AIX バージョン 7.3

Assembler Language Reference

お願い

本書および本書で紹介する製品をご使用になる前に、639 ページの『特記事項』の情報をお読みください。

以後のエディションで別段の指示が出されるまで、このエディションは AIX バージョン 7.3 およびそれ以降のすべてのリリースおよびモディフィケーションに適用されます。

目次

本書について

本書には、オペレーティング・システム内で作動するアセンブラー・プログラムに関する詳細情報が記載されています。 このトピック・コレクションには、疑似命令および命令セットに関する詳細も含まれています。 本書は、オペレーティング・システムに付属のドキュメンテーション CD にも収録されています。

強調表示

本書では、次の強調表示規則を使用しています。

Bold	その名前がシステムによって事前に定義されているコマンド、サブルーチン、キーワード、ファイル、構造、ディレクトリー、およびその他の項目であることを示します。さらに太字の強調表示は、ユーザーが選択するボタン、ラベル、およびアイコンなどのグラフィカル・オブジェクトも示します。
イタリック	ユーザーが入力する実際の名前または値のパラメーターを示します。
`Monospace`	具体的なデータ値の例、表示される可能性があるテキストの例、 プログラマーとして作成する可能性があるものに似たプログラム・コードの一部の例、 システムからのメッセージ、またはユーザーが入力しなければならないテキストを示します。

AIX

AIX® オペレーティング・システムでは、すべてケース・センシティブとなっています。これは、英大文字と小文字を区別するということです。 例えば、**ls** コマンドを使用するとファイルをリストできます。 LS と入力すると、システムはそのコマンドが「`is not found`」と応答します。 同様に、**FILEA**、**FiLea**、および **filea** は、同じディレクトリーにある場合でも、3 つの異なるファイル名です。 予期しない処理が実行されないように、常に正しい大/小文字を使用するようにしてください。

ISO 9000

この製品の開発と製造には、ISO 9000 登録済み高品質システムが使用されました。

Assembler Language Reference

「**アセンブラー言語解説書**」トピックには、オペレーティング・システム内で作動するアセンブラー・プログラムに関する情報が記載されています。

アセンブラーはマシン言語の命令を受け取り、それをマシン・オブジェクト・コードに変換します。

アセンブラーの概要

アセンブラー・プログラムは、マシン言語命令を受け取り、それらをマシン・オブジェクト・コードに変換します。

アセンブラーとは、オペレーティング・システム内で動作するプログラムです。 アセンブラーはマシン言語の命令を受け取り、それをマシン・オブジェクト・コードに変換します。 以下の項目では、アセンブラーの機能について説明します。

AIX® アセンブラーの機能

AIX アセンブラーの機能。

このセクションでは、 AIX® アセンブラーの機能について説明します。

複数のハードウェア・アーキテクチャーと実装プラットフォームのサポート

アセンブラーは、 Power® および PowerPC® プロセッサー・アーキテクチャーのいずれかに固有の命令を含むソース・プログラムをサポートします。

以下の Power および PowerPC プロセッサー・アーキテクチャーがサポートされています。

- 第 1 世代の POWER® ファミリー・プロセッサー (POWER ファミリー・アーキテクチャー)
- PowerPC 601 RISC マイクロプロセッサー、PowerPC 604 RISC マイクロプロセッサー、または PowerPC A35 RISC マイクロプロセッサー (PowerPC アーキテクチャー)
- POWER5、PowerPC 970、POWER5+、POWER6®、POWER7®、POWER8®、POWER9™、および Power10 プロセッサー・ベース・サーバー。

命令にはいくつかのカテゴリーがあり、サポートされる実装ごとに 1 つ以上のカテゴリーの命令が有効です。 さまざまなアーキテクチャーの説明で、各実装でサポートされる指示について説明しています。 **-M** フラグを使用すると、特定のアセンブリー・モードで有効な命令を判別したり、特定の命令を使用できるアセンブリー・モードを判別したりすることができます。

POWER9 プロセッサー・アーキテクチャーは、 Power Instruction Set Architecture バージョン 3.0 仕様に記述されています。 詳しくは、 OpenPOWER Web サイトを参照してください。

>|Power10 プロセッサー・アーキテクチャーは、Power Instruction Set Architecture バージョン 3.1 仕様に記述されています。 詳しくは、 OpenPOWER Web サイトを参照してください。|<

ホスト・マシン独立性およびターゲット環境標識フラグ

ホスト・マシンは、アセンブラーが実行されるハードウェア・プラットフォームです。

ホスト・マシンは、アセンブラーが実行されるハードウェア・プラットフォームです。 ターゲット・マシンは、オブジェクト・コードが実行されるプラットフォームです。 アセンブラーが実行されるホスト・マシンに関係なく、アセンブラーは任意のターゲット・マシン用にソース・プログラムをアセンブルできます。

ターゲット・マシンは、 **as** コマンドのアセンブリー・モード・オプション・フラグ **-m** 、または **.machine** 疑似操作のいずれかを使用して指定できます。 **-m** フラグも **.machine** 疑似命令も使用しない場合は、デフォルトのアセンブリー・モードが使用されます。 **-m** フラグと **.machine** 疑似命令の両方を使用すると、**.machine** 疑似命令は **-m** フラグをオーバーライドします。 ソース・プログラムでは、複数の **.machine**

疑似命令を使用できます。 後の **.machine** 疑似命令の値は、前の **.machine** 疑似命令をオーバーライドします。

AIX® アセンブラーによって提供されるデフォルトのアセンブリー・モードでは、ターゲット環境として POWER® ファミリーと PowerPC® の交点がありますが、POWER/PowerPC® の非互換性エラー (POWER/PowerPC® 交点外の命令と無効なフォーム・エラーを含む) はすべて説明の警告として処理されます。 **-W** および **-w** アセンブラー・フラグは、これらの警告を表示するかどうかを制御します。 **as** コマンドまたは **.machine** 疑似命令の **-m** フラグが指定されていないことに加えて、 **as** コマンドの **-m** フラグまたは **.machine** 疑似命令を使用して、デフォルトのアセンブリー・モードを明示的に指定することもできます。

エラーや警告のない複数のプラットフォームから、プラットフォーム固有の命令を含むソース・プログラムをアセンブルするには、以下のいずれかの方法を使用します。

- ソース・プログラムで **.machine** 疑似命令を使用します。
- アセンブリー・モードを **任意** モード (**as** コマンドの **-m** フラグを使用) に設定してプログラムをアセンブルします。

例えば、ソース・コードに POWER® ファミリー固有の命令と PowerPC® 601 RISC マイクロプロセッサー固有の命令の両方を含めることはできません。 これは、単一のソース・プログラムに含まれる各サブソース・プログラムにも当てはまります。 サブソース・プログラムは、 **.machine** 疑似命令で始まり、次の **.machine** 疑似命令の前に終了します。 1 つのソース・プログラムに複数の **.machine** 疑似命令を含めることができるため、通常は複数のサブソース・プログラムで構成されます。

ニーモニック相互参照

アセンブラーは、PowerPC® と POWER® ファミリーの両方のニーモニックをサポートします。

アセンブラーは、PowerPC® と POWER® ファミリーの両方のニーモニックをサポートします。 アセンブラー・リストには、両方のニーモニックの相互参照があります。 相互参照は、POWER® ファミリーと PowerPC® アーキテクチャーでは異なるニーモニックを持つが、同じ命令コード、関数、およびオペランド入力フォーマットを共有する命令に制限されます。

アセンブラー・リストには、ニーモニック相互参照情報を表示する列が含まれています。

ニーモニック相互参照は、ユーザーがソース・プログラムをあるアーキテクチャーから別のアーキテクチャーにマイグレーションするのに役立ちます。 **as** コマンドの **-s** フラグは、マイグレーションを支援するためにアセンブラー・リストにニーモニック相互参照を提供します。 **-s** フラグを使用しないと、ニーモニック相互参照は提供されません。

CPU ID 定義

アセンブリー・プロセス中に、アセンブラーは、すべての命令を含む最小の命令セットを判別し、命令セットを示す CPU ID 値を与えます。

アセンブリー・プロセス中に、アセンブラーは、(アーキテクチャーまたはプロセッサー・インプリメンテーションで定義されたいくつかの完全な命令セットのリストから) どの命令セットが、プログラムで使用されるすべての命令を含む最も小さい命令セットであるかを判別します。 プログラムには、この命令セットを示す CPU ID 値が与えられます。 したがって、CPU ID は、オブジェクト・コードを実行できるターゲット環境を示します。 プログラムの CPU ID 値は、アセンブラーによって生成される XCOFF オブジェクト・ファイルに含まれるアセンブラー出力値です。

CPU ID には、以下の値を指定できます。

値	説明
com	プログラムで使用されるすべての命令は、PowerPC® と POWER® のファミリー・アーキテクチャーの交差点にあります。 (**com** 命令セットは、最小の命令セットです。)
ppc	プログラムで使用されるすべての命令は、PowerPC® アーキテクチャーの 32 ビット・モードですが、プログラムは CPU ID 値 **com** の条件を満たしていません。 (**ppc** 命令セットは、**com** 命令セットのスーパーセットです。)

値	説明
pwr	プログラムで使用されるすべての命令は POWER® ファミリー・アーキテクチャー、POWER® ファミリー・インプリメンテーションに含まれていますが、プログラムは CPU ID 値 **com** の条件を満たしていません。(**pwr** 命令セットは、 **com** 命令セットのスーパーセットです。)
pwr2	プログラムで使用されるすべての命令は POWER® ファミリー・アーキテクチャーの POWER2™ 実装に含まれていますが、プログラムは CPU ID 値 **com**、 **ppc**、または **pwr** の条件を満たしていません。(**pwr2** 命令セットは、 **pwr** 命令セットのスーパーセットです。)
any	プログラムには、有効なアーキテクチャーまたはインプリメンテーションからの命令が混在しているか、またはインプリメンテーション固有の命令が含まれています。プログラムは、CPU ID 値 **com**、 **ppc**、 **pwr**、または **pwr2** の条件を満たしていません。(**any** 命令セットは最大の命令セットです。)

アセンブラー出力値の CPU ID は、アセンブリー・モードと同じではありません。 アセンブリー・モード (**as** コマンドの **-m** フラグ、およびプログラム内の **.machine** 疑似命令の使用によって決定される) によって、アセンブラーがエラーや警告なしで受け入れる命令が決まります。 CPU ID は、実際に使用される命令を示す出力値です。

出力 XCOFF ファイルでは、CPU ID は、C_FILE ストレージ・クラスを持つシンボル・テーブル・エントリーの n_type フィールドの下位バイトに保管されます。 以下のリストは、下位バイト値とそれに対応する CPU ID を示しています。

低位バイト	CPU ID
0	定義された値ではありません。 CPU-ID フィールドの定義の前に、無効な値またはオブジェクトがアセンブルされました。
1	**ppc**
2	**ppc64**
3	**com**
4	**pwr**
5	**any**
6	**601**
7	**603**
8	**604**
10	**電源**
16	**620**
17	**A35**
18	**pwr5**
19	**ppc970 または 970**
20	**pwr6**
21	**vec**
22	**pwr5x**
23	**pwr6e**
24	**pwr7**
25	**pwr8**
26	**pwr9**
>|>|**27**	**pwr10**|<|<

低位バイト	CPU ID
224	**pwr2 または pwrx**

ソース言語タイプ

アセンブラーは、ソース言語タイプを記録します。

カスケード・コンパイラーの場合、アセンブラーはソース言語タイプを記録します。 XCOFF ファイルでは、C_FILE ストレージ・クラスを持つシンボル・テーブル・エントリーの n_type フィールドの高位バイトに、ソース言語タイプ情報が保持されます。 以下の言語タイプが定義されています。

高位バイト (High-Order Byte)	言語
0x00	C
0x01	FORTRAN
0x02	Pascal
0x03	Ada
0x04	PL/I
0x05	Basic
0x06	Lisp
0x07	COBOL
0x08	Modula2
0x09	C++
0x0A	RPG
0x0B	PL8、PLIX
0x0C	アセンブラー
0x0D-BxFF	予約済み

ソース言語タイプは、 **.source** 疑似命令によって示されます。 デフォルトでは、ソース言語タイプは「アセンブラー」です。 詳しくは、 **.source** 疑似命令を参照してください。

検出エラー条件

ソース・プログラムに無効な命令フォームが含まれている場合は、エラーが報告されます。 POWER® ファミリーと PowerPC® アーキテクチャーの間の非互換性が原因でエラーが発生します。

意図したターゲット環境でサポートされていない命令がソース・プログラムに含まれている場合は、エラー番号 149 が報告されます。

ソース・プログラムに無効な命令フォームが含まれている場合は、エラーが報告されます。 このエラーは、POWER® ファミリーと PowerPC® アーキテクチャーの間の非互換性が原因で発生します。 PowerPC® アーキテクチャーに適用されるいくつかの制約事項は、POWER® ファミリー・アーキテクチャーには適用されません。 PowerPC® アーキテクチャーに従って、以下の無効な命令形式が定義されています。

- Rc ビット、LK ビット、または OE ビットが/(スラッシュ) として定義されているが、1 としてコーディングされているか、または 1 として定義されているが 0 としてコーディングされている場合、この形式は無効です。 通常、アセンブラーは、これらのビットに正しい値が含まれていることを確認します。

  一部のフィールドは、複数の / (スラッシュ) で定義されています (例えば、"///")。 これらがゼロ以外としてコーディングされている場合、形式は無効です。 これらのフィールドに特定の入力オペランドを使用する場合は、それらを検査する必要があります。 このため、以下の指示が検査されます。

– PowerPC ® System Call 命令または POWER ® ファミリーの Supervisor Call 命令では、アセンブリー・モードが PowerPC ® タイプのときに POWER ® ファミリーの **svca** ニーモニックを使用する場合、SV フィールドは 0 でなければなりません。 それ以外の場合、命令フォームは無効であり、エラー番号 165 が報告されます。

  注 : svc および **svcl** 命令は、PowerPC ® ターゲット・モードではサポートされていません。 **svcla** 命令は、PowerPC ® 601 RISC マイクロプロセッサーでのみサポートされます。

– Move to Segment Register Indirect 命令では、POWER ® ファミリーの **mtsri** ニーモニックが PowerPC ® ターゲット・モードで使用される場合、RA フィールドは 0 でなければなりません。 それ以外の場合、命令フォームは無効であり、エラー番号 154 が報告されます。 PowerPC ® **ムツリン** 簡略記号が PowerPC ® ターゲット・モードで使用される場合、必要な入力オペランドは 2 つだけなので、検査は必要ありません。

- すべてのブランチ条件付き命令 (ブランチ条件付き、リンク・レジスターへのブランチ条件付き、およびカウント・レジスターへのブランチ条件付きを含む) について、BO フィールドのビット 0 から 3 が検査されます。 0 を含む必要があるビットにゼロ以外の値が含まれている場合、エラー 150 が報告されます。

  BO フィールドのエンコードは、PowerPC ® アーキテクチャーの「ブランチ・プロセッサー命令」セクションで定義されています。 以下のリストで、このフィールドに指定できる値について簡単に説明します。

BO	**説明**
0000y	カウント・レジスター (CTR) を減らします。次に、減分された CTR の値が 0 でなく、条件が False の場合に分岐します。
0001y	CTR を減らします。次に、減分された CTR の値が 0 ではなく、条件が False の場合に分岐します。
001zy	条件が False の場合に分岐します。
0100y	CTR を減らします。次に、減分された CTR の値が 0 ではなく、条件が True の場合に分岐します。
0101y	CTR を減らします。次に、減分された CTR の値が 0 ではなく、条件が True の場合に分岐します。
011zy	条件が True の場合に分岐します。
1z00y	CTR を減らします。次に、減分された CTR の値が 0 でない場合に分岐します。
1z01y	CTR を減らします。次に、減分された CTR の値が 0 でない場合に分岐します。
1z1zz	常に分岐します。 z ビットは、0 でなければならないビットを表します。 ビットが 0 でない場合、命令形式は無効です。

  注 : y ビットは、条件付き分岐が行われる可能性が高いかどうかに関するヒントを提供します。 このビットの値は 0 または 1 のいずれかです。 デフォルト値は 0 です。 PowerPC ® アーキテクチャーで定義されているブランチ予測の拡張ニーモニックを使用して、このビットを 0 または 1 に設定します。 (詳しくは、 分岐予測の拡張ニーモニック を参照してください。)

  ブランチ命令では、BO フィールドに y ビットはありません。 BO フィールドのビット 4 には 0 が含まれている必要があります。 それ以外の場合、命令フォームは無効です。

  BO フィールドの 3 番目のビットは、「減分およびテスト CTR」オプションとして指定されます。 Branch Conditional to Count Register 命令の場合、BO フィールドの 3 番目のビットは 0 であってはなりません。 それ以外の場合、命令形式は無効であり、エラー 163 が報告されます。

- 固定小数点ロード命令の更新形式の場合、PowerPC ® アーキテクチャーでは、RA フィールドが 0 または RT フィールド値のいずれにも等しくないことが必要です。 それ以外の場合、命令形式は無効であり、エラー番号 151 が報告されます。

  この制限は、以下の命令に適用されます。

  – **LB ツー**

- **LBZUX**
- **lhzu (lhzu)**
- **lhsux (lhsux)**
- **lhau (lhau)**
- **lhaux**
- **lwzu** (POWER® ファミリーの **lu**)
- **lwzux** (POWER® ファミリーの **lux**)

- 固定小数点保管命令および浮動小数点ロードおよび保管命令の更新形式の場合、以下の命令は、RA フィールドが 0 に等しくないことのみを必要とします。 それ以外の場合、命令フォームは無効であり、エラー番号 166 が報告されます。
  - **lfsu**
  - **lfsux (lfsux)**
  - **lfdu (lfdu)**
  - **lfdux**
  - **STBU**
  - **STBUX**
  - **sthu (sthu)**
  - **sthux (sthux)**
  - **stwu** (POWER® ファミリーの **stu**)
  - **stwux** (POWER® ファミリーの **stux**)
  - **stfsu (stfsu)**
  - **標準偏差 (stfux)**
  - **stfdu (stfdu)**
  - **STFDOX**
- 複数のレジスター・ロード命令の場合、PowerPC® アーキテクチャーでは、RA フィールドと RB フィールド (命令形式で存在する場合) が、ロードするレジスターの範囲内にないことが必要です。 また、RA=RT=0 は許可されません。 RA=RT=0 の場合、命令形式が無効であり、エラー 164 が報告されます。 この制限は、以下の命令に適用されます。
  - **lmn** (POWER® ファミリーの **lm**)
  - **l 断絶** (POWER® ファミリーの **lsi**)
  - **lswx** (POWER® ファミリーの **lsx**)

    注 : lswx 命令の場合、アセンブラーは RA=RT=0 のみを検査します。これは、実行時に XER レジスターの内容によってロード・レジスターの範囲が決定されるためです。
- 固定小数点比較命令の場合、PowerPC® アーキテクチャーでは、L フィールドが 0 でなければなりません。 それ以外の場合、命令フォームは無効であり、エラー番号 154 が報告されます。 この制限は、以下の命令に適用されます。
  - **cmp - 2 つのファイルを比較する**
  - **cmpi (cmpi)**
  - **cmpli (cmpli)**
  - **cmpl**

    注 : ターゲット・モードが **com** または **ppc** の場合、アセンブラーは、更新形式の固定小数点ロード命令、更新形式の固定小数点保管命令、更新形式の浮動小数点ロードおよび保管命令、複数レジスター・ロード命令、および固定小数点比較命令を検査し、エラーを報告します。 ターゲット・モードが **any**、**pwr**、 **pwr2**、または **601** の場合、検査は実行されません。

警告メッセージ

as コマンドで **-w** フラグを使用すると、警告メッセージがリストされます。

as コマンドで **-w** フラグを使用すると、警告メッセージがリストされます。 一部の警告メッセージは、POWER ® ファミリーと PowerPC ® の命令コードが同じ命令に関連しています。

- いくつかの命令には、POWER ® ファミリーと PowerPC ® アーキテクチャーの両方で同じ命令コードがありますが、機能上の定義は異なります。 ターゲット・モードが **com** で、 **as** コマンドの **-w** フラグが使用されている場合、アセンブラーはこれらの命令を識別し、警告番号 153 を報告します。 これらのニーモニックは機能的に異なるため、**as** コマンドで **-s** フラグを使用したときに生成されるアセンブラー・リストのニーモニック相互参照にはリストされません。 以下の表に、これらの手順をリストします。

表 *1.* 異なるニーモニックを持つ同じ命令コード

POWER ® ファミリー	**PowerPC (R)**
dcs (dcs)	**sync**
ics	**同期 (isync)**
svca (svca)	**sc**
mtsri (mtsri)	**mtsnn**
lsx (lsx)	**LSWX**

- 以下の命令には、同じニーモニックと命令コードがありますが、POWER ® ファミリーと PowerPC ® アーキテクチャーでは異なる機能定義があります。 これらの違いは、命令が実行されるマシンではなく、命令が実行される保護ドメインに基づいているため、アセンブラーはこれらを検査できません。
  - **mfsr (mfsr)**
  - **mfmsr (mfmsr)**
  - **mfdec (mfdec)**

特殊目的レジスターの変更および特殊目的レジスターのフィールド処理

特殊目的レジスターは、POWER ® ファミリー・アーキテクチャーで定義されます。

TID、 MQ、 SDR0、RTCU、および RTCL は、POWER ® ファミリー・アーキテクチャーで定義されている特殊目的レジスター (SPR) です。 これらは、PowerPC ® アーキテクチャーでは無効です。 ただし、 MQ、RTCU、および RTCL は、引き続き PowerPC ® 601 RISC マイクロプロセッサーで使用できます。

DBATL、DBATU、IBATL、IBATU、TBL、および TBU は、PowerPC ® アーキテクチャーで定義された SPR です。 これらは、PowerPC ® 601 RISC マイクロプロセッサーではサポートされません。 PowerPC ® 601 RISC Microprocessor は、代わりに BATL および BATU SPR を使用します。

アセンブラーは、「SPR との間の移動」命令用の拡張ニーモニックを提供します。 拡張ニーモニックには、POWER ® ファミリーおよび PowerPC ® アーキテクチャーで定義されているすべての SPR が含まれます。 無効な拡張簡略記号が使用されると、エラーが生成されます。 アセンブラーは、以下のいずれに対しても拡張簡略記号をサポートしません。

- POWER2™-固有の SPR (IMR、DABR、DSAR、TSR、および ILCR)
- PowerPC ® 601 RISC マイクロプロセッサー-固有の SPR (HID0、 HID1、 HID2、 HID5、PID、BATL、および BATU)
- PowerPC 603 RISC マイクロプロセッサー-固有の SPR (DMISS、DCMP、 HASH1、 HASH2、IMISS、ICMP、RPA、 HID0、および IABR)
- PowerPC 604 RISC マイクロプロセッサー-固有の SPR (PIE、 HID0、IABR、および DABR)

アセンブラーは、 **mtspr** 命令および **mfspr** 命令の SPR フィールドのエンコード値を検査しません。これは、SPR エンコード・コードが変更または再利用される可能性があるためです。 ただし、アセンブラーは SPR フィールドの値の範囲を検査します。 ターゲット・モードが **pwr**、 **pwr2**、または **com** の場合、SPR

フィールドの長さは 5 ビットで、最大値は 31 です。 それ以外の場合、SPR フィールドの長さは 10 ビットで、最大値は 1023 です。

POWER® ファミリーと PowerPC® アーキテクチャーのソース・コードの互換性を維持するために、アセンブラーは、SPR 番号の下位 5 ビットと上位 5 ビットが、**mfspr** または **mtspr** 命令への入力オペランドとして使用される前に逆にされると想定します。

関連情報

as

アセンブラーのインストール

AIX® アセンブラーは、コマンド、ファイル、およびライブラリーとともに基本オペレーティング・システムと共にインストールされます。

AIX® アセンブラーは、ソフトウェア・アプリケーションを開発するためのコマンド、ファイル、およびライブラリーとともに、基本オペレーティング・システムとともにインストールされます。

関連概念

.machine 疑似命令

.source 疑似命令

関連情報

as

処理とストレージ

プロセッサーは、メイン・メモリーおよびレジスターにデータを保管します。

マシン・アーキテクチャーの特性、および処理とストレージのインプリメンテーションは、プロセッサーのアセンブラー言語に影響を与えます。 アセンブラーは、POWER® ファミリーおよび PowerPC® アーキテクチャーを実装するさまざまなプロセッサーをサポートします。 2 つのアーキテクチャーは多数の命令を共有するため、アセンブラーは POWER® ファミリーと PowerPC® アーキテクチャーの両方をサポートできます。

このトピックでは、POWER® ファミリーと PowerPC® アーキテクチャーの概要と比較を示し、データがメイン・メモリーとレジスターにどのように保管されるかについて説明します。 また、POWER® ファミリーと PowerPC® の両方の命令セットの基本機能についても説明します。

このトピックで説明する手順はすべて非特権です。 したがって、このトピックで説明するすべてのレジスターは、非特権命令に関連しています。 特権命令とその関連レジスターは、PowerPC® アーキテクチャーで定義されます。

以下の処理およびストレージに関する記事では、システム・マイクロプロセッサーの概要を示し、メイン・メモリーとレジスターの両方にデータがどのように保管されるかを説明しています。 ここでは、システム・マイクロプロセッサーの命令セットと疑似操作の機能を理解するために必要な概念的な背景を説明します。

POWER® ファミリーと PowerPC® アーキテクチャーの概要

POWER® ファミリーまたは PowerPC® マイクロプロセッサーには、ブランチ・プロセッサー、固定小数点プロセッサー、および浮動小数点プロセッサーが含まれます。

POWER® ファミリーまたは PowerPC® マイクロプロセッサーには、命令フェッチ、命令実行、および割り込みアクションのためのシーケンス制御と処理制御が含まれており、POWER® ファミリーおよび PowerPC® アーキテクチャーで定義されている命令セット、ストレージ・モデル、およびその他の機能を実装しています。

POWER® ファミリーまたは PowerPC® マイクロプロセッサーには、ブランチ・プロセッサー、固定小数点プロセッサー、および浮動小数点プロセッサーが含まれます。 マイクロプロセッサーは、以下のクラスの命令を実行できます。

- ブランチ命令

- 固定小数点命令
- 浮動小数点命令

次の図は、PowerPC ® マイクロプロセッサーの命令処理の論理表現を示しています。

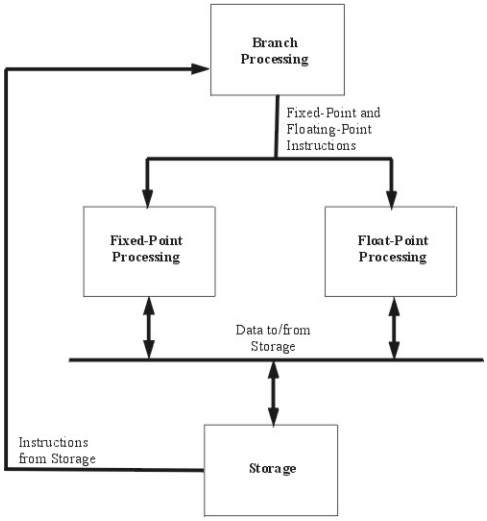

図 *1.* 論理処理モデル

次の表は、PowerPC ® ユーザー命令セット・アーキテクチャーのレジスターを示しています。 これらのレジスターは、32 ビット・アプリケーションに使用される CPU 内にあり、ユーザーが使用できます。

レジスター	**使用可能なビット**
条件レジスター (CR)	0-31
リンク・レジスター (LR)	0-31
カウント・レジスター (CTR)	0-31
汎用レジスター 00 から 31 まで (GPR)	レジスターごとに 0 から 31
固定小数点例外レジスター (XER)	0-31
浮動小数点レジスター 00 から 31 (FPR)	レジスターごとに 0 から 63
浮動小数点状況および制御レジスター (FPSCR)	0-31

次の表は、POWER ® ファミリーのユーザー命令セット・アーキテクチャーのレジスターを示しています。 これらのレジスターは、32 ビット・アプリケーションに使用される CPU 内にあり、ユーザーが使用できます。

レジスター	**使用可能なビット**
条件レジスター (CR)	0-31
リンク・レジスター (LR)	0-31
カウント・レジスター (CTR)	0-31
汎用レジスター 00 から 31 まで (GPR)	レジスターごとに 0 から 31
乗算引用符で囲まれたレジスター (MQ)	0-31
固定小数点例外レジスター (XER)	0-31
浮動小数点レジスター 00 から 31 (FPR)	レジスターごとに 0 から 63
浮動小数点状況および制御レジスター (FPSCR)	0-31

処理装置は、ワード指向の固定小数点プロセッサーであり、ダブルワード指向の浮動小数点プロセッサーと連携して機能します。 マイクロプロセッサーは、32 ビットのワード位置合わせ命令を使用します。 これは、バイト・オペランド、ハーフワード・オペランド、および固定小数点の場合はワード・オペランド、浮動小数点の場合はワード・オペランドとダブルワード・オペランドの取り出しと保管を行います。 これらのフェッチと保管は、主ストレージと 32 個の汎用レジスターのセットの間、および主ストレージと 32 個の浮動小数点レジスターのセットの間で行われます。

命令フォーム

命令は 4 バイトの長さで、ワードで位置合わせされます。

すべての命令は 4 バイトの長さで、ワードで位置合わせされます。 したがって、プロセッサーが命令 (例えば、ブランチ命令) を取り出すと、下位 2 ビットは無視されます。 同様に、プロセッサーが命令アドレスを開発する場合、アドレスの下位 2 ビットは 0 です。

ビット 0 から 5 は常に命令コードを指定します。 多くの命令には、拡張命令コード (例えば、XO 形式命令) もあります。 命令の残りのビットには、1 つ以上のフィールドが含まれています。 各種命令フォームの代替フィールドは、以下のとおりです。

- **I フォーム**

ビット	値
0-5	OPCD (操作)
6-29	LI
30	AA
31	LK

- **B フォーム**

ビット	値
0-5	OPCD (操作)
6 ～ 10	BO
11-15	BI
16-29	BD
30	AA
31	LK

- **SC フォーム**

ビット	値
0-5	OPCD (操作)
6 ～ 10	///
11-15	///
16-29	///
30	Xo
31	/

- **D 形式**

ビット	値
0-5	OPCD (操作)

ビット	値
6 ～ 10	RT、RS、FRT、FRS、TO、または BF、/、および L
11-15	RA
16-31	D、SI、または UI

- **DS 形式**

ビット	値
0-5	OPCD (操作)
6 ～ 10	RT または RS
11-15	RA
16-29	DS
30-31	Xo

- **X 命令形式**

ビット	値
0-5	OPCD (操作)
6 ～ 10	RT、FRT、RS、FRS、TO、BT、または BF、/、および L
11-15	RA、FRA、SR、SPR、または BFA および //
16-20	RB、FRB、SH、NB、または U and/
21-30	XO または EO
31	rc

  - **XL 命令フォーマット**

ビット	値
0-5	OPCD (操作)
6 ～ 10	RT または RS
11-20	spr または/、FXM および/
21-30	XO または EO
31	rc

  - **XFX 命令フォーマット**

ビット	値
0-5	OPCD (操作)
6 ～ 10	RT または RS
11-20	spr または/、FXM および/
21-30	XO または EO
31	rc

  - **XFL 命令形式**

ビット	値
0-5	OPCD (操作)
6	/
7-14	FLM (LM)
15	/
16-20	FRB
21-30	XO または EO
31	rc

- **XO 命令フォーマット**

ビット	値
0-5	OPCD (操作)
6 ～ 10	RT
11-15	RA
16-20	RB
21	大江
22-30	XO または EO
31	rc

- **A フォーム**

ビット	値
0-5	OPCD (操作)
6 ～ 10	FRT (R)
11-15	FRA
16-20	FRB
21-25	FRC
26-30	Xo
31	rc

- **M フォーム**

ビット	値
0-5	OPCD (操作)
6 ～ 10	RS
`11-15	RA
16-20	RB または SH
21-25	MB
26-30	ME
31	rc

一部の命令では、命令フィールドは予約されているか、または特定の値を含んでいなければなりません。これは、前の図には示されていませんが、これらの条件が必要な命令の構文に示されています。 予約フィールドのすべてのビットが 0 に設定されていない場合、または特定の値を含む必要があるフィールドにその値が含まれていない場合、命令フォームは無効です。

分割フィールドの表記

場合によっては、命令フィールドは、順列で使用される複数の連続したビット・シーケンスを占有します。このようなフィールドは、 分割フィールドと呼ばれます。

場合によっては、命令フィールドが複数の連続したビット・シーケンスを占有するか、または順列で使用される連続したビット・シーケンスを占有することがあります。 このようなフィールドは、 分割フィールドと呼ばれます。 前の図および個々の命令の構文では、分割フィールドの名前は、連続するビット・シーケンスごとに 1 つずつ小文字で示されています。 分割フィールドを持つ命令の記述、および分割フィールドの個々のビットが識別される他の特定の場所では、小文字のフィールドの名前は、シーケンスの左から右への連結を表します。 その他のすべての場合、フィールドの名前は大文字になり、シーケンスの連結を何らかの順序で表します。左から右にする必要はありません。 この順序は、影響を受ける命令ごとに説明されています。

命令フィールド

命令フィールドのセット。

項目	説明
AA (30) (AA (30))	絶対アドレス・ビットを指定します。 **0** 現在の命令アドレスに関連するアドレスを指定する即時フィールドを示します。 I 形式のブランチの場合、ブランチ・ターゲットの有効アドレスは、 LI フィールド符号拡張の 64 ビット (PowerPC ®) または 32 ビット (POWER ® ファミリー) とブランチ命令のアドレスの合計になります。 B 形式のブランチの場合、ブランチ・ターゲットの有効アドレスは、 BD フィールド符号拡張の 64 ビット (PowerPC ®) または 32 ビット (POWER ® ファミリー) とブランチ命令のアドレスの合計になります。 **1** 絶対アドレスを指定する即時フィールドを示します。 I 形式のブランチの場合、ブランチ・ターゲットの有効アドレスは、 LI フィールド符号が 64 ビット (PowerPC ®) または 32 ビット (POWER ® ファミリー) に拡張されます。 B 形式のブランチの場合、ブランチ・ターゲットの有効アドレスは、 BD フィールド符号が 64 ビット (PowerPC ®) または 32 ビット (POWER ® ファミリー) に拡張されます。
BA (11:15) (BA (11:15))	ソースとして使用する条件レジスター (CR) のビットを指定します。
BB (16:20) (BB (16:20))	ソースとして使用する CR 内のビットを指定します。
BD (16:29) (BD (16:29))	右側に 0b00 と連結され、64 ビット (PowerPC ®) または 32 ビット (POWER ® ファミリー) に符号拡張された、14 ビットの符号付き 2 の補数の分岐変位を指定します。 これは即時フィールドです。
BF (6: 8) (BF (6: 8))	CR フィールドの 1 つまたは浮動小数点状況および制御レジスター (FPSCR) フィールドの 1 つをターゲットとして指定します。 POWER ® ファミリーの場合、i = BF (6: 8) であれば、 i フィールドはレジスターのビット i*4 から (i*4) + 3 を参照します。
BFA (11:13)	いずれかの CR フィールドまたはいずれかの FPSCR フィールドをソースとして指定します。 POWER ® ファミリーの場合、j = BFA (11:13) であれば、 j フィールドはレジスターのビット j*4 から (j*4) + 3 を参照します。
BI (11:15) (BI (11:15))	分岐条件付き命令の条件として使用する CR 内のビットを指定します。

項目	説明
ビジネス・オブジェクト (6:10)	分岐条件付き命令のオプションを指定します。 BO フィールドに使用できるエンコードは、以下のとおりです。 **BO** **説明** **0000*x*** 減分カウント・レジスター (CTR)。 減分された CTR 値が 0 に等しくなく、条件が偽である場合に分岐します。 **0001*X*** CTR を減らします。 減分された CTR 値が 0 で、条件が偽の場合に分岐します。 **001*xx*** 条件が偽の場合に分岐します。 **0100*x*** CTR を減らします。 減分された CTR 値が 0 ではなく、条件が true の場合に分岐します。 **0101*x*** CTR を減らします。 減分された CTR 値が 0 に等しく、条件が true の場合に分岐します。 **011*x*** 条件が true の場合に分岐します。 **1*x*00*x*** CTR を減らします。 減分された CTR 値が 0 以外の場合に分岐します。 **1*x*01*x*** CTR を減らします。 CTR のビット 32 から 63 が 0 (PowerPC ®) の場合は分岐し、減分された CTR 値が 0 (POWER ® ファミリー) の場合は分岐します。 **1*x*1*xx*** 常に分岐します。
BT (6:10)	命令の結果のターゲットとして、CR または FPSCR 内のビットを指定します。
D (16:31) (D (16:31))	64 ビット (PowerPC ®) または 32 ビット (POWER ® ファミリー) に符号拡張された、16 ビット符号付き 2 の補数整数を指定します。 これは即時フィールドです。
EO (21:30) (EO (21:30))	X 形式命令で使用される a10-bit 拡張命令コードを指定します。
「**EO**」**(22:30)**	XO 形式命令で使用される 9 ビット拡張命令コードを指定します。
FL1 (16:19)	**svc** (監視プログラム呼び出し) 命令の 4 ビット・フィールドを指定します。
FL2 (27:29)	**svc** 命令の 3 ビット・フィールドを指定します。

項目 **説明**

FLM (7:14) **mtfsf** 命令によって更新される FPSCR フィールドを指定するフィールド・マスクを指定します。

ビット
説明

7
FPSCR フィールド 0 (ビット 00:03)

8
FPSCR フィールド 1 (ビット 04:07)

9
FPSCR フィールド 2 (ビット 08:11)

10
FPSCR フィールド 3 (ビット 12:15)

11
FPSCR フィールド 4 (ビット 16:19)

12
FPSCR フィールド 5 (ビット 20:23)

13
FPSCR フィールド 6 (ビット 24:27)

14
FPSCR フィールド 7 (ビット 28:31)

FRA (11:15) 浮動小数点レジスター (FPR) を演算のソースとして指定します。

FRB (16:20) FPR を操作のソースとして指定します。

FRC (21:25) FPR を操作のソースとして指定します。

FRS (6:10) FPR を操作のソースとして指定します。

FRT (6:10) 操作のターゲットとして FPR を指定します。

FXM (12:19) **mtcrf** 命令によって更新される CR フィールドを指定するフィールド・マスクを指定します。

ビット
説明

12
CR フィールド 0 (ビット 00:03)

13
CR フィールド 1 (ビット 04:07)

14
CR フィールド 2 (ビット 08:11)

15
CR フィールド 3 (ビット 12:15)

16
CR フィールド 4 (ビット 16:19)

17
CR フィールド 5 (ビット 20:23)

18
CR フィールド 6 (ビット 24:27)

19
CR フィールド 7 (ビット 28:31)

項目	説明
I (16:19) (I (16:19))	FPSCR 内のフィールドに入れるデータを指定します。 これは即時フィールドです。
LEV (20:26) (LEV (20:26))	これは、SA フィールドが 0 の場合に、b '1' | | LEV | | b ' 00000 によって **svc** ルーチンをアドレッシングする、 **svc** 命令の即時フィールドです。
LI (6:29) (LI (6:29))	右側に `0b00` と連結され、64 ビット (PowerPC ®) または 32 ビット (POWER ® ファミリー) に符号拡張された 24 ビット符号付き 2 の補数整数を指定します。 これは即時フィールドです。
LK (31) (LK (31))	リンク・ビット: **0** リンク・レジスターを設定しません。 **1** リンク・レジスターを設定します。 命令が分岐命令の場合には, 分岐命令に続く命令のアドレスがリンク・レジスターに入れられます。 命令が **svc** 命令の場合、 **svc** 命令に続く命令のアドレスがリンク・レジスターに入れられます。

MB (21:25) および ME (26:30)	(POWER ® ファミリー) 32 ビットのストリングを指定します。 このストリングは、ゼロで囲まれた 1 つのサブストリング、または 1 で囲まれたゼロのサブストリングで構成されます。 エンコードは以下のとおりです。
	MB (21:25)
	1 つのサブストリングの開始ビットの索引。
	ME (26:30) (ME (26:30))
	1 つのサブストリングのビットを停止する索引。
	`Let mstart=MB and mstop=ME:`
	`If mstart < mstop + 1 then`
	`mask(mstart..mstop) = ones`
	`mask(all other) = zeros`
	`If mstart = mstop + 1 then`
	`mask(0:31) = ones`
	`If mstart > mstop + 1 then`
	`mask(mstop+1..mstart-1) = zeros`
	`mask(all other) = ones`

NB (16:20) (NB (16:20))	即時ストリング・ロードまたは保管で移動するバイト数を指定します。
OPCD (0: 5)	基本命令コード・フィールド。
OE (21)	拡張算術計算のために XER の 0V および S0 フィールドを設定できるようにします。
RA (11:15) (RA (11:15))	ソースまたはターゲットとして使用する汎用レジスター (GPR) を指定します。
RB (16:20)	ソースとして使用する GPR を指定します。

項目	説明
Rc (31) (RC (31))	レコード・ビット: **0** CR を設定しません。 **1** 操作の結果を反映するように CR を設定します。 固定小数点命令の場合、CR ビット (0: 3) は符号付き数量として結果を反映するように設定されます。 結果が符号なしの数量であるか、ビット・ストリングであるかは、EQ ビットから判別できます。 浮動小数点命令の場合、CR ビット (4: 7) は、浮動小数点例外、浮動小数点使用可能例外、浮動小数点無効演算例外、および浮動小数点オーバーフロー例外を反映するように設定されます。
RS (6:10) (RS (6:10))	ソースとして使用する GPR を指定します。
RT (6:10)	ターゲットとして使用する GPR を指定します。
SA (30) (SA (30))	SVC 絶対: **0** アドレス `'1' || LEV || b'00000'` の **SVC** ルーチン **1** アドレス `x'1FE0'` の **svc** ルーチン
SH (16:20) (SH (16:20))	シフト量を指定します。
SI (16:31) (SI (16:31))	16 ビットの符号付き整数を指定します。 これは即時フィールドです。
SPR (11:20) (SPR (11:20))	**mtspr** および **mfspr** 命令の SPR を指定します。 SPR エンコードについては、 **mtspr** および **mfspr** の説明を参照してください。
サービス要求 (11:15)	16 個のセグメント・レジスターの 1 つを指定します。 ビット 11 は無視されます。
宛先 (6:10)	トラップする条件を指定します。 条件エンコードについて詳しくは、 Fixed-Point Trap Instructions を参照してください。 **TO ビット** **条件付き AND 演算** **0** より小さいものを比較します。 **1** より大きい値を比較します。 **2** 等しいと比較します。 **3** 論理的により小さいものを比較します。 **4** 論理的により大きいものを比較します。
単位 (16:19)	FPSCR に入れるデータとして使用されます。 これは即時フィールドです。
UI (16:31)	16 ビットの符号なし整数を指定します。 これは即時フィールドです。

項目	説明
XO (21:30、22:30、26:30、または 30)	拡張命令コード・フィールド。

ブランチ・プロセッサー

分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

ブランチ・プロセッサーには、非特権命令に関連する以下の 3 つの 32 ビット・レジスターがあります。

- 条件レジスター
- リンク・レジスター
- カウント・レジスター

これらのレジスターは 32 ビット・レジスターです。 PowerPC ® アーキテクチャーは、32 ビットと 64 ビットの両方の実装をサポートします。

POWER ® ファミリーと PowerPC ® の両方の場合、ブランチ・プロセッサー命令には、ブランチ命令、条件レジスター・フィールドと論理命令、および PowerPC ® の場合はシステム・コール命令、POWER ® ファミリーの場合はスーパーバイザー・リンケージ命令が含まれます。

ブランチ命令

ブランチ命令は、命令実行の順序を変更するために使用されます。

ブランチ命令を使用して、命令実行の順序を変更します。

すべてのブランチ命令がワード境界にあるため、ブランチを実行するプロセッサーは、生成されたブランチ・ターゲット・アドレスのビット 30 と 31 を無視します。 すべてのブランチ命令は、非特権状態で使用できます。

ブランチ命令は、次の 4 つの方法のいずれかでターゲット・アドレスを計算します。

- ターゲット・アドレスは、定数とブランチ命令自体のアドレスの合計です。
- ターゲット・アドレスは、命令のオペランドとして与えられる絶対アドレスです。
- ターゲット・アドレスは、リンク・レジスターで検出されたアドレスです。
- ターゲット・アドレスは、カウント・レジスターで検出されたアドレスです。

これらの方法の最初の 2 つを使用すると、分岐命令の前にターゲット・アドレスを十分に計算して、ターゲット・パスに沿って命令をプリフェッチすることができます。

3 番目と 4 番目の方法を使用すると、ブランチ命令の前にリンク・レジスターまたはカウント・レジスターが十分にロードされていれば、ブランチ・パスに沿って命令をプリフェッチすることもできます。

ブランチ命令には、「無条件ブランチ」と「条件付きブランチ」が含まれます。 各種ターゲット・フォームでは、ブランチ命令は通常、無条件にのみブランチするか、無条件にブランチしてリターン・アドレスを提供するか、条件付きでのみブランチするか、または条件付きでブランチしてリターン・アドレスを提供します。 ブランチ命令のリンク・ビットが 1 に設定されている場合、リンク・レジスターは、呼び出されたサブルーチンが使用する戻りアドレスを保管するように変更されます。 戻りアドレスは、ブランチ命令の直後の命令のアドレスです。

アセンブラーは、 BO フィールドのみ、または BO フィールドと部分的な BI フィールドをニーモニックに組み込むブランチ命令に対して、さまざまな拡張ニーモニックをサポートします。

システム・コール命令

PowerPC ® システム・コール命令は、サービスを実行するための割り込みまたはシステムを生成します。

PowerPC ® システムコール命令は、POWER ® ファミリーではスーパーバイザーコール命令と呼ばれます。 どちらのタイプの命令も、システムがサービスを実行するための割り込みを生成します。 システム・コールおよび監視プログラム・コール命令は、以下のとおりです。

- sc (システム・コール) 命令 (PowerPC ®)
- svc (監視プログラム呼び出し) 命令 (POWER ® ファミリー)

条件レジスター命令

条件レジスター命令は、1 つの CR を別の CR フィールドにコピーするか、または CR ビットに対して論理演算を実行します。

条件レジスター命令は、1 つの CR フィールドを別の CR フィールドにコピーするか、または CR ビットに対して論理演算を実行します。 アセンブラーは、条件レジスター命令用にいくつかの拡張簡略記号をサポートします。

固定小数点プロセッサー

固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

PowerPC ® 固定小数点プロセッサーは、以下のレジスターを非特権命令に使用します。

- 32 ビット汎用レジスター (GPR) (32 ビット)。
- 1 つの 32 ビット固定小数点例外レジスター。

POWER ® ファミリーの固定小数点プロセッサーは、非特権命令に以下のレジスターを使用します。 これらのレジスターは次のとおりです。

- 32 ビット GPR (32 ビット)
- 1 つの 32 ビット固定小数点例外レジスター
- 1 つの 32 ビット Multiply-Quotient (MQ) レジスター

GPR は、固定小数点プロセッサーの基本内部ストレージ・メカニズムです。

関連概念
lhz (ロード・ハーフおよびゼロ) 命令

固定小数点ロードおよび保管の指示

固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。 ロード命令は、データを移動するときに EA を計算します。 ストレージ・アクセスによって位置合わせ割り込みまたはデータ・ストレージ割り込みが発生しない場合は、EA によってアドレス指定されたバイト、ハーフワード、またはワードがターゲット GPR にロードされます。

固定小数点保管命令は、逆機能を実行します。 ストレージ・アクセスによって位置合わせ割り込みまたはデータ・ストレージ割り込みが発生しない場合、ソース GPR の内容は、EA によってアドレス指定されたストレージ内のバイト、ハーフワード、またはワードに保管されます。

ユーザー・プログラムでは、位置合わせされていないデータ位置にアクセスするロード命令および保管命令 (例えば、ワード境界上にないワードをロードしようとする試み) は実行されますが、パフォーマンスが低下する可能性があります。 ハードウェアが位置合わせされていない操作を実行するか、位置合わせ割り込みが発生し、位置合わせされていない操作を実行するためにオペレーティング・システムの位置合わせ割り込みハンドラーが呼び出されます。

固定小数点ロードおよび保管 (更新指示あり)

ロード・インストラクションとストア・インストラクションには更新フォームがあります。このフォームでは、メモリーとの間で通常の情報の移動に加えて、基本 GPR が EA で更新されます。

ロード・インストラクションとストア・インストラクションには「更新」フォームがあります。このフォームでは、メモリーとの間での通常の情報移動に加えて、基本 GPR が EA で更新されます。

POWER® ファミリーのロード命令の場合、EA が基本 GPR に保存されないという結果になる条件が 4 つあります。

1. 更新対象の GPR は、ターゲット GPR と同じです。 この場合、更新されたレジスターには、メモリーからロードされたデータが含まれます。
2. 更新される GPR は GPR 0 です。
3. ストレージ・アクセスにより、位置合わせ割り込みが発生します。
4. ストレージ・アクセスにより、データ・ストレージ割り込みが発生します。

POWER® ファミリー・ストア命令の場合、条件 2、3、および 4 により、EA は基本 GPR に保存されません。

PowerPC® のロードと保管の命令の場合、上記の条件 1 と 2 は無効な命令フォームになります。

ユーザー・プログラムでは、位置合わせされていないデータ位置にアクセスする更新命令のロードおよび保管は、基礎となるオペレーティング・システムのハードウェアまたは位置合わせ割り込みハンドラーによって実行されます。 位置合わせ割り込みにより、EA はベース GPR に含まれなくなります。

固定小数点ストリング命令

固定小数点ストリング命令を使用すると、位置合わせを考慮せずに、ストレージからレジスターへ、またはレジスターからストレージへデータを移動することができます。

固定小数点ストリング命令を使用すると、位置合わせを考慮せずに、ストレージからレジスターへ、またはレジスターからストレージへデータを移動することができます。 これらの命令は、任意のストレージ・ロケーション間の短い移動に使用することも、位置合わせされていないストレージ・フィールド間の長い移動を開始するために使用することもできます。 長さがゼロの Load String Indexed 命令および Store String Indexed 命令は、ターゲット・レジスターを変更しません。

固定小数点アドレス計算命令

POWER® ファミリーの各種アドレス計算命令は、PowerPC® の演算命令に統合されています。

POWER® ファミリーには、いくつかのアドレス計算命令があります。 これらは PowerPC® の演算命令にマージされます。

固定小数点演算命令

固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。 いくつかの減算ニーモニックが、加算ニーモニックの拡張ニーモニックとして提供されています。 これらの拡張ニーモニックについては、90 ページの『条件レジスター論理命令の拡張ニーモニック』 を参照してください。

POWER® ファミリーと PowerPC® の違いは、すべての固定小数点除算命令と、いくつかの固定小数点乗算命令にあります。 両方のアーキテクチャーで実行されるプログラムをアセンブルするには、除算および乗算用のミリコード・ルーチンを使用する必要があります。 使用可能なミリコード・ルーチンについては、「ミリコード・ルーチンの使用」を参照してください。

固定小数点比較命令

固定小数点比較命令は、レジスター RA の内容を代数的または論理的に比較します。

固定小数点比較命令は、レジスター RA の内容を以下のいずれかと代数的または論理的に比較します。

- SI フィールドの符号付き拡張値。
- UI フィールド
- レジスター RB の内容

代数比較は、2 つの符号付き整数を比較します。 論理比較は、2 つの符号なし整数を比較します。

POWER ® ファミリーと PowerPC ® にはさまざまな入力オペランド・フォーマットがあります。例えば、PowerPC ® には L オペランドがあります。また、PowerPC ® には無効な命令フォームの制限があります。アセンブラーは、PowerPC ® アセンブリー・モードで無効な命令フォームがないか検査します。

固定小数点比較命令の拡張簡略記号については、 91 ページの『固定小数点算術命令の拡張簡略記号』で説明しています。

固定小数点トラップ命令

固定小数点トラップ命令は、指定された条件のセットについてテストします。

固定小数点トラップ命令は、指定された条件のセットについてテストします。 トラップは、範囲外の索引や無効文字の使用など、プログラムの実行中に発生してはならないイベントに対して定義できます。 定義済みトラップ条件が発生すると、プログラム割り込みを処理するためにシステム・トラップ・ハンドラーが呼び出されます。 定義されたトラップ条件が発生しない場合は、通常のプログラム実行が継続されます。

レジスター RA の内容は、個々のトラップ命令に応じて、符号拡張 SI フィールドまたはレジスター RB の内容と比較されます。 32 ビット・インプリメンテーションでは、レジスター RA および RB の下位 32 ビットの内容のみが比較に使用されます。

比較結果は、 TO フィールドと AND 演算される 5 つの条件になります。 結果が 0 でない場合は、システム・トラップ・ハンドラーが呼び出されます。 結果の 5 つの条件は、以下のとおりです。

TO フィールド・ビット	**条件付き AND 演算**
0	より小
1	より大きい
2	等しい
3	論理的により小さい
4	論理的により大きい

最も便利な TO フィールド値のための拡張ニーモニックが提供されており、トラップ条件の最も一般的な組み合わせのための標準コード・セットが提供されています。

固定小数点論理命令

固定小数点論理命令は、論理演算をビット単位で実行します。

固定小数点論理命令は、論理演算をビット単位で実行します。 ノーオペレーション命令の拡張ニーモニック、および OR 命令と NOR 命令については、 90 ページの『条件レジスター論理命令の拡張ニーモニック』で説明しています。

固定小数点回転およびシフト指示

固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

固定小数点プロセッサーは、GPR からのデータに対して循環操作を実行します。 これらの命令は、以下のいずれかの方法でレジスターの内容を循環させます。

- 回転の結果は、マスクの制御下でターゲット・レジスターに挿入されます。 マスク・ビットが 1 の場合は、循環データの関連ビットがターゲット・レジスターに入れられます。 マスク・ビットが 0 の場合、ターゲット・レジスター内の関連データ・ビットは変更されません。
- 回転の結果は、ターゲット・レジスターに入れられる前にマスクと AND 演算されます。

左に回転する命令により、(概念的には) レジスタの内容を右に回転させることができる。 32 ビット実装の場合、 *n* ビットの右回転は、32-*n* の左回転によって実行できます。

固定小数点シフト命令は、論理的に左シフトと右シフトを実行します。 シフト命令の結果は、生成されたマスクの制御下でターゲット・レジスターに入れられます。

一部の POWER® ファミリーのシフト命令には、MQ レジスターが含まれます。 このレジスターも更新されます。

拡張簡略記号は、抽出、挿入、回転、シフト、クリア、および左シフトと左シフトの操作用に提供されています。

関連概念
rlwinm または rlinm (左方ワードの即時回転、マスク付き AND) 命令

特殊目的のレジスター命令との間での固定小数点移動

この命令は、ある特殊目的レジスター (SPR) の内容を別の SPR または汎用レジスター (GPR) に移動します。これには、非特権命令と特権命令の両方が含まれます。

いくつかの命令は、1 つの特殊目的レジスター (SPR) の内容を別の SPR または汎用レジスター (GPR) に移動します。 これらの命令は、拡張簡略記号に組み込まれた各 SPR エンコードを持つ拡張簡略記号のセットによってサポートされます。 これには、非特権命令と特権命令の両方が含まれます。

> **注:** SPR フィールド長は、PowerPC® の場合は 10 ビット、POWER® ファミリーの場合は 5 ビットです。 POWER® ファミリーと PowerPC® のソース・コードの互換性を維持するには、 **mfspr** 命令または **mtspr** 命令への入力オペランドとして使用する前に、SPR 番号の下位 5 ビットと上位 5 ビットを逆にする必要があります。 **mfspr** 命令と **mtspr** 命令のエンコード・テーブルで定義されている数値は、既に下位 5 ビットと上位 5 ビットが逆になっています。 **dbx** コマンドを使用してプログラムをデバッグするときは、SPR 番号の下位 5 ビットと上位 5 ビットが **dbx** コマンドからの出力で逆になることに注意してください。

POWER® ファミリーと PowerPC® には異なる SPR セットがあります。同じ SPR のエンコードは、DEC (減分) からの移行を除き、POWER® ファミリーと PowerPC® で同一です。 SPR。

DEC SPR からの移行は、PowerPC® では特権が付与されますが、POWER® ファミリーでは非特権です。 SPR フィールドの 1 ビットは、特権命令の場合は 1、非特権命令の場合は 0 です。 したがって、 **mfdec** 命令の DEC SPR のエンコード番号は、PowerPC® と POWER® ファミリーでは異なる値になります。 DEC エンコード番号は、PowerPC® の場合は 22、POWER® ファミリーの場合は 6 です。 **mfdec** 命令を使用する場合、アセンブラーは現在のアセンブリー・モードに基づいて DEC エンコードを決定します。 以下のリストは、アセンブリー・モード値ごとの **mfdec** 命令のアセンブラー処理を示しています。

- アセンブリー・モードが **pwr**、 **pwr2**、または **601** の場合、DEC エンコードは 6 です。
- アセンブリー・モードが **ppc**、 **603**、または **604** の場合、DEC エンコードは 22 です。
- デフォルトのアセンブリー・モード (POWER® ファミリー /PowerPC® の非互換性エラーを指示警告として扱う) が使用される場合、DEC エンコードは 6 です。 説明警告 158 は、オブジェクト・コードの生成に DEC SPR エンコード 6 が使用されることを報告します。 この警告は、 **-W** フラグを使用して抑止できます。
- アセンブリー・モードが **any** の場合、DEC エンコードは 6 です。 **-w** フラグを使用すると、警告メッセージ (158) により、オブジェクト・コードの生成に DEC SPR エンコード 6 が使用されることが報告されます。
- アセンブリー・モードが **com** の場合、 **mfdec** 命令がサポートされていないことを示すエラー・メッセージが報告されます。 オブジェクト・コードは生成されません。 この状況では、 **mfspr** 命令を使用して DEC 番号をエンコードする必要があります。

関連概念
固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

浮動小数点プロセッサー

浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

POWER ® ファミリーと PowerPC ® 浮動小数点プロセッサーには、非特権命令用に同じレジスターが設定されています。 レジスターは以下のとおりです。

- 32 個の 64 ビット浮動小数点レジスター
- 1 つの 32 ビット浮動小数点状況および制御レジスター (FPSCR)

浮動小数点プロセッサーは、浮動小数点演算のハイパフォーマンス実行を提供します。 浮動小数点レジスターで算術演算、比較演算、およびその他の演算を実行し、ストレージと浮動小数点レジスターの間で浮動小数点データを移動するための命令が提供されています。

PowerPC ® および POWER2™ は、浮動小数点レジスターでの変換操作もサポートします。

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

浮動小数点数

浮動小数点数は符号付き仮数で構成され、符号付き小数部と数値の積である数量を表します。

浮動小数点数は、符号付き指数と符号付き仮数で構成され、符号付き小数部と数値 **2 * *** 指数の積である数量を表します。 エンコードは、以下を表すためにデータ・フォーマットで提供されます。

- 有限数値
- +-無限大
- 「数値ではない」(NaN) の値

無限大を伴う演算は、従来の数学的規則に従って結果を生成します。 NaN には数学的な解釈はありません。 これらのエンコードでは、可変診断情報フィールドが許可されます。 これらは、初期化されていない変数を示すために使用されることがあり、特定の無効な操作によって生成されることがあります。

浮動小数点レジスターの内容の解釈

32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

32 個の 64 ビット浮動小数点レジスターがあり、浮動小数点レジスター 0 から 31 までの番号が付けられています。 すべての浮動小数点命令は、命令の実行に使用する浮動小数点レジスターを指定する 5 ビット・フィールドを提供します。 浮動小数点レジスターの内容を浮動小数点値として解釈するすべての命令は、この解釈に倍精度浮動小数点形式を使用します。

ロードおよび保管以外のすべての浮動小数点命令は、浮動小数点レジスターにあるオペランドに対して実行され、その結果を浮動小数点レジスターに入れます。 浮動小数点状況および制御レジスターおよび条件レジスターは、一部の浮動小数点演算の結果に関する状況情報を保持します。

ロードと保管の二重命令は、浮動小数点プロセッサーのストレージと浮動小数点レジスターの間で変換を行わずに、64 ビットのデータを転送します。 単一ロード命令は、保管された単一浮動形式の値を倍精度浮動形式の同じ値に変換し、その値を浮動小数点レジスターに転送します。 単一保管命令は、ストレージの前に、浮動小数点レジスター内の有効な単精度値を単一の浮動形式の値に変換することによって、その逆を行います。

浮動小数点ロードおよび保管命令

単精度および倍精度の浮動小数点ロード命令が提供されます。 倍精度データは、浮動小数点レジスターに直接ロードされます。 浮動小数点レジスターは浮動小数点倍精度オペランドのみをサポートするため、プロセッサーは、データを浮動小数点レジスターにロードする前に単精度データを倍精度に変換します。

単精度および倍精度の浮動小数点保管命令が提供されます。 単精度ストアは、浮動小数点レジスターの内容を、ストレージの前に単精度に変換します。

POWER2™ は、ロードおよび保管浮動小数点クワッド命令を提供します。 これらは主に、配列演算など、大量の数値に対する算術演算のパフォーマンスを向上させるためのものです。 通常、データ・アクセスは、これらのタイプの操作のパフォーマンス・ボトルネックになります。 これらの命令は、1 回のロードまたは保管操作 (つまり、1 つのストレージ参照) で、64 ビットではなく 128 ビットのデータを転送します。 128 ビットのデータは、1 つの 4 倍長ワード・オペランドとしてではなく、2 つのダブルワード・オペランドとして扱われます。

浮動小数点移動命令

浮動小数点移動命令は、ある FPR から別の FPR にデータをコピーします。

浮動小数点移動命令は、ある FPR から別の FPR にデータをコピーし、それぞれの命令について記述されているようにデータを変更します。 これらの指示によって FPSCR が変更されることはありません。

浮動小数点演算命令

浮動小数点演算命令は、浮動小数点レジスターに含まれる浮動小数点データに対して算術演算を実行します。

浮動小数点演算命令は、浮動小数点レジスターに含まれる浮動小数点データに対して算術演算を実行します。

浮動小数点の乗算加算命令

浮動小数点乗算加算命令は、中間丸め演算を行わずに乗算演算と加算演算を結合します。

浮動小数点乗算加算命令は、中間丸め演算を行わずに、乗算演算と加算演算を結合します。 中間積の小数部分の幅は 106 ビットで、106 ビットすべてが命令の加算部分または減算部分に使用されます。

浮動小数点比較命令

浮動小数点比較命令は、2 つの FPR の内容の順序付き比較および順序なし比較を実行します。

浮動小数点比較命令は、2 つの FPR の内容の順序付き比較および順序なし比較を実行します。 BF フィールドによって指定される CR フィールドは、比較の結果に基づいて設定されます。 比較により、指定された CR フィールドの 1 ビットが 1 に設定され、他のすべてのビットが 0 に設定されます。 浮動小数点条件コード (FPCC) (ビット 16:19) も同じ方法で設定されます。

CR フィールドと FPCC は、次のように解釈されます。

項目	説明	説明
条件-フィールドおよび浮動小数点条件コードの解釈の登録	**条件-フィールドおよび浮動小数点条件コードの解釈の登録**	**条件-フィールドおよび浮動小数点条件コードの解釈の登録**
ビット	**Name**	**説明**
0	FL	(FRA) < (FRB)
1	FG	(FRA)> (FRB)
2	FE	(FRA) = (FRB)

項目	説明	説明
3	FU (U)	(FRA)? (FRB) (非順序)

浮動小数点変換命令

浮動小数点変換命令は、FPR 内の浮動小数点オペランドを 32 ビット符号付き固定小数点レジスターに変換します。

浮動小数点変換命令は、PowerPC ® および POWER2™に対してのみ提供されます。 これらの命令は、FPR 内の浮動小数点オペランドを 32 ビットの符号付き固定小数点整数に変換します。 CR1 フィールドと FPSCR が変更されます。

浮動小数点状況および制御レジスター命令

浮動小数点状況および制御レジスター命令は、FPSCR 内のデータを操作します。

浮動小数点状況および制御レジスター命令は、FPSCR 内のデータを操作します。

構文およびセマンティクス

アセンブラー言語の構文とセマンティクスが定義されます。

この概要では、以下の項目を含む、アセンブラー言語の構文とセマンティクスについて説明します。

文字セット

オペレーティング・システムのアセンブラー言語には定義済みの文字があります。

すべての文字と数字を使用できます。 アセンブラーは、大文字と小文字を区別します。 アセンブラーに対しては、変数 *Name* および *name* は特殊シンボルを識別します。

一部のブランク・スペースは必須ですが、その他のブランク・スペースはオプションです。 アセンブラーでは、スペースの代わりにタブを使用することができます。

以下の文字は、オペレーティング・システムのアセンブラー言語では特別な意味を持ちます。

項目

説明

, (コンマ)

オペランド分離文字。 コンマは、オペランド間のステートメントでのみ使用できます。以下に例を示します。

```
a 3,4,5
```

 # (ポンド記号)

コメント。 # の後から行の終わりまでのすべてのテキストは、アセンブラーによって無視されます。 # は、行の先頭文字にすることも、任意の数の文字、ブランク・スペース、またはその両方を前に付けることもできます。 次に例を示します。

```
a 3,4,5 # Puts the sum of GPR4 and GPR5 into GPR3.
```

: (コロン)

ラベルを定義します。 **:** は、常にラベル名の最後の文字の直後に表示され、アセンブラーがラベルを検出したときにロケーション・カウンターに含まれていた値と等しいラベルを定義します。 次に例を示します。

```
add: a 3,4,5    # Puts add equal to the address
                # where the a instruction is found.
```

項目	説明

; (セミコロン)	命令分離文字。 セミコロンは、同じ行に現れる 2 つの命令を分離します。 セミコロンの前後のスペースはオプションです。 1 行に 1 つの命令を指定する場合は、セミコロンで終了する必要はありません。
	アセンブラー・リストを明瞭かつ理解しやすいものにするために、各行には 1 つの命令のみを含めることをお勧めします。 次に例を示します。
	`a 3,4,5 # These two lines have`
	`a 4,3,5 # the same effect as...`
	`a 3,4,5; a 4,3,5 # ...this line.`

$(ドル記号)	アセンブラーの現行ロケーション・カウンターの現行値を参照します。 次に例を示します。
	`dino: .long 1,2,3`
	`size: .long $ - dino`

@ (アットマーク)	明示的な再配置タイプを指定するために、式の中でシンボル名の後に使用されます。

予約語

オペレーティング・システムのアセンブラー言語に予約語はありません。

オペレーティング・システムのアセンブラー言語に予約語はありません。 命令および疑似命令のニーモニックは予約されていません。 これらは、他のシンボルと同じ方法で使用できます。

他の言語で作成されたプログラムに渡されるシンボルの名前に制限がある場合があります。

線の形式

アセンブラーは、ソース・ファイルのフリー・ライン形式をサポートします。

アセンブラーは、ソース行のフリー・ライン・フォーマットをサポートします。このフォーマットでは、項目が特定の桁位置にある必要はありません。

すべての命令について、読みやすくするために、ステートメントの簡略記号とオペランドの間に分離文字 (スペースまたはタブ) を入れることをお勧めします。 AIX® アセンブラーでは、分岐条件付き命令は、アセンブラーによる明確な処理のために、ニーモニックとオペランドの間に分離文字 (スペースまたはタブ) を必要とします。

アセンブリー言語では、単一の入力行に表示できる文字数に制限はありません。 コード行が端末上の 1 行より長い場合､行の折り返しは使用されるエディターによって異なります。 ただし､リストには 1 行に 512 文字の ASCII 文字のみが表示されます。

ブランク行は許可されますが、アセンブラーはそれらを無視します。

関連概念

簡略記号の後に分離文字を付けない分岐条件ステートメントの移行

AIX® アセンブラーは、前のバージョンのアセンブラーとは異なるいくつかのステートメントを構文解析します。

関連情報

atof

ステートメント

アセンブラー言語には、命令ステートメント、疑似命令ステートメント、およびヌル・ステートメントの 3 種類のステートメントがあります。 アセンブラーは、区切り文字、ラベル、ニーモニック、オペランド、およびコメントも使用します。

命令ステートメントおよび疑似命令ステートメント

命令または疑似命令ステートメントには、事前定義された構文があります。

命令または疑似命令ステートメントの構文は、次のとおりです。

[*label*:] ニーモニック [*operand1*[,*operand2*...]] [# *comment*]

以下のいずれかが表示されると、アセンブラーはステートメントの終わりを認識します。

- ASCII の改行文字。
- # (ポンド記号) (コメント文字)
- A; (セミコロン)

ヌル・ステートメント

ヌル・ステートメントは、主にアセンブラー・ソース・コードを読みやすくするために役立ちます。

ヌル・ステートメントには、簡略記号またはオペランドはありません。 ラベル、コメント、またはその両方を含めることができます。 ヌル・ステートメントを処理しても、ロケーション・カウンターの値は変更されません。

ヌル・ステートメントは、主にアセンブラー・ソース・コードを読みやすくするために役立ちます。

ヌル・ステートメントの構文は、次のとおりです。

[*label*:] [# コメント]

ラベルとコメントの間のスペースはオプションです。

NULL ステートメントにラベルがある場合は、次のステートメントが別の行にあっても、ラベルは次のステートメントの値を受け取ります。 アセンブラーは、現行ロケーション・カウンターに含まれている値をラベルに与えます。 次に例を示します。

```
here:
       a 3,4,5
```

次と同義

```
here:  a 3,4,5
```

注 : ある種の疑似命令 (**.csect**、 **.comm**、および **.lcomm** など) は、ヌル・ステートメントのラベルが次のステートメントのアドレスの値を受け取らないようにすることができます。

区切り文字

区切り文字は、スペース、タブ、およびコンマです。

区切り文字は、スペース、タブ、およびコンマです。 オペランドはコンマで区切ります。 スペースまたはタブは、ステートメントの他の部分を分離します。 このブックでスペースが表示されている場所であればどこでも、タブを使用できます。

命令または疑似命令の構文に示されているスペースは、必須です。

分岐条件付き命令では、アセンブラーによる明確な処理のために、ニーモニックとオペランドの間に分離文字 (スペースまたはタブ) が必要です。

オプションで、1 つ以上のスペースをコンマの後、ポンド記号 (#) の前、および # の後に置くことができます。

ラベル

ラベル項目はオプションです。 アセンブラーは、アセンブラーの現行ロケーション・カウンターに含まれている値をラベルに与えます。

ラベル項目はオプションです。 行には、0 個、1 個、または複数個のラベルを付けることができます。 さらに、線はラベルを持つことができますが、その他の内容を持つことはできません。

ラベルを定義するには、: (コロン) の前にシンボルを置きます。 アセンブラーは、アセンブラーの現行ロケーション・カウンターに含まれている値をラベルに与えます。 この値は、再配置可能アドレスを表します。 次に例を示します。

```
subtr:   sf 3,4,5
# The label subtr: receives the value
# of the address of the sf instruction.
# You can now use subtr in subsequent statements
# to refer to this address.
```

ラベルが、データ位置合わせの原因となる命令を持つステートメント内にある場合、ラベルは、位置合わせが行われる前にその値を受け取ります。 次に例を示します。

```
# Assume that the location counter now
# contains the value of 98.
place:   .long expr
# When the assembler processes this statement, it
# sets place to address 98. But the .long is a pseudo-op that
# aligns expr on a fullword. Thus, the assembler puts
# expr at the next available fullword boundary, which is
# address 100. In this case, place is not actually the address
# at which expr is stored; referring to place will not put you
# at the location of expr.
```

簡略記号

簡略記号フィールドは、ステートメントが命令ステートメントであるか、疑似命令ステートメントであるかを識別します。

簡略記号フィールドは、ステートメントが命令ステートメントであるか、疑似命令ステートメントであるかを識別します。 各ニーモニックには、特定のフォーマットの特定の数のオペランドが必要です。

命令ステートメントの場合、簡略記号フィールドには、ai (即時追加) または sf (減算元) のような省略形が含まれます。 このニーモニックは、システム・マイクロプロセッサーが数値演算コード (命令コード) に関連付けられた単一のマシン・インストラクションを処理する操作を記述します。 すべての命令の長さは 4 バイトです。 アセンブラーが命令を検出すると、アセンブラーは必要なバイト数だけロケーション・カウンターを増分します。

疑似命令ステートメントの場合、簡略記号はアセンブラー・プログラム自体に対する命令を表します。 関連する命令コードはなく、ニーモニックはプロセッサーに対する操作を記述しません。 疑似命令の中には、ロケーション・カウンターを増分するものもあれば、増分しないものもあります。

オペランド

オペランドの存在と意味は、使用されるニーモニックによって異なります。

オペランドの存在と意味は、使用されるニーモニックによって異なります。 ニーモニックによっては、オペランドを必要としないものもあります。 その他のニーモニックには、1 つ以上のオペランドが必要です。

アセンブラーは、オペランドの簡略記号を使用して、コンテキスト内の各オペランドを解釈します。 多くのオペランドは、レジスターまたはシンボルを参照する式です。 命令ステートメントの場合、オペランドは命令に直接アセンブルされる即時データにすることができます。

コメント

コメント・オプションは、アセンブラーではオプションです。

コメントはオプションであり、アセンブラーによって無視されます。 コメントの各行の前には # (ポンド記号) を付ける必要があります。コメントを指定する他の方法はありません。

シンボル

記号はラベル・オペランドとして使用されます。

シンボルは、ラベルまたはオペランドとして使用される単一文字または文字の組み合わせです。

シンボルの構成

記号は、数字、下線、ピリオド、または小文字で構成されます。

記号は、数字、下線、ピリオド、大文字または小文字、あるいはこれらの任意の組み合わせで構成することができます。 記号にブランクまたは特殊文字を含めることはできません。また、数字で始めることもできません。 英大文字と小文字は別です。 シンボル名の最大長は 65535 個の 1 バイト文字です。

シンボルにブランクまたはその他の特殊文字が含まれている必要がある場合、 **.rename** 疑似命令を使用すると、ローカル名をグローバル名の同義語または別名として使用することができます。

制御セクション (csect) または目次 (TOC) 項目名を除き、ストレージ・ロケーションまたは任意のデータを表すためにシンボルが使用されることがあります。 シンボルを 32 ビット・モードでアセンブルする場合、シンボルの値は常に 32 ビット数量になります。64 ビット・モードでシンボルをアセンブルする場合、値は常に 64 ビット数量になります。

シンボル名の有効な例を以下に示します。

- `READER`
- `XC2345`
- `result.a`
- `resultA`
- `balance_old`
- `_label9`
- `.myspot`

以下は有効なシンボル名ではありません。

項目	説明
`7_sum`	(数字で始まります。)
#ofcredits	(# はこれをコメントにします。)
`aa*1`	(*(特殊文字) を含みます。)
領域内	(ブランクが含まれます。)

シンボルを定義するには、以下の 2 つの方法のいずれかを使用します。

- 命令または疑似命令用のラベルとして
- **.set**、 **.comm**、 **.lcomm**、 **.dsect**、 **.csect**、または **.rename** pseudo-op の名前オペランドとして。

ラベルを使用したシンボルの定義

シンボルは、ラベルとして使用することによって定義できます。

ラベルとしてシンボルを使用して、シンボルを定義することができます。 次に例を示します。

```
              .using          dataval[RW],5
loop:
              bgt             cont
  .
  .
```

```
                bdz             loop
cont:           l               3,dataval
                a               4,3,4
   .
   .
.csect dataval[RW]
dataval:          .short    10
```

アセンブラーは、命令または疑似命令の左端のバイトに位置カウンターの値を与えます。 この例では、**l** 命令のオブジェクト・コードに dataval のロケーション・カウンター値が含まれています。

実行時に、*dataval* ラベル、オフセット、および GPR 5 からアドレスが計算されます。GPR 5 には、csect dataval[RW]のアドレスが含まれている必要があります。 この例では、l 命令は、 dataval ラベルのアドレスに保管されている 16 ビットのデータを使用します。

シンボルによって参照される値は、実際にはメモリー・ロケーションを占有します。 ラベルによって定義されるシンボルは、再配置可能な値です。

シンボル自体は実行時には存在しません。 ただし、一部のコードが dataval ラベルによって表される位置の内容を変更する場合は、実行時にシンボルによって表されるアドレスの値を変更することができます。

疑似命令を使用したシンボルの定義

シンボルを定義するには、疑似命令の名前オペランドとしてシンボルを使用します。

シンボルを定義するには、 **.set** 疑似命令の名前オペランドとしてシンボルを使用します。 この疑似命令の形式は次のとおりです。

.set *name*、*exp*

アセンブラーは *exp* オペランドを評価してから、 *exp* オペランドの値とタイプをシンボル *name* に割り当てます。 アセンブラーは、命令内でそのシンボルを検出すると、そのシンボルの値を命令のオブジェクト・コードに入れます。

次に例を示します。

```
       .set               number,10
   .
   .
        ai         4,4,number
```

上記の例では、ai 命令のオブジェクト・コードには、 number に割り当てられた値、つまり 10 が含まれています。

シンボルの値は、命令に直接アセンブルされ、ストレージ・スペースを占有しません。 **.set** 疑似命令で定義されたシンボルは、 *exp* オペランドのタイプに応じて、絶対タイプまたは再配置可能タイプを持つことができます。 また、シンボルはストレージを占有しないため、実行時にシンボルの値を変更することはできません。ファイルを再アセンブルすると、シンボルに新しい値が与えられます。

また、シンボルは、 **.comm**、 **.lcomm**、 **.csect**、 **.dsect**、または **.rename** pseudo-op の 名前 オペランドとして使用して定義することもできます。 **.dsect** 疑似命令の場合を除き、シンボルに割り当てられた値はストレージ・スペースを記述します。

CSECT 入り口名

シンボルは、 **csect** 疑似命令の *qualname* オペランドとして使用できます。

シンボルは、 **.csect** 疑似命令の *qualname* オペランドとして使用するときにも定義できます。 このコンテキストで使用される場合、シンボルは、指定されたストレージ・マッピング・クラスを持つ csect の名前として定義されます。 いったん定義されると、シンボルは、名前修飾子に対応するストレージ・マッピング・クラスを取ります。

qualname オペランドの形式は、次のとおりです。

シンボル**[***XX***]**

または

シンボル**{***XX***}**

ここで、 *XX* はストレージ・マッピング・クラスです。

スレッド・ローカル・シンボル

スレッド・ローカル・シンボルに使用される 2 つのストレージ・マッピング・クラスは、 **TL** および **UL** です。

スレッド・ローカル・シンボルに使用される 2 つのストレージ・マッピング・クラスは、 **TL** および **UL** です。 **TL** ストレージ・マッピング・クラスは、初期化されたスレッド・ローカル・ストレージを定義するために **.csect** 疑似命令と共に使用されます。 **UL** ストレージ・マッピング・クラスは、初期化されていないスレッド・ローカル・ストレージを定義するために、 **.comm** または **.lcomm** 疑似命令と一緒に使用されます。 スレッド・ローカル・シンボルと非スレッド・ローカル・シンボルを結合する式は許可されません。

特殊記号 toc

.toc 疑似命令は、TOC アンカー項目を作成します。

特殊シンボル TOC に関する規定が作成されました。 XCOFF フォーマット・モジュールでは、このシンボルは TOC アンカー用、または TOC 内の最初のエントリー用に予約されています。 シンボル TOC は、シンボル TOC を使用する必要がある場合に参照できるように、アセンブラーで事前定義されています。 **.toc** 疑似命令は、TOC アンカー項目を作成します。 例えば、以下のデータ宣言は、TOC の先頭のアドレスを含むワードを宣言します。

```
.long TOC[TC0]
```

この記号は、 **.toc** 疑似命令がアセンブラー・ファイル内に含まれていない限り、未定義です。

詳しくは、 519 ページの『.toc 疑似命令』を参照してください。

TOC エントリー名

シンボルは、 **.tc** 疑似命令の *Name* オペランドとして使用するときに定義できます。

シンボルは、 **.tc** 疑似命令の *Name* オペランドとして使用するときに定義できます。 この方法で使用される場合、シンボルは、TC のストレージ・マッピング・クラスを持つ TOC エントリーの名前として定義されます。

Name オペランドの形式は、次のとおりです。

シンボル**[TC]**

TOC 終了記号

TOC シンボルは、ストレージ・マッピング・クラスを使用します。

ほとんどの TOC シンボルは、ストレージ・マッピング・クラス TC を使用します。 これらのシンボルは、リンク時に任意の順序で収集されます。 TOC オーバーフローが発生した場合は、一部の TOC シンボルを TOC の末尾に移動し、代替コード・シーケンスを使用してこれらのシンボルを参照すると便利です。 末尾に移動するシンボルは、ストレージ・マッピング・クラス TE を使用する必要があります。 TE ストレージ・マッピング・クラスを持つシンボルは、TC ストレージ・マッピング・クラスを持つシンボルと同じように扱われます。ただし、シンボルが D 形式命令で使用されるときに選択される RLD に関しては例外です。

シンボルを定義する前にシンボルを使用する

シンボルは、使用する前に参照することができます。

シンボルを定義する前に、シンボルを使用することができます。 シンボルを使用し、それを後で同じファイルに定義することは、 順方向参照と呼ばれます。 例えば、以下のように指定できます。

```
# Assume that GPR 6 contains the address of .csect data[RW].
      l 5,ten(6)
```

```
      .
      .
.csect data[RW]
ten: .long 10
```

シンボルが出現するファイルに定義されていない場合は、外部シンボルまたは未定義シンボルである可能性があります。 アセンブラーは、未定義シンボルを検出すると、 **as** コマンドの **-u** フラグが使用されていない限り、エラー・メッセージを出力します。 外部シンボルは、490 ページの『.extern 疑似命令』を使用してステートメント内で宣言できます。

シンボルの可視性

可視性プロパティーは、リンカーがプログラムまたは共有オブジェクトを作成するときに使用するグローバル・シンボルに関連付けることができます。 シンボルの可視性は、 **.extern**、 **.globl**、 **.weak**、または **.comm** 疑似命令のオプション・パラメーターで指定されます。 シンボルには、以下の可視性プロパティーが定義されています。

エクスポート済み: シンボルはエクスポートされ、優先使用可能になります。
protected: シンボルはエクスポートされますが、優先使用可能ではありません。
hidden: シンボルはエクスポートされません。
internal: シンボルをエクスポートできません。

記号の可視性もリンカー・オプションの影響を受けます。

関連概念
.extern 疑似命令
.weak 疑似命令
.globl 疑似命令
関連情報
ld

グローバル・シンボルの宣言

外部シンボルおよび弱いシンボルを含むグローバル・シンボルは、リンク・プロセス中にシンボル解決に関与します。 その他のシンボルはローカルで、現行ソース・ファイルの外部では使用できません。 シンボルは、.**extern**、.**globl**、.**weak**、または.**comm** pseudo-op を使用して、グローバルとして宣言されます。

再配置指定子

再配置指定子は、シンボル名または QualNames の後に使用されます。

一般に、アセンブラーは式のタイプと使用法に基づいて適切な再配置を生成します。 ただし、場合によっては、複数の再配置タイプが可能であり、明示的な再配置指定子が必要です。

明示的な再配置指定子は、シンボル名または QualNames の後に使用できます。 @ (アットマーク) 文字と、1 文字または 2 文字の再配置タイプで構成されます。 再配置タイプは、大文字または小文字で指定できますが、大/小文字混合では指定できません。

以下の表に、有効な再配置タイプをリストします。

タイプ	RLD 名	使用法
u	R_TOCU	大規模な TOC 再配置
l	R_TOCL	大規模な TOC 再配置
GD	R_TLS	スレッド・ローカル・ストレージ
ie	R_TLS_IE	スレッド・ローカル・ストレージ
le	R_TLS_LE	スレッド・ローカル・ストレージ
ld	R_TLS_LD	スレッド・ローカル・ストレージ
m	R_TLSM	スレッド・ローカル・ストレージ

タイプ	RLD 名	使用法
ML	R_TLSML	スレッド・ローカル・ストレージ
TR	[R_TRL]	TOC 参照
TC	R_TOC (R)	TOC 参照
p	R_POS	一般

大規模 TOC 再配置タイプは、XMC_TE ストレージ・マッピング・クラスではデフォルトで使用されますが、XMC_TC シンボルで TOC 相対命令を使用する場合にも使用できます。

スレッド・ローカル・ストレージ再配置タイプは、通常、スレッド・ローカル・シンボルへの TOC 参照で使用されます。 デフォルトでは、 **@gd** 再配置指定子が使用されます。 他の TLS アクセス方式を使用する場合は、明示的な指定子が必要です。

@tr 再配置指定子を使用すると、R_TRL 再配置タイプを TOC 相対ロードで使用することができます。 この再配置タイプは、リンカーがロード命令を即時追加命令に変換しないようにします。

@tc および **@p** 再配置指定子は必要ありませんが、完全を期すために提供されています。

定数

アセンブラー言語には、4 種類の定数が用意されています。

アセンブラーが命令のオペランドとして使用されている算術定数または文字定数を検出すると、その定数の値が命令にアセンブルされます。 アセンブラーが定数として使用されているシンボルを検出すると、そのシンボルの値が命令にアセンブルされます。

算術定数

アセンブラー言語は、アセンブリー・モードに依存する算術定数を提供します。

アセンブラー言語は、以下の 4 種類の算術定数を提供します。

- 10 進数
- 8 進数
- 16 進数
- 浮動小数点

32 ビット・モードでは、表現できる最大の符号付き正整数は、10 進値 (2 * *31)-1 です。 負の最大値は-(2 * *31) です。 64 ビット・モードでは、表現できる最大の符号付き正整数は (2 * *63) -1 です。 負の最大値は-(2 * *63) です。 基数 (例えば、10 進数、16 進数、または 8 進数) に関係なく、アセンブラーは整数を 32 ビット定数と見なします。

定数の解釈は、アセンブリー・モードによって異なります。 32 ビット・モードでは、 AIX® アセンブラーは以前の AIX® バージョンと同じように動作します。アセンブラーは整数を 32 ビット定数と見なします。 64 ビット・モードでは、すべての定数は 64 ビット値として解釈されます。 これにより、期待とは異なる結果が生じる可能性があります。 例えば、32 ビット・モードでは、16 進値 0xFFFFFFFF は 10 進値「-1」に相当します。 ただし、64 ビット・モードでは、同等の 10 進数は 4294967295 です。 値「-1」を取得するには、16 進定数 0xFFFF_FFFF_FFFF_FFFF (または同等の 8 進数)、または 10 進値 -1 を使用する必要があります。

32 ビット・モードと 64 ビット・モードの両方で、ターゲット・ストレージ域のサイズが小さすぎて式の結果を入れることができない場合は、整数式の結果が切り捨てられることがあります。 (このコンテキストでは、切り捨てとは、余分な最上位ビットを除去することを意味します。)

大きな定数、特に 64 ビット値の読みやすさを向上させるために、アセンブラーは下線 ("_") を含む定数を受け入れます。 文字。 下線は、最初の数値位置を除き、数値内の任意の場所に使用できます。 例えば、以下の表について考えてみます。

定数値	有効/無効
1 から 800 _500	有効かどうか
0xFFFFFFFF_00000000	有効かどうか
0b111010_00100_00101_00000000001000_00	有効 (これは "ld 4,8 (5)" 命令です)
0x_FFFF	無効

3 番目の例は、命令内のさまざまなフィールドを説明するために下線文字が使用される命令のバイナリー表現を示しています。 最後の例には 16 進接頭部が含まれていますが、直後の文字が有効な数字ではありません。そのため、定数は無効です。

算術評価
算術計算では、32 ビット・モードと 64 ビット・モードが使用されます。

32 ビット・モードでは、算術計算は 32 ビットの数学を使用して行われます。 64 ビット数量を指定するために使用される **.llong** 疑似命令の場合、ストレージ域の値を初期化するために必要な評価では、32 ビットの算術演算が使用されます。

64 ビット・モードの場合、算術計算は 64 ビットの数学を使用します。 32 ビット・コンテキストで数値が負と見なされる場合でも、符号拡張子は発生しません。 負の数値は、10 進形式を使用して指定するか、または 16 進数字の完全な補数 (16 桁) を使用して指定する必要があります (16 進形式など)。

10 進定数
基数 10 は、算術定数のデフォルトの基数です。

基数 10 は、算術定数のデフォルトの基数です。 10 進数を指定する場合は、適切な場所に数値を入力します。

```
ai 5,4,10
# Add the decimal value 10 to the contents
# of GPR 4 and put the result in GPR 5.
```

10 進数の前に 0 を付けないでください。 先行ゼロは、数値が 8 進数であることを示します。

8 進定数
数値が 8 進数であることを指定するには、数値の前に `0` を付けます。

数値が 8 進数であることを指定するには、数値の前に `0` を付けます。

```
ai 5,4,0377
# Add the octal value 0377 to the contents
# of GPR 4 and put the result in GPR 5.
```

16 進定数
16 進数を指定するには、数値の前に `0X` または `0x` を付けます。

16 進数を指定するには、数値の前に `0X` または `0x` を付けます。 A から F までの 16 進数には、大文字でも小文字でも使用できます。

```
ai 5,4,0xF
# Add the hexadecimal value 0xF to the
# contents of GPR 4 and put the result
# in GPR 5.
```

2 進定数
2 進数を指定するには、数値の前に `0B` または `0b` を付けます。

2 進数を指定するには、数値の前に 0B または 0b を付けます。

```
ori 3,6,0b0010_0001
# OR (the decimal value) 33 with the
# contents of GPR 6 and put the result
# in GPR 3.
```

浮動小数点定数

浮動小数点定数には、指定された順序で異なるコンポーネントがあります。

浮動小数点定数には、以下のコンポーネントが指定された順序で含まれます。

項目	説明
整数部分	1 つ以上の数字でなければなりません。
小数点	. (ピリオド). 小数部分が後に続かない場合はオプション。
小数部 (Fraction Part)	1 つ以上の数字でなければなりません。 小数部はオプションです。
指数部 (Exponent Part)	オプション。 e または E の後に + または -が続き、その後に 1 つ以上の数字が続く場合があります。

アセンブラー入力の場合は、小数部を省略できます。 例えば、以下は有効な浮動小数点定数です。

- 0.45
- 1e+5
- 4E-11
- 0.99E6
- 357.22e12

浮動小数点定数は、 **fcon** 式がある場合にのみ許可されます。

オペランドの境界検査は行われません。

> **注:** AIX® 4.3 以降では、アセンブラーは **stri5/OS** サブルーチンを使用して浮動小数点への変換を行います。 制約事項および戻り値については、現行の資料を確認してください。

文字定数

文字定数は、特定の文字のコードを使用する場合に使用します。

ASCII 文字定数を指定するには、定数の前に ' (単一引用符) を付けます。 文字定数は、算術定数が使用できる場所であればどこにでも指定できますが、一度に 1 つの文字定数しか指定できません。 例えば、'A は文字 A の ASCII コードを表します。

文字定数は、特定の文字のコードを定数として使用する場合に便利です。以下に例を示します。

```
cal 3,'X(0)
# Loads GPR 3 with the ASCII code for
# the character X (that is, 0x58).
# After the cal instruction executes, GPR 3 will
# contain binary
# 0x0000 0000 0000 0000 0000 0000 0101 1000.
```

シンボリック定数

シンボリック定数は、ラベルとして使用するか、 **.set** ステートメントとして使用することによって定義できます。

シンボルに値を与えることによって、シンボルを定数として使用することができます。 その後、値自体を使用する代わりに、シンボル名を使用して値を参照することができます。

プログラム内で値が頻繁に発生する場合は、シンボルを定数として使用すると便利です。 値に名前を付けることによって、シンボリック定数を 1 回定義します。 その値を変更するには、単にプログラム内の定義を変更します (参照するたびに変更するのではありません)。 新しいシンボル定数が有効になる前に、変更されたファイルを再アセンブルする必要があります。

シンボリック定数は、ラベルとして使用するか、 **.set** ステートメントで使用することによって定義できます。

ストリング定数

ストリング定数は、特定の疑似命令 (**.rename**、.**byte**、 **.string** 疑似命令など) に対するオペランドとして使用できます。

ストリング定数は、 **.rename**、 **.byte**、または **.string** 疑似命令など、特定の疑似命令に対するオペランドとしてのみ使用できるという点で、他のタイプの定数とは異なります。

ストリング定数の構文は、"" (二重引用符) で囲まれた任意の数の文字で構成されます。

```
"any number of characters"
```

" を使用するには、 ストリング定数では、二重引用符を 2 回使用します。 次に例を示します。

```
"a double quote character is specified like this "" "
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

関連情報

atof

演算子

アセンブラーは、3 つのタイプの単項演算子を提供します。

すべての演算子は、右から左に評価される単項演算子を除き、左から右に評価されます。

アセンブラーは、以下の単項演算子を提供します。

項目	説明
+	単項正
-	単項負
~	1 の補数 (単項)

アセンブラーは、以下の 2 項演算子を提供します。

項目	説明
*****	乗算
/	除算
>	右シフト
<	左シフト
|	ビット単位包含 OR

項目	説明
&	ビット単位 AND
^	ビット単位排他 OR 演算子
+	追加
-	減算

式の中で括弧を使用して、アセンブラーが式を評価する順序を変更することができます。 括弧内の演算は、括弧外の演算の前に実行されます。 ネストされた括弧が含まれている場合、処理は最も内側の括弧のセットから始まり、外側に進みます。

演算子優先順位 (operator precedence)

演算子優先順位には 32 ビットの式があります。

32 ビット式の演算子優先順位を以下の図に示します。

最高優先順位

```
| ( )
     | unary -, unary +, ~
     | * / < >
     | | ^ &
     | + _
     V
```

最低優先順位

32 ビット・モードでは、すべてのオペレーターが 32 ビット符号付き整数演算を実行します。 64 ビット・モードでは、すべての計算は 64 ビット符号付き整数演算を使用して実行されます。

除算演算子は整数の結果を生成します。剰余は被除数と同じ符号になります。 次に例を示します。

操作	結果	剰余演算子
8/3	2	2
8/-3	-2	2
(-8)/3	-2	-2
(-8)/(-3)	2	-2

左シフト (<) 右シフト (>) 演算子は、右側のオペランドに整数ビット値を取ります。 次に例を示します。

```
.set mydata,1
.set newdata,mydata<2
# Shifts 1 left 2 bits.
# Assigns the result to newdata.
```

式

式は、1 つ以上の項によって形成されます。

用語 は、式の処理時にアセンブラー・パーサーが認識できる最小エレメントです。 各用語には値とタイプがあります。 式は、1 つ以上の項によって形成されます。 アセンブラーは、それぞれの式を単一値に評価し、その値をオペランドとして使用します。 それぞれの式にもタイプがあります。 式が 1 つの項によって形成される場合、式は項のタイプと同じタイプになります。 式が複数の項で構成されている場合、式に含まれるすべての項のタイプに適用される特定の規則に従って、式ハンドラーによってタイプが決定されます。 以下の理由により、式タイプは重要です。

- 一部の疑似命令および命令は、特定のタイプの式を必要とします。
- 特定のタイプの式では、特定の演算子のみが許可されます。

オブジェクト・モードに関する考慮事項

アセンブリー言語式の 1 つの側面として、計算されるデータ値のサイズに対するオブジェクト・モードと再配置の式があります。

アセンブリー言語式の 1 つの側面として、計算されるデータ値のサイズに対するオブジェクト・モードと再配置の式があります。 32 ビット・モードでは、再配置は 32 ビット数量に適用されます。結果として再配置が必要になる式 (例えば、外部シンボルへの参照) では、ワード以外のストレージ域に値を保管することはできません。 **.llong** 疑似命令の場合、.llong の内容を初期化するために使用される式が再配置を必要としないことを指摘する価値があります。 64 ビット・モードでは、ダブルワードの数量に再配置が適用されます。 したがって、再配置を必要とする式の結果は、ダブルワードより小さい位置に値を保管することはできません。

32 ビット・モードでの式の算術評価は、アセンブラーの前のリリースで検出された動作と整合しています。 整数定数は 32 ビットの数量と見なされ、計算は 32 ビットの計算です。 64 ビット・モードでは、定数は 64 ビット値であり、式は 64 ビット計算を使用して評価されます。

用語のタイプと値

使用されている用語のリストと用語の値。

以下は、すべてのタイプの用語と、各タイプの省略名のリストです。

絶対項

プログラムの再配置時に値が変わらない場合、用語は絶対値です。

プログラムの再配置時に値が変わらない場合、用語は絶対値です。 言い換えると、用語の値がコード再配置操作から独立している場合、その用語は絶対値です。

絶対項は、以下の項目のいずれかです。

- 定数 (33 ページの『定数』 で定義されているすべての種類の定数を含む)。
- 絶対式に設定されたシンボル。

絶対項の値は、定数値です。

再配置可能用語

プログラムの再配置時に値が変更された場合、用語は再配置可能です。

プログラムの再配置時に値が変更された場合、用語は再配置可能です。 再配置可能な用語の値は、それを含む制御セクションの場所によって異なります。 制御セクションが別の保管場所に移動した場合 (例えば、csect がバインド時にバインダーによって再配置された場合)、再配置可能項の値はそれに応じて変更されます。

再配置可能用語は、以下の項目のいずれかです。

- TD または TC をストレージ・マッピング・クラス (SMC) として持たない csect 内で定義されたラベル
- 再配置可能式に設定されたシンボル
- dsect 内で定義されたラベル
- Dsect 名
- ロケーション・カウンター参照 (**$、** ドル記号を使用)

D 形式の命令の変位として使用されない場合、csect ラベルまたはロケーション・カウンター参照の値は、その再配置可能アドレスになります。これは、収容 csect アドレスと収容 csect に対するオフセットの合計です。 D 形式命令の変位として使用される場合、アセンブラーは、オフセットのみが変位に使用されるように、含まれている csect アドレスを暗黙的に減算します。 csect アドレスは、ファイルの最初の csect の先頭を基準としたオフセットです。

dsect は、ストレージを実際に予約することなく、ストレージ域内のデータのレイアウトを記述できる参照制御セクションです。 dsect は、データが空のシンボリック・フォーマットを提供します。 アセンブラーは、dsect で定義されているラベルにロケーション・カウンター値を割り当てます。 値は、dsect の先頭か

らの相対的なオフセットです。 実行時の dsect 内のデータは、dsect で定義されたラベルを使用して、シンボルで参照することができます。

dsect ロケーション・カウンター (dsect 名または dsect ラベルのいずれか) に基づく再配置可能用語は、 **.using** ステートメントのコンテキストでのみ意味があります。 これは基底アドレスを dsect に関連付ける唯一の方法であるため、dsect のアドレス可能度はストレージ域と組み合わせて設定されます。

再配置可能項は、再配置可能アドレス定数として使用される場合を除き、すべてのコンテキストにおいて csect または dsect のいずれかの制御セクションに基づくことができます。 csect ラベルがアドレス定数として使用される場合、csect ラベルは再配置可能アドレスを表し、その値は csect に対するオフセットに csect のアドレスを加えたものです。 dsect はデータ・テンプレートにすぎず、アドレスを持たないため、dsect ラベルを再配置可能アドレス定数として使用することはできません。

2 つの dsect ラベルが同じ dsect に定義されている場合、それらの違いは絶対アドレス定数として使用できます。

外部再配置可能用語

用語が外部シンボル (シンボルは定義されていないが、現行ファイル内で宣言されているか、現行ファイルおよびグローバルで定義されている)、csect 名、または TOC 項目名の場合、その用語は外部再配置可能 (E_EXT) です。

この用語は、値が変更されるため、再配置可能です。再配置されるのは、その値、またはそれを含む制御セクションが再配置された場合です。

外部再配置可能項または式は、 **.set** 疑似命令のオペランドとして使用することはできません。

外部再配置可能用語は、以下の項目のいずれかです。

- **.comm** 疑似命令で定義されたシンボル
- **.lcomm** 疑似命令で定義されたシンボル
- Csect 名
- **.globl** 疑似命令で宣言されたシンボル
- TOC エントリー名
- **.extern** 疑似命令を使用して宣言された未定義シンボル

未定義シンボルを除き、この用語が D-形式命令の変位として使用されない場合、その値は再配置可能アドレスです。これは、ファイル内の最初の csect の先頭からの相対オフセットです。 D 形式命令の変位として使用される場合、アセンブラーは、含まれている csect アドレス (TOC エントリー名を除く) を暗黙的に減算します。通常、csect アドレスはそれ自体から減算されるため、変位はゼロになります。 TOC エントリー名が D 形式命令の変位として使用される場合、アセンブラーは TOC アンカーのアドレスを暗黙的に減算するため、TOC アンカーに対するオフセットは変位になります。

未定義のシンボルは、D 形式命令の変位として使用することはできません。 それ以外の場合は、その値はゼロです。

TOC 相対再配置可能用語

TOC 内に含まれるラベルの場合、用語は TOC 相対再配置可能 (E_TREL) です。

TOC 内に含まれるラベルの場合、用語は TOC 相対再配置可能 (E_TREL) です。

TOC が再配置されると値が変更されるため、このタイプの用語は再配置可能です。

TOC 相対再配置可能用語は、以下のいずれかの項目です。

- **.tc** 疑似命令上のラベル。
- ストレージ・マッピング・クラスとして TD または TC を持つ csect 内で定義されたラベル。

この用語が D 形式命令の変位として使用されない場合、その値は再配置可能アドレスになります。これは TOC および TOC アンカー・アドレスに対するオフセットの合計です。 D 形式命令の変位として使用される場合、アセンブラーは TOC アンカー・アドレスを暗黙的に減算するため、TOC アンカーに対するオフセットは変位になります。

TOCOF 再配置可能用語
用語が **.tocof** 疑似命令の第 1 オペランドである場合、その用語は TOCOF 再配置可能 (E_TOC0F) タイプです。

用語が **.tocof** 疑似命令の第 1 オペランドである場合、その用語は TOCOF 再配置可能 (E_TOC0F) タイプです。

このタイプの項の値はゼロです。 D 形式命令の変位として使用することはできません。 算術演算に関与することはできません。

式のタイプと値
用語のタイプとその値式。

式は、項が持つことのできるすべてのタイプを持つことができます。 1 つの項のみを持つ式は、その項と同じ型を持ちます。 式は、制限付き外部再配置可能 (E_REXT) タイプを持つこともできますが、このタイプには少なくとも 2 つの項が必要であるため、このタイプを持つことはできません。

制限付き外部再配置可能式
式に 2 つの再配置可能項が含まれている場合、式には制限付き外部再配置可能 (E_REXT) タイプがあります。

式に制限付き外部再配置可能 (E_REXT) タイプがあるのは、異なる制御セクションで定義された 2 つの再配置可能用語 (41 ページの『結合式の式タイプ』で定義されている 対の再配置可能用語の要件を満たさない用語) が含まれており、その逆の符号がある場合です。

制限付き外部再配置可能式では、スレッド・ローカル・シンボルと非スレッド・ローカル・シンボルの混合は許可されません。 つまり、一方のシンボルがスレッド・ローカル・ストレージ・マッピング・クラス (TL または UL) を持っている場合、他方のシンボルもスレッド・ローカル・ストレージ・マッピング・クラスを持っている必要があります。

以下は、制限された外部再配置可能タイプの式を生成する再配置可能項の組み合わせの例です。

- <E_EXT> - <E_EXT>
- <E_REL> - <E_REL>
- <E_TREL> - <E_TREL>
- <E_EXT> - <E_REL>
- <E_REL> - <E_EXT>
- <E_EXT> - <E_TREL>
- <E_TREL> - <E_REL>

  このタイプの式に割り当てられる値は、その項の値のアセンブラー算術計算の結果に基づいています。 算術演算に関与する場合、項の値はその再配置可能アドレスです。

式の組み合わせ処理
式の中の項は、2 項演算子と組み合わせることができます。

式の中の項は、2 項演算子と組み合わせることができます。 また、1 つの用語は 1 つ以上の単項演算子で始めることができます。 アセンブラー式ハンドラーは、結果の式タイプ、値、および再配置テーブル・エントリーを評価して決定します。

式の値の計算
値の計算時にルールが適用されます。

値を計算する際には、以下の規則が適用されます。

- 算術演算に関与している場合、絶対項の値はその定数値になり、再配置可能項の値 (E_EXT、 E_REL、または E_TREL) はその再配置可能アドレスになります。

- 結果の式が D 形式命令の変位として使用される場合、アセンブラーは、 E_EXT または E_REL 型の式の最終結果から包含する csect アドレスを暗黙的に減算するか、または E_TREL 型の式の TOC アンカー・アドレスを減算します。 E_ABS タイプまたは E_REXT タイプの式の場合、暗黙的な減算は行われません。

式のオブジェクト・ファイル再配置テーブル項目

アセンブラーは、式のオブジェクト・ファイル再配置テーブル・エントリーの要件を決定する際に規則を適用します。

アセンブラーは、式のオブジェクト・ファイル再配置テーブル・エントリーの要件を決定する際に、以下の規則を適用します。

- 式がデータ定義、TOC エントリー定義、またはブランチ・ターゲット・アドレスで使用される場合、式の結果のタイプに応じて、0 から 2 つの再配置テーブル・エントリー (RLD) が必要になる場合があります。
  - E_ABS は、再配置エントリーを必要としません。
  - E_REL は、dsect 名または dsect ラベルが再配置エントリーを必要としないことを除き、1 つの再配置エントリーを必要とします。
  - E_EXT には 1 つの再配置エントリーが必要です
  - E_REXT には 2 つの再配置エントリーが必要です
  - E_TREL には 1 つの再配置エントリーが必要です
  - E_TOCOF には 1 つの再配置エントリーが必要です
- 式が D 形式命令オペランド内の変位として使用される場合、 E_TREL 式と E_REXT 式のみが再配置項目を持ちます。 それぞれが 1 つの再配置エントリーを必要とします。

結合式の式タイプ

アセンブラーは、結合式のタイプを決定する際に規則を適用します。

アセンブラーは、結合式のタイプを決定する際に、以下の規則を適用します。

グループ *1* 演算子を使用した式の結合

アセンブラーは、グループ 1 演算子を使用して式を結合するという規則を適用します。

以下のオペレーターは、グループ #1 に属します。

- ***, /, >, <, |, &, ^**

グループ #1 の演算子には、以下の規則があります。

- <E_ABS> <op1> <E_ABS> ==> E_ABS
- 絶対式以外のすべてのタイプの式にグループ #1 の演算子を適用すると、エラーが発生します。

グループ *2* 演算子を使用した式の結合

アセンブラーは、グループ 2 演算子を使用して式を結合するという規則を適用します。

以下のオペレーターは、グループ # 2 に属します。

- **+, -**

グループ # 2 の演算子には、以下の規則があります。

- <E_ABS> <op2> <E_ABS> ==> E_ABS
- <E_ABS> <op2> <E_REXT> ==> E_REXT
- <E_REXT> <op2> <E_ABS> ==> E_REXT
- <E_ABS> <op2> <E_TOCOF> ==> エラー
- <E_TOCOF> <op2> <E_ABS> ==> エラー
- < non E_ABS> <op2> <E_REXT> ==> エラー
- <E_REXT> <op2> < 非 E_ABS> ==> エラー

- <E_ABS>-<E_TREL> ==> エラー
- 単項 **+** および **-** は、左側の項として絶対値 0 (ゼロ) を持つ 2 項演算子と同じように扱われます。
- 一方の項が絶対式ではないその他の状況では、より複雑な規則が必要になります。

以下の定義は、後の説明で使用します。

項目	説明
対になった再配置可能な用語	反対の符号があり、同じセクションに定義されている。 対になった再配置可能項によって表される値は絶対値です。 再配置可能用語のペアの結果タイプは E_ABS です。 ペアになった再配置可能項は、式の中で連続している必要はありません。 同じセクションに定義されていない 2 つの再配置可能用語をペアにすることはできません。 E_TREL 用語は、別の E_TREL 用語または E_EXT 用語とペアにすることはできますが、 E_REL 用語とペアにすることはできません (同じセクションに存在することはないため)。 E_EXT 用語または E_REL 用語は、別の E_EXT 用語または E_REL 用語とペアにすることができます。 E_REXT 用語をどの用語ともペアにすることはできません。
対向項	反対の符号を持ち、同じシンボル・テーブル項目を指す。 どの用語にも、反対の用語を付けることができます。 反対の項によって表される値はゼロです。 反対の項の結果タイプは、データ定義に使用されるときに R_REF タイプの再配置テーブル・エントリー (RLD) が生成されることを除き、 E_ABS とほぼ同じです。 反対の項は、式の中で連続している必要はありません。

対向項と対になっている再配置可能項の主な相違点は、対になっている再配置可能項が同じセクションに定義されていなければならないが、同じテーブル項目を指す必要がないことです。

以下の例では、 L1 と -L1 は逆の用語であり、 L1 と -L2 は再配置可能な用語のペアです。

```
     .file "f1.s"
     .csect Dummy[PR]
L1:  ai 10, 20, 30
L2:  ai 11, 21, 30
     br
     .csect A[RW]
     .long L1 - L1
     .long L1 - L2
```

以下の表は、複合結合式のタイプを決定するための規則を示しています。

項目	**説明**	
Type	**タイプを持つ式の条件**	**再配置テーブル・エントリー**
E_ABS	式のすべての項は、再配置可能な項、反対の項、および絶対項と対になっています。	タイプ R_REF の RLD は、対向する用語ごとに生成されます。
E_REXT	この式には、2 つの対になっていない再配置可能項が含まれています。これらの項には、対になっているすべての再配置可能項、反対の項、および絶対項に加えて、反対の符号が付いています。	2 つの RLD (1 つは R_POS タイプ、もう 1 つは R_NEG タイプ) が、対になっていない再配置可能用語に対して生成されます。 さらに、タイプが R_REF の RLD が、対向する用語ごとに生成されます。 一方の用語にスレッド・ローカル・ストレージ・マッピング・クラスがあり、もう一方の用語にスレッド・ローカル・ストレージ・マッピング・クラスがない場合は、エラーが報告されます。

項目	説明	
E_REL, E_EXT	式には、対になっていない E_REL または E_RXT 項と、対になっているすべての再配置可能な項、反対の項、および絶対項が 1 つだけ含まれています。	式がデータ定義で使用される場合、タイプが R_POS または R_NEG の RLD が 1 つ生成されます。 さらに、タイプ R_REF の RLD が、反対の用語ごとに生成されます。
E_TREL	式には、対になっているすべての再配置可能項、反対の項、および絶対項に加えて、1 つの対になっていない E_TREL 項のみが含まれています。	式が D 形式命令の変位として使用される場合は、タイプ R_TOC の RLD が 1 つ生成されます。それ以外の場合は、タイプ R_POS または R_NEG の RLD が 1 つ生成されます。 さらに、タイプ R_REF の RLD が、反対の用語ごとに生成されます。
エラー	式に 3 つ以上の対になっていない再配置可能項が含まれている場合、または同じ符号を持つ 2 つの対になっていない再配置可能項が含まれている場合は、エラーが報告されます。	

以下の例は、上記の表を示しています。

```
      .file "f1.s"
      .csect A[PR]
L1:   ai 10, 20, 30
L2:   ai 10, 20, 30
EL1: l 10, 20(20)
     .extern EL1
     .extern EL2
EL2: l 10, 20(20)
      .csect B[PR]
BL1: l 10, 20(20)
BL2: l 10, 20(20)
       ba 16 + EL2 - L2 + L1              # Result is E_REL
       l 10, 16+EL2-L2+L1(20)             # No RLD
      .csect C[RW]
BL3: .long BL2 - B[PR]                   # Result is E_ABS
      .long BL2 - (L1 - L1)              # Result is E_REL
      .long 14-(-EL2+BL1) + BL1 - (L2-L1) # Result is E_REL
      .long 14 + EL2 - BL1 - L2 + L1     # Result is E_REL
      .long (B[PR] -A[PR]) + 32          # Result is E_REXT
```

アドレッシング

アセンブリー言語は、さまざまなアドレッシング・モードとアドレッシングに関する考慮事項を使用します。

アドレッシングに関する記事では、以下のようなアドレッシング・モードとアドレッシングに関する考慮事項について説明します。

絶対アドレッシング

絶対アドレスは、レジスターの内容によって表されます。

絶対アドレスは、レジスターの内容によって表されます。 このアドレッシング・モードは、現行命令アドレスに対して指定されていないという意味では絶対モードです。

「Branch Conditional to Link Register」命令と「Branch Conditional to Count Register」命令の両方で、絶対アドレッシング・モードを使用します。 ターゲット・アドレスは、入力オペランドではなく、特定のレジスターです。 ターゲット・レジスターは、分岐条件付きリンク・レジスター命令のリンク・レジスター (LR) です。 ターゲット・レジスターは、「Branch Conditional to Count Register」命令のカウント・レジスター (CR) です。 これらのレジスターは、レジスター命令に対する分岐条件の実行前にロードする必要があります。

絶対即値アドレッシング

絶対即値アドレスは、即時データによって指定されます。

絶対即値アドレスは、即時データによって指定されます。 このアドレッシング・モードは、現行命令アドレスに対して指定されていないという意味では絶対モードです。

分岐条件付き命令および分岐条件付き命令では、絶対アドレス・ビット (AA ビット) がオンの場合、絶対即値アドレッシング・モードが使用されます。

即時データのオペランドは、絶対式、再配置可能式、または外部式にすることができます。

相対即値アドレッシング

相対即値アドレスは、オブジェクト・コード内で即時日付として指定され、現在の命令位置に対して相対的に計算されます。

相対即値アドレスは、オブジェクト・コード内の即時データとして指定され、現在の命令位置に対して相対的に計算されます。 相対即値アドレッシングを使用する命令は、すべてブランチ命令です。 これらの命令には、現在の命令位置からのフルワードでの変位である即時データがあります。 実行時に、即時データは符号拡張され、論理的に左 2 ビットにシフトされ、分岐ターゲット・アドレスを計算するために分岐命令のアドレスに追加されます。 即時データは再配置可能式または外部式でなければなりません。

明示的ベースのアドレッシング

明示ベースのアドレスは、基底レジスター番号 *RA* および変位 *D* として指定されます。

明示ベースのアドレスは、基底レジスター番号 *RA* および変位 *D* として指定されます。 基底レジスターは基底アドレスを保持します。 実行時に、プロセッサーは、有効アドレスを取得するために基底レジスターの内容に変位を追加します。 命令に *D*(*RA*) のオペランド形式がない場合、その命令は明示ベースのアドレスを持つことができません。 これらの命令に *D*(*RA*) フォームを使用すると、エラー 159 が報告されます。

変位は、絶対式、再配置可能式、制限付き外部式、または TOC 相対式にすることができます。 変位が外部式になることができるのは、それが csect (制御セクション) 名であるか、または **.comm** 疑似命令によって定義された共通ブロックの名前である場合だけです。

注 :

1. 外部化されたラベルは再配置可能であるため、外部化されたラベルを変位として使用することもできます。
2. 変位に再配置可能式が使用される場合、csect のラベルからのオフセット (つまり再配置可能式) のみが変位に使用されるため、RLD 項目は生成されません。

プログラマーは、絶対式を使用して基底レジスター自体を指定する必要がありますが、基底レジスターの内容は、絶対式、再配置可能式、または外部式によって指定できます。 基底レジスターに再配置可能値が入っている場合、有効アドレスは再配置可能です。 基底レジスターに絶対値が入っている場合、有効アドレスは絶対アドレスです。 基底レジスターが外部式によって指定された値を保持する場合、有効アドレスのタイプは、式が最終的に絶対として定義される場合は絶対アドレスになり、式が最終的に再配置可能として定義される場合は再配置可能アドレスになります。

明示的ベースのアドレッシングを使用する場合は、以下の点に注意してください。

- GPR 0 を基底レジスターとして使用することはできません。 0 を指定すると、アセンブラーは基底レジスターをまったく使用しません。
- *D* は最大 16 ビットを占有するため、変位は -2 * *15 から (2 * *15) -1 の範囲でなければなりません。 したがって、基底アドレスと、参照される項目のアドレスとの差は、2 * *15 バイト未満でなければなりません。

  注 : *D* および *RA* は、 *D*(*RA*) フォームに必要です。 形式 0 (*RA*) または *D*(0) を使用できますが、 *D* オペランドと *RA* オペランドの両方が必要です。 ただし、次の 2 つの例外があります。

  – *D* が絶対式の場合、

  – *D* が制限付き外部式である場合。

これらの 2 つのケースで *RA* オペランドが欠落している場合は、 *D*(0) が想定されます。

暗黙ベースのアドレッシング

暗黙ベースのアドレスは、RA オペランドを省略し、命令の前のある時点で **.using** 疑似命令を作成することによって、命令のオペランドとして指定されます。

暗黙ベースのアドレスは、 *RA* オペランドを省略し、命令の前のある時点で **.using** 疑似命令を作成することによって、命令のオペランドとして指定されます。 適切な **.using** および **.drop** 疑似命令をアセンブルした後、アセンブラーは基底レジスターとして使用するレジスターを判別できます。 実行時に、プロセッサーは、命令に基底が明示的に指定されているかのように、有効アドレスを計算します。

暗黙ベースのアドレスは、実行時に *RA* オペランドの内容を指定するために使用される式のタイプに応じて、再配置可能または絶対にすることができます。 通常、 *RA* オペランドの内容は再配置可能式で指定されるため、再配置可能暗黙ベース・アドレスになります。 この場合、アセンブラーによって作成されたオブジェクト・モジュールが再配置されると、基底レジスター *RA* の内容のみが変更されます。 変位は同じままであるため、 *D*(*RA*) は再配置後も正しいアドレスを指します。

dsect は、ストレージを実際に予約することなく、ストレージ域内のデータのレイアウトを記述できる参照制御セクションです。 暗黙ベースのアドレスは、*RA* の内容を dsect 名または dsect ラベルで指定して、ベースをダミー・セクションに関連付けることによって作成することもできます。 *RA* コンテンツの値は、dsect のインスタンス化時に実行時に解決されます。

RA オペランドの内容が絶対式で指定されている場合は、絶対暗黙ベース・アドレスが作成されます。 この場合、オブジェクト・モジュールが再配置されても、 *RA* の内容は変更されません。

アセンブラーは、再配置可能な暗黙ベースのアドレッシングのみをサポートします。

暗黙ベースのアドレッシングを使用する場合は、以下を実行します。

1. **.using** ステートメントを作成して、1 つ以上の汎用レジスター (GPR) が基底レジスターとして使用されるようになったことをアセンブラーに通知します。
2. この **.using** ステートメントでは、実行時に各基底レジスターに含まれる値をアセンブラーに指示します。 **.drop** 疑似命令が検出されるまで、アセンブラーはこの基底付きレジスター値を使用して、基底付きアドレスを必要とするすべての命令を処理します。
3. 各基底レジスターに、前に指定した値をロードします。

暗黙ベースのアドレッシングの場合、*RA* オペランドは常に省略されますが、*D* オペランドは残ります。 *D* オペランドは、絶対式、TOC 相対式、再配置可能式、または制限付き外部式にすることができます。

注:

1. *D* オペランドが絶対式または制限付き外部式である場合、アセンブラーは常にそれを *D*(0) 形式に変換するため、 **.using** 疑似命令は効果がありません。
2. **.using** および **.drop** 疑似命令は、基底付きアドレスにのみ影響します。

```
.toc
T.data: .tc data[tc],data[rw]
.csect data[rw]
        foo: .long 2,3,4,5,6
        bar: .long 777

        .csect text[pr]
        .align 2
        l 10,T.data(2)   # Loads the address of
                                # csect data[rw] into GPR 10.

        .using data[rw], 10      # Specify displacement.
        l 3,foo          # The assembler generates l 3,0(10)
        l 4,foo+4        # The assembler generates l 4,4(10)
        l 5,bar          # The assembler generates l 5,20(10)
```

ロケーション・カウンター (location counter)

アセンブラー言語プログラムには、プログラムのステートメントにストレージ・アドレスを割り当てるために使用されるロケーション・カウンターがあります。

アセンブラー言語プログラムの各セクションには、プログラムのステートメントにストレージ・アドレスを割り当てるために使用されるロケーション・カウンターがあります。 ソース・モジュールの命令がアセンブルされるとき、ロケーション・カウンターはストレージ内の現在の位置を追跡します。 **$** (ドル記号) を命令のオペランドとして使用して、ロケーション・カウンターの現行値を参照することができます。

プログラムのアセンブルとリンク

アセンブリー言語プログラムは、プログラムをアセンブルおよびリンクするためのコマンドを定義します。

このセクションでは、以下に関する情報を提供します。

プログラムのアセンブルとリンク

アセンブリー言語プログラムは、 **as** コマンドまたは **cc** コマンドを使用してアセンブルできます。

アセンブリー言語プログラムは、 **as** コマンドまたは **cc** コマンドを使用してアセンブルできます。 **ld** コマンドまたは **cc** コマンドを使用して、アセンブルされたプログラムをリンクすることができます。 このセクションでは、以下について説明します。

as コマンドを使用したアセンブル

as コマンドはアセンブラーを呼び出します。

as コマンドは、アセンブラーを呼び出します。 **as** コマンドの構文は次のとおりです。

```
as [ -a Option[:Option] ] [ -oObjectFile ] [ -n Name ] [ -u ] [ -l[ListFile] ]
    [ -W | -w ] [ -x[XCrossFile] ] [ -s [ListFile] ] [ -m ModeName ] [ -M ]
    [ -Eoff|on ] [ -poff|on ] [-i] [-v] [ File ]
```

as コマンドは、 *File* パラメーターで指定されたファイルを読み取ってアセンブルします。 規則により、このファイルの接尾部は **.s** になります。 ファイルが指定されていない場合、 **as** コマンドは標準入力を読み取ってアセンブルします。 デフォルトでは、**as** コマンドはその出力を **a.out** という名前のファイルに保管します。 出力は XCOFF ファイル・フォーマットで保管されます。

as コマンドのフラグはすべてオプションです。

ld コマンドは、オブジェクト・ファイルをリンクするために使用されます。 詳しくは、 **ld** コマンドを参照してください。

アセンブラーは、OBJECT_MODE 環境変数の設定に従います。 **-a32** または **-a64** のどちらも使用されていない場合は、この変数に照らして環境が検査されます。 この変数の値が、次の表に示すどの値にも該当しない場合は、エラー・メッセージが生成され、アセンブラーはゼロ以外の戻りコードを戻して終了します。 有効な設定のそれぞれに対応する暗黙の動作は次のとおりです。

項目	説明
OBJECT_MODE=**32**	32 ビット・オブジェクト・コードを生成します。 デフォルトのマシン設定は **com** です。
OBJECT_MODE=**64**	64 ビット・オブジェクト・コード (XCOFF64 ファイル) を生成します。 デフォルトのマシン設定は **ppc64** です。
オブジェクト・モデル **=32_64**	無効。
OBJECT_MODE=その他すべて	無効。

関連情報

as

cc コマンドを使用したアセンブルおよびリンク

cc コマンドを使用して、アセンブリー・ソース・プログラムをアセンブルし、リンクすることができます。

cc コマンドを使用して、アセンブリー・ソース・プログラムをアセンブルし、リンクすることができます。以下の例では、 **cc** コマンドでコンパイルまたはアセンブルされたオブジェクト・ファイルをリンクします。

```
cc pgm.o subs1.o subs2.o
```

cc コマンドを使用してオブジェクト・ファイルをリンクする場合、オブジェクト・ファイルの接尾部は、前の例のように **.o** でなければなりません。

cc コマンドを使用してソース・ファイルをアセンブルおよびリンクする場合、アセンブラー・ソース・ファイルの接尾部は **.s** でなければなりません。 **cc** コマンドは、この接尾部を持つすべてのファイルに対してアセンブラーを呼び出します。 **as** コマンドのオプション・フラグは、 **cc** コマンドを介してアセンブラーに送信できます。 構文は次のとおりです。

```
-Wa,Option1,Option2,...
```

次の例では、 **com** アセンブリー・モードを使用してソース・プログラムをアセンブルするためにアセンブラーを呼び出し、アセンブラー・リストとオブジェクト・ファイルを作成します。

```
cc -c -Wa,-mcom,-l file.s
```

cc コマンドはアセンブラーを呼び出してから、正常に処理を続行します。 したがって、

```
cc -Wa,-l,-oXfile.o file.s
```

アセンブラーによって作成されるオブジェクト・ファイルの名前は `Xfile.o` ですが、 **cc** コマンドによって呼び出されるリンケージ・エディター (**ld** コマンド) が `file.o` を検索するため、失敗します。

コマンド行にオプション・フラグが指定されていない場合、 **cc** コマンドは、 **xlc.cfg** 構成ファイルに定義されている必要なサポート・ライブラリーだけでなく、コンパイラー、アセンブラー、およびリンク・オプションも使用します。

注 : アセンブラーおよびリンケージ・エディターで定義されたオプション・フラグの中には、同じ文字を使用するものがあります。 したがって、 **xlc.cfg** 構成ファイルを使用してアセンブラー・オプション (**asopt**) およびリンク・エディター・オプション (**ldopt**) を定義する場合、 **asopt** および **ldopt** に重複文字があってはなりません。これは、 **cc** コマンドが重複文字を区別できないためです。

cc コマンドに渡されるオプション・フラグについて詳しくは、 **cc** コマンドを参照してください。

アセンブラー・パスについて

as コマンドを入力すると、アセンブラーはソース・プログラムに対して 2 つのパスを行います。

最初のパス

最初のパスでは、アセンブラーは、現行のアセンブリー・モードで命令が正しいかどうかを検査します。

最初のパスで、アセンブラーは以下のタスクを実行します。

- 命令が現行のアセンブリー・モードで正当であるかどうかを検査します。
- 要求する命令およびストレージ域にスペースを割り振ります。
- 可能な場合は、定数の値を入力します。
- 相互参照テーブルとも呼ばれるシンボル・テーブルを作成し、ステートメントのラベル・フィールドで検出されるすべてのシンボルについて、このテーブルにエントリーを作成します。

アセンブラーは一度に 1 行ずつソース・ファイルを読み取ります。 このソース・ステートメントのラベル・フィールドに有効なシンボルがある場合、アセンブラーは、そのシンボルがまだラベルとして使用されていないことを確認します。 シンボルがラベルとして初めて使用された場合、アセンブラーは、ラベルをシンボル・テーブルに追加し、現在のロケーション・カウンターの値をシンボルに割り当てます。 シンボルが既にラベルとして使用されている場合、アセンブラーはエラー・メッセージ `Redefinition of symbol` を戻し、シンボル値を再割り当てします。

次に、アセンブラーは命令のニーモニックを検査します。 簡略記号が現行のアセンブリー・モードで正しいマシン・インストラクションのものである場合、アセンブラーは命令のフォーマット (例えば、XO フォーマット) を決定します。 次に、アセンブラーは、命令のマシン・コードを保持するのに必要なバイト数を割り振ります。 ロケーション・カウンターの内容は、このバイト数だけ増分されます。

アセンブラーがコメント (前に # (ポンド記号)) または行末文字を検出すると、アセンブラーは次の命令ステートメントのスキャンを開始します。 アセンブラーは、読み取るステートメントがなくなるまで、ステートメントをスキャンし、そのシンボル・テーブルを作成し続けます。

最初のパスの終わりに、必要なすべてのスペースが割り振られ、プログラムで定義された各シンボルがシンボル・テーブル内のロケーション・カウンター値に関連付けられています。 読み取るソース・ステートメントがなくなると、2 番目のパスはプログラムの先頭から開始されます。

> **注:** 最初のパスでエラーが検出されると、アセンブリー・プロセスは終了し、2 番目のパスに進みません。 これが発生した場合、アセンブラー・リストには、アセンブラーの最初のパスで生成されたエラーおよび警告のみが含まれます。

2 番目のパス

2 番目のパスでは、アセンブラーは保管場所への記号参照についてオペランドを調べ、記号テーブル内の情報を使用してこれらの記号参照を解決します。

2 番目のパスでは、アセンブラーは次のことを行います。

- ストレージ・ロケーションへのシンボル参照のオペランドを調べ、シンボル・テーブル内の情報を使用してこれらのシンボル参照を解決します。
- 命令に無効な命令形式が含まれていないことを確認します。
- ソース・ステートメントをマシン・コードおよび定数に変換し、割り振られたスペースをオブジェクト・コードで埋めます。
- エラー・メッセージがある場合は、それを含むファイルを作成します。

2 番目のパスの開始時に、アセンブラーは各ソース・ステートメントをもう一度スキャンします。 アセンブラーは、各命令を変換するたびに、ロケーション・カウンターに入っている値を増分します。

特定のシンボルがソース・コードにあっても、そのシンボルがシンボル・テーブルに見つからない場合は、そのシンボルは定義されていません。 つまり、アセンブラーは、最初のパスでスキャンされたどのステートメントのラベル・フィールドにもシンボルを検出しなかったか、シンボルが **.comm**、 **.csect**、 **.lcomm**、 **.sect**、または **.set** 疑似命令の対象ではありませんでした。

これは、意図的な外部参照であるか、または記号名のつづりの誤りなどのプログラマー・エラーである可能性があります。 アセンブラーがエラーを示しています。 すべての外部参照は、 **.extern** または **.globl** ステートメントで指定する必要があります。

アセンブラーは、誤ったデータ位置合わせなどのエラーをログに記録します。 ただし、多くの位置合わせの問題は、アセンブリーを停止しないステートメントによって示されます。 これらの警告メッセージを表示するには、 **-w** フラグを使用する必要があります。

プログラマーがアセンブリー・エラーを訂正した後、プログラムをリンクする準備ができます。

注 : 最初のパスで警告のみが生成された場合、アセンブリー・プロセスは 2 番目のパスまで続行します。 アセンブラー・リストには、アセンブラーの 2 回目の受け渡し中に生成されたエラーおよび警告が含まれています。 最初のパスで生成された警告は、アセンブラー・リストには表示されません。

アセンブラー・リストの解釈

as コマンドの **-l** フラグは、アセンブラー言語ファイルのリストを作成します。

as コマンドの **-l** フラグは、アセンブラー言語ファイルのリストを作成します。

プログラマーが「hello, world」という語を表示したいとします。 C プログラムは次のようになります。

```
main ( )
{
     printf ("hello, world\n");
}
```

以下のコマンドを使用して、 **hello.s** ファイルをアセンブルします。

```
as -l hello.s
```

は、 **hello.lst** という名前の出力ファイルを生成します。 **hello.lst** の完全なアセンブラー・リストは次のとおりです。

```
hello.s                              V4.0               01/25/1994
File# Line#  Mode Name    Loc Ctr   Object Code         Source
0       1 |                                          ##################################
0       2 |                                          # C source code
0       3 |                                          ##################################
0       4 |                                          #   hello()
0       5 |                                          #   {
0       6 |                                          #    printf("hello,world\n");
0       7 |                                          #   }
0       8 |                                          ##################################
0       9 |                                          #   Compile as follows:
0      10 |                                          #   cc -o helloworld hello.s
0      11 |                                          #
0      12 |                                          ##################################
0      13 |                                           .file  "hello.s"
0      14 |                                          #Static data entry in
0      15 |                                          #T(able)O(f)C(ontents)
0      16 |                                           .toc
0      17 |  COM  data   00000000  00000040           T.data:   .tc data[tc],data[rw]
0      18 |                                           .globl main[ds]
0      19 |                                          #main[ds] contains definitions for
0      20 |                                          #runtime linkage of function main
0      21 |                                           .csect   main[ds]
0      22 |  COM  main   00000000  00000000             .long   .main[PR]
0      23 |  COM  main   00000004  00000050             .long   TOC[tc0]
0      24 |  COM  main   00000008  00000000             .long   0
0      25 |                                          #Function entry in
0      26 |                                          #T(able)O(f)C(ontents)
0      27 |                                           .toc
0      28 |  COM  .main  00000000  00000034           T.hello: .tc .main[tc],main[ds]
0      29 |                                           .globl   .main[PR]
0      30 |
0      31 |                                          #Set routine stack variables
0      32 |                                          #Values are specific to
0      33 |                                          #the current routine and can
0      34 |                                          #vary from routine to routine
0      35 |                          00000020          .set   argarea,     32
0      36 |                          00000018          .set   linkarea,    24
0      37 |                          00000000          .set   locstckarea, 0
0      38 |                          00000001          .set   ngprs,       1
0      39 |                          00000000          .set   nfprs,       0
0      40 |                          0000003c          .set   szdsa,       8*nfprs+4*ngprs+linkarea+
                                                                             argarea+locstckarea
0      41 |
0      42 |                                          #Main routine
0      43 |                                           .csect           .main[PR]
0      44 |
0      45 |
0      46 |                                          #PROLOG: Called Routines
0      47 |                                          #        Responsibilities
0      48 |                                           #Get link reg.
0      49 |  COM  .main  00000000  7c0802a6              mflr  0
0      50 |                                           #Not required to Get/Save CR
0      51 |                                           #because current routine does
0      52 |                                           #not alter it.
0      53 |
0      54 |                                           #Not required to Save FPR's
0      55 |                                           #14-31 because current routine
0      56 |                                           #does not alter them.
```

```
0      57 |                                                 #Save GPR 31.
0      58 |
0      59 |  COM  .main  00000004  bfe1fffc                  stm   31, -8*nfprs-4*ngprs(1)
0      60 |                                                 #Save LR if non-leaf routine.
0      61 |  COM  .main  00000008  90010008                  st    0, 8(1)
0      62 |                                                 #Decrement stack ptr and save
0      63 |                                                 #back chain.
0      64 |  COM  .main  0000000c  9421ffc4                  stu   1, -szdsa(1)
0      65 |
0      66 |
0      67 |                                               #Program body
0      68 |                                               #Load static data address
0      69 |  COM  .main  00000010  81c20000                  l     14,T.data(2)
0      70 |                                                 #Line 3, file hello.c
0      71 |                                               #Load address of data string
0      72 |                                               #from data addr.
0      73 |                                               #This is a parameter to printf()
0      74 |  COM  .main  00000014  386e0000                  cal   3,_helloworld(14)
0      75 |                                               #Call printf function
0      76 |  COM  .main  00000018  4bfffe9                   bl    .printf[PR]
0      77 |  COM  .main  0000001c  4def7b82                  cror  15, 15, 15
0      78 |
0      79 |
0      80 |                                               #EPILOG: Return Sequence
0      81 |                                                 #Get saved LR.
0      82 |  COM  .main  00000020  80010044                  l     0, szdsa+8(1)
0      83 |
0      84 |                                                 #Routine did not save CR.
0      85 |                                                 #Restore of CR not necessary.
0      86 |
0      87 |                                                 #Restore stack ptr
0      88 |  COM  .main  00000024  3021003c                  ai    1, 1, szdsa
0      89 |                                                 #Restore GPR 31.
0      90 |  COM  .main  00000028  bbe1fffc                  lm    31, -8*nfprs-4*ngprs(1)
0      91 |
0      92 |                                                 #Routine did not save FPR's.
0      93 |                                                 #Restore of FPR's not necessary.
0      94 |
0      95 |                                                 #Move return address
0      96 |                                                 #to Link Register.
0      97 |  COM  .main  0000002c  7c0803a6                         mtlr0
0      98 |                                                 #Return to address
0      99 |                                                 #held in Link Register.
0     100 |  COM  .main  00000030  4e800021                         brl
0     101 |
0     102 |
0     103 |                                               #External variables
0     104 |                                                   .extern.printf[PR]
0     105 |
0     106 |                                               ##################################
0     107 |                                                           #Data
0     108 |                                               ##################################
0     109 |                                               #String data placed in
0     110 |                                               #static csect data[rw]
0     111 |                                                      .csect data[rw]
0     112 |                                                      .align2
0     113 |                                               _helloworld:
0     114 |  COM  data   00000000  68656c6c                 .byte  0x68,0x65,0x6c,0x6c
0     115 |  COM  data   00000004  6f2c776f                 .byte  0x6f,0x2c,0x77,0x6f
0     116 |  COM  data   00000008  726c640a                 .byte  0x72,0x6c,0x64,0xa,0x0
          |  COM  data   0000000c  00
```

アセンブラー・リストの最初の行には、次の 2 つの情報が表示されます。

- ソース・ファイルの名前 (この場合は **hello.s**)
- リスト・ファイルが作成された日付

アセンブラー・リストには、いくつかの列があります。 列見出しは以下のとおりです。

項目	説明
File#	ソース・ファイル番号をリストします。 M4 マクロ・プロセッサー (**-l** オプション) によって組み込まれるファイルは、ステートメントが検出されたファイルの番号によって表示されます。
Line#	アセンブラー・ソース・コードの行番号を参照します。
Mode	この命令の現在のアセンブリー・モードを示します。

項目	説明
Name	ソース・コードのこの行の起点となる csect の名前をリストします。
Loc Ctr	アセンブラーのロケーション・カウンターに含まれている値をリストします。 リストには、オブジェクト・コードを生成するアセンブラー言語命令の場合にのみ、ロケーション・カウンター値が表示されます。
Object Code	アセンブラー・プログラムの各行によって生成されたオブジェクト・コードの 16 進表記を示します。 各命令は 32 ビットであるため、アセンブラー・リストの各行には最大 4 バイトが表示されます。 アセンブラー・ソース・コードの行の残りのバイトは、以下の行に表示されます。 **注 :** パス 2 が失敗した場合、アセンブラー・リストにはオブジェクト・コードは含まれません。
Source	プログラムのアセンブラー・ソース・コードをリストします。 1 行に表示される ASCII 文字は 100 文字までに制限されます。

コマンド行で **-s** オプション・フラグを使用すると、アセンブラー・リストに簡略記号相互参照情報が含まれます。

アセンブリー・モードが PowerPC ® カテゴリー (**com**、**ppc**、または **601**) の場合、1 つの列見出しは PowerPC ® です。 この列には、ソース・プログラムで POWER ® ファミリーのニーモニックが使用されている各インスタンスの PowerPC ® ニーモニックが含まれます。 **any** アセンブリー・モードはどのカテゴリーにも属しませんが、PowerPC ® カテゴリーの場合と同様に扱われます。

アセンブリー・モードが POWER ® ファミリー・カテゴリー (**pwr** または **pwr2**) の場合、1 つの列見出しは POWER ® ファミリーです。 この列には、ソース・プログラムで PowerPC ® ニーモニックが使用されている各インスタンスの POWER ® ファミリー・ニーモニックが含まれます。

以下のアセンブラー・リストでは、 **com** アセンブリー・モードを使用しています。 ソース・プログラムは POWER ® ファミリーのニーモニックを使用します。 アセンブラー・リストには、PowerPC ® ニーモニック相互参照があります。

```
L_dfmt_1.s                              V4.0                 01/26/1994
File# Line#  Mode  Name   Loc Ctr  Object Code  PowerPC      Source
0       1 |                                                 #%% -L
0       2 |                                                  machine   "com"
0       3 |                                                  csect dfmt[PR]
0       4 |                                                  using data,5
0       5 |
0       6 |  COM  dfmt   00000000  8025000c       lwz        l1,d1              # 8025000c
0       7 |  COM  dfmt   00000004  b8c50018       lmw        lm    6,d0         # b8650018
0       8 |  COM  dfmt   00000008  b0e50040                  sth   7,d8         # b0e50040
0       9 |  COM  dfmt   0000000c  80230020       lwz        l     1,0x20(3)    # 80230020
0      10 |  COM  dfmt   00000010  30220003       addic      ai    1,2,3        # 30220003
0      11 |  COM  dfmt   00000014  0cd78300       twi        ti    6,23,-32000 # 0cd78300
0      12 |  COM  dfmt    00000018  2c070af0                 cmpi  0,7,2800     # 2c070af0
0      13 |  COM  dfmt   0000001c  2c070af0                  cmpi  0,0,7,2800  # 2c070af0
0      14 |  COM  dfmt   00000020  30220003       subic      si    1,2,-3       # 30220003
0      15 |  COM  dfmt   00000024  34220003       subic.     si.   1,2,-3       # 34220003
0      16 |  COM  dfmt   00000028  703e00ff       andi.      andil.30,1,0xFF   # 703e00ff
0      17 |  COM  dfmt   0000002c  2b9401f4                  cmpli 7,20,500     # 2b9401f4
0      18 |  COM  dfmt   00000030  0c2501a4       twlgti     tlgti 5,420        # 0c2501a4
0      19 |  COM  dfmt   00000034  34220003       addic.     ai.   1,2,3        # 34220003
0      20 |  COM  dfmt   00000038  2c9ff380                  cmpi  1,31,-3200  # 2c9ff380
0      21 |  COM  dfmt   0000003c  281f0c80                  cmpli 0,31,3200   # 281f0c80
0      22 |  COM  dfmt   00000040  8ba5000c                  lbz   29,d1        # 8ba5000c
0      23 |  COM  dfmt   00000044  85e5000c       lwzu       lu    15,d1        # 85e5000c
0      24 |  COM  dfmt   00000048  1df5fec0       mulli      muli  15,21,-320  # 1df5fec0
0      25 |  COM  dfmt   0000004c  62af0140       ori        oril  15,21,320   # 62af0140
0      26 |  COM  dfmt   00000050  91e5000c       stw        st    15,d1        # 91e5000c
0      27 |  COM  dfmt   00000054  bde5000c       stmw       stm   15,d1        # bde5000c
0      28 |  COM  dfmt   00000058  95e5000c       stwu       stu   15,d1        # 95e5000c
0      29 |  COM  dfmt   0000005c  69ef0960       xori       xoril 15,15,2400  # 69ef0960
0      30 |  COM  dfmt   00000060  6d8c0960       xoris      xoriu 12,12,2400  # 6d8c0960
0      31 |  COM  dfmt   00000064  3a9eff38                  addi  20,30,-200  # 3a9eff38
0      32 |
0      33 |                                                     .csect also[RW]
```

```
0      34 |                                                    data:
0      35 |  COM  also   00000000  00000000                    .long  0,0,0
          |              00000004    ....
          |  COM  also   00000008  00000000
0      36 |  COM  also   0000000c  00000003              d1:.long  3,4,5   # d1 = 0xC = 12
          |  COM  also   00000010  00000004
          |  COM  also   00000014  00000005
0      37 |  COM  also   00000018  00000068                 d0:    .long   data # d0 = 0x18 = 24
0      38 |  COM  also   0000001c  00000000                 data2: .space  36
          |              00000020    ....
          |  COM  also   0000003c  000000000
       39 |  COM  also   00000040  000023e0                 d8:    .long   9184  # d8 = 0x40 =
64
0      40 |  COM  also   00000044  ffffffff                 d9:    .long   0xFFFFFFFF # d9 =
0x44
0      41 |                                            #
0      42 |                                            # 0000 00000000 00000000 00000000
00000003
0      43 |                                            # 0010 00000004 00000005 0000000C
00000000
0      44 |                                            # 0020 00000000 00000000 00000000
00000000
0      45 |                                            # 0030 000023E0
```

以下のアセンブラー・リストでは、 **pwr** アセンブリー・モードを使用しています。 ソース・プログラムは PowerPC ® ニーモニックを使用します。 アセンブラー・リストには、POWER ® ファミリーのニーモニック相互参照があります。

```
 L_dfmt_2.s                             V4.0               01/26/1994
File# Line#  Mode Name    Loc Ctr  Object Code  POWER       Source
0       1 |                                               #%% -L
0       2 |                                               .machine  "pwr"
0       3 |                                               .csect dfmt[PR]
0       4 |                                               .using data,5
0       5 |  PWR  dfmt   00000000  8025000c       l        lwz      1,d1
0       6 |  PWR  dfmt   00000004  b8650018       lm       lmw      3,d0
0       7 |  PWR  dfmt   00000008  b0e50040                sth      7,d8
0       8 |  PWR  dfmt   0000000c  80230020       l        lwz      1,0x20(3)
0       9 |  PWR  dfmt   00000010  30220003       ai       addic    1,2,3
0      10 |  PWR  dfmt   00000014  0cd78300       ti       twi      6,23,-32000
0      11 |  PWR  dfmt   00000018  2c070af0                cmpi     0,7,2800
0      12 |  PWR  dfmt   0000001c  2c070af0                cmpi     0,0,7,2800
0      13 |  PWR  dfmt   00000020  30220003       si       subic    1,2,-3
0      14 |  PWR  dfmt   00000024  34220003       si.      subic.   1,2,-3
0      15 |  PWR  dfmt   00000028  703e00ff       andil.   andi.    30,1,0xFF
0      16 |  PWR  dfmt   0000002c  2b9401f4                cmpli    7,20,500
0      17 |  PWR  dfmt   00000030  0c2501a4       tlgti    twlgti   5,420
0      18 |  PWR  dfmt   00000034  34220003       ai.      addic.   1,2,3
0      19 |  PWR  dfmt   00000038  2c9ff380                cmpi     1,31,-3200
0      20 |  PWR  dfmt   0000003c  281f0c80                cmpli    0,31,3200
0      21 |  PWR  dfmt   00000040  8ba5000c                lbz      29,d1
0      22 |  PWR  dfmt   00000044  85e5000c       lu       lwzu     15,d1
0      23 |  PWR  dfmt   00000048  1df5fec0       muli     mulli    15,21,-320
0      24 |  PWR  dfmt   0000004c  62af0140       oril     ori      15,21,320
0      25 |  PWR  dfmt   00000050  91e5000c       st       stw      15,d1
0      26 |  PWR  dfmt   00000054  bde5000c       stm      stmw     15,d1
0      27 |  PWR  dfmt   00000058  95e5000c       stu      stwu     15,d1
0      28 |  PWR  dfmt   0000005c  69ef0960       xoril    xori     15,15,2400
0      29 |  PWR  dfmt   00000060  6d8c0960       xoriu    xoris    12,12,2400
0      30 |  PWR  dfmt   00000064  3a9eff38                addi     20,30,-200
0      31 |
0      32 |
0      33 |                                                        .csect also[RW]
0      34 |                                                 data:
0      35 |  PWR  also   00000000  00000000                     .long     0,0,0
          |              00000004    ....
          |  PWR  also   00000008  00000000
0      36 |  PWR  also   0000000c  00000003                  d1: long     3,4,5
          |  PWR  also   00000010  00000004                                # d1 = 0xc = 12
          |  PWR  also   00000014  00000005
0      37 |  PWR  also   00000018  00000068                  d0: long     data       # d0 = 0x18 = 24
0      38 |  PWR  also   0000001c  00000000               data2: space  36
          |              00000020    ....
          |  PWR  also   0000003c  00000000
0      39 |  PWR  also   00000040  000023e0                  d8: long     9184       # d8 = 0x40 = 64
0      40 |  PWR  also   00000044  ffffffff                  d9: long     0xFFFFFFFF # d9 = 0x44
0      41 |                                                     #
0      42 |                                                  # 0000 00000000 00000000 00000000
00000003
```

```
0      43 |                                          # 0010 00000004 00000005 0000000C
00000000
0      44 |                                          # 0020 00000000 00000000 00000000
00000000
0      45 |                                          # 0030 000023E0
```

記号相互参照の解釈

hello.s アセンブリー・プログラムのシンボルの相互参照の例。

hello.s アセンブリー・プログラムのシンボル相互参照の例を以下に示します。

