アドレス名前替えによる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．シミュレーションによる評価により，本方式によるデータ値予測精度は平均 92.1%，最大 99.4% であることが確認されている．

Load Value Prediction using 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 related and the stored value is forwarded to the load instruction. For forwarding the data, *store-indexed value table (SIVT)* and *load-indexed store table (LIST)* are proposed. By indexing the LIST with 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 connect the store and load instructions, *data-indexed store table (DIST)* is proposed. From the experimental evaluation, we have found that this load value prediction mechanism has accuracy of 92.1% on average with a maximum of 99.4%.

1. はじめに

プロセッサの性能向上を妨げる要因の一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エア資源の不足により生じる．これはレジスタ名前替え[10),19)] 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，解消できる方法は未だない．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ふたつのタイプが存在する．ひとつはレジスタを介した依存関係であり，もう一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することができない．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 ら[11)] 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．本稿では，Lipasti らの提案とは異なった，単純なハードウエアで実現できるアドレス名前替えを用いたデータ依存の投機的実行法を提案する．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成る．2節で関連研究をまとめる．ロードデータ値予測手法を 3節で説明し，4節でその一実装例を紹介する．5節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 6節で評価する．最後に 7節で将来の課題を述べ，8節でまとめとする．

2. 関連する研究

Moshovos ら[12)] は，ストア命令とロード命令との間の参照アドレスの衝突を予測する手法を提案している．過去の参照アドレスの履歴を利用して予測する．衝突しないと予測されれば曖昧なメモリ参照の問題が解消できる．彼らの提案するハードウエアは連想検索を多用しており，非常に複雑なものとなっている．

ロードされるデータのデータアドレスを予測する方法にはすでに様々な提案がされている[1),2),4),6),14),17),20)]．これ

† 東芝マイクロエレクトロニクス技術研究所
Toshiba Microelectronics Engineering Laboratory
toshinori.sato@toshiba.co.jp

らの手法の目的は，ロード命令を通常よりも短いサイクルで実行することであり，必ずしもロード命令とその後続の命令を投機的に実行することではない．そのためこれらの多く[1),2),4),6),14)] は，ロード命令やその後続の命令を投機的に実行してはいない．また，投機的実行を行なっていても[17),20)]，ストアされるデータのデータアドレスを予測して活用してはいない．

最近，ストア命令のデータアドレス予測も研究され始めている[7),15),16)]． Gonzalez ら[7)] は，ロード命令だけでなくストア命令の投機的実行も提案している．ストア命令のデータアドレスを予測し，投機状態は address resolution buffer[5)] を用いて保持する． 15)， 16) においては，データアドレスを予測してメモリ参照の曖昧さを投機的に解消している．もし先行するストア命令の (予測された) データアドレスが後続のロード命令の (予測された) データアドレスと異なるなら，そ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．

Lipasti ら[11)] 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．彼らの提案では，最後に使用されたデータ値を繰り返し利用しており，データが異なる値を示す場合は効果が低下すると思われる．

3. ロードデータ値予測法

本節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．実現方法は次節で説明する．

3.1 1 レベルアドレス名前替え

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せずとも，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．われわれはこの方法を 1 レベルアドレス名前替え (1-level reference address renaming:RAR-L1) と呼んでいる．図 1に RAR-L1 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．

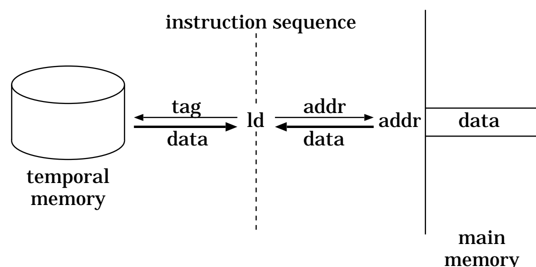

図 1 1 レベルアドレス名前替え

タグにはいろいろな候補があると思われるが，実装が楽な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 らの方法[11)] 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 らの方法は，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-L1 と言える．

