Announcements

- **Discussions Sections for 141:**
  - Fridays, 10:00 - 10:50 am, Peterson Hall 104 (Chris)
  - Fridays, 2:00 - 2:50 pm, Peterson Hall 104 (Leo)

- **Reading Assignment**
  - Chapter 5. Processor: Datapath and Control
    - Sec. 5.1 - 5.4

- **Homework 4: Due Mon., April 25th in class**
  - 5.2, 5.8, 5.9, 5.10, 5.13, 5.20, 5.22, 5.28

- **Quiz**
  - **When:** Mon., April 25th, First 10 minutes of the class
  - **Topic:** Single Cycle CPU, Chapter 5  
  - **Need:** Paper, pen

### Course Schedule

<table>
<thead>
<tr>
<th>Lecture #</th>
<th>Date</th>
<th>Day</th>
<th>Lecture Topic</th>
<th>Quiz Topic</th>
<th>Homework Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3/28</td>
<td>Monday</td>
<td>Introduction, Ch. 1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>3/30</td>
<td>Wednesday</td>
<td>Performance, Ch. 4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>4/4</td>
<td>Monday</td>
<td>ISA, Ch. 2</td>
<td>Performance #1</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>4/6</td>
<td>Wednesday</td>
<td>Arithmetic, Ch. 3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>4/11</td>
<td>Monday</td>
<td>Arithmetic, Ch. 3</td>
<td>ISA #2</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>4/13</td>
<td>Wednesday</td>
<td>Single cycle CPU, Ch. 5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>4/18</td>
<td>Monday</td>
<td>Single cycle CPU, Ch. 5</td>
<td>Arithmetic #3</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>4/20</td>
<td>Wednesday</td>
<td>Multi-cycle CPU, Ch. 5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>4/25</td>
<td>Monday</td>
<td>Multi-cycle CPU, Ch. 5</td>
<td>Single Cycle CPU #4</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>4/27</td>
<td>Wednesday</td>
<td>Review for the Midterm</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>5/2</td>
<td>Friday</td>
<td>Mid-term Exam</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>5/4</td>
<td>Wednesday</td>
<td>Exceptions, Ch. 5 and Pipeline, Ch. 6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>5/9</td>
<td>Monday</td>
<td>Pipeline, Ch. 6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>5/11</td>
<td>Wednesday</td>
<td>Data and control hazards, Ch. 6</td>
<td>Pipeline Hazards #5</td>
<td>-</td>
</tr>
<tr>
<td>15</td>
<td>5/16</td>
<td>Monday</td>
<td>Data and control hazards, Ch. 6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>5/18</td>
<td>Wednesday</td>
<td>Memory &amp; cache design, Ch. 7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>17</td>
<td>5/23</td>
<td>Monday</td>
<td>Memory &amp; cache design, Ch. 7</td>
<td>Cache #6</td>
<td>-</td>
</tr>
<tr>
<td>18</td>
<td>5/25</td>
<td>Wednesday</td>
<td>Virtual Memory &amp; cache design, Ch. 7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>No Class</td>
<td>5/30</td>
<td>Monday</td>
<td>Memorial Day Holiday</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>18a</td>
<td>6/1</td>
<td>Wednesday</td>
<td>Course Review</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>19</td>
<td>6/10</td>
<td>Friday</td>
<td>Final Exam</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Single Cycle CPU

single-cycle CPU Observations

- All instructions take the same amount of time
  - Make common case fast!
- Assume following delays
  - Memory units: 200 ps, ALU & Adders: 100 ps, Reg. File: 50 ps
  
<table>
<thead>
<tr>
<th>Instruction Class</th>
<th>Instruction Memory</th>
<th>Register Read</th>
<th>ALU Operation</th>
<th>Data Memory</th>
<th>Register Write</th>
<th>Total (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU Type</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td>0</td>
<td>50</td>
<td>400</td>
</tr>
<tr>
<td>Load Word</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td>200</td>
<td>50</td>
<td>600</td>
</tr>
<tr>
<td>Store word</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td>200</td>
<td></td>
<td>550</td>
</tr>
<tr>
<td>Branch</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td></td>
<td></td>
<td>350</td>
</tr>
<tr>
<td>Jump</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>200</td>
</tr>
</tbody>
</table>

- Longest instruction (LW) determines the clock cycle for the machine!
- Single-cycle implementation is inefficient in performance and HW cost
Single Cycle CPU: How to improve the Performance?

- Problem with single-cycle CPU
  - Cycle time long enough to complete the longest instruction
- A solution: Multi-cycle CPU
  - Break up execution into smaller tasks, each task taking a cycle, different instructions requiring different numbers of cycles or tasks
- Other advantages of Multi-cycle CPU
  - Reuse of functional units (e.g., alu, memory)
- Performance = Instructions * CPI * Cycle time

Multicycle CPU

- Single Cycle Implementation:
  - Load
  - Store
  - Waste
- Multiple Cycle Implementation:
  - R-type
Breaking Execution Into Clock Cycles

- Introduces extra registers when:
  - Signal is computed in one clock cycle and used in another, AND
  - The inputs to the functional block that outputs this signal can change before the signal is written into a state element.
- Significantly complicates control. Why?
- The goal is to balance the amount of work done each cycle.

