Computer Architecture Great Ideas in Computer Architecture (Machine Structures) Pipeline Parallelism

Pipeline: Introduction

CSCE430/830

Lecturer: Prof. Hong Jiang

Courtesy of Prof. Yifeng Zhu, U of Maine

Fall, 2006

Mike Franklin
Dan Garcia
http://inst.eecs.Berkeley.edu/~cs61c/fa11

Fall, 2011

Pipelining Outline

- Introduction
  - Defining Pipelining
  - Pipelining Instructions
- Hazards
  - Structural hazards
  - Data Hazards
  - Control Hazards
- Performance
- Controller implementation

You Are Here!

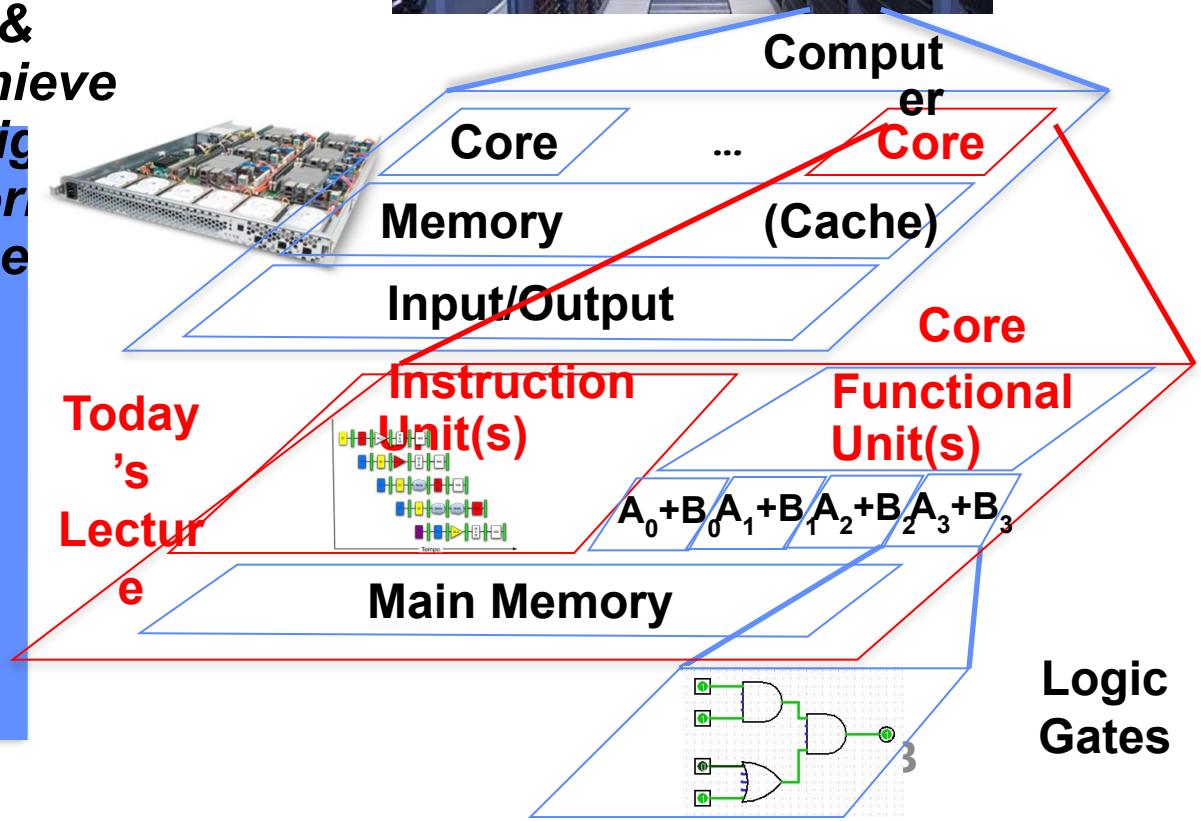
- **Parallel Requests**
  - Assigned to computer
  - e.g., Search "Katz"
- **Parallel Thread**
  - Assigned to core
  - e.g., Lookup, Ads
- **Parallel Instructions**
  - >1 instruction @ one time
  - e.g., 5 pipelined instructions
- **Parallel Data**
  - >1 data item @ one time
  - e.g., Add of 4 pairs of words
- **Hardware descriptions**

Software
Hardware

Harness Parallelism & Achieve High Performance

Warehouse Scale Computer

Smart Phone



All gates functioning in parallel at same time

Pipeline

What is Pipelining?

- ***Pipelining*** is a key implementation technique used to build **fast processors** (speeding up execution of instructions).

- ***Key idea***:

  overlap in time execution of multiple instructions

- A **pipeline** within a processor is similar to a car assembly line. Each assembly station is called a *pipe stage* or a *pipe segment.*

Hyundai car assembly line

Pipeline metrics / notations

- The **throughput** of an instruction pipeline is the measure of how often an instruction exits the pipeline.

- Pipeline **latency**: how long does it take to execute a single instruction in the pipeline.

- Pipeline **depth**: number of stages in a pipeline

- **Design issue - Balance the length of each pipeline stage**

$$\text{Throughput} = \frac{\text{Depth of the pipeline}}{\text{Time per instruction on unpipelined machine}}$$

Single Cycle Performance

- Assume time for actions are
  - 100ps for register read or write; 200ps for other events
- Clock rate is?

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

- **Time per instruction on unpipelined machine differ**
- What can we do to improve clock rate?

Single Cycle Performance

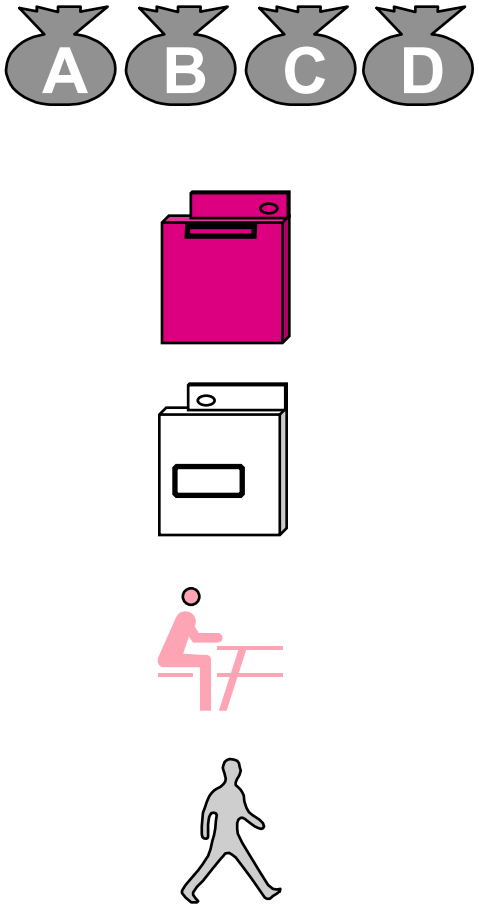
- Assume time for actions are
  - 100ps for register read or write; 200ps for other events
- Clock rate is?

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

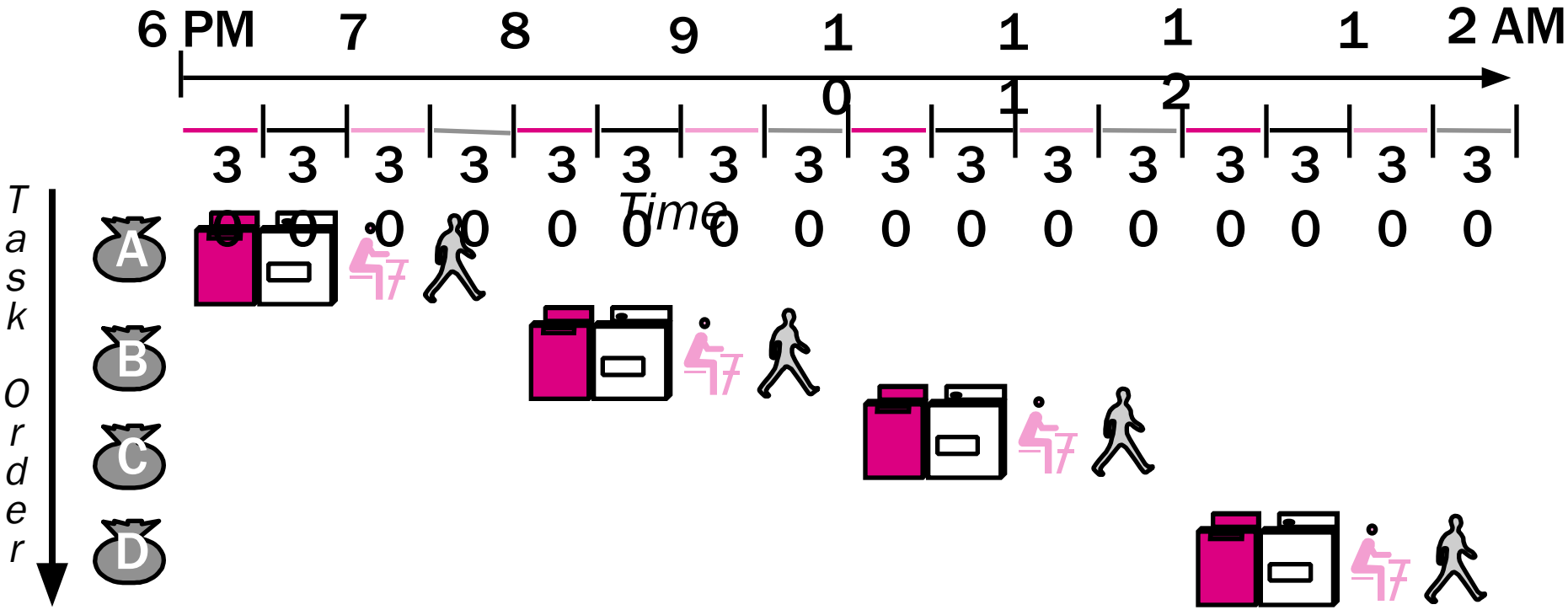
- We adjust clock rate according to **slowest** instruction / operation
- Will this improve performance as well?
- Want increased clock rate to mean faster programs

