ビット・ベクタを利用した選択的命令再発行機構

嶋田 創[†] 三輪 忍[††] 富田 眞治[†]

将来のプロセッサでは，配線遅延の増大により，命令を発行してから実行ユニットで実行を開始するまでの時間である，命令発行レンテンシが増大すると考えられる．この命令発行レイテンシの増大はロード命令に依存する命令の投機的なスケジューリングを増やし，投機ミスによる命令の無効化／再発行を増やすことになる．この投機ミスからの回復手法には，投機ミスの原因となった命令に依存する命令のみを選択的に無効化する手法と，非選択的に無効化する手法がある．前者の方がプロセッサ性能への悪影響は少ないが，実装は複雑になる．

本論文では，この投機ミスからの回復方法のうち，選択的に無効化を行う機構の実装方法の 1 つを提案する．提案機構をシミュレーションによって評価した結果，非選択的な無効化と比較して提案機構は無効化を行う命令を大幅に削減し，IPC の低下を押さえることを示した．

Selective Instruction Re-Issue Mechanism using Bit Vector

Hajime Shimada,[†] Shinobu Miwa[††] and Shinji Tomita[†]

In the future processor, a instruction issue latency which is the latency from instruction issue stage to execution stage will increase because of wire delay increase. Increase of the instruction issue delay increases speculative scheduling for load depend instructions. It increases squash and re-issue caused by speculative execution miss. There are two method to recover from this speculative execution miss. The one in selective squash method which squashes only dependent instructions. The other is non-selective squash method. The prior one is less adverse affect for processor performance but the implementation becomes complex.

We propose a implementation of the selective squash method. Our evaluation result shows that selective squash method can reduce re-issued instruction dramatically compared to the non-selective squash method. It also reduces IPC degradation compared to the non-selective squash method.

1. はじめに

近年のプロセッサ・アーキテクチャで問題となっているものの 1 つに，配線遅延の増大がある．この問題は，半導体プロセス技術の進歩にほぼ比例してゲート遅延はスケーリングされるが，配線遅延のスケーリングはそれよりもはるかに小さいため，プロセス技術の進歩に伴って配線遅延が増大しているように見えるものである．RAM や CAM など，遅延時間の多くが配線遅延によって占められているものは，この配線遅延の増大が特に大きい．

プロセッサを構成する要素のうち，RAM や CAM を主要な構成要素とするキャッシュ，命令スケジューラ，レジスタ・ファイルなどは，この配線遅延の増大の影響を特に大きく受ける．そのため，プロセッサのクロック・サイクル時間を一定の FO4 遅延時間とした場合，半導体プロセス技術の進歩にともなって，上記の構成要素のアクセス・サイクル数は延びてゆくと考えられる[1)]．

本論文では，配線遅延の増大によって起こる問題のうち，命令が発行されてから実際に機能ユニットで実行を開始するまでのサイクル数である，**命令発行レイテンシ**の増大の問題を取り扱う．この命令発行レイテンシの内訳には，RAM や CAM で構成される命令スケジューラ，ペイロード RAM，レジスタ・ファイルの遅延が含まれるため，特に大きく配線遅延の増大の影響を受けると考えられる．

命令発行レイテンシが大きなプロセッサでは，命令スケジューラは先行する命令の実行の完了を待たずに後続の命令の発行を行わなければならない．ここで問題となるのが，ロード命令とそれに依存する後続命令のスケジューリングである．通常の命令の実行レイテンシは一定のため，命令スケジューラは先行する命令の発行後，その命令の実行レイテンシ分のサイクル数後に後続の命令の発行を行えばよい．ところが，ロード命令の実行レイテンシはキャッシュにヒットするかミスするかによって異なり，通常はキャッシュ・ヒットすることを前提として投機的なスケジューリングを行う．この場合，先行するロード命令がキャッシュ・ミスを起こした場合，プロセッサはそれに依存する全ての命令を無効化／再発行しなくてはならない．この無効化すべき命令の数は，先行する命令がキャッシュ・ミスを起こしたかどうかが判別されるまでのサイクル数が増えるほど大きくなる．つまり，発行レイテンシが長いほど，無効化すべき命令数が増える．

