0x0E - Pipelining

ENGR 3410: Computer Architecture

Jon Tse

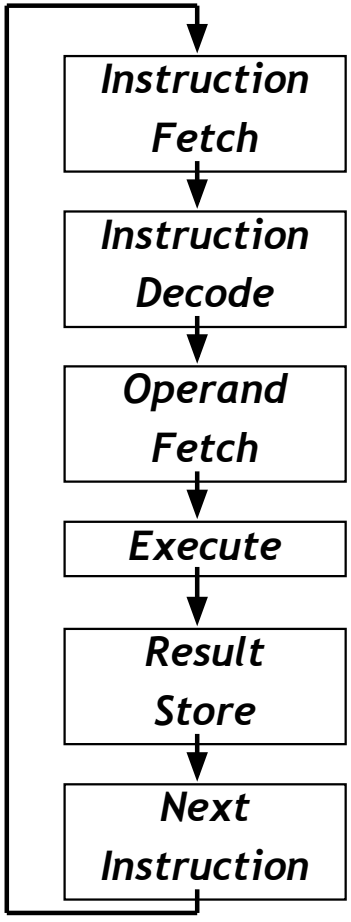
Fall 2020

Today

- Introduction to Pipelining

- Glance at Hazards

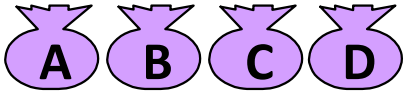
CPU Execution Overview



- Fetch instruction from memory
- Decode instruction into actions/controls
- Fetch/Decode operands
- Compute result value or status
- Push result(s) to storage
- Determine next instruction

