NAIST-IS-MT0651125

修士論文

パイプラインプロセッサ上での MIN-TAGE 分岐予測器の性能評価

牟田口　公洋

2007 年 2 月 7 日

奈良先端科学技術大学院大学
情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
修士 (工学) 授与の要件として提出した修士論文である。

牟田口　公洋

審査委員：

中島 康彦 教授　（主指導教員）
藤原 秀雄 教授　（副指導教員）
山下 茂 准教授　（副指導教員）
中田 尚 助教　（副指導教員）

パイプラインプロセッサ上での MIN-TAGE 分岐予測器の性能評価*

牟田口　公洋

内容梗概

プロセッサの性能を向上させるため，これまでにさまざまな分岐予測器が提案されてきた．近年はプロセッサのマルチコア化が一般的になっており，小規模で電力効率の高い設計が要求されている．従来の分岐予測器はこの要求に適合しなくなってきており，そこで小規模・省電力で予測回路の遅延が小さい MIN-TAGE 予測器が提案されている．MIN-TAGE 予測器は従来予測器のクリティカルパスである予測テーブルのハッシュ値計算から予測結果の生成を効率的にパイプライン化することにより，小さい遅延でありながら，容量 16KB でミス率 0.45 パーセントを下回る位の高い予測精度を示した．しかし，その評価はトレースレベルのシミュレータ上で実行されており，実際のプロセッサ上に実装した場合とでは予測テーブルやレジスタ等の更新のタイミングが異なってくると考えられる．本論文では MIN-TAGE 予測器をパイプラインプロセッサ上に実装し，ベンチマークプログラムで性能を計測することによってどのように挙動が変化するのかについて調べた．その結果パイプラインプロセッサ上に実装すると，MIN-TAGE 予測器のインデックスや予測テーブルを更新するタイミングの違いによって，トレースレベルのシミュレータによる結果よりも平均で約 0.4% ミス率が上昇し，性能が低下することが分かった．

キーワード

*奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 修士論文, NAIST-IS-MT0651125, 2007 年 2 月 7 日.

分岐予測器，MIN-TAGE 予測器，マルチコア，パイプラインプロセッサ，トレースレベルシミュレータ

Performance evaluation of MIN-TAGE branch predictor on processor*

Masahiro Mutaguchi

Abstract

To improve the performance of processors, various branch predictors have been proposed. Small scale and power-efficient processors are recently required because multi-core processors have been popular. When these condition, MIN-TAGE branch predictor that suits complexity-effective implementation and feasible prediction latency has been proposed. The predictor is said to be able to reduce prediction latency in contrast to the conventional branch predictors and save hardware costs. It is also said the accuracy is high. However, the evaluation of MIN-TAGE predictor has been done based on a trace level simulator. When it is implemented on a real processor, there are differences in timing of updating register and prediction table. This paper shows the detailed implementation of MIN-TAGE predictor on a real processor and its performance using clock accurate simulator. The differences in timing of updating register and predictor table result in performance degradation by up to 0.5 percent as compared to the result of a trace level simulator.

Keywords:

branch predictor, MIN-TAGE predictor, multi-core, pipeline processor, trace level simulator

*Master's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT0651125, February 7, 2007.

目 次

図 目 次

表 目 次

1. はじめに

近年のプロセッサは性能向上のため，複数の命令を順次同時に処理するパイプライン処理や命令レベルの並列性を利用し, 同時に複数の命令を実行するスーパースカラ処理を行っている．そのため，パイプライン実行が滞らないようにするために，常に命令を供給することが求められる．このため，分岐予測はプロセッサのパフォーマンス全体に大きな影響を与えるほど重要な技術となってきており，高精度な分岐予測器が常に求められている．これまでにも多くの分岐予測手法が提案されている．例えば gshare 分岐予測器 [1] のようなシンプルで既存のプロセッサに広く採用されているものから，パーセプトロン分岐予測器のような，高性能だが複雑で実際のプロセッサではまだ採用されていないものまでいろいろとある．

従来手法の多くは大規模なコア上に実装することが想定されており，十分なメモリ容量と大きな予測遅延を許容していた．しかし，近年は 1 つの LSI 上に複数のプロセッサコアを搭載するマルチコア化が大きな流れになっており，その影響で小規模でかつ電力効率の高い予測器が求められている．従来の分岐予測器はこれらの要求に適合しなくなってきており，そこで石井らによって小規模，省電力コア向けに，小資源での分岐予測手法として MIN-TAGE 分岐予測器 [2] が提案された．この MIN-TAGE 分岐予測器は高い予測精度だが複雑で予測遅延の大きい TAGE 分岐予測器 [3] を元に効率的にパイプライン化し，TAGE 分岐予測器のクリティカルパスであった予測テーブルのインデックス用ハッシュ値計算，予測テーブルアクセス，および予測結果の生成という一連の動作を 3 サイクルにわけて行うことで予測遅延を小さくしている．予測精度も非常に高く，メモリの容量が 1KB から 16KB 全てにおいて gshare 予測器はもちろん，gshare 予測器よりも高性能なハイブリッド型の予測器である bimode 予測器 [4] や 2BC-Gskew 予測器, [5] また Championship Branch Prediction において高い評価を得た TAGE 予測器や GEHL 予測器 [6] を上回る精度を達成している．ところが，これらは実際のプロセッサ上で行われた評価ではなくトレースレベルのシミュレータ上で評価されている．実際のプロセッサはトレースレベルのシミュレータとは挙動が異なっており，例えばトレースレベルのシミュレータの場合，毎サイクル分岐命令がフェッチされることを前提に処理が進むのに対し，実際のプロセッサでは分岐命令以外

の足し算やロードといった命令がフェッチされてくる．また，MIN-TAGE 予測器は予測部分のみの 3 段パイプラインを想定しており，予測値を生成してから予測テーブル更新の間の遅延がない．しかし高性能なプロセッサはさらにパイプライン長が長く，予測値を生成してから予測テーブルに結果を格納するまで数サイクルを要する．

本研究では MIN-TAGE 予測器を実際のパイプラインプロセッサ上に実装し，トレースレベルのシミュレータと比較したときにどのように性能が変化するのか，ベンチマークプログラムを使用してミス率を計測することで性能を評価する．さらに，MIN-TAGE 予測器は予測テーブルを構成する複数のカウンタやテーブルのエントリ選択で使用する Enhanced Folded Index に含まれる分岐履歴の長さ，分岐パス情報の長さといったパラメータが存在する．そこで，これらのパラメータをいろいろと変化させた上で予測ミス率の変化にいて調べ，考察する．以降では，2 章で分岐予測の必要性について述べ，MIN-TAGE 予測器を構成する予測器について説明する．そして本研究で対象である MIN-TAGE 予測器について説明する．3 章では分岐予測に関して，パイプラインプロセッサの場合とトレースレベルのシミュレータの場合との違いについて説明する．4 章では MIN-TAGE 予測器を実装するプロセッサの構成について述べ，プロセッサへの実装方法について述べる．5 章では MIN-TAGE 予測器を構成する各種パラメータの影響について述べ，そしてパイプラインプロセッサ上に実装した場合とトレースレベルのシミュレータ上に実装した場合とを比較した結果を示し，考察を述べる．そして 6 章では本論文のまとめについて述べる．

2. 分岐予測

本章ではまず分岐予測の必要性についてを述べる．そして，予測器が予測する際に重要な要素となる予測カウンタについて説明し，次に過去に考案された予測器である bimodal 予測器，gshareE 予測器，MIN-TAGE 予測器のもとになった TAGE 予測器，そして最後に MIN-TAGE 予測器について構成および予測値を生成する方法を説明する．

2.1 必要性

分岐予測とは条件分岐命令が成立（Taken）か，不成立（Not Taken）かを分岐命令が実際に実行されて結果が判明する前に予測することである．近年のプロセッサは与えられた命令列に対してフェッチ，デコード，実行，メモリの読み書き，結果のレジスタ書き込みという一連の動作を順次パイプライン処理している．性能を維持するためには常にパイプラインに命令を供給しつづける必要があるが，命令列の中に条件分岐命令が存在すると，命令のフェッチ先が決まらず，分岐命令が実行されて分岐先が判明するまで待つ必要がある．そうすると各ステージはすることが無くなりパイプライン処理が滞ってしまう．近年の高性能なプロセッサは，パイプラインが深くなるとともに，数命令を同時にフェッチし処理するスーパースカラ方式を採用することにより高い性能を達成しているため，命令の供給が滞ると大きな性能の低下となる．そのため，分岐命令の結果が判明するのを待たずに命令をフェッチしてくる必要があるが，間違った方向の命令をフェッチしてくると間違いと判明した時点で破棄しなければならず，大きなペナルティとなる．そこでより正確な方向の命令をフェッチしてくるために分岐予測の技術が重要となってくる．

分岐予測には静的予測と動的予測がある．静的予測は命令が実行される前にあらかじめ振舞が決定されているもので，例えばジャンプ先が現在の位置よりも後方だと成立し，前方だと成立しない BTFN（Backward Taken Forward not Taken）という方法がある．これは，後方へのジャンプはループであることが多く，かつループは多数回実行されることが多いという特徴を利用したものである．一方動

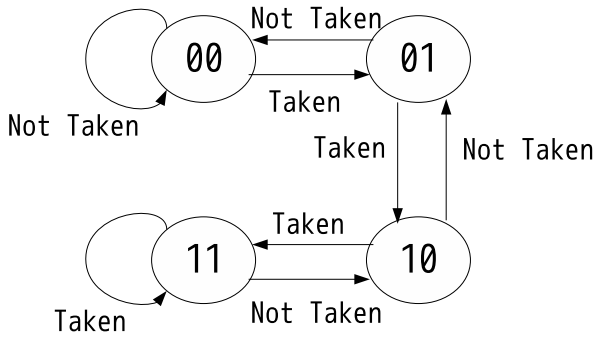

図 1 2 ビット飽和カウンタ

的予測は命令の実行時に予測の振舞を動的に変化させ決定するものである．本研究ではこの動的予測器を対象にしている．以下に過去に考案された動的分岐予測器である bimodal 予測器，gshare 予測器，TAGE 予測器，MIN-TAGE 予測器そしてこれらの予測器の基本要素である予測カウンタについて説明する．

2.2 予測カウンタ

最も単純な構成の予測器は 2 ビット飽和カウンタである．このカウンタは例えば分岐が Taken のときはカウンタ値を増やし，Not Taken のときは減らす．そして，00，11 でそれぞれ飽和する．予測するときはカウンタ値を見て例えば 11，10 のときは Taken，01，00 のときは Not Taken と予測する（図 1）．この 2 ビット飽和カウンタは過去に Taken（Not Taken）と続いたら次の分岐命令も Taken（Not Taken）することを期待して予測するものである．複雑な構成の分岐予測器でもこの飽和カウンタを予測テーブルの中に保持し，予測する際にカウンタ値を参照して予測値を生成するものが多い．本研究で対象としている MIN-TAGE 予測器も予測テーブル中に飽和カウンタを保持している．

