Itanium™ プロセッサ・マイクロアーキテクチャ・リファレンス

ソフトウェアの最適化のために

2000 年 8 月

資料番号 : 245473J-002

【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: 0120-47-8832

Itanium™ 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、インテルまでお問い合わせください。

目次

1.0 概要

Itanium™ プロセッサは、IA-64 アーキテクチャを実装した最初のプロセッサである。本書では、パフォーマンス・チューニング、コンパイル、およびアセンブリ言語プログラミングに関連する、Itanium プロセッサの IA-64 アーキテクチャの特徴について説明する。特記のない限り、本章で説明するすべての制限、規則、サイズ、および容量は、Itanium プロセッサにのみ適用され、他の IA-64 プロセッサには適用されない。

本書は、IA-64 プロセッサの動作について基本的な知識を持ち、IA-64 命令について明確に理解している読者を対象としている (IA-64 についての詳細は、『*Intel® IA-64 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*』第 1 巻 (資料番号 245317J)、第 2 巻 (資料番号 245318J)、第 3 巻 (資料番号 245319J)、第 4 巻 (資料番号 245320J) を参照のこと)。

本書では、ソフトウェアの観点から見た、Itanium プロセッサに関する広範囲にわたる項目について説明する。詳細な説明の部分は、特に高性能を必要とするプログラマやコンパイラ作成者を対象にしている。各項目の説明の量は、必ずしもその項目の重要性に対応していない。例えば、分岐予測、コントロール・スペキュレーション、データ・スペキュレーション、機能ユニットのリソース、およびレイテンシは、通常は分岐ヒントの使用より大きな影響をパフォーマンスに与えるが、本書では分岐ヒントの説明の方により多くのスペースが割かれている。

本書は、Itanium プロセッサ・マイクロアーキテクチャの動作を簡潔に説明し、その中でできるだけ正確に情報を記述している。本書の目的は、広範囲にわたるアプリケーションのパフォーマンス・チューニングのための情報を提供であり、すべての動作を説明することは本書の記述範囲を超えている。したがって、一部の項目 (特に、仮想メモリ、制御レジスタ、分岐とプリフェッチ、およびメモリ・パイプラインの深いレベルに関する内容) は概略の説明になっている。

2.0 機能ユニットと発行規則

この節では、利用可能な機能ユニットの数とタイプ、不要な実行ストールを避けるための規則、および命令の配置と発行について説明する。

2.1 実行モデル

Itanium プロセッサは、ソフトウェアで指定された順序で命令を発行し実行する。したがって、効率的なアセンブリ・コードを生成するためには、コンパイラがストールの条件を理解する必要がある。

一般的に、ある命令がその直前の命令と同時に発行されない場合、このような命令の実行を*分割発行*と呼ぶ。分割発行状態が発生すると、分割点より後のすべての命令は (命令を実行するための十分なリソースがあっ

ても)1 クロック以上ストールする。分割発行は、このような種類のストールのことである。ストールの原因には、スコアボーディング、命令のアライメント、機能ユニットの衝突、その他のパイプラインに関連するストールなどがある。

Itanium プロセッサ上の分割発行の一般的な原因には、以下のものがある。

- 命令を実行するためのリソースが不足している。
- 命令のソース・レジスタがまだ使用できない (データを生成する側の命令が完了していない)。
- ストップが検出された。
- 命令が Itanium プロセッサの発行規則に従って配置されていない。
- 命令グループのアライメントの規則が守られていない。

2.2 機能ユニット・クラスの定義

以下の表は、IA-64 命令を命令クラスごとに分類したものである。これらのクラスは、命令のレイテンシ、バイパス、および機能ユニットの機能の定義に使用される。ここに示したクラス名は、Itanium プロセッサ上のレイテンシとリソースについて説明するための表記であり、IA-64 のアーキテクチャ仕様の一部ではない。

機能ユニット・クラス名	命令のリスト
BR	br.call, br.cond, br.ia, brl.call, brl.cond, br.ret, br.wexit
BR_B2	br.cexit, br.cloop, br.ctop, br.wexit, br.wtop
BRP	brp,brp.ret
CHK_ALAT	chk.a.clr, chk.a.nc
CHK_I	chk.s.i
CHK_M	chk.s.m
CLD	ld.c
FCLD	ldf8.c, ldfd.c, ldfe.c,l dfp8.c, ldfpd.c, ldfps.c, ldfs.c
FCMP	fclass.m, fcmp
FCVTFX	fcvt.fx, fcvt.fxu, fcvt.xf
FLD	ldf8, ldf8.a, ldf8.s, ldf8.sa, ldfd, ldfd.a, ldfd.s, ldfd.sa, ldfe, ldfe.a, ldfe.s, ldfe.sa, ldf.fill, ldfs, ldfs.a, ldfs.s, ldfs.sa
FLDP	ldfp8, ldfp8.a, ldfp8.s, ldfp8.sa, ldfpd, ldfpd.a, ldfpd.s, ldfpd.sa, ldfps, ldfps.a, ldfps.s, ldfps.sa
FMAC	fma, fnma
FMISC	famax, famin, fand, fandcm, fmax, fmerge.ns, fmerge.s, fmerge.se, fmin, fmix, for, fpack, frcpa, frsqrta, fselect, fswap, fsxt, fxor
FOTHER	fchkf, fclrf, fsetc
FRAR_I	mov.i =ar
FRAR_M	mov.m = ar

機能ユニット・クラス名	命令のリスト
FRBR	mov =br
FRCR	mov =cr
FRFR	getf
FRIP	mov =ip
FRPR	mov =pr
IALU	add, addl, adds, shladd, sub
ICMP	cmp4,cmp
ILOG	and, andcm, or, xor
ISHF	dep, dep.z, extr, shrp
LD	ld, ld.a, ld.s, ld.sa, ld.bias
LFETCH	lfetch
LONG_I	movl
MMALU_A	padd, padd4, pavg1, pavg2, pavgsub, pcmp, pshladd2, pshradd2, psub
MMALU_I	pmax, pmin, psad1
MMMUL	pmpy2, pmpyshr2, popcnt
MMSHF	mix, mux, pack, pshl, pshr, shl, shr, unpack
NOP_B	break.b, nop.b
NOP_I	break.i, nop.i
NOP_M	break.m, nop.m
NOP_F	break.f, nop.f
NOP_X	break.x, nop.x
PNT	addp4, shladdp4
RSE_B	clrrrb, cover
RSE_M	flushrs, loadrs
SEM	cmpxchg, fetchadd, xchg
SFCVTFX	fpcvt.fx, fpcvt.fxu
SFMAC	fpma, fpms, fpnma
SFMERGESE	fpmerge.se
SFMISC	fpamax, fpamin, fpcmp, fpmax, fpmerge.ns, fpmerge.s, fpmin, fprcpa, fprsqrta
STF	stf8, stfd, stfe, stfs, stf.spill
ST	st, st8.spill
SYST_B2	bsw,rfi
SYST_B	epc
SYST_M0	alloc, cc, fc, halt, itc.d, itc.i, itr.d, itr.i, mf.a, probe, ptc.e, ptc.g, ptc.ga, ptc.l, ptr.d, ptr.i, rsm, rum, ssm, sum, tak, thash, tpa, ttag, mov psr=, mov =psr, mov rr=, mov =rr, mov pkr=, mov =pkr, mov pmd=, mov =pmd, mov pmc=, mov =pmc, mov msr=, mov =msr, mov ibr=, mov =ibr, mov dbr=, mov =dbr, mov =cpuid
SYST_M	fwb, invala, invala.e, mf, srlz.d, srlz.i, sync.i
TBIT	tbit

機能ユニット・クラス名	命令のリスト
TOAR_I	mov.i ar=
TOAR_M	mov.m ar=
TOBR	mov br=
TOCR	mov cr=
TOFR	setf
TOPR	mov pr=
XMA	xma, xmpy
XTD	czx, sxt, zxt

2.3 機能ユニットの数とタイプ

IA-64 命令グループを構成する命令バンドルの数と各命令タイプの命令数に制限はないが、Itanium プロセッサの実行リソースは有限である。命令グループに、そのタイプの命令用の実行ユニットの数より多くの命令が含まれる場合は、適切なユニットが見つからない最初の命令で、分割発行が発生する。

実行ユニットは、通常はスロットのタイプ (I、F、M、または B) で分類される。A タイプの命令は、IA-64 では、M タイプまたは I タイプのユニットで実行するようにスケジューリングできる。ただし、Itanium プロセッサの実行ユニットは、そのユニットが実行できる一連の命令に対して、一般に非対称である。例えば、2 つの I ユニット (I0 と I1) があるが、I0 だけが `tbit` 命令を実行できる。

すべての計算機能ユニットはフルにパイプライン化されているため、各機能ユニットは、他のタイプのストールがない場合、1 クロック・サイクルごとに 1 つの新しい命令を受け入れられる。システム命令とレジスタへのアクセスについては若干の例外があるが、これらの問題は本書の記述範囲を超えている。

以下の各項では、それぞれの機能ユニットと、そのユニットが実行できる命令のタイプ、数、およびクラスについて説明する。

2.3.1 整数

整数 ALU は、I タイプと A タイプの命令を実行する。Itanium プロセッサは、2 つの整数 (I) ユニットを持つ。各ユニットの機能は次のとおりである。

- I0 は、すべての I タイプおよび A タイプの命令を実行できる。
- I1 は、SYS_I0、FRIP、FRBR、TOBR、MMMUL、TBIT、ISHF、TOPR、FRPR、TOAR_I、FRAR_I を除く、すべての I タイプおよび A タイプの命令クラスを実行できる。

2.3.2 メモリ

メモリ・ユニットは、M タイプおよび A タイプの命令を実行する。Itanium プロセッサは、2 つのメモリ (M) ユニットを持つ。各ユニットの機能は次のとおりである。

- M0 は、すべての M タイプおよび A タイプの命令を実行できる。
- M1 は、SEM、FRFR、SYST_M0、RSE_M、TOAR_M、FRAR_M、TOCR、FRCR を除く、すべての M タイプおよび A タイプの命令クラスを実行できる。

