RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

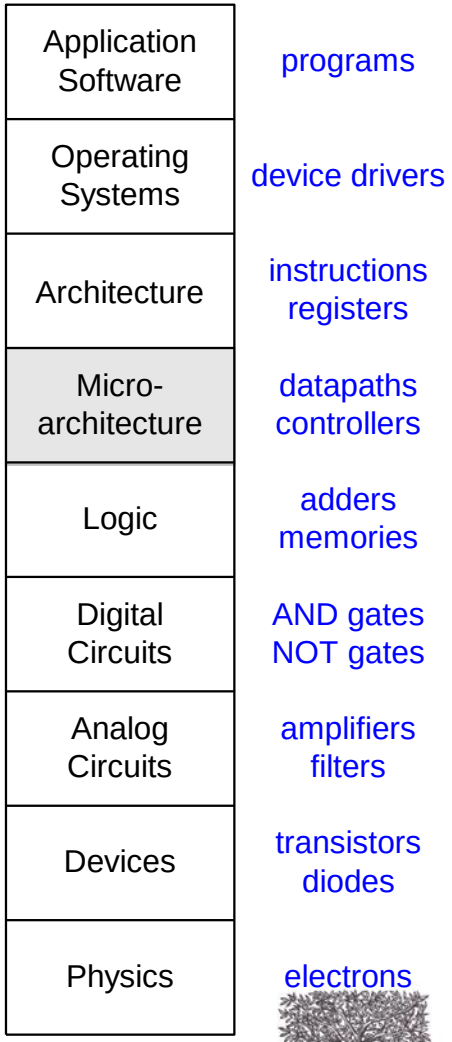
# ECE-332:437
# DIGITAL SYSTEMS DESIGN (DSD)

## Fall 2016 – Lecture 9
## Micro Architecture (MIPS) - Recap

Nagi Naganathan
November 10, 2016

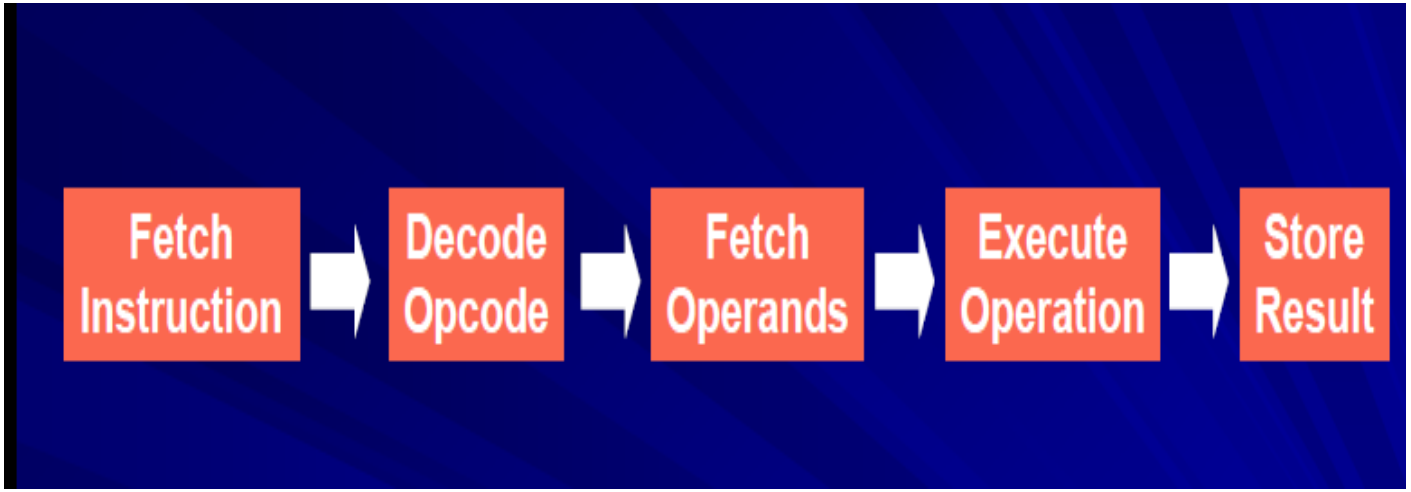
# Introduction

- **Microarchitecture:** how to implement an architecture in hardware
- Processor:
  - **Datapath:** functional blocks
  - **Control:** control signals



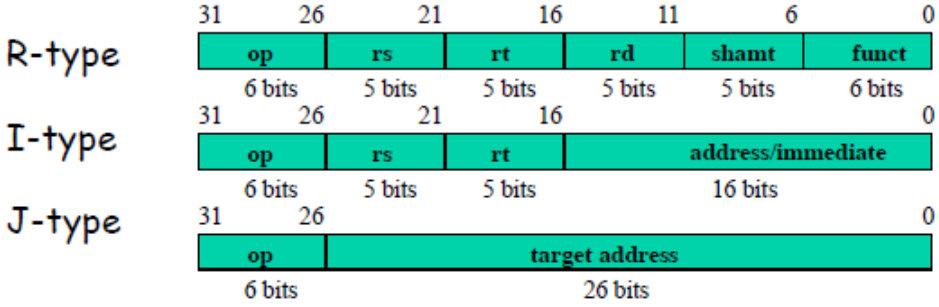
# Microarchitecture

- Multiple implementations for a single architecture:
  - **Single-cycle:** Each instruction executes in a single cycle
  - **Multi-cycle:** Each instruction is broken into series of shorter steps
  - **Pipelined:** Each instruction broken up into series of steps & multiple instructions execute at once


Fetch Instruction
Decode Opcode
Fetch Operands
Execute Operation
Store Result

# Microarchitecture

## Review: The MIPS Instruction




The different fields are:

op: operation ("opcode") of the instruction

rs, rt, rd: the source and destination register specifiers

shamt: shift amount

funct: selects the variant of the operation in the "op" field

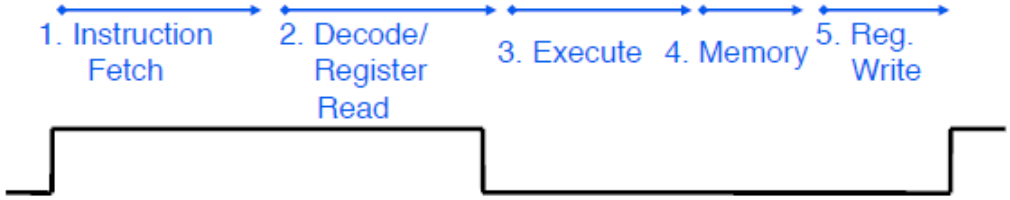
address / immediate: address offset or immediate value

target address: target address of jump instruction

# Microarchitecture

## CPU clocking (1/2)

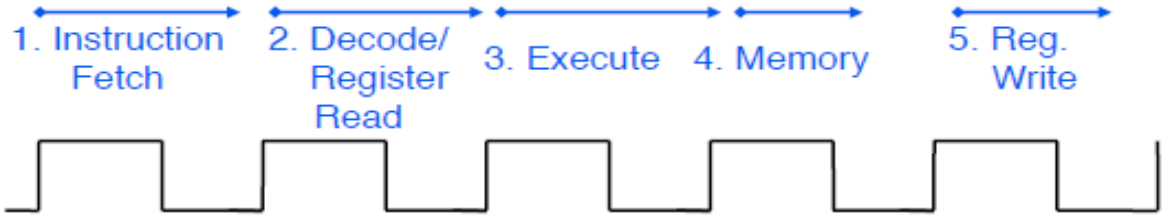
- Single Cycle CPU: All stages of an instruction are completed within one ***long*** clock cycle.
  - The clock cycle is made sufficient long to allow each instruction to complete all stages without interruption and within one cycle.



# Microarchitecture

## CPU clocking (2/2)

- Multiple-cycle CPU: Only one stage of instruction per clock cycle.
  - The clock is made as long as the slowest stage.




Several significant advantages over single cycle execution: Unused stages in a particular instruction can be skipped OR instructions can be pipelined (overlapped).

# The Processor: Datapath & Control

- Simplified MIPS implementation to contain only:
  - memory-reference instructions: `lw, sw`
  - arithmetic-logical instructions: `add, sub, and, or, slt`
  - control flow instructions: `beq, j`
- Generic Implementation:
  - use the program counter (PC) to supply instruction address
  - get the instruction from memory
  - read registers
  - use the instruction to decide exactly what to do

