**COAL Lab # 04**

**Designing Control Unit in Verilog**

**Name:** Muhammad Usman

**Class:** BSCS 3C1

**Reg No:** cs211208

**Lab Task:**

**Literature Review:**

**Control Unit:**

Control Unit The control unit computes the control signals based on the opcode and funct fields of the instruction, Instr[31:25], Instr[14:12] and Instr[6:0]. Most of the control information comes from the opcode, but for further operations function fields are used. Thus, we will simplify our design by factoring the control unit into two blocks of combinational logic.

**Main Decoder:**

All R-type instructions use the same main decoder values; they differ only in the ALU decoder output. Recall that, for instructions that do not write to the register file (e.g., S-type and B-type), the ResultSrc control signals is don't care (X); the address and data to the register write port do not matter because RegWrite is not asserted. The logic for the decoder can be designed using your favorite techniques for combinational logic design.

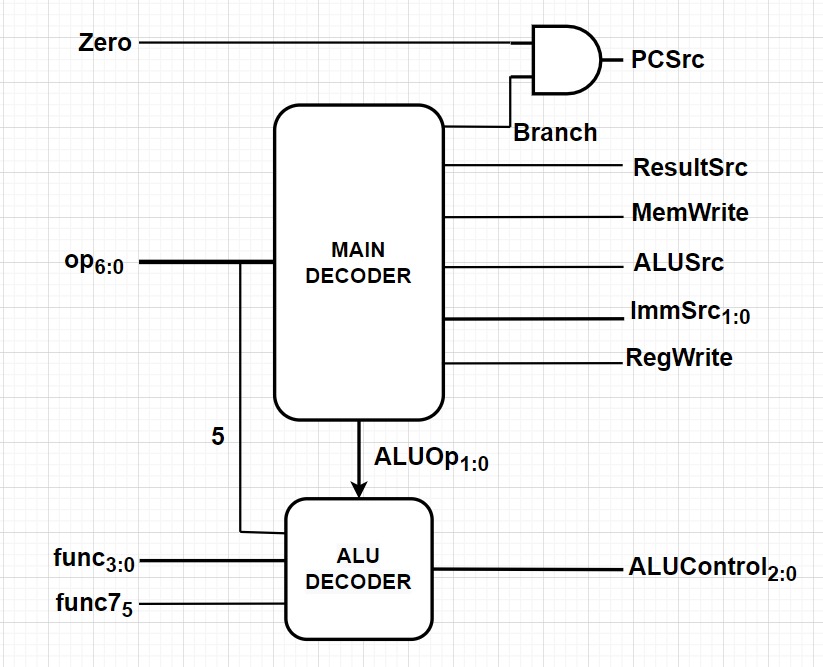
**ALU Decoder:**

The main decoder computes most of the outputs from the opcode. It also determines a 2-bit ALUOp signal. The ALU decoder uses this ALUOp signal in conjunction with the funct field and opcode bit to compute ALUControl.

**Task:**

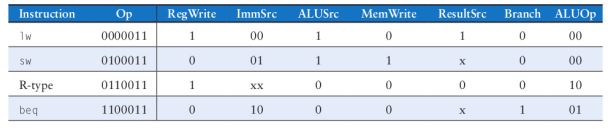
Write a Verilog Code for the RISC-V Control Unit by using the provided truth tables. Make two different modules for the Main decoder and ALU decoder. Use appropriate inputs and outputs for the modules. By making a test bench for each module check whether your design is correct or not. Attach the codes, test benches and the waveforms.

