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

佐藤寿倫†

曖昧なメモリ参照によるデータ依存の投機的解消をシミュレーションにより評価した．ロード/ストア命令の参照するアドレスを予測することで曖昧なデータ依存関係を投機的に解消する．データ依存の投機実行を実現するには，予測に失敗したときにプロセッサ状態を回復させる機構が必要である．本稿では，そのための命令の再発行機構を提案する．投機に失敗した場合でも，投機実行された命令とデータ依存にある命令だけを選択して再実行するため，ミスペナルティを軽減できる．シミュレーションの結果，命令再発行機構は，命令を破棄した場合にはミスペナルティの甚大なプログラムに対しても，データアドレス予測による投機実行を有効にし，命令レベル並列度を最大26.5%向上できることが確認できている．

Improving Efficiency of Dynamic Speculation via Data Address Prediction Using Instruction Reissue Mechanism

TOSHINORI SATO†

In this paper, we evaluate resolving ambiguous memory aliasing speculatively. Utilizing data address prediction, an ambiguous memory dependence is speculatively resolved and a load instruction is speculatively executed. In order to realize the data speculation mechanism, a mechanism which recovers processor state when a misprediction occurs is indispensable. We introduce an instruction reissue mechanism. Since instructions independent of misspeculated instructions are not squashed, the effect of data dependence speculation is enhanced. From the experimental evaluation, we have found that the instruction reissue mechanism improves the efficiency of the dynamic data dependence speculation and instruction level parallelism is increased by up to 26.5%.

1. まえがき

データ依存は命令レベル並列度（instruction level parallelism: ILP）を制限する制約の1つである．データ依存には2種類ある．1つはレジスタを介した依存であり，もう1つはメモリを介した依存である．レジスタを介した依存のうち，WAWハザード（write after write hazard）とWARハザード（write after read hazard）はレジスタ名前替えにより解消可能であるが，RAWハザード（read after write hazard）を解消する方法はない．一方，メモリを介した依存に対しては，レジスタ名前替えのような方法すらない．これは，プログラムの実行時になるまでデータアドレスを知ることができないためである．したがって，あるストア命令が実行保留中であると，後続のロード/ストア命令を実行できない．これが曖昧なメモリ参照による依存（ambiguous memory dependence）であり，ILPの抽出を妨げる深刻な問題である．本稿ではこの曖昧なメモリ参照による依存を取り扱う．

曖昧なメモリ参照による依存を解消する方法は，これまでにもいくつか報告されている．しかしながら，それらはそれぞれ欠点を持っている．本稿では，データアドレスを予測することにより曖昧なメモリ参照による依存の解消を検討する．そのための機構は非常に単純であり，実装も容易である．ロードされるデータのアドレスを予測することでロード命令を投機的に実行し，さらにこのロード命令と依存関係にある命令も投機的に実行する．また，ストアされるデータのアドレスを予測することで，このストア命令と曖昧なメモリ参照による依存関係にあるロード命令を投機的に実行する．さらにこのロード命令と依存関係にある命令も投機実行の対象とする．

データ依存の投機実行を実現させるには，予測に失

† 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

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も考慮する必要がある．本稿では，このための命令ウインドウを提案している．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．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．そして，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ことである．我々は上記の問題を検討し，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，それらを命令ウインドウ内部で再発行する機構を考案した．この機構は，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．

本稿は以下のとおりに構成される．2章で関連する研究をまとめる．3章でアドレス予測手法を説明し，曖昧なメモリ参照による依存を投機的に解消する方法を検討する．4章では投機に失敗した場合にプロセッサの状態を回復するための命令再発行機構を提案する．5章で提案機構の評価方法を説明し，6章で評価する．最後に7章でまとめる．

2. 関連する研究

曖昧なメモリ参照によるデータ依存に関しては以下の研究が存在する．

静的な依存性解析（static dependence analysis）あるいは動的な曖昧性解消（dynamic memory disambiguation）による解消が検討されている．静的な依存性解析はコンパイル時に実行される[6]．多くの場合，実行時にならなければ得ることのできない情報のために，静的解析には限界がある．動的な曖昧性解消法は，プログラムの実行時に依存を解消する[9),11),18)]．この方法では，誤って依存性を解消してしまった場合のために，補償のための命令をプログラム中に含める必要がある．したがって，プログラムサイズの爆発が問題となり，適用できる場合に限界がある．本稿で検討する手法は，動的にデータ依存を解消するため実行時の情報を利用可能であり，また既存のバイナリを用いることが可能であるのでプログラムサイズ増大の心配もない．

アドレス分析バッファ（address resolution buffer: ARB）[7]は曖昧なメモリ参照によるデータ依存を投機的に解消する．すべてのストア命令とロード命令の間には依存性は存在しないと仮定してプログラムを実行し，依存性が存在することが判明した場合には，プログラムの実行をロールバックして再開する．本稿で検討する手法は，依存性がないと予測される場合だけロード命令を投機的に実行するので，不要なロールバックを減少させる効果が期待できる．また，ARBはプロセッサコアとL1データキャッシュとの間に配置されるため，メモリ参照のレイテンシを増大させてしまう問題もある．

Moshovosら[16]はデータアドレスの衝突を予測することを提案している．予測には過去の実行履歴を用いる．もし，あるストア命令と後続のロード命令が同じメモリ領域を参照しないと予測されると，このロード命令はストア命令を追い越して実行可能となる．本稿で検討する手法はMoshovosらの手法と似ているが，Moshovosらの機構が連想検索を多用して実装が困難であるのに対して，提案機構は単純で実装が容易である．

ロードデータアドレス予測には多くの報告がある[1),10),19),21),25),28)]が，ロード命令と依存関係にある命令を投機的に実行していないもの[1),21)]や，後続の命令を投機実行しているがストアデータのアドレスを予測してはいないもの[19),25),28)]がほとんどである．本稿で検討する手法は，ストア命令を予測することで曖昧なメモリ参照によるデータ依存を投機的に解消している．Gonzalezら[10]はストアアドレスの予測を検討しているが，ARBを用いているためにメモリ参照レイテンシ増大の問題が残る．

Lipasti[15]は値の局所性に注目しロードデータ値の予測手法を提案している．あるロード命令がメモリから読み出す値はつねに同じだと仮定し，同じロード命令が前回メモリから読み出した値を用いて投機実行する．したがって，メモリから読み出される値が実際には頻繁に書き変わる場合には，Lipastiの方法は効果的には働かないと予想される．本稿で検討している方法ではデータアドレスを予測してメモリを参照するので，データが更新された場合でも最新の値を用いて投機実行が可能である．

一方，命令再発行に関しても以下の研究が存在する．

ミスペナルティの軽減には命令の再発行[10),15)]が有効である．予測ミスをしたロード命令以降に実行される命令であっても，もしそのロード命令とデータ依存の関係がなければ破棄する必要はない．依存関係にない命令を破棄せずその実行結果を再利用できれば，ミスペナルティを軽減できる．命令間のデータ依存関係

を命令ウインドウ中に保持しておけば，予測ミスを犯したロード命令とデータ依存関係にない命令を検出することは可能である．したがって，依存関係にある命令を破棄したあと再フェッチするのではなく，命令ウインドウ中で再発行することができる．ところが，これまで命令再発行にはそれほど関心が払われていない．文献 10)，15) においてもコンセプトの提案だけであり，実現法は触れられていない．しかし，コンセプトどおりに実装することは非常に困難である．なぜなら，予測に失敗したロード命令と依存関係にあるすべての命令を 1 サイクル以内に検出しなければならないからである．

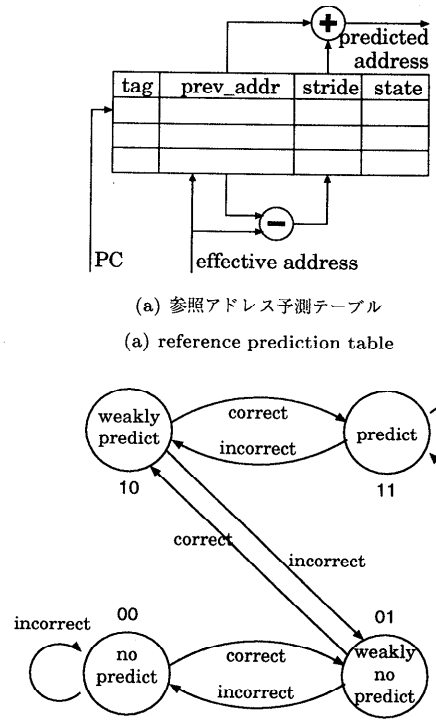
我々の知る限り，命令再発行を詳しく検討しているのは Rotenberg ら[20]による報告だけである．彼らはトレースプロセッサにおいて命令再発行の効果を検証している．しかし，現在主流のスーパースカラプロセッサにおける効果には触れていない．本稿で我々は実現可能な命令再発行機構を提案し，それを用いた際のデータ依存投機実行の効果をスーパースカラプロセッサ上で評価している．

