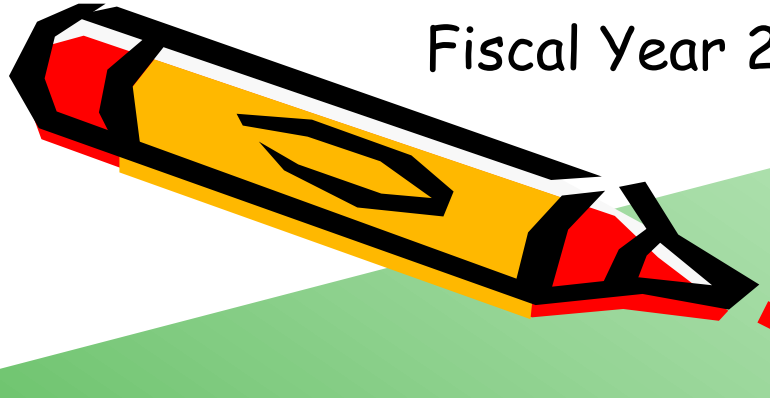
Fiscal Year 2023

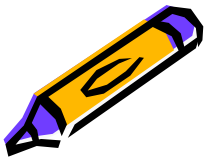
Ver. 2024-01-04a



Course number: CSC.T433
School of Computing,
Graduate major in Computer Science

# Advanced Computer Architecture

## 6. Instruction Level Parallelism: Instruction Fetch and Branch Prediction



www.arch.cs.titech.ac.jp/lecture/ACA/
Room No.W834, Lecture (Face-to-face)
Mon 13:30-15:10, Thr 13:30-15:10

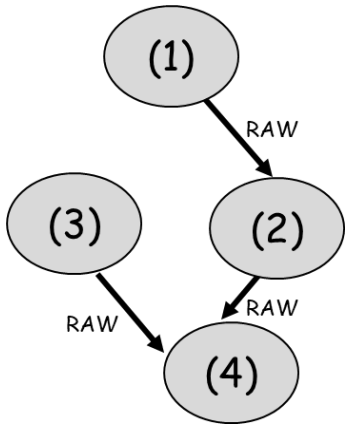
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

# Exploiting Instruction Level Parallelism (ILP)

- A superscalar has to handle some flows efficiently to exploit ILP
  - Control flow (control dependence)
    - To execute ***n*** instructions per clock cycle, the processor has to fetch at least ***n*** instructions per cycle.
    - The main obstacles are branch instruction (BNE)
    - Prediction
    - Another obstacle is instruction cache
  - Register data flow (data dependence)
    - Out-of-order execution
      - Register renaming
      - Dynamic scheduling
  - Memory data flow
    - Out-of-order execution
    - Another obstacle is instruction cache

```
(1) add x5,x1,x2
(2) add x9,x5,x3
(3) lw  x4, 4(x7)
(4) add x8,x9,x4
```

```
(3) lw  x4, 4(x7)
(1) add x5,x1,x2
(2) add x9,x5,x3
(4) add x8,x9,x4
```

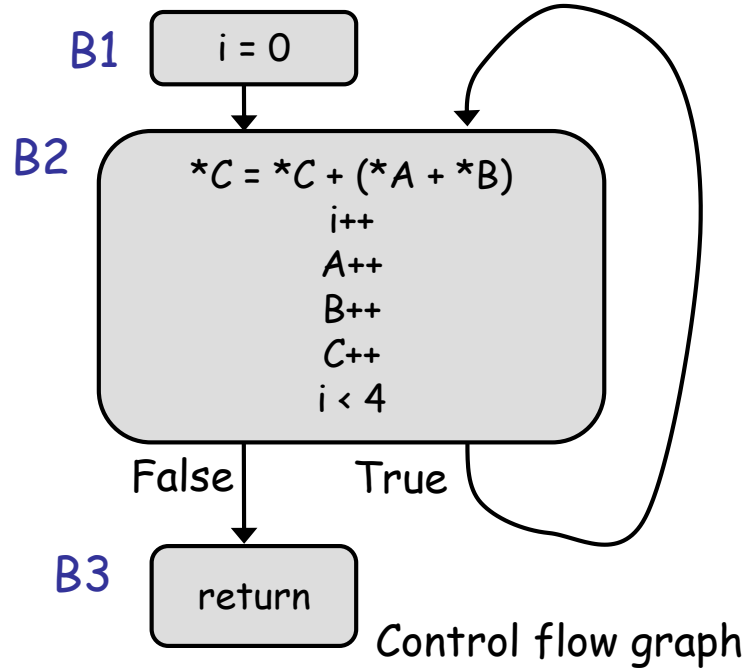


# Branch predictor

- A branch predictor is a digital circuit that tries to guess or predict which way (taken or untaken) a branch will go before this is known definitively.
  - A random predictor will achieve about a 50% hit rate because the prediction output is 1 or 0.
  - Let's guess the accuracy. What is the accuracy of typical branch predictors for high-performance commercial processors?

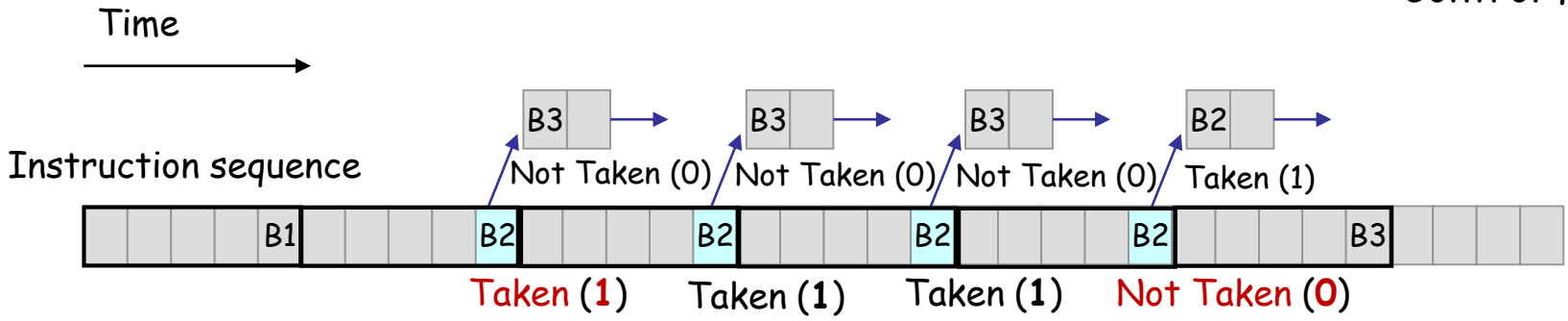
# Sample program: vector add (function v_add)

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



Control flow graph

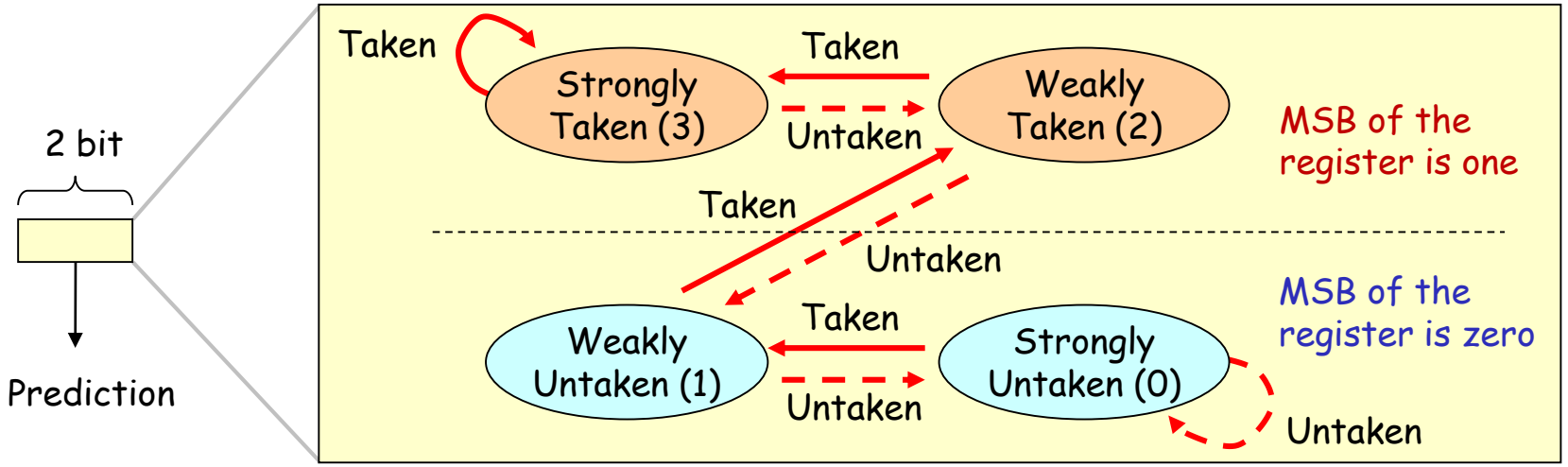
Basic block contains a sequence of statement. The flow of control enters at the beginning of the statement and leave at the end.




