CPE 233 Hardware Assignment 6

Assembling the MCU

Report by:
Ethan Vosburg (evosburg@calpoly.edu)

March 13, 2024

Table of Contents

1 Project Description

In this project the central component of a functional microcontroller (MCU) is constructed by integrating various OTTER modules developed in previous assignments. While the resulting MCU will not represent the complete OTTER system, it will serve as a foundational unit for future expansions. This assignment will build the foundation for the rest of the hardware assignments as the OTTER starts to take shape and we can begin to implement it on the Basys3 board.

1.1 Control Unit Decoder

The Control Unit Decoder is a combinational logic component responsible for generating control signals based on the current instruction. Unlike the Control Unit FSM, the signals generated by the decoder are less time-sensitive, allowing for a simpler, combinational design. The decoder takes the instruction opcode as input and outputs control signals to various components of the MCU, such as the ALU, registers, and memory units. The primary function of the decoder is to interpret the instruction and set the appropriate control signals to execute the operation specified by the instruction.

1.2 Control Unit FSM

The Control Unit FSM is responsible for controlling the timing-critical aspects of the MCU's operation. It operates in three main states: FETCH, EXEC, and WRITEBACK. The FSM ensures that instructions are fetched, executed, and the results are written back in a controlled manner, particularly for operations involving memory access.

1.3 OTTER MCU

The OTTER MCU is a microcontroller unit composed of various modules, including the Control Unit (with its FSM and Decoder), ALU, registers, and memory units. It is designed to execute a set of instructions, manipulating data and performing operations as specified by the program.

2 Structural Design

2.1 OTTER MCU

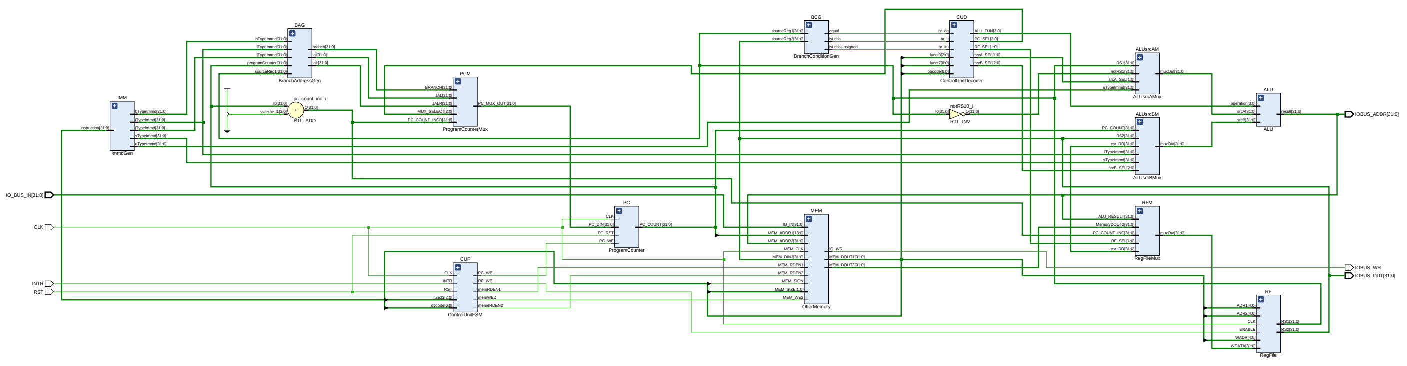
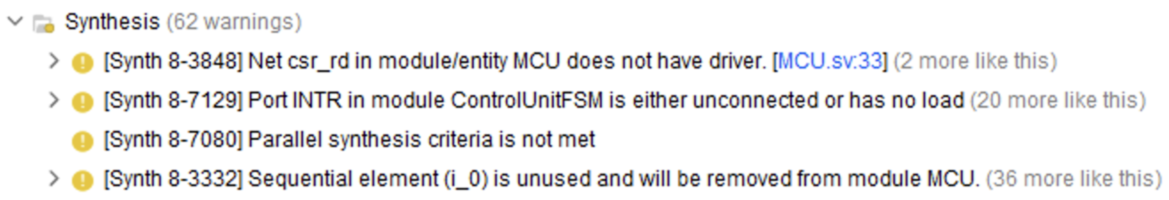


