Alec Bakholdin
Submitted 11/14/2021

```
module instruction_memory(
    input logic clk, rst,
    input logic[2:0] A,

    output logic[31:0] RD
);
    reg[31:0] instruction_regs[4:0] = '{default: 32'b0};

    // this is just to make sure that if someone inputs 5, 6, or 7
    // (which are possible given logic[2:0]), that the output is defined
    logic[2:0] true_address;
    assign true_address = A > 4 ? 4 : A;

    always @(posedge clk or negedge rst)
    begin
        if(!rst) begin
            instruction_regs[0] <= 32'b0;
            instruction_regs[1] <= 32'b010101_00010_00001_00000000000000011;
            instruction_regs[2] <= 32'b010100_01000_00001_00000000000000001;
            instruction_regs[3] <= 32'b100100_00011_00100_00001_00000000000;
            instruction_regs[4] <= 32'b101100_01010_01000_00001_00000000000;
        end
        else begin
            RD <= instruction_regs[true_address];
        end
    end
endmodule
```

```
module register_file(
    input logic clk, rst,
    input logic [4:0] A1, A2, A3,
    input logic [31:0] WD3,
    input logic WE3,
    output logic [31:0] RD1, RD2, probe
);

    reg[31:0] registers [31:0];
    initial begin
       for(int i = 0; i < 32; i++) begin
          registers[i] <= i;
       end
    end

    assign RD1 = registers[A1];
    assign RD2 = registers[A2];
    assign probe = registers[A3];

    always @(posedge clk or negedge rst)
       if(!rst) begin
          for(int i = 0; i < 32; i++) begin
             registers[i] <= i;
          end
       end
       else if(WE3) begin;
          #1;
          registers[A3] <= WD3;
       end;


endmodule
```

```
module ALU(
    input logic[31:0] SrcA, SrcB,
    input logic[2:0] ALUControl,
    output logic[31:0] ALUResult
);

    always_comb
       case(ALUControl)
          3'b010: ALUResult <= SrcA + SrcB;
          3'b110: ALUResult <= SrcA - SrcB;
          default: ALUResult <= 0;
       endcase


endmodule
```

```
module MUX_RegDst(
    input logic RegDst,
    input logic[31:0] Instr,
    output logic[5:0] RegDst_out
);

    assign RegDst_out = RegDst ? Instr[15:11] : Instr[20:16];

endmodule
```

```
module MUX_MemtoReg(
    input logic MemtoReg,
    input logic[31:0] ALUResult, RD,
    output logic[31:0] MemtoReg_out
);

    assign MemtoReg_out = MemtoReg ? RD : ALUResult;

endmodule
```

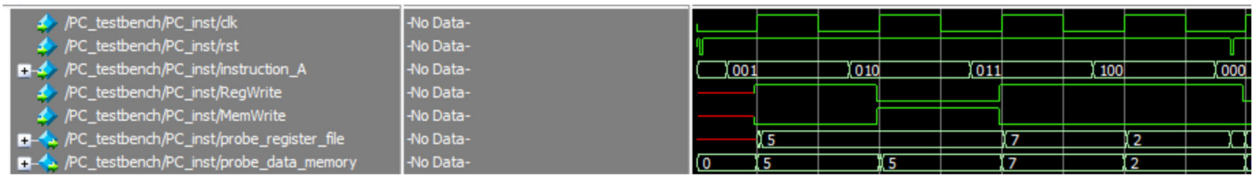
```
module MUX_ALUSrc(
    input logic ALUSrc,
    input logic[31:0] RD2, SignImm,
    output logic[31:0] ALUSrc_out
);

    assign ALUSrc_out = ALUSrc ? SignImm : RD2;

endmodule
```

```
module sign_extend(
    input logic[15:0] Imm,
    output logic[31:0] SignImm
);
    int intImm;
    assign intImm = shortint'(Imm);
    assign SignImm = intImm;

endmodule
```

```
module data_memory(
    input logic clk, rst,
    input logic[31:0] A, WD,
    input logic WE,

    output logic[31:0] RD, probe
);
    reg[31:0] memory [31:0];
    initial begin
        for(int i = 0; i < 32; i++) begin
            memory[i] <= i;
        end
    end

    logic[4:0] low_5_bits;
    assign low_5_bits = A[4:0];

    assign RD = memory[low_5_bits];
    assign probe = memory[low_5_bits];

    always @(posedge clk or negedge rst)
    begin
        if(!rst) begin
