Pipeline

A.1 What is Pipelining?

A.2 The Major Hurdle of Pipelining-Structural Hazards

- Data Hazards
- Control Hazards

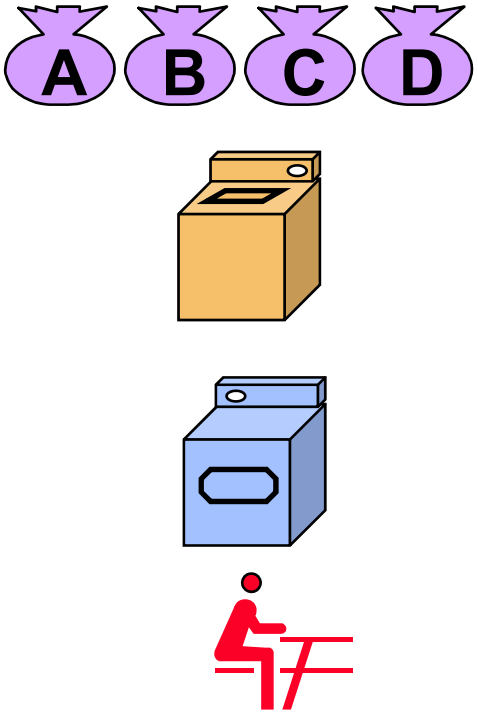
A.3 How is Pipelining Implemented

A.4 What Makes Pipelining Hard to Implement?

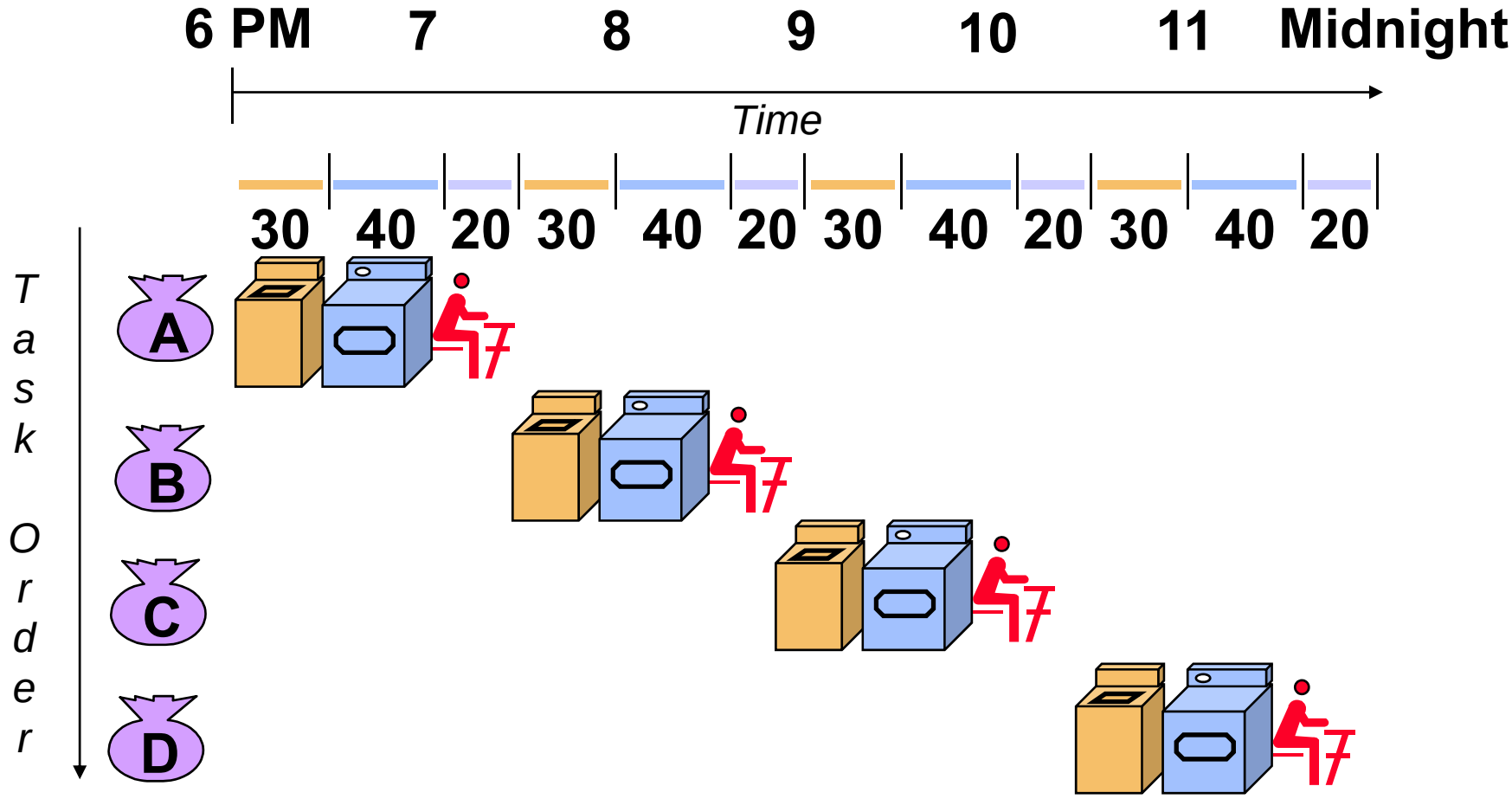
A.5 Extending the MIPS Pipeline to Handle Multi-cycle Operations

