2ホップアドレス名前替えを用いたロード命令の投機的実行

佐藤寿倫†

アドレス名前替えを用いたロード命令の投機的実行を提案する．データアドレスをストア命令アドレスに，ストア命令アドレスをロード命令アドレスに名前替えすることで，データアドレスを計算する前にロード命令が読み出そうとしているデータの値を予測する．同じデータにアクセスするロード命令とストア命令を関連づけ，ストア命令がライトするデータをロード命令に受け渡す．データの受渡しにはStore-Indexed Value Table（SIVT）とLoad-Indexed Store Table（LIST）とを提案する．LISTによりロード命令アドレスはストア命令アドレスに変換され，変換されたストア命令アドレスを用いてSIVTを参照することで所望のデータが得られる．ロード命令とストア命令を関連づけるためにData-Indexed Store Table（DIST）を提案する．SPECint95を用いたシミュレーションにより，命令レベル並列度を平均3.4%，最大9.4%向上できることが確認できている．

Load Value Prediction via Two-hop Reference Address Renaming

TOSHINORI SATO†

In this paper, we present an alternative implementation of load value prediction. Load values are proposed to be predicted using store information. A pair of a load and a store instructions referring a same memory location is linked and the stored value is forwarded to the load instruction. For date forwarding, *store-indexed value table* (*SIVT*) and *load-indexed store table* (*LIST*) are proposed. By indexing the LIST by a load instruction address, the load instruction address is translated to a store instruction address, and the SIVT indexed by the translated address supplies a data value. In order to link the store and load instructions, *data-indexed store table* (*DIST*) is proposed. From the experimental evaluation, we have found that the instruction level parallelism is increased by 3.4% on average with a maximum of 9.4% when this load value prediction mechanism is implemented.

1. まえがき

プロセッサの性能向上を妨げる要因の1つに命令間の依存関係の問題がある．依存関係には制御依存関係（control dependence），資源依存関係（name dependence），そしてデータ依存関係（data dependence）がある．制御依存関係には多くの研究があり，分岐予測や投機的実行により依存関係の解消が試みられている．資源依存関係はレジスタの数というハードウェア資源の不足により生じる．これはレジスタ名前替えを施すことにより解消可能である．しかしデータ依存関係は「真の依存関係」とも称されるように解消できる方法はない．つまり，データ依存関係は，命令レベル並列度（instruction level parallelism: ILP）を向上する妨げとなる非常に深刻な問題である．データ依存関係には2つのタイプが存在する．1つはレジスタを介した依存関係であり，もう1つはメモリを介した依存関係である．本稿では後者に注目する．メモリを介したデータ依存関係は，参照されるデータのアドレスがプログラムの実行時にならないと決定されないために生じる．あるストア命令に続くロードあるいはストア命令は，先行するストア命令が完了しなければ実行できない．これが曖昧なメモリ参照の問題であり，ILPを引き出す際の大きな妨げとなっている．最近Lipasti[5]により，ロードされるデータ値の予測が提案された．データアドレスを計算しないでデータを獲得できるので，曖昧なメモリ参照の問題は生じない．つまり，ストア命令の実行が待機中であっても，ロード命令はそのストア命令を追い越すことができる．これにより命令スケジューリングの自由度が向上し，プロセッサ性能の向上が期待できる．さらに，データキャッシュミスが生じたとしても，ロード命令はキャッシュミスの状態にかかわらず実行可能なので，キャッシュミスによるペナルティを隠蔽でき，やはりプロセッサ性能の向上が期待できる．本稿では，Lipastiの提案

† 株式会社東芝 セミコンダクター社 マイクロエレクトロニクス技術研究所
Toshiba Microelectronics Engineering Laboratory

とは異なるロードデータ値予測手法を検討する．

データ依存関係の投機実行を実現するには，以下の2つの機構が必要である．1つは各命令の実行結果を予測するデータ値予測機構である．もう1つは，データ値の予測に失敗したときにプロセッサを投機状態から正しい状態に回復させるための機構である．本稿では前者を検討し，アドレス名前替えを実現する単純なハードウェアを用いたデータ値予測方法を提案する．データアドレスをストア命令アドレスに，ストア命令アドレスをロード命令アドレスに名前替えすることで，ロード命令が読み出そうとしているデータの値を予測する．同じデータにアクセスするロード命令とストア命令を関連づけ，ストア命令がライトするデータをロード命令に受け渡す．データを受け渡すために *Store-Indexed Value Table*（SIVT）と *Load-Indexed Store Table*（LIST）とを提案する．LISTによりロード命令アドレスはストア命令アドレスに変換され，変換されたストア命令アドレスを用いてSIVTを参照することで所望のデータが得られる．ロード命令とストア命令を関連づけるためには *Data-Indexed Store Table*（DIST）を提案する．予測されたデータ値を用いて，ロード命令に後続する命令が投機実行される．

本稿は以下の構成からなる．ロードデータ値予測手法を2章で説明し，3章でその一実装例を紹介する．4章では提案手法の評価環境について述べる．データ依存投機実行の効果は5章で評価する．6章では関連研究と本稿での提案とを比較する．最後に7章でまとめる．

