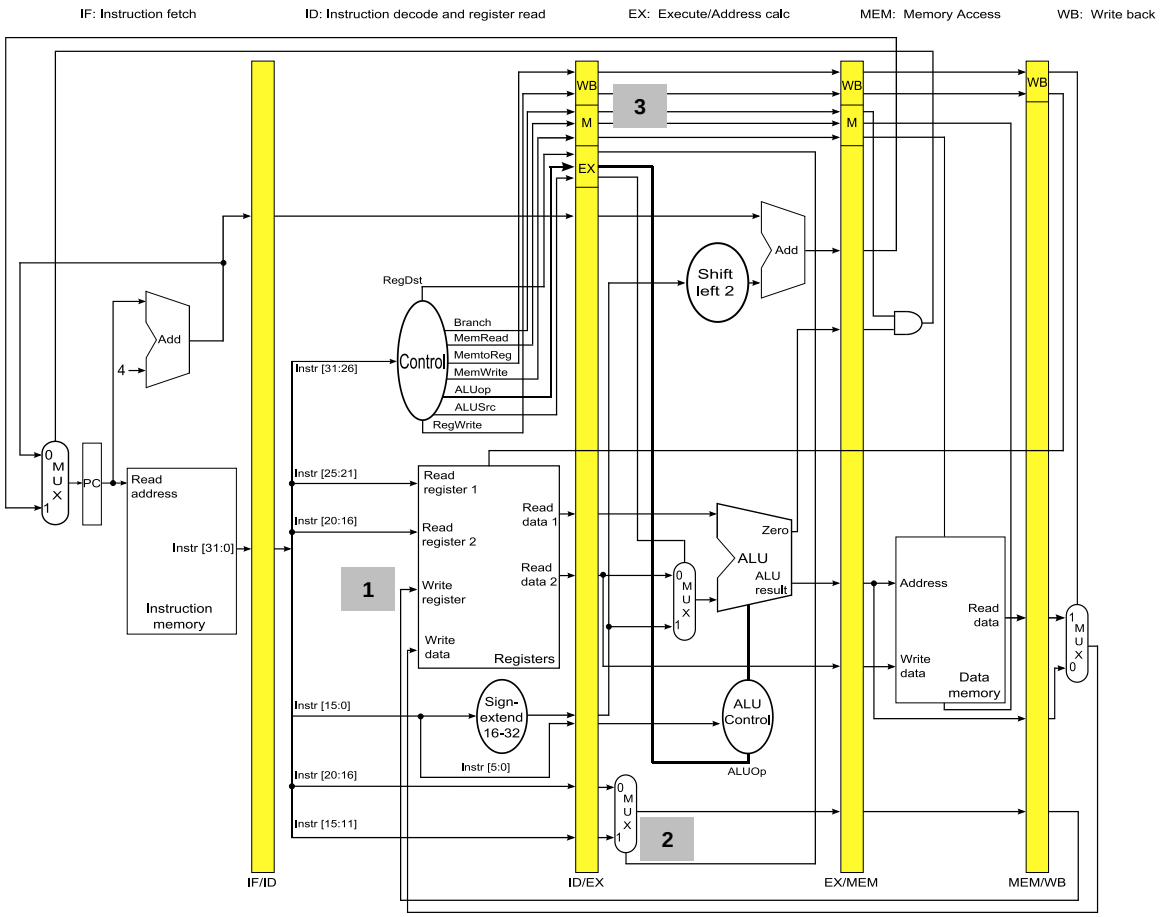
You may work in pairs for this assignment. If you choose to work with a partner, make sure only one of you submits a solution, and you paste a copy of the Partners Template that contains the names and PIDs of both students at the beginning of the file.

Prepare your answers to the following questions in a plain text file. Submit your file to the Curator system by the posted deadline for this assignment. No late submissions will be accepted. For all questions, show supporting work if you want partial credit.

You will submit your answers to the Curator System (www.cs.vt.edu/curator) under the heading `MIPS02`.

For questions 1 through 4, refer to the incomplete preliminary pipeline design, shown below, which includes the interstage buffers needed to synchronize signals and data with the instructions, but no support for forwarding operands. This datapath supports correct execution of any sequence of the following MIPS instructions: `add`, `sub`, `and`, `or`, `slt`, `lw`, and `sw` (assuming that data/control hazards are handled by having the assembler insert `nop` instructions).



1. [6 points] Why is the Write register value to the Registers passed through the MEM/WB interstage buffer, instead of being sent directly from the instruction in the ID stage? Be precise.

2. [6 points] Suppose we want to place the MUX controlled by the RegDst signal into the ID stage (instead of EX stage) to save the buffer space in the ID/EX interstage buffer. How many bits can we save with this design? (Note that you should modify the control signal accordingly as well).

