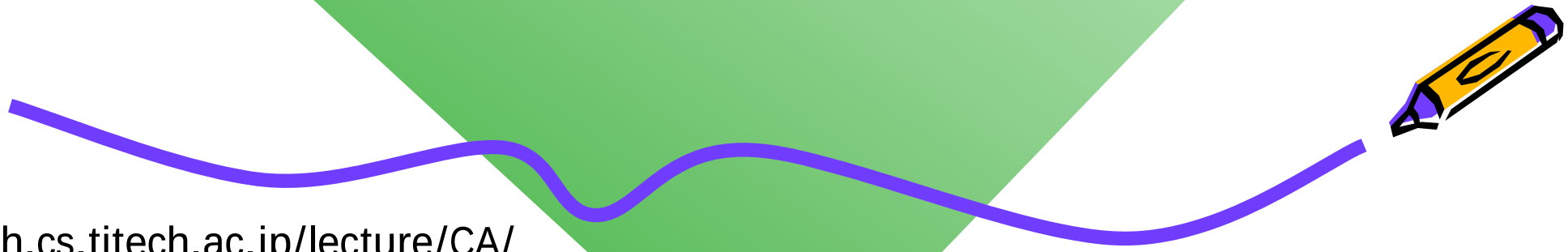
2018年度（平成30年度）版

Ver. 2018-10-25a

Course number: CSC.T363

# コンピュータアーキテクチャ
# Computer Architecture

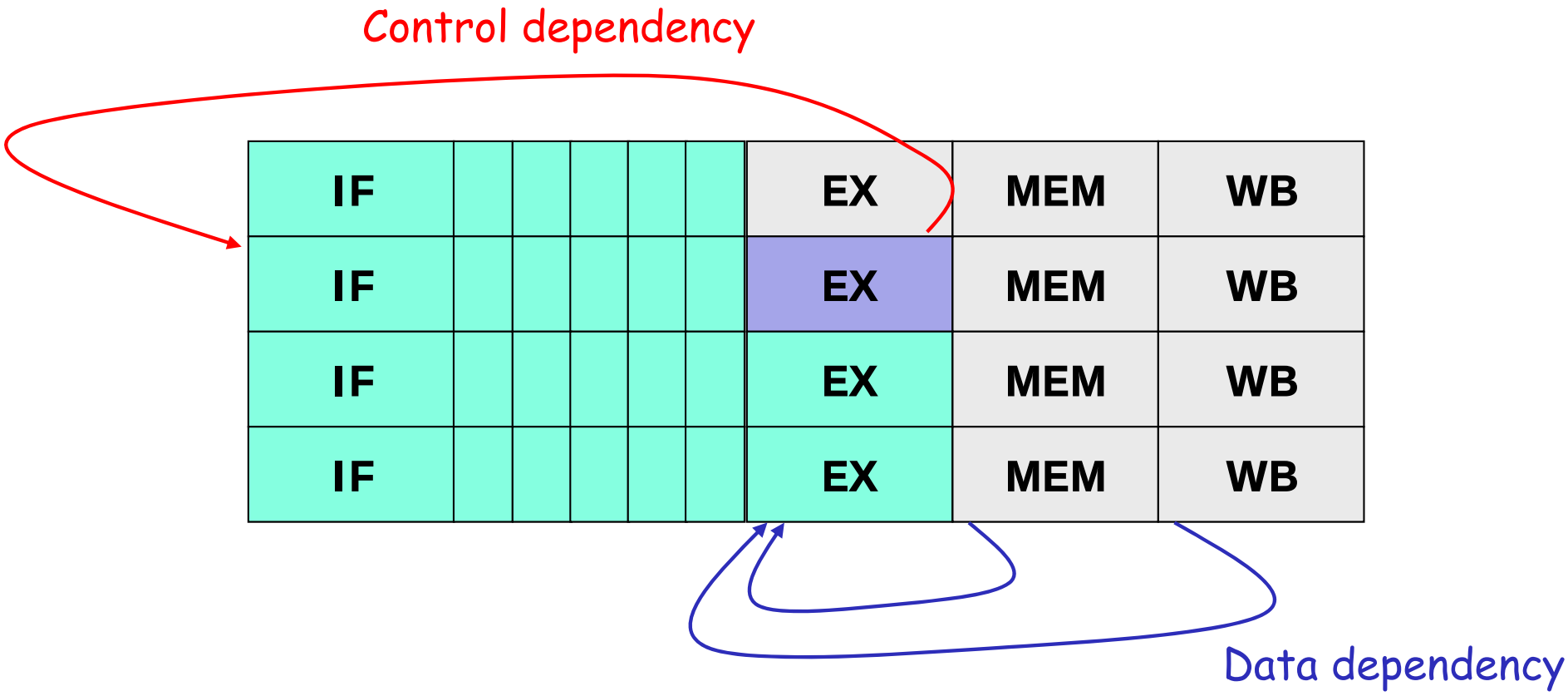
## 9. 分岐予測
## Branch Prediction



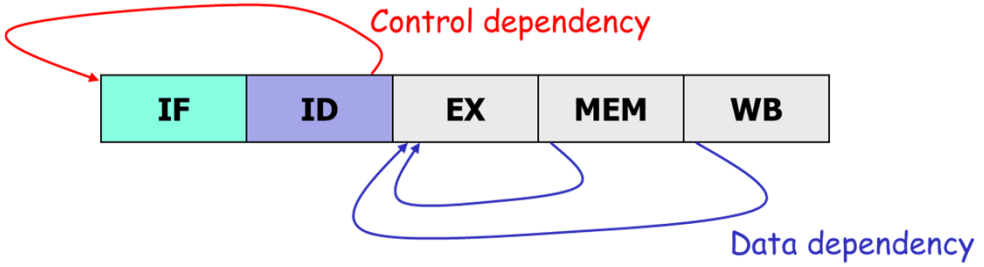
www.arch.cs.titech.ac.jp/lecture/CA/
Room No.W321
Tue 13:20-16:20, Fri 13:20-14:50

吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

# スーパースカラプロセッサと制約ループ




Conventional 5 stage pipeline



# 高いバンド幅の命令フェッチ

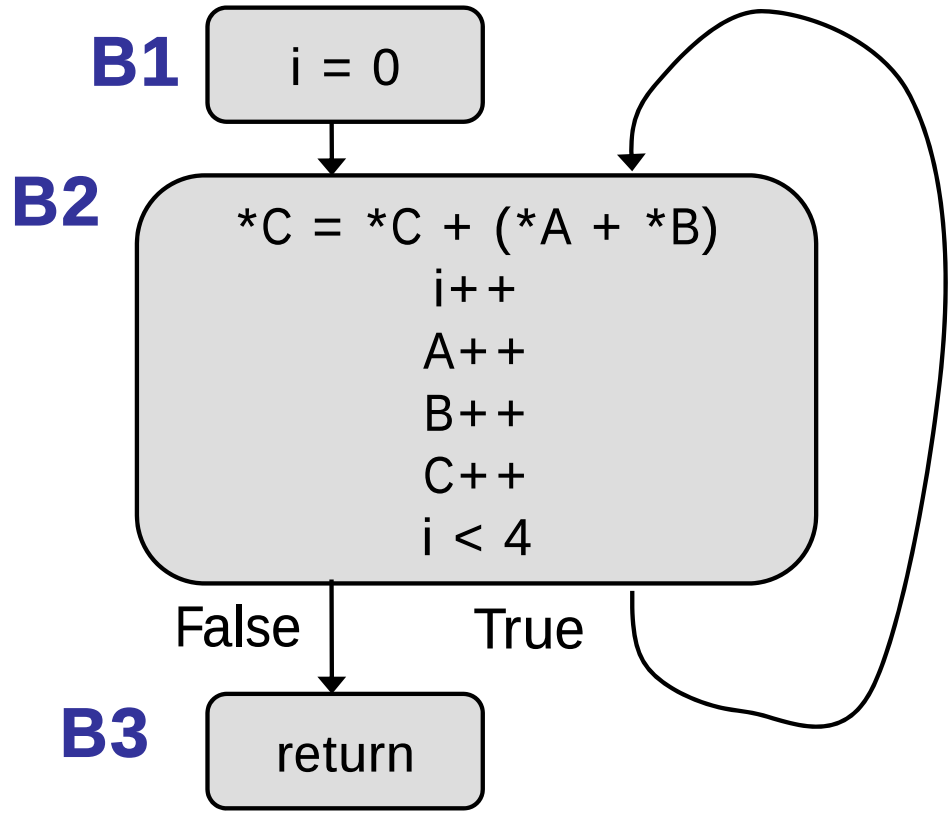
- パイプラインにバブルを生じさせないためには，条件分岐命令をフェッチした時に次の3つを予測 (prediction) しなければならない.
  - フェッチしている命令が分岐命令か？
  - 分岐方向
    - 分岐成立(Taken)か分岐不成立(Not taken, Untaken)か？
  - 分岐先アドレス(32bit)

