CS G22.2233
Computer Systems Design
Spring 2007

Lecture 08: Midterm Review

Mohammad Banikazemi

[Slides from Prof. Mary Jane Irwin, PSU Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005, UCB]
Course Administration

- Midterm exam:
  - March 21th at 5:00pm in the same classroom
  - Open Book
  - No notes allowed

- Lab 1 – Part 1 Due: today at 5pm
  - Extended deadline tonight at midnight with no penalty
  - Sunday March 11 with **10-point penalty**
  - Wednesday March 14 with **20-point penalty**
  - No submission after that

- Lab 1 part 2 Due: March 28\(^{th}\) at 5pm. No extensions.
Introduction

- Note: This is just a collection of some of the topics covered in class. For complete coverage refer to class notes.

- Basic trends
- Instruction Set Architecture
- RISC Processors
### MIPS ISA Review

<table>
<thead>
<tr>
<th>Category</th>
<th>Instr</th>
<th>Op Code</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic (R &amp; I format)</td>
<td>add</td>
<td>0 and 32</td>
<td>add $s1, $s2, $s3</td>
<td>$s1 = $s2 + $s3</td>
</tr>
<tr>
<td></td>
<td>subtract</td>
<td>0 and 34</td>
<td>sub $s1, $s2, $s3</td>
<td>$s1 = $s2 - $s3</td>
</tr>
<tr>
<td></td>
<td>add immediate</td>
<td>8</td>
<td>addi $s1, $s2, 6</td>
<td>$s1 = $s2 + 6</td>
</tr>
<tr>
<td></td>
<td>or immediate</td>
<td>13</td>
<td>ori $s1, $s2, 6</td>
<td>$s1 = $s2 v 6</td>
</tr>
<tr>
<td>Data Transfer (I format)</td>
<td>load word</td>
<td>35</td>
<td>lw $s1, 24($s2)</td>
<td>$s1 = Memory($s2+24)</td>
</tr>
<tr>
<td></td>
<td>store word</td>
<td>43</td>
<td>sw $s1, 24($s2)</td>
<td>Memory($s2+24) = $s1</td>
</tr>
<tr>
<td></td>
<td>load byte</td>
<td>32</td>
<td>lb $s1, 25($s2)</td>
<td>$s1 = Memory($s2+25)</td>
</tr>
<tr>
<td></td>
<td>store byte</td>
<td>40</td>
<td>sb $s1, 25($s2)</td>
<td>Memory($s2+25) = $s1</td>
</tr>
<tr>
<td></td>
<td>load upper imm</td>
<td>15</td>
<td>lui $s1, 6</td>
<td>$s1 = 6 * 2^16</td>
</tr>
<tr>
<td>Cond. Branch (I &amp; R format)</td>
<td>br on equal</td>
<td>4</td>
<td>beq $s1, $s2, L</td>
<td>if ($s1===$s2) go to L</td>
</tr>
<tr>
<td></td>
<td>br on not equal</td>
<td>5</td>
<td>bne $s1, $s2, L</td>
<td>if ($s1 !=$s2) go to L</td>
</tr>
<tr>
<td></td>
<td>set on less than</td>
<td>0 and 42</td>
<td>slt $s1, $s2, $s3</td>
<td>if ($s2&lt;$s3) $s1=1 else $s1=0</td>
</tr>
<tr>
<td></td>
<td>set on less than</td>
<td>10</td>
<td>slti $s1, $s2, 6</td>
<td>if ($s2&lt;6) $s1=1 else $s1=0</td>
</tr>
<tr>
<td>Uncond. Jump (J &amp; R format)</td>
<td>jump</td>
<td>2</td>
<td>j 2500</td>
<td>go to 10000*</td>
</tr>
<tr>
<td></td>
<td>jump register</td>
<td>0 and 8</td>
<td>jr $t1</td>
<td>go to $t1</td>
</tr>
<tr>
<td></td>
<td>jump and link</td>
<td>3</td>
<td>jal 2500</td>
<td>go to 10000*; $ra=PC+4</td>
</tr>
</tbody>
</table>

Assuming 00 in MSB positions of PC
Review of MIPS Operand Addressing Modes

- **Register addressing** – operand is in a register

  \[
  \begin{array}{cccc}
  \text{op} & \text{rs} & \text{rt} & \text{rd} \\
  \end{array}
  \]

  \[
  \text{Register word operand}
  \]

- **Base (displacement) addressing** – operand is at the memory location whose address is the sum of a register and a 16-bit constant contained within the instruction

  \[
  \begin{array}{cccc}
  \text{op} & \text{rs} & \text{rt} & \text{offset} \\
  \end{array}
  \]

  \[
  \text{Memory word or byte operand}
  \]

  - Register relative (indirect) with \(0(a0)\)
  - Pseudo-direct with \(\text{addr}(\text{zero})\)

- **Immediate addressing** – operand is a 16-bit constant contained within the instruction

  \[
  \begin{array}{cccc}
  \text{op} & \text{rs} & \text{rt} & \text{operand} \\
  \end{array}
  \]
Review of MIPS Instruction Addressing Modes

- **PC-relative addressing** – instruction address is the sum of the PC and a 16-bit constant contained within the instruction.