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

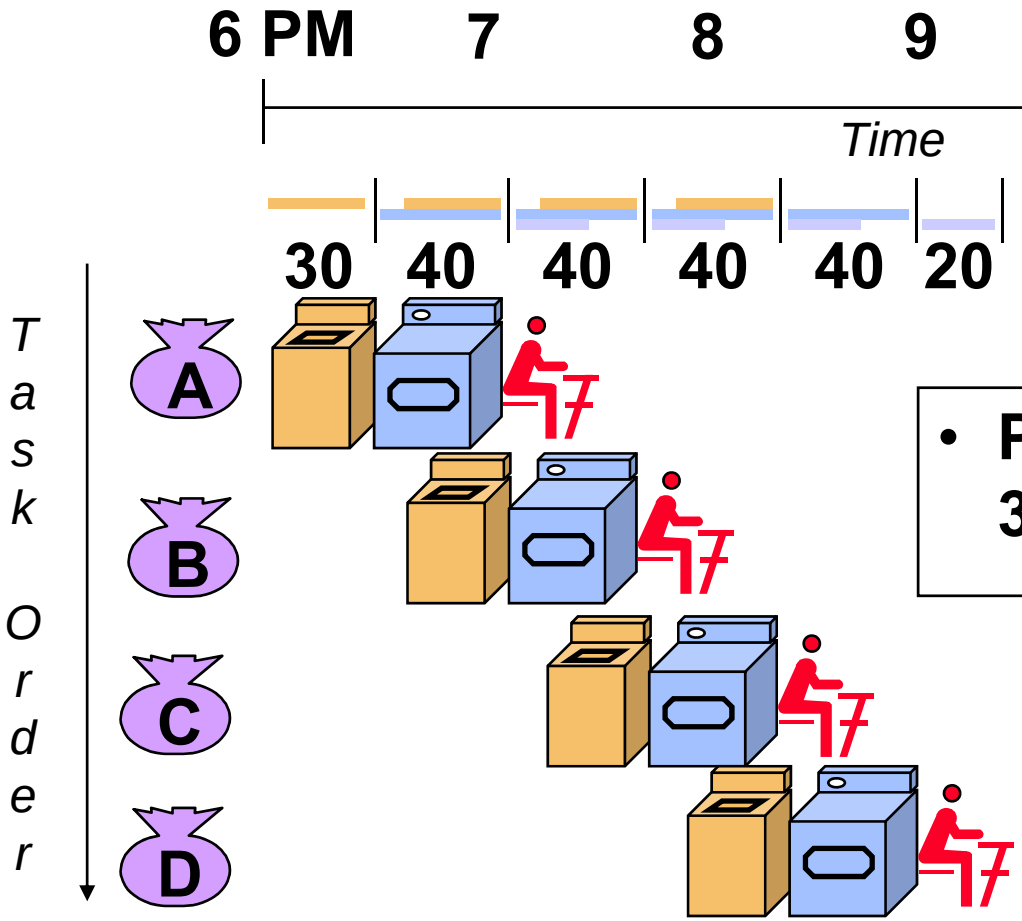


Sequential laundry takes 6 hours for 4 loads

If they learned pipelining, how long would laundry take?

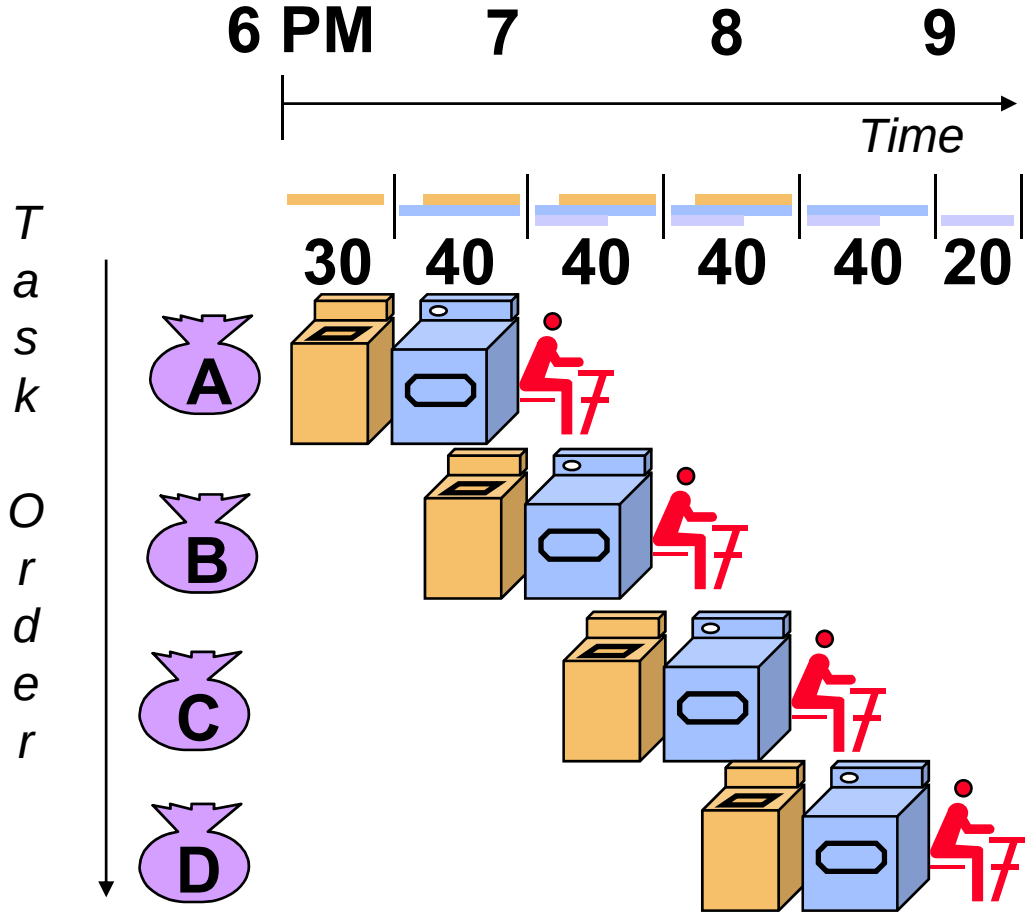
What Is Pipelining

Start work ASAP!