データアドレスの予測手法も RAR-L1 とみなすことができる．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かわりにタグ `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-L1 とみ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，報告例ではロード命令アドレス[2),4),6),7),14)〜17),20)] が用いられている．

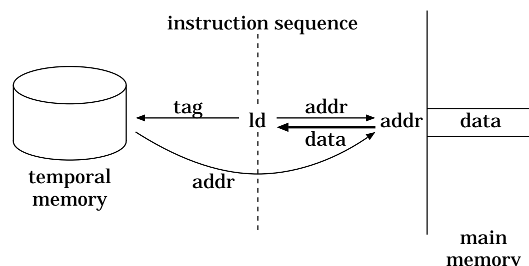

図 2 データアドレス予測

3.2 2 レベルアドレス名前替え

前節で説明した RAR-L1 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-L1 を行なっているに過ぎない．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．われわれ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)` は一旦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．

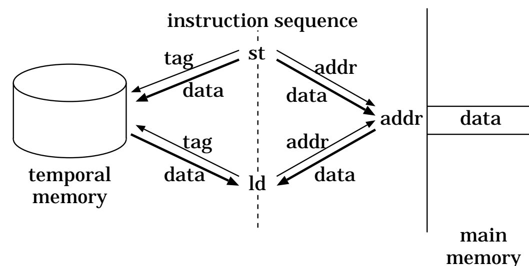

図 3 ストア命令情報を用いた RAR-L1

われわれはこの方法を 2 レベルアドレス名前替え (2-level reference address renaming:RAR-L2) 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-L1 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，こう呼んでいる．

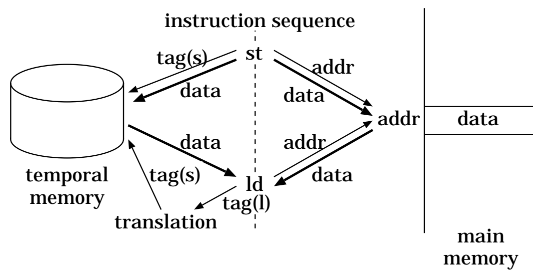

図 4　2 レベルアドレス名前替え

やはり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である．

4. ロードデータ値予測機構

本節で，前節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三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 節で各テーブルについて詳細に説明し，最後にロードデータ値予測機構を提案する．

4.1 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ない，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-L1 を行なっている．図 5に 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，バリッドビットから構成される．

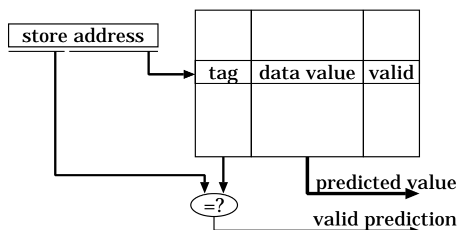

図 5　Store-Indexed Value Table

4.2 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なう．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-L1 を行なっている．図 6に 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．

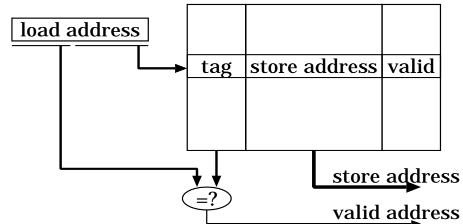

図 6　Load-Indexed Store Table

4.3 Data-Indexed Store Table

DIST は一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時，そのロード命令が読みだ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 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．

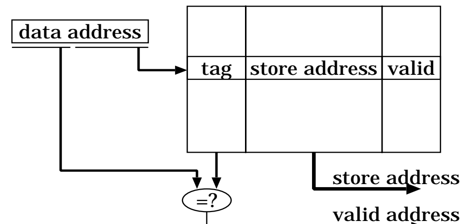

図 7　Data-Indexed Store Table

4.4 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．

この時予測の信頼性[8] 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時にはこのカウンタを +1 増加させ，失敗した時には −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 以上の時には有効な予測とみなして投機的実行を行なう．この 2 ビット飽和型カウンタは分

