

COAL Lab # 04

Designing Control Unit in Verilog

Name: Saad Nisar Butt

Reg. no: cs211246

Class: BSCS 3C1

Lab Task:

Literature Review:

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:

Table below is a truth table for the main decoder that summarizes the control signals as a function of the opcode. 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:

1. 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.

Truth Tables:

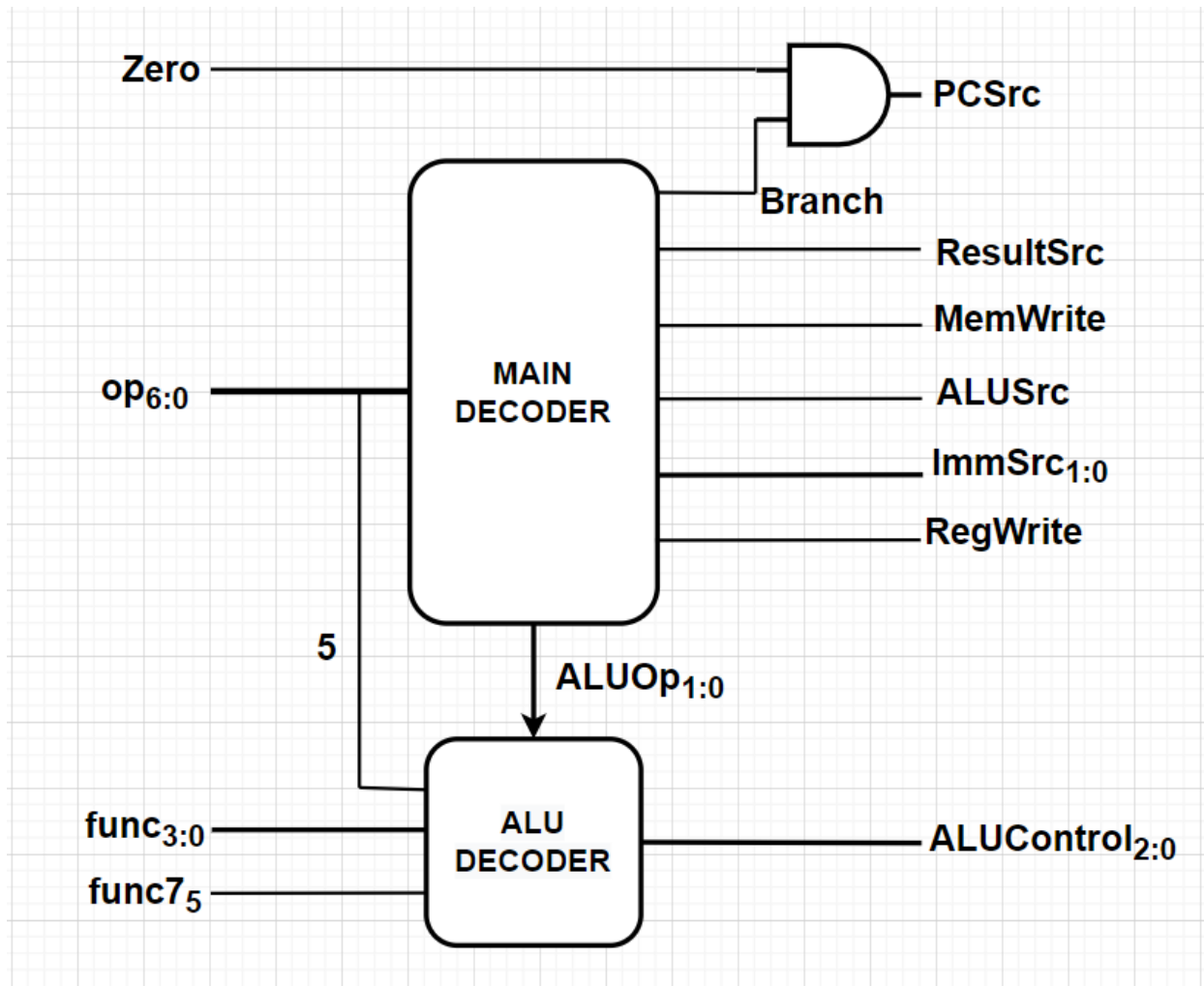
Main Decoder:

Op	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
0000011	1	00	1	0	1	0	00
0100011	0	01	1	1	x	0	00
0110011	1	xx	0	0	0	0	10
1100011	0	10	0	0	x	1	01

ALU Decoder:

ALUOp	func3	{op ₅ , func7 ₅ }	ALUControl	Instruction
00	x	x	000 (add)	lw, sw
01	x	x	001 (subtract)	beq
10	000	00,01,10	000 (add)	add
	000	11	001 (subtract)	sub
	010	x	101 (set less than)	slt
	110	x	011 (or)	or
	111	x	010 (and)	and

Block Diagram:



Verilog Code:

main_decoder.v

```
module main_decoder(op, RegWrite, ALUSrc, MemWrite, ResultSrc, Branch, 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, op5, func7_5, ALUControl);
    input [1:0] ALUOp;
    input [2:0] func3;
    input op5, func7_5;
    wire [1:0] signal;

    output [2:0] ALUControl;

    assign signal = {op5, func7_5};

    assign ALUControl = (ALUOp == 2'b00) ? 3'b000 :
                        (ALUOp == 2'b01) ? 3'b001 :
                        ((ALUOp == 2'b10) & (func3 == 3'b000) & (signal ==
2'b11)) ? 3'b001 :
                        ((ALUOp == 2'b10) & (func3 == 3'b000) & (signal !=
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'b011)) ? 3'b010 : 3'b000;

endmodule

```

control_unit.v

```

`include "./DecoderModules/main_decoder.v"
`include "./DecoderModules/alu_decoder.v"
module control_unit(zero, op, func3, func7, PCSrc, RegWrite, ALUSrc, MemWrite,
ResultSrc, ImmSrc, 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 [1:0] ALUOp;
wire op5, Branch;

assign op5 = op[5];

main_decoder main_dec (
    .op(op), .RegWrite(RegWrite), .ALUSrc(ALUSrc), .MemWrite(MemWrite),
    .ResultSrc(ResultSrc), .Branch(Branch), .ImmSrc(ImmSrc), .ALUOp(ALUOp)
);

alu_decoder alu_dec (
    .ALUOp(ALUOp), .func3(func3), .op5(op5), .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;
    wire PCSrc, RegWrite, ALUSrc, MemWrite, ResultSrc;
    wire [1:0] ImmSrc;
    wire [2:0] ALUControl;

    control_unit dut (
        .zero(zero),
        .op(op),
        .func3(func3),
        .func7(func7),

```

```

        .PCSrc(PCSrc),
        .RegWrite(RegWrite),
        .ALUSrc(ALUSrc),
        .MemWrite(MemWrite),
        .ResultSrc(ResultSrc),
        .ImmSrc(ImmSrc),
        .ALUControl(ALUControl)
    );

```

```

initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0);
end

```

```

initial begin
    zero <= 1'b1;
    op <= 7'b0000011;
    func3 <= 3'b000;
    func7 <= 1'b0;
    #100;
    // ALUOp 00
    // ALUControl 000

```

```

    zero <= 1'b1;
    op <= 7'b0100011;
    func3 <= 3'b000;
    func7 <= 1'b1;
    #100;
    // ALUOp 00
    // ALUControl 000

```

```

    zero <= 1'b1;
    op <= 7'b0110011;
    func3 <= 3'b000;
    func7 <= 1'b1;
    #100;
    // ALUOp 10
    // ALUControl 001

```

```

    zero <= 1'b1;
    op <= 7'b1100011;
    func3 <= 3'b000;
    func7 <= 1'b0;
    #100;
    // ALUOp 01

```

```

        // ALUControl 001

        zero <= 1'b1;
        op <= 7'b0110011;
        func3 <= 3'b010;
        func7 <= 1'b0;
        #100;
        // ALUOp 10
        // ALUControl 011

        zero <= 1'b1;
        op <= 7'b0110011;
        func3 <= 3'b110;
        func7 <= 1'b0;
        #100;
        // ALUOp 10
        // ALUControl 011

    end

endmodule

```

Waveforms:

