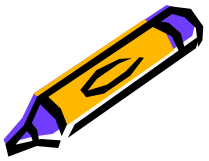
2023年度（令和5年）版

Ver. 2023-10-30a

Course number: CSC.T363

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

## 8. パイプラインプロセッサ (2) の残り
## Pipelined Processor (2)

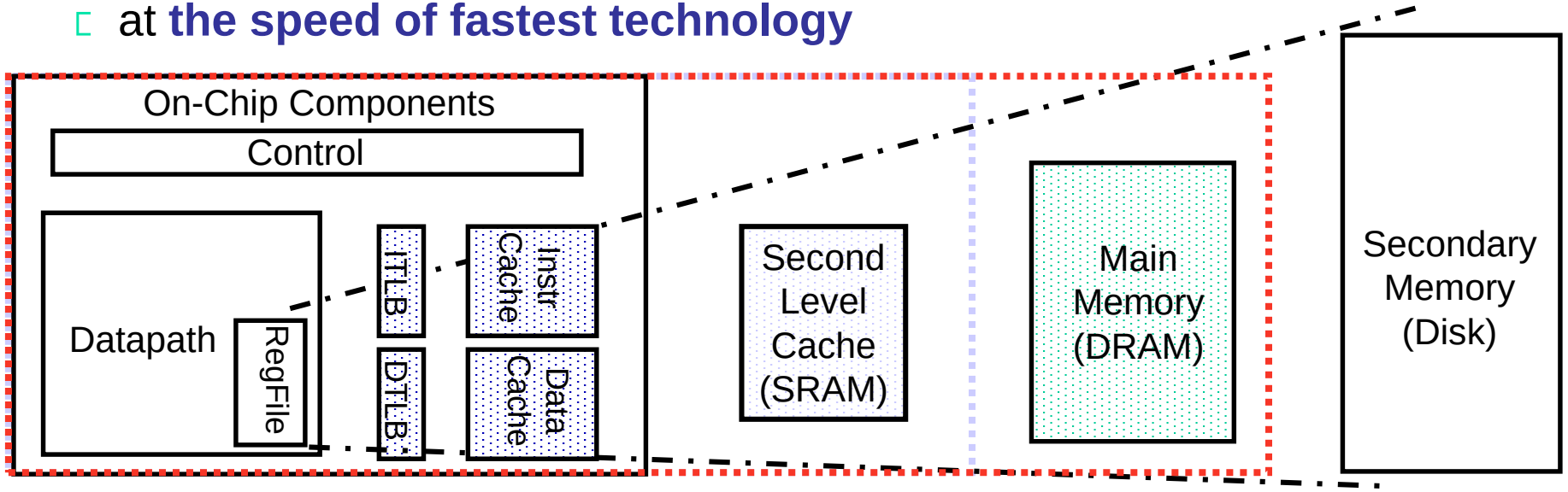


www.arch.cs.titech.ac.jp/lecture/CA/
Tue 13:30-15:10, 15:25-17:05
Fri 13:30-15:10

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

# A Typical Memory Hierarchy

- By taking advantage of **the principle of locality**（局所性）
  - Present **much memory** in **the cheapest technology**
  - at **the speed of fastest technology**




| | | | | | |
|---|---|---|---|---|---|
| **Speed (%cycles):** | ½'s | 1's | 10's | 100's | 1,000's |
| **Size (bytes):** | 100's | K's | 10K's | M's | G's to T's |
| **Cost:** | highest | | | | lowest |

TLB: Translation Lookaside Buffer

# DDR4 SDRAM


- 規格：DDR4 デスクトップ用 動作電圧：1.2v JEDEC準拠品（XMP2.0非搭載）
- 速度：PC4-25600 3200Mhz CL値：22-22-22-52 / 容量：32GBx2枚（64G
- 対応チップセット：Intel：Z590/H570/B560/H510/Z490 ・AMD：

| チップ規格 | モジュール規格 | メモリクロック (MHz) | バスクロック (MHz) | 転送速度 (GB/秒) | JEDEC 規格 |
|---|---|---|---|---|---|
| DDR4-800 | PC4-6400 | 50 | 400 | 6.4 | |
| DDR4-1066 | PC4-8528 | 66 | 533 | 8.5 | |
| DDR4-1333 | PC4-10664 | 83 | 666 | 10.6 | |
| DDR4-1600 | PC4-12800 | 100 | 800 | 12.8 | ○ |
| DDR4-1866 | PC4-14900 | 116 | 933 | 14.9 | ○ |
| DDR4-2133 | PC4-17000 | 133 | 1066 | 17.0 | ○ |
| DDR4-2400 | PC4-19200 | 150 | 1200 | 19.2 | ○ |
| DDR4-2666 | PC4-21333 | 166 | 1333 | 21.3 | ○ |
| DDR4-2800 | PC4-22400 | 175 | 1400 | 22.4 | |
| DDR4-2933 | PC4-23466 | 183 | 1466 | 23.4 | ○ |
| DDR4-3000 | PC4-24000 | 188 | 1500 | 24.0 | |
| DDR4-3200 | PC4-25600 | 200 | 1600 | 25.6 | ○ |
| DDR4-3300 | PC4-26400 | 206 | 1650 | 26.4 | |

Amazon, Wikipedia

# DDR SDRAM Latency

| テクノロジー | モジュール速度 (MT/s) | クロックサイクル時間（ns） | CASレイテンシー | レイテンシー（ns） |
|---|---|---|---|---|
| SDR | 100 | 8.00 | 3 | 24.00 |
| SDR | 133 | 7.50 | 3 | 22.50 |
| DDR | 333 | 6.00 | 2.5 | 15.00 |
| DDR | 400 | 5.00 | 3 | 15.00 |
| DDR2 | 667 | 3.00 | 5 | 15.00 |
| DDR2 | 800 | 2.50 | 6 | 15.00 |
| DDR3 | 1333 | 1.50 | 9 | 13.50 |
| DDR3 | 1600 | 1.25 | 11 | 13.75 |
| DDR4 | 1866 | 1.07 | 13 | 13.93 |
| DDR4 | 2133 | 0.94 | 15 | 14.06 |
| DDR4 | 2400 | 0.83 | 17 | 14.17 |
| DDR4 | 2666 | 0.75 | 19 | 14.25 |
| DDR4 | 2933 | 0.68 | 21 | 14.32 |
| DDR4 | 3200 | 0.62 | 22 | 13.75 |
| DDR5 | 4800 | 0.42 | 40 | 16.67 |

https://www.crucial.jp/articles/about-memory/difference-between-speed-and-latency

# Latest Academic Paper

## Register File Prefetching

Sudhanshu Shukla
sudhanshu.shukla@intel.com
Processor Architecture Research Lab, Intel Labs

Sumeet Bandishte
sumeet.bandishte@intel.com
Processor Architecture Research Lab, Intel Labs

Jayesh Gaur
jayesh.gaur@intel.com
Processor Architecture Research Lab, Intel Labs

Sreenivas Subramoney
sreenivas.subramoney@intel.com
Processor Architecture Research Lab, Intel Labs

*ISCA '22, June 18–22, 2022, New York, NY, USA*

## 4 EVALUATION METHODOLOGY

We simulate a dynamically scheduled x86 core using an in-house, execution driven cycle-accurate simulator that models a 5-wide OOO core clocked at 4 GHz. The core parameters are similar to the

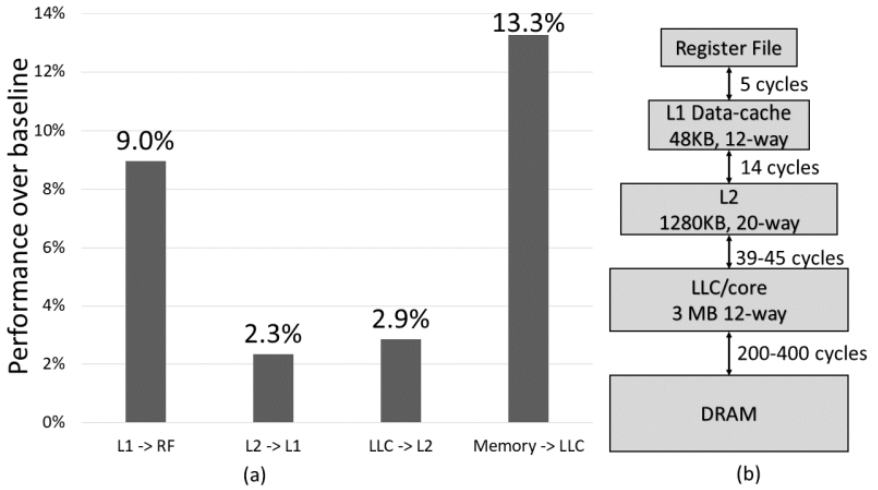
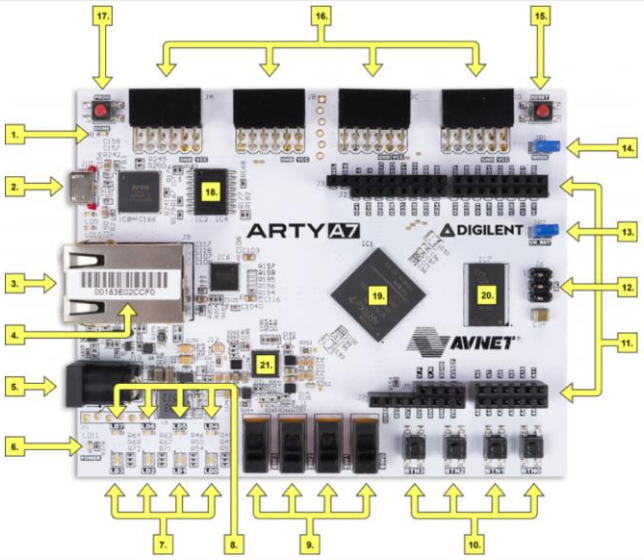



**Figure 1: Performance headroom from oracle prefetching across the different levels of cache and memory hierarchy for a processor similar to Intel Tiger Lake [22]. An oracle prefetching from level N to level N-1 will ensure all hits at level N will be served at the latency of level N -1. Prefetching between L1 to RF and between Memory to LLC provide the highest headroom.**

# Arty A7 and server computer of ACRi Room




| Callout | Description | Callout | Description | Callout | Description |
|---|---|---|---|---|---|
| 1 | FPGA programming DONE LED | 8 | User RGB LEDs | 15 | chipKIT processor reset |
| 2 | Shared USB JTAG / UART port | 9 | User slide switches | 16 | Pmod connectors |
| 3 | Ethernet connector | 10 | User push buttons | 17 | FPGA programming reset button |
| 4 | MAC address sticker | 11 | Arduino/chipKIT shield connectors | 18 | SPI flash memory |
| 5 | Power jack for optional external supply | 12 | Arduino/chipKIT shield SPI connector | 19 | Artix FPGA |
| 6 | Power good LED | 13 | chipKIT processor reset jumper | 20 | Micron DDR3 memory |
| 7 | User LEDs | 14 | FPGA programming mode | 21 | Dialog Semiconductor DA9062 power supply |

## FPGA Server

- CPU: Core i9 (**8 core /16 thread**)
- Memory: DDR4 **128GB** (32GB x 4)
- Storage: SSD M.2 **1TB x 2**

Arty A7-35T (XC7A35TICSG324-1L, Artix-7 device)

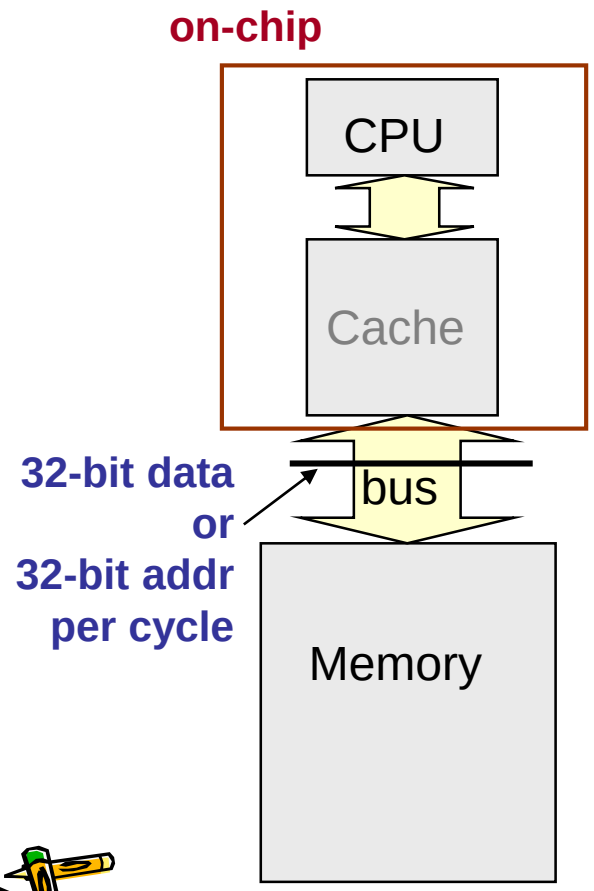
**Arty A7: Artix-7 FPGA Development Board**

