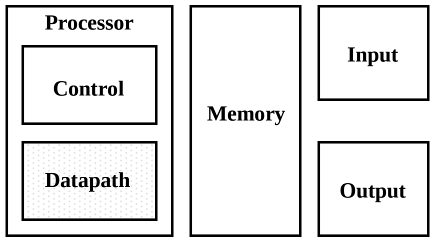
The Big Picture: Where are We Now?

- The Five Classic Components of a Computer

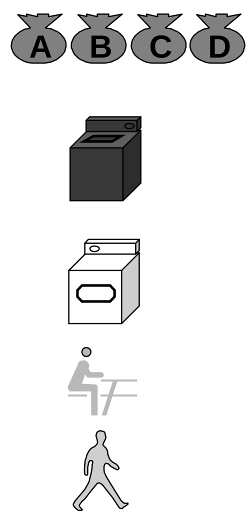


- Today's Topics:
  - Pipelining by Analogy
  - Introduction to MIPS pipelining

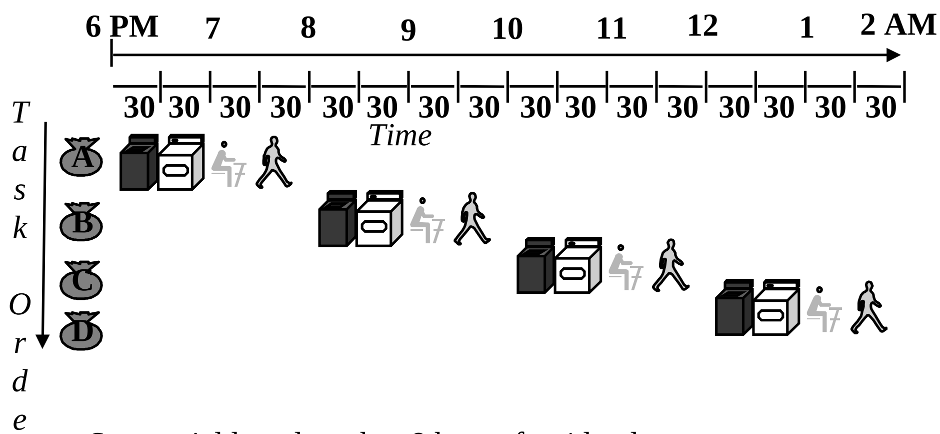
Pipelining is Natural!

Laundry Example

- 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

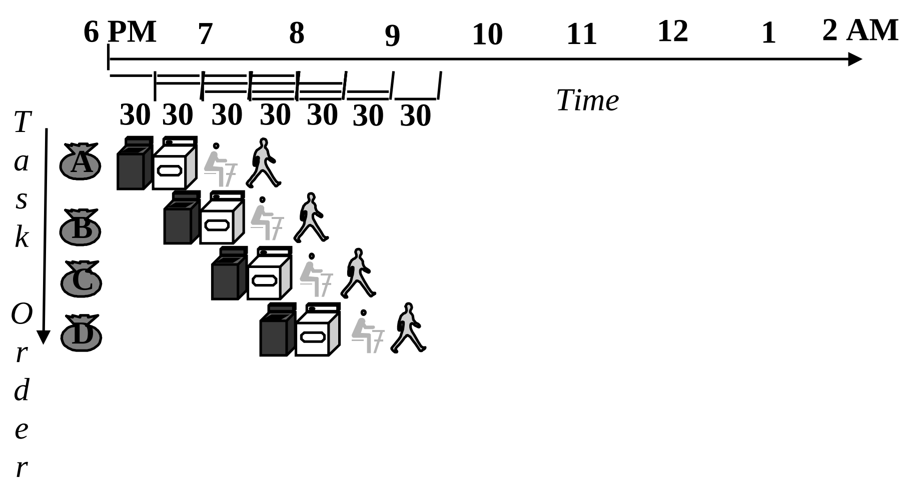


Sequential Laundry



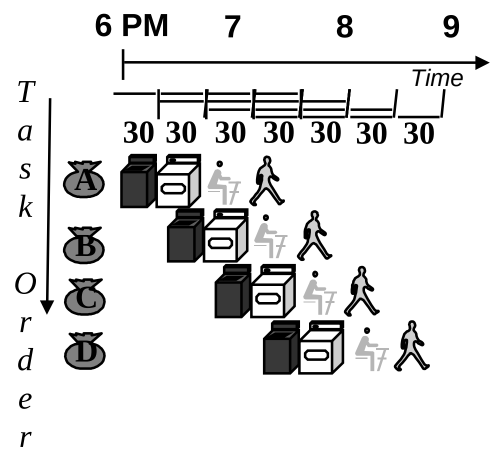
- Sequential laundry takes 8 hours for 4 loads
- If they learned pipelining, how long would laundry take?

