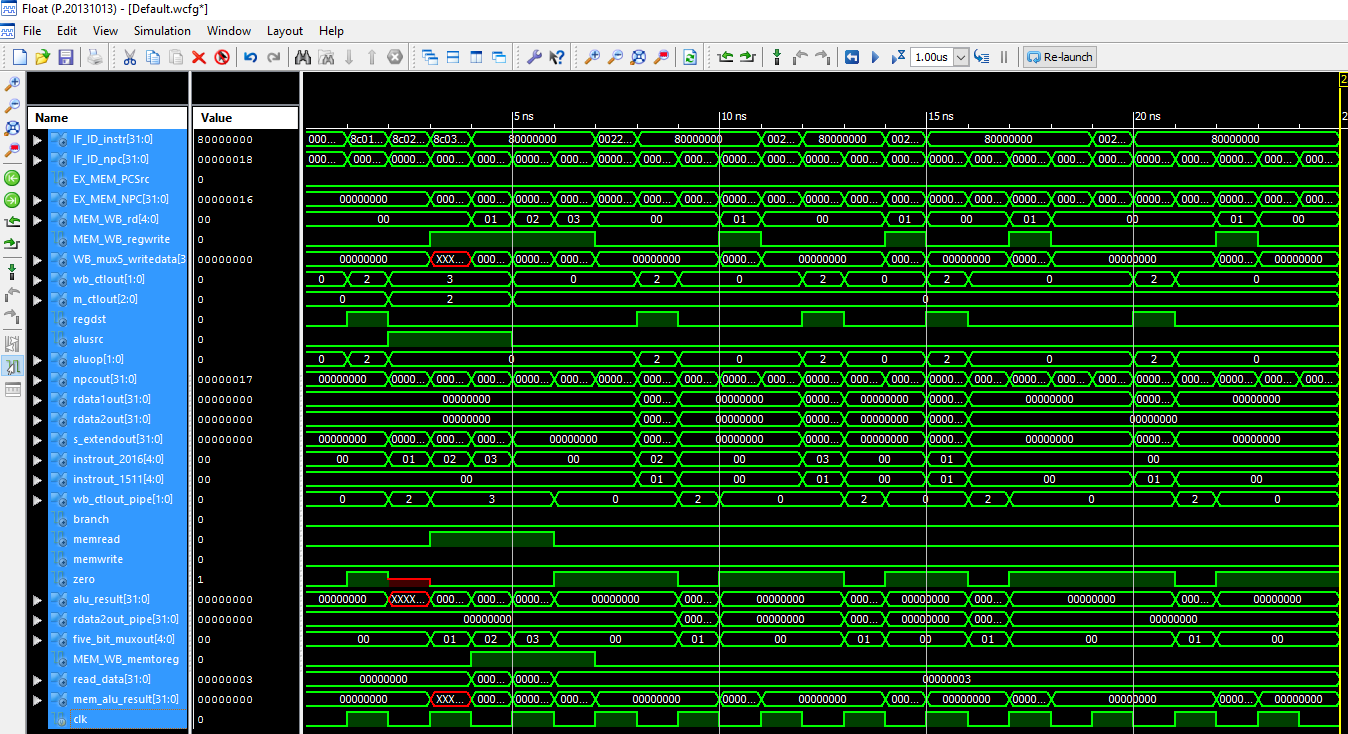
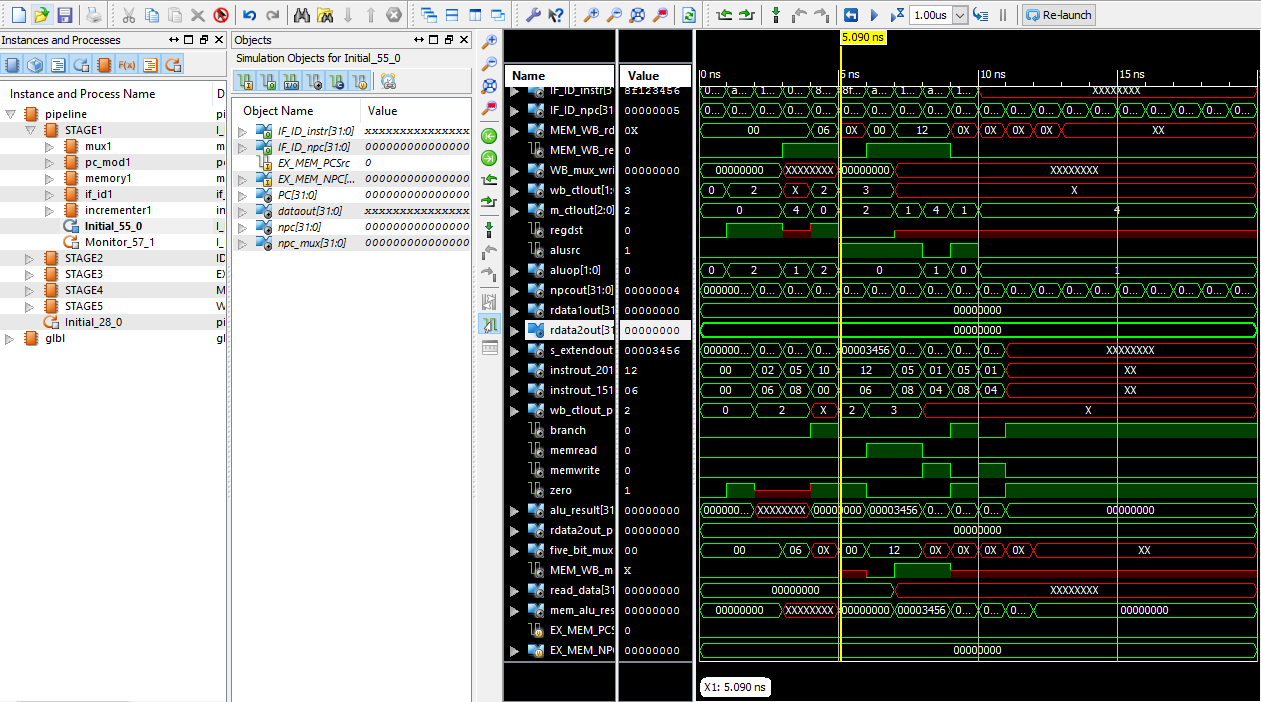
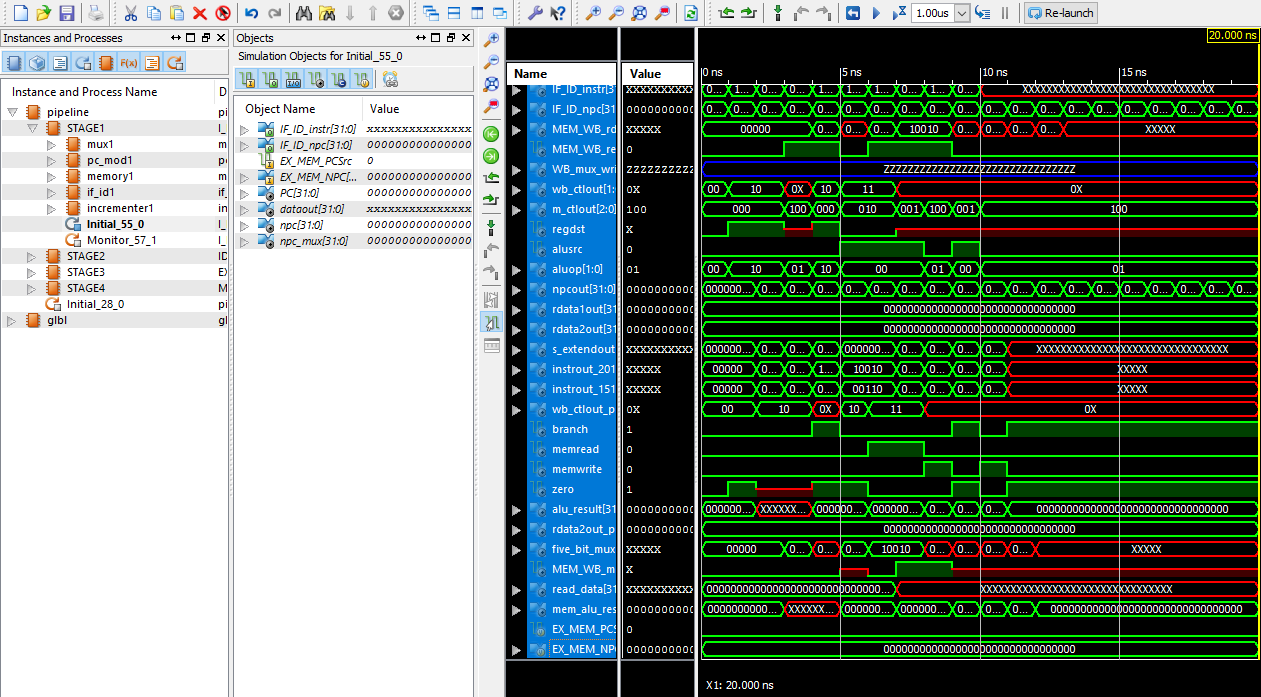
