Question 1 (5 points – 3(a) + 2(b)):

In this question, we examine how latencies of individual components of the processor datapath affect the clock cycle time of the pipelined architecture shown in slide 35 of the class lecture. Assume the following latencies for logic blocks in the datapath and assume that blocks not listed have zero latencies:

- I-Mem: 200ps
- Add: 130ps
- ALU: 280ps
- Regs read: 110ps
- Regs write: 110ps
- D-Mem: 200ps
- Decoder: 150ps
- Mux: 10ps
- Sign-extend: 10ps
- Reading the inter-stage buffer or the PC = 5ps
- Writing the inter-stage buffer or the PC = 5ps

a. Compute the cycle time for the five stage pipelined design assuming that writing into the register file occurs in the first half of a cycle and reading from the register file occurs during the second half of the cycle.

**IF stage:** two parallel paths:

- Path 1: PC read(5) + I-mem(200) + IF/ID buffer write(5) = 210ps
- Path 2: PC read(5) + Add(130) + mux (10) + PC write (5) = 150ps

Hence, the minimum time for this stage is 210ps
CS 1541 – Spring 2018  
Homework 2 Answer Key

**ID stage**: Path 1: IF/ID buffer read(5) + Reg read(110) + ID/EX buffer write(5) = 120ps  
Path 2: IF/ID buffer read(5) + decode(150) + ID/EX buffer write(5) = 160ps  
Hence, the minimum time for this stage is 160ps

**EX stage**: Path 1: buffer read(5) + Add(130) + EX/MEM buffer write(5) = 140ps  
Path 2: buffer read(5) + Mux(10) + ALU(280) + EX/MEM buffer write(5) = 300ps

**MEM stage**: Path 1: buffer read(5) + D-mem(200) + buffer write(5) = 210ps  
Path 2: buffer read(5) + mux(10) + PC write(5) = 20p

**WB stage**: MEM/WB buffer read(5) + Mux(10) + Reg write(110) = 125ps

The cycle time must equal that of the maximum stage i.e. the EX stage which takes 300ps.

b. Now, assume that the ALU unit (and not the Add units) is decomposed into two units, ALU1 and ALU2 with delays of 130ps and 150ps, respectively with the goal of obtaining a 6-stage pipeline (IF, ID, ALU1, ALU2, MEM, WB) that is more balanced than the 5-stage pipeline (IF, ID, ALU, MEM, WB). What would be the cycle time for this 6-stage pipeline and in which stage would you place the "shift left" and the "branch target ADD" units (ALU1 or ALU2)? Note that you have to separate the ALU1 and ALU2 stages by inter-stage buffers.

With the new 6 stage pipeline, we can split the EX stage in EX1 and EX2. The time for the EX stages alone would become:

**EX1 stage**: buffer read(5) + Mux(10) + ALU1(130) + EX1/EX2 buffer write(5) = 150ps  
**EX2 stage**: EX1/EX2 buffer read(5) + ALU2(150) + EX2/MEM buffer write(5) = 160ps

The new cycle time for the pipeline will be 210ps (bound by the IF and MEM stages).

Place units Shift Left in stage EX1 with the ALU1, and Branch Target ADD in stage EX2 (though since the ADD takes 130ps, it could be placed in either stage without affecting cycle time).

**Question 2 (6 points – 4(a) + 2(b))**:

Assume that the pipelined architecture does not use separate memories for instructions and data. That is a structural hazard occurs when one instruction wants to use the data memory (Memory stage) and another instruction wants to use the instruction memory (Fetch stage). Assume also that the hazard is resolved by stalling the instruction that is in the Fetch stage (which means inserting a bubble in the pipeline).

a. Using a table similar to the one shown below, trace the pipeline configuration for the execution of the following code:
Homework 2 Answer Key

I1: lw $1, 100($0)
I2: and $3, $2, $4
I3: add $9, $9, $10
I4: lw $7, 50($6)
I5: and $3, $3, $1
I6: sw $8, 40($6)
I7: sub $2, $2, $2
I8: beq $1, $6, 200

<table>
<thead>
<tr>
<th>IF</th>
<th>ID</th>
<th>EXE</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle 1</td>
<td>lw $1,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycle 2</td>
<td>and $3, $2,</td>
<td>lw $1,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycle 3</td>
<td>add $9, $9,</td>
<td></td>
<td>lw $1,</td>
<td></td>
</tr>
<tr>
<td>Cycle 4</td>
<td>lw $7, 50($6)</td>
<td>add $9, $9,</td>
<td></td>
<td>lw $1,</td>
</tr>
<tr>
<td>Cycle 5</td>
<td>lw $7, 50($6)</td>
<td>add $9, $9,</td>
<td></td>
<td>and $3, $2,</td>
</tr>
<tr>
<td>Cycle 6</td>
<td>and $3, $3,</td>
<td>lw $7, 50($6)</td>
<td>add $9, $9,</td>
<td>and $3, $2,</td>
</tr>
<tr>
<td>Cycle 7</td>
<td>sw $8,</td>
<td>lw $7, 50($6)</td>
<td></td>
<td>add $9, $9,</td>
</tr>
<tr>
<td>Cycle 8</td>
<td>sub $2, $2,</td>
<td>sw $8,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycle 9</td>
<td>sub $2, $2,</td>
<td>sw $8,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycle</td>
<td>beq $1, $6,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycle</td>
<td>beq $1, $6,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycle</td>
<td>beq $1, $6,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycle</td>
<td>beq $1, $6,</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

b. Ignoring any hazard except the above structural hazard, how many cycles would it take to complete the execution of 1000 instructions assuming that 20% of the instructions are lw/sw instructions, 65% are R-type instructions, and 15% are branch instructions?

