Intel® Itanium™ アセンブラ ユーザ・ガイド

2000 年 10 月

資料番号 : 712173J-003

バージョン	改訂履歴	日付
-003	商標を変更	2000 年 10 月

本書は、市場性、他者の権利を侵害しないこと、特定目的への適合、特定の提案、仕様、サンプルから生じる保証を含むがこれに限定されないいかなる保証もなく「無保証で」提供されます。

【輸出規制に関する告知と注意事項】

本資料に掲載されている製品のうち、外国為替および外国為替管理法に定める戦略物資等または役務に該当するものについては、輸出または再輸出する場合、同法に基づく日本政府の輸出許可が必要です。また、米国産品である当社製品は日本からの輸出または再輸出に際し、原則として米国政府の事前許可が必要です。

【資料内容に関する注意事項】

・ 本ドキュメントの内容を予告なしに変更することがあります。

・インテルでは、この資料に掲載された内容について、市販製品に使用した場合の保証あるいは特別な目的に合うことの保証等は、いかなる場合についてもいたしかねます。また、このドキュメント内の誤りについても責任を負いかねる場合があります。

・インテルでは、インテル製品の内部回路以外の使用にて責任を負いません。また、外部回路の特許についても関知いたしません。

・本書の情報はインテル製品を使用できるようにする目的でのみ記載されています。

インテルは、製品について「取引条件」で提示されている場合を除き、インテル製品の販売や使用に関して、いかなる特許または著作権の侵害をも含み、あらゆる責任を負わないものとします。

・いかなる形および方法によっても、インテルの文書による許可なく、この資料の一部またはすべてを複写することは禁じられています。

本資料の内容についてのお問い合わせは、下記までご連絡下さい。

インテル株式会社　資料センタ

〒 305-8603　筑波学園郵便局　私書箱 115 号

Fax: 0120-478832

Intel®Itanium™ **アセンブラに関連するインテル・プロセッサは、エラッタと呼ばれる設計上の欠陥または誤差のために、製品の動作が公表された仕様を外れることがあります。現在確認済みのエラッタについては、インテルまで問い合わせ下さい。**

注文番号を記載した文書あるいは本書で参照したその他のインテル文書は 0120-478832 **に** FAX **でお問い合わせになるか** http://www.intel.co.jp **にアクセスされると入手することができます。**

Intel、Intel **ロゴ、**Itanium、MMX、Pentium **は、アメリカ合衆国およびその他の国における** Intel Corporation **またはその子会社の商標または登録商標です。**

* **一般にブランド名または商品名は各社の商標または登録商標です。**

intel®

目次

概要 1

本書では、Windows NT* または Linux* システム上で Intel® Itanium™ アセンブラ (IAS) を使用する方法について説明する。

本書を十分に活用するためには、Itanium アーキテクチャとアセンブリ言語についてよく理解している必要がある。本書では、Intel Itanium アセンブリ・ツールに固有の機能について解説する。参考資料については、「関連資料」を参照のこと。

本書には、IAS 上でアセンブルされる Itanium アーキテクチャ・アセンブリ言語プログラムを作成する際に必要な情報が記載されている。本書では、IAS の使用方法と機能について説明する。また、本書では、IAS のすべての診断メッセージについても詳しく説明する。

IAS は、異種プラットフォーム・アセンブラである。このアセンブラは、32 ビット・システムおよび Itanium ベースのシステム上で実行され、Itanium アーキテクチャ・オブジェクト・ファイルを生成する。IAS は、IA-32 アセンブリ言語プログラムをアセンブルすることはできない。

1.1 本書について

本書の構成は次のとおりである。

本章「概要」では、関連資料を示し、本書の表記上の規則について説明する。

第 2 章「はじめに」では、IAS の概要と、アプリケーション開発プロセス内での IAS の位置について説明する。また、IAS のコマンドライン構文について説明する。

第 3 章「コマンドライン・オプション」では、IAS のコマンドライン・オプションについて説明する。

第 4 章「依存関係違反とアセンブリ・モード」では、IAS の自動モードと明示モードについて説明する。

第 5 章「機能」では、アセンブリ言語の中で定義された機能を補助する IAS の機能について説明する。

付録 A「診断メッセージ」では、IAS のエラー・メッセージと警告メッセージについて説明する。

付録 B「戻り値」では、IAS の終了時の戻り値について説明する。

付録 C「仕様」では、IAS の仕様について説明する。

付録 D「プレディケート解析」では、IAS でのプレディケート解析の実行方法について説明する。

1.2 システム環境

ハードウェアの必要条件：推奨ハードウェアは、Intel®Pentium® II プロセッサ以上 (メモリ 256MB 以上)。入力ファイルが非常に大きい (アセンブリ・コードが 100 万行を超える) 場合は、1GB のスワップ領域を用意することを推奨する。

ソフトウェアの必要条件：IAS は、Windows NT 4.0 または Linux 上で動作する。

1.3 関連資料およびツール

以下の資料は、補足的な情報を提供する。以下の資料のうちいくつかは、http://developer.intel.com で入手できる。

- 『*DVLoc for Scheduling Library*』(資料番号 748283)
- 『*Intel® Itanium™ アーキテクチャ・アセンブリ言語リファレンス・ガイド (Intel® Itanium™ Architecture Assembly Language Reference Guide)*』
  (資料番号 248801J-003)
- 『*Intel® Itanium™ プロセッサ・プログラマーズ・ガイド・アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル（Intel® Itanium™ Processor Progurammer's Guide Architecture Software Developer's Manual）*』
  『*第 1 巻 アプリケーション・アーキテクチャ*』
  (資料番号 245317J-002)
  『*第 2 巻 システム・アーキテクチャ*』
  (資料番号 245318J-002)
  『*第 3 巻 命令セット・リファレンス*』(資料番号 245319J-002)
  『*第 4 巻 Itanium™ プロセッサ・プログラマーズ・ガイド*』
  (資料番号 245320J-002)
- 『*ソフトウェア規則およびランタイム・アーキテクチャ・ガイド（Software Conventions and Runtime Architecture Guide）*』(資料番号 245358J-002)

以下の資料は、Microsoft 社で提供している。

- 『*Microsoft* Developer Studio, Visual C++* User's Guide, LINK Reference, Version 4.2*』

- 『*Microsoft* Portable Executable and Common Object File Format Specification, Version 4.1*』

1.4 表記上の規則

本書では、以下の表記上の規則を使用する。

`This type style`	構文の要素、予約済みのワード、キーワード、ファイル名、コンピュータの出力、またはプログラム例の一部を示す。大文字に意味がある場合を除いて、テキストは小文字で示す。
`This type style`	ユーザが入力するテキストを示す。
`This type style`	識別子、式、文字列、シンボル、または値のプレースホルダ。プレースホルダは、これらのアイテムのうち 1 つで置き換えられる。
`This type style`	診断メッセージ内の識別子のプレースホルダ。
`[item]`	オプションの要素を示す。
`[item | item]`	可能な選択肢を示す。縦線 (|) は、アイテムの区切りを示す。大括弧で囲まれたアイテムの中から 1 つを選択する。
This type style	デフォルトまたは使用例を示す。

はじめに 2

Intel® Itanium™ アセンブラ (IAS) は、Intel Itanium アセンブリ言語用アセンブラである。IAS は、アーキテクチャを最大に活用する。IAS は、IA-64 SDK for Windows NT Systems および Enabling SDK for UNIX の一部である。

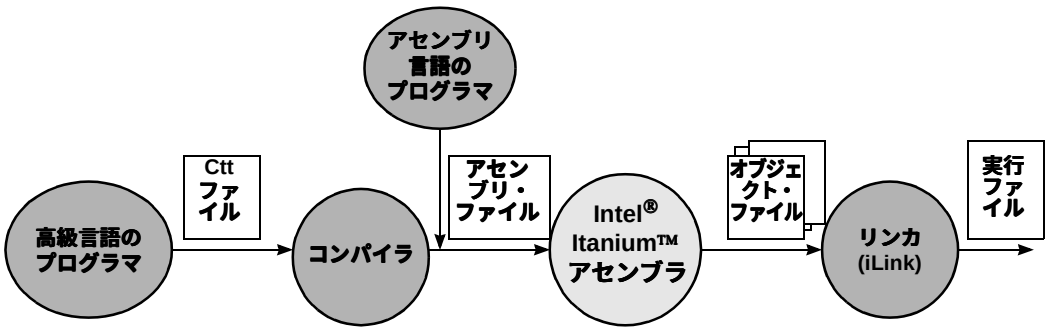
Windows NT ホスト上で IAS を実行し、UNIX 互換オブジェクト・ファイルを作成することができる。

本章では、アプリケーション開発環境内での IAS の位置と、IAS の使用方法について説明する。

2.1 環境

図 2-1 は、アプリケーション開発環境内で IAS が占める位置を示している。IAS は、アセンブリ言語のプログラマまたはコンパイラによって作成された Itanium アーキテクチャ・アセンブリ言語ファイルのアセンブルを実行する。IAS は、オブジェクト・ファイルと (多くの場合は) 診断リストを生成する。診断リストには、アセンブル処理中に IAS が生成したすべてのエラー・メッセージと警告メッセージが含まれる。

図 2-1. アプリケーション開発



C 言語とアセンブリ言語コードを 1 つの実行ファイルにすることについては、『*ソフトウェア規則およびランタイム・アーキテクチャ・ガイド*』を参照のこと。

2.2 IAS の起動

IAS を起動するには、次のコマンド・ラインを使用する。

ias *[options] filename [options]*

options	次の章で説明するコマンドライン・オプション。ファイル名の前後に、任意のオプションを指定することができる。
filename	アセンブリ言語入力ファイルを指定する。

コマンドライン・オプション 3

本章では、IAS のコマンドライン・オプションについて説明する。コマンドライン・オプションは、次のようなカテゴリに分類される。

- 情報
- ファイル処理
- コンパイル・モデル
- エラー処理
- UNIX ABI
- 詳細設定

注： わかりやすいように、最初の文字とそれに続く文字の間にスペースが入っているが、実際にはスペースを入力する必要はない。

3.1 情報

情報コマンドライン・オプションは、画面上に表示されるデータと診断ファイルに書き込まれるデータを制御する。

`[-H | -h]`

IAS は、すべてのコマンドライン・オプションの簡単な説明を表示し、終了する。これ以外のすべてのコマンドライン・オプションは無視される。
デフォルト：オプションの説明は表示されない。
例：`ias -h`

`-N so`

IAS は、IAS に関する情報を示す起動メッセージを、生成される診断ファイルに追加しないか、または画面上に表示しない。
デフォルト：起動メッセージは、診断ファイルに追加されるか、または画面上に表示される。
例：`ias -N so my_file.s`

-Q y

IAS は、IAS に関する情報を含む起動メッセージを、オブジェクト・ファイルの .comment セクションに追加する。
デフォルト：ELF フォーマットでは、起動メッセージはオブジェクト・ファイルに書き込まれる。COFF フォーマットでは、起動メッセージは書き込まれない。
例：`ias -Q y my_file.s`

-S nops

IAS は、次の数値を表示する。
- アセンブリの実行中に、IAS がコード内に挿入した nop の数。
- アセンブリの開始前の命令の数。
- 命令の合計数に対する nop の割合 (%)。

デフォルト：数値は表示されない。
例：`ias my_file.s -S nops`

-v

IAS は、IAS のバージョン情報を出力する。すべてのライブラリがリストされる。
デフォルト：バージョン情報は出力されない。
例：`ias my_file.s -v`

-V

起動メッセージを出力する。この設定はデフォルトである。このオプションは、下方互換性のために残されている。

3.2 ファイル処理

ファイル処理コマンドライン・オプションは、入力ファイルと出力ファイルを定義する。

-F *OMF*

このオプションは、オブジェクト・ファイルのオブジェクト・モジュール・フォーマット (OMF) を定義する。*OMF* の値は、Windows NT の場合は COFF32、目的のオペレーティング・システムが UNIX の場合は ELF32 または ELF64 である。
Windows NT のデフォルト：COFF32
UNIX のデフォルト：ELF64
例：`ias -F COFF32 my_file.s`

-I *pathname*

IAS は、組み込まれる入力ファイルの検索パス・リストに *pathname* を追加する。このオプションを繰り返し指定すれば、検索リストにさらにパスを

追加できる。パスは指定した順番に検索される。
デフォルト：現在のディレクトリ内のファイルだけを検索する。
例：`ias -I c:¥temp¥my_path my_file.s`

`-o` *`fname`*

IAS は、*`fname`* という名前のオブジェクト・ファイルを作成する。
デフォルト：入力ファイル名に `.obj` 拡張子を追加した名前。
例：`ias -o my_file.o my_file.s`
デフォルトでは、IAS は `my_file.obj` を作成する。

3.3 コンパイル・モデル

コンパイル・モデル・オプションは、デフォルトのコンパイル値を変更する。

`-M` *`ilp_model`*

このオプションは、IAS が使用するアドレス・モデルを定義する。*`ilp_model`* の値は、次のいずれかである。
`ilp64 | lp64 | p64` - デフォルト。アドレス・サイズを 64 ビットに設定する。整数のサイズと long 型のサイズは影響を与えない。
`ilp32` - アドレス・サイズを 32 ビットに設定する。COFF32 ファイル・フォーマットに関連する。
Windows NT のデフォルト：`ilp64`
例：`ias -o my_file.o my_file.s`

`-M` *`byte_order`*

このオプションは、データ割り当てステートメントにおけるバイト・オーダのデフォルトを設定する。*`byte_order`* の値は、`le`(リトル・エンディアン) または be(ビッグ・エンディアン) である。`.lsb`ディレクティブまたは`.msb`ディレクティブを使用して、特定のセクションのバイト・オーダをそれぞれリトル・エンディアンまたはビッグ・エンディアンに設定することができる。
デフォルト：`-M le`
例：`ias -M be my_file.s`

`-N pi`

IAS は特権命令を拒否する。このオプションを使用して、コードに特権命令が含まれていないことを確認できる。

デフォルト：特権命令は受け入れられる。

例：`ias -N pi my_file.s`

`-N close_fcalls`

IAS は、グローバル関数呼び出しを解決しない。代わりに、他の箇所で定義された同じ名前によって、他のプロシージャを使用できる。
デフォルト：関数呼び出しは解決されない。
例：`ias -N close_fcalls my_file.s`

`-p 32`

IAS は、32 ビット要素を再配置可能なデータ要素として定義できるようにする。このオプションは、下位互換性のために残されている。

3.4 エラー処理

エラー処理オプションは、IAS の診断メッセージの処理方法を定義する。

`-e` *`fname`*

IASは、*`fname`*という名前の診断ファイルを作成する。エラー・メッセージと警告メッセージはこのファイルに送信される。
デフォルト：エラーは画面上に表示される(stderr)。
例：`ias -e my_err.txt my_file.s`

`-E` *`max_num`*

IAS が検出したエラーの数が *`max_num`* に達すると、IAS は終了する。
デフォルト：`-E 30`
例：`ias -E 3 my_file.s`

`-W` *`warning_level`*

IAS は、さまざまなレベルの警告を表示する。*`warning_level`* の値は、次のいずれかである。

`0`	警告を表示しない。
`1`	致命的な警告を表示する。
`2`	警告を表示する。
`3`	致命度が中位の警告を表示する。
`4`	すべての警告を表示する。
`x`	すべての警告をエラーとして処理する。

