1. Read the single-cycle datapath with control and answer questions: (6 X 5 points)

(1) Explain the purpose of the adder ("Add") at the top right of the diagram (used in which instructions for what functionality).
   It is used to calculate the branch destination address in conditional branches.

(2) When is MemtoReg control signal asserted (being “1”)?
   The value fed to the register Write data input comes from the data memory (lw).

(3) When is ALUSrc control signal asserted?
   The second ALU operand is the sign-extended, lower 16 bits of the instruction. (lw and sw)

(4) Give an example instruction whose execution asserts the RegDst signal, and an example instruction whose execution deasserts the RegDst signal (being “0”).
   RegDst will be asserted when the register destination number for the Write register comes from
   the rd field (bits 15:11).

   Asserted example: add $t0,$s1,$s2
   Reasserted example: lw $t0,32($s3)
(5) Is it possible that both RegWrite and MemWrite are asserted? Why?
No. You cannot write register and memory at the same time with one instruction.

(6) Is it possible that both MemtoReg and RegDst are asserted? Why?
Yes. For branch-on-equal instruction, those two can be asserted at same time because beq instruction doesn’t use those two controls. (‘don’t care’ signals)

<table>
<thead>
<tr>
<th>Field</th>
<th>0</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
</table>

a. R-type instruction

<table>
<thead>
<tr>
<th>Field</th>
<th>35 or 43</th>
<th>rs</th>
<th>rt</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit positions</td>
<td>31:26</td>
<td>25:21</td>
<td>20:16</td>
<td>15:0</td>
</tr>
</tbody>
</table>

b. Load or store instruction

c. Branch instruction

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Effect when deasserted</th>
<th>Effect when asserted</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegDst</td>
<td>The register destination number for the Write register comes from the rt field (bits 20:16).</td>
<td>The register destination number for the Write register comes from the rd field (bits 15:11).</td>
</tr>
<tr>
<td>RegWrite</td>
<td>None.</td>
<td>The register on the Write register input is written with the value on the Write data input.</td>
</tr>
<tr>
<td>ALUSrc</td>
<td>The second ALU operand comes from the second register file output (Read data 2).</td>
<td>The second ALU operand is the sign-extended, lower 16 bits of the instruction.</td>
</tr>
<tr>
<td>PCSrc</td>
<td>The PC is replaced by the output of the adder that computes the value of PC + 4.</td>
<td>The PC is replaced by the output of the adder that computes the branch target.</td>
</tr>
<tr>
<td>MemRead</td>
<td>None.</td>
<td>Data memory contents designated by the address input are put on the Read data output.</td>
</tr>
<tr>
<td>MemWrite</td>
<td>None.</td>
<td>Data memory contents designated by the address input are replaced by the value on the Write data input.</td>
</tr>
<tr>
<td>MemtoReg</td>
<td>The value fed to the register Write data input comes from the ALU.</td>
<td>The value fed to the register Write data input comes from the data memory.</td>
</tr>
</tbody>
</table>
2. Suppose logic blocks in a processor have the following latencies: (3 X 8 points)

(1) In a single cycle, non-pipelined processor, what is the minimum time between instructions for a processor only executing only R-type instructions?
For R-type instruction, there is no Data access stage, so the overall time is: 350 + 150 + 200 + 200 = 900ps.

(2) In a single cycle, non-pipelined processor, what is the minimum time between instructions for a processor executing R, I, and J-type instructions?

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Fetch</th>
<th>Register Read</th>
<th>ALU Operation</th>
<th>Data Access</th>
<th>Register Write</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>350ps</td>
<td>150ps</td>
<td>200ps</td>
<td>450ps</td>
<td>200ps</td>
<td>900ps</td>
</tr>
<tr>
<td>I-type</td>
<td>350ps</td>
<td>150ps</td>
<td>200ps</td>
<td>(450ps)</td>
<td>200ps</td>
<td>900ps(1350ps)</td>
</tr>
<tr>
<td>J-type</td>
<td>350ps</td>
<td>150ps</td>
<td>200ps</td>
<td></td>
<td></td>
<td>700ps</td>
</tr>
</tbody>
</table>

Minimum time should be 700ps.

