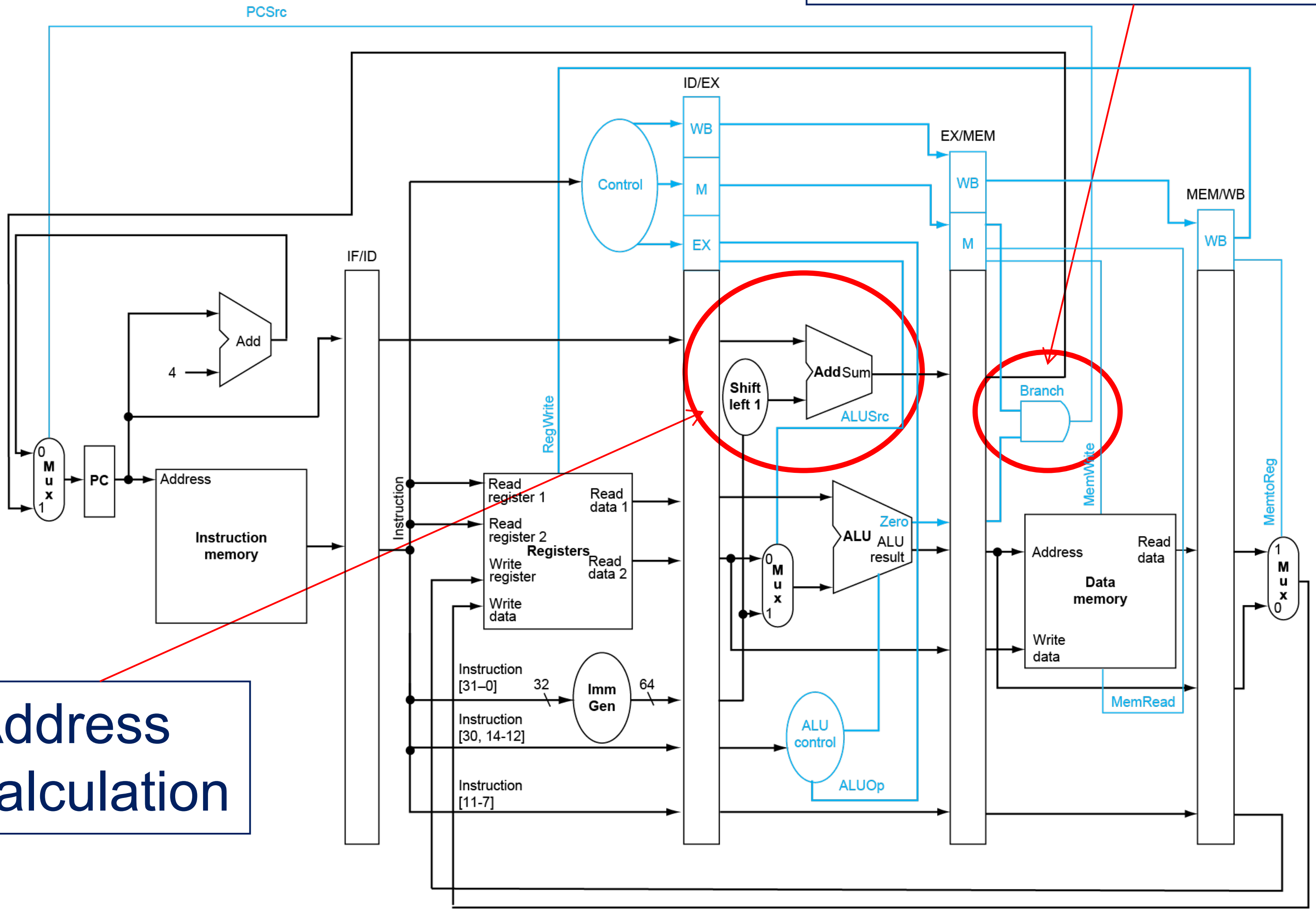
Topic 8

Control Hazards

Branch Hazards

- Current implementation

Determination for branch in MEM stage



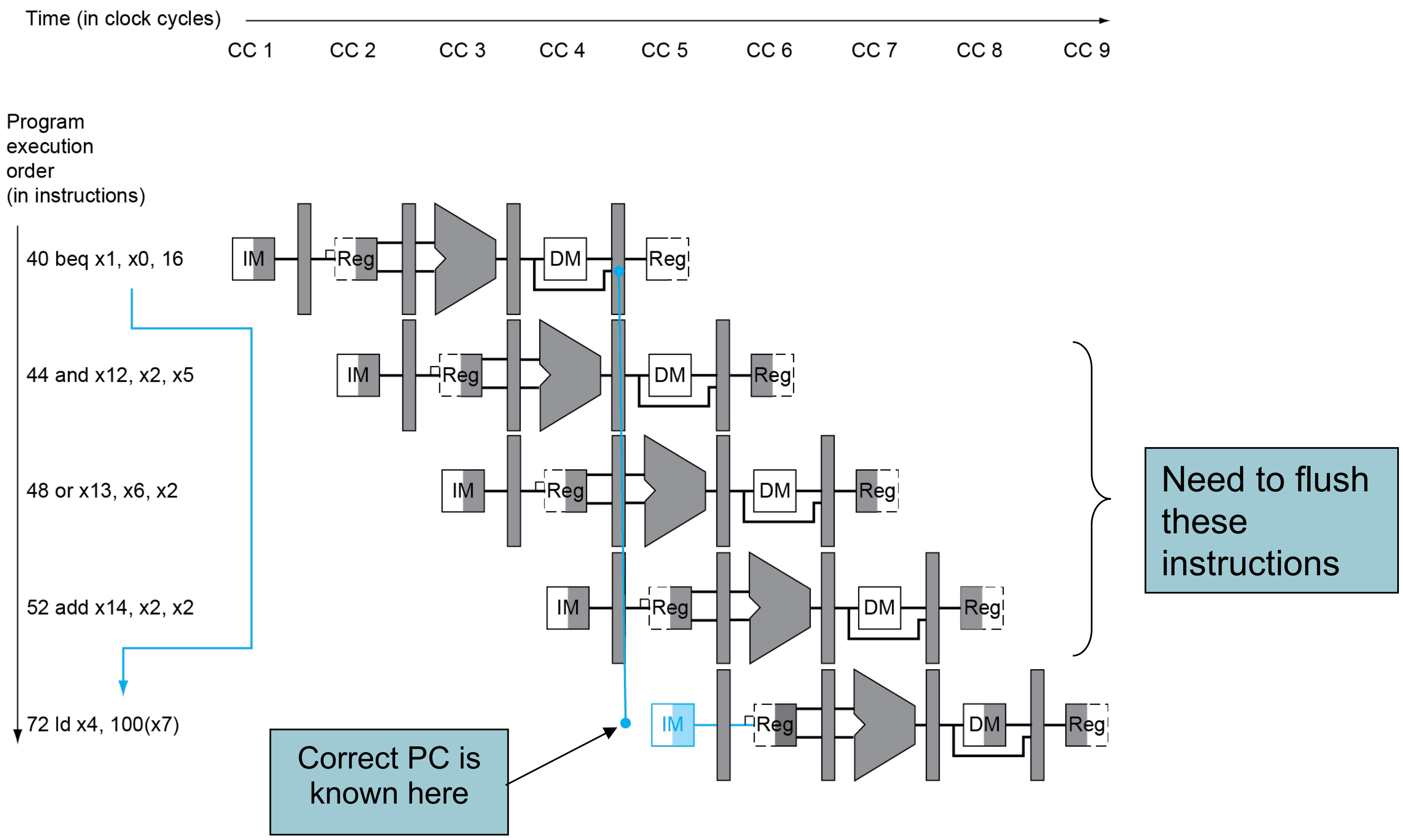
Address calculation

Control Hazards

- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipeline can’t always fetch correct instruction
    - Still working on ID stage of branch
- In RISC-V pipeline
  - Need to compare registers and compute target early in the pipeline