上記の場合，データ依存関係にある後続の命令のみを無効化すればよい．しかし，このデータ依存関係にある後続の命令には，後続の命令に依存する，さらに後続の命令も

† 京都大学大学院情報学研究科
Graduate School of Infomatics, Kyoto University
†† 京都大学大学院法学研究科
Graduate School of Law, Kyoto University

図 1 Pentium 4 の命令発行後のパイプライン

含まれるため，それら全てを特定して無効化するのは難しい．そのため，Compaq Alpha 21264[2] では，後続の命令が発行される可能性があるサイクルに発行された全ての命令を無効化している．Intel Pentium 4[3] では，replay という選択的な無効化方法が使われているが，その詳細については未公開のままである．また，特定の投機を行う命令 1 つに対して，それに依存する後続の命令を再発行する機構を佐藤が提案しているが[4]，この方法を複数命令の投機に対して拡張すると，大容量の CAM が必要となる．

本論文では，RAM をベースとした，データ依存関係にある後続の命令を選択的に無効化する機構について提案する．提案する機構は，命令のデコード時に先行して実行中の全ての命令へのデータ依存を記したビット・ベクタを作成する．これにより，先行するどの命令に起因する投機的スケジューリングのミスが発生しても，その命令に依存する命令のみを無効化できる．

本論文の構成は以下の通りである．2 節では命令発行レイテンシの増大とそれにともなう投機的な命令スケジューリングの問題の詳細な説明を行う．3 節では提案するビット・ベクタを用いた選択的命令再発行機構について述べる．4 節では，選択的命令再発行と非選択的命令再発行のプロセッサ性能に与える影響のシミュレーション結果を示す．5 節では関連文献について述べ，最後に，6 節でまとめる．

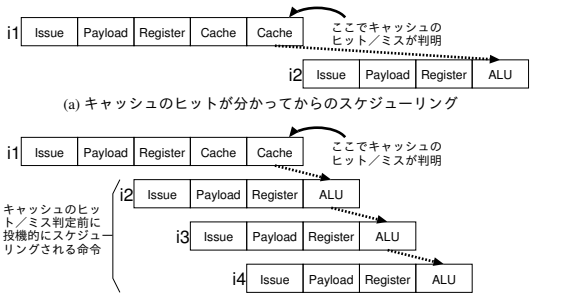
2. 命令発行レイテンシとそれによるペナルティ

本節では，まず，命令発行レイテンシ，および，その増大によって発生する，ロード命令に依存した命令の投機的スケジューリングについて説明する．その後，その投機ミスよって発生する再発行のペナルティ，および，2 種類の再発行のポリシーを説明する．

2.1 命令発行レイテンシと投機的スケジューリング

スーパスカラ・プロセッサの構成によっては，命令発行ステージの後にペイロード RAM(命令の情報を納めた RAM) 読み出しステージやレジスタ読み出しステージが存在するため，発行後の数サイクル後に実行ステージに入るものがある．この時，命令が発行されてから実行されるまでのレイテンシを命令発行レイテンシと呼ぶ．この命令発行レイテンシが大きなプロセッサの例として，Pentium 4[3] がある．Pentium 4 は**図 1** に示すようなパイプラインを採用しているため，命令発行レイテンシは 7 サイクルもある．近年のプロセッサでは，配線遅延の影響の増大による命令スケジュールのパイプライン化，ペイロード RAM やレジスタ・ファイルのアクセスのマルチ・サイクル化等により，命令発行レイテンシは増加する傾向にある．

このような命令発行レイテンシの大きなプロセッサにおいてデータ依存関係にある命令列のスケジューリングを行う場合，先行する命令が実行ステージに入る前に後続の命令を発行しなくてはならない．この時，先行する命令の実行レイテンシが一定であれば何の問題もない．後続の命令は，先行する命令が発行された後，先行する命令の実行レイテンシが経過した後に発行すれば良い．しかし，ロード命令の場合はキャッシュのヒット／ミスによってレイテンシが異なるため，それを考慮したスケジューリングを行わなくてはならない．



