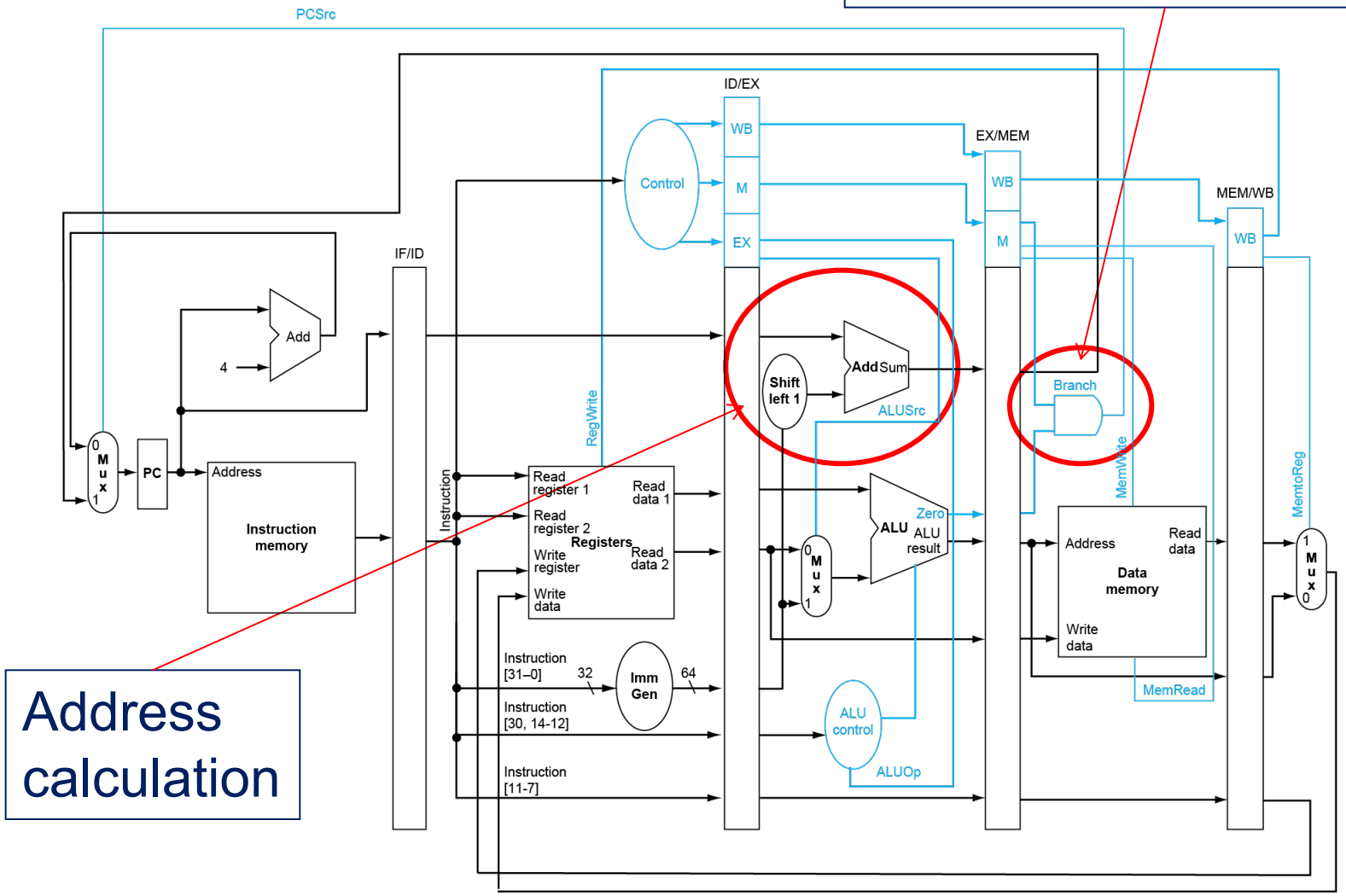
Topic 8

Control Hazards

Control (Branch) Hazards

- Current implementation

Determination for branch in MEM stage



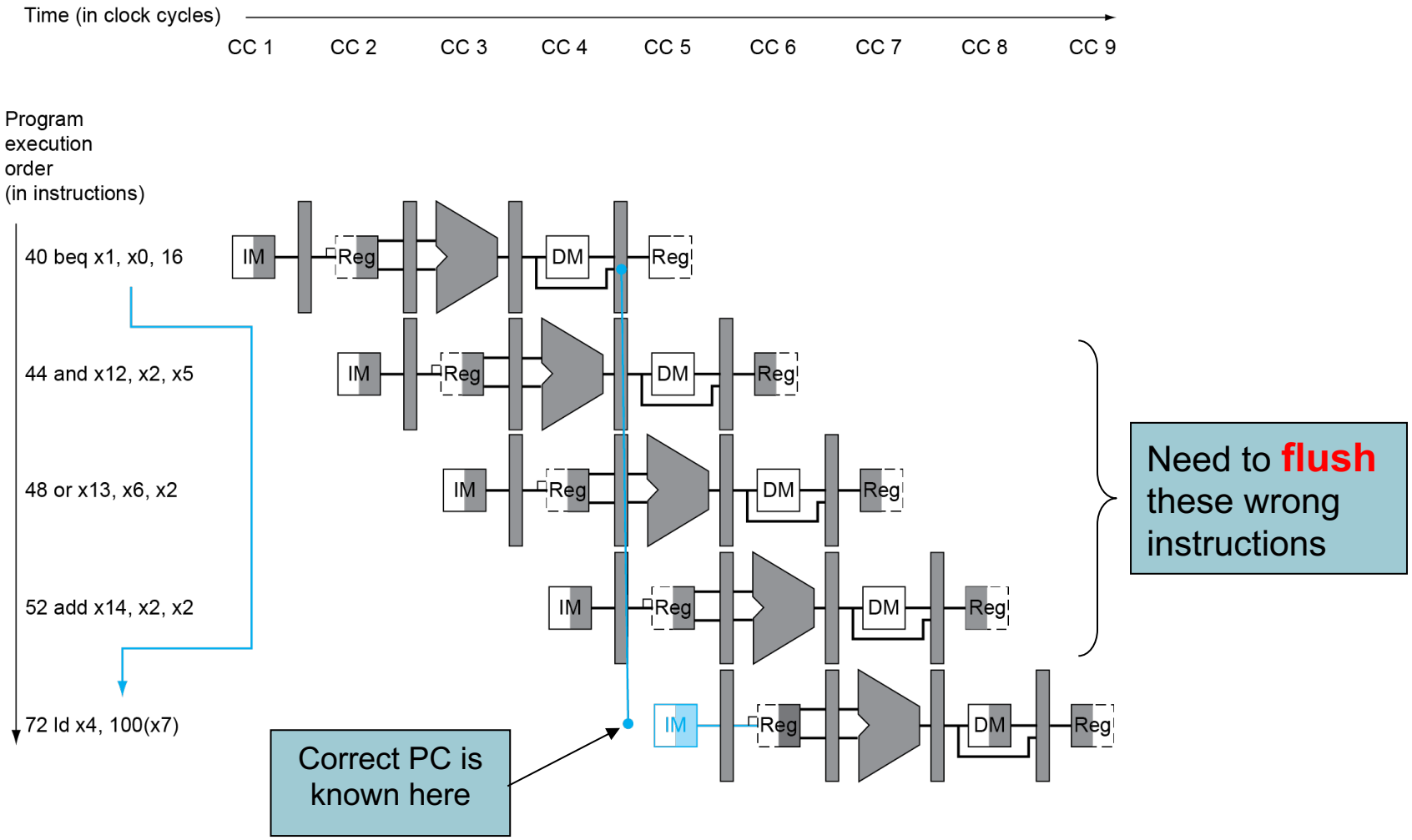
Address calculation

Control Hazards

- Branch determines flow of control
  - Fetching next instruction depends on branch related operations
    - Address calculation in EX
    - Branch decision in MEM
  - Pipeline can’t always fetch correct instruction
    - When we need to know which is the next instruction (PC+4 or branch target), branch is still in the IF stage