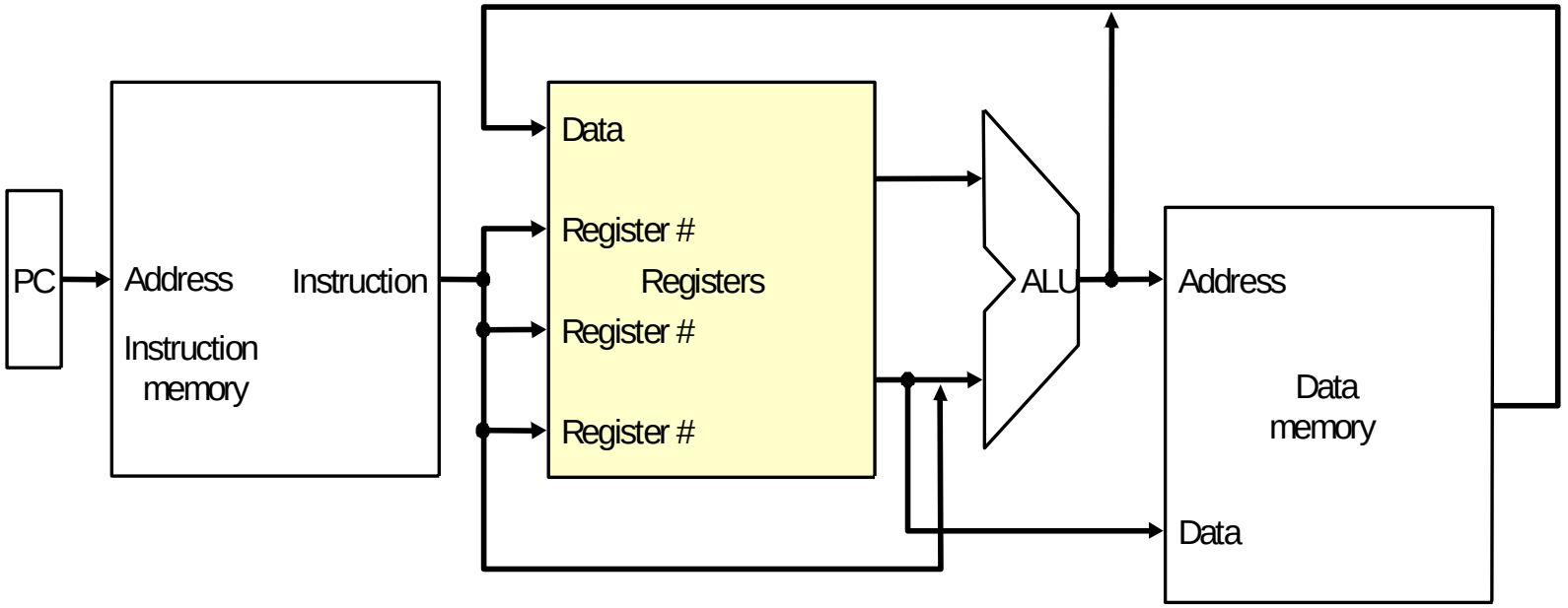
- All instructions use the ALU after reading the registers
  Why?
  - memory-reference?
  - arithmetic?
  - control flow?

# Architectural State

- Determines everything about a processor:
  - PC
  - 32 registers
  - Memory

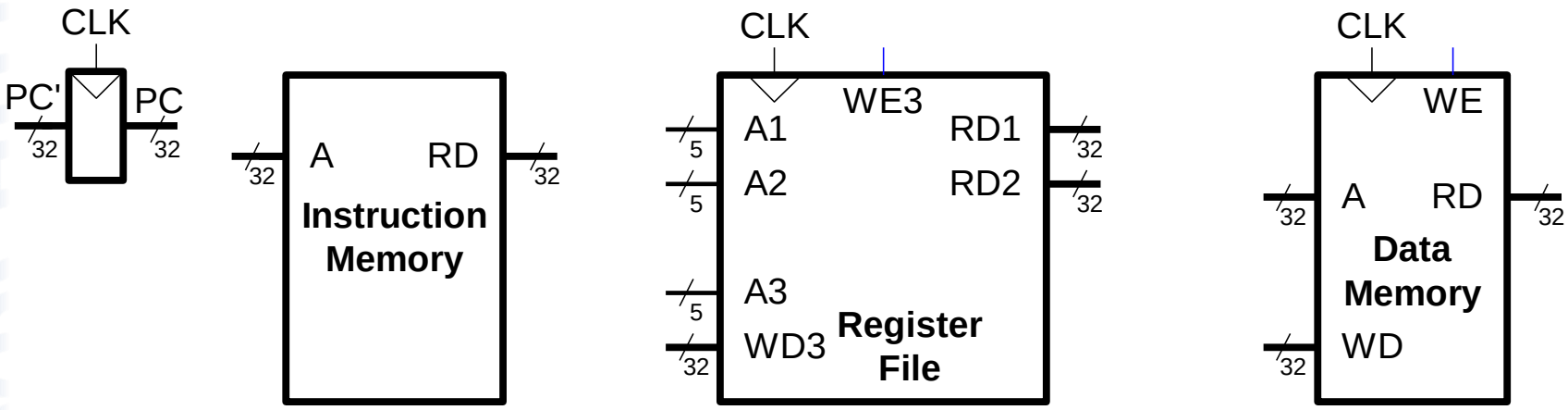
# More Implementation Details

- Abstract / Simplified View:

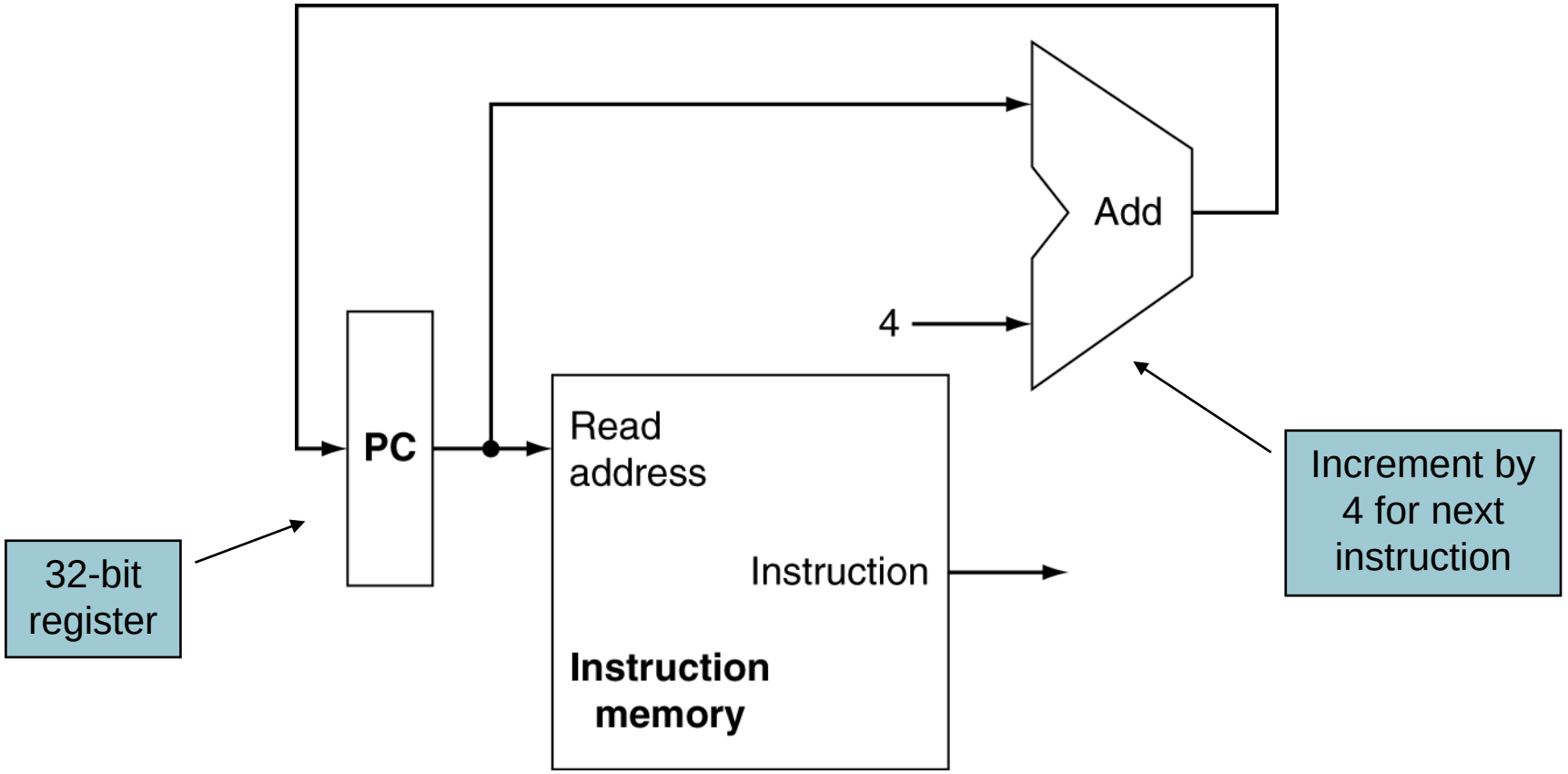



- Two types of functional units:
  - elements that operate on data values (combinational)
  - elements that contain state (sequential)

