CPE 142
Fall 2019
Term Project: Phase II
Minh Nguyen
Kyle Zenarosa
Team 14

Percentages of Project Contribution:
Minh Nguyen: 50%
Kyle Zenarosa: 50%

CPE 142 Term Project Phase 2
Table of contents

Table of Contents

CSc/CPE 142

Term Project Status Report

Name	*% Contribution*	*Grade*
Minh Nguyen	*50*	
Kyle Zenarosa	*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
Signed addition	Yes
Signed subtraction	Yes
Signed multiplication	Yes
Signed division	Yes
AND immediate	Yes

Instructions	Was this instruction fully fuctional as verified by the assembly program provided? If no, explain
OR immediate	Yes
Load byte unsigned	Yes
Store byte	Yes
Load	Yes
Store	Yes
Branch on less than	Yes
Branch on greater than	Yes but it did not flush the next instruction due to timing issue. The signal to flush is arriving at buffer before the next instruction is pulled. We ran out of time to implement a fix for this issue.
Branch on equal	Yes
jump	Yes
halt	Yes
Signed addition	Yes
Signed subtraction	Yes

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	Minh	Yes	
ALU control unit	Yes	Minh	Yes	

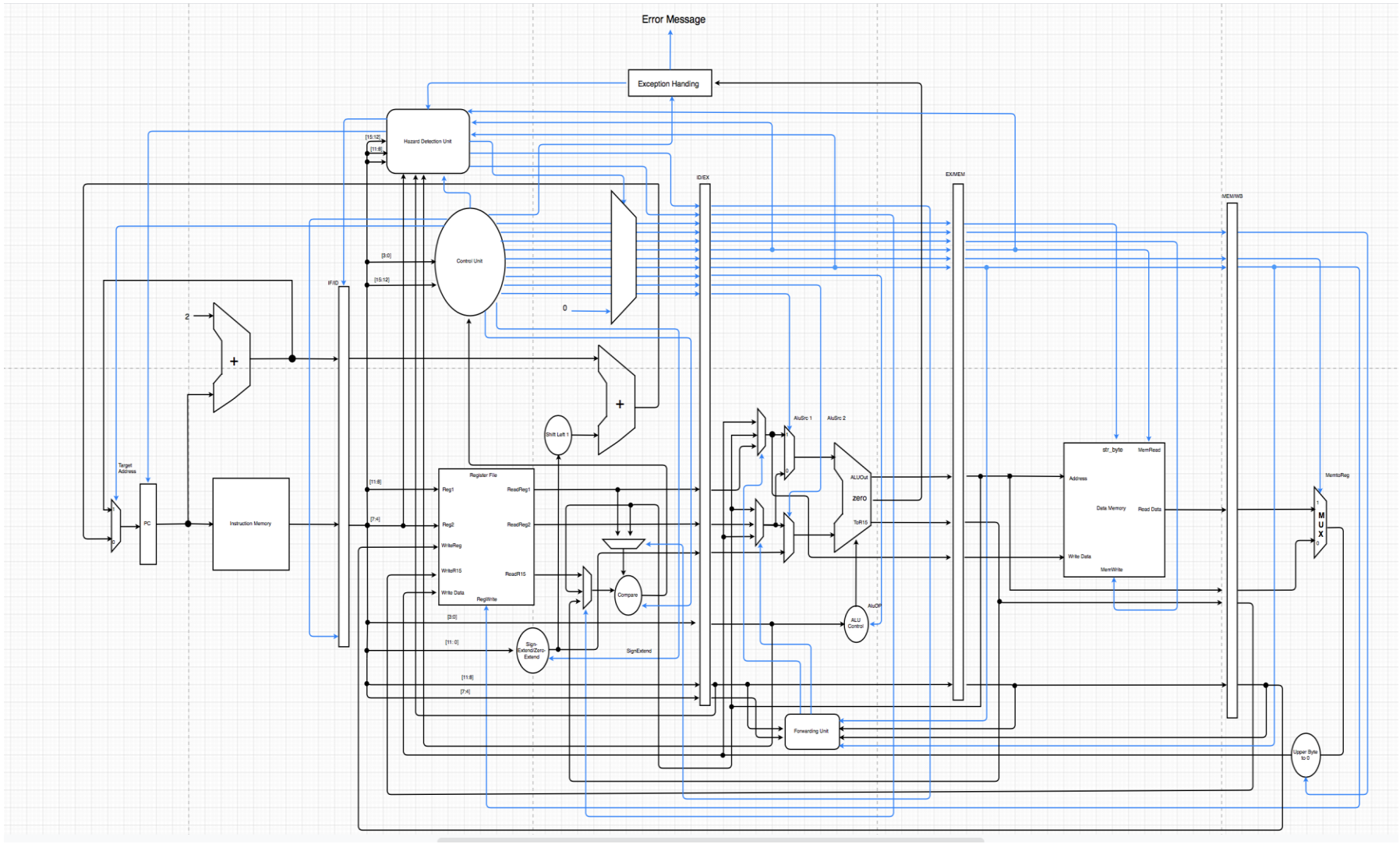
Individual Components	Does your system have this component?	List the student who designed and verified the block	Does it work ?	List problems with the component, if any.
Data Memory Unit	Yes	Kyle	Yes	
Register File	Yes	Kyle	Yes	
PC	Yes	Kyle	Yes	
Instruction Memory	Yes	Kyle	Yes	
Other registers: Pipeline buffers	Yes	Kyle	Yes	
Multiplexors	Yes	Minh & Kyle	Yes	
exception handler 1. Unknown opcode 2. Arith. Overflow …..	Yes	Minh	Yes	
Control Units 1. main 2. forwarding 3. lw hazard detection	Yes	Minh / Kyle	Yes	The main control unit does not flush the IF/ID buffer properly when a branch is taken.

How many stages do you have in your pipeline? …….. 5 stages.

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

When a branch is taken, ideally the main control unit should flush the IF/ID buffer so that the instruction immediately following the branch will not go through the pipeline. Our datapath fails to do that. The proper target address and instruction is reached on a taken branch, but the instruction that should have been flushed goes through the pipeline anyway.

Datapath Diagram



Truth Tables

Instruction	OpCode	FunctionCod	RegWrite	ALUSrc1	ALUSrc2	ALUOp	SignExtend	MemRead	MemWrite	MemtoReg	UpperByteToZero	TargetAddress	IF/ID.Flush	DiscardUpperByte	Comparison	Exception	Branch
SignedAdd	0000	0000	1	1	1	00	00	0	0	0	0	0	1	0	x	0	0
SignedSub	0000	0001	1	1	1	00	00	0	0	0	0	0	1	0	x	0	0
SignedMult	0000	0100	1	1	1	00	00	0	0	0	0	0	1	0	x	0	0
SignedDiv	0000	0101	1	1	1	00	00	0	0	0	0	0	1	0	x	0	0
and	0001	xxxx	1	1	0	10	10	0	0	0	0	0	1	0	x	0	0
or	0010	xxxx	1	1	0	01	10	0	0	0	0	0	1	0	x	0	0
lbu	1010	xxxx	1	0	0	01	01	1	0	1	1	0	1	0	x	0	0
sb	1011	xxxx	0	0	0	01	01	0	1	0	0	0	1	1	x	0	0
lw	1100	xxxx	1	0	0	01	01	1	0	1	0	0	1	0	x	0	0
sw	1101	xxxx	0	0	0	01	01	0	1	0	0	0	1	0	x	0	0
blt	0101	xxxx	1	0	0	xx	10	0	0	0	0	1	0	0	00	0	1
bgt	0100	xxxx	1	0	0	xx	10	0	0	0	0	1	0	0	01	0	1
beq	0110	xxxx	1	0	0	xx	10	0	0	0	0	1	0	0	10	0	1
jump	0111	xxxx	1	0	0	xx	11	0	0	0	0	1	0	0	x	0	0
halt	1111	xxxx	0	0	0	xx	xx	0	0	0	0	0	0	0	x	0	0

Figure 1 Control Unit Truth Table

Instruction	ALUOp	FunctionCod	ALU Action	ALU Input
SignedAdd	00	0000	add	000
SignedSub	00	0001	subtract	001
SignedMult	00	0100	multiply	010
SignedDiv	00	0101	divide	011
And Immed	10	xxxx	and	100
Or Immed	11	xxxx	or	101
lw	01	xxxx	add	000
sw	01	xxxx	add	000
lbu	01	xxxx	add	000
sb	01	xxxx	add	000

Figure 2 ALU Control Truth Table.

Specification of Datapath Components

Sign extension component: All immediate values will go through the sign extension module to be extended to 16-bits. However, the offset sizes differ between instructions. Rather than implementing three different sign extension modules, we used one sign extension that takes in the maximum offset size of 12 bits. A control signal to the module will determine whether to extend the input value from the lower 4 bits, 8 bits, or the full 12 bits. The main control unit will look at the instruction opcode and determine the proper sign extension amount. It will send 01 to extend a 4-bit offset, 10 to extend an 8-bit offset, and 11 for a 12-bit offset.

Hazard Detection Unit: Our hazard detection unit was designed to handle both load word hazards, and hazards that may arise due to branch instructions. The load word hazard detection logic is straightforward: we compare the IF/ID.RegisterOp1 and the IF/ID.Register Op2 with the ID/EX.RegisterOp1 and see if there is a dependency. If there is one and the ID/EX.MemRead signal is high, the hazard detection unit sends a signal to the control unit telling it to insert a stall in the pipeline. The hazard detection unit also sends signals to the PC and the IF/ID buffer to prevent any changes from occurring to them. This delays the ID stage by one clock cycle.

The hazard detection unit was also designed to handle hazards caused by dependencies between a branch instruction and a preceding instruction that wrote into the Op1 register, or both Op1 and R15. In the case of multiplication and division instructions, which wrote back to their Op1 destination registers and register R15, a stall would be inserted into the pipeline by the main control unit. This is to ensure the needed data from the Op1 register and R15 would be ready for forwarding. The hazard detection unit would then control the muxes that determined what values would be written back to the register file (either the ALU values from the WB stage, or directly from the EX/MEM buffer). This would result in a single delay in the pipeline. Other instructions that wrote back to their Op1 register but not R15 would be handled in a similar fashion. The hazard detection unit determines what values to forward by checking the function code from the ID/EX buffer.

If a load word preceded a branch instruction, and there is a dependency between them, then the hazard detection unit would signal the main control unit to insert two bubbles in the pipeline. This would ensure that the data loaded from memory would be written back in time for the branch's ID stage.

Finally, the hazard detection unit is also designed to deal with exceptions. If the main control unit receives an invalid opcode, or if the ALU zero bit is set due to an arithmetic overflow, our exception module will send a signal to the hazard detection unit. Said unit

will then signal the main control unit to halt the execution of the program. Since we were required to display a message indicating an exception and not stop the rest of the program from executing, the hazard detection unit was not designed to implement the halting of the pipeline.

Register Forwarding Unit: For our forwarding unit, the only operand that might cause a dependency is the Op1 register. We compare the incoming Op1 register operand from the ID/EX buffer with the Op1 register operand from the EX/MEM buffer (or the MEM/WB buffer) and see if they match. If they match, and EX/MEM.RegWrite or MEM/WB.RegWrite is asserted, then the values are forwarded accordingly from the EX/MEM buffer or the MEM/WB buffer.

Branching Logic Within Control Unit: Our branch operation logic is done within our main control unit. The control unit receives a two bits wide signal from a comparator that is constantly comparing values of OP1 and R15. We assigned 2'b01 to less than operation, 2'b10 to greater than operation else 2'b11 equal. When a branch instruction opcode is received, the control unit will then compare that branch instruction to the know state input from comparator. If the branch operation matches to the output of the comparator, a signal is then sent to the hazard detection unit where a signal is then sent to the mux controlling new target address, and a signal is also sent to the buffer to flush the following instruction.

Logical Expressions for Control Units

Register Forwarding Unit:

```
//OP1 at EX/MEM = OP1 at ID/EX
if (regWriteMEM && (registerOP1EX != 0) && (registerOP1EX == registerOP1ID))
   forwardA = 2'b01;
//OP1 at MEM/WB = OP1 at ID/EX
else if (regWriteMEM && (registerOP1EX != 0) && (registerOP1MEM == registerOP1ID))
   forwardA = 2'b00;
else
   forwardA = 2'b10;

//MEM stage
if (regWriteMEM && (registerOP1EX != 0) && (registerOP1EX == registerOP2ID))
   forwardB = 2'b10;
else if (regWriteMEM && (registerOP1EX != 0) && (registerOP1MEM == registerOP2ID))
   forwardB = 2'b10;
else
   forwardB = 2'b01;
```

Hazard Detection Unit:

```
if(Opcode == 4'b0101 || Opcode == 4'b0100 || Opcode == 4'b0110) begin

        if((IF_ID_Op1 == ID_EX_Op1 || ID_EX_Op1 == 4'b1111) && ID_EX_MemRead) begin

            Delay = 1;
            Op1_Sel = 0;
            R15_Sel = 2'b00;
            IF_ID_Hold = 0;
```

```
            PC_Hold = 1;

        end

        else if(!ID_EX_MemRead && EX_MEM_MemRead) begin

            Delay = 1;
            Op1_Sel = 0;
            R15_Sel = 2'b00;
            IF_ID_Hold = 0;
            PC_Hold = 1;

        end

        else if(IF_ID_Op1 == ID_EX_Op1 && RegWrite != 2'b00 && IF_ID_Op1 != 4'b1111 && ID_EX_Op1 !=
4'b1111 && (FunctionCode == 4'b0000 || FunctionCode == 4'b0001)) begin

            Delay = 1;
            Op1_Sel = 1;
            R15_Sel = 2'b00;
            IF_ID_Hold = 0;
            PC_Hold = 1;

        end

        else if(RegWrite != 2'b00 && IF_ID_Op1 == 4'b1111 && ID_EX_Op1 == 4'b1111) begin

            Delay = 1;
            Op1_Sel = 1;
            R15_Sel = 2'b10;
            IF_ID_Hold = 0;
            PC_Hold = 1;
```

```
                end

                else if(RegWrite != 2'b00 && ID_EX_Op1 == 4'b1111 && IF_ID_Op1 != 4'b1111 && (FunctionCode ==
4'b0000 || FunctionCode == 4'b0001)) begin

                        Delay = 1;
                        Op1_Sel = 0;
                        R15_Sel = 2'b01;
                        IF_ID_Hold = 0;
                        PC_Hold = 1;

                end

                else if(RegWrite != 2'b00 && ID_EX_Op1 == IF_ID_Op1 && ID_EX_Op1 != 4'b1111 && IF_ID_Op1 !=
4'b1111 && (FunctionCode == 4'b0100 || FunctionCode == 4'b0101)) begin

                        Delay = 1;
                        Op1_Sel = 1;
                        R15_Sel = 2'b01;
                        IF_ID_Hold = 0;
                        PC_Hold = 1;

                end

                else if(RegWrite != 2'b00 && ID_EX_Op1 != IF_ID_Op1 && (FunctionCode == 4'b0100 || FunctionCode ==
4'b0101)) begin

                        Delay = 1;
                        Op1_Sel = 0;
                        R15_Sel = 2'b01;
                        IF_ID_Hold = 0;
```

```
                PC_Hold = 1;

        end


        else if(branch == 1'b1) begin

                IF_ID_Hold = 1;
                PC_Hold = 0;
                Delay = 1;
                Op1_Sel = 0;
                R15_Sel = 2'b00;

        end


        else begin

                Delay = 0;
                Op1_Sel = 0;
                R15_Sel = 2'b00;
                PC_Hold = 0;
                IF_ID_Hold = 1;

        end

end

else if(Opcode == 4'b1111) begin

                IF_ID_Hold = 0;
                PC_Hold = 1;
```

```
                Delay = 1;
                Op1_Sel = 0;
                R15_Sel = 2'b00;

    end

    else if(((IF_ID_Op1 == ID_EX_Op1) || (IF_ID_Op2 == ID_EX_Op1)) && ID_EX_MemRead) begin

                IF_ID_Hold = 0;
                PC_Hold = 1;
                Delay = 1;
                Op1_Sel = 0;
                R15_Sel = 2'b00;

end

    else begin

        IF_ID_Hold = 1;
        PC_Hold = 0;
        Delay = 0;
        Op1_Sel = 0;
        R15_Sel = 2'b00;
```

ALU_Control Code

```
module ALUControl(ALUOp, Function, ALUControl);
  input [1:0] ALUOp;  //Control Signal
  input [3:0] Function;
  wire [5:0] ALUControlCase;
  assign ALUControlCase = {ALUOp, Function};
  output reg [2:0] ALUControl;

  always @(*)
    casex (ALUControlCase)
     6'b000000: ALUControl=3'b000; //add
     6'b000001: ALUControl=3'b001; //sub
     6'b000100: ALUControl=3'b010; //mult
     6'b110101: ALUControl=3'b011; //div
     6'b10xxxx: ALUControl=3'b100; //and
     6'b01xxxx: ALUControl=3'b101; //or
     default: ALUControl=3'b000;
  endcase
endmodule


`include "ALUControl.v"
module ALUControl_fixture;

        reg [1:0] ALUOp;
        reg [3:0] Function;
        wire [2:0] ALUControl;

initial
```

```
        $vcdpluson;
initial
        $monitor ($time, " ALUOp = %b Function = %b ALUControl = %b " , ALUOp, Function, ALUControl);
localparam period = 25;

ALUControl a1(.ALUOp(ALUOp), .Function(Function), .ALUControl(ALUControl));

initial
        begin
        ALUOp = 2'b00;
        Function = 4'b0000;
        #period;
        ALUOp = 2'b00;
        Function = 4'b0001;
        #period;
        ALUOp = 2'b00;
        Function = 4'b0100;
        #period;
        ALUOp = 2'b00;
        Function = 4'b0101;
        #period;
        ALUOp = 2'b10;
        Function = 4'bxxxx;
        #period;
   ALUOp = 2'b01;
        Function = 4'bxxxx;
        #period;
        $finish;
        end
endmodule
```

Comparator Source Code

```
module comparator (
   input [15:0] OP1,
   input [15:0] R15,
   output reg [1:0]  branch
   );

   always @(*)
       if (OP1 < R15)
          branch = 2'b01;
       else if (OP1 > R15)
          branch = 2'b10;
           else
              branch = 2'b11;
endmodule

`include "comparator.v"
module main_alu_fixture;

  reg [15:0] OP1, R15;
  wire [1:0] branch;

initial
      $vcdpluson;
initial
          $monitor ($time, " OP1 = %b R15 = %b branch = %b \n ", OP1, R15, branch);

comparator a1(OP1, R15, branch);
initial
  begin
     OP1 = 16'b0000000000000101;
```

```
    R15 = 16'b0000000000000111;
    #100;
    OP1 = 16'b0000000000000111;
    R15 = 16'b0000000000000101;
    #100;
    OP1 = 16'b0000000000000111;
    R15 = 16'b0000000000000111;
    #100;
    OP1 = 16'b0000000000000000;
    R15 = 16'b0000000000000000;
    #100;
  end
endmodule
```

Main Control Unit Source Code

```
module Main_Control_Unit(
        input [3:0] OPCode, functionCode,
            input [1:0] ComparatorInput,
        output reg ALUSrc1, ALUSrc2, MemRead, MemWrite, MemtoReg, UpperByteToZero, TargetAddress, IFIDFlush, str_Byte,
Exception, branch,
        output reg [1:0] ALUOp, Comparison, RegWrite, SignExtend
   );

          always @(*) begin


                    casex (OPCode)
                      //R type instructions
                     4'b0000 : begin

                        if(functionCode == 4'b0100 || functionCode == 4'b0101) begin
                              RegWrite = 2'b10;
                              ALUSrc1 = 1'b1;
                              ALUSrc2 = 1'b1;
                              ALUOp = 2'b00;
                              SignExtend = 2'b00;
                              MemRead = 1'b0;
                              MemWrite = 1'b0;
                              MemtoReg = 1'b0;
                              UpperByteToZero = 1'b0;
                              TargetAddress = 1'b1;
                              IFIDFlush = 1'b1;
                              str_Byte = 1'b0;
                              Comparison = 2'bxx;
```

```
                    Exception = 1'b0;
                    branch = 1'b0;

                end

                else begin

                    RegWrite = 2'b01;
                    ALUSrc1 = 1'b1;
                    ALUSrc2 = 1'b1;
                    ALUOp = 2'b00;
                    SignExtend = 2'b00;
                    MemRead = 1'b0;
                    MemWrite = 1'b0;
                    MemtoReg = 1'b0;
                    UpperByteToZero = 1'b0;
                    TargetAddress = 1'b1;
                    IFIDFlush = 1'b1;
                    str_Byte = 1'b0;
                    Comparison = 2'bxx;
                    Exception = 1'b0;
                    branch = 1'b0;

                end
            end
//AND instruction*/
4'b0001 : begin
  RegWrite = 2'b01;
  ALUSrc1 = 1'b1;
  ALUSrc2 = 1'b0;
  ALUOp = 2'b10;
  SignExtend = 2'b11;
```

```
  MemRead = 1'b0;
  MemWrite =  1'b0;
  MemtoReg = 1'b0;
  UpperByteToZero = 1'b0;
  TargetAddress = 1'b1;
  IFIDFlush = 1'b1;
  str_Byte = 1'b0;
  Comparison = 2'bxx;
  Exception = 1'b0;
  branch = 1'b0;

end
//OR
4'b0010 : begin
  RegWrite = 2'b01;
  ALUSrc1 = 1'b1;
  ALUSrc2 = 1'b0;
  ALUOp = 2'b11;
  SignExtend = 2'b11;
  MemRead = 1'b0;
  MemWrite =  1'b0;
  MemtoReg = 1'b0;
  UpperByteToZero = 1'b0;
  TargetAddress = 1'b1;
  IFIDFlush = 1'b1;
  str_Byte = 1'b0;
  Comparison = 2'bxx;
  Exception = 1'b0;
        branch = 1'b0;

end
//LBU
```

```
4'b1010 : begin
  RegWrite = 2'b01;
  ALUSrc1 = 1'b0;
  ALUSrc2 = 1'b0;
  ALUOp = 2'b01;
  SignExtend = 2'b10;
  MemRead = 1'b1;
  MemWrite =  1'b0;
  MemtoReg = 1'b1;
  UpperByteToZero = 1'b1;
  TargetAddress = 1'b1;
  IFIDFlush = 1'b1;
  str_Byte = 1'b0;
  Comparison = 2'bxx;
  Exception = 1'b0;
        branch = 1'b0;

end
//SB
4'b1011 : begin
  RegWrite = 2'b00;
  ALUSrc1 = 1'b0;
  ALUSrc2 = 1'b0;
  ALUOp = 2'b01;
  SignExtend = 2'b10;
  MemRead = 1'b0;
  MemWrite =  1'b1;
  MemtoReg = 1'b0;
  UpperByteToZero = 1'b0;
  TargetAddress = 1'b1;
  IFIDFlush = 1'b1;
  str_Byte = 1'b1;
```

```
  Comparison = 2'bxx;
  Exception = 1'b0;
        branch = 1'b0;

end
//LW
4'b1100 : begin
  RegWrite = 2'b01;
  ALUSrc1 = 1'b0;
  ALUSrc2 = 1'b0;
  ALUOp = 2'b01;
  SignExtend = 2'b10;
  MemRead = 1'b1;
  MemWrite =  1'b0;
  MemtoReg = 1'b1;
  UpperByteToZero = 1'b0;
  TargetAddress = 1'b1;
  IFIDFlush = 1'b1;
  str_Byte = 1'b0;
  Comparison = 2'bxx;
  Exception = 1'b0;
        branch = 1'b0;

end
//SW
4'b1101 : begin
  RegWrite = 2'b00;
  ALUSrc1 = 1'b0;
  ALUSrc2 = 1'b0;
  ALUOp = 2'b01;
  SignExtend = 2'b10;
  MemRead = 1'b0;
```

```
  MemWrite =  1'b1;
  MemtoReg = 1'b0;
  UpperByteToZero = 1'b0;
  TargetAddress = 1'b1;
  IFIDFlush = 1'b1;
  str_Byte = 1'b0;
  Comparison = 2'bxx;
  Exception = 1'b0;
        branch = 1'b0;

end
//BLT
4'b0101 : begin
  if (ComparatorInput == 2'b01) begin
                  RegWrite = 2'b00;
                  ALUSrc1 = 1'b0;
                  ALUSrc2 = 1'b0;
                  ALUOp = 2'bxx;
                  SignExtend = 2'b01;
                  MemRead = 1'b0;
                  MemWrite =  1'b0;
                  MemtoReg = 1'b0;
                  UpperByteToZero = 1'b0;
                  TargetAddress = 1'b0;
                  IFIDFlush = 1'b0;
                  str_Byte = 1'b0;
                  Comparison = 2'b00;
                  Exception = 1'b0;
            branch = 1'b1;

  end
        else begin
```

```
                RegWrite = 2'b00;
                ALUSrc1 = 1'b0;
                ALUSrc2 = 1'b0;
                ALUOp = 2'bxx;
                SignExtend = 2'b00;
                MemRead = 1'b0;
                MemWrite = 1'b0;
                MemtoReg = 1'b0;
                UpperByteToZero = 1'b0;
                TargetAddress = 1'b1;
                IFIDFlush = 1'b1;
                str_Byte = 1'b0;
                Comparison = 2'b00;
                Exception = 1'b0;
                branch = 1'b0;
        end
end
//BGT
4'b0100 : begin
  if (ComparatorInput == 2'b10) begin
    RegWrite = 2'b00;
    ALUSrc1 = 1'b0;
    ALUSrc2 = 1'b0;
    ALUOp = 2'bxx;
    SignExtend = 2'b01;
    MemRead = 1'b0;
    MemWrite = 1'b0;
    MemtoReg = 1'b0;
    UpperByteToZero = 1'b0;
    TargetAddress = 1'b0;
    IFIDFlush = 1'b0;
    str_Byte = 1'b0;
```

```
    Comparison = 2'b01;
    Exception = 1'b0;
    branch = 1'b1;
  end
        else begin
                RegWrite = 2'b00;
                ALUSrc1 = 1'b0;
                ALUSrc2 = 1'b0;
                ALUOp = 2'bxx;
                SignExtend = 2'b00;
                MemRead = 1'b0;
                MemWrite =  1'b0;
                MemtoReg = 1'b0;
                UpperByteToZero = 1'b0;
                TargetAddress = 1'b1;
                IFIDFlush = 1'b1;
                str_Byte = 1'b0;
                Comparison = 2'b00;
                Exception = 1'b0;
                branch = 1'b0;
        end
end
//BEQ
4'b0110 : begin
  if (ComparatorInput == 2'b11) begin
  RegWrite = 2'b00;
  ALUSrc1 = 1'b0;
  ALUSrc2 = 1'b0;
  ALUOp = 2'bxx;
  SignExtend = 2'b01;
  MemRead = 1'b0;
  MemWrite =  1'b0;
```

```
  MemtoReg = 1'b0;
  UpperByteToZero = 1'b0;
  TargetAddress = 1'b0;
  IFIDFlush = 1'b0;
  str_Byte = 1'b0;
  Comparison = 2'b10;
  Exception = 1'b0;
  branch = 1'b1;
  end
      else begin
          RegWrite = 2'b00;
          ALUSrc1 = 1'b0;
          ALUSrc2 = 1'b0;
          ALUOp = 2'bxx;
          SignExtend = 2'b00;
          MemRead = 1'b0;
          MemWrite = 1'b0;
          MemtoReg = 1'b0;
          UpperByteToZero = 1'b0;
          TargetAddress = 1'b1;
          IFIDFlush = 1'b1;
          str_Byte = 1'b0;
          Comparison = 2'b00;
          Exception = 1'b0;
          branch = 1'b0;
      end
end
//JUMP
 4'b0111 : begin
  RegWrite = 2'b00;
  ALUSrc1 = 1'b0;
  ALUSrc2 = 1'b0;
```

```
  ALUOp = 2'bxx;
  SignExtend = 2'b00;
  MemRead = 1'b0;
  MemWrite = 1'b0;
  MemtoReg = 1'b0;
  UpperByteToZero = 1'b0;
  TargetAddress = 1'b0;
  IFIDFlush = 1'b0;
  str_Byte = 1'b0;
  Comparison = 2'bxx;
  Exception = 1'b0;
  branch = 1'b1;
 end
 //HALT
 4'b1111 : begin
  RegWrite = 2'b00;
  ALUSrc1 = 1'b0;
  ALUSrc2 = 1'b0;
  ALUOp = 2'bxx;
  SignExtend = 2'bxx;
  MemRead = 1'b0;
  MemWrite = 1'b0;
  MemtoReg = 1'b0;
  UpperByteToZero = 1'b0;
  TargetAddress = 1'b0;
  IFIDFlush = 1'b0;
  str_Byte = 1'b0;
  Comparison = 2'bxx;
  Exception = 1'b0;
  branch = 1'b0;
 end
 default : begin
```

```
    RegWrite = 2'b00;
    ALUSrc1 = 1'b0;
    ALUSrc2 = 1'b0;
    ALUOp = 2'b00;
    SignExtend = 2'b00;
    MemRead = 1'b0;
    MemWrite =  1'b0;
    MemtoReg = 1'b0;
    UpperByteToZero = 1'b0;
    TargetAddress = 1'b1;
    IFIDFlush = 1'b1;
    str_Byte = 1'b0;
    Comparison = 2'bxx;
    Exception = 1'b0;
    branch = 1'b0;
   end

endcase
end

endmodule



`include "Main_Control_Unit.v"


module Main_Control_fixture;
reg         Delay;
reg [3:0]  OPCode, functionCode;
reg [1:0]  ComparatorInput;
```

```
wire      ALUSrc1,
                ALUSrc2,
                MemRead,
                MemWrite,
                MemtoReg,
                UpperByteToZero,
                TargetAddress,
                IFIDFlush,
                str_Byte,
                Exception;

wire[1:0]  ALUOp,
                Comparison,
                RegWrite,
                SignExtend;

initial
      $vcdpluson;
initial
        $monitor ($time , " OPCode = %b functionCOde = %b ComparatorInput = %b ALUSrc1 = %b ALUSrc2 = %b MemRead =
%b MemWrite %b MemtoReg %b UpperByteToZero %b TargetAddress = %b IFIDFlush = %b str_Byte = %b Exception = %b
ALUOp = %b Comparison = %b RegWrite = %b SignExtend = %b " , OPCode, functionCode, ComparatorInput, ALUSrc1,
ALUSrc2, MemRead, MemWrite, MemtoReg, UpperByteToZero, TargetAddress, IFIDFlush, str_Byte, Exception, ALUOp,
Comparison, RegWrite, SignExtend);
localparam period = 25;

Main_Control_Unit c1(.OPCode(OPCode),
                .functionCode(functionCode),
                .ComparatorInput(ComparatorInput),
                .ALUSrc1(ALUSrc1),
            .ALUSrc2(ALUSrc2),
                .MemRead(MemRead),
```

```
        .MemWrite(MemWrite),
        .MemtoReg(MemtoReg),
        .UpperByteToZero(UpperByteToZero),
        .TargetAddress(TargetAddress),
        .IFIDFlush(IFIDFlush),
        .str_Byte(str_Byte),
        .Exception(Exception),
        .ALUOp(ALUOp),
            .Comparison(Comparison),
            .RegWrite(RegWrite),
        .SignExtend(SignExtend));


initial
    begin
            functionCode = 4'b0101;
              ComparatorInput = 2'b11;
            OPCode = 4'b0110;
              Delay = 1'b0;
         #period;
              functionCode = 4'b0101;
              ComparatorInput = 2'b10;
         OPCode = 4'b0101;
              Delay = 1'b0;
        #period;
              functionCode = 4'b0101;
              ComparatorInput = 2'b11;
         OPCode = 4'b0101;
              Delay = 1'b0;
        #period;
```

```
        functionCode = 4'b0100;
        ComparatorInput = 2'b01;
    OPCode = 4'b0100;
        Delay = 1'b0;
    #period;
        functionCode = 4'b0100;
        ComparatorInput = 2'b10;
    OPCode = 4'b0100;
        Delay = 1'b0;
    #period;
        functionCode = 4'b0100;
        ComparatorInput = 2'b11;
    OPCode = 4'b0100;
        Delay = 1'b0;
   #period;

      functionCode = 4'b0110;
        ComparatorInput = 2'b01;
    OPCode = 4'b0110;
        Delay = 1'b0;
    #period;
        functionCode = 4'b0110;
        ComparatorInput = 2'b10;
    OPCode = 4'b0110;
        Delay = 1'b0;
    #period;
        functionCode = 4'b0110;
        ComparatorInput = 2'b11;
    OPCode = 4'b0110;
        Delay = 1'b0;
    #period;
//
```

```
    functionCode = 4'b0100;
       ComparatorInput = 2'b10;
    OPCode = 4'b0000;
       Delay = 1'b1;
    #period;
       functionCode = 4'b0100;
          ComparatorInput = 2'b10;
          OPCode = 4'b0000;
          Delay = 1'b0;
          #period;
          functionCode = 4'b0100;
          ComparatorInput = 2'b10;
          OPCode = 4'b0101;
          Delay = 1'b0;
          #period;
          functionCode = 4'b0100;
          ComparatorInput = 2'b10;
          OPCode = 4'b0100;
          Delay = 1'b0;
          #period;
       functionCode = 4'b0100;
ComparatorInput = 2'b10;
          OPCode = 4'b0110;
          Delay = 1'b0;
          #period;
          functionCode = 4'b0100;
          ComparatorInput = 2'b10;
          OPCode = 4'b0001;
          Delay = 1'b0;
          #period;
          functionCode = 4'b0001;
          ComparatorInput = 2'b10;
```

```
        OPCode = 4'b0010;
        Delay = 1'b0;
        #period;
        functionCode = 4'b0010;
        ComparatorInput = 2'b10;
        OPCode = 4'b1010;
        Delay = 1'b0;
        #period;
        functionCode = 4'b1010;
        ComparatorInput = 2'b10;
        OPCode = 4'b1011;
        Delay = 1'b0;
        #period;
        functionCode = 4'b1011;
         ComparatorInput = 2'b10;
        OPCode = 4'b1100;
        Delay = 1'b0;
        #period;
        functionCode = 4'b1100;
        ComparatorInput = 2'b10;
OPCode = 4'b1101;
        Delay = 1'b0;
#period;
        functionCode = 4'b1101;
ComparatorInput = 2'b10;
   OPCode = 4'b0111;
        Delay = 1'b0;
   #period;
      functionCode = 4'b1101;
        ComparatorInput = 2'b10;
        OPCode = 4'b1111;
        Delay = 1'b0;
```

```
            #period;
            $finish;
        end

endmodule
```

Main Controlcomp Source Code

```
`include "Main_Control.v"
`include "comparator.v"

module controlcomp(Delay, OPCode, functionCode, ComparatorInput, ALUSrc1, ALUSrc2, MemRead, MemWrite, MemtoReg,
UpperByteToZero, TargetAddress, IFIDFlush, str_Byte, Exception, ALUOp, Comparison, RegWrite, SignExtend, OP1, R15, branch);
input Delay;
input signed [15:0] OP1,
                    R15;
input       [3:0]  OPCode,
                    functionCode;
input       [1:0]  ComparatorInput;

output wire        ALUSrc1,
                   ALUSrc2,
                   MemRead,
                   MemWrite,
                   MemtoReg,
                   UpperByteToZero,
                   TargetAddress,
                   IFIDFlush,
                   str_Byte,
                   Exception;

output wire [1:0]  ALUOp,
                   Comparison,
                   RegWrite,
                   SignExtend,
                   branch;
wire[1:0]          branchinput;
```

```
Main_Control u1(.Delay(Delay), .OPCode(OPCode), .functionCode(functionCode), .ComparatorInput(branchinput),
.ALUSrc1(ALUSrc1), .ALUSrc2(ALUSrc2), .MemRead(MemRead), .MemWrite(MemWrite), .MemtoReg(MemtoReg),
.UpperByteToZero(UpperByteToZero), .TargetAddress(TargetAddress), .IFIDFlush(IFIDFlush), .str_Byte(str_Byte),
.Exception(Exception), .ALUOp(ALUOp), .Comparison(Comparison), .RegWrite(RegWrite), .SignExtend(SignExtend));
comparator u2(.OP1(OP1), .R15(R15), .branch(branchinput));

endmodule


`include "controlcomp.v"

module controlcomp_fixture;

// Declare variables for stimulating input
reg         Delay;
reg [15:0] OP1, R15;
reg [3:0]  OPCode, functionCode;


wire       ALUSrc1,
                  ALUSrc2,
                  MemRead,
                  MemWrite,
                  MemtoReg,
                  UpperByteToZero,
                  TargetAddress,
                  IFIDFlush,
                  str_Byte,
                  Exception;

wire[1:0]  ALUOp,
```

```
                Comparison,
                RegWrite,
                SignExtend;

initial
    $vcdpluson;
initial
        $monitor ($time, " Delay = %b OP1 = %h R15 = %h OPCode = %b functionCode = %b  ALUSrc1 = %b ALUSrc2 = %b
MemRead = %b MemWrite %b MemtoReg = %b UpperByteToZero = %b TargetAddress = %b IFIDFlush = %b str_Byte =%b
Exception = %b ALUOp = %b Comparison = %b RegWrite = %b SignExtend = %b" , Delay, OP1, R15, OPCode, functionCode,
ALUSrc1, ALUSrc2, MemRead, MemWrite, MemtoReg, UpperByteToZero, TargetAddress, IFIDFlush, str_Byte, Exception,
ALUOp, Comparison, RegWrite, SignExtend) ;
localparam period = 25;

controlcomp c1(         .Delay(Delay),
                                .OPCode(OPCode),
                                .functionCode(functionCode),
                                .ALUSrc1(ALUSrc1),
            .ALUSrc2(ALUSrc2),
                                .MemRead(MemRead),
            .MemWrite(MemWrite),
            .MemtoReg(MemtoReg),
            .UpperByteToZero(UpperByteToZero),
            .TargetAddress(TargetAddress),
            .IFIDFlush(IFIDFlush),
            .str_Byte(str_Byte),
            .Exception(Exception),
            .ALUOp(ALUOp),
                                .Comparison(Comparison),
                                .RegWrite(RegWrite),
            .SignExtend(SignExtend),
                                .OP1(OP1),
```

```
                                        .R15(R15));

// Stimulate the Clear Signal
   initial
      begin
                        functionCode = 4'b0100;
                              OP1 = 16'b0000000000000101;
         R15 = 16'b0000000000000100;
         OPCode = 4'b0000;
                              Delay = 1'b1;
         #period;
                              functionCode = 4'b0100;
         OP1 = 16'b0000000000000101;
         R15 = 16'b0000000000000100;
         OPCode = 4'b0000;
                              Delay = 1'b0;
         #period;
                              functionCode = 4'b0100;
         OP1 = 16'b0000000000000101;
         R15 = 16'b0000000000000111;
         OPCode = 4'b0101;
                              Delay = 1'b0;
         #period;
                              functionCode = 4'b0100;
         OP1 = 16'b0000000000000101;
         R15 = 16'b0000000000000100;
         OPCode = 4'b0100;
                              Delay = 1'b0;
         #period;
                        functionCode = 4'b0100;
         OP1 = 16'b0000000000000101;
         R15 = 16'b0000000000000101;
```

```
OPCode = 4'b0110;
                Delay = 1'b0;
#period;
                functionCode = 4'b0100;
OP1 = 16'b0000000000000101;
R15 = 16'b0000000000000100;
OPCode = 4'b0001;
                Delay = 1'b0;
#period;
                functionCode = 4'b0001;
OP1 = 16'b0000000000000101;
R15 = 16'b0000000000000100;
OPCode = 4'b0010;
                Delay = 1'b0;
#period;
                functionCode = 4'b0010;
OP1 = 16'b0000000000000101;
R15 = 16'b0000000000000100;
OPCode = 4'b1010;
                Delay = 1'b0;
#period;
                functionCode = 4'b1010;
OP1 = 16'b0000000000000101;
R15 = 16'b0000000000000100;
OPCode = 4'b1011;
                Delay = 1'b0;
#period;
                functionCode = 4'b1011;
OP1 = 16'b0000000000000101;
R15 = 16'b0000000000000100;
OPCode = 4'b1100;
                Delay = 1'b0;
```

```
        #period;
                        functionCode = 4'b1100;
        OP1 = 16'b0000000000000101;
        R15 = 16'b0000000000000100;
        OPCode = 4'b1101;
                        Delay = 1'b0;
        #period;
                    functionCode = 4'b1101;
        OP1 = 16'b0000000000000101;
        R15 = 16'b0000000000000100;
        OPCode = 4'b0111;
                        Delay = 1'b0;
        #period;
                    functionCode = 4'b1101;
        OP1 = 16'b0000000000000101;
        R15 = 16'b0000000000000100;
        OPCode = 4'b1111;
                        Delay = 1'b0;
        #period;
                        $finish;
    end

endmodule


module Main_Control(
          input Delay,
       input [3:0] OPCode, functionCode,
          input [1:0] ComparatorInput,
       output reg ALUSrc1, ALUSrc2, MemRead, MemWrite, MemtoReg, UpperByteToZero, TargetAddress, IFIDFlush, str_Byte,
Exception,
       output reg [1:0] ALUOp, Comparison, RegWrite, SignExtend
```

```
);

    always @(*) begin

      if(Delay) begin

            RegWrite = 2'b00;
            ALUSrc1 = 1'b1;
            ALUSrc2 = 1'b1;
            ALUOp = 2'b00;
            SignExtend = 2'b00;
            MemRead = 1'b0;
            MemWrite = 1'b0;
            MemtoReg = 1'b0;
            UpperByteToZero = 1'b0;
            TargetAddress = 1'b1;
            IFIDFlush = 1'b0;
            str_Byte = 1'b0;
            Comparison = 2'b00;
            Exception = 1'b0;

      end

      else begin

            casex (OPCode)
              //R type instructions
             4'b0000 : begin

                if(functionCode == 4'b0100 || functionCode == 4'b0101) begin
                     RegWrite = 2'b10;
                     ALUSrc1 = 1'b1;
```

```
        ALUSrc2 = 1'b1;
        ALUOp = 2'b00;
        SignExtend = 2'b00;
        MemRead = 1'b0;
        MemWrite = 1'b0;
        MemtoReg = 1'b0;
        UpperByteToZero = 1'b0;
        TargetAddress = 1'b1;
        IFIDFlush = 1'b1;
        str_Byte = 1'b0;
        Comparison = 2'bxx;
        Exception = 1'b0;

    end

    else begin

        RegWrite = 2'b01;
        ALUSrc1 = 1'b1;
        ALUSrc2 = 1'b1;
        ALUOp = 2'b00;
        SignExtend = 2'b00;
        MemRead = 1'b0;
        MemWrite = 1'b0;
        MemtoReg = 1'b0;
        UpperByteToZero = 1'b0;
        TargetAddress = 1'b1;
        IFIDFlush = 1'b1;
        str_Byte = 1'b0;
        Comparison = 2'bxx;
        Exception = 1'b0;
```

```
                    end
                end
//AND instruction*/
4'b0001 : begin
  RegWrite = 2'b01;
  ALUSrc1 = 1'b1;
  ALUSrc2 = 1'b0;
  ALUOp = 2'b10;
  SignExtend = 2'b11;
  MemRead = 1'b0;
  MemWrite = 1'b0;
  MemtoReg = 1'b0;
  UpperByteToZero = 1'b0;
  TargetAddress = 1'b1;
  IFIDFlush = 1'b1;
  str_Byte = 1'b0;
  Comparison = 2'bxx;
  Exception = 1'b0;
end
//OR
4'b0010 : begin
  RegWrite = 2'b01;
  ALUSrc1 = 1'b1;
  ALUSrc2 = 1'b0;
  ALUOp = 2'b11;
  SignExtend = 2'b11;
  MemRead = 1'b0;
  MemWrite = 1'b0;
  MemtoReg = 1'b0;
  UpperByteToZero = 1'b0;
  TargetAddress = 1'b1;
  IFIDFlush = 1'b1;
```

```
  str_Byte = 1'b0;
  Comparison = 2'bxx;
  Exception = 1'b0;
end
//LBU
4'b1010 : begin
  RegWrite = 2'b01;
  ALUSrc1 = 1'b0;
  ALUSrc2 = 1'b0;
  ALUOp = 2'b01;
  SignExtend = 2'b10;
  MemRead = 1'b1;
  MemWrite = 1'b0;
  MemtoReg = 1'b1;
  UpperByteToZero = 1'b1;
  TargetAddress = 1'b1;
  IFIDFlush = 1'b1;
  str_Byte = 1'b0;
  Comparison = 2'bxx;
  Exception = 1'b0;
end
//SB
4'b1011 : begin
  RegWrite = 2'b00;
  ALUSrc1 = 1'b0;
  ALUSrc2 = 1'b0;
  ALUOp = 2'b01;
  SignExtend = 2'b10;
  MemRead = 1'b0;
  MemWrite = 1'b1;
  MemtoReg = 1'b0;
  UpperByteToZero = 1'b0;
```

```
  TargetAddress = 1'b1;
  IFIDFlush = 1'b1;
  str_Byte = 1'b1;
  Comparison = 2'bxx;
  Exception = 1'b0;
end
//LW
4'b1100 : begin
  RegWrite = 2'b01;
  ALUSrc1 = 1'b0;
  ALUSrc2 = 1'b0;
  ALUOp = 2'b01;
  SignExtend = 2'b10;
  MemRead = 1'b1;
  MemWrite = 1'b0;
  MemtoReg = 1'b1;
  UpperByteToZero = 1'b0;
  TargetAddress = 1'b1;
  IFIDFlush = 1'b1;
  str_Byte = 1'b0;
  Comparison = 2'bxx;
  Exception = 1'b0;
end
//SW
4'b1101 : begin
  RegWrite = 2'b00;
  ALUSrc1 = 1'b0;
  ALUSrc2 = 1'b0;
  ALUOp = 2'b01;
  SignExtend = 2'b10;
  MemRead = 1'b0;
  MemWrite = 1'b1;
```

```
  MemtoReg = 1'b0;
  UpperByteToZero = 1'b0;
  TargetAddress = 1'b1;
  IFIDFlush = 1'b1;
  str_Byte = 1'b0;
  Comparison = 2'bxx;
  Exception = 1'b0;
end
//BLT
4'b0101 : begin
  if (ComparatorInput == 2'b01) begin
                RegWrite = 2'b01;
                ALUSrc1 = 1'b0;
                ALUSrc2 = 1'b0;
                ALUOp = 2'bxx;
                SignExtend = 2'b01;
                MemRead = 1'b0;
                MemWrite =  1'b0;
                MemtoReg = 1'b0;
                UpperByteToZero = 1'b0;
                TargetAddress = 1'b1;
                IFIDFlush = 1'b0;
                str_Byte = 1'b0;
                Comparison = 2'b00;
                Exception = 1'b0;
  end
        else begin
                RegWrite = 2'b00;
                ALUSrc1 = 1'b0;
                ALUSrc2 = 1'b0;
                ALUOp = 2'bxx;
                SignExtend = 2'b00;
```

```
                MemRead = 1'b0;
                MemWrite = 1'b0;
                MemtoReg = 1'b0;
                UpperByteToZero = 1'b0;
                TargetAddress = 1'b0;
                IFIDFlush = 1'b1;
                str_Byte = 1'b0;
                Comparison = 2'b00;
                Exception = 1'b0;
        end
end
//BGT
4'b0100 : begin
  if (ComparatorInput == 2'b10) begin
    RegWrite = 2'b01;
    ALUSrc1 = 1'b0;
    ALUSrc2 = 1'b0;
    ALUOp = 2'bxx;
    SignExtend = 2'b01;
    MemRead = 1'b0;
    MemWrite = 1'b0;
    MemtoReg = 1'b0;
    UpperByteToZero = 1'b0;
    TargetAddress = 1'b1;//active high
    IFIDFlush = 1'b0;//low active
    str_Byte = 1'b0;
    Comparison = 2'b01;
    Exception = 1'b0;
  end
        else begin
                RegWrite = 2'b00;
                ALUSrc1 = 1'b0;
```

```
                ALUSrc2 = 1'b0;
                ALUOp = 2'bxx;
                SignExtend = 2'b00;
                MemRead = 1'b0;
                MemWrite = 1'b0;
                MemtoReg = 1'b0;
                UpperByteToZero = 1'b0;
                TargetAddress = 1'b0;
                IFIDFlush = 1'b1;
                str_Byte = 1'b0;
                Comparison = 2'b00;
                Exception = 1'b0;
        end
end
//BEQ
4'b0110 : begin
  if (ComparatorInput == 2'b11) begin
  RegWrite = 2'b01;
  ALUSrc1 = 1'b0;
  ALUSrc2 = 1'b0;
  ALUOp = 2'bxx;
  SignExtend = 2'b01;
  MemRead = 1'b0;
  MemWrite = 1'b0;
  MemtoReg = 1'b0;
  UpperByteToZero = 1'b0;
  TargetAddress = 1'b1;
  IFIDFlush = 1'b0;
  str_Byte = 1'b0;
  Comparison = 2'b10;
  Exception = 1'b0;
  end
```

```
        else begin
                RegWrite = 2'b00;
                ALUSrc1 = 1'b0;
                ALUSrc2 = 1'b0;
                ALUOp = 2'bxx;
                SignExtend = 2'b00;
                MemRead = 1'b0;
                MemWrite =  1'b0;
                MemtoReg = 1'b0;
                UpperByteToZero = 1'b0;
                TargetAddress = 1'b0;
                IFIDFlush = 1'b1;
                str_Byte = 1'b0;
                Comparison = 2'b00;
                Exception = 1'b0;
        end
end
//JUMP
 4'b0111 : begin
   RegWrite = 2'b00;
   ALUSrc1 = 1'b0;
   ALUSrc2 = 1'b0;
   ALUOp = 2'bxx;
   SignExtend = 2'b00;
   MemRead = 1'b0;
   MemWrite =  1'b0;
   MemtoReg = 1'b0;
   UpperByteToZero = 1'b0;
   TargetAddress = 1'b0;
   IFIDFlush = 1'b0;
   str_Byte = 1'b0;
   Comparison = 2'bxx;
```

```
  Exception = 1'b0;
 end
 //HALT
 4'b1111 : begin
  RegWrite = 2'b00;
  ALUSrc1 = 1'b0;
  ALUSrc2 = 1'b0;
  ALUOp = 2'bxx;
  SignExtend = 2'bxx;
  MemRead = 1'b0;
  MemWrite = 1'b0;
  MemtoReg = 1'b0;
  UpperByteToZero = 1'b0;
  TargetAddress = 1'b0;
  IFIDFlush = 1'b0;
  str_Byte = 1'b0;
  Comparison = 2'bxx;
  Exception = 1'b0;
 end
 default : begin
  RegWrite = 2'b00;
  ALUSrc1 = 1'b0;
  ALUSrc2 = 1'b0;
  ALUOp = 2'b00;
  SignExtend = 2'b00;
  MemRead = 1'b0;
  MemWrite = 1'b0;
  MemtoReg = 1'b0;
  UpperByteToZero = 1'b0;
  TargetAddress = 1'b0;
  IFIDFlush = 1'b1;
  str_Byte = 1'b0;
```

```
     Comparison = 2'bxx;
     Exception = 1'b0;
    end

endcase
end
end

endmodule


`include "Main_Control.v"


module Main_Control_fixture;

// Declare variables for stimulating input
reg         Delay;
reg [3:0]  OPCode, functionCode;
reg [1:0]  ComparatorInput;

wire       ALUSrc1,
                    ALUSrc2,
                    MemRead,
                    MemWrite,
                    MemtoReg,
                    UpperByteToZero,
                    TargetAddress,
                    IFIDFlush,
                    str_Byte,
                    Exception;
```

```
wire[1:0]  ALUOp,
                Comparison,
                RegWrite,
                SignExtend;

initial
      $vcdpluson;
initial
        $monitor ($time , " Delay = %b OPCode = %b functionCOde = %b ComparatorInput = %b ALUSrc1 = %b ALUSrc2 = %b
MemRead = %b MemWrite %b MemtoReg %b UpperByteToZero %b TargetAddress = %b IFIDFlush = %b str_Byte = %b
Exception = %b ALUOp = %b Comparison = %b RegWrite = %b SignExtend = %b " , Delay, OPCode, functionCode,
ComparatorInput, ALUSrc1, ALUSrc2, MemRead, MemWrite, MemtoReg, UpperByteToZero, TargetAddress, IFIDFlush, str_Byte,
Exception, ALUOp, Comparison, RegWrite, SignExtend);
localparam period = 25;

Main_Control c1(        .Delay(Delay),
                                .OPCode(OPCode),
                                .functionCode(functionCode),
                                .ComparatorInput(ComparatorInput),
                                .ALUSrc1(ALUSrc1),
            .ALUSrc2(ALUSrc2),
                                .MemRead(MemRead),
            .MemWrite(MemWrite),
            .MemtoReg(MemtoReg),
            .UpperByteToZero(UpperByteToZero),
            .TargetAddress(TargetAddress),
            .IFIDFlush(IFIDFlush),
            .str_Byte(str_Byte),
            .Exception(Exception),
            .ALUOp(ALUOp),
                                .Comparison(Comparison),
                                .RegWrite(RegWrite),
```

```
            .SignExtend(SignExtend));


// Stimulate the Clear Signal
  initial
     begin
                  functionCode = 4'b0101;
                        ComparatorInput = 2'b11;
        OPCode = 4'b0110;
                        Delay = 1'b0;
        #period;
                        functionCode = 4'b0101;
                        ComparatorInput = 2'b10;
        OPCode = 4'b0101;
                        Delay = 1'b0;
        #period;
                        functionCode = 4'b0101;
                        ComparatorInput = 2'b11;
        OPCode = 4'b0101;
                        Delay = 1'b0;
        #period;

                        functionCode = 4'b0100;
                        ComparatorInput = 2'b01;
        OPCode = 4'b0100;
                        Delay = 1'b0;
        #period;
                        functionCode = 4'b0100;
                        ComparatorInput = 2'b10;
        OPCode = 4'b0100;
                        Delay = 1'b0;
        #period;
```

```
                    functionCode = 4'b0100;
                    ComparatorInput = 2'b11;
        OPCode = 4'b0100;
                    Delay = 1'b0;
        #period;

                functionCode = 4'b0110;
                    ComparatorInput = 2'b01;
        OPCode = 4'b0110;
                    Delay = 1'b0;
        #period;
                    functionCode = 4'b0110;
                    ComparatorInput = 2'b10;
        OPCode = 4'b0110;
                    Delay = 1'b0;
        #period;
                    functionCode = 4'b0110;
                    ComparatorInput = 2'b11;
        OPCode = 4'b0110;
                    Delay = 1'b0;
        #period;


                    $finish;
    end

endmodule
```

Exception Handling Source Code

```
module Exception_Handling(
                input ovf, opcode,
                output reg exception_output
                );

always @(*) begin
        if (ovf || opcode)
                exception_output = 1;
        else
                exception_output = 0;
        end
endmodule


`include "Exception_Handling.v"
module Exception_Handling_Fixture;
        reg ovf, opcode;
        wire exception_output;

initial
        $monitor($time, "input_ovf = %b input_opcode = %b exception_output = %b \n" , ovf, opcode, exception_output);

localparam period = 25;

Exception_Handling u1(.ovf(ovf), .opcode(opcode), .exception_output(exception_output));
```

```
initial
begin
        ovf = 1'b0;
        opcode = 1'b0;
        #period;
        ovf = 1'b0;
        opcode = 1'b1;
        #period;
        ovf = 1'b1;
        opcode = 1'b0;
        #period;
        $finish;
end
endmodule
```

Forward Source Code

```
module forward(
    input      [3:0]  registerOP1ID,
                      registerOP2ID,
                      registerOP1EX,
                      registerOP1MEM,
    input             regWriteMEM,
                      regWriteWB,
    output reg [1:0]  forwardA,
                      forwardB
);

    always @ (*) begin
        //OP1 at EX/MEM = OP1 at ID/EX
        if (regWriteMEM && (registerOP1EX != 0) && (registerOP1EX == registerOP1ID))
            forwardA = 2'b01;
        //OP1 at MEM/WB = OP1 at ID/EX
        else if (regWriteMEM && (registerOP1EX != 0) && (registerOP1MEM == registerOP1ID))
            forwardA = 2'b00;
        else
            forwardA = 2'b10;

        //MEM stage
        if (regWriteMEM && (registerOP1EX != 0) && (registerOP1EX == registerOP2ID))
            forwardB = 2'b10;
        else if (regWriteMEM && (registerOP1EX != 0) && (registerOP1MEM == registerOP2ID))
            forwardB = 2'b10;
        else
            forwardB = 2'b01;
    end
endmodule
```

```
`include "forward.v"

module forward_fixture;

reg [3:0] registerOP1ID,
          registerOP2ID,
          registerOP1EX,
          registerOP1MEM;
reg       regWriteMEM,
          regWriteWB;
wire [1:0]forwardA,
          forwardB;

initial
        $monitor ($time, " registerOP1ID = %b registerOP2ID = %b registerOP1EX = %b register OP1MEM = %b regWriteMEM =
%b regWriteWB = %b forwardA = %b forwardB = %b", registerOP1ID, registerOP2ID, registerOP1EX, registerOP1MEM,
regWriteMEM, regWriteWB, forwardA, forwardB);

localparam period = 25;

forward u1(.registerOP1ID(registerOP1ID), .registerOP2ID(registerOP2ID), .registerOP1EX(registerOP1EX),
.registerOP1MEM(registerOP1MEM), .regWriteMEM(regWriteMEM), .regWriteWB(regWriteWB), .forwardA(forwardA),
.forwardB(forwardB));

initial
        begin
        registerOP1ID = 4'b1000;
        registerOP2ID = 4'b1000;
        registerOP1EX = 4'b1000;
        registerOP1MEM = 4'b1100;
```

```
        regWriteMEM = 1'b1;
        regWriteWB = 1'b1;
        #period;
        registerOP1ID = 4'b1100;
        registerOP2ID = 4'b1100;
        registerOP1EX = 4'b1000;
        registerOP1MEM = 4'b1100;
        regWriteMEM = 1'b1;
        regWriteWB = 1'b1;
        #period;
        registerOP1ID = 4'b1000;
        registerOP2ID = 4'b1000;
        registerOP1EX = 4'b1100;
        registerOP1MEM = 4'b1000;
        regWriteMEM = 1'b1;
        regWriteWB = 1'b1;
        #period;
        registerOP1ID = 4'b1000;
        registerOP2ID = 4'b1000;
        registerOP1EX = 4'b1100;
        registerOP1MEM = 4'b1100;
        regWriteMEM = 1'b1;
        regWriteWB = 1'b1;
        #period;
        $finish;
end
endmodule
```

Main ALU Source Code

```
module main_alu (input signed [15:0] a, b, input [2:0] sel, output reg signed [15:0] R15, r, output reg ovf);
always @ (*)
casex (sel)
  3'b000: begin
    ovf = 0;
    {ovf, r} = a + b;
  end
  3'b001: begin
    ovf = 0;
    {ovf, r} = a - b;
  end
  3'b010: begin
    ovf = 0;
    {R15, r} = a * b;
  end
  3'b011: begin
    ovf = 0;
    r = a / b;
    R15 = a % b;
  end
  3'b100: begin
    ovf = 0;
    r = a & b;
  end
  3'b101: begin
    ovf = 0;
    r = a | b;
  end
endcase
endmodule
```

```
`include "main_alu.v"

module main_alu_fixture;

        reg signed [15:0] a, b;
        reg [2:0] sel;
        wire signed [15:0] R15, r;
        wire ovf;


initial
        $monitor ($time , " a = %h b = %h sel = %b R15 = %hex r = %hex ovf = %b" , a , b , sel, R15, r, ovf);
localparam period = 25;

main_alu a1(.a(a), .b(b), .sel(sel), .R15(R15), .r(r), .ovf(ovf));

initial
        begin
        sel = 3'b000;
        a = 16'hfb18;
        b = 16'hf666;
        #period;
        sel = 3'b001;
        a = 16'h0008;
        b = 16'h0004;
        #period;
        sel = 3'b010;
        a = 16'h0008;
        b = 16'h0005;
        #period;
```

```
        sel = 3'b011;
        a = 16'h0051;
        b = 16'h0005;
        #period;
        sel = 3'b100;
        a = 16'h0051;
        b = 16'h0525;
        #period
        sel = 3'b101;
        a = 16'h0542;
        b = 16'h0001;
        #period
        $finish;
end
endmodule
```

Shift Left Source Code

```
module shift_left(
            output [15:0] Out,
            input [15:0] In
           );
   assign Out = {In[14:0],1'b0};
endmodule


`include "shift_left.v"

module shift_left_fixture;

reg [15:0] In;
wire [15:0] Out;

initial
        $monitor ($time , " In = %b Out = %b" , In, Out);
localparam period = 25;

shift_left u1(.In(In), .Out(Out));

initial
        begin
        In = 16'hc526;
        #period;
        In = 16'h0525;
        #period;
        $finish;
end
endmodule
```

Sign Extend Source Code

```
module sign_extend(
  input [1:0] SignalIn,
  input [11:0] in,
  output reg [15:0] Out
  );
  always @(*)
    casex (SignalIn)
     2'b00: Out = {{4{in[11]}}, in[11:0]}; //4 Extend
     2'b01: Out = {{8{in[7]}}, in[7:0]}; //8 Extend
     2'b10: Out = {{12{in[3]}}, in[3:0]}; //12 Extend
     default: Out = {8'b00000000, in[7:0]}; //0 Extend
  endcase
endmodule



`include "sign_extend.v"

module sign_extend_fixture;

        reg [1:0] SignalIn;
        reg [11:0] in;
        wire [15:0] Out;

initial
        $monitor ($time , " SignalIn = %b in = %b Out = %b " , SignalIn, in, Out);
localparam period = 25;

sign_extend u1(.SignalIn(SignalIn), .in(in), .Out(Out));
```

```
initial
        begin
        SignalIn = 2'b00;
        in = 12'b000111110000;
        #period;
        SignalIn = 2'b01;
        in = 12'b000111110000;
        #period;
        SignalIn = 2'b10;
        in = 12'b000111110000;
        #period;
        SignalIn = 2'b11;
        in = 12'b000111110000;
        #period;
        $finish;
end
endmodule
```

Adder Module Source Code

```
module adder(input [15:0] a, b, output reg [15:0] c);

	always@(*) begin

		c = a + b;

	end
endmodule


`include "adder.v"

module adder_fixture;

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

	initial
		 $vcdpluson;

	initial
		$monitor($time, "  a = %h b = %h  c = %h \n", a[15:0], b[15:0], c[15:0]);

	adder dut(.a(a), .b(b), .c(c));

	initial begin
		a = 16'h0005;
		b = 16'h0005;

		#10;
```

```
            #10;
        end

        initial
            #50 $finish;

endmodule
```

Buffer Source Code

```
module buffer #(parameter N = 32) (input [N-1:0] d, input clk, rst, w_enable, output reg [N-1:0] q);

	always@(posedge clk, negedge rst) begin

		if(!rst)
			q <= 0;

		else begin

			if(w_enable)
				q <= d;
		end
	end

endmodule


`include "buffer.v"

module buffer_fixture;

	reg [15:0] d;
	reg w_enable, clk, rst;
	wire [15:0] q;

	initial
		 $vcdpluson;

	initial
		$monitor($time, "  d_in = %h  clk = %b  rst = %b  w_enable = %b  q = %h \n", d[15:0], clk, rst, w_enable, q[15:0]);
```

```
    buffer #(.N(16)) dut(.d(d), .clk(clk), .rst(rst), .w_enable(w_enable), .q(q));

    initial begin

        rst = 1'b0;
        d = 16'h0004;
        w_enable = 1'b1;


        #10 rst = 1'b1;
        #20;

        #10 w_enable = 1'b0;

        #10 d = 16'h000A;
        #20 w_enable = 1'b1;
        #10 rst = 1'b0;
    end

    initial begin
        clk = 1'b0;
        forever #10 clk = ~clk;
    end

    initial
        #100 $finish;

endmodule
```

CPU Source Code

```
`include "adder.v"
`include "ALUControl.v"
`include "ALU.v"
`include "buffer.v"
`include "comparator.v"
`include "DataMem.v"
`include "Exception_Handling.v"
`include "forwarding_unit.v"
`include "Hazard_Detect.v"
`include "Instr_Mem.v"
`include "Main_Control_Unit.v"
`include "mux_A.v"
`include "mux_B.v"
`include "mux_C.v"
`include "PC.v"
`include "RegFile.v"
`include "shift_left.v"
`include "sign_extend.v"
`include "zeroExtend.v"

module cpu(input clk, rst);

        wire h2, h3, h4, h5, c1, c2, c3, c4, c5, c6, c7, c8, c9,
            m1, m2, m3, m4, m5, m6, m7, c10, EX_11, ID_11, c15;

        wire [1:0] h1, c11, c12, c13, c14, EX_9, EX_10, ID_10, m8, m9;

        wire [2:0] EX_8;
```

```
	wire [15:0] IF_1, IF_2, IF_3, IF_4, ID_2, ID_3, ID_4,
			ID_5, ID_6, ID_7, ID_8, ID_9, EX_2, EX_3,
			EX_4, EX_5, EX_6, EX_7, MEM_2, WB_2, WB_3;
	wire [31:0] ID_1;
	wire [73:0] EX_1;
	wire [58:0] MEM_1;
	wire [55:0] WB_1;


	//IF Stage
	mux_A d1(.a(IF_2), .b(ID_2), .s(c7), .c(IF_3));
	PC d2(.PC_in(IF_3), .halt_sig(h4), .clk(clk), .rst(rst), .PC_out(IF_1));
	adder d3(.a(16'h0002), .b(IF_1), .c(IF_2));
	Instr_Mem d4(.pointer(IF_1), .rst(rst), .clk(clk), .instr_out(IF_4));
	buffer #(.N(32)) d5(.d({IF_2, IF_4}), .q(ID_1), .clk(clk), .rst(rst), .w_enable(h3));

	//ID Stage
	Main_Control_Unit d6(.OPCode(ID_1[15:12]), .functionCode(ID_1[3:0]), .ComparatorInput(ID_10), .ALUSrc1(c1),
.ALUSrc2(c2), .MemRead(c3),
			.MemWrite(c4), .MemtoReg(c5), .UpperByteToZero(c6), .TargetAddress(c7),
			.IFIDFlush(c8), .str_Byte(c9), .Exception(c10), .ALUOp(c11), .Comparison(c12),
			.RegWrite (c13), .SignExtend(c14), .branch(c15));
	RegFile d7(.Reg1(ID_1[11:8]), .Reg2(ID_1[7:4]), .WriteReg(WB_1[3:0]), .RegWrite(WB_1[53:52]),
.WriteR15(WB_1[19:4]),
		.WriteData(WB_3), .clk(clk), .rst(rst), .R_D1(ID_3), .R_D2(ID_4), .R_R15(ID_5));
	adder d8(.a(ID_1[31:16]), .b(ID_7), .c(ID_2));
	sign_extend d9(.SignalIn(c14), .in(ID_1[11:0]), .Out(ID_6));
	shift_left d10(.In(ID_6), .Out(ID_7));
	comparator d11(.OP1(ID_8), .R15(ID_9), .branch(ID_10));
	mux_A d12(.a(MEM_1[51:36]), .b(ID_3), .s(EX_1[72]), .c(ID_8));
	mux_B d13(.a(ID_5), .b(MEM_1[35:20]), .c(MEM_1[51:36]), .s(EX_1[71:70]), .d(ID_9));
	Exception_Handling e1(.ovf(EX_11), .opcode(c10), .exception_output(ID_11));
```

```
        Hazard_Detect d14(.IF_ID_Op1(ID_1[11:8]), .IF_ID_Op2(ID_1[7:4]), .ID_EX_Op1(EX_1[7:4]), .Opcode(ID_1[15:12]),
                        .FunctionCode(EX_1[11:8]), .branch(c15), .RegWrite(EX_1[65:64]), .ID_EX_MemRead(EX_1[67]),
.EX_MEM_MemRead(MEM_1[55]),
                        .ExcepSig(ID_11), .R15_Sel(h1), .Op1_Sel(h2), .IF_ID_Hold(h3), .PC_Hold(h4),
                        .Delay(h5));
        mux_C e2(.Delay(h5), .str_byte_in(c9), .UpperByteToZero_in(c6), .MemWrite_in(c4), .MemRead_in(c3), .MemtoReg_in(c5),
.ALUSrc1_in(c1),
                .ALUSrc2_in(c2), .RegWrite_in(c13), .ALUOp_in(c11), .str_byte_out(m1), .UpperByteToZero_out(m2),
.MemWrite_out(m3),
                .MemRead_out(m4), .MemtoReg_out(m5), .ALUSrc1_out(m6), .ALUSrc2_out(m7), .RegWrite_out(m8),
.ALUOp_out(m9));
        buffer #(.N(74)) d15(.d({m1, h2, h1, m2, m3, m4, m5, m8, m9, m7, m6, ID_3, ID_4, ID_6, ID_1[3:0], ID_1[11:8],
ID_1[7:4]}),
                        .q(EX_1), .clk(clk), .rst(rst), .w_enable(1'b1));

        //EX Stage
        mux_A d16(.a(EX_2), .b(EX_3), .s(EX_1[60]), .c(EX_4));
        mux_B d17(.a(WB_3), .b(MEM_1[51:36]), .c(EX_1[59:44]), .s(EX_9), .d(EX_2));
        mux_A d18(.a(EX_3), .b(EX_1[27:12]), .s(EX_1[61]), .c(EX_5));
        mux_B d19(.a(MEM_1[51:36]), .b(EX_1[43:28]), .c(WB_3), .s(EX_10), .d(EX_3));
        ALU d20(.a(EX_4), .b(EX_5), .sel(EX_8), .r(EX_6), .R15(EX_7), .ovf(EX_11));
        ALUControl d21(.ALUOp(EX_1[63:62]), .Function(EX_1[11:8]), .ALUControl(EX_8));
        forwarding_unit d22(.registerOP1ID(EX_1[7:4]), .registerOP2ID(EX_1[3:0]), .registerOP1EX(MEM_1[3:0]),
                        .registerOP1MEM(WB_1[3:0]), .regWriteMEM(MEM_1[53:52]), .regWriteWB(WB_1[53:52]),
                        .forwardA(EX_9), .forwardB(EX_10));
        buffer #(.N(59)) d23(.d({EX_1[73], EX_1[69], EX_1[68], EX_1[67], EX_1[66], EX_1[65:64], EX_6, EX_7, EX_2,
EX_1[7:4]}),
                        .q(MEM_1), .clk(clk), .rst(rst), .w_enable(1'b1));

        //MEM Stage
        DataMem d24(.Address(MEM_1[51:36]), .WriteData(MEM_1[19:4]), .clk(clk), .rst(rst), .MemRead(MEM_1[55]),
.MemWrite(MEM_1[56]),
```

```
                .str_byte(MEM_1[58]), .d_out(MEM_2));
        buffer #(.N(56)) d25(.d({MEM_1[57], MEM_1[54], MEM_1[53:52], MEM_2, MEM_1[51:36], MEM_1[35:20],
MEM_1[3:0]}),
                .q(WB_1), .clk(clk), .rst(rst), .w_enable(1'b1));

        //WB Stage
        mux_A d26(.a(WB_1[51:36]), .b(WB_1[35:20]), .s(WB_1[54]), .c(WB_2));
        zeroExtend d27(.d_in(WB_2), .enable(WB_1[55]), .d_out(WB_3));


endmodule
```

```
`include "cpu.v"

module cpu_fixture;

        reg clk, rst;

        initial
                $vcdpluson;

        always@(negedge clk) begin
```

```
		$display(" \n\nclk = %b rst = %b \n ////////////// IF Stage ////////////// \n\nIF/ID_BufferData: PC = %h Instr = %h Adder = %h \n\n ////////////// ID Stage ////////////// \n\n Main Control Signals: str_byte = %b ALUSrc1 = %b ALUSrc2 = %b MemRead = %b MemWrite = %b MemtoReg = %b UpperByteToZero = %b TargetAddress = %b IF/ID_Flush = %b ExcepSig = %b ALUOp = %b Comparison = %b RegWrite = %b, SignExtend = %b branch = %b \n\n Hazard Detection Signals: PC_Hold = %b IF/ID_Hold = %b Delay = %b Op1_Sel = %b R15_Sel = %b \n\n Comparator Ports: OP1 = %h R15 = %h Branch_Result = %b \n\nID/EX_BufferData: Reg1_Data = %h Reg2_Data = %h Sign Extended Value = %h FunctionCode = %b Op1_RegID = %b Op2_RegID = %b \n\n////////////// EX Stage ////////////// \n\n Hazard Detection Signals: PC_Hold = %b IF/ID_Hold = %b Delay = %b Op1_Sel = %b R15_Sel = %b \n\nALU_OverflowSig = %b\n EX/MEM_BufferData: ALU_Result = %h R15_Result = %h Op1_RegData = %h Op1_RegID = %b str_byte = %b UpperByteToZero = %b MemWrite = %b MemRead = %b MemToReg = %b RegWrite = %b \n\n ////////////// MEM Stage ////////////// \n\nMEM/WB_BufferData: UpperByteToZero = %b MemToReg = %b RegWrite = %b Mem_Data = %h ALU_Result = %h R15_Result = %h Op1_RegID = %b \n\n /////////WB Stage /////////\n\n WriteData = %h WriteReg = %b ",
		clk, rst, cpu.d4.pointer[15:0], cpu.d5.q[15:0], cpu.d5.q[31:16], cpu.d6.str_Byte, cpu.d6.ALUSrc1, cpu.d6.ALUSrc2,
		cpu.d6.MemRead, cpu.d6.MemWrite, cpu.d6.MemtoReg, cpu.d6.UpperByteToZero, cpu.d6.TargetAddress, cpu.d6.IFIDFlush,
		cpu.d6.Exception, cpu.d6.ALUOp[1:0], cpu.d6.Comparison[1:0], cpu.d6.RegWrite[1:0], cpu.d6.SignExtend[1:0], cpu.d6.branch,
		cpu.d14.PC_Hold, cpu.d14.IF_ID_Hold, cpu.d14.Delay, cpu.d14.Op1_Sel, cpu.d14.R15_Sel[1:0], cpu.d11.OP1[15:0],
		cpu.d11.R15[15:0], cpu.d11.branch[1:0], cpu.d15.q[59:44], cpu.d15.q[43:28], cpu.d15.q[27:12], cpu.d15.q[11:8], cpu.d15.q[7:4],
		cpu.d15.q[3:0], cpu.d14.PC_Hold, cpu.d14.IF_ID_Hold, cpu.d14.Delay, cpu.d14.Op1_Sel, cpu.d14.R15_Sel[1:0], cpu.d20.ovf,
		cpu.d23.q[51:36], cpu.d23.q[35:20], cpu.d23.q[19:4], cpu.d23.q[3:0], cpu.d23.q[58], cpu.d23.q[57], cpu.d23.q[56], cpu.d23.q[55],
		cpu.d23.q[54], cpu.d23.q[53:52], cpu.d25.q[55], cpu.d25.q[54], cpu.d25.q[53:52], cpu.d25.q[51:36], cpu.d25.q[35:20],
		cpu.d25.q[19:4], cpu.d25.q[3:0], cpu.d27.d_out[15:0], cpu.d25.q[3:0]);

		if(cpu.d20.ovf)
			$display("\nArithmetic Overflow at time ", $time, ".\n");
		else
			$display("\nNo Exceptions have occurred.\n");


	end

	cpu dut(.clk(clk), .rst(rst));

	initial begin
```

```
        rst = 1'b0;
        #1 rst = 1'b1;

//      #10; #10; #10; #10; #50;
end

initial begin

        clk = 1'b0;
        forever #10 clk = ~clk;

end

initial
        #750 $finish;


Endmodule
```

Data Memory Source Code

```
module DataMem(input [15:0] Address, WriteData, input clk, rst, MemRead, MemWrite, str_byte, output reg [15:0] d_out);

	reg [7:0] Memory [65535:0];
	integer i;

	always@(posedge clk or negedge rst) begin

		if(!rst) begin

			Memory[0] <= 8'h31;
			Memory[1] <= 8'h42;
			Memory[2] <= 8'h00;
			Memory[3] <= 8'h00;
			Memory[4] <= 8'h56;
			Memory[5] <= 8'h78;
			Memory[6] <= 8'hDE;
			Memory[7] <= 8'hAD;
			Memory[8] <= 8'hBE;
			Memory[9] <= 8'hEF;

			for(i = 10; i <= 65535; i = i + 1) begin

				Memory[i] <= 8'h00;

			end
		end

		else if(MemWrite && !str_byte) begin

			Memory[Address] <= WriteData[15:8];
```

```
                Memory[Address+1] <= WriteData[7:0];
            end
            else if(MemWrite && str_byte) begin
                Memory[Address] <= WriteData[7:0];
            end
        end
        always@(*) begin
            if(MemRead) begin
                d_out = {Memory[Address], Memory[Address+1]};
            end
        end
endmodule

`include "DataMem.v"

module DataMem_fixture;

        reg [15:0] WriteData, Address;
        reg clk, rst, MemRead, MemWrite, str_byte;
```

```
	wire [15:0] d_out;


	initial
		$vcdpluson;


	initial
		$monitor($time, "  Addr = %h  WriteData = %h  MemRead = %b  MemWrite = %b  str_byte = %b  rst = %b  clk = %b
d_out = %h\n",
				Address[15:0], WriteData[15:0], MemRead, MemWrite, str_byte, rst, clk, d_out[15:0]);

	DataMem dut(.Address(Address), .WriteData(WriteData), .rst(rst), .clk(clk), .MemRead(MemRead),
		.MemWrite(MemWrite), .str_byte(str_byte), .d_out(d_out));

	initial begin

		rst = 1'b1; WriteData = 16'hABCD; Address = 16'h0004;
	      MemRead = 1'b0; MemWrite = 1'b0; str_byte = 1'b0;
		#10 rst = 1'b0;
		#20 rst = 1'b1; MemRead = 1'b1;
		#10
		#20 MemRead = 1'b0; MemWrite = 1'b1;
		#20 MemRead = 1'b1; MemWrite = 1'b0;
		#10
		#10 rst = 1'b0;
		#20 rst = 1'b1; str_byte = 1'b1; MemWrite = 1'b0; MemRead = 1'b0;
		#10 MemWrite = 1'b1;
		#20 MemRead = 1'b1; str_byte = 1'b0; MemWrite = 1'b0;
		#20 rst = 1'b0;
```

```
        end

        initial begin

                clk = 1'b0;
                forever #10 clk = ~clk;

        end

        initial begin

                #200 $finish;

        end

endmodule
```

Hazard Detection Source Code

```
module Hazard_Detect(input [3:0] IF_ID_Op1, IF_ID_Op2, ID_EX_Op1, Opcode,
                     FunctionCode, input [1:0] RegWrite, input ID_EX_MemRead, EX_MEM_MemRead, ExcepSig, branch,
                     output reg [1:0] R15_Sel, output reg Op1_Sel, IF_ID_Hold, PC_Hold, Delay);

/* input ExcepSig tells the Hazard detection unit whether to hold the IF/ID
 * buffer and program counter. Since we were not required to stop the program
 * but only display a message when an exception occurred, this functionality
 * is not implemented here */

        always@(*) begin

                if(Opcode == 4'b0101 || Opcode == 4'b0100 || Opcode == 4'b0110) begin

                        if((IF_ID_Op1 == ID_EX_Op1 || ID_EX_Op1 == 4'b1111) && ID_EX_MemRead) begin

                                Delay = 1;
                                Op1_Sel = 0;
                                R15_Sel = 2'b00;
                                IF_ID_Hold = 0;
                                PC_Hold = 1;

                        end

                        else if(!ID_EX_MemRead && EX_MEM_MemRead) begin

                                Delay = 1;
                                Op1_Sel = 0;
                                R15_Sel = 2'b00;
                                IF_ID_Hold = 0;
                                PC_Hold = 1;
```

```
        end

        else if(IF_ID_Op1 == ID_EX_Op1 && RegWrite != 2'b00 && IF_ID_Op1 != 4'b1111 && ID_EX_Op1 !=
4'b1111 && (FunctionCode == 4'b0000 || FunctionCode == 4'b0001)) begin

                Delay = 1;
                Op1_Sel = 1;
                R15_Sel = 2'b00;
                IF_ID_Hold = 0;
                PC_Hold = 1;

        end

        else if(RegWrite != 2'b00 && IF_ID_Op1 == 4'b1111 && ID_EX_Op1 == 4'b1111) begin

                Delay = 1;
                Op1_Sel = 1;
                R15_Sel = 2'b10;
                IF_ID_Hold = 0;
                PC_Hold = 1;

        end

        else if(RegWrite != 2'b00 && ID_EX_Op1 == 4'b1111 && IF_ID_Op1 != 4'b1111 && (FunctionCode ==
4'b0000 || FunctionCode == 4'b0001)) begin

                Delay = 1;
                Op1_Sel = 0;
                R15_Sel = 2'b01;
                IF_ID_Hold = 0;
                PC_Hold = 1;
```

```
                end

                else if(RegWrite != 2'b00 && ID_EX_Op1 == IF_ID_Op1 && ID_EX_Op1 != 4'b1111 && IF_ID_Op1 !=
4'b1111 && (FunctionCode == 4'b0100 || FunctionCode == 4'b0101)) begin

                        Delay = 1;
                        Op1_Sel = 1;
                        R15_Sel = 2'b01;
                        IF_ID_Hold = 0;
                        PC_Hold = 1;

                end

                else if(RegWrite != 2'b00 && ID_EX_Op1 != IF_ID_Op1 && (FunctionCode == 4'b0100 || FunctionCode ==
4'b0101)) begin

                        Delay = 1;
                        Op1_Sel = 0;
                        R15_Sel = 2'b01;
                        IF_ID_Hold = 0;
                        PC_Hold = 1;

                end


                else if(branch == 1'b1) begin

                        IF_ID_Hold = 1;
                        PC_Hold = 0;
                        Delay = 1;
                        Op1_Sel = 0;
```

```
                R15_Sel = 2'b00;

        end


        else begin

                Delay = 0;
                Op1_Sel = 0;
                R15_Sel = 2'b00;
                PC_Hold = 0;
                IF_ID_Hold = 1;

        end

end

else if(Opcode == 4'b1111) begin

                IF_ID_Hold = 0;
                PC_Hold = 1;
                Delay = 1;
                Op1_Sel = 0;
                R15_Sel = 2'b00;

end

else if(((IF_ID_Op1 == ID_EX_Op1) || (IF_ID_Op2 == ID_EX_Op1)) && ID_EX_MemRead) begin

                IF_ID_Hold = 0;
                PC_Hold = 1;
                Delay = 1;
```

```
                    Op1_Sel = 0;
                    R15_Sel = 2'b00;

            end

                else begin

                        IF_ID_Hold = 1;
                        PC_Hold = 0;
                        Delay = 0;
                        Op1_Sel = 0;
                        R15_Sel = 2'b00;


                end

        end

endmodule


`include "Hazard_Detect.v"

module Hazard_Detect_fixture;

        reg [3:0] IF_ID_Op1, IF_ID_Op2, ID_EX_Op1, Opcode, FunctionCode;
        reg [1:0] RegWrite;
        reg ID_EX_MemRead, EX_MEM_MemRead, ExcepSig, branch;

        wire [1:0] R15_Sel;
        wire Op1_Sel, IF_ID_Hold, PC_Hold, Delay;
```

```
	initial
		$vcdpluson;

	initial
		$monitor($time, "  R15_Sel = %b Op1_Sel = %b IF_ID_Hold = %b PC_Hold = %b Delay = %b \n", R15_Sel[1:0],
Op1_Sel, IF_ID_Hold, PC_Hold, Delay);

	Hazard_Detect dut(.IF_ID_Op1(IF_ID_Op1), .IF_ID_Op2(IF_ID_Op2), .ID_EX_Op1(ID_EX_Op1), .Opcode(Opcode),
.FunctionCode(FunctionCode),
			.RegWrite(RegWrite), .ID_EX_MemRead(ID_EX_MemRead),
.EX_MEM_MemRead(EX_MEM_MemRead), .ExcepSig(ExcepSig),
			.R15_Sel(R15_Sel), .Op1_Sel(Op1_Sel), .IF_ID_Hold(IF_ID_Hold), .PC_Hold(PC_Hold), .Delay(Delay),
.branch(branch));

	initial begin


		#10 IF_ID_Op1 = 4'b0101; ID_EX_Op1 = 4'b1000; ID_EX_Op1 = 4'b0000; Opcode = 4'b0000; FunctionCode =
4'b0001; RegWrite = 2'b01;
		   ID_EX_MemRead = 1'b1; EX_MEM_MemRead = 1'b0; ExcepSig = 1'b0;


		#20;
	end

	initial
		#80 $finish;


endmodule
```

Instruction Memory Source Code

```
module Instr_Mem(input [15:0] pointer, input rst, clk, output reg [15:0] instr_out);

        reg [7:0] Memory [99:0];
        integer i;

        always@(posedge clk or negedge rst) begin

                if(!rst) begin

                        Memory[0] <= 8'h0E; Memory[1] <= 8'h20; Memory[2] <= 8'h0B; Memory[3] <= 8'h21;
                        Memory[4] <= 8'h23; Memory[5] <= 8'h88; Memory[6] <= 8'h14; Memory[7] <= 8'h9A;
                        Memory[8] <= 8'h05; Memory[9] <= 8'h64; Memory[10] <= 8'h01; Memory[11] <= 8'h65;
                        Memory[12] <= 8'hD5; Memory[13] <= 8'h9A; Memory[14] <= 8'h28; Memory[15] <= 8'h02;
                        Memory[16] <= 8'hCE; Memory[17] <= 8'h9A; Memory[18] <= 8'h0F; Memory[19] <= 8'hF1;
                        Memory[20] <= 8'h01; Memory[21] <= 8'h20; Memory[22] <= 8'h01; Memory[23] <= 8'h21;
                        Memory[24] <= 8'h18; Memory[25] <= 8'h02; Memory[26] <= 8'hA6; Memory[27] <= 8'h94;
                        Memory[28] <= 8'hB6; Memory[29] <= 8'h96; Memory[30] <= 8'hC6; Memory[31] <= 8'h96;
                        Memory[32] <= 8'h07; Memory[33] <= 8'hD1; Memory[34] <= 8'h67; Memory[35] <= 8'h04;
                        Memory[36] <= 8'h0B; Memory[37] <= 8'h10; Memory[38] <= 8'h57; Memory[39] <= 8'h05;
                        Memory[40] <= 8'h0B; Memory[41] <= 8'h20; Memory[42] <= 8'h47; Memory[43] <= 8'h02;
                        Memory[44] <= 8'h01; Memory[45] <= 8'h10; Memory[46] <= 8'h01; Memory[47] <= 8'h10;
                        Memory[48] <= 8'hC8; Memory[49] <= 8'h90; Memory[50] <= 8'h08; Memory[51] <= 8'h80;
                        Memory[52] <= 8'hD8; Memory[53] <= 8'h92; Memory[54] <= 8'hCA; Memory[55] <= 8'h92;
                        Memory[56] <= 8'h0C; Memory[57] <= 8'hC0; Memory[58] <= 8'h0D; Memory[59] <= 8'hD1;
                        Memory[60] <= 8'h0C; Memory[61] <= 8'hD0; Memory[62] <= 8'hF0; Memory[63] <= 8'h00;

                        for(i = 64; i <= 99; i = i+1) begin

                                Memory[i] <= 8'h00;
```

```
                end

            end

        end

        always@(*) begin

            instr_out = {Memory[pointer], Memory[pointer+1]};

        end

endmodule


`include "Instr_Mem.v"

module Instr_Mem__fixture;

        reg [15:0] pointer;
        reg clk, rst;

        wire [15:0] instr_out;


        initial
            $vcdpluson;


        initial
            $monitor($time, "  pointer = %d  clk = %b  rst = %b  instr_out = %h\n",
                pointer[15:0], clk, rst, instr_out[15:0]);
```

```
    Instr_Mem dut(.pointer(pointer), .rst(rst), .clk(clk), .instr_out(instr_out));

    initial begin

        rst = 1'b1; pointer = 0;
        #10 rst = 1'b0;
        #10 rst = 1'b1;
        #20 pointer = 2;
        #10 pointer = 4;
        #10 pointer = 6;
        #20 pointer = 62;


    end

    initial begin

        clk = 1'b0;
        forever #10 clk = ~clk;

    end

    initial begin

        #100 $finish;

    end

endmodule
```

Mux A Source Code

```
module mux_A (input [15:0] a, b, input s, output reg [15:0] c);

	always@(*) begin

		if (s)
			c = a;
		else
			c = b;
	end

endmodule


`include "mux_A.v"

module mux_A_fixture;

	reg [15:0] a, b;
	reg s;
	wire [15:0] c;

	initial
		 $vcdpluson;

	initial
		$monitor($time, "  a = %h b = %h s =  %b c = %h \n", a[15:0], b[15:0], s, c[15:0]);

	mux_A dut(.a(a), .b(b), .c(c), .s(s));

	initial begin
```

```
                a = 16'h4AA2;
                b = 16'hAF9C;
                s = 1'b1;

                #10

                s = 1'b0;

                #10;
        end
        initial
                #30 $finish;
endmodule
```

Mux B Source Code

```
module mux_B (input [15:0] a, b, c, input [1:0] s, output reg [15:0] d);

	always@(*) begin

		if (s == 2'b00)
			d = a;
		else if (s == 2'b01)
			d = b;
		else if (s == 2'b10)
			d = c;
		else
			d = 16'h0000;
	end

endmodule


`include "mux_B.v"

module mux_B_fixture;

	reg [15:0] a, b, c;
	reg [1:0] s;
	wire [15:0] d;

	initial $vcdpluson;

	initial
		$monitor($time, "  a = %h b = %h c = %h s = %b d = %h \n", a[15:0], b[15:0], c[15:0], s[1:0], d[15:0]);
```

```
    mux_B dut(.a(a), .b(b), .c(c), .s(s), .d(d));

    initial begin

        a = 16'h1234;
        b = 16'h4321;
        c = 16'hABCD;
        s = 2'b00;

        #10

        s = 2'b01;

        #10

        s = 2'b10;

        #10

        s = 2'b11;

        #10;
    end

    initial
        #60 $finish;

Endmodule
```

Mux C Source Code

```
module mux_C(input str_byte_in, UpperByteToZero_in, MemWrite_in, MemRead_in,
        MemtoReg_in, ALUSrc1_in, ALUSrc2_in, Delay,
        input [1:0] RegWrite_in, ALUOp_in, output reg str_byte_out,
        UpperByteToZero_out, MemWrite_out, MemRead_out, MemtoReg_out,
        ALUSrc1_out, ALUSrc2_out, output reg [1:0] RegWrite_out, ALUOp_out);

    always@(*) begin

        if(!Delay) begin

            str_byte_out = str_byte_in;
            UpperByteToZero_out = UpperByteToZero_in;
            MemWrite_out = MemWrite_in;
            MemRead_out = MemRead_in;
            MemtoReg_out = MemtoReg_in;
            ALUSrc1_out = ALUSrc1_in;
            ALUSrc2_out = ALUSrc2_in;
            RegWrite_out = RegWrite_in;
            ALUOp_out = ALUOp_in;

        end

        else begin

            str_byte_out = 1'b0;
            UpperByteToZero_out = 1'b0;
            MemWrite_out = 1'b0;
            MemRead_out = 1'b0;
            MemtoReg_out = 1'b0;
            ALUSrc1_out = 1'b0;
```

```
                ALUSrc2_out = 1'b0;
                RegWrite_out = 2'b0;
                ALUOp_out = 2'b0;


            end

        end

endmodule

`include "mux_C.v"

module mux_C_fixture;


        reg str_byte_in, UpperByteToZero_in, MemWrite_in, MemRead_in, MemtoReg_in, ALUSrc1_in, ALUSrc2_in, Delay;
        reg [1:0] RegWrite_in, ALUOp_in;
        wire str_byte_out, UpperByteToZero_out, MemWrite_out, MemRead_out, MemtoReg_out, ALUSrc1_out, ALUSrc2_out;
        wire [1:0] RegWrite_out, ALUOp_out;

        initial
            $vcdpluson;

        initial begin
            $display($time, "  str_byte_in = %b UpperByteToZero_in = %b MemWrite_in = %b MemRead_in = %b \n",
str_byte_in, UpperByteToZero_in, MemWrite_in, MemRead_in);
            $display( "  MemtoReg_in = %b ALUSrc1_in = %b ALUSrc2_in = %b Delay = %b \n", MemtoReg_in, ALUSrc1_in,
ALUSrc2_in, Delay);
            $display( "  RegWrite_in = %b ALUOp_in = %b str_byte_out = %b UpperByteToZero_out = %b \n",
RegWrite_in[1:0], ALUOp_in[1:0], str_byte_out, UpperByteToZero_out);
            $display("  MemWrite_out = %b MemRead_out = %b MemtoReg_out = %b ALUSrc1_out = %b \n", MemWrite_out,
```

```
MemRead_out, MemtoReg_out, ALUSrc1_out);
                $display("  ALUSrc2_out = %b RegWrite_out = %b ALUOp_out = %b \n", ALUSrc2_out, RegWrite_out[1:0],
ALUOp_out[1:0]);

        end

        mux_C dut(.str_byte_in(str_byte_in), .UpperByteToZero_in(UpperByteToZero_in), .MemWrite_in(MemWrite_in),
                .MemRead_in(MemRead_in), .MemtoReg_in(MemtoReg_in), .ALUSrc1_in(ALUSrc1_in),
.ALUSrc2_in(ALUSrc2_in),
                .Delay(Delay), .RegWrite_in(RegWrite_in), .ALUOp_in(ALUOp_in), .str_byte_out(str_byte_out),
.UpperByteToZero_out(UpperByteToZero_out), .MemWrite_out(MemWrite_out),
                .MemRead_out(MemRead_out), .MemtoReg_out(MemtoReg_out), .ALUSrc1_out(ALUSrc1_out),
.ALUSrc2_out(ALUSrc2_out), .RegWrite_out(RegWrite_out), .ALUOp_out(ALUOp_out));

        initial begin

                str_byte_in = 1'b1; UpperByteToZero_in = 1'b1; MemWrite_in = 1'b1; MemRead_in = 1'b1; MemtoReg_in = 1'b1;
ALUSrc1_in = 1'b1; ALUSrc2_in = 1'b1; Delay = 1'b1;
                RegWrite_in = 2'b01; ALUOp_in = 2'b10;


                #10

                Delay = 1'b0;

                #10;
        end

        initial
                #30 $finish;

endmodule
```

PC Source Code

```
module PC(input [15:0] PC_in, input halt_sig, clk, rst, output reg [15:0] PC_out);

	always@(posedge clk, negedge rst) begin

		if(!rst)
			PC_out <= 0;

		else begin

			if(!halt_sig)
				PC_out <= PC_in;

		end

	end


endmodule


`include "PC.v"

module PC_fixture;

	reg [15:0] PC_in;
	reg halt_sig, clk, rst;
	wire [15:0] PC_out;

	initial
		 $vcdpluson;
```

```
	initial
		$monitor($time, "  PC_in = %d  clk = %b  rst = %b  halt_sig = %b  PC_out = %d \n", PC_in[15:0], clk, rst, halt_sig,
PC_out[15:0]);

	PC dut(.PC_in(PC_in), .clk(clk), .rst(rst), .halt_sig(halt_sig), .PC_out(PC_out));

	initial begin

		rst = 1'b0;
		PC_in = 16'h0004;
		halt_sig = 1'b0;

		#10 rst = 1'b1;
		#20;

		#10 halt_sig = 1'b1;

		#10 PC_in = 16'h000A;
		#20 halt_sig = 1'b0;
		#10 rst = 1'b0;
	end

	initial begin
		clk = 1'b0;
		forever #10 clk = ~clk;
	end

	initial
		#100 $finish;

endmodule
```

Register File Source Code

```
module RegFile (input [3:0] Reg1, Reg2, WriteReg, input [1:0] RegWrite,
                input [15:0] WriteR15, WriteData, input clk, rst,
                  output reg [15:0] R_D1, R_D2, R_R15);

        reg [15:0] Register [15:0];


        always@(posedge clk or negedge rst) begin

                if(!rst) begin

                        Register[0] <= 16'h0000;
                        Register[1] <= 16'h7B18;
                        Register[2] <= 16'h245B;
                        Register[3] <= 16'hFF0F;
                        Register[4] <= 16'hF0FF;
                        Register[5] <= 16'h0051;
                        Register[6] <= 16'h6666;
                        Register[7] <= 16'h00FF;
                        Register[8] <= 16'hFF88;
                        Register[9] <= 16'h0000;
                        Register[10] <= 16'h0000;
                        Register[11] <= 16'h3099;
                        Register[12] <= 16'hCCCC;
                        Register[13] <= 16'h0002;
                        Register[14] <= 16'h0011;
                        Register[15] <= 16'h0000;

                end
```

```
            else if(RegWrite == 2'b01)
                Register[WriteReg] = WriteData;

            else if(RegWrite == 2'b10) begin
                Register[WriteReg] = WriteData;
                Register[15] = WriteR15;
            end

        end


        always@(*) begin

            R_D1 = Register[Reg1];
            R_D2 = Register[Reg2];
            R_R15 = Register[15];

        end

endmodule


`include "RegFile.v"

module RegFile_fixture;

        reg [15:0] WriteData, WriteR15;
        reg [3:0] Reg1, Reg2, WriteReg;
        reg rst, clk;
        reg [1:0] RegWrite;
```

```
	wire [15:0] R_D1, R_D2, R_R15;


	initial
		$vcdpluson;


	initial
		$monitor($time, "  Reg1 = %b  Reg2 = %b  WriteData = %h  WriteR15 = %h  WriteReg = %b  RegWrite = %b  rst =
%b  clk = %b  R_D1 = %h  R_D2 = %h  R_R15 = %h \n\n",
			Reg1[3:0], Reg2[3:0], WriteData[15:0], WriteR15[15:0],
			WriteReg[3:0], RegWrite[1:0], rst, clk, R_D1[15:0], R_D2[15:0], R_R15[15:0]);

	RegFile dut(.Reg1(Reg1), .Reg2(Reg2), .WriteReg(WriteReg), .WriteData(WriteData),
		.WriteR15(WriteR15), .RegWrite(RegWrite), .rst(rst), .clk(clk),
		.R_D1(R_D1), .R_D2(R_D2), .R_R15(R_R15));

	initial begin

		rst = 1'b1; Reg1 = 4'b0001; Reg2 = 4'b0010;
		WriteData = 0; WriteR15 = 0; RegWrite = 0; WriteReg = 4'b0001;
		#10 rst = 1'b0;
		#20 rst = 1'b1;
		WriteData = 16'hABCD; WriteR15 = 16'hB45C;
		#10 RegWrite = 2'b01;
		#20 RegWrite = 2'b00; WriteData = 16'h0000;
		#10 RegWrite = 2'b10;
		#10 RegWrite = 2'b00;
		#10 rst = 1'b0;

	end
```

```
        initial begin

                clk = 1'b0;
                forever #10 clk = ~clk;

        end

        initial begin

                #120 $finish;

        end

endmodule
```

Zero Extend Source Code

```
module zeroExtend(input [15:0] d_in, input enable, output reg [15:0] d_out);

	always@(*) begin

		if(enable)
			d_out = {8'b00000000, d_in[7:0]};
		else
			d_out = d_in;

	end

endmodule


`include "zeroExtend.v"

module zeroExtend_fixture;

	reg [15:0] d_in;
	reg enable;
	wire [15:0] d_out;

	initial
		 $vcdpluson;

	initial
		$monitor($time, "  d_in = %h enable = %b d_out = %h \n", d_in[15:0], enable, d_out[15:0]);

	zeroExtend dut(.d_in(d_in), .d_out(d_out), .enable(enable));
```

```
        initial begin
                d_in = 16'h4AA2;
                enable= 1'b1;

                #10

                enable = 1'b0;

                #10;
        end

        initial
                #30 $finish;

endmodule
```

CPU Simulation Result

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 4 15:03 2019
VCD+ Writer O-2018.09-SP2-3_Full64 Copyright (c) 1991-2018 by Synopsys Inc.

clk = 0 rst = 0
////////////// IF Stage //////////////

IF/ID_BufferData: PC = xxxx Instr = xxxx Adder = xxxx

////////////// ID Stage //////////////

Main Control Signals: str_byte = x ALUSrc1 = x ALUSrc2 = x MemRead = x MemWrite = x MemtoReg = x UpperByteToZero = x TargetAddress = x IF/ID_Flush = x ExcepSig = x ALUOp = xx Comparison = xx RegWrite = xx, SignExtend = xx branch = x

Hazard Detection Signals: PC_Hold = x IF/ID_Hold = x Delay = x Op1_Sel = x R15_Sel = xx

Comparator Ports: OP1 = xxxx R15 = xxxx Branch_Result = xx

ID/EX_BufferData: Reg1_Data = xxxx Reg2_Data = xxxx Sign Extended Value = xxxx FunctionCode = xxxx Op1_RegID = xxxx Op2_RegID = xxxx

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = x IF/ID_Hold = x Delay = x Op1_Sel = x R15_Sel = xx

ALU_OverflowSig = x
EX/MEM_BufferData: ALU_Result = xxxx R15_Result = xxxx Op1_RegData = xxxx Op1_RegID = xxxx str_byte = x UpperByteToZero = x MemWrite = x MemRead = x MemToReg = x RegWrite = xx

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = x MemToReg = x RegWrite = xx Mem_Data = xxxx ALU_Result = xxxx R15_Result = xxxx Op1_RegID = xxxx

//////////WB Stage //////////

WriteData = xxxx WriteReg = xxxx

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0002 Instr = 0e20 Adder = 0002

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 1 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 00 Comparison = xx RegWrite = 01, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 0011 R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 0000 Reg2_Data = 0000 Sign Extended Value = 0000 FunctionCode = 0000 Op1_RegID = 0000 Op2_RegID = 0000

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 0000 R15_Result = xxxx Op1_RegData = 0000 Op1_RegID = 0000 str_byte = 0
UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 00

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 00 Mem_Data = xxxx ALU_Result = 0000 R15_Result = 0000 Op1_RegID = 0000

//////////WB Stage //////////

WriteData = 0000 WriteReg = 0000

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0004 Instr = 0b21 Adder = 0004

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 1 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 00 Comparison = xx RegWrite = 01, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 3099 R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 0011 Reg2_Data = 245b Sign Extended Value = fe20 FunctionCode = 0000 Op1_RegID = 1110 Op2_RegID = 0010

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 0000 R15_Result = xxxx Op1_RegData = 0000 Op1_RegID = 0000 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 00 Mem_Data = xxxx ALU_Result = 0000 R15_Result = xxxx Op1_RegID = 0000

//////////WB Stage //////////

WriteData = 0000 WriteReg = 0000

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0006 Instr = 2388 Adder = 0006

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 0 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 11 Comparison = xx RegWrite = 01, SignExtend = 11 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = ff0f R15 = 0000 Branch_Result = 01

ID/EX_BufferData: Reg1_Data = 3099 Reg2_Data = 245b Sign Extended Value = fb21 FunctionCode = 0001 Op1_RegID = 1011 Op2_RegID = 0010

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 246c R15_Result = xxxx Op1_RegData = 0011 Op1_RegID = 1110 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 01 Mem_Data = xxxx ALU_Result = 0000 R15_Result = xxxx Op1_RegID = 0000

/////////WB Stage /////////

WriteData = 0000 WriteReg = 0000

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0008 Instr = 149a Adder = 0008

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 0 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 10 Comparison = xx RegWrite = 01, SignExtend = 11 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = f0ff R15 = 0000 Branch_Result = 01

ID/EX_BufferData: Reg1_Data = ff0f Reg2_Data = ff88 Sign Extended Value = 0088 FunctionCode = 1000 Op1_RegID = 0011 Op2_RegID = 1000

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 0c3e R15_Result = xxxx Op1_RegData = 3099 Op1_RegID = 1011 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 01 Mem_Data = xxxx ALU_Result = 246c R15_Result = xxxx Op1_RegID = 1110

//////////WB Stage //////////

WriteData = 246c WriteReg = 1110

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 000a Instr = 0564 Adder = 000a

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 1 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 00 Comparison = xx RegWrite = 10, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 0051 R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = f0ff Reg2_Data = 0000 Sign Extended Value = 009a FunctionCode = 1010 Op1_RegID = 0100 Op2_RegID = 1001

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = ff8f R15_Result = xxxx Op1_RegData = ff0f Op1_RegID = 0011 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 01 Mem_Data = xxxx ALU_Result = 0c3e R15_Result = xxxx Op1_RegID = 1011

//////////WB Stage //////////

WriteData = 0c3e WriteReg = 1011

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 000c Instr = 0165 Adder = 000c

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 1 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 00 Comparison = xx RegWrite = 10, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 7b18 R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 0051 Reg2_Data = 6666 Sign Extended Value = 0564 FunctionCode = 0100 Op1_RegID = 0101 Op2_RegID = 0110

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 009a R15_Result = xxxx Op1_RegData = f0ff Op1_RegID = 0100 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 01 Mem_Data = xxxx ALU_Result = ff8f R15_Result = xxxx Op1_RegID = 0011

//////////WB Stage //////////

WriteData = ff8f WriteReg = 0011

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 000e Instr = d59a Adder = 000e

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 0 ALUSrc2 = 0 MemRead = 0 MemWrite = 1 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 01 Comparison = xx RegWrite = 00, SignExtend = 10 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 0051 R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 7b18 Reg2_Data = 6666 Sign Extended Value = 0165 FunctionCode = 0101 Op1_RegID = 0001 Op2_RegID = 0110

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 6646 R15_Result = 0020 Op1_RegData = 0051 Op1_RegID = 0101 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 10

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 01 Mem_Data = xxxx ALU_Result = 009a R15_Result = xxxx Op1_RegID = 0100

//////////WB Stage //////////

WriteData = 009a WriteReg = 0100

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0010 Instr = 2802 Adder = 0010

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 0 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0

TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 11 Comparison = xx RegWrite = 01, SignExtend = 11 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = ff88 R15 = 0000 Branch_Result = 01

ID/EX_BufferData: Reg1_Data = 0051 Reg2_Data = 0000 Sign Extended Value = fffa FunctionCode = 1010 Op1_RegID = 0101 Op2_RegID = 1001

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 1
EX/MEM_BufferData: ALU_Result = 0001 R15_Result = 14b2 Op1_RegData = 7b18 Op1_RegID = 0001 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 10

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 10 Mem_Data = xxxx ALU_Result = 6646 R15_Result = 0020 Op1_RegID = 0101

//////////WB Stage //////////

WriteData = 6646 WriteReg = 0101

Arithmetic Overflow at time 160.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0012 Instr = ce9a Adder = 0012

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 0 ALUSrc2 = 0 MemRead = 1 MemWrite = 0 MemtoReg = 1 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 01 Comparison = xx RegWrite = 01, SignExtend = 10 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 246c R15 = 0020 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = ff88 Reg2_Data = 0000 Sign Extended Value = 0002 FunctionCode = 0010 Op1_RegID = 1000 Op2_RegID = 0000

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = fffa R15_Result = 14b2 Op1_RegData = 6646 Op1_RegID = 0101 str_byte = 0 UpperByteToZero = 0 MemWrite = 1 MemRead = 0 MemToReg = 0 RegWrite = 00

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 10 Mem_Data = xxxx ALU_Result = 0001 R15_Result = 14b2 Op1_RegID = 0001

/////////WB Stage /////////

WriteData = 0001 WriteReg = 0001

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0014 Instr = 0ff1 Adder = 0014

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 1 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 00 Comparison = xx RegWrite = 01, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 14b2 R15 = 14b2 Branch_Result = 11

ID/EX_BufferData: Reg1_Data = 246c Reg2_Data = 0000 Sign Extended Value = fffa FunctionCode = 1010 Op1_RegID = 1110 Op2_RegID = 1001

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 1
EX/MEM_BufferData: ALU_Result = ff8a R15_Result = 14b2 Op1_RegData = ff88 Op1_RegID = 1000 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 00 Mem_Data = xxxx ALU_Result = fffa R15_Result =

14b2 Op1_RegID = 0101

//////////WB Stage //////////

WriteData = fffa WriteReg = 0101

Arithmetic Overflow at time 200.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0016 Instr = 0120 Adder = 0016

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 1 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 00 Comparison = xx RegWrite = 01, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 0001 R15 = 14b2 Branch_Result = 01

ID/EX_BufferData: Reg1_Data = 14b2 Reg2_Data = 14b2 Sign Extended Value = fff1 FunctionCode = 0001 Op1_RegID = 1111 Op2_RegID = 1111

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0

EX/MEM_BufferData: ALU_Result = fffa R15_Result = 14b2 Op1_RegData = 246c Op1_RegID = 1110 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 1 MemToReg = 1 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 01 Mem_Data = xxxx ALU_Result = ff8a R15_Result = 14b2 Op1_RegID = 1000

//////////WB Stage //////////

WriteData = ff8a WriteReg = 1000

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0018 Instr = 0121 Adder = 0018

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 1 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 00 Comparison = xx RegWrite = 01, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 0001 R15 = 14b2 Branch_Result = 01

ID/EX_BufferData: Reg1_Data = 0001 Reg2_Data = 245b Sign Extended Value = 0120 FunctionCode = 0000 Op1_RegID = 0001 Op2_RegID = 0010

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 0000 R15_Result = 14b2 Op1_RegData = 14b2 Op1_RegID = 1111 str_byte = 0
UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 1 RegWrite = 01 Mem_Data = 6646 ALU_Result = fffa R15_Result = 14b2 Op1_RegID = 1110

//////////WB Stage //////////

WriteData = 6646 WriteReg = 1110

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 001a Instr = 1802 Adder = 001a

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 0 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0
TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 10 Comparison = xx RegWrite = 01, SignExtend = 11 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = ff8a R15 = 14b2 Branch_Result = 01

ID/EX_BufferData: Reg1_Data = 0001 Reg2_Data = 245b Sign Extended Value = 0121 FunctionCode = 0001 Op1_RegID = 0001 Op2_RegID = 0010

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 245c R15_Result = 14b2 Op1_RegData = 0001 Op1_RegID = 0001 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 01 Mem_Data = 6646 ALU_Result = 0000 R15_Result = 14b2 Op1_RegID = 1111

//////////WB Stage //////////

WriteData = 0000 WriteReg = 1111

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 001c Instr = a694 Adder = 001c

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 0 ALUSrc2 = 0 MemRead = 1 MemWrite = 0 MemtoReg = 1 UpperByteToZero = 1 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 01 Comparison = xx RegWrite = 01, SignExtend = 10 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 6666 R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = ff8a Reg2_Data = 0000 Sign Extended Value = 0002 FunctionCode = 0010 Op1_RegID = 1000 Op2_RegID = 0000

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 0001 R15_Result = 14b2 Op1_RegData = 245c Op1_RegID = 0001 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 01 Mem_Data = 6646 ALU_Result = 245c R15_Result = 14b2 Op1_RegID = 0001

/////////WB Stage /////////

WriteData = 245c WriteReg = 0001

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 001e Instr = b696 Adder = 001e

////////////// ID Stage //////////////

Main Control Signals: str_byte = 1 ALUSrc1 = 0 ALUSrc2 = 0 MemRead = 0 MemWrite = 1 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 01 Comparison = xx RegWrite = 00, SignExtend = 10 branch = 0

Hazard Detection Signals: PC_Hold = 1 IF/ID_Hold = 0 Delay = 1 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 6666 R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 6666 Reg2_Data = 0000 Sign Extended Value = 0004 FunctionCode = 0100 Op1_RegID = 0110 Op2_RegID = 1001

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 1 IF/ID_Hold = 0 Delay = 1 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 0002 R15_Result = 14b2 Op1_RegData = ff8a Op1_RegID = 1000 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 01 Mem_Data = 6646 ALU_Result = 0001 R15_Result = 14b2 Op1_RegID = 0001

//////////WB Stage //////////

WriteData = 0001 WriteReg = 0001

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 001e Instr = b696 Adder = 001e

////////////// ID Stage //////////////

Main Control Signals: str_byte = 1 ALUSrc1 = 0 ALUSrc2 = 0 MemRead = 0 MemWrite = 1 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 01 Comparison = xx RegWrite = 00, SignExtend = 10 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 6666 R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 6666 Reg2_Data = 0000 Sign Extended Value = 0006 FunctionCode = 0110 Op1_RegID = 0110 Op2_RegID = 1001

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 0004 R15_Result = 14b2 Op1_RegData = 6666 Op1_RegID = 0110 str_byte = 0 UpperByteToZero = 1 MemWrite = 0 MemRead = 1 MemToReg = 1 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 01 Mem_Data = 6646 ALU_Result = 0002 R15_Result = 14b2 Op1_RegID = 1000

/////////WB Stage /////////

WriteData = 0002 WriteReg = 1000

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0020 Instr = c696 Adder = 0020

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 0 ALUSrc2 = 0 MemRead = 1 MemWrite = 0 MemtoReg = 1 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 01 Comparison = xx RegWrite = 01, SignExtend = 10 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 6666 R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 6666 Reg2_Data = 0000 Sign Extended Value = 0006 FunctionCode = 0110 Op1_RegID = 0110 Op2_RegID = 1001

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 0006 R15_Result = 14b2 Op1_RegData = 0004 Op1_RegID = 0110 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 00

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 1 MemToReg = 1 RegWrite = 01 Mem_Data = 5678 ALU_Result = 0004 R15_Result = 14b2 Op1_RegID = 0110

//////////WB Stage //////////

WriteData = 0078 WriteReg = 0110

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0022 Instr = 07d1 Adder = 0022

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 1 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 00 Comparison = xx RegWrite = 01, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 00ff R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 6666 Reg2_Data = 0000 Sign Extended Value = 0006 FunctionCode = 0110 Op1_RegID = 0110 Op2_RegID = 1001

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 0006 R15_Result = 14b2 Op1_RegData = 0078 Op1_RegID = 0110 str_byte = 1 UpperByteToZero = 0 MemWrite = 1 MemRead = 0 MemToReg = 0 RegWrite = 00

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 00 Mem_Data = 5678 ALU_Result = 0006 R15_Result = 14b2 Op1_RegID = 0110

//////////WB Stage //////////

WriteData = 0006 WriteReg = 0110

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0024 Instr = 6704 Adder = 0024

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 0 ALUSrc2 = 0 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = xx Comparison = 00 RegWrite = 00, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 1 IF/ID_Hold = 0 Delay = 1 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 00ff R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 00ff Reg2_Data = 0002 Sign Extended Value = 07d1 FunctionCode = 0001 Op1_RegID = 0111 Op2_RegID = 1101

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 1 IF/ID_Hold = 0 Delay = 1 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 0006 R15_Result = 14b2 Op1_RegData = 6666 Op1_RegID = 0110 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 1 MemToReg = 1 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 00 Mem_Data = 5678 ALU_Result = 0006 R15_Result = 14b2 Op1_RegID = 0110

//////////WB Stage //////////

WriteData = 0006 WriteReg = 0110

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0024 Instr = 6704 Adder = 0024

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 0 ALUSrc2 = 0 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = xx Comparison = 00 RegWrite = 00, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 00ff R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 00ff Reg2_Data = 0000 Sign Extended Value = 0704 FunctionCode = 0100 Op1_RegID = 0111 Op2_RegID = 0000

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 00fd R15_Result = 14b2 Op1_RegData = 00ff Op1_RegID = 0111 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 1 RegWrite = 01 Mem_Data = 78ad ALU_Result = 0006 R15_Result = 14b2 Op1_RegID = 0110

//////////WB Stage //////////

WriteData = 78ad WriteReg = 0110

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0026 Instr = 0b10 Adder = 0026

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 1 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 00 Comparison = xx RegWrite = 01, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 0c3e R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 00ff Reg2_Data = 0000 Sign Extended Value = 0704 FunctionCode = 0100 Op1_RegID = 0111 Op2_RegID = 0000

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 0000 R15_Result = 0000 Op1_RegData = 00fd Op1_RegID = 0111 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 00

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 01 Mem_Data = 78ad ALU_Result = 00fd R15_Result = 14b2 Op1_RegID = 0111

//////////WB Stage //////////

WriteData = 00fd WriteReg = 0111

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0028 Instr = 5705 Adder = 0028

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 0 ALUSrc2 = 0 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = xx Comparison = 00 RegWrite = 00, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 00fd R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 0c3e Reg2_Data = 0001 Sign Extended Value = fb10 FunctionCode = 0000 Op1_RegID = 1011 Op2_RegID = 0001

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 0000 R15_Result = 0000 Op1_RegData = 00fd Op1_RegID = 0111 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 00

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 00 Mem_Data = 78ad ALU_Result = 0000 R15_Result = 0000 Op1_RegID = 0111

//////////WB Stage //////////

WriteData = 0000 WriteReg = 0111

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 002a Instr = 0b20 Adder = 002a

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 1 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 00 Comparison = xx RegWrite = 01, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 0c3e R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 00fd Reg2_Data = 0000 Sign Extended Value = 0705 FunctionCode = 0101 Op1_RegID = 0111 Op2_RegID = 0000

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 0c3f R15_Result = 0000 Op1_RegData = 0c3e Op1_RegID = 1011 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 00 Mem_Data = 78ad ALU_Result = 0000 R15_Result = 0000 Op1_RegID = 0111

//////////WB Stage //////////

WriteData = 0000 WriteReg = 0111

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 002c Instr = 4702 Adder = 002c

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 0 ALUSrc2 = 0 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0

TargetAddress = 0 IF/ID_Flush = 0 ExcepSig = 0 ALUOp = xx Comparison = 01 RegWrite = 00, SignExtend = 01 branch = 1

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 1 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 00fd R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 0c3e Reg2_Data = 245b Sign Extended Value = fb20 FunctionCode = 0000 Op1_RegID = 1011 Op2_RegID = 0010

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 1 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 0000 R15_Result = 0000 Op1_RegData = 00fd Op1_RegID = 0111 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 00

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 01 Mem_Data = 78ad ALU_Result = 0c3f R15_Result = 0000 Op1_RegID = 1011

//////////WB Stage //////////

WriteData = 0c3f WriteReg = 1011

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0030 Instr = 0110 Adder = 002e

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 1 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 00 Comparison = xx RegWrite = 01, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 0001 R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 00fd Reg2_Data = 0000 Sign Extended Value = 0002 FunctionCode = 0010 Op1_RegID = 0111 Op2_RegID = 0000

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 309a R15_Result = 0000 Op1_RegData = 0c3f Op1_RegID = 1011 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 00 Mem_Data = 78ad ALU_Result = 0000 R15_Result = 0000 Op1_RegID = 0111

/////////WB Stage /////////

WriteData = 0000 WriteReg = 0111

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0032 Instr = c890 Adder = 0032

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 0 ALUSrc2 = 0 MemRead = 1 MemWrite = 0 MemtoReg = 1 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 01 Comparison = xx RegWrite = 01, SignExtend = 10 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 0002 R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 0001 Reg2_Data = 0001 Sign Extended Value = 0110 FunctionCode = 0000 Op1_RegID = 0001 Op2_RegID = 0001

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 0002 R15_Result = 0000 Op1_RegData = 00fd Op1_RegID = 0111 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 00

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 01 Mem_Data = 78ad ALU_Result = 309a R15_Result =

0000 Op1_RegID = 1011

//////////WB Stage //////////

WriteData = 309a WriteReg = 1011

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0034 Instr = 0880 Adder = 0034

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 1 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 00 Comparison = xx RegWrite = 01, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 1 IF/ID_Hold = 0 Delay = 1 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 0002 R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 0002 Reg2_Data = 0000 Sign Extended Value = 0000 FunctionCode = 0000 Op1_RegID = 1000 Op2_RegID = 1001

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 1 IF/ID_Hold = 0 Delay = 1 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0

EX/MEM_BufferData: ALU_Result = 0002 R15_Result = 0000 Op1_RegData = 0001 Op1_RegID = 0001 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 00 Mem_Data = 78ad ALU_Result = 0002 R15_Result = 0000 Op1_RegID = 0111

//////////WB Stage //////////

WriteData = 0002 WriteReg = 0111

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0034 Instr = 0880 Adder = 0034

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 1 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 00 Comparison = xx RegWrite = 01, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 0002 R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 0002 Reg2_Data = 0002 Sign Extended Value = f880 FunctionCode = 0000 Op1_RegID = 1000 Op2_RegID = 1000

////////////// EX Stage //////////////

 Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 1
 EX/MEM_BufferData: ALU_Result = 0000 R15_Result = 0000 Op1_RegData = 0002 Op1_RegID = 1000 str_byte = 0
UpperByteToZero = 0 MemWrite = 0 MemRead = 1 MemToReg = 1 RegWrite = 01

 ////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 01 Mem_Data = 78ad ALU_Result = 0002 R15_Result = 0000 Op1_RegID = 0001

 /////////WB Stage //////////

 WriteData = 0002 WriteReg = 0001

Arithmetic Overflow at time   560.

clk = 0 rst = 1
 ////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0036 Instr = d892 Adder = 0036

 ////////////// ID Stage //////////////

 Main Control Signals: str_byte = 0 ALUSrc1 = 0 ALUSrc2 = 0 MemRead = 0 MemWrite = 1 MemtoReg = 0 UpperByte ToZero = 0
TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 01 Comparison = xx RegWrite = 00, SignExtend = 10 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 0002 R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = 0002 Reg2_Data = 0002 Sign Extended Value = f880 FunctionCode = 0000 Op1_RegID = 1000 Op2_RegID = 1000

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = f880 R15_Result = 0000 Op1_RegData = 0000 Op1_RegID = 1000 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 00

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 1 RegWrite = 01 Mem_Data = 3142 ALU_Result = 0000 R15_Result = 0000 Op1_RegID = 1000

//////////WB Stage //////////

WriteData = 3142 WriteReg = 1000

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0038 Instr = ca92 Adder = 0038

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 0 ALUSrc2 = 0 MemRead = 1 MemWrite = 0 MemtoReg = 1 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 01 Comparison = xx RegWrite = 01, SignExtend = 10 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 0000 R15 = 0000 Branch_Result = 11

ID/EX_BufferData: Reg1_Data = 0002 Reg2_Data = 0000 Sign Extended Value = 0002 FunctionCode = 0010 Op1_RegID = 1000 Op2_RegID = 1001

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 6284 R15_Result = 0000 Op1_RegData = 3142 Op1_RegID = 1000 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 00 Mem_Data = 3142 ALU_Result = f880 R15_Result = 0000 Op1_RegID = 1000

/////////WB Stage /////////

WriteData = f880 WriteReg = 1000

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 003a Instr = 0cc0 Adder = 003a

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 1 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 00 Comparison = xx RegWrite = 01, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = cccc R15 = 0000 Branch_Result = 01

ID/EX_BufferData: Reg1_Data = 0000 Reg2_Data = 0000 Sign Extended Value = 0002 FunctionCode = 0010 Op1_RegID = 1010 Op2_RegID = 1001

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 0002 R15_Result = 0000 Op1_RegData = 6284 Op1_RegID = 1000 str_byte = 0 UpperByteToZero = 0 MemWrite = 1 MemRead = 0 MemToReg = 0 RegWrite = 00

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 01 Mem_Data = 3142 ALU_Result = 6284 R15_Result = 0000 Op1_RegID = 1000

//////////WB Stage //////////

WriteData = 6284 WriteReg = 1000

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 003c Instr = 0dd1 Adder = 003c

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 1 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 00 Comparison = xx RegWrite = 01, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 0002 R15 = 0000 Branch_Result = 10

ID/EX_BufferData: Reg1_Data = cccc Reg2_Data = cccc Sign Extended Value = fcc0 FunctionCode = 0000 Op1_RegID = 1100 Op2_RegID = 1100

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 1
EX/MEM_BufferData: ALU_Result = 0002 R15_Result = 0000 Op1_RegData = 0000 Op1_RegID = 1010 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 1 MemToReg = 1 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 00 Mem_Data = 3142 ALU_Result = 0002 R15_Result = 0000 Op1_RegID = 1000

//////////WB Stage //////////

WriteData = 0002 WriteReg = 1000

Arithmetic Overflow at time 640.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 003e Instr = 0cd0 Adder = 003e

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 1 ALUSrc2 = 1 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 1 IF/ID_Flush = 1 ExcepSig = 0 ALUOp = 00 Comparison = xx RegWrite = 01, SignExtend = 00 branch = 0

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = cccc R15 = 0000 Branch_Result = 01

ID/EX_BufferData: Reg1_Data = 0002 Reg2_Data = 0002 Sign Extended Value = fdd1 FunctionCode = 0001 Op1_RegID = 1101 Op2_RegID = 1101

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 0 IF/ID_Hold = 1 Delay = 0 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 9998 R15_Result = 0000 Op1_RegData = cccc Op1_RegID = 1100 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 1 RegWrite = 01 Mem_Data = 6284 ALU_Result = 0002 R15_Result = 0000 Op1_RegID = 1010

//////////WB Stage //////////

WriteData = 6284 WriteReg = 1010

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0040 Instr = f000 Adder = 0040

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 0 ALUSrc2 = 0 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 0 IF/ID_Flush = 0 ExcepSig = 0 ALUOp = xx Comparison = xx RegWrite = 00, SignExtend = xx branch = 0

Hazard Detection Signals: PC_Hold = 1 IF/ID_Hold = 0 Delay = 1 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 0000 R15 = 0000 Branch_Result = 11

ID/EX_BufferData: Reg1_Data = cccc Reg2_Data = 0002 Sign Extended Value = fcd0 FunctionCode = 0000 Op1_RegID = 1100 Op2_RegID = 1101

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 1 IF/ID_Hold = 0 Delay = 1 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 1
EX/MEM_BufferData: ALU_Result = 0000 R15_Result = 0000 Op1_RegData = 0002 Op1_RegID = 1101 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 01 Mem_Data = 6284 ALU_Result = 9998 R15_Result = 0000 Op1_RegID = 1100

//////////WB Stage //////////

WriteData = 9998 WriteReg = 1100

Arithmetic Overflow at time 680.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0040 Instr = f000 Adder = 0040

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 0 ALUSrc2 = 0 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 0 IF/ID_Flush = 0 ExcepSig = 0 ALUOp = xx Comparison = xx RegWrite = 00, SignExtend = xx branch = 0

Hazard Detection Signals: PC_Hold = 1 IF/ID_Hold = 0 Delay = 1 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 0000 R15 = 0000 Branch_Result = 11

ID/EX_BufferData: Reg1_Data = 0000 Reg2_Data = 0000 Sign Extended Value = 0000 FunctionCode = 0000 Op1_RegID = 0000 Op2_RegID = 0000

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 1 IF/ID_Hold = 0 Delay = 1 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 9998 R15_Result = 0000 Op1_RegData = 9998 Op1_RegID = 1100 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 01

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 01 Mem_Data = 6284 ALU_Result = 0000 R15_Result = 0000 Op1_RegID = 1101

/////////WB Stage /////////

WriteData = 0000 WriteReg = 1101

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0040 Instr = f000 Adder = 0040

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 0 ALUSrc2 = 0 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 0 IF/ID_Flush = 0 ExcepSig = 0 ALUOp = xx Comparison = xx RegWrite = 00, SignExtend = xx branch = 0

Hazard Detection Signals: PC_Hold = 1 IF/ID_Hold = 0 Delay = 1 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 0000 R15 = 0000 Branch_Result = 11

ID/EX_BufferData: Reg1_Data = 0000 Reg2_Data = 0000 Sign Extended Value = 0000 FunctionCode = 0000 Op1_RegID = 0000 Op2_RegID = 0000

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 1 IF/ID_Hold = 0 Delay = 1 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 0000 R15_Result = 0000 Op1_RegData = 0000 Op1_RegID = 0000 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 00

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 01 Mem_Data = 6284 ALU_Result = 9998 R15_Result = 0000 Op1_RegID = 1100

//////////WB Stage //////////

WriteData = 9998 WriteReg = 1100

No Exceptions have occurred.

clk = 0 rst = 1
////////////// IF Stage //////////////

IF/ID_BufferData: PC = 0040 Instr = f000 Adder = 0040

////////////// ID Stage //////////////

Main Control Signals: str_byte = 0 ALUSrc1 = 0 ALUSrc2 = 0 MemRead = 0 MemWrite = 0 MemtoReg = 0 UpperByteToZero = 0 TargetAddress = 0 IF/ID_Flush = 0 ExcepSig = 0 ALUOp = xx Comparison = xx RegWrite = 00, SignExtend = xx branch = 0

Hazard Detection Signals: PC_Hold = 1 IF/ID_Hold = 0 Delay = 1 Op1_Sel = 0 R15_Sel = 00

Comparator Ports: OP1 = 0000 R15 = 0000 Branch_Result = 11

ID/EX_BufferData: Reg1_Data = 0000 Reg2_Data = 0000 Sign Extended Value = 0000 FunctionCode = 0000 Op1_RegID = 0000 Op2_RegID = 0000

////////////// EX Stage //////////////

Hazard Detection Signals: PC_Hold = 1 IF/ID_Hold = 0 Delay = 1 Op1_Sel = 0 R15_Sel = 00

ALU_OverflowSig = 0
EX/MEM_BufferData: ALU_Result = 0000 R15_Result = 0000 Op1_RegData = 0000 Op1_RegID = 0000 str_byte = 0 UpperByteToZero = 0 MemWrite = 0 MemRead = 0 MemToReg = 0 RegWrite = 00

////////////// MEM Stage //////////////

MEM/WB_BufferData: UpperByteToZero = 0 MemToReg = 0 RegWrite = 00 Mem_Data = 6284 ALU_Result = 0000 R15_Result = 0000 Op1_RegID = 0000

 //////////WB Stage //////////

 WriteData = 0000 WriteReg = 0000

No Exceptions have occurred.

$finish called from file "cpu_fixture.v", line 41.
$finish at simulation time   750
         V C S   S i m u l a t i o n   R e p o r t
Time: 750
CPU Time: 0.460 seconds;   Data structure size: 0.3Mb
Wed Dec 4 15:03:30 2019