Figure 1: OTTER MCU Eleborated Design

3 Synthesis Warnings

3.1 MCU Synthesis Warnings



Synthesis (62 warnings)
[Synth 8-3848] Net csr_rd in module/entity MCU does not have driver. [MCU.sv:33] (2 more like this)
[Synth 8-7129] Port INTR in module ControlUnitFSM is either unconnected or has no load (20 more like this)
[Synth 8-7080] Parallel synthesis criteria is not met
[Synth 8-3332] Sequential element (i_0) is unused and will be removed from module MCU. (36 more like this)

Figure 2: Synthesis Warnings for MCU

The warnings referenced in Figure 2 are not important as there is hardware that will still need to be implemented in future assignments. This means there are various wires that do not have a driver or that do not lead to a defined location. This is fine and in this case does not prevent the design from working. Vivado is simply letting you know that these lines will not be implemented in to the design at this point.

4 Verification

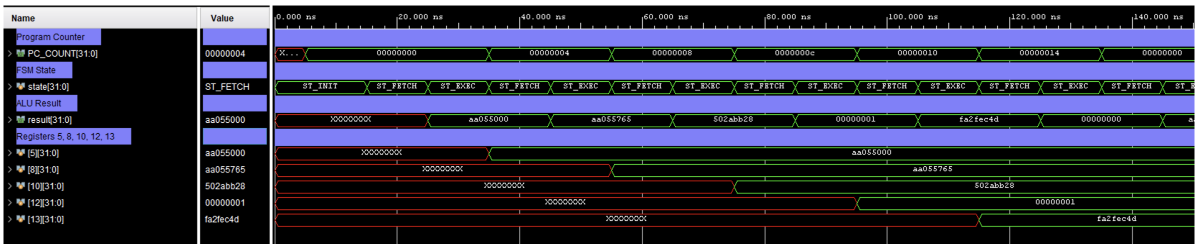
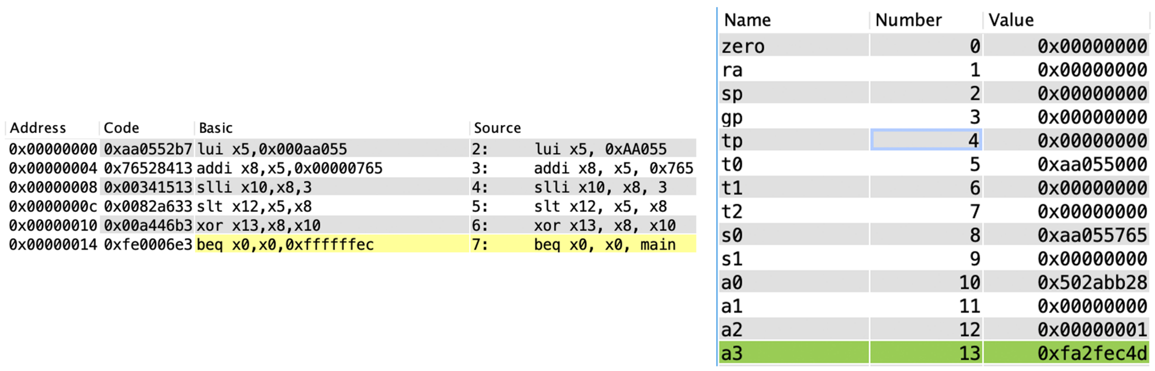


Figure 3: OTTER MCU Verification



Address	Code	Basic	Source
0x00000000	0xaa0552b7	lui x5,0x000aa055	2: lui x5, 0xAA055
0x00000004	0x76528413	addi x8,x5,0x00000765	3: addi x8, x5, 0x765
0x00000008	0x00341513	slli x10,x8,3	4: slli x10, x8, 3
0x0000000c	0x0082a633	slt x12,x5,x8	5: slt x12, x5, x8
0x00000010	0x00a446b3	xor x13,x8,x10	6: xor x13, x8, x10
0x00000014	0xfe0006e3	beq x0,x0,0xffffffec	7: beq x0, x0, main

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x00000000
gp	3	0x00000000
tp	4	0x00000000
t0	5	0xaa055000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0xaa055765
s1	9	0x00000000
a0	10	0x502abb28
a1	11	0x00000000
a2	12	0x00000001
a3	13	0xfa2fec4d

Figure 4: OTTER MCU RARS Simulation

Looking at the simulation we have the outputs that we would expect referencing figure 4 you can compare the hex values for the registers and see that they all match up. They registers have an undefined value before they are written to which is expected. Also There is a longer INIT state becuase of the reset command in the beginning of the file.