Pipelining

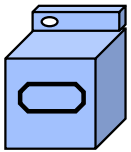
Example: Doing the laundry



Ann, Brian, Cathy, & Dave
each have one load of clothes to wash, dry, and fold

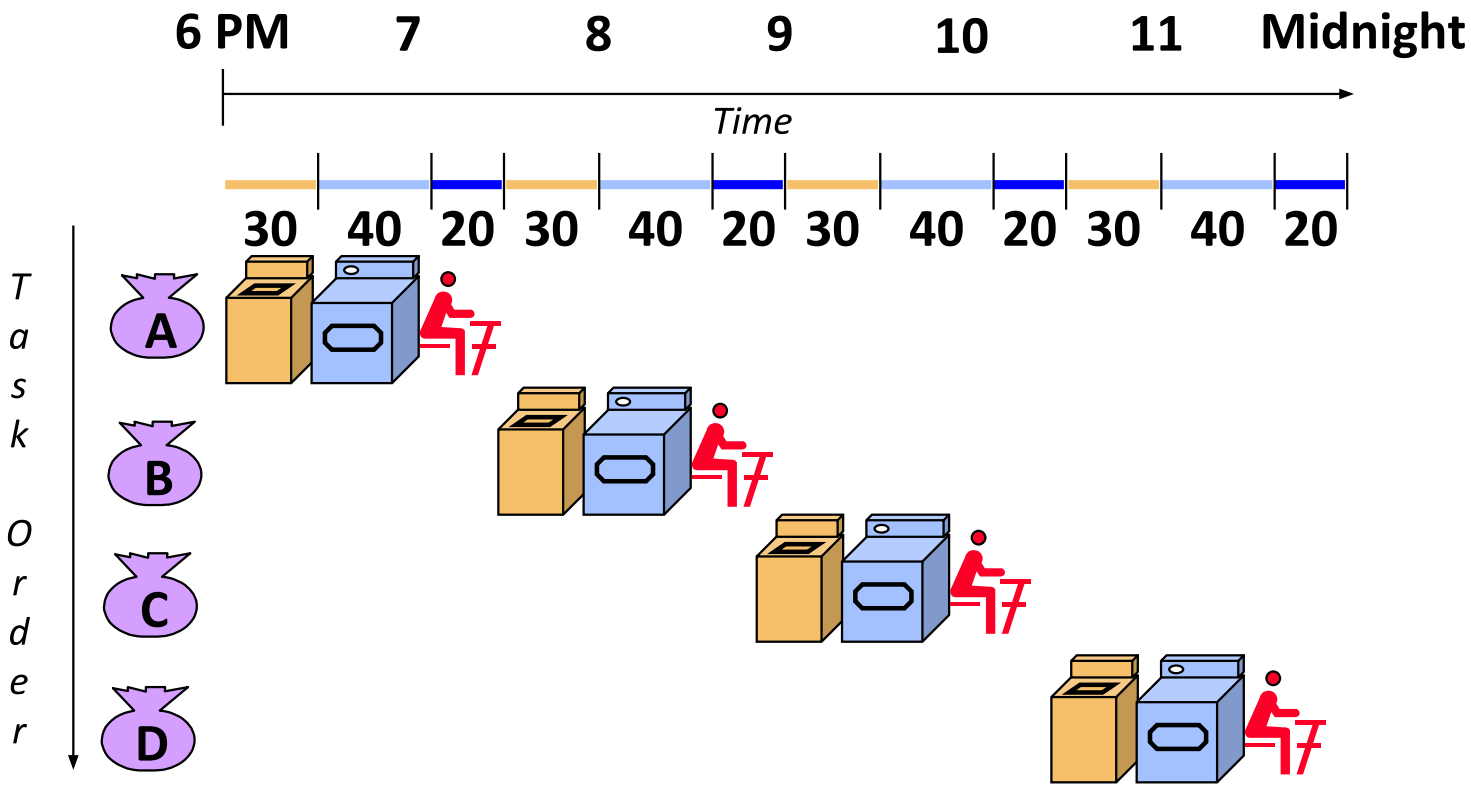
Washer takes 30 minutes

Dryer takes 40 minutes



