



**CAL POLY**

# **CPE 233 Hardware Assignment 6**

***Assembling the MCU***

Report by:

Ethan Vosburg (evosburg@calpoly.edu)

March 13, 2024

# Table of Contents

<b>1 Project Description .....</b>	<b>3</b>
<b>1.1 Control Unit Decoder .....</b>	<b>3</b>
<b>1.2 Control Unit FSM.....</b>	<b>3</b>
<b>1.3 OTTER MCU.....</b>	<b>3</b>
<b>2 Structural Design.....</b>	<b>4</b>
<b>2.1 OTTER MCU.....</b>	<b>4</b>
<b>3 Synthesis Warnings.....</b>	<b>4</b>
<b>3.1 MCU Synthesis Warnings .....</b>	<b>4</b>
<b>4 Verification .....</b>	<b>5</b>
<b>5 Source Code .....</b>	<b>6</b>
<b>5.1 Branch Address Generator Source Code .....</b>	<b>6</b>
<b>6 Conclusion.....</b>	<b>15</b>

# 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

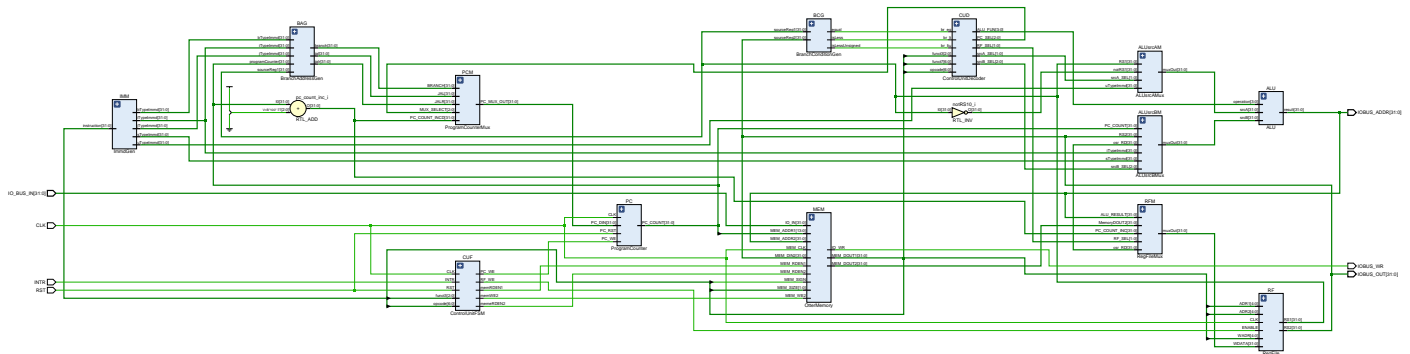


Figure 1: OTTER MCU Elaborated Design

## 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

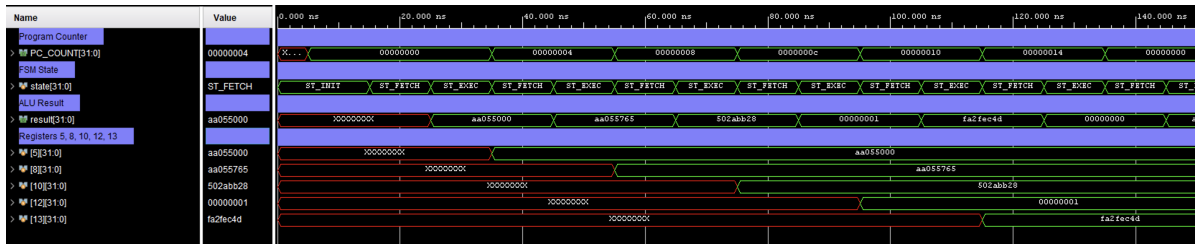


Figure 3: OTTER MCU Verification

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	0xfe006e3	beq x0, x0, 0xfffffec	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 because 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**

```

1  'timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Company: Cal Poly SLO
4  // Engineer: Ethan Vosburg
5  //
6  // Create Date: 02/21/2024 03:42:56 PM
7  // Module Name: ControlUnitFSM
8  // Project Name: Risc-V MCU
9  // Target Devices: Basys3
10 // Description: Control Unit FSM for the Risc-V MCU
11 //
12 // Revision:
13 // Revision 0.01 - File Created
14 //
15 ///////////////////////////////////////////////////////////////////
16
17
18 module ControlUnitFSM(
19     // Inputs
20     input CLK,                // Clock
21     input RST,                // Reset
22     input INTR,               // Interrupt
23     input [6:0] opcode,       // Opcode
24     input [2:0] funct3,       // Function 3
25
26     // Outputs
27     output logic PC_WE,       // Program Counter Write Enable
28     output logic RF_WE,       // Register File Write Enable
29     output logic memWE2,      // Memory Write Enable 2
30     output logic memRDEN1,    // Memory Read Enable 1
31     output logic memRDEN2,    // Memory Read Enable 2
32     output logic reset,       // Reset
33     output logic csr_WE,      // Control and Status Register Write Enable
34     output logic int_taken,    // Interrupt
35     output logic mret_exec    // MRET Execution
36 );
37
38 // Define the state type
39 typedef enum { ST_INIT, ST_FETCH, ST_EXEC, ST_WB, ST_INTR } state_type;
40
41 // Define state and next state
42 state_type state, next_state;
43
44 // Transition State
45 always_ff @(posedge CLK) begin
46     if (RST) begin
47         state <= ST_INIT;
48     end else begin
49         state <= next_state;
50     end
51 end
52
53 // Define the state logic
54 always_comb begin
55     // Initialize all the outputs to zeros
56     PC_WE = 1'b0;
57     RF_WE = 1'b0;
58     memWE2 = 1'b0;
59     memRDEN1 = 1'b0;
60     memRDEN2 = 1'b0;

```

```
61 reset = 1'b0;
62 csi_WE = 1'b0;
63 int_taken = 1'b0;
64 mret_exec = 1'b0;
65