Branch Hazards

- Branch outcome determined in MEM

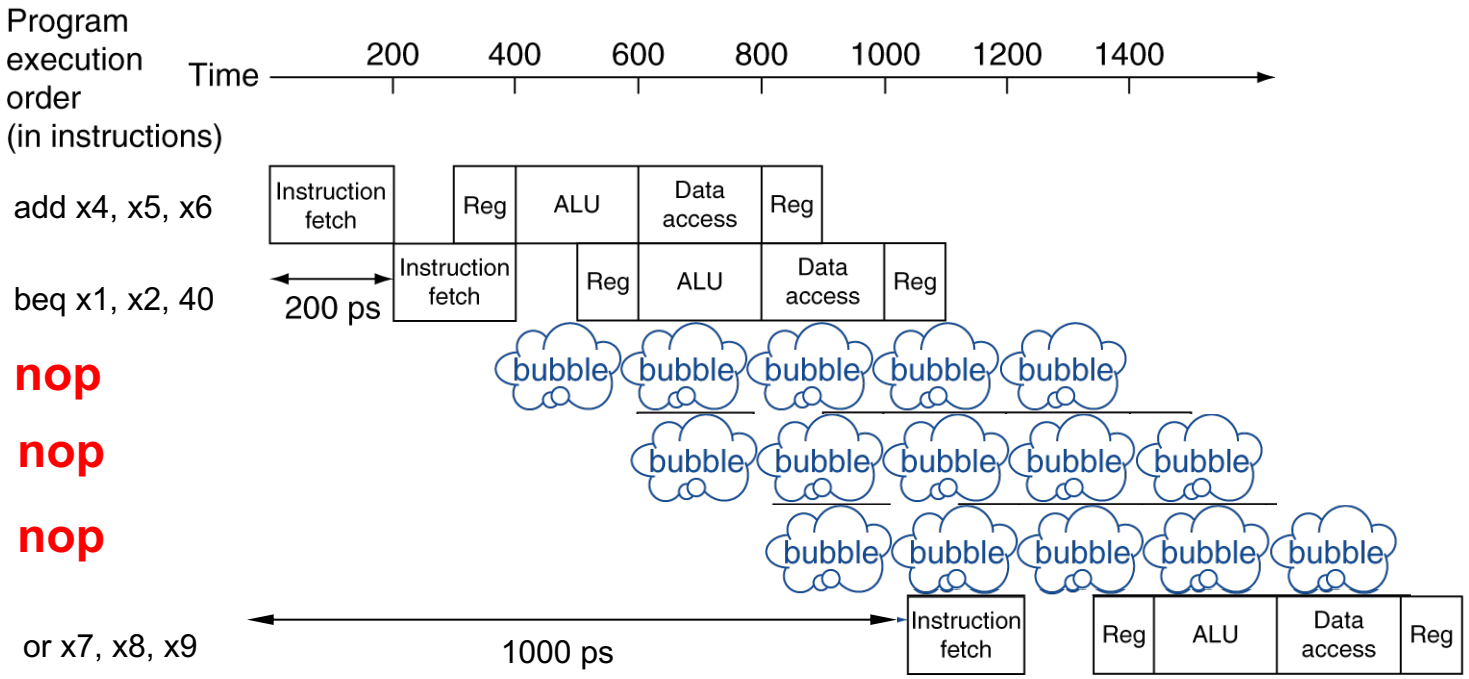


Resolutions to Branch Hazard

- *Stall on branch*
- Always assume branch not taken
- Branch prediction

Stall on Branch

- Wait until branch outcome is determined before fetching the next instruction



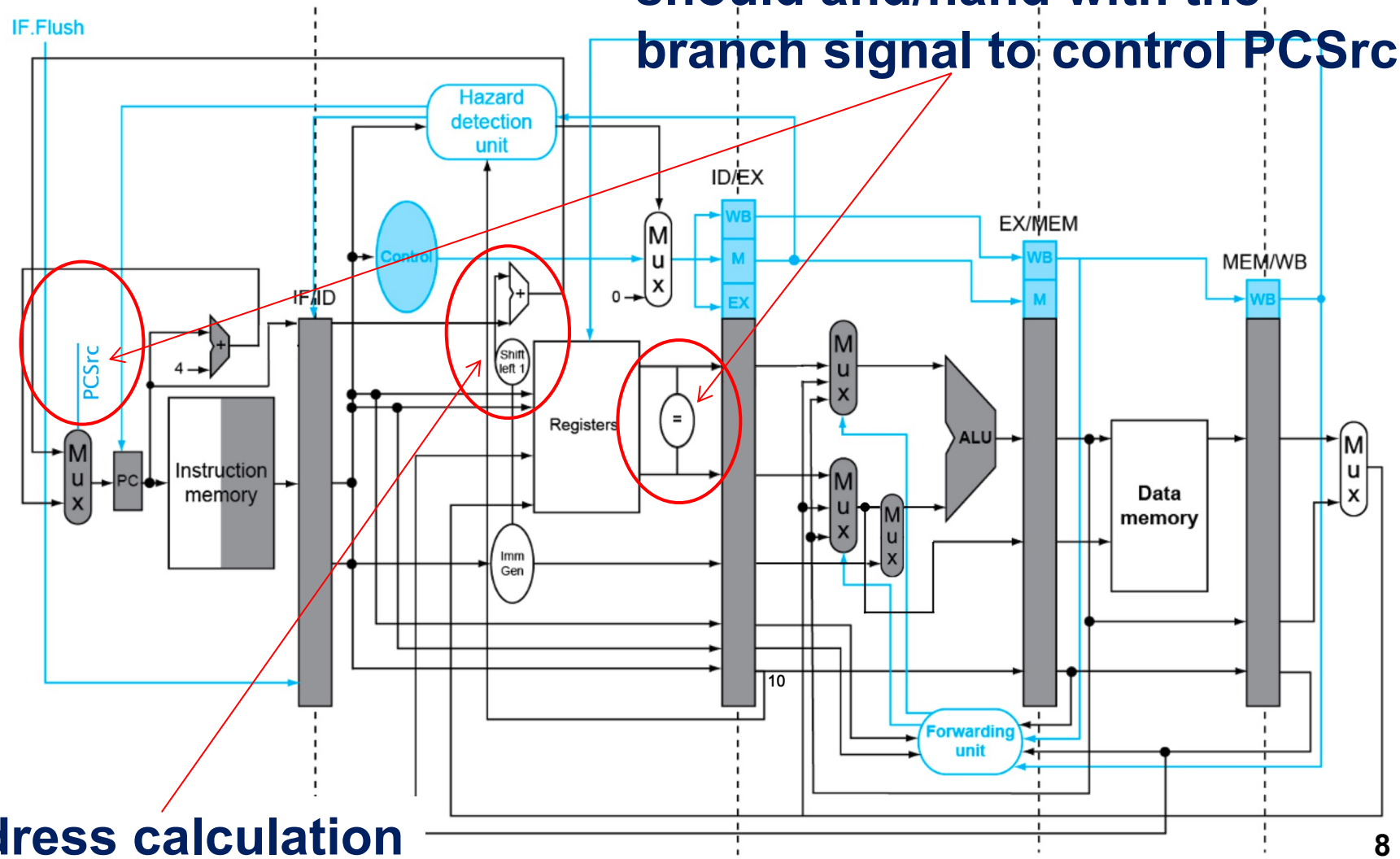
Higher IC, longer time, worse performance

Reducing Branch Delay

- By moving operations and associated components for determining branch outcome to the **ID** stage, including