"Folder" takes 20 minutes

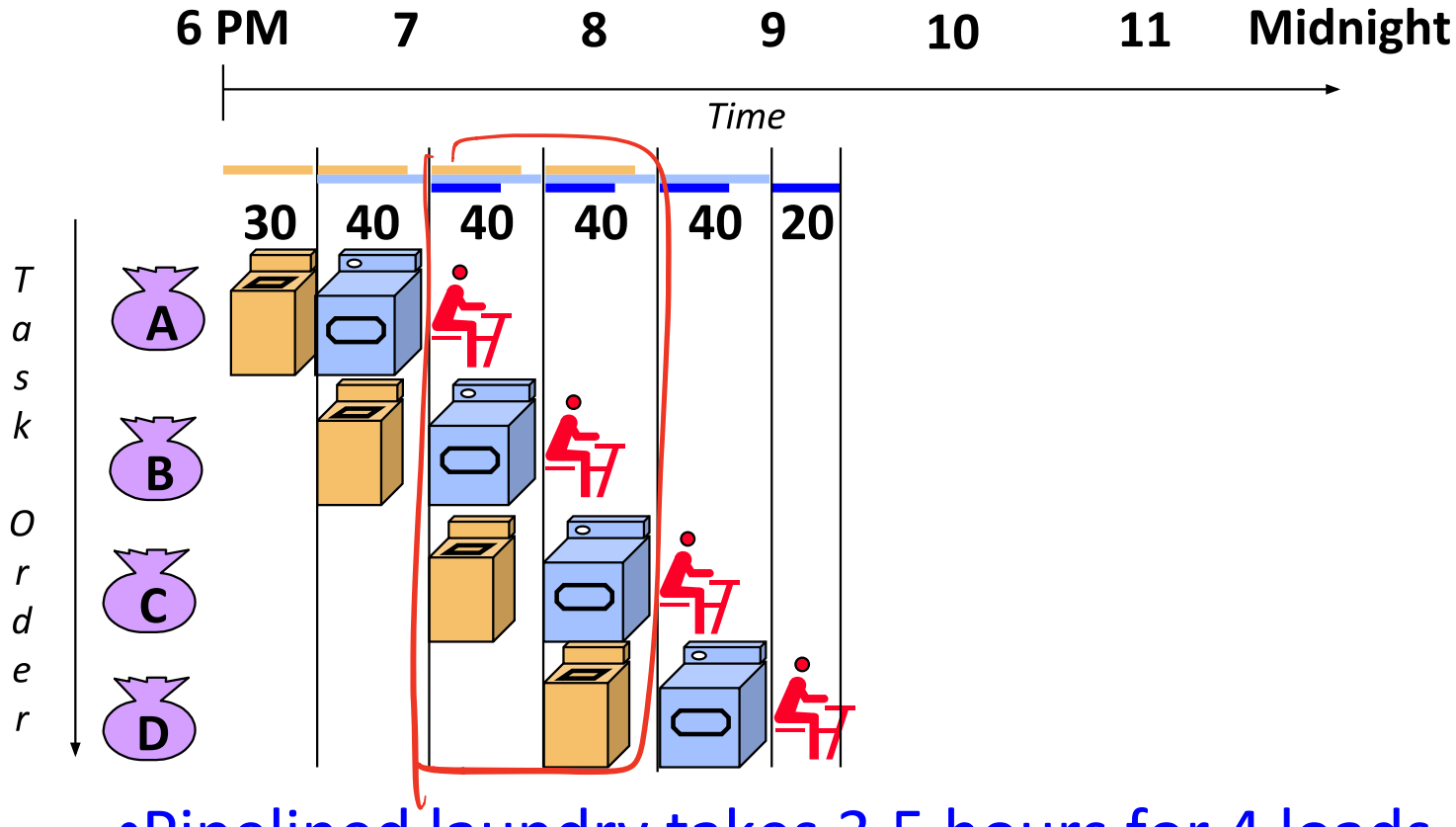
Sequential Laundry



Sequential laundry takes 6 hours for 4 loads

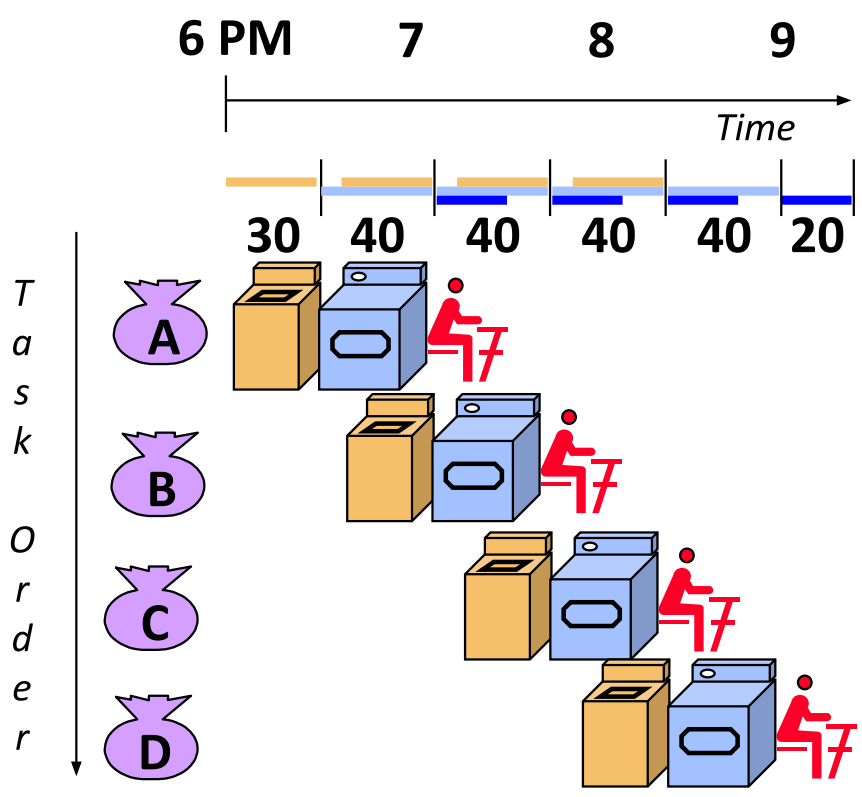
If they learned pipelining, how long would laundry take?