Branch Hazards

- Branch outcome determined in MEM

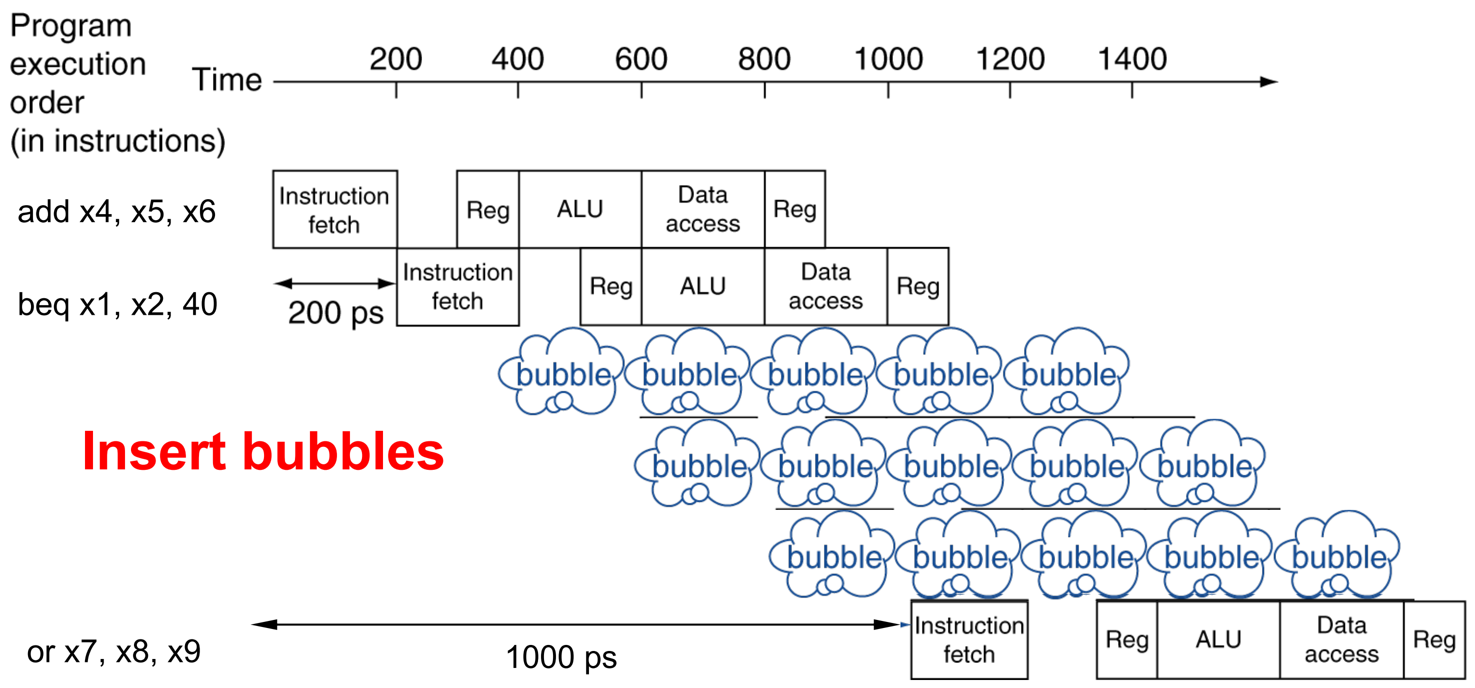


Branch Hazard Resolutions

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

Stall on Branch

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

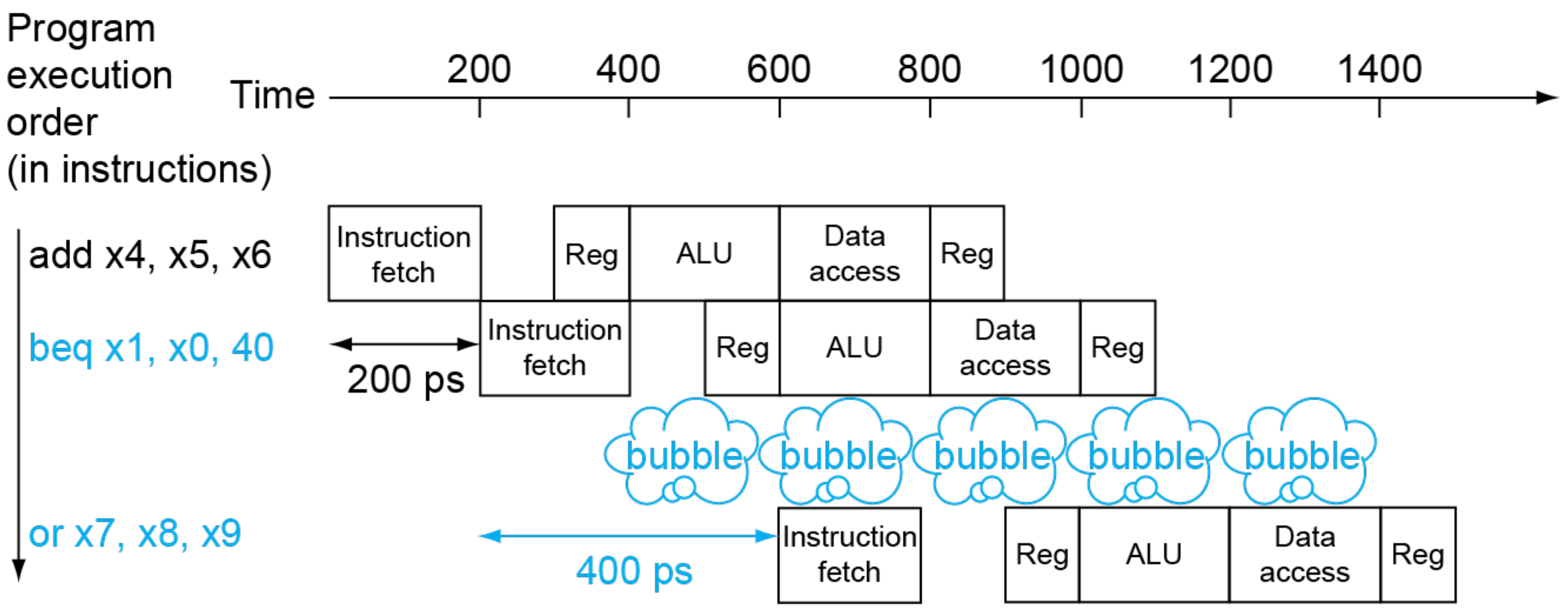