66 case (state)
67 ST_INIT: begin
68     reset = 1'b1;
69     next_state = ST_FETCH;
70 end
71 ST_FETCH: begin
72     memRDEN1 = 1'b1;
73     next_state = ST_EXEC;
74 end
75 ST_EXEC: begin
76     if(INTR && opcode != 7'b11)begin
77         next_state <= ST_INTR;
78     end
79     else if (opcode == 7'b11) begin
80         next_state <= ST_WB;
81     end
82     else begin
83         next_state <= ST_FETCH;
84     end
85     case (opcode)
86     // J-Type Instruction
87     7'b1101111: begin
88         PC_WE = 1'b1;
89         RF_WE = 1'b1;
90         next_state = ST_FETCH;
91     end
92     // R-Type Instruction
93     7'b0110011: begin
94         PC_WE = 1'b1;
95         RF_WE = 1'b1;
96         next_state = ST_FETCH;
97     end
98     // I-Type Instruction
99     7'b0010011: begin
100         PC_WE = 1'b1;
101         memRDEN1 = 1'b1;
102         RF_WE = 1'b1;
103         next_state = ST_FETCH;
104     end
105     // JALR Instruction
106     7'b1100111: begin
107         PC_WE = 1'b1;
108         memRDEN1 = 1'b1;
109         RF_WE = 1'b1;
110         next_state = ST_FETCH;
111     end
112     // Loading Instructions
113     7'b0000011: begin
114         PC_WE = 1;
115         memRDEN1 = 1'b1;
116         memRDEN2 = 1;
117         RF_WE = 1'b1;
118         next_state = ST_WB;
119     end
120     // B-Type Instruction
121     7'b1100011: begin
122         PC_WE = 1'b1;
123         next_state = ST_FETCH;
124     end
125     // U-Type Instruction
126     7'b0110111: begin
127         PC_WE = 1'b1;
128         RF_WE = 1'b1;
129         memRDEN1 = 1'b1;
130         next_state = ST_FETCH;
```

```
131     end
132     // AUIPC Instruction
133     7'b0010111: begin
134         PC_WE = 1'b1;
135         RF_WE = 1'b1;
136         memRDEN1 = 1'b1;
137         next_state = ST_FETCH;
138     end
139     // S-Type Instructions
140     7'b0100011: begin
141         PC_WE = 1'b1;
142         memRDEN1 = 1'b1;
143         memWE2 = 1;
144         next_state = ST_FETCH;
145     end
146     // Interrupt Logic
147     7'b1110011: begin
148         PC_WE = 1'b1;
149         case (funct3)
150             3'b011: begin
151                 RF_WE = 1;
152                 csr_WE = 1;
153             end
154             3'b010: begin
155                 RF_WE = 1;
156                 csr_WE = 1;
157             end
158             3'b001: begin
159                 RF_WE = 1;
160                 csr_WE = 1;
161             end
162             3'b000: begin
163                 mret_exec = 1;
164             end
165             default: csr_WE = 0;
166         endcase
167     end
168
169     default: next_state = ST_INIT; // Should never happen
170 endcase
171 end
172 ST_WB: begin
173     RF_WE = 1'b1;
174     PC_WE = 0;
175     if(INTR) begin
176         next_state <= ST_INTR;
177     end
178     else begin
179         next_state <= ST_FETCH;
180     end
181 end
182 ST_INTR: begin
183     PC_WE = 1;
184     int_taken = 1;
185     next_state = ST_FETCH;
186 end
187 default: next_state = ST_INIT; // Should never happen
188 endcase
189 end
190 endmodule
```