2.3.3 分岐

分岐ユニットは、B タイプの命令を実行する。Itanium プロセッサは、3 つの分岐 (B) ユニットを持つ。各ユニットの機能は次のとおりである。

- B0 と B2 は、すべての B タイプの命令を実行できる。ただし、B2 は SYST_B0 を実行できない。
- ＸB1 は、SYST_B2 以外のすべての B タイプの命令を実行できる。ただし、B1 に送られた `brp` 命令は無視される。

2.3.4 浮動小数点

浮動小数点実行ユニットは、F タイプの命令を実行する。Itanium プロセッサは、2 つの浮動小数点 (F) ユニットを持つ。各ユニットの機能は次のとおりである。

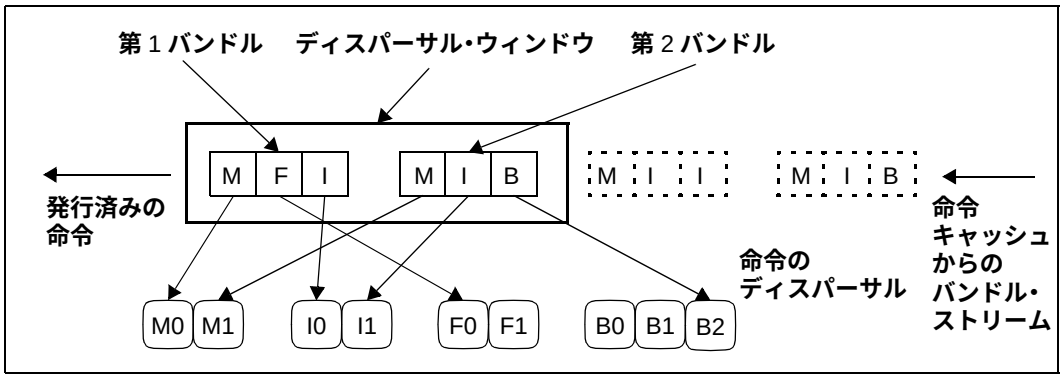
- F0 は、すべての F タイプの命令を実行できる。
- F1 は、FMISC、SFMISC、FCMP、SFMERGESE を除く、すべての F タイプの命令クラスを実行できる。

2.4 命令スロットと機能ユニットの対応関係

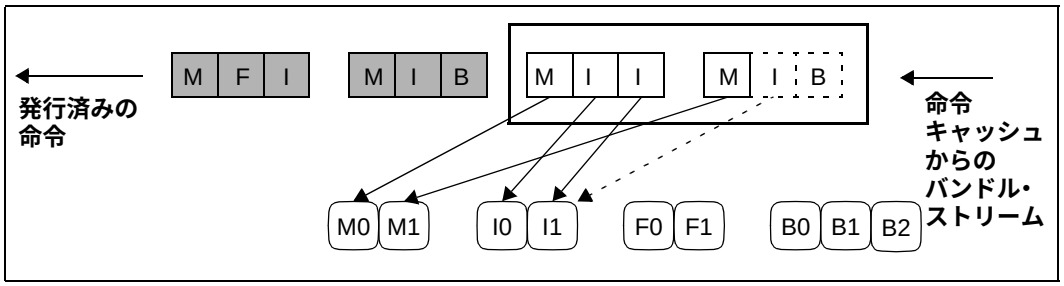
ある命令が Itanium プロセッサのどの機能ユニットに送られるかは、その命令の命令スロットのタイプと、現在発行されている一連の命令の中の位置によって決まる。命令を機能ユニットに送り出すプロセスを、ディスパーサルと呼ぶ。Itanium プロセッサのハードウェアは、命令の順序を変更してストールを避けたりしない。したがって、コンパイラは、命令グループ内の命令の数、タイプ、および順序に注意して、不要なストールを避けなければならない。命令にプレディケートが使用されていても、ディスパーサルには影響を与えない。プレディケートが真、プレディケートが偽、プレディケートなしのいずれの場合も、すべての命令は同じ方法で発行される。同様に、nop も正常な命令と同じように機能ユニットに発行される。

Itanium プロセッサは、命令のディスパーサル用の機能ユニットを決定する際に、一度に最大 2 バンドルまで参照できる。本書では、これらのバ

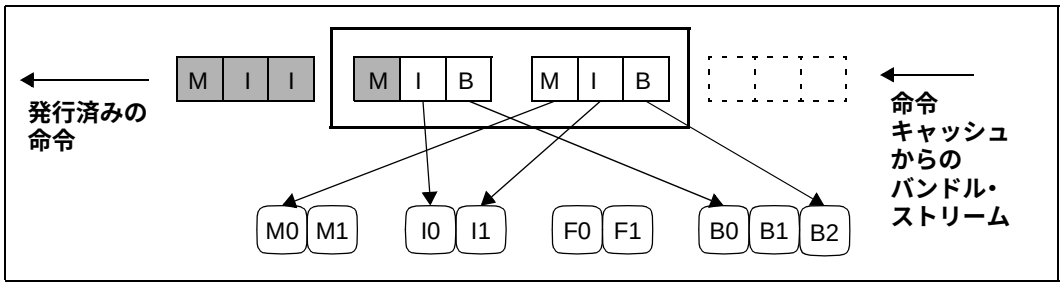
ンドルを*第1*バンドルと*第2*バンドルと呼ぶ。*バンドル*がローテーションされると、発行の対象となる命令の2バンドル幅のウィンドウに、新しいバンドルが送り込まれる。



上図に示すように、第1バンドルと第2バンドルのすべての命令が発行される場合は、2つのバンドルのローテーションが発生する。下図に示すように、このローテーションにより、新しい第1バンドルと第2バンドルがウィンドウに送り込まれる。



上図の状態では、第1バンドルのすべての命令が発行されているが、第2バンドルの命令には発行されていないものもある。最後の2つの命令が発行されなかったのは、第2バンドルの第2スロット用のIユニットが残っていなかったからである。この結果、下図に示すように、1つのバンドルのローテーションが実行される。



第 1 バンドルの命令は実行パイプライン内で続けて処理され、第 2 バンドルが第 1 バンドルになり、新しい第 2 バンドルがウィンドウ内に送り込まれている。バンドル・ローテーションに対する分割発行の影響は、次の規則によって示される。

- 第 1 バンドルに分割発行状態がある場合は、バンドル・ローテーションは行われない。
- 第 1 バンドルに分割発行状態がなく、第 2 バンドルに分割発行状態がある場合は、1 つのバンドルのローテーションが行われる。
- ディスパーサル・ウィンドウ内に分割発行がない場合は、2 つのバンドルのローテーションが行われる。

ディスパーサル規則によって、該当する命令を実行するリソースを持たない機能ユニットに対して命令が発行される場合 (例えば、I1 に対して `tbit` が発行される場合) や、該当する命令を実行できるすべての機能ユニットがそれ以前の命令によって使用されている場合は、リソースの衝突が発生し、その命令の前に発行が分割される。

ディスパーサル・ウィンドウ内の命令の一部が発行されない場合は、バンドル・ローテーション規則の適用後の次のサイクルで、ディスパーサル規則がもう一度適用される。この方法により、分割発行の後は、グループ内の先行する命令は発行済みとなり、発行リソースに対して後続する命令と競合しなくなるため、処理の前方への進行が保証される。

2.4.1 実行幅

Itanium プロセッサは、ディスパーサル時に一度に 2 つのバンドルを参照するため、1 クロック当たり最大 6 つの命令スロットを発行できる。この 6 つのスロットの集合は、スロット内の命令やプレディケートの値に関係なく、最大で 2 つの I スロット、2 つの M スロット、2 つの F スロット、および 3 つの B スロットを含むことができる。

2.4.2 ディスパーサル規則

実行ユニットのディスパーサル規則は、以下に示すように、スロットのタイプによって異なる。

M/I スロットの規則	第 2 バンドルの 3 番目の位置の I スロットは、常に I1 に発行される。 それ以外の場合、M 命令または I 命令は、未使用の一番小さい番号の M ユニットまたは I ユニットに発行される。
F スロットの規則	第 1 バンドル内の F スロットは、F0 に発行される。 第 2 バンドル内の F スロットは、F1 に発行される。
B スロットの規則	MBB または BBB バンドル内の各 B スロットは、それに対応する B ユニットに発行される。すな

わち、テンプレートの最初の位置の B スロットは B0、2 番目の位置の B スロットは B1、3 番目の位置の B スロットは B2 に発行される。

MIB/MFB/MMB バンドル内の B スロットは、`brp` または `nop.b` 命令の場合は B0 に発行され、それ以外の場合は B2 に発行される。

L スロットの規則　MLX バンドルは、MFI バンドルと同じポートを使用する。

2.4.3 命令のディスパーサルの例

Itanium プロセッサは定数幅の整数シフタを 1 つ持つ。このため、以下のシーケンスでは、`add` の後に分割発行が発生し、バンドルのローテーションは行われない。`extr` および第 2 バンドルのすべての命令は次のサイクルで発行され、2 つのバンドルのローテーションが行われる。

```
{ .mii
    ld4  r1=[r5] // Maps to M0, first cycle
    add  r2=r5,r6// Maps to I0, first cycle
    extr r3=r8,5,3// Stall - extr on I0 only
          // extr Issues in second cycle to I0.
}
{ .mbb
    ld4    r20=[r22]
(p2)br.cond L1
(p1)br.cond L2
}
```

しかし、コンパイラが命令の順序を変更して `extr` を先に置けば、6 つの命令がすべて 1 クロック内で発行され、次に 2 つのバンドルのローテーションが行われる。

```
{ .mii
    ld4    r1=[r5]   // Maps to M0
    extr   r3=r8,r5,3// Maps to I0
    add    r2=r5,r6  // Maps to I1
}
{ .mbb
    ld4    r20=[r22] // Maps to M1
    br.cond L1       // Maps to B1
(p1)br.cond L2       // Maps to B2
}
```

2.4.4 Itanium™ プロセッサ上の分割発行とバンドル・タイプ

特定のペア以外のバンドル・テンプレートを組み合わせて使用すると、分割発行状態が発生する。これは、nop が入っている命令スロットを含めて、すべてのスロットが機能ユニットに対して発行されるためである。