(a) キャッシュのヒットが分かってからのスケジューリング

(b) キャッシュがヒットすると仮定した投機的なスケジューリング

図 2 命令発行レイテンシの長いプロセッサにおけるロード命令に依存する命令のスケジューリング

このロード命令に依存する命令のスケジューリングの方法は，大きく分けて 2 つある．1 つは，**図 2**(a) に示すように，キャッシュのヒット／ミスが判明してから後続の命令の発行を行うスケジューリングである．このスケジューリングは確実ではあるが，図から分かるように，データ依存関係にある処理を連続して行えず，命令スループットを大きく低下させる．もう 1 つの方法は，図 2(b) のように，キャッシュにヒットすると仮定して後続の命令を発行する投機的なスケジューリングである．このスケジューリングでは，データ依存関係にある処理を連続して実行できるが，キャッシュ・ミスがあった場合，後続の命令は処理すべきデータがないため，実行ステージにたどりついても実行を開始できない．そのため，投機ミスからの回復処理が必要となる．

2.2 投機的スケジューリングのミスからの回復

前節の投機的スケジューリングのミスからの回復には，大きく分けて 2 つの方法が考えられる．1 つは，ロード命令の結果が来るまでロード命令に依存している命令のあるパイプラインをストールするものであり，もう 1 つは，ロード命令に依存している命令の再スケジューリングを行うものである．

図 2(b) の i3,i4 のように，この回復処理が必要となる命令はロード命令の結果を直接使う命令のみならず，間接的に使う命令も含まれる．そのため，1 つ目のストールさせる方法の場合，多数のパイプラインがストールすることが予想される．近年では，キャッシュ・ミス・レイテンシも配線遅延の増大によって増加する傾向にあるため，ストールによって依存関係のない命令まで長時間ストールさせるのは好ましくない．よって，2 つ目の再スケジューリングを行う方式の方が好ましいと考えられる．

この再スケジューリングを行う場合，再スケジューリング対象の選択方法でまた 2 つの手法が考えられる．1 つ目の手法は，ロード命令に依存した命令のみを再発行する手法である．この手法は再発行による性能低下を最小限に押

さえれる．しかし，発行された命令の中から，図 2(b) の i3,i4 のようにロード命令に間接的に依存している命令も捜し出し，再スケジューリングを行うのは難しい．そこで，2 つ目の手法として，再スケジューリング対象となる命令が発行されたと考えられるサイクル中に発行された，全ての命令の再スケジューリングを行う方法が考えられる．前者の商用プロセッサの採用例として，Pentium 4 の replay が挙げられる[3]．しかし，その機構の詳細は公開されていない．また，後者の商用プロセッサの採用例としては，Alpha 21264 が挙げられる．文献 2) によると，Alpha 21264 はロード命令に依存する命令の投機ミス時に，ロード命令に依存した命令が発行された可能性のある 2 サイクルに発行された全ての命令の再スケジューリングを行う．

3. ビット・ベクタを利用した選択的命令再発行

本節では，提案するビット・ベクタを利用した選択的命令再発行について述べる．まず，3.1 節で動作の概要を，ビット・ベクタの利用方法を中心として述べる．その後，3.2 節で各パイプライン・ステージでビット・ベクタの操作に必要となる回路構成と，その動作について説明する．

3.1 データ依存関係を示すビット・ベクタの概要

提案する機構では，命令ウィンドウの各エントリ，および，発行後に命令が無効化／再発行される可能性のあるステージまでの間のパイプライン・レジスタに，命令ウィンドウのエントリ数と同じサイズのビット・ベクタを持たせる．このビット・ベクタの各ビットは，命令ウィンドウの当該エントリ番号の命令にデータ依存があることを示す．このビット・ベクタによって，1 つ前のデータ依存先のみならず，先行する全ての命令に対するデータ依存を示す．

このビット・ベクタの生成とその利用方法について，i0 → i1 → i5 と i3 → i5 の 2 つのデータ依存関係を持つ，以下の命令列を使って説明する．ビット・ベクタの生成アルゴリズムを簡単にするため，データ依存のある命令ウィンドウのエントリ番号の他に，自身の命令ウィンドウのエントリ番号に対応するビットにも 1 を立てることとする．