3. [8 points] Consider the following sequence of MIPS32 assembly instructions fetched and executed consecutively from cycle 1. Suppose all the interstage buffers were zero initially. Find the control signal values stored in the ID/EX interstage buffer at cycle 5. To be precise, mark "don't care" if a control signal doesn't matter. (Ignore the modifications assumed in question 2, and solve based on the above figure).

```
add   $t0, $t1, $t2   # fetched at cycle 1
lw    $t3, 0($t4)     # fetched at cycle 2
sw    $t5, 0($t6)     # fetched at cycle 3
```

Interstage Buffer		Value
WB	ID/EX.MemtoReg	
	ID/EX.RegWrite	
M	ID/EX.Branch	
	ID/EX.MemRead	
	ID/EX.MemWrite	
EX	ID/EX.RegDst	
	ID/EX.ALUOp	(don't need to specify)
	ID/EX.ALUSrc	

4. Consider the following sequence of MIPS32 assembly instructions:

```
add   $t0, $t0, $t1   # 4.1
lw    $t1, 0($t0)     # 4.2
sw    $t1, 8($t0)     # 4.3
sub   $t2, $t0, $t3   # 4.4
lw    $t2, 16($t3)    # 4.5
add   $t0, $t2, $t2   # 4.6
```

A *data dependency* occurs when a later instruction requires an input value that is set by an earlier instruction. A *data hazard* occurs when one instruction writes a value into a register that will be used as input by a later instruction, but that value does not actually appear in the register by the cycle on which the later instruction attempts to read it. Note that a data hazard always implies a data dependency, but some data dependencies do not imply a data hazard. Also remember that this pipeline design <u>does not include any provision for forwarding operands</u>.

a) [6 points] Identify the data dependencies that would NOT prevent the given sequence of instructions from executing correctly on the given hardware design above, even if we do not insert `nop` instructions. For each such dependency, list the register involved, the writing instruction and the reading instruction.

Write your answers in the following form (answer below is NOT correct for this question):

```
writer    reader    register
-------------------------
 4.1       4.2       $t5
```

b) [10 points] Identify the data hazards that would prevent the given sequence of instructions from executing correctly on the given hardware design above, unless we inserted one or more `nop` instructions. For each such dependency, list the register involved, the writing instruction and the reading instruction.

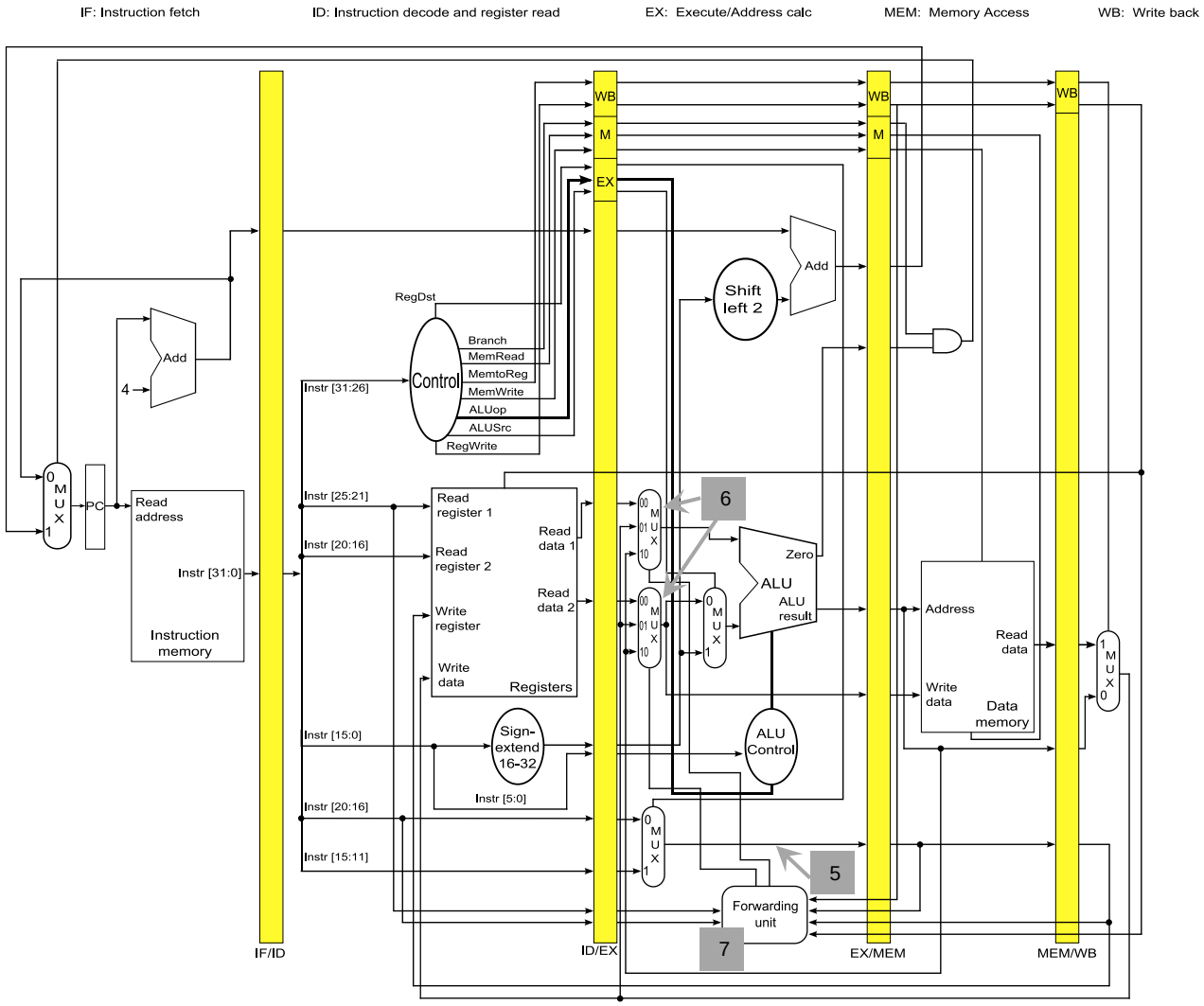
Write your answers in the following form (answer below is NOT correct for this question):

```
writer    reader    register
-------------------------
 4.1       4.2       $t5
```