2. ロードデータ値予測法

本章ではロードされるデータ値の予測方法について述べる．本手法はアドレス名前替えに基づいている．実現方法は次章で説明する．

2.1 1ホップアドレス名前替え

データアドレスをあるタグに名前替えできれば，データアドレスを計算しなくても，タグを用いてデータを読み出すことが可能になる．メインメモリとは別に用意された記憶領域に，あるタグをインデックスとして記憶しておく．ロード命令の実行時にそのタグを用いて上述の記憶領域を参照すれば，データアドレスを計算しなくてもデータを獲得できる．我々はこの方法を1ホップアドレス名前替え（1-hop reference address renaming: RAR-H1）と呼んでいる．**図1** にRAR-H1の例を示す．データアドレス `addr` の代わりにあるタグ `tag` を用いてメインメモリとは別の記憶領域 `temporal memory` にデータを記憶しておく．ロード命令実行時にはデータアドレスが計算される前にそのタグ `tag` を用いて `temporal memory` を参照する．したがって，データアドレスを計算してメインメモリにアクセスする以前に `temporal memory` からデータが獲得できる．

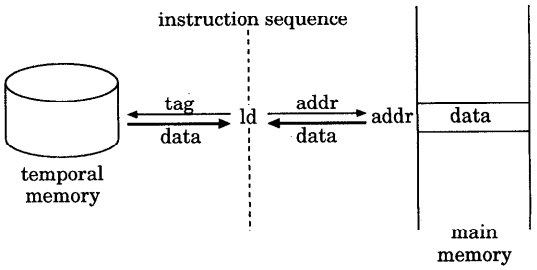


図1 1ホップアドレス名前替え
Fig. 1 1-hop reference address renaming.

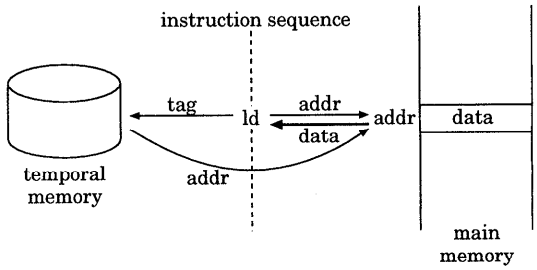


図2 データアドレス予測
Fig. 2 Data address prediciton.

タグにはいろいろな候補があると思われるが，実装に適しているものとしてロード命令アドレスが考えられる．タグにロード命令アドレスを採用した場合，図1の例はLipasti[5]の方法を表していることになる．ロード命令アドレスを用いて `temporal memory` を参照するとデータが得られる．`temporal memory` は，前回このロード命令がメインメモリから読み出したデータを保存しており，そのデータを供給する．すなわちLipastiの方法は，データアドレスをロード命令アドレスに名前替えしたRAR-H1といえる．

データアドレスの予測手法もRAR-H1と見なすことができる．**図2** にデータアドレス予測手法の例を示す．計算されたデータアドレスを用いる代わりにタグ `tag` を用いて `temporal memory` にアクセスする．`temporal memory` はデータアドレスを（予測して）返してくるので，そのアドレスを用いてメインメモリにアクセスしデータを獲得する．メインメモリにアクセスする必要はあるが，データアドレスが計算される前にデータを獲得することが可能である．したがって，RAR-H1と見なすことが可能である．この場合もタグとして様々な候補があるが，たとえばロード命令ア

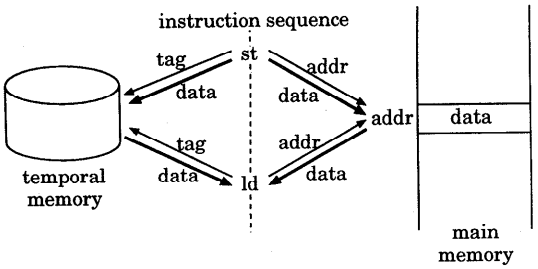


図3 ストア命令情報を用いたRAR-H1
Fig. 3 RAR-H1 using store information.

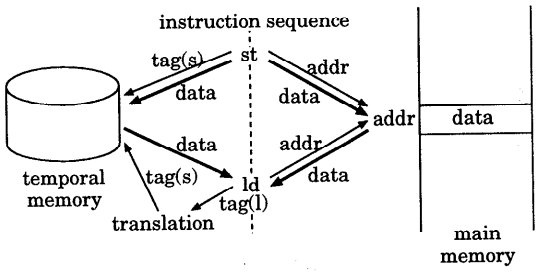


図4 2ホップアドレス名前替え
Fig. 4 2-hop reference address renaming.

ドレス[2]を用いることができる．

2.2 2ホップアドレス名前替え

前節で説明したRAR-H1では，ロードされるデータ値の予測にはロード命令の情報のみしか用いられていない．ロード命令によって読み出される値はストア命令によって書き込まれた値であるので，ストア命令の情報も利用できれば予測精度の向上が期待できる．すなわち，ストア命令がデータアドレスを名前替えしてタグを定義し，同じメモリ領域を参照するロード命令がその同じタグを用いてデータを参照できれば，データアドレスを計算することなくストア命令からロード命令にデータを受け渡すことができる．この例を**図3**に示す．この場合ストア命令とロード命令は同じタグを用いており，データアドレスからタグへのRAR-H1を行っているにすぎない．ここで問題となるのは，いかにしてストア命令が定義したタグをロード命令に知らせるかということである．我々はそうする代わりに，ストア命令とロード命令が独自にタグを定義する方法を選んだ．何らかの方法でロード命令が定義したタグをストア命令が定義したタグに変換できれば，実質的にストア命令とロード命令が同じタグを使用していることになる．これを**図4**に示す．ロード命令が定義したタグ `tag(l)` はいったんストア命令の定義したタグ `tag(s)` に変換され，変換されたタグを用いて記憶領域 `temporal memory` にアクセスする．