  - Target address adder
  - Register comparator
  - Branch logic (the and gate)

Change the Hardware

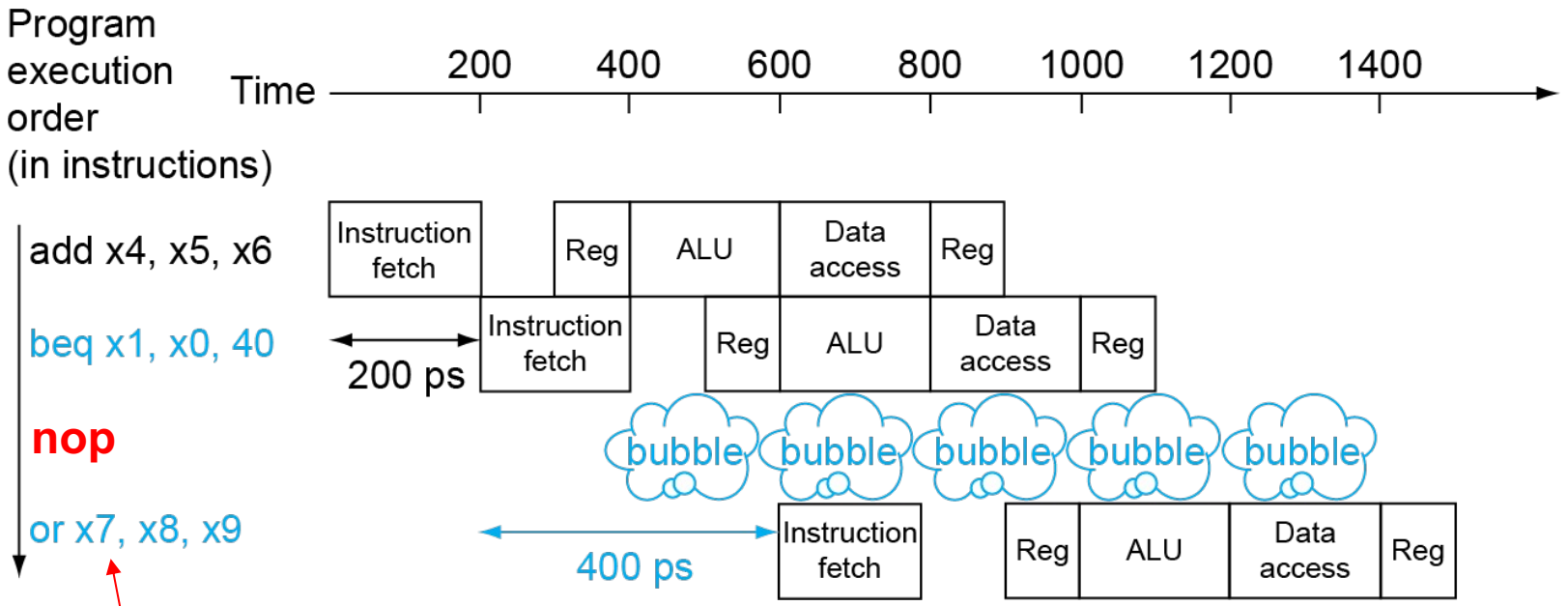
Register comparison, output should and/nand with the branch signal to control PCSrc



Address calculation

Stall on Branch

- Insert only one bubble after moving the branch decision making to ID stage



This instruction might be the one following beq or the one at branch target depending on the comparison result

