CS223 : Computer Architecture & Organization

Lecture 27 [08.04.2022]

Control Hazards & Branch Prediction

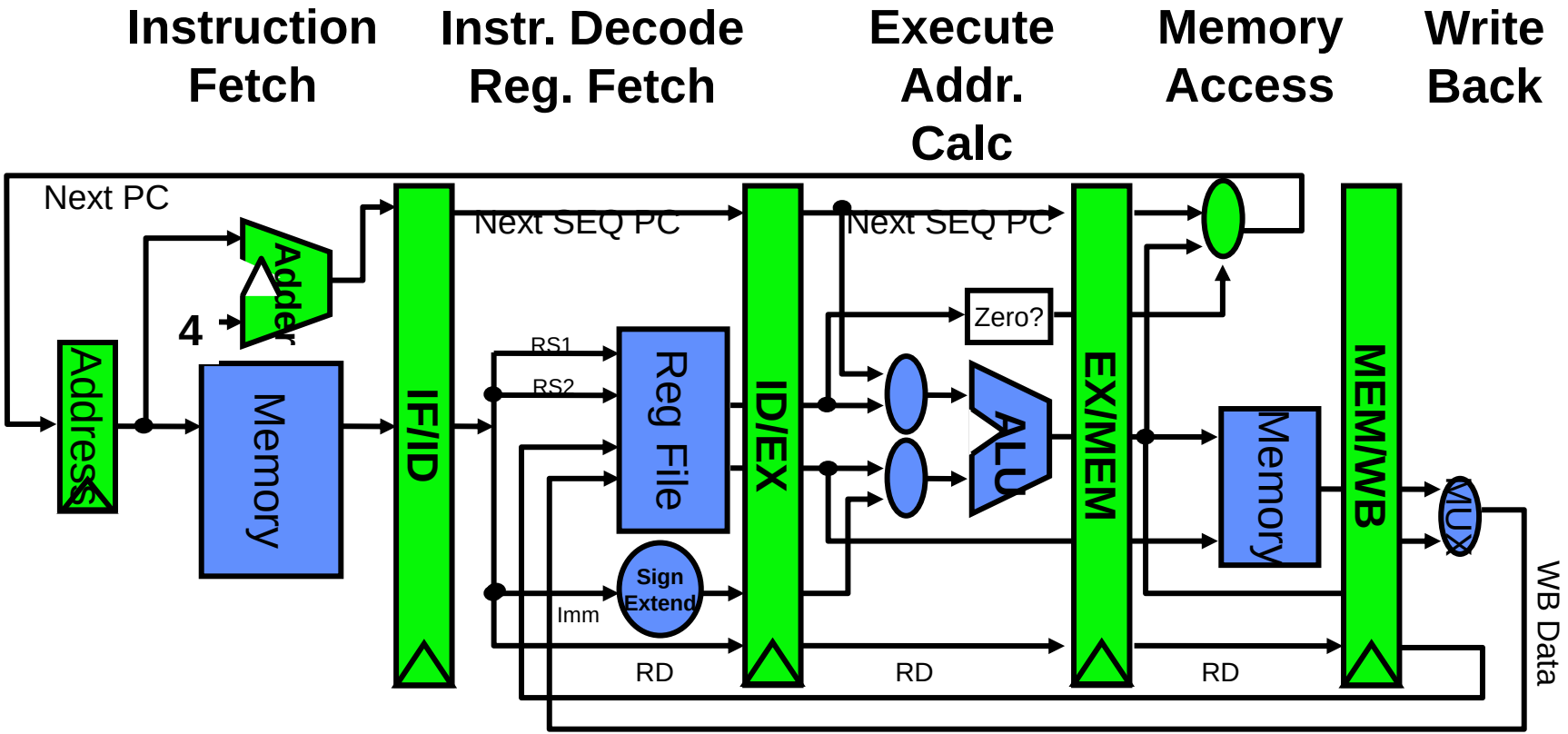
Dr. John Jose

Associate Professor

Department of Computer Science & Engineering

Indian Institute of Technology Guwahati, Assam.

