番号を表します。



説明

BC*z*T 命令の直前の命令を実行中にサンプリングされたコプロセッサ *z* の条件信号 (CPCOND) が真の場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスへ分岐します。ターゲットアドレスは、分岐遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。コプロセッサユニット *z* の条件信号 (CPCOND) が偽の場合は、分岐しません。

例外

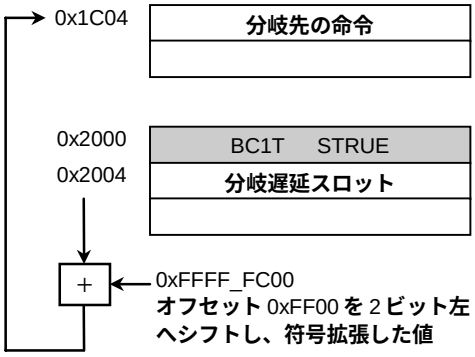
コプロセッサ使用不可例外

使用例

```
BC1T STRUE
```

上記の分岐命令がアドレス 0x2000 に置かれていて、ラベル STRUE が 0x1C04 に絶対アドレス化される場合、以下に図示するように、STRUE はアセンブラ・リンカによりオフセット 0xFF00 に変換されます。

コプロセッサ 1 の条件信号 (CPCOND) が真の場合、上記の命令を実行すると、プログラムの処理は、ターゲットアドレス 0x1C04 に分岐します。この場合、遅延スロット内の命令は、分岐のまえに実行されます。



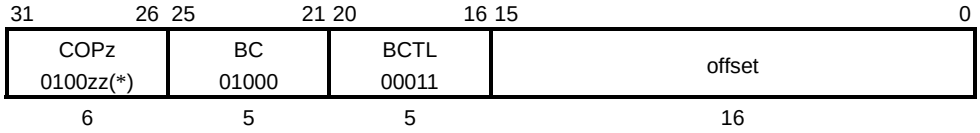
BC*z*TL *offset*

Branch On Coprocessor z True Likely

動作

if コプロセッサ *z* の条件信号が真
　then pc ⇐ pc + *offset*

コード



オペコードのビットコードを以下に示します。オペコードフィールドの下位 2 ビットはコプロセッサユニット番号を表します。

説明

BC*z*TL 命令の直前の命令を実行中にサンプリングされたコプロセッサ *z* の条件信号 (CPCOND) が真の場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスへ分岐します。ターゲットアドレスは、分岐遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。コプロセッサユニット *z* の条件信号 (CPCOND) が偽の場合は、遅延スロット内の命令は無効になります。

例外

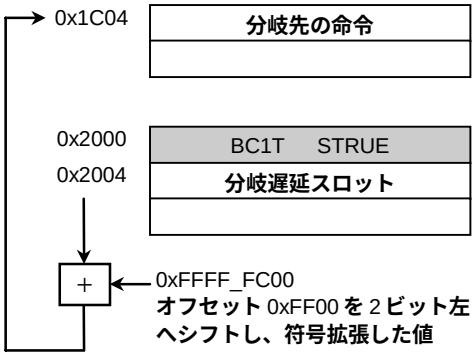
コプロセッサ使用不可例外

使用例

```
BC1TL STRUE
```

上記の分岐命令がアドレス 0x2000 に置かれていて、ラベル STRUE が 0x1C04 に絶対アドレス化される場合、以下に図示するように STURE はアセンブラ・リンカによりオフセット 0xFF00 に変換されます。

コプロセッサ 1 の条件信号 (CPCOND) が真の場合、上記の命令を実行すると、プログラムの処理は、ターゲットアドレス 0x1C04 へ分岐します。この場合、遅延スロット内の命令は、分岐のまえに実行されます。分岐条件が成立しないときは、分岐遅延スロット内の命令は、無効になります。



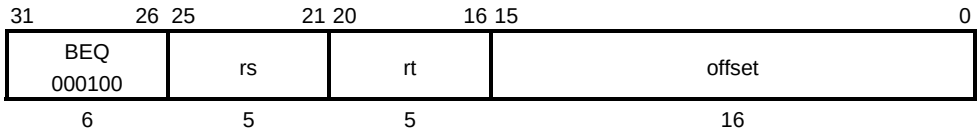
BEQ *rs*, *rt*, *offset*

Branch On Equal

動作

if *rs* = *rt* then pc ⇐ pc + *offset*

コード



説明

汎用レジスタ *rs* の内容と汎用レジスタ *rt* の内容を比較し、両者が等しい場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐します。ターゲットアドレスは、分岐遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。

例外

なし

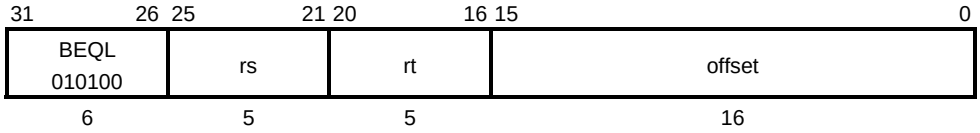
BEQL *rs*, *rt*, *offset*

Branch On Equal Likely

動作

if *rs* = *rt* then pc ⇐ pc + *offset*

コード



説明

汎用レジスタ *rs* の内容と汎用レジスタ *rt* の内容を比較し、両者が等しい場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐します。分岐が成立しない場合は、分岐遅延スロット内の命令は無効になります。ターゲットアドレスは、分岐遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。

例外

なし

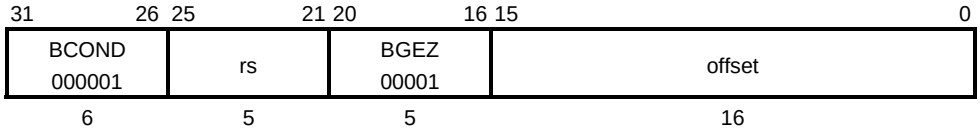
BGEZ *rs*, *offset*

Branch On Greater Than Or Equal To Zero

動作

if *rs* ≥ 0 then pc ⇐ pc + *offset*

コード



説明

汎用レジスタ *rs* の内容が 0 以上の場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐します。ターゲットアドレスは、分岐遅延スロット内の命令アドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。

例外

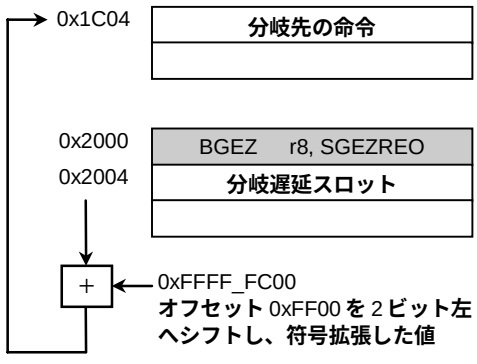
なし

使用例

```
BGEZ r8,SGEZERO
```

上記の分岐命令がアドレス 0x2000 に置かれていて、ラベル SGEZERO が 0x1C04 に絶対アドレス化される場合、以下に図示するように、SGEZERO はアセンブラ・リンカによりオフセット 0xFF00 に変換されます。

レジスタ r8 の内容が 0 以上のとき (符号ビットが 0 であるとき)、プログラムの処理はアドレス 0x1C04 に分岐します。この場合、遅延スロット内の命令は、分岐のまえに実行されます。



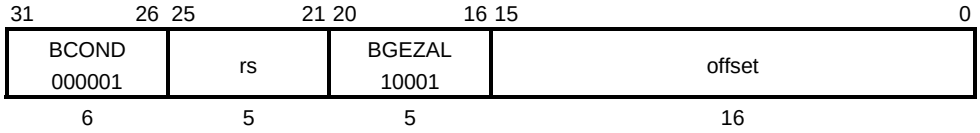
BGEZAL *rs*, *offset*

Branch On Greater Than or Equal To Zero And Link

動作

r31 ⇐ pc +8; if *rs* ≥ 0 then pc ⇐ pc + *offset*

コード



説明

汎用レジスタ *rs* の内容が 0 以上の場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐し、分岐遅延スロットの後続命令のアドレス (PC+8) を、無条件にリンクレジスタ r31 に格納します。ターゲットアドレスは、分岐遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。

汎用レジスタ *rs* に r31 を指定できません。*rs* として r31 を指定すると、戻りアドレスによって rs の内容が破壊されてしまいます。すると、例外や割り込みにより、分岐遅延スロット内の命令が完了しなかった場合、例外処理後、分岐命令から実行を再開できなくなってしまうからです。

例外

なし

使用例

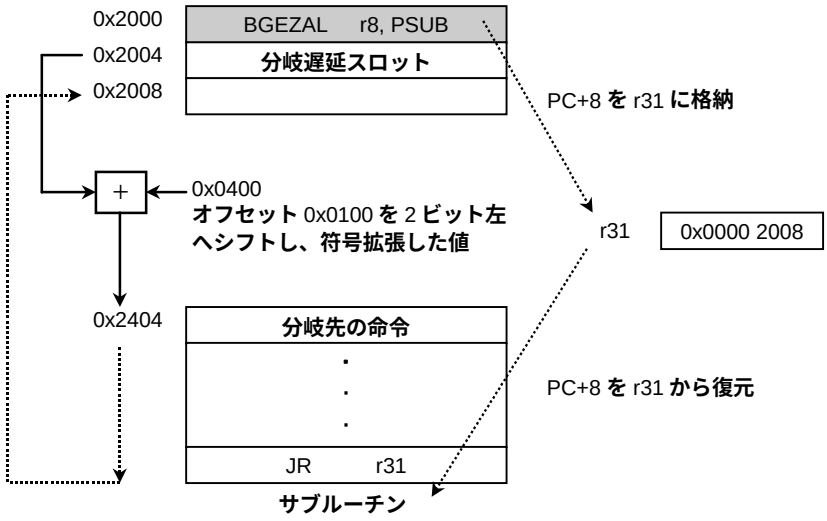
```
BGEZAL r8,PSUB
```

上記の分岐命令がアドレス 0x2000 に置かれていて、ラベル PSUB が 0x2404 に絶対アドレス化される場合、以下に図示するように、PSUB はアセンブラ・リンカによりオフセット 0x0100 に変換されます。

レジスタ r8 の内容が 0 以上のとき (符号ビットが 0 であるとき)、プログラムの処理はアドレス 0x2404 に分岐します。この場合、分岐遅延スロット内の命令は、分岐のまえに実行されます。

コールされたサブルーチンの終わりで JR 命令を実行することにより、分岐遅延スロットの直後の命令 (PC+8) に戻ることができます。

```
JR r31
```


0x2000
BGEZAL　r8, PSUB
0x2004
分岐遅延スロット
0x2008
PC+8 を r31 に格納
+
0x0400
オフセット 0x0100 を 2 ビット左
へシフトし、符号拡張した値
r31
0x0000 2008
0x2404
分岐先の命令
PC+8 を r31 から復元
JR　r31
サブルーチン

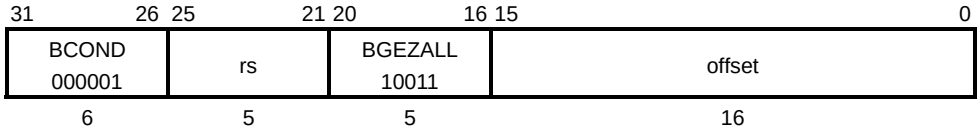
BGEZALL *rs*, *offset*

Branch On Greater Than Or Equal To Zero And Link Likely

動作

r31 ⇐ pc +8; if *rs* ≥ 0 then pc ⇐ pc + *offset*

コード



説明

汎用レジスタ *rs* の内容が 0 以上の場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐し、分岐遅延スロットの後続命令のアドレス (PC + 8) を、無条件にリンクレジスタ r31 に格納します。分岐しない場合は、分岐遅延スロット内の命令は無効になります。ターゲットアドレスは、分岐遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。

汎用レジスタ *rs* に r31 を指定できません。*rs* として r31 を指定すると、戻りアドレスによって rs の内容が破壊されてしまいます。すると、例外や割り込みにより、分岐遅延スロット内の命令が完了しなかった場合、例外処理後、分岐命令から実行を再開できなくなってしまうからです。

例外

なし

使用例

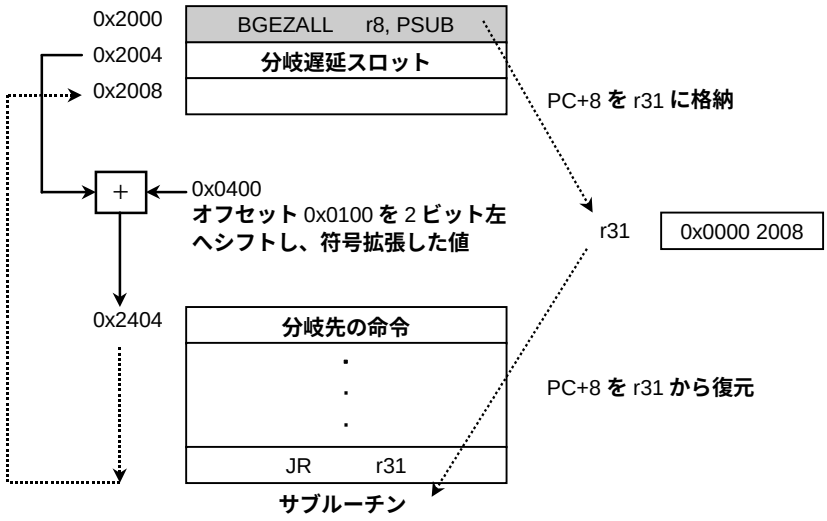
```
BGEZALL r8,PSUB
```

上記の分岐命令がアドレス 0x2000 に置かれていて、ラベル PSUB が 0x2404 に絶対アドレス化される場合、以下に図示するように PSUB はアセンブラ・リンカによりオフセット 0x0100 に変換されます。

レジスタ r8 の内容が 0 以上のとき (符号ビットが 0 であるとき)、プログラムの処理はアドレス 0x2404 に分岐します。この場合、分岐遅延スロット内の命令は、分岐のまえに実行されます。分岐条件が成立しない場合は、分岐遅延スロット内の命令は無効になります。

コールされたサブルーチンの終わりで JR 命令を実行することにより、分岐遅延スロットの直後の命令 (PC+8) に戻ることができます。

```
JR r31
```


0x2000
0x2004
0x2008
BGEZALL r8, PSUB
分岐遅延スロット
PC+8 を r31 に格納
+
0x0400
オフセット 0x0100 を 2 ビット左
へシフトし、符号拡張した値
r31
0x0000 2008
0x2404
分岐先の命令
PC+8 を r31 から復元
JR r31
サブルーチン

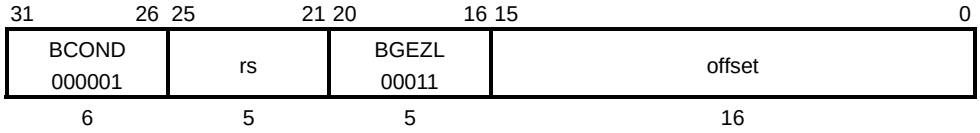
BGEZL *rs*, *offset*

Branch On Greater Than Or Equal To Zero Likely

動作

if *rs* ≥ 0 then pc ⇐ pc + *offset*

コード



説明

汎用レジスタ *rs* の内容が 0 以上の場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスへ分岐します。分岐しない場合は、分岐遅延スロットの命令は無効になります。ターゲットアドレスは、分岐遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。

例外

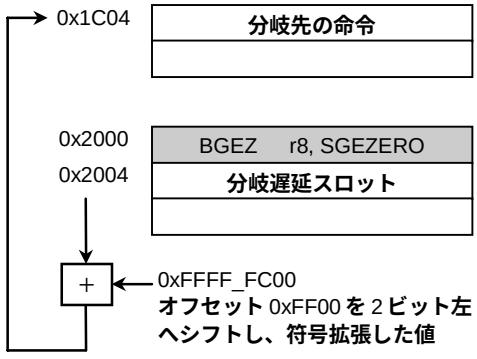
なし

使用例

```
BGEZL r8,SGEZERO
```

上記の分岐命令がアドレス 0x2000 に置かれていて、ラベル SGEZERO が 0x1C04 に絶対アドレス化される場合、以下に図示するように、SGEZERO はアセンブラ・リンカによりオフセット 0xFF00 に変換されます。

レジスタ r8 の内容が 0 以上のとき (符号ビットが 0 であるとき)、プログラムの処理はアドレス 0x1C04 に分岐します。この場合、分岐遅延スロット内の命令は、分岐のまえに実行されます。分岐条件が成立しない場合は、分岐遅延スロット内の命令は無効になります。



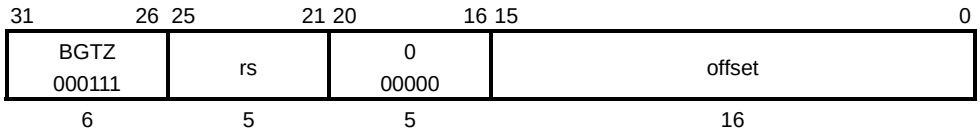
BGTZ *rs*, *offset*

Branch On Greater Than Zero

動作

if *rs* > 0 then pc ⇐ pc + *offset*

コード



説明

汎用レジスタ *rs* の内容が 0 より大きい場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐します。ターゲットアドレスは分岐遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。

例外

なし

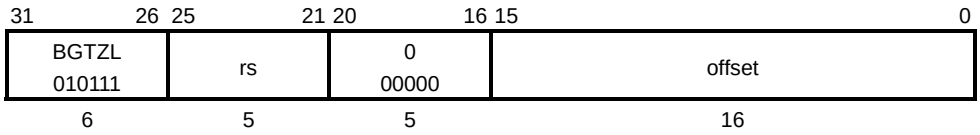
BGTZL *rs*, *offset*

Branch On Greater Than Zero Likely

動作

if *rs* > 0 then pc ⇐ pc + *offset*

コード



説明

汎用レジスタ *rs* の内容が 0 より大きい場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐します。分岐しない場合は、分岐遅延スロット内の命令は無効になります。ターゲットアドレスは分岐遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。

例外

なし

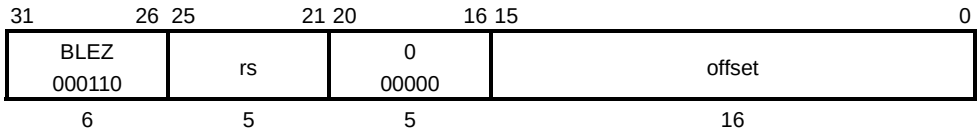
BLEZ *rs*, *offset*

Branch On Less Than Or Equal To Zero

動作

if *rs* ≤ 0 then pc ⇐ pc + *offset*

コード



説明

汎用レジスタ *rs* の内容が 0 以下の場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐します。ターゲットアドレスは分岐遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。

例外

なし

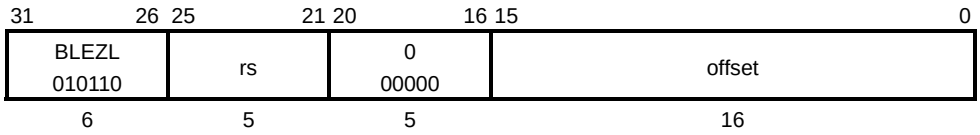
BLEZL *rs*, *offset*

Branch On Less Than Or Equal To Zero Likely

動作

if *rs* ≤ 0 then pc ⇐ pc + *offset*

コード



説明

汎用レジスタ *rs* の内容が 0 以下の場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐します。分岐しない場合は、分岐遅延スロット内の命令は無効になります。ターゲットアドレスは分岐遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。

例外

なし

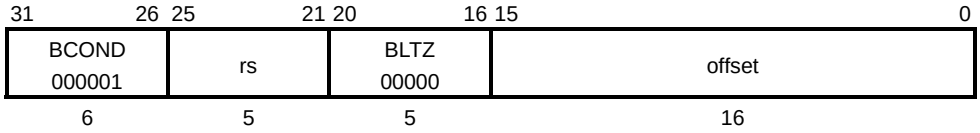
BLTZ *rs*, *offset*

Branch On Less Than Zero

動作

if *rs* < 0 then pc ⇐ pc + *offset*

コード



説明

汎用レジスタ *rs* の内容が 0 より小さい場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐します。ターゲットアドレスは分岐遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。

例外

なし

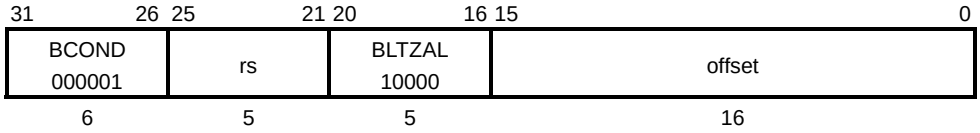
BLTZAL *rs*, *offset*

Branch On Less Than Zero And Link

動作

r31 ⇐ pc +8; if *rs* < 0 then pc ⇐ pc + *offset*

コード



説明

汎用レジスタ *rs* の内容が 0 より小さい場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐し、分岐遅延スロットの後続命令のアドレス (PC+8) を、無条件にリンクレジスタ r31 に格納します。ターゲットアドレスは分岐遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。

汎用レジスタ *rs* に r31 を指定できません。*rs* として r31 を指定すると、戻りアドレスによって rs の内容が破壊されてしまいます。すると、例外や割り込みにより、分岐遅延スロット内の命令が完了しなかった場合、例外処理後、分岐命令から実行を再開できなくなってしまうからです。

例外

なし

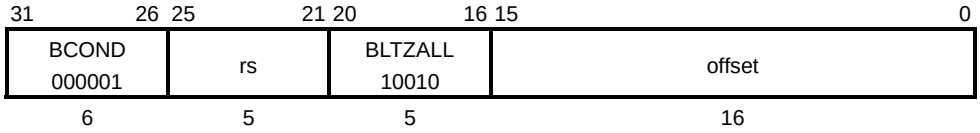
BLTZALL *rs*, *offset*

Branch On Less Than Zero And Link Likely

動作

r31 ⇐ pc +8; if *rs* < 0 then pc ⇐ pc + *offset*

コード



説明

汎用レジスタ *rs* の内容が 0 より小さい場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐し、分岐遅延スロットの後続命令のアドレス (PC+8) を、無条件にリンクレジスタ r31 に格納します。分岐しない場合は、遅延スロット内の命令は無効になります。ターゲットアドレスは分岐遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算さます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。

汎用レジスタ *rs* に r31 を指定できません。*rs* として r31 を指定すると、戻りアドレスによって rs の内容が破壊されてしまいます。すると、例外や割り込みにより、分岐遅延スロット内の命令が完了しなかった場合、例外処理後、分岐命令から実行を再開できなくなってしまうからです。

例外

なし

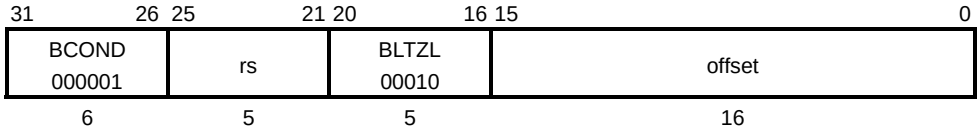
BLTZL *rs*, *offset*

Branch On Less Than Zero Likely

動作

if *rs* < 0 then pc ⇐ pc + *offset*

コード



説明

汎用レジスタ *rs* の内容が 0 より小さい場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐します。分岐しない場合は、分岐遅延スロットの命令は無効になります。ターゲットアドレスは分岐遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。

例外

なし

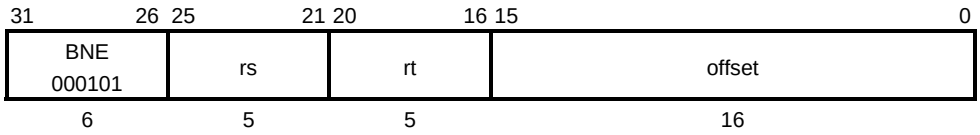
BNE *rs*, *rt*, *offset*

Branch On Not Equal

動作

if *rs* ≠ *rt* then pc ⇐ pc + *offset*

コード



説明

汎用レジスタ *rs* の内容と汎用レジスタ *rt* の内容を比較し、両者が等しくない場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐します。ターゲットアドレスは、分岐遅延スロット内の命令アドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。

例外

なし

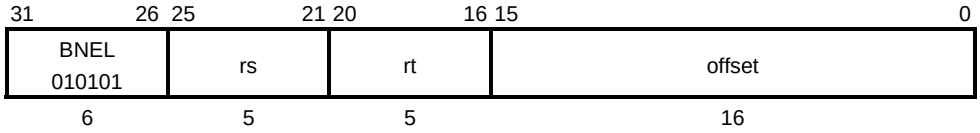
BNEL *rs*, *rt*, *offset*

Branch On Not Equal Likely

動作

if *rs* ≠ *rt* then pc ⇐ pc + *offset*

コード



説明

汎用レジスタ *rs* の内容と汎用レジスタ *rt* の内容を比較し、両者が等しくない場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐します。分岐しない場合は、分岐遅延スロット内の命令は無効になります。ターゲットアドレスは、分岐遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。16 ビット *offset* を 2 ビット左へシフトし、符号拡張した値を PC+4 に加算した結果がターゲットアドレスになります。

例外

なし

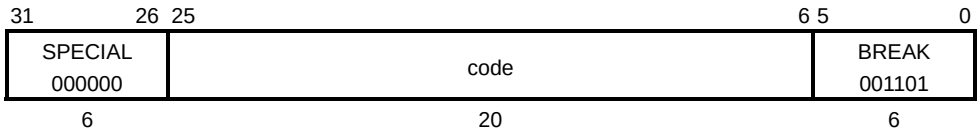
BREAK *code*

Breakpoint Exception

動作

ブレークポイント例外

コード



説明

この命令を実行すると、無条件にブレークポイント例外が発生し、制御を例外ハンドラへ渡します。

命令内の *code* フィールドを使用して、例外ハンドラにパラメータを渡すことができます。例外ハンドラがこのパラメータを使用する場合には、命令を含むメモリワードの内容をデータとしてロードする必要があります。詳細は「9.1.11　ブレークポイント例外」を参照してください。

例外

ブレークポイント例外

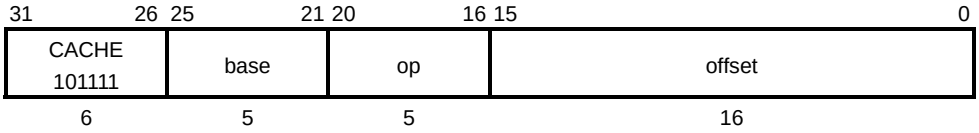
CACHE *op*, *offset* (*base*)

Cache Operation

動作

キャッシュ操作

コード

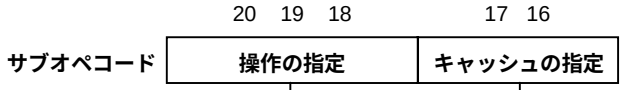


説明

16 ビットの *offset* を符号拡張して、汎用レジスタ *base* の内容に加算し、仮想アドレスを生成します。仮想アドレスは物理アドレスに変換されます。5 ビットのサブオペコード (ビット 20~16) が、その物理アドレスに対するキャッシュの操作を指定します。

ユーザーモードで、Status レジスタの CU[0] ビットがクリアされている場合、CACHE 命令を実行すると、コプロセッサ使用不可例外が発生します。カーネルモードでは、常に CACHE 命令を実行できます。キャッシュが内蔵されていない場合の動作は不定です。

この命令のビット 20~18、17~16 が、以下のようにキャッシュ操作とキャッシュを指定します。



コード[17:16]	名前	キャッシュ
00	I	命令
01	D	データ
1x	-	予約

コード[20:18]	コード[17:16]	名前	オペレーション
000	00/01	Index Invalidate	キャッシュヒット、キャッシュミスに関わらず、物理アドレスで指定されたインデックスのすべてのタグの有効ビットをクリアします。Config レジスタの ICE ビットまたは DCE ビットでキャッシュがディセーブルのときのみ有効です。
001	00/01	Index LRU Bit Clear	物理アドレスで指定されたインデックスのすべてのタグの LRU ビットをクリアします。
010	00/01	Index Lock Bit Clear	物理アドレスで指定されたインデックスのすべてのタグの Lock ビットをクリアします。
100	00/01	Hit Invalidate	キャッシュがヒットした場合、指定されたインデックスのヒットしたタグのみ、その有効ビットをクリアします。

例外

コプロセッサ使用不可例外

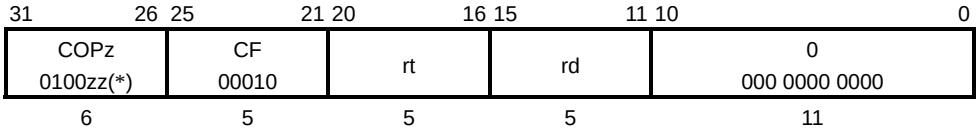
CFC*z* *rt*, *rd*

Move Control From Coprocessor z

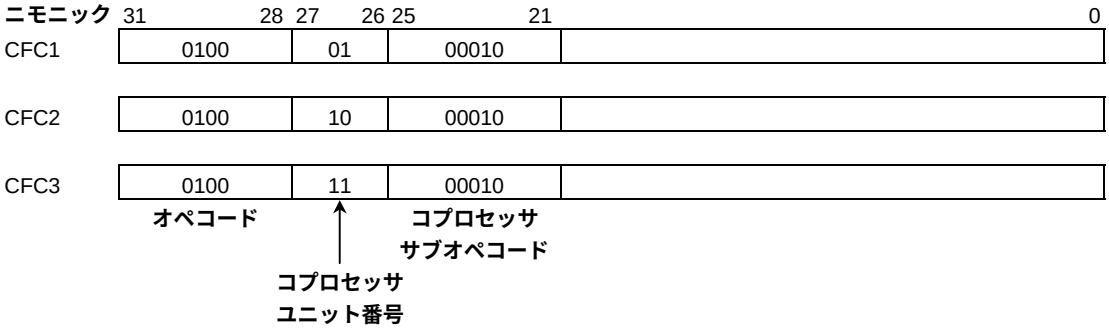
動作

rt ⇐ コプロセッサ *z* のコプロセッサ制御レジスタ *rd*

コード



以下にオペコードのビットコードを示します。オペコードフィールドの下位 2 ビットはコプロセッサユニット番号を示します。



説明

コプロセッサユニット *z* のコプロセッサ制御レジスタ *rd* の内容を、汎用レジスタ *rt* にロードにします。

この命令は CP0 に対しては無効です。

例外

コプロセッサ使用不可例外

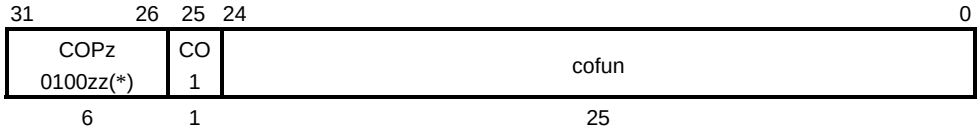
COP*z cofun*

Coprocessor z Operation

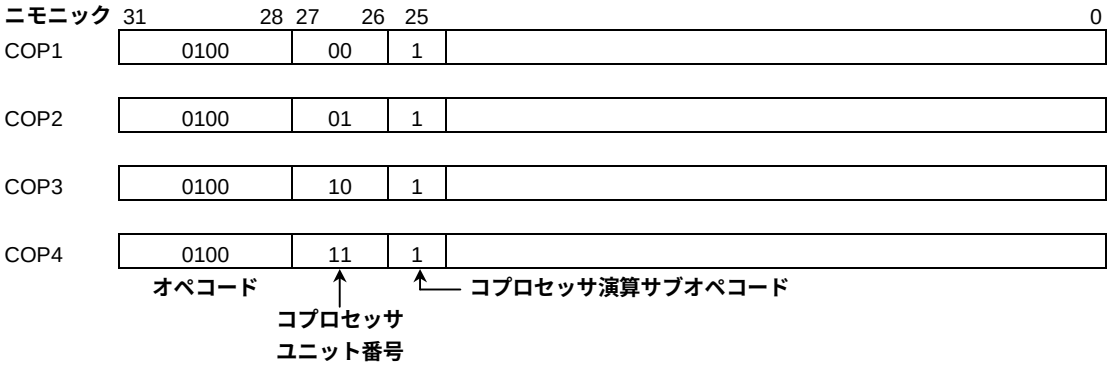
動作

コプロセッサ操作 (*z*, *cofun*)

コード



以下にオペコードのビットコードを示します。オペコードフィールドの下位 2 ビットはプロセッサユニット番号を示します。



説明

コプロセッサ *z* に対して、*cofun* で指定された演算を行います。

COP*z* 命令は、内部コプロセッサレジスタを指定、参照したり、コプロセッサ条件信号 (CPCOND) の状態を変更したりすることはありますが、プロセッサまたはキャッシュ・メモリシステムの内部状態を変更することはありません。

例外

コプロセッサ使用不可例外

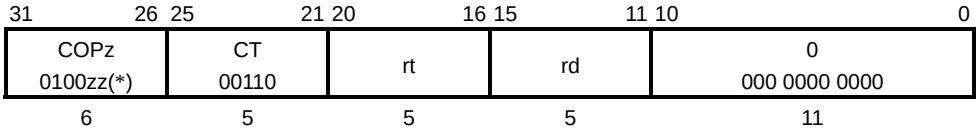
CTC*z rt*, *rd*

Move Control To Coprocessor z

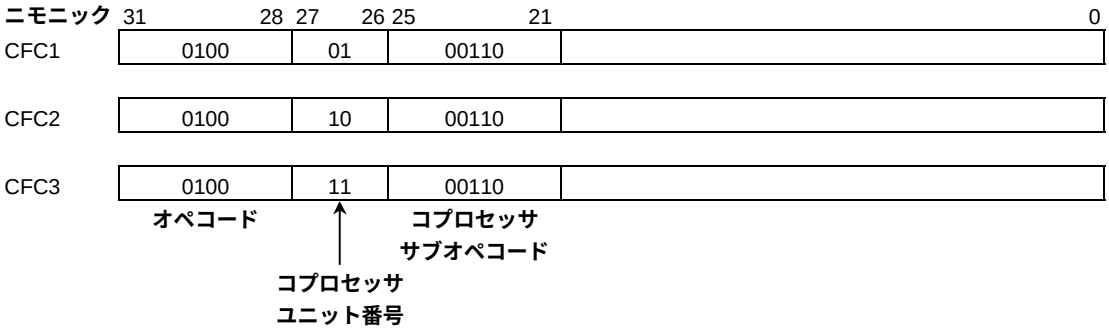
動作

コプロセッサ *z* のコプロセッサ制御レジスタ *rd* ⇐ *rt*

コード



以下にオペコードのビットコードを示します。オペコードフィールドの下位 2 ビットはプロセッサユニット番号を示します。



説明

汎用レジスタ *rt* の内容を、コプロセッサ *z* のコプロセッサ制御レジスタ *rd* にロードします。

この命令は CP0 に対しては無効です。

例外

コプロセッサ使用不可例外

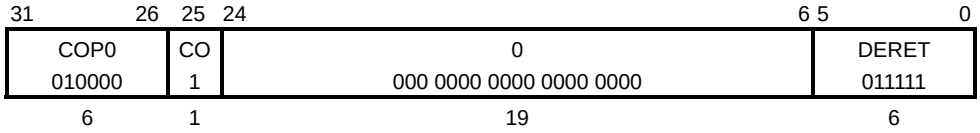
DERET

Debug Exception Return

動作

pc ⇐ DEPC

コード



説明

DERET 命令は、デバッグ例外処理から復帰するための命令です。DEPC レジスタの内容をプログラムカウンタ (PC) にロードすることにより実行されます。詳細については、「9.3.6　デバッグ例外からの復帰」を参照してください。

分岐命令と同様に、DERET 命令には分岐遅延スロットがあり、1 命令 (2 命令サイクル) の遅延後、実行されます。

DERET 命令は、DEPC レジスタのビット 0 を PC の ISA モードビット (ビット 0) を復元し、デバッグ例外が実行される前の ISA モードになります。

DERET 命令の遅延スロットには、NOP 命令を置かなければなりません。また、DERET 命令を、ジャンプまたは分岐遅延スロットに置いてはいけません。

デバッグモードでないとき (Debug レジスタ DM ビットが 0 のとき) は、DERET 命令の動作は不定です。

通常、デバッグ例外が発生すると、例外の原因となった命令のアドレスが自動的に DEPC レジスタに保存されます。MTC0 命令を使って DEPC レジスタに戻りアドレスを設定したい場合は、デバッグ例外ハンドラで少なくとも 2 つの命令を実行してから、DERET 命令を実行しなければなりません。MTC0 命令で Debug レジスタの書き込みを実行した直後に、DERET 命令を実行することは禁止されています。実行した場合は、レジスタの値は不定になります。また、MFC0 命令を使って Debug レジスタを読み出した直後に、DERET 命令を実行することは禁止されています。実行した場合は、Debug レジスタの値が不定になります。

例外

コプロセッサ使用不可例外

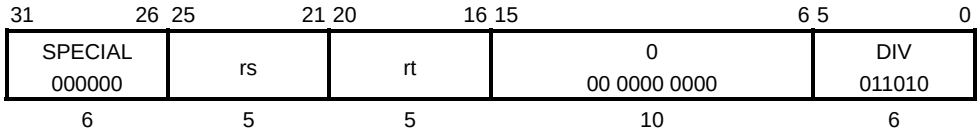
DIV *rs*, *rt*

Divide

動作

LO ⇐ *rs* ÷ *rt*;
HI ⇐ *rs* MOD *rt*

コード



説明

汎用レジスタ *rs* の内容を汎用レジスタ *rt* の内容で除算します。両オペランドとも、符号付き整数として扱われます。商は LO に格納され、剰余は HI レジスタに格納されます。整数オーバフロー例外は発生しません。

除数が 0 の場合、DIV 命令の結果は確定しません。通常、DIV 命令の後に、ゼロ除算とオーバフローを検査する命令を置きます。

除算命令は、専用の除算ユニットで実行されるため、他の命令の実行を並行して継続できます。除算ユニットは、キャッシュミス、遅延サイクル、例外が起きたときでも、実行を継続します。

除算命令が完了する前に、MFHI、MFLO、MADD、 MADDU 命令で除算結果を読もうとすると、パイプラインがストールします (「5.4　除算命令」を参照)。

例外

なし

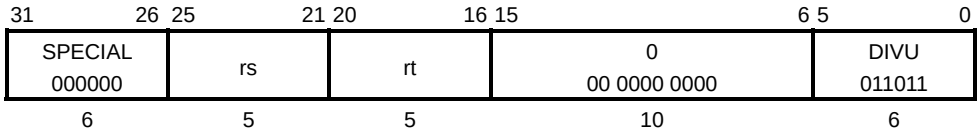
DIVU *rs*, *rt*

Divide Unsigned

動作

LO ⇐ *rs* ÷ *rt*;
HI ⇐ *rs* MOD *rt*

コード



説明

汎用レジスタ *rs* の内容を汎用レジスタ *rt* の内容で除算します。商は LO レジスタに格納され、剰余は HI レジスタに格納されます。整数オーバフロー例外を発生しません。DIV 命令との唯一の違いは、DIVU 命令ではオペランドが符号なし整数として扱われるという点だけです。