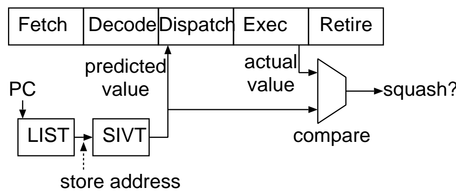

図 8 パイプライン図

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．

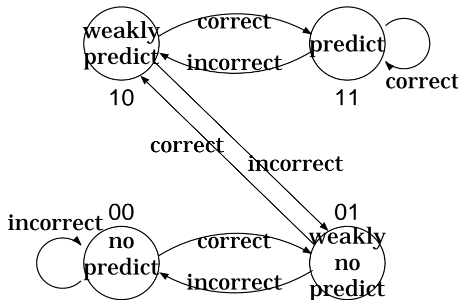

図 9 信頼性のための 2 ビット飽和型カウンタ

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なわれる．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．間違った予測をしたロード命令以降の命令はすべて破棄され，その命令から実行を再開する．このような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[18] を拡張した命令ウインドウを図10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)，予測ビット (Predicted)，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未だ得られない時には，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 情報を伴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．Content フィールドは，予測されたデータを保持する目的でも使用される．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．Predicted ビットは，Destination フィールドの Content に保持されている値が予測された値か否かを示している．Executed ビットか Predicted ビットのどちらか一方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．実際のデータがメモリから読み出されると，予測された値と実際の値は比較され，投機実行の成功 / 失敗が判定される．実際の値は Destination フィールドの Content に保存され，Predicted ビットはリセットされる．最後に Program Counter フィールドは，予測失敗からのプロセッサ状態の回復や，正確な割り込みを実現するために用いられる．

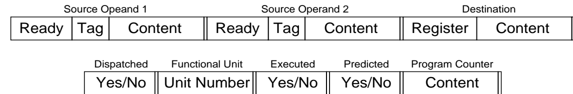

図 10 拡張された命令ウインドウのエントリ

例を用いて命令スケジューリングを説明する．図 11に示す命令シーケンスを用いる．理解を容易にするために，$f_1 - f_4$ の各オペレーションのソースオペランドは 1 つに限定する．実行レイテンシは，命令 I1 を除いて 1 と仮定する．命令 I1 は，例えばデータキャッシュミスを発生したロード命令であり，そのレイテンシを 4 と仮定する．図11の命令シーケンスには 2 つのデータ依存関係が存在する．一つは命令 I1 と命令 I3 の間であり，もう一つは命令 I3 と命令 I4 の間である．

$$\begin{aligned}&\texttt{I1:}\ r11 \leftarrow f_1(r1)\\&\texttt{I2:}\ r12 \leftarrow f_2(r2)\\&\texttt{I3:}\ r13 \leftarrow f_3(r11)\\&\texttt{I4:}\ r14 \leftarrow f_4(r13)\end{aligned}$$

図 11 命令シーケンスの例

図 12に図 11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 1 とする．命令のコミットは省略する．また，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する．レジスタ $r1$ とレジスタ $r2$ は，すでに利用可能になっているとする．最後に，命令 1 だけがロードデータ値を予測できるとする．以上の仮定に基づいて，命令スケジューリング