例えば、スロット内の命令が何であるかに関係なく、バンドルのペアを次のように続けて使用すると、機能ユニットの衝突のために、分割発行が発生する。

MMI **M**MI	M0 と M1 の両方がビジーであるため、3 番目の M スロットの前に発行が分割される。
MII M**I**I	I0 と I1 の両方がビジーであるため、3 番目の I スロットの前に発行が分割される。
MMI **M**II	M0 と M1 がビジーであるため、3 番目の M スロットの前に発行が分割される。
MII MF**I**	I0 と I1 の両方がビジーであるため、3 番目の I スロットの前に発行が分割される。

分割発行の原因にはリソースの衝突以外に、Itanium プロセッサ固有の特殊な条件がいくつかある。これらの条件は、スロットのタイプと命令に基づくものである。

MMF	前後のバンドルやストップに関係なく、最初の M の前と F の後に、必ず発行が分割される。
BBB/MBB	これらのバンドルの後、必ず発行が分割される。
MIB/MFB/MMB	B スロットに `nop.b` または `brp` 命令が入っている場合を除いて、これらのバンドルの後、発行が分割される。
MIB BBB	最初のバンドルの後、発行が分割される。

ただし、ここで説明したすべての分割発行状態は、プロセッサに依存する。また、有効な IA-64 コードを生成するには、明示的なストップを使用して命令グループを区切らなければならない。

これ以外に、いくつかの特殊な分割発行状態がある。Itanium プロセッサは、`mf.a`、`halt.mf`、`inval`、`invala.e` の後と、クラス SEM の任意の命令の後に、命令の発行を分割する。

2.5 命令グループのアライメント

命令グループが 1 次命令キャッシュ内のキャッシュ・ライン境界を超えた場合、その境界上で命令グループの発行を分割するかどうかを決定する複雑な規則が存在する。一般的な規則として、このようなタイプのストールは、基本ブロックの始めで発生する傾向がある。したがって、コンパイラは主に (ブロック内の最初の命令グループがキャッシュ・ラインを超える) 基本ブロックの始点で、命令グループのアライメントに注意する必要がある。

もう 1 つのアライメントの問題は、ディスパーサル・ウィンドウに関連する。Itanium プロセッサのディスパーサル・ウィンドウは、幅がちょうど 2 バンドルで、バンドル単位で移動する。したがって、命令グループが 3 バンドル以上にわたる場合、リソースの衝突がなくても、第 2 バンドルの終わりで必ず発行が分割される。

以下の例に、3 バンドル以上にわたる命令グループを示す。各命令の右側に、実効発行時間を示す。

```
L:
{ .mii
    add     r1=r2,r3 // 0
    add     r4=r5,r6 ;;// 0
    sub     r7=r8,r9 // 1
}
{ .mfi
    ld4     r14=[r56]// 1
    fadd    f10=f12,f13// 1
    add     r16=r18,r19 // 1: Split issue occurs
                        //    after this instr
}
{ .mmi
    st4     [r16]=r67 ;; // 2
    add     r24=r56,r57  // 3
    add     r28=r58,r59  // 3
}
```

このシーケンスでは、Itanium プロセッサは、sub 命令で始まり st4 命令で終わる命令グループを 1 サイクルで実行するのに十分な機能ユニットを持っている。しかし、命令グループがディスパーサル・ウィンドウの幅を超えているため、第 2 バンドルの終わりで命令の発行が分割される。

3.0 Itanium™プロセッサのレイテンシとバイパス

クロック *i* でスケジューリングされた命令 *I* は、レイテンシによるストールを避けるために命令 *J* をサイクル *i+N* 以降にスケジューリングしなければならない場合、命令 *J* に対して *N* サイクルの合計レイテンシを持つことになる。命令のペアの合計レイテンシは、次の 2 つの要素によって決まる。

- 結果を生成する側の命令のレイテンシ
- 結果に依存する操作に対して、この結果をバイパスする時間

次の節では、一般的な場合について合計レイテンシを示す。3.4 節では、これらのレイテンシの重要な例外について説明する。非常に少数であるが、以下の表に含まれていない、変則的な場合も存在する。

Itanium プロセッサ上では、オペランドの準備ができていない (オペランドが書き込み中であるか、まだ書き込みが完了していない) 命令は、オペランドの準備ができるまでストールする。例えば、次のコードの例を考える。

```
    ld4     r1=[r5] ;;
    add     r2=r1,r6
```

add 命令は、ロードの結果の書き込みが完了するまで (Itanium プロセッサ上では 2 サイクル以上) ストールする。同様に、別々の命令グループの 2 つの命令が同じレジスタに書き込もうとした場合 (これらの命令が同じ命令グループに含まれていたとすれば、このコード・シーケンスはアーキテクチャ上未定義になる)、第 2 の命令は第 1 の命令が書き込みを完了するまで待機する。

```
ld4    r1=[r5] ;;
add    r1=r2,r6
```

したがって、第 2 の命令が問題のレジスタの読み出しまたは書き込みを行う場合、第 2 の命令は第 1 の命令が結果の書き込みを完了するまで待って処理を実行する。上の例では、ld4 がキャッシュ・ミスになった場合、add 命令はメモリから値が返されるまで待ってから処理を実行する必要がある。

3.1 重要な機能ユニットの合計レイテンシ

以下の表に、各種のクラスの操作のレイテンシを示す。ただし、これらの規則には多くの例外がある。命令のレイテンシの表の中には、非対称な場合もあるため、結果を生成する側の命令のタイプ、その命令が実行される機能ユニット、結果を参照する側の命令のタイプを考慮に入れる必要がある。ここに示したクラス名は、Itanium プロセッサ上のレイテンシについて説明するための表記であり、IA-64 のアーキテクチャ仕様の一部ではない。

一般的に、結果を生成する側の命令と参照する側の命令がいずれも同じタイプのユニット上で実行される場合は、これらのレイテンシは適切である。機能ユニットのタイプには、マルチメディア (MMxxx クラス)、浮動小数点 (Fxxx)、並列浮動小数点 (SFxxx)、整数 (IALU、ILOG、ICMP、LD、ST) がある。

ソース命令クラス	説明	レイテンシ (サイクル)
FCMP	浮動小数点の比較 (分岐に対して)	1
	浮動小数点の比較 (非分岐命令に対して)	2
FCVTFX	固定小数点への変換	7
FMAC	浮動小数点算術演算	5
FMISC	浮動小数点の min、max、frcpa...	5
FRAR_M, FRAR_I	mov=ar.xx (M/I スロット命令)	6.0 節を参照
FRCR	mov=cr (レジスタによって異なる)	6.0 節を参照
FRFR	FP レジスタから GP レジスタへのコピー	2
FRIP	mov =ip	2
FRPR	mov =pr	2
IALU	整数 ALU	1

ソース命令クラス	説明	レイテンシ（サイクル）
ICMP, TBIT	整数の比較 (結果に依存する分岐に対して)	0
	整数の比較	1
ILOG	論理演算	1
ISHF	dep, extr, shrp	1
MMALU_A	A タイプのマルチメディア命令	2
LONG	long 型の mov	1
MMALU_I	I タイプのマルチメディア命令	2
MMMUL	並列乗算	2
MMSHF	shl, shr, unpack, pshl, pshr,. . . .	2
PNT	ポインタの add/shladd	1
SEM	セマフォの操作	6.0 節を参照
SFCVTFX	SIMD fcvtfx	7
SFMAC	SIMD FMACs	5
SFMERGESE	SIMD fmerge.se	7
SFMISC	その他の SIMD FP	5
SYST	(レイテンシはレジスタによって異なる)	6.0 節を参照
TOAR_I, TOAR_M	I/M タイプの mov ar.xx= (レイテンシはレジスタによって異なる)	6.0 節を参照
TOCR	mov=cr (レイテンシはレジスタによって異なる)	6.0 節を参照
TOFR	GP レジスタから FP レジスタへのコピー	9
TOPR	mov pr=	1
XMA	FP 整数乗算 (他の XMA に対して)	7
XTD	sxt, zxt, czx	1

3.2 メモリ・システムのレイテンシとペナルティ

以下の表に、メモリ操作のレイテンシとメモリに関連するフラッシュのレイテンシを示す。

ソース命令クラスまたはイベント	説明	レイテンシ（サイクル）
CHK_I, CHK_M, CHK_ALAT	chk.a (ALAT ヒット)、chk.s (NaT/NaTVal なし)	0
	chk.a (ALAT ミス)、chk.s (NaT/NaTVal)	50+
CLD, FCLD	ld*.c (ALAT ヒット、L1/L2 ヒット)	0
	ld*.c (ALAT ミス、L1/L2 ヒット)	10
FLD, FLDP	FP ロード (L2 ヒット)	9
	FP ロード (L3 ヒット)	24

ソース命令クラスまたはイベント	説明	レイテンシ（サイクル）
LD	ld.c 以外の整数ロード (L1 ヒット)	2
	ld.c 以外の整数ロード (L2 ヒット)	6
	ld.c 以外の整数ロード (L3 ヒット)	21
DTC Miss	DTC ミスのフラッシュによるパイプライン内のバブル・サイクル数	10

3.3 分岐に関連するレイテンシとペナルティ

以下の表に、分岐操作のレイテンシと分岐に関連するフラッシュのレイテンシを示す。

ソース命令クラスまたはイベント	説明	レイテンシ（サイクル）
FRBR	mov =br	2
TOBR	mov br=	非分岐に対して 1 BR に対して 0
予測ミスになった分岐のペナルティ	分岐が実行された時間から次の命令が開始される時間までのデッド・サイクル	9
実行される分岐のバブル	実行されるものと正しく予測された分岐上に挿入されたフロント・エンド・バブルの数	0/1/2/3

3.4 合計レイテンシの例外のまとめ

あるタイプのユニット上で計算された結果が他のタイプのユニット上で参照される場合や、(より一般的には)Itanium プロセッサによって高速バイパスが提供されない場合は、通常はバイパス・レイテンシの追加が発生する。以下の表に、Itanium プロセッサの特殊な合計レイテンシを示す。