Each lw/sw instruction will result an extra cycle delay at the MEM stage which conflicts with other instructions that are on the IF stage.

The CPI = 1 + 0.2 = 1.2 → hence, it will take 1200 cycles to complete the 1000 instructions (ignoring the 4 cycles needed to fill in the pipeline)

Question 3 (9 points – 2(a) + 2(b) + 2(c) + 1.5(d) + 1.5(e)):

(1) add $6, $2, $2
(2) lw $3, 40($6)
(3) sub $4, $5, $6
(4) sw $6, 50($3)
(5) lw $5, 10($4)
(6) beq $5, $6, L
a. Identify all the data dependencies in the code given above.
1. Instruction 2 depends on instruction 1 (register $6$)
2. Instruction 3 depends on instruction 1 (register $6$)
3. Instruction 4 depends on instruction 1 (register $6$)
4. Instruction 6 depends on instruction 1 (register $6$)
5. Instruction 4 depends on instruction 2 (register $3$)
6. Instruction 5 depends on instruction 3 (register $4$)
7. Instruction 6 depends on instruction 5 (register $5$)

b. Identify which dependencies will cause data hazards in a 5-stage pipeline implementation without forwarding hardware.
The following dependences will cause hazards since the two instructions are separated by fewer than two intervening instructions
1. Instruction 2 depends on instruction 1 (register $6$)
2. Instruction 3 depends on instruction 1 (register $6$)
3. Instruction 4 depends on instruction 2 (register $3$)
4. Instruction 5 depends on instruction 3 (register $4$)
5. Instruction 6 depends on instruction 5 (register $5$)

c. Show the code after adding no-ops to avoid hazards (needed for correctness in the absence of forwarding hardware)
1. add $6$, $2$, $2$
2. no-op
3. no-op
4. lw $3$, 40($6$)
5. sub $4$, $5$, $6$
6. no-op
7. sw $6$, 50($3$)
8. lw $5$, 10($4$)
9. no-op
10. no-op
11. beq $5$, $6$, L

d. Now assume that the architecture has special hardware to implement forwarding and stalling. Trace, by building a table similar to the one you used in question 2, the execution of the six instructions on the 5-stage pipeline.

<table>
<thead>
<tr>
<th>Cycle 1</th>
<th>IF</th>
<th>ID</th>
<th>EXE</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle 2</td>
<td>lw</td>
<td>add</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycle 3</td>
<td>sub</td>
<td>lw</td>
<td>add</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycle 4</td>
<td>sw</td>
<td>sub</td>
<td>lw</td>
<td>add</td>
<td></td>
</tr>
<tr>
<td>Cycle 5</td>
<td>lw</td>
<td>sw</td>
<td>sub</td>
<td>lw</td>
<td>add</td>
</tr>
<tr>
<td>Cycle 6</td>
<td>beq</td>
<td>lw</td>
<td>sw</td>
<td>sub</td>
<td>lw</td>
</tr>
<tr>
<td>Cycle 7</td>
<td>beq</td>
<td>--</td>
<td>lw</td>
<td>sw</td>
<td>sub</td>
</tr>
<tr>
<td>Cycle 8</td>
<td>beq</td>
<td>--</td>
<td>--</td>
<td>lw</td>
<td>sw</td>
</tr>
<tr>
<td>Cycle 9</td>
<td>beq</td>
<td>--</td>
<td>--</td>
<td>beq</td>
<td>--</td>
</tr>
<tr>
<td>Cycle 10</td>
<td>beq</td>
<td>--</td>
<td>--</td>
<td>beq</td>
<td>--</td>
</tr>
<tr>
<td>Cycle 11</td>
<td>beq</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
e. Can you reorder the instructions to obtain a more efficient execution of the code segment? -- Justify your answer.

Yes, we can avoid stalling by interchanging instructions 4 and 5 (no data dependence)

```
add  $6, $2, $2
lw   $3, 40($6)
sub  $4, $5, $6
lw   $5, 10($4)
sw   $6, 50($3)
beq  $5, $6, L
```

**Question 4 (5 points – 3(a) + 2(b))**:  
As discussed in class, even with forwarding hardware, the pipeline has to be stalled when a "lw" instruction which loads data into a register "Rt" is followed by an instruction which reads from register "Rt". In this question, you will explore an exception to this rule in the case where the instruction following "lw $Rt, I1($R1)" is "sw $Rt, I2($R2)" that saves the content of "Rt" (just loaded) into memory.

a. Argue that in this case, it is possible to avoid stalling by providing a forwarding path between the MEM/WB buffer to the input of the data memory.

<table>
<thead>
<tr>
<th>Cycle</th>
<th>IF</th>
<th>ID</th>
<th>EXE</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle 1</td>
<td>lw $Rt, I1($R1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycle 2</td>
<td>sw $Rt, I2($R2)</td>
<td>lw $Rt, I1($R1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycle 3</td>
<td>sw $Rt, I2($R2)</td>
<td>lw $Rt, I1($R1)</td>
<td></td>
<td></td>
<td>Forwarding</td>
</tr>
<tr>
<td>Cycle 4</td>
<td>sw $Rt, I2($R2)</td>
<td>lw $Rt, I1($R1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycle 5</td>
<td>sw $Rt, I2($R2)</td>
<td>lw $Rt, I1($R1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycle 6</td>
<td>sw $Rt, I2($R2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

By passing the value of $Rt from MEM/WB to EXE/MEM at the end of cycle 4, the sw instruction can use the correct value of $Rt at the beginning of cycle 5, and write the correct value to the memory.

b. Draw a diagram for the part of the data path between the EX/MEM and MEM/WB buffers showing the forwarding path described in part (a). Use a simple figure similar to those on slides 46-48.