Advanced Computer Architecture

Lecture 5

Control Hazards & Branch Prediction

Dr. John Jose

Assistant Professor

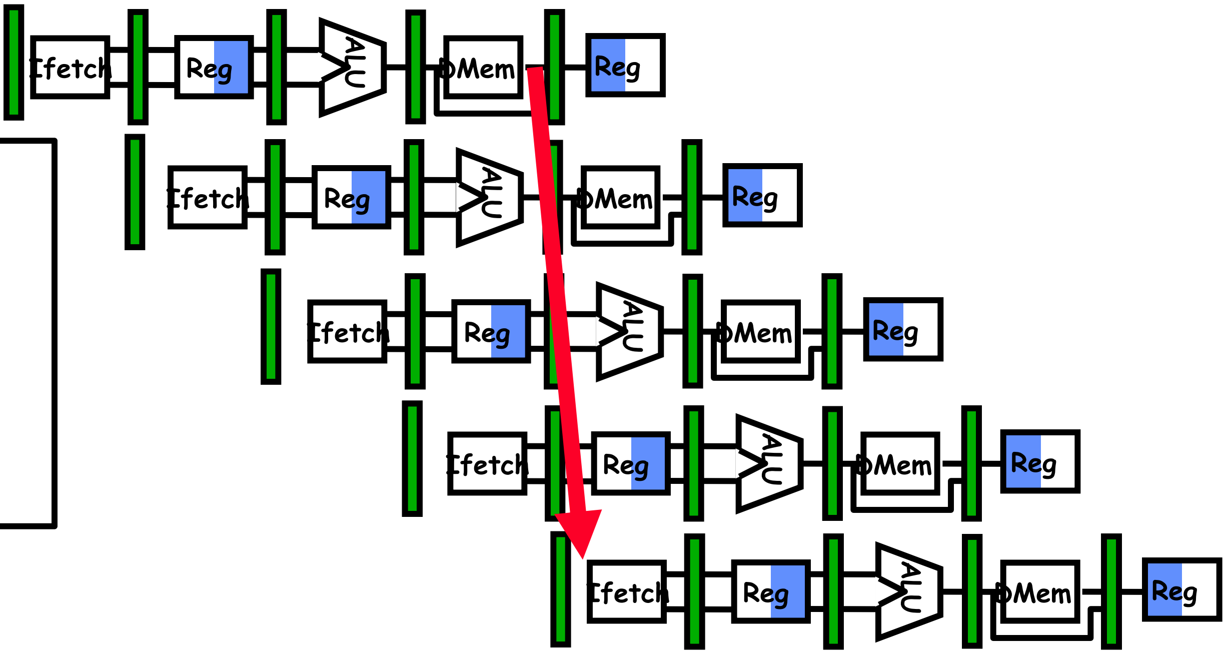
Department of Computer Science & Engineering

Indian Institute of Technology Guwahati, Assam.