我々はこの方法を2ホップアドレス名前替え（2-hop reference address renaming: RAR-H2）と呼んでいる☆．データアドレスがストア命令の定義するタグに名前替えされるだけであればRAR-H1であるが，さらにロード命令の定義するタグに名前替えされるので，このように呼んでいる．

ここでも命令アドレスをタグとして採用すると，以下のようになる．データをストアする際には，データアドレスをストア命令アドレスに名前替えして，メインメモリとは別の記憶領域 `temporal memory` にもデータを保存する．したがって，ストア命令アドレスを用いればデータにアクセス可能である．さらに，ストア命令アドレスをロード命令アドレスに名前替えする．これらにより，ロード命令アドレスでデータにアクセス可能になる．すなわち，ロード命令アドレスをストア命令アドレスに変換し，さらにストア命令アドレスをデータアドレスに変換することで，ロード命令アドレスを用いてデータにアクセス可能になる．

3. ロードデータ値予測機構

本章で，前章で説明したロードデータ値予測法の実装について検討する．ロードデータ値予測機構は以下の3つのテーブルを利用する．すなわち，*Store-Indexed Value Table*（SIVT），*Load-Indexed Store Tabel*（LIST），*Data-Indexed Store Table*（DIST）である．LISTとSIVTはストアされたデータをロード命令に受け渡す．DISTを用いてロード命令とストア命令を関連づける．続く3節で各テーブルを説明し，ロードデータ値予測機構を提案する．最後に，予測失敗時にプロセッサ状態を回復させる方法に触れる．

3.1 Store-Indexed Value Table

SIVTはストア命令アドレスからデータアドレスへの変換を行い，そのデータアドレスに保存されているデータを獲得するために用いられる．SIVTの各エントリにはデータ値が保存されており，それらはそのデータをライトしたストア命令の命令アドレスによってインデックスされている．したがって，あるストア命令アドレスを用いてSIVTを参照することで，そのストア命令が最後にライトしたデータの値を獲得できる．すなわち，データアドレスからストア命令アドレスへのRAR-H1を行っている．**図5**にダイレクトマップで構成されたSIVTの例を示す．SIVTはキャッシュメモリと同様の構成をしている．各エントリは，タグアドレスフィールド，データフィールド，バリッ

☆ 文献7)では2レベルアドレス名前替えと呼んでいたが，2レベル適応型分岐予測機構との混乱を避けるため，2ホップアドレス名前替えと呼びかえることとした．

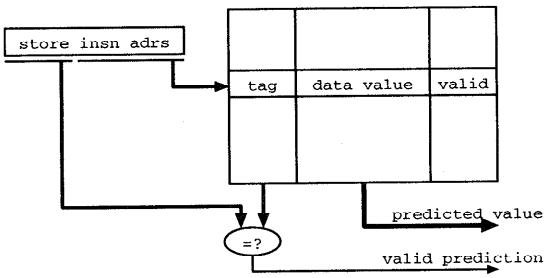


図5 Store-Indexed Value Table
Fig. 5 Store-Indexed Value Table.

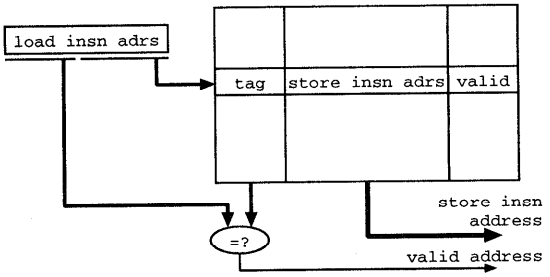


図6 Load-Indexed Store Table
Fig. 6 Load-Indexed Store Table.

ドビットから構成される．

3.2 Load-Indexed Store Table

LISTはロード命令アドレスからストア命令アドレスへの変換を行う．LISTの各エントリにはストア命令アドレスが保存されており，それらは同じメモリ領域を参照するロード命令の命令アドレスによってインデックスされている．したがって，あるロード命令アドレスを用いてLISTを参照することで，そのロード命令が読み出そうとしているデータを保存したストア命令の命令アドレスを獲得できる．すなわち，LISTはロード命令アドレスからストア命令アドレスへのRAR-H1を行っている．**図6**にダイレクトマップで構成されたLISTの実装例を示す．LISTはキャッシュメモリと同様の構成をしている．各エントリは，タグアドレスフィールド，ストア命令アドレスフィールド，バリッドビットから構成される．

3.3 Data-Indexed Store Table

DISTは1つのストア命令と複数のロード命令を関連づけるために用いられる．DISTの各エントリにはストア命令アドレスが保持されており，それらはそのストア命令が保存したデータのデータアドレスによってインデックスされている．したがって，あるロード命令が実行されるとき，そのロード命令が読み出そうとしているデータアドレスでDISTを参照することにより，そのデータをライトしたストア命令の命令アドレスを獲得できる．**図7**にダイレクトマップで構成されたDISTの例を示す．DISTもキャッシュメモリと同様の構成をしている．各エントリは，タグアドレスフィールド，ストア命令アドレスフィールド，バリッドビットから構成される．

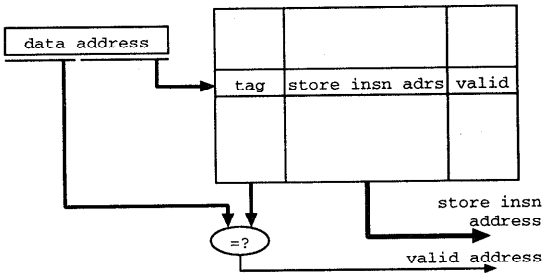


図7 Data-Indexed Store Table
Fig. 7 Data-Indexed Store Table.

