CPE 233-07

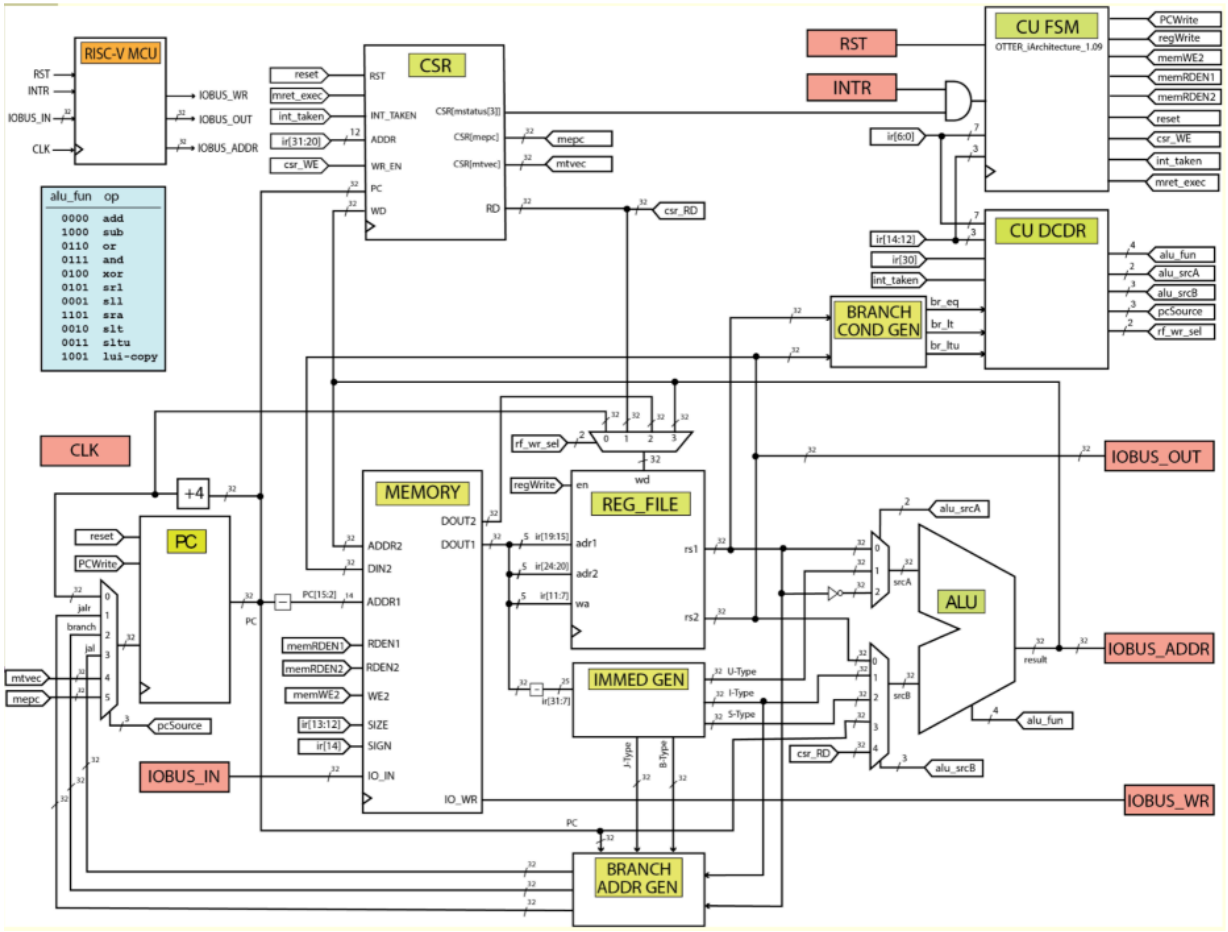
The RISC-V MCU with Interrupts

Kylar Huynh, Dean Vo, Maddie Masiello

Executive Summary:

In this lab, we implemented the complete RISC-V MCU architecture, including the CSR module and interrupts. Past labs, given modules, and newly edited modules were combined to complete the MCU.

Diagrams



Lab 6 RISC-V Otter MCU Architecture Diagram (with Interrupts)

RISC-V otter MCU block diagram schematic with interrupts.