- Pipelined laundry takes 3.5 hours for 4 loads

What Is Pipelining

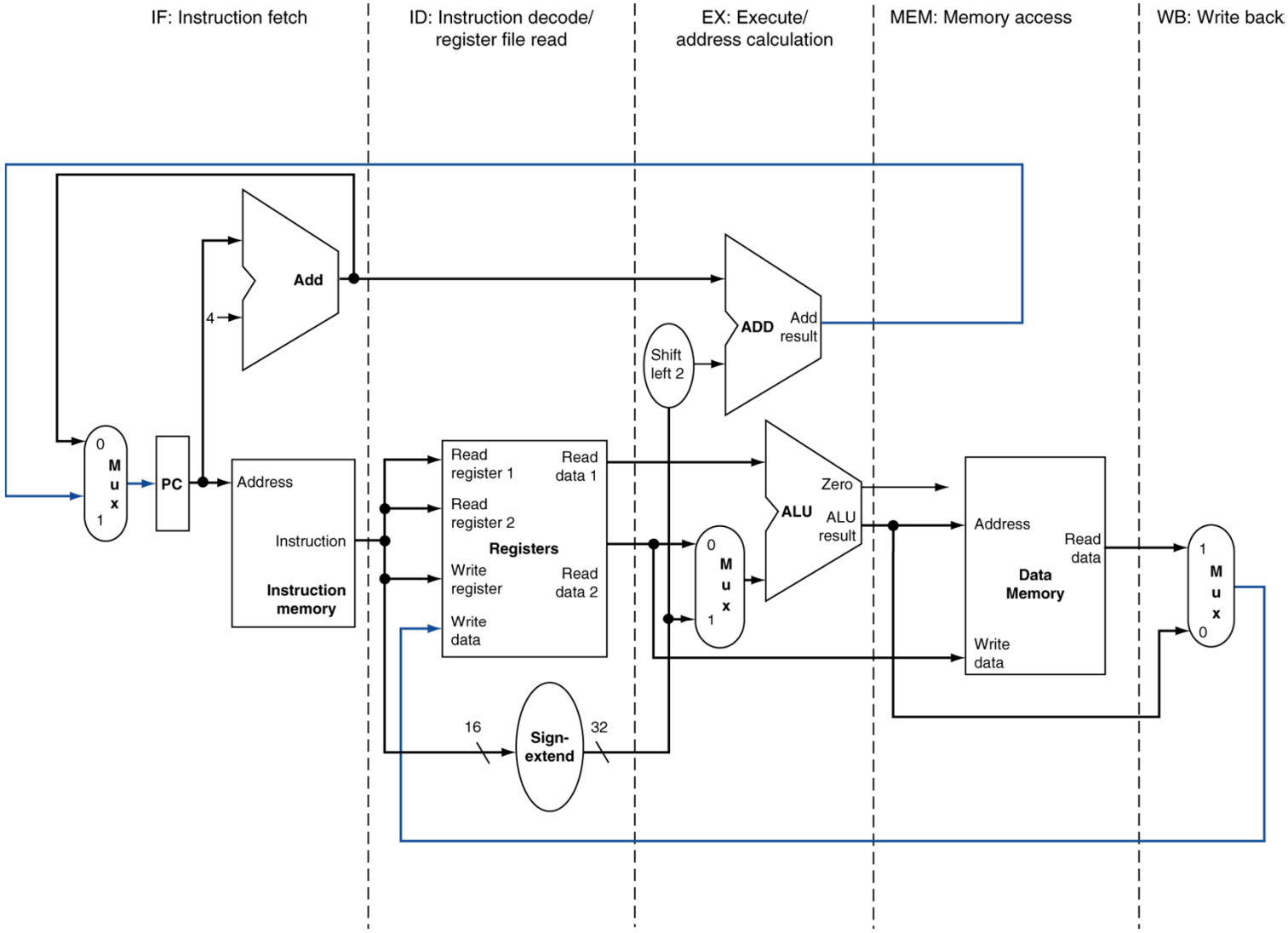


Pipelining Lessons

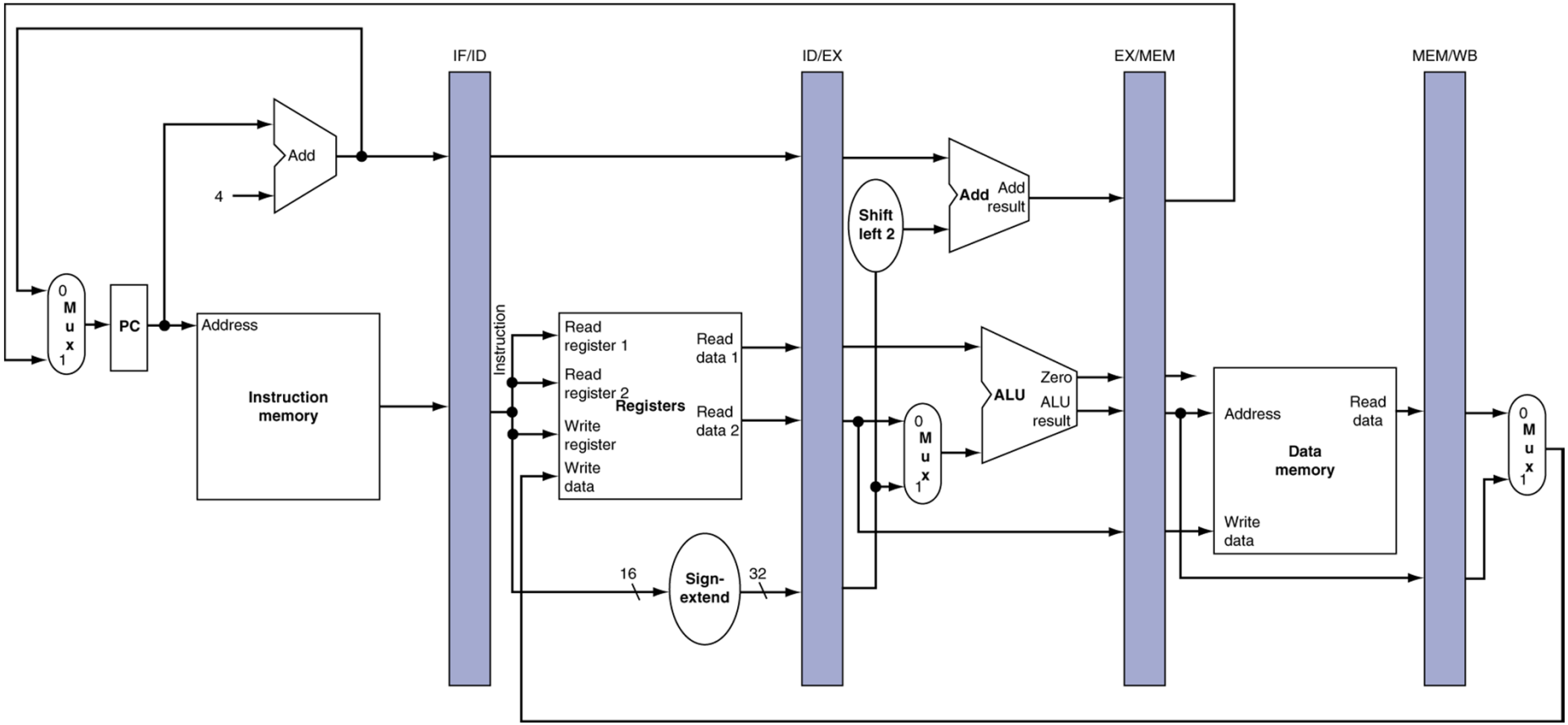
- **Pipelining doesn't help latency of single task, it helps throughput of entire workload**
- **Pipeline rate limited by slowest pipeline stage**
- **Multiple** tasks operating simultaneously
- Potential speedup = **Number pipe stages**
- Unbalanced lengths of pipe stages reduces speedup

