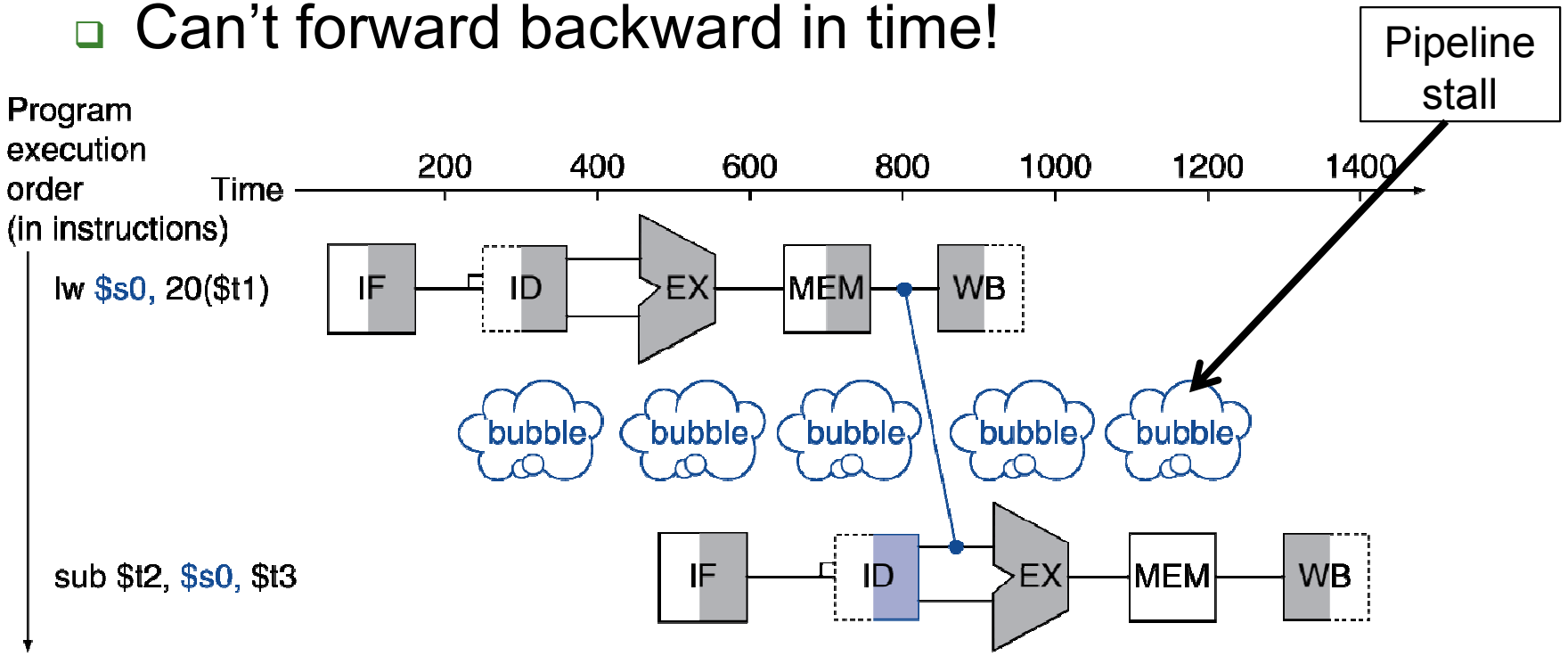
# COMPUTER ARCHITECTURE
## (CS F342)

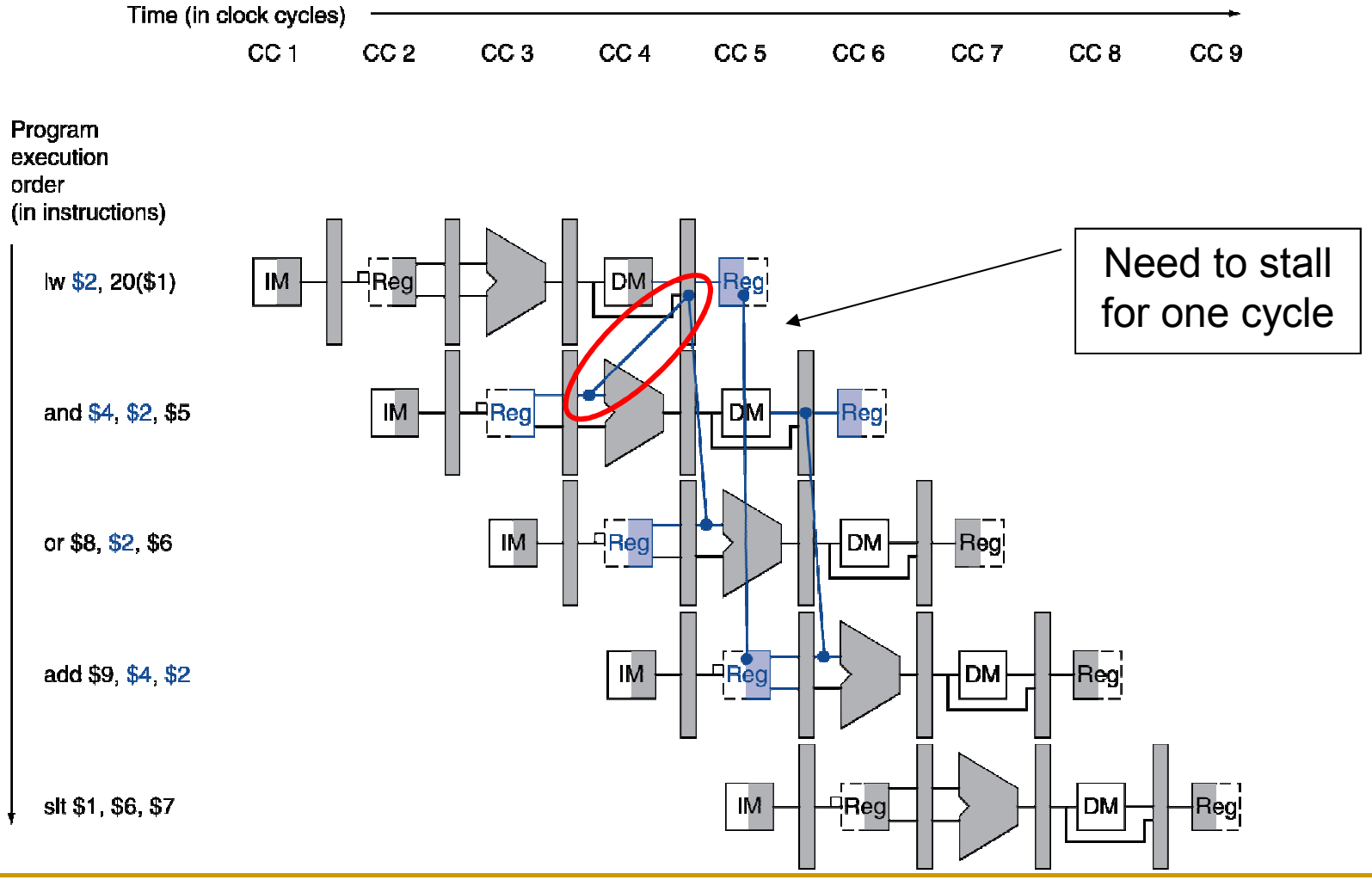
## LECT 33_34: PIPELINING

# Load-Use Data Hazard

- Can't always avoid stalls by forwarding
  - If value not computed when needed
  - Can't forward backward in time!



# Load-Use Data Hazard



# Load-Use Hazard Detection

- Check when using instruction is decoded in ID stage
  - Check for load instruction register and its use
- ALU operand register numbers in ID stage are given by
  - IF/ID.RegisterRs, IF/ID.RegisterRt
- Check for load instruction. Load-use hazard when
  - If (ID/EX.MemRead and
    ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
    (ID/EX.RegisterRt = IF/ID.RegisterRt)))

    Stall the pipeline

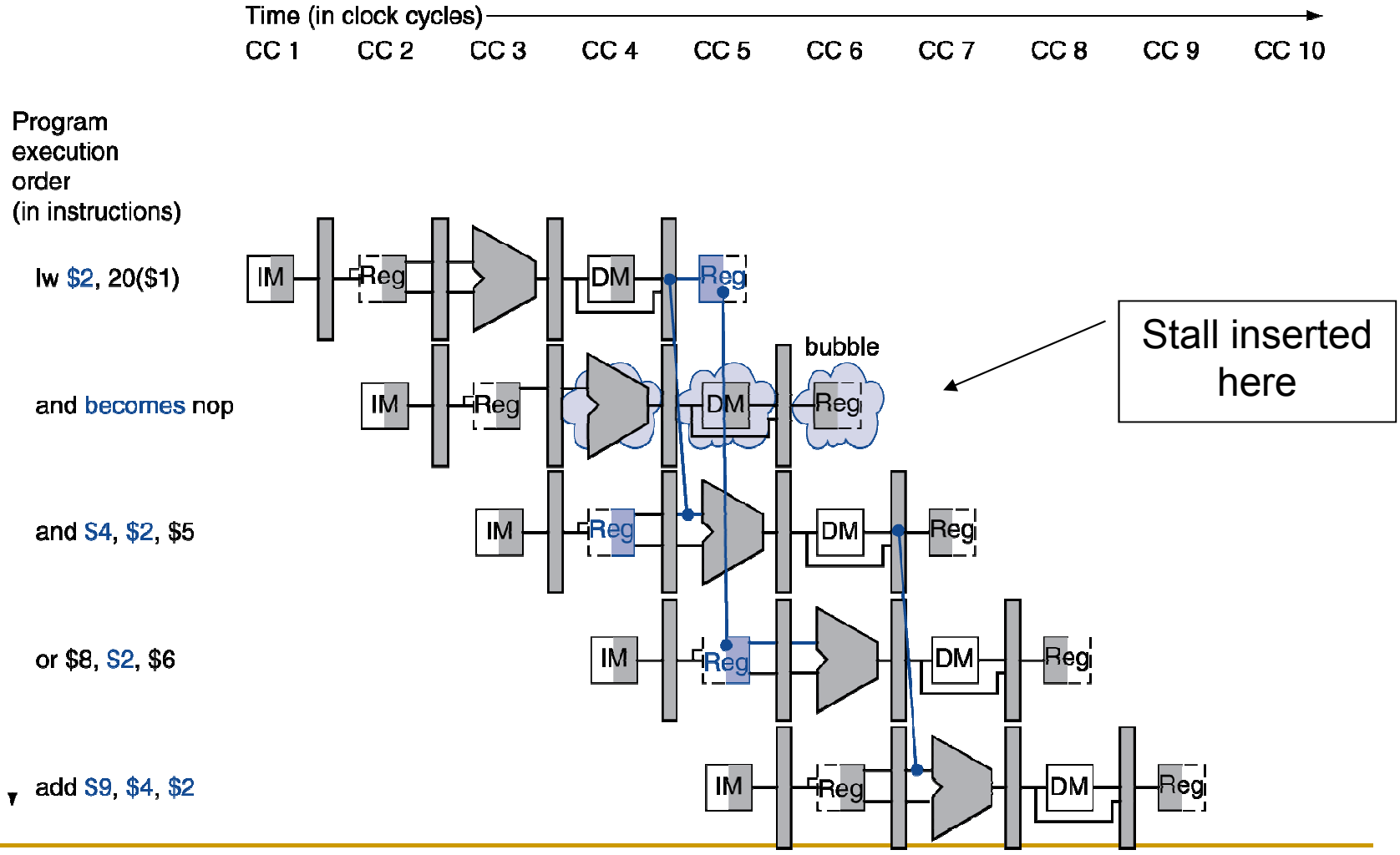
# Stall an instruction in ID stage

- If the instruction in ID stage is stalled the instruction in IF stage must also be stalled
  - Otherwise we will lose the fetched instruction
- How can we achieve this??

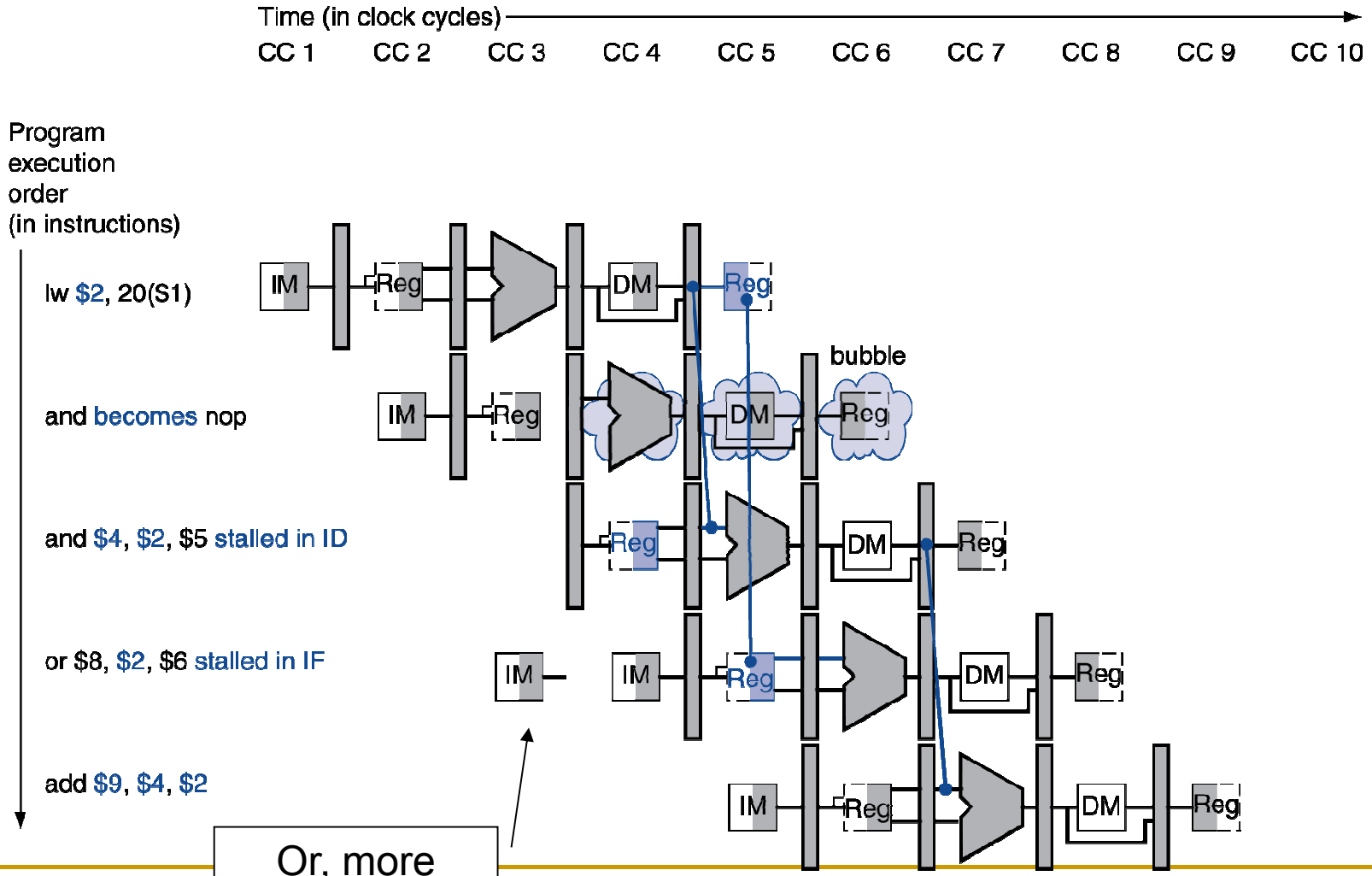
# How to Stall the Pipeline

- Force control values in ID/EX register to 0
  - EX, MEM and WB do nop (no-operation)
- Prevent update of PC and IF/ID register
  - Using instruction is decoded again
  - Following instruction is fetched again
  - 1-cycle stall allows MEM to read data for lw
    - Can subsequently forward to EX stage

