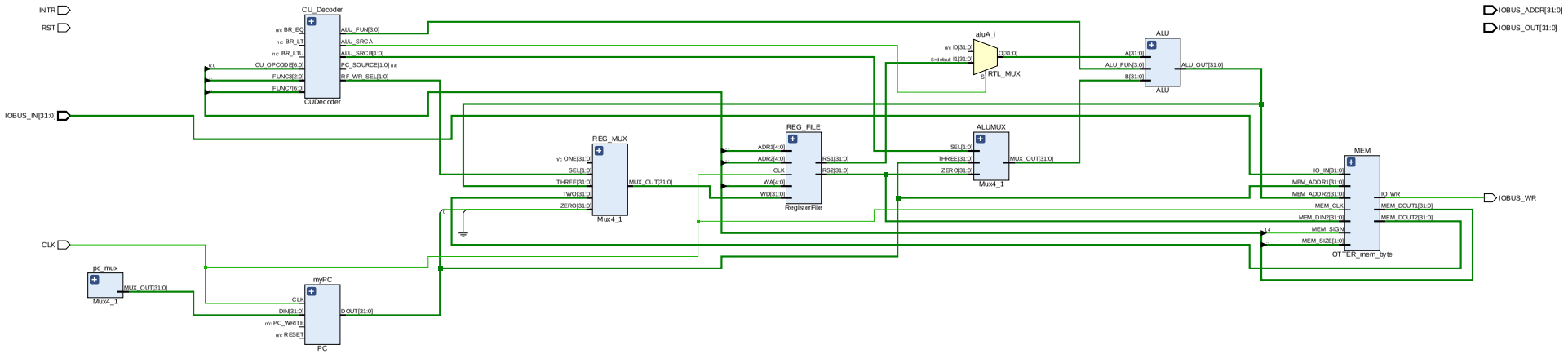
INTR
RST
IOBUS_IN[31:0]
CLK
CU_Decoder
BR_EQ
BR_LT
BR_LTU
CU_OPCODE[6:0]
FUNC3[2:0]
FUNC7[6:0]
ALU_FUN[3:0]
ALU_SRCA
ALU_SRCB[1:0]
PC_SOURCE[1:0]
RF_WR_SEL[1:0]
CUDecoder
pc_mux
Mux4_1
MUX_OUT[31:0]
myPC
CLK
DIN[31:0]
PC_WRITE
RESET
DOUT[31:0]
PC
REG_MUX
ONE[31:0]
SEL[1:0]
THREE[31:0]
TWO[31:0]
ZERO[31:0]
MUX_OUT[31:0]
Mux4_1
REG_FILE
ADR1[4:0]
ADR2[4:0]
CLK
WA[4:0]
WD[31:0]
RS1[31:0]
RS2[31:0]
RegisterFile
aluA_i
I0[31:0]
I1[31:0]
O[31:0]
S
RTL_MUX
ALUMUX
SEL[1:0]
THREE[31:0]
ZERO[31:0]
MUX_OUT[31:0]
Mux4_1
ALU
A[31:0]
ALU_FUN[3:0]
B[31:0]
ALU_OUT[31:0]
ALU
MEM
IO_IN[31:0]
MEM_ADDR1[31:0]
MEM_ADDR2[31:0]
MEM_CLK
MEM_DIN2[31:0]
MEM_SIGN
MEM_SIZE[1:0]
IO_WR
MEM_DOUT1[31:0]
MEM_DOUT2[31:0]
OTTER_mem_byte
IOBUS_ADDR[31:0]
IOBUS_OUT[31:0]
IOBUS_WR

Code for Hardware 3

```
`timescale 1ns / 1ps


module CUDecoder(
    input BR_EQ, BR_LT, BR_LTU,
    input [2:0] FUNC3,
    input [6:0] FUNC7, CU_OPCODE,
    output logic ALU_SRCA,
    output logic [1:0] ALU_SRCB, PC_SOURCE, RF_WR_SEL,
    output logic [3:0] ALU_FUN
    );
    //create enum datatype for opcode so it can be referenced with the name,
not bit
    typedef enum logic [6:0]    {
        LUI     = 7'b0110111,
        AUIPC   = 7'b0010111,
        JAL     = 7'b1101111,
        JALR    = 7'b1100111,
        BRANCH  = 7'b1100011,
        LOAD    = 7'b0000011,
        STORE   = 7'b0100011,
        OP_IMM  = 7'b0010011,
        OP      = 7'b0110011
    } opcode_t;

    opcode_t  OPCODE;
    assign OPCODE = opcode_t' (CU_OPCODE);

    always_comb
    begin
        ALU_FUN = 0; ALU_SRCA = 0; ALU_SRCB = 0; PC_SOURCE = 0; RF_WR_SEL =
0;
        case(CU_OPCODE)
            LUI:
            begin
