CSE 141 – Computer Architecture
Fall 2003

Lectures 17
Course Review

Pramod V. Argade

CSE141: Introduction to Computer Architecture

Web-page:  http://www-cse.ucsd.edu/classes/fa03/cse141

Special Discussion Sections:
Chris Roedel:
   When?   Sunday, December 7 2003, 5:00 - 6:50 PM
   Where?  Center 212 (Regular Classroom)

Anjum Gupta and Eric Liu:
   When?   Monday, December 8 2003, 12:00 - 1:50 PM
   Where?  Center 214

Final Exam: Tuesday, December 9, 2003, 8:00 - 11:00am.
Room:       Price Center Theatre
Schedule

<table>
<thead>
<tr>
<th>Lecture #</th>
<th>Date</th>
<th>Day</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sep. 25</td>
<td>Thursday</td>
<td>Introduction, Ch. 1</td>
</tr>
<tr>
<td>2</td>
<td>Sep. 30</td>
<td>Tuesday</td>
<td>Performance, Ch. 2</td>
</tr>
<tr>
<td>3</td>
<td>Oct. 2</td>
<td>Thursday</td>
<td>ISA, Ch. 3</td>
</tr>
<tr>
<td>4</td>
<td>Oct. 7</td>
<td>Tuesday</td>
<td>Arithmetic, Ch. 4</td>
</tr>
<tr>
<td>5</td>
<td>Oct. 9</td>
<td>Thursday</td>
<td>Arithmetic, Ch. 4, Continued</td>
</tr>
<tr>
<td>6</td>
<td>Oct. 14</td>
<td>Tuesday</td>
<td>Single-cycle CPU, Ch. 5</td>
</tr>
<tr>
<td>7</td>
<td>Oct. 18</td>
<td>Thursday</td>
<td>Single-cycle CPU, Ch. 5</td>
</tr>
<tr>
<td>8</td>
<td>Oct. 21</td>
<td>Tuesday</td>
<td>Multi-cycle CPU, Ch. 5</td>
</tr>
<tr>
<td>9</td>
<td>Oct. 23</td>
<td>Thursday</td>
<td>Multi-cycle CPU, Ch. 5</td>
</tr>
<tr>
<td>10</td>
<td>Oct. 28</td>
<td>Tuesday</td>
<td>Classes cancelled due to wildfires</td>
</tr>
<tr>
<td>11</td>
<td>Oct. 30</td>
<td>Thursday</td>
<td>Exceptions and Review for Midterm</td>
</tr>
<tr>
<td>12</td>
<td>Nov. 4</td>
<td>Tuesday</td>
<td>Mid-term Exam</td>
</tr>
<tr>
<td>13</td>
<td>Nov. 6</td>
<td>Thursday</td>
<td>Pipelining, Ch. 6</td>
</tr>
<tr>
<td>No Class</td>
<td>Nov. 11</td>
<td>Tuesday</td>
<td>Veteran's Day Holiday</td>
</tr>
<tr>
<td>14</td>
<td>Nov. 13</td>
<td>Thursday</td>
<td>Data and Control hazards, Ch. 6</td>
</tr>
<tr>
<td>15</td>
<td>Nov. 18</td>
<td>Tuesday</td>
<td>Control hazards, Ch. 6</td>
</tr>
<tr>
<td>16</td>
<td>Nov. 20</td>
<td>Thursday</td>
<td>Memory and Cache Design Ch. 7</td>
</tr>
<tr>
<td>17</td>
<td>Nov. 25</td>
<td>Tuesday</td>
<td>Memory and Cache Design Ch. 7</td>
</tr>
<tr>
<td>No Class</td>
<td>Nov. 27</td>
<td>Thursday</td>
<td>Thanksgiving Holiday</td>
</tr>
<tr>
<td>18</td>
<td>Dec. 2</td>
<td>Tuesday</td>
<td>Virtual Memory, Ch. 7</td>
</tr>
<tr>
<td>19</td>
<td>Dec. 4</td>
<td>Thursday</td>
<td>Course Review</td>
</tr>
<tr>
<td>Dec. 9</td>
<td>Tuesday</td>
<td></td>
<td>Final Exam</td>
</tr>
</tbody>
</table>

What is a Process?

- Program state consists of:
  - Page tables, PC and the registers
- This state is referred to as a process
- Process is an instance of a program executing on a CPU
Implementing Protection with VM

- Protection is essential for:
  - Allowing a single main memory to be shared among multiple processes
  - Preventing one process from writing into the memory space of another
  - Preventing a user process from modifying its own page tables
  - Controlling access to peripheral devices

- Hardware capabilities needed for protection
  - Two operating modes: user mode and kernel mode of execution
  - A portion of the CPU state that a user process can read, but not write
    - This is the user/kernel mode bit in processor status word
  - A mechanism to switch between user mode and kernel mode
    - Accomplished by a system call

Additional bits in the Page Table

- User or Kernel bit
  - This bit restricts access to some pages to kernel only

- Write bit
  - This bit grants read-only or read/write access to a page

- Referenced bit
  - OS periodically sets this bit to zero
  - It is set by CPU hardware when the page is referenced
  - Used by OS for replacing the page with other memory pages

- Dirty bit
  - If a process writes to a page, the dirty bit is set
  - It is used by OS to write the page to secondary storage before replacing it
Virtual Memory Key Points

- How does virtual memory provide:
  - illusion of large main memory?
  - sharing?
  - performance?
  - protection?

- Virtual Memory requires twice as many memory accesses, so we cache page table entries in the TLB.

- Three things can go wrong on a memory access: cache miss, TLB miss, page fault.

The five classic components of computers
The Instruction Execution Cycle

- **Instruction Fetch**
  - Obtain instruction from program storage

- **Instruction Decode**
  - Determine required actions and instruction size

- **Operand Fetch**
  - Locate and obtain operand data

- **Execute**
  - Compute result value or status

- **Result Store**
  - Deposit results in storage for later use

- **Next Instruction**
  - Determine successor instruction

Performance

- **Performance**
  - Execution Time = (Instruction Count) * CPI * (Cycle Time)
  - Clock rate is in cycles per second
    - MHz (Millions of cycles per second)
    - GHz (Billions of cycles per second)
  - Cycle time = 1/(Clock Rate)
  - Speedup = (exe time without change / exe time with change)

\[
\text{Relative Performance} = \frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = n
\]

- **Amdahl’s Law**
  - Execution time after improvement = \[\frac{\text{Execution Time Affected}}{\text{Amount of Improvement}} + \text{Execution Time Unaffected}\]
ISA

- Instruction Length
  - Variable
  - Fixed
  - MIPS has fixed 32 bit length

- Basic ISA Types
  - Load-store
  - Reg-Mem
  - Stack
  - Accumulator

Overview of MIPS ISA

- 3-operand, load-store architecture
- 32 general-purpose registers
  - R0 always equals 0.
- 2 special-purpose integer registers, HI and LO, because multiply and divide produce more than 32 bits.
- Registers are 32-bits wide (word size is 4 bytes)
- Register, immediate, base+displacement, PC-relative and pseudo-direct addressing modes
- Fixed-length 32-bit instructions
- 3 instruction formats
The MIPS Instruction Formats

- All MIPS instructions are 32 bits long.

<table>
<thead>
<tr>
<th>Instruction Format</th>
<th>R-type</th>
<th>I-type</th>
<th>J-type</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>6 bits</td>
<td>6 bits</td>
<td>6 bits</td>
</tr>
<tr>
<td>rs</td>
<td>5 bits</td>
<td>6 bits</td>
<td>5 bits</td>
</tr>
<tr>
<td>rt</td>
<td>5 bits</td>
<td>6 bits</td>
<td>16 bits</td>
</tr>
<tr>
<td>rd</td>
<td>5 bits</td>
<td>6 bits</td>
<td>16 bits</td>
</tr>
<tr>
<td>shamt</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
</tr>
<tr>
<td>funct</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
</tr>
</tbody>
</table>

MIPS Instruction Summary

### MIPS Operands

<table>
<thead>
<tr>
<th>Name</th>
<th>Example</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 registers</td>
<td>$s0-$s7, $t0-$t9, $zero,</td>
<td>Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register $zero always equals 0.Register $at is reserved for the assembler to handle large constants.</td>
</tr>
<tr>
<td>2nd memory words</td>
<td>Memory[0], ... Memory[4],</td>
<td>Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.</td>
</tr>
<tr>
<td>Memory[4294967292]</td>
<td>Memory[4294967292]</td>
<td></td>
</tr>
</tbody>
</table>

### MIPS Assembly Language

<table>
<thead>
<tr>
<th>Category</th>
<th>Instruction</th>
<th>Example</th>
<th>Meaning</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>add</td>
<td>$s1, $s2, $s3</td>
<td>$s1 = $s2 + $s3</td>
<td>Three operands; data in registers.</td>
</tr>
<tr>
<td></td>
<td>add immediate</td>
<td>$s1, $s2, 100</td>
<td>$s1 = $s2 + 100</td>
<td>Used to add constants.</td>
</tr>
<tr>
<td></td>
<td>sub</td>
<td>$s1, $s2, $s3</td>
<td>$s1 = $s2 - $s3</td>
<td>Three operands; data in registers.</td>
</tr>
<tr>
<td></td>
<td>sub immediate</td>
<td>$s1, 100</td>
<td>$s1 = 100 - $s2</td>
<td></td>
</tr>
<tr>
<td>Data transfer</td>
<td>load word</td>
<td>lw</td>
<td>$s1, 100($s2)</td>
<td>Word from memory to register.</td>
</tr>
<tr>
<td></td>
<td>store word</td>
<td>sw</td>
<td>Memory[100($s2)] = $s1</td>
<td>Word from register to memory.</td>
</tr>
<tr>
<td></td>
<td>load upper immediate</td>
<td>lui</td>
<td>$s1, 100</td>
<td>$s1 = 100 * 2^16</td>
</tr>
<tr>
<td></td>
<td>store immediate</td>
<td>sb</td>
<td>Memory[100($s2)] = $s1</td>
<td>Stores immediate.</td>
</tr>
<tr>
<td>Conditional branch</td>
<td>branch on equal</td>
<td>beq</td>
<td>$s1, $s2, 25</td>
<td>if ($s1 == $s2) go to PC + 4 + 100</td>
</tr>
<tr>
<td></td>
<td>branch on not equal</td>
<td>bne</td>
<td>$s1, $s2, 25</td>
<td>if ($s1 != $s2) go to PC + 4 + 100</td>
</tr>
<tr>
<td></td>
<td>set less than</td>
<td>slt</td>
<td>$s1, $s2, $s3</td>
<td>if ($s2 &lt; $s3) $s1 = 1; else $s1 = 0</td>
</tr>
<tr>
<td></td>
<td>set less than immediate</td>
<td>slti</td>
<td>$s1, $s2, 100</td>
<td>if ($s2 &lt; 100) $s1 = 1; else $s1 = 0</td>
</tr>
<tr>
<td>Unconditional jump</td>
<td>jump</td>
<td>j</td>
<td>2000</td>
<td>go to 10000</td>
</tr>
<tr>
<td></td>
<td>jump register</td>
<td>jr</td>
<td>$ra</td>
<td>go to $ra</td>
</tr>
<tr>
<td></td>
<td>jump and link</td>
<td>jal</td>
<td>2000</td>
<td>go to PC + 4; go to 10000</td>
</tr>
</tbody>
</table>
Arithmetic

- Decimal, binary and hex representation
- Two’s Complement
  - 2’s complement representation of negative numbers
    - Take the bitwise inverse and add 1
- Ripple carry adder
  - Worst case delay for a N-bit adder: 2N-gate delay
- Carry Lookahead adder
  - Generate Carry at Bit i: $g_i = A_i \& B_i$
  - Propagate Carry via Bit i: $p_i = A_i \lor B_i$
  - 2 gate delay to calculate the carry in bits
    - $C_{in1} = g_0 \lor (p_0 \& C_{in0})$
    - $C_{in2} = g_1 \lor (p_1 \& g_0) \lor (p_1 \& p_0 \& C_{in0})$
    - $C_{in3} = g_2 \lor (p_2 \& g_1) \lor (p_2 \& p_1 \& g_0) \lor (p_2 \& p_1 \& p_0 \& C_{in0})$
  - Worst case 5 gate delays
- Overflow flag: $C_{O_{MSB}} \lor C_{I_{MSB}}$

Booth’s algorithm: Signed multiplication

<table>
<thead>
<tr>
<th>Current Bit</th>
<th>Bit to the Right</th>
<th>Explanation</th>
<th>Example</th>
<th>Op</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>Begins run of 1s</td>
<td>0001111100</td>
<td>sub</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Middle of run of 1s</td>
<td>0001111000</td>
<td>none</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>End of run of 1s</td>
<td>0001111000</td>
<td>add</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Middle of run of 0s</td>
<td>0001111000</td>
<td>none</td>
</tr>
</tbody>
</table>

