ELEC 374 CPU Design – Phase 3

David-Alexandre Edwards (20099471),

Elan Bibas (20099396),

Hannah Berthiaume (20114482)

Verilog Code

DataPath

```
//added write signal to DataPath
module DataPath(
    input PCout, Zlowout, MDRout, MARin, Zin, PCin, MDRin, IRin, Yin, Read, Write
,
     input [4:0] aluControl,
     input clock, clear,
    input Gra, Grb, Grc, Rin, Rout, BAout, Cout, ConIn, HIout, LOout, RoutPIn, in
cPC,
     inout [31:0] ramOut,
     output conffOut,
     input RHiIn, RLOIn, Zhighout
);


//registers run behavior every positive clock edge, bus will update BusMuxOUt whe
n a change in register output is detected
//define signals that will connect regiters to bus
wire [31:0] BusMuxOut;
wire [31:0] BusMuxInR0;
wire [31:0] BusMuxInR1;
wire [31:0] BusMuxInR2;
wire [31:0] BusMuxInR3;
wire [31:0] BusMuxInR4;
wire [31:0] BusMuxInR5;
wire [31:0] BusMuxInR6;
wire [31:0] BusMuxInR7;
wire [31:0] BusMuxInR8;
wire [31:0] BusMuxInR9;
wire [31:0] BusMuxInR10;
wire [31:0] BusMuxInR11;
wire [31:0] BusMuxInR12;
wire [31:0] BusMuxInR13;
wire [31:0] BusMuxInR14;
wire [31:0] BusMuxInR15;
wire [31:0] BusMuxInZHi;
wire [31:0] BusMuxInZLo;
wire [31:0] BusMuxInPC;
wire [31:0] BusMuxInMDR;
wire [31:0] BusMuxInRinP;
wire [31:0] BusMuxInCSign;
wire [31:0] BusMuxInRY;
```

```
wire [31:0] ALULoOut;
wire [31:0] ALUHiOut;
//phase 2
wire [8:0] Address;
wire [31:0] BusMuxInIR;
wire r0out;
wire r1out;
wire R2out;
wire r3out;
wire R4out;
wire r5out;
wire r6out;
wire r7out;
wire r8out;
wire r9out;
wire r10out;
wire r11out;
wire r12out;
wire r13out;
wire r14out;
wire r15out;
wire [31:0] BusMuxInRoutP;
wire [31:0] BusMuxInRHi;
// phase 3
wire branch;
wire [31:0] BusMuxInLo;

//instantiate all register
 register R1(clock, clear, R1in, BusMuxOut, BusMuxInR1);
 register R2(clock, clear, R2in, BusMuxOut, BusMuxInR2);
 register R3(clock, clear, R3in, BusMuxOut, BusMuxInR3);
 register R4(clock, clear, R4in, BusMuxOut, BusMuxInR4);
 register R5(clock, clear, R5in, BusMuxOut, BusMuxInR5);
 register R6(clock, clear, R6in, BusMuxOut, BusMuxInR6);
 register R7(clock, clear, R7in, BusMuxOut, BusMuxInR7);
 register R8(clock, clear, R8in, BusMuxOut, BusMuxInR8);
 register R9(clock, clear, R9in, BusMuxOut, BusMuxInR9);
 register R10(clock, clear, R10in, BusMuxOut, BusMuxInR10);
 register R11(clock, clear, R11in, BusMuxOut, BusMuxInR11);
 register R12(clock, clear, R12in, BusMuxOut, BusMuxInR12);
 register R13(clock, clear, R13in, BusMuxOut, BusMuxInR13);
 register R14(clock, clear, R14in, BusMuxOut, BusMuxInR14);
 register R15(clock, clear, R15in, BusMuxOut, BusMuxInR15);
 register RHi(clock, clear, RHiIn, BusMuxOut, BusMuxInRHi);
 register RLO(clock, clear, RLOIn, BusMuxOut, BusMuxInLo);
```

```
 register RZHi(clock, clear, Zin, ALUHiOut, BusMuxInZHi);
 register RZLO(clock, clear, Zin, ALULoOut, BusMuxInZLo);
 //register PC(clock, clear, PCIn, BusMuxOut, BusMuxInPC);
 register RInP(clock, clear, RInPIn, BusMuxOut, BusMuxInRInP);
 //register RCSign(clock, clear, RCSignIn, BusMuxOut, BusMuxInCSign);
 register RoutP(clock, clear, RoutPIn, BusMuxOut, BusMuxInRoutP);
 register RY(clock, clear, Yin, BusMuxOut, BusMuxInRY);


//phase 2 Select and Encode
 SelectEncode selectencode(BusMuxInIR, Gra, Grb, Grc, Rin, Rout, BAout, BusMuxInC
Sign, R0in, R1in, R2in, R3in, R4in, R5in, R6in, R7in, R8in, R9in, R10in, R11in, R
12in, R13in, R14in, R15in, R0out, R1out, R2out, R3out, R4out, R5out, R6out, R7out
, R8out, R9out, R10out, R11out, R12out, R13out, R14out, R15out);

 //phase 2 Memory subsystem
 register IR(clock, clear, IRin, BusMuxOut, BusMuxInIR);
 registerR0 R0(clock, clear, R0in, BAout, BusMuxOut, BusMuxInR0);
 MAR mar(clock, clear, MARin, BusMuxOut,Address);
 ram1 ram(Address, clock, BusMuxInMDR, Write, ramOut);
 MDR mdr(clock, clear, MDRin, Read, BusMuxOut, ramOut, BusMuxInMDR);
 pc PC(clock, clear, PCin, incPC, branch, BusMuxOut, BusMuxInPC); //incPC to PCin
 to '1


 //phase 2 conff
conff CONFF(BusMuxOut, BusMuxInIR, ConIn, conffOut);

//instantiate bus
bus cpuBUS(
    BusMuxInR0,BusMuxInR1,BusMuxInR2, BusMuxInR3,BusMuxInR4,BusMuxInR5,BusMuxInR6
,BusMuxInR7,BusMuxInR8,BusMuxInR9,BusMuxInR10,BusMuxInR11,BusMuxInR12,BusMuxInR13
,BusMuxInR14,BusMuxInR15,
    BusMuxInRHi,BusMuxInLo,BusMuxInZHi,BusMuxInZLo,BusMuxInPC,BusMuxInMDR,BusMuxI
nRInP,BusMuxInCSign,
    R0out,R1out,R2out,R3out,R4out,R5out,R6out,R7out,R8out,R9out,R10out,R11out,R12
out,R13out,R14out,R15out,
    HIout,LOout,Zhighout,Zlowout,PCout,MDRout,INportout,Cout,BusMuxOut
);


ALU alu(aluControl, BusMuxInRY, BusMuxOut, ALULoOut, ALUHiOut);


endmodule
```

```
//we essentially need to perform an action anytime one of the control signals cha
nges
// testbench changes state every rising-
edge of clock, does the job associated with state for each
//#10 = time delay of 10 before instruction is ran, same concept with #15
// wires are used to create connection between to ports (input and output port),
they do not store data, only drive data
// MDRout = 1, updates BusMuxOut with value of BusMuxInMDR, since R2in  =1, R2 wi
ll hold value (write to register)
// the always @ block is used for cyclic behavior
```