Reducing Branch Delay

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

Stall on Branch

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



Branch Hazard Resolutions

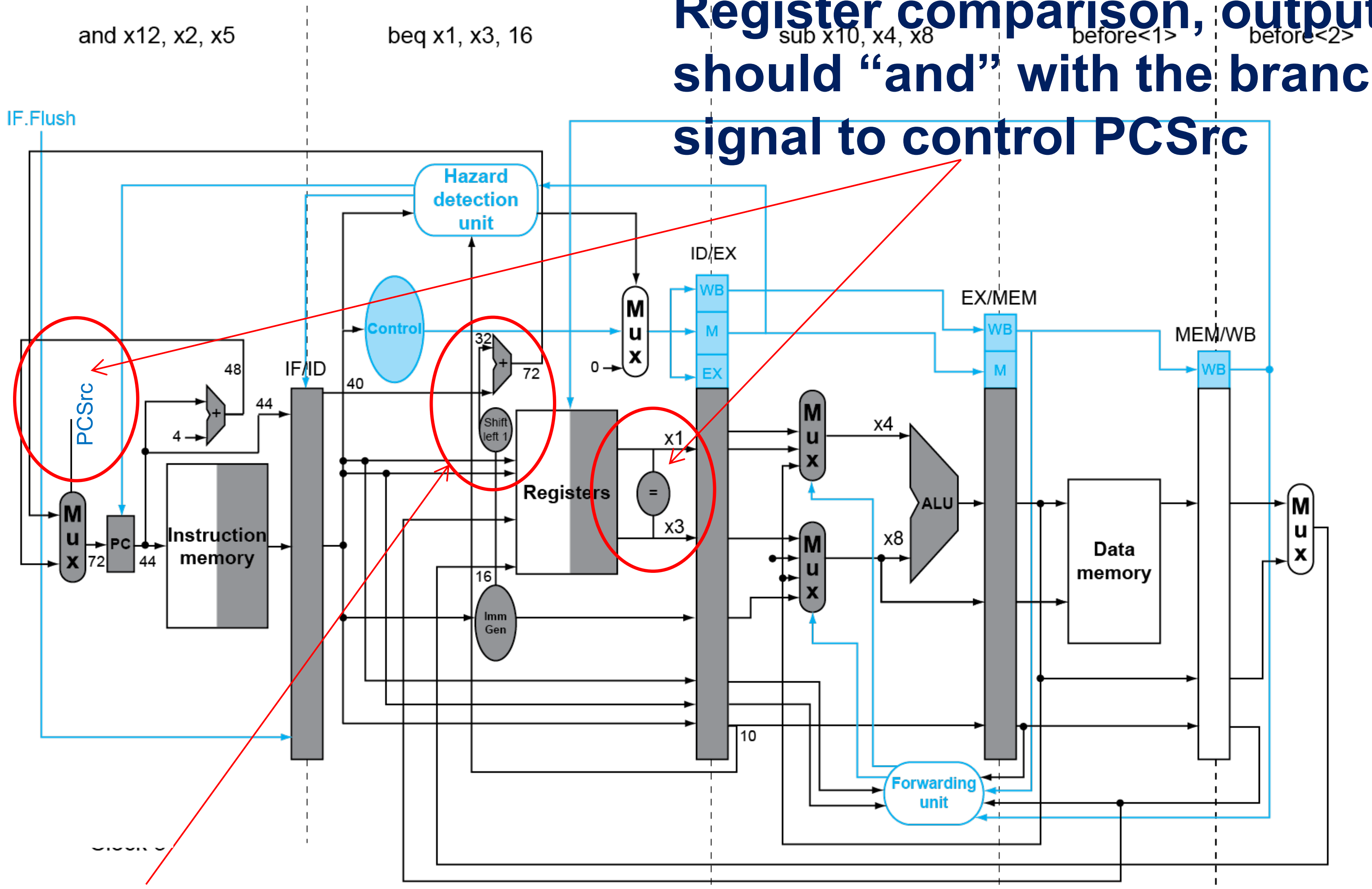
- Stall on branch
- *Always assume branch not taken*
- Branch prediction

