Recitation 13… I guess?

Pipelines

Today’s agenda

- We will discuss in recitation
  - The RISC-V Pipeline
    - Motivation
    - Design
    - Hazards and Forwarding
- For homework tonight
  - R13
    - You will reorder instructions and insert nops to remove hazards

Pipeline

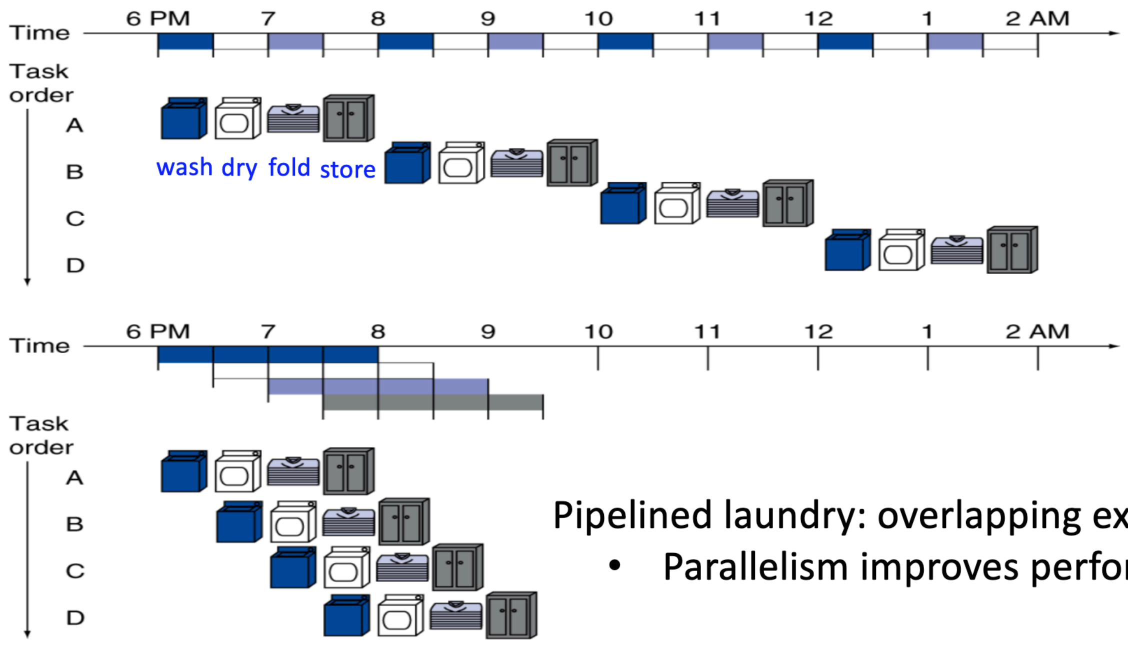
Motivation

CPU Pipeline

- The alternative is to have a single large datapath and execute each instruction within one clock cycle
  - While this may be simpler conceptually, the clock cannot tick faster than the rate to let the **slowest** instruction run to completion
  - This means the clock must be run at a slow rate
- Pipelining allows the clock to run faster
  - This gives us a tradeoff between latency (how long between when an instruction starts and when it completes) and throughput (how many instructions can execute completely within a window of time)
  - If we substantially increase the clock speed, we can execute more instructions per second

