CSE 141 Midterm Exam

2011 Winter

Professor Steven Swanson

1. Please write your name at the top of each page
2. This is a close book, closed notes exam. No outside material may be used.
3. You may use a calculator
4. Show your work. You will get more partial credit that way.
5. If you have any questions, please raise your hand.
6. Good luck!

Name:

Student ID:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Score</th>
<th>Out of</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>112</td>
</tr>
</tbody>
</table>
1. Short Answer Questions:
   1. Give the performance equation and define each of the terms. For each term, give two system components that can affect each term.
      
      \[ \text{Exec time} = \text{inst count} \times \text{cycles/inst} \times \text{sec/cyc} \]
      
      - **Inst count**: compiler, the program, the ISA
      - **Cpi**: microarchitecture, the compiler, the program
      - **sec/cyc**: microarchitecture, process technology
   
2. Give the equation of Amdal’s Law and define each of the terms.
   \[ \text{Stot} = \frac{1}{(x/S) + (1-x)} \]
   Stot = total speedup.  X = fraction of program affected, S = speedup for that portion.

3. Which of SRAMs and DRAMs provide the highest density (i.e., the most bits per square centimeter)? The fastest access time?
   - SRAM is less dense, but faster.

4. Give two aspects of a good ISA that contribute to uniformity and make it easy to design a processor to implement the ISA.
   - Few instruction formats, orthogonality, fixed inst length, constant amount of work per inst.

5. Describe the difference between a stall and flush. Give an example of when each is useful? (3%)
   Stalls freeze part of the pipeline, delaying the execution of some instructions. They are useful for waiting for hazards to resolve. Flushing removes an incorrect instruction from the pipeline. It is useful for canceling instructions fetched because of a misprediction.

6. Describe the 5 stages of the the standard MIPS pipeline.
   - Fetch – fetch the inst; Decode – decipher the inst and read from the reg file;
   - Execute – perform the operation; Memory – access memory; Write back – write the result into the reg file.

7. Give a metric for measuring the quality (i.e., the bigger the value, the better) of a computer system that takes power consumption and performance into account and places equal emphasis on each. Give another metric that places much more emphasis on power consumption.
   \[ \text{MIPS/W} \]
   \[ \text{MIPS/W}^3 \]
   They key is that the W has a larger exponent.
The table below shows the instruction type breakdown and the CPI of each instruction type of a given application running on a 3GHz StoneBear 438 processor.

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Percentage of Instruction Count</th>
<th>CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>55%</td>
<td>1</td>
</tr>
<tr>
<td>Load/Store</td>
<td>30%</td>
<td>3</td>
</tr>
<tr>
<td>Branch</td>
<td>15%</td>
<td>4</td>
</tr>
</tbody>
</table>

Assume the application has a total of $2 \times 10^9$ instructions. Please answer the following questions:

8. What is the total execution time of the given benchmark? (7%)

Total execution time = CPI * # of Instructions * clock cycle period
= $(0.55 \times 1 + 0.30 \times 3 + 0.15 \times 4) \times 2e9 \times 0.33e-9$
= $1.367$

9. A revision of StoneBear 438 processor raises the clock rate to 4GHz, but also increases the CPI of Load/Store instructions to 12. What’s the speedup of this revision vs the original? (7%)

Total execution time = CPI * # of Instructions * clock cycle period
= $(0.55 \times 1 + 0.30 \times 12 + 0.15 \times 4) \times 2e9 \times 0.25e-9$
= $2.375s$

SpeedUp = new time/old time = $1.367/2.375 = 0.576$
2. Performance calculations

1. If 20% of the program is optimized to run at 4x and 40% of the program at 2x, what is the total speedup? (Ans) \( \frac{100}{\left( \frac{20}{2} + \frac{40}{2} + 40 \right)} = \frac{100}{65} = 1.538 \)

2. A software engineer decides to rewrite a portion of the program that accounts for 60% of execution time so that it can run on multiple processors in parallel. What is maximum speedup she can hope to achieve? (Ans) Maximum speedup \( \frac{100}{40} = 2.5 \)

3. Processor A is a single-cycle processor and runs at a clock rate of 1Mhz. Processor B is a pipelined version of A with 12 stages. Assuming ideal pipelining, what is the clock rate of B? Give two reasons why B will probably not attain this clock rate. (Ans) Assuming ideal pipeline, clock rate of B is 12Mhz

2 reasons why B might not attain 12 Ghz
   a. It might not be possible to divide it into 12 equal stages
   b. There are additional delays due to pipeline registers

3. Scaled add:

1. On the pipeline diagram on the next page add the components necessary to execute a “scaled add”: \( \text{adds rd, rs, rt} \Rightarrow R[rd] = R[rs] + (R[rt] \ll 2) \).

2. Rewrite (or show how you would modify) the following code to use the scaled add to accelerate execution

   \[
   \text{sll } t0, a1, 2 \\
   \text{add } t1, a0, t0 \\
   \text{lw } v0, 0(t1) \\
   \text{sll } t2, v0, 2 \\
   \text{add } v0, v0, t2
   \]

   becomes

   \[
   \text{adds } t1, a0, a1 \\
   \text{lw } v0, 0(t1) \\
   \text{adds } v0, v0, v0
   \]

3. What fraction of execution would the above code need to account for in order to speedup total execution by 1.31? (assume that adding the instruction does not affect the cycle time). (5 instructions originally became 3 instructions)

   So, \( s = \frac{5}{3} \)

   \[
   S_{\text{tot}} = 1.31 = 1 / \left( \frac{x}{s} + (1-x) \right) \\
   1.31 = 1 / \left( \frac{x}{(5/3)} + (1-x) \right)
   \]

   Solving for x gives \( x = 0.58 \)
The shifter can also go before the pipeline register.