Resolutions to Branch Hazard

- Stall on branch
- *Always assume branch not taken*
- Branch prediction

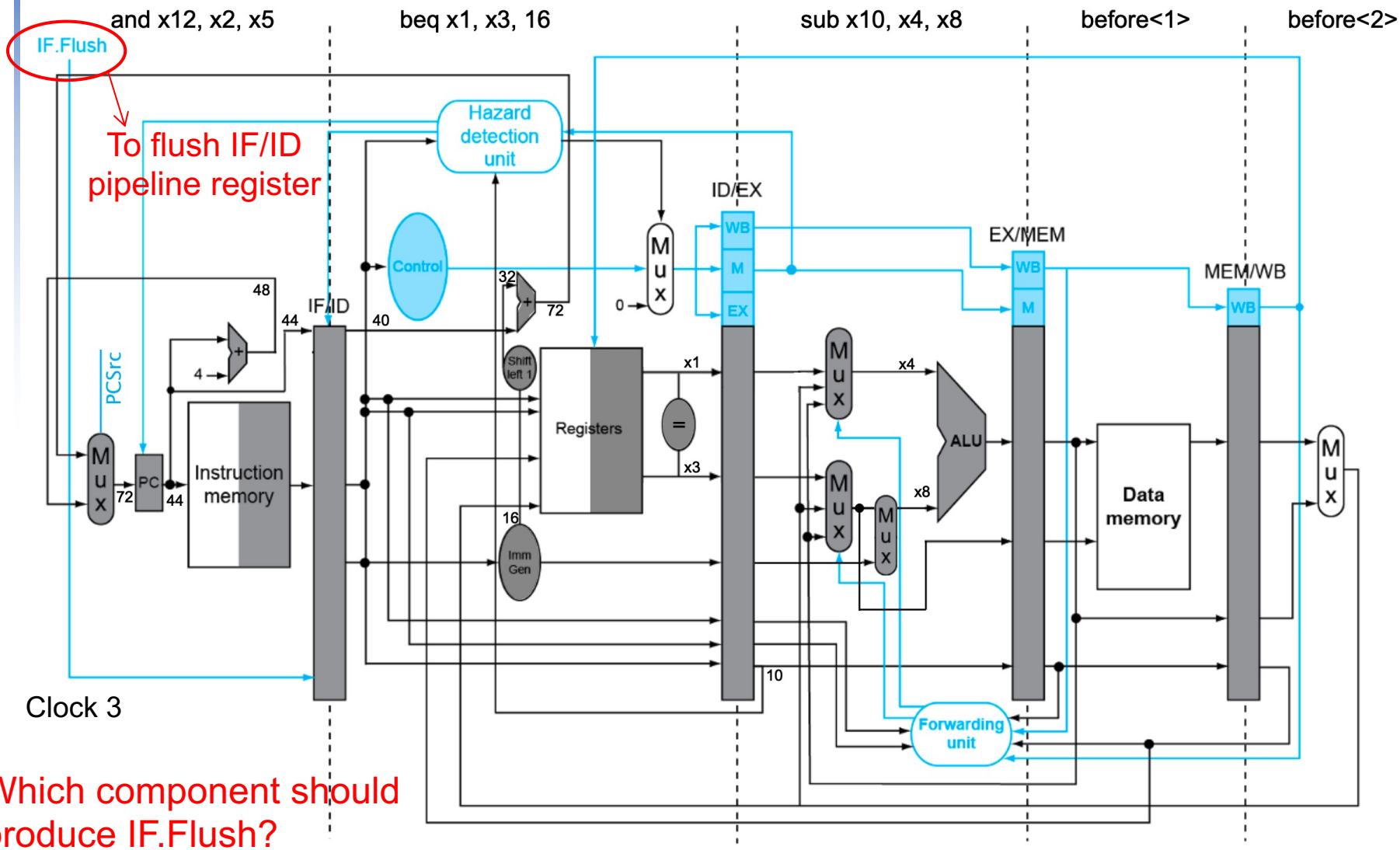
Assume Branch Not Taken

- If we are right, lucky us!
- If we are wrong, penalty will be to **flush** one (previously, three) instructions

Flush an Instruction

- If a branch should have been taken but not, then assumed wrong, then we need to flush the wrongly fetched instruction
- Flush: to discard the wrong instruction in pipeline, equivalent to neutralizing all operations
  - Clear IF/ID pipeline register, by a new control signal **IF.Flush**
    - Flushes the instruction in IF stage
  - All 0 instruction produces all 0 control signals

IF.Flush (synchronous)	Branch	"=" Output	PCSrc	PCin	PCout	IF/ID Register
1	1	1	1	72	44	beq & PC



Clock 3

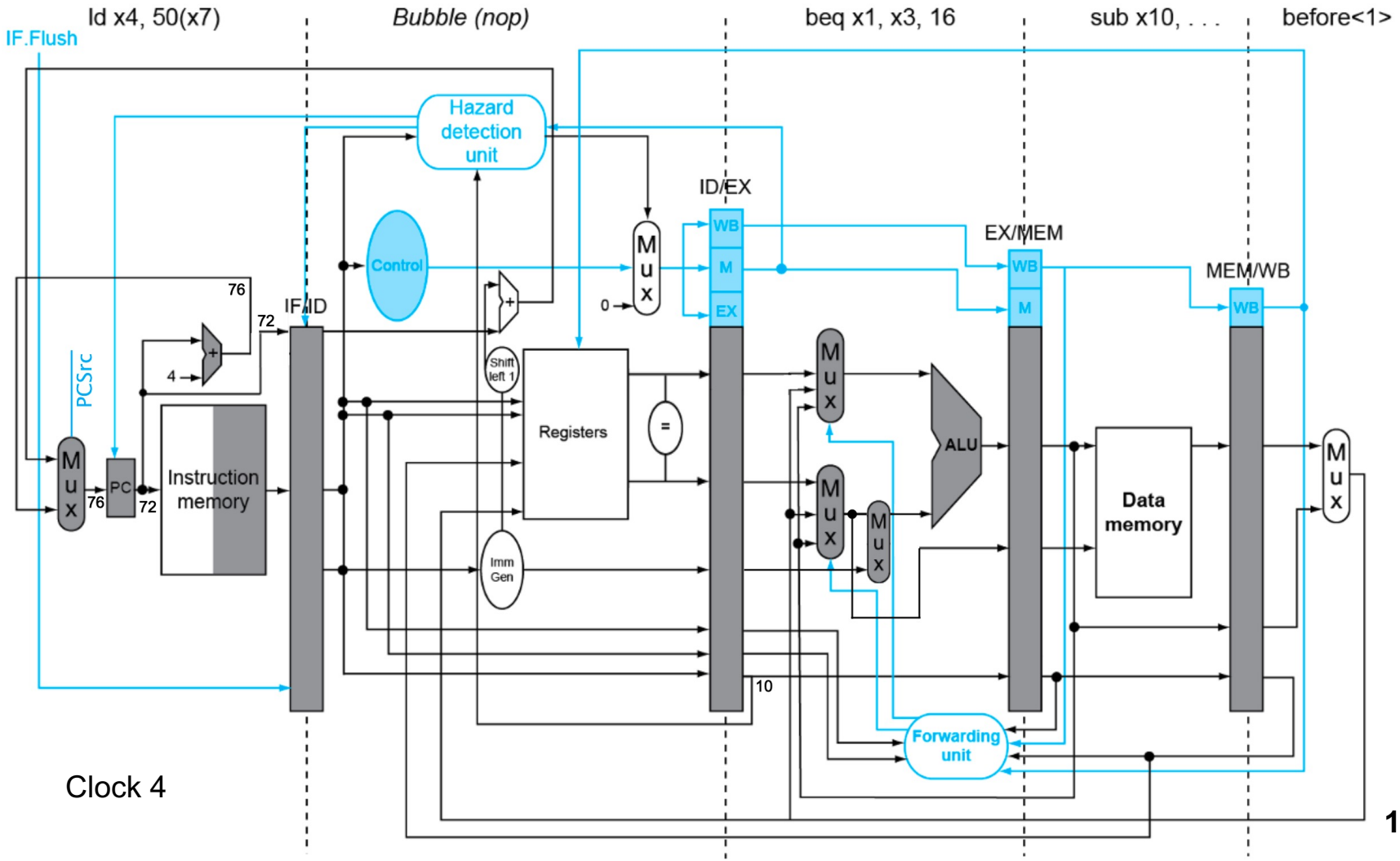
Which component should produce IF.Flush?