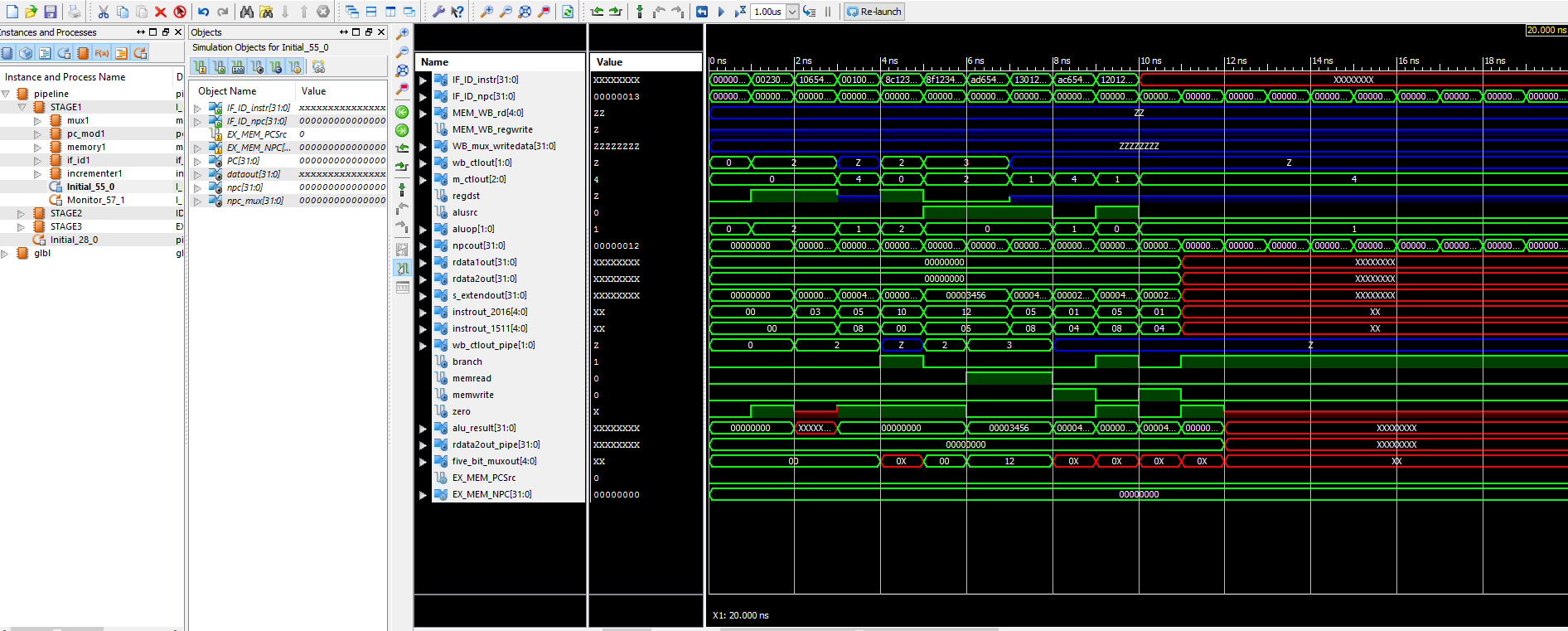
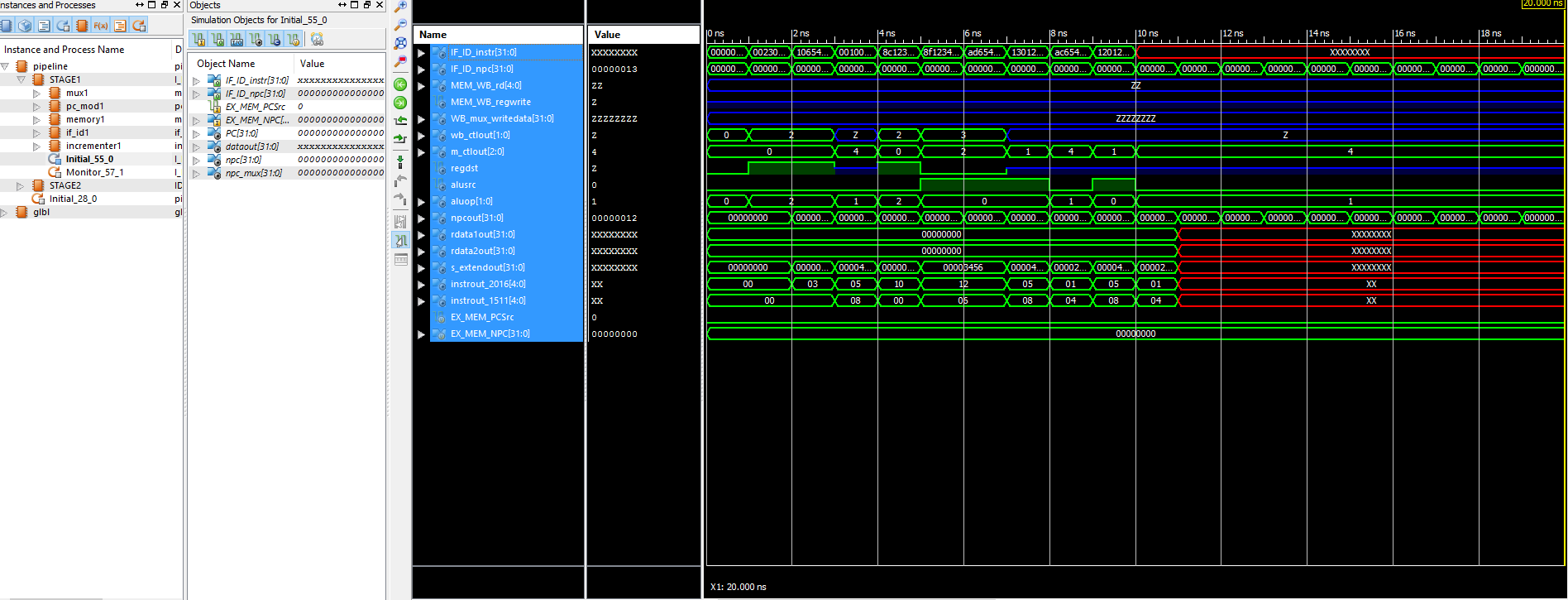
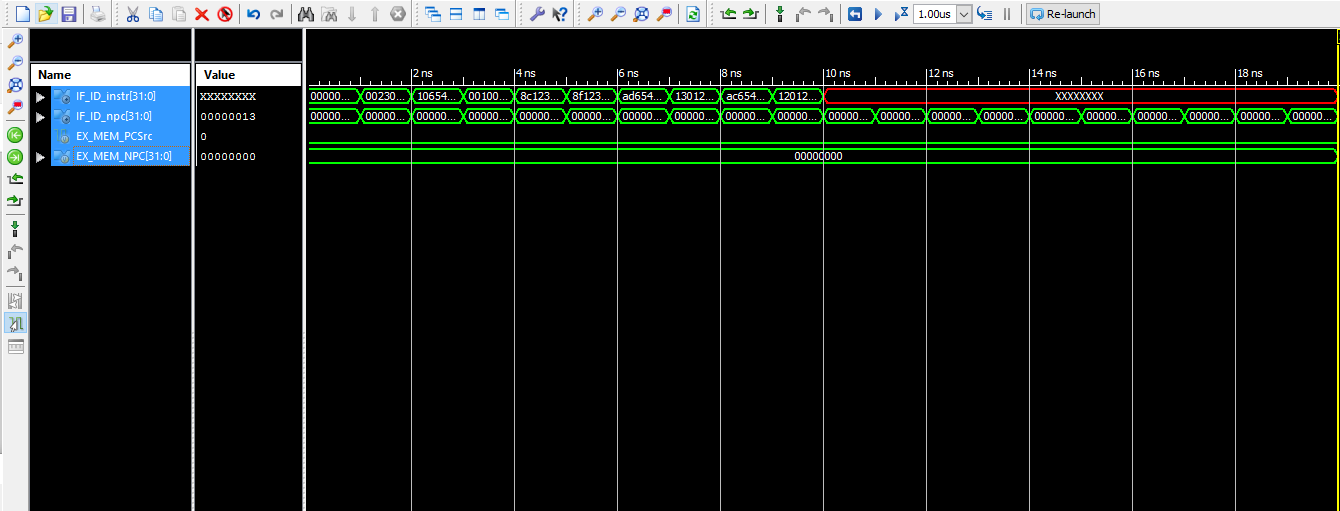
CSE 401 Computer Architecture