The Laundry Analogy

- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 30 minutes
- “Folder” takes 30 minutes
- “Stasher” takes 30 minutes to put clothes into drawers

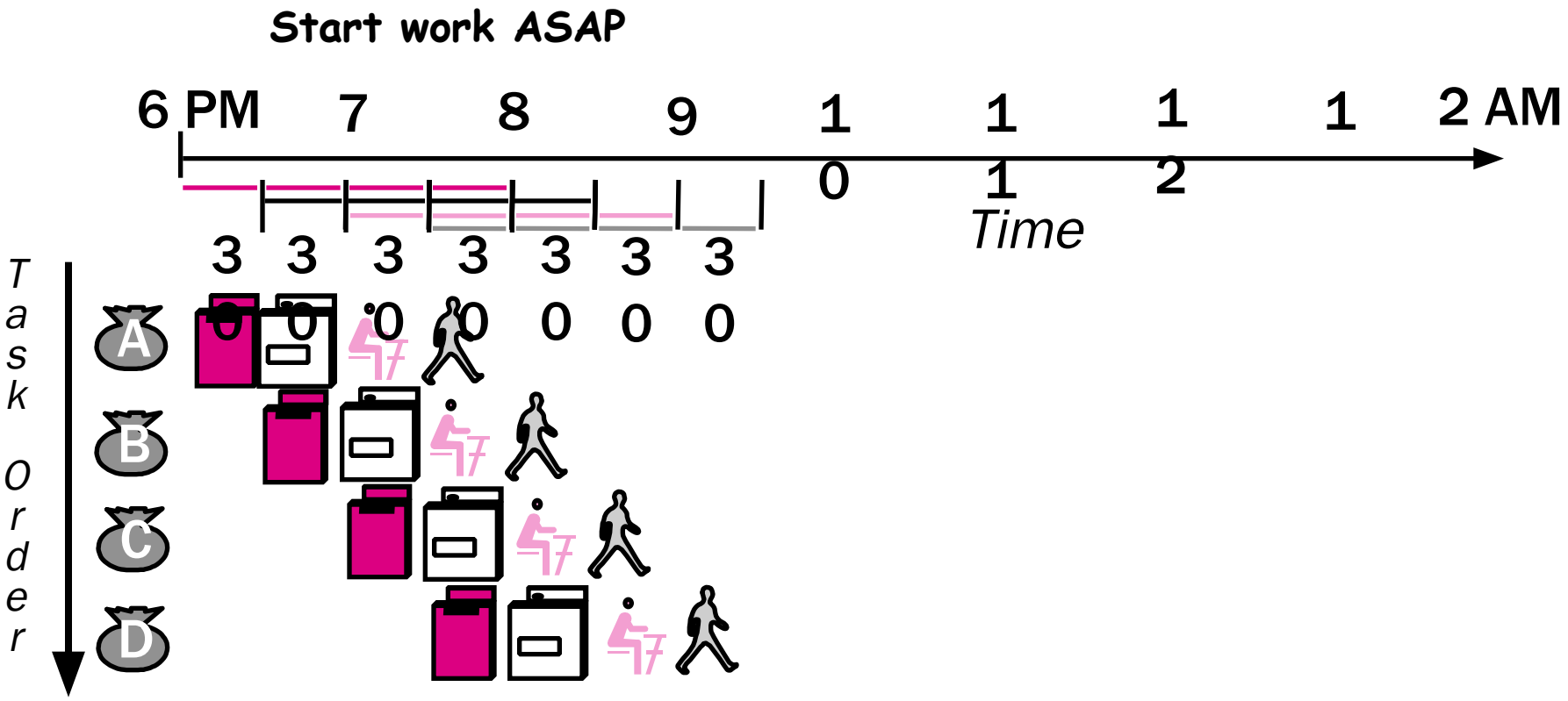


If we do laundry sequentially...



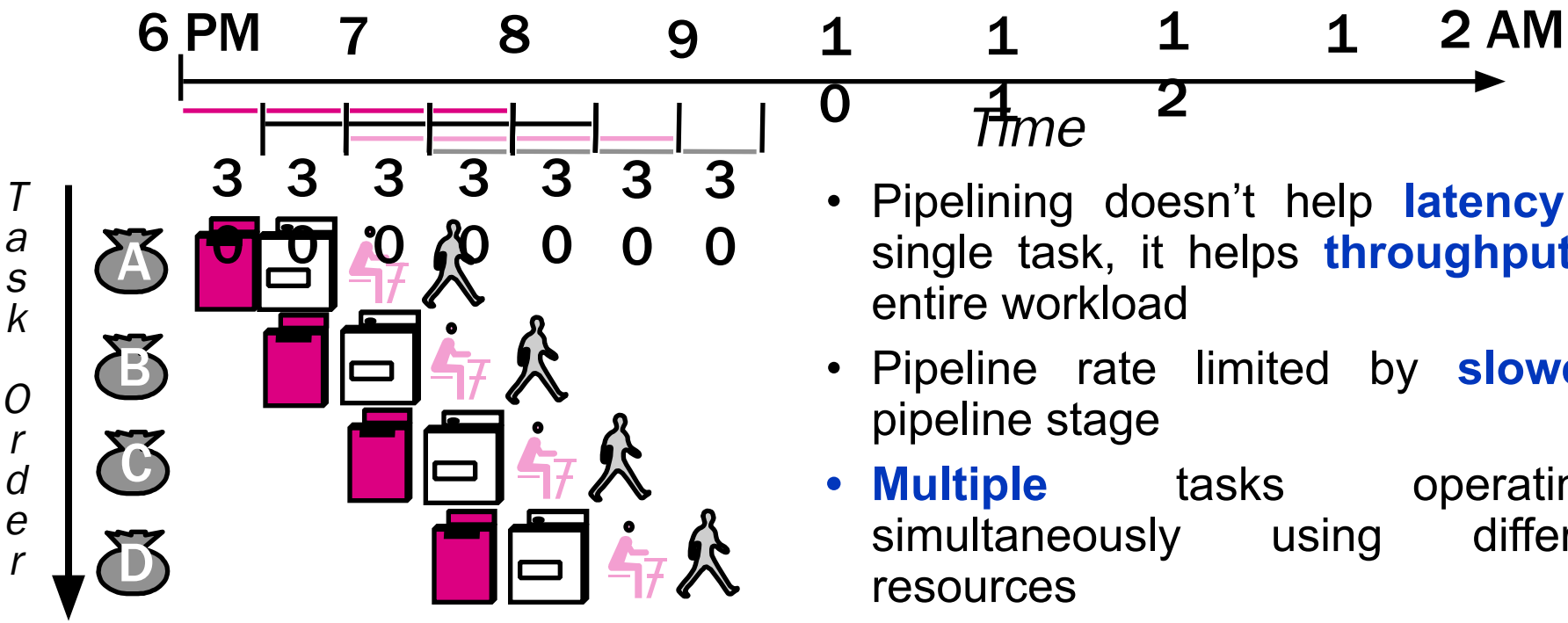
- **Time Required: 8 hours for 4 loads**