- Speculation assuming the prediction is true
  - No penalty by a branch instruction when the prediction hits
  - Recovery when the miss is detected

# 分岐方向の予測(分岐予測)

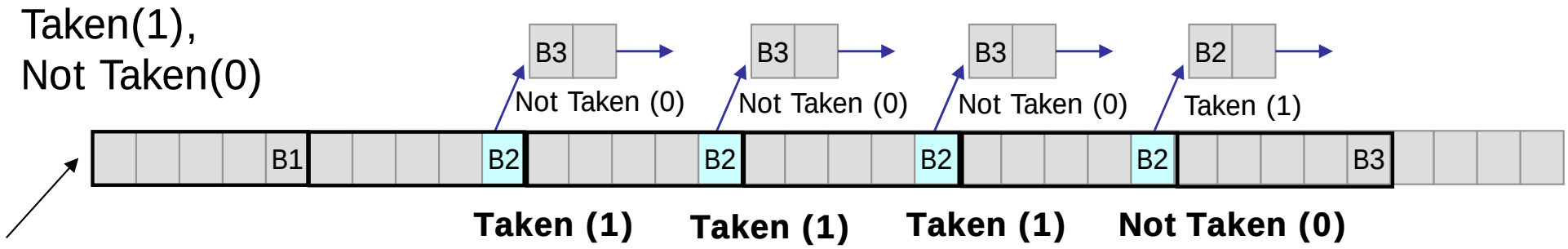
# サンプルプログラム Vector Add

```
#define VSIZE 4
void vadd(long *A, long *B, long *C){
  for(i=0; i<VSIZE; i++)
    C[i] += (A[i] + B[i]);
}
```



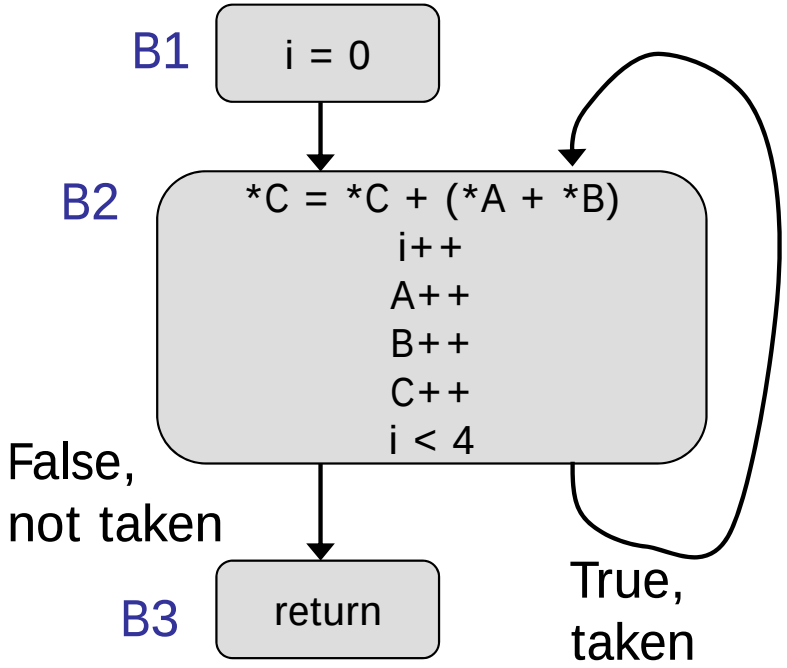

制御フローグラフ

# シンプルな分岐予測: Branch Always




処理すべき機械命令の列

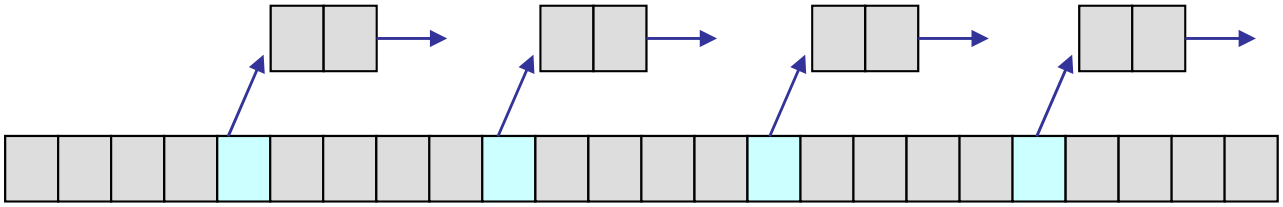
- Branch Always: 常に分岐が成立すると予測する.
  - 上の例では，予測成功率は75%，ミス率25%
  - 予測のためのメモリを必要としない.
  - 予測とよぶほどのものではない.