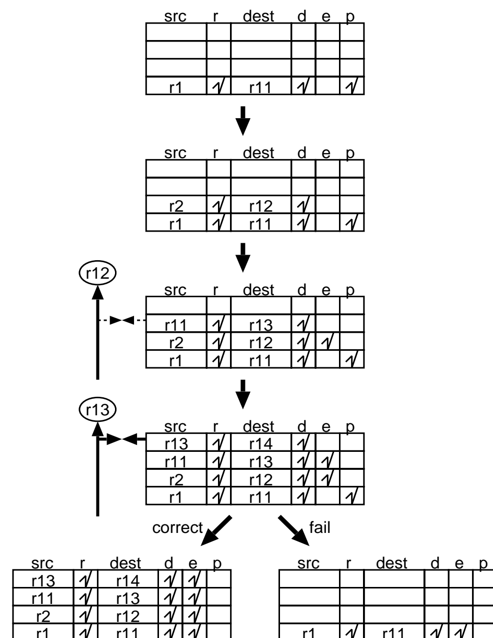

図 12 データ依存投機実行の例

☆ 18) では，ソースオペランドが得られない時に Ready ビットがセットされている．本稿では，後述の説明を理解容易にするために，オペランドが得られた時に Ready ビットがセットされるとした．

を説明する．まず最初のサイクルで，命令 I1 がイシューされる．ソースオペランドタグ $r1$ が Source Operand Tag フィールド (src)，デスティネーションレジスタタグ $r11$ が，Destination Register フィールド (dest) に保持される．r1 は利用可能であるので Ready ビット (r) がセットされる．さらに命令 I1 はディスパッチされ，Dispatched ビット (d) もセットされる．命令 I1 は $r11$ の値を予測し，Predicted ビット (p) をセットする．次のサイクルで，命令 I2 がイシューおよびディスパッチされる．つづいて，命令 I3 がイシューされる．命令 I3 のソースオペランド $r11$ を生成する命令 I1 に Predicted ビットがセットされているので，命令 I3 をディスパッチ可能である．同じサイクルで，命令 I2 は $r12$ に演算結果を書き戻し，Executed ビット (e) をセットする．$r12$ は命令 I3 が必要としている $r11$ に一致しないので利用されない．図で点線で表した矢印は，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なかったことを表している．次のサイクルでは命令 I4 がイシューされる．命令 I3 が演算を終了し $r13$ が利用可能になっているので，命令 I4 は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．次のサイクルには 2 通りの場合が考えられる．命令 I1 の予測が成功した場合と，失敗した場合である．$r11$ の予測に成功した場合，命令 I1 は Predicted ビットをリセットし，Executed ビットをセットする．同じサイクルで命令 I4 が演算を終了し，やはり Executed ビットをセットする．以上でシーケンスは終了する．一方 $r11$ の予測に失敗した場合は，命令 I2−I4 は破棄されなければならない．命令 I1 の Executed ビットをセットし，プロセッサは命令 I2 のフェッチから実行を再開する．したがってこの例では以下のミスペナルティを被ることがわかる．すなわち，上述した間違った投機実行による 4 サイクル，命令を破棄するためのサイクル，そして命令を再フェッチするためのサイクルである．

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なわれる．ストアデータアドレスは計算されている必要はない．この時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なわれる．この時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 13に示すように行なわれる．命令フェッチ時にロード命令アドレスで LIST を参照し，LIST にその命令アドレスが登録されていなかった際に LIST への登録が行なわれる．その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．

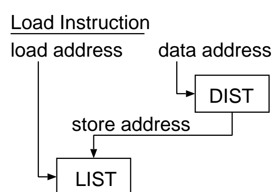

図 13 LIST の登録操作

残る問題は，いかにしてストア命令とロード命令の同期をとるかということである．LIST から命令アドレスが得られたストア命令が命令ウインドウ内に滞在している場合，SIVT から得られるデータが最新である保証はない．これを保証するために，そのストア命令が保持されているエントリの Source Operand Ready ビットをチェックし，SIVT から得られるデータが最新であることを確認する必要がある．もし Ready ビットがセットされていれば，メモリにストアされようとしているデータはすでにレジスタから読み出されており，SIVT はそのデータで更新されている．Ready ビットがリセットされている場合は，ロード命令は Ready ビットがセットされるまで SIVT の参照を中断すればよい．

以上のようにして，DIST により，ひと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なって，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-L2 を行なうことで，データアドレスを計算せずともロード命令は必要なデータを獲得できる．