エラーが検出された場合は、オブジェクト・ファイルを作成しない。
デフォルト：`3`
例：`ias -W 1 my_file.s`

3.5 UNIX ABI セクション

ここでは、UNIX ABI に固有のコマンドライン・オプションについて説明する。これらのオプションを使用して、浮動小数点レジスタの範囲を制限したり、カーネル・モードの呼び出し規則を定義することができる。これらのオプションは、`-F ELF64` オプションと組み合わせて使用しなければならない。

`-M rfp`

IASは、浮動小数点レジスタの範囲をF6～F11に制限する。これによって、カーネル・モードへの移行およびカーネル・モード終了時のレジスタの保存と復元の回数が減るため、システムの処理時間を短縮できる。他の浮動小数点レジスタを使用しようとすると、エラーが発生する。
デフォルト：すべての浮動小数点レジスタを使用できる。
例：`ias -F ELF64 -M rfp my_file.s`

`-M const_gp`

IAS は、オブジェクト・ファイル内でシングル・グローバル・ポインタ (GP) モデルを設定する。カーネルは、1 つの GP を持つシングル・モデルと見なされる。
デフォルト：オブジェクト・ファイル内で追加フラグを設定しない。
例:`ias -F ELF64 -M const_gp my_file.s`

`-M no_plabel`

IAS は、オブジェクト・ファイル内のモデルをシングル GP に設定し、関数ディスクリプタ (plabel) を設定しない。`-M const_gp` オプションと同じように、カーネルはシングル GP と見なされ、plabel を使用しない。
デフォルト：オブジェクト・ファイル内で追加フラグを設定しない。
例:`ias -F ELF64 -M no_plabel my_file.s`

3.6 詳細設定セクション

ここでは、いくつかの詳細設定オプションについて説明する。これらのオプションは、アセンブリ・モードを変更したり、仮想レジスタ割り当てを可能にする。

`-X explicit`

IAS は、デフォルトの初期アセンブリ・モードを自動モードから明示モードに変更する。
デフォルト：IAS は自動モードでアセンブルを実行する。

例：`ias -X explicit my_file.s`
依存関係違反についての詳細は、第 4 章『依存関係違反とアセンブリ・モード』を参照のこと。

`-X vral`

IAS は、レジスタ割り当てエンジン (仮想レジスタ割り当て：**vral**) を起動する。これによって、実際のレジスタ名の代わりにシンボリック名を使用できる。IAS は、すべてのレジスタ割り当ての結果を記録し、サフィックス `.vra` のファイルを作成する。
デフォルト：vral はアクティブでないため、Vral 構文は認識されない。
例：`ias -X vral my_file.s`

`-X unwind`

IAS は、アンワインド生成ユーティリティを起動する。IAS は、ファイル内のすべてのプロシージャについてアンワインド情報を構築し、アンワインド・ディレクティブをすべて無視する。
デフォルト：アンワインド情報は生成されない。
例：`ias -X unwind my_file.s`

`-d debug`

COFF32 オブジェクトの場合、IAS は、Code View デバッグ情報および行情報を生成する。これにより、シンボリック・デバッガを使用して、コード行ごとにデバッグを行い、シンボルを確認することができる。
デフォルト：デバッグ情報および行情報は生成されない。
例：`ias -F COFF32 -d debug my_file.s`

`-a indirect=`*`br_target`*

このコマンドライン・オプションは、注釈なしの間接分岐のデフォルト分岐ターゲットを IAS に指示する。このオプションは、仮想レジスタ割り当てに必要である。***`br_target`*** の値は次のいずれかである。
`exit` 出口が分岐ターゲットと見なされる。
`labels` 任意のラベルが分岐ターゲットと見なされる。
デフォルト：出口が分岐ターゲットと見なされる。
例：`-ias -X explicit -a indirect=labels my_file.s`
または`-ias -a indirect=exit my_file.s`

-N us

このオプションは、符号付き数値と符号なし数値を統合して、数値の範囲を拡張する。IAS は、7 ビット長として、-64 〜 +127 の範囲の数値を受け入れる。
デフォルト：7 ビット数の範囲は、-64 〜 +63 または 0 〜 +127 である。
例：`ias -N us my_file.s`

依存関係違反とアセンブリ・モード　4

本章では、依存関係違反と、Intel® Itanium™ アセンブラがコード内の依存関係違反を解決する方法について説明する。

データ依存関係の違反は、同じ命令グループ内の 2 つの命令が、同じ Itanium アーキテクチャ・リソース (暗黙的オペランドとして現れるリソースを含む) にアクセスすることによって発生する。依存関係違反は、アーキテクチャ上で未定義の動作を発生させる。アセンブラは、アセンブリ・モードに従って、命令グループ内に発生する依存関係違反を検出し、排除することができる。

明示モードでコードを作成する場合は、ユーザがバンドルとストップ (;;) を配置できる。自動モードでは、IAS が自動的にコードをバンドルしてストップを追加し、依存関係違反を解決する。1 つのファイル内で、2 つのモードを併用することもできる。バンドルとストップについては、『*Intel® Itanium™ アーキテクチャ・アセンブリ言語リファレンス・ガイド*』または本書の第 5 章「機能」を参照のこと。

明示モードでコードを作成すると、IAS は検出された依存関係違反を報告する。依存関係を解決する最も簡単な方法は、ストップを挿入することである。ただし、不正確なレポートが作成される可能性がある。この場合は、ユーザの判断で、ある範囲の注釈とコマンドを使用できる。これらについては、本章後半で説明する。

データの依存関係についての詳細は、『*Intel® Itanium™ アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル*』および『*DVLoc for Scheduling Library*』を参照のこと。

4.1　アセンブリ・モード

IAS は、明示モードまたは自動モードで、アセンブリ・コードの読み取りと処理を実行する。経験豊富なユーザが Itanium アーキテクチャについて十分に理解しており、コードに高性能が要求される場合は、明示モードを使用できる。初心者の方や、特に高性能が要求されない場合には、自動モードの使用を推奨する。

4.1.1　自動モード

自動モードは、特に高性能が要求されないコードを作成する場合に最適である。

このモードでは、リニア・コードを作成する際に、バンドル境界を指定したり、アーキテクチャ上の依存関係に注意する必要はない。IAS がコードをバンドルし、必要に応じてストップ (;;) を挿入する。IAS は、ユーザが追加した、依存関係違反に関連するすべての注釈とストップを無視する。

デフォルトの初期モードは、自動モードである。コマンドライン・オプション `-X explicit` を使用して、初期モードを明示モードに変更することができる。

IAS は、モード・ディレクティブ `.auto` の後に中括弧を検出すると、エラーを生成する。

Itanium アーキテクチャは、挿入するストップの数をできるだけ少なく抑えようとする。

注： 自動モードでは、アセンブラは `.pred.rel` 注釈を無視する。

4.1.2 明示モード

明示モードは、高性能が要求されるコードを作成する場合に最適である。

このモードでは、ユーザがコードにストップと注釈を挿入することによって、依存関係違反を回避しなければならない。IAS は、このコードに依存関係違反がないかどうか検査し、何らかの問題が検出された場合はエラーを戻す。

IAS を明示モードに設定するには、次のいずれかの方法を使用する。

- デフォルトの自動モードになっているとき、バンドル境界を意味する中括弧 ({、}) を挿入する (ただし、`.auto` ディレクティブの後に中括弧を挿入するとエラーになる)。
- `.explicit` ディレクティブを挿入する。
- コマンドライン・オプション `-X explicit` を使用する。このオプションは、デフォルト・モードを自動モードから明示モードに変更する。

新しいコード・セクションに移行すると、IAS はアセンブリ・モードをデフォルトの自動モードに戻す。

バンドル境界を指定せずに明示コードを作成すると、IAS がバンドル境界を追加する。ただし、ストップと注釈を追加する役割はユーザが受け持つ。注釈は、プレディケート・レジスタと他のランタイム値の間の関係を定義する。「誤ったレポートの防止」を参照のこと。

4.1.3 IAS の動作

両方のモードで作成されたコードを、1 つのファイル内に混在させることができる。IAS のモードを切り替えるには、次のいずれかの方法を使用する。

- コマンドライン・オプション `-X explicit` を使用する。
- モード・ディレクティブ `.auto`、`.explicit`、および `.default` を使用する。
- デフォルトの初期モードが自動モードになっている場合、IAS は、コード構文に従って自動的にモードを切り替える。

バンドルがない場合は、IAS がコードをバンドルし、バンドルの整合性を保つために nop を追加し、依存関係違反を回避するためにストップを追加する。

ディレクティブ `.explicit` および `.auto` は、現在のコード・セクションについて、デフォルトの初期モードを無効にする。

ディレクティブ `.default` は、IAS をデフォルトの初期モードに戻す。

IAS は、明示的なバンドル内でモード・ディレクティブを検出した場合は、エラーを生成する。

IAS は、モードを切り替えるとき、自動的にストップを挿入する。

Itanium アーキテクチャ・コードの作成時に依存関係違反を回避する方法については、「誤ったレポートの防止」を参照のこと。

例：明示モード

IAS は、明示モードで以下のコードを検出すると、依存関係違反エラーを記録する。

ディレクティブ `.default` は、アセンブラの動作モードを、コマンド・ラインで定義されたデフォルトの初期モードに変更する。この例では、デフォルトの初期モードは自動モードである。

```
.explicit
(p1)   mov r1 = r4
       ;;
(p2)   mov r6 = r2
```

```
          ldfps f4,f5 = [r4]
          fabs f4 = f7     // WAW error on f4
// IAS inserts a stop when the mode switches
   .default
          add r5 = 0, r7
```

例：自動モード

自動モードで、前の例と同じようなコードを使用した場合は、次の例のように、IAS は既存のストップを無視し、依存する命令の間にストップを挿入する。

```
   .auto
   (p1)   mov r1 = r4
          ;;
// IAS ignores this stop
   (p2)   mov r6 = r2
          ldfps f4,f5 = [r4]
// IAS inserts a stop to avoid WAW error on f4
          fabs f4 = f7
```

例：デフォルトの初期モードが自動モードの場合

次の例では、デフォルト・モードは自動モードである。

```
   (p1)   mov r1 = r4
// IAS inserts a stop here
   (p2)   mov r1 = r2
{                  // IAS inserts a stop here
// IAS treats this code as explicit
          ldfps  f4,f5 = [r4]
          fabs   f4 = f7 // write-after-write error
}
```

4.2 シリアル化命令とメモリ同期命令

シリアル化 (srlz) 命令とメモリ同期 (sync) 命令には、命令グループに関して、以下の制約条件が適用される。

- シリアル化命令 (srlz.i または srlz.d) は、シリアル化される処理の後に続く命令グループ内に置かれていなければならない。
- シリアル化に依存する処理は、srlz.i より後の命令グループ内に置かれていなければならない。

- シリアル化に依存する処理は、srlz.d より後に置かれていなければならないが、srlz.d と同じ命令グループ内であってもかまわない。
- sync.i 命令と先行するキャッシュ・フラッシュ処理は、別々の命令グループ内に置かれていなければならない。

安全上の理由で、IAS は、自動モードでは、srlz.d 命令の前、sync.i 命令の前、および srlz.i 命令の前後にストップを挿入する。明示モードでは、IAS は、ストップが欠けていてもエラーを生成しない。

4.3 誤ったレポートの防止

IAS は、明示モードで、誤って依存関係違反をレポートすることがある。情報が不足している場合、IAS はコードのすべての関係を調査することができない。

誤ったレジスタ依存エラーを回避する最も簡単な方法は、ストップを使用することである。依存関係違反の原因となる 2 つの命令の間に、ストップ (;;) を挿入する。この方法は簡単で常に有効であるが、このためにパフォーマンスが低下することがある。

IAS が依存関係違反を分析して誤ったレポートを解決できるように、以下の注釈を使用すれば、パフォーマンスの低下は起こらない。

- .pred.rel
- .reg.val
- .mem.offset

注： 注釈は、IAS が見かけ上の依存関係違反を分析するときに使用する追加情報を提供する。

注釈の構文については、『*Intel® Itanium™ アーキテクチャ・アセンブリ言語リファレンス・ガイド*』を参照のこと。

注釈の追加によって誤ったレポートを防止する方法の例を以下に示す。

4.4 プレディケート関係解析

IAS は、プレディケート関係を分析して、2 つのプレディケート命令間に依存関係違反があるかどうか判断する。次の例は、「書き込みの後の書き込み」（write-after-write）の依存関係違反を示している。

```
(p1) add r5 = 8, r6
(p2) add r5 = r7, r0
```

IAS でのプレディケート解析の実行方法については、付録 D「プレディケート解析」を参照のこと。

比較命令は、プレディケート・レジスタの値を定義するため、プレディケート関係が定義されることがある。

プレディケート関係に関する情報を受け渡すには、プレディケート関係注釈 `.pred.rel` を使用する。

注釈 `.pred.rel` は、次のいずれかの形式になる。

`"mutex"`	mutex 形式は、相互に排他的な (mutually exclusive) 関係を定義する。
`"imply"`	imply 形式は、含意関係を定義する。
`"clear"`	clear 形式は、次に説明するように、mutex 関係と imply 関係を削除する。

競合する命令の前にエントリー・ポイントがある場合は、IAS は、エントリー・ポイントより前に定義されたすべての既存のプレディケート関係を無視する。

入口点は、次のいずれかである。

- ラベル (ローカル、グローバル、または一時的)
- `br.call` 命令の後に続くバンドルのアドレス
- 直接分岐のターゲット

プレディケート関係注釈を使用して、プレディケート間の関係を定義し、依存関係違反エラーを防止することができる。

4.4.1 比較命令

比較命令 (`cmp`、`tbit`、`fclass`、および `fcmp`) は、プレディケートの値を定義する。これらの命令は、プレディケート命令の前に置かれる。比較命令は、指定されたプレディケート・レジスタが相互に排他的であることをアセンブラに指示する。比較命令は、デスティネーション・プレディケート・レジスタとそれ以外のプレディケート・レジスタの間に定義された、他のすべての mutex 関係を無効にする。

わかりやすいように、以下の例はすべて `cmp` 命令の使用例を示す。`tbit`、`fclass`、および `fcmp` 命令を使用しても、同じ結果が得られる。

以下の例では、`cmp` 命令は、`p1` と `p2` は同時に真であることはできないと指示することによって、プレディケート関係違反エラーを回避している。

```
cmp.lt p1, p2 = r13, r0;;
```

```
(p1)   add r5 = 8, r6
(p2)   add r5 = r7, r0
```

cmp 命令は、デスティネーション・プレディケート・レジスタとそれ以外のすべてのプレディケート・レジスタの間の mutex 関係を無効にする。例えば、次のようになる。

```
          cmp.lt p1,p2 = r13,r0;;
// p1 and p2 are mutually exclusive
          cmp.lt p1,p3 = r12,r11;;
// the mutex relation between p1 and
// p2 is destroyed by cmp
       (p1)   add r5 = 8, r6
       (p2)   add r5 = r7, r0  // WAW error
```

