# Final QuickFire Review

Fall 2012

CS 465

Huzefa Rangwala, PhD

Chapters 1 - 6

# Disclaimer:

These slides are not a substitute for reading the book.

To do well: Go over the detailed readings, solve exercises from HW, Quizzes, Classroom Exercises and back of the book*.

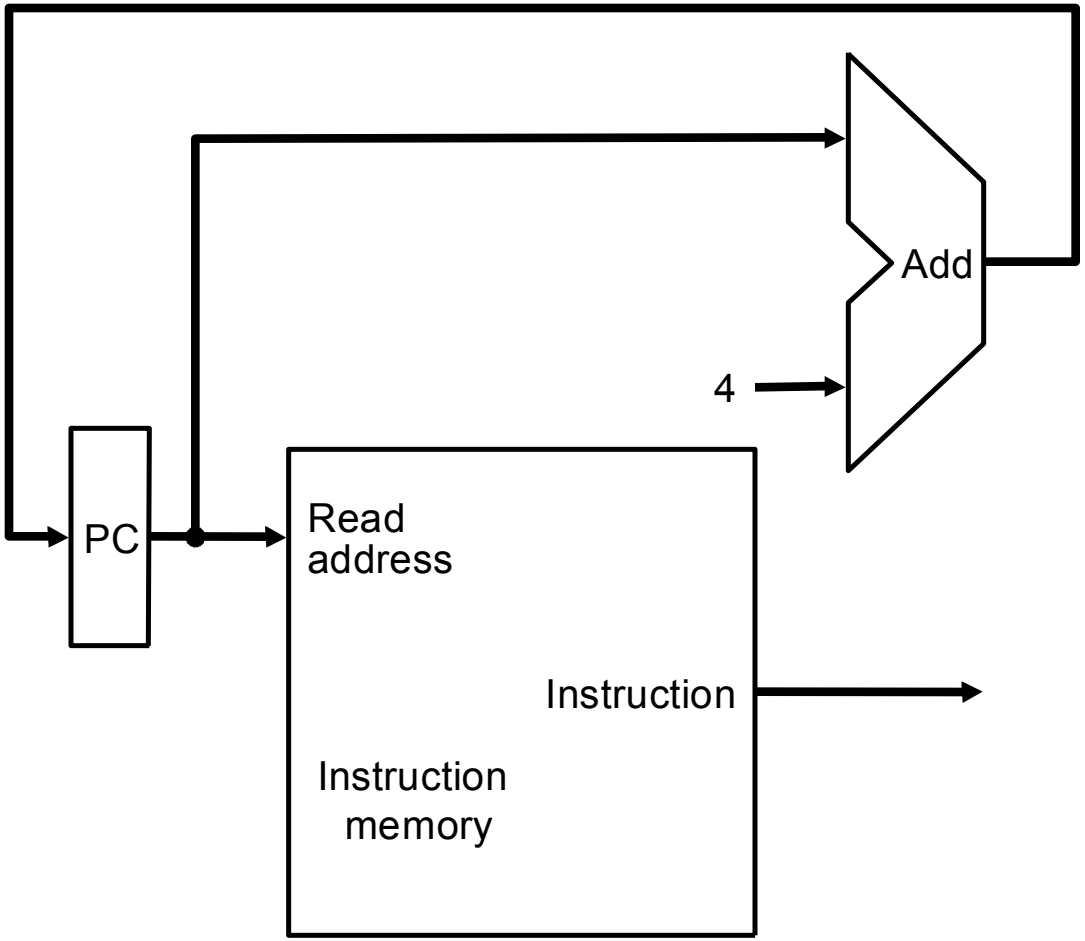
# Detailed Readings

- Look on piazza or class web-site

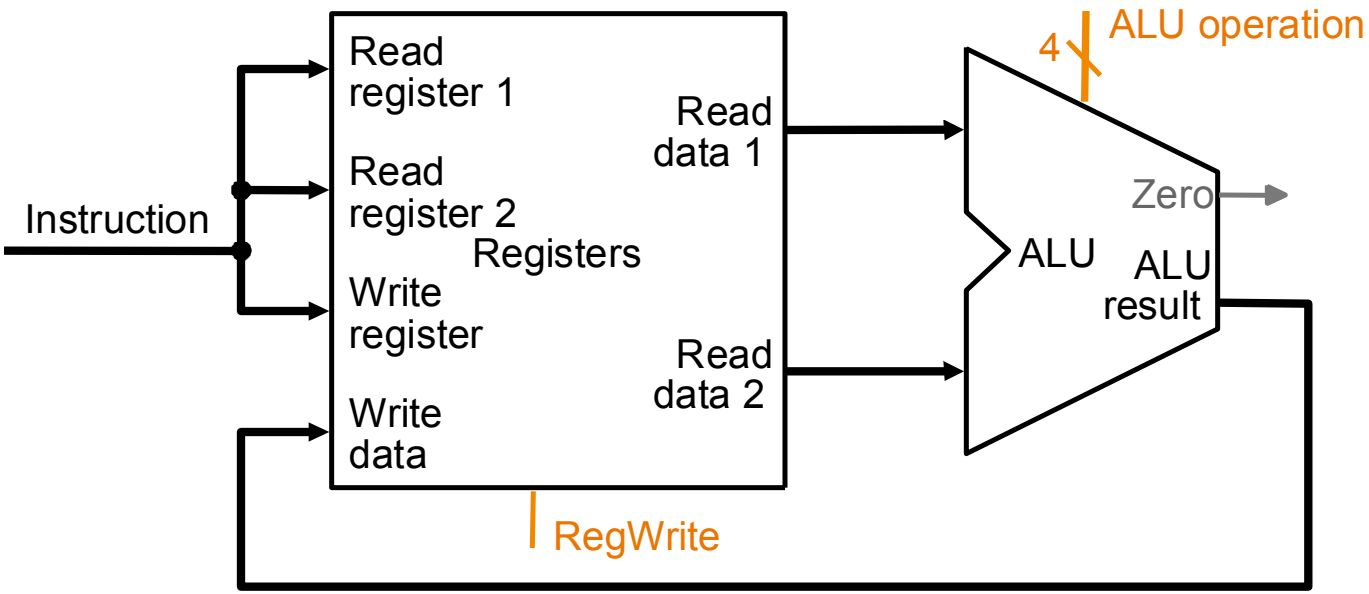
# How to Design a Processor

- Analyze instruction set ⇒ datapath requirements
  - Meaning of each instruction given by register transfers
  - Datapath must include storage element for ISA registers (possibly more)
  - Datapath must support each register transfer
- Select set of datapath components and establish clocking methodology
- Assemble datapath meeting the requirements
- Analyze implementation of each instruction to determine setting of control points that effects the register transfer
- Assemble the control logic