c) [10 points] Rewrite the given sequence of instructions adding `nop` instructions so that the modified sequence would execute correctly on the given hardware design. For full credit, accomplish this with the smallest possible number of `nop` instructions.

For questions 5 through 7, refer to the pipeline design with forwarding, shown below, which supports execution of any sequence of the following MIPS instructions: `add`, `sub`, `and`, `or`, `slt`, and `sw`, (and `lw` so long as no stalls are needed to resolve load-use hazards).

Remember: this pipeline design does not include (load-use) hazard detection hardware, so it can forward operands but it cannot introduce stalls to deal with situations that forwarding alone will not handle.



5. [6 points] Why doesn't the Forwarding unit need the value of the destination register number in the EX stage (i.e., output of the MUX controlled by the RegDst signal) labelled 5 in the diagram? Or does it? Be precise.

6. [8 points] Suppose we have three `add` instructions to execute. Find one possible example of the second/third `add` instructions that make the Forwarding unit set the control signals for the MUXes labelled 6 to be 10 and 10 (for the upper one and lower one, respectively) when the second `add` is in the EX stage; and set 01 and 00 (for the upper one and lower one, respectively) when the third `add` is in the EX stage.

```
add   $t0, $t1, $t2    # 6.1
add   ?  , ?  , ?      # 6.2
add   ?  , ?  , ?      # 6.3
```

7. The Forwarding unit performs the following checks to detect MEM hazard in which the data is forwarded from the MUX controlled by the MemtoReg signal.