# MIPS State Elements

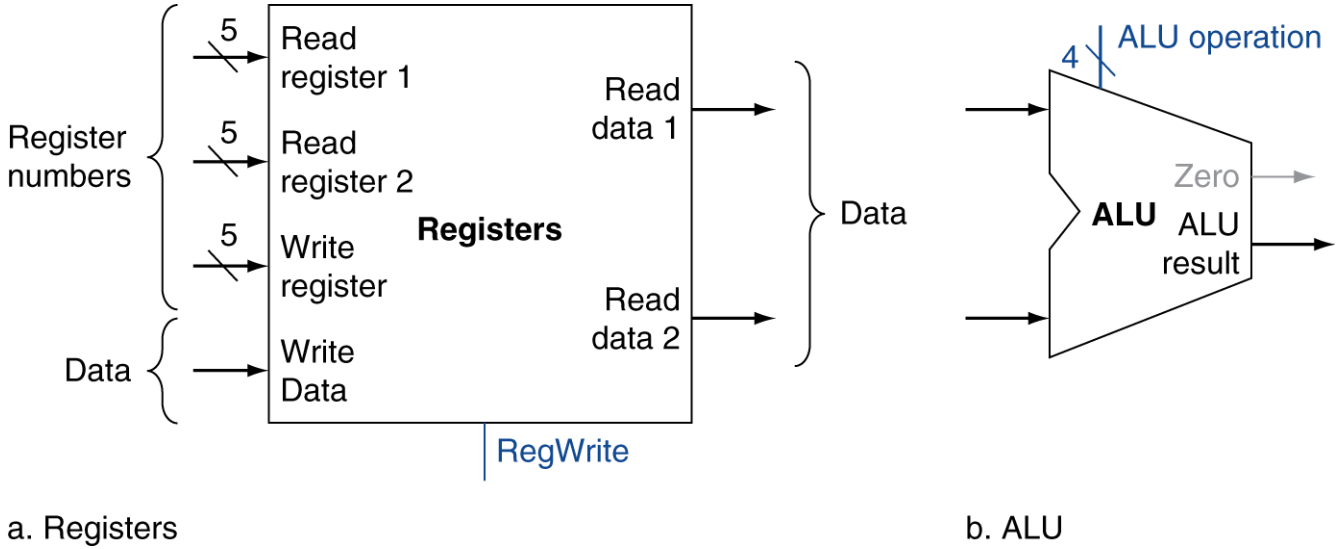


# Instruction Fetch



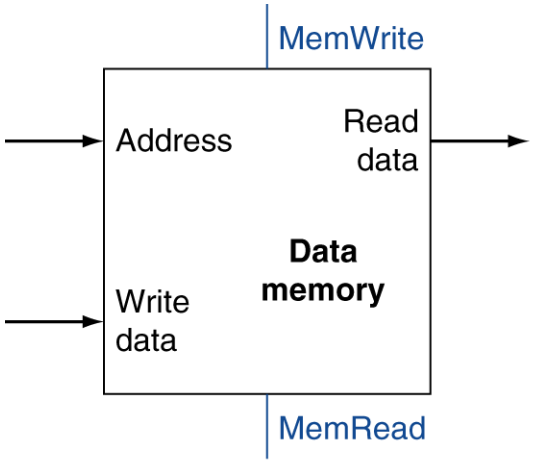
# R-Format Instructions

- Read two register operands
- Perform arithmetic/logical operation
- Write register result

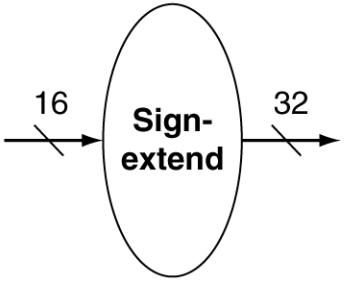


# Load/Store Instructions

- Read register operands
- Calculate address using 16-bit offset
  - Use ALU, but sign-extend offset
- Load: Read memory and update register
- Store: Write register value to memory




a. Data memory unit



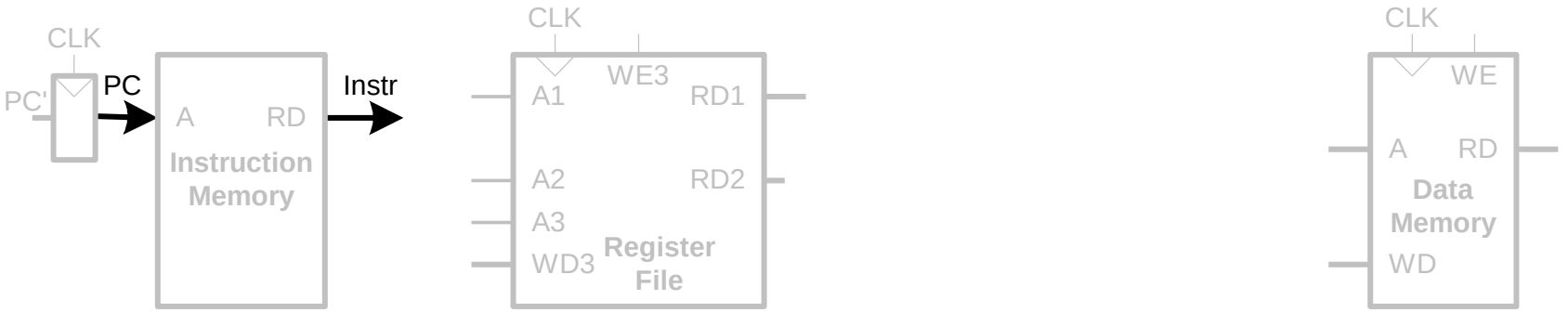

b. Sign extension unit

# Branch Instructions

- Read register operands
- Compare operands
  - Use ALU, subtract and check Zero output
- Calculate target address
  - Sign-extend displacement
  - Shift left 2 places (word displacement)
  - Add to PC + 4
    - Already calculated by instruction fetch

# Single-Cycle Datapath: `lw` fetch

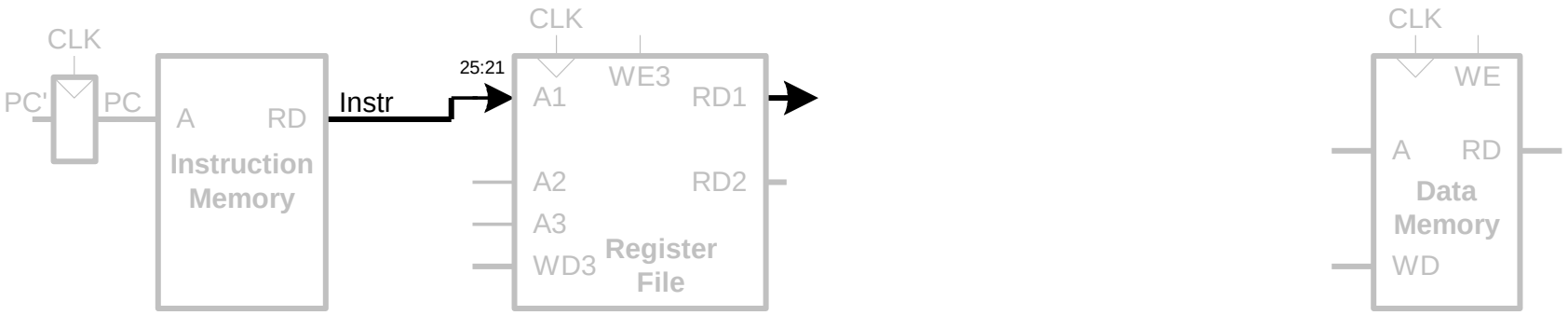
## STEP 1: Fetch instruction



# Single-Cycle Datapath: `lw` Register Read

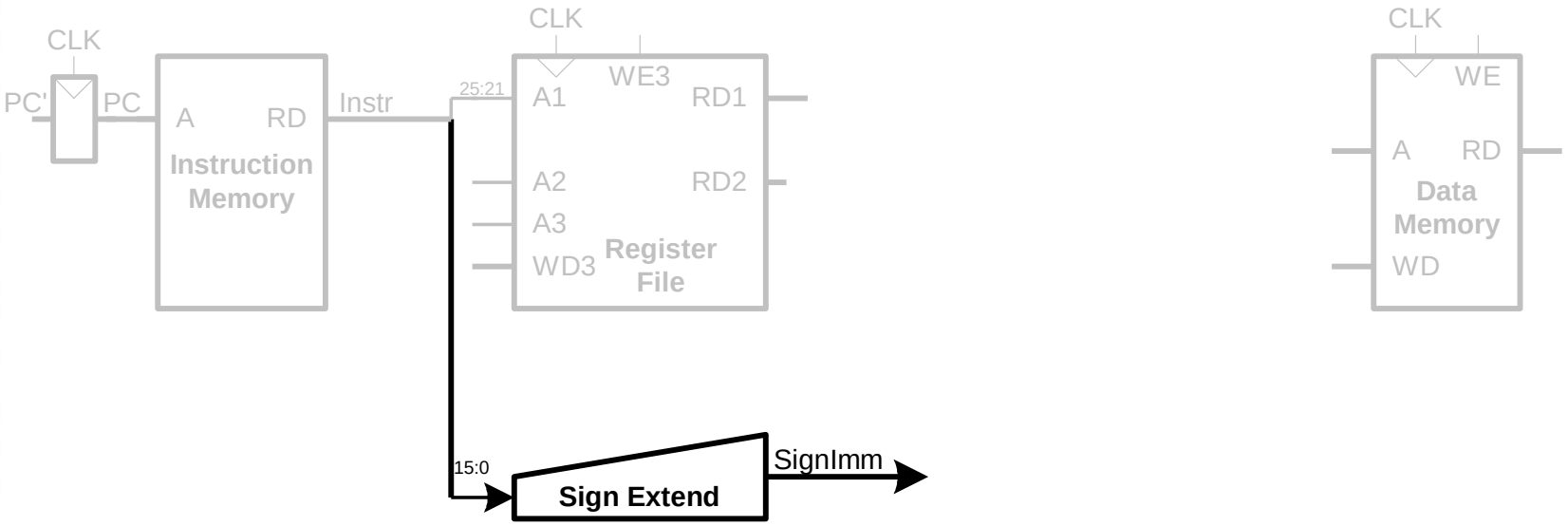
## STEP 2: Read source operands from RF

R[rt] ← DMEM[ R[rs] + sign_ext(Imm16)]