Pipelined RISC Data path



Visualizing Pipelining

	Clock number							
Instruction number	1	2	3	4	5	6	7	8
i	IF	ID	EX	MEM	WB			
i+1		IF	ID	EX	MEM	WB		
i+2			IF	ID	EX	MEM	WB	
i+3				IF	ID	EX	MEM	WB
i+4					IF	ID	EX	MEM

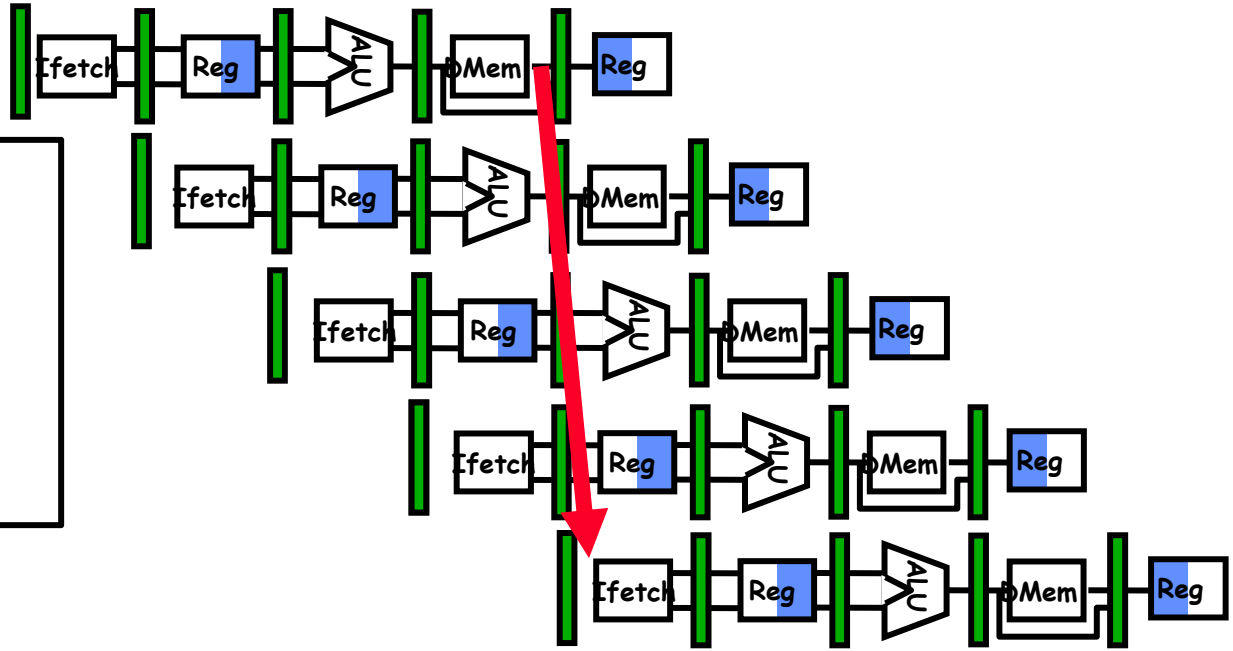
Limits to pipelining

- **Hazards:** circumstances that would cause incorrect execution if next instruction is fetched and executed
  - **Structural hazards:** Different instructions, at different stages, in the pipeline want to use the same hardware resource
  - **Data hazards:** An instruction in the pipeline requires data to be computed by a previous instruction still in the pipeline
  - **Control hazards:** Succeeding instruction, to put into pipeline, depends on the outcome of a previous branch instruction, already in pipeline