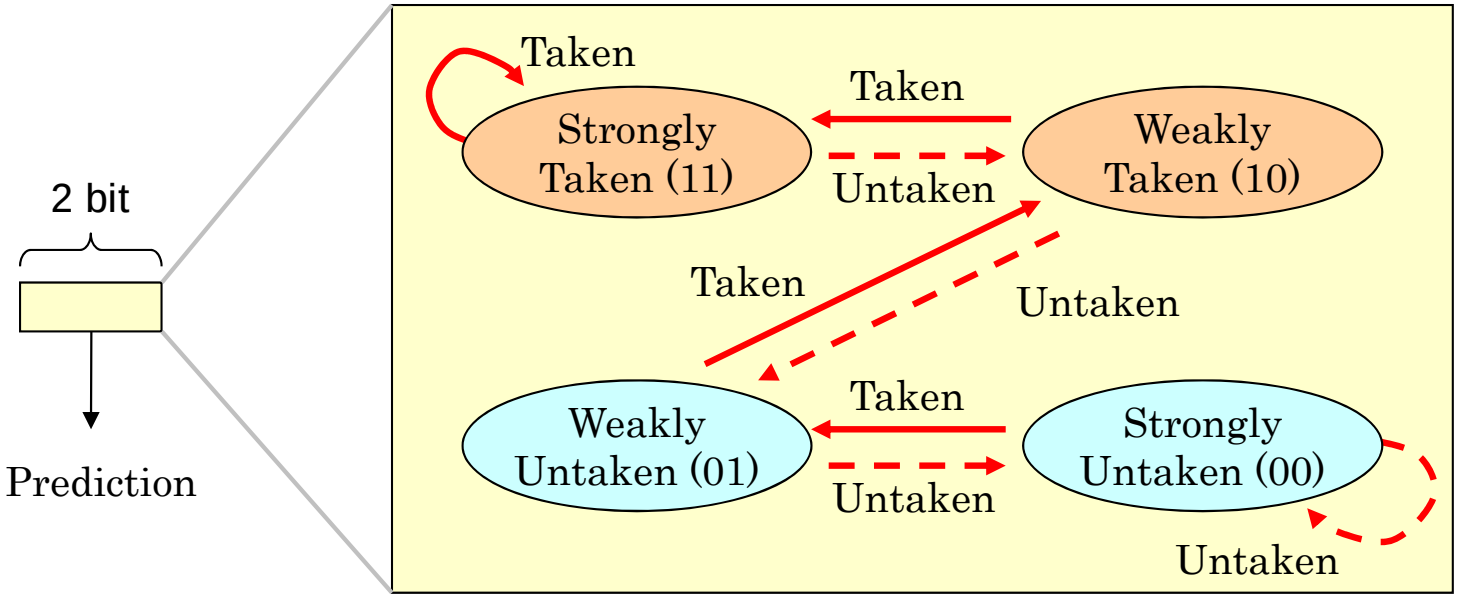
# シンプルな分岐予測: 2ビットカウンタ方式

トレースデータ
（分岐アドレス，分岐結果）

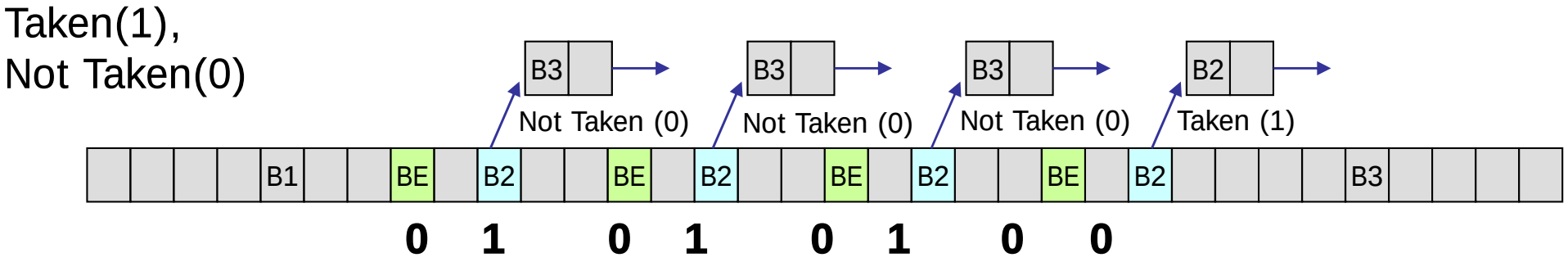
| | | | |
|---|---|---|---|
| 0040d6c5 | 1 | 0040d89c | 1 |
| 0040d6b8 | 1 | 0040d89c | 1 |
| 0040d6bc | 0 | 0040d89c | 1 |
| 0040d6c5 | 0 | 0040d89c | 1 |
| 0040d6df | 0 | 0040d89c | 1 |
| 0040d71f | 0 | 0040d89c | 1 |
| 0040d736 | 0 | 0040d89c | 0 |
| 0040d7ab | 0 | 0040d8a2 | 0 |
| 0040d7cd | 0 | 0040d8c0 | 1 |
| 0040d7f9 | 0 | 0040d8c4 | 0 |
| 0040d81e | 1 | 0040d8cd | 1 |
| 0040d7f9 | 1 | 0040d8c0 | 0 |
| 0040d81e | 1 | 0040d8c4 | 1 |
| 0040d7f9 | 0 | 0040d8cd | 1 |
| 0040d81e | 0 | 0040d8c0 | 1 |
| 0040d83d | 0 | 0040d8c4 | 0 |
| 0040d86d | 1 | 0040d8cd | 0 |
| 0040d86d | 1 | 0040d8e7 | 0 |
| 0040d86d | 1 | 0040d923 | 1 |
| 0040d86d | 1 | 0040d7ab | 0 |
| 0040d86d | 1 | 0040d7cd | 0 |
| 0040d86d | 1 | 0040d7f9 | 0 |



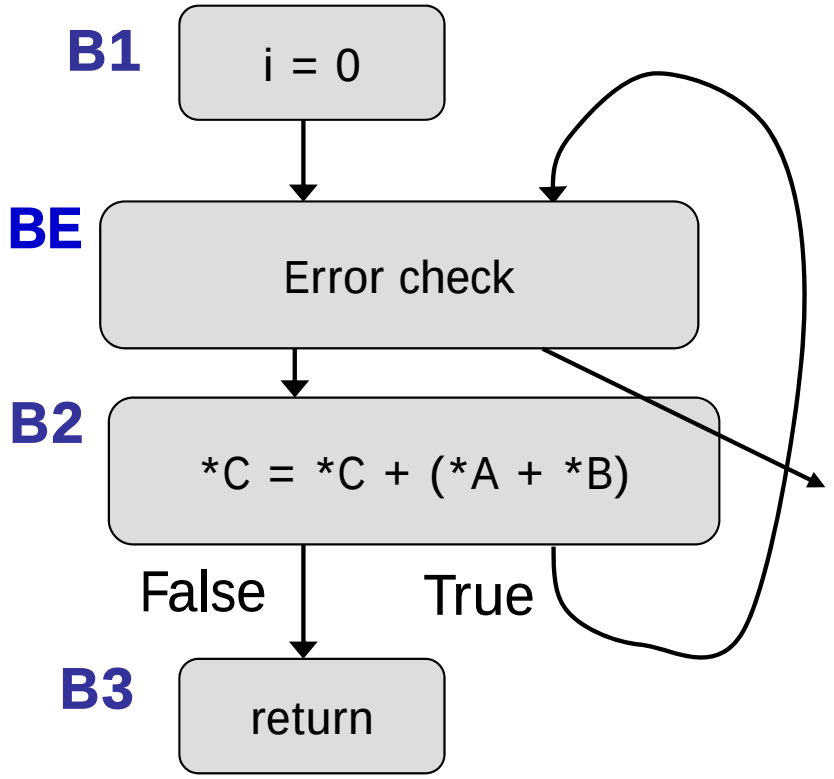
- 2ビットカウンタ方式
  - 大域的な偏り，局所性の利用
  - 2ビットカウンタの状態に応じて予測
  - 予測のためのメモリは2ビット
  - 状態の更新



# サンプルプログラム Vector Add

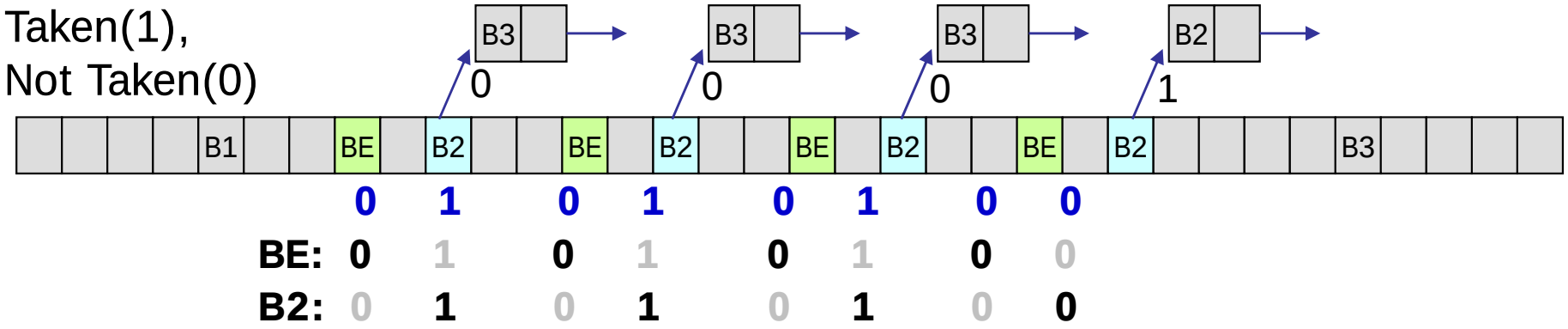



```
#define VSIZE 4
void vadd(long *A, long *B, long *C){
  for(i=0; i<VSIZE; i++) {
    if(A[i]<0) error_routine();
    C[i] += (A[i] + B[i]);
  }
}
```