**Block Diagram:**

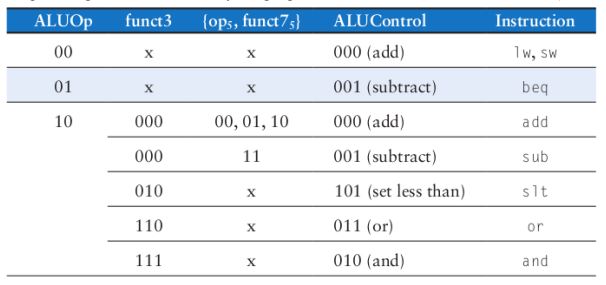
****

**Truth Table:**

**Main Decoder:**

****

**ALU Decoder:**

****

**Verilog Code:**

main\_decoder.v

module main\_decoder(op, RegWrite, MemWrite, Branch, ResultSrc, ALUSrc, ImmSrc, ALUOp);

    input[6:0] op;

    output RegWrite,ALUSrc,MemWrite,ResultSrc,Branch;

    output [1:0] ImmSrc,ALUOp;

    assign RegWrite = ((op == 7'b0000011) | (op == 7'b0110011)) ? 1'b1 : 1'b0;

    assign ALUSrc = ((op == 7'b0000011) | (op == 7'b0100011)) ? 1'b1 : 1'b0;

    assign MemWrite = ((op == 7'b0100011)) ? 1'b1 : 1'b0;

    assign ResultSrc = ((op == 7'b0000011)) ? 1'b1 : 1'b0;

    assign Branch = ((op == 7'b1100011)) ? 1'b1 : 1'b0;

    assign ImmSrc = ((op == 7'b0100011)) ? 2'b01 : (op == 7'b1100011) ? 2'b10 : 2'b00;

    assign ALUOp = ((op == 7'b0110011)) ? 2'b10 : (op == 7'b1100011) ? 2'b01 : 2'b00;

endmodule

alu\_decoder.v

module alu\_decoder(ALUOp, func3, op\_5, func7\_5, ALUControl);

    input[1:0] ALUOp;

    input[2:0] func3;

    input op\_5,func7\_5;

    wire[1:0] cont;

    output[2:0] ALUControl;

    assign cont = {op\_5 , func7\_5};

    assign ALUControl = (ALUOp == 2'b00) ? 3'b000 :

                        (ALUOp == 2'b01) ? 3'b001 :

                        ((ALUOp == 2'b10) & (func3 == 3'b000) & (cont == 2'b11))? 3'b001 :

                        ((ALUOp == 2'b10) & (func3 == 3'b000) & (cont != 2'b11))? 3'b000 :

                        ((ALUOp == 2'b10) & (func3 == 3'b010)) ? 3'b101 :

                        ((ALUOp == 2'b10) & (func3 == 3'b110)) ? 3'b011 :

                        ((ALUOp == 2'b10) & (func3 == 3'b111)) ? 3'b010 : 3'b000;

endmodule

control\_unit.v

`include "./modules/main\_decoder.v"

`include "./modules/alu\_decoder.v"

module control\_unit(Zero,op,func3,func7,PCSrc,ResultSrc,MemWrite,ALUSrc,ImmSrc,RegWrite,ALUControl);

    input Zero, func7;

    input [6:0] op;

    input [2:0] func3;

    output PCSrc, RegWrite, ALUSrc, MemWrite, ResultSrc;

    output [1:0] ImmSrc;

    output [2:0] ALUControl;

    wire Branch,op\_5;

    wire [1:0]ALUReg;

    assign op\_5 = op[5];

    main\_decoder main\_decoder(.op(op),

                                .RegWrite(RegWrite),

                                .MemWrite(MemWrite),

                                .Branch(Branch),

                                .ResultSrc(ResultSrc),

                                .ALUSrc(ALUSrc),

                                .ImmSrc(ImmSrc),

                                .ALUOp(ALUReg)

                                );

    alu\_decoder alu\_decoder(.ALUOp(ALUReg),

                            .func3(func3),

                            .op\_5(op\_5),

                            .func7\_5(func7),

                            .ALUControl(ALUControl)

                            );

    assign PCSrc = Zero & Branch;

endmodule

testbench.v

module tb();

    reg Zero, func7;

    reg [6:0] op;

    reg [2:0] func3;

    output PCSrc, RegWrite, ALUSrc, MemWrite, ResultSrc;

    output [1:0] ImmSrc;

    output [2:0] ALUControl;

    control\_unit dut (.Zero(Zero),

                        .op(op),

                        .func3(func3),

                        .func7(func7),

                        .PCSrc(PCSrc),

                        .ResultSrc(ResultSrc),

                        .MemWrite(MemWrite),

                        .ALUSrc(ALUSrc),

                        .ImmSrc(ImmSrc),

                        .RegWrite(RegWrite),

                        .ALUControl(ALUControl)