Originally for Speed (when shift was faster than add)
- Replace a string of 1s in multiplier with an initial subtract when we first see a one and then later add for the bit after the last one
- Potential speed up recognizing that string of 0’s and 1’s requires no operation!
IEEE Floating Point Standard
Single Precision Floating Point Representation

- **Example:**
  - decimal: \(-0.75 = -3/4 = -3/2^2\)
  - binary: \(-0.11 = -1.1 \times 2^{-1}\)
  - floating point: exponent = 126 = 01111110
  - IEEE single precision: 10111111010000000000000000000000

edge-triggered Clocking

- Values stored in the machine are updated on a clock edge
  - The clock edge can be either rising or falling

- By default a state element is written every clock edge
  - An explicit write control signal is required otherwise.

- Edge triggered methodology allows, in the same clock cycle:
  - read the contents of a register
  - send the value through some combinational logic, and
  - write the contents

- Possible to have the same state element as input and output
CPU Implementations

• **Single-Cycle CPU**

  Load
  Ifetch Reg/Dec Exec Mem Wr

• **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

Single-cycle Datapath and Control
Multi-cycle Datapath and Control

Multicycle CPU: Control

\[
\text{ControlOutput} = f(\text{State, OpCode}) \\
\text{NextState} = f(\text{State, OpCode})
\]
Implementing a New Instruction

- Analyze the instruction
  - What are the operands?
  - What result(s) does the instruction produce?
    - What is (are) the destination register(s)?
  - What operation is performed on the operands?
- Can you use existing datapath?
  - If not, add the required datapath element(s)
- Make changes to the control logic, if any
- For a multi-cycle datapath:
  - Write a series of steps to execute the instruction
  - Include any additional registers for temporary storage
  - Write the state machine with values for all control signals
- Make sure that the other instructions are not affected
Pipelined CPU

Example

Assuming M⇒M Forwarding for LW⇒SW

<table>
<thead>
<tr>
<th>Instruction</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
<th>C7</th>
<th>C8</th>
<th>C9</th>
<th>C10</th>
<th>C11</th>
<th>C12</th>
<th>C13</th>
<th>C14</th>
<th>C15</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD R1, R2, R3</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SW R1, 1000(R2)</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LW R7, 2000(R2)</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD R5, R7, R1</td>
<td>F</td>
<td>D</td>
<td>D</td>
<td>E</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LW R8, 2004(R2)</td>
<td>F</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SW R7, 2008(R8)</td>
<td>F</td>
<td>D</td>
<td>D</td>
<td>E</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD R8, R8, R2</td>
<td>F</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LW R9, 1012(R8)</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SW R9, 1016(R8)</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cache Organization

- A typical cache has three dimensions

Bytes/block (block size)

Blocks/set (associativity)

Number of sets (cache size)

tag  |  index  |  block offset

A Set-associative cache

Total 1K Blocks

Address

Index

0

1

2

255

254

253

V  Tag  Data

V  Tag  Data

V  Tag  Data

V  Tag  Data

4-to-1 multiplexer
Address Translation via the Page Table

- Page offset
- Virtual page number
- Virtual address
- Physical page number
- Physical address
- Valid
- Page table

Notes:
- The page table contains mapping for every possible virtual page
- Valid bit indicates whether the page is present in the main memory
- Extra bits in the page table are used for protection information

TLB: Making Address Translation Fast

Translation Lookaside Buffer: A cache for address translations

- Virtual page number
- Valid
- Tag
- Physical page address
- Physical memory
- Disk storage
TLB and Cache

Virtual page number | Page offset
--------------------|------------------

Valid | Tag | Physical page number
---|-----|---------------------

TLB

Physical page number | Page offset
---------------------|------------------

Physical address tag | Cache index

Cache

Valid | Tag | Data
---|-----|------

Cache hit

Physical address

11 10 9 8 7 6 5 4 3 2 1 0

CSE141: Introduction to Computer Architecture

Web-page:  http://www-cse.ucsd.edu/classes/fa03/cse141

Special Discussion Sections:

Chris Roedel:
When? Sunday, December 7 2003, 5:00 - 6:50 PM
Where? Center 212 (Regular Classroom)

Anjum Gupta and Eric Liu:
When? Monday, December 8 2003, 12:00 - 1:50 PM
Where? Center 214

Final Exam: Tuesday, December 9, 2003, 8:00 - 11:00am.
Room: Price Center Theatre