Chapter 6: Designing a Pipelined CPU

• What are our resources?
  1 washer, 1 dryer,
  1 folder (you), 1 “put awayer” (roommate)

What % of the time are they idle?
Chapter 6: Designing a Pipelined CPU
Chapter 6: Designing a Pipelined CPU

What % of the time are resources idle?

- steady-state
- ramp up
- ramp down
Chapter 6: Designing a Pipelined CPU

What is our roommate takes off? What happens to the pipeline?
Chapter 6: Designing a Pipelined CPU

What if our roommate is gone? What happens to the pipeline?
Chapter 6: Designing a Pipelined CPU

What if our roommate is gone? What happens to the pipeline?

Massive Laundry Pile
Chapter 6: Designing a Pipelined CPU

No Laundry Pile

Scheduling work later reduces “laundry pile”
Chapter 6: Designing a Pipelined CPU

Scheduling work later reduces “laundry pile”
Execution in a Pipelined Datapath
# Instruction Latencies and Throughput

## Single-Cycle CPU

<table>
<thead>
<tr>
<th>Cycle 1</th>
<th>Cycle 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>IF Dec</td>
</tr>
<tr>
<td></td>
<td>EX Mem</td>
</tr>
<tr>
<td></td>
<td>WB</td>
</tr>
</tbody>
</table>

**Latency:**  
**Throughput:**

## Multiple Cycle CPU

<table>
<thead>
<tr>
<th>Cycle 1</th>
<th>Cycle 2</th>
<th>Cycle 3</th>
<th>Cycle 4</th>
<th>Cycle 5</th>
<th>Cycle 6</th>
<th>Cycle 7</th>
<th>Cycle 8</th>
<th>Cycle 9</th>
<th>Cycle 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>IF Dec</td>
<td>EX Mem</td>
<td>WB</td>
<td>IF Dec</td>
<td>EX Mem</td>
<td>WB</td>
<td>IF Dec</td>
<td>EX Mem</td>
<td>WB</td>
</tr>
</tbody>
</table>

**Latency:**  
**Throughput:**

## Pipelined CPU

<table>
<thead>
<tr>
<th>Cycle 1</th>
<th>Cycle 2</th>
<th>Cycle 3</th>
<th>Cycle 4</th>
<th>Cycle 5</th>
<th>Cycle 6</th>
<th>Cycle 7</th>
<th>Cycle 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>IF Dec</td>
<td>EX Mem</td>
<td>WB</td>
<td>IF Dec</td>
<td>EX Mem</td>
<td>WB</td>
<td>IF Dec</td>
</tr>
<tr>
<td></td>
<td>EX Mem</td>
<td>WB</td>
<td>IF Dec</td>
<td>EX Mem</td>
<td>WB</td>
<td>IF Dec</td>
<td>EX Mem</td>
</tr>
<tr>
<td></td>
<td>WB</td>
<td>IF Dec</td>
<td>EX Mem</td>
<td>WB</td>
<td>IF Dec</td>
<td>EX Mem</td>
<td>WB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WB</td>
<td>IF Dec</td>
<td>EX Mem</td>
<td>WB</td>
<td>IF Dec</td>
<td>EX Mem</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WB</td>
<td>IF Dec</td>
<td>EX Mem</td>
<td>WB</td>
<td>IF Dec</td>
</tr>
</tbody>
</table>

**Latency:**  
**Throughput:**
Self Check!

- If my single cycle CPU has a cycle time of 14ns and my multicycle CPU has a cycle time of 3ns and my pipelined CPU has a cycle time of 3ns, what is the relative performance of my machines?
  - What kind of answer would you provide?
  - What kind of information do you need to know?

\[ ET = IC \times CPI \times CT \]

What differs across machines? CT and CPI

Single: CT = 14ns, CPI = 1

Multi: CT = 3ns CPI=?? NEED DYN INST LOAD INFO

PIPEDLINE: CT = 3ns CPI = ?? WHAT IS IT? ALWAYS 5?
Pipelining Advantages

- Higher maximum throughput
- Higher utilization of CPU resources

- But, more hardware needed, perhaps complex control

  before, a simple FSM could guide execution of one instruction at a time

  but, now if we implemented the FSM, it would need to control 5 instructions simultaneously!


Mixed Instructions in the Pipeline

Cycle #

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>IF</td>
<td>Dec</td>
<td>EX</td>
<td>Mem</td>
<td>WB</td>
</tr>
<tr>
<td>add</td>
<td>IF</td>
<td>Dec</td>
<td>EX</td>
<td>WB</td>
<td></td>
</tr>
</tbody>
</table>

What’s wrong with this?
To avoid structural hazard, schedule resource usage homogeneously.

<table>
<thead>
<tr>
<th>Cycle #</th>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>IF</td>
<td>Dec</td>
<td>EX</td>
<td>Mem</td>
<td>WB</td>
<td></td>
</tr>
<tr>
<td>add</td>
<td>IF</td>
<td>Dec</td>
<td>EX</td>
<td></td>
<td>Mem</td>
<td>WB</td>
</tr>
</tbody>
</table>
Pipeline Principles

• All instructions that share a pipeline must have the same stages in the same order.
  - therefore, *add* does nothing during Mem stage
  - *sw* does nothing during WB stage

• All intermediate values must be latched each cycle.

• There is no functional block reuse
  - example: we need 2 adders and ALU (like in single-cycle)
Pipelined Datapath

Instruction Fetch  Instruction Decode/ Register Fetch  Execute/ Address Calculation  Memory Access  Write Back

Is this more similar to multicycle or single cycle datapath?