3. データ依存投機実行機構

本章でデータ依存投機実行機構を説明する．データアドレスを予測するためには，参照アドレス予測テーブル（reference prediction table: RPT）[3]を用いる．まず RPT を説明し，続いてデータアドレス予測を用いたデータ投機実行を説明する．最後に，曖昧なメモリ参照によるデータ依存の投機的解消について述べる．

3.1 参照アドレス予測テーブル

RPT は Chen ら[3]によってハードウェアプリフェッチングを行う機構として提案され，キャッシュに似たハードウェア構成をしている．我々は RPT のハードウェア構成が単純なことに着目し，データアドレスを予測するために RPT を応用することを検討した．図 1 (a) に RPT の構成を示す．RPT にはメモリ参照に関する過去の履歴が保存される．RPT の各エントリはプログラムカウンタ（program counter: PC）でインデックスされており，以下の情報を保存している．すなわち，最も最近参照したデータアドレス（**prev_addr**），データアドレスのストライド（**stride**），そして予測の可否を示す状態（**state**）である．**stride** は同じ命令が最近 2 回に参照したデータアドレスの差から求められる．**state** には過去の履歴がエンコードされて保存されており，次回の予測が可能かどうかを示している．**state** の状態遷移の例を図 1 (b) に示す．これは文献 3) で提案されているオリジナルの状態遷移とは異



(a) 参照アドレス予測テーブル

(a) reference prediction table

(b) 状態遷移図

(b) state transition

図 1 データアドレス予測機構

Fig. 1 Reference address prediction.

なっていることに注意されたい．図 1 (b) で示される状態遷移は分岐予測[14]で用いられる 2 ビット飽和型カウンタ（2bC）と同じものであり，**predict**，**weakly predict**，**no-predict**，**weakly no-predict** の 4 状態をとりうる．

データアドレスは以下のような操作で予測される．タグが一致した場合には，PC で示されるエントリから **prev_addr** と **stride** とが得られる．予測アドレスは **prev_addr** と **stride** との和として計算される．**state** も同時に得られ，もし状態が **predict** または **weakly predict** であれば計算されたアドレスは予測アドレスとして用いられる．そうでない場合は，アドレスの予測を行わない．**state** の状態遷移は以下のとおりである．もし予測が正しければカウンタを +1 増加させる．予測に失敗した場合には逆にカウンタを 1 減少させる．カウンタの最上位ビットが 1 の場合は，状態は **predict** または **weakly predict** であるので，アドレスの予測を行う．

状態遷移に 2bC を選んだ理由は以下のとおりである．**図 2** はオリジナルの状態遷移図である[3]．オリジナルの状態遷移を用いる場合には，状態が **steady** に

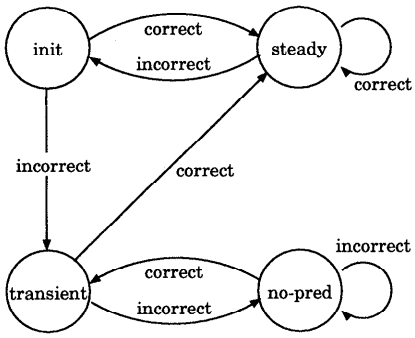


図2 オリジナルの状態遷移図
Fig. 2 Original state machine.

いるときに限りアドレスが予測される．オリジナルの状態遷移を用いたRPTと2bCを用いたRPTとを比較するために，機能シミュレータ[2]を用いてSPEC95ベンチマークを実行し，データアドレスの予測可能性を調べた．ここで用いたRPTは1024エントリでダイレクトマップとした．我々はロード/ストア命令を計数し，RPTがアドレスを予測できる割合と正しくアドレスを予測した割合を計算した．**図3**と**図4**に結果をまとめる．各プログラムに対応する4本のグラフは左から順にそれぞれ，ロードアドレスだけを予測するRPTの結果，ストアアドレスだけを予測するRPTの結果，ロード/ストアアドレスを予測するRPTのロードアドレス予測の結果，そしてロード/ストアアドレスを予測するRPTのストアアドレス予測の結果を表している．各グラフは3つの部分に分けられており，下部（黒）はデータアドレスが正しく予測された割合（*RPT_correct*），中部（白）はデータアドレスを誤って予測した割合（*RPT_miss*），そして上部（灰）はアドレスを予測できなかった割合を表している．RPTがアドレスを予測できた割合は $RPT\_hit = RPT\_correct + RPT\_miss$ で表されるので，アドレス予測精度は $RPT\_accuracy = RPT\_correct/RPT\_hit$ で求めることができる．容易に分かるように，アドレス予測精度 *RPT_accuracy* は2つのRPTで大差はないが，予測可能性 *RPT_hit* には大きな差がある．すなわち2bCを用いたRPTの方が正しくアドレスを予測できる割合 *RPT_correct* に優れており，より投機性の高い実行が可能である．以上の検討により，我々は2bCを用いるRPTを採用することとした．

3.2 ロードアドレス予測によるデータ投機実行

続いてロードデータアドレスの予測を用いたデータ依存性の投機実行を説明する．ロード命令を投機的に実行することにより，データフローグラフにおけるクリティカルパスを短縮することが可能である．命令が命令ウインドウに発行されるまでにRPTが参照され，ロードデータアドレスが予測される．より投機性の高い実行を行うため，RPTがデータアドレスを予測可能な場合はすべてのロード命令を投機的に実行することとする．パイプライン図を**図5**に示す☆．

RPTにより予測されたデータアドレスは，実際のデータアドレスと比較されるまで命令ウインドウ中に保持される．したがって，命令ウインドウはオペランドだけでなくデータアドレスも保持できる必要がある．命令ウインドウの構成については4章で述べる．

ロード命令は予測アドレスを用いて投機的にメモリアクセスを実行する．実際のデータアドレスが計算されると，それは命令ウインドウ中に保持されている予測アドレスと比較される．もし2つのアドレスが一致すれば，アドレス予測は正しかったことになる．データアドレスの生成と2つのアドレスの比較は同時に行うことが可能なので[5]，比較のためのサイクルは必要でない．したがって，アドレス予測が正しかった場合には改めてメモリにアクセスする必要はない．もし一致しなければ予測は間違っており，改めてメモリにアクセスして正しいデータを読み出すと同時に，プロセッサの状態を投機前の状態に回復する必要がある．**図6**に示すように，予測に失敗したロード命令と依存関係にある命令は再実行されなければならない．図において，*のついた命令が再実行されなければならない命令である．予測に失敗したロード命令と依存関係にある命令を再実行する方法には以下の2つの方法がある．1つはロード命令以降のすべての命令を破棄する方法（図6(a)）であり，もう1つは依存関係にある命令だけを選択的に再実行する方法（図6(b)）である．我々は前者を命令破棄（instruction squashing）に基づく方法，後者を命令再発行（instruction reissue）に基づく方法と呼んでいる．命令破棄に基づく方法には予測に失敗した命令と依存関係にない命令までも無効化し再実行しなければならないというペナルティがあるが，命令再発行に基づく方法よりも単純で実装が容易であるという利点を持つ．実際，分岐予測のために用意されているプロセッサ状態の回復機構を流用可能である．命令再発行機構の実装については4章で説明する．

3.3 曖昧なデータ依存の投機的解消

本章の最後として，ストアデータアドレス予測を用

☆ 理解を容易にするために，単純なパイプラインとした．実際には演算に複数のステージを要する命令が存在する．

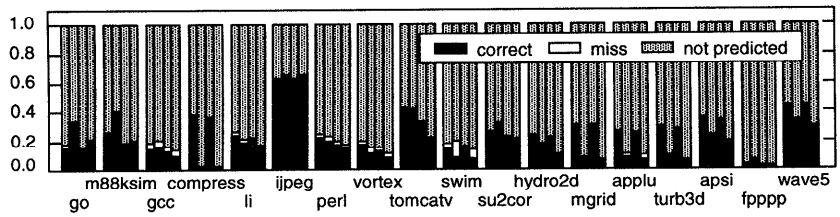


図3 データアドレスの予測可能性（オリジナルの場合）
Fig. 3 Address predictability (original state machine).

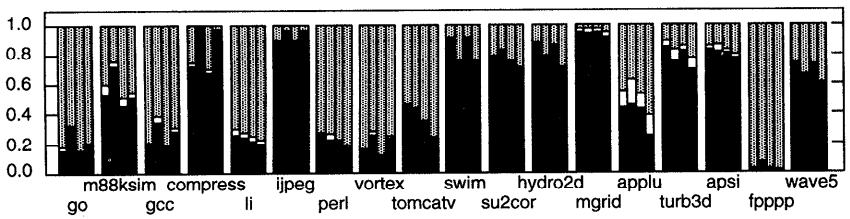


図4 データアドレスの予測可能性（2ビットカウンタの場合）
Fig. 4 Address predictability (2-bit counter state machine).

