SACRAMENTO STATE

Redefine the Possible

CPE 142

Term Project: Phase II

Sergio Zavala

Alejandro Cortez

Team 3

Contribution

50/50 percent

Table of Contents

CSc/CPE 142
Term Project Status Report

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

Name	*% Contribution*	*Grade*
Sergio Zavala	*50%*	
Alejandro Cortez	*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
OR immediate	Yes
Load byte unsigned	Yes
Store byte	Yes
Load	Yes
Store	Yes
Branch on less than	Yes
Branch on greater than	Yes
Branch on equal	Yes

Instructions	Was this instruction fully fuctional as verified by the assembly program provided? If no, explain
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	Sergio	Yes	
ALU control unit	No			
Memory Unit	Yes	Sergio	Yes	Issues with test bench
Register File	Yes	Alejandro	Yes	
PC	Yes	Sergio	Yes	Having issues with Test Bench
IR				
Other registers	Yes	Sergio? Alejandro	Yes	
Multiplexors	Yes	Alejandro	Yes	
exception handler 1. Unknown opcode 2. Arith. Overflow …..				
Control Units 1. main 2. forwarding 3. lw hazard detection	Yes	Sergio / Alejandro	Source Code Yes	Control Unit Test bench is not working Forwarding Test bench is not working

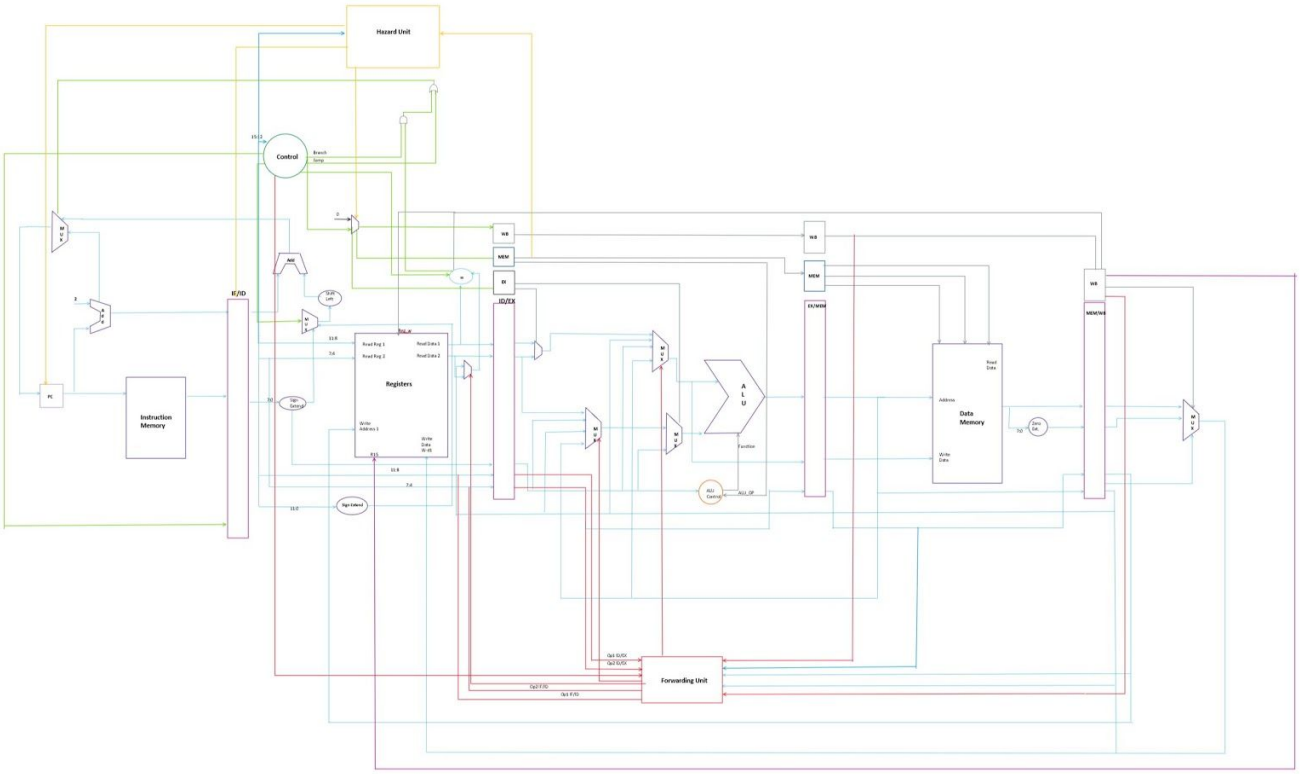
How many stages do you have in your pipeline? …5……..

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

Most of components work. We are missing ALU Op. We had issues with test bench for control_unit, data memory, register file is hardly missing data doesn't go to the expected register, and instruction memory test bench we are getting errors as well.

Also our top level design and fixture didn't compile.