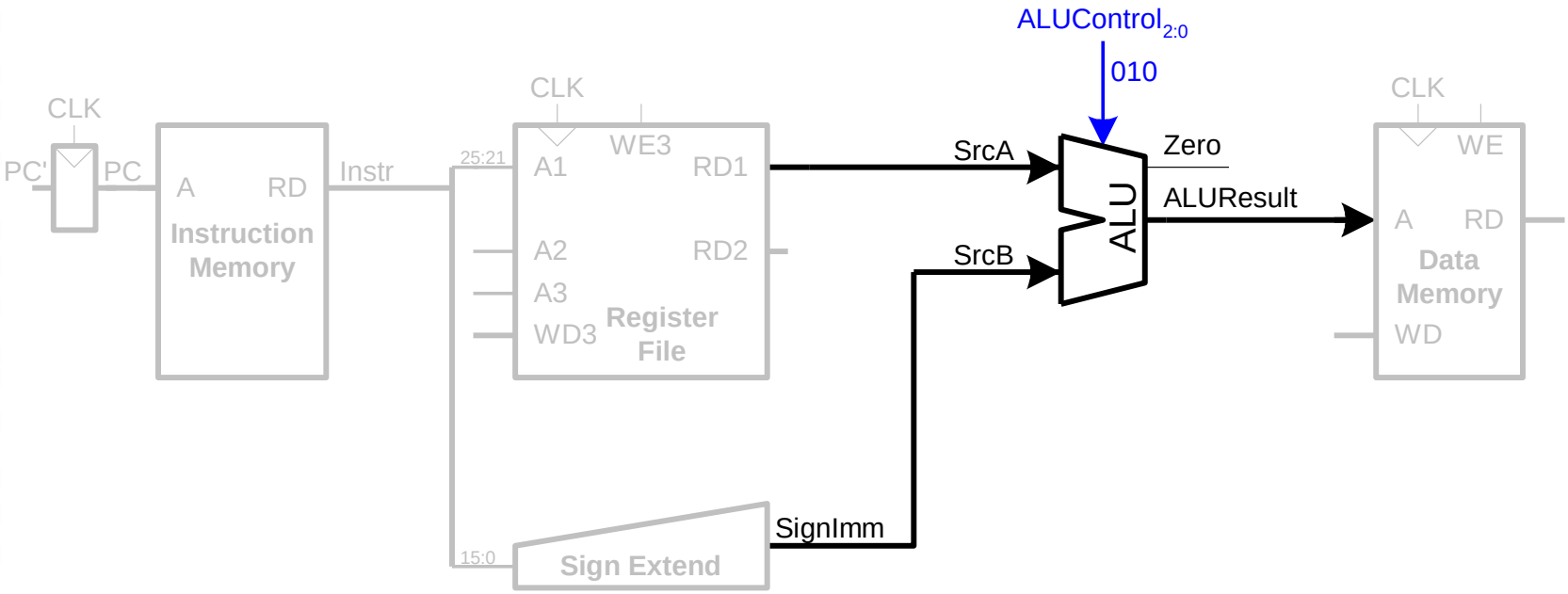
# Single-Cycle Datapath: `lw` Immediate

## STEP 3: Sign-extend the immediate



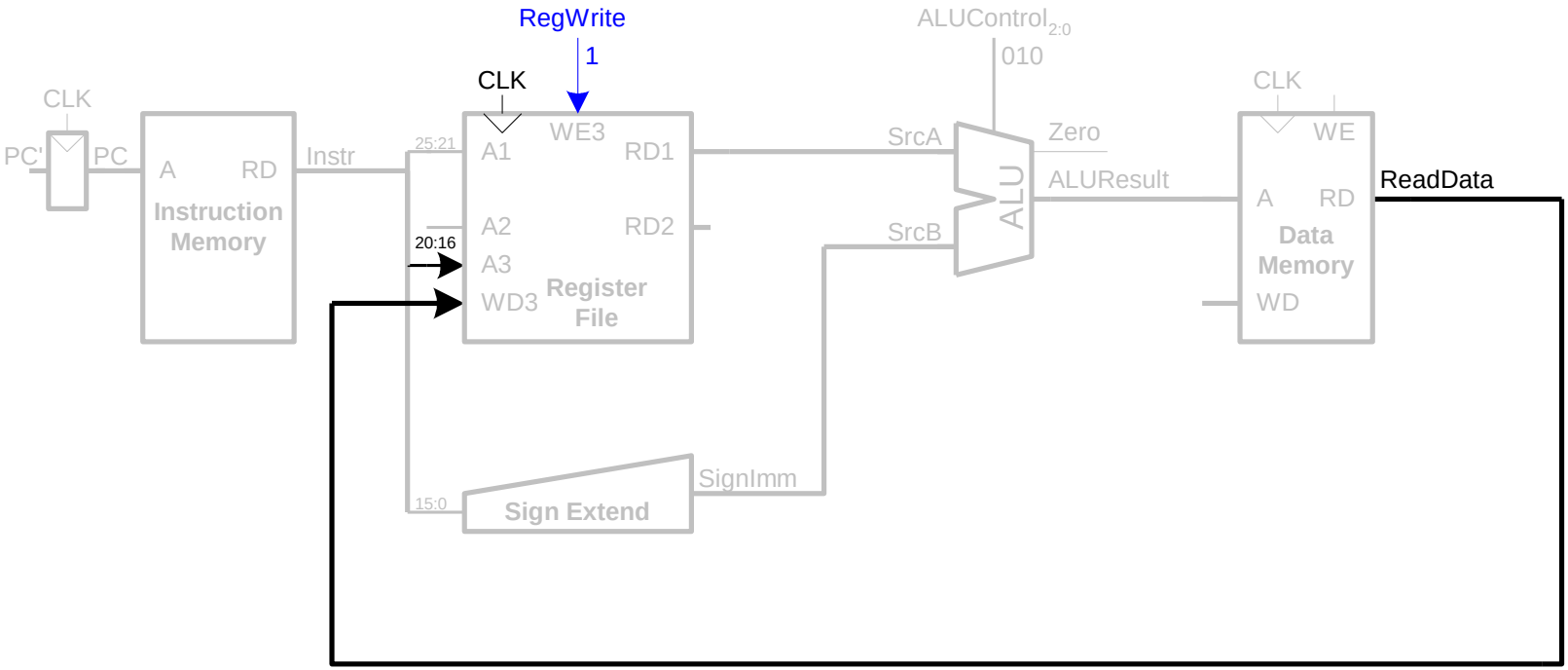
# Single-Cycle Datapath: `lw` address

## STEP 4: Compute the memory address



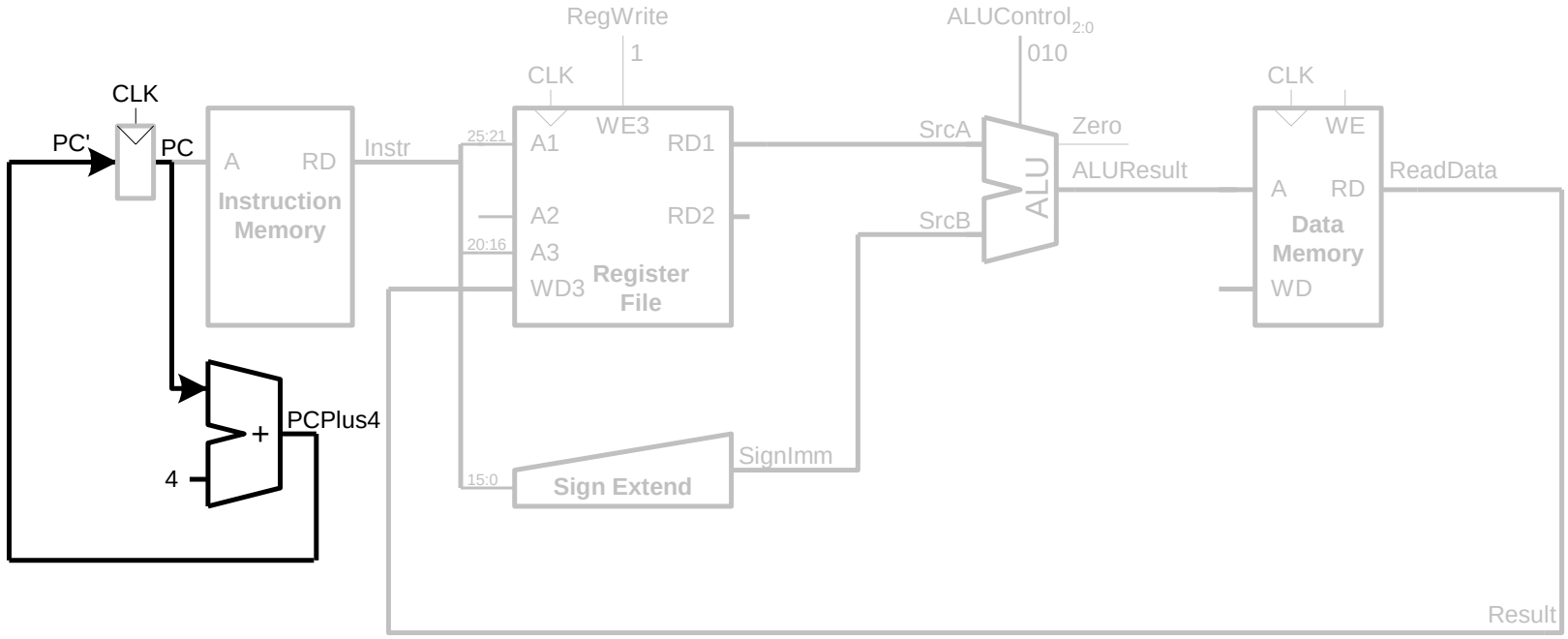
# Single-Cycle Datapath: `lw` Memory Read

