1. (10 pts) Basic pipelining

Use the following code fragment:

- LW R1, 0(R4) ; R1 ← address (0+R4)
- ADDI R2, R1, #8 ; R2 ← R1+8
- MULT R3, R1, R1 ; R3 ← R1*R1
- SW R3, 4(R2) ; address(4+R2) ← R3

a. (2 pts) List all RAW (read-after-write) pipeline hazards in the code, regardless of whether they cause any stalls. Specify hazards in the form (I3 → I5 for R8).

Use the classic MIPS five-stage integer pipeline, show the timing of this instruction sequence. Assume all memory accesses take 1 clock cycle, and a register may be read and written in the same clock cycle.

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

Stalls are needed between both LW/ADDI and MULT/SW 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. (3 pts) Assume normal forwarding and bypassing hardware.

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 LW and ADDI, since the value for R1 loaded from memory is not available until the end of the MEM stage.

d. (3 pts) Describe all forwarding used in part (c). Use the following format (I_m used EX/MEM→EX from I_n) to show instruction I_m forwarded a value from EX/MEM pipeline register to instruction I_n to be used in MEM.

I2 used MEM/WB→EX from I1
I4 used MEM/WB→EX from I2
I4 used MEM/WB→MEM from I3
2. (6 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 format of MIPS register-register instructions are \( \text{rd} = \text{rs} \ \text{OP} \ \text{rt} \), and register immediate instructions (including load/store) are \( \text{rt} = \text{rs} \ \text{OP} \ \text{immed} \). Consider the following check:

\[
\begin{align*}
\text{IF/ID.IR}[\text{op}] &= \text{MUL.D} & \text{A4/MEM.IR}[\text{op}] &= \text{ADD.D} \\
\text{IF/ID.IR}[\text{rt}] &= \text{A4/MEM.IR}[\text{rd}]
\end{align*}
\]

a. (3 pts) Explain what the logic is checking

Whether the MUL.D instruction is in the IF/ID and ADD.D instruction is in the A4/MEM pipeline registers, and whether the destination register of the MUL.D instruction is the same as the operand register of the ADD.D instruction (i.e., whether RAW hazard exists).

b. (3 pts) Explain whether the check is needed

The check **IS** needed. If a RAW hazard exists between the MUL.D and the ADD.D instruction, the value of the MUL.D instruction needs to be forwarded to the EX stage of the ADD.D instruction, since the MUL.D instruction won’t reach the WB stage until after the ID stage of the ADD.D instruction has finished.

3. (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.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF&amp;ID</td>
<td>25ns</td>
</tr>
<tr>
<td>EX</td>
<td>20ns</td>
</tr>
<tr>
<td>MEM</td>
<td>40ns</td>
</tr>
<tr>
<td>WB</td>
<td>15ns</td>
</tr>
</tbody>
</table>

a. (2 pts) Compare the performance of a pipelined vs. unpipelined implementation of processor X.

Unpipelined instructions will take 25+20+40+15=100ns.
Pipeline instructions will take \( (40+40+40+40)/4 = 40 \)ns on average, assuming no stalls.
So performance is improved by \( 100/40 = 5/2 \), about 2.5 times.

b. (2 pts) How would the performance of a pipelined implementation of processor X be affected if every 4\(^{th}\) instruction causes a 1 cycle stall?

If a stall occurs every 4\(^{th}\) instruction, pipeline instructions will take \( (40+40+40+40+40)/4 = 50 \)ns on average. Performance compared to pipeline with no stalls would be \( 40/50 = 4/5 \), about 0.8 or 80\% of the performance with no stalls.