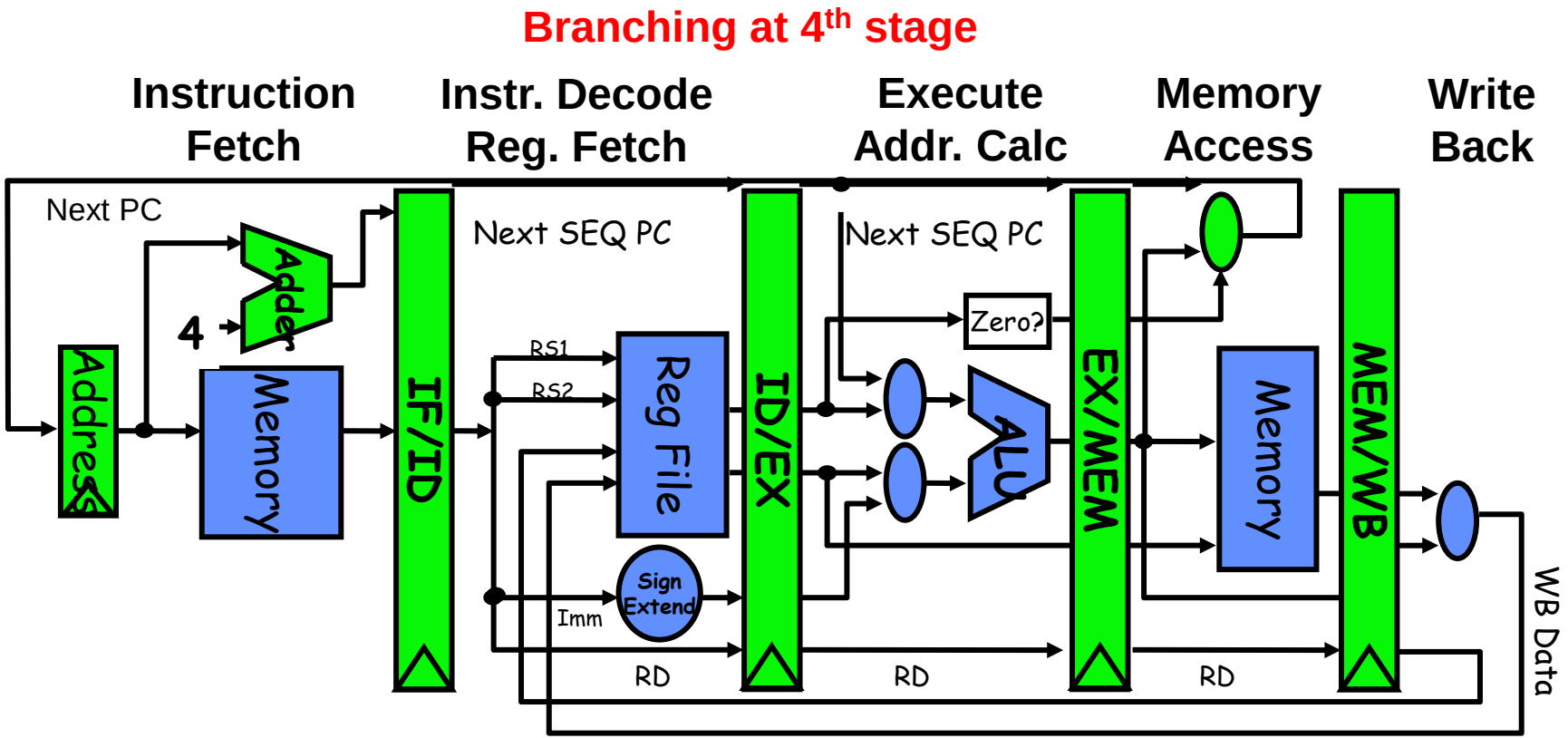
Control Hazard on Branches

=> Three Stage Stall

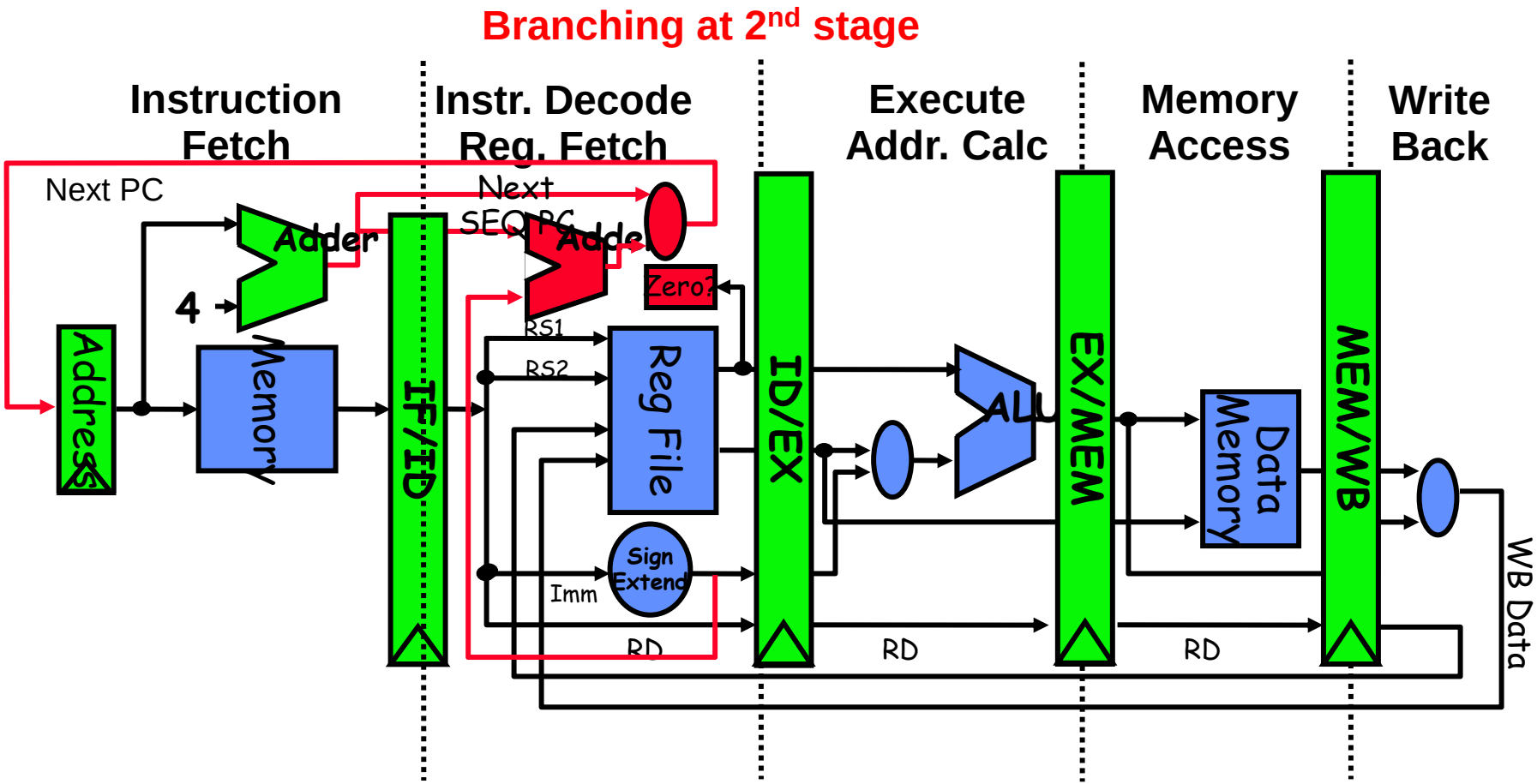
```
10: beq r1,r3,36
14: and r2,r3,r5
18: or  r6,r1,r7
22: add r8,r1,r9
36: xor r10,r1,r11
```