IF.Flush (synchronous)	Branch	"=" Output	PCSrc	IF/ID Register
0	0	1	0	0

Branch target instruction

"and" instruction flushed

Branch condition is true

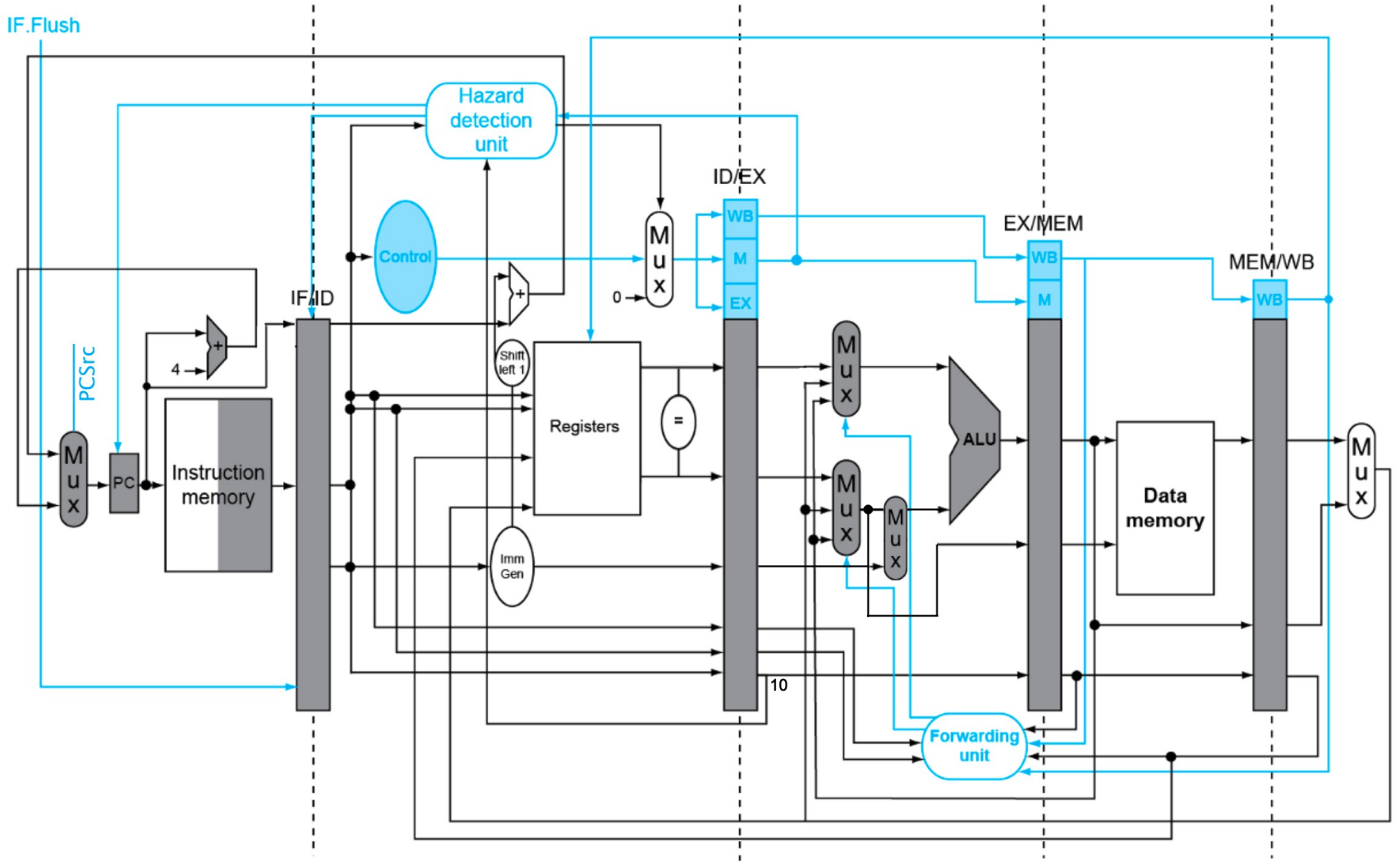


Clock 4

Should we be concerned by the hardware change?

Yes, timing Issue!

Potential Impact to Performance



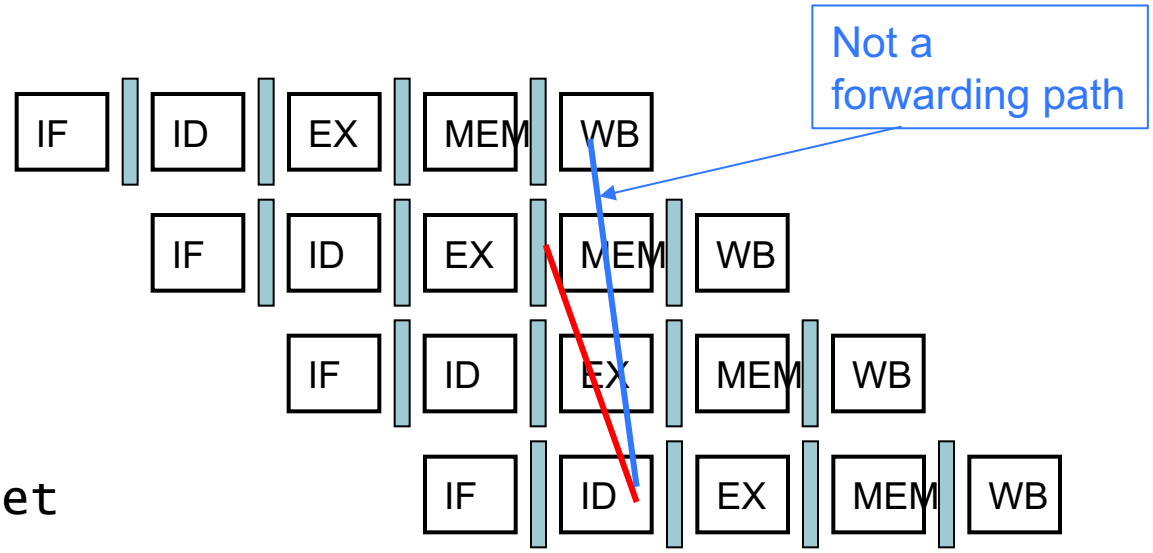
Should we be concerned by the hardware change?