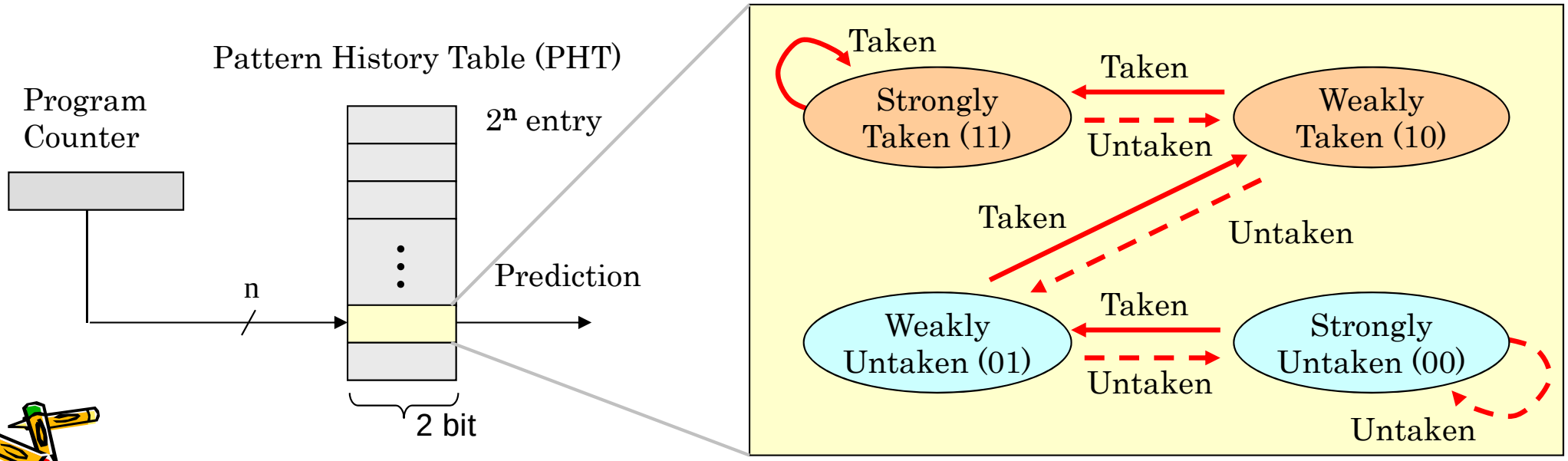



制御フローグラフ

# Bimodal branch predictor (ISCA 1981)

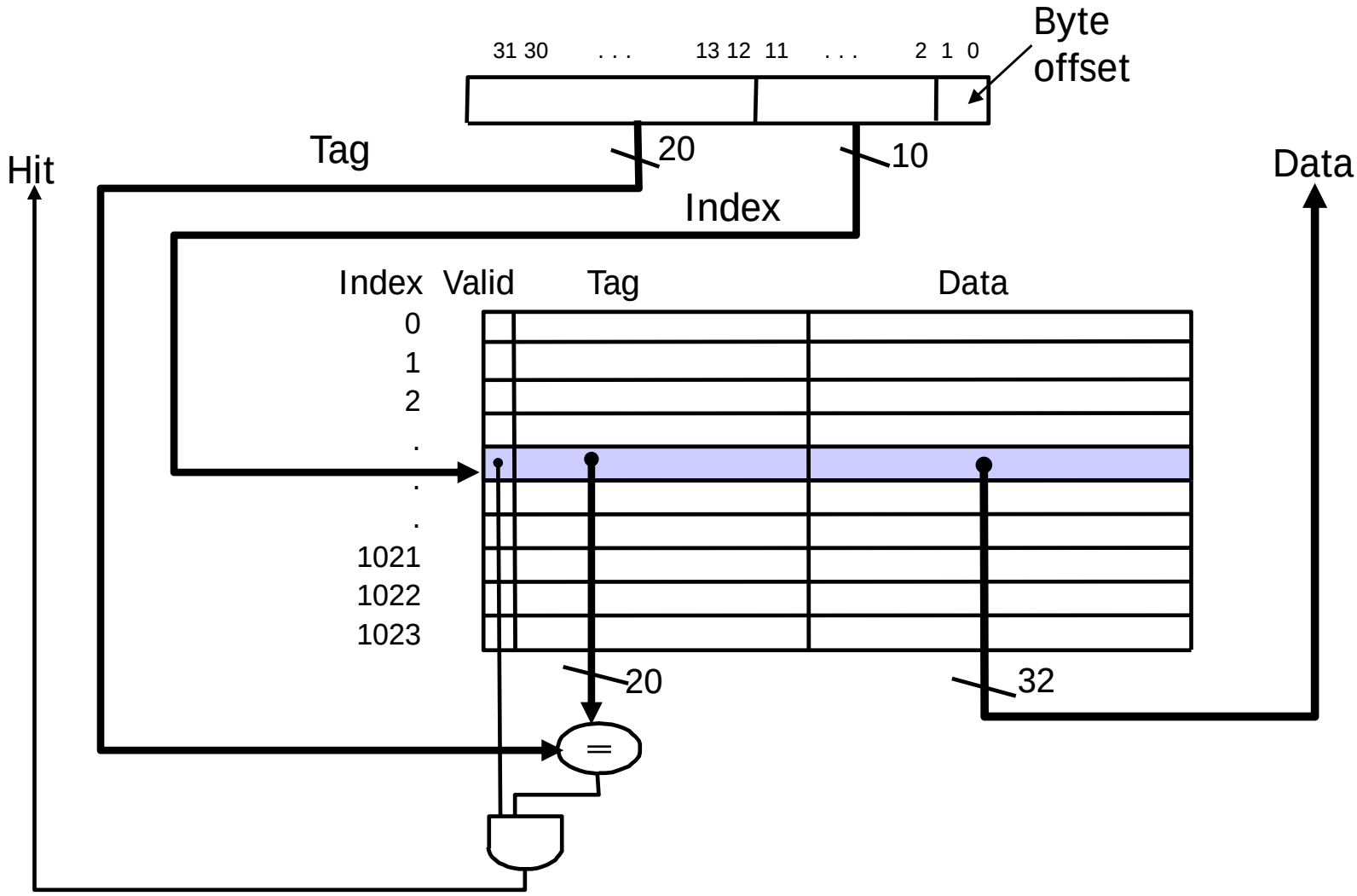



- 分岐アドレス（プログラムカウンタ）毎に履歴を切り替える
  - 分岐アドレスによりパターン履歴表（PHT）のインデックスを作成
- パターン履歴表は2ビットカウンタの配列.



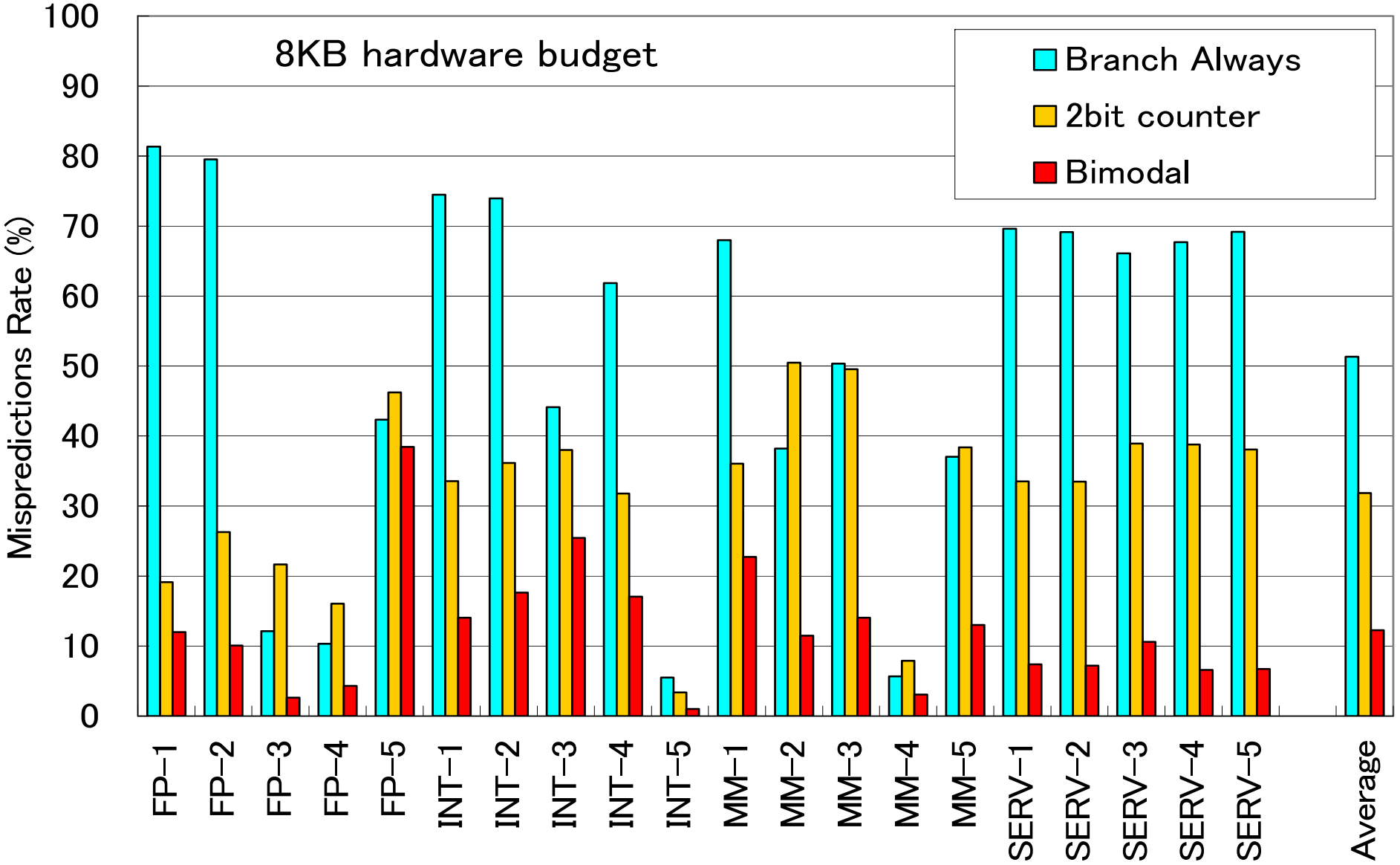
# MIPS Direct Mapped Cache Example

- One word/block, cache size = 1K words



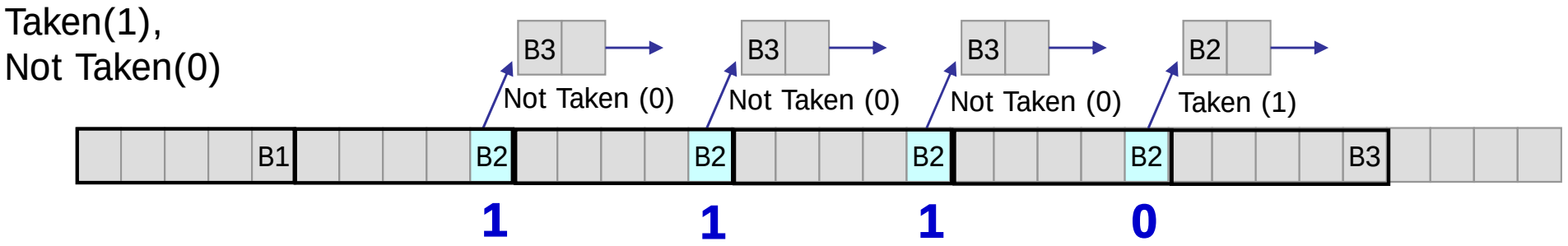

*What kind of locality are we taking advantage of?*

# ここまでの分岐予測の精度(予測ミス率)




Benchmark for CBP(2004) by Intel MRL and IEEE TC uARCH.

