

TAGE

---

分岐予測

## 文献

1. A PPM-like, tag-based branch predictor (2005)  
<https://www.jilp.org/vol7/v7paper10.pdf>  
(単体で読める)
2. A case for (partially) TAgged GEometric history length branch prediction (2006)  
<https://www.jilp.org/vol8/v8paper1.pdf>  
(1を先に読まないとわからない)
3. AMDがZen 2で採用した現在最強の分岐予測「TAGE」  
<https://pc.watch.impress.co.jp/docs/column/kaigai/1192296.html>
4. AMD Zen 2 CPUコアの物理的な姿が明らかに  
<https://pc.watch.impress.co.jp/docs/column/kaigai/1237830.html>

おさらい

### 用語

- イン・オーダー、アウト・オブ・オーダー実行
  - リザーベーションステーション
- レジスタ・リネーミング
  - リオーダーバッファ
- 投機的実行(Speculative execution)

## アウト・オブ・オーダー命令実行

- 命令の並びと実行開始・終了の順番が一致しない
- オペランドのデータ依存関係を解決して並列化
- パイプラインのフラッシュ(巻き戻し)は行わない

## リザーベーションステーション

- アウト・オブ・オーダー実行を実現するアルゴリズムの名前

## レジスタ・リネーミング

- 命令ごとのデータ依存関係を減らすため、レジスタの割り付けをやり直すことで並列度を上げる技術
- アウト・オブ・オーダー実行に必須ではない

## リオーダーバッファ

- レジスタ・リネーミングを実現するアルゴリズムの名前



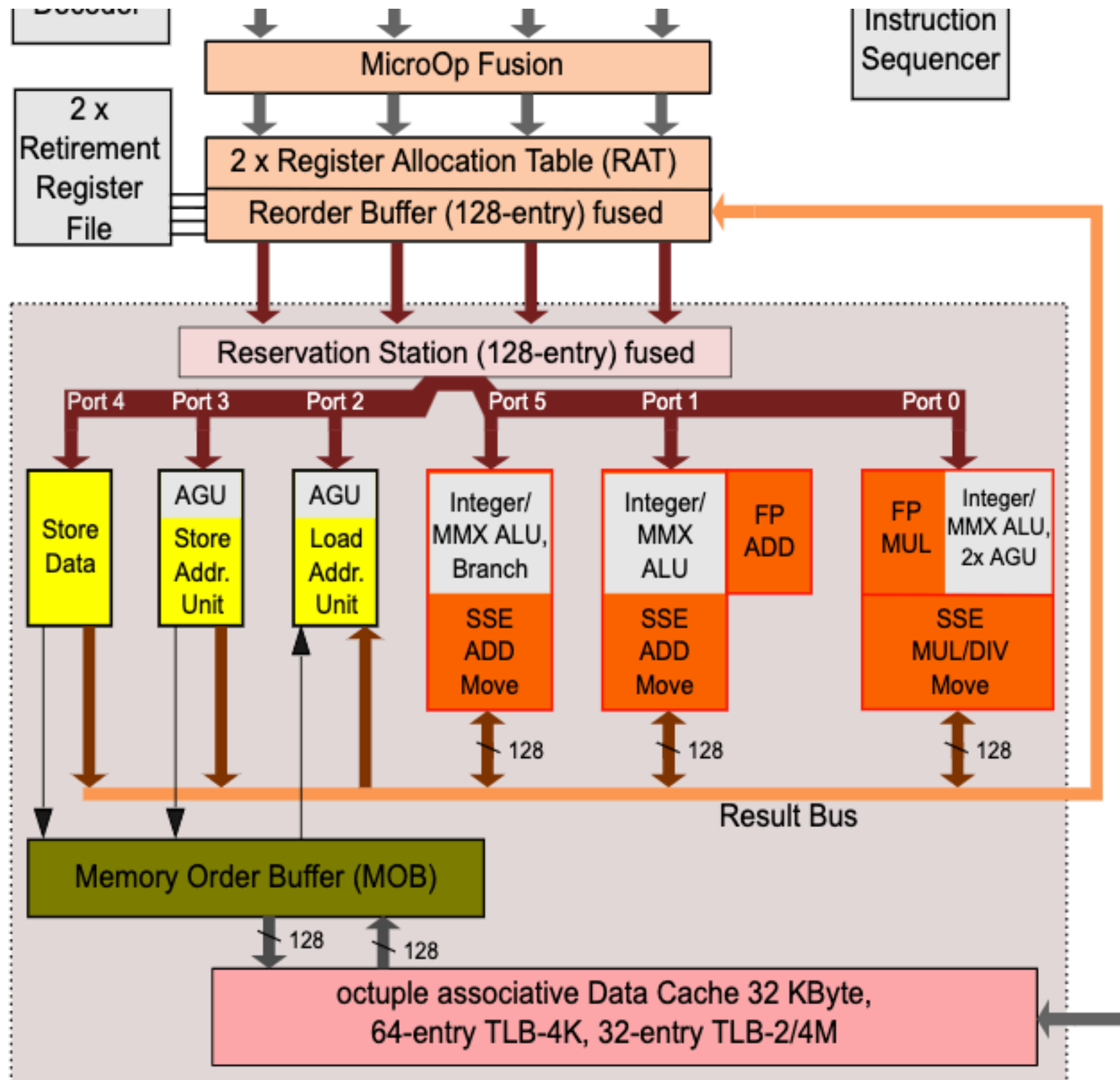
### 投機的実行

- 条件付き分岐(beqとか)を発見したとき、分岐する/しないが確定してないけど予測して試しに実行してみる
- 分岐する/しないが確定したとき、予測が間違ってたらパイプラインをフラッシュ(巻き戻し)して分岐先からやり直す
- パイプラインが深い現代のCPUでIPCを劇的に改善できる(予測精度が95%でも理想値から30%も性能低下する)

### なぜ必要?

- アウト・オブ・オーダー実行