5. 評 価 環 境

本節では，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

提案手法の評価には，SimpleScalar ツールセット[3] を用いた．SimpleScalar 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・オーダー実行を行なうスーパースカラ・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レジスタ更新ユニット[18] を採用している．9) での結果に基づいて，基本モデルの構成は表 1にまとめた通りとした．LIST，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 1024 とした．SIVT はダイレクトマップ方式とし，エントリ数は 64 とした．ロードデータの予測に失敗した際は，予測ミスを犯したロード命令以降の命令は全て破棄され再びフェッチされる．破棄に要するミスペナルティは 3 サイクルと仮定した．

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．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[3]．各プログラムは終了まであるいは最初の一億命令を実行した．

表 2 ベンチマークプログラム

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

6. 評 価 結 果

本節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，つづいてプロセッサの性能向上について述べ

表 1　プロセッサモデル

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, ID 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
LIST,DIST	各々 1024 エントリ, ダイレクトマップ
SIVT	64 エントリ, ダイレクトマップ

る．性能向上の尺度には committed instruction per cycle (IPC) を用いた．

6.1 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関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．表 3に現れているように予測精度は非常に高く，平均で 92.1%，最大 99.4% であることがわかる．

表 3　予測精度と各テーブルのヒット率

プログラム	(%) 予測精度	(%) ヒット率	
		LIST	SIVT
099.go	84.88	66.8	47.9
124.m88ksim	97.00	57.5	57.7
126.gcc	83.58	48.0	63.1
129.compress	96.60	57.8	62.5
130.li	94.73	71.1	41.5
132.ijpeg	99.35	68.0	98.2
134.perl	82.36	66.3	60.9
147.vortex	98.50	51.2	52.3
平均	92.13	60.8	60.5

表 3の右半分には， LIST と SIVT のヒット率をまとめた． 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 LIST と SIVT はともに概ね 60% のヒット率であることがわかる．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 35% 程度と予想されるが，実際にはそれよりもかなり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，結果として予測精度が高くなっている．

図 14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，予測が行なわれなかった割合 (白)，予測が誤った割合 (黒)，予測が正しかった割合 (灰) を表している． `132.ijpeg` を除いて約 2 割のロード命令が予測されたデータを利用していることがわかる．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．

6.2 プロセッサ性能

図 15に提案手法を用いた際のプロセッサ性能の変化を表す．図の縦軸は，投機実行モデルでの IPC の変化を基本

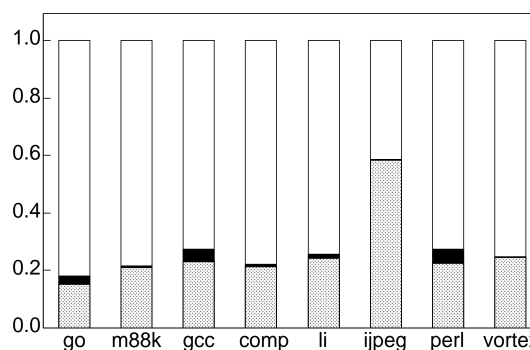

図 14　データ値予測の割合

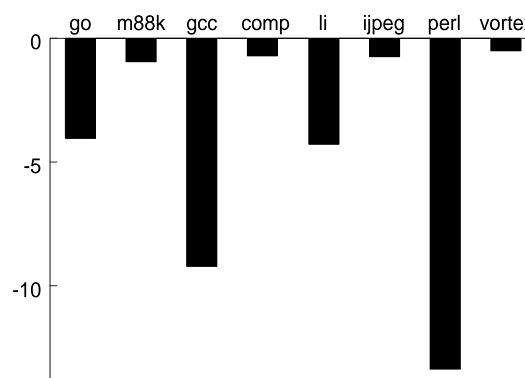

図 15　(%) プロセッサ性能の変化

モデルの IPC で割った割合である．したがって，値が大きいほど性能向上が大きい．しかし，全てのベンチマークプログラムにおいて，投機実行を行なわない場合よりもプロセッサの性能が低下している．原因のひとつはミスペナルティが大きいことと思われる． 4.4節で説明したように，データ値の予測に失敗した時のミスペナルティは非常に大きい．前節で紹介した高い予測精度であっても，予測ミスを犯した時のオーバー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 15を比べると，予測精度とプロセッサ性能の低下との間には相関があることがわかる．すなわち，予測精度の低いものほど，性能低下が著しい．したがって，この問題を解決しデータ依存投機を効果的にするために，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．これ