To Pipeline, We Overlap Tasks



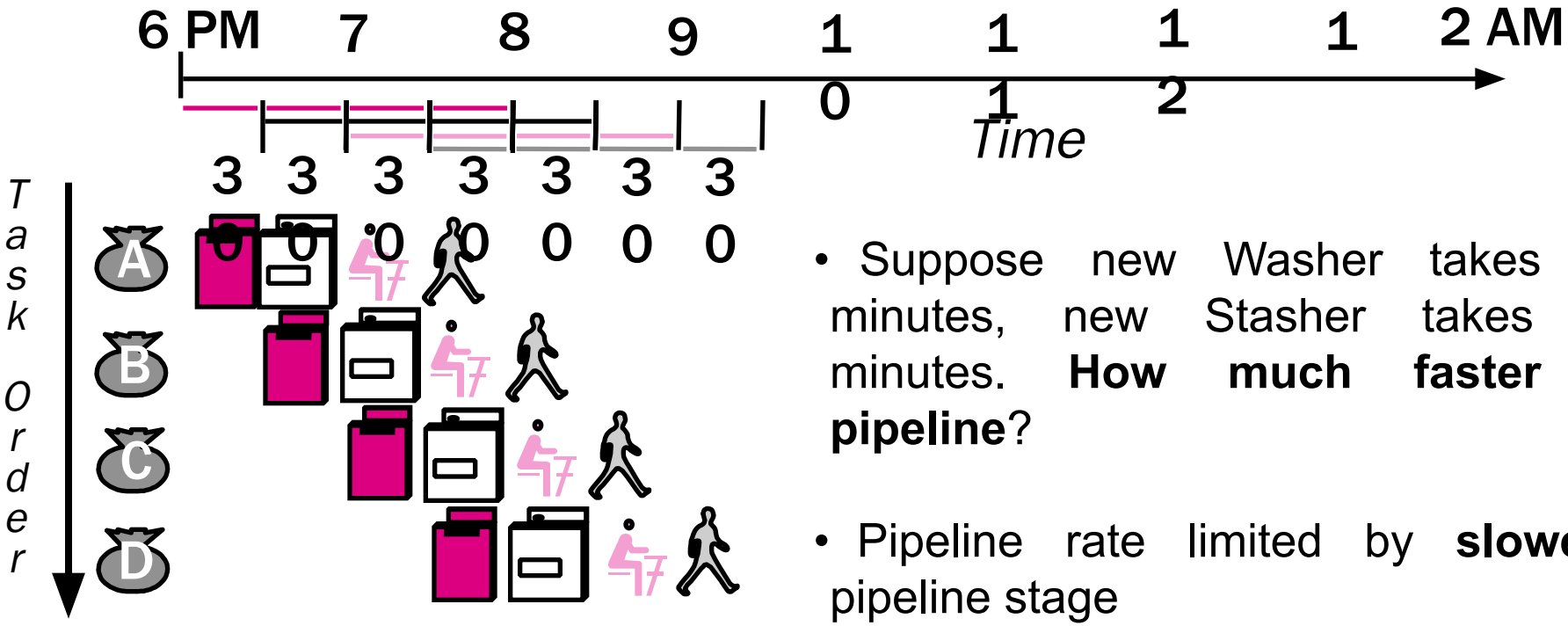
- **Time Required: 3.5 Hours for 4 Loads**

To Pipeline, We Overlap Tasks (I)



- 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 using different resources
- Potential speedup = **Number pipe stages**
- Unbalanced lengths of pipe stages reduces speedup
- Time to "**fill**" pipeline and time to "**drain**" it reduces speedup: **2.3X v. 4X in this example**

To Pipeline, We Overlap Tasks (II)

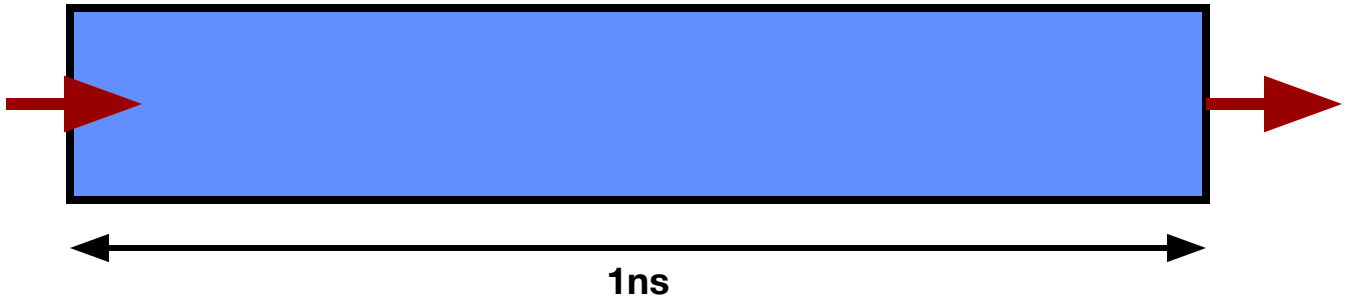


- Suppose new Washer takes 20 minutes, new Stasher takes 20 minutes. **How much faster is pipeline**?
- Pipeline rate limited by **slowest** pipeline stage
- Unbalanced lengths of pipe stages reduces speedup

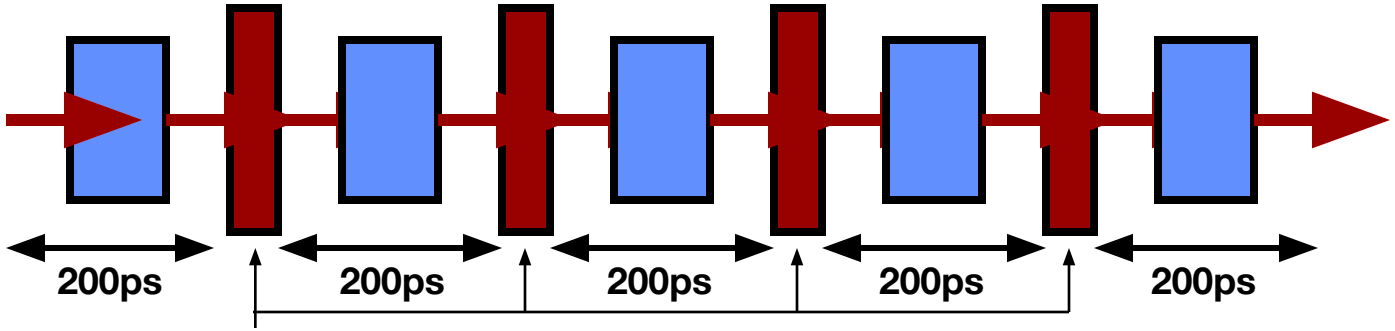
Pipelining a Digital System

1 nanosecond = 10^-9 second
1 picosecond = 10^-12 second

- **Key idea: break big computation up into pieces**



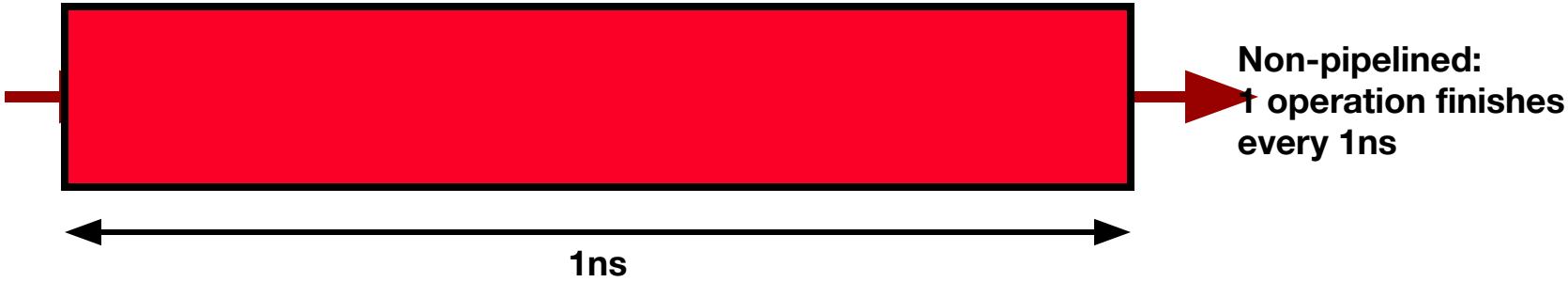
- **Separate each piece with a <u>pipeline register (latch)</u>**

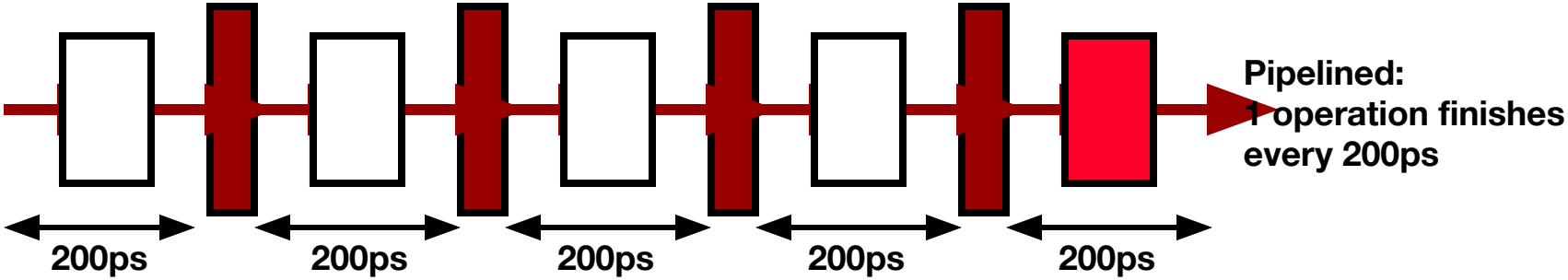


Pipeline Register –
Per phase/stage local information storage unit
Forward traveling signals at each stage are latched