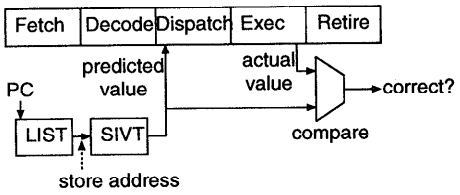


図8 パイプライン図
Fig. 8 Pipeline diagram.

3.4 ロードデータ値予測機構

図8に示す基本的なパイプラインを用いてデータ値予測機構を説明する．予測は命令のディスパッチまでに完了する．まず命令フェッチ時に命令アドレスを用いてLISTを参照する．命令がロード命令であれば，読み出そうとしているデータをライトしたストア命令アドレスが供給される．続いてデコード時に，供給されたストア命令アドレスを用いてSIVTを参照する．SIVTからはそのストア命令が保存したデータが供給される．命令がロード命令の場合にはSIVTから供給されたデータを用いて，投機的に後続の命令を実行する．LISTとSIVTは異なるパイプラインステージで参照されることに注意されたい．このため，プロセッサのサイクルタイムには悪影響を及ぼさない．

このとき予測の信頼性[3]を評価して，投機実行するかしないかを決定する．信頼性は以下のようにして決定される．LISTの各エントリに2ビット飽和型カウンタを用意する．データ値予測に成功したときにはこのカウンタを +1 増加させ，失敗したときには −1 減少させる．予測時にこのカウンタを参照し，その値が2以上のときには有効な予測と見なして投機的実行を

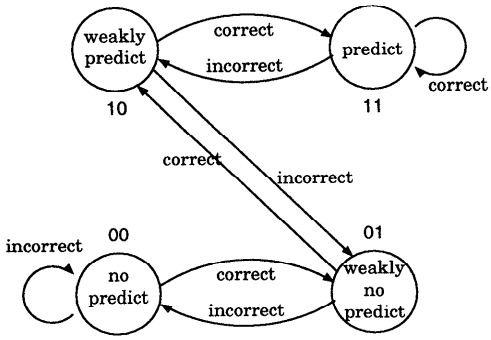


図9 信頼性のための2ビット飽和型カウンタ
Fig. 9 2-bit saturation counter.

行う．この2ビット飽和型カウンタは分岐予測表に用いられるそれと同様のものである．図9にカウンタを示す．

命令ウインドウには投機実行に用いた予測データも保存される．ロード命令がメモリからデータを読み出すと，その値と予測データの比較が行われる．もし両者が一致していれば投機実行は成功である．一致しなかった場合には，プロセッサの状態を投機前の状態に戻す必要がある．この操作はリオーダーバッファを用いれば容易に実現できる．レジスタ更新ユニット(register update unit: RUU)[9]を拡張して実現した命令ウインドウは3.5節で説明する．

続いてSIVT，DIST，LISTの各テーブルへの登録方法を説明する．SIVTの登録はストアされるデータが獲得されるとすぐに行われる．ストアデータアドレスは計算されている必要はない．このときにはストア命令アドレスとデータ値は揃っているので，ストア命令でインデックスされるエントリにデータが保存可能である．DISTの登録はストア命令がデータアドレスを計算するとすぐに行われる．このときにはデータアドレスとストア命令アドレスは揃っているので，データアドレスでインデックスされるエントリにストア命令アドレスを保存可能である．LISTの登録は図10に示すように行われる．まずロード命令の完了時にDISTを参照する．このときにはデータアドレスは計算されておりDISTの参照が可能である．DISTからはストア命令アドレスが供給される．したがって，ロード命令アドレスでインデックスされるLISTのエントリに，DISTから供給されたストア命令アドレスを登録することができる．

以上のようにして，DISTにより1つのストア命令と複数のロード命令が関連づけられ，その関係がLISTに保持される．ロード命令実行時にはLISTにより対応するストア命令を特定でき，さらにSIVTによりそのストア命令がライトしたデータを獲得できる．このようにして，ロード命令アドレスからストア命令アドレス，さらにデータアドレスの変換を行って，ロード命令が読み出そうとしているデータを予測し獲得できる．すなわち，データアドレスからストア命令アドレス，さらにロード命令アドレスへのRAR-H2を行うことで，データアドレスを計算しなくてもロード命令は必要なデータを獲得できる．

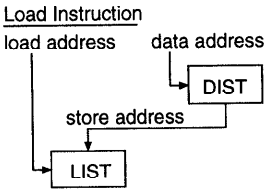


図10 LISTの登録操作
Fig. 10 Updating scheme for LIST.