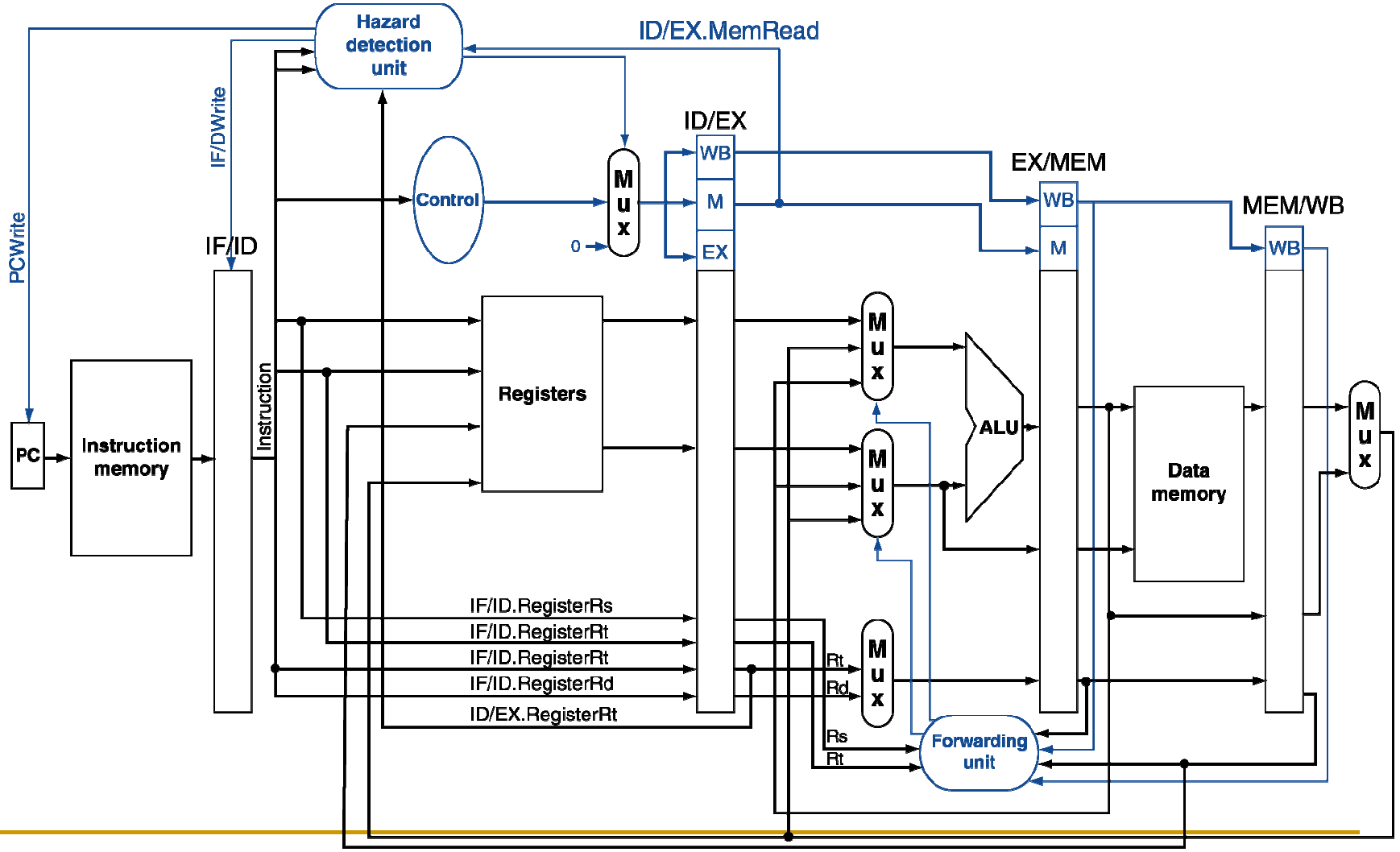
# Stall/Bubble in the Pipeline



# Stall/Bubble in the Pipeline



# Datapath with Hazard Detection

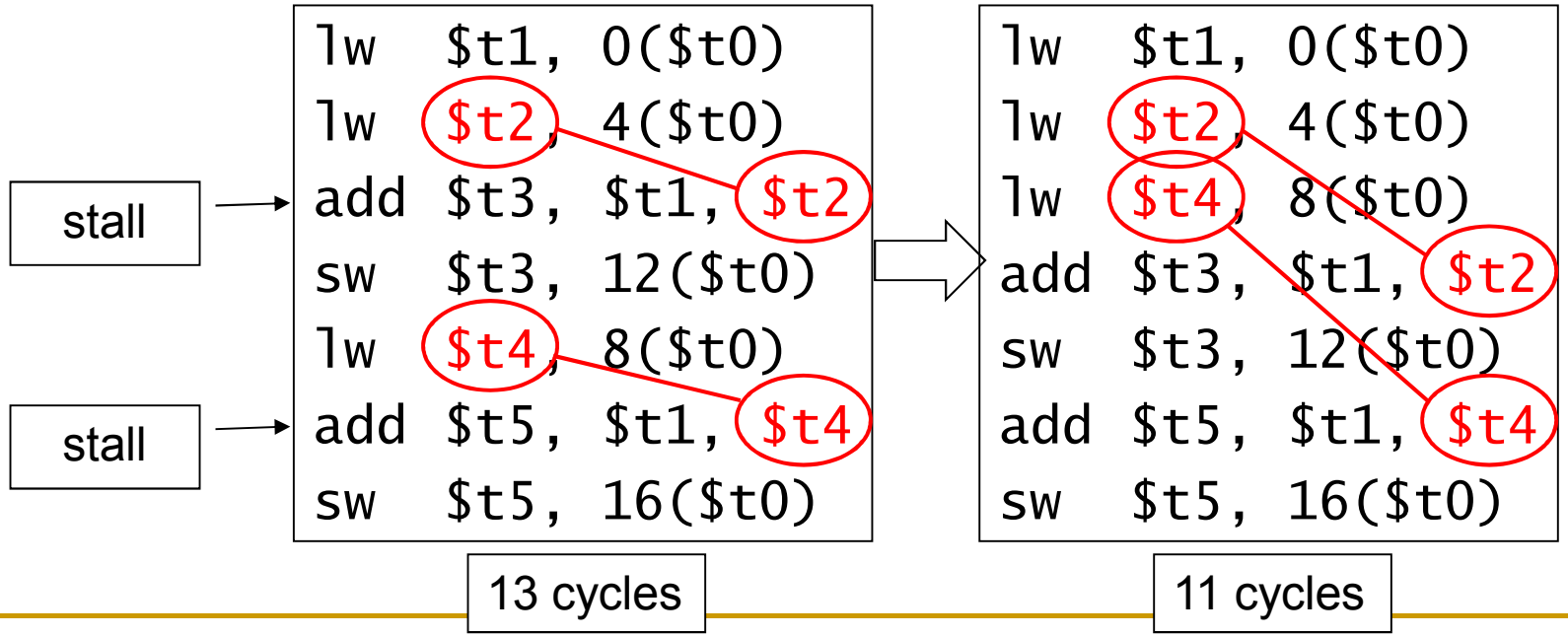


# Stalls and Performance

- Stalls reduce performance
  - But are required to get correct results
- Compiler can arrange code to avoid hazards and stalls
  - Requires knowledge of the pipeline structure

# Code Scheduling to Avoid Stalls

- Reorder code to avoid use of load result in the next instruction
- C code for `A = B + E; C = B + F;`

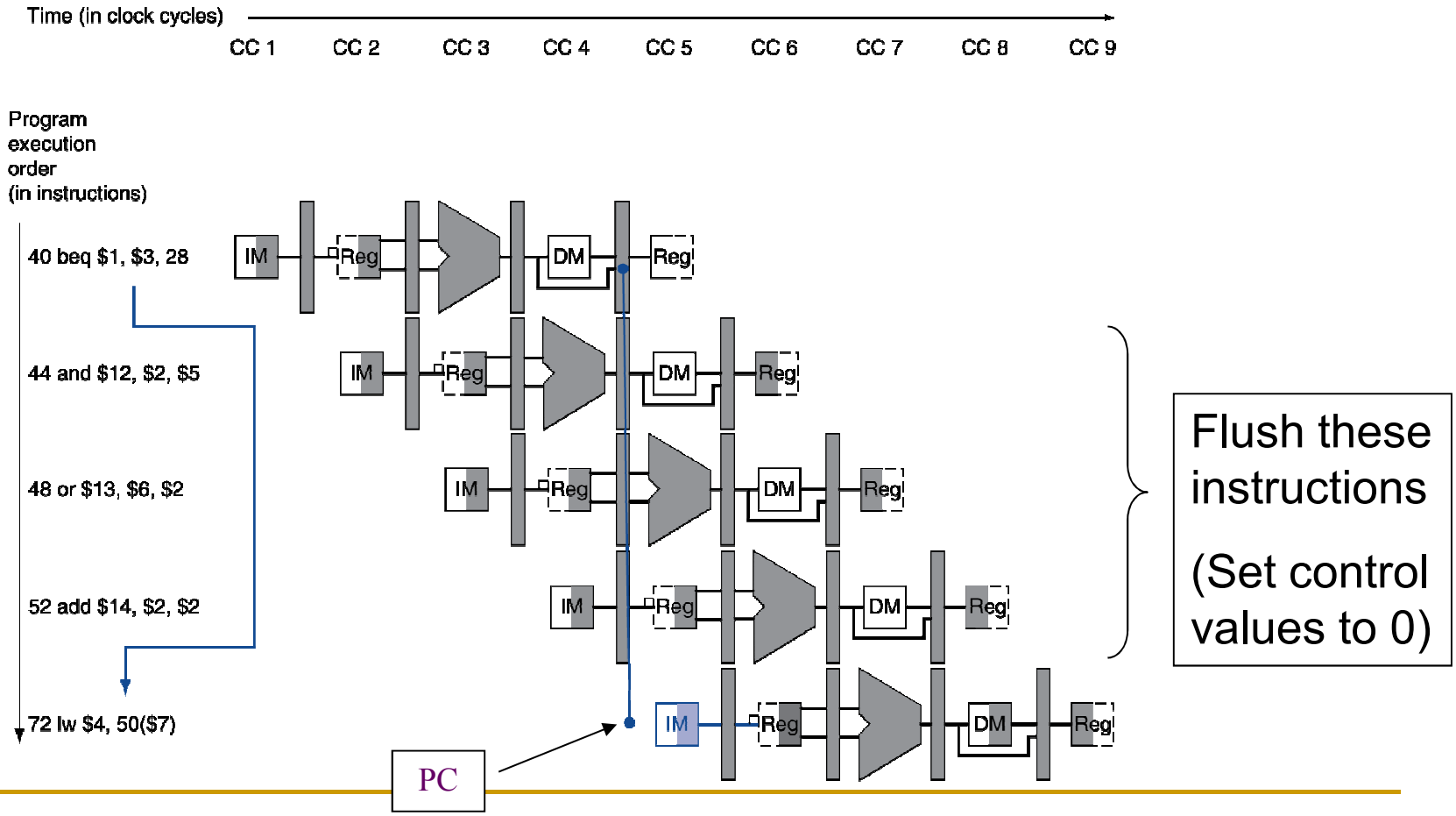


# Control Hazards or Branch Hazards

- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipeline can't always fetch correct instruction