Pipelined Laundry: Start work ASAP



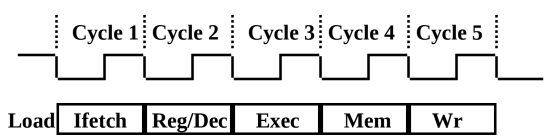
- Pipelined laundry takes 3.5 hours for 4 loads!

Pipelining Lessons



- Pipelining doesn’t help latency of single task, it helps throughput of entire workload
- Multiple tasks operating simultaneously using different resources
- Potential speedup = Number of pipe stages
- Pipeline rate limited by slowest pipeline stage
- Unbalanced lengths of pipe stages reduces speedup
- Time to “fill” pipeline and time to “drain” it reduces speedup
- Stall for Dependences

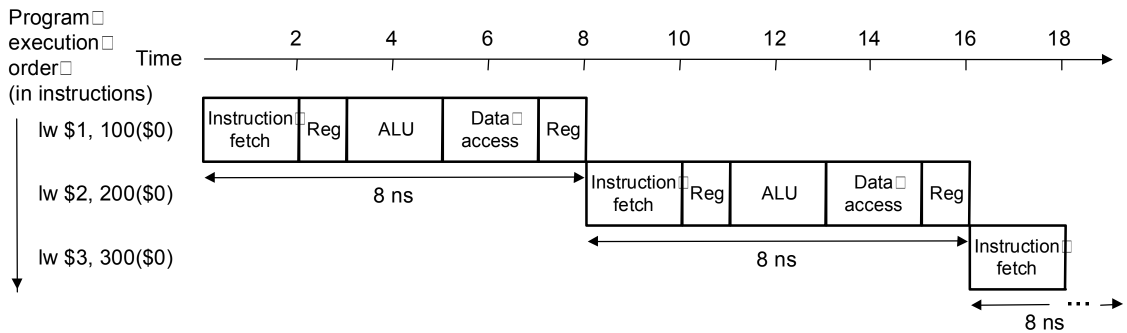
The Five Stages of Load

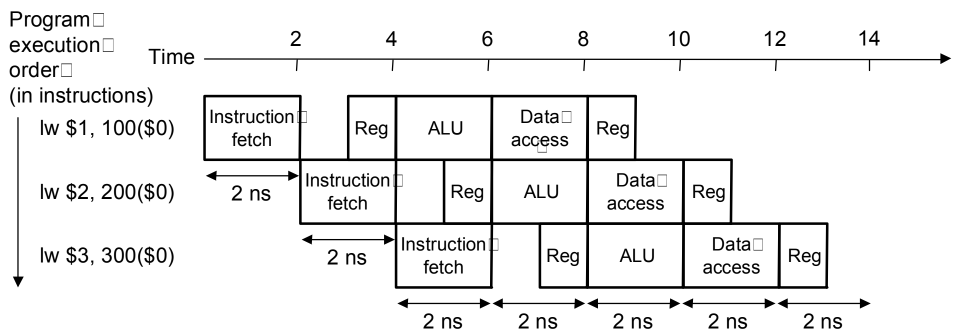


- Ifetch: Instruction Fetch
  - Fetch the instruction from the Instruction Memory
- Reg/Dec: Registers Fetch and Instruction Decode
- Exec: Calculate the memory address
- Mem: Read the data from the Data Memory
- Wr: Write the data back to the register file