Figure 1 The pipelined processor
4. Consider the following MIPS assembly code:
   
   ```
   LOOP:   lw    $t1, 0($a0)       // 1
           lw    $a0, 0($t1)       // 2
           addi $a1, $a1, -1      // 3
           bne  $a1, $zero, LOOP  // 4
           add  $v0, $a0, $zero   // 5
           addi $sp, $sp, 8       // 6
   ```

   Assume that we are using the pipeline processor shown in Figure 1 (The processor we used in class) and $a1$ is initialized to 2. Please answer the following questions:
   
   1. List the sequence of instructions that will be executed during the first two iterations of the loop. (You can just list the numbers after the ‘//’ if you wish.) 
      (3%)

      1, 2, 3, 4, 1, 2, 3, 4, 5, 6

   2. Use arrows to show all the data dependencies between the executed instructions. Circle the instructions that may cause control hazards. (10%)

   ```
   lw    $t1, 0($a0)       // 1
   lw    $a0, 0($t1)       // 2
   addi $a1, $a1, -1      // 3
   bne  $a1, $zero, LOOP  // 4
   lw    $t1, 0($a0)       // 1
   lw    $a0, 0($t1)       // 2
   addi $a1, $a1, -1      // 3
   bne  $a1, $zero, LOOP  // 4
   add  $v0, $a0, $zero   // 5
   addi $sp, $sp, 8       // 6
   ```
Assume that the pipeline processor implements all possible full forwarding paths and a predict not-taken branch predictor. Draw the pipeline diagram for the instructions you listed above. Include instructions, if any, that are fetched incorrectly. Use the letters F, D, E, M, and W to represent pipe stages, S for stall and X for the noop that replaces an instruction when it’s squashed. (12%)
5. Consider the following C code: int foo(int a) {
    int b=1;
    while(a != 0) {
        b = b * a;
        a = a - 1;
    }
    return b;
}

The code is compiled as the assembly code shown in the following table. How many memory accesses and arithmetic operations will the foo function require? Assume a = 5. Please complete the following table: (10%)

<table>
<thead>
<tr>
<th></th>
<th>Static Arithmetic Operations</th>
<th>Static Memory Operations</th>
<th>Dynamic Arithmetic Ops</th>
<th>Dynamic Memory Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pushl %ebp</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>movl %esp, %ebp</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>subl $16, %esp</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>movl $1, -4(%ebp)</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>jmp L2</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>L3:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>movl -4(%ebp), %eax</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>imull 8(%ebp), %eax</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>movl %eax, -4(%ebp)</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>subl $1, 8(%ebp)</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>L2:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmpl $0, 8(%ebp)</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>ine L3</td>
<td>1</td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>movl -4(%ebp), %eax</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>leave</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ret</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Total:</td>
<td>22</td>
<td>10</td>
<td>69</td>
<td>31</td>
</tr>
</tbody>
</table>

2. Is the above code compiled with or without optimization turned on? Why or why not? (5%) Without, since all the variable are stored on the stack rather than in registers.
6. Assume that the following is the outcome of a particular branch when it was executed within a loop: T, T, T, NT, T, T, NT, T, T, NT
   
   1. What is the percentage of misprediction for a “predicted-taken” predictor (Ans)

   Misprediction = 3 / 12 = 25%

   2. Assume that in a typical execution, 80% of your instructions are branches that follow the above outcome pattern. Currently there is a “predicted-not-taken” predictor in your machine. In order to optimize the program, you can either change the predictor to a 2-bit predictor or you can change the code and reduce the number of branch instructions by half. Compute the speedup for each alternative. Which optimization is the best choice?

   2-bit predictor with initial value 00
   00 - Strongly not taken, 01 - weakly not taken, 10 - weak taken, 11 - strongly taken

<table>
<thead>
<tr>
<th>Actual</th>
<th>Predictor State</th>
<th>Predicted State</th>
<th>Correct /Incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>00</td>
<td>NT</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>01</td>
<td>NT</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>10</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>NT</td>
<td>11</td>
<td>T</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>10</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>11</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>11</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>NT</td>
<td>11</td>
<td>T</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>10</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>11</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>NT</td>
<td>11</td>
<td>T</td>
<td>0</td>
</tr>
</tbody>
</table>

   Misprediction = 5 / 12 = 41.67%
(Ans)
CPI-predict-not-taken = not_branches*no_branch_cpi + branches* mis_predict_rate * mispredict_CPI + branches*(1-mis_predict_rate)*correct_predict_CPI

CPI-predict-not-taken = 0.2*1 + 0.8 * 0.75 * (2 + 1) + 0.8*0.25*1 = 2.2

CPI-fewer-branches = = 0.6*1 + 0.4 * 0.75 * (2 + 1) + 0.4*0.25*1 = 1.6

CPI-two-bit = 0.2*1 + 0.8 * 0.416 * (2 + 1) + 0.8*(1-.416)*1 = 1.66

Speedup-fewer-branches = 2.2/1.6 = 1.375

Speedup-2-bit = 2.2/1.66 = 1.325

Second option is better