3.5 予測失敗からの回復

データ値の予測に失敗し，その間違った値を用いて後続の命令が投機実行された場合には，プロセッサの状態を予測前の状態に回復させる必要がある．回復には，命令破棄による方法と，命令再発行による方法が考えられる．

命令破棄による方法は以下のとおりである．分岐予測とデータ値予測の類似性に着目すると，プロセッサ状態を回復させるためには，分岐予測の失敗からの回復に用いられているように，投機に失敗した命令に後続するすべての命令を破棄すればよいように思える．しかし，そのような機構を用いた場合，以下の2点が問題となる．まず，データ依存の投機失敗によるミスペナルティは分岐予測失敗によるそれと比べて非常に大きいことである．予測成功の判定はロード命令がデータをメモリから読み出すまで待たねばならず，特にデータキャッシュがミスした場合など，そのロード命令のフェッチから非常に遅れることになる．さらに，投機開始から予測の成功判定までの間に実行された命令のうち，投機に失敗した命令と依存関係にない命令までも破棄するため，有意義な演算結果も捨てられてしまう．

我々は上記の問題を検討し，再実行されるべき命令だけを選択し，それらを命令ウインドウ内部で再発行する機構をすでに提案している[8]．再実行されるべき命令は，予測に失敗した命令とデータ依存の関係にある命令であって，間違った予測値を用いて機能ユニットで実行された命令である．この命令再発行機構は，命令ウインドウ中でこれらの命令を選択的に再発行する．命令ウインドウ中で再発行されるために，これら

表1 プロセッサモデル
Table 1 Processor model.

命令フェッチ幅	8命令
分岐予測機構	512エントリ2ウエイ・セットアソシアティブBTB, gshare法, 12ビットBHR, PHT4096エントリ, デコードステージで投機的に更新, リターン・アドレス・スタック8エントリ, ミスペナルティ3サイクル
命令ウインドウ	命令キュー64エントリ, ロード・ストアキュー8エントリ
命令ディスパッチ幅	8命令
命令コミット幅	8命令
機能ユニット	5 iALU's, 1 iMUL/DIV, 4 Ld/St, 2 fALU's, 2 fMULs, 2 fDIV/SQRT's
レイテンシ(全/投入間隔)	iALU 1/1, iMUL 3/1, iDIV 35/35, Ld/St 2/1, fADD 2/1, fMUL 3/1, fDIV/SQRT 6/6
レジスタファイル	32ビット整数レジスタ32本, 32ビット浮動小数点レジスタ32本
命令キャッシュ	64K 4ウエイ・セットアソシアティブ, ライン幅32バイト, 4ポート, ミスペナルティ6サイクル
データキャッシュ	64K 4ウエイ・セットアソシアティブ, ライン幅32バイト, 4ポート, ライトバック, ノンブロッキングロード, ミスペナルティ6サイクル
二次キャッシュ	共用, 256K 4ウエイ・セットアソシアティブ, ライン幅64バイト, ミスペナルティ32サイクル

の再実行されるべき命令を再びフェッチする必要はない. すなわち, 投機に失敗した命令と依存関係にある命令だけを選択し, 投機実行に失敗した命令を命令ウインドウ内部で再発行することにより, 命令の破棄に基づく機構と比べてミスペナルティの削減を実現している.

4. 評価環境

本章では, シミュレーションに用いたプロセッサのモデルとベンチマークプログラムを紹介し, 提案手法の評価方法について説明する.

4.1 プロセッサモデル

提案手法の評価には, SimpleScalar ツールセット[1]を用いた. SimpleScalar/PISA アーキテクチャはMIPS アーキテクチャに基づいており, サイクルレベルのシミュレータがSPARCstation 上で動作する. 基本となるプロセッサモデルはアウト・オブ・オーダー実行を行うスーパースカラ・プロセッサである. アウト・オブ・オーダー実行を実現するためにはRUU を採用している. 文献4) での結果に基づいて, 基本モデルの構成は**表1** にまとめたとおりとした.

LIST, SIVT, DIST はダイレクトマップ方式とし, それぞれのエントリ数は4096 とした. 本稿の目的はLIST, SIVT, DIST の各テーブルの最適なエントリ数を導くことではなく, 2 ホップアドレス名前替えに基づくデータ依存の投機実行が有効であることを検証することである. したがって, LIST, SIVT, DIST の他の構成に関しては評価を行わない.

予測失敗時にプロセッサの状態を回復させる機構としては, 命令破棄と命令再発行を評価する.

4.2 ベンチマークプログラム

評価にはSPECint95 ベンチマークプログラムを用いた. 各プログラムの入力ファイルには, SPEC から配布されている **test** ファイルを使用した. **表2** にベンチマークとその入力ファイルをまとめる. ウイスコンシン大学が配布しているオブジェクトファイルを用いて実行した[1]. 各プログラムは終了まであるいは最初の1億命令を実行した.

表2 ベンチマークプログラム
Table 2 Benchmark programs.

プログラム	入力ファイル
099.go	null.in
124.m88ksim	ctl.in
126.gcc	cccp.i
129.compress	test.in
130.li	test.lsp
132.ijpeg	specmun.ppm
134.perl	primes.in
147.vortex	vortex.in

5. 評価結果

本章でシミュレーション結果を紹介する. まず予測精度を評価し, 続いて, 命令破棄および命令再発行を採用した場合のILP の向上について述べる. 向上の尺度にはサイクルあたりにコミットされた命令数(committed instructions per cycle: IPC) を用いた. 最後に, 命令の再発行によるオーバーヘッドを検証する.

5.1 予測精度

表3 の左半分に, ロードされるデータ値の予測精度をまとめる. ここで予測精度とは, 予測が実際に利用されたロード命令中で予測が当たったロード命令の割合を表している. したがって, データ値が予測されたにもかかわらず信頼性が低いために利用されなかったロード命令は含まれていない. 予測精度は非常に高く, 平均で79.0%, 最大93.1%である.

表3 の右半分には, LIST と SIVT のヒット率をま

表3 予測精度と各テーブルのヒット率
Table 3 Prediction accuracy and hit rates.