What Is Pipelining

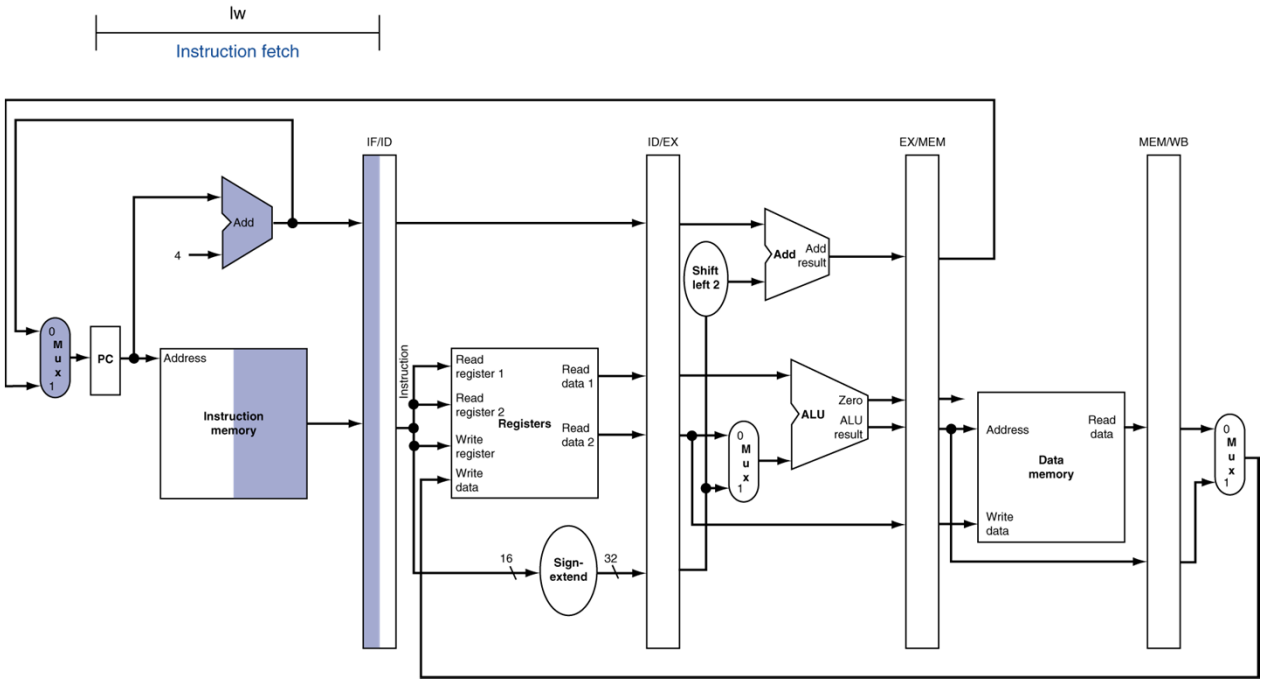
MIPS Without Pipelining



The Basic Pipeline For MIPS



Pipeline stage #1 Instruction Fetch



Passed To Next Stage

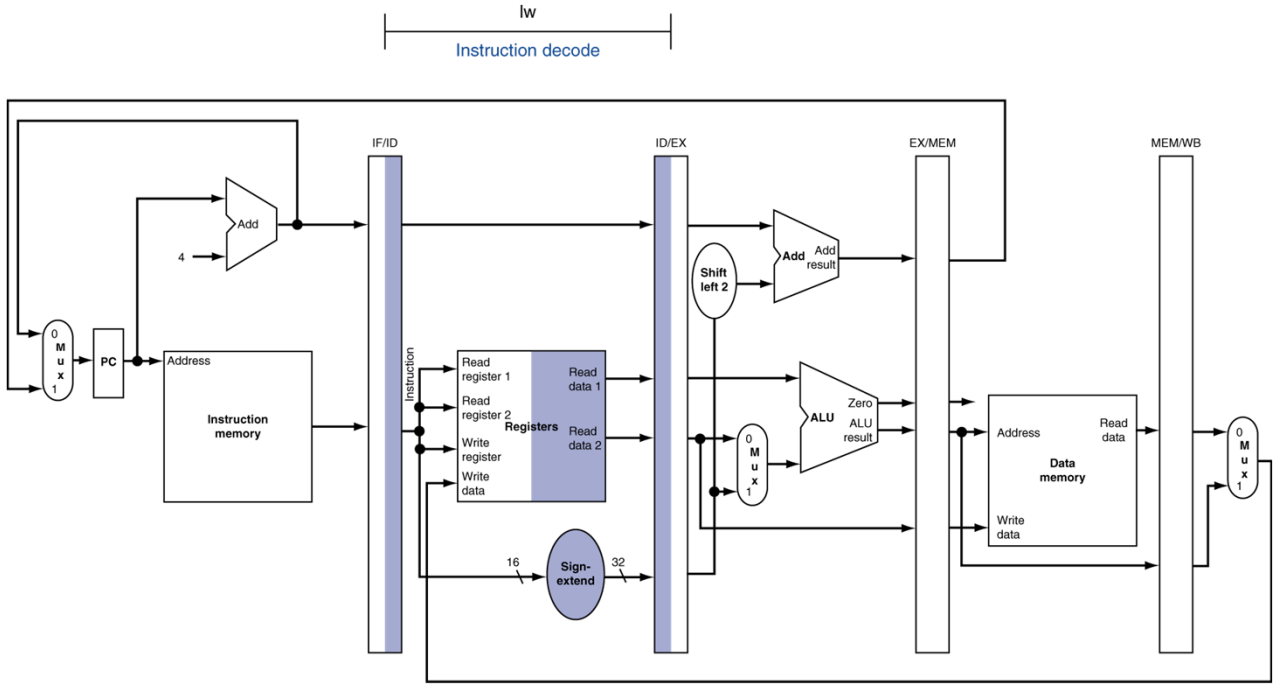
IR <- Mem[PC]