# Datapath for Instruction Fetch

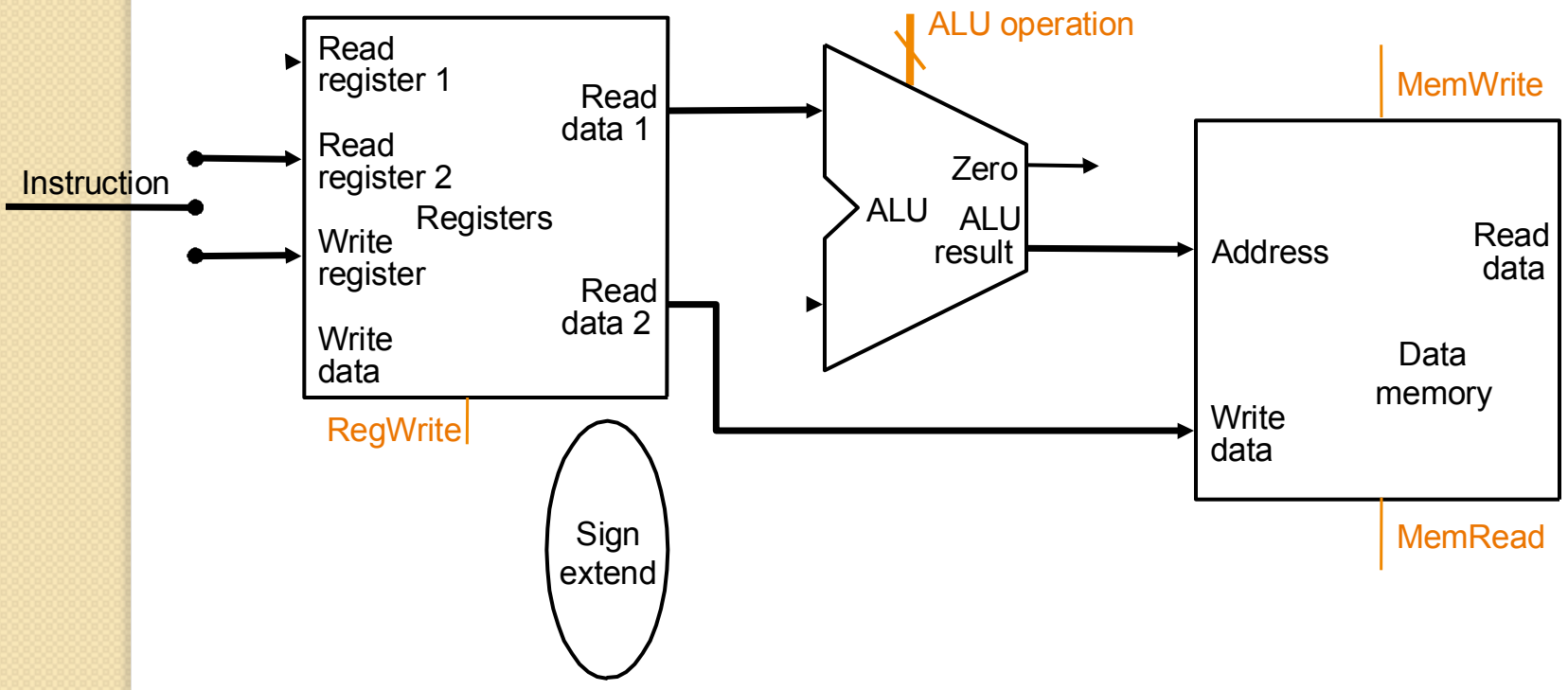


# Datapath for R-format Instructions

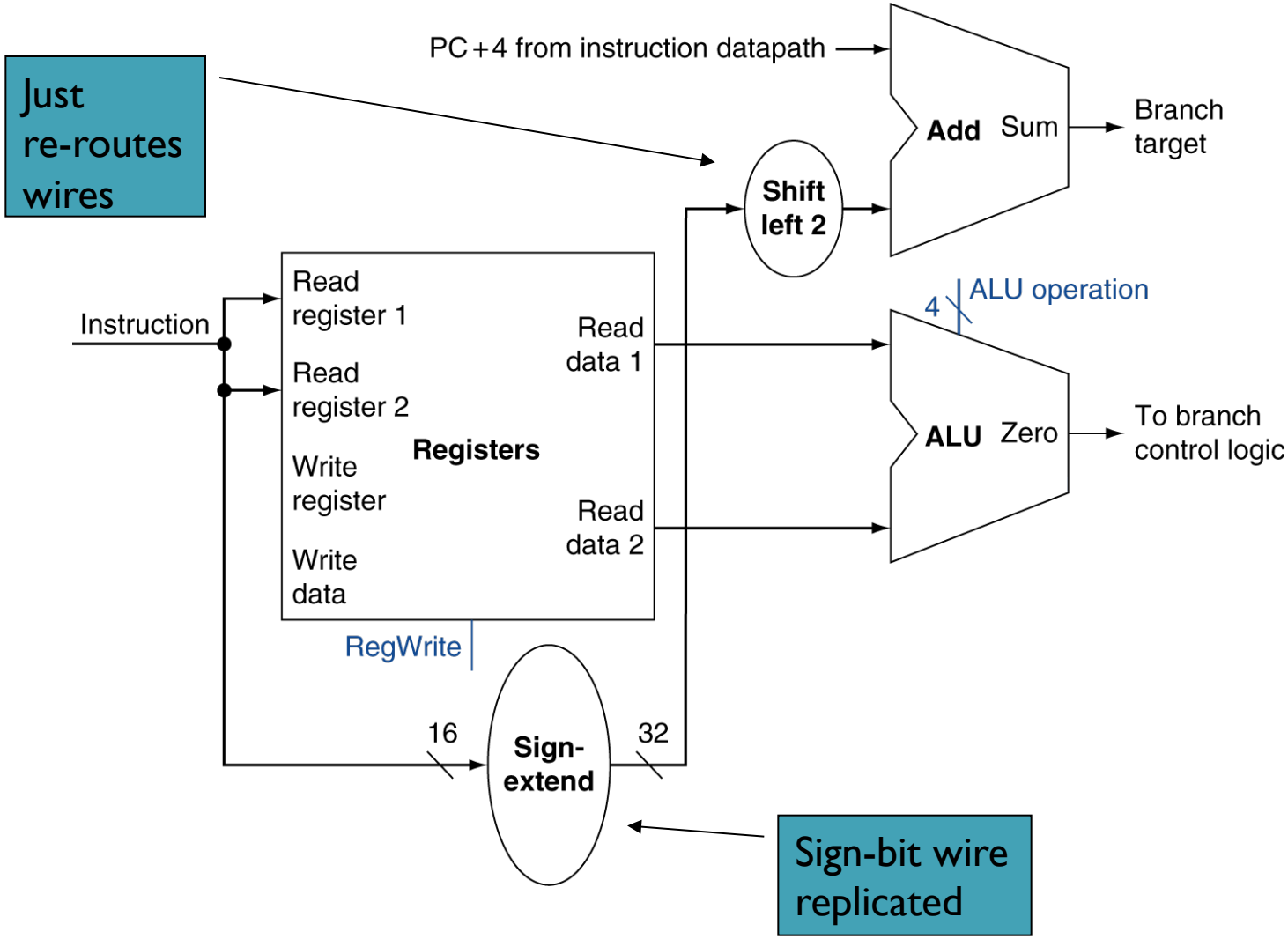


# Datapath for LW & SW (Fix this)

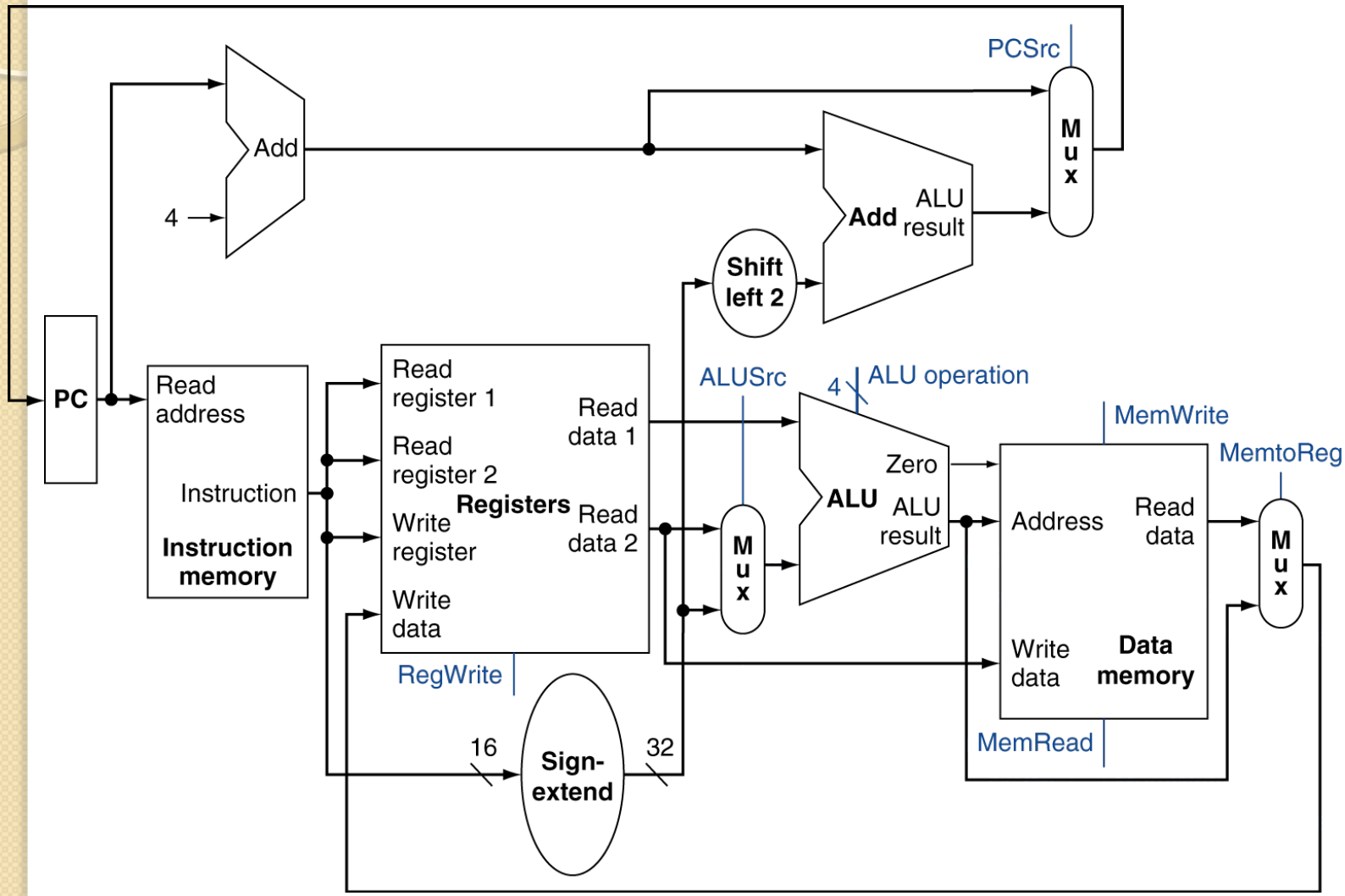
- R[rt] <- Mem[R[rs] + SignExt[imm16]] Example: lw rt, rs, imm16
- Mem[ R[rs] + SignExt[imm16] <- R[rt] ] Example: sw rt, rs, imm16