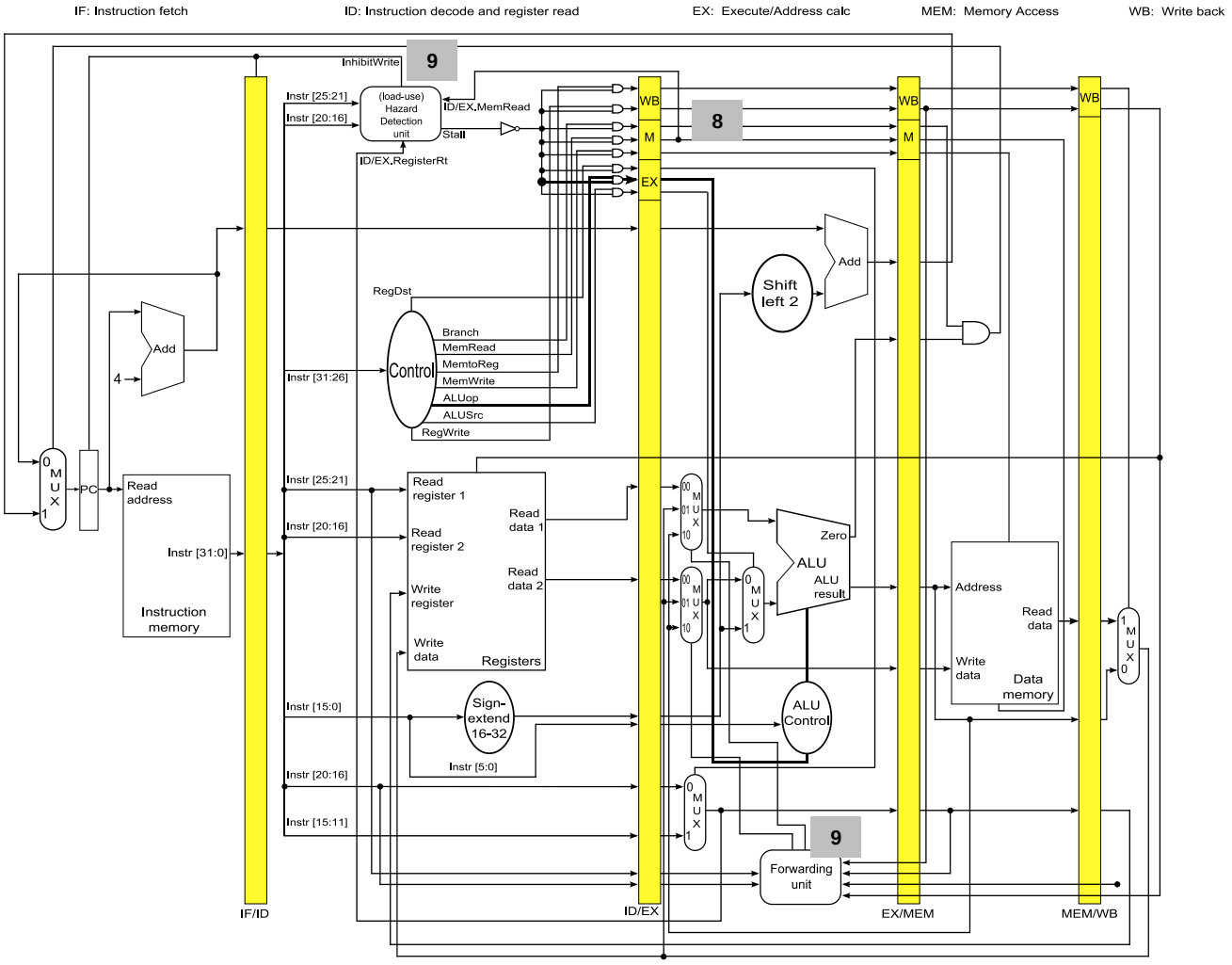
```
// Check for RegisterRs (Check for RegisterRt is similar)

if  (MEM/WB.RegWrite                                        // condition #1

and (MEM/WB.RegisterRd != 0)                                // condition #2

and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd !=0))       // condition #3
        and (EX/MEM.RegisterRd != ID/EX.RegisterRs)         // (included in condition #3)

and (MEM/WB.RegisterRd = ID/EX.RegisterRs))                 // condition #4
```

a) [6 points] Why should the Forwarding unit check condition #1? Be precise.

b) [6 points] Why should the Forwarding unit check condition #2? Be precise.

c) [6 points] Why should the Forwarding unit check condition #3? Be precise.

For questions 8 and 10, refer to the pipeline design with forwarding and (load-use) hazard detection, shown below, which supports execution any sequence of the following MIPS instructions: `add`, `sub`, `and`, `or`, `slt`, `lw`, and `sw`.



8. [6 points] Consider the following sequence of MIPS32 assembly instructions fetched and executed consecutively from cycle 1. Suppose all the interstage buffers were zero initially. Find the control signal values stored in the ID/EX interstage buffer at cycle 5. (Note that the instructions are different from question 3).

```
add   $t0, $t1, $t2    # fetched at cycle 1
lw    $t3, 0($t4)      # fetched at cycle 2
sw    $t5, 0($t3)      # fetched at cycle 3
```

Interstage Buffer		Value
WB	ID/EX.MemtoReg	
	ID/EX.RegWrite	
M	ID/EX.Branch	
	ID/EX.MemRead	
	ID/EX.MemWrite	
EX	ID/EX.RegDst	
	ID/EX.ALUOp	(don't need to specify)
	ID/EX.ALUSrc	

9. [8 points] Suppose we execute the following `lw` (only) instructions. If we have the (load-use) Hazard Detection unit as described in the above figure, then do we still need the Forwarding unit? Justify your answer.

```
lw    $t1, 0($t0)     # 9.1
lw    $t2, 0($t1)     # 9.2
```

10. [8 points] Given the (load-use) Hazard Detection unit and the Forwarding unit, how many clock cycles would be required to execute the following sequences of instructions?

```
add  $t1, $t0, $t0
lw   $t2, ($t1)
lw   $t3, ($t2)
add  $t4, $t2, $t3
add  $t5, $t3, $t4
```