例外

なし

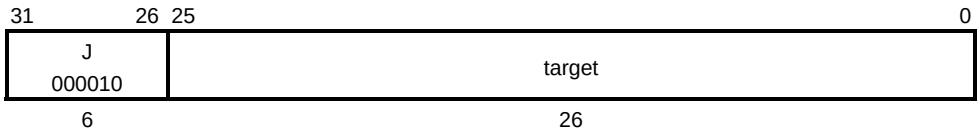
J *target*

Jump

動作

pc ⇐ pc[31:28] ‖ *target* ‖ 00

コード



説明

1 命令 (2 命令サイクル) の遅延後、無条件にターゲットアドレスへジャンプします。ターゲットアドレスは、ジャンプ遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。26 ビット *target* を 2 ビット左へシフトし、PC+4 の上位 4 ビットと連結した結果がターゲットアドレスになります。

J 命令では、ターゲットアドレスは、2^{28} バイトセグメント内でなければなりません。任意の 32 ビットのアドレスへジャンプするには、アドレスをいったんレジスタに格納してから、JR 命令を使います (「3.4.6　32 ビットのアドレスへのジャンプ」を参照)。

例外

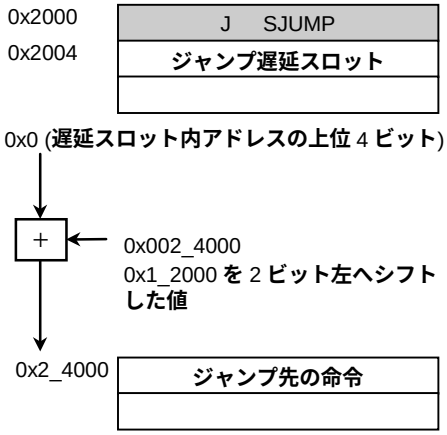
なし

使用例

```
J  SJUMP
```

上記のジャンプ命令がアドレス 0x2000 にあり、ラベル SJUMP が 0x2_4000 に絶対アドレス化される場合、SJUMP はアセンブラ・リンカにより 0x1_2000 に変換されます。

プログラムの処理は、無条件にアドレス 0x2_4000 にジャンプします。ジャンプ遅延スロット内の命令はジャンプのまえに実行されます。


0x2000
J　SJUMP
0x2004
ジャンプ遅延スロット
0x0 (遅延スロット内アドレスの上位 4 ビット)
+
0x002_4000
0x1_2000 を 2 ビット左へシフトした値
0x2_4000
ジャンプ先の命令

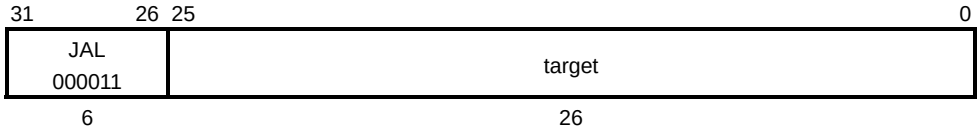
JAL *target*

Jump And Link

動作

r31 ⇐ pc + 8; pc ⇐ pc[31:28] ‖ *target* ‖ 00

コード



説明

1 命令 (2 命令サイクル) の遅延後、無条件にターゲットアドレスにジャンプします。ターゲットアドレスは、ジャンプ遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。26 ビット *target* を 2 ビット左へシフトし、PC+4 の上位 4 ビットと連結した結果がターゲットアドレスになります。プログラムカウンタ (PC) の ISA モードビットは変化しません。

ジャンプ遅延スロットの次の命令のアドレスを、リンクレジスタ r31 (ra) に格納します。また、r31 の最下位ビットに、ジャンプ前の ISA モードビットを格納します。

JAL 命令では、ターゲットアドレスは、2^{28} バイトセグメント内でなければなりません。任意の 32 ビットのアドレスへジャンプするには、アドレスをいったんレジスタに格納してから、JR 命令を使います (「3.4.6　32 ビットのアドレスへのジャンプ」を参照)。

例外

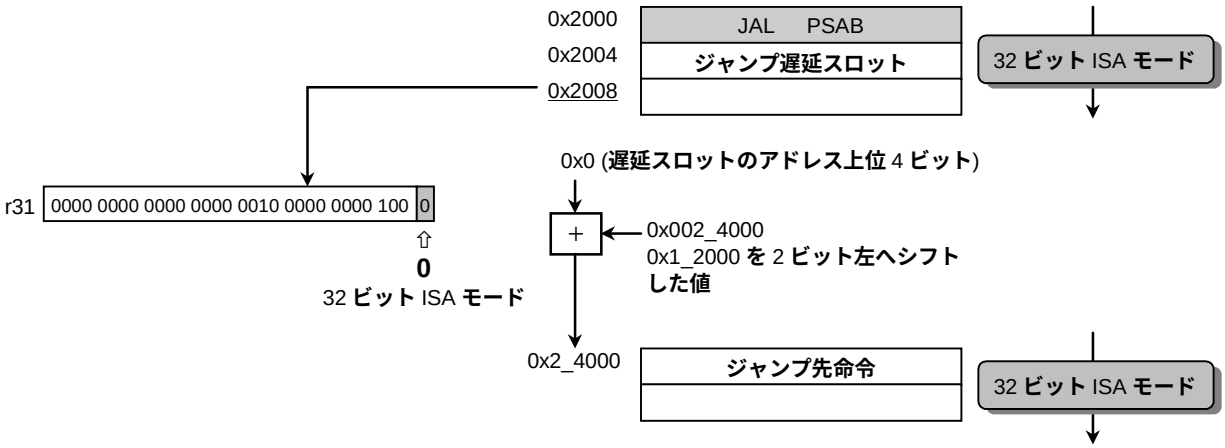
なし

使用例

```
JAL PSUB
```

上記のジャンプ命令がアドレス 0x2000 にあり、ラベル PSUB が 0x2_4000 に絶対アドレス化される場合、以下に図示するように、PSUB はアセンブラ・リンカにより 0x1_2000 に変換されます。

プログラムの処理は、無条件にアドレス 0x2_4000 にジャンプします。ジャンプ遅延スロット内の命令は、ジャンプのまえに実行されます。また、ジャンプ遅延スロットの次の命令のアドレスが、リンクレジスタ r31 に格納されます。


0x2000
0x2004
0x2008
JAL　PSAB
ジャンプ遅延スロット
32 ビット ISA モード
r31
0000 0000 0000 0000 0010 0000 0000 100
0
0
32 ビット ISA モード
0x0 (遅延スロットのアドレス上位 4 ビット)
+
0x002_4000
0x1_2000 を 2 ビット左へシフトした値
0x2_4000
ジャンプ先命令
32 ビット ISA モード

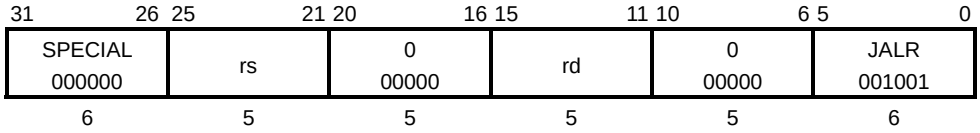
JALR (*rd*,) *rs*

Jump And Link Register

動作

rd or r31 ⇐ pc + 8; pc ⇐ *rs*

コード



説明

1 命令 (2 命令サイクル) の遅延後、汎用レジスタ *rs* の最下位ビットを 0 にマスクしたアドレスに無条件にジャンプします。*rs* の最下位ビットの値によって、ISA モードが切り換わります。また、ジャンプ遅延スロットの次の命令のアドレスを、汎用レジスタ (rd) に格納します。オペランド *rd* を省略すると、デフォルトで r31 (ra) が使用されます。

rs と *rd* に、同じレジスタを指定できません。*rd* として *rs* を指定すると、戻りアドレスによって *rs* の内容が破壊されてしまいます。すると、例外や割り込みにより、ジャンプ遅延スロットの命令が完了しなかった場合、例外処理後、ジャンプ命令から実行を再開できなくなってしまうためです。

32 ビット ISA では、命令はすべてワード境界で位置合わせされなければなりません。したがって、ジャンプ後の ISA モードを 32 ビットに指定する場合、ターゲットレジスタ (*rs*) の下位 2 ビットは 0 でなければなりません。下位 2 ビットが 0 でない場合、ジャンプ先の命令をフェッチするときにアドレスエラー例外が発生します。

例外

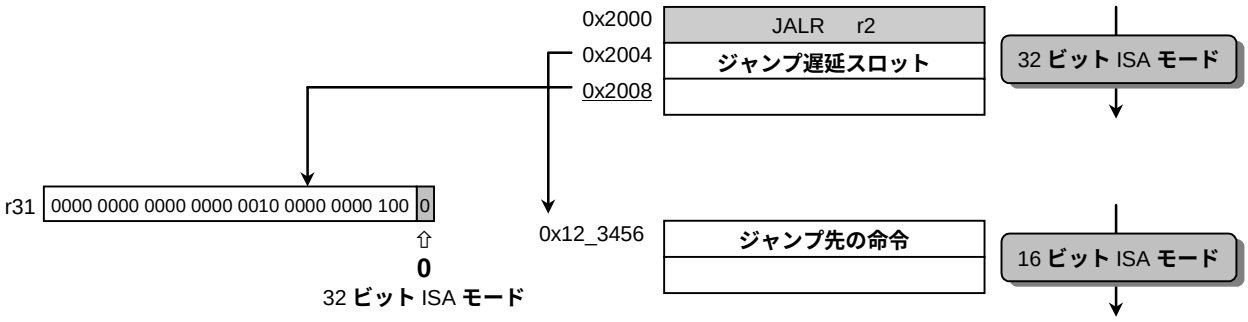
なし

使用例

レジスタ r2 の内容が 0x0012_3457 で、以下のジャンプ命令がアドレス 0x0000_2000 に置かれているとします。

```
JALR r2
```

上記の命令を実行すると、プログラムの処理は、0x0012_3457 の最下位ビットを 0 にマスクしたアドレス 0x0012_3456 にジャンプします。ジャンプ遅延スロット内の命令はジャンプのまえに実行されます。レジスタ r2 の最下位ビットは 1 に設定されているので、ジャンプ後の ISA モードビットは 1 に変化し、16 ビット ISA モードになります。戻りアドレス 0x0000_2008 は、ISA モードビットと共にリンクレジスタ r31 に格納されます。


0x2000
JALR　r2
0x2004
ジャンプ遅延スロット
0x2008
32 ビット ISA モード
r31
0000 0000 0000 0000 0010 0000 0000 100
0
0
32 ビット ISA モード
0x12_3456
ジャンプ先の命令
16 ビット ISA モード

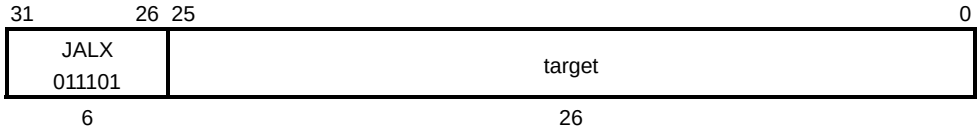
JALX *target*

Jump And Link eXchange

動作

r31 ⇐ pc + 8; pc[31:1] ⇐ pc[31:28] ‖ *target* ‖ 00; pc[0] ⇐ NOT pc[0]

コード



説明

1 命令 (2 命令サイクル) の遅延後、無条件にターゲットアドレスにジャンプします。ターゲットアドレスは、ジャンプ遅延スロット内の命令のアドレス (PC+4) に対して相対的に計算されます。26 ビット *target* を 2 ビット左へシフトし、PC+4 の上位 4 ビットを連結した結果がターゲットアドレスになります。プログラムカウンタ (PC) の ISA モードビットは無条件に変化します。

ジャンプ遅延スロットの次の命令のアドレスを、リンクレジスタ r31 (ra) に格納します。また、r31 の最下位ビットに、ジャンプ前の ISA モードビットを格納します。

例外

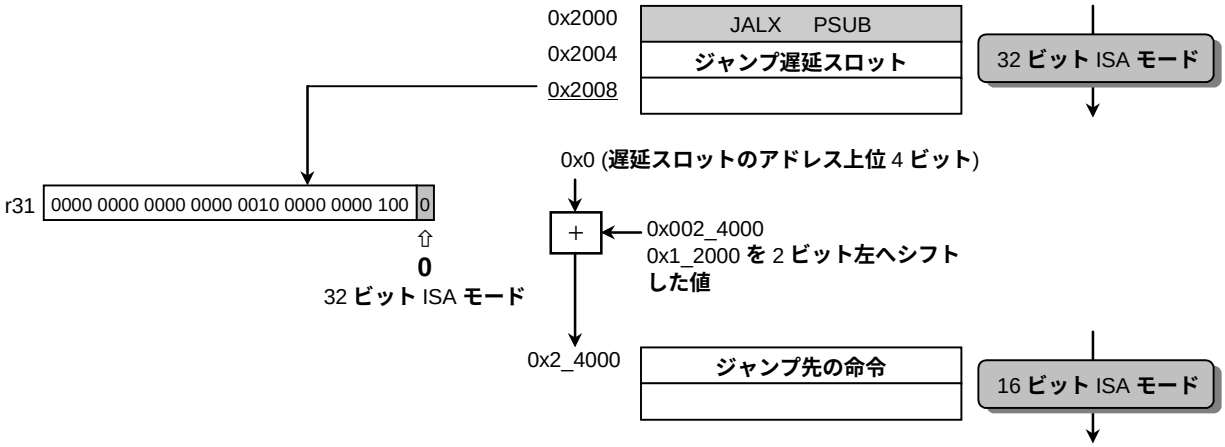
なし

使用例

```
JALX PSUB
```

上記のジャンプ命令がアドレス 0x0000_2000 にあり、ラベル PSUB が 0x2_4000 に絶対アドレス化される場合、以下に図示するように、PSUB はアセンブラ・リンカにより 0x1_2000 に変換されます。

プログラムの処理は、無条件にアドレス 0x2_4000 にジャンプします。ジャンプ遅延スロット内の命令は、ジャンプのまえに実行されます。ISA モードビットは無条件に変化し、16 ビット ISA モードになります。また、戻りアドレス 0x0000_2008 が、ISA モードビットと共にリンクレジスタ r31 に格納されます。


0x2000
JALX　PSUB
0x2004
ジャンプ遅延スロット
0x2008
32 ビット ISA モード
0x0 (遅延スロットのアドレス上位 4 ビット)
r31
0000 0000 0000 0000 0010 0000 0000 100
0
+
0x002_4000
0x1_2000 を 2 ビット左へシフトした値
0
32 ビット ISA モード
0x2_4000
ジャンプ先の命令
16 ビット ISA モード

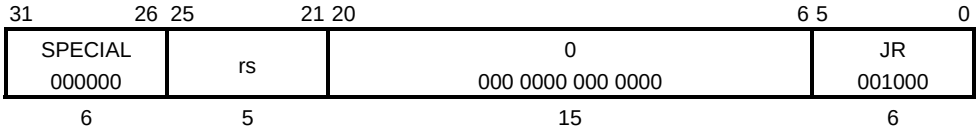
JR *rs*

Jump Register

動作

pc ⇐ *rs*

コード



説明

1 命令 (2 命令サイクル) の遅延後、汎用レジスタ *rs* の最下位ビットを 0 にマスクしたアドレスに無条件にジャンプします。*rs* の最下位ビットの値によって ISA モードが切り換わります。

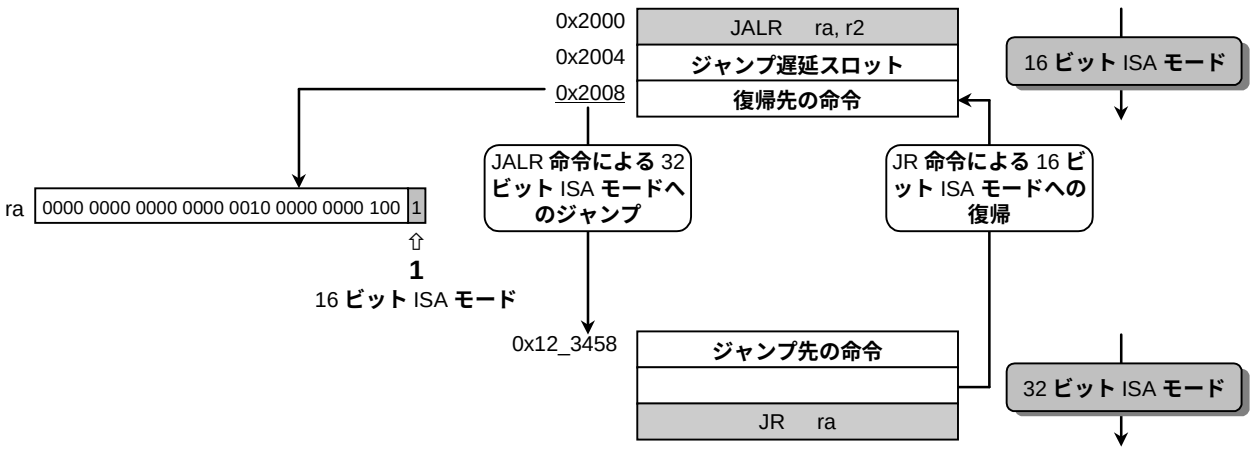
32 ビット ISA では、命令はすべてワード境界に位置合わせされていなければなりません。そのため、32 ビットのモードにジャンプするとき、ターゲットレジスタ (*rs*) の下位 2 ビットは 0 でなければなりません。下位 2 ビットが 0 でない場合、プロセッサがジャンプ先の命令をフェッチするときにアドレスエラー例外が発生します。

例外

なし

使用例

以下の例では、16 ビットの JALR 命令により 32 ビットモードのルーチンにジャンプします。32 ビットのルーチンの最後で、JR 命令により戻りアドレスをリンクレジスタ r31 (ra) からプログラムカウンタ (PC) に復元しています。JALR 命令により、ISA モードが ra の最下位ビットに保存されているので、32 ビットルーチンの終わりで JR 命令を実行すると、16 ビット ISA モードに戻ります。



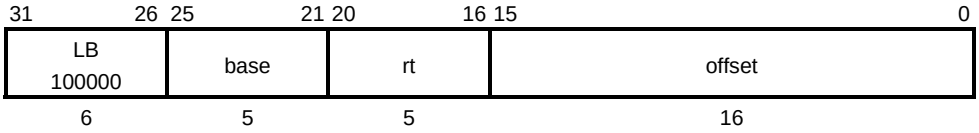
LB *rt*, *offset* (*base*)

Load Byte

動作

rt ⇐ {*offset* (*base*)}

コード



説明

16 ビット *offset* を符号拡張して、汎用レジスタ *base* の内容に加算することにより、実効アドレス (EA) を生成します。EA でアドレス指定されたメモリ中のバイトデータを符号拡張して汎用レジスタ *rt* にロードします。

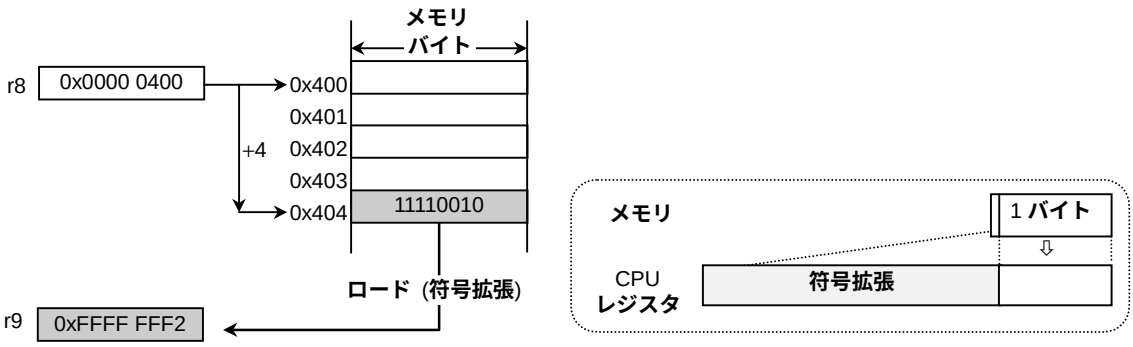
例外

アドレスエラー例外

使用例

レジスタ r8 の値が 0x0000_0400 で、アドレス 0x404 の内容が 0xF2 の場合、以下の命令を実行すると、レジスタ r9 に 0xFFFF_FFF2 がロードされます。

```
LB r9,4(r8)
```



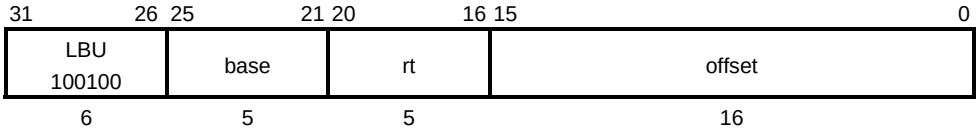
LBU *rt*, *offset* (*base*)

Load Byte Unsigned

動作

rt ⇐ {*offset* (*base*)}

コード



説明

16 ビット *offset* を符号拡張して、汎用レジスタ *base* の内容に加算することにより、実効アドレス (EA) を生成します。EA でアドレス指定されたメモリ中のバイトデータをゼロ拡張して、汎用レジスタ *rt* にロードします。

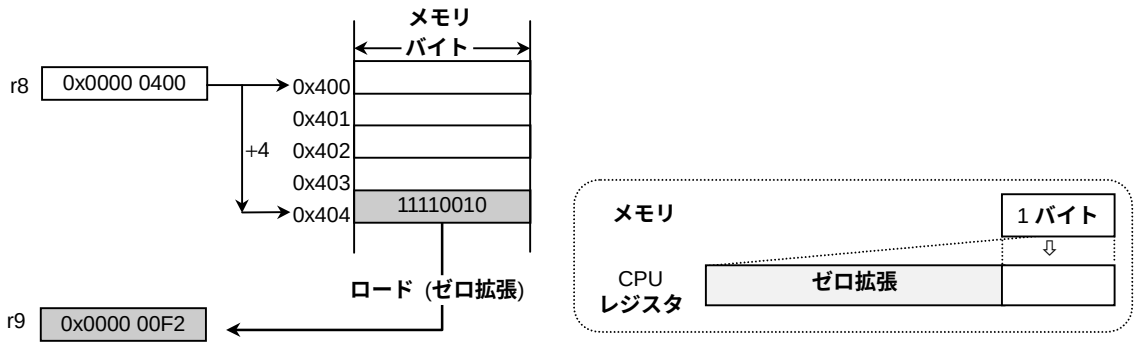
例外

アドレスエラー例外

使用例

レジスタ r8 の値が 0x0000_0400 で、アドレス 0x404 の内容が 0xF2 の場合、以下の命令を実行すると、レジスタ r9 には 0x0000_00F2 がロードされます。

```
LBU r9,4(r8)
```



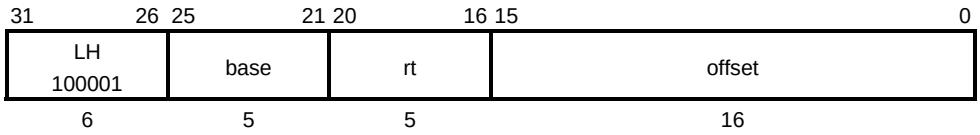
LH *rt*, *offset* (*base*)

Load Halfword

動作

rt ⇐ {*offset* (*base*)}

コード



説明

16 ビット *offset* を符号拡張して汎用レジスタ *base* の内容に加算することにより、実効アドレス (EA) を生成します。EA でアドレス指定されたメモリ中のハーフワードデータを符号拡張して、汎用レジスタ *rt* にロードします。

実効アドレスの最下位ビットが 0 でない (実効アドレスがハーフワード境界でない) 場合、アドレスエラー例外が発生します。

例外

アドレスエラー例外

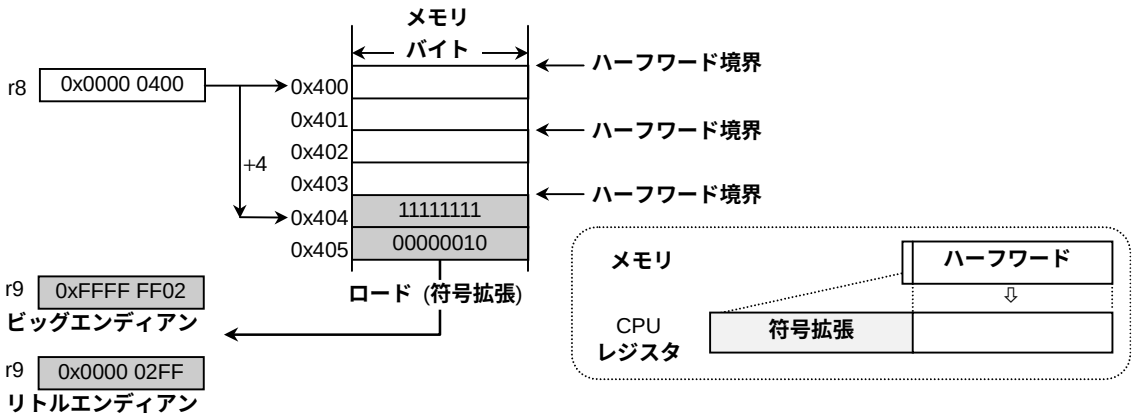
使用例

レジスタ r8 の値が 0x0000_0400 で、アドレス 0x404 と 0x405 の内容がそれぞれ 0xFF と 0x02 の場合、以下の命令を実行すると、レジスタ r9 にはビッグエンディアンのときは 0xFFFF_FF02 がロードされ、リトルエンディアンのときは 0x0000_02FF がロードされます。

```
LH r9,4(r8)
```

また、以下の命令を実行すると、0x403 はハーフワード境界でないので、アドレスエラー例外が発生します。

```
LH r9,3(r8)
```


メモリ
バイト
ハーフワード境界
r8
0x0000 0400
0x400
0x401
0x402
0x403
0x404
0x405
+4
11111111
00000010
ハーフワード境界
ハーフワード境界
メモリ
ハーフワード
CPU
レジスタ
符号拡張
r9
0xFFFF FF02
ビッグエンディアン
ロード（符号拡張）
r9
0x0000 02FF
リトルエンディアン

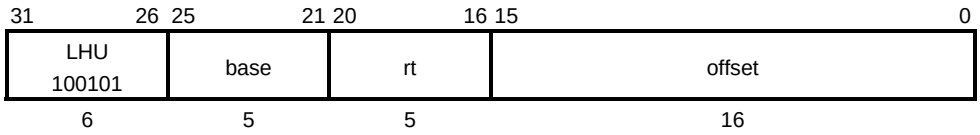
LHU *rt*, *offset* (*base*)

Load Halfword Unsigned

動作

rt ⇐ {*offset* (*base*)}

コード



説明

16 ビット *offset* を符号拡張して、汎用レジスタ *base* の内容に加算することにより、実効アドレス (EA) を生成します。EA でアドレス指定されたメモリ中のハーフワードデータをゼロ拡張し、汎用レジスタ *rt* にロードします。

実効アドレスの最下位ビットが 0 でない (実効アドレスがハーフワード境界でない) 場合、アドレスエラー例外が発生します。

例外

アドレスエラー例外

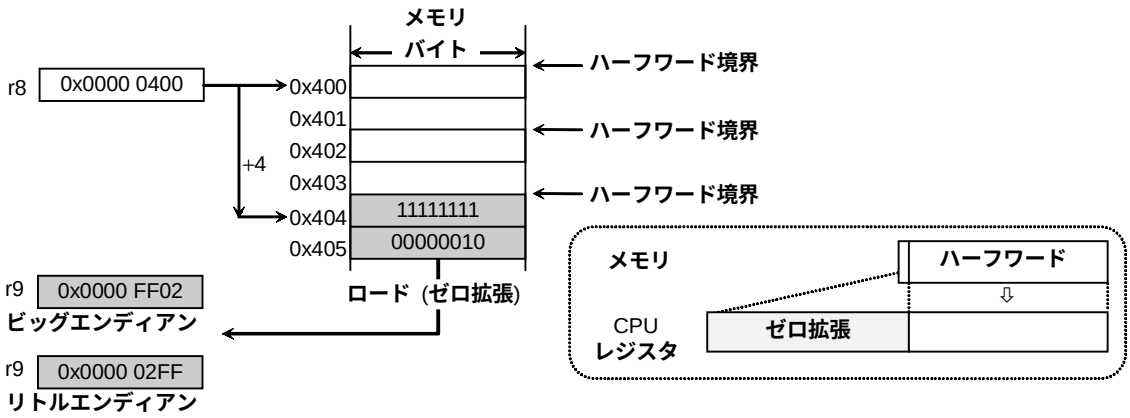
使用例

レジスタ r8 の値が 0x0000_0400 で、アドレス 0x404 と 0x405 の内容がそれぞれ 0xFF と 0x02 の場合、以下の命令を実行すると、レジスタ r9 にはビッグエンディアンのときは 0x0000_FF02 がロードされ、リトルエンディアンのときは 0x0000_02FF がロードされます。

```
LHU r9,4(r8)
```

また、以下の命令を実行すると、0x403 はハーフワード境界でないので、アドレスエラー例外が発生します。

```
LH r9,3(r8)
```


メモリ
バイト
ハーフワード境界
ハーフワード境界
ハーフワード境界
r8
0x0000 0400
0x400
0x401
0x402
0x403
0x404
0x405
+4
11111111
00000010
ロード (ゼロ拡張)
r9
0x0000 FF02
ビッグエンディアン
r9
0x0000 02FF
リトルエンディアン
メモリ
ハーフワード
CPU
レジスタ
ゼロ拡張

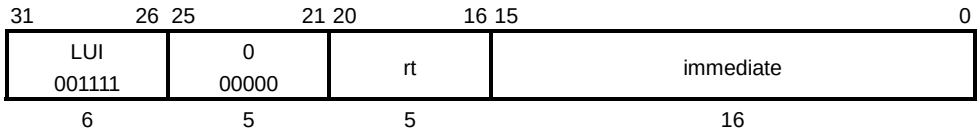
LUI *rt*, *immediate*

Load Upper Immediate

動作

rt ⇐ *immediate* || 0x0000

コード



説明

16 ビット *immediate* を 16 ビット左へシフトし、下位 16 ビットのを 0 で埋めた値を汎用レジスタ *rt* にロードします。

例外

なし

使用例

以下の命令は、レジスタ r9 に 0x1234_0000 をロードします。

```
LUI r9,0x1234
```

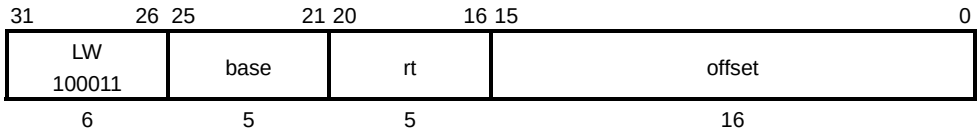
LW *rt*, *offset* (*base*)

Load Word

動作

rt ⇐ {*offset* (*base*)}

コード



説明

16 ビット *offset* を符号拡張して汎用レジスタ *base* の内容に加算することにより、実効アドレス (EA) を生成します。EA でアドレス指定されたメモリ中のワードデータを、汎用レジスタ *rt* にロードします。

実効アドレスの下位 2 ビットが 0 でない (実効アドレスがワード境界でない) 場合、アドレスエラーが発生します。

例外

アドレスエラー例外

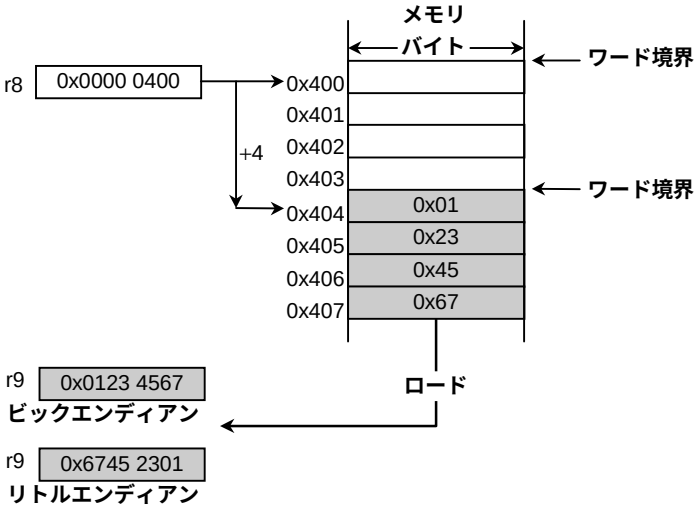
使用例

レジスタ r8 の値が 0x0000_0400 で、アドレス 0x404 から 0x407 の内容が 0x01、0x23、0x45、0x67 の場合、以下の命令を実行すると、レジスタ r9 にはビッグエンディアンモードのときは 0x0123_4567 がロードされ、リトルエンディアンモードのときは 0x6745_2301 がロードされます。

```
LW r9,4(r8)
```

また、以下の命令を実行すると、0x405 はワード境界でないので、アドレスエラー例外が発生します。

```
LW r9,5(r8)
```


メモリ
バイト
ワード境界
r8
0x0000 0400
0x400
0x401
0x402
0x403
ワード境界
+4
0x404
0x405
0x406
0x407
0x01
0x23
0x45
0x67
ロード
r9
0x0123 4567
ビックエンディアン
r9
0x6745 2301
リトルエンディアン

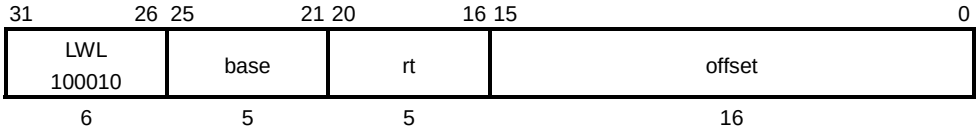
LWL *rt*, *offset* (*base*)

Load Word Left

動作

rt ⇐ {*offset* (*base*)}

コード



説明

16 ビット *offset* を符号拡張し、汎用レジスタ *base* の内容に加算することにより、実効アドレス (EA) を生成します。ワード境界にない、メモリ中のワードデータの上位部分を汎用レジスタ *rt* に左詰めでロードします。

実効アドレスがワード境界に位置していないことによる、アドレスエラー例外は発生しません。

直前のロード命令と後続の LWL 命令は、同じ汎用レジスタを *rt* として指定できます。汎用レジスタ *rt* の内容は、プロセッサコア内で内部的にバイパシング (フォワーディング) されるので、両命令の間に NOP 命令を入れる必要はありません。

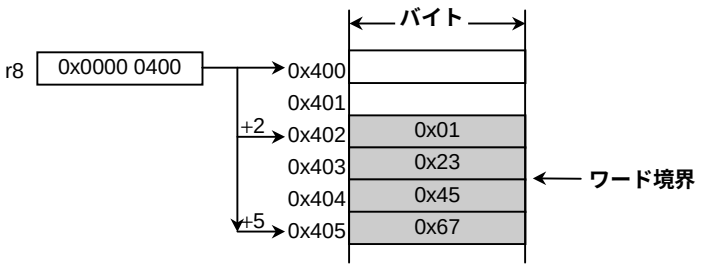
ワード境界に位置合わせされていないメモリ中のワードを、汎用レジスタにロードするときに、LWL 命令と LWR 命令を組み合わせて使用します。

例外

アドレスエラー例外

使用例

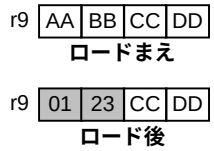
レジスタ r8 の値が 0x0000_0400 で、アドレス 0x402 ~ 0x405 の内容が 0x01、0x23、0x45、0x67 であるとします。



- ビッグエンディアンのとき

```
LWL r9,2(r8)
```

上記の命令は、アドレス 0x402 のバイト位置から、上位アドレス方向へワード境界に達するまで、データを読み出し、r9 の最上位バイトから順にロードします。その結果を以下に示します。



(a) **ビッグエンディアン**

- リトルエンディアンのとき

```
LWL r9,5(r8)
```

上記の命令は、アドレス 0x405 のバイト位置から、下位アドレス方向へワード境界に達するまで、データを読み出し、r9 の最上位バイトから順にロードします。その結果を以下に示します。

(b) **リトルエンディアン**

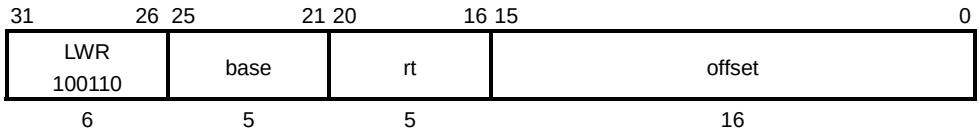
LWR *rt*, *offset* (*base*)

Load Word Right

動作

rt ⇐ {*offset* (*base*)}

コード



説明

16 ビット *offset* を符号拡張し、汎用レジスタ *base* の内容に加算することにより、実効アドレス (EA) を生成します。ワード境界にない、メモリ中のワードデータの下位部分を汎用レジスタ *rt* に右詰めでロードします。

実効アドレスがワード境界に位置していないことによる、アドレスエラー例外は発生しません。

直前のロード命令と後続の LWLR 命令は、同じ汎用レジスタを *rt* として指定できます。汎用レジスタ *rt* の内容は、プロセッサコア内で内部的にバイパシング (またはフォワーディング) されるので、両命令の間に NOP 命令を入れる必要はありません。

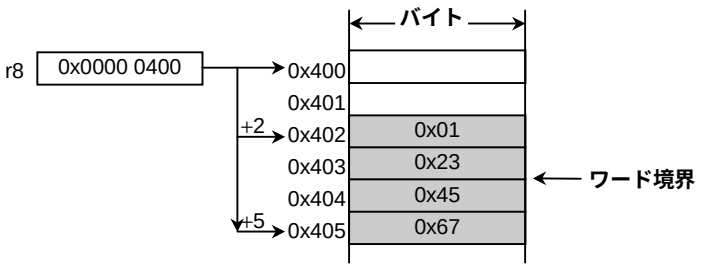
ワード境界に位置合わせされていないメモリ中のワードを、汎用レジスタにロードするときに、LWL 命令と LWR を組み合わせて使用します。

例外

アドレスエラー例外

使用例

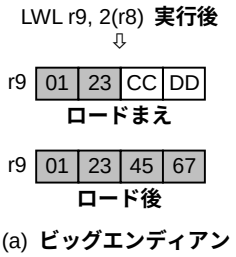
レジスタ r8 の値が 0x0000_0400 で、アドレス 0x402 ~ 0x405 の内容が 0x01、0x23、0x45、0x67 であるとします。



- ビッグエンディアンのとき

```
LWR r9,5(r8)
```

上記の命令は、アドレス 0x405 のバイト位置から、下位アドレス方向へワード境界に達するまで、データを読み出し、r9 の最下位バイトから順にロードします。その結果を以下に示します。

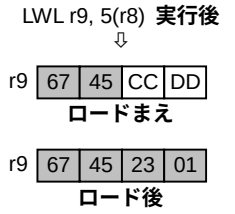


(a) **ビッグエンディアン**

- リトルエンディアンのとき

```
LWR r9,2(r8)
```

上記命令は、アドレス 0x402 のバイト位置から、上位アドレス方向へワード境界に達するまで、データを読み出し、r9 の最下位バイトから順にロードします。その結果を以下に示します。



(b) **リトルエンディアン**

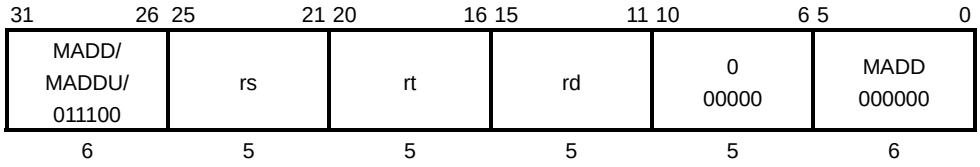
MADD (*rd*,) *rs*, *rt*

Multiply and Add

動作

HI ⇐ (HI ‖ LO) + (*rs* × *rt*) の上位ワード;
LO ⇐ (HI ‖ LO) + (*rs* × *rt*) の下位ワード;
rd ⇐ (HI ‖ LO) + (*rs* × *rt*) の下位ワード