Machine Code

```
.text
main:
init:    li     x15,0x1100C004  # put output address into register
         la     x6,ISR          # load address of ISR into x6
         csrrw  x0,mtvec,x6     # store address as interrupt vector CSR[mtvec]

         mv     x8,x0           # clear x8; use as flag
         mv     x20,x0          # keep track of current output value
         sw     x20,0(x15)      # put LEDs in known state
         li     x7,0x80         # store the MPIE bit position

         li     x6,0x8          # set value in x10
         csrrw  x0,mstatus,x6   # enable interrupts

loop:    nop                    # do nothing (easier to see in simulator)
         beq    x8,x0,loop      # wait for interrupt

         xori   x20,x20,1       # toggle current LED value
         sw     x20,0(x15)      # output LED value

         mv     x8,x0           # clear flag
         csrrs  x0,mstatus,x6   # enable interrupt
         j      loop            # return to loopville

#-------------------------------------------------------------------------
#- The ISR: sets bit x8 to act as flag to task code.
#-------------------------------------------------------------------------
ISR:     li     x8,1            # set flag to non-zero
         csrrc  x0,mstatus,x7   # prepare to disable interrupts

         mret                   # return from interrupt
#-------------------------------------------------------------------------
```

Figure 25: The raw assembly language file for the test program.

Simulation

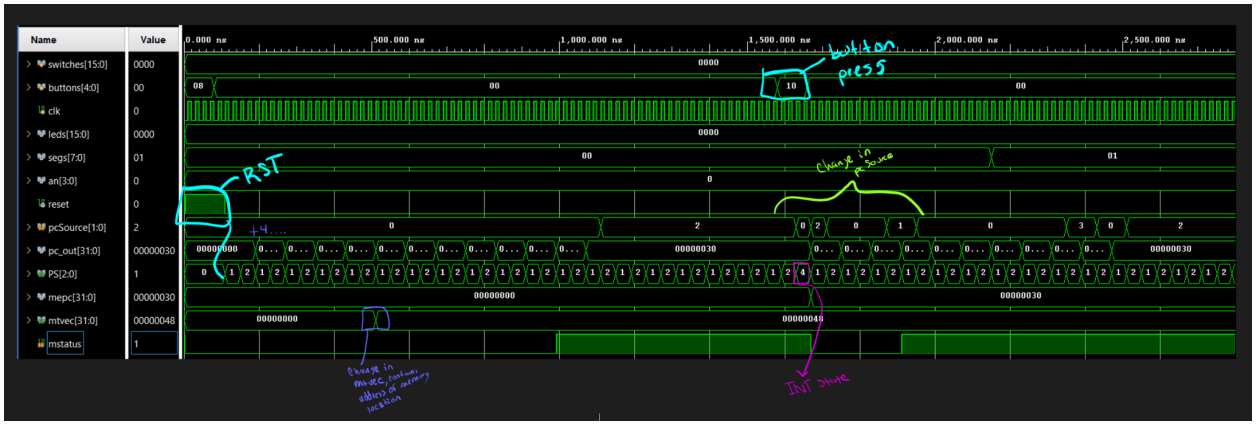


Figure 1.1 Simulation: Broader overview of the simulation

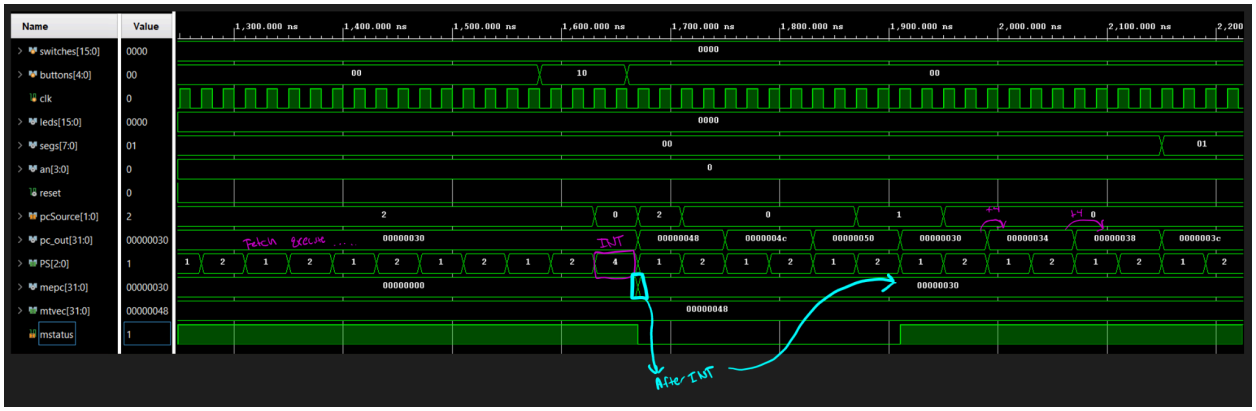


Figure 1.2 Simulation: Closer look at the interrupt and the change in state.