Predicting the branch outcome sequence of 1110 1110 1110 1110 1110 …

# Simple branch predictor: 2-bit counter (2BC)

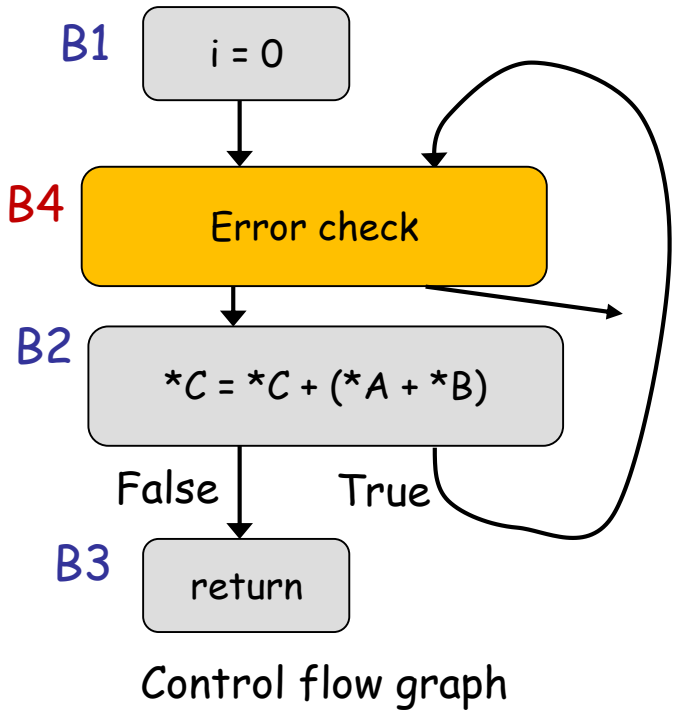
- It uses two bit register as a saturating counter.
- How to update the register
  - If the branch outcome is taken and the value is not 3, then increment the register.
  - If the branch outcome is untaken and the value is not 0, then decrement the register.
- Hot to predict
  - It predicts as 1 if the MSB of the register is one, otherwise predicts as 0.




| | |
|---|---|
| Predicting the sequence of | 1110 1110 1110 1110 1110 ... |
| State of the counter | 2333 2333 2333 2333 2333 ... |
| Prediction | 1111 1111 1111 1111 1111 ... |
| Hit/Miss or the pred. | HHHM HHHM HHHM HHHM HHHM |

# Sample program: vector add with two branches

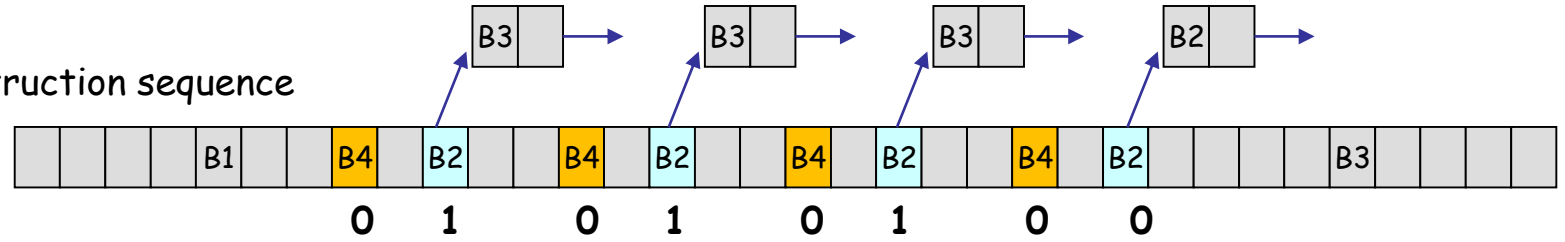
```
#define VSIZE 4
void v_add(int *A, int *B, int *C){
  for(i=0; i<VSIZE; i++) {
    if(A[i]<0) error_routine();
    C[i] += (A[i] + B[i]);
  }
}
```




Control flow graph

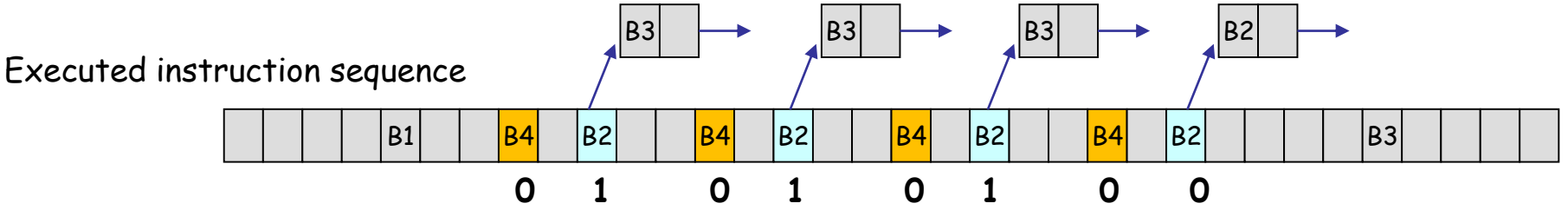
Basic block contains a sequence of statement.
The flow of control enters at the beginning of the statement and leave at the end.

Executed instruction sequence




Predicting the sequence of 01010100 01010100 01010100 ...

# Sample program: vector add with two branches




Predicting the branch outcome sequence of

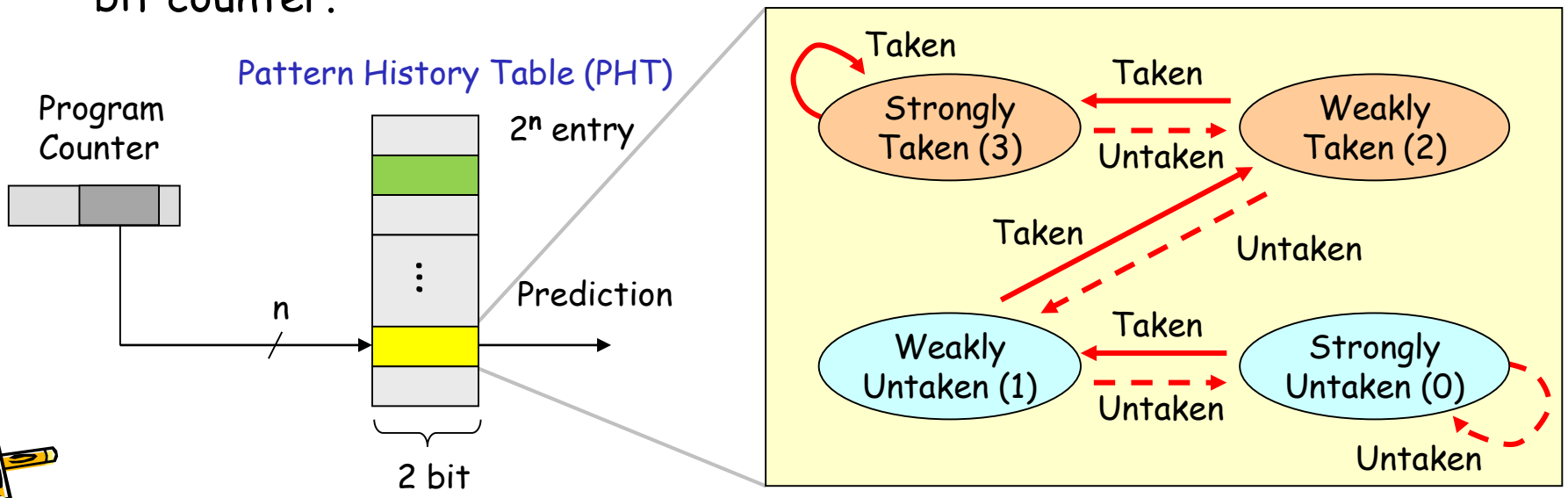
01010100 01010100 01010100 ...

The B4's sequence of 01010100 01010100 01010100 ...

The B2's sequence of 01010100 01010100 01010100 ...

# Simple branch predictor: bimodal

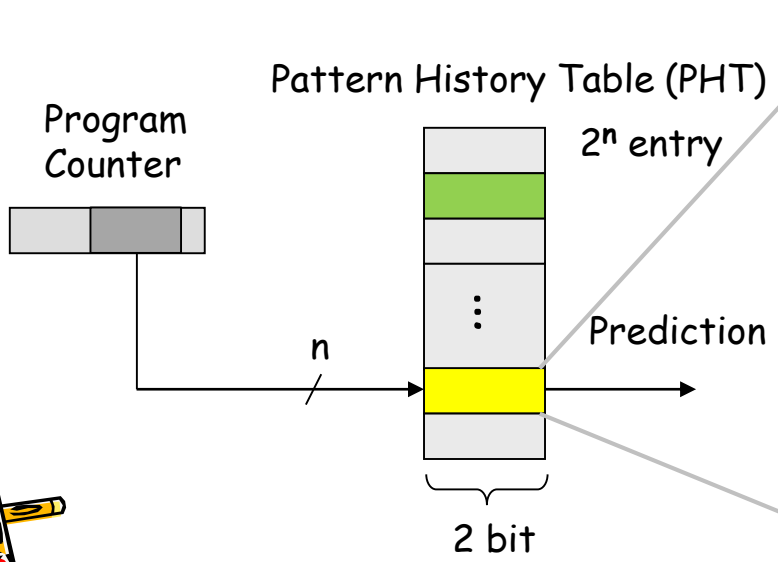
- Program has many static branch instructions. The behavior may depend on each branch. Use plenty of counters (PHT) and assign a counter for a branch instruction.
- How to predict
  - Select a 2-bit counter using PC, and it predicts 1 for taken if the MSB of the register is one; otherwise, it predicts 0 for untaken.
- How to update
  - Select a counter using PC, then update the counter in the same way as 2-bit counter.

