Due: Wednesday, November 27 at 5:30 p.m.

Submission: Hand me a paper copy or slide it under my office door (SCI 3809).

Problems

1. Pipeline control
   Consider the instruction
   \[
   \text{add } $10, $1, $2
   \]
   Show what is stored in each pipeline register as the instruction progresses through the pipeline. Assume
   register 1 contains the value 5 and register 2 contains the value 3. Consider control as well as data.

2. Pipeline performance
   Each pipeline stage has some latency. Additionally, pipelining introduces registers between stages, and
   each register adds further latency.

<table>
<thead>
<tr>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
<th>Pipeline register</th>
</tr>
</thead>
<tbody>
<tr>
<td>200ps</td>
<td>150ps</td>
<td>150ps</td>
<td>300ps</td>
<td>150ps</td>
<td>20ps</td>
</tr>
</tbody>
</table>

   (a) What is the latency of a \textit{lw} instruction in a nonpipelined vs a pipelined processor?
   (b) What is the clock cycle time in a nonpipelined vs a pipelined processor?
   (c) Ignoring hazards, what is the speed-up achieved by introducing pipelining?
   (d) If we could split one stage into two new stages, each with half the latency of the original stage,
       which stage would you split?
   (e) What is the cycle time for your new pipeline? Assuming the CPI remains constant, what speedup
       is obtained?
   (f) What stage(s), if any, would you split next? When would you stop splitting, and why?

3. Data hazards
   Consider the following sequence of instructions executed on a five-stage pipelined datapath:

   \[
   \text{lw } $1, 40($6) \\
   \text{add } $2, $3, $1 \\
   \text{add } $1, $2, $6 \\
   \text{sw } $2, 20($4) \\
   \text{and } $1, $1, $4
   \]

   (a) Assuming there is no forwarding, indicate data hazards and show how to insert \textit{nop} instructions
       to eliminate the hazards.
   (b) Assuming there is forwarding, draw a multiple-clock-cycle pipeline diagram, as we drew in class,
       to show where and when data will be forwarded in the execution of these instructions. Show any
       stalls (“pipeline bubbles”) that are required.

4. Branch prediction accuracy
   (Adapted from Patterson & Hennessy, exercise 4.24) Consider the following sequences of branch out-
   comes. T means the branch is taken; N means the branch is not taken.

   T, T, T, N
   T, N, T, T, N
(a) What is the accuracy of the always-taken branch predictor for each sequence?
(b) What is the accuracy of the always-not-taken branch predictor for each sequence?
(c) What is the accuracy of the 2-bit branch predictor for each sequence? Assume the predictor starts off in state 01 (weak not taken).
(d) What is the accuracy of the 2-bit branch predictor for each sequence, if the sequence repeats thousands of times?

5. Branch prediction & performance

(Adapted from Patterson & Hennessy, exercise 4.23) The importance of having a good branch predictor depends on how often conditional branches are executed. Together with branch predictor accuracy, this will determine how much time is spent stalling due to mispredicted branches.

In this exercise, assume the breakdown of instructions into various categories is as follows:

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>60%</td>
</tr>
<tr>
<td>BEQ</td>
<td>8%</td>
</tr>
<tr>
<td>JMP</td>
<td>2%</td>
</tr>
<tr>
<td>LW</td>
<td>20%</td>
</tr>
<tr>
<td>SW</td>
<td>10%</td>
</tr>
</tbody>
</table>

Also, assume the following branch predictor accuracies:

<table>
<thead>
<tr>
<th>Predictor</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always-taken</td>
<td>65%</td>
</tr>
<tr>
<td>Always-not-taken</td>
<td>35%</td>
</tr>
<tr>
<td>2-bit</td>
<td>98%</td>
</tr>
</tbody>
</table>

(a) Stall cycles due to mispredicted branches increase the CPI. Assume that branch outcomes are determined in the EX stage, that there are no data hazards, and no delay slots are used. What is the CPI with the always-taken predictor?
(b) What is the CPI for the always-not-taken predictor?
(c) What is the CPI for the 2-bit predictor?

6. Multiple issue

(Adapted from Patterson & Hennessy 4.28) In this exercise we compare the performance of 1-issue and 2-issue processors, taking into account program transformations that can be made to optimize for 2-issue execution.

Consider the following loop (written in C):

```
for (i=0; i!=j; i+=2)  
  a[i+1] = a[i];
```

When writing MIPS code, assume that the variable i is stored in register $s0, j is stored in $s1, and a is stored in $s7.

(a) Translate the C code into MIPS assembly. Your translation should be direct, without rearranging instructions to achieve better performance.
(b) Draw a pipeline diagram for your MIPS code from (a) executed on a 2-issue, statically scheduled processor as shown in Figure 4.69. Assume the loop exits after only two iterations and the processor has perfect branch prediction.
(c) Rearrange your code from (a) to achieve better performance on a 2-issue statically scheduled processor.
(d) Draw a pipeline diagram for your revised code.
(e) For this code, what is the speedup of going from a 1-issue processor to a 2-issue processor? Assume the assembler is able to produce optimized code for each processor and that 1,000,000 iterations of the loop are executed.