Datapath Design



Truth Table:

Instructions	Opcode	Type	RegDst	Comparator	MemRead	MemToReg	ALUop	RegWrite	Jump
Signed Addition	0000	A	1	0	0	0	0000	1	0
Signed Subtraction	0000	A	1	0	0	0	0001	1	0
Signed Multiplication	0000	A	1	0	0	0	0100	1	0
Signed Division	0000	A	1	0	0	0	0101	1	0
And i	0001	C	0	0	0	0	N/A	1	0
OR i	0010	C	0	0	0	0	N/A	1	0
Load Byte Unsigned	1010	B	0	0	1	1	N/A	1	0
Store Byte	1011	B	0	0	0	0	N/A	0	0
Load	1100	B	0	0	1	1	N/A	1	0
Store	1101	B	0	0	0	0	N/A	0	0
Branch on less than	0101	C	x	1	0	x	N/A	0	0
Branch on greater than	0100	C	x	1	0	x	N/A	0	0
Branch on equal	0110	C	x	1	0	x	N/A	0	0
Jump	0111	D	x	0	0	x	N/A	0	1
Halt	1111	D	x	0	0	x	N/A	0	1

Components

Adder

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

        always@(*)

        begin
                result = a + b;
        end

endmodule
```

Adder Test Bench

```
`timescale 10ns/1ps
`include "add.v"

module add_tb();

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

        add uut(a,b,result);

        initial
        begin

                $dumpvars;
                $display("a | b | result");
                $monitor("%d | %d | %d", a, b, result);
        end

        initial
        begin
                #10; a= 4; b = 16;
                #10; a= 5; b = 32;
                #10; a= 6; b = 64;
                #10; $finish;
        end

endmodule
```

Adder Results

```
[cs142128@olympia:22]> simv
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
 2 14:19 2019
a | b | result
    x |     x |     x
    4 |    16 |    20
    5 |    32 |    37
    6 |    64 |    70
$finish called from file "add_tb.v", line 24.
$finish at simulation time              400000
           V C S   S i m u l a t i o n   R e p o r t
Time: 400000 ps
CPU Time:      0.290 seconds;       Data structure size:   0.0Mb
Mon Dec  2 14:19:18 2019
[cs142128@olympia:23]>
```

And Code

```
module AND (input a,b, output reg c);

always@(*)

        if (a && b)
                c = 1;
        else
                c = 0;

endmodule
```

And Test Bench

```
`timescale 10ns/1ps
`include "AND.v"

module AND_tb();

reg a, b;
wire c;

AND uut(a, b, c);

initial
begin

        #10; a = 0; b = 0;

        #10; b = 1;

        #10; a = 1; b = 0;

        #10; a = 1; b = 1;

end

initial
begin

        $dumpvars;
        $display("a | b | c");
        $monitor(" %d | %d | %d", a, b, c);

end

endmodule
```

Results

```
a | b | c
 x | x | x
 0 | 0 | 0
 0 | 1 | 0
 1 | 0 | 0
 1 | 1 | 1
            V C S   S i m u l a t i o n   R e p o r t
Time: 400000 ps
CPU Time:      0.290 seconds;       Data structure size:   0.0Mb
```

OR Code

```
module OR (input a, b, output reg c);

always@(*)

if(a || b)
        c = 1;
else
        c = 0;

endmodule
```

OR Test Bench

```
`include "OR.v"

module OR_tb();
reg a, b;
wire c;

OR utt( a, b, c);

initial
begin

        #10; a = 0; b = 0;

        #10; a = 0; b = 1;

        #10; a = 1; b = 0;

        #10; a = 1; b = 1;

end

initial
begin
        $dumpvars;
        $display(" a | b | c");
        $monitor(" %d | %d |%d", a, b, c);

end

endmodule
```

Results

```
 a | b | c
 x | x |x
 0 | 0 |0
 0 | 1 |1
 1 | 0 |1
 1 | 1 |1
          V C S   S i m u l a t i o n   R e p o r t
Time: 40
CPU Time:      0.290 seconds;       Data structure size:   0.0Mb
Mon Dec  2 19:39:46 2019
```

ALU Code

```
module alu( input [15:0] a, b, output reg detect, input [2:0] func, output reg [31:0] result );


always@(*)

case (func)
        4'b0000: begin
                result[15:0] = a + b;
                if( ( result[15:0] < a) || (result[15:0] < b) )
                        detect = 1;
                else
                        detect = 0;
                end
        4'b0001: begin
                result[15:0] = a - b;
                if( result[15:0] > a )
                        detect = 1;
                else
                        detect = 0;
                end
        4'b0100: begin
                        result = a * b;
                        detect = 0;
                end
        4'b0101: begin
                        result[15:0] = a / b;
                        result[31:16] = a % b;
                        detect = 0;
                end
endcase

endmodule
```

ALU Test Bench

```
`include "alu.v"
module alu_tb ();
reg [15:0] a, b;
reg [2:0] func;
wire [31:0] result;
wire detect;

alu DUT(.a(a), .b(b), .func(func), .result(result), .detect(detect));

initial
begin
        $dumpvars;
        $display(" a  |  b  | func | result | detect ");
        $monitor(" %b | %b  | %b | %b  | %b", a, b, func, result, detect);
end

initial
begin
        a = 1000;
        b = 100;
        func = 4'b0000;
        #10;
        func = 4'b0001;
        #10;
        func = 4'b0010;
        #10;
        func = 4'b0011;
        #10;
        func = 4'b0100;
        #10;
        func = 4'b0101;
        #10 $finish;
end


endmodule
```

Results

```
a  |  b  | func | result | detect
0000001111101000 | 0000000001100100  | 000 | xxxxxxxxxxxxxxxx0000010001001100  | 0
0000001111101000 | 0000000001100100  | 001 | xxxxxxxxxxxxxxxx0000001110000100  | 0
0000001111101000 | 0000000001100100  | 010 | xxxxxxxxxxxxxxxx0000001110000100  | 0
0000001111101000 | 0000000001100100  | 011 | xxxxxxxxxxxxxxxx0000001110000100  | 0
0000001111101000 | 0000000001100100  | 100 | 00000000000000001100001101010000  | 0
0000001111101000 | 0000000001100100  | 101 | 00000000000000000000000000001010  | 0
$finish called from file "alu_tb.v", line 33.
$finish at simulation time                   60
          V C S   S i m u l a t i o n   R e p o r t
```

Compare

```
module compare(input[15:0] op1, R15, input [1:0] branchOp, output reg [15:0] result);

        always@(*)

        begin


                        if( op1 > R15)
                        begin
                                assign result = 4'b0100;
                        end

                        else if ( op1 < R15)
                        begin
                                assign result = 4'b0101;
                        end

                        else if( op1 == R15)
                        begin
                                assign result = 4'b0110;
                        end

end

endmodule
```

Compare Test Bench

```
`include "compare.v"

module compare_tb();

reg [15:0] op1, R15;
wire [15:0] result;

Compare DUT(.op1(op1), .R15(R15), .result(result));

initial
        begin
                $dumpvars;
                $display("op1 | R15 | result");
                $monitor (" %b | %b |%b", op1, R15, result);
        end

initial
        begin

        op1 = 0;
        R15 = 10;
        #10;

        R15 = 15;
        #10;

        R15 = 10;
        #10;

        R15 = 5;
        #10;

        R15 = 10;
        #10;

        R15 = 5;
        #10 $finish;

        end

endmodule
```

Results

```
op1 | R15 | result
 0000000000000000 | 0000000000001010 |0000000000000101
 0000000000000000 | 0000000000001111 |0000000000000101
 0000000000000000 | 0000000000001010 |0000000000000101
 0000000000000000 | 0000000000000101 |0000000000000101
 0000000000000000 | 0000000000001010 |0000000000000101
 0000000000000000 | 0000000000000101 |0000000000000101
$finish called from file "compare_tb.v", line 38.
$finish at simulation time                   60
           V C S   S i m u l a t i o n   R e p o r t
```

Data Memory

```
module data_memory(input clk, rst, enable, input [15:0] address, wData, input [1:0] mWrite, mRead, output reg [15:0] rData);

reg [15:0] memory [15:0];

always@(*)
begin

		if(mWrite)
		begin

			if(enable)
				memory[address] = wData[7:0];
			else
			begin
				memory[address] = wData[15:8];
				memory[address + 1] = wData[7:0];

		end
	end
end

always@(*)
begin
		rData = { memory[address], memory[address+2] };

end

always@(posedge clk or negedge rst)
begin
		if(!rst)
		begin

			memory [0] = 16'h3142;
			memory [2] = 16'h0000;
			memory [4] = 16'h5678;
			memory [6] = 16'hDEAD;
			memory [8] = 16'hBEEF;

	end
end

endmodule
```

Instruction Memory

```
module instructon_memory(input [15:0] address, input rst, clk, output reg [15:0] instruction);

parameter amount_of_instructions = 62;
reg [15:0] mem[0:61];

always@(*)
begin
			if(!rst)
			begin
					mem[0] = 16'h246C;
					mem[2] = 16'h3088;
					mem[4] = 16'hFF88;
					mem[6] = 16'hF09A;
					mem[8] = 16'h12F0;
					mem[10] = 16'h005A;
					mem[12] = 16'hFF82;
					mem[14] = 16'h001A;
					mem[16] = 16'h0000;
					mem[18] = 16'h9F73;
					mem[20] = 16'h56BD;
					mem[22] = 16'hFF82;
					mem[24] = 16'h6664;
					mem[26] = 16'h6666;
					mem[28] = 16'h6666;
					mem[30] = 16'h00FD;
					mem[32] = 16'h0FF4;
					mem[34] = 16'hABB1;
					mem[36] = 16'h0FF5;
					mem[38] = 16'h54F4;
					mem[40] = 16'h0FF2;
					mem[42] = 16'hF630;
					mem[44] = 16'hFF88;
					mem[46] = 16'hFF10;
					mem[48] = 16'hFF90;
					mem[50] = 16'h0002;
					mem[52] = 16'h0004;
					mem[54] = 16'h0000;
					mem[56] = 16'h0013;
					mem[58] = 16'h0000;
					mem[60] = 16'h0000;
			end
```

```
                                        mem[28] = 16'h6666;
                                        mem[30] = 16'h00FD;
                                        mem[32] = 16'h0FF4;
                                        mem[34] = 16'hABB1;
                                        mem[36] = 16'h0FF5;
                                        mem[38] = 16'h54F4;
                                        mem[40] = 16'h0FF2;
                                        mem[42] = 16'hF630;
                                        mem[44] = 16'hFF88;
                                        mem[46] = 16'hFF10;
                                        mem[48] = 16'hFF90;
                                        mem[50] = 16'h0002;
                                        mem[52] = 16'h0004;
                                        mem[54] = 16'h0000;
                                        mem[56] = 16'h0013;
                                        mem[58] = 16'h0000;
                                        mem[60] = 16'h0000;
                            end

                            else

                            instruction = mem[address];

                            end

endmodule
```

Instruction Memory Test Bench

```
`include "instruction_memory.v";

module instruction_memory_tb();

                reg rst, clk;
                reg [15:0] address;
                wire [15:0] instruction;

        instruction_memory UUT(.rst(rst), .clk(clk), .address(address), .instruction(instruction));

initial
begin
        $dumpvars;
        $display("rst | clk | address | instruction  ");
        $monitor("%b  | %b  | %b  | %b  ", rst, clk, address, instruction);
end


initial
        clk = 0;
always
begin
         #10;
         clk = !clk;

end

initial
begin
        rst = 0;
        #10;
        rst = 1;
        address = 0;
        #20;
        address = 2;
        #20;
    address = 4;
        #20;
    address = 16;
        #20;
        address = 32;
        #20 $finish;
```

Hazard

```
module hazard(input [3:0] Dec_Op1, Fetch_Op1, Fetch_Op2, Mem_Op1, Mem_Wp1,
                          input branchOp, branch, memR, memRead , Write_mem, regWrite,
                          output reg bubble_mem, Data_write, PC_write);

always@(*)
begin
                begin
                bubble_mem = 0;
                Data_write = 0;
                PC_write = 0;
                end

                if(branch)
                begin
                bubble_mem = 1;
                Data_write = 1;
                PC_write = 1;
                end

                else if( memR && ((Fetch_Op1 == Dec_Op1) || (Fetch_Op2 == Dec_Op1)))
                begin
                bubble_mem = 1;
                Data_write = 1;
                PC_write = 1;
                end

                else if( memRead && branchOp && (Mem_Op1 == Dec_Op1))
                begin
                bubble_mem = 1;
                Data_write = 1;
                PC_write = 1;
                end

                else if( Write_mem && branchOp && (Mem_Wp1 == Dec_Op1))
                begin
                bubble_mem = 1;
                Data_write = 1;
                PC_write = 1;
                end

                else if( regWrite && branchOp && (Mem_Wp1 == Dec_Op1))
                begin
                bubble_mem = 1;
  INSERT
```

```
                else if( memR && ((Fetch_Op1 == Dec_Op1) || (Fetch_Op2 == Dec_Op1)))
                begin
                bubble_mem = 1;
                Data_write = 1;
                PC_write = 1;
                end

                else if( memRead && branchOp && (Mem_Op1 == Dec_Op1))
                begin
                bubble_mem = 1;
                Data_write = 1;
                PC_write = 1;
                end

                else if( Write_mem && branchOp && (Mem_Wp1 == Dec_Op1))
                begin
                bubble_mem = 1;
                Data_write = 1;
                PC_write = 1;
                end

                else if( regWrite && branchOp && (Mem_Wp1 == Dec_Op1))
                begin
                bubble_mem = 1;
                Data_write = 1;
                PC_write = 1;
                end
end

endmodule
```

Hazard Test Bench

```
`include "hazard.v";
module hazard_tb();
reg [3:0] Dec_Op1, Fetch_Op1, Fetch_Op2, Mem_Op1, Mem_Wp1;
reg branchOp, branch, memR, memRead, Write_mem, regWrite;
wire bubble_mem, Data_write, PC_write;

hazard DUT(
        .Fetch_Op1(Fetch_Op1),
        .Fetch_Op2(Fetch_Op2),
        .Dec_Op1(Dec_Op1),
        .Mem_Op1(Mem_Op1),
        .Mem_Wp1(Mem_Wp1),
        .memR(memR),
        .memRead(memRead),
        .Write_mem(Write_mem),
        .regWrite(regWrite),
        .branchOp(branchOp),
        .branch(branch),
        .bubble_mem(bubble_mem),
        .Data_write(Data_write),
        .PC_write(PC_write)
        );

initial
begin
        $dumpvars;
        $display(" Fetch_Op1 | Fetch_Op2 | Dec_Op1 | Mem_Op1 | Mem_Wp1 | memR | memRead | Write_mem | regWrite | branchOp | branch | bubble_mem ");
        $monitor(" %b        | %b        | %b      | %b      | %b      | %b   | %b      | %b        | %b       | %b       | %b     | %b", Fetch_Op1, Fetch_Op2, Dec_Op1, Mem_Op1, Mem_Wp1
, memR, memRead, Write_mem, regWrite, branchOp, branch, bubble_mem);
end

initial
begin
        branchOp = 0;
        branch = 0;
        memR = 0;
        memRead = 0;
        Write_mem = 0;
        regWrite = 0;
        Dec_Op1 = 0;
        Fetch_Op1 = 1;
        Fetch_Op2 = 1;
```

Results

```
 Fetch_Op1 | Fetch_Op2 | Dec_Op1 | Mem_Op1 | Mem_Wp1 | memR | memRead | Write_mem | regWrite | branchOp | branch | bubble_mem
 0001      | 0001      | 0000    | 0010    | 0011    | 0    | 0       | 0         | 0        | 0        | 0      | 0
 0001      | 0000      | 0000    | 0010    | 0011    | 0    | 0       | 0         | 0        | 0        | 0      | 0
 0000      | 0000      | 0000    | 0010    | 0011    | 1    | 0       | 0         | 0        | 0        | 0      | 1
 0000      | 0011      | 0000    | 0010    | 0011    | 0    | 0       | 0         | 0        | 0        | 0      | 0
 0000      | 0011      | 0000    | 0010    | 0000    | 0    | 0       | 0         | 1        | 0        | 0      | 0
 0000      | 0011      | 0000    | 0010    | 0000    | 0    | 0       | 1         | 0        | 1        | 0      | 1
 0000      | 0011      | 0000    | 0010    | 0000    | 0    | 0       | 1         | 1        | 1        | 0      | 1
 0000      | 0011      | 0000    | 0000    | 0000    | 0    | 1       | 1         | 1        | 1        | 0      | 1
 0000      | 0011      | 0000    | 0000    | 0011    | 0    | 1       | 1         | 1        | 0        | 0      | 0
$finish called from file "hazard_tb.v", line 85.
$finish at simulation time                110
          V C S   S i m u l a t i o n   R e p o r t
```

Control Unit

