Instructions:
1. The assignment is split into 2 sections:
2. Only Verilog assignment to be done in teams– Part (a) of the assignment to be done individually.
3. Each Verilog assignment should be accompanied by waveforms (output) and verilog (code) and submitted in HARDCOPY only.

Section 1: HARDCOPY SUBMISSION (100 Points) – 5a

1. (20 Points) Show the forwarding paths needed to execute the following 4 instructions:
   - add $3, $4, $6
   - sub $5, $3, $2
   - lw $7, 100($5)
   - add $8, $7, $2

2. (20 Points) Identify all of the data dependencies in the following code. Which dependencies are data hazards that will be resolved via forwarding? Which dependencies are data hazards that will cause a stall?
   - add $3, $4, $2
   - sub $5, $3, $1
   - lw $6, 200($3)
   - add $7, $3, $6

3. (20 Points) Consider executing the following code on the pipelined datapath of Figure 4.56 on page 370:
   - add $2, $3, $1
   - sub $4, $3, $5
   - add $5, $3, $7
   - add $7, $6, $1
   - add $8, $2, $6
   At the end of the 5th cycle of execution, which registers are being read and which are being written? What are the forwarding unit and hazard detection units doing during the 5th cycle of execution, if any comparisons are being made, mention them.

4. (15 Points) Write the control logic to implement forwarding when we have a memory-memory instruction i.e. lw followed by sw. In which stage will the forwarding be implemented and what is the control logic (similar to what we studied in class)?

5. (25 Points) We have a program core consisting of five conditional branches. The program core will be executed thousands of times. Below are the outcome of each branch for one execution of the program core (T for taken, N for Not taken)

Branch 1: T-T-T
Branch 2: N-N-N-N
Branch 3: T-N-T-N-T-N
Branch 4: T-T-T-N-T
Branch 5: T-T-N-T-T-N-T

Assume the behavior of each branch remains the same for each program core execution. For dynamic schemes assume each branch has its own prediction buffer and each buffer is initialized to the same state before each execution. List the predictions for the following branch prediction schemes:

a. Always taken
b. Always not taken
c. 1-bit predictor, initialized to predict taken
d. 2-bit predictor, initialized to weakly predict taken

6. (10 Points) BONUS: We have a program of 103 instructions in the format of “lw, add, lw, add, ...”. The add instruction depends (and only depends) on the lw instruction right before it. The lw instruction also depends (and only depends) on the add instruction right before it. If the program is executed on the pipelined datapath:
(a) What would be the actual CPI?
(b) Without forwarding, what would be the actual CPI?

Section 2: VERILOG SUBMISSION (100 Points) – 5b

(a) (35 Points) Design the control logic discussed in class. Please refer to the control logic diagram in the lecture notes corresponding to Lecture 13: Processor: Datapath and Control - 4. Use "switch" statements to generate intermediate results Reg, lw, sw, beq, bne, and jump. (NOTE: The figure in the lecture notes need to be modified to include bne, and jump explicitly.)
The module must have the following format:

```
module Control(opcode, ALUSrc, ALUOp, RegDst, MemWrite, MemRead, Beq, Bne, Jump, MemToReg, RegWrite)
input [5:0] opcode; // 6-bit operation code
output ALUSrc, RegDst, MemWrite, MemRead, Beq, Bne, Jump, MemToReg, RegWrite; // Output control lines
output [1:0] ALUOp; // 2-bit intermediate output for controlling ALU
```

(b) (15 Points) The control logic generates output ALUOp. This output is used to generate the 3-bit ALUControl signal that is required in the ALU designed in the previous part. Design a module to generate the 3-bit ALU control signals taking inputs as ALUOp and the 6-bit function field. The module must be as below:

```
module ALUOpToALUControl(ALUOp, Funct, ALUControl)
input [1:0] ALUOp; \\ninput [5:0] Funct;
output [2:0] ALUControl;
```

For the above two parts, submit your verilog code with comments and waveforms with explanation.
(c) (20 Points) Design a register file module that has the following format:

```
module RegisterFile(ReadRegister1, ReadRegister2, WriteRegister, WriteData, RegWrite, Clk, ReadData1, ReadData2);
```

input [4:0] ReadRegister1, ReadRegister2; // Two registers to be read
input [4:0] WriteRegister; // Register address to write into
input [31:0] WriteData; // Data to be written into WriteRegister
input RegWrite; // RegWrite control signal. Data is written only when this signal is enabled
output [31:0] ReadData1, ReadData2;

(d) (10 Points) Design a DataMemory module that has the following format:

```
module DataMemory(Address, WriteData, MemRead, MemWrite, Clk, ReadData);
```

input [6:0] Address; // 7-bit address to memory.
input [31:0] WriteData; // Data to be written into WriteRegister
input MemRead; // Data in memory location Address is read if this control is set
Input MemWrite; // WriteData is written in Address during positive clock edge if this control is set
output [31:0] ReadData; // Value read from memory location Address

Choose a few test cases that demonstrates the vital aspects of the implementation and submit your simulation output along with the code for RegisterFile and DataMemory.

(e) (20 Points) We are going to combine the Control and ALUControl blocks (in (a) and (b)) so that we can generate ALUControl a bit more flexibly. Rename the Control.v file that you created to "OldControl.v". The new Control module that we are going to have will have the following format. The new inputs and outputs are highlighted.

```
module Control(opcode, funct, ALUSrc, RegDst, MemWrite, MemRead, Beq, Bne, Jump, MemToReg, RegWrite, ALUControl)
```

input [5:0] opcode; // 6-bit operation code
input [5:0] funct; // 6-bit function code from the instruction // least significant 6 bits of an instruction
output ALUSrc, RegDst, MemWrite, MemRead, Beq, Bne, Jump, MemToReg, RegWrite; // Output control lines
output [2:0] ALUControl; // 3-bit control for the ALU that specifies the operation