1. (12 pts) Basic pipelining. Use the following code fragment:

DADDI  R1,R2,#4  ; R1 ← R2+4  
LD     R3,8(R1)  ; R3 ← address (8+R1)  
DADD   R4,R1,R3 ; R4 ← R1+R3

a. (2 pts) List all RAW (read-after-write) pipeline hazards in the code, regardless of whether they cause any stalls.

Three RAW hazards (dependences) exist:
- DADDI writes R1, then LD reads R1
- DADDI writes R1, then DADD reads R1
- LD writes R3, then DADD reads R3

b. (2 pts) Assume there is no forwarding or bypassing hardware.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>DADDI</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD</td>
<td>IF</td>
<td>ST</td>
<td>ST</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DADD</td>
<td>IF</td>
<td>ST</td>
<td>ST</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Stalls are needed between both DADDI/LD and LD/DADD due to the RAW pipeline hazards. Two stalls are needed in each case, to ensure ID stage is executed at same time as WB stage of previous instruction.

c. (4 pts) Assume normal forwarding and bypassing hardware.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>DADDI</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DADD</td>
<td>IF</td>
<td>ID</td>
<td>ST</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

No stalls are needed between DADDI and LD, since forwarding can be used to handle the RAW pipeline hazard for R1. A stall is needed between LD and DADD, since the value for R3 loaded from memory is not available until the end of the MEM stage.

d. (4 pts) Describe all forwarding used in part (c).

1) output of EX stage for DADDI is forwarded to EX stage for LD to calculate the load address (in place of the value of R1 from the register file). 2) output of MEM stage for LD is forwarded to EX stage for DADD for calculating the sum (in place of the value of R3 from the register file).
2. (6 pts) Control hazards
   a. (2 pts) Explain why stalls may need to be inserted after a conditional branch instruction.

   Because the next instruction to be executed depends on whether the conditional branch is taken, and a stall is needed because the decision may not be known in 1 clock cycle.

   b. (2 pts) Would stalls ever need to be inserted after a conditional branch instruction if we know if the branch is always taken? Explain.

   Yes, since even if we know the branch is always taken, we still need to calculate the actual destination of the branch to find the address of the instruction to fetch next.

   c. (2 pts) Explain what a branch delay slot is.

   The instruction placed after a branch instruction that is executed regardless of whether the branch is taken (since the branch is delayed).

3. (8 pts) Pipeline hazards. Consider the following MIPS floating point pipeline:

   Processors implement logic to check for potential data hazards (such as RAW and WAW) and forwarding. Recall that the rd field of an instruction is generally the destination register, where the result of an instruction is stored. Consider the following check: IF/ID.IR[op]=MUL.D & A2/A3.IR[op]=ADD.D & IF/ID.IR[rd]=A2/A3.IR[rd]

   a. (4 pts) Explain what the logic is checking

   Whether the MUL.D instruction is in the IF/ID and ADD.D instruction is in the A2/A3 pipeline registers, and whether they have the same destination register.

   b. (4 pts) Explain whether the check is needed

   The check is NOT needed, since a MUL.D instruction takes longer to complete than an ADD.D instruction, so even if the destination register is the same the earlier ADD.D instruction will have finished by the time the MUL.D instruction is ready to store its result into the destination register.

4. (4 pts) Pipeline performance. Suppose processor X executes instructions in the following 4 stages (no pipeline), where each stages could run this fast. Compare the performance of a pipelined vs. unpipelined implementation of processor X.

   Unpipelined instructions will take 15+15+30+10=70ns.

   Pipeline instructions will take (30+30+30+30)/4 = 30ns on average, assuming no stalls. So performance is improved by 70/30 = 7/3, about 2 and 1/3.