



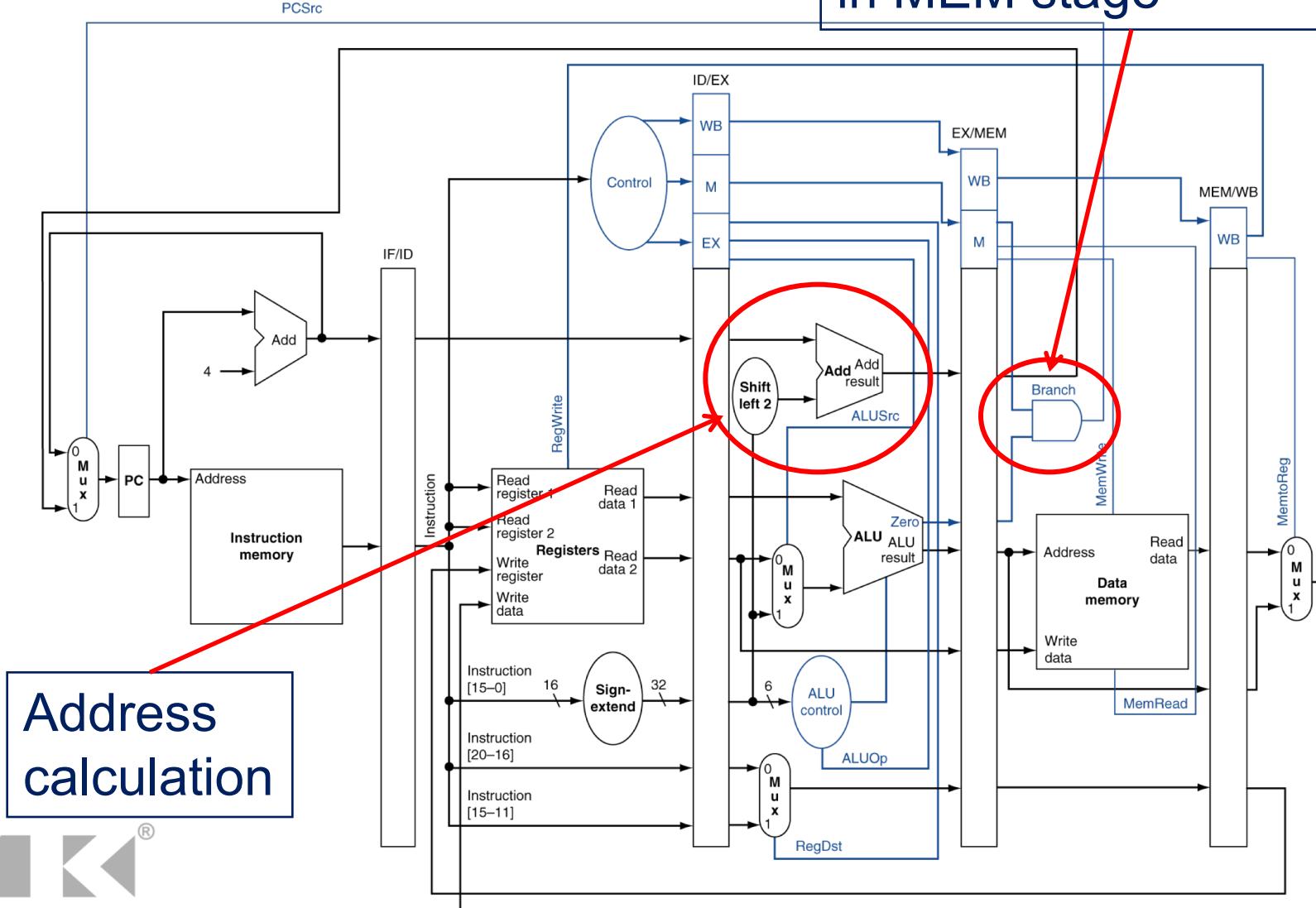
## Topic 9

# Control Hazards

# Branch Hazards

## Current implementation

Determination for branch  
in MEM stage



# Control Hazards

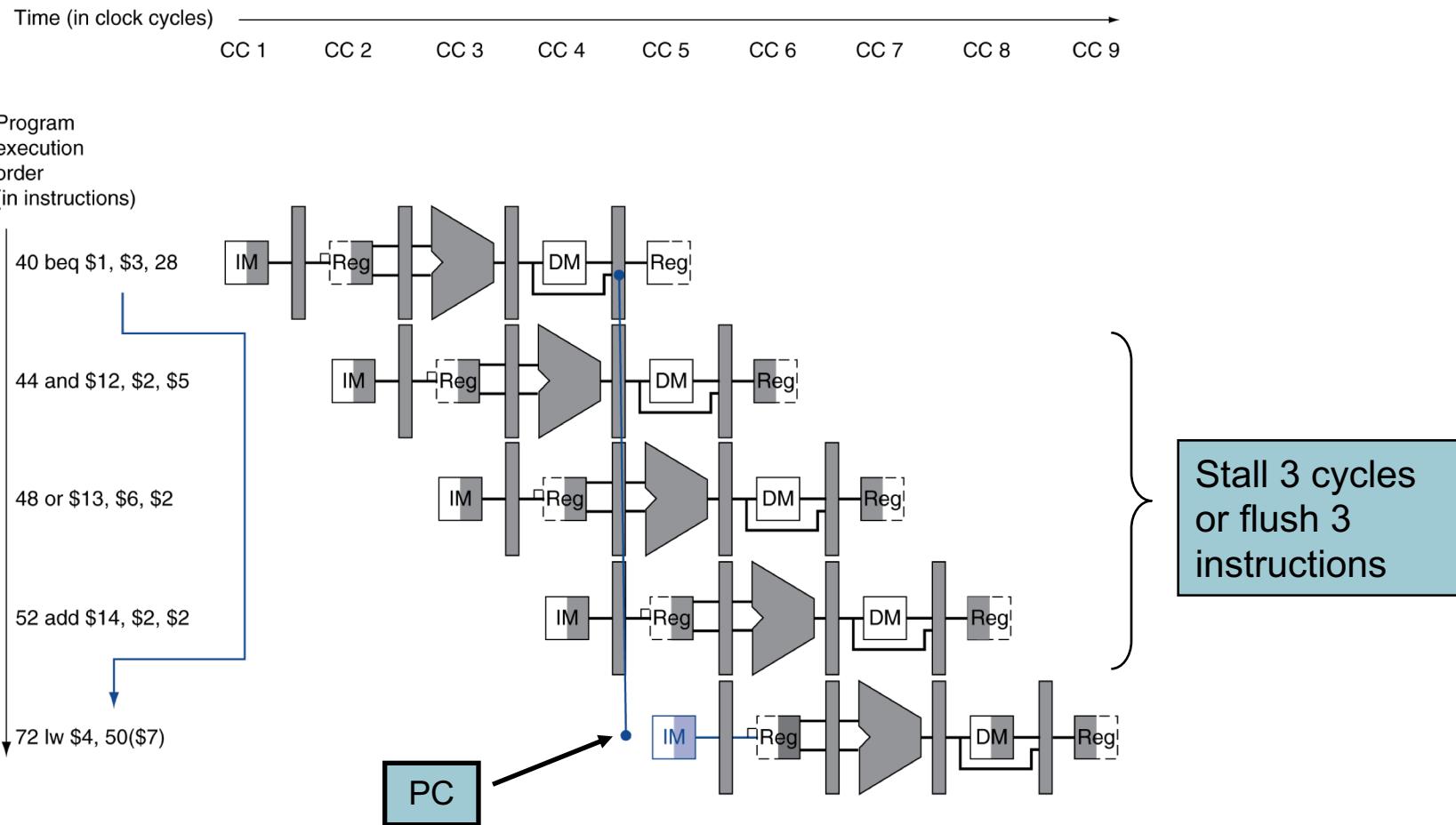
- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipeline can't always fetch correct instruction (Branch Hazard) because
    - When Branch instruction is still in the ID stage, target instruction is needed in the IF stage