プログラム	予測精度（%）	ヒット率（%）	
		LIST	SIVT
099.go	49.39	84.6	96.8
124.m88ksim	91.04	63.3	96.7
126.gcc	77.84	61.0	97.6
129.compress	89.03	60.8	99.9
130.li	71.47	86.0	95.6
132.ijpeg	90.33	77.3	99.7
134.perl	69.82	81.5	96.3
147.vortex	93.10	70.5	95.9
平均	79.00	73.1	97.3

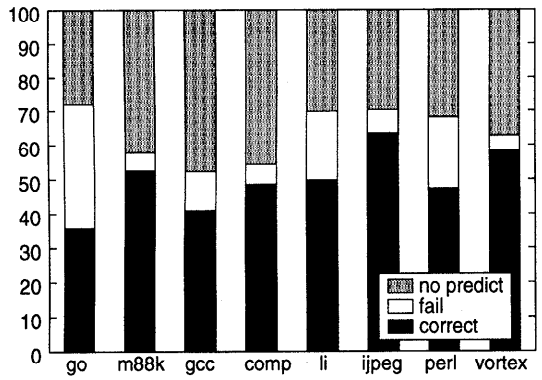


図11 データ値予測の割合（%）
Fig. 11 Performing rate of data prediction (%).

とめた．LISTのヒット率は，LISTを参照した全ロード命令のうちストア命令アドレスが得られた回数の割合である．SIVTのヒット率は，ストア命令アドレスが獲得できてSIVTを参照したロード命令のうちデータ値が得られた回数の割合である．表3から平均してLISTは73.1%，SIVTは97.3%のヒット率であることが分かる．単純に考えるとデータ値予測精度はLISTヒット率とSIVTヒット率の積で表され70%程度と予想されるが，実際にはそれよりも幾分高い精度を示している．これは信頼度を用いているためである．予測が当たると思われる場合にのみ予測されたデータを利用しているため，予測精度が高くなっている．

図11 にデータ値予測の割合を示す．各グラフは3つの領域に分けられており，上から順に，予測が行われなかった割合（灰），予測が誤った割合（白），予測が正しかった割合（黒）を表している．平均49.7%のロード命令が正しくデータを予測できている．このように，予測を刈り込むことで高い予測精度が得られている．

5.2 命令レベル並列度

図12 と**図14** に提案手法を用いた際のILPの変化を表す．図の縦軸は，投機実行モデルでのIPCの変化を基本モデルのIPCで割った割合である．したがって，値が大きいほどILPの向上が大きい．図12は予測に失敗した場合に命令を破棄する場合の結果であり，図14は命令再発行を行う場合の結果である．

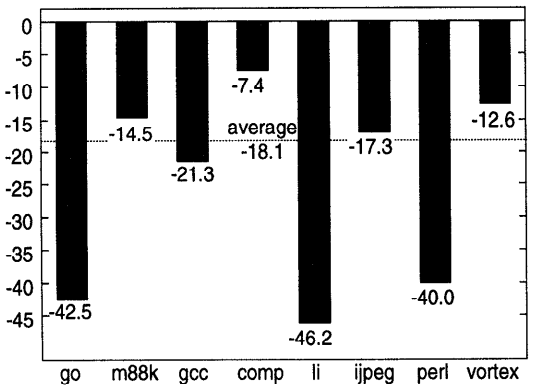


図12 命令レベル並列度の変化（命令破棄）（%）
Fig. 12 Processor performace improvement (squash) (%).

命令を破棄した場合には，すべてのベンチマークプログラムにおいて，投機実行を行わない場合よりもILPが低下している．原因はミスペナルティが大きいことである．予測の成功判定は，ロード命令がデータを読み出すまで待たされる．つまり，判定されるまでの間に非常に多くの命令がフェッチされ実行されている．データ値の予測に失敗したときに命令破棄を用いると，これらの命令フェッチおよび実行はすべて無駄になってしまい，ミスペナルティが非常に大きい．データキャッシュがミスした場合では，ミスペナルティはよりいっそう大きくなる．つまり，前節で紹介した高い予測精度であっても，予測ミスを犯したときのオーバヘッドが非常に大きいため，データ依存投機の効果が得られていないと考えられる．表3と図12を比べると，予測精度とILPの低下との間には相関があることが分かる．すなわち，予測精度の低いものほどILPの低下が著しい．

図13 に，破棄された命令を含む全実行命令数を，基本モデルと比較した場合の割合として示す．したがって基本モデルよりも実行された命令数が多い場合には100%を超えることになる．図12と図13とを見比べると，全実行命令の変化とILPの変化との間には相関があることが分かる．すなわち，全実行命令が増えるに従ってILPの低下が著しくなっている．

一方，図14より，命令再発行を行う場合では平均で3.4%，最大9.4%のILPの向上が確認できる．これにより，ロード値予測によるデータ依存投機実行の

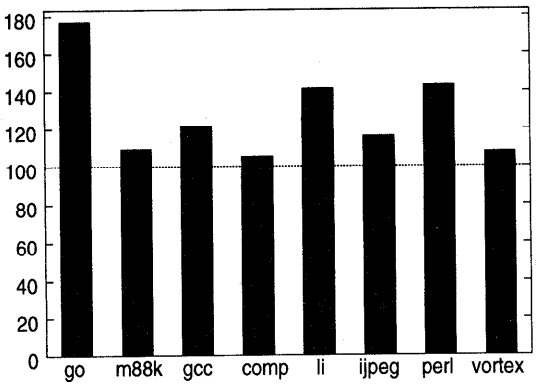


図13 全実行命令の変化（命令破棄）（%）
Fig. 13 Total instructions (squash) (%).