# Branch history（分岐履歴）




## B2の分岐履歴

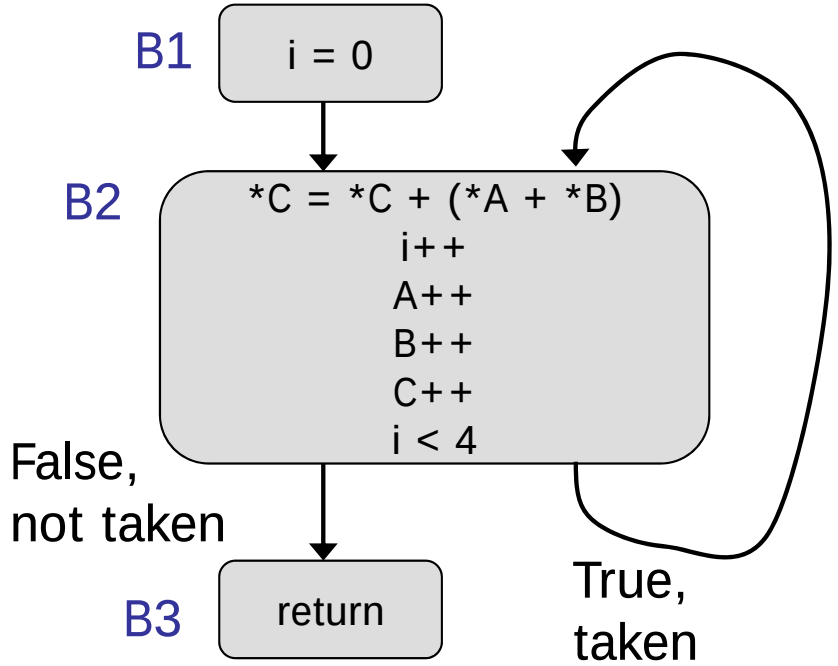
1110 ?

11101 ?

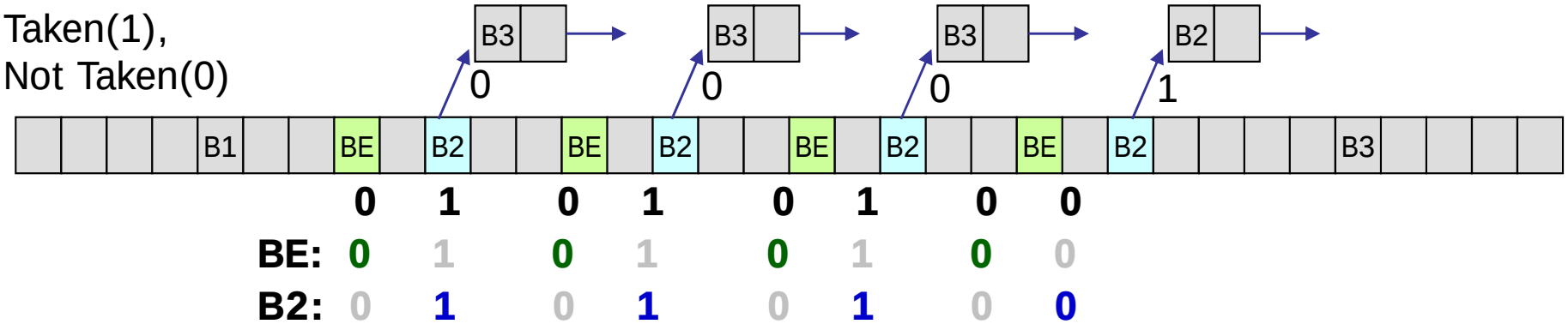
111011 ?

1110111 ?

11101110 ?



# ローカル分岐履歴とグローバル分岐履歴




BEの分岐履歴

0000 ?

00000 ?

000000 ?

0000000 ?

00000000 ?

ローカル分岐履歴

B2の分岐履歴

1110 ?

11101 ?

111011 ?

1110111 ?

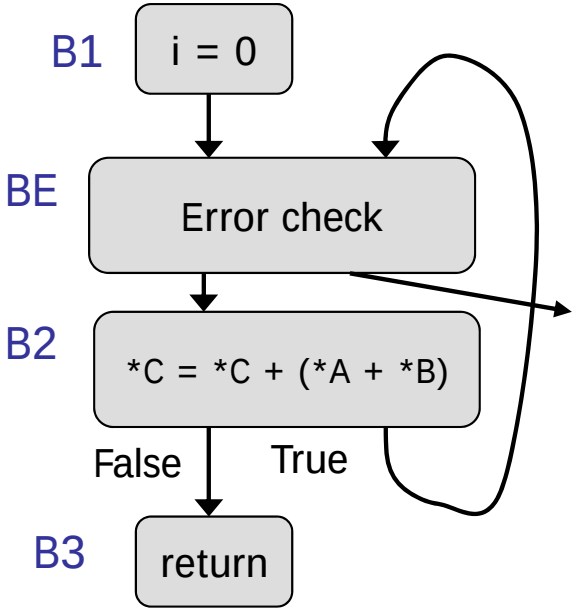
11101110 ?

ローカル分岐履歴

B2とBEの分岐履歴

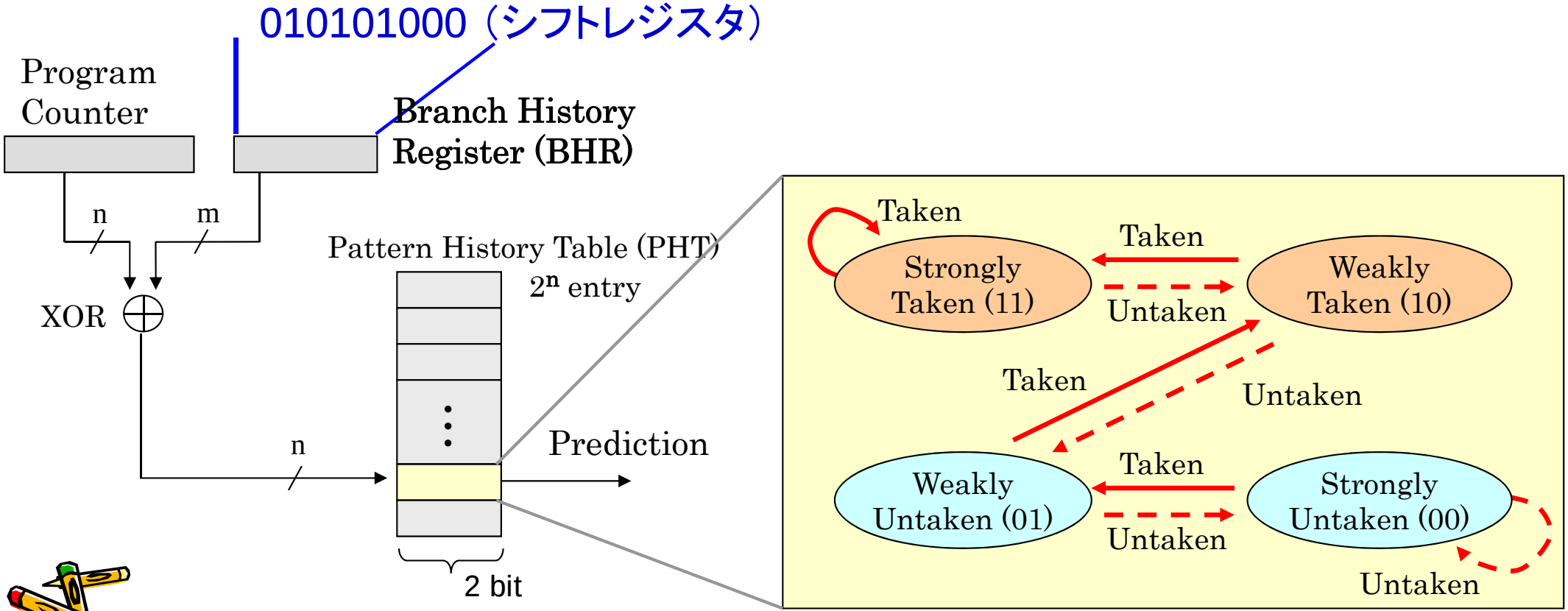
010101000 ?