Select & Encode

```
module SelectEncode(
    input [31:0] IR,
    input Gra, Grb, Grc, Rin, Rout, BAout,
    output [31:0] C_sign,
    output R0in, R1in, R2in, R3in, R4in, R5in, R6in, R7in, R8in, R9in, R10in, R11
in, R12in, R13in, R14in, R15in,
    output R0out, R1out, R2out, R3out, R4out, R5out, R6out, R7out, R8out, R9out,
R10out, R11out, R12out, R13out, R14out, R15out
);

//have two outputs instead
reg [3:0] OpCode, Ra, Rb, Rc;
reg [3:0] DecoderInput;
reg [15:0] DecoderOutput;
reg [15:0] Out;
reg [31:0] C_sign_extended;
reg [15:0] In;
reg temp;
integer i;



always @ (*) begin
    // Gra or Grb or Grc
     OpCode = IR[31:27];
     Ra = IR[26:23];
     Rb = IR[22:19];
     Rc = IR[18:15];

    if(Gra == 1) begin //Add
        DecoderInput = Ra;
```

```
 end
else if(Grb == 1) begin //Sub
    DecoderInput = Rb;
 end
else if(Grc == 1) begin //Shift Right
    DecoderInput = Rc;
 end

if(DecoderInput == 4'b0000) begin
    DecoderOutput = 16'b0000000000000001;
end
else if(DecoderInput == 4'b0001) begin
    DecoderOutput = 16'b0000000000000010;
end
else if(DecoderInput == 4'b0010) begin
    DecoderOutput = 16'b0000000000000100;
end
else if(DecoderInput == 4'b0011) begin
    DecoderOutput = 16'b0000000000001000;
end
else if(DecoderInput == 4'b0100) begin
    DecoderOutput = 16'b0000000000010000;
end
else if(DecoderInput == 4'b0101) begin
    DecoderOutput = 16'b0000000000100000;
end
else if(DecoderInput == 4'b0110) begin
    DecoderOutput = 16'b0000000001000000;
end
else if(DecoderInput == 4'b0111) begin
    DecoderOutput = 16'b0000000010000000;
end
else if(DecoderInput == 4'b1000) begin
    DecoderOutput = 16'b0000000100000000;
end
else if(DecoderInput == 4'b1001) begin
    DecoderOutput = 16'b0000001000000000;
end
else if(DecoderInput == 4'b1010) begin
    DecoderOutput = 16'b0000010000000000;
end
else if(DecoderInput == 4'b1011) begin
    DecoderOutput = 16'b0000100000000000;
end
else if(DecoderInput == 4'b1100) begin
```

```
        DecoderOutput = 16'b0001000000000000;
    end
    else if(DecoderInput == 4'b1101) begin
        DecoderOutput = 16'b0010000000000000;
    end
    else if(DecoderInput == 4'b1110) begin
        DecoderOutput = 16'b0100000000000000;
    end
    else if(DecoderInput == 4'b1111) begin
        DecoderOutput = 16'b1000000000000000;
    end
     else begin
        DecoderOutput = 16'b0000000000000000;
     end
    for (i = 0 ; i < 16 ; i = i + 1)
        In[i] = DecoderOutput[i] & Rin;
     temp = Rout | BAout;
    for (i = 0 ; i < 16 ; i = i + 1)
        Out[i] = DecoderOutput[i] & temp;
    C_sign_extended = {{13{IR[18]}}, {IR[18:0]}};

end
assign {R15in, R14in, R13in, R12in, R11in, R10in, R9in, R8in, R7in, R6in, R5in, R
4in, R3in, R2in, R1in, R0in} = In;
assign {R15out, R14out, R13out, R12out, R11out, R10out, R9out, R8out, R7out, R6ou
t, R5out, R4out, R3out, R2out, R1out, R0out} = Out;
assign C_sign = C_sign_extended;

endmodule
```