Source Code - Vivado

Top Level:

```
`timescale 1ns / 1ps
// Engineer: Maddie Masiello
// Create Date: 02/14/2023 01:50:38 PM
// Design Name: Lab 5
// Module Name: RISC_V_MCU
// Project Name: Lab 5
// Target Devices: Basys3
/* Description: This is the top level for the RISC-V MCU. It "puts everything together" and
incorperates many modules previously created*/

module RISC_V_MCU(
    input clk,
    input [31:0] IOBUS_IN,
    input RST,
    input intr,
    output IOBUS_WR,
    output [31:0] IOBUS_OUT,
    output [31:0] IOBUS_ADDR
    );

    wire intr_wire;
    assign intr_wire = intr;

//ALU MODULE
    wire [31:0] src_A;  //wire going into ALU from 3:1 mux, output of 2:1 mux
    wire [31:0] src_B;  //wire going into ALU from 5:1 mux, output of 4:1 mux
    wire [1:0] alu_srcA;
    wire [2:0] alu_srcB;    //selects for ALU muxes
    wire [31:0] rs1_wire, rs2_wire;
    wire [31:0] result_wire;
    wire [3:0] alu_fun;


//IMMED GEN wires
   wire [31:0] i_type;
   wire [31:0] j_type;
   wire [31:0] s_type;


   wire [31:0] data_out;
   wire [1:0] rf_wr_sel;

   //PC wires
   wire [2:0] pcSource;
   wire reset;
   wire [31:0] pc_to_mux;



 //reg file wires
    wire [1:0] rf_reg_sel;
    wire [31:0] wd_wire;
    wire [31:0] DOUT_wire;
    wire [4:0] adr1_wire;
    wire [4:0] adr2_wire;
    wire [4:0] wa_wire;
    wire regWrite;

//memory wires
    wire memRDEN1_wire, memRDEN2_wire, memWE2_wire;
    wire [1:0] size_wire; //IR[13:12]
```

```
    wire sign_wire; //IR[14]
    wire [31:0] pc_out;
    wire [31:0] ADDR2_wire;
    wire [31:0] DIN2_wire;
    wire IOBUS_WR_wire;
    wire [31:0] IR;
    wire [31:0] jal, jalr, branch;

//CU wires
    wire br_eq_wire;
    wire br_lt_wire;
    wire br_ltu_wire;
    wire [6:0] opcode;
    wire func7_wire;
    wire [2:0] func3_wire;
    wire [31:0] u_type;

  //Inputs for CSR
  //IR[31:20} --> ADDR
  wire MRET_EXEC;
  wire int_taken_wire;
  wire csr_WE;
  //reset
  //result_wire --> WD

  // Outputs for CSR REG
  wire [31:0] CSR_RD_wire;
  wire [31:0] mepc;
  wire [31:0] mtvec;
  wire mstatus;
  wire csr_REG;

//assigning CU wires
    assign opcode = IR[6:0];
    assign func7_wire = IR[30];
    assign func3_wire = IR[14:12];

//IR assignment
    assign size_wire = IR[13:12];
    assign sign_wire = IR[14];
    assign adr1_wire = IR[19:15];
    assign adr2_wire = IR[24:20];
    assign wa_wire = IR[11:7];

//assigning outputs
   assign IOBUS_ADDR = result_wire;
   assign IOBUS_WR = IOBUS_WR_wire;
   assign IOBUS_OUT = rs2_wire;

  //assigning CSR wires
//  wire [31:0] RD_wire;
//  assign CSR_RD_wire = RD_wire;
  //AND gate with intr and CSR[mstatus[3]]
  wire and_gate_wire;
  assign and_gate_wire = intr_wire & mstatus;


  // Instantiate the CSR_MODULE module
CSR  my_csr (
    .CLK        (clk),
    .RST        (reset),
    .MRET_EXEC  (MRET_EXEC),
    .INT_TAKEN  (int_taken_wire),
    .ADDR       (IR[31:20]),
    .PC         (pc_out),
    .WD         (result_wire),
    .WR_EN      (csr_WE),
```

```
    .RD         (CSR_RD_wire),
    .CSR_MEPC   (mepc),
    .CSR_MTVEC  (mtvec),
    .CSR_MSTATUS_MIE (mstatus)    ); //mstatus[3]

BRANCH_COND_GEN my_branch_con_gen(
    .rs1 (rs1_wire),
    .rs2 (rs2_wire),
    .br_eq(br_eq_wire),
    .br_lt(br_lt_wire),
    .br_ltu(br_ltu_wire)
);

ALU_MODULE ALU_MODULE(
    .op_1       (src_A),
    .op_2       (src_B),
    .alu_fun    (alu_fun),