## Listing 2: Control Unit Decoder

```

1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Company: Cal Poly SLO
4  // Engineer: Ethan Vosburg
5  //
6  // Create Date: 02/21/2024 03:47:15 PM
7  // Module Name: ControlUnitDecoder
8  // Project Name: Otter MCU
9  // Target Devices: Basys3
10 // Description: Control Unit Decoder for controlling the ALU
11 //
12 // Revision:
13 // Revision 0.01 - File Created
14 //
15 ///////////////////////////////////////////////////////////////////
16
17 module ControlUnitDecoder(
18     // Inputs
19     input br_eq,           // Branch Equal
20     input br_lt,           // Branch Less Than
21     input br_ltu,          // Branch Less Than Unsigned
22     input [6:0] funct7,    // Function 7
23     input [6:0] opcode,    // Opcode
24     input [2:0] funct3,    // Function 3
25     input int_taken,       // Interrupt taken flag
26
27     // Outputs
28     output logic [3:0] ALU_FUN, // ALU Function
29     output logic [1:0] srcA_SEL, // Source A Select
30     output logic [2:0] srcB_SEL, // Source B Select
31     output logic [2:0] PC_SEL,   // Program Counter Select
32     output logic [1:0] RF_SEL,   // Register File Select
33 );
34
35 always_comb begin
36     // Initialize all the outputs to zeros
37     ALU_FUN = 4'b0000;
38     srcA_SEL = 2'b0;
39     srcB_SEL = 3'b0;
40     PC_SEL = 3'b0;
41     RF_SEL = 2'b00;
42
43     case (opcode)
44         // R-Type Instruction
45         7'b0110011: begin
46             ALU_FUN = {funct7[5], funct3};
47             srcA_SEL = 2'b0;
48             srcB_SEL = 0;
49             PC_SEL = 3'b0;
50             RF_SEL = 2'b11;
51             // Logic for ALU Select B
52             // if ({funct7[5], funct3} == 4'b0010) srcB_SEL = 3'b1;
53             // if ({funct7[5], funct3} == 4'b0100) srcB_SEL = 3'b0;
54         end
55         // I-Type Instruction
56         7'b0010011: begin
57             // Take care of special case with SRAI
58             if (funct3 == 3'b101)
59                 ALU_FUN = {funct7[5], funct3};
60             else
61                 ALU_FUN = {1'b0, funct3};
62
63             // Configure selects
64             srcA_SEL = 2'b0;
65             srcB_SEL = 3'b1;
66             RF_SEL = 2'b11;
67             PC_SEL = 3'b0;
68         end
69     end