命令番号	命令	ビット・ベクタ
i0:	R1 ← load(R2)	000001
i1:	R4 ← R1 + R6	000011
i2:	R2 ← R5 + R3	000100
i3:	R7 ← load(R8)	001000
i4:	R8 ← R5 + R9	010000
i5:	R5 ← R4 + R7	101011

i0 は，先行する命令に対するデータ依存がないので，自身のエントリ番号に対応するビットが 1 となる．i1 は，i0 に対してデータ依存があるため，自身のエントリ番号に対応するビットと i0 に対応するエントリ番号に対応するビットが 1 となる．このビット・ベクタの作成は，自身のエントリ番号に対応するビットを 1 にしたビット・ベクタと，i0 のビット・ベクタの OR を取ることで得ることができる．i2,i3,i4 は i0 と同様に先行する命令にデータ依存がないため，i0 と同様に自身のエントリ番号に対応するビットのみが 1 となっている．i5 は i1 と i3 に対してデータ依存関係があり，さらに，i1 を通して i0 にもデータ依存関係がある．よって，自身のエントリ番号に対応するビットと，i0,i1,i3 のエントリ番号に対応するビットを 1 とする必要がある．これは，i1 のビット・ベクタ，i3 のビット・ベクタの OR を取り，それに，自身のエントリ番号に対応するビットを 1 としたビット・ベクタの OR を取ることで得ることができる．このビット・ベクタを用いることにより，あるロード命令がキャッシュ・ミスを起こした場合，その命令のエントリ番号に対応するビットが 1 のビット・ベクタを持つ命令を無効化することで，当該命令とそれに依存する全ての後続命令を選択的に無効化できる．

上記のビット・ベクタは，命令が命令ウィンドウに入る前に生成する必要がある．そのため，レジスタ・マップ表にビット・ベクタのコピーを持ち，レジスタ・リネーミングと並行してビット・ベクタを生成する．生成されたビット・ベクタは命令ウィンドウに付随する，**再発行行列表 (RIMT: Re-Issue Matrix Table)** に格納され，命令ウィンドウに対する命令の無効化／再発行信号の生成に用いられる．この RIMT は後の節で説明するように，命令スケジューラの一種である依存行列表[5] と同様の RAM をベースとした構造である．命令の命令ウィンドウへの書き込み時には，命令が書き込まれた命令ウィンドウのエントリに対応する RIMT の行にビット・ベクタを書く．命令を無効化する場合，その起点となる命令を示すビット列を生成し，そのビット列の中の 1 に対応する列を読み出す．RIMT より読み出された列方向のビット列のうち，1 となっているエントリに対応する命令ウィンドウのエントリの ready フラグや発行済フラグを取り消す．命令ウィンドウ以降のパイプラインでは，各パイプライン・レジスタにビット・ベクタを保持するエントリと，無効化の起点となる命令を示すビット列との比較器が置かれている．無効化の起点となる命令をを示すビット列と比較のにおいて，1 つでも一致があった命令は，次段のパイプライン・レジスタに結果ではなく，NOP を書くこととする．

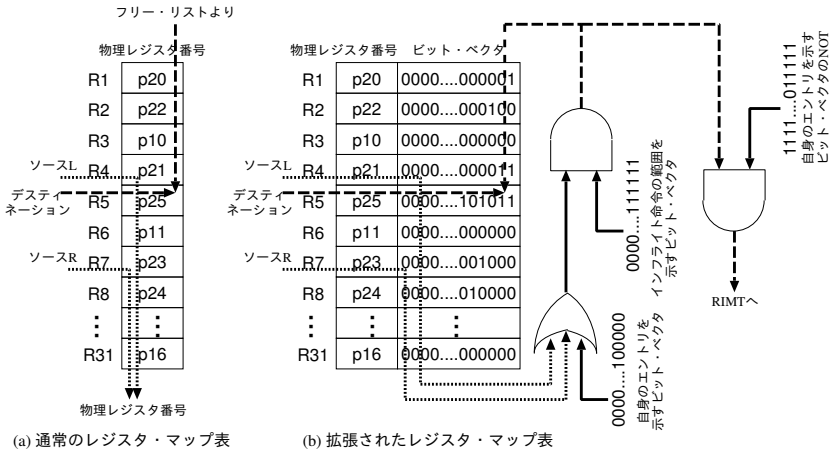
なお，上記ではアルゴリズムを簡単にするために，ビット・ベクタの自身に対応するエントリにも 1 を立てるとしているが，これでは，投機的ミスの原因となった命令まで再発行してしまう．そのため，RIMT に書き込む段階で自身に対応するビットは 0 とする．