Pipelined Laundry: Start work ASAP



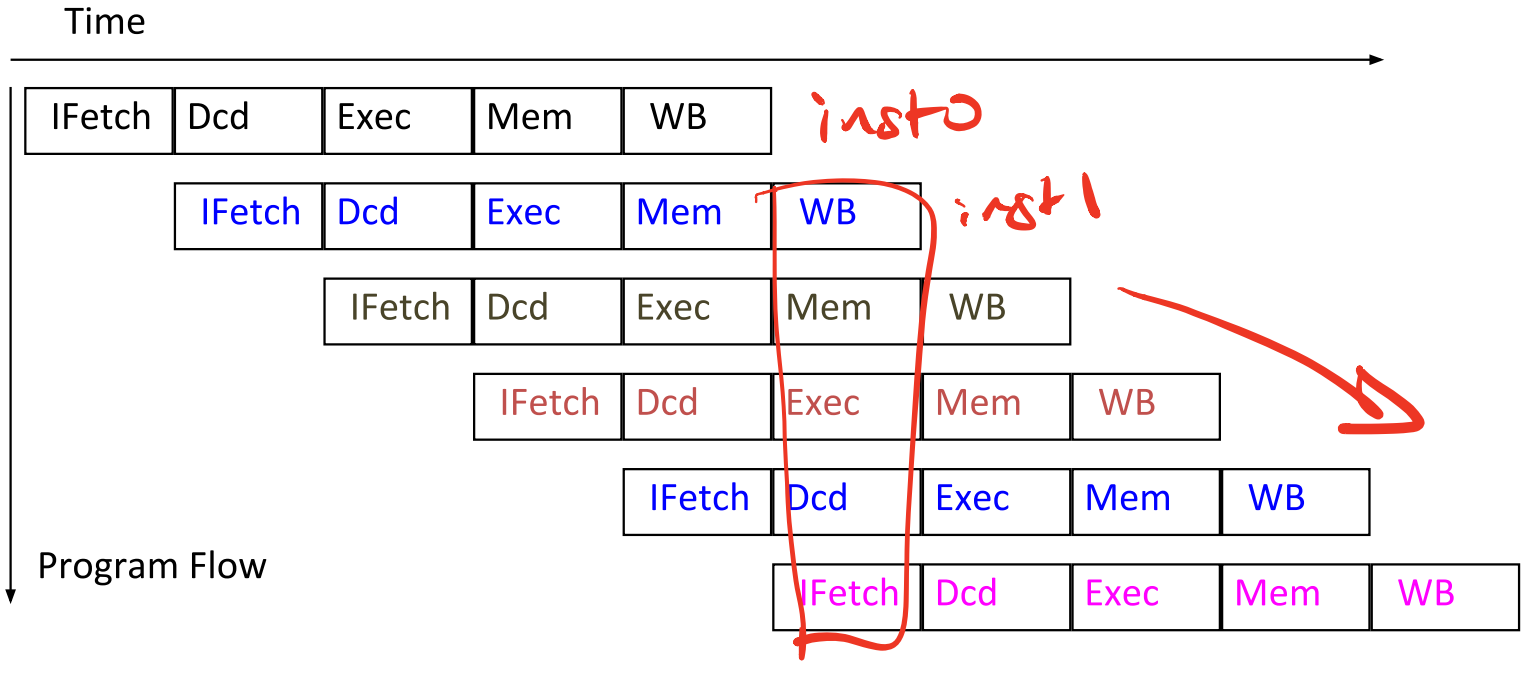
- Pipelined laundry takes 3.5 hours for 4 loads

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 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

Pipelined Execution



- Ignore fill/drain, ignore latency
- Pipelining maximizes **throughput**

Why Pipeline?