ソース命令クラス	ターゲット命令クラス	合計レイテンシ
IALU (I スロット命令の場合のみ)	LD/ST アドレス・レジスタ	2
ILOG, PNT, XTD	LD/ST アドレス・レジスタ	2
LD	LD/ST アドレス・レジスタ	3 (L1 ヒット)
IALU, ILOG	MMMUL, MMSHF, MMALU	3
LD	MM 操作	3 (L1 hit)3 (L1 ヒット)

ソース命令クラス	ターゲット命令クラス	合計レイテンシ
MM 操作	IALU, ILOG, ISHF, ST, LD	4 サイクル未満の間隔でスケジューリングされている場合は、10 クロックのフラッシュ 4 サイクル以上の間隔でスケジューリングされている場合は、4 クロック
TOBR、TOPR、TOAR (pfs のみ)	BR	0
FRBR, FRCR, FRIP, FRAR (FRxx)	MMMUL, MMSHF, MMALU	FRxx + 1
FMAC	FMISC, FCVTXF, XMA	7
	FMAC[a]	7
SFMAC	SFMISC	7
	SFMAC[a]	7
SFxxx (32 ビット並列 FP)	Fxxx (64/82 ビット FP)	SFxxx + 2 サイクル
Fxxx (64/82 ビット FP)	SFxxx (32 ビット並列 FP)	Fxxx + 2 サイクル
すべての FP レジスタ書き込み命令 (ロードを除く)	STF, FRFR	8

a. 一般的に、FMAC (SFMAC) のレイテンシは、他の FMAC によって参照される場合は 5 サイクルである。しかし、生成する側の FMAC (SFMAC の場合はいずれかの半分) の結果が、NaN、無限大、0、または NaTVal である場合は、結果を参照する側の FMAC に対するレイテンシは 7 サイクルになる。これらの条件の有無を検出するアルゴリズムは、厳密なものではない。したがって、実際にはこれらの条件が存在しないにもかかわらず、7 サイクルのレイテンシが発生する場合もまれにある。

3.5 プレディケートとバイパス処理

プレディケートがある場合、2 つの非ユニット・レイテンシ命令の間の実際の依存関係は、しばしばコンパイル時ではなく実行時に決まる。*プレディケートは、ユニット・レイテンシ命令のバイパス・レイテンシに影響を与えない。*この節ではバイパス処理に対するプレディケートの影響について詳しく説明するが、これらの条件がパフォーマンスに与える全体的な影響は、通常は非常に小さい。

以下のコードを考える。ランタイムに p1 が偽であったとすれば、命令 B と命令 C の間に依存関係は発生せず、C はサイクル 2 で発行される。これに対して、p1 が真であったとすれば、命令 C は命令 B に依存するため、B が完了するまでストールすることになる。

```
      cmp.eq  p1,p2=r5,r4 ;; // Cycle 0: instr A
(p1)  ld8     r1=[r3] ;;     // Cycle 1: instr B
      add     r3=r1,r2       // Cycle ?: instr C
```

プレディケートの真偽に関係なく、プレディケートを生成する側の命令とバイパスされる汎用レジスタを参照する側の命令の間隔が 2 サイクル

より小さい場合は、参照する側の命令は、比較命令の後の 2 番目のサイクルまでストールする。以下のコードでは、参照する側の命令 (add) にプレディケートがあり、生成する側の命令 (ld8) にはプレディケートがない。比較命令と参照する側の命令の間隔が 1 サイクルしかないため、add は、プレディケートが真であるかどうかわかるまで、サイクル 2 までストールする。

```
      cmp.eq  p1,p2=r5,r4 // Cycle 0
      ld8     r1=[r3] ;;  // Cycle 0
(p1)  add     r3=r1,r2    // Stalls until cycle 2
                          //  even if p1 is false
```

以下の特殊な例では、比較命令とバイパスされるレジスタ値を参照する側の命令の距離が、3 サイクル必要である。

```
      cmp.eq  p1,p2=r5,r4 ;;// Cycle 0
(p1)  add     r1=r2,r3 ;; // Cycle 1
      ld8     r6=[r1]     // Cycle 3
```

このコードの実行中に、ld8 は 1 サイクルだけストールする。このストールを避けるには、cmp と ld8 の間のどこかに 1 サイクルを追加する必要がある。この条件は、以下の 3 つの条件を満たす場合に適用される。

1. プレディケートを持つアドレス計算命令が、ロード命令に結果を供給する。
2. アドレス計算が M スロットで実行される。
3. アドレス計算が IALU 命令である (ILOG 命令や PNT 命令ではない)。

4.0 メモリ階層

Itanium プロセッサのメモリ階層は、以下の要素で構成される。

- 1 次データ・キャッシュ (L1-D)
- 1 次命令キャッシュ (L1-I)
- 2 次ユニファイド・キャッシュ (L2)
- 3 次ユニファイド・キャッシュ (L3)
- 1 次データ・トランスレーション・ルックアサイド・バッファ (L1-DTLB)
- 2 次データ・トランスレーション・ルックアサイド・バッファ (L2-DTLB)
- 命令トランスレーション・キャッシュ (ITLB)
- メイン・メモリ (フロントサイド) バス

4.1 L1 データ・キャッシュ

L1 データ・キャッシュは、32 バイト・ライン、16K バイト、4 ウェイ・セット・アソシアティブ、ライトスルー、非ライト・アロケート・メモリである。L1 キャッシュは、1 クロック当たり 2 つのロード、2 つのストア、または 1 つのロードと 1 つのストアを連続に実行できる。L1 にヒットする整数ロードは、データを参照する側のほどんどの操作に対して 2 サイクルのレイテンシを持つ。浮動小数点ロードは、常に L1 データ・キャッシュを使用しない。

すぐ後にロードされる値を書き込むストア (ストアからロードへのフォワーディングと呼ばれる) には、余分なサイクルがかかることがある。(アライメントされた 64 ビット領域の任意の部分の中で) 最近 3 サイクル以内にストアが行われたアドレスからロードする場合、ロードは L1 を使用せず、L2 からデータを読み出す。

4.2 L1 命令キャッシュ

L1 命令キャッシュは、32 バイト・ライン、16K バイト、4 ウェイ・セット・アソシアティブである。

4.3 L2 ユニファイド・キャッシュ

L2 キャッシュは、64 バイト・ライン、ユニファイド、96K バイト、6 ウェイ・セット・アソシアティブ、ライト・バック、ライト・アロケート・メモリである。L2 キャッシュは、2 つの汎用ポートを持ち、1 クロック当たり最大 2 つまでのメモリ操作または 1 つのライン・フィル操作を保持できる。L2 にヒットする整数ロードは、データを参照する側のほとんどの操作に対して 6 サイクルのレイテンシを持つ。浮動小数点ロードは、データを参照する側の操作に対して 9 サイクルのレイテンシを持つ (L2 で衝突がない場合)。

この節の後半は非常に詳細な内容であり、大量のメモリ処理を必要とするストリーミング・アプリケーションの開発者のみを対象とする。

大量のメモリ処理を必要とするコードに影響を与える、*L2* キャッシュ・パイプライン・フラッシュ状態は多数存在する。しかし、普通のプログラムでは、これらの条件に関する最適化は不要である。L2 パイプライン・フラッシュとは、メモリ・パイプライン内でメモリ操作を部分的に再実行する必要があるという意味である。L2 パイプライン・フラッシュが 1 回起こると、フラッシュされるメモリ参照のレイテンシが 6 クロック増えるが、メイン・プロセッサのパイプラインには直接影響を与えない。しかし、L2 キャッシュ・パイプライン・フラッシュが繰り返し発生して、多くのメモリ操作が停滞したり、フラッシュされる操作に依存する操作を実行できなかった場合は、その結果メイン・パイプラインがストールする。

メモリ参照が L2 にヒットせず、同じキャッシュ・ラインに未処理のミスがない場合、この状態をプライマリ・ミスと呼ぶ。セカンダリ・ミスとは、既に未処理のプライマリ・ミスがあるラインに対する L2 ミスである。L3 は、1 クロック当たり 1 つのプライマリ L2 ミスを処理できる。プライマリ・ミスが 1 クロックに 2 つ以上発生すると、2 番目のミスで L2 パイプ・フラッシュが発生する。

L2 は、一度に最大 8 本までの未処理のキャッシュ・ライン上のミスを許容する。また、各 L2 ラインの 4 つの 16 バイト部のそれぞれが、最大 2 つまでのメモリ操作による要求を保持できるので、1 つのラインで最大 8 つまでの未処理のミスを保持できる。いずれかの最大値を超える命令は、L2 パイプライン内でフラッシュされる。フラッシュされた L2 要求を待機中の操作がある場合や、このような要求が多数続けて発生する場合を除いて、L2 パイプ・フラッシュはメイン・パイプラインに直接には影響を与えないが、L2 キャッシュの帯域幅を消費する。

複数のストリーミング・ストアが、実行ストリーム内で短い間隔で発生すると、パフォーマンスに影響を与える可能性がある。以下の表に示すように、実行を開始するストアのすぐ後に別のストアが続くとき、両方のストアが同じ 8 バイト・アライメント領域にアクセスする場合は、L2 キャッシュのストールまたはフラッシュが発生することがある (異なる領域へのストアでは、特殊な動作は発生しない)。

後続するストアのサイズ	2 つのストアが 3 クロック以内の間隔で同じ 8 バイト・アライメント領域に書き込んだときの結果			
	0 サイクル	1 サイクル	2 サイクル	3 サイクル以上
st1, st2	L2 パイプフラッシュ	L2 パイプフラッシュ	L2 パイプフラッシュ	なし
st4	1 クロックのストール	なし	L2 パイプフラッシュ	なし
st8	L2 パイプフラッシュ	L2 パイプフラッシュ	L2 パイプフラッシュ	なし

同様に、ストア操作の後 3 サイクル以内に、同じ 64 ビット・アライメント領域の全部または一部にアクセスするロード操作が続くと、L2 パイプライン・フラッシュが発生する。これによって、ロードのレイテンシは、L2 パイプフラッシュのコストだけ大きくなる。

