Lecture 16

# Computer Architecture
## (Spring 2020)

Dynamic Hardware Branch Prediction

& Branch-Target Buffers

Dr. Duo Liu (刘铎)
Office: Main Building 0626
Email: liuduo@cqu.edu.cn

# Review: Reducing CPI (or Increasing IPC)

$$CPI = CPI_{ideal} + stalls_{structural} + stalls_{dataHazard} + stalls_{control}$$

| technique | reduces |
|---|---|
| forwarding/ bypassing | potential data-hazard stalls |
| delayed branches | control-hazard stalls |
| basic dynamic scheduling (scoreboarding) | data-hazard stalls from true dependencies |
| dynamic scheduling with register renaming | data-hazard, anti-dep. & output dep. stalls |
| dynamic branch prediction | control stalls |
| issuing multiple instruction per clock cycle | ideal CPI |
| speculation | data-hazard and control-hazard stalls |
| dynamic memory disambiguation | data-hazard stalls with memory |
| loop unrolling | control hazard stalls |
| basic compiler pipeline scheduling | data-hazard stalls |
| compiler dependency analysis | ideal CPI, data-hazard stalls |
| software pipelining & trace scheduling | ideal CPI, data-hazard stalls |
| compiler speculation | ideal CPI, data-hazard stalls |

# Reducing Branch Penalty

Branch penalty : wasted cycles due to pipeline flushing on mis-predicted branches

Reduce branch penalty:

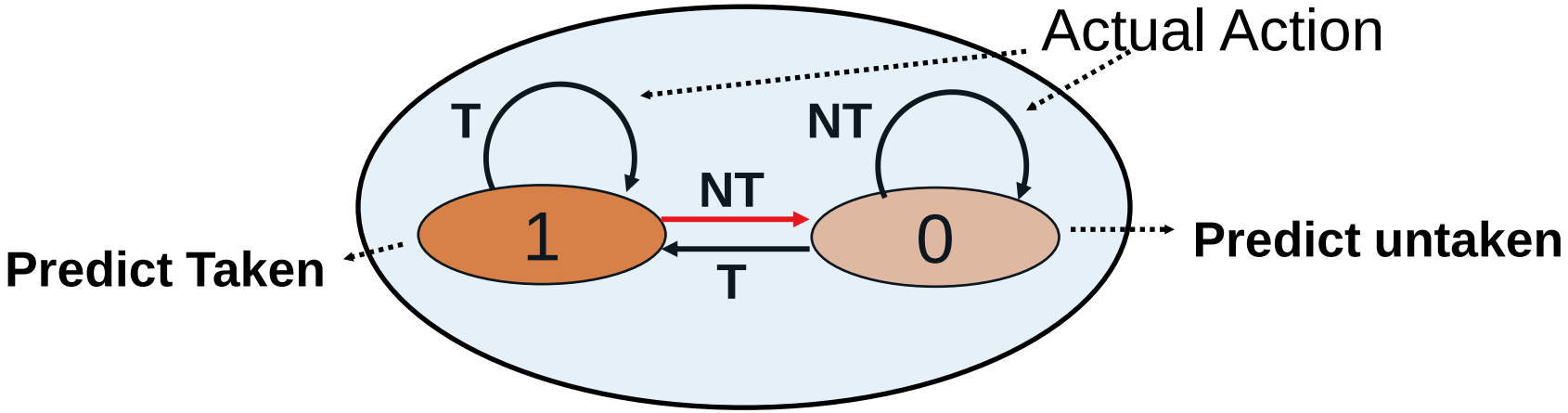
1. Predict branch/jump instructions AND branch direction (taken or not taken)
2. Predict branch/jump target address (for taken branches)
3. Speculatively execute instructions along the predicted path

# Branch Prediction Strategies

- Static
  - Decided before runtime
  - Examples:
    - Always-Not Taken
    - Always-Taken
    - Backwards Taken, Forward Not Taken (BTFNT)
    - Profile-driven prediction
- Dynamic
  - Prediction decisions may change during the execution of the program