Yes, new data hazards!

Data Hazards for Branches

- If a comparison register is a destination of 2nd preceding ALU instruction

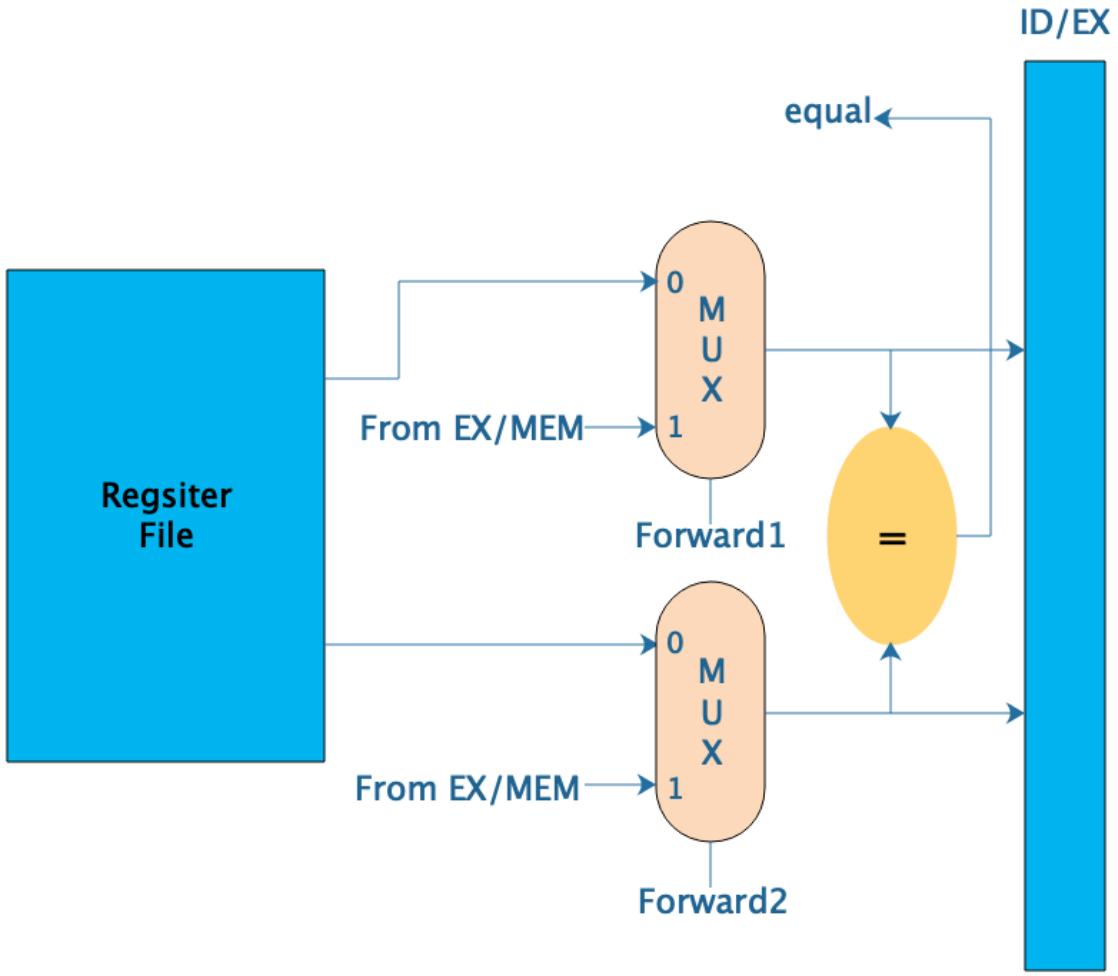
```
add x1, x2, x3
add x4, x5, x6
…
beq x1, x4, target
```



- Can resolve using new forwarding path

Forwarding Paths

- Forwarding paths are created between stage pipeline register and comparator inputs

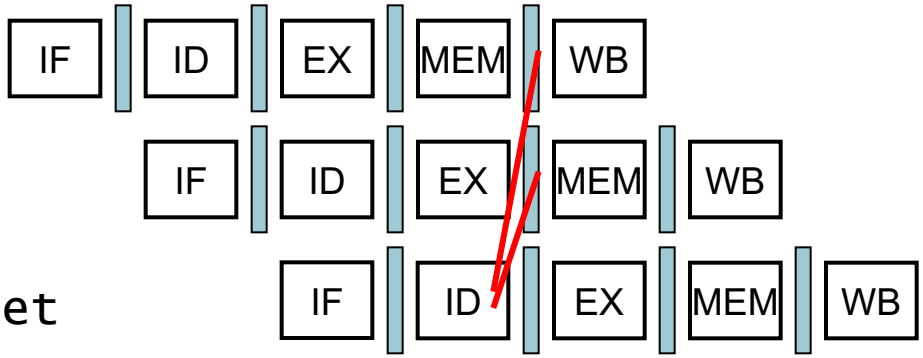


What are the conditions to determine Forward1 and Forward2?

Data Hazards for Branches

- If a comparison register is a destination of *immediately* preceding ALU instruction or *2nd* preceding load instruction

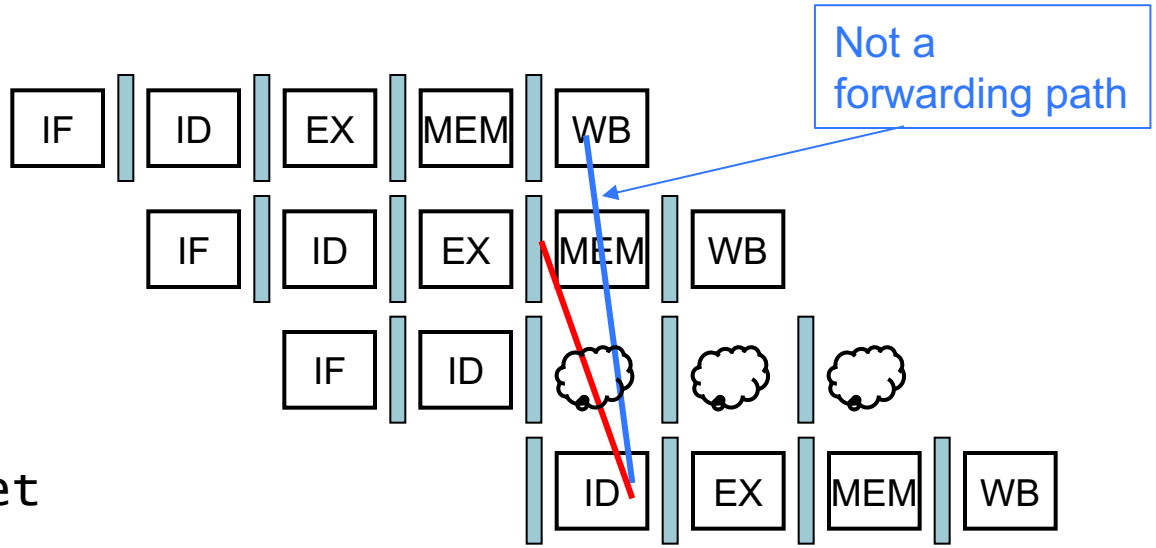
```
lw  x1, addr
add x4, x5, x6
beq x1, x4, target
```