NextPC <- PC + 4

#1 Instruction Fetch (IF stage) :

1. **Send out the PC and fetch the instruction from memory into the instruction register (IR); increment the PC by 4 to address the next sequential instruction.**
2. **IR holds the instruction that will be used in the next stage.**
3. **Next PC holds the value of the next PC.**

Pipeline stage #2 Instruction Decode/Register Fetch



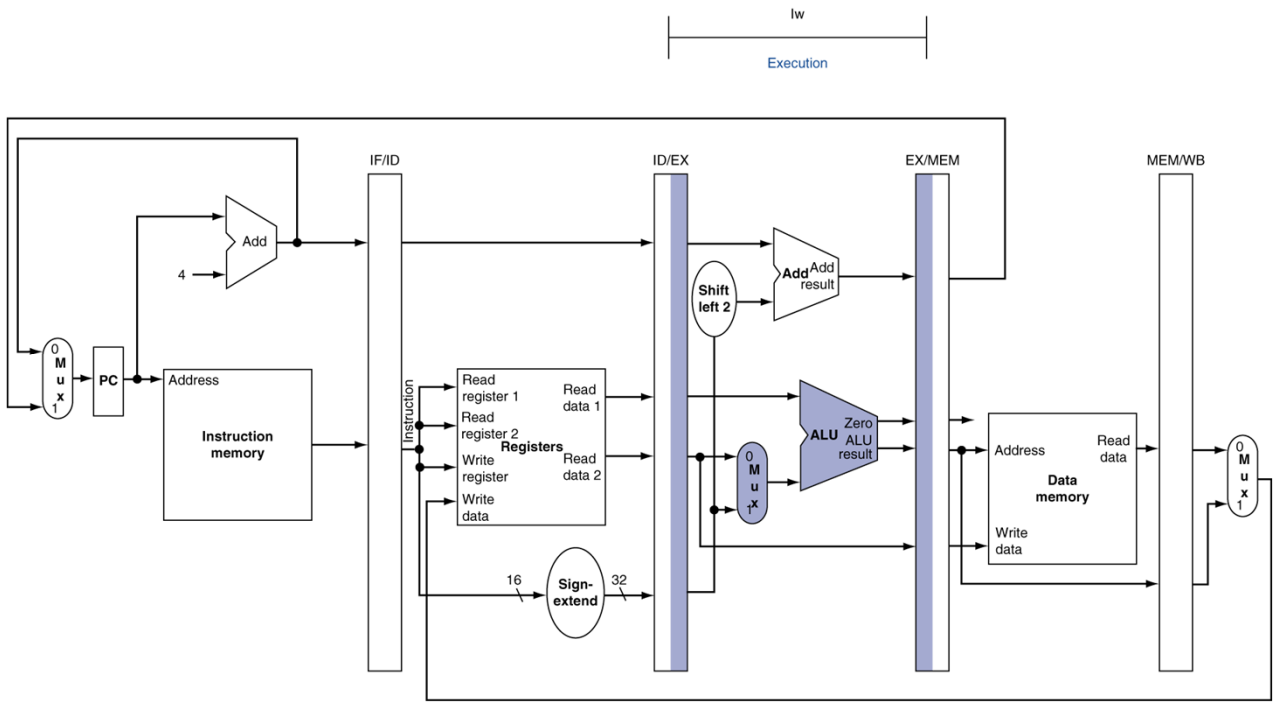
Passed To Next Stage

A <- Regs[IR25..IR21];
B <- Regs[IR20..IR16 or IR15..IR11];
Imm <- ((IR16) ##IR16-31

#2 Instruction Decode/Register Fetch Cycle (ID stage):

1. Decode the instruction and access the register file to read the register values.
2. The outputs of the general purpose registers are read into two temporary registers (A & B) for use in later clock cycles.
3. We extend the sign of the lower 16 bits of the Instruction Register.

Pipeline stage #3 ALU Execution



Passed To Next Stage

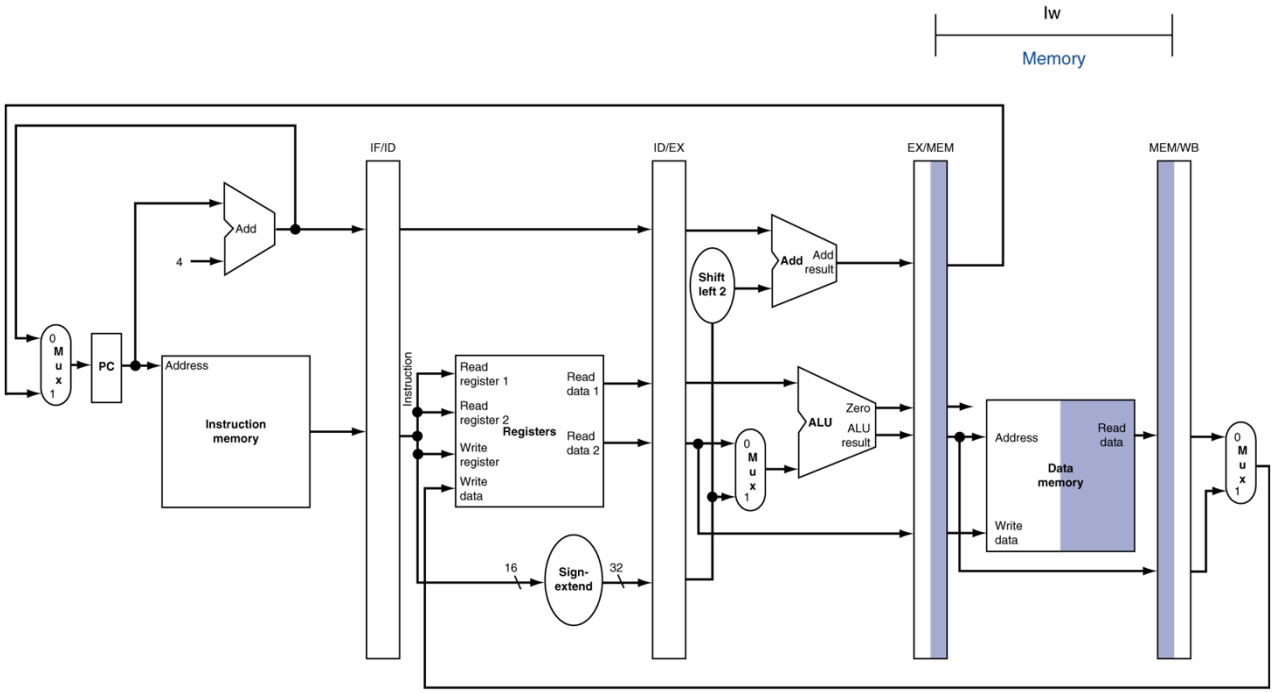
A <- ALU operation with prev. A and prev. B register values