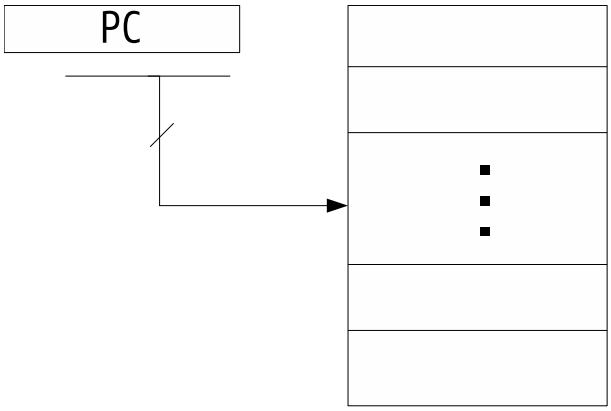

図 2 bimodal 予測器

2.3 bimodal 予測器

bimodal 予測器は 1981 年に Smith によって考案された予測器である．これは，2 ビット飽和カウンタを格納した予測テーブル（PHT：Pattern History Table）で構成されている．予測は分岐命令のアドレスの一部をインデックスとして PHT にアクセスし，読み出したカウンタの値をみて予測値を生成する（図 2）．これは分岐命令が Takne もしくは Not Taken のどちらかの方向に強く偏っているときに効果が高い．しかし，予測テーブルが小さいと他の分岐命令とエントリの競合が発生し，予測精度が低下する．

2.4 gshare 予測器

gshare 予測器は 1993 年に McFarling によって考案された予測器である． PHT と過去の分岐の結果を記録した GHR（Global History Register）で構成されている．GHR はシフトレジスタでできており，分岐結果が判明するとレジスタをシフトして，Taken のときは例えば 1 を Not Taken のときには 0 をレジスタに格納する．予測値の生成は分岐命令のアドレスの一部と GHR との排他的論理和（XOR）をとってインデックスを生成し，PHT からカウンタ値を読み出して行う（図 3）．GSHARE 予測器は同じアドレスでも直前の分岐命令の結果によってアクセスす

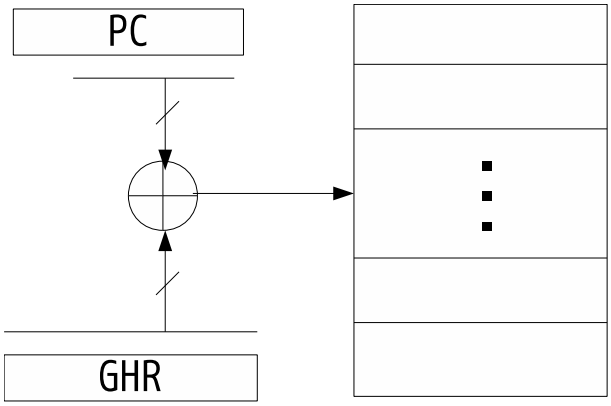

図 3 gshare 予測器

る PHT のエントリを変化させており，分岐間の相関性を利用することで高いヒット率を実現させている．

2.5 TAGE 予測器

TAGE 予測器の全体構成を図 4 に示す．この例では 5 つのテーブルで構成されており，分岐命令のアドレスをインデックスとしてアクセスする T0，そして分岐命令のアドレスとさまざまな長さの GHR のハッシュ値をインデックスとしてアクセスする Ti($1 \leq i < 5$) で構成される．T0 は 2 ビットのカウンタを保持し，Ti($1 \leq i < 5$) はタグと 3 ビットの符号付カウンタ ctr，そして 2 ビットの符号なしカウンタ u で構成される．カウンタ ctr は予測値の生成に使用し，カウンタ u はエントリが有用かどうかを示す．各テーブルで使用する GHR の長さ L(i) は次の式を使用して決める．

$$\begin{aligned} L(0) &= 0 \\ L(i) &= \alpha^{i-1} * L(1) \end{aligned}$$

例えばテーブルが 8 個の場合で $\alpha = 2$，L(1) = 2 とすると L(i)($0 \leq i < 8$) はそれぞれ 0,2,4,8,16,32,64,128 となる．Ti($1 \leq i < 5$) では長い履歴長を利用して予測す

るため，インデックスと同じビット長の CSR(Circular Shift Register) を用意する．この CSR へは GHR と同様に分岐履歴を格納していく．そして GHR のビット長がインデックスのビット長を超えたときは超えたビットを最近の履歴である h[0] と XOR して CSR の LSB（Least Significant Bit）に格納する．GHR が 40 ビットでインデックスが 10 ビットの例を図 5 に示す．図 5 では R9 と h[0] を XOR し，また GHR40 ビット以上の影響をキャンセルするために h[40] も XOR して CSR に格納する．予測テーブルのインデックスは分岐アドレスと CSR の XOR で生成する．GHR40 ビット，インデックス 10 ビットとすると $pc[0:9] \oplus h[0:9] \oplus h[10:19] \oplus h[20:29] \oplus h[30:39]$ となる。予測値の生成は，全ての予測テーブルをテーブル毎のインデックスを使用して同時にアクセスし，T0 は常に予測値を出力するが Ti($1 \leq i < 5$) はタグが一致したものだけが予測値を出力する．そして最終的な予測結果を出力するのはタグが一致したものの中で最も長い履歴長を使用したテーブルである．もしタグが一致するテーブルがないときは T0 の予測値を最終的な予測結果とする． 次に予測テーブルの更新方法を説明する．最終的な予測値を出したテーブルの次に長い GHR を保持し，タグがヒットしたテーブルの予測値を altpred とする．つまり，T2 と T4 のタグが一致し，T1 と T3 のタグが不一致のとき T2 の予測値が altpred となる．まず u カウンタの更新方法は altpred が最終的な予測結果と異なるときに予測結果を出力したテーブルの u カウンタを更新する．予測結果が正しいときは u カウンタをインクリメントし，間違ったときはデクリメントする．さらに u カウンタの最上位ビットは周期的に下位ビットへシフトし 0 ビットにリセットされる．そのようにすると使用されてないエントリにもかかわらず u カウンタの値が大きいまま置き換えの対象とならないエントリを無くすことができる。ctr カウンタは予測結果を出力したテーブルのみを更新する．予測結果が間違っていた場合，まず予測結果を出力したエントリのカウンタを更新し，もし予測値を出力したテーブル Ti が最も長い履歴長を使用していなければ，新しいエントリを Ti よりも履歴長が長いテーブル Tk($i \leq k < 5$) に置く．そして以下のようにする．

- エントリ Tk のカウンタ u の値が 0 のとき Tk にエントリを置く．
- 0 のエントリが無いときは Tj($i < j < 5$) のカウンタ u を 0 にし，そこに新

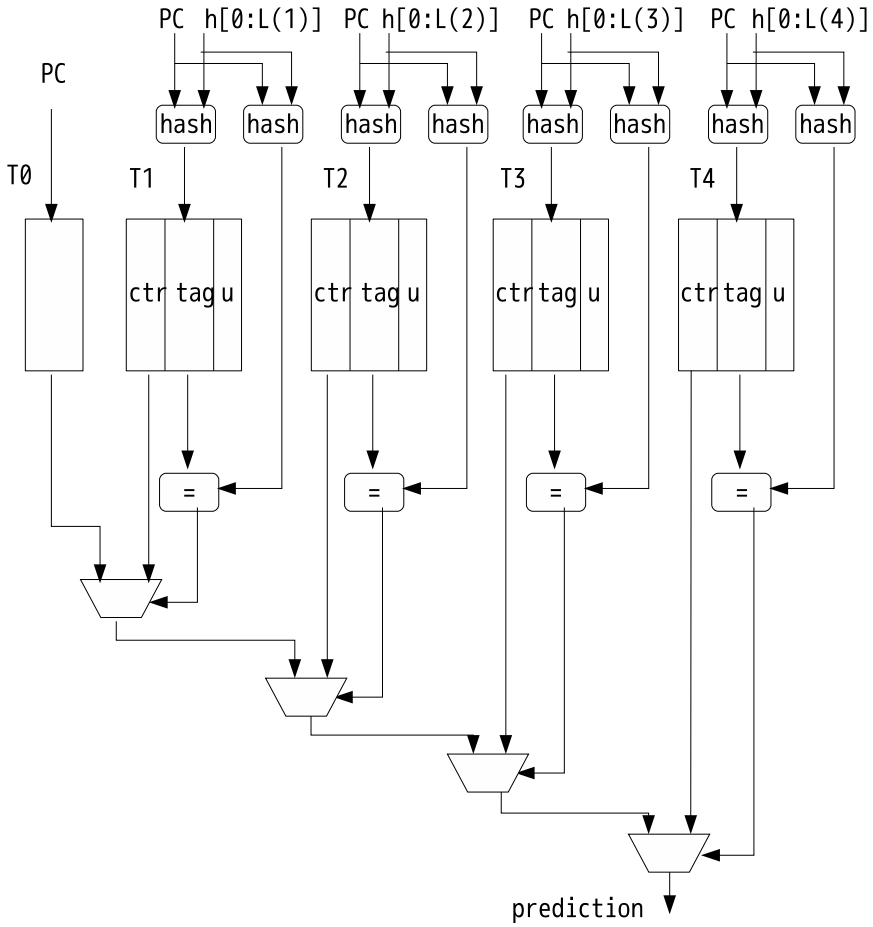

図 4 TAGE 予測器の全体構成

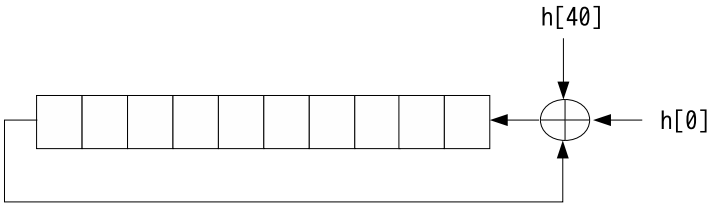

図 5 GHR40 ビット時の CSR

しいエントリをおく．

- 置き換えたエントリは分岐の結果が Taken(Not Takne) のときは弱く Taken（Not Taken）と予測するように ctr カウンタを設定し，カウンタ u は 0 に初期化する．