                        );

    initial begin

      $dumpfile("dump.vcd");

      $dumpvars(0);

    end

    initial begin

      Zero <= 1'b1;

      op <= 7'b0000011;

      func3 <= 3'b000;

      func7 <= 1'b1;

      #100;

      Zero <= 1'b1;

      op <= 7'b0100011;

      func3 <= 3'b000;

      func7 <= 1'b1;

      #100;

      Zero <= 1'b1;

      op <= 7'b0110011;

      func3 <= 3'b000;

      func7 <= 1'b1;

      #100;

      Zero <= 1'b1;

      op <= 7'b1100011;

      func3 <= 3'b000;

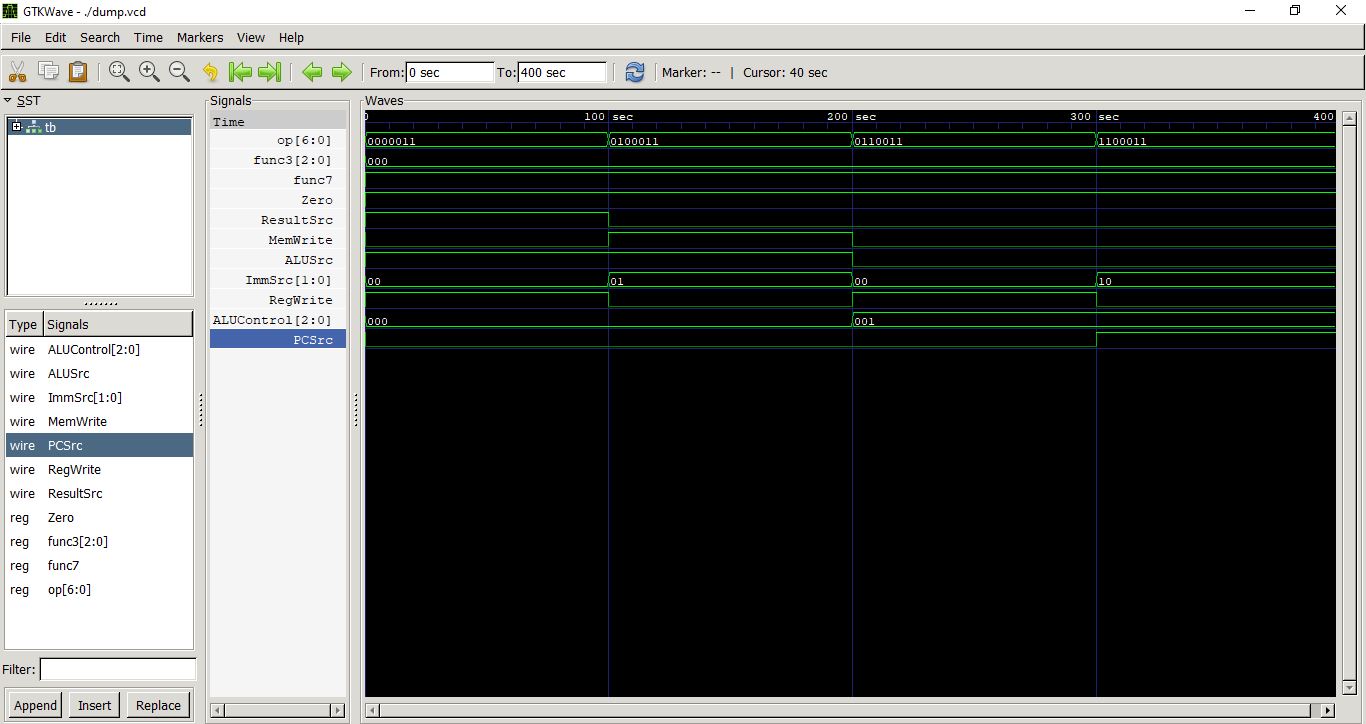
      func7 <= 1'b1;

      #100;

    end

endmodule

**Waveforms:**

****