$159.00

★★★★☆ (22 reviews) Write a Review

# Memory Systems that Support Caches

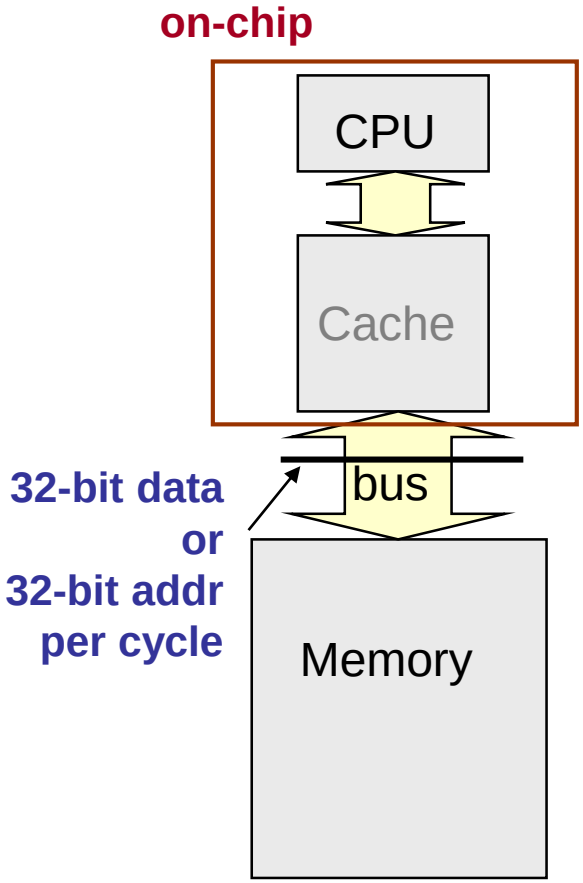
- The off-chip interconnect and memory architecture can affect overall system performance **in dramatic ways**




One word wide organization (one word wide bus and one word wide memory)

- **Assume**
  1. **1** clock cycle to send the address
  2. **25** clock cycles for DRAM **cycle time**, **8** clock cycles **access time**
  3. **1** clock cycle to return a word of data
- **Memory-Bus** to Cache **bandwidth**
  - **number of bytes transferred from memory to cache per clock cycle**

# One Word Wide Memory Organization

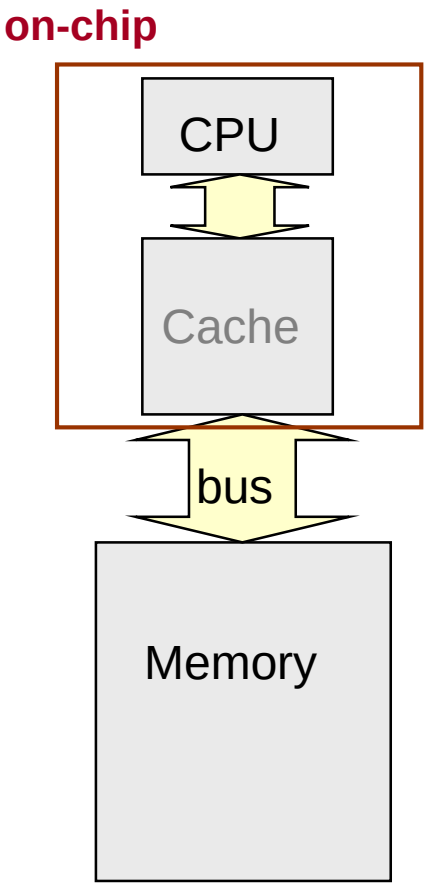


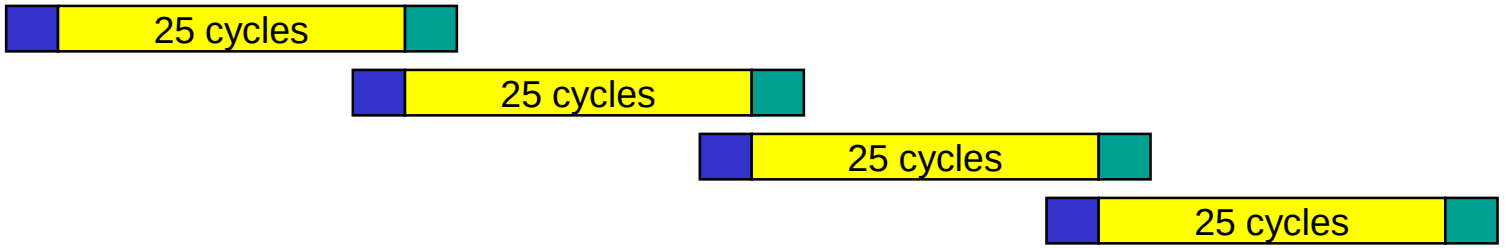

- The pipeline stalls the number of cycles for **one word** (32bit) from memory
  - **1** cycle to send address
  - **25** cycles to read DRAM
  - **1** cycle to return data
  - **27** **total clock cycles** miss penalty


- Number of bytes transferred per clock cycle (**bandwidth**) for a single miss
  - **4 / 27 = 0.148** bytes per clock

# One Word Wide Memory Organization, con't

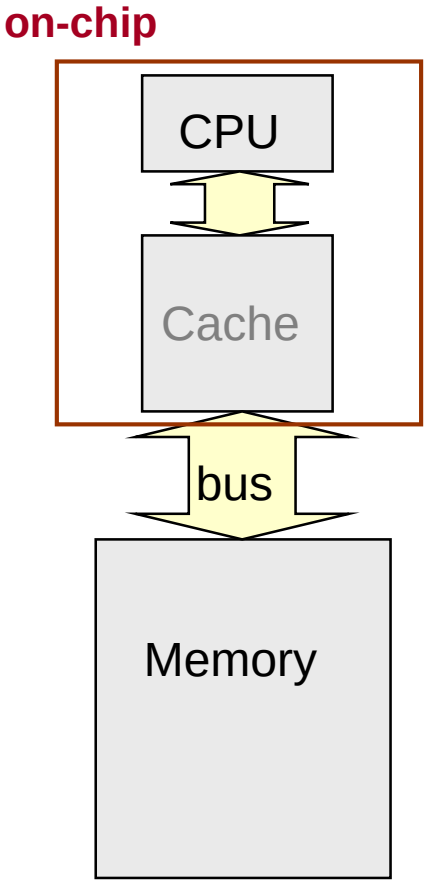
- What if the block size is **four words**?
  - **1** cycle to send 1st address
  - **4 * 25 = 100** cycles to read DRAM
  - **1** cycle to return last data word
  - **102 total clock cycles** miss penalty



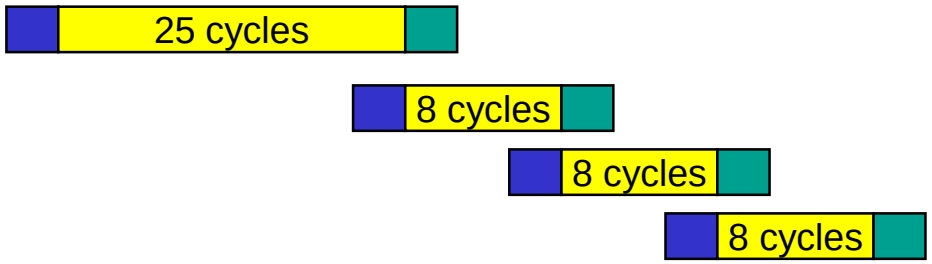




- Number of bytes transferred per clock cycle (**bandwidth**) for a single miss
  - **(4 x 4) / 102 = 0.157** bytes per clock

# One Word Wide Memory Organization, con't

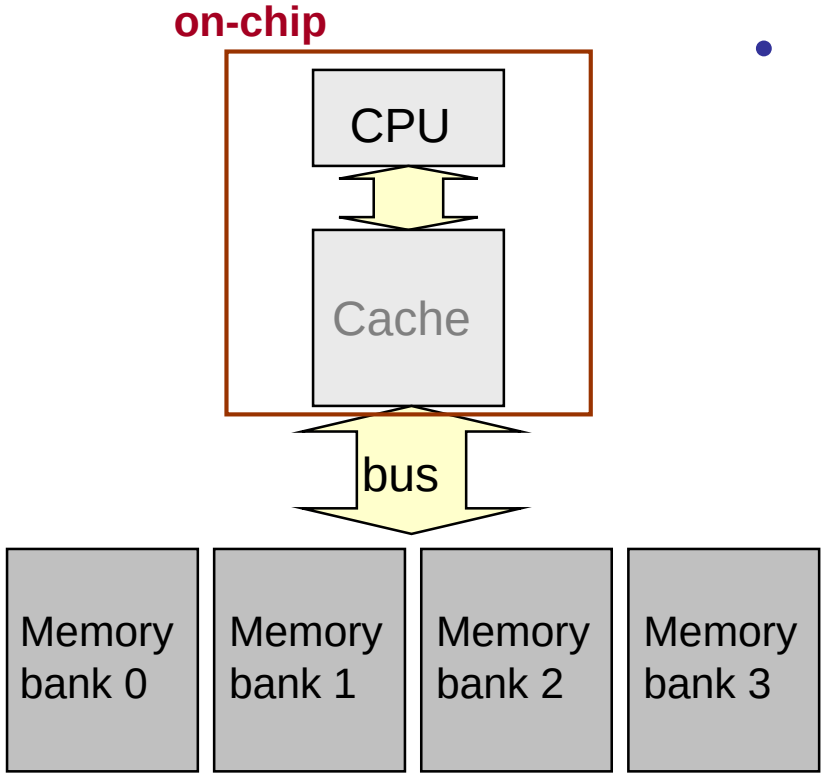



- What if the block size is **four words** and if a **page mode DRAM** is used?
  - **1** cycle to send 1st address
  - **25 + (3 * 8) = 49** cycles to read DRAM
  - **1** cycle to return last data word
  - **51 total clock cycles** miss penalty



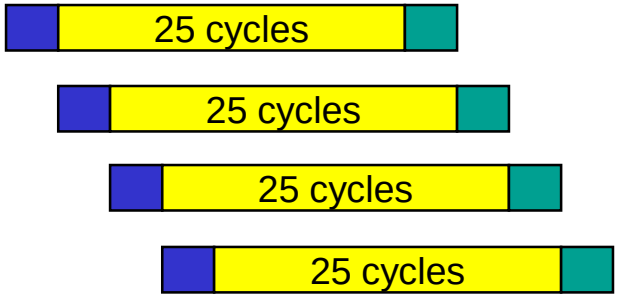

- Number of bytes transferred per clock cycle (**bandwidth**) for a single miss
  - **(4 x 4) / 51 = 0.314** bytes per clock

# Interleaved（インターリーブ）Memory Organization

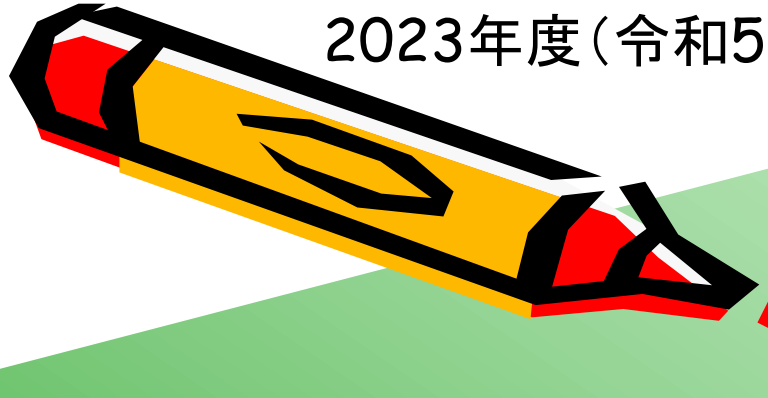



With **parallelism**

- For a block size of **four words** with **interleaved memory (4 banks)**
  - **1** cycle to send 1st address
  - **25 + 3 = 28** cycles to read DRAM
  - **1** cycle to return last data word
  - **30 total clock cycles** miss penalty




- Number of bytes transferred per clock cycle (bandwidth) for a single miss
  - **(4 x 4) / 30 = 0.533 bytes per clock**