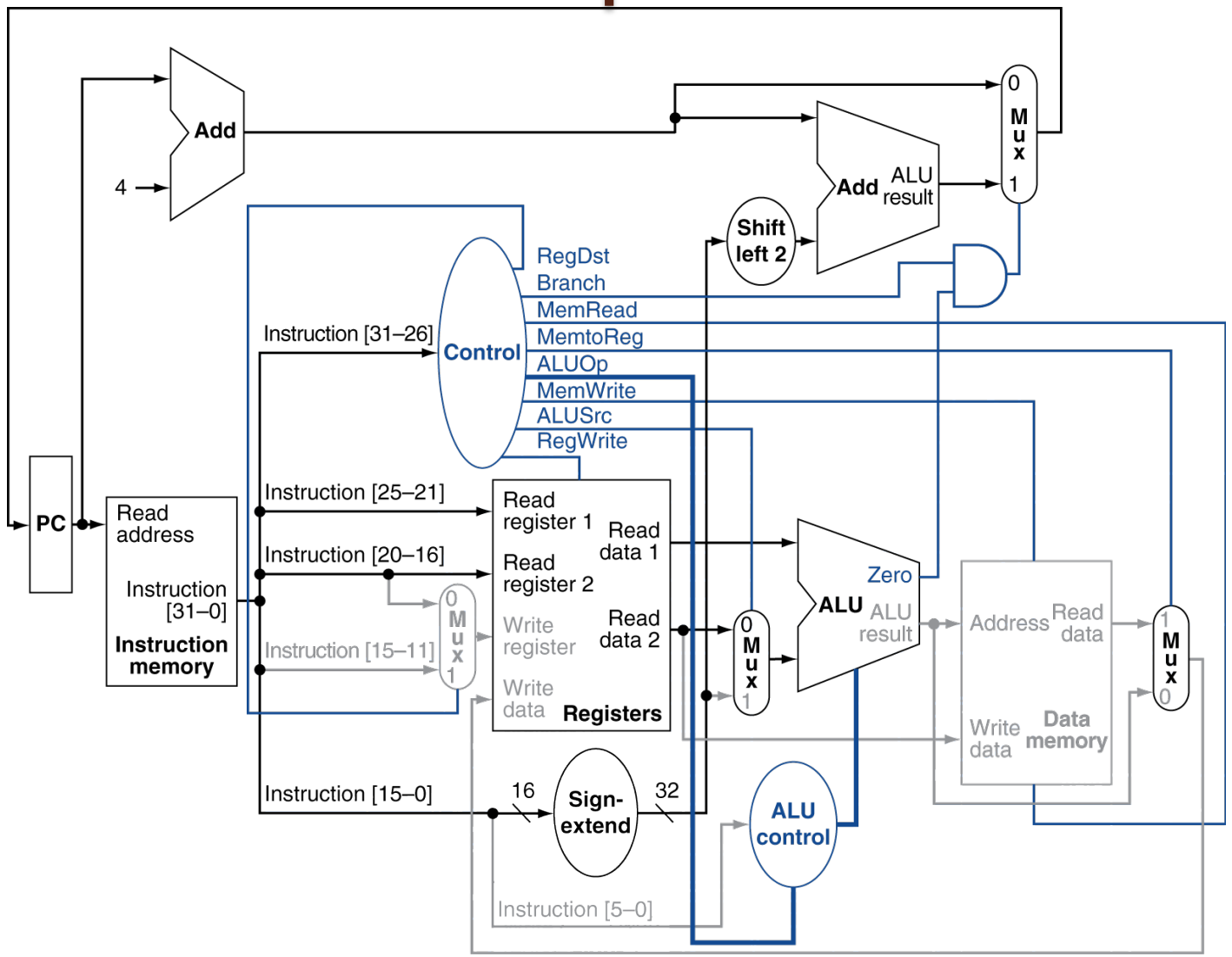
# Branch Instructions



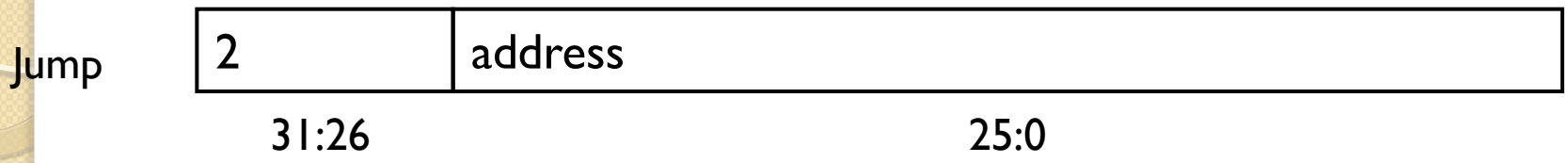
# Full Datapath



# Branch-on-Equal Instruction

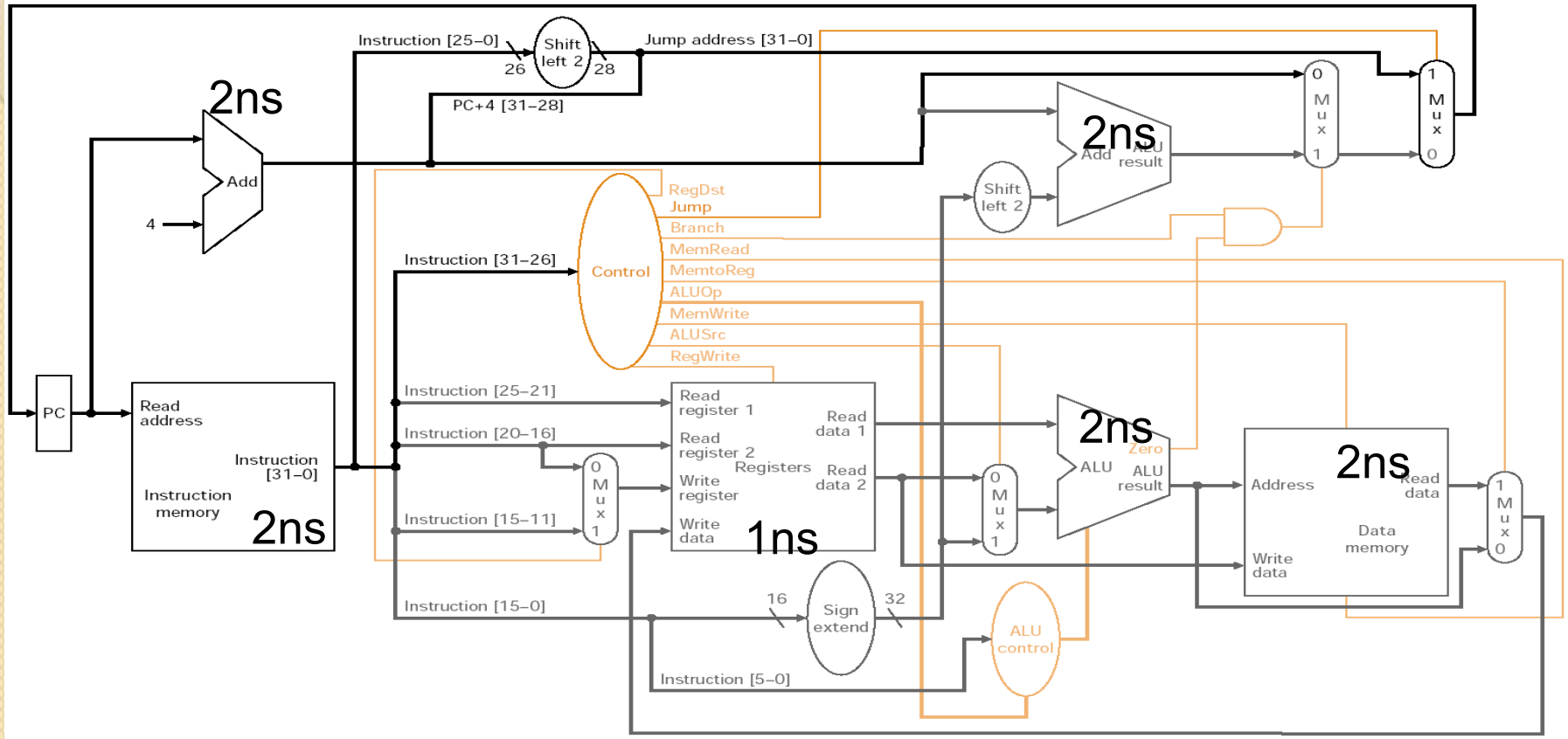


# Implementing Jumps




- Jump uses word address
- Update PC with concatenation of
  - Top 4 bits of old PC
  - 26-bit jump address
  - 00
- Need an extra control signal decoded from opcode