# Branch Hazard Resolutions

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

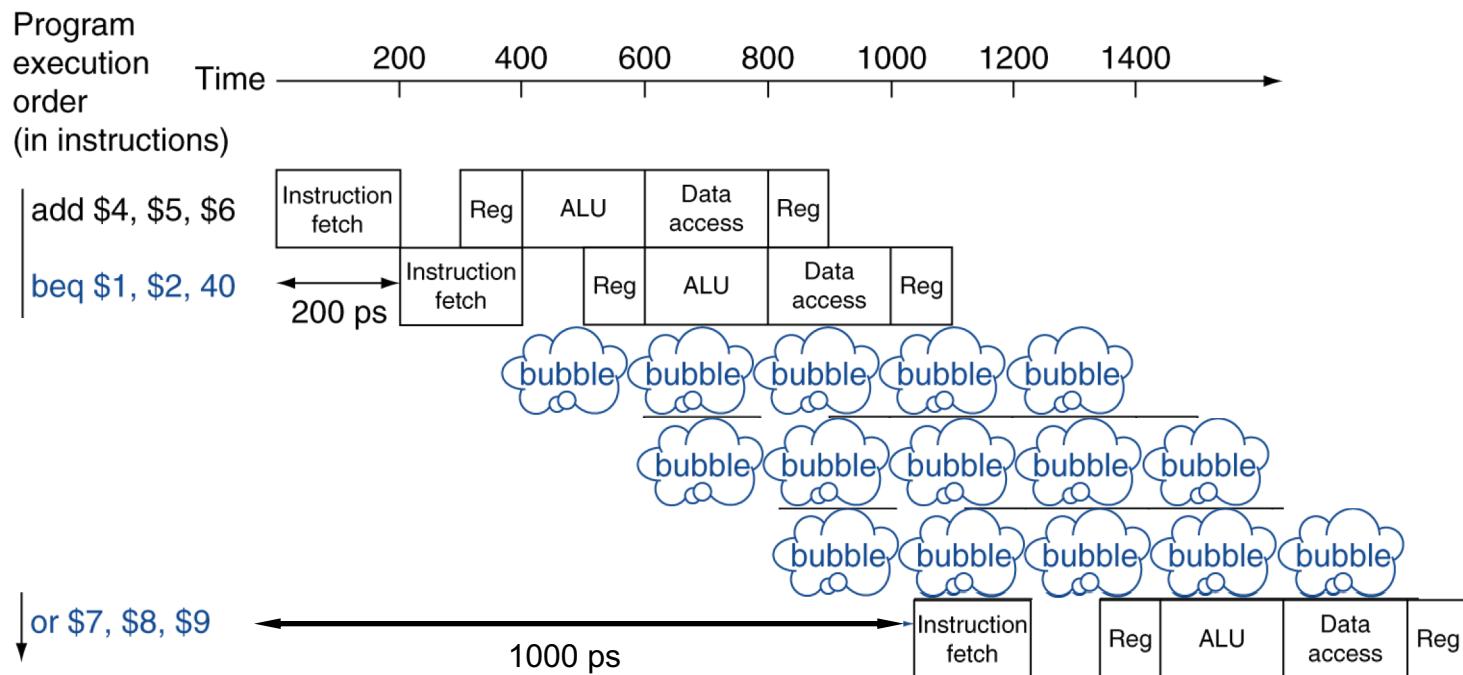
# Branch Hazards

- If branch outcome determined in MEM
  - Stall branch or take penalty of wrong branch



# Stall on Branch

- Wait until branch outcome determined before fetching next instruction



# Branch Hazard Resolutions

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

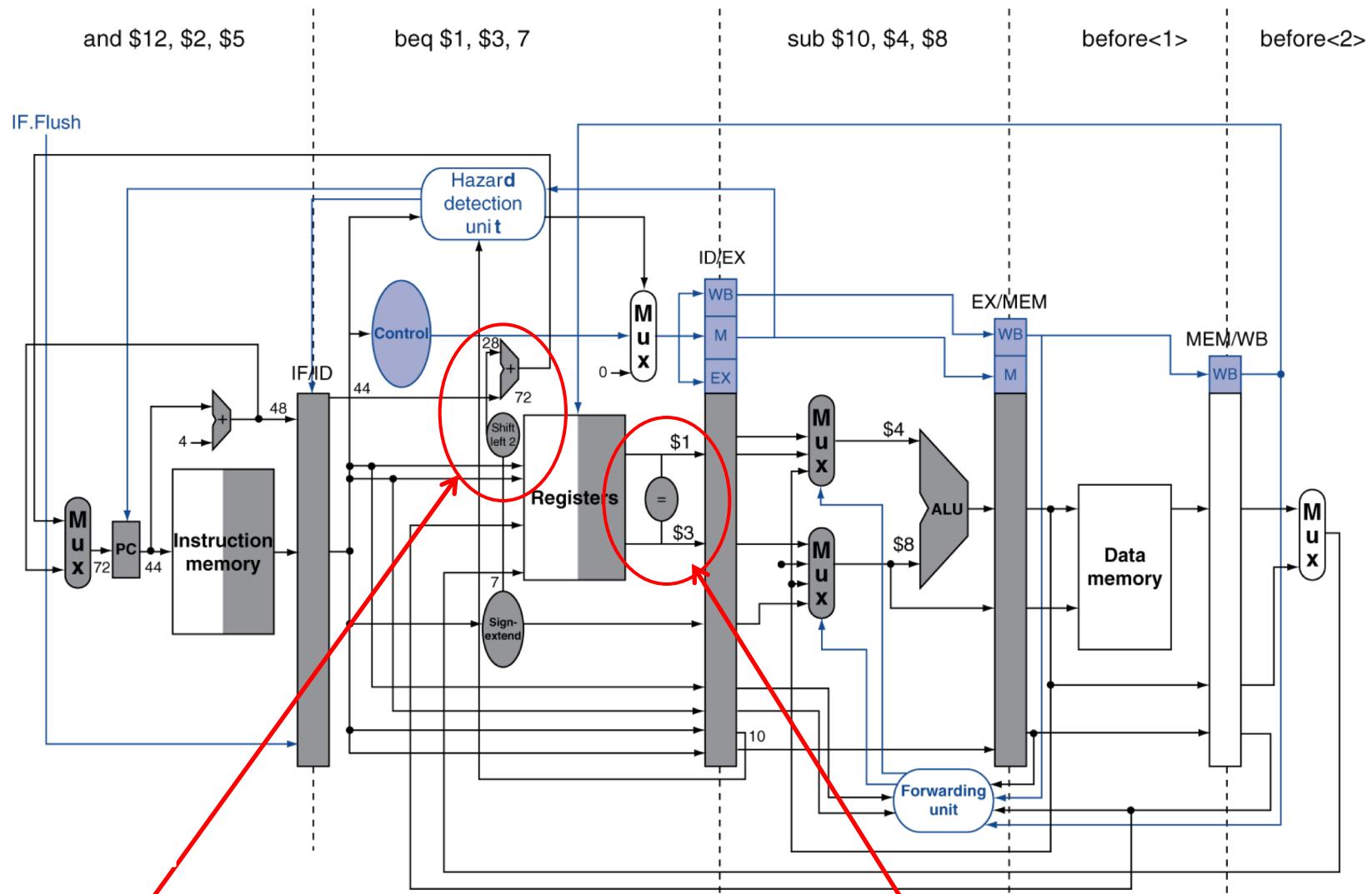
# Assume Branch Not Taken

- If we are right, lucky us!
- If we are wrong, penalty will be to flush some (up to 3) instructions
- Penalty may be reduced
  - By making earlier decision on branch or not
  - Need to compare registers and compute target early in the pipeline