B <- prev. B

#3 ALU Execution (EX stage):

1. **We perform an operation (for an ALU) or an address calculation (if it's a load or a Branch).**
2. **If an ALU, actually do the operation. If an address calculation, figure out how to obtain the address and stash away the location of that address for the next cycle.**

Pipeline stage #4 MEM ACCESS



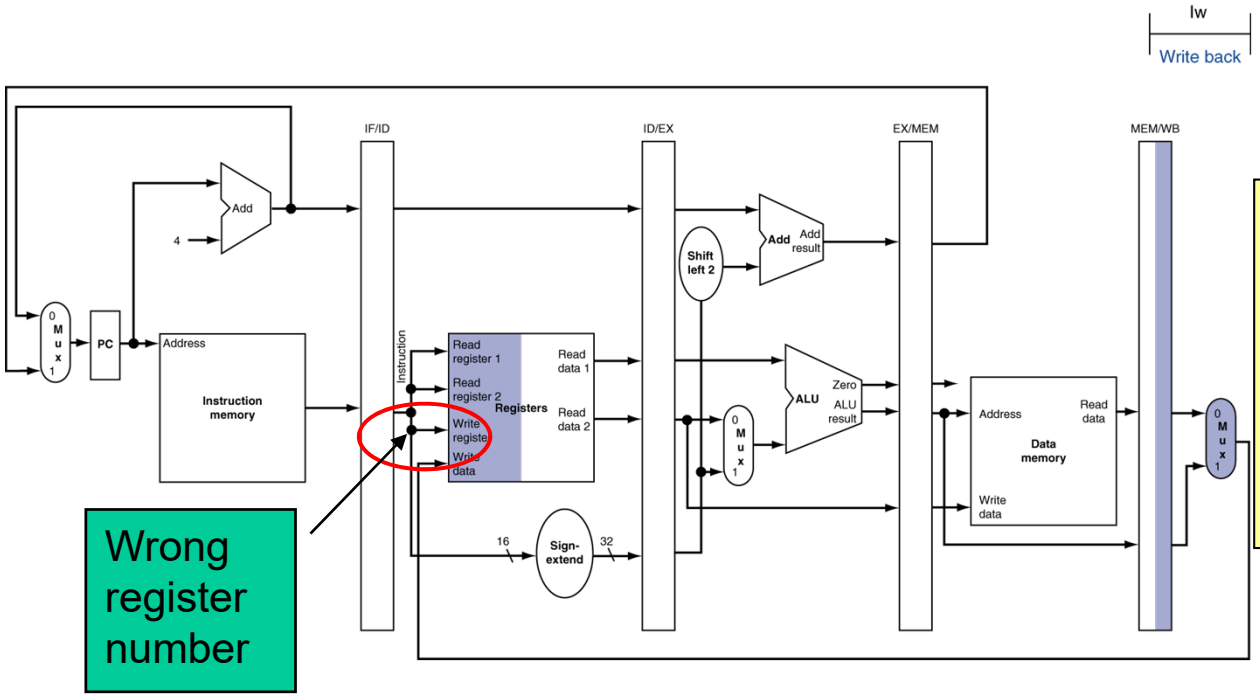
Passed To Next Stage

A = Mem[prev. A]
or
Mem[prev. A] = prev. B
B <- prev. A

#4 MEMORY ACCESS (MEM):

1. If this is an ALU, do nothing.
2. If a load or store, then access memory.

Pipeline stage #5 WRITE BACK



Wrong register number

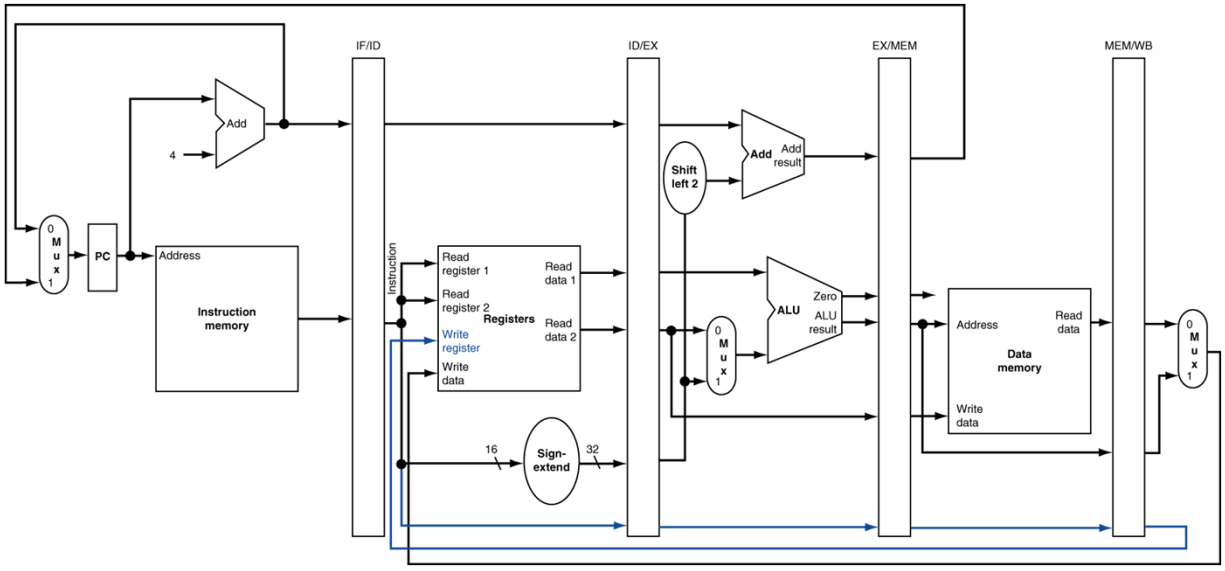
Passed To Next Stage

Regs [?] <- prev. A or prev. B;