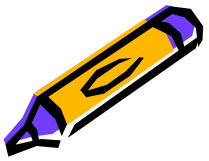
2023年度（令和5年）版

Ver. 2023-10-30a

Course number: CSC.T363

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

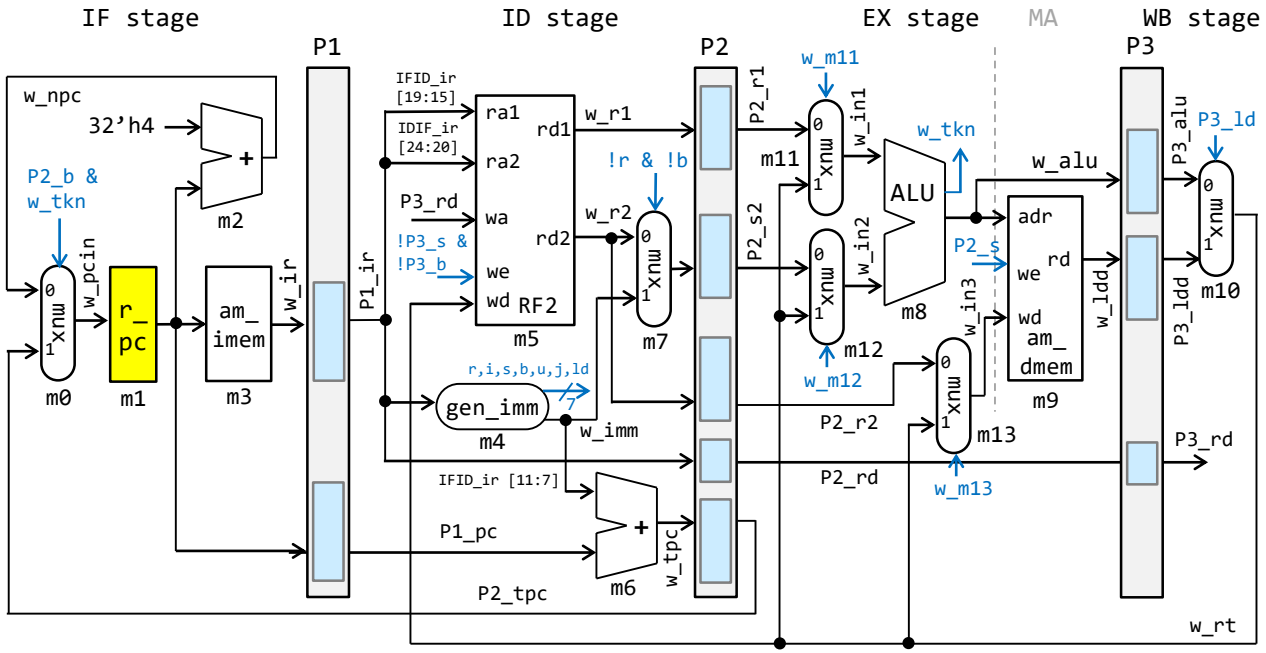
## 8. 分岐予測
## Branch Prediction

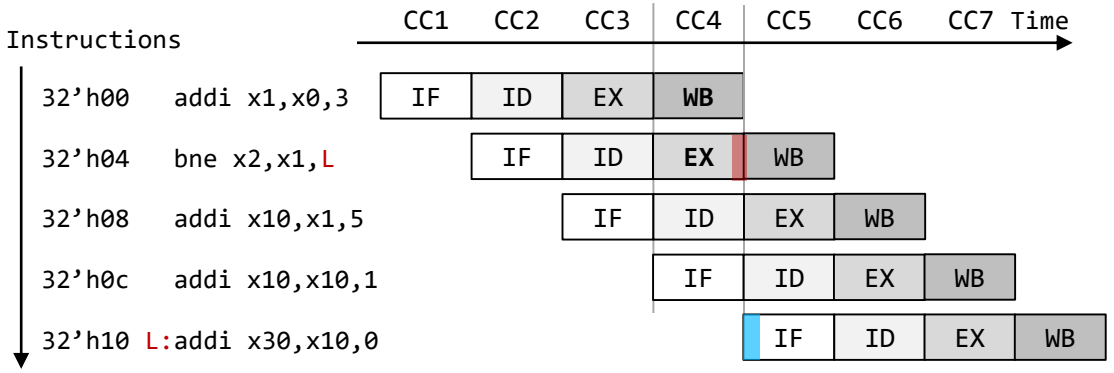


www.arch.cs.titech.ac.jp/lecture/CA/
Tue 13:30-15:10, 15:25-17:05
Fri 13:30-15:10

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

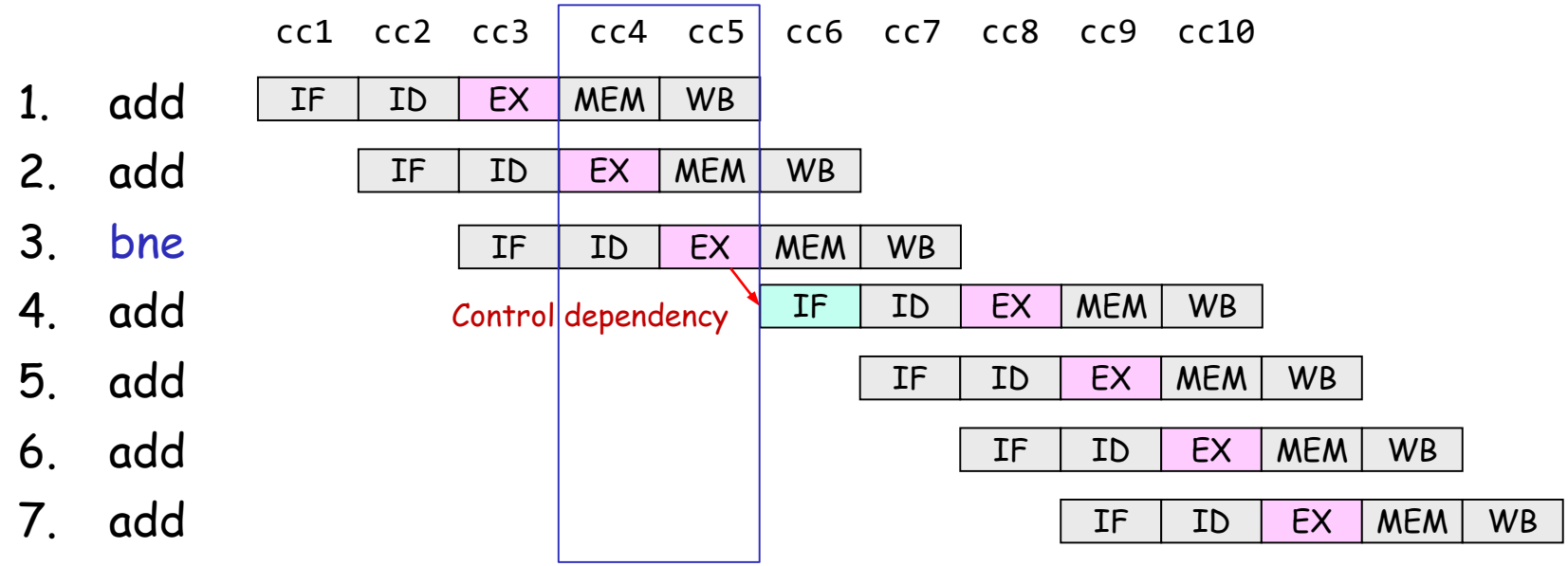
# proc8: 4-stage pipelining processor



# Why do branch instructions degrade IPC?

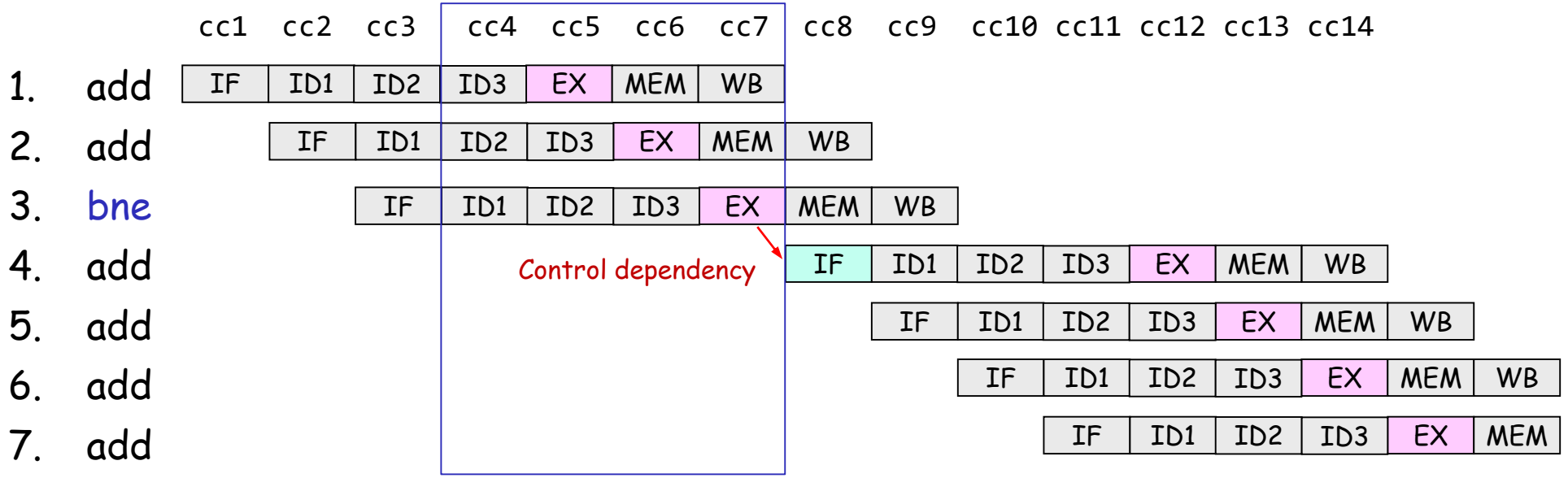
- The branch taken / untaken is determined in the execution stage of the branch.
- The conservative approach of stalling instruction fetch until the branch direction is determined.




five stages pipelined processor executing instruction sequence with a branch

# Deeper pipeline

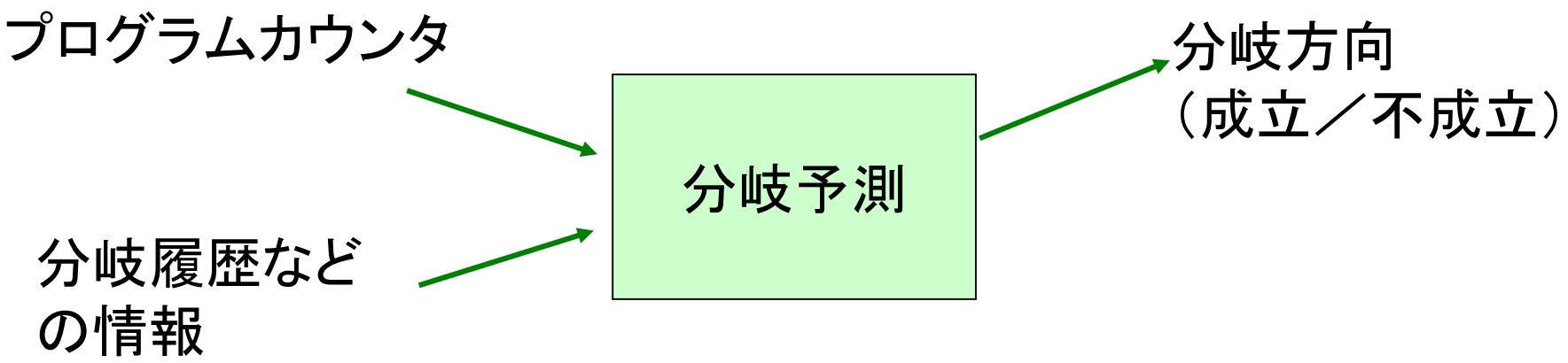
- In conservative approach, IPC degradation will be significant by deeper pipeline



# 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？
  - 分岐先アドレス(32bit)
  - 分岐方向
    - 分岐成立(Taken)か分岐不成立(Not taken, Untaken)か？
- 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)



# bne命令と即値

| 31 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 | |
|---|---|---|---|---|---|---|
| imm[12],imm[10:5] | rs2 | rs1 | funct3 | imm[4:1],imm[11] | opcode | B-type |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| imm[12],imm[10:5] | rs2 | rs1 | 000 | imm[4:1],imm[11] | 1100011 | B-type | **beq** |
| imm[12],imm[10:5] | rs2 | rs1 | 001 | imm[4:1],imm[11] | 1100011 | B-type | **bne** |
| imm[12],imm[10:5] | rs2 | rs1 | 100 | imm[4:1],imm[11] | 1100011 | B-type | **blt** |
| imm[12],imm[10:5] | rs2 | rs1 | 101 | imm[4:1],imm[11] | 1100011 | B-type | **bge** |
| imm[12],imm[10:5] | rs2 | rs1 | 110 | imm[4:1],imm[11] | 1100011 | B-type | **bltu** |
| imm[12],imm[10:5] | rs2 | rs1 | 111 | imm[4:1],imm[11] | 1100011 | B-type | **bgeu** |

| imm[12:2] | imm[12:1] | imm[12:1] | {imm[12],imm[10:5],imm[4:1],imm[11]} |
|---|---|---|---|
| -10 | -20 | 12'b111111101100 | 12'b1111110_11001 |
| -9 | -18 | 12'b111111101110 | 12'b1111110_11101 |
| -8 | -16 | 12'b111111110000 | 12'b1111111_00001 |
| -7 | -14 | 12'b111111110010 | 12'b1111111_00101 |
| -6 | -12 | 12'b111111110100 | 12'b1111111_01001 |
| -5 | -10 | 12'b111111110110 | 12'b1111111_01101 |
| -4 | -8 | 12'b111111111000 | 12'b1111111_10001 |
| -3 | -6 | 12'b111111111010 | 12'b1111111_10101 |
| -2 | -4 | 12'b111111111100 | 12'b1111111_11001 |
| -1 | -2 | 12'b111111111110 | 12'b1111111_11101 |
| 0 | 0 | 12'b000000000000 | 12'b0000000_00000 |
| 1 | 2 | 12'b000000000010 | 12'b0000000_00100 |
| 2 | 4 | 12'b000000000100 | 12'b0000000_01000 |
| 3 | 6 | 12'b000000000110 | 12'b0000000_01100 |
| 4 | 8 | 12'b000000001000 | 12'b0000000_10000 |