カウンタ u はエントリの置き換えを決める重要な要素で，値が 0 より大きいということはそのエントリが最近に有効な予測結果を出力したことを示す．

2.6 MIN-TAGE 予測器

2.6.1 全体構成

MIN-TAGE 予測器の全体構成を図 6 に示す．MIN-TAGE 予測器は 4 つの予測テーブルと 3 つの Enhanced Folded Index レジスタ，タグ比較回路，セレクタで構成される．予測テーブルである T0 から T3 のうち，T0 を BIM とよび，分岐命令のアドレスをインデックスとして使用する bimodal 予測器である．T1 から T3 はタグ（TAG）と 2 つのカウンタ U，P で構成され，Enhanced Folded Index と分岐命令のアドレスのハッシュ値をインデックスとしてアクセスする．この 3 つのテーブルを以降では PHT とする．Enhanced Folded Index は GHR と実行パス履歴（PHR : Path History Register）で構成されている．詳細ついては後に説明する．TAG は分岐命令のアドレスと Enhanced Folded Index の XOR で構成される．PHT のカウンタ P と U は飽和カウンタで，P は分岐の Taken，Not Taken に

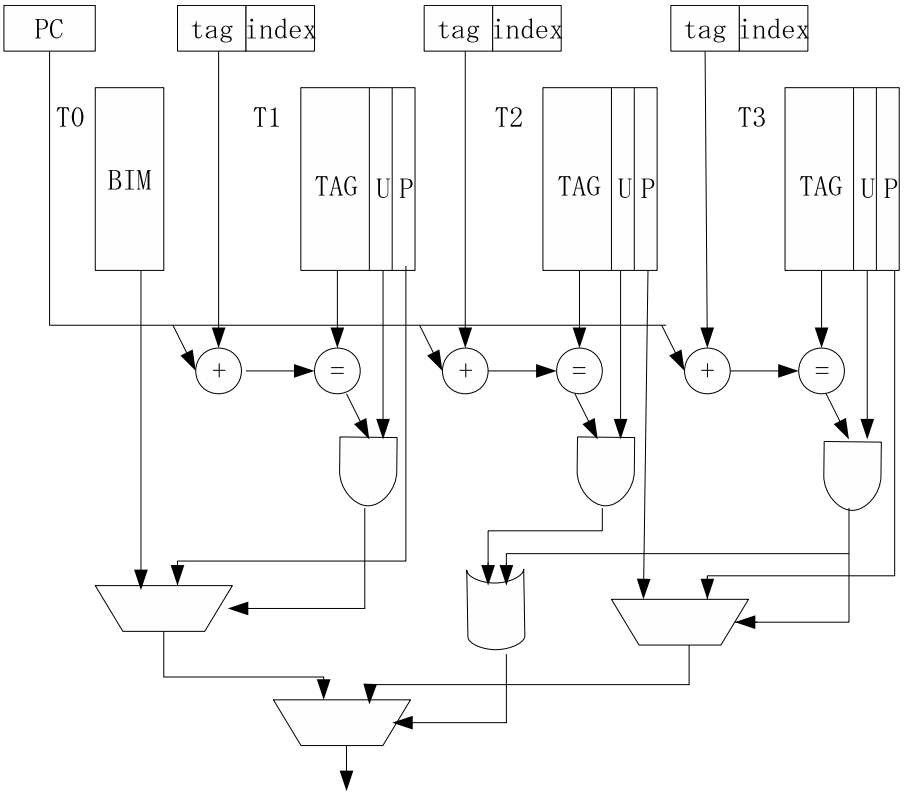

図 6 MIN-TAGE 予測器の全体構成

よって増減し，分岐の予測値を生成する際に使用される．U は予測結果の正否によって増減するカウンタで，エントリの有効性を示す．

2.6.2 Enhanced Folded Index

Enhanced Folded Index の構成を図 7 に示す．この図は 15 ビットの GHR と 11 ビットの PHR で構成した例である．GHR は分岐結果の履歴を格納したシフトレジスタで，H0 は最近の分岐命令の結果を示す．PHR は実行した分岐命令のアドレスのビットの 1 部を格納したシフトレジスタで，P0 は最近の分岐アドレスのビットの 1 部を示す．Enhanced Folded Index は単純なシフトレジスタである GHR や PHR と異なり，図に示すように循環的に XOR をとることで，少ないビット数で長い分岐履歴と実行パス履歴の情報を含むことができる．1 サイクルの動

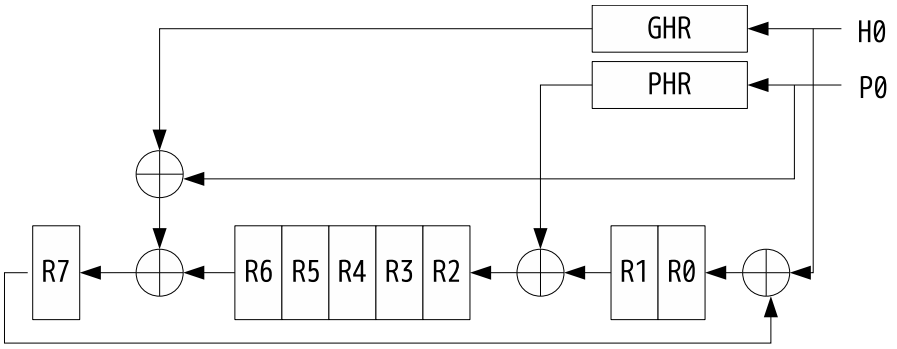

図 7 Enhanced Folded Index

きとしてはレジスタ R0，R2，R3，R4，R5 の値はそれぞれそのままレジスタ R1，R3，R4，R5，R6 にシフトする．R0 は H0 と R7 を XOR した値，R2 は 10 個前の分岐命令のパス情報である P10 と R1 を XOR した値，R7 は 14 個前の分岐命令の結果である H14 と P0 と R6 を XOR した値が格納される．

2.6.3 予測値の生成

予測は予測回路の遅延を軽減するため 3 サイクルかけて行われる．そのため，予測対象の分岐命令が得られるよりも前に予測を開始し，3 サイクル目で分岐命令が得られると，それを使用して予測値を生成する．MIN-TAGE 予測器のパイプライン構成を図 8 に示し，サイクル毎の動きを説明する．

- 1 サイクル目
  Enhanced Folded Index の一部，図 7 の例でいうと R0，R2，R3，R4，R5 の 5 ビットを使って PHT の行選択をする．

- 2 サイクル目
  再び Enhanced Folded Index の一部である R0，R2，R7 の 3 ビットを使って PHT の列選択を行う．また，直前の分岐命令のアドレスを使用して BIM の行選択をする．

- 3 サイクル目
  予測対象の分岐アドレスを使用して BIM の行選択をする．また，PHT から読み出したエントリのタグと分岐アドレスを使用して作成したタグを比較し，タグ同士が一致してかつカウンタ U が 0 でないエントリが予測値を出力する．出力された予測値の中からより長い履歴長をもつテーブルエントリの予測値を優先して最終的な予測結果を出力する．図 6 の場合で言うと T3，T2，T1，T0 の順に優先的に予測値がセレクタによって選択され，予測値を出力する．最終的な予測値は Enhanced Folded Index にフォワーディングされ，次回以降の予測に使用される．

PHT のエントリを読み出しに使用するインデックスは 2 サイクル目の R0 から R7 の 8 ビットを使用するが，2 サイクル目の時点で R1，R3，R4，R5，R6 に格納される値は，1 サイクル目ですでに R0，R2，R3，R4，R5 として決定している．そこで 1 サイクル目に R0，R2，R3，R4，R5 を使用して行選択をすることで，予測を 1 サイクル前倒しが可能となる．しかし，R0，R2，R7 は，2 サイクル目で XOR をとることで値が変化するため，前倒しができない．そのため，それらのレジスタは 2 サイクル目の列選択に用いる必要がある．また，2 サイクル目ではまだ予測対象の分岐アドレスが判明していないため，BIM の行選択に直前の分岐命令のアドレスを使用する．3 サイクル目で最終的な予測値を出力する際に，より長い履歴長を持つテーブルエントリの予測値を選択するのは，長い履歴長が一致する程，予測対象の分岐命令が過去の分岐パターンと強い相関関係があるとみなすことができ，そのテーブルエントリの予測値を使用することで，高い予測精度を期待できるからである．

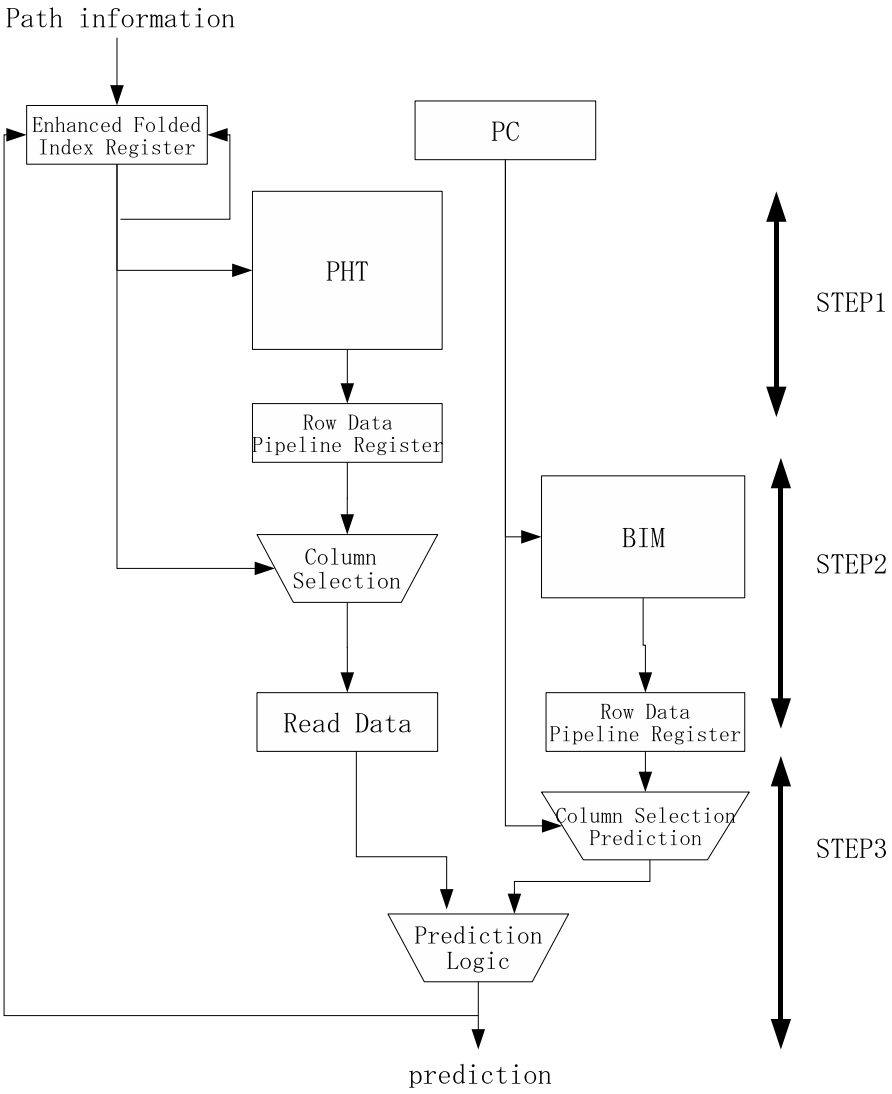

図 8 MIN-TAGE 予測器のパイプライン構成

3. プロセッサとトレースシミュレータの一般的な相違

分岐予測器の性能評価はよくトレースレベルのシミュレータによって行われているが，実際のパイプラインプロセッサとは挙動が異なる．そのため，プロセッサ上に予測器を実装すると，予測器の挙動もトレースレベルのシミュレータとは異なってくる．本章では具体的に予測器をパイプラインプロセッサ上で実装した場合とトレースシミュレータ上に実装した場合の挙動の違いについて述べる．