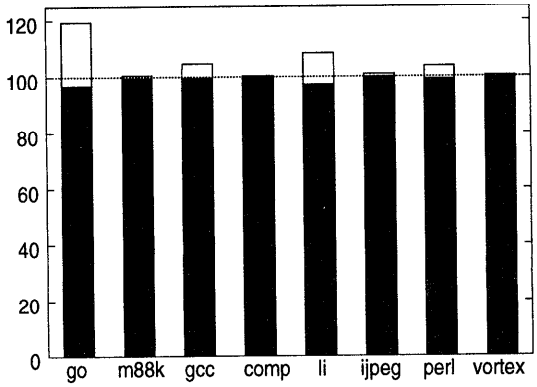


図15 全実行命令と再発行された命令の割合（%）
Fig. 15 Total and reissued instructions (%).

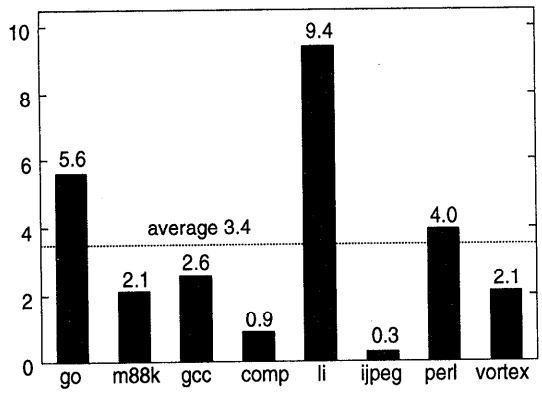


図14 命令レベル並列度の変化（命令再発行）（%）
Fig. 14 Processor performace improvement (reissue) (%).

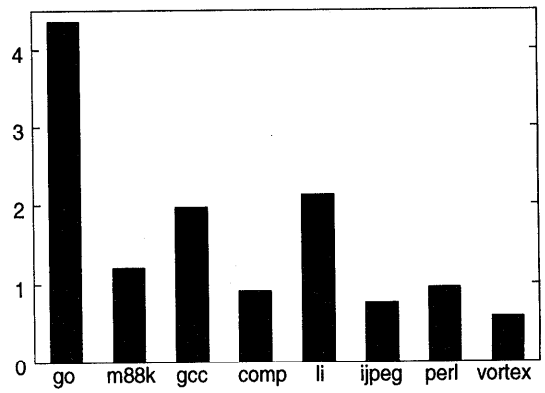


図16 予測失敗命令と再発行された命令の比
Fig. 16 Reissued instructions rate.

有効性と命令再発行機構の効果を確認できる．すなわち，命令の再発行を行うことでデータ依存投機実行を有効に活用できる．これは，投機失敗によるミスペナルティを大きく軽減できたことによる．

5.3 命令再発行による影響

図15 に，基本モデルと比較した場合の，評価モデルでの全実行命令の割合を示す．したがって，基本モデルよりも実行された命令数が多い場合には100%を超えることになる．各グラフは2つの領域に分かれている．上部（白）は再発行された命令の割合を示しており，下部（黒）は残りの命令の割合を示している．

容易に分かるように，ほとんどのプログラムで再発行される命令の割合は全実行命令と比較して非常に小さい．したがって，再発行された命令による資源競合の著しい増大はないと考えられる．また，命令の再発行にはプロセッサの命令フェッチ機構を必要としないことに注意されたい．つまり，命令フェッチ機構を必要とするのはグラフの黒で示された割合であり，すべての場合で100%以下である．すなわち，命令の再発行によって命令フェッチバンド幅に対する要求は増加しないことが分かる．近年のプロセッサでは，命令フェッチ機構が性能のボトルネックのうちの1つであり，フェッチバンド幅に対する要求が増えないことは望ましい．このことが，命令破棄の場合とは異なり，実行される命令が増加してもプロセッサの性能低下を引き起こさない理由である．

図16 には，予測に失敗した命令数と再発行された命令数との比を示す．すなわち，予測に失敗した命令1つあたりのその命令とデータ依存の関係にある命令数である．ほとんどの場合で2命令以下であり，なかには1命令以下のプログラムも存在する．これは，ロードデータ値が予測され投機的実行が可能な場合でも，資源競合などが原因で機能ユニットにディスパッチされず，実際にディスパッチされたときには予測されたデータを用いる必要がない場合が多いことを示している．このことが，図15において再発行された命

令の割合が小さいことの理由である．

6. 関連する研究

Lipasti[5]はデータ値の局所性に注目し，データ値を予測する手法を提案している．彼らの提案は最後に使用されたデータ値を利用し，last-value predictorと呼ばれている．last-value predictorでは予測精度は約50%である[5]のに対し，本稿で提案している予測器の予測精度は約80%である．さらに，提案予測器は全ロード命令の約50%を正しく予測可能であるが，予測精度が50%であるlast-value predictorはとうていこの値には及ばない．したがって，提案予測器がlast-value predictorよりも優れているといえる．