コード



説明

汎用レジスタ *rs* の内容と汎用レジスタ *rt* の内容を乗算し、その積を HI・LO レジスタに格納されているダブルワードの値に加算します。*rs* と *rt* を符号付き整数として扱います。結果の上位ワードを HI レジスタに格納し、下位ワードを LO レジスタに格納します。デスティネーションレジスタ *rd* が指定されている場合、結果の下位ワードを *rd* にも格納します。

rd を省略すると、デフォルトで r0 になり、その結果、汎用レジスタに移送された下位ワードは廃棄されます。

いかなる場合でも、整数オーバフロー例外は発生しません。

例外

なし

使用例

HI レジスタには 0x0000_0000 が、LO レジスタには 0xFFFF_FFFF が格納されていて、汎用レジスタ r2 には 0x0123_4567 が、r3 に 0x89AB_CDEF が格納されているとします。

```
MADD r4,r2,r3
```

このとき、上記の命令は、以下の演算を実行します。

0x0000_0000_FFFF_FFFF + (0x0123_4567 × 0x89AB_CDEF)
= 0x0000_0000_FFFF_FFFF + 0xFF79_5E36_C94E_4629
= 0xFF79_5E37_C94E_4628

結果の上位ワード 0xFF79_5E37 が HI レジスタに格納され、下位ワード 0xC94E_4628 が LO レジスタと r4 に格納されます。

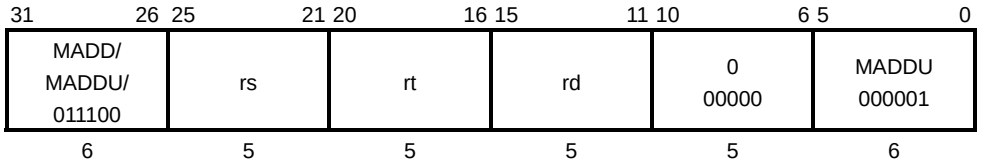
MADDU (*rd*,) *rs*, *rt*

Multiply and Add Unsigned

動作

HI ⇐ (HI ‖ LO) + (*rs* × *rt*) の上位ワード;
LO ⇐ (HI ‖ LO) + (*rs* × *rt*) の下位ワード;
rd ⇐ (HI ‖ LO) + (*rs* × *rt*) の下位ワード

コード



説明

汎用レジスタ *rs* の内容と汎用レジスタ *rt* の内容を乗算し、その積を HI・LO レジスタに格納されているダブルワードの値に加算します。*rs* と *rt* を符号なし整数として扱います。結果の上位ワードを HI レジスタに格納し、下位ワードを LO レジスタに格納します。デスティネーションレジスタ *rd* が指定されている場合、結果の下位ワードを *rd* にも格納します。

rd を省略すると、デフォルトで r0 になり、その結果、汎用レジスタに移送された下位ワードは廃棄されます。

いかなる場合でも、整数オーバフロー例外は発生しません。

例外

なし

使用例

HI レジスタには 0x0000_0000 が、LO レジスタには 0xFFFF_FFFF が格納されていて、汎用レジスタ r2 には 0x0123_4567 が、r3 には 0x89AB_CDEF が格納されているとします。

```
MADDU r4,r2,r3
```

このとき、上記の命令は、以下の演算を実行します。

0x0000_0000_FFFF_FFFF + (0x0123_4567 × 0x89AB_CDEF)
= 0x0000_0000_FFFF_FFFF + 0x009C_A39D_C94E_4629
= 0x009C_A39E_C94E_4628

結果の上位ワード 0x009C_A39E が HI レジスタに格納され、下位ワード 0xC94E_4628 が LO レジスタと r4 に格納されます。

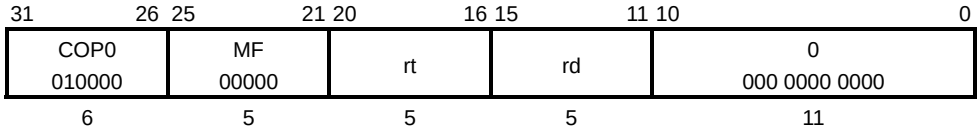
MFC0 *rt*, *rd*

Move From System Control Coprocessor (CP0)

動作

rt ⇐ CP0 のコプロセッサレジスタ *rd*

コード



説明

CP0 レジスタ *rd* の内容を汎用レジスタ *rt* にロードします。

RFE 命令の直前に、MFC0 命令により Status レジスタの内容を読み出すことは禁止されています。そのような読み出しを行うと、Status レジスタの内容は不定になります。

同様に、DERET 命令の直前に MFC0 命令により Debug レジスタの内容を読み出すことは禁止されています。そのような読み出しを行うと、Debug レジスタの内容は不定になります。

例外

コプロセッサ使用不可例外

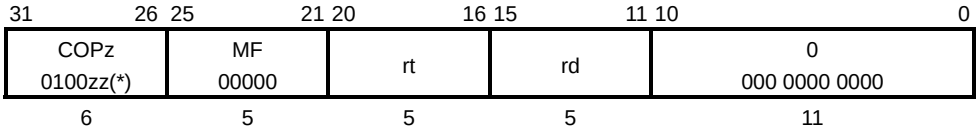
MFC*z* *rt*, *rd*

Move From Coprocessor z

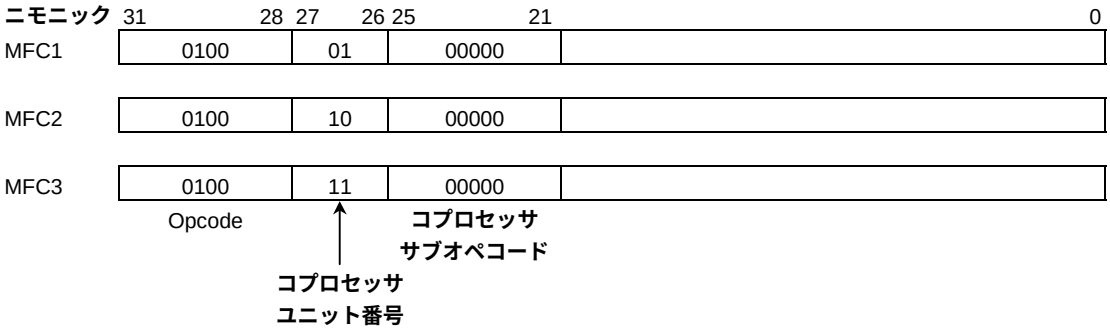
動作

rt ⇐ コプロセッサ *z* のコプロセッサレジスタ *rd*

コード



以外にオペコードビットコードを示します。オペコードフィールドの下位 2 ビットはコプロセッサユニット番号を示します。



説明

コプロセッサユニット *z* のコプロセッサレジスタ *rd* の内容を、汎用レジスタ *rt* にロードします。

例外

コプロセッサ使用不可例外

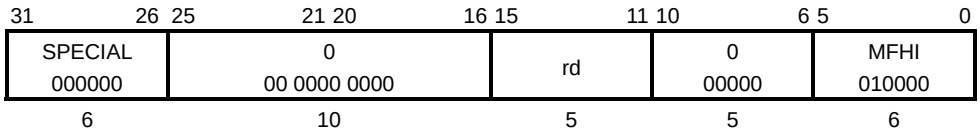
MFHI *rd*

Move From HI

動作

rd ⇐ HI

コード



説明

HI レジスタの内容を汎用レジスタ *rd* にロードします。

例外

なし

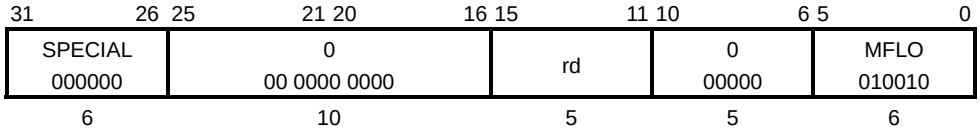
MFLO *rd*

Move From LO

動作

rd ⇐ LO

コード



説明

LO レジスタの内容を汎用レジスタ *rd* にロードします。

例外

なし

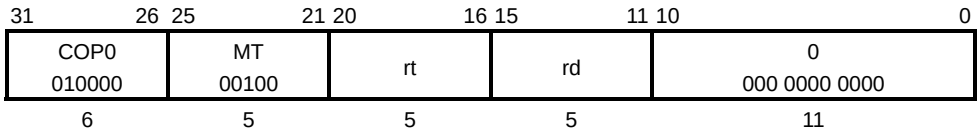
MTC0 *rt*, *rd*

Move To System Control Coprocessor (CP0)

動作

CP0 のコプロセッサレジスタ *rd* ⇐ *rt*

コード



説明

汎用レジスタ *rt* の内容を CP0 レジスタ *rd* にロードします。

RFE 命令の直前に、MTC0 命令により Status レジスタに書き込むことは禁止されています。そのような書き込みを行うと、Status レジスタの内容は不定になります。

同様に、DERET 命令の直前に MTC0 命令により Debug レジスタに書き込むことは禁止されています。そのような書き込みを行うと、Debug レジスタの内容は不定になります。

MTC0 命令により、仮想アドレス変換システムの状態が変わることがあるので、直前や直後のロード・ストア命令の動作は確定しません。

例外

コプロセッサ使用不可例外

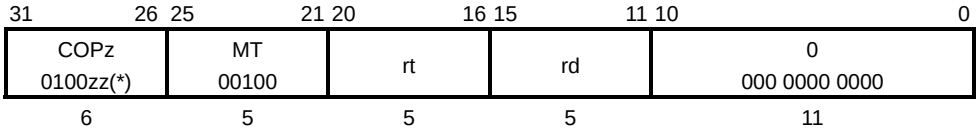
MTC*z rt*, *rd*

Move To Coprocessor z

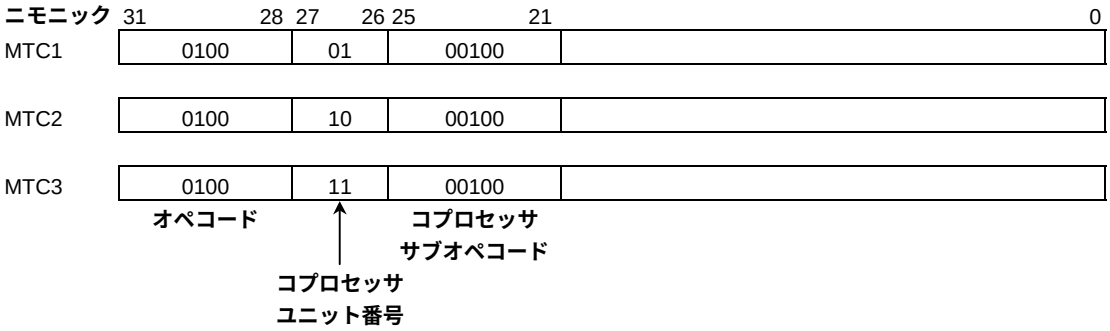
動作

rt ⇐ コプロセッサユニット *z* のコプロセッサレジスタ *rd*

コード



オペコードのビットコードを以下に示します。オペコードフィールドの下位 2 ビットはコプロセッサユニット番号を表わします。



説明

汎用レジスタ *rt* の内容をコプロセッサユニット *z* のコプロセッサレジスタ *rd* にロードします。

例外

コプロセッサ使用不可例外

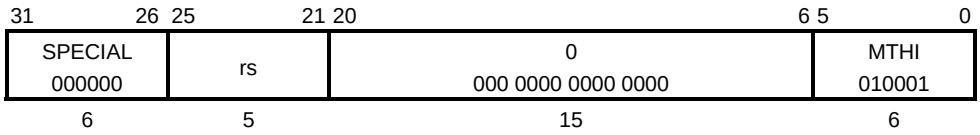
MTHI *rs*

Move To HI

動作

HI ⇐ *rs*

コード



説明

汎用レジスタ *rs* の内容を HI レジスタにロードします。

例外

なし

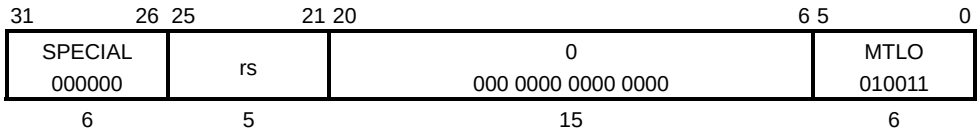
MTLO *rs*

Move To LO

動作

LO ⇐ *rs*

コード



説明

汎用レジスタ *rs* の内容を LO レジスタにロードします。

例外

なし

MULT (*rd*,) *rs*, *rt*

Multiply

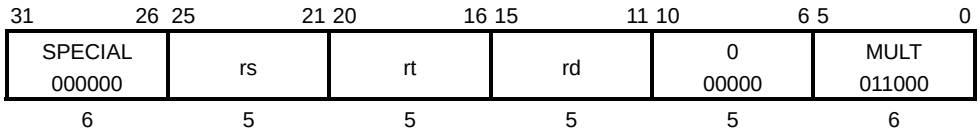
動作

HI ⇐ (*rs* × *rt*) の上位ワード;

LO ⇐ (*rs* × *rt*) の下位ワード;

rd ⇐ (*rs* × *rt*) の下位ワード

コード



説明

汎用レジスタ *rs* の内容と汎用レジスタ *rt* の内容を乗算します。*rs* と *rt* は符号付き整数として扱います。結果の上位ワードを HI レジスタに格納し、下位ワードを LO レジスタに格納します。デスティネーションレジスタ *rd* が指定されている場合、結果の下位ワードを *rd* にも格納します。

rd を省略すると、デフォルトは r0 になり、その結果、汎用レジスタに移送された下位ワードは廃棄されます。

いかなる場合も、整数オーバフロー例外は発生しません。

例外

なし

使用例

汎用レジスタ r2 に 0x0123_4567 が、r3 に 0x89AB_CDEF が格納されているとします。

```
MULT r4,r2,r3
```

このとき、上記の命令は、以下の演算を実行します。

(0x0123_4567 × 0x89AB_CDEF)
= 0xFF79_5E36_C94E_4629

結果の上位ワード 0xFF79_5E36 が HI レジスタに格納され、下位ワード 0xC94E_4629 が LO レジスタと r4 に格納されます。

MULTU (*rd*,) *rs*, *rt*

Multiply Unsigned

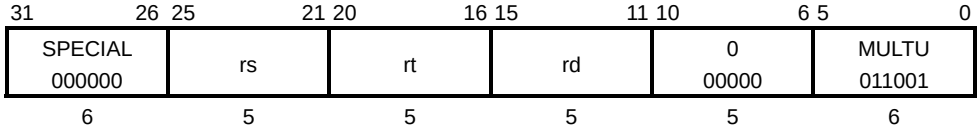
動作

HI ⇐ (*rs* × *rt*) の上位ワード;

LO ⇐ (*rs* × *rt*) の下位ワード;

rd ⇐ (*rs* × *rt*) の下位ワード

コード



説明

汎用レジスタ *rs* の内容と汎用レジスタ *rt* の内容を乗算します。*rs* と *rt* は符号なし整数として扱います。結果の上位ワードを HI レジスタに格納し、下位ワードを LO レジスタに格納します。デスティネーションレジスタ *rd* が指定されている場合、結果の下位ワードを *rd* にも格納します。

rd を省略すると、デフォルトは r0 となり、その結果、汎用レジスタに移送された下位ワードは廃棄されます。

いかなる場合も、整数オーバフロー例外は発生しません。

例外

なし

使用例

汎用レジスタ r2 に 0x0123_4567 が、r3 に 0x89AB_CDEF が格納されているとします。

```
MULTU r4,r2,r3
```

このとき、上記の命令は、以下の演算を実行します。

(0x0123_4567 × 0x89AB_CDEF)
= 0x009C_A39D_C94E_4629

結果の上位ワード 0x009C_A39D が HI レジスタに格納され、下位ワード 0xC94E_4629 が LO レジスタと r4 に格納されます。

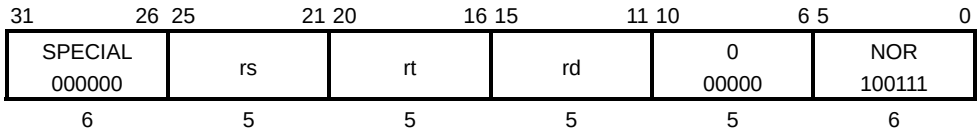
NOR *rd*, *rs*, *rt*

NOR

動作

rd ⇐ *rs* NOR *rt*

コード



説明

汎用レジスタ *rs* の内容と汎用レジスタ *rt* の内容の否定論理和 (NOR) をとり、結果を汎用レジスタ *rd* に格納します。

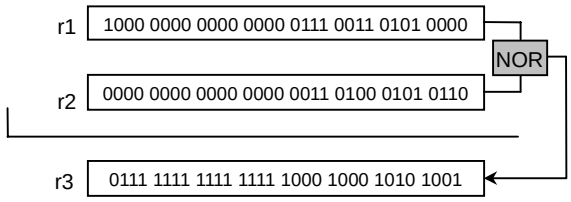
例外

なし

使用例

レジスタ r2 の値が 0x8000_7350 で、r3 の値が 0x0000_3456 のとき、以下の命令を実行すると、図示したように、r2 と r3 の否定論理和 0x7FFF_88A9 がレジスタ r4 に格納されます。

```
NOR r4,r2,r3
```



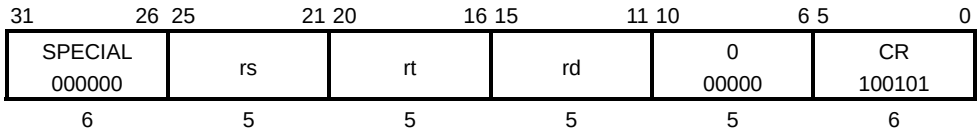
OR *rd*, *rs*, *rt*

OR

動作

rd ⇐ *rs* OR *rt*

コード



説明

汎用レジスタ *rs* の内容と汎用レジスタ *rt* の内容の論理和 (OR) をとり、結果を汎用レジスタ *rd* に格納します。

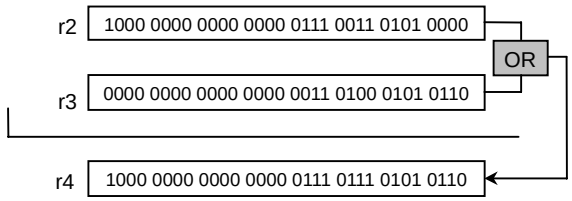
例外

なし

使用例

レジスタ r2 の値が 0x8000_7350 で、r3 の値が 0x0000_3456 のとき、以下の命令を実行すると、図示したように r2 と r3 の論理和 0x8000_7756 がレジスタ r4 に格納されます。

```
OR r4,r2,r3
```



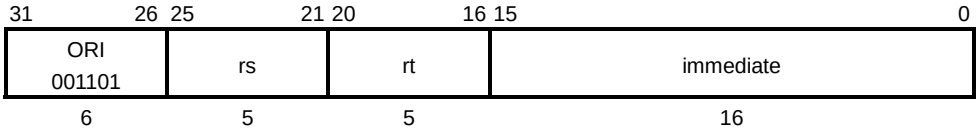
ORI *rt*, *rs*, *immediate*

OR Immediate

動作

rt ⇐ *rs* OR *immediate*

コード



説明

16 ビット *immediate* をゼロ拡張した値と汎用レジスタ *rs* の内容との論理和 (OR) をとり、結果を汎用レジスタ *rt* に格納します。

immediate フィールドは 16 ビット長です。これを超える値を扱いたい場合は、いったん汎用レジスタに格納してから、OR 命令を使います (「3.3.2　32 ビットの定数」を参照)。

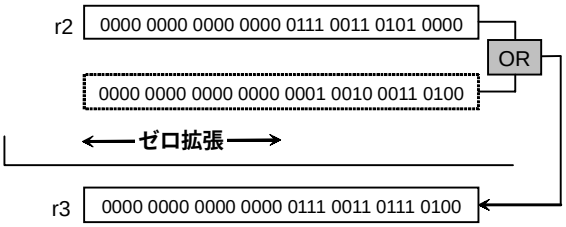
例外

なし

使用例

レジスタ r2 の値が 0x0000_7350 の場合、以下の命令を実行すると、図示したように、0x0000_7350 と 0x0000_1234 の論理和 0x0000_7374 がレジスタ r3 に格納されます。

```
ORI r3,r2,0x1234
```



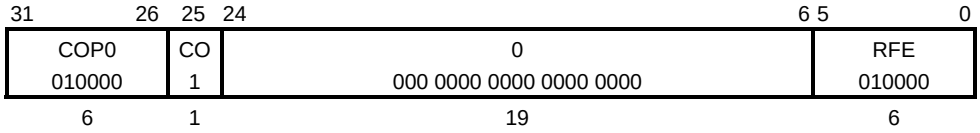
RFE

Restore From Exception

動作

Status ⇐ Status[31:16] ‖ Status[18:16] ‖ Status[12:4] ‖ Status[5:2]

コード



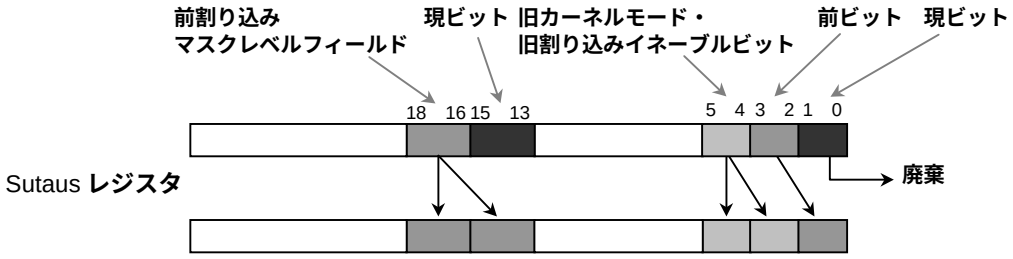
説明

RFE は例外からの復帰命令です。Status レジスタに設定されいてるプロセッサの状態を、例外が発生するまえの状態に復元します。旧カーネルモードビットと旧割り込み許可ビット (KUo・IEo) を、前ビット (KUp・IEp) に復元しません。前ビット (KUp・IEp) を、現ビット (KUc/IEc) に復元します。旧ビット (KUo・IEo) は変更されません。さらに、前割り込みマスクレベルフィールド PMask[2:0]を、現フィールド CMask[2:0] に復元します。PMask[2:0]フィールドは変更されません。

通常、RFE 命令は、例外から戻るときに JR 命令のジャンプ遅延スロットで使いますが、それ以外のところでも使用できます。

MTC0 命令で Status レジスタに書き込みを行った直後、または MFC0 命令で Status レジスタから読み出しを行った直後に RFE 命令を実行することは禁止されています。実行した場合、Status レジスタの値は不定になります。

RFE 命令の実行中に割り込みが発生すると、Status レジスタの値は保証できません。したがって、RFE 命令を実行するまえに、すべての割り込みを禁止しなければなりません。



例外

コプロセッサ使用不可例外

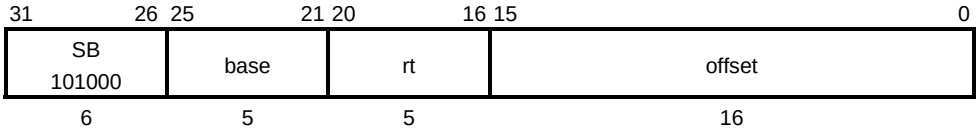
SB *rt*, *offset* (*base*)

Store Byte

動作

rt ⇒ {*offset* (*base*)}

コード



説明

16 ビット *offset* を符号拡張した値と汎用レジスタ *base* の内容を加算することにより、実効アドレス (EA) を生成します。汎用レジスタ *rt* の最下位バイトを、このアドレスにストアします。

rt の上位 3 バイトは無視されるため、符号付きと符号なしの区別がなくなります。

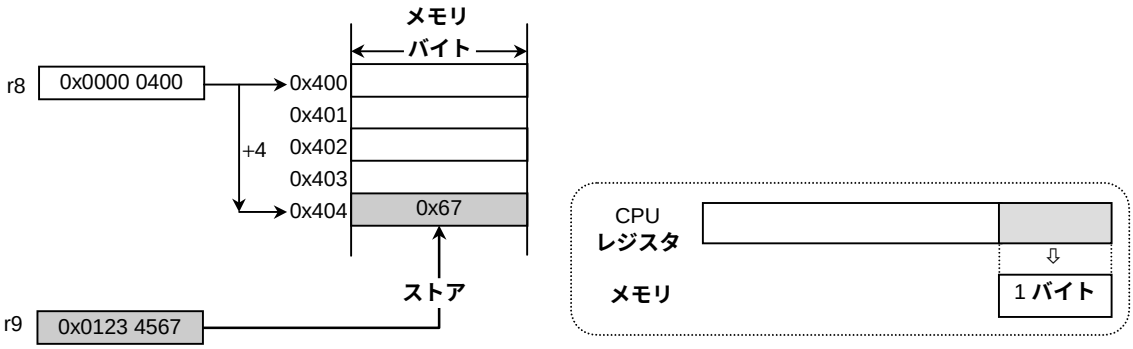
例外

アドレスエラー例外

使用例

レジスタ r8 の値が 0x0000_0400 で、r9 の値が 0x0123_4567 の場合、以下の命令を実行すると、0x67 がアドレス 0x404 に格納されます。

```
SB r9,4(r8)
```



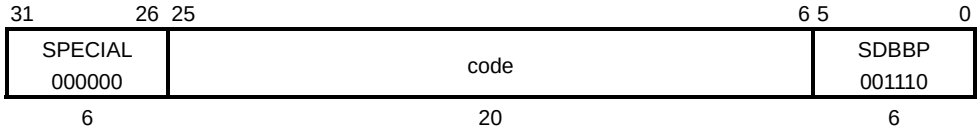
SDBBP *code*

Software Debug Breakpoint Exception

動作

ソフトウエアデバッグブレークポイント例外

コード



説明

デバッグブレークポイントが発生し、無条件に制御を例外ハンドラに移します。

SDBBP 命令の *code* フィールドは、例外ハンドラに情報を渡すために使用できます。例外ハンドラが code フィールドを取り出すには、命令を含むメモリワードの内容をデータとしてロードする必要があります。詳細は「9.3　デバッグ例外」を参照してください。

デバッグ例外の処理中 (Debug レジスタの DM ビットが 1) は、SDBBP 命令は使用しないでください。DM=1 のとき、SDBBP 命令の動作は不定です。

SDBBP 命令は、開発システムで使用するので、ユーザープログラム中では使用しないでください。

例外

デバッグブレークポイント例外

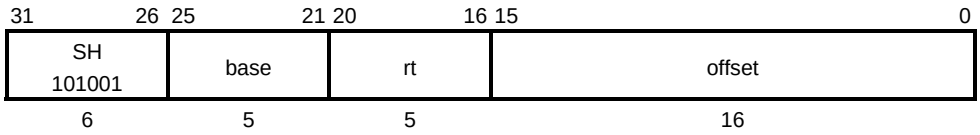
SH *rt*, *offset* (*base*)

Store Halfword

動作

rt ⇒ {*offset* (*base*)}

コード



説明

16 ビット *offset* を符号拡張して、汎用レジスタ *base* の内容に加算することにより、実効アドレス (EA) を生成します。汎用レジスタ *rt* の下位ハーフワードをこのアドレスにストアします。

rt の上位ハーフワードは無視されるので、符号付き、符号なしの区別はありません。

実効アドレスの最下位ビットが 0 でない (実効アドレスがハーフワード境界でない) 場合、アドレスエラー例外が発生します。

例外

アドレスエラー例外

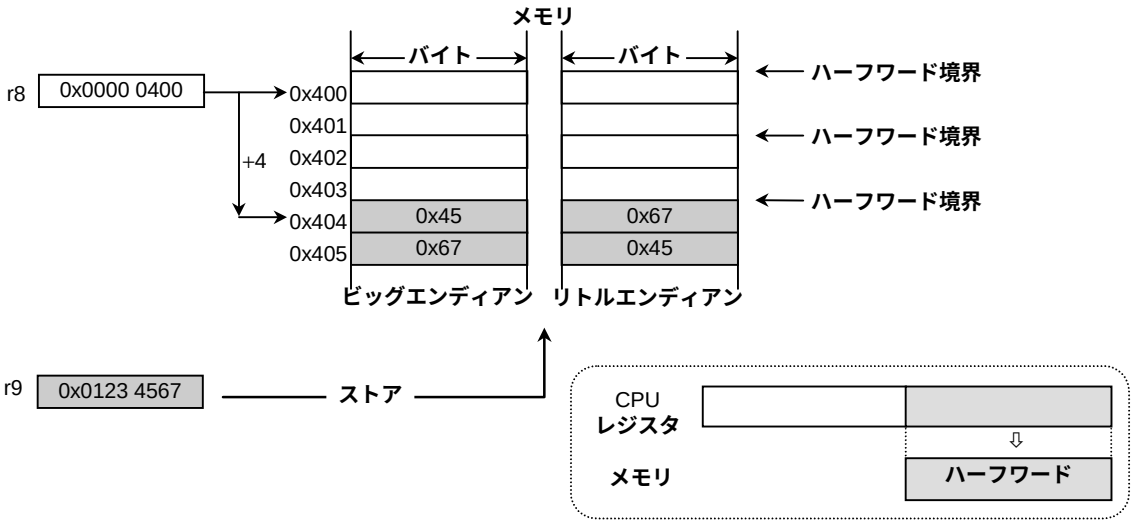
使用例

レジスタ r8 の値が 0x0000_0400 で、r9 の値が 0x0123_4567 の場合、以下の命令を実行すると、ビッグエンディアンのときは、アドレス 0x404 に 0x45 が、アドレス 0x405 に 0x67 がストアされます。リトルエンディアンのときは、アドレス 0x404 に 0x67 が、アドレス 0x405 に 0x45 がストアされます。

```
SH r9,4(r8)
```

また、以下の命令を実行すると、0x403 はハーフワード境界でないので、アドレスエラー例外が発生します。

```
SH r9,3(r8)
```


メモリ
バイト
バイト
ハーフワード境界
ハーフワード境界
ハーフワード境界
r8
0x0000 0400
+4
0x400
0x401
0x402
0x403
0x404
0x405
0x45
0x67
0x67
0x45
ビッグエンディアン
リトルエンディアン
r9
0x0123 4567
ストア
CPU
レジスタ
メモリ
ハーフワード

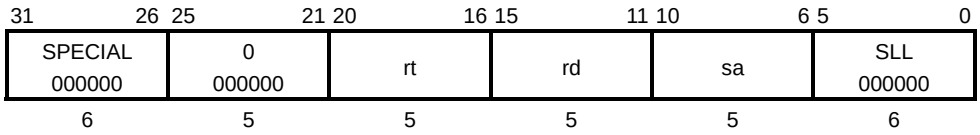
SLL *rd*, *rt*, *sa*

Shift Left Logical

動作

rd ⇐ *rt* << *sa*

コード



説明

汎用レジスタ *rt* の 32 ビットの内容を *sa* ビット左へシフトし、右端の空いたビットを 0 で埋め、結果を汎用レジスタ *rd* に格納します。

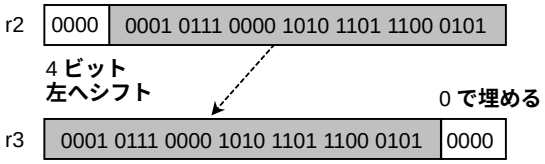
例外

なし

使用例

レジスタ r2 の内容が 0x2170_ADC5 の場合、以下の命令を実行すると、レジスタ r3 に 0x170A_DC50 が格納されます。

```
SLL r3,r2,4
```



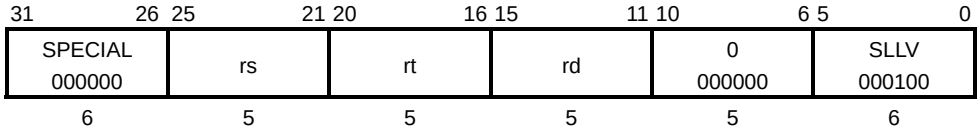
SLLV *rd*, *rt*, *rs*

Shift Left Logical Variable

動作

rd ⇐ *rt* << の下位 5 ビット

コード



説明

汎用レジスタ *rt* の 32 ビットの内容を、汎用レジスタ *rs* の下位 5 ビットで指定されたビット数、左にシフトし、右端の空いたビットをゼロで埋めます。結果を汎用レジスタ *rd* に格納します。

例外

なし

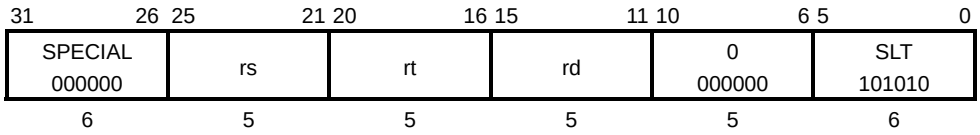
SLT *rd*, *rs*, *rt*

Set On Less Than

動作

if *rs* < *rt* then *rd* ⇐ 1; else *rd* ⇐ 0

コード



説明

汎用レジスタ *rs* の内容と汎用レジスタ *rt* の内容を符号付き整数として比較します。*rs* が *rt* より小さい場合は、汎用レジスタ *rd* は 1 を、そうでない場合は、*rd* は 0 を格納します。

いかなる場合も、整数オーバフロー例外は発生しません。比較の際に行われる減算でオーバフローしても、比較は有効です。

例外

なし

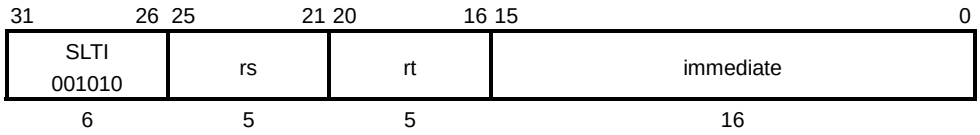
SLTI *rt*, *rs*, *immediate*

Set On Less Than Immediate

動作

if *rs* < *immediate* then *rt* ⇐ 1; else *rt* ⇐ 0

コード



説明

16 ビット *immediate* を符号拡張した値と汎用レジスタ *rs* の内容を符号付き整数として比較します。*rs* が *immediate* より小さい場合は、汎用レジスタ *rt* に 1 を、そうでない場合は、0 を格納します。

いかなる場合も、整数オーバフロー例外は発生しません。比較の際に行われる減算でオーバフローしても、比較は有効です。

immediate フィールドは 16 ビットです。したがって *immediate* で指定できる数値範囲は、-32768 ~ +32767 です。この範囲外の値を扱いたい場合は、いったん汎用レジスタに格納してから、SLT 命令を使います (「3.3.2　32 ビットの定数」を参照)。

例外

なし

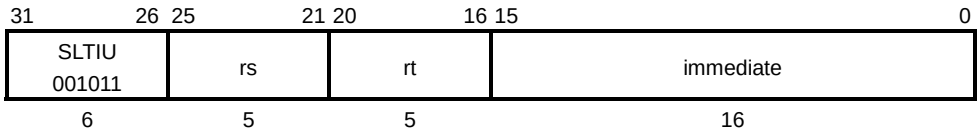
SLTIU *rt*, *rs*, *immediate*

Set On Less Than Immediate Unsigned

動作

if *rs* < *immediate* then *rt* ⇐ 1; else *rt* ⇐ 0

コード



説明

16 ビット *immediate* を符号拡張した値と汎用レジスタ *rs* の内容を符号なし整数として比較します。*rs* が *immediate* より小さい場合は、汎用レジスタ *rt* を 1 に、そうでない場合は、0 を格納します。

いかなる場合も、整数オーバフロー例外は発生しません。比較の際に行われる減算でオーバフローしても、比較は有効です。

immediate フィールドは 16 ビットです。この範囲外の場合は、いったん汎用レジスタに格納してから、SLTU 命令を使います (「3.3.2　32 ビットの定数」を参照)。

例外

なし

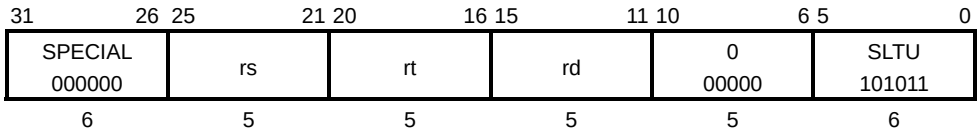
SLTU *rd*, *rs*, *rt*

Set On Less Than Unsigned

動作

if *rs* < *rt* then *rd* ⇐ 1; else *rd* ⇐ 0

コード



説明

汎用レジスタ *rs* の内容と汎用レジスタ *rt* の内容を符号なし整数として比較します。*rs* が *rt* より小さい場合、汎用レジスタ *rd* に 1 を、そうでない場合は、0 を格納します。

いかなる場合も、整数オーバフロー例外は発生しません。比較の際に行われる減算でオーバフローしても、比較は有効です。

例外

なし

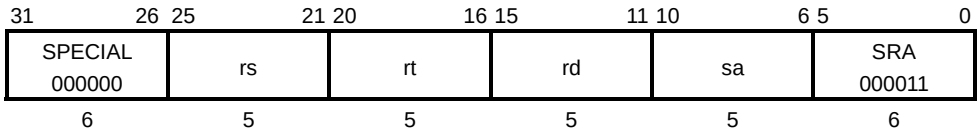
SRA *rd*, *rt*, *sa*

Shift Right Arithmetic

Operand

rd ⇐ *rt* >> *sa*

コード



説明

汎用レジスタ *rt* の 32 ビットの内容を *sa* ビット右へシフトし、左端の空いたビットを符号ビットで埋めます。結果を汎用レジスタ *rd* に格納します。

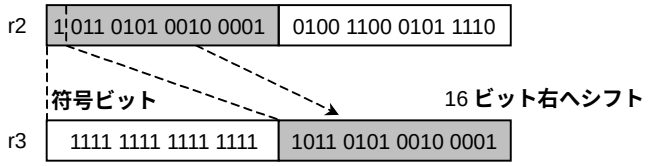
例外

なし

使用例

レジスタ r2 の値が 0xB521_4C5E の場合、以下の命令を実行すると、レジスタ r3 に 0xFFFF_B521 が格納されます。

```
SRA r3,r2,16
```



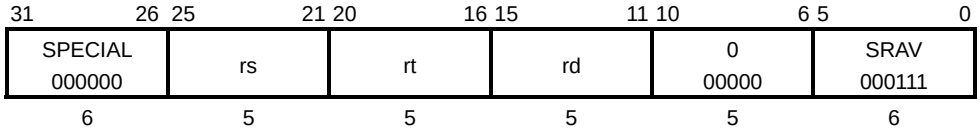
SRAV *rd*, *rt*, *rs*

Shift Right Arithmetic Variable

動作

rd ⇐ *rt* >> の下位 5 ビット

コード



説明

汎用レジスタ *rt* の 32 ビットの内容を、汎用レジスタ *rs* の下位 5 ビットで指定されたビット数、右にシフトし、左端の空いたビットを符号ビットで埋めます。結果を汎用レジスタ *rd* に格納します。

