CSE 141 – Computer Architecture
Summer Session I, 2004

Lectures 7
Exceptions and
Overview of Pipelining

Pramod V. Argade
CSE141: Introduction to Computer Architecture

Instructor: Pramod V. Argade (p2argade@cs.ucsd.edu)
Office Hours:
  Tue.  7:30 - 8:30 PM (AP&M 4141)
  Wed. 4:30 - 5:30 PM (AP&M 4141)

TA:
  Anjum Gupta (a3gupta@cs.ucsd.edu)
  Office Hour: Mon/Wed 12 - 1 PM
  Chengmo Yang (c5yang@cs.ucsd.edu)
  Office Hour: Mon/Thu 2 - 3 PM

Lecture: Mon/Wed. 6 - 8:50 PM, Center 109

Textbook: Computer Organization & Design
Authors: Patterson and Hennessy

Web-page: http://www.cse.ucsd.edu/classes/su04/cse141
Announcements

- **Reading Assignment:**
  - Exceptions, Section 5.6 (Monday)
  - Pipelining, Sections 6.1 - 6.3 (Tuesday)
  - Pipelining, Sections 6.4 - 6.7 (Wednesday)

- **Homework 5:** Due Mon., July 26 in class

  5.29

  Chapter 6 TBA

- **Quiz**
  
  **When:** Mon, July 26, First 10 minutes of the class
  **Topic:** Pipeline Hazards, Chapter 6
  **Need:** Paper, pen

- **Final Exam**
  
  **When:** Sat., July 31, 7 - 10 PM, Center 109
  **(Time and Room may change!)**
# CSE141 Course Schedule

<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. 6/28</td>
<td>6 - 8:50 PM</td>
<td>Center 109</td>
<td>Introduction, Ch. 1, ISA, Ch. 3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Wed. 6/30</td>
<td>6 - 8:50 PM</td>
<td>Center 109</td>
<td>Performance, Ch. 2, Arithmetic, Ch. 4</td>
<td>ISA Ch. 3</td>
<td>#1</td>
</tr>
<tr>
<td>-</td>
<td>Mon. 7/5</td>
<td>No Class</td>
<td></td>
<td>July 4th Holiday</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>Wed. 7/7</td>
<td>6 - 8:50 PM</td>
<td>Center 109</td>
<td>Arithmetic, Ch. 4 Cont. Single-cycle CPU Ch. 5</td>
<td>Performance Ch. 2</td>
<td>#2</td>
</tr>
<tr>
<td>4</td>
<td>Mon. 7/12</td>
<td>6 - 8:50 PM</td>
<td>Center 109</td>
<td>Single-cycle CPU Ch. 5 Cont. Multi-cycle CPU Ch. 5</td>
<td>Arithmetic, Ch. 4</td>
<td>#3</td>
</tr>
<tr>
<td>5</td>
<td>Tue. 7/13</td>
<td>7:30 - 8:50 PM</td>
<td>Center 109</td>
<td>Multi-cycle CPU Ch. 5 Cont. (July 5th make up class)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>Wed. 7/14</td>
<td>6 - 8:50 PM</td>
<td>Center 109</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. 7/19</td>
<td>6 - 8:50 PM</td>
<td>Center 109</td>
<td>Mid-term Exam Exceptions</td>
<td>-</td>
<td>#4</td>
</tr>
<tr>
<td>8</td>
<td>Tue. 7/20</td>
<td>7:30 - 8:50 PM</td>
<td>Center 109</td>
<td>Pipelining Ch. 6 (July 5th make up class)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>Wed. 7/21</td>
<td>6 - 8:50 PM</td>
<td>Center 109</td>
<td>Hazards, Ch. 6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>Mon. 7/26</td>
<td>6 - 8:50 PM</td>
<td>Center 109</td>
<td>Memory Hierarchy &amp; Caches Ch. 7</td>
<td>Hazards Ch. 6</td>
<td>#5</td>
</tr>
<tr>
<td>11</td>
<td>Wed. 7/28</td>
<td>6 - 8:50 PM</td>
<td>Center 109</td>
<td>Virtual Memory, Ch. 7 Course Review</td>
<td>Cache Ch. 7</td>
<td>#6</td>
</tr>
<tr>
<td>12</td>
<td>Sat. 7/31</td>
<td>7 - 10 PM</td>
<td>Center 109</td>
<td>Final Exam</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Exceptions

- There are two sources of non-sequential *control flow* in a processor
  - Explicit branch and jump instructions
  - Exceptions

- *Branches* are synchronous and deterministic
- *Exceptions* are typically asynchronous and non-deterministic

- Guess which is more difficult to handle?

*Control flow* refers to the movement of the program counter through memory
Exceptions and Interrupts

- The terminology is not consistent, but we’ll refer to
  - *Exceptions* as any unexpected change in control flow within the program
  - *Interrupts* as any externally-caused event

So then, what is:

- Arithmetic overflow
- Divide by zero
- I/O device signals completion to CPU
- User program invokes the OS
- Memory parity error
- Illegal instruction
- Timer signal
For now...