Control Hazard on Branches

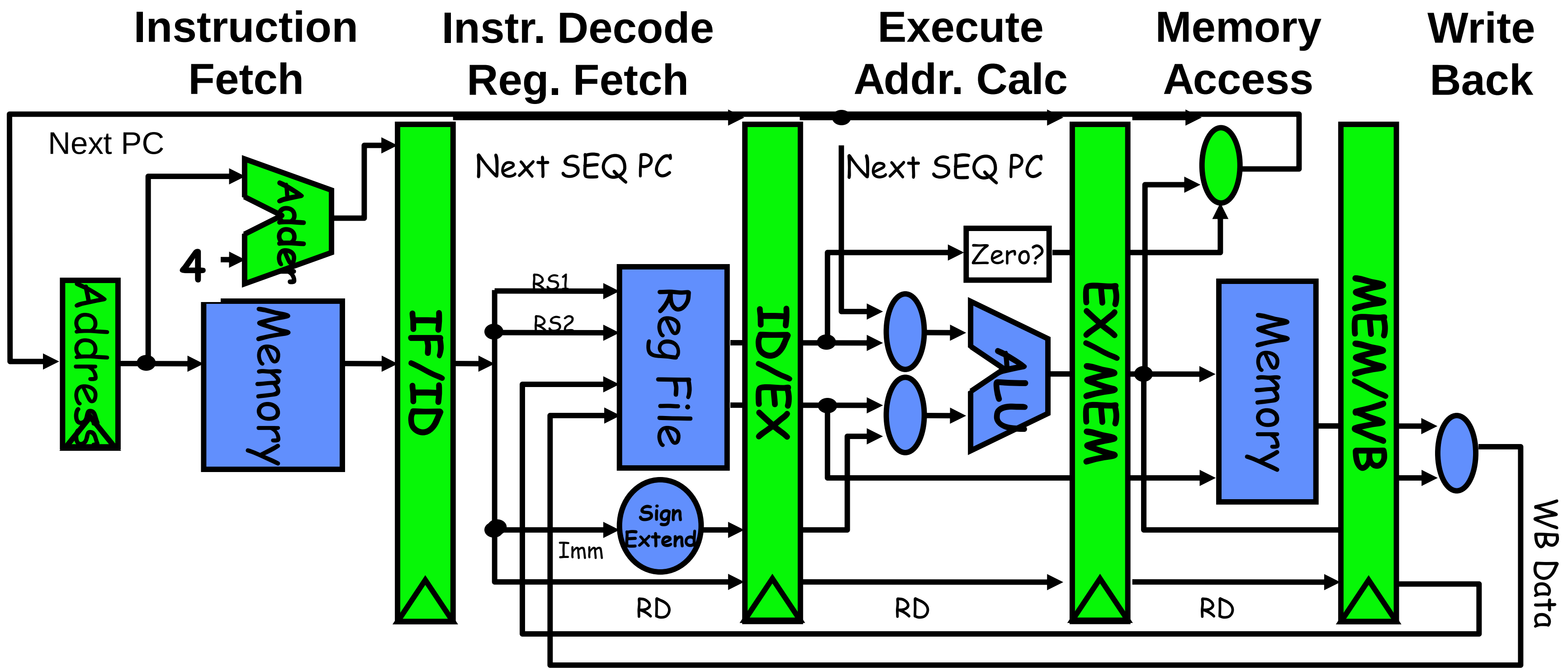
=> Three Stage Stall

```
10: beq r1,r3,36

14: and r2,r3,r5

18: or  r6,r1,r7

22: add r8,r1,r9

36: xor r10,r1,r11
```