Pipelining a Digital System

- Why do this? Because it's <u>faster</u> for repeated computations





Comments about pipelining

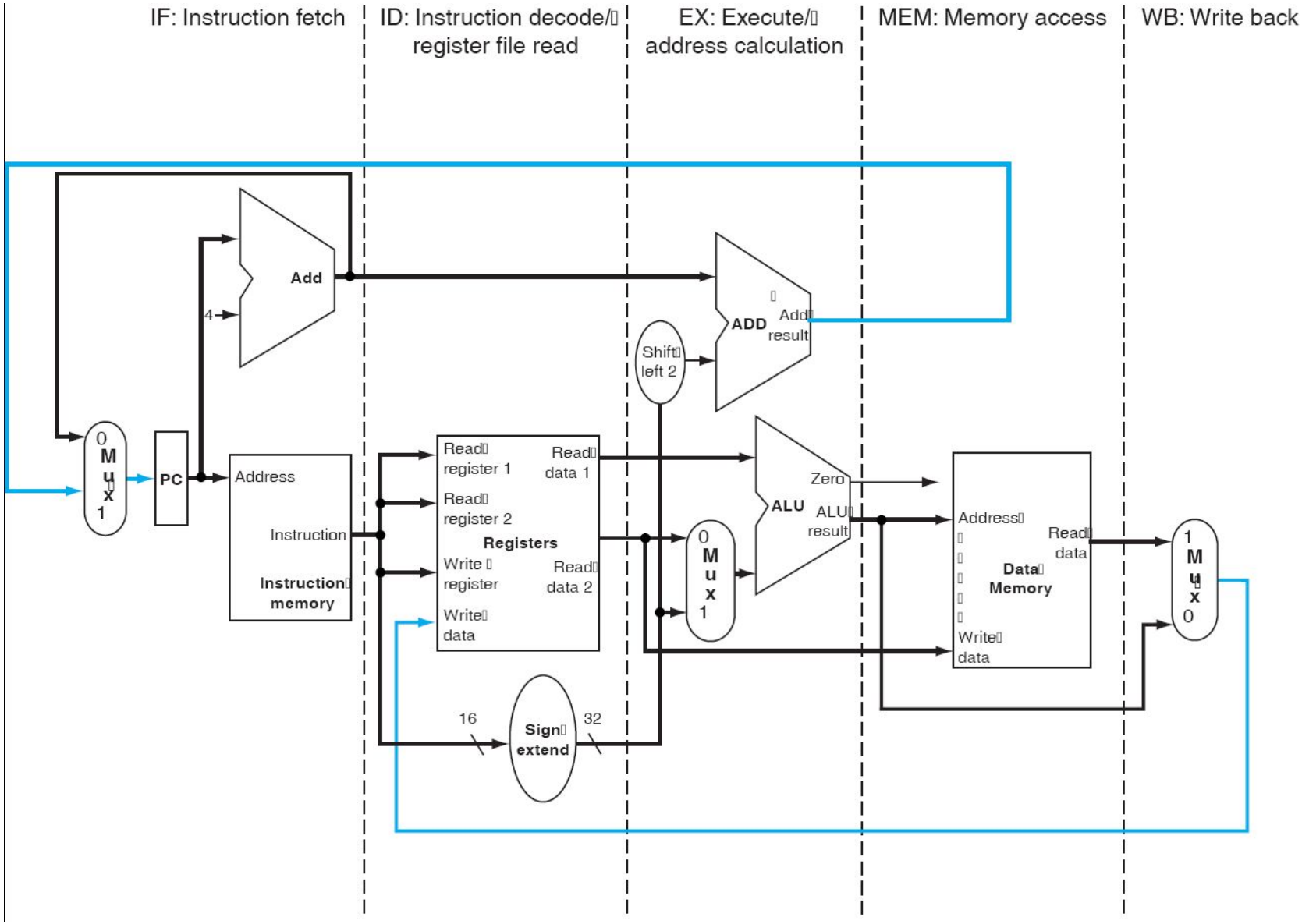
- Pipelining increases throughput, but not latency
  - Answer available every 200ps, BUT
  - A single computation still takes 1ns
- Limitations:
  - Computations must be divisible into stage size
  - Pipeline registers add overhead

Pipelining a Processor

- **Recall the 5 steps in instruction execution:**
  1. **Instruction Fetch, Increment PC (IF)**
  2. **Instruction Decode and Register Read (ID)**
  3. **Execution operation or calculate address (EX)**
  4. **Memory access - Read Data / Write Data from / to Memory (MEM)**
  5. **Write back result into register (WB)**

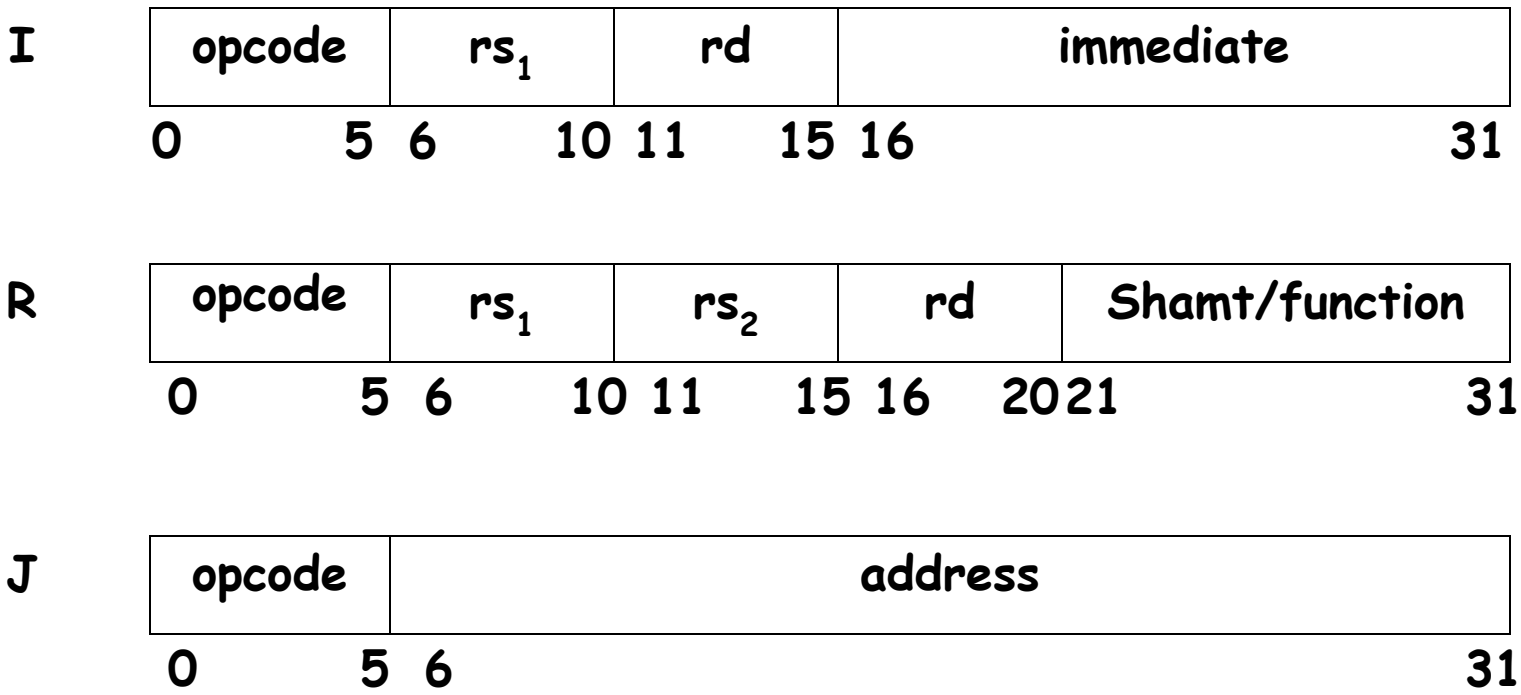
- **Review: Single-Cycle Processor**
  - **All 5 steps done in a single clock cycle**
  - **Dedicated hardware required for each step**

Review - Single-Cycle Processor



- *What do we need to add to actually split the datapath into stages?*

MIPS Instruction Formats



Fixed-field decoding

1st and 2nd Instruction cycles

- **Instruction fetch (IF)**

  $IR \leftarrow Mem[PC];$

  $NPC \leftarrow PC + 4$

- **Instruction decode & register fetch (ID)**

  $A \leftarrow Regs[IR_{6..10}];$

  $B \leftarrow Regs[IR_{11..15}];$

  $Imm \leftarrow ((IR_{16})^{16} \# \# IR_{16..31})$

3rd Instruction cycle

- Execution & effective address (EX)
  - Memory reference
    - » ALUOutput ← A + Imm
  - Register - Register ALU instruction
    - » ALUOutput ← A *func* B
  - Register - Immediate ALU instruction
    - » ALUOutput ← A *op* Imm
  - Branch
    - » ALUOutput ← NPC + Imm; Cond ← (A *op* 0)