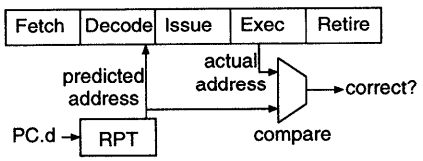


図5 パイプライン図
Fig. 5 Pipeline diagram.

```
ld    r1 <- r10(0)          ld    r1 <- r10(0)
add   r5 <- r3 + r4*        add   r5 <- r3 + r4
add   r2 <- r1 + r5*        add   r2 <- r1 + r5*
fadd  f1 <- f1 + f2 *       fadd  f1 <- f1 + f2
st    r2, r10(0)    *       st    r2, r10(0)    *
      (a)                         (b)
```

* squashed instruction

図6 アドレス予測を失敗したロード命令
Fig. 6 Mispredicted load instruction.

```
@st   r11, r10(0)           @st   r11, r10(0)
  add   r5 <- r3 + r4*        add   r5 <- r3 + r4
  ld    r6 <- r1(0)   *       ld    r6 <- r1(0)
@ld   r7 <- r5(0)   *       @ld   r7 <- r5(0)   *
  fadd  f1 <- f1 + f2 *       fadd  f1 <- f1 + f2 *
  add   r2 <- r1 + r7*        add   r2 <- r1 + r7*
        (a)                         (b)

@st   r11, r10(0)
  add   r5 <- r3 + r4
  ld    r6 <- r1(0)
@ld   r7 <- r5(0)             @ address conflict
  fadd  f1 <- f1 + f2 *       * squashed instruction
  add   r2 <- r1 + r7
                    *
        (c)
```

図7 アドレス予測を失敗したストア命令
Fig. 7 Mispredicted store instruction.

いた曖昧なメモリ参照によるデータ依存の投機的解消手法を説明する．曖昧なデータ依存の投機的解消により，あるストア命令と曖昧な依存関係にあるロード命令を投機的に実行可能になる．ロードデータアドレスの予測と同様に，ストアデータアドレスはストア命令が命令ウインドウに発行されるまでに予測される．もし後続のロード命令が参照するデータアドレス☆が予測されたストアデータアドレスと異なっていれば，曖昧なメモリ参照は解消されこのロード命令はストア命令を追い越して実行可能である．パイプライン図は，図5に示されるロードデータアドレスの予測の場合と同様である．

投機的解消に失敗した場合には，プロセッサ状態を投機前の状態に回復させる必要がある．**図7**に示すように，データアドレスの予測に失敗したストア命令と依存関係にある命令は再実行されなければならない．@のついたロード/ストア命令が実際に同じデータアドレスを参照しており，*のついた命令が再実行の対象となる命令である．再実行手法には以下の3つの選択肢がある．(1) ストアデータアドレスの予測に失敗したストア命令に後続するすべての命令を再実行する（図7(a)）．(2) このストア命令と現実にデータ依存の関係にあるロード命令に後続するすべての命令を再実行する（図7(b)）．(3) このストア命令と現実にデータ依存の関係にある命令を選択的に再実行する

☆ このロードデータアドレスは，実際の値であっても予測された値であってもよい．

(図7(c)). (1)の方法では予測に失敗した命令と依存関係にない命令までも無効化し再実行しなければならないというペナルティがあるが，ほかの2つの方法よりも単純で実装が容易であるという利点を持つ．実際，分岐予測のために用意されているプロセッサ状態の回復機構を流用可能である．(2)の方法は，現実にはストア命令とデータ依存の関係にありながらこの依存関係を無視したロード命令を発見してこのロード命令以降の命令を破棄するので，ストアデータアドレスの予測を失敗したとしても必ずしもプロセッサ状態の回復操作が行われるわけではない．したがって，(1)の方法で生じる無駄な命令破棄を行わない．ストアデータアドレスを予測した場合の4つのケースについて考察しよう．それらは以下のとおりである．(i)アドレス予測の結果データ依存があると予測されたが，現実にはデータ依存は存在しなかった．(ii)アドレス予測の結果データ依存があると予測されたが，現実にデータ依存が存在していた．(iii)アドレス予測の結果データ依存がないと予測されたが，実際にデータ依存は存在しなかった．(iv)アドレス予測の結果データ依存がないと予測されたが，実際にはデータ依存が存在した．ストアアドレスの予測に失敗した場合に，プロセッサ状態の回復をしなければならないのはケース(iv)だけである．他のケースでは回復する必要はないし，逆に性能低下を引き起こしてしまう．上記の(2)の方法ではケース(iv)だけを検出するので，性能低下を引き起こす問題はない．この方法を実現するにはHP PA-8000[12]に採用されているアドレスリオーダーバッファ(address reorder buffer)を用いればよい．アドレスリオーダーバッファはあるストア命令とロード命令が同じデータアドレスを参照しているかどうかを検出し，もし参照していれば，ストア命令を追い越して実行されたロード命令以降の命令を破棄する．つまり(2)の方法を実現している．(3)の方法を実現するには，アドレスリオーダーバッファを改造すればよい．改造アドレスリオーダーバッファは同じアドレスへの参照だけを検出し，命令の破棄は行わない．投機に失敗しストア命令を追い越してしまったロード命令を検出し，そのロード命令と依存関係にある命令を4章で述べる方法で再発行する．したがって，このロード命令と依存関係にある命令だけが選択的に無効化され再発行される．

4. 命令再発行機構

本章でレジスタ更新ユニット(register update unit: RUU)[26]を拡張して実現した命令再発行機構を提案する．まずRUUとそれを用いた動的命令スケジューリングを説明する．続いて命令の破棄に基づくデータ依存投機実行を説明する．最後に，命令再発行を可能にする拡張レジスタ更新ユニットを提案し，それを用いた命令再発行を説明する．

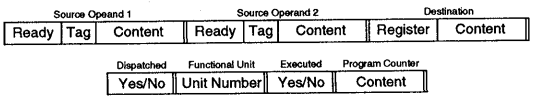


図8 レジスタ更新ユニット
Fig. 8 Register update unit.

4.1 RUUを用いた命令スケジューリング

RUUを用いた命令ウインドウを図8に示す．命令ウインドウの各エントリは，2つのソースオペランドフィールド(Source Operand)，デスティネーションフィールド(Destination)，ディスパッチビット(Dispatched)，機能ユニットフィールド(Functional Unit)，実行ビット(Executed)，そしてプログラムカウンタフィールド(Program Counter)から構成される．もしソースオペランドがいまだ得られないときには，レディビット(Ready)がリセットされソースオペランドが利用不可能であることを示す☆．同時に，そのオペランドを示すタグ(Tag)がセットされる．オペランドが利用可能になると，ソースレジスタの値がContentフィールドにセットされ，Readyビットがセットされる．Tag情報をともなったデスティネーションレジスタ番号はDestinationフィールドのRegisterフィールドに保持され，命令の実行結果はContentに保存される．Dispatchedビットは Functional Unitフィールドで指定される機能ユニットに命令がディスパッチされているかどうかを示す．Executedビットは命令が完了するとセットされる．Executedビットがセットされていると，この命令とデータ依存関係にある後続命令はディスパッチ可能になる．最後にProgram Counterフィールドは，分岐予測失敗からのプロセッサ状態の回復や，正確な割込みを実現するために用いられる．

例を用いて命令再発行を説明する．図9に示す命令シーケンスを用いて説明する．理解を容易にするために以下の仮定をおく．まず，演算f_1–f_6のオペランドは1つとする．また，命令I3はロード命令とし，その演算レイテンシはアドレス計算1サイクルとメモリアクセス1サイクルからなるものとする．最後に，残りの命令のレイテンシはすべて1サイクルとする．

☆ 文献26)では，ソースオペランドが得られないときにReadyビットがセットされる．本稿では，理解を容易にするために，オペランドが得られたときにReadyビットをセットした．

I1: $r11 \leftarrow f_1(r10)$
I2: $r12 \leftarrow f_2(r11)$
I3: $r13 \leftarrow f_3(r12)$
I4: $r14 \leftarrow f_4(r13)$
I5: $r15 \leftarrow f_5(r14)$
I6: $r16 \leftarrow f_6(r15)$

図9 命令シーケンスの例
Fig. 9 Instruction sequence example.