4.4.2 mutex 形式の .pred.rel 注釈

フォーマット： .pred.rel “mutex” *p1, p2* [,…]

p1, p2... はプレディケート・レジスタである。

.pred.rel “mutex” は、指定されたプレディケート・レジスタのうち 1 つだけが真であるか、またはすべてが偽であることをアセンブラに指示する。例えば、次のようになる。

```
       .pred.rel “mutex”, p1, p2, p3
// p1, p2, and p3 are mutually exclusive or all zero
       (p1) add r5 = 8, r6
       (p2) add r5 = r7, r0
       (p3) add r5 = r14, r0
```

mutex 形式には順序付けがない。つまり、プレディケートを指定する順序は重要ではない。

mutex 形式は、デスティネーション・プレディケート・レジスタとそれ以外のプレディケート・レジスタの間のあらかじめ定義された mutex 関係を無効にしない。例えば、次のようになる。

```
       .pred.rel “mutex”,p1,p2
       .pred.rel “mutex”,p1,p3
       .pred.rel “mutex”,p2,p3
// p1,p2 and p3 are mutex
       (p1)   mov r4=r5
       (p2)   mov r4=r6
       (p3)   mov r4=r7  // no WAW error is reported
```

4.4.3 imply 形式の .pred.rel 注釈

フォーマット：.pred.rel “imply” *p1, p2*

p1, p2 はプレディケート・レジスタである。

"imply" 形式の .pred.rel 注釈は、最初のプレディケートが真である場合は 2 番目のプレディケートも真であることをアセンブラに指示する。最初のプレディケートが偽である場合については未定義である。この場合は、2 番目のプレディケートの値は確定されない。imply 形式には順序付けがある。つまり、プレディケートの順序が重要である。

次の例では、p1 が真であれば、p2 も真になる。

```
       .pred.rel “imply”, p1, p2
       (p1)   mov r4=r5
       (p2)   br.cond.dpnt.few b0
              mov r4=r5
// WAW on r4 is not reported as p1 implies p2
```

imply 形式は、推移的な関係である。p1 が p2 を含意し、p2 が p3 を含意する場合は、p1 は p3 も含意する。

4.4.4 clear 形式の .pred.rel 注釈

"clear" 形式の .pred.rel 注釈は、プレディケート関係を削除する。プレディケート・レジスタ p1 を指定すると、IAS は、p1 を含むすべての mutex 関係と、p1 がプレディケート・レジスタになるすべての関係を削除する。プレディケート・レジスタを指定しないと、IAS は、この命令をショートカットとして解釈し、すべてのプレディケート・レジスタを指定する。

フォーマット：.pred.rel “clear” [*p1*[,*p2*[,…]]]

p1, p2 はプレディケート・レジスタである。

例えば、次のようになる。

```
.pred.rel “clear” p1  // clears all the p1 relations
.pred.rel “clear”  // clears all predicate relations
```

次の例は、mutex 関係と imply 関係の両方を使用している。"clear" 形式の注釈は、mutex 関係か imply 関係かによって異なる結果を生じる。

```
   .pred.rel “mutex”,p1,p2
   .pred.rel “mutex”,p3,p1
   .pred.rel “imply”,p1,p4
   .pred.rel “imply”,p5,p1
   .pred.rel “clear”,p1
```

```
// clears the two mutex relations
// and the first implication form
   (p1)   mov r1=r2 // => WAW on r1 is reported
   (p2)   mov r1=r2
          ;;
   (p1)   mov r2=r3
   (p3)   mov r2=r3 // => WAW on r2 is reported
          ;;
   (p1)   mov r3=r4
   (p4)   br.cond.sptk.few b0
          mov r3=r4 // => WAW on r3 is reported
// WAW would not have been reported if p1 -> p4
          ;;
   (p5)   mov r4=r5
   (p1)   br.cond.sptk.few b0
          mov r4=r5 // WAW is not reported.
                    // p5 -> p1 is still valid
```

4.4.5 単純な比較では作成されない mutex 関係

次のコードでは、r10 は一度に 1 つの値しか持てないため、p1、p2、および p3 は相互に排他的である。IAS は、この本質的な mutex 関係を解釈できないため、3 つの WAW(write-after-write) 依存関係違反を報告する。

```
          cmp.eq p1=1,r10
          cmp.eq p2=2,r10
          cmp.eq p3=3,r10;;
(p1)      mov r4=r1
(p2)      mov r4=r2
(p3)      mov r4=r3
```

この問題を解決するには、次のように "mutex" 形式の .pred.rel 注釈を使用する。

```
          cmp.eq p1=1,r10
          cmp.eq p2=2,r10
          cmp.eq p3=3,r10;;
   .pred.rel “mutex”,p1,p2,p3
(p1)      mov r4=r1
(p2)      mov r4=r2
(p3)      mov r4=r3
```

4.4.6 プレディケーション付き分岐によって分離された命令

次の例では、無条件の比較による依存関係違反は発生しない。わかりやすいように、命令には #1 ～ #6 の番号が付いている。

命令 #2 と #5 は並行して実行されないため、r1 上の見かけ上の WAW(Write Aafter Write) は実際には発生しない。すなわち、命令 #2 の実行 (p2 が真) は命令 #4 の実行を含意する (p2 は p1 を含意する) ため、コードの実行は L に分岐し、命令 #5 には到達しない。

r2 上の見かけ上の WAW は、命令 #4 が実行されずに命令 #6 が実行される場合にのみ発生する。命令 #6 の実行 (p3 が真) は命令 #4 の実行を含意する (p3 は p1 を含意する) ため、r2 上の WAW は実際には発生しない。

```
#1     (p1)   cmp.eq.unc p2,p3=r1,r2;;
#2     (p2)   mov r1=r10
#3            mov r2=r11
#4     (p1)   br.cond.dpnt.few L
#5            mov r1=r12
#6     (p3)   mov r2=r13
```

r1 および r2 上の WAW エラーが誤って報告されることを防ぐには、次のように、"imply" 形式の .pred.rel 注釈を挿入する。

```
(p1)   cmp.eq.unc p2,p3=r1,r2;;
.pred.rel “imply”,p2,p1 // if p2 is true, p1 is true
.pred.rel “imply”,p3,p1 // same as above, with p3
(p2)   mov r1=r10
       mov r2=r11
(p1)   br.cond.dpnt.few L
       mov r1=r12
(p3)   mov r2=r13
```

4.4.7 safe_across_calls

.pred.safe_across_calls 注釈によって、他のプロシージャに対する呼び出しの後でも、プレディケート関係を維持することができる。この注釈を使用して、どのプレディケートの関係を維持するかを指定できる。この注釈の有効範囲は、現在のプロシージャまたはモジュールの中である。

1 つの文で、複数のプレディケートを個々に指定したり、プレディケートの範囲を指定することができる。

形式：.pred.safe_across_calls p1, p2, ...

p1、p2 などは、特定のプレディケート・レジスタまたはプレディケート・レジスタの範囲を表す。

次の例では、.pred.safe_across_calls 注釈が含まれていない場合、プロシージャ foo がプレディケートの値を変更する可能性があるため、IAS は最後の 2 つの命令の間の依存関係違反を報告する。

```
.pred.safe_across_calls p2-p6, p10, p11
```

```
.pred.rel “mutex“, p3, p4
      br.call b1=foo
(p3)  mov r5=r32
(p4)  add r5=8, r32
```

.pred.safe_across_calls 注釈によって定義されたプレディケート関係をクリアするには、次の形式で注釈を使用する。

```
.pred.safe_across_calls ”clear”
```

4.4.8 レジスタ・ファイルへの間接アクセス

依存関係違反の有無は、汎用レジスタの値によって決まることがある。例えば、レジスタ・ファイルに間接的にアクセスする場合がこれに当たる。次の例では、2 つの異なるレジスタに間接的にアクセスする。IAS は、インデックス・レジスタの値に関する情報を持っていないため、pmd 上の WAW エラーを報告する。

```
mov r1=2
mov r2=4;;
mov pmd[r1]=r11
mov pmd[r2]=r12
```

この問題を解決するには、次のように、.reg.val 注釈を使用して、pmd への 2 つの書き込みが異なるレジスタにアクセスすることを IAS に指示する。

```
     mov r1=2
     mov r2=4;;
.reg.val r1,2
     mov pmd[r1]=r11
.reg.val r2,4
     mov pmd[r2]=r12
```

4.4.9 同じ命令グループ内の st8.spill と ld8.fill

st8.spill 命令は、アクセスされるメモリのアドレスに従って、UNAT アプリケーション・レジスタの特定のビットに書き込む。ld8.fill 命令は、アクセスされるメモリのアドレスに従って、UNAT アプリケーション・レジスタの特定のビットを読み込む。詳細については、『*Intel® Itanium™ アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル*』を参照のこと。

IAS は、アクセスされるメモリのアドレスを認識できないため、注釈が使用されない場合は、以下の依存関係違反を報告する。

WAW(Write After Write)　　st8.spill 命令のすべてのペアについて。

RAW(Read After Write)	同じ命令グループ内の st8.spill 命令の後に現れる、すべての ld8.fill 命令について。

次のコードでは、アクセスされる UNAT ビットが異なることをコードが保証しているにもかかわらず、1 つの WAW 依存関係違反と 2 つの RAW 依存関係違反が報告される。

```
add r2=r1,8
add r3=r1,16;;
st8.spill [r1]=r11
st8.spill [r2]=r11
ld8.fill   r12=[r3]
```

この誤ったレポートを防止するには、それぞれの st8.spill および ld8.fill 命令の前に、.mem.offset 注釈を使用する。この注釈は、現在のスタックなどの何らかのローカル・メモリ領域を基準とするメモリ・アドレス位置を指定しなければならない。

```
LOCAL_STACK_INDEX=0
add r2=8,r1
add r3=16,r1;;
.mem.offset 0,LOCAL_STACK_INDEX
     st8.spill [r1]=r11
.mem.offset 8,LOCAL_STACK_INDEX
     st8.spill [r2]=r11
.mem.offset 16,LOCAL_STACK_INDEX
     ld8.fill   r12=[r3]
```

.mem.offset 注釈についての詳細は、『*Intel® Itanium™ アセンブリ言語リファレンス・ガイド*』を参照のこと。

IAS でのプレディケート解析の実行方法については、付録 D「プレディケート解析」を参照のこと。

機能 5

本章では、Intel® Itanium™ アセンブラ (IAS) の機能について説明する。

- アセンブリ言語機能の概要。これについては、『*Intel® Itanium™ アーキテクチャ・アセンブリ言語リファレンス・ガイド*』に詳しく定義されている。
- Intel Itanium アーキテクチャ・アセンブリ・ツールに固有の補助機能。

5.1 IA-64 アセンブリ言語機能

IAS は、以下の Itanium アーキテクチャ・アセンブリ言語仕様機能をサポートしている。

- 命令セット
- バンドルの作成
- 命令グループ
- データ割り当て
- アセンブリ言語ディレクティブ
- 64 ビット・アドレス空間
- アライメント
- 代入文
- 別名定義
- 算術式の処理

以下の各項では、これらの機能について簡単に説明する。各機能についての詳細は、『*Intel® Itanium™ アーキテクチャ・アセンブリ言語リファレンス・ガイド*』を参照のこと。

5.1.1 命令セット

IAS は、『*Intel® Itanium™ アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル*』に定義された、すべての Itanium アーキテクチャ命令セットをサポートしている。

5.1.2 バンドルの作成

Itanium プロセッサは、命令をバンドル単位で実行する。バンドルには、最大 3 つまでの命令とそれに関連するテンプレートが入っている。テンプレートは、どのタイプの実行ユニットがバンドル内の各命令を処理するかを定義する。

IAS は、次のいずれかのレベルでバンドルを定義できる。

- 明示的なバンドルの作成とテンプレートの定義。ユーザがバンドル境界とバンドル・テンプレートを定義する。
- テンプレートの定義なしの明示的なバンドルの作成。ユーザがバンドル境界を定義し、IAS が最適なバンドル・テンプレートを選択する。
- 暗黙的なバンドルの作成。IAS が、最適なコード・サイズの編成を選択することによって、バンドル境界とバンドル・テンプレートを選択する。

すべてのバンドル定義レベルで、IAS は必要な NOP を挿入する。

5.1.3 命令グループ

Itanium プロセッサは、複数の命令を並行して実行する。並行して実行できる命令は、命令グループとして編成される。命令グループは、相互に依存関係を持たない一連の連続する命令で構成される。命令グループは、ストップ (;;) によって終わる。IAS は、『*Intel® Itanium™ アーキテクチャ・アセンブリ言語リファレンス・ガイド*』に定義された、明示的なストップをサポートしている。

IAS は、命令グループ内にデータの依存関係がないかどうか検査する。データの依存関係の例には、同じレジスタに対して読み込み命令に続いて書き込み命令を実行する場合がある。依存関係違反についての詳細は、第 4 章「依存関係違反とアセンブリ・モード」を参照のこと。

5.1.4 データ割り当て

IAS は、メモリ内の空間の割り当てと初期化を実行できる。IAS は、以下のデータ・タイプをサポートしている。

- 整数　　1、2、4、または 8 バイト長
- 浮動小数点数　　4、8、10、または 16 バイト長
- 文字列　　最大 1024 ビット長

5.1.5 アセンブリ言語ディレクティブ

IAS は、ローカル・ラベル・ディレクティブを除く、すべての Itanium アーキテクチャ・アセンブリ言語ディレクティブをサポートしている。これらのディレクティブについては、『*Intel® Itanium™ アセンブリ言語リファレンス・ガイド*』を参照のこと。サポートされるディレクティブは、以下の操作または情報を提供する。

- セクションの制御
- シンボルの制御
- ファイルの組み込み
- バンドル・テンプレートの選択
- デバッグ情報
- アンワインド情報

5.1.6 64 ビット・アドレス空間

IAS は、64 ビット・アドレス空間をサポートしている。

`-ilp32` コマンドライン・オプションを使用する場合は (COFF32 出力ファイル・フォーマットではこれがデフォルト)、シンボリック・アドレスは 32 ビット割り当て (`data4`) のみに制限される。64 ビット割り当て (`data8`) で再配置可能な式を使用しようとすると、IAS はエラー・メッセージを表示する。

5.1.7 アライメント

デフォルトでは、IAS は、バンドルのアライメントを 16 バイト境界に合わせる。データ要素のアライメントは、その要素のサイズに従って合わせる。

IAS は、各セクション内の最大アライメント要求に従って、各セクションのアライメントを合わせる。バンドル、データ要素、または `.align` ディレクティブが、アライメント要求を作成する。

セクションのアライメントは、オブジェクト・ファイル・フォーマットによって制限される。`COFF32` オブジェクト・ファイル・フォーマットでは、セクションのアライメント境界は 8KB に限定される。実際の制限は、リンカのアライメント・ポリシーによって異なる。リンカについての詳細は、『*Microsoft* Developer Studio, Visual C++* User's Guide, and LINK Reference*』を参照のこと。

データ割り当て文中で自動アライメントを無効にするには、データ割り当て文に .ua コンプリータを追加する。例えば、次のようにする。

```
data8.ua 0x855
```

5.1.8 代入文

プログラマは、代入文を使用して、シンボルに値を割り当てることによってシンボルを定義することができる。この値には、他のシンボルへの参照、レジスタ名、または式を指定できる。詳細については、『*Intel® Itanium™ アーキテクチャ・アセンブリ言語リファレンス・ガイド*』を参照のこと。

5.1.9 別名定義

IAS は、シンボル名とセクション名の別名定義をサポートしている。別名定義は、次の方法で行われる。

シンボル名
: .alias ディレクティブによって別名が付けられる。別名は、出力ファイルのシンボル・テーブルに記録される。

セクション名
: .secalias ディレクティブによって別名が付けられる。別名は、出力ファイルのシンボル・テーブルに記録される。詳細については、「.secalias ディレクティブ」の項を参照のこと。