3.2 各ステージにおける実装と動作

3.2.1 レジスタ・リネーミング

図 3 に従来のレジスタ・マップ表と拡張されたレジスタ・マップ表を示す．拡張されたマップ表では，ビット・ベクタを保持するエントリが追加されており，その論理レジスタに結果を出力する命令のビット・ベクタが格納されている．図 3 はまた，リネーミング・ステージにおける，マップ表の更新の例も示している．例には，3.1 節に示した命令列の i5 に対する更新を用いている．

図 3(a) では，従来からの役割である，論理レジスタ番号から物理レジスタ番号への変換に関する更新を示してある．細かい破線はソース論理レジスタ番号に対する物理レジスタ番号の読み出しを，粗い破線はデスティネーション



(a) 通常のレジスタ・マップ表　(b) 拡張されたレジスタ・マップ表

図 3　レジスタ・マップ表の拡張

論理レジスタに対してフリー・リストから得られた物理レジスタ番号を書き込む動作を示す．図では 2 つの動作は並列に行われているように見えるが，同一の論理レジスタ番号がデスティネーション側とソース側に現れた場合のリネームを考えると，先に読み出しを行った後，書き込みを行う必要がある．

図 3(b) では，提案機構のビット・ベクタの更新を示している．細かい破線，粗い破線の意味は図 3(a) と同じである．右下の OR ゲートによって，2 つのソースのビット・ベクタと自身を示すビット・ベクタの OR を取る点は，3.1 節で述べた通りである．その後段にある AND ゲートは，依存先をインフライトな命令に制限するためのものである．これは，すでに実行完了している命令によって無効化が行われることはないため，その依存の記述を除くためのものである．また，命令ウィンドウのエントリは循環して利用されるため，このような形で前にそのエントリを占めていた命令に対する依存情報を取り除く必要があるためでもある．生成されたビット・ベクタは，デスティネーション論理レジスタに対応するエントリに書き込まれる．また，自身への依存を示すビットを省いた後，RIMT に書き込まれる．

拡張されたマップ表では，読み出した値に論理演算を行った後に書き込みを行う必要があるため，従来のマップ表よりもクリティカル・パスが長くなる．そのため，リネーム・ステージを複数のステージに分割する必要が出てくる可能性がある．しかし，マルチ・サイクルのリネーミングはすでに Pentium 4 で実現されており[3)]，実装はそれほど困難ではないと考えられる．

3.2.2　再発行行列表 (RIMT)

図 4 に RIMT の概要を示す．図では，3.1 節に示した命令列のビット・ベクタを例として納めてある (ただし，自身を示すビットは立っていない)．リネーム・ステージで生成したビット・ベクタは，RIMT の行方向に書き込まれる．あるロード命令がキャッシュ・ミスを起こしてそれに依存する命令を無効化／再発行を行わなければならない場合，列方向にビット列を読み出し，1 となっているエントリに対応する命令に対して無効化を行う．例えば，エントリ 1 に格納されているロード命令 i0 がキャッシュ・ミスを起こした場合，エントリ 1,5 に入っている i1,i5 が無効化される．

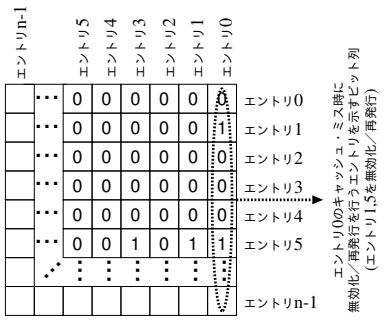


図 4　再発行行列表 (RIMT)

図 5 に RIMT の内部構造の概要を，**図 6** に RIMT の 1bit セルを示す．RIMT では，リネーム・ステージで生成されたビット・ベクタの読み書き時には行方向に書き込み，無効化対象命令を示すビット・ベクタの読み出し時には列方向に読み出すため，図 5 のように，各々のビット・ラインの方向が 90 度異なる．また，行方向の読み書きポートは通常の RAM と同じ構成であるが，列方向の読み出しポートは通常の RAM とは異なる．まず，読み出し時の入力は投機ミスの原因となった命令を示すビット列であるため，ビット・ラインのデコーダは不要である．さらに，メモリ・セルにビット・ラインを直接接続する構成にすると，1 クロック・サイクル中に複数の投機ミスがあった場合，値が 0 のセルと 1 のセルがビット・ラインに接続される可能性がある．そのため，図 6 のように，読み出しポートのビット・ラインはプルダウン・スタックによってセルの値が 1 の時に放電する構成とし，センス・アンプの出力を，放電された場合に 1 を出力する構成とする．