IF ID EX MEM WB

50 ns

Suppose we execute 100 instructions

Single Cycle Machine

- 50 ns/cycle x 1 CPI x 100 inst = 5000 ns

Ideal pipelined machine

- 10 ns/cycle x (1 CPI x 100 inst + 4 cycle drain) = 1040 ns

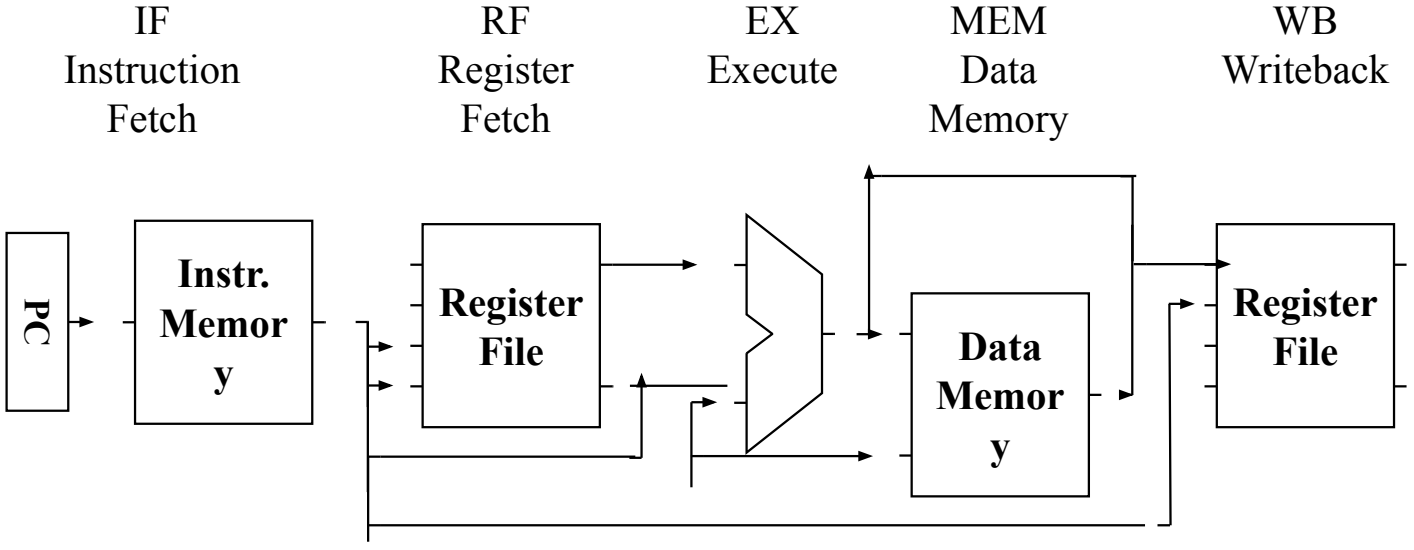
104

Rules of Pipelining

- Divide CPU / Instruction into **stages**
- Each functional unit belongs to one stage
  - Can't reuse e.g. the ALU like a multi-cycle design
  - Reg File R/W ports act as **separate** functional units
  - For now, split memory into instruction & data
- Stage order is the same for all instructions
  - How can you "skip" a stage?
  - This is true for "purely" pipelined processors

