CSE 141 – Computer Architecture
Fall 2003

Lectures 11
Overview of Pipelining

Pramod V. Argade
Schedule

<table>
<thead>
<tr>
<th>Lecture #</th>
<th>Date</th>
<th>Day</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sep. 25</td>
<td>Thursday</td>
<td>Introduction, Ch. 1</td>
</tr>
<tr>
<td>2</td>
<td>Sep. 30</td>
<td>Tuesday</td>
<td>Performance, Ch. 2</td>
</tr>
<tr>
<td>3</td>
<td>Oct. 2</td>
<td>Thursday</td>
<td>ISA, Ch. 3</td>
</tr>
<tr>
<td>4</td>
<td>Oct. 7</td>
<td>Tuesday</td>
<td>Arithmetic, Ch. 4</td>
</tr>
<tr>
<td>5</td>
<td>Oct. 9</td>
<td>Thursday</td>
<td>Arithmetic, Ch. 4, Continued</td>
</tr>
<tr>
<td>6</td>
<td>Oct. 14</td>
<td>Tuesday</td>
<td>Single cycle CPU, Ch. 5</td>
</tr>
<tr>
<td>7</td>
<td>Oct. 16</td>
<td>Thursday</td>
<td>Single-cycle CPU, Ch. 5</td>
</tr>
<tr>
<td>8</td>
<td>Oct. 21</td>
<td>Tuesday</td>
<td>Multi-cycle CPU, Ch. 5</td>
</tr>
<tr>
<td>9</td>
<td>Oct. 23</td>
<td>Thursday</td>
<td>Multi-cycle CPU, Ch. 5</td>
</tr>
<tr>
<td>10</td>
<td>Oct. 28</td>
<td>Tuesday</td>
<td>Classes cancelled due to wildfires</td>
</tr>
<tr>
<td>11</td>
<td>Oct. 30</td>
<td>Thursday</td>
<td>Exceptions and Review for Midterm</td>
</tr>
<tr>
<td>12</td>
<td>Nov. 4</td>
<td>Tuesday</td>
<td>Mid-term Exam</td>
</tr>
<tr>
<td>13</td>
<td>Nov. 6</td>
<td>Thursday</td>
<td>Pipelining, Ch. 6</td>
</tr>
<tr>
<td>No Class</td>
<td>Nov. 11</td>
<td>Tuesday</td>
<td>Veteran's Day Holiday</td>
</tr>
<tr>
<td>14</td>
<td>Nov. 13</td>
<td>Thursday</td>
<td>Data and control hazards, Ch. 6</td>
</tr>
<tr>
<td>15</td>
<td>Nov. 18</td>
<td>Tuesday</td>
<td>Data and control hazards, Ch. 6</td>
</tr>
<tr>
<td>16</td>
<td>Nov. 20</td>
<td>Thursday</td>
<td>Data and control hazards, Ch. 6</td>
</tr>
<tr>
<td>17</td>
<td>Nov. 25</td>
<td>Tuesday</td>
<td>Advanced pipelining issues, Ch. 6</td>
</tr>
<tr>
<td>No Class</td>
<td>Nov. 27</td>
<td>Thursday</td>
<td>Thanksgiving Holiday</td>
</tr>
<tr>
<td>18</td>
<td>Dec. 2</td>
<td>Tuesday</td>
<td>Memory &amp; cache design, Ch. 7</td>
</tr>
<tr>
<td>19</td>
<td>Dec. 4</td>
<td>Thursday</td>
<td>Memory &amp; cache design, Ch. 7</td>
</tr>
<tr>
<td>Dec. 9</td>
<td>Tuesday</td>
<td>Final Exam</td>
<td></td>
</tr>
</tbody>
</table>

Pipelining: Its Natural!

- 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
Sequential Laundry

- Sequential laundry takes 6 hours for 4 loads
- If they learned pipelining, how long would laundry take?

Pipelined Laundry

- Pipelined laundry takes 3.5 hours for 4 loads
Pipelining Overview

- What is pipelining?
  - Multiple instructions are overlapped in execution

- Notes:
  - Time for completion of a single instruction is not shorter
  - Pipelining does not change latency
  - Multiple tasks operate simultaneously
  - Pipelining increases the throughput
  - Pipelining rate is limited by the slowest stage
  - Potential speedup = number of pipeline stages
  - Time to “fill” pipeline and time to “drain” it reduces speedup

Pipelining

- Requires separable jobs/stages
- Requires separate resources
- Achieves parallelism with replication
- Often increases single-task (e.g., instruction, laundry load) latency: all stages must take the same amount of time
- Pipeline efficiency (keeping the pipeline full) critical to performance
- Time between instructions pipelined
  
  \[(\text{Time between instructions non-pipelined})/(\# \text{ Pipe Stages})\]

- Fundamentally invisible to the programmer
Non-Pipelined vs. Pipelined Execution