```
Symbol          File            CSECT           Line #
.main           hello.s         --                22
.main           hello.s         .main             28   *
.main           hello.s         --                29
.main           hello.s         .main             43   *
.printf         hello.s         --                76
.printf         hello.s         --               104
T.data          hello.s         data              17   *
T.data          hello.s         data              69
T.hello         hello.s         .main             28   *
TOC             hello.s         TOC               23
_helloworld     hello.s         --                74
_helloworld     hello.s         data             113   *
argarea         hello.s         --                35   *
argarea         hello.s         --                40
data            hello.s         --                17
data            hello.s         data              17   *
data            hello.s         data             111   *
linkarea        hello.s         --                36   *
linkarea        hello.s         --                40
locstckarea     hello.s         --                37   *
locstckarea     hello.s         --                40
main            hello.s         --                18
main            hello.s         main              21   *
main            hello.s         main              28
nfprs           hello.s         --                39   *
nfprs           hello.s         --                40
nfprs           hello.s         --                59
nfprs           hello.s         --                90
ngprs           hello.s         --                38   *
ngprs           hello.s         --                40
ngprs           hello.s         --                59
ngprs           hello.s         --                90
szdsa           hello.s         --                40   *
szdsa           hello.s         --                64
szdsa           hello.s         --                82
szdsa           hello.s         --                88
```

最初の列には、ソース・プログラムに表示されるシンボル名がリストされています。 2 番目の列には、シンボルが置かれているソース・ファイル名がリストされています。 3 番目の列には、シンボルが定義または配置されている csect 名がリストされています。

csect 名をリストする列で、 -- (二重ダッシュ) は次のいずれかを意味します。

- シンボルの csect がまだ定義されていません。 この例では、1 番目と 3 番目の `.main` (`.main[PR]`) は、行 42 で定義されています。
- シンボルは外部シンボルです。 この例では、 `.printf` は外部シンボルであるため、どの csect にも関連付けられていません。
- 定義されるシンボルは、シンボリック定数です。 **.set** 疑似命令を使用してシンボルを定義する場合、そのシンボルはシンボリック定数であり、それに関連付けられた csect を持ちません。 この例では、 `argarea`、 `linkarea`、 `locstckarea`、 `nfprs`、 `ngprs`、および `szdsa` はシンボリック定数です。

4 番目の列には、記号が置かれている行番号がリストされます。 行番号の後の * (アスタリスク) は、記号がこの行で定義されていることを示します。 行番号の後にアスタリスクがない場合、記号はその行で参照されます。

サブルーチンのリンケージ規約

サブルーチン・リンケージ規約。

この記事では、以下について説明します。

リンケージ規約の概要

サブルーチン・リンケージ規約は、サブルーチンの入り口と出口におけるマシン状態を記述します。

サブルーチン・リンケージ規約は、サブルーチンの入り口と出口におけるマシン状態を記述します。 このスキームに従うと、同じ言語または異なる言語で別々にコンパイルされたルーチンを、呼び出されたときにリンクして実行することができます。

リンケージ規約により、パラメーターの受け渡しと戻り値は、浮動小数点レジスター (FPR)、汎用レジスター (GPR)、またはその両方に入れることができます。

オブジェクト・モードに関する考慮事項

オブジェクト・モードの考慮事項は、32 ビット・モードと 64 ビット・モードの両方に適用されます。

以下の説明は、32 ビット・モードと 64 ビット・モードの両方に適用されます。以下に注意してください。

- 64 ビット・モードの汎用レジスターは、64 ビット幅 (ダブルワード) です。 これは、スタックのスペース使用量が、レジスター・ストレージ用に 2 倍ずつ増加することを意味します。 ワード という用語が使用される場所では (特に明記されていない限り)、問題のオブジェクトのサイズは 32 ビット・モードでは 1 ワード、64 ビット・モードでは 2 ワード (ダブルワード) であると想定します。
- 64 ビット・モードの場合は、ランタイム・スタックの数値に示されているオフセットを 2 倍にする必要があります。 32 ビット・モードでは、示されているスタックには 56 バイトが必要です。
  - 6 つのレジスター CR、LR、コンパイラー予約、リンカー予約、および保管 TOC のそれぞれについて 1 ワード。
  - 8 個の揮発性レジスターを表す 8 ワード。

  これは、合計で 14 ワード (56 バイト) です。 64 ビット・モードでは、各フィールドは 2 倍の大きさ (ダブルワード) であるため、28 ワード (112 バイト) が必要です。
- 浮動小数点レジスターは、両方のモードで同じフォーマットで保管されます。 ストレージの要件は同じです。
- スタック・ポインターの位置合わせの要件は、両方のモードで同じままです。
- 以下にリストされている GPR 保管ルーチンは、32 ビット・モードでレジスターを保管する方法を示しています。 64 ビット・モードの場合、 GPR1(スタック・ポインター・レジスター) からのオフセットは、示されている値の 2 倍になります。 さらに、使用されるロード命令は `ld` になり、ストア命令は `stdu` になります。

レジスターの使用法と規則

PowerPC ® 32 ビット・アーキテクチャーには、32 個の GPR と 32 個の FPR があります。

PowerPC ® 32 ビット・アーキテクチャーには、32 個の GPR と 32 個の FPR があります。 各 GPR の幅は 32 ビット、各 FPR の幅は 64 ビットです。 分岐、例外処理、およびその他の目的のための特殊レジスターもあります。 汎用レジスター規則表は、GPR がどのように使用されるかを示しています。

表 *2.* 汎用レジスター規則 *(General-Purpose Register Conventions)*

レジスター	状況	Use
GPR0	volatile	関数プロローグ内。
GPR1	専用の	スタック・ポインター。
GPR2	専用の	目次 (TOC) のポインター。
GPR3	volatile	関数の引数リストの最初のワード。スカラー関数の最初のワードが戻されます。

表 2. 汎用レジスター規則 *(General-Purpose Register Conventions)* (続き)		
レジスター	**状況**	**Use**
GPR4	volatile	関数の引数リストの 2 番目のワード。スカラー関数の 2 番目のワードが戻されます。
GPR5	volatile	関数の引数リストの 3 番目のワード。
GPR6	volatile	関数の引数リストの 4 番目のワード。
GPR7	volatile	関数の引数リストの 5 番目のワード。
GPR8	volatile	関数の引数リストの 6 番目のワード。
GPR9	volatile	関数の引数リストの 7 番目のワード。
GPR10	volatile	関数の引数リストの 8 番目のワード。
GPR11	volatile	ポインターによる呼び出し、およびそれを必要とする言語用の環境ポインターとしての呼び出し (例えば、PASCAL)。
GPR12	volatile	特定の言語および **glink** コードで必要とされる特殊な例外処理の場合。
GPR13	予約済み	64 ビット環境で予約済み。システム・コール間では復元されません。
GPR14:GPR31	不揮発性	これらのレジスターは、関数呼び出し全体にわたって保持する必要があります。

GPR を使用する推奨される方法は、揮発性レジスターを最初に使用することです。 次に、 GPR31 で始まる不揮発性レジスターを降順で使用します。 GPR1 および GPR2 は、それぞれスタックおよび目次 (TOC) 領域ポインターとして専用にする必要があります。 GPR1 および GPR2 は、呼び出し全体にわたって保管されているように見える必要があり、呼び出しが行われたときと同じ戻り値を持つ必要があります。

揮発性レジスターは、呼び出しの間に破棄されると想定されるスクラッチ・レジスターであるため、呼び出し先によって保管されません。 揮発性レジスターは、前の表に示すように、特定の目的にも使用されます。 不揮発性レジスターおよび専用レジスターは、変更された場合は保管および復元する必要があります。したがって、関数呼び出し全体でその値を保持することが保証されます。

浮動小数点レジスター規則表は、FPR の使用法を示しています。

表 3. 浮動小数点レジスター規則		
レジスター	**状況**	**Use**
FPR0	volatile	スクラッチ・レジスターとして。
FPR1	volatile	最初の浮動小数点パラメーター。浮動小数点スカラー戻りの最初の 8 バイト。
FPR2	volatile	2 番目の浮動小数点パラメーター。浮動小数点スカラー戻りの 2 番目の 8 バイト。
FPR3	volatile	3 番目の浮動小数点パラメーター。浮動小数点スカラー戻りの 3 番目の 8 バイト。
FPR4	volatile	4 番目の浮動小数点パラメーター。浮動小数点スカラー戻りの 4 番目の 8 バイト。
FPR5	volatile	5 番目の浮動小数点パラメーター。
FPR6	volatile	6 番目の浮動小数点パラメーター。
FPR7	volatile	7 番目の浮動小数点パラメーター。
FPR8	volatile	8 番目の浮動小数点パラメーター。

表 3. 浮動小数点レジスター規則 (続き)		
レジスター	**状況**	**Use**
FPR9	volatile	9 番目の浮動小数点パラメーター。
FPR10	volatile	10 番目の浮動小数点パラメーター。
FPR11	volatile	11 番目の浮動小数点パラメーター。
FPR12	volatile	12 番目の浮動小数点パラメーター。
FPR13	volatile	13 番目の浮動小数点パラメーター。
FPR14:FPR31	不揮発性	変更する場合は、呼び出し間で保持する必要があります。

FPR の推奨される使用方法は、最初に揮発性レジスターを使用することです。 次に、不揮発性レジスターは、 FPR31 から始めて FPR14 までの降順で使用されます。

スカラーのみが複数のレジスターに戻されます。 必要なレジスターの数は、スカラーのサイズとタイプによって異なります。 浮動小数点値の場合、以下の結果になります。

- 128 ビット浮動小数点値は、 FPR1 の高位 64 ビットと FPR2 の下位 64 ビットを返します。
- 8 バイトまたは 16 バイトの複素数値は、 FPR1 の実数部と FPR2 の虚数部を返します。
- 32 バイトの複素数値は、 FPR1 および FPR2 の 128 ビット浮動小数点値として実数部分を返します。 FPR1 の高位 64 ビットと FPR2 の下位 64 ビットが含まれます。 32 バイトの複素数値の虚数部分は、 FPR3 の高位 64 ビットと FPR4 の下位 64 ビットを戻します。

複合タイプの呼び出し規則の例

```
complex double foo(complex double);
```

引数は fp1 および fp2 に渡され、結果は fp1 および fp2 に戻されます。 後続の複素数倍精度パラメーターは、偶数または奇数のペアを使用して、次に使用可能な 2 つのレジスター (fp13 まで) に渡されます。 fp13 の後、呼び出し側のスタック・フレームの先頭にあるメモリー内のパラメーター域に渡されます。

注 : スキップされたレジスターは、後のパラメーターには使用されません。 さらに、これらのレジスターは呼び出し元によって初期化されません。また、呼び出される関数は、スキップされたレジスター内に保管されている値に依存してはなりません。

単精度複素数 (複素数浮動小数点) は、倍精度と同じ方法で渡され、値は倍精度に拡張されます。

倍精度および単精度の複素数 (複素数の倍精度および複素数の浮動小数点) は、単精度値が倍精度に拡大された fp1 および fp2 で戻されます。

4 倍精度複素数 (複素数長倍精度) パラメーターは、次に使用可能な 4 つのレジスターで、 fp1 から fp13 に渡され、その後パラメーター域に渡されます。 レジスターが充てんされる順序は、実数部の上半分、実数部の下半分、虚数部の上半分、および虚数部の下半分です。

注 : AIX 構造体では、クラスと共用体は、fprs ではなく gprs (またはメモリー) で渡されます。 これは、クラスおよび共用体に浮動小数点値が含まれている場合にも当てはまります。 PPC 上の Linux では、メモリー内のコピーのアドレスは、次に使用可能な gpr (またはメモリー内) に渡されます。 varargs パラメーターは具体的に処理され、通常は fprs と gprs の両方に渡されます。

10 進浮動小数点型の呼び出し規則 (_Decimal128)

_Decimal64 パラメーターは、次に使用可能な fpr で渡され、結果は fp1 で返されます。

_Decimal32 パラメーターは、次に使用可能な fpr の下半分で渡され、結果は _Decimal64 に変換されずに fp1 の下半分で返されます。

_Decimal128 パラメーターは、レジスターのスキップを意味し、結果が偶数と奇数のペア fpr2 および fpr3 で返される場合でも、次に使用可能な偶数と奇数のペア (またはメモリー) で渡されます。 その理由は、すべての算術命令で偶数と奇数のレジスター・ペアを使用する必要があるからです。

注 : スキップされたレジスターは、後のパラメーターには使用されません。 さらに、これらのレジスターは呼び出し元によって初期化されません。また、呼び出される関数は、スキップされたレジスター内に保管されている値に依存してはなりません。

float または double とは異なり、DFP では常に関数プロトタイプが必要です。 したがって、 _Decimal32 を _Decinmal64 に拡大する必要はありません。

10 進浮動小数点型の呼び出し規則の例 (_Decimal32)

```
#include <float.h>
#define DFP_ROUND_HALF_UP 4

_Decimal32 Add_GST_and_Ontario_PST_d32 (_Decimal32 price)
{
_Decimal32 gst;
_Decimal32 pst;
_Decimal32 total;
long original_rounding_mode = __dfp_get_rounding_mode ( );
__dfp_set_rounding_mode (DFP_ROUND_HALF_UP);
gst = price * 0.06dd;
pst = price * 0.08dd;
total = price + gst + pst;
__dfp_set_rounding_mode (original_rounding_mode);
return (total);
}

| 000000 PDEF Add_GST_and_Ontario_PST_d32
>> 0| PROC price,fp1
0| 000000 stw 93E1FFFC 1 ST4A #stack(gr1,-4)=gr31
0| 000004 stw 93C1FFF8 1 ST4A #stack(gr1,-8)=gr30
0| 000008 stwu 9421FF80 1 ST4U gr1,#stack(gr1,-128)=gr1
0| 00000C lwz 83C20004 1 L4A gr30=.+CONSTANT_AREA(gr2,0)
0| 000010 addi 38A00050 1 LI gr5=80
0| 000014 ori 60A30000 1 LR gr3=gr5
>> 0| 000018 stfiwx 7C211FAE 1 STDFS price(gr1,gr3,0)=fp1
9| 00001C mffs FC00048E 1 LFFSCR fp0=fcr
9| 000020 stfd D8010058 1 STFL #MX_SET1(gr1,88)=fp0
9| 000024 lwz 80010058 1 L4A gr0=#MX_SET1(gr1,88)
9| 000028 rlwinm 5400077E 1 RN4 gr0=gr0,0,0x7
9| 00002C stw 9001004C 1 ST4A original_rounding_mode(gr1,76)=gr0
10| 000030 mtfsfi FF81410C 1 SETDRND fcr=4,fcr
11| 000034 ori 60A30000 1 LR gr3=gr5
11| 000038 lfiwax 7C011EAE 1 LDFS fp0=price(gr1,gr3,0)
11| 00003C dctdp EC000204 1 CVDSDL fp0=fp0,fcr
11| 000040 lfd C83E0000 1 LDFL fp1=+CONSTANT_AREA(gr30,0)
11| 000044 dmul EC000844 1 MDFL fp0=fp0,fp1,fcr
11| 000048 drsp EC000604 1 CVDLDS fp0=fp0,fcr
11| 00004C addi 38600040 1 LI gr3=64
11| 000050 ori 60640000 1 LR gr4=gr3
11| 000054 stfiwx 7C0127AE 1 STDFS gst(gr1,gr4,0)=fp0
12| 000058 ori 60A40000 1 LR gr4=gr5
12| 00005C lfiwax 7C0126AE 1 LDFS fp0=price(gr1,gr4,0)
12| 000060 dctdp EC000204 1 CVDSDL fp0=fp0,fcr
12| 000064 lfd C83E0008 1 LDFL fp1=+CONSTANT_AREA(gr30,8)
12| 000068 dmul EC000844 1 MDFL fp0=fp0,fp1,fcr
12| 00006C drsp EC000604 1 CVDLDS fp0=fp0,fcr
12| 000070 addi 38800044 1 LI gr4=68
12| 000074 ori 60860000 1 LR gr6=gr4
12| 000078 stfiwx 7C0137AE 1 STDFS pst(gr1,gr6,0)=fp0
13| 00007C lfiwax 7C012EAE 1 LDFS fp0=price(gr1,gr5,0)
13| 000080 lfiwax 7C211EAE 1 LDFS fp1=gst(gr1,gr3,0)
13| 000084 dctdp EC000204 1 CVDSDL fp0=fp0,fcr
13| 000088 dctdp EC200A04 1 CVDSDL fp1=fp1,fcr
13| 00008C mffs FC40048E 1 LFFSCR fp2=fcr
13| 000090 stfd D8410058 1 STFL #MX_SET1(gr1,88)=fp2
13| 000094 lwz 80010058 1 L4A gr0=#MX_SET1(gr1,88)
13| 000098 rlwinm 5400077E 1 RN4 gr0=gr0,0,0x7
13| 00009C mtfsfi FF81710C 1 SETDRND fcr=7,fcr
13| 0000A0 dadd EC000804 1 ADFL fp0=fp0,fp1,fcr
13| 0000A4 stw 90010058 1 ST4A #MX_SET1(gr1,88)=gr0
13| 0000A8 lfd C8210058 1 LFL fp1=#MX_SET1(gr1,88)
13| 0000AC mtfsf FC030D8E 1 LFSCR8 fsr,fcr=fp1,1,1
13| 0000B0 addi 38000007 1 LI gr0=7
13| 0000B4 addi 38600000 1 LI gr3=0
13| 0000B8 stw 90610068 1 ST4A #MX_CONVF1_0(gr1,104)=gr3
13| 0000BC stw 9001006C 1 ST4A #MX_CONVF1_0(gr1,108)=gr0
13| 0000C0 lfd C8210068 1 LDFL fp1=#MX_CONVF1_0(gr1,104)
13| 0000C4 drrnd EC010646 1 RRDFL fp0=fp0,fp1,3,fcr
```

```
13| 0000C8 drsp EC000604 1 CVDLDS fp0=fp0,fcr
13| 0000CC lfiwax 7C2126AE 1 LDFS fp1=pst(gr1,gr4,0)
13| 0000D0 dctdp EC000204 1 CVDSDL fp0=fp0,fcr
13| 0000D4 dctdp EC200A04 1 CVDSDL fp1=fp1,fcr
13| 0000D8 mffs FC40048E 1 LFFSCR fp2=fcr
13| 0000DC stfd D8410058 1 STFL #MX_SET1(gr1,88)=fp2
13| 0000E0 lwz 80810058 1 L4A gr4=#MX_SET1(gr1,88)
13| 0000E4 rlwinm 5484077E 1 RN4 gr4=gr4,0,0x7
13| 0000E8 mtfsfi FF81710C 1 SETDRND fcr=7,fcr
13| 0000EC dadd EC000804 1 ADFL fp0=fp0,fp1,fcr
13| 0000F0 stw 90810058 1 ST4A #MX_SET1(gr1,88)=gr4
13| 0000F4 lfd C8210058 1 LFL fp1=#MX_SET1(gr1,88)
13| 0000F8 mtfsf FC030D8E 1 LFSCR8 fsr,fcr=fp1,1,1
13| 0000FC stw 90610068 1 ST4A #MX_CONVF1_0(gr1,104)=gr3
13| 000100 stw 9001006C 1 ST4A #MX_CONVF1_0(gr1,108)=gr0
13| 000104 lfd C8210068 1 LDFL fp1=#MX_CONVF1_0(gr1,104)
13| 000108 drrnd EC010646 1 RRDFL fp0=fp0,fp1,3,fcr
13| 00010C drsp EC000604 1 CVDLDS fp0=fp0,fcr
13| 000110 addi 38600048 1 LI gr3=72
13| 000114 ori 60640000 1 LR gr4=gr3
13| 000118 stfiwx 7C0127AE 1 STDFS total(gr1,gr4,0)=fp0
14| 00011C lwz 8001004C 1 L4A gr0=original_rounding_mode(gr1,76)
14| 000120 stw 90010058 1 ST4A #MX_SET1(gr1,88)=gr0
14| 000124 lfd C8010058 1 LFL fp0=#MX_SET1(gr1,88)
14| 000128 mtfsf FC03058E 1 LFSCR8 fsr,fcr=fp0,1,1
>> 15| 00012C lfiwax 7C211EAE 1 LDFS fp1=total(gr1,gr3,0)
16| CL.1:
16| 000130 lwz 83C10078 1 L4A gr30=#stack(gr1,120)
16| 000134 addi 38210080 1 AI gr1=gr1,128
16| 000138 bclr 4E800020 1 BA lr
```

10 進浮動小数点型の呼び出し規則の例 (_Decimal64)

```
#include <float.h>
#define DFP_ROUND_HALF_UP 4

_Decimal64 Add_GST_and_Ontario_PST_d64 (_Decimal64 price)
{
_Decimal64 gst;
_Decimal64 pst;
_Decimal64 total;
long original_rounding_mode = __dfp_get_rounding_mode ( );
__dfp_set_rounding_mode (DFP_ROUND_HALF_UP);
gst = price * 0.06dd;
pst = price * 0.08dd;
total = price + gst + pst;
__dfp_set_rounding_mode (original_rounding_mode);
return (total);
}

| 000000 PDEF Add_GST_and_Ontario_PST_d64
>> 0| PROC price,fp1
0| 000000 stw 93E1FFFC 1 ST4A #stack(gr1,-4)=gr31
0| 000004 stw 93C1FFF8 1 ST4A #stack(gr1,-8)=gr30
0| 000008 stwu 9421FF80 1 ST4U gr1,#stack(gr1,-128)=gr1
0| 00000C lwz 83C20004 1 L4A gr30=.+CONSTANT_AREA(gr2,0)
>> 0| 000010 stfd D8210098 1 STDFL price(gr1,152)=fp1
9| 000014 mffs FC00048E 1 LFFSCR fp0=fcr
9| 000018 stfd D8010060 1 STFL #MX_SET1(gr1,96)=fp0
9| 00001C lwz 80010060 1 L4A gr0=#MX_SET1(gr1,96)
9| 000020 rlwinm 5400077E 1 RN4 gr0=gr0,0,0x7
9| 000024 stw 90010058 1 ST4A original_rounding_mode(gr1,88)=gr0
10| 000028 mtfsfi FF81410C 1 SETDRND fcr=4,fcr
11| 00002C lfd C8010098 1 LDFL fp0=price(gr1,152)
11| 000030 lfd C83E0000 1 LDFL fp1=+CONSTANT_AREA(gr30,0)
11| 000034 dmul EC000844 1 MDFL fp0=fp0,fp1,fcr
11| 000038 stfd D8010040 1 STDFL gst(gr1,64)=fp0
12| 00003C lfd C8010098 1 LDFL fp0=price(gr1,152)
12| 000040 lfd C83E0008 1 LDFL fp1=+CONSTANT_AREA(gr30,8)
12| 000044 dmul EC000844 1 MDFL fp0=fp0,fp1,fcr
12| 000048 stfd D8010048 1 STDFL pst(gr1,72)=fp0
13| 00004C lfd C8010098 1 LDFL fp0=price(gr1,152)
13| 000050 lfd C8210040 1 LDFL fp1=gst(gr1,64)
13| 000054 dadd EC000804 1 ADFL fp0=fp0,fp1,fcr
13| 000058 lfd C8210048 1 LDFL fp1=pst(gr1,72)
13| 00005C dadd EC000804 1 ADFL fp0=fp0,fp1,fcr
13| 000060 stfd D8010050 1 STDFL total(gr1,80)=fp0
14| 000064 lwz 80010058 1 L4A gr0=original_rounding_mode(gr1,88)
14| 000068 stw 90010060 1 ST4A #MX_SET1(gr1,96)=gr0
```

```
14| 00006C lfd C8010060 1 LFL fp0=#MX_SET1(gr1,96)
14| 000070 mtfsf FC03058E 1 LFSCR8 fsr,fcr=fp0,1,1
>> 15| 000074 lfd C8210050 1 LDFL fp1=total(gr1,80)
16| CL.1:
16| 000078 lwz 83C10078 1 L4A gr30=#stack(gr1,120)
16| 00007C addi 38210080 1 AI gr1=gr1,128
16| 000080 bclr 4E800020 1 BA lr
```

10 進浮動小数点型の呼び出し規則の例 (_Decimal128)

```
include <float.h>
#define DFP_ROUND_HALF_UP 4

_Decimal128 Add_GST_and_Ontario_PST_d128 (_Decimal128 price)
{
_Decimal128 gst;
_Decimal128 pst;
_Decimal128 total;
long original_rounding_mode = __dfp_get_rounding_mode ( );
__dfp_set_rounding_mode (DFP_ROUND_HALF_UP);
gst = price * 0.06dd;
pst = price * 0.08dd;
total = price + gst + pst;
__dfp_set_rounding_mode (original_rounding_mode);
return (total);
}

| 000000 PDEF Add_GST_and_Ontario_PST_d128
>> 0| PROC price,fp2,fp3
0| 000000 stw 93E1FFFC 1 ST4A #stack(gr1,-4)=gr31
0| 000004 stw 93C1FFF8 1 ST4A #stack(gr1,-8)=gr30
0| 000008 stwu 9421FF70 1 ST4U gr1,#stack(gr1,-144)=gr1
0| 00000C lwz 83C20004 1 L4A gr30=.+CONSTANT_AREA(gr2,0)
>> 0| 000010 stfd D84100A8 1 STDFL price(gr1,168)=fp2
>> 0| 000014 stfd D86100B0 1 STDFL price(gr1,176)=fp3
9| 000018 mffs FC00048E 1 LFFSCR fp0=fcr
9| 00001C stfd D8010078 1 STFL #MX_SET1(gr1,120)=fp0
9| 000020 lwz 80010078 1 L4A gr0=#MX_SET1(gr1,120)
9| 000024 rlwinm 5400077E 1 RN4 gr0=gr0,0,0x7
9| 000028 stw 90010070 1 ST4A original_rounding_mode(gr1,112)=gr0
10| 00002C mtfsfi FF81410C 1 SETDRND fcr=4,fcr
11| 000030 lfd C80100A8 1 LDFL fp0=price(gr1,168)
11| 000034 lfd C82100B0 1 LDFL fp1=price(gr1,176)
11| 000038 lfd C85E0000 1 LDFL fp2=+CONSTANT_AREA(gr30,0)
11| 00003C lfd C87E0008 1 LDFL fp3=+CONSTANT_AREA(gr30,8)
11| 000040 dmulq FC001044 1 MDFE fp0,fp1=fp0-fp3,fcr
11| 000044 stfdp F4010040 1 STDFE gst(gr1,64)=fp0,fp1
12| 000048 lfd C80100A8 1 LDFL fp0=price(gr1,168)
12| 00004C lfd C82100B0 1 LDFL fp1=price(gr1,176)
12| 000050 lfd C85E0010 1 LDFL fp2=+CONSTANT_AREA(gr30,16)
12| 000054 lfd C87E0018 1 LDFL fp3=+CONSTANT_AREA(gr30,24)
12| 000058 dmulq FC001044 1 MDFE fp0,fp1=fp0-fp3,fcr
12| 00005C stfdp F4010050 1 STDFE pst(gr1,80)=fp0,fp1
13| 000060 lfd C80100A8 1 LDFL fp0=price(gr1,168)
13| 000064 lfd C82100B0 1 LDFL fp1=price(gr1,176)
13| 000068 lfd C8410040 1 LDFL fp2=gst(gr1,64)
13| 00006C lfd C8610048 1 LDFL fp3=gst(gr1,72)
13| 000070 daddq FC001004 1 ADFE fp0,fp1=fp0-fp3,fcr
13| 000074 lfd C8410050 1 LDFL fp2=pst(gr1,80)
13| 000078 lfd C8610058 1 LDFL fp3=pst(gr1,88)
13| 00007C daddq FC001004 1 ADFE fp0,fp1=fp0-fp3,fcr
13| 000080 stfdp F4010060 1 STDFE total(gr1,96)=fp0,fp1
14| 000084 lwz 80010070 1 L4A gr0=original_rounding_mode(gr1,112)
14| 000088 stw 90010078 1 ST4A #MX_SET1(gr1,120)=gr0
14| 00008C lfd C8010078 1 LFL fp0=#MX_SET1(gr1,120)
14| 000090 mtfsf FC03058E 1 LFSCR8 fsr,fcr=fp0,1,1
>> 15| 000094 lfd C8410060 1 LDFL fp2=total(gr1,96)
>> 15| 000098 lfd C8610068 1 LDFL fp3=total(gr1,104)
16| CL.1:
16| 00009C lwz 83C10088 1 L4A gr30=#stack(gr1,136)
16| 0000A0 addi 38210090 1 AI gr1=gr1,144
16| 0000A4 bclr 4E800020 1 BA lr
```

PowerPC ® の特殊レジスター

「特殊目的レジスター規則」には、PowerPC ® 特殊目的レジスター (SPR) が表示されます。

「特殊目的レジスター規則」表には、PowerPC ® 特殊目的レジスター (SPR) が表示されます。 これらは、レジスター規則が存在する唯一の SPR です。

表 4. 特殊目的レジスターの規則		
フィールドの登録または登録	**状況**	**Use**
LR	volatile	ブランチ・ターゲット・アドレスとして使用されるか、リターン・アドレスを保持します。
CTR	volatile	ループ・カウントの減分および分枝に使用されます。
XER	volatile	固定小数点例外レジスター。
FPSCR	volatile	浮動小数点例外レジスター。
CR0, CR1	volatile	条件レジスター・ビット。
CR2, CR3, CR4	不揮発性	条件レジスター・ビット。
CR5, CR6, CR7	volatile	条件レジスター・ビット。

CR2、 CR3、および CR4 を変更するルーチンは、少なくとも CR のこれらのフィールドを保管および復元する必要があります。 その他の CR フィールドを使用する場合は、保存や復元を行う必要はありません。

ランタイム・プロセス・スタック

スタック・フォーマット規則は、Prolog および epilog 関数の使用、パラメーターの引き渡し、および共有ライブラリー・サポートの効率を向上させるように設計されています。

スタック・フォーマット規則は、以下のものの効率を高めるように設計されています。

- プロローグおよびエピローグ関数の使用法
- パラメーター引き渡し
- 共有ライブラリー・サポート

ランタイム・スタックの図は、ランタイム・スタックを示しています。 これは、**sender** 関数が **catcher** 関数を呼び出した後、 **catcher** 関数が別の関数を呼び出す前のスタックを示します。 この図は、 **catcher** 関数が別の関数を呼び出すという前提に基づいています。 したがって、 **catcher** 関数は、(スタック・レイアウトで説明されているように) 別のリンク域を必要とします。 **PW***n* は、渡されるパラメーターの *n* 番目のワードを指します。

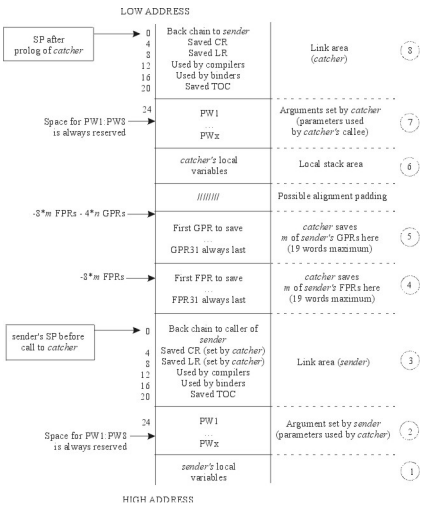

図 2. ランタイム・スタック

スタック・レイアウト
スタック・レイアウトは、数値的に上位のストレージ・アドレスから数値的に下位のアドレスに拡張されます。

スタックのアドレッシングには、スタック・ポインター (SP) と呼ばれる 1 つのレジスターのみが使用され、 GPR1 は専用のスタック・ポインター・レジスターです。 これは、数値的に高いストレージ・アドレスから、数値的に低いアドレスに増加します。

ランタイム・スタックの図は、 **sender** 関数が **catcher** 関数を呼び出したときに何が起こるか、および **catcher** 関数がそれ自体のスタック・フレームをどのように必要とするかを示しています。 関数が呼び出しを行わず、それ自体のローカル・ストレージを必要としない場合、スタック・フレームは不要であり、SP は変更されません。

注 :

1. 混乱を減らすために、 **sender** 関数 (呼び出し元) から渡されるデータは引数として参照され、 **キャッチャー** 関数 (呼び出し先) が受け取るデータと同じデータはパラメーターとして参照されます。 **sender** の出力引数域は、 **catcher** の入力パラメーター域と同じになります。
2. スタック・ポインター内のアドレス値は、4 分位に位置合わせされている必要があります。 (アドレス値は 16 の倍数でなければなりません。)

スタック域
スタック・レイアウトは、図の下部から図の上部まで、1 から 8 までの番号が付いた 8 つの領域に分割されます。

便宜上、スタック・レイアウトは、図の下部 (高位アドレス) から図の上部 (低位アドレス) まで、1 から 8 までの番号が付いた 8 つの領域に分割されています。 **catcher** 関数への呼び出しが行われると、送信側のスタック・ポインターは領域 3 の最上部を指します。これは、 **catcher** 関数がそのプロローグへの入り口で使用するのと同じ SP 値です。 以下に、スタック域の説明を示します。この説明は、図の下部 (領域 1) から始まり、上部 (領域 8) に向かって進みます。

- **領域 1: 送信側のローカル変数域**

  領域 1 は **sender** 関数のローカル変数域であり、この関数に必要なすべてのローカル変数と一時スペースを含んでいます。

- **領域 2: 送信側の出力引数領域**

  領域 2 は、 **sender** 関数の出力引数領域です。 この領域は 8 ワード以上のサイズで、ダブルワードに位置合わせする必要があります。 最初の 8 ワードは、呼び出し側 (**sender** 関数) によって使用されません。これは、対応する値が引数レジスター (GPR3:GPR10) に直接入れられるためです。 ストレージは、呼び出し先 (**catcher** 関数) がそのパラメーターのいずれかのアドレスを取る場合に、 GPR3:GPR10 に渡される値をそれぞれのアドレス・ロケーション (PW1:PW8) に保管できるように予約されています。 **sender** 関数が **catcher** 関数に 8 個を超える引数を渡している場合、余分のパラメーター用にスペースを予約する必要があります。 余分のパラメーターは、 **sender** 関数の SP 値からオフセット 56 から始まる 8 個の予約語を超えて、レジスター・イメージとして保管する必要があります。

  注 : この領域は、言語処理プログラムによって使用されることもあり、他の関数の呼び出し間で揮発性があります。

- **領域 3: 送信側のリンク域**

  領域 3 は、 **sender** 関数のリンク域です。 この領域は 6 ワードで構成され、 **catcher** 関数への呼び出しが行われた時点で、 **sender** 関数の SP からオフセット 0 にあります。 この領域の特定のフィールドは、 **catcher** 関数によってそのプロローグ・コードの一部として使用されます。これらのフィールドは、ランタイム・スタックの図でマークされ、以下で説明します。

  最初のワードはバックチェーンです。これは、SP を変更する前に **sender** 関数が呼び出し元の SP 値を保管した場所です。 2 番目のワード (オフセット 4) は、 **catcher** 関数が不揮発性 CR フィールドのいずれかを変更した場合に CR を保存できる場所です。 3 番目のワード (オフセット 8) は、 **catcher** 関数が何らかの呼び出しを行った場合に **catcher** 関数が LR を保存できる場所です。

  4 番目のワードはコンパイラー用に予約されており、5 番目のワードはバインダー生成命令によって使用されます。 リンク域 (オフセット 20) の最後のワードは、TOC 域レジスターがグローバル・リンケージ

(glink) インターフェース・ルーチンによって保管される場所です。 これは、共有ライブラリー関数が呼び出されるときなど、モジュール外呼び出しが実行されるときに発生します。

- **領域 4: キャッチャーの浮動小数点レジスター保管域**

  区域 4 は、呼び出し先 (**キャッチャー** 関数) の浮動小数点レジスター保管域であり、ダブルワード位置合わせされます。 これは、呼び出し先プログラム (**キャッチャー** 機能) によって使用されるすべての不揮発性 FPR を保管するために必要なスペースを表します。 FPR は、 **送信側** 機能の SP からの負の変位で、リンク域のすぐ上 (下位アドレス) に保管されます。 この領域のサイズは、保管される FPR の数 (最大数はそれぞれ 18 FPR * 8 バイト) に応じて、ゼロから最大 144 バイトまで変化します。

- **領域 5: キャッチャーの汎用レジスターの保管域**

  領域 5 は、 **catcher** 関数用の汎用レジスター保管域であり、少なくともワードで位置合わせされています。 これは、すべての不揮発性 GPR を保管するために呼び出し先プログラム (**キャッチャー** 機能) が必要とするスペースを表します。 GPR は、 **送信側** 機能の SP からの負の変位で、FPR 保管域のすぐ上 (下位アドレス) に保管されます。 この領域のサイズは、保管される GPR の数に応じて、ゼロから最大 76 バイトまで変化します (最大数はそれぞれ 19 GPR * 4 バイトです)。

  注 :

  1. スタックなしリーフ・プロシージャーは、呼び出しを行わず、ローカル変数域を必要としませんが、不揮発性 GPR および FPR を使用することができます。
  2. 保管域は、FPR 保管域 (4) と GPR 保管域 (5) で構成されます。これらの保管域の合計最大サイズは 220 バイトです。 現在実行中の関数のスタック・フロアは、SP の値より 220 バイト小さい位置にあります。 SP の値とスタック・フロアの間の領域は、スタックなしリーフ関数が独自のスタックを取得せずに使用できる最大保管域です。 関数は、この領域を一時スペースとして使用することができます。この一時スペースは、他の関数の呼び出し間で揮発性です。 割り込みハンドラーやバインダー挿入コードなどの実行エレメントは、コンパイルされたコードでは呼び出しとして認識できないため、この領域を使用してはなりません。

  システム定義のスタック・フロアには、可能な最大の保管域が含まれます。 保管域のサイズの公式は、次のとおりです。

```
18*8
(for FPRs)
 + 19*4
(for GPRs)
 = 220
```

- **領域 6: キャッチャーのローカル変数域**

  領域 6 は **catcher** 関数のローカル変数域であり、この関数に必要なローカル変数と一時スペースが含まれています。 **キャッチャー** 機能は、領域 8 の先頭を指す独自の SP を基底レジスターとして使用して、この領域をアドレッシングします。

- **領域 7: キャッチャーの出力引数領域**

  領域 7 は、 **catcher** 関数の出力引数領域であり、サイズが 8 ワード以上で、ダブルワードで位置合わせされている必要があります。 最初の 8 ワードは、対応する値が引数レジスター (GPR3:GPR10) に直接入れられるため、呼び出し側 (**キャッチャー** 関数) では使用されません。 ストレージは、 **catcher** 関数の呼び出し先がそのパラメーターのいずれかのアドレスを取る場合に、 GPR3:GPR10 に渡される値をそのアドレス位置に保管できるように予約されています。 **catcher** 関数が 8 個を超える引数をその呼び出し先 (それぞれ PW1:PW8) に渡す場合、余分なパラメーター用のスペースを予約する必要があります。 余分のパラメーターは、 **catcher** 関数の SP 値からオフセット 56 から始まる 8 つの予約語を超えるレジスター・イメージとして保管する必要があります。

  注 : この領域は、言語処理プログラムによって使用することもでき、他の関数の呼び出しにまたがって揮発性があります。

- **領域 8: キャッチャーのリンク域**

  領域 8 は **キャッチャー** 機能のリンク域であり、**センダー** 機能のリンク域 (領域 3) と同じフィールドが含まれています。

スタック関連システム標準

すべての言語処理プログラムおよびアセンブラーは、SP が単一の命令によってアトミックに更新されなければならないスタック関連システム標準を維持する必要があります。 これにより、結果としてスタック・ポインターが部分的にしか更新されないような割り込みが発生するタイミング・ウィンドウがなくなります。

注 : プログラム・プロローグおよびエピローグの例は、スタック・ポインターを更新する最も効率的な方法を示しています。

プロローグとエピローグ

関数の入り口でのレジスターの設定や関数の出口でのレジスターの復元など、関数にはプロローグとエピローグが使用されます。

関数の入り口でのレジスターの設定や関数の出口でのレジスターの復元など、プロローグおよびエピローグを関数に使用することができます。

関数プロローグおよびエピローグを表す事前定義されたコード・シーケンスは指示されません。 ただし、特定の操作は特定の条件下で実行する必要があります。 次の図は、スタック・フレームのレイアウトを示しています。

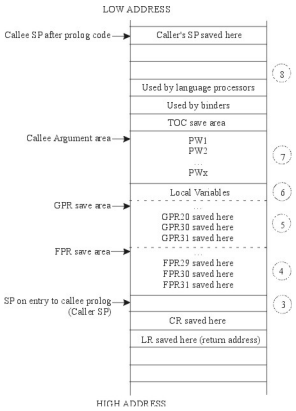

図 *3.* スタック・フレーム・レイアウト

標準的な関数の実行スタックは以下のとおりです。

- プロローグ・アクション
- 関数の本体
- Epilog アクション

「プロローグ・アクション」表と「エピローグ・アクション」表には、プロローグとエピローグに必要な条件とアクションが表示されます。

表 *5.* プロローグ・アクション

以下の場合:	**次に、以下の情報を指定します。**
不揮発性 FPR (FPR14:FPR31) が使用されます。	それらを FPR 保管域 (前の図の区域 4) に保管します。
不揮発性 GPR (GPR13:GPR31) が使用されます。	それらを GPR 保管域 (前の図の領域 5) に保管します。
LR は非リーフ・プロシージャーに使用されます。	呼び出し側機能 SP からのオフセット 8 に LR を保管します。
不揮発性条件レジスター (CR) フィールドのいずれかが使用されます。	呼び出し側機能 SP からのオフセット 4 に CR を保管します。

表 5. プロローグ・アクション (続き)	
以下の場合:	**次に、以下の情報を指定します。**
新規スタック・フレームが必要です	スタック・フレームを取得し、(必要に応じて) 16 の倍数に埋め込まれたフレームのサイズ分だけ SP を減分して、新規 SP を獲得し、新規 SP からのオフセット 0 に呼び出し元の SP を保存します。

注 : ローカル変数および一時変数用のスタック・スペースを必要としないリーフ関数は、実際にスタック・フレームを獲得することなく、呼び出し元 SP からの負のオフセットで呼び出し元レジスターを保管することができます。

表 6. エピローグ・アクション	
以下の場合:	**次に、以下の情報を指定します。**
不揮発性 FPR が保管されました。	使用された FPR を復元します。
不揮発性 GPR が保管された	保管された GPR を復元します。
非リーフ・プロシージャーが呼び出されたため、LR が変更されました。	LR を復元します。
CR が変更された	CR を復元します。
新規スタックが取得されました	古い SP を入り口での値 (呼び出し側の SP) に復元します。 呼び出し元に戻ります。

PowerPC ® アーキテクチャーは、GPR に対してロードと保管の両方の命令を提供しますが、一部のマシンでの実装が最適ではない可能性があるため、使用を推奨しません。 実際、将来のいくつかの実装でロードおよび保管の複数の命令を使用すると、同等の一連の単一ワード・ロードまたはストアよりも大幅に遅くなる可能性があります。 ただし、単一のロード命令または保管命令を使用して、多くの FPR または GPR を関数プロローグまたはエピローグに保管すると、コード・サイズが大きくなります。 このため、システム環境は、FPR および GPR の保管と復元を行う関数プロローグおよびエピローグから呼び出すことができるルーチンを提供する必要があります。 これらのルーチンへのインターフェース、そのソース・コード、およびいくつかのプロローグ・コードとエピローグ・コードのシーケンスが提供されています。

スタック・フレーム・レイアウトに示されているように、GPR 保管域は、呼び出し側 SP または呼び出し先 SP のいずれからも固定位置にありません。 FPR 保管域は、その呼び出し先への入り口で SP (下位アドレス) のすぐ上の固定位置から始まりますが、GPR 保管域の位置は、保管される FPR の数によって異なります。 そのため、SP からの固定変位を使用する汎用 GPR 保存機能を作成することは困難です。

ルーチンが GPR と FPR の両方を保管する必要がある場合は、GPR を保管および復元するためのポインターとして GPR12 を使用します。 (GPR12 は揮発性レジスターですが、入力パラメーターは含まれていません。) その結果、複数レジスターの保管および復元ルーチンが定義され、それぞれが *m* 個の FPR と *n* 個の GPR を保管または復元します。 これは、最低不揮発性レジスターから始めて、複数のエントリー・ポイント (レジスター番号ごとに 1 つ) を含む特別に提供されたルーチンに対して bla (ブランチおよびリンク絶対) 命令を実行することによって達成されます。

注 :

1. 29 より大きい GPR および FPR 番号を保管および復元するためのエントリー・ポイントはありません。保管機能および復元機能を呼び出すよりも、少数のレジスターをプロローグに保管する方が効率的です。
2. LR が以下のコード・セグメントで保管または復元されない場合、言語処理プログラムは必要に応じて保管および復元を実行する必要があります。

言語処理プログラムは、関数呼び出し全体で不揮発性レジスターの値を節約するために、専有メソッドを使用する必要があります。

システム環境では、保管および復元ルーチンの 3 つのセットを使用可能にする必要があります。 これらのルーチンは次のとおりです。

- FPR が保管および復元されない場合に、GPR を保管および復元するルーチンのペア。
- FPR が保管および復元されるときに、GPR を保管および復元するルーチンのペア。
- FPR を保管および復元するためのルーチンのペア。

GPRS のみを保存しています
FPR を使用せずに n 個の GPR を保管および復元する機能の場合、個々の保管およびロード命令を使用して保管を行うことができます。

n 個の GPR を保管および復元し、FPR を復元しない機能の場合は、個々の保管およびロード命令を使用するか、以下の例に示すようにシステム提供のルーチンを呼び出すことによって、保管を行うことができます。

注 : 保管されるレジスターの数は *n* です。 以下の例の <32-n> のようなシーケンスは、保管および復元される最初のレジスター番号を示しています。 <32-n> から 31 までのすべてのレジスターが保管され、復元されます。

```
#Following are the prolog/epilog of a function that saves n GPRS #(n>2):
mflr    r0                          #move LR into GPR0
bla     _savegpr0_<32-n>            #branch and link to save GPRs
stwu    r1,<-frame_size>(r1)        #update SP and save caller's SP
...                                 #frame_size is the size of the
                                    #stack frame to be required
<save CR if necessary>
...
...                                 #body of function
...
<reload save CR if necessary>
...
<reload caller's SP into R!>        #see note below
ba      _restgpr0_<32-n>            #restore GPRs and return
```

注 : 呼び出し機能 SP の復元は、 `frame_size` が認識されるたびに `frame_size` 値を現行 SP に追加するか、現行 SP からオフセット 0 から再ロードすることによって行うことができます。 最初の方法は、より効率的ですが、 **alloca** サブルーチンを使用してスタック・スペースを動的に割り当てる関数では使用できません。

以下の例は、FPR が保管されない場合の GPR 保管ルーチンを示しています。

```
_savegpr0_13    stw     r13,-76(r1)             #save r13
_savegpr0_14    stw     r14,-72(r1)             #save r14
_savegpr0_15    stw     r15,-68(r1)             #save r15
_savegpr0_16    stw     r16,-64(r1)             #save r16
_savegpr0_17    stw     r17,-60(r1)             #save r17
_savegpr0_18    stw     r18,-56(r1)             #save r18
_savegpr0_19    stw     r19,-52(r1)             #save r19
_savegpr0_20    stw     r20,-48(r1)             #save r20
_savegpr0_21    stw     r21,-44(r1)             #save r21
_savegpr0_22    stw     r22,-40(r1)             #save r22
_savegpr0_23    stw     r23,-36(r1)             #save r23
_savegpr0_24    stw     r24,-32(r1)             #save r24
_savegpr0_25    stw     r25,-28(r1)             #save r25
_savegpr0_26    stw     r26,-24(r1)             #save r26
_savegpr0_27    stw     r27,-20(r1)             #save r27
_savegpr0_28    stw     r28,-16(r1)             #save r28
_savegpr0_29    stw     r29,-12(r1)             #save r29
                stw     r30,-8(r1)              #save r30
                stw     r31,-4(r1)              #save r31
                stw     r0 , 8(r1)              #save LR in
                                                #caller's frame
                blr                             #return
```

注 : GPR30 または GPR31、あるいはその両方が保管された唯一のレジスターである場合は、この保管ルーチンを呼び出してはなりません。 このような場合、保存と復元はインラインで行う必要があります。

以下の例は、FPR が保管されない場合の GPR 復元ルーチンを示しています。

```
_restgpr0_13    lwz     r13,-76(r1)             #restore r13
_restgpr0_14    lwz     r14,-72(r1)             #restore r14
```

```
_restgpr0_15    lwz     r15,-68(r1)             #restore r15
_restgpr0_16    lwz     r16,-64(r1)             #restore r16
_restgpr0_17    lwz     r17,-60(r1)             #restore r17
_restgpr0_18    lwz     r18,-56(r1)             #restore r18
_restgpr0_19    lwz     r19,-52(r1)             #restore r19
_restgpr0_20    lwz     r20,-48(r1)             #restore r20
_restgpr0_21    lwz     r21,-44(r1)             #restore r21
_restgpr0_22    lwz     r22,-40(r1)             #restore r22
_restgpr0_23    lwz     r23,-36(r1)             #restore r23
_restgpr0_24    lwz     r24,-32(r1)             #restore r24
_restgpr0_25    lwz     r25,-28(r1)             #restore r25
_restgpr0_26    lwz     r26,-24(r1)             #restore r26
_restgpr0_27    lwz     r27,-20(r1)             #restore r27
_restgpr0_28    lwz     r28,-16(r1)             #restore r28
_restgpr0_29    lwz     r0,8(r1)                #get return
                                                #address from
                                                #frame
                lwz     r29,-12(r1)             #restore r29
                mtlr    r0                      #move return
                                                #address to LR
                lwz     r30,-8(r1)              #restore r30
                lwz     r31,-4(r1)              #restore r31
                blr                             #return
```

注 : GPR30 または GPR31、あるいはその両方が保管されている唯一のレジスターである場合は、この復元ルーチンを呼び出してはなりません。 このような場合、保存と復元はインラインで行う必要があります。

gprs および *fprs* の保存
n 個の GPR および *m* 個の FPR (*n*> 2 および *m*> 2) を保管および復元する機能の場合、保管は、個々の保管およびロード命令を使用するか、システム提供ルーチンを呼び出すことによって行うことができます。

n 個の GPR および *m* 個の FPR (*n*> 2 および *m*> 2) を保管および復元する機能の場合、個々の保管およびロード命令を使用するか、以下の例に示すようにシステム提供ルーチンを呼び出すことによって、保管を行うことができます。

```
#The following example shows the prolog/epilog of a function #which save n GPRs and m FPRs:
mflr    r0                      #move LR into GPR 0
subi    r12,r1,8*m              #compute GPR save pointer
bla     _savegpr1_<32-n>        #branch and link to save GPRs
bla     _savefpr_<32-m>
stwu    r1,<-frame_size>(r1)    #update SP and save caller's SP
...
<save CR if necessary>
...
...                             #body of function
...
<reload save CR if necessary>
...
<reload caller's SP into r1>    #see note below on
subi    r12,r1,8*m              #compute CPR restore pointer
bla     _restgpr1_<32-n>        #restore GPRs
ba      _restfpr_<32-m>         #restore FPRs and return
```

注 : 呼び出し機能 SP は、 `frame_size` が認識されるたびに `frame_size` 値を現行 SP に追加するか、現行 SP からオフセット 0 から再ロードすることによって復元できます。 最初の方法は、より効率的ですが、**alloca** サブルーチンを使用してスタック・スペースを動的に割り当てる関数では使用できません。

以下の例は、FPR が保管されるときの GPR 保管ルーチンを示しています。

```
_savegpr1_13    stw     r13,-76(r12)            #save r13
_savegpr1_14    stw     r14,-72(r12)            #save r14
_savegpr1_15    stw     r15,-68(r12)            #save r15
_savegpr1_16    stw     r16,-64(r12)            #save r16
_savegpr1_17    stw     r17,-60(r12)            #save r17
_savegpr1_18    stw     r18,-56(r12)            #save r18
_savegpr1_19    stw     r19,-52(r12)            #save r19
_savegpr1_20    stw     r20,-48(r12)            #save r20
_savegpr1_21    stw     r21,-44(r12)            #save r21
_savegpr1_22    stw     r22,-40(r12)            #save r22
_savegpr1_23    stw     r23,-36(r12)            #save r23
_savegpr1_24    stw     r24,-32(r12)            #save r24
_savegpr1_25    stw     r25,-28(r12)            #save r25
_savegpr1_26    stw     r26,-24(r12)            #save r26
_savegpr1_27    stw     r27,-20(r12)            #save r27
```

```
_savegpr1_28    stw     r28,-16(r12)            #save r28
_savegpr1_29    stw     r29,-12(r12)            #save r29
                stw     r30,-8(r12)             #save r30
                stw     r31,-4(r12)             #save r31
                blr                             #return
```

以下の例は、FPR 保管ルーチンを示しています。

```
_savefpr_14    stfd     f14,-144(r1)            #save f14
_savefpr_15    stfd     f15,-136(r1)            #save f15
_savefpr_16    stfd     f16,-128(r1)            #save f16
_savefpr_17    stfd     f17,-120(r1)            #save f17
_savefpr_18    stfd     f18,-112(r1)            #save f18
_savefpr_19    stfd     f19,-104(r1)            #save f19
_savefpr_20    stfd     f20,-96(r1)             #save f20
_savefpr_21    stfd     f21,-88(r1)             #save f21
_savefpr_22    stfd     f22,-80(r1)             #save f22
_savefpr_23    stfd     f23,-72(r1)             #save f23
_savefpr_24    stfd     f24,-64(r1)             #save f24
_savefpr_25    stfd     f25,-56(r1)             #save f25
_savefpr_26    stfd     f26,-48(r1)             #save f26
_savefpr_27    stfd     f27,-40(r1)             #save f27
_savefpr_28    stfd     f28,-32(r1)             #save f28
_savefpr_29    stfd     f29,-24(r1)             #save f29
               stfd     f30,-16(r1)             #save f30
               stfd     f31,-8(r1)              #save f31
               stw      r0 , 8(r1)              #save LR in
                                                #caller's frame
               blr                              #return
```

以下の例は、FPR が保管されるときの GPR リストア・ルーチンを示しています。

```
_restgpr1_13    lwz     r13,-76(r12)            #restore r13
_restgpr1_14    lwz     r14,-72(r12)            #restore r14
_restgpr1_15    lwz     r15,-68(r12)            #restore r15
_restgpr1_16    lwz     r16,-64(r12)            #restore r16
_restgpr1_17    lwz     r17,-60(r12)            #restore r17
_restgpr1_18    lwz     r18,-56(r12)            #restore r18
_restgpr1_19    lwz     r19,-52(r12)            #restore r19
_restgpr1_20    lwz     r20,-48(r12)            #restore r20
_restgpr1_21    lwz     r21,-44(r12)            #restore r21
_restgpr1_22    lwz     r22,-40(r12)            #restore r22
_restgpr1_23    lwz     r23,-36(r12)            #restore r23
_restgpr1_24    lwz     r24,-32(r12)            #restore r24
_restgpr1_25    lwz     r25,-28(r12)            #restore r25
_restgpr1_26    lwz     r26,-24(r12)            #restore r26
_restgpr1_27    lwz     r27,-20(r12)            #restore r27
_restgpr1_28    lwz     r28,-16(r12)            #restore r28
_restgpr1_29    lwz     r29,-12(r12)            #restore r29
                lwz     r30,-8(r12)             #restore r30
                lwz     r31,-4(r12)             #restore r31
                blr                             #return
```

以下の例は、FPR 復元ルーチンを示しています。

```
_restfpr_14    lfd     r14,-144(r1)            #restore r14
_restfpr_15    lfd     r15,-136(r1)            #restore r15
_restfpr_16    lfd     r16,-128(r1)            #restore r16
_restfpr_17    lfd     r17,-120(r1)            #restore r17
_restfpr_18    lfd     r18,-112(r1)            #restore r18
_restfpr_19    lfd     r19,-104(r1)            #restore r19
_restfpr_20    lfd     r20,-96(r1)             #restore r20
_restfpr_21    lfd     r21,-88(r1)             #restore r21
_restfpr_22    lfd     r22,-80(r1)             #restore r22
_restfpr_23    lfd     r23,-72(r1)             #restore r23
_restfpr_24    lfd     r24,-64(r1)             #restore r24
_restfpr_25    lfd     r25,-56(r1)             #restore r25
_restfpr_26    lfd     r26,-48(r1)             #restore r26
_restfpr_27    lfd     r27,-40(r1)             #restore r27
_restfpr_28    lfd     r28,-32(r1)             #restore r28
_restfpr_29    lwz     r0,8(r1)                #get return
                                               #address from
                                               #frame
               lfd     r29,-24(r1)             #restore r29
               mtlr    r0                      #move return
```

```
                                        #address to LR
        lfd     r30,-16(r1)             #restore r30
        lfd     r31,-8(r1)              #restore r31
        blr                             #return
```

fprs のみを保存しています
m FPR (*m*> 2) を保管および復元する機能の場合、個々の保管およびロード命令を使用するか、システム提供ルーチンを呼び出すことによって、保管を行うことができます。

m FPR (*m*> 2) を保管および復元する機能の場合、個々の保管およびロード命令を使用するか、以下の例に示すようにシステム提供のルーチンを呼び出すことによって、保管を行うことができます。

```
#The following example shows the prolog/epilog of a function #which saves m FPRs and no GPRs:
mflr    r0                      #move LR into GPR 0
bla     _savefpr_<32-m>
stwu    r1,<-frame_size>(r1)    #update SP and save caller's SP
...
<save CR if necessary>
...
...                             #body of function
...
<reload save CR if necessary>
...
<reload caller's SP into r1>    #see note below
ba      _restfpr_<32-m>         #restore FPRs and return
```

注 :

1. 29 より大きい GPR および FPR 番号を保管および復元するためのエントリー・ポイントはありません。保管および復元機能を呼び出すよりも、少数のレジスターをプロローグに保管する方が効率的です。
2. 呼び出し機能 SP の復元は、 `frame_size` が認識されるたびに `frame_size` 値を現行 SP に追加するか、現行 SP からオフセット 0 から再ロードすることによって行うことができます。 最初の方法は、より効率的ですが、 **alloca** サブルーチンを使用してスタック・スペースを動的に割り当てる関数では使用できません。

スタック・ポインターの更新
PowerPC ® **stwu** (更新付きストア・ワード) 命令は、新しい SP を計算してバック・チェーンを保存するために使用されます。

PowerPC ® **stwu** (Store Word with Update) 命令は、新しい SP を計算してバックチェーンを保存するために使用されます。 この命令には、最大符号付き値 32,768 を表すことができる符号付き 16 ビット変位フィールドがあります。 32K バイトより大きいスタック・フレーム・サイズは、SP を更新するために 2 つの命令を必要とし、更新はアトミックに行う必要があります。

2 つのアセンブリー・コード例は、プロローグ内の SP を更新する方法を示しています。

新しい SP を計算し、 32K バイト以上のスタック・フレームの古い SP を保管するには、次のようにします。

```
addis   r12, r0, (<-frame_size> > 16) & 0XFFFF
                        # set r12 to left half of frame size
ori     r12, r12 (-frame_size> & 0XFFFF
                        # Add right halfword of frame size
stwux   r1, r1, r12     # save old SP and compute new SP
```

新しい SP を計算し、 32K バイトより小さいスタック・フレーム用に古い SP を保管するには、次のようにします。

```
stwu   r1, <-frame_size>(r1)   #update SP and save caller's SP
```

呼び出しルーチンの責任

アセンブラー言語プログラムが別のプログラムを呼び出す場合、呼び出し元は、呼び出し先プログラムのコマンド、関数、またはプロシージャーの名前をグローバル・アセンブラー言語シンボルとして使用して

はなりません。 混乱を避けるために、シンボル名を作成するときは、呼び出し先プログラムの言語の命名規則に従ってください。 例えば、C 言語プログラムを呼び出す場合は、その言語の命名規則を使用するようにしてください。

呼び出し先ルーチンには、関数記述子 (*Name*) とエントリー・ポイント (*.*) の *2* つのシンボルが関連付けられています。名前)。 ルーチンに対して呼び出しが行われると、コンパイラーはエントリー・ポイントに直接分岐します。

関数の呼び出しはコンパイラーによって拡張され、各分岐命令とリンク命令の後に NOP 命令が組み込まれます。 必要であれば、この追加命令は、モジュール外呼び出しからの戻り時に TOC レジスター (レジスター 2) の内容を復元するために、リンケージ・エディターによって変更されます。

コンパイラーによって作成される命令シーケンスは、次のとおりです。

```
bl .foo           #Branch to foo
nop
```

注 : アセンブラーは、 **nop** ニーモニックに対して **ori 0,0,0** (0x60000000) 命令を生成します。 後方互換性のために、リンケージ・エディターは **cror 15,15,15** (0x4def7b82) および **cror 31,31,31** (0x4ffffb82) も有効な NOP 命令として受け入れます。

リンケージ・エディターは、 **bl** 命令を (前の命令シーケンスで、 **foo** 関数の呼び出しで) 検出すると、次の 2 つのいずれかを行います。

- **foo** 関数がインポートされると (同じ実行可能モジュールにはインポートされません)、リンケージ・エディターは次のことを行います。
  - **bl .foo** 命令を **bl .glink_of_foo** (グローバル・リンケージ・ルーチン) に変更します。
  - **.glink** コード・シーケンスを (**/usr/lib/glink.o** ファイル) モジュールに挿入します。
  - TOC レジスターを復元するために、 NOP 命令を lwz (ロード) 命令に置き換えます。

  bl .foo 命令シーケンスは、以下のように変更されます。

  ```
  bl .glink_of_foo  #Branch to global linkage routine for foo
  lwz 2,20(1)       #Restore TOC register instruction 0x80410014
  ```

- **foo** 関数がその後、その呼び出し元と同じ実行可能モジュールにバインドされると、リンケージ・エディターは以下を行います。
  - **bl .glink_of_foo** シーケンス (グローバル・リンケージ・ルーチン) を **bl .foo** に変更します。
  - TOC レジスター復元命令を NOP 命令に置き換えます。

  bl .glink_of_foo 命令シーケンスは、以下のように変更されます。

  ```
  bl .foo         #Branch to foo
  nop
  ```

注 : どのエクスポートの場合も、リンケージ・エディターはプロシージャーの記述子をモジュールに挿入します。

呼び出されるルーチンの責任

プロローグおよびエピローグは、呼び出されたルーチンで使用されます。

プロローグおよびエピローグは、呼び出されたルーチンで使用されます。 ルーチンへの入り口では、以下のステップを実行する必要があります。

1. 「プロローグ・アクション」 表に記載されているプロローグ・アクションの一部またはすべてを使用します。
2. バック・チェーンを保管し、スタック・フレームのサイズだけスタック・ポインター (SP) を減らします。

   注 : スタック・オーバーフローが発生した場合、バック・チェーンの保管が完了するとすぐに認識されます。

プロシージャーの終了時に、「エピローグ・アクション」 表に記載されているエピローグ・アクションの一部またはすべてを使用します。

トレースバック・タグ

アセンブリー (コンパイル済み) プログラムは、実行中にプログラムがトラップまたは異常終了するかどうかを検査するために、デバッガーのトレースバック情報を必要とします。

すべてのアセンブリー (コンパイル済み) プログラムには、実行中にプログラムがトラップまたは異常終了するかどうかをデバッガーが検査するためのトレースバック情報が必要です。 この情報は、プログラム内の最後のマシン・インストラクションの終わり、およびプログラムの定数データの前にあるトレースバック・テーブルにあります。

トレースバック・テーブルがゼロのフルワード X'00000000 ' で始まっていますが、これは有効なシステム命令ではありません。 ゼロの後には、**/usr/include/sys/debug.h** ファイルに定義されているように、必須情報の 2 ワード (64 ビット) とオプション情報のいくつかのワードが続きます。 デバッガーは、このトレースバック情報を使用して、CALL チェーンをアンワインドし、障害が発生したポイントからプログラムの終わり (ゼロのワード) に達するまで順方向に検索することができます。

一般に、トレースバック情報には、ソース言語の名前と、プログラムによって使用されるレジスターに関する情報 (汎用レジスターや浮動小数点レジスターの保管など) が含まれます。

例

C サブルーチンによって呼び出されるアセンブラー・コードの例。

以下は、C ルーチンによって呼び出されるアセンブラー・コードの例です。

```
#      Call this assembly routine from C routine:
#       callfile.c:
#       main()
#       {
#       examlinkage();
#       }
#       Compile as follows:
#       cc -o callfile callfile.c examlinkage.s
#
###############################################################################
#       On entry to a procedure(callee), all or some of the
#       following steps should be done:
#       1.  Save the link register at offset 8 from the
#           stack pointer for non-leaf procedures.
#       2.  If any of the CR bits 8-19(CR2,CR3,CR4) is used
#           then save the CR at displacement 4 of the current
#            stack pointer.
#       3.  Save all non-volatile FPRs used by this routine.
#           If more that three non-volatile FPR are saved,
#            a call to ._savefn can be used to
#           save them (n is the number of the first FPR to be
#            saved).
#       4.  Save all non-volatile GPRs used by this routine
#           in the caller's GPR SAVE area (negative displacement
#           from the current stack pointer r1).
#       5.  Store back chain and decrement stack pointer by the
#           size of the stack frame.
#
#       On exit from a procedure (callee), all or some of the
#       following steps should be done:
#       1.  Restore all GPRs saved.
#       2.  Restore stack pointer to value it had on entry.
#       3.  Restore Link Register if this is a non-leaf
#           procedure.
#       4.  Restore bits 20-31 of the CR is it was saved.
#       5.  Restore all FPRs saved.  If any FPRs were saved then
#           a call to ._savefn can be used to restore them
#           (n is the first FPR to be restored).
#       6.  Return to caller.
###############################################################################
#       The following routine calls printf() to print a string.
#       The routine performs entry steps 1-5 and exit steps 1-6.
#       The prolog/epilog code is for small stack frame size.
#       DSA + 8 < 32k
###############################################################################
```

```
        .file   "examlinkage.s"
#Static data entry in T(able)O(f)C(ontents)
        .toc
T.examlinkage.c:        .tc     examlinkage.c[tc],examlinkage.c[rw]
        .globl  examlinkage[ds]
#examlinkage[ds] contains definitions needed for
#runtime linkage of function examlinkage
        .csect  examlinkage[ds]
        .long   .examlinkage[PR]
        .long   TOC[tc0]
        .long   0
#Function entry in T(able)O(f)C(ontents)
        .toc
T.examlinkage:  .tc     .examlinkage[tc],examlinkage[ds]
#Main routine
        .globl  .examlinkage[PR]
        .csect  .examlinkage[PR]
#       Set current routine stack variables
#       These values are specific to the current routine and
#       can vary from routine to routine
        .set    argarea,     32
        .set    linkarea,    24
       .set    locstckarea,   0
        .set    nfprs,       18
        .set    ngprs,       19
        .set    szdsa,
8*nfprs+4*ngprs+linkarea+argarea+locstckarea
#PROLOG:  Called Routines Responsibilities
        #    Get link reg.
        mflr    0
        #    Get CR if current routine alters it.
        mfcr    12
        #    Save FPRs 14-31.
        bl      ._savef14
        cror 31, 31, 31
        #    Save GPRs 13-31.
        stm     13, -8*nfprs-4*ngprs(1)
        #    Save LR if non-leaf routine.
        st      0, 8(1)
        #    Save CR if current routine alters it.
        st      12, 4(1)
        #    Decrement stack ptr and save back chain.
        stu     1, -szdsa(1)
#####################################
#load static data address
#####################################
        l       14,T.examlinkage.c(2)
        #  Load string address which is an argument to printf.
        cal 3, printing(14)
        #  Call to printf routine
        bl      .printf[PR]
        cror 31, 31, 31
#EPILOG:  Return Sequence
        #    Restore stack ptr
        ai      1, 1, szdsa
        #    Restore GPRs 13-31.
        lm      13, -8*nfprs-4*ngprs(1)
        #    Restore FPRs 14-31.
        bl      ._restf14
        cror 31, 31, 31
    #    Get saved LR.
        l       0, 8(1)
        #    Get saved CR if this routine saved it.
        l       12, 4(1)
        #    Move return address to link register.
        mtlr    0
        #    Restore CR2, CR3, & CR4 of the CR.
        mtcrf   0x38,12
        #    Return to address held in Link Register.
        brl
        .tbtag 0x0,0xc,0x0,0x0,0x0,0x0,0x0,0x0
          #       External variables
        .extern ._savef14
        .extern ._restf14
        .extern .printf[PR]
#####################################
#       Data
#####################################
        .csect  examlinkage.c[rw]
        .align  2
printing:       .byte  'E,'x,'a,'m,'p,'l,'e,' ,'f,'o,'r,'
```

```
                .byte   'P,'R,'I,'N,'T,'I,'N,'G
        .byte   0xa,0x0
```

ミリコード・ルーチンの使用

ミリコード・ルーチンには、マシン依存の機能とパフォーマンス上重要な機能が含まれています。

POWER ® ファミリーと PowerPC ® では、すべての固定小数点除算命令、およびいくつかの乗算命令が異なります。いずれかのアーキテクチャーに基づくシステムでプログラムを実行できるようにするために、オペレーティング・システムによって一連の特殊ルーチンが提供されています。 これらはミリコード・ルーチンと呼ばれ、マシン依存の重要な機能とパフォーマンス上重要な機能を含んでいます。 ミリコード・ルーチンは、カーネル・セグメント内の固定アドレスにあります。 これらのルーチンには、 **bla** 命令で到達できます。 すべてのミリコード・ルーチンは、リンク・レジスターを使用します。

注 :

1. 不要なレジスターは破棄されません。 レジスターの使用法については、各ミリコード・ルーチンの定義を参照してください。
2. ミリコード・ルーチンは、浮動小数点レジスター、カウント・レジスター、または汎用レジスター (GPR) 10 から 12 を変更しません。 リンク・レジスターは、不揮発性 GPR を使用しないリーフ・プロシージャーに呼び出しが現れた場合に、GPR (例えば GPR 10) に保管することができます。
3. ミリコード・ルーチンは TOC を使用しません。

以下のミリコード・ルーチンが使用可能です。

項目 **説明**

__mulh 整数積 *arg1* ***** *arg2* の上位 32 ビットを計算します。

Input
R3 = *arg1* (符号付き整数)
R4 = *arg2* (符号付き整数)

出力
R3 = *arg1* の上位 32 ビット*****arg2*

POWER ® ファミリー レジスターの使用法
GPR3, GPR4, MQ

PowerPC ® 使用状況の登録
GPR3, GPR4

__mull 2 つの 32 ビット・レジスターに戻される、整数積 *arg1* ***** *arg2* の 64 ビットを計算します。

Input
R3 = *arg1* (符号付き整数)
R4 = *arg2* (符号付き整数)

出力
R3 = *arg1* の上位 32 ビット*****arg2*
R4 = *arg1* の下位 32 ビット*****arg2*

POWER ® ファミリー レジスターの使用法
GPR3, GPR4, MQ

PowerPC ® 使用状況の登録
GPR0, GPR3, GPR4

項目	説明
__divss	符号付き整数 *arg1*/*arg2* の 32 ビット商と 32 ビット剰余を計算します。 ゼロ除算およびオーバーフローの場合、商と剰余は未定義であり、実装によって異なる可能性があります。 **Input** R3 = *arg1* (被除数) (符号付き整数) R4 = *arg2* (除数) (符号付き整数) **出力** R3 = *arg1* の商/*arg2* (符号付き整数) R4 = *arg1* の剰余/*arg2* (符号付き整数) **POWER® ファミリー レジスターの使用法** GPR3, GPR4, MQ **PowerPC® 使用状況の登録** GPR0, GPR3, GPR4
__divus	符号なし整数 *arg1*/*arg2* の 32 ビット商と 32 ビット剰余を計算します。 ゼロ除算およびオーバーフローの場合、商と剰余は未定義であり、実装によって異なる可能性があります。 **Input** R3 = *arg1* (被除数) (符号なし整数) R4 = *arg2* (除数) (符号なし整数) **出力** R3 = *arg1*/*arg2* (符号なし整数) の商。 R4 = *arg1* の剰余/*arg2* (符号なし整数) **POWER® ファミリー レジスターの使用法** CR の GPR0、 GPR3、 GPR4、 MQ、 CR0 および CR1 **PowerPC® 使用状況の登録** GPR0, GPR3, GPR4
__quoss (__quoss)	符号付き整数 *arg1*/*arg2* の 32 ビット商を計算します。 ゼロ除算およびオーバーフローの場合、商と剰余は未定義であり、実装によって異なる可能性があります。 **Input** R3 = *arg1* (被除数) (符号付き整数) R4 = *arg2* (除数) (符号付き整数) **出力** R3 = *arg1* の商/*arg2* (符号付き整数) **POWER® ファミリー レジスターの使用法** GPR3, GPR4, MQ **PowerPC® 使用状況の登録** GPR3, GPR4

項目	説明
__quous (__quous)	符号なし整数 *arg1*/*arg2* の 32 ビット商を計算します。 ゼロ除算およびオーバーフローの場合、商と剰余は未定義であり、実装によって異なる可能性があります。 **Input** R3 = *arg1* (被除数) (符号なし整数) R4 = *arg2* (除数) (符号なし整数) **出力** R3 = *arg1*/*arg2* (符号なし整数) の商。 **POWER® ファミリー レジスターの使用法** CR の GPR0、 GPR3、 GPR4、 MQ、 CR0 および CR1 **PowerPC® 使用状況の登録** GPR3, GPR4

以下の例では、アセンブラー・プログラムで **mulh** ミリコード・ルーチンを使用しています。

```
li R3, -900
li R4, 50000
bla .__mulh
...
.extern .__mulh
```

目次の理解とプログラミング

TOC は、XCOFF ファイル内のオブジェクトを検索するために使用されます。

XCOFF ファイルの目次 (TOC) は、ブックの目次と類似しています。 TOC は、XCOFF ファイル内のオブジェクトを検索するために使用されます。 XCOFF ファイルは、特定の目的で使用されるさまざまなタイプのデータを含むセクションで構成されます。 一部のセクションは、さらにサブセクションまたは *csects* に分割することができます。 csect は、XCOFF ファイルの最小の交換可能ユニットです。 実行時に、TOC には csect の位置 (および csects 内のラベルの位置) を含めることができます。

csects を含む 3 つのセクションは、以下のとおりです。

項目	説明
テキスト	この csect にコードまたは読み取り専用データが含まれていることを示します。
.データ	この csect に読み取り/書き込みデータが含まれていることを示します。
.bss	この csect に初期化されていないマップ・データが含まれていることを示します。

csect のストレージ・クラスによって、csect がグループ化されるセクションが決まります。

TOC は、XCOFF オブジェクト・ファイルの **.data** セクションにあり、TOC エントリーで構成されます。 各 TOC エントリーは、TC または TD のストレージ・マッピング・クラスを持つ csect です。

TD ストレージ・マッピング・クラスを持つ TOC エントリーには、TOC から直接アクセスできるスカラー・データが含まれています。 これにより、頻繁に使用されるいくつかのグローバル・シンボルは、TOC 内に含まれるアドレス・ポインター csect を介して間接的にではなく、TOC から直接アクセスすることができます。 TOC 内のスカラー・データにアクセスするには、以下の 2 つの情報が必要です。

- TOC の先頭の位置 (つまり、TOC アンカー)。
- TOC アンカーから、データを含む特定の TOC エントリーまでのオフセット。

TC ストレージ・マッピング・クラスを持つ TOC エントリーには、他の CSECT またはグローバル・シンボルのアドレスが含まれます。 各エントリーには、CSECT またはグローバル・シンボルの 1 つ以上のアドレスを入れることができますが、各 TOC エントリーには 1 つのアドレスのみを入れることをお勧めします。

プログラムがアセンブルされると、**.text** csects が最初に書き込まれ、その後に TOC を除くすべての **.data** csects が続くように csects がソートされます。 TOC は、他のすべての **.data** CSECT の後に書き込まれます。 TOC エントリーは再配置されるため、TC ストレージ・マッピング・クラスを持つ TOC エントリーには、ソース・プログラム内の csect アドレスではなく、ソート後の csect アドレスが含まれます。

XCOFF モジュールがロードされると、TC ストレージ・マッピング・クラスを持つ TOC エントリーは再配置されます。これにより、csects がメモリー内に常駐する実アドレスが TOC エントリーに充てんされます。 モジュール内の csect にアクセスするには、次の 2 つの情報が必要です。

- TOC の開始の位置。
- TOC の先頭から csect を指し示す特定の TOC エントリーまでのオフセット。 TOC エントリーに複数のアドレスがある場合は、オフセットに (0 ... (*n*-1)) * 4 を加算することによって各アドレスを計算できます。ここで、 *n* は 518 ページの『.tc 疑似命令』 で定義された csect アドレスの位置です。

toc の使用

TOC は、特定の規則を使用して作成されます。

TOC を使用するには、以下の特定の規則に従う必要があります。

- 汎用レジスター 2 には、常に TOC へのポインターが入っています。
- アセンブラー・プログラムの **.text** セクションから **.data** または **.bss** セクションへの参照はすべて、TOC を介して行う必要があります。

TOC レジスター (汎用レジスター 2) は、プログラムの呼び出し時にシステムによってセットアップされます。 これは、作成されたコードによって保守される必要があります。 TOC レジスターは、モジュール内のすべてのルーチンがデータ項目にアクセスできるように、モジュール・コンテキストを提供します。

これらの 2 番目の規則では、**.text** セクションと **.data** セクションをメモリー内の異なる場所に簡単にロードできます。 この規則に従うことにより、モジュールの再配置を必要とする部分が TOC エントリーのみであることを確認できます。

tc ストレージ・マッピング・クラスを使用した toc エントリーを介したデータへのアクセス

外部データ項目にアクセスするには、まず TOC からその項目のアドレスを取得し、次にそのアドレスを使用してデータを取得します。

外部データ項目にアクセスするには、まず TOC からその項目のアドレスを取得し、次にそのアドレスを使用してデータを取得します。 これを行うには、正しい TOC エントリーにアクセスするための適切な再配置情報を提供する必要があります。 **.toc** および **.tc** 疑似命令は、TOC エントリーにアクセスするための正しい情報を生成します。 以下のコードは、TOC エントリーを使用して項目 a にアクセスする方法を示しています。

```
       .set     RTOC,2
       .csect prog1[pr]          #prog1 is a csect
                                 #containing instrs.
       ...
       l  5,TCA(RTOC)            #Now GPR5 contains the
                                 #address of a[rw].
...
       .toc
TCA:   .tc  a[tc],a[rw]          #1st parameter is TOC entry
                                 #name, 2nd is contents of
                                 #TOC entry.
       .extern a[rw]             #a[rw] is an external symbol.
```

この同じメソッドは、プログラムの静的内部データにアクセスするために使用されます。このデータは、呼び出しによってその値を保持しますが、データ項目が宣言されているファイル内のプロシージャーによってのみアクセスできます。 以下は、 **static** 属性を持つ C 言語データです。

```
static int xyz;
```

このデータには、規則によって決定される名前が付けられます。 XCOFF では、名前の前に下線が付きます。

```
        .csect prog1[pr]
        ...
        l 1,STprog1(RTOC)         #Load r1 with the address
                                  #prog1's static data.
        ...
        .csect _prog1[rw]         #prog1's static data.
        .long    0
        ...
        .toc
 STprog1: .tc.prog1[tc],_prog1[rw]       #TOC entry with address of
                                         #prog1's static data.
```

ストレージ・マッピング・クラスを使用した toc エントリーを介したデータへのアクセス

TE ストレージ・マッピング・クラスは、外部データにアクセスするために使用されます。

TC ストレージ・マッピング・クラスの場合と同様に、TE ストレージ・マッピング・クラスを使用して外部データにアクセスすることができます。 外部データ項目にアクセスするには、まず TOC からその項目のアドレスをロードし、次にそのアドレスを使用してデータを取得します。 TOC オーバーフロー・コードの生成を避けるために、次の例に示すように、TE シンボルは 2 命令シーケンスで TOC からロードされます。

```
        .toc
        .tc      a[TE],a[RW]

        .extern  a[RW]
        .csect   prog1[PR]
        ...
        addis    3,a[TE](2)       # R_TOCU relocation used by default.
        ld       5,a[TE](3)       # R_TOCL relocation used by default.
        # Now GPR5 contains the address of a[RW]
```

TOC から a[TE] をロードする 2 つの命令は順次である必要はありませんが、参照されるシンボルにオフセットを追加することはできません。 例えば、 ld 命令を以下のようにすることはできません。

```
        ld       5,a[TE]+8(3)       # Invalid reference
```

ストレージ・マッピング・クラスの選択、および R_TOCU と R_TOCL の再配置タイプは、個別に選択できます。 例えば、 a[TE] は、以下の命令で通常の TOC シンボルとして使用できます。

```
        ld       5,a[TE]@tc(2)       # GPR5 contains the address of a[RW]
```

2 命令シーケンスは、前の例の a[TC] で使用することもできます。

```
        addis    5,a[TC]@u(2)
        ld       5,a[TC]@l(5)       # GPR5 contains the address of a[RW]
```

td ストレージ・マッピング・クラスを使用した toc エントリーによるデータへのアクセス

スカラー・データ項目は、TD ストレージ・マッピング・クラスを持つ TOC 項目に保管し、TOC 項目から直接取り出すことができます。

スカラー・データ項目は、TD ストレージ・マッピング・クラスを持つ TOC 項目に保管し、TOC 項目から直接取り出すことができます。

注 : TD ストレージ・マッピング・クラスを持つ TOC エントリーは、 頻繁に使用される スカラーにのみ使用してください。 TOC が大きくなりすぎると (多数の項目または大きな項目のため)、アセンブラーは範囲外の変位を示すメッセージ 1252-171 を報告することがあります。

以下の例は、TD ストレージ・マッピング・クラスを持つ TOC としてスカラー・データ項目を保管および検索するいくつかの方法を示しています。 それぞれの例には、メインプログラム用の C ソース、1 つのモジュール用のアセンブラー・ソース、リンクとアセンブルのための命令、およびプログラムの実行からの出力が含まれています。

td ストレージ・マッピング・クラスで *.csect pseudo-op* を使用する例

メイン C prgram `td1.c` のソース。

1. C メインプログラム `td1.c` のソースは次のとおりです。

```
/* This C module named td1.c   */
extern long   t_data;
extern  void mod_s();
main()
{
         mod_s();
         printf("t_data is %d\n", t_data);
}
```

2. 以下は、モジュール `mod1.s` のアセンブラー・ソースです。

```
        .file  "mod1.s"
        .csect .mod_s[PR]
        .globl .mod_s[PR]
        .set    RTOC, 2
        l   5, t_data[TD](RTOC)  # Now GPR5 contains the
                                 # t_data value 0x10
        ai  5,5,14
        stu 5, t_data[TD](RTOC)
        br
        .globl  t_data[TD]
        .toc
        .csect  t_data[TD]  # t_data is a global symbol
                            # that has value of 0x10
                            # using TD csect will put this
                            # data into TOC area
        .long    0x10
```

3. 以下のコマンドは、ソース・プログラムをアセンブルし、実行可能 `td1` にコンパイルします。

```
as -o mod1.o mod1.s
cc -o td1 td1.c mod1.o
```

4. `td1` を実行すると、以下が出力されます。

```
t_data is 30
```

td ストレージ・マッピング・クラスでの *.comm pseudo-op* の使用例

C メインプログラム `td2.c` のソース。

1. C メインプログラム `td2.c` のソースは次のとおりです。

```
/* This C module named td2.c   */
extern long   t_data;
extern  void mod_s();
main()
{
        t_data = 1234;
         mod_s();
         printf("t_data is %d\n", t_data);
}
```

2. 以下は、モジュール `mod2.s` のアセンブラー・ソースです。

```
        .file "mod2.s"
```

```
        .csect .mod_s[PR]
        .globl .mod_s[PR]
        .set    RTOC, 2
        l   5, t_data[TD](RTOC)  # Now GPR5 contains the
                                 # t_data value
        ai  5,5,14
        stu 5, t_data[TD](RTOC)
        br
        .toc
        .comm  t_data[TD],4  # t_data is a global symbol
```

3. 以下のコマンドは、ソース・プログラムをアセンブルし、実行可能 td2 にコンパイルします。

```
as -o mod2.o mod2.s
cc -o td2 td2.c mod2.o
```

4. td2 を実行すると、以下が出力されます。

```
t_data is 1248
```

外部 *td* シンボルの使用例

外部 TD シンボルの使用例

1.

```
/* This C module named td3.c   */
long   t_data;
extern  void mod_s();
main()
{
        t_data = 234;
         mod_s();
         printf("t_data is %d\n", t_data);
}
```

2. 以下は、モジュール mod3.s のアセンブラー・ソースです。

```
        .file "mod3.s"
        .csect .mod_s[PR]
        .globl .mod_s[PR]
        .set    RTOC, 2
        l   5, t_data[TD](RTOC)  # Now GPR5 contains the
                                 # t_data value
        ai  5,5,14
        stu 5, t_data[TD](RTOC)
        br
        .toc
        .extern  t_data[TD]  # t_data is a external symbol
```

3. 以下のコマンドは、ソース・プログラムをアセンブルし、実行可能 td3 にコンパイルします。

```
./as -o mod3.o mod3.s
cc -o td3 td3.c mod3.o
```

4. td3 を実行すると、以下が出力されます。

```
t_data is 248
```

toc を使用したモジュール間呼び出し

データ・セクションは、モジュール間呼び出しを使用できる機能を使用して TOC を介してアクセスされます。

テキストからデータ・セクションへのアクセスは TOC を介してのみであるため、TOC は、モジュール間呼び出しを使用できるようにする機能を提供します。 その結果、リンク時にすべてのアドレスまたはシンボルを解決することなく、ルーチンをリンクすることができます。 つまり、共通ユーティリティー・ルーチ

ンは、呼び出しルーチンと同じモジュールに実際にリンクすることなく、呼び出しを行うことができます。このようにして、ルーチンのグループをモジュールにすることができ、異なるグループ内のルーチンは互いに呼び出すことができ、バインド時間はロード時まで遅延します。 この機能を使用するには、別のモジュールにあるルーチンを呼び出すときに特定の規則に従う必要があります。

別のモジュール内のルーチンを呼び出すには、コンテキストを現行モジュールから新規モジュールに切り替えるインターフェース・ルーチン (または グローバル・リンケージ ・ルーチン) を呼び出します。 このコンテキスト・スイッチは、TOC ポインターを現行モジュールに保管し、新規モジュールの TOC ポインターをロードしてから、他のモジュールの新規ルーチンに分岐することによって簡単に実行できます。 その後、もう一方のルーチンは元のモジュールの元のルーチンに戻り、元の TOC アドレスが TOC レジスターにロードされます。

グローバル・リンケージをできるだけ透過的にするために、宛先モジュールを指定せずに外部ルーチンを呼び出すことができます。 バインド時に、バインダー (リンケージ・エディター) は、グローバル・リンケージ・コードを呼び出すかどうかを決定し、モジュール間呼び出しを実行するための適切なグローバル・リンケージ・ルーチンを挿入します。 グローバル・リンケージは、インポート・リストによって制御されます。 インポート・リストには、システムから、または別のオブジェクト・ファイルの動的ロードから、実行時に解決される外部シンボルが含まれます。 インポート・リストについては、**ld** コマンドを参照してください。

以下の例では、グローバル・リンケージを経由する可能性のあるルーチンを呼び出します。

```
.csect  prog1[PR]
...
.extern prog2[PR]        #prog2 is an external symbol.
bl      .prog2[PR]       #call prog2[PR], binder may insert
                         #global linkage code.
cror    31,31,31         #place holder for instruction to
                         #restore TOC address.
```

以下の例は、グローバル・リンケージ・ルーチンを介した呼び出しを示しています。

```
#AIX® linkage register conventions:
#        R2      TOC
#        R1      stack pointer
#        R0, R12 work registers, not preserved
#        LR      Link Register, return address.
         .csect  .prog1[PR]
         bl      .prog2[GL]             #Branch to global
                                        #linkage code.
         l        2,stktoc(1)           #Restore TOC address
         .toc
prog2:   .tc       prog2[TC],prog2[DS]  #TOC entry:
                                        # address of descriptor
                                        # for out-of-module
                                        # routine
        .extern   prog2[DS]
##
## The following is an example of global linkage code.
        .set      stktoc,20
        .csect   .prog2[GL]
        .globl   .prog2
.prog2:  l        12,prog2(2)           #Get address of
                                        #out-of-module
                                        #descriptor.
         st       2,stktoc(1)           #save callers' toc.
         l        0,0(12)               #Get its entry address
                                        #from descriptor.
         l        2,4(12)               #Get its toc from
                                        #descriptor.
         mtctr    0                     #Put into Count Register.
         bctr                           #Return to entry address
                                        #in Count Register.
                                        #Return is directly to
                                        #original caller.
```

スレッド・ローカル・ストレージの使用

スレッド・ローカル変数は、TL および UL ストレージ・マッピング・クラスを使用して宣言および定義することができます。

スレッド・ローカル変数は、TL および UL ストレージ・マッピング・クラスを使用して宣言および定義することができます。 スレッド・ローカル変数は、 AIX 実装によって定義されたコード・シーケンスを使用して参照されます。

スレッド・ローカル変数には、領域ハンドルと領域オフセットがあります。 一般に、 **__tls_get_addr ()** 関数は、変数領域ハンドルとオフセットを指定して、呼び出しスレッドのスレッド・ローカル変数のアドレスを計算するために呼び出されます。

その他のアクセス方式は、より制限された状況で使用できます。 **local-exec** アクセス方式は、メインプログラムでも定義されている変数を参照するためにメインプログラムによって使用されます。 **initial-exec** アクセス方式は、メインプログラムで定義されたスレッド・ローカル変数、またはメインプログラムと一緒にロードされた共用オブジェクトを参照するために使用されます。 **ローカル動的** アクセス方式は、同じモジュールで定義されたスレッド・ローカル変数を参照するためにモジュールによって使用されます。

スレッド・ローカル・ストレージへのアクセスでは、標準以外の呼び出し規則を持つ pthread ライブラリー内のルーチンが使用されます。 これらのルーチンは、 **__tls_get_addr ()** および **__tls_get_mod ()**です。 32 ビット・プログラムで使用されるルーチンは **__get_tpointer ()**です。 64 ビット・プログラムでは、現行スレッド・ポインターは常に **gpr13** に含まれています。

初期化されていないスレッド・ローカル・ストレージ変数 *bar* は、次のステートメントで宣言できます。

```
        .comm bar[UL]
```

同様に、スレッド・ローカル、初期化済み、整数変数 *foo* は、以下のステートメントを使用して宣言できます。

```
        .csect foo[TL]
        .long       1
```

アクセス方式	32 ビット・コード	64 ビット・コード (異なる場合)	コメント
一般-動的アクセス方式	.tc foo [TC]、foo [TL]		変数オフセット
	.tc .foo [TC]、foo [TL]@m		領域ハンドル
	lwz 4,foo[TC] (2)	ld 4,foo[TC] (2)	
	lwz 3,.foo[TC] (2)	ld 3,.foo[TC] (2)	
	ブラ __tls_get_addr		r0,r3,r4,r5,r11,lr,cr0 を変更します。
	#r3 = & 動物園		

アクセス方式	32 ビット・コード	64 ビット・コード (異なる場合)	コメント
ローカル動的アクセス方式	.tc foo [TC]、foo [TL]@ld		変数オフセット、 **ld** 再配置指定子
	.tc mh [TC]、mh [TC]@ml		呼び出し元のモジュール・ハンドル
	lwz 3,mh[TC] (2)	ld 3,mh[TC] (2)	
	ブラ __tls_get_mod		r0,r3,r4,r5,r11,lr,cr0 を変更します。
	#r3 = & モジュールの TLS		
	lwz 4,foo[TC] (2)	ld 4,foo[TC] (2)	
	5、3、4 を加える		*& foo* の計算
	.rename mh [TC]、"_$TLSML"		モジュール・ハンドルのシンボルの名前は「_$TLSML」でなければなりません
初期 EXEC アクセス方式	.tc foo [TC]、foo [TL]@ie		変数オフセット、 **ie** 再配置指定子
	bla __get_tpointer ()	# r13 には tpointer が含まれる	**__get_tpointer** は r3 を変更します。
	lwz 4,foo[TC] (2)	ld 4,foo[TC] (2)	
	5、3、4 を加える	5、4、13 を加える	*& foo* の計算
ローカル exec アクセス方式	.tc foo [TC]、foo [TL]@le		変数オフセット、 **le** 再配置指定子
	bla __get_tpointer ()	# r13 には tpointer が含まれる	**__get_tpointer** は r3 を変更します。
	lwz 4,foo[TC] (2)	ld 4,foo[TC] (2)	*& foo* の計算
	5、3、4 を加える	5、4、13 を加える	

ローカル動的およびローカル EXEC アクセス方式は、スレッド・ローカル変数の合計サイズが 62 KB より小さい場合に使用できる、より高速なコード・シーケンスを持っています。 領域の合計サイズが大きすぎる場合、リンケージ・エディターは追加の命令を生成することによってコードにパッチを適用し、より速いコード・シーケンスを使用する利点を否定します。

アクセス方式	32 ビット・コード	コメント
ローカル動的アクセス方式	.tc mh [TC]、mh [TC]@ml	呼び出し元のモジュール・ハンドル
	.rename mh [TC]、"_$TLSML"	モジュール・ハンドルのシンボルの名前は「_$TLSML」でなければなりません
	lwz 3,mh[TC] (2)	
	ブラ __tls_get_mod	
	la 4,foo[TL]@ld(3)	r4 = *&* 動物園

アクセス方式	32 ビット・コード	コメント
ローカル exec アクセス方式	ブラ __get_tpointer	r3 を変更します。
	la 4,foo[TL]@ld(3)	r4 = & 動物園
	# または	
	lwz 5,foo[TL]@le(13)	r5 = 動物園

アクセス方式	64 ビット・コード	コメント
ローカル動的アクセス方式	.tc mh [TC]、mh [TC]@ml	呼び出し元のモジュール・ハンドル
	.rename mh [TC]、"_$TLSML"	モジュール・ハンドルのシンボルの名前は「_$TLSML」でなければなりません
	ld 3,mh[TC] (2)	
	ブラ __tls_get_mod	
	la 4,foo[TL]@ld(3)	r4 = & 動物園
ローカル exec アクセス方式	la 4,foo[TL]@le(13)	r4 = & 動物園
	# または	
	lwz 5,foo[TL]@le(13)	r5 = 動物園

プログラムの実行

プログラムは、アセンブルされてリンクされると、エラー・メッセージを出さずに実行することができます。

プログラムは、アセンブルされてリンクされると、エラー・メッセージを出さずに実行することができます。 プログラムを実行するには、まず、ファイルを実行するためのオペレーティング・システム権限があることを確認してください。 次に、オペレーティング・システムのプロンプトでプログラムの名前を入力します。

```
$ progname
```

デフォルトでは、プログラム出力はすべて標準出力に出力されます。 標準出力以外の場所に出力を送信するには、オペレーティング・システム・シェルの **>** (より大記号) 演算子を使用します。

ランタイム・エラーは、**dbx** コマンドでシンボリック・デバッガーを呼び出すことによって診断できます。 このシンボリック・デバッガーは、XCOFF フォーマット規則に準拠するすべてのコードで機能します。 **dbx** コマンドを使用して、コンパイラー生成およびアセンブラー生成のすべてのコードをデバッグすることができます。

拡張命令ニーモニック

アセンブラーは、アセンブリー言語プログラミングを単純化するために、拡張ニーモニックとシンボルのセットをサポートします。

アセンブラーは、アセンブリー言語プログラミングを単純化するために、拡張ニーモニックとシンボルのセットをサポートします。 すべての拡張ニーモニックは、基本ニーモニックと同じアセンブリー・モードでなければなりません。 POWER® ファミリーと PowerPC® では異なる拡張ニーモニックが提供されていますが、基本ニーモニックが **com** アセンブリー・モードの場合、アセンブラーは拡張ニーモニックに対して同じオブジェクト・コードを生成します。 拡張簡略記号のアセンブリー・モードは、各拡張簡略記号セクションにリストされています。 POWER® ファミリーおよび PowerPC® 拡張ニーモニックは、マイグレーションの目的で以下のセクションに個別にリストされています。

ブランチ命令の拡張ニーモニック

アセンブラーは、さまざまなタイプのレジスター命令の拡張ニーモニックをサポートします。

アセンブラーは、Branch Conditional、Branch Conditional to Link Register、および Branch Conditional to Count Register 命令の拡張ニーモニックをサポートします。 すべてのブランチ条件付き命令の基本ニーモニックは **com** アセンブリー・モードであるため、それらの拡張ニーモニックもすべて **com** アセンブリー・モードになります。

拡張簡略記号は、 *BO* および *BI* 入力オペランドを簡略記号に組み込むことによって構成されます。 拡張ニーモニックは、常に *BH* 入力オペランドを省略し、その値を 0b00 と想定します。

Bo オペランドのみを組み込んだブランチ・ニーモニック

BO フィールドのみを組み込んだ拡張ニーモニックの命令フォーマット。

以下の表は、 *BO* フィールドのみを組み込んだ拡張ニーモニックの命令フォーマットを示しています。 ターゲット・アドレスは、*target_addr* オペランドによって指定されます。 条件比較の条件レジスター内のビットは、 *BI* オペランドによって指定されます。 *BI* オペランドの値は、式で指定できます。 各 CR フィールドには 4 ビットがあるため、CR フィールド番号に 4 を乗算して正しい CR ビットを取得する必要があります。

注 : 一部の拡張簡略記号には、2 つの入力オペランド形式があります。

表 *7. POWER* ® ファミリー拡張ニーモニック *(BO* フィールドのみ*)*

簡略記号	**入力オペランド**	**以下と同等**	
bdz、**bdza**、**bdzl**、**bdzla**	ターゲット・アドレス *(target_addr)*	**bc**、**bca**、**bcl**、**bcla**	**18**、**0**、 *target_addr*
bdn、 **bdna**、 **bdnl**、 **bdnla**	ターゲット・アドレス *(target_addr)*	**bc**、**bca**、**bcl**、**bcla**	**16**、**0**、 *target_addr*
bdzr、 **bdzrl**	なし	**bcr**、**bcrl**	**18**、**0**
bdnr、 **bdnrl**	なし	**bcr**、 **bcrl**	**16**、**0**
bbt、**bbta**、**bbtl**、**bbtla**	1) *BI*、 *target_addr*	**bc**、**bca**、**bcl**、**bcla**	**12**、 *BI*、 *target_addr*
	2) *target_addr*		**12**、**0**、 *target_addr*
bbf、**bbfa**、**bbfl**、**bbfla**	1) *BI*、 *target_addr*	**bc**、**bca**、**bcl**、**bcla**	**4**、 *BI*、 *target_addr*
	2) *target_addr*		**4**、**0**、 *target_addr*
bbtr、 **bbtc**、 **bbtrl**、 **bbtcl**	1) *BI*	**bcr**、**bcc**、**bcrl**、**bccl**	**12**、 *BI*
	2) なし		**12**、**0**
bbfr、**bbfc**、**bbfrl**、**bbfcl**	1) *BI*	**bcr**、**bcc**、**bcrl**、**bccl**	**4**、 *BI*
	2) なし		**4**、**0**
br、 **bctr**、 **brl**、 **bctrl**	なし	**bcr**、**bcc**、**bcrl**、**bccl**	**20**、**0**

表 *8. PowerPC* ® 拡張ニーモニック *(BO* フィールドのみ*)*

簡略記号	**入力オペランド**	**同等**
bdz、 **bdza**、 **bdzl**、 **bdzla**	ターゲット・アドレス *(target_addr)*	**bc**、**bca**、**bcl**、**bcla 18**、**0**、 *target_addr*
bdnz、 **bdnza**、 **bdnzl**、 **bdnzla**	ターゲット・アドレス *(target_addr)*	**bc**、**bca**、**bcl**、**bcla 16**、**0**、 *target_addr*

表 *8. PowerPC* ® 拡張ニーモニック *(BO* フィールドのみ*)* (続き*)*		
簡略記号	**入力オペランド**	**同等**
bdzlr、 **bdzlrl**	なし	**bclr**、**bclrl 18**、**0**
bdnzlr、 **bdnzlrl**	なし	**bclr**、**bclrl 16**、**0**
bt、 **bta**、 **btl**、 **btla**	1) *BI*、 *target_addr*	**bc**、**bca**、**bcl**、**bcla 12**、 *BI*、 *target_addr*
	2) *target_addr*	**12**、**0**、 *target_addr*
bf、 **bfa**、 **bfl**、 **bfla**	1) *BI*、 *target_addr*	**bc**、**bca**、**bcl**、**bcla 4**、 *BI*、 *target_addr*
	2) *target_addr*	**4**、**0**、 *target_addr*
bdzt、 **bdzta**、 **bdztl**、 **bdztla**	1) *BI*、 *target_addr*	**bc**、**bca**、**bcl**、**bcla 10**、 *BI*、 *target_addr*
	2) *target_addr*	**10**、**0**、 *target_addr*
bdzf、 **bdzfa**、 **bdzfl**、 **bdzfla**	1) *BI*、 *target_addr*	**bc**、**bca**、**bcl**、**bcla 2**、 *BI*、 *target_addr*
	2) *target_addr*	**2**、**0**、 *target_addr*
bdnzt、**bdnzta**、**bdnztl**、**bdnztla**	1) *BI*、 *target_addr*	**bc**、**bca**、**bcl**、**bcla 8**、 *BI*、 *target_addr*
	2) *target_addr*	**8**、**0**、 *target_addr*
bdnzf、**bdnzfa**、**bdnzfl**、**bdnzfla**	1) *BI*、 *target_addr*	**bc**、**bca**、**bcl**、**bcla 0**、 *BI*、 *target_addr*
	2) *target_addr*	**0**、**0**、 *target_addr*
btlr、**btctr**、**btlrl**、**btctrl**	1) *BI*	**bclr**、**bcctr**、**bclrl**、**bcctrl 12**、 *BI*
	2) なし	**12**、**0**
bflr、 **bfctr**、 **bflrl**、 **bfctrl**	1) *BI*	**bclr**、**bcctr**、**bclrl**、**bcctrl 4**、 *BI*
	2) なし	**4**、**0**
bdztlr、 **bdztlrl**	1) *BI*	**bclr**、**bclrl 10**、 *BI*
	2) なし	**10**、**0**
bdzflr、 **bdzflrl**	1) *BI*	**bclr**、**bclrl 2**、 *BI*
	2) なし	**2**、**0**
bdnztlr、 **bdnztlrl**	1) *BI*	**bclr**、**bclrl 8**、 *BI*
	2) なし	**8**、**0**
bdnzflr、 **bdnzflrl**	1) *BI*	**bclr**、**bclrl 0**、 *BI*
	2) なし	**0**、**0**
blr、 **bctr**、 **blrl**、 **bctrl**	なし	**bclr**、**bcctr**、**bclrl**、**bcctrl 20**、**0**

Bo フィールドと部分的な BI フィールドを組み込んだ拡張分岐ニーモニック

BO フィールドおよび *BI* フィールドが取り込まれるときの拡張ブランチ・ニーモニック命令フォーマット。

BO フィールドと部分的な *BI* フィールドが取り込まれると、命令形式は以下のいずれかになります。

- 簡略記号 *BIF*、 *target_addr*
- ニーモニック *target_addr*

ここで、*BIF* オペランドは CR フィールド番号 (0 から 7) を指定し、*target_addr* オペランドはターゲット・アドレスを指定します。 CR0 を使用する場合は、 *BIF* オペランドを省略できます。

CR フィールドのビット定義に基づいて、分岐条件の最も一般的な組み合わせに対して、以下の一連のコードが定義されています。

支店コード	**意味**
LT	より小さい *
EQ	* と等しい
GT	* より大きい
so	合計オーバーフロー *
le	より小か等しい * (より大ではない)
GE	より大か等しい * (より小でない)
NE	次と等しくない *
NS	合計オーバーフローではない *
nl	より小さくない
NG	大きくない
z	ゼロ
NU	順不同でない (浮動小数点比較の後)
NZ	ゼロでない
UN	順不同 (浮動小数点比較の後)

アセンブラーは、 *BO* オペランドとして以下の 6 つのエンコード値をサポートします。

- 分岐条件真 (*BO*= 12):

POWER® ファミリー	**PowerPC (R)**
b*xx*	**b***xx*
b*xx***a**	**b***xx***a**
b*xx*	**b***xx*
b*xx* **ラ**	**b***xx* **ラ**
b*xx***r**	**b***xx***lr**
b*xx***rl**	**b***xx***lrl**
b*xx***c**	**b***xx***ctr**
b*xx***cl**	**b***xx***ctrl**

  ここで、 *xx* は、 *BI* オペランドのブランチ・コード lt、 gt、 eq、 so、 z、または un を指定します。

- 分岐条件偽 (*BO*= 04):

POWER® ファミリー	**PowerPC (R)**
b*xx*	**b***xx*
b*xx***a**	**b***xx***a**

POWER® ファミリー	**PowerPC (R)**
b*xx*	**b***xx*
b*xx* **ラ**	**b***xx* **ラ**
b*xx***r**	**b***xx***lr**
b*xx***rl**	**b***xx***lrl**
b*xx***c**	**b***xx***ctr**
b*xx***cl**	**b***xx***ctrl**

ここで、 *xx* は、 *BI* オペランドのブランチ・コード ge、 le、 ne、 ns、 nl、 ng、 nz、または nu を指定します。

- CTR がゼロ以外で、条件が真 (*BO*= 08) の場合は、CTR を減分して分岐します。
  - **bdn***xx*

  ここで、 *xx* は、 *BI* オペランドのブランチ・コード lt、 gt、 eq、または so (ブランチ・コード・リストで * (アスタリスク) のマークが付いている) を指定します。
- CTR がゼロ以外で条件が偽 (*BO*= 00) の場合は、CTR を減分して分岐します。
  - **bdn***xx*

  ここで、 *xx* は、 *BI* オペランドのブランチ・コード le、 ge、 ne、または ns (ブランチ・コード・リストで * (アスタリスク) のマークが付いている) を指定します。
- CTR がゼロで、条件が真 (*BO*= 10) の場合は、CTR を減分して分岐します。
  - **bdz***xx*

  ここで、 *xx* は、 *BI* オペランドのブランチ・コード lt、 gt、 eq、または so (ブランチ・コード・リストで * (アスタリスク) のマークが付いている) を指定します。
- CTR がゼロで条件が偽 (*BO*= 02) の場合は、CTR を減分して分岐します。
  - **bdz***xx*

  ここで、 *xx* は、 *BI* オペランドのブランチ・コード le、 ge、 ne、または ns (ブランチ・コード・リストで * (アスタリスク) のマークが付いている) を指定します。

基本ニーモニックおよび拡張ニーモニックの分岐条件付き命令の BI オペランド

基本ニーモニックおよび拡張ニーモニックの分岐条件付き命令の BI オペランド。

BI オペランドは、条件比較のために条件レジスター内のビット (0:31) を指定します。 ビットは比較命令によって設定されます。 条件レジスターのビットは、8 つの 4 ビット・フィールドにグループ化されます。 これらのフィールドの名前は、CR フィールド 0 から CR フィールド 7 (CR0...CR7). 各フィールドのビットは、次のように解釈されます。

ビット	**説明**
0	より小; 浮動小数点より小
1	より大きい: より大きい浮動小数点
2	等しい; 浮動小数点等しい
3	要約オーバーフロー、浮動小数点順不同

通常、基本および拡張分岐条件付きニーモニック・テーブルの BI オペランド・シンボルに示されるシンボルは、 *BI* オペランドで使用するために定義されています。 アセンブラーは、 *BI* オペランドの式をサポートします。 式は、値と以下の記号の組み合わせです。

表 9. 基本および拡張分岐条件付きニーモニックの *BI* オペランド・シンボル		
シンボル	**値**	**意味**
LT	0	LESS THAN
GT	1	より大きい
EQ	2	等しい
so	3	合計オーバーフロー
UN	3	順不同 (浮動小数点比較の後)
cr0	0	CR フィールド 0
cr1	1	CR フィールド 1
cr2	2	CR フィールド 2
cr3	3	CR フィールド 3
cr4	4	CR フィールド 4
cr5	5	CR フィールド 5
cr6	6	CR フィールド 6
cr7	7	CR フィールド 7

BO フィールドのみが組み込まれた基本ニーモニックまたは拡張ニーモニックで *BI* フィールドの式を使用する場合は、CR フィールド番号に 4 を乗算して正しい CR ビットを取得する必要があります。これは、各 CR フィールドに 4 ビットがあるためです。

1. CTR を減分するには、CTR がゼロでなく、 CR5 の条件が等しい場合にのみ分岐します。

   ```
   bdnzt   4*cr5+eq, target_addr
   ```

   これは次のコマンドと等価です。

   ```
   bc   8, 22, target_addr
   ```

2. CTR を減分するには、CTR がゼロでなく、 CR0 の条件が等しい場合にのみ分岐します。

   ```
   bdnzt   eq, target_addr
   ```

   これは次のコマンドと等価です。

   ```
   bc   8, 2, target_addr
   ```

   BI オペランドが CR0 のビット 0 を指定する場合は、 *BI* オペランドを省略できます。
3. CTR を減分してから、CTR がゼロの場合にのみ分岐するには、次のようにします。

   ```
   bdz   target_addr
   ```

   これは次のコマンドと等価です。

   ```
   bc   18, 0, target_addr
   ```

BO フィールドと部分的な *BI* フィールドが組み込まれた拡張簡略記号の場合、 *BI* オペランドの値は CR フィールド番号を示します。 有効な値は 0 から 7 です。 値 0 を使用する場合は、 *BI* オペランドを省略できます。

1. CR0 が以下の条件を反映している場合に分岐します。

   ```
   bge   target_addr
   ```

   これは次のコマンドと等価です。

   ```
   bc   4, 0, target_addr
   ```

2. CR4 がより大きいことを示している場合に絶対ターゲットに分岐し、リンク・レジスターを設定するには、次のようにします。

   ```
   bgtla   cr4, target_addr
   ```

   これは次のコマンドと等価です。

   ```
   bcla   12, 17, target_addr
   ```

   BI オペランド CR4 は、アセンブラーによって内部的に 16 に拡張されます。 gt (より大きい) が取り込まれると、 *BI* フィールドの結果は 17 になります。

分岐予測の拡張ニーモニック

アセンブラー・ソース・プログラムは、分岐予測接尾部を命令の簡略記号に追加することによって、分岐条件付き命令に関する情報を持つことができます。

指定された分岐条件付き命令の結果 (分岐またはフォールスルー) が判明している場合、プログラマーは、命令のニーモニックに分岐予測接尾部を追加することによって、この情報をアセンブラー・ソース・プログラムに組み込むことができます。 アセンブラーは、分岐予測情報を使用して、マシン・インストラクションのビットの値を判別します。 分岐予測接尾部を使用すると、分岐条件付き命令の平均パフォーマンスが向上する可能性があります。

以下の接尾部は、基本または拡張のいずれかのブランチ条件付きニーモニックに追加できます。

項目	説明
+	選択する分岐の予測
-	分岐を取らないように予測 (フォールスルー)

分岐予測接尾部は、残りのニーモニックの直後に置く必要があります (分離文字なし)。 分岐予測接尾部とオペランドの間には、分離文字 (スペースまたはタブ) を使用する必要があります。

分岐予測接尾部がニーモニックに含まれていない場合、アセンブラーはマシン・インストラクションを構成する際に以下のデフォルトの想定を使用します。

- 負の変位フィールドを持つ相対ブランチまたは絶対ブランチ (**bc**[**l**] [**a**]) の場合、ブランチが取得されると予測されます。
- 負でない変位フィールドを持つ相対または絶対ブランチ (**bc**[**l**] [**a**]) の場合、ブランチは取得されない (フォールスルー予測) と予測されます。
- LR または CTR (**bclr**[**l**]) または (**bcctr**[**l**]) 内のアドレスへの分岐の場合、分岐は取られない (フォールスルーが予測される) ことが予測されます。

分岐予測接尾部によって制御されるマシン・インストラクションの部分は、 *BO* フィールドの *y* ビットです。 *y* ビットは次のように設定されます。

- 分岐予測接尾部を指定しないか、またはデフォルトの想定と同じ接尾部を使用すると、*y* ビットが 0 に設定されます。
- デフォルトの想定の反対である分岐予測接尾部を指定すると、 *y* ビットが 1 に設定されます。

以下の例は、分岐予測接尾部の使用法を示しています。

1. CR0 が条件より小さい場合に分岐します。 通常、命令を実行すると分岐します。

```
blt+  target
```

2. CR0 が条件より小さい場合に分岐します。 ターゲット・アドレスはリンク・レジスターにあります。 通常、命令を実行すると、次の命令に進みます。

```
bltlr-
```

AIX® アセンブラーでサポートされる Branch Prediction 命令のリストを以下に示します。

```
      bc+             bc-             bca+            bca-
      bcctr+          bcctr-          bcctrl+         bcctrl-
      bcl+            bcl-            bcla+           bcla-
      bclr+           bclr-           bclrl+          bclrl-
      bdneq+          bdneq-          bdnge+          bdnge-
      bdngt+          bdngt-          bdnle+          bdnle-
      bdnlt+          bdnlt-          bdnne+          bdnne-
      bdnns+          bdnns-          bdnso+          bdnso-
      bdnz+           bdnz-           bdnza+          bdnza-
      bdnzf+          bdnzf-          bdnzfa+         bdnzfa-
      bdnzfl+         bdnzfl-         bdnzfla+        bdnzfla-
      bdnzflr+        bdnzflr-        bdnzflrl+       bdnzflrl-
      bdnzl+          bdnzl-          bdnzla+         bdnzla-
      bdnzlr+         bdnzlr-         bdnzlrl+        bdnzlrl-
      bdnzt+          bdnzt-          bdnzta+         bdnzta-
      bdnztl+         bdnztl-         bdnztla+        bdnztla-
      bdnztlr+        bdnztlr-        bdnztlrl+       bdnztlrl-
      bdz+            bdz-            bdza+           bdza-
      bdzeq+          bdzeq-          bdzf+           bdzf-
      bdzfa+          bdzfa-          bdzfl+          bdzfl-
      bdzfla+         bdzfla-         bdzflr+         bdzflr-
      bdzflrl+        bdzflrl-        bdzge+          bdzge-
      bdzgt+          bdzgt-          bdzl+           bdzl-
      bdzla+          bdzla-          bdzle+          bdzle-
      bdzlr+          bdzlr-          bdzlrl+         bdzlrl-
      bdzlt+          bdzlt-          bdzne+          bdzne-
      bdzns+          bdzns-          bdzso+          bdzso-
      bdzt+           bdzt-           bdzta+          bdzta-
      bdztl+          bdztl-          bdztla+         bdztla-
      bdztlr+         bdztlr-         bdztlrl+        bdztlrl-
      beq+            beq-            beqa+           beqa-
      beqctr+         beqctr-         beqctrl+        beqctrl-
      beql+           beql-           beqla+          beqla-
      beqlr+          beqlr-          beqlrl+         beqlrl-
      bf+             bf-             bfa+            bfa-
      bfctr+          bfctr-          bfctrl+         bfctrl-
      bfl+            bfl-            bfla+           bfla-
      bflr+           bflr-           bflrl+          bflrl-
      bge+            bge-            bgea+           bgea-
      bgectr+         bgectr-         bgectrl+        bgectrl-
      bgel+           bgel-           bgela+          bgela-
      bgelr+          bgelr-          bgelrl+         bgelrl-
      bgt+            bgt-            bgta+           bgta-
      bgtctr+         bgtctr-         bgtctrl+        bgtctrl-
      bgtl+           bgtl-           bgtla+          bgtla-
      bgtlr+          bgtlr-          bgtlrl+         bgtlrl-
      ble+            ble-            blea+           blea-
      blectr+         blectr-         blectrl+        blectrl-
      blel+           blel-           blela+          blela-
      blelr+          blelr-          blelrl+         blelrl-
      blt+            blt-            blta+           blta-
      bltctr+         bltctr-         bltctrl+        bltctrl-
      bltl+           bltl-           bltla+          bltla-
      bltlr+          bltlr-          bltlrl+         bltlrl-
      bne+            bne-            bnea+           bnea-
      bnectr+         bnectr-         bnectrl+        bnectrl-
```

```
bnel+           bnel-           bnela+          bnela-
bnelr+          bnelr-          bnelrl+         bnelrl-
bng+            bng-            bnga+           bnga-
bngctr+         bngctr-         bngctrl+        bngctrl-
bngl+           bngl-           bngla+          bngla-
bnglr+          bnglr-          bnglrl+         bnglrl-
bnl+            bnl-            bnla+           bnla-
bnlctr+         bnlctr-         bnlctrl+        bnlctrl-
bnll+           bnll-           bnlla+          bnlla-
bnllr+          bnllr-          bnllrl+         bnllrl-
bns+            bns-            bnsa+           bnsa-
bnsctr+         bnsctr-         bnsctrl+        bnsctrl-
bnsl+           bnsl-           bnsla+          bnsla-
bnslr+          bnslr-          bnslrl+         bnslrl-
bnu+            bnu-            bnua+           bnua-
bnuctr+         bnuctr-         bnuctrl+        bnuctrl-
bnul+           bnul-           bnula+          bnula-
bnulr+          bnulr-          bnulrl+         bnulrl-
bnz+            bnz-            bnza+           bnza-
bnzctr+         bnzctr-         bnzctrl+        bnzctrl-
bnzl+           bnzl-           bnzla+          bnzla-
bnzlr+          bnzlr-          bnzlrl+         bnzlrl-
bso+            bso-            bsoa+           bsoa-
bsoctr+         bsoctr-         bsoctrl+        bsoctrl-
bsol+           bsol-           bsola+          bsola-
bsolr+          bsolr-          bsolrl+         bsolrl-
bt+             bt-             bta+            bta-
btctr+          btctr-          btctrl+         btctrl-
btl+            btl-            btla+           btla-
btlr+           btlr-           btlrl+          btlrl-
bun+            bun-            buna+           buna-
bunctr+         bunctr-         bunctrl+        bunctrl-
bunl+           bunl-           bunla+          bunla-
bunlr+          bunlr-          bunlrl+         bunlrl-
bz+             bz-             bza+            bza-
bzctr+          bzctr-          bzctrl+         bzctrl-
bzl+            bzl-            bzla+           bzla-
bzlr+           bzlr-           bzlrl+          bzlrl-
```

条件レジスター論理命令の拡張ニーモニック

条件レジスター論理命令の拡張ニーモニックは、POWER ® ファミリーおよび PowerPC ® で使用できます。

条件レジスター論理命令の拡張ニーモニックは、POWER ® ファミリーおよび PowerPC ® で使用できます。これらの拡張ニーモニックは、 **com** アセンブリー・モードです。 条件レジスター論理命令を使用して、特定の条件レジスター・ビットに対して以下の操作を実行することができます。

- ビットを 1 に設定します。
- クリア・ビットを 0 に。
- コピー・ビット。
- ビットを反転します。

以下の表に示す拡張ニーモニックを使用すると、これらの操作を簡単にコーディングできます。

表 *10.* 条件レジスター論理命令拡張ニーモニック

:NONE.	**同等**	**意味**
crset *bx*	**creqv** *bx*、 *bx*、 *bx*	条件レジスター・セット
crclr *bx*	**crxor** *bx*、 *bx*、 *bx*	条件レジスターのクリア
移動 *BX*, 基準	**cror** *bx*, *by*, *by*	条件レジスターの移動
クレーニ *BX*, 基準	**cr かかわらず** *bx*、 *by*、 *by*	条件レジスター NOT

条件レジスター論理命令は条件レジスター・ビットに対して演算を実行するため、アセンブラーはすべての入力オペランドで式をサポートします。 シンボル名を使用して条件レジスター (CR) フィールドを示す場合、各 CR フィールドには 4 ビットがあるため、シンボル名に 4 を乗算して正しい CR ビットを取得する必要があります。

例

1. CR0: の SO ビット (ビット 3) をクリアするには、次のようにします。

```
crclr   so
```

これは次のコマンドと等価です。

```
crxor 3, 3, 3
```

2. CR3: の EQ ビットをクリアするには、次のようにします。

```
crclr   4*cr3+eq
```

これは次のコマンドと等価です。

```
crxor   14, 14, 14
```

3. CR4 の EQ ビットを反転し、結果を CR5: の SO ビットに入れます。

```
crnot   4*cr5+so, 4*cr4+eq
```

これは次のコマンドと等価です。

```
crnor   23, 18, 18
```

固定小数点算術命令の拡張簡略記号

POWER ® ファミリーおよび PowerPC ® 用の固定小数点演算命令の拡張ニーモニック。

次の表は、POWER ® ファミリーおよび PowerPC ® 用の固定小数点算術命令の拡張ニーモニックを示しています。注記がない限り、これらの拡張ニーモニックは POWER ® ファミリーおよび PowerPC ® 用で、 **com** アセンブリー・モードです。

表 *11.* 固定小数点算術命令拡張ニーモニック

:NONE.	**同等**	**意味**
subi *rx*、*ry*、*value*	**ix** *rx, ry,* -値	**即時減算**
subis *rx*、*ry*、*value*	**addis** *rx*、*ry*、 -値	**即時シフトの減算**
subic [.] *rx*、*ry*、*value*	**addic [.]** *rx*、*ry*、 -値	**即時減算**
subc [o] [.] *rx*、*ry*、*rz*	**subfc [o] [.]** *rx*、*rz*、*ry*	**減算**
si [.] *rt*、*ra*、*value*	**ai [.]** *rt*、*ra*、 -値	**即時減算**
sub [o] [.] *rx*、*ry*、*rz*	**subf [o] [.]** *rx*、*rz*、*ry*	**減算**

注 : sub[**o**] [**.**] 拡張ニーモニックは PowerPC ® 用です。これは、基本ニーモニック **subf**[**o**] [**.**] が PowerPC ® 専用であるためです。

固定小数点比較命令の拡張ニーモニック

固定小数点比較命令の拡張ニーモニック。

以下の表に、固定小数点比較命令の拡張ニーモニックを示します。 オペランドの入力形式は、POWER ® ファミリーと PowerPC ® では異なります。 PowerPC ® の L フィールドは、64 ビット実装をサポートします。 32 ビット実装の場合、このフィールドの値は 0 でなければなりません。 POWER ® ファミリー・アーキテ

クチャーは 32 ビット実装のみをサポートするため、このフィールドは POWER® ファミリーには存在しません。 アセンブラーは、POWER® ファミリー実装の場合、このビットが 0 に設定されていることを確認します。 これらの拡張ニーモニックは、 **com** アセンブリー・モードです。

表 *12.* 固定小数点比較命令拡張ニーモニック

:NONE.	**同等**	**意味**
cmpdi *ra*、値	**cmpi 0**、**1**、*ra*、値	**ワード即時比較 (Compare Word Immediate)**
cmpwi *bf*、*ra*、*si*	**cmpi** *bf*、 **0**、*ra*、*si*	**ワード即時比較 (Compare Word Immediate)**
cmpd *ra*、*rb*	**cmp 0**、**1**、 *ra*、*rb*	**比較語**
cmpw *bf*、*ra*、*rb*	**cmp** *bf*、 **0**、*ra*、*rb*	**比較語**
cmpldi *rA*、値	**cmpli 0**、**1**、*ra*、値	**論理ワード即時比較**
cmplwi *bf*、*ra*、*ui*	**cmpli** *bf*、 **0**、*ra*、*ui*	**論理ワード即時比較**
cmpld *ra*、*rb*	**cmpl 0**、**1**、*ra*、*rb*	**論理ワードの比較**
cmplw *bf*、*ra*、*rb*	**cmpl** *bf,* **0***, ra, rb*	**論理ワードの比較**

固定小数点ロード命令の拡張ニーモニック

POWER® ファミリーおよび PowerPC® 用の固定小数点ロード命令の拡張ニーモニック。

次の表は、POWER® ファミリーおよび PowerPC® 用の固定小数点ロード命令の拡張ニーモニックを示しています。これらの拡張ニーモニックは、 **com** アセンブリー・モードです。

表 *13.* 固定小数点ロード命令拡張ニーモニック

:NONE.	**同等**	**意味**
li *rx*、*value*	**range** *rx*, 0, 値	**即時ロード**
la *rx*、*disp*(*ry*)	**ブリー** *rx*、*ry*、*disp*	**ロード・アドレス**
lil *rt*、*value*	**cal** *rt*、*value*(0)	**すぐ下にロード**
liu *rt*、*value*	**cau** *rt*, 0, 値	**即時上限のロード**
lis *rx*、値	**addis** *rx*、0、 値	**ロード即時シフト (Load Immediate エイト)**

固定小数点論理命令の拡張ニーモニック

固定小数点論理命令の拡張ニーモニック。

固定小数点論理命令の拡張ニーモニックを以下の表に示します。 これらの POWER® ファミリーおよび PowerPC® 拡張ニーモニックは、 **com** アセンブリー・モードです。

表 *14.* 固定小数点論理命令拡張ニーモニック

:NONE.	**同等**	**意味**
nop	**ori 0**、**0**、**0**	**または即時**
mr [.] *rx*、*ry*	**または [.]** *rx, ry, ry*	**または**
not [.] *rx*,*ry*	**でも [.] でも** *rx*、*ry*、*ry*	**NOR**

固定小数点トラップ命令の拡張簡略記号

固定小数点トラップ命令の拡張簡略記号には、最も便利な TO オペランド値が組み込まれています。

固定小数点トラップ命令の拡張簡略記号には、最も便利な TO オペランド値が組み込まれています。 以下の表に示すコードの標準セットは、トラップ条件の最も一般的な組み合わせに採用されています。 これらの拡張ニーモニックは、 **com** アセンブリー・モードです。

表 *15.* 固定小数点トラップ命令コード		
コード	**TO エンコード**	**意味**
LT	**10000**	LESS THAN
Le	**10100**	以下
ng	**10100**	大きくない
Eq	**00100**	等しい
GE	**01100**	以上
nl	**01100**	より小さくない
gt (gt)	「**01000**」	より大きい
NE	**11000**	等しくない
LLT	**00010**	論理的により小さい
lle (lle)	**00110**	論理的により小さいか等しい
長さ	**00110**	論理的により大きくない
Lge (Lge)	**00101**	論理的により大きいか等しい
長さ	**00101**	論理的により小さくない
lgt (lgt)	**00001**	論理的により大きい
ルネ	**00011**	論理的に等しくない
なし	**11111**	無条件

固定小数点トラップ命令用の POWER ® ファミリー拡張ニーモニックのフォーマットは、以下のとおりです。

- **t***xx* または **t***xx***i**

ここで、 *xx* は、上記の表に示されているコードのいずれかです。

ダブルワードの固定小数点トラップ命令用の 64 ビット PowerPC ® 拡張ニーモニックのフォーマットは、以下のとおりです。

- **td***xx* または **td***xx***i**

固定小数点トラップ命令用の PowerPC ® 拡張ニーモニックのフォーマットは以下のとおりです。

- **tw***xx* または **tw***xx***i**

ここで、 *xx* は、上記の表に示されているコードのいずれかです。

trap 命令は無条件トラップです。

- **トラップ (trap)**

例

1. R10 が R20: より小さい場合にトラップするには、次のようにします。

```
tlt   10, 20
```

これは次のコマンドと等価です。

```
t   16, 10, 20
```

2. R4 が 0x10: に等しい場合にトラップするには、次のようにします。

```
teqi   4, 0x10
```

これは次のコマンドと等価です。

```
ti   0x4, 4, 0x10
```

3. 無条件にトラップする場合:

```
trap
```

これは次のコマンドと等価です。

```
tw   31, 0, 0
```

4. R*X* が R*Y* と等しくない場合にトラップするには、次のようにします。

```
twnei   RX. RY
```

これは次のコマンドと等価です。

```
twi   24, RX, RY
```

5. R*X* が 0x7FF: より論理的に大きい場合にトラップするには、次のようにします。

```
twlgti   RX, 0x7FF
```

これは次のコマンドと等価です。

```
twi   1, RX, 0x7FF
```

条件レジスターに移動するための拡張簡略記号 mtcr

mtcr (条件レジスターへの移動) 拡張簡略記号は、汎用レジスター (GPR) の下位 32 ビットの内容をコピーします。

mtcr (Move to Condition Register) 拡張ニーモニックは、 **mfcr** 命令と同じスタイルを使用して、汎用レジスター (GPR) の下位 32 ビットの内容を条件レジスターにコピーします。

拡張簡略記号 **mtcr** *Rx* は、命令 **mtcrf** 0xFF、*Rx* に相当します。

この拡張ニーモニックは、 **com** アセンブリー・モードです。

特殊目的のレジスターから、または特殊目的のレジスターへの移動に関する拡張ニーモニック

特殊目的レジスターからの移動または特殊目的レジスターへの移動の拡張ニーモニック。

この項目では、以下の拡張ニーモニックについて説明します。

POWER® ファミリー用の mfspr 拡張ニーモニック

POWER® ファミリー用の mfspr 拡張ニーモニック

表 *16. POWER®* ファミリー用の *mfspr* 拡張ニーモニック			
:NONE.	**同等**	**privileged**	**SPR 名**
mfxer *rt*	**mfspr** *rt*、**1**	いいえ	XER
mflr *rt*	**mfspr** *rt*、**8**	いいえ	LR
mfctr *rt*	**mfspr** *rt*、**9**	いいえ	CTR
mfmq *rt*	**mfspr** *rt*、**0**	いいえ	MQ
mfrtcu *rt*	**mfspr** *rt*、**4**	いいえ	RTCU (R)
mfrtcl *rt*	**mfspr** *rt*、**5**	いいえ	RTCL (R)
mfdec *rt*	**mfspr** *rt*、**6**	いいえ	DEC
mftid *rt*	**mfspr** *rt*、**17**	はい	TID
mfdsisr *rt*	**mfspr** *rt*、**18**	はい	DSISR
mfdar *rt*	**mfspr** *rt*、**19**	はい	DAR
mfsdr0 *rt*	**mfspr** *rt*、**24**	はい	SDR0
mfsdr1 *rt*	**mfspr** *rt*、**25**	はい	SDR1
mfsrr0 *rt*	**mfspr** *rt*、**26**	はい	SRR0
mfsrr1 *rt*	**mfspr** *rt*、**27**	はい	SRR1

POWER® ファミリー用の mtspr 拡張ニーモニック

POWER® ファミリー用の mtspr 拡張ニーモニック

表 *17. POWER®* ファミリー用の *mtspr* 拡張ニーモニック			
:NONE.	**同等**	**privileged**	**SPR 名**
mfxer *rs*	**mtspr 1**、*rs*	いいえ	XER
mflr *rs*	**mtspr 8**、*rs*	いいえ	LR
mtctr *rs*	**mtspr 9**、*rs*	いいえ	CTR
mtmq *rs*	**mtspr 0**、*rs*	いいえ	MQ
mtrtcu *rs*	**mtspr 20**、*rs*	はい	RTCU (R)
mtrtcl *rs*	**mtspr 21**、*rs*	はい	RTCL (R)
mtdec *rs*	**mtspr 22**、*rs*	はい	DEC
mttid *rs*	**mtspr 17**、*rs*	はい	TID
mtdsisr *rs*	**mtspr 18**、*rs*	はい	DSISR
mtdar *rs*	**mtspr 19**、*rs*	はい	DAR
mtsdr0 *rs*	**mtspr 24**、*rs*	はい	SDR0
mtsdr1 *rs*	**mtspr 25**、*rs*	はい	SDR1
mtsrr0 *rs*	**mtspr 26**、*rs*	はい	SRR0

表 *17. POWER* ® ファミリー用の *mtspr* 拡張ニーモニック *(続き)*			
:NONE.	**同等**	**privileged**	**SPR 名**
mtsrr1 *rs*	**mtspr 27**、*rs*	はい	SRR1

PowerPC ® 用の mfspr 拡張ニーモニック

PowerPC ® 用の mfspr 拡張ニーモニック

表 *18. PowerPC* ® 用の *mfspr* 拡張ニーモニック			
:NONE.	**同等**	**privileged**	**SPR 名**
mfxer *rt*	**mfspr** *rt*、**1**	いいえ	XER
mflr *rt*	**mfspr** *rt*、**8**	いいえ	LR
mfctr *rt*	**mfspr** *rt*、**9**	いいえ	CTR
mfdsisr *rt*	**mfspr** *rt*、**18**	はい	DSISR
mfdar *rt*	**mfspr** *rt*、**19**	はい	DAR
mfdec *rt*	**mfspr** *rt*、**22**	はい	DEC
mfsdr1 *rt*	**mfspr** *rt*、**25**	はい	SDR1
mfsrr0 *rt*	**mfspr** *rt*、**26**	はい	SRR0
mfsrr1 *rt*	**mfspr** *rt*、**27**	はい	SRR1
mfsprg *rt*、**0**	**mfspr** *rt*、**272**	はい	SPRG0
mfsprg *rt*、**1**	**mfspr** *rt*、**273**	はい	SPRG1
mfsprg *rt*、**2**	**mfspr** *rt*、**274**	はい	SPRG2
mfsprg *rt*、**3**	**mfspr** *rt*、**275**	はい	SPRG3
m ーズ *rt*	**mfspr** *rt*、**282**	はい	EAR
mfpvr *rt*	**mfspr** *rt*、**287**	はい	PVR
mfibatu *rt*、**0**	**mfspr** *rt*、**528**	はい	IBAT0U
mfibatl *rt*、**1**	**mfspr** *rt*、**529**	はい	IBAT0L
mfibatu *rt*、**1**	**mfspr** *rt*、**530**	はい	IBAT1U
mfibatl *rt*、**1**	**mfspr** *rt*、**531**	はい	IBAT1L
mfibatu *rt*、**2**	**mfspr** *rt*、**532**	はい	IBAT2U
mfibatl *rt*、**2**	**mfspr** *rt*、**533**	はい	IBAT2L
mfibatu *rt*、**3**	**mfspr** *rt*、**534**	はい	IBAT3U
mfibatl *rt*、**3**	**mfspr** *rt*、**535**	はい	IBAT3L
mfdbatu *rt*、**0**	**mfspr** *rt*、**536**	はい	DBAT0U
mfdbatl *rt*、**0**	**mfspr** *rt*、**537**	はい	DBAT0L
mfdbatu *rt*、**1**	**mfspr** *rt*、**538**	はい	DBAT1U
mfdbatl *rt*、**1**	**mfspr** *rt*、**539**	はい	DBAT1L
mfdbatu *rt*、**2**	**mfspr** *rt*、**540**	はい	DBAT2U

表 *18. PowerPC ® 用の mfspr 拡張ニーモニック (続き)*			
:NONE.	**同等**	**privileged**	**SPR 名**
mfdbatl *rt*、**2**	**mfspr** *rt*、**541**	はい	DBAT2L
mfdbatu *rt*、**3**	**mfspr** *rt*、**542**	はい	DBAT3U
mfdbatl *rt*、**3**	**mfspr** *rt*、**543**	はい	DBAT3L

注 : mfdec 命令は、PowerPC ® の特権命令です。 PowerPC ® でのこの命令のエンコードは、POWER ® ファミリーのエンコードとは異なります。 この命令については、 **mfspr** (特殊目的レジスターからの移動) 命令を参照してください。 POWER ® ファミリーと PowerPC ® Instructions のうち、同じ Op コードを使用する場合の違いは、POWER ® ファミリーと PowerPC ® の違いを要約したものです。

PowerPC ® 用の mtspr 拡張ニーモニック

PowerPC ® 用の mtspr 拡張ニーモニック

表 *19. PowerPC ® 用の mtspr 拡張ニーモニック*			
:NONE.	**同等**	**privileged**	**SPR 名**
mtxer *rs*	**mtspr 1**、*rs*	**no**	**XER**
mtlr *rs*	**mtspr 8**、*rs*	**no**	**LR**
mtctr *rs*	**mtspr 9**、*rs*	**no**	**CTR**
mtdsisr *rs*	**mtspr 19**、*rs*	**yes**	**DSISR**
mtdar *rs*	**mtspr 19**、*rs*	**yes**	**DAR**
mtdec *rs*	**mtspr 22**、*rs*	**yes**	**DEC**
mtsdr1 *rs*	**mtspr 25**、*rs*	**yes**	**SDR1**
mtsrr0 *rs*	**mtspr 26**、*rs*	**yes**	**SRR0**
mtsrr1 *rs*	**mtspr 27**、*rs*	**yes**	**SRR1**
mtsprg 0、*rs*	**mtspr 272**、*rs*	**yes**	**SPRG0**
mtsprg 1、*rs*	**mtspr 273**、*rs*	**yes**	**SPRG1**
mtsprg 2、*rs*	**mtspr 274**、*rs*	**yes**	**SPRG2**
mtsprg 3、*rs*	**mtspr 275**、*rs*	**yes**	**SPRG3**
mtear *rs*	**mtspr 282**、*rs*	**yes**	**EAR**
mttbl *rs* (または mttb *rs*)	**mtspr 284**、*rs*	**yes**	**TBL**
mttbu *rs*	**mtspr 285**、*rs*	**yes**	**TBU**
mtibatu 0、*rs*	**mtspr 528**、*rs*	**yes**	**IBAT0U**
mtibatl 0、*rs*	**mtspr 529**、*rs*	**yes**	**IBAT0L**
mtibatu 1、*rs*	**mtspr 530**、*rs*	**yes**	**IBAT1U**
mtibatl 1,*rs*	**mtspr 531**、*rs*	**yes**	**IBAT1L**
mtibatu 2、*rs*	**mtspr 532**、*rs*	**yes**	**IBAT2U**
mtibatl 2、*rs*	**mtspr 533**、*rs*	**yes**	**IBAT2L**
mtibatu 3、*rs*	**mtspr 534**、*rs*	**yes**	**IBAT3U**

表 *19. PowerPC ® 用の mtspr 拡張ニーモニック (続き)*			
:NONE.	**同等**	**privileged**	**SPR 名**
mtibatl 3、*rs*	**mtspr 535**、*rs*	**yes**	**IBAT3L**
mtdbatu 0、*rs*	**mtspr 536**、*rs*	**yes**	**DBAT0U**
mtdbatl 0、*rs*	**mtspr 537**、*rs*	**yes**	**DBAT0L**
mtdbatu 1、*rs*	**mtspr 538**、*rs*	**yes**	**DBAT1U**
mtdbatl 1、*rs*	**mtspr 539**、*rs*	**yes**	**DBAT1L**
mtdbatu 2、*rs*	**mtspr 540**、*rs*	**yes**	**DBAT2U**
mtdbatl 2、*rs*	**mtspr 541**、*rs*	**yes**	**DBAT2L**
mtdbatu 3、*rs*	**mtspr 542**、*rs*	**yes**	**DBAT3U**
mtdbatl 3、*rs*	**mtspr 543**、*rs*	**yes**	**DBAT3L**

注 : mfdec 命令は、PowerPC ® の特権命令です。 PowerPC ® でのこの命令のエンコードは、POWER ® ファミリーのエンコードとは異なります。

PowerPC ® 601 RISC マイクロプロセッサー用の mfspr 拡張ニーモニック

PowerPC ® 601 RISC マイクロプロセッサー用の mfspr 拡張ニーモニック

表 *20. PowerPC ® 601 RISC* マイクロプロセッサー用の *mfspr* 拡張ニーモニック			
:NONE.	**同等**	**privileged**	**SPR 名**
mfmq *rt*	**mfspr** *rt*、0	**no**	**MQ**
mfxer *rt*	**mfspr** *rt*、1	**no**	**XER**
mfrtcu *rt*	**mfspr** *rt*、4	**no**	**RTCU (RTCU)**
mfrtcl *rt*	**mfspr** *rt*、5	**no**	**RTCL (RTCL)**
mfdec *rt*	**mfspr** *rt*、6	**no**	**DEC**
mflr *rt*	**mfspr** *rt*、8	**no**	**LR**
mfctr *rt*	**mfspr** *rt*、9	**no**	**CTR**
mfdsisr *rt*	**mfspr** *rt*、18	**yes**	**DSISR**
mfdar *rt*	**mfspr** *rt*、19	**yes**	**DAR**
mfsdr1 *rt*	**mfspr** *rt*、25	**yes**	**SDR1**
mfsrr0 *rt*	**mfspr** *rt*、26	**yes**	**SRR0**
mfsrr1 *rt*	**mfspr** *rt*、27	**yes**	**SRR1**
mfsprg *rt*、0	**mfspr** *rt*、272	**yes**	**SPRG0**
mfsprg *rt*、1	**mfspr** *rt*、273	**yes**	**SPRG1**
mfsprg *rt*、2	**mfspr** *rt*、274	**yes**	**SPRG2**
mfsprg *rt*、3	**mfspr** *rt*、275	**yes**	**SPRG3**
m ーズ *rt*	**mfspr** *rt*、282	**yes**	**EAR**
mfpvr *rt*	**mfspr** *rt*、287	**yes**	**PVR**

PowerPC ® 601 RISC マイクロプロセッサー用の mtspr 拡張ニーモニック

PowerPC ® 601 RISC マイクロプロセッサー用の mtspr 拡張ニーモニック

表 *21. PowerPC ® 601 RISC* マイクロプロセッサー用の *mtspr* 拡張ニーモニック			
:NONE.	**同等**	**privileged**	**SPR 名**
mtmq *rs*	**mtspr 0**、*rs*	**no**	**MQ**
mtxer *rs*	**mtspr 1**、*rs*	**no**	**XER**
mtlr *rs*	**mtspr 8**、*rs*	**no**	**LR**
mtctr *rs*	**mtspr 9**、*rs*	**no**	**CTR**
mtdsisr *rs*	**mtspr 18**、*rs*	**yes**	**DSISR**
mtdar *rs*	**mtspr 19**、*rs*	**yes**	**DAR**
mtrtcu *rs*	**mtspr 20**、*rs*	**yes**	**RTCU (RTCU)**
mtrtcl *rs*	**mtspr 21**、*rs*	**yes**	**RTCL (RTCL)**
mtdec *rs*	**mtspr 22**、*rs*	**yes**	**DEC**
mtsdr1 *rs*	**mtspr 25**、*rs*	**yes**	**SDR1**
mtsrr0 *rs*	**mtspr 26**、*rs*	**yes**	**SRR0**
mtsrr1 *rs*	**mtspr 27**、*rs*	**yes**	**SRR1**
mtsprg 0、*rs*	**mtspr 272**、*rs*	**yes**	**SPRG0**
mtsprg 1、*rs*	**mtspr 273**、*rs*	**yes**	**SPRG1**
mtsprg 2、*rs*	**mtspr 274**、*rs*	**yes**	**SPRG2**
mtsprg 3、*rs*	**mtspr 275**、*rs*	**yes**	**SPRG3**
mtear *rs*	**mtspr 282**、*rs*	**yes**	**EAR**

32 ビット固定小数点回転命令およびシフト命令の拡張ニーモニック

32 ビットの固定小数点回転命令およびシフト命令の拡張ニーモニック。

拡張ニーモニックのセットは、抽出、挿入、回転、シフト、クリア、および左シフトと左シフトのクリア操作用に提供されています。 この記事では、以下について説明します。

代替入力形式

POWER ® ファミリーおよび PowerPC ® 命令の代替入力フォーマット。

代替入力フォーマットは、以下の POWER ® ファミリーおよび PowerPC ® 命令に適用されます。

POWER ® ファミリー	**PowerPC (R)**
rlimi[。]	rlwimi [**.**]
rlinm[。]	rlwinm [**.**]
rlnm[**.**]	rlwnm [**.**]
rlmi[**.**]	適用外

通常、これらの命令には 5 つのオペランドが必要です。 これらのオペランドは次のとおりです。

RA、 *RS*、 *SH*、 *MB*、 *ME*

MB はマスク内の値が 1 の最初のビットを示し、*ME* はマスク内の値が 1 の最後のビットを示します。 アセンブラーは、以下のオペランド形式をサポートします。

```
RA, RS, SH, BM
```

BM はマスク自体です。 アセンブラーは、命令の *BM* オペランドから *MB* および *ME* オペランドを生成します。 アセンブラーは、最初に *BM* オペランドを検査します。 無効な *BM* が入力されると、エラー 78 が報告されます。

有効なマスクは、値 z0 を持つゼロ個以上のビットによって囲まれた値 1 を持つ単一の一連の (1 つ以上の) ビットとして定義されます。 値が 0 のすべてのビットのマスクを指定することはできません。

有効な *32* ビット・マスクの例
有効な 32 ビット・マスクの例。

以下に、有効な 32 ビット・マスクの例を示します。

```
                          0            15             31
                          |             |              |
-------------------------------------------------------------
MB = 0       ME = 31      11111111111111111111111111111111
MB = 0       ME = 0       10000000000000000000000000000000
MB = 0       ME = 22      11111111111111111111110000000000
MB = 12      ME = 25      00000000000011111111111110000000
-------------------------------------------------------------
MB = 22      ME = 31      00000000000000000000011111111111
MB = 29      ME = 6       11111110000000000000000000000111
```

有効ではない *32* ビット・マスクの例
有効ではない 32 ビット・マスクの例。

以下に、無効な 32 ビット・マスクの例を示します。