  - ➡ ハードブロックの稼働率を上げIPCを向上する
- 投機的実行
  - ➡ 分岐でのストールを減らしIPCを向上する

どちらもシングルスレッド性能を改善する技術



[https://commons.wikimedia.org/wiki/File:Intel\\_Nehalem\\_arch.svg](https://commons.wikimedia.org/wiki/File:Intel_Nehalem_arch.svg)

# 単純な予測方法

### 必ず分岐する(しない)と予想する

- 分岐しないかもしれないが、するものとして続きを実行する
- コンパイラは分岐しないときのコストが高いことを知ってるので、条件付き分岐命令で分岐するコードを生成する
- バリエーションとしてランダム分岐予測がある(的中率50%)

分岐すると予測するほうが速い

```
    mov r0, #0
1:   add r0, r0, #1
    cmp r0, #10
    blt 1b
    ...
```

分岐しないと予測するほうが速い

```
    mov r0, #0
1:   add r0, r0, #1
    cmp r0, #10
    bge 2f
    b   1b
2:   ...
```

## 1レベル分岐予測

- 条件付き分岐のほとんどはループ処理である
  - 分岐する/しないは前回と同じであることが多い
  - 条件付き分岐が現れるアドレスの一部(タグ)をキーとして前回分岐した/しなかったを1ビットで保存、今回も前回と同じだろう……
- $2^{\text{タグの幅}}$  ビットのストレージが必要

普通は2ないし3ビットの飽和カウンタで実装する

タグ

分岐回数

0000

00

0004

10

...

...

fffc

xx

### 2レベル分岐予測

- 1レベル分岐予測にグローバルな分岐履歴の情報を組み合わせる
- 1レベル分岐予測はベクタだったが、これをテーブルとし、横軸にグローバル分岐履歴(普通は2ビット幅?)をとる
- 過去の状態によってパターンがある場合(引数チェックとか)に適用できる
- 1レベル分岐予測は飽和カウンタを用いたが、これはビットごとに前回・前々回・前々々回…の分岐した/しなかったを表す



分岐履歴

タグ

00

01

10

11

0000

00

10

00

11

0004

10

00

00

10

...

...

...

...

...