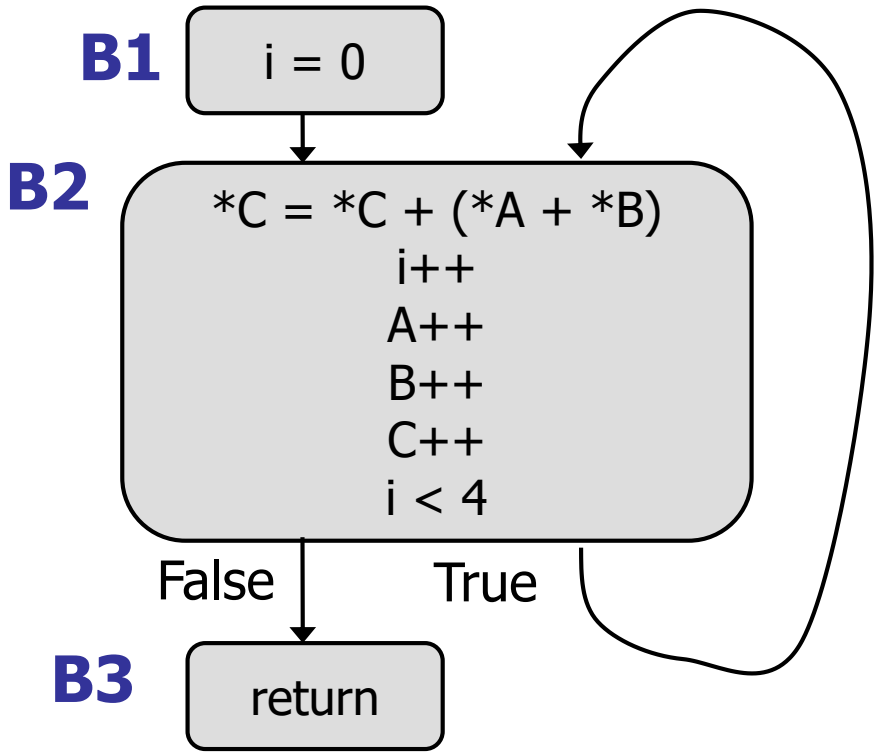
# サンプルプログラム

```
1  `MM[0]={12'd0,5'd0,3'h0,5'd10,7'h13};          // 00   addi x10,x0,0
2  `MM[1]={12'd0,5'd0,3'h0,5'd3,7'h13};           // 04   addi x3,x0,0
3  `MM[2]={12'd101,5'd0,3'h0,5'd1,7'h13};         // 08   addi x1,x0,101
4  `MM[3]={7'd0,5'd3,5'd10,3'h0,5'd10,7'h33};     // 0c L:add  x10,x10,x3
5  `MM[4]={12'd1,5'd3,3'h0,5'd3,7'h13};           // 10   addi x3,x3,1
6  `MM[5]={~12'd0,5'd1,5'd3,3'h1,5'b11001,7'h63};// 14   bne  x3,x1,L
7  `MM[6]=32'h00050f13;                           // 18   HALT
```

```
1  `MM[0]={12'd0,5'd0,3'h0,5'd10,7'h13};          // 00   addi x10,x0,0
2  `MM[1]={12'd0,5'd0,3'h0,5'd3,7'h13};           // 04   addi x3,x0,0
3  `MM[2]={12'd101,5'd0,3'h0,5'd1,7'h13};         // 08   addi x1,x0,101
4  `MM[3]={7'd0,5'd3,5'd10,3'h0,5'd10,7'h33};     // 0c L:add  x10,x10,x3
5  `MM[4]={~12'd0,5'd0,5'd0,3'h1,5'b11101,7'h63};// 10   bne  x0,x0,L
6  `MM[5]={12'd1,5'd3,3'h0,5'd3,7'h13};           // 14   addi x3,x3,1
7  `MM[6]={~12'd0,5'd1,5'd3,3'h1,5'b10001,7'h63};// 18   bne  x3,x1,L
8  `MM[7]=32'h00050f13;                           // 1c   HALT
```

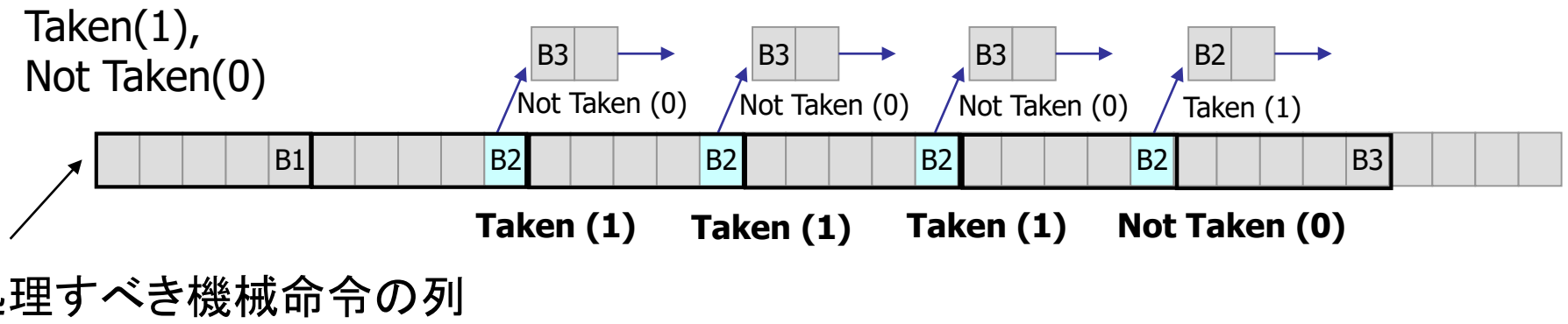
# サンプルプログラム Vector Add

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



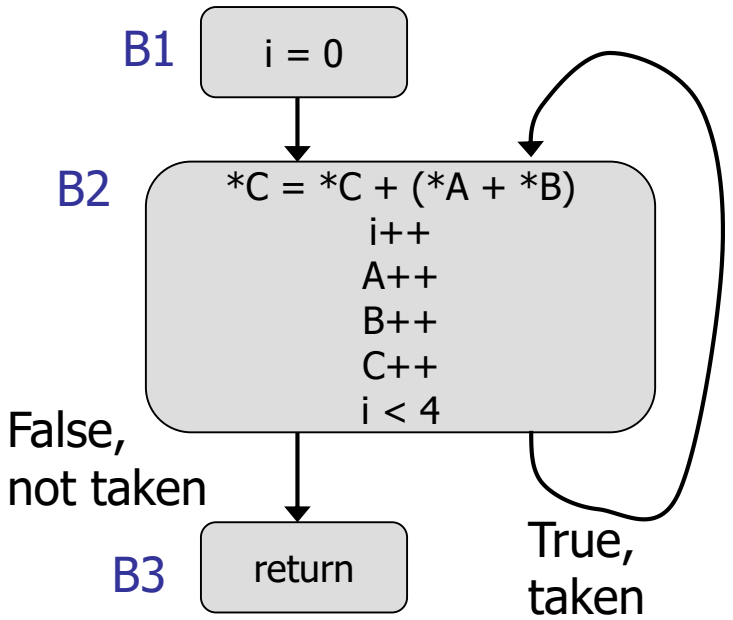

制御フローグラフ

# シンプルな分岐予測: Branch Always




処理すべき機械命令の列

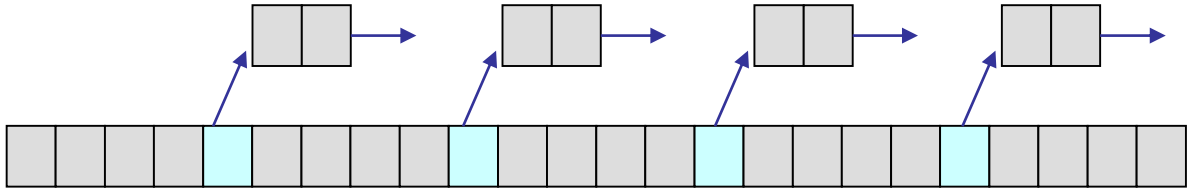
- Branch Always: 常に分岐が成立すると予測する.
  - 上の例では，予測成功率は75%，ミス率25%
  - 予測のためのメモリを必要としない.
  - 予測とよぶほどのものではない.



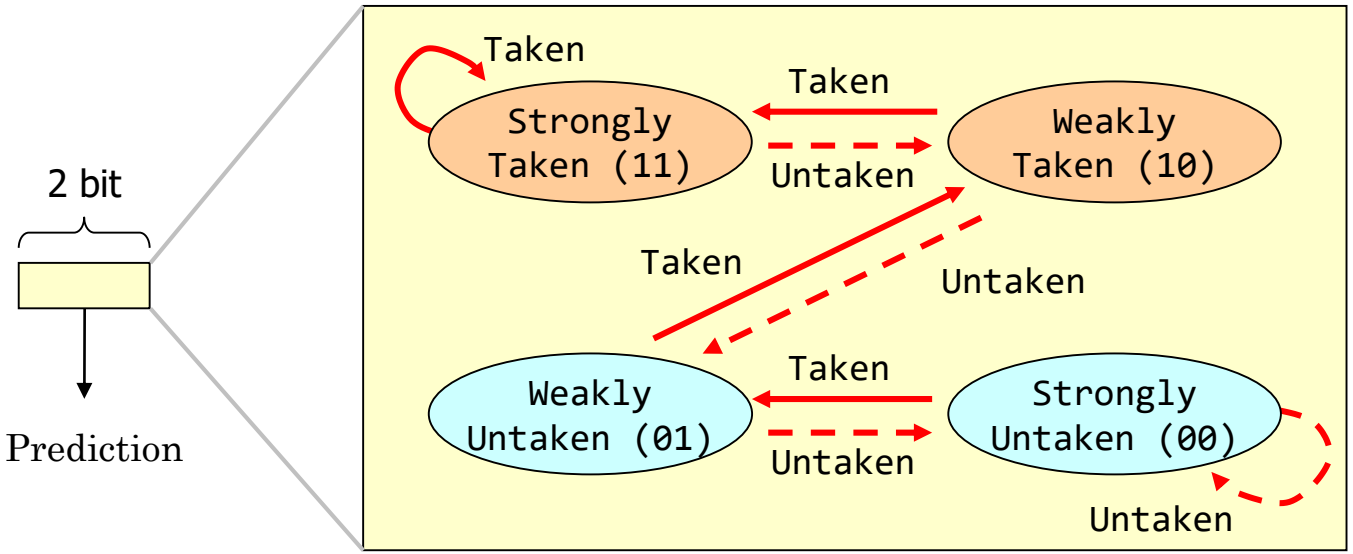
# シンプルな分岐予測: 2ビットカウンタ方式

トレースデータ
（分岐アドレス，分岐結果）

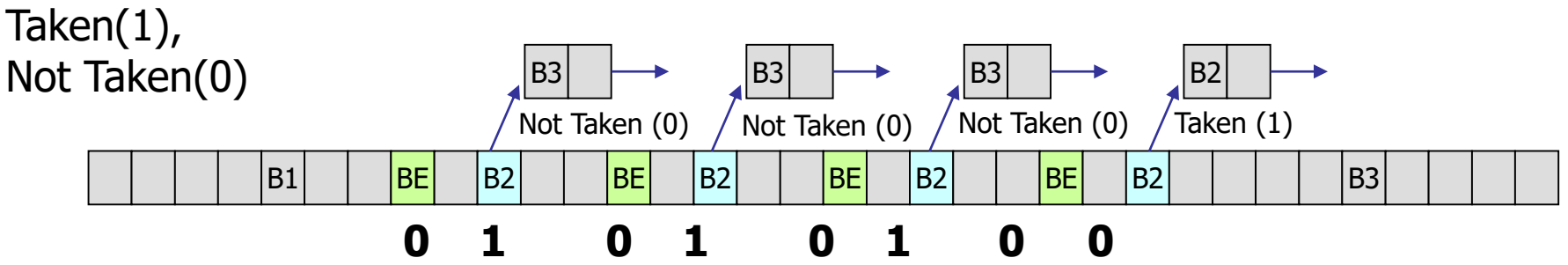
| | |
|---|---|
| 0040d6c5 1 | 0040d89c 1 |
| 0040d6b8 1 | 0040d89c 1 |
| 0040d6bc 0 | 0040d89c 1 |
| 0040d6c5 0 | 0040d89c 1 |
| 0040d6df 0 | 0040d89c 1 |
| 0040d71f 0 | 0040d89c 1 |
| 0040d736 0 | 0040d89c 0 |
| 0040d7ab 0 | 0040d8a2 0 |
| 0040d7cd 0 | 0040d8c0 1 |
| 0040d7f9 0 | 0040d8c4 0 |
| 0040d81e 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 |