Pipelining

- **Improve perfomance by increasing instruction throughput**



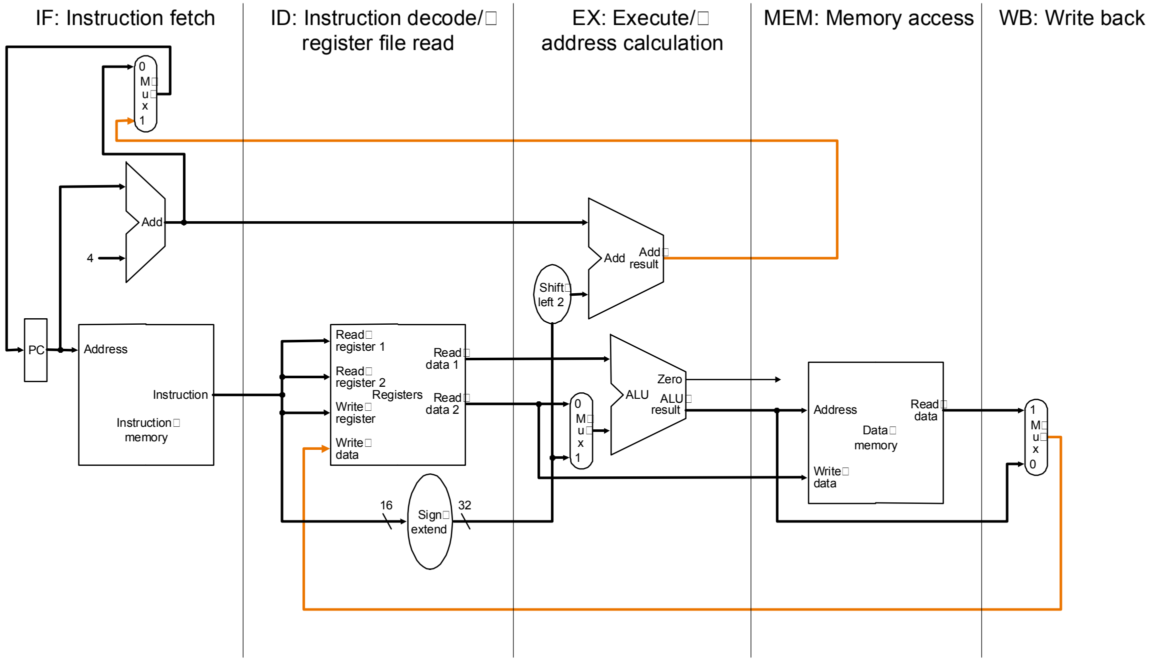


Ideal speedup is number of stages in the pipeline. Do we achieve this?

Pipelining

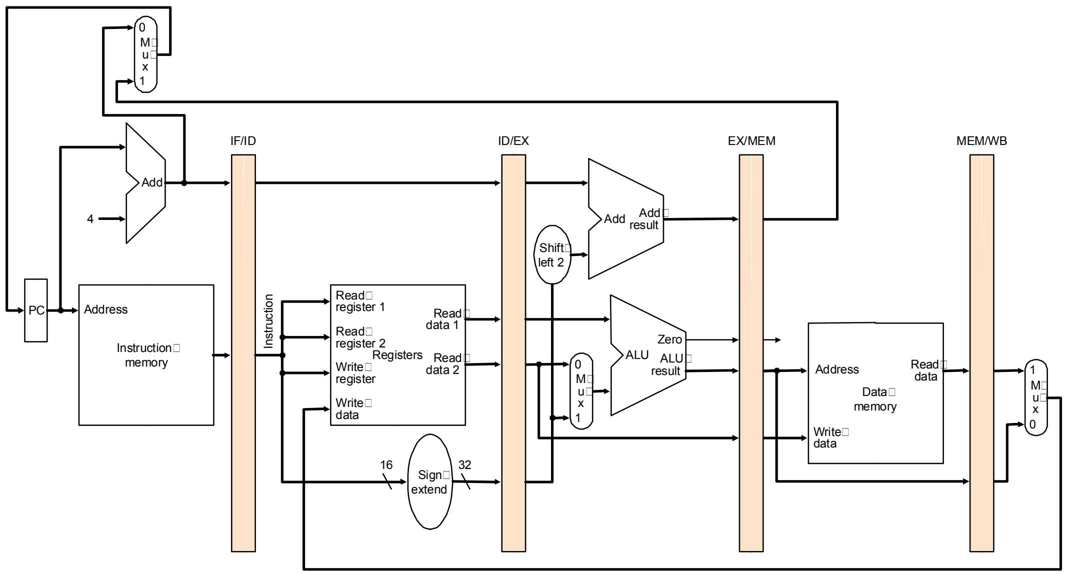
- What makes it easy
  - all instructions are the same length
  - just a few instruction formats
  - memory operands appear only in loads and stores
- What makes it hard?
  - structural hazards: suppose we had only one memory
  - control hazards: need to worry about branch instructions
  - data hazards: an instruction depends on a previous instruction
- We'll build a simple pipeline and look at these issues