# A Simple Scheme: 1-bit Predictor for a Single Branch

**1-bit prediction**



# Basic Branch Prediction: Branch-History Table (or Branch-Prediction Buffer)

- Implemented as a small memory indexed by a portion (usually some low-significant bits) of the address of the branch instruction.
  - So the size of this table is the number of entries $\times$ the number of bit per entry.
  - If unfortunately, the address portions of two branch instructions are identical, they share one entry. Hence, the more entries, the less such conflicts.

e.g., branch instructions at the following addresses share the BHT entry

- 00F2BC 0101 1100
- 010A5D 1001 1100

| Index | Taken? |
|---|---|
| 0000 | 1 |
| 0001 | 0 |
| 0010 | 1 |
| 0011 | 0 |
| 0100 | 1 |
| 0101 | 0 |
| 0110 | 1 |
| 0111 | 1 |
| 1000 | 1 |
| 1001 | 0 |
| 1010 | 1 |
| 1011 | 1 |
| 1100 | 0 |
| 1101 | 1 |
| 1110 | 1 |
| 1111 | 1 |

# 1-bit Predictor Weakness on nested loops

- Consider a nested loop :

```
for (…){
   for (i=0; i<10; i++)
      a[i] = a[i] * 2.0;
}
```

  - Mispredict twice, once on entry, once on exit, every time when the inner loop is executed.

# Example: Accuracy of 1-Bit Prediction Scheme in the Presence of "Loop Branching"

```
loop: L.D F0, 0(R1)
      MUL.D F4, F0, F2
      S.D F4, 0(R1)
      DADDIU R1, R1, #-8
      BNEZ R1 loop
```

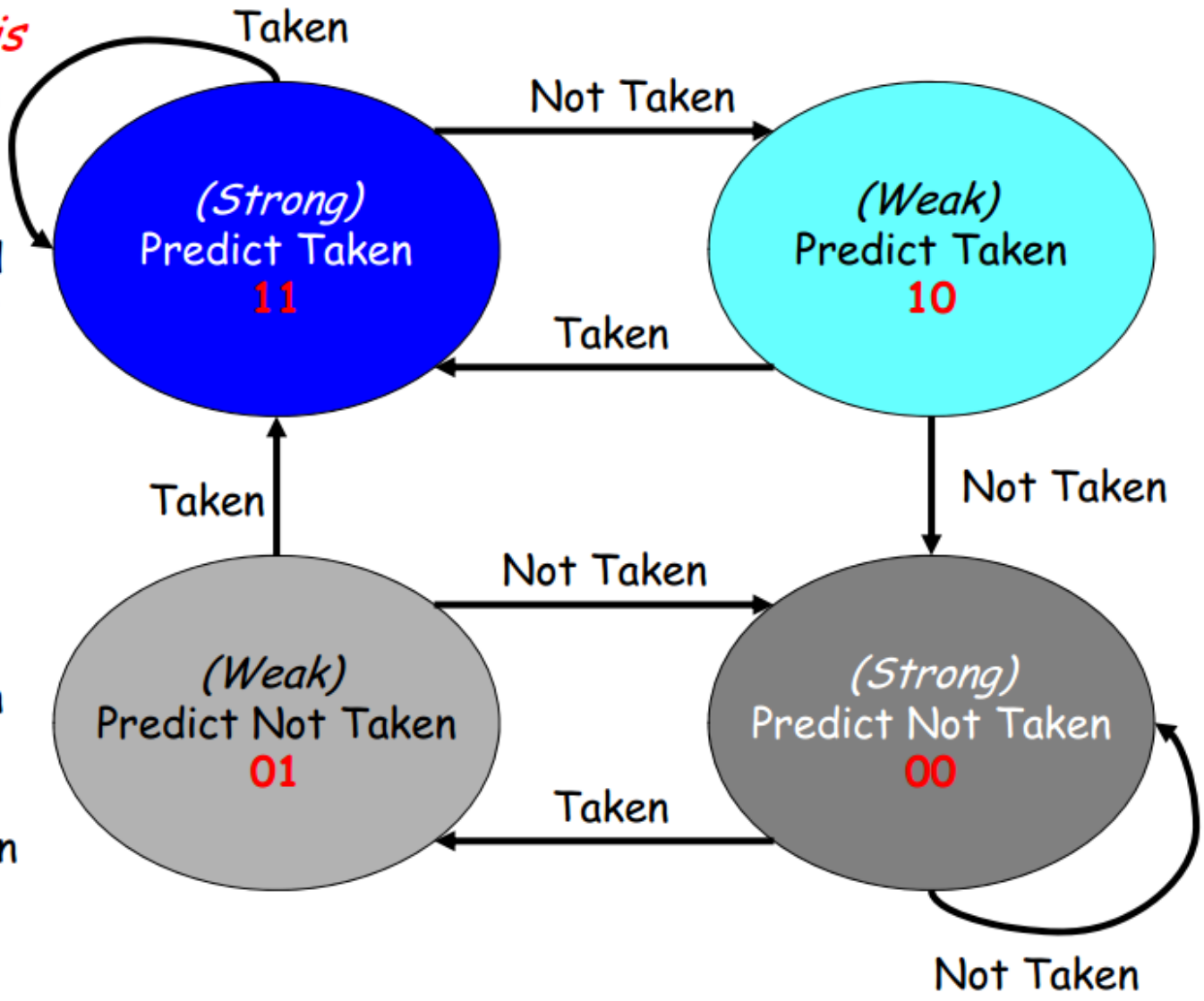
- Assumptions
  - R1 is initialized to #80