図10に図9の命令シーケンスを実行した場合の命令スケジューリングの例を示す．理解を容易にするために以下の仮定をしている．プロセッサのフェッチ幅とディスパッチ幅は，ともに3とする．機能ユニットの数は無限とする．命令のコミットは説明から省略する．最後に，タグ情報を含んだデスティネーションレジスタ番号は，アーキテクチャレジスタ番号と同じであると仮定し，レジスタ $r10$ はすでに利用可能になっているものとする．以上の仮定に基づいて，命令スケジューリングを説明する．まず最初のサイクルで命令 I1–I3 が RUU に発行される．$r10$ が利用可能なので命令 I1 の **Ready** ビット（**r**）がセットされ，さらに命令 I1 は機能ユニットにディスパッチされて **Dispatched** ビット（**d**）がセットされる．2サイクル目で命令 I4–I6 が発行される．命令 I1 が演算を終え **Executed** ビット（**e**）をセットする．$r11$ が利用可能になり，命令 I2 がディスパッチされる．図で太線で表した矢印はソースオペランドタグとデスティネーションタグが一致したことを表している．3サイクル目で，命令 I2 が演算を終え $r12$ が利用可能になり，命令 I3 がディスパッチされる．命令 I3 はロード命令なので，結果が得られるまでに2サイクル必要になる．したがって，4サイクル目にディスパッチされる命令はない．5サイクル目で，ロード命令 I3 が $r13$ を読み出し，命令 I4 がディスパッチされる．6サイクル目で，命令 I4 が演算を終え $r14$ が利用可能になり，命令 I5 がディスパッチされる．7サイクル目で，命令 I5 が演算を終え $r15$ が利用可能になり，命令 I6 がディスパッチされる．最後に8サイクル目で命令 I6 が演算を終え，図9の例の実行が終わる．

以上のように，この例の命令シーケンスはデータ依存のためにシーケンシャルに実行せざるをえない．そのため実行には8サイクルを要する．

4.2 命令の破棄を行うデータ依存投機実行

RUU を拡張した命令ウインドウを**図 11** に示す．各エントリにはデータアドレスフィールド（**Data Address**）と予測ビット（**Predicted**）が追加されている．**Data Address** フィールドには予測されたデータアドレスが保存される．**Data Address** フィールド

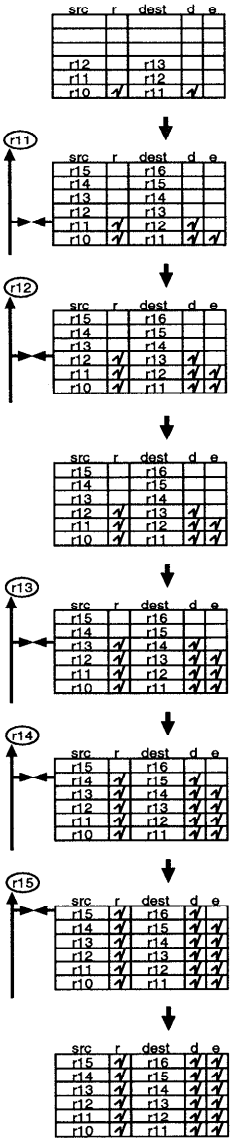


図10 RUU を用いた命令スケジューリング
Fig. 10 Dynamic scheduling with RUU.

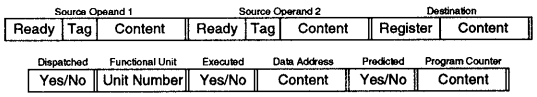


図11 拡張された命令ウインドウのエントリ
Fig. 11 Extended RUU for instruction squashing.

によるハードウエア量の増加は，命令ウインドウをロード/ストア命令用と残りの命令用に分割することで削減できる．**Data Address** フィールドはロード/ストア命令用の命令ウインドウ，すなわちロード/ストアキューにだけあればよい．ソースオペランドが利用可能になると，実際のデータアドレスとの比較

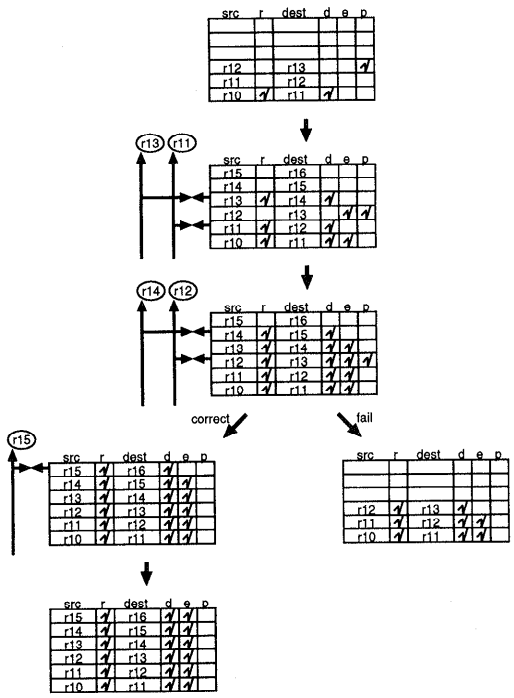


図12 データ依存投機実行の例
Fig. 12 Instruction squashing example.

が行われる．Predicted ビットは，アドレス予測が行われたときにセットされる．また Destination の Content フィールドは，予測されたアドレスを用いて獲得されたデータを保持する目的でも使用される．つまり，Predicted ビットがセットされているとき，この Content フィールドは予測アドレスから読み出された値を保持する．実際のアドレスが計算されると予測された値と実際の値とを比較して投機実行の成功/失敗が判定され，Predicted ビットがリセットされる．予測に失敗した場合には，そのロード命令に後続する命令はすべて破棄され，あらためて命令フェッチからプロセッサの動作が再開される．

図12 に，この拡張された RUU を用いたデータ依存の投機実行例を示す．図9 の命令シーケンスを用い，前節と同じ仮定をする．さらに，命令 I3 がロードデータアドレスを予測し，命令 I3 とデータ依存関係にある命令が投機実行されるとする．まず，命令 I1–I3 が RUU に発行される．$r10$ が利用可能なので，命令 I1 が機能ユニットにディスパッチされ，Ready ビットと Dispatched ビットがセットされる．命令 I3 はロードデータアドレスを予測し，Predicted ビット（p）をセットする．予測されたアドレスは Data Address フィールドに保存されている．次のサイクルで，命令 I4–I6 が発行される．命令 I1 が演算を終え $r11$ が利用可能になり，命令 I2 がディスパッチされる．命令 I3 はメモリから $r13$ を読み出し，それを Destination の Content フィールドに保持する．命令 I4 はその $r13$ を用いてディスパッチされる．演算を終えた命令 I1 と I3 は Executed ビットをセットする．3 サイクル目で命令 I2 と I4 が演算を終えて $r12$ と $r14$ が利用可能になるため，命令 I3 と I5 がディスパッチされる．次のサイクルでは 2 通りの場合が考えられる．すなわち，命令 I3 がアドレス予測に成功した場合と失敗した場合である．アドレス予測に成功した場合は以下のとおりである．命令 I3 は演算を終え予測ビットをリセットする．同時に命令 I5 も演算を終え命令 I6 がディスパッチされる．最後のサイクルで命令 I6 が演算を終え，この例の命令シーケンスは 5 サイクルで完了する．投機実行を行わない場合と比較すると，アドレス予測に成功すると実行時間が 3 サイクル短縮できる．一方，命令 I3 の予測に失敗した場合は，命令 I4–I6 は破棄されなければならない．図中で命令 I4–I6 が RUU から消えていることに注意されたい．命令 I3 は正しいアドレスで再度データを読み出す．同時に，プロセッサは命令 I4–I6 のフェッチから実行を再開する．つまりこの例では以下のミスペナルティを被ることが分かる．すなわち，命令を破棄するためのサイクル，そして命令を再度フェッチしデコードするためのサイクルである．したがって，データ依存の投機実行を効果的にするためには，予測失敗時のミスペナルティを軽減する方法を検討する必要がある．

4.3 命令の再発行を行うデータ依存投機実行

ミスペナルティの軽減には命令の再発行[10),15)]が有効である．予測ミスをしたロード命令以降に実行される命令であっても，もしそのロード命令とデータ依存の関係がなければ破棄する必要はない．依存関係にない命令を破棄せずその実行結果を再利用できれば，ミスペナルティを軽減できる．命令間のデータ依存関係を命令ウインドウ中に保持しておけば，予測ミスを犯したロード命令とデータ依存関係にない命令を検出することは可能である．したがって，依存関係にある命令を破棄したあと再フェッチするのではなく，命令ウインドウ中で再発行することができる．

図11 に示した RUU をさらに拡張して命令再発行を可能にした命令ウインドウを図13 に示す．各エントリにはさらに再発行ビット（Reissued）が付加されている．Reissued ビットは対応する命令が再発行されたことを表している．前節の説明と同様に，予測アドレスは Data Address フィールドに保持され，それを用いて読み出されたデータは Destination の Content

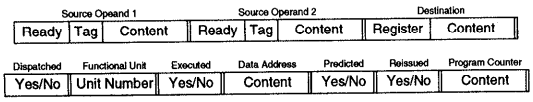


図 13 命令再発行を可能にする命令ウインドウ
Fig. 13 Extended RUU for instruction reissue.