Conff

```
module conff(
    input [31:0] BusMuxIn,
    input [31:0] IR,  // C2 = IR[22:19]
    input ConIn, // phase 3
    output branch
);

/*
make input BusMuxInIR 32 bits and isolate the 4 required bits
output name should be controlOut
you should have an always block that runs anytime inputs change
assign statement will need to be outside of always block (may need a temp reg var
iable and assign output to that outside always block)
```

```
*/

reg temp;
reg temp2;
reg [3:0] BusMuxInIR;
integer i;



always @ (ConIn) begin

    BusMuxInIR = IR[22:19];

    if (BusMuxInIR[0] == 1'b1) begin

         temp = BusMuxInIR[0] & ~(|BusMuxIn);     // IR[0] AND Bus
         // decoder[0] & (|bus)

         if (temp == 1'b1) begin             // if equals 0
              temp2 = 1'b1;
         end
    end
    else if (BusMuxInIR[1] == 1'b1) begin

         temp = BusMuxInIR[1] & (|BusMuxIn);            // IR[1] AND (NOT Bus)

         if (temp != 1'b0) begin                // if not equals 0

              temp2 = temp;
         end
    end
    else if (BusMuxInIR[2] == 1'b1) begin
         temp = BusMuxInIR[2] & ~BusMuxIn[31];          // IR[2] AND (NOT Bus[31]
)

                                                                    // if gre
ater than equal to 0
         temp2 = temp;

    end
    else if (BusMuxInIR[3] == 1'b1) begin
         temp = BusMuxInIR[3] & BusMuxIn[31];           // IR[3] AND Bus[31]
                                                                 // if less th
an 0
```

```
         temp2 = temp;

    end

    if (ConIn == 1'b0) begin
        temp2 = 1'b0;
    end

end

assign branch = temp2; // (((temp[0] | temp[1]) | temp[2]) | temp[3]);


endmodule
```

MAR

```
module MAR(input clock, input clear, input MARin, input [31:0] BusMuxOut, output
[8:0] Address);

reg [31:0] q;
    // Behavioral section for writing to the register
    always @ ( posedge clock)
        begin
            if(clear) begin
                q <= 32'b0;
            end
            else if(MARin) begin
                q <= BusMuxOut;
            end
        end

    assign Address = q[8:0];

endmodule
```

MDR

```
module MDR(
    input clk, clear, MDRin, read,
    input [31:0] BusMuxOut,
    input [31:0] Mdatain,
    output [31:0] BusMuxInMDR
);
```

```
// Behavioral section for writing to the register
reg [31:0] Din;
reg [31:0] In;
always @ (posedge clk)
    begin
        if (read) begin
            Din <= Mdatain;
        end
        else begin
            Din<= BusMuxOut;
        end

        if(clear) begin
            In <= 32'b0;
        end
        else if(MDRin) begin
            In <= Din;
        end
    end

    assign BusMuxInMDR = In;

endmodule
```

RAM

```
module RAM
#(parameter DATA_WIDTH=32, parameter ADDR_WIDTH=9)
(
    input [(DATA_WIDTH-1):0] data,
    input [(ADDR_WIDTH-1):0] addr,
    input read,write, clk,
    output [(DATA_WIDTH-1):0] q
);

    // Declare the RAM variable
    reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];

    // Variable to hold the registered read address
    reg [ADDR_WIDTH-1:0] addr_reg;

    // Specify the initial contents.  You can also use the $readmemb
    // system task to initialize the RAM variable from a text file.
    // See the $readmemb template page for details.
```

```
    initial
    begin : INIT
        integer i;
        for(i = 0; i < 2**ADDR_WIDTH; i = i + 1)
            ram[i] = {DATA_WIDTH{1'b1}};
    end

    always @ (posedge clk)
    begin
        // Write
        if (write)
            ram[addr] <= data;

        addr_reg <= addr;
    end

    // Continuous assignment implies read returns NEW data.
    // This is the natural behavior of the TriMatrix memory
    // blocks in Single Port mode.
    if (read)
        assign q = ram[addr_reg];
    end if

endmodule
```

PC

```
module pc #(parameter VAL = 0)(
    input clock,
    input clear,
    input enable,
    input incPC,
    input branch,
    input [31:0] BusMuxOut,
    output [31:0] BusMuxIn
);

reg [31:0] q;
initial q = VAL;

    // Behavioral section for writing to the register
    always @ (posedge clock)
        begin
```

```
            if(clear) begin
                q <= 32'b0;
            end
            else if(enable) begin
                q <= BusMuxOut;
            end
            else if(incPC) begin
                q <= q + 1;
            end

        end

    assign BusMuxIn = q;

endmodule
```

Bus

```
module bus(
input [31:0] BusMuxInR0, input [31:0] BusMuxInR1, input [31:0] BusMuxInR2,
input [31:0] BusMuxInR3,input [31:0] BusMuxInR4,input [31:0] BusMuxInR5,input [31
:0] BusMuxInR6,
input [31:0] BusMuxInR7,input [31:0] BusMuxInR8,input [31:0] BusMuxInR9,input [31
:0] BusMuxInR10,
input [31:0] BusMuxInR11,input [31:0] BusMuxInR12,input [31:0] BusMuxInR13,input
[31:0] BusMuxInR14,
input [31:0] BusMuxInR15,input [31:0] BusMuxInHi,input [31:0] BusMuxInLo,input [3
1:0] BusMuxInZHi,
input [31:0] BusMuxInZLo,input [31:0] BusMuxInPC, input [31:0] BusMuxInMDR,input
[31:0] BusMuxInRInP,
input [31:0] BusMuxInRCSign, input R0out, input R1out,input R2out,input R3out,inp
ut R4out,input R5out,
input R6out,input R7out,input R8out,input R9out,input R10out,input R11out,input R
12out,
input R13out,input R14out,input R15out,input HIout,input LOout,input Zhighout,inp
ut Zlowout,input PCout,
input MDRout,input InPortout,input Cout, output [31:0] BusMuxOut
);

reg [31:0] out;

always @ (*) begin
```

```
// R0out or R1out or R2out or R4out or R5out or R6out or R7out or R8out or R9out
or R10out or R11out or R12out or R13out or R14out or R15out or MDRout or PCout or
 HIout or LOout or Zhighout or Zlowout or InPortout or Cout
    if(R0out) begin
        out = BusMuxInR0;
    end
    else if(Cout) begin
        out =  BusMuxInRCSign;
    end
    else if(R1out) begin
        out = BusMuxInR1;
    end
    else if(R2out) begin
        out = BusMuxInR2;
    end
    else if(R3out) begin
        out = BusMuxInR3;
    end
    else if(R4out) begin
        out = BusMuxInR4;
    end
    else if(R5out) begin
        out = BusMuxInR5;
    end
    else if(R6out) begin
        out = BusMuxInR6;
    end
    else if(R7out) begin
        out = BusMuxInR7;
    end
    else if(R8out) begin
        out = BusMuxInR8;
    end
    else if(R9out) begin
        out = BusMuxInR9;
    end
    else if(R10out) begin
        out = BusMuxInR10;
    end
    else if(R11out) begin
        out = BusMuxInR11;
    end
    else if(R12out) begin
        out = BusMuxInR12;
    end
```

```
    else if(R13out)begin
        out = BusMuxInR13;
    end
    else if(R14out)begin
        out = BusMuxInR14;
    end
    else if(R15out)begin
        out = BusMuxInR15;
    end
    else if(HIout)begin
        out = BusMuxInHi;
    end
    else if(LOout)begin
        out = BusMuxInLo;
    end
    else if(Zhighout)begin
        out = BusMuxInZHi;
    end
    else if(Zlowout)begin
        out = BusMuxInZLo;
    end
    else if(PCout)begin
        out = BusMuxInPC;
    end
    else if(MDRout)begin
        out = BusMuxInMDR;
    end
    else if(InPortout)begin
        out = BusMuxInRInP;
    end
    else begin
        out = 32'bx;
    end
end
assign BusMuxOut = out;
endmodule
```