例外

なし

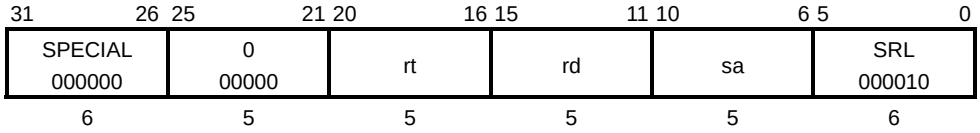
SRL *rd*, *rt*, *sa*

Shift Right Logical

動作

rd ⇐ *rt* >> *sa*

コード



説明

汎用レジスタ *rt* の 32 ビットの内容を *sa* ビット右へシフトし、左端の空いたビットを 0 で埋めます。結果を汎用レジスタ *rd* に格納します。

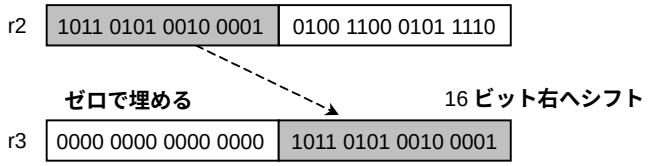
例外

なし

使用例

レジスタ r2 の内容が 0xB521_4C5E の場合、以下の命令を実行すると、レジスタ r3 に 0x0000_B521 が格納されます。

```
SRL r3,r2,16
```



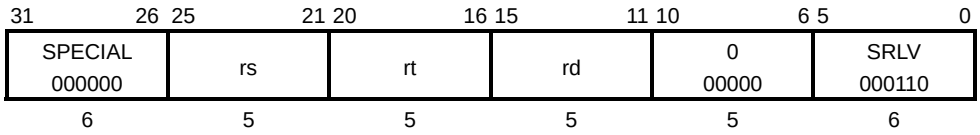
SRLV *rd*, *rt*, *rs*

Shift Right Logical Variable

動作

rd ⇐ *rt* >> の下位 5 ビット

コード



説明

汎用レジスタ *rt* の 32 ビットの内容を、汎用レジスタ *rs* の下位 5 ビットで指定されたビット数、右にシフトし、左端の空いたビットを 0 で埋めます。結果を汎用レジスタ *rd* に格納します。

例外

なし

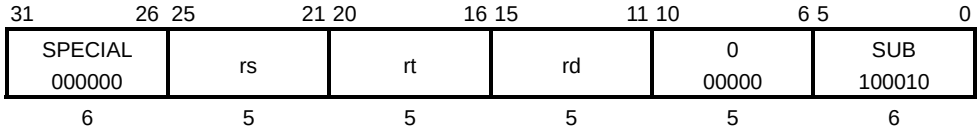
SUB *rd*, *rs*, *rt*

Subtract

動作

$rd \Leftarrow rs - rt$

コード



説明

汎用レジスタ *rs* の内容から汎用レジスタ *rt* の内容を減算し、減算結果を汎用レジスタ *rd* に格納します。*rs* と *rt* は符号付き整数として扱されます。

2 の補数のオーバフローが発生した場合、整数オーバフロー例外が発生します。オーバフローは、c ⇐ a – b の計算において a と b の符号が異なり、かつ a と c の符号が異なる場合に発生します。整数オーバフロー例外が発生すると、デスティネーションレジスタ (*rd*) の内容は変更されません。

例外

整数オーバフロー例外

使用例

1. レジスタ r2 の値が 0x7654_3210 で、レジスタ r3 の値が 0x5000_0000 のとき、以下の命令を実行すると、減数結果 (0x2654_3210) が r4 に格納されます。

```
SUB r4,r2,r3
```

2. レジスタ r2 の値が 0x7FFF_FFFF で、レジスタ r3 の値が 0x8FFF_FFFF のとき、r2 から r3 を減算すると、結果は 0xF000_0000 になります。つまり、r2 と r3 の符号が異なり、r2 の符号と減算結果の符号が異なっているため 2 の補数のオーバフローが発生します。

```
SUB r4,r2,r3
```

このとき、上記の命令を実行すると、整数オーバフロー例外が発生します。レジスタ r4 は変更されません。

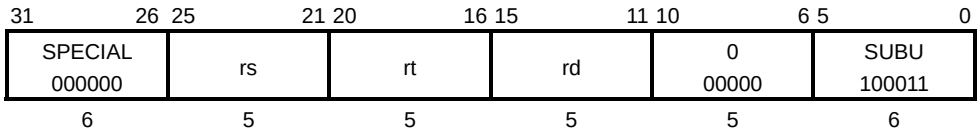
SUBU *rd*, *rs*, *rt*

Subtract Unsigned

動作

$rd \Leftarrow rs - rt$

コード



説明

汎用レジスタ *rs* の内容から汎用レジスタ *rt* の内容を減算し、減算結果を汎用ジレスタ *rd* に格納します。

SUB 命令と SUBU 命令の唯一の違いは、SUBU 命令では整数オーバフロー例外が発生しないという点だけです。

例外

なし

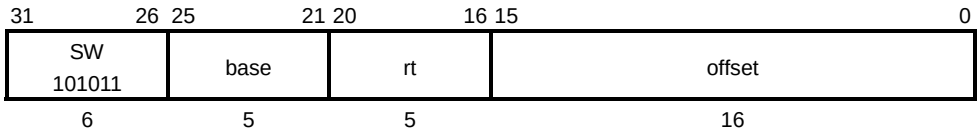
SW *rt*, *offset* (*base*)

Store Word

動作

rt ⇒ {*offset* (*base*)}

コード



説明

16 ビット *offset* を符号拡張して、汎用レジスタ *base* の内容に加算することにより、実効アドレス (EA) を生成します。汎用レジスタ *rt* の内容をこのアドレスにストアします。

実効アドレスの下位 2 ビットが 0 でない (実効アドレスが境界でない) 場合、アドレスエラー例外が発生します。

例外

アドレスエラー例外

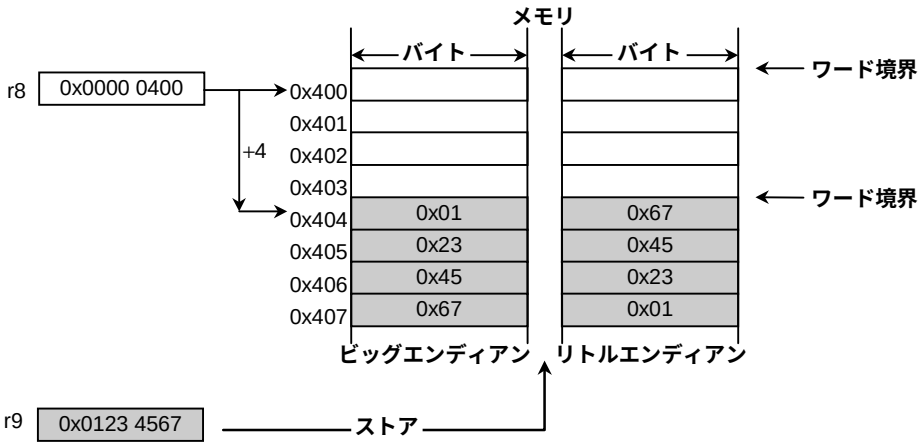
使用例

レジスタ r8 の値が 0x0000_0400 で、レジスタ r9 の値が 0x0123_4567 の場合、以下の命令を実行すると、ビッグエンディアンのときは、アドレス 0x404~0x407 に 0x12、0x23、0x45、0x67 がストアされます。リトルエンディアンのときは、アドレス 0x404~0x407 に 0x67、0x45、0x23、0x01 がストアされます。

```
SW r9,4(r8)
```

また、以下の命令を実行すると、0x405 はハーフワード境界でないので、アドレスエラー例外が発生します。

```
SW r9,5(r8)
```


メモリ
バイト
バイト
ワード境界
r8
0x0000 0400
0x400
0x401
0x402
0x403
+4
ワード境界
0x404
0x405
0x406
0x407
0x01
0x23
0x45
0x67
0x67
0x45
0x23
0x01
ビッグエンディアン
リトルエンディアン
r9
0x0123 4567
ストア

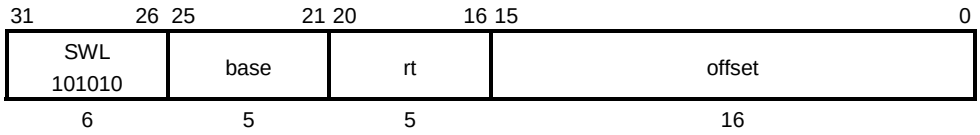
SWL *rt*, *offset* (*base*)

Store Word Left

動作

rt ⇒ {*offset* (*base*)}

コード



説明

16 ビット *offset* を符号拡張し、汎用レジスタ *base* の内容に加算することにより実効アドレス (EA) を生成します。汎用レジスタ *rt* の左部分を、ワード境界をまたがったワード位置の上位にストアします。

実効アドレスがワード境界に位置いていないことによるアドレスエラー例外は発生しません。

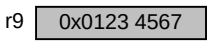
レジスタ中のワードデータをワード境界をまたがってメモリにストアするときに、SWL 命令と SWR 命令を組み合わせて使います。

例外

アドレスエラー例外

使用例

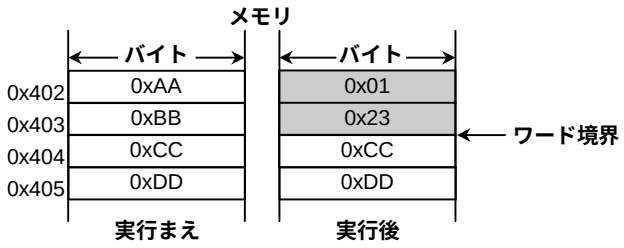
レジスタ r8 の値が 0x0000_0400 で、レジスタ r9 の値が 00123_4567 であるとします。



- ビッグエンディアンのとき

```
SWL r9,2(r8)
```

上記の命令は、レジスタ r9 の左部分をアドレス 0x0402 から上位アドレス方向へ、ワード境界に達するまでストアします。その結果を以下に示します。

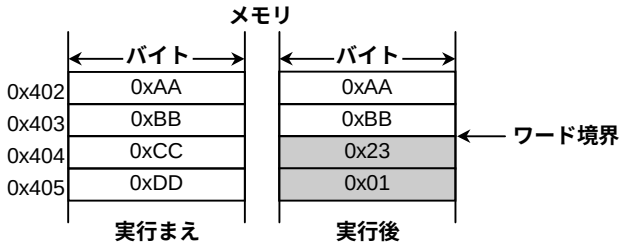


(a) ビッグエンディアン

- リトルエンディアンのとき

```
SWL r9,5(r8)
```

上記の命令は、レジスタ r9 の左部分をアドレス 0x0405 から下位アドレス方向へ、ワード境界に達するまでストアします。その結果を以下に示します。



(b) リトルエンディアン

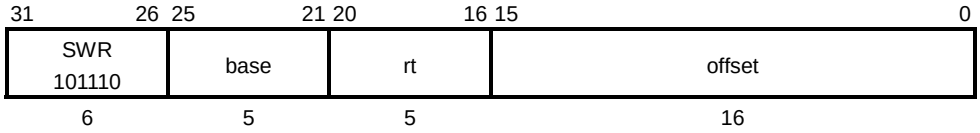
SWR *rt*, *offset* (*base*)

Store Word Right

動作

rt ⇒ {*offset* (*base*)}

コード



説明

16 ビット *offset* を符号拡張し、汎用レジスタ *base* の内容に加算することにより実効アドレス (EA) を生成します。汎用レジスタ *rt* の右部分を、ワード境界をまたがったワード位置の下位にストアします。

ワード境界に位置いていないことによるアドレスエラー例外は発生しません。

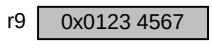
実効アドレスがレジスタ中のワードデータをワード境界をまたがってメモリにストアするときに、SWL 命令と SWR 命令を組み合わせて使います。

例外

アドレスエラー例外

使用例

レジスタ r9 の値が 0x123_4567 とします。

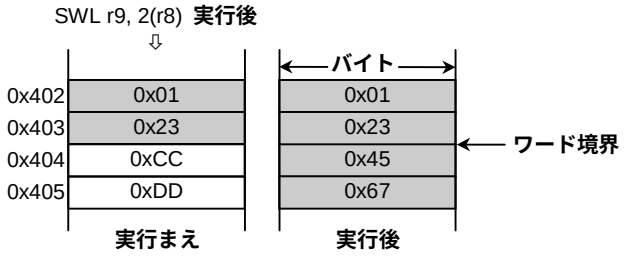


前ページの SWL 命令で説明したように、汎用レジスタの左部分の内容をストアした後に、右部分をストアする方法を以下に示します。

- ビッグエンディアン

```
SWR r9,5(r8)
```

上記の命令は、レジスタ r9 の右部分をアドレス 0x0405 から下位アドレス方向へ、ワード境界に達するまでストアします。その結果を以下に示します。

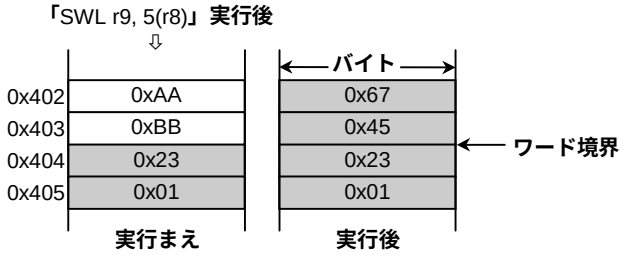


(a) ビッグエンディアン

- リトルエンディアン

```
SWR r9,2(r8)
```

上記の命令は、レジスタ r9 の右部分をアドレス 0x0402 から上位アドレス方向へ、ワード境界に達するまでストアします。その結果を以下に示します。



(b) リトルエンディアン

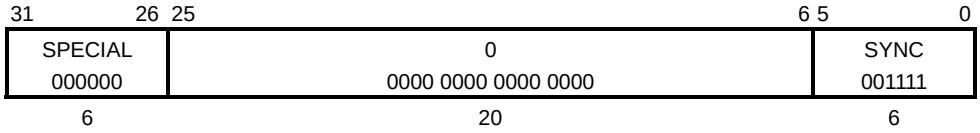
SYNC

Synchronize

動作

メモリ同期操作

コード



説明

SYNC 命令は、SYNC 命令の直前に実行したロード、ストアが完了するまで、命令パイプラインをインタロックし、後続の命令の実行を遅らせます。これにより、SYNC 命令のまえの命令と後続の命令の実行順序を守ることができます。「5.2.4　SYNC 命令 (32 ビット ISA)」を参照してください。

ロード命令の後続命令が、そのロード結果を使用しない場合、パイプラインをストールせず実行が継続されます。この機能をノンブロッキングロードといいます。パイプラインの他の部分は、データと依存関係のない命令の実行を継続します。

SYNC 命令は、ユーザーモードで使用できます。

例外

なし

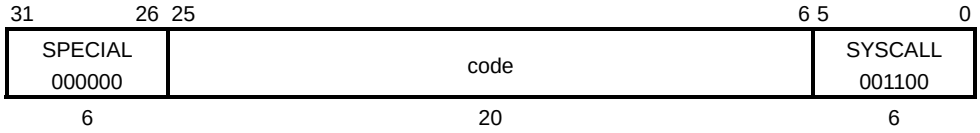
SYSCALL *code*

System Call

動作

システムコール例外

コード



説明

SYSCALL 命令を実行すると、システムコール例外が発生し、無条件に制御を例外ハンドラに渡します。

SYSCALL 命令の *code* フィールドを使用して、例外ハンドラにパラメータを送ることができます。これらのビットを調べるには、EPC レジスタが示す命令の内容をロードします。システムコール例外の詳細は、「9.1.10　システムコール例外」を参照してください。

例外

システムコール例外

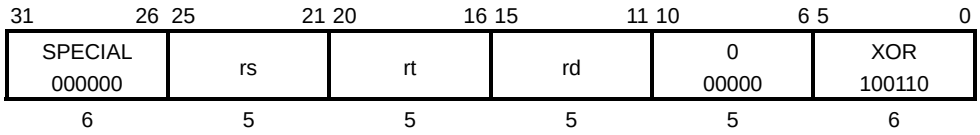
XOR *rd*, *rs*, *rt*

Exclusive OR

動作

rd ⇐ *rs* XOR *rt*

コード



説明

汎用レジスタ *rs* の内容と汎用レジスタ *rt* の内容との排他的論理和 (XOR) をとり、結果を汎用レジスタ *rd* に格納します。

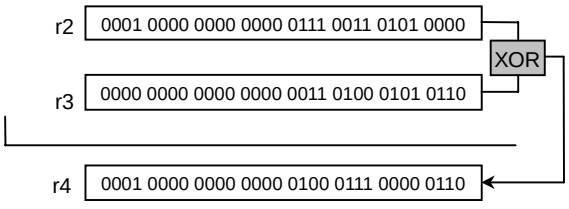
例外

なし

使用例

レジスタ r2 の値が 0x1000_7350 で、レジスタ r3 の値が 0x0000_3456 のとき、以下の命令を実行すると、結果の 0x1000_4706 がレジスタ r4 に格納されます。

```
XOR r4,r2,r3
```



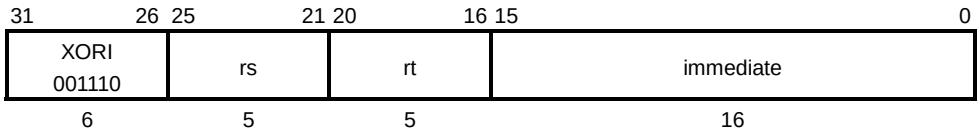
XORI *rt*, *rs*, *immediate*

Exclusive OR Immediate

動作

rt ⇐ *rs* XOR *immediate*

コード



説明

16 ビット *immediate* をゼロ拡張し、汎用レジスタ *rs* の内容との排他的論理和 (XOR) をとり、結果を汎用レジスタ *rt* に格納します。

immediate フィールドは 16 ビットです。これを超える値はいったん汎用レジスタに格納し、XOR 命令を使う必要があります (「3.3.2　32 ビットの定数」を参照)。

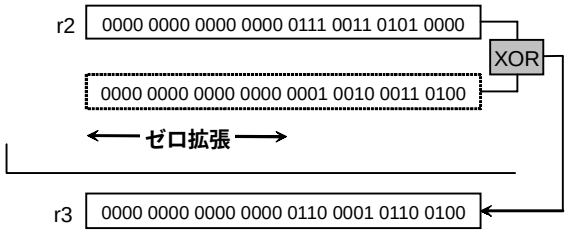
例外

なし

使用例

レジスタ r2 の値が 0x0000_7350 の場合、以下の命令を実行すると、結果の 0x0000_6164 がレジスタ r3 に格納されます。

```
XORI r3,r2,0x1234
```



付録B　16 ビット ISA の詳細

この章では、16 ビット ISA モードの命令について、シンタックス、命令形式、動作、命令の実行によって発生する可能性のある例外などを詳しく説明しています。命令はアルファベット順に記述されています。命令形式については「4.1　命令形式」を参照してください。

16 ビット ISA の命令は、32 ビットの JAL、JALX 命令を除き、すべて 16 ビットです。基本的に、各 16 ビット命令は 32 ビット命令に対応しており、メモリからフェッチされた 16 ビット命令は、MIPS16 伸長回路により、32 ビット命令に変換されてから、通常の命令デコーダに送り込まれます。ただし、16 ビット命令と 32 ビット命令では、機能が異なる命令がいくつかあります。各命令の説明では、伸長前と伸長後の命令コードを対比して示してあります。

16 ビット ISA の命令では各レジスタフィールド (*rx*、*ry*、*rz*、*base*) は 3 ビットしかなく、32 本の汎用レジスタのうち、r2~r7、r16、r17 の 8 本のレジスタしか使用できません。16 ビット ISA におけるレジスタのコードを以下に示します。

コード	レジスタ	コード	レジスタ
000	r16	100	r4
001	r17	101	r5
010	r2	110	r6
011	r3	111	r7

ただし、命令によっては、r24 (t8)、r29 (sp)、r31 (ra) を使用しています。r24 は比較結果を格納するコンディションコードレジスタです。r29 はスタックポインタレジスタです。r31 はサブルーチンの戻りアドレスを格納するリンクレジスタです。これらのレジスタは、命令により暗黙的に使用されます。

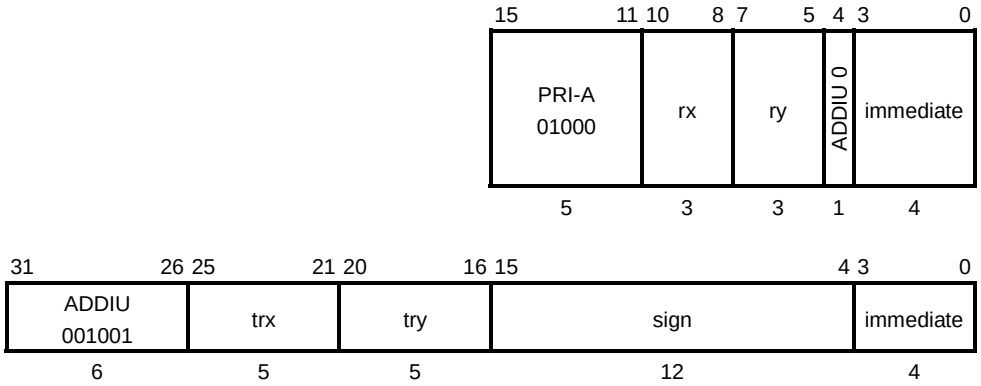
ADDIU *ry*, *rx*, *immediate*

Add Immediate Unsigned

動作

ry ⇐ *rx* + *immediate*

コード

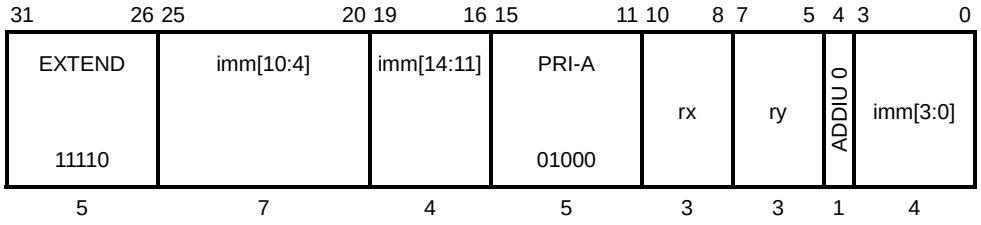


説明

ADDIU は Add Immediate Unsigned を表しますが、4 ビット *immediate* を「符号拡張」して、汎用レジスタ *rx* の内容に加算し、その結果を汎用レジスタ *ry* に格納します。

整数オーバフロー例外は絶対に発生しません。

immediate フィールドは 4 ビットで、扱うことのできる数値の範囲は、–8 ~ +7 です。*immediate* がこの範囲外の場合、EXTEND に命令が自動的に付加されます。これにより、ALU 即値命令の即値フィールドは、16 ビットに拡張されます。ただし、ADDIU 命令の即値フィールドは 4 ビットしかなく、EXTEND 命令の即値フィールドは 11 ビットしかないので、ADDIU 命令は拡張しても扱える即値は 15 ビットまでです。したがって、*immediate* で指定できる数値の範囲は、–16384 ~ +16833 になります。EXTEND により拡張された命令コードを以下に示します。



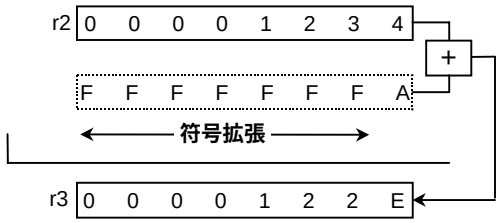
例外

なし

使用例

レジスタ r2 の値が 0x0000_1234 の場合、以下の命令を実行すると、図示したように、和 (0x0000_122E) がレジスタ r3 に格納されます。

```
ADDIU r3,r2,-6
```



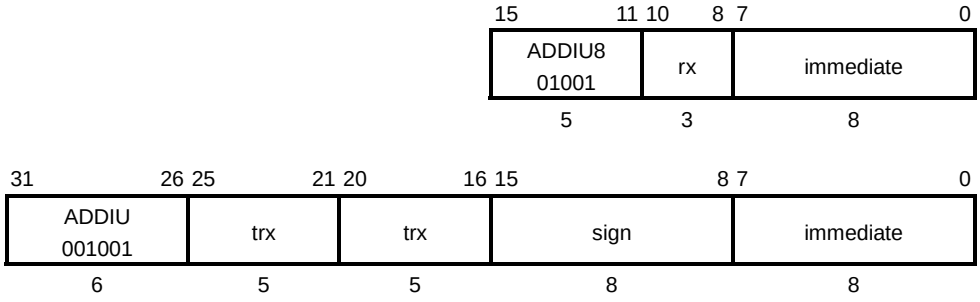
ADDIU *rx, immediate*

Add Immediate Unsigned

動作

rx ⇐ *rx* + *immediate*

コード



説明

ADDIU は Add　Immediate Unsigned を表しますが、8 ビット *immediate* を「符号拡張」して、汎用レジスタ *rx* の内容に加算し、その結果を汎用レジスタ *rx* に格納します。

整数オーバフロー例外は絶対に発生しません。

immediate フィールドは、8 ビットで、扱えることのできる数値の範囲は、–128 ~ +127 です。この範囲外の値を指定すると、ADDIU 命令は EXTEND 命令により拡張され、符号付きの 16 ビットの即値 (–32768 ~ +32767) を扱えるようになります。

例外

なし

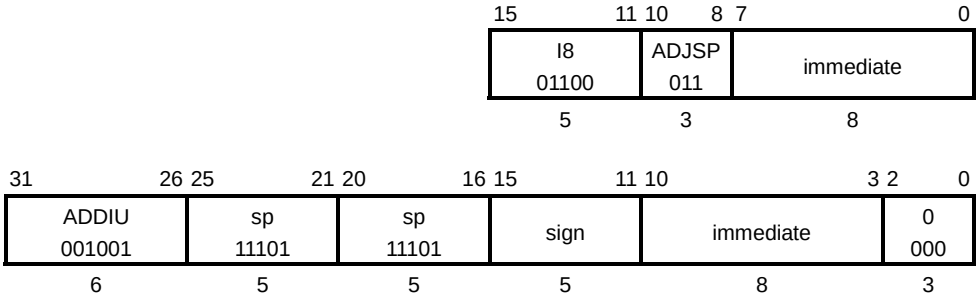
ADDIU sp, *immediate*

Add Immediate Unsigned

動作

sp ⇐ sp + *immediate*

コード



説明

ADDIU は Add Immediate Unsigned を表しますが、8 ビット *immediate* を 3 ビット左へシフトし、「符号拡張」します。その結果をスタックポインタレジスタ sp (r29) の内容に加算します。

整数オーバフロー例外は絶対に発生しません。

immediate フィールドは 8 ビットで、3 ビットシフトすることにより、扱うことのできる数値の範囲は、8 刻みで–1024 ~ +1016 になります。この範囲外の値を指定すると、ADDIU 命令は EXTEND 命令により拡張され、符号付きの 16 ビットの即値 (–32768 ~ +32767) を扱えるようになります。この場合、*immediate* はシフトされません。

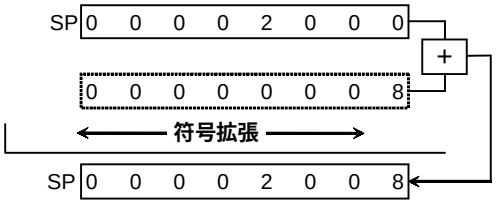
例外

なし

使用例

スタックポインタレジスタ sp の値が 0x0000_2000 の場合、以下の命令を実行すると、図示したように、結果 0x0000_2008 がレジスタ sp に格納されます。

```
ADDIU sp,8
```


SP 0 0 0 0 2 0 0 0
+
0 0 0 0 0 0 0 8
符号拡張
SP 0 0 0 0 2 0 0 8

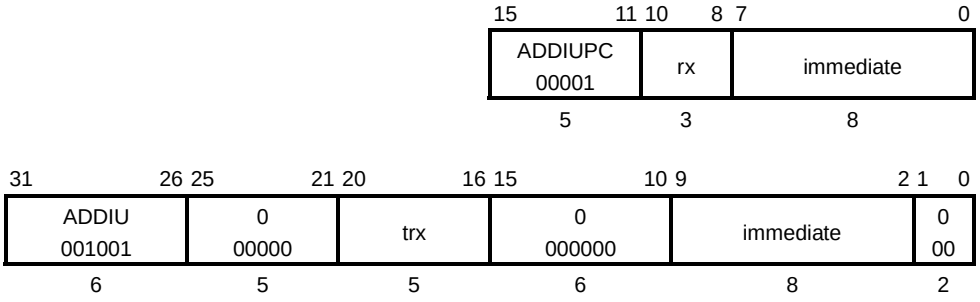
ADDIU *rx*, pc, *immediate*

Add Immediate Unsigned

動作

rx ⇐ マスクベース PC + *immediate*

コード



説明

アドレス計算のベースとして使用される PC 値を、ベース PC 値といい、ベース PC 値の下位 2 ビットを 0 にマスクした値を、マスクベース PC 値といいます。この命令は、8 ビット *immediate* を 2 ビット左へシフトし、ゼロ拡張します。その結果をマスクベース PC 値に加算した和を、仮想アドレスとして汎用レジスタ *rx* に格納します。この命令は、この命令の近くに置かれた命令やデータの PC 相対アドレスを算出するための専用の命令です。

整数オーバフロー例外は絶対に発生しません。

伸長後の 32 ビット命令コードは、有効な 32 ビット ISA 命令ではなく、この命令の機能は、32 ビット ISA の ADDIU 命令とは異なります。

immediate フィールドは 8 ビットで、2 ビットシフトすることにより、扱うことのできる数値の範囲は、4 刻みで 0~1020 になります。この範囲外の値を指定すると、ADDIU 命令は EXTEND 命令により拡張され、符号付きの 16 ビットの即値 (–32768 ~ +32767) を扱えるようになります。この場合、*immediate* はシフトされません。

命令が遅延スロットに置かれているか、拡張されているかにより、ベース・PC 値は以下のように異なります。

ADDIUPC	ベース PC 値
JR・JALR 命令の遅延スロット	JR・JALR 命令のアドレス
JAL・JALX 命令の遅延スロット	JAL・JALX 命令の上位ハーフワードのアドレス
拡張されている	EXTEND 命令のアドレス
拡張されていない	ADDIUPC 命令のアドレス

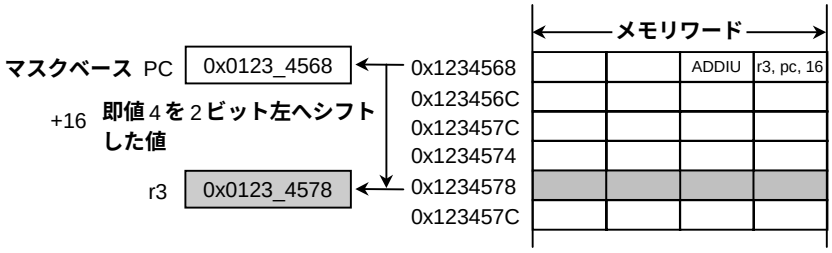
例外

なし

使用例

```
ADDIU r3,pc,16
```

上記の命令がアドレス 0x0123_456A に置かれており、このアドレスが遅延スロットでないものとします。このとき、PC の下位 2 ビットを 0 にマスクすると、マスクベース PC は、0x0123_4568 になります。即値は MIPS16 伸長回路で 2 ビット左へシフトされるため、指定したオペランド (16) はアセンブラにより 4 に変換されます。したがって、上記の ADDIU 命令の命令コードは 0x0B04 になります。下図に示すように、オフセットがマスクベース PC に付加され、その結果がレジスタ r3 に格納されます。



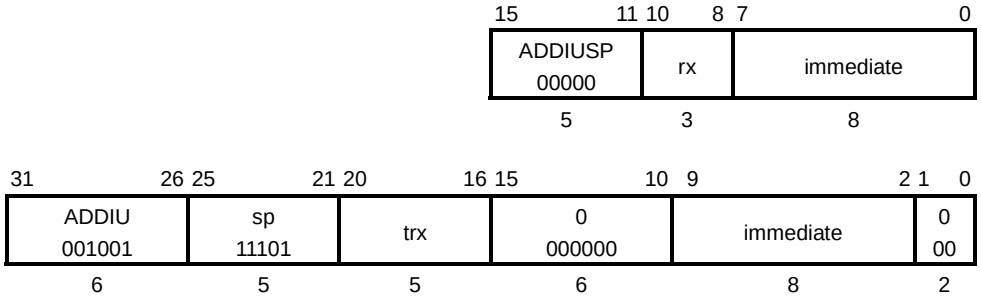
ADDIU *rx*, sp, *immediate*

Add Immediate Unsigned

動作

rx ⇐ sp + *immediate*

コード



説明

8 ビット *immediate* を 2 ビット左へシフトし、ゼロ拡張します。その結果をスタックレジスタポインタ sp (r29) の内容に加算した和を、汎用レジスタ *rx* に格納します。

整数オーバフロー例外は絶対に発生しません。

immediate フィールドは 8 ビットで、2 ビットシフトすることにより、扱うことのできる数値の範囲は、4 刻みで 0~1020 になります。この範囲外の値を指定すると、ADDIU 命令は EXTEND 命令により拡張され、符号付きの 16 ビットの即値 (–32768 ~ +32767) を扱えるようになります。この場合、*immediate* はシフトされません。

例外

なし

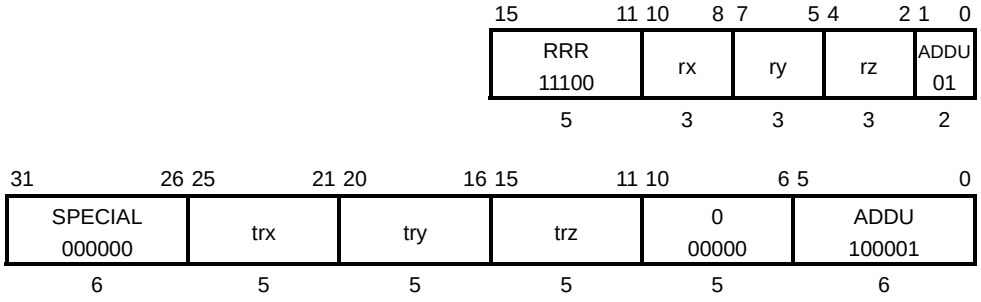
ADDU *rz*, *rx*, *ry*

Add Unsigned

動作

$rz \Leftarrow rx + ry$

コード



説明

汎用レジスタ *rx* の内容と汎用レジスタ *ry* の内容を加算し、その結果を汎用レジスタ *rz* に格納します。整数オーバフロー例外は絶対に発生しません。

例外

なし

使用例

レジスタ r2 の値が 0x0200_0000 で、レジスタ r3 の値が 0x0123_4567 の場合、以下の命令を実行すると、和 (0x0323_4567) がレジスタ r4 に格納されます。

```
ADD r4,r2,r3
```

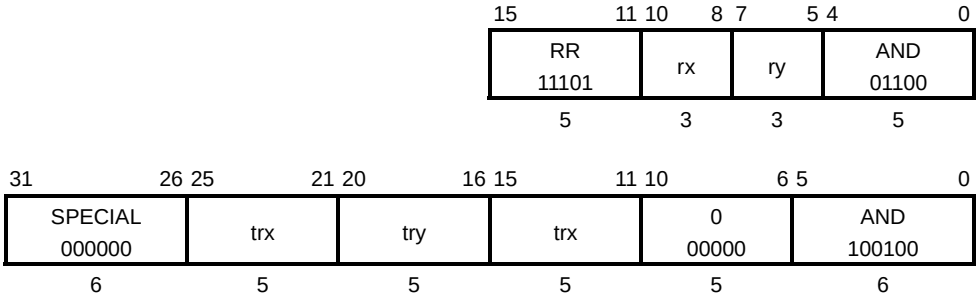
AND *rx*, *ry*

AND

動作

$rx \Leftarrow rx \text{ AND } ry$

コード



説明

汎用レジスタ *rx* の内容と汎用レジスタ *ry* の内容の論理積演算を行い、その結果を汎用レジスタ *rx* に格納します。

例外

なし

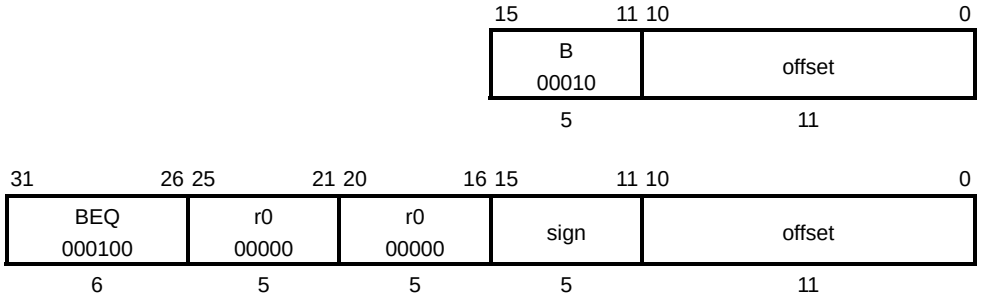
B *offset*

Branch Unconditional

動作

pc ⇐ pc + *offset*

コード



説明

1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスへ無条件に分岐します。パイプラインの遅延については、「5.3.4　分岐命令 (16 ビット ISA)」を参照してください。ターゲットアドレスは、直後の命令のアドレス (PC+2) に対して相対的に計算されます。11 ビット *offset* を 1 ビット左へシフトし、符号拡張した値を PC+2 に加算した結果がターゲットアドレスになります。

伸長後の 32 ビットの命令では、B 命令は r0 と r0 を比較する BEQ 命令として実行され、無条件分岐します。ただし、B 命令には遅延スロットがなく、分岐は直後に置かれた命令の前に実行されるという違いがあります。

offset フィールドは 11 ビットで、扱うことのできる数値の範囲は、–2048 ~ +2046 です。この範囲外の値を指定すると、B 命令は EXTEND 命令により拡張され、符号付きの 17 ビットの即値 (–65536 ~ +65534) を扱えるようになります。この場合も、ターゲットアドレスは、拡張しない場合と同様に計算されます。

例外

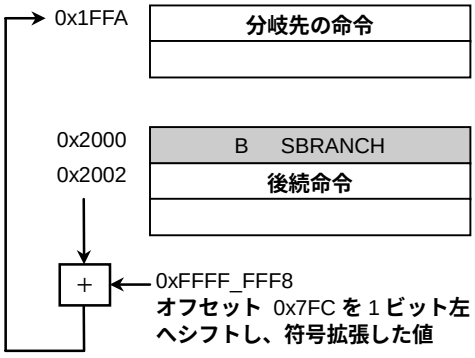
なし

使用例

```
B SBRANCH
```

上記の分岐命令がアドレス 0x2000 に置かれていて、ラベル SBRANCH が 0x1FFA に絶対アドレス化される場合、以下に図示するように、SBRANCH はアセンブラ・リンカによりオフセット 0x7FC に変換されます。したがって、命令コードは 0x17FC になります。

上記の命令を実行すると、プログラムの処理はターゲットアドレス 0x1FFA に分岐します。この場合、B 命令の後続の命令は実行されません。


0x1FFA
分岐先の命令
0x2000
B　SBRANCH
0x2002
後続命令
+
0xFFFF_FFF8
オフセット 0x7FC を 1 ビット左
へシフトし、符号拡張した値

