CIS 662: Midterm

Name: ___________________________               Points: ____/100

First read all the questions carefully and note how many points each question carries and how difficult it is. You have 1 hour 15 minutes. Plan your time accordingly.

1. (20 points)
A processor has the following stages in its pipeline: IF ID ALU1 MEM ALU2 WB. ALU instructions and load/store instructions can access memory. ALU1 stage is used for effective address calculation for ALU operations, loads and stores, and target calculation for branches. ALU2 stage is used for all other calculations and for branch resolution. The only supported addressing mode is displacement addressing. Assume all ALU operations (MUL, ADD, SUB) take one cycle in ALU2 stage.

   a) (10 points) Show the pipeline timing diagram (which instruction goes into which pipeline stage at each clock cycle) without forwarding. How many cycles does it take to execute the following instruction sequence? How many stalls have to be inserted?

   ADD R1, R3, 50(R2)
   MUL R6, R4, 100(R1)
   STORE R6, 50(R2)
   ADD R1, R1, #100
   SUB R2, R2, #8

   Clock cycle       1  2  3  4  5  6  7  8  9  10
   ADD R1, R3, 50(R2) IF  ID  ALU1 MEM  ALU2 WB
   MUL R6, R4, 100(R1) IF  S   S   S   S   ID  ALU1 MEM  ALU2 WB
   STORE R6, 50(R2)    IF  S   S   S   S   ID  
   ADD R1, R1, #100
   SUB R2, R2, #8

   Clock cycle       11 12 13 14 15 16 17 18 19 20
   ADD R1, R3, 50(R2)
   MUL R6, R4, 100(R1)
   STORE R6, 50(R2)    ALU1 MEM  ALU2 WB
   ADD R1, R1, #100    ID  ALU1 MEM  ALU2 WB
   SUB R2, R2, #8      IF  ID  ALU1 MEM  ALU2 WB

16 cycles, 6 stalls
b) (10 points) Show the pipeline timing diagram (which instruction goes into which pipeline stage at each clock cycle) with forwarding. How many cycles does it take to execute the following instruction sequence? Indicate the pipeline stages between which the information is forwarded. How many stalls are left?

<table>
<thead>
<tr>
<th>Clock cycle</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>
</tr>
</thead>
<tbody>
<tr>
<td>ADD R1, R3, 50(R2)</td>
<td>IF</td>
<td>ID</td>
<td>ALU1</td>
<td>MEM</td>
<td>ALU2*</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL R6, R4, 100(R1)</td>
<td>IF</td>
<td>ID</td>
<td>S</td>
<td>S</td>
<td>ALU1</td>
<td>MEM</td>
<td>ALU2*</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE R6, 50(R2)</td>
<td>IF</td>
<td>S</td>
<td>S</td>
<td>ID</td>
<td>ALU1</td>
<td>S</td>
<td>MEM</td>
<td>ALU2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD R1, R1, #100</td>
<td>IF</td>
<td>ID</td>
<td>S</td>
<td>ALU1</td>
<td>MEM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB R2, R2, #8</td>
<td>IF</td>
<td>S</td>
<td>ID</td>
<td>ALU1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Clock cycle</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD R1, R3, 50(R2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL R6, R4, 100(R1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE R6, 50(R2)</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD R1, R1, #100</td>
<td>ALU2</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB R2, R2, #8</td>
<td>MEM</td>
<td>ALU2</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Forwarding is done between ALU2/WB and ID/ALU1, and between ALU2/WB and ALU1/MEM

13 cycles, 3 stalls
2. (40 points) For the processor from question 1
   a) (5 points) How large is the branch penalty?

   Branches are fully resolved only in ALU2 stage, so we can do IF for next
   instruction in WB stage of the branch. We would like to do it in ID stage of the
   branch. The penalty is therefore 4 cycles.

   b) (5 points) Assume that we can introduce an optimization so that branches are
   resolved in ALU1 stage, and the target address for branches is also calculated
   there. How large is the branch penalty now?

   Branches are fully resolved in ALU1 stage, so we can do IF for next instruction in
   MEM stage of the branch. We would like to do it in ID stage of the branch. The
   penalty is therefore 2 cycles.
c) **(18 points)** We are considering an optimization from part b) where branches are resolved in ALU1 stage. This optimization increases clock cycle by 10% for all instructions and can be used in 65% of branches. Branches represent 35% of all instructions in our usual workload. Ideal CPI is 1. How large speedup can we achieve with this optimization?

