

Lab 11: Multicycle Processor Controller SOLUTIONS

Digital Design and Computer Architecture: RISC-V Edition (Harris & Harris, Elsevier © 2021)

1. Please indicate how many hours you spent on this lab. This will be helpful for calibrating the workload for next time the course is taught.
2. Hierarchical SystemVerilog for your controller module matching the declaration given above.

```
module controller(input  logic      clk,
                  input  logic      reset,
                  input  logic [6:0] op,
                  input  logic [2:0] funct3,
                  input  logic      funct7b5,
                  input  logic      Zero,
                  output logic [1:0] ImmSrc,
                  output logic [1:0] ALUSrcA, ALUSrcB,
                  output logic [1:0] ResultSrc,
                  output logic      AdrSrc,
                  output logic [2:0] ALUControl,
                  output logic      IRWrite, PCWrite,
                  output logic      RegWrite, MemWrite);

    logic [1:0] ALUOp;
    logic      Branch, PCUpdate;

    // Main FSM
    mainfsm fsm(clk, reset, op,
                ALUSrcA, ALUSrcB, ResultSrc, AdrSrc,
                IRWrite, PCUpdate, RegWrite, MemWrite,
                ALUOp, Branch);

    // ALU Decoder
    aludec ad(op[5], funct3, funct7b5, ALUOp, ALUControl);

    // Instruction Decoder
    instrdec id(op, ImmSrc);

    // Branch logic
    assign PCWrite = (Branch & Zero) | PCUpdate;

endmodule

module mainfsm(input  logic      clk,
               input  logic      reset,
               input  logic [6:0] op,
               output logic [1:0] ALUSrcA, ALUSrcB,
               output logic [1:0] ResultSrc,
               output logic      AdrSrc,
               output logic      IRWrite, PCUpdate,
               output logic      RegWrite, MemWrite,
               output logic [1:0] ALUOp,
               output logic      Branch);
```

```

typedef enum logic [3:0] {FETCH, DECODE, MEMADR, MEMREAD, MEMWB,
                          MEMWRITE,
                          EXECUTER, EXECUTEI, ALUWB,
                          BEQ, JAL, UNKNOWN} statetype;

statetype state, nextstate;
logic [14:0] controls;

// state register
always @(posedge clk or posedge reset)
    if (reset) state <= FETCH;
    else state <= nextstate;

// next state logic
always_comb
    case(state)
        FETCH:                                nextstate = DECODE;
        DECODE: casez(op)
            7'b0?00011: nextstate = MEMADR;    // lw or sw
            7'b0110011: nextstate = EXECUTER;  // R-type
            7'b0010011: nextstate = EXECUTEI;  // addi
            7'b1100011: nextstate = BEQ;       // beq
            7'b1101111: nextstate = JAL;       // jal
            default:    nextstate = UNKNOWN;
        endcase
        MEMADR:
            if (op[5]) nextstate = MEMWRITE;  // sw
            else       nextstate = MEMREAD;    // lw
        MEMREAD: nextstate = MEMWB;
        EXECUTER: nextstate = ALUWB;
        EXECUTEI: nextstate = ALUWB;
        JAL:      nextstate = ALUWB;
        default:  nextstate = FETCH;
    endcase

// state-dependent output logic
always_comb
    case(state)
        FETCH: controls = 15'b00_10_10_0_1100_00_0;
        DECODE: controls = 15'b01_01_00_0_0000_00_0;
        MEMADR: controls = 15'b10_01_00_0_0000_00_0;
        MEMREAD: controls = 15'b00_00_00_1_0000_00_0;
        MEMWRITE: controls = 15'b00_00_00_1_0001_00_0;
        MEMWB:    controls = 15'b00_00_01_0_0010_00_0;
        EXECUTER: controls = 15'b10_00_00_0_0000_10_0;
        EXECUTEI: controls = 15'b10_01_00_0_0000_10_0;
        ALUWB:    controls = 15'b00_00_00_0_0010_00_0;
        BEQ:      controls = 15'b10_00_00_0_0000_01_1;
        JAL:      controls = 15'b01_10_00_0_0100_00_0;
        default:  controls = 15'bxx_xx_xx_x_xxxx_xx_x;
    endcase

    assign {ALUSrcA, ALUSrcB, ResultSrc, AdrSrc, IRWrite, PCUpdate,
            RegWrite, MemWrite, ALUOp, Branch} = controls;

endmodule

```

```

module aludec(input  logic      opb5,
              input  logic [2:0] funct3,
              input  logic      funct7b5,
              input  logic [1:0] ALUOp,
              output logic [2:0] ALUControl);

    logic RtypeSub;
    assign RtypeSub = funct7b5 & opb5; // TRUE for R-type subtract
    instruction

    always_comb
    case(ALUOp)
        2'b00:          ALUControl = 3'b000; // addition
        2'b01:          ALUControl = 3'b001; // subtraction
        default: case(funct3) // R-type or I-type ALU
            3'b000: if (RtypeSub)
                ALUControl = 3'b001; // sub
            else
                ALUControl = 3'b000; // add, addi
            3'b010:     ALUControl = 3'b101; // slt, slti
            3'b110:     ALUControl = 3'b011; // or, ori
            3'b111:     ALUControl = 3'b010; // and, andi
            default:    ALUControl = 3'bxxx; // ???
        endcase
    endcase
endmodule

module instrdec (input  logic [6:0] op,
                 output logic [1:0] ImmSrc);

    always_comb
    case(op)
        7'b0110011: ImmSrc = 2'bxx; // R-type
        7'b0010011: ImmSrc = 2'b00; // I-type ALU
        7'b0000011: ImmSrc = 2'b00; // lw
        7'b0100011: ImmSrc = 2'b01; // sw
        7'b1100011: ImmSrc = 2'b10; // beq
        7'b1101111: ImmSrc = 2'b11; // jal
        default:     ImmSrc = 2'bxx; // ???
    endcase
endmodule

```

3. Does your controller pass your test vectors? **YES.**

Please indicate any bugs you found in this lab manual, or any suggestions you would have to improve the lab.