ALU

```
module ALU(
    input [4:0] aluControl,
    input [31:0] BusMuxInY,
     input [31:0] BusMuxOut,
    output [31:0] Zlowout,
```

```
    output [31:0] Zhighout
);

//have two outputs instead
//reg [4:0] aluControl;
reg [31:0] COut;
reg [31:0] temp;
reg [31:0] temp1;
reg [31:0] temp2;
wire [63:0] ZOut;
integer i;

boothmult Mult(ZOut, BusMuxInY, BusMuxOut);

always @ (*) begin

    // aluControl = Operator[31:27];

    temp1 = BusMuxOut;
    temp2 = BusMuxInY;
   if(aluControl == 5'b00011 || aluControl == 5'b00000 || aluControl == 5'b00001
 || aluControl == 5'b00010 || aluControl == 5'b10010) begin //Add
       COut = BusMuxInY + BusMuxOut;
    end
    else if (aluControl == 5'b01011) begin //addi
         COut = BusMuxInY + BusMuxOut;
    end
    else if (aluControl == 5'b01101) begin //ori
           for (i = 0 ; i < 32 ; i = i + 1) begin
           COut[i] = BusMuxInY[i] | BusMuxOut[i];
           end
    end
    else if (aluControl == 5'b01100) begin //andi
           for (i = 0 ; i < 32 ; i = i + 1) begin
           COut = (BusMuxInY & temp1);
           end
    end
   else if(aluControl == 5'b00100) begin //Sub
       COut = BusMuxInY - BusMuxOut;
    end
   else if(aluControl == 5'b00101) begin //Shift Right
       for (i = 0 ; i < 31 ; i = i + 1) begin
           COut[i] = BusMuxInY[i+1];
         end
       COut[31] = 0;
```

```
 end
else if(aluControl == 5'b00110) begin //Shift Left
      for (i = 1 ; i < 32 ; i = i + 1) begin
        COut[i] = BusMuxInY[i-1];
      end
    COut[0] = 0;
 end
else if(aluControl == 5'b00111) begin //Rotate Right
    for (i = 0 ; i < 31 ; i = i + 1) begin
        COut[i] = BusMuxInY[i+1];
      end
    COut[31] = BusMuxInY[0];
 end
else if(aluControl == 5'b01000) begin //Rotate Left
      for (i = 1 ; i < 32 ; i = i + 1) begin
        COut[i] = BusMuxInY[i-1];
      end
    COut[0] = BusMuxInY[31];
 end
else if(aluControl == 5'b01001) begin //AND
//loop with for loop, then use logial and for each bit
    for (i = 0 ; i < 32 ; i = i + 1) begin
        COut = (BusMuxInY & temp1);
    end
end
else if(aluControl == 5'b01010) begin //OR
    for (i = 0 ; i < 32 ; i = i + 1) begin
        COut[i] = BusMuxInY[i] | BusMuxOut[i];
    end
end
else if(aluControl == 5'b01110) begin //Multiply
    COut = ZOut[31:0];
    temp = ZOut[63:32];
 end
else if(aluControl == 5'b01111) begin //Divide
    COut = BusMuxInY / BusMuxOut;
      temp = BusMuxInY % BusMuxOut;
 end
else if(aluControl == 5'b10000) begin // Negate
    for (i = 0 ; i < 32 ; i = i + 1) begin
            COut[i] = ~temp2[i];
      end
      COut[0] = COut[0] + 1'b1;
 end
else if(aluControl == 5'b10001) begin // Not
```

```
        for (i = 0 ; i < 32 ; i = i + 1) begin
                COut[i] = ~temp2[i];
            end
     end
     else if (aluControl == 5'b11010) begin //store i
            COut = BusMuxInY;
     end

     if(aluControl != 5'b01111 && aluControl != 5'b01110)
        temp = {32{COut[31]}};
    // assign the results to z high and z low

end

assign Zhighout = temp; // ZOut[31:0]
assign Zlowout = COut;  // ZOut[63:32]

endmodule

// Z register holds the results of the operation in ALU
```