4.4 オフ・チップ、オン・パッケージの L3

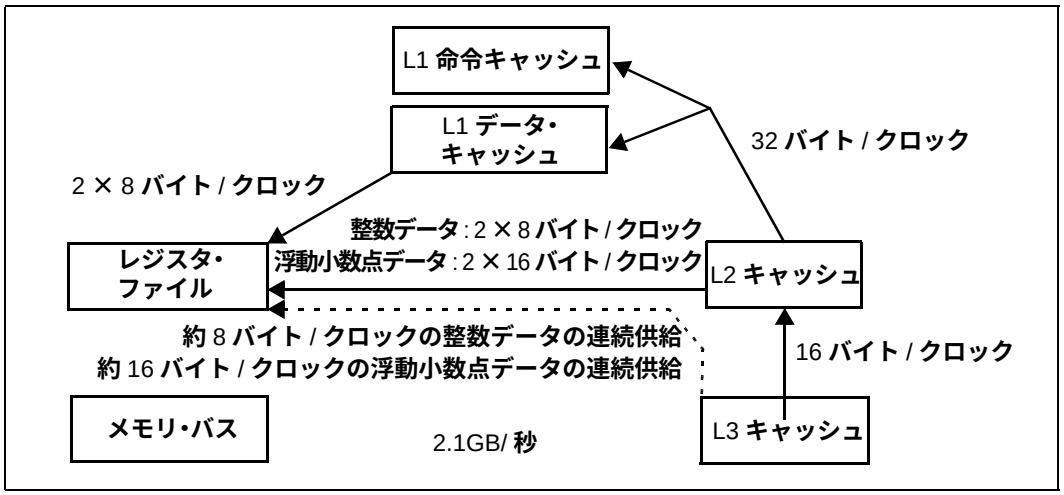
L3 キャッシュ・ヒットは、整数を参照する側の操作に対して 21 クロックのレイテンシを持ち、浮動小数点数を参照する側の操作に対して 24 サイクルのレイテンシを持つ。L3 キャッシュのサイズと構成は、Itanium のパッケージによって異なる。L3 キャッシュから L2 キャッシュへの最大帯域幅は、16 バイト×コア周波数である。

4.5 メイン・メモリ・バス

Itanium プロセッサのフロントサイド・バスのおおよその最大帯域幅は、2.1GB/ 秒である。

4.6 ロード帯域幅のまとめ

次の図に、データの各種のソースとデスティネーションの間のおおよそのピーク帯域幅を示す。図中の線は、Itanium プロセッサの実際のバスやバス幅を示すとは限らない。



レジスタ・ファイルは、キャッシュからの 1 クロック当たり最大 2 つの整数レジスタのフィルまたは最大 2 つの浮動小数点レジスタのフィルを処理できる。

4.7 トランスレーション・ルックアサイド・バッファ

Itanium プロセッサは、2 レベルのデータ TLB (L1-DTLB と L2-DTLB) を持つ。これらの TLB は、いずれもフル・アソシアティブである。L1-DTLB 内でミスがあると、10 サイクルのメイン・パイプ・フラッシュのペナルティが生じる。L2-DTLB 内でミスがあると、すべての必要な情報が L2 データ・キャッシュ内にある場合、23 サイクルのコストがかかる。

L1-DTLB は 32 エントリ、L2-DTLB は 96 エントリを持つ。L1-DTLB と L2-DTLB がサポートしているページ・サイズは、4k、8k、16k、64k、256k、1M、4M、16M、64M、および 256M である。サポートしているパージは、すべてのページ・サイズと 4G である。4G を超えるパージ・ページ・サイズを指定すると、その領域内のすべてのページがパージされる (ただし、パージ動作に関する情報は、説明目的でのみ提供されてい

る。Itanium 固有の動作を使用して必要なパージを実行するコードは、将来のプロセッサでは正常に動作**しない**)。仮想メモリ・システムについての詳細は、『*Intel*® *IA-64* 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。

1 レベルのみの命令 TLB は、64 エントリのフル・アソシアティブ・メモリである。命令 TLB とデータ TLB は、PAL を呼び出すことによって、トランスレーション・レジスタの数を照会できる。

4.8 データ・スペキュレーション、アドバンスト・ロード、および ALAT

`ld.a`、`ld.c`、および `chk.a` で構成される一連の命令は、ロードとストアの間でメモリ・アドレスを動的に明確化する機能を持つ。アーキテクチャ上、`ld.c` 命令と `chk.a` 命令のレイテンシは参照する側の命令に対して 0 サイクルである。しかし、`ld.c` または `chk.a` が ALAT にヒットしなかった場合は、追加のレイテンシが発生する。

`ld.c` が ALAT にヒットしないと、10 サイクルのパイプライン・フラッシュが発生する。ロードの再実行時に `ld.c` 命令がキャッシュ・ミスになった場合は、さらに多くの時間がかかる。`chk.a` が ALAT にヒットしないと、リカバリ・コードへの分岐が実行される。Itanium プロセッサ上では、この分岐は、リカバリ・コードに分岐する、オペレーティング・システムへのトラップとして実現されている。したがって、`chk.a` が ALAT にヒットしないときのコストは、通常は、2 つの分岐の予測ミスのコスト + リカバリ・コードのコスト＋制御レジスタを数回読み出すコスト + リターンのコストになる。これは、約 50+ サイクル＋リカバリ・コードのコストに相当する。

Itanium プロセッサ上では、ALAT は、アドバンスト・ロードの物理ターゲット・レジスタに基づく、32 エントリの 2 ウェイ・セット・アソシアティブ・メモリである。ALAT のエントリには、物理レジスタ番号以外に、ラップ・ビットのタグが付いている。このビットは、RSE 動作による同じ物理レジスタの異なるインスタンスを識別する。

仮想レジスタから物理レジスタへのマッピングは、スタック・レジスタとローテート・レジスタによって複雑な影響を受ける。このため、仮想ベース・レジスタ (r32) に対応付けられる物理レジスタを、コンパイラが常に認識しているとは限らない。したがって、コンパイラはコードを生成する際に、ALAT 内のローテート・レジスタとスタック・レジスタの間の結合関係を正確にモデル化することはできない。

Itanium プロセッサの ALAT は 2 ウェイ・アソシアティブであるため、一連の 3 つ以上のアドバンスト・ロードが、同じ 2 つのエントリに対応付けられることがある。ローテート・レジスタがない場合、いくつかの競合は簡単に予測できる。

```
ld8.a r32=[ r1 ]
ld8.a r48=[ r2 ]
```

```
ld8.a r64=[ r3 ]
```

上の例では、3 番目のロードが、Itanium プロセッサの ALAT のサイズと構造に基づいて以前の 2 つのアドバンスト・ロードによって割り当てられたエントリのうち 1 つと競合する。レジスタを選択する際は、上の例のような明らかな場合を認識することによって、ALAT 内の競合を避ける必要がある。

*レジスタ番号*は、アドバンスト・ロードのインデックスとチェック用に使用される。これに対して、ストア・*アドレス*は、重複するエントリの無効化に使用される。Itanium プロセッサ上では、ロード・アドレスの最下位 20 ビットだけが ALAT 内に保存され、競合する可能性があるストアに対して比較される。照合が不完全なため、最下位 20 ビットが一致する場合、実際には競合しないストアが ALAT のエントリに対して別名参照され、エントリが無効化される可能性がある。ストアの直後に `chk.a` または `ld.c` が続く場合は、次の表のように、20 ビットより少ないビット数を使用して、メモリの競合が発生したかどうかを判定する。

競合検出アルゴリズム	条件の説明
常に競合を報告する	ストアと同じサイクルに `chk.a` が続く。 A store followed by a in the same cycle
	`ld.a` と `ld.c` が同じクロック内で実行される。
最下位 12 ビットを使用して競合を検出する	ストアと同じサイクルまたは次のサイクルに `ld.c` が続く。
	ストアの次のサイクルに `ld.c` が続く。
	スヌープの次のサイクルに `chk.a` または `ld.c` が続く。
最下位 20 ビットを使用して競合を検出する	その他の場合

アソシエティビティの制限と不完全なアドレス・マッチングの影響で、多数のストアまたはアドバンスト・ロード、あるいはその両方が発生した場合、実際にはメモリやレジスタが競合していないにもかかわらず、ALAT のエントリが無効化されることがある。このような特性があるため、プログラマはアドバンスト・ロードを利用できる動的なストアの数に注意する必要がある。

最後に、アライメントの合っていないアドバンスト・ロードを使用すると、必要以上の数の ALAT エントリがパージされたり、エントリが割り当てられない場合がある。したがって、Itanium プロセッサ上で、自然境界にアライメントされていないロードと ALAT を組み合わせて使用することは推奨できない。

4.9 コントロール・スペキュレーション

ほぼすべての IA-64 命令は、デフォルトではスペキュレーティブ命令である。ロード命令には、スペキュレーティブ版と非スペキュレーティブ版がある。コンパイラは、IA-64 コントロール・スペキュレーションを使用するとき、`chk.s` 命令を挿入して、リカバリ・コードを実行する必要があ

るかどうかチェックする責任を負う。chk.s 命令は、セットされた NaT ビット (遅延例外フォルト) を検出すると、リカバリ・コードに分岐する。NaT フォルトからの回復の合計コストは、chk.a の場合と同じ (約 50+ サイクル＋リカバリ・コードのコスト) である。

ただし、このようなコントロール・スペキュレーションによる遅延例外フォルトは非常にまれであり、既に長いレイテンシを持つイベント (TLB ミス、ページ・フォルトなど) を伴う。したがって、これらのフォルトが、パフォーマンスに大きな影響を与える可能性は低い。

Itanium プロセッサ上では、L1-DTLB にヒットしないすべてのメモリ操作は、10 サイクルの L1-DTLB ミス・ペナルティを発生させる。すべてのスペキュレーティブ版および非スペキュレーティブ版のロード命令とストア命令は、オペレーティング・システムの NaT 遅延に関するポリシーに関係なく、このような動作をとる。

5.0 分岐と制御フロー

IA-64 命令は、分岐予測と制御フローの管理に利用できる。この節では、分岐、分岐ヒント、およびプリフェッチに関連するリソース、タイミング、および制限について説明する。5.4 節では、分岐ヒントの機能、操作、解釈、および各種のヒントの組み合わせ例を簡単にまとめて示す。