Multi-cycle Datapath: High-level View
Breaking Execution Into Clock Cycles

- Each stage has at most one of following
  - ALU Operation
  - Register file access
  - One memory access
- Save output from above into a temporary data register
  - Written every clock cycle, no write control needed
- We will have five execution steps (not all instructions use all five)
  - fetch
  - decode & register fetch
  - execute
  - memory access
  - write-back
- Perform tasks early if they are not harmful

Multi-cycle Datapath with Control

Note:
- I-Mem and D-Mem combined into one memory
- PC adder and PC offset adder have been eliminated
1. Instruction Fetch Cycle
(Instruction Independent)

IR = Memory[PC]
PC = PC + 4 [may not be the final value of PC]

2. Instruction Decode and Register Fetch Cycle
(Instruction Independent)

A = Register[IR[25-21]]
B = Register[IR[20-16]]
ALUOut = PC + (sign-extend {IR[15 - 0]} << 2) [May not get used.]
3. Execution, memory address computation, or branch completion

Memory reference (load or store)
ALUOut = A + sign-extend(IR[15-0])

R-type
ALUOut = A op B

Branch (completes in this cycle)
if(A == B) PC = ALUOut (Add bus from ALUOut to PC)

4. Memory access or R-type completion

Memory reference:
load
MDR = Memory[ALUout]

store
Memory[ALUout] = B

R-type (Completes in this cycle):
Reg[IR[15-11]] = ALUout
5. Memory Write-Back

Reg[IR[20-16]] = MDR [Completes in this cycle]

Summary of execution steps

<table>
<thead>
<tr>
<th>Step</th>
<th>R-type</th>
<th>Memory</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Fetch</td>
<td>IR = Memory[PC]</td>
<td></td>
<td>PC = PC + 4</td>
</tr>
<tr>
<td>Instruction Decode/</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>register fetch</td>
<td>A = Registers[IR[25-21]]</td>
<td>B = Registers[IR[20-16]]</td>
<td></td>
</tr>
<tr>
<td>Execution, address</td>
<td>ALUoutput = A op B</td>
<td>ALUoutput = A + sign-extend</td>
<td></td>
</tr>
<tr>
<td>computation, branch</td>
<td></td>
<td>[IR[15-0]] &lt;&lt; 2</td>
<td></td>
</tr>
<tr>
<td>completion</td>
<td></td>
<td>if (A==B) then PC = Target</td>
<td></td>
</tr>
<tr>
<td>Memory access or R-</td>
<td>Reg[IR[15-11]] = ALUoutput</td>
<td>memory-data = or Mem[ALUoutput]</td>
<td></td>
</tr>
<tr>
<td>type completion</td>
<td></td>
<td>or Mem[ALUoutput] = B</td>
<td></td>
</tr>
<tr>
<td>Write-back</td>
<td>Reg[IR[20-16]] = memory-data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Multicycle Control

- Single-cycle control used combinational logic
- Multi-cycle control
  - Need to specify a sequence of controls for each cycle
- FSM defines a succession of states, transitions between states (based on inputs), and outputs (based on state and inputs)
  - Outputs not explicitly asserted are deasserted
  - Multiplexor control is always specified
- First two states same for every instruction, next state depends on opcode.

Review: Finite State Machines

- Finite state machines:
  - a set of states and
  - next state function (determined by current state and the input)
  - output function (determined by current state and possibly input)
  - We’ll use a Moore machine (output based only on current state)
Multi-cycle CPU: Control FSM

- Instruction fetch
- Decode and Register Fetch
- Memory instructions
- R-type instructions
- Branch instructions
- Jump instruction

Complete Datapath and Control
State 1. Instruction Fetch Cycle  
(Instruction Independent)

IR = Memory[PC]
PC = PC + 4  \textit{(may not be final value of PC)}

State 2. Instruction Decode and Reg. Fetch Cycle  
(Instruction Independent)

A = Register[IR[25-21]]
B = Register[IR[20-16]]
ALUOut = PC + (sign-extend (IR[15-0]) \ll 2)  \textit{[May not be used]}
First two states of the FSM

**Instruction Fetch, state 0**
- IR = Memory[PC]
- PC = PC + 4

**Instruction Decode/ Register Fetch, state 1**
- A = Register[IR[25-21]]
- B = Register[IR[20-16]]
- ALUOut = PC + (sign-extend (IR[15-0]) << 2)

---

**Announcements**

- **Discussions Sections for 141:**
  - Fridays, 10:00 - 10:50 am, Peterson Hall 104 (Chris)
  - Fridays, 2:00 - 2:50 pm, Peterson Hall 104 (Leo)

- **Reading Assignment**
  - Chapter 5. Processor: Datapath and Control
    - Sec. 5.1 - 5.4

- **Homework 4: Due Mon., April 25th in class**
  - 5.2, 5.8, 5.9, 5.10, 5.13, 5.20, 5.22, 5.28

- **Quiz**
  - When: Mon., April 25th, First 10 minutes of the class
  - Topic: Single Cycle CPU, Chapter 5
  - Need: Paper, pen