グローバル分岐履歴



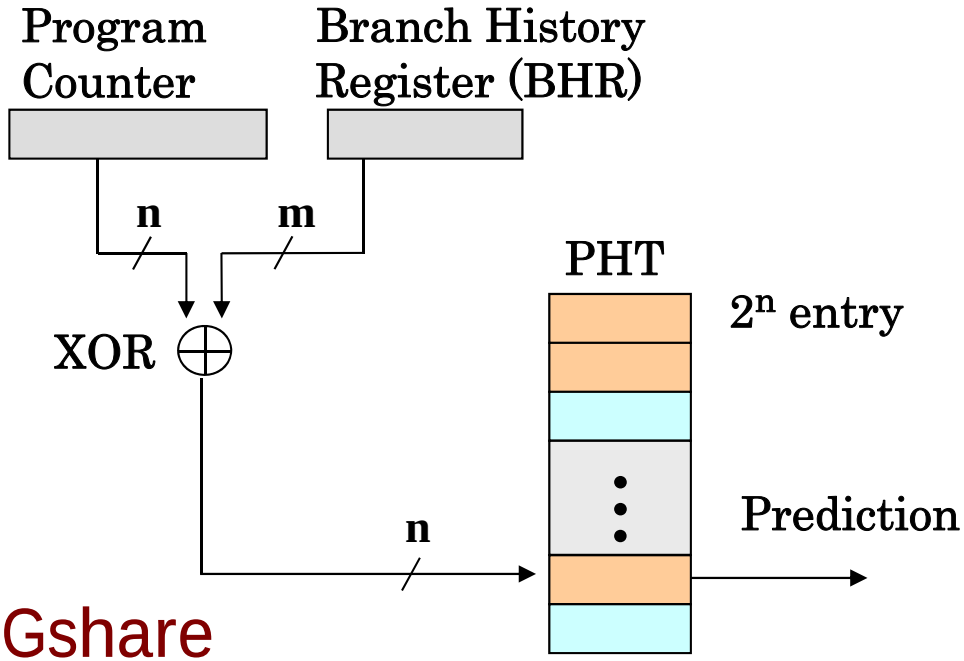
# Gshare (TR-DEC 1993)

- グローバル分岐履歴と分岐アドレスとの排他的論理和によりパターン履歴表へのインデックスを作成
  - パターン履歴表は2ビット飽和型カウンタの配列で，選択された2ビットカウンタの値により分岐方向を予測（bimodalと同じ）
  - 分岐結果を用いて，予測に利用したカウンタを更新

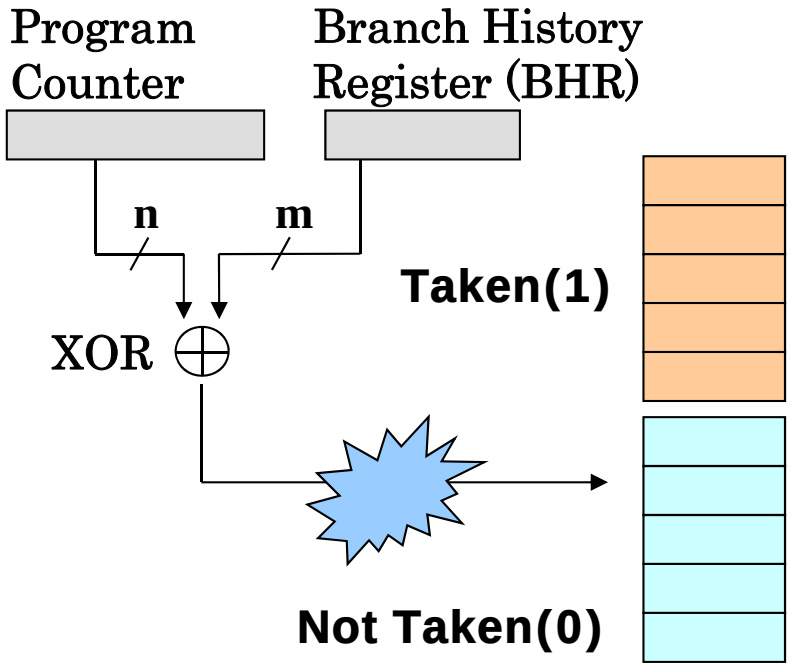


# Gshare の改良

- PHTの競合が発生して性能が低下
- PCとBHRによって特定される予測（成立，不成立）には偏りが存在するので，これらを別のテーブルに格納することで競合の悪影響を緩和
  - 分岐成立に偏っているもの
  - 分岐不成立に偏っているもの