5.1 分岐予測

Itanium プロセッサは、分岐の向きとターゲット・アドレスを予測するための各種の構造を備えている。どの構造が使用されるかは、予測対象の分岐のタイプ、分岐プレディクタの状態、および分岐命令内で指定されるヒント・コンプリータによって決まる。

5.4 節に、分岐予測に関するまとめを示す。

この節は非常に詳細な内容であり、高度な従来の最適化手法、プレディケート、スペキュレーションの経験則、およびソフトウェア・パイプライニングを既に完了したコンパイラにのみお勧めする。この節の内容は、分岐予測に関連する Itanium プロセッサ固有の最適化を積極的に実行しようとするプログラマを対象とする。

5.1.1 分岐の向きの予測

Itanium プロセッサは、分岐が実行されるかどうかを予測する、2 つの主なリソースを備えている。

- MFB、MMB、および MIB バンドル内の分岐には、大きな分岐予測テーブル (BPT) を使用する。

- MBB および BBB バンドル内の分岐には、小さなマルチウェイ分岐予測テーブル (MBPT) を使用する。MBPT のマルチウェイ分岐エントリの数は、BPT の単一分岐エントリの数の 1/8 である。

MBB または BBB バンドルを使用すると、B スロットにヒント、nop、分岐のうちどれが入っているかに関係なく、マルチウェイ分岐プレディクタ内でエントリが割り当てられる。したがって、可能な限り、B スロットに nop を挿入する必要がない、これ以外のタイプのバンドルを使用する方が望ましい。

BPT および MBPT テーブルは、いずれも 4 ウェイ・セット・アソシアティブであり、バンドル・アドレスを使用してキャッシュと同じようにアクセスされる。分岐がこれらの構造のエントリにヒットしなかった場合は、分岐にエンコードされた静的予測が使用される。ただし、分岐が TAC(ターゲット・アドレス・キャッシュ) または TAR(ターゲット・アドレス・レジスタ) にヒットした場合は、その分岐は実行されるものと予測される。

MBPT および BPT 内で使用される予測アルゴリズムは、4 ビットの履歴を持つ 2 レベルのローカル・プレディクタである。MBPT のリソースは BPT のリソースと同じであるが、MBPT 内の各エントリには、1 つの分岐スロットではなく、3 つの分岐スロット用のリソースが含まれる。さらに、BPT と MBPT には、リターンと呼び出しのために適切な処置がとれるように、分岐のタイプを指定するフィールドがある。BPT と MBPT のエントリの割り当ては、分岐に関連付けられたヒントと分岐の結果によって決まる。これについては、分岐ヒントと分岐予測命令に関する項目で説明する。

5.1.2 分岐先アドレスの予測

TAR(ターゲット・アドレス・レジスタ) は、4 エントリのフル・アソシアティブ高速バッファである。`.imp` (*important*) コンプリータを持つ `brp` 命令だけが、TAR に書き込むことができる。TAR にヒットすると、分岐が BPT 内にあるか MBPT 内にあるかを問わず、その分岐は*実行される*ものと予測される。また、予測対象の分岐のターゲット・アドレスも TAR から得られる。ただし、ループ・カウント・レジスタ (LC と EC) がループの終了を指示した場合は、ループ・プレディクタは TAR の予測を無効にできる。

TAC(ターゲット・アドレス・キャッシュ) は、TAR より大きな 64 エントリの構造である。`brp` 命令、分岐、または予測ハードウェアが TAC に書き込める。分岐が BPT または MBPT にヒットした場合、TAC がターゲット・アドレスを供給する責任を負い、BPT または MBPT は分岐が実行されるかどうかを判定する。TAC は 1 バンドル当たり 1 つのアドレスしか保持できない。このため、TAC にはターゲット・アドレスがどのスロットに対応するかを指定するフィールドがある。ターゲット・アドレスの保持以外に、分岐が BPT や MBPT にヒットせずに TAC にヒットした場合、TAC は*実行*の予測結果を返す。

RSB は呼び出しの際にリターン・アドレスがプッシュされ、リターンの際にターゲット・アドレスがポップされるアドレス・スタックである。

BAC は IP 相対分岐のターゲット・アドレスなど、アドレスを迅速に計算できる場合に、分岐の正確なターゲット・アドレスを計算する。BAC には、BAC1 と BAC2 と呼ばれる 2 つのステージがある。

5.1.3 タイミングに関する留意点

分岐のターゲットまたは向きの予測に最後に使用される機構は、実行パイプラインに挿入されるバブルの数に影響を与える。

TAR にヒットした実行される分岐には、それに関連する実行分岐バブルは発生しない。これは、Itanium プロセッサ上で可能な最も高速の分岐形式である。一般的に、TAR にヒットせずに正しく予測される分岐には、1 サイクル以上のパイプライン・バブルが生じる。しかし、このようなバブルは、Itanium プロセッサ内のデカップリング・バッファに吸収され、パフォーマンスに大きな影響を与えない。分岐に先行するコードに他の種類のストールがある場合は、バブルはさらに吸収されやすい。

TAC によってターゲットが与えられた実行される分岐には、それに関連する 1 サイクルの分岐バブルが発生する。

IP 相対分岐が TAR と TAC にヒットしない場合は、分岐バブルが挿入され、BAC が正確なアドレスを計算する。予測された最初の分岐がバンドルの第 3 スロットにある場合は、分岐先アドレスは BAC1 によって計算され、2 サイクルのバブルが生じる。それ以外の場合は、分岐先アドレスは BAC2 によって計算され、3 サイクルのバブルが生じる。ターゲット・アドレスの計算によって、20 番目のビットを超えてキャリーが発生する場合は、BAC2 が使用される。

`br.ret` 命令が、BPT または MBPT によって*実行*すると予測された場合は、1 サイクルの実行分岐バブルが発生する。分岐が BPT と MBPT にヒットせず、分岐にエンコードされた静的予測を使用するときは、2 サイクルのバブルが発生する。

間接分岐が TAR と TAC にヒットしないとき、分岐先アドレスは、RSB のスタックのトップから与えられる (RSB の状態は変更されない)。この予測には、3 サイクルの分岐バブルが生じる。この場合、予測されるターゲット・アドレスは、リターンの場合を除いて不正確である可能性が非常に高い。

5.2 ヒントによる分岐予測の制御

IA-64 は、3 つの方法で Itanium プロセッサの分岐予測ハードウェアを制御できる。3 つの方法とは、明示的な分岐予測命令、分岐レジスタへの移動命令、および分岐命令にエンコードされるヒントである。ヒントの構

文はアーキテクチャ・レベルであるが、ヒントの影響とタイミングはマイクロアーキテクチャ固有である。

brp命令または分岐レジスタへの移動命令によるヒントが有効であるためには、これらの命令を対応する分岐の前に最小限必要な間隔を置いてスケジューリングする必要がある。この間隔はサイクル数で指定されることも、命令フェッチの回数で指定されることもある。サイクルは (命令のレイテンシと同じように) 時間の単位である。フェッチは、時間の単位ではなく、静的または動的な距離の単位 (バンドル単位) である。Itanium プロセッサ上では、1 フェッチは 2 バンドル (すなわち、1 つの命令キャッシュ・ライン) に相当する。

5.2.1 分岐命令にエンコードされるヒント

sptk、spnt、dptk、および dpnt 分岐命令コンプリータは、分岐が動的予測ハードウェア内でヒットしなかったときに、どの予測構造を使用するかを指定する。静的 (sp) ヒントと動的 (dp) ヒントは、予測構造の割り当てと予測に与える影響が異なる。

静的ヒント (spnt/sptk) は、BPT 内でスペースを割り当てないように指定する。これによって、(M)BPT 内で偶然の一致がない限り、静的予測が使用される。ただし、以下の特殊な条件が適用される。

- 分岐に sptk ヒントが付加されているが、分岐が実行されず、リターンでもない場合は、その分岐のエントリが TAC 内で割り当てられる。
- 分岐が呼び出しまたはリターンであり、sptk ヒントが付加されている場合、分岐は常に実行されるものと予測されるが、BPT または MBPT 内にも割り当てられる。

動的ヒント (dpnt/dptk) は、動的予測ハードウェアを使用して以下の動作を行うように、Itanium プロセッサに指示する。

- 非リターン分岐が実行される場合は、TAC エントリが割り当てられる。
- 最初の予測ミスが起こるまで、分岐有無ヒントを使用して分岐を予測する。最初の予測ミスが起こった時点で、BPT または MBPT エントリが割り当てられ、その後の予測に使用される。
- リターンまたは呼び出し命令が実行される場合は、BPT または MBPT エントリが割り当てられる。

dealloc (clr) ヒントは分岐の結果にかかわらず、予測構造の割り当てや更新を行わないように Itanium プロセッサに指示する。分岐にエンコードされた静的予測が使用されるが、この場合も呼び出しとリターンは RSB を更新する。ただし、clr コンプリータは、BPT、MBPT、または TAC 内の既存のエントリを実際に削除するのではなく、割り当てを防ぐだけである。

5.2.2 分岐予測命令

分岐ヒント命令は、分岐ターゲットと分岐の向きを指定できる。これらのヒントは、分岐予測 (brp) 命令によって指定される。brp 命令は単なるヒントであり、プログラムの整合性には影響を与えず、IA-64 プロセッサによっては無視されることがある。Itanium プロセッサでは、バンドルの第 3 スロットに置かれた brp 命令だけが認識される。それ以外の brp 命令は、nop として処理される。

Itanium プロセッサ上の分岐予測命令については、loop、sptk、dptk (exit バージョンはサポートされていない) の各ヒント・タイプに違いはない。これらのヒントが指定されると分岐に TAC エントリが割り当てられ、(M)BPT 内にマッチするエントリがないと分岐は実行するものと予測される。分岐が brp 命令の結果を使用するまでに、TAC への書き込みが有効になっている必要がある。したがって、brp 命令はヒントの対象となる分岐より少なくとも 4 フェッチ前に置かなければならない。

brp 命令に imp コンプリータがある場合は、TAC エントリ以外に TAR エントリも割り当てられる。TAR は TAC より 1 サイクル早く読み出される。したがって、brp.imp 命令が有効であるには、関連する分岐より 5 フェッチ前に置かれていなければならない。

brp 命令にはリターン形式と間接分岐形式もあるが、いずれの形式も Itanium プロセッサではサポートしていないため、nop として処理される。間接分岐のヒント機能は、分岐レジスタへの移動命令によって実現される。

brp 命令は、ヒントの対象となるバンドルのアドレスを指定するだけである。したがって、MBB および BBB(マルチウェイ) 分岐の場合、どのスロットがヒントの対象となるかをソフトウェアによって指定することはできない。Itanium プロセッサは、すべての brp 命令が第 3 スロットを指すものと見なす。によって TAC への書き込みが発生する場合、対応するスロットを示す TAC エントリのフィールドは常に 3 に設定される。