- **STEP 5:** Read data from memory and write it back to register file



# Single-Cycle Datapath: `lw` PC Increment

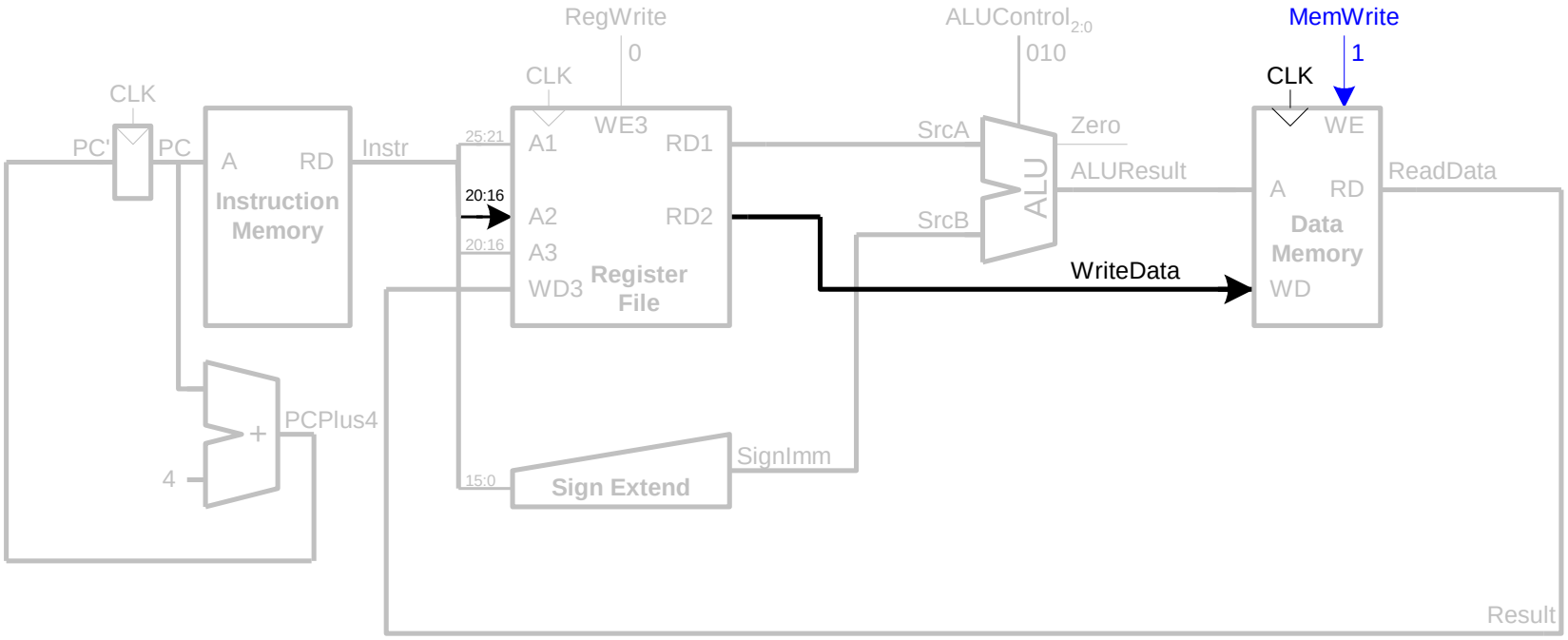
## STEP 6: Determine address of next instruction



# Single-Cycle Datapath: sw

## Write data in rt to memory

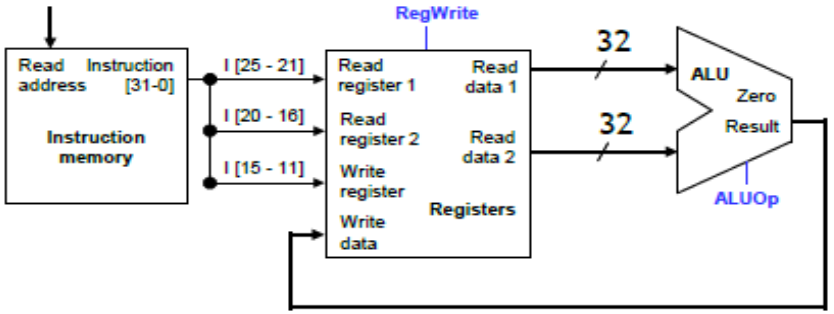
DMEM[ R[rs] + sign_ext(Imm16) ] ← R[rt]

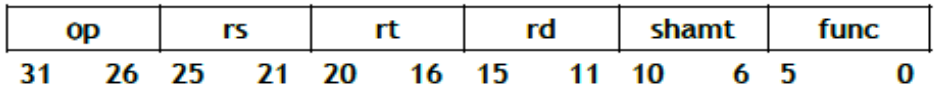


# Single Cycle Processing

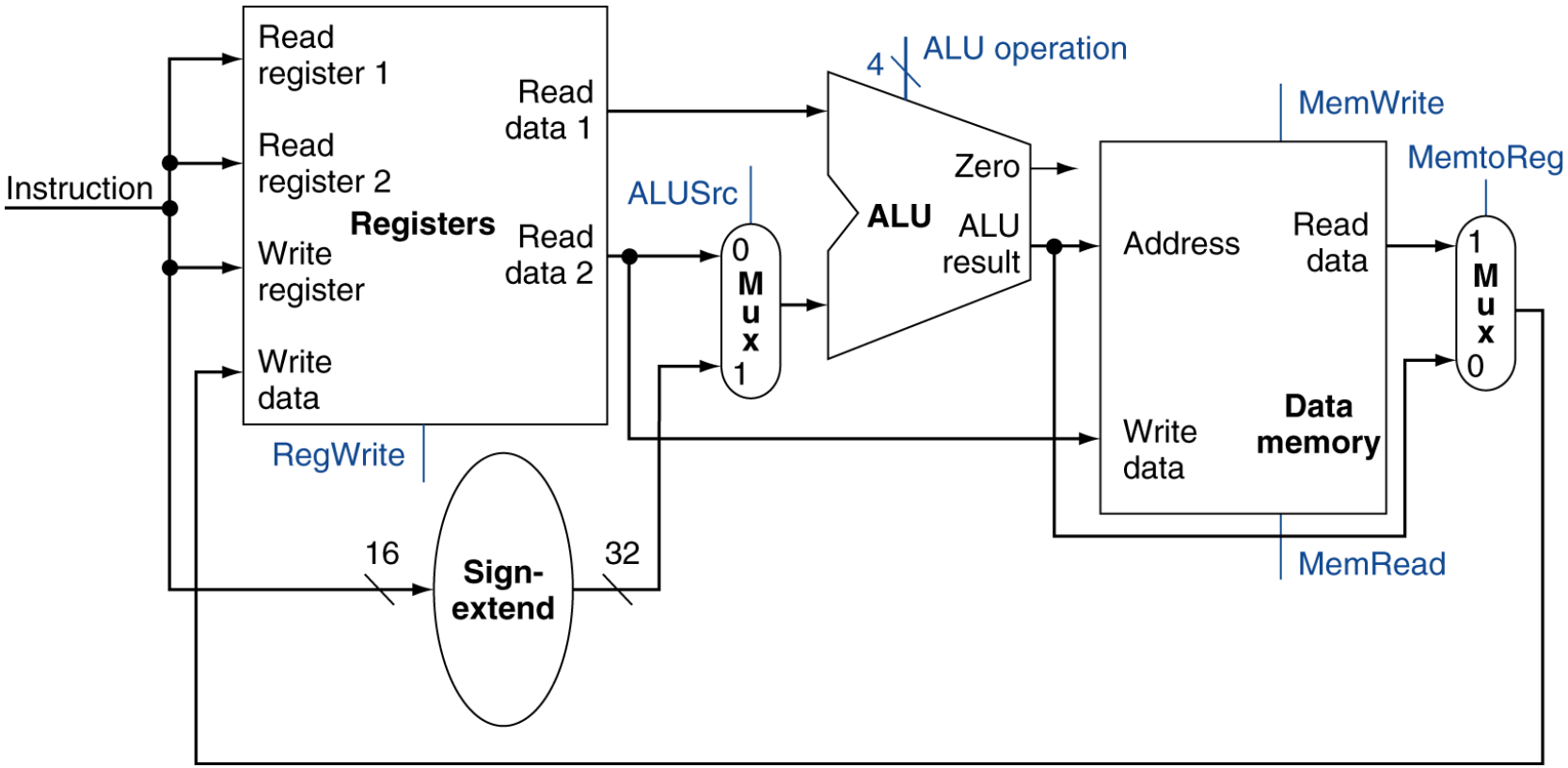
## Executing an R-type instruction

1. Read an instruction from the instruction memory.
2. The source registers, specified by instruction fields rs and rt, should be read from the register file.
3. The ALU performs the desired operation.
4. Its result is stored in the destination register, which is specified by field rd of the instruction word.