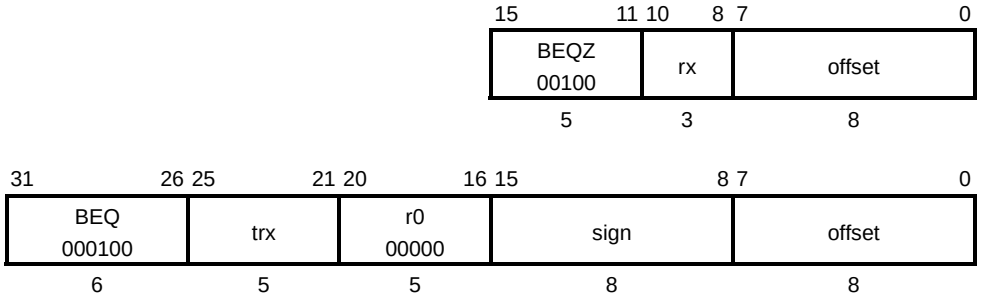
BEQZ *rx*, *offset*

Branch On Equal To Zero

動作

if *rx* = 0 then pc ⇐ pc + *offset*

コード



説明

汎用レジスタ *rx* の内容が 0 の場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐します。パイプライン遅延については「5.3.4 分岐命令 (16 ビット ISA)」を参照してください。ターゲットアドレスは、直後の命令アドレス (PC+2) に対して相対的に計算されます。8 ビット *offset* を 1 ビット左にシフトし、符号拡張した値を PC+2 に加算した結果がターゲットアドレスになります。

伸長後の 32 ビット ISA の BEQ 命令との違いは、16 ビット ISA の BEQZ 命令には遅延スロットがないことです。

offset フィールドは 8 ビットで、扱うことのできる数値の範囲は、–256 ~ +254 です。この範囲外の値を指定すると、BEQZ 命令は EXTEND 命令により拡張され、符号付きの 17 ビットの即値 (–65536 ~ +65534) を扱えるようになります。この場合も、ターゲットアドレスは、拡張しない場合と同様に計算されます。

例外

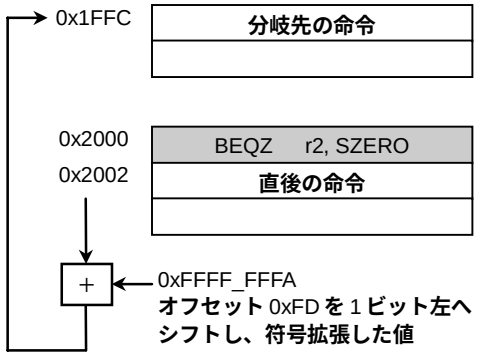
なし

使用例

```
BEQZ r2,SZERO
```

上記の分岐命令がアドレス 0x2000 に置かれていて、ラベル SZERO が 0x1FFC に絶対アドレス化される場合、以下に図示するように、SZERO はアセンブラ・リンカによりオフセット 0xFD に変換されます。したがって、命令コードは 0x22FD になります。

r2 の内容が 0 のとき、プログラムの処理はアドレス 0x1FFC に分岐します。r2 の内容が 0 以外の場合は、分岐せずに次の命令 (アドレス 0x2002) の実行に移ります。


0x1FFC
分岐先の命令
0x2000
BEQZ　r2, SZERO
0x2002
直後の命令
+
0xFFFF_FFFA
オフセット 0xFD を 1 ビット左へ
シフトし、符号拡張した値

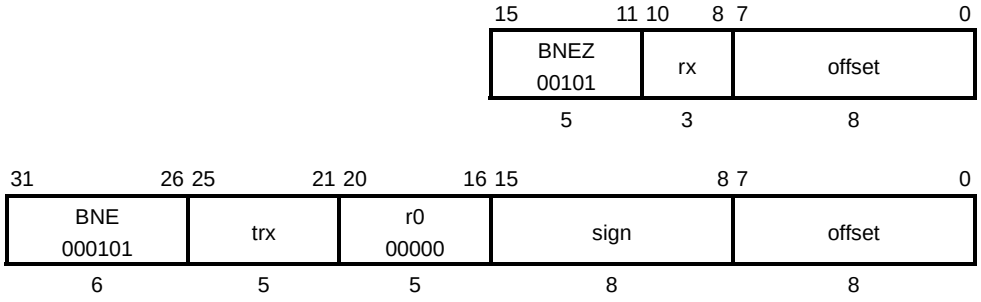
BNEZ *rx*, *offset*

Branch On Not Equal To Zero

動作

if *rx* ≠ 0 then pc ⇐ pc + *offset*

コード



説明

汎用レジスタ *rx* の内容が 0 でない場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐します。パイプライン遅延については「5.3.4 分岐命令 (16 ビット ISA)」を参照してください。ターゲットアドレスは、直後の命令アドレス (PC+2) に対して相対的に計算されます。8 ビット *offset* を 1 ビット左にシフトし、符号拡張した値を PC+2 に加算した結果がターゲットアドレスになります。

伸長後の 32 ビット ISA の BNE 命令との違いは、16 ビット ISA の BNEZ 命令には遅延スロットがないことです。

offset フィールドは 8 ビットで、扱うことのできる数値の範囲は、–256 ~ +254 です。この範囲外の値を指定すると、BNEZ 命令は EXTEND 命令により拡張され、符号付きの 17 ビットの即値 (–65536 ~ +65534) を扱えるようになります。この場合も、ターゲットアドレスは、拡張しない場合と同様に計算されます。

例外

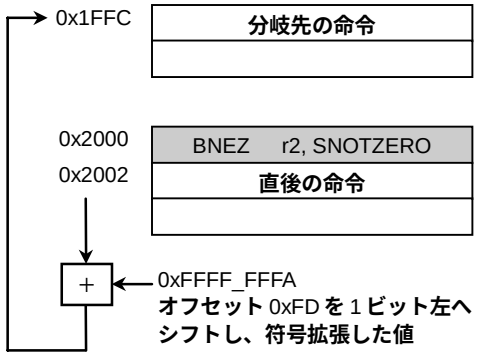
なし

使用例

```
BNEZ r2,SNOTZERO
```

上記の分岐命令がアドレス 0x2000 に置かれていて、ラベル SNOTZERO が 0x1FFC に絶対アドレス化される場合、以下に図示するように、SNOTZERO はアセンブラ・リンカによりオフセット 0xFD に変換されます。したがって、命令コードは 0x2AFD になります。

r2 の内容が 0 以外のとき、プログラムの処理はアドレス 0x1FFC に分岐します。r2 の内容が 0 の場合は、分岐せずに次の命令 (アドレス 0x2002) の実行に移ります。


0x1FFC
分岐先の命令
0x2000
BNEZ　r2, SNOTZERO
0x2002
直後の命令
+
0xFFFF_FFFA
オフセット 0xFD を 1 ビット左へ
シフトし、符号拡張した値

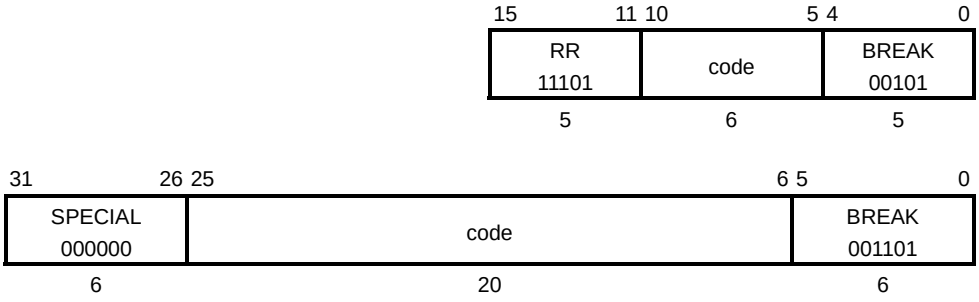
BREAK *code*

Breakpoint Exception

動作

Breakpoint exception

コード



説明

この命令を実行すると、無条件にブレークポイント例外が発生し、制御を例外ハンドラへ渡します。

命令内の *code* フィールドを使用して、例外ハンドラにパラメータを渡すことができます。例外ハンドラがこのパラメータを使用する場合には、命令を含むメモリハーフワードの内容をデータとしてロードする必要があります。詳細は「9.1.11　ブレークポイント例外」を参照してください。

例外

なし

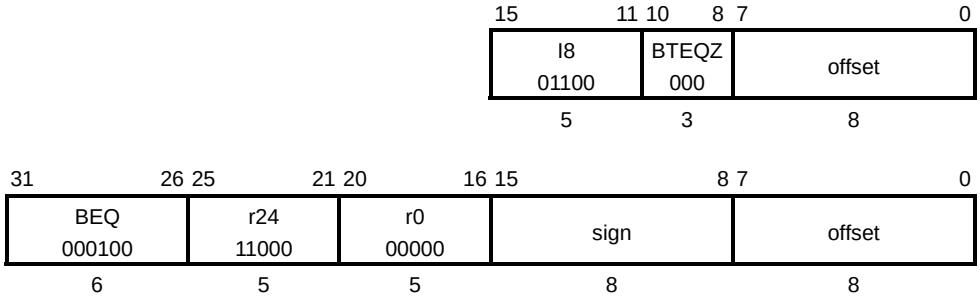
BTEQZ *offset*

Branch On T8 Equal To Zero

動作

if t8 = 0 then pc ⇐ pc + *offset*

コード



説明

コンディションコードレジスタ t8 (r24) の内容が 0 の場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐します。パイプラインの遅延については、「5.3.4　分岐命令 (16 ビット ISA)」を参照してください。ターゲットアドレスは、直後の命令のアドレス (PC+2) に対して相対的に計算されます。8 ビット *offset* を 1 ビット左へシフトし、符号拡張した値を PC+2 に加算した結果がターゲットアドレスになります。

伸長後の 32 ビット ISA の BEQ 命令との違いは、16 ビット ISA の BTEQZ 命令には遅延スロットがないことです。

offset フィールドは 8 ビットで、扱うことのできる数値の範囲は、–256 ~ +254 です。この範囲外の値を指定すると、BTEQZ 命令は EXTEND 命令により拡張され、符号付きの 17 ビットの即値 (–65536 ~ +65534) を扱えるようになります。この場合も、ターゲットアドレスは、拡張しない場合と同様に計算されます。

例外

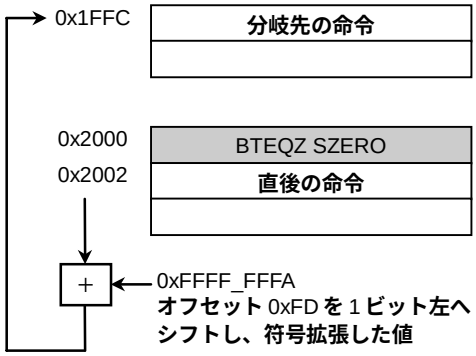
なし

使用例

```
BTEQZ  SZERO
```

上記の分岐命令がアドレス 0x2000 に置かれていて、ラベル SZERO が 0x1FFC に絶対アドレス化される場合、以下に図示するように、SZERO はアセンブラ・リンカによりオフセット 0xFD に変換されます。したがって、命令コードは 0x60FD になります。

t8 の内容が 0 のとき、プログラムの処理はアドレス 0x1FFC に分岐します。t8 の内容が 0 以外の場合は、分岐せずに次の命令 (アドレス 0x2002) の実行に移ります。


0x1FFC
分岐先の命令
0x2000
BTEQZ SZERO
0x2002
直後の命令
+
0xFFFF_FFFA
オフセット 0xFD を 1 ビット左へ
シフトし、符号拡張した値

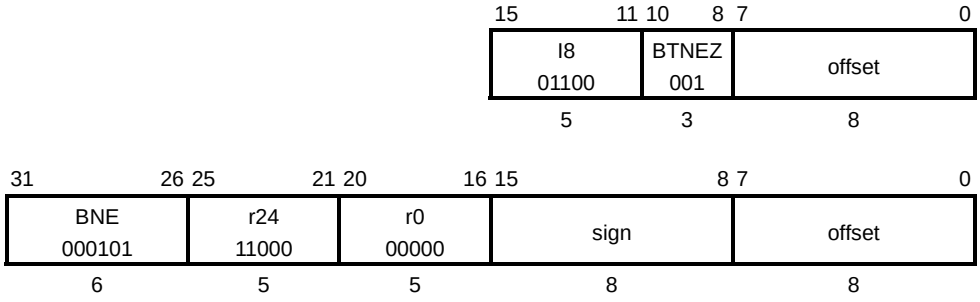
BTNEZ *offset*

Branch On T8 Not Equal To Zero

動作

if t8 ≠ 0 then pc ⇐ pc + *offset*

コード



説明

コンディションコードレジスタ t8 (r24) の内容が 0 でない場合、1 命令 (2 命令サイクル) の遅延後、ターゲットアドレスに分岐します。パイプラインの遅延については、「5.3.4 分岐命令 (16 ビット ISA)」を参照してください。ターゲットアドレスは、直後の命令のアドレス (PC+2) に対して相対的に計算されます。8 ビット *offset* を 1 ビット左へシフトし、符号拡張した値を PC+2 に加算した結果がターゲットアドレスになります。

伸長後の 32 ビット ISA の BNE 命令との違いは、16 ビット ISA の BTNEZ 命令には遅延スロットがないことです。

offset フィールドは 8 ビットで、扱うことのできる数値の範囲は、–256 ~ +254 です。この範囲外の値を指定すると、BTNEZ 命令は EXTEND 命令により拡張され、符号付きの 17 ビットの即値 (–65536 ~ +65534) を扱えるようになります。この場合も、ターゲットアドレスは、拡張しない場合と同様に計算されます。

例外

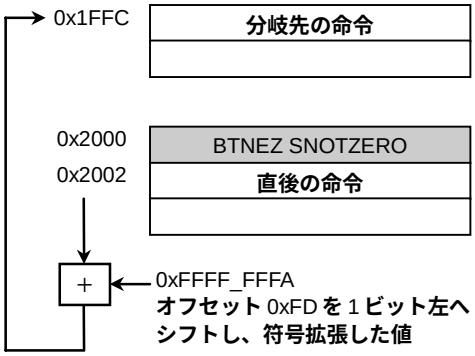
なし

使用例

```
BTNEZ  SNOTZERO
```

上記の分岐命令がアドレス 0x2000 に置かれていて、ラベル SNOTZERO が 0x1FFC に絶対アドレス化される場合、以下に図示するように、SNOTZERO はアセンブラ・リンカによりオフセット 0xFD に変換されます。したがって、命令コードは 0x61FD になります。

t8 の内容が 0 のとき、プログラムの処理はアドレス 0x1FFC に分岐します。t8 の内容が 0 以外の場合は、分岐せずに次の命令 (アドレス 0x2002) の実行に移ります。


0x1FFC
分岐先の命令
0x2000
BTNEZ SNOTZERO
0x2002
直後の命令
+
0xFFFF_FFFA
オフセット 0xFD を 1 ビット左へ
シフトし、符号拡張した値

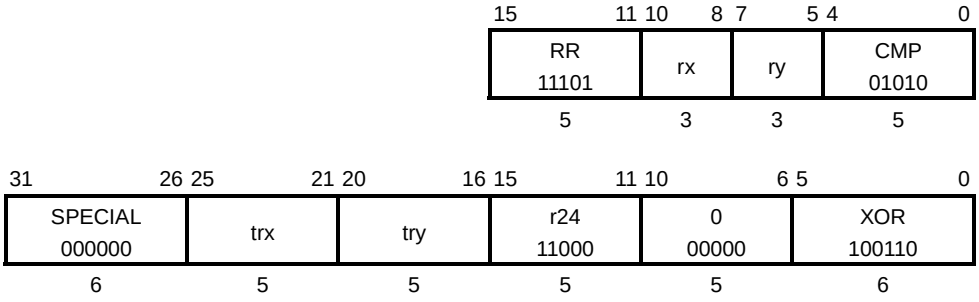
CMP *rx*, *ry*

Compare

動作

if *rx* = *ry* then t8 ⇐ 0; else t8 ⇐ non-zero value

コード



説明

汎用レジスタ *rx* の内容と汎用レジスタ *ry* の内容の排他的論理和 (XOR) をとり、結果をコンディションコードレジスタ t8 (r24) に格納します。したがって、両者が等しい場合、t8 には 0 が入ります。

例外

なし

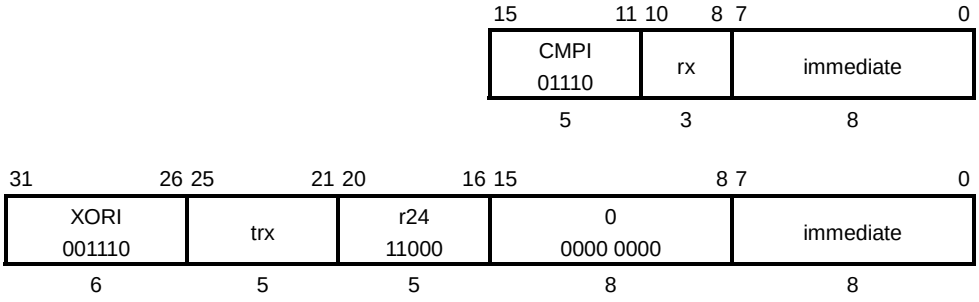
CMPI *rx*, *immediate*

Compare Immediate

動作

if *rx* = *immediate* then t8 ⇐ 0; else t8 ⇐ non-zero value

コード



説明

8 ビット *immediate* をゼロ拡張した値と、汎用レジスタ *rx* の内容の排他的論理和 (XOR) をとり、結果をコンディションコードレジスタ t8 (r24) に格納します。したがって、両者が等しい場合、t8 には 0 が入ります。

immediate フィールドは 8 ビットで、扱うことのできる数値の範囲は、0~255 です。この範囲外の値を指定すると、CMPI 命令は EXTEND 命令により拡張され、符号なしの 16 ビットの即値 (0~65535) を扱えるようになります。

例外

なし

DIV *rx*, *ry*

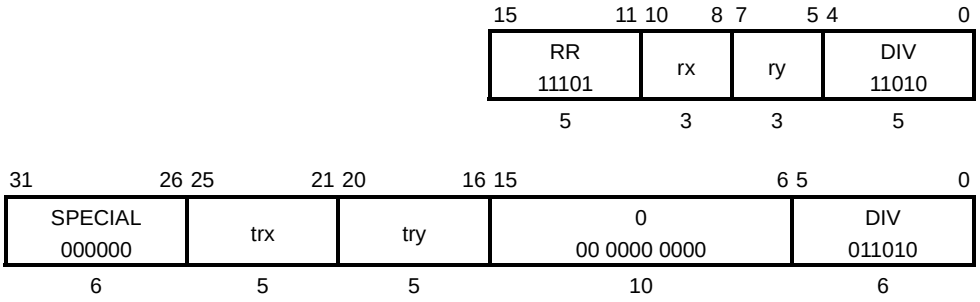
Divide

動作

LO ⇐ *rx* ÷ *ry*;

HI ⇐ *rx* MOD *ry*

コード



説明

汎用レジスタ *rx* の内容を汎用レジスタ *ry* の内容で除算します。両オペランドとも、符号付き整数として扱われます。商は LO レジスタに格納され、剰余は HI レジスタに格納されます。整数オーバフロー例外は発生しません。

除数が 0 の場合、DIV 命令の結果は確定しません。通常、DIV 命令の後に、ゼロ除算とオーバフローを検査する命令を置きます。

除算命令は、専用の除算ユニットで実行されるため、他の命令の実行を並行して継続できます。除算ユニットは、キャッシュミス、遅延サイクル、例外が起きたときでも、実行を継続します。

除算命令が完了する前に、MFHI、MFLO、MADD、 MADDU 命令で除算結果を読もうとすると、パイプラインがストールします (「5.4　除算命令」を参照)。

例外

なし

DIVU *rx*, *ry*

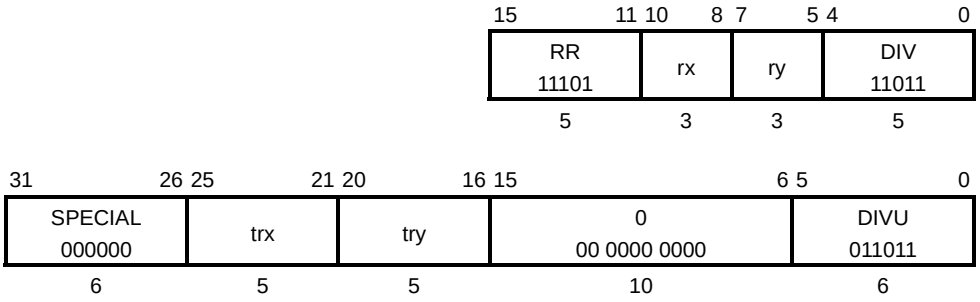
Divide Unsigned

動作

LO ⇐ *rx* ÷ *ry*;

HI ⇐ *rx* MOD *ry*

コード



説明

汎用レジスタ *rx* の内容を汎用レジスタ *ry* の内容で除算します。両オペランドとも、符号なし整数として扱われます。商は LO レジスタに格納され、剰余は HI レジスタに格納されます。整数オーバフロー例外を発生しません。DIV 命令との唯一の違いは、DIVU 命令ではオペランドが符号なし整数として扱われるという点だけです。

例外

なし

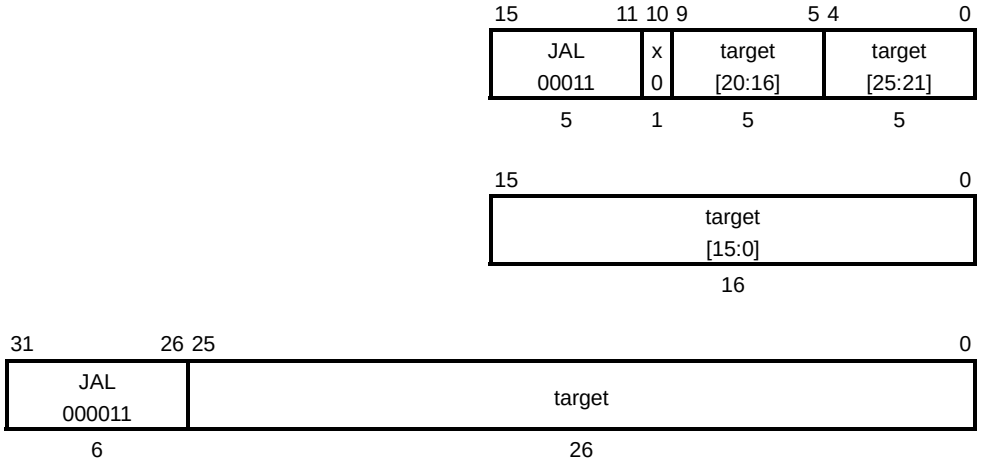
JAL *target*

Jump And Link

動作

ra ⇐ pc + 7; pc ⇐ pc[31:28] ‖ *target* ‖ 00

コード



説明

16 ビット ISA の JAL 命令は、例外的に 32 ビット長で、そのため、2 段階に分けてフェッチされます。1 命令 (2 命令サイクル) の遅延後、無条件にターゲットアドレスにジャンプします。「5.3.3 ジャンプ命令 (16 ビット ISA)」を参照してください。ターゲットアドレスは、ジャンプ遅延スロット内の命令のアドレス (PC+2) に対して相対的に計算されます。26 ビット *target* を 2 ビット左へシフトし、PC+2 の上位 4 ビットと連結した結果がターゲットアドレスになります。プログラムカウンタ (PC) の ISA モードビットは変化しません。

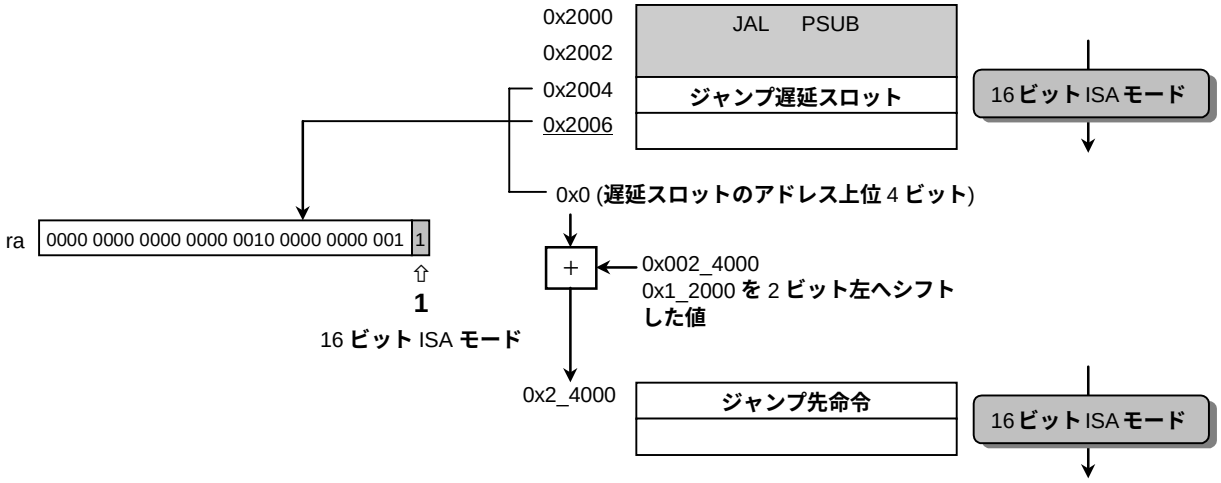
ジャンプ遅延スロットの次の命令のアドレスを、リンクレジスタ ra (r31) に格納します。また、ra の最下位ビットに、ISA モードビット (16 ビット ISA モード = 1) を格納します。

使用例

```
JAL PSUB
```

上記のジャンプ命令がアドレス 0x2000 にあり、ラベル PSUB が 0x2_4000 に絶対アドレス化される場合、以下に図示するように、PSUB はアセンブラ・リンカにより 0x1_2000 に変換されます。

プログラムの処理は、無条件にアドレス 0x2_4000 にジャンプします。ジャンプ遅延スロット内の命令は、ジャンプのまえに実行されます。また、ジャンプ遅延スロットの次の命令のアドレスが、ISA モードビットと共に ra に格納され、ra は 0x0000_2007 になります。


0x2000
0x2002
0x2004
0x2006
JAL　PSUB
ジャンプ遅延スロット
16 ビット ISA モード
0x0 (遅延スロットのアドレス上位 4 ビット)
ra
0000 0000 0000 0000 0010 0000 0000 001
1
1
16 ビット ISA モード
+
0x002_4000
0x1_2000 を 2 ビット左へシフト
した値
0x2_4000
ジャンプ先命令
16 ビット ISA モード

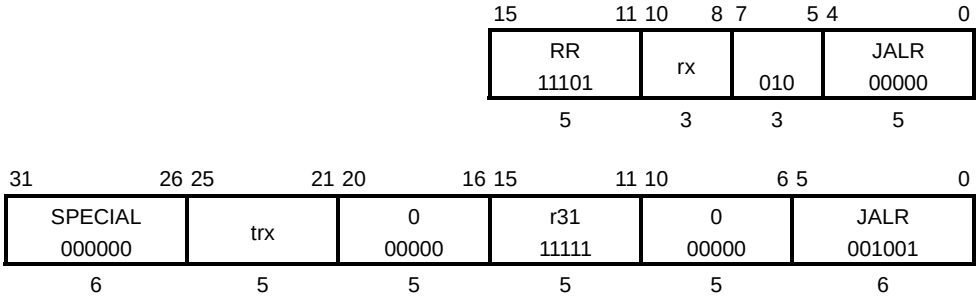
JALR ra, *rx*

Jump And Link Register

動作

ra ⇐ pc + 5; pc ⇐ *rx*

コード



説明

1 命令 (2 命令サイクル) の遅延後、汎用レジスタ *rx* の最下位ビットを 0 にマスクしたアドレスに無条件にジャンプします。*rx* の最下位ビットの値によって、ISA モードが切り換わります。また、ジャンプ遅延スロットの次の命令のアドレスを、ジャンプ前の ISA モードビットと共にリンクレジスタ ra (r31) に格納します。

32 ビット ISA では、命令はすべてワード境界で位置合わせされなければなりません。そのため、32 ビット ISA モードに移行する場合、ターゲットレジスタ (*rx*) の下位 2 ビットは 0 でなければなりません。下位 2 ビットが 0 でない場合、ジャンプ先の命令をフェッチするときにアドレスエラー例外が発生します。

例外

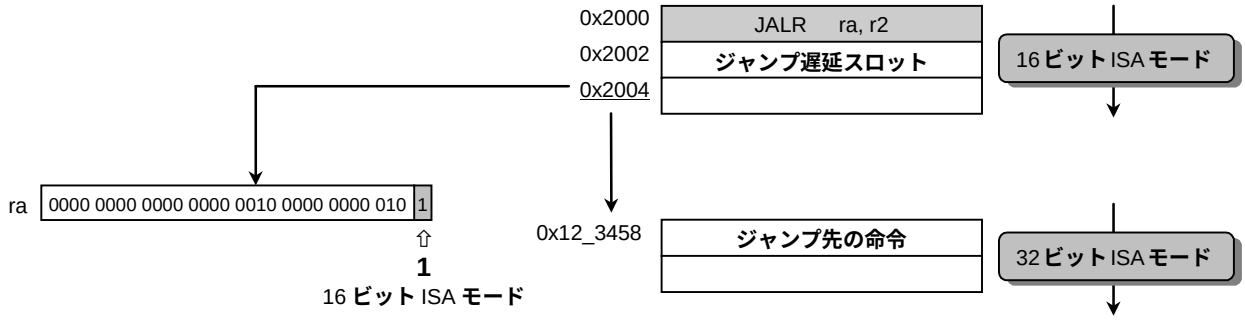
なし

使用例

レジスタ r2 の内容が 0x0012_3458 で、以下のジャンプ命令がアドレス 0x0000_2000 に置かれているとします。

```
JALR ra,r2
```

上記の命令を実行すると、プログラムの処理は、アドレス 0x0012_3458 にジャンプします。レジスタ r2 の最下位ビットは 0 なので、ジャンプ後の ISA モードビットは 0 に変化し、32 ビット ISA モードになります。また、ジャンプ遅延スロットの次の命令のアドレスが、ISA モードビットと共に ra に格納され、ra は 0x0000_2005 になります。


0x2000
JALR　ra, r2
0x2002
ジャンプ遅延スロット
0x2004
16 ビット ISA モード
ra
0000 0000 0000 0000 0010 0000 0000 010
1
1
16 ビット ISA モード
0x12_3458
ジャンプ先の命令
32 ビット ISA モード

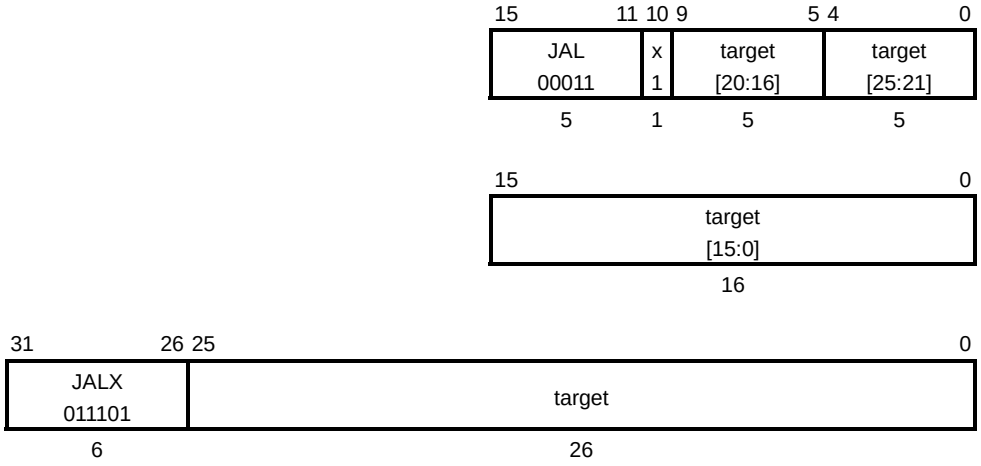
JALX *target*

Jump And Link eXchange

動作

ra ⇐ pc + 7; pc[31:1] ⇐ pc[31:28] ‖ *target* ‖ 00; pc[0] ⇐ NOT pc[0]

コード



説明

16 ビット ISA の JALX 命令は、例外的に 32 ビット長で、そのため、2 段階に分けてフェッチされます。1 命令 (2 命令サイクル) の遅延後、無条件にターゲットアドレスにジャンプします。「5.3.3 ジャンプ命令 (16 ビット ISA)」を参照してください。ターゲットアドレスは、ジャンプ遅延スロット内の命令のアドレス (PC+2) に対して相対的に計算されます。26 ビット *target* を 2 ビット左へシフトし、PC+2 の上位 4 ビットと連結した結果がターゲットアドレスになります。プログラムカウンタ (PC) の ISA モードビットが無条件に変化します。

ジャンプ遅延スロットの次の命令のアドレスを、リンクレジスタ ra (r31) に格納します。また、ra の最下位ビットに、ジャンプまえの ISA モードビットを格納します。

例外

なし

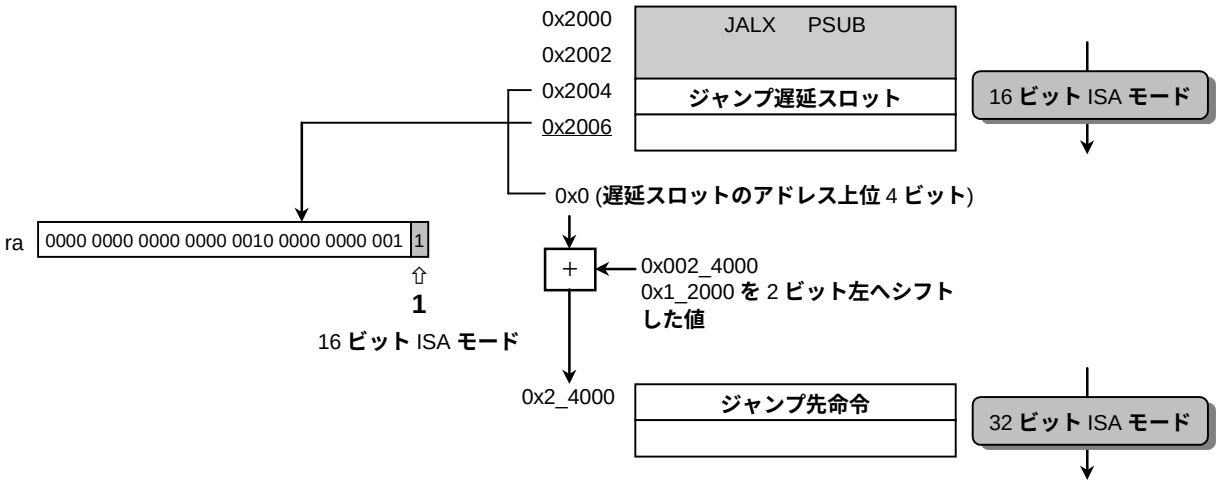
使用例

```
JALX PSUB
```

上記のジャンプ命令がアドレス 0x0000_20000 にあり、ラベル PSUB が 0x2_4000 に絶対アドレス化される場合、以下に図示するように、PSUB はアセンブラ・リンカにより 0x1_2000 に変換されます。

プログラムの処理は、無条件にアドレス 0x2_4000 にジャンプします。ジャンプ遅延スロット内の命令は、ジャンプのまえに実行されます。ISA モードは無条件に変化し、32 ビット ISA モードに切り換

わります。また、ジャンプ遅延スロットの次の命令のアドレスが、ISA モードビットと共に ra に格納され、ra は 0x0000_2007 になります。



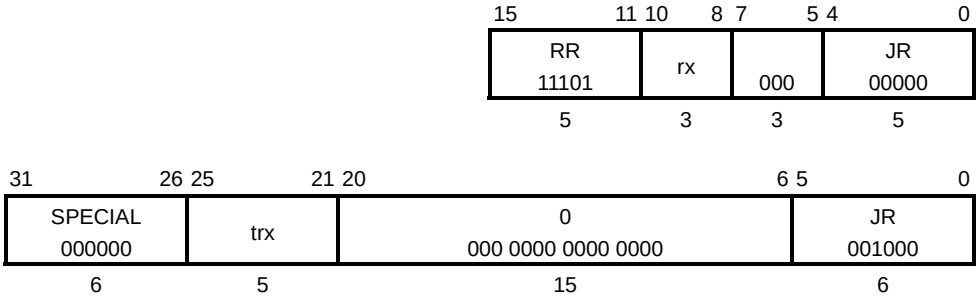
JR *rx*

Jump Register

動作

pc ⇐ *rx*

コード



説明

1 命令 (2 命令サイクル) の遅延後、汎用レジスタ *rx* の最下位ビットを 0 にマスクしたアドレスに無条件にジャンプします。*rx* の最下位ビットの値によって ISA モードが切り換わります。

32 ビット ISA では、命令はすべてワード境界に位置合わせされていなければなりません。そのため、32 ビット ISA モードにジャンプするとき、ターゲットレジスタ (*rx*) の下位 2 ビットは 0 でなければなりません。下位 2 ビットが 0 でない場合、プロセッサがジャンプ先の命令をフェッチするときにアドレスエラー例外が発生します。

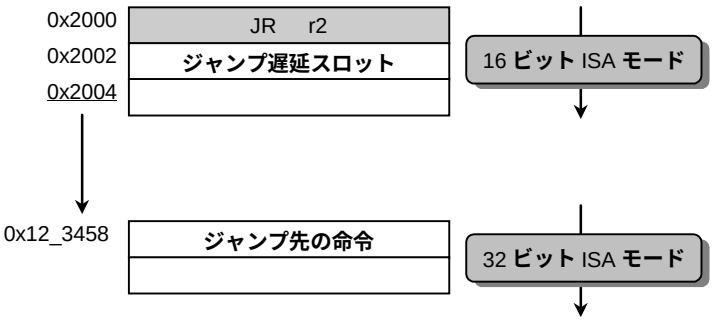
例外

なし

使用例

レジスタ r2 の値が 0x0012_3458 の場合、以下の命令を実行すると、図示したようにプログラムの処理がアドレス 0x0012_3458 へ移ります。r2 の最下位ビットはクリアされ、32 ビット ISA モードに切り換わります。ジャンプ遅延スロット内の命令はジャンプの前に実行されます。

```
JR r2
```


0x2000
0x2002
0x2004
JR　r2
ジャンプ遅延スロット
16 ビット ISA モード
0x12_3458
ジャンプ先の命令
32 ビット ISA モード