Assume Branch Not Taken

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

Example: Branch Taken

Register comparison, output should “and” with the branch signal to control PCSrc

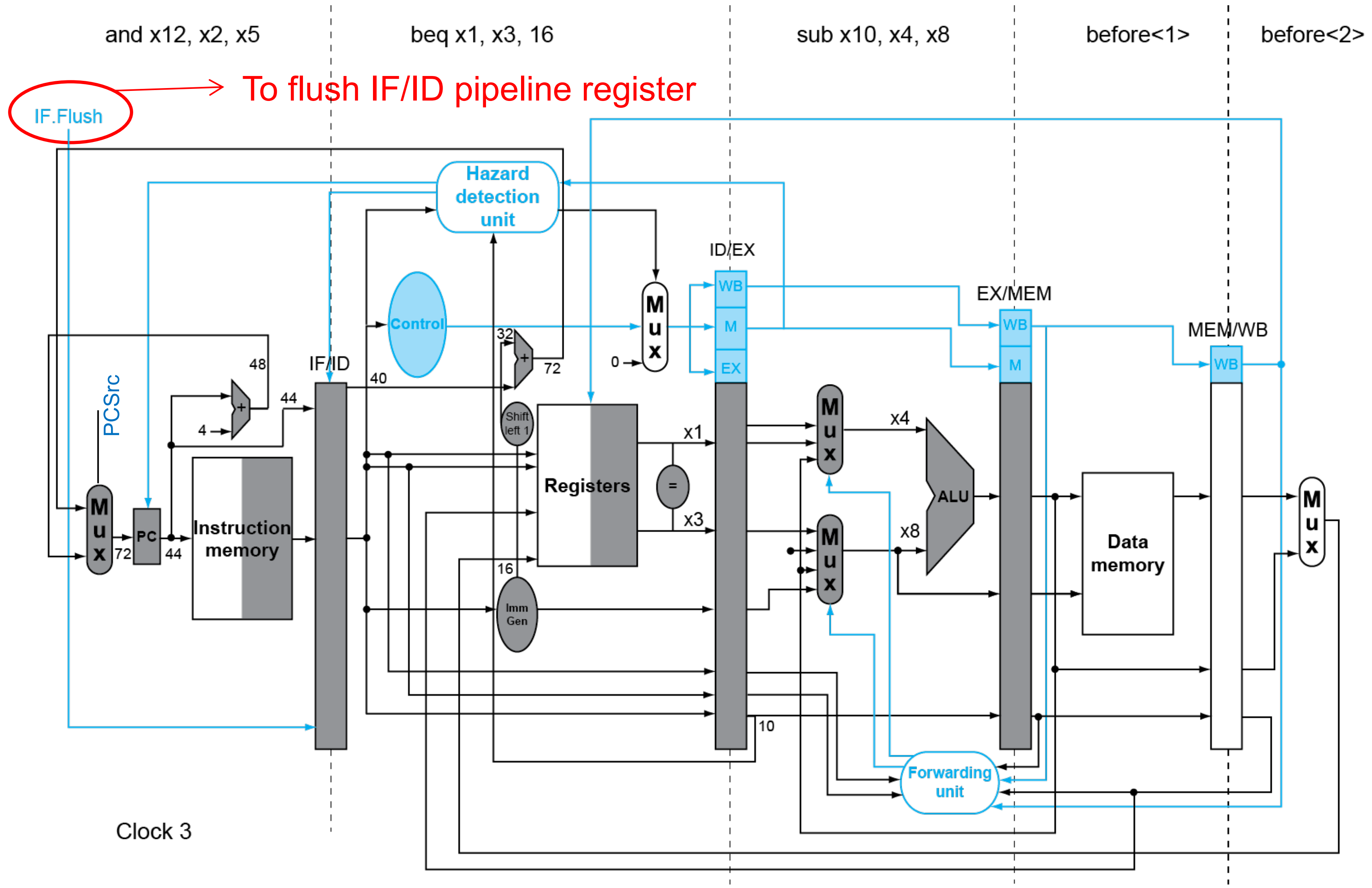


Address calculation

Flush an Instruction

- Assume branch should not be taken, but wrong, then we need to flush the wrongly fetched 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