Winter 2018

Michael Smith

Hon Huynh

March 19th, 2018

****

**PIPELINE:**

// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
// -\*- Mode: Verilog -\*-  
// Filename: pipeline.v  
  
module pipeline();  
  
 wire [31:0] IF\_ID\_instr, IF\_ID\_npc;  
 wire EX\_MEM\_PCSrc; // wire EX\_MEM\_PCSrc;  
 wire [31:0] EX\_MEM\_NPC; // wire [31:0] EX\_MEM\_NPC;  
  
 I\_FETCH STAGE1(  
 .IF\_ID\_instr(IF\_ID\_instr),  
 .IF\_ID\_npc(IF\_ID\_npc),  
 .EX\_MEM\_PCSrc(EX\_MEM\_PCSrc),  
 .EX\_MEM\_NPC(EX\_MEM\_NPC) );  
  
// IDECODE  
 wire [4:0] MEM\_WB\_rd;  
 wire MEM\_WB\_regwrite;  
 wire [31:0] WB\_mux\_writedata;   
 wire [1:0] wb\_ctlout;  
 wire [2:0] m\_ctlout;  
 wire regdst, alusrc;  
 wire [1:0] aluop;   
 wire [31:0] npcout, rdata1out, rdata2out, s\_extendout;  
 wire [4:0] instrout\_2016, instrout\_1511;  
  
 IDECODE STAGE2(  
 .IF\_ID\_instrout(IF\_ID\_instr),  
 .IF\_ID\_npcout(IF\_ID\_npc),  
 .MEM\_WB\_rd(MEM\_WB\_rd),  
 .MEM\_WB\_regwrite(MEM\_WB\_regwrite),  
 .WB\_mux\_writedata(WB\_mux\_writedata),  
 .wb\_ctlout(wb\_ctlout),  
 .m\_ctlout(m\_ctlout),  
 .regdst(regdst),  
 .alusrc(alusrc),  
 .aluop(aluop),  
 .npcout(npcout),  
 .rdata1out(rdata1out),  
 .rdata2out(rdata2out),  
 .s\_extendout(s\_extendout),  
 .instrout\_2016(instrout\_2016),  
 .instrout\_1511(instrout\_1511));  
   