3.1 相違点

プロセッサとトレースシミュレータの大きな違いとして予測テーブルやインデックスに使用するレジスタの更新のタイミングがあげられる．プロセッサは複数のパイプラインステージによって構成されており，パイプラインの前方のステージで予測値を生成して，予測方向へとジャンプする．そして，その分岐命令の結果が予測テーブルに反映されるのは分岐命令の実行が完了するパイプラインの後方のステージである．つまり，予測からテーブル更新までパイプラインのステージ数分だけサイクル数が必要となるため，実行される分岐命令の間隔が小さいと直前のいくつかの分岐の結果が予測テーブルやレジスタに反映されないまま予測することになる．例えば図 9 では I1 で予測する際に I0 の分岐結果を使用することができない．また，命令列を実行していくなかでキャッシュのヒット・ミス等により処理に要するサイクルが異なってくるため，更新されない直前の分岐の数が時間とともに変化する．つまり，あるアドレスの分岐命令を予測するときに 2 個前までの分岐結果が反映されているときもあれば，3 個前までしか結果が反映されていないときもある．一方トレースシミュレータの場合，図 10 に示すように予測値の生成と更新が同じサイクルで行われる．つまり，常に直前の分岐結果がテーブルやレジスタに反映された状態で次の分岐命令の予測をしている．

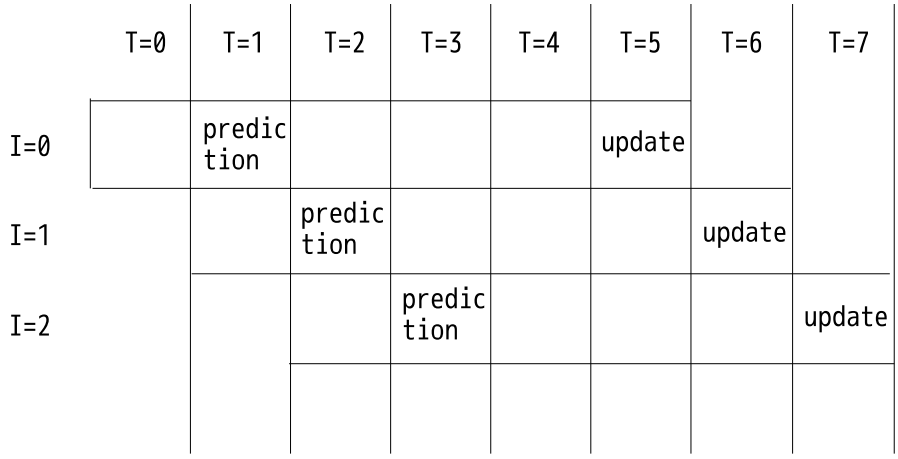

図 9 プロセッサ上での予測と更新のタイミング

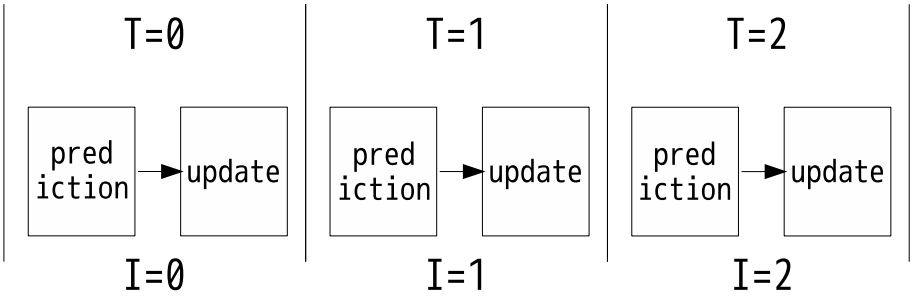

図 10 トレースシミュレータ上での予測と更新のタイミング

4. 実装

本章ではまず，MIN-TAGE 予測器を実装するプロセッサのパイプライン構成について簡単に説明する．次に MIN-TAGE 予測器のプロセッサへの実装方法について，予測値を生成する予測部，更新のためのデータを次段のパイプラインステージに伝送する伝送部，予測テーブルを更新する更新部に分けて説明する．

4.1 プロセッサのパイプライン構成

本研究では ARM のスーパースカラプロセッサを使用する．プロセッサのパイプライン構成を図 11 に示す．まず IA ステージではフェッチする命令のアドレスを生成する．IF ステージでは命令キャッシュ（I1-cache）から連続 2 命令を読み出す．同時に，gshare 分岐予測器（BP）を用いて分岐予測情報を後段に渡す．命令デコーダ（ARM-D）では，ARM 命令を RISC 型内部命令に変換する．また，このステージで分岐命令が判明し，予測が Taken の場合，分岐先にジャンプする．命令デコーダ（HOST-D）では，実行条件付き命令を条件に関わらず依存関係が変化しない命令列に分解する．MAP ステージでは，論理レジスタから物理レジスタへリネーミングを行う．SEL/RD ステージでは演算器からのバイパスが利用可能かどうか調べるとともに，依存関係の待ち合わせと命令発行を行う．そして SFM でシフト演算と積和演算用の補助演算，ALU は一般的な加減算や論理演算，EAG はアドレス計算と積和補助演算の選択，OP1 はデータキャッシュおよびストアバッファを対応付ける．WR では物理レジスタに演算結果の書き込みを行う．RETIRE ステージでは，先行命令が全て完了した命令を物理レジスタから論理レジスタに変える．

4.2 MIN-TAGE 予測器の実装

本節では，IA，IF，ARM-D ステージを分岐予測を行うところとして予測部，HOST-D，MAP，WR ステージを予測テーブルの更新に必要な情報を伝達するデータ伝送部，そして RETIRE ステージを予測テーブルを更新する更新部とする．

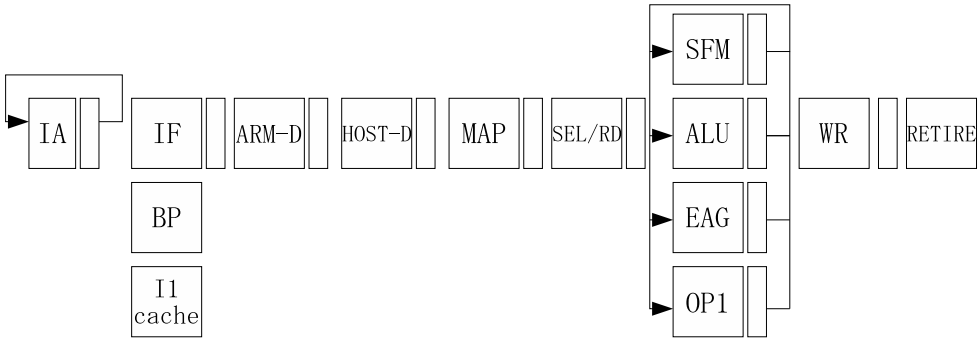

図 11 プロセッサのパイプライン構成

4.2.1 予測部

MIN-TAGE 予測器は予測値を生成するため，まずは PHT から行選択によって予測値の候補となる Row Data を読み出し，次のサイクルで列選択によって予測値を生成するエントリを 1 つに決定する．そして，次のサイクルで予測結果を出力する．予測値を生成するのに 3 サイクルを要するので，IA ステージで PHT の行選択，IF ステージで列選択，そして ARM-D ステージで予測値を生成するように実装する（図 12)．そのため，IA・IF ステージ間のパイプラインレジスタに PHT から読み出した Row Data と Row Data の読み出しに使用したインデックス（Row Index）を格納するレジスタ，IF・ARM-D ステージ間には前段からの Row Index，列選択で選ばれた Read Data，列選択に使用したインデックス（Column Index），また IF ステージから BIM による予測が開始されるので BIM の行選択によって読み出したデータ（BIM Row Data），その行選択に使用したインデックス（BIM Row Index）を格納するレジスタを追加する．

- 予測 1 サイクル目
  IA ステージで Enhanced Folded Index を使用して PHT の行選択をし，読み出したデータを IA・IF ステージ間のパイプラインレジスタに格納する．

- 予測 2 サイクル目
  IF ステージで再び Enhanced Folded Index を使用して PHT の列選択をし，

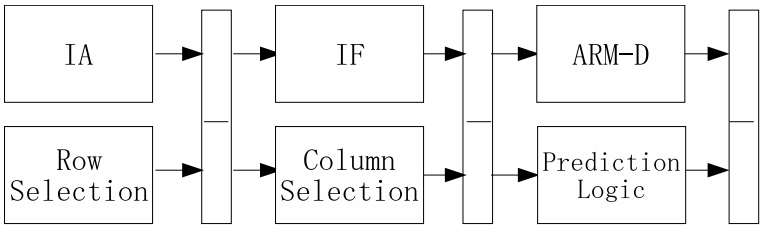

図 12 予測部のパイプライン構成

予測値を生成するエントリを各テーブルごとに 1 つ決定すると同時に分岐アドレスを使用して BIM の行選択をしパイプラインレジスタに格納する．

- 予測 3 サイクル目
  ARM-D ステージで分岐アドレスを使用して BIM の列選択をし，BIM の予測値を決定するとともに分岐アドレスと Enhanced Foleded Index から生成されたタグと選択したエントリのタグとを比較し，履歴長ができるだけ長くタグが一致したエントリの予測カウンタから予測値を生成する．もしタグが一致するエントリが存在しない場合は BIM の予測値を最終的な予測値とする．