- The machine we’ve been designing in class can generate two types of exceptions.
  - Arithmetic overflow
  - Illegal instruction

- On an exception, we need to
  - Save the PC (invisible to user code)
  - Record the nature of the exception/interrupt
  - Transfer control to OS

- Must process exception transparently to the program!
Handling exceptions

- PC saved in EPC (exception program counter), which the OS may read and store in kernel memory
- A status *cause register*, and a single exception handler may be used to record the exception and transfer control, or
- A *vectored interrupt* transfers control to a different location for each possible type of interrupt/exception
Supporting exceptions

- For our MIPS-subset architecture, we will add two registers:
  - EPC: a 32-bit register to hold the user’s PC
  - Cause: A register to record the cause of the exception
    - We’ll assume undefined inst = 0, overflow = 1

- We will also add three control signals:
  - EPCWrite (will need to be able to subtract 4 from PC)
  - CauseWrite
  - IntCause

- We will extend PCSource multiplexor to be able to latch the interrupt handler address into the PC.
  - Interrupt handler address: 0xC000 0000
Exception Datapath
Supporting exceptions in our FSM

Instruction Fetch, state 0
- MemRead
- ALUSelA = 0
- IorD = 0
- IRWrite
- ALUSelB = 01
- ALUOp = 00
- PCWrite
- PCSource = 00

Instruction Decode/ Register Fetch, state 1
- ALUSelA = 0
- ALUSelB = 11
- ALUOp = 00

Opcodes:
- Opcode = LW or SW
- Opcode = R-type
- Opcode = BEQ
- Opcode = JMP
- Opcode = anything else

States:
- Memory Inst FSM
- R-type Inst FSM
- Branch Inst FSM
- Jump Inst FSM

Flow:
- Start to state 10
Supporting exceptions in our FSM

from state 1 
R-type instructions 

ALUSrcA = 1 
ALUSrcB = 00 
ALUOp = 10 

RegDst = 1 
RegWrite 
MemtoReg = 0 

overflow 
To state 11 

To state 0
States to Handle Exceptions

- **Illegal Instruction**
  - IntCause = 0
  - CauseWrite
  - ALUSrcA = 0
  - ALUSrcB = 01
  - ALUOp = 01
  - EPCWrite
  - PCWrite
  - PC++Source = 11

- **Arithmetic Overflow**
  - IntCause = 1
  - CauseWrite
  - ALUSrcA = 0
  - ALUSrcB = 01
  - ALUOp = 01
  - EPCWrite
  - PCWrite
  - PCSource = 11

- To state 0 to begin next instruction
FSM with Exceptions

Instruction fetch

Instruction decode/
Register fetch

Start

MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00

ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00

ALUSrcA = 1
ALUSrcB = 00
ALUOp = 00

ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10

ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01

ALUSrcA = 0
ALUSrcB = 01
ALUOp = 00

ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 01

ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11

IntCause = 1
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11

IntCause = 1
IntCause = 0

MemRead
IorD = 1
MemWrite
IorD = 1
RegWrite
RegWrite
MemtoReg = 1
MemtoReg = 1

R-type completion

Memory access

Memory access

Memory address computation

(Op = 'LW') or (Op = 'SW')

(Op = R-type)

Jump completion

(Op = 'J')

Start

Execution

Branch completion

(Op = 'BEQ')

Jump completion

(Op = 'J')

MemRead
IorD = 1
MemWrite
IorD = 1
RegWrite
RegWrite
MemtoReg = 0
MemtoReg = 0

RegDest = 1
RegDest = 0

MemtoReg = 1
RegDest = 0

Overflow

Write-back step

(Op = other)

EPCWrite
PCWrite
PCSource = 11

EPCWrite
PCWrite
PCSource = 11

0

Instruction fetch

Instruction decode/
Register fetch

Start

MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00

ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00

ALUSrcA = 1
ALUSrcB = 00
ALUOp = 00

ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10

ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01

ALUSrcA = 0
ALUSrcB = 01
ALUOp = 00

ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 01

ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11

IntCause = 1
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11

IntCause = 0
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11

IntCause = 1
IntCause = 0

MemRead
IorD = 1
MemWrite
IorD = 1
RegWrite
RegWrite
MemtoReg = 1
MemtoReg = 1

R-type completion

Memory access

Memory access

Memory address computation

(Op = 'LW') or (Op = 'SW')

(Op = R-type)

Jump completion

(Op = 'J')

Start

Execution

Branch completion

(Op = 'BEQ')

Jump completion

(Op = 'J')

MemRead
IorD = 1
MemWrite
IorD = 1
RegWrite
RegWrite
MemtoReg = 0
MemtoReg = 0

RegDest = 1
RegDest = 0

MemtoReg = 1
RegDest = 0

Overflow

Write-back step

(Op = other)
Overview of Pipelining
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 takes 6 hours for 4 loads
• If they learned pipelining, how long would laundry take?
Pipelined Laundry: Start work ASAP

- 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
- Pipeline efficiency (keeping the pipeline full) critical to performance
- Time between instructions\textsubscript{pipelined}
  \[= \frac{(\text{Time between instructions}\textsubscript{non-pipelined})}{(\# \text{ Pipe Stages})}\]
- 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></td>
<td>7 ns</td>
</tr>
<tr>
<td>R-Format</td>
<td>2 ns</td>
<td>1 ns</td>
<td>2 ns</td>
<td></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></td>
<td></td>
<td>5 ns</td>
</tr>
</tbody>
</table>

Program execution order (in instructions)

Non-Pipelined

Pipelined
Announcements

- Reading Assignment:
  - Exceptions, Section 5.6 (Monday)
  - Pipelining, Sections 6.1 - 6.3 (Tuesday)
  - Pipelining, Sections 6.4 - 6.7 (Wednesday)

- Homework 5: Due Mon., July 26 in class

5.29
Chapter 6 TBA

- Quiz
  **When:** Mon, July 26, First 10 minutes of the class
  **Topic:** Pipeline Hazards, Chapter 6
  **Need:** Paper, pen

- Final Exam
  **When:** Sat., July 31, 7 - 10 PM, Center 109
  (Time and Room may change!)