3.2.3　発行後のステージ

命令ウィンドウから発行された命令は，RIMT より読み出されたビット・ベクタを伴ってステージを進む．このビット・ベクタは，各ステージにおいて，投機ミスの原因となった命令を示すビット列と比較され，一致するビット

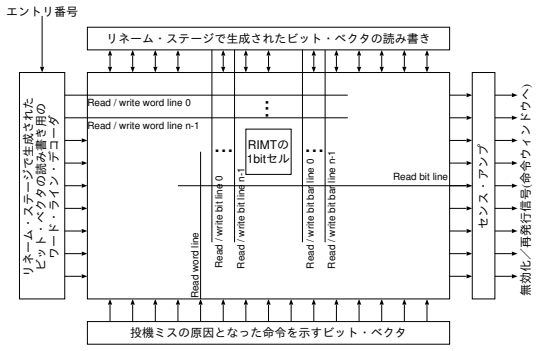


図 5 RIMT の内部構造の概要

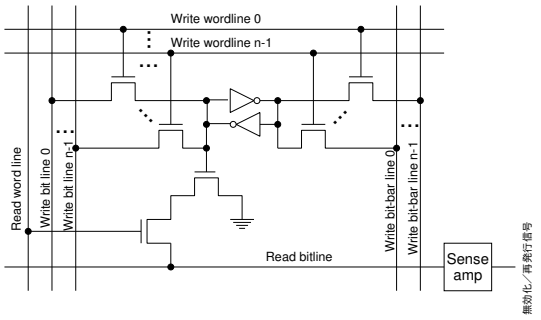


図 6 RIMT の 1bit セルの構成

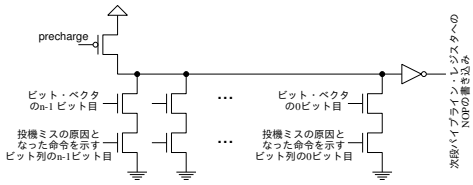


図 7 パイプライン中のビット・ベクタと無効化対象を示すビット列の比較器

があった場合，実行中の命令を無効化するために，次段のパイプライン・レジスタに NOP を書き込む指示を出す．比較を行うビット・ベクタは長く，通常の比較器では比較に時間がかかることが考えられる．そのため，**図 7** のような，ダイナミック回路で構成した比較器を用いることを考えている．

4. 評　　価

この節では，投機からの回復方式の違いによる，IP と再発行された命令数の差異を示す．

4.1 評 価 環 境

SimpleScalar Tool Set[6] 中の out-of-order 実行シミュレータを変更し，全命令の再発行と選択的再発行の測定を行った．命令セットは SimpleScalar PISA である．ベンチマーク・プログラムとして，SPEC2000 のうち，int より 8 本，fp より 9 本を用いた．入力は train もしくは ref を用い，最初の 1G 命令をスキップした後，1.5G 命令を測定に用いた．**表 1** に，シミュレーションにおいて仮定したプロセッサの構成を示す．

4.2 IPC の低下

命令発行レイテンシの増加によって，ロード命令に依存する命令の投機的スケジューリングを行うサイクル数が変化した場合の IPC のベンチマーク平均を**図 8** と**図 9** に示す．グラフの横軸は投機的スケジューリングを行うサイクル数であり，縦軸は IPC である．凡例ごとの 2 本の棒グラフはそれぞれ，投機ミス時に，投機的スケジューリングを行うサイクルに発行された全ての命令を無効化を行う場合と，ロード命令に依存する命令のみを無効化を行う場合である．また，グラフを横切る破線は，投機的スケジューリングが不要な場合の IPC を示している．