// EXECUTE  
 wire [1:0] wb\_ctlout\_pipe;  
 wire branch, memread, memwrite;  
 wire zero;  
 wire [31:0] alu\_result, rdata2out\_pipe;  
 wire [4:0] five\_bit\_muxout;  
   
 EXECUTE STAGE3(  
 // inputs  
 .wb\_ctl(wb\_ctlout),  
 .m\_ctl(m\_ctlout),  
 .regdst(regdst),  
 .alusrc(alusrc),  
 .aluop(aluop),  
 .npcout(npcout),  
 .rdata1(rdata1out),  
 .rdata2(rdata2out),  
 .s\_extendout(s\_extendout),  
 .instrout\_2016(instrout\_2016),  
 .instrout\_1511(instrout\_1511),  
 // outputs  
 .wb\_ctlout(wb\_ctlout\_pipe),  
 .branch(branch),   
 .memread(memread),   
 .memwrite(memwrite),  
 .zero(zero),  
 .alu\_result(alu\_result),  
 .rdata2out(rdata2out\_pipe),  
 .add\_result(EX\_MEM\_NPC),  
 .five\_bit\_muxout(five\_bit\_muxout));  
  
// MEMORY  
 wire MEM\_WB\_memtoreg;  
 wire [31:0] read\_data, mem\_alu\_result;  
  
 MEMORY STAGE4(  
 // inputs  
 .wb\_ctlout(wb\_ctlout\_pipe),  
 .branch(branch),  
 .memread(memread),  
 .memwrite(memwrite),  
 .zero(zero),  
 .alu\_result(alu\_result),  
 .rdata2out(rdata2out\_pipe),  
 .five\_bit\_muxout(five\_bit\_muxout),  
 // outputs  
 .MEM\_PCSrc(EX\_MEM\_PCSrc),  
 .MEM\_WB\_regwrite(MEM\_WB\_regwrite),  
 .MEM\_WB\_memtoreg(MEM\_WB\_memtoreg),  
 .read\_data(read\_data),  
 .mem\_alu\_result(mem\_alu\_result),  
 .mem\_write\_reg(MEM\_WB\_rd));  
   
// WRITEBACK  
 WRITEBACK STAGE5(  
 // inputs  
 .MEM\_WB\_memtoreg(MEM\_WB\_memtoreg),  
 .read\_data(read\_data),  
 .mem\_alu\_result(mem\_alu\_result),  
 // output  
 .WB\_mux\_writedata(WB\_mux\_writedata));   
 endmodule // pipeline

**LAB1**

**If\_id:**

// Class: CSE 401 Computer Architecture

// Term: WINTER2018

// Name(s): Michael Smith

// Hon Huynh

// Lab #1: Instruction Fetch Stage

// -\*- Mode: Verilog -\*-

// Filename: if\_id.v

module if\_id(

output reg [31:0] instrout, // Output of IF/ID Instruction Register

npcout, // Output of IF/ID NPC Register

input wire [31:0] instr, // Input of IF/ID Instruction Register

npc // Input of IF/ID NPC Register

);

initial begin

instrout <= 32'b0;

npcout <= 32'b0;

end

always @\* begin

#1;

instrout <= instr;

npcout <= npc;

end

endmodule // if\_id

**Incrementer:**

// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
// Lab #1: Instruction Fetch Stage  
// -\*- Mode: Verilog -\*-  
// Filename: incr.v  
  
module incrementer (  
 input wire [31:0] pcin, // Input of incrementer  
 output wire [31:0] pcout // Output of incrementer  
 );  
   
 assign pcout = pcin + 1; // Increment PC by 1  
endmodule // incrementer

**Memory:**

// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
// Lab #1: Instruction Fetch Stage  
// -\*- Mode: Verilog -\*-  
// Filename: mem.v

module memory (  
 output reg [31:0] data, // Output of Instruction Memory  
 input wire [31:0] addr // Input of Instruction Memory  
 );  
  
// Register Declarations  
 reg [31:0] MEM[0:127]; // 128 words of 32-bit memory  
  
 integer i;  
  
// Initialize Registers  
 initial begin   
 $readmemb("etc/risc.txt",MEM);  
 end  
   
 always @(addr) begin  
 data <= MEM[addr];  
 end  
endmodule // memory

**Pc\_mod:**

// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
// Lab #1: Instruction Fetch Stage  
// -\*- Mode: Verilog -\*-  
// Filename: pc\_mod.v

module pc\_mod (  
 output reg [31:0] PC, // Output of pc\_mod  
 input wire [31:0] npc // Input of pc\_mod  
 );  
  
 initial begin  
 PC <= 0;  
 end  
 always @ ( npc) begin  
 #1 PC <= npc;  
 end  
  
endmodule // pc\_mod

**MUX:**

// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
// Lab #1: Instruction Fetch Stage  
// -\*- Mode: Verilog -\*-  
// Filename: mux.v  
  
module mux (  
 output wire [31:0] y, // Output of Multiplexer  
 input wire [31:0] a, // Input 1 of Multiplexer  
 b, // Input 0 of Multiplexer  
 input wire sel // Select Input  
 );  
   
 assign y = sel ? a : b;  
endmodule // mux

**Fetch:**

// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
// Lab #1: Instruction Fetch Stage  
// -\*- Mode: Verilog -\*-  
// Filename: ifetch.v  
  
module I\_FETCH(  
 output wire [31:0] IF\_ID\_instr,  
 output wire [31:0] IF\_ID\_npc,  
 input wire EX\_MEM\_PCSrc,  
 input wire [31:0] EX\_MEM\_NPC  
 );  
   
 //signals  
 wire [31:0] PC;  
 wire [31:0] dataout;  
 wire [31:0] npc,npc\_mux;  
  
 //instantiations  
 mux mux1 (.y(npc\_mux),   
 .a(EX\_MEM\_NPC),   
 .b(npc),   
 .sel(EX\_MEM\_PCSrc));  
   
 pc\_mod pc\_mod1 (.PC(PC),   
 .npc(npc\_mux));  
   
 memory memory1 (.data(dataout),  
 .addr(PC));  
   
 if\_id if\_id1 (.instrout(IF\_ID\_instr),  
 .npcout(IF\_ID\_npc),  
 .instr(dataout),   
 .npc(npc));  
   
 incrementer incrementer1 (.pcout(npc),   
 .pcin(PC));  
 initial begin  
 $display("Time\t PC\t npc\t dataout of MEM\t IF\_ID\_instr\t IF\_ID\_npc");  
 $monitor("%0d\t %0d\t %0d\t %h\t %h\t %0d", $time, PC, npc, dataout,IF\_ID\_instr,IF\_ID\_npc);  
 #22 $finish;  
 end   
  
endmodule // I\_FETCH

**Lab2:**

**IDECODE:**

// Class: CSE 401 Computer Architecture

// Term: WINTER2018

// Name(s): Michael Smith

// Hon Huynh

/\* IDECODE.v \*/

module IDECODE(

input wire [31:0] IF\_ID\_instrout,

input wire [31:0] IF\_ID\_npcout,

input wire [4:0] MEM\_WB\_rd,

input wire MEM\_WB\_regwrite,

input wire [31:0] WB\_mux\_writedata,

output wire [1:0] wb\_ctlout,

output wire [2:0] m\_ctlout,

output wire regdst, alusrc,

output wire [1:0] aluop,

output wire [31:0] npcout, rdata1out, rdata2out, s\_extendout,

output wire [4:0] instrout\_2016, instrout\_1511

);

// signals

wire [3:0] ctlex\_out;

wire [2:0] ctlm\_out;

wire [1:0] ctlwb\_out;

wire [31:0] readdat1, readdat2, signext\_out;

// instantiations

control control2 (

// input

.opcode(IF\_ID\_instrout[31:26]),

// outputs

.EX(ctlex\_out),

.M(ctlm\_out),

.WB(ctlwb\_out));

register register2 (

// input

.rs(IF\_ID\_instrout[25:21]),

.rt(IF\_ID\_instrout[20:16]),

.rd(MEM\_WB\_rd),

.writedata(WB\_mux\_writedata),

.regwrite(MEM\_WB\_regwrite),

// outputs

.A(readdat1),

.B(readdat2));

s\_extend s\_extend2 (

// input

.nextend(IF\_ID\_instrout[15:0]),

// output

.extend(signext\_out));

id\_ex id\_ex2 (

// inputs

.ctlwb\_out(ctlwb\_out),

.ctlm\_out(ctlm\_out),

.ctlex\_out(ctlex\_out),

.npc(IF\_ID\_npcout),

.readdat1(readdat1),

.readdat2(readdat2),

.signext\_out(signext\_out),

.instr\_2016(IF\_ID\_instrout[20:16]),

.instr\_1511(IF\_ID\_instrout[15:11]),

// outputs

.wb\_ctlout(wb\_ctlout),

.m\_ctlout(m\_ctlout),

.regdst(regdst),

.alusrc(alusrc),

.aluop(aluop),

.npcout(npcout),

.rdata1out(rdata1out),

.rdata2out(rdata2out),

.s\_extendout(s\_extendout),

.instrout\_2016(instrout\_2016),

.instrout\_1511(instrout\_1511));

endmodule // IDECODE

**Control**

// Class: CSE 401 Computer Architecture

// Term: WINTER2018

// Name(s): Michael Smith

// Hon Huynh

/\* control.v \*/

module control(

input wire [5:0] opcode,

output reg [3:0] EX,

output reg [2:0] M,

output reg [1:0] WB

);

parameter RTYPE = 6'b000000;

parameter LW = 6'b100011;

parameter SW = 6'b101011;

parameter BEQ = 6'b000100;

parameter NOP = 6'b100000;

initial begin

EX <= 4'b0000;

M <= 3'b000;

WB <= 2'b00;

end

always@\* begin

case (opcode)

RTYPE:

begin

EX <= 4'b1100;

M <= 3'b000;

WB <= 2'b10;

end

LW:

begin

EX <= 4'b0001;

M <= 3'b010;

WB <= 2'b11;

end

SW:

begin

EX <= 4'bz001;

M <= 3'b001;

WB <= 2'b0z;

end

BEQ:

begin

EX <= 4'bz010;

M <= 3'b100;

WB <= 2'b0z;

end

NOP:

begin

EX <=4'b1000;

M <=3'b000;

WB <=2'b00;

end

default: $display ("Opcode not recognized.");

endcase

end

endmodule // control

**Register**

// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
/\* reg.v \*/

module register(  
 input [4:0] rs,  
 input [4:0] rt,  
 input [4:0] rd,  
 input [31:0] writedata,  
 input regwrite,  
 output reg [31:0] A,  
 output reg [31:0] B  
 );  
  
// Register Declaration  
reg [31:0] REG [0:31]; //gives us 32 registers, each 32 bits long  
   
integer i;  
  
initial   
begin  
 A <= 0;  
 B <= 0;  
   
 // initialize our registers  
 for (i = 0; i < 32; i = i + 1)  
 REG[i] <= 0;  
end  
  
always @ \*   
begin  
 A <= REG[rs];// Assign the rs index of REG to A ;  
 B <= REG[rt];// Assign the \_\_ index of REG to B ;  
   
 // WRITE data using index rd  
 if ((rd!=0) & (regwrite==1))  
 REG[rd] <= writedata;  
end  
  
endmodule // register

**Extend**

// Class: CSE 401 Computer Architecture

// Term: WINTER2018

// Name(s): Michael Smith

// Hon Huynh

/\* s\_extend.v \*/

module s\_extend(

input wire [15:0] nextend,

output reg [31:0] extend

);

always@ \*

begin

// Replicate signed bit 16 times then cancatinate

extend <= {{16{nextend[15]}}, nextend[15:0]};

end

endmodule // s\_extend

**ID\_EX**

// Class: CSE 401 Computer Architecture

// Term: WINTER2018

// Name(s): Michael Smith

// Hon Huynh

/\* id\_ex.v \*/

module id\_ex(

input wire [1:0] ctlwb\_out,

input wire [2:0] ctlm\_out,

input wire [3:0] ctlex\_out,

input wire [31:0] npc, readdat1, readdat2, signext\_out,

input wire [4:0] instr\_2016, instr\_1511,

output reg [1:0] wb\_ctlout,

output reg [2:0] m\_ctlout,

output reg regdst, alusrc,

output reg [1:0] aluop,

output reg [31:0] npcout, rdata1out, rdata2out, s\_extendout,

output reg [4:0] instrout\_2016, instrout\_1511

);

initial

begin

// Initialize outputs to 0

wb\_ctlout <= 2'b0;

m\_ctlout <= 3'b0;

regdst <= 1'b0;

aluop <= 2'b0;

alusrc <= 1'b0;

npcout <= 32'b0;

rdata1out <= 32'b0;

rdata2out <= 32'b0;

s\_extendout <= 32'b0;

instrout\_2016 <= 5'b0;

instrout\_1511 <= 5'b0;

end

always @ \*

begin

// Inputs wired to corresponding outputs

#1;

wb\_ctlout <= ctlwb\_out;

m\_ctlout <= ctlm\_out;

regdst <= ctlex\_out[3];

aluop <= ctlex\_out[2:1];

alusrc <= ctlex\_out[0];

npcout <= npc;

rdata1out <= readdat1;

rdata2out <= readdat2;

s\_extendout <= signext\_out;

instrout\_2016 <= instr\_2016;

instrout\_1511 <= instr\_1511;

end

endmodule //id\_ex

**LAB3**

**Execute:**

// Class: CSE 401 Computer Architecture

// Term: WINTER2018

// Name(s): Michael Smith

// Hon Huynh

/\* execute.v \*/

module EXECUTE(

input wire [1:0] wb\_ctl,

input wire [2:0] m\_ctl,

input wire regdst, alusrc,

input wire [1:0] aluop,

input wire [31:0] npcout, rdata1, rdata2, s\_extendout,

input wire [4:0] instrout\_2016, instrout\_1511,

output wire [1:0] wb\_ctlout,

output wire branch, memread, memwrite,

output wire zero,

output wire [31:0] alu\_result, rdata2out, add\_result,

output wire [4:0] five\_bit\_muxout

);

// signals

wire [31:0] adder\_out, b, aluout;

wire [4:0] muxout;

wire [2:0] control;

wire aluzero;

// instantiations

adder adder1(

// output

.add\_out(adder\_out),

// inputs

.add\_in2(s\_extendout),

.add\_in1(npcout));

mux mux2(

// output

.y(b),

// inputs

.a(s\_extendout),

.b(rdata2),

.sel(alusrc));

alu alu1(

// inputs

.a(rdata1),

.b(b),

.control(control),

// outputs

.result(aluout),

.zero(aluzero));

alu\_control alu\_control1(

// inputs

.funct(s\_extendout[5:0]),

.aluop(aluop),

// output

.select(control));

bottom\_mux bottom\_mux1(

// output

.y(muxout),

// inputs

.a(instrout\_1511),

.b(instrout\_2016),

.sel(regdst));

ex\_mem ex\_mem1(

//inputs

.ctlwb\_out(wb\_ctl),

.ctlm\_out(m\_ctl),

.adder\_out(adder\_out),

.aluzero(aluzero),

.aluout(aluout),

.readdat2(rdata2),

.muxout(muxout),

// outputs

.wb\_ctlout(wb\_ctlout),

.branch(branch),

.memread(memread),

.memwrite(memwrite),

.add\_result(add\_result),

.zero(zero),

.alu\_result(alu\_result),

.rdata2out(rdata2out),

.five\_bit\_muxout(five\_bit\_muxout));

endmodule // IEXECUTE

**ADDER:**

// Class: CSE 401 Computer Architecture

// Term: WINTER2018

// Name(s): Michael Smith

// Hon Huynh

/\* adder.v \*/

module adder(

output reg [31:0] add\_out,

input [31:0] add\_in1, add\_in2

);

always@ \*

begin

add\_out <= add\_in1 + add\_in2;

end

endmodule // adder

**ALU**

// Class: CSE 401 Computer Architecture

// Term: WINTER2018

// Name(s): Michael Smith

// Hon Huynh

/\* alu.v \*/

module alu(

input wire [31:0] a,

input wire [31:0] b,

input wire [2:0] control,

output reg [31:0] result,

output wire zero

);

parameter ALUadd = 3'b010,

ALUsub = 3'b110,

ALUand = 3'b000,

ALUor = 3'b001,

ALUslt = 3'b111;

// Handles negative inputs

wire sign\_mismatch;

assign sign\_mismatch = a[31] ^ b[31];

initial

result <= 0;

always@\*

case(control)

ALUadd: result = a + b;

ALUsub: result = a - b;

ALUand: result = a && b;

ALUor: result = a || b;

ALUslt: result = a < b ? 1 - sign\_mismatch : 0 + sign\_mismatch;

default: result = 32'bX;

endcase

assign zero = (result==0) ? 1'b1 : 1'b0;

/\* If result is 0, assign it true (1).

Otherwise, assign it false (0) \*/

endmodule // alu

**Bottom MUX:**

// Class: CSCI 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
/\* bottom\_mux \*/

module bottom\_mux(  
 output wire [4:0] y, // Output of Multiplexer  
 input wire [4:0] a, // Input 1 of Multiplexer  
 b, // Input 0 of Multiplexer  
 input wire sel // Select Input  
 );  
   
 assign y = sel ? a : b;  
endmodule // bottom\_mux

**EX\_MEM:**

// Class: CSCI 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
/\* ex\_mem.v \*/

module ex\_mem(  
 input wire [1:0] ctlwb\_out,  
 input wire [2:0] ctlm\_out,  
 input wire [31:0] adder\_out,  
 input wire aluzero,  
 input wire [31:0] aluout, readdat2,  
 input wire [4:0] muxout,  
 output reg [1:0] wb\_ctlout,  
 output reg branch, memread, memwrite,  
 output reg [31:0] add\_result,  
 output reg zero,  
 output reg [31:0] alu\_result, rdata2out,  
 output reg [4:0] five\_bit\_muxout  
 );  
  
 initial begin  
 // Initialize outputs to 0  
 wb\_ctlout <= 0;   
 branch <= 0; memread <= 0; memwrite <= 0;  
 add\_result <= 0;  
 zero <= 0;  
 alu\_result <= 0; rdata2out <= 0;  
 five\_bit\_muxout <= 0;  
 end  
  
 always@\* begin  
 // Inputs wired to corresponding outputs  
 #1;  
 wb\_ctlout <= ctlwb\_out;  
 branch <= ctlm\_out[2];   
 memread <= ctlm\_out[1];  
 memwrite <= ctlm\_out[0];   
 add\_result <= adder\_out;   
 zero <= aluzero;  
 alu\_result <= aluout;  
 rdata2out <= readdat2;  
 five\_bit\_muxout <= muxout;  
 end  
endmodule // ex\_mem