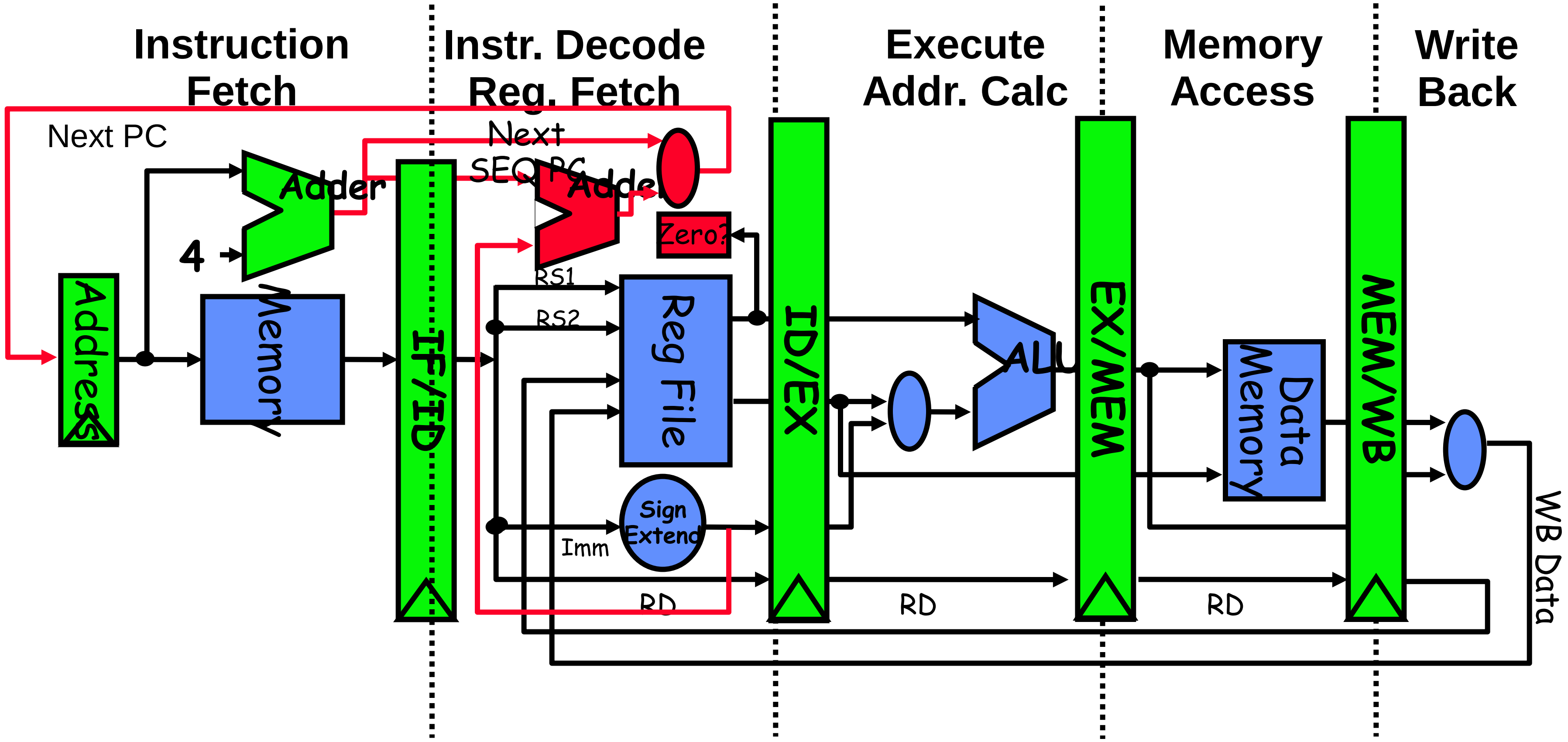
Conventional MIPS Pipeline

Branching at 4th stage



Branch Optimized MIPS Pipeline

Branching at 2nd stage



Four Branch Hazard Alternatives

#1: Stall until branch direction is clear

#2: Predict Branch Not Taken

#3: Predict Branch Taken

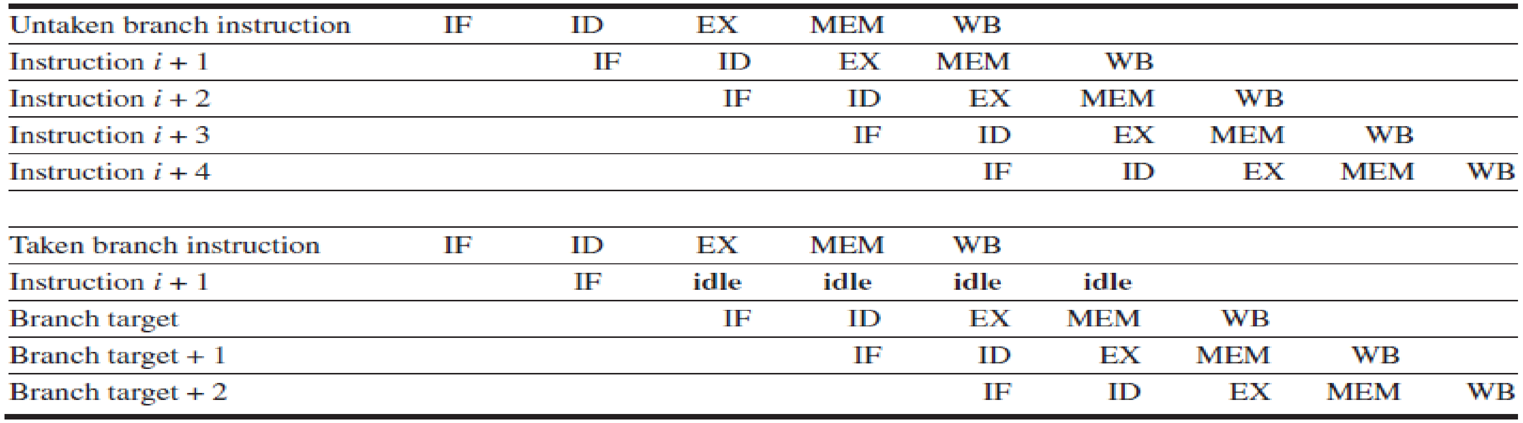
#4: Delayed Branch

Four Branch Hazard Alternatives

#1: Stall until branch direction is clear

#2: Predict Branch Not Taken

- ❖Execute successor instructions in sequence
- ❖“Squash” instructions in pipeline if branch actually taken