fffc

xx

xx

xx

xx

TAGE

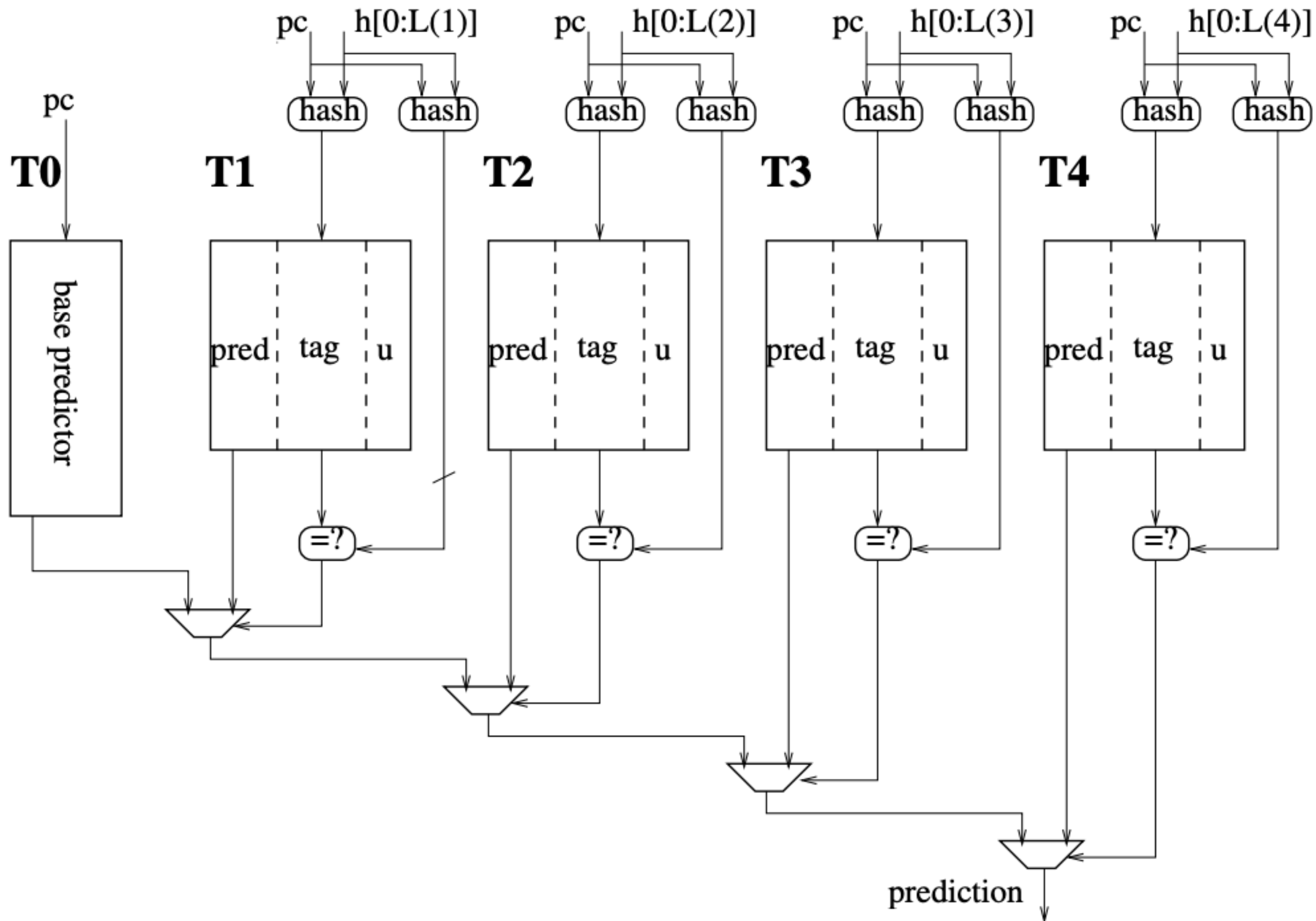


Figure 1: A 5-component TAGE predictor synopsis: a base predictor is backed with several tagged predictor components indexed with increasing history lengths