タグの生成時に使用する Enhanced Folded Index レジスタは PHT のエントリ選択時に使用するものとは別に用意する．具体的には PHT 選択用のインデックスでは使用していた PHR を使用せず GHR のみで構成し，ビット長がタグの長さと同じもの 1 つと，タグの長さより 1 ビット短いもの 1 つの計 2 つ使用する．そしてこれと分岐アドレスを XOR した結果をタグとして使用する．このように 2 つの Enhanced Folded Index を用いるのは 1 つだけだと分岐履歴の周期的なパターンに影響を受けやすいからである．例えば図 13 において，H0 として入ってくるパターンと R7 からでてくるパターンがそろっていると XOR によって常に 0 になってしまう．そこで，もうひとつ 1 ビット短いものを用意してそのようになるのを防ぐ．予測値は GHR にフォワーディングして，GHR の 0 ビット目に格納し，分岐アドレスの下位 2 ビット目はパス情報として PHR の 0 ビット目に格納する．そして Enhanced Folded Index も予測値を使って更新する．

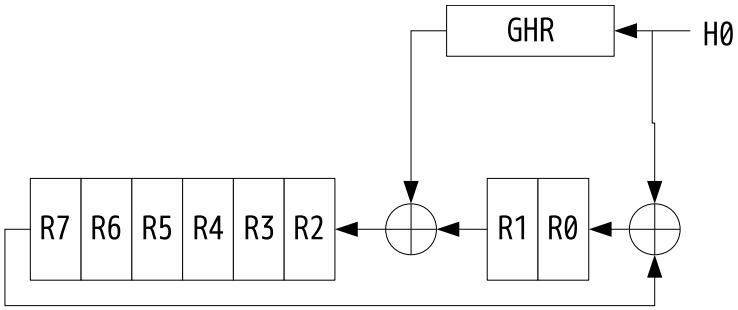

図 13 タグ用の Enhanced Folded Index レジスタ

4.2.2 データ伝送部

データ伝送部ではRETIREステージで予測テーブルを更新するため，各パイプラインレジスタにそれに必要な情報を格納するレジスタを追加し，後段のステージに渡していく．以下に必要な情報とその目的を示す．

- BIM Colum Index：予測値を生成したBIMのエントリを特定．
- BIM Row Index：予測値を生成したBIMのエントリを特定．
- Colum Index：予測値を生成したPHTのエントリを特定．
- Row Index：予測値を生成したPHTのエントリを特定．
- HitBank：予測値を生成したテーブルを特定．
- タグ：PHTのエントリに格納
- 予測値：新しいエントリを書き込むかどうかやUの更新

BIM Colum Index，BIM Row Index，タグ，HitBank，予測値を格納するレジスタはそれぞれ1つずつでよいが，Colum IndexとRow Indexはテーブル毎にどのエントリ，が予測値を生成する候補となったのか特定するため，BIMを除く各テーブル毎に必要となる．そのため，これらを格納するレジスタはそれぞれ3つずつ必要となる．

4.2.3 更新部

予測テーブルの更新は分岐命令の結果が判明し，実行が完了する RETIRE ステージで行う．BIM および PHT の更新の流れは，予測がヒットしていた場合とミスしていた場合とで異なってくる．まず，予測がミスした場合，予測値を生成したテーブルが T3 以外のとき PHT に新しいエントリを追加する．

- パイプラインの前段から来た Row Index と Column Index を使ってカウンタ U の値が 0 のエントリを探す．
- もしあればそのエントリに予測に使用したタグを代入し，結果が Taken だったときは 0，Not Taken だったときは-1 を CTR に代入する．U は 0 を代入する．
- U の値が 0 のエントリが存在しなかったとき，予測値を出したテーブルよりも長い履歴長を使用する PHT の中からランダムにテーブルを選択し，選択したテーブルの Row Index と Column Index が示すエントリを予測に使用したタグで置き換え，結果が Taken だったときは 0，Not Taken だったときは-1 を CTR に代入し，U は 0 を代入する．
- 予測値を生成したテーブルのエントリの CTR を Taken のときはインクリメントし，Not Taken のときデクリメントする．もし，予測値を生成したテーブルが PHT の場合 U もデクリメントして更新する．
- 予測がヒットした場合，エントリの置き換えはせず，予測値を生成したテーブルのカウンタ値 U と CTR を更新する．

次に，GHR，PHR および Enhanced Folded Index の更新について述べる．これらのレジスタは予測値が分岐結果と異なった場合にだけ更新する．まず，GHR は ARM-D ステージで予測値を使用し，投機的に更新していた．なので，予測値と分岐結果が異なっているとき間違った値を含んでいることになる．そこで RETIRE ステージで補正する必要がある．PHR は分岐予測した方向の命令列を投機実行しているので，実際には実行されないパスの情報が含まれており，GHR 同様補

正の必要がある．そして，Enhanced Folded Index は投機的に更新していた GHR と PHR で構成されているので補正する必要がある．上記レジスタを補正する方法はこれらのレジスタをもう一組用意する．一方は予測用に ARM-D ステージで投機的に更新し，もう一方は RETIRE ステージで分岐命令の結果が判明したのちに更新する．そして，予測値が間違いとわかると RETIRE ステージで更新していたレジスタの値を投機的に更新していたレジスタにコピーする．これで次の分岐命令から正しい情報を使用して予測することができる．

4.3 相違

3 章では予測器をプロセッサ上に実装した場合とトレースシミュレータ上に実装した場合の一般的な違いについて述べたが，本節では MIN-TAGE 予測器を 4.1 節で述べたプロセッサ上に実装したときに生じるトレースシミュレータとの違いについて述べる．

まず，MIN-TAGE 予測器をトレースシミュレータに実装した場合図 14 のようなタイミングチャートになる．例えば I=0 が予測するとき，T=0 で PHT の行選択を行い，T=1 で列選択を行う．そして T=2 で予測結果を出力する．このとき T=1 で列選択をするが，これは列選択用のインデックスとして I=-2 が T=0 で予測した結果を Enhanced Folded Index レジスタにフォワーディングし，反映されたものを使用する．つまりトレースレベルのシミュレータの場合，常に 2 個前の分岐結果がインデックスに反映された状態にある．一方今回使用したプロセッサの場合，予測部分のパイプラインレジスタが IA・IF 間は 2 個分のデータ，IF・ARM-D 間のは 4 個分のデータを格納するキュー構造のレジスタとなっている．また，IA・IF 間はプログラムカウンタを格納するレジスタで，使用するプロセッサは最大 2 命令を同時にフェッチするため 1 つのレジスタが 2 命令分に相当するといえる．このような構造において，例えば処理がスムーズに進みキューにデータがたまっていない場合，一定個数前の分岐結果がインデックスに反映されるが，ARM-D 以降のステージで何サイクルも要するような処理をしてキューのデータがはけていかないと分岐命令がパイプラインレジスタにある状態で次の分岐命令もレジスタに格納される可能性がある．つまり，パイプラインステージ後段の処

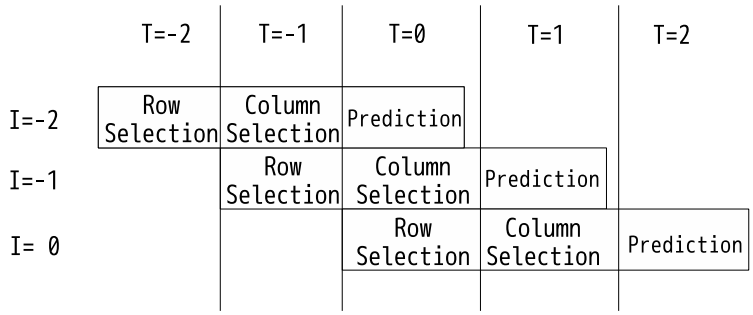

図 14　トレース上の MIN-TAGE 予測器のタイミングチャート

理内容しだいでインデックスに反映される分岐の数が変化し，インデックスも変わってくる．その結果，予測テーブルの使い方が異なってくるので，予測結果に影響してくると考えられる．

5. 評価

本章では4章で組み込んだMIN-TAGE予測器の性能についてPHTやEnhanced Folded Indexを構成するさまざまなパラメータの影響を調べる．また，MIN-TAGE予測器をパイプラインプロセッサ上で実装した場合とトレースレベルのシミュレータ上に実装した場合とでミス率の比較をし，その結果について考察する．

5.1 各種パラメータの影響

MIN-TAGE予測器はPHTに含まれるタグ長やカウンタのビット長，Enhanced Folded Indexレジスタを構成するGHR，PHRの長さといったパラメータが存在する．本節ではこれらのパラメータが予測精度にどのような影響を及ぼすのかを調べる．ベンチマークとしてStanford Benchmarkを使用する．

まずはGHR，PHRおよびタグの長さをいろいろな値に設定したときの影響について調べる．ここではGHR，PHR，タグ長をそれぞれ以下の表1，2，3に設定し，その組み合わせで予測器の平均ミス率を計測する．その結果を図15に示す．縦軸はStanford Benchmark 10個の平均ミス率，横軸は表1，2，3のパラータの組み合わせを示す．左から4本ずつがそれぞれPHRのパターン1から4に対応し，それを1セットとすると左から7セットずつがそれぞれGHRのパターン1から7に対応する．さらにそれを1セットとすると左から順にそれぞれタグのパターン1から4に対応する．

まず，PHRの影響から見ていくと，全体的にPHRを1ビット使用したときに最もミス率が低い結果となった．2ビットや3ビット使用すると逆にミス率が高くなり，PHRを使用しないときとほとんど変わらなくなった．次にGHRの影響をみるとGHRがパターン1や2のように短いと全てのタグ長において高いミス率となった．タグ長が長くなるとGHRのパターン3から7は同じようなミス率を示すが，パターン3で一旦さがり，パターン4，5，6とわずかにミス率が上昇するがパターン7で再び下がる傾向がタグ長パターン2から4で見られた．しかし，タグ長がパターン1のように短いときにGHRのビット数をふやすと逆にミス率が上昇した．

	table1	table2	table3
パターン 1 (bit)	3	5	7
パターン 2 (bit)	5	7	11
パターン 3 (bit)	7	11	19
パターン 4 (bit)	11	19	23
パターン 5 (bit)	7	11	23
パターン 6 (bit)	7	23	35
パターン 7 (bit)	7	23	47

表 1 GHR のパラメータ

	table1 (bit)	table2	table3
パターン 1 (bit)	0	0	0
パターン 2 (bit)	1	1	1
パターン 3 (bit)	2	2	2
パターン 4 (bit)	3	3	3

表 2 PHR のパラメータ

	table1	table2	table3
パターン 1 (bit)	3	4	5
パターン 2 (bit)	5	6	7
パターン 3 (bit)	7	8	9
パターン 4 (bit)	13	14	15

表 3 タグ長のパラメータ

	table1	table2	table3
GHR(bit)	8	12	19
PHR(bit)	1	1	1
tag(bit)	4	5	6
entry 数	128	256	256

表 4 1KB 構成のパラメータ

	table1	table2	table3
GHR(bit)	9	13	19
PHR(bit)	1	1	1
tag(bit)	4	5	6
entry 数	512	512	512