CPU

```
`timescale 1ns/10ps

module cpu(input clock, reset, stop, run);
    //import clock, reset
    //cpu top level entity

    wire PCout, Zlowout, MDRout, MARin, Zin, PCin, MDRin, IRin, Yin, incPC, Read,

    Write, clear, Gra, Grb, Grc, Rin, Rout, BAout, Cout, ConIn, HIout, LOout, Rou
tPIn;
    wire CONFF;
    wire [4:0] aluControl;
    wire [31:0] instr;
    wire RHiIn, RLOIn, ZHIout;

    DataPath DUT(PCout, Zlowout, MDRout, MARin, Zin, PCin, MDRin, IRin, Yin, Read
, Write, aluControl, clock, clear,
    Gra, Grb, Grc, Rin, Rout, BAout, Cout, ConIn, HIout, LOout, RoutPIn, incPC, i
nstr, CONFF, RHiIn, RLOIn, ZHIout);

    control_unit CU(PCout, Zlowout, MDRout, incPC, MARin, Zin, PCin, MDRin, IRin,
 Yin, Read, Write,
```

```
    clear, Gra, Grb, Grc, Rin, Rout, BAout, Cout, ConIn, HIout, LOout, RoutPIn, a
luControl, clock, reset, CONFF,
    instr, RHiIn, RLOIn, ZHIout, run);


endmodule
```

Cpu_tb

```
`timescale 1ns/10ps

module cpu_tb;

reg clock;
reg reset;
reg stop;
reg run;

initial begin
        reset <= 1;
        stop <= 0;
        #10 reset <= 0;
        clock = 0;
        forever #10 clock = ~clock;
end

cpu cpu1(clock, reset, stop, run);


endmodule
```

Control_unit

```
`timescale 1ns/10ps
//do we need to create new file for both control and datapath
//need to define clock, reset, stop, con_FF
//Stop->how do we halt?, Run-
> 1 (do we need if statement to set state to fetch0), Reset -
>clear =1, initialize everything to 0
//if stop=1, run =0 stop clk
module control_unit(
    output reg PCout, Zlowout, MDRout, IncPC,
    MARin, Zin, PCin, MDRin, IRin, Yin,
    Read, Write, clear,
    Gra, Grb, Grc, Rin, Rout, BAout, Cout, ConIn, HIout, LOout, RoutPIn,
   output reg [4:0] aluControl,
```

```
    input clk, reset, CONFF,
   input [31:0] IR,
   output reg RHiIn, RLOIn, ZHIout, run
);

    parameter Reset_state = 6'b000000, fetch0 = 6'b000001, fetch1 = 6'b000010,
        fetch2 = 6'b000011, addi3 = 6'b000100, addi4 = 6'b000101, addi5 = 6'b0001
10, load3 = 6'b000111, load4 = 6'b001000,
        load5 = 6'b001001, load6 = 6'b001010, load7 = 6'b001011, loadi3 = 6'b0011
00, loadi4 = 6'b001101, loadi5 = 6'b001110,
        store3 = 6'b001111, store4 = 6'b010000, store5 = 6'b010001, store6 = 6'b0
10010, alu3 = 6'b010011, alu4 = 6'b010100, alu5 = 6'b010101,
        alureg3 = 6'b010110, alureg4 = 6'b010111, alureg5 = 6'b011000, muldiv3 =
6'b011001, muldiv4 = 6'b011010,
        muldiv5 = 6'b011011, andi3 = 6'b011100, andi4 = 6'b011101, andi5 = 6'b011
110, ori3 = 6'b011111,
        ori4 = 6'b100000, ori5 = 6'b100001, branch3 = 6'b100010, branch4 = 6'b100
011, branch5 = 6'b100100, branch6 = 6'b100101,
        jump3 = 6'b100110, mfhi3 = 6'b100111, inout3 = 6'b101000, storei3 = 6'b10
1001, storei4 = 6'b101010,
        storei5 = 6'b101011, storei6 = 6'b101100, store7 = 6'b101101, muldiv6 = 6
'b101110, muldiv7 = 6'b101111,
        mfhi4 = 6'b110000, mflo3 = 6'b110001, mflo4 = 6'b110010, jump4 = 6'b11001
1, halt3 = 6'b110100,
        alu6 = 6'b110101;
    reg [5:0] Present_state = Reset_state;
    reg setPC = 0;
    //reg [4:0] aluCode;


always @ (posedge clk, posedge reset)
    begin
        aluControl = IR[31:27];
        if (reset == 1'b1) Present_state = Reset_state;
        else case (Present_state)
            Reset_state     :   #40 Present_state = fetch0;
            fetch0          :   #40 Present_state = fetch1;
            fetch1          :   #40 Present_state = fetch2;
            fetch2          :   begin
                                    case(aluControl)
                                        5'b00000     :   #40 Present_state = load
3;
                                        5'b00001     :   #40 Present_state = load
i3;
```

```
                                        5'b00010     :   #40 Present_state = stor
e3;
                                        5'b00011     :   #40 Present_state = alu3
; //add
                                        5'b00100     :   #40 Present_state = alu3
;
                                        5'b00101     :   #40 Present_state = alu3
;
                                        5'b00110     :   #40 Present_state = alu3
;
                                        5'b00111     :   #40 Present_state = alu3
;
                                        5'b01000     :   #40 Present_state = alu3
;
                                        5'b01001     :   #40 Present_state = alu3
;
                                        5'b01010     :   #40 Present_state = alu3
;
                                        5'b01011     :   #40 Present_state = addi
3; //addi
                                        5'b01100     :   #40 Present_state = andi
3; //andi
                                        5'b01101     :   #40 Present_state = ori3
; //ori


                                        5'b01110     :   #40 Present_state = muld
iv3;
                                        5'b01111     :   #40 Present_state = muld
iv3;
                                        5'b10000     :   #40 Present_state = alur
eg3;
                                        5'b10001     :   #40 Present_state = alur
eg3;
                                        5'b10010     :   #40 Present_state = bran
ch3;

                                        5'b10011     :   #40 Present_state = jump
3; //jr
                                        5'b10100     :   #40 Present_state = jump
3; //jal
                                        5'b10011     :   #40 Present_state = inou
t3; //in
                                        5'b10110     :   #40 Present_state = inou
t3; //out
```

```
                                        5'b10111     :   #40 Present_state = mfhi
3; //mfhi
                                        5'b11000     :   #40 Present_state = mflo
3; //mflo

                                        5'b11001     :   #40 Present_state = fetc
h0; //nop

                                        5'b11010     :   #40 Present_state = stor
ei3;
                                        5'b11011     :   #40 Present_state = halt
3; //halt


                                    endcase
                                end

            //any alu related operations with three registers
            alu3            :   #40 Present_state = alu4;
            alu4            :   #40 Present_state = alu5;
            alu5            :   #40 Present_state = alu6;
            alu6            :   #40 Present_state = fetch0;

            //any alu related operations with two registers (not/neg)
            alureg3         :   #40 Present_state = alureg4;
            alureg4         :   #40 Present_state = alureg5;
            alureg5         :   #40 Present_state = fetch0;

            //mul and div
            muldiv3         :   #40 Present_state = muldiv4;
            muldiv4         :   #40 Present_state = muldiv5;
            muldiv5         :   #40 Present_state = muldiv6;
            muldiv6         :   #40 Present_state = muldiv7;
            muldiv7         :   #40 Present_state = fetch0;

            //addi
            addi3            :   #40 Present_state = addi4;
            addi4            :   #40 Present_state = addi5;
            addi5            :   #40 Present_state = fetch0;

            //load
            load3           :   #40 Present_state = load4;
            load4           :   #40 Present_state = load5;
            load5           :   #40 Present_state = load6;
            load6           :   #40 Present_state = load7;
```

```
load7           :   #40 Present_state = fetch0;

//load immediate
loadi3          :   #40 Present_state = loadi4;
loadi4          :   #40 Present_state = loadi5;
loadi5          :   #40 Present_state = fetch0;

//store
store3          :   #40 Present_state = store4;
store4          :   #40 Present_state = store5;
store5          :   #40 Present_state = store6;
store6          :   #40 Present_state = store7;
store7          :   #40 Present_state = fetch0;

//andi
andi3           :   #40 Present_state = andi4;
andi4           :   #40 Present_state = andi5;
andi5           :   #40 Present_state = fetch0;

//ori
ori3            :   #40 Present_state = ori4;
ori4            :   #40 Present_state = ori5;
ori5            :   #40 Present_state = fetch0;

//branch
branch3         :   #40 Present_state = branch4;
branch4         :   #40 Present_state = branch5;
branch5         :   #40 Present_state = branch6;
branch6         :   #40 Present_state = fetch0;

//jump
jump3           :   #40 Present_state = jump4;
jump4           :   #40 Present_state = fetch0;

//mfhi
mfhi3           :   #40 Present_state = mfhi4;
mfhi4           :   #40 Present_state = fetch0;

//mflo
mflo3           :   #40 Present_state = mflo4;
mflo4           :   #40 Present_state = fetch0;

//inout
inout3          :   #40 Present_state = fetch0;
```

```
            //store i
            storei3           :   #40 Present_state = storei4;
            storei4           :   #40 Present_state = storei5;
            storei5           :   #40 Present_state = storei6;
            storei6           :   #40 Present_state = fetch0;

            //halt
            // nothing

        endcase
    end




always @(Present_state)
    begin
         case (Present_state)
            Reset_state: begin
                PCout <= 0;       Zlowout <= 0; MDRout<= 0;   //initialize the si
gnals
                MARin <= 0;   Zin <= 0;
                PCin <=0;   MDRin <= 0;   IRin  <= 0;   Yin <= 0;
                IncPC <= 0;   Read <= 0;
                Gra<= 0;    Grb <= 0; Grc <= 0; Rin<= 0; Rout<= 0; BAout<= 0; Cou
t<=0;
                clear <= 1; run <= 1; //initialize registers to 0
                //#10 clear <= 0;
         end

            //fetch instruction
         fetch0: begin
            Zlowout <= 0;

            clear <= 0;
            PCout <= 1;
            MARin <= 1;

         end

         fetch1: begin
            PCout <= 0;
            MARin <= 0;
            Read <= 1;
            MDRin <= 1;
```

```
        end

        fetch2: begin
          Read <= 0;
          MDRin <= 0;
          MDRout<= 1;
          IRin <= 1;
          IncPC <= 1;
          #30 IncPC <= 0;
        end

           //alu three reg
        alu3:begin
          MDRout<= 0;
          IRin <= 0;
          Grb <= 1;
          Rout <= 1;
          Yin <= 1;
          #5 Grb <= 0;
        end
        alu4:begin
          //BAout <= 0;
          Yin <= 0;
          Grc <= 1;
          Rout <= 1;
          #10 Zin <= 1;
          #10 Grc <= 0;
          //#10 Rout <= 0;
        end
        alu5:begin
          Rout <= 0;
          Zin <= 0;
        end
        alu6: begin

          Zlowout <= 1;
          Gra <= 1;
          Rin <= 1;
          #10 Gra <= 0;
          #10 Rin <= 0;
        end

        //alu two reg
        alureg3:begin
          MDRout<= 0;
```

```
            IRin <= 0;
            Grb <= 1;
            BAout <= 1;
            Yin <= 1;
            #5 Grb <= 0;
          end
         alureg4:begin
            BAout <= 0;
            Yin <=0;
            #5 Zin <= 1;
          end
          alureg5:begin
            Zin <= 0;
            Zlowout <= 1;
            Gra <= 1;
            Rin <= 1;
            #10 Gra <= 0;
            #10 Rin <= 0;
          end

          //muldiv
          muldiv3:begin
            MDRout<= 0;
            IRin <= 0;
            Gra <= 1;
            BAout <= 1;
            Yin <= 1;
            #5 Gra <= 0;
          end
          muldiv4:begin
             Yin <= 0;
             Grb <= 1;
             BAout <= 1;
             #5 Zin <= 1;
             #10 Grb <= 0;
          end
          muldiv5:begin
            Zin <= 0;
            BAout <= 0;
          end
          muldiv6: begin
            //ZHIout <= 1;
            //RHiIn <= 1;
            Zlowout <= 1;
            RLOIn <= 1;
```

```
        end
        muldiv7: begin
          Zlowout <= 0;
          RLOIn <= 0;
          #5 ZHIout <= 1;
          #5 RHiIn <= 1;
          #10 RHiIn <= 0;
          #10 ZHIout <= 0;
        end

        //addi
        addi3:begin
          MDRout<= 0;
          IRin <= 0;
          Grb <= 1;
          BAout <= 1;
          Yin <= 1;
          #5 Grb <= 0;
        end
        addi4:begin
           BAout <= 0;
           Yin <= 0;
           Cout <=1;
           //aluCode <= 5'b00011;
           Zin <=1;
        end
        addi5:begin
          Cout <= 0;
          Zin <=0;
          Zlowout <= 1;
          Gra <= 1;
          Rin <= 1;
          #10 Gra <= 0;
          #10 Rin <= 0;
        end

           //load
        load3:begin
           MDRout<= 0;
              IRin <= 0;
              Grb <= 1;
              BAout <= 1;
              Yin <= 1;
          #5 Grb <= 0;
```

```
        end

        load4:begin
           BAout <= 0;
              Yin <= 0;
              Cout <=1;
              //aluCode <= 5'b00011;
              Zin <=1;

        end
           load5:begin
          Cout <= 0;
              Zin <=0;
              Zlowout <= 1;
              MARin <= 1;
        end

           load6:begin
          Zlowout <=0;
             MARin<=0;
             Read <= 1;
             MDRin <= 1;
        end

           load7:begin
          Read <= 0;
             MDRin <= 0;
          MDRout <= 1;
             Gra <= 1;
             Rin <= 1;
             #15 Gra <= 0;
              #15 Rin <= 0;

        end

           //load immediate
           loadi3:begin
              MDRout<= 0;
              IRin <= 0;
              Grb <= 1;
              BAout <= 1;
              Yin <= 1;
              #5 Grb <= 0;

           end
```

```
 loadi4:begin
    BAout <= 0;
    Yin <= 0;
    Cout <= 1;
    //aluCode <= 5'b00011;
    Zin <= 1;

 end

 loadi5:begin
    Cout <= 0;
    Zin <=0;
    Zlowout <= 1;
    Gra <= 1;
    Rin <= 1;
    #15 Gra <= 0;
    #15 Rin <= 0;
 end

 //store
store3:begin
    MDRout<= 0;
    IRin <= 0;
    Gra <= 1;
    BAout <= 1;
    Yin <= 1;
    #5 Gra <= 0;
 end
store4:begin
   BAout <= 0;
   Yin <= 0;
   Cout <=1;
   //aluCode <= 5'b00011;
   Zin <=1;
end
store5:begin
  Cout <= 0;
  Zin <=0;
  Zlowout <= 1;
  MARin <= 1;
end
store6:begin
   Zlowout <=0;
   MARin <= 0;
```

```
            Grb <= 1;
            BAout <= 1;
            MDRin <= 1;
        end
        store7:begin
            MDRin <= 0;
            BAout <= 0;
            Grb <= 0;
            Write <= 1;
            #20 Write <= 0;
        end

        //store immediate
        storei3:begin
             MDRout <= 0;
             IRin <= 0;
             Gra <= 1;
             BAout <= 1;
             Yin <= 1;
             MDRin <= 1;
             #5 Gra <= 0;
         end
        storei4:begin
            MDRin <= 0;
            BAout <= 0;
            Yin <= 0;
            //Zin <= 1;
        end
        storei5:begin
            Cout <= 1;
            //aluCode <= 5'b00011;
            //Zin <= 1;
            MARin <= 1;
        end
        storei6:begin
            Zlowout <= 0;
            MARin <= 0;
            Cout <= 0;
            //MDRin <= 1;
            Write <= 1;
            #20 Write <= 0;
            //#10 MDRin <= 0;
        end

        // and immediate
```

```
            andi3: begin
                MDRout<= 0;
                IRin <= 0;
                Grb <= 1;
                Rout <= 1;
                Yin <= 1;
             #5 Grb <= 0;
            end
            andi4: begin
                Rout <= 0;
                Yin <= 0;
                Cout <= 1;
            //aluCode <= 5'b01001;
                Zin <=1;
            end
            andi5: begin
                Cout <= 0;
                Zin <=0;
                Zlowout <= 1;
                Gra <= 1;
                Rin <= 1;
                #10 Gra <= 0;
                #10 Rin <= 0;
            end

            //or immediate
            ori3: begin
                MDRout<= 0;
                IRin <= 0;
                Grb <= 1;
                Rout <= 1;
                Yin <= 1;
            #5 Grb <= 0;
            end
            ori4: begin
                Rout <= 0;
                Yin <= 0;
                Cout <= 1;
                //aluCode <= 5'b01010; // Or
                Zin <=1;
            end
            ori5: begin
                Cout <= 0;
                Zin <= 0;
                Zlowout <= 1;
```

```
            Gra <= 1;
            Rin <= 1;
            #10 Gra <= 0;
            #10 Rin <= 0;
        end

        //branch
        branch3: begin
            MDRout <= 0;
            IRin <= 0;
            Gra <= 1;
            Rout <= 1;
            #5 ConIn <= 1;
        end
        branch4: begin
            if (CONFF) begin
              setPC = 1;
            end
            Gra <= 0;
            Rout <= 0;
            ConIn <= 0;
            //Cout <= 1;
            //#5 PCin <= 1;
            PCout <= 1;
            Yin <= 1;
        end
        branch5: begin
            //PCin <= 0;
            PCout <= 0; Yin <= 0;
            Cout <= 1;
            //aluCode <= 5'b00011; // ADD
            Zin <= 1;
        end
        branch6: begin
            #3 Cout <= 0;
            Zin <= 0;
            Zlowout <= 1;
            if (setPC) begin
              PCin = 1;
            end
            #30 PCin <= 0;
        end

        //jump
        jump3: begin
```

```
            Gra <= 1;
            Rout <= 1;
            PCin <= 1;
            BAout <= 1;
            #30 PCin <= 0;
        end
        jump4: begin
            Gra <= 0;
            Rout <= 0;
            BAout <= 0;
        end

        //mfhilo
        mflo3: begin
            LOout <= 1;
            Gra <= 1;
            Rin <= 1;
        end
        mflo4: begin
            LOout <= 0;
            Gra <= 0;
            Rin <= 0;
        end

        mfhi3: begin
            HIout <= 1;
            Gra <= 1;
            Rin <= 1;
        end
        mfhi4: begin
           HIout <= 0;
           Gra <= 0;
           Rin <= 0;
        end

        //inout
        inout3: begin
            MDRout <= 0;
            IRin <= 0;
            Gra <= 1;
            Rout <= 1;
            RoutPIn <= 1;
            #10 Gra <= 0;
            #10 Rout <= 0;
            #10 RoutPIn <= 0;
```