# Delays in Single Cycle Datapath



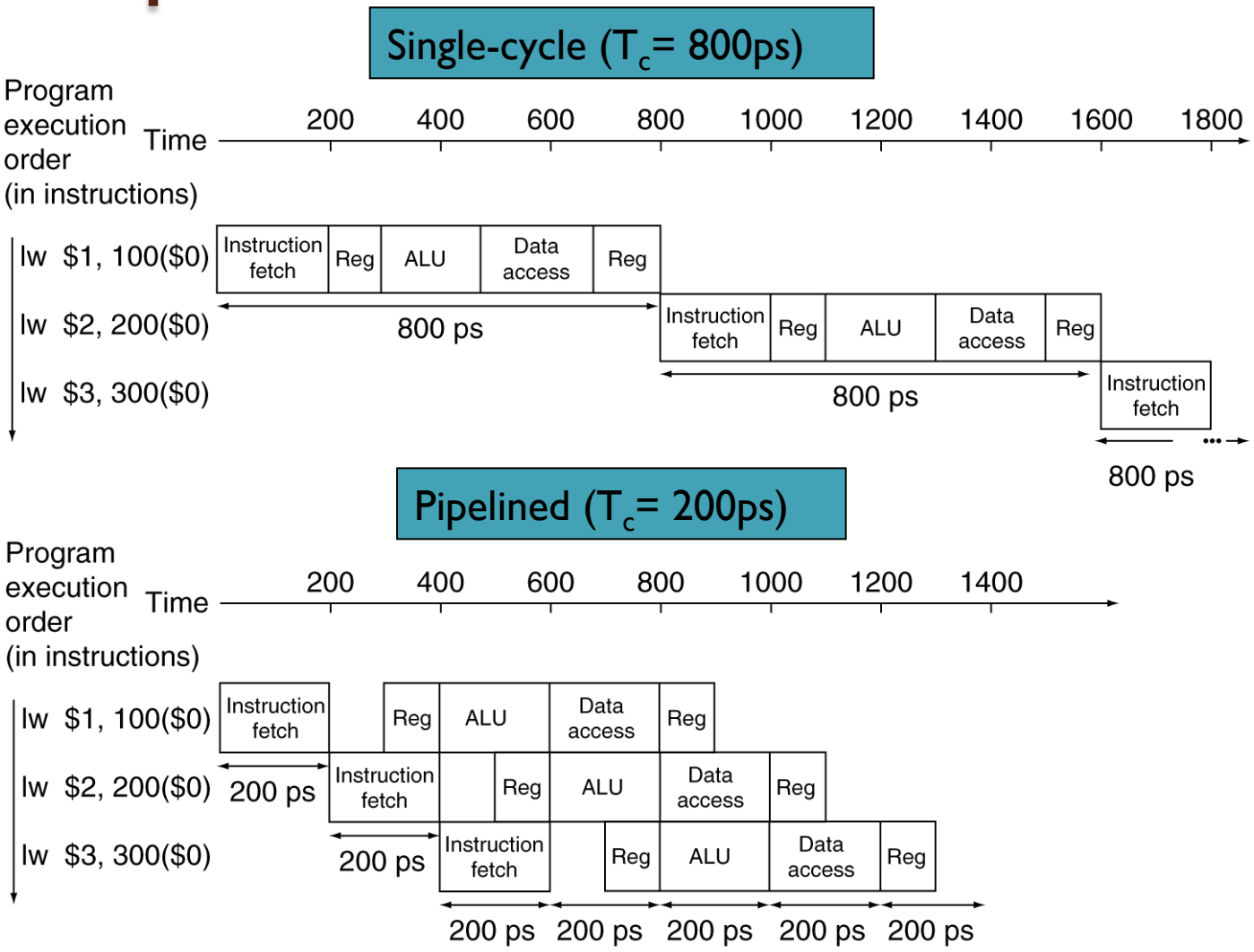

**What are the delays for lw, sw, R-Type, beq, j instructions?**

# Single Cycle Implementation

- Calculate cycle time assuming negligible delays except:
  - memory (2ns), ALU and adders (2ns), register file access (1ns)

| Instruction class | Instruction Fetch | Register Access | ALU | Register/ Memory Access | Register Access | |
|---|---|---|---|---|---|---|
| R-Type | X | X | X | R | | 6 |
| Load | X | X | X | M | X | 8 |
| Store | X | X | X | M | | 7 |
| Branch | X | X | X | | | 5 |
| Jump | X | | | | | 2 |

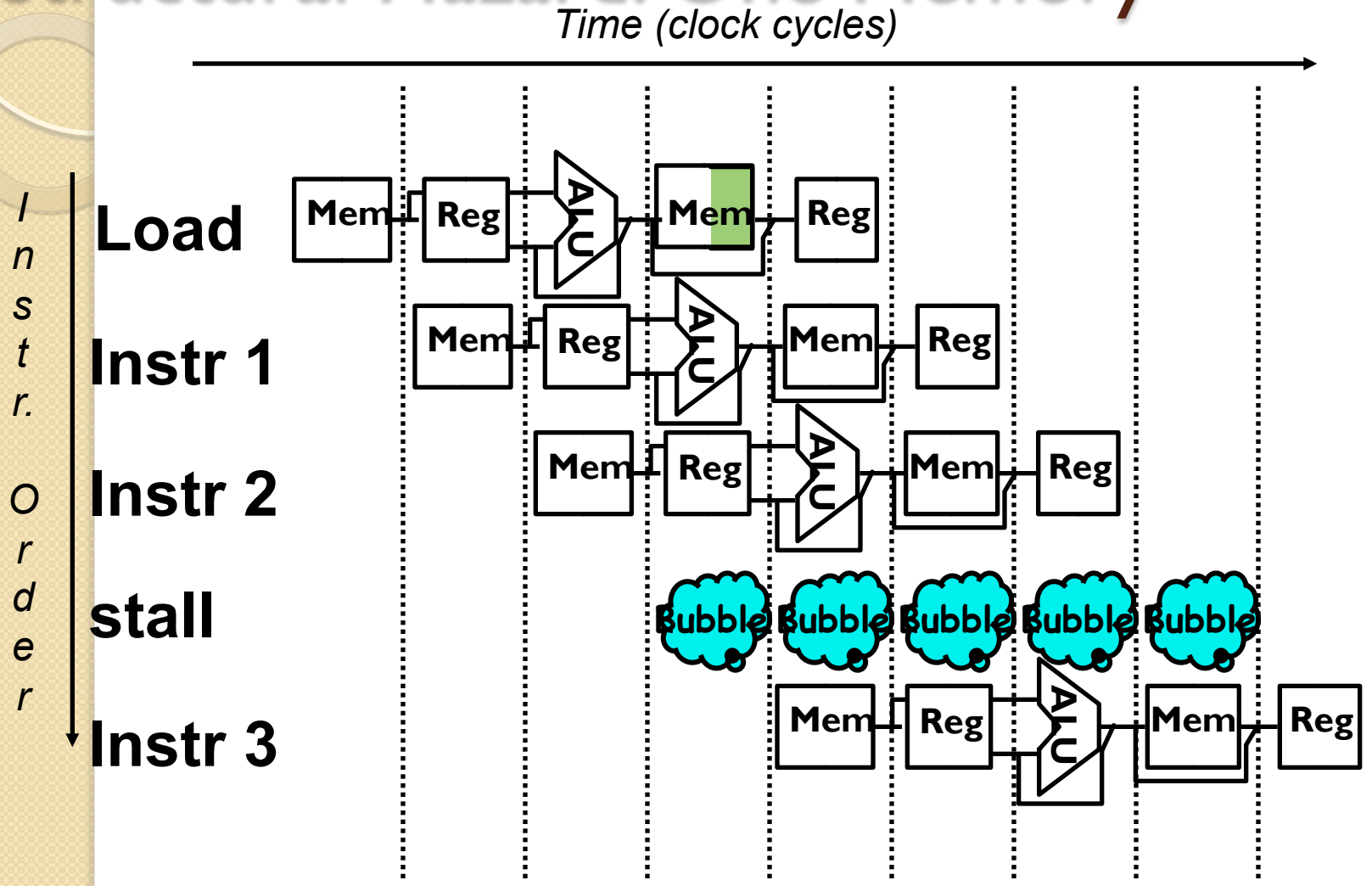
# Pipeline Performance



# Hazards

- Situations that prevent starting the next instruction in the next cycle
- _____hazards
  - A required resource is busy
- _____hazard
  - Need to wait for previous instruction to complete its data read/write
- _____hazard
  - Deciding on control action depends on previous instruction