Basic Idea



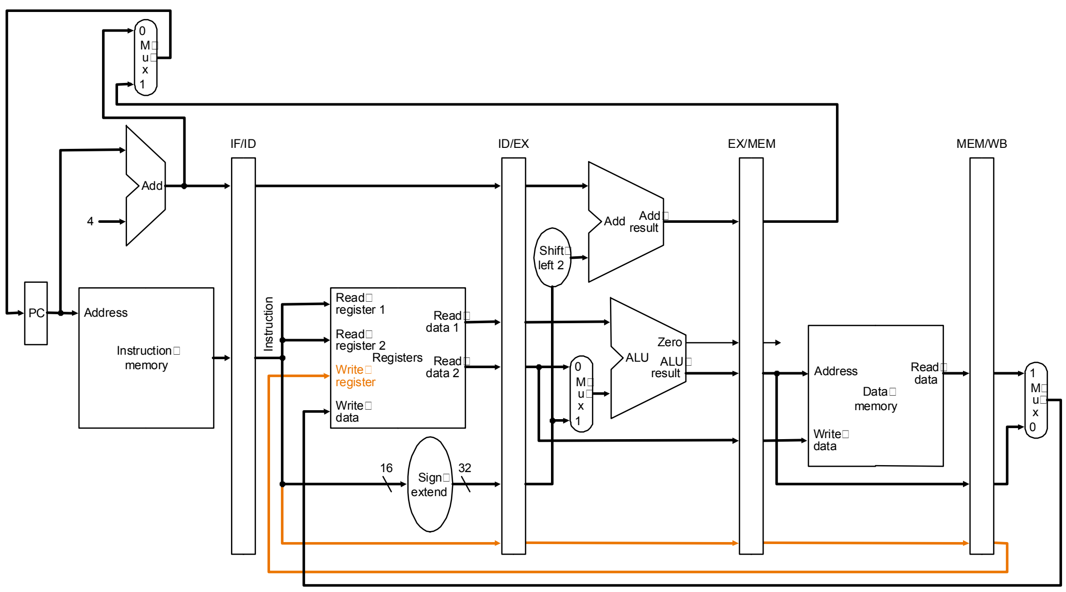
- *What do we need to add to actually split the datapath into stages?*

Pipelined Datapath

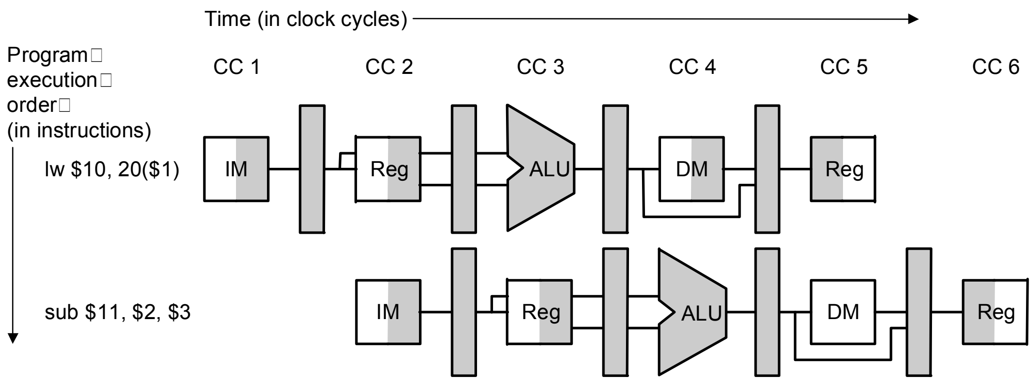


Can you find a problem even if there are no dependencies?
What instructions can we execute to manifest the problem?

Corrected Datapath

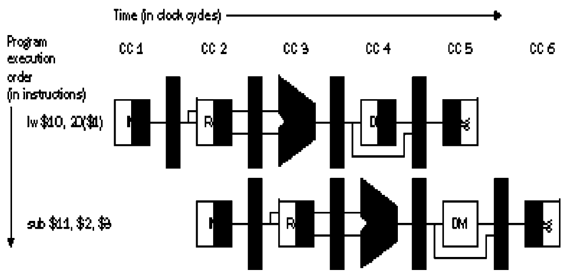


Graphically Representing Pipelines



- Can help with answering questions like:
  - how many cycles does it take to execute this code?
  - what is the ALU doing during cycle 4?
  - use this representation to help understand datapaths

Graphically Representing Pipelines



- Can help with answering questions like:
  - how many cycles does it take to execute this code?
  - what is the ALU doing during cycle 4?
  - use this representation to help understand datapaths