5 Source Code

5.1 Branch Address Generator Source Code

Listing 1: Control Unit FSM

```
'timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Ethan Vosburg
//
// Create Date: 02/21/2024 03:42:56 PM
// Module Name: ControlUnitFSM
// Project Name: Risc-V MCU
// Target Devices: Basys3
// Description: Control Unit FSM for the Risc-V MCU
//
// Revision:
// Revision 0.01 - File Created
//
//////////////////////////////////////////////////////////////////////////////////


module ControlUnitFSM(
    // Inputs
    input CLK,                          // Clock
    input RST,                          // Reset
    input INTR,                         // Interrupt
    input [6:0] opcode,                 // Opcode
    input [2:0] funct3,                 // Function 3

    // Outputs
    output logic PC_WE,                 // Program Counter Write Enable
    output logic RF_WE,                 // Register File Write Enable
    output logic memWE2,                // Memory Write Enable 2
    output logic memRDEN1,              // Memory Read Enable 1
    output logic memRDEN2,              // Memory Read Enable 2
    output logic reset,                 // Reset
    output logic csr_WE,                // Control and Status Register Write Enable
    output logic int_taken,             // Interrupt
    output logic mret_exec              // MRET Execution
);
    // Define the state type
    typedef enum { ST_INIT, ST_FETCH, ST_EXEC, ST_WB, ST_INTR} state_type;

    //  Define state and next state
    state_type state, next_state;

    // Transition State
    always_ff @(posedge CLK) begin
        if (RST) begin
            state <= ST_INIT;
        end else begin
            state <= next_state;
        end
    end


    // Define the state logic
    always_comb begin
        // Initialize all the outputs to zeros
        PC_WE = 1'b0;
        RF_WE = 1'b0;
        memWE2 = 1'b0;
        memRDEN1 = 1'b0;
        memRDEN2 = 1'b0;
        reset = 1'b0;
        csr_WE = 1'b0;
        int_taken = 1'b0;
        mret_exec = 1'b0;

        case (state)
            ST_INIT: begin
                reset = 1'b1;
                next_state = ST_FETCH;
            end
            ST_FETCH: begin
                memRDEN1 = 1'b1;
                next_state = ST_EXEC;
            end
            ST_EXEC: begin
                if(INTR && opcode != 7'b11)begin
                    next_state <= ST_INTR;
                end
                else if (opcode == 7'b11) begin
                    next_state <= ST_WB;
                end
                else begin
                    next_state <= ST_FETCH;
                end
                case (opcode)
                // J-Type Instruction
                7'b1101111: begin
                    PC_WE = 1'b1;
                    RF_WE = 1'b1;
                    next_state = ST_FETCH;
                end
                // R-Type Instruction
                7'b0110011: begin
                    PC_WE = 1'b1;
                    RF_WE = 1'b1;
                    next_state = ST_FETCH;
                end
                // I-Type Instruction
                7'b0010011: begin
                    PC_WE = 1'b1;
                    memRDEN1 = 1'b1;
                    RF_WE = 1'b1;
                    next_state = ST_FETCH;
                end
                // JALR Instruction
                7'b1100111: begin
                    PC_WE = 1'b1;
                    memRDEN1 = 1'b1;
                    RF_WE = 1'b1;
                    next_state = ST_FETCH;
                end
                // Loading Instructions
                7'b0000011: begin
                    PC_WE = 1;
                    memRDEN1 = 1'b1;
                    memRDEN2 = 1;
                    RF_WE = 1'b1;
                    next_state = ST_WB;
                end
                // B-Type Instruction
                7'b1100011: begin
                    PC_WE = 1'b1;
                    next_state = ST_FETCH;
                end
                // U-Type Instruction
                7'b0110111: begin
                    PC_WE = 1'b1;
                    RF_WE = 1'b1;
                    memRDEN1 = 1'b1;
                    next_state = ST_FETCH;
                end
                // AUIPC Instruction
                7'b0010111: begin
                    PC_WE = 1'b1;
                    RF_WE = 1'b1;
                    memRDEN1 = 1'b1;
                    next_state = ST_FETCH;
                end
                // S-Type Instructions
                7'b0100011: begin
                    PC_WE = 1'b1;
                    memRDEN1 = 1'b1;
                    memWE2 = 1;
                    next_state = ST_FETCH;
                end
                // Interrupt Logic
                7'b1110011: begin
                    PC_WE = 1'b1;
                    case (funct3)
                        3'b011: begin
                            RF_WE = 1;
                            csr_WE = 1;
                        end
                        3'b010: begin
                            RF_WE = 1;
                            csr_WE = 1;
                        end
                        3'b001: begin
                            RF_WE = 1;
                            csr_WE = 1;
                        end
                        3'b000: begin
                            mret_exec = 1;
                        end
                        default: csr_WE = 0;
                    endcase
                end

                default: next_state = ST_INIT; // Should never happen
                endcase
            end
            ST_WB: begin
                RF_WE = 1'b1;
                PC_WE = 0;
                if(INTR) begin
                    next_state <= ST_INTR;
                end
                else begin
                    next_state <= ST_FETCH;
                end
            end
            ST_INTR: begin
                PC_WE = 1;
                int_taken = 1;
                next_state = ST_FETCH;
            end
            default: next_state = ST_INIT; // Should never happen
        endcase
    end
endmodule
```

