CSE141: Introduction to Computer Architecture

Instructor: Pramod V. Argade (p2argade@cs.ucsd.edu)
Office Hour:
Tue. 7:30 - 9:00 (Center 105)
Wed. 5:00 - 6:00 PM (HSS 1330)
By Appointment

TAs:
Chengmo Yang: c5yang@cs.ucsd.edu
Wenjing Rao: wrao@cs.ucsd.edu

Lecture: Mon/Wed. 6:00 - 8:50 PM, HSS 1330

Textbook: Computer Organization & Design
The Hardware Software Interface, 3rd Edition.
Authors: Patterson and Hennessy

Web-page: http://www.cse.ucsd.edu/classes/su05/cse141
<table>
<thead>
<tr>
<th>Lecture #</th>
<th>Date</th>
<th>Time</th>
<th>Room</th>
<th>Topic</th>
<th>Quiz topic</th>
<th>Homework Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mon. 6/27</td>
<td>6 - 8:50 PM</td>
<td>HSS 1330</td>
<td>Introduction, Ch. 1, ISA, Ch. 2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Wed. 6/29</td>
<td>6 - 8:50 PM</td>
<td>HSS 1330</td>
<td>Arithmetic, Ch. 3</td>
<td>ISA Ch. 2</td>
<td>#1</td>
</tr>
<tr>
<td>-</td>
<td>Mon. 7/4</td>
<td>No Class</td>
<td></td>
<td>July 4th Holiday</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>Wed. 7/6</td>
<td>6 - 8:50 PM</td>
<td>HSS 1330</td>
<td>Performance, Ch. 4, Single-cycle CPU Ch. 5</td>
<td>Arithmetic Ch. 3</td>
<td>#2</td>
</tr>
<tr>
<td>4</td>
<td>Mon. 7/11</td>
<td>6 - 8:50 PM</td>
<td>HSS 1330</td>
<td>Single-cycle CPU Ch. 5 Cont. Multi-cycle CPU Ch. 5</td>
<td>Performance Ch. 4</td>
<td>#3</td>
</tr>
<tr>
<td>5</td>
<td>Tue. 7/12</td>
<td>7:30 - 8:50 PM</td>
<td>HSS 1330</td>
<td>Multi-cycle CPU Ch. 5 Cont. (July 4th make up class)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>Wed. 7/13</td>
<td>6 - 8:50 PM</td>
<td>HSS 1330</td>
<td>Single and Multicycle CPU Examples and Review for Midterm</td>
<td>Single-cycle CPU Ch. 5</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>Mon. 7/18</td>
<td>6 - 8:50 PM</td>
<td>HSS 1330</td>
<td>Mid-term Exam Exceptions</td>
<td></td>
<td>#4</td>
</tr>
<tr>
<td>8</td>
<td>Tue. 7/19</td>
<td>7:00 - 8:50 PM</td>
<td>HSS 1330</td>
<td>Pipelining Ch. 6 (July 4th make up class)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>Wed. 7/20</td>
<td>6 - 8:50 PM</td>
<td>HSS 1330</td>
<td>Hazards, Ch. 6</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>Mon. 7/25</td>
<td>6 - 8:50 PM</td>
<td>HSS 1330</td>
<td>Memory Hierarchy &amp; Caches Ch. 7</td>
<td>Hazards Ch. 6</td>
<td>#5</td>
</tr>
<tr>
<td>11</td>
<td>Wed. 7/27</td>
<td>6 - 8:50 PM</td>
<td>HSS 1330</td>
<td>Virtual Memory, Ch. 7 Course Review</td>
<td>Cache Ch. 7</td>
<td>#6</td>
</tr>
<tr>
<td>12</td>
<td>Sat. 7/30</td>
<td>TBD</td>
<td>TBD</td>
<td>Final Exam</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>
Announcements

● **Reading Assignment**
  – Chapter 5. The Processor: Datapath and Control
    Sections 5.6
  – Chapter 6. Enhancing Performance with Pipelining
    Sections 6.1 - 6.10

● **Homework 5: Due Mon., July 25th in class**
  5.49, 5.50
  6.1, 6.2, 6.4, 6.6, 6.15, 6.17

● **Quiz**
  **When:** Mon., July 25th  
  **Topic:** Pipelining and Hazards
MIPS ISA and Pipelining

- All instructions are the same length
  - Easier to fetch from instruction memory
  - Easier to decode in second stage

- Only a few instruction formats
  - Register field location(s) 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
  - One memory data transfer requires a single memory access

- Following instructions to be implemented
  - LW, SW, ADD, SUB, AND, OR, SLT, BEQ
Pipelining Challenges

- Hazards: Situations where next instruction cannot execute
  - Structural hazards:
    - Suppose we had only one memory for instructions and data
  - Control hazards:
    - Need to worry about branch instructions
  - Data hazards:
    - An instruction depends on the result of preceding 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

The diagram illustrates the execution of instructions in a single-cycle CPU. The flow of instructions and data is as follows:

1. **Memory Read**: The instruction is read from memory and placed in the instruction register.
2. **Register Read**: The registers are read as required by the instruction.
3. **Memory Write**: Data is written to memory if the instruction requires.
4. **ALU Operations**: Arithmetic and logical operations are performed in the ALU.
5. **Shift Left**: Data is shifted left by 2 bits.
6. **Addressing**: Addressing modes like PC+4 or PC+8 are used to access memory.
7. **Result Store**: The result of the operation is stored in registers or memory.
8. **Branch Control**: Branch instructions are handled to determine the next instruction address.