Conventional MIPS Pipeline



Branch Optimized MIPS Pipeline



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

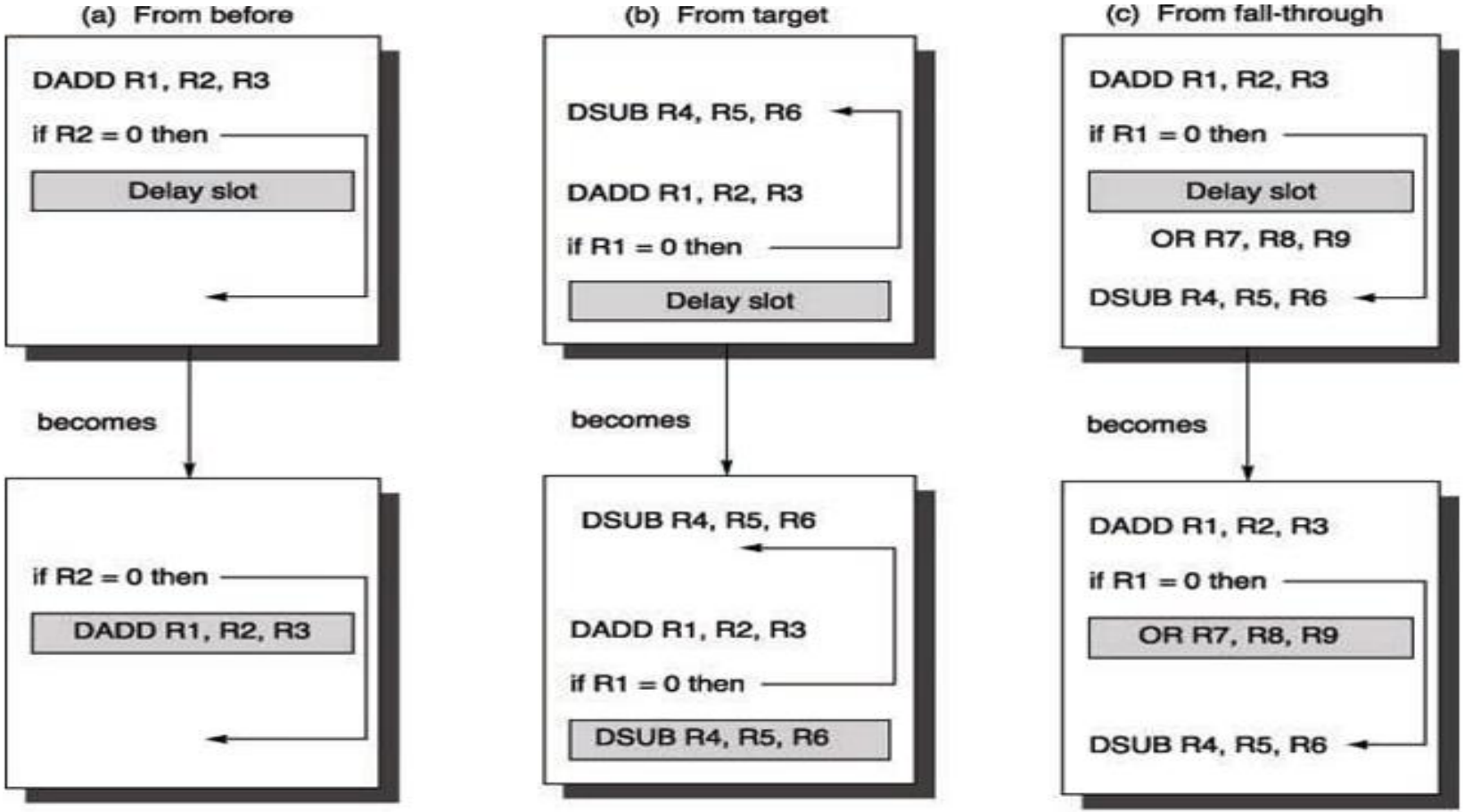
#4: Delayed Branch

- 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 sophisticated solutions for following cases
  - Modern pipelines are deep (10 + stages)
  - Several instructions issued/cycle
  - Several predicted branches in-flight at the same time

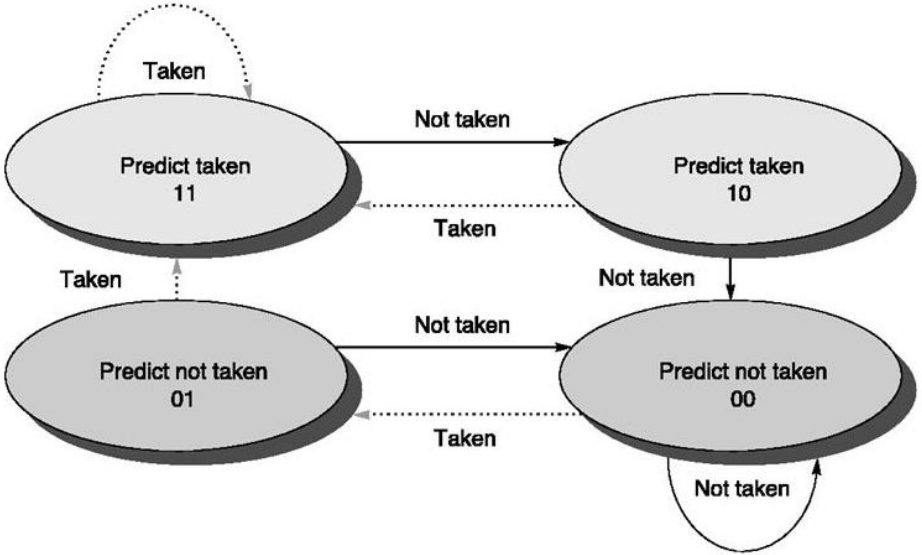
Dynamic branch prediction

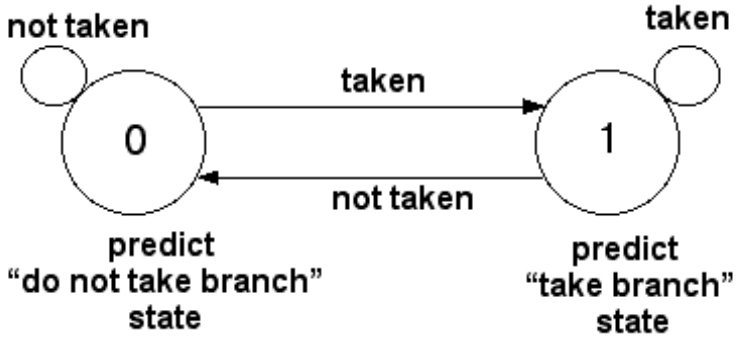
- Execution of a branch requires knowledge of:
  - Branch instruction - encode whether instruction is a branch or not.
  - Decide on taken or not taken (at IF stage)
  - Whether the branch is actually Taken/Not-Taken? (at ID/EX stage)
  - If the branch is taken what is the target address (can be computed 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.
- 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)

Basic Predictors using FSM





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 OR/AND previous outcomes of the branch.
- Have an FSM based predictor in each entry.

