CSE 141 – Computer Architecture
Fall 2005
Lectures 11

Exceptions
And
Introduction to Pipelining

Pramod V. Argade
November 2, 2005
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:** Due Mon., November 14
  5.49, 5.50 (More exercises to be assigned)

- **Quiz**
  **When:** Mon., November 14th
  **Topic:** Pipelining and Hazards
## 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>9/26</td>
<td>Monday</td>
<td>Introduction, Ch. 1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>9/28</td>
<td>Wednesday</td>
<td>ISA, Ch. 2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>10/3</td>
<td>Monday</td>
<td>Arithmetic Part 1, Ch. 4</td>
<td>ISA</td>
<td>#1</td>
</tr>
<tr>
<td>4</td>
<td>10/5</td>
<td>Wednesday</td>
<td>Arithmetic Part 2, Ch. 4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>10/10</td>
<td>Monday</td>
<td>Performance, Ch. 3</td>
<td>Arithmetic</td>
<td>#2</td>
</tr>
<tr>
<td>6</td>
<td>10/12</td>
<td>Wednesday</td>
<td>Single cycle CPU, Ch. 5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>10/17</td>
<td>Monday</td>
<td>Single cycle CPU, Ch. 5</td>
<td>Performance</td>
<td>#3</td>
</tr>
<tr>
<td>8</td>
<td>10/19</td>
<td>Wednesday</td>
<td>Multi-cycle CPU, Ch. 5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>10/24</td>
<td>Monday</td>
<td>Multi-cycle CPU, Ch. 5</td>
<td>Single Cycle CPU</td>
<td>#4</td>
</tr>
<tr>
<td>10</td>
<td>10/26</td>
<td>Wednesday</td>
<td>Review for the Midterm</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>11/2</td>
<td>Wednesday</td>
<td>Exceptions, Ch. 5 and Pipelining, Ch. 6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>11/7</td>
<td>Monday</td>
<td>Pipelining, Ch. 6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>11/9</td>
<td>Wednesday</td>
<td>Data and control hazards, Ch. 6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>11/14</td>
<td>Monday</td>
<td>Data and control hazards, Ch. 6</td>
<td>Pipeline Hazards</td>
<td>#5</td>
</tr>
<tr>
<td>15</td>
<td>11/16</td>
<td>Wednesday</td>
<td>Memory &amp; cache design, Ch. 7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>11/21</td>
<td>Monday</td>
<td>Memory &amp; cache design, Ch. 7</td>
<td>Cache</td>
<td>#6</td>
</tr>
<tr>
<td>17</td>
<td>11/23</td>
<td>Wednesday</td>
<td>Virtual Memory &amp; cache design, Ch. 7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>18</td>
<td>11/28</td>
<td>Monday</td>
<td>Course Review</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>12/8</td>
<td>Thursday</td>
<td></td>
<td>Final Exam 7 - 10 PM</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
  - *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

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

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

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

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

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
State to Support Exceptions

illegal instruction

arithmetic overflow

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

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

To state 0 (fetch)

Interrupt Handler Address

sub 4

IntCause

CauseWrite

PCWrite

EPCWrite

PCWrite

EPC

PCSource

Cause

11

10
FSM with Exceptions
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
- 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 $t_{\text{pipelined}}$
  
  $t_{\text{pipelined}} = \frac{(\text{Time between instructions } t_{\text{non-pipelined}})}{\# \text{ Pipe Stages}}$

- 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>200 ps</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

Non-Pipelined: Time for 3 instructions $3 \times 800$ ps

Pipelined: Time for 3 instructions $3 \times 200$ ps

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

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: Due Mon., November 14**
  5.49, 5.50 (More exercises to be assigned)

● **Quiz**
  **When:** Mon., November 14th
  **Topic:** Pipelining and Hazards