<table>
<thead>
<tr>
<th>Instruction Class</th>
<th>Instruction Fetch</th>
<th>Register Read</th>
<th>ALU Operation</th>
<th>Data Access</th>
<th>Register Write</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Word (lw)</td>
<td>2 ns</td>
<td>1 ns</td>
<td>2 ns</td>
<td>2 ns</td>
<td>1 ns</td>
<td>8 ns</td>
</tr>
<tr>
<td>Store Word (sw)</td>
<td>2 ns</td>
<td>1 ns</td>
<td>2 ns</td>
<td>2 ns</td>
<td>7 ns</td>
<td>10 ns</td>
</tr>
<tr>
<td>R-Format</td>
<td>2 ns</td>
<td>1 ns</td>
<td>2 ns</td>
<td>2 ns</td>
<td>1 ns</td>
<td>6 ns</td>
</tr>
<tr>
<td>Branch (beq)</td>
<td>2 ns</td>
<td>1 ns</td>
<td>2 ns</td>
<td>2 ns</td>
<td>5 ns</td>
<td>9 ns</td>
</tr>
</tbody>
</table>

MIPS ISA and Pipelining

- All instructions are the same length
  - Easier to fetch
  - Easier to decode in second stage
- Only a few instruction formats
  - Register field location fixed
  - Operand fetch and instruction decode in parallel
- Load/store architecture
  - Memory operands appear only in load and store instructions
  - Execute stage calculates memory address and result for R-type
- Operands must be aligned in memory
  - Single data transfer requires single memory access
Pipelining Challenges

- Hazards: Situation where next instruction cannot execute
  - 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 talk about modern processors and what really makes it hard:
  - Exception handling
  - Trying to improve performance with out-of-order execution, etc.

Review: Single-cycle CPU
Review: Multi-cycle CPU

Review -- Instruction Latencies

- **Single-Cycle CPU**

<table>
<thead>
<tr>
<th>Load</th>
<th>Ifetch</th>
<th>Reg/Dec</th>
<th>Exec</th>
<th>Mem</th>
<th>Wr</th>
</tr>
</thead>
</table>

- **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>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Mem</td>
</tr>
<tr>
<td>Add</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Wr</td>
</tr>
</tbody>
</table>
Instruction Latencies and Throughput

• Single-Cycle CPU

• Multiple Cycle CPU

• Pipelined CPU

Pipeline Performance Considerations

- CPU throughput = Instructions Per Cycle (IPC)
  - number of instructions completed per cycle
- Execution Time = (Instruction Count) * CPI * (Cycle Time)
- Complexity has a cost
  - e.g., Register overhead
  - Uneven stage latencies
- Pipeline clock cannot run faster than
  - slowest pipeline stage
  - pipeline overhead
- Can’t always keep the pipeline full
  - why not?
Pipeline Stages

IF: Instruction fetch
ID: Instruction decode and register fetch
EX: Execution and effective address calculation
MEM: Memory access
WB: Write back

Pipelining: Basic Idea

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

- Instructions advance from one stage to another every clock
- Instructions and data moves from left to right
  - Exceptions
    - WB stage writes to register file (Potential data hazard)
    - PC = Branch address from Mem stage (Potential control hazard)
- No registers in WB stage
  - Write registers already exist
Graphical Representation of a Pipeline

Execution in a Pipelined Datapath
Mixed Instructions in the Pipeline

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 registered each cycle.
- There is no functional block reuse
Pipelined Datapath

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

Pipelined Datapath in Action

add $10, $1, $2
Instruction Decode/
Register Fetch
Execute/
Address Calculation
Memory Access
Write Back

Pramod Argade
UCSD CSE 141, Fall 2003
Pipelined Datapath in Action

lw S12, 1000(S4)  add S10, S1, S2  Execute/Address Calculation  Memory Access  Write Back

lw S12, 1000(S4)  add S10, S1, S2  Memory Access  Write Back
Pipelined Datapath in Action

Instruction Fetch

Instruction Decode/ Register Fetch

sub $15, $4, $1  lw $12, 1000($4)  add $10, $1, $2  Write Back

Pipelined Datapath in Action

Instruction Fetch

Instruction Decode/ Register Fetch

sub $15, $4, $1  lw $12, 1000($4)  add $10, $1, $2  Write Back
Pipeline with Destination Register

Instruction Fetch
Instruction Decode/ Register Fetch

sub S15, S4, S1
lw S12, 1000(S4)
add S10, S1, S2

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/

execute/
Pipeline control

- We have 5 stages. What needs to be controlled in each stage?
  - Instruction Fetch and PC Increment
  - Instruction Decode / Register Fetch
  - Execution
  - Memory Stage
  - Write Back

- How would control be handled in an automobile plant?
  - a fancy control center telling everyone what to do?
Pipeline Control

- Use combinational Logic!
  - Signals generated once, but follow instruction through the pipeline

```
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Reg</th>
<th>Dest</th>
<th>ALU Op1</th>
<th>ALU Op0</th>
<th>ALU Src</th>
<th>Branch</th>
<th>Mem Read</th>
<th>Mem Write</th>
<th>Reg Write</th>
<th>Mem to Reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-format</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lw</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lw</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Pipelined Datapath and Control