Listing 2: Control Unit Decoder

```
'timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Ethan Vosburg
//
// Create Date: 02/21/2024 03:47:15 PM
// Module Name: ControlUnitDecoder
// Project Name: Otter MCU
// Target Devices: Basys3
// Description: Control Unit Decoder for controlling the ALU
//
// Revision:
// Revision 0.01 - File Created
//
//////////////////////////////////////////////////////////////////////////////////


module ControlUnitDecoder(
    // Inputs
    input br_eq,                            // Branch Equal
    input br_lt,                            // Branch Less Than
    input br_ltu,                           // Branch Less Than Unsigned
    input [6:0] funct7,                     // Function 7
    input [6:0] opcode,                     // Opcode
    input [2:0] funct3,                     // Function 3
    input int_taken,                        // Interrupt taken flag

    // Outputs
    output logic [3:0] ALU_FUN,             // ALU Function
    output logic [1:0] srcA_SEL,            // Source A Select
    output logic [2:0] srcB_SEL,            // Source B Select
    output logic [2:0] PC_SEL,              // Program Counter Select
    output logic [1:0] RF_SEL               // Register File Select
    );

    always_comb begin
        // Initialize all the outputs to zeros
        ALU_FUN = 4'b0000;
        srcA_SEL = 2'b0;
        srcB_SEL = 3'b0;
        PC_SEL = 3'b0;
        RF_SEL = 2'b00;

        case (opcode)
        // R-Type Instruction
        7'b0110011: begin
            ALU_FUN = {funct7[5], funct3};
            srcA_SEL = 2'b0;
            srcB_SEL = 0;
            PC_SEL = 3'b0;
            RF_SEL = 2'b11;
            // Logic for ALU Select B
            // if ({funct7[5], funct3} == 4'b0010) srcB_SEL = 3'b1;
            // if ({funct7[5], funct3} == 4'b0100) srcB_SEL = 3'b0;
        end
        // I-Type Instruction
        7'b0010011: begin
            // Take care of special case with SRAI
            if (funct3 == 3'b101)
                ALU_FUN = {funct7[5], funct3};
            else
                ALU_FUN = {1'b0, funct3};

            // Configure selects
            srcA_SEL = 2'b0;
            srcB_SEL = 3'b1;
            RF_SEL = 2'b11;
            PC_SEL = 3'b0;
        end
        // JALR Instructions
        7'b1100111: begin
            PC_SEL = 3'b1;
            // ALU_FUN = {1'b0, funct3};
            // srcA_SEL = 2'b0;
            // srcB_SEL = 3'b1;
            RF_SEL = 2'b0;
        end
        // Loading Instructions
        7'b0000011: begin
            PC_SEL = 3'b0;
            RF_SEL = 2;
            srcA_SEL = 0;
            srcB_SEL = 1;
            ALU_FUN = 4'b0000;
        end
        // J-Type Instruction
        7'b1101111: begin
            PC_SEL = 3'b11;
            RF_SEL = 2'b0;
        end
        // B-Type Instruction
        7'b1100011: begin
            case(funct3)
                3'b000: begin
                    if (br_eq)
                        PC_SEL = 3'b10;
                    else
                        PC_SEL = 3'b0;
                end

                3'b101: begin
                    if (!br_lt)
                        PC_SEL = 3'b10;
                    else
                        PC_SEL = 3'b0;
                end

                3'b111: begin
                    if (!br_ltu)
                        PC_SEL = 3'b10;
                    else
                        PC_SEL = 3'b0;
                end

                3'b100: begin
                    if (br_lt)
                        PC_SEL = 3'b10;
                    else
                        PC_SEL = 3'b0;
                end

                3'b110: begin
                    if (br_ltu)
                        PC_SEL = 3'b10;
                    else
                        PC_SEL = 3'b0;
                end

                3'b001: begin
                    if (!br_eq)
                        PC_SEL = 3'b10;
                    else
                        PC_SEL = 3'b0;
                end

                default: PC_SEL = 3'b111;
            endcase
        end
        // U-Type Instruction
        7'b0110111: begin
            ALU_FUN = 4'b1001;
            srcA_SEL = 2'b1;
            PC_SEL = 3'b0;
            RF_SEL = 2'b11;
        end

        // AUIPC Command
        7'b0010111: begin
            ALU_FUN = 4'b0000;
            srcA_SEL = 2'b1;
            srcB_SEL = 3'b11;
            PC_SEL = 3'b0;
            RF_SEL = 2'b11;
        end
        // S-Type Instruction
        7'b0100011: begin
            ALU_FUN = 0;
            srcA_SEL = 0;
            srcB_SEL = 2;
            PC_SEL = 0;
        end
        // Interrupt functions
        7'b1110011: begin
            RF_SEL = 2'b01;
            srcB_SEL = 3'b100;

            case(funct3)
                // CSRRW
                3'b001: begin
                    ALU_FUN = 4'b1001;
                    srcA_SEL = 2'b00;
                end
                3'b010: begin
                    ALU_FUN = 4'b0110;
                    srcA_SEL = 2'b00;
                end
                3'b011:begin
                    ALU_FUN = 4'b0111;
                    srcA_SEL = 2'b10;
                    // RF_SEL = 2'b10;
                end
                3'b000: begin
                    PC_SEL = 5;
                end

            endcase


        end

        default: begin
            // Should not be used
            ALU_FUN = 4'b0000;
            srcA_SEL = 2'b1;
            srcB_SEL = 3'b11;
            PC_SEL = 3'b0;
            RF_SEL = 2'b11;
        end
        endcase

        if(int_taken)begin
            PC_SEL = 3'b100;
        end

    end


endmodule
```