# Reducing Branch Penalty

- Move hardware for determining PC to *ID stage* including
  - Target address calculation
  - Register comparator
- For branch not taken, correct assumption, no penalty
- For taken branch, penalty reduced
  - 1 instruction to be flushed

# Example: Branch Taken



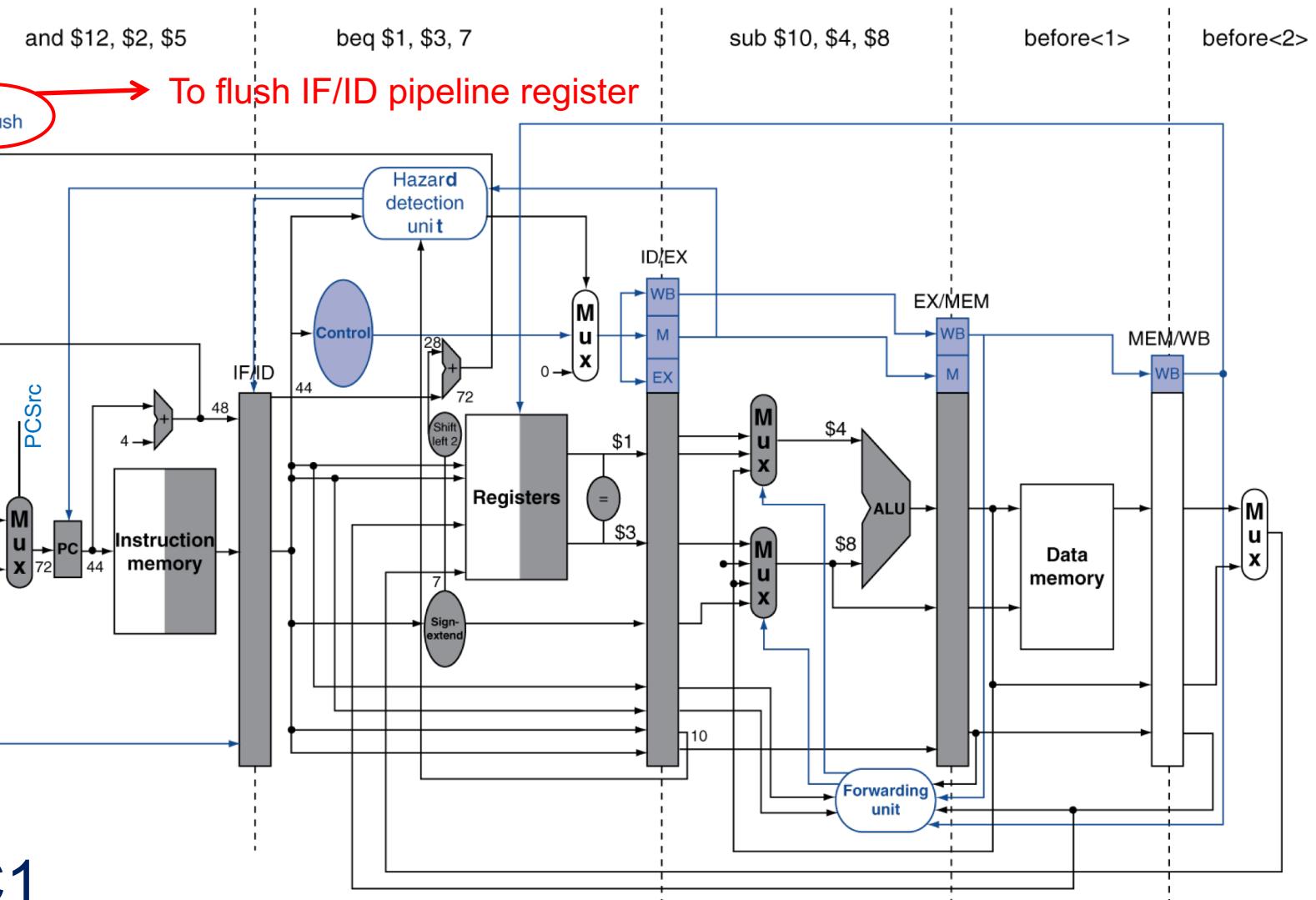
Address calculation

Register comparison

# Flush an Instruction

- Flush: To discard the wrong instruction in pipeline, equivalent to neutralize all operations
  - Clear IF/ID pipeline register, by a new control signal **IF.Flush**
    - Flushes the instruction in IF stage

IF.Flush (synchronous)	Branch	“=” Output	PCSrc	PCin	PCout	IF/ID Register
1	1	1	0	72	44	beq & PC+4



CC1

IF.Flush (synchronous)