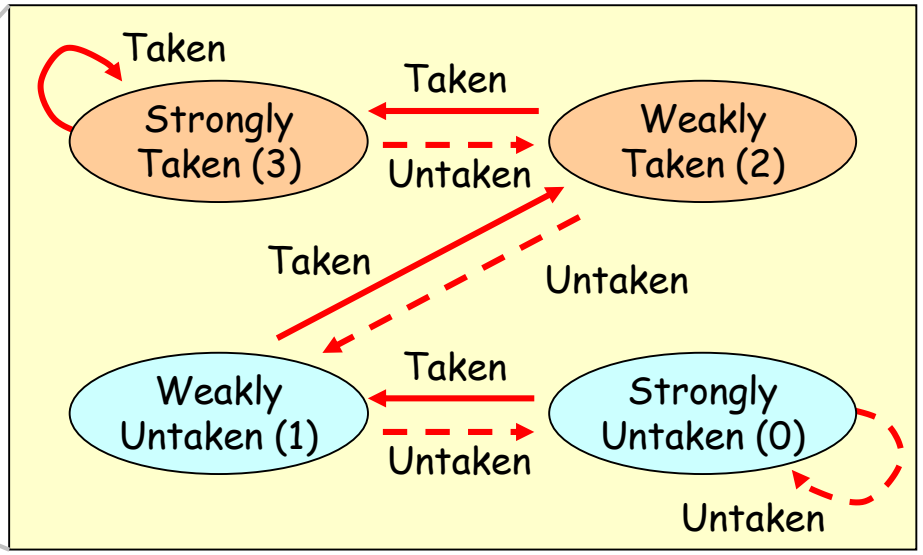


# Simple branch predictor: bimodal

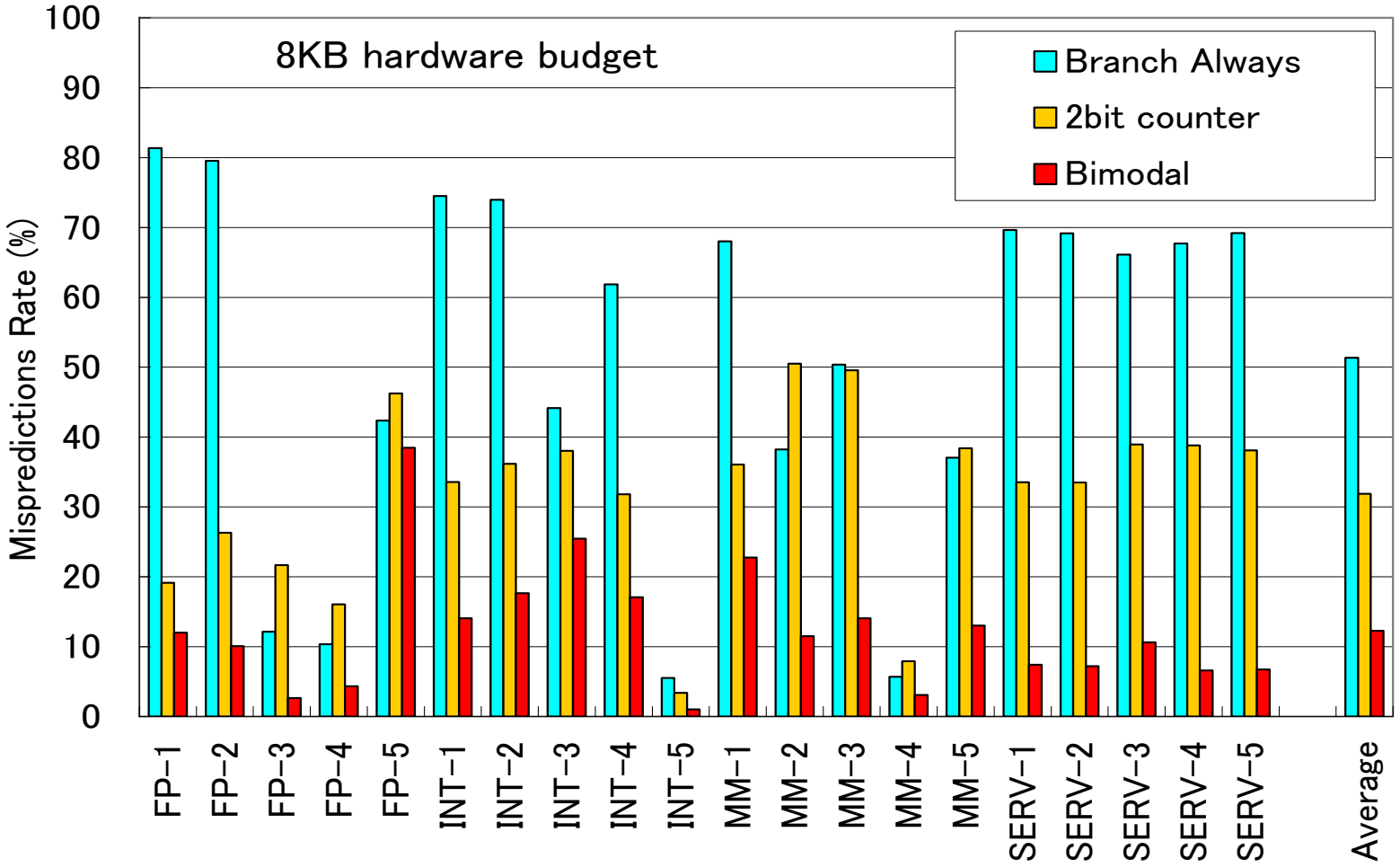
Predicting the sequence of 01010100 01010100 01010100 ...

| | |
|---|---|
| The B4's sequence of | 01010100 01010100 01010100 ... |
| State of the counter | 2 1 0 0 0 0 0 0 0 0 0 0 ... |
| Prediction | 1 0 0 0 0 0 0 0 0 0 0 0 ... |
| Hit/Miss or the pred. | M H H H H H H H H H H H ... |
| The B2's sequence of | 01010100 01010100 01010100 ... |
| State of the counter | 2 3 3 3 2 3 3 3 2 3 3 3 ... |
| Prediction | 1 1 1 1 1 1 1 1 1 1 1 1 ... |
| Hit/Miss or the pred. | H H H M H H H M H H H M ... |



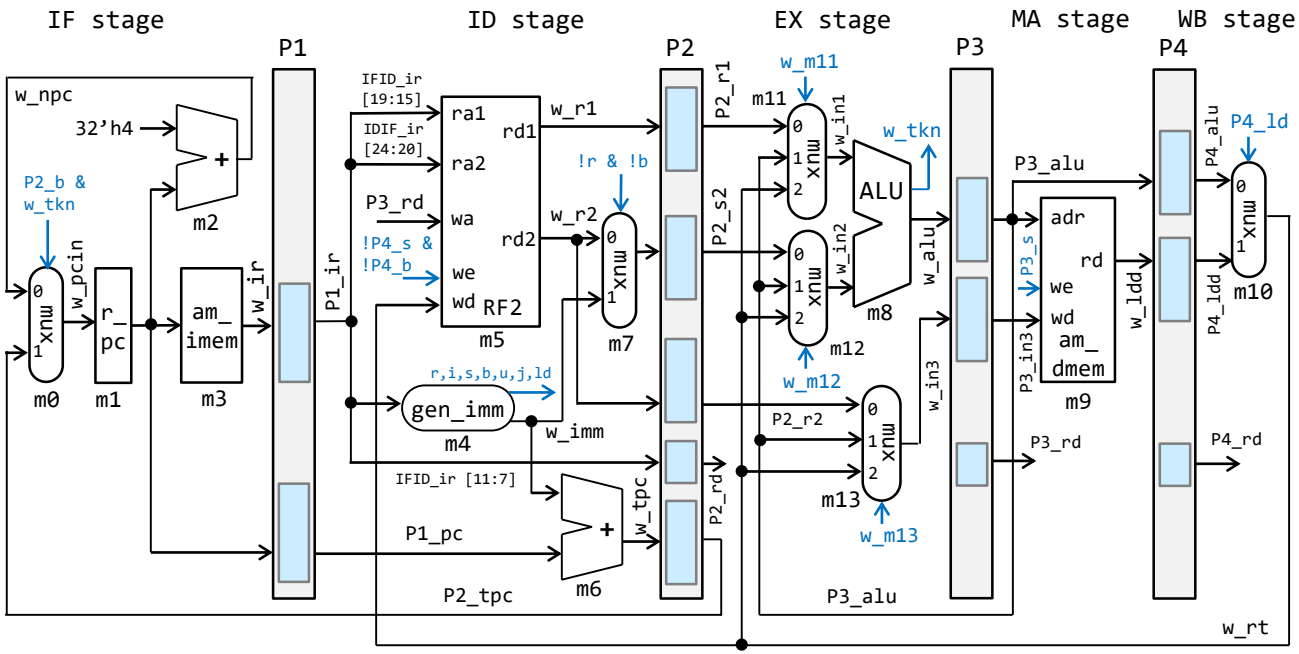
# Accuracy of simple predictors with 8KB HW budget




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

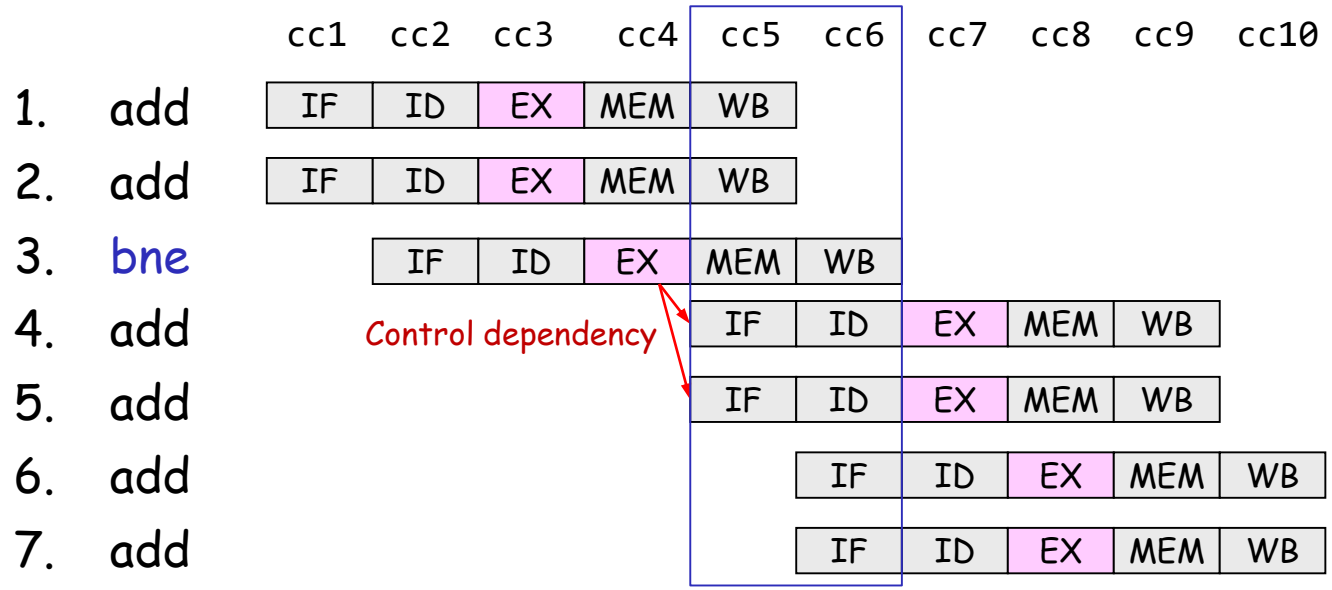
# 5-stage pipelining RISC-V processor with data forwarding

- The strategy is to separate instruction fetch step (IF), instruction decode step (ID), execution step (EX), memory access step (MA), and write back step (WB).
- Use the pipeline registers P1, P2, P3, P4.