5.1.10 算術式の処理

IAS は、標準的な算術表記法を使用した、定数とアドレスに関する算術式の使用をサポートしている。算術式には、シンボル、数値定数、および演算子を使用できる。

IAS は、リンク再配置演算子の使用により、ランタイムにリンカ・テーブルにアクセスする式をサポートしている。リンク再配置演算子についての詳細は、『*Intel® Itanium™ アーキテクチャ・アセンブリ言語リファレンス・ガイド*』を参照のこと。

入力ファイルの定数は、符号付き 128 ビット数として内部で表現される。IAS は、すべての整数計算を 128 ビット精度で実行し、浮動小数点計算 (実数) を拡張倍精度 (long double) で実行する。

5.2 補助機能

IAS は、『*Intel® Itanium™ アーキテクチャ・アセンブリ言語リファレンス・ガイド*』では定義していない、次のような追加機能を持っている。

- IA-32 `jmpe` 命令
- `instenc` 疑似命令
- 文字列等式
- `.secalias` ディレクティブ
- デバッグ・ツール用の行情報
- #line のサポート
- あらかじめ定義されたシンボル
- 仮想レジスタ割り当て
- アンワインド情報生成

以下の各項では、これらの機能について説明する。

5.2.1 IA-32 `jmpe` 命令

IAS は、Itanium アーキテクチャ・アセンブリ言語ファイル内からの IA-32 から Itanium アーキテクチャへの移行命令 (`jmpe`) をサポートしている。`jmpe` 命令を使用して Itanium アーキテクチャ・ファイルをアセンブルすると、IAS は、IA-32 コードから Itanium アーキテクチャ・コードへの移行を可能にする IA-32 `jmpe` 命令を生成する。

以下のディレクティブを使用できる。

`jmpe.next`	16 バイトにアライメントの合った次のアドレスにジャンプする。
`jmpe.abs` *`address`*	数値または再配置可能な式として指定されたアドレスにジャンプする。
`jmpe.`*`IA-reg32`*	IA-32 レジスタ内で指定されたアドレスに間接的にジャンプする。例：`jmpe.eax`

5.2.2 `instenc` 疑似命令

instenc 疑似命令を使用して、バンドル内のスロットに 41 ビットの即値を入力できる。Itanium プロセッサは、この即値を命令として認識する。ただし、IAS は、入力された即値が有効な Itanium 命令に対応しているかどうか確認しない。

アセンブラの現在のバージョンでは、無効とされる命令を含む実行コードを作成する場合この疑似命令が効果的である。

5.2.2.1 構文

```
instenc.completer   imm41
```

completer バンドル内の命令の役割を定義する。この命令のコンプリータには、次のものがある。

a	ALU 命令
m	メモリ命令
i	整数命令
b	分岐命令
f	浮動小数点命令

imm41 Itanium 命令に対応する即値。

5.2.2.2 例

この例では、バンドルに浮動小数点命令を挿入する。

```
{
add r1 = r2, r3
instenc.f 0x1F423C02DA9
}
```

5.2.3 文字列等式

シンボルと値またはレジスタを等価にする等式文 (==) は、シンボルと文字列を等価にすることもできる。例えば、次のようになる。

```
save_file_name == @filename または
source_file == “my_file.s”
```

文字列等式文を順方向に参照することはできない。

5.2.4 .secalias ディレクティブ

.secalias ディレクティブは、セクション名の別名を定義する。.alias がシンボル名の別名を定義するのと同じ方法で、.secalias はセクション名の別名を定義する。詳細については、「別名定義」の項を参照のこと。

セクションは、入力ファイル内ではセクション名によって参照され、出力ファイル内では別名によって参照される。このディレクティブは、通常は、セクション名が有効なアセンブリ識別子でない場合に、そのセクションを指定するために使用される。

注： .secalias を使用してセクションに別名を付ける前に、セクションを定義しておかなければならない。

5.2.4.1 構文

```
.secalias section_name, “output-section_name”
```

section_name 入力ファイル内のセクションの名前。

output-section_name 出力ファイル内のセクションの名前。

5.2.4.2 例

この例は、.secalias ディレクティブを使用してセクション名の別名を定義する方法を示している。

```
.section                sec1, “ax”, “progbits”
. . .
.secalias               sec1, “sec++”
. . .
.text
   .xdata               sec1, 5
. . .
```

5.2.5 デバッグ・ツール用の行情報

デバッグ・ディレクティブは、オブジェクト・ファイル内にデバッグ情報を作成するための行情報を生成する。それぞれの行情報ディレクティブは、デバッグ・レコードを生成する。デバッグ・レコードは、各ディレクティブの後の命令によって生成されるコードの位置を指す。2 つのデバッグ・レコードが同じ位置を指すことはできない。したがって、2 つのデバッグ・ディレクティブの間にコード行があることを確認する必要がある。

デバッグ・レコード内の行情報は、バンドル内の個々の命令スロットに対応している。

`-d debug` コマンドライン・オプションを使用すると、IAS は `.bf`、`.ln`、および `.ef` ディレクティブを無視する。

次の汎用テンプレートを使用して、行情報を生成できる。

```
.file “source-file-name”
...
.proc entry [,...]
...
entry:
...
.bf entry, source-line-no
        //prologue code
.ln source-line-no
        // assembly code
.ln source-line-no
        // assembly code
...
...
.ln source-line-no
       // assembly code
.ef entry, source-line-no, procedure-size
       // epilogue code
.endp [entry]
```

5.2.6 #line のサポート

`#line` ディレクティブは、次のコード行の行番号を定義する。また、このディレクティブは、オブジェクト・ファイルのファイル名を置換できる。`#line` ディレクティブは、ユーザが明示的に入力することも、プリプロセッサに挿入させることもできる。

#line の定義は、診断メッセージと (コマンドライン・オプションで -d line を指定した場合に生成される) アセンブリレベルの行情報に影響を与える。詳細については、第 3 章の「コンパイル・モデル」を参照のこと。

IAS が認識する #line ディレクティブには、次のものがある。

`#line` *`line-no`*
: IAS は、続き番号に関係なく、次の行を現在のファイル内の *`line-no`* 行として処理する。

`#line` *`line-no "file-name"`*
: IAS は、次の行を *`"file-name"`* 内の *`line-no`* 行として処理する。このファイル名は、以前のオブジェクト・ファイル名を置き換える。このディレクティブは、複数のオペランドの間にカンマを入れることもできる。

5.2.7 予約されたシンボル

IAS は、あらかじめ予約されたシンボルを持っている。アセンブリ言語ファイル内で以下のシンボルを使用できる。

`@line`
: 現在の行番号を指定する整数。
使用例：
```
data8 @line
```

`@filename`
: 現在のファイル名を指定する文字列。
使用例：
```
stringz @filename
```

`@filepath`
: 現在のパスとファイル名を指定する文字列。
使用例：
```
stringz @filepath
```

5.2.8 仮想レジスタ割り当て

仮想レジスタ割り当て (Vral) は、レジスタ名の代わりにシンボリック名を使用できるようにする機能である。この機能は、レジスタまたはレジスタのグループを、意味のある名前で置き換える。これによって、コードが次のように改善される。

- 簡単に書ける。
- 素早く読める。
- 保守が簡単になる。

Vral が起動されると、アセンブラは、制御フローとデータ・フローを分析して、各レジスタの有効範囲を決定し、シンボリック名をユーザが割り当てたレジスタで置き換える。

1 つのディレクティブによって、Vral は、1 つの名前をレジスタのグループに割り当てることができる。これによって、アセンブラは、そのグループ内の個々のレジスタの使用を制御できる。その後、Vral は、各レジスタが安全に再使用されるように保証する役割を受け持つ。

シンボリック名をレジスタに割り当てるには、以下のディレクティブを使用する。

- `.vreg.allocatable`
- `.vreg.safe_across_calls`

レジスタ変数を宣言するには、以下のディレクティブを使用する。

- `.vreg.var Family, Xcounter`
- `.vreg.family LocalIntFamily, reg_range`

変数の定義の取り消しや再定義を行うには、以下のディレクティブを使用する。

- `.vreg.undef Xcounter`
- `.vreg.redef Xcounter`

Vral を使用するときは、以下の注釈が効果的である。

- `.br.target` 注釈
- `.entry` 注釈
- `.bank` 切り換え注釈

5.2.8.1 レジスタの割り当て

.vreg.allocatable ディレクティブは、割り当て用のレジスタを指定して、プロシージャ内のこの時点から、VRAL がそれらのレジスタを使用できるようにする。各プロシージャ内で、割り当てディレクティブを 2 つ以上使用することもできる。ただし、これらのレジスタの値が呼び出しの前後で維持されることは保証されない。このディレクティブは、次の構文を持つ。

```
.vreg.allocatable reg_range
```

reg_range 1 つのレジスタ、レジスタの範囲、またはその両方。

次の例では、整数レジスタ 14 ～ 26 とレジスタ 30 が割り当てられる。

```
.vreg.allocatable r14-26, r30
```

あるいは、.vreg.safe_across_calls ディレクティブを使用して、指定したレジスタの内容が呼び出しの前後で維持されることをアセンブラに指示することもできる。このディレクティブは、このディレクティブの後の外部プロシージャへの分岐が、指定されたレジスタにアクセスしたり破壊したりしないことを保証する。このディレクティブは、次の構文を持つ。

```
.vreg.safe_across_calls reg_range
```

reg_range この引数は、.vreg.allocatable ディレクティブで割り当てられたレジスタのみに限られない。

例：

```
.vreg.safe_across_calls f16, f18-f21
```

5.2.8.2 変数の宣言

次の構文を使用して、レジスタ変数を宣言できる。

```
.vreg.var Family, Xcounter
または
.vreg.var predef, Xcounter
```

Family 新しい変数のユーザ定義のファミリ名。

Xcounter 新しいレジスタ変数名。

predef 以下に説明する、4 つのあらかじめ定義されたファミリのうち 1 つ。

各変数は、1 つのレジスタ・ファミリに所属する。次の構文を使用して、ファミリを定義できる。

```
.vreg.family LocalIntFamily, reg_range
```

LocalIntFamily ユーザ定義のファミリ名。

reg_range 1 つのレジスタ、レジスタの範囲、またはその両方。

例：

```
.vreg.family MyLocalFamily, loc0-loc3
.vreg.family FpUsedRegisters, f17-f25
```

1 つのレジスタが 2 つ以上のファミリに所属することもある。各ファミリに所属するレジスタのタイプ (int、float など) は 1 つだけである。

アセンブラ構文には、4 つのあらかじめ定義されたファミリがある。

`@int` r1 ～ r127 のすべてのレジスタ。

`@float` f2 ～ f127 のすべてのレジスタ。

`@branch` b0 ～ b7 のすべてのレジスタ。

`@pred` p1 ～ p63 のすべてのレジスタ。

5.2.8.3 変数の定義の取り消しと再定義

VRAL ディレクティブは、プロシージャ内の .proc ディレクティブと .endp ディレクティブの間でのみ使用できる。VRAL ディレクティブによって宣言された変数は、変数の宣言からプロシージャの終了まで、または変数の定義が取り消されるか再定義されるまで有効である。

次の構文を使用して、プロシージャ内でその変数名をもう一度使用できるように、変数の定義を取り消すことができる。

```
.vreg.undef Xcounter
```

次の構文を使用して、変数を再定義できる (変数の定義を取り消す必要はない)。ただし、異なるファミリを指定することはできない。

```
.vreg.redef Xcounter
```

例 5-1 は、VRAL ディレクティブの使用例を示している。

例 5-1　　仮想レジスタ割り当ての例

```
.proc foo
.vreg.allocatable r19-r21, r27
.vreg.safe_across_calls r20, r21, p5-p6
.vreg.var @pred, HL1, L1H, HL2, L2H, HX, XH
.vreg.family MyGlobals, r19-r20
.vreg.var MyGlobals, High, Low1, Low2
foo::
alloc loc0 = 3,1,1,0
        ld8 High  = [in0]
        ld8 Low1 = [in1];;
        cmp.gt HL1, L1H = High, Low1
(L1H)   br.cond.sptk.few  LE
        sub out0 = High, Low1
GT:     add r22 = 32, r5;;
//...
END:
cmp.eq HX, XH = High, r22
(HX)    br.call.spnt.many rp = bar;;
(XH)    st8 [r23] = High
br.ret.sptk.clr b2
LE:     ld8 Low2 = [in2] ;;
cmp.gt HL2, L2H = High, Low2
(HL2)   sub out0 = High, Low2
(HL2)   br.cond.sptk.few  GT ;;
mov out0 = 0
//...
br.cond.sptk END
.endp foo
```

5.2.8.4　分岐ターゲット注釈

分岐ターゲット注釈 .br.target は、間接分岐の前に置かれ、分岐命令の分岐先アドレスをアセンブラに明示的に指示する。この注釈は、この注釈の直後の分岐命令にのみ適用される。.br.target 注釈は、次の構文を持つ。

.br.target*target1*[=*prob1*] [,*target2*[=*prob2*]...]

target　　次の間接分岐命令の分岐先を指定する。次のいずれかの値をとる。

ラベル　指定されたラベルがオブジェクト・ファイル・シンボル・テーブル内で定義されていない場合や、次の命令が間接分岐命令でない場合は、アセンブラはこの注釈を無視する。

`@fallthrough`	分岐は発生しない。
`@external`	分岐は現在の関数の外にジャンプする。

prob 指定された分岐先への分岐が発生する確率を示す実数。

例 5-2 は、分岐ターゲット注釈の例を示している。

例 5-2 分岐ターゲット注釈の使用法 1

```
.br.target a=0.6, b, @fallthrough=0.2, @external=0.1
```

例 5-3 は、別の分岐ターゲット注釈の例を示している。

例 5-3 分岐ターゲット注釈の使用法 2

```
br.target Target002
(p4)br.cond.sptk.many.b1
```

`Target002` プロシージャ内のラベルの名前。

5.2.8.5 レジスタ値注釈

レジスタ値注釈 `.reg.val` は、レジスタの内容をアセンブラに指示する。この情報は、依存関係違反の検出に使用される。

この注釈は、次の構文を持つ。

`.reg.val` *`reg, val`*

`reg` r0 ～ r127 *の任意の整数レジスタ。*

`val` 任意の実数。

例 5-4 は、`.reg.val` 注釈の例を示している。

例 5-4　　レジスタ値注釈の使用法

```
.reg.val r5,3
```

5.2.8.6　バンク・レジスタ注釈

デフォルトでは、アセンブラは、エントリー・ポイントでのレジスタ・バンクはバンク 1 であると見なす。この設定を変更するには、.bank ディレクティブを使用する。Vral では、このディレクティブは、bsw 命令を含むプロシージャにのみ必要である。

この注釈は、命令がどのレジスタ・バンクを参照するかをアセンブラに明示的に指示する。

.bank スイッチ注釈は、次の構文を持つ。

```
.bank      n
```

n　　0 または 1 を表す。

例 5-5 は、.bank 注釈の例を示している。

例 5-5　　バンク・スイッチ注釈の使用法

```
.proc A                 //entry annotation
A:
   .bank 0
   ...
   bsw.1
   ...
   bsw.0
   ...
.endp
```

5.2.8.7　アンワインド情報生成

IAS は、アンワインド記録を自動的に生成するために、プロシージャ・コードのスタティック分析を行う。この機能は、中間要素としてのプロシージャが、呼び出し先関数から呼び出し元プロシージャのアンワインド・ハンドラにスタック解放プロセスを安全に伝播させる必要がある場合に使用する。

アセンブラは、ファイル内のすべてのプロシージャについてアンワインド情報を構築する。これをプロシージャの最初のエントリー・ポイントから開始し、.endp まで続ける。