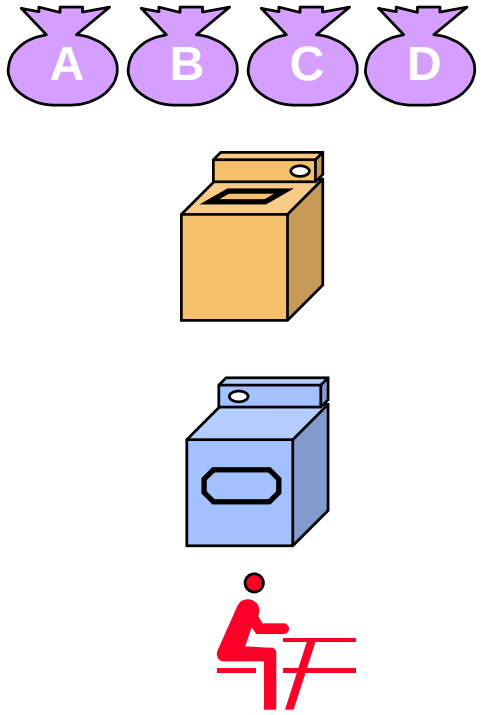



# R-Type/Load/Store Datapath

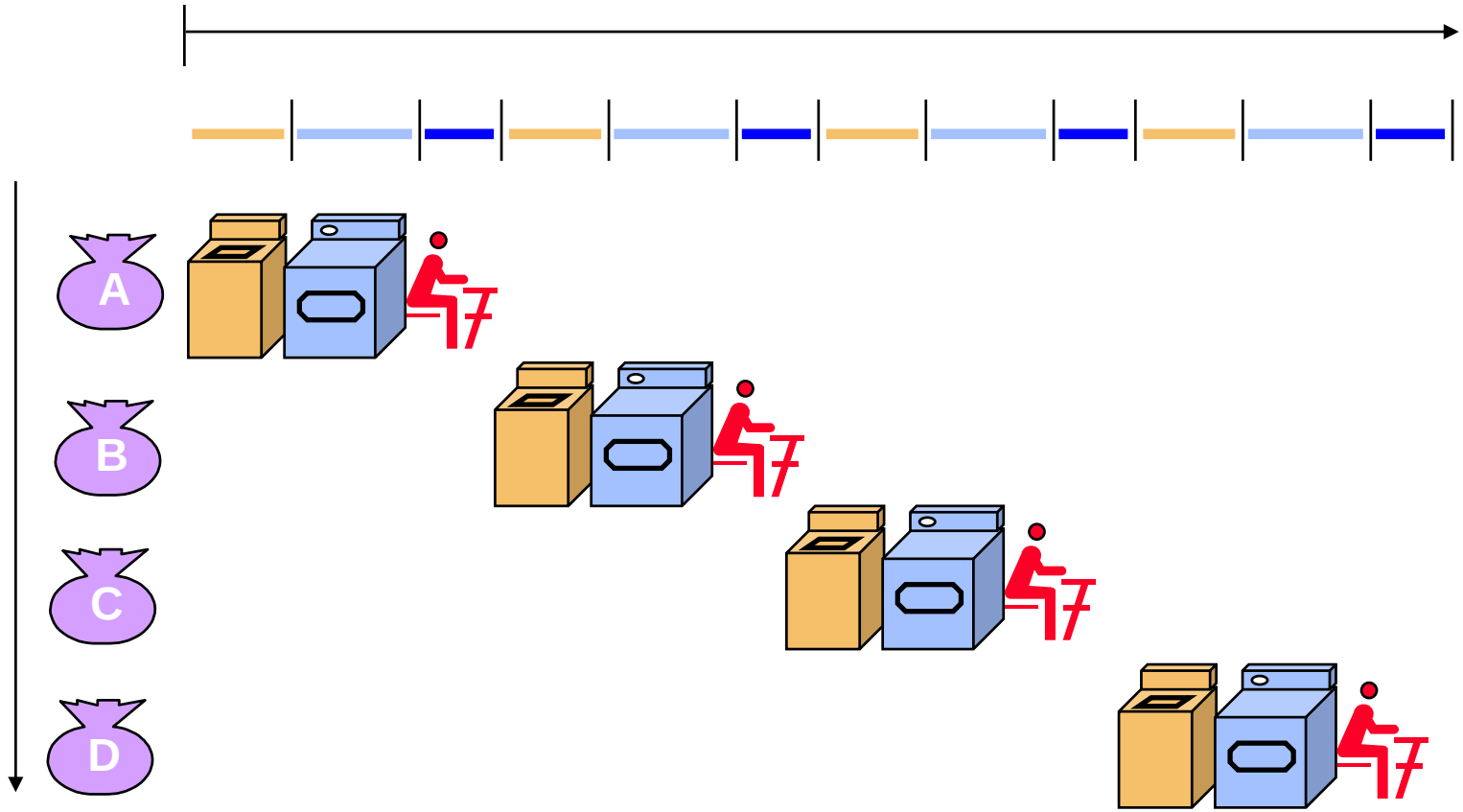


# What Is Pipelining

- Laundry Example
- Ann, Brian, Cathy, Dave
  each have one load of clothes
  to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- "Folder" takes 20 minutes



# What Is Pipelining

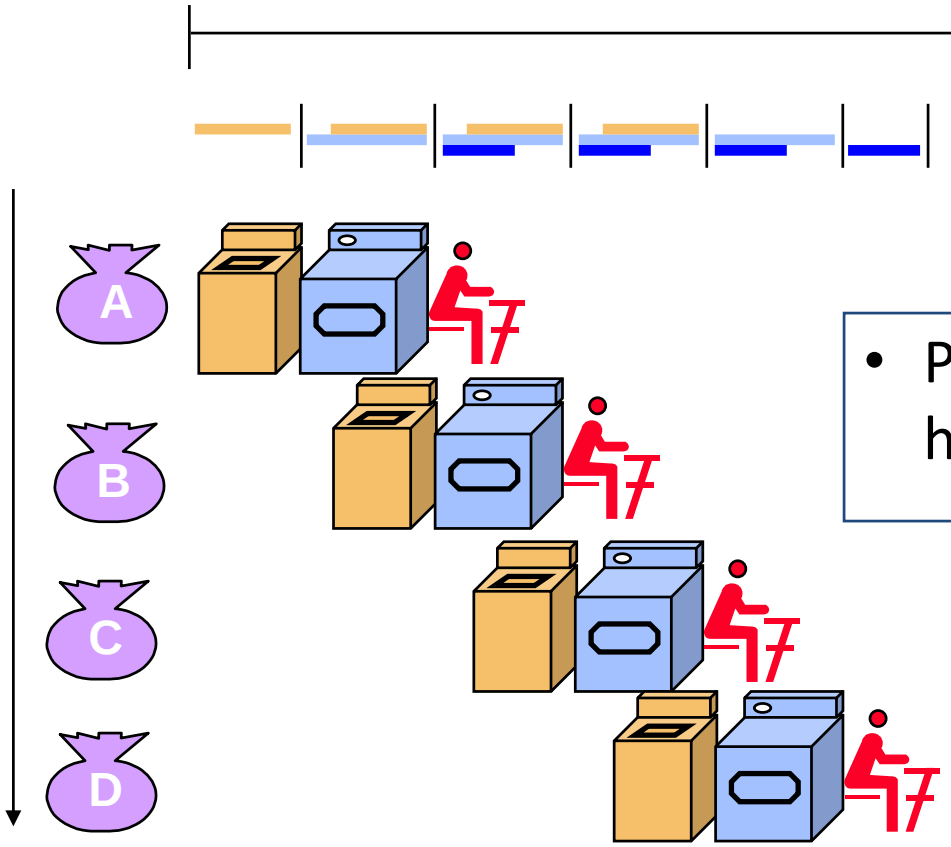



Sequential laundry takes 6 hours for 4 loads

If they learned pipelining, how long would laundry take?

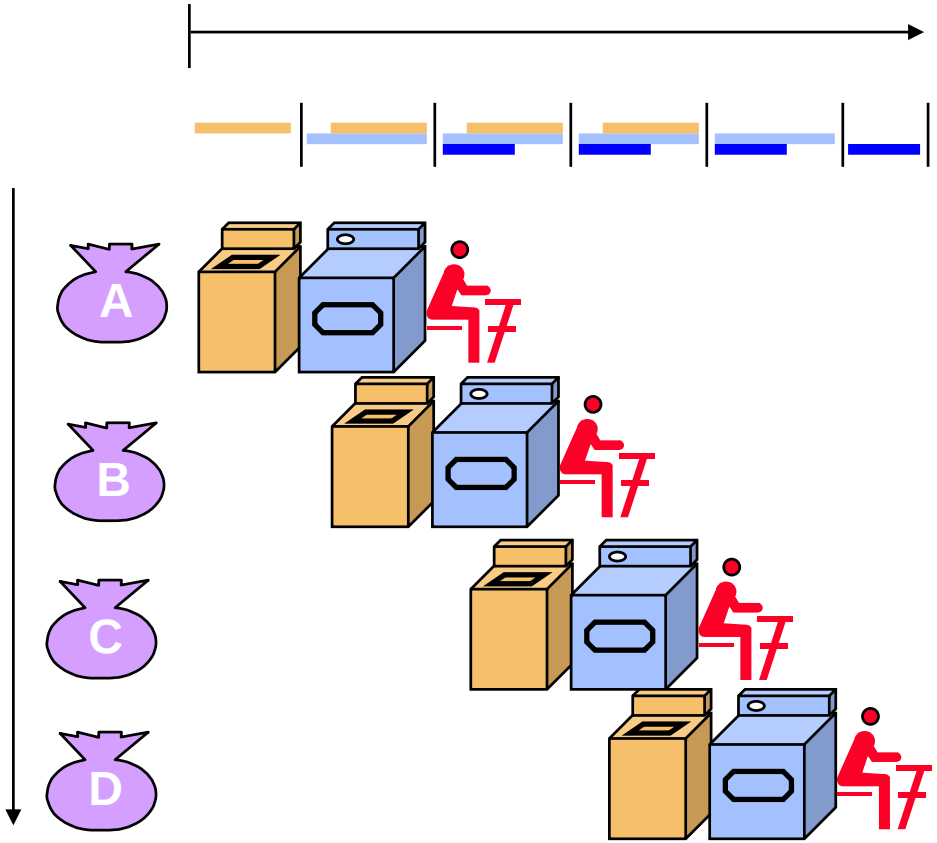
# What Is Pipelining

## Start work ASAP




- Pipelined laundry takes 3.5 hours for 4 loads

# What Is Pipelining




## Pipelining Lessons

- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup

# Pipelining

## Instruction execution review

- Executing a MIPS instruction can take up to five steps.