# Branch Hazards

- If branch outcome determined in MEM



- In MIPS pipeline
  - Need to compare registers and compute target early in the pipeline
  - Add hardware to do it in ID stage

# Reducing Branch Delay

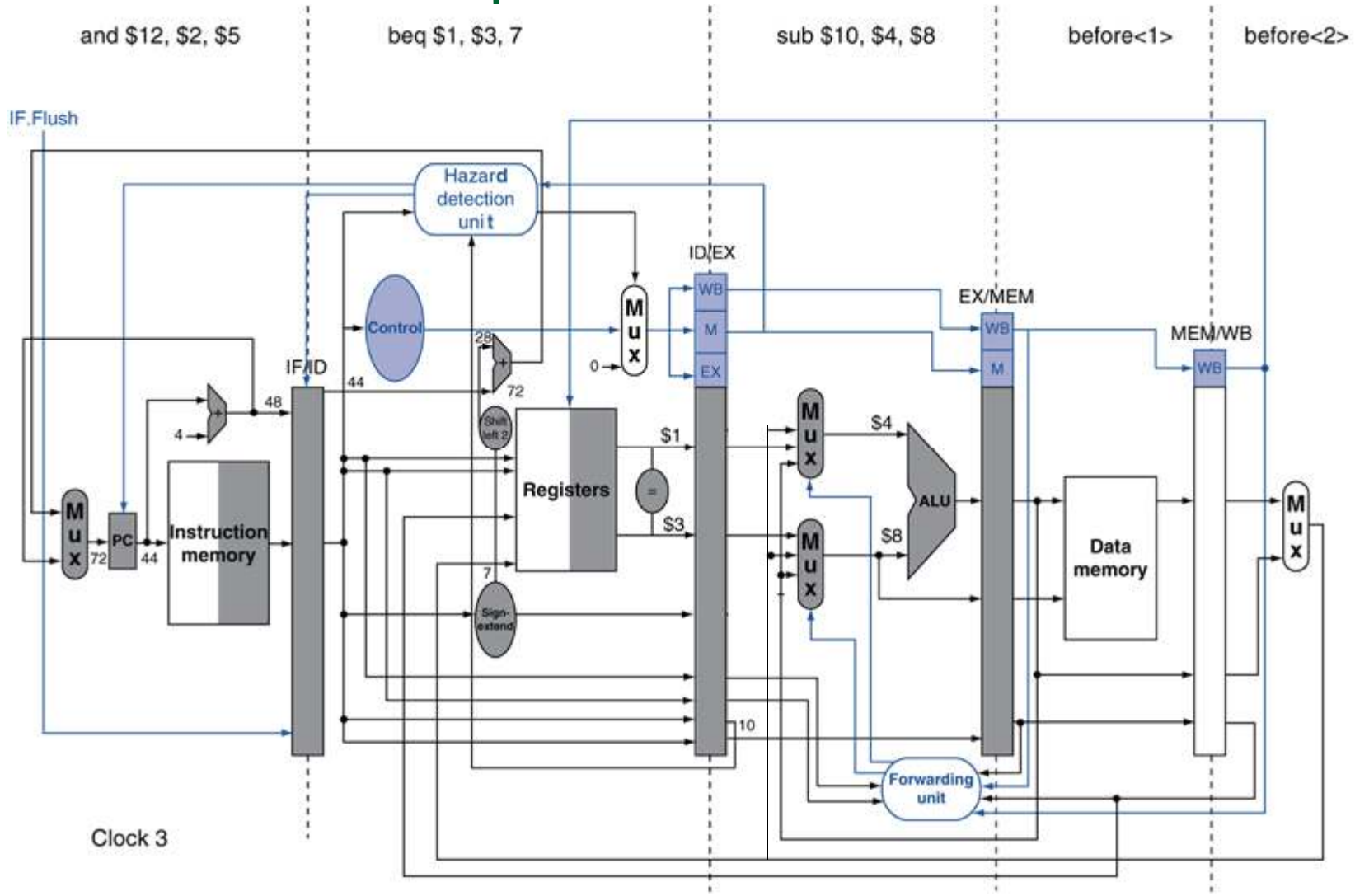
- Move hardware to determine outcome to ID stage. Requires 2 actions to occur earlier
  - Computing branch target address
    - Move the branch target address calculation from EX to ID
    - PC and immediate field in IF/ID pipeline register is used for it
    - Disadvantage: branch target address calculation will be performed for all instructions (used only when needed)
  - Evaluating the branch decision
    - Branch equal
      - Compare 2 registers read during ID stage to see they are equal
      - XOR their respective bits and ORing all the results

# Pipelined Branch

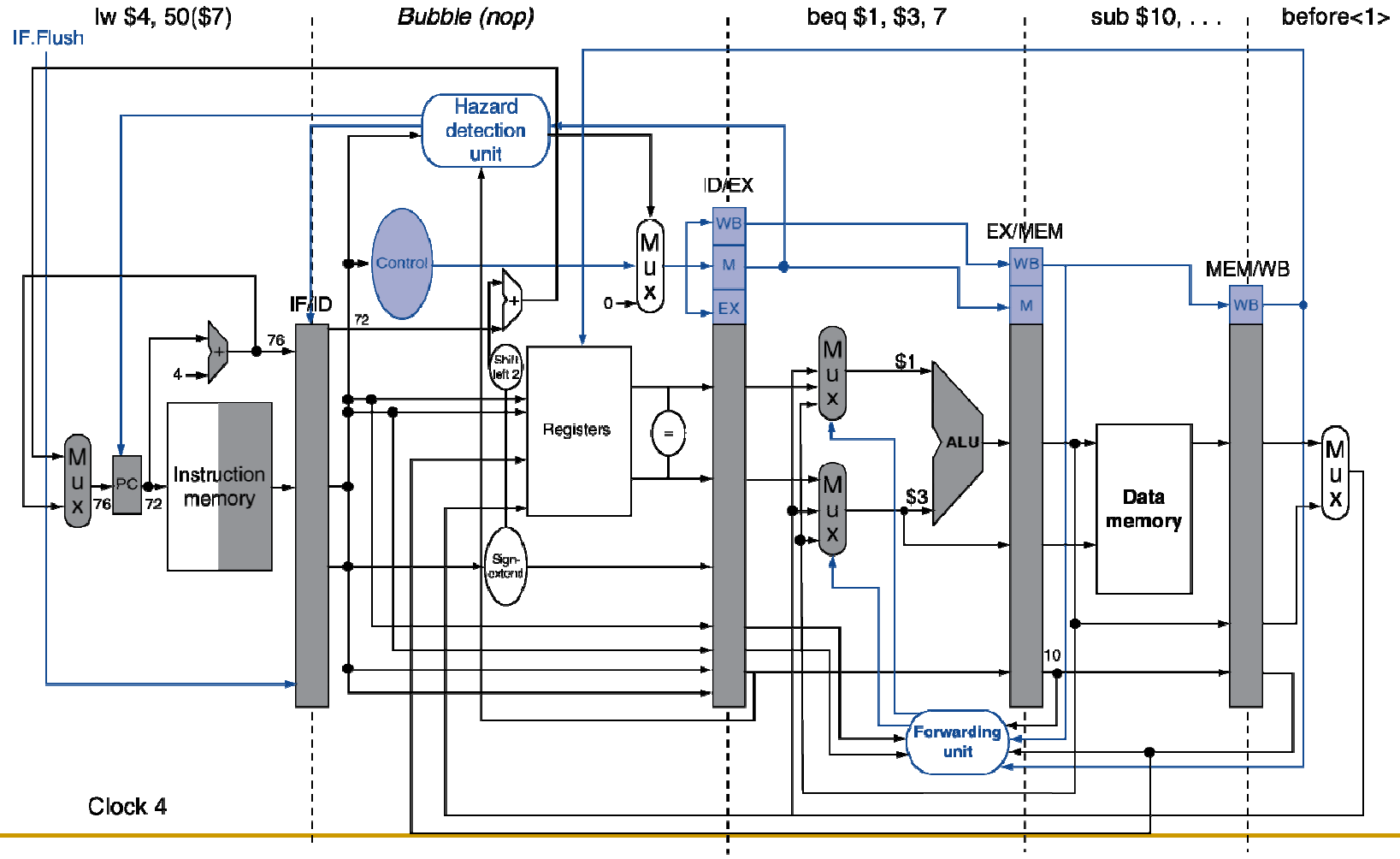
- Advantages of moving branch to ID stage
  - Reduces the penalty of branch to only one instruction if the branch is taken
- To flush instruction in control stage, a control line called IF.Flush is added
  - Zeroes the IF/ID pipeline register
- Example: branch taken

```
36:   sub   $10, $4, $8
40:   beq   $1,  $3, 7
44:   and   $12, $2, $5
48:   or    $13, $2, $6
52:   add   $14, $4, $2
56:   slt   $15, $6, $7
      ...
72:   lw    $4, 50($7)
```