5.2.3 分岐レジスタへの移動命令

分岐レジスタへの移動命令が sptk または dptk コンプリータを持つ場合は、TAC の更新が行われる。

Itanium プロセッサはパイプライン構造を持つため、分岐レジスタへの移動命令がヒントの対象となる分岐よりどれだけ先行しなければならないかを示す、最小限の命令キャッシュ・ライン数はかなり大きくなる。この値は実行されるコードの特性と、実行パイプラインの状態によって決まる。通常は、分岐がヒントを参照するには、分岐レジスタへの移動命令と分岐の間に、実行時に少なくとも 9 サイクルの間隔が必要である (ただし、分岐命令は、*mov to br* を待ってストールすることはない。このレイテンシは *mov to br* 命令が指定したヒントが、ヒントを参照する側の分岐によって認識されるかどうかにのみ影響を与える)。より厳密には、この方法でメリットを得るには、9 ～ 13 フェッチの間隔が必要である。

5.2.4 最後の反復の予測 : br.ctop と br.cloop

Itanium プロセッサは、ループの終わりに実行されない分岐がくるのを避けるために、br.ctop および br.cloop 分岐用の特殊な分岐プレディクタを持っている。この項では、このプレディクタが有効になるために満たさなければならない制限と、特殊なコードの例を示す。

br.cloop と br.ctop のいずれについても、完全な予測を行うには、分岐予測命令 (brp.loop.imp) を使用して、ループの分岐アドレスを指定しなければならない。

また、(br.ctop ではなく)br.cloop の場合、プレディクタが最後の反復を正しく予測するためには、br.cloop と EC レジスタの値の間にアーキテクチャ上の関係がなくても、AR[EC] レジスタの値が 1 になっていなければならない。したがって、br.cloop とループ分岐終了プレディクタを組み合わせて使用するコードは、上記の brp.loop.imp を使用する以外に、br.cloop に移行する前に EC の値を 1 に初期化する必要がある。

カウント指定ループの分岐プレディクタは、極端に短いループでは、最初の数回の反復の処理がパイプライン内でまだ完了していないため、最後の分岐の予測ミスを避けられない。

5.3 ヒントによる命令プリフェッチの制御

命令のプリフェッチにより、L1-I キャッシュのレイテンシを短縮したり、場合によっては除去が可能である。Itanium プロセッサでは、以下の方法でプリフェッチを要求できる。

- 分岐命令にエンコードされるヒント

各サイクルで最大 2 つの brp を発行できるが、プリフェッチ・ヒントは 1 クロック当たり 1 つしか処理できない。一般的に、各サイクルで 2 つ以上のプリフェッチを発行しようとすると、結局はプリフェッチの一部が廃棄される。プリフェッチは発行されてキューに入れられると、分岐の予測ミスのために取り消しまたはフラッシュされない限り、結局はキャッシュに送られる。プリフェッチ・キューが一杯になると、その後のプリフェッチは廃棄される。

Itanium プロセッサでは、バンドルの第 3 スロットに置かれた brp 命令だけが認識される。その他の brp 命令は、nop として処理される。

一部の IA-64 命令は、多数 (*many*) または少数 (*few*) の命令キャッシュ・ラインをフェッチするように指定するプリフェッチ・ヒントを含む。*many* と *few* の定義は、プロセッサによって異なる。Itanium プロセッサは、ストリーミング・プリフェッチとライン・プリフェッチの 2 つのプリフェッチ・アルゴリズムをサポートしている。ライン・プリフェッチは、2 つの L1 キャッシュ・ラインを取り出す。ストリーミング・プリフェッ

チは、実行されると予測された分岐がメイン・パイプライン内で検出されるまで、キャッシュ・ラインを続けてフェッチする。

5.3.1 分岐命令上のストリーミング・プリフェッチ・ヒント

Itanium プロセッサでは、分岐命令上でプリフェッチ・ヒントが指定されると、その分岐が実行されると予測される場合にのみ、プリフェッチが実行される。プリフェッチされる一連の命令は、分岐のターゲットの後の最初の L2 キャッシュ・ラインから始まる。プリフェッチは、実行されると予測される次の分岐が検出されるまで続く。ただし、ストリーミング・プリフェッチャは、メイン・パイプラインから切り離され、実行すると予測される分岐がメイン・パイプライン内で検出されるまでフェッチを続ける。したがって、実行されるコードに多くのストールがある場合、プリフェッチャは次の領域の終わりよりかなり先のデータまでフェッチすることがある。

バンドルの第 3 スロット内の分岐だけが、ストリーミング・プリフェッチを開始できる。

5.4 分岐予測とプリフェッチのまとめ

分岐予測、命令プリフェッチ、およびヒントについては、大量の複雑な情報があるため、この節ではこれらを箇条書きにして示す。詳しい説明は、この節の前半に記載されている。

5.4.1 構文のまとめと解釈

この項は、分岐上でのヒントの指定、分岐予測命令、分岐への移動命令に使用される IA-64 命令の構文と Itanium プロセッサ上でのその解釈についてまとめたものである。

分岐命令上のヒントの解釈

ここでは Itanium プロセッサのハードウェアが分岐命令上の各種のヒント・コンプリータをどのように解釈するかについて説明し、いくつかの例を示す。

分岐有無ヒント：

- *sp*XX – この分岐に対して (M)BPT 内でスペースを割り当てない。
- *dp*XX – この分岐の最初の予測ミスの後、(M)BPT 内でスペースを割り当てる。
- XX*nt* – (M)BPT エントリが見つからない場合、この分岐が実行されないものと予測する。
- XX*tk* – (M)BPT エントリが見つからない場合、この分岐が実行されるものと予測し、分岐ターゲットを TAC に書き込む。

割り当て解除ヒント :

- *clr* – この分岐が実行されるとき、BPT、TAC、または TAR 内でスペースを割り当てない。ただし、別名参照されるエントリや brp 命令 / 分岐レジスタへの移動命令によって、エントリが存在する可能性がある。
- コンプリータなし – その他の規則に従ってスペースを割り当てる。

プリフェッチ・ヒント :

- コンプリータなしまたは *few* – プリフェッチは実行されない。
- *many* – 分岐ターゲットの後の 2 次キャッシュ・ラインから、ストリーミング・プリフェッチを開始する。

その他の規則 :

- *CLR* または *spnt* が指定されている場合や分岐が `br.ret` である場合を除いて、最初に分岐が実行されたとき、TAC エントリが割り当てられる。
- *sptk* がエンコードされた br.call と `br.ret` については、(M)BPT は割り当てられるが、分岐は常に実行するものと予測される。

例 : `br.cond.sptk L1`

- (M)BPT エントリが見つからない場合は、分岐が実行されるものと予測する。それ以外の場合は、(M)BPT の予測を使用する。
- この命令を実行しても、新しい (M)BPT エントリは割り当てられない。
- `brp` 命令または他の分岐の別名参照によって、(M)BPT にヒットする可能性がある。
- 分岐が実行された場合は、TAC エントリが割り当てられる。
- プリフェッチは実行されない。

例 : `br.cond.dpnt.many L1`

- この分岐に対応する (M)BPT エントリが見つからない場合は、分岐が実行されないものと予測する。それ以外の場合は、(M)BPT の予測を使用する。
- この分岐が予測ミスになった (実行された) 場合は、(M)BPT エントリを割り当てる。
- 分岐が実行された場合は、TAC エントリが割り当てられる。
- 分岐が実行されると予測される場合は、分岐の後の 2 次キャッシュ・ラインからストリーミング・プリフェッチを開始する。

例 : `br.cond.sptk.few.clr L1`

- この分岐に対応する (M)BPT エントリが見つからない場合は、分岐が実行されるものと予測する。それ以外の場合は、(M)BPT の予測を使用する。
- この命令を実行しても、新しい (M)BPT エントリは割り当てられず、既存のエントリも更新されない。
- brp 命令または他の分岐の別名参照によって、(M)BPT にヒットする可能性がある。
- プリフェッチは実行されない。
- TAC エントリは割り当てられない。

BRP 命令の解釈

ここでは、brp 命令の構文と、Itanium プロセッサのハードウェアが各ヒント・コンプリータをどのように解釈するかについて説明し、いくつかの例を示す。

IP 相対分岐有無ヒント：

- *sptk, dptk, loop* – TAC エントリを割り当て、(M)BPT は更新しない。
- *exit* – (M)BPT や TAC に影響を与えない。

間接分岐のヒント：

- Itanium プロセッサではサポートしていない。
- 分岐への移動命令上のヒントが、この機能を実現する。

重要度ヒント：

- *imp* – TAC 以外に TAR エントリを割り当てる。
- コンプリータなし – その他の規則に従ってスペースを割り当てる。

例：`brp.loop.imp target25, L1`

- ラベル L1 の分岐に対して、target25 を指す TAR エントリを割り当てる。

例：`brp.dptk target25, L1`

- ラベル L1 の分岐に対して、target25 を指す TAC エントリを割り当てる。

分岐レジスタへの移動命令上のヒントの解釈

ここでは、分岐への移動命令の一部として指定されるヒントの構文と解釈について説明する。

分岐有無ヒント :

- *sptk, dptk* – TAC エントリを割り当て、(M)BPT は変更しない。
- コンプリータなし – TAC の割り当てを実行しない。

重要度ヒント :

- *imp* – TAC 以外に TAR エントリを割り当てる。
- コンプリータなし – その他の規則に従ってスペースを割り当てる。

5.4.2 分岐動作、予測、タイミング、およびリソース

この項では、分岐の向きとターゲットを判定する分岐予測を制御する、Itanium プロセッサ上のリソースと規則について説明する。これらの規則は、Itanium プロセッサ上で分岐がどのように実行されるかを完全に制御する。分岐ヒントはここで説明する動作以外は、特定の分岐の動作に直接に影響を与えない。分岐ヒントは分岐リソースの内容と割り当てを制御するだけであり、これらのリソースが分岐の動作を直接に制御する。