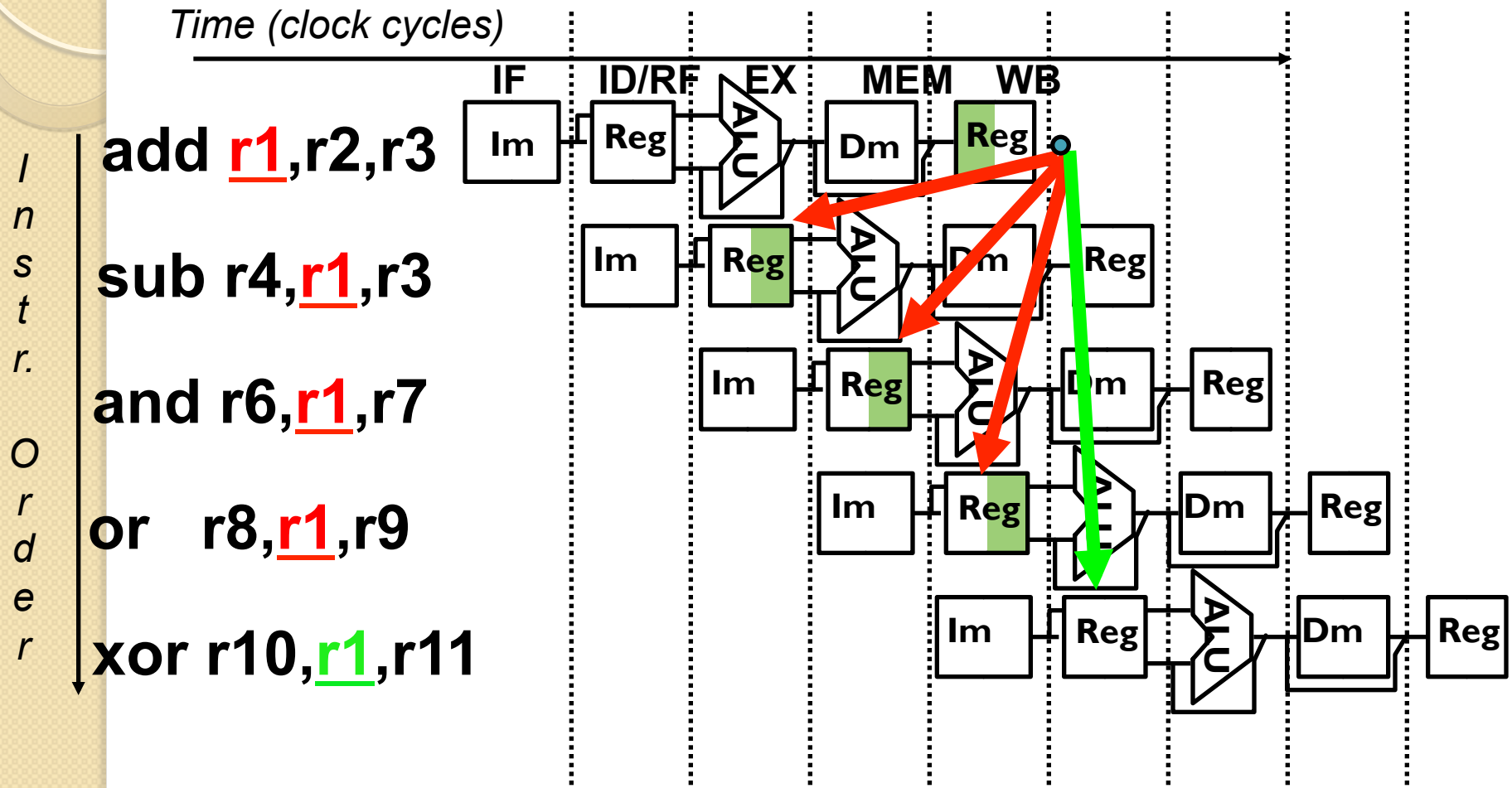
# Structural Hazard: One Memory




- Hazards can always be resolved by waiting

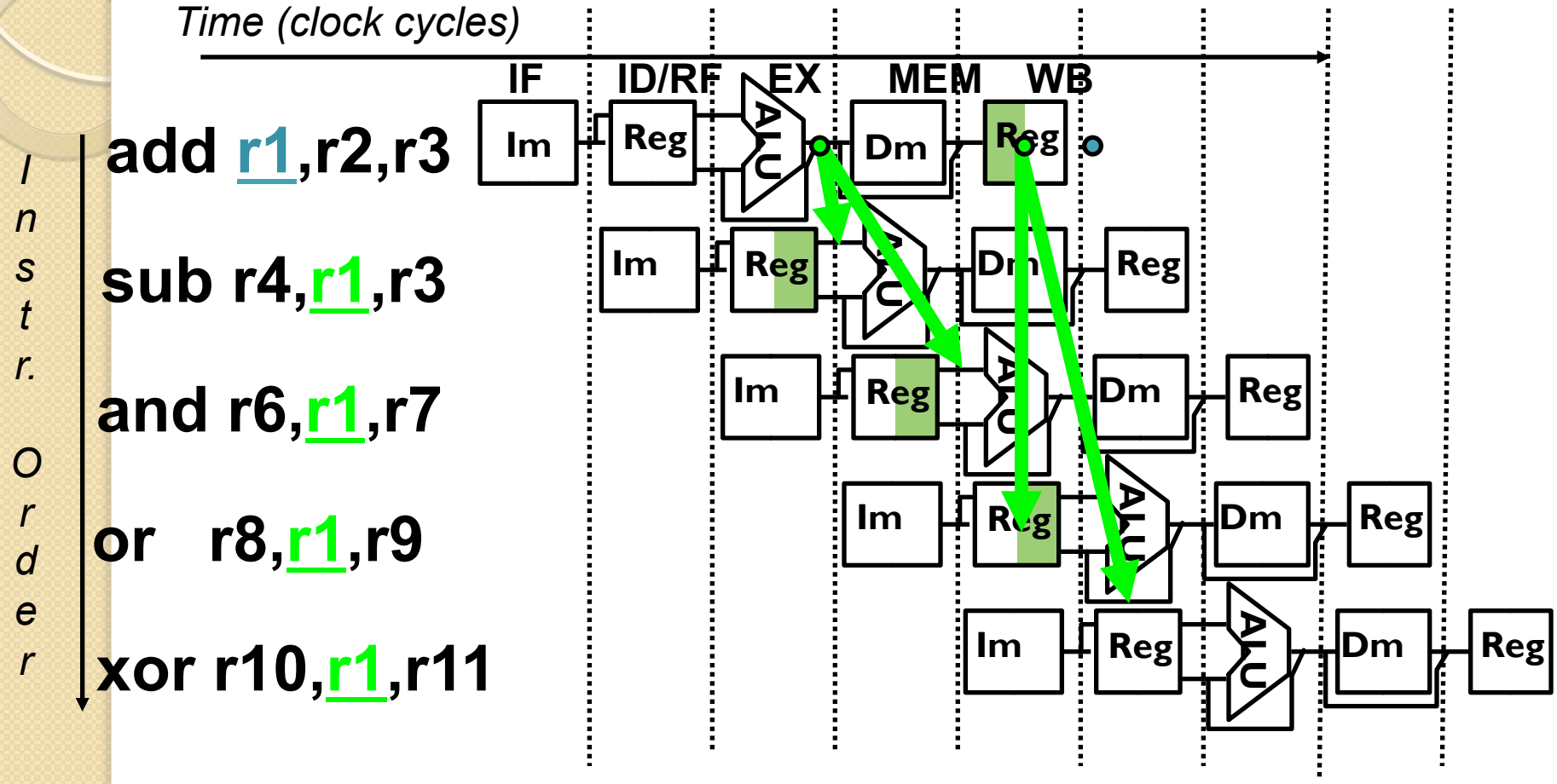
# Data Hazard Example

- Dependences backward in time are hazards




- Compilers can help, but it gets messy and difficult

# Data Hazard Solution




- Solution :"forward" result from one stage to another

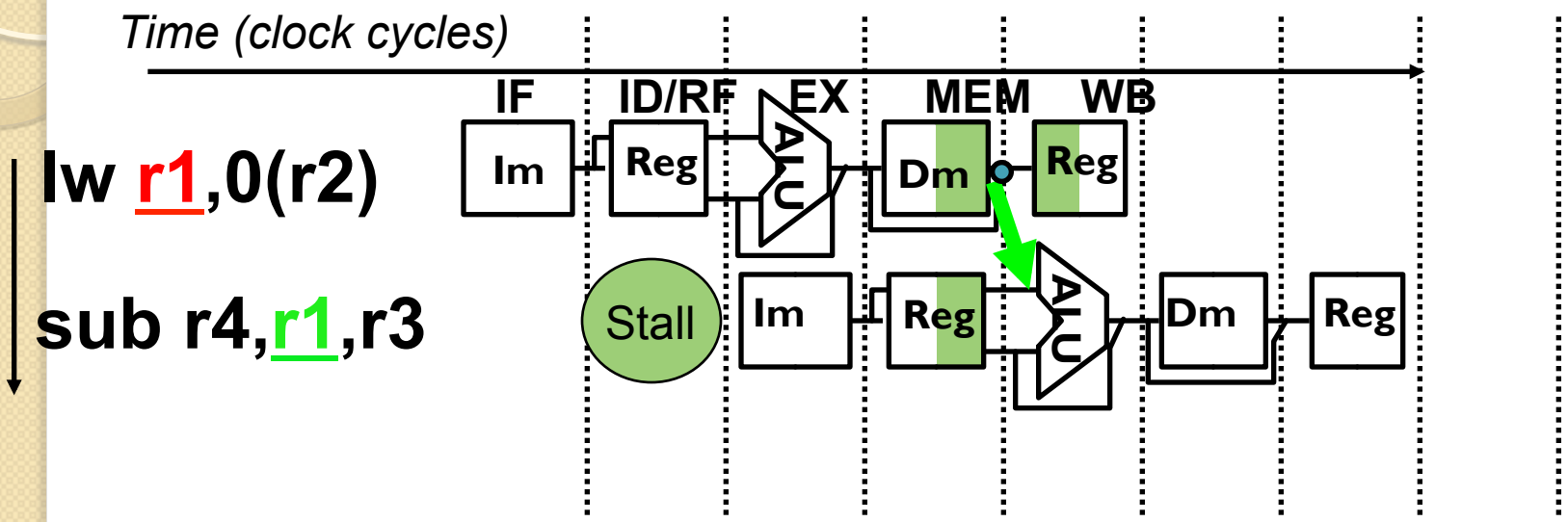
# Data Hazard Even with Forwarding

- Can you resolve this ?