4th Instruction cycle

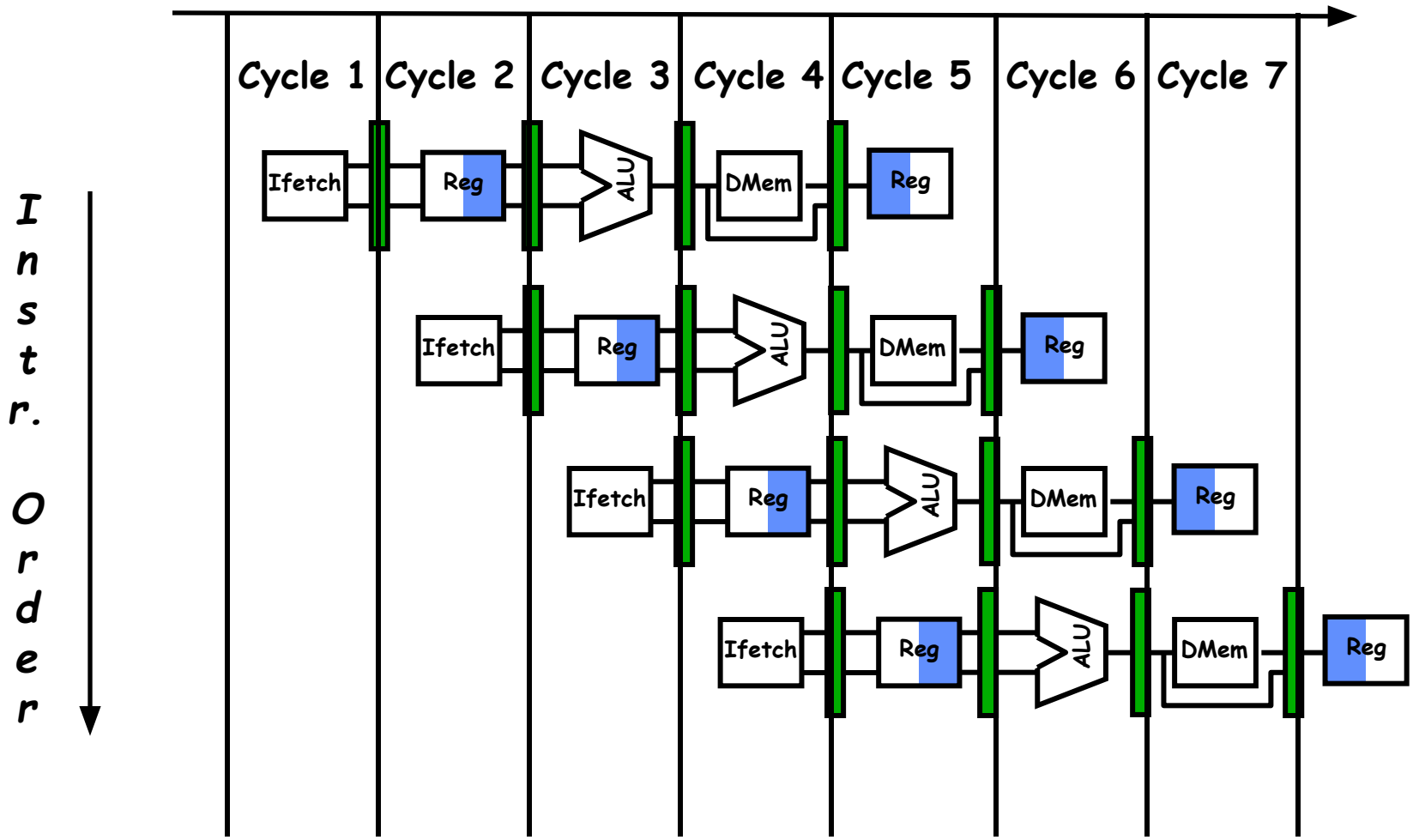
- Memory access & branch completion (MEM)
  - Memory reference
    - » PC ← NPC
    - » LMD ← Mem[ALUOutput] (load)
    - » Mem[ALUOutput] ← B (store)
  - Branch
    - » if (cond) PC ← ALUOutput; else PC ← NPC

5th Instruction cycle

- Write-back (WB)
  - Register - register ALU instruction
    - » $Regs[IR_{16..20}] \leftarrow ALUOutput$
  - Register - immediate ALU instruction
    - » $Regs[IR_{11..15}] \leftarrow ALUOutput$
  - Load instruction
    - » $Regs[IR_{11..15}] \leftarrow LMD$