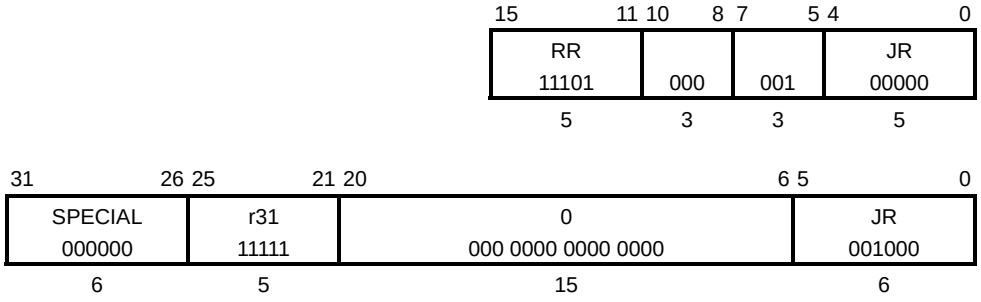
JR ra

Jump Register

動作

pc ⇐ ra

コード



説明

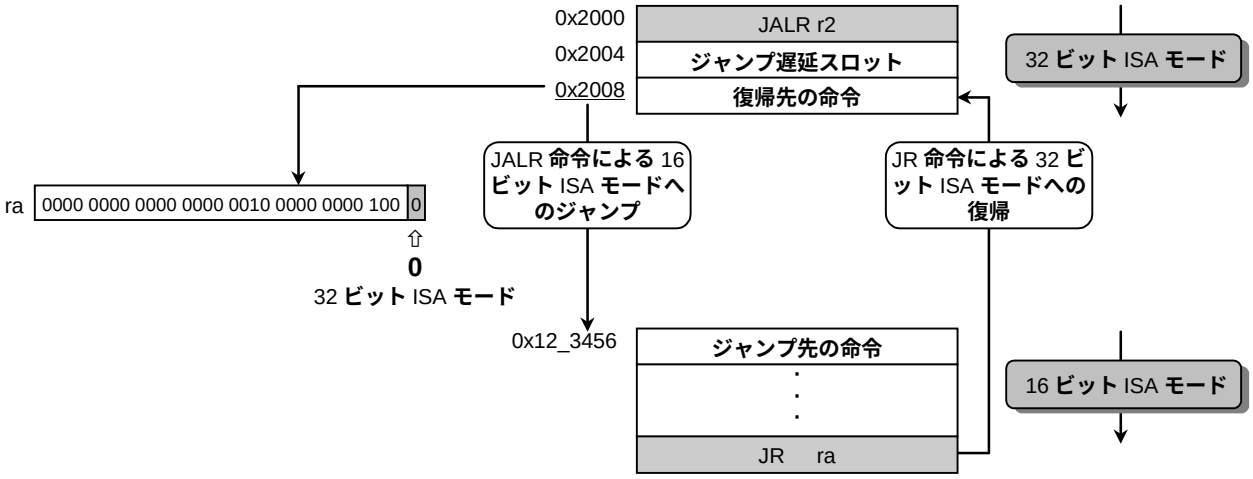
1 命令 (2 命令サイクル) の遅延後、リンクレジスタ ra (r31) の最下位ビットを 0 にマスクしたアドレスに無条件にジャンプします。ra の最下位ビットの値によって ISA モードが切り換わります。

例外

なし

使用例

以下の例では、32 ビットのルーチンの最後で JALR 命令により 16 ビット ISA モードに切り換えています。そして、16 ビットのルーチンの最後で、JR 命令により戻りアドレスをリンクレジスタ ra (r31) からプログラムカウンタ (PC) に復元しています。32 ビットの JALR 命令により、ISA モードが ra の最下位ビットに保存されているので、16 ビットルーチンの終わりで JR 命令を実行すると、32 ビット ISA モードに戻ります。


0x2000
JALR r2
0x2004
ジャンプ遅延スロット
0x2008
復帰先の命令
32 ビット ISA モード
JALR 命令による 16 ビット ISA モードへのジャンプ
JR 命令による 32 ビット ISA モードへの復帰
ra
0000 0000 0000 0000 0010 0000 0000 100
0
0
32 ビット ISA モード
0x12_3456
ジャンプ先の命令
JR ra
16 ビット ISA モード

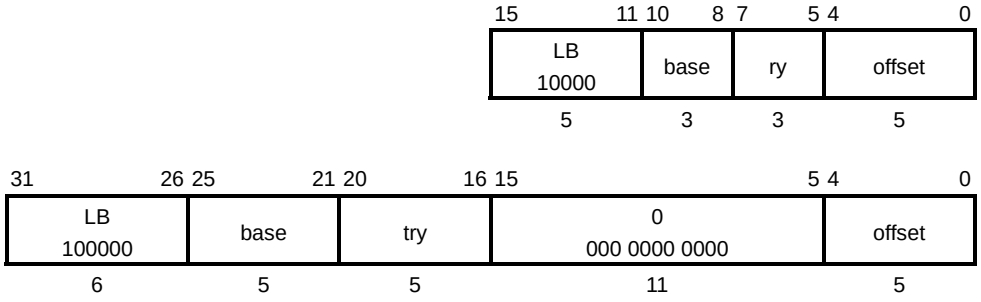
LB *ry*, *offset* (*base*)

Load Byte

動作

ry ⇐ {*offset* (*base*)}

コード



説明

5 ビット *offset* をゼロ拡張して、汎用レジスタ *base* の内容に加算することにより、実効アドレス (EA) を生成します。EA でアドレス指定されたメモリ中のバイトデータを符号拡張して、汎用レジスタ *ry* にロードします。

offset フィールドは 5 ビットで、扱うことのできる数値の範囲は、0~31 です。この範囲外の値を指定すると、LB 命令は EXTEND 命令により拡張され、符号付きの 16 ビット即値 (–32768 ~ +32767) を扱えるようになります。

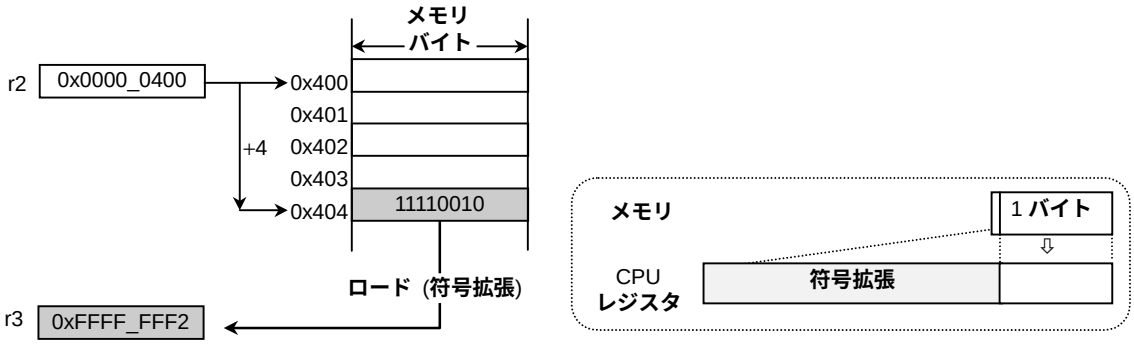
例外

アドレスエラー例外

使用例

レジスタ r2 の値が 0x0000_0400 で、アドレス 0x404 の内容が 0xF2 の場合、以下の命令を実行すると、レジスタ r3 に 0xFFFF_FFF2 がロードされます。

```
LB r3,4(r2)
```


メモリ
バイト
r2
0x0000_0400
0x400
0x401
+4
0x402
0x403
0x404
11110010
ロード (符号拡張)
r3
0xFFFF_FFF2
メモリ
1 バイト
CPU
レジスタ
符号拡張

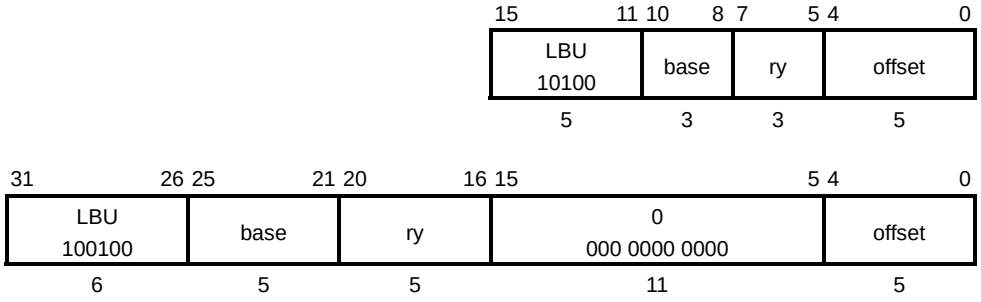
LBU *ry*, *offset* (*base*)

Load Byte Unsigned

動作

ry ⇐ {*offset* (*base*)}

コード



説明

5 ビット *offset* をゼロ拡張して、汎用レジスタ *base* の内容に加算することにより、実効アドレス (EA) を生成します。EA でアドレス指定されたメモリ中のバイトデータをゼロ拡張して、汎用レジスタ *ry* にロードします。

offset フィールドは 5 ビットで、扱うことのできる数値の範囲は、0~31 です。この範囲外の値を指定すると、LBU 命令は EXTEND 命令により拡張され、符号付きの 16 ビット即値 (–32768 ~ +32767) を扱えるようになります。

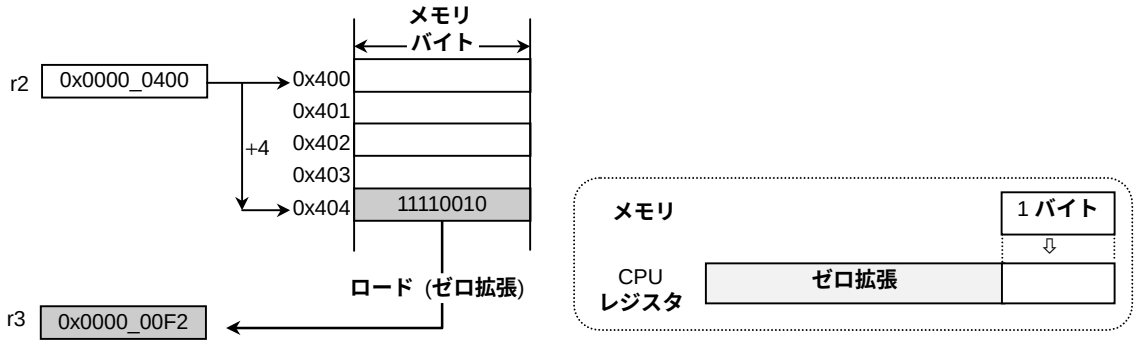
例外

アドレスエラー例外

使用例

レジスタ r2 の値が 0x0000_0400 で、アドレス 0x404 の内容が 0xF2 の場合、以下の命令を実行すると、レジスタ r3 に 0x0000_00F2 がロードされます。

```
LBU r3,4(r2)
```


メモリ
バイト
r2
0x0000_0400
0x400
0x401
0x402
0x403
0x404
+4
11110010
ロード（ゼロ拡張）
r3
0x0000_00F2
メモリ
1 バイト
CPU
レジスタ
ゼロ拡張

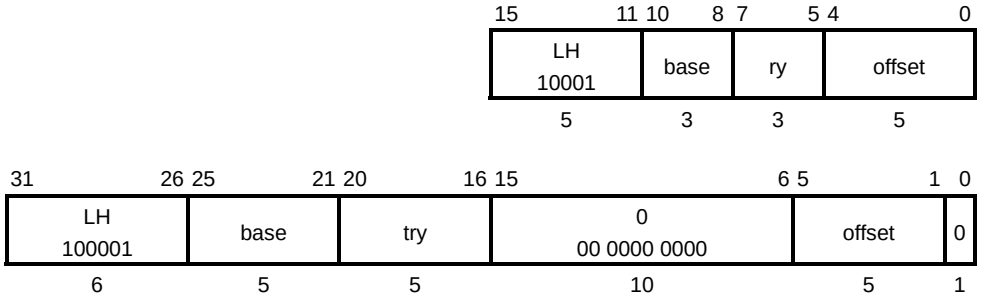
LH *ry*, *offset* (*base*)

Load Halfword

動作

ry ⇐ {*offset* (*base*)}

コード



説明

5 ビット *offset* を 1 ビット左へシフトして、ゼロ拡張し、汎用レジスタ *base* に加算することにより、実効アドレス (EA) を生成します。EA によってアドレス指定されたメモリ中のハーフワードデータを符号拡張し、汎用レジスタ *ry* にロードします。

offset は 5 ビットで、1 ビットシフトすることにより扱うことのできる数値の範囲は、2 刻みで 0~62 になります。この範囲外の値を指定すると、LH 命令は EXTEND 命令により拡張され、符号付きの 16 ビットの即値 (–32768 ~ +32767) を扱えるようになります。この場合、offset はシフトされません。

例外

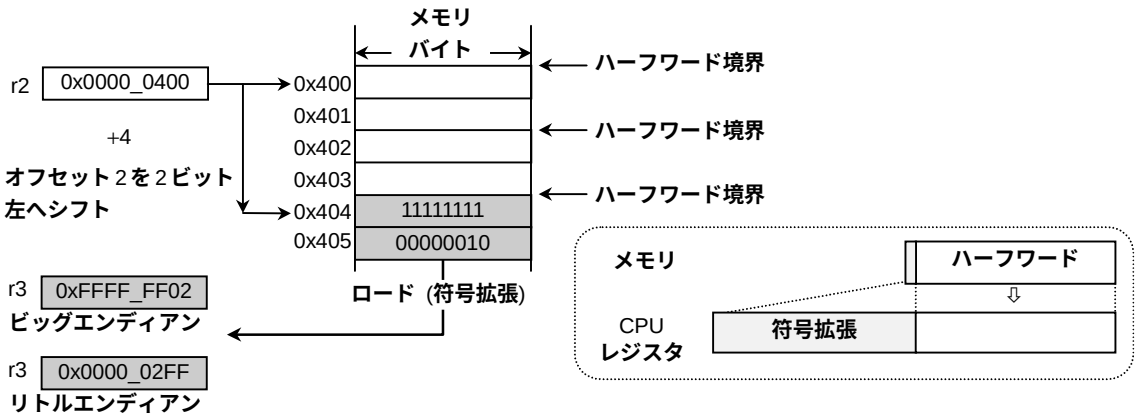
アドレスエラー例外

使用例

```
LH r3,4(r2)
```

レジスタ r2 の内容が 0x0000_0400 で、アドレス 0x404 と 0x405 の内容がそれぞれ 0xFF と 0x02 であるとします。オフセット値は伸張回路により 1 ビット左へシフトされるので、指定されたオフセット値 (4 = 2 進数 0100) は、アセンブラ・リンカにより、2 (2 進数 0010) に変換されます。したがって、命令コードは 0x8A62 になります。

上記の命令を実行すると、レジスタ r3 には、ビッグエンディアンのときは 0xFFFF_FF02 がロードされ、リトルエンディアンのときは 0x0000_02FF がロードされます。


メモリ
バイト
ハーフワード境界
r2
0x0000_0400
0x400
0x401
0x402
0x403
0x404
0x405
11111111
00000010
+4
オフセット 2 を 2 ビット
左へシフト
ハーフワード境界
ハーフワード境界
ロード（符号拡張）
r3
0xFFFF_FF02
ビッグエンディアン
r3
0x0000_02FF
リトルエンディアン
メモリ
ハーフワード
CPU
レジスタ
符号拡張

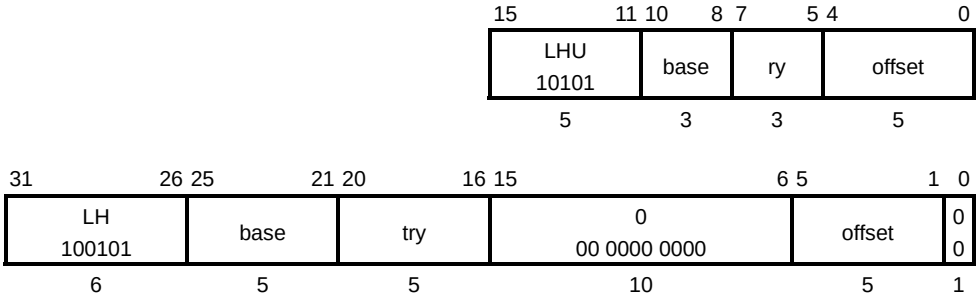
LHU *ry*, *offset* (*base*)

Load Halfword Unsigned

動作

ry ⇐ {*offset* (*base*)}

コード



説明

5 ビット *offset* を 1 ビット左へシフトして、ゼロ拡張し、汎用レジスタ *base* の内容に加算することにより、実効アドレス (EA) を生成します。EA によってアドレス指定されたハーフワードデータをゼロ拡張し、汎用レジスタ *ry* にロードします。

offset は 5 ビットで、1 ビットシフトすることにより扱うことのできる数値の範囲は、2 刻みで 0~62 になります。この範囲外の値を指定すると、LHU 命令は EXTEND 命令により拡張され、符号付きの 16 ビットの即値 (–32768 ~ +32767) を扱えるようになります。この場合、offset はシフトされません。

例外

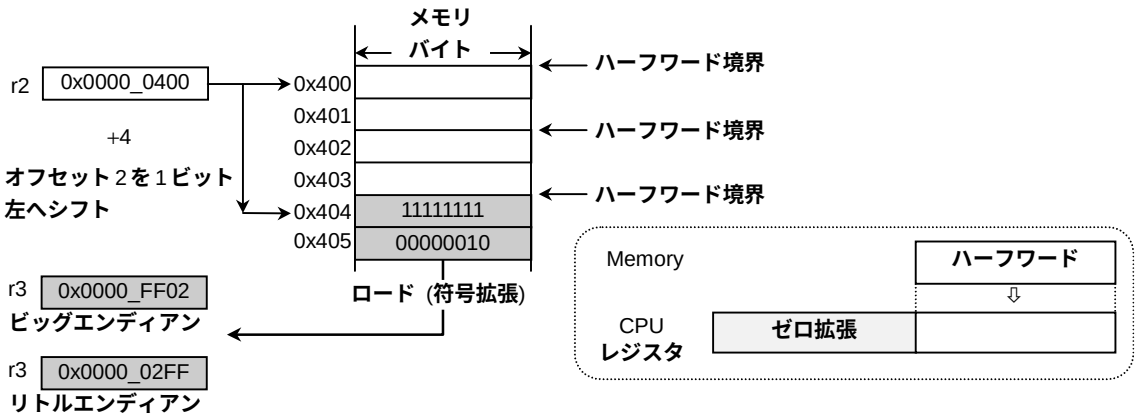
アドレスエラー例外

使用例

```
LHU r3,4(r2)
```

レジスタ r2 の内容が 0x0000_0400 で、アドレス 0x404 と 0x405 の内容がそれぞれ 0xFF と 0x02 であるとします。オフセット値は伸張回路により 1 ビット左へシフトされるので、指定されたオフセット値 (4 = 2 進数 0100) は、アセンブラ・リンカにより、2 (2 進数 0010) に変換されます。したがって、命令コードは 0xAA62 になります。

上記の命令を実行すると、レジスタ r3 には、ビッグエンディアンのときは 0x0000_FF02 がロードされ、リトルエンディアンのときは 0x0000_02FF がロードされます。


メモリ
バイト
ハーフワード境界
r2
0x0000_0400
0x400
0x401
0x402
0x403
0x404
0x405
11111111
00000010
+4
オフセット 2 を 1 ビット
左へシフト
ロード (符号拡張)
r3
0x0000_FF02
ビッグエンディアン
r3
0x0000_02FF
リトルエンディアン
Memory
ハーフワード
CPU
レジスタ
ゼロ拡張

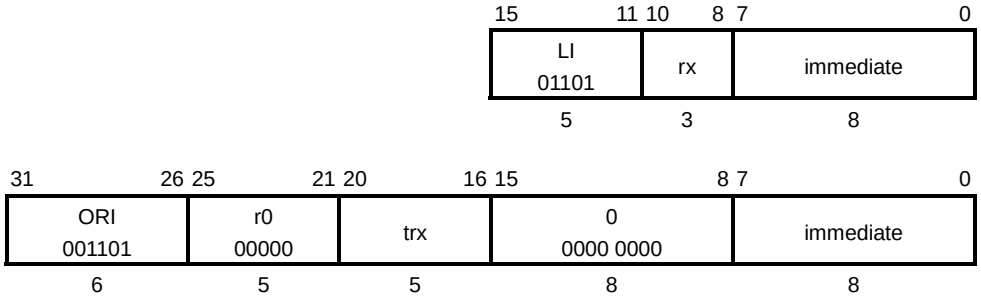
LI *rx*, *immediate*

Load Immediate

動作

rx ⇐ *immediate*

コード



説明

8 ビット *immediate* をゼロ拡張し、汎用レジスタ *rx* に格納します。

immediate フィールドは 8 ビットで、扱うことのできる数値の範囲は、0~255 です。この範囲外の値を指定すると、EXTEND 命令により拡張され、符号なしの 16 ビットの即値 (0~65535) が扱えるようになります。

例外

なし

使用例

以下の命令を実行すると、レジスタ r3 に 0x0000_0012 がロードされます。

```
LI r3,0x12
```

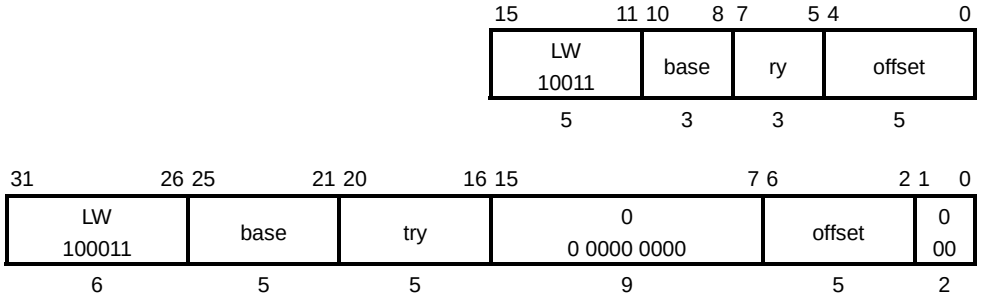
LW *ry*, *offset* (*base*)

Load Word

動作

ry ⇐ (*offset* (*base*)}

コード



説明

5 ビット *offset* を 2 ビット左へシフトして、ゼロ拡張し、汎用レジスタ *base* の内容に加算することにより、実効アドレス (EA) を生成し、EA によってアドレス指定されたワードデータを汎用レジスタ *ry* にロードします。

offset は 5 ビットで、2 ビット左へシフトすることにより扱うことのできる数値の範囲は、4 刻みで 0~124 になります。この範囲外の値を指定すると、LW 命令は EXTEND 命令により拡張され、符号付きの 16 ビットの即値 (–32768 ~ +32767) を扱えるようになります。この場合、*offset* はシフトされません。

例外

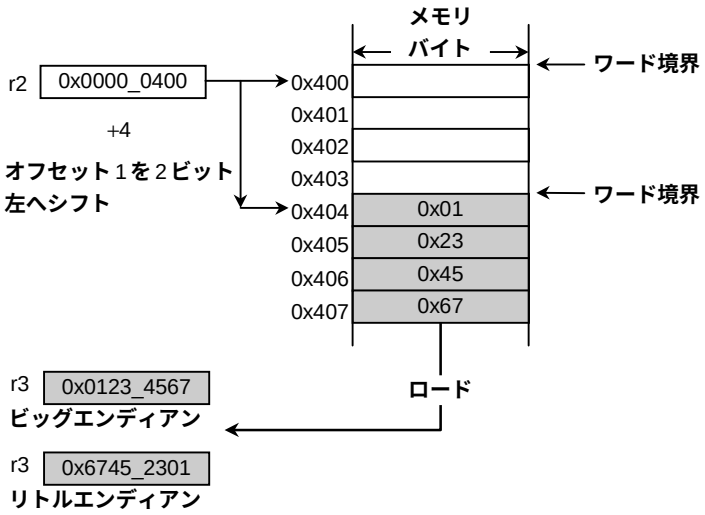
アドレスエラー例外

使用例

```
LW r3,4(r2)
```

レジスタ r2 の内容が 0x0000_0400 で、アドレス 0x404~0x407 の内容がそれぞれ 0x01、0x23、0x45、0x67 であるとします。オフセット値は伸張回路により 2 ビット左へシフトされるので、指定されたオフセット値 (4 = 2 進数 0100) は、アセンブラ・リンカにより、1 (2 進数 0001) に変換されます。したがって、命令コードは 0x9A61 になります。

上記の命令を実行すると、レジスタ r3 には、ビッグエンディアンのときは 0x0123_4567 がロードされ、リトルエンディアンのときは 0x6745_2301 がロードされます。


メモリ
バイト
ワード境界
r2
0x0000_0400
+4
オフセット 1 を 2 ビット
左へシフト
0x400
0x401
0x402
0x403
0x404
0x405
0x406
0x407
0x01
0x23
0x45
0x67
ワード境界
ロード
r3
0x0123_4567
ビッグエンディアン
r3
0x6745_2301
リトルエンディアン

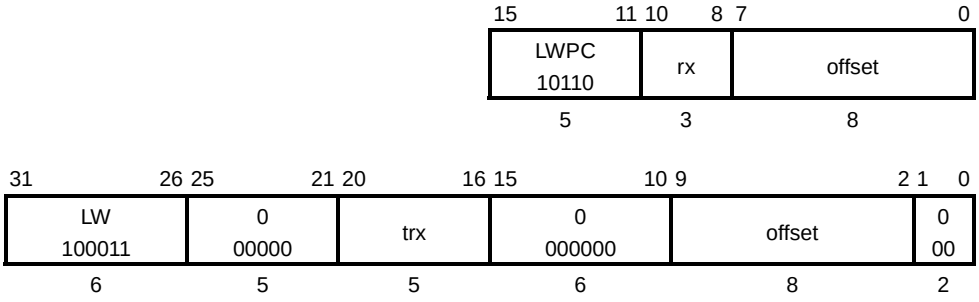
LW *rx*, *offset* (pc)

Load Word

動作

rx ⇐ {*offset* (Masked Base PC)}

コード



説明

8 ビット *offset* を 2 ビット左へシフトして、ゼロ拡張し、下位 2 ビットを 0 にマスクしたプログラムカウンタ (PC) 値に加算した結果が、実効アドレス (EA) になります。EA によってアドレス指定された 32 ビットの定数を、汎用レジスタ *rx* にロードします。

この命令により、32 ビットの定数をコード中に埋め込むことができます。この命令の近くに置かれた命令は、1 命令でこの定数を参照できます。

伸長後の 32 ビットコードのビット 25~21 は 0 で埋められます。32 ビット ISA の LW 命令はベースレジスタとして PC を使用できないので、16 ビット ISA の LW 命令は 32 ビット ISA の LW 命令と動作が異なります。

offset は 8 ビットで、2 ビットシフトすることにより扱うことのできる数値の範囲は、4 刻みで 0~1020 になります。この範囲外の値を指定すると、LW 命令は EXTEND 命令により拡張され、符号付きの 16 ビットの即値 (–32768 ~ +32767) を扱えるようになります。また、PC 相対アドレスを計算し、その結果を汎用レジスタに格納するための専用の命令 (ADDIUPC) も用意されています。

この命令では、PC 値をベース値として使用するので、その値をベース PC 値といいます。また、下位 2 ビットを 0 にマスクしたベース PC 値を、マスクベース PC 値といいます。命令が遅延スロット内に置かれているか、EXTEND 命令によって拡張されているかどうかによって、ベース PC 値は、以下のように異なります。

LW	ベース PC
JR・JALR 命令の遅延スロット内にある	JR・JALR 命令のアドレス
JAL・JALX 命令の遅延スロット内にある	JAL・JALX 命令の上位ハーフワードのアドレス
拡張されている	EXTEND コードのアドレス
拡張されていない (遅延スロット外)	LWPC 命令のアドレス

例外

アドレスエラー例外

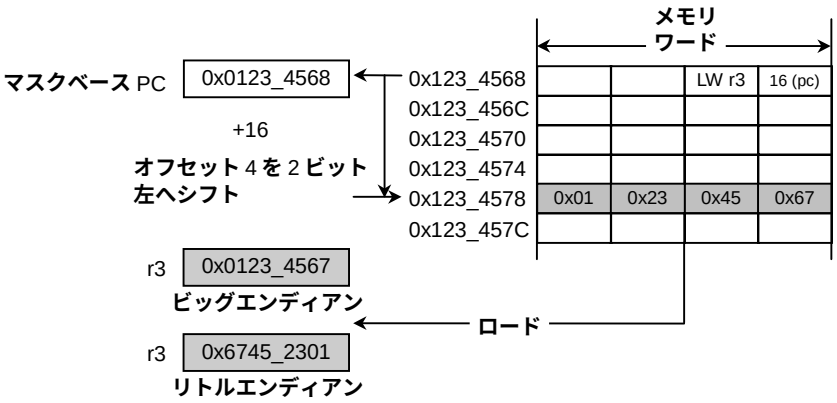
使用例

マスクベース PC がアドレス 0x0123_4568 を示し、アドレス 0x1234_5678~0x0123_457B の内容がそれぞれ 0x01、0x23、0x45、0x67 であるとします。

```
LW r3,16(pc)
```

以下に図示するように、オフセット値は伸長回路により 2 ビット左へシフトされるので、指定されたオフセット値 (16＝2 進数 0001_0000) は、アセンブラ・リンカにより、コード 4 (2 進数 0000_0100) に変換されます。したがって、上記のロード命令の命令コードは 0xB304 になります。

上記の命令を実行すると、レジスタ r3 には、ビッグエンディアンのときは 0x123_4567 がロードされ、リトルエンディアンのときは 0x6745_2301 がロードされます。



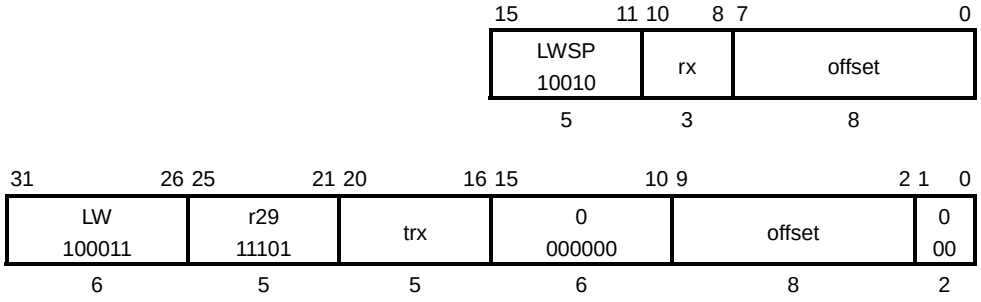
LW *rx*, *offset* (sp)

Load Word

動作

rx ⇐ {*offset* (sp)}

コード



説明

8 ビット *offset* を 2 ビット左へシフトして、ゼロ拡張し、スタックポインタレジスタ sp (r29) の内容に加算した結果が実効アドレス (EA) になります。EA によってアドレス指定されたワードデータを汎用レジスタ *rx* に格納します。

offset は 8 ビットで、2 ビットシフトすることにより扱うことのできる数値の範囲は、4 刻みで 0~1020 になります。この範囲外の値を指定すると、LW 命令は EXTEND 命令により拡張され、符号付きの 16 ビットの即値 (–32768 ~ +32767) を扱えるようになります。

例外

アドレスエラー例外

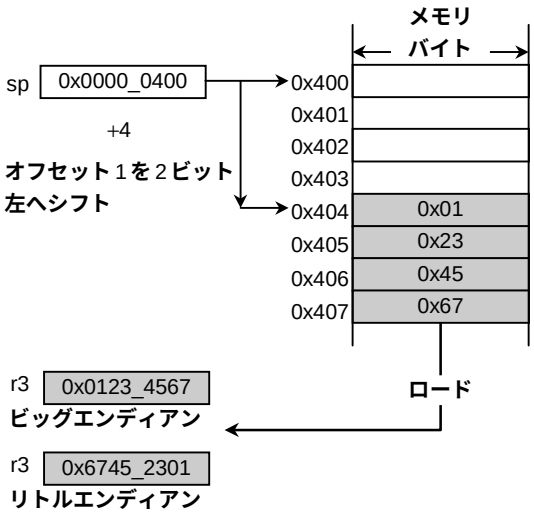
使用例

スタックポインタレジスタ sp がアドレス 0x0000_0400 を示し、アドレス 0x0404~0x0407 の内容が、0x01、 0x23、 0x45、 0x67 であるとします。

```
LW r3,4(sp)
```

以下に図示するように、オフセット値は伸長回路により 2 ビット左へシフトされるので、指定されたオフセット値 (4 = 2 進数 0100) は、アセンブラ・リンカにより、1 (2 進数 0001) に変換されます。したがって、上記の命令コードは 0x9301 になります。

上記の命令を実行するとレジスタ r3 には、ビッグエンディアンのときは 0x0123_4567 がロードされ、リトルエンディアンのときは 0x6745_23_01 がロードされます。


メモリ
バイト
sp
0x0000_0400
+4
オフセット 1 を 2 ビット
左へシフト
0x400
0x401
0x402
0x403
0x404
0x405
0x406
0x407
0x01
0x23
0x45
0x67
ロード
r3
0x0123_4567
ビッグエンディアン
r3
0x6745_2301
リトルエンディアン

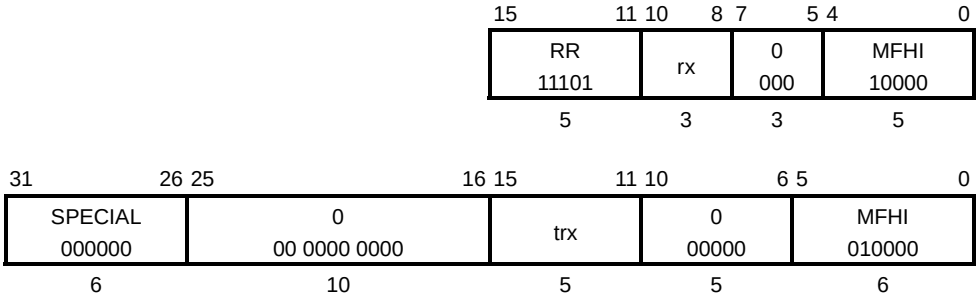
MFHI *rx*

Move From HI

動作

rx ⇐ HI

コード



説明

HI レジスタの内容を汎用レジスタ *rx* にロードします。

例外

なし

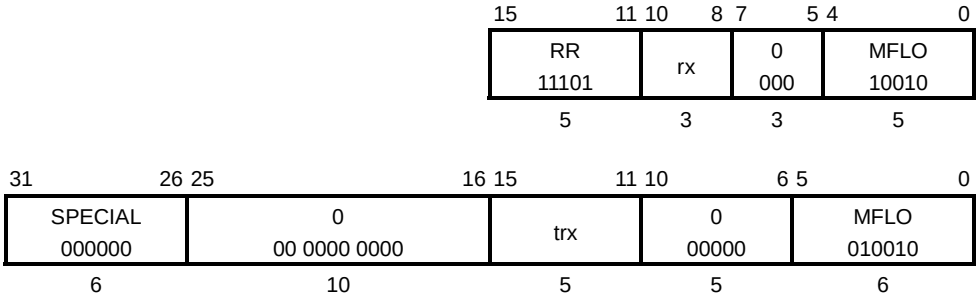
MFLO *rx*

Move From LO

動作

rx ⇐ LO

コード



説明

LO レジスタの内容を汎用レジスタ *rx* にロードします。

例外

なし

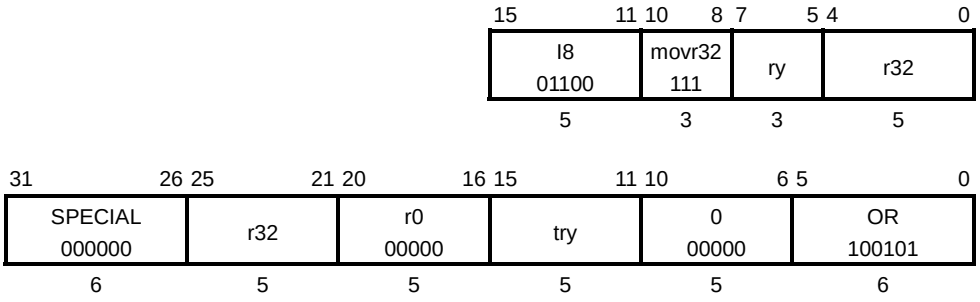
MOVE *ry*, *r32*

Move

動作

ry ⇐ *r32*

コード



説明

汎用レジスタ *r32* の内容を汎用レジスタ *ry* に格納します。*r32* には 32 本の汎用レジスタ (r0~r31) のうちのどれでも指定できます。*ry* には 16 ビット ISA でアクセスできる 8 本のレジスタのうちのいずれかを指定します。

16 ビット ISA では、32 本の汎用レジスタのうち r2～r7、r16、r17 の 8 本のレジスタしかアクセスできません。ただし、プロセッサには 32 ビット ISA モードで使用する 32 本のレジスタが存在するため、16 ビット ISA でアクセスできる 8 本のレジスタとその他の 24 本のレジスタ間でデータを移送するための MOVE 命令が 16 ビット ISA に用意されています。MOVE 命令により、16 ビット ISA モードでも、32 本のすべての汎用レジスタを使用できるようになります。

16 ビット命令の *r32* フィールドのビット配置は、32 ビット ISA と同じです。つまり、コード 00000 が r0、00001 が r1、00010 が r2、00011 が r3 になります。

例外

なし

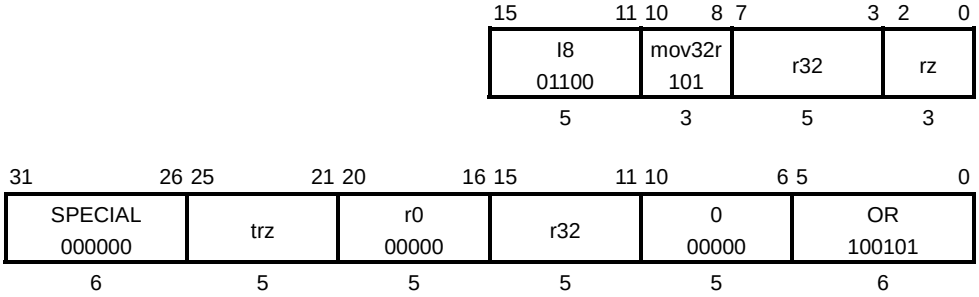
MOVE *r32*, *rz*

Move

動作

r32 ⇐ *rz*

コード



説明

汎用レジスタ *rz* の内容を汎用レジスタ *r32* に格納します。*rz* には 16 ビット ISA でアクセスできる 8 本のレジスタのうちのいずれかを指定します。*r32* には 32 本の汎用レジスタ (r0~r31) のうちのどれでも指定できます。

16 ビット ISA では、32 本の汎用レジスタのうち r2～r7、r16、r17 の 8 本のレジスタしかアクセスできません。ただし、プロセッサには 32 ビット ISA モードで使用する 32 本のレジスタが存在するため、16 ビット ISA でアクセスできる 8 本のレジスタとその他の 24 本のレジスタ間で値を移送するための MOVE 命令が 16 ビット ISA に用意されています。MOVE 命令により、16 ビット ISA モードでも、32 本のすべての汎用レジスタを使用できるようになります。

16 ビットコードの r32 フィールドのビット配置は、32 ビット ISA と異なります。r32 フィールドのビット配置は、[2:0] [4:3] のようになっています。コードとレジスタの対応を以下に示します。

コード	レジスタ
00000	r0
00001	r8
00010	r16
00011	r24
00100	r1
00101	r9
00110	r17
00111	r25
01000	r2
01001	r10
01010	r18
01011	r26
01100	r3
01101	r11
01110	r19
01111	r27

コード	レジスタ
10000	r4
10001	r12
10010	r20
10011	r28
10100	r5
10101	r13
10110	r21
10111	r29
11000	r6
11001	r14
11010	r22
11011	r30
11100	r7
11101	r15
11110	r23
11111	r31

例外

なし

MULT *rx*, *ry*

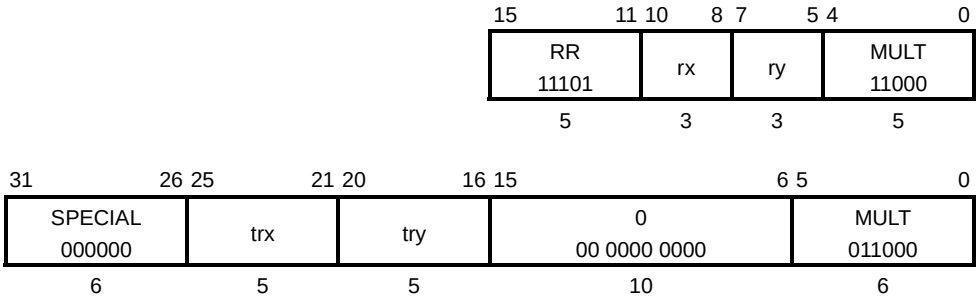
Multiply

動作

HI ⇐ (*rx* × *ry*) の上位ワード;