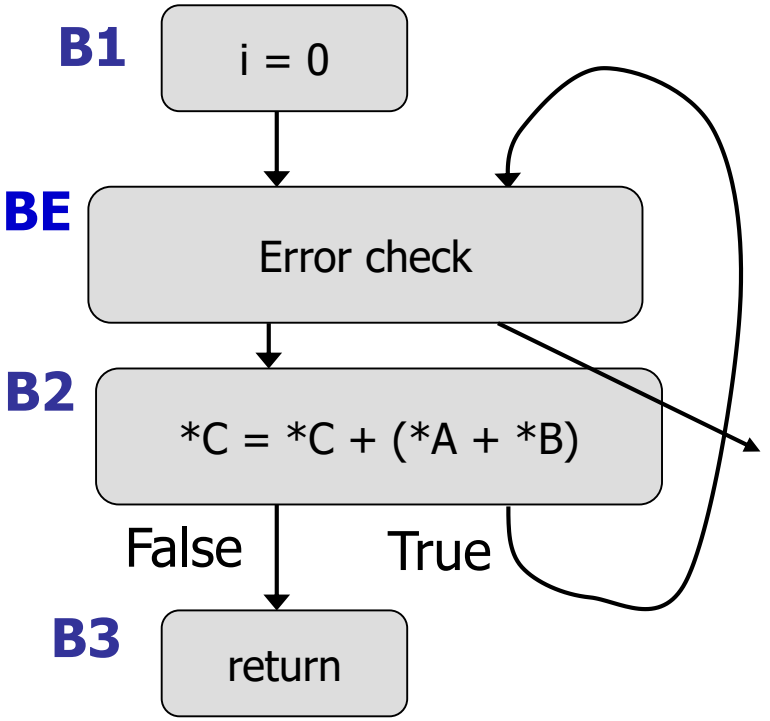
- 2ビットカウンタ方式
  - 大域的な偏り，局所性の利用
  - 2ビットカウンタの状態に応じて予測
  - 予測のためのメモリは2ビット
  - 状態の更新



# サンプルプログラム Vector Add

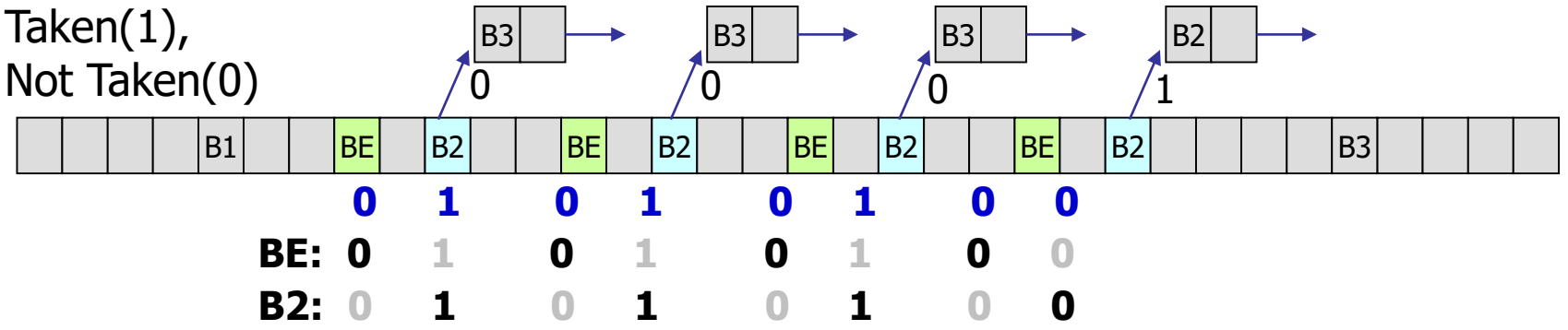



```
#define VSIZE 4
void vadd(long *A, long *B, long *C){
  for(i=0; i<VSIZE; i++) {
    if(A[i]<0) error_routine();
    C[i] += (A[i] + B[i]);
  }
}
```

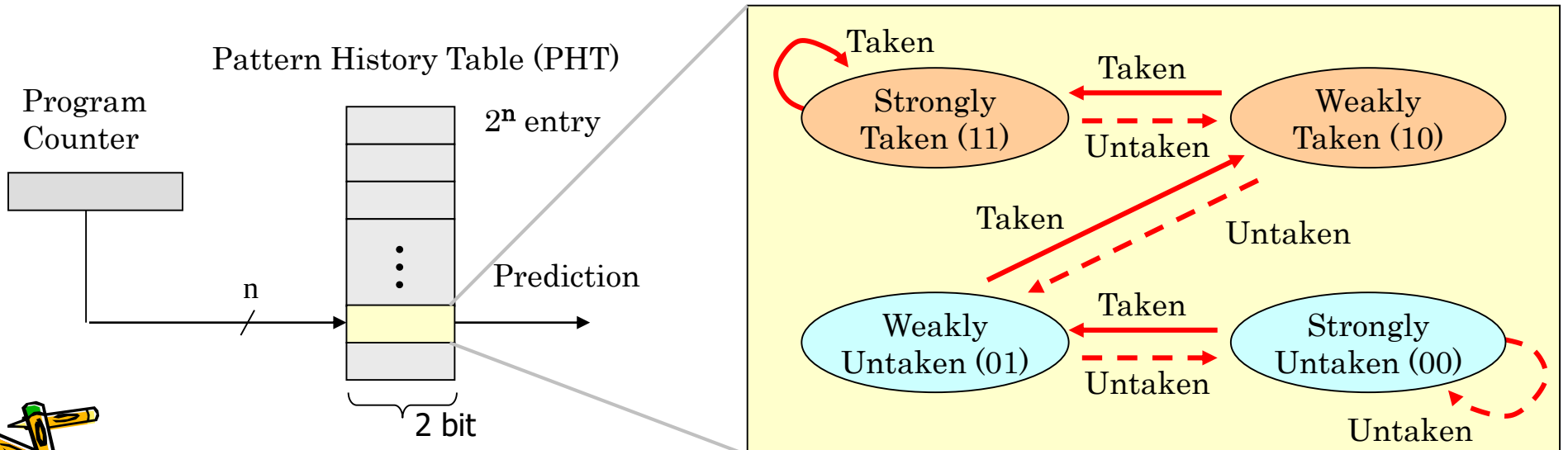



制御フローグラフ

# Bimodal branch predictor (ISCA 1981)

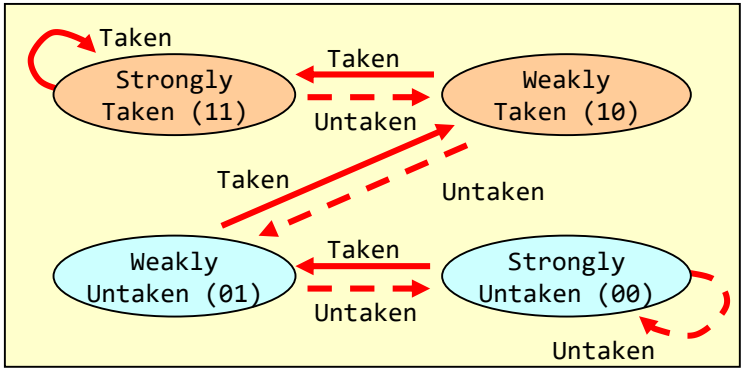



- 分岐アドレス（プログラムカウンタ）毎に履歴を切り替える
  - 分岐アドレスによりパターン履歴表（PHT）のインデックスを作成
- パターン履歴表は2ビットカウンタの配列.



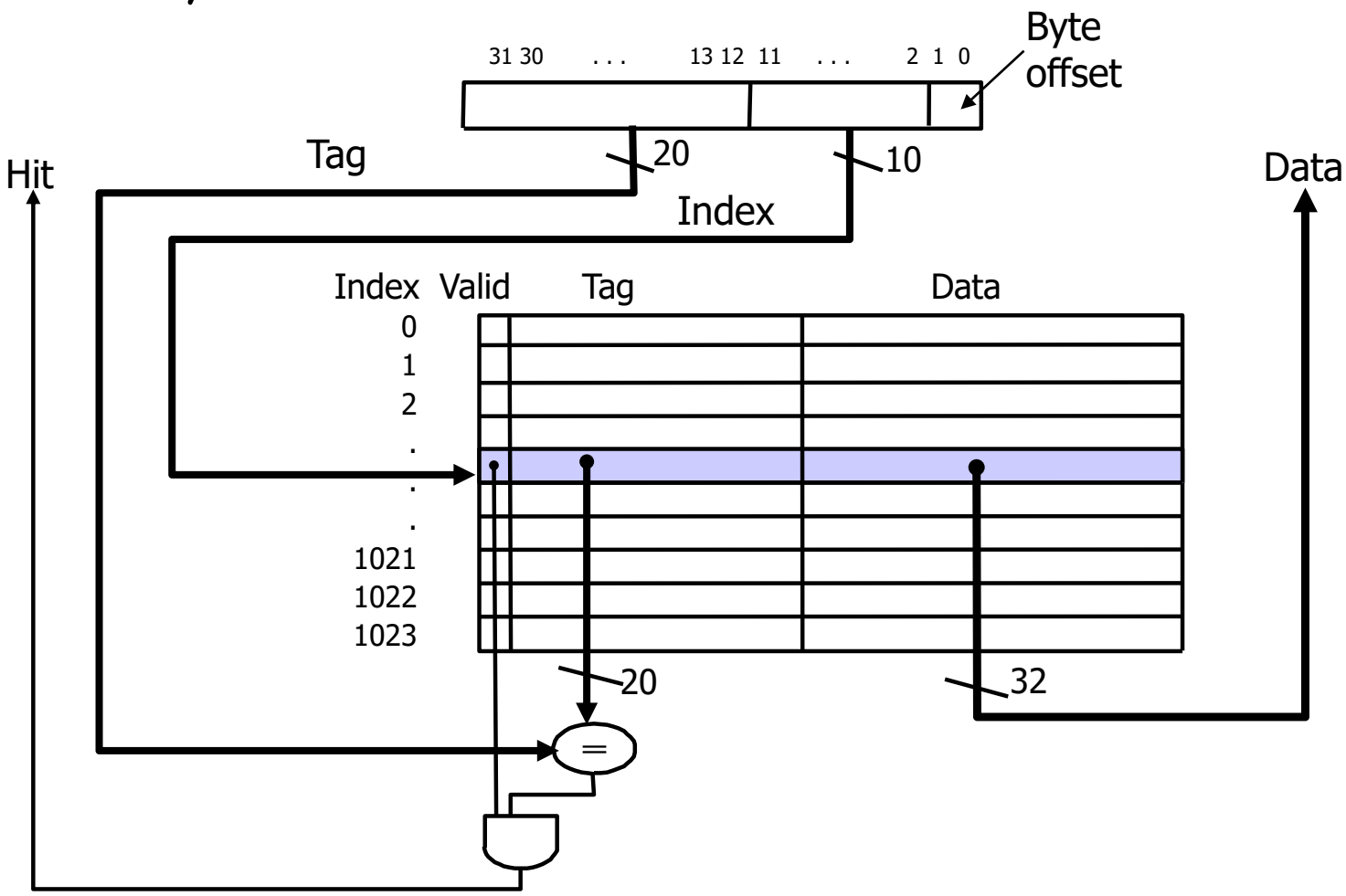
# Bimodal branch predictor (ISCA 1981)

```
 1  module m_bimodal(w_clk, w_radr, w_pred, w_wadr, w_we, w_tkn);
 2    input  wire w_clk, w_we;
 3    input  wire [4:0] w_wadr, w_radr;
 4    input  wire w_tkn;
 5    output wire w_pred;
 6    reg [1:0] mem [0:31];
 7    wire [1:0] w_data = mem[w_radr];
 8    assign w_pred = w_data[1];
 9    wire [1:0] w_cnt  = mem[w_wadr];
10    always @(posedge w_clk) if (w_we)
11      mem[w_wadr] <= (w_cnt < 3 &  w_tkn) ? w_cnt + 1 :
12                     (w_cnt > 0 & !w_tkn) ? w_cnt - 1 : w_cnt;
13    integer i; initial for (i=0; i<32; i=i+1) mem[i] = 1;
14  endmodule
```



