1) If this exam were a CPU, you’d be halfway through the pipeline (Sp15 Final)
We found that the instruction fetch and memory stages are the critical path of our 5-stage pipelined MIPS CPU. Therefore, we changed the IF and MEM stages to take two cycles while increasing the clock rate. You can assume that the register file is written at the falling edge of the clock.

Assume that no pipelining optimizations have been made, and that branch comparisons are made by the ALU. Here’s how our pipeline looks when executing two add instructions:

<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>
</tr>
</thead>
<tbody>
<tr>
<td>add $t0, $t1, $t2</td>
<td>IF1</td>
<td>IF2</td>
<td>ID</td>
<td>EX</td>
<td>MEM1</td>
<td>MEM2</td>
<td>WB</td>
<td></td>
</tr>
<tr>
<td>add $t3, $t0, $t5</td>
<td>IF1</td>
<td>IF2</td>
<td>ID</td>
<td>EX</td>
<td>MEM1</td>
<td>MEM2</td>
<td>WB</td>
<td></td>
</tr>
</tbody>
</table>

a) How many stalls would a data hazard between back-to-back instructions require?
3 stalls
b) How many stalls would be needed after a branch instruction?
4 stalls
c) Suppose the old clock period was 150 ns and the new clock period is now 100ns. Would our processor have a significant speedup executing a large chunk of code...

   i) Without any pipelining hazards? Explain your answer in 1-2 sentences.
   Yes, due to 1.5x throughput

   ii) With 50% of the code containing back-to-back data hazards? Explain your answer in 1-2 sentences.
   Yes, penalty is 300 ns per hazard in both cases, so our new processor will still have higher throughput.
Problem 2: movz and movnz (Sp15 MT2)

Consider adding the following instruction to MIPS (disregard any existing definition in the green sheet):

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>movz rd, rs, rt</td>
<td>if (R[rs] == 0) R[rd]&lt;-R[rt]</td>
</tr>
<tr>
<td>movnz rd, rs, rt</td>
<td>if (R[rs] != 0) R[rd]&lt;-R[rt]</td>
</tr>
</tbody>
</table>

a) Translate the following C code using movz and movnz. Do not use branches.

```
// a -> $s0, b-> $s1, c-> $s2
int a = b < c ? b : c;
```

```
MIPS

slt $t0, $s1, $s2
movnz $s0, $t0, $s1
movz $s0, $t0, $s2
```

b) Implement movz (but not movnz) in the datapath. Choose the correct implementation for (a), (b), and (c). Note that you do not need to use all the signals provided to each box, and the control signal MOVZ is 1 if and only if the instruction is movz.
c) Generate the control signals for movz. The values should be 0, 1, or X (don’t care) terms. You must use don’t care terms where possible.

<table>
<thead>
<tr>
<th>MOVZ</th>
<th>RegDst</th>
<th>ExpOp</th>
<th>RegWr</th>
<th>ALUSrc</th>
<th>ALUCtr</th>
<th>MEMWr</th>
<th>MemToReg</th>
<th>Jump</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0001, 0010, or 0110</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

This table shows the ALUCtr values for each operation of the ALU:

<table>
<thead>
<tr>
<th>Operation</th>
<th>AND</th>
<th>OR</th>
<th>ADD</th>
<th>SUB</th>
<th>SLT</th>
<th>NOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALUCtr</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0110</td>
<td>0111</td>
<td>1100</td>
</tr>
</tbody>
</table>
Problem 3) Watch Your (Time) Step! (Su12 Final)

Consider the following difference equation (i.e. differential equation that is discretized in time):

\[ x[n+1] - x[n] = -\alpha x[n] \]
\[ x[n+1] = (1-\alpha) x[n] \]

Here we restrict ourselves to integer values of \( \alpha \) and use the following integration function:

<table>
<thead>
<tr>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 ps</td>
<td>100 ps</td>
<td>400 ps</td>
<td>200 ps</td>
<td>100 ps</td>
</tr>
</tbody>
</table>

```
integrate:
1   beq   $a1,$0,exit
2   sub   $t0,$0,$a2       # $t0 = -\alpha
3   addi  $t0,$t0,1        # $t0 = 1-\alpha
4   lw    $t1,0($a0)      #$t1=x[n]
5   mult  $t0,$t1
6   mflo  $t1             #$t1 = (1-\alpha)x[n]
7   sw    $t1,4($a0)       # x[n+1] = (1-\alpha)x[n]
8   addiu $a0,$a0,4
9   addiu $a1,$a1,-1
10  j     integrate
11  jr    $ra
```

We are using a 5-stage MIPS pipelined datapath with separate I$ and D$ that can read and write to
registers in a single cycle. Assume no other optimizations (no forwarding, etc.). The default behavior
is to stall when necessary. Multiplication and branch checking are done during EX and the HI and LO
registers are read during ID and written during WB.

a) As a reminder, \( T_c \) stands for “time between completions of instructions.” Given the following
datapath stage times, what is the ratio \( T_{c,\text{single-cycle}}/T_{c,\text{pipelined}} \)?

\[
\frac{1000}{400} = \frac{5}{2}
\]

b) Count the number of the different types of potential hazards found in the code above:

Structural:____0_______ Data:____6_______ Control:____3_______

None @ 2-3, 3-5, 4-5, 5-6, 6-7, 9-1 @ 1, 10, 11

For the following questions, examine a SINGLE ITERATION of the loop (do not consider the jr).

c) With no optimizations, how many clock cycles does our 5-stage pipelined datapath take in one
loop iteration (end your count exactly on completion of j)?

\[ 24 = 10 \text{ instructions} - 4 \text{ for draining pipeline} + 2^*(\# \text{ data hazards} - 2) + 2 \text{ for beq control hazard}. \text{ Beq hazard is 2 stalls because branch comparison done in EX stage.} \]

\# data hazards minus 2 because 3-5 and 9-1 hazards taken care of by other stalls.

d) How many clock cycles LESS would be taken if we had FORWARDING?

\[ 7 = 2^*(\# \text{ of data hazards} - 1) - 1 \text{ because of load hazard. Again, you can ignore the 3-5 hazard here.} \]
e) Assuming that we introduce both FORWARDING and DELAY SLOTS, describe independent changes to integrate that will reduce the total clock cycles taken. Changes include replacing or moving instructions. You may not need all spaces given.

<table>
<thead>
<tr>
<th>Instr</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td><em><strong>3</strong></em></td>
<td>Move after 4 (or move instr 4 before instr 3)</td>
</tr>
<tr>
<td><em><strong>8/9</strong></em></td>
<td>Move after 10</td>
</tr>
</tbody>
</table>

OR

<table>
<thead>
<tr>
<th>Instr</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td><em><strong>9</strong></em></td>
<td>Move after 4</td>
</tr>
<tr>
<td><em><strong>8</strong></em></td>
<td>Move after 10</td>
</tr>
</tbody>
</table>

**F4) Please Pass Professor’s Pretty Pipeline-pourri Problem... (30 pts, 42 min)**