フィールドに保持される．予測アドレスに対して実際のアドレスが計算されると，両者を比較して予測成功の判定が行われる．このとき **Predicted** ビットはリセットされる．もし比較結果が一致しなければ予測は失敗であり，**Reissued** ビットがセットされる．命令が終了し結果が得られると，従来の命令スケジューリングと同様にデスティネーションタグと実行結果が放送される．同時に再発行ビットも放送される．以後この信号を再発行信号と呼ぶことにする．予測に成功した場合は従来のスケジューリングと同様に，デスティネーションタグとソースオペランドタグの一致した後続の命令がオペランドを獲得する．一方，予測に失敗したときには，デスティネーションタグとソースオペランドタグの一致した命令は再発行される可能性がある．タグの一致した命令の **Dispatched** ビットがすでにセットされている場合には，その命令は誤ったソースオペランドを用いて演算を実行しているため再発行されなければならない．**Dispatched** ビットと **Executed** ビットはリセットされ，**Reissued** ビットがセットされる．再発行された命令の実行が終了した場合にも再発行信号が放送される．したがって，再発行信号は，アドレス予測に失敗した命令あるいは再発行された命令が実行を終了したことを表している．以降は上述の説明と同様に，タグが一致しかつ **Dispatched** ビットがセットされた命令が再発行される．こうして **Reissued** ビットが伝搬していくことで，再発行されなければならない命令が順に検出される．また，再発行されるべき命令がなくなると **Reissued** ビットが消えるので，たとえ RUU 内に命令が存在したとしても再発行は起こらない．本機構は従来の命令ウインドウと比較して，完了命令 1 命令あたり再発行信号が 1 本増えただけであり，ハードウェアの増大とサイクルタイムの延長を防いでいる．再発行されるべき命令を検出するためにタグを放送する機構は，動的にスケジューリングを行うプロセッサにはすでに存在していることに注意されたい．そのため，この機構がボトルネックにはならない．一方で同じ命令が複数回発行される可能性があるが，依存関係にある命令間の距離が大きかったり[8]機能ユニットの数の制限があったりするため，必ずしも演算結果を予測できた命令が投機実行されるわけではなく，冗長に発行される命令の数も制限されると思われるので深刻な問題ではない．

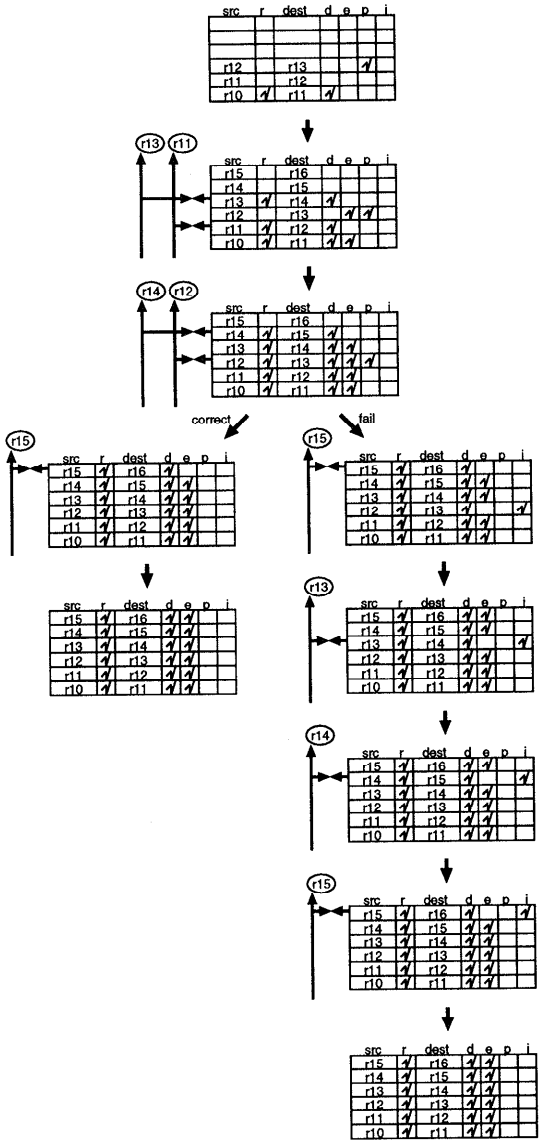


図 14 命令再発行の例
Fig. 14 Instruction reissue example.

図 14 に命令再発行の例を示す．図 12 の説明で用いた同じ仮定を用いる．命令 I3 の予測成功判定までは図 12 での説明と同様であるので省略し，予測に失敗した場合だけを説明する．予測に失敗した場合は以下のとおりである．命令 I3 の **Predicted** ビットはリセットされ **Reissued** ビット（i）がセットされる．命令 I3 はもう一度メモリにアクセスする必要がある．同時に命令 I5 が演算を終え命令 I6 がディスパッチされる．次のサイクルで，命令 I3 がメモリアクセスを終

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

え $r13$ が利用可能になる．命令 I3 の Reissued ビットがセットされているので再発行信号が放送される．タグの一致した命令 I4 は値の間違った $r13$ を用いてディスパッチされているので，$r13$ の放送によって再発行の対象として検出される．命令 I4 の Reissued ビットがセットされ，再発行されるべき命令の検出が伝搬する．6サイクル目で命令 I4 は演算を終え，命令 I5 が再発行される．理由は命令 I4 の再発行と同様である．続くサイクルで命令 I5 が演算を終え命令 I6 が再発行される．そして次のサイクルで命令 I6 が演算を終え，この例の命令シーケンスは8サイクルで完了する．この例では，データ値の予測に失敗した場合に必要なサイクル数は，データ依存の投機実行を行わない場合に必要なサイクル数と同じである．したがって，前節の命令破棄に基づく機構では必要だった，命令破棄のためのサイクルと命令の再フェッチおよびデコードのためのサイクルが必要ない．このように，命令再発行はミスペナルティを軽減できる．ただし現実には，機能ユニット数の制約などによって再発行された命令をただちにディスパッチできない場合があり，予測失敗によるペナルティが生じる場合があると考えられる．以上のように，命令の再発行は Reissued ビットと再発行信号の追加だけでよく，規模の小さなハードウェアを用いて実現可能である．

4.4 アドレス予測以外への応用

本章で提案した命令再発行機構は，アドレス予測を用いたデータ投機実行だけでなく，他の方法を用いた投機実行にも適用できる．たとえば，我々はすでにデータ値予測を用いた投機実行に適用することを検討している[23]．この場合には予測アドレスを保持する必要はないので，図13に示された RUU の各エントリにある Data Address フィールドは不要になる．また，予測失敗時のスケジューリング方法も異なる．アドレス予測と異なり，データ値予測では予測失敗検出時に命令の実行結果が得られているので，ただちに再発行信号を放送できる．したがって，予測に失敗した命令は Reissued ビットをセットする必要はない．この命令と依存関係にある命令に対しては，前章で説明した同じ操作が行われる．

5. 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．

5.1 プロセッサモデル

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13)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．RPT は1024エントリでダイレクトマップとした．命令破棄を採用した場合には，命令を破棄するために3サイクルのペナルティを仮定する．ストアアドレス予測失敗時の命令破棄は，3.3節で説明した(2)の方法を採用する．評価には以下の3つのモデルを用いた．まず，ロードデータアドレスだけを予測し，ロード命令を投機実行するモデル．続いて，ストアデータアドレスだけを予測し曖昧なデータ依存を投機的に解消するモデル．最後に，ロード/ストアデータアドレスを予測し，ロード命令を投機実行するとともに，曖昧なデータ依存を投機的に解消するモデルである．これらのモデルをそれぞれ **load**, **store**, **ldst** モデルと呼ぶことにする．また基本モデルを **base** モデルと呼ぶことにする．

評価には，SimpleScalar ツールセット[2]を用いた．SimpleScalar/PISA アーキテクチャは MIPS アーキテ

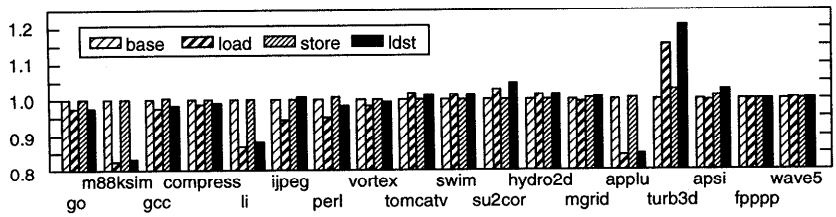


図15 命令レベル並列度の変化（命令破棄の場合）
Fig. 15 Performance improvement (squashing).

表2 ベンチマークプログラム
Table 2 Benchmark program and input file.