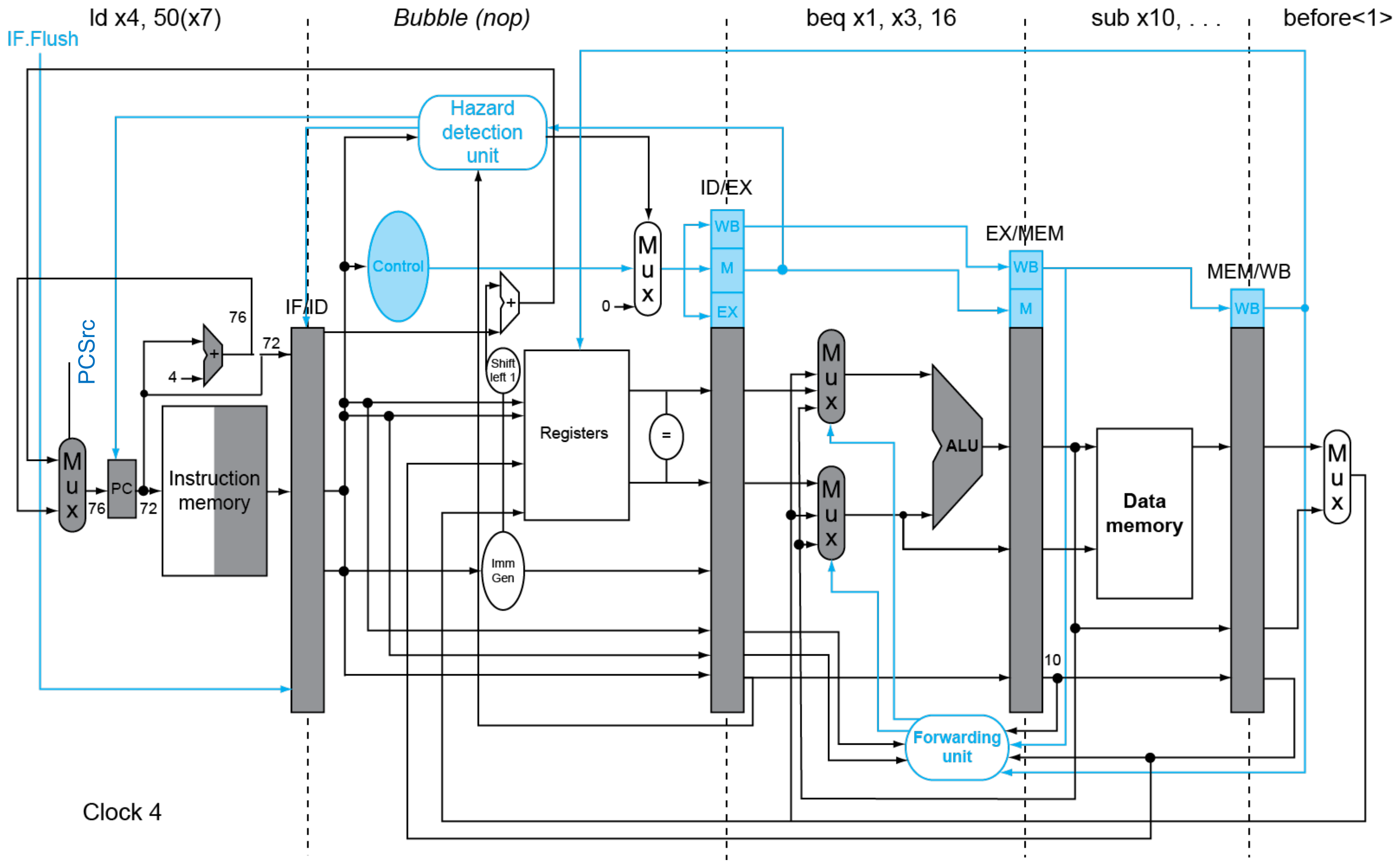


IF.Flush (synchronous)	Branch	"=" Output	PCSrc	IF/ID Register
0	0	1	1	0

Branch target instruction

"and" instruction flushed

Branch condition is true



Data Hazards for Branches

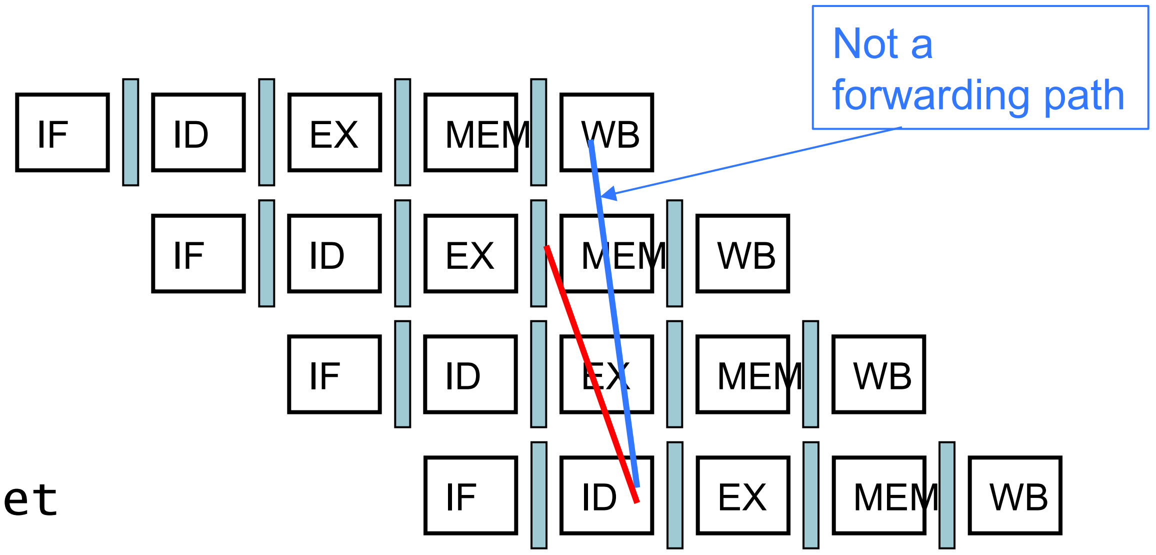
- If a comparison register is a destination of 2nd preceding ALU instruction

```
add x1, x2, x3

add x4, x5, x6

…

beq x1, x4, target
```

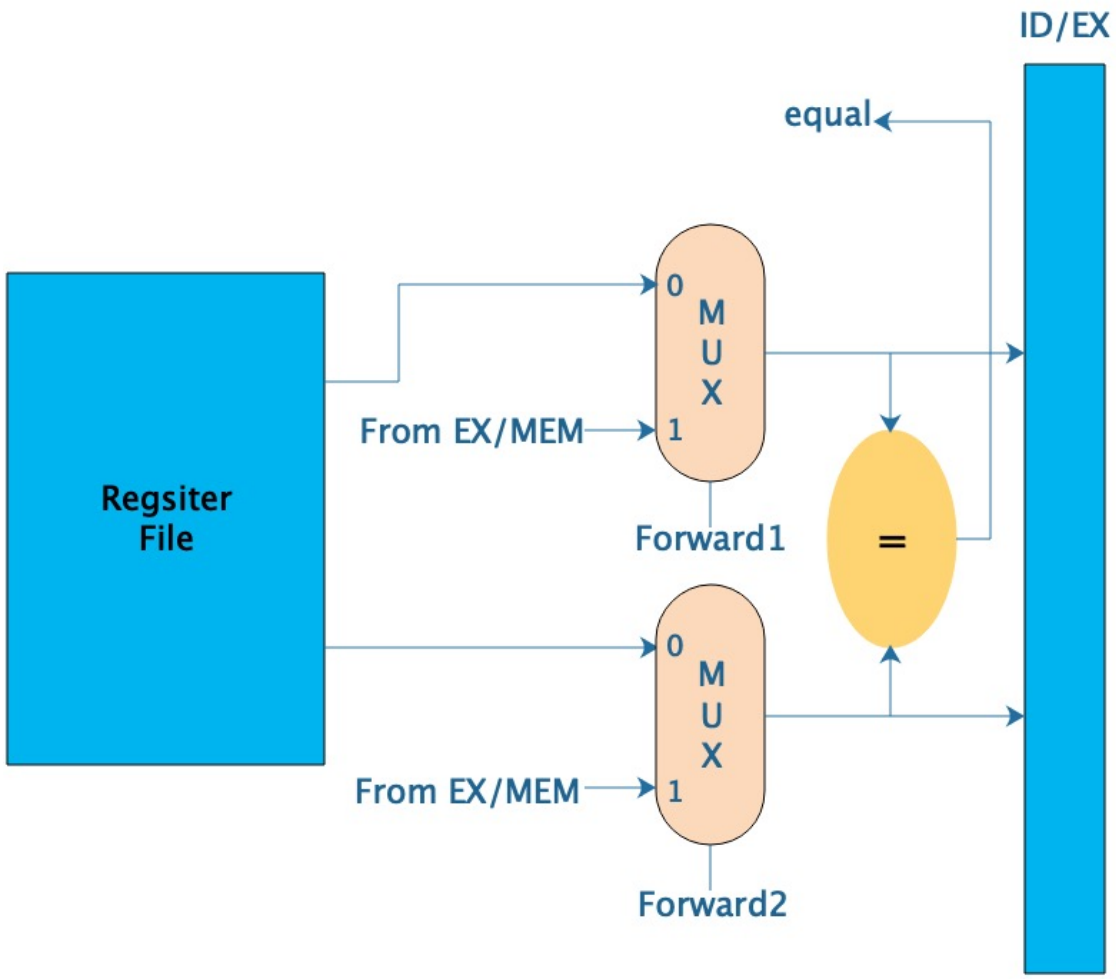


- Can resolve using new forwarding path

Forwarding Paths

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

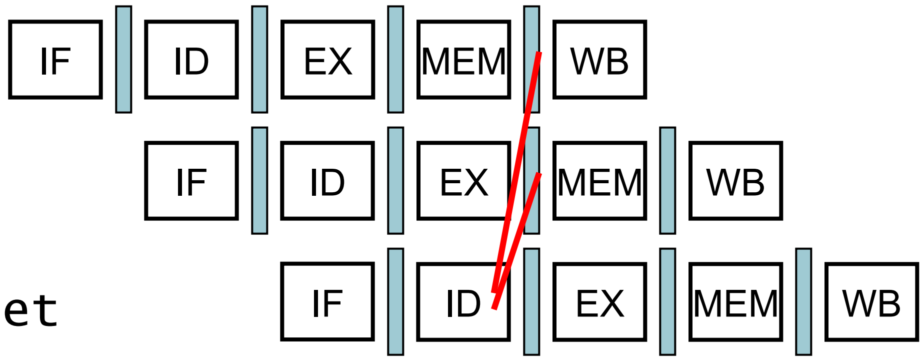
Conditions to determine Forward1 and Forward2?