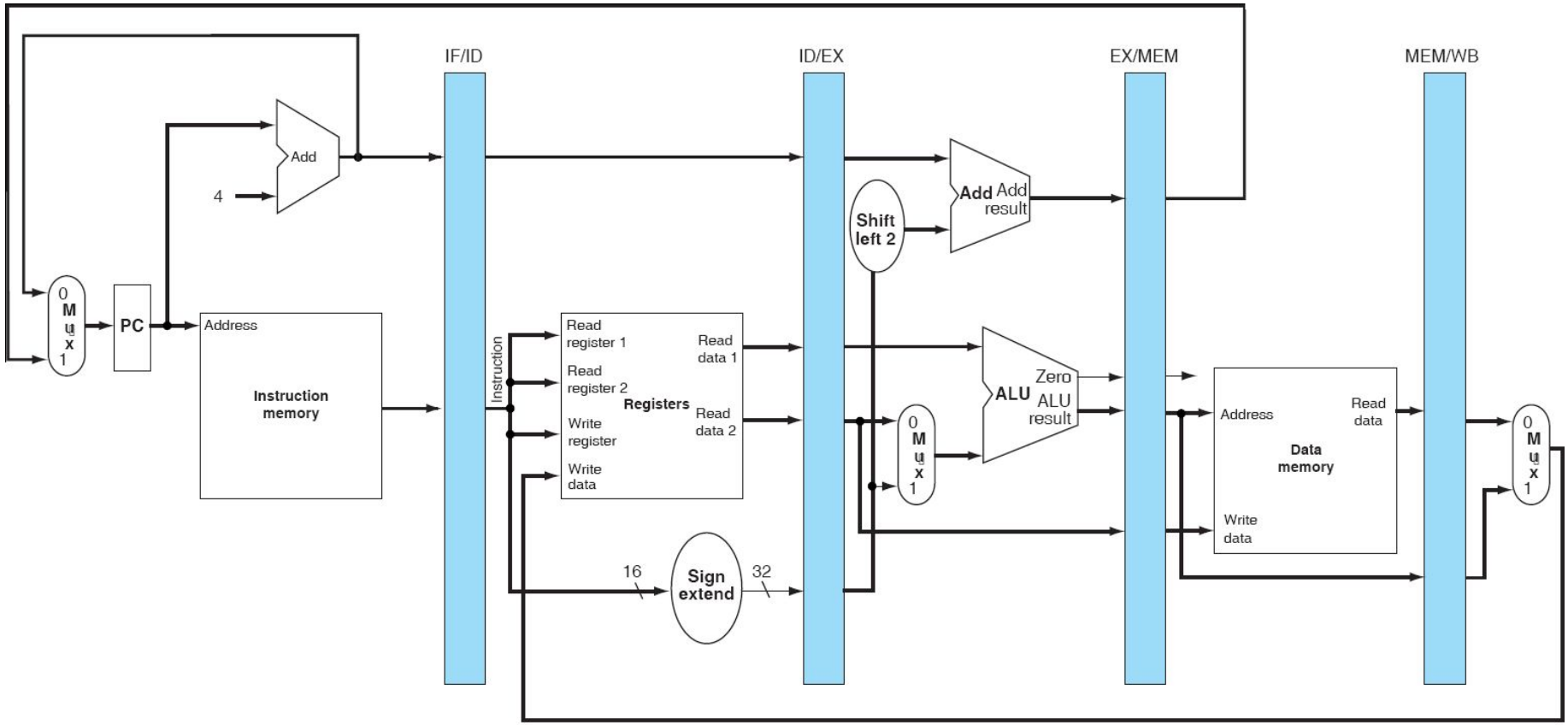
The Basic Pipeline For MIPS

In Reg stage, right half highlight *read*, left half *write*



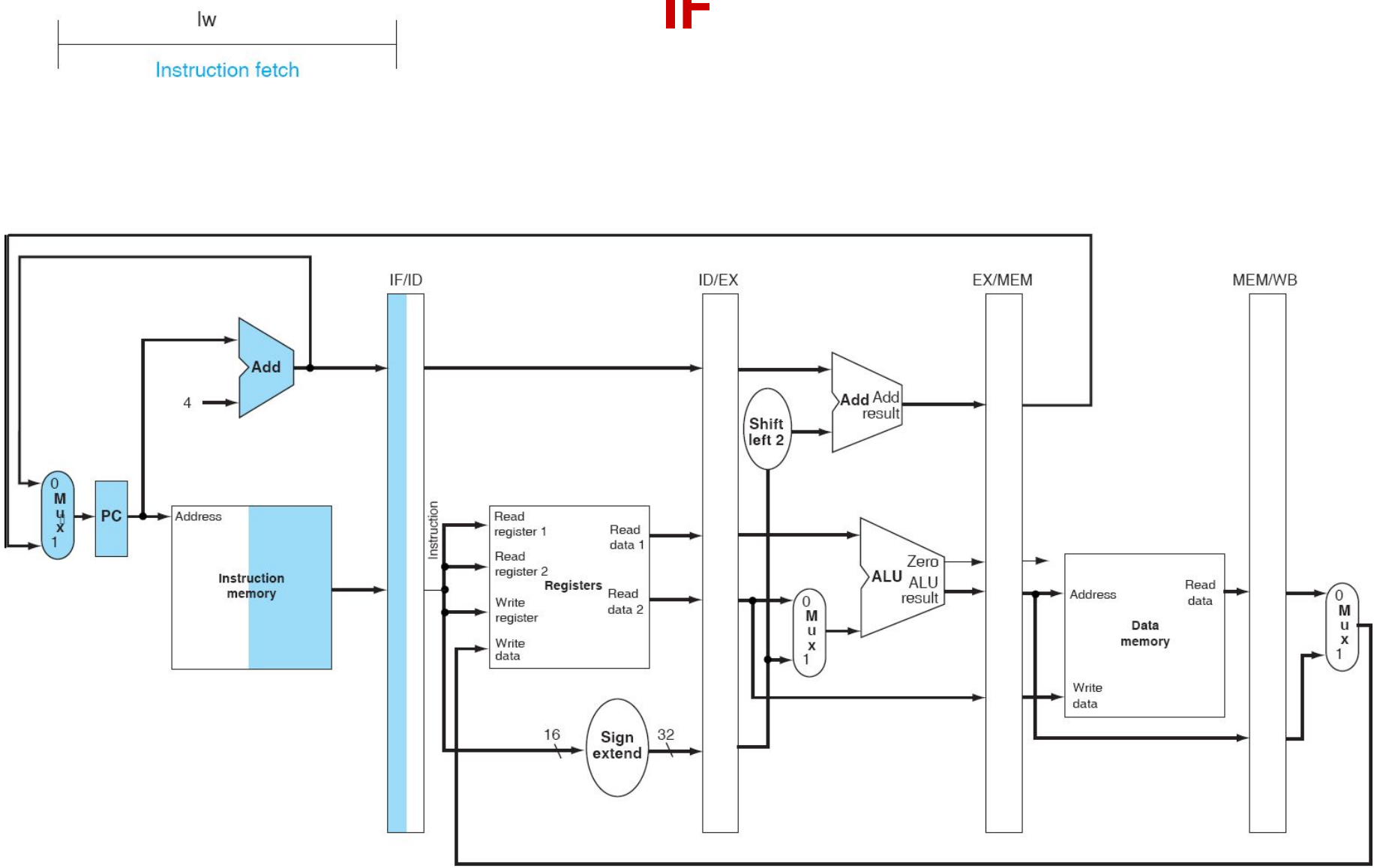
What do we need to add to actually split the datapath into stages?