表 5 2KB 構成のパラメータ

	table1	table2	table3
GHR(bit)	9	15	21
PHR(bit)	1	1	1
tag(bit)	4	5	6
entry 数	1024	1024	1024

表 6 4KB 構成のパラメータ

	table1	table2	table3
GHR(bit)	7	23	47
PHR(bit)	1	1	1
tag(bit)	14	15	16
entry 数	1024	1024	1024

表 7 8KB 構成のパラメータ

	table1	table2	table3
GHR(bit)	7	23	47
PHR(bit)	1	1	1
tag(bit)	15	16	17
entry 数	2048	2048	2048

表 8 16KB 構成のパラメータ

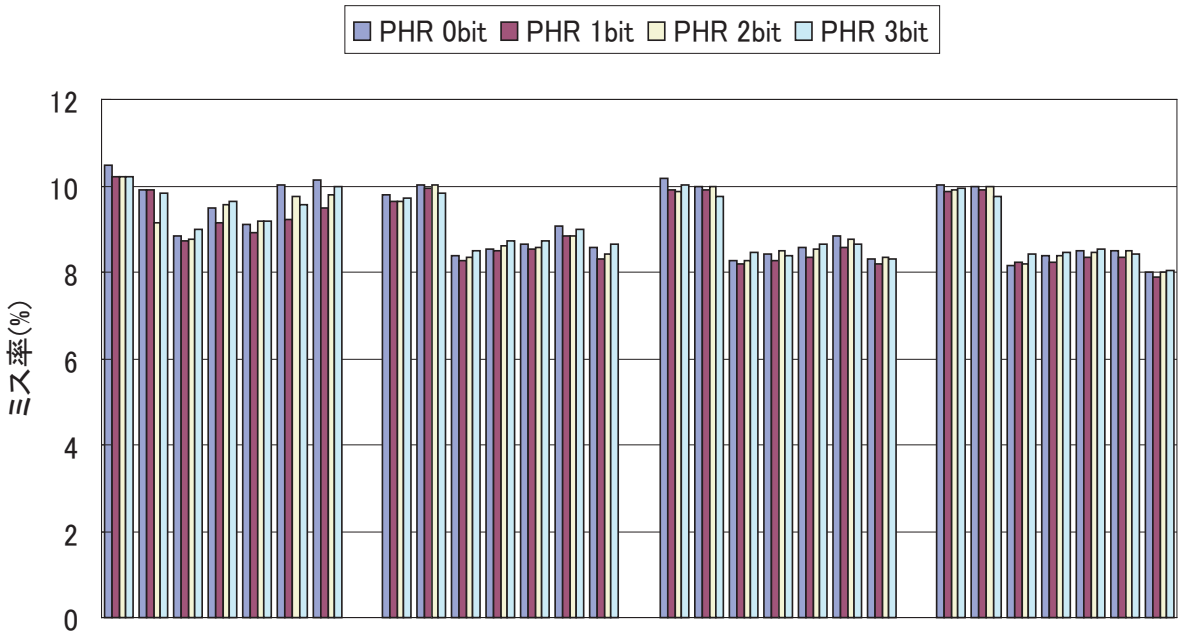

図 15 各種パラメータ設定時の予測ミス率

次にPHTを構成するカウンタのビット数による影響について調べる．まずはカウンタCTRのビット数の影響を調べるため容量1KB，2KB，4KB，8KB，16KB構成についてカウンタUを2ビットに固定し，カウンタCTRを2，3，4ビットと設定したときの平均ミス率を調べた．1KB，2KB，4KB，8KB，16KBで使用したパラメータをそれぞれ表4，5，6，7，8に示す．これらは実験した中で最も低いミス率を示したものを使用した．結果を図16に示す．縦軸は平均ミス率，横軸は容量を示す．

結果はカウンタCTRのビット数が4ビット，3ビット，2ビットの順にミス率が低くなった．また，2ビットを使用するよりも3ビットや4ビットを使用したほうが1KB，2KB，4KB構成でミス率が約0.2%と相対的に大きく減少した．それに対し3ビットと4ビットを比較した場合，4ビットの方がわずかに低いミス率を示すがほぼ同じ結果となった．

つづいてカウンタUのビット数による影響を調べる．上記と同じく容量1KB

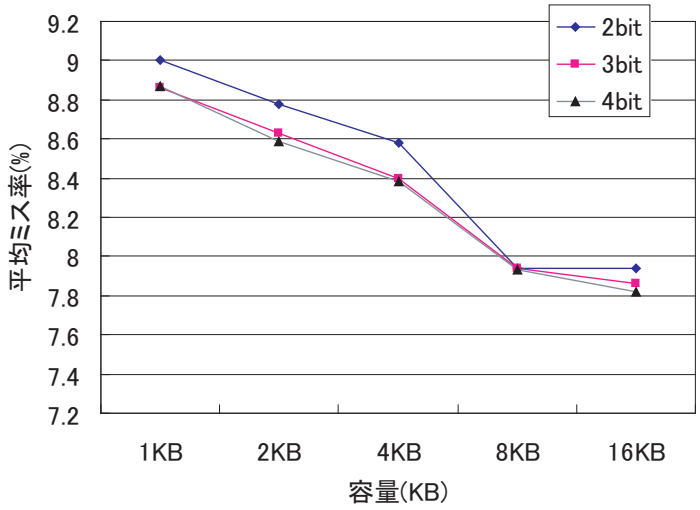

図 16　カウンタ CTR のビット数と予測ミス率

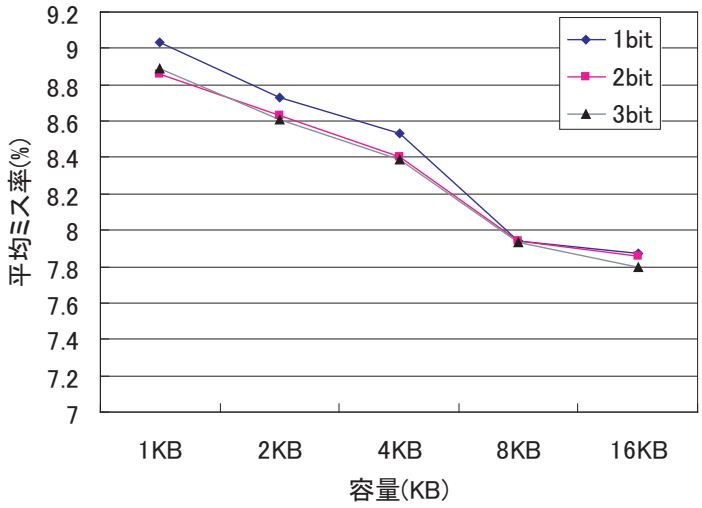

図 17　カウンタ U のビット数と予測ミス率

から 16KB の構成についてカウンタ CTR を 3 ビットに固定し，カウンタ U を 1，2，3 ビットと設定したときの平均ミス率を調べた．結果を図 17 に示す．結果は容量が 1KB，2KB，4KB の構成のとき，カウンタ U のビット数を 1 ビット使用したときと 2 ビット，3 ビット使用したときの平均ミス率は相対的に大きな差があり，2 ビット，3 ビット使用した方が低いミス率となった．8KB，16KB 構成の場合については 1，2，3 ビットほぼ同じミス率となった．また，2 ビットと 3 ビット使用したときについては全体的にほぼ同じミス率で最高でも 0.04%の差しかなかった．

5.2 プロセッサとトレースの結果

次に容量 1KB から 16KB についてプロセッサとトレースのミス率を図 18 に示す．縦軸は stanford benchmark 10 個の平均ミス率，横軸は容量を示す．結果をみると gsahre 予測器はトレースよりもプロセッサの方が低いミス率となった．一方で MIN-TAGE 予測器ではトレースの方が低いミス率となった．また，全体的にみると gshare 予測器は容量を増やしてもミス率の減少が見られなかったのに対し，MIN-TAGE 予測器は容量を増やすと徐々にミス率が減少していった．

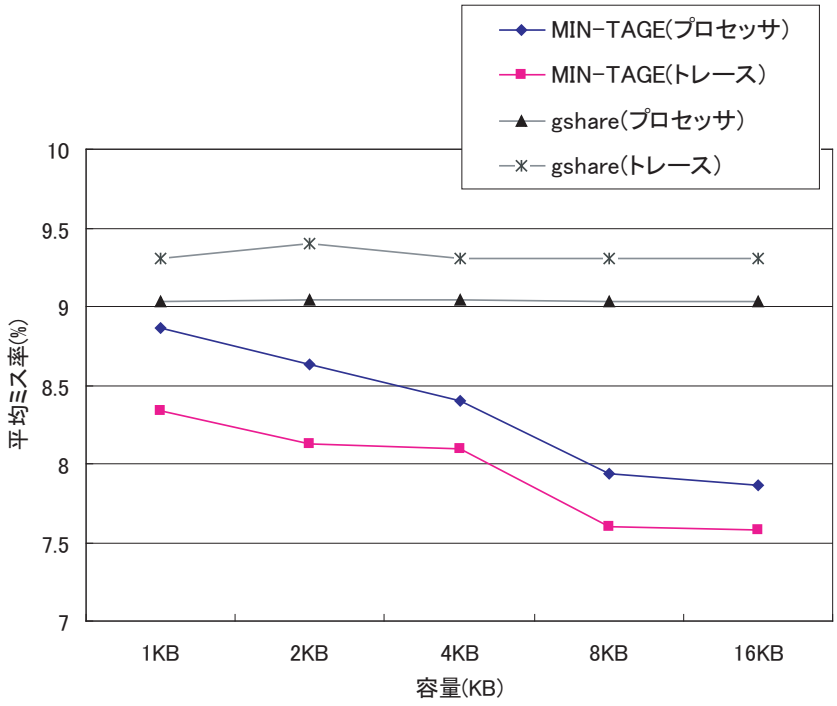

図 18　gshare 予測器と MIN-TAGE 予測器の予測ミス率

6. 考察

まずは Enhanced Folded Index に使用する GHR の影響を見ていくと，GHR がパターン 1 や 2 のように短いと PHT 選択やタグで使用する Enhanced Folded Index も短くなる．そのため他のアドレスと競合する割合が多くなり，ミス率が相対的に大きくなったと考えられる．しかし，tag パターン 5 において GHR パターン 3，4，5 にみられるように，単に大きくすればミス率が減少するわけではく，逆に大きくなっていることから適切な長さの組み合わせが存在すると考えられる．