    .result     (result_wire) );


wire [31:0] not_rs1_wire;   //creating not gate wire
assign not_rs1_wire = ~rs1_wire;

//3:1 MUX going into ALU (SRCA)
mux_4t1_nb  #(.n(32)) alu_3t1_mux  (
     .SEL   (alu_srcA),
     .D0    (rs1_wire),
     .D1    (u_type),
     .D2    (not_rs1_wire), //new mux input
     .D3    (0),
     .D_OUT (src_A) );


//5:1 MUX going into ALU (SRC_b)
       mux_8t1_nb  #(.n(32)) alu_8t1_mux  (
       .SEL   (alu_srcB),
       .D0    (rs2_wire),
       .D1    (i_type),
       .D2    (s_type),
       .D3    (pc_out),
       .D4    (CSR_RD_wire),
       .D5    (0),
       .D6    (0),
       .D7    (0),
       .D_OUT (src_B) );

  //- Register file instantiation template
RegFile my_regfile (
    .wd   (wd_wire),
    .clk  (clk),
    .en   (regWrite),
    .adr1 (adr1_wire),
    .adr2 (adr2_wire),
    .wa   (wa_wire),
    .rs1  (rs1_wire),
    .rs2  (rs2_wire)  );

//reg file 4:1 mux
  mux_4t1_nb  #(.n(32)) reg_file_mux  (
       .SEL   (rf_wr_sel),
       .D0    (pc_to_mux),
       .D1    (CSR_RD_wire),
       .D2    (DOUT_wire),
       .D3    (result_wire),
       .D_OUT (wd_wire) );

//assigning pc_to_mux wire
```

```
  assign pc_to_mux = pc_out + 4;

//IG module assign statments
    assign u_type = {IR[31:12], 12'b000000000000};
    assign i_type = {{21{IR[31]}}, IR[31:25], IR[24:20]};
    assign s_type = {{21{IR[31]}}, IR[30:25], IR[11:7]};
    assign j_type = {{12{IR[31]}}, IR[19:12], IR[20], IR[30:21], 1'b0};
    assign b_type = {{20{IR[31]}}, IR[7], IR[30:25], IR[11:8], 1'b0};

//BAG module assign statements
    assign jal = pc_out + j_type;
    assign branch = pc_out + b_type;
    assign jalr = rs1_wire + i_type;

//program counter module
PC_MODULE PC_MODULE(
    .reset (reset),
    .pcWrite (pcWrite),
    .pcSource (pcSource),
    .clk (clk),
    .jal  (jal),
    .jalr (jalr),
    .branch  (branch),
    .pc_out (pc_out),
    .mtvec (mtvec),
    .mepc (mepc) );

//memory
Memory OTTER_MEMORY (
        .MEM_CLK    (clk),
        .MEM_RDEN1  (memRDEN1_wire),
        .MEM_RDEN2  (memRDEN2_wire),
        .MEM_WE2    (memWE2_wire),
        .MEM_ADDR1  (pc_out[15:2]), //pc_out [15:2]
        .MEM_ADDR2  (result_wire),
        .MEM_DIN2   (rs2_wire),
        .MEM_SIZE   (size_wire),
        .MEM_SIGN   (sign_wire),
        .IO_IN      (IOBUS_IN),
        .IO_WR      (IOBUS_WR_wire),
        .MEM_DOUT1  (IR),
        .MEM_DOUT2  (DOUT_wire)  );

 //CU DCDR
 CU_DCDR my_cu_dcdr(
   .br_eq     (br_eq_wire),
   .br_lt     (br_lt_wire),
   .br_ltu    (br_ltu_wire),
   .opcode    (opcode),    //-  ir[6:0]
   .func7     (func7_wire),    //-  ir[30]
   .func3     (func3_wire),    //-  ir[14:12]
   .alu_fun   (alu_fun),
   .pcSource  (pcSource),
   .alu_srcA  (alu_srcA), //selects for ALU
   .alu_srcB  (alu_srcB),
   .rf_wr_sel (rf_wr_sel),
   .int_taken  (int_taken_wire)
     );


//- instantiation template
CU_FSM my_fsm(
   .intr     (and_gate_wire), //and gate
   .clk      (clk),
   .RST      (RST),     //input RST
   .opcode   (opcode),   // ir[6:0]
   .pcWrite  (pcWrite),
```

```
    .regWrite (regWrite),
    .memWE2   (memWE2_wire),
    .memRDEN1 (memRDEN1_wire),
    .memRDEN2 (memRDEN2_wire),
    .reset    (reset),
    .int_taken (int_taken_wire),
    .func3    (IR[14:12]),
    .mret_exec (MRET_EXEC),
    .csr_WE   (csr_WE)
     );       //output reset

endmodule
```