Data Hazards for Branches

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

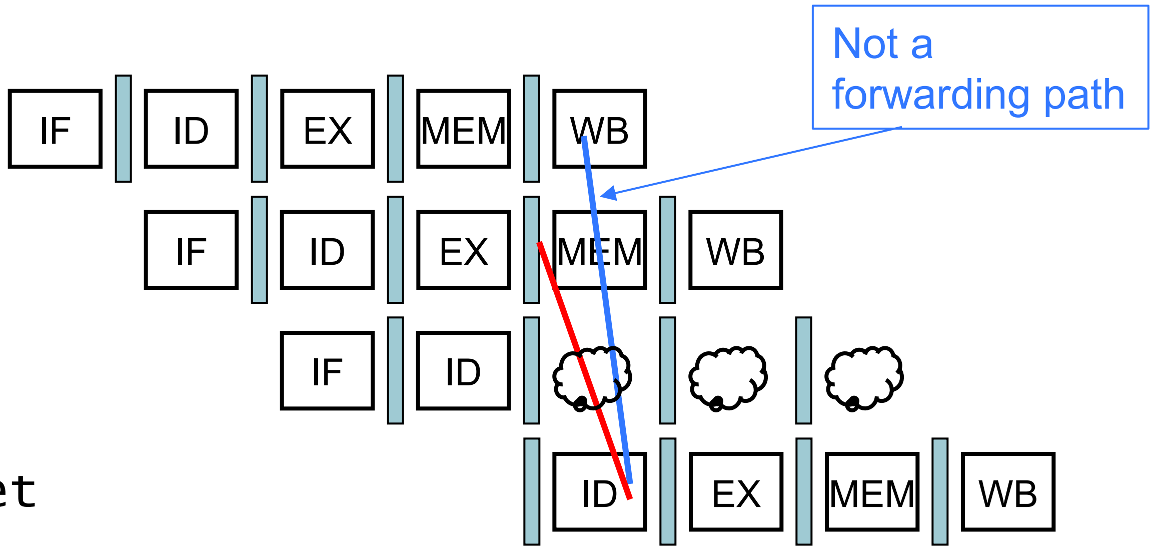
```
lw  x1, addr
add x4, $x5, $x6
beq x1, x4, target
```



Data Hazards for Branches

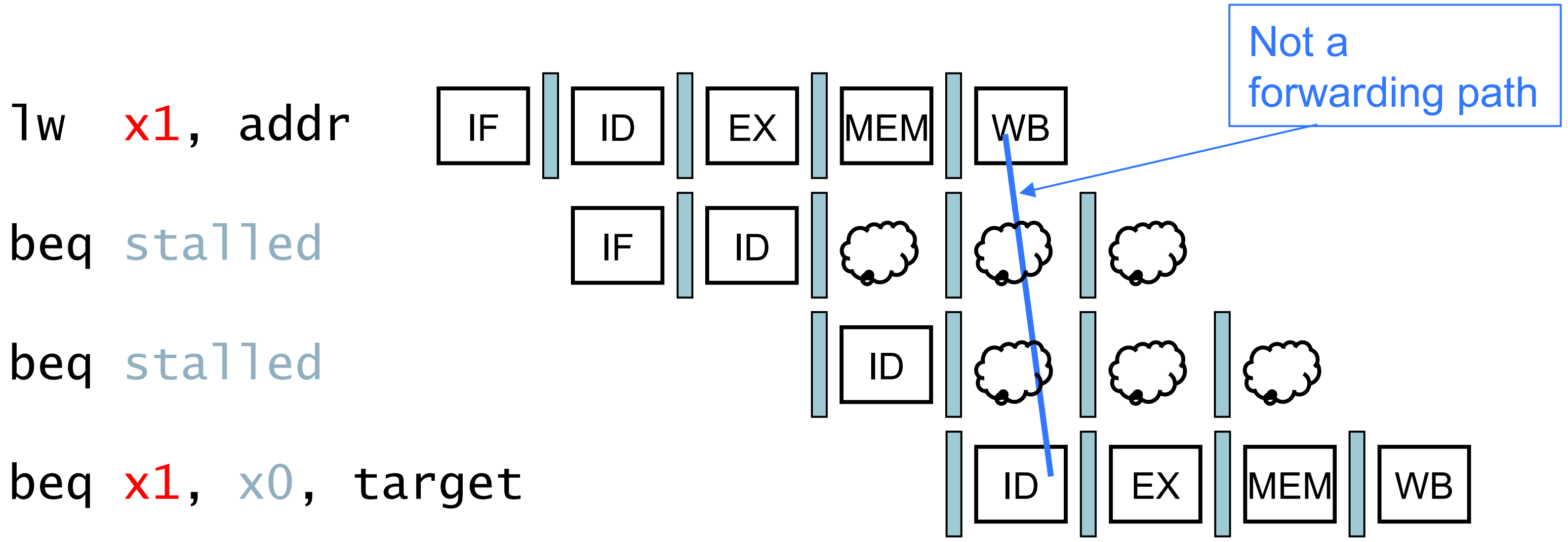
- Need 1 stall cycle even with forwarding

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



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
- *Branch prediction (instead of assumption)*