```
0             15             31
|              |              |
00000000000000000000000000000000
01010101010101010101010101010101
00000000000001111000011000000000
11111100000111111111111111000000
```

POWER ® ファミリーおよび PowerPC ® 向けの 32 ビット・ローテートおよびシフト拡張ニーモニック

回転命令とシフト命令の拡張ニーモニックは、POWER ® ファミリーと PowerPC ® の交差領域にあります。

回転命令およびシフト命令の拡張ニーモニックは、POWER ® ファミリーと PowerPC ® 交差域 (**com** アセンブリー・モード) にあります。 ローテートおよびシフト拡張簡略記号のセットは、以下の操作を提供します。

項目	説明
抽出	ソース・レジスターのビット位置 *b* から始まる *n* ビットのフィールドを選択します。 このフィールドは、ターゲット・レジスター内で右寄せまたは左寄せされます。 ターゲット・レジスターの他のビットはすべて 0 にクリアされます。

項目	説明
挿入	ソース・レジスター内の *n* ビットの左寄せまたは右寄せフィールドを選択します。 このフィールドは、ターゲット・レジスターのビット位置 *b* から挿入されます。 ターゲット・レジスターの他のビットは変更されません。 ダブルワードで操作する場合、左揃えのフィールドを挿入するための拡張簡略記号は提供されません。これは、そのような挿入には複数の命令が必要であるためです。
回転	マスキングを行わずに、レジスターの内容を右または左 *n* ビットに回転させます。
Shift	レジスターの内容を右または左 *n* ビットにシフトします。 空のビットは 0 (論理シフト) にクリアされます。
消去	レジスターの左端または右端の *n* ビットを 0 にクリアします。
左にクリアして左にシフト	レジスターの左端の *b* ビットをクリアしてから、 *n* ビット分だけレジスターをシフトします。 この操作は、既知の負でない配列指標をエレメントの幅でスケーリングするために使用できます。

以下の表に、回転およびシフト拡張簡略記号を示します。 *N* オペランドは、抽出、挿入、回転、またはシフトするビットの数を指定します。 拡張簡略記号が基本簡略記号にマップされるときに式が導入されるため、式の結果が *SH*、 *MB*、または *ME* オペランドでオーバーフローを引き起こさないようにするために、いくつかの制限が課されます。

以前のバージョンの AIX®との互換性を維持するために、 *n* の値は 0 に制限されていません。 *n* が 0 の場合、アセンブラーは 32-*n* を 0 の値として扱います。

表 *22. PowerPC ®* 用の *32* ビット回転およびシフト拡張ニーモニック

操作	**:NONE.**	**同等**	**制約事項**
抽出して即時に左揃え	**extlwi** *RA*、*RS*、*n*、*b*	**rlwinm** *RA*、*RS*、*b*、 **0**、*n*-**1**	32> *n* > 0
即時に抽出して右揃え	**extrwi** *RA*、*RS*、*n*、*b*	**rlwinm** *RA*、*RS*、*b*+*n*、**32-***n*、 **31**	32> *n* > 0 & *b*+*n* = < 32
左から即時に挿入	**inslwi** *RA*、*RS*、*n*、*b*	**rlwinm** *RA*、*RS*、 **32-***b*、*b*、*(b + n)*-**1**	*b*+*n* < =32 & 32>*n* > 0 & 32> *b* > = 0
右から即時に挿入	**insrwi** *RA*、*RS*、*n*、*b*	**rlwinm** *RA*、*RS*、 **32-***(b + n)*、*b*、*(b + n)*-**1**	*b*+*n* < = 32 & 32>*n* > 0
即時に左に回転	**rotlwi** *RA*、*RS*、*n*	**rlwinm** *RA*、*RS*、*n*、 **0**、**31**	32> *n* > = 0
即時に右に回転	**rotrwi** *RA*、*RS*、*n*	**rlwinm** *RA*、*RS*、 **32-***n*、**0**、**31**	32> *n* > = 0
左に回転	**rotlw** *RA*、*RS*、*b*	**rlwinm** *RA*、*RS*、*RB*、 **0**、**31**	なし
即時シフト	**slwi** *RA*、*RS*、*n*	**rlwinm** *RA*、*RS*、*n*、 **0**、**31-***n*	32> *n* > = 0
右に即時シフト	**srwi** *RA*、*RS*、*n*	**rlwinm** *RA*、*RS*、 **32-***n*、*n*、**31**	32> *n* > = 0
左方即時消去	**clrlwi** *RA*、*RS*、*n*	**rlwinm** *RA*、*RS*、 **0**、*n*、**31**	32> *n* > = 0

表 *22. PowerPC* ® 用の *32* ビット回転およびシフト拡張ニーモニック *(*続き*)*			
操作	**:NONE.**	**同等**	**制約事項**
右方即時消去	**clrrwi** *RA*、*RS*、*n*	**rlwinm** *RA*、*RS*、 **0**、**0**、**31-***n*	32> *n* > = 0
左にクリアして即時に左にシフト	**clrslwi** *RA*、*RS*、*b*、*n*	**rlwinm** *RA*、*RS*、*b-n*、**31-***n*	*b-n* > = 0 & 32> *n* > = 0 & 32> *b*> = 0

注 :

1. POWER ® ファミリーでは、ニーモニック **slwi**[**.**] は **スリ**[**.**] です。 ニーモニック **srwi**[**.**] は **sri**[**.**] です。
2. これらの拡張ニーモニックはすべて、final でコーディングできます。 (ピリオド) を使用すると、基礎となる命令に Rc ビットが設定されます。

例

POWER ® ファミリーおよび PowerPC ® 用の 32 ビット回転およびシフト拡張ニーモニックの例

1. レジスター *RY* の符号ビット (ビット 31) を抽出し、結果の右寄せをレジスター *RX* に入れます。

```
extrwi   RX, RY, 1, 0
```

   これは次のコマンドと等価です。

```
rlwinm   RX, RY, 1, 31, 31
```

2. 例 1 で抽出したビットをレジスター *RX* の符号ビット (ビット 31) に挿入するには、次のようにします。

```
insrwi   RZ, RX, 1, 0
```

   これは次のコマンドと等価です。

```
rlwimi   RZ, RX, 31, 0, 0
```

3. レジスター *RX* の内容を 8 ビット左にシフトし、高位 32 ビットをクリアするには、次のようにします。

```
slwi   RX, RX, 8
```

   これは次のコマンドと等価です。

```
rlwinm   RX, RX, 8, 0, 23
```

4. レジスター *RY* の下位 32 ビットの上位 16 ビットをクリアし、結果をレジスター *RX* に入れ、レジスター *RX* の上位 32 ビットをクリアするには、次のようにします。

```
clrlwi   RX, RY, 16
```

   これは次のコマンドと等価です。

```
rlwinm   RX, RY, 0, 16, 31
```

64 ビット固定小数点回転命令およびシフト命令の拡張ニーモニック

64 ビットの固定小数点回転命令およびシフト命令の拡張ニーモニック。

拡張ニーモニックのセットは、抽出、挿入、回転、シフト、クリア、および左シフトと左シフトのクリア操作用に提供されています。 この記事では、以下について説明します。

- 代替入力形式
- 32 ビット固定小数点回転命令およびシフト命令の拡張ニーモニック

代替入力形式

POWER ® ファミリーおよび PowerPC ® 命令に適用される代替入力フォーマット。

代替入力フォーマットは、以下の POWER ® ファミリーおよび PowerPC ® 命令に適用されます。

POWER ® ファミリー	**PowerPC (R)**
rlimi[。]	rlwimi [**.**]
rlinm[。]	rlwinm [**.**]
rlnm[**.**]	rlwnm [**.**]
rlmi[**.**]	適用外

通常、これらの命令には 5 つのオペランドが必要です。 これらのオペランドは次のとおりです。

RA、 *RS*、 *SH*、 *MB*、 *ME*

MB はマスク内の値が 1 の最初のビットを示し、*ME* はマスク内の値が 1 の最後のビットを示します。 アセンブラーは、以下のオペランド形式をサポートします。

```
RA, RS, SH, BM
```

BM はマスク自体です。 アセンブラーは、命令の *BM* オペランドから *MB* および *ME* オペランドを生成します。 アセンブラーは、最初に *BM* オペランドを検査します。 無効な *BM* が入力されると、エラー 78 が報告されます。

有効なマスクは、値 z0 を持つゼロ個以上のビットによって囲まれた値 1 を持つ単一の一連の (1 つ以上の) ビットとして定義されます。 値が 0 のすべてのビットのマスクを指定することはできません。

POWER ® ファミリーおよび PowerPC ® 向けの 64 ビット・ローテートおよびシフト拡張ニーモニック

POWER ® ファミリーおよび PowerPC ® 向けの 64 ビット・ローテートおよびシフト拡張ニーモニック

回転命令およびシフト命令の拡張ニーモニックは、POWER ® ファミリーと PowerPC ® 交差域 (**com** アセンブリー・モード) にあります。 ローテートおよびシフト拡張簡略記号のセットは、以下の操作を提供します。

項目	**説明**
抽出	ソース・レジスターのビット位置 *b* から始まる *n* ビットのフィールドを選択します。 このフィールドは、ターゲット・レジスター内で右寄せまたは左寄せされます。 ターゲット・レジスターの他のビットはすべて 0 にクリアされます。
挿入	ソース・レジスター内の *n* ビットの左寄せまたは右寄せフィールドを選択します。 このフィールドは、ターゲット・レジスターのビット位置 *b* から挿入されます。 ターゲット・レジスターの他のビットは変更されません。 ダブルワードで操作する場合、左揃えのフィールドを挿入するための拡張簡略記号は提供されません。これは、そのような挿入には複数の命令が必要であるためです。

項目	説明
回転	マスキングを行わずに、レジスターの内容を右または左 *n* ビットに回転させます。
Shift	レジスターの内容を右または左 *n* ビットにシフトします。空のビットは 0 (論理シフト) にクリアされます。
消去	レジスターの左端または右端の *n* ビットを 0 にクリアします。
左にクリアして左にシフト	レジスターの左端の *b* ビットをクリアしてから、 *n* ビット分だけレジスターをシフトします。 この操作は、既知の負でない配列指標をエレメントの幅でスケーリングするために使用できます。

以下の表に、回転およびシフト拡張簡略記号を示します。 *N* オペランドは、抽出、挿入、回転、またはシフトするビットの数を指定します。 拡張簡略記号が基本簡略記号にマップされるときに式が導入されるため、式の結果が *SH*、 *MB*、または *ME* オペランドでオーバーフローを引き起こさないようにするために、いくつかの制限が課されます。

以前のバージョンの AIX®との互換性を維持するために、 *n* の値は 0 に制限されていません。 *n* が 0 の場合、アセンブラーは 32-*n* を 0 の値として扱います。

表 *23. PowerPC ®* 用の *63* ビット回転ニーモニックとシフト拡張ニーモニック

操作	:NONE.	同等	制約事項
ダブルワードを抽出し、即時に右寄せします	**extrdi** *RA*、*RS*、*n*、*b*	**rldicl** *RA*、*RS*、*b + n*、**64** *-n*	*n* > 0
ダブルワードを左方に即時に回転	**rotldi** *RA*、*RS*、*n*	**rldicl** *RA*、*RS*、*n*、 **0**	なし
ダブルワードを右方に即時に回転	**rotrdi** *RA*、*RS*、*n*	**rldicl** *RA*、*RS*、 64 *-n*、**0**	なし
ダブルワードを右方に即時に回転	**srdi** *RA*、*RS*、*n*	**rldicl** *RA*、*RS*、 64 *-n*、*n*	*n* < 64
左方ダブルワード即時消去	**clrldi** *RA*、*RS*、*n*	**rldicl** *RA*、*RS*、 **0**、*n*	*n* < 64
ダブルワードを抽出し、即時に左寄せします。	**extldi** *RA*、*RS*、*n*、*b*	**rldicr** *RA*、*RS*、*b*、*n* -1	なし
ダブルワード即時シフト	**sldi** *RA*、*RS*、*n*	**rldicr** *RA*、*RS*、*n*、**63** *-n*	なし
右方ダブルワード即時消去	**clrrdi** *RA*、*RS*、*n*	**rldicr** *RA*、*RS*、 **0**、**63** *-n*	なし
左ダブルワードをクリアして即時に左にシフト	**clrlsldi** *RA*、*RS*、*b*、*n*	**rldic** *RA*、*RS*、*n*、*b-n*	なし
右方即値からダブルワードを挿入	**insrdi** *RA*、*RS*、*n*、*b*	**rldimi** *RA*、*RS*、*64-(b + n)*、*b*	なし
ダブルワードを左に回転	**rotld** *RA*、*RS*、*RB*	**rldcl** *RA*、*RS*、*RB*、 **0**	なし

注 : これらの拡張ニーモニックはすべて、final でコーディングできます。 (ピリオド) を使用すると、基礎となる命令に Rc ビットが設定されます。

ソース・プログラムのマイグレーション

現行のアセンブリー・モードではない命令がソース・プログラムに含まれている場合、アセンブラーはエラーおよび警告を出します。

現行のアセンブリー・モードではない命令がソース・プログラムに含まれている場合、アセンブラーはエラーおよび警告を出します。 POWER® ファミリー・プログラムのソース互換性は、PowerPC® プラットフォームで維持されています。 POWER® ファミリーのすべてのユーザー指示は、オペレーティング・システムによって PowerPC® でエミュレートされます。 命令のエミュレーションは、ハードウェアがサポートする命令の実行よりもずっと低速であるため、パフォーマンス上の理由から、ハードウェアがサポートする命令を使用するようにソース・プログラムを変更することが望ましい場合があります。

「無効な命令フォーム」問題は、PowerPC® では制限が必要とされるが、POWER® ファミリーでは必要とされない場合に発生します。 アセンブラーは無効な命令形式エラーを検査しますが、これらのエラーについて **lswx** 命令を検査することはできません。 **lswx** 命令では、2 番目と 3 番目のオペランド (*RA* および *RB*) によって指定されたレジスターが、ロードされるレジスターの範囲内にないことが必要です。 これは、実行時に固定小数点例外レジスター (XER) の内容によって決定されるため、アセンブラーは **lswx** 命令に対して無効な命令フォーム・チェックを実行することができません。 実行時に、これらのエラーの中には、無音の障害の原因となるものと、中断の原因となるものがあります。 これらのエラーを除去することが望ましい場合があります。 無効な命令フォームについて詳しくは、「**検出エラー状態**」を参照してください。

mfspr および **mtspr** 命令が使用されている場合は、特殊目的レジスター (SPR) オペランドが正しくコーディングされているかどうかを検査してください。 アセンブラーでは、SPR オペランドを入力オペランドとして使用する前に、下位 5 ビットと上位 5 ビットを逆にする必要があります。 POWER® ファミリーと PowerPC® では、非特権命令用の SPR オペランドのセットが異なります。 これらのオペランドのエンコードが正しいことを確認してください。 5 つの POWER® ファミリー SPR (TID、 SDR0、 MQ、RTCU、および RTCL) は PowerPC® から除去されますが、 MQ、RTCU、および RTCL 命令は PowerPC® でエミュレートされます。これらの命令は引き続き使用できますが、エミュレーションによってパフォーマンスが低下します。 (リアルタイム・クロックまたは時間ベースの SPR にアクセスするコードの代わりに、**read_real_time** および **time_base_to_time** ルーチンを使用することができます。)

POWER® ファミリーと PowerPC® 命令の機能の違い

POWER® ファミリーと PowerPC® 命令は、POWER® ファミリーと PowerPC® プラットフォームで同じ命令コードを共有しますが、機能定義は異なります。

以下の表に、POWER® ファミリーと PowerPC® 命令をリストします。これらは、POWER® ファミリーと PowerPC® プラットフォームで同じ命令コードを共有しますが、機能定義は異なります。 **com** アセンブリー・モードでこれらの命令を使用する場合は注意してください。

表 *24. POWER® ファミリーと PowerPC® の機能の違いに関する説明*

POWER® ファミリー	**PowerPC (R)**	**説明**
dcs (dcs)	**sync**	**sync** 命令は、POWER® ファミリーの **dcs** 命令よりも広範囲の同期を PowerPC® で引き起こします。
ics	**同期 (isync)**	**isync** 命令は、POWER® ファミリーの **ics** 命令よりも PowerPC® の方が広範囲な同期を引き起こします。
svca (svca)	**sc**	POWER® ファミリーでは、MSR からの情報は CTR に保存されます。 PowerPC® では、この情報は SRR1 に保存されます。 PowerPC® は 1 つのベクトルのみをサポートします。 POWER® ファミリーでは、128 の任意の場所で命令の取り出しを続行できます。 POWER® ファミリーは、CTR の命令の下位 16 ビットを保管します。 PowerPC® は、命令の下位 16 ビットを保存しません。

表 *24. POWER ® ファミリーと PowerPC ® の機能の違いに関する説明 (続き)*		
POWER ® ファミリー	**PowerPC (R)**	**説明**
mtsri (mtsri)	**mtsnn**	POWER ® ファミリーは、 RA フィールドを使用してセグメント・レジスター番号を計算し、場合によっては有効アドレス (EA) が保管されます。 PowerPC ® には RA フィールドがなく、EA は保管されません。
lsx (lsx)	**LSWX**	ストリングの長さが 0 の場合、POWER ® ファミリーはターゲット・レジスター *RT* を変更しません。 ストリングの長さが 0 の場合、PowerPC ® はターゲット・レジスター *RT* の内容を未定義のままにします。
mfsr (mfsr)	**mfsr (mfsr)**	これは POWER ® ファミリーの非特権命令です。 これは、PowerPC ® の特権命令です。
mfmsr (mfmsr)	**mfmsr (mfmsr)**	これは POWER ® ファミリーの非特権命令です。 これは、PowerPC ® の特権命令です。
mfdec (mfdec)	**mfdec (mfdec)**	**mfdec** 命令は POWER ® ファミリーでは非特権命令ですが、PowerPC ® では特権命令になります。その結果、 **mfdec** 命令の DEC エンコード番号は、POWER ® ファミリーと PowerPC ® では異なります。
マフス	**マフス**	POWER ® ファミリーは、結果の上位 32 ビットを 0xFFFF FFFF に設定します。 PowerPC ® では、結果の上位 32 ビットは未定義です。

アセンブラーの PowerPC ® 固有の機能について詳しくは、1 ページの『AIX® アセンブラーの機能』 を参照してください。

同じ命令コードを使用する POWER ® ファミリーと PowerPC ® 命令の違い

同じ命令コードを使用する POWER ® ファミリーと PowerPC ® 命令の違い

このセクションでは、以下について説明します。

命令コード、ニーモニック、および機能が同じ命令

これらの命令は、POWER ® ファミリーと PowerPC ® で使用できます。これらの命令は、同じ命令コードとニーモニックを共有し、POWER ® ファミリーと PowerPC ® で同じ機能を持ちます。

以下の命令は POWER ® ファミリーと PowerPC ® で使用できます。これらの命令は、同じ命令コードとニーモニックを共有し、POWER ® ファミリーと PowerPC ® で同じ機能を持ちますが、入力オペランドの形式は異なります。

- **cmp - 2 つのファイルを比較する**
- **cmpi (cmpi)**
- **cmpli (cmpli)**
- **cmpl**

POWER ® ファミリーの場合の入力オペランドの形式は次のとおりです。

BF、 *RA*、 *SI* | *RB* | *UI*

PowerPC ® の入力オペランドの形式は次のとおりです。

BF、 *L*、 *RA*、 *SI* | *RB* | *UI*

アセンブラーは、POWER® ファミリーと PowerPC® ではこれらを同じ命令として扱いますが、入力オペランドの形式は異なります。 *L* オペランドは 1 ビットです。 POWER® ファミリーの場合、アセンブラーはこのビットを 0 に事前設定します。 32 ビット PowerPC® プラットフォームの場合、このビットは 0 に設定する必要があります。0 に設定しないと、無効な命令フォームが生成されます。

同じ命令コードと機能を持つ命令

同じ命令コードと機能を共有するが、ニーモニックと入力オペランドの形式が異なる、POWER® ファミリーと PowerPC® で使用可能な命令。

以下の表にリストされている命令は、POWER® ファミリーと PowerPC® で使用できます。これらの命令は、同じ命令コードと機能を共有しますが、ニーモニックと入力オペランドの形式は異なります。 同じバイナリー・コードが生成されるため、アセンブラーは POWER® ファミリー /PowerPC® 交差域にそれらを配置します。 **-s** オプションを使用する場合、相互参照は行われません。POWER® ファミリーから PowerPC® へ、またはその逆にマイグレーションする場合は、ソース・コードを変更する必要があるためです。

表 *25.* 同じ運用コードおよび機能を持つ命令

POWER® ファミリー	**PowerPC (R)**
cal	**アディ**
mtsri (mtsri)	**mtsnn**
svca (svca)	**sc**
cau	**アドレス**

注 :

1. **lil** は **cal** の拡張ニーモニックであり、 **li** は **ニル**の拡張ニーモニックです。 命令コード、関数、および入力オペランドの形式は同じであるため、アセンブラーは **lil** と **li** の相互参照を提供します。
2. **liu** は **cau** の拡張ニーモニックであり、**lis** は **addis** の拡張ニーモニックです。 入力オペランドの形式が異なるため、アセンブラーは **liu** と **lis** の相互参照を提供しません。
3. **cau** 命令の即値は、16 ビットの符号なし整数ですが、 **addis** 命令の即値は、16 ビットの符号付き整数です。 アセンブラーは、 UI フィールドに対して (0, 65535) 値範囲検査を実行し、 SI フィールドに対して (-32768, 32767) 値範囲検査を実行します。

   cau 命令と **addis** 命令のソース互換性を維持するために、アセンブラーは **addis** 命令の値範囲検査を (-65536, 65535) に拡張します。 符号ビットは無視され、アセンブラーは即時値が 16 ビットに収まるだけであることを確認します。 この拡張は、32 ビット実装の動作には影響しません。

   64 ビット実装の場合は、ビット 32 が設定されていると、64 ビット汎用レジスター (GPR) の上位 32 ビットに伝搬されます。 したがって、32 ビット・モードの **addis** 命令で範囲内の即時値 (32768、65535) または (-65536、-32767) が使用される場合、この即時値は 64 ビット・モードに直接移植できません。

mfdec 命令

アセンブリー・モードごとの **mfdec** 命令のアセンブラー処理。

DEC (減分) 特殊目的レジスターからの移行は、PowerPC® では特権が付与されますが、POWER® ファミリーでは特権が付与されません。 レジスターを指定する命令フィールドの 1 ビットは、特権命令の場合は 1、非特権命令の場合は 0 です。 その結果、PowerPC® と POWER® ファミリーでは、 **mfdec** 命令の DEC SPR のエンコード番号の値が異なります。 DEC エンコード番号は、PowerPC® の場合は 22、POWER® ファミリーの場合は 6 です。 **mfdec** 命令を使用する場合、アセンブラーは現在のアセンブリー・モードに基づいて DEC エンコードを決定します。 以下のリストは、アセンブリー・モード値ごとの **mfdec** 命令のアセンブラー処理を示しています。

- アセンブリー・モードが **pwr**、 **pwr2**、または **601** の場合、DEC エンコードは 6 です。
- アセンブリー・モードが **ppc**、 **603**、または **604** の場合、DEC エンコードは 22 です。

- デフォルトのアセンブリー・モード (POWER® ファミリー /PowerPC® の非互換性エラーを指示警告として扱う) が使用される場合、DEC エンコードは 6 です。 説明警告 158 は、オブジェクト・コードの生成に DEC SPR エンコード 6 が使用されることを報告します。 この警告は、 **-W** フラグを使用して抑止できます。
- アセンブリー・モードが **any** の場合、DEC エンコードは 6 です。 **-w** フラグを使用すると、警告メッセージ (158) により、オブジェクト・コードの生成に DEC SPR エンコード 6 が使用されることが報告されます。
- アセンブリー・モードが **com** の場合、 **mfdec** 命令がサポートされていないことを示すエラー・メッセージが報告されます。 オブジェクト・コードは生成されません。 この状況では、 **mfspr** 命令を使用して DEC 番号をエンコードする必要があります。

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

拡張ニーモニックの変更

POWER® ファミリーおよび PowerPC® 用の拡張ニーモニック。

以下のリストは、POWER® ファミリーおよび PowerPC® 用に追加された拡張ニーモニックを示しています。 POWER® ファミリーと PowerPC® 拡張ニーモニックの基本的なニーモニックがこの領域にある場合、アセンブラーは、POWER® ファミリーと PowerPC® 拡張ニーモニックをすべて POWER® ファミリーと PowerPC® の交差領域に配置します。 拡張ニーモニックは、マイグレーションの目的でのみ POWER® ファミリーと PowerPC® 用に分離されています。 詳しくは、82 ページの『拡張命令ニーモニック』を参照してください。

通信モードでの拡張ニーモニック

分岐命令、論理命令、ロード命令、および算術命令の拡張ニーモニック。

ブランチ条件付き命令用の以下の PowerPC® 拡張ニーモニックが追加されました。

- **bdzt (bdzt)**
- **bdzta (bdzta)**
- **bdztl (bdztl)**
- **bdztla**
- **bdzf (bdzf)**
- **bdzfa (bdzfa)**
- **bdzfl**
- **bdzfla (bdzfla)**
- **bdnzt (bdnzt)**
- **bdnzta (bdnzta)**
- **bdnztl (bdnztl)**
- **bdnztla**
- **bdnzf (bdnzf)**
- **bdnzfa (bdnzfa)**
- **bdnzfl**
- **bdnzfla (bdnzfla)**
- **bdztlr (bdztlr)**
- **bdztlrl (bdztlrl)**

- **bdzflr**
- **bdzflrl**
- **bdnztlr (bdnztlr)**
- **bdnztlrl (bdnztlrl)**
- **bdnzflr**
- **bdnzflrl**
- **分**
- **ブナ**
- **bunl**
- **bunla (ブナ)**
- **bunlr**
- **ブレル (bunlrl)**
- 「**bunctr**」
- 「**bunctrl**」
- **BNU**
- **ブヌア**
- **ブヌール**
- **ブラ (bnula)**
- **ヌル (bnulr)**
- **ヌル (bnulrl)**
- **NUBNUTR**
- **NUBNUTRL (B)**

条件レジスター論理命令用の以下の PowerPC ® 拡張ニーモニックが追加されました。

- **CRSET**
- **CRCLR**
- **CRMOVE**
- **crnot (crnot)**

固定小数点ロード命令用の以下の PowerPC ® 拡張ニーモニックが追加されました。

- **Li**
- **LIS**
- **LA**

固定小数点演算命令用の以下の PowerPC ® 拡張ニーモニックが追加されました。

- **サブ I**
- **サブ**
- **サブ**

固定小数点比較命令用の以下の PowerPC ® 拡張ニーモニックが追加されました。

- **cmpwi (cmpwi)**
- **cmpw (cmpw)**
- **cmplwi (cmplwi)**
- **cmplw**

固定小数点トラップ命令用の以下の PowerPC ® 拡張ニーモニックが追加されました。

- **トラップ (trap)**

- **twlng**
- **twlngi (twlngi)**
- **twlnl (twlnl)**
- **twlnli (twlnli)**
- **twng**
- **twngi (twngi)**
- **twnl (twnl)**
- **twnli (twnli)**

固定小数点論理命令用の以下の PowerPC ® 拡張ニーモニックが追加されました。

- **nop**
- **mr**[。]
- **は[.]** ではありません。

固定小数点回転命令およびシフト命令用の以下の PowerPC ® 拡張ニーモニックが追加されました。

- **extlwi**[.]
- **追加情報**[.]
- **inslwi**[.]
- **insrwi**[.]
- **rotlw**[.]
- **rotlwi**[。]
- **rotrwi**[.]
- **clrlwi**[.]
- **clrwi**[.]
- **clrlslwi**[.]

ppc モードでの拡張ニーモニック

固定小数点演算命令の PowerPC ® 拡張ニーモニック。

ppc モードでは、固定小数点演算命令用の以下の PowerPC ® 拡張ニーモニックが追加されました。

- **サブ**

PowerPC ® から削除された POWER ® ファミリーの説明

POWER ® ファミリーの説明が PowerPC ® から削除されました。

以下の表に、PowerPC ® から削除されたものの、まだ PowerPC ® 601 RISC マイクロプロセッサーによってサポートされている POWER ® ファミリーの命令をリストします。 PowerPC 603 RISC マイクロプロセッサーや PowerPC 604 RISC マイクロプロセッサーなど、命令が含まれていないプロセッサーでこれらの命令の 1 つを実行しようとしても、 **MTRCL**、 **mtrtcu**、または **スヴラ** 命令に対してエミュレーション・サービスが提供されない場合、AIX® はこれらの命令のほとんどをエミュレートするサービスを提供します。 コードを使用して命令をエミュレートする方が、命令を実行するよりもはるかに時間がかかります。

表 *26. PowerPC ®* から削除された *POWER ®* ファミリー命令、サポート対象 *PowerPC ® 601 RISC* マイクロプロセッサー

項目	説明		
abs [o] [.]	**clcs (clcs)**	**div [o] [.]**	**div [o] [.]**
doz [o] [.]	ドジ	**lscbx [.]**	**maskg [。]**

表 *26. PowerPC ® から削除された POWER ® ファミリー命令、サポート対象 PowerPC ® 601 RISC マイクロプロセッサー (続き)*			
項目	**説明**		
maskir [.]	**mfmq (mfmq)**	**mfrtcl**	**mfrtcu (mfrtcu)**
MTMQ	**mtrtcl (mtrtcl)**	**mtrtcu (mtrtcu)**	**mul [o] [.] (mul [o] [.])**
nabs [o] [.]	**rlmi [.]**	**rrib [.]**	**sle [.]**
スレッド [.]	**sliq [.]**	**slliq [.] (slliq [.])**	**sllq [.]**
slq [.] (slq [.])	**sraiq [.] (sraiq [.])**	**sraq [.]**	**sre [.]**
srea [.] (srea [.])	**sreq [.] (sreq [.])**	**sriq [.]**	**SRLIQ [.]**
SRLQ [.]	**SRQ [.]**	**svcla (svcla)**	

注 : 拡張ニーモニックは、 **mfspr** 命令および **mtspr** 命令の拡張ニーモニックを除き、前の表には含まれていません。

以下の表に、PowerPC ® から削除され、PowerPC ® 601 RISC マイクロプロセッサーでサポートされない POWER ® ファミリーの命令をリストします。 AIX® は、これらの手順のほとんどをエミュレートするサービスを提供 していません 。 ただし、 **clf**、 **dclst**、および **dclz** 命令用にエミュレーション・サービスが提供されています。 また、 **cli** 命令はエミュレートされますが、特権モードで実行される場合に限られます。

表 *27. PowerPC ® から削除された POWER ® ファミリーの説明 (PowerPC ® 601 RISC マイクロプロセッサーではサポート対象外)*			
項目	**説明**		
clf	**cli**	**dclst (dclst)**	**dclz (dclz)**
mfsdr0	**mfsri (mfsri)**	**mftid (mftid)**	**mtsdr0**
mttid (mttid)	**ラック**[.]	**rfsvc (Rfsvc)**	**SVC**
svcl (svcl)	**TLBI**		

PowerPC ® の説明が追加されました。

PowerPC ® に追加されたが、POWER ® ファミリーには含まれていない命令。

以下の表は、PowerPC ® に追加されているが、POWER ® ファミリーには含まれていない手順をリストしています。 これらの手順は、PowerPC ® 601 RISC マイクロプロセッサーによってサポートされます。

表 *28. PowerPC ® 命令が追加されました。PowerPC ® 601 RISC マイクロプロセッサーでサポートされます。*			
項目	**説明**		
dcbf (dcbf)	**dcbi**	**dcbst (dcbst)**	**dcbt (dcbt)**
dcbtst (dcbtst)	**dcbz (dcbz)**	**divw [o] [.]**	**divwu [o] [.]**
Eeio (Eeio)	**extsb [.] (extsb [.])**	**fadd [.]**	**fdivs [.]**
fmadd [.]	**fmsubs [.] (fmsubs [.])**	**fmuls [.]**	**fnmadd [.]**
fnmsub [.]	**fsubs [.]**	**ICBI**	**Lwarx (Lwarx)**
恐怖	**mfpvr (mfpvr)**	**mfsprg**	**mfsrin (mfsrin)**
mtear (mtear)	**mtsprg (mtsprg)**	**マルチウ [.]**	**mulhwu [.] (mulhwu [.])**
stwcx。	**サブ f [o] [.]**		

注 : 拡張ニーモニックは、 **mfspr** 命令および **mtspr** 命令の拡張ニーモニックを除き、前の表には含まれていません。

以下の表は、PowerPC ® に追加されているが、POWER ® ファミリーには含まれていない手順をリストしています。 これらの手順は、PowerPC ® 601 RISC マイクロプロセッサーではサポートされていません。

表 *29. PowerPC ® 命令 (PowerPC ® 601 RISC* マイクロプロセッサーではサポートされません*)*			
項目	**説明**		
mfdbatl	**mfdbatu (mfdbatu)**	**mtdbatl**	**mtdbatu (M)**
mttb	**mttbu (mttbu)**	**mftb**	**mftbu (mftbu)**
mfibatl (Mfibatl)	**mfibatu (mfibatu)**	**mtibatl (mtibatl)**	**mtibatu (mtibatu)**

関連概念

ソース・プログラムのマイグレーション
現行のアセンブリー・モードではない命令がソース・プログラムに含まれている場合、アセンブラーはエラーおよび警告を出します。

拡張ニーモニックの変更
POWER ® ファミリーおよび PowerPC ® 用の拡張ニーモニック。

PowerPC ® から削除された POWER ® ファミリーの説明
POWER ® ファミリーの説明が PowerPC ® から削除されました。

PowerPC ® 601 RISC マイクロプロセッサーにのみ使用可能な手順
PowerPC ® 命令は、PowerPC ® 601 RISC マイクロプロセッサーでのみ使用可能です。

PowerPC ® 601 RISC マイクロプロセッサーにのみ使用可能な手順

PowerPC ® 命令は、PowerPC ® 601 RISC マイクロプロセッサーでのみ使用可能です。

以下の表に、PowerPC ® 601 RISC マイクロプロセッサーに実装されている PowerPC ® のオプションの指示をリストします。

表 *30. PowerPC ® 601 RISC* マイクロプロセッサー-固有の指示			
項目	**説明**		
エシウクス (eciwx)	**エコー (ecowx)**	**mfbatl (mfbatl)**	**mfbatu (mfbatu)**
mtbatl	**mtbatu (mtbatu)**	**TL ビー**	

注 : mfspr および **mtspr** 拡張ニーモニックを除き、拡張ニーモニックは提供されません。

簡略記号の後に分離文字を付けない分岐条件ステートメントの移行

AIX® アセンブラーは、前のバージョンのアセンブラーとは異なるいくつかのステートメントを構文解析します。

AIX® アセンブラーは、前のバージョンのアセンブラーとは異なるいくつかのステートメントを構文解析する場合があります。 この異なる構文解析は、以下のすべての条件を満たすステートメントに対してのみ可能です。

- ステートメントの簡略記号とオペランドの間に区切り文字 (スペースまたはタブ) がありません。
- 第 1 オペランドの先頭文字は、正符号 (+) または負符号 (-) です。
- ニーモニックは、分岐条件付き命令を表します。

アセンブラー・プログラムに上記のすべての条件を満たすステートメントがあり、負符号 (-) または正符号 (+) が簡略記号の一部ではなくオペランドの一部である場合、ソース・プログラムを変更する必要があります。 負符号 (-) を移動すると、ステートメントの意味が大きく変わる可能性があるため、これは特に負符号 (-) の場合に重要です。

AIX® では、アセンブラーが、ニーモニックの一部として正符号と負符号を使用する分岐予測拡張ニーモニックをサポートするように変更されたため、構文解析が異なる可能性があります。 以前のバージョンのアセンブラーでは、ニーモニックに使用できる文字は文字とピリオド (.) のみでした。

例

1. 以下のステートメントは AIX® アセンブラーによって解析され、負符号 (-) がニーモニックの一部になります (ただし、前のバージョンのアセンブラーでは、オペランドの一部として負符号 (-) が解析されています)。負符号 (-) をオペランドの一部にする場合は、このステートメントを変更する必要があります。

```
bnea-   16  #  Separator after the - , but none before
            #  Now: bnea- is a Branch Prediction Mnemonic
            #       and 16 is operand.
            #  Previously: bnea was mnemonic
            #              and -16 was operand.
```

2. 以下は、 AIX® アセンブラーが前のアセンブラーと同じように解析するいくつかのサンプル・ステートメントです (負符号 (-) はオペランドの一部として解釈されます)。

```
bnea  -16  # Separator in source program - Good practice
bnea-16    #  No separators before or after minus sign
bnea  - 16 # Separators before and after the minus sign
```

関連概念

ブランチ命令の拡張ニーモニック

アセンブラーは、さまざまなタイプのレジスター命令の拡張ニーモニックをサポートします。

命令セット

このトピックには、オペレーティング・システムのアセンブラー命令セットの参照項目が記載されています。

Power 命令セット・アーキテクチャー (ISA) の更新により、本書に記載されている情報と比較して、既存の命令が変更されたり、既存の命令が非推奨になったり、新しい命令が追加されたりする場合があります。 OpenPOWER Web サイト (https://openpowerfoundation.org/specifications/isa/) で、 Power ISA 資料の最新バージョンを参照してください。

abs (絶対) 命令

目的

汎用レジスターの内容の絶対値を取得し、その結果を別の汎用レジスターに入れます。

注: abs 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	///
21	大江
22 - 30	360
31	rc

POWER® ファミリー

abs	*RT*、*RA*
アブ	*RT*、*RA*
アブソ	*RT*、*RA*
abso:	*RT*、*RA*

説明

abs 命令は、汎用レジスター (GPR) *RA* の内容の絶対値を、ターゲット GPR *RT* に入れます。

GPR *RA* に最も負の数値 ('8000 0000') が含まれている場合、命令の結果は最も負の数値になり、OE ビットが 1 に設定されている場合は、命令は固定小数点例外レジスターのオーバーフロー・ビットを 1 に設定します。

abs 命令には 4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	**説明**			
構文形式	**オーバーフロー例外 (OE)**	**固定小数点例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
abs	0	なし	0	なし
アブ	0	なし	1	LT、GT、EQ、SO
アブソ	1	SO、OV	0	なし
abso:	1	SO、OV	1	LT、GT、EQ、SO

abs 命令の 4 つの構文形式は、常に固定小数点例外レジスターのカリー・ビット (CA) に影響します。 構文形式がオーバーフロー例外 (OE) ビットを 1 に設定する場合、命令は固定小数点例外レジスターの要約オーバーフロー (SO) およびオーバーフロー (OV) ビットに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	**説明**
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作用のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容の絶対値を取り、結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x7000 3000.
abs 6,4
# GPR 6 now contains 0x7000 3000.
```

2. 以下のコードは、GPR 4 の内容の絶対値を取り、結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xFFFF FFFF.
abs. 6,4
# GPR 6 now contains 0x0000 0001.
```

3. 以下のコードは、GPR 4 の内容の絶対値を取り、結果を GPR 6 に保管し、演算の結果を反映するように固定小数点例外レジスターに要約オーバーフロー・ビットとオーバーフロー・ビットを設定します。

```
# Assume GPR 4 contains 0xB004 3000.
abso 6,4
# GPR 6 now contains 0x4FFB D000.
```

4. 以下のコードは、GPR 4 の内容の絶対値を取り、結果を GPR 6 に保管し、演算の結果を反映するために、固定小数点例外レジスターおよび条件レジスター・フィールド 0 にサマリー・オーバーフロー・ビットとオーバーフロー・ビットを設定します。

```
# Assume GPR 4 contains 0x8000 0000.
abso. 6,4
# GPR 6 now contains 0x8000 0000.
```

add (Add) または cax (Compute Address) 命令

目的

2 つの汎用レジスターの内容を追加します。

構文

PowerPC (R)

add	*RT*、*RA*、*RB*
追加。	*RT*、*RA*、*RB*
アドド	*RT*、*RA*、*RB*
addo。	*RT*、*RA*、*RB*

POWER® ファミリー

cax (cax)	*RT*、*RA*、*RB*
cax。	*RT*、*RA*、*RB*
caxo (caxo)	*RT*、*RA*、*RB*
caxo。	*RT*、*RA*、*RB*

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21	大江
22 - 30	266
31	rc

説明

add 命令と **cax** 命令は、汎用レジスター (GPR) *RA* と GPR *RB* の内容の合計をターゲット GPR *RT* に入れます。

add 命令と **cax** 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件レジスター・フィールド 0**
add	0	なし	0	なし
追加。	0	なし	1	LT、GT、EQ、SO
アドド	1	SO、OV	0	なし
addo。	1	SO、OV	1	LT、GT、EQ、SO
cax (cax)	0	なし	0	なし
cax。	0	なし	1	LT、GT、EQ、SO
caxo (caxo)	1	SO、OV	0	なし
caxo。	1	SO、OV	1	LT、GT、EQ、SO

add 命令の 4 つの構文形式と、 **cax** 命令の 4 つの構文形式は、固定小数点例外レジスターの Carry ビット (CA) には影響しません。 構文形式がオーバーフロー例外 (OE) ビットを 1 に設定する場合、命令は固定小数点例外レジスターの要約オーバーフロー (SO) およびオーバーフロー (OV) ビットに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 6 のアドレスまたは内容を GPR 3 のアドレスまたは内容に追加し、結果を GPR 4 に保管します。

```
# Assume GPR 6 contains 0x0004 0000.
# Assume GPR 3 contains 0x0000 4000.
add 4,6,3
# GPR 4 now contains 0x0004 4000.
```

2. 以下のコードは、GPR 6 のアドレスまたは内容を GPR 3 のアドレスまたは内容に追加し、結果を GPR 4 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 6 contains 0x8000 7000.
# Assume GPR 3 contains 0x7000 8000.
 add. 4,6,3
# GPR 4 now contains 0xF000 F000.
```

3. 以下のコードは、GPR 6 のアドレスまたは内容を GPR 3 のアドレスまたは内容に追加し、結果を GPR 4 に保管し、演算の結果を反映するために固定小数点例外レジスターのサマリー・オーバーフロー・ビット、オーバーフロー・ビット、およびキャリー・ビットを設定します。

```
# Assume GPR 6 contains 0xEFFF FFFF.
# Assume GPR 3 contains 0x8000 0000.
addo 4,6,3
# GPR 4 now contains 0x6FFF FFFF.
```

4. 以下のコードは、GPR 6 のアドレスまたは内容を GPR 3 のアドレスまたは内容に追加し、結果を GPR 4 に保管し、演算の結果を反映するために固定小数点例外レジスターおよび条件レジスター・フィールド 0 のサマリー・オーバーフロー、オーバーフロー、およびキャリー・ビットを設定します。

```
# Assume GPR 6 contains 0xEFFF FFFF.
# Assume GPR 3 contains 0xEFFF FFFF.
addo. 4,6,3
# GPR 4 now contains 0xDFFF FFFE.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点アドレス計算命令
POWER® ファミリーの各種アドレス計算命令は、PowerPC® の演算命令に統合されています。

addc または a (キャライングの追加) 命令

.

目的

2 つの汎用レジスターの内容を追加し、その結果を汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21	大江
22 - 30	10
31	rc

PowerPC (R)

追加	*RT*、 *RA*、 *RB*
addc:	*RT*、 *RA*、 *RB*
アドコ	*RT*、 *RA*、 *RB*
addco:	*RT*、 *RA*、 *RB*

項目	説明
a	*RT*、 *RA*、 *RB*
a.	*RT*、 *RA*、 *RB*
ao	*RT*、 *RA*、 *RB*
アオ	*RT*、 *RA*、 *RB*

説明

addc 命令と **a** 命令は、汎用レジスター (GPR) *RA* と GPR *RB* の内容の合計をターゲット GPR *RT* に入れます。

addc 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

a 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	オーバーフロー例外 (OE)	固定小数点例外レジスター	レコード・ビット (RC)	条件レジスター・フィールド 0
追加	0	CA	0	なし
addc:	0	CA	1	LT、GT、EQ、SO
アドコ	1	SO、OV、CA	0	なし
addco:	1	SO、OV、CA	1	LT、GT、EQ、SO
a	0	CA	0	なし
a.	0	CA	1	LT、GT、EQ、SO
ao	1	SO、OV、CA	0	なし
アオ	1	SO、OV、CA	1	LT、GT、EQ、SO

addc 命令の 4 つの構文形式と **a** 命令の 4 つの構文形式は、常に固定小数点例外レジスターのカリー・ビット (CA) に影響を与えます。 構文形式がオーバーフロー例外 (OE) ビットを 1 に設定する場合、命令は固定小数点例外レジスターの要約オーバーフロー (SO) およびオーバーフロー (OV) ビットに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容を GPR 10 の内容に追加し、その結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 10 contains 0x8000 7000.
addc 6,4,10
# GPR 6 now contains 0x1000 A000.
```

2. 以下のコードは、GPR 4 の内容を GPR 10 の内容に追加し、結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0x7000 3000.
# Assume GPR 10 contains 0xFFFF FFFF.
addc. 6,4,10
# GPR 6 now contains 0x7000 2FFF.
```

3. 以下のコードは、GPR 4 の内容を GPR 10 の内容に追加し、その結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターにサマリー・オーバーフロー・ビット、オーバーフロー・ビット、およびキャリー・ビットを設定します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 10 contains 0x7B41 92C0.
addco 6,4,10
# GPR 6 now contains 0x0B41 C2C0.
```

4. 次のコードは、GPR 4 の内容を GPR 10 の内容に追加し、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターおよび条件レジスター・フィールド 0 にサマリー・オーバーフロー・ビット、オーバーフロー・ビット、およびキャリー・ビットを設定します。

```
# Assume GPR 4 contains 0x8000 0000.
# Assume GPR 10 contains 0x8000 7000.
addco. 6,4,10
# GPR 6 now contains 0x0000 7000.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

adde または ae (Add Extended) 命令

目的

2 つの汎用レジスターの内容を、固定小数点例外レジスターのキャリー・ビットの値に追加し、その結果を汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21	大江
2 2 - 30	138
31	rc

PowerPC (R)

ADDE *RT*、*RA*、*RB*

adde。 *RT*、*RA*、*RB*

アドデオ *RT*、*RA*、*RB*

アドデオ *RT*、*RA*、*RB*

POWER® ファミリー

AE *RT*、*RA*、*RB*

ae。 *RT*、*RA*、*RB*

エオ (aeo) *RT*、*RA*、*RB*

POWER® ファミリー

aeo: *RT*、 *RA*、 *RB*

説明

adde 命令と **ae** 命令は、汎用レジスター (GPR) *RA*、GPR *RB*、およびカリー・ビットの内容の合計をターゲット GPR *RT* に入れます。

adde 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

ae 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	オーバーフロー例外 (OE)	固定小数点例外レジスター	レコード ビット (RC)	条件レジスター・フィールド 0
ADDE	0	CA	0	なし
adde。	0	CA	1	LT、GT、EQ、SO
アドデオ	1	SO、OV、CA	0	なし
アドデオ	1	SO、OV、CA	1	LT、GT、EQ、SO
AE	0	CA	0	なし
ae。	0	CA	1	LT、GT、EQ、SO
エオ (aeo)	1	SO、OV、CA	0	なし
aeo:	1	SO、OV、CA	1	LT、GT、EQ、SO

adde 命令の 4 つの構文形式、および **ae** 命令の 4 つの構文形式は、常に固定小数点例外レジスターのカリー・ビット (CA) に影響します。 構文形式がオーバーフロー例外 (OE) ビットを 1 に設定する場合、命令は固定小数点例外レジスターの要約オーバーフロー (SO) およびオーバーフロー (OV) ビットに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容、GPR 10 の内容、および Fixed-Point Exception Register Carry ビットを追加し、結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x1000 0400.
# Assume GPR 10 contains 0x1000 0400.
# Assume the Carry bit is one.
adde 6,4,10
# GPR 6 now contains 0x2000 0801.
```

2. 以下のコードは、GPR 4 の内容、GPR 10 の内容、および固定小数点例外レジスター・キャリー・ビットを追加し、結果を GPR 6 に保管し、演算の結果を反映するために条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 10 contains 0x7B41 92C0.
# Assume the Carry bit is zero.
adde. 6,4,10
# GPR 6 now contains 0x0B41 C2C0.
```

3. 以下のコードは、GPR 4 の内容、GPR 10 の内容、および固定小数点例外レジスター・カリー・ビットを追加し、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターにサマリー・オーバーフロー・ビット、オーバーフロー・ビット、およびカリー・ビットを設定します。

```
# Assume GPR 4 contains 0x1000 0400.
# Assume GPR 10 contains 0xEFFF FFFF.
# Assume the Carry bit is one.
addeo 6,4,10
# GPR 6 now contains 0x0000 0400.
```

4. 以下のコードは、GPR 4 の内容、GPR 10 の内容、および固定小数点例外レジスター・ビットの内容を追加し、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターおよび条件レジスター・フィールド 0 に要約オーバーフロー・ビット、オーバーフロー・ビット、およびキャリー・ビットを設定します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 10 contains 0x8000 7000.
# Assume the Carry bit is zero.
addeo. 6,4,10
# GPR 6 now contains 0x1000 A000.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

アディ (即時追加) またはカル (計算アドレス下限) 命令

目的

オフセットおよび基底アドレスからのアドレスを計算し、その結果を汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	14
6 - 10	RT
11 - 15	RA
16 - 31	SI/D (I/D)

PowerPC (R)

アディ *RT*、 *RA*、 *SI*

POWER® ファミリー

cal *RT*、 *D*(*RA*)

詳しくは、「固定小数点演算命令の拡張ニーモニック」および「固定小数点ロード命令の拡張ニーモニック」を参照してください。

説明

代入 命令と **cal** 命令は、汎用レジスター (GPR) *RA* と、32 ビットに符号拡張された 16 ビット 2 の補数整数 *SI* または *D* の内容の合計を、ターゲット GPR *RT* に入れます。 GPR *RA* が GPR 0 の場合、*SI* または *D* はターゲット GPR *RT* に保管されます。

アディ 命令と **cal** 命令には 1 つの構文形式があり、条件レジスター・フィールド 0 または固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。
D	32 ビットに拡張された 16 ビット 2 の補数整数符号を指定します。
SI	演算のための 16 ビットの符号付き整数を指定します。

例

以下のコードは、GPR 5 の内容から 0xFFFF 8FF0 のオフセットを使用してアドレスまたは内容を計算し、結果を GPR 4 に保管します。

```
# Assume GPR 5 contains 0x0000 0900.
cal 4,0xFFFF8FF0(5)
# GPR 4 now contains 0xFFFF 98F0.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点アドレス計算命令
POWER® ファミリーの各種アドレス計算命令は、PowerPC® の演算命令に統合されています。

addic または ai (即値携帯の追加) 命令

目的

汎用レジスターおよび 16 ビットの符号付き整数の内容を追加し、その結果を汎用レジスターに入れ、固定小数点例外レジスターのカリー・ビットに影響を与えます。

構文

ビット	VALUE
0 - 5	12
6 - 10	RT
11 - 15	RA
16 - 31	SI

PowerPC (R)

addic (追加) *RT*、 *RA*、 *SI*

POWER® ファミリー

AI *RT*、 *RA*、 *SI*

詳しくは、 固定小数点演算命令の拡張ニーモニック を参照してください。

説明

addic 命令と **ai** 命令は、汎用レジスター (GPR) *RA* と 16 ビット符号付き整数 *SI,* の内容の合計を、ターゲット GPR *RT* に入れます。

即時データとして提供される 16 ビット整数は、加算演算を実行する前に 32 ビットに符号拡張されます。

addic 命令と **ai** 命令には 1 つの構文形式があり、固定小数点例外レジスターの Carry ビットを設定できます。これらの命令は条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。
SI	演算のための 16 ビットの符号付き整数を指定します。

例

以下のコードは、 0xFFFF FFFF を GPR 4 の内容に追加し、結果を GPR 6 に保管し、演算の結果を反映するように Carry ビットを設定します。

```
# Assume GPR 4 contains 0x0000 2346.
addic 6,4,0xFFFFFFFF
# GPR 6 now contains 0x0000 2345.
```

Addic。 または ai。 (即時保持および記録の追加) 指示

目的

汎用レジスターの内容と即時値を繰り上げ、加算を実行します。

構文

ビット	VALUE
0 - 5	13
6 - 10	RT
11 - 15	RA
16 - 31	SI

PowerPC (R)

addic。 *RT*、 *RA*、 *SI*

POWER® ファミリー

ai。 *RT*、 *RA*、 *SI*

詳しくは、 固定小数点演算命令の拡張ニーモニック を参照してください。

説明

addic 。 および **ai** 命令は、汎用レジスター (GPR) *RA* と 16 ビット符号付き整数 *SI,* の内容の合計を、ターゲット GPR *RT* に入れます。

即時データとして提供される 16 ビット整数 *SI* は、追加操作を実行する前に 32 ビットに符号拡張されます。

addic 。 および **ai** 命令には 1 つの構文形式があり、固定小数点例外レジスターのキャリー・ビットを設定できます。 これらの命令は、条件レジスター・フィールド 0 にも影響します。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。
SI	演算のための 16 ビットの符号付き整数を指定します。

例

以下のコードは、GPR 4 の内容に 16 ビットの符号付き整数を追加し、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスター・ビットおよび条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xEFFF FFFF.
addic. 6,4,0x1000
# GPR 6 now contains 0xF000 0FFF.
```

addis または cau (即時シフトの追加) 命令

目的

連結されたオフセットと基底アドレスからアドレスを計算し、その結果を汎用レジスターにロードします。

構文

ビット	VALUE
0 - 5	15
6 - 10	RT
11 - 15	RA
16 - 31	SI/ UI

表 *31. PowerPC*	
PowerPC	**PowerPC**
アドレス	*RT*、 *RA*、 *SI*
アドレス	*RT*、 *D*(*RA*)

表 *32. POWER* ファミリー	
POWER ファミリー	**POWER ファミリー**
cau	*RT*、 *RA*、 *UI*

詳しくは、「固定小数点演算命令の拡張ニーモニック」および「固定小数点ロード命令の拡張ニーモニック」を参照してください。

説明

addis および **cau** 命令は、汎用レジスター (GPR) *RA* の内容と、16 ビット符号なし整数、*SI* または *UI*、および x '0000' の連結の合計を、ターゲット GPR *RT* に入れます。 GPR *RA* が GPR 0 の場合、0、 *SI* または *UI*、および x '0000' の連結の合計がターゲット GPR *RT* に保管されます。

addis および **cau** 命令は、条件レジスター・フィールド 0 または固定小数点例外レジスターには影響しません。 **cau** 命令には、1 つの構文形式があります。 **addis** 命令には 2 つの構文形式がありますが、2 番目の形式が有効なのは、R_TOCU 再配置タイプが *D* 式で使用される場合のみです。 R_TOCU 再配置タイプは、**@u** 再配置指定子を使用して明示的に指定するか、TE ストレージ・マッピング・クラスを指定した **QualName** パラメーターを使用して暗黙的に指定することができます。

> **注: cau** 命令の即値は、16 ビットの符号なし整数ですが、 **addis** 命令の即値は、16 ビットの符号付き整数です。 この違いは、アーキテクチャーを 64 ビットに拡張した結果です。

アセンブラーは、 *UI* フィールドについては 0 から 65535 までの値範囲検査を行い、 *SI* フィールドについては -32768 から 32767 までの値範囲検査を行います。

addis および **cau** 命令のソース互換性を維持するために、アセンブラーは **addis** 命令の値範囲検査を -65536 から 65535 に拡張します。 符号ビットは無視され、アセンブラーは即時値が 16 ビットに収まることのみを保証します。 この拡張は、64 ビット実装での 32 ビット実装または 32 ビット・モードの動作には影響しません。

addis 命令のセマンティクスは、64 ビット・モードの場合とは異なります (32 ビット・モードの場合)。 ビット 32 が設定されている場合は、64 ビット汎用レジスターの上位 32 ビットに伝搬されます。 **addis** 命令を使用して符号なし整数を構成する場合は注意してください。 32 ビットの符号なし整数を指定した **addis** 命令は、64 ビット・モードに直接移植することはできません。 64 ビット・モードで符号なし整数を構成するために必要なコード・シーケンスは、32 ビット・モードで必要なコード・シーケンスとは大きく異なります。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のための最初のソース汎用レジスターを指定します。
UI	演算用の 16 ビット符号なし整数を指定します。
SI	指定する
演算用の 16 ビット符号付き整数。	演算用の 16 ビット符号付き整数。

例

以下のコードは、オフセット 0x0011 0000 を GPR 6 に含まれるアドレスまたは内容に追加し、結果を GPR 7 にロードします。

```
# Assume GPR 6 contains 0x0000 4000.
addis 7,6,0x0011
# GPR 7 now contains 0x0011 4000.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点アドレス計算命令
POWER® ファミリーの各種アドレス計算命令は、PowerPC® の演算命令に統合されています。

addme または ame (Add to Minus One Extended) 命令

目的

汎用レジスターの内容、固定小数点例外レジスターの中のカリー・ビット、および -1 を追加し、その結果を汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	///
21	大江
22 - 30	234
31	rc

PowerPC (R)

addme (addme)	*RT*、 *RA*
addme。	*RT*、 *RA*
addmeo (addmeo)	*RT*、 *RA*
addmeo。	*RT*、 *RA*

POWER® ファミリー

ame	*RT*、 *RA*
名前。	*RT*、 *RA*
アメオ	*RT*、 *RA*
ameo。	*RT*、 *RA*

説明

addme 命令と **ame** 命令は、汎用レジスター (GPR) *RA*、固定小数点例外レジスターのカリー・ビット、および -1 (0xFFFF FFFF)) の内容の合計をターゲット GPR *RT* に入れます。

addme 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

ame 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー例外 (OE)**	**固定小数点 例外レジスター**	**レコード・ビット (RC)**	**条件 フィールド 0 の登録**
addme (addme)	0	CA	0	なし
addme。	0	CA	1	LT、GT、EQ、SO
addmeo (addmeo)	1	SO、OV、CA	0	なし
addmeo:	1	SO、OV、CA	1	LT、GT、EQ、SO

項目	説明			
ame	0	CA	0	なし
名前。	0	CA	1	LT、GT、EQ、SO
アメオ	1	SO、OV、CA	0	なし
ameo:	1	SO、OV、CA	1	LT、GT、EQ、SO

addme 命令の 4 つの構文形式と **ame** 命令の 4 つの構文形式は、常に固定小数点例外レジスターのカリー・ビット (CA) に影響します。 構文形式がオーバーフロー例外 (OE) ビットを 1 に設定する場合、命令は固定小数点例外レジスターの要約オーバーフロー (SO) およびオーバーフロー (OV) ビットに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目 **説明**

RT 操作の結果が保管されるターゲット汎用レジスターを指定します。

RA 操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容、固定小数点例外レジスターのカリー・ビット、および -1 を追加し、その結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume the Carry bit is zero.
addme 6,4
# GPR 6 now contains 0x9000 2FFF.
```

2. 以下のコードは、GPR 4 の内容、固定小数点例外レジスターのカリー・ビット、および -1 を追加し、結果を GPR 6 に保管し、演算の結果を反映するために条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB000 42FF.
# Assume the Carry bit is zero.
addme. 6,4
# GPR 6 now contains 0xB000 42FE.
```

3. 以下のコードは、GPR 4 の内容、固定小数点例外レジスターのカリー・ビット、および -1 を追加し、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターのサマリー・オーバーフロー・ビット、オーバーフロー・ビット、およびカリー・ビットを設定します。

```
# Assume GPR 4 contains 0x8000 0000.
# Assume the Carry bit is zero.
addmeo 6,4
# GPR 6 now contains 0x7FFF FFFF.
```

4. 以下のコードは、GPR 4 の内容、固定小数点例外レジスターのカリー・ビット、および -1 を追加し、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターおよび条件レジスター 0 に要約オーバーフロー・ビット、オーバーフロー・ビット、およびカリー・ビットを設定します。

```
# Assume GPR 4 contains 0x8000 0000.
# Assume the Carry bit is one.
addmeo. 6,4
# GPR 6 now contains 0x8000 000.
```

addze または aze (Add to Zero Extended) 命令

目的

汎用レジスターの内容、ゼロ、および FIxed-Point Exception Register の中の Carry ビットの値を追加し、結果を汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	///
21	大江
22 - 30	202
31	rc

PowerPC (R)

addze (追加) *RT*、 *RA*

addze: *RT*、 *RA*

addzeo (addzeo) *RT*、 *RA*

addzeo: *RT*、 *RA*

POWER® ファミリー

aze (アズ) *RT*、 *RA*

"aze" *RT*、 *RA*

アゼオ *RT*、 *RA*

azeo。 *RT*、 *RA*

説明

addze 命令および **aze** 命令は、汎用レジスター (GPR) *RA*、カリー・ビット、および 0x0000 0000 の内容を追加し、結果をターゲット GPR *RT* に入れます。

addze 命令には 4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

aze 命令には 4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
addze (追加)	0	CA	0	なし
addze:	0	CA	1	LT、GT、EQ、SO

項目	説明			
addzeo (addzeo)	1	SO、OV、CA	0	なし
addzeo:	1	SO、OV、CA	1	LT、GT、EQ、SO
aze (アズ)	0	CA	0	なし
"aze"	0	CA	1	LT、GT、EQ、SO
アゼオ	1	SO、OV、CA	0	なし
azeo。	1	SO、OV、CA	1	LT、GT、EQ、SO

addze 命令の 4 つの構文形式と **aze** 命令の 4 つの構文形式は、常に固定小数点例外レジスターの Carry ビット (CA) に影響を与えます。 構文形式がオーバーフロー例外 (OE) ビットを 1 に設定する場合、命令は固定小数点例外レジスターの要約オーバーフロー (SO) およびオーバーフロー (OV) ビットに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4、0、および Carry ビットの内容を追加し、その結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x7B41 92C0.
# Assume the Carry bit is zero.
addze 6,4
# GPR 6 now contains 0x7B41 92C0.
```

2. 以下のコードは、GPR 4、0、および Carry ビットの内容を追加し、結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xEFFF FFFF.
# Assume the Carry bit is one.
addze. 6,4
# GPR 6 now contains 0xF000 0000.
```

3. 以下のコードは、GPR 4、0、および Carry ビットの内容を追加し、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターにサマリー・オーバーフロー・ビット、オーバーフロー・ビット、およびキャリー・ビットを設定します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume the Carry bit is one.
addzeo 6,4
# GPR 6 now contains 0x9000 3001.
```

4. 以下のコードは、GPR 4、0、および Carry ビットの内容を追加し、結果を GPR 6 に保管し、演算の結果を反映するために、要約オーバーフロー、オーバーフロー、および Carry ビットを固定小数点例外レジスターおよび条件レジスター・フィールド 0 に設定します。

```
# Assume GPR 4 contains 0xEFFF FFFF.
# Assume the Carry bit is zero.
adzeo. 6,4
# GPR 6 now contains 0xEFFF FFFF.
```

AND (AND) 命令

目的

2 つの汎用レジスターの内容を論理的に AND 演算し、その結果を汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	28
31	rc

項目	説明
AND	*RA*、*RS*、*RB*
and。	*RA*、*RS*、*RB*

説明

命令および 命令は、汎用レジスター (GPR) *RS* の内容を GPR *RB* の内容と論理的に AND し、結果をターゲット GPR *RA* に入れます。

命令と 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
AND	なし	なし	0	なし
and。	なし	なし	1	LT、GT、EQ、SO

命令と 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容と GPR 7 の内容を論理的に AND 演算し、その結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0xFFF2 5730.
# Assume GPR 7 contains 0x7B41 92C0.
and 6,4,7
# GPR 6 now contains 0x7B40 1200.
```

2. 以下のコードは、GPR 4 の内容と GPR 7 の内容を論理的に AND 演算し、結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xFFF2 5730.
# Assume GPR 7 contains 0xFFFF EFFF.
and. 6,4,7
# GPR 6 now contains 0xFFF2 4730.
```

andc (AND with Complement) 命令

目的

論理的には、汎用レジスターの内容と汎用レジスターの内容を補完するものとして AND 演算します。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	60
31	rc

項目	説明
および	*RA*、*RS*、*RB*
andc:	*RA*、*RS*、*RB*

説明

andc 命令は、汎用レジスター (GPR) *RS* の内容と GPR *RB* の内容とを論理的に AND し、その結果を GPR *RA* に入れます。

andc 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
および	なし	なし	0	なし
andc:	なし	なし	1	LT、GT、EQ、SO

andc 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容と GPR 5 の内容の補数を論理的に AND し、その結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 5 contains 0xFFFF FFFF.
# The complement of 0xFFFF FFFF becomes 0x0000 0000.
andc 6,4,5
# GPR 6 now contains 0x0000 0000.
```

2. 以下のコードは、GPR 4 の内容と GPR 5 の内容を論理的に AND 演算し、その結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 5 contains 0x7676 7676.
# The complement of 0x7676 7676 is 0x8989 8989.
andc. 6,4,5
# GPR 6 now contains 0x8000 0000.
```

アンディー または、andil。 (AND Immediate) 命令

目的

即時値を使用して汎用レジスターの内容を論理的に AND します。

構文

ビット	VALUE
0 - 5	28
6 - 10	RS
11 - 15	RA
16 - 31	UI

PowerPC (R)

アンディー *RA*、 *RS*、 *UI*

POWER® ファミリー

andil。 *RA*、 *RS*、 *UI*

説明

andi 。 および **andil** 命令は、汎用レジスター (GPR) *RS* の内容を、x '0000' と 16 ビット符号なし整数 *UI* を連結して論理的に AND 演算し、その結果を GPR *RA* に入れます。

andi 。 および **andil** 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。
andi 。 および **andil** 命令は、要約オーバーフロー (SO) ビットを固定小数点例外レジスターから条件レジス

ター・フィールド 0 にコピーし、条件レジスター・フィールド 0 の「より小 (LT)」ビット、「より大 (GT)」ビット、または「等しい (EQ)」ビットのいずれかを設定します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
UI	演算用の 16 ビット符号なし整数を指定します。

例

以下のコードは、GPR 4 の内容を 0x0000 5730 と論理的に AND し、結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0x7B41 92C0.
andi. 6,4,0x5730
# GPR 6 now contains 0x0000 1200.
# CRF 0 now contains 0x4.
```

andis. または andiu。 (AND 即時シフト) 命令

目的

論理的には、16 ビットの符号なし整数を持つ汎用レジスターの内容の最上位 16 ビットを AND 演算し、その結果を汎用レジスターに保管します。

構文

ビット	VALUE
0 - 5	29
6 - 10	RS
11 - 15	RA
16 - 31	UI

PowerPC (R)

andis. *RA*、 *RS*、 *UI*

POWER® ファミリー

アンディウ *RA*、 *RS*、 *UI*

説明

マンディス および **andiu** 命令は、汎用レジスター (GPR) *RS* の内容と、16 ビット符号なし整数 *UI* および x '0000' を連結したものを論理的に AND 演算し、その結果をターゲット GPR *RA* に入れます。

マンディス および **andiu** 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。**マンディス** および **andiu** 命令は、条件レジスター・フィールド 0 に「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、または「要約オーバーフロー (SO)」ビットを設定します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
UI	演算用の 16 ビット符号なし整数を指定します。

例

以下のコードは、GPR 4 の内容を 0x5730 0000 と論理的に AND 演算し、結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0x7B41 92C0.
andis. 6,4,0x5730
# GPR 6 now contains 0x5300 0000.
```

b (分岐) 命令

目的

指定されたターゲット・アドレスに分岐します。

構文

ビット	VALUE
0 - 5	18
6 - 29	LL
30	AA
31	LK

項目	説明
B	ターゲット・アドレス
BA	ターゲット・アドレス
BL	ターゲット・アドレス
ブラ	ターゲット・アドレス

説明

b 命令は、ブランチ・ターゲット・アドレスによって指定された命令にブランチします。 分岐ターゲット・アドレスは、2 つの方法のいずれかで計算されます。

b 命令を使用する場合は、以下の点を考慮してください。

- 絶対アドレス・ビット (AA) が 0 の場合、ブランチ・ターゲット・アドレスは 24 ビットの *LI* フィールドを連結して計算されます。 このフィールドは、ターゲット・アドレスから命令のアドレスを減算し、その結果を 4 および b'00' で除算することによって計算されます。 結果は 32 ビットに符号拡張され、このブランチ命令のアドレスに追加されます。
- AA ビットが 1 の場合、分岐ターゲット・アドレスは、32 ビットに拡張された b'00' 符号で連結された *LI* フィールドです。 *LI* フィールドは、ターゲット・アドレスの下位 26 ビットを 4 で除算した値です。

b 命令には、4 つの構文形式があります。 各構文形式は、リンク・ビットおよびリンク・レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	**絶対 アドレス・ビット (AA)**	**固定小数点 例外レジスター**	**リンク・ビット (LK)**	**条件 フィールド 0 の登録**
B	0	なし	0	なし
BA	1	なし	0	なし
BL	0	なし	1	なし
ブラ	1	なし	1	なし

b 命令の 4 つの構文形式は、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。 構文形式は、AA ビットとリンク・ビット (LK) を設定し、分岐ターゲット・アドレスの計算方法を決定します。 リンク・ビット (LK) が 1 に設定されている場合は、命令の有効アドレスがリンク・レジスターに入れられます。

パラメーター

項目	説明
ターゲット・アドレス	ターゲット・アドレスを指定します。

例

1. 以下のコードは、プログラムの実行を there に転送します。

```
here: b there
      cror 31,31,31
# The execution of the program continues at there.
there:
```

2. 以下のコードは、プログラムの実行を here に転送し、リンク・レジスターを設定します。

```
        bl here
return: cror 31,31,31
# The Link Register now contains the address of return.
# The execution of the program continues at here.
here:
```

関連概念

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

b (分岐) 命令

bc (分岐条件付き) 命令

目的

指定されたターゲット・アドレスに条件付きで分岐します。

構文

ビット	値
0 - 5	16
6 - 10	BO
11 - 15	BI
16 - 29	BD
30	AA

ビット	値
31	LK

項目	説明
bc	BO、 BI、 target_address
bca (bca)	BO、 BI、 target_address
bcl (bcl)	BO、 BI、 target_address
bcla (bcla)	BO、 BI、 target_address

説明

bc 命令は、分岐ターゲット・アドレスによって指定された命令に分岐します。 ブランチ・ターゲット・アドレスは、次の 2 つの方法のいずれかで計算されます。

- 絶対アドレス・ビット (AA) が 0 の場合、分岐ターゲット・アドレスは、14 ビット分岐変位 (BD) と b '00' を連結し、これを 32 ビットに符号拡張し、結果をこの分岐命令のアドレスに加算することによって計算されます。
- AA が 1 の場合、分岐ターゲット・アドレスは、32 ビットに拡張された b '00' 符号付きで連結された BD です。

bc 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	**絶対アドレス・ビット (AA)**	**固定小数点例外レジスター**	**リンク・ビット (LK)**	**条件レジスター・フィールド 0**
bc	0	なし	0	なし
bca (bca)	1	なし	0	なし
bcl (bcl)	0	なし	1	なし
bcla (bcla)	1	なし	1	なし

bc 命令の 4 つの構文形式は、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。 構文形式は、AA ビットとリンク・ビット (LK) を設定し、分岐ターゲット・アドレスの計算方法を決定します。 リンク・ビット (LK) が 1 に設定されている場合は、命令の有効アドレスがリンク・レジスターに入れられます。

分岐オプション・フィールド (BO) は、さまざまなタイプの分岐を単一の命令に結合するために使用されます。 分岐オプション・フィールドを自動的に設定するために、拡張ニーモニックが用意されています。

BO フィールドのエンコードは、PowerPC ® アーキテクチャーで定義されます。 以下のリストは、 pre-V2.00 エンコードを使用する場合にこのフィールドで使用可能な値の要旨を示しています。

表 *33. pre-V2.00* エンコードを使用した *BO* フィールド値

BO	説明
0000y	CTR を減らします。その後、減分された CTR が 0 ではなく、条件が False の場合に分岐します。
0001y	CTR を減らします。その後、減分された CTR が 0 で条件が False の場合に分岐します。
001zy	条件が False の場合に分岐します。
0100y	CTR を減らします。次に、減分された CTR のビットが 0 ではなく、条件が True の場合に分岐します。
0101y	CTR を減らします。その後、減分された CTR が 0 で、条件が True の場合に分岐します。

表 *33. pre-V2.00* エンコードを使用した *BO* フィールド値 *(*続き*)*

BO	説明
011zy	条件が True の場合に分岐します。
1z00y	CTR を減らします。その後、減分された CTR が 0 でない場合は分岐します。
1z01y	CTR を減らします。その後、減分された CTR が 0 の場合は分岐します。
1z1zz	常に分岐します。

PowerPC ® アーキテクチャーでは、ビットは以下のとおりです。

- z ビットは、0 でなければならないビットを示します。 ビットが 0 でない場合、命令形式は無効です。
- y ビットは、条件付き分岐が行われる可能性が高いかどうかについてのヒントを提供します。 このビットの値は 0 または 1 のいずれかです。 デフォルト値は 0 です。

POWER ® ファミリー・アーキテクチャーでは、z ビットと y ビットは 0 または 1 のいずれかです。

V2.00 エンコードを使用した BO フィールドのエンコードについて、以下に簡単に説明します。

表 *34. V2.00* エンコードを使用した *BO* フィールド値

BO	説明
0000z	CTR を減らします。その後、減分された CTR が 0 ではなく、条件が False の場合に分岐します。
0001z	CTR を減らします。その後、減分された CTR が 0 で条件が False の場合に分岐します。
001at	条件が False の場合に分岐します。
0100z	CTR を減らします。次に、減分された CTR のビットが 0 ではなく、条件が True の場合に分岐します。
0101z	CTR を減らします。その後、減分された CTR が 0 で、条件が True の場合に分岐します。
011at	条件が True の場合に分岐します。
1a00t	CTR を減らします。その後、減分された CTR が 0 でない場合は分岐します。
1a01t	CTR を減らします。その後、減分された CTR が 0 の場合は分岐します。
1z1zz	常に分岐します。

ソフトウェアは、以下に示すように、BO フィールドの a ビットと t ビットを使用して、分岐が行われる可能性があるかどうかを示すヒントを提供することができます。

項目	説明
at	Hint
00	ヒントは提供されません。
01	予約済み
10	分岐が行われない可能性があります。
11	分岐が行われる可能性があります。

パラメーター

項目	説明
ターゲット・アドレス	ターゲット・アドレスを指定します。 **bca** や **bcla** などの絶対ブランチの場合、ターゲット・アドレスは、16 ビットで包含可能な即時データにすることができます。
BI	条件比較のための条件レジスターのビットを指定します。

項目	説明
BO	命令で使用される分岐オプション・フィールドを指定します。

例

以下のコードは、カウント・レジスターの値に応じてターゲット・アドレスに分岐します。

```
addi 8,0,3
# Loads GPR 8 with 0x3.
mtctr 8
# The Count Register (CTR) equals 0x3.
addic. 9,8,0x1
# Adds one to GPR 8 and places the result in GPR 9.
# The Condition Register records a comparison against zero
# with the result.
bc 0xC,0,there
# Branch is taken if condition is true. 0 indicates that
# the 0 bit in the Condition Register is checked to
# determine if it is set (the LT bit is on). If it is set,
# the branch is taken.
bcl 0x8,2,there
# CTR is decremented by one, becomming 2.
# The branch is taken if CTR is not equal to 0 and CTR bit 2
# is set (the EQ bit is on).
# The Link Register contains address of next instruction.
```

関連概念

アセンブラーの概要
アセンブラー・プログラムは、マシン言語命令を受け取り、それらをマシン・オブジェクト・コードに変換します。

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

b (分岐) 命令

bcctr または bcc (カウント・レジスターへの分岐条件付き) 命令

目的

カウント・レジスター内に含まれるアドレスに条件付きで分岐します。

構文

ビット	値
0 - 5	19
6 - 10	BO
11 - 15	BI
16 - 18	///
19 - 20	BH
21 - 30	528
31	LK

PowerPC (R)

bcctr	BO、 BI、 BH
bcctrl	BO、 BI、 BH

POWER® ファミリー

BCC BO、 BI、 BH

bccl BO、 BI、 BH

説明

bcctr および **bcc** 命令は、カウント・レジスターに含まれているブランチ・ターゲット・アドレスによって指定された命令に条件付きで分岐します。 ブランチ・ターゲット・アドレスは、カウント・レジスター・ビット 0 から 29 と b '00' を連結したものです。

bcctr および **bcc** 命令には、2 つの構文形式があります。 各構文形式は、リンク・ビットおよびリンク・レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	**絶対アドレス・ビット (AA)**	**固定小数点例外レジスター**	**リンク・ビット (LK)**	**条件レジスター・フィールド 0**
bcctr	なし	なし	0	なし
bcctrl	なし	なし	1	なし
BCC	なし	なし	0	なし
bccl	なし	なし	1	なし

bcctr および **bcc** 命令の 2 つの構文形式は、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。 リンク・ビットが 1 の場合、ブランチ命令に続く命令の有効アドレスがリンク・レジスターに入れられます。

分岐オプション・フィールド (BO) は、さまざまなタイプの分岐を単一の命令に結合するために使用されます。 分岐オプション・フィールドを自動的に設定するために、拡張ニーモニックが用意されています。

BO フィールドのエンコードは、PowerPC® アーキテクチャーで定義されます。 以下のリストは、 pre-V2.00 エンコードを使用する場合にこのフィールドで使用可能な値の要旨を示しています。

BO **説明**

`0000y` CTR を減らします。その後、減分された CTR が 0 ではなく、条件が False の場合に分岐します。

`0001y` CTR を減らします。その後、減分された CTR が 0 で条件が False の場合に分岐します。

`001zy` 条件が False の場合に分岐します。

`0100y` CTR を減らします。次に、減分された CTR のビットが 0 ではなく、条件が True の場合に分岐します。

`0101y` CTR を減らします。その後、減分された CTR が 0 で、条件が True の場合に分岐します。

`011zy` 条件が True の場合に分岐します。

`1z00y` CTR を減らします。その後、減分された CTR が 0 でない場合は分岐します。

`1z01y` CTR を減らします。その後、減分された CTR が 0 の場合は分岐します。

`1z1zz` 常に分岐します。

PowerPC® アーキテクチャーでは、ビットは以下のとおりです。

- z ビットは、0 でなければならないビットを示します。 ビットが 0 でない場合、命令形式は無効です。
- y ビットは、条件付き分岐が行われる可能性が高いかどうかについてのヒントを提供します。 このビットの値は 0 または 1 のいずれかです。 デフォルト値は 0 です。

POWER® ファミリー・アーキテクチャーでは、 z ビットと y ビットは 0 または 1 のいずれかです。

V2.00 エンコードを使用した BO フィールドのエンコードについて、以下に簡単に説明します。

表 *35. V2.00* エンコードを使用した *BO* フィールド値

BO	説明
0000z	CTR を減らします。その後、減分された CTR が 0 ではなく、条件が False の場合に分岐します。
0001z	CTR を減らします。その後、減分された CTR が 0 で条件が False の場合に分岐します。
001at	条件が False の場合に分岐します。
0100z	CTR を減らします。次に、減分された CTR のビットが 0 ではなく、条件が True の場合に分岐します。
0101z	CTR を減らします。その後、減分された CTR が 0 で、条件が True の場合に分岐します。
011at	条件が True の場合に分岐します。
1a00t	CTR を減らします。その後、減分された CTR が 0 でない場合は分岐します。
1a01t	CTR を減らします。その後、減分された CTR が 0 の場合は分岐します。
1z1zz	常に分岐します。

ソフトウェアは、以下に示すように、BO フィールドの a ビットと t ビットを使用して、分岐が行われる可能性があるかどうかを示すヒントを提供することができます。

at	HINT
00	ヒントは提供されません。
01	予約済み
10	分岐が行われない可能性が高い。
11	その枝は取られる可能性が高い。

詳しくは、 IBM Power ISA PDF を参照してください。

分岐ヒント・フィールド (BH) は、以下に示すように、命令の使用に関するヒントを提供するために使用されます。

BH	HINT
00	命令はサブルーチン戻りではありません。ターゲット・アドレスは、分岐が行われた前に使用されたターゲット・アドレスと同じである可能性があります。
01	予約済み
10	予約済み
11	ターゲット・アドレスは予測できません。

パラメーター

項目	説明
BO	分岐オプション・フィールドを指定します。
BI	条件比較のための条件レジスターのビットを指定します。
BIF (BIF)	条件比較に使用する条件レジスター・ビット (LT、GT、EQ、または SO) を指定する条件レジスター・フィールドを指定します。
BH	命令の使用に関するヒントを提供します。

例

以下のコードは、条件レジスターのビットに依存する特定のアドレスから、カウント・レジスターに含まれるアドレスに分岐し、分岐ヒントは提供されません。

```
bcctr 0x4,0,0
cror 31,31,31
# Branch occurs if LT bit in the Condition Register is 0.
# The branch will be to the address contained in
# the Count Register.
bcctrl 0xC,1,0
return: cror 31,31,31
# Branch occurs if GT bit in the Condition Register is 1.
# The branch will be to the address contained in
# the Count Register.
# The Link register now contains the address of return.
```

関連概念

アセンブラーの概要
アセンブラー・プログラムは、マシン言語命令を受け取り、それらをマシン・オブジェクト・コードに変換します。

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

b (分岐) 命令

bclr または bcr (分岐条件付きリンク・レジスター) 命令

目的

条件付きで、リンク・レジスターに含まれているアドレスに分岐します。

構文

ビット	値
0 - 5	19
6 - 10	BO
11 - 15	BI
16 - 18	///
19 - 20	BH
21 - 30	16
31	LK

PowerPC (R)

BCLR BO、 BI、 BH

bclrl BO、 BI、 BH

POWER® ファミリー

bcr (bcr) BO、 BI、 BH

BCRL BO、 BI、 BH

説明

bclr および **bcr** 命令は、分岐ターゲット・アドレスによって指定された命令に分岐します。 分岐ターゲット・アドレスは、リンク・レジスターのビット 0 から 29 と b '00' を連結したものです。

bclr 命令と **bcr** 命令には、2 つの構文形式があります。 各構文形式は、リンク・ビットおよびリンク・レジスターに対して異なる影響を与えます。

シンタックス・フォーム	絶対アドレス・ビット (AA)	固定小数点例外レジスター	リンク・ビット (LK)	条件レジスター・フィールド 0
BCLR	なし	なし	0	なし
bclrl	なし	なし	1	なし
bcr (bcr)	なし	なし	0	なし
BCRL	なし	なし	1	なし

bclr および **bcr** 命令の 2 つの構文形式は、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。 リンク・ビット (LK) が 1 の場合、ブランチ命令に続く命令の有効アドレスがリンク・レジスターに入れられます。

分岐オプション・フィールド (BO) は、さまざまなタイプの分岐を単一の命令に結合するために使用されます。 分岐オプション・フィールドを自動的に設定するために、拡張ニーモニックが用意されています。

BO フィールドのエンコードは、PowerPC ® アーキテクチャーで定義されます。 以下のリストで、このフィールドに指定できる値について簡単に説明します。

BO **説明**

`0000y` CTR を減らします。その後、減分された CTR が 0 ではなく、条件が False の場合に分岐します。

`0001y` CTR を減らします。その後、減分された CTR が 0 で条件が False の場合に分岐します。

`001zy` 条件が False の場合に分岐します。

`0100y` CTR を減らします。次に、減分された CTR のビットが 0 ではなく、条件が True の場合に分岐します。

`0101y` CTR を減らします。その後、減分された CTR が 0 で、条件が True の場合に分岐します。

`011zy` 条件が True の場合に分岐します。

`1z00y` CTR を減らします。その後、減分された CTR が 0 でない場合は分岐します。

`1z01y` CTR を減らします。その後、減分された CTR が 0 の場合は分岐します。

`1z1zz` 常に分岐します。

PowerPC ® アーキテクチャーでは、ビットは以下のとおりです。

- z ビットは、0 でなければならないビットを示します。 ビットが 0 でない場合、命令形式は無効です。
- y ビットは、条件付き分岐が行われる可能性が高いかどうかについてのヒントを提供します。 このビットの値は 0 または 1 のいずれかです。 デフォルト値は 0 です。

POWER ® ファミリー・アーキテクチャーでは、 z ビットと y ビットは 0 または 1 のいずれかです。

V2.00 エンコードを使用した BO フィールドのエンコードについて、以下に簡単に説明します。

表 *36. V2.00* エンコードを使用した *BO* フィールド値

BO **説明**

`0000z` CTR を減らします。その後、減分された CTR が 0 ではなく、条件が False の場合に分岐します。

`0001z` CTR を減らします。その後、減分された CTR が 0 で条件が False の場合に分岐します。

`001at` 条件が False の場合に分岐します。

`0100z` CTR を減らします。次に、減分された CTR のビットが 0 ではなく、条件が True の場合に分岐します。

`0101z` CTR を減らします。その後、減分された CTR が 0 で、条件が True の場合に分岐します。

表 *36. V2.00* エンコードを使用した *BO* フィールド値 *(続き)*

BO	説明
011at	条件が True の場合に分岐します。
1a00t	CTR を減らします。その後、減分された CTR が 0 でない場合は分岐します。
1a01t	CTR を減らします。その後、減分された CTR が 0 の場合は分岐します。
1z1zz	常に分岐します。

ソフトウェアは、以下に示すように、BO フィールドの a ビットと t ビットを使用して、分岐が行われる可能性があるかどうかを示すヒントを提供することができます。

at	HINT
00	ヒントは提供されません。
01	予約済み
01	分岐が行われない可能性が高い。
11	その枝は取られる可能性が高い。

分岐ヒント・フィールド (BH) は、以下に示すように、命令の使用に関するヒントを提供するために使用されます。

BH	HINT
00	命令はサブルーチン戻りではありません。ターゲット・アドレスは、分岐が行われた前に使用されたターゲット・アドレスと同じである可能性があります。
01	予約済み
10	予約済み
11	ターゲット・アドレスは予測できません。

パラメーター

項目	説明
BO	分岐オプション・フィールドを指定します。
BI	条件比較のための条件レジスターのビットを指定します。
BH	命令の使用に関するヒントを提供します。

例

以下のコードは、条件レジスターのビット 0 に依存する計算されたブランチ・ターゲット・アドレスにブランチし、ブランチ・ヒントは提供されません。

```
bclr 0x0,0,0
# The Count Register is decremented.
# A branch occurs if the LT bit is set to zero in the
# Condition Register and if the Count Register
# does not equal zero.
# If the conditions are met, the instruction branches to
# the concatenation of bits 0-29 of the Link Register and b'00'.
```

clcs (キャッシュ・ライン・コンピュート・サイズ) 命令

目的

指定されたキャッシュ・ライン・サイズを汎用レジスターに入れます。

注: clcs 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0-5	31
6 ～ 10	RT
11-15	RA
16-20	///
21-30	531
31	rc

POWER® ファミリー

clcs (clcs) *RT*、 *RA*

説明

clcs 命令は、 *RA* で指定されたキャッシュ・ライン・サイズをターゲット汎用レジスター (GPR) *RT* に入れます。 *RA* の値によって、GPR *RT* で戻されるキャッシュ・ライン・サイズが決まります。

項目	説明
RA の値	**RT で戻されたキャッシュ行サイズ**
00xxx	未定義
010xx	未定義
01100	命令キャッシュの行サイズ
01101	Data Cache 行サイズ
01110	最小キャッシュ行サイズ
01111	最大キャッシュ行サイズ
1xxxx	未定義

注: GPR *RT* の値は 64 以上 4096 以下でなければなりません。そうでない場合、結果は未定義になります。

clcs 命令の構文形式は 1 つだけで、固定小数点例外レジスターには影響しません。 レコード (Rc) ビットが 1 に設定されている場合、条件レジスター・フィールド 0 は未定義です。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	要求されたキャッシュ回線サイズを指定します。

例

以下のコードは、最大キャッシュ・ライン・サイズを GPR 4 にロードします。

```
# Assume that 0xf is the cache
```

```
line size requested
.
    clcs 4,0xf
# GPR 4 now contains the maximum Cache Line size.
```

clf (キャッシュ・ライン・フラッシュ) 命令

目的

変更されたデータの行をデータ・キャッシュからメイン・メモリーに書き込むか、キャッシュされた命令または変更されていないデータを無効にします。

注: clf 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0-5	31
6 ～ 10	///
11-15	RA
16-20	RB
21-30	118
31	rc

POWER® ファミリー

clf *RA*、 *RB*

説明

clf 命令は、汎用レジスター (GPR) *RA* の内容を GPR *RB* の内容に追加して、有効アドレス (EA) を計算します。 *RA* フィールドが 0 の場合、EA は *RB* と 0 の内容の合計です。 *RA* フィールドが 0 でなく、命令によってデータ・ストレージ割り込みが発生しない場合、操作の結果は GPR *RA* に戻されます。

clf 命令を使用する場合は、以下の点を考慮してください。

- マシン状態レジスター (MSR) のデータ再配置 (DR) ビットが 0 に設定されている場合、有効アドレスは実アドレスとして扱われます。
- MSR DR ビットが 1 に設定されている場合、有効アドレスは仮想アドレスとして扱われます。 この場合、MSR 命令再配置ビット (IR) は無視されます。
- EA によってアドレス指定されたバイトを含む行がデータ・キャッシュ内にあり、変更されている場合は、メイン・メモリーへの行の書き込みが開始されます。 EA によってアドレス指定されたバイトを含む行がキャッシュの 1 つにある場合、その行は無効です。
- MSR (DR) = 1 の場合、仮想アドレスに変換がないと、データ・ストレージ割り込みが発生し、データ・ストレージ割り込みセグメント・レジスターの最初のビットが 1 に設定されます。
- マシン・チェック割り込みは、仮想アドレスが無効な実アドレスに変換され、回線がデータ・キャッシュに存在する場合に発生します。
- アドレス変換は、保護とデータ・ロックを無視して、アドレスされたバイトへのロードとして命令を扱います。 この命令によって変換ルック・アサイド・バッファー (TLB) ミスが発生した場合、参照ビットが設定されます。
- EA が入出力アドレスを指定すると、命令はノーオペレーションとして扱われますが、EA は GPR *RA* に置かれます。

clf 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。 レコード (Rc) ビットが 1 に設定されている場合、条件レジスター・フィールド 0 は未定義です。

パラメーター

項目	説明
RA	EA 計算用のソース汎用レジスターを指定します。 *RA* が GPR 0 でない場合は、操作用のターゲット汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

プロセッサーは、命令ストレージとデータ・ストレージの整合性を保つ必要はありません。 以下のコードは、変更された命令を実行する前に、ストレージ同期命令を実行します。

```
# Assume that instruction A is assigned to storage location
# ox0033 0020.
# Assume that the storage location to which A is assigned
# contains 0x0000 0000.
# Assume that GPR 3 contains 0x0000 0020.
# Assume that GPR 4 contains 0x0033 0020.
# Assume that GPR 5 contains 0x5000 0020.
st      R5,R4,R3         # Store branch instruction in memory
clf     R4,R3            # Flush A from cache to main memory
dcs                      # Ensure clf is complete
ics                      # Discard prefetched instructions
b       0x0033 0020      # Go execute the new instructions
```

ストアの後、 **clf**、 **dcs**、および **ics** 命令の実行前に、キャッシュ内の A のコピーにはブランチ命令が含まれます。 ただし、メイン・メモリー内の A のコピーにはまだ 0 が含まれている可能性があります。 **clf** 命令は、新しい命令をメイン・メモリーにコピーして戻し、命令キャッシュとデータ・キャッシュの両方で位置 A を含むキャッシュ・ラインを無効にします。 **dcs** 命令とそれに続く **ics** 命令のシーケンスにより、新しい命令がメイン・メモリー内にあること、およびデータ・キャッシュと命令キャッシュ内の場所のコピーが次の命令をフェッチする前に無効になることが保証されます。

cli (キャッシュ行無効化) 命令

目的

データ・キャッシュまたは命令キャッシュのいずれかでアドレス指定されたバイトを含む行を無効にします。これにより、後続の参照では、メイン・メモリーからその行が再度取得されます。

注: cli 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0-5	31
6 ～ 10	///
11-15	RA
16-20	RB
21-30	502
31	rc

POWER® ファミリー

cli *RA*、 *RB*

説明

cli 命令は、データ・キャッシュまたは命令キャッシュのいずれかでアドレス指定されたバイトを含む行を無効にします。 *RA* が 0 でない場合、**cli** 命令は、汎用レジスター (GPR) *RA* の内容を GPR *RB* の内容に追加することによって、実効アドレス (EA) を計算します。 *RA* が GPR 0 でない場合、または命令によってデータ・ストレージ割り込みが発生しない場合、計算の結果は GPR *RA* に戻されます。

cli 命令を使用する際には、以下のことを考慮してください。

- マシン状態レジスター (MSR) のデータ再配置 (DR) ビットが 0 の場合、有効アドレスは実アドレスとして扱われます。
- MSR DR ビットが 1 の場合、有効アドレスは仮想アドレスとして扱われます。 この場合、MSR 再配置 (IR) ビットは無視されます。
- EA によってアドレス指定されたバイトを含む行がデータ・キャッシュまたは命令キャッシュ内にある場合、その行は使用不可になるため、その行への次の参照はメイン・メモリーから取得されます。
- MSR (DR) = 1 の場合、仮想アドレスに変換がないと、データ・ストレージ割り込みが発生し、データ・ストレージ割り込みセグメント・レジスターの最初のビットが 1 に設定されます。
- アドレス変換は、保護とデータ・ロックを無視して、 **cli** 命令をバイト・アドレスへの保管として扱います。 この命令によって変換ルック・アサイド・バッファー (TLB) ミスが発生した場合、参照ビットが設定されます。
- EA が入出力アドレスを指定すると、命令はノーオペレーションとして扱われますが、EA は引き続き *RA* に置かれます。

cli 命令の構文形式は 1 つだけで、固定小数点例外レジスターには影響しません。 レコード (Rc) ビットが 1 に設定されている場合、条件レジスター・フィールド 0 は未定義です。

パラメーター

項目	説明
RA	EA 計算用のソース汎用レジスター、および操作用のターゲット汎用レジスター (*RA* が GPR 0 でない場合) を指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

セキュリティー

cli 命令には特権があります。

cmp (比較) 命令

目的

2 つの汎用レジスターの内容を代数的に比較します。

構文

ビット	VALUE
0-5	31
6-8	BF
9	/
10	L

ビット	VALUE
11-15	RA
16-20	RB
21-30	0
31	/

項目	説明
cmp - 2 つのファイルを比較する	*BF*、 *L*、 *RA*、 *RB*

詳しくは、 固定小数点比較命令の拡張ニーモニック を参照してください。

説明

cmp 命令は、汎用レジスター (GPR) *RA* の内容を GPR *RB* の内容と符号付き整数として比較し、条件レジスター・フィールド *BF* にビットの 1 つを設定します。

BF は、条件レジスター・フィールド 0 から 7 にすることができます。プログラマーは、操作の結果を示す条件レジスター・フィールドを指定できます。

条件レジスター・フィールド *BF* のビットは、次のように解釈されます。

項目	説明	
ビット	**Name**	**説明**
0	LT (L)	(RA) < SI
1	GT	(RA)> SI
2	EQ	(RA) = SI。
3	SO	SO、OV

cmp 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。 条件レジスター・フィールド 0 は、プログラマーによって *BF* として指定されていない限り、影響を受けません。

パラメーター

項目	説明
BF	比較の結果を示す条件レジスター・フィールド 0 から 7 を指定します。
L	32 ビット・サブセット・アーキテクチャーの場合は 0 に設定する必要があります。
RA	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

以下のコードは、GPR 4 および GPR 6 の内容を符号付き整数として比較し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xFFFF FFE7.
# Assume GPR 5 contains 0x0000 0011.
# Assume 0 is Condition Register Field 0.
cmp 0,4,6
# The LT bit of Condition Register Field 0 is set.
```

関連概念
cmpi (即時比較) 命令
cmpli (論理即時比較) 命令
固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

cmpi (即時比較) 命令

目的

汎用レジスターと指定された値の内容を代数的に比較します。

構文

ビット	VALUE
0-5	11
6-8	BF
9	/
10	L
11-15	RA
16-31	SI

項目	説明
cmpi (cmpi)	*BF*、 *L*、 *RA*、 *SI*

詳しくは、 固定小数点比較命令の拡張ニーモニック を参照してください。

説明

cmpi 命令は、汎用レジスター (GPR) *RA* と 16 ビットの符号付き整数 *SI* の内容を符号付き整数として比較し、条件レジスター・フィールド *BF* のビットの 1 つを設定します。

BF は、条件レジスター・フィールド 0 から 7 にすることができます。プログラマーは、操作の結果を示す条件レジスター・フィールドを指定できます。

条件レジスター・フィールド *BF* のビットは、次のように解釈されます。

項目	説明	
ビット	**Name**	**説明**
0	LT (L)	(RA) < SI
1	GT	(RA)> SI
2	EQ	(RA) = SI。
3	SO	SO、OV

cmpi 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。 条件レジスター・フィールド 0 は、プログラマーによって *BF* として指定されていない限り、影響を受けません。

パラメーター

項目	説明
BF	比較の結果を示す条件レジスター・フィールド 0 から 7 を指定します。

項目	説明
L	32 ビット・サブセット・アーキテクチャーの場合は 0 に設定する必要があります。
RA	操作のための最初のソース汎用レジスターを指定します。
SI	演算のための 16 ビットの符号付き整数を指定します。

例

以下のコードは、GPR 4 と符号付き整数 0x11 の内容を比較し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xFFFF FFE7.
cmpi 0,4,0x11
# The LT bit of Condition Register Field 0 is set.
```

関連概念

cmp (比較) 命令
cmpl (論理比較) 命令
cmpli (論理即時比較) 命令
固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

cmpl (論理比較) 命令

目的

2 つの汎用レジスターの内容を論理的に比較します。

構文

ビット	VALUE
0-5	31
6-8	BF
9	/
10	L
11-15	RA
16-20	RB
21-30	32
31	/

項目	説明
cmpl	*BF*、 *L*、 *RA*、 *RB*

詳しくは、 固定小数点比較命令の拡張ニーモニック を参照してください。

説明

cmpl 命令は、汎用レジスター (GPR) *RA* の内容を GPR *RB* の内容と符号なし整数として比較し、条件レジスター・フィールド *BF* にビットの 1 つを設定します。

BF は、条件レジスター・フィールド 0 から 7 にすることができます。プログラマーは、操作の結果を示す条件レジスター・フィールドを指定できます。

条件レジスター・フィールド *BF* のビットは、次のように解釈されます。

項目	説明	
ビット	Name	説明
0	LT (L)	(RA) < SI
1	GT	(RA)> SI
2	EQ	(RA) = SI。
3	SO	SO、OV

cmpl 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。 条件レジスター・フィールド 0 は、プログラマーによって *BF* として指定されていない限り、影響を受けません。

パラメーター

項目	説明
BF	比較の結果を示す条件レジスター・フィールド 0 から 7 を指定します。
L	32 ビット・サブセット・アーキテクチャーの場合は 0 に設定する必要があります。
RA	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

以下のコードは、GPR 4 と GPR 5 の内容を符号なし整数として比較し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xFFFF 0000.
# Assume GPR 5 contains 0x7FFF 0000.
# Assume 0 is Condition Register Field 0.
cmpl 0,4,5
# The GT bit of Condition Register Field 0 is set.
```

関連概念

cmp (比較) 命令
cmpi (即時比較) 命令
cmpli (論理即時比較) 命令
固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

cmpli (論理即時比較) 命令

目的

汎用レジスターの内容と特定の値を論理的に比較します。

構文

ビット	VALUE
0-5	10
6-8	BF

ビット	VALUE
9	/
10	L
11-15	RA
16-31	UI

項目	説明
cmpli (cmpli)	*BF*、 *L*、 *RA*、 *UI*

詳しくは、 固定小数点比較命令の拡張ニーモニック を参照してください。

説明

cmpli 命令は、汎用レジスター (GPR) *RA* の内容を、x`0000' と 16 ビット符号なし整数 *UI* を符号なし整数として連結したものと比較し、条件レジスター・フィールド *BF* のビットの 1 つを設定します。

BF は、条件レジスター・フィールド 0 から 7 にすることができます。プログラマーは、操作の結果を示す条件レジスター・フィールドを指定できます。

条件レジスター・フィールド *BF* のビットは、次のように解釈されます。

項目	説明	
ビット	**Name**	**説明**
0	LT (L)	(RA) < SI
1	GT	(RA)> SI
2	EQ	(RA) = SI。
3	SO	SO、OV

cmpli 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。 条件レジスター・フィールド 0 は、プログラマーによって *BF* として指定されていない限り、影響を受けません。

パラメーター

項目	説明
BF	比較の結果を示す条件レジスター・フィールド 0 から 7 を指定します。
L	32 ビット・サブセット・アーキテクチャーの場合は 0 に設定する必要があります。
RA	操作のソース汎用レジスターを指定します。
UI	演算用の 16 ビット符号なし整数を指定します。

例

以下のコードは、GPR 4 と符号なし整数 0xff の内容を比較し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0x0000 00ff.
cmpli 0,4,0xff
# The EQ bit of Condition Register Field 0 is set.
```

関連概念

cmp (比較) 命令

cmpi (即時比較) 命令
cmpli (論理即時比較) 命令
固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

cntlzd (先行ゼロ・ダブルワードのカウント) 命令

目的

高位ビットから始めて、汎用レジスターの内容の中の連続したゼロ・ビットの数をカウントします。

この命令は、64 ビット・アプリケーションを実行する 64 ビット PowerPC プロセッサーでのみ使用してください。

構文

ビット	VALUE
0-5	31
6 ～ 10	S
11-15	A
16-20	00000
21-30	58
31	rc

PowerPC64

cntlzd *rA*、 *rS* (Rc=0)

cntlzd。 *rA*、 *rS*(Rc=1)

説明

レジスター GPR *RS* のビット 0 (高位ビット) から始まる連続ゼロ・ビットの数のカウントが、GPR *RA* に入れられます。 この数値の範囲は 0 から 64 (両端を含む) です。

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

変更されたその他のレジスター:

条件レジスター (CR0 フィールド):

影響を受ける: LT、 GT、EQ、SO (Rc = 1 の場合)

注: Rc = 1 の場合、LT は CR0 フィールドでクリアされます。

パラメーター

項目	説明
RA	命令の結果のターゲット汎用レジスターを指定します。
RS	調べるダブルワードが入っているソース汎用レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

cntlzw または cntlz (先行ゼロ・ワードのカウント) 命令

目的

ソース汎用レジスター (GPR) 内の 32 ビット値の先行ゼロの数をカウントし、その結果を GPR に保管します。

構文

ビット	VALUE
0 - 5	31
6 ～ 10	RS
11-15	RA
16-20	///
21-30	26
31	rc

PowerPC (R)

cntlzw (cntlzw) *RA*、 *RS*

cntlzw。 *RA*、 *RS*

POWER® ファミリー

cntlz (cntlz) *RA*、 *RS*

cntlz。 *RA*、 *RS*

説明

cntlzw および **cntlz** 命令は、GPR *RS* の下位 32 ビットの連続ゼロ・ビットの数 (0 から 32) をカウントし、その結果をターゲット GPR *RA* に保管します。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
cntlzw (cntlzw)	なし	なし	0	なし
cntlzw。	なし	なし	1	LT、GT、EQ、SO
cntlz (cntlz)	なし	なし	0	なし
cntlz。	なし	なし	1	LT、GT、EQ、SO

cntlzw 命令の 2 つの構文形式と **cntlz** 命令の 2 つの構文形式は、固定小数点例外レジスターには影響を与えません。 構文形式でレコード (Rc) ビットが 1 に設定されている場合、命令は、**Condition Register** フィールド 0 の「Less Than (LT) zero」、「greater Than (GT) zero」、「Equal To (EQ) zero」、および「Summary Overflow (SO)」ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。

項目	説明
RS	操作のソース汎用レジスターを指定します。

例

以下のコードは、GPR 3 に含まれる 32 ビット値の先行ゼロの数をカウントし、結果を GPR 3 に戻します。

```
# Assume GPR 3 contains 0x0FFF FFFF 0061 9920.
cntlzw 3,3
# GPR 3 now holds 0x0000 0000 0000 0009. Note that the high-order 32 bits
 are ignored when computing the result.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点論理命令
固定小数点論理命令は、論理演算をビット単位で実行します。

crand (条件レジスター AND) 命令

目的

```
Places the result of ANDing two Condition Register bits in a Condition Register bit.
```

構文

ビット	VALUE
0-5	19
6 〜 10	BT
11-15	BA
16-20	BB
21-30	257
31	/

項目	説明
crand (crand)	*BT*、 *BA*、 *BB*

説明

crand 命令は、 *BA* で指定された条件レジスター・ビットと *BB* で指定された条件レジスター・ビットを論理的に AND し、結果を *BT* で指定されたターゲット条件レジスター・ビットに入れます。

crand 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
BT	操作の結果が保管されるターゲット条件レジスター・ビットを指定します。

項目	説明
BA	操作のソース条件レジスター・ビットを指定します。
BB	操作のソース条件レジスター・ビットを指定します。

例

以下のコードは、論理的に条件レジスター・ビット 0 と 5 を AND 演算し、その結果を条件レジスター・ビット 31 に保管します。

```
# Assume Condition Register bit 0 is 1.
# Assume Condition Register bit 5 is 0.
crand 31,0,5
# Condition Register bit 31 is now 0.
```

crandc (条件レジスター AND (Complement)) 命令

目的

1 つの条件レジスター・ビットと条件レジスター・ビットの補数を AND 演算した結果を条件レジスター・ビットに入れます。

構文

ビット	VALUE
0-5	19
6 ～ 10	BT
11-15	BA
16-20	BB
21-30	129
31	/

項目	説明
crandc (crandc)	*BT*、 *BA*、 *BB*

説明

crandc 命令は、*BA* で指定された条件レジスター・ビットと *BB* で指定された条件レジスター・ビットの補数を論理的に AND し、結果を *BT* で指定されたターゲット条件レジスター・ビットに入れます。

crandc 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
BT	操作の結果が保管されるターゲット条件レジスター・ビットを指定します。
BA	操作のソース条件レジスター・ビットを指定します。
BB	操作のソース条件レジスター・ビットを指定します。

例

以下のコードは、論理的に条件レジスター・ビット 0 と条件レジスター・ビット 5 の補数を AND 演算し、結果をビット 31 に入れます。

```
# Assume Condition Register bit 0 is 1.
# Assume Condition Register bit 5 is 0.
crandc 31,0,5
# Condition Register bit 31 is now 1.
```

creqv (条件レジスターと同等のもの) 命令

目的

2 つの条件レジスター・ビットの XOR の補完結果を条件レジスター・ビットに入れます。

構文

ビット	VALUE
0-5	19
6 ～ 10	BT
11-15	BA
16-20	BB
21-30	289
31	/

項目	説明
creqv (creqv)	*BT*、 *BA*、 *BB*

詳しくは、「Extended Mnemonics of Condition Register Logical Instructions 」を参照してください。

説明

creqv 命令は、 *BA* で指定された条件レジスター・ビットと *BB* で指定された条件レジスター・ビットを論理的に XOR し、補完された結果を *BT* で指定されたターゲット条件レジスター・ビットに入れます。

creqv 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
BT	操作の結果が保管されるターゲット条件レジスター・ビットを指定します。
BA	操作のソース条件レジスター・ビットを指定します。
BB	操作のソース条件レジスター・ビットを指定します。

以下のコードは、XOR 条件レジスター・ビット 8 および 4 の補完結果を条件レジスター・ビット 4 に入れます。

```
# Assume Condition Register bit 8 is 1.
# Assume Condition Register bit 4 is 0.
creqv 4,8,4
# Condition Register bit 4 is now 0.
```

crnand (条件レジスター NAND) 命令

目的

2 つの条件レジスター・ビットを AND 演算した結果を条件レジスター・ビットに入れます。

構文

ビット	VALUE
0-5	19
6 ～ 10	BT
11-15	BA
16-20	BB
21-30	225
31	/

項目	説明
crnand (crnand)	*BT*、 *BA*、 *BB*

説明

crnand 命令は、 *BA* で指定された条件レジスター・ビットと *BB* で指定された条件レジスター・ビットを論理的に AND 演算し、補完された結果を *BT* で指定されたターゲット条件レジスター・ビットに入れます。

crnand 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
BT	操作の結果が保管されるターゲット条件レジスター・ビットを指定します。
BA	操作のソース条件レジスター・ビットを指定します。
BB	操作のソース条件レジスター・ビットを指定します。

例

以下のコードは、論理的に条件レジスター・ビット 8 および 4 を AND 演算し、補完された結果を条件レジスター・ビット 4 に入れます。

```
# Assume Condition Register bit 8 is 1.
# Assume Condition Register bit 4 is 0.
crnand 4,8,4
# Condition Register bit 4 is now 1.
```

cror (条件レジスター NOR) 命令

目的

```
Places the complemented result of ORing two Condition Register bits in a Condition Register bit.
```

構文

ビット	VALUE
0-5	19
6 〜 10	BT
11-15	BA
16-20	BB
21-30	33
31	/

項目 **説明**

異常終了 *BT*、 *BA*、 *BB*

説明

cr 除外 命令は、*BA* で指定された条件レジスター・ビットと *BB* で指定された条件レジスター・ビットを論理和演算し、補完された結果を *BT* で指定されたターゲット条件レジスター・ビットに入れます。

cr かかわらず 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
BT	操作の結果が保管されるターゲット条件レジスター・ビットを指定します。
BA	操作のソース条件レジスター・ビットを指定します。
BB	操作のソース条件レジスター・ビットを指定します。

例

以下のコードは、条件レジスターのビット 8 と 4 を論理 OR 演算し、補完された結果を条件レジスターのビット 4 に保管します。

```
# Assume Condition Register bit 8 is 1.
# Assume Condition Register bit 4 is 0.
crnor 4,8,4
# Condition Register bit 4 is now 0.
```

cror (条件レジスター OR) 命令

目的

2 つの条件レジスター・ビットの OR 演算の結果を条件レジスター・ビットに入れます。

構文

ビット	VALUE
0-5	19
6 〜 10	BT
11-15	BA
16-20	BB
21-30	449

ビット	VALUE
31	/

項目 **説明**

cror (恐怖) *BT*、 *BA*、 *BB*

詳しくは、「Extended Mnemonics of Condition Register Logical Instructions 」を参照してください。

説明

cror 命令は、 *BA* によって指定された条件レジスター・ビットと *BB* によって指定された条件レジスター・ビットを論理和演算し、その結果を *BT* によって指定されたターゲット条件レジスター・ビットに入れます。

cror 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。

パラメーター

項目 **説明**

BT 操作の結果が保管されるターゲット条件レジスター・ビットを指定します。

BA 操作のソース条件レジスター・ビットを指定します。

BB 操作のソース条件レジスター・ビットを指定します。

例

以下のコードは、条件レジスター 8 および 4 の結果を条件レジスター 4 に入れます。

```
# Assume Condition Register bit 8 is 1.
# Assume Condition Register bit 4 is 0.
cror 4,8,4
# Condition Register bit 4 is now 1.
```

crorc (条件レジスターまたは準拠との OR) 命令

目的

条件レジスター・ビットの OR 演算の結果および条件レジスター・ビットの補数を条件レジスター・ビットに入れます。

構文

ビット	VALUE
0-5	19
6 〜 10	BT
11-15	BA
16-20	BB
21-30	417
31	/

項目 **説明**

crorc (crorc) *BT*、 *BA*、 *BB*

説明

crorc 命令は、 *BA* で指定された条件レジスター・ビットと、 *BB* で指定された条件レジスター・ビットの補数との論理和を取り、結果を *BT* で指定されたターゲット条件レジスター・ビットに入れます。

crorc 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
BT	操作の結果が保管されるターゲット条件レジスター・ビットを指定します。
BA	操作のソース条件レジスター・ビットを指定します。
BB	操作のソース条件レジスター・ビットを指定します。

例

以下のコードは、条件レジスター・ビット 8 と条件レジスター・ビット 4 の補数の結果を条件レジスター・ビット 4 に入れます。

```
# Assume Condition Register bit 8 is 1.
# Assume Condition Register bit 4 is 0.
crorc 4,8,4
# Condition Register bit 4 is now 1.
```

crxor (条件レジスター XOR) 命令

目的

2 つの条件レジスター・ビットの XOR の結果を条件レジスター・ビットに入れます。

構文

ビット	VALUE
0-5	19
6 〜 10	BT
11-15	BA
16-20	BB
21-30	193
31	/

項目	説明
crxor (crxor)	*BT*、 *BA*、 *BB*

詳しくは、「Extended Mnemonics of Condition Register Logical Instructions 」を参照してください。

説明

crxor 命令は、 *BA* で指定された条件レジスター・ビットと *BB* で指定された条件レジスター・ビットを論理的に XOR し、その結果を *BT* で指定されたターゲット条件レジスター・ビットに入れます。

crxor 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
BT	操作の結果が保管されるターゲット条件レジスター・ビットを指定します。
BA	操作のソース条件レジスター・ビットを指定します。
BB	操作のソース条件レジスター・ビットを指定します。

例

以下のコードは、XOR 条件レジスター・ビット 8 および 4 の結果を条件レジスター・ビット 4 に入れます。

```
# Assume Condition Register bit 8 is 1.
# Assume Condition Register bit 4 is 1.
crxor 4,8,4
# Condition Register bit 4 is now 0.
```

dcbf (Data Cache ・ブロック・フラッシュ) 命令

目的

変更されたキャッシュ・ブロックを主記憶域にコピーし、データ・キャッシュ内のコピーを無効にします。

注: dcbf 命令は、PowerPC ® アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0-5	31
6 ～ 10	///
11-15	RA
16-20	RB
21-30	86
31	/

PowerPC (R)

dcbf (dcbf) *RA*、 *RB*

説明

dcbf 命令は、汎用レジスター (GPR) *RA* の内容を GPR *RB* の内容に追加することによって、有効アドレス (EA) を計算します。 *RA* フィールドが 0 の場合、EA は *RB* と 0 の内容の合計です。 ターゲット・ストレージ・ロケーションを含むキャッシュ・ブロックがデータ・キャッシュ内にある場合、主ストレージ・コピーと異なる場合は、それが主ストレージにコピーされて戻されます。

dcbf 命令を使用する場合は、以下の点を考慮してください。

- EA によってアドレス指定されたバイトを含むブロックがデータ・キャッシュ内にあり、変更されている場合、そのブロックはメイン・メモリーにコピーされます。 EA によってアドレス指定されたバイトを含むブロックがキャッシュの 1 つにある場合、そのブロックは無効になります。
- EA が直接ストア・セグメント・アドレスを指定している場合、命令はノーオペレーションとして扱われます。

dcbf 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
RA	操作用のソース汎用レジスターを指定します。
rb	操作用のソース汎用レジスターを指定します。

例

ソフトウェアは、プロセッサーと別のシステム・コンポーネント (ストレージ・コヒーレンシー・プロトコルに参加しない入出力装置など) によって共有されるストレージのコヒーレンシーを管理します。 以下のコードは、別のシステム・コンポーネントがストレージにアクセスできるようにする前に、データ・キャッシュから共有ストレージをフラッシュします。

```
# Assume that the variable A is assigned to storage location
# 0x0000 4540.
# Assume that the storage location to which A is assigned
# contains 0.
# Assume that GPR 3 contains 0x0000 0040.
# Assume that GPR 4 contains 0x0000 4500.
# Assume that GPR 5 contains -1.
st      R5,R4,R3         # Store 0xFFFF FFFF to A
dcbf    R4,R3            # Flush A from cache to main memory
sync                     # Ensure dcbf is complete. Start I/O
                         # operation
```

ストアの後、 **dcbf** および **sync** 命令の実行前に、キャッシュ内の A のコピーには -1 が入ります。 ただし、メイン・メモリー内の A のコピーにはまだ 0 が含まれている可能性があります。 **sync** 命令が完了した後、A がメイン・メモリー内で割り当てられる位置には -1 が入り、プロセッサー・データ・キャッシュには位置 A のコピーが含まれなくなります。

dcbi (Data Cache ブロック無効化) 命令

目的

データ・キャッシュ内でアドレス指定されたバイトを含むブロックを無効にし、後続の参照でメイン・メモリーからブロックを再度取得します。

注: dcbi 命令は、PowerPC ® アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0-5	31
6 ～ 10	///
11-15	RA
16-20	RB
21-30	470
31	/

PowerPC (R)

dcbi *RA*、 *RB*

説明

汎用レジスター (GPR) *RA* の内容が 0 でない場合、 **dcbi** 命令は、GPR *RA* の内容を GPR *RB* の内容に追加することによって、有効アドレス (EA) を計算します。 それ以外の場合、EA は GPR *RB* の内容です。

アドレス指定されたバイトを含むキャッシュ・ブロックがデータ・キャッシュ内にある場合、そのブロックは無効になります。 その後、ブロック内の 1 バイトを参照すると、メイン・メモリーが参照されます。

dcbi 命令は、保護に関して、アドレス指定されたキャッシュ・ブロックへの保管として扱われます。

dcbi 命令の構文形式は 1 つだけであり、固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

セキュリティー

dcbi 命令は特権を持っています。

dcbst (Data Cache Block Store) 命令

目的

プログラムが変更されたブロックの内容をメイン・メモリーにコピーできるようにします。

注: dcbst 命令は、PowerPC ® アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0-5	31
6 ～ 10	///
11-15	RA
16-20	RB
21-30	54
31	/

PowerPC (R)

dcbst (dcbst)	*RA*、 *RB*

説明

dcbst 命令により、変更されたブロックのコピーがメイン・メモリーにコピーされます。 *RA* が 0 でない場合、 **dcbst** 命令は、汎用レジスター (GPR) *RA* の内容を GPR *RB* の内容に追加することによって、有効アドレス (EA) を計算します。 それ以外の場合、EA は *RB* の内容です。 アドレス指定されたバイトを含むキャッシュ・ブロックがデータ・キャッシュ内にあり、変更された場合、そのブロックはメイン・メモリーにコピーされます。

dcbst 命令を使用して、メイン・メモリー内のロケーションのコピーに最新の更新が含まれるようにすることができます。 これは、コヒーレンス・プロトコルに参加していない入出力装置とメモリーを共有する場合に重要になることがあります。 さらに、 **dcbst** 命令は、更新が即時にグラフィックス・フレーム・バッファーにコピーされるようにすることができます。

dcbst 命令を、アドレス変換およびアドレス保護に関して、アドレス指定されたバイトからのロードとして扱います。

dcbst 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

1. 以下のコードは、コヒーレンス・プロトコルに関与しない入出力装置とメモリーを共有します。

```
# Assume that location A is memory that is shared with the
# I/O device.
# Assume that GPR 2 contains a control value indicating that
# and I/O operation should start.
# Assume that GPR 3 contains the new value to be placed in
# location A.
# Assume that GPR 4 contains the address of location A.
# Assume that GPR 5 contains the address of a control register
# in the I/O device.
st      3,0,4           # Update location A.
dcbst   0,4             # Copy new content of location A and
                        # other bytes in cache block to main
                        # memory.
sync                    # Ensure the dcbst instruction has
                        # completed.
st      2,0,5           # Signal I/O device that location A has
                        # been update.
```

2. 以下のコードは、新しい値が遅延なく表示されるように、グラフィックス・フレーム・バッファーにコピーします。

```
# Assume that target memory is a graphics frame buffer.
# Assume that GPR 2, 3, and 4 contain new values to be displayed.
# Assume that GPR 5 contains the address minus 4 of where the
# first value is to be stored.
# Assume that the 3 target locations are known to be in a single
# cache block.
addi    6,5,4           # Compute address of first memory
                        # location.
stwu    2,4(5)          # Store value and update address ptr.
stwu    3,4(5)          # Store value and update address ptr.
stwu    4,4(5)          # Store value and update address ptr.
dcbst   0,6             # Copy new content of cache block to
                        # frame buffer. New values are displayed.
```

dcbt (Data Cache Block Touch) 命令

目的

プログラムが実際に必要とする前に、プログラムがキャッシュ・ブロック・フェッチを要求できるようにします。

注 : dcbt 命令は、 POWER5™ 以降のアーキテクチャーでサポートされます。

構文

ビット	VALUE
0-5	31
6 〜 10	TH
11-15	RA
16-20	RB

ビット	VALUE
21-30	278
31	/

POWER5™

dcbt (dcbt) *RA* 、 *RB*、 *TH*

説明

dcbt 命令は、アドレス指定されたバイトからのロードを予期することにより、パフォーマンスを向上させることができます。 プログラムがブロックを必要とする前に、有効アドレス (EA) によってアドレス指定されたバイトを含むブロックがデータ・キャッシュにフェッチされます。 プログラムは、後でブロックからロードを実行することができますが、ブロックをキャッシュにフェッチすることによって生じる追加の遅延を経験することはありません。 **dcbt** 命令を実行しても、システム・エラー・ハンドラーは呼び出されません。

汎用レジスター (GPR) *RA* が 0 でない場合、有効アドレス (EA) は、GPR *RA* の内容と GPR *RB* の内容の合計になります。 それ以外の場合、EA は GPR *RB* の内容です。

dcbt 命令を使用する場合は、以下のことを考慮してください。

- EA が直接ストア・セグメント・アドレスを指定している場合、命令はノーオペレーションとして扱われます。
- アクセスは、保護に関しては、アドレス指定されたキャッシュ・ブロックからのロードとして扱われます。 保護によってアドレス指定されたバイトへのアクセスが許可されない場合、**dcbt** 命令は操作を実行しません。

  注 : プログラムがデータ・キャッシュ・ブロックに保管する必要がある場合は、**dcbtst** (Data Cache Block Touch for Store) 命令を使用します。

タッチ・ヒント *(TH)* フィールドは、EA および *TH* フィールドで指定されたストレージ・ロケーションからプログラムがすぐにロードされる可能性があることを示すヒントを提供するために使用されます。 キャッシング禁止または保護されているロケーションの場合、ヒントは無視されます。 *TH* フィールドのエンコードは、 **-m** フラグまたは .machine pseudo-op で選択されたターゲット・アーキテクチャーによって異なります。 POWER5™ およびそれ以降のアーキテクチャーでの *TH* フィールドのエンコードは、以下のとおりです。

TH 値	説明
0000	プログラムは、EA によってアドレス指定されたバイトからすぐにロードされる可能性があります。
0001	プログラムは、おそらく、EA によってアドレス指定されたバイトを含むブロックと、それに続く無制限の数の順次ブロックで構成されるデータ・ストリームからすぐにロードされます。 順次先行ブロックは、EA + n * block_size (n = 0、1、2 など) によってアドレス指定されるバイトです。
0011	プログラムは、おそらく、EA によってアドレス指定されたバイトを含むブロックと、無制限の数の順次先行ブロックで構成されるデータ・ストリームからすぐにロードされます。 順次先行ブロックは、EA-n * block_size (n = 0、1、2、以下同様) によってアドレス指定されるバイトです。
1000	**dcbt** 命令は、データ・ストリームの特定の属性を記述するヒントを提供し、オプションで、プログラムがストリームから間もなくロードされる可能性があることを示します。 EA は、 167 ページの表 37 で説明されているように解釈されます。

TH 値	説明
1010	**dcbt** 命令は、データ・ストリームの特定の属性を記述するヒントを提供します。あるいは、TH [0] = 1 を指定した **dcbt** 命令を使用して記述されたデータ・ストリームからプログラムがすぐにロードされること、またはそのようなデータ・ストリームからプログラムがロードされなくなることを示します。 EA は、 167 ページの表 38 で説明されているように解釈されます。

dcbt 命令は、基本ニーモニックと拡張ニーモニックの両方として機能します。 3 つのオペランドを持つ **dcbt** 簡略記号は基本形式であり、2 つのオペランドを持つ **dcbt** は拡張形式です。 拡張形式では、 *TH* フィールドは省略され、 0b0000 であると想定されます。

表 *37. TH=0b1000* の場合の *EA* エンコード

ビット	名前	説明
0-56	EA_TRUNC	データ・ストリームの最初の単位の実効アドレスの上位 57 ビット。
57	D	方向 **0** 後続の単位は、順次後続の単位です。 **1** 後続の単位は、順次先行する単位です。
58	UG	**0** UG フィールドによって情報が提供されることはありません。 **1** データ・ストリーム内の単位数には制限がなく、ストリームの各ブロックに対するプログラムの必要性は一時的なものではない可能性があり、プログラムはストリームから間もなくロードされる可能性があります。
59	予約済み	予約済み
60–63	ID	このストリームに使用するストリーム ID。

表 *38. EA TH=0b1010* の場合のエンコード

ビット	名前	説明
0-31	予約済み	予約済み
32	GO	**0** GO フィールドによって情報が提供されない **1** プログラムは、完全に記述されたすべての初期データ・ストリームからすぐにロードされ、他のすべてのデータ・ストリームからはロードされなくなる可能性があります。

表 *38. EA TH=0b1010* の場合のエンコード *(続き)*		
ビット	**名前**	**説明**
33-34	S	停止 **00** S フィールドには情報は提供されません。 **01** 予約済み **10** プログラムは、ストリーム ID に関連付けられたストリームからロードしなくなる可能性があります (EA の他のすべてのフィールドは、ID フィールドを除き無視されます)。 **11** プログラムは、すべてのストリーム ID に関連付けられたデータ・ストリームからロードしなくなる可能性があります (EA の他のすべてのフィールドは無視されます)。
35-46	予約済み	予約済み
47-56	単位の通信数	データ・ストリーム内の単位数。
57	T	**0** T フィールドには情報が提供されません。 **1** データ・ストリームの各ブロックに対するプログラムの必要性は一時的なものである可能性があります (つまり、プログラムがブロックにアクセスする時間間隔は短くなる可能性があります)。
58	U	**0** U フィールドには情報は提供されません。 **1** データ・ストリーム内の単位数は無制限です (UNIT_CNT フィールドは無視されます)。
59	予約済み	予約済み
60-63	ID	このストリームに使用するストリーム ID。

dcbt 命令には 1 つの構文形式があり、条件レジスター・フィールド 0 または固定小数点例外レジスターには影響しません。

パラメーター

項目	**説明**
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。
TH	データ・キャッシュ・ブロックのシーケンスがいつ必要になる可能性があるかを示します。

例

以下のコードは、1 次元ベクトルの内容を合計します。

```
# Assume that GPR 4 contains the address of the first element
# of the sum.
# Assume 49 elements are to be summed.
# Assume the data cache block size is 32 bytes.
```

```
# Assume the elements are word aligned and the address
# are multiples of 4.
        dcbt    0,4             # Issue hint to fetch first
                                # cache block.
        addi    5,4,32          # Compute address of second
                                # cache block.
        addi    8,0,6           # Set outer loop count.
        addi    7,0,8           # Set inner loop counter.
        dcbt    0,5             # Issue hint to fetch second
                                # cache block.
        lwz     3,4,0           # Set sum = element number 1.
bigloop:
        addi    8,8,-1          # Decrement outer loop count
                                # and set CR field 0.
        mtspr   CTR,7           # Set counter (CTR) for
                                # inner loop.
        addi    5,5,32          # Computer address for next
                                # touch.
lttlloop:
        lwzu    6,4,4           # Fetch element.
        add     3,3,6           # Add to sum.
        bc      16,0,lttlloop   # Decrement CTR and branch
                                # if result is not equal to 0.
        dcbt    0,5             # Issue hint to fetch next
                                # cache block.
        bc      4,3,bigloop     # Branch if outer loop CTR is
                                # not equal to 0.
        end                     # Summation complete.
```

dcbtst (Data Cache Block Touch for Store) 命令

目的

プログラムが実際に必要とする前に、プログラムがキャッシュ・ブロック・フェッチを要求できるようにします。

構文

ビット	VALUE
0-5	31
6 ～ 10	TH
11-15	RA
16-20	RB
21-30	246
31	/

PowerPC (R)

dcbtst (dcbtst) *RA*、 *RB*、 *TH*

説明

dcbtst 命令は、アドレス指定されたバイトへの保管を予期することによってパフォーマンスを向上させます。 プログラムがブロックを必要とする前に、有効アドレス (EA) によってアドレス指定されたバイトを含むブロックがデータ・キャッシュにフェッチされます。 プログラムは、後でブロックに対して保管を実行することができますが、ブロックをキャッシュにフェッチすることによって生じる追加の遅延を経験することはありません。 **dcbtst** 命令を実行しても、システム・エラー・ハンドラーは呼び出されません。

dcbtst 命令は、汎用レジスター (GPR) *RA* の内容を GPR *RB* の内容に追加することによって、有効アドレス (EA) を計算します。 *RA* フィールドが 0 の場合、EA は *RB* と 0 の内容の合計です。

dcbtst 命令を使用する場合は、以下のことを考慮してください。

- EA が直接ストア・セグメント・アドレスを指定している場合、命令はノーオペレーションとして扱われます。
- アクセスは、保護に関しては、アドレス指定されたキャッシュ・ブロックからのロードとして扱われます。 保護によってアドレス指定されたバイトへのアクセスが許可されない場合、**dcbtst** 命令は操作を実行しません。
- プログラムがデータ・キャッシュ・ブロックに保管する必要がない場合は、**dcbt** (Data Cache Block Touch) 命令を使用します。

dcbtst 命令には 1 つの構文形式があり、条件レジスター・フィールド 0 または固定小数点例外レジスターには影響しません。

Touch Hint *(TH)* フィールドは、プログラムが EA および *TH* フィールドで指定されたストレージ・ロケーションにすぐに保管する可能性があることを示すヒントを提供するために使用されます。 キャッシング禁止または保護されているロケーションの場合、ヒントは無視されます。 *TH* フィールドのエンコードは、 **-m** フラグまたは .machine pseudo-op で選択されたターゲット・アーキテクチャーによって異なります。 *TH* フィールドのエンコードは、 **dcbt** 命令の場合と同じになります。

dcbtst 命令は、基本ニーモニックと拡張ニーモニックの両方として機能します。 3 つのオペランドを持つ **dcbtst** 簡略記号は基本形式であり、2 つのオペランドを持つ **dcbtst** は拡張形式です。 拡張形式では、 *TH* オペランドは省略され、0 と見なされます。 POWER5™ およびそれ以降のアーキテクチャーでの *TH* フィールドのエンコードは、以下のとおりです。

TH 値	**説明**
0000	プログラムは、EA によってアドレス指定されたバイトに保管される可能性があります。
0001	プログラムは、EA によってアドレス指定されたバイトを含むブロックと、それに続く順次ブロックの数に制限のないブロックで構成されるデータ・ストリームに保管される可能性があります。 順次先行ブロックは、EA + n * block_size (n = 0、1、2 など) によってアドレス指定されるバイトです。
0011	プログラムは、EA によってアドレス指定されたバイトを含むブロックと、無制限の数の順次先行ブロックからなるデータ・ストリームに保管される可能性があります。 順次先行ブロックは、EA-n * block_size (n = 0、1、2、以下同様) によってアドレス指定されるバイトです。
1000	**dcbt** 命令は、データ・ストリームの特定の属性を記述するヒントを提供し、オプションで、プログラムがおそらくストリームに保管することを示します。 EA は、TH=0b1000 の場合の EA エンコード で説明されているように解釈されます。
1010	**dcbt** 命令は、データ・ストリームの特定の属性を記述するヒントを提供します。または、TH [0] = 1 を指定した **dcbt** 命令を使用して記述されたデータ・ストリームにプログラムが保管する可能性が高いこと、またはそのようなデータ・ストリームにプログラムが保管しない可能性が高いことを示します。 EA は、 TH=0b1010 の場合の EA エンコード で説明されているように解釈されます。

パラメーター

項目	**説明**
RA	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。
TH	データ・キャッシュ・ブロックのシーケンスがいつ変更される可能性があるかを示します。

dcbz または dclz (Data Cache Block Set to Zero) 命令

目的

PowerPC ® 命令 **dcbz** は、キャッシュ・ブロックのすべてのバイトを 0 に設定します。

POWER ® ファミリー命令 **dclz** は、キャッシュ・ラインのすべてのバイトを 0 に設定します。

構文

ビット	VALUE
0-5	31
6 〜 10	///
11-15	RA
16-20	RB
21-30	1014
31	/

PowerPC (R)

dcbz (dcbz) *RA*、 *RB*

POWER ® ファミリー

dclz (dclz) *RA*、 *RB*

説明

dcbz 命令と **dclz** 命令は、それぞれデータ・キャッシュ・ブロックとデータ・キャッシュ・ラインを処理します。 *RA* が 0 でない場合、 **dcbz** および **dclz** 命令は、汎用レジスター (GPR) *RA* の内容を GPR *RB* の内容に追加することによって、有効アドレス (EA) を計算します。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

アドレス指定されたバイトを含むキャッシュ・ブロックまたは行がデータ・キャッシュ内にある場合、ブロックまたは行内のすべてのバイトが 0 に設定されます。 それ以外の場合、ブロックまたは行はストレージを参照せずにデータ・キャッシュ内に設定され、ブロックまたは行のすべてのバイトは 0 に設定されます。

POWER ® ファミリー命令 **dclz** では、GPR *RA* が 0 でない場合、EA は GPR *RA* の内容を置き換えます。

dcbz および **dclz** 命令は、保護に関しては、アドレス指定されたキャッシュ・ブロックまたは回線への保管として扱われます。

dcbz および **dclz** 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。 ビット 31 が 1 に設定されている場合、命令形式は無効です。

パラメーター

PowerPC (R)

RA	EA 計算のソース・レジスターを指定します。
rb	EA 計算のソース・レジスターを指定します。

POWER ® ファミリー

RA	EA 計算用のソース・レジスターと EA 更新用のターゲット・レジスターを指定します。
rb	EA 計算のソース・レジスターを指定します。

セキュリティー

dclz 命令には特権があります。

dclst (Data Cache ・ライン・ストア) 命令

目的

データ・キャッシュ内の変更されたデータの行をメイン・メモリーに保管します。

構文

ビット	VALUE
0-5	31
6 ～ 10	///
11-15	RA
16-20	RB
21-30	630
31	rc

POWER® ファミリー

dclst (dclst) *RA*、 *RB*

説明

dclst 命令は、汎用レジスター (GPR) *RA* の内容を GPR *RB* の内容に追加します。 次に、 *RA* が 0 でなく、命令によってデータ・ストレージ割り込みが発生しない場合は、合計を有効アドレス (EA) として *RA* に保管します。

RA が 0 の場合、有効アドレス (EA) は GPR *RB* の内容の合計と 0 です。

dclst 命令を使用する場合は、以下のことを考慮してください。

- EA によってアドレス指定されたバイトを含む行がデータ・キャッシュ内にあり、変更されている場合、**dclst** 命令はその行をメイン・メモリーに書き込みます。
- データ・アドレス変換が使用可能で (つまり、マシン状態レジスター (MSR) データ再配置 (DR) ビットが 1)、仮想アドレスに変換がない場合、データ・ストレージ割り込みセグメント・レジスターのビット 1 が 1 に設定された状態でデータ・ストレージ割り込みが発生します。
- データ・アドレス変換が使用可能な場合 (MSR DR ビットが 1)、仮想アドレスは使用不可の実アドレスに変換され、回線はデータ・キャッシュに存在し、マシン・チェック割り込みが発生します。
- データ・アドレス変換が使用不可 (MSR DR ビットが 0) の場合、アドレスは使用不可の実アドレスを指定し、回線はデータ・キャッシュに存在し、マシン・チェック割り込みが発生します。
- EA が入出力アドレスを指定すると、命令はノーオペレーションとして扱われますが、有効アドレスは GPR *RA* に入れられます。
- アドレス変換は、 **dclst** 命令を、保護およびデータ・ロックを無視して、アドレス指定されたバイトへのロードとして扱います。 この命令が原因で変換ルック・アサイド・バッファー (TLB) ミスが発生した場合、参照ビットが設定されます。

dclst 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。 レコード (Rc) ビットが 1 に設定されている場合、条件レジスター・フィールド 0 は未定義です。

パラメーター

項目	説明
RA	操作の結果が保管されるソースおよびターゲット汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、GPR 4 と GPR 6 の内容の合計を有効アドレスとして GPR 6 に保管します。

```
# Assume that GPR 4 contains 0x0000 3000.
# Assume that GPR 6 is the target register and that it
# contains 0x0000 0000.
dclst 6,4
# GPR 6 now contains 0x0000 3000.
```

div (除算) 命令

目的

MQ レジスターに連結された汎用レジスターの内容を汎用レジスターの内容で除算し、その結果を汎用レジスターに保管します。

注: div 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0-5	31
6 〜 10	RT
11-15	RA
16-20	RB
21	大江
22-30	331
31	rc

POWER® ファミリー

div	*RT*、 *RA*、 *RB*
Div.	*RT*、 *RA*、 *RB*
divo	*RT*、 *RA*、 *RB*
ディボ	*RT*、 *RA*、 *RB*

説明

div 命令は、汎用レジスター (GPR) *RA* の内容と Multiply Quotient (MQ) レジスターの内容を連結し、その結果を GPR *RB* の内容で除算し、その結果をターゲット GPR *RT* に保管します。 剰余は被除数と同じ符号を持ちますが、ゼロ商またはゼロ剰余は常に正の値になります。 結果は次の式に従います。

```
dividend = (divisor x quotient) + remainder
```

ここで、 `dividend` は元の (RA) || (MQ)、 `divisor` は元の (*RB*)、 `quotient` は最終 (*RT*)、 `remainder` は最終 (MQ) です。

`-2**31 P -1` の場合、 MQ Register は 0 に設定され、 `-2**31` は GPR *RT* に入れられます。 その他のすべてのオーバーフローの場合、 MQ、ターゲット GPR *RT*、および条件レジスター・フィールド 0 (レコード・ビット (Rc) が 1 の場合) の内容は未定義です。

div 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
div	0	なし	0	なし
Div.	0	なし	1	LT、GT、EQ、SO
divo	1	SO、OV	0	なし
ディボ	1	SO、OV	1	LT、GT、EQ、SO

div 命令の 4 つの構文形式は、固定小数点例外レジスターのカリー・ビット (CA) には影響しません。 構文形式がオーバーフロー例外 (OE) ビットを 1 に設定する場合、命令は固定小数点例外レジスターの要約オーバーフロー (SO) およびオーバーフロー (OV) ビットに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、 MQ レジスターに連結された GPR 4 の内容を GPR 6 の内容で分割し、結果を GPR 4 に保管します。

```
# Assume the MQ Register contains 0x0000 0001.
# Assume GPR 4 contains 0x0000 0000.
# Assume GPR 6 contains 0x0000 0002.
div 4,4,6
# GPR 4 now contains 0x0000 0000.
# The MQ Register now contains 0x0000 0001.
```

2. 以下のコードは、 MQ レジスターと連結された GPR 4 の内容を GPR 6 の内容で除算し、結果を GPR 4 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume the MQ Register contains 0x0000 0002.
# Assume GPR 4 contains 0x0000 0000.
# Assume GPR 6 contains 0x0000 0002.
div. 4,4,6
# GPR 4 now contains 0x0000 0001.
# MQ Register contains 0x0000 0000.
```

3. 以下のコードは、 MQ レジスターに連結された GPR 4 の内容を GPR 6 の内容で分割し、結果を GPR 4 に入れ、演算の結果を反映するために固定小数点例外レジスターに要約オーバーフロー・ビットとオーバーフロー・ビットを設定します。

```
# Assume GPR 4 contains 0x0000 0001.
# Assume GPR 6 contains 0x0000 0000.
# Assume the MQ Register contains 0x0000 0000.
divo 4,4,6
# GPR 4 now contains an undefined quantity.
# The MQ Register is undefined.
```

4. 以下のコードは、 MQ Register と連結された GPR 4 の内容を GPR 6 の内容で分割し、結果を GPR 4 に入れ、要約オーバーフロー・ビットとオーバーフロー・ビットを固定小数点例外レジスターおよび条件レジスター・フィールド 0 に設定して、演算の結果を反映します。

```
# Assume GPR 4 contains 0x-1.
# Assume GPR 6 contains 0x2.
# Assume the MQ Register contains 0xFFFFFFFF.
divo. 4,4,6
# GPR 4 now contains 0x0000 0000.
# The MQ Register contains 0x-1.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

divd (除算ダブルワード) 命令

目的

汎用レジスターの内容を汎用レジスターの内容で除算し、その結果を汎用レジスターに格納します。

この命令は、64 ビット・アプリケーションを実行する 64 ビット PowerPC プロセッサーでのみ使用してください。

構文

ビット	VALUE
0-5	31
6 〜 10	D
11-15	A
16-20	B
21	大江
22-30	489
31	rc

PowerPC64

div	*RT*、 *RA*、 *RB* (OE=0 Rc=0)
ディビド	*RT*、 *RA*、 *RB* (OE=0 Rc=1)
dido	*RT*、 *RA*、 *RB* (OE=1 Rc=0)
ディヴォド	*RT*、 *RA*、 *RB* (OE=1 Rc=1)

説明

64 ビットの配当は、*RA* の内容です。 64 ビット除数は、*RB* の内容です。 64 ビットの商は *RT* に入れられます。 残りは結果として提供されません。

オペランドと商は両方とも符号付き整数として解釈されます。 商は、式-被除数 = (quotient * divisor) + r を満たす固有の符号付き整数です。ここで、0 < = r < |divisor | (被除数が負でない場合) および-|divisor | < r < = 0 (被除数が負の場合) です。

0x8000_0000_0000_0000 /-1 または/0 の除算を実行しようとすると、条件レジスター 0 フィールドの LT、 GT、および EQ ビットの内容と同様に、 *RT* の内容は未定義になります (レコード・ビット (Rc) = 1 (**did) の場合)**。 または **divdo**)) を参照してください。 この場合、オーバーフロー使用可能 (OE) = 1 であれば、オーバーフロー・ビット (OV) が設定されます。

(*RA*) を (*RB*) で除算した 64 ビット符号付き剰余は、(*RA*) = -2 * *63 かつ (*RB*) = -1 の場合を除き、次のように計算できます。

項目	**説明**	
div	**RT、RA、RB**	# *RT* = quotient
マルチルド	**RT、RT、RB**	# *RT* = quotient * 除数
サブ	**RT、RT、RA**	# *RT* = 剰余

パラメーター

項目	**説明**
RT	計算の結果のターゲット汎用レジスターを指定します。
RA	被除数のソース汎用レジスターを指定します。
rb	除数のソース汎用レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

divdu (Double Word Unsigned の除算) 命令

目的

汎用レジスターの内容を汎用レジスターの内容で除算し、その結果を汎用レジスターに格納します。

構文

ビット	VALUE
0-5	31
6 ～ 10	D
11-15	A
16-20	B
21	大江
22-30	457
31	rc

PowerPC (R)

didu (didu)	*RT*、 *RA*、 *RB* (OE=0 Rc=0)
ディヴドゥ	*RT*、 *RA*、 *RB* (OE=0 Rc=1)
divduo (divduo)	*RT*、 *RA*、 *RB* (OE=1 Rc=0)
ディヴドゥー	*RT*、 *RA*、 *RB* (OE=1 Rc=1)

説明

64 ビットの配当は、*RA* の内容です。 64 ビット除数は、*RB* の内容です。 64 ビットの商は *RT* に入れられます。 残りは結果として提供されません。

オペランドと商は両方とも符号なし整数として解釈されます。ただし、レコード・ビット (Rc) が 1 に設定されている場合は、条件レジスター 0 (CR0) フィールドの最初の 3 ビットは、結果の符号付き比較によってゼロに設定されます。 商は、次の式を満たす固有の符号なし整数です。被除数 = (quotient * divisor) + r。ここで、0 < = r < 除数。

除算 (すべて)/0 を実行しようとすると、 CR0 フィールドの LT、 GT、および EQ ビット (Rc = 1 の場合) の内容と同様に、 *RT* の内容は未定義になります。 この場合、オーバーフロー使用可能ビット (OE) = 1 であれば、オーバーフロー・ビット (OV) が設定されます。

64 ビット符号なし剰余 (*RA*) を (*RB*) で除算すると、次のように計算できます。

項目	説明	
didu (didu)	**RT、RA、RB**	# *RT* = quotient
マルチルド	**RT、RT、RB**	# *RT* = quotient * 除数
サブ	**RT、RT、RA**	# *RT* = 剰余

変更されたその他のレジスター:

- 条件レジスター (CR0 フィールド):

  影響を受ける: LT、 GT、EQ、SO (Rc = 1 の場合)
- XER: 影響を受ける: SO、OV (OE = 1 の場合)

  注: XER 内の影響を受けるビットの設定はモードに依存せず、64 ビットの結果のオーバーフローを反映します。

パラメーター

項目	説明
RT	計算の結果のターゲット汎用レジスターを指定します。
RA	被除数のソース汎用レジスターを指定します。
rb	除数のソース汎用レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

Div (Bride Short) 命令

目的

汎用レジスターの内容を汎用レジスターの内容で分割し、その結果を汎用レジスターに保管します。

注: divs 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0-5	31
6 ～ 10	RT
11-15	RA

ビット	VALUE
16-20	RB
21	
22-30	363
31	rc

POWER® ファミリー

div *RT*、 *RA*、 *RB*

div。 *RT*、 *RA*、 *RB*

divso (divso) *RT*、 *RA*、 *RB*

ディヴソ *RT*、 *RA*、 *RB*

説明

divs 命令は、汎用レジスター (GPR) *RA* の内容を GPR *RB* の内容で分割し、結果をターゲット GPR *RT* に保管します。 剰余は被除数と同じ符号を持ちますが、ゼロ商またはゼロ剰余は常に正の値になります。 結果は次の式に従います。

```
dividend = (divisor x quotient) + remainder
```

ここで、 `dividend` は元の (*RA*)、 `divisor` は元の (*RB*)、 `quotient` は最終 (*RT*)、 `remainder` は最終 (MQ) です。

`-2**31 P -1` の場合、 MQ Register は 0 に設定され、 `-2**31` は GPR *RT* に入れられます。 その他のすべてのオーバーフローの場合、 MQ、ターゲット GPR *RT* 、および条件レジスター・フィールド 0 (レコード・ビット (Rc) が 1 の場合) の内容は未定義です。

divs 命令には 4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
div	0	なし	0	なし
div。	0	なし	1	LT、GT、EQ、SO
divso (divso)	1	SO、OV	0	なし
ディヴソ	1	SO、OV	1	LT、GT、EQ、SO

divs 命令の 4 つの構文形式は、固定小数点例外レジスターのカリー・ビット (CA) には影響しません。 構文形式がオーバーフロー例外 (OE) ビットを 1 に設定する場合、命令は固定小数点例外レジスターの要約オーバーフロー (SO) およびオーバーフロー (OV) ビットに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。

項目	説明
RA	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容を GPR 6 の内容で分割し、その結果を GPR 4 に保管します。

```
# Assume GPR 4 contains 0x0000 0001.
# Assume GPR 6 contains 0x0000 0002.
divs 4,4,6
# GPR 4 now contains 0x0.
# The MQ Register now contains 0x1.
```

2. 以下のコードは、GPR 4 の内容を GPR 6 の内容で除算し、結果を GPR 4 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0x0000 0002.
# Assume GPR 6 contains 0x0000 0002.
divs. 4,4,6
# GPR 4 now contains 0x0000 0001.
# The MQ Register now contains 0x0000 0000.
```

3. 以下のコードは、GPR 4 の内容を GPR 6 の内容で分割し、結果を GPR 4 に保管し、演算の結果を反映するために固定小数点例外レジスターにサマリー・オーバーフロー・ビットとオーバーフロー・ビットを設定します。

```
# Assume GPR 4 contains 0x0000 0001.
# Assume GPR 6 contains 0x0000 0000.
divso 4,4,6
# GPR 4 now contains an undefined quantity.
```

4. 以下のコードは、GPR 4 の内容を GPR 6 の内容で分割し、結果を GPR 4 に保管し、演算の結果を反映するために、固定小数点例外レジスターおよび条件レジスター・フィールド 0 にサマリー・オーバーフロー・ビットとオーバーフロー・ビットを設定します。

```
# Assume GPR 4 contains 0x-1.
# Assume GPR 6 contains 0x0000 00002.
# Assume the MQ Register contains 0x0000 0000.
divso. 4,4,6
# GPR 4 now contains 0x0000 0000.
# The MQ register contains 0x-1.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

divw (ワードの分割) 命令

目的

汎用レジスターの内容を別の汎用レジスターの内容で分割し、その結果を 3 番目の汎用レジスターに保管します。

注: divw 命令は、PowerPC ® アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0-5	31
6 ～ 10	RT
11-15	RA
16-20	RB
21	大江
22-30	491
31	rc

PowerPC (R)

divw	*RT*、*RA*、*RB*
ディヴュー	*RT*、*RA*、*RB*
divwo (divwo)	*RT*、*RA*、*RB*
ディヴォウォ	*RT*、*RA*、*RB*

説明

divw 命令は、汎用レジスター (GPR) *RA* の内容を GPR *RB* の内容で分割し、その結果をターゲット GPR *RT* に保管します。 被除数、除数、および商は、符号付き整数として解釈されます。

-2 * *31/-1 の場合、およびオーバーフローの原因となる他のすべての場合については、GPR *RT* の内容は未定義です。

divw 命令には 4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	**説明**			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
divw	0	なし	0	なし
ディヴュー	0	なし	1	LT、GT、EQ、SO
divwo (divwo)	1	SO、OV	0	なし
ディヴォウォ	1	SO、OV	1	LT、GT、EQ、SO

divw 命令の 4 つの構文形式は、固定小数点例外レジスターのカリー・ビット (CA) には影響しません。 構文形式がオーバーフロー例外 (OE) ビットを 1 に設定する場合、命令は固定小数点例外レジスターの要約オーバーフロー (SO) およびオーバーフロー (OV) ビットに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	**説明**
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	配当用のソース汎用レジスターを指定します。