WRITE BACK (WB):

1. **Update the registers from either the ALU or from the data loaded.**

Pipeline stage #5 WRITE BACK



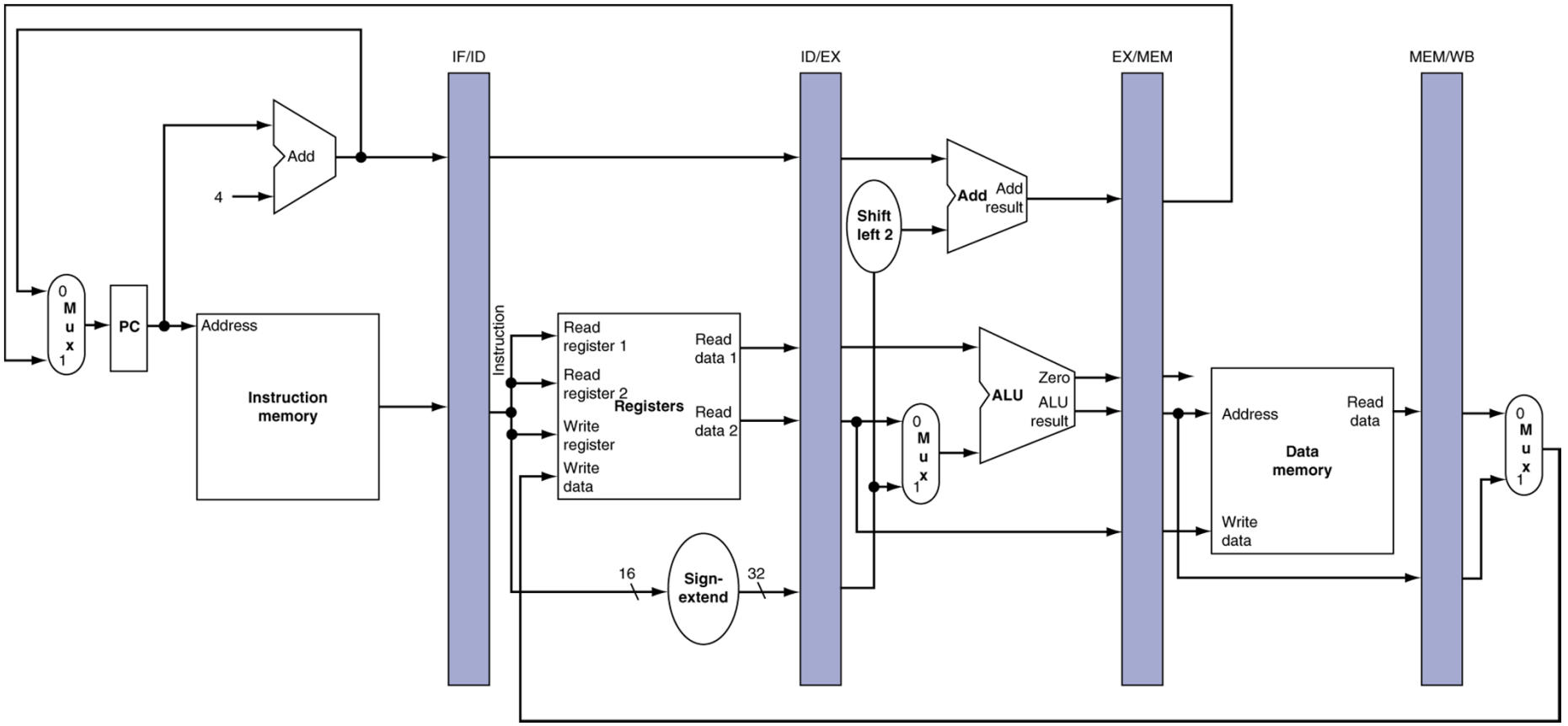
Passed To Next Stage

Regs [prev. C] <- prev. A or prev. B;

WRITE BACK (WB):

1. **Update the registers from either the ALU or from the data loaded.**

The Basic Pipeline For MIPS



Pipeline buffers (registers) between two stages provide data each next stage for pipelining operation .