# Why do branch instructions degrade IPC?

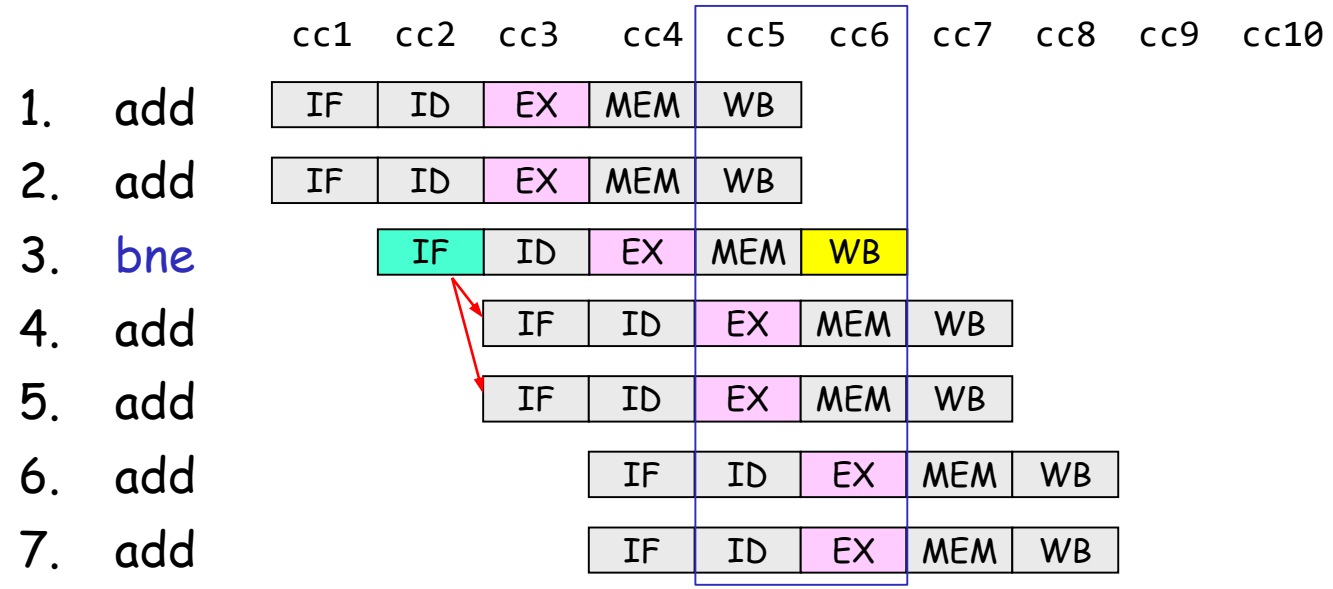
- The branch taken / untaken is determined in the execution (EX) stage of the branch.
- The conservative approach is stalling instruction fetch until the branch direction is determined.
  - It is too conservative to be practical.




2-way superscalar processor executing instruction sequence with a branch

# Why do branch instructions degrade IPC?

- The branch taken / untaken is determined in the execution (EX) stage of the branch.
- Prediction and speculation, then training
- Recovery whan a prediction miss

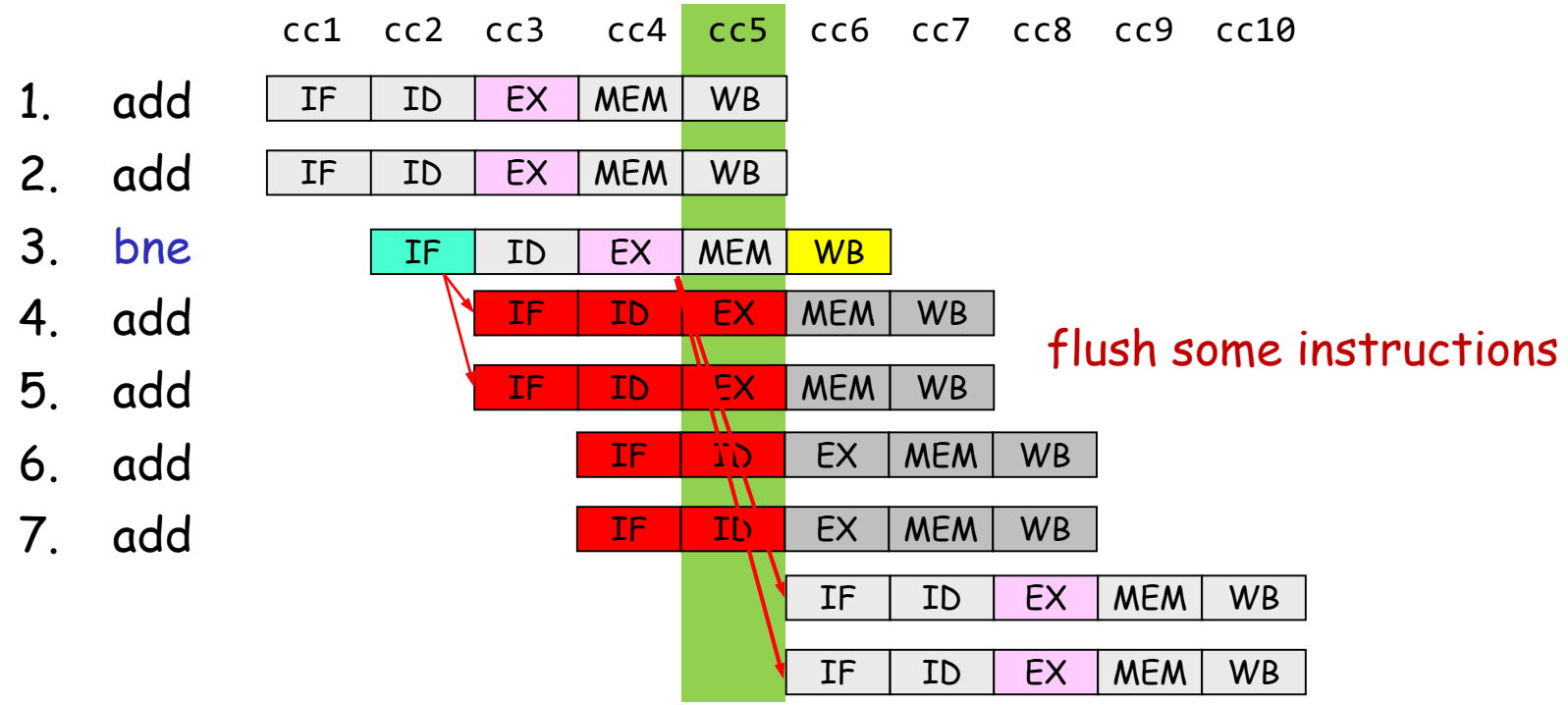



2-way superscalar processor executing instruction sequence with a branch

Speculative execution performs some task that may not be needed. Work is done before it is known whether it is actually needed, so as to prevent a delay that would have to be incurred by doing the work after it is known that it is needed.

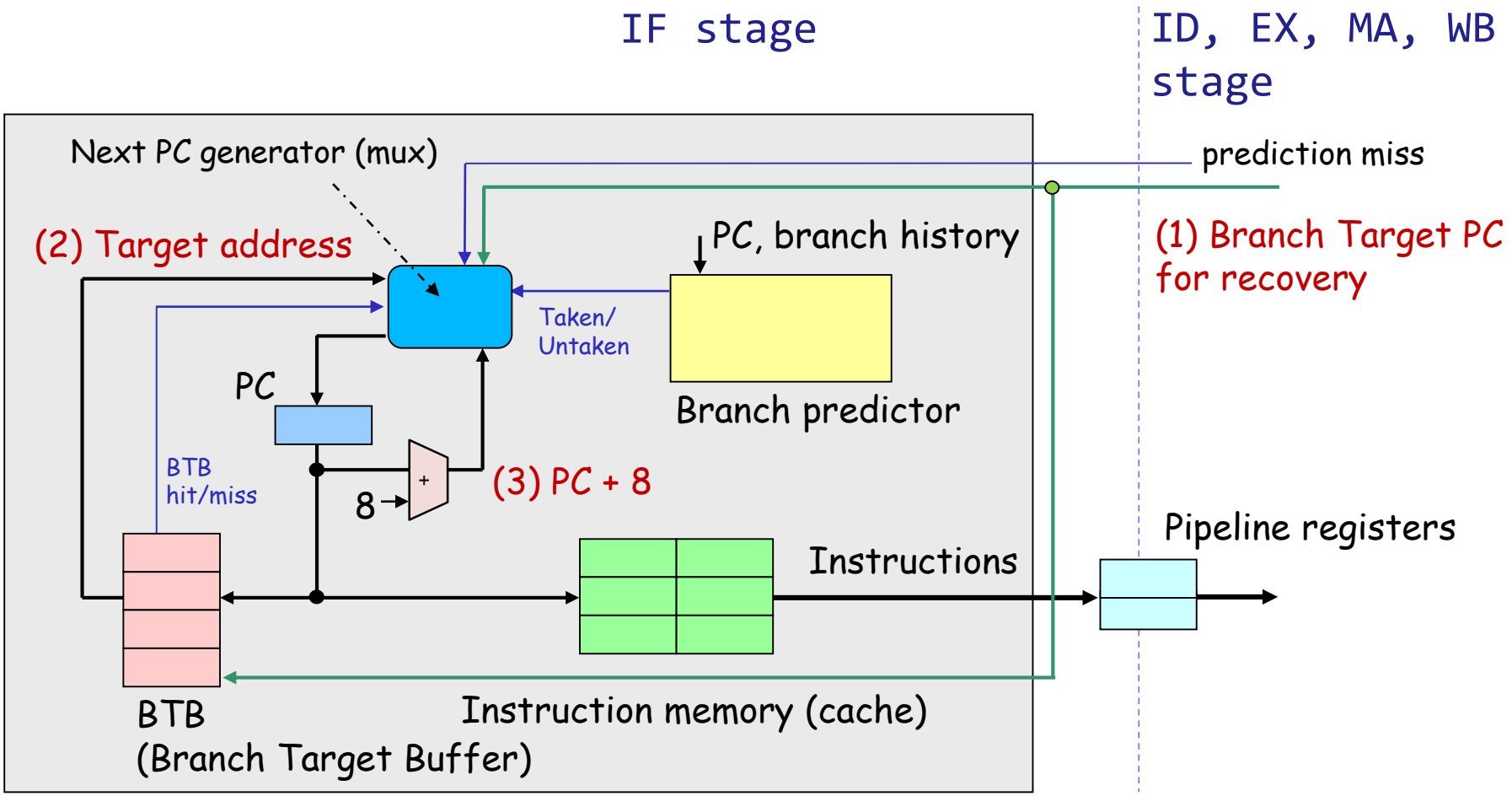
# Why do branch instructions degrade IPC?