スタティック分析が完了しないとき、例えば間接分岐に分岐ターゲット注釈がない場合、IAS は警告メッセージを送信し、プロシージャに 1 つのプロローグ部と複数のエピローグ部があると仮定して分析を簡略化しようとする。ほとんどの場合、このような方法をとり、これが成功しない場合には IAS はエラー・メッセージを発行する。

アンワインド・ジェネレータは Itanium アーキテクチャ・ソフトウェアの表記規則に基づいている。『*ソフトウェア規則およびランタイム・アーキテクチャ・ガイド*』を参照のこと。アンワインドの生成は、`-X unwind` コマンドライン・オプションを使用して起動する。このフラグを使用すると、IAS はすべてのアンワインド・ディレクティブを無視し、警告を発行する。

診断メッセージ A

IAS は、疑わしい入力や不適当な入力を検出したり、何らかの操作に失敗した場合、診断メッセージを生成する。診断メッセージは、画面上に表示することも、ファイルに出力することもできる。詳細については、第 3 章の「エラー処理」を参照のこと。

本章では、診断メッセージの構文について説明した後、個々の診断メッセージについて番号順に説明する。

注： IAS は、ソース・コード内の対応する行の順序に従って診断メッセージを表示する。この順序は、エラーが検出された順序と一致するとは限らない。したがって、元になるエラーの診断メッセージより前に、派生したエラーの診断メッセージが表示されることがある。

A.1 診断メッセージのタイプ

IAS は、以下のタイプの診断メッセージを送信する。

致命的エラー・メッセージ IAS は、プログラムを終了させる不適当な入力を検出した。この場合、IAS はオブジェクト・ファイルを作成しない。致命的エラー・メッセージ番号のフォーマットは、次のとおりである。

```
A1xxx
```

エラー・メッセージ	IAS は、不適当な入力を検出した。この場合、プログラムの実行は続行されるが、IAS はオブジェクト・ファイルを作成しない。エラー・メッセージ番号のフォーマットは、次のとおりである。 A2xxx
警告メッセージ	IAS は、有効ではあるが疑わしい入力を検出した。この場合、プログラムの実行は続行され、IAS はオブジェクト・ファイルを作成する。警告メッセージ番号のフォーマットは、次のとおりである。 A3xxx

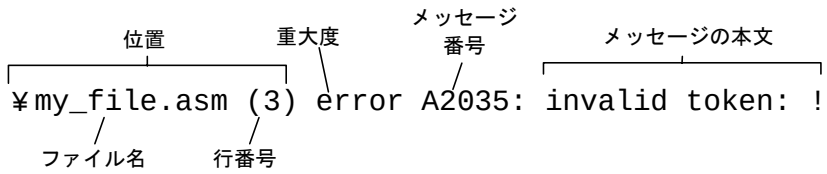
A.2 診断メッセージの構文

診断メッセージは、次の説明のように、エラーの位置、エラーのタイプ、およびエラーの簡単な説明を指定する。図 A-1 を参照のこと。

位置	ファイル名と行番号の情報によって、修正を必要とするコードの正確な位置がわかる。場合によっては、派生的なエラーが検出された位置を示すこともある。
重大度	この情報は、エラーの重大度を示す。
メッセージ番号	IAS メッセージ番号には、プリフィックス A が付いている。メッセージ番号に従って、メッセージの説明を見つけることができる。
メッセージの本文	このテキストは、不正確な入力または疑わしい入力について 1 行で説明する。

図 A-1 は、エラー・メッセージの例とメッセージの各要素を示している。

図 A-1.　診断メッセージの構文の例



診断メッセージの説明のフォーマットを以下に示す。

メッセージ番号　　メッセージの本文

メッセージの詳しい説明。

A.3　致命的エラー・メッセージ

ここでは、致命的エラー・メッセージについて説明する。致命的エラーが検出されると、IAS は、オブジェクト・ファイルを作成せずにただちに終了する。IAS が表示する致命的エラー・メッセージには、次のものがある。

A1012　　`cannot open input file` ***`file`***

IAS はこのファイルを開けなかった。この致命的エラー・メッセージは、通常はファイル名またはパスが不適当であるために発生する。

A1013　　`cannot open input file` ***`file`*** `included from` ***`file (line)`***

IAS はこのファイルを開けなかった。この致命的エラー・メッセージは、通常は `.include` ディレクトリ内のファイル名またはパスが不適当であるために発生する。

A1014　　`cannot open registers allocation log file` ***`file`***

IAS は、仮想レジスタ割り当ての結果を記録したファイルを開けなかった。サフィックス `.vra` のファイル名が使用されていないかどうか確認すること。サフィックス `.vra` の読み取り専用ファイルをすべて削除すること。

A1015 `creation of section ***section*** failed: ***reason***`

アセンブラは、指定された理由のために、セクションを作成できなかった。

A1018 `too many errors: ***number***`

エラーの最大許容数を超えたため、IAS の実行が終了した。`-E ***n***` コマンドライン・オプションを使用して、許容されるエラーの数を設定することができる。

A1020 `section stack underflow`

`.popsection` ディレクティブが空のスタックを操作している。このディレクティブについての詳細は、『*Intel® Itanium™ アーキテクチャ・アセンブリ言語リファレンス・ガイド*』を参照のこと。

A1021 `unable to open ***file*** as an error file`

IAS は、コマンド・ラインで診断ファイルとして指定されたファイルを開けなかった。同じ名前のファイルが、他のプロシージャによってロックされている可能性がある。

A1022 `command-line option is missing an argument ***Usage message***`

このコマンドライン・オプションには引数が欠けている。この致命的エラー・メッセージは、IAS コマンド・ラインの使用法メッセージも表示する。IAS コマンド・ラインの使用方法については、第 3 章「コマンドライン・オプション」を参照のこと。

A1025 `unknown command-line option ***option Usage message***`

IAS はこのコマンドライン・オプションを認識できない。この致命的エラー・メッセージは、IAS コマンド・ラインの使用法メッセージも表示する。詳細については、第 3 章「コマンドライン・オプション」を参照のこと。

A1026 `***option*** command-line option is incompatible with ***sub-argument*** sub-argument ***Usage message***`

指定されたサブ引数は、このコマンドライン・オプションには無効である。この致命的エラー・メッセージは、IAS コマンド・ラインの使用法メッセージも表示する。IAS コマンドライン・オプションおよびのサブ引数についての詳細は、第 3 章「コマンドライン・オプション」を参照のこと。

A1027 `.include directive has illegal placing/format`

`.include` ディレクティブが不適当である。この致命的エラー・メッセージは、ファイル名オペランドとして文字列以外の値を入力したために発生する。このディレクティブについての詳細は、『*Intel® Itanium™ アーキテクチャ・アセンブリ言語リファレンス・ガイド*』を参照のこと。

このメッセージの原因となるコードの例を次に示す。

```
.include data.s
```

A1050 `virtual register allocation failed: not enough allocatable registers from family` ***`family`***

仮想レジスタ割り当てディレクティブによって割り当てられたレジスタの数が少なすぎる。

A1099 `nesting level (`***`number`***`) of .include directive exceeded for included file` ***`file`***

`.include` ディレクティブが、IAS のネスト制限を超えるレベルにネストされている。IAS は、最大 20 までのネスト・レベルを許可している。

A.4 エラー・メッセージ

ここでは、エラー・メッセージについて説明する。エラーが発生した場合、IAS の実行は終了しないが、オブジェクト・ファイルは作成されない。IAS が表示するエラー・メッセージには、次のものがある。

A2000 `too long symbol name`

シンボル名の長さは 4096 文字を超えてはならない。

A2023 `there should be a prologue region in the function`

このディレクティブには、関数内のプロローグ領域が必要である。

A2024 `the personality routine is not defined for the language specific data`

このディレクティブを使用するには、ディレクティブの前にパーソナリティ・ルーチンが定義されている必要がある。`.handlerdata` ディレクティブの前に、`.personality` ディレクティブを追加すること。これらのディレクティブについての詳細は、『*Intel® Itanium™ アーキテクチャ・アセンブリ言語リファレンス・ガイド*』を参照のこと。

A2025 `directive ".proc" is not allowed within section ".xdata."`

`.xdata` セクション内で `.proc` ディレクティブを使用することはできない。

A2026 `section switch is not allowed within handlerdata region`

`handlerdata` 領域内でセクションを切り替えることはできない。

A2027 `debug directive points outside the function`

`debug` ディレクティブのオペランドが、現在の関数の外側を指している。

A2028 `***directive*** is allowed only within an explicit bundle`

このディレクティブは、明示的なバンドルの中で指定された場合にのみ有効である。このディレクティブは、2 つの中括弧 "{" と "}" の間に置くこと。

A2029 `***directive*** is allowed only within an explicit bundle`

このディレクティブは、明示的なバンドルの中で指定された場合は無効である。このディレクティブが 2 つの中括弧 "{" と "}" の間に置かれていないことを確認すること。

A2030 `misplaced or missing '}'`

中括弧の不一致がある。前のバンドルの中括弧の構成を確認すること。

A2031 `Unclosed parenthesis at start-of-statement`

この文は、左括弧 "(" で始まっているが、右括弧 ")" が欠けている。この文の修飾プレディケートは閉じられていない。

A2032 `Unexpected ***element*** instead of predicate register`

プレディケート・レジスタのために予約されている位置に、プレディケート・レジスタ以外の要素が指定されている。

正しい使用例を次に示す。

```
(p62) add r2 = r3, r6
```

この例では、p62 がプレディケート・レジスタである。

このエラー・メッセージの原因となるコードの例を次に示す。

```
(p64) add r2 = r3, r6
```

プレディケート・レジスタの範囲は p0 ～ p63 である。

A2033 `Unexpected ***element*** instead of tag`

タグのために予約されている位置に、タグ以外の要素が指定されている。

正しい使用例を次に示す。

```
.save pr, r3, T
[T:] mov r3=pr
```

A2034 Unexpected token at end-of-statement: *token*

このトークンは文の終わりには使用できない。そのトークンを削除または変更すること。

このエラー・メッセージの原因となるコードの例を次に示す。

```
add r1=r2,r3,
```

A2035 invalid token: *token*

このトークンは無効である。

このエラー・メッセージの原因となるコードの例を次に示す。

```
add r1=r2,r3!
```

A2036 illegal usage of reserved register: *register*

このレジスタは予約済みレジスタである。他のレジスタを使用すること。

このエラー・メッセージの原因となるコードの例を次に示す。

```
mov r5=ar8
```

A2037 Unexpected token at start-of-statement: *token*

文の先頭では、このトークンは無効である。そのトークンを削除または移動すること。

このエラー・メッセージの原因となるコードの例を次に示す。

```
)add r1=r2,r3
```

このエラー・メッセージは、ニーモニックのスペルが間違っているために発生することもある。このエラー・メッセージの原因となる、ニーモニックのスペルの誤りの例を次に示す。

```
br.cal b5=L
L:
```

A2038 *symbol/section* already aliased as *name*

このシンボルまたはセクションには既に他の別名が定義されているため、この段階でこのシンボルまたはセクションに別名を付けることはできない。

A2039 `label already defined: label`

このラベルは他の箇所で既に定義されているため、この段階でこのラベルを定義することはできない。この定義には新しいラベルを使用すること。

A2040 `Unexpected token token`

指定されたトークンは、この位置では使用できない。

A2042 `symbol symbol for definition type is already defined`

このシンボルは他の箇所で既に定義されている。この定義には新しい名前を使用すること。

このエラー・メッセージの原因となるコードの例を次に示す。

```
L:
L=8
```

A2047 `unexpected character character in string`

16 進エスケープ・シーケンスに、使用できない文字が含まれている。16 進エスケープ・シーケンスに使用できるのは、0 ～ 9 の数字と A ～ F の文字である。

正しい 16 進エスケープ・シーケンスの例を次に示す。

`\xa`, or `\xD9`

A2048 `illegal bundle brace in automatic mode`

IAS が自動アセンブリ・モードで中括弧 ({) または (}) を検出した。自動モードは、`.auto` ディレクティブによって指定される。

A2049 `relocatable expressions based on symbols symbol and symbol from different sections cannot be subtracted`

これらの再配置可能な式は、異なるセクションから得られたものである。2 つの再配置可能な式を減算する場合、それらは同じセクションから得られたものでなければならない。

A2050 `cannot subtract relocatable expressions based on an external or common symbol`

2 つの再配置可能な式のうち一方または両方が、おそらく外部シンボルまたは共有シンボルに基づいている。2 つの再配置可能な式を減算する場合、それらは同じセクション内で定義されたシンボルに基づいていなければならない。

A2051 `wrong operand parenthesis structure`

オペランドの括弧の構成が不適当である。

このエラー・メッセージの原因となるコードの例を次に示す。

```
nop ((5+3)
```

A2052 `wrong operand bracket '['']' structure`

オペランドの大括弧の構成が不適当である。

このエラー・メッセージの原因となるコードの例を次に示す。

```
ld8 r6 = [r4]]
```

A2055 `illegal argument [`***`argument-type`***`] for unary-operator `***`operator`***`, or misplaced/missing operator`

指定された単項演算子である場合は、この引数タイプは無効である。

このエラー・メッセージの原因となるコードの例を次に示す。

```
and r3=r2,+r5
```

A2056 `missing arguments for binary-operator: `***`operator`***

この二項演算子の引数が欠けている。

このエラー・メッセージの原因となるコードの例を次に示す。

```
mov rr[] = r6
```

A2057 illegal argument-pair [left: ***argument*** right: ***argument***] for binary-operator ***operator***, or misplaced/missing operator

これらの引数を組み合わせて演算を実行することはできない。このエラーの原因となる、よく見られる誤りは、二項演算子とそれに対して無効な (少なくとも 1 つの) オペランドを組み合わせて使用することである。

このエラー・メッセージの原因となるコードの例を次に示す。

```
or r4 = dbr[f4], r6
```

A2061 a sequence of unary-operator ***operator*** and ***element*** is illegal

この単項演算子を、指定された要素の後に続けることはできない。

このエラー・メッセージの原因となるコードの例を次に示す。

```
add r1 = ~, r2
```

A2063 a sequence of binary-operator ***operator*** and operands ***operand1*** and ***operand2*** is illegal

この二項演算子を、指定された 2 つのオペランドの後に続けることはできない。

演算子の位置が間違っている可能性がある。例：`or r3 = 4 5+, r6`

正しいコードは次のようになる。`or r3 = 4+5, r6`

A2065 wrong operand syntax

このコード行のオペランド構文は不適当である。

オペランドの組み合わせが無効であるために、このエラー・メッセージが表示される場合もある。詳細については、エラー・メッセージ A2069 および A2070 を参照のこと。

A2066 `missing operator [possibly intended binary +/- taken as unary]`

演算子が抜けている。演算子の位置が間違っている可能性がある。
例： `2*-3 5`

正しいコードは次のようになる。`2*3-5`

あるいは、演算子の間のカンマが抜けている可能性がある。
例：`add r1 = r2 r3`

正しいコードは次のようになる。`add r1 = r2, r3`

A2067 `incorrect tag usage: ***tag*** [might need to use label instead]`

このタグは不適当である。タグをラベルで置き換えてみること。

A2068 `value of operand ***operand number*** for ***element*** is not available when needed`