Advanced Branch Prediction Techniques

- **Correlating predictor:**
  - Multiple 2-bit predictors for each branch
  - One for each possible combination of outcomes of preceding *n* branches
- **Tournament predictor:**
  - Combine correlating predictor with other local predictor and pick one dynamically

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

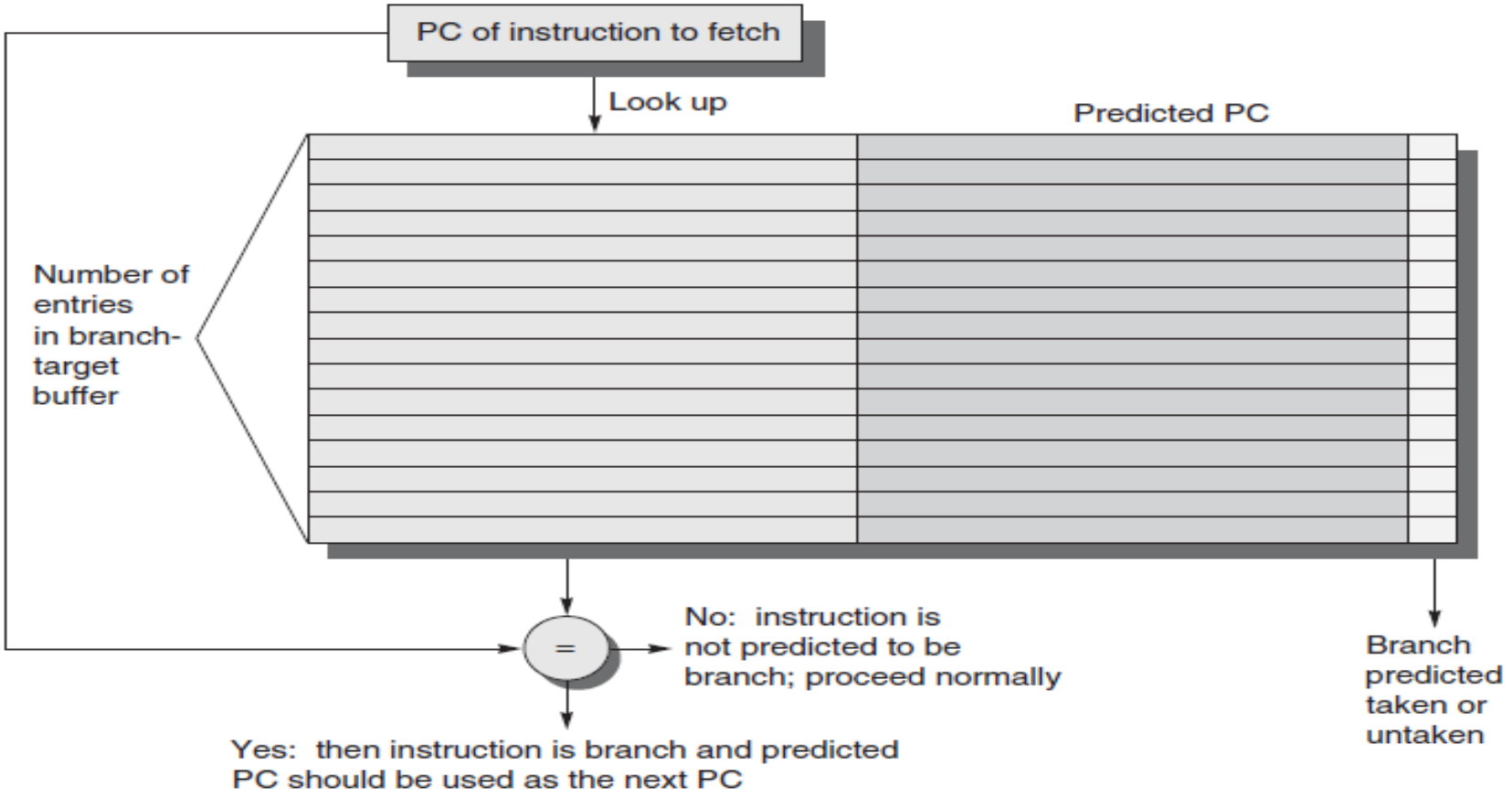
Branch-Target Buffer

- 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.