```

```
70 // JALR Instructions
71 7'b1100111: begin
72     PC_SEL = 3'b1;
73     // ALU_FUN = {1'b0, funct3};
74     // srcA_SEL = 2'b0;
75     // srcB_SEL = 3'b1;
76     RF_SEL = 2'b0;
77 end
78 // Loading Instructions
79 7'b0000011: begin
80     PC_SEL = 3'b0;
81     RF_SEL = 2;
82     srcA_SEL = 0;
83     srcB_SEL = 1;
84     ALU_FUN = 4'b0000;
85 end
86 // J-Type Instruction
87 7'b1101111: begin
88     PC_SEL = 3'b11;
89     RF_SEL = 2'b0;
90 end
91 // B-Type Instruction
92 7'b1100011: begin
93     case(funct3)
94         3'b000: begin
95             if (br_eq)
96                 PC_SEL = 3'b10;
97             else
98                 PC_SEL = 3'b0;
99         end
100
101         3'b101: begin
102             if (!br_lt)
103                 PC_SEL = 3'b10;
104             else
105                 PC_SEL = 3'b0;
106         end
107
108         3'b111: begin
109             if (!br_ltu)
110                 PC_SEL = 3'b10;
111             else
112                 PC_SEL = 3'b0;
113         end
114
115         3'b100: begin
116             if (br_lt)
117                 PC_SEL = 3'b10;
118             else
119                 PC_SEL = 3'b0;
120         end
121
122         3'b110: begin
123             if (br_ltu)
124                 PC_SEL = 3'b10;
125             else
126                 PC_SEL = 3'b0;
127         end
128
129         3'b001: begin
130             if (!br_eq)
131                 PC_SEL = 3'b10;
132             else
133                 PC_SEL = 3'b0;
134         end
135
136         default: PC_SEL = 3'b111;
137     endcase
138 end
139 // U-Type Instruction
```

```
140     7'b0110111: begin
141         ALU_FUN = 4'b1001;
142         srcA_SEL = 2'b1;
143         PC_SEL = 3'b0;
144         RF_SEL = 2'b11;
145     end
146
147     // AUIPC Command
148     7'b0010111: begin
149         ALU_FUN = 4'b0000;
150         srcA_SEL = 2'b1;
151         srcB_SEL = 3'b11;
152         PC_SEL = 3'b0;
153         RF_SEL = 2'b11;
154     end
155     // S-Type Instruction
156     7'b0100011: begin
157         ALU_FUN = 0;
158         srcA_SEL = 0;
159         srcB_SEL = 2;
160         PC_SEL = 0;
161     end
162     // Interrupt functions
163     7'b1110011: begin
164         RF_SEL = 2'b01;
165         srcB_SEL = 3'b100;
166
167         case(func3)
168             // CSRRW
169             3'b001: begin
170                 ALU_FUN = 4'b1001;
171                 srcA_SEL = 2'b00;
172             end
173             3'b010: begin
174                 ALU_FUN = 4'b0110;
175                 srcA_SEL = 2'b00;
176             end
177             3'b011: begin
178                 ALU_FUN = 4'b0111;
179                 srcA_SEL = 2'b10;
180                 // RF_SEL = 2'b10;
181             end
182             3'b000: begin
183                 PC_SEL = 5;
184             end
185         endcase
186     endcase
187
188 end
189
190 default: begin
191     // Should not be used
192     ALU_FUN = 4'b0000;
193     srcA_SEL = 2'b1;
194     srcB_SEL = 3'b11;
195     PC_SEL = 3'b0;
196     RF_SEL = 2'b11;
197 end
198 endcase
199
200 if(int_taken) begin
201     PC_SEL = 3'b100;
202 end
203
204 end
205
206
207 endmodule
```