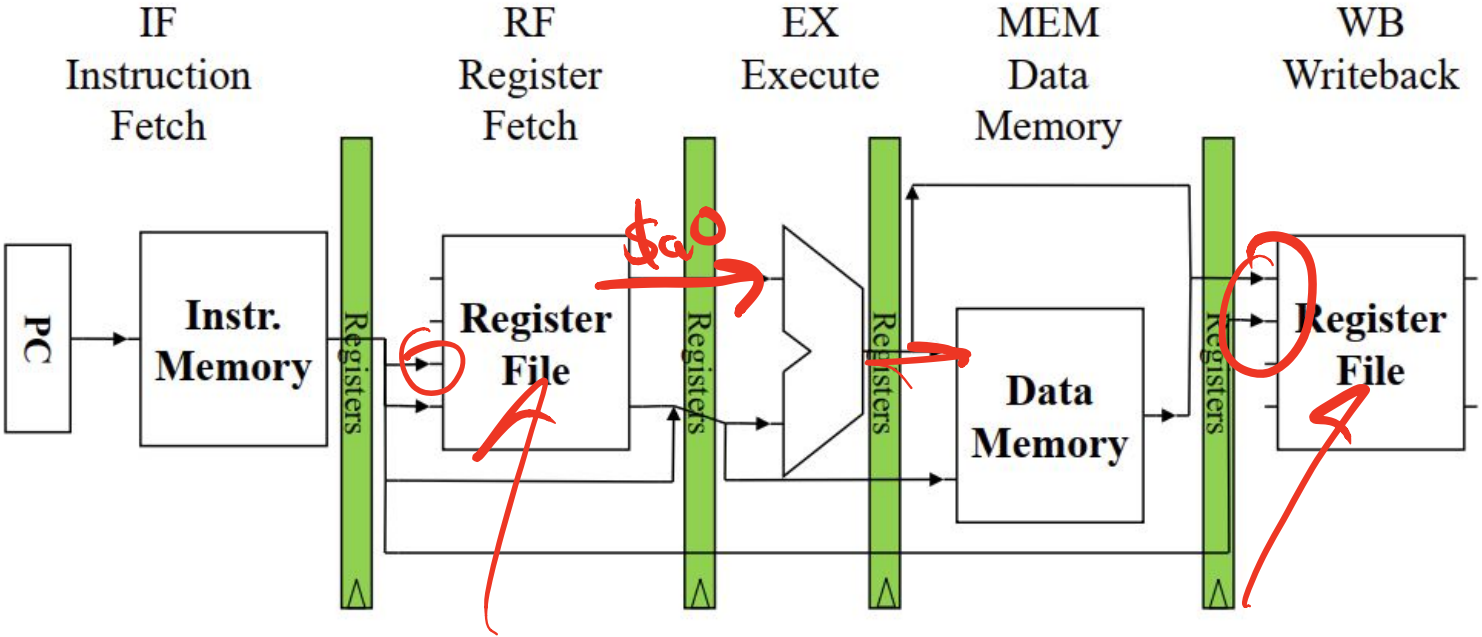
Pipelined Datapath

- Divide datapath into multiple pipeline stages



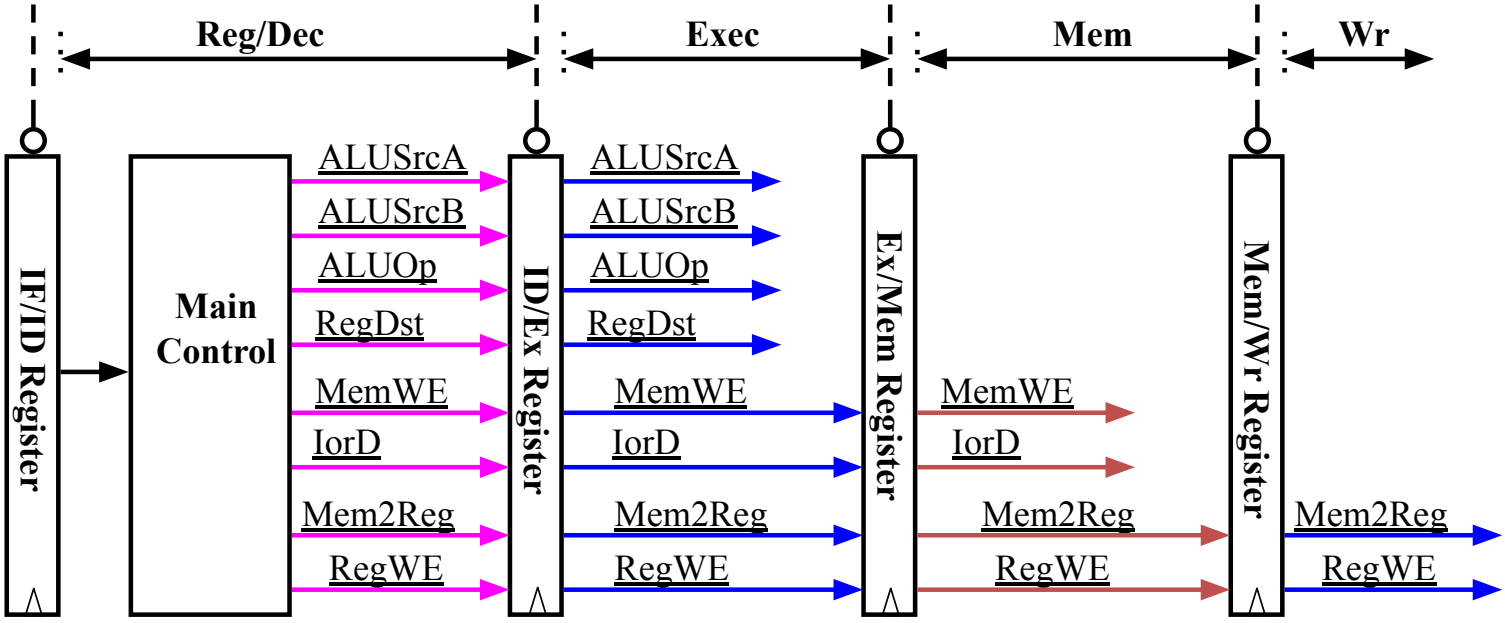
Pipelined Datapath

- Divide datapath into multiple pipeline stages



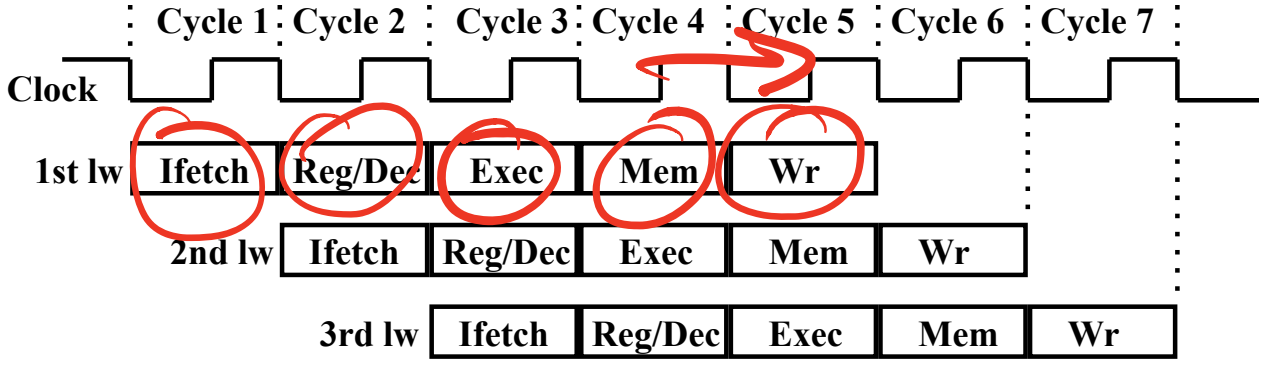
Pipelined Control

- The Main Control generates the control signals during Reg/Dec
  - Control signals for Exec (ALUOp, ALUSrcA, …) are used 1 cycle later
  - Control signals for Mem (MemWE, IorD, …) are used 2 cycles later
  - Control signals for Wr (Mem2Reg, RegWE, …) are used 3 cycles later



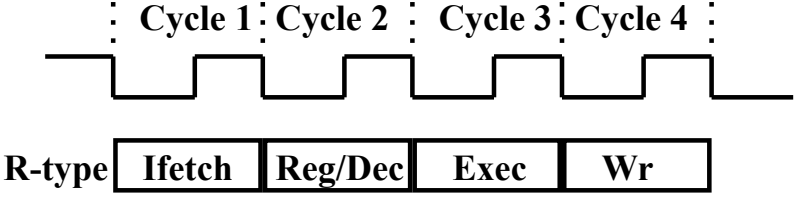
Pipelining the Load Instruction

- The five independent functional units in the pipeline datapath are:
  - Instruction Memory for the Ifetch stage
  - Register File's Read ports (bus A and busB) for the Reg/Dec stage
  - ALU for the Exec stage
  - Data Memory for the Mem stage
  - Register File's Write port (bus W) for the Wr stage