プログラム	略称	入力ファイル
099.go	go	null.in
124.m88ksim	m88	ctl.in
126.gcc	gcc	cccp.i
129.compress	com	test.in
130.li	li	test.lsp
132.ijpeg	ijp	specmun.ppm
134.perl	per	primes.in
147.vortex	vor	vortex.in
101.tomcatv	tom	tomcatv.in
102.swim	swi	swim.in
103.su2cor	su2	su2cor.in
104.hydro2d	hyd	hydro2d.in
107.mgrid	mgr	mgrid.in
110.applu	app	applu.in
125.turb3d	tur	turb3d.in
141.apsi	aps	apsi.in
145.fpppp	fpp	natoms.in
146.wave5	wav	wave5.in

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．SimpleScalarツールではトレースベースではなく実行ベースのシミュレーションを行えるため，投機に失敗した場合に誤って実行されてしまう命令もシミュレーションできている．

5.2 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．以後の表では各プログラムを名前の先頭3文字で表すことにする．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[2]．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．

6. シミュレーション結果

本章で評価結果を紹介する．まず命令破棄の影響を，続いて命令再発行を使用した場合のILPの向上を示す．

6.1 命令破棄の効果

図15は，命令破棄を採用した場合のILPの向上を表している．評価モデルのサイクルあたりの命令数（instructions per cycle: IPC）を基本モデルのIPCで正規化して評価している．コミットされた命令だけを数えて評価している．各プログラムに対応する4本のグラフは左から`base`, `load`, `store`, `ldst`の性能を示している．付録の表7には詳細な値をまとめた．

図15よりロード命令の投機実行は予想したほどILPの向上に貢献していないことが分かる．18本のうちの11本のプログラムでILPが低下している．例外は`turb3d`で，15.7%と著しくILPが向上している．このことは，ロード命令の投機実行は潜在的にはILPを向上できる可能性があることを示している．したがって，投機実行の効果を妨げる要因を突き止めそれらを排除できれば，ILPを向上できると思われる．一方，曖昧なデータ依存の投機的解消も期待したほどの効果がなく，たかだか3%のILPの向上にとどまっている．しかし，曖昧なデータ依存の投機的解消を用いた場合では，`tomcatv`を除いて必ずILPは向上している．したがって，曖昧なデータ依存の投機的解消はILP向上に有効であると結論できる．最後に，ロード命令の投機実行と曖昧なデータ依存の投機的解消を組み合わせると，ILPを最大21.0%向上できることが分かる．

以前に行った評価[22]と比較すると，上記の結果ではILPの向上度が小さい．これは以下の理由による．本稿で用いたシミュレータとベンチマークプログラムは以前の評価で用いたそれらと異なっている．文献22)でのプロセッサモデルは命令フェッチ機構が改良されており，interleaved sequential法[4]や改良された分岐予測バッファ[29]を用いている．またUhligら[27]によると，以前の評価で使用したSPEC92ベンチマークよりも今回使用したSPEC95ベンチマークの方が命令フェッチバンド幅に対する要求が大きい．以上から，以前の評価で用いたモデルよりも今回用いたプロセッサモデルの命令供給能力が小さいことが分かる．データ投機実行の効果は命令フェッチ能力に著しく影響を受ける[8]ので，以前の評価で用いたプロセッサモデルの方がデータ投機実行による恩恵をより大きく受ける

表3 アドレス予測精度とL1キャッシュミス率（命令破棄の場合）
Table 3 Address prediction accuracy and L1 cache miss rate (squashing).

	アドレス予測精度（%）				キャッシュミス率（%）							
	load	store	ldst		base		load		store		lsdt	
			ロード	ストア	命令	データ	命令	データ	命令	データ	命令	データ
go	88.25	95.37	87.93	97.36	0.50	1.05	0.48	0.91	0.50	1.05	0.48	0.91
m88	89.20	96.31	88.33	95.41	0.00	0.11	0.00	0.11	0.00	0.11	0.00	0.12
gcc	93.73	93.83	94.10	94.80	0.92	1.54	0.87	1.46	0.92	1.54	0.88	1.48
com	96.19	99.80	96.47	99.86	0.03	10.88	0.03	10.88	0.03	10.88	0.03	10.89
li	86.62	89.95	85.60	92.49	0.00	0.01	0.00	0.02	0.00	0.01	0.00	0.02
ijp	99.55	99.58	99.59	99.66	0.24	0.72	0.24	0.79	0.24	0.72	0.24	0.73
per	95.65	83.69	96.47	86.83	0.14	0.04	0.13	0.04	0.14	0.04	0.13	0.04
vor	94.38	92.79	95.27	94.27	1.08	0.68	1.04	0.69	1.08	0.68	1.05	0.69
tom	99.53	100.0	99.78	100.0	0.00	0.12	0.00	0.12	0.00	0.12	0.00	0.12
swi	100.0	100.0	100.0	100.0	0.00	0.13	0.00	0.13	0.00	0.13	0.00	0.13
su2	99.08	99.08	99.41	99.69	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20
hyd	99.27	99.90	99.76	99.91	0.00	0.14	0.00	0.14	0.00	0.14	0.00	0.14
mgr	97.53	97.18	97.72	97.13	0.00	3.92	0.00	3.97	0.00	3.92	0.00	3.97
app	83.05	81.19	83.02	78.98	0.00	3.44	0.00	3.47	0.00	3.44	0.00	3.46
tur	97.75	96.22	97.83	96.61	0.00	12.44	0.00	12.47	0.00	12.44	0.00	12.54
aps	96.50	94.86	97.01	96.34	0.04	3.39	0.04	3.36	0.04	3.40	0.04	3.36
fpp	98.74	98.93	98.48	96.57	10.89	0.03	10.87	0.03	10.89	0.03	10.87	0.03
wav	98.99	99.93	98.96	99.93	0.00	0.64	0.00	0.82	0.00	0.64	0.00	0.64

ことができる．したがって今回の評価では投機実行の効果が小さくなったと思われる．

続いて，データ投機実行の効果を妨げると思われる以下の要因について検討する．すなわち，アドレス予測精度，キャッシュミス率，分岐予測精度を検討する．

6.1.1 アドレス予測精度の影響

表3の左側にアドレス予測精度をまとめる．最初のコラムはベンチマークの名前である．次の2つのコラムは`load`，`store`モデルの予測精度である．そして次の2つのコラムは`ldst`モデルでの，ロードアドレス，ストアアドレスの予測精度である．

`load`モデルが著しくILPの低下を示している3つのプログラム（`m88ksim`，`li`，`applu`）で，ロードアドレス予測精度が低いことが分かる．これは，ILPの低下とアドレス予測精度には相関があることを示している．ロード命令の投機実行ではデータアドレスが生成できるまで予測の成功/失敗を決定できないため，予測に失敗したときのペナルティが非常に大きい．したがって，予測精度が低いと大きなペナルティを被るためにILPが低下していると考えられる．

6.1.2 キャッシュミス率の影響

L1キャッシュのミス率を表3の右側に示す．最初の2コラムは`base`モデルでの命令キャッシュおよびデータキャッシュのミス率を表している．残りはそれぞれ`load`，`store`，`ldst`モデルの結果である．ロード命令の投機実行や曖昧なデータ依存の投機的解消を行っても，キャッシュのミス率には大きな変化は見られない．したがって，キャッシュのミス率の変化がILPの低下を引き起こしたわけではないと考えられる．

6.1.3 分岐予測精度の影響

表4に分岐予測精度をまとめる．3コラムずつまとめられた4組はそれぞれ`base`，`load`，`store`，`ldst`モデルの結果である．3つのコラムは左から順に，分岐先アドレス予測精度（`addr`），分岐方向予測精度（`dir`），そして間接ジャンプアドレス予測精度（`jr`）である．表4から，一般に`load`，`ldst`モデルにおいては分岐先アドレスの予測精度が低下していることが分かる．主な原因は間接ジャンプ先アドレスの予測精度が低下していることである．間接ジャンプ先アドレス予測精度は最も落ち込みの大きい`li`の場合で14.6%低下している．分岐予測精度の低下を引き起こす原因は，投機に失敗したロード命令と依存関係にある分岐命令が間違った分岐の履歴を記憶するためと考えられる．分岐予測精度がわずかに低下しただけでILPが著しく低下することはよく知られている．たとえば，分岐先アドレス予測精度が1%以上低下している`m88ksim`と`li`は，ILPもやはり著しく低下している．この問題を解決するには以下の2つの方法がある．1つはコンパイル時に間接ジャンプ命令を減少させることである[17]．この方法は本稿での検討の対象外とする．もう1つは，投機実行の頻度を減らして誤った分岐履歴を記憶する可能性を低下させることである．すべてのロード命令を投機の対象とするのではなく，ソースオペランドの揃っていないロード命令だけを投機の対象とすれば，

表4 分岐予測精度（命令破棄の場合）（%）
Table 4 Branch prediction accuracy (squashing) (%).

	base			load			store			ldst		
	addr	dir	jr	addr	dir	jr	addr	dir	jr	addr	dir	jr
go	78.80	80.50	97.31	78.53	80.54	92.57	78.81	80.50	97.31	78.55	80.54	92.97
m88	95.78	96.51	90.08	94.65	96.51	75.42	95.78	96.51	90.08	94.67	96.51	75.72
gcc	87.34	91.13	79.80	87.00	91.11	75.78	87.33	91.12	79.80	87.06	91.12	76.51
com	96.61	97.02	96.47	96.49	97.00	95.04	96.62	97.03	96.47	96.39	97.02	93.32
li	93.61	95.60	86.31	92.06	96.14	71.72	93.61	95.59	86.31	92.30	96.17	73.21
ijp	98.40	98.80	99.74	98.38	98.80	99.63	98.40	98.80	99.74	98.38	98.80	99.63
per	91.64	96.92	64.86	90.75	96.79	59.11	91.70	96.98	64.86	91.36	96.90	62.87
vor	92.48	94.72	97.48	91.70	94.71	91.23	92.47	94.71	97.48	91.87	94.69	92.76
tom	96.80	98.16	91.03	96.84	98.26	90.46	96.80	98.16	91.03	96.98	98.34	91.03
swi	98.36	98.36	99.97	98.36	98.36	99.97	98.36	98.36	99.97	98.36	98.36	99.97
su2	96.69	99.07	80.76	96.49	98.87	80.75	96.69	99.07	80.76	96.89	99.27	80.75
hyd	95.93	97.91	87.02	96.14	98.12	87.01	95.93	97.91	87.02	96.00	97.99	87.00
mgr	98.07	98.17	64.42	98.06	98.17	62.61	98.07	98.17	64.42	98.06	98.17	62.20
app	97.76	97.80	96.60	97.75	97.80	94.48	97.76	97.80	96.60	97.76	97.80	95.08
tur	98.19	98.21	99.83	98.10	98.13	99.71	98.19	98.21	99.83	98.14	98.17	99.74
aps	96.04	97.22	96.46	95.89	97.25	93.91	96.06	97.24	96.46	95.93	97.25	94.59
fpp	93.17	93.31	80.43	92.98	93.31	77.98	93.17	93.31	80.43	92.99	93.33	77.98
wav	98.42	98.43	99.99	98.42	98.43	99.99	98.42	98.43	99.99	98.42	98.43	99.99

表5 予測精度（命令破棄/消極的な投機の場合）
Table 5 Prediction accuracy (squashing/conservative).