| Step | Name | Description |
|---|---|---|
| Instruction Fetch | IF | Read an instruction from memory. |
| Instruction Decode | ID | Read source registers and generate control signals. |
| Execute | EX | Compute an R-type result or a branch outcome. |
| Memory | MEM | Read or write the data memory. |
| Writeback | WB | Store a result in the destination register. |

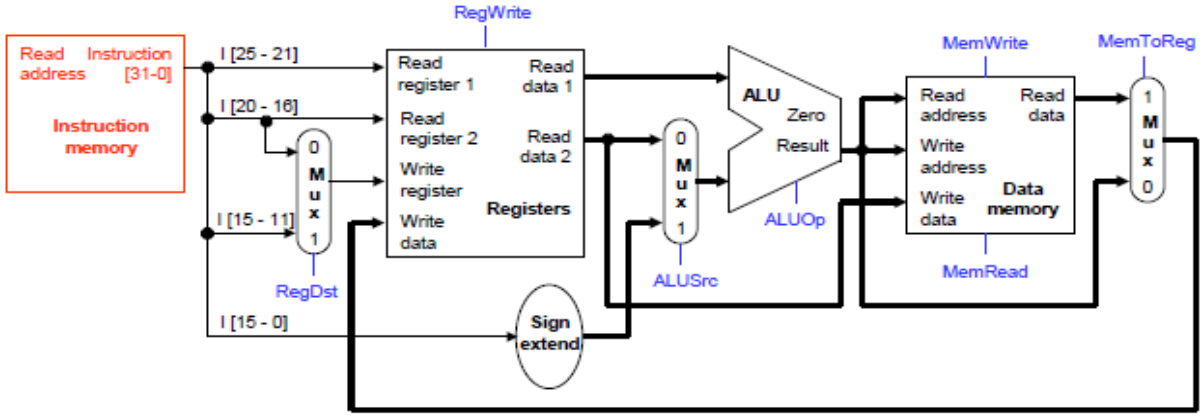
- However, as we saw, not all instructions need all five steps.

| Instruction | Steps required | | | | |
|---|---|---|---|---|---|
| beq | IF | ID | EX | | |
| R-type | IF | ID | EX | | WB |
| sw | IF | ID | EX | MEM | |
| lw | IF | ID | EX | MEM | WB |

# Pipelining

## Review: Instruction Fetch (IF)

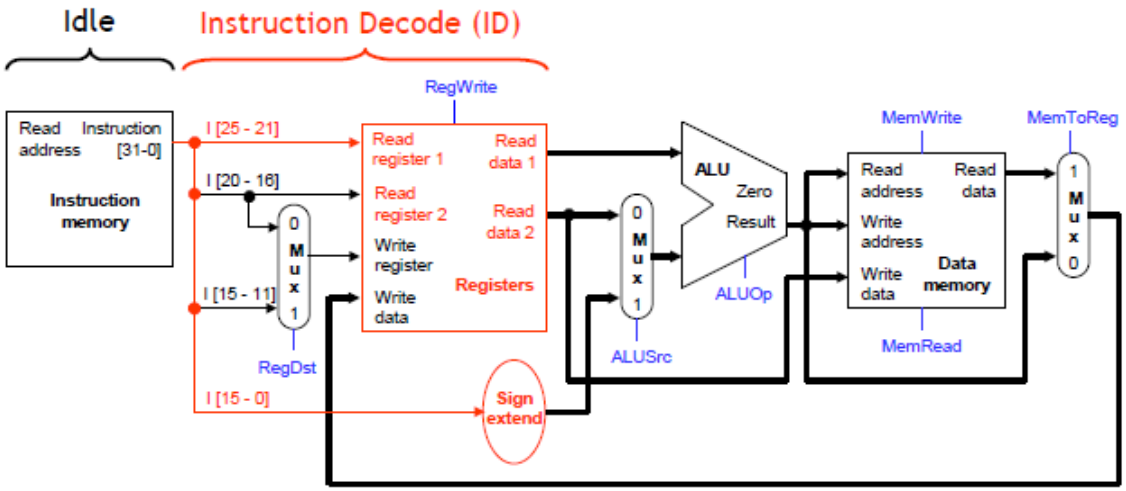
- Let's quickly review how lw is executed in the single-cycle datapath.
- We'll ignore PC incrementing and branching for now.
- In the Instruction Fetch (IF) step, we read the instruction memory.



# Pipelining

## Putting those slackers to work

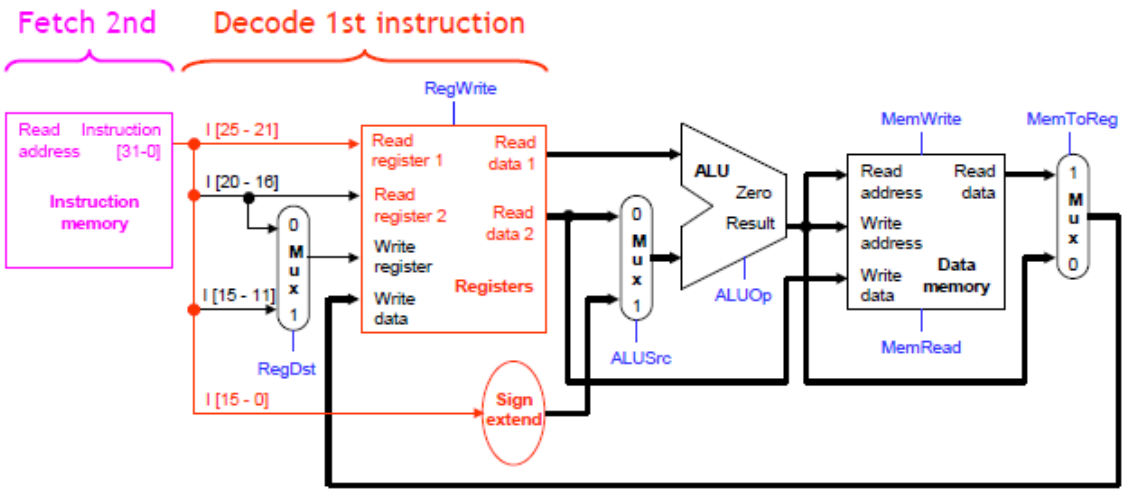
- We shouldn't have to wait for the entire instruction to complete before we can re-use the functional units.
- For example, the instruction memory is free in the Instruction Decode step as shown below, so...



# Pipelining

## Decoding and fetching together

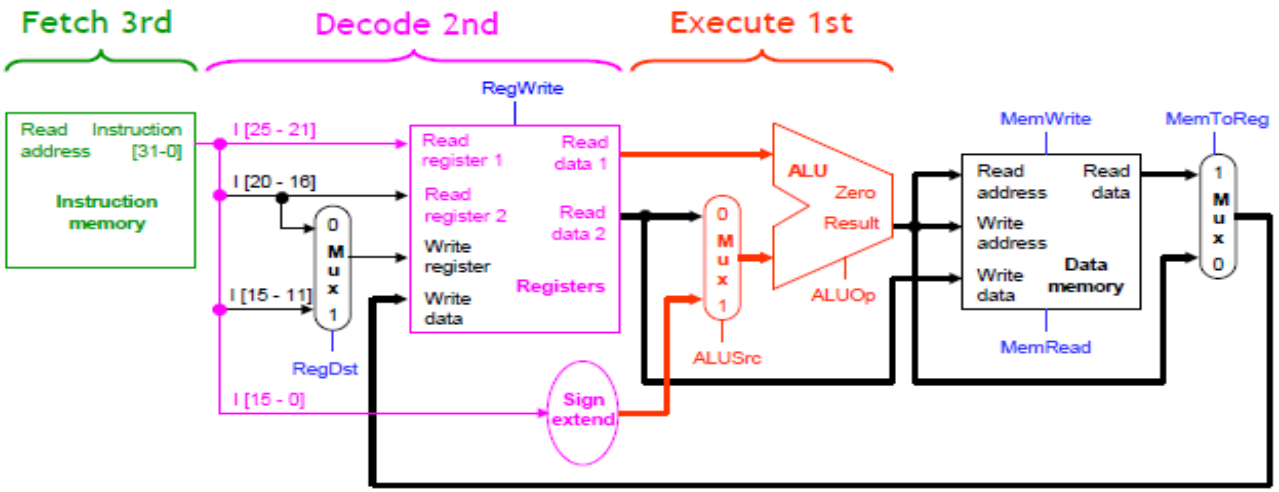
- Why don't we go ahead and fetch the *next* instruction while we're decoding the first one?