            for(int i = 0; i < 32; i++) begin
                memory[i] <= i;
            end
        end
        else if(WE) begin
          #1;
            memory[low_5_bits] <= WD;
        end
    end

endmodule
```

```
module PC_testbench;

    logic clk = 0, instr_clk = 0, rst = 1, RegWrite, MemWrite;
    logic[2:0] instruction_A = 0;
    logic[31:0] target_instruction;
    logic[31:0] probe_register_file, probe_data_memory;

    instruction_memory instruction_memory_inst(instr_clk, rst, instruction_A, target_instruction);
    PC PC_inst(clk, rst, instruction_A, RegWrite, MemWrite, probe_register_file, probe_data_memory);

    logic [5:0] opcode, sw_opcode = 6'b010100;
    assign opcode = target_instruction[31:26];
    assign RegWrite = opcode != 0 & opcode != sw_opcode;
    assign MemWrite = opcode == sw_opcode;

    initial begin
        #1; rst = 0; #1; rst = 1;
    end

    always
        begin
            instr_clk = 0;
            clk = 0; #10;
            instruction_A = (instruction_A + 1) % 5;
            if(instruction_A == 0) begin
                #5; rst = 0; #1; rst = 1; #3;
            end
            else #9;
            instr_clk = 1; #1;
            clk = 1; #20;
        end

endmodule
```

```
module PC(
    input logic clk, rst,
    input logic[2:0] instruction_A,
    input logic RegWrite, MemWrite,

    output logic[31:0] probe_register_file,
    output logic[31:0] probe_data_memory
);
    logic[31:0] instruction;
    instruction_memory instruction_memory_inst(clk, rst, instruction_A, instruction);

    // parsing opcode
    logic[5:0] opcode;
    assign opcode = instruction[31:26];
    logic RegDst, ALUSrc, MemtoReg;
    logic[2:0] ALUControl;
    assign RegDst = opcode[5];
    assign ALUSrc = opcode[4];
    assign ALUControl = opcode[3:1];
    assign MemtoReg = opcode[0];

    // register_file variables
    logic[31:0] RD1, RD2, WD3;
    logic[4:0] A1, A2, A3;
    assign A1 = instruction[25:21];
    assign A2 = instruction[20:16];
    MUX_RegDst MUX_RegDst_inst(RegDst, instruction, A3);
    register_file register_file_inst(clk, rst, A1, A2, A3, WD3, RegWrite, RD1, RD2, probe_register_file);

    // sign_extend variables
    logic[15:0] low_16_bits;
    logic[31:0] SignImm;
    assign low_16_bits = instruction[15:0];
    sign_extend sign_extend_inst(low_16_bits, SignImm);

    // ALU variables
    logic[31:0] SrcA, SrcB, ALUResult;
    assign SrcA = RD1;
    MUX_ALUSrc MUX_ALUSrc_inst(ALUSrc, RD2, SignImm, SrcB);
    ALU ALU_inst(SrcA, SrcB, ALUControl, ALUResult);

    // data_memory variables
    logic[4:0] A;
    logic[31:0] WD, RD;
    assign A = ALUResult;
    assign WD = RD2;
    data_memory data_memory_inst(clk, rst, A, WD, MemWrite, RD, probe_data_memory);

    // MemtoReg MUX
    MUX_MemtoReg MUX_MemtoReg_inst(MemtoReg, ALUResult, RD, WD3);

endmodule
```



I found this lab very cool! I've learned about circuits, transistors, combinational logic, then sequential logic, so I've learned a fair amount of how electronics work at a low level. Finally I have a step that links logic circuits to assembly, which I found fascinating. I've written my own little assembly compiler that compiled RISC-V code into 32-bit instructions to test different instructions. Again, all very cool.