	分岐予測精度（%）						アドレス予測精度（%）		
	load			ldst			load	ldst	
	addr	dir	jr	addr	dir	jr		ロード	ストア
go	78.60	80.52	93.91	78.61	80.52	94.03	73.81	73.95	97.36
m88	94.96	96.49	79.65	94.98	96.50	79.90	88.14	88.16	95.41
gcc	87.23	91.10	78.78	87.25	91.11	78.93	92.10	92.53	94.80
com	96.49	97.00	95.08	96.39	97.02	93.32	91.00	91.59	99.86
li	93.20	96.37	78.07	93.37	96.38	79.18	79.19	79.39	92.49
ijp	98.39	98.80	99.69	98.40	98.80	99.69	99.65	99.69	99.66
per	91.52	96.89	64.17	91.49	96.86	64.17	97.21	96.97	86.83
vor	92.34	94.72	96.32	92.35	94.72	96.48	93.80	94.66	94.27
tom	96.79	98.22	90.47	96.86	98.22	91.03	98.38	99.20	100.0
swi	98.36	98.36	99.97	98.36	98.36	99.97	98.41	98.71	100.0
su2	96.69	99.07	80.76	96.89	99.27	80.76	97.65	98.41	99.69
hyd	96.01	97.99	87.02	96.01	97.99	87.02	97.82	99.27	99.91
mgr	98.07	98.17	63.99	98.07	98.17	64.01	96.69	96.60	97.13
app	97.76	97.80	95.83	97.76	97.80	95.83	79.07	79.31	78.98
tur	98.11	98.13	99.80	98.14	98.17	99.80	96.76	96.91	96.61
aps	95.96	97.22	95.32	95.98	97.22	95.58	95.72	96.37	96.34
fpp	93.11	93.32	79.55	93.12	93.33	79.55	94.32	94.39	96.57
wav	98.42	98.43	99.99	98.42	98.43	99.99	95.97	95.97	99.93

投機実行の頻度を減少できる．前者の投機的実行を積極的な投機実行，後者を消極的な投機実行と呼ぶことにする．本項の以下の部分では，消極的な投機実行を検討する．

表5 の左側に消極的な投機実行を行った場合の分岐予測精度を示す．表4と比較すると分岐予測精度の顕著な向上が確認できる．特に間接ジャンプ先アドレスの予測精度の向上が著しい．表5の右側にはデータアドレス予測精度をまとめた．表3と比べると，いくつかのプログラムではわずかに予測精度が低下していることが確認できるが，概して大きな変化は見られない．**図16** はILPの変化を示している．各評価モデルのIPCは図15に示した `base` モデルのIPCで正規化してある．さて，消極的な投機実行の結果を積極的な投機実行の結果と比較しよう．積極的な投機実行を行うと，整数系プログラムの場合で，`load` モデルで平均6.1%，`ldst` モデルで平均4.4%ILPが低下している．浮動小数点プログラムの場合では，`load` モデルで平

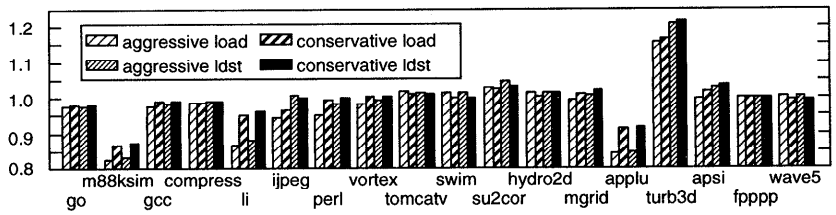


図 16 命令レベル並列度の変化（命令破棄/消極的な投機の場合）
Fig. 16 Performance improvement (squashing/conservative).

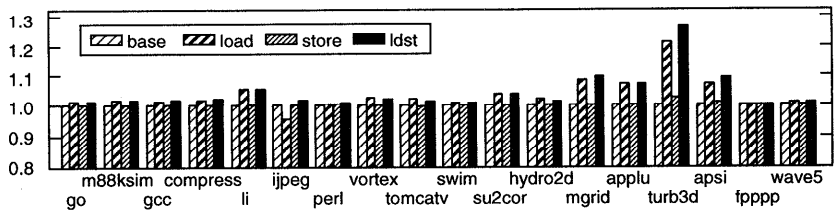


図 17 命令レベル並列度の変化（命令再発行の場合）
Fig. 17 Performance improvement (reissue).

均 0.7%，ldst モデルで平均 1.8%ILP が向上している．一方，消極的な投機実行を行うと，整数系プログラムの場合で，load モデルで平均 3.2%，ldst モデルで平均 2.5%ILP が向上している．浮動小数点プログラムの場合では，load モデルで平均 1.4%，load モデルで平均 2.4%ILP が向上している．一般的に，積極的な投機実行時に ILP の低下が著しいものほど，消極的な投機実行の効果が現れている．

6.2 命令再発行の効果

前節での検討によって，主にデータアドレス予測失敗によるペナルティと分岐予測精度の低下が原因で ILP が低下していることが分かった．すでに 4 章で述べたように，データアドレス予測失敗によるペナルティは命令再発行を採用することによって削減できる．そこで本節では，命令再発行を採用したときの分岐予測精度への影響を検証し，ペナルティの削減と合わせて命令再発行の ILP 向上に対する効果を評価する．投機実行は積極的な投機実行を行うことにする．

表 6 は，命令再発行を採用した場合の load，ldst モデルでの分岐予測精度を示している．表 4 と異なり，分岐予測精度が低下する現象はほとんど見られない．逆に予測精度が向上するケースすら見られる．予測精度が向上する理由は，データ依存の投機実行によって分岐先の確定が早くなるケースがあるためである．分岐予測精度が向上した結果，ILP も向上すると期待できる．

図 17 は，命令再発行を採用した場合の ILP の変化である．詳細な値は付録の表 7 にまとめた．図 17 のレイアウトは図 15 と同じである．図 15 の結果と異なり，load モデルでは ijpeg を除いて ILP が向上している．したがって，命令再発行を採用することでロード命令の投機実行がより有効になっていることが分かる．その結果 ILP は最大 21.0%向上している．命令破棄を採用したモデルと比較して著しい向上が見られるのは，m88ksim，li，applu である．図 15 で示したように，命令破棄のモデルではこれらのプログラムは著しく ILP を低下させている．しかし命令再発行モデルでは ILP が最大 7.2%向上している．このように，命令破棄を採用するとミスペナルティが甚大でプロセッサ性能に悪影響が現れるプログラムにおいて

表 6 分岐予測精度（命令再発行の場合）(%)
Table 6 Branch prediction accuracy (reissue) (%).

	load			ldst		
	addr	dir	jr	addr	dir	jr
go	78.81	80.51	97.32	78.81	80.50	97.31
m88	95.81	96.54	90.08	95.77	96.50	90.08
gcc	87.34	91.13	79.82	87.34	91.13	79.82
com	96.59	97.00	96.46	96.61	97.02	96.47
li	93.78	95.74	86.47	93.80	95.76	86.47
ijp	98.40	98.80	99.74	98.40	98.80	99.74
per	91.70	96.98	64.86	91.71	96.99	64.86
vor	92.46	94.70	97.48	92.44	94.68	97.48
tom	96.91	98.26	91.03	96.92	98.28	91.03
swi	98.36	98.36	99.97	98.36	98.36	99.97
su2	96.50	98.87	80.76	96.69	99.07	80.76
hyd	96.14	98.12	87.02	95.93	97.92	87.02
mgr	98.07	98.17	64.44	98.07	98.17	64.44
app	97.76	97.80	96.59	97.76	97.80	96.59
tur	98.11	98.13	99.83	98.15	98.17	99.83
aps	96.05	97.23	96.49	96.06	97.24	96.49
fpp	93.17	93.30	80.43	93.18	93.31	80.43
wav	98.42	98.43	99.99	98.42	98.43	99.99