Basic Pipelined Processor

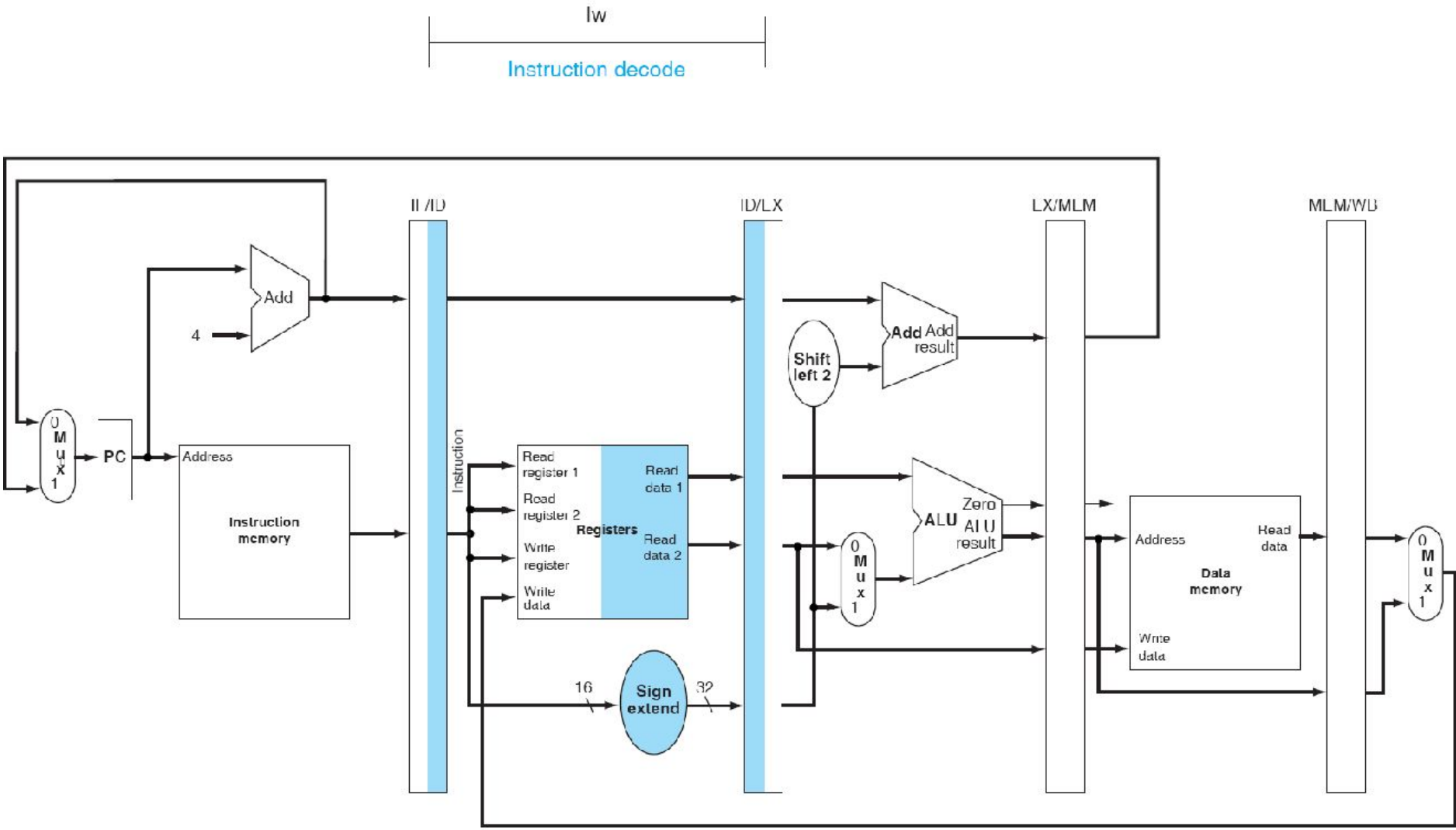


Pipeline example: lw

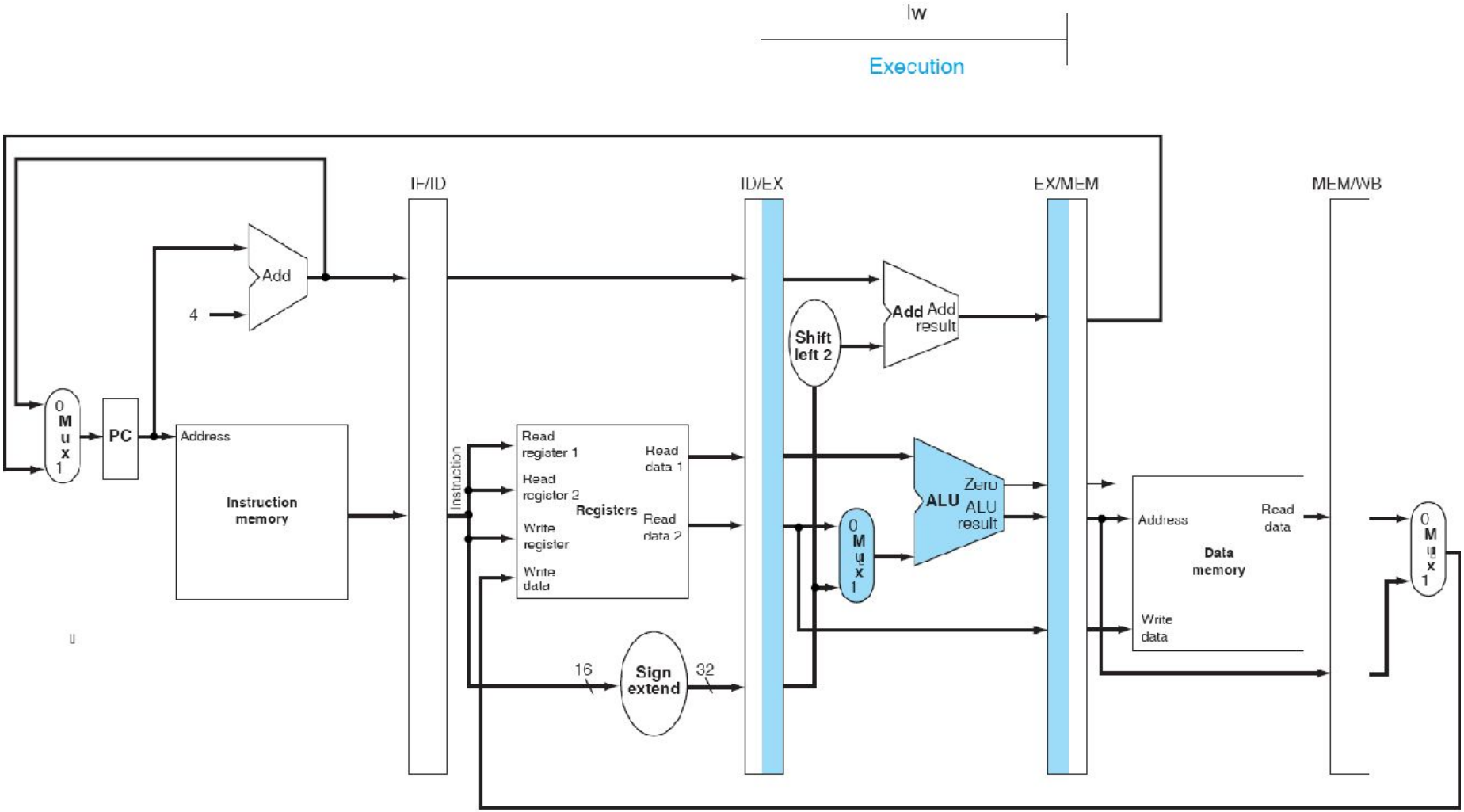
IF



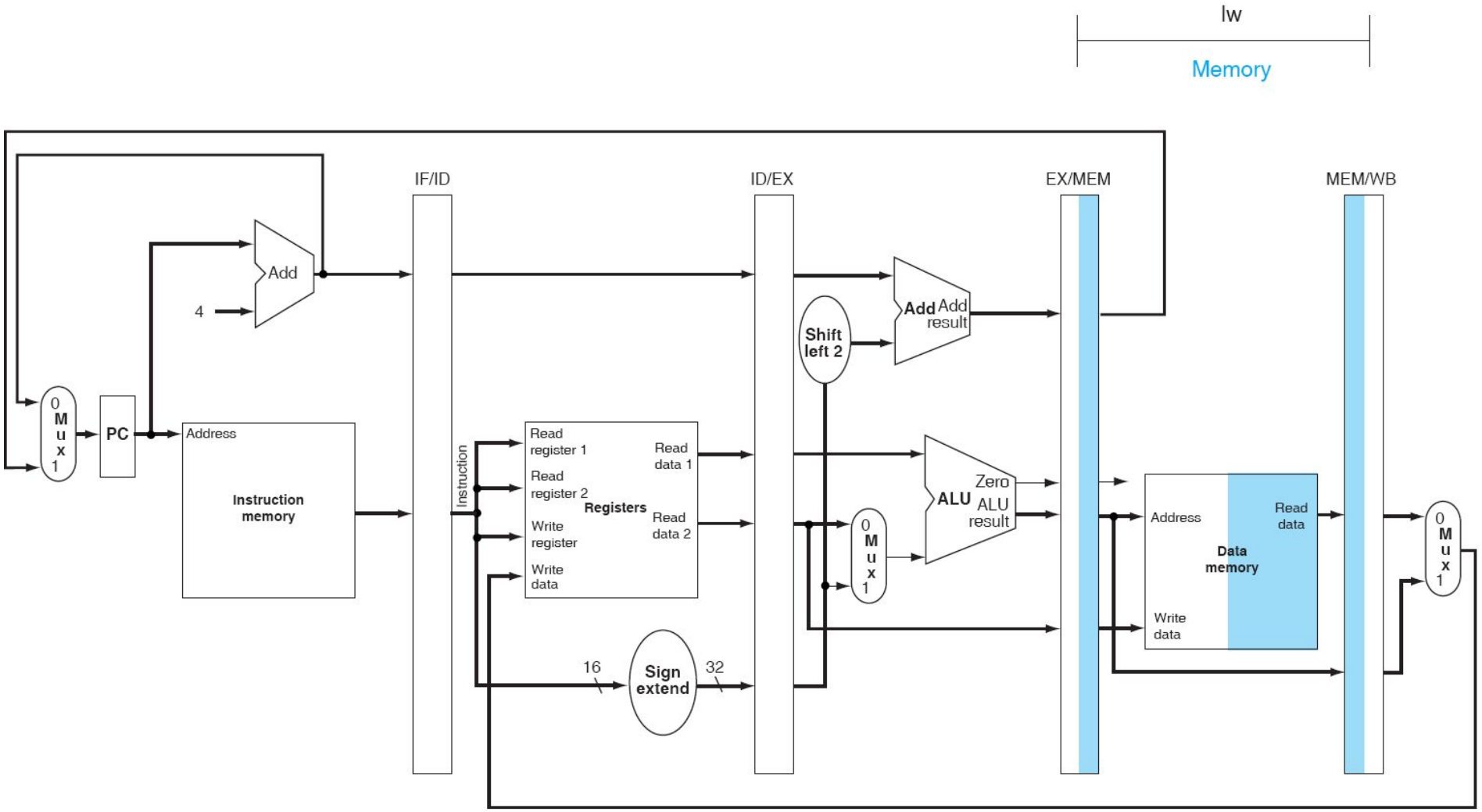
Pipeline example: lw
ID



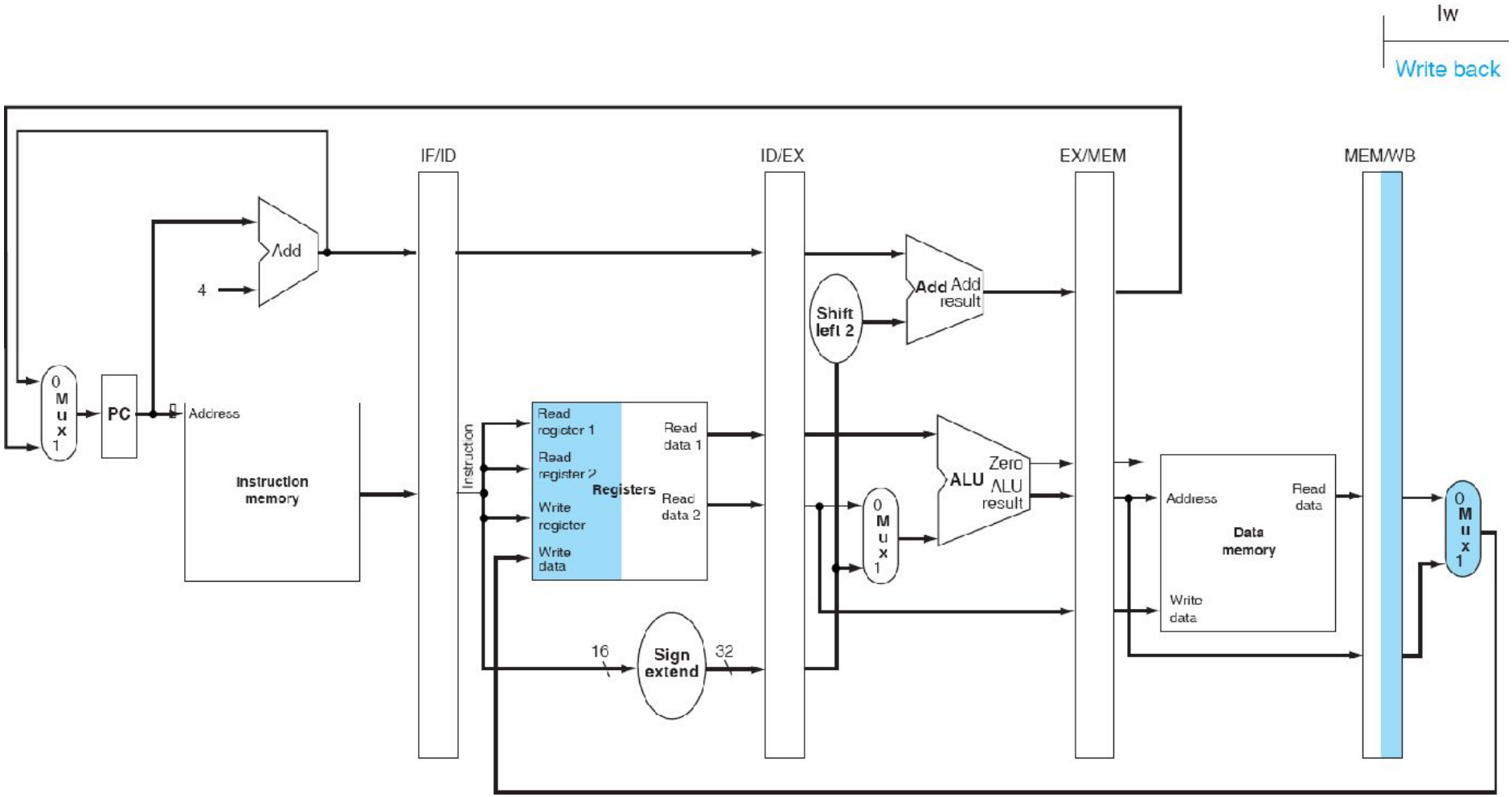
Pipeline example: lw
EX (Address Calculation)