Listing 3: Master MCU Linking all Modules Together

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Ethan Vosburg
//
// Create Date: 02/21/2024 03:42:56 PM
// Module Name: MCU
// Project Name: Risc-V MCU
// Target Devices: Basys3
// Description: MCU linking all the submodules together
//
// Revision:
// Revision 0.01 - File Created
//
//////////////////////////////////////////////////////////////////////////////////


module MCU(
    // Inputs
    input CLK,                          // Clock
    input RST,                          // Reset
    input INTR,                         // Interrupt
    input [31:0] IO_BUS_IN,             // IO Bus In

    // Outputs
    output [31:0] IOBUS_OUT,            // IO Bus Out
    output [31:0] IOBUS_ADDR,           // IO Bus Address
    output IOBUS_WR                     // IO Bus Write/Read Data
    );

    // Internal Wires
    logic [31:0] pc_mux, pc_count, pc_count_inc, alu_result, rs1, rs2, ir, ir2,
            rf_mux, utype, itype, stype, jtype, btype, csr_rd, jalr, branch,
            jal,alu_srca_mux, alu_srcb_mux, mtvec, mepc;
    logic pc_we, rf_we, memwe2, memrden1, memrden2, reset, csr_we, int_taken,
            mret_exec, mstatus;
    logic [3:0] alu_fun;
    logic [1:0] alu_srca_mux_select, rf_mux_select;
    logic [2:0] alu_srcb_mux_select, pc_mux_select;

    // Submodules
    // Assign statments for single lines
    assign IOBUS_OUT = rs2;
    assign IOBUS_ADDR = alu_result;
    assign pc_count_inc = pc_count + 4;
    assign intr = INTR & mstatus;

    // Linking all modules together
    ProgramCounter PC(
        .PC_RST(RST),
        .PC_WE(pc_we),
        .PC_DIN(pc_mux),
        .CLK(CLK),
        .PC_COUNT(pc_count)
    );

    ProgramCounterMux PCM(
        .PC_COUNT_INCD(pc_count_inc),
        .JALR(jalr),
        .BRANCH(branch),
        .JAL(jal),
        .MTVEC(mtvec),
        .MEPC(mepc),
        .MUX_SELECT(pc_mux_select),
        .PC_MUX_OUT(pc_mux)
    );