- The branch taken / untaken is determined in execution stage of the branch.
- Prediction and speculation, then training
- Recovery whan a prediction miss
  - If it turns out a prediction miss, some results are ignored and some changes made by the speculative execution are recovered.



# Instruction fetch unit of 2-way super-scalar

- High-bandwidth instruction delivery using prediction, and speculation



# An innovation in branch predictors in 1993

- Using branch history
  - global branch history
  - local branch history
- 2-level branch predictor and gshare
- Assume predicting the sequence 1110 1110 1110 1110 1110 ...

```
1110111 0
11101110 ?
111011101 ?
1110111011 ?
11101110111 ?
111011101110 ?
```

| adr | pred |
|---|---|
| 000 | |
| 001 | |
| 010 | |
| 011 | 1 |
| 100 | |
| 101 | 1 |
| 110 | 1 |
| 111 | 0 |

Use the recent branch history as an address of a table.

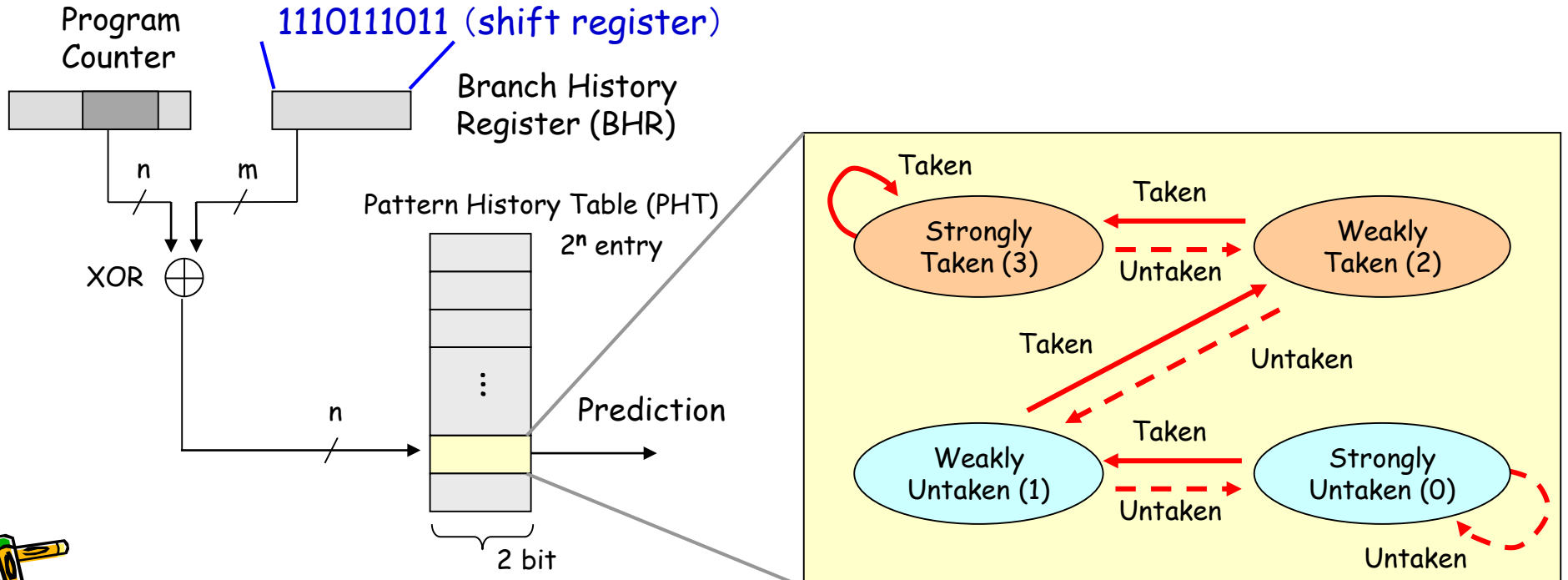
# Recommended Reading

- Combining Branch Predictors
  - Scott McFarling, Digital Western Research Laboratory
  - WRL Technical Note TN-36, 1993

- A quote:
  "In this paper, we have presented two new methods for improving branch prediction performance. First, we showed that using the bit-wise exclusive OR of the global branch history and the branch address to access predictor counters results in better performance for a given counter array size."

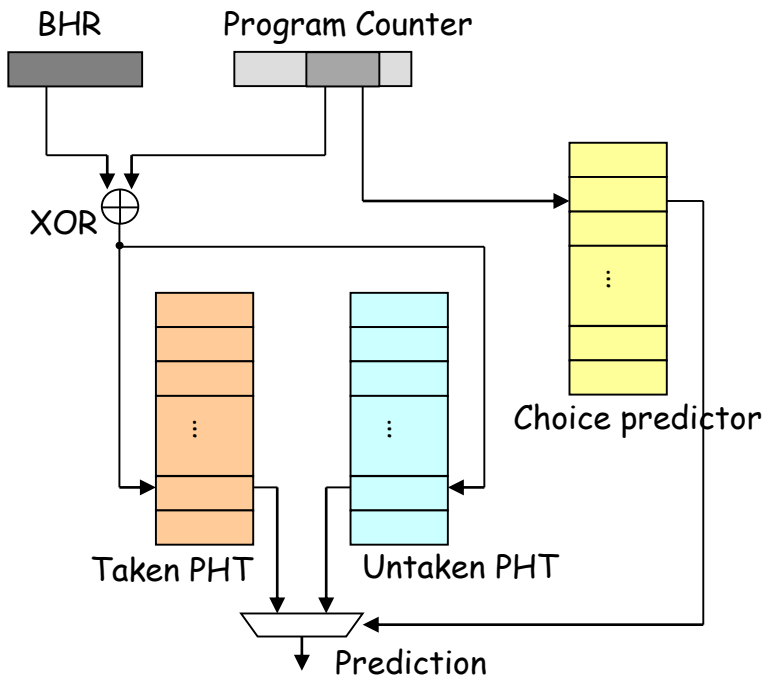
# Gshare (TR-DEC 1993)

- How to predict
  - Using the exclusive OR of the global branch history and PC to access PHT, then MSB of the selected counter is the prediction.
- How to update
  - Shifting BHR one bit left and update LSB by branch outcome in IF stage.
  - Update the used counter in the same way as 2BC in WB stage.



# Bi-Mode (MICRO 1997)

- A choice predictor (bimodal) is used as a meta-predictor
- How to predict
  - Like gshare, both of Taken PHT and Untaken PHT make two predictions.
  - Select one among them by the choice predictor which tracks the global bias of a branch.
- How to update
  - The used PHT is updated in the same way as 2BC.
  - Choice predictor is updated in the same way as bimodal.



# To go beyond gshare

- Using branch history
  - global branch history
  - local branch history
- 2-level branch predictor and gshare

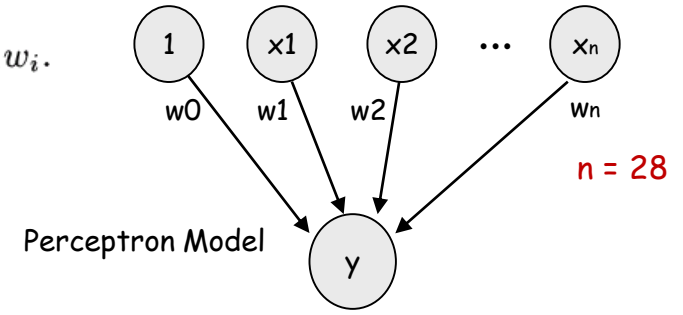
- Assume predicting the sequence 1110 1110 1110 1110 1110 …

```
11101110 ?          1110 1110 ?
111011101 ?         11101 1101 ?
1110111011 ?        111011 1011 ?
11101110111 ?       1110111 0111 ?
111011101110 ?      11101110 1110 ?
```

# Perceptron (HPCA 2001)

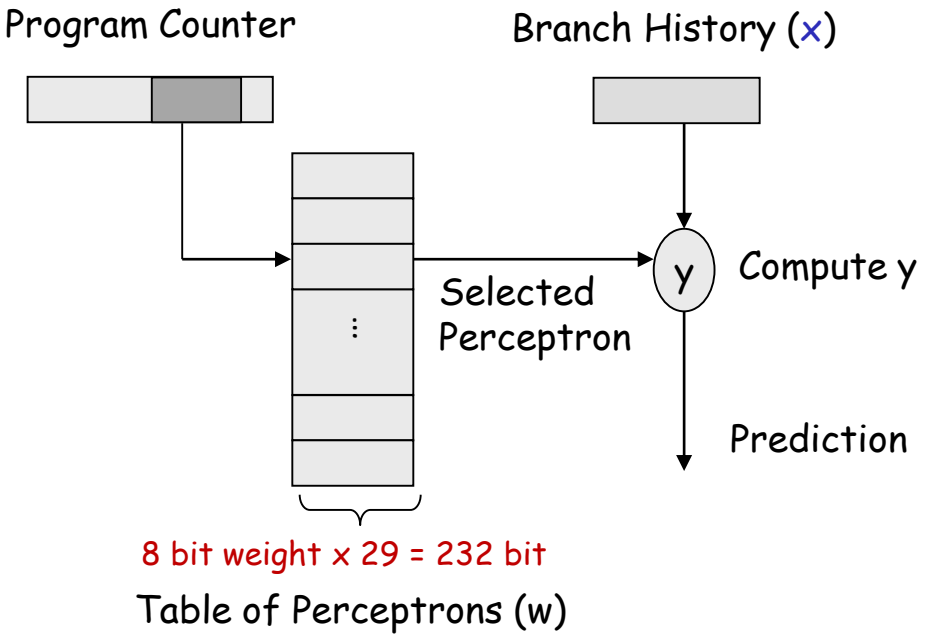
- How to predict
  - Select one perceptron by PC
  - Compute y using the equation. It predicts 1 if y>=0, predicts 0 if y<0
  - x is branch history. xi is either -1, meaning not taken or 1, meaning taken
- How to update
  - Train the weights of used perceptron when the prediction miss or |y| < T (Threshold)

$$y = w_0 + \sum_{i=1}^{n} x_i w_i.$$




```
if sign(y_out) ≠ t or |y_out| ≤ θ then
        for i := 0 to n do
                w_i := w_i + t x_i
        end for
end if
```