Data Hazards for Branches

- Need 1 stall cycle even with forwarding

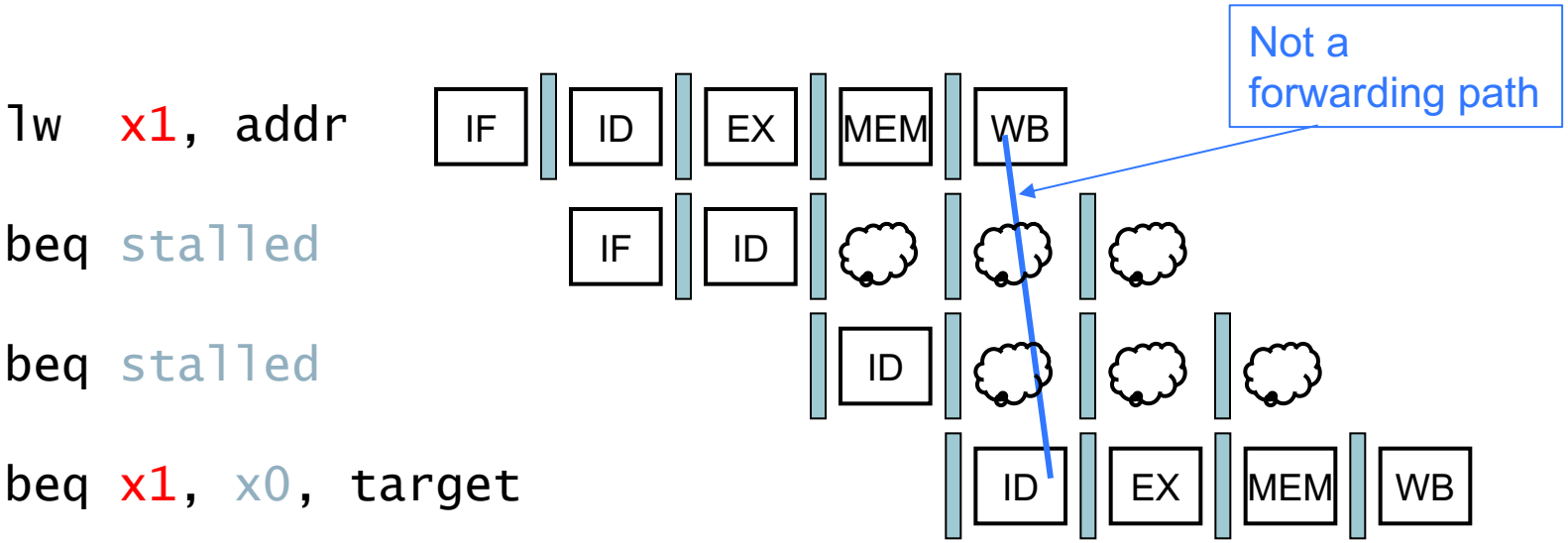
```
lw   x1, addr
add  x4, x5, x6
beq  stalled
beq  x1, x4, target
```



How to detect data hazard for branch?
How to stall?

Data Hazards for Branches

- If a comparison register is a destination of *immediately* preceding load instruction
  - Need 2 stall cycles



Resolutions to Branch Hazard

- Stall on branch
- Always assume branch not taken
- *Branch prediction (instead of assumption)*

Branch Prediction

- Static prediction
  - Based on typical branch behavior
  - Example: loop and if-statement branches
    - Could predict: backward branches always taken
    - Could predict: forward branches always not taken
- Dynamic prediction
  - Hardware measures actual branch behavior
    - e.g., record recent history of each branch in a table
  - Assume future behavior will continue the trend
    - If wrong, take penalty, and update history

Dynamic Branch Prediction

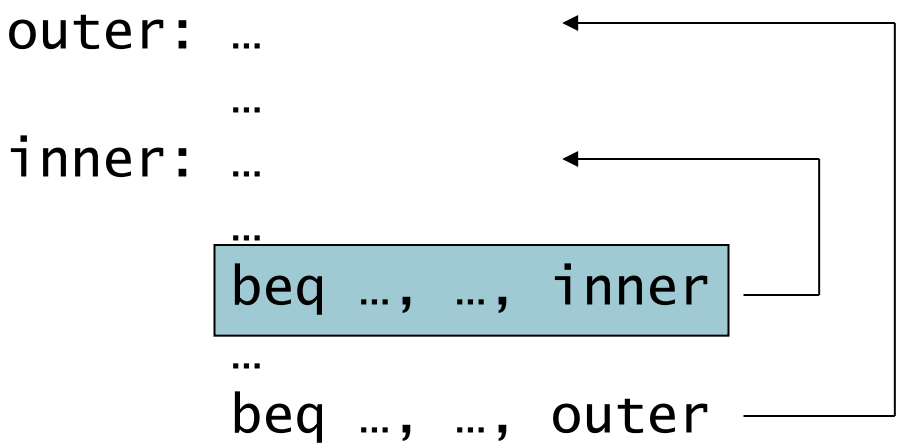
- Dynamic prediction is more precise with more overhead
  - Branch prediction buffer (aka branch history table)
  - Indexed by addresses of branch instructions
  - Stores outcome (taken/not taken) as binary bits
  - To execute a branch
    - Check table, expect the same outcome
    - Start fetching from fall-through or target
    - If wrong, flush pipeline and flip prediction
- In deeper and superscalar pipelines, branch penalty is more significant

Branch Prediction Buffer

address	instruction	1-bit prediction	Target address
0x0008F000	beq	1 (taken)	0x0008F020
0x0008F0C4	beq	0 (not taken)	0x0008F080
0x0008F310	bne	0	.
0x00090000	beq	1	.
. . .	. . .	. . .	.