Branch

“=” Output

PCSrc

IF/ID Register

0

0

1

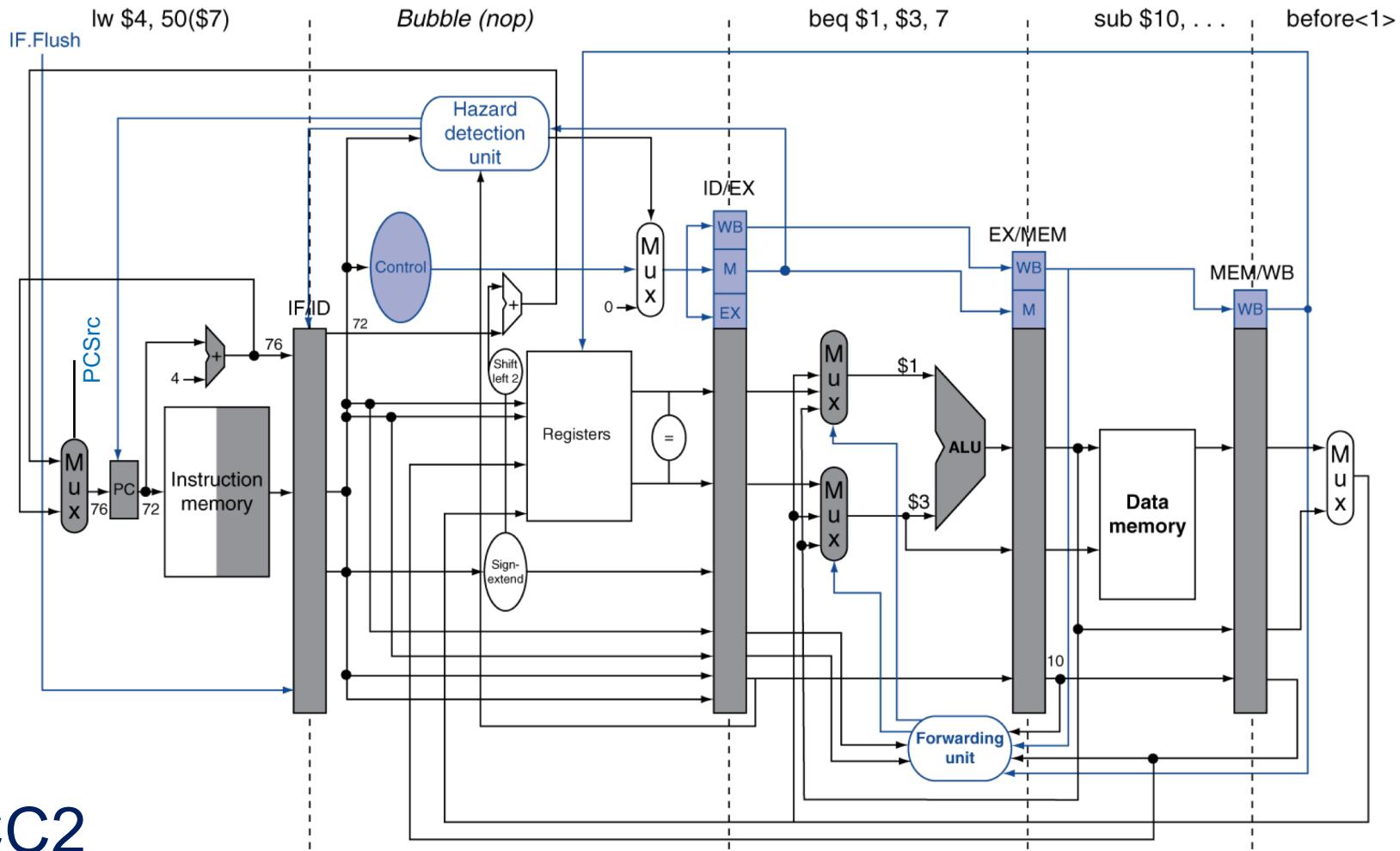
1

0

Branch target instruction

“and” instruction flushed

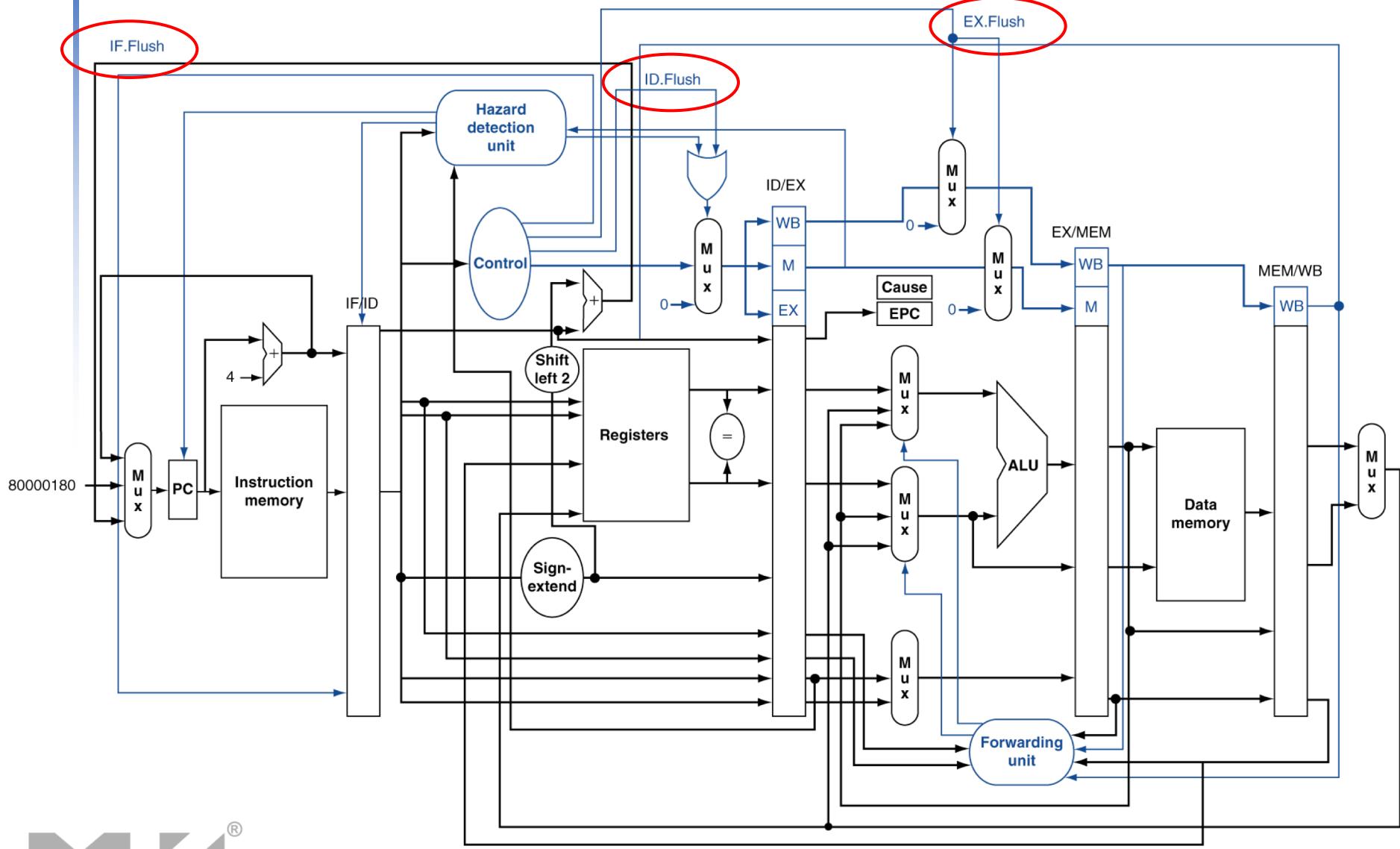
Branch condition is true



# Flush Multiple Instructions

- What's in IF stage is previous 1 instruction, how to flush previous 2 or 3 instructions in ID and EX stages?
  - Send 0 control signals through the pipeline
    - Flushes the instruction in ID stage
    - Flushes instructions in following stage

# Pipeline with Extra Controls



**IF.Flush****ID.Flush****EX.Flush**

1

1

1

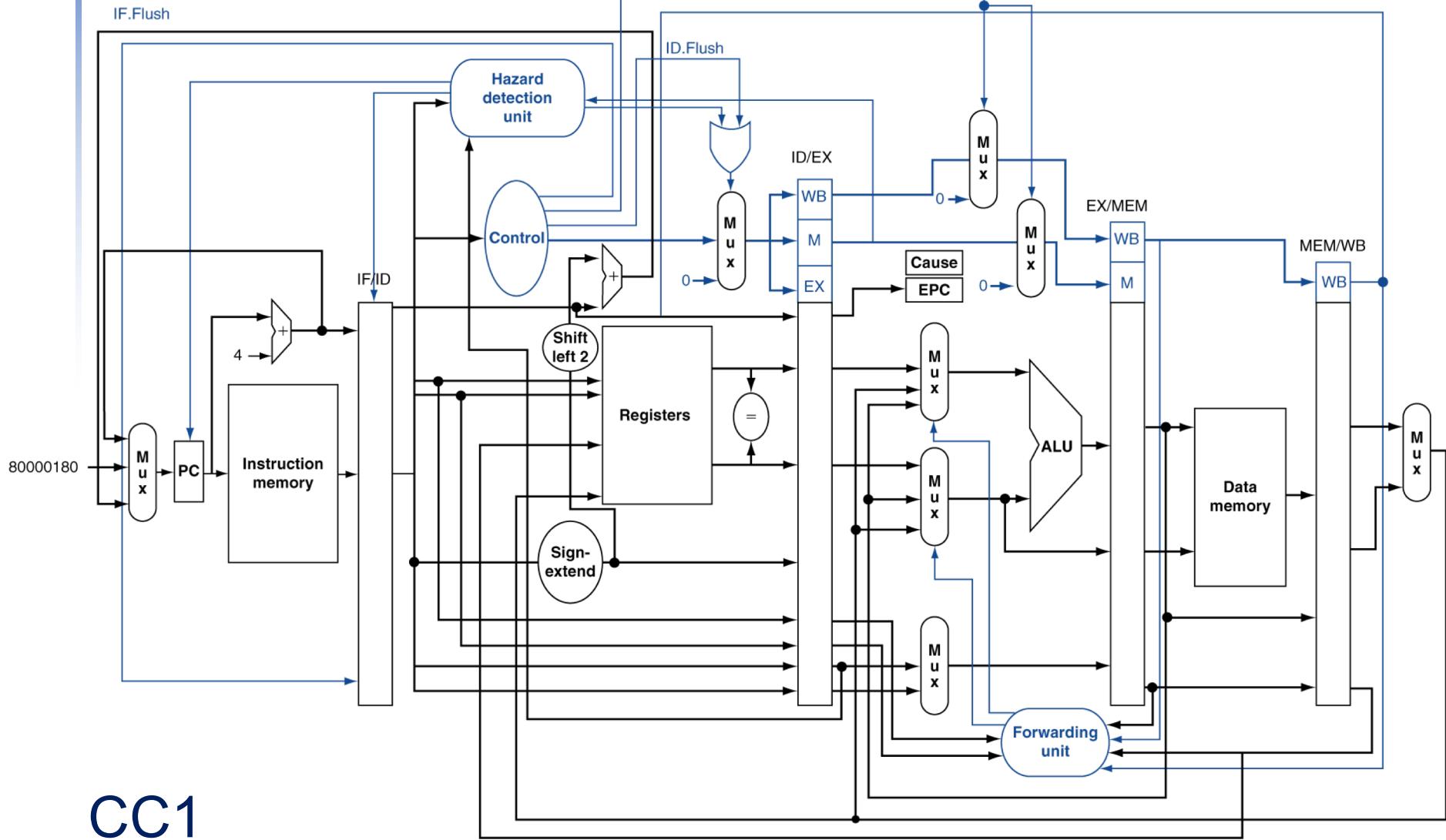
add

or

and

beq

add



**IF.Flush****ID.Flush****EX.Flush**

0

0

0

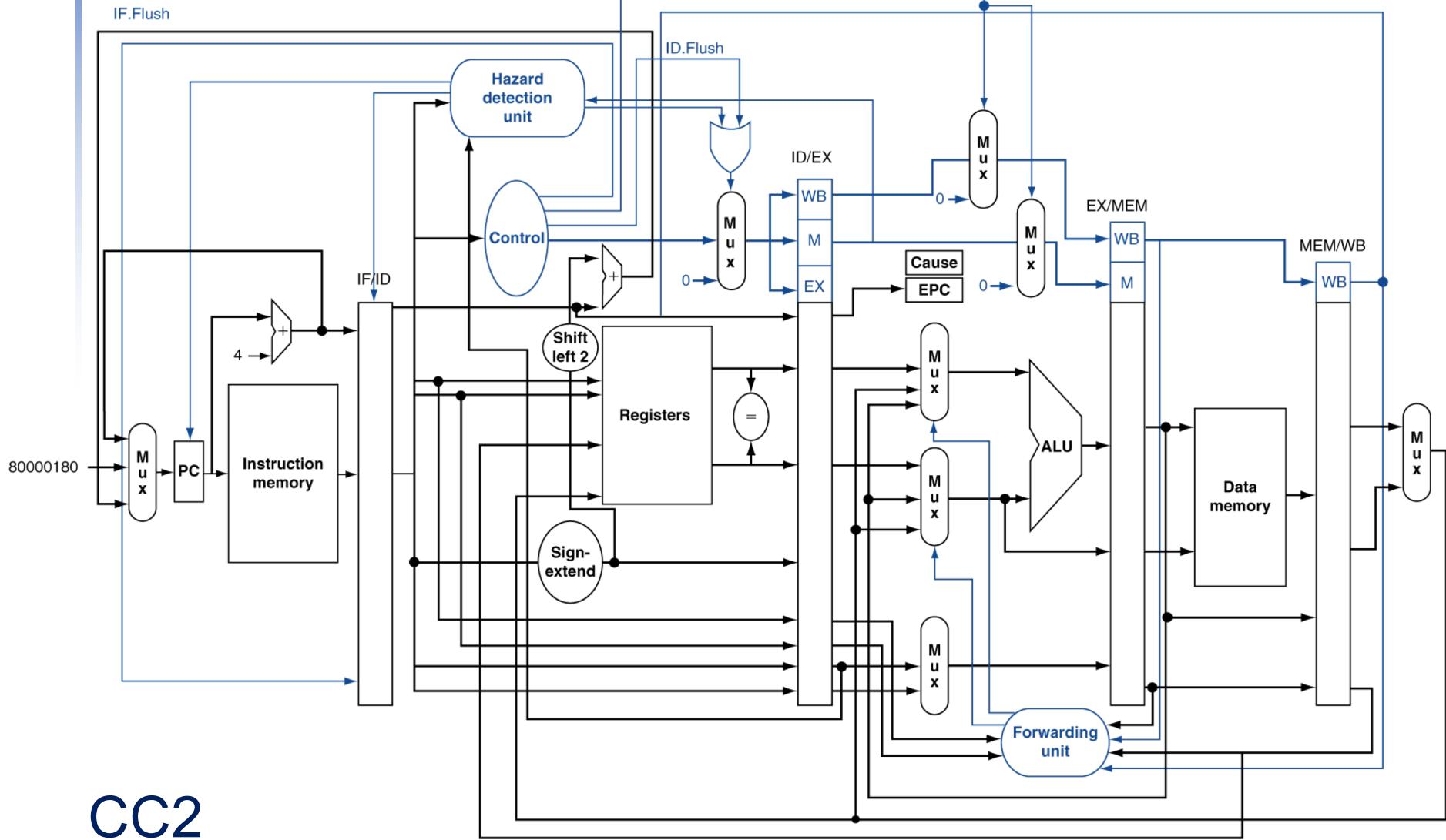
addi

nop

nop

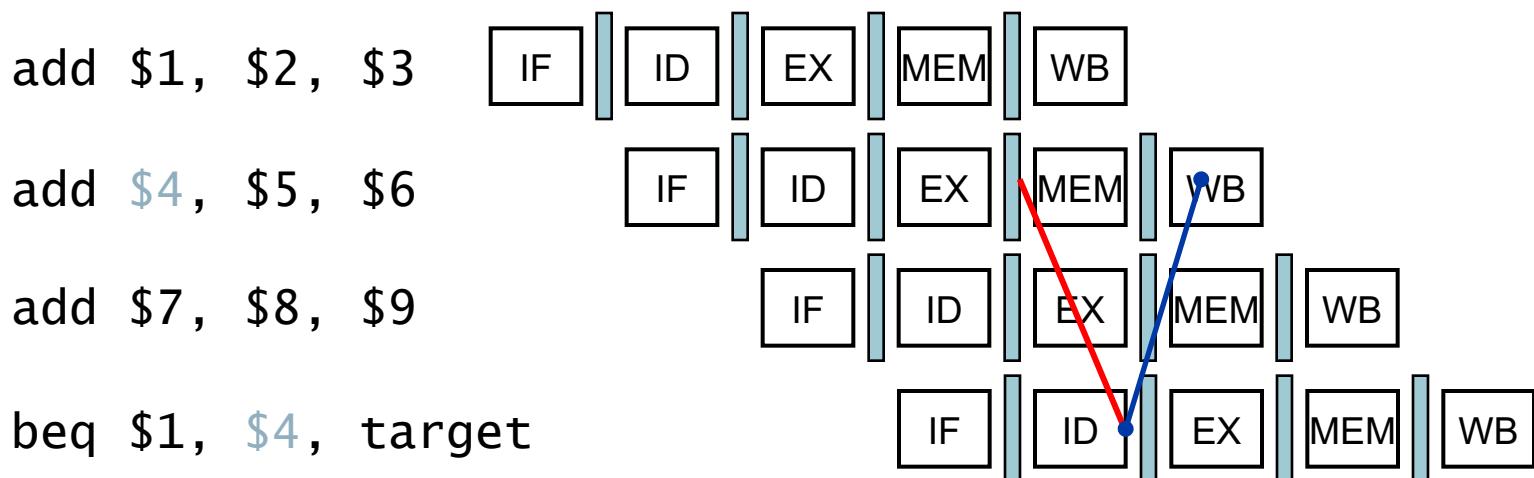
nop

beq



# Data Hazards for Branches

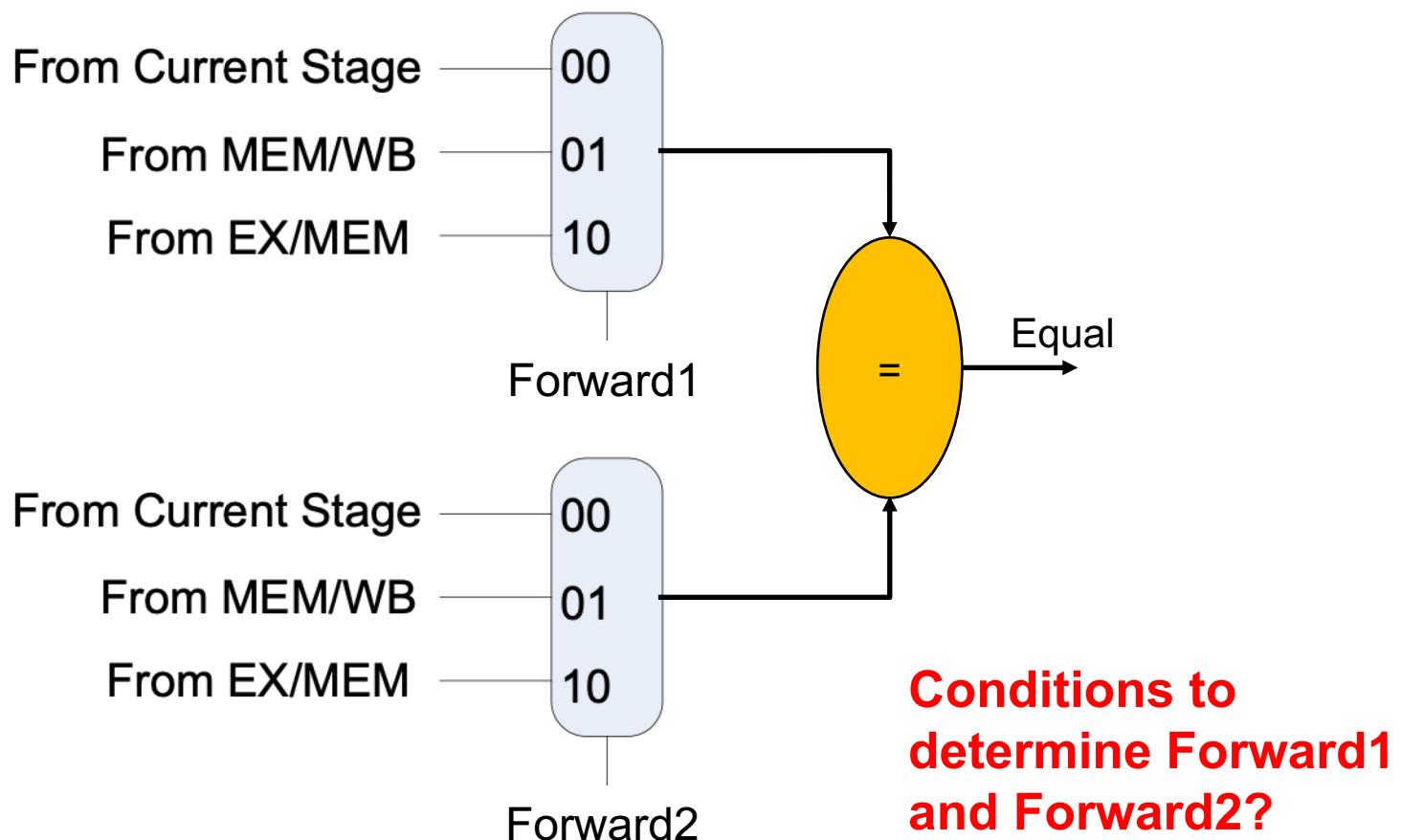
- Changing datapath structure causes more possible data hazards
- If a register for comparison is a destination of 2<sup>nd</sup> preceding ALU instruction



- Can resolve using new forwarding paths
- Is \$1 a data hazard?