T = 1.93n + 14



# Exercise 1

- How to predict
  - Select one perceptron by PC
  - Compute y using the equation. It predicts 1 if y>=0, predicts 0 if y<0
  - x is branch history. xi is either -1, meaning not taken or 1, meaning taken
- How to update
  - Train the weights of used perceptron when the prediction miss or |y| < T (Threshold)

```
if sign(y_out) ≠ t or |y_out| ≤ θ then
        for i := 0 to n do
                w_i := w_i + t x_i
        end for
end if
```

T = 1.93n + 14

```
  1  T=21.72: bias(  1)  -1  -1  -1  -1 : pred=1 outcome=1 : hit
  2  T=21.72: bias(  2)   0  -2  -2  -2 : pred=1 outcome=1 : hit
  3  T=21.72: bias(  3)   1  -1  -3  -3 : pred=1 outcome=1 : hit
  4  T=21.72: bias(  2)   0  -2  -4  -2 : pred=1 outcome=0 : miss
  5  T=21.72: bias(  3)  -1  -1  -3  -1 : pred=0 outcome=1 : miss
  6  T=21.72: bias(  4)   0  -2  -2   0 : pred=0 outcome=1 : miss
  7  T=21.72: bias(  5)   1  -1  -3   1 : pred=1 outcome=1 : hit
  8  T=21.72: bias(  4)   0  -2  -4   2 : pred=1 outcome=0 : miss
  9  T=21.72: bias(  5)  -1  -1  -3   3 : pred=1 outcome=1 : hit
 10  T=21.72: bias(  6)   0  -2  -2   4 : pred=1 outcome=1 : hit
 11  T=21.72: bias(  7)   1  -1  -3   5 : pred=1 outcome=1 : hit
 12  T=21.72: bias(  6)   0  -2  -4   6 : pred=0 outcome=0 : hit
 13  T=21.72: bias(  7)  -1  -1  -3   7 : pred=1 outcome=1 : hit
 14  T=21.72: bias(  8)   0  -2  -2   8 : pred=1 outcome=1 : hit
 15  T=21.72: bias(  9)   1  -1  -3   9 : pred=1 outcome=1 : hit
 16  T=21.72: bias(  8)   0  -2  -4  10 : pred=0 outcome=0 : hit
 17  T=21.72: bias(  9)  -1  -1  -3  11 : pred=1 outcome=1 : hit
 18  T=21.72: bias( 10)   0  -2  -2  12 : pred=1 outcome=1 : hit
 19  T=21.72: bias( 10)   0  -2  -2  12 : pred=1 outcome=1 : hit
 20  T=21.72: bias(  9)  -1  -3  -3  13 : pred=0 outcome=0 : hit
 21  T=21.72: bias( 10)  -2  -2  -2  14 : pred=1 outcome=1 : hit
 22  T=21.72: bias( 10)  -2  -2  -2  14 : pred=1 outcome=1 : hit
 23  T=21.72: bias( 10)  -2  -2  -2  14 : pred=1 outcome=1 : hit
 24  T=21.72: bias(  9)  -3  -3  -3  15 : pred=0 outcome=0 : hit
 25  T=21.72: bias( 10)  -4  -2  -2  16 : pred=1 outcome=1 : hit
 26  T=21.72: bias( 10)  -4  -2  -2  16 : pred=1 outcome=1 : hit
 27  T=21.72: bias( 10)  -4  -2  -2  16 : pred=1 outcome=1 : hit
 28  T=21.72: bias(  9)  -5  -3  -3  17 : pred=0 outcome=0 : hit
 29  T=21.72: bias(  9)  -5  -3  -3  17 : pred=1 outcome=1 : hit
 30  T=21.72: bias( 10)  -4  -4  -2  18 : pred=1 outcome=1 : hit
 31  T=21.72: bias( 10)  -4  -4  -2  18 : pred=1 outcome=1 : hit
 32  T=21.72: bias(  9)  -5  -5  -3  19 : pred=0 outcome=0 : hit
 33  T=21.72: bias(  9)  -5  -5  -3  19 : pred=1 outcome=1 : hit
 34  T=21.72: bias(  9)  -5  -5  -3  19 : pred=1 outcome=1 : hit
 35  T=21.72: bias( 10)  -4  -4  -4  20 : pred=1 outcome=1 : hit
```

# Perceptron (HPCA 2001)

## The Neural Network in Your CPU

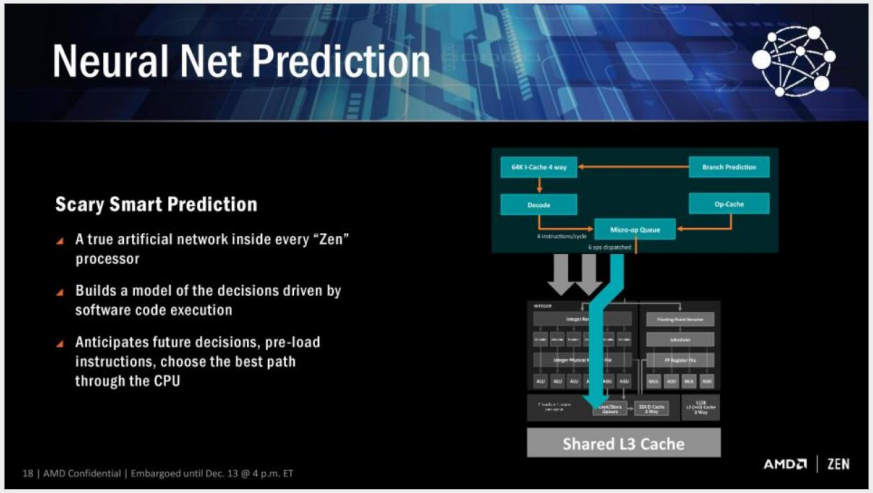
Sun, Aug 6, 2017

Machine learning and artificial intelligence are the current hype (again). In their new Ryzen processors, AMD advertises the Neural Net Prediction. It turns out this is was already used in their older (2012) Piledriver architecture used for example in the AMD A10-4600M. It is also present in recent Samsung processors such as the one powering the Galaxy S7. What is it really?

The basic idea can be traced to a paper from Daniel Jimenez and Calvin Lin "Dynamic Branch Prediction with Perceptrons", more precisely described in the subsequent paper "Neural methods for dynamic branch prediction". Branches typically occur in `if-then-else` statements. Branch prediction consists in guessing which code branch, the `then` or the `else`, the code will execute, thus allowing to precompute the branch in parallel for faster evaluation.

Jimenez and Lin rely on a simple single-layer perceptron neural network whose input are the branch outcome (global or hybrid local and global) histories and the output predicts which branch will be taken. In reality, because there is a single layer,