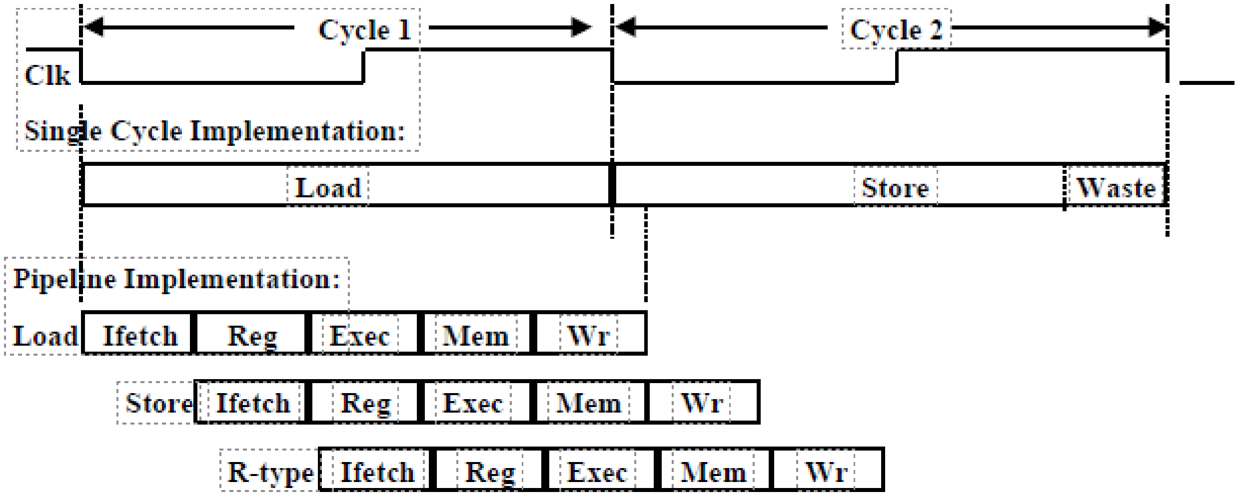
Conventional Pipelined Execution Representation

Time →

IFetch	Dcd	Exec	Mem	WB					
	IFetch	Dcd	Exec	Mem	WB				
		IFetch	Dcd	Exec	Mem	WB			
			IFetch	Dcd	Exec	Mem	WB		
				IFetch	Dcd	Exec	Mem	WB	
					IFetch	Dcd	Exec	Mem	WB

↓ Program Flow

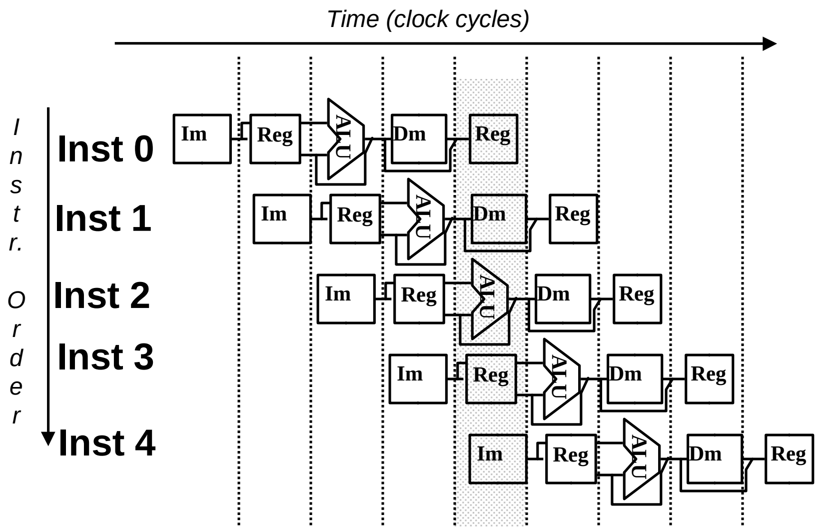
Single Cycle vs. Pipeline



Why Pipeline?

- Suppose we execute 100 instructions
- Single Cycle Machine
  - 45 ns/cycle x 1 CPI x 100 inst = 4500 ns
- Ideal pipelined machine
  - 10 ns/cycle x (1 CPI x 100 inst + 4 cycle drain) = 1040 ns

Why Pipeline? Because the resources are there!



Can pipelining get us into trouble?

- Yes: Pipeline Hazards
  - structural hazards: attempt to use the same resource two different ways at the same time
    - E.g., combined washer/dryer would be a structural hazard or folder busy doing something else (watching TV)
  - data hazards: attempt to use item before it is ready
    - E.g., one sock of pair in dryer and one in washer; can't fold until get sock from washer through dryer
    - instruction depends on result of prior instruction still in the pipeline
  - control hazards: attempt to make a decision before condition is evaulated
    - E.g., washing football uniforms and need to get proper detergent level; need to see after dryer before next load in
    - branch instructions
- Can always resolve hazards by waiting
  - pipeline control must detect the hazard
  - take action (or delay action) to resolve hazards