も，命令再発行に基づく機構を用いれば，データ投機実行の効果を得ることができる．浮動小数点系のプログラムと比較して整数系のプログラムではデータ投機実行の効果が小さい．整数系プログラムの性能を改善するためには，データアドレス予測に基づく投機実行とデータ値予測に基づくデータ投機実行[23]を組み合わせることが有効であると期待されるが，最近の評価結果[24]はこの予想を裏付けている．`store` モデルでは，命令再発行を採用することによる効果は確認できない．これは，曖昧なデータ依存の投機に失敗する場合が非常に稀なためである．`ldst` モデルでは `load` モデルと同様に命令再発行は有効であり，ILP が最大 26.5%向上している．いくつかのプログラムでは ILP の向上率が `load` モデルの場合よりも小さい．これは RPT にストア命令も割り当てられるためにデータアドレスを予測可能なロード命令が減少するためと思われる．ストアデータアドレス予測による曖昧なデータ依存の投機的解消よりも，ロードデータアドレス予測によるロード命令の投機実行の方が，ILP 向上の効果が大きいためである．

以上より，命令再発行はデータ投機実行の効果を向上させる目的に非常に有効であることが確認できた．

7. むすび

本稿では，データアドレス予測を用いたロード命令の投機実行と曖昧なメモリ参照の投機的解消をシミュレーションにより評価した．またこれらの投機実行の効果を改善するために，命令再発行を実現するための現実的な機構を提案した．シミュレーションの結果，投機失敗時に命令を破棄してしまうと場合によっては ILP を低下させてしまうことが分かった．一方，命令の再発行を行うと，ほとんどすべてのプログラムで ILP を向上できることが分かった．ILP は最大 26.5%向上できた．その結果，ロード命令の投機実行，曖昧なデータ依存の投機的解消，そして命令再発行は，ILP の向上に有効であることを確認できた．

参考文献

1) Austin, T.M. and Sohi, G.S.: Zero-cycle loads: Microarchitecture support for reducing load latency, *Proc. 28th Ann. Int'l Symp. on Microarchitecture*, pp.82–92 (1995).
2) 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).
3) Chen, T-F. and Baer, J-L.: Effective hardware-based data prefetching for high-performance processors, *IEEE Trans. Comput.*, Vol.44, No.5, pp.609–623 (1995).
4) Conte, T.M., Menezes, K.N., Mills, P.M. and Patel, B.A.: Optimization of instruction fetch mechanisms for high issue rates, *Proc. 22nd Ann. Int'l Symp. on Computer Architecture*, pp.333–344 (1995).
5) Cortadella, J. and Llaberia, J.M.: Evaluation of A+B=K conditions without carry propagation, *IEEE Trans. Comput.*, Vol.41, No.11, pp.1484–1488 (1992).
6) Ellis, J.R.: *Bulldog: A compiler for VLIW architectures*, The MIT Press (1986).
7) Franklin, M. and Sohi, G.S.: ARB: A hardware mechanism for dynamic reordering of memory references, *IEEE Trans. Comput.*, Vol.45, No.5, pp.552–571 (1996).
8) Gabbay, F. and Mendelson, A.: The effect of instruction fetch bandwidth on value prediction, *Proc. 25th Ann. Int'l Symp. on Computer Architecture*, pp.272–281 (1998).
9) Gallagher, D.M., Chen, W.Y., Mahlke, S.A., Gyllenhaal, J.C. and Hwu, W.W.: Dynamic memory disambiguation using the memory conflict buffer, *Proc. Architectural Support for Programming Languages and Operation Systems VI*, pp.183–195 (1994).
10) Gonzalez, J. and Gonzalez, A.: Speculative execution via address prediction and data prefetching, *Proc. 11th Int'l Conf. on Supercomputing*, pp.196–203 (1997).
11) Huang, A.S., Slavenburg, G.S. and Shen, J.P.: Speculative disambiguation: A compilation technique for dynamic memory disambiguation, *Proc. 21st Ann. Int'l Symp. on Computer Architecture*, pp.200–210 (1994).
12) Hunt, D.: Advanced performance features of the 64-bit PA-8000, *Proc. COMPCON '95*, pp.123–128 (1995).
13) 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).
14) Lee, J.K.F. and Smith, A.J.: Branch prediction strategies and branch target buffer design, *IEEE Computer*, Vol.17, No.1, pp.6–22 (1984).
15) Lipasti, M.H.: Value locality and speculative execution, Ph.D. Dissertation, Depertment of Electrical and Computer Engineering, Carnegie Mellon University (1997).
16) Moshovos, A.I., Breach, S.E., Vijakumar, T.N., and Sohi, G.S.: Dynamic speculation

and synchronization of data dependences, *Proc. 24th Ann. Int'l Symp. on Computer Architecture*, pp.181–193 (1997).
17) Mueller, F. and Whalley, D.B.: Avoiding unconditional jumps by code replication, *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation*, pp.322–330 (1992).
18) Nicolau, A.: Run-time memory disambiguation: Coping with statically unpredictable dependencies, *IEEE Trans. Comput.*, Vol.38, No.5, pp.663–678 (1989).
19) 西本，勝野，木村：ロードアドレス予測による命令並列度の向上，情報処理学会研究報告，96-ARC-119, pp.49–54 (1996).
20) Rotenberg, E., Jacobson, Q., Sazeidas, Y. and Smith, J.: Trace processors, *Proc. 30th Ann. Int'l Symp. on Microarchitecture*, pp.138 148 (1997).
21) Sato, T., Fujii, H. and Suzuki, S.: Hiding data cache latency with load address prediction, *IEICE Trans. Information and Systems*, Vol.E79-D, No.11, pp.1523–1532 (1996).
22) Sato, T.: Data dependence speculation combining memory disambiguation with address prediction, *IPS Japan SIG Notes*, 97-ARC-125, pp.1–6 (1997).
23) 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).
24) 佐藤寿倫：データ値予測とアドレス予測を組み合わせたデータ投機実行，並列処理シンポジウム *JSPP '99* 発表予定 (1999).
25) Sazeides, Y., Vassiliadis, S. and Smith, J.E.: The performance potential of data dependence speculation & collapsing, *Proc. 29th Ann. Int'l Symp. on Microarchitecture*, pp.238–247 (1996).
26) 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).
27) Uhlig, R., Nagle, D., Mudge, T., Sechrest, S. and Emer, J.: Instruction fetching: Coping with code bloat, *Proc. 22nd Ann. Int'l Symp. on Computer Architecture*, pp.345–356 (1995).
28) Widigen, L., Sowadsky, E. and McGrath, K.: Eliminating operand read latency, *ACM SIGARCH Computer Architecture News*, Vol.24, No.5, pp.18–22 (1996).
29) Yeh, T-Y. and Patt, Y.N.: Branch history table indexing to prevent pipeline bubbles in wide-issue superscalar processors, *Proc. 26th Int. Symp. on Microarchitecture*, pp.164–175 (1993).

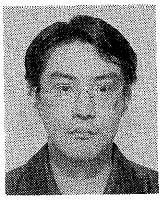
表 7 命令レベル並列度の向上率（%）
Table 7 Performance improvement (%).

	命令破棄			命令再発行		
	load	store	ldst	load	store	ldst
go	−2.59	0.01	−2.34	1.54	0.01	1.50
m88	−17.5	0.10	−16.8	2.07	0.10	1.79
gcc	−2.68	0.30	−1.85	1.61	0.29	2.03
com	−1.33	0.07	−0.84	2.15	0.07	2.37
li	−13.3	0.17	−11.8	5.39	0.17	5.36
ijp	−5.70	0.15	0.88	−4.48	0.15	2.19
per	−4.67	0.80	−1.58	0.82	0.80	1.28
vor	−1.63	0.05	−0.63	2.82	0.05	2.43
tom	1.66	−0.01	1.56	2.50	−0.01	1.50
swi	1.32	0.00	1.32	1.32	0.00	1.32
su2	2.78	0.01	4.37	3.96	0.01	4.09
hyd	1.43	0.04	1.47	2.41	0.04	1.40
mgr	−0.60	0.46	0.92	8.41	0.46	9.61
app	−15.6	0.43	−15.4	7.16	0.43	7.25
tur	15.7	2.79	21.0	21.0	2.79	26.5
aps	−0.25	1.14	2.95	7.01	1.13	9.14
fpp	0.02	0.00	0.01	0.08	0.00	0.06
wav	0.22	0.00	0.22	1.20	0.00	1.16

付　　録

表 7 に ILP の向上率をまとめる．向上率は増加した IPC を `base` モデルの IPC で割った値である．1 コラム目はプログラムの名前である．続く 3 コラムは命令破棄を採用したモデル，残りの 3 コラムは命令再発行を採用したモデルの結果である．3 コラムはそれぞれ左から順に，`load`, `store`, `ldst` モデルの向上率である．

(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．