このオペランド値は、必要になった時点で使用できない。IAS は、このような順方向参照を実行できない。

このエラー・メッセージの原因となるコードの例を次に示す。

```
.skip L1-L2
L1: data8 1
L2:
```

A2069 `illegal operand combination for ***element***`

この命令のニーモニックとオペランドの間に不整合がある。このエラー・メッセージには、オペランドが欠けている、オペランドのタイプが不適当である、シンボルとして解釈されるレジスタ名が無効である、ニーモニックの選択が不適当である、などの原因が考えられる。

A2070 `illegal operand ***operand*** for ***element***`

このオペランドは、指定された要素に対して不適当である。

A2072 `invalid section attribute: attribute`

このセクション属性は無効である。セクション属性は、オブジェクト・モジュール・フォーマット (OMF) によって異なる。有効な属性には、a、w、x、および s などがある。

A2073 `more than one comdat section flag defined: flag`

comdat セクションに対して定義できる comdat フラグは 1 つだけである。フラグの定義には、D、S、E、または Y などがある。フラグは大文字と小文字を区別する。

A2074 `comdat flag is only applicable for comdat section`

この comdat フラグは、非 comdat セクションに対して定義されている。

A2075 `comdat section flag not defined`

comdat セクションには、1 つの comdat フラグが定義されていなければならない。フラグの定義には、D、S、E、または Y などがある。フラグは大文字と小文字を区別する。

A2076 `comdat section section associative symbol is not defined`

comdat セクションには、少なくとも 1 つのラベルが定義されていなければならない。

A2077 `invalid section type: type`

このセクションのタイプは無効である。有効なセクションのタイプは、progbits、nobits、comdat、および note である。

A2078 `absolute sections section [address to address] and section [starting at address] overlap`

指定された 2 つの絶対セクションが重複している。

A2079 `absolute section section [starting at address] exceeds the 64-bit limit by value`

この絶対セクションは、64 ビット・アドレス空間の上限値を超えている。指定された値は、上限値をどれだけ超えているかを示す。

A2080 `relocatable expression for ***element*** requires -p32 or -M ilp32 command-line options`

この再配置可能な式は、現在のコンパイル・モデルのコマンドライン・オプションと競合する。詳細については、第 3 章の「コンパイル・モデル」の項を参照のこと。

A2081 `nobits section ***section*** cannot be written to`

この nobits セクションへの書き込みを試みた。nobits セクションにデータを書き込むことはできない。この問題を解決するには、nobits セクション内のデータを削除する、セクションのタイプを progbits に変更する、.skip ディレクティブを使用してデータを置き換える、などの処置が必要である。

A2082 `nobits section ***section*** contains data`

nobits セクションはデータを格納できない。この問題を解決するには、nobits セクション内のデータを削除する、セクションのタイプを progbits に変更する、.skip ディレクティブを使用してデータを置き換える、などの処置が必要である。

A2083 `integer constant token does not fit in ***number*** bits: ***token***`

入力された数値トークンのビット数が、整数定数の許容範囲を超えている。

A2084 `integer number does not fit in ***number*** bits: ***number***`

この命令には、この数値は大きすぎる。この数値は、おそらく内部計算の結果である。

A2086 `alignment request is too big: ***alignment***`

アライメント要求は、2^{32}-1 までに制限されている。

A2087 `alignment request is not a power of 2: ***alignment***`

アライメント要求は、2 の累乗でなければならない。

A2088 symbol ***symbol*** is undefined

このシンボルは、オブジェクト・ファイルのシンボル・テーブル内に見つからない。グローバル・シンボルまたは弱いシンボルは、定義または宣言されている必要がある。ローカル・シンボルは、定義されている必要がある。

A2089 illegal global declaration of assigned symbol: ***symbol***

オブジェクト・ファイルのシンボル・テーブル内の宣言されたシンボルを代入することはできない。代わりに、等式 (==) 文を使用することができる。

このエラー・メッセージの原因となるコードの例を次に示す。

```
B = 8
.global B
```

A2090 assigned/equated symbol ***symbol*** cannot be used in ***statement***

代入文または等式文で定義されたシンボルを、この文中で使用することはできない。

このエラー・メッセージの原因となるコードの例を次に示す。

```
A == L
L:
.weak A = S
S:
```

A2091 symbol ***symbol*** is undefined

このシンボルは未定義である。

A2092 symbol size of ***symbol*** exceeds 32-bit word size

共有シンボルのサイズが、64 ビットの上限値を超えている。

A2093 symbol ***symbol*** is already bound as ***binding***

このシンボルのバインディングは既に宣言されている。シンボルのバインディングを再定義することはできない。

A2094 `symbol size of ***symbol*** is already set to ***size***`

このシンボルのサイズは既に宣言されている。シンボルのサイズを再定義することはできない。

A2095 `symbol type of ***symbol*** is already set to ***type***`

このシンボルのタイプは既に宣言されている。シンボルのタイプを再定義することはできない。

A2096 `***type*** is an illegal type for symbol ***symbol***`

このタイプは、有効なシンボル・タイプ (`@notype`、`@object`、および `@function`) ではない。

A2097 `instruction cannot be predicated`

この命令にはプレディケートを使用できない。詳細については、「用語集」を参照のこと。

A2098 `there is no template for this combination of instructions in a bundle`

命令がテンプレートに適合するように命令を再編成するか、またはバンドルの自動作成を実行すること。

A2100 `one and only one operand must follow an assignment/equation sign`

代入記号または等式記号の後に 1 つのオペランドがあることを確認すること。

A2101 `invalid section name: ***section***`

セクション名は、任意の有効な識別子である。`.secalias` ディレクティブを使用して、オブジェクト・ファイルのセクション・テーブル内にセクション名を作成できる。

このメッセージは、新しいセクションを定義するとき、属性またはフラグ、あるいはその両方が欠けているために発生する。

A2103 `symbol ***symbol*** is already defined as a section name`

セクション名とシンボル名が競合している。セクションとシンボルに同じ名前を選択してはならない。

A2104 `invalid operand immediate value: ***value***`

この命令オペランドには、この即値は無効である。詳細については、『*Intel® Itanium™ アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル*』を参照のこと。

このエラー・メッセージの原因となるコードの例を次に示す。

```
fetchadd4.acq r3 = [r4], 7
```

A2105 `this relocatable expression does not fit in ***number*** bits`

この命令には、この再配置可能な式は長すぎる。詳細については、『*Intel® Itanium™ アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル*』を参照のこと。

A2107 `requested register stack frame size ***size*** exceeds register stack limit ***limit***`

要求されたレジスタ・スタック・フレームのサイズが 96 を超えている。レジスタ・スタック・フレームのサイズは、入力レジスタ、ローカル・レジスタ、および出力レジスタの合計である。

A2108 `input stack register cannot exceed in***xx***`

要求された入力レジスタは、現在の入力レジスタ・フレーム ***xx*** 内にない。レジスタ・フレームは、前の `alloc` 命令または `.register` ディレクティブで定義される。

A2109 `local stack register cannot exceed loc***xx***`

要求されたローカル・レジスタは、現在のローカル・レジスタ・フレーム ***xx*** 内にない。レジスタ・フレームは、前の `alloc` 命令または `.register` ディレクティブで定義される。

A2110 `output stack register cannot exceed out***xx***`

要求された出力レジスタは、現在の出力レジスタ・フレーム ***xx*** 内にない。レジスタ・フレームは、前の `alloc` 命令または `.register` ディレクティブで定義される。

A2111 `The requested number of rotating-registers ***number*** is not a multiple of 8`

ローテート・レジスタの数は、8 の倍数でなければならない。

A2112 `The requested number of rotating-registers ***number*** is larger than the register stack frame size ***number***`

ローテート・レジスタの数は、レジスタ・スタック・フレームのサイズを超えることはできない。レジスタ・スタック・フレームは、入力レジスタ、ローカル・レジスタ、および出力レジスタの合計である。

A2113 `Loop dependency is detected in equate expression for symbol ***symbol***`

指定されたシンボルの等式にループ依存関係がある。逆方向参照または再帰参照がないかどうか確認すること。

逆方向参照の結果、このエラー・メッセージの原因となるコードの例を次に示す。

```
x==y
y==x
```

A2114 `invalid operand type: ***symbol***`

この文には、このオペランド・タイプは無効である。

A2115 `stop (;;) for empty instruction group`

このストップは、命令を含まない命令グループを作成する。このストップを削除すること。

このエラー・メッセージの原因となるコードの例を次に示す。

```
;;
add r1=r2,r3
```

A2116 `bundle content contradicts template request`

バンドルの内容は、異なるテンプレートを要求している。他のテンプレートを選択するか、またはテンプレート・ディレクティブを省略すること。

このエラー・メッセージの原因となるコードの例を次に示す。

```
{
.mii
nop.m '0'
nop.f '2'
nop.i '1'
}
```

A2117 `same ***register type*** register [***register***] cannot be used for both destinations`

この命令は、同じ 2 つのデスティネーションに書き込めない。

A2118 `cannot use the same registers for base and destination in the post-increment ***form*** form of the load instruction`

インクリメント後のロード命令は、ベース・レジスタとデスティネーション・レジスタに同じレジスタを使用できない。このエラー・メッセージの原因となるコードの例を次に示す。

```
ld8 r9 = [r9], r4
```

A2121 `alias name ***name***[***number***] is not defined in .rotX directive`

前の .rotr、.rotf、または .rotp ディレクティブ内でローテート・レジスタを定義すること。

A2122 `constant ***integer string*** does not conform to ***style/radix*** style`

この整数文字列のフォーマットは、前の .radix ディレクティブによって定義された現在のスタイルに適合しない。

A2124 `previous procedure is not yet ended`

現在のプロシージャが終了するまで、新しいプロシージャを開始することはできない。.endp ディレクティブを使用して、現在のプロシージャを終了すること。

A2126 `there is an open procedure in section: ***section***`

このセクション内に終わっていないプロシージャがある。.endp ディレクティブを使用して、現在のプロシージャを終了すること。

A2128 `line entry is valid only in ***type*** section`

現在のセクションのタイプは、デバッグ情報ディレクティブを受け入れない。

A2129 `offset operand for ***element*** must be greater or equal to current location counter`

このオフセット・オペランドは、現在の位置より上位のアドレスを指定しなければならない。

このエラー・メッセージの原因となるコードの例を次に示す。

```
L:
.skip 5
.org L
```

A2130 `somewhere, symbol ***symbol*** is equated to an incompatible type`

このシンボルは、適合しないシンボル・タイプと等価にされている。IAS は、等式が無効であることを認識したが、この等式の正確な位置を確定できない。

このエラー・メッセージは、無効な循環定義によって発生することもある。このエラー・メッセージの原因となるコードの例を次に示す。

```
B == r5
.global B
```

A2131 `equation of symbol ***symbol*** is based on undefined symbol ***symbol***`

等式の右辺のシンボルのうち 1 つが宣言されていないため、IAS はこの等式を解けない。

A2132 `illegal register value ***number***`

レジスタ番号が無効である。有効なレジスタ番号は、レジスタのタイプによって異なる。

A2133 `reference symbol ***symbol*** is not defined in the current section`

このシンボルは、現在のセクション内で定義されていなければならない。

A2134 `***element*** is supported for COFF32 object file format only`

この要素は、COFF32 以外のファイル・フォーマットではサポートされない。

A2135 `there is no open debug function`

`.ef` および `.ln` ディレクティブを使用するには、`.bf` ディレクティブによってデバッグ関数が開始されていなければならない。このディレクティブについての詳細は、『*Intel® Itanium™ アーキテクチャ・アセンブリ言語リファレンス・ガイド*』を参照のこと。

A2136 `symbol` ***`symbol`*** `does not match the current debug function`

行エントリは、現在のプロシージャ・シンボルを参照しなければならない。

A2137 `previous debug function is not yet ended`

以前の `.bf` 関数がまだ終了していないため、`.bf` ディレクティブを使用してデバッグ関数を開始しようとしたが失敗した。`.ef` ディレクティブを使用して、以前のデバッグ関数を終了すること。

A2138 `two debug directives pointing to the same instruction`

`.ln` ディレクティブは、それに続く最も近い命令を指す。命令の前に 2 つ以上の `.ln` ディレクティブを置くことはできない。

A2139 `there is an open debug function in section` ***`section`***

このセクション内に終わっていない `.bf` ディレクティブがある。`.ef` ディレクティブを使用して、以前のデバッグ関数を終了すること。

A2140 `source file is not defined`

デバッグ情報ディレクティブを使用するには、ソース・ファイルが定義されていなければならない。`.file` ディレクティブを使用して、ソース・ファイルを定義すること。

A2141 ***`unwind directive`*** `cannot be placed in the` ***`location`***

このアンワインド・ディレクティブを、指定された位置に置くことはできない。このディレクティブについての詳細は、『*Intel® Itanium™ アーキテクチャ・アセンブリ言語リファレンス・ガイド*』を参照のこと。

A2142 `unwind directive` ***`directive`*** `is not within a function`

このアンワインド・ディレクティブはプロシージャの中にない。`.proc` ディレクティブを使用して、プロシージャを開始すること。

A2143 `tag operand ***tag*** in the unwind directive is not defined within the current region`

この指定されたアンワインド・ディレクティブのオペランド・タグは、現在のアンワインド領域 (プロローグ領域または本体領域) の外側の命令を指している。

A2144 `unwind directive points outside the current region`

このアンワインド・ディレクティブは、現在のアンワインド領域 (プロローグまたは本体領域) の外側の命令を指している。

A2145 `the first unwind directive must point to the procedure ***procedure*** entry point`

最初のアンワインド・ディレクティブとプロシージャのエントリー・ポイントの間に命令がある。この命令を削除または移動して、最初のアンワインド・ディレクティブがこのプロシージャのエントリー・ポイントのアドレスを指すようにすること。

このエラー・メッセージの原因となるコードの例を次に示す。

```
.proc foo
.prologue 0x1, r1

nop 0
foo::
.endp
```

このコードを修正するには、エントリー・ポイント `foo::` の直前に `.prologue` ディレクティブを置く必要がある。

A2146 `***unwind*** directive interrupts uncompleted set of spill instructions`

前の `.save` ディレクティブによって定義された、一連の連続するスピル命令が、他のアンワインド・ディレクティブによって中断されている。

A2148 `***directive*** directive with no spill is invalid`

アンワインド・ディレクティブ内のスピル引数の数をゼロに定義することはできない。

A2149 `duplicate spill of the same ***register type*** register is invalid`

指定されたレジスタ・タイプのスピル領域は、アンワインド領域に 1 つだけしか入れられない。

A2150 `unwind directive ***directive*** is already specified in the current procedure`

このディレクティブは、プロシージャ内で一度だけしか使用できない。

A2152 `explicit empty bundle is illegal`

明示的なバンドルには、少なくとも 1 つの命令が含まれていなければならない。

A2153 `no ***type*** registers are allowed within current register stack frame`

このレジスタ・スタック・フレームには、指定されたタイプのレジスタが 1 つもない。

A2154 `vral directive ***dirname*** is not within a function`

仮想レジスタ割り当て (Vral) ディレクティブは、プロシージャ内に置かれた場合にのみ有効である。

A2173 `both destination fp registers refer to the same register bank`

デスティネーション・レジスタは、1 つの奇数番号の浮動小数点レジスタと 1 つの偶数番号の浮動小数点レジスタを指定していなければならない。

A2180 `***register*** register dependency violation with ***line***`

指定された行にレジスタの依存関係がある。この依存関係が避けられるように、いずれかの行を再配置してみること。2 つの依存する要素の間にストップ (`;;`) を置くこと。