図に示されるように，選択的な無効化の方が IPC の低下ははるかに少ない．Pentium 4 と同様に，投機的スケジューリングを行うサイクル数が 7 サイクルの場合，全命令の無効化は IPC を int で 5.3%，fp で 6.2%低下させる．それに対し，選択的な無効化を行った場合，IPC の低下は int で 0.4%，fp で 1.0%に押さえられる．

表 1 プロセッサの構成

プロセッサ・コア	発行幅 8, RUU 128 エントリ, LSQ 64 エントリ, int ALU 8, int mult/div 4, fp ALU 8, fp mult/div 4, メモリ・ポート 8
分岐予測	PHT 8K エントリ／履歴長 6 の gshare, BTB 2K エントリ, RAS 16 エントリ
L1 I-キャッシュ L1 D-キャッシュ	64KB/32B ライン/2-way ヒット・レイテンシ 3 サイクル
L2 統合キャッシュ	2MB/64B ライン/4-way ヒット・レイテンシ 24 サイクル
メモリ	初期参照 128 サイクル, 転送間隔 2 サイクル
TLB	命令 16 エントリ, データ 32 エントリ ミス・レイテンシ 134 サイクル

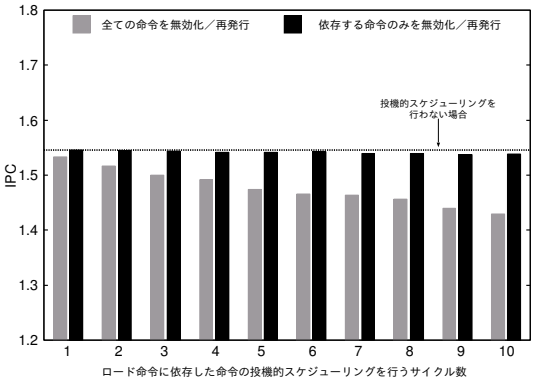


図 8 投機的スケジューリングを行うサイクル数と IPC の低下 (SPECint2000)

4.3 再発行された命令の割合

命令発行レイテンシの増加によって，ロード命令に依存する命令の投機的スケジューリングを行うサイクル数が変化した場合に再発行された命令数のベンチマーク平均を**図 10** と**図 11** に示す．グラフの縦軸は再発行された命令数であり，グラフの横軸，凡例ごとの 2 本の棒グラフは図 8，図 9 と同様である．

図に示されるように，選択的な無効化の方が再発行される命令数がはるかに少ない．投機的スケジューリングを行うサイクル数が 7 サイクルの場合，選択的な無効化において再発行される命令数は，全命令の無効化を行う場合と比較して，int で 6.2%，fp で 2.8%に過ぎない．このように，

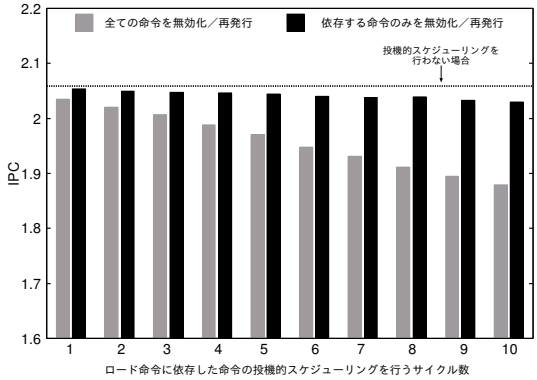


図 9 投機的スケジューリングを行うサイクル数と IPC の低下 (SPECfp2000)

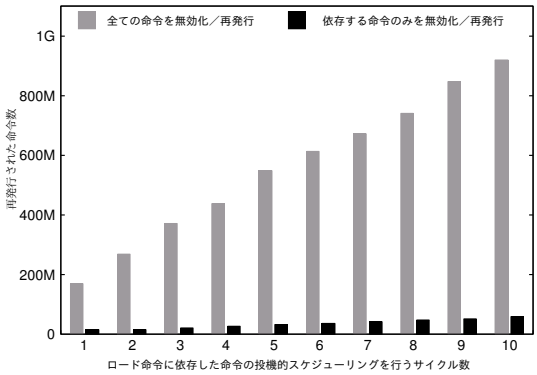


図 10 投機的スケジューリングを行うサイクル数と再発行される命令数 (SPEint2000)