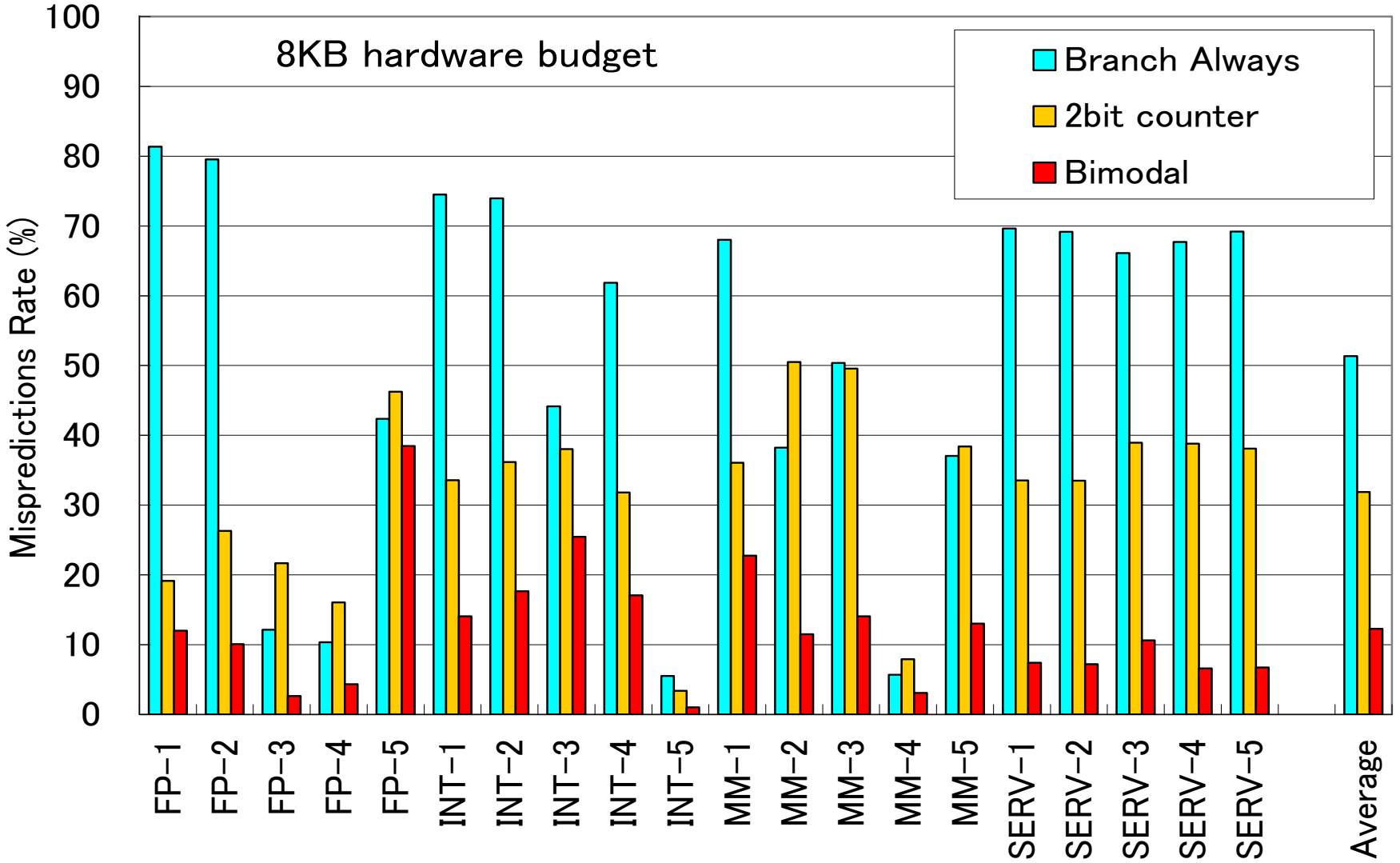
# MIPS Direct Mapped Cache Example

- 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)



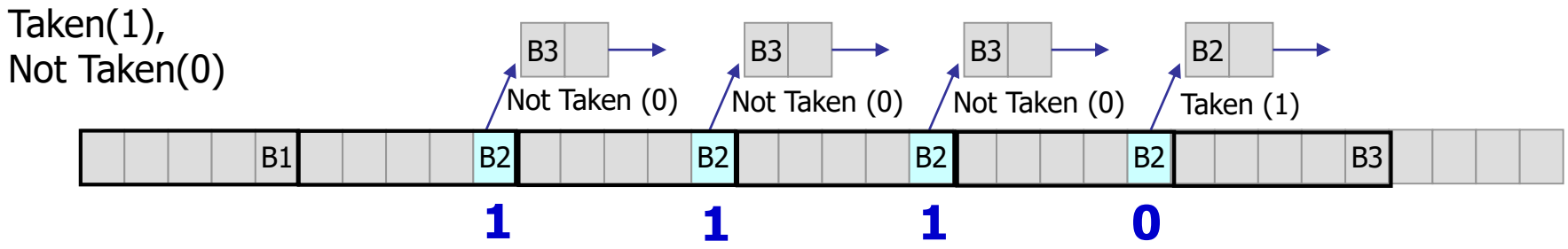

Benchmark for CBP(2004) by Intel MRL and IEEE TC uARCH.

# サンプルプログラム

```
1  `MM[0]={12'd0,5'd0,3'h0,5'd10,7'h13};          // 00   addi x10,x0,0
2  `MM[1]={12'd0,5'd0,3'h0,5'd3,7'h13};           // 04   addi x3,x0,0
3  `MM[2]={12'd101,5'd0,3'h0,5'd1,7'h13};         // 08   addi x1,x0,101
4  `MM[3]={7'd0,5'd3,5'd10,3'h0,5'd10,7'h33};     // 0c L:add  x10,x10,x3
5  `MM[4]={12'd1,5'd3,3'h0,5'd3,7'h13};           // 10   addi x3,x3,1
6  `MM[5]={~12'd0,5'd1,5'd3,3'h1,5'b11001,7'h63};// 14   bne  x3,x1,L
7  `MM[6]=32'h00050f13;                           // 18   HALT
```

```
 1  `MM[0] ={12'd4,  5'd0, 3'h0,5'd1, 7'h13};      // 00   addi x1,x0,4
 2  `MM[1] ={12'd101,5'd0, 3'h0,5'd2, 7'h13};      // 04   addi x2,x0,101
 3  `MM[2] ={12'd0,  5'd0, 3'h0,5'd10,7'h13};      // 08   addi x10,x0,0
 4  `MM[3] ={12'd0,  5'd0, 3'h0,5'd4 ,7'h13};      // 0c   addi x4,x0,0
 5  `MM[4] ={12'd0,  5'd0, 3'h0,5'd3 ,7'h13};      // 10 M:addi x3,x0,0
 6  `MM[5] ={12'd1,  5'd3 ,3'h0,5'd3, 7'h13};      // 14 L:addi x3,x3,1
 7  `MM[6] ={12'd1,  5'd10,3'h0,5'd10,7'h13};      // 18   addi x10,x10,1
 8  `MM[7] ={~12'd0,5'd1,5'd3,3'h1,5'b11001,7'h63};// 1c   bne  x3,x1,L
 9  `MM[8] ={12'd1,  5'd4 ,3'h0,5'd4, 7'h13};      // 20   addi x4,x4,1
10  `MM[9] ={12'd1,  5'd10,3'h0,5'd10,7'h13};      // 24   addi x10,x10,1
11  `MM[10]={~12'd0,5'd2,5'd4,3'h1,5'b01001,7'h63};// 28   bne  x4,x2,M
12  `MM[11]=32'h00050f13;                          // 2c   HALT
```

# Branch history（分岐履歴）




## B2の分岐履歴

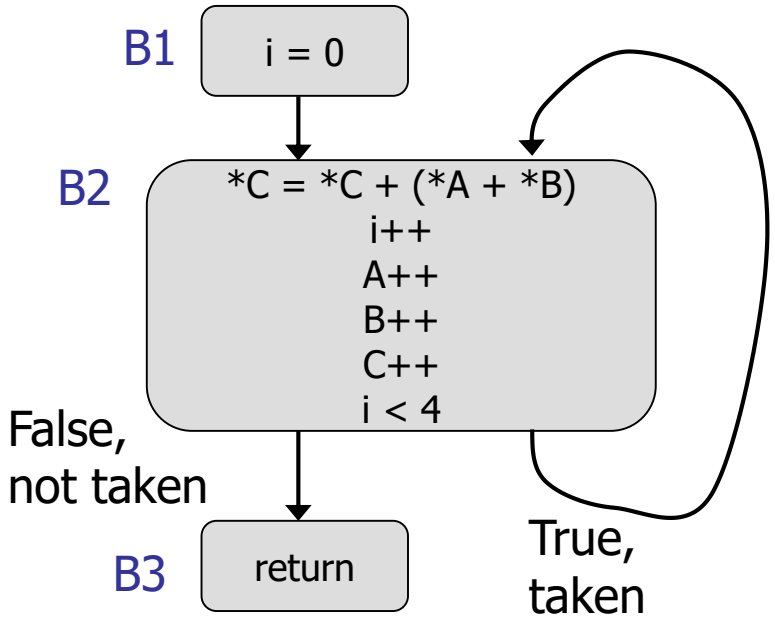
1110 ?

11101 ?

111011 ?

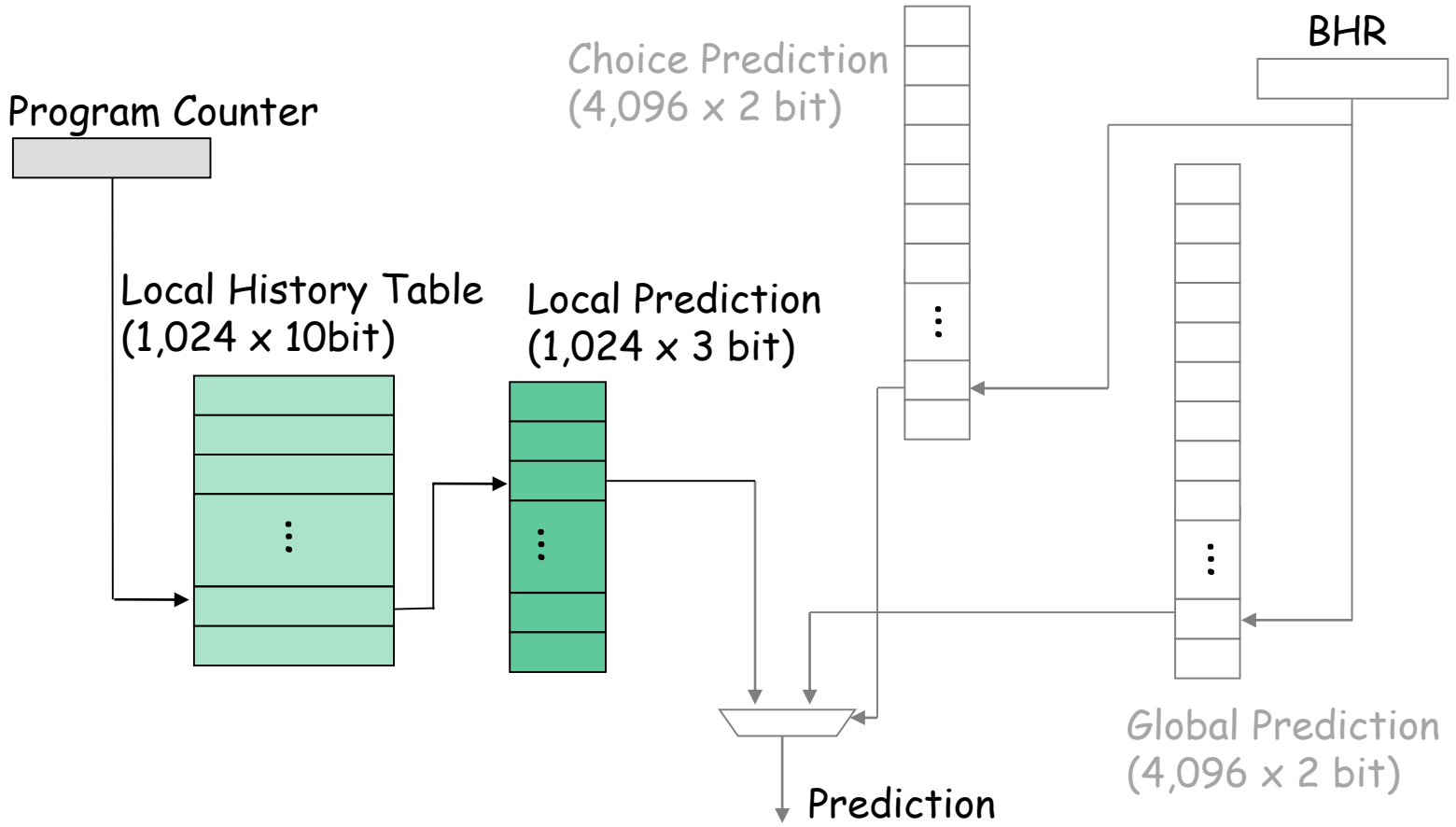
1110111 ?

11101110 ?