A2181 `instruction must be ***position*** in an instruction group`

このエラー・メッセージは、IAS 依存関係違反機能によって生成される。この命令は、命令の必要条件に従って、命令グループ内の最初または最後に置くこと。

A2186 `statement element is not allowed after statement element statement`

連続する文の要素をこの順序で組み合わせることはできない。

このエラー・メッセージの原因となるコードの例を次に示す。

```
foo:  .radix C
```

A2187 `alias for symbol type "symbol name" is already defined`

指定されたシンボル名は、他のシンボルの別名として既に使用されている。

A2192 `symbol name used in @fptr operator must be a function`

`@fprt` オペランドの後に続く演算子は、関数でなければならない。

A2194 `the directive: directive is not supported in this configuration`

現在のコマンドライン・オプションを指定して IAS を実行する場合、指定されたディレクティブはサポートされない。詳細については、第 3 章「コマンドライン・オプション」を参照のこと。

A2197 `Radix stack underflow`

基数スタックが空になっている。空のスタックに対してポップ操作を実行することはできない。

A2198 `operand no. number: relocation's addend doesn't fit in size bits`

指定された再配置加数は大きすぎる。加数が指定されたサイズを超えないようにすること。

A2199 `privileged instruction instruction rejected`

現在の IAS の設定では、特権命令は拒否される。この命令は特権命令であるため、拒否される。

A2200 `line group size ***value*** exceeds 32 bits word size or less than actual size`

.EF ディレクティブの 3 番目のパラメータ (コード・サイズ) が無効である。

A2201 `Global label cannot begin with dot`

グローバル・ラベルをドット "." 文字で始めることはできない。この問題を解決するには、ラベルが表示されるようにする、ラベルを変更する、またはグローバルな定義をシンボル名の定義で置き換える必要がある。

A2202 `Division by zero`

式の分母がゼロになっている。この演算の結果は未定義である。

A2203 `invalid register type for register range operand`

レジスタ範囲オペランドは、整数レジスタ、浮動小数点レジスタ、分岐レジスタ、またはプレディケート・レジスタのペアで構成されていなければならない。

A2205 `virtual register has already been defined`

プロシージャ内で、このレジスタに対してディレクティブ .vreg.var が既に指定されており、.vreg.undef ディレクティブは使用されていない。

A2207 `inconsistent request for allocation of even/odd floating point registers`

浮動小数点仮想レジスタについて、同じ有効範囲上で、偶数 / 奇数に関する両立しない必要条件が指定されている。

このエラー・メッセージの原因となるコードの例を次に示す。

```
.vreg.var @float, vfp
fand vfp = f8,f9
(p2) ldfps vfp, f4 = [r3] // vfp should be odd
;;
(p2) ldfps vfp, f5 = [r4] //vfp should be even
for f12 = vfp, f12
```

A2208 an ambiguity in register bank setting

いくつかの命令に対して、2 つの .bank 注釈が競合している。通常は分岐命令が原因である。

A2209 temporary label can not be aliased

一時的ラベルに対してディレクティブ .alias を使用してはならない。

A2210 More than one template selection directive for current bundle

現在のバンドルに 2 つ以上のテンプレートが割り当てられている。最適なテンプレートを選択し、その他のテンプレートを削除すること。

A2211 Template selection directive allowed only as first statement in explicit bundle

テンプレート選択ディレクティブは、バンドルを開始する中括弧 "{" の直後に置くこと。

A2212 Symbol ***symbol name*** was not defined ***location***

指定されたシンボルはこのプロシージャ内で定義されていない。

A2213 ***feature*** has different syntax in COFF32 object file format

指定された機能は、間違ったファイル・フォーマットの構文を使用している。詳細については、『*Intel® Itanium™ アーキテクチャ・アセンブリ言語リファレンス・ガイド*』を参照のこと。

A2214 The right-hand expression of the assignment contains forward reference

代入式の右辺に順方向参照が含まれていてはならない。

A2215 Somewhere, symbol ***assignment symbol*** is assigned to expression that contains forward reference to symbol ***undefined symbol***

代入記号を含む式は、未定義のシンボルへの順方向参照を使用できない。

A2216 Missing the right-hand operand of the assignment

コードの行が完成していない。代入式の右辺オペランドを追加すること。

A2217 Symbol ***symbol name*** was not defined within procedure

指定されたシンボル名は、このプロシージャ内で定義されていなければならない。この警告の原因となるコードの例を次に示す。

```
.proc  A
mov r1=r2
.endp
A:
```

A2219 Invalid usage of an undefined symbol with addend

ここでは加数を含む未定義シンボルを使用できない。再配置が解決できない。

A2221 somewhere, symbol ***symbol*** is equated to a value/offset ***offset*** out of positive ***size*** bit range

指定されたシンボルがディレクティブによってシンボル・テーブルに入れられたが、このシンボルには上限値を超える値が代入されている。代入される値を許容範囲内にすること。

A2222 symbol *sym_name* is unknown, add alias is ambiguous in vral mode

不明な即値を持つ別名 add を 2 番目のオペランドとして使用し、仮想レジスタを 3 番目のオペランドとして使用すると、アセンブラが混乱して割り当てが失敗することがある。

この問題を解決するには、add 命令の前に即値を定義するか、または add の代わりに adds または addl を明示的に使用する必要がある。

この警告の原因となるコードの例を次に示す。この例では、アセンブラは、adds を選択できるほど A が小さいかどうか判断できない。A が十分に小さい場合は、Vr1 の割り当てに制限はない。それ以外の場合は、addl が選択され、Vr1 は r0 ～ r3 の範囲内に制限される。

```
add r6 = A, Vr1
...
A == 5
```

このコードを修正するには、次のようにする。

```
adds r6 = A, Vrl
```

または

```
A == 5
add r6 = A, Vr1
...
```

A2223 invalid syntax of Register File operand

レジスタ・ファイル・オペランドの構文が不適当である。

この警告の原因となるコードの例を次に示す。

```
mov dbr=r5
```

正しい構文の例を次に示す。

```
mov dbr[r6]=r5
```

A2225 illegal instruction

指定された命令は、Itanium アーキテクチャ構文では無効である。

A2226 illegal usage of *register* in RFP model

コマンドライン・オプション -M rfp が起動されると、使用可能な浮動小数点レジスタの範囲は F6 ～ F11 に制限される。その他の浮動小数点レジスタにアクセスしようとすると、このエラーが発生する (ELF64 のみ)。

A2227 `Associative Comdat section ***sec_name*** must have an associated section`

タイプ A (アソシエティブ) の comdat セクションについて関連するセクションを示す必要がある。

A2228 `symbol name ***sym name*** contains period, not allowed in the COFF32 format`

COFF32 フォーマット・テーブルでは、シンボルの中にピリオドが含まれてはならない。

A.5 警告メッセージ

警告メッセージは、有効ではあるが疑わしいアセンブリ言語コードを報告する。警告が生成されても、IAS の実行と出力ファイルの生成は中止されない。IAS が表示する警告には、次のものがある。

A3100 `unexpected usage of ***tag*** tag in ***element***`

このタグはこの方法では使用できない。多くの場合、この問題を解決するには、タグをラベルで置き換えればよい。

A3102 `***symbol*** is a symbol and also an alias name`

この名前は、同時にシンボル名と別名として定義されている。このため、出力ファイルに、同じ名前を持つ 2 つの異なるシンボルが含まれている。

A3103 `***register*** register dependency violation with ***line***`

指定された行にレジスタの依存関係の違反がある。この依存関係が発生しないように、いずれかの行を再配置してみること。2 つの依存する要素の間にストップ (`;;`) を置くこと。

A3105 `symbol ***name*** defined in a TLS section can't be referenced this way`

指定されたシンボル名は、addl 命令の secrel 演算子のオペランドとして参照されていなければならない。詳細については、『*Intel® Itanium™* アーキテクチャ・アセンブリ言語リファレンス・ガイド』を参照のこと。

この警告の原因となるコードの例を次に示す。

```
addl r3 = @gprel(sym), r3
```

正しいシンボル参照の例は次のようになる。

```
addl r3 = @secrel(sym), r3
```

A3106 `symbol ***symbol*** is undefined`

このシンボルは、オブジェクト・ファイルのシンボル・テーブル内に見つからない。グローバル・シンボルまたは弱いシンボルは、定義または宣言されている必要がある。ローカル・シンボルは、定義されている必要がある。

A3200 `32-bit relocatable expression in ***element***`

このモデルのアドレス・サイズの条件は 32 ビットであるが、指定された要素はこの条件にに対応していない。

A3201 `alignment operand of symbol ***symbol*** is ***relation*** than the size operand`

COFF32 出力ファイル・フォーマットは、シンボル・アライメント・フィールドを持たない。リンク時に、リンカは、アライメントがサイズ・オペランドに等しいと見なす。このサイズ・オペランドは、要求されたアライメントと異なる。

A3202 `alignment is greater than 64, the section alignment is restricted to 64`

COFF32 出力ファイル・フォーマットでは、セクションのアライメント要求が 64 バイトを超えてはならない。一部のリンカは、64 バイトより大きい境界にセクションのアライメントを合わせることができない。実際のアライメントは、リンカのポリシーによって決まる。

A3203 `symbol ***symbol*** aliased to ***name*** does not appear in the object file symbol table`

このシンボルは未定義である。したがって、.alias ディレクティブは無効になる。この問題を解決するには、.global、.local、または .weak ディレクティブを使用して、シンボルを定義する必要がある。

A3204 `integer number does not fit in ***number*** bits: ***number***`

この命令には、この数値は大きすぎる。この数値は、おそらく内部計算の結果である。

A3205 `invalid operand immediate value: ***value***`

この命令オペランドには、この即値は無効である。詳細については、『*Intel® Itanium™ アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル*』を参照のこと。

このエラー・メッセージの原因となるコードの例を次に示す。

```
fetchadd4.acq r3 = [r4], 7
```

A3300 `.lcomm/.common directive for defined symbol ***symbol*** is ignored`

.lcomm または .common ディレクティブを使用してシンボルを定義するときは、相対アドレス定義を使用すること。ローカル・シンボルまたはグローバル・シンボルの定義には、特定の位置を使用できる。しかし、.lcomm または .common ディレクティブと特定の位置の定義を組み合わせると、その位置は無視される。

特定の位置の定義の例を次に示す。

```
L:
.size L,16
```

相対位置の定義の例を次に示す。

```
.lcomm L,16,n
.lcomm L,4,n
```

この例では、リンカは、大きい方のサイズの定義を選択する。

A3301 `.common directive for symbol ***symbol*** overrides the local common declaration`

このシンボルは、同時にローカル共有シンボル (.lcomm) と共有シンボル (.common) として定義されている。この場合は、.common ディレクティブの定義が使用される。

A3302 `size setting for undefined symbol: ***symbol***`

このシンボルは未定義である。したがって、.size ディレクティブは無効になる。この問題を解決するには、.global、.local、または .weak ディレクティブを使用して、シンボルを定義する必要がある。

A3303 `dangerous use of a symbolic address [can exceed ***number*** bits]`

このシンボリック・アドレスは、この命令の許容範囲を超えている。次の方法で、より安全にシンボリック・アドレスをロードすることができる。

- `movl` 命令を使用する。例えば、次のようにする。

  ```
  movl r2 = <address>
  ```

- メモリ・テーブルからの間接的なロードを使用する。例えば、次のようにする。

  ```
  add r3 = @gprel(symbol), gp
  ld8 r4 = [r3]
  ```

A3304 `Reference to current location in assignment directive may be incorrectly resolved when it appears within open bundle`

バンドルを暗黙的に作成する場合、閉じられていないバンドル内に代入ディレクティブが置かれていると、誤った値が戻される。

この警告の原因となるコードの例を次に示す。

```
nop 5
A = $ + 5
```

A3305 `Bundle was closed to resolve current location reference`

この警告は、暗黙的バンドル・モードで、「現在の位置」を示す特殊記号 ("$" または ".") が文中で参照されている場合に生成される。この場合、IAS は、あいまいさを解決するためにバンドルを閉じる。

この警告の原因となるコードの例を次に示す。

```
L::
nop.i 0
.size L, $ - L
```

このコードを修正するには、「現在の位置」を示す特殊記号の直前に、ラベルまたは一時的ラベルを置く必要がある。

A3306 `label is undefined ***label_name***`

IAS 注釈内で参照されるラベルが未定義である。IAS はこの注釈を無視する。

A3307 `label is not defined in the current section` ***`label_name`***

IAS 注釈内のラベルは、他のセクションで定義されている。IAS はこの注釈を無視する。

このコードを修正するには、構文エラーがないかどうか確認し、適切なラベルを持つセクションに注釈を移動する。

A3308 `annotation is ignored`

IAS は、このタイプのディレクティブまたはオペランドの組み合わせをサポートしていない。

A3309 `branch target is specified for non-branch instruction`

`.br.target` ディレクティブは、分岐命令の前に置かなければならない。

A3310 `branch target is not specified for branch instruction`

`.br.target` ディレクティブが、間接的な分岐命令の前に置かれていない。

A3311 `vral directive is ignored. Use -X vral flag`

仮想レジスタ割り当てディレクティブを使用するためには、コマンド・ラインで `-X vral` を指定しなければならない。

A3312 `explicit usage of allocatable register register`

このレジスタを明示的に使用すると、IAS は、割り当て可能なレジスタの集合からこのレジスタを削除する。

A3313 `This predicate relationship is currently ignored`

このオペランドの組み合わせに対して、ディレクティブ `.pred.rel` を使用することはできない。

A3315 `Code is present in the non-executable section` ***`sec_name`***

非実行セクションのコードは実行されない。

A3316 `Directive ***unwind directive*** is ignored in the unwind genertion mode`

`-X unwind` コマンドライン・オプションを使用しているとき、アンワインド・ディレクティブは無視される。

A3401 `.plabel directive is obsolete. This directive is ignored`

`.plabel` ディレクティブは現在では使用できない。この警告の原因となるコードの例を次に示す。

```
.proc foo
foo::
.plabel foo
.endp
```

このコードを修正するには、関数シンボルを次のように宣言するか、

```
.proc foo
```

または、次のように入力する。

```
foo, @function
```

A3403 `virtual register has never been defined`

`.vreq.undef` 内で指定された仮想レジスタを定義する必要がある。

A3410 `***Unwind generator message*** in procedure ***procedure***`

スタティック分析は完了しない。IAS には間接分岐についての追加の注釈が必要である場合がある。この警告は、プロシージャ・コードが Itanium アーキテクチャ・ソフトウェアの表記規則と互換性がない場合にも発行される。

戻り値 B

Intel® Itanium™ アセンブラ (IAS) は、実行の終了時に、終了の理由を示す値を戻す。戻り値は次のとおりである。

0	IAS の実行は完了した。
2	IAS は、他の値が対象としない一般的なエラーによって終了した。
5	IAS は内部エラーによって終了した。
10	IAS は致命的エラーによって終了した。致命的エラーは、付録 A の「致命的エラー・メッセージ」の項に記載されている。
11	IAS は主入力ファイルを開けなかった。
12	IAS は、プログラムに組み込まれたファイルのうち 1 つを開けなかった。
13	IAS は、要求されたファイルを開けなかった。
15	IAS の実行中のエラー数が上限値に達した。
20	IAS は、誤ったコマンド・ライン構文のために実行されなかった。
25	IAS はメモリの障害のために終了した。

仕様 C

本章では、IAS の仕様を示す。

文字列の長さ	最大 1024 ビット
シンボル名の長さ	最大 4KB
アライメント要求	最大 4GB
整数計算	最大 128 ビット、符号付き
インクルード・ファイルのサイズ	システムによって異なる
行の長さ	システムによって異なる