Untaken branch instruction	IF	ID	EX	MEM	WB				
Instruction $i + 1$		IF	ID	EX	MEM	WB			
Instruction $i + 2$			IF	ID	EX	MEM	WB		
Instruction $i + 3$				IF	ID	EX	MEM	WB	
Instruction $i + 4$					IF	ID	EX	MEM	WB

Taken branch instruction	IF	ID	EX	MEM	WB				
Instruction $i + 1$		IF	idle	idle	idle	idle			
Branch target			IF	ID	EX	MEM	WB		
Branch target + 1				IF	ID	EX	MEM	WB	
Branch target + 2					IF	ID	EX	MEM	WB

Four Branch Hazard Alternatives

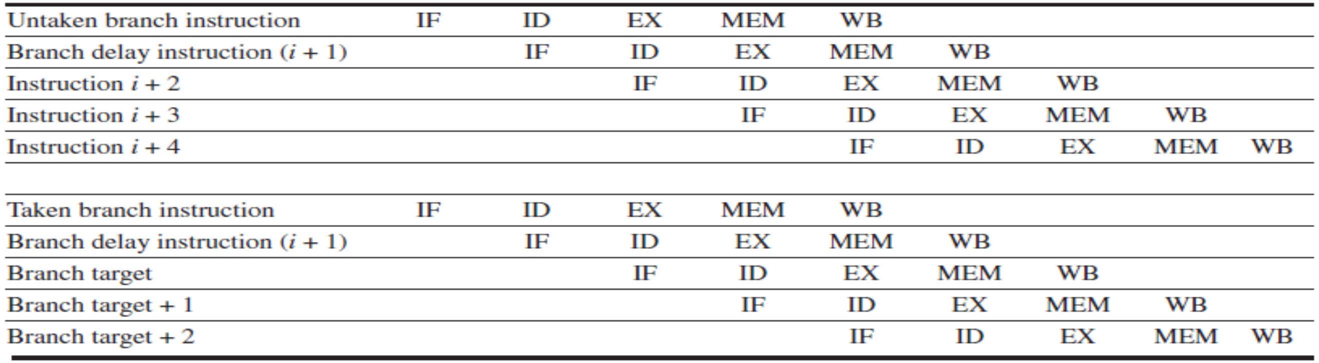
#3: Predict Branch Taken

- ❖But branch target address in is not known by IF stage
- ❖Target is known at same time as branch outcome (IDstage)
- ❖ MIPS still incurs 1 cycle branch penalty