項目	説明
rb	除数のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容を GPR 6 の内容で分割し、その結果を GPR 4 に保管します。

```
# Assume GPR 4 contains 0x0000 0000.
# Assume GPR 6 contains 0x0000 0002.
divw 4,4,6
# GPR 4 now contains 0x0000 0000.
```

2. 以下のコードは、GPR 4 の内容を GPR 6 の内容で除算し、結果を GPR 4 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0x0000 0002.
# Assume GPR 6 contains 0x0000 0002.
divw. 4,4,6
# GPR 4 now contains 0x0000 0001.
```

3. 以下のコードは、GPR 4 の内容を GPR 6 の内容で除算し、結果を GPR 4 に入れ、演算の結果を反映するために固定小数点例外レジスターに要約オーバーフロー・ビットとオーバーフロー・ビットを設定します。

```
# Assume GPR 4 contains 0x0000 0001.
# Assume GPR 6 contains 0x0000 0000.
divwo 4,4,6
# GPR 4 now contains an undefined quantity.
```

4. 以下のコードは、GPR 4 の内容を GPR 6 の内容で除算し、結果を GPR 4 に入れ、演算の結果を反映するために、固定小数点例外レジスターおよび条件レジスター・フィールド 0 にサマリー・オーバーフロー・ビットとオーバーフロー・ビットを設定します。

```
# Assume GPR 4 contains 0x8000 0000.
# Assume GPR 6 contains 0xFFFF FFFF.
divwo. 4,4,6
# GPR 4 now contains undefined quantity.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

divwu (ワード符号なし除算) 命令

目的

汎用レジスターの内容を別の汎用レジスターの内容で分割し、その結果を 3 番目の汎用レジスターに保管します。

注: divwu 命令は PowerPC ® アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0-5	31

ビット	VALUE
6 ～ 10	RT
11-15	RA
16-20	RB
21	大江
22-30	459
31	rc

PowerPC (R)

divwu (divwu) *RT*、*RA*、*RB*

ディブ *RT*、*RA*、*RB*

divwuo *RT*、*RA*、*RB*

ディブウオ *RT*、*RA*、*RB*

説明

divwu 命令は、汎用レジスター (GPR) *RA* の内容を GPR *RB* の内容で除算し、その結果をターゲット GPR *RT* に保管します。 被除数、除数、および商は、符号なし整数として解釈されます。

0 による除算の場合、GPR *RT* の内容は未定義です。

> **注:** 演算は結果を符号なし整数として扱いますが、Rc が 1 の場合、条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、および「等しい (EQ) ゼロ・ビット」は、結果が符号付き整数として解釈されたかのように設定されます。

divwu 命令には 4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
divwu (divwu)	0	なし	0	なし
ディブ	0	なし	1	LT、GT、EQ、SO
divwuo	1	SO、OV、	0	なし
ディブウオ	1	SO、OV	1	LT、GT、EQ、SO

divwu 命令の 4 つの構文形式は、固定小数点例外レジスターのカリー・ビット (CA) には影響しません。 構文形式がオーバーフロー例外 (OE) ビットを 1 に設定する場合、命令は固定小数点例外レジスターの要約オーバーフロー (SO) およびオーバーフロー (OV) ビットに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。

項目	説明
rb	EA 計算用のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容を GPR 6 の内容で分割し、その結果を GPR 4 に保管します。

```
# Assume GPR 4 contains 0x0000 0000.
# Assume GPR 6 contains 0x0000 0002.
divwu 4,4,6
# GPR 4 now contains 0x0000 0000.
```

2. 以下のコードは、GPR 4 の内容を GPR 6 の内容で除算し、結果を GPR 4 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0x0000 0002.
# Assume GPR 6 contains 0x0000 0002.
divwu. 4,4,6
# GPR 4 now contains 0x0000 0001.
```

3. 以下のコードは、GPR 4 の内容を GPR 6 の内容で除算し、結果を GPR 4 に入れ、演算の結果を反映するために固定小数点例外レジスターに要約オーバーフロー・ビットとオーバーフロー・ビットを設定します。

```
# Assume GPR 4 contains 0x0000 0001.
# Assume GPR 6 contains 0x0000 0000.
divwuo 4,4,6
# GPR 4 now contains an undefined quantity.
```

4. 以下のコードは、GPR 4 の内容を GPR 6 の内容で除算し、結果を GPR 4 に入れ、演算の結果を反映するために、固定小数点例外レジスターおよび条件レジスター・フィールド 0 にサマリー・オーバーフロー・ビットとオーバーフロー・ビットを設定します。

```
# Assume GPR 4 contains 0x8000 0000.
# Assume GPR 6 contains 0x0000 0002.
divwuo. 4,4,6
# GPR 4 now contains 0x4000 0000.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

doz (Difference または Zero) 命令

目的

2 つの汎用レジスターの内容の差を計算し、その結果またはゼロの値を汎用レジスターに保管します。

注: doz 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0-5	31
6 ～ 10	RT

ビット	VALUE
11-15	RA
16-20	RB
21	大江
22-30	264
31	rc

POWER® ファミリー

doz *RT*、*RA*、*RB*

doz: *RT*、*RA*、*RB*

ドゾ (dozo) *RT*、*RA*、*RB*

ドゾ *RT*、*RA*、*RB*

説明

doz 命令は、汎用レジスター (GPR) *RA*、1 の内容、および GPR *RB* の内容の補数を追加し、結果をターゲット GPR *RT* に保管します。

GPR *RA* の値が GPR *RB* の値より代数的に大きい場合、GPR *RT* は 0 に設定されます。

doz 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
doz	0	なし	0	なし
doz:	0	なし	1	LT、GT、EQ、SO
ドゾ (dozo)	1	SO、OV	0	なし
ドゾ	1	SO、OV	1	LT、GT、EQ、SO

doz 命令の 4 つの構文形式は、固定小数点例外レジスターのカリー・ビット (CA) には影響しません。 構文フォームでオーバーフロー例外 (OE) ビットが 1 に設定されている場合、命令は固定小数点例外レジスターの要約オーバーフロー (SO) ビットとオーバーフロー (OV) ビットに影響します。オーバーフロー (OV) ビットは、正のオーバーフローにのみ設定できます。 構文形式がレコード (Rc) ビットを 1 に設定すると、条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および「要約オーバーフロー (SO)」ビットが命令の影響を受けます。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 と GPR 6 の内容の違いを判別し、結果を GPR 4 に保管します。

```
# Assume GPR 4 holds 0x0000 0001.
# Assume GPR 6 holds 0x0000 0002.
doz 4,4,6
# GPR 4 now holds 0x0000 0001.
```

2. 以下のコードは、GPR 4 と GPR 6 の内容の違いを判別し、結果を GPR 4 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 holds 0x0000 0001.
# Assume GPR 6 holds 0x0000 0000.
doz. 4,4,6
# GPR 4 now holds 0x0000 0000.
```

3. 以下のコードは、GPR 4 と GPR 6 の内容の違いを判別し、結果を GPR 4 に保管し、演算の結果を反映するために固定小数点例外レジスターにサマリー・オーバーフロー・ビットとオーバーフロー・ビットを設定します。

```
# Assume GPR 4 holds 0x0000 0002.
# Assume GPR 6 holds 0x0000 0008.
dozo 4,4,6
# GPR 4 now holds 0x0000 0006.
```

4. 以下のコードは、GPR 4 と GPR 6 の内容の違いを判別し、結果を GPR 4 に保管し、演算の結果を反映するために、要約オーバーフロー・ビットとオーバーフロー・ビットを固定小数点例外レジスターおよび条件レジスター・フィールド 0 に設定します。

```
# Assume GPR 4 holds 0xEFFF FFFF.
# Assume GPR 6 holds 0x0000 0000.
dozo. 4,4,6
# GPR 4 now holds 0x1000 0001.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

dozi (Difference または Zero Immediate) 命令

目的

汎用レジスターの内容と符号付き 16 ビット整数との差を計算し、その結果または値ゼロを汎用レジスターに保管します。

注: dozi 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0-5	09
6 ～ 10	RT
11-15	RA
16-31	SI

POWER® ファミリー

ドジ *RT*、*RA*、*SI*

説明

dozi 命令は、汎用レジスター (GPR) *RA*、16 ビット符号付き整数 *SI*、および 1 の内容の補数を追加し、結果をターゲット GPR *RT* に保管します。

GPR *RA* の値が *SI* フィールドの 16 ビット符号付きの値より代数的に大きい場合、GPR *RT* は 0 に設定されます。

dozi 命令には 1 つの構文形式があり、条件レジスター・フィールド 0 または固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。
SI	演算用の符号付き 16 ビット整数を指定します。

以下のコードは、GPR 4 と 0x0 の違いを判別し、結果を GPR 4 に保管します。

```
# Assume GPR 4 holds 0x0000 0001.
dozi 4,4,0x0
# GPR 4 now holds 0x0000 0000.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

eciwx (ワード索引付き外部制御) 命令

目的

有効アドレス (EA) を実アドレスに変換し、実アドレスをコントローラーに送信し、コントローラーによって戻されたワードをレジスターにロードします。

注 : eciwx 命令は、PowerPC® アーキテクチャーでのみ定義され、オプションの命令です。 PowerPC® 601 RISC Microprocessor、PowerPC 603 RISC Microprocessor、および PowerPC 604 RISC Microprocessor でサポートされています。

構文

ビット	値
0-5	31
6 ～ 10	RT
11-15	RA
16-20	RB
21-30	310

ビット	値
31	/

項目 **説明**

エシウクス (eciwx) *RT*、 *RA*、 *RB*

説明

eciwx 命令は EA を実アドレスに変換し、実アドレスをコントローラーに送信し、コントローラーによって戻されたワードを汎用レジスター *RT* に入れます。 *RA* = 0 の場合、EA は *RB* の内容です。それ以外の場合、EA は *RA* の内容と *RB* の内容の合計です。

EAR (E) = 1 の場合、EA に対応する実アドレスのロード要求は、キャッシュをバイパスして、EAR (RID) によって識別されるコントローラーに送信されます。 コントローラーによって戻されるワードは、*RT* に入れられます。

注 :

1. EA は 4 の倍数 (ワードで位置合わせされたアドレス) でなければなりません。そうでない場合、結果は未定義になります。
2. 操作は、保護に関するアドレス指定されたバイトへのロードとして扱われます。

パラメーター

項目 **説明**

RT 操作の結果が保管されるターゲット汎用レジスターを指定します。

RA 操作のソース汎用レジスターを指定します。

rb 操作のソース汎用レジスターを指定します。

関連概念

ecowx (External Control Out Word Indexed) 命令

処理とストレージ

プロセッサーは、メイン・メモリーおよびレジスターにデータを保管します。

ecowx (External Control Out Word Indexed) 命令

目的

有効アドレス (EA) を実アドレスに変換し、実アドレスとレジスターの内容を制御装置に送ります。

> **注: ecowx** 命令は、PowerPC ® アーキテクチャーでのみ定義されており、オプションの命令です。PowerPC ® 601 RISC Microprocessor、PowerPC 603 RISC Microprocessor、および PowerPC 604 RISC Microprocessor でサポートされています。

構文

ビット	VALUE
0-5	31
6 〜 10	RS
11-15	RA
16-20	RB

ビット	VALUE
21-30	438
31	/

項目	説明
エコー (ecowx)	*RS*、 *RA*、 *RB*

説明

ecowx 命令は、EA を実アドレスに変換し、実アドレスと汎用レジスター *RS* の内容をコントローラーに送信します。 *RA* = 0 の場合、EA は *RB* の内容です。それ以外の場合、EA は *RA* の内容と *RB* の内容の合計です。

EAR (E) = 1 の場合、EA に対応する実アドレスの保管要求は、キャッシュをバイパスして、EAR (RID) によって識別されるコントローラーに送信されます。 *RS* の内容は、保管要求とともに送信されます。

注:

1. EA は 4 の倍数 (ワードで位置合わせされたアドレス) でなければなりません。そうでない場合、結果は未定義になります。
2. 操作は、保護に関してアドレス指定されたバイトに対する保管として扱われます。

パラメーター

項目	説明
RS	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

関連概念
eciwx (ワード索引付き外部制御) 命令
処理とストレージ
プロセッサーは、メイン・メモリーおよびレジスターにデータを保管します。

eieio (入出力の順次実行の強制) 命令

目的

キャッシュ禁止ストレージ・アクセスが、プログラムによって指定された順序でメイン・メモリー内で実行されるようにします。

注: eio 命令は、PowerPC ® アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0-5	31
6 ～ 10	///
11-15	///
16-20	///
21-30	854

ビット	VALUE
31	/

PowerPC (R)

Eeio (Eeio)

説明

eieio 命令は、**eieio** 命令の前に開始されたすべてのロード命令および保管命令が、**eieio** 命令のアクセス・メモリーの後にロードまたは保管される前にメイン・メモリー内で完了することを保証する順序付け機能を提供します。 **eieio** 命令がプログラムから省略され、メモリーの位置が固有の場合、主記憶域へのアクセスは任意の順序で実行できます。

> **注: eieio** 命令は、入出力装置によって認識されるストレージ参照の順序を制御することが唯一の要件である場合に適しています。 ただし、 **sync** (同期化) 命令は、すべての命令に対して順序付け機能を提供します。

eieio 命令には 1 つの構文形式があり、条件レジスター・フィールド 0 または固定小数点例外レジスターには影響しません。

例

以下のコードにより、メモリー・ロケーションがキャッシュ禁止ストレージ内にある場合、ロケーション CC のコンテンツがフェッチされる前、またはロケーション DD のコンテンツが更新される前に、ロケーション AA およびストアからロケーション BB へのロードが主ストレージ内で確実に実行されます。

```
lwz   r4,AA(r1)
stw   r4,BB(r1)
eieio
lwz   r5,CC(r1)
stw   r5,DD(r1)
```

> **注:** AA、 BB、 CC、および DD のメモリー位置がキャッシュ禁止メモリー内にない場合、 **eieio** 命令は、命令がメモリーにアクセスする順序に影響を与えません。

関連概念

sync (同期化) または dcs (Data Cache 同期化) 命令

処理とストレージ

プロセッサーは、メイン・メモリーおよびレジスターにデータを保管します。

extsw (拡張符号ワード) 命令

目的

汎用レジスターの下位 32 ビットを別の汎用レジスターにコピーし、そのフルワードをサイズ (64 ビット) のダブルワードに拡張する符号を付けます。

構文

ビット	VALUE
0-5	31
6 〜 10	S
11-15	A
16-20	00000
21-30	986
31	rc

PowerPC (R)

extsw	*RA*、 *RS* (Rc=0)
extsw。	*RA*、 *RS*(Rc=1)

説明

汎用レジスター (GPR) の下位 32 ビット (*RS*) の内容は、GPR *RA* の下位 32 ビットに入れられます。 GPR *RS* のビット 32 は、GPR *RA* の上位 32 ビットを埋めるために使用されます。

変更されたその他のレジスター:

- 条件レジスター (CR0 フィールド):

  影響を受ける: LT、 GT、EQ、SO (Rc = 1 の場合)
- XER:

  影響を受ける: CA

パラメーター

項目	説明
RA	操作の結果のターゲット汎用レジスターを指定します。
RS	命令のオペランドのソース汎用レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

eqv (同等) 命令

目的

2 つの汎用レジスターの内容を論理的に XOR し、補完された結果を汎用レジスターに入れます。

構文

ビット	VALUE
0-5	31
6 ～ 10	RS
11-15	RA
16-20	RB
21-30	284
31	rc

項目	説明
eqv	*RA*、 *RS*、 *RB*
eqv:	*RA*、 *RS*、 *RB*

説明

eqv 命令は、汎用レジスター (GPR) *RS* の内容を GPR *RB* の内容と論理的に XOR し、補完された結果をターゲット GPR *RA* に保管します。

eqv 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
eqv	なし	なし	0	なし
eqv:	なし	なし	1	LT、GT、EQ、SO

eqv 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 および GPR 6 の内容を論理的に XOR し、補完された結果を GPR 4 に保管します。

```
# Assume GPR 4 holds 0xFFF2 5730.
# Assume GPR 6 holds 0x7B41 92C0.
eqv 4,4,6
# GPR 4 now holds 0x7B4C 3A0F.
```

2. 以下のコードは、GPR 4 および GPR 6 の内容を XOR し、補完された結果を GPR 4 に保管し、演算の結果を反映するために条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 holds 0x0000 00FD.
# Assume GPR 6  holds 0x7B41 92C0.
eqv. 4,4,6
# GPR 4 now holds 0x84BE 6DC2.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点論理命令
固定小数点論理命令は、論理演算をビット単位で実行します。

extsb (拡張符号バイト) 命令

目的

下位バイトの符号を拡張します。

注: extsb 命令は、PowerPC ® アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0-5	31
6 〜 10	RS
11-15	RA
16-20	///
21-30	954
31	rc

PowerPC (R)

extsb *RA*、 *RS*

extsb。 *RA*、 *RS*

説明

extsb 命令は、汎用レジスター (GPR) *RS* のビット 24 から 31 を GPR *RA* のビット 24 から 31 に入れ、レジスター *RA* のビット 0 から 23 にレジスター *RS* のビット 24 をコピーします。

extsb 命令には、2 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	拡張されるバイトを含むソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 に含まれる最下位バイトの符号を拡張し、結果を GPR 6 に入れます。

```
# Assume GPR 6 holds 0x5A5A 5A5A.
extsb 4,6
# GPR 6 now holds 0x0000 005A.
```

2. 以下のコードは、GPR 4 に含まれる最下位バイトの符号を拡張し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 holds 0xA5A5 A5A5.
extsb. 4,4
# GPR 4 now holds 0xFFFF FFA5.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点論理命令
固定小数点論理命令は、論理演算をビット単位で実行します。

extsh 命令または exts (拡張符号ハーフワード) 命令

目的

汎用レジスターの下位 16 ビットの内容を拡張します。

構文

ビット	VALUE
0-5	31
6 〜 10	RS
11-15	RA
16-20	///
21	大江
22-30	922
31	rc

PowerPC (R)

EXT *RA*、 *RS*

(extsh)。 *RA*、 *RS*

POWER® ファミリー

EXTS *RA*、 *RS*

exts。 *RA*、 *RS*

説明

extsh および **exts** 命令は、汎用レジスター (GPR) *RS* のビット 16-31 を GPR *RA* のビット 16-31 に入れ、GPR *RA* のビット 0-15 に GPR *RS* のビット 16 をコピーします。

extsh および **exts** 命令には、それぞれ 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	オーバーフロー 例外 (OE)	固定小数点 例外レジスター	レコード ビット (RC)	条件 フィールド 0 の登録
EXT	なし	なし	0	なし
(extsh)。	なし	なし	1	LT、GT、EQ、SO
EXTS	なし	なし	0	なし
exts。	なし	なし	1	LT、GT、EQ、SO

extsh 命令の 2 つの構文形式、および **extsh** 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	汎用レジスターが拡張整数を受け取ることを指定します。

項目	説明
RS	操作のソース汎用レジスターを指定します。

例

1. 次のコードは、GPR 6 のビット 16 から 31 を GPR 4 のビット 16 から 31 に入れ、GPR 6 のビット 16 を GPR 4 のビット 0 から 15 にコピーします。

```
# Assume GPR 6 holds 0x0000 FFFF.
extsh 4,6
# GPR 6 now holds 0xFFFF FFFF.
```

2. 次のコードは、GPR 6 のビット 16 から 31 を GPR 4 のビット 16 から 31 に入れ、GPR 6 のビット 16 を GPR 4 のビット 0 から 15 にコピーし、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 holds 0x0000 2FFF.
extsh. 6,4
# GPR 6 now holds 0x0000 2FFF.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点論理命令
固定小数点論理命令は、論理演算をビット単位で実行します。

fabs (Floating Absolute Value) 命令

目的

浮動小数点レジスターの内容の絶対値を別の浮動小数点レジスターに保管します。

構文

ビット	VALUE
0-5	63
6 〜 10	FRT (R)
11-15	///
16-20	FRB
21-30	264
31	rc

項目	説明
fabs	*FRT*、 *FRB*
ファブズ	*FRT*、 *FRB*

説明

fabs 命令は、浮動小数点レジスター (FPR) *FRB* のビット 0 を 0 に設定し、結果を FPR *FRT* に入れます。

fabs 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
fabs	なし	0	なし
ファブズ	なし	1	FX、FEX、VX、OX

fabs 命令の 2 つの構文形式は、浮動小数点状況および制御レジスターには影響を与えません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は、条件レジスター・フィールド 1 の浮動小数点例外要約 (FX)、浮動小数点使用可能例外要約 (FEX)、浮動小数点無効演算例外要約 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

パラメーター

項目	説明
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRB	操作のソース浮動小数点レジスターを指定します。

例

1. 以下のコードは、FPR 4 のビット 0 をゼロに設定し、結果を FPR 6 に入れます。

```
# Assume FPR 4 holds 0xC053 4000 0000 0000.
fabs 6,4
# GPR 6 now holds 0x4053 4000 0000 0000.
```

2. 以下のコードは、FPR 25 のビット 0 をゼロに設定し、結果を FPR 6 に入れ、演算の結果を反映するように条件レジスター・フィールド 1 を設定します。

```
# Assume FPR 25 holds 0xFFFF FFFF FFFF FFFF.
fabs. 6,25
# GPR 6 now holds 0x7FFF FFFF FFFF FFFF.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点移動命令
浮動小数点移動命令は、ある FPR から別の FPR にデータをコピーします。

浮動小数点レジスターの内容の解釈
32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

fadd または fa (Floating Add) 命令

目的

2 つの浮動小数点オペランドを追加し、結果を浮動小数点レジスターに入れます。

構文

ビット	VALUE
0-5	63
6 〜 10	FRT (R)
11-15	FRA
16-20	FRB
21-25	///
26-30	21
31	rc

PowerPC (R)

追加 *FRT*、 *FRA*、 *FRB*

fadd。 *FRT*、 *FRA*、 *FRB*

POWER® ファミリー

FA *FRT*、 *FRA*、 *FRB*

fa。 *FRT*、 *FRA*、 *FRB*

ビット	値
0-5	59
6 〜 10	FRT (R)
11-15	FRA
16-20	FRB
21-25	///
26-30	21
31	rc

PowerPC (R)

fadd *FRT*、 *FRA*、 *FRB*

追加します。 *FRT*、 *FRA*、 *FRB*

説明

fadd および **fa** 命令は、浮動小数点レジスター (FPR) *FRA* の 64 ビット倍精度浮動小数点オペランドを FPR *FRB* の 64 ビット倍精度浮動小数点オペランドに追加します。

fadd 命令は、FPR *FRA* の 32 ビット単精度浮動小数点オペランドを FPR *FRB* の 32 ビット単精度浮動小数点オペランドに追加します。

結果は、浮動小数点状況および制御レジスターの浮動小数点丸め制御フィールド *RN* の制御の下で丸められ、FPR *FRT* に入れられます。

2 つの浮動小数点数の加算は、指数比較と 2 つの仮数の加算に基づいています。 2 つのオペランドの指数が比較され、小さい方の指数に付随する仮数が右にシフトされます。その指数は、2 つの指数が等しくなるまで、ビットごとに 1 ずつシフトされます。 次に、2 つの仮数が代数的に加算されて、中間合計が形成されます。 3 つのガード・ビット (G、R、および X) だけでなく、仮数内の 53 ビットすべてが計算に入ります。

浮動小数点状況および制御レジスターの浮動小数点結果フィールドは、浮動小数点状況および制御レジスターの浮動小数点無効演算例外使用可能 (VE) ビットが 1 に設定されている場合の無効演算例外を除き、結果のクラスおよび符号に設定されます。

fadd、 **fadd**、および **fa** 命令には、それぞれ 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	**説明**		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
追加	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI	0	なし
fadd。	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI	1	FX、FEX、VX、OX
fadd	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI	0	なし
追加します。	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI	1	FX、FEX、VX、OX
FA	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI	0	なし
fa。	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI	1	FX、FEX、VX、OX

fadd、 **fadd**、および **fa** 命令のすべての構文形式は、常に浮動小数点状況および制御レジスターに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は、条件レジスター・フィールド 1 の浮動小数点例外要約 (FX)、浮動小数点使用可能例外要約 (FEX)、浮動小数点無効演算例外要約 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

パラメーター

項目	**説明**
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRA	操作のソース浮動小数点レジスターを指定します。
FRB	操作のソース浮動小数点レジスターを指定します。

例

1. 以下のコードは、FPR 4 と FPR 5 の内容を追加し、結果を FPR 6 に入れ、演算の結果を反映するように浮動小数点状況および制御レジスターを設定します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
# Assume FPR 5 contains 0x400C 0000 0000 0000.
fadd 6,4,5
# FPR 6 now contains 0xC052 6000 0000 0000.
```

2. 以下のコードは、FPR 4 と FPR 25 の内容を追加し、結果を FPR 6 に入れ、条件レジスター・フィールド 1 と浮動小数点状況および制御レジスターを設定して、操作の結果を反映します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
```

```
# Assume FPR 25 contains 0xFFFF FFFF FFFF FFFF.
fadd. 6,4,25
# GPR 6 now contains 0xFFFF FFFF FFFF FFFF.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点演算命令
浮動小数点演算命令は、浮動小数点レジスターに含まれる浮動小数点データに対して算術演算を実行します。

浮動小数点レジスターの内容の解釈
32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

fcfid (Integer Double Word からの浮動変換) 命令

目的

浮動小数点レジスターの固定小数点の内容を倍精度浮動小数点数に変換します。

構文

ビット	VALUE
0-5	63
6 〜 10	D
11-15	00000
16-20	B
21-30	846
31	rc

PowerPC (R)

FCD *FRT*、 *FRB* (Rc=0)

fcfid。 *FRT*、 *FRB* (Rc=1)

説明

浮動小数点レジスター (FPR) *FRB* の 64 ビット符号付き固定小数点オペランドは、無限に正確な浮動小数点整数に変換されます。 変換の結果は、FPSCR [RN] で指定された丸めモードを使用して倍精度に丸められ、FPR *FRT* に入れられます。

FPSCR [FPRF] は、結果のクラスと符号に設定されます。 FPSCR [FR] は、丸め時に結果が増分される場合に設定されます。 結果が不正確な場合は、FPSCR [FI] が設定されます。

fcfid 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
FCD	FPRF、FR、FI、FX、XX	0	なし
fcfid。	FPRF、FR、FI、FX、XX	1	FX、FEX、VX、OX

パラメーター

項目	説明
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRB	操作のソース浮動小数点レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

fcmpo (浮動比較の順序) 命令

目的

2 つの浮動小数点レジスターの内容を比較します。

構文

ビット	VALUE
0-5	63
6-8	BF
9 から 10	//
11-15	FRA
16-20	FRB
21-30	32
31	/

項目	説明
fcmpo (fcmpo)	*BF*、 *FRA*、 *FRB*

説明

fcmpo 命令は、浮動小数点レジスター (FPR) *FRA* の 64 ビット倍精度浮動小数点オペランドを、FPR *FRB* の 64 ビット倍精度浮動小数点オペランドと比較します。 浮動小数点状況および制御レジスター (FPSCR) の浮動小数点条件コード・フィールド (FPCC) は、オペランド FPR *FRB* に関するオペランド FPR *FRA* の値を反映するように設定されます。 値 *BF* は、条件レジスター内のどのフィールドが 4 つの FPCC ビットを受け取るかを決定します。

fcmpo 命令を使用する場合は、以下の点を考慮してください。

- オペランドの 1 つが Quiet NaN (QNaN) または Signaling NaN (SNaN) のいずれかである場合、浮動小数点条件コードは unordered (FU) を反映するように設定されます。
- オペランドの 1 つが SNaN の場合は、浮動小数点状況および制御レジスターの浮動小数点無効演算例外ビット VXSNAN が設定されます。 また、以下のとおりです。
  - 無効な操作が使用不可 (つまり、浮動小数点状況および制御レジスターの浮動小数点無効演算例外使用可能ビットが 0) の場合、浮動小数点無効演算例外ビット VXVC が設定されます (無効な比較を通知します)。

– オペランドの 1 つが QNaN の場合、浮動小数点無効演算例外ビット VXVC が設定されます。

fcmpo 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターの FT、FG、FE、FU、VXSNAN、および VXVC ビットに常に影響します。

パラメーター

項目	説明
BF	4 つの FPCC ビットを受け取る条件レジスター内のフィールドを指定します。
FRA	ソース浮動小数点レジスターを指定します。
FRB	ソース浮動小数点レジスターを指定します。

例

以下のコードは、FPR 4 と FPR 6 の内容を比較し、条件レジスター・フィールド 1 と浮動小数点状況および制御レジスターを設定して、操作の結果を反映します。

```
# Assume CR = 0 and FPSCR = 0.
# Assume FPR 5 contains 0xC053 4000 0000 0000.
# Assume FPR 4 contains 0x400C 0000 0000 0000.
fcmpo 6,4,5
# CR now contains 0x0000 0040.
# FPSCR now contains 0x0000 4000.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点比較命令
浮動小数点比較命令は、2 つの FPR の内容の順序付き比較および順序なし比較を実行します。

fcmpu (非順序浮動比較) 命令

目的

2 つの浮動小数点レジスターの内容を比較します。

構文

ビット	VALUE
0-5	63
6-8	BF
9 から 10	//
11-15	FRA
16-20	FRB
21-30	0
31	/

項目	説明
fcmpu (fcmpu)	*BF*、 *FRA*、 *FRB*

説明

fcmpu 命令は、浮動小数点レジスター (FPR) *FRA* の 64 ビット倍精度浮動小数点オペランドを、FPR *FRB* の 64 ビット倍精度浮動小数点オペランドと比較します。 浮動小数点状況および制御レジスター (FPSCR) の浮動小数点条件コード・フィールド (FPCC) は、オペランド *FRB* に関するオペランド *FRA* の値を反映するように設定されます。 値 *BF* は、条件レジスター内のどのフィールドが 4 つの FPCC ビットを受け取るかを決定します。

fcmpu 命令を使用する場合は、以下の点を考慮してください。

- オペランドの 1 つが Quiet NaN または Signaling NaN のいずれかである場合、浮動小数点条件コードは unordered (FU) を反映するように設定されます。
- オペランドの 1 つがシグナリング NaN の場合、浮動小数点状況および制御レジスターの浮動小数点無効演算例外ビット VXSNAN が設定されます。

fcmpu 命令には 1 つの構文形式があり、FPSCR 内の FT、FG、FE、FU、および VXSNAN ビットに常に影響します。

パラメーター

項目	説明
BF	4 つの FPCC ビットを受け取る条件レジスター内のフィールドを指定します。
FRA	ソース浮動小数点レジスターを指定します。
FRB	ソース浮動小数点レジスターを指定します。

例

以下のコードは、FPR 5 と FPR 4 の内容を比較します。

```
# Assume FPR 5 holds 0xC053 4000 0000 0000.
# Assume FPR 4 holds 0x400C 0000 0000 0000.
# Assume CR = 0 and FPSCR = 0.
fcmpu 6,4,5
# CR now contains 0x0000 0040.
# FPSCR now contains 0x0000 4000.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点比較命令
浮動小数点比較命令は、2 つの FPR の内容の順序付き比較および順序なし比較を実行します。

fctid (Integer Double Word への浮動小数点変換) 命令

目的

浮動小数点レジスターの内容を 64 ビット符号付き固定小数点整数に変換し、結果を別の浮動小数点レジスターに入れます。

構文

ビット	VALUE
0-5	63
6 ～ 10	D
11-15	00000
16-20	B
21-30	814
31	rc

PowerPC (R)

fctid (fctid) *FRT*、 *FRB* (Rc=0)

Fctid。 *FRT*、 *FRB* (Rc=1)

説明

浮動小数点レジスター (FPR) *FRB* の浮動小数点オペランドは、FPSCR [RN] で指定された丸めモードを使用して 64 ビット符号付き固定小数点整数に変換され、FPR *FRT* に入れられます。

FRB のオペランドが 2 * *63-1 より大きい場合、FPR *FRT* は 0x7FFF_FFFF_FFFF_FFFF に設定されます。
FRB のオペランドが 2 * *63 より小さい場合、FPR *FRT* は 0x8000_0000_0000_0000 に設定されます。

有効な無効演算例外を除き、FPSCR [FPRF] は未定義です。 FPSCR [FR] は、丸め時に結果が増分される場合に設定されます。 結果が不正確な場合は、FPSCR [FI] が設定されます。

fctid 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
fctid (fctid)	FPRF (未定義)、FR、FI、FX、XX、VXSNAN、VXCVI	0	なし
Fctid。	FPRF (未定義)、FR、FI、FX、XX、VXSNAN、VXCVI	1	FX、FEX、VX、OX

パラメーター

項目	説明
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRB	操作のソース浮動小数点レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

fctidz (ゼロ方向へのラウンドによる整数ダブルワードへの浮動小数点変換) 命令

目的

ゼロ方向丸めモードを使用して、浮動小数点レジスターの内容を 64 ビット符号付き固定小数点整数に変換します。 結果を別の浮動小数点レジスターに入れます。

構文

ビット	VALUE
0-5	63
6 ～ 10	D
11-15	00000
16-20	B
21-30	815
31	rc

PowerPC (R)

fctidz (fctidz) *FRT*、 *FRB* (Rc=0)

fctidz。 *FRT*、 *FRB* (Rc=1)

説明

浮動小数点レジスター (FRP) *FRB* の浮動小数点オペランドは、丸めモードをゼロの方向に丸めて 64 ビット符号付き固定小数点整数に変換され、FPR *FRT* に入れられます。

FPR *FRB* のオペランドが 2 * *63-1 より大きい場合、FPR *FRT* は 0x7FFF_FFFF_FFFF_FFFF に設定されます。 frB のオペランドが 2 * *63 より小さい場合、FPR *FRT* は 0x8000_0000_0000_0000 に設定されます。

有効な無効演算例外を除き、FPSCR [FPRF] は未定義です。 FPSCR [FR] は、丸め時に結果が増分される場合に設定されます。 結果が不正確な場合は、FPSCR [FI] が設定されます。

fctidz 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
fctidz (fctidz)	FPRF (未定義)、FR、FI、FX、XX、VXSNAN、VXCVI	0	なし
fctidz。	FPRF (未定義)、FR、FI、FX、XX、VXSNAN、VXCVI	1	FX、FEX、VX、OX

パラメーター

項目	説明
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRB	操作のソース浮動小数点レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

fctiw または fcir (整数ワードへの浮動変換) 命令

目的

浮動小数点オペランドを 32 ビットの符号付き整数に変換します。

構文

ビット	VALUE
0-5	63
6 ～ 10	FRT (R)
11-15	///
16-20	FRB
21-30	14
31	rc

PowerPC (R)

fctiw (fctiw) *FRT*、 *FRB*

Fctiw。 *FRT*、 *FRB*

POWER2™

fcir *FRT*、 *FRB*

FCIR *FRT*、 *FRB*

説明

fctiw および **fcir** 命令は、浮動小数点状況制御レジスター (FPSCR) によって指定された丸めモードを使用して、浮動小数点レジスター (FPR) *FRB* の浮動小数点オペランドを 32 ビット符号付き固定小数点整数に変換します。 RN。 結果は、FPR *FRT* のビット 32 から 63 に入れられます。 FPR *FRT* のビット 0 から 31 は未定義です。

FPR *FRB* のオペランドが 231-1 より大きい場合、FPR *FRT* のビット 32-63 は 0x7FFF FFFF に設定されます。 FPR *FRB* のオペランドが -231 より小さい場合、FPR *FRT* のビット 32 から 63 は 0x8000 0000 に設定されます。

fctiw および **fcir** 命令には、それぞれ 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
fctiw (fctiw)	C、FL、FG、FE、FU、FR、FI、FX、XX、VXCVI、VXSNAN	0	なし
Fctiw。	C、FL、FG、FE、FU、FR、FI、FX、XX、VXCVI、VXSNAN	1	FX、FEX、VX、OX
fcir	C、FL、FG、FE、FU、FR、FI、FX、XX、VXCVI、VXSNAN	0	なし

項目	説明		
FCIR	C、FL、FG、FE、FU、FR、FI、FX、XX、VXCVI、VXSNAN	1	FX、FEX、VX、OX

fctiw 命令および **fcir** 命令の構文形式は、常に FPSCR に影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 1 の浮動小数点例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。 FPSCR (C、FI、FG、FE、FU) は未定義です。

パラメーター

項目	説明
FRT (FRT)	整数の結果が入れられる浮動小数点レジスターを指定します。
FRB	浮動小数点オペランドのソース浮動小数点レジスターを指定します。

例

以下のコードは、浮動小数点値の配列で索引として使用するために、浮動小数点値を整数に変換します。

```
# Assume GPR 4 contains the address of the first element of
# the array.
# Assume GPR 1 contains the stack pointer.
# Assume a doubleword TEMP variable is allocated on the stack
# for use by the conversion routine.
# Assume FPR 6 contains the floating-point value for conversion
# into an index.
fctiw   5,6                     # Convert floating-point value
                                # to integer.
stfd    5,TEMP(1)               # Store to temp location.
lwz     3,TEMP+4(1)             # Get the integer part of the
                                # doubleword.
lfd     5,0(3)                  # Get the selected array element.
# FPR 5 now contains the selected array element.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点演算命令
浮動小数点演算命令は、浮動小数点レジスターに含まれる浮動小数点データに対して算術演算を実行します。

浮動小数点レジスターの内容の解釈
32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

fctiwz または fcirz (Round to Zero による整数ワードへの浮動変換) 命令

目的

浮動小数点オペランドを 32 ビットの符号付き整数に変換し、結果を 0 に丸めます。

構文

ビット	VALUE
0-5	63
6 ～ 10	FRT (R)
11-15	///
16-20	FRB
21-30	15
31	rc

PowerPC (R)

fctiwz (fctiwz) *FRT*、 *FRB*

fctiwz。 *FRT*、 *FRB*

POWER2™

fcirz (fcirz) *FRT*、 *FRB*

FCirz。 *FRT*、 *FRB*

説明

fctiwz および **fcirz** 命令は、浮動小数点レジスター (FPR) *FRB* の浮動小数点オペランドを 32 ビットの符号付き固定小数点整数に変換し、オペランドを 0 の方向に丸めます。 結果は、FPR *FRT* のビット 32 から 63 に入れられます。 FPR *FRT* のビット 0 から 31 は未定義です。

FPR *FRB* のオペランドが 231-1 より大きい場合、FPR *FRT* のビット 32-63 は 0x7FFF FFFF に設定されます。 FPR *FRB* のオペランドが -231 より小さい場合、FPR *FRT* のビット 32 から 63 は 0x8000 0000 に設定されます。

fctiwz 命令と **fcirz** 命令には、それぞれ 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
fctiwz (fctiwz)	C、FL、FG、FE、FU、FR、FI、FX、XX、VXCVI、VXSNAN	0	なし
fctiwz。	C、FL、FG、FE、FU、FR、FI、FX、XX、VXCVI、VXSNAN	1	FX、FEX、VX、OX
fcirz (fcirz)	C、FL、FG、FE、FU、FR、FI、FX、XX、VXCVI、VXSNAN	0	なし
FCirz。	C、FL、FG、FE、FU、FR、FI、FX、XX、VXCVI、VXSNAN	1	FX、FEX、VX、OX

fctiwz および **fcirz** 命令の構文形式は、常に浮動小数点状況および制御レジスターに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 1 の浮動小数点例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。 FPSCR (C、FI、FG、FE、FU) は未定義です。

パラメーター

項目	説明
FRT (FRT)	整数の結果が入れられる浮動小数点レジスターを指定します。
FRB	浮動小数点オペランドのソース浮動小数点レジスターを指定します。

例

以下のコードは、2 番目の浮動小数点値に基づいて選択された配列エレメントに浮動小数点値を追加します。 value2 が n 以上、 n+1 未満の場合は、配列の n 番目の要素に value1 を追加します。

```
# Assume GPR 4 contains the address of the first element of
# the array.
# Assume GPR 1 contains the stack pointer.
# Assume a doubleword TEMP variable is allocated on the stack
# for use by the conversion routine.
# Assume FPR 6 contains value2.
# Assume FPR 4 contains value1.
fctiwz  5,6                     # Convert value2 to integer.
stfd    5,TEMP(1)               # Store to temp location.
lwz     3,TEMP+4(1)             # Get the integer part of the
                                # doubleword.
lfdx    5,3,4                   # Get the selected array element.
fadd    5,5,4                   # Add value1 to array element.
stfd    5,3,4                   # Save the new value of the
                                # array element.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点演算命令
浮動小数点演算命令は、浮動小数点レジスターに含まれる浮動小数点データに対して算術演算を実行します。

浮動小数点レジスターの内容の解釈
32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

fdiv または fd (浮動小数点除算) 命令

目的

浮動小数点オペランドを別のオペランドで除算します。

構文

ビット	VALUE
0-5	63
6 〜 10	FRT (R)
11-15	FRA
16-20	FRB
21-25	///
26-30	18

ビット	VALUE
31	rc

PowerPC (R)

fdiv (fdiv) *FRT*、 *FRA*、 *FRB*

FDIV *FRT*、 *FRA*、 *FRB*

POWER® ファミリー

fd *FRT*、 *FRA*、 *FRB*

FD. *FRT*、 *FRA*、 *FRB*

ビット	値
0-5	59
6 〜 10	FRT (R)
11-15	FRA
16-20	FRB
21-25	///
26-30	18
31	rc

PowerPC (R)

fdivs *FRT*、 *FRA*、 *FRB*

Fdiv。 *FRT*、 *FRA*、 *FRB*

説明

fdiv および **fd** 命令は、浮動小数点レジスター (FPR) *FRA* 内の 64 ビットの倍精度浮動小数点オペランドを、FPR *FRB* 内の 64 ビットの倍精度浮動小数点オペランドで除算します。 残りは保持されません。

fdivs 命令は、FPR *FRA* の 32 ビット単精度浮動小数点オペランドを FPR *FRB* の 32 ビット単精度浮動小数点オペランドで除算します。 残りは保持されません。

結果は、浮動小数点状況および制御レジスター (FPSCR) の浮動小数点丸め制御フィールド *RN* の制御の下に丸められ、ターゲット FPR *FRT* に入れられます。

浮動小数点除算演算は、2 つの仮数の指数減算と除算に基づいています。

> **注:** オペランドが非正規化された数値の場合、そのオペランドは、操作が開始される前に非正規化されます。

浮動小数点状況および制御レジスターの浮動小数点結果フラグ・フィールドは、浮動小数点無効演算例外使用可能ビットが 1 の場合に、無効な演算例外を除き、結果のクラスおよび符号に設定されます。

fdiv、 **fdivs**、および **fd** 命令には、それぞれ 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**

項目	説明		
fdiv (fdiv)	C、FL、FG、FE、FU、FR、FI、OX、UX、ZX、XX、VXSNAN、VXIDI、VXZDZ	0	なし
FDIV	C、FL、FG、FE、FU、FR、FI、OX、UX、ZX、XX、VXSNAN、VXIDI、VXZDZ	1	FX、FEX、VX、OX
fdivs	C、FL、FG、FE、FU、FR、FI、OX、UX、ZX、XX、VXSNAN、VXIDI、VXZDZ	0	なし
Fdiv。	C、FL、FG、FE、FU、FR、FI、OX、UX、ZX、XX、VXSNAN、VXIDI、VXZDZ	1	FX、FEX、VX、OX
fd	C、FL、FG、FE、FU、FR、FI、OX、UX、ZX、XX、VXSNAN、VXIDI、VXZDZ	0	なし
FD.	C、FL、FG、FE、FU、FR、FI、OX、UX、ZX、XX、VXSNAN、VXIDI、VXZDZ	1	FX、FEX、VX、OX

fdiv、**fdivs**、および **fd** 命令のすべての構文形式は、常に浮動小数点状況および制御レジスターに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 1 の浮動小数点例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

パラメーター

項目	説明
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRA	被除数を含むソース浮動小数点レジスターを指定します。
FRB	除数を含むソース浮動小数点レジスターを指定します。

例

1. 以下のコードは、FPR 4 の内容を FPR 5 の内容で分割し、結果を FPR 6 に入れ、演算の結果を反映するように浮動小数点状況および制御レジスターを設定します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
# Assume FPR 5 contains 0x400C 0000 0000 0000.
# Assume FPSCR = 0.
fdiv 6,4,5
# FPR 6 now contains 0xC036 0000 0000 0000.
# FPSCR now contains 0x0000 8000.
```

2. 次のコードは、FPR 4 の内容を FPR 5 の内容で分割し、結果を FPR 6 に入れ、条件レジスター・フィールド 1 と浮動小数点状況および制御レジスターを設定して、操作の結果を反映します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
# Assume FPR 5 contains 0x400C 0000 0000 0000.
# Assume FPSCR = 0.
fdiv. 6,4,5
# FPR 6 now contains 0xC036 0000 0000 0000.
# FPSCR now contains 0x0000 8000.
# CR contains 0x0000 0000.
```

関連概念

浮動小数点プロセッサー

浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点演算命令
浮動小数点演算命令は、浮動小数点レジスターに含まれる浮動小数点データに対して算術演算を実行します。

浮動小数点レジスターの内容の解釈
32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

fmadd または fma (Floating Multiply-Add) 命令

目的

中間丸め演算を行わずに 2 つの浮動小数点オペランドを乗算した結果に、1 つの浮動小数点オペランドを追加します。

構文

ビット	VALUE
0-5	63
6 〜 10	FRT (R)
11-15	FRA
16-20	FRB
21-25	FRC
26-30	29
31	rc

PowerPC (R)

FMADD *FRT*、 *FRA*、 *FRC*、 *FRB*

fmadd。 *FRT*、 *FRA*、 *FRC*、 *FRB*

POWER® ファミリー

FMA *FRT*、 *FRA*、 *FRC*、 *FRB*

fma: *FRT*、 *FRA*、 *FRC*、 *FRB*

ビット	値
0-5	59
6 〜 10	FRT (R)
11-15	FRA
16-20	FRB
21-25	FRC
26-30	29
31	rc

PowerPC (R)

fmadd *FRT*、 *FRA*、 *FRC*、 *FRB*

fmadd。 *FRT*、 *FRA*、 *FRC*、 *FRB*

説明

fmadd および **fma** 命令は、浮動小数点レジスター (FPR) *FRA* の 64 ビット倍精度浮動小数点オペランドに、FPR *FRC* の 64 ビット倍精度浮動小数点オペランドを乗算してから、この演算の結果を FPR *FRB* の 64 ビット倍精度浮動小数点オペランドに加算します。

fmadd 命令は、FPR *FRA* の 32 ビットの単精度浮動小数点オペランドに FPR *FRC* の 32 ビットの単精度浮動小数点オペランドを乗算し、この演算の結果を FPR *FRB* の 32 ビットの単精度浮動小数点オペランドに加算します。

結果は、浮動小数点の状況および制御レジスターの浮動小数点丸め制御フィールド *RN* の制御下で丸められ、ターゲット FPR *FRT* に入れられます。

> **注:** オペランドが非正規化された数値の場合、そのオペランドは、操作が開始される前に非正規化されます。

浮動小数点状況および制御レジスターの浮動小数点結果フラグ・フィールドは、浮動小数点無効演算例外使用可能ビットが 1 の場合に、無効な演算例外を除き、結果のクラスおよび符号に設定されます。

fmadd、 **fmadd**、および **fm** 命令には、それぞれ 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	**説明**		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
FMADD	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI、VXIMZ	0	なし
fmadd。	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI、VXIMZ	1	FX、FEX、VX、OX
fmadd	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI、VXIMZ	0	なし
fmadd。	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI、VXIMZ	1	FX、FEX、VX、OX
FMA	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI、VXIMZ	0	なし
fma:	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI、VXIMZ	1	FX、FEX、VX、OX

fmadd、 **fmadd**、および **fm** 命令のすべての構文形式は、常に浮動小数点状況および制御レジスターに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 1 の浮動小数点例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

パラメーター

項目	**説明**
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。

項目	説明
FRA	乗数を含むソース浮動小数点レジスターを指定します。
FRB	addend が入っているソース浮動小数点レジスターを指定します。
FRC	乗数を含むソース浮動小数点レジスターを指定します。

例

1. 以下のコードは、FPR 4 と FPR 5 の内容を乗算し、FPR 7 の内容を加算し、結果を FPR 6 に入れ、演算の結果を反映するように浮動小数点状況および制御レジスターを設定します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
# Assume FPR 5 contains 0x400C 0000 0000 0000.
# Assume FPR 7 contains 0x3DE2 6AB4 B33C 110A.
# Assume FPSCR = 0.
fmadd 6,4,5,7
# FPR 6 now contains 0xC070 D7FF FFFF F6CB.
# FPSCR now contains 0x8206 8000.
```

2. 次のコードは、FPR 4 と FPR 5 の内容を乗算し、FPR 7 の内容を加算し、結果を FPR 6 に入れ、演算の結果を反映するために浮動小数点状況および制御レジスターと条件レジスターのフィールド 1 を設定します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
# Assume FPR 5 contains 0x400C 0000 0000 0000.
# Assume FPR 7 contains 0x3DE2 6AB4 B33C 110A.
# Assume FPSCR = 0 and CR = 0.
fmadd. 6,4,5,7
# FPR 6 now contains 0xC070 D7FF FFFF F6CB.
# FPSCR now contains 0x8206 8000.
# CR now contains 0x0800 0000.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点レジスターの内容の解釈
32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

fmr (浮動移動レジスター) 命令

目的

ある浮動小数点レジスターの内容を別の浮動小数点レジスターにコピーします。

構文

ビット	VALUE
0-5	63
6 ～ 10	FRT (R)
11-15	///
16-20	FRB

ビット	VALUE
21-30	72
31	rc

項目	説明
FMR	*FRT*、 *FRB*
fmr。	*FRT*、 *FRB*

説明

fmr 命令は、浮動小数点レジスター (FPR) *FRB* の内容をターゲット FPR *FRT* に入れます。

fmr 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
FMR	なし	0	なし
fmr。	なし	1	FX、FEX、VX、OX

fmr 命令の 2 つの構文形式は、浮動小数点状況および制御レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 1 の浮動小数点例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

パラメーター

項目	説明
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRB	操作のソース浮動小数点レジスターを指定します。

例

1. 以下のコードは、FPR 4 の内容を FPR 6 にコピーし、操作の結果を反映するように浮動小数点状況および制御レジスターを設定します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
# Assume FPSCR = 0.
fmr 6,4
# FPR 6 now contains 0xC053 4000 0000 0000.
# FPSCR now contains 0x0000 0000.
```

2. 以下のコードは、FPR 25 の内容を FPR 6 にコピーし、浮動小数点状況および制御レジスターと条件レジスターのフィールド 1 を、操作の結果を反映するように設定します。

```
# Assume FPR 25 contains 0xFFFF FFFF FFFF FFFF.
# Assume FPSCR = 0 and CR = 0.
fmr. 6,25
# FPR 6 now contains 0xFFFF FFFF FFFF FFFF.
# FPSCR now contains 0x0000 0000.
# CR now contains 0x0000 0000.
```

fmsub または fms (Floating Multiply-Subtract) 命令

目的

中間丸め演算を行わずに、2 つの浮動小数点オペランドを乗算した結果から 1 つの浮動小数点オペランドを減算します。

構文

ビット	VALUE
0-5	63
6 ～ 10	FRT (R)
11-15	FRA
16-20	FRB
21-25	FRC
26-30	28
31	rc

PowerPC (R)

fmsub (fmsub) *FRT*、 *FRA*、 *FRC*、 *FRB*

fmsub。 *FRT*、 *FRA*、 *FRC*、 *FRB*

POWER® ファミリー

fms *FRT*、 *FRA*、 *FRC*、 *FRB*

FMS。 *FRT*、 *FRA*、 *FRC*、 *FRB*

ビット	値
0-5	59
6 ～ 10	FRT (R)
11-15	FRA
16-20	FRB
21-25	FRC
26-30	28
31	rc

PowerPC (R)

fmsubs (fmsubs) *FRT*、 *FRA*、 *FRC*、 *FRB*

fmsub。 *FRT*、 *FRA*、 *FRC*、 *FRB*

説明

fmsub および **fms** 命令は、浮動小数点レジスター (FPR) *FRA* 内の 64 ビット倍精度浮動小数点オペランドに、FPR *FRC* 内の 64 ビット倍精度浮動小数点オペランドを乗算の結果から 64 ビット倍精度浮動小数点オペランドを減算します。 *FRB*

fmsubs 命令は、FPR *FRA* の 32 ビットの単精度浮動小数点オペランドを FPR *FRC* の 32 ビットの単精度浮動小数点オペランドで乗算し、乗算の結果から FPR *FRB* の 32 ビットの単精度浮動小数点オペランドを減算します。

結果は、浮動小数点の状況および制御レジスターの浮動小数点丸め制御フィールド *RN* の制御下で丸められ、ターゲット FPR *FRT* に入れられます。

> **注:** オペランドが非正規化された数値の場合、そのオペランドは、操作が開始される前に非正規化されます。

浮動小数点状況および制御レジスターの浮動小数点結果フラグ・フィールドは、浮動小数点無効演算例外使用可能ビットが 1 の場合に、無効な演算例外を除き、結果のクラスおよび符号に設定されます。

fmsub、 **fmsubs**、および **fms** 命令には、それぞれ 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	**説明**		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
fmsub (fmsub)	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXSI、VXIMZ	0	なし
fmsub。	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXSI、VXIMZ	1	FX、FEX、VX、OX
fmsubs (fmsubs)	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXSI、VXIMZ	0	なし
fmsub。	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXSI、VXIMZ	1	FX、FEX、VX、OX
fms	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXSI、VXIMZ	0	なし
FMS。	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXSI、VXIMZ	1	FX、FEX、VX、OX

fmsub、 **fmsubs**、および **fms** 命令のすべての構文形式は、常に浮動小数点状況および制御レジスターに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 1 の浮動小数点例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

パラメーター

項目	**説明**
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRA	乗数を含むソース浮動小数点レジスターを指定します。
FRB	減算する数量を含むソース浮動小数点レジスターを指定します。
FRC	乗数を含むソース浮動小数点レジスターを指定します。

例

1. 次のコードは、FPR 4 と FPR 5 の内容を乗算し、乗算の積から FPR 7 の内容を減算し、結果を FPR 6 に入れ、演算の結果を反映するように浮動小数点状況と制御レジスターを設定します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
# Assume FPR 5 contains 0x400C 0000 0000 0000.
# Assume FPR 7 contains 0x3DE2 6AB4 B33c 110A.
# Assume FPSCR = 0.
fmsub 6,4,5,7
# FPR 6 now contains 0xC070 D800 0000 0935.
# FPSCR now contains 0x8202 8000.
```

2. 次のコードは、FPR 4 と FPR 5 の内容を乗算し、乗算の積から FPR 7 の内容を減算し、その結果を FPR 6 に入れ、演算の結果を反映するように浮動小数点状況および制御レジスターと条件レジスターのフィールド 1 を設定します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
# Assume FPR 5 contains 0x400C 0000 0000 0000.
# Assume FPR 7 contains 0x3DE2 6AB4 B33c 110A.
# Assume FPSCR = 0 and CR = 0.
fmsub. 6,4,5,7
# FPR 6 now contains 0xC070 D800 0000 0935.
# FPSCR now contains 0x8202 8000.
# CR now contains 0x0800 0000.
```

fmul または fm (Floating Multiply) 命令

目的

2 つの浮動小数点オペランドを乗算します。

構文

ビット	VALUE
0-5	63
6 ～ 10	FRT (R)
11-15	FRA
16-20	///
21-25	FRC
26-30	25
31	rc

PowerPC (R)

fmul (fmul) *FRT*、 *FRA*、 *FRC*

フムル *FRT*、 *FRA*、 *FRC*

POWER® ファミリー

FM *FRT*、 *FRA*、 *FRC*

fm: *FRT*、 *FRA*、 *FRC*

ビット	値
0-5	59
6 ～ 10	FRT (R)
11-15	FRA
16-20	///
21-25	FRC
26-30	25
31	rc

PowerPC (R)

マルチボリューム	*FRT*、 *FRA*、 *FRC*
fmuls。	*FRT*、 *FRA*、 *FRC*

説明

fmul および **fm** 命令は、浮動小数点レジスター (FPR) *FRA* の 64 ビット倍精度浮動小数点オペランドに、FPR *FRC* の 64 ビット倍精度浮動小数点オペランドを乗算します。

fmuls 命令は、FPR *FRA* の 32 ビット単精度浮動小数点オペランドに、FPR *FRC* の 32 ビット単精度浮動小数点オペランドを乗算します。

結果は、浮動小数点の状況および制御レジスターの浮動小数点丸め制御フィールド *RN* の制御下で丸められ、ターゲット FPR *FRT* に入れられます。

2 つの浮動小数点数の乗算は、2 つの仮数の指数の加算と乗算に基づいています。

> **注:** オペランドが非正規化された数値の場合、そのオペランドは、操作が開始される前に非正規化されます。

浮動小数点状況および制御レジスターの浮動小数点結果フラグ・フィールドは、浮動小数点無効演算例外使用可能ビットが 1 の場合に、無効な演算例外を除き、結果のクラスおよび符号に設定されます。

fmul、 **fmuls**、および **fm** 命令には、それぞれ 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	**説明**		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
fmul (fmul)	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXIMZ	0	なし
フムル	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXIMZ	1	FX、FEX、VX、OX
マルチボリューム	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXIMZ	0	なし
fmuls。	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXIMZ	1	FX、FEX、VX、OX
FM	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXIMZ	0	なし
fm:	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXIMZ	1	FX、FEX、VX、OX

fmul、**fmuls**、および **fm** 命令のすべての構文形式は、常に浮動小数点状況および制御レジスターに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 1 の浮動小数点例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

パラメーター

項目	説明
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRA	操作のソース浮動小数点レジスターを指定します。
FRC	操作のソース浮動小数点レジスターを指定します。

例

1. 以下のコードは、FPR 4 と FPR 5 の内容を乗算し、結果を FPR 6 に入れ、演算の結果を反映するように浮動小数点状況および制御レジスターを設定します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
# Assume FPR 5 contains 0x400C 0000 0000 0000.
# Assume FPSCR = 0.
fmul 6,4,5
# FPR 6 now contains 0xC070 D800 0000 0000.
# FPSCR now contains 0x0000 8000.
```

2. 以下のコードは、FPR 4 と FPR 25 の内容を乗算し、結果を FPR 6 に入れ、条件レジスター・フィールド 1 と浮動小数点状況および制御レジスターを設定して、操作の結果を反映します。

```
# Assume FPR  4 contains 0xC053 4000 0000 0000.
# Assume FPR 25 contains 0xFFFF FFFF FFFF FFFF.
# Assume FPSCR = 0 and CR = 0.
fmul. 6,4,25
# FPR 6 now contains 0xFFFF FFFF FFFF FFFF.
# FPSCR now contains 0x0001 1000.
# CR now contains 0x0000 0000.
```

fnabs (浮動負の絶対値) 命令

目的

浮動小数点レジスターの絶対内容を否定し、結果を別の浮動小数点レジスターに入れます。

構文

ビット	VALUE
0-5	63
6 ～ 10	FRT (R)
11-15	///
16-20	FRB
21-30	136
31	/

項目	説明
fnabs (fnabs)	*FRT*、 *FRB*
fnabs.	*FRT*、 *FRB*

説明

fnabs 命令は、ビット 0 が 1 に設定された浮動小数点レジスター (FPR) *FRB* の内容の負の絶対値をターゲット FPR *FRT* に入れます。

fnabs 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
fnabs (fnabs)	なし	0	なし
fnabs.	なし	1	FX、FEX、VX、OX

fnabs 命令の 2 つの構文形式は、浮動小数点状況および制御レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 1 の浮動小数点例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

パラメーター

項目	説明
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRB	操作のソース浮動小数点レジスターを指定します。

例

1. 以下のコードは、FPR 5 の絶対内容を否定し、結果を FPR 6 に入れます。

```
# Assume FPR 5 contains 0x400C 0000 0000 0000.
fnabs 6,5
# FPR 6 now contains 0xC00C 0000 0000 0000.
```

2. 以下のコードは、FPR 4 の絶対内容を否定し、結果を FPR 6 に入れ、演算の結果を反映するように条件レジスター・フィールド 1 を設定します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
# Assume CR = 0.
fnabs. 6,4
# FPR 6 now contains 0xC053 4000 0000 0000.
# CR now contains 0x0.
```

fneg (フローティング否定) 命令

目的

浮動小数点レジスターの内容を否定し、その結果を別の浮動小数点レジスターに入れます。

構文

ビット	VALUE
0-5	63
6 〜 10	FRT (R)
11-15	///
16-20	FRB
21-30	40
31	rc

項目	説明
フネグ (fneg)	*FRT*、 *FRB*
fneg。	*FRT*、 *FRB*

説明

fneg 命令は、浮動小数点レジスター *FRB* の否定された内容をターゲット FPR *FRT* に入れます。

fneg 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
フネグ (fneg)	なし	0	なし
fneg。	なし	1	FX、FEX、VX、OX

fneg 命令の 2 つの構文形式は、浮動小数点状況および制御レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 1 の浮動小数点例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

パラメーター

項目	説明
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRB	操作のソース浮動小数点レジスターを指定します。

例

1. 以下のコードは、FPR 5 の内容を否定し、結果を FPR 6 に入れます。

```
# Assume FPR 5 contains 0x400C 0000 0000 0000.
```

```
fneg 6,5
# FPR 6 now contains 0xC00C 0000 0000 0000.
```

2. 以下のコードは、FPR 4 の内容を否定し、結果を FPR 6 に入れ、演算の結果を反映するように条件レジスター・フィールド 1 を設定します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
fneg. 6,4
# FPR 6 now contains 0x4053 4000 0000 0000.
# CR now contains 0x0000 0000.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点移動命令
浮動小数点移動命令は、ある FPR から別の FPR にデータをコピーします。

浮動小数点レジスターの内容の解釈
32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

fnmadd 命令または fnma (負の浮動乗算-加算) 命令

目的

2 つの浮動小数点オペランドを乗算し、結果を 1 つの浮動小数点オペランドに加算し、結果の負の値を浮動小数点レジスターに入れます。

構文

ビット	VALUE
0-5	63
6 ～ 10	FRT (R)
11-15	FRA
16-20	FRB
21-25	FRC
26-30	31
31	rc

PowerPC (R)

fnmadd *FRT*、 *FRA*、 *FRC*、 *FRB*

fnmadd。 *FRT*、 *FRA*、 *FRC*、 *FRB*

POWER® ファミリー

fnma (fnma) *FRT*、 *FRA*、 *FRC*、 *FRB*

fnma: *FRT*、 *FRA*、 *FRC*、 *FRB*

ビット	値
0-5	59

ビット	値
6 ～ 10	FRT (R)
11-15	FRA
16-20	FRB
21-25	FRC
26-30	31
31	rc

PowerPC (R)

fnmadd *FRT*、 *FRA*、 *FRC*、 *FRB*

fnmadd。 *FRT*、 *FRA*、 *FRC*、 *FRB*

説明

fnmadd および **fnma** 命令は、浮動小数点レジスター (FPR) *FRA* 内の 64 ビット倍精度浮動小数点オペランドに、FPR *FRC* 内の 64,bit、倍精度浮動小数点オペランドを乗算し、FPR *FRB* 内の 64 ビット倍精度浮動小数点オペランドを加算します。

fnmadd 命令は、FPR *FRA* の 32 ビットの単精度浮動小数点オペランドに FPR *FRC* の 32 ビットの単精度浮動小数点オペランドを乗算し、FPR *FRB* の 32 ビットの単精度浮動小数点オペランドに乗算の結果を加算します。

加算の結果は、浮動小数点状況および制御レジスターの浮動小数点丸め制御フィールド *RN* の制御の下で丸められます。

> **注:** オペランドが非正規化された数値の場合、そのオペランドは、操作が開始される前に非正規化されます。

fnmadd および **fnma** 命令は、最終結果が否定された **fmadd** および **fma** (Floating Multiply-Add Single) 命令と同じですが、以下の例外があります。

- 静止 NaN (QNaN) は、「符号」ビットに影響を与えずに伝搬します。
- 無効化された無効演算例外の結果として生成される QNaN の「符号」ビットは 0 です。
- 無効な演算例外の結果として QNaN に変換されるシグナリング NaN (SNaN) は、その「符号」ビットには影響しません。

浮動小数点状況および制御レジスターの浮動小数点結果フラグ・フィールドは、浮動小数点無効演算例外使用可能ビットが 1 の場合に、無効な演算例外を除き、結果のクラスおよび符号に設定されます。

fnmadd、 **fnmadd**、および **fnma** 命令には、それぞれ 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコードビット (RC)**	**条件 レジスター・フィールド 1**
fnmadd	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI、VXIMZ	0	なし
fnmadd。	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI、VXIMZ	1	FX、FEX、VX、OX
fnmadd	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI、VXIMZ	0	なし

項目	説明		
fnmadd。	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI、VXIMZ	1	FX、FEX、VX、OX
fnma (fnma)	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI、VXIMZ	0	なし
fnma:	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI、VXIMZ	1	FX、FEX、VX、OX

fnmadd、 **fnmadd**、および **fnma** 命令のすべての構文形式は、常に浮動小数点状況および制御レジスターに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 1 の浮動小数点例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

> **注:** 丸めは、加算の結果が否定される前に行われます。 *RN* によっては、不正確な値になる場合があります。

パラメーター

項目	説明
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRA	操作のソース浮動小数点レジスターを指定します。
FRB	操作のソース浮動小数点レジスターを指定します。
FRC	操作のソース浮動小数点レジスターを指定します。

例

1. 以下のコードは、FPR 4 と FPR 5 の内容を乗算し、結果を FPR 7 の内容に加算し、否定された結果を FPR 6 に保管し、浮動小数点状況と制御レジスターを演算の結果を反映するように設定します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
# Assume FPR 5 contains 0x400C 0000 0000 0000.
# Assume FPR 7 contains 0x3DE2 6AB4 B33c 110A.
# Assume FPSCR = 0.
fnmadd 6,4,5,7
# FPR 6 now contains 0x4070 D7FF FFFF F6CB.
# FPSCR now contains 0x8206 4000.
```

2. 以下のコードは、FPR 4 と FPR 5 の内容を乗算し、結果を FPR 7 の内容に加算し、否定された結果を FPR 6 に保管し、演算の結果を反映するために浮動小数点状況および制御レジスターと条件レジスターのフィールド 1 を設定します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
# Assume FPR 5 contains 0x400C 0000 0000 0000.
# Assume FPR 7 contains 0x3DE2 6AB4 B33c 110A.
# Assume FPSCR = 0 and CR = 0.
fnmadd. 6,4,5,7
# FPR 6 now contains 0x4070 D7FF FFFF F6CB.
# FPSCR now contains 0x8206 4000.
# CR now contains 0x0800 0000.
```

関連概念

浮動小数点プロセッサー

浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点レジスターの内容の解釈
32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

fnmsub または fnms (浮動小数点負乗算-減算) 命令

目的

2 つの浮動小数点オペランドを乗算し、結果から 1 つの浮動小数点オペランドを減算し、結果の負の値を浮動小数点レジスターに入れます。

構文

ビット	VALUE
0-5	63
6 ～ 10	FRT (R)
11-15	FRA
16-20	FRB
21-25	FRC
26-30	30
31	rc

PowerPC (R)

fnmsub *FRT*、 *FRA*、 *FRC*、 *FRB*

fnmsub。 *FRT*、 *FRA*、 *FRC*、 *FRB*

POWER® ファミリー

fnms *FRT*、 *FRA*、 *FRC*、 *FRB*

fnms。 *FRT*、 *FRA*、 *FRC*、 *FRB*

ビット	VALUE
0-5	59
6 ～ 10	FRT (R)
11-15	FRA
16-20	FRB
21-25	FRC
26-30	
30	rc

PowerPC (R)

fnmsubs (fnmsubs) *FRT*、 *FRA*、 *FRC*、 *FRB*

PowerPC (R)

fnmsubs。 *FRT*、 *FRA*、 *FRC*、 *FRB*

説明

FnMS および **FnMSUB** 命令は、浮動小数点レジスター (FPR) の 64 ビットの倍精度浮動小数点オペランド *FRA (R)* に、FPR *FRC (R)*の 64 ビットの倍精度浮動小数点オペランドを乗算し、乗算の結果から FPR 連邦準備制度 の 64 ビットの倍精度浮動小数点オペランドを減算し、否定結果をターゲット FPR *FRT (R)*に入れます。

FnMSUB 命令は、FPR 内の 32 ビットの単精度浮動小数点オペランドに *FRA (R)* FPR 内の 32 ビットの単精度浮動小数点オペランドを乗算し *FRC (R)*、乗算の結果から FPR 内の 32 ビットの単精度浮動小数点オペランドを減算し 連邦準備制度 、その否定結果をターゲット FPR に入れます *FRT (R)*。

減算結果は、浮動小数点の状況および制御レジスターの浮動小数点丸め制御フィールド *RN* の制御下で丸められます。

> **注:** オペランドが非正規化された数値の場合、そのオペランドは、操作が開始される前に非正規化されます。

fnms および **fnmsub** 命令は、**fmsub** および **fms** (Floating Multiply-Subtract Single) 命令と同じで、最終結果は否定されますが、以下の例外があります。

- 静止 NaN (QNaN) は、「符号」ビットに影響を与えずに伝搬します。
- 無効化された無効演算例外の結果として生成される QNaN の「符号」ビットはゼロです。
- 無効な演算例外の結果として QNaN に変換されるシグナリング NaN (SNaN) は、その「符号」ビットには影響しません。

浮動小数点状況および制御レジスターの浮動小数点結果フラグ・フィールドは、浮動小数点無効演算例外使用可能ビットが 1 の場合に、無効な演算例外を除き、結果のクラスおよび符号に設定されます。

fnmsub、 **fnmsubs**、および **fnms** 命令には、それぞれ 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	**説明**		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
fnmsub	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI、VXIMZ	0	なし
fnmsub。	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI、VXIMZ	1	FX、FEX、VX、OX
fnmsubs (fnmsubs)	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI、VXIMZ	0	なし
fnmsubs。	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI、VXIMZ	1	FX、FEX、VX、OX
fnms	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI、VXIMZ	0	なし
fnms。	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI、VXIMZ	1	FX、FEX、VX、OX

fnmsub、 **fnmsubs**、および **fnms** 命令のすべての構文形式は、常に浮動小数点状況および制御レジスターに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 1 の浮動小数点例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

注: 丸めは、加算の結果が否定される前に行われます。 *RN* によっては、不正確な値になる場合があります。

パラメーター

項目	説明
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRA	操作の最初のソース浮動小数点レジスターを指定します。
FRB	演算用の 2 番目のソース浮動小数点レジスターを指定します。
FRC	操作用の 3 番目のソース浮動小数点レジスターを指定します。

例

1. 次のコードは、FPR 4 と FPR 5 の内容を乗算し、結果から FPR 7 の内容を減算し、否定された結果を FPR 6 に保管し、演算の結果を反映するために浮動小数点状況および制御レジスターと条件レジスターのフィールド 1 を設定します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
# Assume FPR 5 contains 0x400C 0000 0000 0000.
# Assume FPR 7 contains 0x3DE2 6AB4 B33c 110A.
# Assume FPSCR = 0.
fnmsub 6,4,5,7
# FPR 6 now contains 0x4070 D800 0000 0935.
# FPSCR now contains 0x8202 4000.
```

2. 次のコードは、FPR 4 と FPR 5 の内容を乗算し、結果から FPR 7 の内容を減算し、否定された結果を FPR 6 に保管し、演算の結果を反映するために浮動小数点状況および制御レジスターと条件レジスターのフィールド 1 を設定します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
# Assume FPR 5 contains 0x400C 0000 0000 0000.
# Assume FPR 7 contains 0x3DE2 6AB4 B33c 110A.
# Assume FPSCR = 0 and CR = 0.
fnmsub. 6,4,5,7
# FPR 6 now contains 0x4070 D800 0000 0935.
# FPSCR now contains 0x8202 4000.
# CR now contains 0x0800 0000.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点レジスターの内容の解釈
32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

fres (Floating Reciprocal Estimate Single) 命令

目的

浮動小数点オペランドの逆数の単精度推定値を計算します。

注: fres 命令は、PowerPC ® アーキテクチャーでのみ定義されており、オプションの命令です。PowerPC 603 RISC Microprocessor、および PowerPC 604 RISC Microprocessor でサポートされていますが、PowerPC ® 601 RISC Microprocessor ではサポートされていません。

構文

ビット	VALUE
0-5	59
6 〜 10	FRT (R)
11-15	///
16-20	FRB
21-25	///
26-30	24
31	rc

PowerPC (R)

fres *FRT*、 *FRB*

fres。 *FRT*、 *FRB*

説明

fres 命令は、浮動小数点レジスター (FPR) *FRB* 内の 64 ビット倍精度浮動小数点オペランドの逆数の単精度推定値を計算し、その結果を FPR *FRT* に入れます。

レジスター *FRT* に入れられる見積もりは、 *FRB* の逆数の 256 のうちの 1 つの部分の精度まで正確です。*FRT* に入れられる値は、インプリメンテーション間、および同じインプリメンテーションでの異なる実行間で異なる場合があります。

次の表は、特殊な条件を要約したものです。

項目	**説明**	
特殊な条件		
オペランド	**結果**	**例外**
負の無限大	負の 0	なし
負の 0	負の無限大 [1]	ZX (X)
正 0	正の無限大 [1]	ZX (X)
正の無限大	正 0	なし
SNAN	QNaN[2]	VXSNAN (V)
QNaN (N)	QNaN (N)	なし

FPSCRZE = 1 の場合は 1No になります。

FPSCRVE = 1 の場合は、 2No になります。

FPSCRVE = 1 の場合の無効な操作例外と、FPSCRZE = 1 の場合のゼロ除算例外を除き、FPSCRFPRF は結果のクラスと符号に設定されます。

fres 命令には 2 つの構文形式があります。 どちらの構文形式も、常に FPSCR レジスターに影響します。各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
fres	C、FL、FG、FE、FU、FR、FI、FX、OX、UX、ZX、VXSNAN	0	なし
fres。	C、FL、FG、FE、FU、FR、FI、FX、OX、UX、ZX、VXSNAN	1	FX、FEX、VX、OX

fres 。 シンタックス・フォームはレコード (Rc) ビットを 1 に設定します。命令は、条件レジスター・フィールド 1 (CR1) の浮動小数点例外 (FX)、浮動小数点有効例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。 **fres** 構文形式は、レコード (Rc) ビットを 0 に設定し、条件レジスター・フィールド 1 (CR1) には影響しません。

パラメーター

項目	説明
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRB	操作のソース浮動小数点レジスターを指定します。

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点演算命令
浮動小数点演算命令は、浮動小数点レジスターに含まれる浮動小数点データに対して算術演算を実行します。

浮動小数点レジスターの内容の解釈
32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

frsp (単精度への浮動丸め) 命令

目的

64 ビットの倍精度浮動小数点オペランドを単精度に丸め、結果を浮動小数点レジスターに入れます。

構文

ビット	VALUE
0-5	63
6 〜 10	FRT (R)
11-15	///
16-20	FRB
21-30	12
31	rc

項目	説明
FRSP	*FRT*、 *FRB*
frsp:	*FRT*、 *FRB*

説明

frsp 命令は、浮動小数点状況および制御レジスターの浮動小数点制御フィールドによって指定された丸めモードを使用して、浮動小数点レジスター (FPR) *FRB* の 64 ビット倍精度浮動小数点オペランドを単精度に丸め、その結果をターゲット FPR *FRT* に入れます。

浮動小数点状況および制御レジスターの浮動小数点結果フラグ・フィールドは、浮動小数点状況および制御レジスター浮動小数点無効演算例外使用可能ビットが 1 の場合に、無効演算 (SNaN) を除き、結果のクラスおよび符号に設定されます。

frsp 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
FRSP	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN	0	なし
frsp:	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN	1	FX、FEX、VX、OX

frsp 命令の 2 つの構文形式は、常に浮動小数点状況および制御レジスターに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 1 の浮動小数点例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

注:

1. **frsp** 命令は、直前の浮動小数点算術演算のターゲット・レジスターをソース・レジスター (*FRB*) として使用します。 **frsp** 命令は、ソースにこのレジスターを使用する場合、先行する浮動小数点算術演算に 依存 していると言われます。
2. **frsp** 命令とそれが従属する演算との間では、非従属浮動小数点算術演算は 2 つより少なくなります。
3. 算術演算の倍精度結果の大きさが、丸めの前に 2 * *128 より小さくなっています。
4. 丸め後の倍精度結果の大きさは正確に 2 * *128 です。

エラー結果

エラーが発生した場合、ターゲット・レジスター *FRT* に入れられる結果の大きさは 2 * *128 です。

```
X'47F0000000000000' or X'C7F0000000000000'
```

これは有効な単精度値ではありません。 浮動小数点状況および制御レジスターの設定と条件レジスターの設定は、結果がオーバーフローしない場合と同じになります。

エラーの回避

上記のエラーによってアプリケーションで重大な問題が発生する場合は、以下の 2 つの方法のいずれかを使用してエラーを回避することができます。

1. 浮動小数点算術演算と従属 **frsp** 命令の間には、2 つの非従属浮動小数点演算を入れます。 これらの非従属浮動小数点演算のターゲット・レジスターは、 **frsp** 命令がソース・レジスター *FRB* として使用するレジスターと同じであってはなりません。

2. 2 つの **frsp** 命令を挿入するのは、 **frsp** 命令が 3 つより少ない浮動小数点命令に先行する算術演算に依存している可能性がある場合です。

どちらのソリューションでも、特定のアプリケーションに依存する量だけパフォーマンスが低下します。

パラメーター

項目	説明
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRB	操作のソース浮動小数点レジスターを指定します。

例

1. 以下のコードは、FPR 4 の内容を単精度に丸め、結果を FPR 6 に入れ、演算の結果を反映するように浮動小数点状況および制御レジスターを設定します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
# Assume FPSCR = 0.
frsp 6,4
# FPR 6 now contains 0xC053 4000 0000 0000.
# FPSCR now contains 0x0000 8000.
```

2. 以下のコードは、FPR 4 の内容を単精度に丸め、結果を FPR 6 に入れ、演算の結果を反映するために浮動小数点状況および制御レジスターと条件レジスターのフィールド 1 を設定します。

```
# Assume CR contains 0x0000 0000.
# Assume FPR 4 contains 0xFFFF FFFF FFFF FFFF.
# Assume FPSCR = 0.
frsp. 6,4
# FPR 6 now contains 0xFFFF FFFF E000 0000.
# FPSCR now contains 0x0001 1000.
# CR now contains 0x0000 0000.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点レジスターの内容の解釈
32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

浮動小数点演算命令
浮動小数点演算命令は、浮動小数点レジスターに含まれる浮動小数点データに対して算術演算を実行します。

frsqrte (Floating Reciprocal Square Root Estimate) 命令

目的

浮動小数点オペランドの平方根の逆数の倍精度の推定値を計算します。

> **注: frsqrte** 命令は PowerPC ® アーキテクチャーでのみ定義され、オプションの命令です。 PowerPC 603 RISC マイクロプロセッサーおよび PowerPC 604 RISC マイクロプロセッサーでサポートされますが、PowerPC ® 601 RISC マイクロプロセッサーではサポートされません。

構文

ビット	VALUE
0-5	63
6 ～ 10	FRT (R)
11-15	///
16-20	FRB
21-25	///
26-30	26
31	rc

PowerPC (R)

frsqrte (frsqrte) *FRT*、 *FRB*

frsqrte: *FRT*、 *FRB*

説明

frsqrte 命令は、浮動小数点レジスター (FPR) *FRB* 内の 64 ビット倍精度浮動小数点オペランドの平方根の逆数の倍精度推定値を計算し、結果を FPR *FRT* に入れます。

レジスター *FRT* に入れられる見積もりは、 *FRB* の平方根の逆数の 32 のうちの 1 つの部分の精度に正しくなります。 *FRT* に入れられる値は、インプリメンテーション間、および同じインプリメンテーションでの異なる実行間で異なる場合があります。

次の表は、特殊な条件を要約したものです。

項目	**説明**	
特殊な条件		
オペランド	**結果**	**例外**
負の無限大	QNaN[1]	VXSQRT (R)
0 より小さい値	QNaN[1]	VXSQRT (R)
負の 0	負の無限大 [2]	ZX (X)
正 0	正の無限大 [2]	ZX (X)
正の無限大	正 0	なし
SNAN	QNaN[1]	VXSNAN (V)
QNaN (N)	QNaN (N)	なし

1No の結果 (FPSCRVE = 1 の場合)。

FPSCRZE = 1 の場合は、 2No になります。

FPSCRVE = 1 の場合の無効な操作例外と、FPSCRZE = 1 の場合のゼロ除算例外を除き、FPSCRFPRF は結果のクラスと符号に設定されます。

frsqrte 命令には 2 つの構文形式があります。 どちらの構文形式も、常に FPSCR に影響します。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	**説明**		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**

項目	説明		
frsqrte (frsqrte)	C、FL、FG、FE、FU、FR、FI、FX、ZX、VXSNAN、VXSQRT	0	なし
frsqrte:	C、FL、FG、FE、FU、FR、FI、FX、ZX、VXSNAN、VXSQRT	1	FX、FEX、VX、OX

frstrte 。 シンタックス・フォームはレコード (Rc) ビットを 1 に設定します。命令は、条件レジスター・フィールド 1 (CR1) の浮動小数点例外 (FX)、浮動小数点有効例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。 **frstrte** 構文形式は、レコード (Rc) ビットを 0 に設定します。命令は、条件レジスター・フィールド 1 (CR1) には影響しません。

パラメーター

項目	説明
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRB	操作のソース浮動小数点レジスターを指定します。

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点演算命令
浮動小数点演算命令は、浮動小数点レジスターに含まれる浮動小数点データに対して算術演算を実行します。

浮動小数点レジスターの内容の解釈
32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

fsel (浮動小数点選択) 命令

目的

別の浮動小数点オペランドをゼロと比較した結果に基づいて、2 つの浮動小数点オペランドのいずれかをターゲット・レジスターに入れます。

> **注: fsel** 命令は、PowerPC ® アーキテクチャーでのみ定義され、オプションの命令です。 PowerPC 603 RISC マイクロプロセッサーおよび PowerPC 604 RISC マイクロプロセッサーでサポートされますが、PowerPC ® 601 RISC マイクロプロセッサーではサポートされません。

構文

ビット	VALUE
0-5	63
6 ～ 10	FRT (R)
11-15	FRA
16-20	FRB
21-25	FRC

ビット	VALUE
26-30	23
31	rc

PowerPC (R)

fsel *FRT*、 *FRA*、 *FRC*、 *FRB*

Fsel。 *FRT*、 *FRA*、 *FRC*、 *FRB*

説明

浮動小数点レジスター (FPR) *FRA* の倍精度浮動小数点オペランドは、値ゼロと比較されます。 *FRA* の値がゼロ以上の場合、浮動小数点レジスター *FRT* は浮動小数点レジスター *FRC* の内容に設定されます。 *FRA* の値がゼロより小さいか NaN の場合、浮動小数点レジスター *FRT* は浮動小数点レジスター *FRB* の内容に設定されます。比較では、ゼロの符号は無視されます。+ 0 と -0 の両方がゼロになります。

fesl 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文形式	**FPSCR ビット**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
fsel	なし	0	なし
Fsel。	なし	1	FX、FEX、VX、OX

fsel 命令の 2 つの構文形式は、浮動小数点状況および制御レジスター・フィールドには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は、条件レジスター・フィールド 1 の浮動小数点例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

パラメーター

項目	説明
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRA	ゼロと比較する値を持つ浮動小数点レジスターを指定します。
FRB	*FRA* がゼロより小さいか NaN である場合に使用される値が入っているソース浮動小数点レジスターを指定します。
FRC	*FRA* がゼロ以上の場合に使用される値が入っているソース浮動小数点レジスターを指定します。

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点レジスターの内容の解釈
32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

fsqrt (Floating Square Root, Double-Precision) 命令

目的

浮動小数点レジスターの内容の平方根を計算し、結果を浮動小数点レジスターに入れます。

構文

ビット	VALUE
0-5	63
6 ～ 10	D
11-15	00000
16-20	B
21-25	00000
26-30	22
31	rc

PowerPC (R)

fsqrt (fsqrt) *FRT*、 *FRB* (Rc=0)

fsqrt: *FRT*、 *FRB* (Rc=1)

説明

浮動小数点レジスター (FPR) *FRB* のオペランドの平方根は、レジスター FPR *FRT* に入れられます。

結果の仮数の最上位ビットが 1 でない場合、結果は正規化されます。 結果は、FPSCR の浮動小数点丸め制御フィールド RN の制御下でターゲット精度に丸められ、レジスター FPR *FRT* に入れられます。

オペランドのさまざまな特殊値を使用した演算の要約を以下に示します。

オペランド	結果	例外
-無限大	QNaN* (N)	VXSQRT (R)
< 0	QNaN* (N)	VXSQRT (R)
- 0	- 0	なし
+ 無限大	+ 無限大	なし
SNAN	QNaN* (N)	VXSNAN (V)
QNaN (N)	QNaN (N)	なし

注: * FPSCR [VE] = 1 の場合は結果なし

FPSCR [FPRF] は、FPSCR [VE] = 1 の場合の無効な演算例外を除き、結果のクラスと符号に設定されます。

fsqrt 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
fsqrt (fsqrt)	FPRF、FR、FI、FX、XX、VXSNAN、VXSQRT	0	なし

項目	説明		
fsqrt:	FPRF、FR、FI、FX、XX、VXSNAN、VXSQRT	1	FX、FEX、VX、OX

パラメーター

項目	説明
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRB	操作のソース浮動小数点レジスターを指定します。

インプリメンテーション

この命令は、オプションで PowerPC 実装に対して定義されます。 この命令をサポートしないインプリメンテーションでこの命令を使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

この命令は、PowerPC ® アーキテクチャーのオプションの命令であり、すべてのマシンに実装できるわけではありません。

fsqrts (浮動平方根単一) 命令

目的

浮動小数点レジスターの内容の単精度平方根を計算し、結果を浮動小数点レジスターに入れます。

構文

ビット	VALUE
0-5	59
6 ～ 10	D
11-15	00000
16-20	B
21-25	00000
26-30	22
31	rc

PowerPC (R)

fsqrts *FRT*、 *FRB* (Rc=0)

fsqrts。 *FRT*、 *FRB* (Rc=1)

説明

浮動小数点レジスター (FPR) *FRB* の浮動小数点オペランドの平方根は、レジスター FPR *FRT* に入れられます。

結果の仮数の最上位ビットが 1 でない場合、結果は正規化されます。 結果は、FPSCR の浮動小数点丸め制御フィールド RN の制御下でターゲット精度に丸められ、レジスター FPR *FRT* に入れられます。

オペランドのさまざまな特殊値を使用した演算の要約を以下に示します。

オペランド	結果	例外
-無限大	QNaN* (N)	VXSQRT (R)
< 0	QNaN* (N)	VXSQRT (R)
- 0	- 0	なし
+ 無限大	+ 無限大	なし
SNAN	QNaN* (N)	VXSNAN (V)
QNaN (N)	QNaN (N)	なし

注: * FPSCR [VE] = 1 の場合は結果なし

FPSCR [FPRF] は、FPSCR [VE] = 1 の場合の無効な演算例外を除き、結果のクラスと符号に設定されます。

fsqrts 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
fsqrts	FPRF、FR、FI、FX、XX、VXSNAN、VXSQRT	0	なし
fsqrts。	FPRF、FR、FI、FX、XX、VXSNAN、VXSQRT	1	FX、FEX、VX、OX

パラメーター

項目	説明
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRB	操作のソース浮動小数点レジスターを指定します。

インプリメンテーション

この命令は、オプションで PowerPC 実装に対して定義されます。 この命令をサポートしないインプリメンテーションでこの命令を使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

この命令は、PowerPC ® アーキテクチャーのオプションの命令であり、すべてのマシンに実装できるわけではありません。

fsub または fs (浮動減算) 命令

目的

ある浮動小数点オペランドを別の浮動小数点オペランドから減算し、その結果を浮動小数点レジスターに入れます。

構文

ビット	VALUE
0-5	63
6 ～ 10	FRT (R)

ビット	VALUE
11-15	FRA
16-20	FRB
21-25	///
26-30	20
31	rc

PowerPC (R)

fsub *FRT*、 *FRA*、 *FRB*

fsub: *FRT*、 *FRA*、 *FRB*

PowerPC (R)

Fs *FRT*、 *FRA*、 *FRB*

fs。 *FRT*、 *FRA*、 *FRB*

ビット	値
0-5	59
6 ～ 10	FRT (R)
11-15	FRA
16-20	FRB
21-25	///
26-30	20
31	rc

PowerPC (R)

fsub *FRT*、 *FRA*、 *FRB*

fsub。 *FRT*、 *FRA*、 *FRB*

説明

fsub および **fs** 命令は、FPR *FRA* の 64 ビット倍精度浮動小数点オペランドから浮動小数点レジスター (FPR) *FRB* の 64 ビット倍精度浮動小数点オペランドを減算します。

fsubs 命令は、FPR *FRA* の 32 ビット単精度浮動小数点オペランドから FPR *FRB* の 32 ビット単精度浮動小数点オペランドを減算します。

結果は、浮動小数点の状況および制御レジスターの浮動小数点丸め制御フィールド *RN* の制御下で丸められ、ターゲット FPR *FRT* に入れられます。

fsub 命令の実行は、 **fadd** の実行と同じです。ただし、FPR *FRB* の内容は、ビット 0 が反転した操作に関与します。

fs 命令の実行は、 **fa** の実行と同じです。ただし、FPR *FRB* の内容は、ビット 0 を反転した操作に関与します。

浮動小数点状況および制御レジスターの浮動小数点結果フラグ・フィールドは、浮動小数点無効演算例外使用可能ビットが 1 の場合に、無効な演算例外を除き、結果のクラスおよび符号に設定されます。

fsub、 **fsubs**、および **fs** 命令には、それぞれ 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文 Form	**浮動小数点状況および 制御レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
fsub	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI	0	なし
fsub:	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI	1	FX、FEX、VX、OX
fsub	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI	0	なし
fsub。	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI	1	FX、FEX、VX、OX
Fs	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI	0	なし
fs。	C、FL、FG、FE、FU、FR、FI、OX、UX、XX、VXSNAN、VXISI	1	FX、FEX、VX、OX

fsub、 **fsubs**、および **fs** 命令のすべての構文形式は、常に浮動小数点状況および制御レジスターに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 1 の浮動小数点例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

パラメーター

項目	説明
FRT (FRT)	操作のターゲット浮動小数点レジスターを指定します。
FRA	操作のソース浮動小数点レジスターを指定します。
FRB	操作のソース浮動小数点レジスターを指定します。

例

1. 以下のコードは、FPR 4 の内容から FPR 5 の内容を減算し、結果を FPR 6 に入れ、演算の結果を反映するように浮動小数点状況および制御レジスターを設定します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
# Assume FPR 5 contains 0x400C 0000 0000 0000.
# Assume FPSCR = 0.
fsub 6,4,5
# FPR 6 now contains 0xC054 2000 0000 0000.
# FPSCR now contains 0x0000 8000.
```

2. 以下のコードは、FPR 4 の内容から FPR 5 の内容を減算し、結果を FPR 6 に入れ、演算の結果を反映するように浮動小数点状況および制御レジスターと条件レジスターのフィールド 1 を設定します。

```
# Assume FPR 4 contains 0xC053 4000 0000 0000.
# Assume FPR 5 contains 0x400C 0000 0000 0000.
# Assume FPSCR = 0 and CR = 0.
fsub. 6,5,4
```

```
# FPR 6 now contains 0x4054 2000 0000 0000.
# FPSCR now contains 0x0000 4000.
# CR now contains 0x0000 0000.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点演算命令
浮動小数点演算命令は、浮動小数点レジスターに含まれる浮動小数点データに対して算術演算を実行します。

浮動小数点レジスターの内容の解釈
32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

icbi (Instruction Cache Block Invalidate) 命令

目的

命令キャッシュ内でアドレス指定されたバイトを含むブロックを無効にし、後続の参照でメイン・メモリーからブロックを取得します。

注: icbi 命令は、PowerPC ® アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0-5	31
6 ～ 10	///
11-15	RA
16-20	RB
21-30	982
31	/

PowerPC (R)

ICBI *RA*、 *RB*

説明

icbi 命令は、命令キャッシュでアドレス指定されたバイトを含むブロックを無効にします。 *RA* が 0 でない場合、 **icbi** 命令は、汎用レジスター (GPR) *RA* の内容を GPR *RB* の内容に追加することによって、有効アドレス (EA) を計算します。

icbi 命令を使用する際には、以下のことを考慮してください。

- マシン状態レジスター (MSR) のデータ再配置 (DR) ビットが 0 の場合、有効アドレスは実アドレスとして扱われます。
- MSR DR ビットが 1 の場合、有効アドレスは仮想アドレスとして扱われます。 この場合、MSR 再配置 (IR) ビットは無視されます。
- EA によってアドレス指定されたバイトを含むブロックが命令キャッシュ内にある場合、そのブロックは使用できなくなるため、ブロックへの次の参照はメイン・メモリーから取得されます。

icbi 命令には 1 つの構文形式があり、条件レジスター・フィールド 0 または固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードにより、変更された命令を実行できるようになります。

```
# Assume GPR 3 contains a modified instruction.
# Assume GPR 4 contains the address of the memory location
# where the modified instruction will be stored.
stw     3,0(4)          # Store the modified instruction.
dcbf    0,4             # Copy the modified instruction to
                        # main memory.
sync                    # Ensure update is in main memory.
icbi    0,4             # Invalidate block with old instruction.
isync                   # Discard prefetched instructions.
b       newcode         # Go execute the new code.
```

isync または ics (命令同期化) 命令

目的

この命令の前に取り出された可能性のある命令をすべて再フェッチします。

構文

ビット	VALUE
0-5	19
6 〜 10	///
11-15	///
16-20	///
21-30	150
31	/

PowerPC (R)

同期 (isync)

POWER® ファミリー

ics

説明

isync および **ics** 命令により、プロセッサーは **isync** または **ics** 命令の前にフェッチされた命令をすべて再フェッチします。

PowerPC® 命令 **isync** を使用すると、プロセッサーは前のすべての命令が完了するまで待機します。 その後、既に取り出された命令は破棄され、前の命令によって設定された環境で命令処理が続行されます。

POWER® ファミリー命令 **ics** を使用すると、プロセッサーは以前の **dcs** 命令が完了するのを待機します。 その後、既に取り出された命令は破棄され、マシン状態レジスターの内容によって設定された条件の下で命令処理が続行されます。

isync および **ics** 命令には 1 つの構文形式があり、条件レジスター・フィールド 0 または固定小数点例外レジスターには影響しません。

例

以下のコードは、続行する前に命令を再フェッチします。

```
# Assume GPR 5 holds name.
# Assume GPR 3 holds 0x0.
name: dcbf 3,5
isync
```

lbz (バイトおよびゼロのロード) 命令

目的

メモリー内の指定された位置から汎用レジスターに 1 バイトのデータをロードし、残りの 24 ビットを 0 に設定します。

構文

ビット	VALUE
0-5	34
6 〜 10	RT
11-15	RA
16-31	D

項目	説明
lbz (lbz)	*RT*、 *D*(*RA*)

説明

lbz 命令は、有効アドレス (EA) によってアドレス指定されたストレージ内の 1 バイトを、ターゲット汎用レジスター (GPR) *RT* のビット 24 から 31 にロードし、GPR *RT* のビット 0 から 23 を 0 に設定します。

RA が 0 でない場合、EA は GPR *RA* と *D*(16 ビット、符号付き 2 の補数整数符号-32 ビットに拡張) の内容の合計です。 *RA* が 0 の場合、EA は *D* です。

lbz 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
D	16 ビット、符号付き 2 の補数整数符号-EA 計算のために 32 ビットに拡張されます。
RA	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、メモリー内の指定された位置から 1 バイトのデータを GPR 6 にロードし、残りの 24 ビットを 0 に設定します。

```
.csect data[rw]
storage:  .byte 'a
```

```
# Assume GPR 5 contains the address of csect data[rw].
.csect text[pr]
lbz 6,storage(5)
# GPR 6 now contains 0x0000 0061.
```

関連概念
固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

lbzu (Load Byte and Zero with Update) 命令

目的

メモリー内の指定された位置から 1 バイトのデータを汎用レジスターにロードし、残りの 24 ビットを 0 に設定し、場合によってはアドレスを 2 番目の汎用レジスターに入れます。

構文

ビット	VALUE
0-5	35
6 ～ 10	RT
11-15	RA
16-31	D

項目	説明
LB ツー	*RT*、 *D*(*RA*)

説明

lbzu 命令は、有効アドレス (EA) によってアドレス指定されたストレージ内の 1 バイトをターゲット汎用レジスター (GPR) *RT* のビット 24 から 31 にロードし、GPR *RT* のビット 0 から 23 を 0 に設定します。

RA が 0 でない場合、EA は、32 ビットに拡張された 16 ビット符号付き 2 の補数整数符号である GPR *RA* と *D* の内容の合計です。 *RA* が 0 の場合、EA は *D* です。

RA が *RT* と等しくなく、 *RA* が 0 と等しくなく、ストレージ・アクセスによってアライメント割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA は GPR *RA* に保管されます。

lbzu 命令は、1 つの構文形式を持ち、固定小数点例外レジスターまたは条件レジスターのフィールド 0 には影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
D	16 ビット、符号付き 2 の補数整数符号-EA 計算のために 32 ビットに拡張されます。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。

例

以下のコードは、メモリー内の指定された位置から 1 バイトのデータを GPR 6 にロードし、残りの 24 ビットを 0 に設定し、そのアドレスを GPR 5 に入れます。

```
.csect data[rw]
storage:  .byte 0x61
# Assume GPR 5 contains the address of csect data[rw].
```

```
.csect text[pr]
lbzu 6,storage(5)
# GPR 6 now contains 0x0000 0061.
# GPR 5 now contains the storage address.
```

関連概念

固定小数点プロセッサー

固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管 (更新指示あり)

ロード・インストラクションとストア・インストラクションには更新フォームがあります。このフォームでは、メモリーとの間で通常の情報の移動に加えて、基本 GPR が EA で更新されます。

lbzux (Load Byte and Zero with Update Indexed) 命令

目的

メモリー内の指定された位置から 1 バイトのデータを汎用レジスターにロードし、残りの 24 ビットを 0 に設定し、そのアドレスを 2 番目の汎用レジスターに入れます。

構文

ビット	VALUE
0-5	31
6 〜 10	RT
11-15	RA
16-20	RB
21-30	119
31	/

項目	説明
LBZUX	*RT*、 *RA*、 *RB*

説明

lbzux 命令は、有効アドレス (EA) によってアドレス指定されたストレージ内の 1 バイトをターゲット汎用レジスター (GPR) *RT* のビット 24 から 31 にロードし、GPR *RT* のビット 0 から 23 を 0 に設定します。

RA が 0 でない場合、EA は GPR *RA* および GPR *RB* の内容の合計です。 *RA* が 0 の場合、EA は *RB* の内容です。

RA が *RT* と等しくなく、 *RA* が 0 と等しくなく、ストレージ・アクセスによってアライメント割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA は GPR *RA* に保管されます。

lbzux 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、 storage にある値を GPR 6 にロードし、 storage のアドレスを GPR 5 にロードします。

```
storage: .byte 0x40
  .
  .
# Assume GPR 5 contains 0x0000 0000.
# Assume GPR 4 is the storage address.
lbzux 6,5,4
# GPR 6 now contains 0x0000 0040.
# GPR 5 now contains the storage address.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管 (更新指示あり)
ロード・インストラクションとストア・インストラクションには更新フォームがあります。このフォームでは、メモリーとの間で通常の情報の移動に加えて、基本 GPR が EA で更新されます。

lbzx (Load Byte and Zero Indexed) 命令

目的

メモリー内の指定された位置から汎用レジスターに 1 バイトのデータをロードし、残りの 24 ビットを 0 に設定します。

構文

ビット	VALUE
0-5	31
6 〜 10	RT
11-15	RA
16-20	RB
21-30	87
31	/

項目	説明
lbzx (lbzx)	*RT*、 *RA*、 *RB*

説明

lbzx 命令は、有効アドレス (EA) によってアドレス指定されたストレージ内の 1 バイトをターゲット汎用レジスター (GPR) *RT* のビット 24 から 31 にロードし、GPR *RT* のビット 0 から 23 を 0 に設定します。

RA が 0 でない場合、EA は GPR *RA* および GPR *RB* の内容の合計です。 *RA* が 0 の場合、EA は *D* です。

lbzx 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。

項目	説明
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、 storage にある値を GPR 6 にロードします。

```
storage:  .byte 0x61
   .
   .
# Assume GPR 5 contains 0x0000 0000.
# Assume GPR 4 is the storage address.
lbzx 6,5,4
# GPR 6 now contains 0x0000 0061.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

ld (ダブルワードのロード) 命令

目的

指定された汎用レジスターにダブルワードのデータをロードします。

注 : この命令は、64 ビット・アプリケーションを実行する 64 ビット PowerPC プロセッサーでのみ使用してください。

構文

ビット	VALUE
0 - 5	58
6 - 10	RT
11 - 15	RA
16 - 29	DS
30 - 31	0b00

PowerPC 64

ld *RT*、 *Disp*(*RA*)

説明

ld 命令は、有効アドレス (EA) によってアドレス指定されたメモリー内の指定された位置から、ターゲット汎用レジスター (GPR) *RT* にダブルワードをロードします。

DS は 14 ビットの符号付き 2 の補数で、64 ビットに符号拡張され、さらに 4 を乗算して変位 *Disp* を提供します。 GPR *RA* が 0 でない場合、EA は GPR *RA* と *Disp* の内容の合計です。 GPR *RA* が 0 の場合、EA は *Disp* です。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
処理	4 の倍数である 16 ビットの符号付き数値を指定します。 アセンブラーは、命令の生成時にこの数値を 4 で除算します。
RA	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、メモリーから GPR 4 にダブルワードをロードします。

```
.extern  mydata[RW]
.csect   foodata[RW]
.local   foodata[RW]
storage: .llong mydata  # address of mydata

.csect   text[PR]
                        # Assume GPR 5 contains address of csect foodata[RW].
ld       4,storage(5)   # GPR 4 now contains the address of mydata.
```

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

ldarx (Load Doubleword Reserve Indexed) 命令

目的

ldarx 命令は、指定されたメモリー・ロケーションで読み取り/変更/書き込み操作をエミュレートするために、後続の **stdcx** 命令と一緒に使用されます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21 - 30	84
31	/

PowerPC64

ldarx (ldarx) *RT*、 *RA*、 *RB*

説明

ldarx および **stdcx** (**Store Doubleword Conditional Indexed**) 命令は、ストレージへの読み取り/変更/書き込み操作を実行するために使用されます。 保管操作が実行される場合、 **ldarx** 命令および **stdcx** 命令を使用することにより、 **ldarx** 命令が実行されてから **stdcx** 命令が完了するまでの間に、他のプロセッサーまたはメカニズムがターゲット・メモリーの位置を変更しないようにすることができます。

汎用レジスター (GPR) *RA* が 0 の場合、有効アドレス (EA) は GPR *RB* の内容になります。 それ以外の場合、EA は GPR *RA* の内容と GPR *RB* の内容の合計です。

ldarx 命令は、EA によって指定されたストレージ内の位置からターゲット GPR *RT* にワードをロードします。 さらに、後続の **stwcx** で使用するために、メモリー・ロケーションの予約が作成されます。 命令。

ldarx 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。 EA が 8 の倍数でない場合、システム位置合わせハンドラーが呼び出されるか、結果が未定義と呼ばれます。

パラメーター

項目	説明
RT	保管データのソース GPR を指定します。
RA	EA 計算のソース GPR を指定します。
rb	EA 計算のソース GPR を指定します。

例

1. 以下のコードは、ストレージ内のワードをアトミックにロードして置き換えることにより、フェッチ操作と保管操作を実行します。

```
# Assume that GPR 4 contains the new value to be stored.
# Assume that GPR 3 contains the address of the word
# to be loaded and replaced.
loop:   lwarx   r5,0,r3          # Load and reserve
        stwcx.  r4,0,r3          # Store new value if still
                                 # reserved
        bne-    loop             # Loop if lost reservation
# The new value is now in storage.
# The old value is returned to GPR 4.
```

2. 以下のコードは、レジスター内の値をストレージ内のワードとアトミックに比較することにより、比較およびスワップ操作を実行します。

```
# Assume that GPR 5 contains the new value to be stored after
# a successful match.
# Assume that GPR 3 contains the address of the word
# to be tested.
# Assume that GPR 4 contains the value to be compared against
# the value in memory.
loop:   lwarx   r6,0,r3          # Load and reserve
        cmpw    r4,r6            # Are the first two operands
                                 # equal?
        bne-    exit             # Skip if not equal
        stwcx.  r5,0,r3          # Store new value if still
                                 # reserved
        bne-    loop             # Loop if lost reservation
exit:   mr      r4,r6            # Return value from storage
# The old value is returned to GPR 4.
# If a match was made, storage contains the new value.
```

   レジスターの値がストレージ内のワードと等しい場合は、2 番目のレジスターの値がストレージ内のワードに保管されます。 それらが等しくない場合は、ストレージからのワードが最初のレジスターにロードされ、条件レジスター・フィールド 0 の EQ ビットが比較の結果を示すように設定されます。

ldu (更新付きダブルワードのロード) 命令

目的

指定された汎用レジスター (GPR) にダブルワードのデータをロードし、アドレス・ベースを更新します。

注 : この命令は、64 ビット・アプリケーションを実行する 64 ビット PowerPC プロセッサーでのみ使用してください。

構文

ビット	VALUE
0 - 5	58
6 - 10	RT
11 - 15	RA
16 - 29	DS
30 - 31	0b01

PowerPC 64

ldu (ldu) *RT*、 *Disp*(*RA*)

説明

ldu 命令は、有効アドレス (EA) によってアドレス指定されたメモリー内の指定された位置から、ストレージ内のダブルワードをターゲット GPR *RT* にロードします。

DS は 14 ビットの符号付き 2 の補数で、64 ビットに符号拡張され、さらに 4 を乗算して変位 *(Disp)*を提供します。 GPR *RA* が 0 でない場合、EA は GPR *RA* と *Disp* の内容の合計です。

RA が 0 または *RA* が *RT* の場合、命令形式は無効です。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット GPR を指定します。
処理	4 の倍数である 16 ビットの符号付き数値を指定します。 アセンブラーは、命令の生成時にこの数値を 4 で除算します。
RA	EA 計算のソース GPR を指定します。

例

以下のコードは、4 つのダブルワードのうち最初のワードをメモリーから GPR 4 にロードし、メモリー内の次のダブルワードを指すように GPR 5 に増分します。

```
.csect   foodata[RW]
storage: .llong 5,6,7,12  # Successive doublewords.

.csect   text[PR]
                          # Assume GPR 5 contains address of csect foodata[RW].
ldu       4,storage(5)    # GPR 4 now contains the first doubleword of
                          # foodata; GRP 5 points to the second doubleword.
```

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット・インプリメンテーションで使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管 (更新指示あり)

ロード・インストラクションとストア・インストラクションには更新フォームがあります。このフォームでは、メモリーとの間で通常の情報の移動に加えて、基本 GPR が EA で更新されます。

ldux (Update Indexed によるダブルワードのロード) 命令

目的

指定されたメモリー位置からデータのダブルワードを汎用レジスター (GPR) にロードし、アドレス・ベースを更新します。

構文

ビット	VALUE
0 - 5	31
6 - 10	D
11 - 15	A
16 - 20	B
21 - 30	53
31	0

PowerPC (R)

ldux *RT*、 *RA*、 *RB*

説明

有効アドレス (EA) は、GPR、 *RA* 、および *RB* の合計から計算されます。 データのダブルワードは、EA によって参照されるメモリー位置から読み取られ、GPR *RT* に入れられます。 GPR *RA* が EA で更新されました。

RA が 0 または RA が RD と等しい場合、命令フォームは無効です。

パラメーター

項目	説明
RT	保管データのソース GPR を指定します。
RA	EA 計算のソース GPR を指定します。
rb	EA 計算のソース GPR を指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット・インプリメンテーションで使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

ldx (ダブルワード索引付きロード) 命令

目的

指定されたメモリー位置から汎用レジスター (GPR) にダブルワードをロードします。

構文

ビット	VALUE
0 - 5	31
6 - 10	D
11 - 15	A
16 - 20	B
21 - 30	21
31	0

PowerPC (R)

LDX *RT RA*、 *RB*

説明

ldx 命令は、実効アドレス (EA) によって参照される指定のメモリー位置から GPR *RT* にダブルワードをロードします。

GRP *RA* が 0 でない場合、有効アドレス (EA) は、GRP、 *RA* 、および *RB* の内容の合計です。 それ以外の場合、EA は *RB* の内容と等しくなります。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット GPR を指定します。
RA	EA 計算のソース GPR を指定します。
rb	EA 計算のソース GPR を指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット・インプリメンテーションで使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

lfd (Load Floating-Point Double) 命令

目的

メモリー内の指定された位置からデータのダブルワードを浮動小数点レジスターにロードします。

構文

ビット	VALUE
0 - 5	50
6 - 10	FRT (R)
11 - 15	RA
16 - 31	D

項目	説明
lfd (lfd)	*FRT*、 *D*(*RA*)

説明

lfd 命令は、有効アドレス (EA) によってアドレス指定されたメモリー内の指定された位置から、ストレージ内のダブルワードをターゲット浮動小数点レジスター (FPR) *FRT* にロードします。

汎用レジスター (GPR) *RA* が 0 でない場合、EA は、GPR *RA* と *D*(16 ビット、符号付き 2 の補数整数符号-32 ビットに拡張) の内容の合計です。 GPR *RA* が 0 の場合、EA は *D* です。

lfd 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
FRT (FRT)	操作の結果が保管されるターゲット汎用レジスターを指定します。
D	16 ビット、符号付き 2 の補数整数符号-EA 計算のために 32 ビットに拡張されます。
RA	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、メモリーから FPR 6 にダブルワードをロードします。

```
.csect data[rw]
storage: .double 0x1
# Assume GPR 5 contains the address of csect data[rw].
.csect text[pr]
lfd 6,storage(5)
# FPR 6 now contains 0x3FF0 0000 0000 0000.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

lfdu (更新を伴う Load Floating-Point Double) 命令

目的

メモリー内の指定された位置からデータのダブルワードを浮動小数点レジスターにロードし、指定されたアドレスを汎用レジスターに入れることができます。

構文

ビット	VALUE
0 - 5	51
6 - 10	FRT (R)
11 - 15	RA
16 - 31	D

項目	説明
lfdu (lfdu)	*FRT*、 *D*(*RA*)

説明

lfdu 命令は、実効アドレス (EA) によってアドレス指定されたメモリー内の指定された位置から、ストレージ内のダブルワードをターゲット浮動小数点レジスター (FPR) *FRT* にロードします。

RA が 0 でない場合、EA は GPR *RA* と *D*(16 ビット、符号付き 2 の補数整数符号-32 ビットに拡張) の内容の合計です。 *RA* が 0 の場合、有効アドレス (EA) は *D* です。

RA が 0 でなく、ストレージ・アクセスによって位置合わせ割り込みまたはデータ・ストレージ割り込みが発生しない場合、有効アドレスは GPR *RA* に保管されます。

lfdu 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
FRT (FRT)	操作の結果が保管されるターゲット汎用レジスターを指定します。
D	EA 計算用に 16 ビットの符号付き 2 の補数整数符号拡張を 32 ビットに指定します。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。

例

以下のコードは、メモリーから FPR 6 にダブルワードをロードし、そのアドレスを GPR 5 に保管します。

```
.csect data[rw]
storage: .double 0x1
# Assume GPR 5 contains the address of csect data[rw].
.csect text[pr]
lfdu 6,storage(5)
# FPR 6 now contains 0x3FF0 0000 0000 0000.
# GPR 5 now contains the storage address.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

lfdux (Update Indexed を使用した Load Floating-Point Double) 命令

目的

メモリー内の指定された位置からデータのダブルワードを浮動小数点レジスターにロードし、指定されたアドレスを汎用レジスターに入れることができます。

構文

ビット	VALUE
0 - 5	31
6 - 10	FRT (R)
11 -15	RA
16 - 20	RB
21 - 30	631

ビット	VALUE
31	/

項目	説明
lfdux	*FRT*、 *RA*、 *RB*

説明

lfdux 命令は、実効アドレス (EA) によってアドレス指定されたメモリー内の指定された位置から、ストレージ内のダブルワードをターゲット浮動小数点レジスター (FPR) *FRT* にロードします。

RA が 0 でない場合、EA は、汎用レジスター (GPR) *RA* と GPR *RB* の内容の合計です。 *RA* が 0 の場合、EA は *RB* の内容です。

RA が 0 ではなく、ストレージ・アクセスによってアライメント割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA は GPR RA に保管されます。

lfdux 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターには影響しません。

パラメーター

項目	説明
FRT (FRT)	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、メモリーから FPR 6 にダブルワードをロードし、そのアドレスを GPR 5 に保管します。

```
.csect data[rw]
storage: .double 0x1
# Assume GPR 5 contains the address of csect data[rw].
# Assume GPR 4 contains the displacement of storage relative
# to .csect data[rw].
.csect text[pr]
lfdux 6,5,4
# FPR 6 now contains 0x3FF0 0000 0000 0000.
# GPR 5 now contains the storage address.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

lfdx (Load Floating-Point Double-Indexed) 命令

目的

メモリー内の指定された位置からデータのダブルワードを浮動小数点レジスターにロードします。

構文

ビット	VALUE
0 - 5	31
6 - 10	FRT (R)
11 - 15	RA
16 - 20	RB
21 - 30	599
31	/

項目	説明
lfdx (lfdx)	*FRT*、 *RA*、 *RB*

説明

lfdx 命令は、有効アドレス (EA) によってアドレス指定されたメモリー内の指定された位置から、ストレージ内のダブルワードをターゲット浮動小数点レジスター (FPR) *FRT* にロードします。

RA が 0 でない場合、EA は、汎用レジスター (GPR) *RA* と GPR *RB* の内容の合計です。 *RA* が 0 の場合、EA は GPR *RB* の内容です。

lfdx 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
FRT (FRT)	データが保管されるターゲット浮動小数点レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、メモリーから FPR 6 にダブルワードをロードします。

```
storage: .double 0x1
   .
   .
# Assume GPR 4 contains the storage address.
lfdx 6,0,4
# FPR 6 now contains 0x3FF0 0000 0000 0000.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

lfq (浮動小数点クワッドのロード) 命令

目的

2 つの倍精度値を浮動小数点レジスターにロードします。

注: lfq 命令は、POWER® ファミリー・アーキテクチャーの POWER2™ 実装でのみサポートされます。

構文

ビット	VALUE
0 - 5	56
6 - 10	FRT (R)
11 - 15	RA
16 - 29	DS
30 - 31	00

POWER2™

lfq (lfq) *FRT*、 *DS*(*RA*)

説明

lfq 命令は、有効アドレス (EA) によって指定されたメモリー内の位置から 2 つのダブルワードを 2 つの浮動小数点レジスター (FPR) にロードします。

DS は 30 ビットまで符号拡張され、右側に b '00' が連結されてオフセット値を形成します。 汎用レジスター (GPR) *RA* が 0 の場合、オフセット値は EA になります。 GPR *RA* が 0 でない場合は、オフセット値が GPR *RA* に追加され、EA が生成されます。 EA のダブルワードは FPR *FRT* にロードされます。 *FRT* が 31 の場合、 EA+8 のダブルワードは FPR 0 にロードされます。それ以外の場合は、 *FRT*+ 1 にロードされます。

lfq 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
FRT (FRT)	2 つのターゲット浮動小数点レジスターの最初のレジスターを指定します。
DS	EA 計算の即時値として使用される 14 ビット・フィールドを指定します。
RA	EA 計算用の 1 つのソース汎用レジスターを指定します。

例

以下のコードは、2 つの倍精度浮動小数点値を、メモリー内の 1 つの場所からメモリー内の 2 番目の場所にコピーします。

```
# Assume GPR 3 contains the address of the first source
# floating-point value.
# Assume GPR 4 contains the address of the target location.
lfq     7,0(3)          # Load first two values into FPRs 7 and
                        # 8.
stfq    7,0(4)          # Store the two doublewords at the new
                        # location.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

lfqu (更新付き浮動小数点クワッド・ロード) 命令

目的

2 つの倍精度値を浮動小数点レジスターにロードし、アドレス基数を更新します。

> **注: lf 先** 命令は、POWER® ファミリー・アーキテクチャーの POWER2™ 実装でのみサポートされます。

構文

ビット	VALUE
0 - 5	57
6 - 10	FRT (R)
11 - 15	RA
16 - 29	DS
30 - 31	00

POWER2™

LF 屈曲 *FRT*、 *DS*(*RA*)

説明

lfqu 命令は、有効アドレス (EA) によって指定されたメモリー内の位置から 2 つのダブルワードを 2 つの浮動小数点レジスター (FPR) にロードします。

DS は 30 ビットまで符号拡張され、右側に b '00' が連結されてオフセット値を形成します。 汎用レジスター GPR *RA* が 0 の場合、オフセット値は EA になります。 GPR *RA* が 0 でない場合は、オフセット値が GPR *RA* に追加され、EA が生成されます。 EA のダブルワードは FPR *FRT* にロードされます。 *FRT* が 31 の場合、 EA+8 のダブルワードは FPR 0 にロードされます。それ以外の場合は、 *FRT*+ 1 にロードされます。

GPR *RA* が 0 でない場合、EA は GPR *RA* に置かれます。

lf 先 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
FRT (FRT)	2 つのターゲット浮動小数点レジスターの最初のレジスターを指定します。
DS	EA 計算の即時値として使用される 14 ビット・フィールドを指定します。
RA	EA 計算用に 1 つのソース汎用レジスターを指定し、EA 更新用にターゲット・レジスターを指定します。

例

以下のコードは、メモリー内の連続するダブルワードにある 6 つの倍精度浮動小数点値の合計を計算します。

```
# Assume GPR 3 contains the address of the first
# floating-point value.
# Assume GPR 4 contains the address of the target location.
lfq     7,0(3)          # Load first two values into FPRs 7 and
                        # 8.
lfqu    9,16(3)         # Load next two values into FPRs 9 and 10
                        # and update base address in GPR 3.
fadd    6,7,8           # Add first two values.
lfq     7,16(3)         # Load next two values into FPRs 7 and 8.
fadd    6,6,9           # Add third value.
fadd    6,6,10          # Add fourth value.
fadd    6,6,7           # Add fifth value.
fadd    6,6,8           # Add sixth value.
stfqx   7,0,4           # Store the two doublewords at the new
                        # location.
```

関連概念

浮動小数点プロセッサー

浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

lfqux (索引付き更新付き浮動小数点クワッド・ロード) 命令

目的

2 つの倍精度値を浮動小数点レジスターにロードし、アドレス基数を更新します。

> **注: lfqux** 命令は、POWER® ファミリー・アーキテクチャーの POWER2™ 実装環境でのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	FRT (R)
11 - 15	RA
16 - 20	RB
21 - 30	823
31	rc

POWER2™

lfqux (lfqux) *FRT*、 *RA*、 *RB*

説明

lfqux 命令は、有効アドレス (EA) によって指定されたメモリー内の位置から 2 つのダブルワードを 2 つの浮動小数点レジスター (FPR) にロードします。

汎用レジスター (GPR) *RA* が 0 でない場合、EA は GPR *RA* および GPR *RB* の内容の合計になります。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。 EA のダブルワードは FPR *FRT* にロードされます。 *FRT* が 31 の場合、 EA+8 のダブルワードは FPR 0 にロードされます。それ以外の場合は、 *FRT*+ 1 にロードされます。

GPR *RA* が 0 でない場合、EA は GPR *RA* に置かれます。

lfqux 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスターのフィールド 0 には影響しません。

パラメーター

項目	説明
FRT (FRT)	2 つのターゲット浮動小数点レジスターの最初のレジスターを指定します。
RA	EA 計算用の最初のソース汎用レジスターと EA 更新用のターゲット・レジスターを指定します。
rb	EA 計算の 2 番目のソース汎用レジスターを指定します。

例

次のコードは、3 つの倍精度、浮動小数点、2 次元座標の合計を計算します。

```
# Assume the two-dimensional coordinates are contained
# in a linked list with elements of the form:
# list_element:
#       .double          # Floating-point value of X.
#       .double          # Floating-point value of Y.
#       .next_elem       # Offset to next element;
#                        # from X(n) to X(n+1).
#
# Assume GPR 3 contains the address of the first list element.
# Assume GPR 4 contains the address where the resultant sums
# will be stored.
lfq     7,0(3)           # Get first pair of X_Y values.
lwz     5,16(3)          # Get the offset to second element.
lfqux   9,3,5            # Get second pair of X_Y values.
lwz     5,16(3)          # Get the offset to third element.
fadd    7,7,9            # Add first two X values.
fadd    8,8,10           # Add first two Y values.
lfqux   9,3,5            # Get third pair of X_Y values.
fadd    7,7,9            # Add third X value to sum.
fadd    8,8,10           # Add third Y value to sum.
stfq    7,0,4            # Store the two doubleword results.
```

関連概念

浮動小数点プロセッサー

浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

lfqx (Load Floating-Point Quad Indexed) 命令

目的

2 つの倍精度値を浮動小数点レジスターにロードします。

> **注: lfqx** 命令は、POWER® ファミリー・アーキテクチャーの POWER2™ 実装でのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	FRT (R)
11 - 15	RA

ビット	VALUE
16 - 20	RB
21 - 30	791
31	rc

POWER2™

lfqx (lfqx) *FRT*、 *RA*、 *RB*

説明

lfqx 命令は、有効アドレス (EA) によって指定されたメモリー内の位置から 2 つのダブルワードを 2 つの浮動小数点レジスター (FPR) にロードします。

汎用レジスター (GPR) *RA* が 0 でない場合、EA は GPR *RA* および GPR *RB* の内容の合計になります。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。 EA のダブルワードは FPR *FRT* にロードされます。 *FRT* が 31 の場合、 EA+8 のダブルワードは FPR 0 にロードされます。それ以外の場合は、 *FRT*+ 1 にロードされます。

lfqx 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
FRT (FRT)	2 つのターゲット浮動小数点レジスターの最初のレジスターを指定します。
RA	EA 計算用の 1 つのソース汎用レジスターを指定します。
rb	EA 計算の 2 番目のソース汎用レジスターを指定します。

例

以下のコードは、メモリー内の連続ダブルワードにある 2 つの倍精度浮動小数点値の合計を計算します。

```
# Assume GPR 3 contains the address of the first floating-point
# value.
# Assume GPR 4 contains the address of the target location.
lfqx    7,0,3            # Load values into FPRs 7 and 8.
fadd    7,7,8            # Add the two values.
stfdx   7,0,4            # Store the doubleword result.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

lfs (浮動小数点単一ロード) 命令

目的

浮動小数点数、倍精度数値に変換された単精度の浮動小数点数を浮動小数点数レジスターにロードします。

構文

ビット	VALUE
0 - 5	48
6 - 10	FRT (R)
11 - 15	RA
16 - 31	D

項目	説明
lfs	*FRT*、 *D*(*RA*)

説明

lfs 命令は、有効アドレス (EA) によってアドレス指定されたストレージ内の浮動小数点の単精度ワードを浮動小数点の倍精度ワードに変換し、その結果を浮動小数点レジスター (FPR) *FRT* にロードします。

RA が 0 でない場合、EA は GPR *RA* と *D*(16 ビット、符号付き 2 の補数整数符号-32 ビットに拡張) の内容の合計です。 *RA* が 0 の場合、EA は *D* です。

lfs 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスターのフィールド 0 には影響しません。

パラメーター

項目	説明
FRT (FRT)	データが保管されるターゲット浮動小数点レジスターを指定します。
D	16 ビット、符号付き 2 の補数整数符号-EA 計算のために 32 ビットに拡張されます。
RA	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、ストレージの単精度内容を FPR 6 にロードします。

```
.csect data[rw]
storage: .float 0x1
# Assume GPR 5 contains the address csect data[rw].
.csect text[pr]
lfs 6,storage(5)
# FPR 6 now contains 0x3FF0 0000 0000 0000.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

lfsu (Load Floating-Point Single with Update) 命令

目的

浮動小数点、倍精度の数値に変換された単精度の浮動小数点数を浮動小数点レジスターにロードし、場合によっては有効アドレスを汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	49
6 - 10	FRT (R)
11 - 15	RA
16 - 31	D

項目	説明
lfsu	*FRT*、 *D*(*RA*)

説明

lfsu 命令は、有効アドレス (EA) によってアドレス指定されたストレージ内の浮動小数点単精度ワードを浮動小数点倍精度ワードに変換し、その結果を浮動小数点レジスター (FPR) *FRT* にロードします。

RA が 0 でない場合、EA は、32 ビットに拡張された 16 ビット符号付き 2 の補数整数符号である汎用レジスター (GPR) *RA* および *D* の内容の合計です。 *RA* が 0 の場合、EA は *D* です。

RA が 0 に等しくなく、ストレージ・アクセスによってアライメント割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA は GPR *RA* に保管されます。

lfsu 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
FRT (FRT)	データが保管されるターゲット浮動小数点レジスターを指定します。
D	16 ビット、符号付き 2 の補数整数符号-EA 計算のために 32 ビットに拡張されます。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。

例

以下のコードは、倍精度に変換されるストレージの単精度の内容を FPR 6 にロードし、有効アドレスを GPR 5 に保管します。

```
.csect data[rw]
storage: .float 0x1
.csect text[pr]
# Assume GPR 5 contains the storage address.
lfsu 6,0(5)
# FPR 6 now contains 0x3FF0 0000 0000 0000.
# GPR 5 now contains the storage address.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

lfsux (Load Floating-Point Single with Update Indexed) 命令

目的

浮動小数点、倍精度の数値に変換された単精度の浮動小数点数を浮動小数点レジスターにロードし、場合によっては有効アドレスを汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	FRT (R)
11 - 15	RA
16 - 20	RB
21 - 30	567
31	/

項目	説明
lfsux (lfsux)	*FRT*、 *RA*、 *RB*

説明

lfsux 命令は、有効アドレス (EA) によってアドレス指定されたストレージ内の浮動小数点の単精度ワードを浮動小数点の倍精度ワードに変換し、その結果を浮動小数点レジスター (FPR) *FRT* にロードします。

汎用レジスター (GPR) *RA* が 0 でない場合、EA は GPR *RA* および GPR *RB* の内容の合計になります。 *RA* が 0 の場合、EA は GPR *RB* の内容です。

GPR *RA* が 0 ではなく、ストレージ・アクセスによってアライメント割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA は GPR *RA* に保管されます。

lfsux 命令には 1 つの構文形式があり、浮動小数点状況制御レジスターには影響しません。

パラメーター

項目	説明
FRT (FRT)	データが保管されるターゲット浮動小数点レジスターを指定します。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、ストレージの単精度の内容を FPR 6 にロードし、有効アドレスを GPR 5 に保管します。

```
.csect data[rw]
storage: .float 0x1
# Assume GPR 4 contains the address of csect data[rw].
# Assume GPR 5 contains the displacement of storage
# relative to .csect data[rw].
.csect text[pr]
lfsux 6,5,4
# FPR 6 now contains 0x3FF0 0000 0000 0000.
# GPR 5 now contains the storage address.
```

関連概念

浮動小数点プロセッサー

浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

lfsx (Load Floating-Point Single Indexed) 命令

目的

浮動小数点数、倍精度数値に変換された単精度の浮動小数点数を浮動小数点数レジスターにロードします。

構文

ビット	VALUE
0 - 5	31
6 - 10	FRT (R)
11 - 15	RA
16 - 20	RB
21 - 30	535
31	/

項目	説明
lfsx (lfsx)	*FRT*、 *RA*、 *RB*

説明

lfsx 命令は、有効アドレス (EA) によってアドレス指定されたストレージ内の浮動小数点単精度ワードを浮動小数点倍精度ワードに変換し、その結果を浮動小数点レジスター (FPR) *FRT* にロードします。

汎用レジスター (GPR) *RA* が 0 でない場合、EA は GPR *RA* および GPR *RB* の内容の合計になります。 *RA* が 0 の場合、EA は GPR *RB* の内容です。

lfsx 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターには影響しません。

パラメーター

項目	説明
FRT (FRT)	データが保管されるターゲット浮動小数点レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、ストレージの単精度内容を FPR 6 にロードします。

```
storage: .float 0x1.
# Assume GPR 4 contains the address of storage.
lfsx 6,0,4
# FPR 6 now contains 0x3FF0 0000 0000 0000.
```

関連概念

浮動小数点プロセッサー

浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

lha (Load Half Algebraic) 命令

目的

メモリー内の指定された位置からデータのハーフワードを汎用レジスターにロードし、そのハーフワードのビット 0 を汎用レジスターの残りの 16 ビットにコピーします。

構文

ビット	VALUE
0 - 5	42
6 - 10	RT
11 - 15	RA
16 - 31	D

項目	説明
lha (lha)	*RT*、 *D*(*RA*)

説明

lha 命令は、メモリー内の指定された場所から、実効アドレス (EA) によってアドレス指定されたデータのハーフワードを、ターゲット汎用レジスター (GPR) *RT* のビット 16 から 31 にロードし、そのハーフワードのビット 0 から 15 を GPR *RT* のビット 0 から 15 にコピーします。

GPR *RA* が 0 でない場合、EA は、32 ビットに拡張された 16 ビット符号付き 2 の補数整数符号である GPR *RA* と *D* の内容の合計です。 GPR *RA* が 0 の場合、EA は *D* です。

lha 命令は、構文形式が 1 つあり、固定小数点例外レジスターまたは条件レジスターのフィールド 0 には影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
D	16 ビット、符号付き 2 の補数整数符号-EA 計算のために 32 ビットに拡張されます。
RA	EA 計算用のソース汎用レジスターを指定します。

例

次のコードは、データのハーフワードを GPR 6 のビット 16 から 31 にロードし、そのハーフワードのビット 0 を GPR 6 のビット 0 から 15 にコピーします。

```
.csect data[rw]
storage: .short 0xffff
# Assume GPR 5 contains the address of csect data[rw].
.csect text[pr]
lha 6,storage(5)
# GPR 6 now contains 0xffff ffff.
```

関連概念

固定小数点プロセッサー

固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

lhau (Update による半分の Algebraic のロード) 命令

目的

メモリー内の指定された位置からデータのハーフワードを汎用レジスターにロードし、そのハーフワードのビット 0 を汎用レジスターの残りの 16 ビットにコピーし、アドレスを別の汎用レジスターに入れることができます。

構文

ビット	VALUE
0 - 5	43
6 - 10	RT
11 - 15	RA
16 - 31	D

項目	説明
lhau (lhau)	*RT*、 *D*(*RA*)

説明

lhau 命令は、メモリー内の指定された位置から、有効アドレス (EA) によってアドレス指定されたデータのハーフワードをターゲット汎用レジスター (GPR) *RT* のビット 16 から 31 にロードし、そのハーフワードのビット 0 から 15 を GPR *RT* のビット 0 から 15 にコピーします。

GPR *RA* が 0 でない場合、EA は、GPR *RA* と *D*(16 ビット、符号付き 2 の補数整数符号-32 ビットに拡張) の内容の合計です。 GPR *RA* が 0 の場合、EA は *D* です。

RA が *RT* と等しくなく、 *RA* が 0 と等しくなく、ストレージ・アクセスによってアライメント割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA は GPR *RA* に入れられます。

lhau 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
D	16 ビット、符号付き 2 の補数整数符号-EA 計算のために 32 ビットに拡張されます。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。

例

以下のコードは、データのハーフワードを GPR 6 のビット 16 から 31 にロードし、ハーフワードのビット 0 を GPR 6 のビット 0 から 15 にコピーし、有効アドレスを GPR 5 に保管します。

```
.csect data[rw]
storage: .short 0xffff
# Assume GPR 5 contains the address of csect data[rw].
.csect text[pr]
```

```
lhau 6,storage(5)
# GPR 6 now contains 0xffff ffff.
# GPR 5 now contains the address of storage.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管 (更新指示あり)
ロード・インストラクションとストア・インストラクションには更新フォームがあります。このフォームでは、メモリーとの間で通常の情報の移動に加えて、基本 GPR が EA で更新されます。

lhaux (Update Indexed を使用した Load Half Algebraic) 命令

目的

メモリー内の指定された位置からデータのハーフワードを汎用レジスターにロードし、そのハーフワードのビット 0 を汎用レジスターの残りの 16 ビットにコピーし、アドレスを別の汎用レジスターに入れることができます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21 - 30	375
31	/

項目	説明
lhaux	*RT*、 *RA*、 *RB*

説明

lhaux 命令は、有効アドレス (EA) によってアドレス指定されたメモリー内の指定された場所からターゲット汎用レジスター (GPR) *RT* のビット 16 から 31 にデータのハーフワードをロードし、そのハーフワードのビット 0 から 15 を GPR *RT* のビット 0 から 15 にコピーします。

GPR *RA* が 0 でない場合、EA は GPR *RA* と GPR *RB* の内容の合計です。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

RA が *RT* と等しくなく、 *RA* が 0 と等しくなく、ストレージ・アクセスによってアライメント割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA は GPR *RA* に入れられます。

lhaux 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算および可能なアドレス更新のための最初のソース汎用レジスターを指定します。
rb	EA 計算用の 2 番目のソース汎用レジスターを指定します。

例

以下のコードは、データのハーフワードを GPR 6 のビット 16 から 31 にロードし、ハーフワードのビット 0 を GPR 6 のビット 0 から 15 にコピーし、有効アドレスを GPR 5 に保管します。

```
.csect data[rw]
storage: .short 0xffff
# Assume GPR 5 contains the address of csect data[rw].
# Assume GPR 4 contains the displacement of storage relative
# to data[rw].
.csect text[pr]
lhaux 6,5,4
# GPR 6 now contains 0xffff ffff.
# GPR 5 now contains the storage address.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管 (更新指示あり)
ロード・インストラクションとストア・インストラクションには更新フォームがあります。このフォームでは、メモリーとの間で通常の情報の移動に加えて、基本 GPR が EA で更新されます。

lhax (ロード・ハーフ代数索引付き) 命令

目的

メモリー内の指定された位置からデータのハーフワードを汎用レジスターにロードし、そのハーフワードのビット 0 を汎用レジスターの残りの 16 ビットにコピーします。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21 - 30	343
31	/

項目	説明
lhax	*RT*、 *RA*、 *RB*

説明

lhax 命令は、メモリー内の指定された場所から、有効アドレス (EA) によってアドレス指定されたデータのハーフワードをターゲット汎用レジスター (GPR) *RT* のビット 16 から 31 にロードし、そのハーフワードのビット 0 から 15 を GPR *RT* のビット 0 から 15 にコピーします。

GPR *RA* が 0 でない場合、EA は GPR *RA* と GPR *RB* の内容の合計です。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

lhax 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

次のコードは、データのハーフワードを GPR 6 のビット 16 から 31 にロードし、そのハーフワードのビット 0 を GPR 6 のビット 0 から 15 にコピーします。

```
.csect data[rw]
.short 0x1
# Assume GPR 5 contains the address of csect data[rw].
# Assume GPR 4 contains the displacement of the halfword
# relative to data[rw].
.csect text[pr]
lhax 6,5,4
# GPR 6 now contains 0x0000 0001.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

lhbrx (Load Half Byte-Reverse Indexed) 命令

目的

メモリー内の指定された位置からデータのバイト反転ハーフワードを汎用レジスターにロードし、汎用レジスターの残りの 16 ビットをゼロに設定します。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21 - 30	790
31	/

項目	説明
lhbrx (lhbrx)	*RT*、 *RA*、 *RB*

説明

lhbrx 命令は、有効アドレス (EA) によってアドレス指定されたストレージ内のハーフワードのビット 00 から 07 およびビット 08 から 15 を汎用レジスター (GPR) *RT* のビット 24 から 31 およびビット 16 から 23 にロードし、GPR *RT* のビット 00 から 15 を 0 に設定します。

GPR *RA* が 0 でない場合、EA は GPR *RA* と GPR *RB* の内容の合計です。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

lhbrx 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、ストレージ内のハーフワードのビット 00 から 07 およびビット 08 から 15 を GPR 6 のビット 24 から 31 およびビット 16 から 23 にロードし、GPR 6 のビット 00 から 15 を 0 に設定します。

```
.csect data[rw]
.short 0x7654
# Assume GPR 4 contains the address of csect data[rw].
# Assume GPR 5 contains the displacement relative
# to data[rw].
.csect text[pr]
lhbrx 6,5,4
# GPR 6 now contains 0x0000 5476.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

lhz (ロード・ハーフおよびゼロ) 命令

目的

メモリー内の指定された位置から汎用レジスターにデータのハーフワードをロードし、残りの 16 ビットを 0 に設定します。

構文

ビット	VALUE
0 - 5	40
6 - 10	RT
11 - 15	RA
16 - 31	D

項目	説明
lhz (lhz)	*RT*、 *D*(*RA*)

説明

lhz 命令は、メモリー内の指定された場所から、有効アドレス (EA) によってアドレス指定されたデータのハーフワードをターゲット汎用レジスター (GPR) *RT* のビット 16 から 31 にロードし、GPR *RT* のビット 0 から 15 を 0 に設定します。

GPR *RA* が 0 でない場合、EA は、GPR *RA* と *D*(16 ビット、符号付き 2 の補数整数符号-32 ビットに拡張) の内容の合計です。 GPR *RA* が 0 の場合、EA は *D* です。

lhz 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
D	16 ビット、符号付き 2 の補数整数符号-EA 計算のために 32 ビットに拡張されます。
RA	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、データのハーフワードを GPR 6 のビット 16 から 31 にロードし、GPR 6 のビット 0 から 15 を 0 に設定します。

```
.csect data[rw]
storage: .short 0xffff
# Assume GPR 4 holds the address of csect data[rw].
.csect text[pr]
lhz 6,storage(4)
# GPR 6 now holds 0x0000 ffff.
```

関連概念

固定小数点ロードおよび保管の指示

固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

lhzu (更新による半分とゼロのロード) 命令

目的

メモリー内の指定された位置から汎用レジスターにデータのハーフワードをロードし、汎用レジスターの残りの 16 ビットを 0 に設定し、場合によってはアドレスを別の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	41
6 - 10	RT
11 - 15	RA
16 - 31	D

項目	説明
lhzu (lhzu)	*RT*、 *D*(*RA*)

説明

lhzu 命令は、メモリー内の指定された場所 (有効アドレス (EA) によってアドレス指定された) から、ターゲット汎用レジスター (GPR) *RT* のビット 16 から 31 にハーフワードのデータをロードし、GPR *RT* のビット 0 から 15 を 0 に設定します。

GPR *RA* が 0 でない場合、EA は、GPR *RA* と *D*(16 ビット、符号付き 2 の補数整数符号-32 ビットに拡張) の内容の合計です。 GPR *RA* が 0 の場合、EA は *D* です。

RA が *RT* と等しくなく、 *RA* が 0 と等しくなく、ストレージ・アクセスによってアライメント割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA は GPR *RA* に入れられます。

lhzu 命令は、1 つの構文形式を持ち、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
D	16 ビット、符号付き 2 の補数整数符号-EA 計算のために 32 ビットに拡張されます。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。

例

以下のコードは、データのハーフワードを GPR 6 のビット 16 から 31 にロードし、GPR 6 のビット 0 から 15 を 0 に設定し、有効アドレスを GPR 4 に保管します。

```
.csect data[rw]
.short 0xffff
# Assume GPR 4 contains the address of csect data[rw].
.csect text[pr]
lhzu 6,0(4)
# GPR 6 now contains 0x0000 ffff.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管 (更新指示あり)
ロード・インストラクションとストア・インストラクションには更新フォームがあります。このフォームでは、メモリーとの間で通常の情報の移動に加えて、基本 GPR が EA で更新されます。

lhzux (索引付き更新による半分とゼロのロード) 命令

目的

メモリー内の指定された位置から汎用レジスターにデータのハーフワードをロードし、汎用レジスターの残りの 16 ビットを 0 に設定し、場合によってはアドレスを別の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21 - 30	331

ビット	VALUE
31	/

項目	説明
lhzux (lhzux)	*RT*、 *RA*、 *RB*

説明

lhzux 命令は、メモリー内の指定された場所から、有効アドレス (EA) によってアドレス指定されたデータのハーフワードをターゲット汎用レジスター (GPR) *RT* のビット 16 から 31 にロードし、GPR *RT* のビット 0 から 15 を 0 に設定します。

GPR *RA* が 0 でない場合、EA は GPR *RA* と GPR *RB* の内容の合計です。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

RA が *RT* と等しくなく、 *RA* が 0 と等しくなく、ストレージ・アクセスによってアライメント割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA は GPR *RA* に入れられます。

lhzux 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、データのハーフワードを GPR 6 のビット 16 から 31 にロードし、GPR 6 のビット 0 から 15 を 0 に設定し、有効アドレスを GPR 5 に保管します。

```
.csect data[rw]
storage: .short 0xffff
# Assume GPR 5 contains the address of csect data[rw].
# Assume GPR 4 contains the displacement of storage
# relative to data[rw].
.csect text[pr]
lhzux 6,5,4
# GPR 6 now contains 0x0000 ffff.
# GPR 5 now contains the storage address.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管 (更新指示あり)
ロード・インストラクションとストア・インストラクションには更新フォームがあります。このフォームでは、メモリーとの間で通常の情報の移動に加えて、基本 GPR が EA で更新されます。

lhzx (Load Half and Zero Indexed) 命令

目的

メモリー内の指定された位置から汎用レジスターにデータのハーフワードをロードし、汎用レジスターの残りの 16 ビットを 0 に設定します。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21 - 30	279
31	/

項目	説明
lhzx	*RT*、 *RA*、 *RB*

説明

lhzx 命令は、メモリー内の指定された場所から、有効アドレス (EA) によってアドレス指定されたデータのハーフワードをターゲット汎用レジスター (GPR) *RT* のビット 16 から 31 にロードし、GPR *RT* のビット 0 から 15 を 0 に設定します。

GPR *RA* が 0 でない場合、EA は GPR *RA* と GPR *RB* の内容の合計です。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

lhzx 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、データのハーフワードを GPR 6 のビット 16 から 31 にロードし、GPR 6 のビット 0 から 15 を 0 に設定します。

```
.csect data[rw]
.short 0xffff
.csect text[pr]
# Assume GPR 5 contains the address of csect data[rw].
# Assume 0xffff is the halfword located at displacement 0.
# Assume GPR 4 contains 0x0000 0000.
lhzx 6,5,4
# GPR 6 now contains 0x0000 ffff.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

lmw または lm (複数ワードのロード) 命令

目的

指定された位置にある連続したワードを複数の汎用レジスターにロードします。

構文

ビット	VALUE
0 - 5	46
6 - 10	RT
11 - 15	RA
16 - 31	D

PowerPC (R)

lmw (lmw) *RT*、 *D*(*RA*)

POWER ® ファミリー

LM *RT*、 *D*(*RA*)

説明

lmw 命令および **lm** 命令は、計算された有効アドレス (EA) から始まる *N* 個の連続ワードを、GPR *RT* で始まり、GPR 31 を介してすべての GPR を充てんするいくつかの汎用レジスター (GPR) にロードします。 *N* は 32-*RT* フィールドに等しく、連続したレジスターに入れられる連続したワードの合計数です。

GPR *RA* が 0 でない場合、EA は GPR *RA* と *D* の内容の合計です。 GPR *RA* が 0 の場合、EA は *D* です。

PowerPC ® 命令 **lmw** を使用する場合は、以下の点を考慮してください。

- GPR *RA* または GPR *RB* が、ロードされるレジスターの範囲内にある場合、または *RT* = *RA* = 0 の場合、結果は未定義になります。
- EA は 4 の倍数でなければなりません。 そうでない場合は、システム位置合わせエラー・ハンドラーが呼び出されるか、結果が予期せずに未定義になる可能性があります。

POWER ® ファミリー命令 **lm** では、GPR *RA* が 0 でなく、GPR *RA* がロードされる範囲内にある場合、GPR *RA* は書き込まれません。 通常は *RA* に書き込まれていたはずのデータは破棄され、操作は正常に続行されます。

lmw および **lm** 命令には 1 つの構文があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

> **注: lmw** および **lm** 命令は、データ・ストレージ割り込みのために割り込み可能です。 このような割り込みが発生した場合は、命令を最初から再始動する必要があります。

パラメーター

項目	説明
RT	操作の開始ターゲット汎用レジスターを指定します。
D	EA 計算用に 32 ビットに拡張された 16 ビット符号付き 2 の補数整数符号を指定します。
RA	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、GPR 29 および GPR 31 にデータをロードします。

```
.csect data[rw]
.long 0x8971
.long -1
.long 0x7ffe c100
# Assume GPR 30 contains the address of csect data[rw].
.csect text[pr]
lmw 29,0(30)
# GPR 29 now contains 0x0000 8971.
# GPR 30 now contains the address of csect data[rw].
# GPR 31 now contains 0x7ffe c100.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

lq (ロード・クワッド・ワード) 命令

目的

指定された汎用レジスターにクワッド・ワードのデータをロードします。

注 : この命令は、64 ビット・アプリケーションを実行する 64 ビット PowerPC プロセッサーでのみ使用してください。

構文

ビット	値
0 - 5	56
6 - 10	RS
11 - 15	RA
16 - 27	Dq
28 - 31	0

PowerPC 64

Lq RT、 Disp(RA)

説明

lq 命令は、有効アドレス (EA) によってアドレス指定されたメモリー内の指定された位置から、ターゲット汎用レジスター (GPR) *RT* および *RT+1* にクワッド・ワードをロードします。

DQ は 12 ビットの符号付き 2 の補数で、64 ビットに拡張された符号に 16 を乗算して変位 *Disp* を提供します。 GPR *RA* が 0 でない場合、EA は GPR *RA* と *Disp* の内容の合計です。 GPR *RA* が 0 の場合、EA は *Disp* です。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。 RT が奇数の場合、命令形式は無効です。
処理	16 の倍数である 16 ビット符号付き数値を指定します。 アセンブラーは、命令の生成時にこの数値を 16 で除算します。
RA	EA 計算用のソース汎用レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

lscbx (Load String and Compare Byte Indexed) 命令

目的

ストレージ内の連続したバイトを連続したレジスターにロードします。

注: lscbx 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21 - 30	277
31	rc

項目	説明
POWER ファミリー	POWER ファミリー
LSCBX	*RT*、 *RA*、 *RB*
lscbx:	*RT*、 *RA*、 *RB*

説明

LSCBX 命令は、有効アドレス (EA) によってアドレス指定された *N* 連続バイトを汎用レジスター (GPR) *RT* にロードします。これは、レジスター *RT* の左端のバイトから *RT* + *NR (R)* -1 までであり、必要に応じて、XER16-23 または *N* バイトで一致するものがロードされるまで GPR 0 で折り返します。 一致するバイトが見つかった場合は、そのバイトもロードされます。