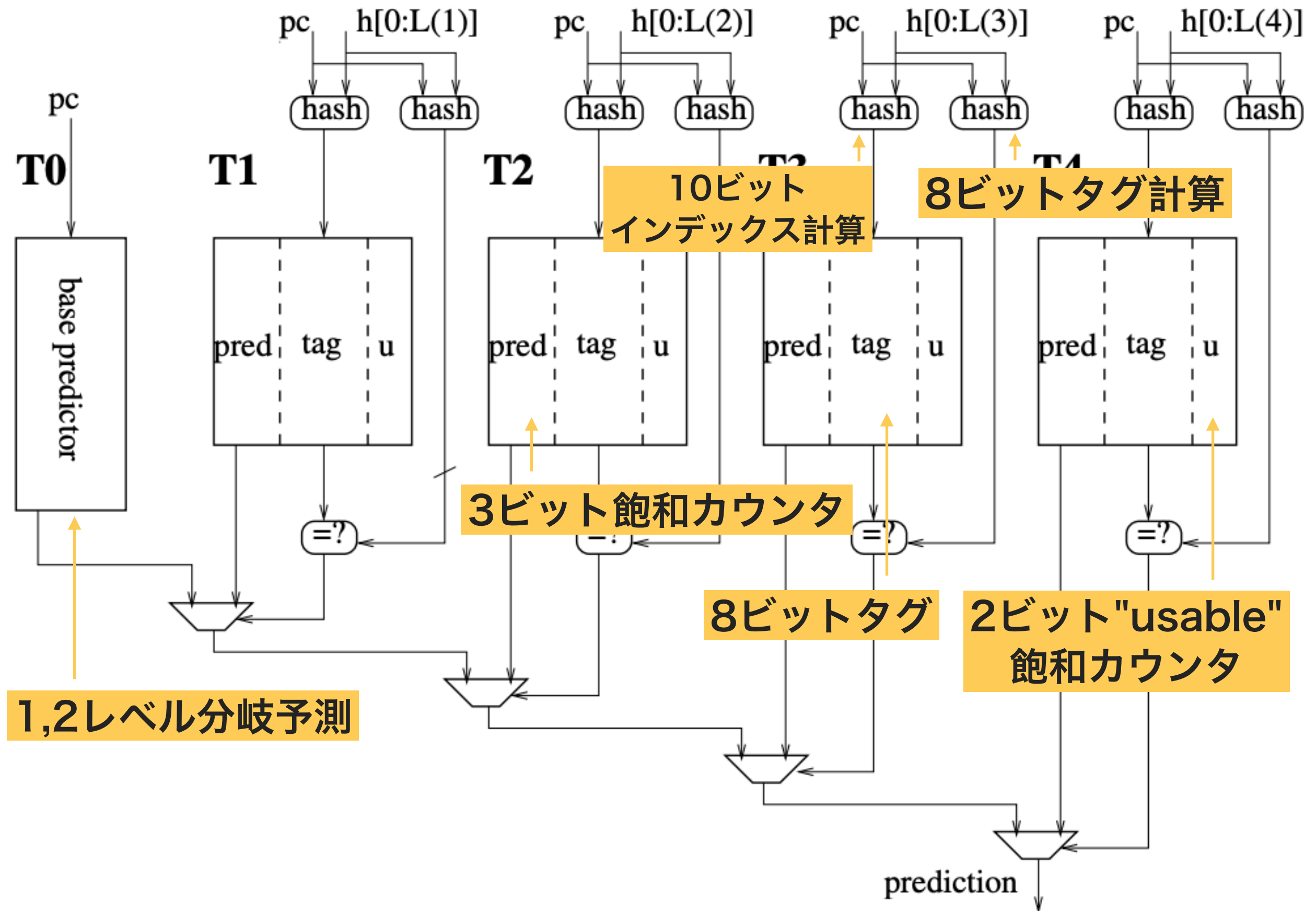


Figure 1: A 5-component TAGE predictor synopsis: a base predictor is backed with several tagged predictor components indexed with increasing history lengths

10 bit index	pred	8 bit tag	u
0000000000	000	11111111	00
0000000001	011	00000000	10
...	...	...	...
1111111111	xxx	xxxxxxxxxx	x

## 疑似コード1

```
// iは0スタート
#define L(i) (i*10 - 1) // 文献1
#define L(i) (alpha**i * L(1)) // 文献2
int pred_from_Ti(i, pc, h)
{
    index = calc_index(pc, h[0:L(i)]);
    tag    = calc_tag  (pc, h[0:L(i)]);
    if (GET_TAG(T[i][index]) != tag)
        return ERROR; // エイリアス
    return ((GET_PRED(T[i][index]) >> 2) & 1 // 飽和カウンタMSBをとる
           ? true // 分岐すると予想
           : false // 分岐しないと予想
    );
}
```

### 疑似コード2

```
int pred(pc, h)
{
    for (i = 4; i >= 1; i--) {
        result = pred_from_Ti(i - 1, pc, h);
        if (!IS_ERROR(result))
            return result;
    }
    // fallback to base predictor
    return pred_base(pc, h);
}
```