Four Branch Hazard Alternatives

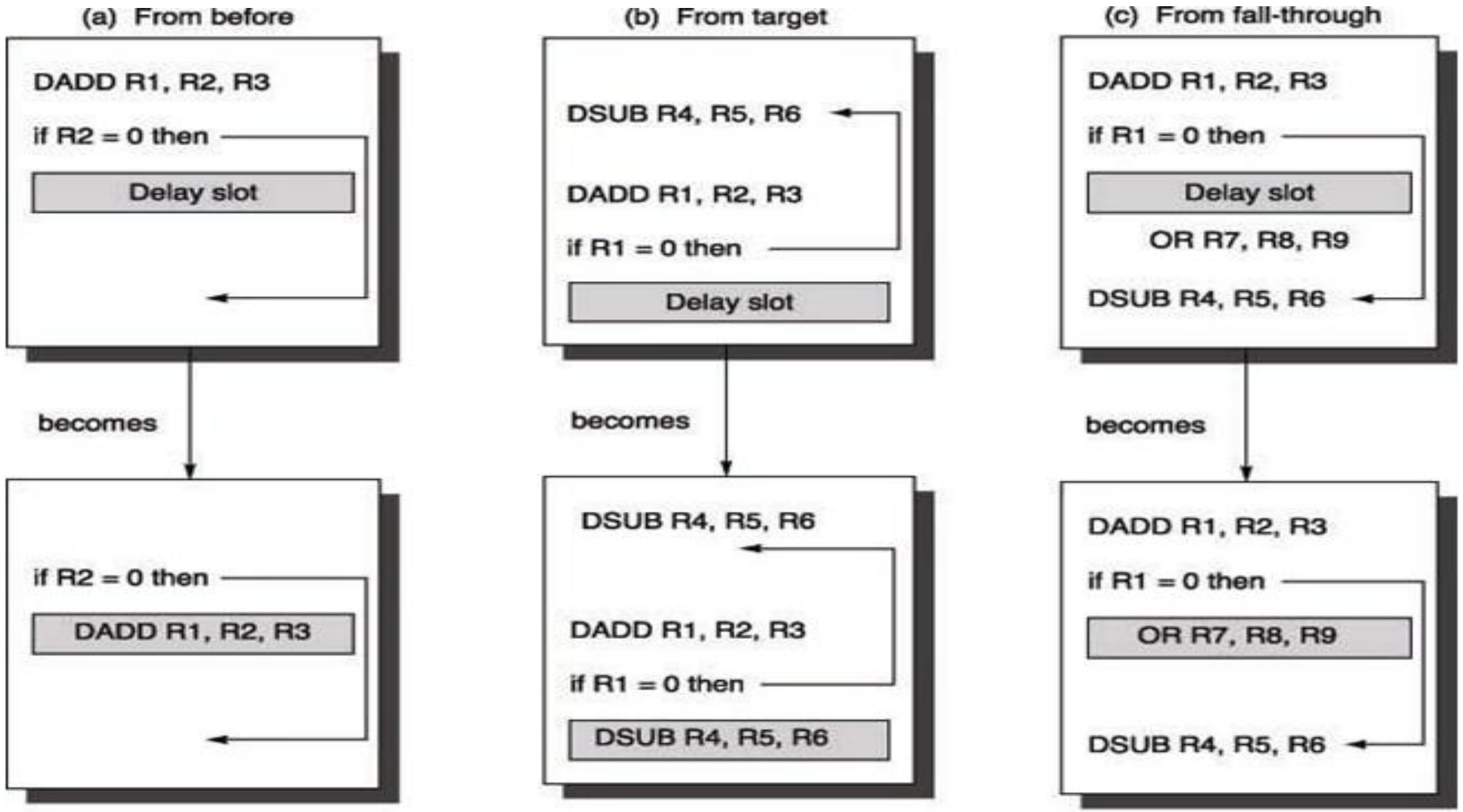
#4: Delayed Branch

- ❖Define branch to take place AFTER one instruction following the branch instruction.
- ❖ 1 slot delay allows proper decision and branch target address in 5 stage pipeline (MIPS uses this approach)
- ❖**Where to get instructions to fill branch delay slot?**



Untaken branch instruction	IF	ID	EX	MEM	WB				
Branch delay instruction ($i + 1$)		IF	ID	EX	MEM	WB			
Instruction $i + 2$			IF	ID	EX	MEM	WB		
Instruction $i + 3$				IF	ID	EX	MEM	WB	
Instruction $i + 4$					IF	ID	EX	MEM	WB
Taken branch instruction	IF	ID	EX	MEM	WB				
Branch delay instruction ($i + 1$)		IF	ID	EX	MEM	WB			
Branch target			IF	ID	EX	MEM	WB		
Branch target + 1				IF	ID	EX	MEM	WB	
Branch target + 2					IF	ID	EX	MEM	WB

Filling Branch Delay Slot



Conditional Branches

- When do you know you have a branch?
  - During ID cycle (Could you know before that?)
- When do you know if the branch is Taken or Not-Taken
  - During EXE cycle/ ID stage depending on the design
- We need for sophisticated solutions for following cases
  - Modern pipelines are deep (10 + stages)
  - Several instructions issued/cycle
  - Several predicted branches in-flight at the same time

Dynamic branch prediction

- Execution of a branch requires knowledge of:
  - Branch instruction - encode whether instruction is a branch or not. Decide on taken or not taken (i.e., prediction can be done at IF stage)
  - Whether the branch is Taken/Not-Taken (hence a branch prediction mechanism)
  - If the branch is taken what is the target address (can be computed but can also be “precomputed”, i.e., stored in some table)
  - If the branch is taken what is the instruction at the branch target address (saves the fetch cycle for that instruction)