# Example: Branch Taken



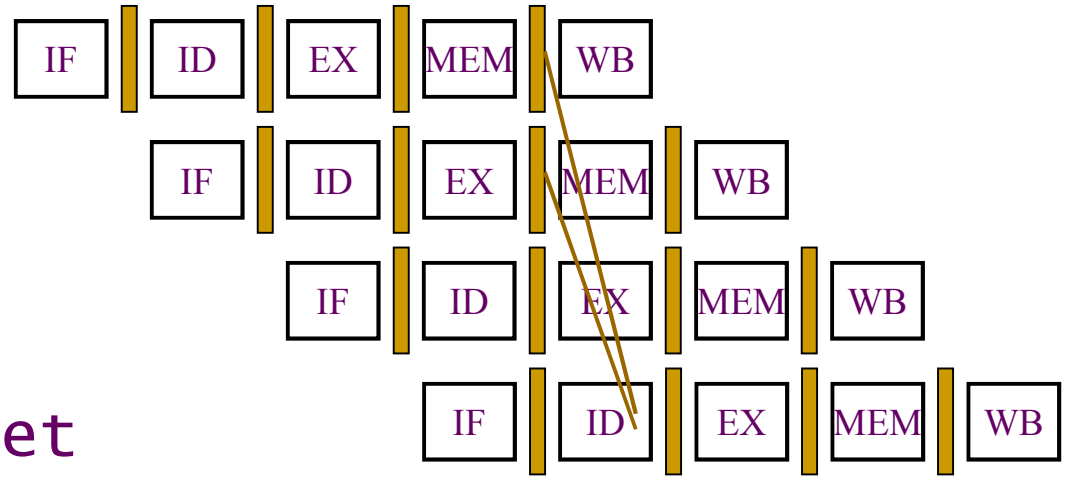
# Example: Branch Taken



# Data Hazards for Branches

- If a comparison register is a destination of 2nd or 3rd preceding ALU instruction

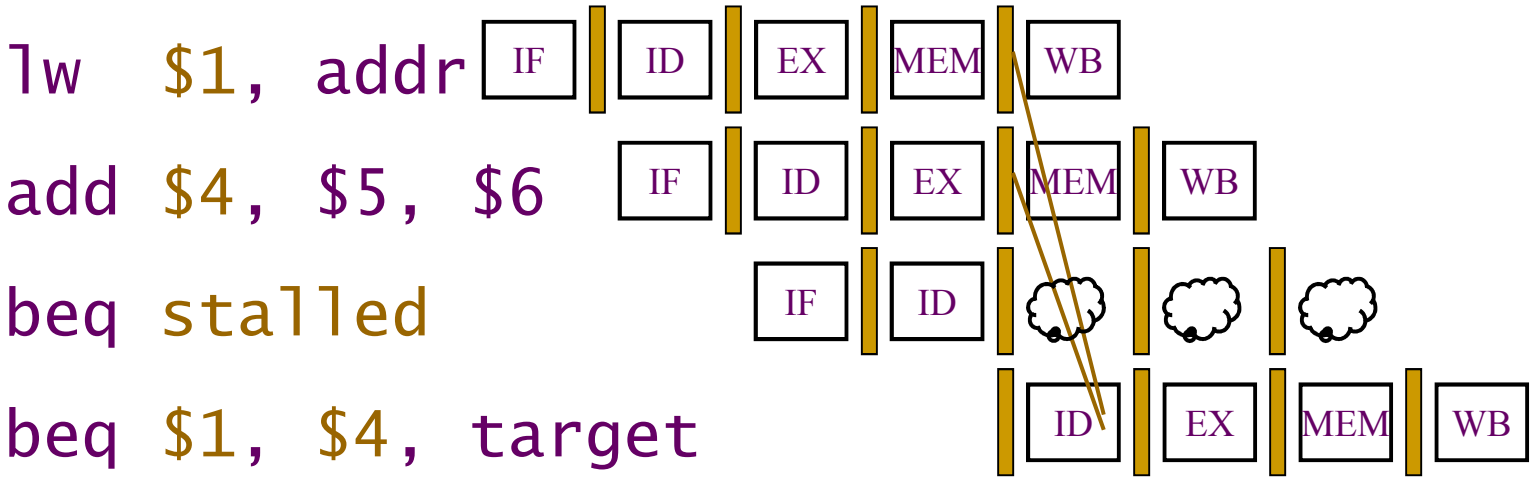
```
add $1, $2, $3
add $4, $5, $6
…
beq $1, $4, target
```




- Can resolve using forwarding

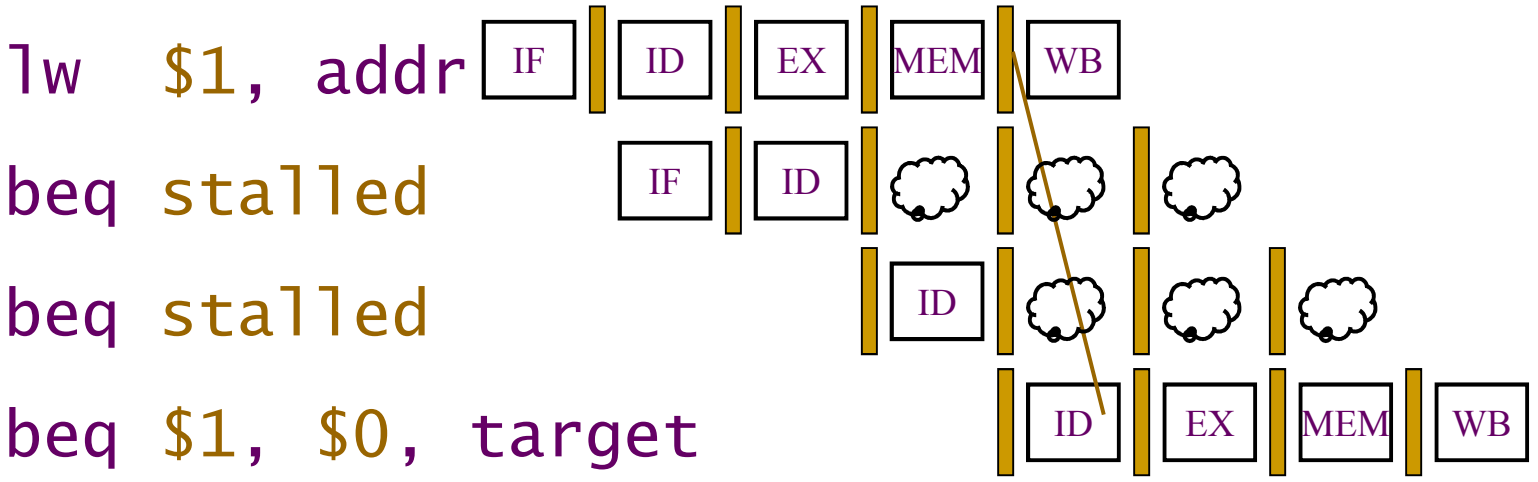
# Data Hazards for Branches

- If a comparison register is a destination of preceding ALU instruction or 2nd preceding load instruction
  - Need 1 stall cycle