The diagram highlights the control signals and data paths that enable the single-cycle operation of the CPU.
Review: Multi-cycle CPU
Review -- Instruction Latencies

- **Single-Cycle CPU**

- **Multiple Cycle CPU**

Load: Ifetch Reg/Dec Exec Mem Wr

Add: Ifetch Reg/Dec Exec Wr
Instruction Latencies and Throughput

**Single-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>
</tbody>
</table>

**Multiple Cycle CPU**

Cycle 1 Cycle 2 Cycle 3 Cycle 4 Cycle 5

| Load    | Ifetch  | Reg/Dec | Exec    | Mem     | Wr      |

**Pipelined CPU**

Cycle 1 Cycle 2 Cycle 3 Cycle 4 Cycle 5 Cycle 6 Cycle 7 Cycle 8

| Load    | Ifetch  | Reg/Dec | Exec    | Mem     | Wr      |
| Load    | Ifetch  | Reg/Dec | Exec    | Mem     | Wr      |
| Load    | Ifetch  | Reg/Dec | Exec    | Mem     | Wr      |
| Load    | Ifetch  | Reg/Dec | Exec    | Mem     | Wr      |
Pipeline Performance Considerations

- CPU throughput = Instructions Per Cycle (IPC)
  Number of instructions completed per cycle = 1/CPI

- Execution Time = (Instruction Count) * CPI * (Cycle Time)

- Complexity has a cost
  - e.g., Pipeline 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
What do we need to add to actually split the datapath into stages?
Pipelined Datapath

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

IF/ID  ID/EX  EX/MEM  MEM/WB

PC  Address  Instruction memory  Instruction

Add  4  Mux  0

Add result

Shift left 2

Add

ALU

Zero

MUX 1

MUX 1

MUX 0

Read register 1  Read register 2

Read data 1  Read data 2

Write register

Write data 16

Sign extend 32

Read data

Write data

Address

Data memory

Write data

Read data
LW Through the Pipeline
LW in EX Pipeline Stage
Problem: Destination Register
Corrected Pipelined Datapath

[Diagram of a pipelined datapath with instruction and data paths, illustrating the flow of data and control signals between different stages of the pipeline.]
Observations

- Instructions advance from one stage to another every clock
- Instructions and data move 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

- Shading indicates that the element is used by the instruction
  - e.g. ADD instruction does not use MEM
- Shading in left half means that the element is written in that stage
- Shading in the right half means that the element is read in that stage
Execution in a Pipelined Datapath

IF | ID | EX | MEM | WB
---|----|----|-----|----
IM | Reg | ALU | DM | Reg
IM | Reg | ALU | DM | Reg
IM | Reg | ALU | DM | Reg
IM | Reg | ALU | DM | Reg
IM | Reg | ALU | DM | Reg

CC1 | CC2 | CC3 | CC4 | CC5 | CC6 | CC7 | CC8 | CC9
---|----|----|----|-----|-----|-----|-----|-----
lw | IF | ID | EX | MEM | WB | steady state
lw | IF | ID | EX | MEM | WB
lw | IF | ID | EX | MEM | WB
lw | IF | ID | EX | MEM | WB
lw | IF | ID | EX | MEM | WB
lw | IF | ID | EX | MEM | WB
Mixed Instructions in the Pipeline

Resource conflict!
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
Pipeline with Control
Pipeline control

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

- Centralized control?
  - Too complicated?

- Better approach
  - Control embedded in the pipeline
  - Compute control information in decode stage
    - Pass it along through pipeline registers
Pipeline Control

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

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

Execution/Address Calculation stage control lines
Memory access stage control lines
Write-back stage control lines
Pipelined Datapath and Control
Would our pipeline design work in any case?

- What happens when...
  
  add $3, $10, $11
  lw $8, 1000($3)
  sub $11, $8, $7
Announcements

- **Reading Assignment**
  - Chapter 5. The Processor: Datapath and Control
    Sections 5.6
  - Chapter 6. Enhancing Performance with Pipelining
    Sections 6.1 - 6.10

- **Homework 5: Due Mon., July 25th in class**
  5.49, 5.50
  6.1, 6.2, 6.4, 6.6, 6.15, 6.17

- **Quiz**
  **When:** Mon., July 25th
  **Topic:** Pipelining and Hazards