LO ⇐ (*rx* × *ry*) の下位ワード;

コード



説明

汎用レジスタ *rx* の内容と汎用レジスタ *ry* の内容を乗算します。*rx* と *ry* は符号付き整数として扱います。結果の上位ワードを HI レジスタに格納し、下位ワードを LO レジスタに格納します。

いかなる場合も、整数オーバフロー例外は発生しません。

例外

なし

使用例

汎用レジスタ r3 に 0x0123_4567 が、r4 に 0x89AB_CDEF が格納されているとします。

```
MULT r3,r4
```

このとき、上記の命令は、以下の演算を実行します。

(0x0123_4567 × 0x89AB_CDEF)
= 0xFF79_5E36_C94E_4629

結果の上位ワード 0xFF79_5E36 が HI レジスタに格納され、下位ワード 0xC94E_4629 が LO レジスタに格納されます。

MULTU *rx*, *ry*

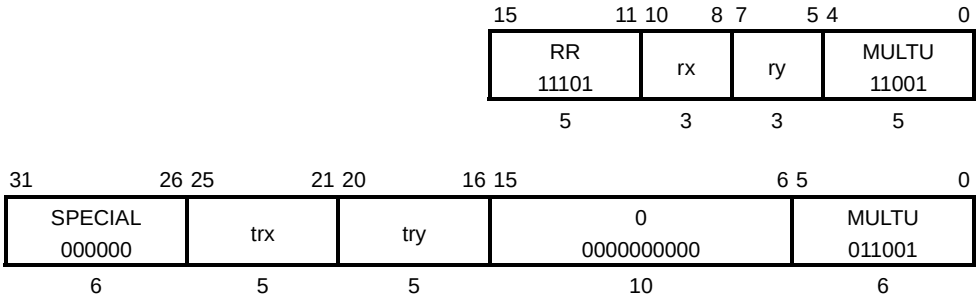
Multiply Unsigned

動作

HI ⇐ (*rx* × *ry*) の上位ワード;

LO ⇐ (*rx* × *ry*) の下位ワード;

コード



説明

汎用レジスタ *rx* の内容と汎用レジスタ *ry* の内容を乗算します。*rx* と *ry* は符号なし整数として扱います。結果の上位ワードを HI レジスタに格納し、下位ワードを LO レジスタに格納します。

いかなる場合も、整数オーバフロー例外は発生しません。

例外

なし

使用例

汎用レジスタ r3 に 0x0123_4567 が、r4 に 0x89AB_CDEF が格納されているとします。

```
MULTU r3,r4
```

このとき、上記の命令は、以下の演算を実行します。

(0x0123_4567 × 0x89AB_CDEF)
= 0x009C_A39D_C94E_4629

結果の上位ワード 0x009C_A39D が HI レジスタに格納され、下位ワード 0xC94E_4629 が LO レジスタに格納されます。

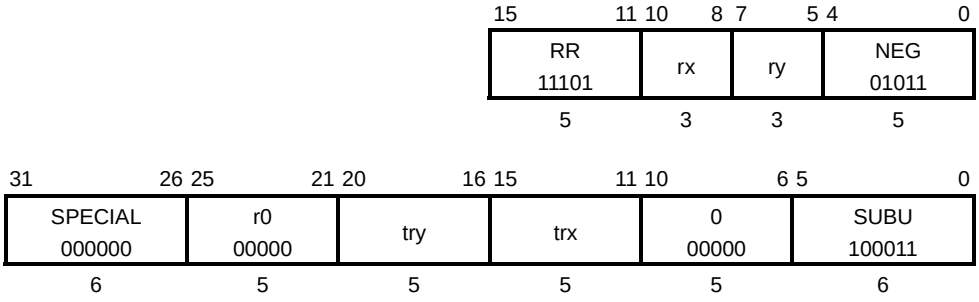
NEG *rx*, *ry*

Negate

動作

$rx = 0 - ry$

コード



説明

汎用レジスタ *ry* の内容の 2 の補数をとり、結果を汎用レジスタ *rx* に格納します。つまり、0 から *ry* の値を引いた結果が *rx* に格納されます。

例外

なし

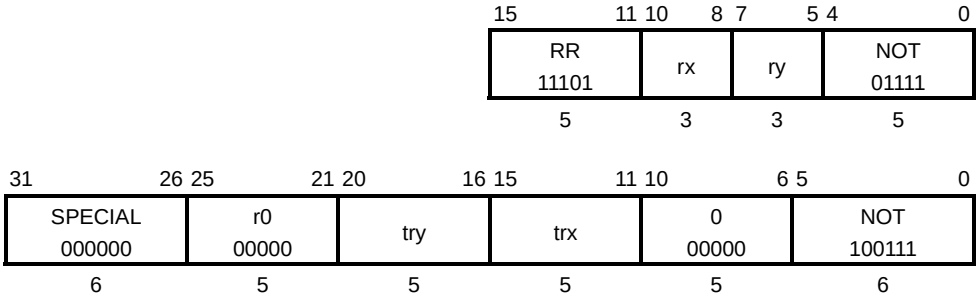
NOT *rx*, *ry*

NOT

動作

rx ⇐ *ry* NOR 0x0000_0000

コード



説明

汎用レジスタ *ry* の内容の 1 の補数をとり、結果を汎用レジスタ *rx* に格納します。*ry* の各ビットは反転されます。つまり、*rx* は *ry* と 0 の否定論理和 (NOR) をとった値となります。

例外

なし

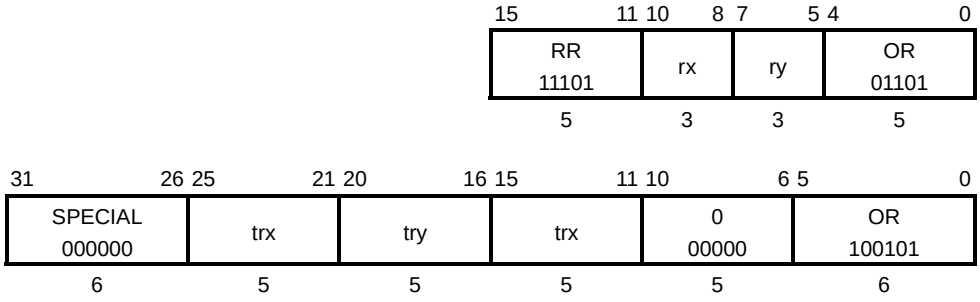
OR *rx*, *ry*

OR

動作

rx ⇐ *rx* OR *ry*

コード



説明

汎用レジスタ *rx* の内容と汎用レジスタ *ry* の内容の論理和 (OR) をとり、結果を汎用レジスタ *rx* に格納します。

例外

なし

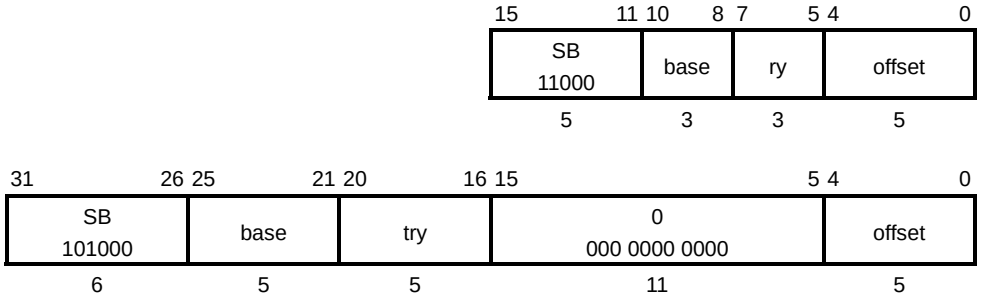
SB *ry*, *offset* (*base*)

Store Byte

動作

ry ⇒ {*offset* (*base*)}

コード



説明

5 ビット *offset* をゼロ拡張した値と汎用レジスタ *base* の内容を加算することにより、実効アドレス (EA) を生成します。汎用レジスタ *ry* の最下位バイトを、このアドレスにストアします。

ry の上位 3 バイトは無視されるため、符号付きと符号なしの区別はありません。

offset フィールドは 5 ビットで、扱うことのできる数値の範囲は、0~31 です。この範囲外の値を指定すると SB 命令は EXTEND 命令により拡張され、符号付きの 16 ビット即値 (–32768 ~ +32767) を扱えるようになります。

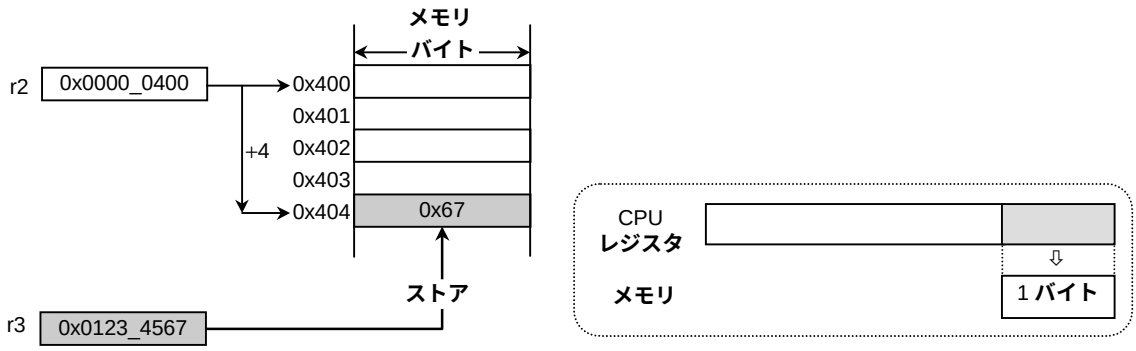
例外

アドレスエラー例外

使用例

レジスタ r2 の値が 0x0000_0400 で、r3 の値が 0x0123_4567 の場合、以下の命令を実行すると、0x67 がアドレス 0x404 に格納されます。

```
SB r3,4(r2)
```


メモリ
バイト
r2
0x0000_0400
0x400
0x401
0x402
0x403
0x404
+4
0x67
ストア
r3
0x0123_4567
CPU
レジスタ
メモリ
1 バイト

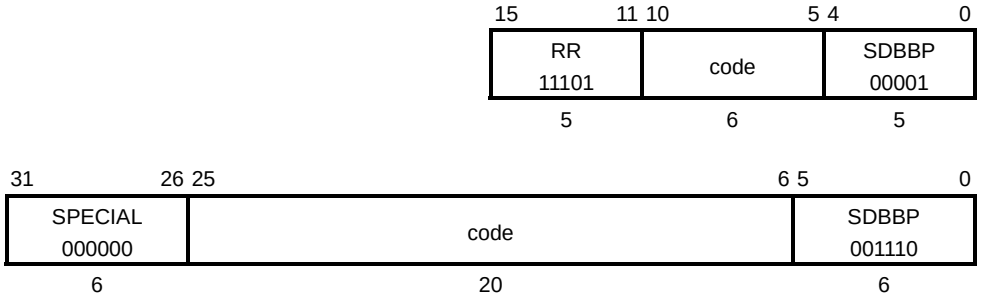
SDBBP *code*

Software Debug Breakpoint

動作

ソフトウエアデバッグブレークポイント例外

コード



説明

デバッグブレークポイントが発生し、無条件に制御を例外ハンドラに移します。

SDBBP 命令の *code* フィールドは、例外ハンドラに情報を渡すために使用できます。例外ハンドラが code フィールドを取り出すには、命令を含むメモリワードの内容をデータとしてロードする必要があります。詳細は「9.3　デバッグ例外」を参照してください。

デバッグ例外の処理中 (Debug レジスタの DM ビットが 1) は、SDBBP 命令は使用しないでください。DM=1 のとき、SDBBP 命令の動作は不定です。

SDBBP 命令は、開発システムで使用するので、ユーザープログラム中では使用しないでください。

例外

デバッグブレークポイント例外

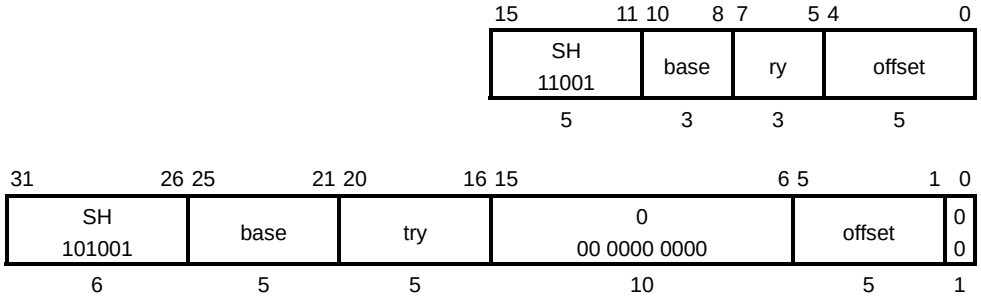
SH *ry*, *offset* (*base*)

Store Halfword

動作

ry ⇒ {*offset* (*base*)}

コード



説明

5 ビット *offset* を 1 ビット左へシフトして、ゼロ拡張し、汎用レジスタ *base* の内容に加算することにより、実効アドレス (EA) を生成します。汎用レジスタ *ry* の下位ハーフワードをこのアドレスにストアします。

ry の上位ハーフワードは無視されるので、符号付き、符号なしの区別はありません。

offset は 5 ビットで、1 ビットシフトすることにより扱うことのできる数値の範囲は、2 刻みで 0~62 になります。この範囲外の値を指定すると、SH 命令は EXTEND 命令により拡張され、符号付きの 16 ビットの即値 (–32768 ~ +32767) を扱えるようになります。この場合、offset はシフトされません。

例外

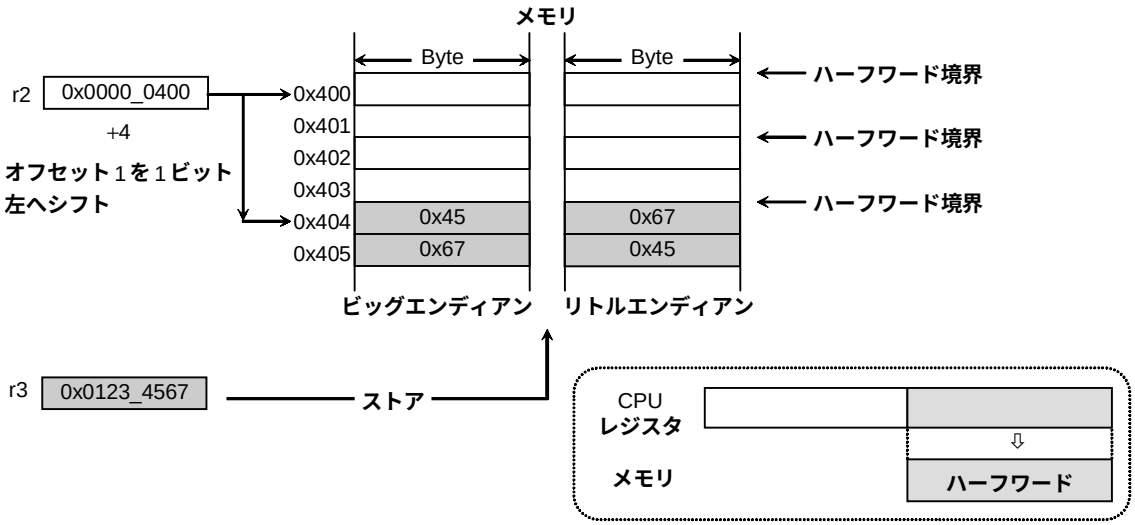
アドレスエラー例外

使用例

```
SH r3,4(r2)
```

レジスタ r2 の値が 0x0000_0400 で、r3 の値が 0x0123_4567 であるとします。オフセット値は伸長回路により 1 ビット左へシフトされるので、指定されたオフセット値 (4 = 2 進数 0100) は、アゼンブラ・リンカにより、2 (2 進数 0010) に変換されます。したがって、命令コードは 0xCA62 になります。

上記の命令を実行すると、ビッグエンディアンのときは、アドレス 0x404 に 0x45 が、アドレス 0x405 に 0x67 がストアされます。リトルエンディアンのときは、アドレス 0x404 に 0x67 が、アドレス 0x405 に 0x45 がストアされます。


メモリ
Byte
Byte
ハーフワード境界
ハーフワード境界
ハーフワード境界
r2
0x0000_0400
+4
オフセット 1 を 1 ビット
左へシフト
0x400
0x401
0x402
0x403
0x404
0x405
0x45
0x67
0x67
0x45
ビッグエンディアン
リトルエンディアン
r3
0x0123_4567
ストア
CPU
レジスタ
メモリ
ハーフワード

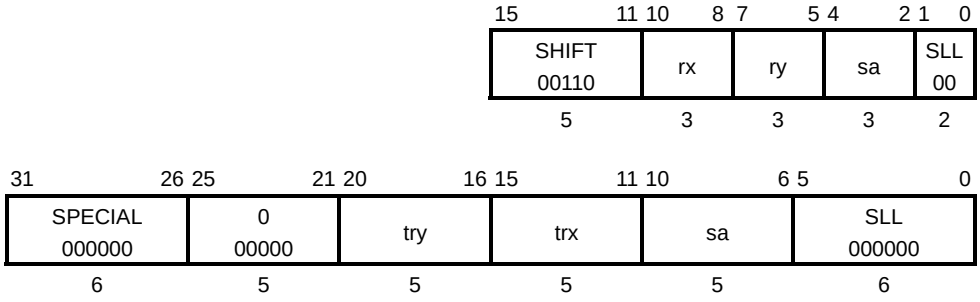
SLL *rx*, *ry*, *sa*

Shift Left Logical

動作

$rx \Leftarrow ry << sa$

コード



説明

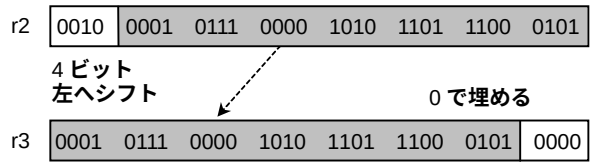
汎用レジスタ *ry* の 32 ビットの内容を *sa* ビット左へシフトし、右端の空いたビットを 0 で埋め、結果を汎用レジスタ *rx* に格納します。*sa* フィールドは 3 ビットです。したがって *sa* で指定できる数値の範囲は 1~8 です。000 は 8 ビットのシフトとして定義されています。

この範囲外のシフト量を指定すると、SLL 命令は EXTEND 命令により拡張され、sa フィールドは 5 ビットになり、0~31 のシフト量が扱えるようになります。

使用例

レジスタ r2 の内容が 0x2170_ADC5 の場合、以下の命令を実行すると、レジスタ r3 に 0x170A_DC50 が格納されます。

```
SLL r3,r2,4
```



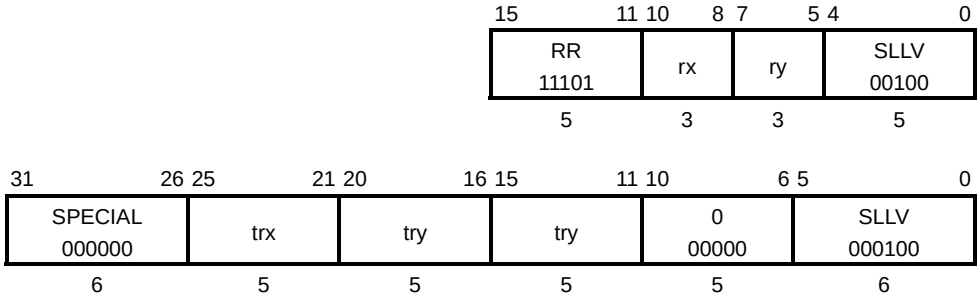
SLLV *ry*, *rx*

Shift Left Logical Variable

動作

ry << *rx* の下位 5 ビット

コード



説明

汎用レジスタ *ry* の 32 ビットの内容を、汎用レジスタ *rx* の下位 5 ビットで指定されたビット数、左へシフトし、右側の空いたビットを 0 で埋めます。結果を汎用レジスタ *ry* に格納します。

例外

なし

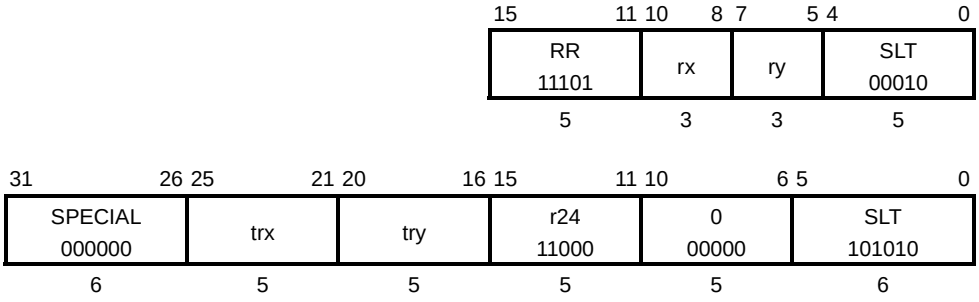
SLT *rx*, *ry*

Set On Less Than

動作

if *rx* < *ry* then t8 ⇐ 1; else t8 ⇐ 0

コード



説明

汎用レジスタ *rx* の内容と汎用レジスタ *ry* の内容を、符号付き整数として比較します。*rx* が *ry* より小さい場合は、コンディションコードレジスタ t8 (r24) に 1 を、そうでない場合は、t8 に 0 を格納します。

いかなる場合も、整数オーバフロー例外は発生しません。比較の際に行われる減算でオーバフローしても、比較は有効です。

例外

なし

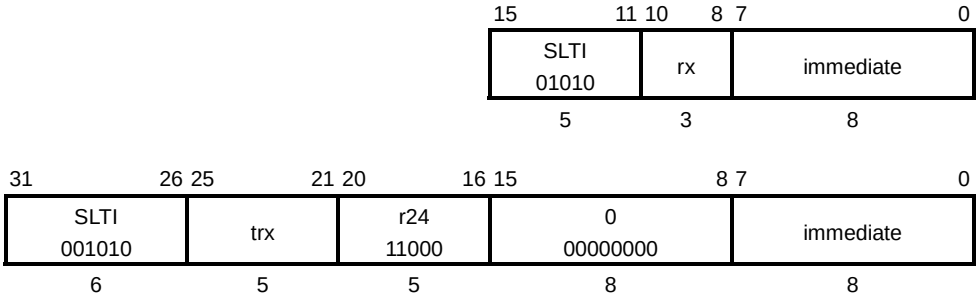
SLTI *rx*, *immediate*

Set On Less Than Immediate

動作

if *rx* < *immediate* then t8 ⇐ 1; else t8 ⇐ 0

コード



説明

8 ビット *immediate* をゼロ拡張した値と汎用レジスタ *rx* の内容を符号付き整数として比較します。*rx* が *immediate* より小さい場合、コンディションコードレジスタ t8 (r24) に 1 を、そうでない場合は 0 を格納します。

いかなる場合も、整数オーバフロー例外は発生しません。比較の際に行われる減算でオーバフローしても、比較は有効です。

immediate は 8 ビットで、扱うことのできる数値の範囲は 0~255 です。この範囲外の値を指定すると、SLTI 命令は EXTEND 命令により拡張され、符号付きの 16 ビットの即値を (–32768 ~ +32767) を扱えるようになります。

例外

なし

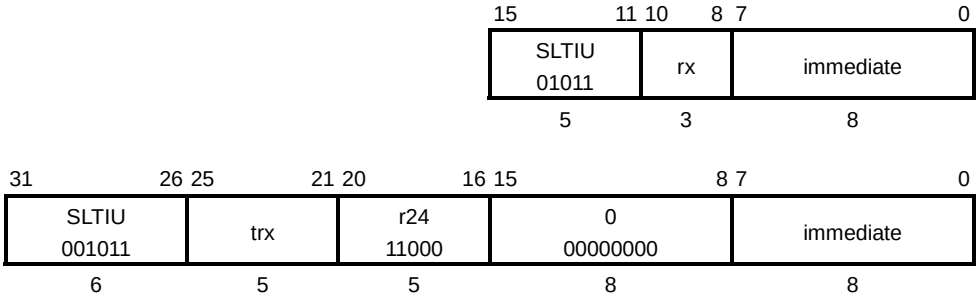
SLTIU *rx*, *immediate*

Set On Less Than Immediate Unsigned

動作

if *rx* < *immediate* then t8 ⇐ 1; else t8 ⇐ 0

コード



説明

8 ビット *immediate* をゼロ拡張した値と汎用レジスタ *rx* の内容を、符号なし整数として比較します。*rx* が *immediate* より小さい場合は、コンディションコードレジスタ t8 (r24) に 1 を、そうでない場合は、0 を格納します。

いかなる場合も、整数オーバフロー例外は発生しません。比較の際に行われる減算でオーバフローしても、比較は有効です。

immediate は 8 ビットで、扱うことのできる数値の範囲は 0~255 です。この範囲外の値を指定すると、SLTIU 命令は EXTEND 命令により拡張され、符号付きの 16 ビットの即値を (–32768 ~ +32767) を扱えるようになります。

例外

なし

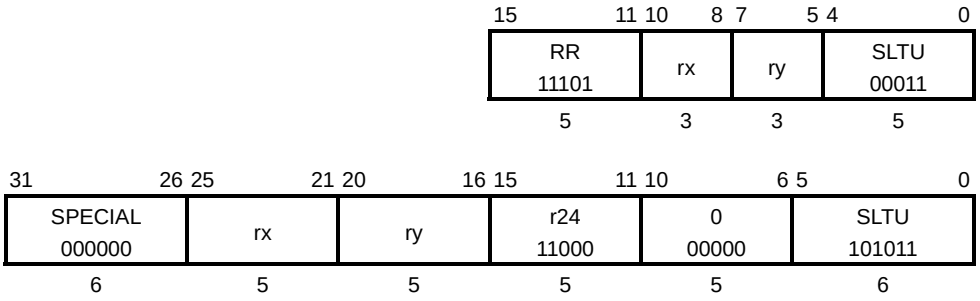
SLTU *rx*, *ry*

Set On Less Than Unsigned

動作

if *rx* < *ry* then t8 ⇐ 1; else t8 ⇐ 0

コード



説明

汎用レジスタ *rx* の内容と汎用レジスタ *ry* の内容を、符号なし整数として比較します。*rx* が *ry* より小さい場合は、コンディションコードレジスタ t8 (r24) に 1 を、そうでない場合は、0 を格納します。

いかなる場合も、整数オーバフロー例外は発生しません。比較の際に行われる減算でオーバフローしても、比較は有効です。

例外

なし

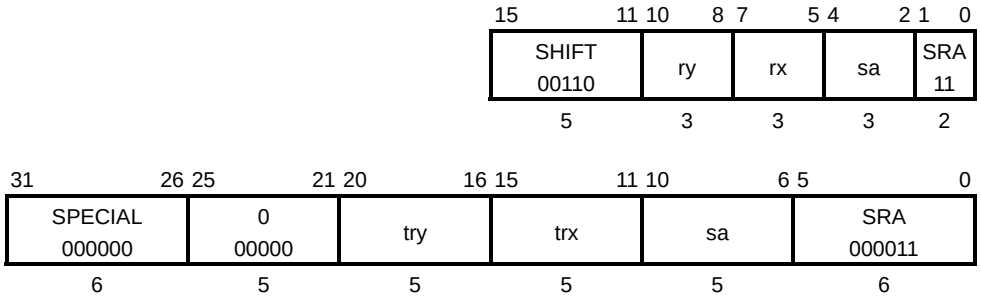
SRA *rx*, *ry*, *sa*

Shift Right Arithmetic

動作

rx ⇐ *ry* >> *sa*

コード



説明

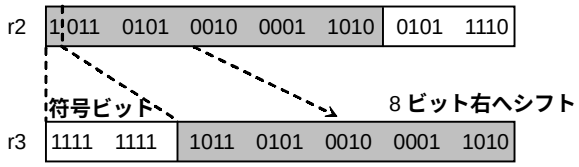
汎用レジスタ *ry* の 32 ビットの内容を *sa* ビット右へシフトし、左端の空いたビットを符号ビットで埋め、結果を汎用レジスタ *ry* に格納します。*sa* フィールドは 3 ビットです。したがって *sa* で指定できる数値の範囲は 1~8 です。000 は 8 ビットのシフトとして定義されています。

この範囲外のシフト量を指定すると、SRA 命令は EXTEND 命令により拡張され、sa フィールドは 5 ビットになり、0~31 のシフト量が扱えるようになります。

使用例

レジスタ r2 の内容が 0xB521_AE5E の場合、以下の命令を実行すると、レジスタ r3 に 0xFFB5_21AE が格納されます。

```
SRA r3,r2,8
```



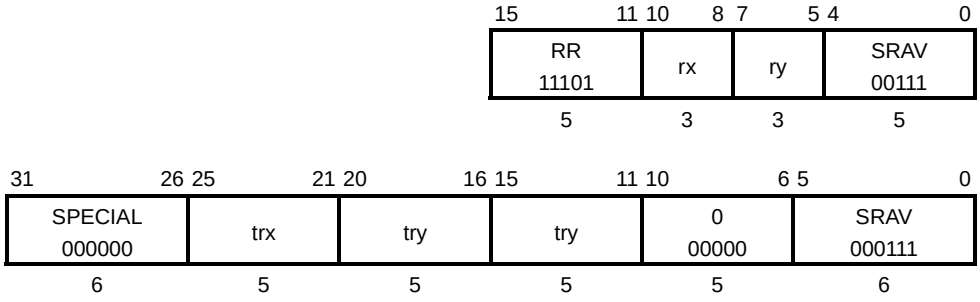
SRAV *ry*, *rx*

Shift Right Arithmetic Variable

動作

ry >> *rx* の下位 5 ビット

コード



説明

汎用レジスタ *ry* の 32 ビットの内容を、汎用レジスタ *rx* の下位 5 ビットで指定されたビット数、右にシフトし、左端の空いたビットを符号ビットで埋めます。結果を汎用レジスタ *ry* に格納します。

例外

なし

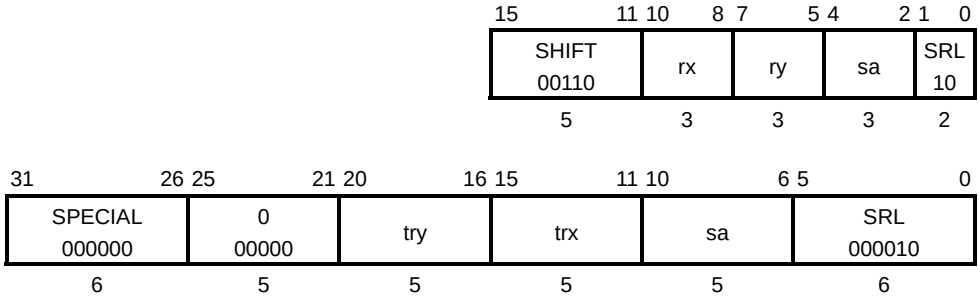
SRL *rx*, *ry*, *sa*

Shift Right Logical

動作

rx ⇐ *ry* >> *sa*

コード



説明

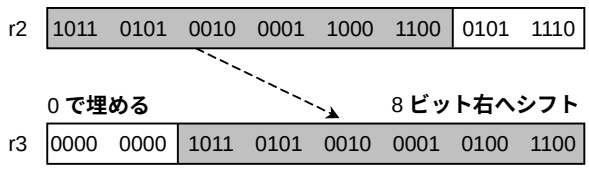
汎用レジスタ *ry* の 32 ビットの内容を *sa* ビット右へシフトし、左端の空いたビットを 0 で埋め、結果を汎用レジスタ *rx* に格納します。*sa* フィールドは 3 ビットです。したがって *sa* で指定できる数値の範囲は 1~8 です。000 は 8 ビットのシフトとして定義されています。

この範囲外のシフト量を指定すると、SRL 命令は EXTEND 命令により拡張され、sa フィールドは 5 ビットになり、0~31 のシフト量が扱えるようになります。

使用例

レジスタ r2 の内容が 0xB521_4C5E の場合、以下の命令を実行すると、レジスタ r3 に 0x00B5_214C が格納されます。

```
SRL r3,r2,8
```



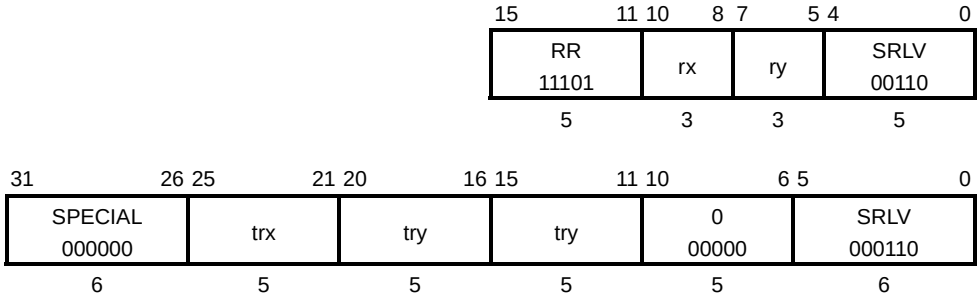
SRLV *ry*, *rx*

Shift Right Logical Variable

動作

ry >> 5 LSBs of *rx*

コード



説明

汎用レジスタ *ry* の 32 ビットの内容を、汎用レジスタ *rx* の下位 5 ビットで指定されたビット数、右にシフトし、左端の空いたビットを 0 で埋めます。結果を汎用レジスタ *ry* に格納します。

使用例

なし

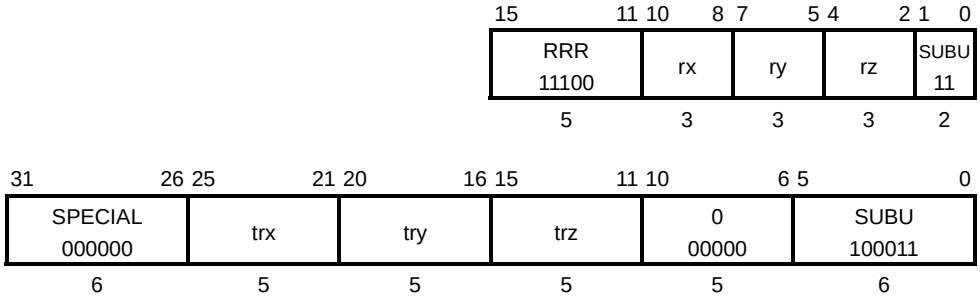
SUBU *rz*, *rx*, *ry*

Subtract Unsigned

動作

$rz \Leftarrow rx - ry$

コード



説明

汎用レジスタ *rx* の内容から汎用レジスタ *ry* の内容を減算し、結果を汎用ジレスタ *rz* に格納します。

整数オーバフロー例外は絶対発生しません。

例外

なし

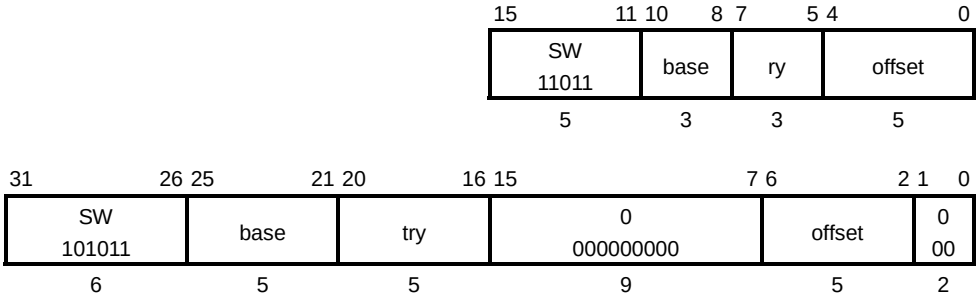
SW *ry*, *offset* (*base*)

Store Word

動作

ry ⇒ {*offset* (*base*)}

コード



説明

5 ビット *offset* を 2 ビット左へシフトして、ゼロ拡張し、汎用レジスタ *base* の内容に加算することにより実効アドレス (EA) を生成します。汎用レジスタ *ry* の内容を、このアドレスに格納します。

offset は 5 ビットで、2 ビット左へシフトすることにより扱うことのできる数値の範囲は、4 刻みで 0~124 になります。この範囲外の値を指定すると、SW 命令は EXTEND 命令により拡張され、符号付きの 16 ビットの即値 (–32768 ~ +32767) を扱えるようになります。この場合、*offset* はシフトされません。

例外

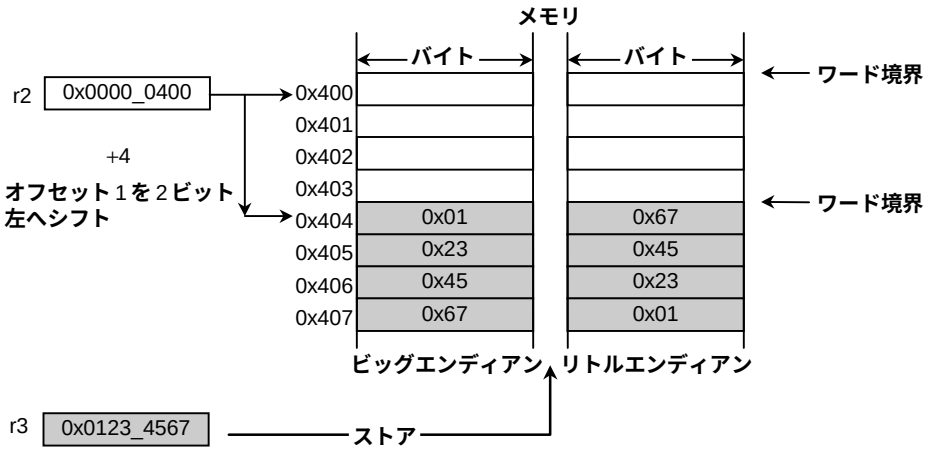
アドレスエラー例外

使用例

```
SW r3,4(r2)
```

レジスタ r2 の内容が 0x0000_0400 で、レジスタ r3 の内容が 0x0123_4567 であるとします。オフセット値は、伸長回路により 2 ビット左へシフトされるので、指定されたオフセット値 (4 = 2 進数 0100) は、アセンブラ・リンカにより、1 (2 進数 0001) に変換されます。したがって、上記のストア命令の命令コードは 0xDAE1 になります。

上記の命令を実行すると、ビッグエンディアンのときは、0x01、0x23、0x45、0x67 がアドレス 0x404~0x407 にストアされます。リトルエンディアンのときは、0x67、0x45、0x23、0x01 がアドレス 0x404~0x407 にストアされます。


メモリ
バイト
バイト
ワード境界
r2
0x0000_0400
0x400
0x401
0x402
0x403
+4
オフセット 1 を 2 ビット
左へシフト
0x404
0x405
0x406
0x407
0x01
0x23
0x45
0x67
0x67
0x45
0x23
0x01
ワード境界
ビッグエンディアン
リトルエンディアン
r3
0x0123_4567
ストア

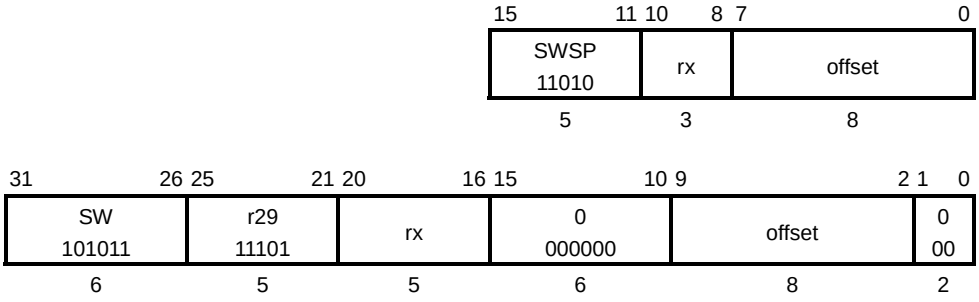
SW *rx*, *offset* (sp)