    OtterMemory MEM(
        .MEM_CLK(CLK),
        .MEM_RDEN1(memrden1),
        .MEM_RDEN2(memrden2),
        .MEM_WE2(memwe2),
        .MEM_ADDR1(pc_count[15:2]),
        .MEM_ADDR2(alu_result),
        .MEM_DIN2(rs2),
        .MEM_SIZE(ir[13:12]),
        .MEM_SIGN(ir[14]),
        .IO_IN(IO_BUS_IN),
        .IO_WR(IOBUS_WR),
        .MEM_DOUT1(ir),
        .MEM_DOUT2(ir2)
        );

    RegFile RF(
        .CLK(CLK),
        .ADR1(ir[19:15]),
        .ADR2(ir[24:20]),
        .WADR(ir[11:7]),
        .ENABLE(rf_we),
        .WDATA(rf_mux),
        .RS1(rs1),
        .RS2(rs2)
    );

    ImmdGen IMM(
        .instruction(ir),
        .uTypeImmd(utype),
        .iTypeImmd(itype),
        .sTypeImmd(stype),
        .jTypeImmd(jtype),
        .bTypeImmd(btype)
    );

    BranchAddressGen BAG(
        .programCounter(pc_count),
        .jTypeImmd(jtype),
        .bTypeImmd(btype),
        .iTypeImmd(itype),
        .sourceReg1(rs1),
        .jal(jal),
        .branch(branch),
        .jalr(jalr)
    );

    ALU ALU(
        .srcA(alu_srca_mux),
        .srcB(alu_srcb_mux),
        .operation(alu_fun),
        .result(alu_result)
    );

    BranchConditionGen BCG(
        .sourceReg1(rs1),
        .sourceReg2(rs2),
        .equal(br_eq),
        .isLess(br_lt),
        .isLessUnsigned(br_ltu)
    );

    ControlUnitDecoder CUD(
        .br_eq(br_eq),
        .br_lt(br_lt),
        .br_ltu(br_ltu),
        .funct7(ir[31:25]),
        .opcode(ir[6:0]),
        .funct3(ir[14:12]),
        .int_taken(int_taken),
        .ALU_FUN(alu_fun),
        .srcA_SEL(alu_srca_mux_select),
        .srcB_SEL(alu_srcb_mux_select),
        .PC_SEL(pc_mux_select),
        .RF_SEL(rf_mux_select)
    );

    ControlUnitFSM CUF(
        .CLK(CLK),
        .RST(RST),
        .INTR(intr),
        .opcode(ir[6:0]),
        .funct3(ir[14:12]),
        .PC_WE(pc_we),
        .RF_WE(rf_we),
        .memWE2(memwe2),
        .memRDEN1(memrden1),
        .memRDEN2(memrden2),
        .reset(reset),
        .csr_WE(csr_we),
        .int_taken(int_taken),
        .mret_exec(mret_exec)
    );

    RegFileMux RFM(
        .RF_SEL(rf_mux_select),
        .PC_COUNT_INC(pc_count_inc),
        .csr_RD(csr_rd),
        .MemoryDOUT2(ir2),
        .ALU_RESULT(alu_result),
        .muxOut(rf_mux)
    );

    ALUsrcAMux ALUsrcAM(
        .srcA_SEL(alu_srca_mux_select),
        .RS1(rs1),
        .uTypeImmd(utype),
        .notRS1(~rs1),
        .muxOut(alu_srca_mux)
    );

    ALUsrcBMux ALUsrcBM(
        .srcB_SEL(alu_srcb_mux_select),
        .RS2(rs2),
        .iTypeImmd(itype),
        .sTypeImmd(stype),
        .PC_COUNT(pc_count),
        .csr_RD(csr_rd),
        .muxOut(alu_srcb_mux)
    );

    CSR CSR(
        .CLK(CLK),
        .RESET(RST),
        .MRET_EXEC(mret_exec),
        .INT_TAKEN(int_taken),
        .ADDR(ir[31:20]),
        .CSR_WE(csr_we),
        .PC(pc_count),
        .WD(alu_result),
        .MSTATUS(mstatus),
        .MEPC(mepc),
        .MTVEC(mtvec),
        .RD(csr_rd)
    );

endmodule
```

6 Conclusion

In this project, the MCU was sussesfully constructed and tested with a limited set of code. The ground work for future hardware compnents was made tested to be working. This is the culmination of all the modules that have been made to this point and represents the first time that the compnents have all worked together. From here more moduels can be added to add more funcitonality. All code for this assignment can be found here.