The Basic Pipeline For MIPS

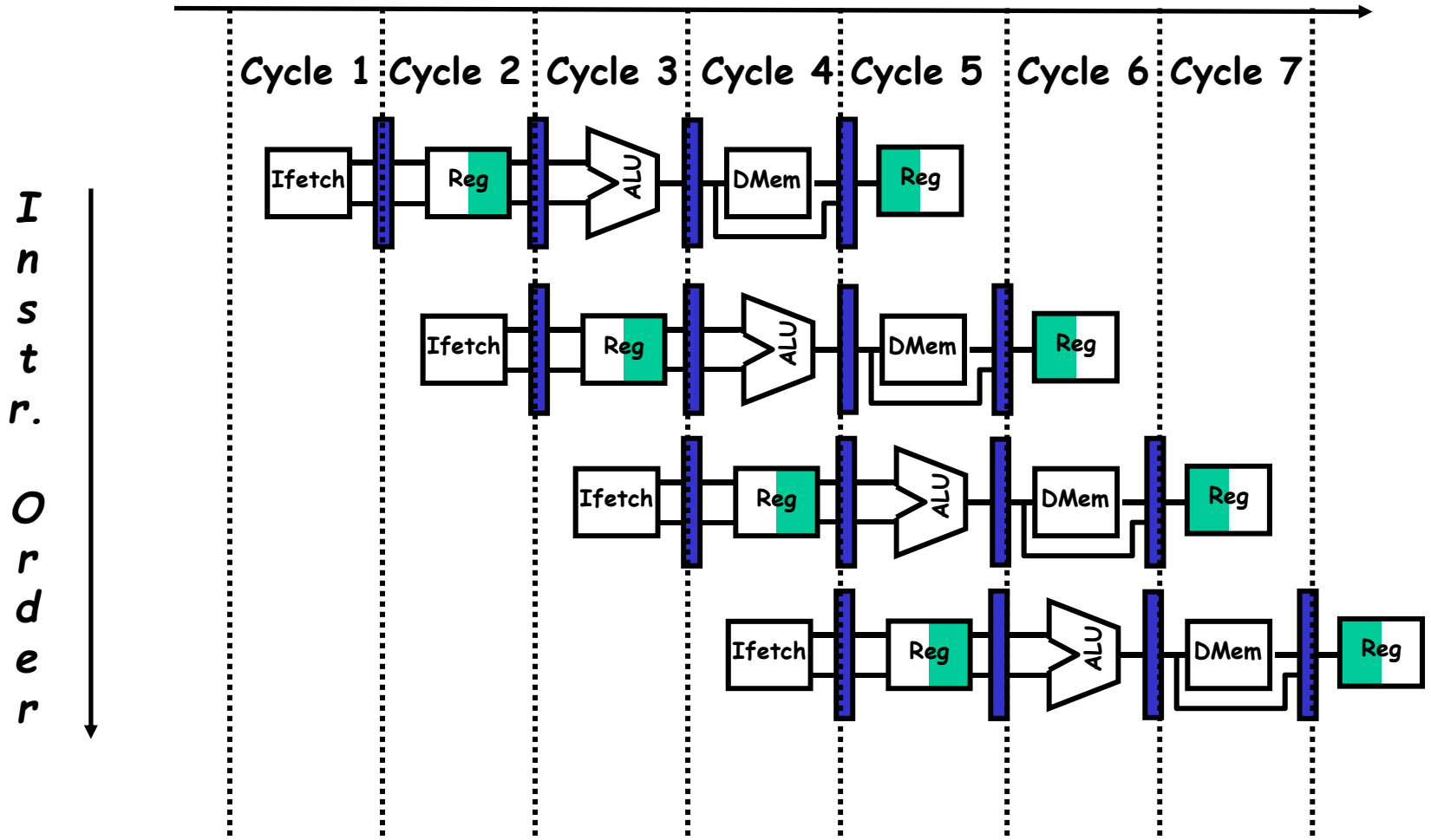


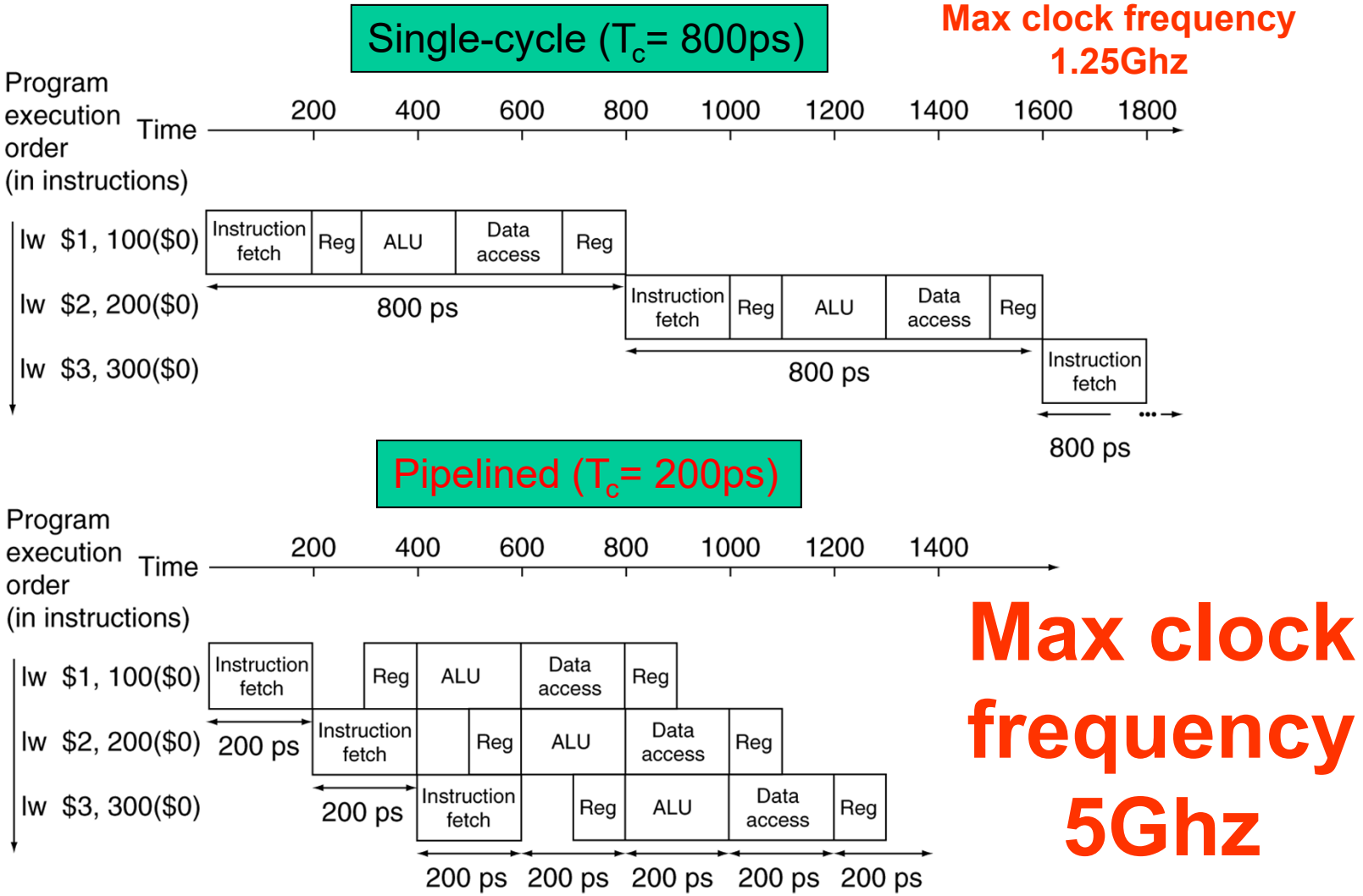
Figure 3.3

Pipeline Performance

- **Assume time for stages is**
  - 100ps for register read or write
  - 200ps for other stages
- **Compare pipelined datapath with single-cycle datapath**

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

Pipeline Performance



- Clock frequency limited by a combinational logic path with the longest delay time