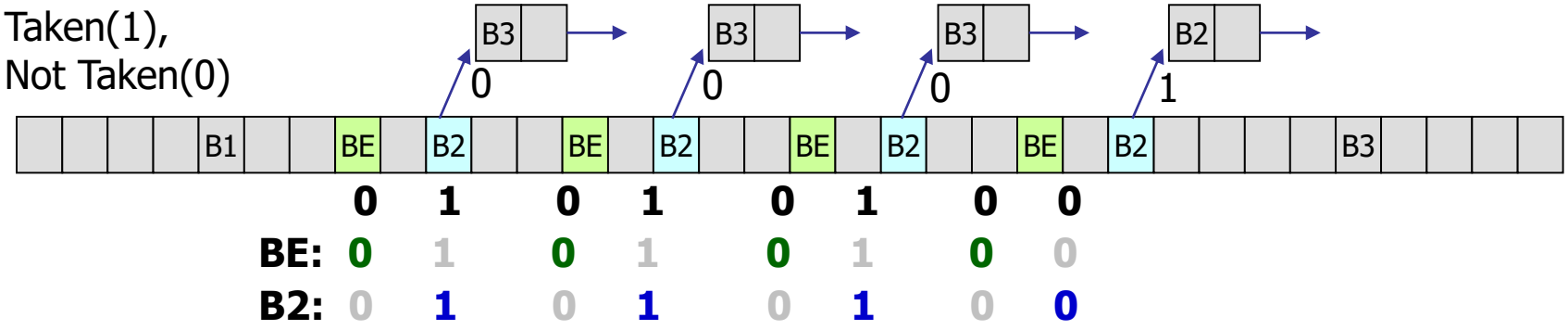


# Alpha 21264's hybrid branch predictor in 1996

- A hybrid of local prediction and global prediction implemented in DEC Alpha 21264 which was the state-of-the art commercial processor.
- A choice predictor is used as a meta-predictor



# 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 ?

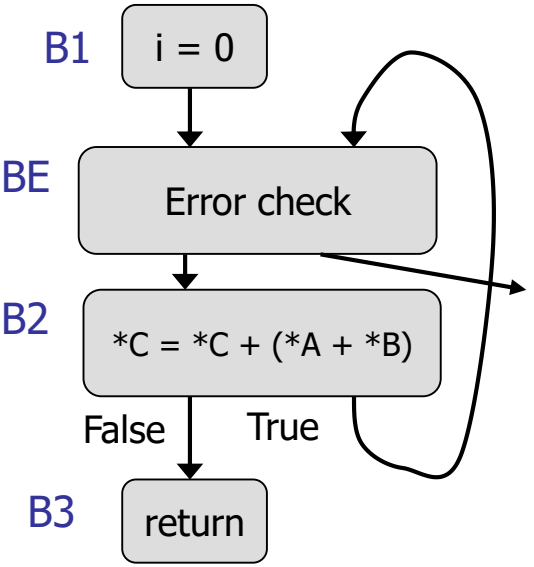
11101110 ?

ローカル分岐履歴

B2とBEの分岐履歴

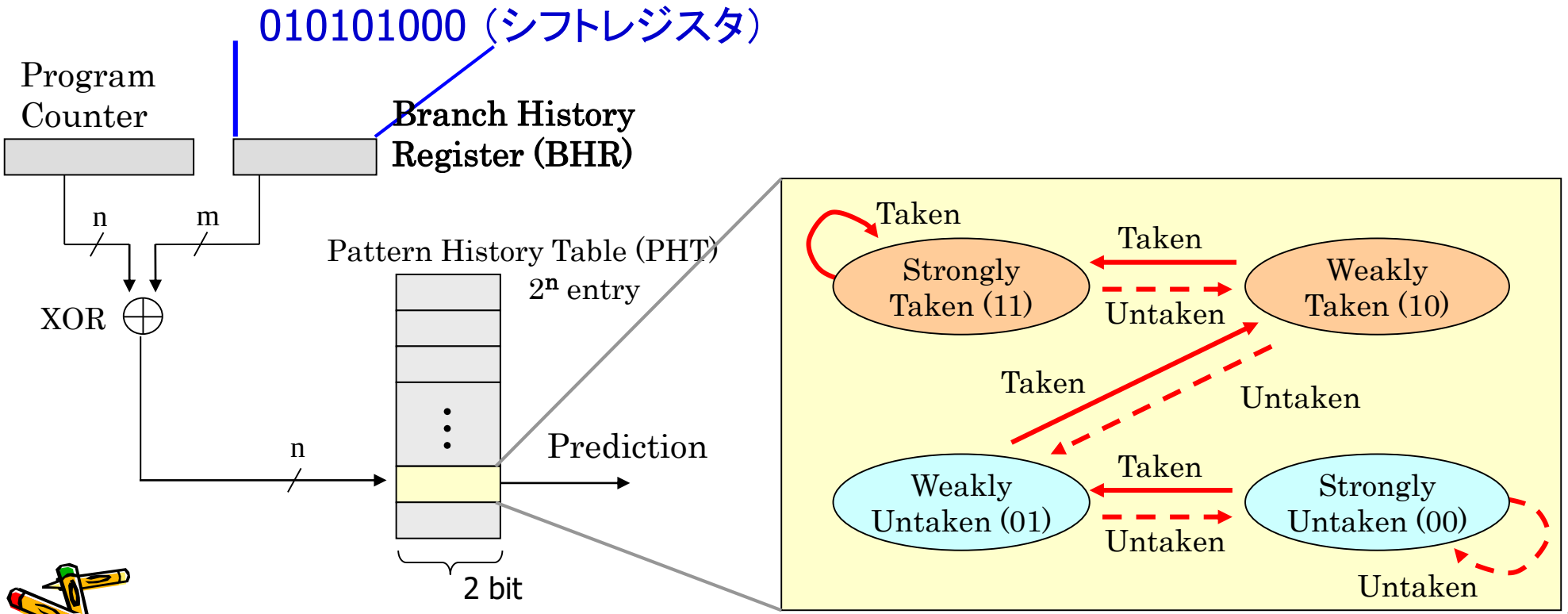
010101000 ?

グローバル分岐履歴



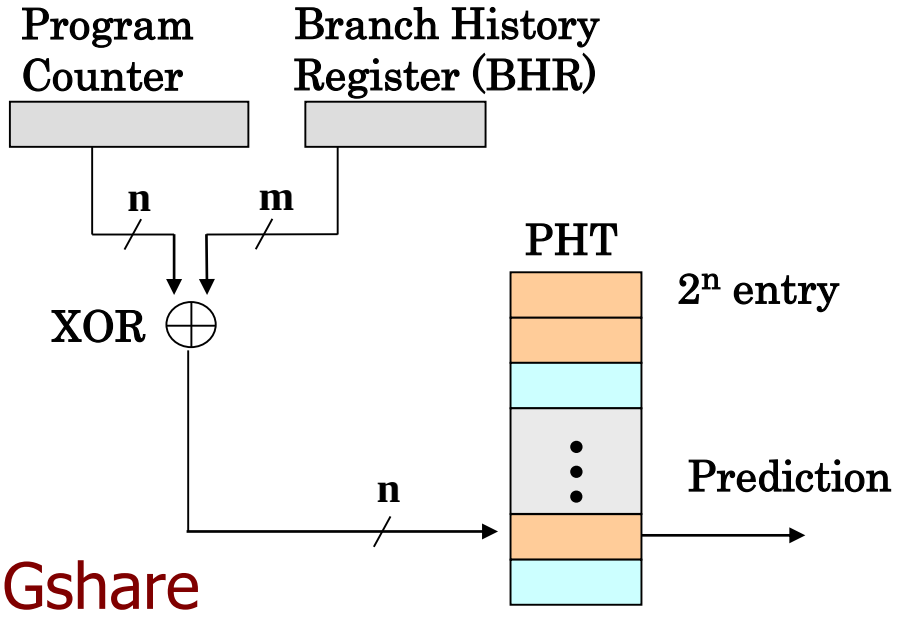
# Gshare (TR-DEC 1993)

- 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

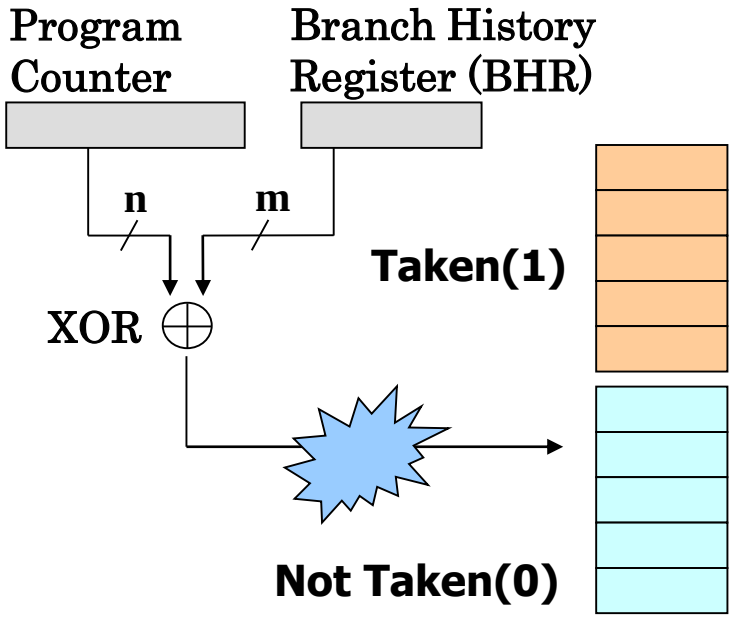


# Gshare の改良

- 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

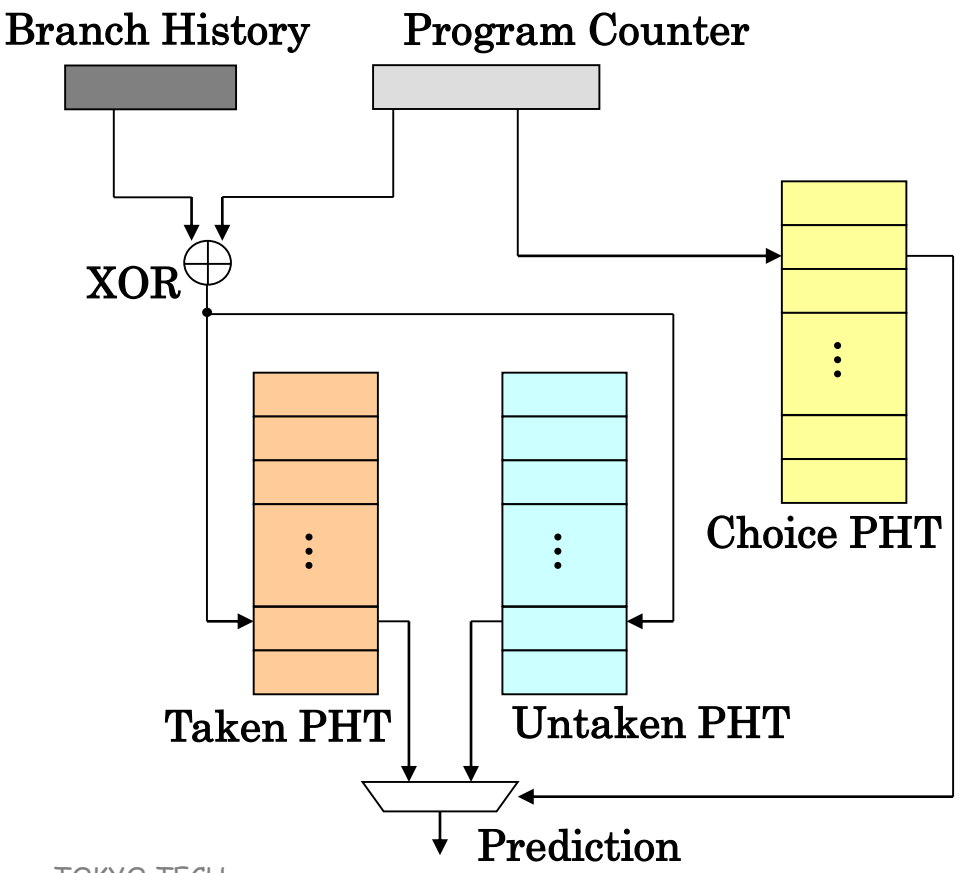



Gshare

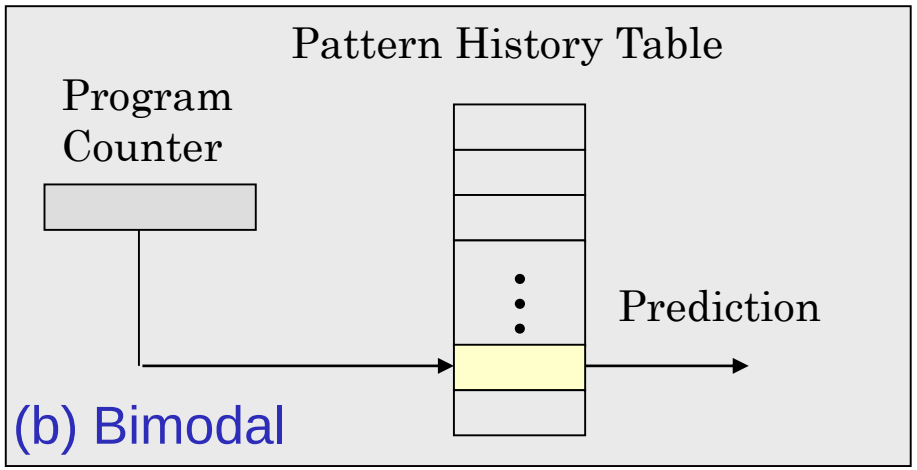


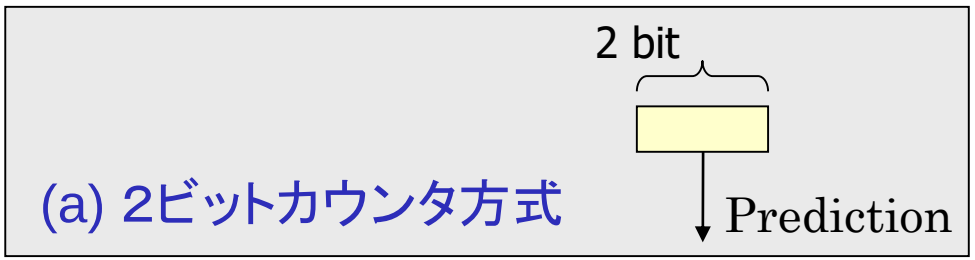
# Bimode (MICRO 1997)