に関しては7節で詳細に説明する．

一方，分岐予測精度の低下もプロセサの性能低下を引き起こしていると考えられる．表4に分岐予測精度の変化をまとめる．左半分が基本モデルの分岐予測精度であり，右半分が投機実行モデルの分岐予測精度である．それぞれ，分岐先アドレスの予測精度 (addr)，分岐方向の予測精度 (dir)，間接ジャンプの予測精度 (jr) から構成されている．addr と dir には無条件分岐の結果も含まれている．概して，投機的実行を行なうことで分岐予測精度の低下が確認できる．特に間接ジャンプ予測精度の低下が著しい．一般に，数％の分岐予測精度の低下であっても，プロセッサの性能を著しく低下させるといわれている．この問題は最適化コンパイラにより解消可能である．最適化コンパイラによって実行されるジャンプ命令の数を削減できれば[13]，分岐予測精度の低下も防ぐことが出来ると思われる．

表 4 (%) 分岐予測精度

プログラム	基本モデル			投機実行モデル		
	addr	dir	jr	addr	dir	jr
go	78.8	80.5	97.3	77.9	80.5	82.7
m88ksim	95.8	96.5	90.1	95.7	96.5	88.7
gcc	82.3	87.4	79.3	81.4	87.4	70.1
compress	96.6	97.0	96.5	96.6	97.0	96.1
li	93.6	95.6	86.3	92.8	95.6	80.6
ijpeg	98.4	98.8	99.7	98.4	98.8	99.7
perl	91.6	96.9	64.9	90.1	96.8	53.8
vortex	92.5	94.7	97.5	92.3	94.7	96.7

7. 今後の課題

前節で述べたように，本方式によるデータ依存の投機実行は期待した効果が得られていない．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．

ミスペナルティの軽減には命令の再発行[7),11)] が有効であると考えている．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．ところが我々の知る限り，命令再発行の具体的な実装に関する報告は未だない． 7)， 11) 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，コンセプト通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データ依存関係にある全ての命令を 1 サイクル以内に検出しなければならないからである．そこでわれわれは，命令再発行の現実的な機構を検討した．

Source Opeand 1			Source Operand 2			Destination	
Ready	Tag	Content	Ready	Tag	Content	Register	Content

Dispatched	Functional Unit	Executed	Predicted	Reissued	Program Counter
Yes/No	Unit Number	Yes/No	Yes/No	Yes/No	Content

図 16 命令再発行を可能にする命令ウインドウ

図 10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 16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．

4.4節で説明した命令スケジューリングと同様に，予測されたデータ値は Destination Content フィールドに保持される．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．同時に予測の成功 / 失敗を表す信号も放送される．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時には，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，データ値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，タグが一致し且つ Dispatched ビットがセットされた命令が再発行される．こうして，再発行されなければならない命令が順に検出される．本機構は従来の命令ウインドウと比較して再発行信号が一本増えただけであり，ハードウエアの増大とサイクルタイムの延長を防いでいる．一方で同じ命令が複数回発行されるが，機能ユニットの数の制限により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問題ないと考えている．