# Data Hazards for Branches

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



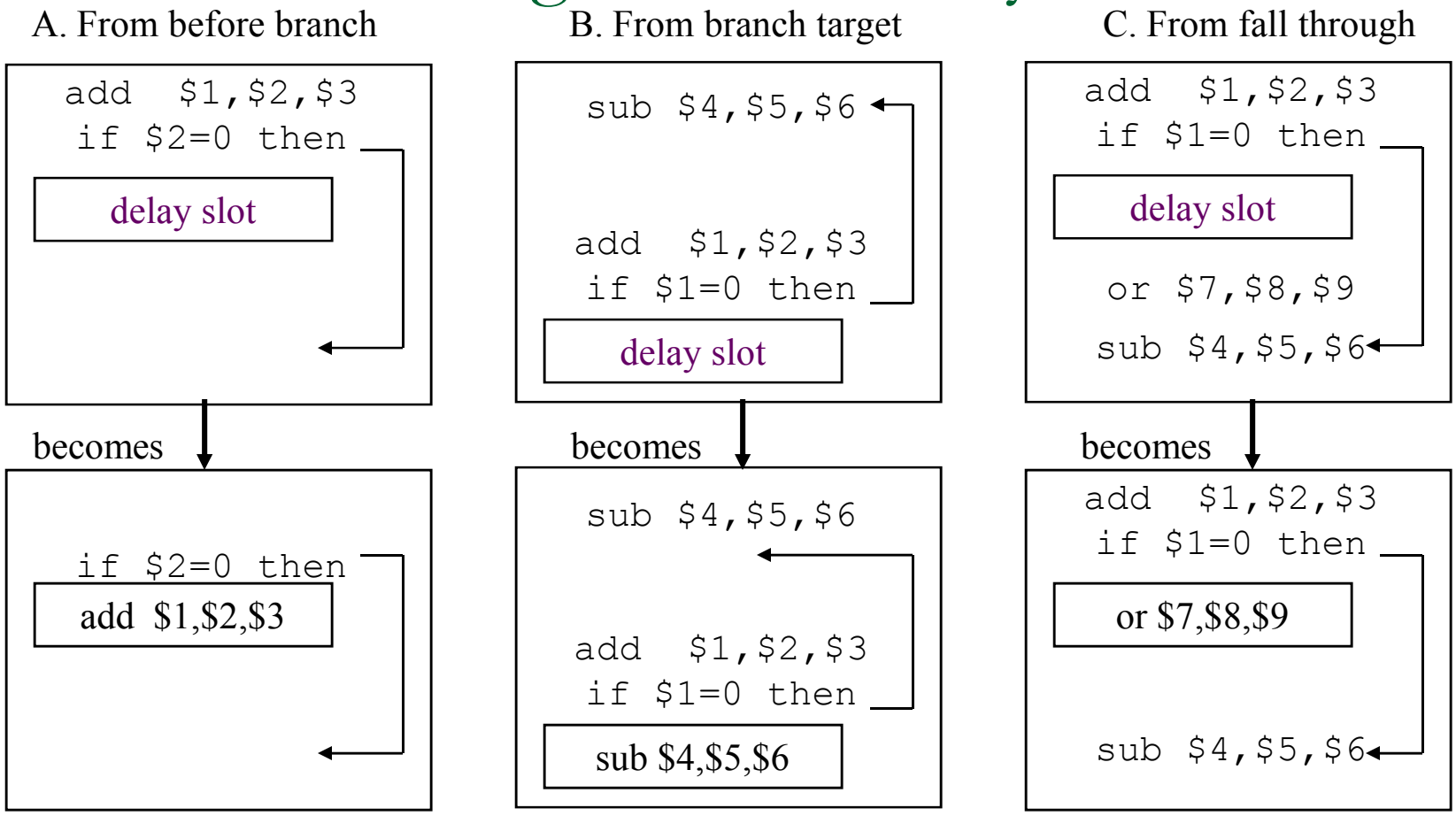
# Handling Conditional Branches in Pipeline

- Multiple Streams
  - Have two pipelines
  - Prefetch each branch into a separate pipeline
  - Use appropriate pipeline
  - Drawbacks
    - Leads to bus & register contention
    - Multiple branches lead to further pipelines being needed
    - Used in IBM 370/168
- Pre-fetch Branch Target
  - Target of branch is prefetched in addition to instructions following branch
  - Keep target until branch is executed
  - Used by IBM 360/91

# Delayed Branches

- In MIPS, branch takes place in ID stage
  - we can eliminate all branch stalls with delayed branches
  - Delayed branch is defined as always executing the next sequential instruction after the branch instruction – the branch takes effect *after* that next instruction
  - MIPS compiler moves an instruction to immediately after the branch that is not affected by the branch (a safe instruction) thereby hiding the branch delay

# Scheduling Branch Delay Slots



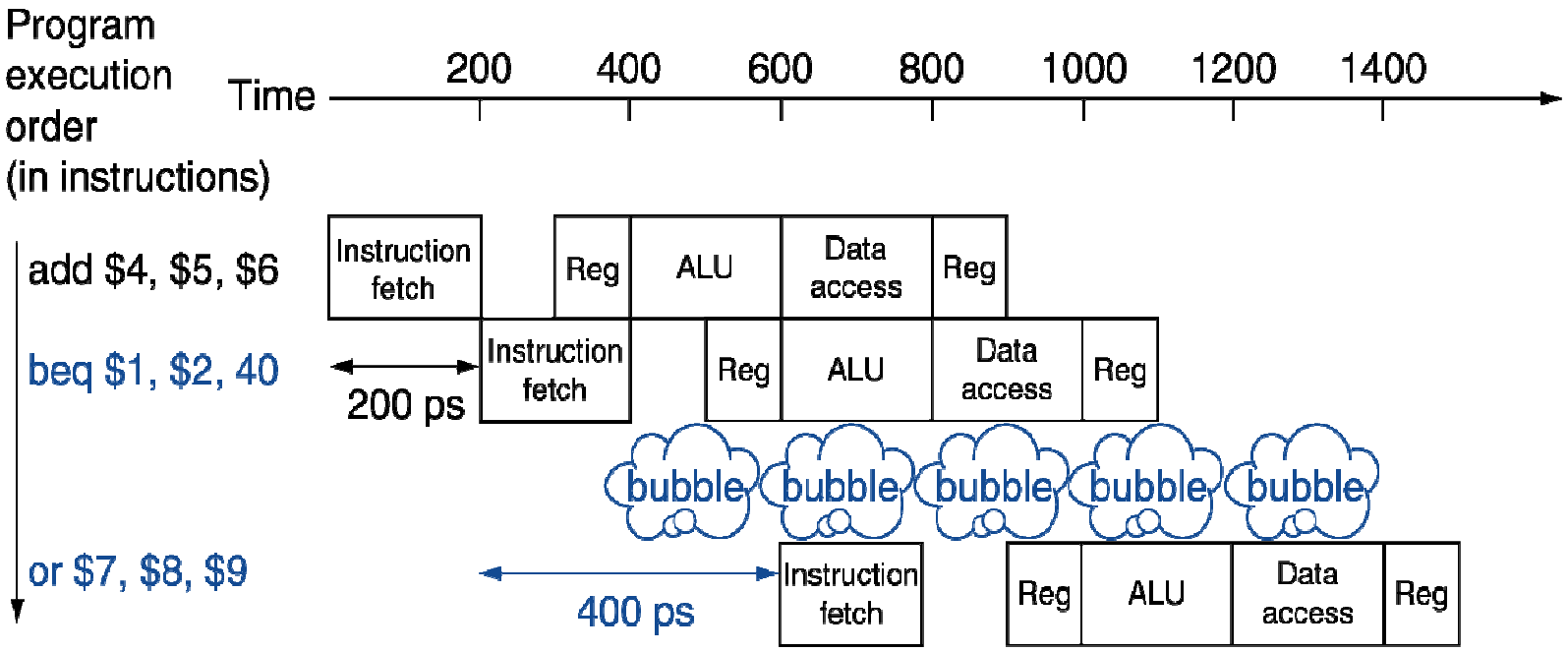

- A is the best choice, fills delay slot and reduces IC
- In B, the `sub` instruction may need to be copied, increasing IC. Preferred when the branch taken probability is high
- In C, must be okay to execute or when branch fails

# Branch Prediction

- Longer pipelines can't readily determine branch outcome early
  - Stall penalty becomes unacceptable
- Predict outcome of branch
  - Only stall if prediction is wrong
- In MIPS pipeline
  - Can predict branches not taken
  - Fetch instruction after branch, with no delay