**lw r1,0(r2)**

**sub r4,r1,r3**

# Data Hazard Even with Forwarding



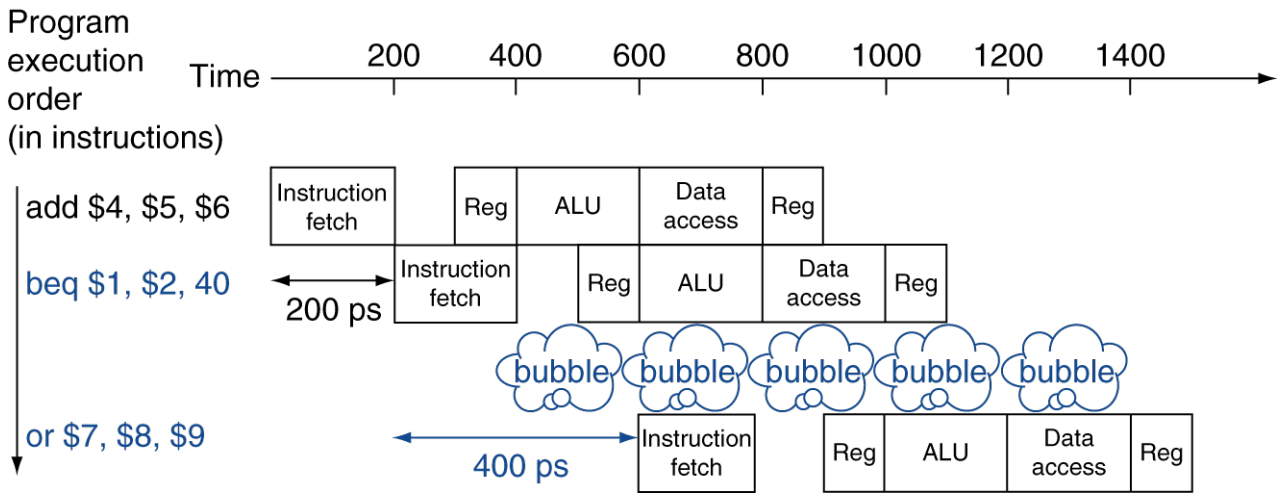

- Must delay/stall instruction dependent on loads
- Sometimes the instruction sequence can be reordered to avoid pipeline stalls