Dynamic branch prediction

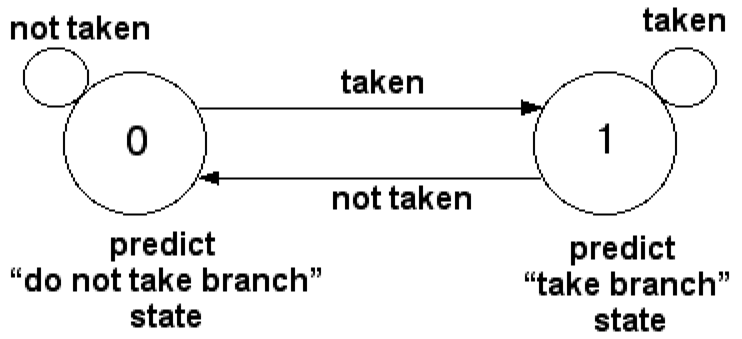
- Use a Branch Prediction Buffer (BPB)
  - Also called Branch Prediction Table (BPT), Branch History Table (BHT)
  - Records previous outcomes of the branch instruction.
  - How to index into the table is an issue.
- A prediction using BPB is attempted when the branch instruction is fetched (IF stage or equivalent)
- It is acted upon during ID stage (when we know we have a branch)

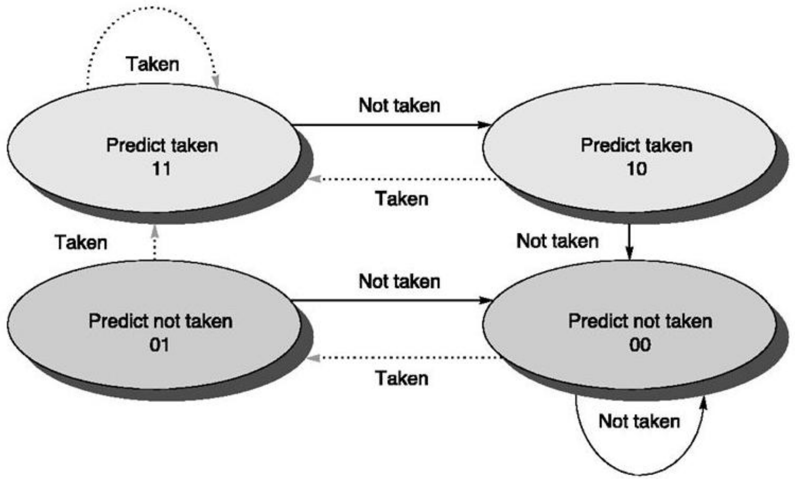
Dynamic branch prediction

- Has a prediction been made (Y/N)
  - If not use default “Not Taken”
- Is it correct or incorrect ?
- Two cases:
  - Case 1: Yes and the prediction was correct (known at ID stage) or No but the default was correct: No delay
  - Case 2: Yes and the prediction was incorrect or No and the default was incorrect: Delay

Prediction Scheme with 1 or 2 bit FSM

- ❖ The use of a 2-bit predictor will allow branches that favor taken (or not taken) to be mispredicted less often than the one-bit case. (reinforcement learning)





Branch Prediction In Hardware

- Branch prediction is extremely useful in loops.
- A simple branch prediction can be implemented using a small amount of memory indexed by lower order bits of the address of the branch instruction. **(branch prediction buffer)**
- One bit stores whether the branch was taken or not.
- The next time the branch instruction is fetched refer this bit

Advanced Branch Prediction Techniques

- **Basic 2-bit predictor:**
  - For each branch:- Predict T or NT
  - If the prediction is wrong for two consecutive times, change prediction
- **Correlating predictor:**
  - Multiple 2-bit predictors for each branch
  - One for each possible combination of outcomes of preceding *n* branches

```
if ( x = = 2)      /* br-1*/
      x = 0;
if ( y = = 2)      /* br-2*/
      y = 0;
if ( x != y)       /* br-3/
          do this
else do that
```