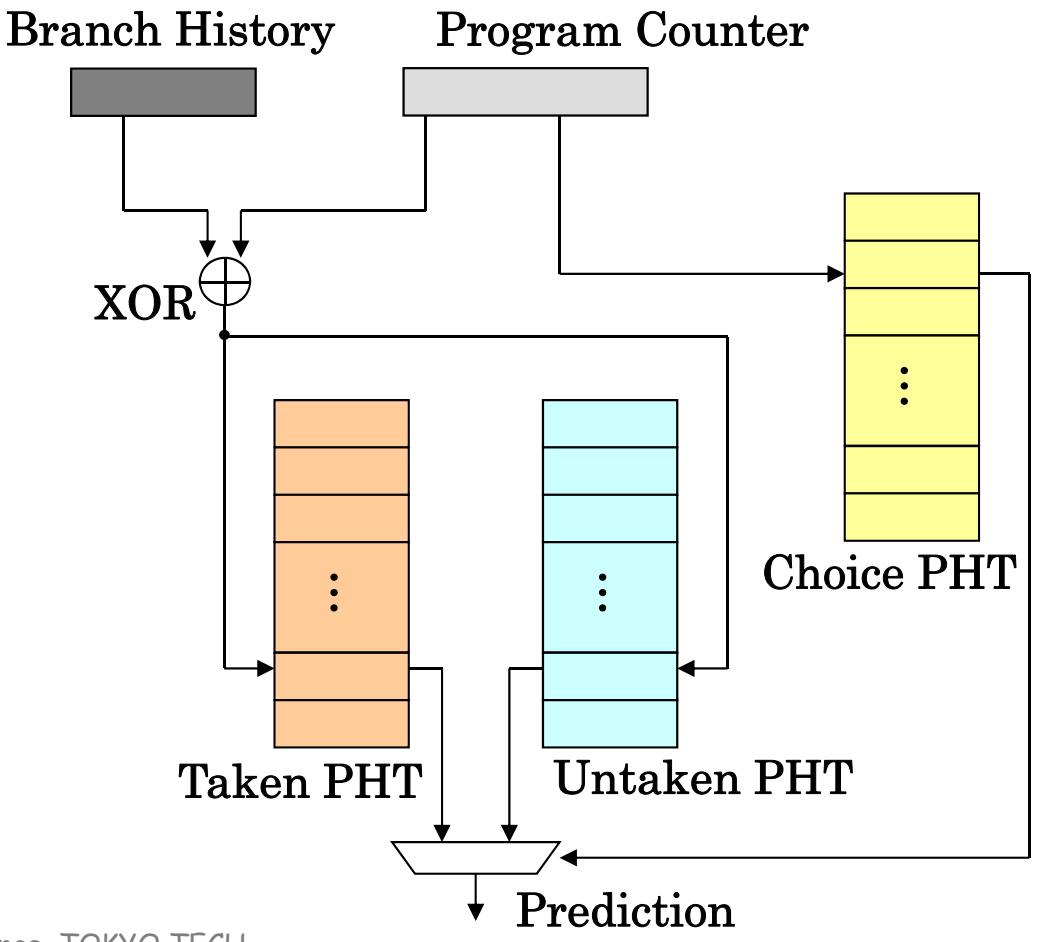



Gshare

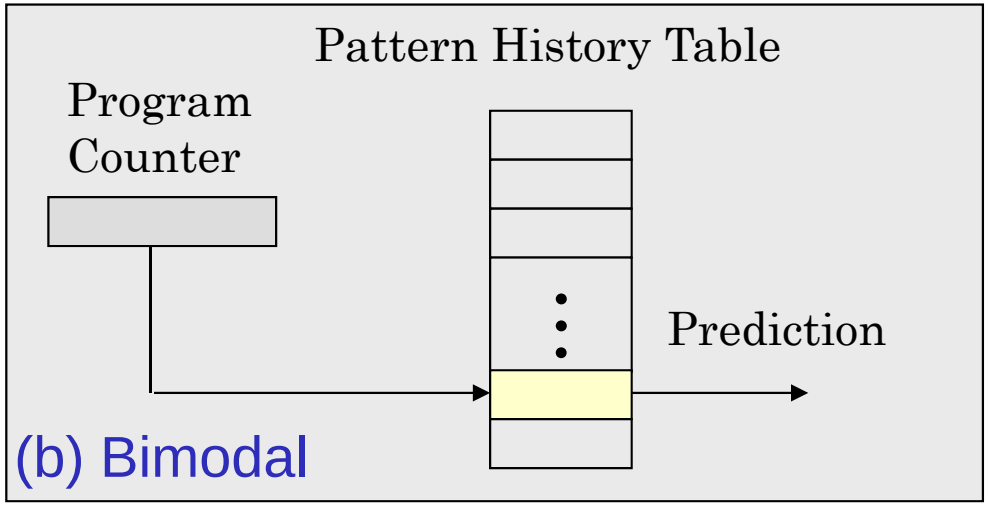


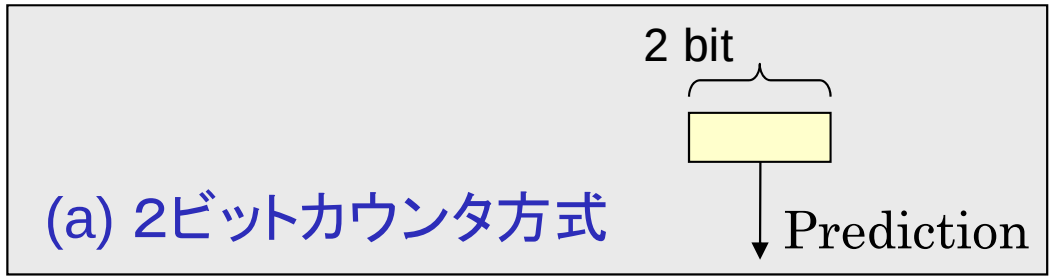
# Bimode (MICRO 1997)

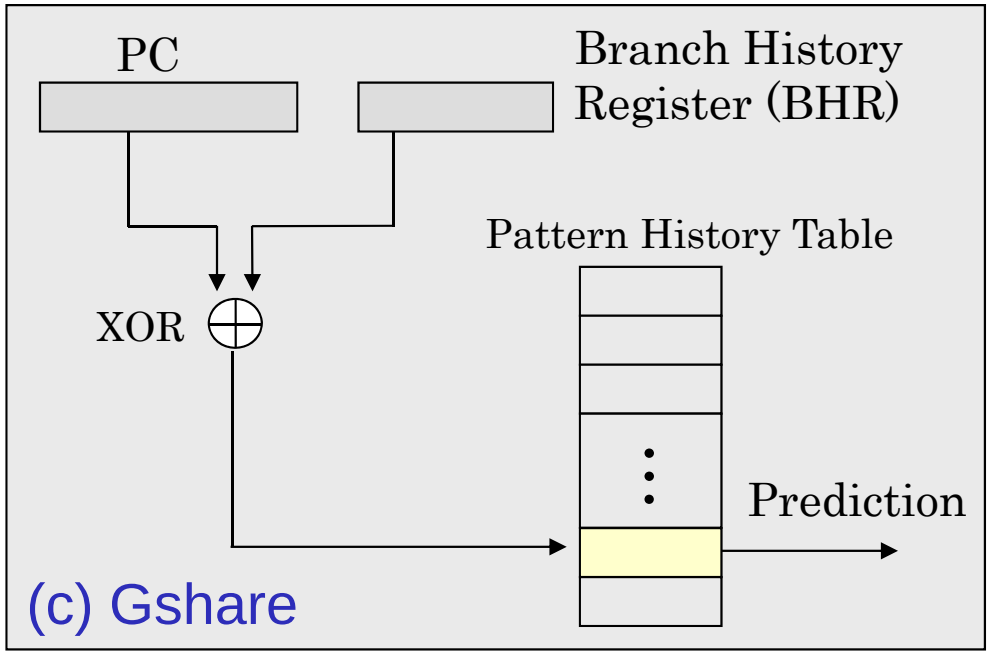
- 偏りを利用して競合の悪影響を緩和
  - 分岐成立に偏っているものをTaken PHTに格納
  - 分岐不成立に偏っているものをUntaken PHTに格納
- Choice PHT の内容で，どちらのテーブルを利用するか選択
- インデックスを工夫
  - Choice PHT は命令アドレス
  - Taken PHT, Untaken PHT は命令アドレスと分岐履歴