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

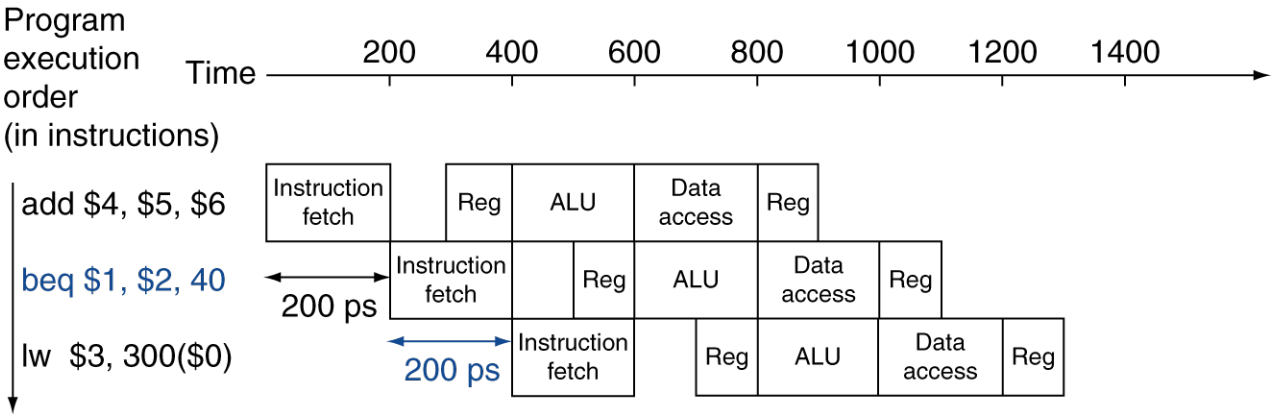
# Stall on Branch

- Wait until branch outcome determined before fetching next instruction

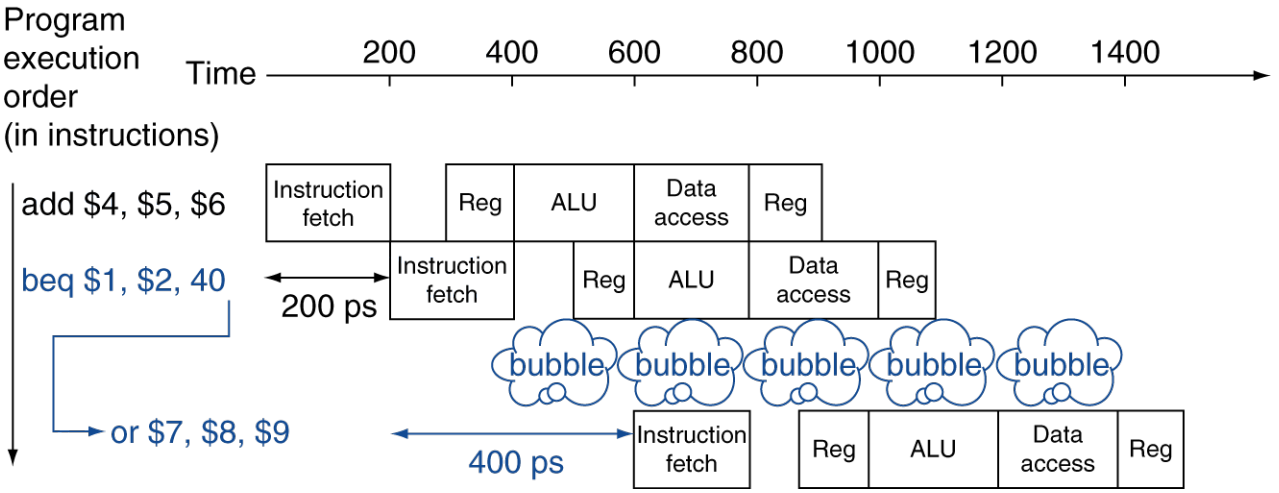


# MIPS with Predict Not Taken

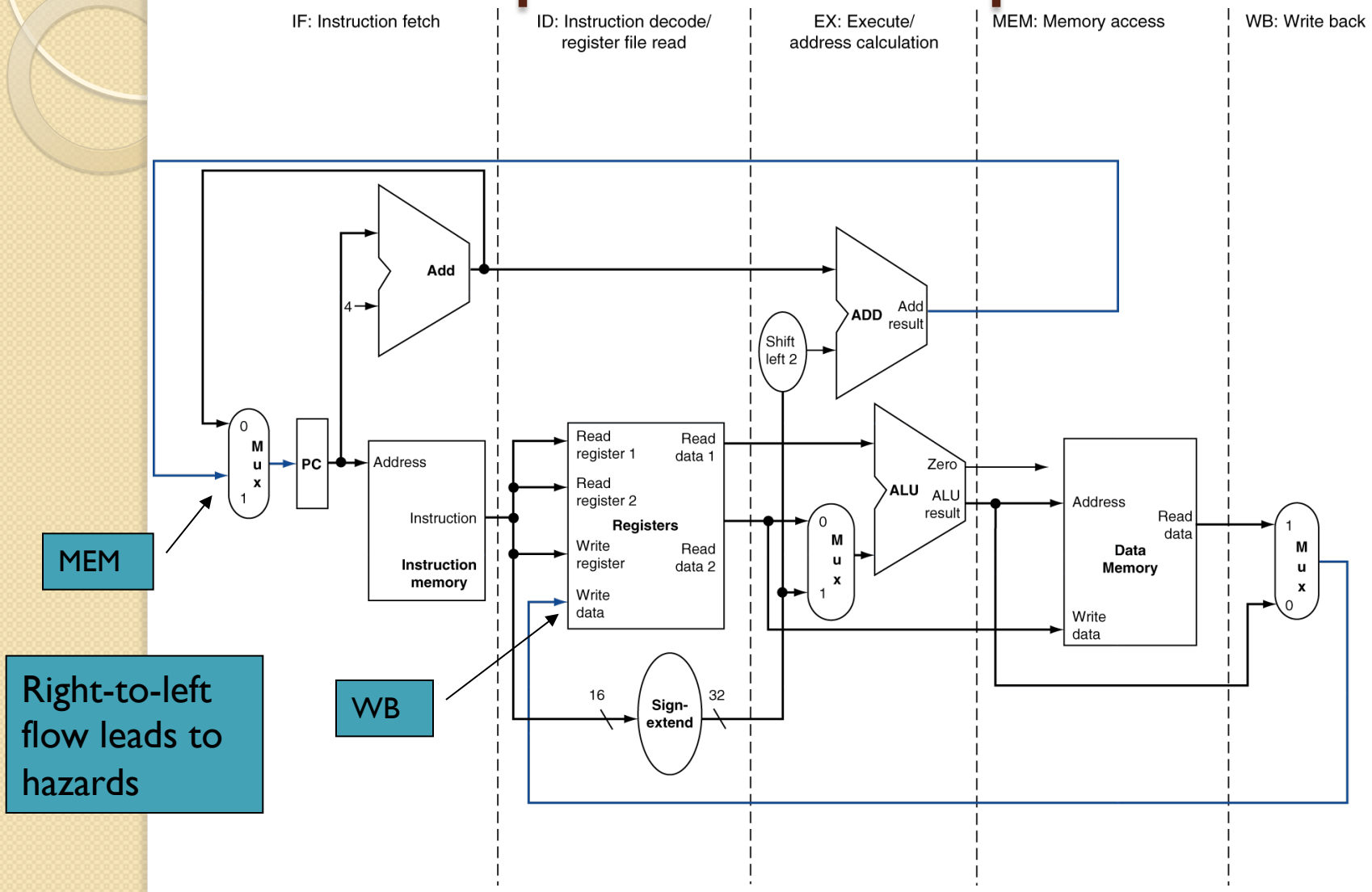
Prediction correct




Prediction incorrect

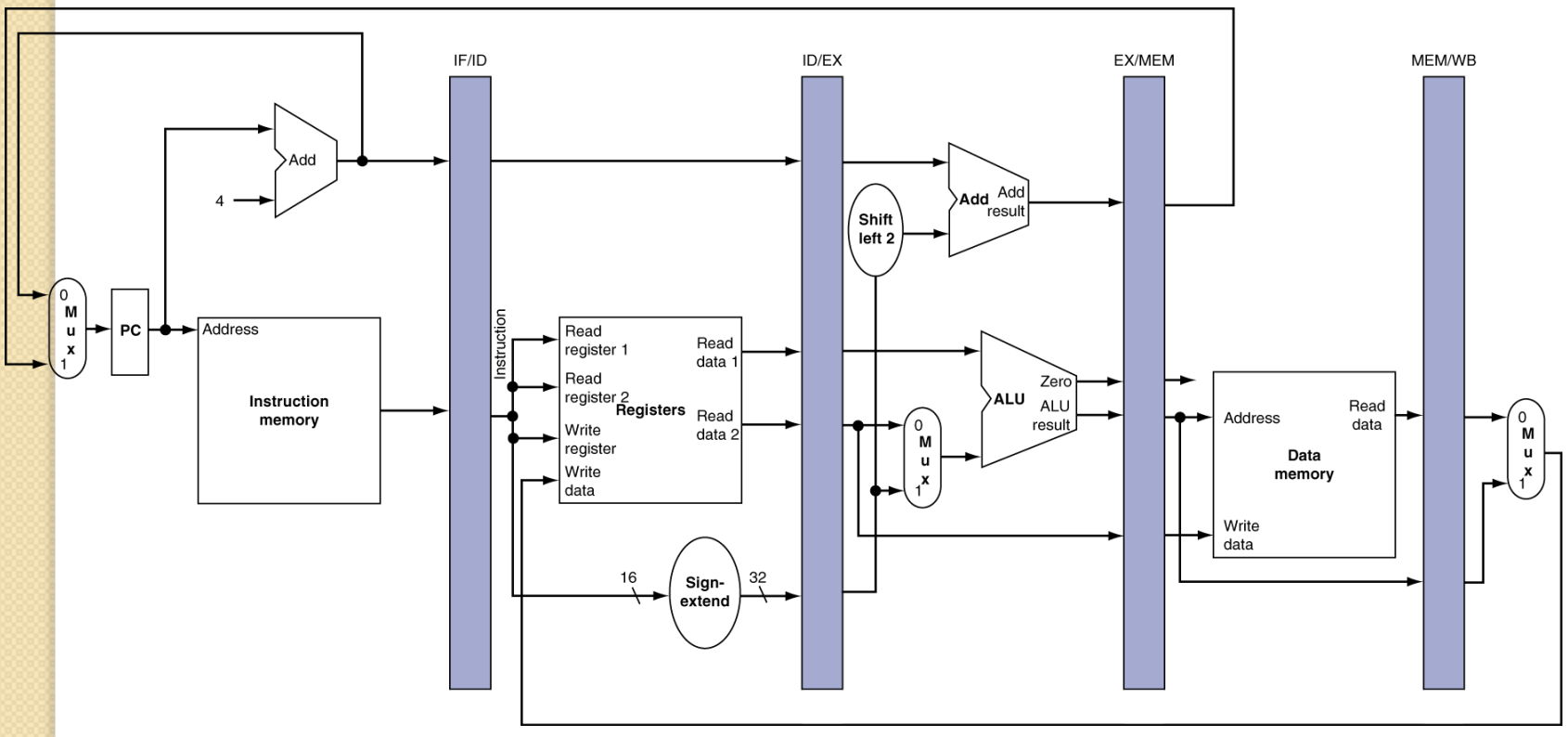


# MIPS Pipelined Datapath



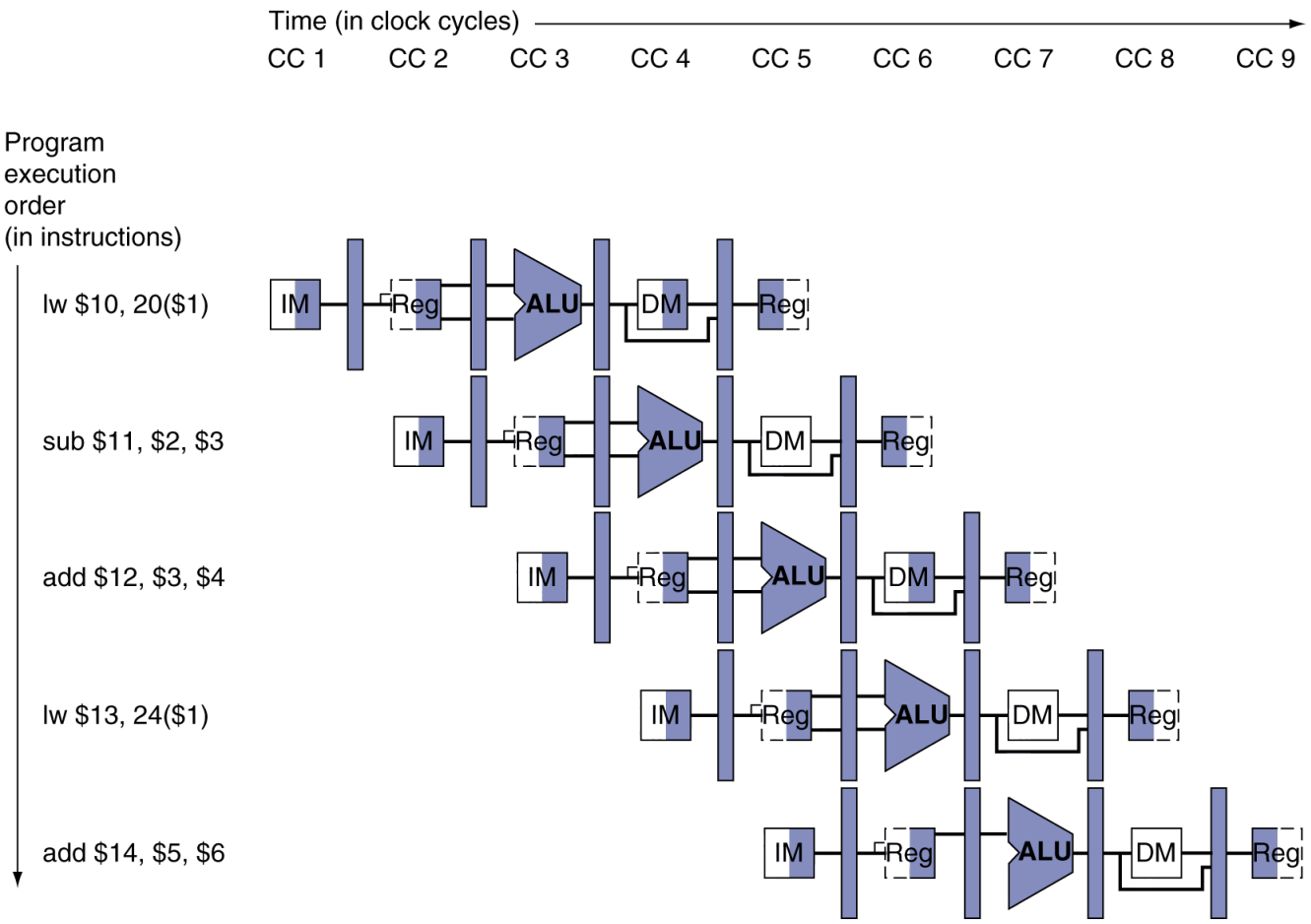
# Pipeline registers

- Need registers between stages (Why ?)