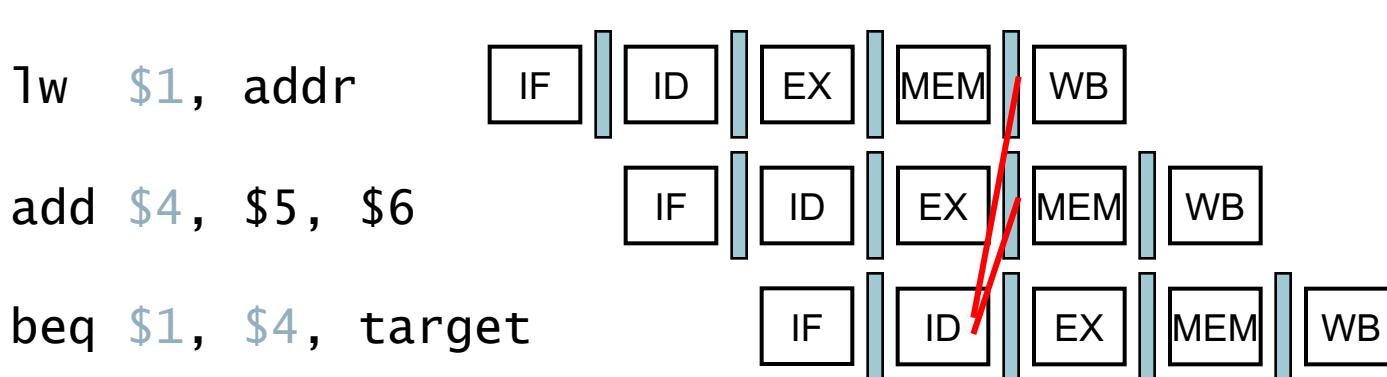
# Forwarding Paths

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



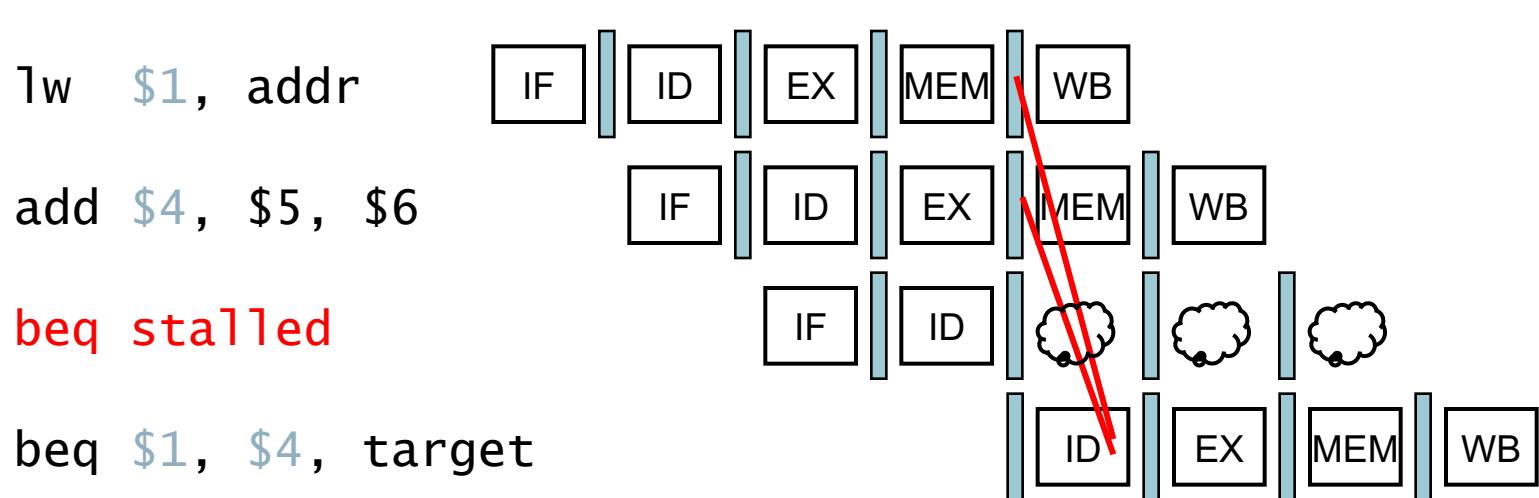
# Data Hazards for Branches

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



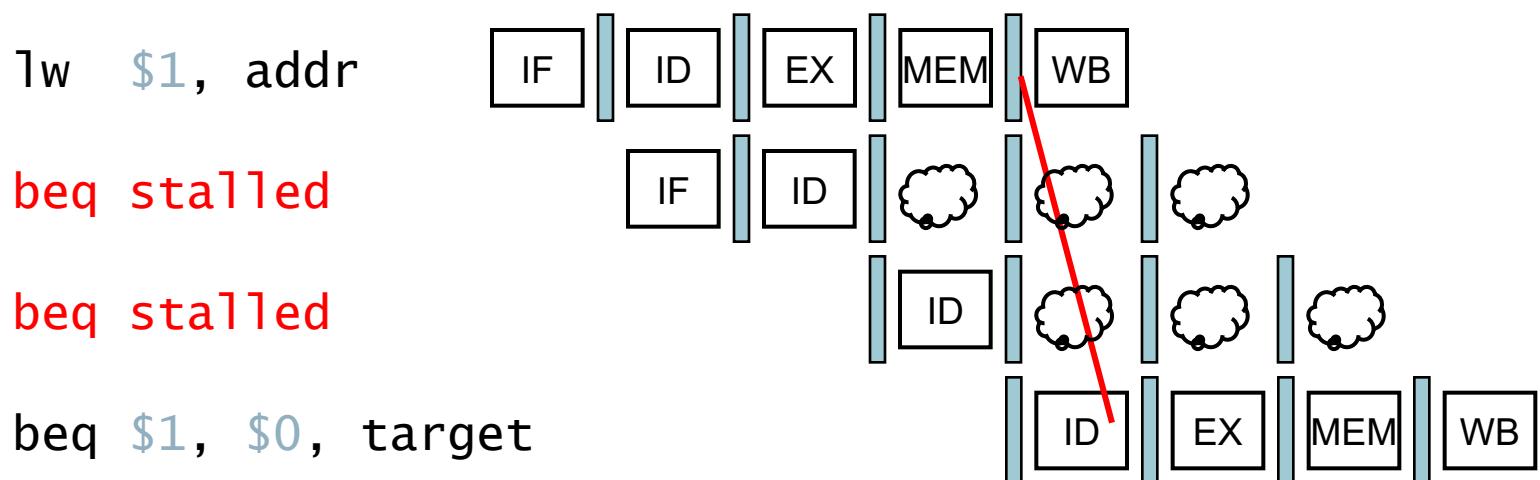
# Data Hazards for Branches

- Need 1 stall cycle even with forwarding



# Data Hazards for Branches

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



# Branch Hazard Resolutions

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

# Branch Prediction

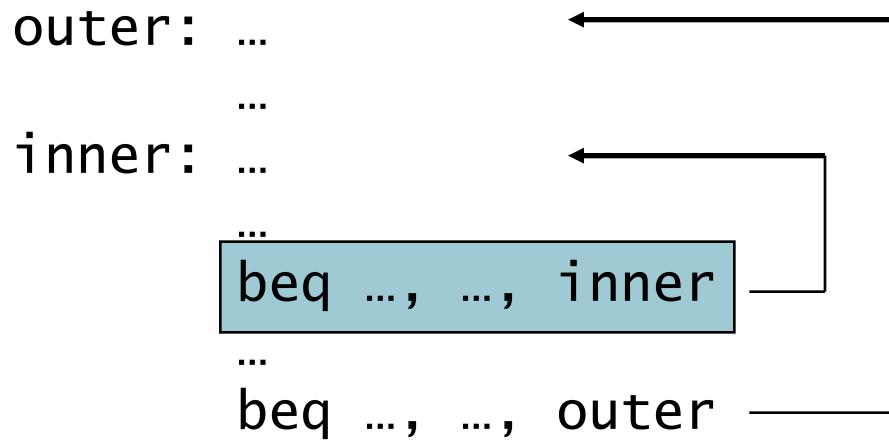
- Static prediction
  - Based on typical branch behavior
  - Example: loop and if-statement branches
    - Could predict backward branches taken
    - Could predict forward branches not taken
- Dynamic prediction
  - Hardware measures actual branch behavior
    - e.g., record recent history of each branch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

# Dynamic Branch Prediction

- Dynamic prediction is better
  - Branch prediction buffer (aka branch history table)
    - Indexed by recent branch instruction addresses
    - Stores outcome (taken/not taken)
  - To execute a branch
    - Check table, expect the same outcome
    - Start fetching from fall-through (next) or target address
    - If wrong, flush pipeline and flip prediction in the buffer
- In deeper pipelines (more stages), branch penalty is more significant