                ALU_FUN = 9; ALU_SRCA = 1; ALU_SRCB = 0; PC_SOURCE = 0;
RF_WR_SEL = 3;
            end

            AUIPC:
            begin
                ALU_FUN = 0; ALU_SRCA = 1; ALU_SRCB = 3; PC_SOURCE = 0;
RF_WR_SEL = 3;
            end

            JAL:
            begin
                ALU_FUN = 0; ALU_SRCA = 0; ALU_SRCB = 0; PC_SOURCE = 3;
RF_WR_SEL = 0;
            end

            JALR:
            begin
                ALU_FUN = 0; ALU_SRCA = 0; ALU_SRCB = 0; PC_SOURCE = 1;
RF_WR_SEL = 0;
```

```
            end

            BRANCH:
            begin
                if((FUNC3 == 3'b000) && (BR_EQ == 1))
                    PC_SOURCE = 2;
                else if(FUNC3 == 3'b001 && BR_EQ == 0)
                    PC_SOURCE = 2;
                else if(FUNC3 == 3'b100 && BR_LT == 1)
                    PC_SOURCE = 2;
                    //BGE
                else if((FUNC3 == 3'b101) && ((BR_LT == 0) || (BR_EQ == 1)))
                    PC_SOURCE = 2;
                else if((FUNC3 == 3'b110) && (BR_LTU == 1))
                    PC_SOURCE = 2;
                    //BGEU
                else if((FUNC3 == 3'b111) && ((BR_LTU == 0) || (BR_EQ == 1)))
                    PC_SOURCE = 2;
                else
                    PC_SOURCE = 0;
            end

            LOAD:
            begin
                ALU_FUN = 0; ALU_SRCA = 0; ALU_SRCB = 1; PC_SOURCE = 0;
RF_WR_SEL = 2;
            end

            STORE:
            begin
                ALU_FUN = 0; ALU_SRCA = 0; ALU_SRCB = 2; PC_SOURCE = 0;
RF_WR_SEL = 0;
            end

            OP_IMM:
            begin
                ALU_SRCA = 0; ALU_SRCB = 1; PC_SOURCE = 0; RF_WR_SEL = 3;
                if (FUNC3 == 3'b101)
                //is the 2nd to most signifigant bit 5 or 1?