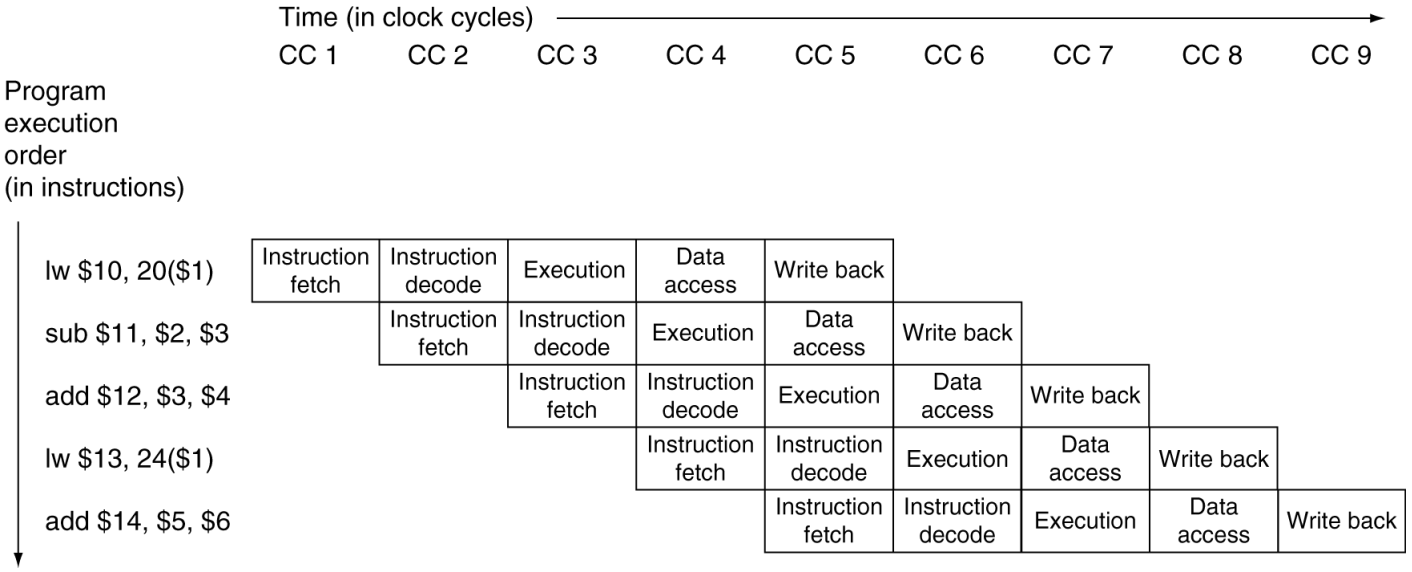
# Multi-Cycle Pipeline Diagram

- Form showing resource usage



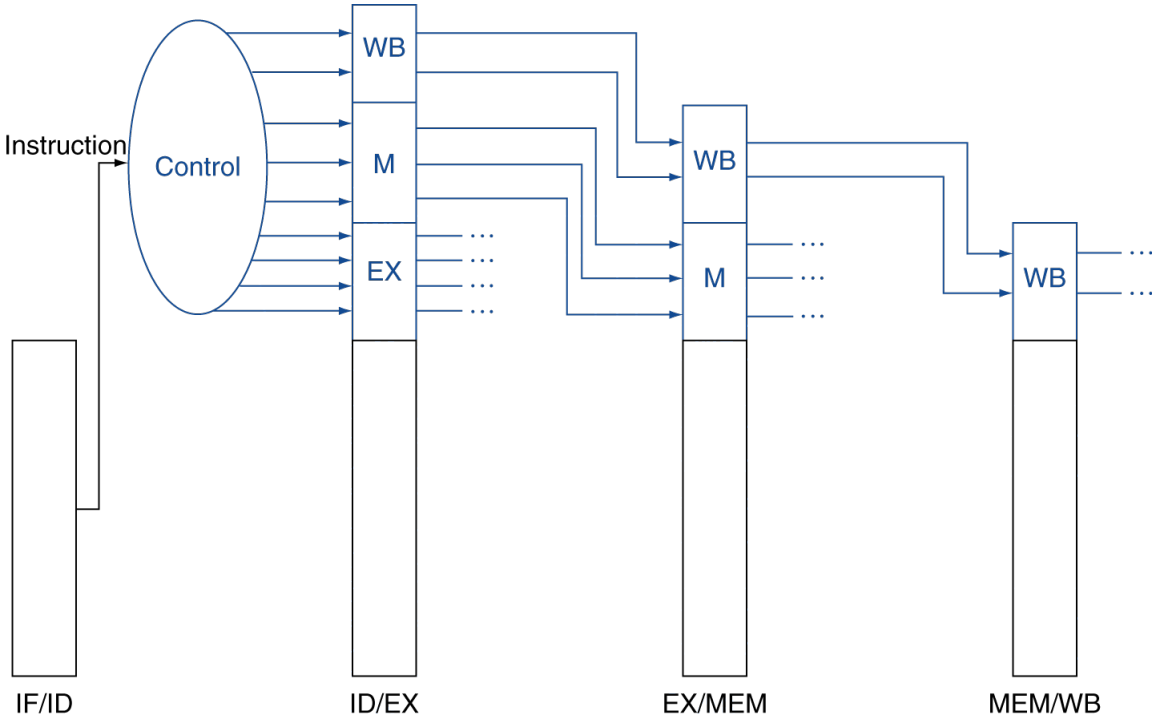
# Multi-Cycle Pipeline Diagram

- Traditional form



# Pipelined Control

- Control signals derived from instruction
  - As in single-cycle implementation



# Instruction-Level Parallelism (ILP)

- Pipelining: executing multiple instructions in parallel
- To increase ILP
  - Deeper pipeline
    - Less work per stage ⇒ shorter clock cycle
  - Multiple issue
    - Replicate pipeline stages ⇒ multiple pipelines
    - Start multiple instructions per clock cycle
    - CPI < 1, so use Instructions Per Cycle (IPC)
    - E.g., 4GHz 4-way multiple-issue
      - 16 BIPS, peak CPI = 0.25, peak IPC = 4
    - But dependencies reduce this in practice

# MIPS with Static Dual Issue

- Two-issue packets
  - One ALU/branch instruction
  - One load/store instruction
  - 64-bit aligned
    - ALU/branch, then load/store
    - Pad an unused instruction with nop

| Address | Instruction type | Pipeline Stages | | | | | | |
|---|---|---|---|---|---|---|---|---|
| n | ALU/branch | IF | ID | EX | MEM | WB | | |
| n + 4 | Load/store | IF | ID | EX | MEM | WB | | |
| n + 8 | ALU/branch | | IF | ID | EX | MEM | WB | |
| n + 12 | Load/store | | IF | ID | EX | MEM | WB | |
| n + 16 | ALU/branch | | | IF | ID | EX | MEM | WB |
| n + 20 | Load/store | | | IF | ID | EX | MEM | WB |

# Loop Unrolling Example

| | ALU/branch | Load/store | cycle |
|---|---|---|---|
| Loop: | addi $s1, $s1,-16 | lw $t0, 0($s1) | 1 |
| | nop | lw $t1, 12($s1) | 2 |
| | addu $t0, $t0, $s2 | lw $t2, 8($s1) | 3 |
| | addu $t1, $t1, $s2 | lw $t3, 4($s1) | 4 |
| | addu $t2, $t2, $s2 | sw $t0, 16($s1) | 5 |
| | addu $t3, $t4, $s2 | sw $t1, 12($s1) | 6 |
| | nop | sw $t2, 8($s1) | 7 |
| | bne $s1, $zero, Loop | sw $t3, 4($s1) | 8 |

- IPC = 14/8 = 1.75
  - Closer to 2, but at cost of registers and code size

# Dynamic Multiple Issue

- "Superscalar" processors
- CPU decides whether to issue 0, 1, 2, … each cycle
  - Avoiding structural and data hazards
- Avoids the need for compiler scheduling
  - Though it may still help
  - Code semantics ensured by the CPU

# Does Multiple Issue Work?

**The BIG Picture**

- Yes, but not as much as we'd like
- Programs have real dependencies that limit ILP
- Some dependencies are hard to eliminate
  - e.g., pointer aliasing
- Some parallelism is hard to expose
  - Limited window size during instruction issue
- Memory delays and limited bandwidth
  - Hard to keep pipelines full
- Speculation can help if done well

# Concluding Remarks for Processors

- ISA influences design of datapath and control
- Datapath and control influence design of ISA
- Pipelining improves instruction throughput using parallelism
  - More instructions completed per second
  - Latency for each instruction not reduced
- Hazards: structural, data, control
- Multiple issue and dynamic scheduling (ILP)
  - Dependencies limit achievable parallelism
  - Complexity leads to the power wall

# QUESTIONS

**What is the difference between a conflict miss and a compulsory miss? How would you reduce each type?**

**What are two different strategies for dealing with cache writes? What is an advantage and disadvantage of each type?**

- **Why might we want more than one level of a cache?**

- **Suppose a processor has a CPI of 3.0 given a perfect cache. If there are 1.4 memory accesses per instruction, a miss penalty of 20 cycles, and a miss rate of 5%, what is the effective CPI with the real cache?**

- **What are two advantages of using virtual memory ?**

- **What is a TLB? Why do we need it?**
- **Design a cache controller.**

- **How does pipelining improve performance?**

- **What is multiple issue? Is this the same as VLIW?**

**Why is branch prediction so important? Does multiple issue increase or decrease the need for such prediction?**

- Explain the stored-program concept.

- For a data cache with a 92% hit rate and a 2-cycle hit latency, calculate the average memory access latency (AMAT). Assume that the latency to memory and a cache miss penalty together is 124 clock cycles.

  - AMAT = 2 + 0.08*124
    = 11.92 cycles

# Yes, CPI ?

(5 pts) A compiler designer is trying to decide between two code segments for a particular machine. The hardware designers have provided the following data below about the CPI for each class, and the instruction counts being considered for each code sequence.

| Class | CPI for this instruction class |
|---|---|
| A | 2 |
| B | 3 |

| | Instruction Counts for Instruction Classes | |
|---|---|---|
| Code sequence | A | B |
| 1 | 3 | 5 |
| 2 | 7 | 2 |

How many cycles are required for each code sequence?

Code sequence #1:

Code sequence #2:

Which is faster and how by how much?

What is the CPI for each code sequence?

CPI for code sequence #1:

CPI for code sequence #2: