CSC / CPE – 142, SECTION 1

TEAM 5 PROJECT

PHASE 2

NAMES: ANDREW PETERSON and MARIO PALACIOS

CONTRIBUTION: 50% and 50%

Table of Contents

CSc/CPE 142
Term Project Status Report
Team #5

Complete this form by typing the requested information and include the completed form in your report after TOC. Gray cells will be filled by the instructor.

Name	*% Contribution*	*Grade*
Andrew Peterson	*50%*	
Mario Palacios	*50%*	

Please do not write in the first table

Project Report/Presentation 20%	/200
Functionality of the individual components 40%	/400
Functionality of the overall design 25%	/250
Design Approach 5%	/50
Total points	/900

A: List all the instructions that were implemented correctly and verified by the assembly program on your system:

Instructions	Was this instruction fully fuctional as verified by the assembly program provided? If no, explain. This refres to validation using the complete CPU and not its components.
Signed addition	It is functional by the assembly program provided
Signed subtraction	It is functional by the assembly program provided
Move	It is functional by the assembly program provided
SWAP	***Will show zero to signify it was not implemented.***
AND immediate	It is functional by the assembly program provided
OR immediate	It is functional by the assembly program provided
Load byte unsigned	Works but the memory component does not receive the correct address.
Store byte	Works but the memory component does not receive the correct address.
Load	Works but the memory component does not receive the correct address.
Store	Works but the memory component does not receive the correct address.
Branch on less than	It is functional by the assembly program provided

Instructions	Was this instruction fully fuctional as verified by the assembly program provided? If no, explain. This refres to validation using the complete CPU and not its components.
Branch on greater than	It is functional by the assembly program provided
Branch on equal	It is functional by the assembly program provided
jump	This instruction is not called in the assembly program.
halt	This instruction is not called in the assembly program.

B: Fill out the next table:

Individual Components	Does your system have this component?	List the student who designed and verified the block	Does it work ?	List problems with the component, if any.
ALU	Yes	Mario	It is at 90% functionality	could not implement swap
ALU control unit	Yes	Andrew	Yes, 100%	N/A
Memory Unit	Yes	Andrew	Yes, 100%	N/A
Register File	Yes	Mario	Yes, 100%	N/A
PC	Yes	Mario	Yes, 100%	N/A
IR (Instruction Mem.)	Yes	Andrew	Yes, 100%	N/A
Other registers	Yes	Mario	Yes	N/A
Multiplexors	Yes	Andrew/Mario	Yes, 100%	N/A
exception handler 1. Unknown opcode 2. Arith. Overflow …..	Yes	Andrew	Yes By default it will flush all the control values if it is an unkoen opcode	No problems, but only implemented the Unkown opcode handler
Control Units 1. main 2. forwarding 3. lw hazard detection	Yes	Andrew/Mario	Yes, 100%	N/A

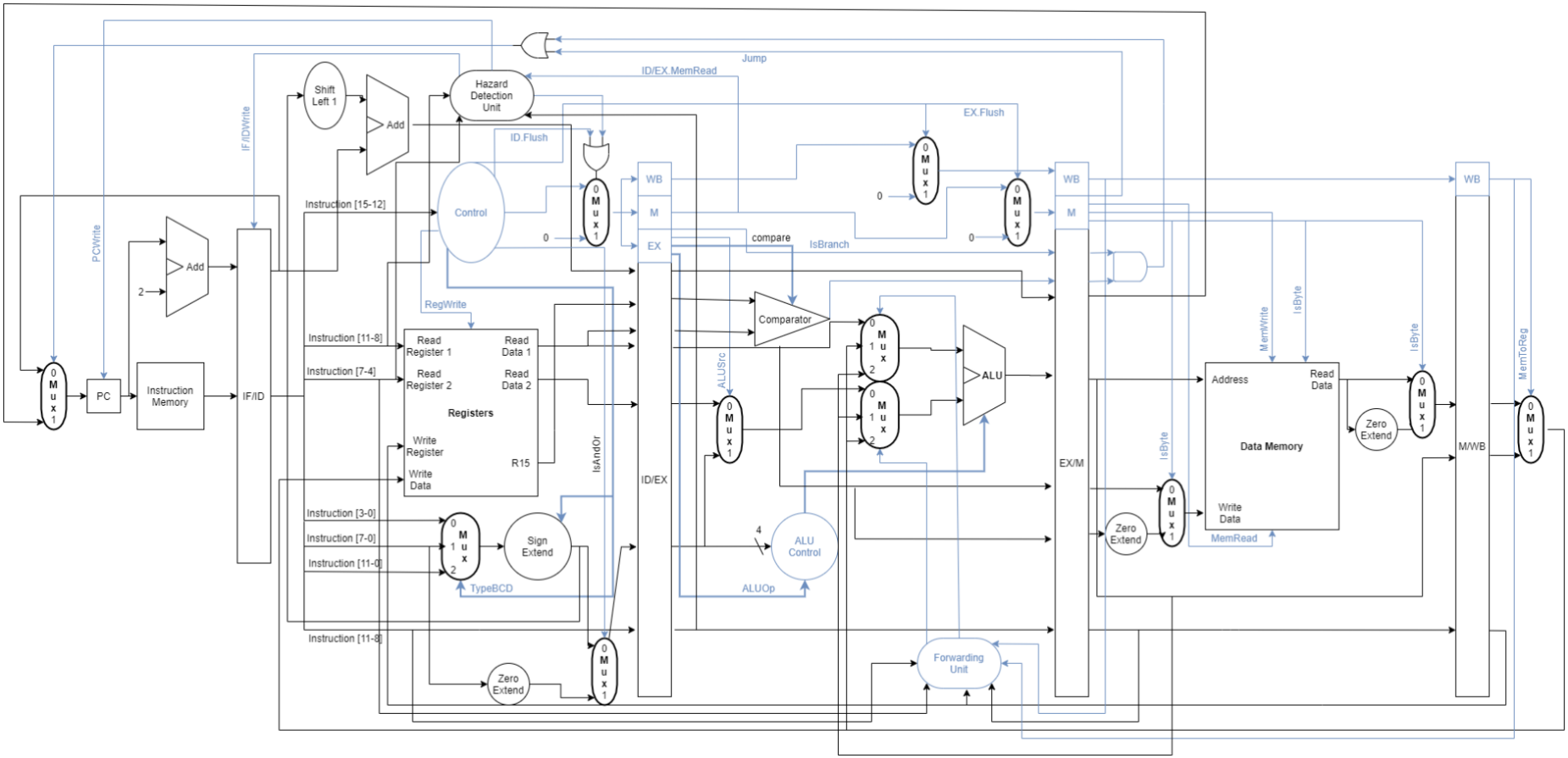
How many stages do you have in your pipeline? → Our pipleline has <u>five</u> stages.

C: **State any issue regarding the overall operation of the datapath?** **Be Specific.**

- trouble with load/store instruction, we were not able to retrieve the correct address from the ALU.
    > this led to improper values being used due to incorrect memory data placed in the registers

D. Datapath

The Datapath shown is not updated to show the changes made during the CPU (Top Level) Design.



E. Truth Table

Instruction	TypeBCD	IsBranch	ALUSrc	RegWrite	ALUOp	MemRead	MemToReg	MemWrite	IsByte	Jump	IsAndOr	compare	ID.Flush	EX.Flush
Signed Addition (A)	0	0	0	1	1XX	0	1	0	0	0	0	0	0	0
Signed Subtraction (A)	0	0	0	1	1XX	0	1	0	0	0	0	0	0	0
Move (A)	0	0	0	1	1XX	0	1	0	0	0	0	0	0	0
SWAP (A)	0	0	0	1	1XX	0	1	0	0	0	0	0	0	0
AND immediate (C)	1	0	1	1	10	0	1	0	0	0	1	0	0	0
OR immediate (C)	1	0	1	1	11	0	1	0	0	0	1	0	0	0
Load byte unsigned (B)	0	0	1	1	0	1	0	0	1	0	0	0	0	0
Store byte (B)	0	0	1	0	0	0	0	1	1	0	0	0	0	0
Load (B)	0	0	1	1	0	1	0	0	0	0	0	0	0	0
Store (B)	0	0	1	0	0	0	0	1	0	0	0	0	0	0
Branch on less than (C)	1	1	0	0	1	0	0	0	0	0	0	1	0	0
Branch on greater than (C)	1	1	0	0	1	0	0	0	0	0	0	10	0	0
Branch on equal (C)	1	1	0	0	1	0	0	0	0	0	0	11	0	0
Jump (D)	10	0	1	0	0	0	0	0	0	1	0	0	0	0
Halt (D)	10	0	0	0	0	0	0	0	0	0	0	0	0	0
Other Op Codes													1	1

F. Verilog Source Code and Fixtures

Adder

```
module adder #(parameter N=16) (input signed [N-1:0]a, b, output reg signed [N-1:0] sum);

        always@(*)
         sum <= a + b;
endmodule

`include "adder.v"
module adder_fixture;

reg [15:0] a, b;
wire [15:0] sum;

initial
        $vcdpluson;

initial
        $monitor ($time, " a= %h b= %h sum = %h", a, b, sum);


adder #(.N(16)) adder_instant (.a(a), .b(b), .sum(sum));

initial begin
                a=16'h00; b=16'h02;
        #10;    a=16'h02;
        #10;    a=16'h04;
        #10;    a=16'h06;
        #10;    a=16'h08;
        #10;    a=16'h0A;
        #10;    a=16'h0C;
        #10;    a=16'h0E;
        #10;    a=16'h10;
        #10;    a=16'h12;
        #10;    a=16'h14;
      #10    $finish;

end
endmodule
```

ALU

```
module alu #(parameter N=16) (input signed [N-1:0] a,b,
                        input [2:0] aluOp,
                        output reg [N-1:0] r);
   always@(*)
   begin
      casex(aluOp)
         3'b100: r = a + b;
```

```
            3'b101: r = a - b;
            3'b110: r = b;
            3'b111: r = 0; // SWAP?
            3'b000: r = b;
            3'b010: r = a & b;
            3'b011: r = a | b;
            default: r = 0;
        endcase
    end
endmodule

`include "alu.v"
module alu_fixture;
    reg  [15:0] a, b;
    reg  [2:0]  aluOp;
    wire [15:0] r;

    initial
        $vcdpluson;
    initial
        $monitor ($time, "CTRL = %b A = %h B = %h Result = %h", aluOp, a, b, r);

    alu #(.N(16)) alu_instant (.a(a), .b(b), .aluOp(aluOp), .r(r));

    initial begin
             aluOp = 3'b000; a = 16'h0000; b = 16'h0000;
        #10; aluOp = 3'b100; a = 16'h1234; b = 16'h0d23;
        #10; aluOp = 3'b101; a = 16'hAAAA; b = 16'h0d23;
        #10; aluOp = 3'b110; a = 16'h1234; b = 16'h0d23;
        #10; aluOp = 3'b010; a = 16'h1234; b = 16'h0d23;
        #10; aluOp = 3'b011; a = 16'h1234; b = 16'h0d23;
        #10; $finish;
    end
endmodule
```

ALU Control

```
module aluCtrl (input [2:0] opCode, input [3:0] functCode, output reg [2:0] ctrl);
    always@(*)
    begin
        casex(opCode)
            3'b1xx:
            begin
              case(functCode)
                  4'b0000: ctrl = 4'b100; // add
                  4'b0001: ctrl = 4'b101; // sub
                  4'b1110: ctrl = 4'b110; // move
                  4'b1111: ctrl = 4'b111; // swap
                  default: ctrl = 0;
              endcase
```

```
        end
        3'b000: ctrl = 4'b000; // load/store
        3'b001: ctrl = 4'b001; // branch instructions
        3'b010: ctrl = 4'b010; // and immediate
        3'b011: ctrl = 4'b011; // or immediate
        default: ctrl = 0;
      endcase
   end
endmodule

`include "aluCtrl.v"
module aluCtrl_fixture;

reg [2:0] opCode;
reg [3:0] functCode;
wire [2:0] ctrl;

initial
        $vcdpluson;

initial
        $monitor ($time, " OPCode = %b FunctCode = %b Ctrl = %b", opCode, functCode, ctrl);


aluCtrl aluCtrl_instant (.opCode(opCode), .functCode(functCode), .ctrl(ctrl));

initial begin

              opCode=3'b000; functCode=4'b0000;
        #10;     opCode=3'b001;
        #10;     opCode=3'b010;
        #10;     opCode=3'b011;
        #10;     opCode=3'b100; functCode=4'b0000;
        #10;     opCode=3'b100; functCode=4'b0001;
        #10;     opCode=3'b100; functCode=4'b1110;
        #10;     opCode=3'b100; functCode=4'b1111;
      #10    $finish;

end
endmodule
```

Comparator

```
module comparator #(parameter N=16) (input signed [N-1:0] reg15, op1, input [1:0] branch, output reg
out);

        always@(*)
   begin
      case (branch)
        2'b00: out <= 0; //No branch instruction
```

```
        2'b01: begin //Branch on less than
           if(op1 < reg15)
              out <= 1;
           else
              out <= 0;
        end
        2'b10: begin //Branch on greater than
           if(op1 > reg15)
              out <= 1;
           else
              out <= 0;
        end
        2'b11: begin //Branch on equal to
           if(op1 == reg15)
              out <= 1;
           else
              out <= 0;
        end
        default: out <= 0;
      endcase
   end
endmodule

`include "comparator.v"
module comparator_fixture;

reg [15:0] reg15, op1;
reg [1:0] branch;
wire out;

initial
        $vcdpluson;

initial
        $monitor ($time, " Register15 = %h Op1 = %h Branch Type = %b Output = %b", reg15, op1,
branch, out);


comparator #(.N(16)) comparator_instant (.reg15(reg15), .op1(op1), .branch(branch), .out(out));

initial begin

                reg15=16'h0000; op1=16'h0000; branch=2'b01;
        #10;    branch=2'b10;
        #10;    branch=2'b11;
        #10;    op1=16'h00AA; branch=2'b01;
   #10; branch=2'b10;
        #10;    branch=2'b11;
        #10;    op1=16'hFF01; branch=2'b01;
   #10; branch=2'b10;
```

```
        #10;      branch=2'b11;
    #10     $finish;

end
endmodule
```

```
Control
module ctrl (input [3:0] opCode,
          output reg [1:0] typeBCD, compare,
          output reg [2:0] aluOp,
          output reg isBranch, aluSrc, regWrite, memRead,
          memToReg, memWrite, isByte, jump, isAndor, idFlush, exFlush);

   always@(*)
   begin
      case(opCode)
         // MOVE, SWAP, sSUB, sADD
         4'b0001:
         begin
            typeBCD = 2'b00; compare = 2'b00; aluOp = 3'b1XX;
            isBranch = 0; aluSrc = 0; regWrite = 1; memRead = 0;
            memToReg = 1; memWrite = 0; isByte = 0; jump = 0;
            isAndor = 0; idFlush <= 0; exFlush <= 0;
         end

         // ANDi
         4'b1001:
         begin
            typeBCD = 2'b01; compare = 2'b00; aluOp = 3'b010;
            isBranch = 0; aluSrc = 1; regWrite = 1; memRead = 0;
            memToReg = 1; memWrite = 0; isByte = 0; jump = 0;
            isAndor = 1; idFlush <= 0; exFlush <= 0;
         end

         // ORi
         4'b1010:
         begin
            typeBCD = 2'b01; compare = 2'b00; aluOp = 3'b011;
            isBranch = 0; aluSrc = 1; regWrite = 1; memRead = 0;
            memToReg = 1; memWrite = 0; isByte = 0; jump = 0;
            isAndor = 1; idFlush <= 0; exFlush <= 0;
         end

         // LOAD BYTE UNASSIGNED
         4'b0100:
         begin
            typeBCD = 2'b00; compare = 2'b00; aluOp = 3'b000;
            isBranch = 0; aluSrc = 1; regWrite = 1; memRead = 1;
            memToReg = 0; memWrite = 0; isByte = 1; jump = 0;
```

```
   isAndor = 0; idFlush <= 0; exFlush <= 0;
end

// STORE BYTE
4'b0101:
begin
   typeBCD = 2'b00; compare = 2'b00; aluOp = 3'b000;
   isBranch = 0; aluSrc = 1; regWrite = 0; memRead = 0;
   memToReg = 0; memWrite = 1; isByte = 1; jump = 0;
   isAndor = 0; idFlush <= 0; exFlush <= 0;
end

// LOAD
4'b0110:
begin
   typeBCD = 2'b00; compare = 2'b00; aluOp = 3'b000;
   isBranch = 0; aluSrc = 1; regWrite = 1; memRead = 1;
   memToReg = 0; memWrite = 0; isByte = 0; jump = 0;
   isAndor = 0; idFlush <= 0; exFlush <= 0;
end

// STORE
4'b0111:
begin
   typeBCD = 2'b00; compare = 2'b00; aluOp = 3'b000;
   isBranch = 0; aluSrc = 1; regWrite = 0; memRead = 0;
   memToReg = 0; memWrite = 1; isByte = 0; jump = 0;
   isAndor = 0; idFlush <= 0; exFlush <= 0;
end

// BREANCH ON LESS
4'b1100:
begin
   typeBCD = 2'b01; compare = 2'b01; aluOp = 3'b001;
   isBranch = 1; aluSrc = 0; regWrite = 0; memRead = 0;
   memToReg = 0; memWrite = 0; isByte = 0; jump = 0;
   isAndor = 0; idFlush <= 0; exFlush <= 0;
end

// BRANCH ON GREATER
4'b1101:
begin
   typeBCD = 2'b01; compare = 2'b10; aluOp = 3'b001;
   isBranch = 1; aluSrc = 0; regWrite = 0; memRead = 0;
   memToReg = 0; memWrite = 0; isByte = 0; jump = 0;
   isAndor = 0; idFlush <= 0; exFlush <= 0;
end

// BRANCH ON EQUAL
4'b1110:
```

```
        begin
           typeBCD = 2'b01; compare = 2'b11; aluOp = 3'b001;
           isBranch = 1; aluSrc = 0; regWrite = 0; memRead = 0;
           memToReg = 0; memWrite = 0; isByte = 0; jump = 0;
           isAndor = 0; idFlush <= 0; exFlush <= 0;
        end

        // JUMP
        4'b0010:
        begin
           typeBCD = 2'b10; compare = 2'b00; aluOp = 3'b000;
           isBranch = 0; aluSrc = 1; regWrite = 0; memRead = 0;
           memToReg = 0; memWrite = 0; isByte = 0; jump = 1;
           isAndor = 0; idFlush <= 0; exFlush <= 0;
        end

        // HALT
        4'b0011:
        begin
           typeBCD = 2'b10; compare = 2'b00; aluOp = 3'b000;
           isBranch = 0; aluSrc = 0; regWrite = 0; memRead = 0;
           memToReg = 0; memWrite = 0; isByte = 0; jump = 0;
           isAndor = 0; idFlush <= 0; exFlush <= 0;
        end

        default:
        begin
           idFlush <= 1;
           exFlush <= 1;
        end
      endcase
   end
endmodule
```

EX/MEM Buffer

```
module EXMEMBuffer #(parameter N=16) (input [N-1:0] newPC, newALURes, newOp1,
                        input [3:0] newWriteReg,
                        input clk, reset, newIsBranch, newCompare, newIsByte, newMemWrite,
newMemRead, newMemToReg, newJump, newRegWrite,
                        output reg [N-1:0] pc, aluRes, op1,
                        output reg [3:0] writeReg,
                        output reg isBranch, compare, isByte, memWrite, memRead, memToReg, jump,
regWrite);

          always@(posedge clk or negedge reset)
   begin
     if(!reset)
      begin
        pc <= 16'h0; aluRes <= 16'h0; op1 <= 16'h0;
```

```
        writeReg <= 4'h0;
        isBranch <= 0; compare <= 0; isByte <= 0; memWrite <= 0;
        memRead <= 0; memToReg <= 0; jump <= 0; regWrite <= 0;
      end
      else
      begin
        pc <= newPC; aluRes <= newALURes; op1 <= newOp1;
        writeReg <= newWriteReg;
        isBranch <= newIsBranch; compare <= newCompare; isByte <= newIsByte; memWrite <=
newMemWrite;
        memRead <= newMemRead; memToReg <= newMemToReg; jump <= newJump; regWrite <=
newRegWrite;
      end
   end

endmodule

`include "EXMEMBuffer.v"
module EXMEMBuffer_fixture;

reg [15:0] newPC, newALURes, newOp1;
reg [3:0] newWriteReg;
reg clk, reset, newIsBranch, newCompare, newIsByte, newMemWrite, newMemRead, newMemToReg,
newJump, newRegWrite;
wire [15:0] pc, aluRes, op1;
wire [3:0] writeReg;
wire isBranch, compare, isByte, memWrite, memRead, memToReg, jump, regWrite;

initial
        $vcdpluson;

initial
        $monitor ($time, " Sample Values - Incoming: (ALU Result = %h MemWrite = %b), Outgoing:
(ALU Result = %h MemWrite = %b)", newALURes, newMemWrite, aluRes, memWrite);

EXMEMBuffer #(.N(16)) EXMEMBuffer_instant (.newPC(newPC), .newALURes(newALURes),
.newOp1(newOp1), .newWriteReg(newWriteReg),
                        .clk(clk), .reset(reset), .newIsBranch(newIsBranch), .newCompare(newCompare),
                        .newIsByte(newIsByte), .newMemWrite(newMemWrite),
.newMemRead(newMemRead), .newMemToReg(newMemToReg),
                        .newJump(newJump), .newRegWrite(newRegWrite), .pc(pc), .aluRes(aluRes),
.op1(op1),
                        .writeReg(writeReg), .isBranch(isBranch), .compare(compare), .isByte(isByte),
                        .memWrite(memWrite), .memRead(memRead), .memToReg(memToReg),
.jump(jump), .regWrite(regWrite));

initial begin
   clk = 0;
   forever #5 clk = ~clk;
end
```

```
initial begin

                reset = 0; newALURes=16'h0; newMemWrite=0;
          #10;    reset = 1;
          #10;    newALURes=16'hABDF; newMemWrite=1;
          #10;    newALURes=16'h1289; newMemWrite=0;
       #10     $finish;

end
endmodule
```

Forwarding Unit

```
module fwding #(parameter N=4)(input exmMemReg, mwbMemReg,
                        input  [N-1:0] op1, op2, exmemOp1, memwbOp1,
                        output reg [1:0] fwdA, fwdB);

   // Forward A
          always@(*)
          begin
       if(exmMemReg && ((exmemOp1 != 0) && (exmemOp1 == op1)))
          fwdA <= 2'b10;
                    else if(mwbMemReg && ((memwbOp1 != 0) && (memwbOp1 == op1)))
                              fwdA <= 2'b01;
       else
          fwdA <= 2'b00;
           end

           // Forward B
           always@(*)
           begin
       if(exmMemReg && ((exmemOp1 != 0) && (exmemOp1 == op2)))
          fwdB <= 2'b10;
       else if(mwbMemReg && ((memwbOp1 != 0) && (memwbOp1 == op2)))
                              fwdB <= 2'b01;
       else
                              fwdB <= 2'b00;
           end
endmodule

`include "fwding.v"
module fwding_fixture;
   reg exmMemReg, mwbMemReg;
   reg  [3:0] op1, op2, exmemOp1, memwbOp1;
   wire [1:0] fwdA, fwdB;

   initial
      $vcdpluson;
```

```
   initial
      $monitor ($time, "FwdA = %b FwdB = %b", fwdA, fwdB);

   fwding #(.N(4)) fwding_instant (.exmMemReg(exmMemReg), .mwbMemReg(mwbMemReg),
.op1(op1),
                                   .op2(op2), .exmemOp1(exmemOp1), .memwbOp1(memwbOp1),
                                   .fwdA(fwdA), .fwdB(fwdB));

   initial begin
      exmemOp1 = 4'b0; exmMemReg = 1'b0; memwbOp1 =4'b0;
      op1 = 4'b0; op2 = 4'b0; mwbMemReg = 1'b0;

      #10; exmemOp1 = 4'b0110; exmMemReg = 4'b1;
      #10; op1 = 4'b0110;
      #10; op2 = 4'b0110;
      #10; memwbOp1 = 4'b1100; mwbMemReg = 4'b1;
      #10; op1= 4'b1100;
      #10; op2 = 4'b1100;
      #10; op1 = 4'b1111; op2 = 4'b1111;
      #10; $finish;
   end
endmodule
```

Hazard Detection Unit

```
module hazDetect #(parameter N=4) (input [N-1:0]ifidOp1, ifidOp2, idexOp1, input idexMemRead,
output reg noOp, ifidWrite, pcWrite);

          always@(*)
           if(idexMemRead && (ifidOp1 == idexOp1 || ifidOp2 == idexOp1))
           begin
             noOp <= 1;
             ifidWrite <= 1;
             pcWrite <= 1;
           end
           else
           begin
              noOp <= 0;
             ifidWrite <= 0;
             pcWrite <= 0;
           end
endmodule

`include "hazDetect.v"
module hazDetect_fixture;

reg [3:0] ifidOp1, ifidOp2, idexOp1;
reg idexMemRead;
wire noOp, ifidWrite, pcWrite;
```

```
initial
	$vcdpluson;

initial
	$monitor ($time, " ID/EX.MemRead = %b %b = %b OR %b = %b Hazard Detection = %b",
idexMemRead, idexOp1, ifidOp1, idexOp1, ifidOp2, noOp);


hazDetect #(.N(4)) hazDetect_instant (.ifidOp1(ifidOp1), .ifidOp2(ifidOp2), .idexOp1(idexOp1),
.idexMemRead(idexMemRead), .noOp(noOp), .ifidWrite(ifidWrite), .pcWrite(pcWrite));

initial begin

		idexMemRead=0; ifidOp1=4'b0; ifidOp2=4'b0; idexOp1=4'b0;
	#10;	ifidOp1=4'b0; ifidOp2=4'b0; idexOp1=4'b0110;
	#10;	idexMemRead=1;
	#10;	ifidOp1=4'b0110;
	#10;	ifidOp1=4'b0; ifidOp2=4'b0001;
	#10;	ifidOp2=4'b0110;
	#10;	idexOp1=4'b0;
	#10;	idexMemRead=0;
    #10    $finish;

end
endmodule
```

```
ID/EX Buffer
module IDEXBuffer #(parameter N=16)(input clk, rst, isBranch, aluSrc, regWrite, memRead,
memToReg, memWrite, isByte, jump,
			input [1:0] compare,
			input [2:0] aluOp,
								input [N-1:0] rdData1,
compRdData1, rdData2, r15, muxExtend, adderRes,
								input [3:0] opCode1,
								output reg outIsBranch,
outALUSrc, outRegWrite, outMemRead, outMemToReg, outMemWrite, outIsByte, outJump,
								output reg [1:0] outCompare,
								output reg [2:0] outALUOp,
								output reg [3:0] outOpCode1,
								output reg [N-1:0]
outMuxExtend, outCompRdData1, outRdData1, outRdData2, outR15, outAdderRes);

	always@(posedge clk or negedge rst)
	begin
		if(!rst)
		begin
			outIsBranch <= 0; outALUSrc <= 0; outRegWrite <= 0; outMemRead <= 0;
outMemToReg <= 0; outJump <= 0;
			outMemWrite <= 0; outIsByte <= 0; outCompare <= 2'b00; outALUOp <=
```

```
3'b000; outOpCode1 <= 4'b0000;
                        outMuxExtend <= 16'h0000; outCompRdData1 <= 16'h0000; outRdData1 <=
16'h0000;
                        outRdData2 <= 16'h0000; outR15 <= 16'h0000; outAdderRes <= 16'h0000;
                end
                else
                begin
                        outIsBranch <= isBranch; outALUSrc <= aluSrc; outRegWrite <= regWrite;
outMemRead <= memRead; outMemToReg <= memToReg;
                        outMemWrite <= memWrite; outIsByte <= isByte; outCompare <= compare;
outJump <= jump; outALUOp <= aluOp; outOpCode1 <= opCode1;
                        outMuxExtend <= muxExtend; outCompRdData1 <= compRdData1;
outRdData1 <= rdData1;
                        outRdData2 <= rdData2; outR15 <= r15; outAdderRes <= adderRes;
                end
        end
endmodule

`include "IDEXBuffer.v"
module IDEXBuffer_fixture;
        reg clk, rst, isBranch, aluSrc, regWrite, memRead, memToReg, memWrite, isByte, jump,
   reg [1:0] compare,
   reg [2:0] aluOp,
        reg [15:0] rdData1, compRdData1, rdData2, r15, muxExtend, adderRes,
        reg [3:0] opCode1,
        wire outIsBranch, outALUSrc, outRegWrite, outMemRead, outMemToReg, outMemWrite,
outIsByte, outJump,
        wire [1:0] outCompare,
        wire [2:0] outALUOp,
        wire [3:0] outOpCode1,
        wire [N-1:0] outMuxExtend, outCompRdData1, outRdData1, outRdData2, outR15,
outAdderRes);

        initial
                $vcdpluson;
        initial
                $monitor ($time, "In:(aluSrc = %b OpCode1 = %h RdD1 = %h) Out:(Comp = %b
OutopCode1 = %h RdD1Comp = %h)", aluSrc, opCode1, rdData1, outCompare,
outOpCode1, outCompRdData1);

        IDEXBuffer #(.N(16)) IDEXBuffer_instant (.clk(clk), .rst(reset), .isBranch(isBranch),
.aluSrc(aluSrc), .regWrite(regWrite),
                        .memRead(memRead), .memToReg(memToReg), .memWrite(memWrite),
.isByte(isByte), .jump(jump),
                        .compare(compare), .aluOp(aluOp), .rdData1(rdData1),
.compRdData1(compRdData1),
                        .rdData2(rdData2), .r15(r15), .muxExtend(muxExtend), .adderRes(adderRes),
.opCode1(opCode1),
                        .outIsBranch(outIsBranch), .outALUSrc(outALUSrc),
.outRegWrite(outRegWrite),
```

```
                        .outMemRead(outMemRead), .outMemToReg(outMemToReg),
.outMemWrite(outMemWrite),
                        .outIsByte(outIsByte), .outJump(outJump), .outCompare(outCompare),
.outALUOp(outALUOp),
                        .outOpCode1(outOpCode1), .outMuxExtend(outMuxExtend),
.outCompRdData1(outCompRdData1),
                        .outRdData1(outRdData1), .outRdData2(outRdData2), .outR15(outR15),
.outAdderRes(outAdderRes));

        initial begin
                clk = 0;
                forever #5 clk = ~clk;
        end

        initial begin
                        rst = 0; aluSrc = 0; opCode1 = 4'b0; rdData1 = 16'h0;
                #10; rst = 1;
                #10; aluSrc = 1;
                #10; opCode1 = 4'b0101;
                #10; rdData1 = 16'hABC1;
    #10; rst = 0;
                #10; $finish;
        end
endmodule
```

IF/ID Buffer

```
module IFIDBuffer #(parameter N=16) (input [N-1:0] newInstr, newPC,
                        input clk, reset, ifidWrite,
                        output reg [N-1:0] instr, pc);

        always@(posedge clk or negedge reset)
  begin
    if(!reset)
     begin
        instr <= 16'h0000;
        pc <= 16'h0000;
     end
     else
     begin
        if(!ifidWrite)
        begin
           instr <= newInstr;
           pc <= newPC;
        end
     end
  end

endmodule
```

```
`include "IFIDBuffer.v"
module IFIDBuffer_fixture;

reg [15:0] newInstr, newPC;
reg clk, reset, ifidWrite;
wire [15:0] instr, pc;

initial
        $vcdpluson;

initial
        $monitor ($time, " Incoming: (pc = %h instruction = %h), IF/ID.Write = %b, Outgoing: (pc = %h
instruction = %h)", newPC, newInstr, ifidWrite, pc, instr);

IFIDBuffer #(.N(16)) IFIDBuffer_instant (.newInstr(newInstr), .newPC(newPC), .clk(clk), .reset(reset),
.ifidWrite(ifidWrite), .instr(instr), .pc(pc));

initial begin
   clk = 0;
   forever #5 clk = ~clk;

end

initial begin

            reset = 0; newPC=16'h0; newInstr=16'h0; ifidWrite=0;
        #10;    reset = 1;
        #10;    newPC=16'h02; newInstr=16'h00A1;
        #10;    newPC=16'h04; newInstr=16'h00A3;
        #10;    ifidWrite=1; newPC=16'h06; newInstr=16'h0FAD;
   #10;    ifidWrite=0; newPC=16'h06; newInstr=16'h0FAD;
      #10     $finish;

end
endmodule
```

Instruction Memory

```
module instrMem #(parameter MSize=16'h8000) (input [15:0] pc, input reset, output reg [15:0] instr);

integer index;
reg [7:0] data[MSize-1:0];

        always@(pc or negedge reset) //can't use "*" because then it infinitely tries to reset when reset is
activated
   begin
      if(!reset)
      begin
         for(index = 0; index < MSize; index = index + 1)
         begin
```

```
   data[index] <= 8'b0;
end

data[16'h00] <= 8'b0001_0001; //ADD R1, R2
data[16'h01] <= 8'b0010_0000;

data[16'h02] <= 8'b0001_0010; //SUB R2, R13
data[16'h03] <= 8'b1101_0001;

data[16'h04] <= 8'b0001_0100; //MOV R4, R8
data[16'h05] <= 8'b1000_1110;

data[16'h06] <= 8'b1010_1000;//OR R8, 0000
data[16'h07] <= 8'b0000_0000;

data[16'h08] <= 8'b0001_0100; //SWP R4, R6
data[16'h09] <= 8'b0110_1111;

data[16'h0A] <= 8'b0100_0111; //LBU R7, 4(R9)
data[16'h0B] <= 8'b1001_0100;

data[16'h0C] <= 8'b1001_0011; //ANDi R3, 4C
data[16'h0D] <= 8'b0100_1100;

data[16'h0E] <= 8'b0001_1110; //SUB R14, R14
data[16'h0F] <= 8'b1110_0001;

data[16'h10] <= 8'b0101_0111; //SB R7, 6(R9)
data[16'h11] <= 8'b1001_0110;

data[16'h12] <= 8'b0110_0110; //LW R6, 8(R9)
data[16'h13] <= 8'b1001_1000;

data[16'h14] <= 8'b1110_0111; //BEQ R7, 4
data[16'h15] <= 8'b0000_0100;

data[16'h16] <= 8'b0001_1011; //ADD R11, R1
data[16'h17] <= 8'b0001_0000;

data[16'h18] <= 8'b1100_0111; //BLT R7, 5
data[16'h19] <= 8'b0000_0101;

data[16'h1A] <= 8'b0001_1011; //ADD R11, R2
data[16'h1B] <= 8'b0010_0000;

data[16'h1C] <= 8'b1101_0111; //BGT R7, 2
data[16'h1D] <= 8'b0000_0010;

data[16'h1E] <= 8'b0001_0001; //ADD R1, R1
data[16'h1F] <= 8'b0001_0000;
```

```
            data[16'h20] <= 8'b0001_0001; //ADD R1, R1
            data[16'h21] <= 8'b0001_0000;

            data[16'h22] <= 8'b0110_1000; //LW R8, 0(R9)
            data[16'h23] <= 8'b1001_0000;

            data[16'h24] <= 8'b0001_1000; //ADD R8, R8
            data[16'h25] <= 8'b1000_0000;

            data[16'h26] <= 8'b0111_1000; //SW R8, 2 (R9)
            data[16'h27] <= 8'b1001_0010;

            data[16'h28] <= 8'b0110_1010; //LW R10, 2 (R9)
            data[16'h29] <= 8'b1001_0010;

            data[16'h2A] <= 8'b0001_1100; //ADD R12, R10
            data[16'h2B] <= 8'b1010_0000;

            data[16'h2C] <= 8'b0001_1100; //SUB R12, R13
            data[16'h2D] <= 8'b1101_0001;

            data[16'h2E] <= 8'b0001_1100; //ADD R12, R13
            data[16'h2F] <= 8'b1101_0000;

            data[16'h30] <= 8'b1110_1111; //EF20
            data[16'h31] <= 8'b0010_0000;
        end

        else
        begin
            instr <= {data[pc], data[pc + 1]};
        end
    end
endmodule

`include "instrMem.v"
module instrMem_fixture;

reg [15:0] pc;
reg reset;
wire [15:0] instr;

initial
        $vcdpluson;

initial
        $monitor ($time, " pc = %h instruction = %h", pc, instr);
```

```
instrMem #(.MSize(16'h8000)) instrMem_instant (.pc(pc), .reset(reset), .instr(instr));

initial begin

                reset = 0;   pc=16'h0;
          #10;      reset = 1;
          #10;      pc=16'h02;
          #10;      pc=16'h04;
          #10;      pc=16'h06;
          #10;      pc=16'h08;
          #10;      pc=16'h0A;
          #10;      pc=16'h0C;
          #10;      pc=16'h0E;
          #10;      pc=16'h10;
          #10;      pc=16'h12;
    #10;  pc=16'h14;
          #10;      pc=16'h16;
          #10;      pc=16'h18;
          #10;      pc=16'h1A;
          #10;      pc=16'h1C;
          #10;      pc=16'h1E;
    #10;  pc=16'h20;
          #10;      pc=16'h22;
          #10;      pc=16'h24;
          #10;      pc=16'h26;
          #10;      pc=16'h28;
          #10;      pc=16'h2A;
    #10;  pc=16'h2C;
          #10;      pc=16'h2E;
          #10;      pc=16'h30;
          #10;      pc=16'h32;
          #10;      pc=16'h34;
          #10;      pc=16'h36;
       #10     $finish;

end
endmodule
```

Data Memory

```
module mem #(parameter MSize=16'h8000) (input [15:0] inData, address, input reset, memWrite,
memRead, isByte, output reg [15:0] outData);

integer index;
reg [7:0] data[MSize-1:0];

          always@(inData or address or memWrite or memRead or isByte or negedge reset) //can't use "*"
because then it infinitely tries to reset when reset is activated
    begin
       if(!reset)
```

```
begin
   for(index = 0; index < MSize; index = index + 1)
   begin
      data[index] <= 8'b0;
   end

   data[16'h00] <= 8'b0011_1100; //3CAD
   data[16'h01] <= 8'b1010_1101;

   data[16'h02] <= 8'b0000_0000; //0000
   data[16'h03] <= 8'b0000_0000;

   data[16'h04] <= 8'b0001_0100; //1463
   data[16'h05] <= 8'b0110_0011;

   data[16'h06] <= 8'b1101_1010; //DAED
   data[16'h07] <= 8'b1110_1101;

   data[16'h08] <= 8'b1111_1110; //FEEB
   data[16'h09] <= 8'b1110_1011;

   data[16'h0A] <= 8'b1111_1111; //FFFF
   data[16'h0B] <= 8'b1111_1111;

   data[16'h0E] <= 8'b1100_1100; //CCCC
   data[16'h0F] <= 8'b1100_1100;
end

else
if(memWrite)
begin
   if(isByte)
   begin
      data[address] <= inData[7:0];
   end

   else
   begin
      {data[address], data[address + 1]} <= inData[15:0];
   end
end

else
if(memRead)
begin
   if(isByte)
   begin
      outData <= {8'b0, data[address]};
   end
```

```
        else
        begin
           outData <= {data[address], data[address + 1]};
        end
      end

   end
endmodule

`include "mem.v"
module mem_fixture;

reg [15:0] inData, address;
reg reset, memWrite, memRead, isByte;
wire [15:0] outData;

initial
        $vcdpluson;

initial
        $monitor ($time, " data = %h address = %h output = %h", inData, address, outData);


mem #(.MSize(16'h8000)) mem_instant (.inData(inData), .address(address), .reset(reset),
.memWrite(memWrite),
    .memRead(memRead), .isByte(isByte), .outData(outData));

initial begin

              reset = 0; inData=16'h0; address=16'h0; memWrite=0; memRead=0; isByte=0;
        #10;      reset = 1; memRead=1;
   #10;    isByte=1;
   #10;    memRead=0; isByte=0; inData=16'h1234; address=16'h02; memWrite=1;
   #10;    memWrite=0; memRead=1;
   #10;    memRead=0; isByte=1; inData=16'h56; address=16'h04; memWrite=1;
   #10;    memWrite=0; memRead=1;
   #10;
      #10     $finish;

end
endmodule
```

```
2 by 1 Mux
module mux2 #(parameter N = 16) (input signed [N-1:0] a,b,
                        input s,
                        output reg signed [N-1:0] out);
   always@(*)
   begin
      case(s)
```

```
        1'b0: out <= a;
        1'b1: out <= b;
        default: out <= 0;
      endcase
   end
endmodule

`include "mux2.v"
module mux2_fixture;
   reg  [15:0] a,b;
   reg  [1:0]  s;
   wire [15:0] out;

   initial
      $vcdpluson;
   initial
      $monitor ($time, "a = %h b = %h Signal = %h Output = %h", a, b, s, out);

   mux2 #(.N(16)) mux2_instant (.a(a), .b(b), .s(s), .out(out));

   initial begin
         a = 16'h0001; b= 16'h0FAD; s = 2'b11;
      #10; s = 2'b00;
      #10; s = 2'b01;
      #10; s = 2'b11;
      #10; $finish;
   end
endmodule
```

---

3 by 1 Mux

```
module mux3 #(parameter N=16) (input signed [N-1:0]a, b, c, input [1:0] s, output reg signed [N-1:0]
out);

         always@(*)
   begin
      case (s)
         2'b00: out <= a;
         2'b01: out <= b;
         2'b10: out <= c;
         2'b11: out <= 0;
         default: out <= 0;
      endcase
   end
endmodule

`include "mux3.v"
module mux3_fixture;

reg [15:0] a, b, c;
```

```
reg [1:0] s;
wire [15:0] out;

initial
	$vcdpluson;

initial
	$monitor ($time, " a = %h b = %h c = %h signal = %h out = %h", a, b, c, s, out);


mux3 #(.N(16)) mux3_instant (.a(a), .b(b), .c(c), .s(s), .out(out));

initial begin

		a=16'h0001; b=16'h0FAD; c=16'hFFFE; s=2'b11;
	#10;	s=2'b00;
	#10;	s=2'b01;
	#10;	s=2'b10;
	#10;	s=2'b11;
    #10    $finish;

end
endmodule
```

M/WB Buffer

```
module MWBBuffer #(parameter N=16)(input clk, rst, wb, regWrite,
                                                                input [N-1:0] aluRes, mux,
                                                                input [3:0] opCode1,
                                                                output reg memToReg, outRegWrite,
                                                                output reg [3:0] outOpCode1,
                                                                output reg [N-1:0] outAluRes,
outMux);

	always@(posedge clk or negedge rst)
	begin
		if(!rst)
		begin
			outAluRes <= 16'h0; memToReg <= 0; outRegWrite <= 0;
			outMux <= 16'h0; outOpCode1 <= 4'b0;
		end
		else
		begin
			outAluRes <= aluRes; memToReg <= wb; outRegWrite <= regWrite;
			outMux <= mux; outOpCode1 <= opCode1;
		end
	end
endmodule
```

```
`include "MWBBuffer.v"
module IDEXBuffer_fixture;
	reg clk, rst, wb, regWrite;
	reg [15:0] aluRes, mux;
	reg [3:0] opCode1;
	wire memToReg, outRegWrite;
	wire [3:0] outOpCode1;
	wire [15:0] outAluRes, outMux;

	initial
		$vcdpluson;
	initial
		$monitor ($time, "In:(wb = %b OpCode1 = %h mux = %h) Out:(MemToReg = %b
OutopCode1 = %h MuxResult = %h)", wb, opCode1, mux, memToReg, outOpCode1, outMux);

	MWBBuffer #(.N(16)) MWBBuffer_instant (.clk(clk), .rst(rst), .wb(wb), .regWrite(regWrite),
.aluRes(aluRes), .mux(mux),
                                                                .opCode1(opCode1),
.memToReg(memToReg), .outRegWrite(outRegWrite), .outOpCode1(outOpCode1),

.outAluRes(outAluRes), .outMux(outMux));

	initial begin
		clk = 0;
		forever #5 clk = ~clk;
	end

	initial begin
			rst = 0; wb = 0; opCode1 = 4'b0; mux = 16'h0;
		#10; rst = 1;
		#10; wb = 1;
		#10; opCode1 = 4'b0101;
		#10; mux = 16'hABC1;
      #10; rst = 0;
		#10; $finish;
	end
endmodule
```

```
Program Counter
module programCounter #(parameter N=16) (input rst, pcWr,
                        input [N-1:0] pcIn,
                        output reg [N-1:0] count);

   always@(pcWr or pcIn or negedge rst)
      if (!rst)
      begin
         count <= 16'h0000;
      end
      else begin
```

```
        if (pcWr)
        begin
            count <= count;
        end
        else begin
            count <= pcIn;
        end
    end
endmodule

`include "programCounter.v"
module programCounter_fixture;
  reg rst, pcWr;
  reg [15:0] pcIn;
  wire [15:0] count;

  initial
    $vcdpluson;
  initial
    $monitor ($time, "Data In = %h count = %h PCWrite = %b Reset = %b", pcIn, count, pcWr, rst);

  programCounter #(.N(16)) programCounter_instant (.rst(rst), .pcIn(pcIn), .pcWr(pcWr), .count(count));


  initial begin
    #10;  rst = 1'b1; pcWr = 1'b0; pcIn = 16'h0F0F;
    #50;

    #10;  rst = 1'b1; pcWr = 1'b1; pcIn = 16'h0F0F;
    #50;

    #10;  rst = 1'b0; pcWr = 1'b0; pcIn = 16'h0F0F;
    #50;

    #10;  rst = 1'b0; pcWr = 1'b1; pcIn = 16'h0F11;
    #50;

    #10; $finish;
  end
endmodule
```

Register

```
module register #(parameter N = 16) (input [3:0] rd_reg1, rd_reg2, wr_reg,
                                     input [N-1:0] wr_data,
                                     input wr_enable, rst,
                                     output reg [N-1:0] r_15, rd_data1, rd_data2);
  reg [15:0] data [15:0];

  always@(*)
```

```
    begin
        rd_data1 = data[rd_reg1];
        rd_data2 = data[rd_reg2];
        r_15 = data[15];
    end

    always@(wr_enable or wr_data or wr_reg or negedge rst)
    begin
        if (!rst)
        begin
            data[0]  <= 16'h0E12;
            data[1]  <= 16'h0045;
            data[2]  <= 16'hF08F;
            data[3]  <= 16'hF076;
            data[4]  <= 16'h0084;
            data[5]  <= 16'h6789;
            data[6]  <= 16'h00EB;
            data[7]  <= 16'hFF56;
            data[8]  <= 16'h0000;
            data[9]  <= 16'h0000;
            data[10] <= 16'h0000;
            data[11] <= 16'hCC90;
            data[12] <= 16'h0002;
            data[13] <= 16'h0000;
            data[14] <= 16'h0000;
            data[15] <= 16'h0000;
        end
        else
        begin
            if (wr_enable)
            begin
                data[wr_reg] <= wr_data;
            end
        end
    end
endmodule

`include "register.v"
module register_fixture;
    reg rst, w_en;
    reg [3:0] rd_reg1, rd_reg2, wr_reg;
    reg [15:0] wr_data;
    wire [15:0] rd_data1, rd_data2, r_15;

    initial
        $vcdpluson;
    initial
        $monitor ($time, "RdReg1 = %h RdReg2 = %h WEnable = %b RdData1 = %h RdData2 = %h",
                rd_reg1, rd_reg2, w_en, rd_data1, rd_data2);
```

```
   register #(.N(16)) register_instant (.rd_reg1(rd_reg1), .rd_reg2(rd_reg2),
                                        .rst(rst), .wr_enable(w_en),
                                        .wr_data(wr_data), .wr_reg(wr_reg),
                                        .rd_data1(rd_data1), .rd_data2(rd_data2), .r_15(r_15));

   initial
   begin
         rst = 1'b0;
      #8 rst = 1'b1;
   end

   initial begin
                  rd_reg1 = 4'b0000; rd_reg2 = 4'b0000; w_en = 1'b0;
      @(posedge clk); rd_reg1 = 4'b0001; rd_reg2 = 4'b0010; w_en = 1'b0;
      @(posedge clk); rd_reg1 = 4'b0011; rd_reg2 = 4'b1001; w_en = 1'b0;
      @(posedge clk); rd_reg1 = 4'b0110; rd_reg2 = 4'b0111; w_en = 1'b0;
      @(posedge clk); rd_reg1 = 4'b1000; rd_reg2 = 4'b1010; w_en = 1'b0;
      @(posedge clk); rd_reg1 = 4'b1000; rd_reg2 = 4'b1010; w_en = 1'b1; wr_reg = 4'b1010; wr_data =
16'hFFFF;
      @(posedge clk); rd_reg1 = 4'b1000; rd_reg2 = 4'b1010; w_en = 1'b0;
      @(posedge clk); rd_reg1 = 4'b1111; rd_reg2 = 4'b0100; w_en = 1'b0;
      #10; $finish;
   end
endmodule
```

Shift Left by 1

```
module sftLeft #(parameter N=16) (input signed [N-1:0] in,
                                  output reg signed [N-1:0] out);

   always@(*)
      out <= {in << 1};
endmodule

`include "sftLeft.v"
module sftLeft_fixture;
   reg [15:0] in;
   wire [15:0] out;

   initial
      $vcdpluson;
   initial
      $monitor ($time, "Data In = %h Data Out = %h", in, out);

   sftLeft #(.N(16)) sftLeft_instant (.in(in), .out(out));

   initial begin
         in = 16'h0000;
      #10 in = 16'h1000;
      #10 in = 16'hFFFF;
```

```
      #10 in = 16'h00DC;
      #10 in = 16'h0001;
      #10 $finish;
   end
endmodule
```

Sign Extend

```
module signExtend #(parameter N = 16) (input [1:0] typeBCD,
                                       input signed [N-1:0] datain,
                                       output reg signed [N-1:0] extended);

   always@(*)
   begin
      case(typeBCD)
         2'b01: extended = {{12{datain[3]}}, datain[3:0]};
         2'b10: extended = {{8{datain[7]}}, datain[7:0]};
         2'b11: extended = {{4{datain[11]}}, datain [11:0]};
         default: extended = {{4{datain[11]}}, datain [11:0]};
      endcase
   end
endmodule

`include "signExtend.v"
module signExtend_fixture;
   reg [1:0] type;
   reg [15:0] in;
   wire [15:0] out;

   initial
      $vcdpluson;
   initial
      $monitor ($time, "Type = %b DataIn = %b DataOut = %b", type, in, out);

   signExtend #(.N(16)) signExtend_instant (.typeBCD(type), .datain(in), .extended(out));

   initial
   begin
         type = 2'b00; in = 16'h0000;
      #10; type = 2'b01; in = 4'hF;
      #10; type = 2'b10; in = 8'hF0;
      #10; type = 2'b11; in = 12'hF00;
      #10; type = 2'b01; in = 4'h0;
      #10; type = 2'b10; in = 8'hFA;
      #10; type = 2'b11; in = 12'hF01;
      #10; $finish;
   end
endmodule
```

Zero Extend

```
module zeroExt #(parameter N=16) (input signed [N-1:0] in, output reg signed [N-1:0] out);

        always@(*)
         out <= {8'b00000000, in[7:0]};
endmodule

`include "zeroExt.v"
module zeroExt_fixture;

reg [15:0] in;
wire [15:0] out;

initial
        $vcdpluson;

initial
        $monitor ($time, " in = %h out = %h", in, out);

zeroExt #(.N(16)) zeroExt_instant (.in(in), .out(out));

initial begin

                in=16'h0000;
        #10;    in=16'h0001;
        #10;    in=16'h007F;
        #10;    in=16'h00FF;
        #10;    in=16'h0100;
        #10;    in=16'h01FF;
        #10;    in=16'hFF00;
        #10;    in=16'hFFFF;
      #10    $finish;

end
endmodule
```

G. Test Assembly Program

pc	instruction	machine code
00	ADD R1, R2	0001 0001 0010 0000
02	SUB R2, R13	0001 0010 1101 0001
04	MOV R4, R8	0001 0100 1000 1110
06	OR R8, 0000	1010 1000 0000 0000
08	SWP R4, R6	0001 0100 0110 1111
0A	LBU R7, 4(R9)	0100 0111 1001 0100
0C	ANDi R3, 4C	1001 0011 0100 1100
0E	SUB R14, R14	0001 1110 1110 0001
10	SB R7, 6(R9)	0101 0111 1001 0110
12	LW R6, 8(R9)	0110 0110 1001 1000
14	BEQ R7, 4	1110 0111 0000 0100
16	ADD R11, R1	0001 1011 0001 0000
18	BLT R7, 5	1100 0111 0000 0101
1A	ADD R11, R2	0001 1011 0010 0000
1C	BGT R7, 2	1101 0111 0000 0010
1E	ADD R1, R1	0001 0001 0001 0000
20	ADD R1, R1	0001 0001 0001 0000
22	LW R8, 0(R9)	0110 1000 1001 0000
24	ADD R8, R8	0001 1000 1000 0000
26	SW R8, 2 (R9)	0111 1000 1001 0010
28	LW R10, 2 (R9)	0110 1010 1001 0010
2A	ADD R12, R10	0001 1100 1010 0000
2C	SUB R12, R13	0001 1100 1101 0001
2E	ADD R12, R13	0001 1100 1101 0000
30	EF20	1110 1111 0010 0000

I. Stimulus Module

```
`include "adder.v"
`include "alu.v"
`include "aluCtrl.v"
`include "comparator.v"
`include "ctrl.v"
`include "EXMEMBuffer.v"
`include "fwding.v"
`include "hazDetect.v"
`include "IDEXBuffer.v"
`include "IFIDBuffer.v"
`include "instrMem.v"
`include "mem.v"
`include "mux2_1bit.v"
`include "mux2_2bit.v"
`include "mux2_3bit.v"
`include "mux2.v"
`include "mux3.v"
`include "MWBBuffer.v"
`include "programCounter.v"
`include "register.v"
`include "sftLeft.v"
`include "signExtend.v"
`include "zeroExt.v"

module cpu(input clk, reset);

wire [15:0] wMux2PC, wIFID2Mux, wEXM2PCMux, wPC2Instr, wSum2IFID, wInstr2IFID, wIFIDInstr,
wR152IDEX, wEXR152Comp,
        wWBMux2WData, wRData12IDEX, wRData22IDEX, wSft2Add, wAdd2IDEX, wSignEx2Sft,
wMux32SignEx, wZeroEx2MuxID,
        wMux2IDEX, wIDEXOp1, wExAdder, wAdderID2M, wExt2Mux, wEXRData2,
wEXLSMux2ALUMux, wALUMux2MuxB, wMuxA2ALU,
        wMuxB2ALU, wALURes2EXM, wEXMOp1, w0Ext2WMux, wEXMALURes, wMMux2WData,
wMemRData, w0Ext2RMux, wRMux2MWB,
        wMWBMux, wMWBALURes;
wire [3:0] wIDEXOp12EXM, wWB2WReg, wEXMOp12MWB;
wire [2:0] wALUOp, wEXALUOp, wALUCtrl, wALUOp2IDEX;
wire [1:0] wCompare, wIDEXCompare, wFWDA, wFWDB, wCompare2IDEX, wTypeBCD;
wire wHaz2PC, wBrchOrJump, wHaz2IFID, wMWB2RegWrite, wIsAndOr, wALUSrc, wFlushHazard,
wRegWrite, wMemRead, wMemToReg, wMemWRite,
      wIsByte, wHazOp, wBranchEX, wOrMem, wEXALUSrc, wCompareOut, wRWriteMux2EXM,
wMReadMux2EXM, wM2RMux2EXM, wMWriteMux2EXM,
      wIsByteMux2EXM, wEXMIsByte, wEXMBranch, wEXMCompare, wEXMMWrite, wEXMMRead,
wEXMM2R, wEXMJump, wBrchAndComp,
      wEXMRWrite, wMWBM2Reg, wIDFlush, wEXFlush, wIsBranch, wALUSrc2IDEX, wRWrite2IDEX,
wMRead2IDEX, wM2R2IDEX, wMWrite2IDEX,
      wIsByte2IDEX, wRWrite2Mux, wMRead2Mux, wM2R2Mux, wMWrite2Mux, wIsByte2Mux, wJump,
wJump2EXM;
//IF
```

```
   mux2 #(.N(16)) mux2IF (.a(wIFID2Mux), .b(wEXM2PCMux), .s(wBrchOrJump), .out(wMux2PC)); //PC +
2 or Branch/Jump location

   programCounter #(.N(16)) pcIF (.rst(reset), .pcWr(wHaz2PC), .pcIn(wMux2PC), .count(wPC2Instr));

   instrMem #(.MSize(16'h8000)) instrMemIF (.pc(wPC2Instr), .reset(reset), .instr(wInstr2IFID));

   adder #(.N(16)) adderIF (.a(wPC2Instr), .b(16'h02), .sum(wSum2IFID));

//IF
   IFIDBuffer #(.N(16)) IFIDBuffer_instant (.newInstr(wInstr2IFID), .newPC(wSum2IFID), .clk(clk),
                                .reset(reset), .ifidWrite(wHaz2IFID), .instr(wIFIDInstr), .pc(wIFID2Mux));
//ID

   ctrl ctrlID (.opCode(wIFIDInstr[15:12]), .typeBCD(wTypeBCD), .compare(wCompare), .aluOp(wALUOp),
.isBranch(wIsBranch), .aluSrc(wALUSrc),
            .regWrite(wRegWrite), .memRead(wMemRead), .memToReg(wMemToReg),
.memWrite(wMemWRite), .isByte(wIsByte),
            .jump(wJump), .isAndor(wIsAndOr), .idFlush(wIDFlush), .exFlush(wEXFlush));

   register #(.N(16)) registerID (.rd_reg1(wIFIDInstr[11:8]), .rd_reg2(wIFIDInstr[7:4]),
.wr_reg(wWB2WReg), .wr_data(wWBMux2WData),
                              .wr_enable(wMWB2RegWrite), .rst(reset), .r_15(wR152IDEX),
                              .rd_data1(wRData12IDEX), .rd_data2(wRData22IDEX));

   adder #(.N(16)) adderID (.a(wSft2Add), .b(wIFID2Mux), .sum(wAdd2IDEX)); //Adds the shifted value to
PC + 2

   sftLeft #(.N(16)) sftLeftID (.in(wSignEx2Sft), .out(wSft2Add)); //Shifts the sign extended value to the left
once (multiplying by 2 since each instruction is 2 bytes)

   signExtend #(.N(16)) signExtendID (.typeBCD(wTypeBCD), .datain(wMux32SignEx),
.extended(wSignEx2Sft)); //Sign extends the immediate value from the instruction

   zeroExt #(.N(16)) zeroExtID (.in(wIFIDInstr), .out(wZeroEx2MuxID)); //Zero extends the 8bit value for
ANDi or ORi instructions

   mux3 #(.N(16)) mux3ID (.a(wIFIDInstr), .b(wIFIDInstr), .c(wIFIDInstr), .s(wTypeBCD),
.out(wMux32SignEx)); //Passes either the 4bit, 8bit, or 12bit immediate value to be sign extended

   mux2 #(.N(16)) mux2IDimm (.a(wSignEx2Sft), .b(wZeroEx2MuxID), .s(wIsAndOr), .out(wMux2IDEX));
//Passes either the zero extended value or the sign extended value


   // This is displayed in the datapath as a single mux
   mux2_1bit mux2IDaluSrc (.a(wALUSrc), .b(1'b0), .s(wFlushHazard), .out(wALUSrc2IDEX)); //If hazard or
flush, passes 0 instead of aluSrc

   mux2_1bit mux2IDrWrite (.a(wRegWrite), .b(1'b0), .s(wFlushHazard), .out(wRWrite2IDEX)); //If hazard
or flush, passes 0 instead of regWrite

   mux2_1bit mux2IDmRead (.a(wMemRead), .b(1'b0), .s(wFlushHazard), .out(wMRead2IDEX)); //If hazard
```

```
or flush, passes 0 instead of memRead

    mux2_1bit mux2IDm2r (.a(wMemToReg), .b(1'b0), .s(wFlushHazard), .out(wM2R2IDEX)); //If hazard or
flush, passes 0 instead of memToReg

    mux2_1bit mux2IDmWrite (.a(wMemWRite), .b(1'b0), .s(wFlushHazard), .out(wMWrite2IDEX)); //If
hazard or flush, passes 0 instead of memWrite

    mux2_1bit mux2IDisByte (.a(wIsByte), .b(1'b0), .s(wFlushHazard), .out(wIsByte2IDEX)); //If hazard or
flush, passes 0 instead of isByte

    mux2_2bit #(.N(2)) mux2IDcompare (.a(wCompare), .b(2'b0), .s(wFlushHazard), .out(wCompare2IDEX));
//If hazard or flush, passes 0 instead of compare

    mux2_3bit #(.N(3)) mux2IDaluOp (.a(wALUOp), .b(3'b0), .s(wFlushHazard), .out(wALUOp2IDEX)); //If
hazard or flush, passes 0 instead of aluOP
    //


    or orIDctrl (wFlushHazard, wIDFlush, wHazOp); //If flush or hazard

//ID
    IDEXBuffer #(.N(16)) IDEXBuffer_instant (.clk(clk), .rst(reset), .isBranch(wIsBranch),
.aluSrc(wALUSrc2IDEX), .regWrite(wRWrite2IDEX),
                                .memRead(wMRead2IDEX), .memToReg(wM2R2IDEX),
.memWrite(wMWrite2IDEX), .isByte(wIsByte2IDEX), .jump(wJump),
                                .compare(wCompare2IDEX), .aluOp(wALUOp2IDEX), .rdData1(wRData12IDEX),
.compRdData1(wRData12IDEX),
                                .rdData2(wRData22IDEX), .r15(wR152IDEX), .muxExtend(wMux2IDEX),
.adderRes(wAdd2IDEX),
                                .opCode1(wIFIDInstr[11:8]), .outIsBranch(wBranchEX), .outALUSrc(wEXALUSrc),
.outRegWrite(wRWrite2Mux),
                                .outMemRead(wMRead2Mux), .outMemToReg(wM2R2Mux),
.outMemWrite(wMWrite2Mux), .outIsByte(wIsByte2Mux), .outJump(wJump2EXM),
                                .outCompare(wIDEXCompare), .outALUOp(wEXALUOp),
.outOpCode1(wIDEXOp12EXM), .outMuxExtend(wExt2Mux),
                                .outCompRdData1(wIDEXOp1), .outRdData1(wIDEXOp1),
.outRdData2(wEXRData2), .outR15(wEXR152Comp),
                                .outAdderRes(wAdderID2M));

//EX

    adder #(.N(16)) adderEX (.a(wExt2Mux), .b(wEXRData2), .sum(wExAdder)); //Adds the sign extended
immediate value and op2

    or orEXLoadStore (wOrMem, wMRead2Mux, wMWrite2Mux); //If memRead or memWrite instruction

    mux2 #(.N(16)) mux2EXLoadStore (.a(wExt2Mux), .b(wExAdder), .s(wOrMem),
.out(wEXLSMux2ALUMux)); //If a load/store instruction, pass the imm+op2, else pass imm

    comparator #(.N(16)) comparatorEX (.reg15(wEXR152Comp), .op1(wIDEXOp1),
.branch(wIDEXCompare), .out(wCompareOut));
```

```
    mux2 #(.N(16)) mux2EXaluSrc (.a(wEXRData2), .b(wEXLSMux2ALUMux), .s(wEXALUSrc),
.out(wALUMux2MuxB)); //Passes either op2 or the sign extended immediate value (or imm+op2)

    mux3 #(.N(16)) mux3EXA (.a(wIDEXOp1), .b(wWBMux2WData), .c(wEXMALURes), .s(wFWDA),
.out(wMuxA2ALU)); //Passes from multiple sections to the ALU

    mux3 #(.N(16)) mux3EXB (.a(wALUMux2MuxB), .b(wWBMux2WData), .c(wEXMALURes),
.s(wFWDB), .out(wMuxB2ALU)); //Passes from multiple sections to the ALU

    aluCtrl aluCtrlEX (.opCode(wEXALUOp), .functCode(wExt2Mux[3:0]), .ctrl(wALUCtrl));

    alu #(.N(16)) aluEX (.a(wMuxA2ALU), .b(wMuxB2ALU), .aluOp(wALUCtrl), .r(wALURes2EXM));


    // This is displayed in the datapath as a single mux
    mux2_1bit mux2EXrWrite (.a(wRWrite2Mux), .b(1'b0), .s(wEXFlush), .out(wRWriteMux2EXM)); //If
flush, passes 0 instead of regWrite

    mux2_1bit mux2EXmRead (.a(wMRead2Mux), .b(1'b0), .s(wEXFlush), .out(wMReadMux2EXM)); //If
flush, passes 0 instead of memRead

    mux2_1bit mux2EXm2r (.a(wM2R2Mux), .b(1'b0), .s(wEXFlush), .out(wM2RMux2EXM)); //If flush,
passes 0 instead of memToReg

    mux2_1bit mux2EXmWrite (.a(wMWrite2Mux), .b(1'b0), .s(wEXFlush), .out(wMWriteMux2EXM)); //If
flush, passes 0 instead of memWrite

    mux2_1bit mux2EXisByte (.a(wIsByte2Mux), .b(1'b0), .s(wEXFlush), .out(wIsByteMux2EXM)); //If flush,
passes 0 instead of isByte
    //


//EX
    EXMEMBuffer #(.N(16)) EXMEMBuffer_instant (.newPC(wAdderID2M),
.newALURes(wALURes2EXM), .newOp1(wIDEXOp1), .newWriteReg(wIDEXOp12EXM),
                                .clk(clk), .reset(reset), .newIsBranch(wBranchEX), .newCompare(wCompareOut),
                                .newIsByte(wIsByteMux2EXM), .newMemWrite(wMWriteMux2EXM),
.newMemRead(wMReadMux2EXM), .newMemToReg(wM2RMux2EXM),
                                .newJump(wJump2EXM), .newRegWrite(wRWriteMux2EXM), .pc(wEXM2PCMux),
.aluRes(wEXMALURes), .op1(wEXMOp1),
                                .writeReg(wEXMOp12MWB), .isBranch(wEXMBranch), .compare(wEXMCompare),
.isByte(wEXMIsByte),
                                .memWrite(wEXMMWrite), .memRead(wEXMMRead), .memToReg(wEXMM2R),
.jump(wEXMJump), .regWrite(wEXMRWrite));
//MEM

    zeroExt #(.N(16)) zeroExtMWrite (.in(wEXMOp1), .out(w0Ext2WMux)); //Zero extends the incoming op1

    mux2 #(.N(16)) mux2MWrite (.a(wEXMOp1), .b(w0Ext2WMux), .s(wEXMIsByte),
.out(wMMux2WData)); //Chooses between op1 and the zero extended op1
```

```
   mem #(.MSize(16'h8000)) memM (.inData(wMMux2WData), .address(wEXMALURes), .reset(reset),
.memWrite(wEXMMWrite),
                                .memRead(wEXMMRead), .isByte(wEXMIsByte), .outData(wMemRDData));

   and andMBranch (wBrchAndComp, wEXMBranch, wEXMCompare); //Both isBranch and the comparator

   or orMBranchJump (wBrchOrJump, wBrchAndComp, wEXMJump); //If (isBranch and compare) or Jump

   zeroExt #(.N(16)) zeroExtMRead (.in(wMemRDData), .out(w0Ext2RMux)); //Zero extends the read data

   mux2 #(.N(16)) mux2MRead (.a(wMemRDData), .b(w0Ext2RMux), .s(wEXMIsByte),
.out(wRMux2MWB)); //Chooses between the read data and zero extended read data

//MEM
   MWBBuffer #(.N(16)) MWBBuffer_instant (.clk(clk), .rst(reset), .wb(wEXMM2R),
.regWrite(wEXMRWrite), .aluRes(wEXMALURes), .mux(wRMux2MWB),

.opCode1(wEXMOp12MWB), .memToReg(wMWBM2Reg), .outRegWrite(wMWB2RegWrite),
.outOpCode1(wWB2WReg),

.outAluRes(wMWBALURes), .outMux(wMWBMux));
//WB

   mux2 #(.N(16)) mux2WB (.a(wMWBMux), .b(wMWBALURes), .s(wMWBM2Reg),
.out(wWBMux2WData)); //Chooses between the read data and ALU result

//WB

   fwding #(.N(4)) fwding_instant (.exmMemReg(wEXMM2R), .mwbMemReg(wMWBM2Reg),
.op1(wIFIDInstr[11:8]),
                               .op2(wIFIDInstr[7:4]), .exmemOp1(wEXMOp12MWB), .memwbOp1(wWB2WReg),
                               .fwdA(wFWDA), .fwdB(wFWDB));

   hazDetect #(.N(4)) hazDetect_instant (.ifidOp1(wIFIDInstr[11:8]), .ifidOp2(wIFIDInstr[7:4]),
.idexOp1(wIDEXOp12EXM),
                                                                    .idexMemRead(wMRead2Mux),
.noOp(wHazOp), .ifidWrite(wHaz2IFID), .pcWrite(wHaz2PC));

endmodule

`include "cpu.v"
module cpu_fixture;

reg clk, reset;

initial
        $vcdpluson;

cpu cpu_instant (.clk(clk), .reset(reset));

always@(negedge clk)
begin
```

```
    $display("      PC Count                    = %h", cpu.wPC2Instr);
    $display("      Instruction         = %h", cpu.wInstr2IFID);

    $display("\n\n ****** REGISTER ******",);
    $display("(Rd Reg 1 = %b) AND (Rd Reg 2 = %b)", cpu.wIFIDInstr[11:8], cpu.wIFIDInstr[7:4]);
    $display("           R15 = %h", cpu.wR152IDEX);

    $display("\n\n    ****** ALU ******",);
    $display("   ALU Ctrl              = %h", cpu.wALUCtrl);
    $display("(ALU Input A) = %h (ALU Input B) = %h", cpu.wMuxA2ALU, cpu.wMuxB2ALU);
    $display("   ALU Result                  = %h", cpu.wALURes2EXM);

    $display("\n\n ****** BRANCH_CONTROL ******",);
    $display("        *** COMPARATOR(EX) ***",);
    $display("   Register 15              = %h", cpu_instant.wEXR152Comp);
    $display("   Op1 Register             = %h", cpu_instant.wIDEXOp1);
    $display("   Branch Type (<, >, =)     = %h", cpu_instant.wIDEXCompare);
    $display("   Branch Flag              = %b", cpu_instant.wCompareOut);
    $display("   isBranch                 = %b", cpu_instant.wBranchEX);
    $display("   Current PC               = %h", cpu_instant.wPC2Instr);
    $display("   New PC (PC+2+ShiftOp2)    = %h", cpu_instant.wAdderID2M);

    $display("\n\n ****** LOAD/STORE ******",);
    $display("          *** EX ***",);
    $display("   ALU Result               = %h", cpu_instant.wEXMALURes);
    $display("   Read from Memory?        = %b", cpu_instant.wEXMMRead);
    $display("   Write to Memory?         = %b", cpu_instant.wEXMMWrite);
    $display("   Is Byte?                 = %b", cpu_instant.wEXMIsByte);
    $display("   InData                   = %h", cpu_instant.wMMux2WData);
    $display("   OutData                  = %h", cpu_instant.wMemRData);

    $display("          *** WB ***",);
    $display("   Mem or ALU to Reg?       = %b", cpu_instant.wMWBM2Reg);
    $display("   Going to Register        = %h", cpu_instant.wWBMux2WData);
    $display("   Register Written to      = %b", cpu_instant.wMWB2RegWrite);
end

initial begin
    clk = 0;
    forever #5 clk = ~clk;

end

initial begin

                reset = 0;
            #10;    reset = 1;
    #310    $finish;

end
endmodule
```

```
Chronologic VCS simulator copyright 1991-2018
Contains Synopsys proprietary information.
Compiler version O-2018.09-SP2-3_Full64; Runtime version O-2018.09-SP2-3_Full64;  Dec 1
0 02:43 2020
VCD+ Writer O-2018.09-SP2-3_Full64 Copyright (c) 1991-2018 by Synopsys Inc.
        PC Count                = xxxx
        Instruction     = xxxx


        ****** REGISTER ******
(Rd Reg 1 = xxxx) AND (Rd Reg 2 = xxxx)
             R15 = xxxx


          ****** ALU ******
   ALU Ctrl             = x
(ALU Input A) = xxxx (ALU Input B) = xxxx
   ALU Result           = xxxx


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = xxxx
   Op1 Register               = xxxx
   Branch Type (<, >, =)      = x
   Branch Flag                = x
   isBranch                   = x
   Current PC                 = xxxx
   New PC (PC+2+ShiftOp2)     = xxxx


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = xxxx
   Read from Memory?          = x
   Write to Memory?           = x
   Is Byte?                   = x
   InData                     = xxxx
   OutData                    = xxxx
          *** WB ***
   Mem or ALU to Reg?         = x
   Going to Register          = xxxx
   Register Written to        = x
        PC Count                = 0000
        Instruction     = xxxx


        ****** REGISTER ******
(Rd Reg 1 = 0000) AND (Rd Reg 2 = 0000)
             R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 0
(ALU Input A) = 0000 (ALU Input B) = 0000
   ALU Result           = 0000


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 0000
   Branch Type (<, >, =)      = 0
```

```
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 0000
   New PC (PC+2+ShiftOp2)     = 0000


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = 0000
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = 0000
   OutData                    = xxxx
           *** WB ***
   Mem or ALU to Reg?         = 0
   Going to Register          = 0000
   Register Written to        = 0
        PC Count                = 0002
        Instruction     = 12d1


        ****** REGISTER ******
(Rd Reg 1 = xxxx) AND (Rd Reg 2 = xxxx)
             R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 0
(ALU Input A) = 0e12 (ALU Input B) = 0e12
   ALU Result           = 0e12


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 0e12
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = x
   Current PC                 = 0002
   New PC (PC+2+ShiftOp2)     = 0000


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = 0000
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = 0000
   OutData                    = xxxx
           *** WB ***
   Mem or ALU to Reg?         = 0
   Going to Register          = xxxx
   Register Written to        = 0
        PC Count                = 0000
        Instruction     = 1120


        ****** REGISTER ******
(Rd Reg 1 = 0010) AND (Rd Reg 2 = 1101)
             R15 = 0000
```

```
          ****** ALU ******
   ALU Ctrl             = 0
(ALU Input A) = xxxx (ALU Input B) = xxxx
   ALU Result           = xxxx


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = xxxx
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = x
   Current PC                 = 0000
   New PC (PC+2+ShiftOp2)     = 0002


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = 0e12
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = 0e12
   OutData                    = xxxx
           *** WB ***
   Mem or ALU to Reg?         = 0
   Going to Register          = xxxx
   Register Written to        = 0
        PC Count                = 0000
        Instruction     = 1120


        ****** REGISTER ******
(Rd Reg 1 = 0001) AND (Rd Reg 2 = 0010)
              R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 5
(ALU Input A) = f08f (ALU Input B) = 0000
   ALU Result           = f08f


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = f08f
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 0000
   New PC (PC+2+ShiftOp2)     = 05a6


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = xxxx
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
```

```
   InData                     = xxxx
   OutData                    = xxxx
          *** WB ***
   Mem or ALU to Reg?         = 0
   Going to Register          = xxxx
   Register Written to        = 0
        PC Count                = 0002
        Instruction     = 12d1


        ****** REGISTER ******
(Rd Reg 1 = 0001) AND (Rd Reg 2 = 0010)
             R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 4
(ALU Input A) = 0045 (ALU Input B) = f08f
   ALU Result           = f0d4


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 0045
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 0002
   New PC (PC+2+ShiftOp2)     = 0242


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = f08f
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = f08f
   OutData                    = xxxx
          *** WB ***
   Mem or ALU to Reg?         = 0
   Going to Register          = xxxx
   Register Written to        = 0
        PC Count                = 0004
        Instruction     = 148e


        ****** REGISTER ******
(Rd Reg 1 = 0010) AND (Rd Reg 2 = 1101)
             R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 4
(ALU Input A) = f08f (ALU Input B) = f08f
   ALU Result           = e11e


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 0045
```

```
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 0004
   New PC (PC+2+ShiftOp2)     = 0242


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = f0d4
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = 0045
   OutData                    = xxxx
           *** WB ***
   Mem or ALU to Reg?         = 1
   Going to Register          = f08f
   Register Written to        = 1
        PC Count                = 0006
        Instruction     = a800


        ****** REGISTER ******
(Rd Reg 1 = 0100) AND (Rd Reg 2 = 1000)
             R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 5
(ALU Input A) = f08f (ALU Input B) = 0000
   ALU Result           = f08f


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = f08f
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 0006
   New PC (PC+2+ShiftOp2)     = 05a6


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = e11e
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = 0045
   OutData                    = xxxx
           *** WB ***
   Mem or ALU to Reg?         = 1
   Going to Register          = f0d4
   Register Written to        = 1
        PC Count                = 0008
        Instruction     = 146f


        ****** REGISTER ******
(Rd Reg 1 = 1000) AND (Rd Reg 2 = 0000)
```

```
              R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 6
(ALU Input A) = 0084 (ALU Input B) = 0000
   ALU Result           = 0000


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 0084
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 0008
   New PC (PC+2+ShiftOp2)     = 0922


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = f08f
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = f08f
   OutData                    = xxxx
           *** WB ***
   Mem or ALU to Reg?         = 1
   Going to Register          = e11e
   Register Written to        = 1
        PC Count                = 000a
        Instruction     = 4794


        ****** REGISTER ******
(Rd Reg 1 = 0100) AND (Rd Reg 2 = 0110)
              R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 3
(ALU Input A) = 0000 (ALU Input B) = 0000
   ALU Result           = 0000


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 0000
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 000a
   New PC (PC+2+ShiftOp2)     = 0008


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = 0000
   Read from Memory?          = 0
   Write to Memory?           = 0
```

```
   Is Byte?                  = 0
   InData                    = 0084
   OutData                   = xxxx
           *** WB ***
   Mem or ALU to Reg?        = 1
   Going to Register         = f08f
   Register Written to       = 1
        PC Count               = 000c
        Instruction     = 934c


        ****** REGISTER ******
(Rd Reg 1 = 0111) AND (Rd Reg 2 = 1001)
             R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 7
(ALU Input A) = 0084 (ALU Input B) = 00eb
   ALU Result           = 0000


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15               = 0000
   Op1 Register              = 0084
   Branch Type (<, >, =)     = 0
   Branch Flag               = 0
   isBranch                  = 0
   Current PC                = 000c
   New PC (PC+2+ShiftOp2)    = 08e8


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                = 0000
   Read from Memory?         = 0
   Write to Memory?          = 0
   Is Byte?                  = 0
   InData                    = 0000
   OutData                   = xxxx
           *** WB ***
   Mem or ALU to Reg?        = 1
   Going to Register         = 0000
   Register Written to       = 1
        PC Count               = 000e
        Instruction     = 1ee1


        ****** REGISTER ******
(Rd Reg 1 = 0011) AND (Rd Reg 2 = 0100)
             R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 0
(ALU Input A) = ff56 (ALU Input B) = 0000
   ALU Result           = 0000


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15               = 0000
```

```
   Op1 Register               = ff56
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 000e
   New PC (PC+2+ShiftOp2)     = 0f34


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = 0000
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = 0084
   OutData                    = xxxx
           *** WB ***
   Mem or ALU to Reg?         = 1
   Going to Register          = 0000
   Register Written to        = 1
        PC Count                = 0010
        Instruction     = 5796


        ****** REGISTER ******
(Rd Reg 1 = 1110) AND (Rd Reg 2 = 1110)
             R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 2
(ALU Input A) = f076 (ALU Input B) = 004c
   ALU Result           = 0044


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = f076
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 0010
   New PC (PC+2+ShiftOp2)     = 0006


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = 0000
   Read from Memory?          = 1
   Write to Memory?           = 0
   Is Byte?                   = 1
   InData                     = 0056
   OutData                    = 0000
           *** WB ***
   Mem or ALU to Reg?         = 1
   Going to Register          = 0000
   Register Written to        = 1
        PC Count                = 0012
        Instruction     = 6698


        ****** REGISTER ******
```

```
(Rd Reg 1 = 0111) AND (Rd Reg 2 = 1001)
              R15 = 0000


           ****** ALU ******
   ALU Ctrl             = 5
(ALU Input A) = 0000 (ALU Input B) = 0000
   ALU Result           = 0000


         ****** BRANCH_CONTROL ******
           *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 0000
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 0012
   New PC (PC+2+ShiftOp2)     = fdd2


         ****** LOAD/STORE ******
             *** EX ***
   ALU Result                 = 0044
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = f076
   OutData                    = 0000
           *** WB ***
   Mem or ALU to Reg?         = 0
   Going to Register          = 0000
   Register Written to        = 1
        PC Count                = 0014
        Instruction     = e704


         ****** REGISTER ******
(Rd Reg 1 = 0110) AND (Rd Reg 2 = 1001)
              R15 = 0000


           ****** ALU ******
   ALU Ctrl             = 0
(ALU Input A) = 0000 (ALU Input B) = 0796
   ALU Result           = 0796


         ****** BRANCH_CONTROL ******
           *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 0000
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 0014
   New PC (PC+2+ShiftOp2)     = 0f3e


         ****** LOAD/STORE ******
             *** EX ***
   ALU Result                 = 0000
   Read from Memory?          = 0
```

```
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = 0000
   OutData                    = 0000
           *** WB ***
   Mem or ALU to Reg?         = 1
   Going to Register          = 0044
   Register Written to        = 1
        PC Count                = 0016
        Instruction     = 1b10


        ****** REGISTER ******
(Rd Reg 1 = 0111) AND (Rd Reg 2 = 0000)
             R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 0
(ALU Input A) = 00eb (ALU Input B) = 0698
   ALU Result           = 0698


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 00eb
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 0016
   New PC (PC+2+ShiftOp2)     = 0d44


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = 0796
   Read from Memory?          = 0
   Write to Memory?           = 1
   Is Byte?                   = 1
   InData                     = 0000
   OutData                    = 0000
           *** WB ***
   Mem or ALU to Reg?         = 1
   Going to Register          = 0000
   Register Written to        = 1
        PC Count                = 0018
        Instruction     = c705


        ****** REGISTER ******
(Rd Reg 1 = 1011) AND (Rd Reg 2 = 0001)
             R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 1
(ALU Input A) = 0000 (ALU Input B) = 0e12
   ALU Result           = 0000


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
```

```
   Register 15                = 0000
   Op1 Register               = 0000
   Branch Type (<, >, =)      = 3
   Branch Flag                = 1
   isBranch                   = 1
   Current PC                 = 0018
   New PC (PC+2+ShiftOp2)     = 001e


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = 0698
   Read from Memory?          = 1
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = 00eb
   OutData                    = 0000
           *** WB ***
   Mem or ALU to Reg?         = 0
   Going to Register          = 0000
   Register Written to        = 0
        PC Count                = 001e
        Instruction     = 1110


        ****** REGISTER ******
(Rd Reg 1 = 0111) AND (Rd Reg 2 = 0000)
             R15 = 0000


           ****** ALU ******
   ALU Ctrl             = 4
(ALU Input A) = cc90 (ALU Input B) = e11e
   ALU Result           = adae


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = cc90
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 001e
   New PC (PC+2+ShiftOp2)     = f638


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = 0000
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = 0000
   OutData                    = 0000
           *** WB ***
   Mem or ALU to Reg?         = 0
   Going to Register          = 0000
   Register Written to        = 1
        PC Count                = 0020
        Instruction     = 1110
```

```
        ****** REGISTER ******
(Rd Reg 1 = 0001) AND (Rd Reg 2 = 0001)
             R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 1
(ALU Input A) = 0000 (ALU Input B) = 0e12
   ALU Result           = 0000


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 0000
   Branch Type (<, >, =)      = 1
   Branch Flag                = 0
   isBranch                   = 1
   Current PC                 = 0020
   New PC (PC+2+ShiftOp2)     = 0024


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = adae
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = cc90
   OutData                    = 0000
           *** WB ***
   Mem or ALU to Reg?         = 0
   Going to Register          = 0000
   Register Written to        = 0
        PC Count                = 0022
        Instruction     = 6890


        ****** REGISTER ******
(Rd Reg 1 = 0001) AND (Rd Reg 2 = 0001)
             R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 4
(ALU Input A) = e11e (ALU Input B) = e11e
   ALU Result           = c23c


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = e11e
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 0022
   New PC (PC+2+ShiftOp2)     = 0240


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = 0000
```

```
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = 0000
   OutData                    = 0000
          *** WB ***
   Mem or ALU to Reg?         = 1
   Going to Register          = adae
   Register Written to        = 1
        PC Count                = 0024
        Instruction     = 1880


        ****** REGISTER ******
(Rd Reg 1 = 1000) AND (Rd Reg 2 = 1001)
              R15 = 0000


          ****** ALU ******
   ALU Ctrl              = 4
(ALU Input A) = e11e (ALU Input B) = e11e
   ALU Result            = c23c


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = e11e
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 0024
   New PC (PC+2+ShiftOp2)     = 0242


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = c23c
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = e11e
   OutData                    = 0000
          *** WB ***
   Mem or ALU to Reg?         = 0
   Going to Register          = 0000
   Register Written to        = 0
        PC Count                = 0024
        Instruction     = 1880


        ****** REGISTER ******
(Rd Reg 1 = 1000) AND (Rd Reg 2 = 1000)
              R15 = 0000


          ****** ALU ******
   ALU Ctrl              = 0
(ALU Input A) = 0000 (ALU Input B) = f890
   ALU Result            = f890


        ****** BRANCH_CONTROL ******
```

```
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 0000
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 0024
   New PC (PC+2+ShiftOp2)     = f144


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = c23c
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = e11e
   OutData                    = 0000
           *** WB ***
   Mem or ALU to Reg?         = 1
   Going to Register          = c23c
   Register Written to        = 1
        PC Count                = 0026
        Instruction     = 7892


        ****** REGISTER ******
(Rd Reg 1 = 1000) AND (Rd Reg 2 = 1000)
              R15 = 0000


           ****** ALU ******
   ALU Ctrl             = 0
(ALU Input A) = 0000 (ALU Input B) = 0000
   ALU Result           = 0000


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 0000
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 0026
   New PC (PC+2+ShiftOp2)     = f126


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = f890
   Read from Memory?          = 1
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = 0000
   OutData                    = xxxx
           *** WB ***
   Mem or ALU to Reg?         = 1
   Going to Register          = c23c
   Register Written to        = 1
        PC Count                = 0028
        Instruction     = 6a92
```

```
        ****** REGISTER ******
(Rd Reg 1 = 1000) AND (Rd Reg 2 = 1001)
              R15 = 0000


           ****** ALU ******
   ALU Ctrl             = 4
(ALU Input A) = 0000 (ALU Input B) = 0000
   ALU Result           = 0000


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 0000
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 0028
   New PC (PC+2+ShiftOp2)     = f126


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = 0000
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = 0000
   OutData                    = xxxx
           *** WB ***
   Mem or ALU to Reg?         = 0
   Going to Register          = xxxx
   Register Written to        = 1
        PC Count                = 002a
        Instruction     = 1ca0


        ****** REGISTER ******
(Rd Reg 1 = 1010) AND (Rd Reg 2 = 1001)
              R15 = 0000


           ****** ALU ******
   ALU Ctrl             = 0
(ALU Input A) = xxxx (ALU Input B) = f892
   ALU Result           = f892


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = xxxx
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 002a
   New PC (PC+2+ShiftOp2)     = f14c


        ****** LOAD/STORE ******
            *** EX ***
```

```
   ALU Result                 = 0000
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = 0000
   OutData                    = xxxx
           *** WB ***
   Mem or ALU to Reg?         = 0
   Going to Register          = xxxx
   Register Written to        = 0
        PC Count                = 002a
        Instruction     = 1ca0


        ****** REGISTER ******
(Rd Reg 1 = 1100) AND (Rd Reg 2 = 1010)
             R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 0
(ALU Input A) = 0000 (ALU Input B) = fa92
   ALU Result           = fa92


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 0000
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 002a
   New PC (PC+2+ShiftOp2)     = f54e


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = f892
   Read from Memory?          = 0
   Write to Memory?           = 1
   Is Byte?                   = 0
   InData                     = xxxx
   OutData                    = xxxx
           *** WB ***
   Mem or ALU to Reg?         = 1
   Going to Register          = 0000
   Register Written to        = 1
        PC Count                = 002c
        Instruction     = 1cd1


        ****** REGISTER ******
(Rd Reg 1 = 1100) AND (Rd Reg 2 = 1010)
             R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 0
(ALU Input A) = 0002 (ALU Input B) = 0000
   ALU Result           = 0000
```

```
        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 0002
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 002c
   New PC (PC+2+ShiftOp2)     = f96c


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = fa92
   Read from Memory?          = 1
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = 0000
   OutData                    = xxxx
           *** WB ***
   Mem or ALU to Reg?         = 0
   Going to Register          = xxxx
   Register Written to        = 0
        PC Count                = 002e
        Instruction     = 1cd0


        ****** REGISTER ******
(Rd Reg 1 = 1100) AND (Rd Reg 2 = 1101)
             R15 = 0000


           ****** ALU ******
   ALU Ctrl             = 4
(ALU Input A) = 0002 (ALU Input B) = 0000
   ALU Result           = 0002


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 0002
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 002e
   New PC (PC+2+ShiftOp2)     = f96c


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = 0000
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = 0002
   OutData                    = xxxx
           *** WB ***
   Mem or ALU to Reg?         = 0
   Going to Register          = xxxx
   Register Written to        = 1
        PC Count                = 0030
        Instruction     = ef20
```

```
        ****** REGISTER ******
(Rd Reg 1 = 1100) AND (Rd Reg 2 = 1101)
              R15 = 0000


           ****** ALU ******
   ALU Ctrl             = 5
(ALU Input A) = 0002 (ALU Input B) = 0000
   ALU Result           = 0002


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 0002
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 0030
   New PC (PC+2+ShiftOp2)     = f9d0


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = 0002
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = 0002
   OutData                    = xxxx
           *** WB ***
   Mem or ALU to Reg?         = 0
   Going to Register          = xxxx
   Register Written to        = 0
        PC Count                = 0032
        Instruction     = 0000


        ****** REGISTER ******
(Rd Reg 1 = 1111) AND (Rd Reg 2 = 0010)
              R15 = 0000


           ****** ALU ******
   ALU Ctrl             = 4
(ALU Input A) = 0002 (ALU Input B) = 0000
   ALU Result           = 0002


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 0002
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 0
   Current PC                 = 0032
   New PC (PC+2+ShiftOp2)     = f9d0


        ****** LOAD/STORE ******
```

```
           *** EX ***
   ALU Result                = 0002
   Read from Memory?         = 0
   Write to Memory?          = 0
   Is Byte?                  = 0
   InData                    = 0002
   OutData                   = xxxx
          *** WB ***
   Mem or ALU to Reg?        = 1
   Going to Register         = 0002
   Register Written to       = 1
        PC Count               = 0034
        Instruction     = 0000


        ****** REGISTER ******
(Rd Reg 1 = 0000) AND (Rd Reg 2 = 0000)
             R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 1
(ALU Input A) = 0000 (ALU Input B) = f08f
   ALU Result           = 0000


        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15               = 0000
   Op1 Register              = 0000
   Branch Type (<, >, =)     = 3
   Branch Flag               = 1
   isBranch                  = 1
   Current PC                = 0034
   New PC (PC+2+ShiftOp2)    = 0032


        ****** LOAD/STORE ******
           *** EX ***
   ALU Result                = 0002
   Read from Memory?         = 0
   Write to Memory?          = 0
   Is Byte?                  = 0
   InData                    = 0002
   OutData                   = xxxx
          *** WB ***
   Mem or ALU to Reg?        = 1
   Going to Register         = 0002
   Register Written to       = 1
        PC Count               = 0032
        Instruction     = 0000


        ****** REGISTER ******
(Rd Reg 1 = 0000) AND (Rd Reg 2 = 0000)
             R15 = 0000


          ****** ALU ******
   ALU Ctrl             = 0
(ALU Input A) = 0e12 (ALU Input B) = 0e12
   ALU Result           = 0e12
```

```
        ****** BRANCH_CONTROL ******
          *** COMPARATOR(EX) ***
   Register 15                = 0000
   Op1 Register               = 0e12
   Branch Type (<, >, =)      = 0
   Branch Flag                = 0
   isBranch                   = 1
   Current PC                 = 0032
   New PC (PC+2+ShiftOp2)     = 0034


        ****** LOAD/STORE ******
            *** EX ***
   ALU Result                 = 0000
   Read from Memory?          = 0
   Write to Memory?           = 0
   Is Byte?                   = 0
   InData                     = 0000
   OutData                    = xxxx
           *** WB ***
   Mem or ALU to Reg?         = 1
   Going to Register          = 0002
   Register Written to        = 1
$finish called from file "cpu_fixture.v", line 60.
$finish at simulation time                  320
          V C S   S i m u l a t i o n   R e p o r t
Time: 320
CPU Time:      0.320 seconds;       Data structure size:   0.0Mb
Thu Dec 10 02:43:22 2020
```