Moshovosら[6]によるspeculative memory cloaking，Tysonら[10]によるmemory renaming，そして本稿で提案している2ホップアドレス名前替えは，同時期に独立に考案されたにもかかわらず互いによく似ている．speculative memory cloakingにおいては，以下の操作でストア命令からロード命令にデータを受け渡す．ロード命令が発行されると，依存性予測器によりデータ依存に関係にあるストア命令が特定できる．続いて，このロード命令とストア命令を結び付けるタグを用いてsynosym fileと呼ばれるバッファにアクセスする．synosym fileにはストア命令がライトしたデータが保持されており，ロード命令はデータアドレスを計算しないでデータを獲得できる．speculative memory cloakingで用いられる依存性予測器は複雑な連想検索を多用しており，プロセッササイクルタイムを引き延ばしプロセッサの性能を低下させる恐れがある．

memory renamingにおいては，ストア命令とロード命令の間の依存関係をstore/load cacheに保持している．ロード命令が発行されるとstore/load cacheにアクセスし，データアドレスを獲得する．続いてそれを用いてvalue fileをアクセスする．value fileにはストア命令がライトしたデータが保持されているので，ロード命令はデータアドレスを計算しなくてもデータを獲得できる．しかし，文献10)には，ストア命令とロード命令を関連づけるための具体的な方法は説明されていない．

speculative memory cloakingにおけるsynosym file，memory renamingにおけるvalue fileは，2ホップアドレス名前替えで用いられるSIVTと目的は同じである．しかし実現法が異なっている．synosym fileではロード命令とストア命令を結び付けるタグが必要になる．このタグを決定するためには，連想検索を行う依存性予測器が必要になる．value fileではデータアドレスが必要である．データアドレスはstore/load cacheから獲得できる．一方，SIVTではストア命令アドレスを必要とする．ストア命令アドレスは構造の単純なLISTから獲得できる．すなわち，speculative memory cloakingおよびmemory renamingと比較して本稿で提案した機構の優れている点は，ハードウェアが非常に単純なことである．提案機構は複雑なハードウェアを必要としないので，プロセッサのサイクルタイムを引き延ばしてしまう心配がない．これは，2ホップアドレス名前替えを実現するために必要な3つの操作を，独立した3つのハードウェアで実現しているためである．たとえばspeculative memory cloakingでは，LISTとDISTの動作を1つの依存性予測器で実現しているため，複雑なハードウェアを必要としている．

7. むすび

データ依存性の投機的実行を検討した．ロードされるデータの値を予測して，データ依存関係を投機的に解消する．ストア命令の情報を利用してロードデータ値を予測する．同じメモリ領域を参照するストア命令とロード命令とを関連づけ，ストアされるデータをロード命令に受け渡す．SIVTを用いてデータアドレスをストア命令アドレスに名前替えし，名前替えされたストア命令アドレスをLISTを用いてさらにロード命令アドレスに名前替えする．我々は，この方法を2ホップアドレス名前替えと呼んでいる．ストア命令とロード命令を関連づけるためにはDISTを用いる．SIVT，LIST，DISTを用いたデータ値予測機構は非常にシンプルで実装が容易である．

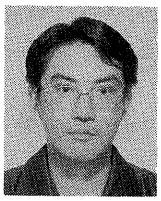
シミュレーションによりデータ値予測の効果を評価した．シミュレーションの結果，データ値の予測精度は平均79.0%，最大93.1%であった．この予測器を用いることで，ILPを平均3.4%，最大9.4%向上できる．以上により，本提案のロードデータ値予測機構の有効性が確認できた．

参考文献

1) Burger, D. and Austin, T.M.: The SimpleScalar tool set, version 2.0, *ACM SIGARCH Computer Architecture News*, Vol.25, No.3, pp.13–25 (1997).
2) Gonzalez, J. and Gonzalez, A.: Speculative execution via address prediction and data prefetching, *Proc. 11th Int'l Conf. on Supercomputing*, pp.196–203 (1997).

3) Jacobsen, E., Rotenberg, E. and Smith, J.E.: Assigning confidence to conditional branch predictions, *Proc. 29th Ann. Int'l Symp. on Microarchitecture*, pp.142–152 (1996).
4) Jourdan, S., Sainrat, P. and Litaize, D.: Exploring configuration of functional units in an out-of-order superscalar processor, *Proc. 22nd Ann. Int'l Symp. on Computer Architecture*, pp.117–125 (1995).
5) Lipasti, M.H.: Value locality and speculative execution, Ph.D. Dissertation, Depertment of Electrical and Computer Engineering, Carnegie Mellon University (1997).
6) Moshovos, A.I. and Sohi, G.S.: Streamlining inter-operation memory communication via data dependence prediction, *Proc. 30th Ann. Int'l Symp. on Microarchitecture*, pp.235–245 (1997).
7) 佐藤寿倫：アドレス名前替えによるロード命令の投機的実行，並列処理シンポジウム JSPP'98 予稿集，pp.15–22 (1998).
8) 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).
9) Sohi, G.S.: Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers, *IEEE Trans. Comput.*, Vol.39, No.3, pp.349–359 (1990).
10) Tyson, G. and Austin, T.M.: Improving the accuracy and performance of memory communication through renaming, *Proc. 30th Ann. Int'l Symp. on Microarchitecture*, pp.218–227 (1997).

(平成10年8月20日受付)
(平成11年3月 5 日採録)



佐藤 寿倫（正会員）

平成元年京都大学工学部卒業．平成3年同大学大学院工学研究科修士課程修了．同年，(株) 東芝入社．ULSI研究所においてマルチプロセッサアーキテクチャ，および消費電力見積り手法の研究に従事．平成8年よりマイクロエレクトロニクス技術研究所に所属．以来，組み込み用マイクロプロセッサの開発に従事．マイクロプロセッサアーキテクチャ，VLSI設計手法に興味を持つ．博士（工学）．電子情報通信学会，ACM，IEEE-CS各会員．