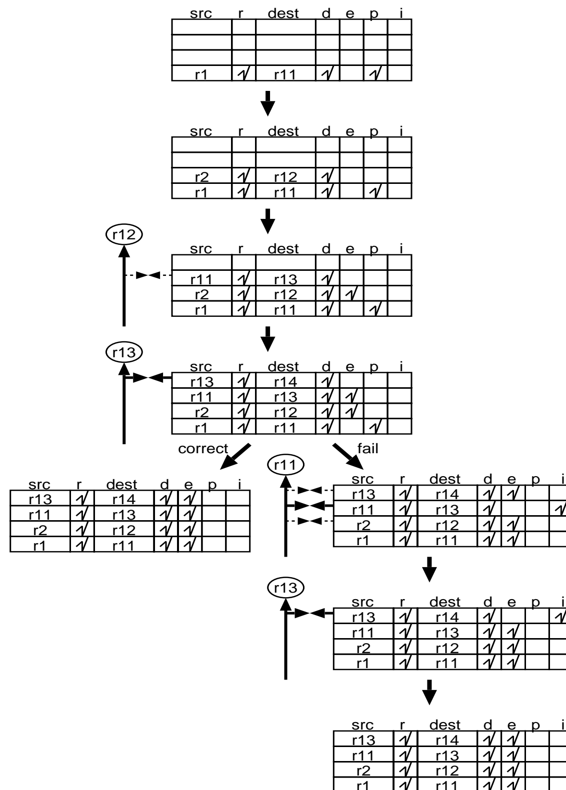

図 17 命令再発行の例

図 17に命令再発行の例を示す．図 12で用いた同じ仮定を用いる．命令 I1 の実行が終了するまでは図 12での説明と同様であるので省略し， 5 サイクル目から説明する．命令 I1 での予測に成功した場合，命令 I1 は Predicted ビットをリセットし， Executed ビットをセットする．同じサイ

クルで命令 I4 が演算を終了し，やはり Executed ビットをセットする．以上でシーケンスは終了する．一方予測に失敗した時には，再発行シグナルが放送され，命令の再発行が開始される．タグの一致した命令 I3 は間違ったソースオペランドを用いてすでにディスパッチされているので，再発行の対象となり Reissued ビット (i) がセットされる．命令 I3 はディスパッチされ Dispatched ビットがセットされる．次のサイクルで命令 I3 は終了し，Executed ビットをセットして Reissued ビットをリセットする．同時に再発行シグナルが放送される．前のサイクルと同様にして命令 I4 が再発行の対象として検出される．次のサイクルで命令 I4 は終了し，この命令シーケンスは 7 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なわない場合に必要なサイクル数と同じである☆． 現実には機能ユニット数の制約などにより，予測失敗によるペナルティが生じると考えられる．

以上のように，命令の再発行は規模の小さなハードウエアを用いて，且つサイクルタイムに影響を及ぼさないで実現可能である．現在，この機構を用いた場合のデータ依存投機実行の効果を評価中である．

8. まとめ

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 レベル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．

サイクルレベルの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．シミュレーションにより，データ値の予測精度は平均 92.1%，最大 99.4% であった．残念ながらプロセッサの性能向上は確認できなかった．原因は予測ミスによるペナルティが非常に大きいことと，分岐予測精度が低下することであると考えられる．

ミスペナルティの軽減の目的で，命令の再発行を行なう機構の実現方法を検討した．これまで命令再発行はコンセプトの提案だけがされており，我々の知る限りでは実現法に関する報告はなかった．現在，命令再発行の機構を用いてデータ依存投機実行の効果を評価中である．