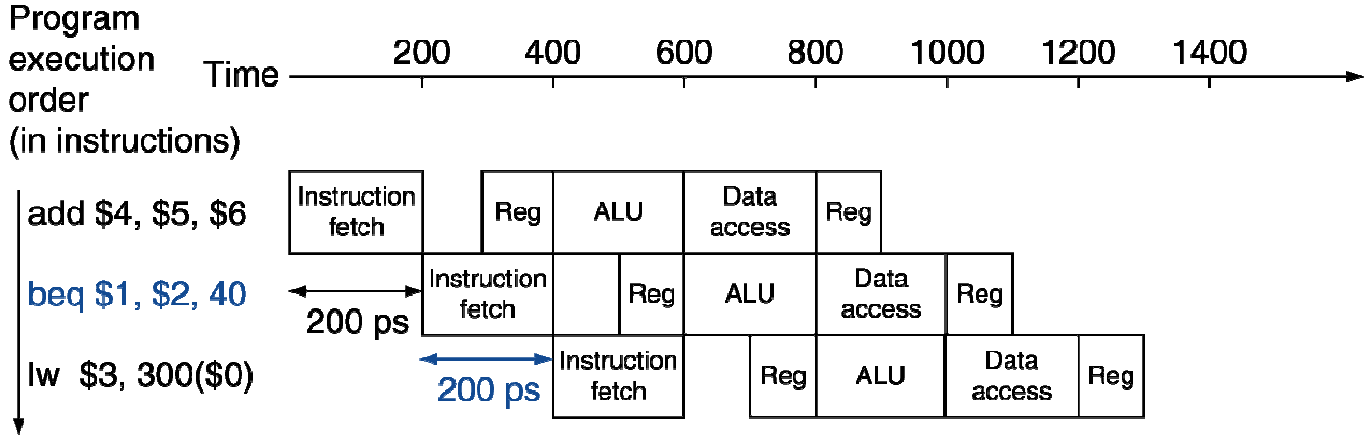
# Stall on Branch

- Wait until branch outcome determined before fetching next instruction

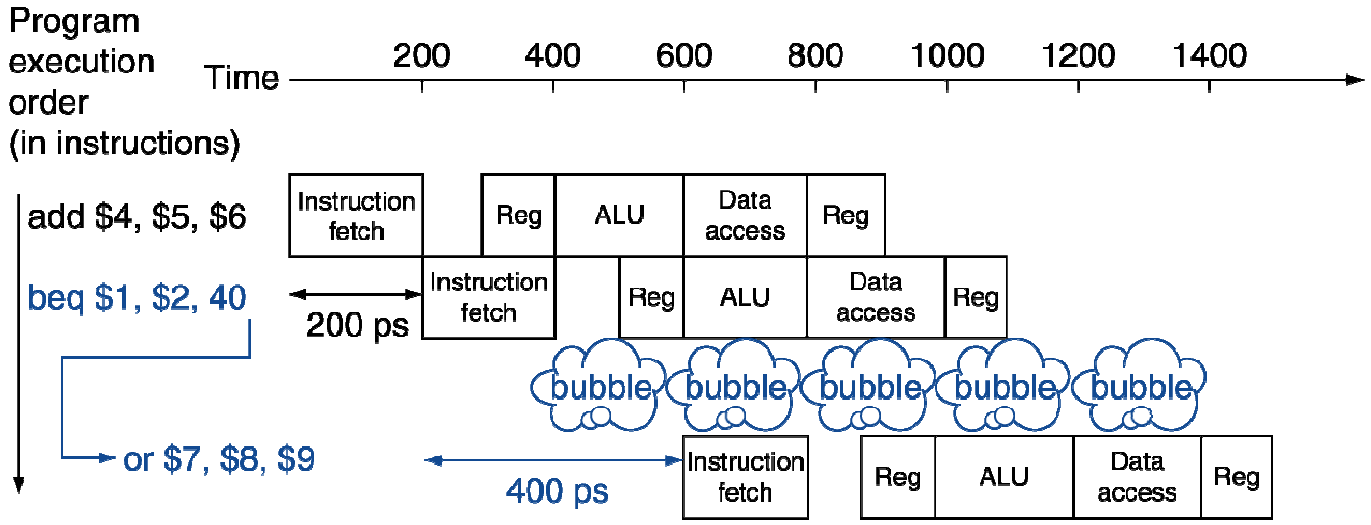


# MIPS with Predict Not Taken

Prediction correct




Prediction incorrect



# More-Realistic Branch Prediction

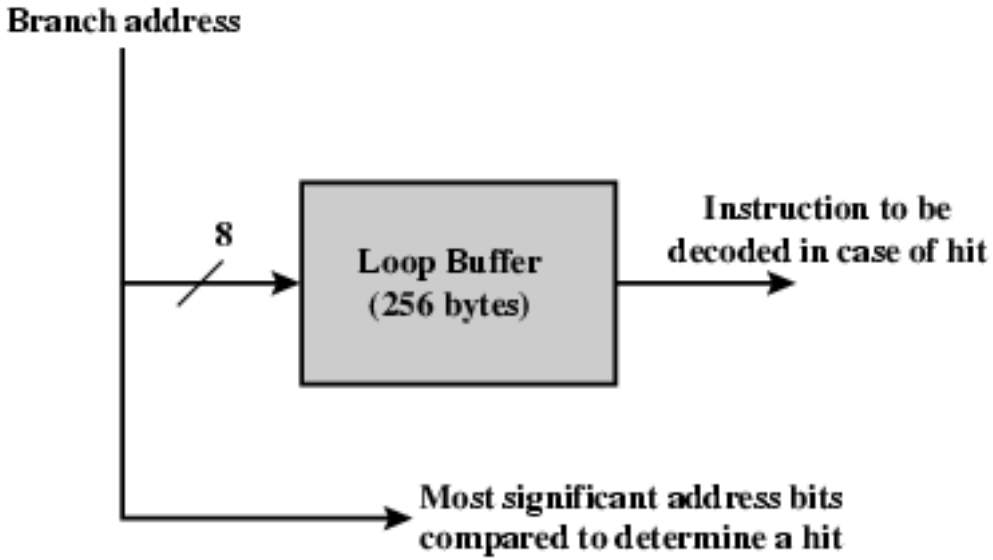
- Static branch prediction
  - Based on typical branch behavior
  - Example: loop and if-statement branches
    - Predict backward branches taken
    - Predict forward branches not taken
- Dynamic branch prediction
  - Hardware measures actual branch behavior
    - e.g., record recent history of each branch
  - Assume future behavior will continue the trend
    - When wrong, stall while re-fetching, and update history

# Handling Conditional Branches in Pipeline

- In deeper and superscalar pipelines, branch penalty is more significant
- Use dynamic prediction
  - Prediction of branches at runtime using runtime information

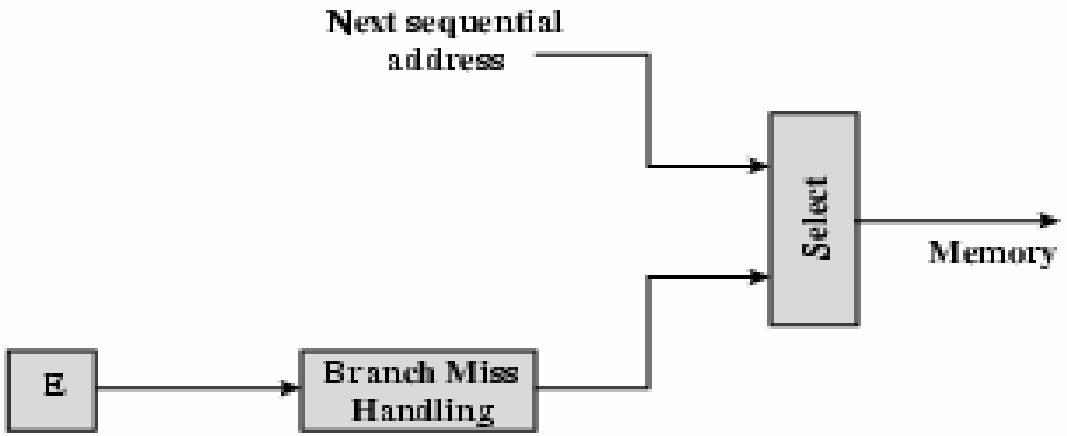
# Handling Conditional Branches in Pipeline

- Loop Buffer
  - Very fast memory. In principle same to cache.
  - Maintained by fetch stage of pipeline
  - Check buffer before fetching from memory
  - Very good for small loops or jumps
  - Similar to a cache. Used by CRAY-1



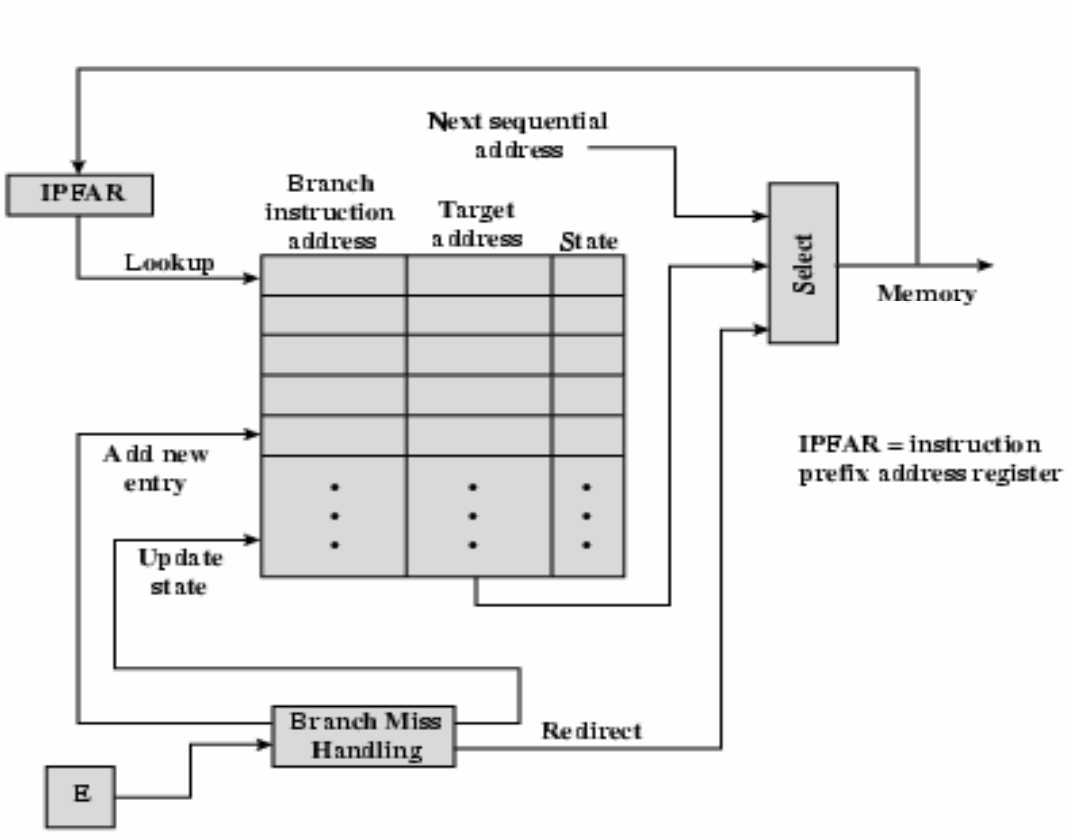
# Dynamic Branch Prediction

- Branch prediction
  - Predict never taken
    - Assume that jump will not happen
    - Always fetch next instruction
    - 68020 & VAX 11/780
    - VAX will not prefetch after branch if a page fault would result