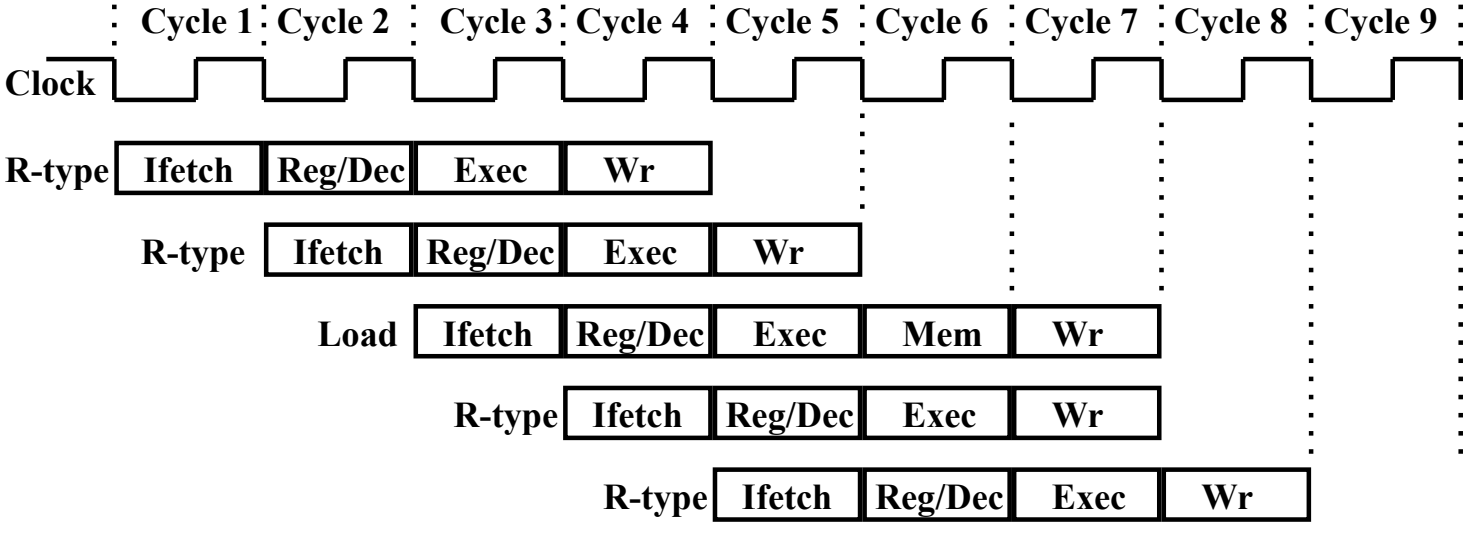
The Four Stages of R-type

- Ifetch: Fetch the instruction from the Instruction Memory
- Reg/Dec: Register Fetch and Instruction Decode
- Exec: ALU operates on the two register operands
- Wr: Write the ALU output back to the register file



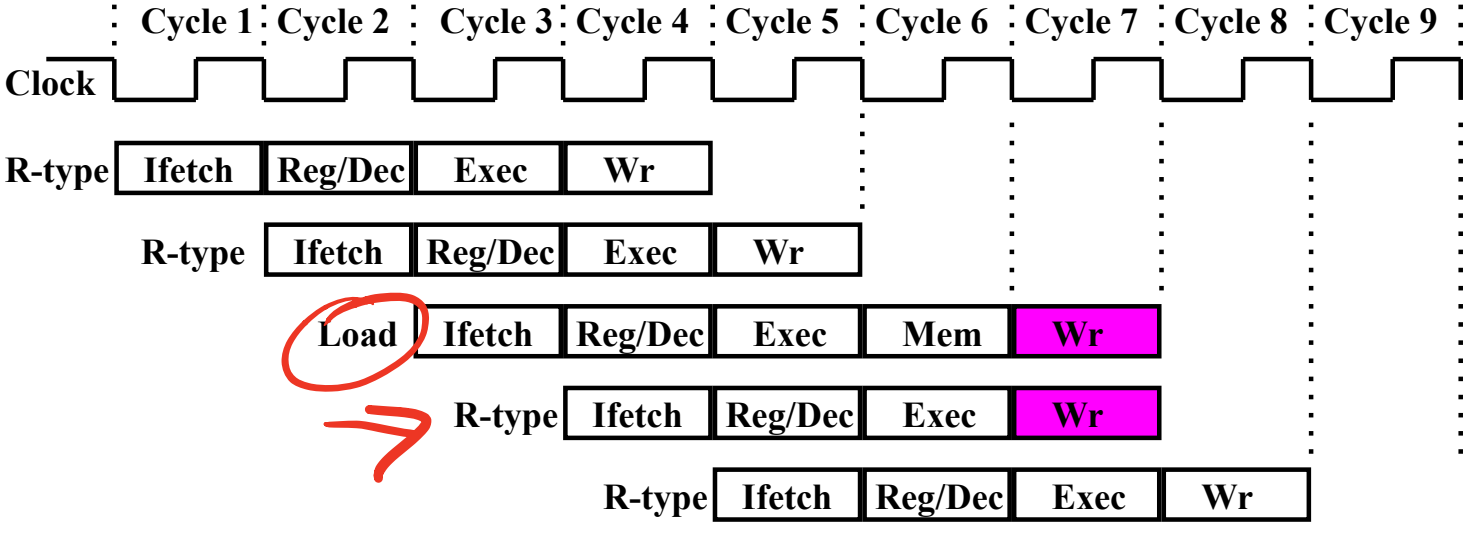
Structural Hazard

- Interaction between R-type and loads causes structural hazard on writeback



Structural Hazard

- Interaction between R-type and loads causes structural hazard on writeback



Structural Hazard: Solved

- Add a do-nothing (NO OP or NOP) MEM stage