次にタグ長の影響について見てみる．タグ長が短いと GHR が短いときと同様に他のアドレスと同じパターンのタグになりやすいので競合の割合が多くなり，相対的にミス率が大きくなる．しかし，タグ長パターン 2 よりもタグ長をのばしても大きなミス率の減少は見られず，パターン 4 以降ほぼ同じミス率となった．本来ならばタグ長を長くすれば一致したときに過去の分岐パターンと強い相関性が存在することから，一致したエントリの予測精度は高いはずである．それにより，PHT での予測ミス率が減少し，全体でみても性能が向上することが期待できる．そこでタグ長を長くしたときに PHT の予測ミス率をテーブル毎に見てみる．図 19 は tabel1，table2，table3 のタグ長がそれぞれ 11，12，13 のときと 27，28，29 のときを示す．

図 19 を見るとタグ長を長くしても各テーブルのミス率はほぼ変わっていない．つまりこの結果から一定のアドレスでかつ過去の履歴長と実行パス情報が同じにも関わらず，分岐方向が一定でない命令が多く含まれているといえる．なので，タグ長をのばしても大きくミス率が減少しなかったと考えられる．またこのことからアドレスと GHR を使用して予測する gshare もテーブルの容量を大きくしてもミス率がほとんど変化しなかったと考えられる．

次に PHR の影響についてみていくと，結果では 1 ビット使用したとき，2 ビットや 3 ビット使用するよりも低いミス率を示したが，トレースでタグ長パターン 2 と 4，GHR パターン 7 について PHR の影響を見てみると図 20 のようになった．縦軸は平均ミス率を示す．この図をみると PHR を 2 ビット使用すると最も低いミス率を示し，次に 1 ビット，0 ビットと続く．つまり，より長い PHR を使用す

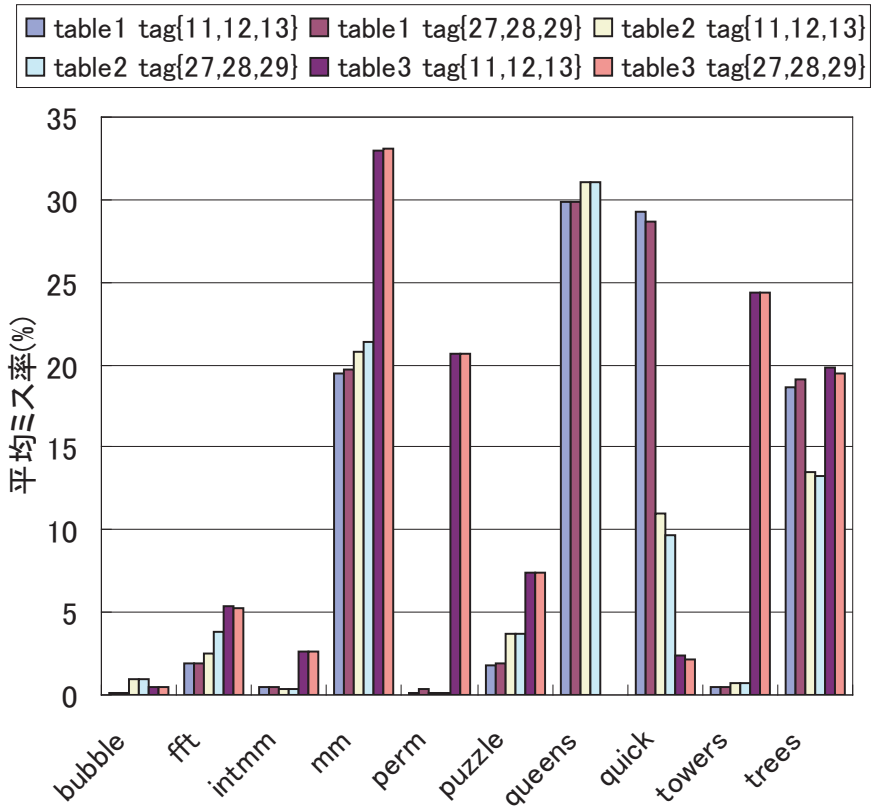

図 19 各テーブルの予測ミス率

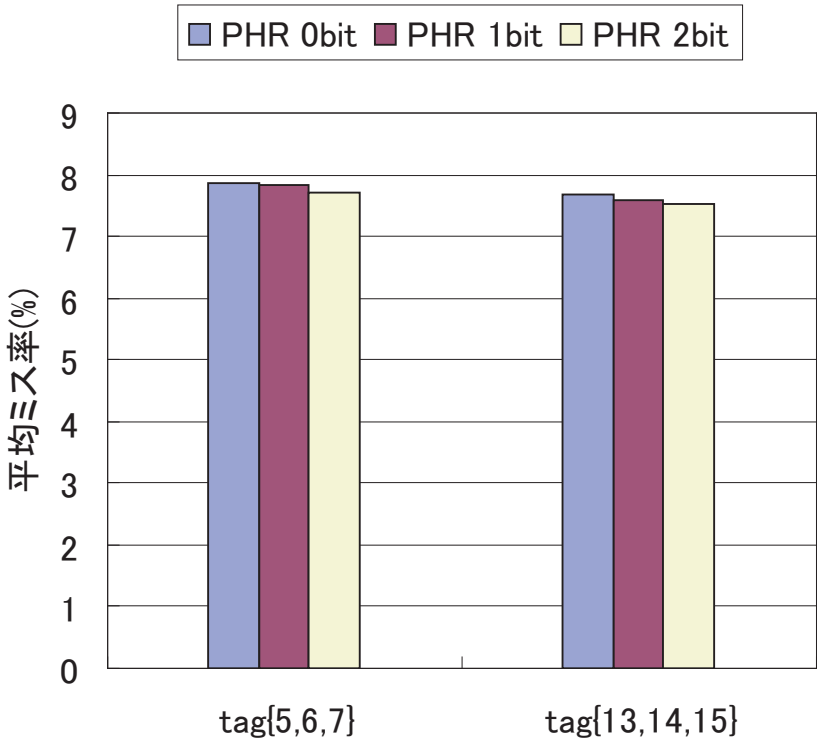

図 20　トレースシミュレータ上での PHR のビット数と予測ミス率

るとよい性能を示す結果となった．PHR を使用すると分岐履歴以外に実行経路も情報として含まれる．例えば，図 21 で E の分岐命令の分岐方向を予測するときに実行パス ACDE，BCDE どちらも Taken，Not Takek，Taken である．しかし，パス ACDE は E で Not Taken になるがパス BCDE は Taken になる．分岐履歴だけを見ると Taken なのか Not Taken なのかが判断できないが，実行パス履歴を利用することで分岐方向が特定できる．そのため，PHR を使用すると分岐方向の情報以外に実行経路の相関情報も利用することで，本来ならば図 20 のように PHR をまったく使用しないときよりも使用したほうがミス率が減少するはずである．

しかし，PHR が 2 ビット，3 ビット以上だと 0 ビットと同じ性能になった理由として PHT エントリを選択するインデックスにあると考えられる．まずトレースのインデックスは 4.6 節でも述べたように毎サイクル予測結果が Enhanced Folded

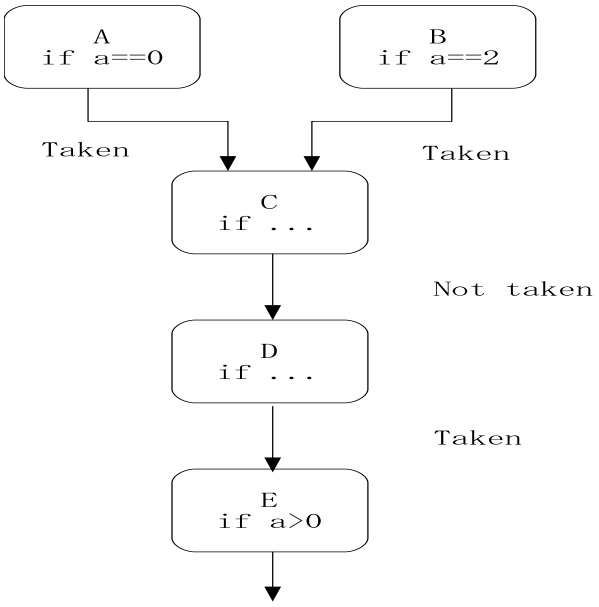

図 21 実行経路情報

Index にフォワーディングされシフトするので PHT の行選択と列選択で Enhanced Folded Index が異なっていた．ところがプロセッサに組み込んだ場合，毎サイクル Enhanced Folded Index をシフトすることはなく，分岐命令が IA ステージの時点でタイミングよく予測結果をフォワーディングしない限り，行選択と列選択で Enhanced Folded Index は同じになる．すると次のようなことが起こる．Enhanced Folded Index を 10 ビット，GHR を 13 ビット，PHR を 1 ビット使用した場合，図 22（a）に示すように行選択にレジスタ R0,R1,R5,R6,R7,R8，列選択にレジスタ R0,R3,R4,R5 を使用し，PHR を 2 ビット使用した場合，図 22（b）に示すように行選択にレジスタ {R0,R1,R3,R5,R7,R8}，列選択にレジスタ {R0,R3,R5,R7} を使用するが，Enhanced Folded Index がシフトしなかったとき，列選択のインデックスに行選択で使用したレジスタを使用することになる．つまり，PHR が 1 ビットの場合だとレジスタ R2 と R9 がインデックスとして使用されず，R0，R5 が行選択と列選択で同じ値になる．PHR が 2 ビットの場合だと，レジスタ R2，R4，R6，R9 がインデックスとして使用されず，列選択使用するレジスタ R0，R3，R5，R7 全てが行選択で使用したものと同じになる．これによる影響を調べるため，

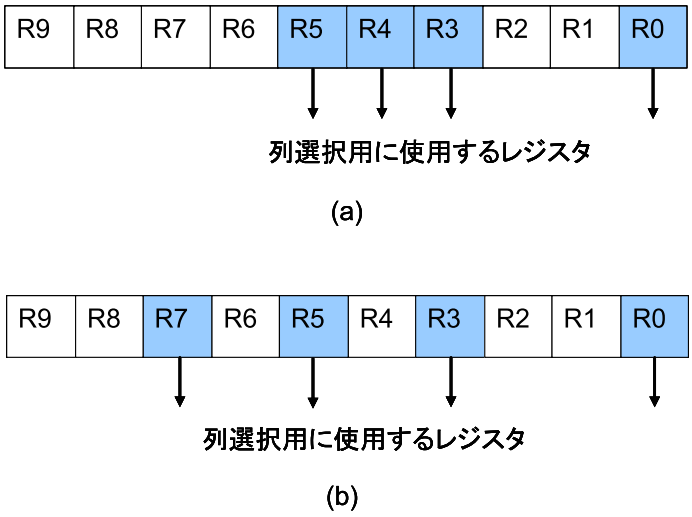

図 22　PHR1 ビット，2 ビット時の行・列選択インデックス

IF ステージで列選択をするときに IA ステージで行選択をしたときの Enhanced Folded Index と同じだったら予測結果のフォワーディングによるシフトがなかったとみなし，列選択のインデックスに行選択で使用されなかったレジスタを使用するようにした．例えば，図 22（a）でいうと列選択にレジスタ R2,R3,R4,R9 を，図 22（b）ではレジスタ R2,R4,R6,R9 を使用するようにした．図 20 で使用したパラメータと同じにしてミス率を測定した結果を図 23 に示す．