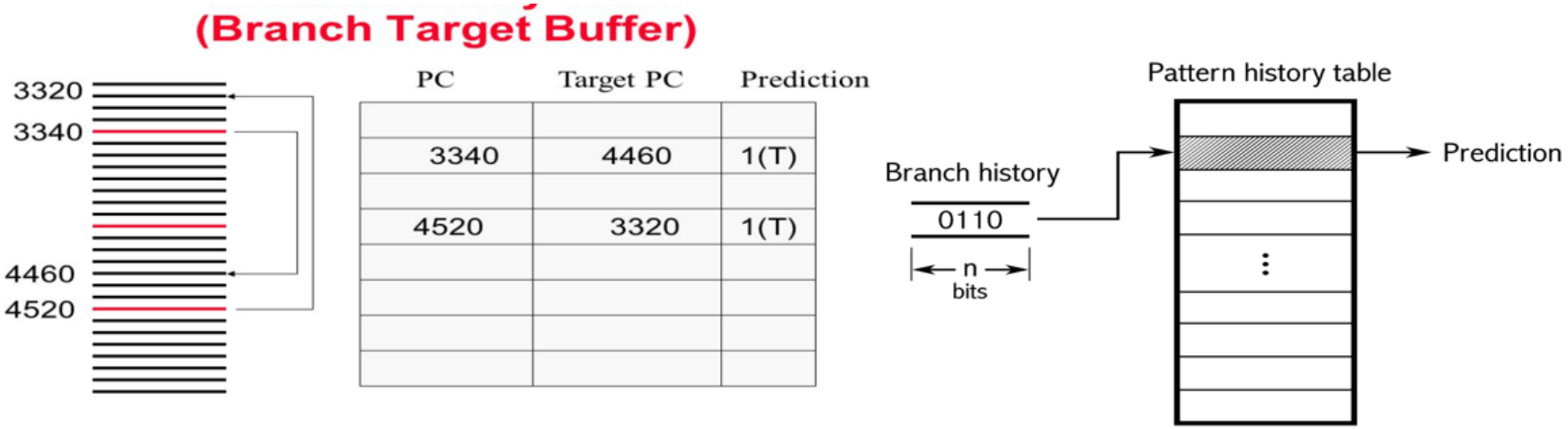
- If the instruction is a branch and we know what the next PC should be, we can have a branch penalty of zero.

- 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



Branch-Target Buffer



Branch Folding

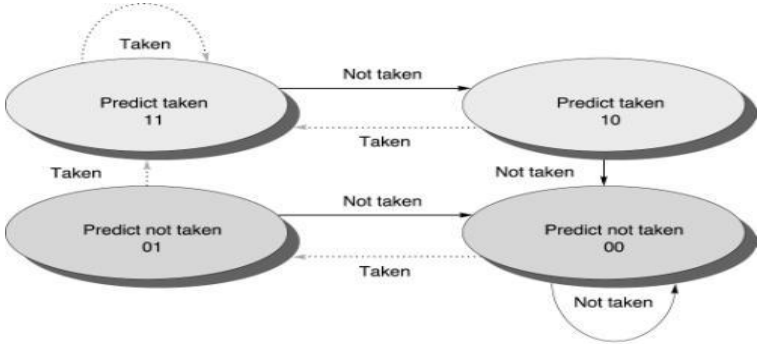
- Optimization on BTB to make zero cycle branch
  - Larger branch-target buffer- store one or more target instructions
  - Add target instruction into BTB to deal with longer decoding time required by larger buffer
  - Branch folding can be used to obtain 0-cycle unconditional branches and sometimes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

Branch Prediction- Example 2

Consider the last 16 actual outcomes of a single static branch. T means branch is taken and N means not taken.

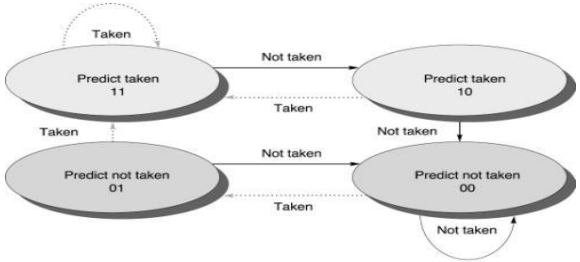
{oldest→ T T N N T N T T T N T N T T N T ← latest}

A two level branch predictor of (1,2) type is used. Since there is only one branch in the program indexing to BHT with PC is irrelevant. Hence only last branch outcome only is used to index to the BHT. How many mis-predictions are there and which of the branches in this sequence would be mis-predicted? Fill up the table for 16 branch outcomes.