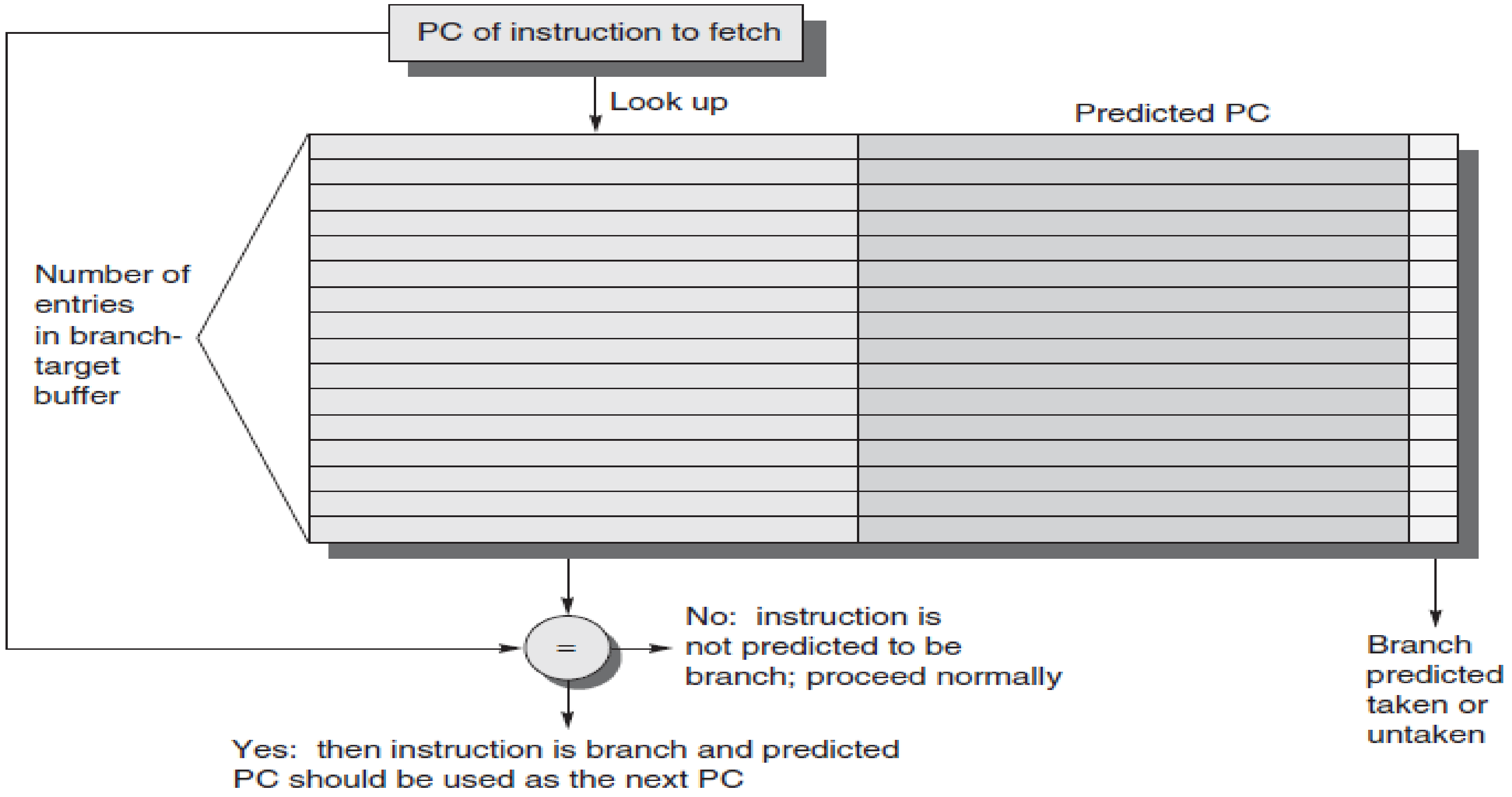
Advanced Branch Prediction Techniques

- **Local predictor:**
  - Multiple 2-bit predictors for each branch
  - One for each possible combination of outcomes for the last *n* occurrences of this branch
- **Tournament predictor:**
  - Combine correlating predictor with local predictor

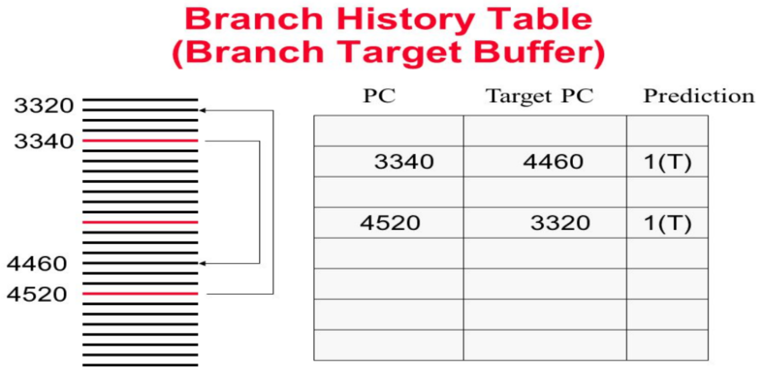
Branch-Target Buffer

- ❖ To reduce the branch penalty, know whether the as-yet-un-decoded instruction is a branch. If so, what the next program counter (PC) should be.
- ❖ If the instruction is a branch and we know what the next PC should be, we can have a branch penalty of zero.
- ❖ A branch-prediction cache that stores the predicted address for the next instruction after a branch is called a **branch-target buffer (BTB)** or **branch-target cache.**

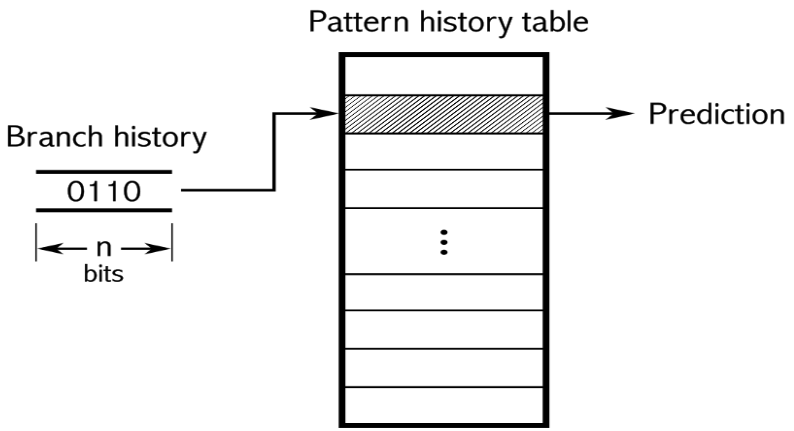
Branch-Target Buffer

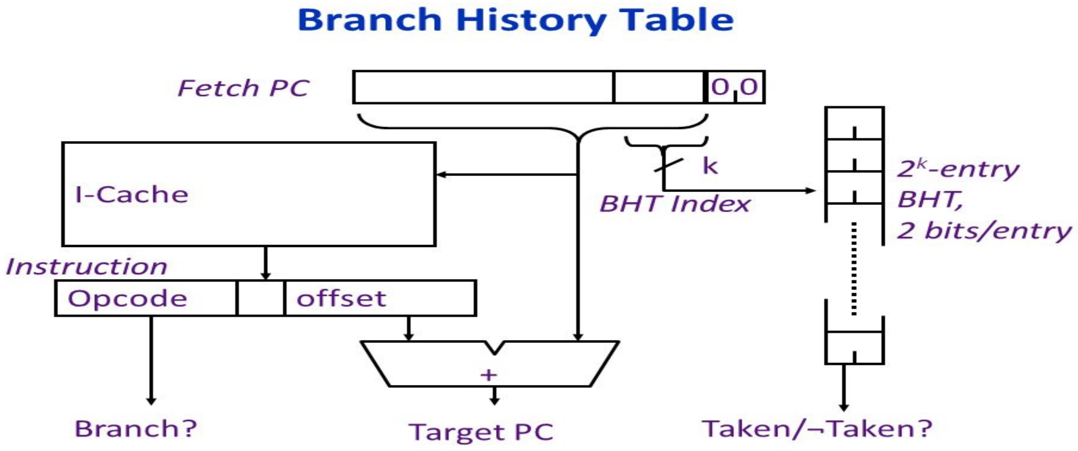

PC of instruction to fetch
Look up
Predicted PC
Number of entries in branch-target buffer
=
No: instruction is not predicted to be branch; proceed normally
Yes: then instruction is branch and predicted PC should be used as the next PC
Branch predicted taken or untaken

Branch-Target Buffer



PC	Target PC	Prediction
3340	4460	1(T)
4520	3320	1(T)





4K-entry BHT, 2 bits/entry, ~80-90% correct predictions

Branch Folding

- Optimization on BTB to make zero cycle branch
  - Larger branch-target buffer- store one or more target instructions
  - Add target instruction into BTB to deal with longer decoding time required by larger buffer
  - Branch folding can be used to obtain 0-cycle unconditional branches and sometimes 0-cycle conditional branches.

johnjose@iitg.ac.in

http://www.iitg.ac.in/johnjose/