**LAB4**

**Memory:**

// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
/\* MEMORY.v \*/

module MEMORY(  
 input wire [1:0] wb\_ctlout,  
 input wire branch, memread, memwrite,  
 input wire zero,  
 input wire [31:0] alu\_result, rdata2out,  
 input wire [4:0] five\_bit\_muxout,  
 output wire MEM\_PCSrc,  
 output wire MEM\_WB\_regwrite, MEM\_WB\_memtoreg,  
 output wire [31:0] read\_data, mem\_alu\_result,  
 output wire [4:0] mem\_write\_reg  
 );  
  
 // signals  
 wire [31:0] read\_data\_in;  
   
 // instantiations  
 AND AND\_4(  
 // inputs  
 .membranch(branch),  
 .zero(zero),  
 // output   
 .PCSrc(MEM\_PCSrc));  
   
 data\_memory data\_memory4(  
 //inputs  
 .addr(alu\_result),  
 .write\_data(rdata2out),  
 .memwrite(memwrite),  
 .memread(memread),  
 // output  
 .read\_data(read\_data\_in));  
   
 mem\_wb mem\_wb4(  
 //inputs  
 .control\_wb\_in(wb\_ctlout),  
 .read\_data\_in(read\_data\_in),  
 .alu\_result\_in(alu\_result),  
 .write\_reg\_in(five\_bit\_muxout),  
 // outputs  
 .regwrite(MEM\_WB\_regwrite),  
 .memtoreg(MEM\_WB\_memtoreg),  
 .read\_data(read\_data),  
 .mem\_alu\_result(mem\_alu\_result),  
 .mem\_write\_reg(mem\_write\_reg));  
endmodule // MEMORY

**AND\_4:**

`timescale 1ns / 1ps  
// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
/\* and.v \*/

module AND(  
 input wire membranch, zero,  
 output wire PCSrc  
 );  
  
 assign PCSrc = membranch && zero;  
endmodule // and

**DATA\_MEM:**

// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh

/\* data\_memory.v \*/

module data\_memory (  
 output reg [31:0] read\_data,

input wire [31:0] addr, // Memory Address  
 input wire [31:0] write\_data, // Memory Address Contents  
 input wire memwrite, memread  
 );  
  
 // Register Declaration  
 reg [31:0] DMEM[0:255]; // 256 words of 32-bit memory  
  
 integer i;  
   
 initial begin  
 read\_data <= 0;  
   
 // Initialize DMEM[0-5] from data.txt  
 $readmemb("etc/data.txt",DMEM);  
   
 end  
   
 always@(addr)begin  
 if ((memread == 1) && (memwrite == 0))  
 // Read data  
 read\_data <= DMEM[addr];  
 if ((memwrite == 1) && (memwrite == 0))  
 // Write data  
 DMEM[addr] <= write\_data;  
 end  
endmodule // data\_memory

**MEM\_WB:**

// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
/\* mem\_wb.v \*/

module mem\_wb(  
 input wire [1:0] control\_wb\_in,  
 input wire [31:0] read\_data\_in, alu\_result\_in,  
 input wire [4:0] write\_reg\_in,  
 output reg regwrite, memtoreg,  
 output reg [31:0] read\_data, mem\_alu\_result,  
 output reg [4:0] mem\_write\_reg  
 );  
  
initial begin  
 // Initialize outputs to 0  
 regwrite <= 0;  
 memtoreg <= 0;  
 read\_data <= 0;  
 mem\_alu\_result <= 0;  
 mem\_write\_reg <= 0;  
   
end  
  
always@\* begin  
 // Inputs wired to corresponding outputs  
 #1;  
 regwrite <= control\_wb\_in[1];  
 memtoreg <= control\_wb\_in[0];  
 read\_data <= read\_data\_in;  
 mem\_alu\_result <= alu\_result\_in;  
 mem\_write\_reg <= write\_reg\_in;  
end  
endmodule // mem\_wb

**LAB5**

**WRITEBACK:**

// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
/\* WRITEBACK.v \*/

module WRITEBACK(  
 input wire MEM\_WB\_memtoreg,  
 input wire [31:0] read\_data, mem\_alu\_result,  
 output wire [31:0] WB\_mux\_writedata  
 );  
   
 // instantiation  
 mux mux3(  
 // output  
 .y(WB\_mux\_writedata),  
 // inputs  
 .sel(MEM\_WB\_memtoreg),  
 .a(read\_data),  
 .b(mem\_alu\_result)   
 );  
endmodule // WRITEBACK

**REG\_TEST.v:**

`timescale 1ns / 1ps  
// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
// Mode: reg\_test

module reg\_test;  
 wire [31:0] data\_rs, data\_rt;  
  
 reg [4:0] addr\_rs, addr\_rt, write\_addr;  
 reg [31:0] write\_data;  
 reg regWrite;  
  
 initial  
 begin  
 addr\_rs = 0;  
 addr\_rt = 1;  
 regWrite = 0;  
  
 #1;  
 addr\_rs = 2;  
 addr\_rt = 3;  
 write\_addr = 3;  
 write\_data = 'b100;  
   
 #1;  
 addr\_rs = 4;  
 addr\_rt = 5;  
 regWrite = 1;  
  
 #1;   
 regWrite = 0;  
 addr\_rt = 3;  
  
 #1;  
 addr\_rs = 6;  
 regWrite = 1;  
 write\_addr = 6;  
 write\_data = 'b100;  
   
 #1;  
 $finish;  
 end  
  
 register regfile(  
 .rs(addr\_rs),  
 .rt(addr\_rt),  
 .rd(write\_addr),  
 .writedata(write\_data),  
 .regwrite(regWrite),  
 .A(data\_rs), //output  
 .B(data\_rt) // output  
 );  
   
endmodule

**REG.v:**

// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
/\* reg.v \*/

module register(  
 input [4:0] rs,  
 input [4:0] rt,  
 input [4:0] rd,  
 input [31:0] writedata,  
 input regwrite,  
 output reg [31:0] A,  
 output reg [31:0] B  
 );  
  
// Register Declaration  
reg [31:0] REG [0:31]; //gives us 32 registers, each 32 bits long  
   
integer i;  
  
initial   
begin  
 A <= 0;  
 B <= 0;  
   
 // initialize our registers  
 for (i = 0; i < 32; i = i + 1)  
 REG[i] <= 0;  
end  
  
always @ \*   
begin  
 A <= REG[rs];// Assign the rs index of REG to A ;  
 B <= REG[rt];// Assign the \_\_ index of REG to B ;  
   