# Pipelining

## Executing, decoding and fetching

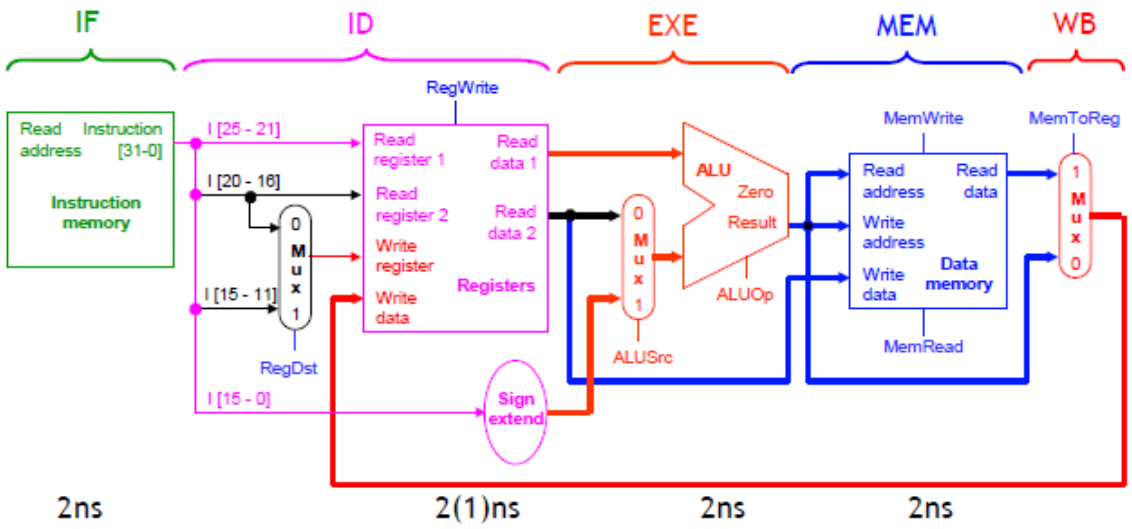
- Similarly, once the first instruction enters its Execute stage, we can go ahead and decode the second instruction.
- But now the instruction memory is free again, so we can fetch the third instruction!



# Pipelining

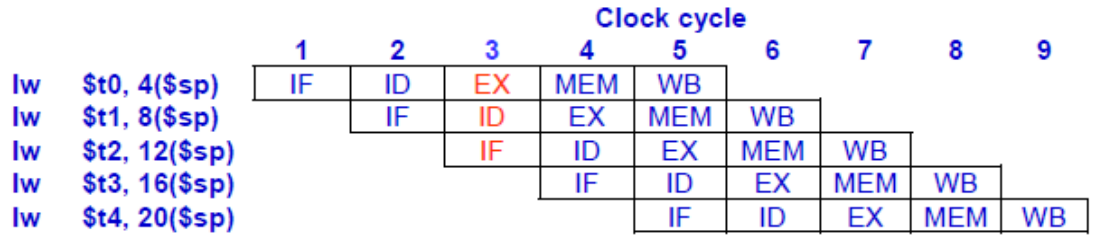
## Break datapath into 5 stages

- Each stage has its own functional units.
- Each stage can execute in 2ns



# Pipelining

## Pipelining Loads

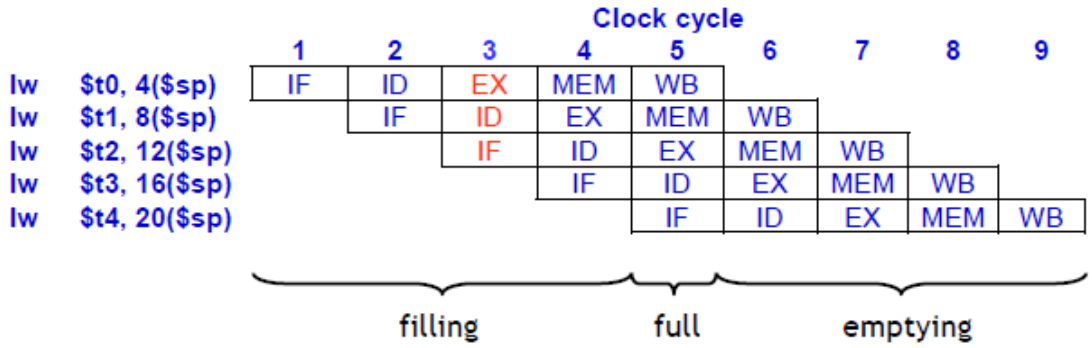



- **A pipeline diagram shows the execution of a series of instructions.**
  - The instruction sequence is shown vertically, from top to bottom.
  - Clock cycles are shown horizontally, from left to right.
  - Each instruction is divided into its component stages. (We show five stages for every instruction, which will make the control unit easier.)

- **This clearly indicates the overlapping of instructions. For example, there are three instructions active in the third cycle above.**
  - The "lw $t0" instruction is in its Execute stage.
  - Simultaneously, the "lw $t1" is in its Instruction Decode stage.
  - Also, the "lw $t2" instruction is just being fetched.

# Pipelining

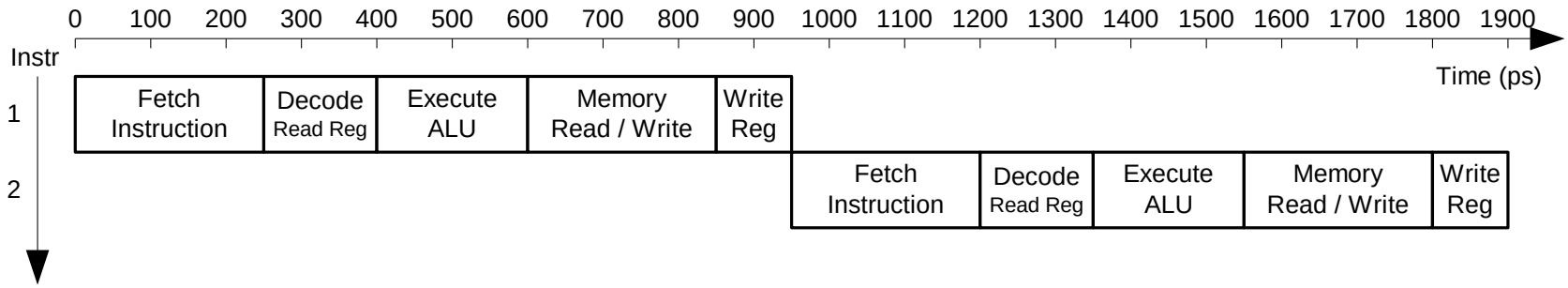
## Pipelining terminology



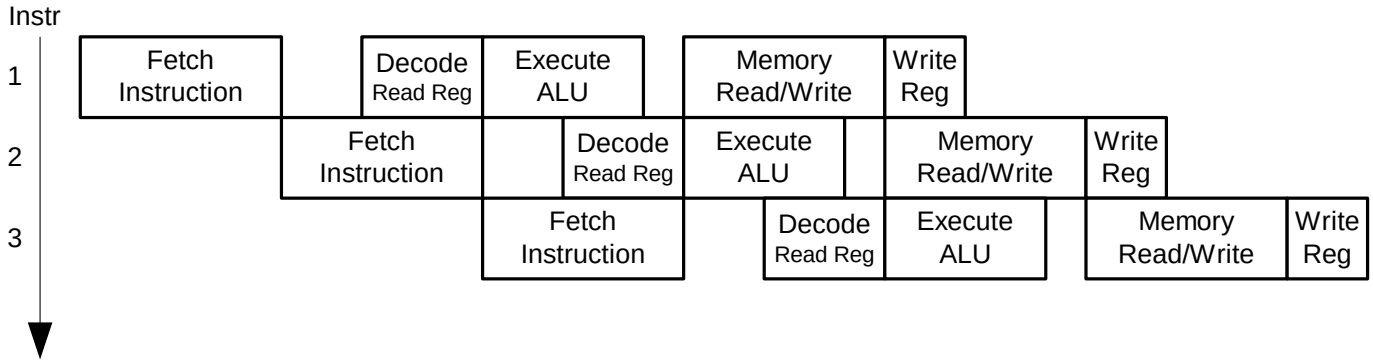

- The pipeline depth is the number of stages—in this case, five.
- In the first four cycles here, the pipeline is filling, since there are unused functional units.
- In cycle 5, the pipeline is full. Five instructions are being executed simultaneously, so all hardware units are in use.
- In cycles 6-9, the pipeline is emptying.

# Single-Cycle vs. Pipelined

## Single-Cycle




## Pipelined



# Pipeline Hurdles

**A.1 What is Pipelining?**

**A.2 The Major Hurdle of Pipelining-Structural Hazards**

-- Structural Hazards

– Data Hazards

– Control Hazards

**A.3 How is Pipelining Implemented**

**A.4 What Makes Pipelining Hard to Implement?**

**A.5 Extending the MIPS Pipeline to Handle Multi-cycle Operations**

Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle

- **Structural hazards:** HW cannot support this combination of instructions (single person to fold and put clothes away) – Required resource is busy
- **Data hazards:** Instruction depends on result of prior instruction still in the pipeline (missing sock)
- **Control hazards:** Pipelining of branches & other instructions that change the PC
- Common solution is to **stall** the pipeline until the hazard is resolved, inserting one or more "**bubbles**" in the pipeline

# Recap: Pipeline Hazards

- Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: attempt to use the same resource two different ways at the same time
    - One memory
  - Data hazards: attempt to use data before it is ready
    - Instruction depends on result of prior instruction still in the pipeline
  - Control hazards: attempt to make a decision before condition is evaluated
    - Branch instructions
- Pipeline implementation need to detect and resolve hazards