10/26/2004 Comp 120 Fall 2004 1

26 October
• Only 11 to go!
• Questions?
• Today Exam Review and Instruction Execution

10/26/2004 Comp 120 Fall 2004 2

[10] How many bits are required to store a 10-digit number (i.e. a phone number that might range from 0000000000 to 9999999999) as a binary integer? As ASCII characters?

\[ 10^{10} = 10 \times 10^9 = 10 \times 2^{30} < 2^{34} = 34 \text{ bits} \]

10 characters \times 8 \text{ bits/character} = 80 \text{ bits}

10/26/2004 Comp 120 Fall 2004 3

[5] Suppose the byte at location 1000 in memory contains the hex value 0x87. What is the content of register $S0$ after the instruction "lb $0,1000($)zero"? What is its content after the instruction "lbu $0,1000($)zero"? We're comparing the "load byte" instruction to the "load byte unsigned" instruction.

The lb instruction will "sign extend" so the result will be 0xffffffff87 while the lbu will produce 0x00000087

10/26/2004 Comp 120 Fall 2004 4

[10] Which of the following cannot be EXACTLY represented by an IEEE double-precision floating-point number?
(a) 0
(b) 10.2
(c) 1.625
(d) 11.5

10/26/2004 Comp 120 Fall 2004 5

[12] All of the following equations are true for the idealized numbers you studied in algebra. Which ones are true for IEEE floating point numbers? Assume that all of the numbers are well within the range of largest and smallest possible numbers (that is, underflow and overflow are not a problem)

- \( A \times B = A \) if and only if \( B = 0 \)
- \( (A \times B) \times C = A \times (B \times C) \)
- \( A \times B = B \times A \)
- \( A \times (B + C) = A \times B + A \times C \)

10/26/2004 Comp 120 Fall 2004 6

[12] Consider the characteristics of two machines M1 and M2. M1 has a clock rate of 1GHz. M2 has a clock rate of 2GHz. There are 4 classes of instructions (A-D) in the instruction set. In a set of benchmark programs, the frequency of each class of instructions is shown in the table.

<table>
<thead>
<tr>
<th>Instruction Class</th>
<th>Frequency</th>
<th>M1 CPI</th>
<th>M2 CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>40%</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>20%</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>20%</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>10%</td>
<td>5</td>
<td>8</td>
</tr>
</tbody>
</table>

What is the average CPI for each machine? Which machine is faster? By what factor faster is it? What is the cycle time of each machine?

CPI1 = 2\( \times 0.4\) + 3\( \times 0.25\) + 3\( \times 0.2\) + 5\( \times 0.15\) = 2.9

CPI2 = 6\( \times 0.4\) + 6\( \times 0.25\) + 6\( \times 0.2\) + 8\( \times 0.15\) = 6.3

M1 is faster by \( (1000/2.9) / (2000/6.3) = 1.086 \)

M1 cycle time = 1ns, M2 cycle time = 0.5ns
6

[10] How can the ALU we talked about in class be used to compare two values for equality?

Subtract (invert, add, with Cin=1) and OR the 32 bits of the result together with an additional gate. If the output of that gate is 9; then the values are equal.

7

[5] Using 2’s complement arithmetic, negate 0x1234ffff. What is its value in hex?

0x1234ffff inverted = 0xedcb0000 plus 1 = 0xedcb0001

8

[10] Show, step by step, how to add -17 (negative 17) and 20 (positive 20) using 2’s complement arithmetic and 8 bit integers.

17 is 0x11, so ~17 is 0xeed, so -17 is 0xef = 11101111
20 is 0x14 = 00010100
Add them bit-by-bit to get 00000011 = 0x03

9

[6] When multiplying unsigned numbers that are N bits long using the shift and add algorithm, how many additions are required on average if all multipliers are equally likely?

N/2

Are all multipliers equally likely?

10

[15] Draw a block diagram showing how to implement 2’s complement subtraction of 4 bit numbers (A minus B) using 1-bit full-adder blocks with inputs A, B, and Cin and outputs Sum and Cout and any other AND, OR, or INVERT blocks you may need.
**Instruction Execution**

**Five Execution Steps**
- Instruction Fetch
- Instruction Decode and Register Fetch
- Execution, Memory Address Computation, or Branch Completion
- Memory Access or R-type instruction completion
- Memory Read Completion

INSTRUCTIONS TAKE FROM 3 - 5 CYCLES!
- A FSM looks at the op-code to determine how many...

**Step 1: Instruction Fetch**
- Use PC to get instruction and put it in the Instruction Register.
- Increment the PC by 4 and put the result back in the PC.
- Can be described succinctly using RTL "Register-Transfer Language"

\[
\text{IR} = \text{Memory}[\text{PC}]; \quad \text{IR is "Instruction Register"} \\
\text{PC} = \text{PC} + 4;
\]

What is the advantage of updating the PC now?

**Step 2: Instruction Decode and Register Fetch**
- Read registers rs and rt in case we need them
- Compute the branch address in case the instruction is a branch
- RTL:

\[
A = \text{Reg}[\text{IR}[25-21]]; \\
B = \text{Reg}[\text{IR}[20-16]]; \\
\text{ALUOut} = \text{PC} + (\text{sign-extend(} \text{IR}[15-0] \text{)} <\!< 2); \\
\]

- We aren’t setting any control lines based on the instruction type (we are busy “decoding” it in our control logic)

**Step 3 (instruction dependent)**
- ALU is performing one of three functions, based on instruction type
- Memory Reference:

\[
\text{ALUOut} = A + \text{sign-extend}(\text{IR}[15-0]);
\]
- R-type:

\[
\text{ALUOut} = A \text{ op } B;
\]
- Branch:

\[
\text{if } (A==B) \text{ PC = ALUOut};
\]

**Step 4 (R-type or memory-access)**
- Loads and stores access memory

\[
\text{MDR} = \text{Memory[ALUOut]}; \quad \text{MDR is Memory Data Register} \\
\text{Memory[ALUOut]} = B;
\]
- R-type instructions finish

\[
\text{Reg}[\text{IR}[15-11]] = \text{ALUOut};
\]
Step 5 Memory Read Completion

- Reg[IR[20–16]] = MDR;

Summary:

<table>
<thead>
<tr>
<th>Step name</th>
<th>Action for R-type instructions</th>
<th>Action for memory-reference instructions</th>
<th>Action for branches</th>
<th>Action for jumps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction</td>
<td>PC = Memory[PC]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>A = Reg [IR[15–11]]</td>
<td>B = Reg [IR[20–16]]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution, address computation, branch, data completion</td>
<td>ALUOut = A + B</td>
<td>ALUOut = A + sign-extend IR[15–0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory access or R-type completion</td>
<td>Reg[IR[15–11]] = Load: Memory(ALUOut)</td>
<td>Store: Memory(ALUOut) = B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory read completion</td>
<td>Load: Reg[IR[20–16]] = MDR</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>