Stolen Slide

Pipelining: a laundry analogy



Pipelined laundry: overlapping execution

- Parallelism improves performance

Pipeline

Design

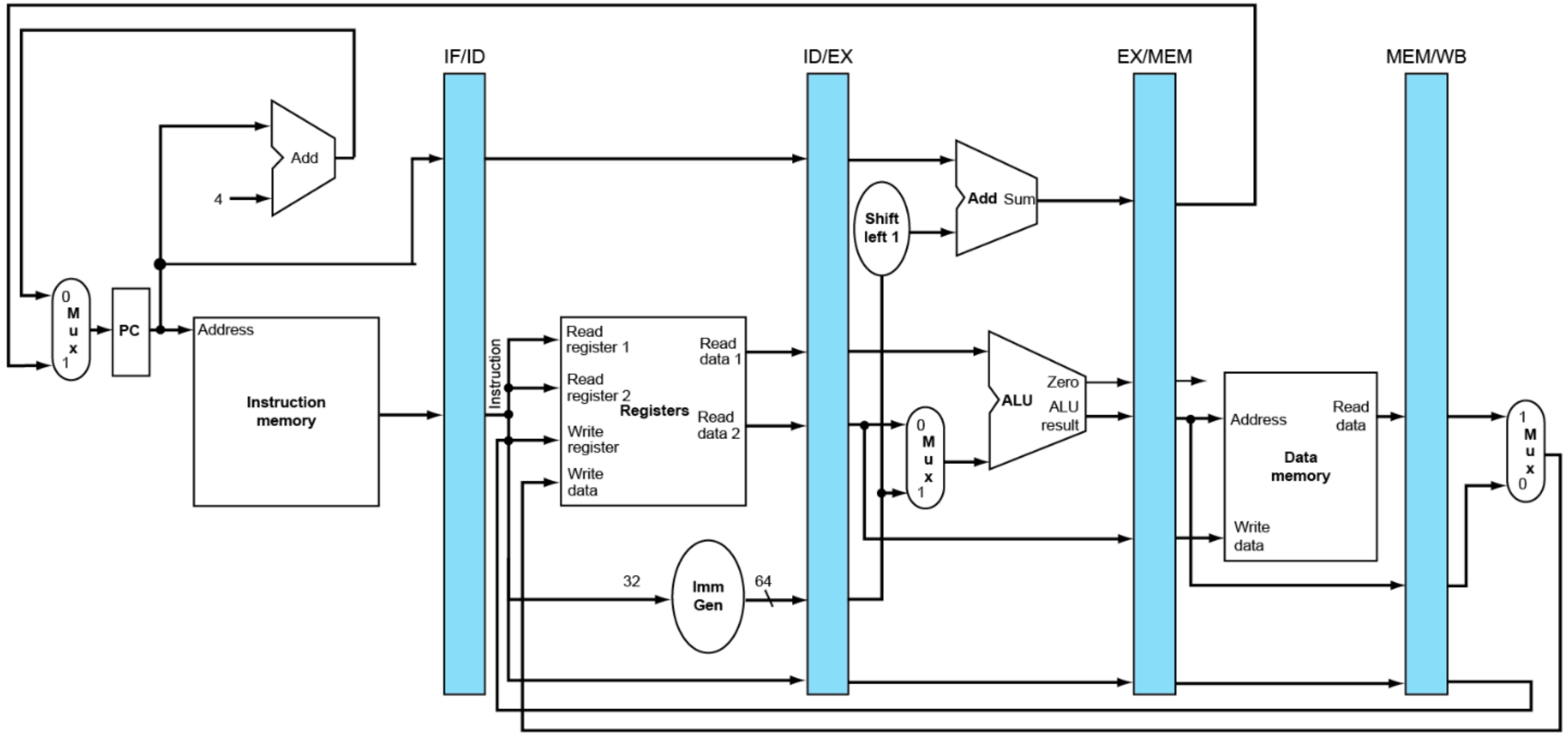
RISC-V Pipeline

- We split the instruction memory from the data memory
  - Otherwise reading data would delay reading an instruction
- There are 5 stages in the RISC-V pipeline
  - Instruction Fetch – IF
    - It takes a cycle to read an instruction from instruction memory
  - Instruction Decode – ID
    - Decode the instruction, read registers, and predict branching
  - Execute – EX
    - Performs computations using the ALU or performs shift operations
  - Memory Access - MEM
    - Read or write to memory
  - Write Back – WB
    - Write results to registers

Pipeline Registers

- In between each stage of the pipeline, we have registers store information
  - Each stage reads from the previous register to figure out what to do
  - Each stage writes to the next register to pass the instruction along

Stolen Slide 2



Pipeline

Hazards

Hazard Overview

- Sometimes instructions depend on each other
- In fact, that is very often the case, that some instruction uses the result of a previous instruction
- This becomes an issue, however, when an instruction depends on a value that has been computed but not written back yet

Hazard Motivation

- Consider the following C Program:

```
int calc(int a, int b, int c) {
    int d = a + b;
    d = d + c;
    return d;
}
```

Hazard Motivation

- Consider the following C Program:

```
int calc(int a, int b, int c) {
	a = a + b;
	a = a + c;
	return a;
}
```

- Which becomes :

```
add x10, x10, x11
add x10, x10, x12
```

What might cause sadness here?

Hazard Fix

- There are two approaches to avoiding hazards
  1. Forwarding
     - Be able to send data to an earlier location in the pipeline before write back
     - We do this by adding some multiplexers in the EX stage to choose between more recent values and the value from the register file
  2. Stalling
     - Add a “bubble” to the pipeline to give us time to resolve the hazard
     - We do this by setting some of the controls to 0 and by preventing the PC from changing

Hazard Detection - Discussion

- When do we encounter the hazards?
- Is forwarding always enough?