Lectures 10  
Mid-term Test Review

Pramod V. Argade  
April 27, 2005
Announcements

- Homework: None!
- Extra Office Hour Monday
- Midterm
  When: Mon., May 2nd
  Where: Center Room 101
  Need: Paper, pen, calculator (only for calculations, not for notes!)
  Topics:
  - Chapters 1 - 5
  - All the material covered in lectures
  - All topics covered in homework’s

The five classic components of computers

- Computer
- Control
- Memory
- Datapath
- Input
- Output
Performance

- Execution Time = (Instruction Count) * CPI * (Cycle Time)
- Clock rate is in cycles per second
  - MHz (Millions of cycles per second, $10^6$ Hz)
  - GHz (Billions of cycles per second, $10^9$ Hz)
- Cycle time = $1$/ (Clock Rate)
- Speedup = (exe time without change / exe time with change)

Relative Performance = \( \frac{\text{Performance}_X}{\text{Performance}_Y} \) = \( \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} \) = \( n \)

Amdahl’s Law

\[
\frac{\text{Execution time after improvement}}{\text{Execution Time Affected}} = \frac{\text{Amount of Improvement}}{\text{Execution Time Unaffected}}
\]

ISA

- Instruction Length
  - Variable
  - Fixed
  - Hybrid
  - 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 (integer, floating point)
  - R0 always equals 0
- 2 special-purpose integer registers, HI and LO, because multiply and divide produce more than 32 bits
- Fixed 32-bit instructions
- 3 instruction formats
- Registers are 32-bits wide (word)
- Register, immediate, base+displacement, PC-relative and pseudo-direct addressing modes

MIPS Addressing Modes

1. Immediate addressing e.g. addi $t0, $t1, 4
2. Register addressing e.g. sub $t0, $t1, $t2
3. Base addressing e.g. lw $t0, 4( $t2)
4. PC-relative addressing e.g. beq $t1, $t2, 32
5. Pseudodirect addressing e.g. j 0x1000
The MIPS Instruction Formats

- All MIPS instructions are 32 bits long.

### R-type

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>6 bits</td>
</tr>
</tbody>
</table>

### I-type

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

### J-type

<table>
<thead>
<tr>
<th>op</th>
<th>target address</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>26 bits</td>
</tr>
</tbody>
</table>

### MIPS operands

- **Category**: Instruction Example Comments
- **Arithmetic**
  - add $s1, $s2, $s3 $s1 = $s2 + $s3
  - subtract $s1, $s2, $s3 $s1 = $s2 - $s3
  - add immediate $s1, $s2, 100 $s1 = $s2 + 100
  - load word $s1, 100($s2) $s1 = Memory[$s2 + 100]
  - store word $s1, 100($s2) Memory[$s2 + 100] = $s1
  - load upper immediate $s1, 100 $s1 = 100 * 2
  - branch on equal $s1, $s2, 25 if ($s1 == $s2) go to PC + 4 + 100
  - branch on not equal $s1, $s2, 25 if ($s1 != $s2) go to PC + 4 + 100
  - set on less than $s1, $s2, $s3 if ($s2 < $s3) $s1 = 1; else $s1 = 0
  - set less than immediate $s1, $s2, 100 if ($s2 < 100) $s1 = 1; else $s1 = 0
  - jump $s1, 2500 go to 10000
  - unconditional jump $s1, 2500 go to PC + 4
  - jump and link $s1, 2500 $ra = PC + 4; go to 10000

- **Data transfer**
  - load $s1, $s2, $s3 $s1 = $s2 + $s3
  - load immediate $s1, $s2, 100 $s1 = $s2 + 100
  - data from memory to register $s1 = Memory + 100
  - data from register to memory $s1 = Memory + 100
  - data constant in upper 16 bits $s1 = 100 * 2
  - equal less PC-relative branch $s1, $s2, 100 $s1 = Memory + 100

- **Conditional branch**
  - if $s1 < $s2, $s3 $s1 = $s2 + $s3
  - if $s1 == $s2, $s3 $s1 = $s2 + $s3
  - if $s1 < 100, $s2, $s3 $s1 = $s2 + $s3
  - if $s1 == PC + 4, $s2, $s3 $s1 = $s2 + $s3

To summarize:

- MIPS operands
  - **Register**
    - 32 registers
    - Fast locations for data; In MIPS data must be in registers to perform arithmetic.
    - MIPS register files always equal 0. Register 0 is reserved for the assembler to handle large constants.
  - **Memory**
    - Accessed only by data transfer instructions. MIPS uses 32-bit addresses, so sequential words differ by 4.
    - Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.

- MIPS assembly language
  - **Category**: Instruction Example Meaning Comments
  - **Arithmetic**
    - add $s1, $s2, $s3 $s1 = $s2 + $s3
    - subtract $s1, $s2, $s3 $s1 = $s2 - $s3
    - add immediate $s1, $s2, 100 $s1 = $s2 + 100
    - load word $s1, 100($s2) $s1 = Memory[$s2 + 100]
    - store word $s1, 100($s2) Memory[$s2 + 100] = $s1
    - load upper immediate $s1, 100 $s1 = 100 * 2
    - branch on equal $s1, $s2, 25 if ($s1 == $s2) go to PC + 4 + 100
    - branch on not equal $s1, $s2, 25 if ($s1 != $s2) go to PC + 4 + 100
    - set on less than $s1, $s2, $s3 if ($s2 < $s3) $s1 = 1; else $s1 = 0
    - set less than immediate $s1, $s2, 100 if ($s2 < 100) $s1 = 1; else $s1 = 0
    - jump $s1, 2500 go to 10000
    - unconditional jump $s1, 2500 go to PC + 4
    - jump and link $s1, 2500 $ra = PC + 4; go to 10000

Pramod Argade CSE 141, Spring 2005
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: \( g_i = A_i \land B_i \)
  - Propagate Carry via Bit i: \( p_i = A_i \lor B_i \)
  - 2 gate delay to calculate the carry in bits
    - \( C_{i+1} = g_0 \land (p_0 \land C_{i-1}) \)
    - \( C_{i+1} = g_1 \land (p_1 \land g_0) \land (p_1 \land p_0 \land C_{i-1}) \)
    - \( C_{i+1} = g_2 \land (p_2 \land g_1) \land (p_2 \land p_1 \land g_0) \land (p_2 \land p_1 \land p_0 \land C_{i-1}) \)
  - Worst case 5 gate delays
- Overflow flag: \( C_0^{MSB} \land C_1^{MSB} \)

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>0001111 1000</td>
<td>sub</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Middle of run of 1s</td>
<td>0001111 1000</td>
<td>none</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>End of run of 1s</td>
<td>0001111 1000</td>
<td>add</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Middle of run of 0s</td>
<td>0001111 1000</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

- Example:
  - Convert decimal: -0.75 to IEEE Single Precision Floating Point

Single Cycle CPU
Multi-cycle CPU

Complete FSM
Problem 1: Single-cycle CPU
Implementation of Loop Instruction

Take the Single Cycle datapath posted on the web, and modify it so that it can also execute the instruction

```
loop r1, r2, offset
```

This is a branch instruction that increments register `r1`, and compares it to a loop bound of `r2`. If these two values are not equal then the PC is set to PC + offset. The instruction uses the Immediate MIPS instruction format. This instruction has the same effect as sequentially executing the following two instructions on the MIPS architecture:

```
addi r1, r1, 1
bne r1, r2, offset
```

For this question do NOT modify the instruction memory, data memory, or the register file. You can only modify/add control lines, MUXs, ALUs, and data path lines. To answer this question (1) Give format for the loop instruction and clearly specify each field, (2) Draw the parts of the datapath that have changed, and (3) Specify the control for this instruction in the modified datapath. Show the values for all the control lines and any MUX in the original Single Cycle Datapath along with any new control lines added for this problem.
Problem 2: Multi-cycle CPU
Implementation of MemIndAdd Instruction

Modify multi-cycle datapath figure posted on the web so that it can also execute the instruction
MemIndAdd r1, offset(r2)

This instruction calculates the address M[offset + r2], which is a pointer back into memory. It then
loads the value stored at address M[offset + r2], and adds it to r1, storing the result back into r1.
The instruction uses the Immediate MIPS instruction format. This instruction has the same
effect as sequentially executing the following code:

tmp = Memory[offset + r2]
tmp = Memory[tmp]
r1 = r1 + tmp

For this question, do not add any new ALUs, do not modify the instruction and data memory, and
do not modify the register file. In addition, do not add any registers to the register file or
temporary registers to the data path. You can only modify/add data paths, control lines, and
MUXs. To answer this question give

- How would you encode the instruction in I-format?
- Modify the multiple cycle processor to execute the new instruction, and
- Provide the finite state machine (FSM) for the control for this datapath. For the finite state
  machine show the values for the control lines and all MUXs in the figure. For the ALU control
  line, just give the ALU operation for each cycle. Show the important control lines and MUX
  values for every cycle (starting with the fetch cycle). In addition, show how many cycles it
takes to execute the MemIndAdd instruction with your FSM.

Pramod Argade
CSE 141, Spring 2005
Problem 3: MIPS Assembly Code

- Comment the following MIPS assembly code and describe what is going on in the code and then convert it to the matching C code. $s0 contains a pointer to a data memory location. Describe what is stored in memory. (A picture might help)

```
sw $zero, 0($s0)
addi $t0, $zero, 1
addi $s0, $s0, 4
sw $t0, 0($s0)
loop: addi $t0, $t0, 1
    slti $t1, $t0, 10
    be $t1, $zero, end
addi $s0, $s0, 4
lw $t2, -4($s0)
lw $t3, -8($s0)
add $t4, $t2, $t3
sw $t4, 0($s0)
j loop
end: ...
```

Problem 4: Booth’s Algorithm

- What is the result of $-7 \times -5$ using Booth’s algorithm? Assume 4 bit 2’s complement representation for the multiplicand and multiplier. Clearly show all the steps.