FSM:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:  Ratner Surf Designs
// Engineer: James Ratner
//
// Create Date: 01/07/2020 09:12:54 PM
// Design Name:
// Module Name: top_level
// Project Name:
// Target Devices:
// Tool Versions:
// Description: Control Unit Template/Starter File for RISC-V OTTER
//
//     //- instantiation template
//     CU_FSM my_fsm(
//        .intr     (xxxx),
//        .clk      (xxxx),
//        .RST      (xxxx),
//        .opcode   (xxxx),   // ir[6:0]
//        .pcWrite  (xxxx),
//        .regWrite (xxxx),
//        .memWE2   (xxxx),
//        .memRDEN1 (xxxx),
//        .memRDEN2 (xxxx),
//        .reset    (xxxx)   );
//
// Dependencies:
//
// Revision:
// Revision 1.00 - File Created - 02-01-2020 (from other people's files)
//          1.01 - (02-08-2020) switched states to enum type
//          1.02 - (02-25-2020) made PS assignment blocking
//                              made rst output asynchronous
//          1.03 - (04-24-2020) added "init" state to FSM
//                              changed rst to reset
//          1.04 - (04-29-2020) removed typos to allow synthesis
//          1.05 - (10-14-2020) fixed instantiation comment (thanks AF)
//          1.06 - (12-10-2020) cleared most outputs, added commentes
//
//////////////////////////////////////////////////////////////////////////////////

module CU_FSM(
    input intr,
    input clk,
    input RST,
    input [6:0] opcode,     // ir[6:0]
    input [2:0] func3,    //-  ir[14:12]
    output logic pcWrite,
    output logic regWrite,
    output logic memWE2,
    output logic memRDEN1,
    output logic memRDEN2,
    output logic reset,
    output logic csr_WE,
    output logic int_taken,
    output logic mret_exec  );

    typedef  enum logic [2:0] {
       st_INIT,
          st_FET,
       st_EX,
       st_WB,
       st_INT
    }  state_type;
```

```
state_type  NS,PS;

//- datatypes for RISC-V opcode types
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_RG3 = 7'b0110011,
    SYS    = 7'b1110011
} opcode_t;
   opcode_t OPCODE;    //- symbolic names for instruction opcodes

   assign OPCODE = opcode_t'(opcode); //- Cast input as enum


   //- state registers (PS)
   always @ (posedge clk)
    if (RST == 1)             //if reset is 2, go back to init state
        PS <= st_INIT;
    else
        PS <= NS;

always_comb
begin
    //- schedule all outputs to avoid latch
    pcWrite = 1'b0;    regWrite = 1'b0;    reset = 1'b0;
          memWE2 = 1'b0;     memRDEN1 = 1'b0;    memRDEN2 = 1'b0;
          csr_WE = 1'b0;     int_taken = 1'b0;   mret_exec = 1'b0;

    case (PS)

        st_INIT: //waiting state
        begin
            reset = 1'b1;
            NS = st_FET;
        end

        st_FET: //waiting state
        begin
            memRDEN1 = 1'b1;     // initialize fetch cycle
            NS = st_EX;
        end

        st_EX: //decode + execute
        begin
            pcWrite = 1'b1;
                        case (OPCODE)
                            LOAD:
                    begin
                       NS = st_WB;
                       memRDEN2 = 1'b1;
                       regWrite = 1'b0;
                       pcWrite = 1'b0;
                    end

                AUIPC:
                    begin
                       NS = st_FET;
                       memRDEN2 = 1'b0;
                       regWrite = 1'b1;
                       if (intr == 1)
                          NS = st_INT;
```

```
end

            STORE:
begin
   regWrite = 1'b0;
                  memWE2 = 1'b1;
   NS = st_FET;
   if (intr == 1)
      NS = st_INT;
end

            BRANCH:
begin
   NS = st_FET;
   if (intr == 1)
      NS = st_INT;
end

            LUI:
               begin
   regWrite = 1'b1;
                  NS = st_FET;
                  if (intr == 1)
      NS = st_INT;
               end

            OP_IMM:  // addi
               begin
                  regWrite = 1'b1;
                  NS = st_FET;
                  if (intr == 1)
      NS = st_INT;
               end

            OP_RG3:  // OP RG3
               begin
                  regWrite = 1'b1;
                  NS = st_FET;
                  if (intr == 1)
      NS = st_INT;
               end

JAL:
               begin
                  regWrite = 1'b1;
                  NS = st_FET;
                  if (intr == 1)
      NS = st_INT;
               end

            JALR:
               begin
                  regWrite = 1'b1;
                  NS = st_FET;
                  if (intr == 1)
      NS = st_INT;
               end

         SYS:
            begin
   case(func3)
            3'b001:   // CSRRW
            begin
         csr_WE = 1'b1;
         regWrite = 1'b1;
            end
```

```
                                    3'b011:   // CSRRC
                                    begin
                                csr_WE = 1'b1;
                                regWrite = 1'b1;
                                    end

                                    3'b010:   // CSRRS
                                    begin
                                csr_WE = 1'b1;
                                regWrite = 1'b1;
                                    end

                                    3'b000:   // MRET
                                    begin
                                mret_exec = 1'b1;
                                          end
                                endcase
                                NS = st_FET;
                                          if (intr == 1)
                            NS = st_INT;
                             end

                    default:
                                          begin
                                             NS = st_FET;
                                             if (intr == 1)
                            NS = st_INT;
                                          end

                endcase
            end

            st_WB:
            begin
               memRDEN2 = 1'b0;
               regWrite = 1'b1;
               pcWrite = 1'b1;
               NS = st_FET;
               if (intr == 1)
                  NS = st_INT;
            end

            st_INT: // interrupt state
            begin
                int_taken = 1'b1;     // pulse int taken
                pcWrite = 1'b1;       // write mtvec into pc
                NS = st_FET;
            end

            default: NS = st_FET;

        endcase //- case statement for FSM states
    end

endmodule
```

CU_DCDR:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Ratner Surf Designs
// Engineer: James Ratner
//
// Create Date: 01/29/2019 04:56:13 PM
// Design Name:
// Module Name: CU_Decoder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// CU_DCDR my_cu_dcdr(
//   .br_eq     (),
//   .br_lt     (),
//   .br_ltu    (),
//   .opcode    (),    //-  ir[6:0]
//   .func7     (),    //-  ir[30]
//   .func3     (),    //-  ir[14:12]
//   .alu_fun   (),
//   .pcSource  (),
//   .alu_srcA  (),
//   .alu_srcB  (),
//   .rf_wr_sel ()   );
//
//
// Revision:
// Revision 1.00 - File Created (02-01-2020) - from Paul, Joseph, & Celina
//          1.01 - (02-08-2020) - removed unneeded else's; fixed assignments
//          1.02 - (02-25-2020) - made all assignments blocking
//          1.03 - (05-12-2020) - reduced func7 to one bit
//          1.04 - (05-31-2020) - removed misleading code
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module CU_DCDR(
    input br_eq,
       input br_lt,
       input br_ltu,
    input [6:0] opcode,   //-  ir[6:0]
       input func7,          //-  ir[30]
    input [2:0] func3,    //-  ir[14:12]
    input int_taken,
    output logic [3:0] alu_fun,
    output logic [2:0] pcSource,
    output logic [1:0] alu_srcA,
    output logic [2:0] alu_srcB,
       output logic [1:0] rf_wr_sel   );

    //- datatypes for RISC-V opcode types
    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_RG3 = 7'b0110011,
        SYS    = 7'b1110011
    } opcode_t;
    opcode_t OPCODE; //- define variable of new opcode type

    assign OPCODE = opcode_t'(opcode); //- Cast input enum

    //- datatype for func3Symbols tied to values
    typedef enum logic [2:0] {
        //BRANCH labels
        BEQ = 3'b000,
        BNE = 3'b001,
        BLT = 3'b100,
        BGE = 3'b101,
        BLTU = 3'b110,
        BGEU = 3'b111
    } func3_t;
    func3_t FUNC3; //- define variable of new opcode type

    assign FUNC3 = func3_t'(func3); //- Cast input enum

    always_comb
    begin
        //- schedule all values to avoid latch
              pcSource = 3'b000;  alu_srcB = 3'b000;    rf_wr_sel = 2'b00;
              alu_srcA = 2'b00;   alu_fun  = 4'b0000;

              if (int_taken  == 1'b1)
                  pcSource = 3'b100;
             else

              case(OPCODE)
                     LUI:
                     begin
                            alu_fun = 4'b1001;
                            alu_srcA = 2'b01;
                            rf_wr_sel = 2'b11;
                     end

                   AUIPC: // auipc
            begin
                alu_fun = 4'b0000;  //add
                alu_srcA = 2'b01;    //Set alu_srcA to 1 to use the immediate value as the
operand
                alu_srcB = 3'b011;   //3 because it takes value from PC
                rf_wr_sel = 2'b11;  //result of ALU
            end

            STORE: // store
            begin
                rf_wr_sel = 2'b00;
                alu_srcB = 3'b010;
            end

                     JAL:
                     begin
                            pcSource = 3'b011;
                            alu_fun = 4'b0000;
                            alu_srcA = 2'b00;
                            alu_srcB = 3'b000;
                            rf_wr_sel = 2'b00;
                     end

                     JALR:
                     begin
                            pcSource = 3'b001;
                            alu_fun = 4'b0000;
```

```
       alu_srcA = 2'b00;
       alu_srcB = 3'b000;
       rf_wr_sel = 2'b00;
end

LOAD:
begin
       alu_fun = 4'b0000;
       alu_srcA = 2'b00;
       alu_srcB = 3'b001;
       rf_wr_sel = 2'b10;     // load mem dout2 into reg
end

BRANCH:
begin
 case(FUNC3)
     3'b000:   //beq
     begin
     if (br_eq == 1'b1)
       pcSource = 3'b010;
     else
       pcSource = 3'b000;
     end

     3'b001:   //bne
     begin
     if (br_eq == 1'b0)
       pcSource = 3'b010;
     else
       pcSource = 3'b000;
     end

     3'b100:   //blt
     begin
     if (br_lt == 1'b1)
       pcSource = 3'b010;
     else
       pcSource = 3'b000;
     end

     3'b101:   //bge
     begin
     if (br_lt == 1'b0)
       pcSource = 3'b010;
     else
       pcSource = 3'b000;
     end

     3'b110:   //bltu
     begin
     if (br_ltu == 1'b1)
       pcSource = 3'b010;
     else
       pcSource = 3'b000;
     end

     3'b111:   //bgeu
     begin
     if (br_ltu == 1'b0)
       pcSource = 3'b010;
     else
       pcSource = 3'b000;
     end

     default:
     begin
       pcSource = 3'b000;
```

```
     end
  endcase
end


OP_RG3:    // R type
begin
    rf_wr_sel = 2'b11;
    alu_srcA = 2'b00;
       alu_srcB = 3'b000;
       case(func3)
              3'b000: // instr: ADD and SUB
              begin
                 case(func7)
                     1'b0:   // ADD
                     begin
                         alu_fun = 4'b0000;
                     end

                     1'b1:   // SUB
                     begin
                         alu_fun = 4'b1000;
                     end

                        default:
                        begin
                           alu_fun = 4'b0000;
                        end
                     endcase
              end

              3'b001: // SLL
              begin
                     alu_fun = 4'b0001;
              end

              3'b010: // SLT
              begin
                     alu_fun = 4'b0010;
              end

              3'b011: // SLTU
              begin
                     alu_fun = 4'b0011;
              end

              3'b100: // XOR
              begin
                     alu_fun = 4'b0100;
          end

              3'b101: // SRL and SRA
              begin
              case(func7)
                     1'b0:   // SRL
                     begin
                         alu_fun = 4'b0101;
                     end

                     1'b1:   // SRA
                     begin
                         alu_fun = 4'b1101;
                     end

                        default:
                        begin
                           alu_fun = 4'b0000;
```

```
                        end
                    endcase
                end

                3'b110: // OR
                begin
                    alu_fun = 4'b0110;
                end

                3'b111: // AND
                begin
                    alu_fun = 4'b0111;
                end

                default:
                begin
                    pcSource = 3'b000;
                    alu_fun = 4'b0000;
                    alu_srcA = 2'b00;
                    alu_srcB = 3'b000;
                    rf_wr_sel = 2'b11;
                end
        endcase
end

OP_IMM:
begin
    if (int_taken) begin   // if in interrupt pc is set to mtvec
         pcSource = 3'b100;
    end
    rf_wr_sel = 2'b11;
    alu_srcA = 2'b00;
       alu_srcB = 3'b001;
       case(func3)
                3'b000: // instr: ADDI
                begin
                    alu_fun = 4'b0000;
                end

                3'b010: // SLTI
                begin
                    alu_fun = 4'b0010;
                end

                3'b011: // SLTIU
                begin
                    alu_fun = 4'b0011;
                end

                3'b110: // ORI
                begin
                    alu_fun = 4'b0110;
                end

                3'b100: // XORI
                begin
                    alu_fun = 4'b0100;
                end

                3'b111: // ANDI
                begin
                    alu_fun = 4'b0111;
                end

                3'b001: // SLLI
                begin
                    alu_fun = 4'b0001;
```

```
                end

                3'b101: // SRLI and SRAI
                begin
                case(func7)
                       1'b0:   // SRLI
                       begin
                           alu_fun = 4'b0101;
                       end

                       1'b1:   // SRAI
                       begin
                           alu_fun = 4'b1101;
                       end

                          default:
                          begin
                             alu_fun = 4'b0000;
                          end
                       endcase
                end

                default:
                begin
                       pcSource = 3'b000;
                       alu_fun = 4'b0000;
                       alu_srcA = 2'b00;
                       alu_srcB = 3'b000;
                       rf_wr_sel = 2'b11;
                end
        endcase
  end

  SYS:
  begin
   case(FUNC3)
       3'b001:   // CSRRW
       begin
pcSource = 3'b000;
             rf_wr_sel = 2'b01;
       end

       3'b011:   // CSRRC
       begin
       pcSource = 3'b000;
             rf_wr_sel = 2'b01;
       end

       3'b010:   // CSRRS
       begin
       pcSource = 3'b000;
             rf_wr_sel = 2'b01;
       end

       3'b000:   // MRET
       begin
       pcSource = 3'b101;
             end
    endcase
  end



  default:
  begin
          pcSource = 2'b00;
```

```
                            alu_srcB = 3'b000;
                            rf_wr_sel = 2'b00;
                            alu_srcA = 2'b00;
                            alu_fun = 4'b0000;
                        end
                        endcase
    end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Ratner Surf Designs
// Engineer:  James Ratner
//
// Create Date: 01/07/2020 12:59:51 PM
// Design Name:
// Module Name: Ex6_6_testbench
// Project Name:
// Target Devices:
// Tool Versions:
// Description: Testbench file for Exp 6
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module otter_tb();

   reg [15:0] switches;
   reg [4:0] buttons;
   reg clk;
   wire [15:0] leds;
   wire [7:0] segs;
   wire [3:0] an;


OTTER_Wrapper my_wrapper(
   .clk    (clk),
   .buttons  (buttons),
   .switches  (switches),
   .leds  (leds),
   .segs  (segs),
   .an  (an)
     );


  //- Generate periodic clock signal
   initial
      begin
         clk = 0;   //- init signal
         forever  #10 clk = ~clk;
      end

   initial
   begin
      buttons = 5'b01000;    //rst
      switches = 16'h0000;
```

```
      #80//wait a while

      buttons = 5'b00000;

      #1500;

      buttons = 5'b10000;    //intr
      #80//wait a while

      buttons = 5'b00000;   //intr off

    end

endmodule
```