  ![PC-relative addressing diagram]

- **Pseudo-direct addressing** – instruction address is the 26-bit constant contained within the instruction concatenated with the upper 4 bits of the PC.

  ![Pseudo-direct addressing diagram]
MIPS (RISC) Design Principles

- Simplicity favors regularity
  - fixed size instructions – 32-bits
  - small number of instruction formats
  - opcode always the first 6 bits

- Good design demands good compromises
  - three instruction formats

- Smaller is faster
  - limited instruction set
  - limited number of registers in register file
  - limited number of addressing modes

- Make the common case fast
  - arithmetic operands from the register file (load-store machine)
  - allow instructions to contain immediate operands
Numbers

- Binary representation
- MIPS arithmetic
- Add/subtract/Multiply/Divide
- Floating point numbers
- Floating point operations
IEEE 754 FP Standard Encoding

- Most (all?) computers these days conform to the IEEE 754 floating point standard: $(-1)^{\text{sign}} \times (1+F) \times 2^{E-\text{bias}}$
  
  - Formats for both single and double precision
  - $F$ is stored in normalized form where the msb in the fraction is 1 (so there is no need to store it!) – called the hidden bit
  - To simplify sorting FP numbers using integer comparisons
    - $E$ comes before $F$ in the word and
    - $E$ is represented in excess (biased) notation

<table>
<thead>
<tr>
<th>Single Precision</th>
<th>Double Precision</th>
<th>Object Represented</th>
</tr>
</thead>
<tbody>
<tr>
<td>E (8)</td>
<td>E (11)</td>
<td>true zero (0)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>± denormalized number</td>
</tr>
<tr>
<td>0</td>
<td>nonzero</td>
<td>± floating point number</td>
</tr>
<tr>
<td>1-254</td>
<td>1-2046</td>
<td>± infinity</td>
</tr>
<tr>
<td>255</td>
<td>2047</td>
<td>not a number (NaN)</td>
</tr>
<tr>
<td>255</td>
<td>nonzero</td>
<td></td>
</tr>
</tbody>
</table>
Summary: Evaluating ISAs

- **Design-time metrics:**
  - Can it be implemented, in how long, at what cost?
  - Can it be programmed? Ease of compilation?

- **Static Metrics:**
  - How many bytes does the program occupy in memory?

- **Dynamic Metrics:**
  - How many instructions are executed? How many bytes does the processor fetch to execute the program?
  - How many clocks are required per instruction?
  - How "lean" a clock is practical?

**Best Metric:** **Time to execute the program!**

depends on the instructions set, the processor organization, and compilation techniques.
MIPS Architecture

- Basic MIPS Architecture
- Components of MIPS Datapath
- Pipelining
- Pipeline Hazards
  - **structural hazards**: attempt to use the same resource by two different instructions at the same time
  - **data hazards**: attempt to use data before it is ready
    - An instruction’s source operand(s) are produced by a prior instruction still in the pipeline
  - **control hazards**: attempt to make a decision about program control flow before the condition has been evaluated and the new PC target address calculated
    - branch instructions
Pipelining Issues

- Forwarding
- Hazard Detection
- Stalling

Possible approaches for control hazards
- Stall (impacts CPI)
- Move decision point as early in the pipeline as possible, thereby reducing the number of stall cycles
- Delay decision (requires compiler support)
- Predict and hope for the best

- Two types of stalls
- Dealing with Exception
Multiple Issue Introduction

- Superpipelining

- Multiple-issue:
  - Super scalar
  - VLIW

- Instruction vs Machine Parallelism

- Multiple-issue types:
  - In-order issue with in-order completion
  - In-order issue with out-of-order completion
  - Out-of-order issue with out-of-order completion
  - Out-of-order issue with out-of-order completion and in-order commit

- True data dependency, Output dependency, Antidependency
Next Lecture and Reminders

- Next lecture
  - March 28th

- Reminders
  - Midterm exam: March 21th @ 5:00pm
  - Lab 1 due today