分岐の向きの予測リソース :

- MIB、MFB、および MMB バンドルの予測に使用される大きな分岐予測テーブル (BPT)
- MBB および BBB バンドルの予測に使用される小さなマルチウェイ分岐予測テーブル (MBPT)

分岐の向きの判定規則 :

- TAR 内でマッチするエントリが見つかった場合は、EC/LC がカウント指定ループ内でループの終了を指示した場合を除いて、分岐は実行すると予測される。EC/LC がループの終了を指示した場合は、マッチする TAR エントリは無効にされ、分岐は*実行しない*と予測される。
- BPT 内 (MIB、MFB、または MMB バンドルの場合) または MBPT 内 (MBB/BBB バンドルの場合) にエントリが見つかった場合は、その予測が使用される。
- (M)BPT エントリは存在しないが、TAR または TAC にヒットした場合は、分岐が*実行される*と予測する。
- 分岐にマッチする (M)BPT、TAC、または TAR のエントリが存在しない場合は、分岐命令にエンコードされた分岐有無ヒントを使用する。

分岐ターゲットの予測リソース :

- 64 エントリのターゲット・アドレス・キャッシュ (TAC)(1 バンドル当たり 1 エントリ)
- 4 エントリのターゲット・アドレス・レジスタ (TAR)(2 バンドル当たり 1 エントリ)
- リターン命令用の 8 エントリのリターン・スタック・バッファ (RSB)

- IP 相対分岐のために、2 つの分岐アドレス・コレクタ (BAC1 と BAC2) が存在する。BAC は、TAC と TAR にヒットしなかった分岐について、2 または 3 サイクルのバブルの後に、正確なアドレスを計算する。

分岐ターゲットの判定規則 :

- 分岐がリターンである場合は、RSB のトップのアドレスを使用する。
- TAR のエントリにヒットした場合は、そのアドレスを使用する。
- TAC のエントリにヒットした場合は、そのアドレスを使用する。
- IP 相対分岐が TAR と TAC にヒットしない場合は、BAC1 または BAC2 を使用してアドレスを計算する。
- 非 IP 相対分岐の場合は、予測ミスになる。

分岐タイミング / バブル :

- 分岐の予測ミスがあると、パイプライン内で 9 サイクルのバブルが発生する。
- TAR のエントリにヒットした分岐は、実行されるものと予測され、バブルは発生しない。
- TAC のエントリにヒットした分岐は、実行されるものと予測され、1 サイクルのバブルが発生する。
- TAC と TAR にヒットしない IP 相対分岐については、BAC1 が正確なアドレスを計算し、2 サイクルのバブルが発生する。
- TAC と TAR にヒットしない IP 相対分岐については、BAC1 が使用されないときは BAC2 が正確なアドレスを計算し、3 サイクルのバブルが発生する。
- RSB によってアドレスが指定されるリターン命令では、BPT または MBPT による予測の場合は 1 サイクルのバブルが発生し、分岐にエンコードされた分岐有無ヒントによる予測の場合は 2 サイクルのバブルが発生する。
- LC/EC がループの終了を指示した場合は、ループ・プレディクタは TAR を無効にできる。

6.0 一般的でない操作のレイテンシ

ここでは使用頻度の低い命令や、1 つの数値では簡単に説明できない (パイプライン、メモリ、TLB の状態などによって異なる) レイテンシを持つ命令について説明する。以下に挙げるレイテンシの多くは、システム・レベルのソフトウェアまたはコンパイラが使用する特殊な用途のために用意されている。このようなレイテンシの条件は複雑であるため、レイテンシは命令実行時のプロセッサの状態に影響を受けることを理解した上で、この節で示す数値はおおよその値と考える必要がある。

AR レジスタからの移動のレイテンシ	説明	レイテンシ（サイクル）
FRAR_M, FRAR_I	mov =ar.ccv	6
	mov =ar.unat	6
	mov =ar.rnat	6
	mov =ar.kr[0-7]	13
	mov =ar.bsp	13
	mov =ar.bspstore	13
	mov =ar.rsc	13
	mov =ar.fpsr	13
	mov =ar.eflag	13
	mov =ar.csd	13
	mov =ar.ssd	13
	mov =ar.cflg	13
	mov =ar.fsr	13
	mov =ar.lc	2
	mov =ar.ec	2
	mov =ar.pfs	2
	mov =ar.itc	38
	mov =ar.fir	38
	mov =ar.fdr	13
	mov =ar.fcr	38

未処理の `mov ar` 命令の数には制限があるが、この影響は通常のコードには現れない。

AR への移動のレイテンシ	説明	レイテンシ (サイクル)
TOAR_I, TOAR_M	mov ar.ccv=	5
	mov ar.unat=	5
	mov ar.lc=	1
	mov ar.ec=	1
	mov ar.rnat=	9(スピル / フィルに対して) 5(明示的な読み出しに対して)
	mov ar.kr=	2(明示的な読み出しに対して)
	mov bsp=	なし
	mov ar.bspstore=	10(スピル / フィルに対して) 5(暗黙的な ar.mat アクセスに対して)
	mov ar.rsc=	10(スピル / フィルに対して) 即値形式の場合 : 1(暗黙的な ar.bspstore または ar.rnat アクセスに対して) レジスタ形式の場合 : 10(暗黙的な ar.bspstore または ar.rnat アクセスに対して)
	mov ar.fpsr=	9(浮動小数点操作に対して) 2(明示的な読み出しに対して)
	mov ar.eflag=	2
	mov ar.csd=	2
	mov ar.ssd=	2
	mov ar.cflg=	2
	mov ar.itc=	35
	mov ar.fir=	4
	mov ar.fcr=	4
	mov ar.fsr=	23
	mov ar.pfs=	0(br.ret に対して)
	mov ar.fdr=	2

未処理の `mov ar` 命令の数には制限があるが、この影響は通常のコードには現れない。

未処理の `mov cr` 命令の数には制限があるが、この影響は通常のコードには現れない。

セマフォのレイテンシ	説明	レイテンシ (サイクル)
SEM	cmpxchg	おおよそ、L2、L3、またはメモリ + 5 クロックのレイテンシ。 これらの操作はパイプライン化されていない。
	xchg	
	fetchadd	

Itanium プロセッサ上では、セマフォを操作すると、パイプラインがストールする (レイテンシを隠蔽できない)。

各種のシステム命令のレイテンシ	説明	レイテンシ (サイクル)
SYST_I, SYST_I0, SYST_B, SYST_B2, SYST_M0, SYST_M	alloc	1(レジスタ引数に対して) ストールは RSE の状態によって異なる。
	flushrs	ストールは RSE の状態によって異なる。
	loadrs	ストールは RSE の状態によって異なる。
	probe, probe.fault	可変
	ttag	13
	thash	13
	tpa	6
	tak	6
	fc	可変
	mov =pkr	13
	mov =rr	13
	mov =psr	13
	mov =pmc	38
	mov =pmd	38
	mov =ibr	38
	mov =dbr	38
	mov =cpuid	38
	mov pkr=	10 (srlz に対して)
	mov rr=	10 (srlz に対して)
	mov pmc=	35 (srlz に対して)
	mov pmd=	35 (srlz に対して)
	mov ibr=	35 (srlz に対して)
	mov dbr=	35 (srlz に対して)
	rum	4 (使用に対して)
	sum	4 (使用に対して)
	mov psr.um=	4 (使用に対して)
	mov psr.l=	5 (srlz に対して)
	rsm	5 (srlz に対して)
	ssm	5 (srlz に対して)
	itc.i, itc.d	可変
	itr.i, itr.d	
	ptr.i, ptr.d	
	ptc.l	
	ptc.e	
	ptc.g, ptc.ga	
	invala.e	
	invala	

未処理のシステム命令の数には制限があるが、この影響は通常のコードには現れない。

CR への移動のレイテンシ	説明	レイテンシ (サイクル)
TOCR	mov cr.isr=	5 (srlz に対して)
	mov cr.iip=	5 (srlz に対して)
	mov cr.iipa=	5 (srlz に対して)
	mov cr.iim=	10 (srlz に対して)
	mov cr.iva=	5 (srlz に対して)
	mov cr.itir=	5 (itc/itr に対して) 10 (srlz に対して)
	mov cr.ifa=	5 (itc/itr に対して) 10 (srlz に対して)
	mov cr.ifs=	10 (srlz に対して)
	mov cr.ipsr=	10 (srlz に対して)
	mov cr.dcr=	5 (srlz に対して)
	mov cr.iha=	10 (srlz に対して)
	mov cr.pta=	10 (srlz に対して)
	mov cr.itm=	35 (srlz に対して)
	mov cr.lid=	35 (srlz に対して)
	mov cr.tpr=	35 (srlz に対して)
	mov cr.eoi=	35 (srlz に対して)
	mov cr.itv=	35 (srlz に対して)
	mov cr.pmv=	35 (srlz に対して)
	mov cr.cmcv=	35 (srlz に対して)
	mov cr.lrr[0-1]=	35 (srlz に対して)

未処理の `mov cr` 命令の数には制限があるが、この影響は通常のコードには現れない。

CR レジスタからの移動のレイテンシ	説明	レイテンシ (サイクル)
FRCR	mov =cr.isr	2
	mov =cr.iip	2
	mov =cr.iipa	2
	mov =cr.iim	2
	mov =cr.iva	2
	mov =cr.itir	13
	mov =cr.ifa	13
	mov =cr.ifs	13
	mov =cr.ipsr	13
	mov =cr.dcr	13
	mov =cr.iha	13
	mov =cr.pta	13
	mov =cr.itm	38
	mov =cr.lid	38
	mov =cr.ivr	38
	mov =cr.tpr	38
	mov =cr.eoi	38
	mov =cr.irr[0-3]	38
	mov =cr.itv	38
	mov =cr.pmv	38
	mov =cr.cmcv	38
	mov =cr.lrr[0-1]	38