CSE 141 – Computer Architecture
Summer Session 1, 2005
Lectures 7

Exceptions
And
Introduction to Pipelining

Pramod V. Argade
July 18, 2005
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
## Summer Session I, 2005
### 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. 6/27</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>7:30 - 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 (More exercises to be assigned)

- **Quiz**
  When: Mon., July 25th  
  Topic: Pipelining and Hazards
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
  - *Interrupts* as any externally-caused exception

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
Handling exceptions

- PC saved in EPC (Exception Program Counter), which the OS may read and store in kernel memory

- Two ways of signaling
  - 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
    - Undefined inst: Cause = 0
    - Overflow: Cause = 1

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

- Need to force PC
  - Select the interrupt handler address into the PC.
Exception Datapath
Supporting exceptions in our FSM

Instruction Fetch, \textit{state 0}

- MemRead
- ALUSelA = 0
- IorD = 0
- IRWrite
- ALUSelB = 01
- ALUOp = 00
- PCWrite
- PCSource = 00

Start

Instruction Decode/ Register Fetch, \textit{state 1}

- ALUSelA = 0
- ALUSelB = 11
- ALUOp = 00

 Opcode = anything else

to state 10

- Opcode = \text{LW or SW}
- Opcode = \text{R-type}
- Opcode = \text{BEQ}
- Opcode = \text{JMP}

Memory Inst FSM

R-type Inst FSM

Branch Inst FSM

Jump Inst FSM

Opcode = \text{LW or SW}

Opcode = \text{R-type}

Opcode = \text{BEQ}

Opcode = \text{JMP}

Pramod V. Argade

CSE 141, Summer Session I, 2005
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
State to Support Exceptions

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

To state 0 (fetch)

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

interruption overflow

illegal instruction

11

10

11

sub 4

EPC

PC

PCSource

interrupt handler address

CauseWrite

IntCause

EPCWrite

PCWrite

CSE 141, Summer Session I, 2005
FSM with Exceptions

Instruction fetch

Instruction decode/ Register fetch

Start

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

0

ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00

1

ALUSrcA = 0
ALUSrcB = 00
ALUOp = 10

PCWriteCond
PCSource = 01

MemRead
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 00

Memory address computation

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

2

ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10

Execution

(Op = R-type)

3

MemRead
IorD = 1

MemWrite
IorD = 1

Memory access

MemWrite
IorD = 1

3

MemWrite
IorD = 1

Memory access

RegDst = 1
RegWrite
MemtoReg = 0

5

R-type completion

(Op = 'J')

Jump completion

6

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

7

ALUSrcA = 0
ALUSrcB = 00
ALUOp = 00

8

ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
PCWriteCond
PCSource = 01

ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00

9

MemWrite
IorD = 0

IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00

10

RegWrite
MemtoReg = 1
RegDst = 0

11

Write-back step

Overflow

Overflow

Overflow

RegWrite
MemtoReg = 1
RegDst = 0

4

Write-back step

Overflow

Overflow

Overflow

RegWrite
MemtoReg = 1
RegDst = 0

4

PCWrite
PCSource = 10

12

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

13

Instruction completion

Execute

Branch completion

2

5

9

13

Pramod V. Argade

CSE 141, Summer Session I, 2005

7-14
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

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
- Sequential laundry takes 6 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
  – Multiple tasks operate simultaneously
  – Pipelining does not change latency
  – 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 per stage
- Requires separate resources
- Achieves parallelism with replication
- Pipeline efficiency (keeping the pipeline full) critical to performance
- Time between instructions: \[ \text{Time between instructions}_{\text{pipelined}} = \frac{\text{Time between instructions}_{\text{non-pipelined}}}{(\# \text{ Pipe Stages})} \]
- Fundamentally invisible to the programmer
Pipelining: Instructions Implemented

- Only following instructions will be implemented
  - Memory: LW, SW
  - Arithmetic: ADD, SUB, AND, OR, SLT
  - Branch: BEQ

<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>200 ps</td>
<td>100 ps</td>
<td>200 ps</td>
<td>200 ps</td>
<td>100 ps</td>
<td>800 ps</td>
</tr>
<tr>
<td>Store Word (sw)</td>
<td>200 ps</td>
<td>100 ps</td>
<td>200 ps</td>
<td>200 ps</td>
<td></td>
<td>700 ps</td>
</tr>
<tr>
<td>R-Format</td>
<td>200 ps</td>
<td>100 ps</td>
<td>200 ps</td>
<td></td>
<td>100 ps</td>
<td>600 ps</td>
</tr>
<tr>
<td>Branch (beq)</td>
<td>200 ps</td>
<td>100 ps</td>
<td>200 ps</td>
<td></td>
<td></td>
<td>500 ps</td>
</tr>
</tbody>
</table>
Non-Pipelined vs. Pipelined Execution

Program execution order (in instructions)
- lw $1, 100($0)
- lw $2, 200($0)
- lw $3, 300($0)

Non-Pipelined: Time for 3 instructions 3 * 800 ps

Register file writes in the first half and reads in the second half of the cycle

Pipelined: Time for 3 instructions 3 * 200 ps

Pipelining offers 4x improvement in this example
Pipelining for MIPS

- All MIPS instructions are the same length.
  - Makes it easier to fetch them in the first pipeline stage
  - Decode stage has to always process 32 bits

- MIPS has only a few instruction formats
  - Various fields are in the same location in different formats
    - e.g. Opcode, Rs, Rt
  - Register file can be read in parallel with decoding

- Memory operands for MIPS appear only in load/store
  - ALU can be used to calculate memory address
  - Memory can be accessed in the following stage
Pipeline Hazards

- What are hazards?
  - Instruction cannot advance to the next stage in the following cycle
- Types of hazards
  - Structural hazards
    - Hardware does not support combination of instructions
  - Data hazards
    - One instruction must wait for another to complete
  - Control hazards
    - Need to make a decision based on the result of one instruction while others are executing
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 (More exercises to be assigned)

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