| Iteration | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Predicted behavior | NT | T | T | T | T | T | T | T | T | T | T | NT | T | T | T | T | T | T | T | T | T | T |
| Actual behavior | T | T | T | T | T | T | T | T | T | T | NT | T | T | T | T | T | T | T | T | T | T | NT |

- Branch taken 90% of the times, but branch prediction accuracy is only 80%
- A 1-bit predictor for "loop branches" mispredicts at twice the rate that the branch is not taken

# 2-Bit Prediction Scheme (a.k.a. Bimodal Predictor)

- Adding *hysteresis* to the prediction scheme
  - a prediction must be missed twice before it is changed
- Implementation
  - a *(2-bit) saturated counter* that is incremented on a taken branch and decremented on an untaken branch



# Example: 2-Bit Prediction Scheme in Action with "Loop Branching"

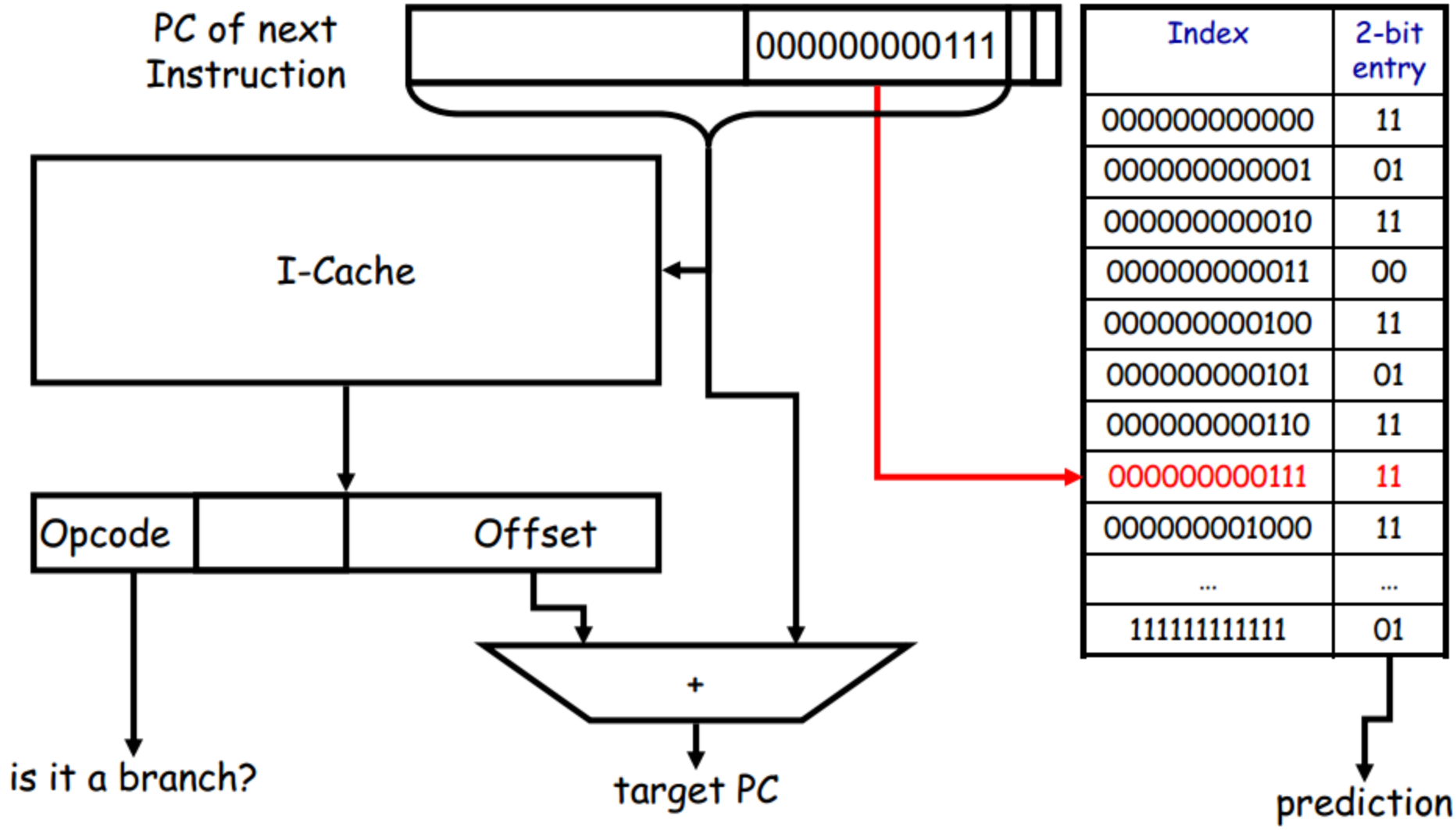
```
loop: L.D F0, 0(R1)
      MUL.D F4, F0, F2
      S.D F4, 0(R1)
      DADDIU R1, R1, #-8
      BNEZ R1 loop
```