Old_CPU_time = Old_CPI*Old_IC*Old_CC
Old_CPI = 1 + 0.35*branch_penalty = 1+0.35*4=2.4
Old_CPU_time = 2.4*Old_IC*Old_CC

New_CPU_time = New_CPI*New_IC*New_CC
New_IC=Old_IC
New_CC=Old_CC*1.1
Old_CPI = 1 + 0.35*branch_penalty = 1+0.35*(0.65*2 + 0.35*4) = 1.945
New_CPU_time = 1.945*Old_IC*Old_CC*1.1 = 2.1395*Old_IC*Old_CC

Speedup = Old_CPU_time/New_CPU_time = 1.12
d) (12 points) We are considering a branch prediction mechanism, for the original architecture from question 1. This optimization does not increase clock cycle time. Branches represent 35% of all instructions in our usual workload. Ideal CPI is 1. If 50% of branches are not taken, 30% are taken and 20% are jumps:

a. How large is penalty for taken and not taken branches with predict-taken strategy?

We know the target in ALU1. For taken branches penalty will be 2 cycles, for not-taken we have to wait until after ALU2 so the penalty will be 4 cycles.

b. How large is penalty for taken and not taken branches with predict-not-taken strategy?

For taken branches penalty will be 4 cycles, for not-taken it will be 0.

c. How large is new average CPI with predict-taken strategy?

\[ 1 + 0.35\times(0.5\times4 + 0.5\times2) = 2.05 \text{ cycles} \]

d. How large is new average CPI with predict-not-taken strategy?

\[ 1 + 0.35\times(0.5\times0 + 0.5\times4) = 1.7 \text{ cycles} \]
3. **(20 points)** Consider Tomasulo’s algorithm (TA).

   **a) (10 points)** Explain how TA handles WAW hazards and how this differs from the way scoreboarding handles WAW hazards.

   **TA:** When an instruction is issued the destination register is associated with functional unit producing the result. If a later instruction wants to write to the same destination, the destination register will be associated with the functional unit producing the result of the later instruction. There will be no delay in issuing the later instruction.

   **Scoreboarding:** When an instruction is issued the destination register is associated with functional unit producing the result. If a later instruction wants to write to the same destination, the issue of this instruction will be stalled until the prior instruction performs write to the destination register.
b) (10 points) Explain how TA handles WAR hazards and how this differs from the way scoreboard handles WAR hazards.

TA: When an instruction is issued, the reservation station holds the available operand values. If the values are not available, the reservation station holds the reference to functional unit producing the values. If a later instruction wants to write to a register that holds the source operand of the prior instruction it is free to do so, since the current value of that register (that will be overwritten by later instruction) is either saved in the appropriate reservation station for the instruction that will use it as source operand, or will be stored there once it is produced by another functional unit.

Scoreboarding: When an instruction is issued, the functional unit status table holds the reference to registers holding source operands. These registers will be read only when all source operands are available. If a later instruction wants to write to a register that holds the source operand of the prior instruction it will be stalled until the prior instruction has read its operands.
4. (20 points) Explain what are delayed branches, how do we decide the number of delay slots, what choices do we have to populate these slots and what considerations we must observe to keep the program execution correct. Finally, describe the difference between delayed and nullifying branches.

Delayed branches are branches followed by specially chosen instructions that will be executed while we wait for the branch to resolve. The number of delay slots is equal to the number of cycles of waiting for a branch to completely resolve – until both the condition and the target are known. Instructions placed into a delay slot must always be executed, regardless of the branch outcome. We can choose instructions to populate delay slots “from before”, “from target” and “from fall through”. When choosing “from before” we must select instructions that do no affect the branch condition, so that they can be moved after then branch. When choosing “from target” we must select instructions that do no affect registers that are used in “fall through” part of the code. Similarly, when choosing “from fall-through” we must select instructions that do no affect registers that are used in “target” part of the code.

Nullifying branches are similar to delayed branches – they are also followed by delay slots where we can place useful instructions. However, a compiler tries to predict a nullifying branch’s outcome and chooses the code “from fall-through” for predict-not-taken or code “from target” for predict-taken branches. Compiler also inserts the prediction into the executable code. While instructions in the delay slots of delayed branches must always be executed, instructions following a nullifying branch can be turned into no-ops at run time if the branch was mispredicted. Because of this opportunity to cancel instructions from delay slots, a compiler can choose harmful instructions to populate delay slots for nullifying branches.