GPR *RA* が 0 でない場合、EA は GPR *RA* の内容と GPR *RB* に保管されているアドレスの合計です。 *RA* が 0 の場合、EA は GPR *RB* の内容です。

lscbx 命令を使用する際には、以下のことを考慮してください。

- XER (16-23) には、比較されるバイトが含まれます。
- XER (25-31) には、命令が呼び出される前のバイト・カウントと、命令が完了した後にロードされたバイト数が入ります。
- XER (25-31) = 0 の場合、GPR *RT* は変更されません。
- *N* は XER (25-31) で、これはロードするバイト数です。
- *NR* は上限 (*N*/4) です。これは、連続したバイトを入れるために必要なレジスターの合計数です。

バイトは常にレジスター内で左から右にロードされます。 *N* バイトがロードされる前に一致が検出された場合、そのレジスターからロードされない右端のバイトの内容、およびレジスター *RT* + *NR* -1 までの後続のすべてのレジスターの内容は未定義です。 また、一致したバイトが検出された後は、ストレージへの参照は行われません。 一致が検出されなかった場合、レジスター *RT* + *NR* -1 からロードされなかった右端のバイトの内容は未定義です。

GPR *RA* が 0 でなく、GPR *RA* および *RB* がロードされる範囲内にある場合、GPR *RA* および *RB* は書き込まれません。 書き込まれたはずのデータは破棄され、操作は正常に続行されます。 XER (16-23) のバイトが、GPR *RA* または *RB* にロードされていたが、再始動可能性のために廃棄されている 4 バイトのいずれかと比較される場合、条件レジスターの EQ ビットと、XER (25-31) で戻されるカウントは未定義です。 Multiply Quotient (MQ) レジスターは、この操作の影響を受けません。

lscbx 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明	説明	説明	説明
構文 Form	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
LSCBX	なし	XER (25-31) = ロードされたバイト数	0	なし
lscbx:	なし	XER (25-31) = ロードされたバイト数	1	LT、GT、EQ、SO

lscbx 命令の 2 つの構文形式は、固定小数点例外レジスター (XER) のビット 25 から 31 にロードされるバイト数を配置します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。 Rc = 1 および XER (25-31) = 0 の場合、条件レジスター・フィールド 0 は未定義です。 Rc = 1 および XER (25-31) <> 0 の場合、条件レジスター・フィールド 0 は次のように設定されます。

```
LT, GT, EQ, SO = b'00'||match||XER(SO)
```

> **注:** この命令は、データ・ストレージ割り込みによって割り込まれることがあります。 このような割り込みが発生すると、命令は最初から再開されます。

パラメーター

項目	説明
RT	開始ターゲット汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

1. 次のコードは、連続したバイトを GPR 6、7、および 8 にロードします。

```
.csect data[rw]
string: "Hello, world"
# Assume XER16-23 = 'a.
# Assume XER25-31 = 9.
# Assume GPR 5 contains the address of csect data[rw].
# Assume GPR 4 contains the displacement of string relative
# to csect data[rw].
.csect text[pr]
lscbx 6,5,4
# GPR 6 now contains 0x4865 6c6c.
# GPR 7 now contains 0x6f2c 2077.
# GPR 8 now contains 0x6fXX XXXX.
```

2. 次のコードは、連続したバイトを GPR 6、7、および 8 にロードします。

```
# Assume XER16-23 = 'e.
# Assume XER25-31 = 9.
# Assume GPR 5 contains the address of csect data[rw].
# Assume GPR 4 contains the displacement of string relative
# to csect data[rw].
.csect text[pr]
lscbx. 6,5,4
# GPR 6 now contains 0x4865 XXXX.
# GPR 7 now contains 0xXXXX XXXX.
# GPR 8 now contains 0xXXXX XXXX.
# XER25-31 = 2.
# CRF 0 now contains 0x2.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ストリング命令
固定小数点ストリング命令を使用すると、位置合わせを考慮せずに、ストレージからレジスターへ、またはレジスターからストレージへデータを移動することができます。

lAfter または lsi (Load String Word Immediate) 命令

目的

ストレージ内の連続したバイトを、メモリー内の指定された位置から連続した汎用レジスターにロードします。

構文

ビット	VALUE
0 -5	31
6 - 10	RT
11 - 15	RA
16 - 20	NB (B)
21 - 30	597
31	/

PowerPC (R)

ルシオル *RT*、 *RA*、 *NB*

POWER® ファミリー

LSI *RT*、 *RA*、 *NB*

説明

l 返送 および **lsi** 命令は、実効アドレス (EA) によってアドレス指定されたストレージ内の *N* 個の連続バイトを汎用レジスター GPR *RT* にロードします。これは左端のバイトから GPR *RT*+*NR*-1 で始まり、必要に応じて GPR 0 で折り返します。

GPR *RA* が 0 でない場合、EA は GPR *RA* の内容です。 GPR *RA* が 0 の場合、EA は 0 です。

l 真剣に および **lsi** 命令を使用する場合は、以下のことを考慮してください。

- *NB* はバイト・カウントです。
- *RT* は、開始汎用レジスターです。

- *N* は *NB* で、これはロードするバイト数です。 *NB* が 0 の場合、 *N* は 32 です。
- NR は上限 (N/4) です。これは、データを受け取る汎用レジスターの数です。

PowerPC ® 命令 **L れている**では、GPR *RA* とは がロードされるレジスターの範囲内にある場合、または *RT* = *RA* とは = 0 の場合、命令形式は無効です。

POWER ® ファミリー命令 **lsi** を使用する場合は、以下の点を考慮してください。

- GPR *RT* + *NR* -1 が左側に部分的にしか埋められていない場合、その汎用レジスターの右端のバイトは 0 に設定されます。
- GPR *RA* がロードされる範囲内にあり、GPR *RA* が 0 に等しくない場合、GPR *RA* はこの命令によって書き込まれません。 書き込まれていたはずのデータは破棄され、操作は正常に続行されます。

l 判断 命令と **lsi** 命令には、固定小数点例外レジスターまたは条件レジスター・フィールド 0 に影響しない 1 つの構文形式があります。

> **注: l 方言** および **lsi** 命令は、データ・ストレージ割り込みによって割り込むことができます。 このような割り込みが発生すると、命令は最初から再開されます。

パラメーター

項目	説明
RT	保管データの汎用レジスターの開始を指定します。
RA	EA 計算の汎用レジスターを指定します。
NB *(NB)*	バイト・カウントを指定します。

例

以下のコードは、GPR 7 によってアドレス指定されたメモリー内の位置に含まれるバイトを GPR 6 にロードします。

```
.csect data[rw]
.string "Hello, World"
# Assume GPR 7 contains the address of csect data[rw].
.csect text[pr]
lswi 6,7,0x6
# GPR 6 now contains 0x4865 6c6c.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ストリング命令
固定小数点ストリング命令を使用すると、位置合わせを考慮せずに、ストレージからレジスターへ、またはレジスターからストレージへデータを移動することができます。

lswx または lsx (Load String Word Indexed) 命令

目的

ストレージ内の連続したバイトを、メモリー内の指定された位置から連続した汎用レジスターにロードします。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21 - 30	533
31	/

PowerPC (R)

LSWX *RT*、 *RA*、 *RB*

POWER ® ファミリー

lsx (lsx) *RT*、 *RA*、 *RB*

説明

lswx および **lsx** 命令は、実効アドレス (EA) によってアドレス指定されたストレージ内の *N* 個の連続バイトを、GPR *RT* + *NR* -1 を介して左端のバイトから始まる汎用レジスター (GPR) *RT* にロードし、必要に応じて GPR 0 で折り返します。

GPR *RA* が 0 でない場合、EA は GPR *RA* の内容と GPR *RB* に保管されているアドレスの合計です。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

lswx および **lsx** 命令を使用する場合は、以下のことを考慮してください。

- XER (25-31) にはバイト・カウントが入ります。
- *RT* は、開始汎用レジスターです。
- *N* は XER (25-31) で、これはロードするバイト数です。
- *NR* は上限 (N/4) です。これは、データを受け取るレジスターの数です。
- XER (25-31) = 0 の場合、汎用レジスター *RT* は変更されません。

PowerPC ® 命令 **lswx** では、*RA* または *RB* がロードされるレジスターの範囲内にある場合、または *RT* = *RA* = 0 である場合、結果は未定義になります。

POWER ® ファミリー命令 **lsx** を使用する場合は、以下の点を考慮してください。

- GPR *RT* + *NR* -1 が左側に部分的にしか埋められていない場合、その汎用レジスターの右端のバイトは 0 に設定されます。
- GPR *RA* および *RB* がロードされる範囲内にあり、GPR *RA* が 0 に等しくない場合、GPR *RA* および *RB* はこの命令によって書き込まれません。 書き込まれたはずのデータは破棄され、操作は正常に続行されます。

lswx および **lsx** 命令には、固定小数点例外レジスターまたは条件レジスター・フィールド 0 に影響しない 1 つの構文形式があります。

> **注: lswx** および **lsx** 命令は、データ・ストレージ割り込みによって割り込むことができます。 このような割り込みが発生すると、命令は最初から再開されます。

パラメーター

項目	説明
RT	保管データの汎用レジスターの開始を指定します。
RA	EA 計算の汎用レジスターを指定します。

項目	説明
rb	EA 計算の汎用レジスターを指定します。

例

以下のコードは、GPR 5 によってアドレス指定されたメモリー内の位置に含まれるバイトを GPR 6 にロードします。

```
# Assume XER25-31 = 4.
csect data[rw]
storage: .string "Hello, world"
# Assume GPR 4 contains the displacement of storage
# relative to data[rw].
# Assume GPR 5 contains the address of csect data[rw].
.csect text[pr]
lswx 6,5,4
# GPR 6 now contains 0x4865 6c6c.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ストリング命令
固定小数点ストリング命令を使用すると、位置合わせを考慮せずに、ストレージからレジスターへ、またはレジスターからストレージへデータを移動することができます。

POWER ® ファミリーと PowerPC ® 命令の機能の違い
POWER ® ファミリーと PowerPC ® 命令は、POWER ® ファミリーと PowerPC ® プラットフォームで同じ命令コードを共有しますが、機能定義は異なります。

lwa (Load Word Algebraic) 命令

目的

ストレージから、指定された汎用レジスターの下位 32 ビットにフルワードのデータをロードします。 符号は、データをレジスターの高位 32 ビットに拡張します。

構文

ビット	VALUE
0 - 5	58
6 - 10	RT
11 - 15	RA
16 - 29	DS
30 - 31	0b10

PowerPC 64

Lwa (Lwa) *RT*、 *Disp* (*RA*)

説明

有効アドレス (EA) にあるストレージ内のフルワードは、ターゲット汎用レジスター (GRP) *RT* の下位 32 ビットにロードされます。 その後、値は符号拡張され、レジスターの上位 32 ビットを埋めます。

DS は 14 ビットの符号付き 2 の補数で、64 ビットに符号拡張され、さらに 4 を乗算して変位 *Disp* を提供します。 GPR *RA* が 0 でない場合、EA は GPR *RA* と *Disp* の内容の合計です。 GPR *RA* が 0 の場合、EA は *Disp* です。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
処理	4 の倍数である 16 ビットの符号付き数値を指定します。 アセンブラーは、命令の生成時にこの数値を 4 で除算します。
RA	EA 計算用のソース汎用レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

lwarx (Load Word and Reserve Indexed) 命令

目的

後続の **stwcx** と一緒に使用されます。 指定されたメモリー・ロケーションで読み取り/変更/書き込み操作をエミュレートする命令。

注: lwarx 命令は、PowerPC ® アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21- 30	20
31	/

PowerPC (R)

Lwarx (Lwarx)	*RT*、 *RA*、 *RB*

説明

lwarx および **stwcx.** 命令は、ストレージに対して読み取り/変更/書き込み操作を実行するために使用される、基本的または単純な命令です。 ストアが実行される場合、 **lwarx** および **stwcx** の使用。 命令により、**lwarx** 命令が実行されてから **stwcx** が実行されるまでの間に、他のプロセッサーまたはメカニズムがターゲット・メモリーの位置を変更していないことが保証されます。 命令が完了する。

汎用レジスター (GPR) *RA* = 0 の場合、有効アドレス (EA) は GPR *RB* の内容になります。 それ以外の場合、EA は GPR *RA* の内容と GPR *RB* の内容の合計です。

lwarx 命令は、EA によって指定されたストレージ内の位置からターゲット GPR *RT* にワードをロードします。 さらに、後続の **stwcx** で使用するために、メモリー・ロケーションの予約が作成されます。 命令。

lwarx 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。 EA が 4 の倍数でない場合、結果は予期しないものになります。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

1. 以下のコードは、ストレージ内のワードをアトミックにロードして置き換えることにより、「取り出しと保管」を実行します。

```
# Assume that GPR 4 contains the new value to be stored.
# Assume that GPR 3 contains the address of the word
# to be loaded and replaced.
loop:   lwarx   r5,0,r3          # Load and reserve
        stwcx.  r4,0,r3          # Store new value if still
                                 # reserved
        bne-    loop             # Loop if lost reservation
# The new value is now in storage.
# The old value is returned to GPR 4.
```

2. 以下のコードは、レジスター内の値をストレージ内のワードとアトミックに比較することにより、「比較とスワップ」を実行します。

```
# Assume that GPR 5 contains the new value to be stored after
# a successful match.
# Assume that GPR 3 contains the address of the word
# to be tested.
# Assume that GPR 4 contains the value to be compared against
# the value in memory.
loop:   lwarx   r6,0,r3          # Load and reserve
        cmpw    r4,r6            # Are the first two operands
                                 # equal?
        bne-    exit             # Skip if not equal
        stwcx.  r5,0,r3          # Store new value if still
                                 # reserved
        bne-    loop             # Loop if lost reservation
exit:   mr      r4,r6            # Return value from storage
# The old value is returned to GPR 4.
# If a match was made, storage contains the new value.
```

   レジスターの値がストレージ内のワードと等しい場合は、2 番目のレジスターの値がストレージ内のワードに保管されます。 それらが等しくない場合は、ストレージからのワードが最初のレジスターにロードされ、条件レジスター・フィールド 0 の EQ ビットが比較の結果を示すように設定されます。

関連概念

stwcx。 (Store Word Conditional Indexed) 命令

処理とストレージ

プロセッサーは、メイン・メモリーおよびレジスターにデータを保管します。

lwaux (Update Indexed による Word Algebraic のロード) 命令

目的

ストレージからフルワードのデータを、指定された汎用レジスターの下位 32b にロードします。 符号は、データをレジスターの高位 32 ビットに拡張します。 アドレス・ベースを更新します。

構文

ビット	VALUE
0 - 5	31
6 - 10	D
11 - 15	A
16 - 20	B
21 - 30	373
31	0

POWER® ファミリー

lwaux *RT*、 *RA*、 *RB*

説明

有効アドレス (EA) にあるストレージ内のフルワードは、ターゲット汎用 puspose レジスター (GRP) の下位 32 ビットにロードされます。 その後、値は符号拡張され、レジスターの上位 32 ビットを埋めます。 EA は、GRP *RA* および GRP *RB* の内容の合計です。

RA = 0 または *RA* = *RT* の場合、命令形式は無効です。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

lwax (ロード・ワード代数インデックス付き) 命令

目的

ストレージから、指定された汎用レジスターの下位 32 ビットにフルワードのデータをロードします。 符号は、データをレジスターの高位 32 ビットに拡張します。

構文

ビット	VALUE
0 - 5	31
6 - 10	D
11 - 15	A
16 - 20	B
21 - 30	341
31	0

POWER® ファミリー

lwax *RT*、 *RA*、 *RB*

説明

有効アドレス (EA) にあるストレージ内のフルワードは、ターゲット汎用 puspose レジスター (GRP) の下位 32 ビットにロードされます。 その後、値は符号拡張され、レジスターの上位 32 ビットを埋めます。

GRP *RA* が 0 でない場合、EA は GRP *RA* と *B* の内容の合計です。それ以外の場合、EA は *RB* の内容と等しくなります。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

lwbrx または lbrx (Load Word Byte-Reverse Indexed) 命令

目的

メモリー内の指定された位置から汎用レジスターにデータのバイト反転ワードをロードします。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21 - 30	534
31	/

PowerPC (R)

lwbrx (lwbrx) *RT*、 *RA*、 *RB*

POWER® ファミリー

lbrx (lbrx) *RT*、 *RA*、 *RB*

説明

lwbrx および **lbrx** 命令は、実効アドレス (EA) によってアドレス指定されたメモリー内の指定された位置から、ストレージ内のバイト反転ワードをターゲット汎用レジスター (GPR) *RT* にロードします。

lwbrx および **lbrx** 命令を使用する場合は、以下のことを考慮してください。

- EA によってアドレス指定されたストレージ内のワードのビット 00 から 07 は、GPR *RT* のビット 24 から 31 に入れられます。
- EA によってアドレス指定されたストレージ内のワードのビット 08 から 15 は、GPR *RT* のビット 16 から 23 に入れられます。
- EA によってアドレス指定されたストレージ内のワードのビット 16 から 23 は、GPR *RT* のビット 08 から 15 に入れられます。
- EA によってアドレス指定されたストレージ内のワードのビット 24 から 31 は、GPR *RT* のビット 00 から 07 に入れられます。

GPR *RA* が 0 でない場合、EA は GPR *RA* と GPR *RB* の内容の合計です。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

lwbrx および **lbrx** 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、バイト反転ワードをメモリーから GPR 6 にロードします。

```
storage:  .long 0x0000 ffff
 .
 .
# Assume GPR 4 contains 0x0000 0000.
# Assume GPR 5 contains address of storage.
lwbrx 6,4,5
# GPR 6 now contains 0xffff 0000.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

lwz または l (Load Word and Zero) 命令

目的

メモリー内の指定された位置から汎用レジスターにデータのワードをロードします。

構文

ビット	VALUE
0 - 5	32
6 - 10	RT
11 - 15	RA
16 - 31	D

PowerPC (R)

LWZ *RT*、 *D*(*RA*)

POWER® ファミリー

l *RT*、 *D*(*RA*)

説明

lwz および l 命令は、実効アドレス (EA) によってアドレス指定されたメモリー内の指定された位置から、ストレージ内のワードをターゲット汎用レジスター (GPR) *RT* にロードします。

GPR *RA* が 0 でない場合、EA は、GPR *RA* と *D*(16 ビット、符号付き 2 の補数整数符号-32 ビットに拡張) の内容の合計です。 GPR *RA* が 0 の場合、EA は *D* です。

lwz および l 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
D	EA 計算用に 16 ビットの符号付き 2 の補数整数符号拡張を 32 ビットに指定します。
RA	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、ワードをメモリーから GPR 6 にロードします。

```
.csect data[rw]
# Assume GPR 5 contains address of csect data[rw].
storage: .long 0x4
.csect text[pr]
lwz 6,storage(5)
# GPR 6 now contains 0x0000 0004.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

lwzu または lu (ゼロ更新によるワードのロード) 命令

目的

メモリー内の指定された位置から汎用レジスターにデータのワードをロードし、場合によっては有効アドレスを 2 番目の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	33
6 - 10	RT

ビット	VALUE
11 - 15	RA
16 - 31	D

PowerPC (R)

LW ツー *RT*、 *D*(*RA*)

POWER® ファミリー

Lu *RT*、 *D*(*RA*)

説明

lwzu および **lu** 命令は、実効アドレス (EA) によってアドレス指定されたメモリー内の指定された位置から、ストレージ内のワードをターゲット汎用レジスター (GPR) *RT* にロードします。

GPR *RA* が 0 でない場合、EA は、GPR *RA* と *D*(16 ビット、符号付き 2 の補数整数符号-32 ビットに拡張) の内容の合計です。 GPR *RA* が 0 の場合、EA は *D* です。

RA が *RT* と等しくなく、 *RA* が 0 と等しくなく、ストレージ・アクセスによってアライメント割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA は GPR *RA* に入れられます。

lwzu および **lu** 命令は、1 つの構文形式を持ち、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
D	EA 計算用に 16 ビットの符号付き 2 の補数整数符号拡張を 32 ビットに指定します。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。

例

以下のコードは、ワードをメモリーから GPR 6 にロードし、有効アドレスを GPR 4 に入れます。

```
.csect data[rw]
storage: .long 0xffdd 75ce
.csect text[pr]
# Assume GPR 4 contains address of csect data[rw].
lwzu 6,storage(4)
# GPR 6 now contains 0xffdd 75ce.
# GPR 4 now contains the storage address.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管 (更新指示あり)
ロード・インストラクションとストア・インストラクションには更新フォームがあります。このフォームでは、メモリーとの間で通常の情報の移動に加えて、基本 GPR が EA で更新されます。

lwzux または lux (Load Word and Zero with Update Indexed) 命令

目的

メモリー内の指定された位置から汎用レジスターにデータのワードをロードし、場合によっては有効アドレスを 2 番目の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21 - 30	55
31	/

PowerPC (R)

LWZUX *RT*、 *RA*、 *RB*

POWER® ファミリー

lux (ラックス) *RT*、 *RA*、 *RB*

説明

lwzux および lux 命令は、メモリー内の指定された場所 (有効アドレス (EA) によってアドレス指定された) から 1 ワードのデータをターゲット汎用レジスター (GPR) *RT* にロードします。

GPR *RA* が 0 でない場合、EA は GPR *RA* と GPR *RB* の内容の合計です。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

GPR *RA* が RT と等しくなく、 *RA* が 0 と等しくなく、ストレージ・アクセスによってアライメント割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA は GPR *RA* に入れられます。

lwzux および **lux** 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスターのフィールド 0 には影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、ワードをメモリーから GPR 6 にロードし、有効アドレスを GPR 5 に入れます。

```
.csect data[rw]
storage: .long 0xffdd 75ce
# Assume GPR 5 contains the address of csect data[rw].
# Assume GPR 4 contains the displacement of storage
# relative to csect data[rw].
.csect text[pr]
lwzux 6,5,4
# GPR 6 now contains 0xffdd 75ce.
# GPR 5 now contains the storage address.
```

関連概念

固定小数点プロセッサー

固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管 (更新指示あり)

ロード・インストラクションとストア・インストラクションには更新フォームがあります。このフォームでは、メモリーとの間で通常の情報の移動に加えて、基本 GPR が EA で更新されます。

lwzx または lx (Load Word and Zero Indexed) 命令

目的

メモリー内の指定された位置から汎用レジスターにデータのワードをロードします。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21 - 30	23
31	/

PowerPC (R)

LWZX *RT*、 *RA*、 *RB*

POWER® ファミリー

Lx *RT*、 *RA*、 *RB*

説明

lwzx 命令および **lx** 命令は、メモリー内の指定された場所 (有効アドレス (EA) によってアドレス指定される) から、ターゲット汎用レジスター (GPR) *RT* にデータのワードをロードします。

GPR *RA* が 0 でない場合、EA は GPR *RA* と GPR *RB* の内容の合計です。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

lwzx および **lx** 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、ワードをメモリーから GPR 6 にロードします。

```
.csect data[rw]
.long 0xffdd 75ce
# Assume GPR 4 contains the displacement relative to
# csect data[rw].
# Assume GPR 5 contains the address of csect data[rw].
.csect text[pr]
lwzx 6,5,4
# GPR 6 now contains 0xffdd 75ce.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

maskg (マスク生成) 命令

目的

1 と 0 のマスクを生成し、汎用レジスターにロードします。

注: maskg 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	29
31	rc

POWER® ファミリー

マスク *RA*、*RS*、*RB*

maskg。 *RA*、*RS*、*RB*

説明

maskg 命令は、汎用レジスター (GPR) *RS* のビット 27 から 31 によって定義された開始点から GPR *RB* のビット 27 から 31 によって定義された終点までのマスクを生成し、そのマスクを GPR *RA* に保管します。

maskg 命令を使用する際には、以下のことを考慮してください。

- 開始点ビットが終了点ビット + 1 より小さい場合には, 開始点と終了点の間 (開始点を含む) のビットが 1 に設定されます。 その他のビットはすべて 0 に設定されます。
- 開始点ビットが終了点ビット + 1 と同じ場合は、32 ビットすべてが 1 に設定されます。
- 開始点ビットが終了点ビット + 1 より大きい場合は、終了点ビット + 1 と開始点ビット -1 の間にあるすべてのビットがゼロに設定されます。 その他のビットはすべて 1 に設定されます。

maskg 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
マスク	なし	なし	0	なし
maskg。	なし	なし	1	LT、GT、EQ、SO

maskg 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	マスクの開始のためのソース汎用レジスターを指定します。
rb	マスクの終わりのソース汎用レジスターを指定します。

例

1. 以下のコードは、5 個のマスクを生成し、その結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x0000 0014.
# Assume GPR 5 contains 0x0000 0010.
maskg 6,5,4
# GPR 6 now contains 0x0000 F800.
```

2. 以下のコードは、残りのビットが 1 に設定された 6 個のゼロのマスクを生成し、結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0x0000 0010.
# Assume GPR 5 contains 0x0000 0017.
# Assume CR = 0.
maskg. 6,5,4
# GPR 6 now contains 0xFFFF 81FF.
# CR now contains 0x8000 0000.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

maskir (Mask Insert from Register) 命令

目的

1 つの汎用レジスターの内容を、ビット・マスクの制御下にある別の汎用レジスターに挿入します。

注: maskir 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31

ビット	VALUE
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	541
31	rc

POWER® ファミリー

マスク (maskir) *RA*、*RS*、*RB*

マスキル *RA*、*RS*、*RB*

説明

maskir は、汎用レジスター (GPR) *RS* の内容を、GPR *RB* のビット・マスクの制御下にある GPR *RA* に保管します。

ターゲット GPR *RA* の各ビットの値は、以下のように決定されます。

- マスク GPR *RB* 内の対応するビットが 1 の場合、ターゲット GPR *RA* 内のビットには、ソース GPR *RS* 内の対応するビットの値が与えられます。
- マスク GPR *RB* 内の対応するビットが 0 の場合、ターゲット GPR *RA* 内のビットは変更されません。

maskir 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
マスク (maskir)	なし	なし	0	なし
マスキル	なし	なし	1	LT、 GT、EQ、SO

maskir 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	ビット・マスクのソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 のビット・マスクの制御下で GPR 5 の内容を GPR 6 に挿入します。

```
# Assume GPR 6 (RA) target contains 0xAAAAAAAA.
# Assume GPR 4 (RB) mask contains   0x000F0F00.
# Assume GPR 5 (RS) source contains 0x55555555.
maskir 6,5,4
# GPR 6 (RA) target now contains    0xAAA5A5AA.
```

1. 以下のコードは、GPR 4 のビット・マスクの制御下で GPR 5 の内容を GPR 6 に挿入し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 6 (RA) target contains 0xAAAAAAAA.
# Assume GPR 4 (RB) mask contains   0x0A050F00.
# Assume GPR 5 (RS) source contains 0x55555555.
maskir. 6,5,4
# GPR 6 (RA) target now contains    0xA0AFA5AA.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

mcrf (条件移動レジスター・フィールド) 命令

目的

ある条件レジスター・フィールドの内容を別の条件レジスター・フィールドにコピーします。

構文

ビット	VALUE
0 - 5	19
6 - 8	BF
9 - 10	//
11 - 13	BFA とは
14 - 15	//
16 - 20	///
21 - 30	0
31	/

項目	説明
mcrf (mcrf)	*BF*、 *BFA*

説明

mcrf 命令は、*BFA* で指定された条件レジスター・フィールドの内容を、*BF* で指定された条件レジスター・フィールドにコピーします。 他のすべてのフィールドは影響を受けません。

mcrf 命令には 1 つの構文形式があり、条件レジスター・フィールド 0 または固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
BF	操作のターゲット条件レジスター・フィールドを指定します。
BFA (BFA)	操作のソース条件レジスター・フィールドを指定します。

例

以下のコードは、条件レジスター・フィールド 3 の内容を条件レジスター・フィールド 2 にコピーします。

```
# Assume Condition Register Field 3 holds b'0110'.
mcrf 2,3
# Condition Register Field 2 now holds b'0110'.
```

関連概念

ブランチ・プロセッサー

分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

mcrfs (FPSCR から条件レジスターへの移動) 命令

目的

浮動小数点状況および制御レジスターの 1 つのフィールドから条件レジスターにビットをコピーします。

構文

ビット	VALUE
0 - 5	63
6 - 8	BF
9 - 10	//
11 - 13	BFA とは
14 - 15	//
16 - 20	///
21 - 30	64
31	/

項目	説明
mcrfs (mcrfs)	*BF*、 *BFA*

説明

mcrfs 命令は、 *BFA* によって指定された浮動小数点状況制御レジスター (FPSCR) の 4 ビットを条件レジスター・フィールド *BF* にコピーします。 他のすべての条件レジスター・ビットは変更されません。

BFA によって指定されたフィールドに予約済みまたは未定義のビットが含まれている場合、ゼロ値のビットがコピー用に提供されます。

mcrfs 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターのビットを設定できます。

BFA とは	FPSCR ビット・セット
0	FX、OX
1	UX、ZX、XX、VXSNAN
2	VXISI、VXIDI、VXZDZ、VXIMZ
3	VXVC (V)

パラメーター

項目	説明
BF	操作の結果が保管されるターゲット条件レジスター・フィールドを指定します。
BFA (BFA)	FPSCR フィールドの 1 つを指定します (0 から 7)。

例

以下のコードは、浮動小数点状況および制御レジスター・フィールド 4 から条件レジスター・フィールド 3 にビットをコピーします。

```
# Assume FPSCR 4 contains b'0111'.
mcrfs 3,4
# Condition Register Field 3 contains b'0111'.
```

関連概念

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

浮動小数点レジスターの内容の解釈
32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

mcrxr (XER から条件レジスターへの移動) 命令

目的

要約オーバーフロー・ビット、オーバーフロー・ビット、キャリー・ビット、およびビット 3 を、固定小数点例外レジスターから条件レジスターの指定されたフィールドにコピーします。

構文

ビット	VALUE
0 - 5	31
6 - 8	BF
9 - 10	//
11 - 15	///
16 - 20	///
21 - 30	512
31	/

項目	説明
mcrxr (mcrxr)	*BF (BF)*

説明

mcrxr は、固定小数点例外レジスター・フィールド 0 ビット 0 から 3 の内容を条件レジスター・フィールド *BF* にコピーし、固定小数点例外レジスター・フィールド 0 を 0 にリセットします。

mcrxr 命令には 1 つの構文形式があり、固定小数点例外レジスターのビット 0 から 3 を 0 にリセットします。

パラメーター

項目	説明
BF	操作の結果が保管されるターゲット条件レジスター・フィールドを指定します。

例

以下のコードは、要約オーバーフロー・ビット、オーバーフロー・ビット、キャリー・ビット、およびビット 3 を、固定小数点例外レジスターから条件レジスターのフィールド 4 にコピーします。

```
# Assume bits 0-3 of the Fixed-Point Exception
# Register are set to b'1110'.
mcrxr 4
# Condition Register Field 4 now holds b'1110'.
```

関連概念

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

特殊目的のレジスター命令との間での固定小数点移動
この命令は、ある特殊目的レジスター (SPR) の内容を別の SPR または汎用レジスター (GPR) に移動します。これには、非特権命令と特権命令の両方が含まれます。

mfcr (条件レジスターからの移動) 命令

目的

条件レジスターの内容を、汎用レジスターにコピーします。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	///
16 - 20	///
21 - 30	19
31	rc

項目	説明
mfcr (mfcr)	*RT*

説明

mfcr 命令は、条件レジスターの内容をターゲット汎用レジスター (GPR) *RT* にコピーします。

mfcr 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。

例

以下のコードは、条件レジスターを GPR 6 にコピーします。

```
# Assume the Condition Register contains 0x4055 F605.
mfcr 6
# GPR 6 now contains 0x4055 F605.
```

関連概念

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

特殊目的のレジスター命令との間での固定小数点移動
この命令は、ある特殊目的レジスター (SPR) の内容を別の SPR または汎用レジスター (GPR) に移動します。これには、非特権命令と特権命令の両方が含まれます。

m プログラム (FPSCR からの移動) 命令

目的

浮動小数点状況および制御レジスターの内容を浮動小数点レジスターにロードし、上位 32 ビットに 1 を入れます。

構文

ビット	値
0 - 5	63
6 - 10	FRT (R)
11 - 15	///
16 - 20	///
21 - 30	583
31	rc

項目	説明
マフス	*FRT*
マフス	*FRT*

説明

m ォス 命令は、浮動小数点状況および制御レジスターの内容を、浮動小数点レジスター (FPR) *FRT* のビット 32 から 63 に入れます。 浮動小数点レジスター *FRT* のビット 0 から 31 は未定義です。

m 対して 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文形式	**FPSCR ビット**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
マフス	なし	0	なし
マフス	なし	1	FX、FEX、VX、OX

m 防 命令の 2 つの構文形式は、浮動小数点状況および制御レジスター・フィールドには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は、条件レジスター・フィールド 1 の浮動小数点

例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

パラメーター

項目	説明
FRT (FRT)	演算結果が保管されるターゲット浮動小数点レジスターを指定します。

例

以下のコードは、浮動小数点状況および制御レジスターの内容を FPR 14 にロードし、そのレジスターの上位 32 ビットに 1 を入れます。

```
# Assume FPSCR contains 0x0000 0000.
mffs 14
# FPR 14 now contains 0xFFFF FFFF 0000 0000.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点レジスターの内容の解釈
32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

POWER® ファミリーと PowerPC® 命令の機能の違い
POWER® ファミリーと PowerPC® 命令は、POWER® ファミリーと PowerPC® プラットフォームで同じ命令コードを共有しますが、機能定義は異なります。

mfmsr (マシン状態レジスターからの移動) 命令

目的

マシンの状態レジスターの内容を汎用レジスターにコピーします。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	///
16 - 20	///
21 - 30	83
31	/

項目	説明
mfmsr (mfmsr)	*RT*

説明

mfmsr 命令は、マシン状態レジスターの内容をターゲット汎用レジスター (GPR) *RT* にコピーします。

mfmsr 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスターのフィールド 0 には影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。

例

以下のコードは、マシン状態レジスターの内容を GPR 4 にコピーします。

```
mfmsr 4
# GPR 4 now holds a copy of the bit
# settings of the Machine State Register.
```

セキュリティー

mfmsr 命令は、PowerPC ® アーキテクチャーでのみ特権を付与されます。

関連概念

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

特殊目的のレジスター命令との間での固定小数点移動
この命令は、ある特殊目的レジスター (SPR) の内容を別の SPR または汎用レジスター (GPR) に移動します。これには、非特権命令と特権命令の両方が含まれます。

POWER ® ファミリーと PowerPC ® 命令の機能の違い
POWER ® ファミリーと PowerPC ® 命令は、POWER ® ファミリーと PowerPC ® プラットフォームで同じ命令コードを共有しますが、機能定義は異なります。

mfocrf (1 つの条件レジスター・フィールドからの移動) 命令

目的

1 つの条件レジスター・フィールドの内容を汎用レジスターにコピーします。

構文

ビット	値
0 - 5	31
6 - 10	RT
11	1
12 - 19	FXM (M)
20	///
21 - 30	19
31	///

項目	説明
mfocrf (mfocrf)	RT、 FXM

説明

mfocrf 命令は、フィールド・マスク FXM によって指定された 1 つの条件レジスター・フィールドの内容を、ターゲット汎用レジスター (GPR) *RT* にコピーします。

フィールド・マスク FXM は、次のように定義されます。

ビット	説明
12	CR 00 から 03 は GPR *RS* 00-03 にコピーされます。
13	CR 04-07 は GPR *RS* 04-07 にコピーされます。
14	CR 08-11 が GPR *RS* 08-11 にコピーされます。
15	CR 12-15 は GPR *RS* 12-15 にコピーされます。
16	CR 16 から 19 は、GPR *RS* 16 から 19 にコピーされます。
17	CR 20 から 23 は、GPR *RS* 20 から 23 にコピーされます。
18	CR 24 から 27 は、GPR *RS* 24 から 27 にコピーされます。
19	CR 28-31 は GPR *RS* 28-31 にコピーされます。

mfocrf 命令は 1 つの構文形式を持ち、固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
FXM (FXM)	フィールド・マスクを指定します。 指定できるビットは 1 つだけです。

例

以下のコードは、条件レジスター・フィールド 3 を GPR 6 にコピーします。

```
# Assume the Condition Register contains 0x4055 F605.
# Field 3 (0x10 = b'0001 0000')
mfocrf 6, 0x10
# GPR 6 now contains 0x0005 0000.
```

関連概念

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

特殊目的のレジスター命令との間での固定小数点移動
この命令は、ある特殊目的レジスター (SPR) の内容を別の SPR または汎用レジスター (GPR) に移動します。これには、非特権命令と特権命令の両方が含まれます。

mfspr (特殊目的レジスターからの移動) 命令

目的

特殊目的レジスターの内容を、汎用レジスターにコピーします。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 20	SPR
21 - 30	339
31	rc

項目	説明
mfspr (mfspr)	*RT*、 *SPR*

注: 特殊目的レジスターは、分割フィールドです。

詳しくは、「特殊目的レジスターとの間の移動の拡張ニーモニック」を参照してください。

説明

mfspr 命令は、特殊目的レジスター *SPR* の内容をターゲット汎用レジスター (GPR) *RT* にコピーします。

特殊目的のレジスター ID *SPR* には、以下の表に示す値のいずれかを指定できます。 SPR 番号の 2 つの 5 ビット半分の順序が逆になります。

項目	説明	説明	説明
SPR 値	**SPR 値**	**SPR 値**	**SPR 値**
10 進数	**spr**$^{5:9}$ spr$^{0:4}$	**レジスター名**	**privileged**
1	00000 00001	XER	いいえ
8	00000 01000	LR	いいえ
9	00000 01001	CTR	いいえ
18	00000 10010	DSISR	はい
19	00000 10011	DAR	はい
22	00000 10110	DEC^{2}	はい
25	00000 11001	SDR1	はい
26	00000 11010	SRR0	はい
27	00000 11011	SRR1	はい
272	01000 10000	SPRG0	はい
273	01000 10001	SPRG1	はい
274	01000 10010	SPRG2	はい
275	01000 10011	SPRG3	はい

項目	説明	説明	説明
282	01000 11010	EAR	はい
284	01000 11100	TBL	はい
285	01000 11101	TBU (U)	はい
528	10000 10000	IBAT0U	はい
529	10000 10001	IBAT0L	はい
530	10000 10010	IBAT1U	はい
531	10000 10011	IBAT1L	はい
532	10000 10100	IBAT2U	はい
533	10000 10101	IBAT2L	はい
534	10000 10110	IBAT3U	はい
535	10000 10111	IBAT3L	はい
536	10000 11000	DBAT0U	はい
537	10000 11001	DBAT0L	はい
538	10000 11010	DBAT1U	はい
539	10000 11011	DBAT1L	はい
540	10000 11100	DBAT2U	はい
541	10000 11101	DBAT2L	はい
542	10000 11110	DBAT3U	はい
543	10000 11111	DBAT3L	はい
0	00000 00000	MQ[1]	いいえ
4	00000 00100	RTCU[1]	いいえ
5	00000 00101	RTCL[1]	いいえ
6	00000 00110	DEC[2]	いいえ

1Supported (POWER ファミリー・アーキテクチャーのみ)。

2In : DEC レジスターから移動する PowerPC アーキテクチャーには特権があり、SPR 値は 22 です。 DEC レジスターから移行する POWER ファミリー・アーキテクチャーでは、特権はなく、SPR 値は 6 です。 詳しくは、「Fixed-Point Move to or from Special-Purpose Registers Instructions 」を参照してください。

SPR フィールドに SPR 値テーブルにリストされている値以外の値が含まれている場合、命令フォームは無効です。

mfspr 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
SPR	操作用のソース特殊目的レジスターを指定します。

例

以下のコードは、固定小数点例外レジスターの内容を GPR 6 にコピーします。

```
mfspr 6,1
# GPR 6 now contains the bit settings of the Fixed
# Point Exception Register.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

特殊目的のレジスター命令との間での固定小数点移動
この命令は、ある特殊目的レジスター (SPR) の内容を別の SPR または汎用レジスター (GPR) に移動します。これには、非特権命令と特権命令の両方が含まれます。

mfsr (セグメント・レジスターからの移動) 命令

目的

セグメント・レジスターの内容を汎用レジスターにコピーします。

構文

ビット	VALUE
0 - 5	31
6 - 8	RT
11	/
12 - 14	SR
16 - 20	///
21 - 30	595
31	/

項目	説明
mfsr (mfsr)	*RT*、 *SR*

説明

mfsr 命令は、セグメント・レジスター (SR) の内容をターゲット汎用レジスター (GPR) *RT* にコピーします。

mfsr 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。 レコード (Rc) ビットが 1 に設定されている場合、条件レジスター・フィールド 0 は未定義です。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
SR	操作のソース・セグメント・レジスターを指定します。

例

以下のコードは、セグメント・レジスター 7 の内容を GPR 6 にコピーします。

```
# Assume that the source Segment Register is SR 7.
# Assume that GPR 6 is the target register.
mfsr 6,7
# GPR 6 now holds a copy of the contents of Segment Register 7.
```

セキュリティー

mfsr 命令は、PowerPC® アーキテクチャーでのみ特権を付与されます。

関連概念

mfsri (セグメント・レジスター間接からの移動) 命令

処理とストレージ

プロセッサーは、メイン・メモリーおよびレジスターにデータを保管します。

POWER® ファミリーと PowerPC® 命令の機能の違い

POWER® ファミリーと PowerPC® 命令は、POWER® ファミリーと PowerPC® プラットフォームで同じ命令コードを共有しますが、機能定義は異なります。

mfsri (セグメント・レジスター間接からの移動) 命令

目的

計算されたセグメント・レジスターの内容を汎用レジスターにコピーします。

注: mfsri 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	627
31	rc

POWER® ファミリー

mfsri (mfsri) *RS*、 *RA*、 *RB*

説明

mfsri 命令は、汎用レジスター (GPR) *RA* の計算内容のビット 0 から 3 で指定されたセグメント・レジスター (SR) の内容を GPR *RS* にコピーします。 *RA* が 0 でない場合、GPR *RA* 内の指定ビットは、 *RA* の元の内容を GPR *RB* に加算し、その合計を *RA* に入れることによって計算されます。 *RA = RS* の場合、合計は *RA* に入れられません。

mfsri 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。 レコード (Rc) ビットが 1 に設定されている場合、条件レジスター・フィールド 0 は未定義です。

パラメーター

項目	説明
RS	操作のターゲット汎用レジスターを指定します。
RA	SR 計算のソース汎用レジスターを指定します。

項目	説明
rb	SR 計算のソース汎用レジスターを指定します。

例

以下のコードは、GPR 4 と GPR 5 の内容の合計の最初の 4 ビットによって指定されたセグメント・レジスターの内容を GPR 6 にコピーします。

```
# Assume that GPR 4 contains 0x9000 3000.
# Assume that GPR 5 contains 0x1000 0000.
# Assume that GPR 6 is the target register.
mfsri 6,5,4
# GPR 6 now contains the contents of Segment Register 10.
```

関連概念

処理とストレージ
プロセッサーは、メイン・メモリーおよびレジスターにデータを保管します。

mfsrin (Move from Segment Register Indirect) 命令

mfsrin (Move from Segment Register Indirect) 命令

目的

指定されたセグメント・レジスターの内容を汎用レジスターにコピーします。

注: mfsrin 命令は、PowerPC ® アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	///
16 - 20	RB
21 - 30	659
31	/

PowerPC (R)

mfsrin (mfsrin) *RT*、 *RB*

説明

mfsrin 命令は、汎用レジスター (GPR) *RB* のビット 0 から 3 で指定されたセグメント・レジスター (SR) の内容を GPR *RT* にコピーします。

mfsrin 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。 レコード (Rc) ビットが 1 に設定されている場合、条件レジスター・フィールド 0 は未定義です。

パラメーター

項目	説明
RT	操作のターゲット汎用レジスターを指定します。
rb	SR 計算のソース汎用レジスターを指定します。

セキュリティー

mfsrin 命令には特権があります。

関連概念

処理とストレージ
プロセッサーは、メイン・メモリーおよびレジスターにデータを保管します。

POWER® ファミリー用の mfspr 拡張ニーモニック
POWER® ファミリー用の mfspr 拡張ニーモニック

mfsrin (Move from Segment Register Indirect) 命令

mtcrf (条件レジスター・フィールドへの移動) 命令

目的

汎用レジスターの内容を、フィールド・マスクの制御下にある条件レジスターにコピーします。

構文

ビット	値
0 - 5	31
6 - 10	RS
11	/
12 - 19	FXM (M)
20	/
21 - 30	144
31	rc

項目	説明
mtcrf	*FXM*、 *RS*

詳しくは、「Extended Mnemonics of Condition Register Logical Instructions 」を参照してください。

説明

mtcrf 命令は、ソース汎用レジスター (GPR) *RS* の内容を、フィールド・マスク *FXM* の制御下で条件レジスターにコピーします。

フィールド・マスク *FXM* は、次のように定義されます。

ビット	説明
12	CR 00 から 03 は、GPR *RS* 00-03 の内容で更新されました。
13	CR 04-07 が GPR *RS* 04-07 の内容で更新されました。
14	CR 08-11 が更新され、GPR *RS* 08-11 の内容が追加されました。
15	CR 12-15 が更新され、GPR *RS* 12-15 の内容が追加されました。

ビット	説明
16	CR 16 から 19 は、GPR *RS* 16 から 19 の内容で更新されました。
17	CR 20-23 は、GPR *RS* 20-23 の内容で更新されました。
18	CR 24-27 は、GPR *RS* 24-27 の内容で更新されました。
19	CR 28-31 が更新され、GPR *RS* 28-31 の内容が追加されました。

mtcrf 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。

mtcrf 命令の推奨形式では、 *FXM* フィールドに 1 ビットのみが設定されます。

パラメーター

項目	説明
FXM (FXM)	フィールド・マスクを指定します。
RS	操作のソース汎用レジスターを指定します。

例

以下のコードは、GPR 5 のビット 00 から 03 を条件レジスター・フィールド 0 にコピーします。

```
# Assume GPR 5 contains 0x7542 FFEE.
# Use the mask for Condition Register
# Field 0 (0x80 = b'1000 0000').
mtcrf 0x80,5
# Condition Register Field 0 now contains b'0111'.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

特殊目的のレジスター命令との間での固定小数点移動
この命令は、ある特殊目的レジスター (SPR) の内容を別の SPR または汎用レジスター (GPR) に移動します。これには、非特権命令と特権命令の両方が含まれます。

mtfsb0 (Move to FPSCR Bit 0) 命令

目的

指定された浮動小数点状況および制御レジスター・ビットを 0 に設定します。

構文

ビット	VALUE
0 - 5	63
6 - 10	BT
11 - 15	///
16 - 20	///

ビット	VALUE
21 - 30	70
31	rc

項目	説明
mtfsb0	*BT*
mtfsb0.	*BT*

説明

mtfsb0 命令は、*BT* によって指定された浮動小数点状況および制御レジスター・ビットを 0 に設定します。

mtfsb0 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明		
構文形式	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
mtfsb0	なし	0	なし
mtfsb0.	なし	1	FX、FEX、VX、OX

mtfsb0 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は、条件レジスター・フィールド 1 の浮動小数点例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

注: ビット 1-2 を明示的に設定またはリセットすることはできません。

パラメーター

項目	説明
BT	演算によって設定される浮動小数点状況および制御レジスター・ビットを指定します。

例

1. 以下のコードは、浮動小数点状況および制御レジスター浮動小数点オーバーフロー例外ビット (ビット 3) を 0 に設定します。

```
mtfsb0 3
# Now bit 3 of the Floating-Point Status and Control
# Register is 0.
```

2. 以下のコードは、浮動小数点状況および制御レジスター浮動小数点オーバーフロー例外ビット (ビット 3) を 0 に設定し、演算の結果を反映するように条件レジスター・フィールド 1 を設定します。

```
mtfsb0. 3
# Now bit 3 of the Floating-Point Status and Control
# Register is 0.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点レジスターの内容の解釈

32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

mtfsb1 (FPSCR ビット 1 への移動) 命令

目的

指定された浮動小数点状況および制御レジスター・ビットを 1 に設定します。

構文

ビット	VALUE
0-5	63
6 ～ 10	BT
11-15	///
16-20	///
21-30	38
31	rc

項目	説明
mtfsb1	*BT*
mtfsb1.	*BT*

説明

mtfsb1 命令は、 *BT* で指定された浮動小数点状況制御レジスター (FPSCR) ビットを 1 に設定します。

mtfsb1 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明		
構文形式	**FPSCR ビット**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
mtfsb1	なし	0	なし
mtfsb1.	なし	1	FX、FEX、VX、OX

mtfsb1 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は、条件レジスター・フィールド 1 の浮動小数点例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

注: ビット 1-2 を明示的に設定またはリセットすることはできません。

パラメーター

項目	説明
BT	FPSCR ビットが命令によって 1 に設定されることを指定します。

例

1. 以下のコードは、浮動小数点状況および制御レジスターのビット 4 を 1 に設定します。

```
mtfsb1 4
# Now bit 4 of the Floating-Point Status and Control
# Register is set to 1.
```

2. 以下のコードは、浮動小数点状況および制御レジスター・オーバーフロー例外ビット (ビット 3) を 1 に設定し、演算の結果を反映するように条件レジスター・フィールド 1 を設定します。

```
mtfsb1. 3
# Now bit 3 of the Floating-Point Status and Control
# Register is set to 1.
```

関連概念

浮動小数点プロセッサー

浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点レジスターの内容の解釈

32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

mtfsf (FPSCR フィールドへの移動) 命令

目的

浮動小数点レジスターの内容を、フィールド・マスクの制御下で浮動小数点状況および制御レジスターにコピーします。

構文

ビット	VALUE
0 - 5	63
6	/
7 - 14	FLM (LM)
15	/
16 - 20	FRB
21 - 30	771
31	rc

項目	説明
mtfsf (mtfsf)	*FLM*、 *FRB*
mtfsf。	*FLM*、 *FRB*

詳しくは、「Extended Mnemonics of Condition Register Logical Instructions 」を参照してください。

説明

mtfsf 命令は、浮動小数点レジスター (FPR) *FRB* の内容のビット 32 から 63 を、 *FLM* によって指定されたフィールド・マスクの制御下で浮動小数点状況および制御レジスターにコピーします。

フィールド・マスク *FLM* は、次のように定義されます。

```
Bit   Description
```

```
7   FPSCR 00-03 is updated with the contents of FRB 32-35.
```

```
8   FPSCR 04-07 is updated with the contents of FRB 36-39.
```

```
9   FPSCR 08-11 is updated with the contents of FRB 40-43.
```

```
10   FPSCR 12-15 is updated with the contents of FRB 44-47.
```

```
11   FPSCR 16-19 is updated with the contents of FRB 48-51.
```

```
12   FPSCR 20-23 is updated with the contents of FRB 52-55.
```

```
13   FPSCR 24-27 is updated with the contents of FRB 56-59.
```

```
14   FPSCR 28-31 is updated with the contents of FRB 60-63.
```

mtfsf 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	**説明**		
構文形式	**FPSCR ビット**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
mtfsf (mtfsf)	なし	0	なし
mtfsf。	なし	1	FX、FEX、VX、OX

mtfsf 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は、条件レジスター・フィールド 1 の浮動小数点例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

注: FPSCR 0-3 を指定する場合、一部のビットは明示的に設定またはリセットできません。

パラメーター

項目	**説明**
FLM (FLM)	フィールド・マスクを指定します。
FRB	操作のソース浮動小数点レジスターを指定します。

例

1. 以下のコードは、浮動小数点レジスター 5 のビット 32 から 35 の内容を浮動小数点状況および制御レジスター・フィールド 0 にコピーします。

```
# Assume bits 32-63 of FPR 5
# contain 0x3000 3000.
```

```
mtfsf 0x80,5
# Floating-Point Status and Control Register
# Field 0 is set to b'0001'.
```

2. 以下のコードは、浮動小数点レジスター 5 ビット 32 から 43 の内容を浮動小数点状況および制御レジスター・フィールド 0 から 2 にコピーし、演算の結果を反映するように条件レジスター・フィールド 1 を設定します。

```
# Assume bits 32-63 of FPR 5
# contains 0x2320 0000.
mtfsf. 0xE0,5
# Floating-Point Status and Control Register Fields 0-2
# now contain b'0010 0011 0010'.
# Condition Register Field 1 now contains 0x2.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点レジスターの内容の解釈
32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

mtfsfi (Move to FPSCR Field Immediate) 命令

目的

指定された浮動小数点状況および制御レジスター・フィールドに即時値をコピーします。

構文

ビット	VALUE
0 - 5	63
6 - 8	BF
9 - 10	//
11 - 15	///
16 - 19	U
20	/
21 - 30	134
31	rc

項目	説明
mtfsfi (mtfsfi)	*BF*、 *I*
mtfsfi。	*BF*、 *I*

説明

mtfsfi 命令は、 *I* パラメーターで指定された即時値を、 *BF* で指定された浮動小数点状況および制御レジスター (Floating-Point Status and Control Register) フィールドにコピーします。 浮動小数点状況および制御レジスターの他のフィールドは影響を受けません。

mtfsfi 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 1 に対して異なる影響を与えます。

項目	説明		
構文 Form	**FPSCR ビット**	**レコード ビット (RC)**	**条件 レジスター・フィールド 1**
mtfsfi (mtfsfi)	なし	0	なし
mtfsfi。	なし	1	FX、FEX、VX、OX

mtfsfi 命令の 2 つの構文形式は、浮動小数点状況および制御レジスター・フィールドには影響を与えません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は、条件レジスター・フィールド 1 の浮動小数点例外 (FX)、浮動小数点使用可能例外 (FEX)、浮動小数点無効演算例外 (VX)、および浮動小数点オーバーフロー例外 (OX) ビットに影響します。

注: FPSCR 0-3 を指定する場合、一部のビットは明示的に設定またはリセットできません。

パラメーター

項目	説明
BF	操作のターゲット浮動小数点状況および制御レジスター・フィールドを指定します。
I	操作のソース即値を指定します。

例

1. 以下のコードは、浮動小数点状況および制御レジスター・フィールド 6 を b '0100' に設定します。

```
mtfsfi 6,4
# Floating-Point Status and Control Register Field 6
# is now b'0100'.
```

2. 以下のコードは、浮動小数点状況および制御レジスター・フィールド 0 を b '0100' に設定し、操作の結果を反映するように条件レジスター・フィールド 1 を設定します。

```
mtfsfi. 0,1
# Floating-Point Status and Control Register Field 0
# is now b'0001'.
# Condition Register Field 1 now contains 0x1.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点レジスターの内容の解釈
32 個の 64 ビット浮動小数点レジスターがあります。 浮動小数点レジスターは、命令を実行するために使用されます。

mtbuilf (Move to One Condition Register Field) 命令

目的

汎用レジスターの内容を、フィールド・マスクの制御下にある 1 つの条件レジスター・フィールドにコピーします。

構文

ビット	値
0 - 5	31
6 - 10	RT

ビット	値
11	/
12 - 19	FXM (M)
20	/
21 - 30	144
31	/

項目	説明
民主主義者	FXM、 RS

説明

mtids 命令は、ソース汎用レジスター (GPR) *RS* の内容を、フィールド・マスク *FXM* の制御下で条件レジスターにコピーします。

フィールド・マスク *FXM* は、次のように定義されます。

ビット	説明
12	CR 00 から 03 は、GPR *RS* 00-03 の内容で更新されました。
13	CR 04-07 が GPR *RS* 04-07 の内容で更新されました。
14	CR 08-11 が更新され、GPR *RS* 08-11 の内容が追加されました。
15	CR 12-15 が更新され、GPR *RS* 12-15 の内容が追加されました。
16	CR 16 から 19 は、GPR *RS* 16 から 19 の内容で更新されました。
17	CR 20-23 は、GPR *RS* 20-23 の内容で更新されました。
18	CR 24-27 は、GPR *RS* 24-27 の内容で更新されました。
19	CR 28-31 が更新され、GPR *RS* 28-31 の内容が追加されました。

mt くり 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
FXM (FXM)	フィールド・マスクを指定します。
RS	操作のソース汎用レジスターを指定します。

例

以下のコードは、GPR 5 のビット 00 から 03 を条件レジスター・フィールド 0 にコピーします。

```
# Assume GPR 5 contains 0x7542 FFEE.
# Use the mask for Condition Register
# Field 0 (0x80 = b'1000 0000').
mtocrf 0x80,5
# Condition Register Field 0 now contains b'0111'.
```

関連概念

固定小数点プロセッサー

固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

特殊目的のレジスター命令との間での固定小数点移動

この命令は、ある特殊目的レジスター (SPR) の内容を別の SPR または汎用レジスター (GPR) に移動します。これには、非特権命令と特権命令の両方が含まれます。

mtspr (特殊目的レジスターへの移動) 命令

目的

汎用レジスターの内容を、特殊目的レジスターにコピーします。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 20	SPR
21 - 30	467
31	rc

項目	説明
mtspr (mtspr)	*SPR*、 *RS*

注: 特殊目的レジスターは、分割フィールドです。

説明

mtspr 命令は、ソース汎用レジスター *RS* の内容を、ターゲット特殊目的レジスター *SPR* にコピーします。

特殊目的のレジスター ID *SPR* には、以下の表に示す値のいずれかを指定できます。 SPR 番号の 2 つの 5 ビット半分の順序が逆になります。

項目	説明	説明	説明
SPR 値	**SPR 値**	**SPR 値**	**SPR 値**
10 進数	**spr**$^{5:9}$ spr$^{0:4}$	**レジスター名**	**privileged**
1	00000 00001	XER	いいえ
8	00000 01000	LR	いいえ
9	00000 01001	CTR	いいえ
18	00000 10010	DSISR	はい
19	00000 10011	DAR	はい
22	00000 10110	DEC	可 [1]

項目	説明	説明	説明
25	00000 11001	SDR1	はい
26	00000 11010	SRR0	はい
27	00000 11011	SRR1	はい
272	01000 10000	SPRG0	はい
273	01000 10001	SPRG1	はい
274	01000 10010	SPRG2	はい
275	01000 10011	SPRG3	はい
282	01000 11010	EAR	はい
284	01000 11100	TBL	はい
285	01000 11101	TBU (U)	はい
528	10000 10000	IBAT0U	はい
529	10000 10001	IBAT0L	はい
530	10000 10010	IBAT1U	はい
531	10000 10011	IBAT1L	はい
532	10000 10100	IBAT2U	はい
533	10000 10101	IBAT2L	はい
534	10000 10110	IBAT3U	はい
535	10000 10111	IBAT3L	はい
536	10000 11000	DBAT0U	はい
537	10000 11001	DBAT0L	はい
538	10000 11010	DBAT1U	はい
539	10000 11011	DBAT1L	はい
540	10000 11100	DBAT2U	はい
541	10000 11101	DBAT2L	はい
542	10000 11110	DBAT3U	はい
543	10000 11111	DBAT3L	はい
0	00000 00000	MQ[2]	いいえ
20	00000 10100	RTCU[2]	はい
21	00000 10101	RTCL (RTCL)[2]	はい

1. DEC レジスターへの移行は、PowerPC ® アーキテクチャーおよび POWER ® ファミリー・アーキテクチャーで特権を付与されます。 ただし、 *DEC* レジスターからの移動 は、PowerPC ® アーキテクチャーでのみ特権が付与されます。
2. 2Supported は、POWER ® ファミリー・アーキテクチャーでのみサポートされます。

SPR フィールドに SPR 値テーブルにリストされている値以外の値が含まれている場合、命令フォームは無効です。

mtspr 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
SPR	操作のターゲット特殊目的レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。

例

以下のコードは、GPR 5 の内容をリンク・レジスターにコピーします。

```
# Assume GPR 5 holds 0x1000 00FF.
mtspr 8,5
# The Link Register now holds 0x1000 00FF.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

特殊目的のレジスター命令との間での固定小数点移動
この命令は、ある特殊目的レジスター (SPR) の内容を別の SPR または汎用レジスター (GPR) に移動します。これには、非特権命令と特権命令の両方が含まれます。

mul (乗算) 命令

目的

2 つの汎用レジスターの内容を乗算し、その結果を 3 番目の汎用レジスターに保管します。

注: mul 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21	大江
22 - 30	107
31	rc

POWER® ファミリー

mul	*RT*、*RA*、*RB*
ミュール	*RT*、*RA*、*RB*
ムロ	*RT*、*RA*、*RB*
ムロ	*RT*、*RA*、*RB*

説明

mul 命令は、汎用レジスター (GPR) *RA* および GPR *RB* の内容を乗算し、結果のビット 0 から 31 をターゲット GPR *RT* に保管し、結果のビット 32 から 63 を MQ レジスターに保管します。

mul 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
mul	0	なし	0	なし
ミュール	0	なし	1	LT、GT、EQ、SO
ムロ	1	SO、OV	0	なし
ムロ	1	SO、OV	1	LT、GT、EQ、SO

mul 命令の 4 つの構文形式は、固定小数点例外レジスターのカリー・ビット (CA) には影響しません。 シンタックス・フォームがオーバーフロー例外 (OE) ビットを 1 に設定すると、製品が 32 ビットより大きい場合、命令は固定小数点例外レジスターのサマリー・オーバーフロー (SO) ビットとオーバーフロー (OV) ビットを 1 に設定します。 構文形式でレコード (Rc) ビットが 1 に設定されている場合、条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、および「等しい (EQ) ゼロ・ビット」は、 MQ レジスターの下位 32 ビットの結果を反映します。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容と GPR 10 の内容を乗算し、その結果を GPR 6 および MQ レジスターに保管します。

```
# Assume GPR 4 contains 0x0000 0003.
# Assume GPR 10 contains 0x0000 0002.
mul 6,4,10
# MQ Register now contains 0x0000 0006.
# GPR 6 now contains 0x0000 0000.
```

2. 以下のコードは、GPR 4 の内容に GPR 10 の内容を乗算し、その結果を GPR 6 および MQ レジスターに保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0x0000 4500.
# Assume GPR 10 contains 0x8000 7000.
mul. 6,4,10
# MQ Register now contains 0x1E30 0000.
# GPR 6 now contains 0xFFFF DD80.
# Condition Register Field 0 now contains 0x4.
```

3. 以下のコードは、GPR 4 の内容に GPR 10 の内容を乗算し、結果を GPR 6 および MQ レジスターに保管し、演算の結果を反映するために固定小数点例外レジスターに要約オーバーフロー・ビットとオーバーフロー・ビットを設定します。

```
# Assume GPR 4 contains 0x0000 4500.
# Assume GPR 10 contains 0x8000 7000.
# Assume XER = 0.
```

```
mulo 6,4,10
# MQ Register now contains 0x1E30 0000.
# GPR 6 now contains 0xFFFF DD80.
# XER now contains 0xc000 0000.
```

4. 以下のコードは、GPR 4 の内容に GPR 10 の内容を乗算し、その結果を GPR 6 および MQ レジスターに保管し、演算の結果を反映するために、固定小数点例外レジスターおよび条件レジスター・フィールド 0 の要約オーバーフロー、オーバーフロー、およびキャリー・ビットを設定します。

```
# Assume GPR 4 contains 0x0000 4500.
# Assume GPR 10 contains 0x8000 7000.
# Assume XER = 0.
mulo. 6,4,10
# MQ Register now contains 0x1E30 0000.
# GPR 6 now contains 0xFFFF DD80.
# Condition Register Field 0 now contains 0x5.
# XER now contains 0xc000 0000.
```

mulhd (乗算ハイ・ダブル・ワード) 命令

目的

2 つの 64 ビット値を乗算します。 結果の上位 64 ビットを 1 つのレジスターに入れます。

構文

ビット	VALUE
0-5	31
6 〜 10	D
11-15	A
16-20	B
21	0
22-30	73
31	rc

POWER® ファミリー

マルチド *RT*、 *RA*、 *RB* (Rc=0)

マルチド *RT*、 *RA*、 *RB* (Rc=1)

説明

64 ビット・オペランドは、汎用レジスター (GPR) *RA* および *RB* の内容です。 オペランドの 128 ビットの積の上位 64 ビットは、 *RT* に入れられます。

オペランドと積の両方が符号付き整数として解釈されます。

この命令は、絶対値が小さいオペランドが *RB* に含まれていると、インプリメンテーションによってはより速く実行できます。

パラメーター

項目	説明
RT	計算の結果のターゲット汎用レジスターを指定します。
RA	オペランドのソース汎用レジスターを指定します。

項目	説明
rb	オペランドのソース汎用レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

mulhdu (Multiply High Double Word Unsigned) 命令

目的

2 つの符号なし 64 ビット値を乗算します。 結果の上位 64 ビットを 1 つのレジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	D
11 - 15	A
16 - 20	B
21	0
22 - 30	9
31	rc

POWER® ファミリー

マルチゥ *RT*、 *RA*、 *RB* (Rc=0)

ムルドゥ *RT*、 *RA*、 *RB* (Rc=1)

説明

オペランドと積は両方とも符号なし整数として解釈されます。ただし、Rc = 1 (**mulhw**) の場合を除きます。 条件レジスター 0 フィールドの最初の 3 ビットは、結果の符号付き比較によってゼロに設定されます。

64 ビット・オペランドは、 *RA* および *RB* の内容です。 オペランドの 128 ビットの積の下位 64 ビットは、*RT* に入れられます。

変更されたその他のレジスター:

- 条件レジスター (CR0 フィールド):

  影響を受ける: LT、 GT、EQ、SO (Rc = 1 の場合)

  注: CR0 ビット LT、 GT、および EQ の設定はモードに依存し、64 ビット結果のオーバーフローを反映します。

この命令は、絶対値が小さいオペランドが *RB* に含まれていると、インプリメンテーションによってはより速く実行できます。

パラメーター

項目	説明
RT	計算の結果のターゲット汎用レジスターを指定します。

項目	説明
RA	多重化のソース汎用レジスターを指定します。
rb	乗数のソース汎用レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

mulhw (乗算高位ワード) 命令

目的

2 つの 32 ビット整数の 64 ビット積の最上位 32 ビットを計算します。

注: mulhw 命令は、PowerPC ® アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21	/
22 - 30	75
31	rc

PowerPC (R)

マルチウ (mulhw) *RT*、 *RA*、 *RB*

mulhw: *RT*、 *RA*、 *RB*

説明

mulhw 命令は、汎用レジスター (GPR) *RA* および GPR *RB* の内容を多重化し、64 ビット製品の最上位 32 ビットをターゲット GPR *RT* に入れます。 オペランドと積の両方が符号付き整数として解釈されます。

mulhw 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明	
構文形式	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
マルチウ (mulhw)	0	なし
mulhw:	1	LT、GT、EQ、SO

構文書式でレコード (Rc) ビットが 1 に設定されている場合、条件レジスター・フィールド 0 の「より小 (LT)」ゼロ、「より大 (GT)」ゼロ、および「等しい (EQ)」ゼロ・ビットは、GPR *RT* に入れられた結果を反映し、要約オーバーフロー (SO) ビットが XER から XER にコピーされます。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容に GPR 10 の内容を乗算し、その結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x0000 0003.
# Assume GPR 10 contains 0x0000 0002.
mulhw 6,4,10
# GPR 6 now contains 0x0000 0000.
```

2. 以下のコードは、GPR 4 の内容に GPR 10 の内容を乗算し、その結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0x0000 4500.
# Assume GPR 10 contains 0x8000 7000.
# Assume XER(SO) = 0.
mulhw. 6,4,10
# GPR 6 now contains 0xFFFF DD80.
# Condition Register Field 0 now contains 0x4.
```

mulhwu (Multiply High Word Unsigned) 命令

目的

2 つの符号なし 32 ビット整数の 64 ビット積の最上位 32 ビットを計算します。

注: mulhwu 命令は、PowerPC ® アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21	/
22 - 30	11
31	rc

PowerPC (R)

マルチウ (mulhwu)	*RT*、*RA*、*RB*
mulhwu。	*RT*、*RA*、*RB*

説明

mulhwu 命令は、汎用レジスター (GPR) *RA* および GPR *RB* の内容を乗算し、64 ビット製品の最上位の 32 ビットをターゲット GPR *RT* に配置します。 オペランドと積の両方とも、符号なし整数として解釈されます。

> **注:** 演算では結果は符号なし整数として扱われますが、より小 (LT) ゼロ、より大 (GT) ゼロ、および等しい (EQ) ゼロ・ビットに対する条件レジスター・フィールド 0 の設定は符号付き整数として解釈されます。

mulhwu 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	**説明**	
構文形式	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
マルチウ (mulhwu)	0	なし
mulhwu。	1	LT、GT、EQ、SO

構文書式でレコード (Rc) ビットが 1 に設定されている場合、条件レジスター・フィールド 0 の「より小 (LT)」ゼロ、「より大 (GT)」ゼロ、および「等しい (EQ)」ゼロ・ビットは、GPR *RT* に入れられた結果を反映し、要約オーバーフロー (SO) ビットが XER から XER にコピーされます。

パラメーター

項目	**説明**
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容に GPR 10 の内容を乗算し、その結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x0000 0003.
# Assume GPR 10 contains 0x0000 0002.
mulhwu 6,4,10
# GPR 6 now contains 0x0000 0000.
```

2. 以下のコードは、GPR 4 の内容に GPR 10 の内容を乗算し、その結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0x0000 4500.
# Assume GPR 10 contains 0x8000 7000.
# Assume XER(SO) = 0.
mulhwu. 6,4,10
# GPR 6 now contains 0x0000 2280.
# Condition Register Field 0 now contains 0x4.
```

mulld (乗算-低ダブルワード) 命令

目的

2 つの 64 ビット値を乗算します。 結果の下位 64 ビットを 1 つのレジスターに入れます。

構文

ビット	**VALUE**
0 - 5	31

ビット	VALUE
6 - 10	D
11 - 15	A
16 - 20	B
21	大江
22 - 30	233
31	rc

POWER® ファミリー

マルチルド *RT*、 *RA*、 *RB* (OE=0 Rc=0)

mulld。 *RT*、 *RA*、 *RB* (OE=0 Rc=1)

マルチド *RT*、 *RA*、 *RB* (OE=1 Rc=0)

mulldo。 *RT*、 *RA*、 *RB* (OE=1 Rc=1)

説明

64 ビット・オペランドは、汎用レジスター (GPR) *RA* および *RB* の内容です。 オペランドの 128 ビットの積の下位 64 ビットは、 *RT* に入れられます。

オペランドと積の両方が符号付き整数として解釈されます。 製品の下位 64 ビットは、オペランドが符号付き 64 ビット整数と見なされるか、符号なし 64 ビット整数と見なされるかには関係ありません。 OE = 1 の場合 (**mulldo** および **mulldo** 。 製品を 64 ビットで表示できない場合は、OV が設定されます。

この命令は、絶対値が小さいオペランドが *RB* に含まれていると、インプリメンテーションによってはより速く実行できます。

変更されたその他のレジスター:

- 条件レジスター (CR0 フィールド):

  影響を受ける: LT、 GT、EQ、SO (Rc = 1 の場合)

  注: オーバーフローが発生した場合、 CR0 フィールドは無限に正確な結果を反映しないことがあります (下記の XER を参照)。
- XER:

  影響を受ける: SO、OV (OE = 1 の場合)

  注: XER 内の影響を受けるビットの設定はモードに依存せず、64 ビット結果のオーバーフローを反映します。

パラメーター

項目	説明
RT	計算の rsult のターゲット汎用レジスターを指定します。
RA	オペランドのソース汎用レジスターを指定します。
rb	オペランドのソース汎用レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

mulli または muli (Multiply Low Immediate) 命令

目的

汎用レジスターの内容を 16 ビットの符号付き整数で乗算し、その結果を別の汎用レジスターに保管します。

構文

ビット	VALUE
0 - 5	07
6 - 10	RT
11 - 15	RA
16 - 31	SI

PowerPC (R)

マルチリ (mulli) *RT*、 *RA*、 *SI*

POWER® ファミリー

ムリ (muli) *RT*、 *RA*、 *SI*

説明

mulli および **muli** 命令符号により、 *SI* フィールドが 32 ビットに拡張され、拡張された値に汎用レジスター (GPR) *RA* の内容が乗算されます。 64 ビット製品の最下位 32 ビットは、ターゲット GPR *RT* に配置されます。

mulli 命令と **muli** 命令には 1 つの構文形式があり、条件レジスター・フィールド 0 または固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。
SI	演算のための 16 ビットの符号付き整数を指定します。

例

以下のコードは、GPR 4 の内容に 10 を乗算し、その結果を GPR 6 に入れます。

```
# Assume GPR 4 holds 0x0000 3000.
mulli 6,4,10
# GPR 6 now holds 0x0001 E000.
```

mullw または muls (Multiply Low Word) 命令

目的

2 つの 32 ビット整数の 64 ビット積の下位 32 ビットを計算します。

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21	大江
22 - 30	235
31	rc

PowerPC (R)

マルチロー (mullw) *RT*、*RA*、*RB*

マルチウ *RT*、*RA*、*RB*

マルチウ (mullwo) *RT*、*RA*、*RB*

マルチル *RT*、*RA*、*RB*

POWER® ファミリー

マルチボリューム *RT*、*RA*、*RB*

マルチボリューム *RT*、*RA*、*RB*

マルチソ *RT*、*RA*、*RB*

mulso。 *RT*、*RA*、*RB*

説明

mullw および **muls** 命令により、汎用レジスター (GPR) *RA* の内容に GPR *RB* の内容が乗算され、結果の最下位 32 ビットがターゲット GPR *RT* に入れられます。

mullw 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

muls 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
マルチロー (mullw)	0	なし	0	なし
マルチウ	0	なし	1	LT、GT、EQ
マルチウ (mullwo)	1	SO、OV	0	なし
マルチル	1	SO、OV	1	LT、GT、EQ
マルチボリューム	0	なし	0	なし
マルチボリューム	0	なし	1	LT、GT、EQ

項目	説明			
マルチソ	1	SO、OV	0	なし
mulso。	1	SO、OV	1	LT、GT、EQ

mullw 命令の 4 つの構文形式、および **muls** 命令の 4 つの構文形式は、固定小数点例外レジスターのカリー・ビット (CA) には影響しません。 構文形式がオーバーフロー例外 (OE) ビットを 1 に設定する場合、結果が 32 ビットで表現するには大きすぎると、命令は固定小数点例外レジスターの要約オーバーフロー (SO) およびオーバーフロー (OV) ビットを 1 に設定します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容に GPR 10 の内容を乗算し、その結果を GPR 6 に保管します。

```
# Assume GPR 4 holds 0x0000 3000.
# Assume GPR 10 holds 0x0000 7000.
mullw 6,4,10
# GPR 6 now holds 0x1500 0000.
```

2. 以下のコードは、GPR 4 の内容に GPR 10 の内容を乗算し、その結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 holds 0x0000 4500.
# Assume GPR 10 holds 0x0000 7000.
# Assume XER(SO) = 0.
mullw. 6,4,10
# GPR 6 now holds 0x1E30 0000.
# Condition Register Field 0 now contains 0x4.
```

3. 以下のコードは、GPR 4 の内容に GPR 10 の内容を乗算し、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターにサマリー・オーバーフロー・ビットとオーバーフロー・ビットを設定します。

```
# Assume GPR 4 holds 0x0000 4500.
# Assume GPR 10 holds 0x0007 0000.
# Assume XER = 0.
mullwo 6,4,10
# GPR 6 now holds 0xE300 0000.
# XER now contains 0xc000 0000
```

4. 以下のコードは、GPR 4 の内容に GPR 10 の内容を乗算し、その結果を GPR 6 に保管し、演算の結果を反映するために、要約オーバーフロー、オーバーフロー、およびキャリー・ビットを固定小数点例外レジスターおよび条件レジスター・フィールド 0 に設定します。

```
# Assume GPR 4 holds 0x0000 4500.
# Assume GPR 10 holds 0x7FFF FFFF.
# Assume XER = 0.
mullwo. 6,4,10
# GPR 6 now holds 0xFFFF BB00.
# XER now contains 0xc000 0000
# Condition Register Field 0 now contains 0x9.
```

nabs (負の絶対値) 命令

目的

汎用レジスターの内容の絶対値を否定し、その結果を別の汎用レジスターに格納します。

注: nabs 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	///
21	大江
22 - 30	488
31	rc

POWER® ファミリー

nabs (nabs) *RT*、*RA*

nabs。 *RT*、*RA*

ナブソ *RT*、*RA*

nabso。 *RT*、*RA*

説明

nabs 命令は、汎用レジスター (GPR) *RA* の内容の負の絶対値をターゲット GPR *RT* に入れます。

nabs 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
nabs (nabs)	0	なし	0	なし
nabs。	0	なし	1	LT、GT、EQ、SO
ナブソ	1	SO、OV	0	なし
nabso。	1	SO、OV	1	LT、GT、EQ、SO

nabs 命令の 4 つの構文形式は、固定小数点例外レジスターのカリー・ビット (CA) には影響しません。 構文形式がオーバーフロー例外 (OE) ビットを 1 に設定した場合、要約オーバーフロー (SO) ビットは変更されず、オーバーフロー (OV) ビットはゼロに設定されます。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容の負の絶対値を取り、結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x0000 3000.
nabs 6,4
# GPR 6 now contains 0xFFFF D000.
```

2. 以下のコードは、GPR 4 の内容の負の絶対値を取り、結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xFFFF FFFF.
nabs. 6,4
# GPR 6 now contains 0xFFFF FFFF.
```

3. 次のコードは、GPR 4 の内容の負の絶対値を取り、結果を GPR 6 に保管し、固定小数点例外レジスターのオーバーフロー・ビットを 0 に設定します。

```
# Assume GPR 4 contains 0x0000 0001.
nabso 6,4
# GPR 6 now contains 0xFFFF FFFF.
```

4. 以下のコードは、GPR 4 の内容の負の絶対値を取り、結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定し、固定小数点例外レジスターのオーバーフロー・ビットを 0 に設定します。

```
# Assume GPR 4 contains 0x8000 0000.
nabso 6,4
# GPR 6 now contains 0x8000 0000.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

nand (NAND) 命令

目的

2 つの汎用レジスターの内容を AND 演算した結果を論理的に補完し、その結果を別の汎用レジスターに保管します。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA

ビット	VALUE
16 - 20	RB
21 - 30	476
31	rc

項目	説明
nand (Nand)	*RA*、 *RS*、 *RB*
Nand。	*RA*、 *RS*、 *RB*

説明

Nand 命令は、汎用レジスター (GPR) の内容「*RS*」 を GPR の内容 要求ブロック と論理的に AND し、結果の補数をターゲット GPR *RA* とはに保管します。

nand 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
nand (Nand)	なし	なし	0	なし
Nand。	なし	なし	1	LT、GT、EQ、SO

nand 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 と GPR 7 の内容を AND 演算した結果を補完し、結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 7 contains 0x789A 789B.
nand 6,4,7
# GPR 6 now contains 0xEFFF CFFF.
```

2. 以下のコードは、GPR 4 および GPR 7 の内容の ANDing の結果を補完し、結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 7 contains 0x789A 789B.
nand. 6,4,7
# GPR 6 now contains 0xCFFF CFFF.
```

関連概念

固定小数点プロセッサー

固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点論理命令
固定小数点論理命令は、論理演算をビット単位で実行します。

neg (否定) 命令

目的

汎用レジスターの内容の算術符号を変更して、その結果を別の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	///
21	大江
22 - 30	104
31	rc

項目	説明
NEG	*RT*、*RA*
ネグ	*RT*、*RA*
ネゴ (nego)	*RT*、*RA*
nego:	*RT*、*RA*

説明

neg 命令は、汎用レジスター (GPR) *RA* の内容の 1 の補数に 1 を加算し、その結果を GPR *RT* に保管します。

GPR *RA* に最も負の数値 (つまり、 0x8000 0000) が含まれている場合、命令の結果は最も負の数値になり、OE が 1 の場合は固定小数点例外レジスターのオーバーフロー・ビットをシグナル通知します。

neg 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
NEG	0	なし	0	なし
ネグ	0	なし	1	LT、GT、EQ、SO
ネゴ (nego)	1	SO、OV	0	なし
nego:	1	SO、OV	1	LT、GT、EQ、SO

neg 命令の 4 つの構文形式は、固定小数点例外レジスターのカリー・ビット (CA) には影響しません。 構文形式がオーバーフロー例外 (OE) ビットを 1 に設定する場合、命令は固定小数点例外レジスターの要約オーバーフロー (SO) およびオーバーフロー (OV) ビットに影響します。 構文形式がレコード (Rc) ビットを 1

に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容を否定し、結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x9000 3000.
neg 6,4
# GPR 6 now contains 0x6FFF D000.
```

2. 以下のコードは、GPR 4 の内容を否定し、結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0x789A 789B.
neg. 6,4
# GPR 6 now contains 0x8765 8765.
```

3. 以下のコードは、GPR 4 の内容を否定し、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスター要約オーバーフローとオーバーフロー・ビットを設定します。

```
# Assume GPR 4 contains 0x9000 3000.
nego 6,4
# GPR 6 now contains 0x6FFF D000.
```

4. 以下のコードは、GPR 4 の内容を否定し、結果を GPR 6 に保管し、条件レジスター・フィールド 0 と固定小数点例外レジスター要約オーバーフローおよびオーバーフロー・ビットを設定して、演算の結果を反映します。

```
# Assume GPR 4 contains 0x8000 0000.
nego. 6,4
# GPR 6 now contains 0x8000 0000.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

非 (NOR) 命令

目的

2 つの汎用レジスターの内容を OR 演算した結果を論理的に補完し、その結果を別の汎用レジスターに保管します。

構文

ビット	VALUE
0 - 5	31

ビット	VALUE
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	124
31	rc

項目	説明
NOR	*RA*、*RS*、*RB*
または	*RA*、*RS*、*RB*

詳しくは、 固定小数点論理命令の拡張ニーモニック を参照してください。

説明

命令または 命令は、汎用レジスター (GPR) *RS* の内容と GPR *RB* の内容との論理和を取り、補完された結果を GPR *RA* に保管します。

命令または 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
NOR	なし	なし	0	なし
または	なし	なし	1	LT、GT、EQ、SO

命令と 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 および GPR 7 の内容を NORs し、結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 6 contains 0x789A 789B.
nor 6,4,7
# GPR 7 now contains 0x0765 8764.
```

2. 次のコードは、GPR 4 および GPR 7 の内容を NORs し、結果を GPR 6 に保管し、演算の結果を反映するために条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 7 contains 0x789A 789B.
```

```
nor. 6,4,7
# GPR 6 now contains 0x0761 8764.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点論理命令
固定小数点論理命令は、論理演算をビット単位で実行します。

OR (OR) 命令

目的

2 つの汎用レジスターの内容を論理 OR し、その結果を別の汎用レジスターに保管します。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	444
31	rc

項目	説明
または	*RA*、 *RS*、 *RB*
または	*RA*、 *RS*、 *RB*

詳しくは、 固定小数点論理命令の拡張ニーモニック を参照してください。

説明

または 命令は、汎用レジスター (GPR) *RS* の内容と GPR *RB* の内容との論理和を取り、結果を GPR *RA* に保管します。

または 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
または	なし	なし	0	なし
または	なし	なし	1	LT、GT、EQ、SO

または 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 と GPR 7 の内容を論理 OR し、結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 7 contains 0x789A 789B.
or 6,4,7
# GPR 6 now contains 0xF89A 789B.
```

2. 以下のコードは、GPR 4 と GPR 7 の内容を論理 OR し、結果を GPR 6 にロードし、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 7 contains 0x789A 789B.
or. 6,4,7
# GPR 6 now contains 0xF89E 789B.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点論理命令
固定小数点論理命令は、論理演算をビット単位で実行します。

orc (OR with Complement) 命令

目的

汎用レジスターの内容を別の汎用レジスターの内容の補数と論理的に論理和演算し、その結果を第 3 汎用レジスターに保管します。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	412
31	rc

項目	説明
orc	*RA*、 *RS*、 *RB*
orc:	*RA*、 *RS*、 *RB*

説明

orc 命令は、汎用レジスター (GPR) *RS* の内容と GPR *RB* の内容との論理和を取り、その結果を GPR *RA* に保管します。

orc 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
orc	なし	なし	0	なし
orc:	なし	なし	1	LT、GT、EQ、SO

orc 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容と GPR 7 の内容の補数との論理和をとり、その結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 7 contains 0x789A 789B, whose
# complement is 0x8765 8764.
orc 6,4,7
# GPR 6 now contains 0x9765 B764.
```

2. 以下のコードは、GPR 4 の内容と内容 GPR 7 の補数との論理和をとり、結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 7 contains 0x789A 789B, whose
# complement is 0x8765 8764.
orc. 6,4,7
# GPR 6 now contains 0xB765 B764.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点論理命令
固定小数点論理命令は、論理演算をビット単位で実行します。

オリまたはオリル (OR 即値) 命令

目的

16 ビットの符号なし整数を持つ汎用レジスターの内容の下位 16 ビットを論理 OR 演算し、その結果を別の汎用レジスターに保管します。

構文

ビット	VALUE
0 - 5	24
6 - 10	RS
11 - 15	RA
16 - 31	UI

PowerPC (R)

オリ *RA*、 *RS*、 *UI*

POWER® ファミリー

オリル (oril) *RA*、 *RS*、 *UI*

詳しくは、 固定小数点論理命令の拡張ニーモニック を参照してください。

説明

ori および **oril** 命令は、汎用レジスター (GPR) *RS* の内容と、x '0000' と 16 ビット符号なし整数 *UI* の連結との論理 OR 演算を行い、結果を GPR *RA* に入れます。

ori 命令と **oril** 命令には 1 つの構文形式があり、条件レジスター・フィールド 0 や固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
UI	演算用の a16-bit 符号なし整数を指定します。

例

以下のコードは、GPR 4 の内容の下位 16 ビットを 0x0079 と OR し、結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x9000 3000.
ori 6,4,0x0079
# GPR 6 now contains 0x9000 3079.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点論理命令
固定小数点論理命令は、論理演算をビット単位で実行します。

oris または oriu (OR 即値シフト) 命令

目的

16 ビットの符号なし整数を持つ汎用レジスターの内容の上位 16 ビットを論理 OR 演算し、その結果を別の汎用レジスターに保管します。

構文

ビット	VALUE
0 - 5	25
6 - 10	RS
11 - 15	RA
16 - 31	UI

PowerPC (R)

oris (oris) *RA*、 *RS*、 *UI*

POWER® ファミリー

オリウ (oriu) *RA*、 *RS*、 *UI*

説明

oris および **oriu** 命令は、16 ビット符号なし整数、 *UI*、および x '0000' を連結した汎用レジスター (GPR) *RS* の内容を論理的に OR し、結果を GPR *RA* に保管します。

oris および **oriu** 命令には 1 つの構文形式があり、条件レジスター・フィールド 0 または固定小数点例外レジスターには影響しません。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
UI	演算用の a16-bit 符号なし整数を指定します。

例

以下のコードは、GPR 4 の内容の上位 16 ビットを 0x0079 で OR し、結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x9000 3000.
oris 6,4,0x0079
# GPR 6 now contains 0x9079 3000.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点論理命令
固定小数点論理命令は、論理演算をビット単位で実行します。

popcntbd (取り込みカウント・バイトのダブルワード) 命令

目的

プログラムがダブルワード内の 1 ビットの数をカウントできるようにします。

注 : popcntbd 命令は、 POWER5™ アーキテクチャーでのみサポートされます。

構文

ビット	値
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	///
21 – 30	122
31	/

POWER5™

項目	説明
popcntbd	RS、 RA

説明

popcntbd 命令は、レジスター *RS* の各バイトの中の 1 ビットの数をカウントし、そのカウントをレジスター *RA* の対応するバイトに入れます。 数値の範囲は 0 から 8 (両端を含む) です。

popcntbd 命令には 1 つの構文形式があり、特殊レジスターには影響しません。

パラメーター

項目	説明
RS	ソース汎用レジスターを指定します。
RA	宛先汎用レジスターを指定します。

関連概念
cntlzw または cntlz (先行ゼロ・ワードのカウント) 命令

rac (実アドレス計算) 命令

目的

有効アドレスを実アドレスに変換し、その結果を汎用レジスターに保管します。

注: rac 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21 - 30	818
31	rc

POWER® ファミリー

ラック *RT*、 *RA*、 *RB*

POWER® ファミリー

ラック *RT*、 *RA*、 *RB*

説明

rac 命令は、汎用レジスター (GPR) *RA* の内容と GPR *RB* の内容の合計から有効アドレス (EA) を計算し、EA を仮想アドレスに拡張します。

RA が 0 でなく、 *RA* が *RT* でない場合、 **rac** 命令は EA を GPR *RA* に保管し、結果を実アドレスに変換し、実アドレスを GPR *RT* に保管します。

rac 命令を使用する場合は、以下のことを考慮してください。

- GPR *RA* が 0 の場合、EA は GPR *RB* の内容と 0 の合計です。
- データ変換が有効になっているかどうかに関係なく、EA は仮想アドレスに拡張され、実アドレスに変換されます。
- 変換が成功すると、条件レジスターの EQ ビットが設定され、実アドレスが GPR *RT* に入れられます。
- 変換が失敗すると、EQ ビットが 0 に設定され、0 が GPR *RT* に入れられます。
- 有効アドレスが入出力アドレスを指定する場合、EQ ビットは 0 に設定され、0 は GPR *RT* に入れられます。
- 実アドレスが Translation Look-Aside buffer (TLB) にない場合は、参照ビットが設定されます。

rac 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	**説明**			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
ラック	なし	なし	0	なし
ラック	なし	なし	1	EQ、SO

rac 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定すると、命令は条件レジスター・フィールド 0 の等価 (EQ) および要約オーバーフロー (SO) ビットに影響します。

> **注:** ハードウェアは、最初に変換ルック・アサイド・バッファーでアドレスを検索する場合があります。 これが失敗した場合は、ページ・フレーム・テーブルを検索する必要があります。 この場合、変換ルック・アサイド・バッファー項目をロードする必要はありません。

パラメーター

項目	**説明**
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

セキュリティー

rac 命令には特権があります。

関連概念

処理とストレージ
プロセッサーは、メイン・メモリーおよびレジスターにデータを保管します。

rfi (割り込みからの戻り) 命令

目的

マシン状態レジスターを再初期化し、割り込み後に処理を続行します。

構文

ビット	VALUE
0 - 5	19
6 - 10	///
11 - 15	///
16 - 20	///
21 - 30	50
31	/

RFi

説明

rfi 命令は、Save Restore Register1 (SRR1) のビット 16 から 31 を Machine State Register (MSR) のビット 16 から 31 に入れ、新しい MSR 値を使用して、Save Restore Register0 (SRR0) に含まれているアドレスで命令のフェッチと処理を開始します。

リンク・ビット (LK) が 1 に設定されている場合、リンク・レジスターの内容は未定義です。

rfi 命令には 1 つの構文形式があり、条件レジスター・フィールド 0 または固定小数点例外レジスターには影響しません。

セキュリティー

rfi 命令には特権があり、同期しています。

関連概念

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

rfid (割り込みダブルワードからの戻り) 命令

目的

マシン状態レジスターを再初期化し、割り込み後に処理を続行します。

構文

ビット	VALUE
0 - 5	19
6 - 10	00000
11 - 15	00000
16 - 20	00000
21 - 30	18
31	0

rfid (rfid)

説明

保管復元レジスター 1 (SRR1) のビット 0、48 から 55、57 から 59、および 62 から 63 は、マシン状態レジスター (MSR) の対応するビットに入れられます。 新しい MSR 値で保留中の例外が有効になっていない場合は、次の命令が新しい MSR 値の制御下でフェッチされます。 MSR 内の SF ビットが 1 の場合、 SRR0 (フルワードで位置合わせされたアドレス) のビット 0 から 61 にあるアドレスが、次の命令アドレスになります。 SF ビットがゼロの場合、 SRR0 の 32 ビットから 61 ビットまでをゼロに連結してワード位置合わせ adderss を作成すると、SRR0 の下位 32 ビットに配置されます。 上位 32 ビットはクリアされます。 新しい MSR 値が 1 つ以上の保留中の例外を使用可能にすると、最も優先順位の高い保留中の例外に関連付けられた例外が生成されます。この場合、例外処理メカニズムによって SRR0 に入れられる値は、例外が発生しなかった場合に次に実行される命令のアドレスです。

変更されたその他のレジスター:

- MSR

セキュリティー

rfid 命令は特権を持ち、同期しています。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット・インプリメンテーションで使用すると、正しくない命令タイプのプログラム例外が発生します。

rfsvc (SVC からの戻り) 命令

目的

マシン状態レジスターを再初期化し、監視プログラム呼び出し (**svc**) の後で処理を開始します。

注: rfsvc 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	19
6 - 10	///
11 - 15	///
16 - 20	///
21 - 30	82
31	LK

説明

rfsvc 命令は、マシン状態レジスター (MSR) を再初期化し、監視プログラム呼び出しの後に処理を開始します。 この命令は、カウント・レジスターのビット 16 から 31 をマシン状態レジスター (MSR) のビット 16 から 31 に入れ、新しい MSR 値を使用して、リンク・レジスターに含まれるアドレスから命令の取り出しと処理を開始します。

リンク・ビット (LK) が 1 に設定されている場合、リンク・レジスターの内容は未定義です。

rfsvc 命令には 1 つの構文形式があり、条件レジスター・フィールド 0 または固定小数点例外レジスターには影響しません。

セキュリティー

rfsvc 命令は特権を持ち、同期しています。

関連概念

ブランチ・プロセッサー

分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

システム・コール命令

PowerPC ® システム・コール命令は、サービスを実行するための割り込みまたはシステムを生成します。

rldcl (左ダブルワードから左に回転) 命令

目的

汎用レジスターの内容を、別の汎用レジスターの内容によって指定されたビット数だけ循環させます。 シフト操作の結果と AND 演算されるマスクを生成します。 この操作の結果を別の汎用レジスターに保管します。

構文

ビット	VALUE
0 - 5	30
6 - 10	S
11 - 15	A
16 - 20	B
21 - 26	MB
27 - 30	8
31	rc

POWER ® ファミリー

RLDCL *RA*、 *RS*、 *RB*、 *MB* (Rc=0)

rldcl: *RA*、 *RS*、 *RB*、 *MB* (Rc=1)

説明

汎用レジスター (GPR) *RS* の内容は、 *RB* の下位 6 ビットのオペランドで指定されたビット数だけ左に回転されます。 マスクは、ビット *MB* からビット 63 までの 1 ビットと、その他の場所に 0 ビットを持つように生成されます。 ローテーションされたデータは、生成されたマスクと AND 演算され、結果は *RA* に入れられます。

rldcl 命令は、以下に示す方法を使用してビット・フィールドを抽出および回転するために使用できることに注意してください。

- n ビット・フィールドを抽出するには、レジスター *RS* の可変ビット位置 b から開始し、右寄せで *RA* にします (*RA* の残りの 64 ビットをクリアします)。 *RB* の下位 6 ビットを b + n に設定し、 *MB* を 64-n に設定します。
- レジスターの内容を変数 n ビットだけ左に回転させるには、*RB* の下位 6 ビットを n と *MB* = 0 に設定し、レジスターの内容を右にシフトさせるには、 *RB* の下位 6 ビットを (64-n) に設定し、 *MB* = 0 に設定します。

変更されたその他のレジスター:

- 条件レジスター (CR0 フィールド):

  影響を受ける: LT、 GT、EQ、SO (Rc = 1 の場合)

パラメーター

項目	説明
RA	命令の結果のターゲット汎用レジスターを指定します。
RS	オペランドが入っているソース汎用レジスターを指定します。
rb	シフト値が入っているソース汎用レジスターを指定します。
MB	操作のマスクの開始値 (ビット番号) を指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

rldicl (ROTATE LEFT DOUBLE WORD IMMEDIATE THEN CLEAR LEFT) 命令

目的

この命令は、64 ビット・アプリケーションを実行する 64 ビット PowerPC プロセッサーでのみ使用してください。

構文

ビット	VALUE
0 - 5	30
6 - 10	S
11 - 15	A
16 - 20	sh
21 - 26	MB
27 - 29	0
30	sh
31	rc

PowerPC64

rldicl (rldicl) *rA*、 *rS*、 *rB*、 *MB* (Rc=0)

rldicl. *rA*、 *rS*、 *rB*、 *MB* (Rc=1)

説明

rS の内容は、オペランド SH によって指定されたビット数だけ左に回転されます。 マスクは、ビット MB からビット 63 までの 1 ビットと、それ以外の場所に 0 ビットを持つように生成されます。 ローテーションされたデータは、生成されたマスクと AND 演算され、結果は rA に入れられます。

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

以下に示す方法を使用して、rldicl を使用して、ビット・フィールドの抽出、回転、シフト、およびクリアを行うことができます。

n ビット・フィールドを抽出するには、rS 内のビット位置 b から開始し、右寄せして rA に入れます (rA の残りの 64 ビットをクリアします)。SH = b + n および MB = 64-n を設定します。

n ビット単位でレジスターの内容を回転させるには、SH = n および MB = 0 を設定します。n ビット単位でレジスターの内容を右に回転させるには、SH = (64-n) および MB = 0 を設定します。

レジスターの内容を n ビット分だけ右にシフトするには、SH = 64-n および MB = n を設定します。

レジスターの高位 n ビットをクリアするには、SH = 0 および MB = n を設定します。

変更されたその他のレジスター:

- 条件レジスター (CR0 フィールド):

  影響を受ける: LT、 GT、EQ、SO (Rc = 1 の場合)

パラメーター

項目	説明
RA	***DESCRIPTION* **
RS	***DESCRIPTION* **
rb	***DESCRIPTION* **
MB	***DESCRIPTION* **

関連概念

固定小数点プロセッサー

固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

ブランチ・プロセッサー

分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

rldcr (左ダブルワードから右に回転) 命令

目的

汎用レジスターの内容を、別の汎用レジスターの内容によって指定されたビット数だけ循環させます。 シフト操作の結果と AND 演算されるマスクを生成します。 この操作の結果を別の汎用レジスターに保管します。

構文

ビット	VALUE
0 - 5	30
6 - 10	S
11 - 15	A
16 - 20	B
21 - 26	ME
27 - 30	9
31	rc

POWER® ファミリー

RLDCR *RA*、 *RS*、 *RB*、 *ME* (Rc=0)

rldcr。 *RA*、 *RS*、 *RB*、 *ME* (Rc=1)

説明

汎用レジスター (GPR) *RS* の内容は、 *RB* の下位 6 ビットによって指定されたビット数だけ左に回転されます。 ビット 0 からビット *ME* までの 1 ビット、およびその他の 0 ビットを持つマスクが生成されます。 ローテーションされたデータは、生成されたマスクと AND 演算され、結果は *RA* に入れられます。

rldcr は、以下に示す方法を使用してビット・フィールドを抽出および回転するために使用できます。

- レジスター *RS* の可変ビット位置 b から始まる n ビット・フィールドを、*RA* に左寄せで抽出する (*RA* の残りの 64 ビットをクリアする) には、 *RB* の下位 6 ビットを b に設定し、 *ME* = n-1 に設定します。
- レジスターの内容を変数 n ビットだけ左に回転させるには、*RB* の下位 6 ビットを n および *ME* = 63 に設定し、レジスターの内容を右にシフトさせるには、 *RB* の下位 6 ビットを (64-n) に設定し、 *ME* = 63 に設定します。

変更されたその他のレジスター:

- 条件レジスター (CR0 フィールド):

  影響を受ける: LT、 GT、EQ、SO (Rc = 1 の場合)

パラメーター

RS SH 操作のシフト値を指定します。 MB 操作のマスクの開始値を指定します。 ME BM 32 ビット・マスクの値を指定します。

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	シフト値が入っているソース汎用レジスターを指定します。
ME	操作のマスクの終了値を指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

rldic (左ダブルワードの即時回転後にクリア) 命令

目的

汎用レジスターの内容は、指定されたビット数だけ左に回転し、次にビット・フィールドでマスクして、いくつかの下位ビットと上位ビットをクリアします。 結果は別の汎用レジスターに入れられます。

構文

ビット	VALUE
0 - 5	30
6 - 10	S
11 - 15	A
16 - 20	sh
21 - 26	MB
27 - 29	2
30	sh
31	rc

POWER® ファミリー

rldicl (rldicl) *RA*、 *RS*、 *SH*、 *MB* (Rc=0)

rldicl。 *RA*、 *RS*、 *SH*、 *MB* (Rc=1)

説明

汎用レジスター (GPR) *RS* の内容は、オペランド *SH* で指定されたビット数だけ左に回転されます。 マスクは、ビット *MB* からビット 63- *SH* までの 1 ビット、およびその他の 0 ビットで生成されます。 ローテーションされたデータは、生成されたマスクと AND 演算され、結果は GPR *RA* に入れられます。

rldic を使用すると、以下に示す方法でビット・フィールドをクリアおよびシフトできることに注意してください。

- レジスターの内容の上位 b ビットをクリアしてから、結果を n ビット分左にシフトするには、 *SH* = n および *MB* = b-n を設定します。
- レジスターの高位 n ビットをクリアするには、 *SH* = 0 および *MB* = n を設定します。

変更されたその他のレジスター:

- 条件レジスター (CR0 フィールド):

  影響を受ける: LT、 GT、EQ、SO (Rc = 1 の場合)

パラメーター

項目	説明
RA	命令の結果のターゲット汎用レジスターを指定します。
RS	オペランドが入っているソース汎用レジスターを指定します。
SH	操作の (即時) シフト値を指定します。
MB	操作のビット・マスクの開始値を指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

rldicl (ROTATE LEFT DOUBLE WORD IMMEDIATE THEN CLEAR LEFT) 命令

目的

汎用レジスターの内容を、指定されたビット数だけ左に回転させ、指定された数の高位ビットをクリアします。 結果は別の汎用レジスターに入れられます。

構文

ビット	VALUE
0 - 5	30
6 - 10	S
11 - 15	A
16 - 20	sh
21 - 26	MB
27 - 29	0

ビット	VALUE
30	sh
31	rc

POWER® ファミリー

rldicl (rldicl) *RA*、 *RS*、 *SH*、 *MB* (Rc=0)

rldicl. *RA*、 *RS*、 *SH*、 *MB* (Rc=1)

説明

汎用レジスター *RS* の内容は、オペランド *SH* で指定されたビット数だけ左に回転されます。 ビット *MB* からビット 63 までの 1 ビット、およびその他の 0 ビットを含むマスクが生成されます。 ローテーションされたデータは、生成されたマスクと AND 演算され、結果は GPR *RA* に入れられます。

rldicl は、以下に示す方法を使用して、ビット・フィールドを抽出、回転、シフト、およびクリアするために使用できます。

- *RS* のビット位置 b から始まり、GPR *RA* (GPR *RA* の残りの 64 ビットをクリア) に右寄せされた n ビット・フィールドを抽出するには、 *SH* = b + n および *MB* = 64-n を設定します。
- レジスターの内容を n ビット分左に回転させるには、*SH* = n および *MB* = 0 を設定します。レジスターの内容を n ビット分右に回転させるには、 *SH* = (64-n) および *MB* = 0 を設定します。
- レジスターの内容を n ビット分だけ右にシフトするには、 *SH* = 64-n および *MB* = n を設定します。
- レジスターの高位 n ビットをクリアするには、 *SH* = 0 および *MB* = n を設定します。

変更されたその他のレジスター:

- 条件レジスター (CR0 フィールド):

  影響を受ける: LT、 GT、EQ、SO (Rc = 1 の場合)

パラメーター

項目	説明
RA	命令の結果のターゲット汎用レジスターを指定します。
RS	オペランドが入っているソース汎用レジスターを指定します。
SH	操作の (即時) シフト値を指定します。
MB	操作のマスクの開始値 (ビット番号) を指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

rldicr (ROTATE LEFT DOUBLE WORD IMMEDIATE THEN CLEAR RIGHT) 命令

目的

汎用レジスターの内容を、即時値で指定されたビット数だけ循環させます。 指定された数の下位ビットをクリアします。 結果を別の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	30
6 - 10	S
11 - 15	A
16 - 20	sh
21 - 26	ME
27 - 29	1
30	sh
31	rc

POWER® ファミリー

rldicr (rldicr) *RA*、 *RS*、 *SH*、 *MB* (Rc=0)

rldicr。 *RA*、 *RS*、 *SH*、 *MB* (Rc=1)

説明

汎用レジスター (GPR) *RS* の内容は、オペランド *SH* で指定されたビット数だけ左に回転されます。 ビット 0 からビット *ME* までの 1 ビット、およびその他の 0 ビットを持つマスクが生成されます。 ローテーションされたデータは、生成されたマスクと AND 演算され、結果は GPR *RA* に入れられます。

rldicr は、以下に示す方法を使用して、ビット・フィールドを抽出、回転、シフト、およびクリアするために使用できます。

- GPR *RA* (GPR *RA* の残りの 64 ビットをクリア) に左寄せで、GPR *RS* のビット位置 b から始まる n ビット・フィールドを抽出するには、 *SH* = b および *ME* = n-1 を設定します。
- レジスターの内容を n ビットずつ左 (右) に回転させるには、 *SH* = n (64-n) および *ME* = 63 を設定します。
- *SH* = n および *ME* = 63-n を設定して、レジスターの内容を n ビットだけ左にシフトします。
- レジスターの下位 n ビットをクリアするには、 *SH* = 0 および *ME* = 63-n を設定します。

変更されたその他のレジスター:

- 条件レジスター (CR0 フィールド):

  影響を受ける: LT、 GT、EQ、SO (Rc = 1 の場合)

パラメーター

項目	説明
RA	命令の結果のターゲット汎用レジスターを指定します。
RS	オペランドが入っているソース汎用レジスターを指定します。
SH	操作の (即時) シフト値を指定します。
ME	操作のマスクの終了値 (ビット番号) を指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

rldimi (ROTATE LEFT DOUBLE WORD IMMEDIATE THEN MASK INSERT) 命令

目的

汎用レジスターの内容は、指定されたビット数だけ左に回転されます。 生成されたマスクは、指定されたビット・フィールドを別の汎用レジスターの対応するビット・フィールドに挿入するために使用されます。

構文

ビット	VALUE
0 - 5	30
6 - 10	S
11 - 15	A
16 - 20	sh
21 - 26	MB
27 - 29	3
30	sh
31	rc

POWER® ファミリー

rldimi (rldimi)	*RA*、 *RS*、 *SH*、 *MB* (Rc=0)
ルディミ	*RA*、 *RS*、 *SH*、 *MB* (Rc=1)

説明

汎用レジスター (GPR) *RS* の内容は、オペランド *SH* で指定されたビット数だけ左に回転されます。 マスクは、ビット *MB* からビット 63- *SH* までの 1 ビット、およびその他の 0 ビットで生成されます。 ローテーションされたデータは、生成されたマスクの制御下で *RA* に挿入されます。

rldimi は、 *SH* = 64-(b + n) および *MB* = b を設定することによって、 *RS* で右寄せされた n ビット・フィールドをビット位置 b から *RA* に挿入するために使用できます。

変更されたその他のレジスター:

- 条件レジスター (CR0 フィールド):

  影響を受ける: LT、 GT、EQ、SO (Rc = 1 の場合)

パラメーター

項目	説明
RA	命令の結果のターゲット汎用レジスターを指定します。
RS	オペランドが入っているソース汎用レジスターを指定します。
SH	操作の (即時) シフト値を指定します。
MB	操作のビット・マスクの開始値を指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

rlmi (左に回転、次にマスク挿入) 命令

目的

汎用レジスターの内容を、別の汎用レジスターで指定されたビット数だけ左に回転させ、その結果を、生成されたマスクの制御下にある第 3 汎用レジスターに保管します。

注: rlmi 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	22
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 -2 5	MB
26 - 30	ME
31	rc

POWER® ファミリー

rlmi	*RA*、*RS*、*RB*、*MB*、*ME*
ルミ	*RA*、*RS*、*RB*、*MB*、*ME*
rlmi	*RA*、*RS*、*RB*、*BM*
ルミ	*RA*、*RS*、*RB*、*BM*

説明

ルミ 命令は、ソース汎用レジスター (GPR)「*RS*」の内容を GPR 要求ブロック のビット 27 から 31 で指定されたビット数だけ左に回転させ、マスク開始 (*MB*) およびマスク終了 (私は) の値で定義された 32 ビット生成マスクの制御下で、回転されたデータを GPR *RA* とは に保管します。

rlmi 命令を使用する場合は、以下のことを考慮してください。

- マスク・ビットが 1 の場合、命令は回転されたデータの関連ビットを GPR *RA* に入れます。マスク・ビットが 0 の場合、GPR *RA* ビットは変更されません。
- *MB* 値が *ME* 値 + 1 より小さい場合、開始点と終了点の間 (開始点を含む) のマスク・ビットは 1 に設定されます。 その他のビットはすべてゼロに設定されます。
- *MB* 値が *ME* 値 + 1 と同じ場合、32 個すべてのマスク・ビットが 1 に設定されます。
- *MB* 値が *ME* 値 + 1 より大きい場合、 *ME* 値 + 1 と *MB* 値 -1 の間にあるすべてのマスク・ビットがゼロに設定されます。 その他のビットはすべて 1 に設定されます。

パラメーター *BM* を使用して、この命令のマスクを指定することもできます。 アセンブラーは、 *BM* から *MB* および *ME* パラメーターを生成します。

rlmi 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**

項目	説明			
rlmi	なし	なし	0	なし
ルミ	なし	なし	1	LT、GT、EQ、SO

rlmi 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	データの回転のためのビット数が入っている汎用レジスターを指定します。
MB	操作のマスクの開始値を指定します。
ME	操作のマスクの終了値を指定します。
BM	32 ビット・マスクの値を指定します。

例

1. 以下のコードは、GPR 4 の内容を GPR 5 のビット 27 から 31 に含まれる値だけ回転させ、マスクされた結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 5 contains 0x0000 0002.
# Assume GPR 6 contains 0xFFFF FFFF.
rlmi 6,4,5,0,0x1D
# GPR 6 now contains 0x4000 C003.
# Under the same conditions
# rlmi 6,4,5,0xFFFFFFFC
# will produce the same result.
```

2. 以下のコードは、GPR 5 のビット 27 から 31 に含まれる値によって GPR 4 の内容を回転させ、マスクされた結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 5 contains 0x0000 0002.
# GPR 6 is the target register and contains 0xFFFF FFFF.
rlmi. 6,4,5,0,0x1D
# GPR 6 now contains 0xC010 C003.
# CRF 0 now contains 0x8.
# Under the same conditions
# rlmi. 6,4,5,0xFFFFFFFC
# will produce the same result.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

rlwimi または rlimi (Rotate Left Word Immediate Then Mask Insert) 命令

目的

汎用レジスターの内容を指定されたビット数だけ左に回転させ、生成されたマスクの制御下にある別の汎用レジスターに結果を保管します。

構文

ビット	VALUE
0 - 5	20
6 - 10	RS
11 - 15	RA
16 - 20	SH
21 - 25	ME
26 - 30	MB
31	rc

PowerPC (R)

rlwimi (rlwimi)	*RA*、 *RS*、 *SH*、 *MB*、 *ME*
rlwimi。	*RA*、 *RS*、 *SH*、 *MB*、 *ME*
rlwimi (rlwimi)	*RA*、 *RS*、 *SH*、 *BM*
rlwimi。	*RA*、 *RS*、 *SH*、 *BM*

POWER® ファミリー

Rlimi (Rlimi)	*RA*、 *RS*、 *SH*、 *MB*、 *ME*
rlimi:	*RA*、 *RS*、 *SH*、 *MB*、 *ME*
Rlimi (Rlimi)	*RA*、 *RS*、 *SH*、 *BM*
rlimi:	*RA*、 *RS*、 *SH*、 *BM*

説明

ルウィミ 命令および **リミ** 命令は、ソース汎用レジスター (GPR) 「*RS*」 の内容を *SH* パラメーターによるビット数分だけ循環させ、マスク開始 (*MB*) およびマスク終了 (私は) の値によって定義された 32 ビット生成マスクの制御下で、循環したデータを GPR *RA* とは に保管します。 マスク・ビットが 1 の場合、命令は関連付けられた回転データのビットを GPR *RA* に入れます。マスク・ビットが 0 の場合、GPR *RA* ビットは変更されません。

rlwimi および **rlimi** の指示を使用する場合は、以下のことを考慮してください。

- *MB* 値が *ME* 値 + 1 より小さい場合、開始点と終了点の間 (開始点を含む) のマスク・ビットは 1 に設定されます。 その他のビットはすべてゼロに設定されます。
- *MB* 値が *ME* 値 + 1 と同じ場合、32 個すべてのマスク・ビットが 1 に設定されます。
- *MB* 値が *ME* 値 + 1 より大きい場合、 *ME* 値 + 1 と *MB* 値 -1 の間にあるすべてのマスク・ビットがゼロに設定されます。 その他のビットはすべて 1 に設定されます。

BM パラメーターを使用して、これらの命令のマスクを指定することもできます。 アセンブラーは、*BM* 値から *MB* および *ME* パラメーターを生成します。

rlwimi および **rlimi** 命令には、それぞれ 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
rlwimi (rlwimi)	なし	なし	0	なし
rlwimi。	なし	なし	1	LT、GT、EQ、SO
Rlimi (Rlimi)	なし	なし	0	なし
rlimi:	なし	なし	1	LT、GT、EQ、SO

rlwimi 命令および **rlimi** 命令の構文形式は、固定小数点例外レジスターに影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は、条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
SH	操作のシフト値を指定します。
MB	操作のマスクの開始値を指定します。
ME	操作のマスクの終了値を指定します。
BM	32 ビット・マスクの値を指定します。

例

1. 以下のコードは、GPR 4 の内容を 2 ビットずつ左に回転させ、マスクされた結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 6 contains 0x0000 0003.
rlwimi 6,4,2,0,0x1D
# GPR 6 now contains 0x4000 C003.
# Under the same conditions
#  rlwimi 6,4,2,0xFFFFFFFC
# will produce the same result.
```

2. 以下のコードは、GPR 4 の内容を 2 ビットずつ左に回転させ、マスクされた結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0x789A 789B.
# Assume GPR 6 contains 0x3000 0003.
rlwimi. 6,4,2,0,0x1A
# GPR 6 now contains 0xE269 E263.
# CRF 0 now contains 0x8.
# Under the same conditions
#  rlwimi. 6,4,2,0xFFFFFFE0
# will produce the same result.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

rlwinm または rlinm (左方ワードの即時回転、マスク付き AND) 命令

目的

論理的には、生成されたマスクを、汎用レジスターの内容の中の指定されたビット数だけ左に回転させることによって AND 演算します。

構文

ビット	VALUE
0 - 5	21
6 - 10	RS
11 - 15	RA
16 - 20	SH
21 - 25	MB
26 - 30	ME
31	rc

PowerPC (R)

rlwinm (rlwinm)	*RA*、*RS*、*SH*、*MB*、*ME*
rlwinm。	*RA*、*RS*、*SH*、*MB*、*ME*
rlwinm (rlwinm)	*RA*、*RS*、*SH*、*BM*
rlwinm。	*RA*、*RS*、*SH*、*BM*

POWER® ファミリー

リンク (rlinm)	*RA*、*RS*、*SH*、*MB*、*ME*
rlinm。	*RA*、*RS*、*SH*、*MB*、*ME*
リンク (rlinm)	*RA*、*RS*、*SH*、*BM*
rlinm。	*RA*、*RS*、*SH*、*BM*

詳しくは、 Fixed-Point Rotate and Shift Instructions の拡張ニーモニック を参照してください。

説明

RLWIM 命令と **リンク** 命令は、 「*RS*」 *SH* パラメーターで指定されたビット数だけソース汎用レジスター (GPR) の内容を循環させ、循環したデータをマスク開始 (*MB*) とマスク終了 (私は) の値で定義された 32 ビット生成マスクと論理 AND 演算し、結果を GPR に保管します。 *RA* とは

rlwinm および **rlinm** 命令を使用する場合は、以下のことを考慮してください。

- *MB* 値が *ME* 値 + 1 より小さい場合、開始点と終了点の間 (開始点を含む) のマスク・ビットは 1 に設定されます。 その他のビットはすべてゼロに設定されます。
- *MB* 値が *ME* 値 + 1 と同じ場合、32 個すべてのマスク・ビットが 1 に設定されます。
- *MB* 値が *ME* 値 + 1 より大きい場合、 *ME* 値 + 1 と *MB* 値 -1 の間にあるすべてのマスク・ビットがゼロに設定されます。 その他のビットはすべて 1 に設定されます。

BM パラメーターを使用して、これらの命令のマスクを指定することもできます。 アセンブラーは、*BM* 値から *MB* および *ME* パラメーターを生成します。

rlwinm 命令と **rlinm** 命令には、それぞれ 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
rlwinm (rlwinm)	なし	なし	0	なし
rlwinm。	なし	なし	1	LT、GT、EQ、SO
リンク (rlinm)	なし	なし	0	なし
rlinm。	なし	なし	1	LT、GT、EQ、SO

rlwinm 命令および **rlinm** 命令の構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は、条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
SH	操作のシフト値を指定します。
MB	操作のマスクの開始値を指定します。
ME	操作のマスクの終了値を指定します。
BM	32 ビット・マスクの値を指定します。

例

1. 以下のコードは、GPR 4 の内容を 2 ビット左に回転させ、その結果を 29 ビットのマスクで論理的に AND 演算します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 6 contains 0xFFFF FFFF.
rlwinm 6,4,2,0,0x1D
# GPR 6 now contains 0x4000 C000.
# Under the same conditions
# rlwinm 6,4,2,0xFFFFFFFC
# will produce the same result.
```

2. 以下のコードは、GPR 4 の内容を 2 ビットずつ左に回転させ、結果を 29 ビットのマスクで論理的に AND 演算し、演算の結果を反映するために条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 6 contains 0xFFFF FFFF.
rlwinm. 6,4,2,0,0x1D
# GPR 6 now contains 0xC010 C000.
# CRF 0 now contains 0x8.
# Under the same conditions
# rlwinm. 6,4,2,0xFFFFFFFC
# will produce the same result.
```

関連概念

固定小数点プロセッサー

固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

rlwnm または rlnm (左ワードを回転した後、マスクと AND) 命令

目的

汎用レジスターの内容を、別の汎用レジスターで指定されたビット数だけ左に回転させ、生成されたマスクを使用してローテートされたデータを論理的に AND 演算し、その結果を 3 番目の汎用レジスターに保管します。

構文

ビット	VALUE
0 - 5	23
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 25	MB
26 - 30	ME
31	rc

PowerPC (R)

rlwnm (rlwnm)	*RA*、*RS*、*RB*、*MB*、*ME*
rlwnm。	*RA*、*RS*、*RB*、*MB*、*ME*
rlwnm (rlwnm)	*RA*、*RS*、*SH*、*BM*
rlwnm。	*RA*、*RS*、*SH*、*BM*

POWER® ファミリー

rlnm (rlnm)	*RA*、*RS*、*RB*、*MB*、*ME*
rlnm。	*RA*、*RS*、*RB*、*MB*、*ME*
rlnm (rlnm)	*RA*、*RS*、*SH*、*BM*
rlnm。	*RA*、*RS*、*SH*、*BM*

詳しくは、 Fixed-Point Rotate and Shift Instructions の拡張ニーモニック を参照してください。

説明

RLWNM 命令および **RLM** 命令は、ソース汎用レジスター (GPR) 「*RS*」 の内容を、GPR 要求ブロックのビット 27 から 31 で指定されたビット数だけ左に回転させ、マスク開始 (*MB*) およびマスク終了 (私は) の値によって定義された 32 ビット生成マスクと論理的に AND 演算し、結果を GPR に保管します。 *RA* とは

rlwnm および **rlnm** 命令を使用する場合は、以下のことを考慮してください。

- *MB* 値が *ME* 値 + 1 より小さい場合、開始点と終了点の間 (開始点を含む) のマスク・ビットは 1 に設定されます。 その他のビットはすべてゼロに設定されます。
- *MB* 値が *ME* 値 + 1 と同じ場合、32 個すべてのマスク・ビットが 1 に設定されます。
- *MB* 値が *ME* 値 + 1 より大きい場合、 *ME* 値 + 1 と *MB* 値-1 の間にあるすべてのマスク・ビットがゼロに設定されます。 その他のビットはすべて 1 に設定されます。

BM パラメーターを使用して、これらの命令のマスクを指定することもできます。 アセンブラーは、*BM* 値から *MB* および *ME* パラメーターを生成します。

rlwnm および **rlnm** 命令には、それぞれ 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	**説明**			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
rlwnm (rlwnm)	なし	なし	0	なし
rlwnm。	なし	なし	1	LT、GT、EQ、SO
rlnm (rlnm)	なし	なし	0	なし
rlnm。	なし	なし	1	LT、GT、EQ、SO

rlwnm および **rlnm** 命令の構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は、条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	**説明**
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	データの回転のためのビット数が入っている汎用レジスターを指定します。
MB	操作のマスクの開始値を指定します。
ME	操作のマスクの終了値を指定します。
SH	操作のシフト値を指定します。
BM	32 ビット・マスクの値を指定します。

例

1. 以下のコードは、GPR 4 の内容を 2 ビット分左方に回転させ、29 のマスクで結果を論理的に AND 演算し、その結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 5 contains 0x0000 0002.
# Assume GPR 6 contains 0xFFFF FFFF.
rlwnm 6,4,5,0,0x1D
# GPR 6 now contains 0x4000 C000.
# Under the same conditions
#  rlwnm 6,4,5,0xFFFFFFFC
# will produce the same result.
```

2. 以下のコードは、GPR 4 を 2 ビット左に回転させ、結果を 29 ビットのマスクで論理的に AND 演算し、結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 5 contains 0x0000 0002.
# Assume GPR 6 contains 0xFFFF FFFF.
rlwnm. 6,4,5,0,0x1D
# GPR 6 now contains 0xC010 C000.
# CRF 0 now contains 0x8.
# Under the same conditions
#  rlwnm. 6,4,5,0xFFFFFFFC
# will produce the same result.
```

関連概念
固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

rrib (右に回転してビットを挿入) 命令

目的

汎用レジスター内のビット 0 を、別の汎用レジスターによって指定されたビット数だけ右に回転させ、回転されたビットを 3 番目の汎用レジスターに保管します。

注: rrib 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	537
31	rc

POWER® ファミリー

rrib (rrib) *RA*、*RS*、*RB*

rrib。 *RA*、*RS*、*RB*

説明

rrib 命令は、GPR *RB* のビット 27 から 31 で指定されたビット数だけ、ソース汎用レジスター (GPR) *RS* のビット 0 を右に回転させ、回転されたビットを GPR *RA* に保管します。

rrib 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	オーバーフロー 例外 (OE)	固定小数点 例外レジスター	レコード ビット (RC)	条件 フィールド 0 の登録
rrib (rrib)	なし	なし	0	なし
rrib。	なし	なし	1	LT、GT、EQ、SO

rrib 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	データの回転のためのビット数が入っている汎用レジスターを指定します。

例

1. 以下のコードは、GPR 5 のビット 0 を 4 ビットだけ右に回転させ、その値を GPR 4 に保管します。

```
# Assume GPR 5 contains 0x0000 0000.
# Assume GPR 6 contains 0x0000 0004.
# Assume GPR 4 contains 0xFFFF FFFF.
rrib 4,5,6
# GPR 4 now contains 0xF7FF FFFF.
```

2. 次のコードは、GPR 5 のビット 0 を 4 ビット右に回転させ、その値を GPR 4 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 5 contains 0xB004 3000.
# Assume GPR 6 contains 0x0000 0004.
# Assume GPR 4 contains 0x0000 0000.
rrib. 4,5,6
# GPR 4 now contains 0x0800 0000.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

sc (システム・コール) 命令

目的

サービスを提供するためにシステムを呼び出します。

注 : sc 命令は、PowerPC ® アーキテクチャーでのみサポートされます。

構文

ビット	値
0 - 5	17
6 - 10	///
11 - 15	///
16 - 19	///
20 - 26	LEV
27 - 29	///
30	1
31	/

PowerPC (R)

項目	説明
sc	LEV (LEV)

説明

sc 命令は、システム・コール割り込みを発生させます。 **sc** 命令に続く命令の有効アドレス (EA) は、保管復元レジスター 0 (SRR0) に入れられます。 マシン状態レジスター (MSR) のビット 0、5 から 9、および 16 から 31 は、保管復元レジスター 1 (SRR1) の対応するビットに入れられます。 SRR1 のビット 1 から 4 および 10 から 15 は、未定義の値に設定されます。

sc 命令は、基本ニーモニックと拡張ニーモニックの両方として機能します。 拡張形式では、*LEV* フィールドは省略され、0 と見なされます。

sc 命令には、1 つの構文形式があります。 構文形式は、マシン状態レジスターには影響しません。

注 : sc 命令には、 451 ページの『svc (監視プログラム呼び出し) 命令』と同じ命令コードがあります。

パラメーター

項目	説明
LEV	0 または 1 でなければなりません。

関連概念

svc (監視プログラム呼び出し) 命令

システム・コール命令

PowerPC ® システム・コール命令は、サービスを実行するための割り込みまたはシステムを生成します。

POWER ® ファミリーと PowerPC ® 命令の機能の違い

POWER ® ファミリーと PowerPC ® 命令は、POWER ® ファミリーと PowerPC ® プラットフォームで同じ命令コードを共有しますが、機能定義は異なります。

scv (システム・コール・ベクトル) 命令

目的

サービスを提供するためにシステムを呼び出します。

注 : scv 命令は、PowerPC ® アーキテクチャーでのみサポートされます。

構文

ビット	値
0 - 5	17
6 - 10	///
11 - 15	///
16 - 19	///
20 -26	LEV
27 - 29	///
30	0
31	1

PowerPC (R)

項目	説明
SCV	LEV (LEV)

説明

scv 命令は、システム・コール割り込みを発生させます。 **scv** 命令に続く命令の有効アドレス (EA) がリンク・レジスターに入れられます。 マシン状態レジスター (MSR) のビット 0 から 32、37 から 41、および 48 から 63 は、カウント・レジスターの対応するビットに入れられます。 カウント・レジスターのビット 33 から 36 および 42 から 47 は、未定義の値に設定されます。

scv 命令には、1 つの構文形式があります。 構文形式は、マシン状態レジスターには影響しません。

注 : scv 命令には、 362 ページの『scv (システム・コール・ベクトル) 命令』と同じ命令コードがあります。

パラメーター

項目	説明
LEV	0 または 1 でなければなりません。

関連概念

svc (監視プログラム呼び出し) 命令

システム・コール命令

PowerPC ® システム・コール命令は、サービスを実行するための割り込みまたはシステムを生成します。

POWER ® ファミリーと PowerPC ® 命令の機能の違い

POWER ® ファミリーと PowerPC ® 命令は、POWER ® ファミリーと PowerPC ® プラットフォームで同じ命令コードを共有しますが、機能定義は異なります。

si (即時減算) 命令

目的

符号付き整数の値を汎用レジスターの内容から減算して、その結果を汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	12
6 - 10	RT
11 - 15	RA
16 - 31	SI

項目	説明
Si	*RT*、 *RA*、 *SINT*

説明

si 命令は、 *SINT* パラメーターで指定された 16 ビット符号付き整数を汎用レジスター (GPR) *RA* の内容から減算し、その結果をターゲット GPR *RT* に保管します。 この命令には、負の *SINT* 値を指定して使用した **ai** 命令と同じ効果があります。 アセンブラーは *SINT* を否定し、この値 (*SI*) をマシン・インストラクションに入れます。

```
ai RT,RA,-SINT
```

si 命令には 1 つの構文形式があり、固定小数点例外レジスターのキャリー・ビットを設定できます。条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RT	操作のターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。
シント	演算のための 16 ビットの符号付き整数を指定します。
SI	*SINT* 値の負の値を指定します。

例

以下のコードは、GPR 4 の内容から 0xFFFF F800 を減算し、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターに Carry ビットを設定します。

```
# Assume GPR 4 contains 0x0000 0000
si 6,4,0xFFFFF800
# GPR 6 now contains 0x0000 0800
# This instruction has the same effect as
#     ai 6,4,-0xFFFFF800.
```

関連概念

addic または ai (即値携帯の追加) 命令

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

SI (即時およびレコードの減算) 命令

目的

汎用レジスターの内容から符号付き整数の値を減算し、その結果を 2 番目の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	13
6 - 10	RT
11 - 15	RA
16 - 31	SI

項目	説明
SI	*RT*、 *RA*、 *SINT*

説明

si. 命令は、 *SINT* パラメーターによって指定された 16 ビット符号付き整数を汎用レジスター (GPR) *RA* の内容から減算し、その結果をターゲット GPR *RT* に保管します。 この命令には、**ai** と同じ効果があります。命令が負の *SINT* で使用されています。 アセンブラーは *SINT* を否定し、この値 (*SI*) をマシン・インストラクションに入れます。

```
ai. RT,RA,-SINT
```

si. 命令には 1 つの構文形式があり、固定小数点例外レジスターのキャリー・ビットを設定できます。 この命令は、条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、または「要約オーバーフロー (SO)」ビットにも影響します。

パラメーター

項目	説明
RT	操作のターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。
シント	演算のための 16 ビットの符号付き整数を指定します。
SI	*SINT* 値の負の値を指定します。

例

以下のコードは、GPR 4 の内容から 0xFFFF F800 を減算し、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターおよび条件レジスター・フィールド 0 に Carry ビットを設定します。

```
# Assume GPR 4 contains 0xEFFF FFFF.
si. 6,4,0xFFFFF800
# GPR 6 now contains 0xF000 07FF.
# This instruction has the same effect as
#     ai. 6,4,-0xFFFFF800.
```

関連概念

addic または ai (即値携帯の追加) 命令

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

sld (左ダブルワードのシフト) 命令

目的

汎用レジスターの内容を、別の汎用レジスターの内容で指定されたビット数だけ左にシフトします。

構文

ビット	VALUE
0 - 5	31
6 - 10	S
11 - 15	A
16 -20	B
21 - 30	27
31	rc

POWER® ファミリー

SLD *RA*、 *RS*、 *RB* (OE=0 Rc=0)

sld。 *RA*、 *RS*、 *RB* (OE=0 Rc=1)

説明

汎用レジスター (GPR) *RS* の内容は、GPR *RB* の下位 7 ビットによって指定されたビット数だけ左にシフトされます。 位置 0 からシフトアウトされたビットは失われます。 右側の空いた位置にはゼロが指定されます。 結果は GPR *RA* に入れられます。 64 から 127 にシフトすると、結果はゼロになります。

変更されたその他のレジスター:

- 条件レジスター (CR0 フィールド):

  影響を受ける: LT、 GT、EQ、SO (Rc = 1 の場合)

パラメーター

項目	説明
RA	操作の結果のターゲット汎用レジスターを指定します。
RS	これは、thr シフト操作のオペランドが入っているソース汎用レジスターを指定します。
rb	下位 7 ビットは、オペランドをシフトする距離を指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

sle (左シフト拡張) 命令

目的

汎用レジスターの内容を、指定されたビット数だけ左にシフトし、ローテーションされたデータのコピーを MQ レジスターに入れ、その結果を別の汎用レジスターに入れます。

注: sle 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	153
31	rc

POWER® ファミリー

sle	*RA*、 *RS*、 *RB*
Sle。	*RA*、 *RS*、 *RB*

説明

sle 命令は、ソース汎用レジスター (GPR) *RS* の内容を *N* ビットだけ左に回転させます。ここで、 *N* は GPR *RB* のビット 27 から 31 に指定されたシフト量です。 また、この命令は、回転されたワードを MQ レジスターに保管し、回転されたワードと生成されたマスクの論理 AND を GPR *RA* に保管します。 マスクは、32 マイナス *N* 1 とそれに続く *N* 個のゼロで構成されます。

sle 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
sle	なし	なし	0	なし
Sle。	なし	なし	1	LT、GT、EQ、SO

sle 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容を 4 ビットずつ左に回転させ、回転されたデータのコピーを MQ レジスターに入れ、回転されたデータとマスクを AND 演算した結果を GPR 6 に入れます。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 5 contains 0x0000 0004.
sle 6,4,5
# GPR 6 now contains 0x0003 0000.
# The MQ Register now contains 0x0003 0009.
```

2. 以下のコードは、GPR 4 の内容を 4 ビット分左に回転させ、回転されたデータのコピーを MQ レジスターに入れ、回転されたデータをマスクと AND 演算した結果を GPR 6 に入れ、演算の結果を反映するために条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 5 contains 0x0000 0004.
sle. 6,4,5
# GPR 6 now contains 0x0043 0000.
# The MQ Register now contains 0x0043 000B.
# Condition Register Field 0 now contains 0x4.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

sleq (MQ で拡張されたシフト) 命令

目的

汎用レジスターの内容を指定されたビット数だけ左に回転させ、その結果をマスクの制御下にある MQ レジスターの内容とマージし、回転後のワードを MQ レジスターに入れ、マスクされた結果を別の汎用レジスターに入れます。

注: sleq 命令は POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	217
31	rc

POWER® ファミリー

スレイク *RA*、*RS*、*RB*

スレイク *RA*、*RS*、*RB*

説明

sleq 命令は、ソース汎用レジスター (GPR) *RS* left *N* ビットの内容を回転させます。ここで、 *N* は GPR *RB* のビット 27 から 31 に指定されているシフト量です。 この命令は、ローテートされたワードをマスクの制御下で MQ レジスターの内容とマージし、ローテートされたワードを MQ Register およびマージされたワードを GPR *RA* に保管します。 マスクは、32 マイナス *N* 1 とそれに続く *N* 個のゼロで構成されます。

sleq 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
スレイク	なし	なし	0	なし
スレイク	なし	なし	1	LT、GT、EQ、SO

sleq 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容を 4 ビット左に回転させ、ローテーションされたデータと MQ レジスターの内容を生成されたマスクの下でマージし、ローテーションされたワードを MQ レジスターに入れ、結果を GPR 6 に入れます。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 5 contains 0x0000 0004.
```

```
# Assume the MQ Register contains 0xFFFF FFFF.
sleq 6,4,5
# GPR 6 now contains 0x0003 000F.
# The MQ Register now contains 0x0003 0009.
```

2. 以下のコードは、GPR 4 の内容を 4 ビット左に回転させ、ローテーションされたデータと MQ レジスターの内容を生成されたマスクの下にマージし、ローテーションされたワードを MQ レジスターに入れ、結果を GPR 6 に入れ、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 5 contains 0x0000 0004.
# Assume the MQ Register contains 0xFFFF FFFF.
sleq. 6,4,5
# GPR 6 now contains 0x0043 000F.
# The MQ Register now contains 0x0043 000B.
# Condition Register Field 0 now contains 0x4.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

sliq (MQ による即時シフト) 命令

目的

汎用レジスターの内容を、即時値の中の指定されたビット数だけ左にシフトし、回転後の内容を MQ レジスターに入れ、その結果を別の汎用レジスターに入れます。

注: sliq 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	SH
21 - 30	184
31	rc

POWER® ファミリー

スリーク *RA*、 *RS*、 *SH*

スリーク *RA*、 *RS*、 *SH*

説明

sliq 命令は、ソース汎用レジスター (GPR) *RS* の内容を *N* ビットずつ左に回転させます。ここで、 *N* は *SH* で指定されたシフト量です。 この命令は、回転されたワードを MQ レジスターおよび回転されたワードの論理 AND に保管し、生成されたマスクを GPR *RA* に入れます。 マスクは、32 マイナス *N* 1 とそれに続く *N* 個のゼロで構成されます。

sliq 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
スリーク	なし	なし	0	なし
スリーク	なし	なし	1	LT、GT、EQ、SO

sliq 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
SH	シフト量の即時値を指定します。

例

1. 以下のコードは、GPR 4 の内容を左側に 20 ビット回転させ、生成されたマスクで回転されたデータを AND 演算し、回転されたワードを MQ レジスターに入れ、結果を GPR 6 に入れます。

```
# Assume GPR 4 contains 0x1234 5678.
sliq 6,4,0x14
# GPR 6 now contains 0x6780 0000.
# MQ Register now contains 0x6781 2345.
```

2. 以下のコードは、GPR 4 の内容を 16 ビット左に回転させ、生成されたマスクを使用して回転されたデータを AND 演算し、回転されたワードを MQ レジスターに入れ、GPR 6 に結果を入れ、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0x1234 5678.
sliq. 6,4,0x10
# GPR 6 now contains 0x5678 0000.
# The MQ Register now contains 0x5678 1234.
# Condition Register Field 0 now contains 0x4.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

slliq (MQ を使用した Shift Left Long Immediate) 命令

目的

汎用レジスターの内容を、即時値の指定されたビット数だけ左に回転させ、その結果をマスクの制御下にある MQ レジスターの内容とマージし、ローテートされたワードを MQ レジスターに入れ、マスクされた結果を別の汎用レジスターに入れます。

注: slliq 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	SH
21 - 30	248
31	rc

POWER® ファミリー

slliq (slliq) *RA*、 *RS*、 *SH*

slliq: *RA*、 *RS*、 *SH*

説明

Slliq 命令は、ソース汎用レジスター (GPR) 「*RS*」 の内容を *N* ビット単位で左に回転させます。ここで、*N* は *SH* で指定されたシフト量であり、結果を MQ レジスターの内容とマージし、ローテートされたワードを MQ レジスターに保管し、最終結果を GPR *RA* とはに保管します。 マスクは、32 マイナス *N* 1 とそれに続く *N* 個のゼロで構成されます。

slliq 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
slliq (slliq)	なし	なし	0	なし
slliq:	なし	なし	1	LT、GT、EQ、SO

slliq 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
SH	シフト量の即時値を指定します。

例

1. 以下のコードは、GPR 4 の内容を左に 3 ビット回転させ、ローテーションされたデータを MQ レジスターの内容と生成されたマスクの下でマージし、ローテーションされたワードを MQ レジスターに入れ、その結果を GPR 6 に入れます。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume the MQ Register contains 0xFFFF FFFF.
slliq 6,4,0x3
# GPR 6 now contains 0x8001 8007.
# The MQ Register now contains 0x8001 8004.
```

2. 以下のコードは、GPR 4 の内容を 4 ビット左に回転させ、ローテーションされたデータと MQ レジスターの内容を生成されたマスクの下にマージし、ローテーションされたワードを MQ レジスターに入れ、結果を GPR 6 に入れ、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume the MQ Register contains 0xFFFF FFFF.
slliq. 6,4,0x4
# GPR 6 now contains 0x0043 000F.
# The MQ Register contains 0x0043 000B.
# Condition Register Field 0 now contains 0x4.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

sllq (MQ を使用した左シフト) 命令

目的

汎用レジスターの内容を、汎用レジスターで指定されたビット数だけ左に回転させ、循環したデータまたはゼロのワードを MQ レジスターの内容とマージし、その結果を 3 番目の汎用レジスターに入れます。

注: sliq 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	216
31	rc

POWER® ファミリー

sllq *RA*、 *RS*、 *RB*

SLLQ *RA*、 *RS*、 *RB*

説明

sllq 命令は、ソース汎用レジスター (GPR) *RS* の内容を *N* ビットだけ左に回転させます。ここで、*N* は GPR *RB* のビット 27 から 31 に指定されたシフト量です。 マージは、GPR *RB* のビット 26 の値によって異なります。

sllq 命令を使用する場合は、以下の点を考慮してください。

- GPR *RB* のビット 26 が 0 の場合、 *N* 個のゼロとそれに続く 32 個のマイナス *N* 個のマスクが生成されます。 ローテートされたワードは、この生成されたマスクの制御下で MQ レジスターの内容とマージされます。
- GPR *RB* のビット 26 が 1 の場合、 *N* 個の 1 とそれに続く 32 から *N* 個のゼロのマスクが生成されます。 次に、この生成されたマスクの制御下で、ゼロのワードが MQ レジスターの内容とマージされます。

マージされたワードは、GPR *RA* に保管されます。 MQ レジスターは変更されません。

sllq 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
sllq	なし	なし	0	なし
SLLQ	なし	なし	1	LT、GT、EQ、SO

sllq 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容を 4 ビット左に回転させ、ゼロのワードと MQ レジスターの内容をマスクの下にマージし、マージされた結果を GPR 6 に入れます。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 5 contains 0x0000 0024.
# Assume MQ Register contains 0xABCD EFAB.
sllq 6,4,5
# GPR 6 now contains 0xABCD EFA0.
# The MQ Register remains unchanged.
```

2. 以下のコードは、GPR 4 の内容を 4 ビット左に回転させ、回転されたデータをマスクの下の MQ レジスターの内容とマージし、マージされた結果を GPR 6 に入れ、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 5 contains 0x0000 0004.
# Assume MQ Register contains 0xFFFF FFFF.
sllq. 6,4,5
# GPR 6 now contains 0x0043 000F.
# The MQ Register remains unchanged.
# Condition Register Field 0 now contains 0x4.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

slq (MQ の左シフト) 命令

目的

汎用レジスターの内容を、汎用レジスターに指定されたビット数だけ左に回転させ、回転されたワードを MQ レジスターに配置し、回転されたワードと生成されたマスクの論理 AND を 3 番目の汎用レジスターに配置します。

注: slq 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	152
31	rc

POWER® ファミリー

slq (slq) *RA*、*RS*、*RB*

slq。 *RA*、*RS*、*RB*

説明

slq 命令は、ソース汎用レジスター (GPR) *RS* の内容を *N* ビットだけ左に回転させます。ここで、*N* は GPR *RB* のビット 27 から 31 で指定されたシフト量で、回転されたワードを MQ レジスターに保管します。 マスクは、GPR *RB* のビット 26 に依存します。

slq 命令を使用する際には、以下のことを考慮してください。

- GPR *RB* のビット 26 が 0 の場合、32 から *N* を引いたマスクと、それに続く *N* 個のゼロが生成されます。
- GPR *RB* のビット 26 が 1 の場合は、すべてゼロのマスクが生成されます。

この命令は、回転されたワードと生成されたマスクの論理 AND を GPR *RA* に保管します。

slq 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
slq (slq)	なし	なし	0	なし
slq。	なし	なし	1	LT、GT、EQ、SO

slq 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容を 4 ビット左に回転させ、回転されたワードを MQ レジスターに入れ、回転されたワードと生成されたマスクの論理 AND を GPR 6 に入れます。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 5 contains 0x0000 0024.
slq 6,4,5
# GPR 6 now contains 0x0000 0000.
# The MQ Register now contains 0x0003 0009.
```

2. 以下のコードは、GPR 4 の内容を 4 ビット左に回転させ、回転されたワードを MQ レジスターに入れ、回転されたワードと生成されたマスクの論理 AND を GPR 6 に入れ、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 5 contains 0x0000 0004.
slq. 6,4,5
# GPR 6 now contains 0x0043 0000.
# The MQ Register now contains 0x0043 000B.
# Condition Register Field 0 now contains 0x4.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

slw または sl (Shift Left Word) 命令

目的

汎用レジスターの内容を、指定されたビット数だけ左に回転させて、マスクされた結果を別の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	24
31	rc

PowerPC (R)

スラウ (slw) *RA*、 *RS*、 *RB*

slw: *RA*、 *RS*、 *RB*

POWER® ファミリー

sl *RA*、 *RS*、 *RB*

SSL *RA*、 *RS*、 *RB*

説明

slw および **sl** 命令は、ソース汎用レジスター (GPR) *RS* の内容を左側の *N* ビットにローテーションします。ここで、 *N* は GPR *RB* のビット 27 から 31 で指定されたシフト量であり、ローテーションされたワードと生成されたマスクの論理 AND を GPR *RA* に保管します。

slw および **sl** 命令を使用する場合は、以下の点を考慮してください。

- GPR *RB* のビット 26 が 0 の場合、32-*N* 個の 1 とそれに続く *N* 個のゼロのマスクが生成されます。
- GPR *RB* のビット 26 が 1 の場合は、すべてゼロのマスクが生成されます。

slw および **sl** 命令には、それぞれ 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	**説明**			
構文 Form	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード・ビット (RC)**	**条件 フィールド 0 の登録**
スラウ (slw)	なし	なし	0	なし
slw:	なし	なし	1	LT、GT、EQ、SO
sl	なし	なし	0	なし
SSL	なし	なし	1	LT、GT、EQ、SO

slw 命令の 2 つの構文形式、および **sl** 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、これらの命令は、条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	**説明**
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容を左に 15 ビット回転させ、回転されたデータと生成されたマスクを AND 演算した結果を GPR 6 に保管します。

```
# Assume GPR 5 contains 0x0000 002F.
# Assume GPR 4 contains 0xFFFF FFFF.
slw 6,4,5
# GPR 6 now contains 0x0000 0000.
```

2. 以下のコードは、GPR 4 の内容を左に 5 ビットずつ回転させ、回転されたデータと生成されたマスクとの AND 演算の結果を GPR 6 に保管し、演算の結果を反映するために条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 5 contains 0x0000 0005.
slw. 6,4,5
# GPR 6 now contains 0x0086 0000.
# Condition Register Field 0 now contains 0x4.
```

関連概念

固定小数点プロセッサー

固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

srad (Shift Right Algebraic Double Word) 命令

目的

別の汎用レジスターの内容によって指定されたビット数だけ、汎用レジスターの内容を右に代数的にシフトします。 操作の結果を別の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	S
11 - 15	A
16 - 20	B
21 - 30	794
31	rc

POWER® ファミリー

SRAD *RA*、 *RS*、 *RB* (Rc=0)

srad。 *RA*、 *RS*、 *RB* (Rc=1)

説明

汎用レジスター (GPR) *RS* の内容は、GPR *RB* の下位 7 ビットによって指定されたビット数だけ右にシフトされます。 位置 63 からシフトアウトされたビットは失われます。 GPR *RS* のビット 0 は、左側の空き位置を埋めるために複製されます。 結果は GRP *RA* に入れられます。 GPR *RS* が負の場合は XER [CA] が設定され、1 ビットは 63 桁目からシフトされます。それ以外の場合は XER [CA] がクリアされます。 シフト量がゼロの場合、GRP *RA* は GPR *RS* と等しく設定され、XER [CA] はクリアされます。 64 から 127 へのシフトは、GRP *RA* で 64 個の符号ビットの結果を返し、XER [CA] が GPR *RS* の符号ビットを受け取るようにします。

srad 命令の後に addze を指定すると、 2**n で素早く除算できることに注意してください。 **srad** による CA ビットの設定は、モードに依存しません。

変更されたその他のレジスター:

- 条件レジスター (CR0 フィールド):

  影響を受ける: LT、 GT、EQ、SO (Rc = 1 の場合)
- XER:

  影響を受ける: CA

パラメーター

項目	説明
RA	操作の結果のターゲット汎用レジスターを指定します。
RS	これは、thr シフト操作のオペランドが入っているソース汎用レジスターを指定します。
rb	オペランドをシフトする距離を指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

sradi (Shift Right Algebra Double Word Immediate) 命令

目的

汎用レジスターの内容を、即値で指定されたビット数だけ代数的に右にシフトします。 操作の結果を別の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	S
11 - 15	A
16 - 20	sh
21 - 29	413
30	sh
31	rc

POWER® ファミリー

スラディ (sradi) *RA*、 *RS*、 *SH* (Rc=0)

sradi: *RA*、 *RS*、 *SH* (Rc=1)

説明

汎用レジスター (GPR) *RS* の内容は、右に *SH* ビットシフトされます。 位置 63 からシフトアウトされたビットは失われます。 GPR *RS* のビット 0 は、左側の空き位置を埋めるために複製されます。 結果は GPR *RA* に入れられます。 GPR *RS* が負の場合は XER [CA] が設定され、1 ビットは 63 桁目からシフトされます。それ以外の場合は XER [CA] がクリアされます。 シフト量がゼロの場合、GPR *RA* は GPR *RS* と等しく設定され、XER [CA] はクリアされます。

sradi 命令の後に addze を指定すると、 2**n ですぐに除算できることに注意してください。 **sradi** による CA ビットの設定は、モードに依存しません。