- Assumptions
  - R1 is initialized to #80

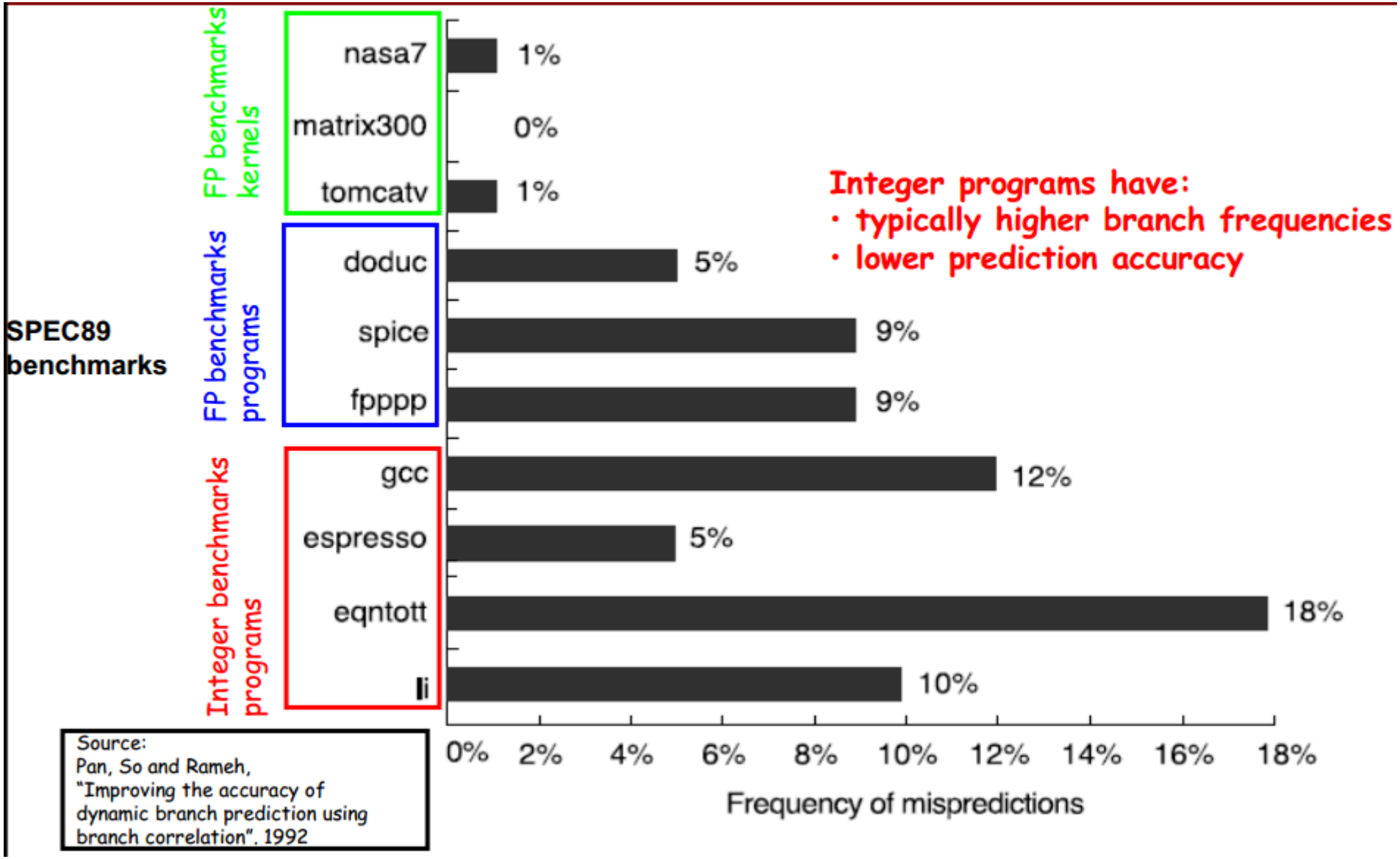
| Iterationss & steps | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Predicted behavior | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T |
| Actual behavior | T | T | T | T | T | T | T | T | T | T | NT | T | T | T | T | T | T | T | T | T | T | NT |

- Branch taken 90% of the times, and branch prediction accuracy is now 90%
- The 2-bit predictor mispredicts at the 10th step of the 1st iteration, but *doesn't change its mind*. It just moves to "weak-predict-taken" for the 1st step of the following iteration

# Branch History Table for 4K-Entry 2-Bit Predictor: 4K Entry × 2 bits/Entry = 8K bit



# Measure: Prediction Accuracy of 4K-entry 2-bit-prediction buffer with SPEC89



# Example

A snapshot of the taken/not-taken behavior of a branch is:

... T T T T T T T T **N N T T N N T N N T**

If the branch predictor used is a 2-bit saturating counter, how many of the last ten branches are predicted correctly?

**Answer：**

According to the branch predictor in textbook, the prediction for the last ten branches are:

ST, WT, SN, WN, ST, WT, SN, WN, SN, SN

According to the 2-bit saturating counter:

ST, WT, WN, WT, ST, WT, WN, WT, WN, SN

No matter which one you use, the answer is 2 branches are predicted correctly.