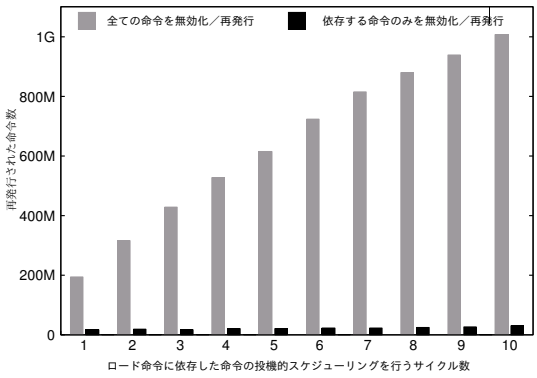


図 11 投機的スケジューリングを行うサイクル数と再発行される命令数 (SPECfp2000)

無効化される命令を削減することは，近年のプロセッサで大きな課題となっている，消費電力の削減にも効果があると考えられる．

5. 関 連 研 究

福田らは，0 次キャッシュとも言えるライン・バッファをプロセッサに搭載した場合における，ロード命令に依存する命令の投機的命令スケジュール・ミスに関連する研究を行った[7]．彼らは，ライン・バッファを搭載した場合は，投機ミスによる再発行が頻発するため，ライン・バッファ・ヒット／ミス予測器を用いないと，ライン・バッファは性能向上に結びつかないことを示した．

佐藤は，ロード命令のアドレスを値予測器で予測して投機的なロード行う場合，提案する命令再発行機構を用いることを投機実行の効果が改善されることを示した[4]．彼の提案する再発行機構では，再発行ビットを命令スケジューラの CAM に追加し，その伝播によってデータ依存関係にある後続命令のみを選択的に無効化する．文献では，再発行ビットは 1 ビットのみであるが，複数ビットを準備することによって複数の投機に容易に対応できると考えられる．しかし，この拡張は大容量の CAM を必要とするため，面積コストや消費電力コストの増加を招くと考えられる．

6. ま と め

将来のプロセッサでは，配線遅延の増大により，命令発行レンテンシが増大する可能性がある．この命令発行レイテンシの増大はロード命令に依存する命令の投機的スケジューリングを増やし，投機ミスによる命令の無効化／再発行を増やすことになる．

本論文では，この投機ミスからの回復方法のうち，投機ミスの原因となった命令に依存する命令のみを特定して無効化を行う機構の実装方法の 1 つを提案した．選択的再発行と非選択的再発行をシミュレーションによって評価した結果，選択的再発行は再発行を行う命令を大幅に削減し，IPC の低下を押さえることを示した．

謝辞 本研究の一部は日本学術振興会科学研究費補助金基盤研究 S(課題番号 16100001) による．

参 考 文 献

1) Agarwal, V., Hrishikesh, M. S., Keckler, S. W. and Burger, D.: Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures, *ISCA-28*, pp. 248–259 (2001).
2) Kessler, R.: The Alpha 21264 Microprocessor, *IEEE Micro*, Vol. 19, No. 2, pp. 22–36 (1999).
3) Hinton, G., Sager, D., Upton, M., Boggs, D., Carmeand, D., Kyker, A. and Roussel, P.: The Microarchitecture of the Pentium 4 Processor, Technical Report Vol. 5, Issue 1, Intel Terchnology Journal (2001).
4) 佐藤寿倫: 命令再発行機構によるデータアドレス予測に基づく投機実行の効果改善, 情報処理学会論文誌, Vol. 40, No. 5, pp. 2093–2108 (1999).
5) 五島正裕, 西野賢悟, 小西将人, 中島康彦, 森眞一郎, 北村俊明, 富田眞治: 行列に基づく Out-of-Order スケジューリング方式の評価, 情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム, Vol. 43, No. SIG 6(HPS 5), pp. 13–23 (2002).
6) Burger, D. and Austin, T. M.: The SimpleScalar Tool Set, Version 2.0, Technical Report CS-TR-97-1342, University of Wisconsin-Madison Computer Sciences Dept. (1997).
7) 福田祥貴, 片山清和, 島田俊夫: ライン・バッファ・ヒット／ミス予測を利用した動的命令スケジューリングの高精度化手法, SACSIS2003, pp. 227–234 (2003).