変更されたその他のレジスター:

- 条件レジスター (CR0 フィールド):

  影響を受ける: LT、 GT、EQ、SO (Rc = 1 の場合)
- XER:

  影響を受ける: CA

パラメーター

項目	説明
RA	操作の結果のターゲット汎用レジスターを指定します。
RS	シフト操作のオペランドが入っているソース汎用レジスターを指定します。

項目	説明
SH	操作のシフト値を指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

sraiq (MQ を使用する Shift Right Algebra Immediate) 命令

目的

汎用レジスターの内容を、指定されたビット数だけ左に回転させ、生成されたマスクの制御下でその汎用レジスターからの 32 個の符号ビットのワードで回転されたデータをマージし、回転されたワードを MQ レジスターに入れ、マージされた結果を別の汎用レジスターに入れます。

注: sraiq 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	SH
21 - 30	952
31	rc

POWER® ファミリー

ソース・キュー	*RA*、 *RS*、 *SH*
sraiq。	*RA*、 *RS*、 *SH*

説明

SRAQ 命令は、ソース汎用レジスター (GPR) の内容を 「*RS*」 左側に 32 マイナス *N* ビット回転させます。 ここで、 *N* は *SH* によって指定されたシフト量であり、生成されたマスクの制御下で回転されたデータを GPR 「*RS*」 からの 32 個の符号ビットのワードとマージし、その回転されたワードを MQ Register とマージされた結果を GPR *RA* とはに保管します。 32 個の符号ビットのワードは、GPR の符号ビットを取り、フルワードを作成するために 32 回繰り返すことによって生成されます。 このワードは、GPR の値に応じて、0x0000 0000 または 0xFFFF FFFF のいずれかになります。 マスクは、*N* 個のゼロと、それに続く 32 個のマイナス *N* 個の 1 で構成されます。

次に、この命令は、循環したデータを生成されたマスクの補数で AND 処理し、32 ビットの結果をまとめて OR 処理し、GPR *RS* のビット 0 でビット結果を AND 演算して、Carry ビット (CA) を生成します。

sraiq 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**

項目	説明			
ソース・キュー	なし	CA	0	なし
sraiq。	なし	CA	1	LT、GT、EQ、SO

sraiq 命令の 2 つの構文形式は、常に固定小数点例外レジスターのカリー・ビット (CA) に影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
SH	シフト量の即時値を指定します。

例

1. 以下のコードは、GPR 4 の内容を左に 28 ビット回転させ、生成されたマスクの制御下で 32 個の符号ビットと結果をマージし、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターに Carry ビットを設定します。

```
# Assume GPR 4 contains 0x9000 3000.
sraiq 6,4,0x4
# GPR 6 now contains 0xF900 0300.
# MQ now contains 0x0900 0300.
```

2. 以下のコードは、GPR 4 の内容を左に 28 ビット回転させ、生成されたマスクの制御下で 32 個の符号ビットと結果をマージし、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターおよび条件レジスター・フィールド 0 にカリー・ビットを設定します。

```
# Assume GPR 4 contains 0xB004 3000.
sraiq. 6,4,0x4
# GPR 6 now contains 0xFB00 4300.
# MQ now contains 0x0B00 4300.
# Condition Register Field 0 now contains 0x8.
```

関連概念

addze または aze (Add to Zero Extended) 命令

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

sraq (MQ での Shift Right Algebraic) 命令

目的

指定されたビット数の汎用レジスターを左に回転させ、生成されたマスクの制御下にあるその汎用レジスターからの 32 個の符号ビットのワードと結果をマージし、回転されたワードを MQ レジスターに入れ、マージされた結果を別の汎用レジスターに入れます。

注: sraq 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	920
31	rc

POWER® ファミリー

スラック *RA*、 *RS*、 *RB*

sraq: *RA*、 *RS*、 *RB*

説明

sraq 命令は、ソース汎用レジスター (GPR) *RS* の内容を左に 32 から *N* ビットを引いた値に回転させます。 ここで、 *N* は GPR *RB* のビット 27 から 31 で指定されたシフト量です。 その後、命令は、生成されたマスクの制御下で GPR *RS* からの 32 個の符号ビットのワードと循環データをマージし、マージされたワードを GPR *RA* に保管します。 回転されたワードは、MQ レジスターに保管されます。 マスクは、GPR *RB* のビット 26 の値によって異なります。

sraq 命令を使用する場合は、以下の点を考慮してください。

- GPR *RB* のビット 26 が 0 の場合、 *N* 個のゼロとそれに続く 32 個のマイナス *N* 個のマスクが生成されます。
- GPR *RB* のビット 26 が 1 の場合は、すべてゼロのマスクが生成されます。

32 個の符号ビットのワードは、GPR の符号ビットを取り、32 回繰り返してフルワードを作成することによって生成されます。 このワードは、GPR の値に応じて、 0x0000 0000 または 0xFFFF FFFF のいずれかになります。

次に、この命令は、循環したデータを生成されたマスクの補数で AND 処理し、32 ビットの結果をまとめて OR 処理し、GPR *RS* のビット 0 でビット結果を AND 演算して、Carry ビット (CA) を生成します。

sraq 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
スラック	なし	CA	0	なし
sraq:	なし	CA	1	LT、GT、EQ、SO

sraq 命令の 2 つの構文形式は、常に固定小数点例外レジスターのカリー・ビット (CA) に影響を与えます。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。

項目	説明
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容を 28 ビット左に回転させ、生成されたマスクの制御下にある 32 個の符号ビットと結果をマージし、結果を GPR 6 に入れ、回転されたワードを MQ レジスターに入れ、演算の結果を反映するために Carry ビットを固定小数点例外レジスターに設定します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 7 contains 0x0000 0024.
sraq 6,4,7
# GPR 6 now contains 0xFFFF FFFF.
# The MQ Register now contains 0x0900 0300.
```

2. 以下のコードは、GPR 4 の内容を 28 ビット左に回転させ、生成されたマスクの制御下で 32 個の符号ビットと結果をマージし、結果を GPR 6 に入れ、回転されたワードを MQ レジスターに入れ、演算の結果を反映するために固定小数点例外レジスターおよび条件レジスター・フィールド 0 に Carry ビットを設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 7 contains 0x0000 0004.
sraq. 6,4,7
# GPR 6 now contains 0xFB00 4300.
# The MQ Register now contains 0x0B00 4300.
# Condition Register Field 0 now contains 0x4.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

sraw または sra (Shift Right Algebraic Word) 命令

目的

汎用レジスターの内容を指定されたビット数だけ左に回転させ、そのレジスターからの 32 個の符号ビットのワードを持つ回転されたデータを、生成されたマスクの制御下でマージし、その結果を別の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	792
31	rc

PowerPC (R)

生 (sraw) *RA*、 *RS*、 *RB*

PowerPC (R)

スロー *RA*、 *RS*、 *RB*

POWER ® ファミリー

スラ *RA*、 *RS*、 *RB*

sra。 *RA*、 *RS*、 *RB*

説明

スロー 命令と **スラ** 命令は、ソース汎用レジスター (GPR) 「*RS*」 の内容を左に 32 から *N* ビットを引いた値に循環します。ここで、 *N* は、GPR 要求ブロックのビット 27 から 31 に指定されたシフト量であり、生成されたマスクの制御下で GPR 「*RS*」 からの 32 個の符号ビットのワードと循環ワードをマージします。 32 個の符号ビットのワードは、GPR の符号ビットを取り、32 回繰り返してフルワードを作成することによって生成されます。 このワードは、GPR の値に応じて、 0x0000 0000 または 0xFFFF FFFF のいずれかになります。

マスクは、GPR *RB* のビット 26 の値によって異なります。

sraw および **sra** 命令を使用する場合は、以下のことを考慮してください。

- GPR *RB* のビット 26 がゼロの場合、 *N* 個のゼロとそれに続く 32 から *N* 個の 1 を引いたマスクが生成されます。
- GPR *RB* のビット 26 が 1 の場合は、すべてゼロのマスクが生成されます。

マージされたワードは GPR *RA* に置かれます。 **sraw** 命令と **sra** 命令は、生成されたマスクの補数とローテーションされたデータを AND 演算し、32 ビットの結果を OR 演算し、ビットの結果を GPR *RS* のビット 0 と AND 演算して、Carry ビット (CA) を生成します。

sraw 命令と **sra** 命令には、それぞれ 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	**説明**			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
生 (sraw)	なし	CA	0	なし
スロー	なし	CA	1	LT、GT、EQ、SO
スラ	なし	CA	0	なし
sra。	なし	CA	1	LT、GT、EQ、SO

sraw 命令の 2 つの構文形式、および **sra** 命令の 2 つの構文形式は、常に固定小数点例外レジスターの Carry ビット (CA) に影響を与えます。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は、条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	**説明**
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容を左に 28 ビット回転させ、生成されたマスクの制御下で 32 個の符号ビットと結果をマージし、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターに Carry ビットを設定します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 5 contains 0x0000 0024.
sraw 6,4,5
# GPR 6 now contains 0xFFFF FFFF.
```

2. 以下のコードは、GPR 4 の内容を左に 28 ビット回転させ、生成されたマスクの制御下で 32 個の符号ビットと結果をマージし、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターおよび条件レジスター・フィールド 0 にカリー・ビットを設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 5 contains 0x0000 0004.
sraw. 6,4,5
# GPR 6 now contains 0xFB00 4300.
# Condition Register Field 0 now contains 0x8.
```

関連概念

addze または aze (Add to Zero Extended) 命令

固定小数点プロセッサー

固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示

固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

srawi または sraa (Shift Right Algebra Word Immediate) 命令

目的

指定されたビット数の汎用レジスターの内容を左に回転させ、生成されたマスクの制御下でそのレジスターからの 32 個の符号ビットのワードと循環させたデータをマージし、その結果を別の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	SH
21 - 30	824
31	rc

PowerPC (R)

スラウィ *RA*、 *RS*、 *SH*

srawi。 *RA*、 *RS*、 *SH*

POWER® ファミリー

未来 *RA*、 *RS*、 *SH*

スライ *RA*、 *RS*、 *SH*

説明

スラウィ 命令および **スライ** 命令は、ソース汎用レジスター (GPR) 「*RS*」 の内容を左に 32- *N* ビットだけ回転させます。ここで、 *N* は *SH* によって指定されたシフト量であり、生成されたマスクの制御下で回転されたデータを GPR 「*RS*」 からの 32 個の符号ビットのワードとマージし、マージされた結果を GPR *RA* とはに保管します。 32 個の符号ビットのワードは、GPR の符号ビットを取り、32 回繰り返してフルワードを作成することによって生成されます。 このワードは、GPR の値に応じて、 0x0000 0000 または 0xFFFF FFFF のいずれかになります。 マスクは、 *N* 個のゼロと、それに続く 32 個のマイナス *N* 個の 1 で構成されます。

次に、 **srawi** 命令と **sids** 命令が、生成されたマスクの補数を使用して回転されたデータを AND 演算し、32 ビットの結果を一緒に OR 演算し、GPR *RS* のビット 0 を使用してビット結果を AND 演算して、Carry ビット (CA) を生成します。

srawi 命令と **sprocessor** 命令には、それぞれ 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	**説明**			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
スラウィ	なし	CA	0	なし
srawi。	なし	CA	1	LT、GT、EQ、SO
未来	なし	CA	0	なし
スライ	なし	CA	1	LT、GT、EQ、SO

srawi 命令の 2 つの構文形式、および **s 減免** 命令の 2 つの構文形式は、常に固定小数点例外レジスターの Carry ビット (CA) に影響を与えます。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は、条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	**説明**
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
SH	シフト量の即時値を指定します。

例

1. 以下のコードは、GPR 4 の内容を左に 28 ビット回転させ、生成されたマスクの制御下で 32 個の符号ビットと結果をマージし、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターに Carry ビットを設定します。

```
# Assume GPR 4 contains 0x9000 3000.
srawi 6,4,0x4
# GPR 6 now contains 0xF900 0300.
```

2. 以下のコードは、GPR 4 の内容を左に 28 ビット回転させ、生成されたマスクの制御下にある 32 個の符号ビットと結果をマージし、結果を GPR 6 に入れ、演算の結果を反映するために固定小数点例外レジスターおよび条件レジスター・フィールド 0 にカリー・ビットを設定します。

```
# Assume GPR 4 contains 0xB004 3000.
srawi. 6,4,0x4
# GPR 6 now contains 0xFB00 4300.
# Condition Register Field 0 now contains 0x8.
```

関連概念

addze または aze (Add to Zero Extended) 命令

固定小数点プロセッサー

固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示

固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

srd (Shift Right Double Word) 命令

目的

汎用レジスターの内容を、別の汎用レジスターの内容によって指定されたビット数だけシフトします。

構文

ビット	VALUE
0 - 5	31
6 - 10	S
11 - 15	A
16 - 20	B
21 - 30	539
31	rc

POWER® ファミリー

サード *RA*、 *RS*、 *RB* (Rc=0)

サード *RA*、 *RS*、 *RB* (Rc=1)

説明

汎用レジスター (GPR) *RS* の内容は、GPR *RB* の下位 7 ビットによって指定されたビット数だけ右にシフトされます。 位置 63 からシフトアウトされたビットは失われます。 左側の空き位置にはゼロが指定されます。 結果は GRP *RA* に入れられます。 64 から 127 にシフトすると、結果はゼロになります。

変更されたその他のレジスター:

- 条件レジスター (CR0 フィールド):

  影響を受ける: LT、 GT、EQ、SO (Rc = 1 の場合)

パラメーター

項目	説明
RA	操作の結果のターゲット汎用レジスターを指定します。
RS	これは、thr シフト操作のオペランドが入っているソース汎用レジスターを指定します。
rb	下位 7 ビットは、オペランドをシフトする距離を指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

sre (右シフト拡張) 命令

目的

汎用レジスターの内容を、指定されたビット数だけ右にシフトし、ローテートされたデータのコピーを MQ レジスターに入れ、その結果を汎用レジスターに入れます。

注: sre 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	665
31	rc

POWER® ファミリー

スレ *RA*、*RS*、*RB*

スレ *RA*、*RS*、*RB*

説明

スレ 命令は、ソース汎用レジスター (GPR) の内容を「*RS*」左側に 32 マイナス *N* ビット回転させます。ここで、 *N* は GPR 要求ブロックのビット 27 から 31 で指定されたシフト量であり、 MQ レジスターと、回転されたワードと生成されたマスクの論理 AND を GPR *RA* とはに保管します。 マスクは、 *N* 個のゼロと、それに続く 32 個のマイナス *N* 個の 1 で構成されます。

sre 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
スレ	なし	なし	0	なし
スレ	なし	なし	1	LT、GT、EQ、SO

sre 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容を左に 20 ビット回転させ、回転されたデータのコピーを MQ レジスターに入れ、回転されたデータとマスクを AND 演算した結果を GPR 6 に入れます。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 5 contains 0x0000 000C.
sre 6,4,5
# GPR 6 now contains 0x0009 0003.
# The MQ Register now contains 0x0009 0003.
```

2. 以下のコードは、GPR 4 の内容を 17 ビット左に回転させ、回転されたデータのコピーを MQ レジスターに入れ、回転されたデータをマスクと AND 演算した結果を GPR 6 に入れ、演算の結果を反映するために条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 5 contains 0x0000 000F.
sre. 6,4,5
# GPR 6 now contains 0x0001 6008.
# The MQ Register now contains 0x6001 6008.
# Condition Register Field 0 now contains 0x4.
```

srea (Shift Right Extended Algebraic) 命令

目的

汎用レジスターの内容を指定されたビット数だけ左に回転させ、回転されたデータのコピーを MQ レジスターに入れ、回転されたワードと、マスクの制御下にある汎用レジスターからの 32 個の符号ビットのワードをマージし、その結果を別の汎用レジスターに入れます。

注: srea 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	921
31	rc

POWER® ファミリー

srea (srea) *RA*、 *RS*、 *RB*

srea。 *RA*、 *RS*、 *RB*

説明

The **スレーア** instruction rotates the contents of the source general-purpose register (GPR) 「*RS*」 to the left by 32 minus *N* bits, where *N* is the shift amount specified in bits 27-31 of GPR 要求ブロック, stores the rotated word in the MQ Register, and merges the rotated word and a word of 32 sign bits from GPR 「*RS*」 under control of a generated mask. 32 個の符号ビットのワードは、汎用レジスターの符号ビットを取り、それを 32 回繰り返してフルワードにすることによって生成されます。 このワードは、汎用レジスターの値に応じて、0x0000 0000 または 0xFFFF FFFF のいずれかになります。 マスクは、*N* 個のゼロと、それに続く 32 個のマイナス *N* 個の 1 で構成されます。 マージされたワードは、GPR *RA* に保管されます。

この命令は、生成されたマスクの補数でローテーションされたデータを AND 演算し、32 ビットの結果を OR 演算し、GPR *RS* のビット 0 でビット結果を AND 演算して、カリー・ビット (CA) を生成します。

srea 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	**説明**			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
srea (srea)	なし	CA	0	なし
srea (srea)	なし	CA	1	LT、GT、EQ、SO

srea 命令の 2 つの構文形式は、常に固定小数点例外レジスターのカリー・ビット (CA) に影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	**説明**
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容を 28 ビットずつ左に回転させ、生成されたマスクの制御下にある 32 個の符号ビットと結果をマージし、回転されたワードを MQ レジスターに入れ、結果を GPR 6 に入れ、演算の結果を反映するために固定小数点例外レジスターにカリー・ビットを設定します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 7 contains 0x0000 0004.
srea 6,4,7
# GPR 6 now contains 0xF900 0300.
# The MQ Register now contains 0x0900 0300.
```

2. 以下のコードは、GPR 4 の内容を 28 ビット左に回転させ、生成されたマスクの制御下にある 32 個の符号ビットと結果をマージし、回転されたワードを MQ レジスターに入れ、結果を GPR 6 に入れ、演算の結果を反映するために固定小数点例外レジスターおよび条件レジスター・フィールド 0 にカリー・ビットを設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 7 contains 0x0000 0004.
srea. 6,4,7
# GPR 6 now contains 0xFB00 4300.
# The MQ Register now contains 0x0B00 4300.
# Condition Register Field 0 now contains 0x8.
```

関連概念

addze または aze (Add to Zero Extended) 命令

固定小数点プロセッサー

固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示

固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

sreq (MQ による右シフト拡張) 命令

目的

汎用レジスターの内容を指定されたビット数だけ左に回転させ、生成されたマスクの制御下にある MQ レジスターの内容と結果をマージし、回転されたワードを MQ レジスターに入れ、マージされた結果を別の汎用レジスターに入れます。

注: sreq 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	729
31	rc

POWER® ファミリー

要求 (sreq) *RA*、 *RS*、 *RB*

sreq: *RA*、 *RS*、 *RB*

説明

SREQ 命令は、ソース汎用レジスター (GPR) の内容を 「*RS*」 左側に 32 から *N* ビット分だけ回転させます。ここで、 *N* は GPR 要求ブロックのビット 27 から 31 に指定されたシフト量であり、ローテーションされたワードを、生成されたマスクの下の MQ レジスターの内容とマージされたワード *RA* とはにマージされ、ローテートされたワードを MQ レジスターに保管します。 マスクは、 *N* 個のゼロと、それに続く 32 個のマイナス *N* 個の 1 で構成されます。

sreq 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
要求 (sreq)	なし	なし	0	なし
sreq:	なし	なし	1	LT、GT、EQ、SO

sreq 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。

項目	説明
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容を左に 28 ビット回転させ、ローテーションされたデータを生成されたマスクの下の MQ レジスターの内容とマージし、ローテーションされたワードを MQ レジスターに入れ、結果を GPR 6 に入れます。

```
# Assume GPR 4 contains 0x9000 300F.
# Assume GPR 7 contains 0x0000 0004.
# Assume the MQ Register contains 0xEFFF FFFF.
sreq 6,4,7
# GPR 6 now contains 0xE900 0300.
# The MQ Register now contains 0xF900 0300.
```

2. 以下のコードは、GPR 4 の内容を左に 28 ビット回転させ、ローテーションされたデータを生成されたマスクの下の MQ レジスターの内容とマージし、ローテーションされたワードを MQ レジスターに入れ、結果を GPR 6 に入れ、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB00 300F.
# Assume GPR 18 contains 0x0000 0004.
# Assume the MQ Register contains 0xEFFF FFFF
sreq. 6,4,18
# GPR 6 now contains 0xEB00 0300.
# The MQ Register now contains 0xFB00 0300.
# Condition Register Field 0 now contains 0x8.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

sriq (MQ による即時シフト) 命令

目的

汎用レジスターの内容を指定されたビット数だけ右にシフトし、ローテートされた内容を MQ レジスターに入れ、その結果を別の汎用レジスターに入れます。

注: sriq 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	SH
21 - 30	696
31	rc

POWER® ファミリー

sriq (sriq) *RA*、 *RS*、 *SH*

sriq: *RA*、 *RS*、 *SH*

説明

SREQ 命令は、ソース汎用レジスター (GPR) の内容を 「*RS*」 左に 32 マイナス *N* ビットに回転させます。 ここで、*N* は *SH* によって指定されたシフト量であり、循環したワードを MQ レジスターに保管し、循環したワードと生成されたマスクの論理 AND を GPR に保管します。 *RA* とは マスクは、 *N* 個のゼロと、それに続く 32 個のマイナス *N* 個の 1 で構成されます。

sriq 命令には 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
sriq (sriq)	なし	なし	0	なし
sriq:	なし	なし	1	LT、GT、EQ、SO

sriq 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
SH	シフト量の値を指定します。

例

1. 以下のコードは、GPR 4 の内容を左側に 20 ビット回転させ、生成されたマスクで回転されたデータを AND 演算し、回転されたワードを MQ レジスターに入れ、結果を GPR 6 に入れます。

```
# Assume GPR 4 contains 0x9000 300F.
sriq 6,4,0xC
# GPR 6 now contains 0x0009 0003.
# The MQ Register now contains 0x00F9 0003.
```

2. 以下のコードは、GPR 4 の内容を 12 ビット左方に回転させ、生成されたマスクで回転されたデータを AND 演算し、回転されたワードを MQ レジスターに入れ、GPR 6 に結果を入れ、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB000 300F.
sriq. 6,4,0x14
# GPR 6 now contains 0x0000 0B00.
# The MQ Register now contains 0x0300 FB00.
# Condition Register Field 0 now contains 0x4.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示

固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

srliq (MQ を使用した Shift Right Long Immediate) 命令

目的

汎用レジスターの内容を指定されたビット数だけ左に回転させ、その結果を、生成されたマスクの制御下にある MQ レジスターの内容とマージし、その結果を別の汎用レジスターに入れます。

注: srliq 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	SH
21 - 30	760
31	rc

POWER® ファミリー

SRLIQ *RA*、*RS*、*SH*

srliq。 *RA*、*RS*、*SH*

説明

SRREQ 命令は、ソース汎用レジスター (GPR)「*RS*」の内容を左側に 32 から *N* ビットを引いた値に回転させます。ここで、*N* は *SH* によって指定されたシフト量であり、生成されたマスクの制御下にある MQ レジスターの内容と結果をマージし、ローテーションされたワードを MQ レジスターおよびマージされた結果を GPR *RA* とはに保管します。 マスクは、 *N* 個のゼロと、それに続く 32 個のマイナス *N* 個の 1 で構成されます。

srliq 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
SRLIQ	なし	なし	0	なし
srliq。	なし	なし	1	LT、GT、EQ、SO

srliq 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。

項目	説明
SH	シフト量の値を指定します。

例

1. 以下のコードは、GPR 4 の内容を左に 28 ビット回転させ、ローテーションされたデータを生成されたマスクの下の MQ レジスターの内容とマージし、ローテーションされたワードを MQ レジスターに入れ、結果を GPR 6 に入れます。

```
# Assume GPR 4 contains 0x9000 300F.
# Assume the MQ Register contains 0x1111 1111.
srliq 6,4,0x4
# GPR 6 now contains 0x1900 0300.
# The MQ Register now contains 0xF900 0300.
```

2. 以下のコードは、GPR 4 の内容を左に 28 ビット回転させ、ローテーションされたデータを生成されたマスクの下の MQ レジスターの内容とマージし、ローテーションされたワードを MQ レジスターに入れ、結果を GPR 6 に入れ、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000
# Assume the MQ Register contains 0xFFFF FFFF.
srliq. 6,4,0x4
# GPR 6 now contains 0xFB00 4300.
# The MQ Register contains 0x0B00 4300.
# Condition Register Field 0 now contains 0x8.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

srlq (MQ を使用した Shift Right Long) 命令

目的

汎用レジスターの内容を、指定されたビット数だけ左に回転させ、循環したデータまたはゼロのワードのいずれかを、生成されたマスクの制御下にある MQ レジスターの内容とマージし、その結果を汎用レジスターに入れます。

注: srlq 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	728
31	rc

POWER® ファミリー

SRLQ *RA*、 *RS*、 *RB*

srlq。 *RA*、 *RS*、 *RB*

説明

srlq 命令は、ソース汎用レジスター (GPR) *RS* の内容を左 32 から *N* ビットを引いたものに回転させます。 ここで、*N* は GPR *RB* のビット 27 から 31 で指定されたシフト量です。 マージは、GPR *RB* のビット 26 の値によって異なります。

srlq 命令を使用する際には、以下のことを考慮してください。

- GPR *RB* のビット 26 が 0 の場合、 *N* 個のゼロとそれに続く 32 個のマイナス *N* 個のマスクが生成されます。 ローテートされたワードは、この生成されたマスクの制御下で MQ レジスターの内容とマージされます。
- GPR *RB* のビット 26 が 1 の場合、 *N* 個の 1 とそれに続く 32 から *N* 個のゼロのマスクが生成されます。 次に、この生成されたマスクの制御下で、ゼロのワードが MQ レジスターの内容とマージされます。

マージされたワードは、GPR *RA* に保管されます。 MQ レジスターは変更されません。

srlq 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	**説明**			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
SRLQ	なし	なし	0	なし
srlq。	なし	なし	1	LT、GT、EQ、SO

srlq 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	**説明**
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容を左に 28 ビットずつ回転させ、ゼロのワードと MQ レジスターの内容をマスクの下にマージし、マージされた結果を GPR 6 に入れます。

```
# Assume GPR 4 contains 0x9000 300F.
# Assume GPR 8 contains 0x0000 0024.
# Assume the MQ Register contains 0xFFFF FFFF.
srlq 6,4,8
# GPR 6 now contains 0x0FFF FFFF.
# The MQ Register remains unchanged.
```

2. 以下のコードは、GPR 4 の内容を左に 28 ビット回転させ、回転されたデータをマスクの下の MQ レジスターの内容とマージし、マージされた結果を GPR 6 に入れ、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 8 contains 0x00000 0004.
# Assume the MQ Register contains 0xFFFF FFFF.
srlq. 6,4,8
# GPR 6 now holds 0xFB00 4300.
# The MQ Register remains unchanged.
# Condition Register Field 0 now contains 0x8.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

srq (MQ を使用した Shift Right) 命令

目的

汎用レジスターの内容を、指定されたビット数だけ左に回転させ、回転されたワードを MQ レジスターに入れ、回転されたワードの論理 AND と生成されたマスクを汎用レジスターに入れます。

注: srq 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	664
31	rc

POWER® ファミリー

SRQ *RA*、 *RS*、 *RB*

srq。 *RA*、 *RS*、 *RB*

説明

srq 命令は、ソース汎用レジスター (GPR) *RS* の内容を左に 32 から *N* ビットを引いた値に回転させます。 ここで、 *N* は GPR *RB* のビット 27 から 31 で指定されたシフト量で、回転後のワードを MQ レジスターに保管します。 マスクは、GPR *RB* のビット 26 に依存します。

srq 命令を使用する際には、以下のことを考慮してください。

- GPR *RB* のビット 26 が 0 の場合、 *N* 個のゼロとそれに続く 32 個のマイナス *N* 個のマスクが生成されます。
- GPR *RB* のビット 26 が 1 の場合は、すべてゼロのマスクが生成されます。

この命令は、回転されたワードと生成されたマスクの論理 AND を GPR *RA* に保管します。

srq 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
SRQ	なし	なし	0	なし
srq。	なし	なし	1	LT、GT、EQ、SO

srq 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容を左に 28 ビット回転させ、回転されたワードを MQ レジスターに入れ、回転されたワードと生成されたマスクの論理 AND を GPR 6 に入れます。

```
# Assume GPR 4 holds 0x9000 300F.
# Assume GPR 25 holds 0x0000 00024.
srq 6,4,25
# GPR 6 now holds 0x0000 0000.
# The MQ Register now holds 0xF900 0300.
```

2. 以下のコードは、GPR 4 の内容を左に 28 ビット回転し、回転後のワードを MQ レジスターに入れ、回転後のワードと生成されたマスクの論理 AND を GPR 6 に入れ、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 holds 0xB000 300F.
# Assume GPR 25 holds 0x0000 0004.
srq. 6,4,8
# GPR 6 now holds 0x0B00 0300.
# The MQ Register now holds 0xFB00 0300.
# Condition Register Field 0 now contains 0x4.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

srw または sr (Shift Right Word) 命令

目的

汎用レジスターの内容を、指定されたビット数だけ左方に回転させ、マスクされた結果を汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	536
31	rc

PowerPC (R)

SRW *RA*、 *RS*、 *RB*

srw: *RA*、 *RS*、 *RB*

POWER® ファミリー

sr *RA*、 *RS*、 *RB*

SR *RA*、 *RS*、 *RB*

説明

SRW 命令と **SR** 命令は、ソース汎用レジスター (GPR)「*RS*」 の内容を左に 32 から *N* ビットを引いた値に回転させます。ここで、 *N* は GPR 要求ブロックのビット 27 から 31 で指定されたシフト量であり、回転されたワードと生成されたマスクの論理 AND を GPR *RA* とはに保管します。

srw および **sr** 命令を使用する際には、以下のことを考慮してください。

- GPR *RB* のビット 26 が 0 の場合、 *N* 個のゼロとそれに続く 32- *N* 個のマスクが生成されます。
- GPR *RB* のビット 26 が 1 の場合は、すべてゼロのマスクが生成されます。

srw 命令と **sr** 命令には、それぞれ 2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
SRW	なし	なし	0	なし
srw:	なし	なし	1	LT、GT、EQ、SO
sr	なし	なし	0	なし
SR	なし	なし	1	LT、GT、EQ、SO

sr 命令の 2 つの構文形式、および **srw** 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、これらの命令は、条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目 **説明**

RA 操作の結果が保管されるターゲット汎用レジスターを指定します。

RS 操作のソース汎用レジスターを指定します。

項目	説明
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容を左に 28 ビット回転させ、ローテーションされたデータと生成されたマスクとの AND 演算の結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 5 contains 0x0000 0024.
srw 6,4,5
# GPR 6 now contains 0x0000 0000.
```

2. 以下のコードは、GPR 4 の内容を左側に 28 ビット回転させ、回転後のデータと生成されたマスクとの AND 演算の結果を GPR 6 に保管し、演算の結果を反映するために条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3001.
# Assume GPR 5 contains 0x0000 0004.
srw. 6,4,5
# GPR 6 now contains 0x0B00 4300.
# Condition Register Field 0 now contains 0x4.
```

関連概念

addze または aze (Add to Zero Extended) 命令

固定小数点プロセッサー

固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示

固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

stb (バイト保管) 命令

目的

汎用レジスターからの 1 バイトのデータを、メモリー内の指定された位置に保管します。

構文

ビット	VALUE
0 - 5	38
6 - 10	RS
11 - 15	RA
16 - 31	D

項目	説明
STB	*RS*、 *D*(*RA*)

説明

stb 命令は、汎用レジスター (GPR) *RS* のビット 24 から 31 を、有効アドレス (EA) によってアドレス指定された 1 バイトのストレージに保管します。

GPR *RA* が 0 でない場合、EA は、GPR *RA* および *D* の内容の合計です。これは、16 ビット符号付き 2 の補数の整数で、32 ビットに拡張されます。 GPR *RA* が 0 の場合、EA は *D* です。

stb 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RS	保管データのソース汎用レジスターを指定します。
D	EA 計算用に 16 ビットの符号付き 2 の補数整数符号拡張を 32 ビットに指定します。
RA	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、GPR 6 のビット 24 から 31 をメモリー内の場所に保管します。

```
.csect data[rw]
buffer: .long 0
# Assume GPR 4 contains address of csect data[rw].
# Assume GPR 6 contains 0x0000 0060.
.csect text[pr]
stb 6,buffer(4)
# 0x60 is now stored at the address of buffer.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

stbu (Store Byte with Update) 命令

目的

汎用レジスターからの 1 バイトのデータをメモリー内の指定された位置に保管し、場合によってはアドレスを別の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	39
6 - 10	RS
11 - 15	RA
16 - 31	D

項目	説明
STBU	*RS*、 *D*(*RA*)

説明

stbu 命令は、ソース汎用レジスター (GPR) *RS* のビット 24 から 31 を、有効アドレス (EA) によってアドレス指定されたストレージ内のバイトに保管します。

GPR *RA* が 0 でない場合、EA は、GPR *RA* および *D* の内容の合計です。これは、16 ビット符号付き 2 の補数の整数で、32 ビットに拡張されます。 GPR *RA* が 0 の場合、EA は *D* です。

RA が 0 でなく、ストレージ・アクセスによって位置合わせ割り込みが発生しない場合、EA は GPR *RA* に保管されます。

stbu 命令は 1 つの構文形式を持ち、固定小数点例外レジスターまたは条件レジスターのフィールド 0 には影響しません。

パラメーター

項目	説明
RS	保管データのソース汎用レジスターを指定します。
D	EA 計算用に 16 ビットの符号付き 2 の補数整数符号拡張を 32 ビットに指定します。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。

例

以下のコードは、GPR 6 のビット 24 から 31 をメモリー内の位置に保管し、そのアドレスを GPR 16 に入れます。

```
.csect data[rw]
buffer: .long 0
# Assume GPR 6 contains 0x0000 0060.
# Assume GPR 16 contains the address of csect data[rw].
.csect text[pr]
stbu 6,buffer(16)
# GPR 16 now contains the address of buffer.
# 0x60 is stored at the address of buffer.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管 (更新指示あり)
ロード・インストラクションとストア・インストラクションには更新フォームがあります。このフォームでは、メモリーとの間で通常の情報の移動に加えて、基本 GPR が EA で更新されます。

stbux (索引付き更新バイト保管) 命令

目的

汎用レジスターからの 1 バイトのデータをメモリー内の指定された位置に保管し、場合によってはアドレスを別の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	247
31	/

項目	説明
STBUX	*RS*、 *RA*、 *RB*

説明

stbux 命令は、ソース汎用レジスター (GPR) *RS* のビット 24 から 31 を、有効アドレス (EA) によってアドレス指定されたストレージ内のバイトに保管します。

GPR *RA* が 0 でない場合、EA は、GPR *RA* の内容と GPR *RB* の内容の合計です。 *RA* が 0 の場合、EA は GPR *RB* の内容です。

GPR *RA* が 0 でなく、ストレージ・アクセスによって位置合わせ割り込みが発生しない場合、EA は GPR *RA* に保管されます。

stbux 命令は 1 つの構文形式でのみ存在し、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RS	保管データのソース汎用レジスターを指定します。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、GPR 6 の内容をメモリー内の場所に保管し、そのアドレスを GPR 4 に入れます。

```
.csect data[rw]
buffer: .long 0
# Assume GPR 6 contains 0x0000 0060.
# Assume GPR 4 conteains 0x0000 0000.
# Assume GPR 19 contains the address of buffer.
.csect text[pr]
stbux 6,4,19
# Buffer now contains 0x60.
# GPR 4 contains the address of buffer.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管 (更新指示あり)
ロード・インストラクションとストア・インストラクションには更新フォームがあります。このフォームでは、メモリーとの間で通常の情報の移動に加えて、基本 GPR が EA で更新されます。

stbx (索引付きバイト保管) 命令

目的

汎用レジスターの 1 バイトを、メモリー内の指定された位置に保管します。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	215

ビット	VALUE
31	/

項目	説明
STBX	*RS*、 *RA*、 *RB*

説明

stbx 命令は、汎用レジスター (GPR) *RS* のビット 24 から 31 を、有効アドレス (EA) によってアドレス指定された 1 バイトのストレージに保管します。 GPR *RS* の内容は変更されません。

GPR *RA* が 0 でない場合、EA は、GPR *RA* の内容と GPR *RB* の内容の合計です。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

stbx 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスターのフィールド 0 には影響しません。

パラメーター

項目	説明
RS	保管データのソース汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、GPR 6 のビット 24 から 31 をメモリー内の場所に保管します。

```
.csect data[rw]
buffer: .long 0
# Assume GPR 4 contains the address of buffer.
# Assume GPR 6 contains 0x4865 6C6F.
.csect text[pr]
stbx 6,0,4
# buffer now contains 0x6F.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

std (ダブルワードの保管) 命令

目的

汎用レジスターからのダブルワードのデータを、指定されたメモリー・ロケーションに保管します。

構文

ビット	VALUE
0 - 5	62
6 - 10	RS

ビット	VALUE
11 - 15	RA
16 - 29	DS
30 - 31	0

PowerPC 64

std *RS*、 *Disp*(*RA*)

説明

std 命令は、ソース汎用レジスター (GPR) *RS* から、有効アドレス (EA) によって参照されるメモリー内の指定された位置に、ダブルワードをストレージに保管します。

DS は 14 ビットの符号付き 2 の補数で、64 ビットに符号拡張され、さらに 4 を乗算して変位 *Disp* を提供します。 GPR *RA* が 0 でない場合、EA は GPR *RA* と *Disp* の内容の合計です。 GPR *RA* が 0 の場合、EA は *Disp* です。

パラメーター

項目	説明
RS	データが入っているソース汎用レジスターを指定します。
処理	4 の倍数である 16 ビットの符号付き数値を指定します。 アセンブラーは、命令の生成時にこの数値を 4 で除算します。
RA	EA 計算用のソース汎用レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

stdcx。 (Store Double Word Conditional Indexed) 命令

目的

既存の予約に基づいて、条件付きで汎用レジスターの内容を保管場所に保管します。

構文

ビット	VALUE
0 - 5	31
6 - 10	S
11 -- 15	A
16 - 20	B
21 - 30	214
31	1

POWER® ファミリー

stdcx。 *RS*、 *RA*、 *RB*

説明

予約が存在する場合、および **stdcx** によって指定されたメモリー・アドレス。 命令は、予約を設定したロードおよび予約命令によって指定されたものと同じです。 *RS* の内容は、有効アドレス (EA) によってアドレス指定されたメモリー内のダブルワードに保管されます。予約はクリアされます。

GPR *RA* が 0 でない場合、EA は、GPR *RA* および *D* (変位のサイズ) の内容の合計 (16 ビット、符号付き 2 の補数整数、フルワード位置合わせ、64 ビットへの符号拡張) です。 GPR *RA* が 0 の場合、EA は *D* です。

予約が存在するが、 **stdcx** によって指定されたメモリー・アドレスが存在する場合。 命令は、予約を設定したロードおよび予約命令によって指定されたものとは異なります。予約はクリアされ、EA によってアドレス指定されたメモリー内のダブルワードに *RS* の内容が保管されるかどうかは定義されません。

予約が存在しない場合、命令はメモリーを変更せずに完了します。

保管が正常に実行されると、条件レジスター・フィールド 0 のビット 0 から 2 は 0b001 に設定され、それ以外の場合は 0b000 に設定されます。 XER の SO ビットは、条件レジスター・フィールド 0 のビット 4 にコピーされます。

EA は 8 の倍数でなければなりません。 そうでない場合は、システム位置合わせ例外ハンドラーが呼び出されるか、結果が予期せずに未定義になります。

正しく使用すると、ロードおよび予約および保管の条件付き命令は、メモリーの単一の位置合わせされたワード (ロード・ワードおよび予約および保管ワード条件付き) またはダブルワード (ダブルワードおよび予約および保管ダブルワード条件付き) に対してアトミック更新機能を提供することができます。

一般に、正しい使用法では、ロード・ワードと予約はストア・ワードの条件付きと対になっている必要があり、ダブルワードと予約は、ペアの両方の命令で指定された同じメモリー・アドレスを使用して、ストア・ダブルワード条件付きでロードする必要があります。 唯一の例外は、ペアになっていないワード条件付き保管命令またはダブルワード条件付き保管命令を任意の (スクラッチ) EA に使用して、プロセッサーが保持している予約をクリアできることです。

以下のいずれかのイベントが発生すると、予約はクリアされます。

- 予約を保持しているプロセッサーは、別のロードおよび予約命令を実行します。これにより、最初の予約がクリアされ、新しい予約が設定されます。
- 予約を保留しているプロセッサーは、任意のアドレスに対して条件付き保管命令を実行します。
- 別のプロセッサーが、予約に関連したアドレスへの保管命令を実行する。
- 予約を保持しているプロセッサー以外のメカニズムは、予約に関連付けられたアドレスに保管します。

パラメーター

項目	説明
RS	保管データのソース汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

stdu (更新付きダブル・ワード保管) 命令

目的

汎用レジスターからのダブルワードのデータを、指定されたメモリー・ロケーションに保管します。 アドレス・ベースを更新します。

注 : この命令は、64 ビット・アプリケーションを実行する 64 ビット PowerPC プロセッサーでのみ使用してください。

構文

ビット	VALUE
0 - 5	62
6 - 10	RS
11 - 15	RA
16 - 29	DS
30 - 31	0b01

PowerPC 64

stdu (stdu) *RS*、 *Disp*(*RA*)

説明

stdu 命令は、ソース汎用レジスター (GPR) *RS* から実効アドレス (EA) によって参照されるメモリー内の指定された位置に、ダブルワードをストレージに保管します。

DS は 14 ビットの符号付き 2 の補数で、64 ビットに符号拡張され、さらに 4 を乗算して変位 *Disp* を提供します。 GPR *RA* が 0 でない場合、EA は GPR *RA* と *Disp* の内容の合計です。 GPR *RA* が 0 の場合、EA は *Disp* です。

GPR *RA* = 0 の場合、命令形式は無効です。

パラメーター

項目	説明
RS	データが入っているソース汎用レジスターを指定します。
処理	4 の倍数である 16 ビットの符号付き数値を指定します。 アセンブラーは、命令の生成時にこの数値を 4 で除算します。
RA	EA 計算用のソース汎用レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管 (更新指示あり)
ロード・インストラクションとストア・インストラクションには更新フォームがあります。このフォームでは、メモリーとの間で通常の情報の移動に加えて、基本 GPR が EA で更新されます。

stdux (索引更新付きダブルワード保管) 命令

目的

汎用レジスターからのダブルワードのデータを、指定されたメモリー・ロケーションに保管します。 アドレス・ベースを更新します。

構文

ビット	VALUE
0 - 5	31
6 - 10	S
11 - 15	A
16 - 20	B
21 - 30	181
31	0

POWER® ファミリー

STDOX *RS*、 *RA*、 *RB*

説明

stdux 命令は、ソース汎用レジスター (GPR) *RS* から有効アドレス (EA) で指定されたストレージ内の位置に、ダブルワードをストレージに保管します。

EA は、GPR *RA* と *RB* の内容の合計です。 GRP *RA* が EA で更新されました。

rA = 0 の場合、命令形式は無効です。

パラメーター

項目	説明
RS	データが入っているソース汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

stdx (ダブルワード索引付き保管) 命令

目的

汎用レジスターからのダブルワードのデータを、指定されたメモリー・ロケーションに保管します。

構文

ビット	VALUE
0 - 5	31
6 - 10	S
11 - 15	A
16 - 20	B
21 - 30	149
31	0

POWER® ファミリー

標準 (stdx) *RS*、 *RA*、 *RB*

説明

stdx 命令は、ソース汎用レジスター (GPR) *RS* からストレージ内のダブルワードを、有効アドレス (EA) によって指定されたストレージ内の位置に保管します。

GPR *RA* が 0 でない場合、EA は GPR *RA* と *RB* の内容の合計です。 GPR *RA* が 0 の場合、EA は *RB* です。

パラメーター

項目	説明
RS	データが入っているソース汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

stfd (浮動小数点倍精度浮動小数点保管) 命令

目的

メモリー内の指定された場所にダブルワードのデータを保管します。

構文

ビット	VALUE
0 - 5	54
6 - 10	FRS
11 - 15	RA
16 - 31	D

項目	説明
STFD	*FRS*、 *D*(*RA*)

説明

stfd 命令は、浮動小数点レジスター (FPR) *FRS* の内容を、有効アドレス (EA) によってアドレス指定されたダブルワード・ストレージに保管します。

汎用レジスター (GPR) *RA* が 0 でない場合、EA は GPR *RA* と *D* の内容の合計になります。 合計は、16 ビット符号付き 2 の補数整数符号で、32 ビットに拡張されます。 GPR *RA* が 0 の場合、EA は *D* です。

stfd 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
FRS	保管データのソース浮動小数点レジスターを指定します。
D	EA 計算用に 32 ビットに拡張された a16-bit 符号付き 2 の補数整数符号を指定します。
RA	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、FPR 6 の内容をメモリー内の場所に保管します。

```
.csect data[rw]
buffer: .long 0,0
# Assume FPR 6 contains 0x4865 6C6C 6F20 776F.
# Assume GPR 4 contains the address of csect data[rw].
.csect text[pr]
stfd 6,buffer(4)
# buffer now contains 0x4865 6C6C 6F20 776F.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

stfdu (更新による浮動小数点の倍精度浮動小数点の保管) 命令

目的

データのダブルワードをメモリー内の指定された位置に保管し、場合によっては、アドレスを汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	55
6 - 10	FRS
11 - 15	RA
16 - 31	D

項目	説明
stfdu (stfdu)	*FRS*、 *D*(*RA*)

説明

stfdu 命令は、浮動小数点レジスター (FPR) *FRS* の内容を、有効アドレス (EA) によってアドレス指定されたダブルワード・ストレージに保管します。

汎用レジスター (GPR) *RA* が 0 でない場合、EA は GPR *RA* と *D* の内容の合計になります。 合計は、16 ビット符号付き 2 の補数整数符号で、32 ビットに拡張されます。 GPR *RA* が 0 の場合、EA は *D* です。

GPR *RA* が 0 ではなく、ストレージ・アクセスによって位置合わせ割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA は GPR *RA* に保管されます。

stfdu 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
FRS	保管データのソース浮動小数点レジスターを指定します。
D	EA 計算用に 32 ビットに拡張された 16 ビット符号付き 2 の補数整数符号を指定します。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。

例

以下のコードは、FPR 6 のダブルワードの内容をメモリー内の場所に保管し、そのアドレスを GPR 4 に保管します。

```
.csect data[rw]
buffer: .long 0,0
# Assume FPR 6 contains 0x4865 6C6C 6F20 776F.
# GPR 4 contains the address of csect data[rw].
.csect text[pr]
stfdu 6,buffer(4)
# buffer now contains 0x4865 6C6C 6F20 776F.
# GPR 4 now contains the address of buffer.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

stfdux (索引付き更新による浮動小数点倍精度浮動小数点の保管) 命令

目的

データのダブルワードをメモリー内の指定された位置に保管し、場合によっては、アドレスを汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	FRS
11 - 15	RA
16 - 20	RB
21 - 30	759
31	/

項目	説明
STFDOX	*FRS*、 *RA*、 *RB*

説明

stfdux 命令は、浮動小数点レジスター (FPR) *FRS* の内容を、有効アドレス (EA) によってアドレス指定されたダブルワード・ストレージに保管します。

汎用レジスター (GPR) *RA* が 0 でない場合、EA は、GPR *RA* と *RB* の内容の合計になります。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

GPR *RA* が 0 ではなく、ストレージ・アクセスによって位置合わせ割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA は GPR *RA* に保管されます。

stfdux 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
FRS	保管データのソース浮動小数点レジスターを指定します。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、FPR 6 の内容をメモリー内の場所に保管し、そのアドレスを GPR 4 に保管します。

```
.csect data[rw]
buffer: .long 0,0,0,0
# Assume FPR 6 contains 0x9000 3000 9000 3000.
# Assume GPR 4 contains 0x0000 0008.
# Assume GPR 5 contains the address of buffer.
.csect text[pr]
stfdux 6,4,5
# buffer+8 now contains 0x9000 3000 9000 3000.
# GPR 4 now contains the address of buffer+8.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

stfdx (浮動小数点二重索引付き保管) 命令

目的

メモリー内の指定された場所にダブルワードのデータを保管します。

構文

ビット	VALUE
0 - 5	31
6 - 10	FRS
11 - 15	RA
16 - 20	RB
21 - 30	727
31	/

項目	説明
標準偏差 (stfdx)	*FRS*、 *RA*、 *RB*

説明

stfdx 命令は、浮動小数点レジスター (FPR) *FRS* の内容を、有効アドレス (EA) によってアドレス指定されたダブルワード・ストレージに保管します。

汎用レジスター (GPR) *RA* が 0 でない場合、EA は、GPR *RA* と *RB* の内容の合計になります。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

stfdx 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
FRS	保管データのソース浮動小数点レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、FPR 6 の内容を、GPR 5 および GPR 4 によってアドレス指定されたメモリー内の場所に保管します。

```
.csect data[rw]
buffer: .long 0,0,0,0
# Assume FPR 6 contains 0x4865 6C6C 6F20 776F.
# Assume GPR 4 contains 0x0000 0008.
# Assume GPR 5 contains the address of buffer.
.csect text[pr]
stfdx 6,4,5
# 0x4865 6C6C 6F20 776F is now stored at the
# address buffer+8.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

stfiwx (浮動小数点を索引付き整数語として保管)

目的

指定された浮動小数点レジスターの下位 32 ビットを、メモリー内の指定されたワード位置に保管します。

> **注: stfiwx** 命令は、PowerPC ® アーキテクチャーでのみ定義され、オプションの命令です。 PowerPC 603 RISC マイクロプロセッサーおよび PowerPC 604 RISC マイクロプロセッサーでサポートされますが、PowerPC ® 601 RISC マイクロプロセッサーではサポートされません。

構文

ビット	VALUE
0 - 5	31

ビット	VALUE
6 - 10	FRS
11 - 15	RA
16 - 20	RB
21 - 30	983
31	/

項目	説明
stfiwx (stfiwx)	*FRS*、 *RA*、 *RB*

説明

stfifx 命令は、浮動小数点レジスター (FPR) *FRS* の下位 32 ビットの内容を変換せずに、有効アドレス (EA) によってアドレス指定されたワード・ストレージに保管します。

汎用レジスター (GPR) *RA* が 0 でない場合、EA は、GPR *RA* と *RB* の内容の合計になります。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

stfiwx 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスターのフィールド 0 には影響しません。

レジスター *FRS* の内容が、Load Floating Point Single Instruction、単精度算術命令、または **frsp** (Floating Round to Single Precision) 命令によって直接または間接的に生成された場合、保管される値は未定義です。(*FRS* の内容は、FRS がそのような命令のターゲット・レジスターである場合、その命令によって直接作成されます。 レジスター *FRS* の内容は、*FRS* が 1 つ以上の浮動小数点移動命令のシーケンスの最終ターゲット・レジスターであり、シーケンスの入力がそのような命令によって直接生成された場合に、そのような命令によって間接的に生成されます。)

パラメーター

項目	説明
FRS	保管データのソース浮動小数点レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、FPR 6 の内容を、GPR 5 および GPR 4 によってアドレス指定されたメモリー内の場所に保管します。

```
.csect data[rw]
buffer: .long 0,0,0,0
# Assume FPR 6 contains 0x4865 6C6C 6F20 776F.
# Assume GPR 4 contains 0x0000 0008.
# Assume GPR 5 contains the address of buffer.
.csect text[pr]
stfiwx 6,4,5
# 6F20 776F is now stored at the
# address buffer+8.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

stfq (浮動小数点クワッド保管) 命令

目的

2 つの倍精度値を 2 つの連続するダブルワード位置にメモリーに保管します。

> **注: stfq** 命令は、POWER® ファミリー・アーキテクチャーの POWER2™ 実装でのみサポートされます。

構文

ビット	VALUE
0 - 5	60
6 - 10	FRS
11 - 15	RA
16 - 29	DS
30 - 31	00

POWER2™

STFQ *FRS*、 *DS*(*RA*)

説明

stfq 命令は、有効アドレス (EA) で指定された位置にある 2 つの連続した浮動小数点レジスター (FPR) の内容をメモリーに保管します。

DS は 30 ビットまで符号拡張され、右側に b '00' が連結されてオフセット値を形成します。 汎用レジスター (GPR) *RA* が 0 の場合、オフセット値は EA になります。 GPR *RA* が 0 でない場合は、オフセット値が GPR *RA* に追加され、EA が生成されます。 FPR *FRS* の内容は、EA のストレージのダブルワードに保管されます。 FPR *FRS* が 31 の場合、FPR 0 の内容は EA+8; のダブルワードに保管されます。それ以外の場合、*FRS*+ 1 の内容は EA+8 のダブルワードに保管されます。

stfq 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
FRS	保管する値が入っている 2 つの浮動小数点レジスターのうちの最初のレジスターを指定します。
DS	EA 計算の即時値として使用される 14 ビット・フィールドを指定します。
RA	EA 計算用の 1 つのソース汎用レジスターを指定します。

関連概念

lfqux (索引付き更新付き浮動小数点クワッド・ロード) 命令

浮動小数点プロセッサー

浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

stf 屈曲 (Store Floating-Point Quad with Update) 命令

目的

2 つの倍精度値を 2 つの連続するダブルワード位置に保管し、アドレス・ベースを更新します。

> **注: stfqu** 命令は、POWER® ファミリー・アーキテクチャーの POWER2™ インプリメンテーションでのみサポートされます。

構文

ビット	VALUE
0 - 5	61
6 - 10	FRS
11 - 15	RA
16 - 29	DS
30 - 31	01

POWER2™

STF 屈曲 *FRS*、 *DS*(*RA*)

説明

stfqu 命令は、有効アドレス (EA) によって指定された位置にある 2 つの連続した浮動小数点レジスター (FPR) の内容をメモリーに保管します。

DS は 30 ビットまで符号拡張され、右側に b '00' が連結されてオフセット値を形成します。 汎用レジスター (GPR) *RA* が 0 の場合、オフセット値は EA になります。 GPR *RA* が 0 でない場合は、オフセット値が GPR *RA* に追加され、EA が生成されます。 FPR *FRS* の内容は、EA のストレージのダブルワードに保管されます。 FPR *FRS* が 31 の場合、FPR 0 の内容は EA+8; のダブルワードに保管されます。それ以外の場合、*FRS*+ 1 の内容は EA+8 のダブルワードに保管されます。

GPR *RA* が 0 でない場合、EA は GPR *RA* に置かれます。

stf 先 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
FRS	保管する値が入っている 2 つの浮動小数点レジスターのうちの最初のレジスターを指定します。
DS	EA 計算の即時値として使用される 14 ビット・フィールドを指定します。
RA	EA 計算用に 1 つのソース汎用レジスターを指定し、EA 更新用にターゲット・レジスターを指定します。

関連概念

lfqux (索引付き更新付き浮動小数点クワッド・ロード) 命令

浮動小数点プロセッサー

浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

stfqux (更新索引付き浮動小数点クワッド保管) 命令

目的

2 つの倍精度値を 2 つの連続するダブルワード位置に保管し、アドレス・ベースを更新します。

> **注: stfqux** 命令は、POWER ® ファミリー・アーキテクチャーの POWER2™ 実装でのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	FRS
11 - 15	RA
16 - 20	RB
21 - 30	951
31	rc

POWER2™

stfqux (stfqux) *FRS*、 *RA*、 *RB*

説明

stfqux 命令は、有効アドレス (EA) で指定された位置に、2 つの連続した浮動小数点レジスター (FPR) の内容をメモリーに保管します。

汎用レジスター (GPR) *RA* が 0 でない場合、EA は GPR *RA* および GPR *RB* の内容の合計になります。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。 FPR *FRS* の内容は、EA のストレージのダブルワードに保管されます。 FPR *FRS* が 31 の場合、FPR 0 の内容は EA+8; のダブルワードに保管されます。それ以外の場合、*FRS*+ 1 の内容は EA+8 のダブルワードに保管されます。

GPR *RA* が 0 でない場合、EA は GPR *RA* に置かれます。

stfqux 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスターのフィールド 0 には影響しません。

パラメーター

項目	説明
FRS	保管する値が入っている 2 つの浮動小数点レジスターのうちの最初のレジスターを指定します。
RA	EA 計算用の最初のソース汎用レジスターと EA 更新用のターゲット・レジスターを指定します。
rb	EA 計算の 2 番目のソース汎用レジスターを指定します。

関連概念

lfqux (索引付き更新付き浮動小数点クワッド・ロード) 命令

浮動小数点プロセッサー

浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

stfqx (浮動小数点クワッド索引付き保管) 命令

目的

2 つの倍精度値を 2 つの連続するダブルワード位置にメモリーに保管します。

> **注: stfqx** 命令は、POWER® ファミリー・アーキテクチャーの POWER2™ 実装でのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	FRS
11 - 15	RA
16 - 20	RB
21 - 30	919
31	rc

POWER2™

STFQX *FRS*、 *RA*、 *RB*

説明

stfqx 命令は、有効アドレス (EA) で指定された位置に浮動小数点レジスター (FPR) *FRS* の内容をメモリーに保管します。

汎用レジスター (GPR) *RA* が 0 でない場合、EA は GPR *RA* および GPR *RB* の内容の合計になります。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。 FPR *FRS* の内容は、EA のストレージのダブルワードに保管されます。 FPR *FRS* が 31 の場合、FPR 0 の内容は EA+8; のダブルワードに保管されます。それ以外の場合、*FRS*+ 1 の内容は EA+8 のダブルワードに保管されます。

stfqx 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
FRS	保管する値が入っている 2 つの浮動小数点レジスターのうちの最初のレジスターを指定します。
RA	EA 計算用の 1 つのソース汎用レジスターを指定します。
rb	EA 計算の 2 番目のソース汎用レジスターを指定します。

関連概念

lfqux (索引付き更新付き浮動小数点クワッド・ロード) 命令

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

stfs (浮動小数点単一保管) 命令

目的

浮動小数点レジスターからのデータのワードを、メモリー内の指定された位置に保管します。

構文

ビット	VALUE
0 - 5	52
6 - 10	FRS
11 - 15	RA
16 - 31	D

項目	説明
stfs (stfs)	*FRS*、 *D*(*RA*)

説明

stfs 命令は、浮動小数点レジスター (FPR) *FRS* の内容を単精度に変換し、その結果を有効アドレス (EA) によってアドレス指定されたストレージのワードに保管します。

汎用レジスター (GPR) *RA* が 0 でない場合、EA は GPR *RA* および *D* の内容の合計であり、16 ビット符号付き 2 の補数符号-32 ビットに拡張されます。 GPR *RA* が 0 の場合、EA は *D* です。

stfs 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスターのフィールド 0 には影響しません。

パラメーター

項目	説明
FRS	保管データの浮動小数点レジスターを指定します。
D	EA 計算用に 16 ビットの符号付き 2 の補数整数符号拡張を 32 ビットに指定します。
RA	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、FPR 6 の単精度の内容をメモリー内の場所に保管します。

```
.csect data[rw]
buffer: .long 0
# Assume FPR 6 contains 0x4865 6C6C 6F20 776F.
# Assume GPR 4 contains the address of csect data[rw].
.csect text[pr]
stfs 6,buffer(4)
# buffer now contains 0x432B 6363.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

stfsu (Store Floating-Point Single with Update) 命令

目的

浮動小数点レジスターからのデータのワードをメモリー内の指定された位置に保管し、アドレスを汎用レジスターに入れることができます。

構文

ビット	VALUE
0 - 5	53
6 - 10	FRS
11 - 15	RA
16 - 31	D

項目	説明
stfsu (stfsu)	*FRS*、 *D*(*RA*)

説明

stfsu 命令は、浮動小数点レジスター (FPR) *FRS* の内容を単精度に変換し、その結果を有効アドレス (EA) によってアドレス指定されたストレージのワードに保管します。

汎用レジスター (GPR) *RA* が 0 でない場合、EA は GPR *RA* および *D* の内容の合計であり、16 ビット符号付き 2 の補数符号-32 ビットに拡張されます。 GPR *RA* が 0 の場合、EA は *D* です。

GPR *RA* が 0 ではなく、ストレージ・アクセスによって位置合わせ割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA は GPR *RA* に保管されます。

stfsu 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
FRS	保管データの浮動小数点レジスターを指定します。
D	EA 計算用に 16 ビットの符号付き 2 の補数整数符号拡張を 32 ビットに指定します。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。

例

以下のコードは、FPR 6 の単精度の内容をメモリー内の場所に保管し、そのアドレスを GPR 4 に保管します。

```
.csect data[rw]
buffer: .long 0
# Assume FPR 6 contains 0x4865 6C6C 6F20 776F.
# Assume GPR 4 contains the address of csect data[rw].
.csect text[pr]
stfsu 6,buffer(4)
# GPR 4 now contains the address of buffer.
# buffer now contains 0x432B 6363.
```

関連概念

浮動小数点プロセッサー

浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

stfsux (Store Floating-Point Single with Update Indexed) 命令

目的

浮動小数点レジスターからのデータのワードをメモリー内の指定された位置に保管し、アドレスを汎用レジスターに入れることができます。

構文

ビット	VALUE
0 - 5	31
6 - 10	FRS
11 - 15	RA
16 - 20	RB
21 - 30	695
31	/

項目	説明
STFSUX	*FRS*、 *RA*、 *RB*

説明

stfsux 命令は、浮動小数点レジスター (FPR) *FRS* の内容を単精度に変換し、その結果を有効アドレス (EA) によってアドレス指定されたストレージのワードに保管します。

汎用レジスター (GPR) *RA* が 0 でない場合、EA は GPR *RA* および GPR *RB* の内容の合計になります。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

GPR *RA* が 0 ではなく、ストレージ・アクセスによって位置合わせ割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA は GPR *RA* に保管されます。

stfsux 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスターのフィールド 0 には影響しません。

パラメーター

項目	説明
FRS	保管データの浮動小数点レジスターを指定します。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、FPR 6 の単精度の内容をメモリー内の場所に保管し、そのアドレスを GPR 5 に保管します。

```
.csect data[rw]
buffer: .long 0,0,0,0
# Assume GPR 4 contains 0x0000 0008.
# Assume GPR 5 contains the address of buffer.
```

```
# Assume FPR 6 contains 0x4865 6C6C 6F20 776F.
.csect text[pr]
stfsux 6,5,4
# GPR 5 now contains the address of buffer+8.
# buffer+8 contains 0x432B 6363.
```

関連概念

浮動小数点プロセッサー

浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

stfsx (浮動小数点単一索引の保管) 命令

目的

浮動小数点レジスターからのデータのワードを、メモリー内の指定された位置に保管します。

構文

ビット	VALUE
0 - 5	31
6 - 10	FRS
11 - 15	RA
16 -20	RB
21 - 30	663
31	/

項目	説明
stfsx (stfsx)	*FRS*、 *RA*、 *RB*

説明

stfsx 命令は、浮動小数点レジスター (FPR) *FRS* の内容を単精度に変換し、その結果を有効アドレス (EA) によってアドレス指定されたストレージのワードに保管します。

汎用レジスター (GPR) *RA* が 0 でない場合、EA は GPR *RA* および GPR *RB* の内容の合計になります。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

stfsx 命令には 1 つの構文形式があり、浮動小数点状況および制御レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
FRS	保管データのソース浮動小数点レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、FPR 6 の単精度の内容をメモリー内の場所に保管します。

```
.csect data[rw]
buffer: .long 0
# Assume FPR 6 contains 0x4865 6C6C 6F20 776F.
# Assume GPR 4 contains the address of buffer.
.csect text[pr]
stfsx 6,0,4
# buffer now contains 0x432B 6363.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

sth (半分保管) 命令

目的

汎用レジスターからのデータのハーフワードを、メモリー内の指定された位置に保管します。

構文

ビット	VALUE
0 - 5	44
6 - 10	RS
11 - 15	RA
16 - 31	D

項目	説明
STH	*RS*、 *D*(*RA*)

説明

sth 命令は、汎用レジスター (GPR) *RS* のビット 16 から 31 を、有効アドレス (EA) によってアドレス指定されたストレージのハーフワードに保管します。

GPR *RA* が 0 でない場合、EA は、GPR *RA* および *D* の内容の合計です。これは、16 ビット符号付き 2 の補数の整数で、32 ビットに拡張されます。 GPR *RA* が 0 の場合、EA は *D* です。

sth 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RS	保管データのソース汎用レジスターを指定します。
D	EA 計算用に 32 ビットに拡張された a16-bit 符号付き 2 の補数整数符号を指定します。
RA	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、GPR 6 のビット 16 から 31 をメモリー内の場所に保管します。

```
.csect data[rw]
```

```
buffer: .long 0
# Assume GPR 4 contains the address of csect data[rw].
# Assume GPR 6 contains 0x9000 3000.
.csect text[pr]
sth 6,buffer(4)
# buffer now contains 0x3000.
```

関連概念

浮動小数点プロセッサー

浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

sthbrx (ハーフバイト逆索引付け保管) 命令

目的

汎用レジスターからのデータのハーフワードを、メモリー内の指定された位置に、2 バイト反転して保管します。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 -20	RB
21 - 30	918
31	/

項目	説明
sthbrx	*RS*、 *RA*、 *RB*

説明

sthbrx 命令は、汎用レジスター (GPR) *RS* のビット 16 から 31 を、有効アドレス (EA) によってアドレス指定されたストレージのハーフワードに保管します。

sthbrx 命令を使用する場合は、以下のことを考慮してください。

- GPR *RS* のビット 24 から 31 は、EA によってアドレス指定されたストレージのハーフワードのビット 00 から 07 に保管されます。
- GPR *RS* のビット 16 から 23 は、EA によってアドレス指定されたストレージ内のワードのビット 08 から 15 に保管されます。

GPR *RA* が 0 でない場合、EA は GPR *RA* と GPR *RB* の内容の合計です。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

sthbrx 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスターのフィールド 0 には影響しません。

パラメーター

項目	説明
RS	保管データのソース汎用レジスターを指定します。

項目	説明
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、GPR 6 のハーフワードの内容を格納し、バイトをメモリー内の位置に反転させます。

```
.csect data[rw]
buffer: .long 0
# Assume GPR 6 contains 0x9000 3456.
# Assume GPR 4 contains the address of buffer.
.csect text[pr]
sthbrx 6,0,4
# buffer now contains 0x5634.
```

関連概念

浮動小数点プロセッサー
浮動小数点プロセッサーは、算術演算、比較演算、およびその他の演算を実行するための命令を提供します。

浮動小数点ロードおよび保管命令

sthu (更新による半分の保管) 命令

目的

汎用レジスターからのデータのハーフワードをメモリー内の指定された位置に保管し、場合によってはアドレスを別の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	45
6 - 10	RS
11 - 15	RA
16 - 31	D

項目	説明
sthu (sthu)	*RS*、 *D*(*RA*)

説明

sthu 命令は、汎用レジスター (GPR) *RS* のビット 16 から 31 を、有効アドレス (EA) によってアドレス指定されたストレージのハーフワードに保管します。

GPR *RA* が 0 でない場合、EA は、GPR *RA* および *D* の内容の合計です。これは、16 ビット符号付き 2 の補数の整数で、32 ビットに拡張されます。 GPR *RA* が 0 の場合、EA は *D* です。

GPR *RA* が 0 ではなく、ストレージ・アクセスによって位置合わせ割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA は GPR *RA* に入れられます。

sthu 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスターのフィールド 0 には影響しません。

パラメーター

項目	説明
RS	保管データのソース汎用レジスターを指定します。
D	EA 計算用に 32 ビットに拡張された a16-bit 符号付き 2 の補数整数符号を指定します。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。

例

以下のコードは、GPR 6 のハーフワードの内容をメモリー位置に保管し、そのアドレスを GPR 4 に保管します。

```
.csect data[rw]
buffer: .long 0
# Assume GPR 6 contains 0x9000 3456.
# Assume GPR 4 contains the address of csect data[rw].
.csect text[pr]
sthu 6,buffer(4)
# buffer now contains 0x3456
# GPR 4 contains the address of buffer.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管 (更新指示あり)
ロード・インストラクションとストア・インストラクションには更新フォームがあります。このフォームでは、メモリーとの間で通常の情報の移動に加えて、基本 GPR が EA で更新されます。

sthux (索引付き更新付きストア・ハーフ) 命令

目的

汎用レジスターからのデータのハーフワードをメモリー内の指定された位置に保管し、場合によってはアドレスを別の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16	RB
21 - 30	439
31	/

項目	説明
sthux (sthux)	*RS*、 *RA*、 *RB*

説明

sthux 命令は、汎用レジスター (GPR) *RS* のビット 16 から 31 を、有効アドレス (EA) によってアドレス指定されたストレージのハーフワードに保管します。

GPR *RA* が 0 でない場合、EA は GPR *RA* と GPR *RB* の内容の合計です。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

GPR *RA* が 0 でなく、ストレージ・アクセスによって位置合わせ割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA はレジスター GPR *RA* に入れられます。

sthux 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスターのフィールド 0 には影響しません。

パラメーター

項目	説明
RS	保管データのソース汎用レジスターを指定します。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、GPR 6 のハーフワードの内容をメモリー位置に保管し、そのアドレスを GPR 4 に保管します。

```
.csect data[rw]
buffer: .long 0,0,0,0
# Assume GPR 6 contains 0x9000 3456.
# Assume GPR 4 contains 0x0000 0007.
# Assume GPR 5 contains the address of buffer.
.csect text[pr]
sthux 6,4,5
# buffer+0x07 contains 0x3456.
# GPR 4 contains the address of buffer+0x07.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管 (更新指示あり)
ロード・インストラクションとストア・インストラクションには更新フォームがあります。このフォームでは、メモリーとの間で通常の情報の移動に加えて、基本 GPR が EA で更新されます。

sthx (索引付き半分保管) 命令

目的

汎用レジスターからのデータのハーフワードを、メモリー内の指定された位置に保管します。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	407
31	/

項目	説明
sthx	*RS*、 *RA*、 *RB*

説明

sthx 命令は、汎用レジスター (GPR) *RS* のビット 16 から 31 を、有効アドレス (EA) によってアドレス指定されたストレージのハーフワードに保管します。

GPR *RA* が 0 でない場合、EA は GPR *RA* と GPR *RB* の内容の合計です。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

sthx 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RS	保管データのソース汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、GPR 6 のハーフワードの内容をメモリー内の場所に保管します。

```
.csect data[rw]
buffer: .long 0
# Assume GPR 6 contains 0x9000 3456.
# Assume GPR 5 contains the address of buffer.
.csect text[pr]
sthx 6,0,5
# buffer now contains 0x3456.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

stmw または stm (複数ワード保管) 命令

連続したレジスターの内容を、指定されたメモリー位置に保管します。

構文

ビット	VALUE
0 - 5	47
6 - 10	RT
11 - 15	RA
16 - 31	D

PowerPC (R)

STMW *RS*、 *D*(*RA*)

POWER® ファミリー

STM *RS*、 *D*(*RA*)

説明

stmw 命令および **stm** 命令は、汎用レジスター (GPR) *RS* から GPR 31 までの *N* 個の連続ワードを保管します。 ストレージは有効アドレス (EA) から始まります。 *N* は、32 から *RS* を引いた値に等しいレジスター番号です。

GPR *RA* が 0 でない場合、EA は GPR *RA* と *D* の内容の合計です。 合計は、16 ビット符号付き 2 の補数整数符号で、32 ビットに拡張されます。 GPR *RA* が 0 の場合、EA は *D* です。

stmw 命令には 1 つの構文形式があります。 EA が 4 の倍数でない場合、結果は予期しないものになります。

stm 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RS	保管データのソース汎用レジスターを指定します。
D	EA 計算用に 32 ビットに拡張された 16 ビット符号付き 2 の補数整数符号を指定します。
RA	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、GPR 29 から GPR 31 の内容をメモリー内の場所に保管します。

```
.csect data[rw]
buffer: .long 0,0,0
# Assume GPR 29 contains 0x1000 2200.
# Assume GPR 30 contains 0x1000 3300.
# Assume GPR 31 contains 0x1000 4400.
.csect text[pr]
stmw 29,buffer(4)
# Three consecutive words in storage beginning at the address
# of buffer are now 0x1000 2200 1000 3300 1000 4400.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

stq (ストア・クワッド・ワード) 命令

目的

汎用レジスターからのクワッド・ワードのデータを、指定されたメモリー・ロケーションに保管します。

構文

ビット	値
0 - 5	62
6 - 10	RS
11 - 15	RA
16 - 29	DS
30 - 31	0b10

PowerPC 64

標準 RS、 Disp(RA)

説明

stq 命令は、ソース汎用レジスター (GPR) *RS* および *RS+1* からのクワッド・ワードを、有効アドレス (EA) によって参照されるメモリー内の指定された位置に保管します。

DS は 14 ビットの符号付き 2 の補数で、64 ビットに符号拡張され、さらに 4 を乗算して変位 *Disp* を提供します。 GPR *RA* が 0 でない場合、EA は GPR *RA* と *Disp* の内容の合計です。 GPR *RA* が 0 の場合、EA は *Disp* です。

パラメーター

項目	説明
RS	データが入っているソース汎用レジスターを指定します。 *RS* が奇数の場合、命令形式は無効です。
処理	4 の倍数である 16 ビットの符号付き数値を指定します。 アセンブラーは、命令の生成時にこの数値を 4 で除算します。
RA	EA 計算用のソース汎用レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

stimal または stsi (Store String Word Immediate) 命令

目的

連続したレジスターからの連続したバイトを、メモリー内の指定された位置に保管します。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA

ビット	VALUE
16 - 20	NB (B)
21 - 30	725
31	/

PowerPC (R)

標準化 *RS*、 *RA*、 *NB*

POWER ® ファミリー

STSI *RS*、 *RA*、 *NB*

説明

stInstructions および **stsi** 命令は、GPR *RS* から GPR *RS* + *NR* -1 までの有効アドレス (EA) にある汎用レジスター (GPR) *RS* の左端バイトから始まる *N* 個の連続バイトを保管します。

GPR *RA* が 0 でない場合、EA は GPR *RA* の内容です。 *RA* が 0 の場合、EA は 0 です。

stStandard および **stsi** 命令を使用する場合は、以下の点を考慮してください。

- *NB* はバイト・カウントです。
- *RS* は開始レジスターです。
- *N* は *NB* で、これは保管するバイト数です。 *NB* が 0 の場合、 *N* は 32 です。
- *NR* は ceiling (N/4) です。これは、データを保管するレジスターの数です。

POWER ® ファミリー命令 **stsi** の場合、 MQ レジスターの内容は未定義です。

stprocessor 命令と **stsi** 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RS	保管データのソース汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
NB (NB)	EA 計算のバイト・カウントを指定します。

例

以下のコードは、GPR 6 から GPR 8 に含まれるバイトをメモリー内の場所に保管します。

```
.csect data[rw]
buffer: .long 0,0,0
# Assume GPR 4 contains the address of buffer.
# Assume GPR 6 contains 0x4865 6C6C.
# Assume GPR 7 contains 0x6F20 776F.
# Assume GPR 8 contains 0x726C 6421.
.csect text[pr]
stswi 6,4,12
# buffer now contains 0x4865 6C6C 6F20 776F 726C 6421.
```

関連概念

固定小数点プロセッサー

固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ストリング命令
固定小数点ストリング命令を使用すると、位置合わせを考慮せずに、ストレージからレジスターへ、またはレジスターからストレージへデータを移動することができます。

stswx または stsx (Store String Word Indexed) 命令

目的

連続したレジスターからの連続したバイトを、メモリー内の指定された位置に保管します。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	661
31	/

PowerPC (R)

STSWX *RS*、 *RA*、 *RB*

POWER® ファミリー

STSX *RS*、 *RA*、 *RB*

説明

stswx および **stsx** 命令は、汎用レジスター (GPR) *RS* から GPR *RS* + *NR* -1 を介して、レジスター *RS* の左端バイトから始まる *N* 個の連続バイトを有効アドレス (EA) に保管します。

GPR *RA* が 0 でない場合、EA は、GPR *RA* の内容と GPR *RB* の内容の合計です。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

stswx および **stsx** 命令を使用する場合は、以下のことを考慮してください。

- *XER25-31* には、バイト・カウントが入ります。
- *RS* は開始レジスターです。
- *N* は *XER25-31* で、これは保管するバイト数です。
- *NR* は ceiling (N/4) です。これは、データを保管するレジスターの数です。

POWER® ファミリー命令 **stsx** の場合、 MQ レジスターの内容は未定義です。

stswx および **stsx** 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RS	保管データのソース汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、GPR 6 から GPR 7 に含まれるバイトを、メモリー内のロケーションの指定されたバイトに保管します。

```
.csect data[rw]
buffer: .long 0,0,0
# Assume GPR 5 contains 0x0000 0007.
# Assume GPR 4 contains the address of buffer.
# Assume GPR 6 contains 0x4865 6C6C.
# Assume GPR 7 contains 0x6F20 776F.
# The Fixed-Point Exception Register bits 25-31 contain 6.
.csect text[pr]
stswx 6,4,5
# buffer+0x7 now contains 0x4865 6C6C 6F20.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ストリング命令
固定小数点ストリング命令を使用すると、位置合わせを考慮せずに、ストレージからレジスターへ、またはレジスターからストレージへデータを移動することができます。

stw または st (保管) 命令

目的

汎用レジスターからのデータのワードを、メモリー内の指定された位置に保管します。

構文

ビット	VALUE
0 - 5	36
6 - 10	RS
11 - 15	RA
16 - 31	D

PowerPC (R)

標準 *RS*、 *D*(*RA*)

POWER® ファミリー

st *RS*、 *D*(*RA*)

説明

stw 命令および **st** 命令は、汎用レジスター (GPR) *RS* からのワードを、有効アドレス (EA) によってアドレス指定されたストレージのワードに保管します。

GPR *RA* が 0 でない場合、EA は、GPR *RA* および *D* の内容の合計です。これは、16 ビット符号付き 2 の補数の整数で、32 ビットに拡張されます。 GPR *RA* が 0 の場合、EA は *D* です。

stw および **st** 命令は、1 つの構文形式を持ち、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RS	保管データのソース汎用レジスターを指定します。
D	EA 計算用に 32 ビットに拡張された a16-bit 符号付き 2 の補数整数符号を指定します。
RA	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、GPR 6 の内容をメモリー内の場所に保管します。

```
.csect data[rw]
buffer: .long 0,0
# Assume GPR 6 contains 0x9000 3000.
# Assume GPR 5 contains the address of buffer.
.csect text[pr]
stw 6,4(5)
# 0x9000 3000 is now stored at the address buffer+4.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

stwbrx または stbrx (Store Word Byte-Reverse Indexed) 命令

目的

汎用レジスターからのデータのバイト反転ワードを、メモリー内の指定された位置に保管します。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	662
31	/

PowerPC (R)

stwbrx (stwbrx) *RS*、 *RA*、 *RB*

POWER® ファミリー

stbrx (stbrx) *RS*、 *RA*、 *RB*

説明

stwbrx および **stbrx** 命令は、汎用レジスター (GPR) *RS* からのバイト反転ワードを、有効アドレス (EA) によってアドレス指定されたストレージのワードに保管します。

stwbrx および **stbrx** 命令を使用する場合は、以下のことを考慮してください。

- GPR *RS* のビット 24 から 31 は、EA によってアドレス指定されたストレージ内のワードのビット 00 から 07 に保管されます。
- GPR *RS* のビット 16 から 23 は、EA によってアドレス指定されたストレージ内のワードのビット 08 から 15 に保管されます。
- GPR *RS* のビット 08 から 15 は、EA によってアドレス指定されたストレージ内のワードのビット 16 から 23 に保管されます。
- GPR *RS* のビット 00 から 07 は、EA によってアドレス指定されたストレージ内のワードのビット 24 から 31 に保管されます。

GPR *RA* が 0 でない場合、EA は GPR *RA* と GPR *RB* の内容の合計です。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

stwbrx および **stbrx** 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスターのフィールド 0 には影響しません。

パラメーター

項目	説明
RS	保管データのソース汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、GPR 6 のバイト反転ワードをメモリー内の位置に保管します。

```
.csect data[rw]
buffer: .long 0
# Assume GPR 4 contains the address of buffer.
# Assume GPR 9 contains 0x0000 0000.
# Assume GPR 6 contains 0x1234 5678.
.csect text[pr]
stwbrx 6,4,9
# 0x7856 3412 is now stored at the address of buffer.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

stwcx。 (Store Word Conditional Indexed) 命令

目的

指定されたメモリー位置で読み取り/変更/書き込み操作をエミュレートするために、先行する **lwarx** 命令と一緒に使用します。

注: stwcx。 命令は、PowerPC ® アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31

ビット	VALUE
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	150
31	1

PowerPC (R)

stwcx。 *RS*、 *RA*、 *RB*

説明

stwcx 。 および **lwarx** 命令は、ストレージへの読み取り/変更/書き込み操作を実行するために使用される、基本的または単純な命令です。 ストアが実行される場合は、**stwcx** を使用します。 および **lwarx** 命令により、 **lwarx** 命令が実行されてから **stwcx** が実行されるまでの間に、他のプロセッサーまたはメカニズムによってターゲット・メモリーの位置が変更されていないことが保証されます。 命令が完了する。

stwcx を使用する場合は、以下の点を考慮してください。 指示:

- 汎用レジスター (GPR) *RA* が 0 の場合、有効アドレス (EA) は GPR *RB* の内容です。それ以外の場合、EA は GPR *RA* の内容と GPR *RB* の内容の合計です。
- **lwarx** 命令によって作成された予約が存在する場合、GPR *RS* の内容はストレージ内のワードに保管され、EA によってアドレス指定され、予約はクリアされます。 それ以外の場合、ストレージは変更されません。
- 保管が実行される場合、条件レジスター・フィールド 0 のビット 0 から 2 は 0b001 に設定され、それ以外の場合は 0b000 に設定されます。 XER の SO ビットは、条件レジスター・フィールド 0 のビット 4 にコピーされます。

stwcx 。 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。 EA が 4 の倍数でない場合、結果は未定義です。

パラメーター

項目	説明
RS	保管データのソース汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

1. 以下のコードは、ストレージ内のワードをアトミックにロードして置き換えることにより、「取り出しと保管」を実行します。

```
# Assume that GPR 4 contains the new value to be stored.
# Assume that GPR 3 contains the address of the word
# to be loaded and replaced.
loop:   lwarx   r5,0,r3          # Load and reserve
        stwcx.  r4,0,r3          # Store new value if still
                                 # reserved
        bne-    loop             # Loop if lost reservation
# The new value is now in storage.
# The old value is returned to GPR 4.
```

2. 以下のコードは、レジスター内の値をストレージ内のワードとアトミックに比較することにより、「比較とスワップ」を実行します。

```
# Assume that GPR 5 contains the new value to be stored after
# a successful match.
# Assume that GPR 3 contains the address of the word
# to be tested.
# Assume that GPR 4 contains the value to be compared against
# the value in memory.
loop:   lwarx   r6,0,r3          # Load and reserve
        cmpw    r4,r6            # Are the first two operands
                                 # equal?
        bne-    exit             # Skip if not equal
        stwcx.  r5,0,r3          # Store new value if still
                                 # reserved
        bne-    loop             # Loop if lost reservation
exit:   mr      r4,r6            # Return value from storage
# The old value is returned to GPR 4.
# If a match was made, storage contains the new value.
```

レジスターの値がストレージ内のワードと等しい場合は、2 番目のレジスターの値がストレージ内のワードに保管されます。 等しくない場合は、ストレージからのワードが最初のレジスターにロードされ、条件レジスター・フィールド 0 の EQ ビットが比較の結果を示すように設定されます。

関連概念

lwarx (Load Word and Reserve Indexed) 命令

処理とストレージ

プロセッサーは、メイン・メモリーおよびレジスターにデータを保管します。

stwu または stu (Store Word with Update) 命令

目的

汎用レジスターからのデータのワードをメモリー内の指定された位置に保管し、場合によってはアドレスを別の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	37
6 - 10	RS
11 - 15	RA
16 - 31	D

PowerPC (R)

STWU *RS*、 *D*(*RA*)

POWER® ファミリー

標準 *RS*、 *D*(*RA*)

説明

stwu および **stu** 命令は、汎用レジスター (GPR) *RS* の内容を、有効アドレス (EA) によってアドレス指定されたストレージのワードに保管します。

GPR *RA* が 0 でない場合、EA は、GPR *RA* および *D* の内容の合計です。これは、16 ビット符号付き 2 の補数の整数で、32 ビットに拡張されます。 GPR *RA* が 0 の場合、EA は *D* です。

GPR *RA* が 0 でなく、ストレージ・アクセスによって位置合わせ割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA は GPR *RA* に入れられます。

stwu および **stu** 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスターのフィールド 0 には影響しません。

パラメーター

項目	説明
RS	保管データの汎用レジスターを指定します。
D	Specifies16-bit 符号付き 2 の補数整数符号-EA 計算のために 32 ビットに拡張されます。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。

例

以下のコードは、GPR 6 の内容をメモリー内の場所に保管します。

```
.csect data[rw]
buffer: .long 0
# Assume GPR 4 contains the address of csect data[rw].
# Assume GPR 6 contains 0x9000 3000.
.csect text[pr]
stwu 6,buffer(4)
# buffer now contains 0x9000 3000.
# GPR 4 contains the address of buffer.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管 (更新指示あり)
ロード・インストラクションとストア・インストラクションには更新フォームがあります。このフォームでは、メモリーとの間で通常の情報の移動に加えて、基本 GPR が EA で更新されます。

stwux または stux (索引付きストア・ワード) 命令

目的

汎用レジスターからのデータのワードをメモリー内の指定された位置に保管し、場合によってはアドレスを別の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
212 - 30	183
31	/

PowerPC (R)

STWUX *RS*、 *RA*、 *RB*

POWER® ファミリー

スタブ *RS*、 *RA*、 *RB*

説明

stwux および **stux** 命令は、汎用レジスター (GPR) *RS* の内容を、有効アドレス (EA) によってアドレス指定されたストレージのワードに保管します。

GPR *RA* が 0 でない場合、EA は GPR *RA* と GPR *RB* の内容の合計です。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

GPR *RA* が 0 ではなく、ストレージ・アクセスによって位置合わせ割り込みまたはデータ・ストレージ割り込みが発生しない場合、EA は GPR *RA* に入れられます。

stwux 命令と **stux** 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RS	保管データのソース汎用レジスターを指定します。
RA	EA 計算および可能なアドレス更新のためのソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、GPR 6 の内容をメモリー内の場所に保管します。

```
.csect data[rw]
buffer: .long 0,0
# Assume GPR 4 contains 0x0000 0004.
# Assume GPR 23 contains the address of buffer.
# Assume GPR 6 contains 0x9000 3000.
.csect text[pr]
stwux 6,4,23
# buffer+4 now contains 0x9000 3000.
# GPR 4 now contains the address of buffer+4.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管 (更新指示あり)
ロード・インストラクションとストア・インストラクションには更新フォームがあります。このフォームでは、メモリーとの間で通常の情報の移動に加えて、基本 GPR が EA で更新されます。

stwx または stx (Store Word Indexed) 命令

目的

汎用レジスターからのデータのワードを、メモリー内の指定された位置に保管します。

構文

ビット	VALUE
0 - 5	31
6 - 10	RS
11 - 15	RA
16 - 20	RB
21 - 30	151

ビット	VALUE
31	/

PowerPC (R)

STWX *RS*、 *RA*、 *RB*

POWER ® ファミリー

stx (テキスト開始) *RS*、 *RA*、 *RB*

説明

stwx および **stx** 命令は、汎用レジスター (GPR) *RS* の内容を、有効アドレス (EA) によってアドレス指定されたストレージのワードに保管します。

GPR *RA* が 0 でない場合、EA は GPR *RA* と GPR *RB* の内容の合計です。 GPR *RA* が 0 の場合、EA は GPR *RB* の内容です。

stwx および **stx** 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
RS	保管データのソース汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

以下のコードは、GPR 6 の内容をメモリー内の場所に保管します。

```
.csect data[pr]
buffer: .long 0
# Assume GPR 4 contains the address of buffer.
# Assume GPR 6 contains 0x4865 6C6C.
.csect text[pr]
stwx 6,0,4
# Buffer now contains 0x4865 6C6C.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点ロードおよび保管の指示
固定小数点ロード命令は、有効アドレス (EA) によってアドレス指定された位置から GPR の 1 つに情報を移動します。

subf (減算元) 命令

目的

2 つの汎用レジスターの内容を減算し、その結果を 3 番目の汎用レジスターに入れます。

注: subf 命令は、PowerPC ® アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21	大江
22 - 30	40
31	rc

PowerPC (R)

サブ *RT*、 *RA*、 *RB*

サブ *RT*、 *RA*、 *RB*

サブフォ *RT*、 *RA*、 *RB*

subfo。 *RT*、 *RA*、 *RB*

詳しくは、 固定小数点演算命令の拡張ニーモニック を参照してください。

説明

subf 命令は、汎用レジスター (GPR) *RA* および 1 の内容の 1 の補数を GPR *RB* の内容に追加し、結果をターゲット GPR *RT* に保管します。

subf 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
サブ	0	なし	0	なし
サブ	0	なし	1	LT、GT、EQ、SO
サブフォ	1	SO、OV、CA	0	なし
subfo。	1	SO、OV、CA	1	LT、GT、EQ、SO

subf 命令の 4 つの構文形式は、固定小数点例外レジスターのカリー・ビット (CA) には影響しません。 構文形式がオーバーフロー例外 (OE) ビットを 1 に設定する場合、命令は固定小数点例外レジスターの要約オーバーフロー (SO) およびオーバーフロー (OV) ビットに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	EA 計算用のソース汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 10 の内容から GPR 4 の内容を減算し、その結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x8000 7000.
# Assume GPR 10 contains 0x9000 3000.
subf 6,4,10
# GPR 6 now contains 0x0FFF C000.
```

2. 以下のコードは、GPR 10 の内容から GPR 4 の内容を減算し、その結果を GPR 6 に保管し、条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0x0000 4500.
# Assume GPR 10 contains 0x8000 7000.
subf. 6,4,10
# GPR 6 now contains 0x8000 2B00.
```

3. 以下のコードは、GPR 10 の内容から GPR 4 の内容を減算し、その結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターにサマリー・オーバーフロー・ビットとオーバーフロー・ビットを設定します。

```
# Assume GPR 4 contains 0x8000 0000.
# Assume GPR 10 contains 0x0000 4500.
subfo 6,4,10
# GPR 6 now contains 0x8000 4500.
```

4. 以下のコードは、GPR 10 の内容から GPR 4 の内容を減算し、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターおよび条件レジスター・フィールド 0 にサマリー・オーバーフロー・ビットとオーバーフロー・ビットを設定します。

```
# Assume GPR 4 contains 0x8000 0000.
# Assume GPR 10 contains 0x0000 7000.
subfo. 6,4,10
# GPR 6 now contains 0x8000 7000.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

subfc または sf (持ち出しから減算) 命令

目的

汎用レジスターの内容を別の汎用レジスターの内容から減算し、その結果を 3 番目の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21	大江
22 - 30	8

ビット	VALUE
31	rc

PowerPC (R)

サブ FC *RT*、 *RA*、 *RB*

subfc。 *RT*、 *RA*、 *RB*

サブフコ *RT*、 *RA*、 *RB*

subfco: *RT*、 *RA*、 *RB*

POWER ® ファミリー

sf *RT*、 *RA*、 *RB*

sf: *RT*、 *RA*、 *RB*

SFO *RT*、 *RA*、 *RB*

SFO。 *RT*、 *RA*、 *RB*

詳しくは、 固定小数点演算命令の拡張ニーモニック を参照してください。

説明

subfc および **sf** 命令は、汎用レジスター (GPR) *RA* および 1 の内容の補数を GPR *RB* の内容に追加し、結果をターゲット GPR *RT* に保管します。

subfc 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

sf 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
サブ FC	0	CA	0	なし
subfc。	0	CA	1	LT、GT、EQ、SO
サブフコ	1	SO、OV、CA	0	なし
subfco:	1	SO、OV、CA	1	LT、GT、EQ、SO
sf	0	CA	0	なし
sf:	0	CA	1	LT、GT、EQ、SO
SFO	1	SO、OV、CA	0	なし
SFO。	1	SO、OV、CA	1	LT、GT、EQ、SO

subfc 命令の 4 つの構文形式、および **sf** 命令の 4 つの構文形式は、常に固定小数点例外レジスターの Carry ビット (CA) に影響を与えます。 構文形式がオーバーフロー例外 (OE) ビットを 1 に設定する場合、命令は固定小数点例外レジスターの要約オーバーフロー (SO) およびオーバーフロー (OV) ビットに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 10 の内容から GPR 4 の内容を減算し、結果を GPR 6 に保管し、演算の結果を反映するように Carry ビットを設定します。

```
# Assume GPR 4 contains 0x8000 7000.
# Assume GPR 10 contains 0x9000 3000.
subfc 6,4,10
# GPR 6 now contains 0x0FFF C000.
```

2. 以下のコードは、GPR 10 の内容から GPR 4 の内容を減算し、結果を GPR 6 に保管し、演算の結果を反映するために条件レジスター・フィールド 0 とキャリー・ビットを設定します。

```
# Assume GPR 4 contains 0x0000 4500.
# Assume GPR 10 contains 0x8000 7000.
subfc. 6,4,10
# GPR 6 now contains 0x8000 2B00.
```

3. 以下のコードは、GPR 10 の内容から GPR 4 の内容を減算し、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターにサマリー・オーバーフロー・ビット、オーバーフロー・ビット、およびキャリー・ビットを設定します。

```
# Assume GPR 4 contains 0x8000 0000.
# Assume GPR 10 contains 0x0000 4500.
subfco 6,4,10
# GPR 6 now contains 0x8000 4500.
```

4. 以下のコードは、GPR 10 の内容から GPR 4 の内容を減算し、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターおよび条件レジスター・フィールド 0 にサマリー・オーバーフロー・ビット、オーバーフロー・ビット、およびキャリー・ビットを設定します。

```
# Assume GPR 4 contains 0x8000 0000.
# Assume GPR 10 contains 0x0000 7000.
subfco. 6,4,10
# GPR 6 now contains 0x8000 7000.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

subfe または sfe (拡張から減算) 命令

目的

汎用レジスターの内容の 1 の補数を別の汎用レジスターの合計に加算してから、固定小数点例外レジスター・カリー・ビットの値を加算し、その結果を 3 番目の汎用レジスターに保管します。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	RB
21	大江
22 - 30	136
31	rc

PowerPC (R)

サブ *RT*、 *RA*、 *RB*

Subfe。 *RT*、 *RA*、 *RB*

サブフェオ (subfeo) *RT*、 *RA*、 *RB*

サブフェオ *RT*、 *RA*、 *RB*

POWER® ファミリー

sfe (sfe) *RT*、 *RA*、 *RB*

スフ *RT*、 *RA*、 *RB*

sfeo (sfeo) *RT*、 *RA*、 *RB*

sfeo: *RT*、 *RA*、 *RB*

説明

subfe および **sfe** 命令は、固定小数点例外レジスター・カリー・ビットの値、汎用レジスター (GPR) *RB* の内容、および GPR *RA* の内容の 1 の補数を追加し、結果をターゲット GPR *RT* に保管します。

subfe 命令には 4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

sfe 命令には 4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	オーバーフロー 例外 (OE)	固定小数点 例外レジスター	レコード ビット (RC)	条件 フィールド 0 の登録
サブ	0	CA	0	なし
Subfe。	0	CA	1	LT、GT、EQ、SO
サブフェオ (subfeo)	1	SO、OV、CA	0	なし
サブフェオ	1	SO、OV、CA	1	LT、GT、EQ、SO
sfe (sfe)	0	CA	0	なし
スフ	0	CA	1	LT、GT、EQ、SO
sfeo (sfeo)	1	SO、OV、CA	0	なし

項目	説明			
sfeo:	1	SO、OV、CA	1	LT、GT、EQ、SO

subfe 命令の 4 つの構文形式、および **sfe** 命令の 4 つの構文形式は、常に固定小数点例外レジスターの Carry ビット (CA) に影響します。 構文形式がオーバーフロー例外 (OE) ビットを 1 に設定する場合、命令は固定小数点例外レジスターの要約オーバーフロー (SO) およびオーバーフロー (OV) ビットに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容の 1 の補数、GPR 10 の内容、および固定小数点例外レジスター・キャリー・ビットの値を追加し、その結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 10 contains 0x8000 7000.
# Assume the Carry bit is one.
subfe 6,4,10
# GPR 6 now contains 0xF000 4000.
```

2. 以下のコードは、GPR 4 の内容の 1 の補数、GPR 10 の内容、および固定小数点例外レジスター・カーリー・ビットの値を加算し、結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0x0000 4500.
# Assume GPR 10 contains 0x8000 7000.
# Assume the Carry bit is zero.
subfe. 6,4,10
# GPR 6 now contains 0x8000 2AFF.
```

3. 次のコードは、GPR 4 の内容の 1 の補数、GPR 10 の内容、および固定小数点例外レジスター・カーリー・ビットの値を加算し、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターに要約オーバーフロー、オーバーフロー、およびキャリー・ビットを設定します。

```
# Assume GPR 4 contains 0x8000 0000.
# Assume GPR 10 contains 0xEFFF FFFF.
# Assume the Carry bit is one.
subfeo 6,4,10
# GPR 6 now contains 0x6FFF FFFF.
```

4. 以下のコードは、GPR 4 の内容の 1 の補数、GPR 10 の内容、および固定小数点例外レジスター・ビットの値を加算し、結果を GPR 6 に保管し、演算の結果を反映するために固定小数点例外レジスターおよび条件レジスター・フィールド 0 に要約オーバーフロー、オーバーフロー、およびキャリー・ビットを設定します。

```
# Assume GPR 4 contains 0x8000 0000.
# Assume GPR 10 contains 0xEFFF FFFF.
# Assume the Carry bit is zero.
subfeo. 6,4,10
# GPR 6 now contains 0x6FFF FFFE.
```

関連概念

固定小数点プロセッサー

固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点演算命令

固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

特定または SFI (即値携帯からの減算) 命令

目的

16 ビットの符号付き整数から汎用レジスターの内容を減算して、その結果を別の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	08
6 - 10	RT
11 - 15	RA
16 - 31	SI

PowerPC (R)

特定 *RT*、 *RA*、 *SI*

POWER® ファミリー

SFI *RT*、 *RA*、 *SI*

説明

特定の 命令および **sfi** 命令は、汎用レジスター (GPR) *RA*、1、および 16 ビット符号付き整数 *SI* の内容の 1 の補数を追加します。 結果はターゲット GPR *RT* に入れられます。

> **注:** *SI* が -1 の場合、 **sub 検閲** 命令と **sfi** 命令は、GPR *RA* の内容の 1 の補数を GPR *RT* に入れます。

sub 満ちた 命令と **sfi** 命令には 1 つの構文形式があり、条件レジスター・フィールド 0 には影響しません。これらの命令は常に、固定小数点例外レジスターのカリー・ビットに影響します。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。
SI	演算のための 16 ビットの符号付き整数を指定します。

例

以下のコードは、符号付き整数 0x0000 7000 から GPR 4 の内容を減算し、その結果を GPR 6 に保管します。

```
# Assume GPR 4 holds 0x9000 3000.
```

```
subfic 6,4,0x00007000
# GPR 6 now holds 0x7000 4000.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

subfme または sfme (Minus One Extended から減算) 命令

目的

汎用レジスターの 1 の補数を桁上げ付き -1 に加算します。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	///
21	大江
22 - 30	232
31	rc

PowerPC (R)

subfme (subfme) *RT*、 *RA*

サブメ *RT*、 *RA*

subfmeo (subfmeo) *RT*、 *RA*

サブメオ *RT*、 *RA*

POWER® ファミリー

sfme *RT*、 *RA*

スフメ *RT*、 *RA*

sfmeo (sfmeo) *RT*、 *RA*

sfmeo。 *RT*、 *RA*

説明

subfme 命令および **sfme** 命令は、汎用レジスター (GPR) *RA*、固定小数点例外レジスターのキャリー・ビット、および x'FFFFFFFF' の内容の 1 の補数を追加し、その結果をターゲット GPR *RT* に入れます。

subfme 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

sfme 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	説明			
構文形式	オーバーフロー 例外 (OE)	固定小数点 例外レジスター	レコード ビット (RC)	条件 フィールド 0 の登録
subfme (subfme)	0	CA	0	なし
サブメ	0	CA	1	LT、GT、EQ、SO
subfmeo (subfmeo)	1	SO、OV、CA	0	なし
サブメオ	1	SO、OV、CA	1	LT、GT、EQ、SO
sfme	0	CA	0	なし
スフメ	0	CA	1	LT、GT、EQ、SO
sfmeo (sfmeo)	1	SO、OV、CA	0	なし
sfmeo。	1	SO、OV、CA	1	LT、GT、EQ、SO

subfme 命令の 4 つの構文形式、および **sfme** 命令の 4 つの構文形式は、常に固定小数点例外レジスターのカリー・ビット (CA) に影響します。 構文形式がオーバーフロー例外 (OE) ビットを 1 に設定する場合、命令は固定小数点例外レジスターの要約オーバーフロー (SO) ビットとオーバーフロー (OV) ビットに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 の内容の 1 の補数、固定小数点例外レジスターのカリー・ビット、および x 'FFFFFFFF' を追加し、結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume the Carry bit is set to one.
subfme 6,4
# GPR 6 now contains 0x6FFF CFFF.
```

2. 以下のコードは、GPR 4 の内容の 1 の補数、固定小数点例外レジスターのキャリー・ビット、および x 'FFFFFFFF' を追加し、結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume the Carry bit is set to zero.
subfme. 6,4
# GPR 6 now contains 0x4FFB CFFE.
```

3. 以下のコードは、GPR 4 の内容の 1 の補数、固定小数点例外レジスターのカリー・ビット、および x 'FFFFFFFF' を追加し、結果を GPR 6 に保管し、演算の結果を反映するように固定小数点例外レジスターを設定します。

```
# Assume GPR 4 contains 0xEFFF FFFF.
# Assume the Carry bit is set to one.
subfmeo 6,4
# GPR 6 now contains 0x1000 0000.
```

4. 以下のコードは、GPR 4 の内容の 1 の補数、固定小数点例外レジスターのカリー・ビット、および x 'FFFFFFFF' を追加し、結果を GPR 6 に保管し、演算の結果を反映するために条件レジスター・フィールド 0 と固定小数点例外レジスターを設定します。

```
# Assume GPR 4 contains 0xEFFF FFFF.
# Assume the Carry bit is set to zero.
subfmeo. 6,4
# GPR 6 now contains 0x0FFF FFFF.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

subfze または sfze (ゼロ拡張からの減算) 命令

目的

汎用レジスターの内容の 1 の補数、固定小数点例外レジスターのカリー・ビット、および 0 を加算し、その結果を 2 番目の汎用レジスターに入れます。

構文

ビット	VALUE
0 - 5	31
6 - 10	RT
11 - 15	RA
16 - 20	///
21	大江
22 - 30	200
31	rc

PowerPC (R)

subfze (サブ)	*RT*、 *RA*
Subfze。	*RT*、 *RA*
サブサイズ (subfzeo)	*RT*、 *RA*
subfzeo。	*RT*、 *RA*

POWER® ファミリー

sfze (sfze)	*RT*、 *RA*
sfze:	*RT*、 *RA*

POWER® ファミリー

sfzeo (sfzeo) *RT*、 *RA*

sfzeo: *RT*、 *RA*

説明

subfze 命令および **sfze** 命令は、汎用レジスター (GPR) *RA* の内容の 1 の補数、固定小数点例外レジスターのカリー・ビット、および x '00000000' を追加し、その結果をターゲット GPR *RT* に保管します。

subfze 命令には 4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

sfze 命令には、4 つの構文形式があります。 各構文形式は、条件レジスター・フィールド 0 および固定小数点例外レジスターに対して異なる影響を与えます。

項目	**説明**			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
subfze (サブ)	0	CA	0	なし
Subfze。	0	CA	1	LT、GT、EQ、SO
サブサイズ (subfzeo)	1	SO、OV、CA	0	なし
subfzeo。	1	SO、OV、CA	1	LT、GT、EQ、SO
sfze (sfze)	0	CA	0	なし
sfze:	0	CA	1	LT、GT、EQ、SO
sfzeo (sfzeo)	1	SO、OV、CA	0	なし
sfzeo:	1	SO、OV、CA	1	LT、GT、EQ、SO

subfze 命令の 4 つの構文形式、および **sfze** 命令の 4 つの構文形式は、常に固定小数点例外レジスターのカリー・ビット (CA) に影響します。 構文形式がオーバーフロー例外 (OE) ビットを 1 に設定する場合、命令は固定小数点例外レジスターの要約オーバーフロー (SO) ビットとオーバーフロー (OV) ビットに影響します。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	**説明**
RT	操作の結果が保管されるターゲット汎用レジスターを指定します。
RA	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4、カリー・ビット、およびゼロの内容の 1 の補数を加算し、結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume the Carry bit is set to one.
subfze 6,4
# GPR 6 now contains 0x6FFF D000.
```

2. 以下のコードは、GPR 4、Carry ビット、およびゼロの内容の 1 の補数を加算し、結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume the Carry bit is set to one.
subfze. 6,4
# GPR 6 now contains 0x4FFB D000.
```

3. 以下のコードは、GPR 4、カリー・ビット、およびゼロの内容の 1 の補数を加算し、結果を GPR 6 に保管し、演算の結果を反映するように固定小数点例外レジスターを設定します。

```
# Assume GPR 4 contains 0xEFFF FFFF.
# Assume the Carry bit is set to zero.
subfzeo 6,4
# GPR 6 now contains 0x1000 0000.
```

4. 以下のコードは、GPR 4、カリー・ビット、およびゼロの内容の 1 の補数を追加し、結果を GPR 6 に保管し、演算の結果を反映するために条件レジスター・フィールド 0 および固定小数点例外レジスターを設定します。

```
# Assume GPR 4 contains 0x70FB 6500.
# Assume the Carry bit is set to zero.
subfzeo 6,4
# GPR 6 now contains 0x8F04 9AFF.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

svc (監視プログラム呼び出し) 命令

目的

監視プログラム呼び出し割り込みを生成します。

注: svc 命令は、POWER® ファミリー・アーキテクチャーでのみサポートされます。

構文

ビット	VALUE
0 - 5	17
6 - 10	///
11 - 15	///
16 - 19	ライセンス情報 (LI)
20 - 26	LEV
27 - 29	FL2
30	SA
31	LK

POWER® ファミリー

SVC *LEV*、 *FL1*、 *FL2*

POWER® ファミリー

svcl (svcl) *LEV*、 *FL1*、 *FL2*

ビット	VALUE
0 - 5	17
6 - 10	///
11 - 15	///
16 - 29	SV
30	SA
31	LK

項目	説明
svca (svca)	「*SV*」
svcla (svcla)	「*SV*」

説明

svc 命令は、監視プログラム呼び出し割り込みを生成し、 **svc** 命令のビット 16 から 31 をカウント・レジスター (CR) のビット 0 から 15 に入れ、マシン状態レジスター (MSR) のビット 16 から 31 を CR のビット 16 から 31 に入れます。

svc 命令を使用する際には、以下の点を考慮してください。

- SVC 絶対ビット (SA) が 0 に設定されている場合、命令のフェッチと実行は、MSR の IP ビットの設定によって示される基底有効アドレス (EA) に対して、128 個のオフセット (b '1' | | LEV | |b '00000') のいずれかで続行されます。 *FL1* および *FL2* フィールドは、SVC ルーチンにデータを渡すために使用できますが、ハードウェアでは無視されます。
- SVC 絶対ビット (SA) が 1 に設定されている場合、命令のフェッチと実行は、MSR の IP ビットの設定によって示される基本 EA に対するオフセット x '1FE0' から続行されます。
- リンク・ビット (LK) が 1 に設定されている場合、 **svc** 命令に続く命令の EA がリンク・レジスターに入れられます。

注:

1. 操作が正しく行われるようにするには、 **svc** 命令の前に無条件ブランチまたは CR 命令がなければなりません。 有用な命令を指定どおりにスケジュールできない場合は、以下の構文の **cror** 命令の no-op バージョンを使用します。

```
cror BT,BA,BB      No-op when BT = BA = BB
```

2. **svc** 命令には、 **sc** (システム・コール) 命令と同じ命令コードがあります。

svc 命令には、4 つの構文形式があります。 各構文形式は MSR に影響します。

項目	説明		
構文形式	**リンク ビット (LK)**	**SVC 絶対ビット (SA)**	**マシン状態レジスター・ビット**
SVC	0	0	EE、PR、FE をゼロに設定
svcl (svcl)	1	0	EE、PR、FE をゼロに設定
svca (svca)	0	1	EE、PR、FE をゼロに設定
svcla (svcla)	1	1	EE、PR、FE をゼロに設定

svc 命令の 4 つの構文形式は、MSR の FP、ME、AL、IP、IR、または DR ビットには影響しません。 MSR の EE、PR、および FE ビットは常に 0 に設定されます。 固定小数点例外レジスターおよび条件レジスター・フィールド 0 は、 **svc** 命令の影響を受けません。

パラメーター

項目	説明
LEV	実行アドレスを指定します。
FL1	SVC ルーチンに渡すオプション・データのフィールドを指定します。
FL2	SVC ルーチンに渡すオプション・データのフィールドを指定します。
SV	SVC ルーチンに渡すオプション・データのフィールドを指定します。

関連概念

cror (条件レジスター OR) 命令

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

システム・コール命令
PowerPC ® システム・コール命令は、サービスを実行するための割り込みまたはシステムを生成します。

POWER ® ファミリーと PowerPC ® 命令の機能の違い
POWER ® ファミリーと PowerPC ® 命令は、POWER ® ファミリーと PowerPC ® プラットフォームで同じ命令コードを共有しますが、機能定義は異なります。

sync (同期化) または dcs (Data Cache 同期化) 命令

目的

PowerPC ® 命令 **sync** は、次の命令が開始される前に、前のすべての命令が完了したことを確認します。

POWER ® ファミリー命令 **dcs** を使用すると、プロセッサーはすべてのデータ・キャッシュ・ラインが書き込まれるまで待機します。

構文

ビット	値
0 - 5	31
6 - 9	///
10	L
11 - 15	///
16 - 20	///
21 - 30	598
31	/

PowerPC (R)

sync L

POWER ® ファミリー

dcs (dcs)

説明

PowerPC ® 命令 **sync** は、**sync** 命令の前に開始されたすべての命令が完了し、**sync** 命令が完了するまで後続の命令が開始されないようにする順序付け機能を提供します。 **sync** 命令が完了すると、 **sync** 命令の前に開始されたすべてのストレージ・アクセスが完了します。

L フィールドは、 ヘビー・ウェイト 同期 (L = 0) または ライトウェイト 同期 (L = 1) を指定するために使用されます。

注 : sync 命令は、完了するまでにかなりの時間がかかります。 eieio (Enforce In-order Execution of I/O) 命令は、入出力装置へのストレージ参照の順序を制御することだけが必要な場合により適しています。

POWER ® ファミリー命令 **dcs** を使用すると、メイン・メモリーへの書き込みまたは書き込みがスケジュールされているすべてのデータ・キャッシュ・ラインの書き込みが完了するまで、プロセッサーは待機します。

dcs および **sync** 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。 レコード (Rc) ビットが 1 に設定されている場合、命令形式は無効です。

パラメーター

項目	説明
L	負荷の大きい同期または軽量の同期を指定します。

例

以下のコードは、 **dcbf** 命令の結果がメイン・メモリーに書き込まれるまで、プロセッサーを待機させます。

```
# Assume that GPR 4 holds 0x0000 3000.
dcbf 1,4
sync
# Wait for memory to be updated.
```

関連概念
処理とストレージ
プロセッサーは、メイン・メモリーおよびレジスターにデータを保管します。

.ei 疑似命令

td (ダブルワードのトラップ) 命令

目的

特定の条件が真の場合にプログラム割り込みを生成します。

この命令は、64 ビット・アプリケーションを実行する 64 ビット PowerPC プロセッサーでのみ使用してください。

構文

ビット	VALUE
0 - 5	31
6 - 10	終了
11 - 15	A
16 - 20	B
21 - 30	68
31	0

PowerPC64

TD	*TO*、 *RA*、 *RB*

説明

汎用レジスター (GPR) *RA* の内容は、GPR *RB* の内容と比較されます。 TO フィールドのいずれかのビットが設定され、それに対応する条件が比較の結果によって満たされると、トラップ・タイプのプログラム割り込みが生成されます。

TO ビット条件は、次のように定義されます。

TO ビット	条件付き AND 演算
0	「Less Than」を比較します。
1	「Greater Than」を比較します。
2	等しいと比較します。
3	論理的により小さい値を比較します。
4	論理的により大きいものを比較します。

パラメーター

項目	説明
宛先	比較結果と AND 演算される *TO* ビットを指定します。
RA	比較のためのソース汎用レジスターを指定します。
rb	比較のためのソース汎用レジスターを指定します。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

例

以下のコードは、プログラム割り込みを生成します。

```
# Assume GPR 3 holds 0x0000_0000_0000_0001.
# Assume GPR 4 holds 0x0000_0000_0000_0000.
td 0x2,3,4    # A trap type Program Interrupt occurs.
```

関連概念

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

固定小数点トラップ命令
固定小数点トラップ命令は、指定された条件のセットについてテストします。

tdi (トラップ・ダブルワード即時) 命令

目的

特定の条件が真の場合にプログラム割り込みを生成します。

この命令は、64 ビット・アプリケーションを実行する 64 ビット PowerPC プロセッサーでのみ使用してください。

構文

ビット	VALUE
0 - 5	02
6 - 10	終了
11 - 15	A
16 - 31	SIMM

PowerPC64

tdi (tdi) *TO*、 *RA*、 *SIMM*

説明

汎用レジスター *RA* の内容は、SIMM フィールドの符号付き拡張値と比較されます。 *TO* フィールドのいずれかのビットが設定され、比較の結果として対応する条件が満たされると、システム・トラップ・ハンドラーが呼び出されます。

TO ビット条件は、次のように定義されます。

TO ビット	条件付き AND 演算
0	「Less Than」を比較します。
1	「Greater Than」を比較します。
2	等しいと比較します。
3	論理的により小さい値を比較します。
4	論理的により大きいものを比較します。

パラメーター

項目	説明
宛先	比較結果と AND 演算される *TO* ビットを指定します。
RA	比較のためのソース汎用レジスターを指定します。
SIMM	比較のために符号拡張される 16 ビット 2 の補数値。

インプリメンテーション

この命令は、64 ビット・インプリメンテーションの場合にのみ定義されます。 これを 32 ビット実装で使用すると、システムの正しくない命令エラー・ハンドラーが呼び出されます。

関連概念

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

固定小数点トラップ命令
固定小数点トラップ命令は、指定された条件のセットについてテストします。

tlbie または tlbi (Translation Look-Aside Buffer Invalidate Entry) 命令

目的

後続のアドレス変換に対して、変換ルックアサイド・バッファー項目を無効にします。

注 :

1. **tlbie** 命令は、PowerPC ® アーキテクチャーではオプションです。 PowerPC ® 601 RISC マイクロプロセッサー、PowerPC 603 RISC マイクロプロセッサー、および PowerPC 604 RISC マイクロプロセッサーでサポートされます。
2. **tlbi** は、POWER ® ファミリーの命令です。

構文

ビット	値
0 - 5	31
6 - 9	///
10	L
11 - 15	///
16 - 20	RB
21 - 30	306
31	/

PowerPC (R)

TL ビー RB、 L

POWER ® ファミリー

TLBI RA、 RB

説明

PowerPC ® 命令 **tlbie** は、変換ルック・アサイド・バッファー (TLB) で、有効アドレス (EA) に対応するエントリーを検索します。 検索は、マシン状態レジスター (MSR) 命令再配置ビットまたは MSR データ再配置ビットの設定に関係なく行われます。 検索では、最下位ビットを含む EA の一部が使用され、セグメント・レジスターの内容は無視されます。 検索基準を満たす項目は無効になるため、後続のストレージ・アクセスの変換には使用されません。

POWER ® ファミリー命令 **tlbi** は、MSR 命令再配置ビットまたは MSR データ再配置ビットの設定に関係なく、EA をその仮想アドレスに拡張し、その仮想アドレスの TLB 内のすべての情報を無効にします。 EA は、汎用レジスター (GPR) *RA* に入れられます。

POWER ® ファミリー命令 **tlbi** を使用する場合は、以下の点を考慮してください。

- GPR *RA* が 0 でない場合、EA は GPR *RA* と GPR *RB* の内容の合計です。 GPR *RA* が 0 の場合、EA は GPR *RB* と 0 の内容の合計です。
- GPR *RA* が 0 でない場合、EA は GPR *RA* に入れられます。
- EA が入出力アドレスを指定した場合、命令はノーオペレーションとして扱われますが、GPR *RA* が 0 でない場合、EA は GPR *RA* に置かれます。

L フィールドは、4 KB ページ・サイズ (L = 0) またはラージ・ページ・サイズ (L = 1) を指定するために使用されます。

tlbie および **tlbi** 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。 レコード・ビット (Rc) が 1 に設定されている場合、命令形式は無効です。

パラメーター

以下のパラメーターは、PowerPC ® 命令 **tlbie** にのみ関係します。

項目	説明
rb	検索用の EA を含むソース汎用レジスターを指定します。
L	ページ・サイズを指定します。

以下のパラメーターは、POWER® ファミリーの命令 **tlbi** にのみ関係します。

項目	説明
RA	EA 計算用のソース汎用レジスターを指定します。 *RA* が GPR 0 でない場合は、操作用のターゲット汎用レジスターを指定します。
rb	EA 計算用のソース汎用レジスターを指定します。

セキュリティー

tlbie および **tlbi** 命令には特権があります。

関連概念
処理とストレージ
プロセッサーは、メイン・メモリーおよびレジスターにデータを保管します。

tlbld (データ TLB 項目のロード) 命令

目的

PowerPC 603 RISC マイクロプロセッサー上のソフトウェアで実行される TLB 再ロード機能を支援するために、データ変換ルック・アサイド・バッファー (TLB) エントリーをロードします。

注 :

1. **tlbld** 命令は、PowerPC 603 RISC マイクロプロセッサーでのみサポートされます。 PowerPC® アーキテクチャーの一部ではなく、POWER® ファミリー・アーキテクチャーの一部でもありません。
2. TLB 再ロードは通常ハードウェアによって行われますが、PowerPC 603 RISC マイクロプロセッサーではソフトウェアによって行われます。
3. AIX が PowerPC 603 RISC マイクロプロセッサーを使用してシステムにインストールされている場合、TLB 再ロード機能を実行するためのソフトウェアがオペレーティング・システムの一部として提供されます。 この手順を使用する必要があるのは、AIX®なしで動作するように意図された PowerPC 603 RISC マイクロプロセッサー用のソフトウェアを作成する場合のみです。

構文

ビット	値
0 - 5	31
6 - 10	///
11 - 15	///
16 - 20	RB
21 - 30	978
31	/

項目	説明
PowerPC 603 RISC マイクロプロセッサー	PowerPC 603 RISC マイクロプロセッサー
TLBLD	*RB (RB)*

説明

理解を深めるために、以下の情報が示されています。

- **tlbld** 命令を呼び出す標準的な TLB 再ロード機能に関する情報。
- **tlbld** 命令の機能について説明します。

標準的な TLB 再ロード機能

アドレス変換の処理では、最初に有効アドレス (EA) が仮想アドレス (VA) に変換されます。 仮想アドレスの部分は、TLB エントリーを選択するために使用されます。 TLB 内にエントリーが見つからない場合は、ミスが検出されます。 ミスが検出されると、EA はデータ TLB ミス・アドレス (DMISS) レジスターにロードされます。 ターゲット・ページ・テーブル項目の最初のワードがデータ TLB ミス比較 (DCMP) レジスターにロードされます。 ルーチンは、DCMP の内容を、 HASH1 レジスターが指す 1 次ページ・テーブル項目グループ (PTEG) のすべての項目、および HASH2 レジスターが指す 2 次 PTEG のすべての項目と比較するために呼び出されます。 一致するものがあると、 **tlbld** 命令が呼び出されます。

tlbld 命令関数

tlbld 命令は、レジスター *RB* の内容によって選択されたデータ変換ルック・アサイド・バッファー (TLB) 項目を以下のようにロードします。

- データ TLB ミス比較 (DCMP) レジスターの内容は、データ TLB 項目の上位ワードにロードされます。
- RPA レジスターの内容とデータ TLB ミス・アドレス (DMISS) レジスターの内容がマージされ、データ TLB 項目の下位ワードにロードされます。

tlbld 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。 レコード・ビット (Rc) が 1 に設定されている場合、命令形式は無効です。

パラメーター

項目	説明
rb	EA 用のソース汎用レジスターを指定します。

セキュリティー

tlbld 命令には特権があります。

関連概念

tlbie または tlbi (Translation Look-Aside Buffer Invalidate Entry) 命令

tlbli (Load Instruction TLB Entry) 命令

目的

PowerPC 603 RISC Microprocessor 上のソフトウェアで実行される TLB 再ロード機能を支援するために、Translation Look-Aside Buffer (TLB) エントリーをロードします。

注 :

1. **tlbli** 命令は、PowerPC 603 RISC マイクロプロセッサーでのみサポートされます。 PowerPC アーキテクチャーの一部ではなく、POWER ファミリー・アーキテクチャーの一部でもありません。
2. TLB 再ロードは通常ハードウェアによって行われますが、PowerPC 603 RISC マイクロプロセッサーではソフトウェアによって行われます。

3. AIX が PowerPC 603 RISC マイクロプロセッサーを使用してシステムにインストールされている場合、TLB 再ロード機能を実行するためのソフトウェアがオペレーティング・システムの一部として提供されます。 この命令を使用する必要が生じる可能性があるのは、 AIX なしで動作するように意図された PowerPC 603 RISC マイクロプロセッサー用のソフトウェアを作成する場合のみです。

構文

ビット	値
0 - 5	31
6 - 10	///
11 - 15	///
16 - 20	RB
21 - 30	1010
31	/

項目	説明
PowerPC 603 RISC マイクロプロセッサー	PowerPC 603 RISC マイクロプロセッサー

TLB 李 (tlbli) *rb*

説明

理解を深めるために、以下の情報が示されています。

- **tlbli** 命令を呼び出す標準的な TLB 再ロード関数に関する情報。
- **tlbli** 命令の機能について説明します。

標準的な TLB 再ロード機能

アドレス変換の処理では、最初に有効アドレス (EA) が仮想アドレス (VA) に変換されます。 仮想アドレスの部分は、TLB エントリーを選択するために使用されます。 TLB 内にエントリーが見つからない場合は、ミスが検出されます。 ミスが検出されると、EA は命令 TLB ミス・アドレス (IMISS) レジスターにロードされます。 ターゲット・ページ・テーブル項目の最初のワードは、TLB Miss Compare (ICMP) 命令レジスターにロードされます。 ICMP の内容を、 HASH1 レジスターが指す 1 次ページ・テーブル項目グループ (PTEG) 内のすべての項目、および HASH2 レジスターが指す 2 次 PTEG 内のすべての項目と比較するために、ルーチンが呼び出されます。 一致するものがあると、 **tlbli** 命令が呼び出されます。

tlbli 命令関数

tlbli 命令は、レジスター *RB* の内容によって選択された命令 Translation Look-Aside Buffer (TLB) 項目を以下のようにロードします。

- 命令 TLB ミス比較 (DCMP) レジスターの内容は、命令 TLB 項目の上位ワードにロードされます。
- RPA レジスターおよび命令 TLB ミス・アドレス (IMISS) レジスターの内容がマージされ、命令 TLB 項目の下位ワードにロードされます。

tlbli 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。 レコード・ビット (Rc) が 1 に設定されている場合、命令形式は無効です。

パラメーター

項目	説明
rb	EA 用のソース汎用レジスターを指定します。

セキュリティー

tlbli 命令には特権があります。

tlbsync (Translation Look-Aside Buffer Synchronize) 命令

目的

1 つのプロセッサーによって実行される **tlbie** および **tlby** 命令が、他のすべてのプロセッサーで完了したことを確認します。

> **注: tlbsync** 命令は、PowerPC ® アーキテクチャーでのみ定義され、オプションの命令です。 PowerPC 603 RISC マイクロプロセッサーおよび PowerPC 604 RISC マイクロプロセッサーではサポートされますが、PowerPC ® 601 RISC マイクロプロセッサーではサポートされません。

構文

ビット	VALUE
0 - 5	31
6 - 10	///
11 - 15	///
16 - 20	///
21 - 30	566
31	/

PowerPC (R)

TLBSYNC

説明

tlbsync 命令は、 **tlbsync** 命令を実行するプロセッサーによって実行された以前のすべての **tlbie** および **tlby** 命令が、他のすべてのプロセッサーによって受信され、完了されるまで完了しません。

tlbsync 命令には 1 つの構文形式があり、固定小数点例外レジスターには影響しません。 レコード・ビット (Rc) が 1 に設定されている場合、命令形式は無効です。

セキュリティー

tlbsync 命令には特権があります。

関連概念

処理とストレージ
プロセッサーは、メイン・メモリーおよびレジスターにデータを保管します。

tw または t (ワードのトラップ) 命令

目的

指定された条件が真の場合にプログラム割り込みを生成します。

構文

ビット	VALUE
0-5	31
6 ～ 10	終了

ビット	VALUE
11-15	RA
16-20	RB
21-30	4
31	/

PowerPC (R)

TW *TO*、 *RA*、 *RB*

POWER® ファミリー

T *TO*、 *RA*、 *RB*

説明

tw 命令と **t** 命令は、汎用レジスター (GPR) *RA* の内容を GPR *RB* の内容と比較し、比較結果を *TO* と比較し、結果が 0 でない場合はトラップ・タイプのプログラム割り込みを生成します。

TO ビット条件は、次のように定義されます。

TO ビット	条件付き AND 演算
0	「Less Than」を比較します。
1	「Greater Than」を比較します。
2	等しいと比較します。
3	論理的により小さい値を比較します。
4	論理的により大きいものを比較します。

tw 命令と **t** 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
宛先	比較結果と AND 演算される *TO* ビットを指定します。
RA	比較のためのソース汎用レジスターを指定します。
rb	比較のためのソース汎用レジスターを指定します。

例

以下のコードは、プログラム割り込みを生成します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 7 contains 0x789A 789B.
tw 0x10,4,7
# A trap type Program Interrupt occurs.
```

関連概念

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

固定小数点トラップ命令

固定小数点トラップ命令は、指定された条件のセットについてテストします。

twi または ti (Trap Word Immediate) 命令

目的

指定された条件が真の場合にプログラム割り込みを生成します。

構文

ビット	VALUE
0-5	03
6 〜 10	終了
11-15	RA
16-31	SI

PowerPC (R)

twi (Twi) *TO*、 *RA*、 *SI*

POWER® ファミリー

TI *TO*、 *RA*、 *SI*

詳しくは、 固定小数点トラップ命令の拡張ニーモニック を参照してください。

説明

twi および **ti** 命令は、汎用レジスター (GPR) *RA* の内容を符号拡張 *SI* フィールドと比較し、比較結果を *TO* と比較し、結果が 0 でない場合はトラップ・タイプのプログラム割り込みを生成します。

TO ビット条件は、次のように定義されます。

TO ビット	条件付き AND 演算
0	「Less Than」を比較します。
1	「Greater Than」を比較します。
2	等しいと比較します。
3	論理的により小さい値を比較します。
4	論理的により大きいものを比較します。

twi および **ti** 命令には 1 つの構文形式があり、固定小数点例外レジスターまたは条件レジスター・フィールド 0 には影響しません。

パラメーター

項目	説明
宛先	比較結果と AND 演算される *TO* ビットを指定します。
RA	比較のためのソース汎用レジスターを指定します。
SI	比較のための符号拡張値を指定します。

例

以下のコードは、プログラム割り込みを生成します。

```
# Assume GPR 4 holds 0x0000 0010.
twi 0x4,4,0x10
# A trap type Program Interrupt occurs.
```

関連概念

ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

固定小数点トラップ命令
固定小数点トラップ命令は、指定された条件のセットについてテストします。

xor (XOR) 命令

目的

2 つの汎用レジスターの内容を XOR し、その結果を別の汎用レジスターに入れます。

構文

ビット	VALUE
0-5	31
6 ～ 10	RS
11-15	RA
16-20	RB
21-30	316
31	rc

項目	説明
xor	*RA*、*RS*、*RB*
xor:	*RA*、*RS*、*RB*

説明

xor 命令は、汎用レジスター (GPR) *RS* の内容を GPR *RB* の内容と XOR し、その結果を GPR *RA* に保管します。

xor 命令には、2 つの構文形式があります。 各シンタックス・フォームは、条件レジスター・フィールド 0 に対して異なる影響を与えます。

項目	説明			
構文形式	**オーバーフロー 例外 (OE)**	**固定小数点 例外レジスター**	**レコード ビット (RC)**	**条件 フィールド 0 の登録**
xor	なし	なし	0	なし
xor:	なし	なし	1	LT、GT、EQ、SO

xor 命令の 2 つの構文形式は、固定小数点例外レジスターには影響しません。 構文形式がレコード (Rc) ビットを 1 に設定する場合、命令は条件レジスター・フィールド 0 の「より小 (LT) ゼロ」、「より大 (GT) ゼロ」、「等しい (EQ) ゼロ」、および要約オーバーフロー (SO) ビットに影響します。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
rb	操作のソース汎用レジスターを指定します。

例

1. 以下のコードは、GPR 4 および GPR 7 の内容を XOR し、結果を GPR 6 に保管します。

```
# Assume GPR 4 contains 0x9000 3000.
# Assume GPR 7 contains 0x789A 789B.
xor 6,4,7
# GPR 6 now contains 0xE89A 489B.
```

2. 以下のコードは、GPR 4 および GPR 7 の内容を XOR し、結果を GPR 6 に保管し、演算の結果を反映するように条件レジスター・フィールド 0 を設定します。

```
# Assume GPR 4 contains 0xB004 3000.
# Assume GPR 7 contains 0x789A 789B.
xor. 6,4,7
# GPR 6 now contains 0xC89E 489B.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点論理命令
固定小数点論理命令は、論理演算をビット単位で実行します。

xori または xoril (XOR 即時) 命令

目的

16 ビットの符号なし整数を持つ汎用レジスターの下位 16 ビットを XOR し、その結果を別の汎用レジスターに入れます。

構文

ビット	VALUE
0-5	26
6 ～ 10	RS
11-15	RA
16-31	UI

PowerPC (R)

xori (xori) *RA*、 *RS*、 *UI*

POWER® ファミリー

xoril (xoril) *RA*、 *RS*、 *UI*

説明

xori および **xoril** は、汎用レジスター (GPR) *RS* の内容を、x '0000' と 16 ビット符号なし整数 *UI* と連結して XOR し、結果を GPR *RA* に保管します。

xori および **xoril** 命令の構文形式は 1 つだけであり、固定小数点例外レジスターまたは条件レジスターのフィールド 0 には影響しません。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
UI	演算用の 16 ビット符号なし整数を指定します。

例

以下のコードは、 0x0000 5730 の GPR 4 を XOR し、結果を GPR 6 に配置します。

```
# Assume GPR 4 contains 0x7B41 92C0.
xori 6,4,0x5730
# GPR 6 now contains 0x7B41 C5F0.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点論理命令
固定小数点論理命令は、論理演算をビット単位で実行します。

xoris または xoriu (XOR 即時シフト) 命令

目的

16 ビットの符号なし整数を持つ汎用レジスターの最上位の 16 ビットを XOR し、その結果を別の汎用レジスターに入れます。

構文

ビット	VALUE
0-5	27
6 ～ 10	RS
11-15	RA
16-31	UI

PowerPC (R)

xoris (xoris) *RA*、 *RS*、 *UI*

POWER ® ファミリー

xoriu (xoriu) *RA*、 *RS*、 *UI*

説明

xoris および **xoriu** は、16 ビット符号なし整数 *UI* および 0x'0000' を連結した汎用レジスター (GPR) *RS* の内容を XOR し、その結果を GPR *RA* に保管します。

xoris および **xoriu** 命令の構文形式は 1 つだけで、固定小数点例外レジスターまたは条件レジスターのフィールド 0 には影響しません。

パラメーター

項目	説明
RA	操作の結果が保管されるターゲット汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
UI	演算用の 16 ビット符号なし整数を指定します。

例

以下のコードは、 0x0079 0000 の GPR 4 を XOR し、結果を GPR 6 に保管します。

```
# Assume GPR 4 holds 0x9000 3000.
xoris 6,4,0x0079
# GPR 6 now holds 0x9079 3000.
```

関連概念

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点論理命令
固定小数点論理命令は、論理演算をビット単位で実行します。

疑似操作

疑似命令の参照情報には、アセンブラー疑似命令の概要が含まれています。

このトピックでは、アセンブラー疑似命令の概要と、すべての疑似命令の参照情報を示します。

疑似操作の概要

疑似命令は、アセンブラーに対する命令です。

疑似命令 (一般に 疑似命令と呼ばれる) は、マシン・コードを生成しないアセンブラーに対する命令です。アセンブラーは、実行時にのみ解決されるマシン・インストラクションとは異なり、アセンブリー中に疑似オペレーションを解決します。 疑似命令は、アセンブラー命令、アセンブラー演算子、またはアセンブラー・ディレクティブと呼ばれることがあります。

一般に、疑似命令は、データの位置合わせ、ブロックとセグメントの定義、および基底レジスターの割り当てに関するアセンブラー情報を提供します。 アセンブラーは、浮動小数点定数およびシンボリック・デバッガー情報 (**dbx**) に関するアセンブラー情報を提供する疑似命令もサポートします。

マシン・コードは生成されませんが、以下の疑似命令によって、アセンブラーのロケーション・カウンターの内容を変更することができます。

- **.align (位置合わせ)**
- **.byte**
- **.comm (.comm)**
- **.csect (.csect)**
- **.double**

- **.dsect (.dsect)**
- **.float (.float)**
- **.lcomm (.lcomm)**
- **.long (.long)**
- **組織 (.org)**
- **.short**
- **スペース**
- **ストリング**
- **.vbyte**

関数別にグループ化された疑似操作

疑似命令は機能別にグループ化されます。

疑似命令は、機能に応じて以下のグループに関連付けることができます。

データの位置合わせ

疑似命令は、データ位置合わせに使用されます。

以下の疑似命令は、プログラムのデータ・セクションまたはテキスト・セクションで使用されます。

- **.align (位置合わせ)**

データ定義

データ定義に対して異なる疑似命令が定義されています。

以下の疑似命令は、プログラムによって使用されるデータ域を作成するために使用されます。

- **.byte**
- **.double**
- **.float (.float)**
- **.leb128**
- **.long (.long)**
- **.ptr (.ptr)**
- **.quad (.quad)**
- **.short**
- **スペース**
- **ストリング**
- **.uleb128**
- **.vbyte**

ストレージ定義

データ域のレイアウト用に定義された疑似命令。

以下の疑似命令は、データ域のレイアウトを定義します。

- **.dsect (.dsect)**

アドレッシング

基底レジスターを登録するために定義された疑似命令。

以下の pseudo-ops は、基底レジスターとしてレジスターを割り当てたり、却下したりします。

- **.drop (.drop)**
- **. 使用**

関連概念
.drop pseudo-op
. 疑似命令の使用

アセンブラー・セクション定義
アセンブリー言語プログラムのセクションを定義するために使用される疑似命令。

以下の疑似命令は、アセンブリー言語プログラムのセクションを定義します。

- 「**.comm**」
- **.csect (.csect)**
- **.lcomm**
- **.tc**
- **.toc**

DWARF セクション定義
アセンブリー・プログラムの DWARF セクションを定義するために使用される疑似命令。

以下の疑似命令は、シンボリック・デバッガーが使用できるアセンブリー言語プログラムの DWARF セクションを定義します。

- **.dwsect (.dwsect)**

外部シンボル定義
グローバル変数を定義するために使用される疑似命令。

以下の疑似命令は、変数をグローバル変数または外部変数 (外部モジュールで定義された変数) として定義します。

- **.extern**
- **.globl**
- **.weak**

関連概念
.extern 疑似命令
.globl 疑似命令
.xline 疑似命令
固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。
ブランチ・プロセッサー
分岐プロセッサー命令には、分岐命令、条件レジスター・フィールド、および論理命令が含まれます。

静的シンボル定義
静的シンボルを定義するために使用される疑似命令。

以下の疑似命令は、静的シンボルを定義します。

- **.lglobl (.lglobl)**

呼び出し規則のサポート
デバッグ・トレースバックを定義するために使用される疑似命令。

以下の疑似命令は、プログラムのデバッグ時にトレースバックを実行するためのデバッグ・トレースバック・タグを定義します。

- **.tbtag (.tbtag)**

各種
各種機能を実行するために使用される疑似命令。

以下の疑似命令は、各種機能を実行します。

項目	説明
コメント	`.comment` セクションに任意の情報を追加します。
.except	`.except` セクションにエントリーを追加します。
ハッシュ	タイプ・チェック情報を提供します。
.info (.info)	`.info` セクションに任意の情報を追加し、C_INFO シンボルを生成します。
組織 (.org)	現在のロケーション・カウンターの値を設定します。
参照	再配置テーブルに特殊なタイプのエントリーを作成します。
.rename	正しくない名前または望ましくない名前の同義語または別名を作成します。
.set (.set)	シンボルに値とタイプを割り当てます。
ソース	ソース言語タイプを識別します。
.tocof (.tocof)	シンボルを別のモジュールの目次 (TOC) として定義します。
.x 行	ファイルと行番号の情報を提供します。

デバッガーのシンボル・テーブル・エントリー
シンボリック・デバッガーに追加情報を提供するために使用される疑似命令。

以下の疑似命令は、シンボリック・デバッガー (**dbx**) が必要とする追加情報を提供します。

- **.bb**
- **.bc**
- **.bf**
- **.bi**
- **.bs**
- **.eb**
- **.ec**
- **.ef**
- **.ei**
- 489 ページの『.ei 疑似命令』
- **.file**
- **.function**
- **行**

- 「**.stabx**」
- **.x 行**

ターゲット環境の表示
ターゲット環境を定義するために使用される疑似命令。

以下の疑似命令は、意図されたターゲット環境を定義します。

- **マシン**

表記規則

特に指定がない限り、空白文字は必須です。 オプションで、コンマの後にスペースを入れることができます。 空白文字は、1 つ以上の空白文字で構成できます。

疑似命令の中には、ラベルを使用しないものもあります。 ただし、 **.csect** 疑似命令を除き、マシン・インストラクション・ステートメントの場合と同様に、疑似命令ステートメントの前にラベルを置くことができます。

疑似命令を記述するために、以下の表記規則が使用されます。

項目	説明
名前	任意の有効なラベル。
クアルネーム	名前{StorageMappingClass} オプションの *Name* の後に、中括弧または大括弧で囲まれたオプションのストレージ・マッピング・クラスが続きます。
レジスター	汎用レジスター (general-purpose register)。 *Register* は、0 から 31 までの整数に評価される式です。
Number	評価結果が整数になる式。
式	特に明記されていない限り、 *Expression* 変数は再配置可能定数または絶対式を意味します。
浮動定数 *(FloatingConstant)*	浮動小数点定数。
StringConstant	ストリング定数。
[ ]	大括弧はオプションのオペランドを囲みます。ただし、479 ページの『.csect 疑似演算子』 と 518 ページの『.tc 疑似命令』は例外で、これらは構文上大括弧を必要とします。

>I. 疑似命令の位置合わせ

目的

Number パラメーターで指定された境界に達するまで、現在のロケーション・カウンターを進めます。

構文

```
.align Number [, Number2]
```

説明

.align 疑似命令は、通常、データが含まれている制御セクション (csect)、または接頭部付き命令が含まれている csect で使用されます。

Number パラメーターは、ロケーション・カウンターの必要な位置合わせのための 2 を底とする対数 (2 進対数) を指定します。 例えば、2 を指定した場合、 .align 疑似命令はワード位置合わせを要求し、3 を指定した場合、 .align 疑似命令はダブルワード位置合わせを要求します。

Number2 パラメーターを指定すると、位置合わせは、必要な位置合わせ境界の *Number2* バイト以内にある場合にのみ行われます。 これは、命令が 64 バイト境界を超えないようにするために、接頭部付き命令の前で最も役立ちます。

Number2 パラメーターを指定しないと、アセンブラーはバイトを挿入することによってロケーション・カウンターを位置合わせします。 ほとんどの場合、 0s が挿入されますが、現行 csect に PR または GL のストレージ・マッピング・クラスがある場合、ノーオペレーション命令が挿入されます。 ストレージ・マッピング・クラスについて詳しくは、 .csect 疑似命令を参照してください。

パラメーター

項目	説明
Number	評価結果が 0 から 12 までの整数値になる絶対式を指定します。 この値は、希望する位置合わせの対数底 2 を示します。 例えば、位置合わせ 8 (ダブルワード) は、整数値 3 で表されます。 4096 (1 ページ) の位置合わせは、整数値 12 で表されます。
Number 2	数値に評価される絶対式を指定します。 位置合わせは、ロケーション・カウンターが *Number2* バイトより望ましい位置合わせ境界に近い場合にのみ行われます。 *Number2* パラメーターによって指定された値が 2^{Number} より大きい場合、 *Number2* パラメーターは無視されます。

例

以下の例は、 **.align** 疑似命令の使用法を示しています。

1.
```
                .csect   progdata[RW]
                .byte    1
                         # Location counter now at odd number
                .align   1
                         # Location counter is now at the next
                         # halfword boundary.
                .byte    3,4
                .
                .
                .
                .align   2       # Insure that the label cont
                                 # and the .long pseudo-op are
                                 # aligned on a full word
                                 # boundary.
        cont:   .long    5004381
```

2.
```
                .csect [PR]
                …
                .align 6,8    # Align the program counter if the
                                 # current location counter is less
                                 # than 8 bytes from a 64-bit byte
                                 # boundary
                plw r3,0x10000(r4)    # Load a word into r3
```

.bb 疑似命令

目的

内部ブロックの先頭を識別し、内部ブロックの先頭に固有の情報を提供します。

構文

項目	説明
.bb	数値

説明

.bb 疑似命令は、シンボリック・デバッガーの使用時に必要なシンボル・テーブル情報を提供します。

.bb 疑似命令は、アセンブリーに他の影響を与えることはなく、コンパイラーによって通常挿入されます。

パラメーター

項目	説明
Number	内部ブロックが始まる元のソース・ファイルの行番号を指定します。

例

以下の例は、 **.bb** 疑似命令の使用法を示しています。

```
.bb   5
```

関連概念

疑似操作の概要

疑似命令は、アセンブラーに対する命令です。

.eb 疑似命令

.bc 疑似命令

目的

共通ブロックの先頭を識別し、共通ブロックの先頭に固有の情報を提供します。

構文

項目	説明
.bc	*StringConstant*

説明

.bc 疑似命令は、シンボリック・デバッガーを使用するときに必要なシンボル・テーブル情報を提供します。

.bc 疑似命令は、アセンブリーに他の影響を与えることはなく、コンパイラーによって通常挿入されます。

パラメーター

項目	説明
StringConstant (StringConstant)	元のソース・ファイルに定義されている共通ブロックのシンボル名を表します。

例

以下の例は、 **.bc** 疑似命令の使用法を示しています。

```
.bc   "commonblock"
```

関連概念
固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。

.bf 疑似命令

目的

関数の開始を識別し、関数の開始に固有の情報を提供します。

構文

項目	説明
.bf	数値

説明

.bf 疑似命令は、シンボリック・デバッガーの使用時に必要なシンボル・テーブル情報を提供します。

.bf 疑似命令は、アセンブリーに他の影響を与えることはなく、コンパイラーによって通常挿入されます。

注: .bf pseudo-op を使用する場合は、 **.function** pseudo-op を使用する必要があります。

パラメーター

項目	説明
Number	関数が開始される元のソース・ファイル内の絶対行番号を表します。

例

以下の例は、 **.bf** 疑似命令の使用法を示しています。

```
.bf   5
```

関連概念
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。
.ef 疑似命令
.function 疑似命令

.bi 疑似命令

目的

インクルード・ファイルの先頭を識別し、インクルード・ファイルの先頭に固有の情報を提供します。

構文

項目	説明
.BI	*StringConstant*

説明

.bi 疑似命令は、シンボリック・デバッガーの使用時に必要なシンボル・テーブル情報を提供します。

.bi 疑似命令は、アセンブリーに他の効果はなく、コンパイラーによって通常挿入されます。

.bi 疑似命令は、 **.line** 疑似命令と一緒に使用する必要があります。

パラメーター

項目	説明
StringConstant (StringConstant)	元のソース・ファイルの名前を表します。

関連概念
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.line 疑似命令
.ei 疑似命令

.bs 疑似命令

目的

静的ブロックの先頭を識別し、静的ブロックの先頭に固有の情報を提供します。

構文

項目	説明
.BS	*name*

説明

.bs 疑似命令は、シンボリック・デバッガーを使用するときに必要なシンボル・テーブル情報を提供します。

.bs 疑似命令は、アセンブリーに他の影響を与えることはなく、コンパイラーによって通常は挿入されます。

パラメーター

項目	説明
Name	元のソース・ファイルに定義されている静的ブロックのシンボル名を表します。

例

以下の例は、 **.bs** 疑似命令の使用法を示しています。

```
.lcomm cgdat, 0x2b4
.csect .text[PR]
.bs cgdat
.stabx   "ONE:1=Ci2,0,4;",0x254,133,0
.stabx   "TWO:S2=G5TWO1:3=Cc5,0,5;,0,40;;",0x258,133,8
.es
```

関連概念
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.comm 疑似命令
.lcomm 疑似命令

.byte 疑似命令

目的

Expression パラメーターによって表される指定された値を、連続するバイトにアセンブルします。

構文

.byte 式[**,***Expression*...]