Branch Prediction

{oldest→ T T N N T N T T T N T N T T N T ← latest}
A two level branch predictor of (1,2) type is used.



Sl. No	Last Outcome	BHT N/T	Prediction	Outcome
1	N (Initial)	00/11	N	

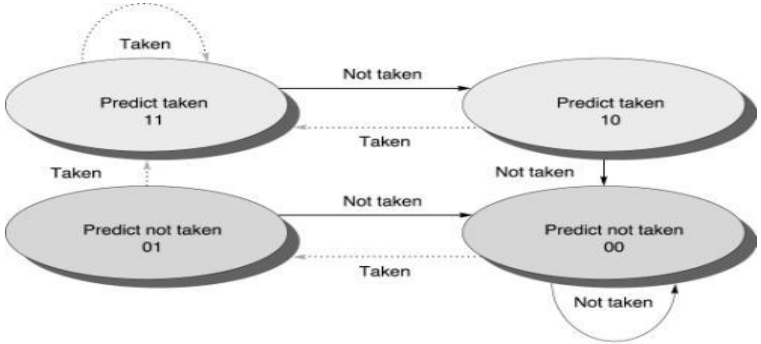
Branch Prediction Illustration

Sl.No	Last Outcome	BHT N/T	Prediction	Outcome	Mis-Pre Y/N ?
1	N (initial)	00 / 11	N	T	YES
2	T	01 / 11	T	T	NO
3	T	01 / 11	T	N	YES
4	N	01 / 10	N	N	NO
5	N	00 / 10	N	T	YES
6	T	01 / 10	T	N	YES
7	N	01 / 00	N	T	YES
8	T	11 / 00	N	T	YES
9	T	11 / 01	N	T	YES
10	T	11 / 11	T	N	YES
11	N	11 / 10	T	T	NO
12	T	11 / 10	T	N	YES
13	N	11 / 00	T	T	NO
14	T	11 / 00	N	T	YES
15	T	11 / 01	N	N	NO
16	N	11 / 00	T	T	NO

T
T
N
N
T
N
T
T
T
N
T
N
T
T
N
T

Branch Prediction – Example 2

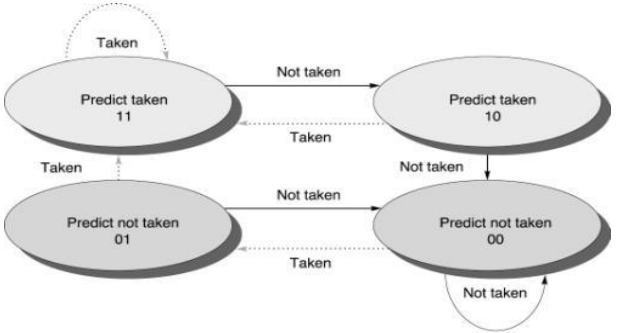
Consider a (2,2) type branch predictor. BHT is indexed by the outcome of the last 2 branches. The BPB is initialized for NN/NT/TN/TT as 00/00/11/11 and is indexed with an NN entry in the first reference. Consider the last 8 actual outcomes of a single static branch, {oldest N N T T T N N T latest} where T means branch is taken and N means not taken. What will be the contents of BPB after the execution of the above mentioned 6 branch outcomes?



Branch Prediction – Illustration

(2,2) type branch predictor. Initial: NN/NT/TN/TT - 00/00/11/11
Last 8 actual outcomes, {oldest N N T T T N N T latest}

S. No.	Last Outcome	BPB NN/NT/TN/TT	Prediction	Outcome	Misprediction Y/N?
1.	NN (Initial)	00/00/11/11	N	N	No
2.	NN	00/00/11/11	N	N	No
3.	NN	00/00/11/11	N	T	Yes
4.	NT	01/00/11/11	N	T	Yes
5.	TT	01/01/11/11	T	T	No
6.	TT	01/01/11/11	T	N	Yes
7.	TN	01/01/11/10	T	N	Yes
8.	NN	01/01/10/10	N	T	Yes
		11/01/10/10			



Reference

- **Computer Architecture-A Quantitative Approach** (5th edition), John L. Hennessy, David A. Patterson, Morgan Kaufman.
- Appendix C: **Pipelining: Basic and Intermediate Concepts**
  - Section C2: **The Major Hurdle of Pipelining—Branch Hazards**
  - Section C3: **How is pipelining implemented?**

- **NPTEL Video Links:**
  - https://tinyurl.com/yco9ge36
  - https://tinyurl.com/yd64ey53

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