**Listing 3: Master MCU Linking all Modules Together**

```

1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Company: Cal Poly SLO
4  // Engineer: Ethan Vosburg
5  //
6  // Create Date: 02/21/2024 03:42:56 PM
7  // Module Name: MCU
8  // Project Name: Risc-V MCU
9  // Target Devices: Basys3
10 // Description: MCU linking all the submodules together
11 //
12 // Revision:
13 // Revision 0.01 - File Created
14 //
15 ///////////////////////////////////////////////////////////////////
16
17 module MCU(
18     // Inputs
19     input CLK,                // Clock
20     input RST,                // Reset
21     input INTR,               // Interrupt
22     input [31:0] IO_BUS_IN,   // IO Bus In
23
24     // Outputs
25     output [31:0] IOBUS_OUT,   // IO Bus Out
26     output [31:0] IOBUS_ADDR,  // IO Bus Address
27     output IOBUS_WR            // IO Bus Write/Read Data
28 );
29
30 // Internal Wires
31 logic [31:0] pc_mux, pc_count, pc_count_inc, alu_result, rs1, rs2, ir, ir2,
32             rf_mux, utype, itype, stype, jtype, btype, csr_rd, jalr, branch,
33             jal, alu_srca_mux, alu_srcb_mux, mtvec, mepc;
34 logic pc_we, rf_we, memwe2, memrden1, memrden2, reset, csr_we, int_taken,
35       mret_exec, mstatus;
36 logic [3:0] alu_fun;
37 logic [1:0] alu_srca_mux_select, rf_mux_select;
38 logic [2:0] alu_srcb_mux_select, pc_mux_select;
39
40 // Submodules
41 // Assign statements for single lines
42 assign IOBUS_OUT = rs2;
43 assign IOBUS_ADDR = alu_result;
44 assign pc_count_inc = pc_count + 4;
45 assign intr = INTR & mstatus;
46
47 // Linking all modules together
48 ProgramCounter PC(
49     .PC_RST(RST),
50     .PC_WE(pc_we),
51     .PC_DIN(pc_mux),
52     .CLK(CLK),
53     .PC_COUNT(pc_count)
54 );
55
56 ProgramCounterMux PCM(
57     .PC_COUNT_INCD(pc_count_inc),
58     .JALR(jalr),
59     .BRANCH(branch),
60     .JAL(jal),
61     .MTVEC(mtvec),
62     .MEPC(mepc),
63     .MUX_SELECT(pc_mux_select),
64     .PC_MUX_OUT(pc_mux)
65 );
66
67 OtterMemory MEM(
68     .MEM_CLK(CLK),