Store Word

動作

rx ⇒ {*offset* (sp)}

コード



説明

8 ビット *offset* を 2 ビット左へシフトして、ゼロ拡張し、スタックポインタレジスタ sp (r29) の内容に加算することにより実効アドレス (EA) を生成します。汎用レジスタ *rx* の内容をこのアドレスに格納します。

offset は 8 ビットで、2 ビット左へシフトすることにより扱うことのできる数値の範囲は、4 刻みで 0~1020 になります。この範囲外の値を指定すると、SW 命令は EXTEND 命令により拡張され、符号付きの 16 ビットの即値 (–32768 ~ +32767) を扱えるようになります。

例外

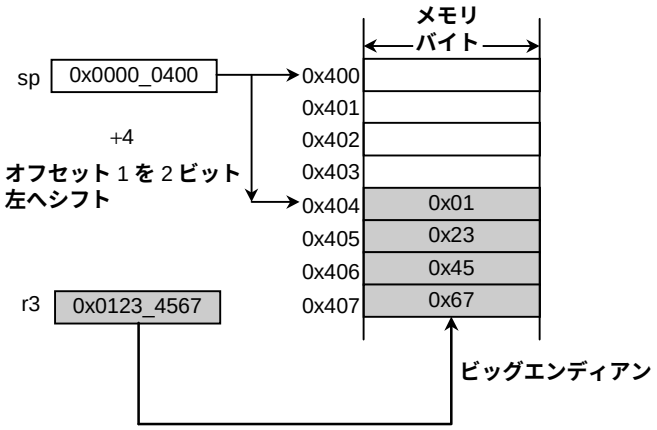
アドレスエラー例外

使用例

```
SW r3,4(sp)
```

レジスタ sp の内容が 0x0000_0400 で、レジスタ r3 の内容が 0x0123_4567 であるとします。オフセット値は伸長回路により 2 ビット左へシフトされるので、オフセット値 (4 = 2 進数 0100) は、アセンブラ・リンカにより、1 (2 進数 0001) に変換されます。したがって、上記のストア命令の命令コードは 0xD301 になります。

ビッグエンディアンのときは、アドレス 0x404~0x407 に 0x1234_4567 がストアされます。


メモリ
バイト
sp
0x0000_0400
0x400
0x401
0x402
0x403
0x404
0x405
0x406
0x407
+4
オフセット 1 を 2 ビット
左へシフト
0x01
0x23
0x45
0x67
r3
0x0123_4567
ビッグエンディアン

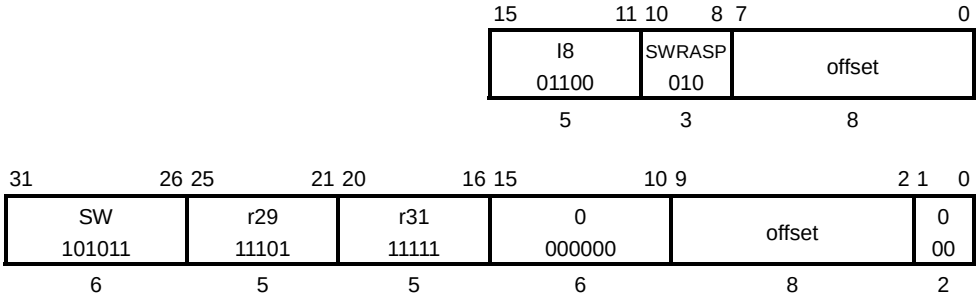
SW ra, *offset* (sp)

Store Word

動作

ra ⇒ {*offset* (sp)}

コード



説明

8 ビット *offset* を 2 ビット左へシフトして、ゼロ拡張し、スタックポインタレジスタ sp (r29) の内容に加算することにより実効アドレス (EA) を生成します。リンクレジスタ ra (r31) の内容をこのアドレスに格納します。

offset は 8 ビットで、2 ビット左へシフトすることにより扱うことのできる数値の範囲は、4 刻みで 0~1020 になります。この範囲外の値を指定すると、SW 命令は EXTEND 命令により拡張され、符号付きの 16 ビットの即値 (–32768 ~ +32767) を扱えるようになります。

例外

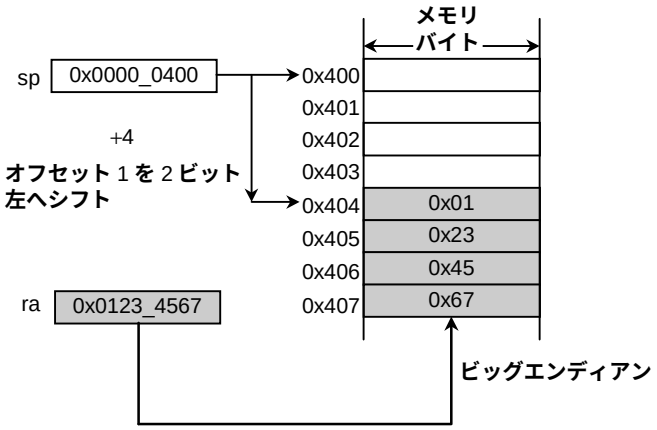
アドレスエラー例外

使用例

```
SW ra,4(sp)
```

レジスタ sp の内容が 0x0000_0400 で、レジスタ ra の内容が 0x0123_4567 であるとします。オフセット値は伸長回路により 2 ビット左へシフトされるので、オフセット (4 = 2 進数 0100) は、アセンブラ・リンカにより、1 (2 進数 0001) に変換されます。したがって、上記のストア命令の命令コードは 0x3101 になります。

ビッグエンディアンのときは、アドレス 0x404~0x407 に 0x1234_4567 がストアされます。


メモリ
バイト
sp
0x0000_0400
0x400
0x401
0x402
0x403
0x404
0x405
0x406
0x407
+4
オフセット 1 を 2 ビット
左へシフト
0x01
0x23
0x45
0x67
ra
0x0123_4567
ビッグエンディアン

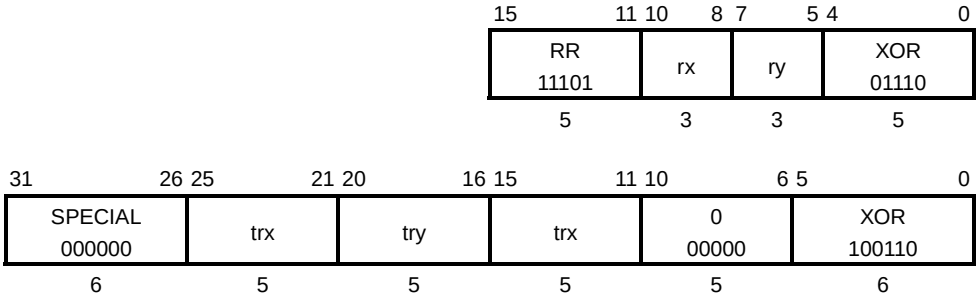
XOR *rx*, *ry*

Exclusive OR

動作

rx ⇐ *rx* XOR *ry*

コード



説明

汎用レジスタ *rx* の内容と汎用レジスタ *ry* の内容との排他的論理和 (XOR) をとり、結果を汎用レジスタ *rx* に格納します。

例外

なし

付録C　プログラミング制約

TX19 のようなパイプラインを備えた CPU では、パイプラインのスムーズな流れを乱す命令があります。この章では、アセンブリ言語プログラムで守らなければならない制約について説明します。

C.1　32 ビット ISA の制約

表 C-1　ロードストア命令

命令	制　約
LH　*rt*, *offset*(*base*) LHU　*rt*, *offset*(*base*) SH　*rt*, *offset*(*base*)	これらの命令で生成されるターゲットアドレスは、ハーフワード境界に位置合わせされていなければなりません。つまり、最下位ビットが 0 でなければなりません。0 でない場合、アドレスエラー例外が発生します。
LW　*rt*, *offset*(*base*) LWU　*rt*, *offset*(*base*) SW　*rt*, *offset*(*base*)	これらの命令で生成されるターゲットアドレスは、ワード境界に位置合わせされていなければなりません。つまり、下位 2 ビットが 0 でなければなりません。下位 2 ビットが 0 でない場合、アドレスエラー例外が発生します。

表 C-2　ジャンプ命令

命令	制　約
JALR　(*rd*,) *rs*	• *rd* として *rs* を指定できません。これは例外処理後、ジャンプ命令から実行を再開できなくなってしまうからです。 • 32 ビット ISA では、命令はすべてワード境界で位置合わせされなければなりません。したがって、32 ビット ISA モードのルーチンにジャンプするときは、ターゲットレジスタ (*rs*) の下位 2 ビットは 0 でなければなりません。下位 2 ビットが 0 でない場合、ジャンプ先の命令をフェッチするときにアドレスエラー例外が発生します。
JR　*rs*	32 ビット ISA では、命令はすべてワード境界に位置合わせされていなければなりません。そのため、32 ビット ISA モードのルーチンにジャンプするときは、ターゲットレジスタ (*rs*) の下位 2 ビットは 0 でなければなりません。下位 2 ビットが 0 でない場合、ジャンプ先の命令をフェッチするときにアドレスエラー例外が発生します。
すべてのジャンプ命令	ジャンプ命令はすべて、ジャンプ・分岐遅延スロットに置いてはいけません。ジャンプ・分岐遅延スロットに置いた場合、ジャンプ命令の動作は不定になります。

表 C-3　分岐・分岐ライクリ命令

命令	制　約
BGEZAL(L)　*rs*, *offset* BLTZAL(L)　*rs*, *offset*	レジスタ *rs* に r31 を指定できません。*rs* として r31 を指定すると、例外処理後、分岐命令から実行を再開できなくなってしまうからです。
すべての分岐命令	分岐命令はすべて、ジャンプ・分岐遅延スロットに置いてはいけません。ジャンプ・分岐遅延スロットに置いた場合、分岐命令の動作は不定になります。

表 C-4　システム制御コプロセッサ (CP0) 命令

命令	制　約
CACHE　*op*, *offset*(*base*) DERET MTC0　*rt*, *rd* MFC0　*rt*, *rd* RFE	Status レジスタの CU[0] ビットがクリアされている場合、ユーザーモードでこれらの命令を実行すると、コプロセッサ使用不可例外が発生します。カーネルモードでは、CU[0] ビットの設定に関係なく、これらの命令を実行できます。
DERET	• DERET 命令の遅延スロットには、NOP 命令を置かなければなりません。 • デバッグモードでないとき(Debug レジスタの DM ビットが 0 のとき) は、DERET 命令の動作は不定です。 • MTC0 命令を使って DEPC レジスタに戻りアドレスを設定する場合は、デバッグ例外ハンドラで少なくとも 2 つの命令を実行してから、DERET 命令を実行しなければなりません。 • MTC0 命令で Debug レジスタに書き込んだ直後、または MFC0 命令を使って Debug レジスタを読み出した直後に、DERET 命令を実行することは禁止されています。実行した場合は、Debug レジスタの値は不定になります。
MTC0　*rt*, *rd*	• RFE 命令の直前に、MTC0 命令により Status レジスタに書き込むことは禁止されています。そのような書き込みを行うと、Status レジスタの内容は不定になります。 • DERET 命令の直前に、MTC0 命令により Debug レジスタに書き込むことは禁止されています。そのような書き込みを行うと、Debug レジスタの内容は不定になります。
MFC0　*rt*, *rd*	• RFE 命令の直前に、MFC0 命令により Status レジスタの内容を読み出すことは禁止されています。そのような読み出しを行うと、Status レジスタの内容は不定になります。 • DERET 命令の直前に、MFC0 命令により Debug レジスタの内容を読み出すことは禁止されています。そのような読み出しを行うと、Debug レジスタの内容は不定になります。 • MFC0 命令には、遅延スロットがあります。
RFE	• MTC0 命令で Status レジスタに書き込みを行った直後、または MFC0 命令でレジスタから読み出しを行った直後に、RFE 命令を実行することは禁止されています。実行した場合、Status レジスタの内容は不定になります。 • RFE 命令の実行中に割り込みが発生すると、Status レジスタの内容は保証できません。したがって、RFE 命令を実行するまえに、すべての割り込みを禁止しなければなりません。

表 C-5　コプロセッサ命令

命令	制　約
BC*z*F(L)　*offset* BC*z*T(L)　*offset* CFC*z*　*rt*, *rd* CTC*z*　*rt*, *rd* COP*z*　*cofun* MFC*z*　*rt*, *rd* MTC*z*　*rt*, *rd*	Status レジスタの CU ビットがクリアされている場合、対応するコプロセッサユニットに対してこれらの命令を実行すると、コプロセッサ使用不可例外が発生します。

表 C-6　特殊命令

命令	制　約
SDBBP	デバッグ例外の処理中 (Debug レジスタの DM ビットが 1) は、SDBBP 命令は使用しないでください。DM = 1 のとき、SDBBP 命令の動作は不定です。

C.2　16 ビット ISA の制約

表 C-7　ロード・ストア命令

命令	制　約
LH *ry*, *offset*(*base*) LHU *ry*, *offset*(*base*) SH *ry*, *offset*(*base*)	これらの命令で生成されるターゲットアドレスは、ハーフワード境界で位置合わせされていなければなりません。つまり、最下位ビットが 0 にクリアされていなければなりません。0 でない場合、アドエスエラー例外が発生します。

LW *ry*, *offset*(*base*)
SW *ry*, *offset*(*base*)
SW *ry*, *offset*(sp)
SW *ra*, *offset*(sp)
	これらの命令で生成されるターゲットアドレスは、ワード境界に位置合わせされていなければなりません。つまり、下位 2 ビットが 0 でなければなりません。下位 2 ビットが 0 でない場合、アドレスエラー例外が発生します。

表 C-8　ジャンプ命令

命令	制　約
JALR ra, *rx*	• レジスタ *rx* に ra を指定できません。*rx* として ra を指定すると、例外処理後、ジャンプ命令から実行を再開できなくなってしまうからです。 • 32 ビット ISA では、命令はすべてワード境界に位置合わせされていなければなりません。そのため、32 ビット ISA モードのルーチンにジャンプするときは、ターゲットレジスタ (*rx*) の下位 2 ビットは 0 でなければなりません。下位 2 ビットが 0 でない場合、ジャンプ先の命令をフェッチするときにアドレスエラー例外が発生します。
JR *rx*	32 ビット ISA では、命令はすべてワード境界に位置合わせされていなければなりません。そのため、32 ビット ISA モードのルーチンにジャンプするときは、ターゲットレジスタ (*rx*) の下位 2 ビットは 0 でなければなりません。下位 2 ビットが 0 でない場合、ジャンプ先の命令をフェッチするときにアドレスエラー例外が発生します。
JR ra	32 ビット ISA では、命令はすべてワード境界に位置合わせされていなければなりません。そのため、32 ビット ISA モードのルーチンにジャンプするときは、ra の下位 2 ビットは 0 でなければなりません。下位 2 ビットが 0 でない場合、ジャンプ先の命令をフェッチするときにアドレスエラー例外が発生します。
すべてのジャンプ命令	ジャンプ命令は、遅延スロットに置いてはいけません。

表 C-9　分岐命令

命令	制　約
すべての分岐命令	分岐命令は、ジャンプ遅延スロットに置いてはいけません。

表 C-10　特殊命令

命令	制　約
SDBBP	デバッグ例外の処理中 (Debug レジスタの DM ビットが 1 のとき) は、SDBBP 命令は使用しないでください。DM = 1 のとき、SDBBP 命令の動作は不定です。

表 C-11　EXTEND 命令

命令	制　約
すべての EXTEND 命令	EXTEND 命令で拡張された命令は、ジャンプ遅延スロットに置いてはいけません。

付録D　TX19・TX39・R3000A の共通点

表 D-1に当社 TX19 と TX39 の相違点を示します。

表 D-1　TX19 と TX39 の比較

特　長	TX19		TX39
アプリケーション	低消費電力、高コード効率		高速
命令セット	32 ビット ISA	16 ビット ISA	• 32 ビット固定 • 84 命令
	• TX39 とオブジェクトレベルで上位互換性があります。 • ISA モードを切り換える JALX 命令を含む 85 命令	• ダブルワード命令と LWU 命令を除き、MIPS16 ASE とオブジェクトレベルで上位互換があります。 • 58 命令	
CPU レジスタ	• TX39 と同じ • PC の最下位ビットが ISA モードを表します。		• 32 本の汎用レジスタ • プログラムカウンタ (PC) • 2 本の乗除算レジスタ (HI・LO) すべての CPU レジスタは 32 ビットです。
CP0 レジスタ	割り込み制御 (IE) レジスタが追加され、割り込み許可・禁止が 1 命令で記述可能です。		• 1 本のシステム構成レジスタ • 6 本の一般例外処理レジスタ • 2 本のデバッグ例外処理レジスタ すべての CPU レジスタは 32 ビットです。
	TX19 と TX39 では、以下のレジスタの構成が異なります。		
	PRId[15:8] Implementation=0x2C Cause[11:8] Sw Cause[15:13] IL Status[11:8] SWiMask Status[15:13] CMask Status[18:16] PMask Status[25] 予約		PRId[15:8] Implementation=0x22 Cause[9:8] Sw Cause[15:10] IP Status[9:8] IntMask (Sw) Status[15:10] IntMask (Int) Status[18:16] 0 Status[25] RE
バインプライン	5 段		5 段
乗算命令	レイテンシ / 実行 = 2/ 1 サイクル		レイテンシ / 実行 = 2 / 1 サイクル
除算命令	レイテンシ / 実行 = 35 / 34 サイクル 除算命令が終了するまえに、MFHI (Move From HI) 命令、MFLO (Move From LO) 命令で除算結果を読もうとすると、パイプラインがストールします。		レイテンシ / 実行 = 35 / 34 サイクル 除算命令が終了するまえに、MFHI (Move From HI) 命令、MFLO (Move From LO) 命令で除算結果を読もうとすると、除算命令は中止されます。
積和命令	レイテンシ / 実行 = 2 / 1 サイクル		レイテンシ / 実行 = 2 / 1 サイクル
割り込み応答	• 割り込み要求は、ハードウエアで処理されます。 • 例外と割り込みは別々のアドレスです。 • 割り込みマスクレベルはハードウエアで自動的に更新されます。 • 割り込みスタック用の 1 クロックアクセス RAM を内蔵できます。		• 割り込み要求は、ソフトウエアで処理されます。 • 例外と割り込みのアドレスが共通です。 • 割り込みマスクレベルは、ソフトウエアで更新が必要です。

特　長	TX19	TX39
マスカブル割り込み	• 4 本のソフトウエア割り込み • 1 本の割り込みコントローラからのハードウエア割り込み (7 つの割り込みレベル)	• 2 本のソフトウエア割り込み • 6 本のハードウエア割り込み
仮想アドレス空間	4 G バイト	4 G バイト
クロック周波数	20 MHz (標準バージョン)、高速バージョンは開発中	70 MHz

表 D-2に TX19 (32 ビット ISA)、TX39、MIPS R300A の命令セットの比較を示します。相違点は網かけで示してあります。

表 D-2　TX19、TX39、MIPS R300A の命令セット

分　類	命　令	TX19 32 ビット ISA		TX39		R3000A	
ロード・ストア命令	Load Byte	LB	*rt*, *offset*(*base*)	LB	*rt*, *offset*(*base*)	LB	*rt*, *offset*(*base*)
	Load Byte Unsigned	LBU	*rt*, *offset*(*base*)	LBU	*rt*, *offset*(*base*)	LBU	*rt*, *offset*(*base*)
	Load Halfword	LH	*rt*, *offset*(*base*)	LH	*rt*, *offset*(*base*)	LH	*rt*, *offset*(*base*)
	Load Halfword Unsigned	LHU	*rt*, *offset*(*base*)	LHU	*rt*, *offset*(*base*)	LHU	*rt*, *offset*(*base*)
	Load Word	LW	*rt*, *offset*(*base*)	LW	*rt*, *offset*(*base*)	LW	*rt*, *offset*(*base*)
	Load Word Left	LWL	*rt*, *offset*(*base*)	LWL	*rt*, *offset*(*base*)	LWL	*rt*, *offset*(*base*)
	Load Word Right	LWR	*rt*, *offset*(*base*)	LWR	*rt*, *offset*(*base*)	LWR	*rt*, *offset*(*base*)
	Store Byte	SB	*rt*, *offset*(*base*)	SB	*rt*, *offset*(*base*)	SB	*rt*, *offset*(*base*)
	Store Halfword	SH	*rt*, *offset*(*base*)	SH	*rt*, *offset*(*base*)	SH	*rt*, *offset*(*base*)
	Store Word	SW	*rt*, *offset*(*base*)	SW	*rt*, *offset*(*base*)	SW	*rt*, *offset*(*base*)
	Store Word Left	SWL	*rt*, *offset*(*base*)	SWL	*rt*, *offset*(*base*)	SWL	*rt*, *offset*(*base*)
	Store Word Right	SWR	*rt*, *offset*(*base*)	SWR	*rt*, *offset*(*base*)	SWR	*rt*, *offset*(*base*)
	Sync	SYNC		SYNC			
ALU 即値命令	Add Immediate	ADDI	*rt*, *rs*, *immediate*	ADDI	*rt*, *rs*, *immediate*	ADDI	*rt*, *rs*, *immediate*
	Add Immediate Unsigned	ADDIU	*rt*, *rs*, *immediate*	ADDIU	*rt*, *rs*, *immediate*	ADDIU	*rt*, *rs*, *immediate*
	Set On Less Than Immediate	SLTI	*rt*, *rs*, *immediate*	SLTI	*rt*, *rs*, *immediate*	SLTI	*rt*, *rs*, *immediate*
	Set On Less Than Immediate Unsigned	SLTIU	*rt*, *rs*, *immediate*	SLTIU	*rt*, *rs*, *immediate*	SLTIU	*rt*, *rs*, *immediate*
	AND Immediate	ANDI	*rt*, *rs*, *immediate*	ANDI	*rt*, *rs*, *immediate*	ANDI	*rt*, *rs*, *immediate*
	OR Immediate	ORI	*rt*, *rs*, *immediate*	ORI	*rt*, *rs*, *immediate*	ORI	*rt*, *rs*, *immediate*
	Exclusive-OR Immediate	XORI	*rt*, *rs*, *immediate*	XORI	*rt*, *rs*, *immediate*	XORI	*rt*, *rs*, *immediate*
	Load Upper Immediate	LUI	*rt*, *immediate*	LUI	*rt*, *immediate*	LUI	*rt*, *immediate*
3 オペランドレジスタタイプ命令	Add	*ADD*	*rd*, *rs*, *rt*	ADD	*rd*, *rs*, *rt*	ADD	*rd*, *rs*, *rt*
	Add Unsigned	ADDU	*rd*, *rs*, *rt*	ADDU	*rd*, *rs*, *rt*	ADDU	*rd*, *rs*, *rt*
	Subtract	SUB	*rd*, *rs*, *rt*	SUB	*rd*, *rs*, *rt*	SUB	*rd*, *rs*, *rt*
	Subtract Unsigned	SUBU	*rd*, *rs*, *rt*	SUBU	*rd*, *rs*, *rt*	SUBU	*rd*, *rs*, *rt*
	Set On Less Than	SLT	*rd*, *rs*, *rt*	SLT	*rd*, *rs*, *rt*	SLT	*rd*, *rs*, *rt*
	Set On Less Than Unsigned	SLTU	*rd*, *rs*, *rt*	SLTU	*rd*, *rs*, *rt*	SLTU	*rd*, *rs*, *rt*
	AND	AND	*rd*, *rs*, *rt*	AND	*rd*, *rs*, *rt*	AND	*rd*, *rs*, *rt*
	OR	OR	*rd*, *rs*, *rt*	OR	*rd*, *rs*, *rt*	OR	*rd*, *rs*, *rt*
	Exclusive-OR	XOR	*rd*, *rs*, *rt*	XOR	*rd*, *rs*, *rt*	XOR	*rd*, *rs*, *rt*
	NOR	NOR	*rd*, *rs*, *rt*	NOR	*rd*, *rs*, *rt*	NOR	*rd*, *rs*, *rt*

分　類	命　令	TX19 32 ビット ISA		TX39		R3000A	
シフト命令	Shift Left Logical	SLL	*rd*, *rt*, *sa*	SLL	*rd*, *rt*, *sa*	SLL	*rd*, *rt*, *sa*
	Shift Left Logical Variable	SLLV	*rd*, *rt*, *rs*	SLLV	*rd*, *rt*, *rs*	SLLV	*rd*, *rt*, *rs*
	Shift Right Logical	SRL	*rd*, *rt*, *sa*	SRL	*rd*, *rt*, *sa*	SRL	*rd*, *rt*, *sa*
	Shift Right Logical Variable	SRLV	*rd*, *rt*, *rs*	SRLV	*rd*, *rt*, *rs*	SRLV	*rd*, *rt*, *rs*
	Shift Right Arithmetic	SRA	*rd*, *rt*, *sa*	SRA	*rd*, *rt*, *sa*	SRA	*rd*, *rt*, *sa*
	Shift Right Arithmetic Variable	SRAV	*rd*, *rt*, *rs*	SRAV	*rd*, *rt*, *rs*	SRAV	*rd*, *rt*, *rs*
乗算・除算命令	Multiply	MULT	*rs*, *rt*	MULT	*rs*, *rt*	MULT	*rs*, *rt*
		MULT	*rd*, *rs*, *rt*	MULT	*rd*, *rs*, *rt*		
	Multiply Unsigned	MULTU	*rs*, *rt*	MULTU	*rs*, *rt*	MULTU	*rs*, *rt*
		MULTU	*rd*, *rs*, *rt*	MULTU	*rd*, *rs*, *rt*		
	Divide	DIV	*rs*, *rt*	DIV	*rs*, *rt*	DIV	*rs*, *rt*
	Divide Unsigned	DIVU	*rs*, *rt*	DIVU	*rs*, *rt*	DIVU	*rs*, *rt*
	Move From HI	MFHI	*rd*	MFHI	*rd*	MFHI	*rd*
	Move From LO	MFLO	*rd*	MFLO	*rd*	MFLO	*rd*
	Move To HI	MTHI	*rd*	MTHI	*rd*	MTHI	*rd*
	Move To LO	MTLO	*rd*	MTLO	*rd*	MTLO	*rd*
積和命令	Multiply-and-Add	MADD	*rs*, *rt*	MADD	*rs*, *rt*		
		MADD	*rd*, *rs*, *rt*	MADD	*rd*, *rs*, *rt*		
	Multiply-and-Add Unsigned	MADDU	*rs*, *rt*	MADDU	*rs*, *rt*		
		MADDU	*rd*, *rs*, *rt*	MADDU	*rd*, *rs*, *rt*		
ジャンプ命令	Jump	J	*target*	J	*target*	J	*target*
	Jump And Link	JAL	*target*	JAL	*target*	JAL	*target*
	Jump And Link eXchange	JALX	*target*				
	Jump Register	JR	*rs*	JR	*rs*	JR	*rs*
	Jump And Link Register	JALR	(*rd*,) *rs*	JALR	(*rd*,) *rs*	JALR	(*rd*,) *rs*
分岐命令	Branch On Equal	BEQ	*rs*, *rt*, *offset*	BEQ	*rs*, *rt*, *offset*	BEQ	*rs*, *rt*, *offset*
	Branch On Not Equal	BNE	*rs*, *rt*, *offset*	BNE	*rs*, *rt*, *offset*	BNE	*rs*, *rt*, *offset*
	Branch On Greater Than Zero	BGTZ	*rs*, *offset*	BGTZ	*rs*, *offset*	BGTZ	*rs*, *offset*
	Branch On Greater Than or Equal to Zero	BGEZ	*rs*, *offset*	BGEZ	*rs*, *offset*	BGEZ	*rs*, *offset*
	Branch On Less Than Zero	BLTZ	*rs*, *offset*	BLTZ	*rs*, *offset*	BLTZ	*rs*, *offset*
	Branch On Less Than or Equal to Zero	BLEZ	*rs*, *offset*	BLEZ	*rs*, *offset*	BLEZ	*rs*, *offset*
	Branch On Less Than Zero And Link	BLTZAL	*rs*, *offset*	BLTZAL	*rs*, *offset*	BLTZAL	*rs*, *offset*
	Branch On Greater Than Zero And Link	BGEZAL	*rs*, *offset*	BGEZAL	*rs*, *offset*	BGEZAL	*rs*, *offset*
分岐ライクラリ命令	Branch On Equal Likely	BEQL	*rs*, *rt*, *offset*	BEQL	*rs*, *rt*, *offset*		
	Branch On Not Equal Likely	BNEL	*rs*, *rt*, *offset*	BNEL	*rs*, *rt*, *offset*		
	Branch On Greater Than Zero Likely	BGTZL	*rs*, *offset*	BGTZL	*rs*, *offset*		
	Branch On Greater Than or Equal to Zero Likely	BGEZL	*rs*, *offset*	BGEZL	*rs*, *offset*		
	Branch On Less Than Zero Likely	BLTZL	*rs*, *offset*	BLTZL	*rs*, *offset*		
	Branch On Less Than or Equal to Zero Likely	BLEZL	*rs*, *offset*	BLEZL	*rs*, *offset*		

分　類	命　令	TX19 32 ビット ISA	TX39	R3000A
分岐ライクラリ命令	Branch On Less Than Zero And Link Likely	BLTZALL *rs*, *offset*	BLTZALL *rs*, *offset*	
	Branch On Greater Than Zero And Link Likely	BGEZALL *rs*, *offset*	BGEZALL *rs*, *offset*	
コプロセッサ命令	Move To Coprocessor	MTC*z* *rt*, *rd*	MTC*z* *rt*, *rd*	MTC*z* *rt*, *rd*
	Move From Coprocessor	MFC*z* *rt*, *rd*	MFC*z* *rt*, *rd*	MFC*z* *rt*, *rd*
	Move Control To Coprocessor	CTC*z* *rt*, *rd*	CTC*z* *rt*, *rd*	CTC*z* *rt*, *rd*
	Move Control From Coprocessor	CFC*z* *rt*, *rd*	CFC*z* *rt*, *rd*	CFC*z* *rt*, *rd*
	Coprocessor Operation	COP*z* *cofun*	COP*z* *cofun*	COP*z* *cofun*
	Branch On Coprocessor *z* True	BC*z*T *offset*	BC*z*T *offset*	BC*z*T *offset*
	Branch On Coprocessor *z* True Likely	BC*z*TL *offset*	BC*z*TL *offset*	BC*z*TL *offset*
	Branch On Coprocessor *z* False	BC*z*F *offset*	BC*z*F *offset*	BC*z*F *offset*
	Branch On Coprocessor *z* False Likely	BC*z*FL *offset*	BC*z*FL *offset*	BC*z*FL *offset*
	Load Word To Coprocessor			LWC*z* *rt*, *offset*(*base*)
	Store Word From Coprocessor			SWC*z* *rt*, *offset*(*base*)
システム制御コプロセッサ命令	Move To CP0	MTC0 *rt*, *rd*	MTC0 *rt*, *rd*	
	Move From CP0	MFC0 *rt*, *rd*	MFC0 *rt*, *rd*	
	Restore From Exception	RFE	RFE	
	Debug Exception Return	DERET	DERET	
	Cache	CACHE *op*, *offset*(*base*)	CACHE *op*, *offset*(*base*)	
	Read Indexed TLB Entry†	(TLBR)	(TLBR)	TLBR
	Write Indexed TLB Entry†	(TLBWI)	(TLBWI)	TLBWI
	Write Random TLB Entry†	(TLBWR)	(TLBWR)	TLBWR
	Probe TLB For Matching Entry†	(TLBP)	(TLBP)	TLBP
特殊命令	System Call	SYSCALL *code*	SYSCALL *code*	SYSCALL *code*
	Breakpoint	BREAK *code*	BREAK *code*	BREAK *code*
	Software Debug Breakpoint Exception	SDBBP *code*	SDBBP *code*	

†　TX19 と TX39 では NOP 動作になります。

表 D-3に、TX19 の 16 ビット ISA モードと MIPS16 ASE の命令セットの比較を示します。ダブルワード命令と LWU (Load Word Unsigned) 命令は TX19 では使用できませんが、それ以外の命令は、TX19 と MIPS16 ASE で互換性があります。

表 D-3　TX19 16 ビット ISA と MIPS16 ASE の命令セット

分　類	命　令	TX19　16 ビット ISA	MIPS16 ASE
ロード・ストア命令	Load Byte	LB *ry*, *offset*(*base*)	LB *ry*, *offset*(*base*)
	Load Byte Unsigned	LBU *ry*, *offset*(*base*)	LBU *ry*, *offset*(*base*)
	Load Halfword	LH *ry*, *offset*(*base*)	LH *ry*, *offset*(*base*)
	Load Halfword Unsigned	LHU *ry*, *offset*(*base*)	LHU *ry*, *offset*(*base*)
	Load Word	LW *ry*, *offset*(*base*)	LW *ry*, *offset*(*base*)
		LW *ry*, *offset*(pc)	LW *ry*, *offset*(pc)
		LW *ry*, *offset*(sp)	LW *ry*, *offset*(sp)
	Load Word Unsigned		LWU *ry*, *offset*(sp)
	Load Doubleword		LD *ry*, *offset*(*base*)
			LD *ry*, *offset*(pc)
			LD *ry*, *offset*(sp)
	Store Byte	SB *ry*, *offset*(*base*)	SB *ry*, *offset*(*base*)
	Store Halfword	SH *ry*, *offset*(*base*)	SH *ry*, *offset*(*base*)
	Store Word	SW *ry*, *offset*(*base*)	SW *ry*, *offset*(*base*)
		SW *ry*, *offset*(pc)	SW *ry*, *offset*(pc)
		SW *ry*, *offset*(sp)	SW *ry*, *offset*(sp)
	Store Doubleword		SD *ry*, *offset*(*base*)
			SD *ry*, *offset*(pc)
			SD *ry*, *offset*(sp)
ALU 即値命令	Add Immediate	ADDIU *ry*, *rx*, *immediate*	ADDIU *ry*, *rx*, *immediate*
		ADDIU *rx*, *immediate*	ADDIU *rx*, *immediate*
		ADDIU sp, *immediate*	ADDIU *sp, immediate*
		ADDIU *rx*, pc, *immediate*	ADDIU *rx*, pc, *immediate*
		ADDIU *rx*, sp, *immediate*	ADDIU *rx*, sp, *immediate*
	Doubleword Add Immediate		DADDIU *ry*, *rx*, *immediate*
			DADDIU *ry*, *immediate*
			DADDIU *ry*, sp, *immediate*
			DADDIU sp, *immediate*
			DADDIU *ry*, pc, *immediate*
	Set On Less Than Immediate	SLTI *rx*, *immediate*	SLTI *rx*, *immediate*
	Set On Less Than Immediate Unsigned	SLTIU *rx*, *immediate*	SLTIU *rx*, *immediate*
	Compare Immediate	CMPI *rx*, *immediate*	CMPI *rx*, *immediate*
	Load Immediate	LI *rx*, *immediate*	LI *rx*, *immediate*
2・3 オペランドレジスタタイプ命令	Add Unsigned	ADDU *rz*, *rx*, *ry*	ADDU *rz*, *rx*, *ry*
	Doubleword Add Unsigned		DADDU *rz*, *rx*, *ry*
	Subtract Unsigned	SUBU *rz*, *rx*, *ry*	SUBU *rz*, *rx*, *ry*
	Doubleword Subtract Unsigned		DSUBU *rz*, *rx*, *ry*
	Set On Less Than	SLT *rx, ry*	SLT *rx, ry*
	Set On Less Than Unsigned	SLTU *rx, ry*	SLTU *rx, ry*
	Compare	CMP *rx, ry*	CMP *rx, ry*
	Negate	NEG *rx, ry*	NEG *rx, ry*
	AND	AND *rx, ry*	AND *rx, ry*

分　類	命　令	TX19 16 ビット ISA		MIPS16 ASE	
2・3 オペランドレジスタタイプ命令	OR	OR	*rx, ry*	OR	*rx, ry*
	Exclusive-R	XOR	*rx, ry*	XOR	*rx, ry*
	Not	NOT	*rx, ry*	NOT	*rx, ry*
	Move	MOVE	*ry*, *r32*	MOVE	*ry*, *r32*
		MOVE	*r32*, *rz*	MOVE	*r32*, *rz*
シフト命令	Shift Left Logical	*SLL*	*rx*, *ry*, *sa*	SLL	*rx*, *ry*, *sa*
	Shift Left Logical Variable	SLLV	ry, *rx*	SLLV	*ry, rx*
	Shift Right Logical	SRL	*rx*, *ry*, *sa*	SRL	*rx*, *ry*, *sa*
	Shift Right Logical Variable	SRLV	*ry*, *rx*	SRLV	*ry*, *rx*
	Shift Right Arithmetic	SRA	*rx*, *ry*, *sa*	SRA	*rx*, *ry*, *sa*
	Shift Right Arithmetic Variable	SRAV	*ry*, *rx*	SRAV	*ry*, *rx*
	Doubleword Shift Left Logical			DSLL	*rx*, *ry*, *sa*
	Doubleword Shift Left Logical Variable			DSLLV	*ry*, *rx*
	Doubleword Shift Right Logical			DSRL	*ry*, *sa*
	Doubleword Shift Right Logical Variable			DSRLV	*ry*, *rx*
	Doubleword Shift Right Arithmetic			DSRA	*ry*, *sa*
	Doubleword Shift Right Arithmetic Variable			DSRAV	*ry*, *rx*
積和命令	Multiply	MULT	*rx, ry*	MULT	*rx, ry*
	Multiply Unsigned	MULTU	*rx, ry*	MULTU	*rx, ry*
	Doubleword Multiply			DMULT	*rx, ry*
	Doubleword Multiply Unsigned			DMULTU	*rx, ry*
	Divide	DIV	*rx, ry*	DIV	*rx, ry*
	Divide Unsigned	DIVU	*rx, ry*	DIVU	*rx, ry*
	Doubleword Divide	DIV	*rx, ry*	DDIV	*rx, ry*
	Doubleword Divide Unsigned	DIVU	*rx, ry*	DDIVU	*rx, ry*
	Move From HI	MFHI	*rx*	MFHI	*rx*
	Move From LO	MFLO	*rx*	MFLO	*rx*
ジャンプ命令	Jump And Link	JAL	*target*	JAL	*target*
	Jump And Link eXchange	JALX	*target*	JALX	*target*
	Jump Register	JR	*rx*	JR	*rx*
		JR	ra	JR	ra
	Jump And Link Register	JALR	ra, *rx*	JALR	*ra, rx*
分岐命令	Branch On Equal To Zero	BEQZ	*rx*, *offset*	BEQZ	*rx*, *offset*
	Branch On Not Equal To Zero	BNEZ	*rx*, *offset*	BNEZ	*rx*, *offset*
	Branch On T8 Equal To Zero	BTEQZ	*offset*	BTEQZ	*offset*
	Branch On T8 Not Equal to Zero	BTNEZ	*offset*	BTNEZ	*offset*
	Branch Unconditional	B	*offset*	B	*offset*
特殊命令	Breakpoint	BREAK	*code*	BREAK	*code*
	Software Debug Breakpoint Exception	SDBBP	*code*	SDBBP	*code*
	Extend	EXTEND	*immediate*	EXTEND	*immediate*