AMD Ryzen 2016-12-13 Slide Deck — Back to Post

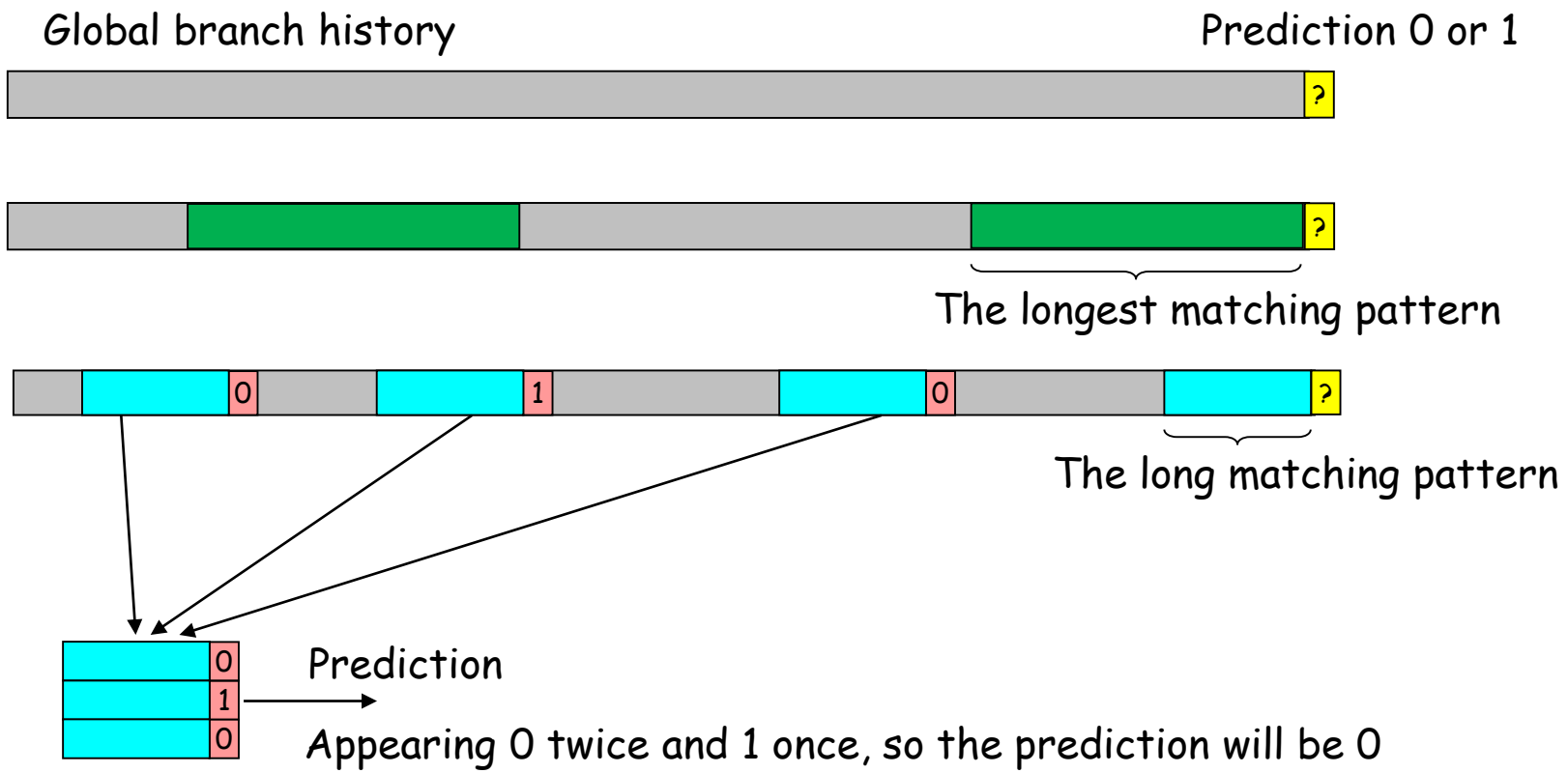



https://www.anandtech.com/Gallery/Album/5197#18

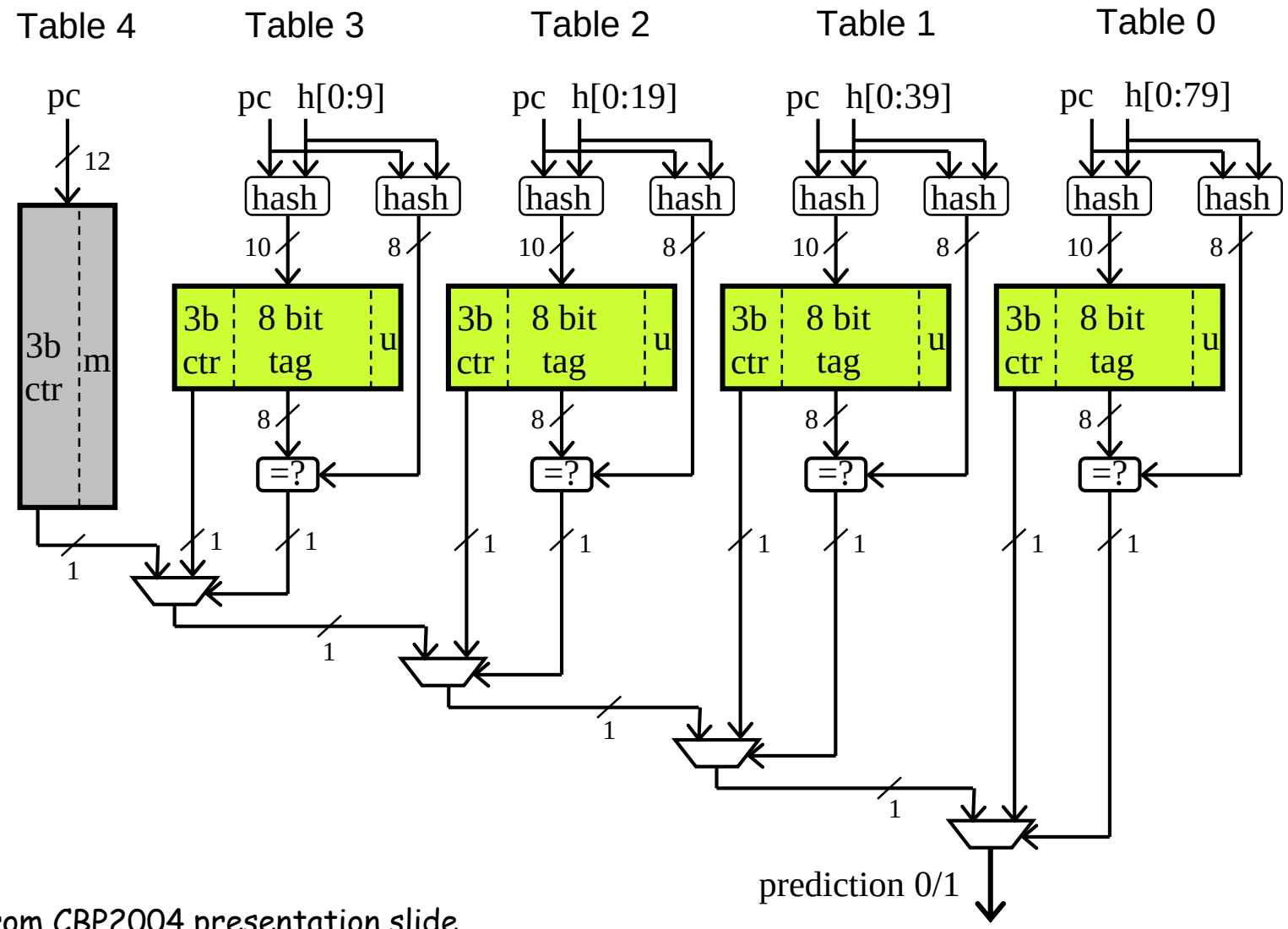
https://chasethedevil.github.io/post/the_neural_network_in_your_cpu/

# Branch predictors based on pattern matching

- Find the longest matching pattern (green rectangle)
- Select the proper matching length or long matching pattern (blue rectangle)
- Count the number of 0 and the number of 1 after the long matting patterns (red rectangle), then predict by majority vote.

