CSE332: Computer Architecture and Organization
Assignment#5
Spring 2024

Name: _____
Student ID: _____

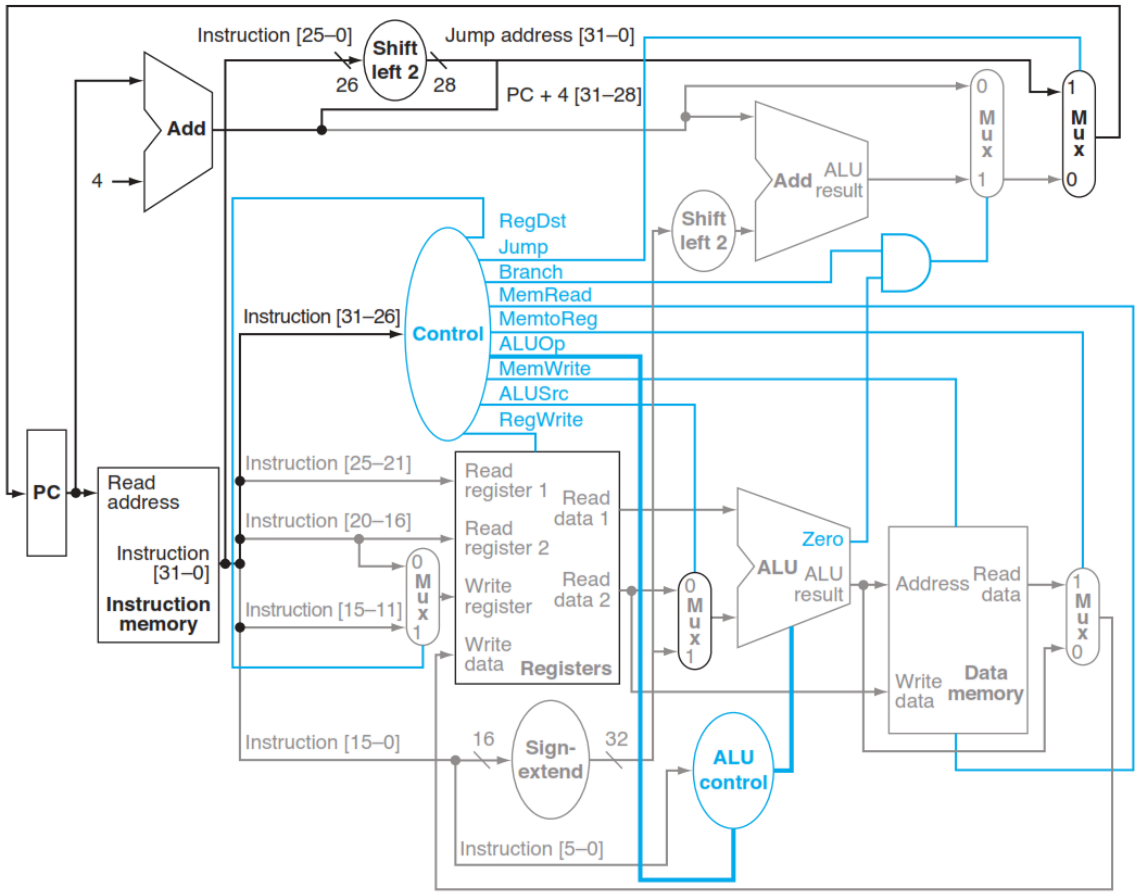
Direction: You can consult any resources such as books, online references, and videos for this assignment, however, you have to properly cite and paraphrase your answers when it is necessary. There will be points for partial attempts. You can upload a typed or handwritten copy of your assignment to the canvas. However, you have to show the original copy of your assignment in case of arising questions about the authenticity. You should upload your assignment to canvas.

Submission guidelines: All our assignments will be based on 100 points so that we can assign weights to get your points for final grade. Whoever fails to submit within the time period assigned through NSU canvas but submit the assignment within the next few days will be punished by 20% deduction of points (e.g. start with -20 points within 100 points). The late submission will be open until semester's end through email to TA and CC to the instructor. After the few days of grace period, the deduction will be 35%

Note: You are allowed to use only instructions implemented by the actual MIPS hardware provided in the Canvas references. Use assembly language format from the references or the book. Note, base ten numbers are listed as normal (e.g. 23), binary numbers are prefixed with 0b or format such as XX_2 and hexadecimal numbers are prefixed with 0x or format such as XX_{16}/XX_{HEX}

Note: If you have handwritten your assignment, you have to submit a scanned copy of it. Actions against plagiarism will be strictly enforced and proper action will be taken.