1-Bit Predictor: Shortcoming

- Inner loop branches mispredicted twice!



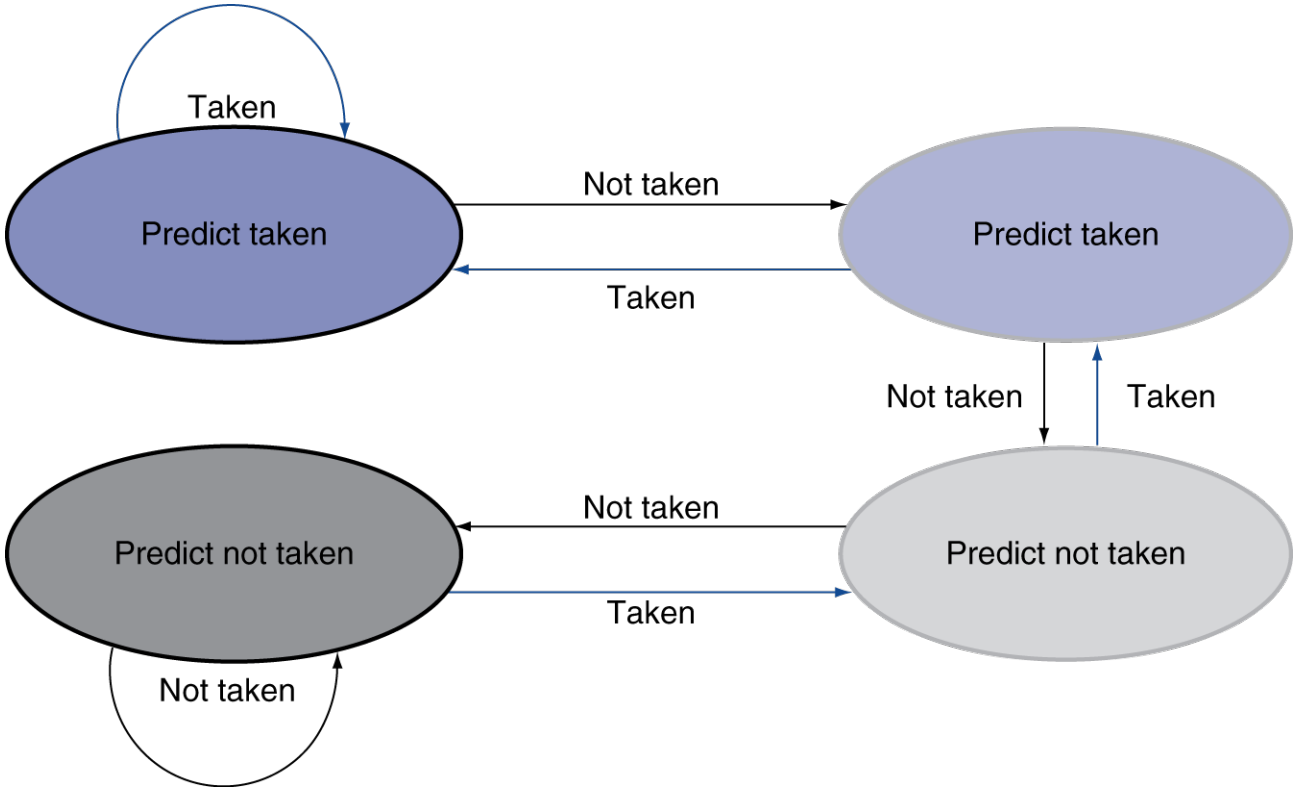
instruction	prediction
beq (inner)	
beq (outer)	

prediction=1: taken
prediction=0: not taken

  - Mispredict as taken on last iteration of inner loop
  - Then mispredict as not taken on first iteration of inner loop next time around

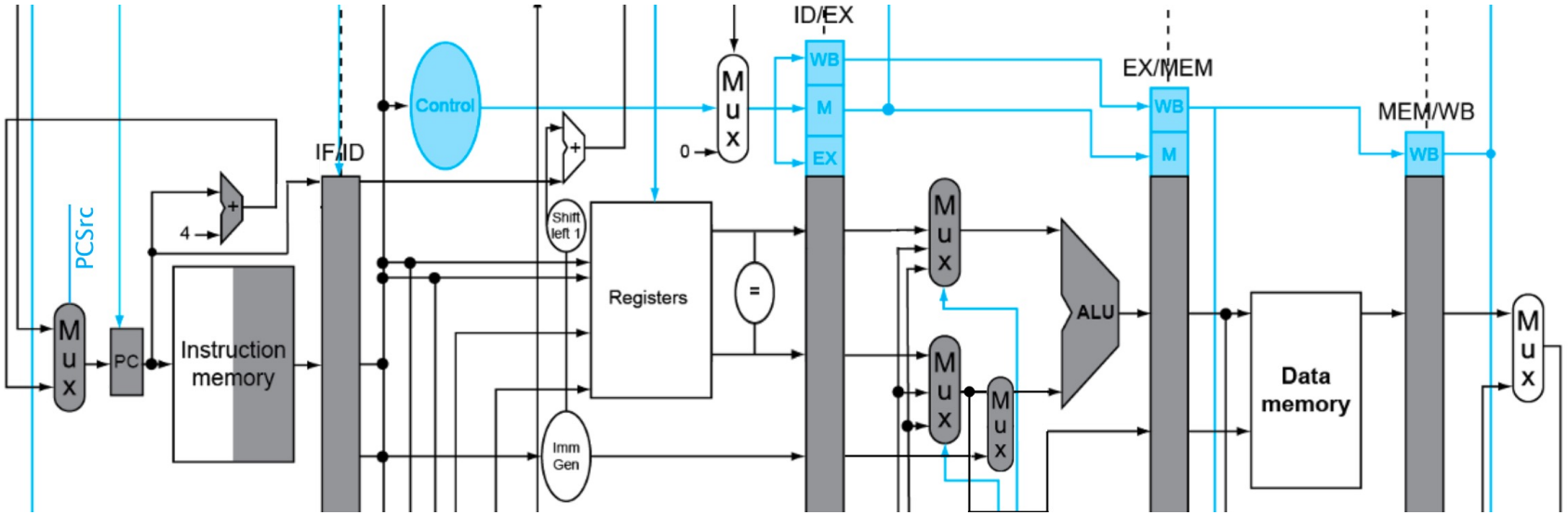
2-Bit Predictor

- Only change prediction on two successive mispredictions



How About J-type?

- How to implement a **`jal`**?
  1. rd <= PC + 4
  2. PC <= Target PC = Current PC + immediate [19:0] × 2



- Is there a control hazard?