                    ALU_FUN = {FUNC7[5], FUNC3};
                else
                    ALU_FUN = {1'b0, FUNC3};
            end

            OP:
            begin
            //again, is it func7[5] or func7[1]
                ALU_FUN = {FUNC7[5], FUNC3}; ALU_SRCA = 0; ALU_SRCB = 0;
PC_SOURCE = 0; RF_WR_SEL = 3;
            end

            default:
            begin
                ALU_FUN = 0; ALU_SRCA = 0; ALU_SRCB = 0; PC_SOURCE = 0;
RF_WR_SEL = 0;
            end
        endcase
```

```
    end
endmodule
```

```
`timescale 1ns / 1ps

module OTTER_MCU(
    input CLK, RST,INTR,
    input [31:0] IOBUS_IN,
    output logic IOBUS_WR,
    output logic [31:0] IOBUS_OUT, IOBUS_ADDR
    );

    //PC
    logic [31:0] pc_in;
    logic PC_WRITE;

    //PC Mux
    logic [31:0] JUMP, BRANCH, JALR, pc_4;
    logic [1:0] PC_SOURCE;

    //REG mux
    logic [1:0] rf_wr_sel;
    logic [31:0] dout2;

    //REG file
    logic [31:0] regWd;
    logic regWrite;

    //MEM
    logic [31:0] ir;
    logic [31:0] addr1, addr2;
    logic memRead1, memRead2, memWrite;

    //AluMUX A
    logic [31:0] rs1;
    logic alu_srcA;

    //AluMux B
    logic [31:0] rs2;
    logic [1:0] alu_srcB;

    //ALU
    logic [3:0] alu_fun;
    logic [31:0] aluA, aluB;

    //CU Decoder
    logic br_eq, br_lt, br_ltu;

    //Immedgen types
    logic [31:0] Jtype, Btype, Utype, Itype, Stype;

    //unconnected
    logic [31:0] csrreg;

    //PC MUX
```

```
Mux4_1 pc_mux(
.ZERO(pc_4),
.ONE(JALR),
.TWO(BRANCH),
.THREE(JUMP),
.SEL(PC_SOURCE),
.MUX_OUT(pc_in));

//PC
PC myPC(
.CLK(CLK),
.RESET(RESET),
.PC_WRITE(PC_WRITE),
.DIN(pc_in),
.DOUT(addr1));

assign pc4 = addr1 + 4;

//MEM
OTTER_mem_byte MEM(
.MEM_ADDR1(addr1),
.MEM_ADDR2(addr2),
.MEM_READ1(memRead1),
.MEM_READ2(memRead2),
.MEM_WRITE2(memWrite),
.MEM_DOUT1(ir),
.MEM_DOUT2(dout2),
.MEM_CLK(CLK),
.MEM_DIN2(rs2),
.MEM_SIZE(ir[13:12]),
.IO_WR(IOBUS_WR),
.IO_IN(IOBUS_IN),
.MEM_SIGN(ir[14]) );

//Reg mux
Mux4_1 REG_MUX(
.ZERO(pc4),
.ONE(csrreg),
.TWO(dout2),
.THREE(addr2),
.SEL(rf_wr_sel),
.MUX_OUT(regWd) );

//Register
RegisterFile REG_FILE(
.ADR1(ir[19:15]),
.ADR2(ir[24:20]),
.WA(ir[11:7]),
.EN(regWrite) ,
.WD(regWd),
.RS1(rs1),
.RS2(rs2),
.CLK(CLK) );

//alu 2:1 mux
always_comb
begin
```

```
        if (alu_srcA)
            aluA= Utype;
        else
            aluA = rs1;
    end

  //ALU MUX
   Mux4_1 ALUMUX(
   .ZERO(rs2),
   .ONE(Itype),
   .TWO(Stype),
   .THREE(addr1),
   .SEL(alu_srcB),
   .MUX_OUT(aluB) );

   //ALU
   ALU ALU(
   .A(aluA),
   .B(aluB),
   .ALU_FUN(alu_fun),
   .ALU_OUT(addr2) );

    //DECODER
   CUDecoder CU_Decoder(
   .BR_EQ(br_eq),
   .BR_LT(br_lt),
   .BR_LTU(br_ltu),
   .CU_OPCODE(ir[6:0]),
   .FUNC3(ir[14:12]),
   .FUNC7(ir[31:25]),
   .ALU_FUN(alu_fun),
   .ALU_SRCA(alu_srcA),
   .ALU_SRCB(alu_srcB),
   .PC_SOURCE(pc_source),
   .RF_WR_SEL(rf_wr_sel) );


endmodule
```