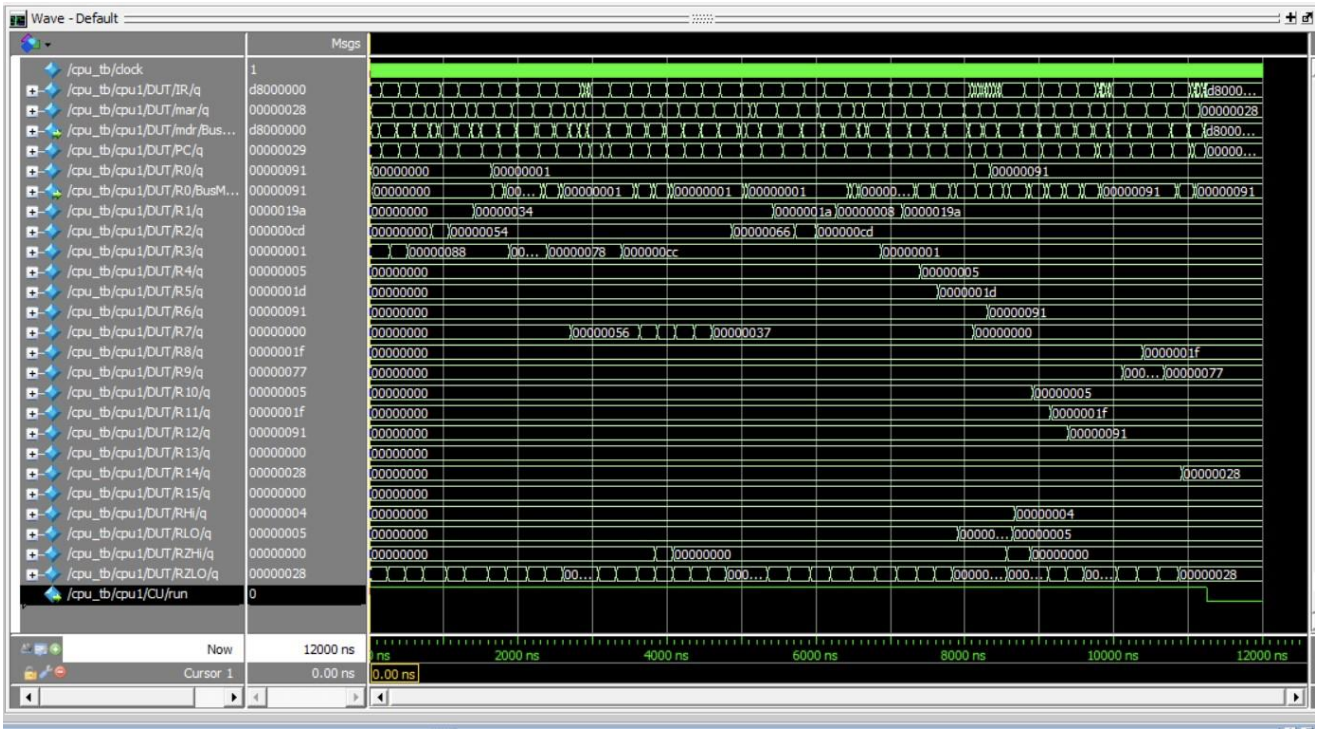
```
            end

            //halt
            halt3: begin
               run <= 0;
            end

      endcase

end
endmodule
```

Functional Simulation Run



Memory Contents

Contents before program run

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
000	09800087	09980001	01000075	0917FFFE	00900004	08000001	09800073	91C00003	.....
008	09980005	039FFFFD	C8000000	93A00002	0A080006	09A00002	19918000	5BB80003	.....
010	83B80000	8BB80000	63B8000F	6B880003	29180000	D1000058	38880000	41100000	.....
018	51180000	48908000	10900067	21918000	30900000	0A000005	0A80001D	72A00000	.....
020	BB800000	C3000000	7AA00000	0D200000	0DA80002	0E300000	0EB80000	A6000000	.....
028	D8000000	00000001	00000000	00000000	00000000	00000000	00000000	00000000	.....
030	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
038	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
040	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
048	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
050	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
058	00000034	00000000	00000000	00000000	00000000	00000000	00000000	00000000	4.....
060	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
068	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
070	00000000	00000000	00000000	00000000	00000000	00000056	00000000	00000000	....V...
078	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
080	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
088	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
090	00000000	1CD60000	245E8000	24CC0000	0F000028	9F000000	00000000	00000000	.....
098	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
0a0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
0a8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....

Contents after program has run