説明

.byte 疑似命令は、式または式のストリングを連続したデータ・バイトに変更します。 ASCII 文字定数 (例えば、 **'X**) およびストリング定数 (例えば、 `Hello, world`) も、 **.byte** 疑似命令を使用してアセンブルできます。 各文字は、連続したバイトにアセンブルされます。 ただし、式に外部定義シンボルを含めることはできません。 また、1 バイトより長い式の値は、左側が切り捨てられます。

パラメーター

項目	説明
式	連続したバイトにアセンブルされる値を指定します。

例

以下の例は、 **.byte** 疑似命令の使用法を示しています。

```
        .set olddata,0xCC
        .csect data[rw]
mine:   .byte 0x3F,0x7+0xA,olddata,0xFF
```

```
# Load GPR 3 with the address of csect data[rw].
        .csect text[pr]
        l 3,mine(4)
```

```
# GPR 3 now holds 0x3F11 CCFF.
# Character constants can be represented in
# several ways:
```

```
        .csect data[rw]
        .byte "Hello, world"
        .byte 'H,'e,'l,'l,'o,',,' ,'w,'o,'r,'l,'d
```

```
# Both of the .byte statements will produce
# 0x4865 6C6C 6F2C 2077 6F72 6C64.
```

関連概念

疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.string 疑似命令
.vbyte 疑似命令
.llong 疑似命令

.comm 疑似命令

目的

共通ブロックと呼ばれるストレージの未初期化ブロックを定義します。

構文

項目	説明
.comm **(.comm)**	*Qualname* **,** *Expression*, [, *Number* [, *Visibility*]]

ここで、 *QualName* = *Name*[**[***StorageMappingClass***]**]

> **注:** 「名前」 は必須です。 *StorageMappingClass* はオプションで、指定する場合は大括弧で囲みます。 *StorageMappingClass* を省略した場合は、RW がデフォルトとして想定されます。 「可視性」 が指定されている場合、 *Number* は省略できます。

説明

.comm 疑似命令は、初期化されていないストレージ・ブロックである共通ブロックを定義します。 *QualName* パラメーターは、共通ブロックの名前を指定します。 *QualName* はグローバル・シンボルとして定義されます。

Expression パラメーターは、共通ブロックのサイズをバイト単位で指定します。

> **注:** TD *StorageMappingClass* を持つシンボルは、通常、ポインターより長くはありません。 (ポインターの長さは、32 ビット・モードでは 4 バイト、64 ビット・モードでは 8 バイトです。)

StorageMappingClass パラメーターの有効な値は、BS、RW、TD、UC、および UL です。 これらの値については、 **.csect** 疑似命令の説明に記載されています。 *StorageMappingClass* パラメーターに他の値が使用されている場合は、デフォルト値 RW が使用され、 **-w** フラグが使用されている場合は、警告メッセージが報告されます。

TD がストレージ・マッピング・クラスに使用される場合、ゼロのブロック (*Expression* パラメーターで指定された長さ) が、 **.data** セクション内の初期化された csect として TOC 域に書き込まれます。 RW、UC、または BS がストレージ・マッピング・クラスとして使用される場合、ブロックは現行モジュール内で初期化されず、シンボル・タイプ CM (共通) を持ちます。 ロード時に、RW、UC、または BC のストレージ・マッピング・クラスを持つ CM 制御セクションのスペースが、 **.data** セクションの末尾の **.bss** セクションに作成されます。

複数のモジュールが同じ共通ブロックを共用できます。 これらのモジュールのいずれかに同じ名前の外部制御セクション (csect) があり、同じ名前の csect に BS または UC 以外のストレージ・マッピング・クラスがある場合、共通ブロックは初期化され、他の制御セクションになります。 共通ブロックのストレージ・マッピング・クラスが TD である場合、csect は TOC 域に入ります。 これはバインド時に行われます。

リンク時に同じ *Qualname* を持つ初期化されていない共通ブロックが複数見つかった場合は、最大のブロック用にスペースが予約されます。

共通ブロックは、 *Number* パラメーターを使用して位置合わせすることができます。このパラメーターは、必要な位置合わせの対数基数 2 として指定されます。

共通ブロックの可視性は、 *Visibility* パラメーターを使用して指定できます。

パラメーター

項目	説明
クアルネーム	共通ブロックの名前とストレージ・マッピング・クラスを指定します。 パラメーターの *StorageMappingClass* 部分を省略すると、デフォルト値 RW が使用されます。 共通ブロックの有効な *StorageMappingClass* 値は、RW、TD、UC、BS、および UL です。
式	指定された共通ブロックの長さをバイト単位で示す絶対式を指定します。
Number	指定された共通ブロックのオプションの位置合わせを指定します。 これは、必要な位置合わせの対数底 2 として指定されます。 例えば、8 (ダブルワード) の位置合わせは 3 になり、2048 の位置合わせは 11 になります。 これは、 **.align** 疑似命令の引数に似ています。
可視性	シンボルの可視性を指定します。 有効な値は、 *exported*、 *protected*、 *hidden*、および *internal* です。 記号の可視性はリンカーによって使用されます。

例

1. 以下の例は、 **.comm** 疑似命令の使用法を示しています。

```
        .comm proc,5120
# proc is an uninitialized common block of
# storage 5120 bytes long which is
# global

# Assembler SourceFile A contains:
        .comm st,1024
# Assembler SourceFile B contains:

        .globl st[RW]
        .csect st[RW]
        .long 1
        .long 2

# Using st in the above two programs refers to
# Control Section st in Assembler SourceFile B.
```

2. この例は、2 つの異なるモジュールが同じデータにアクセスする方法を示しています。

   a. C モジュールのソース・コード td2.c:

```
/* This C module named td2.c   */
extern long   t_data;
extern  void mod_s();
main()
{
        t_data = 1234;
         mod_s();
         printf("t_data is %d\n", t_data);
}
```

   b. アセンブラー・モジュールのソース mod2.s:

```
        .file "mod2.s"
        .csect .mod_s[PR]
        .globl .mod_s[PR]
        .set    RTOC, 2
        l   5, t_data[TD](RTOC)  # Now GPR5 contains the
                                 # t_data value
        ai  5,5,14
        stu 5, t_data[TD](RTOC)
        br
        .toc
        .comm  t_data[TD],4  # t_data is a global symbol
```

   c. C およびアセンブラー・ソースから 実行可能ファイルを作成するための指示 td2 :

```
as -o mod2.o mod2.s
cc -o td2 td2.c mod2.o
```

   d. 実行中 td2 により、以下が出力されます。

```
t_data is 1248
```

3. 以下の例は、シンボルの可視性を指定する方法を示しています。

```
       .comm block, 1024, , exported
      # block is an uninitialized block of storage 1024 bytes
      # long. At link time, block is exported.
```

.comment 疑似命令

目的

出力ファイルのコメント・セクションに任意の情報を追加します。

構文

```
.comment StringConstant | Number
```

説明

.comment 疑似命令では、任意の情報を出力オブジェクト・ファイルに入れることができます。 この情報は、 .comment セクションに追加されます。

.comment 疑似命令は、カスケード・コンパイラーによって生成されます。 *StringConstant* または *Number* パラメーターは、アセンブラーによって解釈されず、 **ld** コマンドによって保持されません。

パラメーター

StringConstant (StringConstant)
出力ファイルの .comment セクションに追加する任意のストリングを指定します。

Number
出力ファイルの .comment セクションに追加する任意のバイト値を指定します。

.csect 疑似演算子

目的

コードまたはデータを csect (制御セクション) にグループ化し、その csect に名前、ストレージ・マッピング・クラス、および位置合わせを与えます。

構文

項目	説明
.csect (.csect)	*QualName*[, *Number*]

ここで、 *QualName* = [*Name*] [**[***StorageMappingClass***]**]

> **注:** *StorageMappingClass* を含む太字体の大括弧は構文の一部であり、オプション・パラメーターを指定 しません 。

説明

以下では、 **.csect** 疑似命令の使用について説明します。

- csect *QualName* パラメーターの形式は次のとおりです。

```
symbol[XX]
```

または

シンボル{*XX*}

ここで、[] (大括弧) または { } (中括弧) は、2 文字または 3 文字のストレージ・マッピング・クラス ID を囲みます。 どちらのタイプのブラケットも同じ結果になります。

QualName パラメーターは省略できます。 省略すると、csect に名前が付けられ、[PR] *StorageMappingClass* が使用されます。 *QualName* を使用する場合は、 *Name* パラメーターはオプションで、 *StorageMappingClass* は必須です。 *Name* を指定しない場合、csect には名前が付きません。

各 csect 疑似命令には、ストレージ・マッピング・クラスが関連付けられています。 ストレージ・マッピング・クラスは、csect 疑似命令がグループ化されるオブジェクト・セクションを決定します。 **.text** セクションには、通常、命令や定数などの読み取り専用データが含まれます。 **.data**、 **.bss**、 **.tdata**、および **.tbss** セクションには、読み取り/書き込みデータが含まれます。 **.bss** のストレージ・マッピング・

クラス。 および **.tbss** は、 **.csect** pseudo-op ではなく、 **.comm** および **.lcomm** pseudo-ops とともに使用する必要があります。

また、ストレージ・マッピング・クラスは、どの種類のデータを csect に含めるかを示します。 リストされているストレージ・マッピング・クラスの多くには、特定の実装と規則の詳細があります。 一般に、命令はストレージ・マッピング・クラス PR の CSECT 内に入れることができます。 変更可能データは、ストレージ・マッピング・クラス RW の CSECT 内に入れることができます。

csect 疑似命令は、以下のいずれかのストレージ・マッピング・クラスに関連付けられています。 ストレージ・マッピング・クラス ID には大/小文字の区別はありません。 ストレージ・マッピング・クラス ID は、 **.data**、 **.tbss**、 **.tdata**、 **.text**、および **.tss** オブジェクト・ファイル・セクションのグループにリストされます。

.text セクション・ストレージ-マッピング・クラス

Pr
プログラム・コード。 モジュールの実行可能命令を提供するセクションを識別します。

RO
読み取り専用データ。 実行中に変更されない定数を含むセクションを識別します。

db
デバッグ・テーブル。 読み取り専用データと同じ特性を持つセクションのクラスを識別します。

gl
グルー・コード プログラム・コードと同じ特性を持つセクションを識別します。 このタイプのセクションには、別のモジュール内のルーチンとのインターフェースとなるコードがあります。 インターフェース・コード要件の一部は、呼び出し全体で TOC アドレス可能度を維持することです。

Xo
拡張操作。 TOC に依存しないコードのセクションを識別します (TOC を介した参照はありません)。 これは、絶対アドレスへの分岐のターゲットになることができるように、メモリー内の固定アドレスに常駐することを目的としています。

> **注:** このストレージ・マッピング・クラスは、アセンブラー・ソース・プログラムでは使用しないでください。

SV
監視プログラム呼び出し。 監視プログラム呼び出しとして扱われるコードのセクションを識別します。

Tb
トレースバック・テーブル。 トレースバック・テーブルに関連したデータを含むセクションを識別します。

TI
トレースバック索引。 トレースバック索引に関連付けられたデータを含むセクションを識別します。

.data セクション・ストレージ-マッピング・クラス

TC0
TOC アンカーは、事前定義の TOC シンボルによってのみ使用されます。 特殊シンボル TOC を識別します。 TOC アンカーにのみ使用されます。

TC
TOC エントリー。 通常、他の csects またはグローバル・シンボルのアドレスを含む csect を示します。 1 つのアドレスのみが含まれている場合、csect の長さは通常 4 バイトです。

TD
TOC エントリー。 TOC から直接アクセスできるスカラー・データを含む csect を識別します。 頻繁に使用されるグローバル・シンボルの場合、これは TOC 内のアドレス・ポインター csect を介した間接アクセスに代わるものです。 規則により、TD セクションは 4 バイトを超えてはなりません。 プログラムの実行中に変更できる初期化されたデータが含まれます。

UA
不明なタイプです。 不明なストレージ・マッピング・クラスのデータを含むセクションを識別します。

rw
読み取り/書き込みデータ。 実行中に変更が必要であることが分かっているデータを含むセクションを識別します。

.data セクション・ストレージ-マッピング・クラス

Ds 記述子。 関数記述子を識別します。 この情報は、C や FORTRAN などの言語で関数ポインターを記述するために使用されます。

.bss セクション・ストレージ-マッピング・クラス

BS BSS クラス。 初期化されていない読み取り/書き込みデータを含むセクションを識別します。

UC (UC) 名前のない FORTRAN 共通。 読み取り/書き込みデータを含むセクションを識別します。

csect は、以下のいずれかのシンボル・タイプです。

ER 外部参照 (external reference)

SD CSECT セクション定義

LD エントリー・ポイント-ラベル定義

CM 共通 (BSS)

.tdata セクション・ストレージ-マッピング・クラス

TL スレッド・ローカル・ストレージを初期化しました。 プログラム内のスレッドごとに実行時にインスタンス化される csect を識別します。

.tbss セクション・ストレージ-マッピング・クラス

UL 初期化されていないスレッド・ローカル・ストレージ。 プログラム内のスレッドごとに実行時にインスタンス化される csect を識別します。

- 同じ *QualName* 値を持つすべての csects がグループ化され、同じ *QualName* を持つ **.csect** ステートメントでセクションを継続することができます。 異なる CSECT は、同じ名前と異なるストレージ・マッピング・クラスを持つことができます。 したがって、他の疑似命令または命令のオペランドとして csect 名を参照する場合は、ストレージ・マッピング・クラス ID を使用する必要があります。

  ただし、指定された名前の場合、外部化できる csect は 1 つだけです。 同じ名前を持つ複数の CSECT が外部化されると、実行エラーが発生する可能性があります。これは、リンケージ・エディターが csects を重複シンボル定義として扱い、そのうちの 1 つのみを選択して使用するためです。
- csect は本体として再配置されます。
- 名前 (*Name*) が指定されていない csects は、そのストレージ・マッピング・クラスによって識別され、各ストレージ・マッピング・クラスの名前なし csect が存在する可能性があります。 これらは、ストレージ・マッピング・クラスのみを持つ *QualName* で指定されます (例えば、 **.csect** [RW] の *QualName* は [RW] です)。
- 命令の前に **.csect** 疑似命令が指定されていない場合は、名前のないプログラム・コード ([PR]) csect が想定されます。
- **BS** または **UC** ストレージ・マッピング・クラスを持つ csect は、csect タイプ CM (共通) を持ちます。これはスペースを予約しますが、初期化されたデータはありません。 **.csect** 疑似命令で定義されている他のすべての csects は、タイプ SD (セクション定義) です。 **.comm** または **.lcomm** 疑似命令を使用して、タイプ CM の CSECT を定義することもできます。 タイプ CM の csect に外部ラベルを定義することはできません。
- **.csect** ステートメントにラベルを付けません。 **.csect** はその *QualName* によって参照することができ、ラベルは **.csect** の個々のエレメントに置くことができます。

パラメーター

項目	説明
Number	評価結果が 0 から 31 までの整数値になる絶対式を指定します。 この値は、必要な位置合わせの対数底 2 を示します。 例えば、8 (ダブルワード) の位置合わせは、整数値 3 で表されます。2048 の位置合わせは、整数値 11 で表されます。 これは、**.align** 疑似命令の *Number* パラメーターの使用法に似ています。 位置合わせは csect の先頭で行われます。 csect のエレメントは個別に位置合わせされません。 *Number* パラメーターはオプションです。 指定しない場合、デフォルト値は 2 です。
クアルネーム	csect の *Name* および *StorageMappingClass* を指定します。 *Name* が指定されていない場合、csect はその *StorageMappingClass* で識別されます。 *Name* も *StorageMappingClass* も指定されていない場合、csect は名前なしで、ストレージ・マッピング・クラス [PR] を持ちます。 *Name* を指定する場合は、 *StorageMappingClass* も指定する必要があります。

例

次の例では、3 つの csects を定義します。

```
# A csect of name proga with Program Code Storage-Mapping Class.
.csect proga[PR]
lh      30,0x64(5)
# A csect of name pdata_ with Read-Only Storage-Mapping Class.
.csect pdata_[RO]
```

```
l1:    .long  0x7782
l2:    .byte  'a,'b,'c,'d,'e
.csect  [RW],3          # An unnamed csect with Read/Write
```

```
                        # Storage-Mapping Class and doubleword
                        # alignment.
```

```
.float -5
```

.double 疑似命令

目的

倍精度浮動小数点定数をアセンブルします。

構文

項目	説明
.double	*FloatingConstant*[, *FloatingConstant*...]

説明

.double 疑似命令は、倍精度浮動小数点定数を連続するフルワードにアセンブルします。 現行セクションが DWARF セクションでない限り、フルワードの位置合わせが行われます。

パラメーター

項目	説明
浮動定数 *(FloatingConstant)*	アセンブルする浮動小数点定数を指定します。

例

以下の例は、 **.double** 疑似命令の使用法を示しています。

```
.double 3.4
.double -77
.double 134E12
.double 5e300
.double 0.45
```

関連概念

疑似操作の概要

疑似命令は、アセンブラーに対する命令です。

.float 疑似命令

.drop pseudo-op

目的

指定されたレジスターを基底レジスターとして使用することを停止します。

構文

項目	説明
.drop (.drop)	数値

説明

.drop pseudo-op は、*Number* パラメーターで指定されたレジスターを、操作における基底レジスターとしてプログラムが使用するのを停止します。 **.drop** 疑似命令は、基底アドレスを変更するときに **.using** 疑似命令の前に置く必要はなく、 **.drop** 疑似命令はプログラムの終わりに置く必要はありません。

パラメーター

項目	説明
Number	0 から 31 までの整数に評価される式を指定します。

例

以下の例は、 **.drop** 疑似命令の使用法を示しています。

```
.using _subrA,5
        # Register 5 can now be used for addressing
        # with displacements calculated
        # relative to _subrA.
```

```
        # .using does not load GPR 5 with the address
        # of _subrA. The program must contain the
        # appropriate code to ensure this at runtime.
```

```
   .
   .
   .
.drop 5
```

```
        # Stop using Register 5.
.using _subrB,5
        # Now the assembler calculates
        # displacements relative to _subrB
```

関連概念
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。
. 疑似命令の使用

.dsect 疑似命令

目的

ダミー制御セクションの開始または継続を識別します。

構文

項目	説明
.dsect **(.dsect)**	*name*

説明

.dsect 疑似命令は、ダミー制御セクションの開始または継続を識別します。 ダミー制御セクションで宣言された実際のデータは無視されます。ロケーション・カウンターだけが増分されます。 ダミー・セクション内のすべてのラベルは、ダミー・セクションの先頭からの相対オフセットと見なされます。 前の dsect と同じ名前を持つ dsect は、そのダミー制御セクションの継続です。

.dsect 疑似命令は、ストレージのブロックをマップするために使用できるデータ・テンプレートを宣言できます。 これを行うには、 **.using** 疑似命令を使用します。

パラメーター

項目	説明
Name	ダミー制御セクションを指定します。

例

1. 以下の例は、 **.dsect** 疑似命令の使用法を示しています。

```
        .dsect datal
d1:     .long 0

                          # 1 Fullwordd2:     .short 0,0,0,0,0,0,0,0,0       # 10 Halfwords
d3:     .byte 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0      # 15 bytes

        .align 3                    #Align to a double word.
d4:     .space 64                   #Space 64 bytes

.csect main[PR]
.using datal,7
l 5,d2

# This will actually load
# the contents of the
# effective address calculated
# by adding the offset d2 to
# that in GPR 7 into GPR 5.
```

2. 次の例には、暗黙ベース・アドレッシングでの **.dsect** および **.using** 疑似命令の使用法を示すいくつかのソース・プログラムが含まれています。

   a. ソース・プログラム `foo_pm.s`:

```
        .csect   foo_data[RW]
        .long   0xaa
```

```
        .short  10
        .short   20
        .globl   .foo_pm[PR]
        .csect   .foo_pm[PR]
        .extern  l1
        .using  TOC[TC0], 2
        l    7, T.foo_data
        b   l1
        br
        .toc
T.foo_data:   .tc  foo_data[TC], foo_data[RW]
```

b. ソース・プログラム bar_pm.s:

```
        .csect   bar_data[RW]
        .long   0xbb
        .short  30
        .short   40
        .globl   .bar_pm[PR]
        .csect   .bar_pm[PR]
        .extern  l1
        .using TOC[TC0], 2
        l    7, T.bar_data
        b  l1
        br
        .toc
T.bar_data:   .tc  bar_data[TC], bar_data[RW]
```

c. ソース・プログラム c1_s:

```
        .dsect   data1
d1:     .long   0
d2:     .short   0
d3:     .short   0
        .globl   .c1[PR]
        .csect   .c1[PR]
        .globl   l1
l1:     .using   data1, 7
        l   5, d1
        stu 5, t_data[TD](2)
        br                  # this br is necessary.
                            # without it, prog hangs
        .toc
        .comm  t_data[TD],4
```

d. メインプログラムのソース mm.c:

```
extern  long  t_data;
main()
{
     int  sw;
     sw = 2;
     if ( sw == 2 ) {
       foo_pm();
       printf ( "when sw is 2, t_data is 0x%x\n", t_data );
     }
     sw = 1;
     if ( sw == 1 ) {
       bar_pm();
       printf ( "when sw is 1, t_data is 0x%x\n", t_data );
     }
}
```

e. ソースから実行可能ファイルを作成するための指示:

```
as -o foo_pm.o foo_pm.s
as -o bar_pm.o bar_pm.s
as -o c1.o c1.s
cc -o mm mm.c foo_pm.o bar_pm.o c1.o
```

f. mm を実行すると、次のように出力されます。

```
when sw is 2, t_data is 0xaa
when sw is 1, t_data is 0xbb
```

関連概念
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.csect 疑似演算子
. 疑似命令の使用

.dwsect 疑似命令

目的

シンボリック・デバッガーが使用する DWARF デバッグ情報を保管するためのセクションを定義します。

構文

項目	説明
.dwsect **(.dwsect)**	*flag* [, *opt-label*]

説明

.dwsect 疑似命令は、シンボリック・デバッガーが使用するデータが入っているセクションの先頭または継続を識別します。 対応する DWARF シンボルも生成されます。 すべてのデータ定義疑似命令は .dwsect 疑似命令の後で使用できますが、DWARF セクション内で生成されるデータは位置合わせされません。 dwarf セクションは、同じフラグを持つ **.dwsect** ステートメントで継続することができます。

opt-label パラメーターを指定することにより、同じフラグ値に対して複数の DWARF シンボルを生成できます。 特定の *opt-label* および *flag* パラメーターのデータが収集され、長さが前に付加されます (長さフィールドがない **.dwabrev** セクションを除く)。 32 ビット・モードでは、長さフィールドの長さは 4 バイトで、64 ビット・モードでは 12 バイトです。 その後、同じフラグ値を持つすべてのセクションのデータが連結され、フラグによって識別される DWARF セクションに保管されます。

flag パラメーターには、以下のいずれかの値を指定できます。

- 特定の DWARF セクションにマップされる数値。
- XCOFF 形式で文書化されている短縮セクション名、または DWARF 仕様で文書化されているフルサイズのセクション名のストリング。

  注 : flag 値がストリングの場合、 **length** フィールドは生成されません。

DWARF セクションが指定されている場合、デバッガーのシンボル・テーブル・エントリーを指定するために使用される疑似命令は必要ありません。

パラメーター

項目	説明
フラグ	DWARF セクションのタイプを指定します。 有効な値は以下のとおりです。

項目	説明

```
0x10000, ".dwinfo", ".debug_info"
0x20000, ".dwline", ".debug_line"
0x30000, ".dwpbnms", ".debug_pubnames"
0x40000, ".dwpbtyp", ".debug_pubtypes"
0x50000, ".dwarnge", ".debug_aranges"
0x60000, ".dwabrev", ".debug_abbrev"
0x70000, ".dwstr", ".debug_str"
0x80000, ".dwrnges", ".debug_ranges"
0x90000, ".dwloc", ".debug_loc"
0xA0000, ".dwframe", ".debug_frame"
0xB0000, ".dwmac", ".debug_macinfo"
```

OPT-ラベル　固有のフラグ、OPT とラベルのペアごとに生成される DWARF シンボルを指定します。 *opt-label* ID は、同じフラグ値を持つ他の **.dwsect** ステートメントと突き合わせるためにのみ使用され、出力オブジェクト・ファイルには現れません。 *opt-label* パラメーターは、特定の DWARF シンボルを識別するオプションのラベルです。

例

1. **.dwinfo** セクションを定義します。

```
       .dwsect       0x00010000              # section name .dwinfo
.dwinfo:
       .short        0x0002                  # dwarf version
       .long         .dwabrev                # Reference to .dwabrev section
       .llong        0x04012f7473743133
       ...
```

2. **.dwabrev** セクションを定義します。

```
       .dwsect       0x00060000              # section name .dwabrev
.dwabrev:
       .llong       0x0111010308100611
       .llong       0x011201130b1b0825
```

3. 最初の例の最初の **.dwinfo** シンボルのデータを継続します。

```
       .dwsect 0x10000
       .llong       0x312f6469732f6432
```

4. 2 番目の **.dwinfo** シンボルを定義します。

```
       .dwsect 0x10000,dwinfo_2
       .short  2       # Version number
```

.eb 疑似命令

目的

内部ブロックの終わりを識別し、内部ブロックの終わりに固有の追加情報を提供します。

構文

項目	説明
.eb	数値

説明

.eb 疑似命令は、内部ブロックの終わりを識別し、シンボリック・デバッガーの使用時に必要なシンボル・テーブル情報を提供します。

.eb 疑似命令は、アセンブリーに他の影響を与えることはなく、コンパイラーによって通常挿入されます。

パラメーター

項目	説明
Number	内部ブロックが終了する元のソース・ファイルの行番号を指定します。

例

以下の例は、 **.eb** 疑似命令の使用法を示しています。

```
.eb   10
```

関連概念
固定小数点演算命令
固定小数点演算命令は、レジスターの内容を 32 ビット符号付き整数として扱います。
.bb 疑似命令

.ec 疑似命令

目的

共通ブロックの終わりを識別し、共通ブロックの終わりに固有の追加情報を提供します。

構文

.ec

説明

.ec 疑似命令は、共通ブロックの終わりを識別し、シンボリック・デバッガーを使用するときに必要なシンボル・テーブル情報を提供します。

.ec 疑似命令は、アセンブリーに他の影響を与えることはなく、コンパイラーによって通常挿入されます。

例

以下の例は、 **.ec** 疑似命令の使用法を示しています。

```
    .bc   "commonblock"
    .ec
```

関連概念
.bc 疑似命令
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.ef 疑似命令

目的

関数の終わりを識別し、関数の終わりに固有の追加情報を提供します。

構文

項目	説明
.ef	数値

説明

.ef 疑似命令は、関数の終わりを識別し、シンボリック・デバッガーを使用するときに必要なシンボル・テーブル情報を提供します。

.ef 疑似命令は、アセンブリーに他の効果はなく、コンパイラーによって通常挿入されます。

パラメーター

項目	説明
Number	関数が終了する元のソース・ファイルの行番号を指定します。

例

以下の例は、 **.ef** 疑似命令の使用法を示しています。

```
.ef   10
```

関連概念
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。
.bf 疑似命令

.ei 疑似命令

目的

インクルード・ファイルの終わりを識別し、インクルード・ファイルの終わりに固有の追加情報を提供します。

構文

.ei

説明

.ei 疑似命令は、インクルード・ファイルの終わりを識別し、シンボリック・デバッガーの使用時に必要なシンボル・テーブル情報を提供します。

.ei 疑似命令はアセンブリーに他の影響を与えず、コンパイラーによって通常挿入されます。

例

以下の例は、 **.ei** pseudo-op の使用法を示しています。

```
.ei   "file.s"
```

関連概念
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。
.bi 疑似命令

.es 疑似命令

目的

静的ブロックの終わりを識別し、静的ブロックの終わりに固有の追加情報を提供します。

構文

.es (.es)

説明

.es 疑似命令は、静的ブロックの終わりを識別し、シンボリック・デバッガーの使用時に必要なシンボル・テーブル情報を提供します。

.es 疑似命令は、アセンブリーに他の影響を与えることはなく、コンパイラーによって通常挿入されます。

例

以下の例は、 **.es** 疑似命令の使用法を示しています。

```
.lcomm cgdat, 0x2b4
.csect .text[PR]
.bs cgdat
```

```
.stabx "ONE:1=Ci2,0,4;",0x254,133,0
.stabx "TWO:S2=G5TWO1:3=Cc5,0,5;,0,40;;",0x258,133,8
.es
```

関連概念

疑似操作の概要

疑似命令は、アセンブラーに対する命令です。

.bs 疑似命令

.except 疑似命令

目的

出力ファイルの .except セクションにエントリーを追加し、指定されたシンボルからの情報への参照を追加します。

構文

```
.except Name, StringConstant | Number1, Number2
```

説明

.except セクションには、トラップ命令作成の理由に関する情報が含まれています。 .except 疑似命令は、ソース・プログラム内の対応するトラップ命令の前に使用する必要があります。 シンボルは複数の .except 疑似命令を持つことができ、これらの疑似命令は、そのシンボルに関連付けられた例外のグループに結合されます。

パラメーター

Name
.except エントリーに関連付けられるシンボルの名前です。 この名前は、 .globl または .lglobl ステートメントに指定する必要があります。

StringConstant (StringConstant)
言語の名前。 有効な言語名は、 .source 疑似命令ステートメントの資料にリストされています。

Number1
言語を示す式。 このパラメーターの有効な値については、/usr/include/storclass.h ヘッダー・ファイルを参照してください。

Number2
.except エントリーの理由です。 値を 0 にすることはできません。

.extern 疑似命令

目的

シンボルを、別のファイルで定義されている外部シンボルとして宣言します。

構文

項目	説明
.extern **(.extern)**	名前 [, *Visibility*]

説明

.extern 疑似命令は、 *Name* 値を別のソース・ファイルに定義されているシンボルとして識別し、 *Name* パラメーターは外部シンボルになります。 現在のアセンブリーで使用されているが定義されていない外部シンボルは、**.extern** ステートメントを使用して宣言する必要があります。 **.extern** ステートメントに現れるローカル定義のシンボルは、 **.globl** ステートメントでそのシンボルを使用するのと同等です。 **.globl** ステートメントでローカルに定義されていないシンボルは、**.extern** ステートメントでそのシンボルを使用するのと同等です。 未定義のシンボルは、 **as** コマンドの **-u** フラグが使用されていない限り、エラーとしてフラグが立てられます。

パラメーター

項目	説明
Name	外部シンボルとして宣言するシンボルの名前を指定します。 「名前」 には、 *Qualname* を指定できます。 *Qualname* パラメーターは、制御セクションの *Name* 値と *StorageMappingClass* 値を指定します。
可視性	シンボルの可視性を指定します。 有効な可視性の値は、 *exported*、 *protected*、 *hidden*、および *internal* です。 シンボル可視性はリンカーによって使用されます。

例

以下の例は、 **.extern** 疑似命令の使用法を示しています。

```
    .extern proga[PR]
    .toc
T.proga:    .tc proga[TC],proga[PR]
```

関連概念

疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.csect 疑似演算子
.globl 疑似命令
シンボルの可視性

.file 疑似命令

目的

ソース・ファイルのファイル名およびコンパイラー関連情報を識別します。

構文

```
    .file StringConstant
    .file StringConstant, StringConstant1
    .file StringConstant,[StringConstant1], StringConstant2
    .file StringConstant,[StringConstant1],[StringConstant2], StringConstant3
```

説明

.file 疑似命令は、シンボル・デバッガーおよび **ld** コマンドにシンボル・テーブル情報を提供します。 *StringConstant* はファイル名で、補助シンボル x_ftype == XTY_FN の名前として使用されます。 *StringConstant1*、 *StringConstant2*、および *StringConstant3* が指定された場合、これらの値は、 x_ftype 記号が XTY_CT に設定されたコンパイラー・タイム・スタンプ、x_ftype 記号が XTY_CV に設定されたコンパイラー・バージョン、および x_ftype 記号が XTY_CD に設定されたコンパイラー提供情報としてシンボル・テーブルに追加されます。 .file 疑似命令は、アセンブラー命令に対してこれ以外の変更を行いません。疑似命令は、カスケード・コンパイラーによってオブジェクト・コードに追加されます。

.file 疑似命令がソース・コードに指定されていない場合、アセンブラーは .file 疑似命令を最初のステートメントと見なし、プログラムを処理します。 アセンブラーは、ファイル名としてソース・プログラム名を持つエントリーをシンボル・テーブルに追加することによって、 .file 疑似命令を最初のステートメントとして作成します。 ソース・プログラムが標準入力の場合、ファイル名は noname です。 アセンブラー・リストには、この挿入されたエントリーはありません。

パラメーター

文字列定数
ファイル名を指定するストリング。

StringConstant1
コンパイラーのタイム・スタンプを指定するストリング。

StringConstant2
コンパイラーのバージョンを指定するストリング。

StringConstant3
任意のコンパイラー情報を指定するストリング。

例

1. ファイル名を指定してください。

```
.file “myfile.c”
```

2. 名前、バージョン、タイム・スタンプ、およびコンパイラー情報を指定します。

```
.file “myfile.c”, “Version 99”, “01 Jan 2099”, “no options”
```

3. ファイル名とタイム・スタンプを指定しますが、バージョンとコンパイラー情報は指定しません。

```
.file “myfile.c”, , “Jan 1, 2099”
```

.float 疑似命令

目的

浮動小数点定数をアセンブルします。

構文

項目	説明
.float **(.float)**	*FloatingConstant*[, *FloatingConstant*]

説明

.float は、浮動小数点定数を連続するフルワードにアセンブルします。 現行セクションが DWARF セクションでない限り、フルワードの位置合わせが行われます。

パラメーター

項目	説明
浮動定数 *(FloatingConstant)*	アセンブルする浮動小数点定数を指定します。

例

以下の例は、 **.float** 疑似命令の使用法を示しています。

```
.float 3.4
.float -77
.float 134E-12
```

.function 疑似命令

目的

機能を識別し、その機能に固有の追加情報を提供します。

構文

項目	説明
.function	*Name*、 *Expression1*、 *Expression2*、 *Expression3*、[*Expression4*]

説明

.function 疑似命令は、関数を識別し、シンボリック・デバッガーの使用に必要なシンボル・テーブル情報を提供します。

.function 疑似命令は、アセンブリーに他の影響を与えることはなく、コンパイラーによって通常挿入されます。

パラメーター

項目	説明
Name	関数 *Name* を表し、現行アセンブリー内でシンボルまたは制御セクション (csect) *Qualname* として定義する必要があります。 (*Qualname* は、制御セクションの *Name* および *StorageMappingClass* を指定します。)
Expression1	関数の最上部を表します。
Expression2	関数のストレージ・マッピング・クラスを表します。
Expression3	関数のタイプを表します。

.function 疑似命令に対する 3 番目と 4 番目のパラメーターは、プレースホルダーとしてのみ機能します。これらのパラメーターは、以前のシステム (RT、System V) との下位互換性のために保持されています。

項目	説明
Expression4	関数のサイズ (バイト単位) を表します。 このパラメーターは絶対式でなければなりません。 このパラメーターはオプションです。 **注:** *Expression4* パラメーターを省略した場合、関数サイズは、関数が属する csect のサイズに設定されます。 csect に 1 つの関数が含まれ、csect の開始と終了が関数の開始と終了と同じである場合にのみ、csect サイズは関数サイズと等しくなります。

例

以下の例は、 **.function** 疑似命令の使用法を示しています。

```
.globl   .hello[pr]
.csect   .hello[pr]
.function   .hello[pr],L.1B,16,044,0x86
L.1B:
```

.globl 疑似命令

目的

シンボルをグローバル・シンボルとして宣言します。

構文

項目	説明
.globl (.globl)	名前 [, *Visibility*]

説明

.globl 疑似命令は、シンボル名がグローバル・シンボルであり、リンク中に他のファイルによって参照できることを示します。 **.extern**、 **.weak**、または **.comm** 疑似命令を使用して、グローバル・シンボルを作成することもできます。

グローバル・シンボルの可視性は、 *Visibility* パラメーターで指定できます。

パラメーター

項目	説明
Name	グローバルとして宣言するラベルまたはシンボルの名前を指定します。「名前」には、*Qualname* を指定できます。 (*Qualname* は、制御セクションの *Name* および *StorageMappingClass* を指定します。)
可視性	シンボルの可視性を指定します。 有効な可視性の値は、 *exported*、 *hidden*、 *internal*、および *protected* です。 シンボル可視性はリンカーによって使用されます。

例

次の例は、 **.globl** 疑似命令の使用法を示しています。

```
   .globl main
main:
   .csect data[rw]
   .globl data[rw], protected

# data[RW] is a global symbol and is to be exported with
# protected visibility at link time.
```

.hash pseudo-op

目的

ハッシュ値を外部シンボルに関連付けます。

構文

項目	説明
ハッシュ	*Name*、 *StringConstant*

説明

ハッシュ・ストリング値には、タイプ検査情報が含まれます。 これは、プログラムの実行前に変数の不一致および引数インターフェース・エラーを検出するために、リンケージ・エディターおよびプログラム・ローダーによって使用されます。

ハッシュ・ストリング値は通常、厳密に型付けされた言語のコンパイラーによって生成されます。 シンボルのハッシュ値は、アセンブリー内で一度しか設定できません。 タイプ・エンコードおよび検査について詳しくは、XCOFF オブジェクト (a.out) ファイル・フォーマットの タイプ・チェック・セクション を参照してください。

パラメーター

項目	説明
Name	シンボルを表します。 これは外部シンボルでなければならないため、 *Name* は **.extern** または **.globl** ステートメントで指定する必要があります。
StringConstant (StringConstant)	タイプ・チェック・ハッシュ・ストリング値を表します。 このパラメーターは、16 進ハッシュ・コードを表す文字で構成され、[0-9A-F] または [0-9a-f] のセット内になければなりません。 ハッシュ・ストリングは、以下の 3 つのフィールドで構成されます。 • 言語 ID は、各言語を表す 2 バイトのフィールドです。 最初のバイトは 0x00 です。 2 番目のバイトには、 **.source** 疑似命令にリストされているものと同じ事前定義言語コードが含まれています。 • 汎用ハッシュは、データ・シンボルまたは関数を記述できる最も一般的な形式を表す 4 バイト・フィールドです。 これは、 AIX®でサポートされる言語の中で最も一般的な分母です。 このフィールドにはユニバーサル・ハッシュを使用できます。 • Language Hash は、データ記号または関数の、より詳細な言語指定表現を含む 4 バイト・フィールドです。 **注:** ハッシュ・ストリングの長さは 10 バイトでなければなりません。 それ以外の場合は、**-w** フラグを使用すると警告メッセージが報告されます。 各文字は 2 つの ASCII コードで表されるため、10 バイトのハッシュ文字ストリングは 20 桁の 16 進数字のストリングで表されます。

例

以下の例は、 **.hash** 疑似命令の使用法を示しています。

```
.extern b[pr]
.extern a[pr]
.extern e[pr]
.hash b[pr],"0000A9375C1F51C2DCF0"
.hash a[pr],"ff0a2cc12365de30"  # warning may report
.hash e[pr],"00002020202051C2DCF0"
```

.info 疑似命令

目的

出力ファイルの .info セクションに任意の情報を追加し、指定された名前で C_INFO シンボルを生成します。

構文

```
.info Name, Number0 [, Number] …
.info , Number [, Number]...
```

説明

.info 疑似命令は、出力オブジェクト・ファイルの .info セクションに情報を追加します。 **Number0** 値およびその他の **Number** 値は、 .info セクションに単語値として追加されます。 **Name** パラメーターを指定すると、指定した名前の C_INFO シンボルが生成されます。このシンボルの値は、 .info セクションにある **Number0** の後のワードのオフセットです。

Number0 値は、 **C_INFO** 記号を持つ関連データの長さを示します。 **Name** および **Number0** を省略すると、情報を複数のソース・ファイル行に分割することができます。 この場合、**Number0** は、特定の **C_INFO** シンボルに関連付けられたすべての情報を考慮します。 埋め込みメタデータは、**Name** および **Number0** を省略して生成することもできます。 組み込みメタデータについて詳しくは、 XCOFF オブジェクト・ファイル・フォーマット を参照してください。 .info 疑似命令は、カスケード・コンパイラーによって使用されます。 アセンブラーおよび **ld** コマンドは、通常、セクションの内容を解釈しません。

パラメーター

Name
出力ファイルの .comment セクションに追加する任意のストリングを指定します。

Number0
出力ファイルの .info セクションに追加する情報の総計の長さをバイト単位で指定します。

Number
出力ファイルの .info セクションに追加する任意のワードを指定します。

.lcomm 疑似命令

目的

ストレージのローカル未初期化ブロックを定義します。

構文

項目	説明
.lcomm (.lcomm)	*Name1*, *Expression1*[, *Sectname* [, *Expression2*]]

説明

.lcomm psuedo-op は、ローカルの未初期化ストレージ・ブロックを定義します。 *Sectname* パラメーターが QualName オペランドで、 **StorageMappingClass** クラスが UL の場合、ストレージはスレッド・ローカル変数用です。

.lcomm 疑似命令は、おそらく他のソース・ファイルではアクセスされないデータに使用されます。

Name1 パラメーターは、ストレージ・ブロックの先頭にあるラベルです。 このストレージ・ブロックのロケーション・カウンターは、*Expression1* パラメーターの値だけ増分されます。 *Sectname* パラメーターを使用して、特定のストレージ・ブロックを指定できます。 *Sectname* パラメーターが、**StorageMappingClass** クラスが UL の QualName オペランドである場合、ストレージ・ブロックは **.tbss** セクションに割り当てられます。 それ以外の場合、ストレージ・ブロックは **.bss** セクションに割り当てられます。 *Sectname* が指定されていない場合は、名前なしストレージ・ブロックが使用されます。

ストレージ・ブロックの位置合わせは、*Expression2* パラメーターで指定できます。 *Expression2* を省略すると、ストレージのブロックはハーフワード境界に位置合わせされます。

パラメーター

項目	説明
Name1	ストレージのブロック上のラベル。 *Name1* は、 **.globl** ステートメントのオペランドでない限り、シンボル・テーブルに現れません。
Expression1	ストレージ・ブロックの長さを指定する絶対式。

項目	説明
セキュリティー名	オプションの **csect** 名。 *Sectname* を省略すると、ストレージ・マッピング・クラスが BS の名前なしストレージ・ブロックが使用されます。 *Sectname* が QualName の場合、**StorageMappingClass** は BS または UL にすることができます。 *Sectname* がシンボル名の場合は、デフォルトのストレージ・マッピング・クラス BS が使用されます。 同じ *Sectname* を複数の **.lcomm** ステートメントで使用できます。 **.lcomm** ステートメントによって指定されたストレージのブロックは、単一の **csect** に結合されます。
Expression2	必要な位置合わせの対数底 2 を指定する絶対式。 *Expression2* を省略すると、値 2 が使用され、ハーフワードに位置合わせされます。

例

1. 5KB のストレージをセットアップし、それを `buffer` として参照するには、以下のようにします。

```
.lcomm buffer,5120
        # Can refer to this 5K
        # of storage as "buffer".
```

2. `proga` という名前のラベルをセットアップするには、以下のようにします。

```
.lcomm b3,4,proga
        # b3 will be a label in a csect of class BS
        # and type CM with name "proga".
```

3. スレッド・ローカル・ストレージのローカル・ブロックを定義するには、以下のようにします。

```
.lcomm tls1,32,tls_static[UL],3
        # tls1 is a label on a block of thread-local storage 32 bytes
        # long aligned on a doubleword boundary. The name of the block of
        # storage is tls_static[UL].
```

関連概念

疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.comm 疑似命令

.leb128 疑似命令

目的

可変長の式をアセンブルします。

構文

項目	説明
.leb128	*Expression*[, *Expression*, ...]

説明

.leb128 疑似命令は、符号付きリトル・エンディアン・ベース 128 (LEB128) フォーマットを使用して、絶対式を連続バイトにアセンブルします。 各式は、符号付き 64 ビット式として扱われ、 LEB128 形式に変換されてから、値に必要な数のバイトにアセンブルされます。

.leb128 疑似命令では、位置合わせは行われません。

パラメーター

項目	説明
式	絶対式。

例

以下の例は、 .leb128 疑似命令の使用法を示しています。

```
       .leb128             128         #Emits 0x80, 0x01
       .leb128             -129         #Emits 0xFF, 0x7E
       .leb128             127         #Emits 0xFF, 0x00
```

関連概念

疑似操作の概要

疑似命令は、アセンブラーに対する命令です。

.uleb128 疑似命令

.lglobl 疑似命令

目的

シンボル・テーブル内の静的名の情報を保持する手段を提供します。

構文

項目	説明
.lglobl (.lglobl)	*name*

説明

静的ラベルまたは静的関数名を参照できるように、制御セクション (csect) 内で定義された静的ラベルまたは静的関数名をシンボル・テーブルに保持する必要があります。 このシンボルは「非表示外部」のクラスを持ち、グローバル・シンボルとは異なります。 **.lglobl** 疑似命令は、 *Name* パラメーターで指定されたシンボルに LD のシンボル・タイプと C_HIDEXT のクラスを与えます。

> **注: .lglobl** pseudo-op は、どの csect 名にも適用する必要はありません。 アセンブラーは、csect 名に明示的な **.globl** 疑似命令が適用されていない限り、C_HIDEXT のクラスを持つ csect 名のシンボル・テーブル・エントリーを自動的に生成します。 明示的な **.globl** 疑似命令が csect 名に適用される場合、csect のシンボル・テーブル・エントリー・クラスは C_EXT です。

パラメーター

項目	説明
Name	シンボル・テーブルに保持する必要がある静的ラベルまたは静的関数名を指定します。

例

以下の例は、 **.lglobl** pseudo-op の使用法を示しています。

```
        .toc
        .file    "test.s"
        .lglobl  .foo
        .csect   foo[DS]
foo:
        .long    .foo,TOC[tc0],0
        .csect   [PR]
.foo:
        .stabx   "foo:F-1",.foo,142,0
        .function        .foo,.foo,16,044,L..end_foo-.foo
```

```
        .
        .
        .
```

>

関連概念
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.function 疑似命令
.globl 疑似命令

.line 疑似命令

目的

行番号を識別し、その行番号に固有の追加情報を提供します。

構文

項目	説明
行	数値

説明

.line 疑似命令は、行番号を識別し、 **.bi** 疑似命令と一緒に使用して、シンボル・デバッガーの使用に必要なシンボル・テーブルおよびその他の情報を提供します。

この疑似命令は、コンパイラーによって通常挿入され、アセンブリーにはその他の影響を与えません。

パラメーター

項目	説明
Number	元のソース・ファイルの行番号を表します。

例

以下の例は、 **.line** 疑似命令の使用法を示しています。

```
.globl .hello[pr]
.csect .hello[pr]
.align 1
.function   .hello[pr],L.1B,16,044
```

```
.stabx "hello:f-1",0,142,0
.bf 2
.line 1
.line 2
```

関連概念
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.bi 疑似命令
.bf 疑似命令
.function 疑似命令

.long 疑似命令

目的

式を連続するフルワードにアセンブルします。

構文

項目	説明
.long (.long)	*Expression*[,*Expression, ...*]

説明

.long 疑似命令は、式を連続するフルワードにアセンブルします。 現行セクションが DWARF セクションでない限り、フルワードの位置合わせが行われます。

パラメーター

項目	説明
式	フルワードにアセンブルされる式を表します。

例

以下の例は、 **.long** 疑似命令の使用法を示しています。

```
.long 24,3,fooble-333,0
```

関連概念
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。
.byte 疑似命令
.short 疑似命令
.vbyte 疑似命令

.llong 疑似命令

目的

式を連続するダブルワードにアセンブルします。

構文

項目	説明
.llong (.llong)	*Expression*[,*Expression, ...*]

説明

.llong 疑似命令は、式を連続するダブルワードにアセンブルします。 ダブルワードの位置合わせは、現行セクションが DWARF セクションでない限り行われます。

パラメーター

項目	説明
式	フルワード/ダブルワードにアセンブルされる式を表します。

例

以下の例は、 **.llong** 疑似命令の使用法を示しています。

```
.extern  fooble
.llong   24,3,fooble-333,0
```

これは、4 ダブルワード (32 バイト) のストレージを埋めます。

関連概念
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.short 疑似命令
.vbyte 疑似命令
.llong 疑似命令

.machine 疑似命令

目的

対象となるターゲット環境を定義します。

構文

項目	説明
.machine	*StringConstant*

説明

.machine 疑似命令は、ターゲット・マシンの正しい命令ニーモニック・セットを選択します。 リンケージ・エディターの使用に必要なシンボル・テーブル情報を提供します。 **.machine** 疑似命令は、 **as** コマンドの **-m** フラグの設定をオーバーライドします。このフラグは、ターゲット・マシンの命令ニーモニック・セットを指定するためにも使用できます。

.machine 疑似命令は、ソース・プログラムで複数回使用することができます。 **.machine** 疑似命令によって指定された値は、以前の **.machine** 疑似命令によって指定された値をオーバーライドします。 最初の **.machine** 疑似命令をソース・プログラムの先頭に置く必要はありません。 ソース・プログラムの先頭で **.machine** 疑似命令が発生せず、 **as** コマンドで **-m** フラグが使用されていない場合は、デフォルトのアセンブリー・モードが使用されます。 デフォルトのアセンブリー・モードは、最初の **.machine** 疑似命令によってオーバーライドされます。

.machine 疑似命令が無効な値を指定すると、エラーが報告されます。 その結果、アセンブラーの残りの命令妥当性検査では、デフォルト・モード値、 **-m** フラグ、または直前の **.machine** 疑似命令によって指定された最後の有効な値が使用されます。

パラメーター

項目	説明
StringConstant (StringConstant)	アセンブリー・モードを指定します。 このパラメーターには大/小文字の区別はありません。このパラメーターには、コマンド行で **-m** フラグを使用して指定できる任意の値を指定できます。 引用符で囲まれた有効な値は、以下のとおりです。 **ヌル・ストリング ("") またはなし** デフォルトのアセンブリー・モードを指定します。 ソース・プログラムには、POWER ファミリーと PowerPC の両方に共通の命令のみを含めることができます。 その他の命令はすべてエラーとなります。 **push** 現在のアセンブリー・モードをアセンブリー・モードのプッシュダウン・スタックに保存します。 **ポップ (pop)** 以前に保存された値をアセンブリー・モードのプッシュダウン・スタックから削除し、アセンブリー・モードをこの保存された値に復元します。 **注 : push** および **pop** の使用目的は、現在のアセンブリー・モードを変更するインクルード・ファイル内にあります。 `.machine "push"` は、現在のアセンブリー・モードを別の **.machine** に変更する前に、組み込みファイルで使用する必要があります。 同様に、入力アセンブリー・モードを復元するには、組み込みファイルの末尾に `.machine "pop"` を使用する必要があります。 アセンブリー・モードのプッシュダウン・スタックで 100 を超える値を保持しようとすると、アセンブリー・エラーが発生します。 pseudo-ops `.machine "push"` と `.machine "pop"` はペアで使用されます。 **ppc** PowerPC 共通アーキテクチャー、32 ビット・モードを指定します。 ソース・プログラムに含めることができるのは、 PowerPC® 共通アーキテクチャー、32 ビット命令のみです。 その他の命令はすべてエラーとなります。 **ppc64** PowerPC 64 ビット・モードを指定します。 ソース・プログラムには、PowerPC 64 ビット命令のみを含めることができます。 その他の命令はすべてエラーとなります。 **com** POWER ファミリーと PowerPC アーキテクチャー交差モードを指定します。 ソース・プログラムには、 POWER ファミリーと PowerPC の両方に共通の命令のみを含めることができます。 その他の命令はすべてエラーとなります。 **pwr** POWER ファミリーのアーキテクチャー、 POWER ファミリーの実装モードを指定します。 ソース・プログラムには、 POWER ファミリー・アーキテクチャーの POWER ファミリー実装用の命令のみを含めることができます。 その他の命令はすべてエラーとなります。 **pwr2 または pwrx** POWER ファミリー・アーキテクチャー、 POWER2™ 実装。 ソース・プログラムには、 POWER® ファミリー・アーキテクチャーの POWER2™ 実装用の命令のみを含めることができます。 その他の命令はすべてエラーとなります。 **pwr4 または 620** POWER4 モードを指定します。 ソース・プログラムには、 POWER4 プロセッサーと互換性のある命令のみを含めることができます。 **pwr5** AIX 5.3 以降、 POWER ファミリー・アーキテクチャーの場合、POWER5™ 実装。 ソース・プログラムには、 POWER® ファミリー・アーキテクチャーの POWER5™ 実装用の命令のみを含めることができます。 その他の命令はすべてエラーとなります。 **pwr5x** POWER5+™ モードを指定します。 ソース・プログラムには、 POWER5+™ プ

項目	説明
	pwr6 POWER6 モードを指定します。 ソース・プログラムには、 POWER6 プロセッサーと互換性のある命令しか入れることができません。
	pwr6e POWER6E モードを指定します。 ソース・プログラムには、 POWER6E プロセッサーと互換性のある命令のみを含めることができます。
	pwr7 POWER7 モードを指定します。 ソース・プログラムには、 POWER7 プロセッサーと互換性のある命令しか入れることができません。
	pwr8 POWER8 モードを指定します。 ソース・プログラムには、 POWER8 プロセッサーと互換性のある命令しか入れることができません。
	>|**pwr9** POWER9 モードを指定します。 ソース・プログラムには、 POWER9 プロセッサーと互換性のある命令しか入れることができません。|<
	>|**pwr10** これは Power10 モードを指定します。 ソース・プログラムには、 Power10 プロセッサーと互換性のある命令しか入れることができません。|<
	any 非特定の POWER ファミリー/PowerPC アーキテクチャーまたは実装モード。 これには、有効なアーキテクチャーまたは実装のいずれかからの命令の組み合わせが含まれます。
	601 PowerPC アーキテクチャー、 PowerPC 601 RISC マイクロプロセッサー・モードを指定します。 ソース・プログラムには、 PowerPC アーキテクチャー、 PowerPC 601 RISC マイクロプロセッサー用の命令のみを含めることができます。 その他の命令はすべてエラーとなります。  **重要 :** 将来の PowerPC システムに移植できるようにするアプリケーションでは、 **601** アセンブリー・モードを使用しないことをお勧めします。 このようなアプリケーションには、**com** または **ppc** アセンブリー・モードを使用する必要があります。 PowerPC 601 RISC Microprocessor は、 PowerPC アーキテクチャーと、 PowerPC アーキテクチャーには含まれていないいくつかの POWER® ファミリー命令を実装しています。 これにより、既存の POWER ファミリー・アプリケーションを PowerPC システムで許容できるパフォーマンスで実行できます。 将来の PowerPC システムには、この機能はありません。 **601** アセンブリー・モードでは PowerPC 601 RISC マイクロプロセッサーによって提供されるすべての命令の使用が許可されるため、 **601** アセンブリー・モードでは、既存の POWER ファミリー・システムでは実行されず、将来の PowerPC システムでは許容されるパフォーマンスが得られない可能性があります。

項目	説明
	603 PowerPC® アーキテクチャー、 PowerPC 603 RISC マイクロプロセッサー・モードを指定します。 ソース・プログラムには、 PowerPC® アーキテクチャー、 PowerPC 603 RISC マイクロプロセッサー用の命令のみを含めることができます。 その他の命令はすべてエラーとなります。 **604** PowerPC アーキテクチャー、 PowerPC 604 RISC マイクロプロセッサー・モードを指定します。 ソース・プログラムには、 PowerPC アーキテクチャー、 PowerPC 604 RISC マイクロプロセッサー用の命令のみを含めることができます。 その他の命令はすべてエラーとなります。 **ppc970 または 970** PPC970 モードを指定します。 ソース・プログラムには、 PPC970 プロセッサーと互換性のある命令しか入れることができません。 **A35** A35 モードを指定します。 ソース・プログラムには、A35 用の命令しか入れることはできません。 その他の命令はすべてエラーとなります。

例

1. ターゲット環境を POWER® ファミリー・アーキテクチャー、 POWER® ファミリー実装に設定するには、以下のようにします。

```
.machine "pwr"
```

2. ターゲット環境を非特定の POWER® ファミリー/PowerPC® アーキテクチャーまたは実装モードに設定するには、以下のようにします。

```
.machine "any"
```

3. デフォルトのアセンブリー・モードを明示的に選択するには、以下のようにし

```
.machine ""
```

4. 以下のコード・フラグメントのアセンブラー出力の例は、 `.machine "push"` および `.machine "pop"`の使用法を示しています。

```
 push1.s                              V4.1               04/15/94
File# Line#  Mode Name   Loc Ctr   Object Code           Source
0       1 |                                            .machine   "pwr2"
0       2 |                                            .csect    longname1[PR]
0       3 |  PWR2 longna 00000000  0000000a            .long   10
0       4 |  PWR2 longna 00000004  329e000a            ai    20,30,10
0       5 |  PWR2 longna 00000008  81540014            l    10, 20(20)
0       6 |                                            .machine "push"
0       7 |                                            .machine  "ppc"
0       8 |                                            .csect  a2[PR]
0       9 |  PPC  a2     00000000  7d4c42e6             mftb  10
0      10 |                                            .machine "pop"
0      11 |  PWR2 a2     00000004  329e000a            ai    20,30,10
0      12 |
```

関連概念

疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

as コマンドを使用したアセンブル
as コマンドはアセンブラーを呼び出します。

固定小数点プロセッサー
固定小数点プロセッサーは非特権命令を使用し、GPR は内部ストレージ・メカニズムとして使用されます。

固定小数点回転およびシフト指示
固定小数点回転命令とシフト命令は、レジスターの内容を回転させます。

.option 疑似命令

目的

プログラムのアセンブル時に使用するオプションを指定します。

構文

```
.option StringConstant
```

説明

.option 疑似命令は、プログラムのアセンブル時に使用するオプションを指定します。 **.option** 疑似命令は、アセンブラーの呼び出し時に **-a** フラグで指定されたオプションをオーバーライドします。

.option 疑似命令は、ソース・プログラム内で複数回呼び出すことができます。 呼び出しで指定された新しいオプションは、 **.option** 疑似命令が再び呼び出されるまで有効のままです。

パラメーター

項目	説明
StringConstant (StringConstant)	1 つ以上のオプションを指定します。 このパラメーターを指定しない場合、または空ストリングを指定した場合は、すべてのオプションが対応するデフォルト値に復元されます。 このパラメーターには、以下の値を指定できます。 **push** 現在のオプションをスタックに保存します。 **ポップ (pop)** **.option "push"** 疑似命令の最新の呼び出しによって保管されたオプション値を復元します。 **注 : push** パラメーターおよび **pop** パラメーターの使用目的は、組み込まれているファイル内にあり、これらのパラメーターによってオプションが変更されます。 オプションを変更する前に、組み込みファイルで **.option "push"** 疑似命令を使用してください。 オプションをリストアするには、組み込みファイルの末尾に **.option "pop"** 疑似命令を使用します。 **.option "push"** 疑似命令の各オカレンスには、対になっている **.option "pop"** 疑似命令がなければなりません。 **align-prefixed-csect=<yes/no>** これは、必要であれば、接頭部付きの命令を含む制御セクション (csect) の配置を少なくとも 64 バイト境界にまで増やすかどうかを指定します。 64 バイトが最小配置であり、その場合は、アセンブルされたプログラムがリンクされるときに接頭部付きの命令が正しく配置されます。 **align-prefixed-csect** オプションに対して **yes** を指定すると、必要に応じて、接頭部付きの命令を含む csect の配置が増やされます。 **align-prefixed-csect** オプションに対して **no** を指定した場合に、コマンド・ラインで **-w** フラグが使用されているときは、接頭部付きの命令が、十分には厳密になっていない配置を持つ csect に含まれていると、警告メッセージが表示されます。 詳しくは、『.align pseudo-op』を参照してください。 **align-prefixed-op=<yes/no>** これは、接頭部付きの命令が 64 バイト境界を超える場合に、命令の前に no-op 命令を付けて接頭部付きの命令を配置するどうかを指定します。 **align-prefixed-op** オプションに対して **yes** を指定すると、必要に応じて、接頭部付きの命令が配置されます。 **align-prefixed-op** オプションに対して **no** を指定した場合に、コマンド・ラインで **-w** フラグが使用されているときは、接頭部付きの命令が 64 バイト境界を超える場合に、警告メッセージが表示されます。 詳しくは、『.align pseudo-op』を参照してください。

.org 疑似命令

目的

現在のロケーション・カウンターの値を設定します。

構文

項目	説明
組織 (.org)	式

説明

.org 疑似命令は、現在のロケーション・カウンターの値を *Expression* に設定します。 この疑似命令は、ロケーション・カウンターを減分することもできます。 アセンブラーは制御セクション (csect) 指向であるため、ロケーション・カウンターを現行 csect の外部に置く絶対式または式は許可されません。

パラメーター

項目	説明
式	現在のロケーション・カウンターの値を表します。

例

以下の例は、 **.org** 疑似操作の使用法を示しています。

```
# Assume assembler location counter is 0x114.
.org $+100
#Skip 100 decimal byte (0x64 bytes).
.
.
# Assembler location counter is now 0x178.
```

関連概念
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.space 疑似命令

.ptr 疑似命令

目的

式を連続するポインター・サイズ・エレメントにアセンブルします。

構文

項目	説明
.ptr (.ptr)	*Expression*[, *Expression*, ...]

説明

.ptr 疑似命令は、式を 32 ビット・モードでは連続ワードに、64 ビット・モードでは連続ダブルワードにアセンブルします。 **.ptr** 疑似命令では、同じソース・コードを 32 ビット・モードと 64 ビット・モードの両方で使用することができます。これは、式に再配置可能参照が含まれている場合に最も便利です。 **.ptr** 疑似命令は、32 ビット・モードでは **.long** 疑似命令に相当し、64 ビット・モードでは **.llong** 疑似命令に相当します。

フルワード位置合わせは、現行セクションが DWARF セクションでない限り、必要に応じて行われます。

パラメーター

項目	説明
式	絶対式。

例

以下の例は、 **.ptr** 疑似命令の使用法を示しています。

```
        .extern             foo[RW]
        .csect              mydata[RW]
        .ptr                foo[RW]       # Pointer to foo with appropriate relocation
                                          # 4 bytes in 32-bit mode
                                          # 8 bytes in 32-bit mode
```

関連概念
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.long 疑似命令

.quad 疑似命令

目的

次のフルワード位置にクワッド浮動小数点定数を保管します。 浮動小数点データの位置合わせ要件は、32 ビット・モードと 64 ビット・モードの間で一貫しています。

構文

項目	説明
.quad **(.quad)**	浮動定数 *(FloatingConstant)*

例

以下の例は、 **.quad** 疑似命令の使用法を示しています。

```
.quad 3.4
.quad -77
.quad 134E12
.quad 5e300
.quad 0.45
```

上記の宣言は、それぞれ 16 バイトのストレージを予約します。

関連概念
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.float 疑似命令
.double 疑似命令

.ref 疑似命令

目的

1 つ以上のシンボルについて、再配置テーブルに R_REF タイプ・エントリーを作成します。

構文

.ref 名前[**,***Name*...]

説明

.ref 疑似命令は、同じ場所に複数の RLD 項目を作成することをサポートします。 この psuedo-op は、一部のコンパイラーの出力で使用され、リンケージ・エディターが、使用されているがテキスト・セクションまたはデータ・セクションで明示的に参照されていないルーチンを破棄しないようにします。

例えば、C++ では、コンストラクターとデストラクターを使用して、クラス・オブジェクトを構成および破棄します。 コンストラクターおよびデストラクターは、テキスト・セクションで明示的な参照を行わずに、ランタイム環境からのみ呼び出されることがあります。

ソース・プログラム内の **.ref** 疑似命令の配置には、以下の規則が適用されます。

- **.ref** pseudo-op は、ストレージ・マッピング・クラスが BS または UC の dsect または csect に含めることはできません。

- **.ref** 疑似命令は、共通セクションまたはローカル共通セクションに組み込むことはできません。

.ref 疑似命令 (*Name* パラメーター) のオペランドには、以下の規則が適用されます。

- シンボルは、現行ソース・モジュールで定義する必要があります。
- 外部シンボルは、 **.extern** または **.globl** によって定義される場合に使用できます。
- 現行ソース・モジュール内で、シンボルは csect 名 (Qualname を意味する) または csect で定義されたラベルにすることができます。
- 以下のシンボルは、 **.ref** オペランドには使用できません。
  - pseudo-op **.dsect** 名
  - dsect 内で定義された ラベル
  - ストレージ・マッピング・クラスが BS または UC の csect 名
  - ストレージ・マッピング・クラスが BS または UC の csect 内で定義された ラベル
  - 再配置不可の式タイプを表す疑似演算子 **.set** *Name* オペランド

パラメーター

項目	説明
Name	再配置テーブル内の R_REF タイプ・エントリーを作成するシンボルを指定します。

例

以下の例は、 **.ref** 疑似命令の使用法を示しています。

```
       .csect a1[pr]
C1:    l 10, 20(20)
       .long  0xff
       .csect  a2[pr]
       .set   r10,10
       .extern C4
C2:    .long   10
C3:    .long   20
       .ref C1,C2,C3
       .ref C4
```

関連概念

疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

式の組み合わせ処理
式の中の項は、2 項演算子と組み合わせることができます。

.rename 疑似命令

目的

正しくない名前または望ましくない名前の同義語または別名を作成します。

構文

項目	説明
.rename	名前, *StringConstant*

説明

アセンブラー・ソース・ファイル内のシンボルに使用できる文字に関する制約事項は、29 ページの『シンボルの構成』に定義されています。 記号にブランクまたは特殊文字を含めることはできません。また、数字で始めることもできません。

特殊文字を含む必要があるグローバル・シンボル、またはアセンブラー構文では正しくない文字を含む必要があるグローバル・シンボルの場合、 **.rename** 疑似命令によってそれを行う方法が提供されます。

.rename pseudo-op は、アセンブリーの最後に、すべてのグローバル・シンボルの *Name* パラメーターを *StringConstant* 値に変更します。 ローカル・アセンブリーへの内部参照は、 *Name* に対して行われます。 グローバル名は *StringConstant* です。

パラメーター

項目	説明
Name	シンボルを表します。 グローバルにするには、**.extern** または **.globl** ステートメントに *Name* パラメーターを指定する必要があります。
StringConstant (StringConstant)	アセンブリーの終了時に *Name* パラメーターが変更される値を表します。

例

以下の例は、 **.rename** 疑似命令の使用法を示しています。

```
        .csect mst_sect[RW]
        .globl mst_sect[RW]
OK_chars:
        .globl OK_chars
        .long OK_chars
        .rename OK_chars,"$_SPECIAL_$_char"
        .rename mst_sect[RW],"MST_sect_renamed"
```

.set 疑似命令

目的

タイプと値の両方の式に等しい記号を設定します。

構文

項目	説明
.set (.set)	名前, *Expression*

説明

.set 疑似命令は、 *Name* 記号を type と value の *Expression* 値に等しく設定します。 **.set** 疑似命令を使用すると、頻繁に使用される式でのエラーを回避するのに役立ちます。 式を記号と等価にしてから、式ではなく記号を参照してください。 式の値を変更するには、 **.set** ステートメント内でのみ変更します。 ただし、 **.set** 割り当てはアセンブリー時にのみ行われるため、プログラムの再アセンブルが必要です。

Expression パラメーターは、アセンブラーが **.set** 疑似命令を検出したときに評価されます。 この評価は、式の組み合わせ処理 の規則を使用して行われ、評価結果のタイプと値は内部的に保管されます。 「式」を評価した結果が無効なタイプになった場合、シンボル 「名前」 を使用するすべての命令でエラーが発生します。

Name が他の命令で使用されている場合は、元の式定義ではなく、シンボル *Name* の保管タイプと値が使用されます。

パラメーター

項目	説明
Name	**.set** ステートメントの定義の前に使用できるシンボルを表します。モジュール内では順方向参照が許可されます。

項目	説明
式	シンボル *Name* のタイプと値を定義します。 式で参照されるシンボルを定義する必要があります。順方向参照は許可されません。 シンボルを未定義の外部式にすることはできません。シンボルは、**.set** 疑似命令が表示される制御セクション内にある必要はありません。 *Expression* パラメーターは、レジスター番号を参照することもできますが、実行時のレジスターの内容を参照することはできません。

例

1. 以下の例は、 **.set** 疑似命令の使用法を示しています。

```
.set ap,14 # Assembler assigns value 14
           # to the symbol ap -- ap
           # is absolute.
  .
  .
lil ap,2

        # Assembler substitutes value 14
        # for the symbol.
        # Note that ap is a register
        # number in context
        # as lil's operand.
```

2. 以下の例では、無効なタイプが原因でアセンブリー・エラーが発生します。

```
    .csect  a1[PR]
 L1:  l       20,30(10)
      .csect  a2[rw]
      .long   0x20
 L2:  .long   0x30
      .set    r1, L2 - L1  # r1 has type of E_REXT
                           # r1 has value of 8
      .long   r1 + 10
      .long   L2 - r1      #  Error will be reported.
                           #    L2 is E_REL
                           #    r1 is E_REXT
                           #  E_REL - E_REXT ==> Invalid type
```

.short 疑似命令

目的

式を連続するハーフワードにアセンブルします。

構文

項目	説明
.short	式[, *Expression, ...*]

説明

.short 疑似命令は、式 を連続するハーフワードにアセンブルします。 現行セクションが DWARF セクションでない限り、ハーフワード位置合わせが行われます。

パラメーター

項目	説明
式	命令がハーフワードにアセンブルする式を表します。 *Expression* パラメーターは、どのレジスターの内容も参照できません。 *Expression* 値がハーフワードより長い場合は、左側が切り捨てられます。

例

以下の例は、 **.short** 疑似命令の使用法を示しています。

```
.short 1,0x4444,fooble-333,0
```

.source 疑似命令

目的

ソース言語タイプを識別します。

構文

項目	説明
ソース	*StringConstant*

説明

.source 疑似命令は、ソース言語タイプを識別し、リンケージ・エディターに必要なシンボル・テーブル情報を提供します。 カスケード・コンパイラーの場合、シンボル・テーブル情報は、高水準ソース言語タイプを示すためにコンパイラーからアセンブラーに渡されます。 デフォルトのソース言語タイプは「アセンブラー」です。

パラメーター

項目	説明
StringConstant (StringConstant)	有効なプログラム言語名を指定します。 このパラメーターには大/小文字の区別はありません。 指定された値が有効でない場合、言語 ID は「アセンブラー」にリセットされます。 以下の値が定義されています。 **0x00** C **0x01** FORTRAN **0x02** Pascal **0x03** Ada **0x04** PL/1 **0x05** BASIC **0x06** LISP **0x07** COBOL **0x08** Modula2 **0x09** C++ **0x0a** RPG **0x0b** PL8、PLIX **0x0c** アセンブラー

例

ソース言語タイプを C++ に設定するには、以下のようにします。

```
.source "C++"
```

関連概念

疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

ソース言語タイプ
アセンブラーは、ソース言語タイプを記録します。

.space 疑似命令

目的

出力ファイル内の指定されたバイト数をスキップし、それらを 2 進ゼロで埋めます。

構文

項目	説明
スペース	数値

説明

.space は、出力ファイル内の *Number* で指定されたバイト数をスキップし、それらを 2 進ゼロで埋めます。 **.space** 疑似命令は、制御セクション (csect) 内のストレージのチャンクを予約するために使用できます。

パラメーター

項目	説明
Number	スキップするバイト数を指定する絶対式を表します。

例

以下の例は、 **.space** 疑似命令の使用法を示しています。

```
.csect data[rw]
.space 444
  .
  .
foo:    # foo currently located at offset 0x1BC within
        # csect data[rw].
```

関連概念
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.stabx 疑似命令

目的

デバッガーが必要とする追加情報を提供します。

構文

項目	説明
「**.stabx**」	*StringConstant*、 *Expression1*、 *Expression2*、 *Expression3*

説明

.stabx 疑似命令は、デバッガーが必要とする追加情報を提供します。 アセンブラーは、デバッガーに必要な stabstring 情報を提供する *StringConstant* 引数を .debug セクションに配置します。

.stabx 疑似命令は、コンパイラーによって通常挿入されます。

パラメーター

項目	説明
StringConstant	デバッガーに必要な Stabstring 情報を提供します。
Expression1	文字ストリングのシンボル値を表します。 この値は、ストレージ・マッピング・クラスに依存します。 例えば、ストレージ・マッピング・クラスが C_LSYM の場合、値はスタック・フレームに関連したオフセットです。 ストレージ・マッピング・クラスが C_FUN の場合、値は収容制御セクション (csect) 内のオフセットです。
Expression2	文字ストリングのストレージ・クラスを表します。

項目	説明
Expression3	文字ストリングのシンボル・タイプを表します。

例

以下の例は、 **.stabx** 疑似命令の使用法を示しています。

```
.stabx "INTEGER:t2=-1",0,140,4
```

関連概念
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.function 疑似命令
関連情報
デバッグ・セクション

.string 疑似命令

目的

文字値を連続するバイトにアセンブルし、ストリングをヌル文字で終了します。

構文

項目	説明
ストリング	*StringConstant*

説明

.string 疑似命令は、 *StringConstant* によって表される文字値を連続バイトにアセンブルし、ストリングをヌル文字で終了させます。

パラメーター

項目	説明
StringConstant (StringConstant)	連続したバイトにアセンブルされた文字値のストリングを表します。

例

以下の例は、 **.string** 疑似命令の使用法を示しています。

```
mine:   .string "Hello, world!"
# This produces
# 0x48656C6C6F2C20776F726C642100.
```

関連概念
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.byte 疑似命令
.vbyte 疑似命令

.tbtag 疑似命令

目的

デバッグ・プログラムのトレースバックを実行できるデバッグ・トレースバック・タグを定義します。前にゼロのワードが付きます。

構文

.tbtag *Expression1*、*Expression2*、*Expression3*、*Expression4*、*Expression5*、*Expression6*、*Expression7*、*Expression8*、[*Expression9*、*Expression10*、*Expression11*、*Expression12*、*Expression13*、*Expression14*、*Expression15*、 *Expression16*]

説明

.tbtag 疑似命令は、フィールド要件に応じて *Expression* を連続したバイト、ワード、およびハーフワードにアセンブルすることにより、トレースバック・タグを定義します。 命令には、8 つの式 (*Expression1* から *Expression8* まで) または 16 個の式 (*Expression1* から *Expression16* まで) を含めることができます。 それ以外は構文エラーです。 コンパイラーは、通常、トレースバック情報をマシン・インストラクションの最後にプログラムに挿入し、情報の開始をシグナル通知するためにゼロのストリングを追加します。

パラメーター

項目	説明	説明
Expression1	version	/*Traceback フォーマット・バージョン */
Byte	Byte	Byte
Expression2	LANG	/* 言語値 */
Byte	Byte	Byte
	TB_C (B)	0
	TB_FORTRAN	1
	TB_PASCAL	2
	TB_ADA	3
	TB_PL1	4
	基本 (TB_BASIC)	5
	ISP (TB_LISP)	6
	TB_COBOL	7
	TB_MODULA2	8
	TB_CPLUSPLUS	9
	TB_RPG	10
	TB_PL8	11
	TB_ ASM (ASM)	12
Expression3		/*Traceback 制御ビット */
Byte	Byte	Byte
	グローバル・リンク	ビット 7。 ルーチンがグローバル・リンケージの場合に設定されます。
	is_eprol	ビット 6。 市外のエピローグ/プロローグの場合に設定します。
	Has_tboff	ビット 5。 プロシージャーの先頭からのオフセットが保管されている場合に設定されます。
	処理中	ビット 4。 ルーチンが内部ルーチンの場合に設定されます。
	HAS_CTL	ビット 3。 ルーチンが制御されたストレージを含む場合に設定されます。

項目	説明	説明
	タクレス	ビット 2。 ルーチンに TOC が含まれていない場合に設定されます。
	Fp_Present	ビット 1。 ルーチンが FP 操作を実行する場合に設定されます。
	ログの異常終了	ビット 0。 ルーチンが制御されたストレージを含む場合に設定されます。
Expression4		/*Traceback 制御ビット (続き) */
Byte	Byte	Byte
	内部 hndl	ビット 7。 ルーチンが割り込みハンドラーの場合に設定されます。
	名前が存在	ビット 6。 プロシージャー・テーブルに名前が存在する場合に設定されます。
	jpes_alloca	ビット 5。 alloca がストレージの割り振りに使用される場合に設定されます。
	cl_dis_inv	ビット 4、3、2。 On-condition ディレクティブ WALK_ONCOND,0、Walk スタック; 状態を復元しない DISCARD_ONCOND,1。スタックをウォークして破棄します。 INVOKE_ONCOND,1、特定のシステム・ルーチンの呼び出し
	保管中 (saves_cr)	ビット 1。 プロシージャーが条件レジスターを保管する場合に設定します。
	保存 _lr	ビット 0。 プロシージャーがリンク・レジスターを保管する場合に設定します。
Expression5		/*Traceback 制御ビット (続き) */
Byte	Byte	Byte
	ストレージ _bc	ビット 7。 プロシージャーがバックチェーンを保管する場合に設定します。
	spare2	ビット 6。 予備ビット。
	FPR_SAVE	ビット 5、4、3、2、1、0。 保管された FPR の数、最大 32。
Expression6		/*Traceback 制御ビット (続き) */
Byte	Byte	Byte
	spare3	ビット 7、6。 スペア・ビット。
	GPR_SAVE	ビット 5、4、3、2、1、0。 保管された GPR の数、最大 32。
Expression7	fixedparms	/*Traceback 制御ビット (続き) */
Byte	Byte	Byte
Expression8		
Byte	Byte	Byte
	浮動小数点パラメーター	ビット 7、6、5、4、3、2、1。 浮動小数点パラメーターの数。
	パルムソンスク	ビット 0。 すべてのパラメーターをスタックに配置する場合に設定します。
Expression9	パルミノ	/* パラメーターの順序およびタイプ・コーディング */
ワード	ワード	ワード

項目	説明	説明
	'0'	固定パラメーター。
	'10'	単精度浮動小数点パラメーター。
	'11'	倍精度浮動小数点パラメーター。
Expression10	TB オフセット	/* コードの先頭から tb テーブルまでのオフセット */
ワード	ワード	ワード
Expression11	ハンドマスク	/* 処理される割り込み */
ワード	ワード	ワード
Expression12	制御情報	/*CTL アンカーの数 */
ワード	ワード	ワード
Expression13	ctl_info_disp	/* スタックへの各アンカーの変位 */
ワード	ワード	ワード
Expression14	名前の長さ	/* プロシージャー名の長さ */
ハーフワード (halfword)	ハーフワード (halfword)	ハーフワード (halfword)
Expression15	名前	/*Name (名前) */
Byte	Byte	Byte
Expression16	割り振りレジスター	/* alloca 自動ストレージの登録 */
Byte	Byte	Byte

例

次の例は、 **.tbtag** 疑似命令の使用法を示しています。

```
.tbtag 1,0,0xff,0,0,16,0,0
```

関連概念

疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.tbtag 疑似命令
.byte 疑似命令

.tc 疑似命令

目的

式を目次 (TOC) 項目にアセンブルします。

構文

項目	説明
.tc	[名前]**[**TC**]**, *Expression*[, *Expression, ...*]

注 : TC を含む太字体の大括弧は構文の一部であり、オプション・パラメーターを指定 しません 。

説明

.tc 疑似命令は、 式 を TOC エントリーにアセンブルします。このエントリーには、ルーチンのアドレス、関数記述子のアドレス、または外部変数のアドレスが含まれます。 **.tc** ステートメントは、**.toc** 疑似命令の有効範囲内にのみ指定できます。 TOC エントリーは、本文として再配置できます。 TOC エントリー・ステートメントは、最初の **.toc** ステートメントによって宣言された TOC 全体の先頭に相対するローカル・ラベルを持つことができます。 TOC エントリーに含まれるアドレスは、これらのローカル・ラベルと TOC レジスター GPR 2 を使用してアクセスできます。

1 つのアドレスのみを含む TOC エントリーは、バインダーによって結合されます。 これは、TOC エントリーが同じ名前を持ち、同じ制御セクション (csect) (シンボル) を参照している場合に発生する可能性があります。 csect 内でゼロ以外のオフセットを参照する TOC 項目をコーディングする場合は、注意してください。 TOC エントリーが意図せずに結合されないようにするには、csect 内の異なるオフセットを参照する TOC エントリーに固有の名前を割り当てる必要があります。

パラメーター

項目	説明
Name	作成される TOC エントリーの名前を指定します。 *StorageMappingClass* は、TOC エントリーの TC です。 名前**[TC]** は、該当する場合に TOC エントリーを参照するために使用できます。
式	TOC エントリーに入れるシンボルまたは式を指定します。

例

次の例は、 **.tc** 疑似命令の使用法を示しています。

```
.toc
# Create three TOC entries, the first
# with the name proga, the second
# with the name progb, and the last
# unnamed.
T.proga:        .tc proga[TC],progr[RW],dataA
T.progb:        .tc progb[TC],proga[PR],progb[PR]
T.progax:       .tc  proga[TC],dataB
                .tc       [TC],dataB
                .csect proga[PR]
# A .csect should precede any statements following a
# .toc/.tc section which do not belong in the TOC.
                l 5,T.proga(2)   # The address of progr[RW]
                                 # is loaded into GPR 5.
                l 5,T.progax(2)  # The address of progr[RW]
                                 # is loaded into GPR 5.
                l 5,T.progb+4(2) # The address of progb[PR]
                                 # is loaded into GPR 5.
```

関連概念

疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.csect 疑似演算子
.toc 疑似命令
.tocof 疑似命令

.toc 疑似命令

目的

モジュールの目次を定義します。

構文

.toc (.toc)

説明

.toc 疑似命令は、モジュールの目次 (TOC) アンカーを定義します。 TOC セクション内の項目は、 **.toc** 疑似命令の有効範囲内で **.tc** 疑似命令を使用して宣言できます。 **.toc** 疑似命令の有効範囲は、 **.csect** 疑似命令の有効範囲と類似しています。 TOC は、 **.toc** が現れる場所であれば、アセンブリー全体にわたって継続することができます。

例

以下の例は、 **.toc** 疑似命令の使用法を示しています。

```
.toc
# Create two TOC entries. The first entry, named proga,
# is of type TC and contains the address of proga[RW] and dataA.

# The second entry, named progb, is of type TC and contains
# the address of progb[PR] and progc[PR].

T.proga:          .tc proga[TC],proga[RW],dataA
T.progb:          .tc progb[TC],progb[PR],progc[PR]

.csect proga[RW]

# A .csect should precede any statements following a .toc/.tc
# section which do not belong in the TOC.

.long TOC[tc0]

# The address of the TOC for this module is placed in a fullword.
```

関連概念
.tc 疑似命令
.tocof 疑似命令

.tocof 疑似命令

目的

ローカル・シンボルの定義を外部シンボルの目次にして、そのローカル・シンボルを式の中で使用できるようにします。

構文

項目	説明
.tocof (.tocof)	*Name1*, *Name2*

説明

.tocof pseudo-op は、 *Name2* パラメーターをグローバル・シンボルとして宣言し、シンボル *Name2* を含む別のモジュールの目次 (TOC) として *Name1* シンボルにマークを付けます。 結果として、ローカル・シンボルを外部シンボルの TOC として定義することができます。これにより、ローカル・シンボルを式で使用したり、呼び出されたモジュールの TOC (通常は **.tc** ステートメント) を参照したりすることができます。この疑似命令は、このデータを TOC 外部シンボルのアドレスに初期化する再配置辞書項目 (RLD) を生成します。 **.tocof** 疑似命令は、呼び出し元が制御権を移動する前に、呼び出されたモジュールの TOC のアドレスを最初にロードする必要があるモジュール間呼び出しに使用できます。

パラメーター

項目	説明
Name1	*Name2* 値を含むモジュールの TOC として機能するローカル・シンボルを指定します。 *Name1* シンボルは、 **.tc** ステートメントに現れる必要があります。
Name2	TOC を含むモジュール内に存在するグローバル・シンボルを指定します。

例

以下の例は、 **.tocof** 疑似命令の使用法を示しています。

```
tocbeg:   .toc
apb:   .tc [tc],pb,tpb
# This is an unnamed TOC entry
# that contains two addresses:
# the address of pb and
# the address of the TOC
# containing pb.
.tocof tpb,pb
.set always,0x14
.csect [PR]
.using tocbeg,rtoc
l 14,apb
# Load R14 with the address
# of pb.
l rtoc,apb+4
# Load the TOC register with the
# address pb's TOC.
mtspr lr,14
# Move to Link Register.
bcr always,0
# Branch Conditional Register branch
# address is contained in the Link
# register.
```

関連概念

疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

目次の理解とプログラミング
TOC は、XCOFF ファイル内のオブジェクトを検索するために使用されます。

.tc 疑似命令
.toc 疑似命令

.uleb128 疑似命令

目的

可変長の式をアセンブルします。

構文

項目	説明
.uleb128	*Expression*[, *Expression*, ...]

説明

.uleb128 疑似命令は、符号なしリトル・エンディアン・ベース 128 (LEB128) フォーマットを使用して、絶対式を連続バイトにアセンブルします。 各式は、符号なしの 64 ビット式として扱われ、符号なしの LEB128 形式に変換されてから、値に必要な数のバイトにアセンブルされます。

.uleb128 疑似命令では、位置合わせは行われません。

パラメーター

項目	説明
式	絶対式。

例

1. 以下の例は、 .uleb128 疑似命令の使用法を示しています。

```
       .uleb128      127         #Emits 0x7F
```

```
        .uleb128      128         #Emits 0x80, 0x01
        .uleb128      0x8000000000000000 # Emits 0x80,0x80,0x80,0x80,
                                  #       0x80,0x80,0x80,0x80,0x00
```

関連概念
疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.leb128 疑似命令

. 疑似命令の使用

目的

ユーザーが基底アドレスを指定し、基底レジスター番号を割り当てることができるようにします。

構文

項目	説明
. 使用	*Expression*, *Register*

説明

.using 疑似命令は、実行時に *Register* パラメーターに *Expression* のプログラム・アドレスが含まれていると想定して、基底アドレスとして式を指定し、基底レジスターを割り当てます。 シンボル名は、事前に定義しておく必要はありません。

> **注: .using** pseudo-op は基底レジスターをロードしません。プログラマーは、暗黙のアドレス参照を使用する前に、基底アドレスが基底レジスターにロードされていることを確認する必要があります。

.using 疑似命令は、暗黙ベースのアドレスを持つ命令にのみ影響します。 これは、制御セクション (csect) 名および csects 内のすべてのラベルに対して発行できます。 また、dsect 名および dsects 内のすべてのラベルで使用することもできます。 その他のタイプの外部シンボルは許可されません (**.extern**)。

範囲の使用 (Using Range)

.using 疑似命令 (範囲を使用) の範囲は -32768 または 32767 バイトで、**.using** 疑似命令で指定された基底アドレスから始まります。 アセンブラーは、使用範囲内にある各暗黙的アドレス参照 (または式) を、明示的ベースのアドレス形式に変換します。 使用範囲外の参照についてはエラーが報告されます。

1 つの **.using** 疑似命令の基底アドレスが別の **.using** 疑似命令の範囲内にある場合、2 つの使用範囲がオーバーラップします。 範囲オーバーラップを使用すると、アセンブラーは、基底アドレスからの最小の符号付きオフセットを変位として選択することにより、暗黙アドレス参照を変換します。 対応する基底レジスターは、明示アドレス形式で使用されます。 これは、2 番目の **.using** 疑似命令の後に現れる暗黙アドレスにのみ適用されます。

次の例では、 base2 と data[PR] の使用範囲がオーバーラップしています。 2 番目の **l** 命令は、2 番目の **.using** pseudo-op の後にあります。 data[PR] から d12 へのオフセットは、base2 から d12 へのオフセットより大きいため、 base2 が引き続き選択されます。

```
        .csect  data[PR]
        .long   0x1
dl:     .long   0x2
base2:  .long   0x3
        .long   0x4
        .long   0x4
        .long   0x5
d12:    .long   0x6
        l 12, data_block.T(2)   # Load addr. of data[PR] into r12
        ca1 14, base2(12)       # Load addr. of base2 into r14
        .using base2, 14
        l 4, d12                # Convert to 1 4, 0xc(14)
        .using data[PR], 12
        l 4, d12                # Converts to 1 4, 0xc(14)
                                # because base2 is still chosen
```

```
        .toc
data_block.T:   tc data_block[tc], data[PR]
```

.using 疑似命令を追跡するためにアセンブラーが使用する内部使用テーブルがあります。 使用する表の各項目は、 **.using** 疑似命令の *Expression* パラメーターで指定された式またはラベルを含む csect を指します。 使用中の表は、 **.using** 疑似命令によってのみ更新されます。 ソース・プログラム内の **.using** 疑似命令の位置は、暗黙ベース・アドレスの変換の結果に影響を与えます。 次の 2 つの例は、この変換を示しています。

例 1:

```
            .using   label1,4
            .using   label2,5
            .csect   data[RW]
label1:     .long    label1
            .long    label2
            .long    8
label1_a:   .long    16
            .long    20
label2:     .long    label2
            .long    28
            .long    32
label2_a:   .long    36
            .long    40
            .csect   sub1[pr]
            1        6,label1_a      # base address label2 is
                                     # chosen, so convert to:
                                     # 1 6, -8(5)
            1        6,label2_a      # base address label2 is
                                     # chosen, so convert to:
                                     # 1 6, 0xc(5)
```

例 2:

```
            .csect  data[RW]
label1:     .long   label1
            .long   label2
            .long   12
label1_a:   .long   16
            .long   20
label2:     .long   label2
            .long   28
            .csect  sub2[pr]
            .using  label1,4
            1       6,label1_a       # base address label1 is
                                     # chosen, so convert to:
                                     # 1 6, 0xc(4)
           .using   label2,5
            1       6,label1_a       # base address label2 is
                                     # chosen, so convert to:
                                     # 1 6, -8(5)
```

2 つの異なる **.using** 疑似命令に同じ基底アドレスが指定されているが、使用される基底レジスターが異なる場合は、2 つの使用範囲が一致します。 アセンブラーは、明示ベースのアドレス形式に変換するときに、基底レジスターとして小さい番号のレジスターを使用します。これは、テーブルを使用すると、最も小さい番号のレジスターから最も大きい番号のレジスターへと検索されるためです。 次の例は、このケースを示しています。

```
        .csect  data[PR]
        .long   0x1
dl:     .long   0x2
base2;  .long   0x3
        .long   0x4
        .long   0x5
dl2:    .long   0x6
        1 12, data_block.T(2)   # Load addr. of data[PR] into r12
        1 14, data_block.T(2)   # Load addr. of data[PR] into r14
        .using data[PR], 12
        1 4, dl2                # Convert to: 1 4, 0x14(12)
        .using data[PR], 14
        1 4, dl2                # Convert to: 1 4, 0x14(12)
```

```
        .toc
data_block.T:   .tc data_block[tc], data[PR]
```

ドメインの使用

.using 疑似命令のドメイン (使用ドメイン) は、 **.using** 疑似命令が csect に現れる位置から始まり、以下の場合を除き、ソース・モジュールの終わりまで続きます。

- 後続の **.drop** 疑似命令は、先行する **.using** 疑似命令によって割り当てられた同じ基底レジスターを指定します。
- 後続の **.using** 疑似命令は、先行する **.using** 疑似命令によって割り当てられた同じ基底レジスターを指定します。

これらの 2 つの例外は、新しい基底アドレスを使用する方法を提供します。 次の 2 つの例は、これらの例外を示しています。

例 1:

```
        .csect  data[PR]
        .long   0x1
dl:     .long   0x2
base2;  .long   0x3
        .long   0x4
        .long   0x5
dl2:    .long   0x6
        1 12, data_block.T(2)   # Load addr. of data[PR] into r12
        ca1 14, base2(12)       # Load addr. of base2 into r14
        .using base2, 14
        1 4, dl2                # Convert to: 1 4, 0xc(14)
                                # base address base2 is used
        1 14, data_block.T(2)   # Load addr. of data[PR] into r14
        .using data[PR], 14
        1 4, dl2                # Convert to: 1 4, 0x14(14)
        .toc
data_block.T:   .tc data_block[tc], data[PR]
```

例 2:

```
        .csect  data[PR]
        .long   0x1
dl:     .long   0x2
base2;  .long   0x3
        .long   0x4
        .long   0x5
dl2:    .long   0x6
        1 12, data_block.T(2)   # Load addr. of data[PR] into r12
        ca1 14, base2(12)       # Load addr. of base2 into r14
        .using base2, 14
        1 4, dl2                # Convert to: 1 4, 0xc(14)
        .drop 14
        .using data[PR], 12
        1 4, dl2                # Convert to: 1 4, 0x14(12)
        .toc
data_block.T:    .tc data_block[tc], data[PR]
```

注 : アセンブラーは、「ドメインの使用」の外部にある暗黙的なアドレス参照を変換しません。 したがって、これらの暗黙的なアドレス参照が、現行 csect の基底アドレスを定義する **.using** 疑似命令の前、または **.drop** 疑似命令の後に、現行 csect の基底アドレスをすべて除去すると、エラーが報告されます。

次の例は、エラー条件を示しています。

```
        .csect  data[PR]
        .long   0x1
dl:     .long   0x2
base2;  .long   0x3
        .long   0x4
        .long   0x5
dl2:    .long   0x6
        1 4, dl2                # Error is reported here
        1 12, data_block.T(2)   # Load addr. of data[PR] into r12
        1 14, data_block.T(2)   # Load addr. of data[PR] into r14
```

```
        .using data[PR], 12
        l 4, dl2
        l 4, 0x14(12)
        .drop 12
        l 4, dl2                # Error is reported here
        .using data[PR], 14
        l 4, dl2
        l 4, 0x14(14)
        .toc
data_block.T:   .tc data_block[tc], data[PR]
        .csect  data1[PR]
dl3:    .long   0x7
        .using  data[PR], 5
        l 5, dl3                # Error is reported
                                # here, dl3 is in csect
                                # data1[PR] and
                                # Using table has no entry of
                                # csect data1[PR]
        l 5, dl2                # No error, because dl2 is in
                                # data [PR]
```

パラメーター

項目	説明
レジスター	式のレジスター番号を表します。 これは絶対値でなければならず、0 から 31 までの整数に評価されなければなりません。
式	ラベル、またはプログラムへの変位または相対オフセットを表すラベルを含む式を指定します。 シンボルがアセンブリー内で定義された csect または目次 (TOC) 項目である場合、 *Expression* パラメーターは外部シンボルにすることができます。

例

以下の例は、 **.using** 疑似命令の使用法を示しています。

```
.csect data[rw]
.long 0x0, 0x0
d1:     .long 0x25
# A read/write csect contains the label d1.
.csect text[pr]
.using data[rw], 12
l 4,d1
# This will actually load the contents of
# the effective address, calculated by
# adding the address d1 to the address in
# GPR 12, into GPR 4
```

.vbyte 疑似命令

目的

式によって表される値を連続するバイトにアセンブルします。

構文

項目	説明
.vbyte	数値、 「式」

説明

.vbyte 疑似命令は、 *Expression* パラメーターによって表される値を、指定された連続バイト数にアセンブルします。

パラメーター

項目	説明
Number	連続バイト数を指定します。 *Number* 値の範囲は 1 から 4 でなければなりません。
式	連続したバイトにアセンブルされる値を指定します。 *Expression* パラメーターには、外部で定義されたシンボルを入れることはできません。 *Expression* 値が指定されたバイト数より長い場合は、左側が切り捨てられます。

例

以下の例は、 **.vbyte** 疑似命令の使用法を示しています。

```
.csect data[RW]
mine:   .vbyte 3,0x37CCFF
# This pseudo-op also accepts character constants.
.vbyte 1,'c
# Load GPR 4 with address of .csect data[RW].
.csect text[PR]
l 3,mine(4)
# GPR 3 now holds 0x37CCFF.
```

関連概念

疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.byte 疑似命令

.weak 疑似命令

目的

弱いバインディングを持つグローバル・シンボルとしてシンボルを宣言します。

構文

.weak 名前 [, *Visibility*]

説明

.weak pseudo-op は、シンボル *Name* が、リンク時に他のファイルから参照できる、弱いバインディングを持つグローバル・シンボルであることを示します。 **.extern**、 **.globl**、または **.comm** pseudo-op を使用して、グローバル・シンボルを作成することもできます。

.weak 疑似命令がシンボルに使用されると、 **.globl**、 **.extern**、または **.comm** 疑似命令を同じシンボルに使用しても、シンボルの弱いバインディング・プロパティーには影響しません。

リンカーは、弱いバインディングを持つシンボルの重複定義を無視します。 グローバル・シンボルが 1 つのファイルでは弱いものではなく、他のファイルでは弱いものである場合、グローバル定義が使用され、弱い定義は無視されます。 すべての定義が弱い場合は、最初の弱い定義が使用されます。

弱いシンボルの可視性は、 *Visibility* パラメーターを使用して指定できます。

パラメーター

項目	説明
name	弱いバインディングを持つグローバル・シンボルとして Name を宣言します。「名前」には、*Qualname* を指定できます。 (*Qualname* は、制御セクションの *Name* および *StorageMappingClass* を指定します。)
可視性	シンボルの可視性を指定します。 有効な可視性の値は、*exported*、 *hidden*、 *internal*、および *protected* です。 シンボル可視性はリンカーによって使用されます。

例

次の例は、 .weak 疑似命令の使用法を示しています。

```
.weak foo[RW]
.csect data[RW]
```

関連概念

疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

.globl 疑似命令

.extern 疑似命令

シンボルの可視性

関連情報

ld

.xline 疑似命令

目的

行番号を表します。

構文

項目	説明
.x 行	*Number1*, *StringConstant*[, *Number2*]

説明

.xline 疑似命令は、アセンブラーに追加のファイルおよび行番号情報を提供します。 *Number2* パラメーターを使用すると、シンボリック・デバッガーが使用する **.bi** および **.ei** タイプ・エントリーを生成できます。この疑似命令は、通常、 M4 マクロ・プロセッサーによって挿入されます。

パラメーター

項目	説明
Number1	元のソース・ファイルの行番号を表します。
StringConstant (StringConstant)	元のソース・ファイルのファイル名を表します。
Number2	C_BINCL クラスと C_EINCL クラスを表します。これらのクラスは、インクルード・ファイルの開始と終了をそれぞれ示します。

例

以下の例は、 **.xline** 疑似命令の使用法を示しています。

```
.xline 1,"hello.c",108
.xline 2,"hello.c"
```

関連概念

疑似操作の概要
疑似命令は、アセンブラーに対する命令です。

付録 A メッセージ

この付録のメッセージは、エラー・メッセージまたは警告メッセージです。 各メッセージには 3 つのセクションがあります。

- メッセージ番号とメッセージ・テキスト
- メッセージの原因
- 取るべき処置

ファイル見出しに使用される一部のメッセージでは、アクション・セクションは省略されます。

項目	説明
1252 から 001	*< name>* は既に定義されています。 **原因** ユーザーは以前に定義タイプ・ステートメントで *name* を使用しており、それを再度定義しようとしていますが、これは許可されていません。 このメッセージが表示されるのは、以下の 3 つの場合です。 • ラベル名は、以前にソース・コードに定義されています。 • **.set** 疑似命令名は、以前にソース・コードに定義されています。 • **.lcomm** または **.comm** 疑似命令名は、以前にソース・コードに定義されています。 **アクション** 名前再定義エラーを訂正してください。
1252 から 002	ネスト・オーバーフローがあります。 一致する **.ef**、 **.eb**、または **.ei** 疑似操作を指定せずに、100 を超える **.function**、 **.bb**、または **.bi** 疑似操作を指定しないでください。 **原因** この構文エラー・メッセージは、デバッガー疑似命令が使用されている場合にのみ表示されます。 **.function**、 **.bb**、および **.bi** 疑似命令は、100 ポインターの制限サイズでスタックに保管されるポインターを生成します。 一致する **.ef** および **.eb** pseudo-ops が検出されずに、100 を超える **.function** および **.bb** pseudo-ops が検出された場合、この構文エラー・メッセージが表示されます。 **アクション** このネストを避けるために、コードを書き直してください。 **注:** デバッガー疑似命令は通常、プログラマーによってソース・コードに挿入されるのではなく、コンパイラーによって生成されます。
1252 から 003	**.set** オペランドが定義されていないか、順方向参照です。 **原因** **.set** 疑似命令の構文は、次のとおりです。 `.set name,expr` *expr* パラメーターには、整数、定義済みの名前 (ラベルで指定するか、 **.lcomm** または **.comm** 疑似命令で指定する)、または整数と名前の代数的な組み合わせを指定できます。 この構文エラー・メッセージは、 *expr* パラメーターが定義されていない場合に表示されます。 **アクション** *expr* パラメーターのすべての要素が **.set** ステートメントの前に定義されていることを確認してください。

項目	説明
1252 から 004	**.globl** 記号が正しくありません。 **.globl** 名が再配置可能式であることを確認してください。 **原因** **.globl** 名は再配置可能式でなければなりません。 この構文エラー・メッセージは、 **.globl** 疑似命令の *Name* パラメーターが再配置可能式でない場合に表示されます。 再配置とは、アドレスまたはロケーションがランタイム・ロケーションを反映するように変更でき、変更される可能性があるメモリー・ロケーションを表すエンティティーのことです。 再配置可能または再配置不能として定義されているエンティティーおよびシンボル名。 a. 再配置可能式には、ラベル名、 **.lcomm**、名前、 **.comm** 、および **.csect** 名が含まれます。 以下に、再配置不能項目と再配置不能式を示します。 • **.dsect** 名 • **.dsect** に含まれているラベル • ストレージ・クラスが BS または UC の csect 内に含まれるラベル • **.set** 名 • 絶対式 (定数または整数) • tocrelative (**.tc** ラベルまたは名前) • tocofrelative (**.tocof** ラベルまたは名前) • 不明 (アセンブラーのパス 2 で未定義) **アクション** **.globl** 疑似命令の *Name* パラメーターが再配置可能式であることを確認してください。 定義されていない場合、名前は外部名と見なされます。
1252 から 005	ストレージ・クラスが無効です。 csect 名にサポートされるストレージ・クラスを指定してください。 **原因** この構文エラー・メッセージは、 **.csect** 疑似命令で *Qualname* を指定するために使用されるストレージ・マッピング・クラス値が事前定義値の 1 つでない場合に表示されます。 **アクション** 事前定義ストレージ・マッピング・クラスのリストについては、 **.csect** 疑似命令を参照してください。 プログラム・エラーを訂正して, プログラムを再アセンブルし, リンクしてください。
1252 から 006	**ICSECT ERRTOK** の **ERRTOK** は認識されません。 この製品をどこで購入したかに応じて、サービス担当員または承認されたサプライヤーに連絡してください。 **原因** これは内部エラー・メッセージです。 **アクション** サービス担当員または承認された提供業者に連絡して, 問題を報告してください。

項目	説明
1252 から 007	位置合わせは絶対式でなければなりません。 **原因** この構文エラー・メッセージは、 **.csect** 疑似命令に対する誤ったオペランド (オプションの位置合わせパラメーター) が原因で出されます。 この位置合わせパラメーターは、絶対式 (整数) であるか、代数的に絶対式に解決する必要があります。 **アクション** 位置合わせパラメーターを訂正してから, プログラムを再びアセンブルしてリンクしてください。
1252 から 008	**.tocof** *name1* は有効ではありません。 *name1* が以前に定義されていないことを確認してください。 **原因** **.tocof** 疑似命令の *Name1* パラメーターは、現行モジュールの他の場所で定義されています。 **処置:** *name1* シンボルが **.tocof** 疑似命令にのみ定義されていることを確認してください。
1252 から 009	Begin ブロック、End ブロック、または **.function** 疑似命令が欠落しています。 **.bb** ステートメントごとに一致する **.eb** ステートメントがあること、および **.bf** ステートメントごとに一致する **.ef** ステートメントがあることを確認してください。 **原因** **.bb** 疑似命令ごとに一致する **.eb** 疑似命令がない場合、または **.bf** 疑似命令ごとに一致する **.ef** 疑似命令がない場合、このエラー・メッセージが表示されます。 **アクション** **.bb** 疑似命令ごとに一致する **.eb** 疑似命令があることを確認し、 **.bf** 疑似命令ごとに一致する **.ef** 疑似命令があることを確認します。
1252 から 010	**.tocof** *Name2* が有効ではありません。 *name2* が外部シンボルであることを確認してください。 **原因** **.tocof** 疑似命令の *Name2* パラメーターが正しく定義されていません。 **アクション** *Name2* パラメーターが外部で定義されている (**.extern** または **.globl** pseudo-op に現れる必要がある) こと、およびこのソース・モジュールでローカルに定義されていないことを確認してください。 **注:** *Name2* パラメーターがローカルに定義され、 **.extern** 疑似命令を使用して外部化される場合、このメッセージも表示されます。
1252 から 011	**.space** パラメーターは未定義です。 **原因** **.space** 疑似演算子に対する *Number* パラメーターは、正の絶対式でなければなりません。 このメッセージは、*Number* パラメーターに未定義のエレメント (後で定義される **.lcomm** または **.csect** 疑似命令のラベルまたは名前など) が含まれていることを示します。 **アクション** *Number* パラメーターが、絶対式、整数式、または絶対式に解決される代数式であることを確認してください。

項目	説明
1252 から 012	**.space** サイズは絶対式でなければなりません。 **原因** **.space** 疑似演算子に対する *Number* パラメーターは、正の絶対式でなければなりません。 このメッセージは、 *Number* パラメーターに非絶対エレメント (**.lcomm**、 **.comm**、または **.csect** pseudo-op のラベルまたは名前など) が含まれていることを示します。 **アクション** *Number* パラメーターに絶対式、または絶対式に解決される整数または代数式が指定されていることを確認してください。
1252 から 013	**.space** サイズは正の絶対式でなければなりません。 **原因** **.space** 疑似演算子に対する *Number* パラメーターは、正の絶対式でなければなりません。 このメッセージは、 *Number* パラメーターが負の絶対式に解決されることを示します。 **アクション** *Number* パラメーターが正の絶対式であることを確認してください。
1252 から 014	**.rename** 「名前」 シンボルは、ソース・コードで定義する必要があります。 **原因** **.rename** 疑似命令への *Name* パラメーターは、ソース・コード内のどこかに定義する必要があります。 このメッセージは、 *Name* パラメーターが定義されていないことを示します。 **アクション** *Name* パラメーターがソース・コードのどこかに定義されていることを確認してください。
1252 から 015	疑似命令パラメーターが定義されていません。 **原因** これは、 **.line**、 **.xline**、 **.bf**、 **.ef**、 **.bb**、および **.eb** pseudo-ops について表示される構文エラー・メッセージです。 これらの式には、解決しなければならない式オペランドがあります。 **アクション** 式が解決または定義されるようにソース・コードを変更してください。
1252 から 016	指定された命令コードまたは疑似命令が無効です。 サポートされている命令または疑似命令のみを使用してください。 **原因** ソース行の最初のエレメント (ラベルの後) は、命令または疑似命令として認識されません。 **アクション** サポートされている命令または疑似命令のみを使用してください。
1252 から 017	*args* パラメーター内の **ERRTOK** は有効ではありません。 この製品をどこで購入したかに応じて、サービス担当員または承認されたサプライヤーに連絡してください。 **原因** これは内部エラー・メッセージです。 **アクション** サービス担当員または承認された提供業者に連絡して, 問題を報告してください。

項目	説明
1252 から 018	**.tc** は、**.toc** スコープ内でのみ使用してください。 **.tc** ステートメントの前に **.toc** ステートメントを置きます。 **原因** **.tc** pseudo-op が有効なのは、 **.toc** pseudo-op の後、かつ **.csect** pseudo-op の前のみです。 それ以外の場合は、このメッセージが表示されます。 **アクション** **.toc** 疑似命令が **.tc** 疑似命令の前にあることを確認してください。 その他の疑似命令の前には、 **.csect** 疑似命令を付ける必要があります。 **.tc** 疑似命令の後に **.csect** 疑似命令を続ける必要はありません (ソース・ファイル内の最後の疑似命令の場合)。
1252 から 019	外部定義シンボルを **.byte** または **.vbyte** 式パラメーターとして指定しないでください。 **原因** **.byte** または **.vbyte** 疑似命令の *Expression* パラメーターに外部定義シンボルが含まれている場合 (シンボルは **.extern** または **.globl** 疑似命令で表示されます)、このメッセージが表示されます。 **アクション** **.byte** または **.vbyte** 疑似命令の *Expression* パラメーターに外部定義シンボルが含まれていないことを確認してください。
1252 から 020	外部定義シンボルを **.short** *Expression* パラメーターとして指定しないでください。 **原因** **.short** 疑似命令の *Expression* パラメーターに外部定義シンボルが含まれている場合 (シンボルが **.extern** または **.globl** pseudo-op に表示されます)、このメッセージが表示されます。 **アクション** **.short** 疑似命令の *Expression* パラメーターに、外部で定義されたシンボルが含まれていないことを確認してください。
1252 から 021	式は絶対式でなければなりません。 **原因** **.vbyte** pseudo-op の *Expression* パラメーターは、絶対式ではありません。 **アクション** 式が絶対式であることを確認してください。
1252 から 022	最初のパラメーターは、1 から 4 までの絶対式に解決する必要があります。 **原因** **.vbyte** pseudo-op の最初のパラメーターは、1 から 4 の範囲の絶対式でなければなりません。 **アクション** **.vbyte** 疑似命令の最初のパラメーターが、1 から 4 までの絶対式に解決されることを確認します。
1252 から 023	シンボル < *name*> が定義されていません。 **原因** ソース・プログラムで未定義の記号が使用されています。 **アクション** シンボルは、ラベルとして、または **.csect**、 **.通信**、 **.lcomm**、 **.dsect**、 **.セット**、 **.外部**、または **.グローバル** 疑似命令の 名前 パラメーターとして定義できます。 **as** コマンドの **-u** フラグは、このメッセージを抑止します。

項目	説明
1252 から 024	**.stab** ストリングには、：文字が含まれている必要があります。 **原因** **.stabx** pseudo-op の最初のパラメーターは、ストリング定数です。：(コロン) が含まれている必要があります。 それ以外の場合は、このメッセージが表示されます。 **アクション** **.stabx** 疑似命令の最初のパラメーターに：(コロン) が含まれていることを確認します。
1252 から 025	レジスター、基底レジスター、またはマスク・パラメーターが無効です。 レジスター番号は、マシン上のレジスターの数に制限されます。 **原因** 命令または疑似命令のオペランドとして使用されているレジスター番号が絶対値でないか、または値がアーキテクチャーの範囲外です。 **アクション** この値を指定するには、絶対式を使用する必要があります。 PowerPC ® および POWER ® ファミリーの場合、有効な値は 0 から 31 の範囲です。
1252-026	一時ファイルを作成できません。 **/tmp** ディレクトリーの許可を確認してください。 **原因** このメッセージは、 **/tmp** ファイル・システムの許可に問題があることを示しています。 **アクション** **/tmp** ディレクトリーの許可を確認してください。
1252 から 027	警告: ゼロによる位置合わせ: **.short** 疑似命令がハーフワード境界にありません。 **原因** この警告は、 **.short** 疑似命令がハーフワード境界にないことを示します。 アセンブラーは、ステートメントがハーフワード境界に位置合わせされるまで、現在位置にゼロを入れます。 **アクション** ユーザーが位置合わせを制御したい場合は、 **.short** 疑似命令の前に *Number* パラメーターを 1 に設定して **.align** 疑似命令を使用すると、同じ機能が実行されます。 **.short** 疑似命令の前に、 *Expression* パラメーターが 0 に設定された **.byte** pseudo-op は、アセンブラーが内部で実行する機能と同じ機能を実行します。
1252 から 028	**/tmp** ディレクトリーの中間結果ファイルを再オープンできません。 **/tmp** ファイルシステムのサイズがファイルを保管するのに十分であること、およびファイルシステムが損傷していないことを確認してください。 **原因** このメッセージは、中間ファイルを閉じてからファイルを再度開くときにシステム問題が発生したことを示します。 **アクション** 通常、中間ファイルは **/tmp** ファイル・システムにあります。 **/tmp** ファイル・システムのスペースを調べて、中間ファイルを入れるのに十分な大きさがあるかどうかを確認してください。

項目	説明
1252 から 029	現在使用可能なメモリーが不足しています。 テキスト・セクションとデータ・セクションを割り振ることができません。 後でやり直すか, あるいはソフトウェア障害報告手順を使用してください。 **原因** これはメモリー管理の問題です。 これは、テキストおよびデータ・セクションの割り振り中に **malloc** 関数が呼び出されたときに報告されます。 十分なメイン・メモリーがないか、メモリー・ポインターが破壊されています。 **アクション** 後でもう一度試してください。 問題が引き続き発生する場合は、メモリーのアプリケーション・ロードを確認するか、システム管理者に連絡してください。
1252 から 030	ファイル <*filename*>を作成できません。 パス名と許可を確認してください。 **原因** このメッセージは、アセンブラーが出力ファイル (オブジェクト・ファイル) を作成できないことを示します。 **as** コマンドの **-o** フラグを使用すると、指定した場所にオブジェクト・ファイルが作成されます。 **-o** フラグを使用しないと、デフォルト名 **a.out** のオブジェクト・ファイルが現行ディレクトリーに作成されます。 ディレクトリーの許可に問題がある場合、またはパス名が無効である場合は、このメッセージが表示されます。 **アクション** パス名と許可を確認してください。
1252 から 031	現在使用可能なメモリーが不足しています。 ESD セクションを割り振ることができない。 後でやり直すか, あるいはソフトウェア障害報告手順を使用してください。 **原因** これはメモリー管理の問題です。 これは、ESD セクションの割り振り中に **malloc** 関数が呼び出されたときに報告されます。 十分なメイン・メモリーがないか、メモリー・ポインターが破壊されています。 **アクション** 後でもう一度試してください。 問題が引き続き発生する場合は、メモリーのアプリケーション・ロードを確認するか、システム管理者に連絡してください。
1252 から 032	現在使用可能なメモリーが不足しています。 RLD セクションを割り振ることができません。 後でやり直すか, あるいはソフトウェア障害報告手順を使用してください。 **原因** これはメモリー管理の問題です。 これは、RLD セクションの割り振り中に **malloc** 関数が呼び出されたときに報告されます。 十分なメイン・メモリーがないか、メモリー・ポインターが破壊されています。 **アクション** 後でもう一度試してください。 問題が引き続き発生する場合は、メモリーのアプリケーション・ロードを確認するか、システム管理者に連絡してください。
1252 から 033	現在使用可能なメモリーが不足しています。 ストリング・セクションを割り振ることができません。 後でやり直すか, あるいはソフトウェア障害報告手順を使用してください。 **原因** これはメモリー管理の問題です。 これは、ストリング・セクションの割り振り中に **malloc** 関数が呼び出されたときに報告されます。 十分なメイン・メモリーがないか、メモリー・ポインターが破壊されています。 **アクション** 後でもう一度試してください。 問題が解決しない場合は、メモリーのアプリケーション・ロードを確認するか、システム管理者に連絡してください。

項目	説明
1252 から 034	現在使用可能なメモリーが不足しています。 行番号セクションを割り振ることができません。 後でやり直すか, あるいはソフトウェア障害報告手順を使用してください。 **原因** これはメモリー管理の問題です。 これは、行番号セクションの割り振り中に **malloc** 関数が呼び出されたときに報告されます。 十分なメイン・メモリーがないか、メモリー・ポインターが破壊されています。 **アクション** 後でもう一度試してください。 問題が引き続き発生する場合は、メモリーのアプリケーション・ロードを確認するか、システム管理者に連絡してください。
1252-035 から 1252-037	廃止されたメッセージ。
1252 から 038	ファイル <*filename*>をオープンできません。 パス名と許可を確認してください。 **原因** 指定されたソース・ファイルが見つからないか、読み取り権限がありません。 *listfile* または *xcrossfile* に書き込み権限がないか、指定されたパスが存在しません。 **アクション** パス名と読み取り/書き込み許可を確認してください。
1252 から 039	現在使用されていません。
1252 から 040	指定された式が無効です。 すべてのシンボルが定義されていることを確認してください。 再配置に関する算術式で使用されているシンボルの規則を確認してください。 **原因** 示された式は、絶対式、再配置可能式、外部式、toc 相対式、tocof 記号、または制限付き外部式に解決されません。 **アクション** すべてのシンボルが定義されていることを確認してください。 また、演算式でシンボルを使用できる再配置に関する規則もいくつかあります。 詳しくは、37 ページの『式』を参照してください。
1252 から 041	算術除算中に値を 0 で除算することはできません。 **原因** 算術除算の間、除数はゼロです。 **アクション** 値がゼロで除算されていないことを確認してください。
1252 から 042	内部算術演算子が認識されていません。 この製品をどこで購入したかに応じて、サービス担当員または承認されたサプライヤーに連絡してください。 **原因** これは内部エラー・メッセージです。 **アクション** サービス担当員または承認された提供業者に連絡して, 問題を報告してください。

項目	説明
1252 から 043	再配置可能アセンブラー式が正しくありません。 式を結合できることを確認してください。 **原因** このメッセージは、式の無効な算術組み合わせが使用された場合に表示されます。 **アクション** 正しい算術の組み合わせが使用されていることを確認してください。 式の有効な算術組み合わせの特定の規則については、37 ページの『式』 を参照してください。
1252 から 044	指定されたソース文字 < *char*> は、使用されるコマンド・コンテキストでは意味を持ちません。 **原因** ソース文字は、それが使用されるコンテキストでは意味がありません。 例えば、`.long 3@1` の場合、 `@` は算術演算子でも整数桁でもなく、このコンテキストでは意味がありません。 **アクション** すべての文字が有効であり、使用されているコンテキストで意味があることを確認してください。
1252 から 045	リスト・ファイル < *filename*>をオープンできません。 ファイル・システムの品質を確認してください。 **原因** これは、アセンブラーのパス 2 の間に発生し、ファイル・システムの問題または元のリスト・ファイルのクローズの問題の可能性を示します。 **アクション** ファイル・パス名に従ってファイル・システムを確認してください。
1252 から 046	現在使用されていません。
1252 から 047	ネスト・アンダーフローがあります。 **.function**、 **.bi**、または **.bb** pseudo-ops が欠落していないか確認します。 **原因** この構文エラー・メッセージは、デバッガー疑似命令が使用されている場合にのみ表示されます。 **.function**、 **.bb**、および **.bi** 疑似命令は、100 ポインターの制限サイズでスタックに保管されるポインターを生成します。 **.ef**、**.eb**、および **.ei** pseudo-ops は、これらのポインターをスタックから除去します。 検出された **.ef**、 **.eb**、および **.ei** 疑似命令の数が、スタック上のポインターの数より多い場合、このメッセージが表示されます。 **アクション** この問題を回避するために、コードを書き直してください。
1252 から 048	外部シンボルの作成時に無効なシンボル・タイプを検出しました。 この製品をどこで購入したかに応じて、サービス担当員または承認されたサプライヤーに連絡してください。 **原因** これは内部エラー・メッセージです。 **アクション** サービス担当員または承認された提供業者に連絡して, 問題を報告してください。

項目	説明
1252 から 049	すべてのハッシュ・ストリングを格納するための十分なメモリーがありません。 この製品をどこで購入したかに応じて、サービス担当員または承認されたサプライヤーに連絡してください。 **原因** これは内部エラー・メッセージです。 **アクション** サービス担当員または承認された提供業者に連絡して, 問題を報告してください。
1252 から 050	現在使用可能なメモリーが不足しています。 デバッグ・セクションを割り振ることができません。 後でやり直すか, あるいはソフトウェア障害報告手順を使用してください。 **原因** これはメモリー管理の問題です。 これは、デバッグ・セクションの割り振り中に **malloc** 関数が呼び出されたときに報告されます。 十分なメイン・メモリーがないか、メモリー・ポインターが破壊されています。 **アクション** 後でもう一度試してください。 問題が引き続き発生する場合は、メモリーのアプリケーション・ロードを確認するか、システム管理者に連絡してください。
1252 から 051	無効な *Number=< number>* の *sclass* タイプがあります。 この製品をどこで購入したかに応じて、サービス担当員または承認されたサプライヤーに連絡してください。 **原因** これは内部エラー・メッセージです。 **アクション** サービス担当員または承認された提供業者に連絡して, 問題を報告してください。
1252 から 052	指定する **.align** パラメーターは、0 から 12 までの絶対値でなければなりません。 **原因** **.align** 疑似命令の *Number* パラメーターが絶対値でないか、値が 0 から 12 の範囲内にありません。 **アクション** *Number* パラメーターが 0 から 12 の範囲の絶対式に解決されることを確認します。
1252 から 053	**.org** パラメーターの値を、現在の csect に含まれるまで変更します。 **原因** **.org** 疑似命令のパラメーターの値により、ロケーション・カウンターは現在の csect の外側に置かれます。 **アクション** 最初のパラメーターの値が以下の基準を満たしていることを確認します。 正の値 (0 を含む) でなければなりません。 結果として、現在の csect に含まれているアドレスになる必要があります。 外部 (E_EXT) 式または再配置可能 (E_REL) 式でなければなりません。

項目	説明
2363-054	**.using** のレジスター・パラメーターは絶対値でなければならず、現行マシン上のレジスターを表すものでなければなりません。 **原因** **.using** pseudo-op の 2 番目のパラメーターが絶対値を表していないか、値が有効なレジスター番号の範囲外です。 **アクション** PowerPC ® および POWER ® ファミリーの場合は、値が絶対値であり、0 から 31 の範囲内であることを確認してください。
1252 から 055	**.using** に、有効でない基底アドレスがあります。 基底アドレスは再配置可能式でなければなりません。 **原因** **.using** 疑似命令の最初のパラメーターが再配置可能式ではありません。 **アクション** 最初のパラメーターが再配置可能であることを確認してください。 最初のパラメーターは、TOC 相対ラベル、再配置可能なラベル/名前 (再配置可能 = REL)、または現在のアセンブリー・ソース内で csect 名 /TOC 項目として定義されている外部シンボルにすることができます。
1252 から 056	TOC セクションの先頭のみを参照する **.using** 引数を指定します。 引数は、TOC セクション内に含まれるロケーションを参照できません。 **原因** **.using** 疑似命令の最初のパラメーターは TOC 相対式ですが、TOC の先頭をポイントしていません。 **アクション** TOC が TOC 相対の場合は、最初のパラメーターが TOC の先頭を記述していることを確認してください。
1252 から 057	外部式が正しくありません。 シンボルを外部シンボルにすることはできません。 シンボルが外部シンボルである場合は、 **.toc** または **.csect** エントリーを使用して、そのシンボルをアセンブリー内に定義する必要があります。 **原因** **.using** 疑似命令の最初のパラメーターには、csect 名または TOC 項目以外の外部式が使用されます。 **アクション** シンボルが外部でない (**.extern** 疑似命令によって指定されていない) か、または TOC エントリーまたは csect エントリーを使用してアセンブリー・ソース内で定義されていることを確認してください。
1252 から 058	警告: ラベル < *name*> は csect < *csectname*>に位置合わせされます。 **原因** ラベルが、 **.csect** 疑似命令の同じ行にある場合。 この警告は、 **as** コマンドの **-w** フラグが使用されたときに報告されます。 このメッセージは、ラベルが意図したとおりに位置合わせされていない可能性があることを示します。 ラベルが csect の先頭を指す必要がある場合、ラベルは csect 内の **.csect** 疑似命令の隣の最初の行に含まれていなければなりません。 **アクション** ラベルの意図を評価します。

項目	説明
1252 から 059	**.drop** のレジスターは、有効なレジスター番号である絶対値でなければなりません。 **原因** **.drop** pseudo-op のパラメーターが絶対値ではないか、値が有効なレジスター番号の範囲内にありません。 **アクション** 有効なレジスターを示すには、絶対値を使用します。 PowerPC ® および POWER ® ファミリーの場合、有効なレジスター番号は 0 から 31 の範囲です。
1252 から 060	**.drop** のレジスターは使用中ではありません。 この行を削除するか、この **.drop** 行の前に **.using** 行を挿入します。 **原因** このメッセージは、**.drop** 疑似命令のパラメーターによって表されるレジスターが、前の **.using** ステートメントで使用されたことがないことを示します。 **アクション** **.drop** 疑似命令を削除するか、この **.drop** 疑似命令の前に使用する必要があった **.using** 疑似命令を挿入します。
1252 から 061	**.toc** 有効範囲内のステートメントが無効です。 **.tc** pseudo-op を使用して、 **.toc** スコープ内でエントリーを定義します。 **原因** **.tc** 疑似命令以外のステートメントが **.toc** 有効範囲内で使用されると、このメッセージが表示されます。 **アクション** **.tc** 疑似命令は、 **.toc** 有効範囲内にのみ配置してください。
1252 から 062	位置合わせは 0 から 31 の値でなければなりません。 **原因** **.csect** パラメーターのオプションの 2 番目のパラメーター (*Number*) は、現在の csect の上部の位置合わせを定義します。 位置合わせは 0 から 31 の範囲でなければなりません。 それ以外の場合は、このメッセージが表示されます。 **アクション** 2 番目のパラメーターが有効な範囲内にあることを確認してください。
1252 から 063	廃止されたメッセージ。
1252 から 064	**.comm** サイズは絶対式でなければなりません。 **原因** **.comm** 疑似命令の 2 番目のパラメーターは、絶対式でなければなりません。 それ以外の場合は、このメッセージが表示されます。 **アクション** 2 番目のパラメーターが絶対式であることを確認してください。
1252 から 065	現在使用されていません。

項目	説明
1252 から 066	現在使用可能なメモリーが不足しています。 typchk セクションを割り振ることができません。 後でやり直すか, あるいはソフトウェア障害報告手順を使用してください。 **原因** これはメモリー管理の問題です。 これは、デバッグ・セクションの割り振り中に **malloc** 関数が呼び出されたときに報告されます。 十分なメイン・メモリーがないか、メモリー・ポインターが破壊されています。 **アクション** 後でもう一度試してください。 問題が引き続き発生する場合は、メモリーのアプリケーション・ロードを確認するか、システム管理者に連絡してください。
1252 から 067	指定された共通ストレージ・クラスが無効です。 この製品をどこで購入したかに応じて、サービス担当員または承認されたサプライヤーに連絡してください。 **原因** これは内部エラー・メッセージです。 **アクション** サービス担当員または承認された提供業者に連絡して, 問題を報告してください。
1252 から 068	**.hash** ストリングは、シンボル *name* に既に設定されています。 これが、シンボル名に関連付けられた唯一の **.hash** ステートメントであることを確認してください。 **原因** **.hash** 疑似命令の *Name* パラメーターには、前の **.hash** ステートメントのストリング値が既に割り当てられています。 **アクション** *Name* パラメーターが **.hash** 疑似命令ごとに固有であることを確認してください。
1252 から 069	ハッシュ・ストリング内の文字 < *char*> は無効です。 ストリング内の文字は、セット [0-9A-Fa-f] 内になければなりません。 **原因** ハッシュ・ストリング値 (**.hash** 疑似命令の 2 番目のパラメーター) の文字は、セット [0-9A-Fa-f] に含まれている必要があります。 文字は 16 進ハッシュ・コードを表します。 それ以外の場合は、このメッセージが表示されます。 **アクション** *StringConstant* パラメーターで指定した文字がこのセットに含まれていることを確認してください。
1252 から 070	ハッシュ値に指定されたシンボルまたはシンボル・タイプが無効です。 **原因** **.hash** 疑似命令の *Name* パラメーターが定義済みの外部シンボルでない場合、このメッセージが表示されます。 **注:** 1. このメッセージは、**as** コマンドの **-u** フラグを使用することによって抑止できます。 2. 定義済みの内部シンボル (例えば、ローカル・ラベル) によっても、このメッセージが表示されることがあります。 **アクション** **as** コマンドの **-u** フラグを使用するか、**.extern** または **.globl** pseudo-op を使用して、 *Name* パラメーターを外部シンボルとして定義します。

項目	説明
1252-071 および 1252-072	現在使用されていません。
1252 から 073	現在使用可能なメモリーが不足しています。 メモリー内にセグメントを割り振ることができません。 後でやり直すか, あるいはソフトウェア障害報告手順を使用してください。 **原因** これは、 **malloc**、 **realloc**、または **calloc** の問題を示します。 以下の問題により、このタイプのエラーが生成される可能性があります。 • 割り振るためのメイン・メモリーが不足しています • メモリー・ポインターの破損 • ファイル・システムの破損 **アクション** ファイル・システムとメモリーの状況を確認してください。
1252 から 074	疑似命令がテキスト・セクション内にありません。 **.function**、 **.bf**、および **.ef** pseudo-ops は、RO、PR、XO、SV、DB、GL、TI、または TB のいずれかのストレージ・クラスを持つ csect 内に含まれている必要があります。 **原因** **.function**、 **.bf** 、および **.ef** pseudo-ops が、ストレージ・マッピング・クラスが RO、PR、XO、SV、DB、GL、TI、または TB の csect 内にない場合、この構文エラー・メッセージが表示されます。 **アクション** **.function**、 **.bf**、および **.ef** pseudo-ops が、テキスト csect のスコープ内にあることを確認してください。
1252 から 075	指定されたパラメーターの数が無効です。 **原因** これは構文エラー・メッセージです。 命令に指定されたパラメーターの数が正しくありません。 **アクション** この命令に正しい数のパラメーターが指定されていることを確認してください。
1252 から 076	**.line** pseudo-op は、テキストまたはデータ **.csect** 内に含まれていなければなりません。 **原因** これは構文エラー・メッセージです。 **.line** 疑似命令は、テキスト・セクションまたはデータ・セクション内になければなりません。 **.line** pseudo-op が **.dsect** pseudo-op に含まれている場合、またはストレージ・マッピング・クラスが BS または UC の **.csect** pseudo-op に含まれている場合、このエラーが表示されます。 **アクション** **.line** 疑似命令が .dsect; の有効範囲内に含まれていないこと、またはストレージ・マッピング・クラスが BS または UC の **.csect** 疑似命令に含まれていないことを確認してください。

項目	説明
1252 から 077	ファイル・テーブルがいっぱいです。 単一のアセンブリー・ソース・ファイルに 99 個を超えるファイルを含めないでください。 **原因** **.xline** pseudo-op は、番号とともにファイル名を示します。 これらの疑似命令は、**m4** コマンドの **-l** オプションで生成されます。 このオプションには最大 99 個のファイルを組み込むことができます。 99 を超えるファイルが組み込まれている場合には, このメッセージが表示されます。 **アクション** **m4** コマンドが、単一のアセンブリー・ソース・ファイルに 99 個を超えるファイルを含んでいないことを確認してください。
1252 から 078	< *positionnumber*> で始まるビット・マスク・パラメーターが無効です。 **原因** これは構文エラー・メッセージです。 左に回転する命令には、 **rl***xx RA*、*RS*、*SH*、*MB*、*ME*、または **rl***xx RA*、*RS*、*SH*、*BM* の 2 つの入力オペランド形式があります。 このメッセージは、2 番目の形式が使用された場合にのみ表示されます。 *BM* パラメーターは, この命令のマスクを指定します。 特定の規則に従って構成する必要があります。 それ以外の場合は、このメッセージが表示されます。 *BM* パラメーターの構成については、 99 ページの『32 ビット固定小数点回転命令およびシフト命令の拡張ニーモニック』 を参照してください。 **アクション** ビット・マスク値を訂正してください。
1252 から 079	RLD のカウント時に無効なタイプを検出しました。 この製品をどこで購入したかに応じて、サービス担当員または承認されたサプライヤーに連絡してください。 **原因** これは内部エラー・メッセージです。 **アクション** サービス担当員または承認された提供業者に連絡して, 問題を報告してください。
1252 から 080	指定されたブランチ・ターゲットはフルワード境界上になければなりません。 **原因** これは構文エラー・メッセージです。 分岐命令には、プログラム・ロジックがジャンプするターゲットまたは位置があります。 これらのターゲット・アドレスは、フルワード境界上になければなりません。 **アクション** 分岐ターゲットがフルワード・アドレス (0、4、8、または c で終わるアドレス) にあることを確認してください。 アセンブラー・リストは、ロケーション・カウンター・アドレスを示します。 これは、このタイプの問題を追跡する場合に役立ちます。
1252 から 081	命令が正しく位置合わせされていません。 命令にはマシン固有の位置合わせが必要です。 **原因** PowerPC ® および POWER ® ファミリーでは、位置合わせはフルワードでなければなりません。 このメッセージが表示された場合は、現行の命令より前の命令または疑似命令が、フルワードにならないアドレスになるようにロケーション・カウンターを変更した可能性があります。 **アクション** 命令がフルワード・アドレスにあることを確認してください。

項目	説明
1252 から 082	命令にはさらにパラメーターを使用してください。 **原因** 各命令には、一連の数の引数が渡される必要があります。 使用する引数が少なすぎると、このエラーが表示されます。 **アクション** 命令定義を調べて、この命令に必要な引数の数を確認してください。
1252 から 083	命令に使用するパラメーターの数を減らしてください。 **原因** 各命令には、一連の数の引数が渡される必要があります。 使用されている引数が多すぎると、このエラーが表示されます。 **アクション** 命令定義を調べて、この命令に必要な引数の数を確認してください。
1252-084 および 1252-085	廃止されたメッセージ。
1252 から 086	分岐命令のターゲットは再配置可能式または外部式でなければなりません。 **原因** 絶対式ターゲットは、分岐命令で再配置可能式または外部式が受け入れ可能な場合に使用されます。 **アクション** 現行の分岐命令を絶対分岐命令で置き換えるか, あるいは絶対式ターゲットを再配置可能ターゲットで置き換えてください。
1252 から 087	分岐命令のターゲットは再配置可能式または外部式でなければなりません。 **原因** これは構文エラー・メッセージです。 分岐命令のターゲットは、再配置可能または外部のいずれかでなければなりません。 **アクション** このブランチ命令のターゲットが再配置可能または外部のいずれかであることを確認してください。 再配置可能式には、ラベル名、 **.lcomm** 名、 **.comm** 名、および **.csect** 名が含まれます。 再配置とは、アドレスまたはロケーションがランタイム・ロケーションを反映するように変更でき、変更される可能性があるメモリー・ロケーションを表すエンティティーのことです。 再配置可能または再配置不可として定義されているエンティティーおよびシンボル名については、 37 ページの『式』 で説明しています。
1252 から 088	ブランチ・アドレスが範囲外です。 ターゲット・アドレスは、ブランチ・アドレス値のビット・サイズを表す命令の能力を超えることはできません。 **原因** これは構文エラー・メッセージです。 ブランチ命令は、ターゲット・アドレスのサイズを 26 ビット、16 ビット、およびその他の命令固有のサイズに制限します。 ターゲット・アドレス値が命令固有の制限スペースで表現できない場合、このメッセージが表示されます。 **アクション** ターゲット・アドレス値が、ターゲット・アドレスを表す命令の能力 (ビット・サイズ) を超えないようにしてください。
1252-089 から 1252-098	廃止されたメッセージ。

項目	説明
1252 から 099	指定された変位が無効です。 命令変位は、再配置可能、絶対、または外部のいずれかでなければなりません。 **原因** これは構文エラー・メッセージです。 命令変位は、マシン・プラットフォームに応じて、再配置可能、絶対、XTY_SD または STY_CM シンボル・タイプ (csect または共通ブロック名) を持つ外部、または TOC 相対 (ただし負の TOC 相対ではない) のいずれかでなければなりません。 **アクション** 変位がこの命令に対して有効であることを確認してください。
1252 から 100	変位値または指定された汎用レジスターの内容、あるいはその両方が、有効なアドレスを生成しません。 **原因** 無効な *d*(*r*) オペランドを示します。 *d* または *r* のいずれかが欠落しています。 **アクション** 基本/変位オペランドが正しく形成されていることを確認してください。 プログラミング・エラーを訂正してから、プログラムを再アセンブルしてリンクしてください。 **注:** *d* または *r* を指定する必要がない場合は、0 を指定する必要があります。
1252-101 および 1252-102	廃止されたメッセージ。
1252 から 103	指定された命令はこのマシンではサポートされていません。 **原因** これは内部エラー・メッセージです。 **アクション** サービス担当員または承認された提供業者に連絡して, 問題を報告してください。
1252 から 104	*< parm #>* パラメーターは絶対パスでなければなりません。 **原因** 示されたパラメーターは絶対 (再配置不能、非外部) でなければなりません。 **アクション** 説明の構文については、特定の説明記事を参照してください。
1252 から 105	廃止されたメッセージ。
1252 から 106	現在は使用されていません。

項目	説明
1252 から 107	パラメーター < *parm #*> は、特定の命令の範囲内でなければなりません。 **原因** このエラーは、以下の状況で発生します。 • パラメーター値が下限と上限の範囲内にありません。 • SPR エンコードのパラメーター値が未定義です。 • 循環指示およびシフト指示のパラメーター値は、この制限を超えています。 **アクション** インストラクションの定義については、特定のインストラクションの記事を参照してください。 SPR エンコードのリストについては、94 ページの『特殊目的のレジスターから、または特殊目的のレジスターへの移動に関する拡張ニーモニック』 を参照してください。 通常、アセンブリー・モードが **com**、 **pwr**、または **pwr2** の場合、SPR 範囲は 0 から 31 です。 それ以外の場合、SPR の範囲は 0 から 1023 です。 制約事項については、 479 ページの『.csect 疑似演算子』 を参照してください。 ソース・コードを変更してから, プログラムを再びアセンブルしてリンクしてください。
1252 から 108	警告: ラベル < *name*> の位置合わせが無効です。 ラベルにはマシン固有の位置合わせが必要です。 **原因** ラベルがブランチのサブジェクトになるように適切に位置合わせされていないことを示します。 つまり、ラベルはフルワード・アドレス (0、4、8、または c で終わるアドレス) に位置合わせされません。 **アクション** 位置合わせを制御するために、ラベルの前の **.align** 疑似命令は位置合わせ機能を実行します。 また、ラベルの前にパラメーター 0 が指定された **.byte** 疑似命令、またはパラメーター 0 が指定された **.short** 疑似命令は、ラベルの位置合わせをシフトします。
1252 から 109	警告: ゼロによる位置合わせ: **.long** 疑似演算子がフルワード境界上にありません。 **原因** フルワード内部アドレス (0、4、8、または c で終わるアドレス) で適切に位置合わせされていない **.long** 疑似命令が存在することを示します。 アセンブラーは、ステートメントを正しく位置合わせするためにゼロを生成します。 **アクション** 位置合わせを制御するために、 **.long** 疑似命令の前にパラメーター 2 を指定した **.align** 疑似命令が位置合わせを実行します。 また、パラメーター 0 を指定した **.byte** 疑似命令、または **.long** 疑似命令の前にパラメーター 0 を指定した **.short** 疑似命令が位置合わせを実行します。
1252 から 110	警告: プログラム csect でゼロに位置合わせしています。 **原因** **.align** pseudo-op がタイプ [PR] または [GL] の .csect 内で使用され、 **.align** pseudo-op がフルワード・アドレス上にない場合 (PowerPC ® および POWER ® ファミリーの場合、すべての命令の長さは 4 バイトであり、フルワードで位置合わせされています)、アセンブラーは埋め込みゼロによって位置合わせを行い、この警告メッセージが表示されます。 また、他の疑似命令ステートメントでフルワードの位置合わせが行われた場合にも表示されます。 **アクション** 位置合わせがフルワード上にない理由を調べてください。 これは、疑似命令または命令が間違った場所にあることを示している可能性があります。

項目	説明
1252 から 111	警告: CSECT の位置合わせが変更されました。 位置合わせを変更するには、前の **.csect** ステートメントを検査します。 **原因** csect の先頭は、デフォルト値 (2、フルワード) または *Number* パラメーターに従って位置合わせされます。 この警告は、csect の作成時に有効であった位置合わせが、後でソース・コードで変更されたことを示します。 csect の位置合わせの変更は、以下のいずれかによって起こる可能性があります。 • **.csect** 疑似操作の *Number* パラメーターに、同じ *Qualname* を持つ以前の **.csect** 疑似操作より大きい値が指定されています。 • **.align** 疑似命令の *Number* パラメーターに、現在の csect 位置合わせより大きい値が指定されています。 • **.double** 疑似命令が使用されます。これにより、位置合わせが 3 に増加します。 現在の csect 位置合わせが 3 より小さい場合、この警告が報告されます。 **アクション** このメッセージは、ユーザーの意図によって、問題を示している場合と示していない場合があります。 問題が発生したかどうかを評価します。
1252 から 112	警告: < のインスタンス。 フォーマット> 命令は、このマシンではサポートされていません。 **原因** これは内部エラー・メッセージです。 **アクション** サービス担当員または承認された提供業者に連絡して, 問題を報告してください。
1252-113 および 1252-114	廃止されたメッセージ。
1252 から 115	ソートが状況 < *number*>で失敗しました。 システム・ソート・コマンドの状態を調べるか, あるいはソフトウェア障害報告手続きを使用してください。 **原因** **as** コマンドの **-x** フラグがコマンド行から使用されると、システム・ソート・ルーチンが呼び出されます。 この呼び出しが成功しない場合は、このメッセージが表示されます。 ソート・ユーティリティーが使用できないか、システム問題が発生しました。 **アクション** システム・ソート・コマンドの状態を検査するか、システム自体を検査するか (**fsck** コマンドを使用)、またはソフトウェア障害報告手続きを行ってください。
1252 から 116	< *name*>からのシステム・エラーがあります。 システム・ソート・コマンドの状態を調べるか, あるいはソフトウェア障害報告手続きを使用してください。 **原因** *name* にはソート・コマンドがあります。 **as** コマンドの **-x** フラグがコマンド行から使用されると、システム・ソート・ルーチンが呼び出されます。 アセンブラーは, ソート・ユーティリティーを呼び出すプロセスを fork します。 この fork がソート・ルーチンの実行に失敗すると、このメッセージが表示されます。 ソート・ユーティリティーが使用できないか、システム問題が発生しました。 **アクション** システム・ソート・コマンドの状態を検査するか、システム自体を検査するか (**fsck** コマンドを使用)、またはソフトウェア障害報告手続きを行ってください。

項目	説明
1252 から 117	"アセンブラー:" **原因** この行は、アセンブリー・プログラムであることを示すヘッダーを標準エラー出力に定義します。
1252 から 118	"行 < *number*>" **原因** *number* は、エラーまたは警告が存在する行番号を示します。 ソース・プログラムをアセンブルするとき、このメッセージは画面上のエラー/警告メッセージの前に表示されます。 このメッセージは、アセンブラー・リスト・ファイル内のエラー/警告メッセージの前にも印刷されます。
1252 から 119	"xref" **原因** このメッセージは、シンボル相互参照ファイルのファイル名のデフォルトの接尾部拡張子を定義します。
1252 から 120	". lst" **原因** このメッセージは、アセンブラー・リスト・ファイルのファイル名のデフォルトの接尾部拡張子を定義します。
1252 から 121	「SYMBOL FILE CSECT LINENO」 **原因** この行は、シンボル相互参照ファイルの見出しを定義します。
1252-122 から 1252-123	アセンブラー・リスト・ファイルで使用されるいくつかのフォーマットを定義します。
1252 から 124	廃止。1252-179 に置き換えられました。
1252-125 から 1252-132	アセンブラー・リスト・ファイルのスペースまたはフォーマットを定義します。
1252-133 から 1252-134	出力の数値と名前の形式を定義します。
1252 から 135	リスト・ファイルで使用される 8 つのスペースを定義します。
1252 から 136	リスト・ファイルで使用されるフォーマットを定義します。
1252-137 から 1252-140	数値の出力の形式を設定します。
1252 から 141	収集ポインターにエラーがあります。 ソフトウェア障害報告手続きを行ってください。 **原因** これは内部エラー・メッセージです。 **アクション** サービス担当員または承認された提供業者に連絡して, 問題を報告してください。

項目	説明
1252 から 142	構文エラー **原因** アセンブリー処理でエラーが発生し、そのエラーがメッセージ・カタログに定義されていない場合は、この汎用エラー・メッセージが使用されます。 このメッセージは、疑似命令と命令の両方を扱います。 したがって、使用法ステートメントは役に立ちません。 **アクション** 意図およびソース行の構造を判別してから、特定の指示項目を参照してソース行を訂正してください。
1252 から 143	**.function** *Size* は絶対式でなければなりません。 **原因** **.function** 疑似命令の *Size* パラメーターは、関数のサイズを表します。 絶対式でなければなりません。 **アクション** *Size* パラメーターを変更してから、プログラムを再アセンブルしてリンクしてください。
1252 から 144	警告: BS または UC ストレージ・クラスの < *name*> csect に初期化されたデータはすべて無視されますが、長さを設定するために必要です。 **原因** ストレージ・マッピング・クラスが BS または UC の csect 内のステートメントが csect の長さの計算に使用され、データの初期化には使用されないことを示します。 **アクション** なし。
1252-145 および 1252-146	廃止。1252-180 および 1252-181 に置き換えられました。
1252 から 147	無効な **.machine** アセンブリー・モード・オペランド: < *name*> **原因** **.machine** 疑似命令は、アセンブリー・モード値を示すためにソース・プログラムで使用されます。 このメッセージは、未定義の値が使用されたことを示します。 **アクション** 定義されているアセンブリー・モード値のリストについては、「501 ページの『.machine 疑似命令』」を参照してください。
1252 から 148	無効な **.source** 言語 ID オペランド: < *name*> **原因** **.source** 疑似命令は、ソース言語タイプ (C、FORTRAN など) を示します。 このメッセージは、無効なソース言語タイプが使用されたことを示します。 **アクション** 定義済み言語タイプのリストについては、 **.source** 疑似命令を参照してください。

項目	説明
1252 から 149	命令 *<name1>* は、現行のアセンブリー・モード *<name2>*では実装されません。 **原因** POWER ® ファミリー /PowerPC ® 交差領域にない命令は、特定のアセンブリー・モードでのみ実装されます。 このメッセージは、ソース・プログラムの命令が、示されているアセンブリー・モードではサポートされていないことを示します。 **アクション** 別のアセンブリー・モードまたは別の命令を使用してください。
1252 から 150	*value* の第 1 オペランドの値は、PowerPC ® では無効です。6、7 14、15、または 20 より大きい *BO* フィールドは無効です。 **原因** 分岐条件付き命令では、第 1 オペランドは B0 フィールドです。 入力値が必要な値の範囲外である場合は、このメッセージが表示されます。 **アクション** 入力オペランドの正しい値を見つけるには、BO フィールド・エンコード情報について 1 ページの『AIX® アセンブラーの機能』 を参照してください。
1252 から 151	この命令形式は、PowerPC ® では無効です。 オペランド 2 で使用されるレジスターは、ゼロであってはなりません。また、オペランド 1 で使用されるレジスターと同じであってはなりません。 **原因** 更新形式の固定小数点ロード命令では、PowerPC ® は、 *RA* オペランドがゼロではなく、RT と等しくないことを必要とします。 これらの要件に違反すると、このメッセージが表示されます。 **アクション** これらの説明のリストについては、 1 ページの『AIX® アセンブラーの機能』 を参照してください。また、これらの説明の構文と制約事項については、説明記事を参照してください。 ソース・コードを変更してから, プログラムを再びアセンブルしてリンクしてください。
1252 から 152	ソース・プログラム・ドメインに関連した内部エラー。 この製品を取得した場所に応じて、サービス担当員または承認されたサプライヤーに連絡してください。 **原因** これは内部エラー・メッセージです。 **アクション** サービス担当員または承認された提供業者に連絡して, 問題を報告してください。
1252 から 153	警告: 命令 < *name*> は、PowerPC ® と POWER では異なる機能をします。 **原因** この警告メッセージは、 **as** コマンドの **-w** フラグがコマンド行で使用されない限り、表示されません。 一部の命令には、PowerPC ® と POWER で同じ命令コードがありますが、機能的には異なります。 このメッセージは、アセンブリー・モードが **com** で、以下の指示が使用されている場合に警告を出します。 **アクション** 同じ命令コードを持つが、POWER と PowerPC ® では機能的に異なる命令については、 105 ページの『POWER ® ファミリーと PowerPC ® 命令の機能の違い』 を参照してください。