```

```
70     .MEM_RDEN1(memrden1),
71     .MEM_RDEN2(memrden2),
72     .MEM_WE2(memwe2),
73     .MEM_ADDR1(pc_count[15:2]),
74     .MEM_ADDR2(alu_result),
75     .MEM_DIN2(rs2),
76     .MEM_SIZE(ir[13:12]),
77     .MEM_SIGN(ir[14]),
78     .IO_IN(IO_BUS_IN),
79     .IO_WR(IOBUS_WR),
80     .MEM_DOUT1(ir),
81     .MEM_DOUT2(ir2)
82 );
83
84 RegFile RF(
85     .CLK(CLK),
86     .ADR1(ir[19:15]),
87     .ADR2(ir[24:20]),
88     .WADR(ir[11:7]),
89     .ENABLE(rf_we),
90     .WDATA(rf_mux),
91     .RS1(rs1),
92     .RS2(rs2)
93 );
94
95 ImmdGen IMM(
96     .instruction(ir),
97     .uTypeImmd(utype),
98     .iTypeImmd(itype),
99     .sTypeImmd(stype),
100    .jTypeImmd(jtype),
101    .bTypeImmd(btype)
102 );
103
104 BranchAddressGen BAG(
105     .programCounter(pc_count),
106     .jTypeImmd(jtype),
107     .bTypeImmd(btype),
108     .iTypeImmd(itype),
109     .sourceReg1(rs1),
110     .jal(jal),
111     .branch(branch),
112     .jalr(jalr)
113 );
114
115 ALU ALU(
116     .srcA(alu_srca_mux),
117     .srcB(alu_srcb_mux),
118     .operation(alu_fun),
119     .result(alu_result)
120 );
121
122 BranchConditionGen BCG(
123     .sourceReg1(rs1),
124     .sourceReg2(rs2),
125     .equal(br_eq),
126     .isLess(br_lt),
127     .isLessUnsigned(br_ltu)
128 );
129
130 ControlUnitDecoder CUD(
131     .br_eq(br_eq),
132     .br_lt(br_lt),
133     .br_ltu(br_ltu),
134     .funct7(ir[31:25]),
135     .opcode(ir[6:0]),
136     .funct3(ir[14:12]),
137     .int_taken(int_taken),
138     .ALU_FUN(alu_fun),
139     .srcA_SEL(alu_srca_mux_select),
```

```
140         .srcB_SEL(alu_srcb_mux_select),
141         .PC_SEL(pc_mux_select),
142         .RF_SEL(rf_mux_select)
143     );
144
145     ControlUnitFSM CUF(
146         .CLK(CLK),
147         .RST(RST),
148         .INTR(int1),
149         .opcode(ir[6:0]),
150         .funct3(ir[14:12]),
151         .PC_WE(pc_we),
152         .RF_WE(rf_we),
153         .memWE2(memwe2),
154         .memRDEN1(memrden1),
155         .memRDEN2(memrden2),
156         .reset(reset),
157         .csr_WE(csr_we),
158         .int_taken(int_taken),
159         .mret_exec(mret_exec)
160     );
161
162     RegFileMux RFM(
163         .RF_SEL(rf_mux_select),
164         .PC_COUNT_INC(pc_count_inc),
165         .csr_RD(csr_rd),
166         .MemoryDOUT2(ir2),
167         .ALU_RESULT(alu_result),
168         .muxOut(rf_mux)
169     );
170
171     ALUSrcAMux ALUSrcAM(
172         .srcA_SEL(alu_srca_mux_select),
173         .RS1(rs1),
174         .uTypeImmd(utype),
175         .notRS1(~rs1),
176         .muxOut(alu_srca_mux)
177     );
178
179     ALUSrcBMux ALUSrcBM(
180         .srcB_SEL(alu_srcb_mux_select),
181         .RS2(rs2),
182         .iTypeImmd(itype),
183         .sTypeImmd(stype),
184         .PC_COUNT(pc_count),
185         .csr_RD(csr_rd),
186         .muxOut(alu_srcb_mux)
187     );
188
189     CSR CSR(
190         .CLK(CLK),
191         .RESET(RST),
192         .MRET_EXEC(mret_exec),
193         .INT_TAKEN(int_taken),
194         .ADDR(ir[31:20]),
195         .CSR_WE(csr_we),
196         .PC(pc_count),
197         .WD(alu_result),
198         .MSTATUS(mstatus),
199         .MEPC(mepc),
200         .MTVEC(mtvec),
201         .RD(csr_rd)
202     );
203
204 endmodule
```

## 6 Conclusion

In this project, the MCU was successfully constructed and tested with a limited set of code. The ground work for future hardware components 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 components have all worked together. From here more modules can be added to add more functionality. All code for this assignment can be found [here](#).