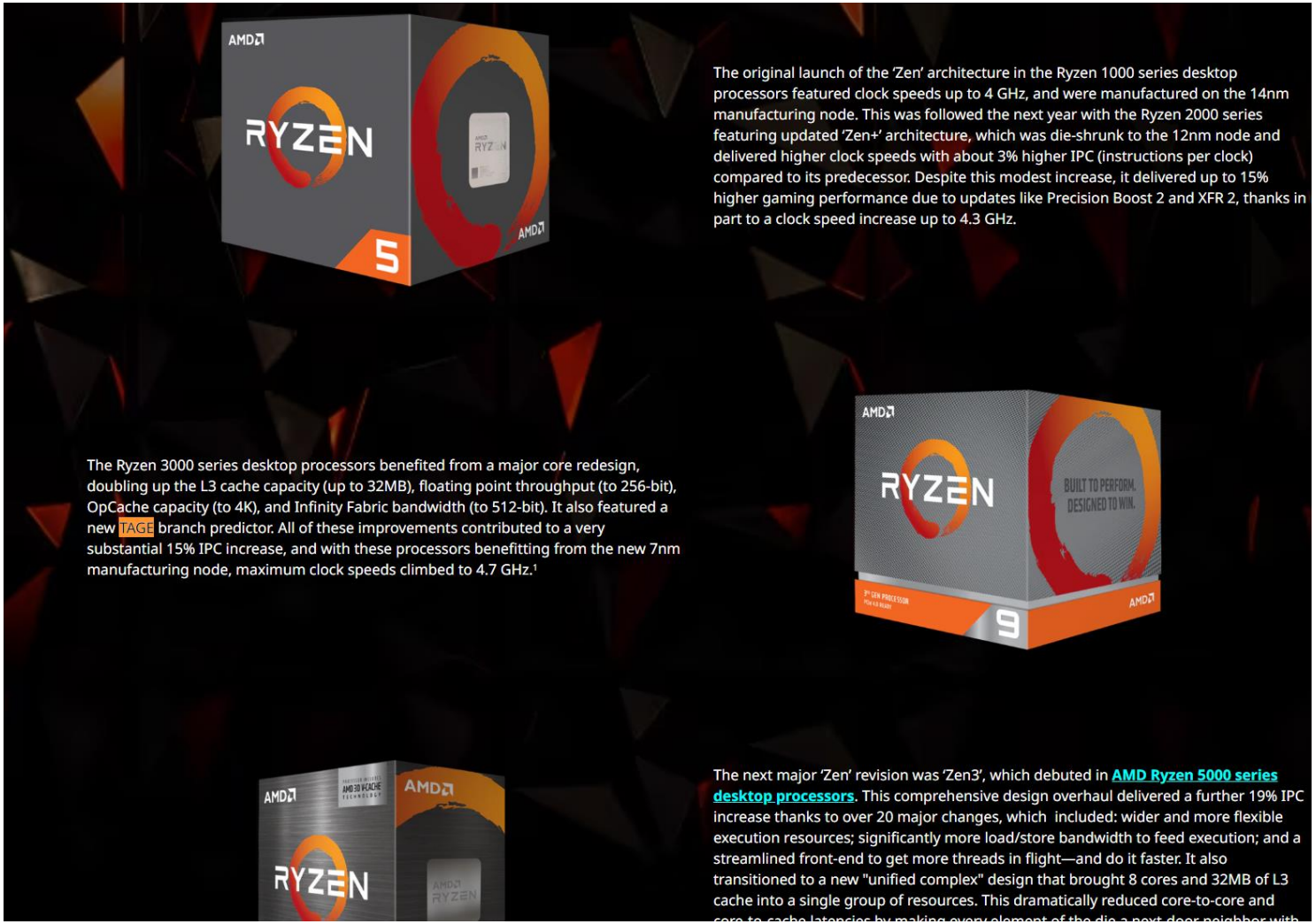


# Partial Pattern Matching, PPM or TAGE (CBP 2004)




From CBP2004 presentation slide

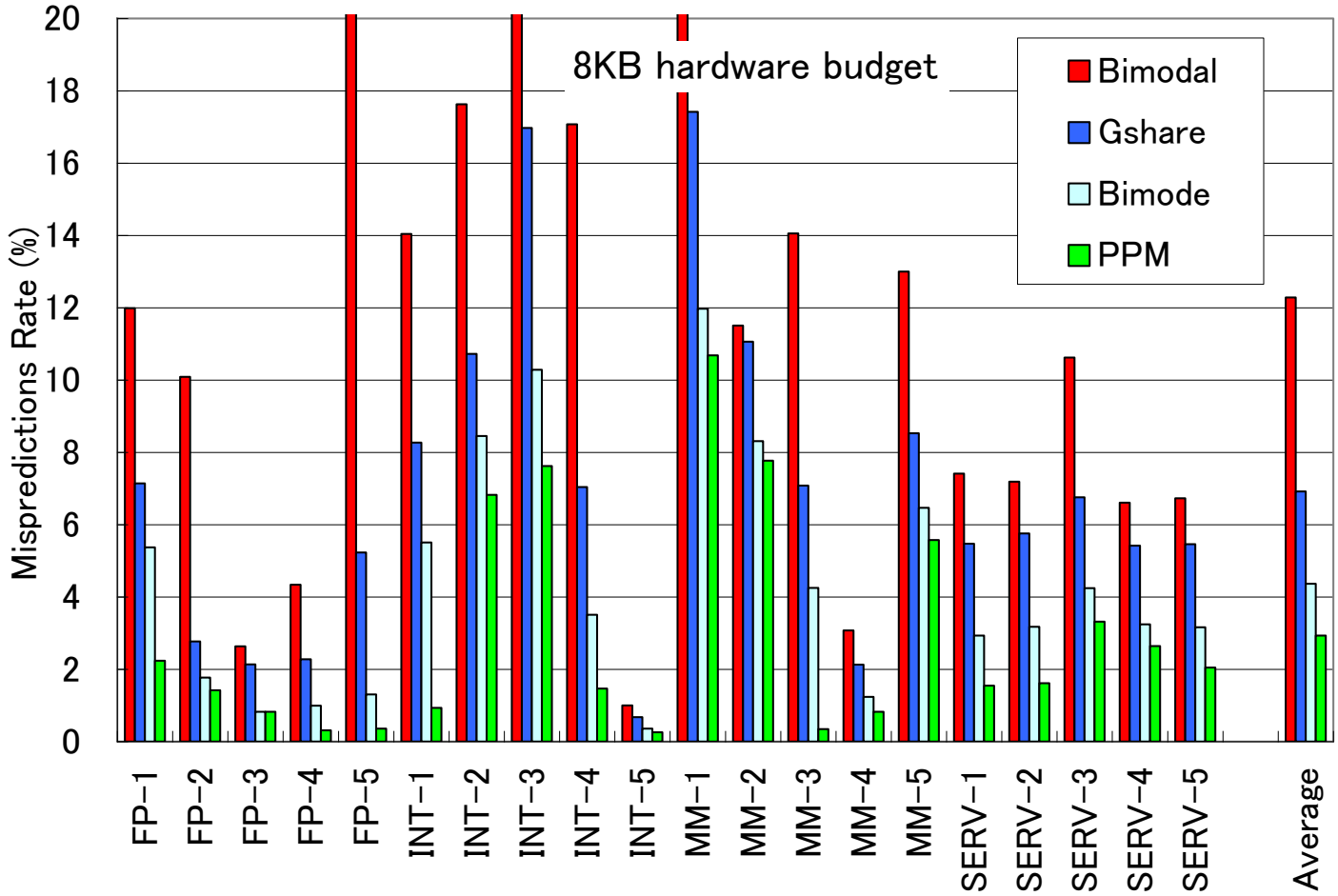
# Partial Pattern Matching, PPM or TAGE (CBP 2004)




https://www.amd.com/en/technologies/zen-core

# Prediction accuracy

- The accuracy of 4KB Gshare is about 93%.
- The accuracy of 4KB PPM is about 97%.



# Recommended Reading

- Prophet-Critic Hybrid Branch Prediction
  - Ayose Falcon, UPC, Jared Stark, Intel, Alex Ramirez, UPC, Konrad Lai, Intel, Mateo Valero
  - ISCA-31 pp. 250-261 (2004)

# A quote from Introduction (1/2)

Conventional predictors are analogous to a taxi with just one driver.

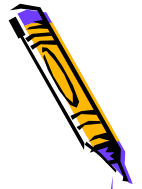
He gets the passenger to the destination using knowledge of the roads acquired from previous trips; i. e., using history information stored in the predictor's memory structures.

When he reaches an intersection, he uses this knowledge to decide which way to turn.

The driver accesses this knowledge in the context of his current location.

Modern branch predictors access it in the context of the current location (the program counter) plus a history of the most recent decisions that led to the current location.

# A quote from Introduction (2/2)

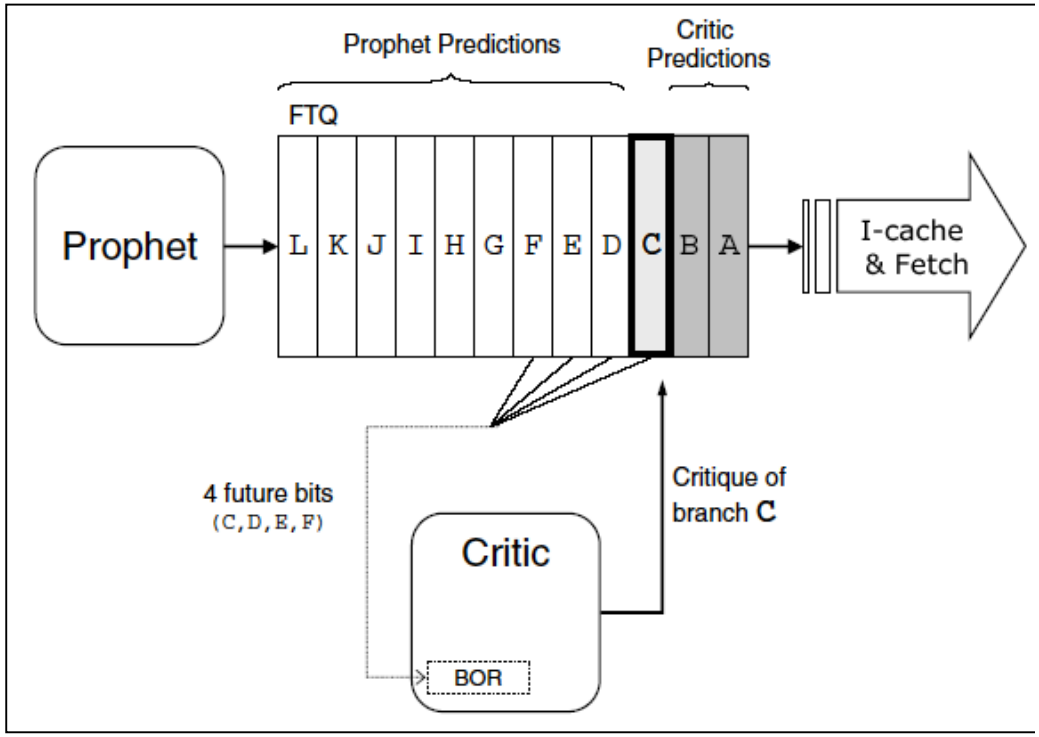


Prophet/critic hybrids are analogous to a taxi with two drivers: the front-seat and the back-seat. The front-seat driver has the same role as the driver in the single-driver taxi. This role is called the prophet.

The back-seat driver has the role of critic. She watches the turns the prophet makes at intersections. She doesn't say anything unless she thinks he's made a wrong turn. When she thinks he's made a wrong turn, she waits until he's made a few more turns to be certain they are lost. (Sometimes the prophet makes turns that initially look questionable, but, after he makes a few more turns, in hindsight appear to be correct.) Only when she's certain does she point out the mistake.

To recover, they backtrack to the intersection where she believes the wrong-turn was made and try a different direction.

# Prophet-Critic Hybrid Branch Prediction



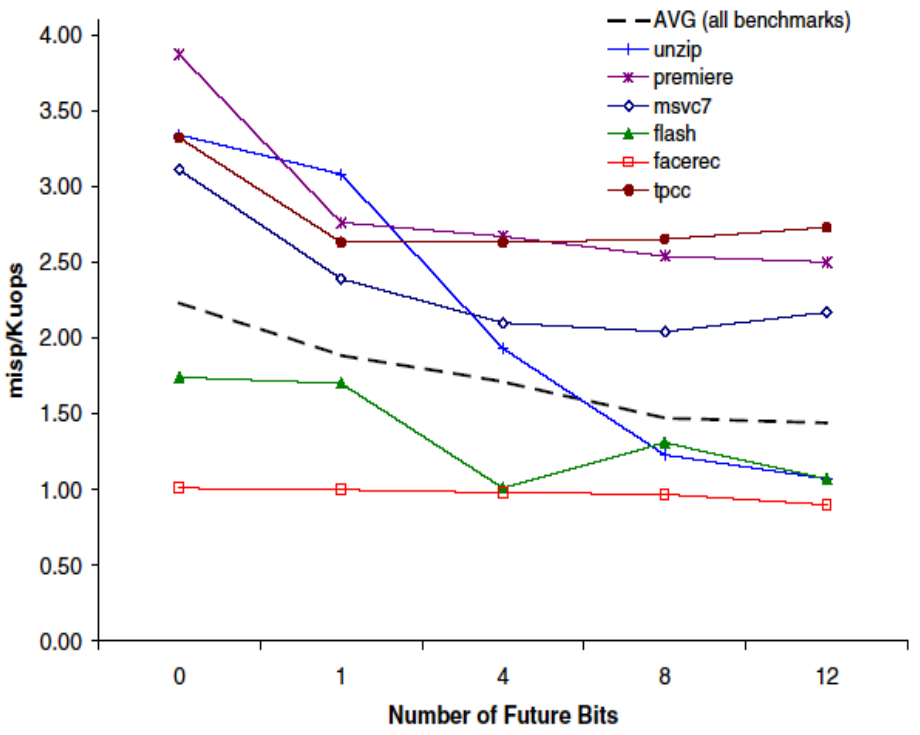




**Figure 5. Effect of varying the number of future bits used by the critic on prediction accuracy for selected benchmarks. (prophet: 8KB perceptron; critic: 8KB tagged gshare)**