プレディケート解析 D

この付録では、IAS でのプレディケート解析の実行方法について説明する。

依存関係違反とアセンブリ・モードについては、第 4 章の「依存関係違反とアセンブリ・モード」を参照のこと。

D.1 mutex 関係

相互に排他的な (mutex) 関係とは、プレディケートのグループで 2 つ以上のプレディケートが同時に真であることはないことを示す。

次の例では、プレディケート `p1`、`p2`、および `p3` が mutex（いずれか一つしか有効でない）である場合、「書き込みの後の書き込み」(write-after-write) 依存関係違反はない。これらの命令のいずれか 1 つだけが実際に実行されるからである。

```
(p1) mov r4 = 2

(p2) mov r4 = 5

(p3) mov r4 = 7
```

IAS は、次の場合に mutex 関係を作成する。

- プレディケート付きではない通常の比較命令
  次のコードでは、プレディケート修飾 (qp) が `p0` のときだけ、プレディケート `p1` および `p2` は mutex となる。
  ```
  (qp) cmp.eq p1, p2 = r1, r2
  ```
  通常の比較命令には、2 つのプレディケートに書き込むすべての命令、`cmp`、`fcmp`、`tbit`、および `tnat` が含まれる。パラレル比較命令と条件なし比較命令はこれらのカテゴリには含まれない。
- 条件なし (unc) 比較命令
  次のコードでは、プレディケート修飾の値に関係なく、プレディケート `p1` および `p2` は mutex である。
  ```
  (p3) cmp.eq.unc p1, p2 = r1, r2
  ```
- 関係定義 “ `mutex` ”
  次のコードでは、ユーザの注釈 `pred.rel` によってプレディケート `p1`、`p2`、および `p3` の間の mutex 関係が設定される。
  ```
  .pred.rel “mutex“, p1, p2, p3
  ```

D.2 imply 関係

imply 関係は、2 つのプレディケートの間で定義される関係である。imply 関係とは、1 つのプレディケート・レジスタのステートがもう 1 つのプレディケート・レジスタのステートを含意することを意味する。

例えば、プレディケート p1 がもう 1 つのプレディケート p2 を含意する場合である。p1 が真のとき、p2 は常に真である。p1 が偽のとき、p2 は真または偽である。次のコードを参照のこと。

```
(p1) mov r4 = 2

(p2) br.cond L

     mov r4 = 7
```

p1 が p2 を含意する場合、「書き込みの後の書き込み」(write-after-write) 依存関係違反はない。p1 が真の場合、p2 もまた真であり、分岐は実行されるからである。p1 が偽の場合、最初の命令は実行されず、第 3 の命令が安全に実行される。

次の例では、p1 が p2 を含意する場合、「書き込みの後の書き込み」(write-after-write) 依存関係違反はない。

```
     mov r4 = 2

(p2) br.cond L

(p1) mov r4 = 7
```

IAS は、次の場合に imply 関係を作成する。

- 条件なし比較命令
  次の例では、p1 は p3 を含意し、p2 は p3 を含意する。p3 が偽のとき、p1 と p2 の両方とも偽に設定されるからである。言い換えると、p3 も真のときだけ p1 または p2 が真になる。
  ```
  (p3) cmp.eq.unc p1, p2 = r1, r2
  ```
- 関係定義 “ imply ”
  次のコードでは、ユーザの注釈 pred.rel によって imply 関係が設定される。プレディケート p1 はプレディケート p2 を含意する。
  ```
  .pred.rel “imply“, p1, p2
  ```

D.3 プレディケートの有効範囲

IAS はプレディケート関係をデータベースに入力し、そのデータベースは偽のレポートの確認に使用される。IAS は、次の場合にプレディケート関係をこのデータベースから削除する。

- プレディケート・レジスタへの書き込み

  次の命令のいずれか 1 つがこれらの関係に関連したプレディケートに書き込みを行うとき、プレディケート関係はデータベースから削除される。

  — 比較命令

  — マスクがプレディケートを示す場合の pr 命令への移動

  — pr-rot 命令への移動 (ローテート・プレディケートへの書き込みのみ)。マスク内のビット i が 0 の場合、Pi が imply のターゲットであるすべての imply 関係を削除する。マスク内のビット i が 1 の場合、Pi が imply のソースであり、すべての mutex 関係が Pi に関連しているすべての imply 関係を削除する。

  — br.ctop、br.cloop、br.wtop、および br.wtop がすべてのローテート・プレディケートに書き込むようなモジュロ・スケジューリング・ループ分岐命令

- ユーザの注釈

  次の例では、ユーザの注釈 pred.rel によってプレディケート関係が削除される。プレディケート p1、p2、および p3 に関するすべてのプレディケート関係がデータベースから削除される。

```
.pred.rel “clear“, p1, p2, p3
```

D.4 プレディケート関係の有効範囲の例外

有効範囲規則にはいくつかの例外がある。

- パラレル比較命令

  いくつかの同時に存在する条件があるとき、パラレル比較命令はプレディケート関係を維持し、強化する。

  デスティネーション・レジスタが imply 関係のターゲットであるとき、命令 cmp.rel.or は imply 関係を削除しない。次の例では、imply 関係は、p1 が p3 を含意し p2 が p3 を含意するような最初の比較命令の中で生成される。p3 はデスティネーション・レジスタなので、命令 cmp.eq.or はこれらの関係を削除しない。

```
(p3) cmp.eq p1,p2 = r1, r2 // p1 implies p3
     cmp.eq.or p3,p4 = r5, r6
(p1) mov r4 = 2
(p3) br.cond.sptk L// Imply still exists
     mov r4 = 7 // No write-after-write on r4
```

デスティネーション・レジスタが imply 関係のソースであるとき、命令 cmp.rel.and は mutex 関係と imply 関係を削除しない。次の例は、命令とパラレル比較を示している。mutex 関係はユーザの注釈の中で生成され (p1 と p2 はいずれか一方しか使えない)、命令 cmp.ne.and はこの関係を削除しない。

```
.pred.rel “mutex“,p1,p2
     cmp.ne.and p4,p1 = r5, r0    // Mutex still exists
(p1) mov r4 = 2
(p2) mov r4 = 5             // No write-after-write on r4
```

命令 cmp.rel.or.andcm p1,p2 = . . . は、同じプレディケート p1 および p2 の間の mutex 関係を再度作成し、p1 が imply 関係のソースであるとき imply 関係を削除せず、p2 が imply 関係のターゲットであるとき imply 関係を削除しない。

- 制御フロー・グラフなし

  IAS は制御フロー・グラフ (CFG) を構築しない。したがって、すべての既知の関係は、ラベルでも分岐ターゲットでも、エントリー・ポイントからハイパーブロックの範囲でデータベースから削除される。ただし、条件付き分岐をまたぐパス (フォールスルー) は、最初の命令の有効範囲にしたがって解析される。次の例では、まだ IAS はレジスタ r4 についての依存関係違反を見つけていないが、実行パスが L に分岐できるので、レジスタ r5 についての依存関係違反を報告する。この場合、IAS は p3 と p4 の間の新しい関係については未確認である。

  ```
  cmp.eq p1, p2 = r1, r2
  cmp.eq p3, p4 = r3, r0
  (p1) mov r4 = 2
  L:
  (p2) mov r4 = 5
  (p3) mov r5 = r7
  (p4) mov r5 = r8
  ```

  プレディケート関係がこれらの条件下でも維持されることがわかっている場合は、注釈を使ってアセンブラに通知する。

D.5 組み合わせの解析

IAS は、既知の関係の組み合わせに基づいて関係を推測することができる場合がある。

- imply 関係のチェーン

  p1 が p2 を含意し、p2 が p3 を含意し、したがって p1 が p3 を含意する場合。

- imply 関係および mutex 関係の組み合わせ

  p1 が p2 を含意し、p2 が p3 と mutex であり、したがって p1 が p3 と mutex である場合。

ただし、その他の場合には、IAS の複合関係の解析には限界がある。

- プレディケート付き比較命令

  次の例では、IAS は p2 と p3 を mutex として設定することはできない。最後の 2 つの比較命令がプレディケート付きで、関係がプレディケート付きではない通常の比較命令について作成されているからである。

```
     cmp.eq p1, p4 = r1, r2 ;;
(p1) cmp.ge p2, p3 = r1, r3
(p4) cmp.ge p2, p3 = r1, r4
```

- 条件解析

  IAS は比較命令の条件を解析しない。次の例では、IAS は p1、p2、および p3 を mutex として設定しない。

```
cmp.eq p1 = 0, r1
cmp.eq p2 = 1, r1
cmp.eq p3 = 2, r1
```

- CFG 解析

  IAS は CFG を計算せず、2 つ以上のパスによって生成された関係を探し出さない。これは、いずれのエントリー・ポイントでも、IAS はプレディケート関係に関しては初期の点から開始し、この初期の点ではすべてのプレディケートの間の関係は未知であることを意味する。

  次の例では、IAS はラベルの後 p1 と p2 を mutex として設定しない。

```
     cmp.eq p1, p2 = r1, r2 ;;
L:
(p1) mov r4 = 2
(p2) mov r4 = 5
     cmp.eq p1, p2 = r1, r2 ;;
     br.cond.sptk L ;;
```

  この規則の例外は、「制御フロー・グラフなし」で説明したようなフォールスルーの場合である。次に例を示す。

```
     cmp.eq p1,p2 = r1, r2 ;;
(p1) mov r4 = 2
(p3) br.cond.sptk L
(p2) mov r4 = 5
```

  この場合、r4 についての「書き込みの後の書き込み」(write-after-write) 依存関係違反はない。IAS は mutex 関係がまだ存在しているので、違反を報告しない。

用語集

絶対アドレス

絶対値として計算される、プロセスのアドレス空間内の仮想アドレス (物理アドレスではない)。

別名 (alias)

他の識別子と等価にされた識別子。

アプリケーション・レジスタ

各種の機能に使用される専用レジスタ。使用頻度の高いレジスタは、アセンブラ別名を持っている。例えば、エピローグ・カウンタとして使用されるレジスタ ar66 は、ar.ec とも呼ばれる。別名を参照。

アセンブラ

アセンブリ言語を機械語に変換するプログラム。

アセンブリ言語

マシンコード言語によく似たシンボリック言語。

ビッグ・エンディアン

最初にアドレス指定されるバイトに最上位バイトが格納される、数値の格納方法。

バインディング

あるモジュール内のシンボリック参照を解決するために、他のモジュール内でそのシンボルの定義を見つけて、シンボリック参照をその定義のアドレスで置き換えるプロセス。リンカは再配置可能なオブジェクト・モジュールをバインドし、DLL ローダは実行ロード・モジュールをバインドする。リンカと DLL ローダは、シンボルの定義を検索するとき、指定された順序で各モジュールを検索する。先に検索されるモジュール内のシンボルの定義は、それより後のモジュール内の同じシンボルの定義に優先する。この順序は、バインディング順序と呼ばれる。

バンドル

3 つの命令と 1 つのテンプレート・フィールドを含む 128 ビット。

COFF

共有オブジェクト・ファイル・フォーマット (Common Object File Format)。オブジェクト・モジュールのフォーマットの 1 つ。

データ要素

データ要素は、バイト、ワード、ダブルワード、またはクワッドワードである。MMX® テクノロジは、データ要素を、新たに定義されたパック・データ・タイプ (64 ビットにパックされた、8 バイト、4 ワード、または 2 ダブルワードのグループ) にパックする。

ディレクティブ

実行コードを生成しないアセンブリ言語文。

GB

ギガバイト。

グローバル・シンボル

シンボルが定義されているコンパイル・ユニットの外部で認識可能なシンボル。

IA-32

インテル・アーキテクチャ 32。インテルの 32 ビット命令セット・アーキテクチャ (ISA) の名前。

IA-32 システム環境

Pentium®およびPentium Proプロセッサについて定義されたシステム環境。

インデックス・レジスタ

汎用レジスタ `eax`、`ebc`、`ecx`、`edx`、`ebp`、`esp`、`esi`、および `edi` のうち任意のレジスタ。

命令

特定のマシン動作を実行するオペレーション・コード (オペコード)。

命令グループ

Itanium™ アーキテクチャ命令は、命令グループに編成されている。各命令グループには、並行して実行される、静的に隣接する 1 つ以上の命令が含まれる。命令グループには、少なくとも 1 つの命令が含まれていなければならない。命令グループ内の命令の数に上限はない。

命令グループは、ストップによって静的に終了し、分岐によって動的に終了する。

ストップは、二つのセミコロン (;;) で表現される。ストップは、ユーザが明示的に定義できる。ストップは、命令の直後に置くことも、別の行に置くこともできる。また、セミコロン (;) を使用して 2 つの命令を区切るように、ストップを同じ行上の 2 つの命令の間に挿入することもできる。

命令ポインタ (IP)

現在実行中の命令を含むバンドルのアドレスを保持する 64 ビット命令。IP は、命令が実行されるたびにインクリメントされる。分岐によって、IP を新しい値に設定することができる。

命令セット・アーキテクチャ

ユーザレベルの命令やユーザが認識できるレジスタ・ファイルなど、アプリケーション・レベルのリソースを定義するアーキテクチャ。

IP

命令ポインタを参照。

IP を基準とするアドレス指定

他のコードおよびデータをアドレス指定するためのベース・レジスタとして、自分のアドレスを使用するコード。このアドレスは、通常はプログラム・カウンタ (PC) と呼ばれ、Itanium アーキテクチャでは命令ポインタ (IP) とも呼ばれる。

ISA

命令セット・アーキテクチャを参照。

KB

キロバイト。

リトル・エンディアン

アドレスが最下位のバイトに最下位バイトが格納される、数値の格納方法。

ロード・モジュール

リンカによって生成される実行ユニット (メイン・プログラムまたは DLL)。プログラムは、少なくとも 1 つのメイン・プログラムで構成される。また、依存関係の条件を満たすために、1 つ以上の DLL を必要とすることもある。

MB

メガバイト。

nop

「ノー・オペレーション」命令は、プロセッサに対する実際の命令であるが、プロセッサは何も動作を実行しない。

OMF

オブジェクト・モジュール・フォーマット (Object Module Format)。オブジェクト・モジュールの内部構造と内容を示す。COFF は、OMF の 1 つの例である。

プレディケート・レジスタ

命令の実行を制御する、64 個の 1 ビット・プレディケート・レジスタ。最初のレジスタ `p0` の値は、常に `1` として処理される。

プレディケーション

命令の条件付き実行。これによって、コード内の分岐を削減できる。

特権命令セクション

1 つのユニットとしてバインドされた、オブジェクト・ファイルの部分 (コードやデータなど)。

共有シンボル

ダイナミック・リンカによって結合されたすべてのオブジェクト・ファイルとの間で、エクスポートやインポートが行えるシンボル。

文

アセンブリ言語プログラムは、一連の文で構成される。アセンブリ言語文には、以下の 5 つの主なタイプがある。

命令文
ラベル文
データ割り当て文
ディレクティブ文
代入文および等式文

ストップ

命令グループの終わりを示す。

シンボル宣言

必ずしも現在のモジュールに基づかずに、シンボルのアドレスが解決されるようにすること。シンボルを宣言するには、`.global` または `.weak` ディレクティブを使用する。

トークン

アセンブリ言語の最小の語彙要素。トークンは、一連の隣接する文字 1 つ以上で構成される。

索引