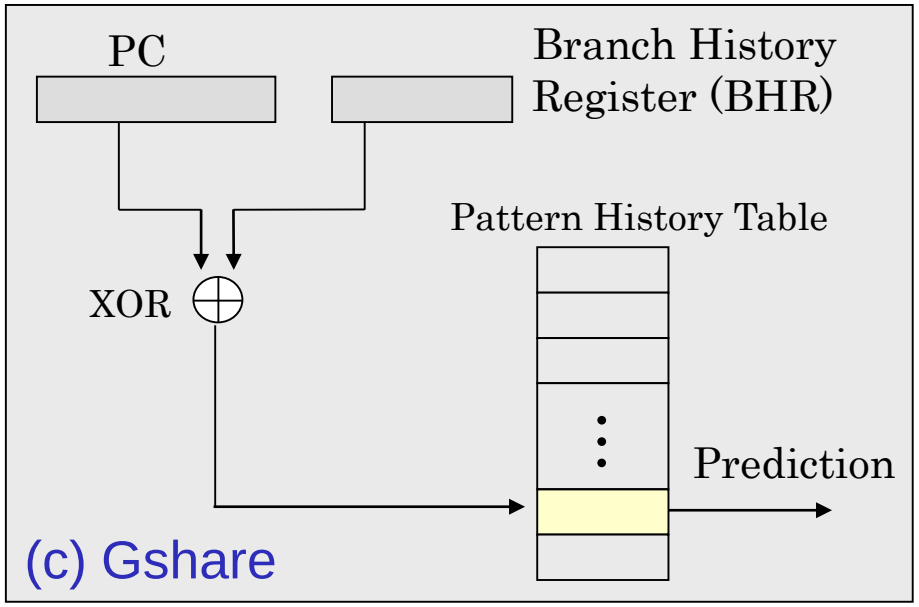
- 偏りを利用して競合の悪影響を緩和
  - 分岐成立に偏っているものをTaken PHTに格納
  - 分岐不成立に偏っているものをUntaken PHTに格納
- Choice PHT の内容で，どちらのテーブルを利用するか選択
- インデックスを工夫
  - Choice PHT は命令アドレス
  - Taken PHT, Untaken PHT は命令アドレスと分岐履歴

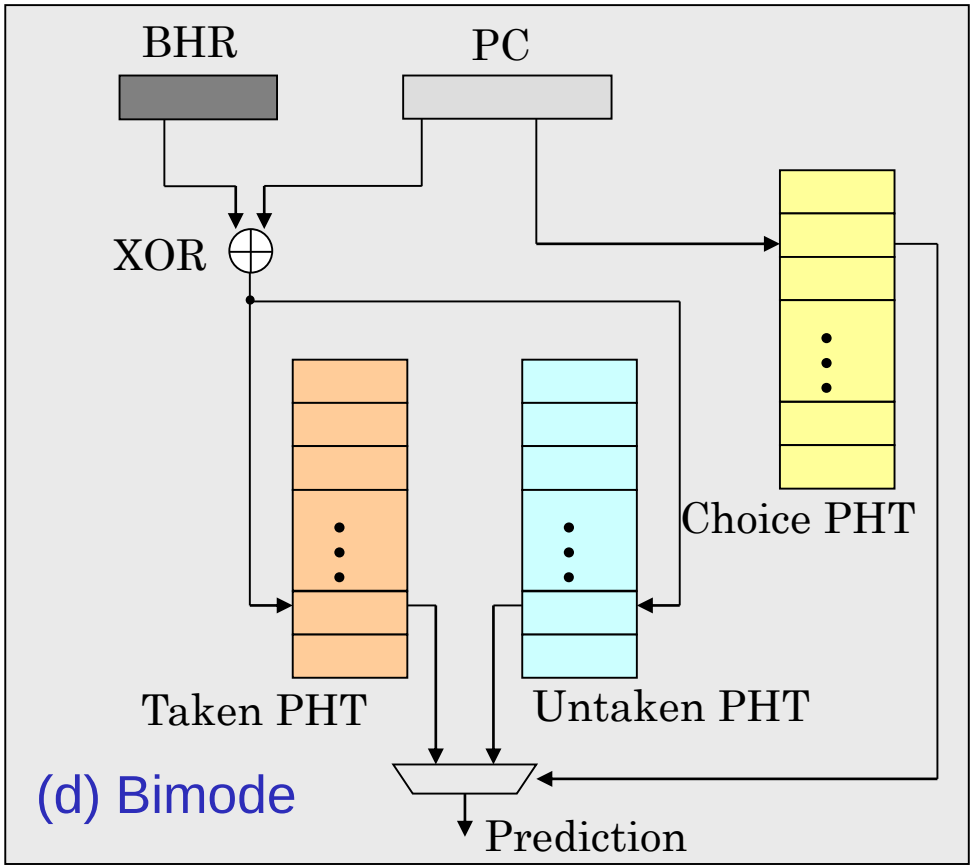


# 幾つかの分岐予測器

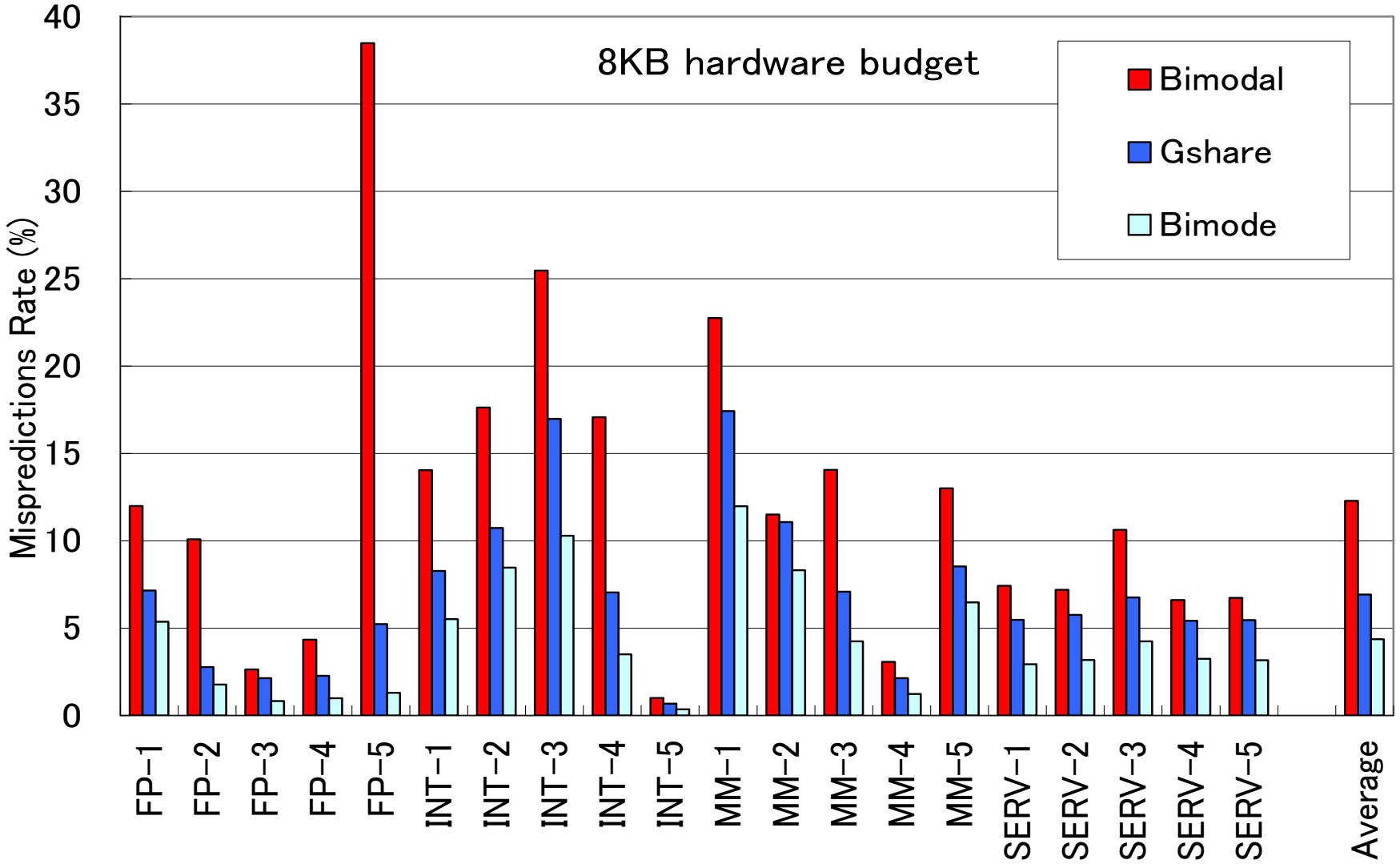

Pattern History Table
Program Counter
Prediction
(b) Bimodal


2 bit
(a) 2ビットカウンタ方式
Prediction


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


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

# 予測精度(予測ミス率)



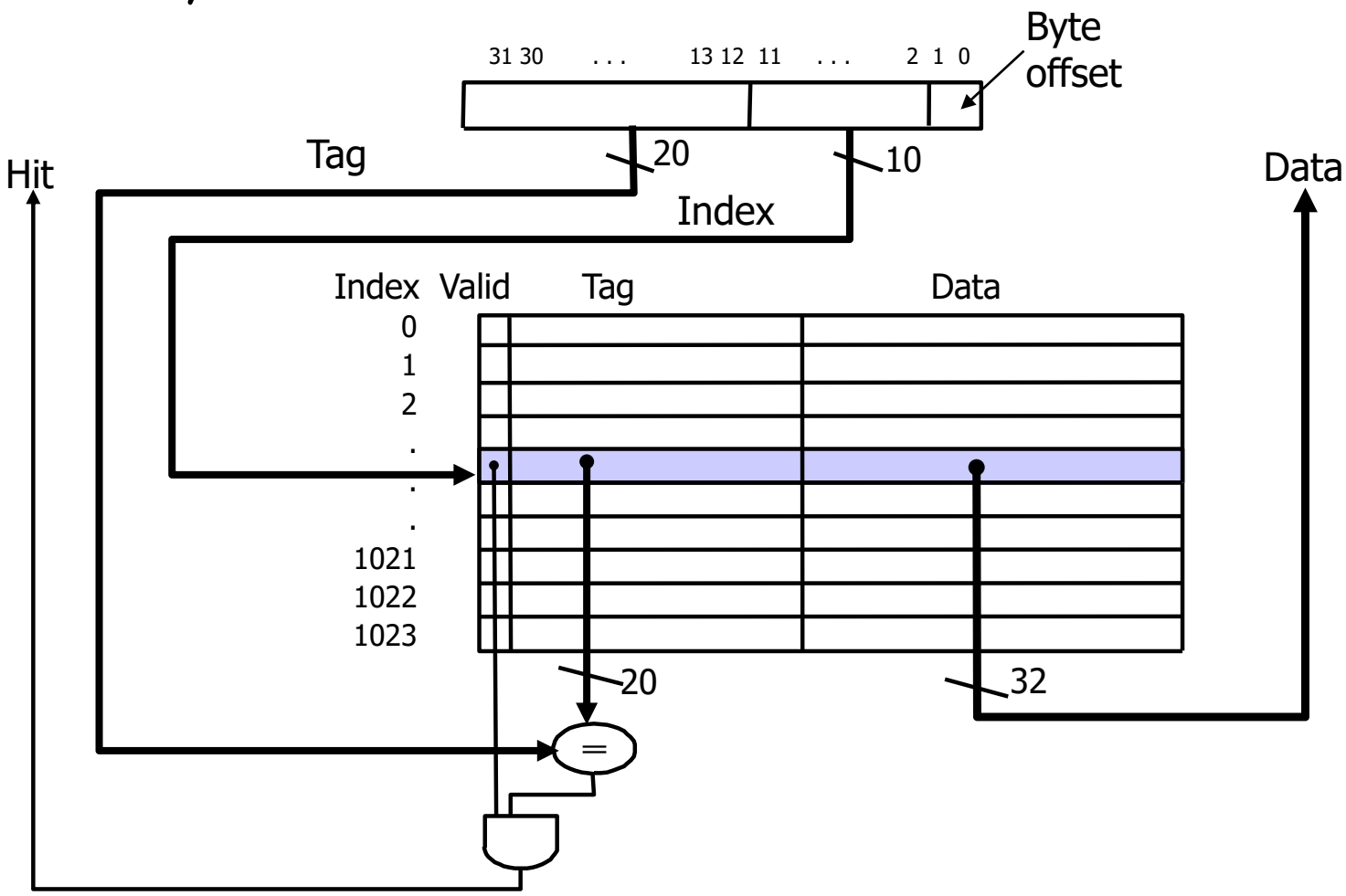

Benchmark for CBP(2004) by Intel MRL and IEEE TC uARCH.

# 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)

# Direct Mapped Cache Example

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




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