(3) If the logic blocks above are each implemented as individual pipeline stages, what would be the minimum time between instructions for this pipelined CPU if hazards are ignored?
<table>
<thead>
<tr>
<th></th>
<th>Instruction Fetch</th>
<th>Register Read</th>
<th>ALU Operation</th>
<th>Data Access</th>
<th>Register Write</th>
<th>Time with pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load word</td>
<td>350ps</td>
<td>150ps</td>
<td>200ps</td>
<td>450ps</td>
<td>200ps</td>
<td>450ps</td>
</tr>
<tr>
<td>Store word</td>
<td>350ps</td>
<td>150ps</td>
<td>200ps</td>
<td>450ps</td>
<td>200ps</td>
<td>450ps</td>
</tr>
<tr>
<td>R-type</td>
<td>350ps</td>
<td>150ps</td>
<td>200ps</td>
<td></td>
<td>200ps</td>
<td>350ps</td>
</tr>
<tr>
<td>Branch</td>
<td>350ps</td>
<td>150ps</td>
<td>200ps</td>
<td></td>
<td></td>
<td>350ps</td>
</tr>
</tbody>
</table>

In a pipeline implementation, the minimum time between instructions will be 350ps.

3. **Consider the following sequence of instructions executed on the basic five-stage pipeline: (2 X 8 points)**

   add $t3, $t2, $t1  
or $t5, $t3, $t4  
and $t6, $t7, $t4  
add $t7, $t2, $t1

(1) Assuming our processor has no forwarding or hazard detection, insert a minimal amount of pipeline stalls to ensure correct execution (you may stall the pipeline with the instruction “nop”)

   add $t3, $t2, $t1  
nop  
nop  
or $t5, $t3, $t4  
and $t6, $t7, $t4  
add $t7, $t2, $t1

(2) Assuming our processor has no forwarding or hazard detection, rearrange the instructions and add stalls only if necessary to ensure proper execution. The values contained in the registers after executing your modified code should be equivalent to the unmodified execution.

   add $t3, $t2, $t1  
and $t6, $t7, $t4  
add $t7, $t2, $t1  
or $t5, $t3, $t4

4. **Problems in this exercise refer to the following sequence of instructions: (5 X 6 points)**

   lw r1, 0(r3)  
or r2, r1, r4  
or r1, r1, r2

Also, assume the cycle time for a datapath without forwarding is 250ps, the cycle time with full forwarding is 300ps, and the cycle time with only ALU-ALU forwarding is 290ps.
(1) Assume there is no forwarding in this pipelined processor. Indicate hazards and add nop instructions to eliminate them.

```
lw r1, 0(r3)
nop
nop
or r2, r1, r4 (Data Hazard)
nop
nop
or r1, r1, r2 (Data Hazard)
```

(2) Assume there is full forwarding. Indicate hazards and add NOP instructions to eliminate them.

```
lw r1, 0(r3)
nop
or r2, r1, r4
or r1, r1, r2
```

(3) What is the total execution time of this instruction sequence without forwarding and with full forwarding? What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?

The total execution time is the clock cycle time times the number of cycles. Without any stalls, a three-instruction sequence executes in 7 cycles (5 to complete the first instruction, then one per instruction)

The execution must add a stall for every NOP.

<table>
<thead>
<tr>
<th></th>
<th>No forwarding</th>
<th>With full forwarding</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(7 + 4) * 250ps  = 2750ps</td>
<td>(7 + 1) * 300 = 2400ps</td>
<td>1.15</td>
</tr>
</tbody>
</table>

(4) Add nop instructions to this code to eliminate hazards if there is ALU-ALU forwarding only (no forwarding from the MEM to the EX stage).

```
lw r1, 0(r3)
nop
nop
or r2, r1, r4
or r1, r1, r2 (ALU-ALU forwarding)
```

(5) What is the total execution time of this instruction sequence with only ALU-ALU forwarding? What is the speedup over a no-forwarding pipeline?

<table>
<thead>
<tr>
<th></th>
<th>No forwarding</th>
<th>With ALU-ALU forwarding only</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(7 + 4) * 250ps  = 2750ps</td>
<td>(7 + 2) * 290 = 2610ps</td>
<td>1.05</td>
</tr>
</tbody>
</table>