図 23 を見ると，PHR を 2 ビット使用したときが 1 番低いミス率を示した．また，タグ長パターン 2 において 0 ビットと 1 ビットの平均ミス率が同じになったが，ベンチマーク個々をみていくと Queens の 1 つだけミス率が高くなっただけでそれ以外は同じかより低いミス率となっている．このことから，PHT の行選択と列選択で同じ Enhanced Folded Index を使用するとミス率が大きくなってしまうといえる．特に，PHR を 2 ビット使用したときは Enhanced Folded Index のシフトがないと行選択のインデックスが決まった時点で列選択のインデックスが一意に決まってしまう．そのため，異なるアドレス同士でエントリの競合が発生し，ミス率の増加につながったと考えられる．PHR が 1 ビットの場合の方が 2 ビットの場合よりもミス率が低くなったのも，列選択のインデックスがレジスタ

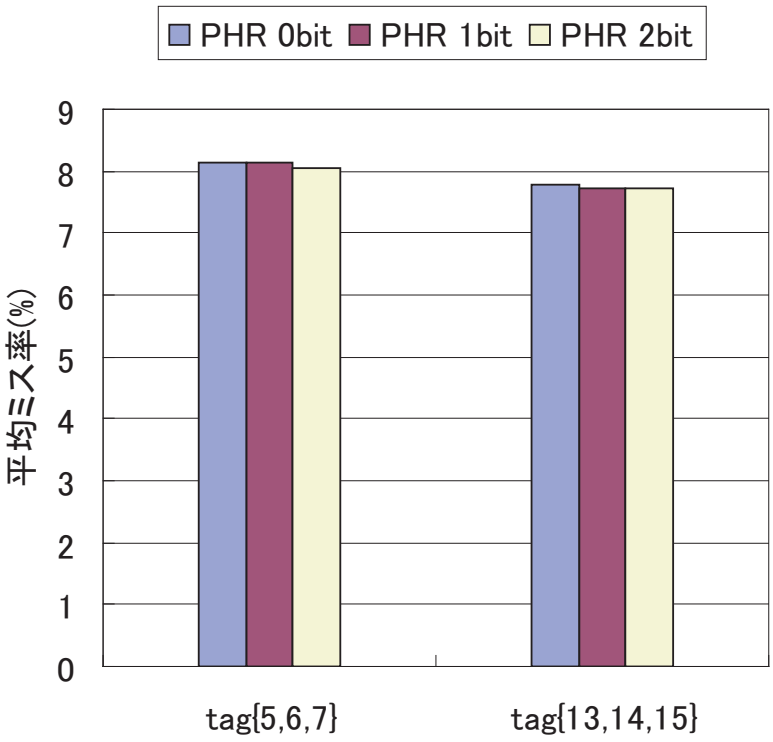

図 23 列選択方法変更時の予測ミス率

2 ビット分しか行選択のインデックスと同じにならなかったので，その分競合の割合が減ったからだと考えられる．

MIN-TAGE 予測器をプロセッサ上に実装したときにトレースと比較してミス率が高くなる要因の一つは上記で述べた PHT の行選択と列選択で同じ Enhanced Folded Index を使用していたことがあげられる．実際にその影響を調べるため，上記と同じように PHT の行選択と列選択で Enhanced Folded Index が同じ場合，列選択で使用するレジスタは行選択で使用されていないものを使用するようにした．その結果を図 24 に示す．縦軸は平均ミス率，横軸は容量を示す．図をみると全ての容量においてミス率が下がった．トレースとのミス率の差が元のものと比べると，大体半分くらいになり，平均で約 0.4%，最大で 0.52%あった差が平均で約 0.2%，最大 0.31%に減った．これは，列選択のインデックスに使用するレジスタが，行選択のインデックスで使用したレジスタの 1 部を使っていることから，選択できるエントリ数が減り，競合が発生していたためといえる．

以上の要因を取り除いてもまだ，トレースよりもミス率が高くなったのは 4.6 節で述べたプロセッサ上で実装した場合とトレースシミュレータ上に実装した場合の違いによるものと考えられる．

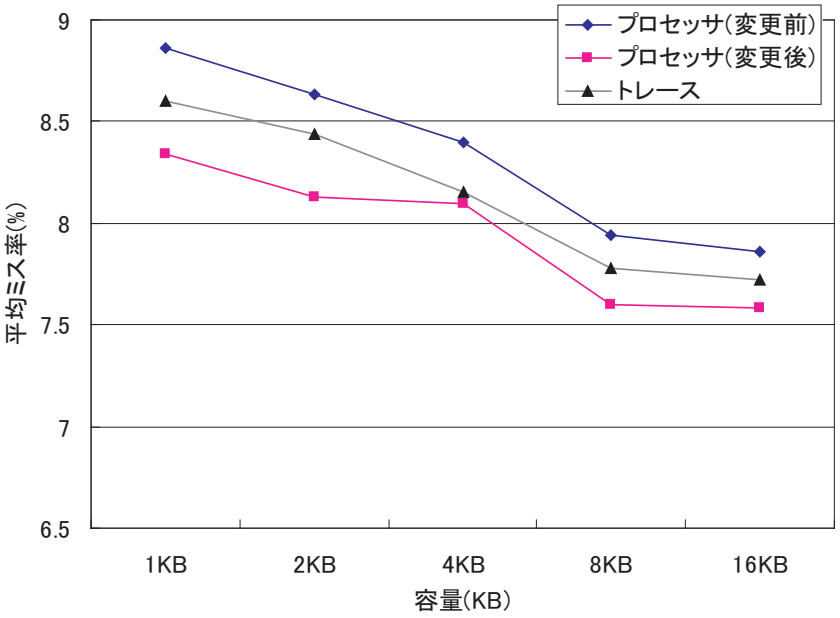

図 24　行・列選択方法を変更前と後の予測ミス率

7. まとめ

本論文ではまず，分岐予測器の挙動がパイプラインプロセッサ上とトレースレベルのシミュレータ上ではレジスタや予測テーブルの更新するタイミングの違いによって異なってくることを述べた．そして，実際に MIN-TAGE 予測器をパイプラインプロセッサ上に実装し，ベンチマークを使用して，トレースレベルのシミュレータとミス率の比較を行った．その結果，MIN-TAGE 予測器をパイプラインプロセッサ上に実装するとトレースレベルのシミュレータ上に実装するよりも平均で 0.4%ミス率が高くなり，性能が低下することが分かった．その大きな要因の一つがトレースレベルのシミュレータでは PHT の行選択と列選択で Enhnanced Folded Index が常に 1 ビットシフトしていたのに対し，パイプラインプロセッサでは，IA ステージでの行選択と IF ステージでの列選択の間で予測結果のフォワーディングによるシフトが常には発生しない．それにより，行選択と列選択で同じ Enhanced Folded Index を使用したため，列選択のインデックスで使用するレジスタが行選択のインデックスで使用するレジスタの 1 部を使用することになり，参照できるエントリの数が減少するからということが，Enhanced Folded Index のレジスタを全て使用するようにした実験から分かった．

また，本研究で用いたプロセッサの場合，Enhanced Folded Index に使用する PHR のビット数は 1 ビットが最も低いミス率となり，最適だということが分かった．2 ビット以上使用すると，使用しないときとほとんど変わらないくらい性能が低下した．その理由が上記の理由と同じで行選択と列選択で同じ Enhanced Folded Index を使用し，同じレジスタの値を行選択，列選択のインデックスで使用したためだということが分かった．実際に，きちんと全てのレジスタを使用すると 1 ビットよりも 2 ビット使用した方がミス率が減少した．また Enhanced Folded Index に使用する GHR の長さについては，使用する GHR が長いとミス率は減少するが，各テーブルに適切な長さが存在し，単純に長くしていってもミス率は減少しないことが分かった．また，PHT に使用するタグはベンチマークに含まれる分岐命令の特性から 13 ビット以上のばしてもミス率が減少しないことが分かった．

今後の課題としては，実際に MIN-TAGE 予測器をハードウェアで実装し，正確な遅延時間と回路規模を評価することである．MIN-TAGE 予測器は予測テー

ブルに使用するメモリ容量を同じにして gshare 予測器と比較した場合，低い予測ミス率を示すが，実際にはメモリ容量以外に，タグ同士の比較をする回路や，各テーブルからの予測値を選択するセレクタ，テーブル毎に持っている Enhanced Folded Index レジスタ，さらにパイプラインの RETIRE ステージで予測テーブルを更新するための情報を格納するパイプラインレジスタ等，メモリ以外の回路規模が gshare 予測器と比較して大きい．そのため，これらを考慮することがさらに詳細な規模に対する性能を評価する上で必要である．

8. 謝辞

本研究を行うにあたり，多大なる御指導を頂いた中島康彦教授，藤原秀雄教授，山下茂准教授，中田尚助教には深く感謝の意を表します．

平素から御助言，御指導を頂きました中西正樹助教に深く感謝いたします．最後に，本研究の内容に対して研究報告会などにおいて有意義な指摘や助言を頂いたコンピューティング・アーキテクチャ講座の皆様にお礼申し上げます．

参考文献

[1] McFarling, S.: Combining branch predictors, *Digital Western Research Laboratory Technical Note TN-36* (1993).

[2] 石井康雄, 平木敬: 小規模・省電力コアのための省資源分岐予測方式, 情処研報 2007-ARC-174, pp. 205–210 (2007).

[3] Seznec, A., Michaud, P.: A case for (partially)-tagged geometric history length predictors, *Journal of InstructionLevel Parallelism* (2006).

[4] Kise, K., Katagiri, T., Honda, H. and Yuba, T.: The bimode++ branch predictor, *Proceedings of the Innovative Architecture on Future Generation Hight-Performance Processors and Systems*, pp. 19–26 (2005).

[5] Seznec, A., Michaud, P.: De-aliased Hybrid Branch Predictors, *Technical report, INRIA RR-3618* (1999).

[6] Seznec, A.: Analysis of the o-gehl branch predictor, *Proceedings of the 32nd Annual International Symposium on Computer Architecture* (2005).

発表文献

[1] 牟田口公洋, 中田尚，中島康彦: パイプラインプロセッサ上での MIN-TAGE 予測器の性能評価，情報処理学会 研究報告 2008-ARC-177, P.37-42(2008).