項目	説明
1252 から 154	第 2 オペランドが無効です。 32 ビット実装の場合、第 2 オペランドの値はゼロでなければなりません。 **原因** 固定小数点比較命令では、32 ビット実装の場合、 L フィールドの値はゼロでなければなりません。 また、 **mtsri** 命令が PowerPC ® アセンブリー・モードのいずれかで使用されている場合、 *RA* オペランドにはゼロが含まれている必要があります。 それ以外の場合は、このメッセージが表示されます。 **アクション** 第 2 オペランドに正しい値を指定してから、プログラムを再アセンブルしてリンクしてください。
1252 から 155	変位は 4 で割り切れる必要があります。 **原因** 命令が DS 形式の場合、その 16 ビット符号付き変位値は 4 で割り切れる必要があります。 それ以外の場合は、このメッセージが表示されます。 **アクション** 変位値を変更してから, プログラムを再びアセンブルしてリンクしてください。
1252 から 156	引数 3 と 4 の合計は 33 より小さくなければなりません。 **原因** ワード回転およびシフト命令の拡張簡略記号が基本命令に変換されると、 SH フィールド、 MB フィールド、または ME フィールドを計算するために、3 番目と 4 番目のオペランドの値が追加されます。 これらのフィールドの長さは 5 ビットであるため、3 番目と 4 番目のオペランドの合計が 32 を超えてはなりません。 **アクション** 拡張簡略記号を基本命令に変換する方法については、99 ページの『32 ビット固定小数点回転命令およびシフト命令の拡張ニーモニック』 を参照してください。 入力オペランドの値を適宜変更してから、プログラムを再アセンブルしてリンクしてください。
1252 から 157	オペランド 3 の値は、オペランド 4 の値以上でなければなりません。 **原因** ワード・ローテート命令およびシフト命令の拡張簡略記号が基本命令に変換されると、 ME または MB フィールドを取得するために、第 3 オペランドの値から第 4 オペランドの値が減算されます。 結果は正の数でなければなりません。 それ以外の場合は、このメッセージが表示されます。 **アクション** 拡張簡略記号を基本命令に変換する方法については、99 ページの『32 ビット固定小数点回転命令およびシフト命令の拡張ニーモニック』 を参照してください。 入力オペランドの値を適宜変更してから、プログラムを再アセンブルしてリンクしてください。
1252 から 158	警告: アセンブリー・モードが *name* の場合は、特殊目的レジスター番号 6 を使用して DEC レジスターを指定します。 **原因** この警告は、 **mfdec** 命令が使用され、アセンブリー・モードが **any** である場合に表示されます。 **mfdec** 命令の DEC エンコードは、PowerPC ® の場合は 22、POWER の場合は 6 です。 アセンブリー・モードが **any** の場合、オブジェクト・コードの生成には POWER エンコード番号が使用され、このことを示すためにこのメッセージが表示されます。 **アクション** なし。

項目	説明
1252 から 159	オペランド < *value*>に対して d (r) 形式は無効です。 **原因** アセンブリー・プログラミング・エラーを示します。 レジスター番号または即時値が必要な場所では、d (r) 形式が使用されます。 **アクション** プログラミング・エラーを訂正してから、プログラムを再アセンブルしてリンクしてください。
1252 から 160	警告: ハッシュ・コード値の長さは 10 バイトでなければなりません。 **原因** **.hash** pseudo-op が使用されている場合、2 番目のパラメーター *StringConstant* は、実際のハッシュ・コード値を指定します。 この値には、2 バイトの言語 ID、4 バイトの汎用ハッシュ、および 4 バイトの言語ハッシュが含まれている必要があります。 ハッシュ・コード値の長さは 10 バイトでなければなりません。 値の長さが 10 バイトではなく、**as** コマンドの **-w** フラグが使用されると、この警告が表示されます。 **アクション** 正しいハッシュ・コード値を使用してください。
1252 から 161	ファイル < *filename*>の処理中にシステム問題が発生しました。 **原因** 動的に作成されたシステム入出力の問題。 このメッセージは、 **fwrite**、 **putc**、または **fclose** エラーを示すためにアセンブラーによって生成されます。 入出力の問題は、ファイル・システムの破損またはファイル・システム内の十分なスペースがないことが原因である可能性があります。 **アクション** 報告されたパス名に従って、適切なファイル・システムを検査してください。
1252 から 162	無効な **-m** フラグ・アセンブリー・モード・オペランド: < *name*>。 **原因** **as** コマンドの **-m** フラグを使用してコマンド行に無効なアセンブリー・モードを入力すると、このメッセージが表示されます。 **アクション** 定義されているアセンブリー・モードについては、 46 ページの『プログラムのアセンブルとリンク』 を参照してください。
1252 から 163	第 1 オペランドの値 < *value*> は、PowerPC ® には無効です。 BO フィールドの 3 番目のビットは、Branch Conditional to Count Register 命令の場合は 1 でなければなりません。 **原因** 138 ページの『bcctr または bcc (カウント・レジスターへの分岐条件付き) 命令』 の *BO* オペランドの 3 番目のビットがゼロの場合、命令形式は無効であり、このメッセージが表示されます。 **アクション** 3 番目のビットを 1 に変更してから, プログラムを再びアセンブルしてリンクしてください。

項目	説明
1252 から 164	この命令形式は、PowerPC ® では無効です。 *RA* および *RB* (命令に存在する場合) は、ロードされるレジスターの範囲内にあってはなりません。 また、 *RA*=*RT*= 0 は許可されません。 **原因** 複数レジスター・ロード命令では、PowerPC ® は、 *RA* オペランド、および命令形式で存在する場合は *RB* オペランドが、ロードするレジスターの範囲内にないことを必要とします。 また、*RA*=*RT*= 0 は許可されません。 それ以外の場合は、このメッセージが表示されます。 **アクション** *RA*、 *RB*、または *RT* オペランドのレジスター番号を調べて、この要件が満たされていることを確認してください。
1252 から 165	PowerPC ® の場合、第 1 オペランドの値はゼロでなければなりません。 **原因** POWER **svca** 命令がいずれかの PowerPC ® アセンブリー・モードで使用される場合、第 1 オペランドは *SV* オペランドです。 このオペランドはゼロでなければなりません。 それ以外の場合は、このメッセージが表示されます。 **アクション** 第 1 オペランドにゼロを入れるか、オペランドを必要としない PowerPC ® **sc** 命令を使用してください。
1252 から 166	この命令形式は、PowerPC ® では無効です。 オペランド 2 で使用されるレジスターは、ゼロであってはなりません。 **原因** 更新形式の固定小数点ストア命令および浮動小数点ロード/ストア命令の場合、PowerPC ® では、*RA* オペランドがゼロに等しくないことが必要です。 それ以外の場合は、このメッセージが表示されます。 **アクション** *RA* オペランドで指定されたレジスター番号を確認してから、ソース・コードを再アセンブルしてリンクしてください。
1252 から 167	-< *flagname*> フラグで名前を指定します。 **原因** **as** コマンドの **-n** フラグと **-o** フラグは、パラメーターとしてファイル名を必要とします。 **as** コマンドの **-m** フラグには、パラメーターとしてモード名が必要です。 必要な名前が欠落している場合は、このエラー・メッセージが表示されます。 このメッセージは、メッセージ 1252-035 に代わるものです。 **アクション** **as** コマンドの **-n** フラグと **-o** フラグを使用してファイル名を指定し、**as** コマンドの **-m** フラグを使用してモード名を指定します。
1252 から 168	**-**< *name*> は認識されるフラグではありません。 **原因** コマンド行で未定義のフラグが使用されました。 このメッセージは、メッセージ 1252-036 に代わるものです。 **アクション** 訂正を行い、コマンドを再実行してください。

項目	説明
1252 から 169	入力ファイルは 1 つしか許可されません。 **原因** コマンド行に複数の入力ソース・ファイルが指定されました。 このメッセージは、メッセージ 1252-037 に代わるものです。 **アクション** 一度に 1 つの入力ソース・ファイルのみを指定してください。
1252 から 170	アセンブラー・コマンドの構文は、 **-l として**[リスト・ファイル] **-s**[リスト・ファイル] **N** 名前 **-** オブジェクト・ファイル [**-w (W)**|**-W (W)**] **-x (X)**[クロス・ファイル] **-u から m** モード名 [入力ファイル] です。 **原因** このメッセージは、 **as** コマンドの使用法を表示します。 **アクション** なし。
1252 から 171	変位は、 *<value1>* より大か等しく、 *<value2>*以下でなければなりません。 **原因** 16 ビット変位の場合、制限は 32767 および -32768 です。 変位が範囲外の場合は、このメッセージが表示されます。 このメッセージは、メッセージ 1252-106 に代わるものです。 **アクション** 変位要件については、具体的な説明記事を参照してください。
1252 から 172	**.extern** 記号は有効ではありません。 **.extern** *Name* が再配置可能式であることを確認してください。 **原因** **.extern** 疑似命令の *Name* パラメーターは、再配置可能式を指定する必要があります。 このメッセージは、 **.extern** 疑似命令の *Name* パラメーターに再配置可能式が指定されていない場合に表示されます。 再配置可能式および再配置不能式については、メッセージ 1252-004 を参照してください。 **アクション** **.extern** 疑似操作の *Name* パラメーターが再配置可能式であることを確認してください。
1252 から 173	警告: 命令 < *name*> の即時値は < *value*>です。 この値が符号なしの値として扱われる場合は、64 ビット・マシンに移植できない可能性があります。 **原因** この警告は、 **addis** 命令 (または **addis** 命令の **lis** 拡張簡略記号) についてのみ報告されます。 これらの命令の即時値フィールドは符号付き整数として定義され、-32768 から 32767 までの有効な値の範囲を持つ必要があります。 ただし、**cau** 命令との互換性を維持するために、この範囲は -65536 から 65535 に拡張されました。 これにより、32 ビット・モードで問題が発生することはありません。符号拡張がどこにもないためです。 ただし、これにより、64 ビット・マシンで問題が発生します。これは、符号拡張がレジスターの上位 32 ビットに伝搬するためです。 **アクション** **addis** 命令を使用して符号なし整数を構成する場合は注意してください。 **addis** 命令は、32 ビット・インプリメンテーション (または 64 ビット・インプリメンテーションの場合は 32 ビット・モード) では、64 ビット・モードとは異なるセマンティクスを持っています。 32 ビット・モードの符号なし整数を持つ **addis** 命令を 64 ビット・モードに直接移植することはできません。 64 ビット・モードで符号なし整数を構成するコード・シーケンスは、32 ビット・モードで必要なコード・シーケンスとは大きく異なります。

項目	説明
1252 から 174	対応する .machine "pop" 命令のない .machine "push" 命令が多すぎます。 **原因** アセンブリー・スタックの最大サイズを超えました。 **machine "push"** で 100 を超える項目がスタックに追加されましたが、 **.machine "pop"**では削除されませんでした。 **アクション** ソース・プログラムを変更して、アセンブリー・スタック・オーバーフロー条件を除去してください。
1252 から 175	.machine "push" が一致しない .machine "pop" が表示されています。 **原因** 疑似命令 **.machine "pop"** がアセンブリー・スタックから項目を除去しようとしましたが、スタックが空です。 ソース・プログラムに **.machine "push"**が欠落している可能性があります。 **アクション** ソース・プログラムを訂正してください。
1252 から 176	セクション <*name*> に .ref pseudo-op を指定することはできません。 **原因** **.ref** 疑似命令は、ストレージ・マッピング・クラスが BS または UC の dsect または csect に現れますが、これは許可されていません。 **アクション** ソース・プログラムを変更してください。
1252 から 177	.ref <*name*> のオペランドが再配置可能シンボルではありません。 **原因** **.ref** pseudo-op オペランド *name* は、dsect 名またはラベル、BS または UC のストレージ・マッピング・クラスを持つ csect 名またはラベル、再配置可能ではない項目を表す **.set** オペランド、または定数値のいずれかです。 **アクション** ソース・プログラムを訂正してください。
1252 から 178	式が参照できるセクションまたはシンボルの最大数を超過しました。 **原因** 式が 50 を超える制御セクション (CSECT または DSECT) を参照しています。 **アクション** ソース・プログラムを訂正してください。
1252 から 179	ファイル # 行 # モード名ロケーション・オブジェクト・コード・ソース **原因** この行は、POWER と PowerPC ® のニーモニック相互参照を使用せずに、アセンブラー・リスト・ファイルの見出しを定義します。
1252 から 180	ファイル # 行 # モード名 Loc OBJ コード PowerPC ® ソース **原因** これは、POWER と PowerPC ® のニーモニック相互参照を含むアセンブラー・リスト・ファイルの見出しの 1 つです。 PowerPC ® のラベルが付いたアセンブラー・リスト列には、ソース・プログラムが POWER ニーモニックを使用するステートメントの PowerPC ® ニーモニックが含まれています。 このメッセージは、PowerPC ® カテゴリーのアセンブリー・モード (**com**、 **ppc**、 **601**、および **any** を含む) に使用されます。

項目	説明
1252 から 181	File# Line# モード名 Loc OBJ コード POWER ソース **原因** これは、POWER と PowerPC ® のニーモニック相互参照を含むアセンブラー・リスト・ファイルの見出しの 1 つです。 POWER というラベルのアセンブラー・リスト列には、ソース・プログラムが PowerPC ® ニーモニックを使用するステートメントの POWER ニーモニックが含まれています。 このメッセージは、POWER カテゴリー (**pwr** および **pwr2** を含む) のアセンブリー・モードに使用されます。
1252 から 182	ストレージ・マッピング・クラス <*name*> は、.comm pseudo-op には無効です。 RW は、オブジェクト・コードのストレージ・マッピング・クラスとして使用されます。 **原因** **.comm** 疑似命令のストレージ・マッピング・クラスは、有効な値 (TD、RW、BS、および UC) 以外の値です。 アセンブラーはこれを警告として報告し、ストレージ・マッピング・クラスとして RW を使用します。 **アクション** ソース・プログラムを変更してください。
1252 から 183	TD csect は ". toc" スコープ内でのみ許可されます。 **原因** 最初に **.toc** 疑似命令を使用せずに、ストレージ・マッピング・クラス TD を持つ csect が使用されました。 **アクション** この命令の前に **.toc** 疑似命令を使用します。
1252 から 184	TOC アンカーは、<*name*> への TOC 相対参照を使用するように定義する必要があります。 ソースに .toc pseudo-op を組み込みます。 **原因** TOC 相対参照が使用されていますが、TOC アンカーが定義されていません。 これは、外部 TD シンボルが定義され、D 形式命令の変位として使用されているが、ソース・プログラムに **.toc** 疑似命令がない場合に起こることがあります。 **アクション** プログラムで **.toc** 疑似命令を使用します。
1252 から 185	警告: オペランドが疑似命令から欠落しています。 **原因** 疑似命令 **.byte**、 **.vbyte**、 **.short**、 **.long**、または **.llong** に必要なオペランドが欠落しています。 **アクション** これらの疑似命令によって作成されるデータ・ストレージ域の初期値を指定します。
1252 から 186	警告: スタブ・ストリングの最大長は <*number*> 文字です。 余分な文字は破棄されました。 **原因** stabstring の長さが制限されています。指定された stabstring は、単一ストリングの最大長を超えています。 **アクション** ストリングを 2 つ以上のストリングに分割し、1 つの stabstring から次の stabstring まで情報を継続します。

項目	説明
1252 から 187	警告: 現行 csect の位置合わせが、.align pseudo-op で指定された位置合わせより小さくなっています。 **原因** csect の位置合わせは、その csect 内で **.align** 疑似命令を使用するために必要な位置合わせほど厳密ではありません。 **アクション** **.align** 疑似命令は、csect 内の項目の位置合わせを指定します。csect に指定する位置合わせは、この値以上でなければなりません。 例えば、csect がワード位置合わせを必要とし、csect 内の .llong がダブルワード位置合わせを必要とする場合、.llong 値が最終的に (リンク後に) ワード位置合わせのみになる可能性があります。 これは、ユーザーが意図したものではない可能性があります。
1252 から 188	ゼロは、<命令> 命令の L オペランドで使用されます。 **原因** 一部の比較命令では、32 ビット・モードで L オペランドをオプションにすることができました。 64 ビット・モードでは、オペランドはオプションではありません。 **アクション** 命令には 4 つのオペランドすべてを指定するか、あるいは拡張簡略記号を使用する必要があります。
1252 から 189	環境変数 OBJECT_MODE の値が無効です。 OBJECT_MODE 環境変数を 32 または 64 に設定するか、 -a32 または -a64 オプションを使用してください。 **原因** *OBJECT_MODE* 環境変数の値は、アセンブラーによって認識されません。 **アクション** *OBJECT_MODE* 環境変数を **32** または **64** に設定するか、 -a32 または -a64 コマンド行オプションを使用します。 環境変数のその他の値は、アセンブラーにとって意味がありません。
1252 から 190	ラベル <*name*> への無効な参照: .function pseudo-op は csect を参照する必要があります。 **原因** **.function** 疑似命令がローカル・ラベルを参照しました。 **アクション** 参照 <*name*> は、csect の名前 (ラベル) でなければなりません。
1252 から 191	再配置可能式には、<*name*> のみを使用する必要があります。 **原因** <*name*> の初期化に使用される式には、外部定義シンボルへの参照が含まれています (つまり、シンボルは **.extern** 疑似命令に現れます)。 **アクション** <*name*> の式オペランド内に外部定義シンボルが含まれていないことを確認してください。 32 ビット・モードでの再配置は、32 ビット数量にのみ適用できます。64 ビット・モードでの再配置は、64 ビット数量にのみ適用できます。
1252 から 192	アセンブリー・モードが指定されていません。 *OBJECT_MODE* 環境変数を 32 または 64 に設定するか、 -a32 または -a64 オプションを使用します。 **原因** 環境変数には値 32_64 が含まれます。 **アクション** *OBJECT_MODE* 環境変数を **32** または **64** に設定するか、 -a32 または -a64 コマンド行オプションを使用します。

項目	説明
1252 から 193	**.set** psuedo-op で指定された値は、32 ビットの符号付き数値として扱われます。 これらの値を **.llong** 式で使用すると、予期しない結果が生じる可能性があります。 **原因** 32 ビット・モードでは、 **.set** を使用した結果の式は、 **.llong** の初期値を設定するために使用されています。 **アクション** 32 ビット・モードで .llong を初期化する場合、値は 64 ビットとして扱われます。 最上位ビットが設定されている .set シンボルが初期化の一部として使用される場合、その値はユーザーが意図した方法で解釈されない可能性があります。 例えば、値 0xFFFF_0000 は 64 ビットの正の数量を意図していた可能性がありますが、負の 32 ビットの数値で、符号が 0xFFFF_FFFF_FFFF_0000 に拡張されます。
1252 から 194	**警告:** 命令 <*instruction*> の即時値は <*number*> です。 この値が符号なしの値として扱われる場合は、64 ビット・マシンに移植できない可能性があります。 **原因** これは、メッセージ 173 の代替バージョンです。詳細については、上記を参照してください。

付録 B ニーモニックでソートされた命令セット

「ニーモニック別にソートされた命令セット」テーブルの「実装」列には、以下の情報が含まれています。

インプリメンテーション	説明
com	POWER ファミリー、 POWER2™、および PowerPC の実装によってサポートされます。
POWER ファミリー	POWER ファミリーおよび POWER2™ 実装によってのみサポートされます。
POWER2™	POWER2™ 実装によってのみサポートされます。
PowerPC	PowerPC アーキテクチャーでのみサポートされます。
PPC オプション。	PowerPC アーキテクチャーでのみ定義され、オプションの命令です。
603 のみ	PowerPC 603 RISC マイクロプロセッサーでのみサポート

項目	説明	インプリメンテーション	フォーマット	主要命令コード	拡張命令コード
ニーモニックでソートされた命令セット	ニーモニックでソートされた命令セット	ニーモニックでソートされた命令セット	ニーモニックでソートされた命令セット	ニーモニックでソートされた命令セット	ニーモニックでソートされた命令セット
略号 (mnemonic)	**命令**	**インプリメンテーション**	**フォーマット**	**主要命令コード**	**拡張命令コード**
a [o] [.]	繰越の追加	POWER ファミリー	Xo	31	10
abs [o] [.]	絶対	POWER ファミリー	Xo	31	360
add [o] [.]	追加	PowerPC	Xo	31	266
addc [o] [.]	繰越の追加	PowerPC	Xo	31	10

項目	説明	インプリメンテーション	フォーマット	主要命令コード	拡張命令コード
adde [o] [.]	拡張の追加	PowerPC	Xo	31	138
アディ	即時追加	PowerPC	D	14	
Addic	即時保持の追加	PowerPC	D	12	
Addic。	即時繰越およびレコードの追加	PowerPC	D	13	
追加	即時シフトの追加	PowerPC	D	15	
addme [o] [.]	1 つの拡張をマイナスに追加	PowerPC	Xo	31	234
addze [o] [.]	ゼロ拡張に追加	PowerPC	Xo	31	202
ae [o] [.]	拡張の追加	POWER ファミリー	Xo	31	138
AI	即時追加	POWER ファミリー	D	12	
ai。	即時およびレコードの追加	POWER ファミリー	D	13	
ame [o] [.]	1 つの拡張をマイナスに追加	POWER ファミリー	Xo	31	234
および [.]	AND	com	X	31	28
および [.]	コンプリンスを伴う AND	com	X	31	60
アンディー	AND 即時	PowerPC	D	28	
アンディル	AND 直下	POWER ファミリー	D	28	
andis.	AND 即時シフト	PowerPC	D	29	
アンディウ	AND すぐ上	POWER ファミリー	D	29	
AZE [O] [.]	ゼロ拡張に追加	POWER ファミリー	Xo	31	202
b [l] [a]	分岐 (branch)	com	I	18	
bc [l] [a]	分岐条件付き	com	B	16	
BCC [L]	カウント・レジスターへの分岐条件付き	POWER ファミリー	XL	19	528
BCCTR [L]	カウント・レジスターへの分岐条件付き	PowerPC	XL	19	528
BCLR [L]	分岐条件リンク・レジスター	PowerPC	XL	19	16
BCR [L]	分岐条件付きレジスター	POWER ファミリー	XL	19	16
cal	アドレスの下限の計算	POWER ファミリー	D	14	

項目	説明	インプリメンテーション	フォーマット	主要命令コード	拡張命令コード
cau	アドレスの上限の計算	POWER ファミリー	D	15	
cax [o] [.]	アドレスの計算	POWER ファミリー	Xo	31	266
Clcs	キャッシュ・ラインの計算サイズ	POWER ファミリー	X	31	531
clf	キャッシュ・ライン・フラッシュ	POWER ファミリー	X	31	118
cli	キャッシュ・ライン無効化	POWER ファミリー	X	31	502
cmp - 2 つのファイルを比較する	比較	com	X	31	0
CMPI	即時比較	com	D	11	
CMPL	論理比較	com	X	31	32
Cmpli	論理比較 (即時)	com	D	10	
cntlz [.]	先行ゼロのカウント	POWER ファミリー	X	31	26
cntlzw [.]	先行ゼロ・ワードのカウント	PowerPC	X	31	26
カニ	条件レジスター AND	com	XL	19	257
Crandc	条件レジスターとコンプリンスを伴う AND	com	XL	19	129
クレクフ	条件レジスターに相当するもの	com	XL	19	289
クラナン	条件レジスター NAND	com	XL	19	225
クナー	条件レジスター NOR	com	XL	19	33
恐怖	条件レジスター OR	com	XL	19	449
Crorc	条件登録 OR (完了あり)	com	XL	19	417
クソル	条件レジスター XOR	com	XL	19	193
DCBF	Data Cache ブロック・フラッシュ	PowerPC	X	31	86
DCBI	Data Cache ブロック無効化	PowerPC	X	31	470

項目	説明	インプリメンテーション	フォーマット	主要命令コード	拡張命令コード
DCBST	Data Cache ブロック・ストア	PowerPC	X	31	54
DCBT	Data Cache ブロック・タッチ	PowerPC	X	31	278
DCBTST	Data Cache 保管のためのブロック・タッチ	PowerPC	X	31	246
DCBZ	Data Cache ブロックがゼロに設定されました	PowerPC	X	31	1014
DCLST	Data Cache ライン・ストア	POWER ファミリー	X	31	630
DCLZ	Data Cache ゼロに設定された行	POWER ファミリー	X	31	1014
DCS	Data Cache 同期化	POWER ファミリー	X	31	598
div [o] [.]	DIVIDE	POWER ファミリー	Xo	31	331
divs [o] [.]	短い分割	POWER ファミリー	Xo	31	363
divw [o] [.]	ワードの分割	PowerPC	Xo	31	491
divwu [o] [.]	ワード符号なし除算	PowerPC	Xo	31	459
doz [o] [.]	差異またはゼロ	POWER ファミリー	Xo	31	264
ドジ	差分またはゼロ即時	POWER ファミリー	D	09	
エシウクス	Word インデックスの外部コントロール	PPC オプション	X	31	310
エコックス	外部制御がワード索引を作成しました	PPC オプション	X	31	438
エイイオ	入出力の順次実行の強制	PowerPC	X	31	854
eqv [.]	同等	com	X	31	284
exts [.]	拡張記号	POWER ファミリー	X	31	922
extsb [.]	拡張符号バイト	PowerPC	X	31	954
extsh [.]	拡張符号ハーフワード	PowerPC	Xo	31	922
fa [.]	浮動追加	POWER ファミリー	A	63	21
fabs [.]	浮動絶対値	com	X	63	264
fadd [.]	浮動追加	PowerPC	A	63	21

項目	説明	インプリメンテーション	フォーマット	主要命令コード	拡張命令コード
fadd [.]	フローティング・シングルの追加	PowerPC	A	59	21
fcir [.]	整数ワードへの浮動変換	POWER ファミリー	X	63	14
fcirz [.]	ゼロへの丸めによる整数ワードへの浮動変換	POWER ファミリー	X	63	15
FC モ	浮動比較順序	com	X	63	32
FCMPU	非順序浮動比較	com	XL	63	0
fctiw [.]	整数ワードへの浮動変換	PowerPC	X	63	14
fctiwz [.]	ゼロへの丸めによる整数ワードへの浮動変換	PowerPC	XL	63	15
FD [.]	浮動小数点除算	POWER ファミリー	A	63	18
fdiv [.]	浮動小数点除算	PowerPC	A	63	18
Fdiv [.]	フローティング・ディバイド・シングル	PowerPC	A	59	18
FM [.]	浮動乗算	POWER ファミリー	A	63	25
FMA [.]	浮動乗算-加算	POWER ファミリー	A	63	29
fmadd [.]	浮動乗算-加算	PowerPC	A	63	29
fmadd [.]	浮動乗算-加算単一	PowerPC	A	59	29
fmr [.]	浮動移動レジスター	com	X	63	72
FMS [.]	浮動乗算-減算	POWER ファミリー	A	63	28
fmsub [.]	浮動乗算-減算	PowerPC	A	63	28
fmsubs [.]	浮動乗算-減算単数	PowerPC	A	59	28
fmul [.]	浮動乗算	PowerPC	A	63	25
fmuls [.]	浮動乗算単数	PowerPC	A	59	25
fnabs [.]	負の浮動絶対値	com	X	63	136
fneg [.]	浮動否定	com	X	63	40
FNMA [.]	負の浮動乗算-加算	POWER ファミリー	A	63	31
fnmadd [.]	負の浮動乗算-加算	PowerPC	A	63	31

項目	説明	インプリメンテーション	フォーマット	主要命令コード	拡張命令コード
fnmadd [.]	負の浮動乗算-単一加算	PowerPC	A	59	31
FNMS [.]	負の浮動乗算-減算	POWER ファミリー	A	63	30
fnmsub [.]	負の浮動乗算-減算	PowerPC	A	63	30
fnmsubs [.]	負の浮動乗算-単精度減算	PowerPC	A	59	30
fres [.]	単精度浮動小数点数の逆数推定値	PPC オプション	A	59	24
frsp [.]	単精度への浮動丸め	com	X	63	12
frsqrte [.]	浮動小数点の平方根の推定値	PPC オプション	A	63	26
fs [.]	浮動減算	POWER ファミリー	A	63	20
F ゼル [.]	浮動小数点選択	PPC オプション	A	63	23
fsqrt [.]	浮動平方根	POWER2™	A	63	22
fsub [.]	浮動減算	PowerPC	A	63	20
fsubs [.]	単精度浮動小数点数	PowerPC	A	59	20
イクビ	命令キャッシュ・ブロック無効化	PowerPC	X	31	982
ics	命令キャッシュの同期化	POWER ファミリー	X	19	150
同期	命令の同期化	PowerPC	X	19	150
l	ロード	POWER ファミリー	D	32	
LBRX	ロード・バイト-逆索引付き	POWER ファミリー	X	31	534
LBZ	ロード・バイトおよびゼロ	com	D	34	
LB ツー	ロード・バイトおよびゼロ (更新あり)	com	D	35	
LBZUX	索引付き更新でバイトおよびゼロをロード	com	X	31	119
LBZX	ロード・バイトおよびゼロ索引	com	X	31	87
LFD	倍精度浮動小数点数のロード	com	D	50	

項目	説明	インプリメンテーション	フォーマット	主要命令コード	拡張命令コード
LFU	Load Floating-Point Double (更新付き)	com	D	51	
LFduX	Load Floating-Point Double with Update Indexed	com	X	31	631
LFDX	浮動小数点二重索引のロード	com	X	31	599
LFQ	浮動小数点クワッドのロード	POWER2™	D	56	
LF 屈曲	更新付き浮動小数点クワッドのロード	POWER2™	D	57	
LFQUX	更新索引付き浮動小数点クワッドのロード	POWER2™	X	31	823
LFQX	浮動小数点クワッド索引付きのロード	POWER2™	X	31	791
LFS	単一の浮動小数点のロード	com	D	48	
LFIU	Load Floating-Point Single with Update (更新付き浮動小数点単一ロード)	com	D	49	
LFSUX	Load Floating-Point Single with Update Indexed (索引付き更新付きロード浮動小数点単一)	com	X	31	567
LFSX	浮動小数点単一索引のロード	com	X	31	535
LHA	ハーフ代数のロード	com	D	42	
ラウ	更新によるハーフ代数のロード	com	D	43	
ラックス	Load Half Algebra with Update Indexed (索引付き更新付きハーフ代数のロード)	com	X	31	375

項目	説明	インプリメンテーション	フォーマット	主要命令コード	拡張命令コード
ラックス	Load Half Algebra Indexed (ハーフ代数索引付きロード)	com	X	31	343
LHBRX	ハーフバイト反転索引付きロード	com	X	31	790
LHZ	ロード・ハーフおよびゼロ	com	D	40	
ラズ	更新による半分とゼロのロード	com	D	41	
LHZUX	更新索引付きのロード・ハーフおよびゼロ	com	X	31	331
LHZX	ロード・ハーフおよびゼロ索引付き	com	X	31	279
LM	複数ロード	POWER ファミリー	D	46	
LMW	複数ワードのロード	PowerPC	D	46	
LSCBX	ストリングのロードとバイト索引の比較	POWER ファミリー	X	31	277
LSI	Load String Immediate (ストリング即時ロード)	POWER ファミリー	X	31	597
L れている	ストリング・ワード即時ロード	PowerPC	X	31	597
LSWX	索引付きストリング・ワードのロード	PowerPC	X	31	533
LsX	ロード・ストリング索引付き	POWER ファミリー	X	31	533
Lu	更新によるロード	POWER ファミリー	D	33	
ルクス	索引付き更新でロード	POWER ファミリー	X	31	55
Lwarx	ロード・ワードおよび予約索引付き	PowerPC	X	31	20
LWBRX	ワード・バイト反転索引付きロード	PowerPC	X	31	534

項目	説明	インプリメンテーション	フォーマット	主要命令コード	拡張命令コード
LWZ	ワードとゼロのロード	PowerPC	D	32	
LW ツー	ゼロ更新でワードをロード	PowerPC	D	33	
LWZUX	索引付き更新でワードとゼロをロード	PowerPC	X	31	55
LWZX	ロード・ワードおよびゼロ索引	PowerPC	X	31	23
LX	索引付きロード	POWER ファミリー	X	31	23
maskg [.]	マスク生成	POWER ファミリー	X	31	29
マスキル [.]	レジスターからのマスク挿入	POWER ファミリー	X	31	541
MCRF	移動条件登録フィールド	com	XL	19	0
MCRFS	FPSCR から条件レジスターへの移動	com	X	63	64
MCRXR	XER から条件レジスターへの移動	com	X	31	512
MFC	条件レジスターから移動	com	X	31	19
マフス [.]	FPSCR から移動	com	X	63	583
MfMSR	マシン状態レジスターから移動	com	X	31	83
MFSPR	特殊目的レジスターからの移動	com	X	31	339
MFS	セグメント・レジスターから移動	com	X	31	595
MFSRI	セグメント・レジスター間接から移動	POWER ファミリー	X	31	627
ムフスライン	セグメント・レジスター間接から移動	PowerPC	X	31	659
MTCRF	条件登録フィールドに移動	com	XFX (X)	31	144
mtfsb0[.]	FPSCR ビット 0 に移動	com	X	63	70

項目	説明	インプリメンテーション	フォーマット	主要命令コード	拡張命令コード
mtfsb1[.]	FPSCR ビット 1 に移動	com	X	63	38
mtfsf [.]	FPSCR フィールドへの移動	com	XFL (X)	63	711
mtfsfi [.]	FPSCR フィールドへの即時移動	com	X	63	134
MTMSR	マシン状態レジスターに移動	com	X	31	146
MTSPR	特殊目的レジスターへの移動	com	X	31	467
MTSR	セグメント・レジスターに移動	com	X	31	210
MTSRI	セグメント・レジスターへの移動 (間接)	POWER ファミリー	X	31	242
ムツリン	セグメント・レジスターへの移動 (間接)	PowerPC	X	31	242
mul [o] [.]	MULTIPLY	POWER ファミリー	Xo	31	107
mulhw [.]	上位ワードの乗算	PowerPC	Xo	31	75
mulhwu [.]	高ワード符号なしの乗算	PowerPC	Xo	31	11
ムリ	乗算-即時	POWER ファミリー	D	07	
ムリ	Multiply Low Immediate	PowerPC	D	07	
mullw [o] [.]	下位ワードの乗算	PowerPC	Xo	31	235
muls [o] [.]	短時間の乗算	POWER ファミリー	Xo	31	235
nabs [o] [.]	負の絶対値	POWER ファミリー	Xo	31	488
および [.]	ナンド	com	X	31	476
否定 [o] [.]	否定する (negate)	com	Xo	31	104
または [.]	NOR	com	X	31	124
または [.]	または	com	X	31	444
OrC [.]	OR (コンプリンスあり)	com	X	31	412
オリ	OR 即時	PowerPC	D	24	
オリル	OR 直下	POWER ファミリー	D	24	
オリス	OR 即時シフト	PowerPC	D	25	

項目	説明	インプリメンテーション	フォーマット	主要命令コード	拡張命令コード
オリウ	OR 即時 (大文字)	POWER ファミリー	D	25	
ラック [.]	実アドレス計算	POWER ファミリー	X	31	818
RFi	割り込みから戻る	com	X	19	50
RFSVC	SVC からの戻り	POWER ファミリー	X	19	82
リミ [.]	すぐに左に回転してから挿入をマスク	POWER ファミリー	M	20	
rlinm [.]	「Rotate Left Immediate then AND with Mask」	POWER ファミリー	M	21	
RLMI [.]	左に回転してから挿入をマスク	POWER ファミリー	M	22	
rlnm [.]	左に回転してマスクと AND	POWER ファミリー	M	23	
rlwimi [.]	「Rotate Left Word Immediate」、「Mask Insert」	PowerPC	M	20	
rlwinm [.]	「Left Word Immediate の回転 (Rotate Left Word Immediate)」、「マスク (Mask)」	PowerPC	M	21	
rlwnm [.]	左にワードを回転してからマスクと AND を実行	PowerPC	M	23	
rrib [.]	右に回転してビットを挿入	POWER ファミリー	X	31	537
sc	システム・コール	PowerPC	SC	17	
sf [o] [.]	減算元	POWER ファミリー	Xo	31	08
sfe [o] [.]	拡張から減算	POWER ファミリー	Xo	31	136
SFI	即時から減算	POWER ファミリー	D	08	
sfme [o] [.]	Minus One Extended からの減算	POWER ファミリー	Xo	31	232
sfze [o] [.]	ゼロ拡張から減算	POWER ファミリー	Xo	31	200

項目	説明	インプリメンテーション	フォーマット	主要命令コード	拡張命令コード
si (シフトイン)	即時減算	com	D	12	
SI	即時およびレコードの減算	com	D	13	
SSL [.]	左にシフト	POWER ファミリー	X	31	24
(sle [.])	左にシフト (拡張)	POWER ファミリー	X	31	153
スレッド [.]	MQ を使用して左にシフトを拡張	POWER ファミリー	X	31	217
sliq [.]	MQ での左への即時シフト	POWER ファミリー	X	31	184
slliq [.]	MQ での長時間即時シフト	POWER ファミリー	X	31	248
SLLQ [.]	MQ での Shift Left Long	POWER ファミリー	X	31	216
slq [.]	MQ を左にシフト	POWER ファミリー	X	31	152
Slw [.]	ワードを左にシフト	PowerPC	X	31	24
SR [.]	右にシフト	POWER ファミリー	X	31	536
スラ [.]	右方代数のシフト	POWER ファミリー	X	31	792
SAI [.]	右方代数即値シフト	POWER ファミリー	X	31	824
sraiq [。]	Shift Right Algebra (右代数), MQ を使用した即時	POWER ファミリー	X	31	952
スラック [.]	MQ を使用した右方代数のシフト	POWER ファミリー	X	31	920
スロー [.]	右方代数ワードのシフト	PowerPC	X	31	792
スラウィ [.]	右方代数語即値シフト	PowerPC	X	31	824
SRE [.]	右にシフト (拡張)	POWER ファミリー	X	31	665
SREA [.]	右方拡張代数シフト	POWER ファミリー	X	31	921
SREQ [.]	MQ による右へのシフトの拡張	POWER ファミリー	X	31	729
Sriq [.]	MQ による右への即時シフト	POWER ファミリー	X	31	696

項目	説明	インプリメンテーション	フォーマット	主要命令コード	拡張命令コード
srliq [.]	MQ による右方への即時シフト	POWER ファミリー	X	31	760
srlq [.]	MQ でのシフト・ライト・ロング	POWER ファミリー	X	31	728
ソース [.]	MQ での Shift RIcatch	POWER ファミリー	X	31	664
ソース [.]	右方ワードのシフト	PowerPC	X	31	536
st	ストア	POWER ファミリー	D	36	
STB	ストア・バイト	com	D	38	
Stbrx	ストア・バイト-逆方向索引付き	POWER ファミリー	X	31	662
Stbu	ストア・バイト(更新あり)	com	D	39	
StBux	索引付き更新でバイトを保管	com	X	31	247
STBX	索引付き保管バイト	com	X	31	215
標準	浮動小数点の倍精度浮動小数点数の格納	com	D	54	
Stfdu	Store Floating-Point Double (更新付き)	com	D	55	
Stfdux	索引付き更新による浮動小数点の倍精度浮動小数点の保管	com	X	31	759
StFDX	ストア浮動小数点二重索引付き	com	X	31	727
StFiwX	浮動小数点を索引付き整数ワードとして保管	PPC オプション	X	31	983
STFQ	ストア・フローティング・ポイント・クワッド	POWER2™	DS	60	
シュテクチュ	ストア・フローティング・ポイント・クワッド(更新あり)	POWER2™	DS	61	

項目	説明	インプリメンテーション	フォーマット	主要命令コード	拡張命令コード
StFQUX	更新索引付きストア・フローティング・ポイント・クワッド	POWER2™	X	31	951
標準 FQX	ストア浮動小数点クワッド索引付き	POWER2™	X	31	919
StFS	単一浮動小数点の保管	com	D	52	
StFsu	ストア・フローティング・ポイント・シングル (更新あり)	com	D	53	
StFSUX	Store Floating-Point Single with Update Indexed (索引付き更新付きストア浮動小数点単一)	com	X	31	695
StfsX	ストア浮動小数点単一索引付き	com	X	31	663
STH	ストア・ハーフ	com	D	44	
STHBRX	ハーフバイト反転索引付き保管	com	X	31	918
STHU	更新付きストア・ハーフ	com	D	45	
STHUX	更新索引付きストア・ハーフ	com	X	31	439
STHX	ストア・ハーフ索引付き	com	X	31	407
STM	複数保管	POWER ファミリー	D	47	
STMW	複数ワードの保管	PowerPC	D	47	
STSI	ストア・ストリング即時	POWER ファミリー	X	31	725
標準	ストリング・ワード即時保管	PowerPC	X	31	725
Stswx	索引付きストリング・ワードの保管	PowerPC	X	31	661
STSX	索引付きストア・ストリング	POWER ファミリー	X	31	661

項目	説明	インプリメンテーション	フォーマット	主要命令コード	拡張命令コード
ストゥー	ストア (更新あり)	POWER ファミリー	D	37	
スタックス	索引付き更新で保管	POWER ファミリー	X	31	183
標準	ストア	PowerPC	D	36	
Stwbrx	ストア・ワード・バイト-逆方向索引付き	PowerPC	X	31	662
stwcx。	条件付き索引付きワードの保管	PowerPC	X	31	150
ストウ	更新付きで Word を保管	PowerPC	D	37	
StWUX	索引付きストア・ワード	PowerPC	X	31	183
StWX	索引付けされたワードの保管	PowerPC	X	31	151
stx (テキスト開始)	索引付きストア	POWER ファミリー	X	31	151
subf [o] [.]	減算元	PowerPC	Xo	31	40
subfc [o] [.]	Subtract from カッリイング	PowerPC	Xo	31	08
サブスクリプション [o] [.]	拡張から減算	PowerPC	Xo	31	136
下位	即時保持からの減算	PowerPC	D	08	
サブフメ [o] [.]	Minus One Extended からの減算	PowerPC	Xo	31	232
サブスクリプション [o] [.]	ゼロ拡張から減算	PowerPC	Xo	31	200
svc [l] [a]	監視プログラム呼び出し	POWER ファミリー	SC	17	
sync	同期化	PowerPC	X	31	598
t	trap	POWER ファミリー	X	31	04
TI	即時トラップ	POWER ファミリー	D	03	
トルビ	変換ルックアサイド・バッファー無効化項目	POWER ファミリー	X	31	306
トルビー	変換ルックアサイド・バッファー無効化項目	PPC オプション	X	31	306

項目	説明	インプリメンテーション	フォーマット	主要命令コード	拡張命令コード
TLBLD	データ TLB エントリーのロード	603 のみ	X	31	978
トルブリ	命令 TLB 項目のロード	603 のみ	X	31	1010
TLBSYNC	変換ルックアサイド・バッファー同期化	PPC オプション	X	31	566
TW	ワードのトラップ	PowerPC	X	31	04
ツイ	トラップ・ワード即時	PowerPC	D	03	
xor [.]	XOR	com	X	31	316
ソリ	XOR 即時	PowerPC	D	26	
ホリル	XOR 即値下限	POWER ファミリー	D	26	
ホリス	XOR 即時シフト	PowerPC	D	27	
Xoriu	XOR 即時上限	POWER ファミリー	D	27	

基本命令コードおよび拡張命令コードでソートされた付録 C 命令セット

「Instruction Set Sorted by Primary and Extended Op Code」テーブルには、命令セットがリストされます。最初に基本命令コードでソートされ、次に拡張命令コードでソートされます。 表の列「実装」には、以下の情報が含まれています。

表 *39. 1* 次 *Op* コードと拡張 *Op* コードでソートされた命令セット

インプリメンテーション	説明
com	POWER ファミリー、 POWER2™、および PowerPC の実装によってサポートされます。
POWER ファミリー	POWER ファミリーおよび POWER2™ 実装によってのみサポートされます。
POWER2™	POWER2™ 実装によってのみサポートされます。
PowerPC	PowerPC アーキテクチャーでのみサポートされます。
PPC オプション。	PowerPC アーキテクチャーでのみ定義され、オプションの命令です。
603 のみ	PowerPC 603 RISC マイクロプロセッサーでのみサポート

項目	説明	説明	説明	説明	説明
1 次 Op コードと拡張 Op コードでソートされた命令セット	**1 次 Op コードと拡張 Op コードでソートされた命令セット**	**1 次 Op コードと拡張 Op コードでソートされた命令セット**	**1 次 Op コードと拡張 Op コードでソートされた命令セット**	**1 次 Op コードと拡張 Op コードでソートされた命令セット**	**1 次 Op コードと拡張 Op コードでソートされた命令セット**
略号 (mnemonic)	**命令**	**インプリメンテーション**	**フォーマット**	**主要命令コード**	**拡張命令コード**
TI	即時トラップ	POWER ファミリー	D	03	
ツイ	トラップ・ワード即時	PowerPC	D	03	
ムリ	乗算-即時	POWER ファミリー	D	07	
ムリ	Multiply Low Immediate	PowerPC	D	07	
SFI	即時から減算	POWER ファミリー	D	08	
下位	即時保持からの減算	PowerPC	D	08	
ドジ	差分またはゼロ即時	POWER ファミリー	D	09	
Cmpli	論理比較 (即時)	com	D	10	
CMPI	即時比較	com	D	11	
Addic	即時保持の追加	PowerPC	D	12	
AI	即時追加	POWER ファミリー	D	12	
si (シフトイン)	即時減算	com	D	12	
Addic。	即時繰越およびレコードの追加	PowerPC	D	13	
SI	即時およびレコードの減算	com	D	13	
ai。	即時およびレコードの追加	POWER ファミリー	D	13	
アディ	即時追加	PowerPC	D	14	
cal	アドレスの下限の計算	POWER ファミリー	D	14	
追加	即時シフトの追加	PowerPC	D	15	
cau	アドレスの上限の計算	POWER ファミリー	D	15	
bc [l] [a]	分岐条件付き	com	B	16	
sc	システム・コール	PowerPC	SC	17	

項目	説明	説明	説明	説明	説明
svc [l] [a]	監視プログラム呼び出し	POWER ファミリー	SC	17	
b [l] [a]	分岐 (branch)	com	I	18	
MCRF	移動条件登録フィールド	com	XL	19	0
BCLR [L]	分岐条件リンク・レジスター	PowerPC	XL	19	16
BCR [L]	分岐条件付きレジスター	POWER ファミリー	XL	19	16
クナー	条件レジスター NOR	com	XL	19	33
RFi	割り込みから戻る	com	X	19	50
RFSVC	SVC からの戻り	POWER ファミリー	X	19	82
Crandc	条件レジスターとコンプリンスを伴う AND	com	XL	19	129
ics	命令キャッシュの同期化	POWER ファミリー	X	19	150
同期	命令の同期化	PowerPC	X	19	150
クソル	条件レジスター XOR	com	XL	19	193
クラナン	条件レジスター NAND	com	XL	19	225
カニ	条件レジスター AND	com	XL	19	257
クレクフ	条件レジスターに相当するもの	com	XL	19	289
Crorc	条件登録 OR (完了あり)	com	XL	19	417
恐怖	条件レジスター OR	com	XL	19	449
BCC [L]	カウント・レジスターへの分岐条件付き	POWER ファミリー	XL	19	528
BCCTR [L]	カウント・レジスターへの分岐条件付き	PowerPC	XL	19	528
リミ [.]	すぐに左に回転してから挿入をマスク	POWER ファミリー	M	20	

項目	説明	説明	説明	説明	説明
rlwimi [.]	「Rotate Left Word Immediate」、「Mask Insert」	PowerPC	M	20	
rlinm [.]	「Rotate Left Immediate then AND with Mask」	POWER ファミリー	M	21	
rlwinm [.]	「Left Word Immediate の回転 (Rotate Left Word Immediate)」、「マスク (Mask)」	PowerPC	M	21	
RLMI [.]	左に回転してから挿入をマスク	POWER ファミリー	M	22	
rlnm [.]	左に回転してマスクと AND	POWER ファミリー	M	23	
rlwnm [.]	左にワードを回転してからマスクと AND を実行	PowerPC	M	23	
オリ	OR 即時	PowerPC	D	24	
オリル	OR 直下	POWER ファミリー	D	24	
オリス	OR 即時シフト	PowerPC	D	25	
オリウ	OR 即時 (大文字)	POWER ファミリー	D	25	
ソリ	XOR 即時	PowerPC	D	26	
ホリル	XOR 即値下限	POWER ファミリー	D	26	
ホリス	XOR 即時シフト	PowerPC	D	27	
Xoriu	XOR 即時上限	POWER ファミリー	D	27	
アンディー	AND 即時	PowerPC	D	28	
アンディル	AND 直下	POWER ファミリー	D	28	
andis.	AND 即時シフト	PowerPC	D	29	
アンディウ	AND すぐ上	POWER ファミリー	D	29	
cmp - 2 つのファイルを比較する	比較	com	X	31	0
t	trap	POWER ファミリー	X	31	04
TW	ワードのトラップ	PowerPC	X	31	04

項目	説明	説明	説明	説明	説明
sf [o] [.]	減算元	POWER ファミリー	Xo	31	08
subfc [o] [.]	Subtract from カッリイング	PowerPC	Xo	31	08
a [o] [.]	繰越の追加	POWER ファミリー	Xo	31	10
addc [o] [.]	繰越の追加	PowerPC	Xo	31	10
mulhwu [.]	高ワード符号なしの乗算	PowerPC	Xo	31	11
MFC	条件レジスターから移動	com	X	31	19
Lwarx	ロード・ワードおよび予約索引付き	PowerPC	X	31	20
LWZX	ロード・ワードおよびゼロ索引	PowerPC	X	31	23
LX	索引付きロード	POWER ファミリー	X	31	23
SSL [.]	左にシフト	POWER ファミリー	X	31	24
Slw [.]	ワードを左にシフト	PowerPC	X	31	24
cntlz [.]	先行ゼロのカウント	POWER ファミリー	X	31	26
cntlzw [.]	先行ゼロ・ワードのカウント	PowerPC	X	31	26
および [.]	AND	com	X	31	28
maskg [.]	マスク生成	POWER ファミリー	X	31	29
CMPL	論理比較	com	X	31	32
subf [o] [.]	減算元	PowerPC	Xo	31	40
DCBST	Data Cache ブロック・ストア	PowerPC	X	31	54
ルクス	索引付き更新でロード	POWER ファミリー	X	31	55
LWZUX	索引付き更新でワードとゼロをロード	PowerPC	X	31	55
および [.]	コンプリンスを伴う AND	com	X	31	60
mulhw [.]	上位ワードの乗算	PowerPC	Xo	31	75
MfMSR	マシン状態レジスターから移動	com	X	31	83
DCBF	Data Cache ブロック・フラッシュ	PowerPC	X	31	86

項目	説明	説明	説明	説明	説明
LBZX	ロード・バイトおよびゼロ索引	com	X	31	87
否定 [o] [.]	否定する (negate)	com	Xo	31	104
mul [o] [.]	MULTIPLY	POWER ファミリー	Xo	31	107
clf	キャッシュ・ライン・フラッシュ	POWER ファミリー	X	31	118
LBZUX	索引付き更新でバイトおよびゼロをロード	com	X	31	119
または [.]	NOR	com	X	31	124
sfe [o] [.]	拡張から減算	POWER ファミリー	Xo	31	136
サブスクリプション [o] [.]	拡張から減算	PowerPC	Xo	31	136
adde [o] [.]	拡張の追加	PowerPC	Xo	31	138
ae [o] [.]	拡張の追加	POWER ファミリー	Xo	31	138
MTCRF	条件登録フィールドに移動	com	XFX (X)	31	144
MTMSR	マシン状態レジスターに移動	com	X	31	146
stwcx。	条件付き索引付きワードの保管	PowerPC	X	31	150
StWX	索引付けされたワードの保管	PowerPC	X	31	151
stx (テキスト開始)	索引付きストア	POWER ファミリー	X	31	151
slq [.]	MQ を左にシフト	POWER ファミリー	X	31	152
(sle [.])	左にシフト (拡張)	POWER ファミリー	X	31	153
スタックス	索引付き更新で保管	POWER ファミリー	X	31	183
StWUX	索引付きストア・ワード	PowerPC	X	31	183
sliq [.]	MQ での左への即時シフト	POWER ファミリー	X	31	184
sfze [o] [.]	ゼロ拡張から減算	POWER ファミリー	Xo	31	200
サブスクリプション [o] [.]	ゼロ拡張から減算	PowerPC	Xo	31	200
addze [o] [.]	ゼロ拡張に追加	PowerPC	Xo	31	202

項目	説明	説明	説明	説明	説明
AZE [O] [.]	ゼロ拡張に追加	POWER ファミリー	Xo	31	202
MTSR	セグメント・レジスターに移動	com	X	31	210
Stbu	ストア・バイト (更新あり)	com	D	39	
STBX	索引付き保管バイト	com	X	31	215
SLLQ [.]	MQ での Shift Left Long	POWER ファミリー	X	31	216
スレッド [.]	MQ を使用して左にシフトを拡張	POWER ファミリー	X	31	217
sfme [o] [.]	Minus One Extended からの減算	POWER ファミリー	Xo	31	232
サブフメ [o] [.]	Minus One Extended からの減算	PowerPC	Xo	31	232
addme [o] [.]	1 つの拡張をマイナスに追加	PowerPC	Xo	31	234
ame [o] [.]	1 つの拡張をマイナスに追加	POWER ファミリー	Xo	31	234
mullw [o] [.]	下位ワードの乗算	PowerPC	Xo	31	235
muls [o] [.]	短時間の乗算	POWER ファミリー	Xo	31	235
MTSRI	セグメント・レジスターへの移動 (間接)	POWER ファミリー	X	31	242
ムツリン	セグメント・レジスターへの移動 (間接)	PowerPC	X	31	242
DCBTST	Data Cache 保管のためのブロック・タッチ	PowerPC	X	31	246
StBux	索引付き更新でバイトを保管	com	X	31	247
slliq [.]	MQ での長時間即時シフト	POWER ファミリー	X	31	248
doz [o] [.]	差異またはゼロ	POWER ファミリー	Xo	31	264
add [o] [.]	追加	PowerPC	Xo	31	266
cax [o] [.]	アドレスの計算	POWER ファミリー	Xo	31	266
LSCBX	ストリングのロードとバイト索引の比較	POWER ファミリー	X	31	277

項目	説明	説明	説明	説明	説明
DCBT	Data Cache ブロック・タッチ	PowerPC	X	31	278
LHZX	ロード・ハーフおよびゼロ索引付き	com	X	31	279
eqv [.]	同等	com	X	31	284
トルビ	変換ルックアサイド・バッファー無効化項目	POWER ファミリー	X	31	306
トルビー	変換ルックアサイド・バッファー無効化項目	PPC オプション	X	31	306
エシウクス	Word インデックスの外部コントロール	PPC オプション	X	31	310
xor [.]	XOR	com	X	31	316
div [o] [.]	DIVIDE	POWER ファミリー	Xo	31	331
LHZUX	更新索引付きのロード・ハーフおよびゼロ	com	X	31	331
MFSPR	特殊目的レジスターからの移動	com	X	31	339
ラックス	Load Half Algebra Indexed (ハーフ代数索引付きロード)	com	X	31	343
abs [o] [.]	絶対	POWER ファミリー	Xo	31	360
divs [o] [.]	短い分割	POWER ファミリー	Xo	31	363
ラックス	Load Half Algebra with Update Indexed (索引付き更新付きハーフ代数のロード)	com	X	31	375
STHX	ストア・ハーフ索引付き	com	X	31	407
OrC [.]	OR (コンプリンスあり)	com	X	31	412
エコックス	外部制御がワード索引を作成しました	PPC オプション	X	31	438
STHUX	更新索引付きストア・ハーフ	com	X	31	439

項目	説明	説明	説明	説明	説明
または [.]	または	com	X	31	444
divwu [o] [.]	ワード符号なし除算	PowerPC	Xo	31	459
MTSPR	特殊目的レジスターへの移動	com	X	31	467
DCBI	Data Cache ブロック無効化	PowerPC	X	31	470
および [.]	ナンド	com	X	31	476
nabs [o] [.]	負の絶対値	POWER ファミリー	Xo	31	488
divw [o] [.]	ワードの分割	PowerPC	Xo	31	491
cli	キャッシュ・ライン無効化	POWER ファミリー	X	31	502
MCRXR	XER から条件レジスターへの移動	com	X	31	512
Clcs	キャッシュ・ラインの計算サイズ	POWER ファミリー	X	31	531
LSWX	索引付きストリング・ワードのロード	PowerPC	X	31	533
LsX	ロード・ストリング索引付き	POWER ファミリー	X	31	533
LBRX	ロード・バイト-逆索引付き	POWER ファミリー	X	31	534
LWBRX	ワード・バイト反転索引付きロード	PowerPC	X	31	534
LFSX	浮動小数点単一索引のロード	com	X	31	535
SR [.]	右にシフト	POWER ファミリー	X	31	536
ソース [.]	右方ワードのシフト	PowerPC	X	31	536
rrib [.]	右に回転してビットを挿入	POWER ファミリー	X	31	537
マスキル [.]	レジスターからのマスク挿入	POWER ファミリー	X	31	541
TLBSYNC	変換ルックアサイド・バッファー同期化	PPC オプション	X	31	566

項目	説明	説明	説明	説明	説明
LFSUX	Load Floating-Point Single with Update Indexed (索引付き更新付きロード浮動小数点単一)	com	X	31	567
MFS	セグメント・レジスターから移動	com	X	31	595
LSI	Load String Immediate (ストリング即時ロード)	POWER ファミリー	X	31	597
L れている	ストリング・ワード即時ロード	PowerPC	X	31	597
DCS	Data Cache 同期化	POWER ファミリー	X	31	598
sync	同期化	PowerPC	X	31	598
LFDX	浮動小数点二重索引のロード	com	X	31	599
MFSRI	セグメント・レジスター間接から移動	POWER ファミリー	X	31	627
DCLST	Data Cache ライン・ストア	POWER ファミリー	X	31	630
LFduX	Load Floating-Point Double with Update Indexed	com	X	31	631
ムフスライン	セグメント・レジスター間接から移動	PowerPC	X	31	659
Stswx	索引付きストリング・ワードの保管	PowerPC	X	31	661
STSX	索引付きストア・ストリング	POWER ファミリー	X	31	661
Stbrx	ストア・バイト-逆方向索引付き	POWER ファミリー	X	31	662
Stwbrx	ストア・ワード・バイト-逆方向索引付き	PowerPC	X	31	662
StfsX	ストア浮動小数点単一索引付き	com	X	31	663

項目	説明	説明	説明	説明	説明
ソース [.]	MQ での Shift RIcatch	POWER ファミリー	X	31	664
SRE [.]	右にシフト (拡張)	POWER ファミリー	X	31	665
StFSUX	Store Floating-Point Single with Update Indexed (索引付き更新付きストア浮動小数点単一)	com	X	31	695
Sriq [.]	MQ による右への即時シフト	POWER ファミリー	X	31	696
STSI	ストア・ストリング即時	POWER ファミリー	X	31	725
標準	ストリング・ワード即時保管	PowerPC	X	31	725
StFDX	ストア浮動小数点二重索引付き	com	X	31	727
srlq [.]	MQ でのシフト・ライト・ロング	POWER ファミリー	X	31	728
SREQ [.]	MQ による右へのシフトの拡張	POWER ファミリー	X	31	729
Stfdux	索引付き更新による浮動小数点の倍精度浮動小数点の保管	com	X	31	759
srliq [.]	MQ による右方への即時シフト	POWER ファミリー	X	31	760
LHBRX	ハーフバイト反転索引付きロード	com	X	31	790
LFQX	浮動小数点クワッド索引付きのロード	POWER2™	X	31	791
スラ [.]	右方代数のシフト	POWER ファミリー	X	31	792
スロー [.]	右方代数ワードのシフト	PowerPC	X	31	792
ラック [.]	実アドレス計算	POWER ファミリー	X	31	818
LFQUX	更新索引付き浮動小数点クワッドのロード	POWER2™	X	31	823

項目	説明	説明	説明	説明	説明
SAI [.]	右方代数即値シフト	POWER ファミリー	X	31	824
スラウィ [.]	右方代数語即値シフト	PowerPC	X	31	824
エイイオ	入出力の順次実行の強制	PowerPC	X	31	854
STHBRX	ハーフバイト反転索引付き保管	com	X	31	918
標準 FQX	ストア浮動小数点クワッド索引付き	POWER2™	X	31	919
スラック [.]	MQ を使用した右方代数のシフト	POWER ファミリー	X	31	920
SREA [.]	右方拡張代数シフト	POWER ファミリー	X	31	921
exts [.]	拡張記号	POWER ファミリー	X	31	922
extsh [.]	拡張符号ハーフワード	PowerPC	Xo	31	922
StFQUX	更新索引付きストア・フローティング・ポイント・クワッド	POWER2™	X	31	951
sraiq [。]	MQ を使用した右方代数の即時シフト	POWER ファミリー	X	31	952
extsb [.]	拡張符号バイト	PowerPC	X	31	954
TLBLD	データ TLB エントリーのロード	603 のみ	X	31	978
イクビ	命令キャッシュ・ブロック無効化	PowerPC	X	31	982
StFiwX	浮動小数点を索引付き整数ワードとして保管	PPC オプション	X	31	983
トルブリ	命令 TLB 項目のロード	603 のみ	X	31	1010
DCBZ	Data Cache ブロックがゼロに設定されました	PowerPC	X	31	1014
DCLZ	Data Cache ゼロに設定された行	POWER ファミリー	X	31	1014
l	ロード	POWER ファミリー	D	32	

項目	説明	説明	説明	説明	説明
LWZ	ワードとゼロのロード	PowerPC	D	32	
Lu	更新によるロード	POWER ファミリー	D	33	
LW ツー	ゼロ更新でワードをロード	PowerPC	D	33	
LBZ	ロード・バイトおよびゼロ	com	D	34	
LB ツー	ロード・バイトおよびゼロ (更新あり)	com	D	35	
st	ストア	POWER ファミリー	D	36	
標準	ストア	PowerPC	D	36	
ストゥー	ストア (更新あり)	POWER ファミリー	D	37	
ストウ	更新付きで Word を保管	PowerPC	D	37	
STB	ストア・バイト	com	D	38	
LHZ	ロード・ハーフおよびゼロ	com	D	40	
ラズ	更新による半分とゼロのロード	com	D	41	
LHA	ハーフ代数のロード	com	D	42	
ラウ	更新によるハーフ代数のロード	com	D	43	
STH	ストア・ハーフ	com	D	44	
STHU	更新付きストア・ハーフ	com	D	45	
LM	複数ロード	POWER ファミリー	D	46	
LMW	複数ワードのロード	PowerPC	D	46	
STM	複数保管	POWER ファミリー	D	47	
STMW	複数ワードの保管	PowerPC	D	47	
LFS	単一の浮動小数点のロード	com	D	48	
LFIU	Load Floating-Point Single with Update (更新付き浮動小数点単一ロード)	com	D	49	

項目	説明	説明	説明	説明	説明
LFD	倍精度浮動小数点数のロード	com	D	50	
LFU	Load Floating-Point Double (更新付き)	com	D	51	
StFS	単一浮動小数点の保管	com	D	52	
StFsu	ストア・フローティング・ポイント・シングル (更新あり)	com	D	53	
標準	浮動小数点の倍精度浮動小数点数の格納	com	D	54	
Stfdu	Store Floating-Point Double (更新付き)	com	D	55	
LFQ	浮動小数点クワッドのロード	POWER2™	D	56	
LF 屈曲	更新付き浮動小数点クワッドのロード	POWER2™	D	57	
Fdiv [.]	フローティング・ディバイド・シングル	PowerPC	A	59	18
fsubs [.]	単精度浮動小数点数	PowerPC	A	59	20
fadd [.]	フローティング・シングルの追加	PowerPC	A	59	21
fres [.]	単精度浮動小数点数の逆数推定値	PPC オプション	A	59	24
fmuls [.]	浮動乗算単数	PowerPC	A	59	25
fmsubs [.]	浮動乗算-減算単数	PowerPC	A	59	28
fmadd [.]	浮動乗算-加算単一	PowerPC	A	59	29
fnmsubs [.]	負の浮動乗算-単精度減算	PowerPC	A	59	30
fnmadd [.]	負の浮動乗算-単一加算	PowerPC	A	59	31
STFQ	ストア・フローティング・ポイント・クワッド	POWER2™	DS	60	

項目	説明	説明	説明	説明	説明
シュテクチュ	ストア・フローティング・ポイント・クワッド (更新あり)	POWER2™	DS	61	
FCMPU	非順序浮動比較	com	XL	63	0
frsp [.]	単精度への浮動丸め	com	X	63	12
fcir [.]	整数ワードへの浮動変換	POWER ファミリー	X	63	14
fctiw [.]	整数ワードへの浮動変換	PowerPC	X	63	14
fcirz [.]	ゼロへの丸めによる整数ワードへの浮動変換	POWER ファミリー	X	63	15
fctiwz [.]	ゼロへの丸めによる整数ワードへの浮動変換	PowerPC	XL	63	15
FD [.]	浮動小数点除算	POWER ファミリー	A	63	18
fdiv [.]	浮動小数点除算	PowerPC	A	63	18
fs [.]	浮動減算	POWER ファミリー	A	63	20
fsub [.]	浮動減算	PowerPC	A	63	20
fa [.]	浮動追加	POWER ファミリー	A	63	21
fadd [.]	浮動追加	PowerPC	A	63	21
fsqrt [.]	浮動平方根	POWER2™	A	63	22
F ゼル [.]	浮動小数点選択	PPC オプション	A	63	23
FM [.]	浮動乗算	POWER ファミリー	A	63	25
fmul [.]	浮動乗算	PowerPC	A	63	25
frsqrte [.]	浮動小数点の平方根の推定値	PPC オプション	A	63	26
FMS [.]	浮動乗算-減算	POWER ファミリー	A	63	28
fmsub [.]	浮動乗算-減算	PowerPC	A	63	28
FMA [.]	浮動乗算-加算	POWER ファミリー	A	63	29
fmadd [.]	浮動乗算-加算	PowerPC	A	63	29
FNMS [.]	負の浮動乗算-減算	POWER ファミリー	A	63	30
fnmsub [.]	負の浮動乗算-減算	PowerPC	A	63	30
FNMA [.]	負の浮動乗算-加算	POWER ファミリー	A	63	31
fnmadd [.]	負の浮動乗算-加算	PowerPC	A	63	31

項目	説明	説明	説明	説明	説明
FC モ	浮動比較順序	com	X	63	32
mtfsb1[.]	FPSCR ビット 1 に移動	com	X	63	38
fneg [.]	浮動否定	com	X	63	40
MCRFS	FPSCR から条件レジスターへの移動	com	X	63	64
mtfsb0[.]	FPSCR ビット 0 に移動	com	X	63	70
fmr [.]	浮動移動レジスター	com	X	63	72
mtfsfi [.]	FPSCR フィールドへの即時移動	com	X	63	134
fnabs [.]	負の浮動絶対値	com	X	63	136
fabs [.]	浮動絶対値	com	X	63	264
マフス [.]	FPSCR から移動	com	X	63	583
mtfsf [.]	FPSCR フィールドへの移動	com	XFL (X)	63	711

POWER ファミリー、 POWER2™、および PowerPC に共通する付録 D の説明

項目	説明	説明	説明	説明
POWER ファミリー、 POWER2™、および PowerPC に共通の手順	POWER ファミリー、 POWER2™、および PowerPC に共通の手順	POWER ファミリー、 POWER2™、および PowerPC に共通の手順	POWER ファミリー、 POWER2™、および PowerPC に共通の手順	POWER ファミリー、 POWER2™、および PowerPC に共通の手順
略号 (mnemonic)	**命令**	**フォーマット**	**主要命令コード**	**拡張命令コード**
および [.]	AND	X	31	28
および [.]	コンプリンスを伴う AND	X	31	60
b [l] [a]	分岐 (branch)	I	18	
bc [l] [a]	分岐条件付き	B	16	
cmp - 2 つのファイルを比較する	比較	X	31	0
CMPI	即時比較	D	11	
CMPL	論理比較	X	31	32
Cmpli	論理比較 (即時)	D	10	

項目	説明	説明	説明	説明
カニ	条件レジスター AND	XL	19	257
Crandc	条件レジスターとコンプリンスを伴う AND	XL	19	129
クレクフ	条件レジスターに相当するもの	XL	19	289
クラナン	条件レジスター NAND	XL	19	225
クナー	条件レジスター NOR	XL	19	33
恐怖	条件レジスター OR	XL	19	449
Crorc	条件登録 OR (完了あり)	XL	19	417
クソル	条件レジスター XOR	XL	19	193
エシウクス	Word インデックスの外部コントロール	X	31	310
エコックス	外部制御がワード索引を作成しました	X	31	438
eqv [.]	同等	X	31	284
fabs [.]	浮動絶対値	X	63	264
FC モ	浮動比較順序	X	63	32
FCMPU	非順序浮動比較	XL	63	0
fmr [.]	浮動移動レジスター	X	63	72
fnabs [.]	負の浮動絶対値	X	63	136
fneg [.]	浮動否定	X	63	40
frsp [.]	単精度への浮動丸め	X	63	12
LBZ	ロード・バイトおよびゼロ	D	34	
LB ツー	ロード・バイトおよびゼロ (更新あり)	D	35	
LBZUX	索引付き更新でバイトおよびゼロをロード	X	31	119
LBZX	ロード・バイトおよびゼロ索引	X	31	87
LFD	倍精度浮動小数点数のロード	D	50	

項目	説明	説明	説明	説明
LFU	Load Floating-Point Double (更新付き)	D	51	
LFduX	Load Floating-Point Double with Update Indexed	X	31	631
LFDX	浮動小数点二重索引のロード	X	31	599
LFS	単一の浮動小数点のロード	D	48	
LFIU	Load Floating-Point Single with Update (更新付き浮動小数点単一ロード)	D	49	
LFSUX	Load Floating-Point Single with Update Indexed (索引付き更新付きロード浮動小数点単一)	X	31	567
LFSX	浮動小数点単一索引のロード	X	31	535
LHA	ハーフ代数のロード	D	42	
ラウ	更新によるハーフ代数のロード	D	43	
ラックス	Load Half Algebra with Update Indexed (索引付き更新付きハーフ代数のロード)	X	31	375
ラックス	Load Half Algebra Indexed (ハーフ代数索引付きロード)	X	31	343
LHBRX	ハーフバイト反転索引付きロード	X	31	790
LHZ	ロード・ハーフおよびゼロ	D	40	
ラズ	更新による半分とゼロのロード	D	41	
LHZUX	更新索引付きのロード・ハーフおよびゼロ	X	31	331
LHZX	ロード・ハーフおよびゼロ索引付き	X	31	279
MCRF	移動条件登録フィールド	XL	19	0

項目	説明	説明	説明	説明
MCRFS	FPSCR から条件レジスターへの移動	X	63	64
MCRXR	XER から条件レジスターへの移動	X	31	512
MFC	条件レジスターから移動	X	31	19
マフス [.]	FPSCR から移動	X	63	583
MfMSR	マシン状態レジスターから移動	X	31	83
MFSPR	特殊目的レジスターからの移動	X	31	339
MFS	セグメント・レジスターから移動	X	31	595
MTCRF	条件登録フィールドに移動	XFX (X)	31	144
mtfsb0[.]	FPSCR ビット 0 に移動	X	63	70
mtfsb1[.]	FPSCR ビット 1 に移動	X	63	38
mtfsf [.]	FPSCR フィールドへの移動	XFL (X)	63	711
mtfsfi [.]	FPSCR フィールドへの即時移動	X	63	134
MTMSR	マシン状態レジスターに移動	X	31	146
MTSPR	特殊目的レジスターへの移動	X	31	467
MTSR	セグメント・レジスターに移動	X	31	210
および [.]	ナンド	X	31	476
否定 [o] [.]	否定する (negate)	Xo	31	104
または [.]	NOR	X	31	124
または [.]	または	X	31	444
OrC [.]	OR (コンプリンスあり)	X	31	412
RFi	割り込みから戻る	X	19	50
si (シフトイン)	即時減算	D	12	
SI	即時およびレコードの減算	D	13	
STB	ストア・バイト	D	38	

項目	説明	説明	説明	説明
Stbu	ストア・バイト (更新あり)	D	39	
StBux	索引付き更新でバイトを保管	X	31	247
STBX	索引付き保管バイト	X	31	215
標準	浮動小数点の倍精度浮動小数点数の格納	D	54	
Stfdu	Store Floating-Point Double (更新付き)	D	55	
Stfdux	索引付き更新による浮動小数点の倍精度浮動小数点の保管	X	31	759
StFDX	ストア浮動小数点二重索引付き	X	31	727
StFS	単一浮動小数点の保管	D	52	
StFsu	ストア・フローティング・ポイント・シングル (更新あり)	D	53	
StFSUX	Store Floating-Point Single with Update Indexed (索引付き更新付きストア浮動小数点単一)	X	31	695
StfsX	ストア浮動小数点単一索引付き	X	31	663
STH	ストア・ハーフ	D	44	
STHBRX	ハーフバイト反転索引付き保管	X	31	918
STHU	更新付きストア・ハーフ	D	45	
STHUX	更新索引付きストア・ハーフ	X	31	439
STHX	ストア・ハーフ索引付き	X	31	407
xor [.]	XOR	X	31	316

付録 E POWER ファミリーおよび POWER2™ の説明

以下の POWER ファミリーおよび POWER2™ の指示表では、 POWER2™ 実装でのみサポートされる指示は、 POWER2™ 「**のみ**」 列の「POWER2™」で示されています。

項目	**説明**	**説明**	**説明**	**説明**	**説明**
POWER ファミリー **および** POWER2™ **の説明**	POWER ファミリー **および** POWER2™ **の説明**	POWER ファミリー **および** POWER2™ **の説明**	POWER ファミリー **および** POWER2™ **の説明**	POWER ファミリー **および** POWER2™ **の説明**	POWER ファミリー **および** POWER2™ **の説明**
略号 (mnemonic)	**命令**	POWER2™ **のみ**	**フォーマット**	**主要命令コード**	**拡張命令コード**
a [o] [.]	繰越の追加		Xo	31	10
abs [o] [.]	絶対		Xo	31	360
ae [o] [.]	拡張の追加		Xo	31	138
AI	即時追加		D	12	
ai。	即時およびレコードの追加		D	13	
ame [o] [.]	1 つの拡張をマイナスに追加		Xo	31	234
および [.]	AND		X	31	28
および [.]	コンプリンスを伴う AND		X	31	60
アンディル	AND 直下		D	28	
アンディウ	AND すぐ上		D	29	
AZE [O] [.]	ゼロ拡張に追加		Xo	31	202
b [l] [a]	分岐 (branch)		I	18	
bc [l] [a]	分岐条件付き		B	16	
BCC [L]	カウント・レジスターへの分岐条件付き		XL	19	528
BCR [L]	分岐条件付きレジスター		XL	19	16
cal	アドレスの下限の計算		D	14	
cau	アドレスの上限の計算		D	15	
cax [o] [.]	アドレスの計算		Xo	31	266
Clcs	キャッシュ・ラインの計算サイズ		X	31	531
clf	キャッシュ・ライン・フラッシュ		X	31	118
cli	キャッシュ・ライン無効化		X	31	502

項目	説明	説明	説明	説明	説明
cmp - 2 つのファイルを比較する	比較		X	31	0
CMPI	即時比較		D	11	
CMPL	論理比較		X	31	32
Cmpli	論理比較 (即時)		D	10	
cntlz [.]	先行ゼロのカウント		X	31	26
カニ	条件レジスター AND		XL	19	257
Crandc	条件レジスターとコンプリンスを伴う AND		XL	19	129
クレクフ	条件レジスターに相当するもの		XL	19	289
クラナン	条件レジスター NAND		XL	19	225
クナー	条件レジスター NOR		XL	19	33
恐怖	条件レジスター OR		XL	19	449
Crorc	条件登録 OR (完了あり)		XL	19	417
クソル	条件レジスター XOR		XL	19	193
DCLST	Data Cache ライン・ストア		X	31	630
DCLZ	Data Cache ゼロに設定された行		X	31	1014
DCS	Data Cache 同期化		X	31	598
div [o] [.]	DIVIDE		Xo	31	331
divs [o] [.]	短い分割		Xo	31	363
doz [o] [.]	差異またはゼロ		Xo	31	264
ドジ	差分またはゼロ即時		D	09	
エシウクス	Word インデックスの外部コントロール		X	31	310
エコックス	外部制御がワード索引を作成しました		X	31	438

項目	説明	説明	説明	説明	説明
eqv [.]	同等		X	31	284
exts [.]	拡張記号		X	31	922
fa [.]	浮動追加		A	63	21
fabs [.]	浮動絶対値		X	63	264
fcir [.]	整数ワードへの浮動変換		X	63	14
fcirz [.]	ゼロへの丸めによる整数ワードへの浮動変換		X	63	15
FC モ	浮動比較順序		X	63	32
FCMPU	非順序浮動比較		XL	63	0
FD [.]	浮動小数点除算		A	63	18
FM [.]	浮動乗算		A	63	25
FMA [.]	浮動乗算-加算		A	63	29
fmr [.]	浮動移動レジスター		X	63	72
FMS [.]	浮動乗算-減算		A	63	28
fnabs [.]	負の浮動絶対値		X	63	136
fneg [.]	浮動否定		X	63	40
FNMA [.]	負の浮動乗算-加算		A	63	31
FNMS [.]	負の浮動乗算-減算		A	63	30
frsp [.]	単精度への浮動丸め		X	63	12
fs [.]	浮動減算		A	63	20
fsqrt [.]	浮動平方根	POWER2™	A	63	22
ics	命令キャッシュの同期化		X	19	150
l	ロード		D	32	
LBRX	ロード・バイト-逆索引付き		X	31	534
LBZ	ロード・バイトおよびゼロ		D	34	
LB ツー	ロード・バイトおよびゼロ (更新あり)		D	35	
LBZUX	索引付き更新でバイトおよびゼロをロード		X	31	119

項目	説明	説明	説明	説明	説明
LBZX	ロード・バイトおよびゼロ索引		X	31	87
LFD	倍精度浮動小数点数のロード		D	50	
LFU	Load Floating-Point Double (更新付き)		D	51	
LFduX	Load Floating-Point Double with Update Indexed		X	31	631
LFDX	浮動小数点二重索引のロード		X	31	599
LFQ	浮動小数点クワッドのロード	POWER2™	D	56	
LF 屈曲	更新付き浮動小数点クワッドのロード	POWER2™	D	57	
LFQUX	更新索引付き浮動小数点クワッドのロード	POWER2™	X	31	823
LFQX	浮動小数点クワッド索引付きのロード	POWER2™	X	31	791
LFS	単一の浮動小数点のロード		D	48	
LFIU	Load Floating-Point Single with Update (更新付き浮動小数点単一ロード)		D	49	
LFSUX	Load Floating-Point Single with Update Indexed (索引付き更新付きロード浮動小数点単一)		X	31	567
LFSX	浮動小数点単一索引のロード		X	31	535
LHA	ハーフ代数のロード		D	42	
ラウ	更新によるハーフ代数のロード		D	43	

項目	説明	説明	説明	説明	説明
ラックス	Load Half Algebra with Update Indexed (索引付き更新付きハーフ代数のロード)		X	31	375
ラックス	Load Half Algebra Indexed (ハーフ代数索引付きロード)		X	31	343
LHBRX	ハーフバイト反転索引付きロード		X	31	790
LHZ	ロード・ハーフおよびゼロ		D	40	
ラズ	更新による半分とゼロのロード		D	41	
LHZUX	更新索引付きのロード・ハーフおよびゼロ		X	31	331
LHZX	ロード・ハーフおよびゼロ索引付き		X	31	279
LM	複数ロード		D	46	
LSCBX	ストリングのロードとバイト索引の比較		X	31	277
LSI	Load String Immediate (ストリング即時ロード)		X	31	597
LsX	ロード・ストリング索引付き		X	31	533
Lu	更新によるロード		D	33	
ルクス	索引付き更新でロード		X	31	55
LX	索引付きロード		X	31	23
maskg [.]	マスク生成		X	31	29
マスキル [.]	レジスターからのマスク挿入		X	31	541
MCRF	移動条件登録フィールド		XL	19	0

項目	説明	説明	説明	説明	説明
MCRFS	FPSCR から条件レジスターへの移動		X	63	64
MCRXR	XER から条件レジスターへの移動		X	31	512
MFC	条件レジスターから移動		X	31	19
マフス [.]	FPSCR から移動		X	63	583
MfMSR	マシン状態レジスターから移動		X	31	83
MFSPR	特殊目的レジスターからの移動		X	31	339
MFS	セグメント・レジスターから移動		X	31	595
MFSRI	セグメント・レジスター間接から移動		X	31	627
MTCRF	条件登録フィールドに移動		XFX (X)	31	144
mtfsb0[.]	FPSCR ビット 0 に移動		X	63	70
mtfsb1[.]	FPSCR ビット 1 に移動		X	63	38
mtfsf [.]	FPSCR フィールドへの移動		XFL (X)	63	711
mtfsfi [.]	FPSCR フィールドへの即時移動		X	63	134
MTMSR	マシン状態レジスターに移動		X	31	146
MTSPR	特殊目的レジスターへの移動		X	31	467
MTSR	セグメント・レジスターに移動		X	31	210
MTSRI	セグメント・レジスターへの移動 (間接)		X	31	242
mul [o] [.]	MULTIPLY		Xo	31	107
ムリ	乗算-即時		D	07	
muls [o] [.]	短時間の乗算		Xo	31	235
nabs [o] [.]	負の絶対値		Xo	31	488

項目	説明	説明	説明	説明	説明
および [.]	ナンド		X	31	476
否定 [o] [.]	否定する (negate)		Xo	31	104
または [.]	NOR		X	31	124
または [.]	または		X	31	444
OrC [.]	OR (コンプリンスあり)		X	31	412
オリル	OR 直下		D	24	
オリウ	OR 即時 (大文字)		D	25	
ラック [.]	実アドレス計算		X	31	818
RFi	割り込みから戻る		X	19	50
RFSVC	SVC からの戻り		X	19	82
リミ [.]	すぐに左に回転してから挿入をマスク		M	20	
rlinm [.]	「Rotate Left Immediate then AND with Mask」		M	21	
RLMI [.]	左に回転してから挿入をマスク		M	22	
rlnm [.]	左に回転してマスクと AND		M	23	
rrib [.]	右に回転してビットを挿入		X	31	537
sf [o] [.]	減算元		Xo	31	08
sfe [o] [.]	拡張から減算		Xo	31	136
SFI	即時から減算		D	08	
sfme [o] [.]	Minus One Extended からの減算		Xo	31	232
sfze [o] [.]	ゼロ拡張から減算		Xo	31	200
si (シフトイン)	即時減算		D	12	
SI	即時およびレコードの減算		D	13	
SSL [.]	左にシフト		X	31	24
(sle [.])	左にシフト (拡張)		X	31	153

項目	説明	説明	説明	説明	説明
スレッド [.]	MQ を使用して左にシフトを拡張		X	31	217
sliq [.]	MQ での左への即時シフト		X	31	184
slliq [.]	MQ での長時間即時シフト		X	31	248
SLLQ [.]	MQ での Shift Left Long		X	31	216
slq [.]	MQ を左にシフト		X	31	152
SR [.]	右にシフト		X	31	536
スラ [.]	右方代数のシフト		X	31	792
SAI [.]	右方代数即値シフト		X	31	824
sraiq [。]	MQ を使用した右方代数の即時シフト		X	31	952
スラック [.]	MQ を使用した右方代数のシフト		X	31	920
SRE [.]	右にシフト (拡張)		X	31	665
SREA [.]	右方拡張代数シフト		X	31	921
SREQ [.]	MQ による右へのシフトの拡張		X	31	729
Sriq [.]	MQ による右への即時シフト		X	31	696
srliq [.]	MQ による右方への即時シフト		X	31	760
srlq [.]	MQ でのシフト・ライト・ロング		X	31	728
ソース [.]	MQ での Shift RIcatch		X	31	664
st	ストア		D	36	
STB	ストア・バイト		D	38	
Stbrx	ストア・バイト-逆方向索引付き		X	31	662
Stbu	ストア・バイト (更新あり)		D	39	

項目	説明	説明	説明	説明	説明
StBux	索引付き更新でバイトを保管		X	31	247
STBX	索引付き保管バイト		X	31	215
標準	浮動小数点の倍精度浮動小数点数の格納		D	54	
Stfdu	Store Floating-Point Double (更新付き)		D	55	
Stfdux	索引付き更新による浮動小数点の倍精度浮動小数点の保管		X	31	759
StFDX	ストア浮動小数点二重索引付き		X	31	727
STFQ	ストア・フローティング・ポイント・クワッド	POWER2™	DS	60	
シュテクチュ	ストア・フローティング・ポイント・クワッド (更新あり)	POWER2™	DS	61	
StFQUX	更新索引付きストア・フローティング・ポイント・クワッド	POWER2™	X	31	951
標準 FQX	ストア浮動小数点クワッド索引付き	POWER2™	X	31	919
StFS	単一浮動小数点の保管		D	52	
StFsu	ストア・フローティング・ポイント・シングル (更新あり)		D	53	
StFSUX	Store Floating-Point Single with Update Indexed (索引付き更新付きストア浮動小数点単一)		X	31	695
StfsX	ストア浮動小数点単一索引付き		X	31	663
STH	ストア・ハーフ		D	44	

項目	説明	説明	説明	説明	説明
STHBRX	ハーフバイト反転索引付き保管		X	31	918
STHU	更新付きストア・ハーフ		D	45	
STHUX	更新索引付きストア・ハーフ		X	31	439
STHX	ストア・ハーフ索引付き		X	31	407
STM	複数保管		D	47	
STSI	ストア・ストリング即時		X	31	725
STSX	索引付きストア・ストリング		X	31	661
ストゥー	ストア (更新あり)		D	37	
スタックス	索引付き更新で保管		X	31	183
stx (テキスト開始)	索引付きストア		X	31	151
svc [l] [a]	監視プログラム呼び出し		SC	17	
t	trap		X	31	04
TI	即時トラップ		D	03	
トルビ	変換ルックアサイド・バッファー無効化項目		X	31	306
xor [.]	XOR		X	31	316
ホリル	XOR 即値下限		D	26	
Xoriu	XOR 即時上限		D	27	

付録 F PowerPC ® の説明

表 *40. PowerPC ®* の説明

ニーモニック	命令	Format	主要命令コード	拡張命令コード
add [o] [.]	追加	Xo	31	266
addc [o] [.]	繰越の追加	Xo	31	10
adde [o] [.]	拡張の追加	Xo	31	138
アディ	即時追加	D	14	
Addic	即時保持の追加	D	12	

表 *40. PowerPC ®* の説明 *(続き)*				
ニーモニック	**命令**	**Format**	**主要命令コード**	**拡張命令コード**
Addic。	即時繰越およびレコードの追加	D	13	
追加	即時シフトの追加	D	15	
addme [o] [.]	1 つの拡張をマイナスに追加	Xo	31	234
addze [o] [.]	ゼロ拡張に追加	Xo	31	202
および [.]	AND	X	31	28
および [.]	コンプリンスを伴う AND	X	31	60
アンディー	AND 即時	D	28	
andis.	AND 即時シフト	D	29	
b [l] [a]	分岐 (branch)	I	18	
bc [l] [a]	分岐条件付き	B	16	
BCCTR [L]	カウント・レジスターへの分岐条件付き	XL	19	528
BCLR [L]	分岐条件リンク・レジスター	XL	19	16
cmp - 2 つのファイルを比較する	比較	X	31	0
CMPI	即時比較	D	11	
CMPL	論理比較	X	31	32
Cmpli	論理比較 (即時)	D	10	
cntlzd	先行ゼロ・ダブルワードのカウント	X	31	58
cntlzw [.]	先行ゼロ・ワードのカウント	X	31	26
カニ	条件レジスター AND	XL	19	257
Crandc	条件レジスターとコンプリンスを伴う AND	XL	19	129
クレクフ	条件レジスターに相当するもの	XL	19	289
クラナン	条件レジスター NAND	XL	19	225
クナー	条件レジスター NOR	XL	19	33
恐怖	条件レジスター OR	XL	19	449

表 *40. PowerPC ®* の説明 *(続き)*				
ニーモニック	**命令**	**Format**	**主要命令コード**	**拡張命令コード**
Crorc	条件登録 OR (完了あり)	XL	19	417
クソル	条件レジスター XOR	XL	19	193
DCBF	Data Cache ブロック・フラッシュ	X	31	86
DCBI	Data Cache ブロック無効化	X	31	470
DCBST	Data Cache ブロック・ストア	X	31	54
DCBT	Data Cache ブロック・タッチ	X	31	278
DCBTST	Data Cache 保管のためのブロック・タッチ	X	31	246
DCBZ	Data Cache ブロックがゼロに設定されました	X	31	1014
Div	ダブルワードの除算	Xo	31	489
ディヴドゥ	ダブルワード符号なし除算	Xo	31	457
divw [o] [.]	ワードの分割	Xo	31	491
divwu [o] [.]	ワード符号なし除算	Xo	31	459
エシウクス	Word 索引付きの外部コントロール (オプション)	X	31	310
エコックス	外部コントロール・アウト・ワード索引付き (オプション)	X	31	438
エイイオ	入出力の順次実行の強制	X	31	854
eqv [.]	同等	X	31	284
extsb [.]	拡張符号バイト	X	31	954
extsh [.]	拡張符号ハーフワード	Xo	31	922
EXT	拡張符号ワード	X	31	986
fabs [.]	浮動絶対値	X	63	264
fadd [.]	浮動追加	A	63	21

表 *40. PowerPC ®* の説明 *(続き)*				
ニーモニック	**命令**	**Format**	**主要命令コード**	**拡張命令コード**
fadd [.]	フローティング・シングルの追加	A	59	21
フィド	整数ダブルワードからの浮動変換	X	63	846
FC モ	浮動比較順序	X	63	32
FCMPU	非順序浮動比較	XL	63	0
FC ID	整数ダブルワードへの浮動変換	X	63	814
Fctidz	ゼロ方向への丸めによる整数ダブルワードへの浮動変換	X	63	815
fctiw [.]	整数ワードへの浮動変換	X	63	14
fctiwz [.]	ゼロへの丸めによる整数ワードへの浮動変換	XL	63	15
fdiv [.]	浮動小数点除算	A	63	18
Fdiv [.]	フローティング・ディバイド・シングル	A	59	18
fmadd [.]	浮動乗算-加算	A	63	29
fmadd [.]	浮動乗算-加算単一	A	59	29
fmr [.]	浮動移動レジスター	X	63	72
fmsub [.]	浮動乗算-減算	A	63	28
fmsubs [.]	浮動乗算-減算単数	A	59	28
fmul [.]	浮動乗算	A	63	25
fmuls [.]	浮動乗算単数	A	59	25
fnabs [.]	負の浮動絶対値	X	63	136
fneg [.]	浮動否定	X	63	40
fnmadd [.]	負の浮動乗算-加算	A	63	31
fnmadd [.]	負の浮動乗算-単一加算	A	59	31
fnmsub [.]	負の浮動乗算-減算	A	63	30
fnmsubs [.]	負の浮動乗算-単精度減算	A	59	30
fres [.]	単精度浮動小数点数の逆数推定値 (オプション)	A	59	24

表 *40. PowerPC ®* の説明 *(*続き*)*

ニーモニック	命令	Format	主要命令コード	拡張命令コード
frsp [.]	単精度への浮動丸め	X	63	12
frsqrte [.]	浮動小数点の平方根の推定値 (オプション)	A	63	26
F ゼル [.]	浮動小数点選択 (オプション)	A	63	23
fsub [.]	浮動減算	A	63	20
fsubs [.]	単精度浮動小数点数	A	59	20
イクビ	命令キャッシュ・ブロック無効化	X	31	982
同期	命令の同期化	X	19	150
LBZ	ロード・バイトおよびゼロ	D	34	
LB ツー	ロード・バイトおよびゼロ (更新あり)	D	35	
LBZUX	索引付き更新でバイトおよびゼロをロード	X	31	119
LBZX	ロード・バイトおよびゼロ索引	X	31	87
ld	ダブルワードのロード	DS	58	0
Ldarx	ダブルワードのロードおよび索引付き予約	X	31	84
LDU	更新によるダブルワードのロード	DS	58	1
Ldux	索引付き更新でダブルワードをロード	X	31	53
LDX	ロード・ダブルワード索引付き	X	31	21
LFD	倍精度浮動小数点数のロード	D	50	
LFU	Load Floating-Point Double (更新付き)	D	51	
LFduX	Load Floating-Point Double with Update Indexed	X	31	631
LFDX	浮動小数点二重索引のロード	X	31	599

表 *40. PowerPC ®* の説明 *(続き)*				
ニーモニック	**命令**	**Format**	**主要命令コード**	**拡張命令コード**
LFS	単一の浮動小数点のロード	D	48	
LFIU	Load Floating-Point Single with Update (更新付き浮動小数点単一ロード)	D	49	
LFSUX	Load Floating-Point Single with Update Indexed (索引付き更新付きロード浮動小数点単一)	X	31	567
LFSX	浮動小数点単一索引のロード	X	31	535
LHA	ハーフ代数のロード	D	42	
ラウ	更新によるハーフ代数のロード	D	43	
ラックス	Load Half Algebra with Update Indexed (索引付き更新付きハーフ代数のロード)	X	31	375
ラックス	Load Half Algebra Indexed (ハーフ代数索引付きロード)	X	31	343
LHBRX	ハーフバイト反転索引付きロード	X	31	790
LHZ	ロード・ハーフおよびゼロ	D	40	
ラズ	更新による半分とゼロのロード	D	41	
LHZUX	更新索引付きのロード・ハーフおよびゼロ	X	31	331
LHZX	ロード・ハーフおよびゼロ索引付き	X	31	279
LMW	複数ワードのロード	D	46	
L れている	ストリング・ワード即時ロード	X	31	597
LSWX	索引付きストリング・ワードのロード	X	31	533

表 *40. PowerPC ®* の説明 *(続き)*

ニーモニック	命令	Format	主要命令コード	拡張命令コード
ルワ	Load Word Algebra (ロード・ワード代数)	DS	58	2
Lwarx	ロード・ワードおよび予約索引付き	X	31	20
LWAUX	Load Word Algebra with Update Indexed (索引付き更新付きロード・ワード代数)	X	31	373
LWAX	Load Word Algebra Indexed (ロード・ワード代数索引付き)	X	31	341
LWBRX	ワード・バイト反転索引付きロード	X	31	534
LWZ	ワードとゼロのロード	D	32	
LW ツー	ゼロ更新でワードをロード	D	33	
LWZUX	索引付き更新でワードとゼロをロード	X	31	55
LWZX	ロード・ワードおよびゼロ索引	X	31	23
MCRF	移動条件登録フィールド	XL	19	0
MCRFS	FPSCR から条件レジスターへの移動	X	63	64
MCRXR	XER から条件レジスターへの移動	X	31	512
MFC	条件レジスターから移動	X	31	19
マフス [.]	FPSCR から移動	X	63	583
MfMSR	マシン状態レジスターから移動	X	31	83
MFSPR	特殊目的レジスターからの移動	X	31	339
MFS	セグメント・レジスターから移動	X	31	595
ムフスライン	セグメント・レジスター間接から移動	X	31	659
MTCRF	条件登録フィールドに移動	XFX (X)	31	144

表 *40. PowerPC ®* の説明 *(続き)*				
ニーモニック	**命令**	**Format**	**主要命令コード**	**拡張命令コード**
mtfsb0[.]	FPSCR ビット 0 に移動	X	63	70
mtfsb1[.]	FPSCR ビット 1 に移動	X	63	38
mtfsf [.]	FPSCR フィールドへの移動	XFL (X)	63	711
mtfsfi [.]	FPSCR フィールドへの即時移動	X	63	134
MTMSR	マシン状態レジスターに移動	X	31	146
MTSPR	特殊目的レジスターへの移動	X	31	467
MTSR	セグメント・レジスターに移動	X	31	210
ムツリン	セグメント・レジスターへの移動 (間接)	X	31	242
マルチド	高ダブルワードの乗算	Xo	31	73
ムルドゥ	乗算-高ダブルワード符号なし	Xo	31	9
mulhw [.]	上位ワードの乗算	Xo	31	75
mulhwu [.]	高ワード符号なしの乗算	Xo	31	11
マルチルド	下位ダブルワードの乗算	Xo	31	233
ムリ	Multiply Low Immediate	D	07	
mullw [o] [.]	下位ワードの乗算	Xo	31	235
および [.]	ナンド	X	31	476
否定 [o] [.]	否定する (negate)	Xo	31	104
または [.]	NOR	X	31	124
または [.]	または	X	31	444
OrC [.]	OR (コンプリンスあり)	X	31	412
オリ	OR 即時	D	24	
オリス	OR 即時シフト	D	25	
RFi	割り込みから戻る	X	19	50

表 *40. PowerPC ®* の説明 *(続き)*

ニーモニック	命令	Format	主要命令コード	拡張命令コード
RLDCL	ダブルワードを左に回転してから左にクリア	MDS (M)	30	8
RLDCR	左ダブルワードに回転してから右にクリア	MDS (M)	30	9
RLDIC	ダブルワードをすぐに左に回転してから消去	MD	30	2
RLDCL	ダブルワードをすぐに左に回転してから左にクリア	MD	30	0
RLDICR	ダブルワードをすぐに左に回転してから右にクリア	MD	30	1
ルディミ	ダブルワードをすぐ左に回転してからマスクを挿入	MD	30	3
rlwimi [.]	「Rotate Left Word Immediate」、「Mask Insert」	M	20	
rlwinm [.]	「Left Word Immediate の回転 (Rotate Left Word Immediate)」、「マスク (Mask)」	M	21	
rlwnm [.]	左にワードを回転してからマスクと AND を実行	M	23	
sc	システム・コール	SC	17	
si (シフトイン)	即時減算	D	12	
SI	即時およびレコードの減算	D	13	
スラビア	SLB すべて無効化	X	31	498
スビー	SLB 無効化エントリー	X	31	434
SLD	左ダブルワードにシフト	X	31	27
Slw [.]	ワードを左にシフト	X	31	24
SRAD	右方シフト (ダブルワード)	X	31	794

表 *40. PowerPC ®* の説明 *(続き)*				
ニーモニック	**命令**	**Format**	**主要命令コード**	**拡張命令コード**
スラディ	右方へのシフト代数ダブルワード即時	XS	31	413
サード	右ダブルワードにシフト	X	31	539
スロー [.]	右方代数ワードのシフト	X	31	792
スラウィ [.]	右方代数語即値シフト	X	31	824
ソース [.]	右方ワードのシフト	X	31	536
STB	ストア・バイト	D	38	
Stbu	ストア・バイト (更新あり)	D	39	
StBux	索引付き更新でバイトを保管	X	31	247
STBX	索引付き保管バイト	X	31	215
std	ダブルワードの保管	DS	62	0
stdcx (標準)	ダブルワード条件付き索引付き保管	X	31	214
Stdu	更新付きでダブルワードを保管	DS	62	1
Stdux	索引付き更新でダブルワードを保管	X	31	181
標準	ダブルワード索引付き保管	X	31	149
標準	浮動小数点の倍精度浮動小数点数の格納	D	54	
Stfdu	Store Floating-Point Double (更新付き)	D	55	
Stfdux	索引付き更新による浮動小数点の倍精度浮動小数点の保管	X	31	759
StFDX	ストア浮動小数点二重索引付き	X	31	727

表 *40. PowerPC ®* の説明 *(続き)*				
ニーモニック	**命令**	**Format**	**主要命令コード**	**拡張命令コード**
StFiwX	Store Floating-Point as Integer Word Indexed (オプション)	X	31	983
StFS	単一浮動小数点の保管	D	52	
StFsu	ストア・フローティング・ポイント・シングル (更新あり)	D	53	
StFSUX	Store Floating-Point Single with Update Indexed (索引付き更新付きストア浮動小数点単一)	X	31	695
StfsX	ストア浮動小数点単一索引付き	X	31	663
STH	ストア・ハーフ	D	44	
STHBRX	ハーフバイト反転索引付き保管	X	31	918
STHU	更新付きストア・ハーフ	D	45	
STHUX	更新索引付きストア・ハーフ	X	31	439
STHX	ストア・ハーフ索引付き	X	31	407
STMW	複数ワードの保管	D	47	
標準	ストリング・ワード即時保管	X	31	725
Stswx	索引付きストリング・ワードの保管	X	31	661
標準	ストア	D	36	
Stwbrx	ストア・ワード・バイト-逆方向索引付き	X	31	662
stwcx。	条件付き索引付きワードの保管	X	31	150
ストウ	更新付きで Word を保管	D	37	
StWUX	索引付きストア・ワード	X	31	183
StWX	索引付けされたワードの保管	X	31	151

表 40. PowerPC ® の説明 (続き)				
ニーモニック	**命令**	**Format**	**主要命令コード**	**拡張命令コード**
subf [o] [.]	減算元	Xo	31	40
subfc [o] [.]	Subtract from カッリイング	Xo	31	08
サブスクリプション [o] [.]	拡張から減算	Xo	31	136
下位	即時保持からの減算	D	08	
サブフメ [o] [.]	Minus One Extended からの減算	Xo	31	232
サブスクリプション [o] [.]	ゼロ拡張から減算	Xo	31	200
sync	同期化	X	31	598
TD	トラップ・ダブルワード	X	31	68
TDI	Trap Doubleword Immediate (ダブルワード即時トラップ)	D	2	
トルビー	変換ルックアサイド・バッファー無効化項目 (オプション)	X	31	306
TLBSYNC	変換ルックアサイド・バッファー同期 (オプション)	X	31	566
TW	ワードのトラップ	X	31	04
ツイ	トラップ・ワード即時	D	03	
xor [.]	XOR	X	31	316
ソリ	XOR 即時	D	26	
ホリス	XOR 即時シフト	D	27	

付録 G PowerPC 601 RISC マイクロプロセッサー命令

項目	**説明**	**説明**	**説明**	**説明**
PowerPC 601 RISC マイクロプロセッサー **手順**	PowerPC 601 RISC マイクロプロセッサー **手順**	PowerPC 601 RISC マイクロプロセッサー **手順**	PowerPC 601 RISC マイクロプロセッサー **手順**	PowerPC 601 RISC マイクロプロセッサー **手順**
略号 (mnemonic)	**命令**	**フォーマット**	**主要命令コード**	**拡張命令コード**
a [o] [.]	繰越の追加	Xo	31	10

項目	説明	説明	説明	説明
abs [o] [.]	絶対	Xo	31	360
add [o] [.]	追加	Xo	31	266
addc [o] [.]	繰越の追加	Xo	31	10
adde [o] [.]	拡張の追加	Xo	31	138
アディ	即時追加	D	14	
Addic	即時保持の追加	D	12	
Addic。	即時繰越およびレコードの追加	D	13	
追加	即時シフトの追加	D	15	
addme [o] [.]	1 つの拡張をマイナスに追加	Xo	31	234
addze [o] [.]	ゼロ拡張に追加	Xo	31	202
ae [o] [.]	拡張の追加	Xo	31	138
AI	即時追加	D	12	
ai。	即時およびレコードの追加	D	13	
ame [o] [.]	1 つの拡張をマイナスに追加	Xo	31	234
および [.]	AND	X	31	28
および [.]	コンプリンスを伴う AND	X	31	60
アンディー	AND 即時	D	28	
アンディル	AND 直下	D	28	
andis.	AND 即時シフト	D	29	
アンディウ	AND すぐ上	D	29	
AZE [O] [.]	ゼロ拡張に追加	Xo	31	202
b [l] [a]	分岐 (branch)	I	18	
bc [l] [a]	分岐条件付き	B	16	
BCC [L]	カウント・レジスターへの分岐条件付き	XL	19	528
BCCTR [L]	カウント・レジスターへの分岐条件付き	XL	19	528
BCLR [L]	分岐条件リンク・レジスター	XL	19	16
BCR [L]	分岐条件付きレジスター	XL	19	16
cal	アドレスの下限の計算	D	14	

項目	説明	説明	説明	説明
cau	アドレスの上限の計算	D	15	
cax [o] [.]	アドレスの計算	Xo	31	266
Clcs	キャッシュ・ラインの計算サイズ	X	31	531
cmp - 2 つのファイルを比較する	比較	X	31	0
CMPI	即時比較	D	11	
CMPL	論理比較	X	31	32
Cmpli	論理比較 (即時)	D	10	
cntlz [.]	先行ゼロのカウント	X	31	26
cntlzw [.]	先行ゼロ・ワードのカウント	X	31	26
カニ	条件レジスター AND	XL	19	257
Crandc	条件レジスターとコンプリンスを伴う AND	XL	19	129
クレクフ	条件レジスターに相当するもの	XL	19	289
クラナン	条件レジスター NAND	XL	19	225
クナー	条件レジスター NOR	XL	19	33
恐怖	条件レジスター OR	XL	19	449
Crorc	条件登録 OR (完了あり)	XL	19	417
クソル	条件レジスター XOR	XL	19	193
DCBF	Data Cache ブロック・フラッシュ	X	31	86
DCBI	Data Cache ブロック無効化	X	31	470
DCBST	Data Cache ブロック・ストア	X	31	54
DCBT	Data Cache ブロック・タッチ	X	31	278
DCBTST	Data Cache 保管のためのブロック・タッチ	X	31	246

項目	説明	説明	説明	説明
DCBZ	Data Cache ブロックがゼロに設定されました	X	31	1014
DCS	Data Cache 同期化	X	31	598
div [o] [.]	DIVIDE	Xo	31	331
divs [o] [.]	短い分割	Xo	31	363
divw [o] [.]	ワードの分割	Xo	31	491
divwu [o] [.]	ワード符号なし除算	Xo	31	459
doz [o] [.]	差異またはゼロ	Xo	31	264
ドジ	差分またはゼロ即時	D	09	
エシウクス	Word インデックスの外部コントロール	X	31	310
エコックス	外部制御がワード索引を作成しました	X	31	438
エイイオ	入出力の順次実行の強制	X	31	854
eqv [.]	同等	X	31	284
exts [.]	拡張記号	X	31	922
extsb [.]	拡張符号バイト	X	31	954
extsh [.]	拡張符号ハーフワード	Xo	31	922
fa [.]	浮動追加	A	63	21
fabs [.]	浮動絶対値	X	63	264
fadd [.]	浮動追加	A	63	21
fadd [.]	フローティング・シングルの追加	A	59	21
fcir [.]	整数ワードへの浮動変換	X	63	14
fcirz [.]	ゼロへの丸めによる整数ワードへの浮動変換	X	63	15
FC モ	浮動比較順序	X	63	32
FCMPU	非順序浮動比較	XL	63	0
fctiw [.]	整数ワードへの浮動変換	X	63	14

項目	説明	説明	説明	説明
fctiwz [.]	ゼロへの丸めによる整数ワードへの浮動変換	XL	63	15
FD [.]	浮動小数点除算	A	63	18
fdiv [.]	浮動小数点除算	A	63	18
Fdiv [.]	フローティング・ディバイド・シングル	A	59	18
FM [.]	浮動乗算	A	63	25
FMA [.]	浮動乗算-加算	A	63	29
fmadd [.]	浮動乗算-加算	A	63	29
fmadd [.]	浮動乗算-加算単一	A	59	29
fmr [.]	浮動移動レジスター	X	63	72
FMS [.]	浮動乗算-減算	A	63	28
fmsub [.]	浮動乗算-減算	A	63	28
fmsubs [.]	浮動乗算-減算単数	A	59	28
fmul [.]	浮動乗算	A	63	25
fmuls [.]	浮動乗算単数	A	59	25
fnabs [.]	負の浮動絶対値	X	63	136
fneg [.]	浮動否定	X	63	40
FNMA [.]	負の浮動乗算-加算	A	63	31
fnmadd [.]	負の浮動乗算-加算	A	63	31
fnmadd [.]	負の浮動乗算-単一加算	A	59	31
FNMS [.]	負の浮動乗算-減算	A	63	30
fnmsub [.]	負の浮動乗算-減算	A	63	30
fnmsubs [.]	負の浮動乗算-単精度減算	A	59	30
frsp [.]	単精度への浮動丸め	X	63	12
fs [.]	浮動減算	A	63	20
fsub [.]	浮動減算	A	63	20
fsubs [.]	単精度浮動小数点数	A	59	20
イクビ	命令キャッシュ・ブロック無効化	X	31	982
ics	命令キャッシュの同期化	X	19	150
同期	命令の同期化	X	19	150

項目	説明	説明	説明	説明
l	ロード	D	32	
LBRX	ロード・バイト-逆索引付き	X	31	534
LBZ	ロード・バイトおよびゼロ	D	34	
LB ツー	ロード・バイトおよびゼロ (更新あり)	D	35	
LBZUX	索引付き更新でバイトおよびゼロをロード	X	31	119
LBZX	ロード・バイトおよびゼロ索引	X	31	87
LFD	倍精度浮動小数点数のロード	D	50	
LFU	Load Floating-Point Double (更新付き)	D	51	
LFduX	Load Floating-Point Double with Update Indexed	X	31	631
LFDX	浮動小数点二重索引のロード	X	31	599
LFS	単一の浮動小数点のロード	D	48	
LFIU	Load Floating-Point Single with Update (更新付き浮動小数点単一ロード)	D	49	
LFSUX	Load Floating-Point Single with Update Indexed (索引付き更新付きロード浮動小数点単一)	X	31	567
LFSX	浮動小数点単一索引のロード	X	31	535
LHA	ハーフ代数のロード	D	42	
ラウ	更新によるハーフ代数のロード	D	43	
ラックス	Load Half Algebra with Update Indexed (索引付き更新付きハーフ代数のロード)	X	31	375

項目	説明	説明	説明	説明
ラックス	Load Half Algebra Indexed (ハーフ代数索引付きロード)	X	31	343
LHBRX	ハーフバイト反転索引付きロード	X	31	790
LHZ	ロード・ハーフおよびゼロ	D	40	
ラズ	更新による半分とゼロのロード	D	41	
LHZUX	更新索引付きのロード・ハーフおよびゼロ	X	31	331
LHZX	ロード・ハーフおよびゼロ索引付き	X	31	279
LM	複数ロード	D	46	
LMW	複数ワードのロード	D	46	
LSCBX	ストリングのロードとバイト索引の比較	X	31	277
LSI	Load String Immediate (ストリング即時ロード)	X	31	597
L れている	ストリング・ワード即時ロード	X	31	597
LSWX	索引付きストリング・ワードのロード	X	31	533
LsX	ロード・ストリング索引付き	X	31	533
Lu	更新によるロード	D	33	
ルクス	索引付き更新でロード	X	31	55
Lwarx	ロード・ワードおよび予約索引付き	X	31	20
LWBRX	ワード・バイト反転索引付きロード	X	31	534
LWZ	ワードとゼロのロード	D	32	
LW ツー	ゼロ更新でワードをロード	D	33	
LWZUX	索引付き更新でワードとゼロをロード	X	31	55

項目	説明	説明	説明	説明
LWZX	ロード・ワードおよびゼロ索引	X	31	23
LX	索引付きロード	X	31	23
maskg [.]	マスク生成	X	31	29
マスキル [.]	レジスターからのマスク挿入	X	31	541
MCRF	移動条件登録フィールド	XL	19	0
MCRFS	FPSCR から条件レジスターへの移動	X	63	64
MCRXR	XER から条件レジスターへの移動	X	31	512
MFC	条件レジスターから移動	X	31	19
マフス [.]	FPSCR から移動	X	63	583
MfMSR	マシン状態レジスターから移動	X	31	83
MFSPR	特殊目的レジスターからの移動	X	31	339
MFS	セグメント・レジスターから移動	X	31	595
ムフスライン	セグメント・レジスター間接から移動	X	31	659
MTCRF	条件登録フィールドに移動	XFX (X)	31	144
mtfsb0[.]	FPSCR ビット 0 に移動	X	63	70
mtfsb1[.]	FPSCR ビット 1 に移動	X	63	38
mtfsf [.]	FPSCR フィールドへの移動	XFL (X)	63	711
mtfsfi [.]	FPSCR フィールドへの即時移動	X	63	134
MTMSR	マシン状態レジスターに移動	X	31	146
MTSPR	特殊目的レジスターへの移動	X	31	467
MTSR	セグメント・レジスターに移動	X	31	210
MTSRI	セグメント・レジスターへの移動 (間接)	X	31	242

項目	説明	説明	説明	説明
ムツリン	セグメント・レジスターへの移動 (間接)	X	31	242
mul [o] [.]	MULTIPLY	Xo	31	107
mulhw [.]	上位ワードの乗算	Xo	31	75
mulhwu [.]	高ワード符号なしの乗算	Xo	31	11
ムリ	乗算-即時	D	07	
ムリ	Multiply Low Immediate	D	07	
mullw [o] [.]	下位ワードの乗算	Xo	31	235
muls [o] [.]	短時間の乗算	Xo	31	235
nabs [o] [.]	負の絶対値	Xo	31	488
および [.]	ナンド	X	31	476
否定 [o] [.]	否定する (negate)	Xo	31	104
または [.]	NOR	X	31	124
または [.]	または	X	31	444
OrC [.]	OR (コンプリンスあり)	X	31	412
オリ	OR 即時	D	24	
オリル	OR 直下	D	24	
オリス	OR 即時シフト	D	25	
オリウ	OR 即時 (大文字)	D	25	
RFi	割り込みから戻る	X	19	50
リミ [.]	すぐに左に回転してから挿入をマスク	M	20	
rlinm [.]	「Rotate Left Immediate then AND with Mask」	M	21	
RLMI [.]	左に回転してから挿入をマスク	M	22	
rlnm [.]	左に回転してマスクと AND	M	23	
rlwimi [.]	「Rotate Left Word Immediate」、「Mask Insert」	M	20	

項目	説明	説明	説明	説明
rlwinm [.]	「Left Word Immediate の回転 (Rotate Left Word Immediate)」、「マスク (Mask)」	M	21	
rlwnm [.]	左にワードを回転してからマスクと AND を実行	M	23	
rrib [.]	右に回転してビットを挿入	X	31	537
sc	システム・コール	SC	17	
sf [o] [.]	減算元	Xo	31	08
sfe [o] [.]	拡張から減算	Xo	31	136
SFI	即時から減算	D	08	
sfme [o] [.]	Minus One Extended からの減算	Xo	31	232
sfze [o] [.]	ゼロ拡張から減算	Xo	31	200
si (シフトイン)	即時減算	D	12	
SI	即時およびレコードの減算	D	13	
SSL [.]	左にシフト	X	31	24
(sle [.])	左にシフト (拡張)	X	31	153
スレッド [.]	MQ を使用して左にシフトを拡張	X	31	217
sliq [.]	MQ での左への即時シフト	X	31	184
slliq [.]	MQ での長時間即時シフト	X	31	248
SLLQ [.]	MQ での Shift Left Long	X	31	216
slq [.]	MQ を左にシフト	X	31	152
Slw [.]	ワードを左にシフト	X	31	24
SR [.]	右にシフト	X	31	536
スラ [.]	右方代数のシフト	X	31	792
SAI [.]	右方代数即値シフト	X	31	824
sraiq [。]	MQ を使用した右方代数の即時シフト	X	31	952
スラック [.]	MQ を使用した右方代数のシフト	X	31	920

項目	説明	説明	説明	説明
スロー [.]	右方代数ワードのシフト	X	31	792
スラウィ [.]	右方代数語即値シフト	X	31	824
SRE [.]	右にシフト (拡張)	X	31	665
SREA [.]	右方拡張代数シフト	X	31	921
SREQ [.]	MQ による右へのシフトの拡張	X	31	729
Sriq [.]	MQ による右への即時シフト	X	31	696
srliq [.]	MQ による右方への即時シフト	X	31	760
srlq [.]	MQ でのシフト・ライト・ロング	X	31	728
ソース [.]	MQ での Shift RIcatch	X	31	664
ソース [.]	右方ワードのシフト	X	31	536
st	ストア	D	36	
STB	ストア・バイト	D	38	
Stbrx	ストア・バイト-逆方向索引付き	X	31	662
Stbu	ストア・バイト (更新あり)	D	39	
StBux	索引付き更新でバイトを保管	X	31	247
STBX	索引付き保管バイト	X	31	215
標準	浮動小数点の倍精度浮動小数点数の格納	D	54	
Stfdu	Store Floating-Point Double (更新付き)	D	55	
Stfdux	索引付き更新による浮動小数点の倍精度浮動小数点の保管	X	31	759
StFDX	ストア浮動小数点二重索引付き	X	31	727
StFS	単一浮動小数点の保管	D	52	

項目	説明	説明	説明	説明
StFsu	ストア・フローティング・ポイント・シングル (更新あり)	D	53	
StFSUX	Store Floating-Point Single with Update Indexed (索引付き更新付きストア浮動小数点単一)	X	31	695
StfsX	ストア浮動小数点単一索引付き	X	31	663
STH	ストア・ハーフ	D	44	
STHBRX	ハーフバイト反転索引付き保管	X	31	918
STHU	更新付きストア・ハーフ	D	45	
STHUX	更新索引付きストア・ハーフ	X	31	439
STHX	ストア・ハーフ索引付き	X	31	407
STM	複数保管	D	47	
STMW	複数ワードの保管	D	47	
STSI	ストア・ストリング即時	X	31	725
標準	ストリング・ワード即時保管	X	31	725
Stswx	索引付きストリング・ワードの保管	X	31	661
STSX	索引付きストア・ストリング	X	31	661
ストゥー	ストア (更新あり)	D	37	
スタックス	索引付き更新で保管	X	31	183
標準	ストア	D	36	
Stwbrx	ストア・ワード・バイト-逆方向索引付き	X	31	662
stwcx。	条件付き索引付きワードの保管	X	31	150
ストウ	更新付きで Word を保管	D	37	
StWUX	索引付きストア・ワード	X	31	183

項目	説明	説明	説明	説明
StWX	索引付けされたワードの保管	X	31	151
stx (テキスト開始)	索引付きストア	X	31	151
subf [o] [.]	減算元	Xo	31	40
subfc [o] [.]	Subtract from カッリイング	Xo	31	08
サブスクリプション [o] [.]	拡張から減算	Xo	31	136
下位	即時保持からの減算	D	08	
サブフメ [o] [.]	Minus One Extended からの減算	Xo	31	232
サブスクリプション [o] [.]	ゼロ拡張から減算	Xo	31	200
sync	同期化	X	31	598
t	trap	X	31	04
TI	即時トラップ	D	03	
トルビー	変換ルックアサイド・バッファー無効化項目	X	31	306
TW	ワードのトラップ	X	31	04
ツイ	トラップ・ワード即時	D	03	
xor [.]	XOR	X	31	316
ソリ	XOR 即時	D	26	
ホリル	XOR 即値下限	D	26	
ホリス	XOR 即時シフト	D	27	
Xoriu	XOR 即時上限	D	27	

付録 H の値の定義

ビット 0 から 5

これらのビットは、マシン・インストラクションの命令コード部分を表します。

ビット 6-30

これらのビットには、以下の値に従って定義されたフィールドが含まれます。 多くの命令には、この範囲のビットの一部を占める拡張命令コードも含まれていることに注意してください。 使用される形式を理解するには、具体的な説明を参照してください。

値	定義
/, //, ///	予約済み/未使用。名目上はゼロ (0)。

値	定義
A	一部のダイアグラムでの RA の偽名。
AA	絶対アドレス・ビット。 • 0-即値フィールドは、現行命令アドレスに対する相対アドレスを表します。 • 1-即時フィールドは絶対アドレスを表します。
B	一部のダイアグラムでの RB の変名。
BA	操作のソース条件レジスター・ビットを指定します。
BB	操作のソース条件レジスター・ビットを指定します。
BD	分岐変位として使用される 14 ビット値を指定します。
BF	比較の結果を示す条件レジスター・フィールド 0 から 7 を指定します。
BFA とは	操作のソース条件レジスター・フィールドを指定します。
BI	条件比較の条件レジスター内のビットを指定します。
BO	命令で使用される分岐オプション・フィールドを指定します。
BT	操作の結果が保管されるターゲット条件レジスター・ビットを指定します。
D	32 ビットに拡張された 16 ビットの 2 の補数整数符号を指定します。
DS	有効アドレス (EA) の計算の即時値として使用される 14 ビット・フィールドを指定します。
FL1	SVC ルーチンを渡すオプション・データのフィールドを指定します。
FL2	SVC ルーチンを渡すオプション・データのフィールドを指定します。
FLM (LM)	フィールド・マスクを指定します。
FRA	操作のソース浮動小数点レジスターを指定します。
FRB	操作のソース浮動小数点レジスターを指定します。
FRC	操作のソース浮動小数点レジスターを指定します。
FRS	保管データのソース浮動小数点レジスターを指定します。
FRT (R)	操作のターゲット浮動小数点レジスターを指定します。
FXM (M)	フィールド・マスクを指定します。
I	操作のソース即値を指定します。
L	32 ビット・サブセット・アーキテクチャーの場合は 0 に設定する必要があります。
LEV	実行アドレスを指定します。
LI	右側に 0b00 と連結され、64 ビットに符号拡張された 24 ビット符号付き 2 の補数整数を指定する即時フィールド (32 ビット実装の場合は 32 ビット)。
LK	LK=1 の場合、分岐命令に続く命令の有効アドレスがリンク・レジスターに入れられます。
MB	操作のマスクの開始値 (ビット番号) を指定します。
ME	操作のマスクの終了値 (ビット番号) を指定します。
NB (B)	操作のバイト・カウントを指定します。
大江	操作の結果がオーバーフローになった場合に、固定小数点例外レジスターのオーバーフロー・ビットが影響を受けることを指定します。
RA	操作のソース汎用レジスターを指定します。

値	定義
RB	操作のソース汎用レジスターを指定します。
RS	操作のソース汎用レジスターを指定します。
RT	操作が保管されるターゲット汎用レジスターを指定します。
S	一部のダイアグラムでの RS の偽名。
SA	svc 命令に記載されています。
SH	操作の (即時) シフト値を指定します。
SI	演算の 16 ビットの符号付き整数を指定します。
SIMM	比較のために符号拡張される 16 ビット 2 の補数値。
SPR	操作のソース特殊目的レジスターを指定します。
SR	操作のソース・セグメント・レジスターを指定します。
ST	操作のターゲット・セグメント・レジスターを指定します。
終了	比較結果と AND 演算される TO ビットを指定します。
U	操作のソース即値を指定します。
UI	演算用の 16 ビット符号なし整数を指定します。

ビット 31

ビット 31 はレコード・ビットです。

値	定義
0	条件レジスターを更新しません。
1	操作の結果を反映するように条件レジスターを更新します。

付録 I ベクトル・プロセッサー

この付録では、ベクトル・プロセッサーの概要と、ベクトル・プロセッサーをサポートする AIX® ABI 拡張機能およびリンケージ規則について説明します。

関連情報

AIX ベクトル・プログラミング

ストレージ・オペランドと位置合わせ

ベクトル・データ型のサイズは 16 バイトであり、16 バイト (4 倍長ワード) 境界に位置合わせする必要があります。

すべてのベクトル・データ型は、サイズが 16 バイトであり、16 バイト (4 倍長ワード) 境界に位置合わせされている必要があります。 ベクトル・タイプを含む集合体 は、通常の規約 (その最大のメンバーの要件に集合体を位置合わせする) を遵 守する必要があります。 ベクトル・タイプを含む集合体がパックされている場合、そのベクトル ・タイプの 16 バイト位置あわせは保証されません。

表 *41.* データ・タイプ

内容	新規 C/C++ タイプ
16 符号なし文字	ベクトル符号なし char
16 符号付き文字	ベクトル符号付き char
16 符号なし文字	ベクター・ブール文字

表 41. データ・タイプ (続き)	
内容	**新規 C/C++ タイプ**
8 個の無符号簡略	ベクトル無符号簡略
8 個の有符号簡略	ベクトル有符号簡略
8 個の無符号簡略	ベクトル・ブール簡略
4 符号なし整数	ベクトル符号なし整数
4 符号付き整数	ベクトル符号付き int
4 符号なし整数	ベクトルブール整数
4 個の浮動小数点	ベクトル浮動小数点

レジスターの使用規則

PowerPC ベクトル拡張アーキテクチャーは、32 個のベクトル・レジスター (VR) を追加します。

PowerPC Vector Extension Architecture は、32 個のベクトル・レジスター (VR) を追加します。 各 VR の幅は 128 ビットです。 また、32 ビット特殊目的レジスター (VRSAVE) と、32 ビット・ベクトル状況および制御レジスター (VSCR) もあります。 VR 規則表は、VR がどのように使用されるかを示しています。

表 42. *VR* 規則		
レジスター	**状況**	**Use**
VR0	揮発性	スクラッチ・レジスター。
VR1	揮発性	スクラッチ・レジスター。
VR2	揮発性	最初のベクトル引数。 関数の戻り値の最初のベクトル。
VR3	揮発性	2 番目のベクトル引数、スクラッチ。
VR4	揮発性	3 番目のベクトル引数、スクラッチ。
VR5	揮発性	4 番目のベクトル引数、スクラッチ。
VR6	揮発性	5 番目のベクトル引数、スクラッチ。
VR7	揮発性	6 番目のベクトル引数、スクラッチ。
VR8	揮発性	7 番目のベクトル引数、スクラッチ。
VR9	揮発性	8 番目のベクトル引数、スクラッチ。
VR10	揮発性	9 番目のベクトル引数、スクラッチ。
VR11	揮発性	10 番目のベクトル引数、スクラッチ。
VR12	揮発性	11 番目のベクトル引数、スクラッチ。
VR13	揮発性	12 番目のベクトル引数、スクラッチ。
VR14:19	揮発性	スクラッチ。
VR20:31	予約済み (デフォルト・モード) 不揮発性 (拡張 ABI モード)	デフォルトのベクトル使用可能モードを使用する場合、これらのレジスターは予約済みであり、使用してはなりません。 拡張 ABI ベクトル使用可能モードでは、これらのレジスターは不揮発性であり、その値は関数呼び出し間で保持されます。
VRSAVE	予約済み	AIX® ABI では、VRSAVE は使用されません。 ABI 準拠プログラムは VRSAVE を使用または変更することはできません。

表 *42. VR* 規則 *(*続き*)*		
レジスター	**状況**	**Use**
VSCR	揮発性	ベクトル状況および制御レジスター。 飽和状況ビットおよび Java™ モード制御ビット以外が含まれます。

AltiVec プログラミング・インターフェース仕様は、使用されるベクトル・レジスターのビット・マスクとして使用する VRSAVE レジスターを定義しています。 AIX® では、アプリケーションが VRSAVE レジスターを変更しないことが必要です。

ランタイム・スタック

ランタイム・スタックは、32 ビット・プロセスと 64 ビット・プロセスの両方で位置合わせされた 4 倍長ワードで始まります。

ランタイム・スタックは、32 ビット・プロセスと 64 ビット・プロセスの両方で位置合わせされた 4 倍長ワードで始まります。 以下の 4 つの段落で説明されている規則は、VR のスタック保管域、およびスタックで渡されるベクトル・パラメーターの規則について定義されています。

VRSAVE は、 AIX® ABI によって認識されません。また、ABI 準拠のプログラムによって使用または変更してはなりません。 VRSAVE ランタイム・スタック保管場所は、レガシー・コンパイラー・リンケージ規則との互換性のために予約されたままです。

位置合わせ埋め込みスペースは、ベクトル保管域を 4 倍長ワード境界に位置合わせするために、必要に応じて 0、4、8、または 12 バイトのいずれかになります。 不揮発性 VR を使用する前に、スタック上のその VR 保管域に VR31 から始めて、VR20 まで保管しておく必要があります。 メモリーに保存する必要があるベクトル・データ型のローカル変数は、他の型のローカル変数に使用されるものと同じスタック・フレーム領域に保存されますが、16 バイト境界上に保存されます。

スタック・フロアは、32 ビット・モードの場合は 220 バイト、64 ビット・モードの場合は 288 バイトのままです。 関数が不揮発性汎用レジスター (GPR)、浮動小数点レジスター (FPR)、およびそれぞれのモードのフロア・サイズより合計が多い VR を保管する必要がある場合、関数は、不揮発性 VR を保管する前に、まずスタック・ポインターをアトミックに更新する必要があります。

ローカル変数領域内のベクトル変数は、16 バイト境界に位置合わせする必要があります。

32 ビット・ランタイム・スタックは、以下のようになります (プロローグ前)。

表 *43. 32* ビット・ランタイム・スタックの例	
項目	**説明**
SP->	バック・チェーン
	FPR31 (必要な場合)
	...
-nFPRs*8	...
	GPR31 (必要な場合)
	...
-nGPRs*4	...
	VRSAVE
	位置合わせ埋め込み (16 バイト境界まで)
	VR31 (必要な場合)
-nVRs*16	...

表 *43. 32* ビット・ランタイム・スタックの例 (続き)	
項目	**説明**
-220 (最大)	...
	ローカル変数
NF+24	パラメーター・リスト域
NF+20	保存された TOC
NF+16	予約済み (バインダー)
NF+12	予約済み (コンパイラー)
NF+8	保存された LR
NF+4	保存された CR
NF->	Sp (NF-newframe が割り振られた後)

64 ビット・ランタイム・スタックは、以下のようになります (プロローグ前)。

表 *44. 64* ビット・ランタイム・スタックの例	
項目	**説明**
SP->	バック・チェーン
	FPR31 (必要な場合)
	...
-nFPRs*8	...
	GPR31 (必要な場合)
	...
-nGPRs*8	...
	VRSAVE
	位置合わせ埋め込み (16 バイト境界まで)
	VR31 (必要な場合)
-nVRs*16	...
-288 (最大)	...
	ローカル変数
NF+48	パラメーター・リスト域
NF+40	保存された TOC
NF+32	予約済み (バインダー)
NF+24	予約済み (コンパイラー)
NF+16	保存された LR
NF+8	保存された CR
NF->	Sp (NF-newframe が割り振られた後)

ベクトル・レジスター保管および復元プロシージャー

ベクトル保管および復元機能は、言語コンパイラーの補助としてシステム (libc) によって提供されます。

下記のベクトル保管および復元機能は、言語コンパイラーの補助としてシステム (libc) によって提供されます。

入り口では、 **r0** には、ベクトル保管域のすぐ上に位置合わせされた 16 バイトのアドレスが入っていなければなりません。 **r0** は変更されませんが、 **r12** は変更されます。

項目	説明

```
_savev20:   addi    r12,    r0,     -192
            stvx    v20,    r12,    r0      # save v20
_savev21:   addi    r12,    r0,     -176
            stvx    v21,    r12,    r0      # save v21
_savev22:   addi    r12,    r0,     -160
            stvx    v22,    r12,    r0      # save v22
_savev23:   addi    r12,    r0,     -144
            stvx    v23,    r12,    r0      # save v23
_savev24:   addi    r12,    r0,     -128
            stvx    v24,    r12,    r0      # save v24
_savev25:   addi    r12,    r0,     -112
            stvx    v25,    r12,    r0      # save v25
_savev26:   addi    r12,    r0,     -96
            stvx    v26,    r12,    r0      # save v26
```

項目	説明				
_savev27:	addi	r12,	r0,	-80	
	stvx	v27,	r12,	r0	# save v27
_savev28:	addi	r12,	r0,	-64	
	stvx	v28,	r12,	r0	# save v28
_savev29:	addi	r12,	r0,	-48	
	stvx	v29,	r12,	r0	# save v29
_savev30:	addi	r12,	r0,	-32	
	stvx	v30,	r12,	r0	# save v30
_savev31:	addi	r12,	r0,	-16	
	stvx	v31,	r12,	r0	# save v31
	br				
_restv20:	addi	r12,	r0,	-192	
	lvx	v20,	r12,	r0	# restore v20
_restv21:	addi	r12,	r0,	-176	
	lvx	v21,	r12,	r0	# restore v21
_restv22:	addi	r12,	r0,	-160	

項目	説明				
	lvx	v22,	r12,	r0	# restore v22
_restv23:	addi	r12,	r0,	-144	
	lvx	v23,	r12,	r0	# restore v23
_restv24:	addi	r12,	r0,	-128	
	lvx	v24,	r12,	r0	# restore v24
_restv25:	addi	r12,	r0,	-112	
	lvx	v25,	r12,	r0	# restore v25
_restv26:	addi	r12,	r0,	-96	
	lvx	v26,	r12,	r0	# restore v26
_restv27:	addi	r12,	r0,	-80	
	lvx	v27,	r12,	r0	# restore v27
_restv28:	addi	r12,	r0,	-64	
	lvx	v28,	r12,	r0	# restore v28
_restv29:	addi	r12,	r0,	-48	
	lvx	v29,	r12,	r0	# restore v29
_restv30:	addi	r12,	r0,	-32	
	lvx	v30,	r12,	r0	# restore v30

項目	説明				
`_restv31:`	`addi`	`r12,`	`r0,`	`-16`	
	`lvx`	`v31,`	`r12,`	`r0`	`# restore v31`
	`br`				

プロシージャー呼び出しシーケンス

引数の受け渡しと戻り値に関するプロシージャー呼び出し規則。

以下のセクションでは、引数の引き渡しと戻り値に関するプロシージャー呼び出し規則について説明します。

引数の引き渡し

機能に対する最初の 12 個のベクトル・パラメーターは、レジスター VR2 から VR13 に入れられます。

機能に対する最初の 12 個のベクトル・パラメーターは、レジスター VR2 から VR13 に入れられます。 不要なベクトル・パラメーター・レジスターには、function.Non への入り口で未定義の値が含まれています。 可変長引数リスト・ベクトル・パラメーターは、GPR でシャドーイングされません。 追加のベクトル・パラメーター (13th 以上) は、パラメーター・リスト内の位置に対応するパラメーター領域内の適切なマップされた位置に、プログラム・スタック上の 16 バイト位置合わせされたメモリーを介して渡されます。

可変長の引数リストの場合、**va_list** は引き続き次のパラメーターのメモリー・ロケーションへのポインターになります。 **va_arg()** がベクトル・タイプにアクセス する際、**va_list** はまず 16 バイト境界に位置合わせされる必要があります。 可変長引数リストの受信側とコンシューマーは、ベクトル型パラメーターを検索する前に、この位置合わせを行う必要があります。

値によって渡され、その中のどこかにベクトル・メンバーがある非パック構造体または共用体は、スタック上の 16 バイト境界に位置合わせされます。

可変長引数リストを取る関数には、引数領域にマップされたすべてのパラメーターが、そのタイプに従って順序付けされ、位置合わせされます。 可変長引数リストの最初の 8 ワード (32 ビット) またはダブルワード (64 ビット) は、GPR **r3** から **r10** にシャドーイングされます。 これには、ベクトル・パラメーターが含まれます。 以下の表は、可変長引数リスト・パラメーターを示しています。

表 *45. 32* ビット可変長引数リスト・パラメーター *(*ポストプロローグ*)*

項目	説明	
旧 Sp->	バック・チェーン (bc)	
-nFPRs*8	...	
-nGPRs*4	...	
-220 (最大)	VRSAVE	
	ローカル変数	
SP+56	...	
SP+52	PW7	ベクトル・パラメーター 2b、GPR10 内のシャドー
SP+48	PW6	ベクトル･パラメーター 2a、GPR9 のシャドー

表 45. 32 ビット可変長引数リスト・パラメーター (ポストプロローグ) (続き)		
項目	**説明**	
SP+44	PW5	ベクトル・パラメーター 1d、GPR8 のシャドー
SP+40	PW4	ベクトル・パラメーター 1c、GPR7 のシャドー
SP+36	PW3	ベクトル・パラメーター 1b、GPR6 内のシャドー
SP+32	PW2	ベクトル・パラメーター 1a、GPR5 のシャドー
SP+28	PW1	
SP+24	PW0	
SP+20	保存された TOC	
SP+16	予約済み (バインダー)	
SP+12	予約済み (コンパイラー)	
SP+8	保存された LR	
SP+4	保存された CR	
SP->	オールド SP	

表 46. 64 ビット可変長引数リスト・パラメーター (ポストプロローグ)		
項目	**説明**	
旧 Sp->	バック・チェーン (bc)	
-nFPRs*8	...	
-nGPRs*8	...	
-288 (最大)	VRSAVE	
	ローカル変数	
SP+112	...	
SP+104	PW7	ベクトル・パラメーター 4c、 4d、 GPR10 内のシャドー
SP+96	PW6	ベクトル・パラメーター 4a、 4b、 GPR9 のシャドー
SP+88	PW5	ベクトル・パラメーター 3c、 3d、 GPR8 のシャドー
SP+80	PW4	ベクトル・パラメーター 3a、 3b、 GPR7 内のシャドー
SP+72	PW3	ベクトル・パラメーター 2c、 2d、 GPR6 のシャドー
SP+64	PW2	ベクトル・パラメーター 2a、 2b、 GPR5 内のシャドー
SP+56	PW1	ベクトル・パラメーター 1c、 1d、 GPR4 内のシャドー

表 *46. 64* ビット可変長引数リスト・パラメーター *(*ポストプロローグ*) (*続き*)*		
項目	**説明**	
SP+48	PW0	ベクトル・パラメーター 1a、 1b、 GPR3 のシャドー
SP+40	保存された TOC	
SP+32	予約済み (バインダー)	
SP+24	予約済み (コンパイラー)	
SP+16	保存された LR	
SP+8	保存された CR	
SP->	オールド SP	

関数からの戻り値

ベクトル型を戻す関数、またはベクトル・パラメーターを持つ関数には、関数プロトタイプが必要です。

ベクトル・データ型として宣言された戻り値を持つ関数は、その戻り値を VR2 上に置きます。 ベクトル・タイプを戻すか、またはベクトル・パラメーターを 持つ関数にはすべて、関数プロトタイプが必要です。 これにより、コンパイラーは一般的なケースのために GPR 内の VR をシャドー処理する必要がなくなります。

トレースバック・テーブル

トレースバック・テーブルは、関数のスタック・フレーム内にベクトル状態が存在するかどうかを判別するために必要な情報を提供します。

トレースバック・テーブル情報は、関数のスタック・フレーム内にベクトル状態が存在するかどうかを判別するために必要な情報を提供するように拡張されています。 **spare3** フィールドからの未使用ビットの 1 つが、トレースバック・テーブルにベクトル情報が含まれていることを示すために要求されます。 そのため、必須のトレースバック・テーブル情報に以下の変更が加えられます。

項目	**説明**
`unsigned spare3:1;`	`/* Spare bit */`
`unsigned has_vec:1;`	`/* Set if optional vector info is present */`

has_vec フィールドが設定されている場合は、オプションの **parminfo** フィールドと、以下のオプションの拡張情報が表示されます。 新しいオプション・ベクトル情報が存在する場合は、他の定義済みオプション・フィールドの後に続き、 **alloca_reg** オプション情報の後になります。

項目	**説明**
`unsigned vr_saved:6;`	`/* Number of non-volatile vector registers saved */`
	`/* first register saved is assumed to be */`
	`/* 32 - vr_saved */`

項目	説明
`unsigned saves_vrsave:1;`	`/* Set if vrsave is saved on the stack */`
`unsigned has_varargs:1;`	`/* Set if the function has a variable length argument list */`
`unsigned vectorparms:7;`	`/* number of vector parameters if not variable */`
	`/* argument list.  Otherwise the mandatory field*/`
	`/* parmsonstk field must be set */`
`unsigned vec_present:1;`	`/* Set if routine performs vector instructions */`
`unsigned char vecparminfo[4];`	`/* bitmask array for each vector parm in */`
	`/* order as found in the original parminfo, */`
	`/* describes the type of vector: */`
	`/*  b '00 = vector char  */`
	`/*  b '01 = vector short  */`
	`/*  b '10 = vector int  */`
	`/*  b '11 = vector float  */`
vectorparms がゼロ以外の場合、 **parminfo** フィールドは次のように解釈されます。	**vectorparms** がゼロ以外の場合、 **parminfo** フィールドは次のように解釈されます。
	`/*  b '00' = fixed parameter  */`
	`/*  b '01' = vector parameter  */`
	`/*  b '10' = single-precision float parameter  */`

項目	説明
	`/*  b '11' = double-precision float parameter  */`

デバッグ・スタブ・ストリング

stabstring コードは、VR 内のオブジェクトのロケーションを指定するために定義されます。

VR 内のオブジェクトの位置を指定するために、新しいスタブ・ストリング・コードが定義されます。 "X" のコードは、指定されたベクトル・レジスターで値によって渡されるパラメーターを記述します。 "x" のコードは、指定された VR にあるローカル変数を記述します。 C_LSYM (スタック上のローカル変数)、C_PSYM (スタック上のパラメーター) の既存のストレージ・クラスは、メモリー内のベクトル・データ型に使用されます。この場合、対応するスタブ・ストリングは、データを表すために既存の基本型の配列を使用します。 C_RPSYM (レジスター内のパラメーター) および C_RSYM (レジスター内の変数) の既存のストレージ・クラスは、ベクトル・レジスター内のベクトル・データ・タイプを表すために、それぞれ新しいスタブ・ストリング・コード 'X' および' x ' と一緒に使用されます。

レガシー ABI 互換性およびインターオペラビリティー

レガシー ABI 互換性およびインターオペラビリティー。

setjmp ()、 **longjmp ()**、 **sigsetjmp ()**、 **siglongjmp ()**、 **_setjmp ()**、 **_longjmp () (_R)**、 **getcontext ()**、 **setcontext ()**、 **makecontext ()**、および **スワップ・コンテキスト ()**などのインターフェースの性質により、不揮発性マシン状態を保存および復元する必要があるため、レガシー ABI モジュールとベクトル拡張 ABI モジュールの間の依存関係を考慮する際にリスクが生じます。 問題を複雑にするために、 **libc** の **setjmp** ファミリーの関数は、 **libc** の静的メンバーに常駐します。これは、既存のすべての AIX® バイナリーに、リンク先の AIX® のバージョンで存在していた **setjmp** の静的にバインドされたコピーがあることを意味します。 さらに、既存の AIX® バイナリーには、追加の不揮発性ベクトル・レジスター状態を格納するには不十分な **jmpbufs** および **ucontext** データ構造定義があります。

前のバージョンのモジュールと新規モジュールが呼び出しまたはコールバックをインターリーブする場合、前のバージョンのモジュールが、ベクトル拡張モジュールの通常のリンケージ規約をバイパスして **longjmp ()** または **setcontext ()** を実行する可能性がある場合は、不揮発性 VR 状態になるリスクがあります。

このため、 AIX® ABI は不揮発性 VR を定義しますが、 AIX® コンパイラーでベクトル (AltiVec) を使用する場合のデフォルトのコンパイル・モードは、不揮発性 VR を使用しません。 これにより、デフォルトのコンパイル環境が安全にベクトル (AltiVec) を利用できるようになりますが、前のバージョンのバイナリーとのインターオペラビリティーに関するリスクは生じません。

インターオペラビリティーとモジュール依存性が完全に認識されているアプリケーションの場合、不揮発性 VR の使用を可能にする追加のコンパイル・オプションを使用可能にすることができます。 このモードは、すべての従属する前のバージョンのモジュールと動作が完全に認識され、 **setjmp ()**、 **sigsetjmp ()**、 **_setjmp ()**、または **getcontext ()**などの関数に依存しないか、あるいはすべてのモジュール遷移が通常のサブルーチン・リンケージ規約によって実行されないように理解されている場合にのみ使用してください。

このアプローチにより、デフォルト・モードであるベクトル (AltiVec) を完全にセーフ・モードで活用することができます。また、リスクが判明している場合には、不揮発性レジスターを使用して明示的なチューニングとさらなる最適化を行うこともできます。 また、将来のための柔軟な ABI とアーキテクチャーも提供します。

デフォルトの AltiVec コンパイル環境は、「*AltiVec Programming Interface Manual*」で説明されているように、 **__VEC__**を事前定義します。

不揮発性 VR を使用するオプションが使用可能な場合、コンパイル環境も **__EXTABI__**を事前定義する必要があります。 これは、不揮発性 VR を使用できるようになっているベクトル対応モジュールと対話する非ベクトル対応モジュールをコンパイルまたは再コンパイルするときにも定義する必要があります。

特記事項

本書は米国が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。 日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。 本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。 これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。 ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。 本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。 実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒 *103-8510*

東京都中央区日本橋箱崎町 *19* 番 *21* 号

日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing

Legal and Intellectual
Property Law
IBM Japan Ltd.

19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。 国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。 本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。 それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

〒 *103-8510*

東京都中央区日本橋箱崎町 *19* 番 *21* 号

日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

記載されている性能データとお客様事例は、例として示す目的でのみ提供されています。 実際の結果は特定の構成や稼働条件によって異なります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述は、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

表示されている IBM の価格は IBM が小売り価格として提示しているもので、現行価格であり、通知なしに変更されるものです。 卸価格は、異なる場合があります。

本書はプランニング目的としてのみ記述されています。 記述内容は製品が使用可能になる前に変更になる場合があります。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。 より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、類似する個人や企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。 お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。 このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。 従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。 これらのサンプル・プログラムは特定物として現存するままの状態で提供されるものであり、いかなる保証も提供されません。 IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生した創作物には、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (年).

このコードの一部は、IBM Corp. から取られています。 サンプル・プログラム

© Copyright IBM Corp. _年を入れる_.

プライバシー・ポリシーに関する考慮事項

サービス・ソリューションとしてのソフトウェアも含めた IBM® ソフトウェア製品 (「ソフトウェア・オファリング」) では、製品の使用に関する情報の収集、エンド・ユーザーの使用感の向上、エンド・ユーザーとの対話またはその他の目的のために、Cookie はじめさまざまなテクノロジーを使用することがあります。多くの場合、ソフトウェア・オファリングにより個人情報が収集されることはありません。 IBM の「ソフトウェア・オファリング」の一部には、個人情報を収集できる機能を持つものがあります。 ご使用の「ソ

フトウェア・オファリング」が、これらの Cookie およびそれに類するテクノロジーを通じてお客様による個人情報の収集を可能にする場合、以下の具体的事項をご確認ください。

この「ソフトウェア・オファリング」は、Cookie もしくはその他のテクノロジーを使用して個人情報を収集することはありません。

この「ソフトウェア・オファリング」が Cookie およびさまざまなテクノロジーを使用してエンド・ユーザーから個人を特定できる情報を収集する機能を提供する場合、お客様は、このような情報を収集するにあたって適用される法律、ガイドライン等を遵守する必要があります。これには、エンドユーザーへの通知や同意の要求も含まれますがそれらには限られません。

このような目的での Cookie などの各種テクノロジーの使用について詳しくは、『IBM オンラインでのプライバシー・ステートメントのハイライト』(http://www.ibm.com/privacy/jp/ja/)、『IBM オンラインでのプライバシー・ステートメント』(http://www.ibm.com/privacy/details/jp/ja/) の『クッキー、ウェブ・ビーコン、その他のテクノロジー』というタイトルのセクション、および『IBM Software Products and Software-as-a-Service Privacy Statement』(http://www.ibm.com/software/info/product-privacy) を参照してください。

商標

IBM、 IBM ロゴ、および ibm.com は、世界の多くの国で登録された International Business Machines Corporation の商標です。 世界中の多くの国で登録されています。 他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。 現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

登録商標 Linux® は、世界中で商標の所有者である Linux Torvalds の独占的ライセンシーである Linux Foundation のサブライセンスに従って使用されています。

Java およびすべての Java 関連の商標およびロゴは、 Oracle およびその関連会社の米国およびその他の国における商標または登録商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

［ア行］

［カ行］

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

[数字]

A

B

C

D

E

F

I

L

M

N

O

P

R

S

T

X

[特殊文字]