# 1-Bit Dynamic Predictor

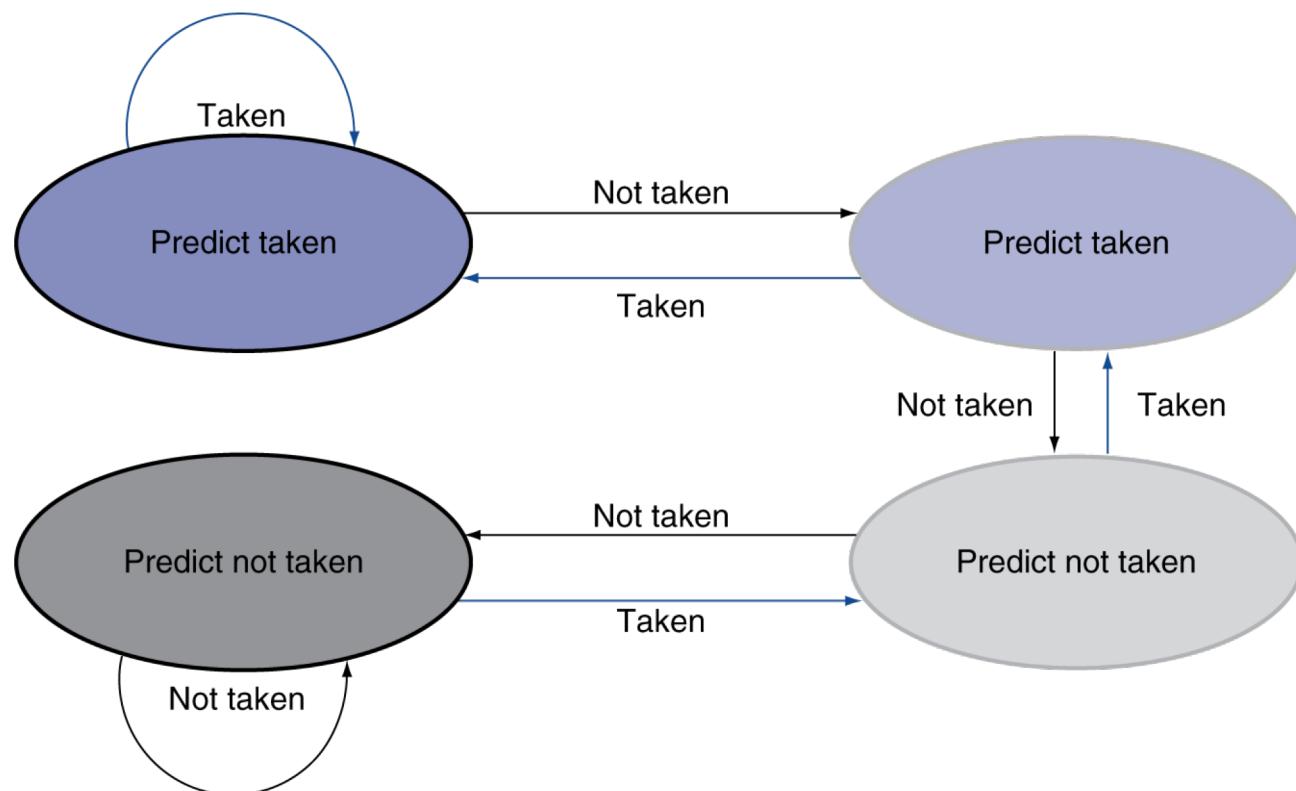
- Assume branch taken
- Inner loop branches mis-predicted twice in every outer loop iteration



- Mis-predict as taken on last iteration of inner loop
- Then mis-predict as not taken on first iteration of inner loop next time around

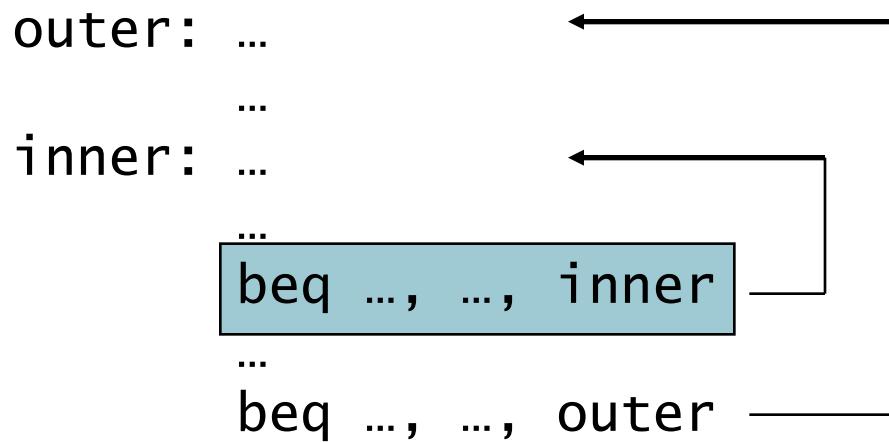
# 2-Bit Dynamic Predictor

- Only change prediction on two successive mispredictions



# 2-Bit Dynamic Predictor

- Assume initial state of strong branch taken
- Inner loop branches mis-predicted once in every outer loop iteration



- Mis-predict as taken on last iteration of inner loop

# Branch Hazard Resolutions

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

# Delayed Branch

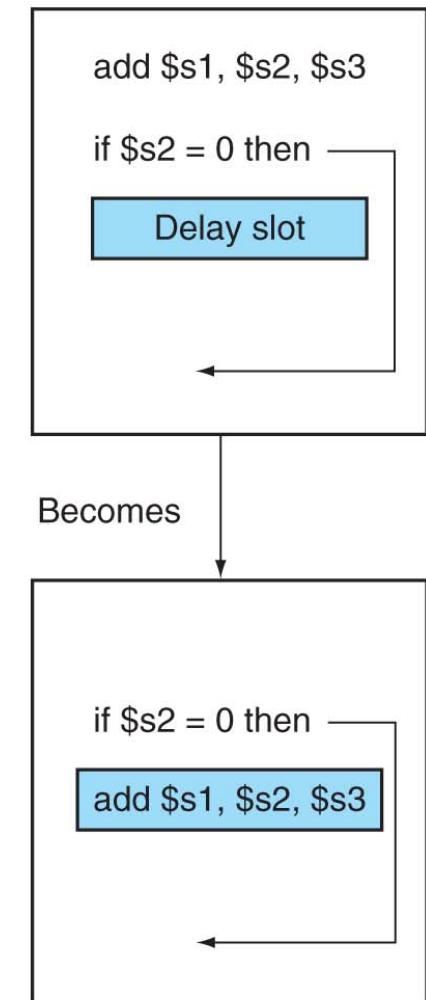
- Always delay the branch
  - With one instruction, NOP or usually a real instruction, for 5 stage pipeline
  - Maybe more delays for deeper pipeline
- Requires carefully designed compiler

# Delayed Branch

- Always execute the instruction immediately following branch
  - Called Branch delay slot
  - Further reduce branch penalty
- One branch delay slot becomes insufficient
  - When pipeline gets deeper or multiple instructions per clock cycle
  - Dynamic branch prediction is better choice

# Example for Delayed Branch

- Will remove the 1 clock cycle penalty
- Will work only if instructions can be found to fill the delay slot



# Overall Structure