参考文献

1) T.M.Austin, D.N.Pnevmatikatos, G.S.Sohi: Streamlining Data Cache Access with Fast Address Calculation, *Proc. of 22nd Ann. Int'l Symp. on Computer Architecture*, pp.369-380 (1995).
2) T.M.Austin, G.S.Sohi: Zero-Cycle Loads: Microarchitecture Support for Reducing Load Latency, *Proc. of 28th Ann. Int'l Symp. on Microarchitecture*, pp.82-92 (1995).
3) D.Burger, T.M.Austin: The SimpleScalar Tool Set, version 2.0, *ACM SIGARCH Computer Architecture News, vol.25, no.3*, pp.13-25 (1997).
4) R.J.Eickemeyer, S.Vassiliadis: A Load-Instruction Unit for Pipelined Processors, *IBM Journal of Research and Development, vol.37, no.4*, pp.547-564 (1993).
5) M.Franklin, G.S.Sohi: ARB: A Hardware Mechanism for Dynamic Reordering of Memory References, *IEEE Trans. on Computers, vol.45, no.5*, pp.552-571 (1996).
6) M.Golden, T.N.Mudge: Hardware Support for Hiding Cache Latency, *Technical Report CSE-TR-152-93*, Department of Electrical Engineering and Computer Science, University of Michigan (1993).
7) J.Gonzalez, A.Gonzalez: Speculative Execution via Address Prediction and Data Prefetching, *Proc. of 11th Int'l Conf. on Supercomputing*, pp.196-203 (1997).
8) E.Jacobsen, E.Rotenberg, J.E.Smith: Assigning Confidence to Conditional Branch Predictions, *Proc. of 29th Ann. Int'l Symp. on Microarchitecture*, pp.142-152 (1996).
9) S.Jourdan, P.Sainrat, D.Litaize: Exploring Configuration of Functional Units in an Out-of-Order Superscalar Processor, *Proc. of 22nd Ann. Int'l Symp. on Computer Architecture*, pp.117-125, (1995).
10) R.M.Keller: Look-Ahead Processors, *ACM Computing Surveys, vol.7, no.4*, pp.177-195 (1975).
11) M.H.Lipasti, C.B.Wilkerson, J.P.Shen: Value Locality and Load Value Prediction, *Proc. of Architectural Support for Programming Languages and Operation Systems VII*, pp.138-147 (1996).
12) A.I.Moshovos, S.E.Breach, T.N.Vijakumar, G.S.Sohi: Dynamic Speculation and Synchronization of Data Dependences, *Proc. of 24th Ann. Int'l Symp. on Computer Architecture*, pp.181-193 (1997).
13) F.Mueller, D.B.Whalley: Avoiding Unconditional Jumps by Code Replication, *Proc. of ACM SIGPLAN Conf. on Programming Language Design and Implementation*, pp.322-330 (1992).
14) T.Sato, H.Fujii, S.Suzuki: Hiding Data Cache Latency with Load Address Prediction, *IEICE Trans. on Information and Systems, vol.E79-D, no.11*, pp.1523-1532 (1996).
15) T.Sato: Data Dependence Speculation Combining Memory Disambiguation with Address Prediction, *Proc. of 10th Summer United Workshop on Parallel, Distributed, and Cooperative Processing(IPSJ SIG Notes 97-ARC-125-1)*, pp.1-6 (1997).
16) T.Sato: Speculative Resolution of Ambiguous Memory Aliasing, *Proc. of Int'l Workshop on Innovative Architecture for Future Generation Parallel Processors and Systems*, (1997) .
17) Y.Sazeides, S.Vassiliadis, J.E.Smith: The Performance Potential of Data Dependence Speculation & Collapsing, *Proc. of 29th Ann. Int'l Symp. on Microarchitecture*, pp.238-247 (1996).
18) G.S.Sohi: Instruction Issue Logic for High-Performance, Interruptible, Multiple Functional Unit, Pipelined Computers, *IEEE Trans. on Computer, vol.39, no.3*, pp.349-359 (1990).
19) R.M.Tomasulo: An Efficient Algorithm for Exploiting Multiple Arithmetic Units, *IBM Journal, vol.11*, pp.25-33 (1967).
20) L.Widigen, E.Sowadsky, K.McGrath: Eliminating Operand Read Latency, *ACM SIGARCH Computer Architecture News, vol.24, no.5*, pp.18-22 (1996).

☆ 紙面の制約によりデータ依存の投機実行を行なわない場合の命令スケジューリングの例示は省略した．