# Dynamic Branch Prediction

- Taken/Not taken switch
  - Based on previous history
  - Good for loops



# Dynamic Branch Prediction

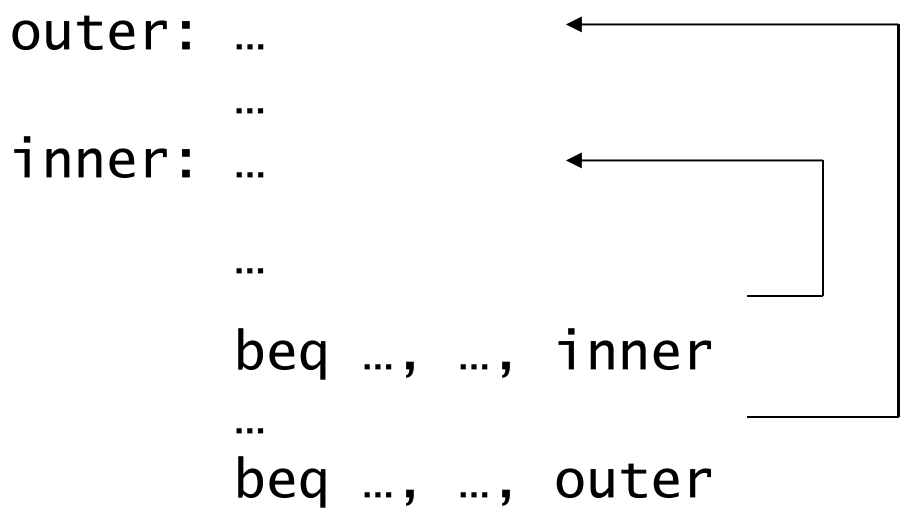
- Predict always taken
  - Assume that jump will happen
  - Always fetch target instruction
- Predict by Opcode
  - Some instructions are more likely to result in a jump than others
  - Can get up to 75% success

# Dynamic Branch Prediction

- Use dynamic prediction
  - Prediction of branches at runtime using runtime information
  - Branch prediction buffer (aka branch history table)
  - Memory contains a bit that says whether branch was taken not.
    - Another branch with same lower order address bits will update this memory. So the prediction may not be the right one.
  - To execute a branch
    - Check table, expect the same outcome
    - Start fetching from fall-through or target
    - If wrong, flush pipeline and flip prediction

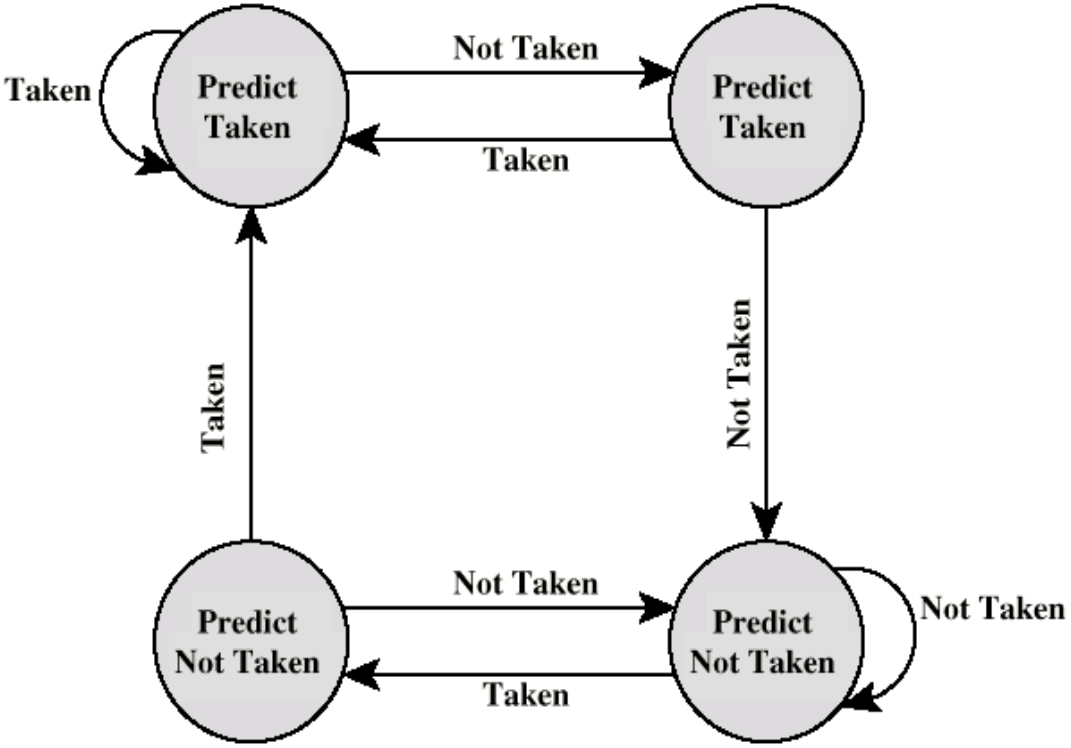
# 1-Bit Predictor: Shortcoming

- Inner loop branches mispredicted twice!

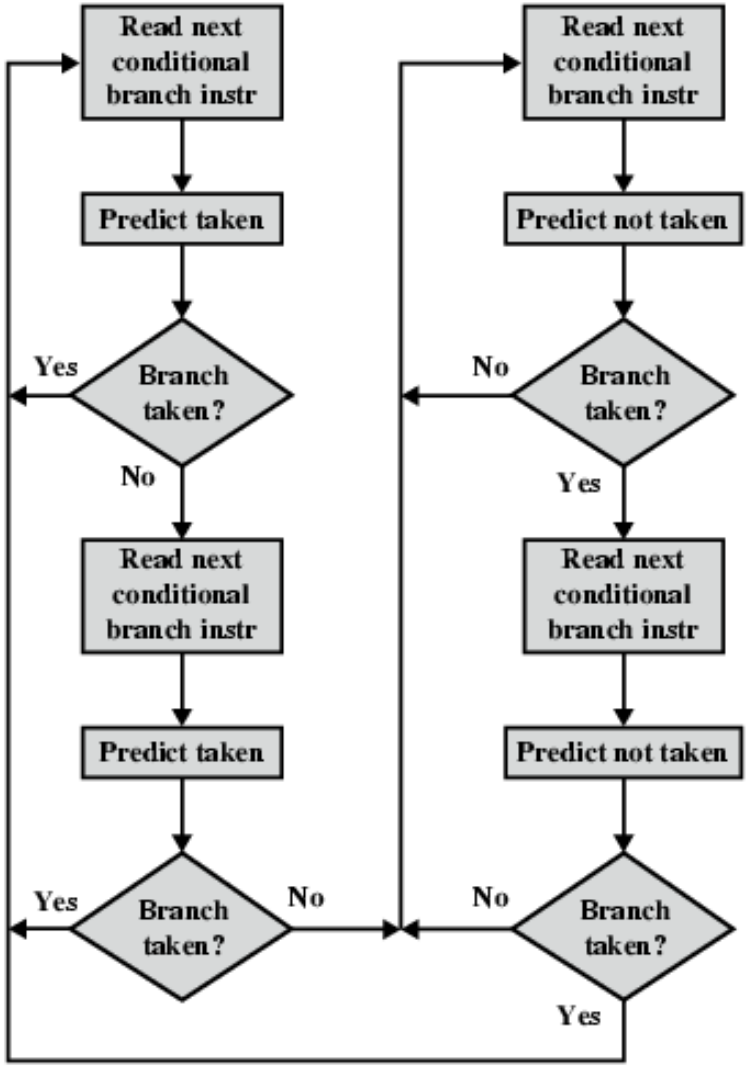



- Mispredict as taken on last iteration of inner loop
- Then mispredict as not taken on first iteration of inner loop next time around

# Branch Prediction State Diagram



# Prediction by Branch History table



# Branch Predictors

- Branch target buffer
  - A structure that caches the destination PC or destination instruction for a branch
  - This avoids 1 cycle penalty (avoids calculation time of branch target)
- Correlating predictor
  - Combines local behavior of a particular branch and global information about the behavior of some recent number of executed branches
- Tournament branch predictor
  - A branch predictor with multiple predictors for each branch and a selection mechanism that chooses which predictor to enable for a given branch

# Pipeline Performance a relook

- Pipeline CPI = Ideal Pipeline CPI + Structural Stalls + Data Hazard Stalls + Control Stalls