### 疑似コード3

```
int update(i          // 予測に使ったテーブル
           pc, h,
           pred,      // 最終的な予測内容
           succeed) // 予測は成功したか?
{
    for (k = 0; k < i; k++) {
        result = pred_from_Ti(k, pc, h);
        if (!IS_ERROR(result) && result != pred)
            // 最終予想と異なる予想が存在した
            // 最終的に成功したなら+1, 失敗したなら-1
            update_usable_counter(i, pc, h, succeed ? 1 : -1);
    }
    update_pred_counter(i, pc, h, succeed ? 1 : -1);
    if (!succeed && i < 3) {
        // 長い履歴を使うテーブルに新しい行を追加
        // どのテーブルに追加するか(k)は各テーブルのusableカウンタの値によって決める
        k = calc_alloc_k(i, pc, h);
        alloc_row(k, pc, h);
    }
}
```



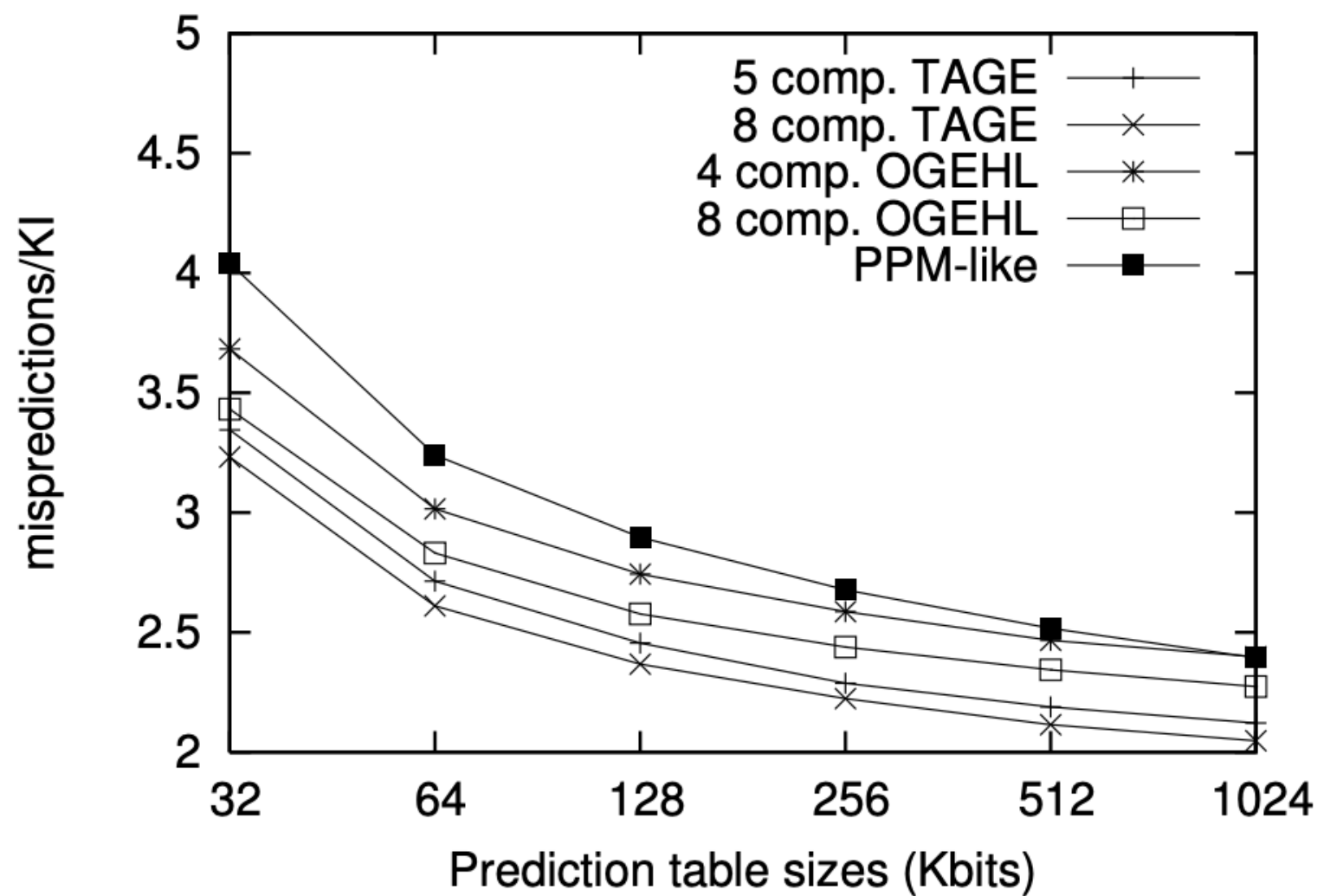


Figure 2: Conditional branch prediction accuracy on the TAGE predictor