Branch Prediction

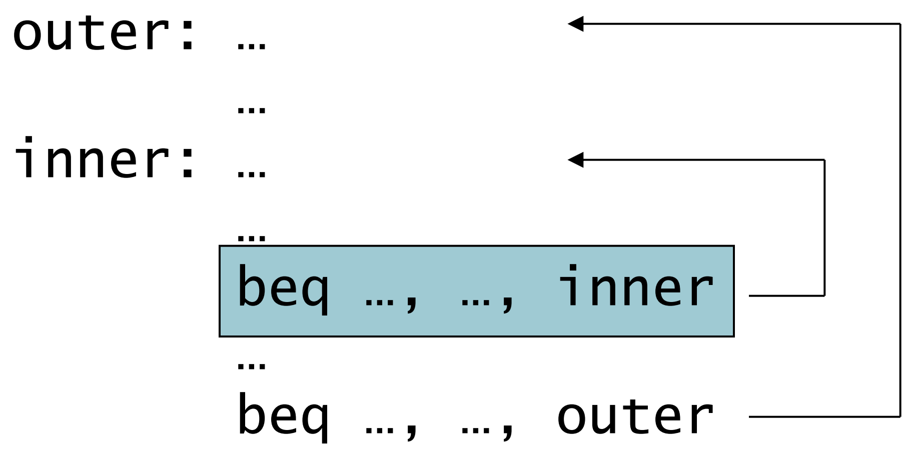
- Static prediction
  - Based on typical branch behavior
  - Example: loop and if-statement branches
    - Could predict backward branches taken
    - Could predict forward branches not taken
- Dynamic prediction
  - Hardware measures actual branch behavior
    - e.g., record recent history of each branch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

Dynamic Branch Prediction

- In deeper and superscalar pipelines, branch penalty is more significant
- Use dynamic prediction
  - 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 or target
    - If wrong, flush pipeline and flip prediction

1-Bit Predictor: Shortcoming

- Inner loop branches mispredicted twice!

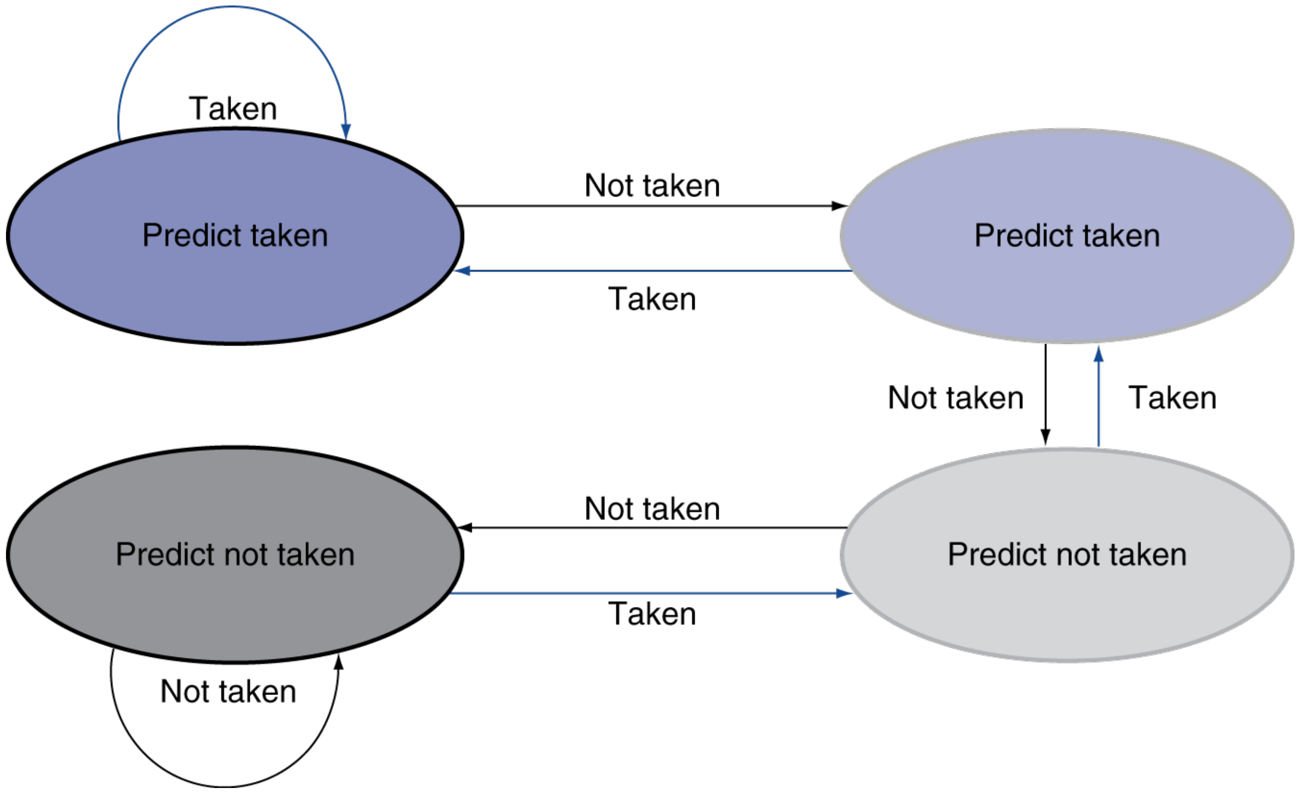


```
outer: …
       …
inner: …
       …
       beq …, …, inner
       …
       beq …, …, outer
```

  - 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


Taken
Predict taken
Not taken
Predict taken
Taken
Not taken
Taken
Not taken
Predict not taken
Predict not taken
Taken
Not taken