```
module control_unit(input [3:0] OpCode, funcIn,
                    output reg [2:0] aluOp,
                    output reg [1:0] branchOp, wbSrc,
                    output reg alu_src_1, alu_src_2, enable, memW, memR, r15_write, regW, branch, jump, branchDetect, halt);

        always@(*)
        begin

                case(funcIn)

                        4'b0000:
                        begin
                        alu_src_1 = 0;
                        alu_src_2 = 0;
                        enable = 0;
                        memR = 0;
                        memW = 0;
                        wbSrc = 2'b00;
                        branch = 0;
                        branchOp = 2'b00;
                        branchDetect = 0;
                        jump = 0;
                        aluOp = 3'b000;
                        regW = 0;
                        halt = 0;
                        end

                        4'b0001:
                        begin
                        alu_src_1 = 0;
                        alu_src_2 = 1;
                        enable = 0;
                        memR = 0;
                        memW = 0;
                        wbSrc = 2'b01;
                        branch = 0;
                        branchOp = 2'b01;
                        branchDetect = 0;
                        jump = 0;
                        aluOp = 3'b001;
                        regW = 1;
                        r15_write = 0;
                        end
```

```
                4'b0010:
                begin
                alu_src_1 = 0;
                alu_src_2 = 1;
                enable = 0;
                memR = 0;
                memW = 0;
                wbSrc = 2'b10;
                branch = 0;
                branchOp = 2'b10;
                branchDetect = 0;
                jump = 0;
                aluOp = 3'b010;
                regW = 1;
                r15_write = 0;
                end

                4'b1010:
                begin
                alu_src_1 = 1;
                alu_src_2 = 1;
                enable = 1;
                memR = 1;
                memW = 0;
                wbSrc = 2'b10;
                branch = 0;
                branchOp = 2'b10;
                branchDetect = 0;
                jump = 0;
                aluOp = 3'b010;
                regW = 1;
                r15_write = 0;
                end

                4'b1011:
                begin
                alu_src_1 = 1;
                alu_src_2 = 1;
                enable = 1;
                memR = 0;
                memW = 1;
```

```
                        halt = 0;
                        end

                        4'b0100:
                        begin
                        branch = 1;
                        branchOp = 2'b00;
                        branchDetect = 1;
                        jump = 0;
                        halt = 0;
                        end

                        4'b0110:
                        begin
                        branch = 1;
                        branchOp = 2'b10;
                        branchDetect = 1;
                        jump = 0;
                        halt = 0;
                        end

                        4'b0111:
                        begin
                        branch = 0;
                        branchDetect = 0;
                        jump = 1;
                        halt = 0;
                        end

                        4'b1111:
                        begin
                        halt = 1;
                        end

                endcase

        end
endmodule
```

Control Unit Test Bench

```
`include "control_unit.v";

module control_unit_tb();


control_unit UUT(
        .funcIn(funcIn),
        .OpCode(OpCode),
        .alu_src_1(alu_src_1),
        .alu_src_2(alu_src_2),
        .aluOp(aluOp),
        .enable(enable),
        .memW(memW),
        .memR(memR),
        .branch (branch ),
        .branchDetect (branchDetect),
        .jump(jump),
        .r15_write(r15_write),
        .halt(halt)
        );

initial
begin
        $dumpvars;
        $display(" funcIn | OpCode | alu_src_1 | alu_src_2 | aluOp | enable | memW | memR | branch  | branchDetect  | jump | r15_write | halt ");
        $monitor(" %b    | %b     | %b        | %b        | %b    | %b     | %b   | %b    | %b      | %b       | %b   | %b    | %b ", funcIn, OpCode, alu_src_1, alu_src_2, aluOp, enable, me
mW, memR, branch , branchDetect , jump, r15_write, halt);
end

initial
begin
        funcIn = 4'b1111;
        OpCode = 4'b0000;
        #1;
        OpCode = 4'b0001;
        #1;
        OpCode = 4'b0100;
        #1;
        OpCode = 4'b0101;
        #1;
        funcIn = 4'b1000;
        OpCode = 4'b0000;
```

PC

```
module pc(input [15:0] write_back, input clk, w_enable, output reg [15:0] result);

        always@(*)
        begin

        if(w_enable)
        begin
                assign result = write_back;
        end

        else
        begin
                assign result = result;
        end

        end

endmodule
```

PC Test Bench

```
`include "pc.v"
module pc_tb();

        reg [15:0] write_back;
        reg clk, w_enable;

        wire [15:0] result;

        pc UUT(write_back, clk, w_enable, result);

        initial
        begin
                $dumpvars;
                $display("write_back | w_enable | result");
                $monitor("%d | %d | %d", write_back, w_enable, result);
        end

        initial
        begin
                clk = 0;
                write_back = 0;
                w_enable = 0;
        end

        initial
        begin
                 w_enable = 0;
                #0 write_back = 0;
                #5 w_enable = 1;
                #5 write_back = 5;
                #5 w_enable = 0;
                #5 write_back = 15;
                #5 w_enable = 1;
                #5 write_back = 10;
                #5 write_back = 1;
                #5 w_enable = 0;
                #5 write_back = 10;
                #5 write_back = 0;
                #10 $finish;

        end

        initial
```

Mux 2 to 1:

```
module MUX2_1( d0, d1, m, sel );
input [15:0] d0, d1;
input  sel;
output reg [15:0] m;

always@(*)

if( sel )
        m = d1;
else
        m = d0;
endmodule
```

Mux 2 to 1 Testbench:

```
`include "MUX2_1.v"
module MUX2_1_tb();
reg [15:0] d0, d1;
reg sel;
wire [15:0] m;

MUX2_1 DUT(.d0(d0), .d1(d1), .sel(sel), .m(m));

initial
begin
        d0 <= 200;
        d1 <= 345;
        sel <= 0;

        #20;
        sel <= 1;

        #20;
        sel <= 0;

        #5;

end

initial
begin

        $dumpvars;
        $display("   d0      d1     sel    m");
        $monitor(" %d   %d    %b   %d ", d0, d1, sel, m);
end
endmodule
```

Mux 2 to 1 Results:

```
  d0      d1     sel    m
  200     345    0     200
  200     345    1     345
  200     345    0     200
          V C S   S i m u l a t i o n   R e p o r t
```

Mux 3 to 1:

```
module MUX3_1(d0, d1, d2, sel, m);
input [15:0] d0, d1, d2;
input  [1:0] sel;
output reg [15:0] m;

always@(*)
case ( sel )

        0: m = d0;
        1: m = d1;
        2: m = d2;

        default: m = d0;

endcase
endmodule
```

Mux 3 to 1 Testbench:

```
`include "MUX3_1.v"
module MUX3_1_tb();
reg [15:0] d0, d1, d2;
reg [1:0] sel;
wire [15:0] m;

MUX3_1 DUT( .d0(d0), .d1(d1), .d2(d2),.sel(sel), .m(m));

initial
begin
        d0 <= 10;
        d1 <= 22;
        d2 <= 764;
        sel <= 0;

        #5; sel <= 1;
        #5; sel <= 2;
        #5; sel <= 3;
        #5;

end

initial
begin

        $dumpvars;
        $display("    d0    d1    d2    sel    m");
        $monitor(" %d  %d  %d  %b  %d", d0, d1, d2, sel, m);

end
endmodule
```

Mux 3 to 1 Results:

```
  d0    d1    d2    sel    m
  10     22    764  00     10
  10     22    764  01     22
  10     22    764  10    764
  10     22    764  11     10
         V C S   S i m u l a t i o n   R e p o r t
```

Mux 5 to 1:

```
module MUX5_1 (d0, d1, d2, d3, d4, sel, m );
input [15:0] d0, d1, d2, d3, d4;
input  [2:0] sel;
output reg [15:0] m;

always@(*)
case ( sel )

        0: m = d0;
        1: m = d1;
        2: m = d2;
        3: m = d3;
        4: m = d4;

        default: m = d0;

endcase
endmodule
```

Mux 5 to 1 Testbench:

```
`include "MUX5_1.v"
module MUX5_1_tb();
reg [15:0] d0, d1, d2, d3, d4;
reg [2:0] sel;
wire [15:0] m;
MUX5_1 DUT(.d0(d0), .d1(d1), .d2(d2),.d3(d3),.d4(d4),.sel(sel), .m(m));

initial
begin
        d0 <= 400;
        d1 <= 974;
        d2 <= 1024;
        d3 <= 2059;
        d4 <= 4097;
        sel <= 0;

        #5; sel <= 1;

        #5; sel <= 2;

        #5; sel <= 3;

        #5; sel <= 4;

        #5; sel <= 5;

        #5;
end

initial
begin
    $dumpvars;
    $display("  do    d1   d2    d3    d4   sel   m");
    $monitor("%d %d %d %d %d %b %d" ,d0, d1, d2, d3, d4, sel, m);

end
endmodule
```

Mux 5 to 1 Results:

```
  do    d1   d2    d3    d4   sel   m
  400   974  1024  2059  4097 000   400
  400   974  1024  2059  4097 001   974
  400   974  1024  2059  4097 010  1024
  400   974  1024  2059  4097 011  2059
  400   974  1024  2059  4097 100  4097
  400   974  1024  2059  4097 101   400
         V C S   S i m u l a t i o n   R e p o r t
```

Shift Left:

```
module SHL( in, out );
input [15:0] in;
output reg [15:0] out;

always@(*)

out = { in[14:0], 1'b0 };

endmodule
```

Shift Left Testbench:

```
`include "SHL.v"
module SHL_tb();
reg [15:0] in;
wire [15:0] out;
SHL  DUT(.in(in),.out(out));

initial
begin
        #5
        in = 8'b01110111;
        #5;
        in = 8'b11111110;
        #5;
        in = 16'b1010101010101010;
        #5;

end

initial
begin

        $monitor(" in %b, out %b  ", in, out);

end
endmodule
```

Shift Left Testbench:

```
 in xxxxxxxxxxxxxxxx, out xxxxxxxxxxxxxxxx
 in 0000000001110111, out 0000000011101110
 in 0000000011111110, out 0000000111111100
 in 1010101010101010, out 0101010101010100
          V C S   S i m u l a t i o n   R e p o r t
```

Sign Extend:

```
module SIGN_EXT ( in, out );
parameter int = 8;
input [int - 1:0] in;
output reg [15:0] out;

always@(*)

out = {{(16 - int){in[int - 1]}}, in };

endmodule
```

Sign Extend Testbench:

```
`include "SIGN_EXT.v"
module SIGN_EXT_tb();
reg [11:0] unex;
wire [15:0] ext;
SIGN_EXT #(.int(11)) DUT(.in(unex),.out(ext));

initial
begin
        #5;
        unex = 8'b11111111;
        #5;
        unex = 8'b10100100;
        #5;
        unex = 8'b00011111;
        #5;
end

initial
begin
        $dumpvars;
        $display(" unex        ext");
        $monitor("%b %b", unex, ext);
end
endmodule
```

Sign Extend Results:

```
 unex        ext
xxxxxxxxxxxx xxxxxxxxxxxxxxxx
000011111111 0000000011111111
000010100100 0000000010100100
000000011111 0000000000011111
         V C S   S i m u l a t i o n   R e p o r t
```

Zero Extend:

```
module ZERO_EXT(in, out);
input [7:0] in;
output reg [15:0] out;

always@(*)

        out = {{8{0}}, in };

endmodule
```

Zero Extend Testbench:

```
`include "ZERO_EXT.v"
module ZERO_EXT_tb ();
reg [7:0] unex;
wire [15:0] ext;
ZERO_EXT  DUT(.in(unex),.out(ext));

initial
begin
        #5;
        unex = 8'b11111111;
        #5;
        unex = 8'b01110111;
        #5;
        unex = 8'b10101010;
end

initial
begin

    $dumpvars;
    $display("unex     ext");
    $monitor("%b %b", unex, ext);
end
endmodule
```

Zero Extend Results:

```
unex     ext
xxxxxxxx xxxxxxxxxxxxxxxx
11111111 0000000011111111
01110111 0000000001110111
10101010 0000000010101010
         V C S   S i m u l a t i o n   R e p o r t
```

Register File:

```
module REG_FILE(clk, rst, reg_we, reg14_we,op1_data,
                op2_data, reg14_data,op1_addr,
                op2_addr, w_addr, w_data, w_reg14);
input clk, rst, reg_we, reg14_we;
input[3:0] op1_addr, op2_addr, w_addr;
input[15:0] w_data, w_reg14;
output reg [15:0] op1_data, op2_data, reg14_data;
reg [15:0] register [15:1];

always@(posedge clk,negedge rst)
begin
        if(~rst)
        begin
                register [0] = 16'h7b18;
                register [1] = 16'h245b;
                register [2] = 16'hff0f;
                register [3] = 16'hf0ff;
                register [4] = 16'h0051;
                register [5] = 16'h6666;
                register [6] = 16'h00ff;
                register [7] = 16'hff88;
                register [8] = 16'h0000;
                register [9] = 16'h0000;
                register [10] = 16'h3099;
                register [11] = 16'hcccc;
                register [12] = 16'h0002;
                register [13] = 16'h0011;
                register [14] = 16'h0000;
        end

        else
                begin
                if(reg_we)
                        register[w_addr] = w_data;
                if(reg14_we)
                        register[14] = w_reg14;
        end
end

always@(*)
begin
        op1_data = register[ op1_addr ];
        op2_data = register[ op2_addr ];
        reg14_data = register[14];
end
endmodule
```

Register Forwarding:

```
module REG_FWD( fetch_op1, fetch_op2, decodeOp1, decodeOp2, ex_memop2,
                mem_wop2, branch, m_wregwrite, exRegW, exReg14w,
                memReg14w, alu1, alu2, cmpsrc);
input [3:0] fetch_op1, fetch_op2, decodeOp1, decodeOp2, ex_memop2, mem_wop2;
input branch, m_wregwrite, exRegW, exReg14w, memReg14w;
output reg [2:0] alu1, alu2;
output reg [1:0] cmpsrc;

always@(*)
begin

        if( exRegW && (ex_memop2 != 0) && (ex_memop2 == decodeOp1) )
                alu1 = 4'b0100;
        else if( memReg14w && (decodeOp1 == 4'b1111) )
                alu1 = 4'b0001;
        else
                alu1 = 4'b0000;

        if( exRegW && (ex_memop2 != 0) && (ex_memop2 == decodeOp2) )
                alu2 = 4'b0100;
        else if( memReg14w && (decodeOp2 == 4'b1111) )
                alu2 = 4'b0001;
        else
                alu2 = 4'b0000;

        if( branch && exReg14w )
                cmpsrc = 2'b10;
        else if( branch && memReg14w && !exReg14w )
                cmpsrc = 2'b01;
        else
                cmpsrc = 2'b00;
end
endmodule
```

Top Level Code

```
`include "add.v";
`include "alu.v";
`include "AND.v";
`include "compare.v";
`include "control_unit.v";
`include "data_memory.v";
`include "hazard.v";
`include "instruction_memory.v";
`include "MUX2_1.v";
`include "MUX3_1.v";
`include "MUX5_1.v";
`include "OR.v";
`include "REG_FILE.v";
`include "REG_FWD.v";
`include "SHL.v";
`include "SIGN_EXT.v";
`include "ZERO_EXT.v";

module cpu( clk, rst );
input clk, rst;

wire [15:0] pcPlus2, bnchjumpaddress, instruction;
wire pcSrc, PC_write, halt;
wire [31:0] funcIn;

wire [15:0] op1Data, op2Data, R15Data, R15, op1, writeBackSelect;
wire [15:0] signExt12result, signExt8result, JBsel, JBoffset,zeroExtresult;
wire [2:0] aluOP;
wire [1:0] cmpSrc, branchOp, wbSrc;
wire result, branch, alu_src_1, alu_src_2, enable, memW, memR, regW, r15_write, branchDetect;
wire [67:0] DEregresult;
wire [11:0] controlSignals;
assign controlSignals = {aluOP, alu_src_2, alu_src_1, enable, memW, memR, wbSrc, regW, r15_write};

wire [15:0] a, b;
wire detect;
wire [59:0] enable2;

wire [15:0] memresult, zeroresult, WBdata;
wire [37:0] mem_wResult;

wire bubble_mem, regWrite;
```

```
MUX2_1 pcMux (
        .d0(pcPlus2),
        .d1(bnchjumpaddress),
        .m(nextaddress),
        .sel(pcSrc)
        );

instruction_memory instruction_memory (
        .clk(clk),
        .rst(rst),
        .addressess(addressess),
        .instruction(instruction)
        );

REG_FILE registers (
        .clk(clk),
        .rst(rst),
        .op1_address(funcIn[11:8]),
        .op2_address(funcIn[7:4]),
        .w_data(mem_wResult[17:2]),
        .w_address(mem_wResult[37:34]),
        .w_reg14(mem_wResult[33:18]),
        .op1_data(op1Data),
        .op2_data(op2Data),
        .reg14_data(R15Data),
        .reg_we(mem_wResult[1]),
        .reg14_we(mem_wResult[0])
        );
SIGN_EXT #(.IN_SIZE( 12 )) signExt12 (
        .in(funcIn[11:0]),
        .result(signExt12result)
        );
SIGN_EXT #(.IN_SIZE( 8 )) signExt8 (
        .in(funcIn[7:0]),
        .result(signExt8result)
        );
ZERO_EXT decodeZero (
        .in(funcIn[7:0]),
        .result(zeroExtresult)
        );
MUX2_1 b_jaddressSel (
        .d0(signExt8result),
       .d1(signExt12result),
```

```
MUX3_1 mux3 (
        .d0(memresult),
        .d1(zeroresult),
        .d2(enable[38:23]),
        .m(WBdata),
        .sel(enable[3:2])
        );


REG_FWD register_fwd (
        .fetch_op1(funcIn[11:8]),
        .decodeOp1(DEregresult[63:60]),
        .decodeOp2(DEregresult[67:64]),
        .ex_memop2(enable[58:55]),
        .mWriteop2(mem_wResult[37:34]),
        .branch(branchDetect),
        .m_wregwrite(mem_wResult[1]),
        .exRegW(enable[1]),
        .memReg14w(mem_wResult[0]),
        .exReg14w(enable[0]),
        .alufwd1(fwdA),
        .alufwd2(fwdB),
        .cmpsrc(cmpSrc),
        );


hazard hazard_detection (
        .branchOp(branchDetect),
        .branch(branch),
        .bubble_mem(bubble_mem),
        .memRead(memR),
        .memR(enable[4]),
        .Data_write(enable[1]),
        .Write_mem(),
        .regWrite(regWrite),
        .PC_write(PC_write),
        .Dec_Op1(DEregresult[62:60]),
        .Fetch_Op1(funcIn[11:8]),
        .Fetch_Op2(funcIn[7:4]),
        .Mem_Op1(enable[58:55]),
        .Mem_Wp1(mem_wResult[37:34])
        );
endmodule
```

Top Level Fixture

```
`include "cpu.v";
module cpu_fixture();
reg clk, rst;

cpu DUT(
        .clk(clk),
        .rst(rst)
        );
initial
        $vcdpluson;

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

always@( negedge clk) //display
begin
        if($time > 1) begin
        $display("\n\t\tTime = %4d\tPC = %h\n\tRegister Values:", $time, cpu.pc.rdata);
        $display("\t\tR0 = %h\tR1 = %h\tR2 = %h\tR3 = %h", cpu.register.register[0], cpu.register.register[1], cpu.register.register[2], cpu.register.register[3]);

        $display("\t\tR4 = %h\tR5 = %h\tR6 = %h\tR7 = %h", cpu.register.register[4], cpu.register.register[5], cpu.register.register[6], cpu.register.register[7]);

        $display("\t\tR8 = %h\tR9 = %h\tR10 = %h\tR11 = %h", cpu.register.register[8], cpu.register.register[9], cpu.register.register[10], cpu.register.register[11]);

        $display("\t\t12 = %h\t13 = %h\tR14 = %h", cpu.register.register[12], cpu.register.register[13], cpu.register.register[14], );

        $display("\tMemory Values:");
        $display("\t\tmem[00] = %h\tmem[02] = %h",{cpu.dataMem.memory[0],cpu.dataMem.memory[1]}, {cpu.dataMem.memory[2],cpu.dataMem.memory[3]} );

        $display("\t\tmem[04] = %h\tmem[06] = %h",{cpu.dataMem.memory[4],cpu.dataMem.memory[5]}, {cpu.dataMem.memory[6],cpu.dataMem.memory[7]} );

        $display("\t\tmem[08] = %h",{cpu.dataMem.memory[8],cpu.dataMem.memory[9]});
        end
end
```