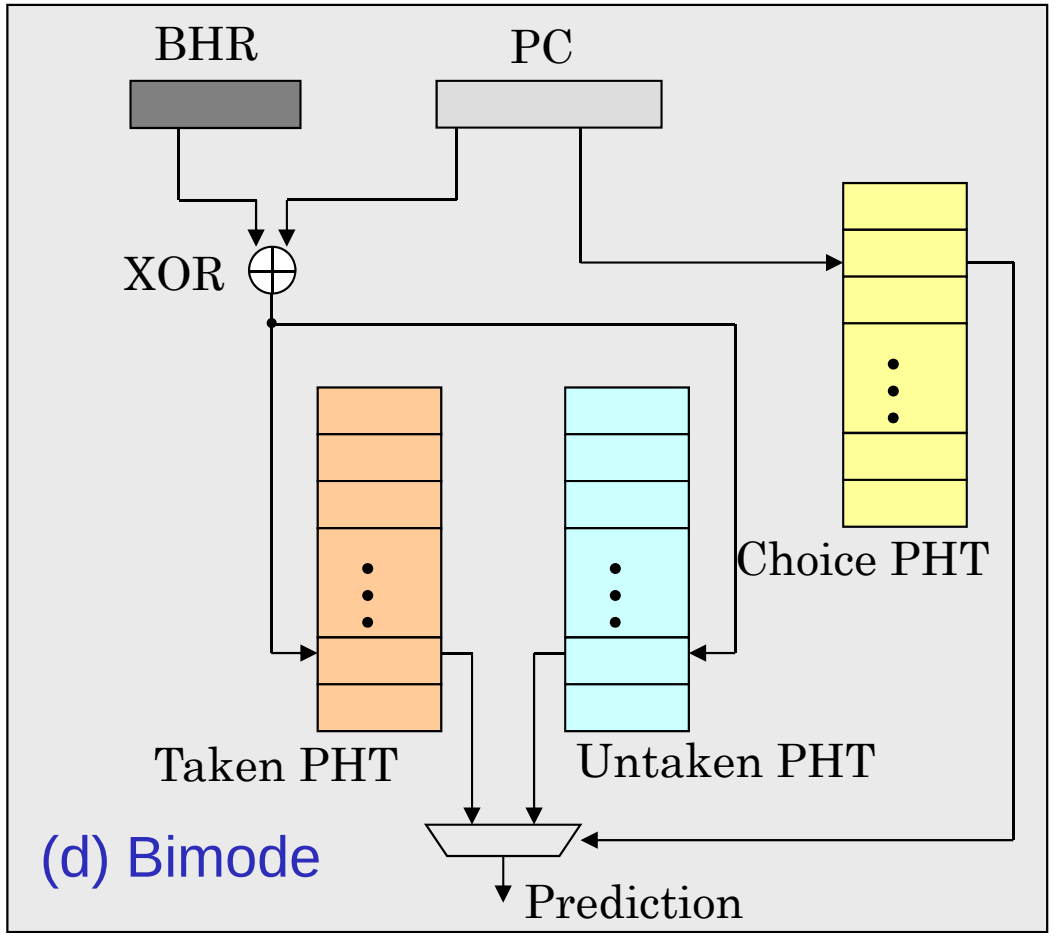


# 幾つかの分岐予測器

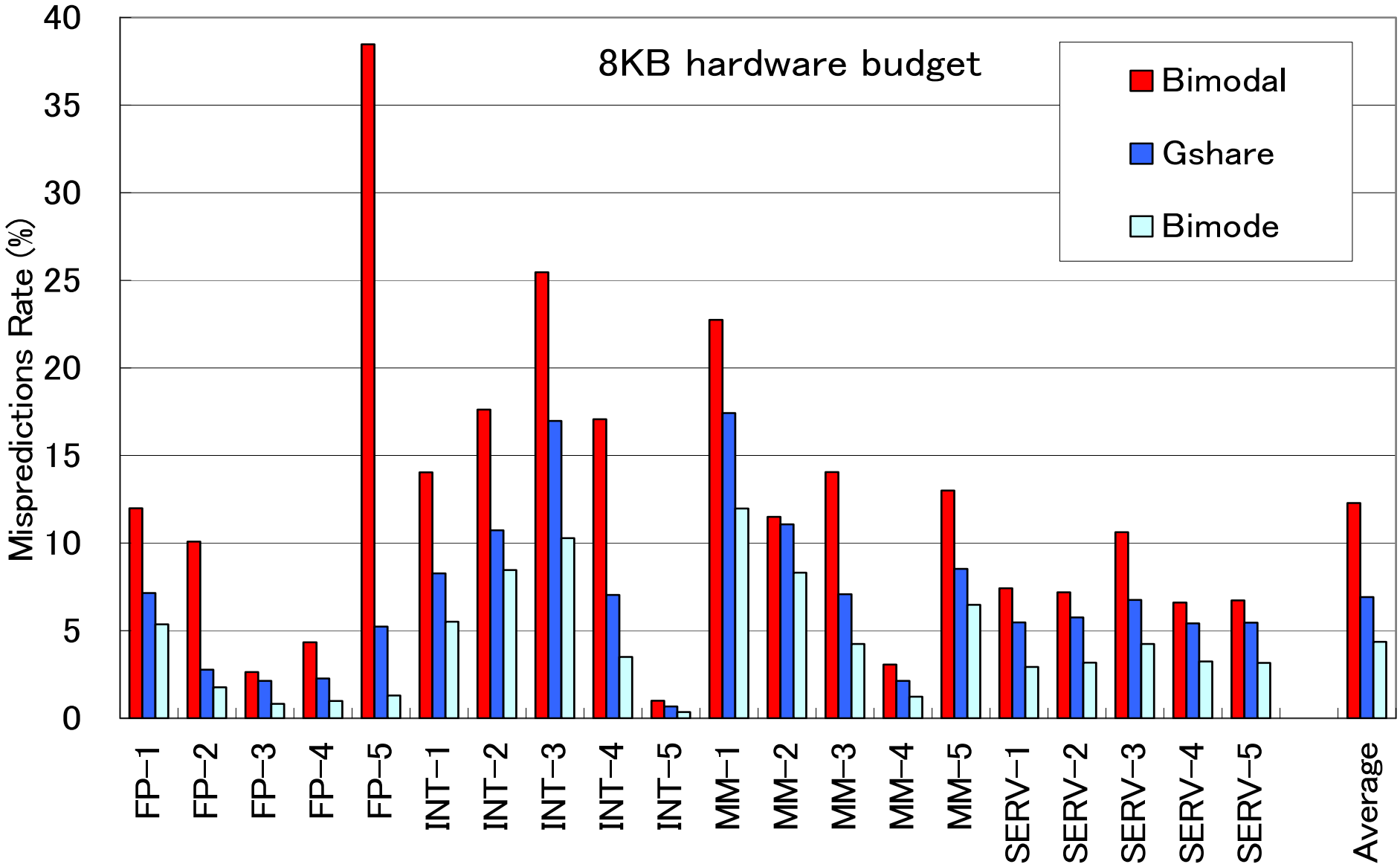

Pattern History Table
Program Counter
Prediction
(b) Bimodal


2 bit
(a) 2ビットカウンタ方式
Prediction


PC
Branch History Register (BHR)
XOR
Pattern History Table
Prediction
(c) Gshare


BHR
PC
XOR
Choice PHT
Taken PHT
Untaken PHT
(d) Bimode
Prediction

# 予測精度(予測ミス率)




Benchmark for CBP(2004) by Intel MRL and IEEE TC uARCH.

# BTB (Branch Target Buffer)

- フェッチしている命令が分岐命令か？
- 分岐先アドレス(32bit)

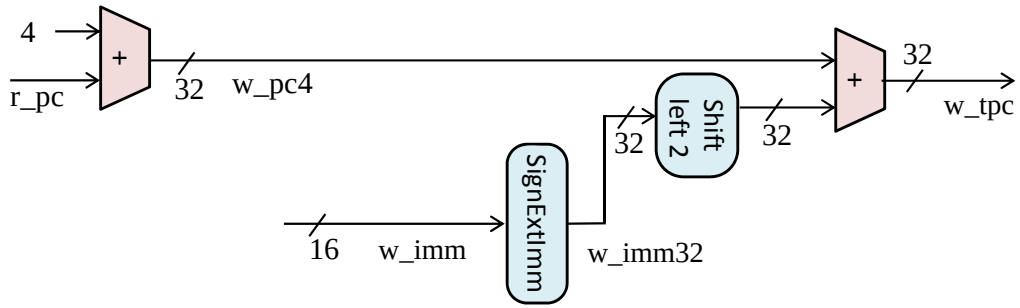
# MIPS Control Flow Instructions

- MIPS conditional branch instructions:

```
bne $s0, $s1, Lbl   # go to Lbl if $s0≠$s1
beq $s0, $s1, Lbl   # go to Lbl if $s0=$s1
```

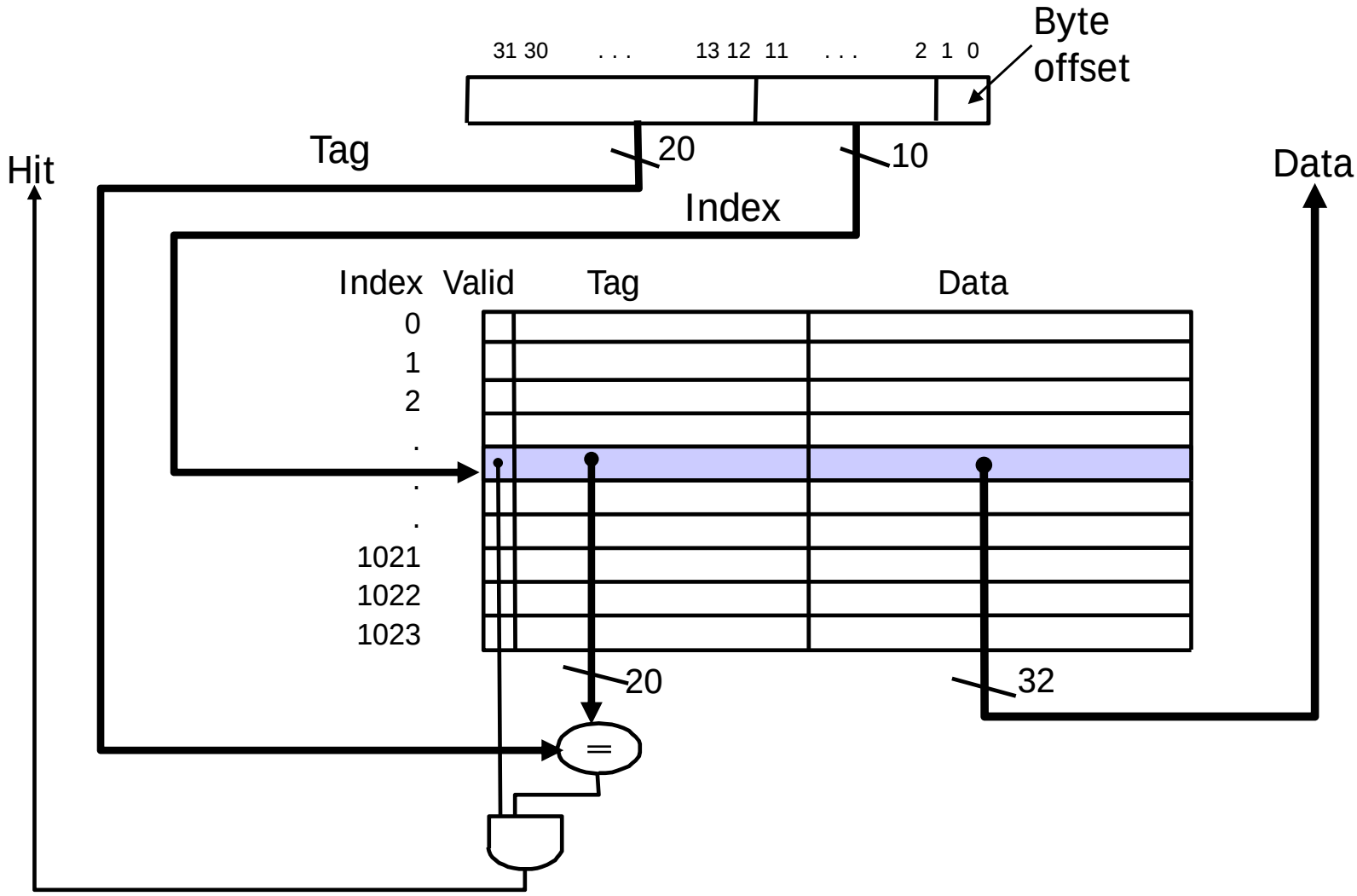
| | | | |
|---|---|---|---|
| Branch On Equal | beq | I | if(R[rs]==R[rt]) PC=PC+4+BranchAddr |
| Branch On Not Equal | bne | I | if(R[rs]!=R[rt]) PC=PC+4+BranchAddr |

(4) BranchAddr = {14{immediate[15]},immediate,2'b0 }



# MIPS Direct Mapped Cache Example

- One word/block, cache size = 1K words




*What kind of locality are we taking advantage of?*