Pipeline example: lw
MEM

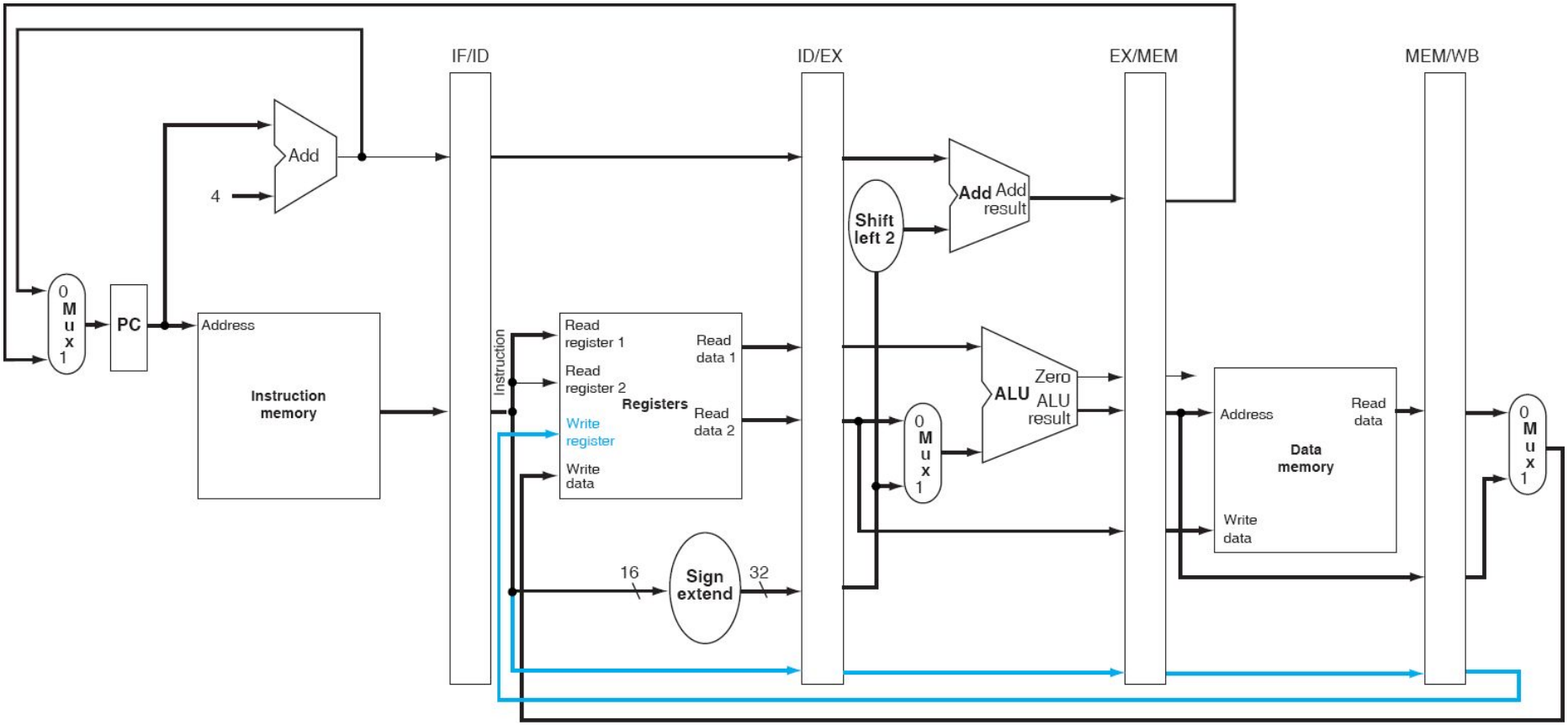


Pipeline example: lw
WB



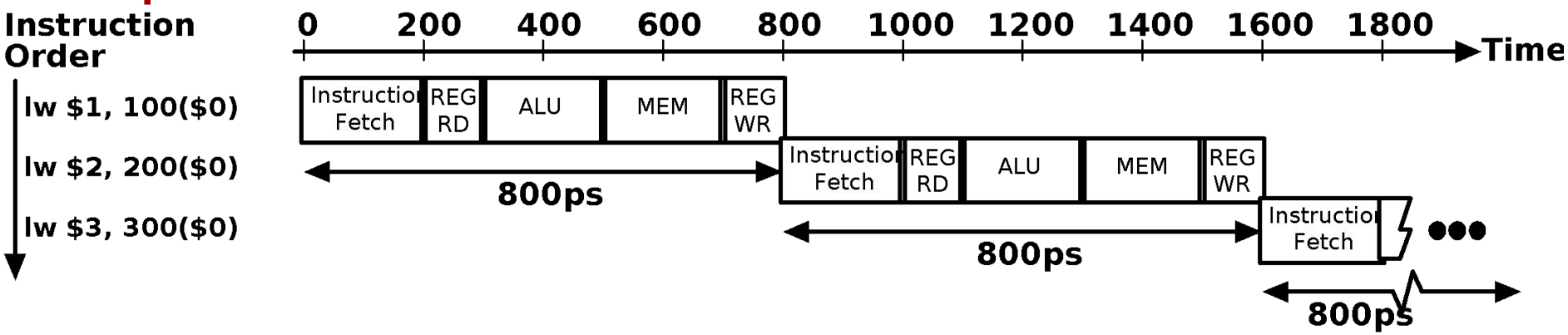
Can you find a problem?

Basic Pipelined Processor (Corrected)

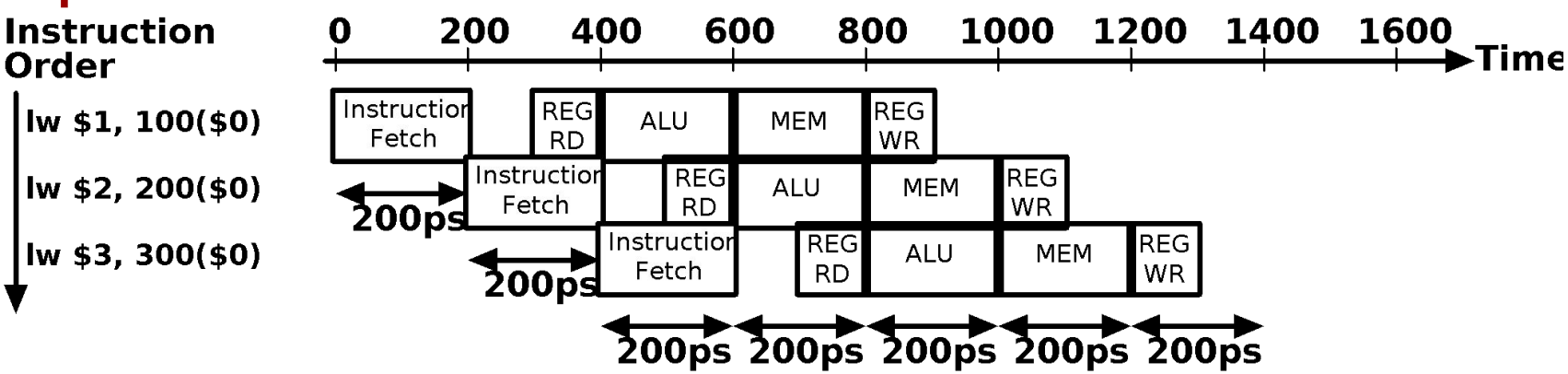


Single-Cycle vs. Pipelined Execution

Non-Pipelined



Pipelined



Speedup

- Consider the unpipelined processor introduced previously. Assume that it has a 1 ns clock cycle and it uses 4 cycles for ALU operations and branches, and 5 cycles for memory operations, assume that the relative frequencies of these operations are 40%, 20%, and 40%, respectively. Suppose that due to clock skew and setup, pipelining the processor adds 0.2ns of overhead to the clock. Ignoring any latency impact, how much speedup in the instruction execution rate will we gain from a pipeline?

Average instruction execution time

= 1 ns * ((40% + 20%)*4 + 40%*5)

= 4.4ns

Speedup from pipeline

= Average instruction time unpiplined/Average instruction time pipelined

= 4.4ns/1.2ns = 3.7

Comments about Pipelining

- The good news
  - Multiple instructions are being processed at same time
  - This works because stages are isolated by registers
  - Best case speedup of N
- The bad news
  - Instructions interfere with each other - hazards
    - » Example: different instructions may need the same piece of hardware (e.g., memory) in same clock cycle
    - » Example: instruction may require a result produced by an earlier instruction that is not yet complete

Pipeline Hazards

- **Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle**
  - **Structural hazards: two different instructions use same h/w in same cycle**
  - **Data hazards: Instruction depends on result of prior instruction still in the pipeline**
  - **Control hazards: Pipelining of branches & other instructions that change the PC**

Summary - Pipelining Overview

- Pipelining increase <u>throughput</u> (but not latency)
- Hazards limit performance
  - Structural hazards
  - Control hazards
  - Data hazards

Pipelining Outline

- Introduction
  - Defining Pipelining
  - Pipelining Instructions
- Hazards
  - Structural hazards
  - Data Hazards
  - Control Hazards
- Performance
- Controller implementation