1. **(100 points, 70+30)** Modify (add drawing to the provided figure or draw by yourselves) the following MIPS architecture to add JAL, JR, SLL, SRL instructions. JAL instruction works similarly to J instruction but stores next instruction to $ra ($31) register. Therefore, JAL performs two operations at the same time: It not only stores the address of the next instruction ([PC+4]) in the return address register ($ra), but also jumps to the target address similar to the Jump instruction. JR instructions is an R-type instruction that reads the address for jump from the register $ra and jumps to that particular address. Also, decode your JAL, JR, SRL and SLL instructions to fill up and add more control signals to the truth table (page#3) as you require. **Also, modify the control.v to support JAL and JR instruction. You do not need to write a testbench or simulate the verilog program.**


Instruction [25–0]
Shift left 2
Jump address [31–0]
26
28
PC + 4 [31–28]
Add
4
0
M
u
x
1
1
M
u
x
0
Add
ALU result
Shift left 2
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
16
Sign-extend
32
ALU control
Zero
ALU
ALU result
Address
Read data
Write data
Data memory

Instruction	RegDst	ALU Src	MemTo Reg	Reg Write	Mem Read	MemW rite	Branch	Jump	ALU OP[1]	ALU OP[0]		
JAL												
JR												
SLL												
SRL												

```
// Code your design here
// file: control.v


`timescale 1ns/1ns

module Control_unit(input [5:0] Opcode,
                    input [5:0] Func,
                    input Zero,
                    output reg MemtoReg,
                    output reg  MemWrite,
                    output reg  ALUSrc,
                    output reg  RegDst,
                    output reg  RegWrite,
                    output reg  Jump,
                    output PCSrc,
                    output reg  [3:0] ALUControl
                    );

                reg [7:0] temp;
                reg Branch,BNE;

                always @(*) begin

                   case (Opcode)
                      6'b000000: begin                    // R-type
                                 temp <= 8'b11000000;

                                 case (Func)
                                 6'b100000: ALUControl <= 4'b0000;    // ADD
                                 6'b100001: ALUControl <= 4'b0000;    // ADDU
                                 6'b100010: ALUControl <= 4'b0001;    // SUB
                                 6'b100011: ALUControl <= 4'b0001;    // SUBU
                                 6'b100100: ALUControl <= 4'b0010;    // AND
```

```
            6'b100101: ALUControl <= 4'b0011;    // OR
            6'b100110: ALUControl <= 4'b0100;    // XOR
            6'b100111: ALUControl <= 4'b1010;    // NOR
            6'b101010: ALUControl <= 4'b1000;    // SLT
            6'b101011: ALUControl <= 4'b1001;    // SLTU
            6'b000000: ALUControl <= 4'b0101;    // SLL
            6'b000010: ALUControl <= 4'b0110;    // SRL
            6'b000011: ALUControl <= 4'b0111;    // SRA
            6'b000100: ALUControl <= 4'b1011;    // SLLV
            6'b000110: ALUControl <= 4'b1100;    // SRLV
            6'b000111: ALUControl <= 4'b1101;    // SRAV
        endcase

    end

6'b100011: begin                    // LW
            temp <= 8'b10100100;
            ALUControl <= 4'b0000;
        end

6'b101011: begin                    // SW
             temp <= 8'b00101000;
             ALUControl <= 4'b0000;
        end

6'b000100: begin                    // BEQ
             temp <= 8'b00010000;
            ALUControl <= 4'b0001;
        end

6'b000101: begin                    // BNE
            temp <= 8'b00010001;
            ALUControl <= 4'b0001;
        end

6'b001000: begin                    // ADDI
            temp <= 8'b10100000;
            ALUControl <= 4'b0000;
        end

6'b001001: begin                    // ADDIU
            temp <= 8'b10100000;
            ALUControl <= 4'b0000;
        end

6'b001100: begin                    // ANDI
            temp <= 8'b10100000;
            ALUControl <= 4'b0010;
        end

6'b001101: begin                    // ORI
```

```
                temp <= 8'b10100000;
                ALUControl <= 4'b0011;
            end

        6'b001110: begin                    // XORI
                temp <= 8'b10100000;
                ALUControl <= 4'b0100;
            end

        6'b001010: begin                    // SLTI
                temp <= 8'b10100000;
                ALUControl <= 4'b1000;
            end

        6'b001011: begin                    // SLTIU
                temp <= 8'b10100000;
                ALUControl <= 4'b1001;
            end

        6'b000010: begin                    // J
                temp <= 8'b00000010;
                ALUControl <= 4'b0010;
            end

        6'b001111:  begin                   // LUI
                temp <= 8'b10100000;
                ALUControl <= 4'b1110;
            end
        default:   temp <= 12'bxxxxxxxxxxxx;      // NOP
    endcase



    {RegWrite,RegDst,ALUSrc,Branch,MemWrite,MemtoReg,Jump,BNE} = temp;

end

assign PCSrc = Branch & (Zero ^ BNE);

endmodule
```