 // WRITE data using index rd  
 if ((rd!=0) & (regwrite==1))  
 REG[rd] <= writedata;  
end  
  
endmodule // register

**TEST5MUX.v:**

`timescale 1ns / 1ps  
// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
// Mode: test\_5bitmux

module test\_5bitmux();  
  
// Wire Ports  
 wire [4:0] Y;  
   
// Register Declarations  
 reg [4:0] A, B;  
 reg sel;  
   
 bottom\_mux mux5 (.y(Y), .a(A), .b(B), .sel(sel)); // instantiate the mux  
   
 initial begin  
 A = 5'b01010;  
 B = 5'b10101;  
 sel = 1'b1;  
 #10;  
 A = 5'b00000;  
 #10;  
 sel = 1'b1;  
 #10;  
 B = 5'b11111;  
 #5;  
 A = 5'b00101;  
 #5;  
 sel = 1'b0;  
 B = 5'b11101;  
 #5;  
 sel = 1'bx;  
 end  
   
 always @(A or B or sel)  
 begin  
 #1;  
 $display("At t = %0d sel = %b A = %b B = %b Y = %b", $time, sel, A, B, Y);  
 end  
  
  
  
endmodule // test

**BUTTOM\_MUX.v:**

// Class: CSCI 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
/\* bottom\_mux \*/

module bottom\_mux(  
 output wire [4:0] y, // Output of Multiplexer  
 input wire [4:0] a, // Input 1 of Multiplexer  
 b, // Input 0 of Multiplexer  
 input wire sel // Select Input  
 );  
   
 assign y = sel ? a : b;  
endmodule // bottom\_mux

**Test\_alu:**

`timescale 1ns / 1ps  
// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
// Mode: test\_alu   
module test\_alu();  
  
// Register Declarations  
 reg [31:0] A, B;  
 reg [2:0] control;  
   
// Wire Ports  
 wire [31:0] result;  
 wire zero;  
   
 initial begin  
 A <= 'b1010; //10  
 B <= 'b0111; //7  
 control <= 3'b011;  
 $display ("A = %b\tB = %b", A, B);  
 $monitor ("ALUOp = %b\tresult = %b", control, result);  
 #1;  
 control <= 'b100;  
 #1;  
 control <= 'b010;  
 #1;  
 control <= 'b111; // set on less than  
 #1;  
 control <= 'b011;  
 #1;  
 control <= 'b110;  
 #1;  
 control <= 'b001;  
 #1;  
 control <= 'b000;  
 #1;  
 $finish;  
   
 end  
   
 alu alu1 (.result(result), .zero(zero), .a(A), .b(B), .control(control));  
endmodule

**ALU.V:**

// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
/\* alu.v \*/

module alu(  
 input wire [31:0] a,  
 input wire [31:0] b,  
 input wire [2:0] control,  
 output reg [31:0] result,  
 output wire zero  
 );  
  
 parameter ALUadd = 3'b010, // Get these values from  
 ALUsub = 3'b110, // Figure 3.2 in Lab Manual  
 ALUand = 3'b000,  
 ALUor = 3'b001,  
 ALUslt = 3'b111;  
   
   
// Handles negative inputs  
 wire sign\_mismatch;  
 assign sign\_mismatch = a[31] ^ b[31];   
   
 initial  
 result <= 0;  
   
 always@\*  
 case(control)  
 ALUadd: result = a + b;  
 ALUsub: result = a - b;  
 ALUand: result = a && b;  
 ALUor: result = a || b;  
 ALUslt: result = a < b ? 1 - sign\_mismatch : 0 + sign\_mismatch;   
 default: result = 32'bX;  
 endcase  
   
 assign zero = (result==0) ? 1'b1 : 1'b0;  
 /\* If result is 0, assign it true (1).  
 Otherwise, assign it false (0) \*/   
endmodule // alu

**TEST\_ALUCONTROL.V:**

`timescale 1ns / 1ps  
// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
// Mode: test\_alucontrol

module test\_alucontrol();  
 // Wire Ports  
 wire [2:0] select;  
   
 // Register Declarations  
 reg [1:0] alu\_op;  
 reg [5:0] funct;  
   
 alu\_control alu\_control1 (.select(select), .funct(funct), .aluop(alu\_op) );  
   
 initial begin  
 alu\_op = 2'b00;  
 funct = 6'b100000;  
 $monitor ("ALUOp = %b\tfunct = %b\tselect = %b", alu\_op, funct, select);  
 #1  
 alu\_op = 2'b01;  
 funct = 6'b100000;  
 #1  
 alu\_op = 2'b10;  
 funct = 6'b100000;  
 #1  
 funct = 6'b100010;  
 #1  
 funct = 6'b100100;  
 #1  
 funct = 6'b100101;  
 #1  
 funct = 6'b101010;  
 #1  
 $finish;  
 end  
  
endmodule //test

**ALU\_CONTROL:**

// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
/\* alu\_control.v \*/

module alu\_control(  
 input wire [5:0] funct,  
 input wire [1:0] aluop,  
 output reg [2:0] select  
 );  
   
 parameter Rtype = 2'b10,  
 Radd = 6'b100000,  
 Rsub = 6'b100010,  
 Rand = 6'b100100,  
 Ror = 6'b100101,  
 Rslt = 6'b101010;  
   
 parameter lwsw = 2'b00,  
 Itype = 2'b01,  
 xis = 6'bXXXXXX;  
   
   
 parameter ALUadd = 3'b010,  
 ALUsub = 3'b110,  
 ALUand = 3'b000,  
 ALUor = 3'b001,  
 ALUslt = 3'b111;  
   
 parameter unknown = 2'b11,  
 ALUx = 3'b011;  
   
 initial  
 select <= 0;  
   
 always@\* begin  
 if (aluop == Rtype)  
 begin  
 case(funct)  
 Radd: select <= ALUadd;  
 Rsub: select <= ALUsub;  
 Rand: select <= ALUand;  
 Ror: select <= ALUor;  
 Rslt: select <= ALUslt;  
 default: select <= ALUx;  
 endcase  
 end  
   
 else if (aluop == lwsw)  
 begin  
 select <= ALUadd;  
 end  
   
 else if (aluop == Itype)  
 begin   
 select <= ALUsub;  
 end  
   
 else if (aluop == unknown)  
 begin   
 select <= ALUx;  
 end  
   
 else  
 begin  
 select <= select;   
 end   
 end  
endmodule // alu\_control

**TEST\_INCR.V:**

`timescale 1ns / 1ps  
// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
// Mode: test\_incr

module test\_incr();  
  
// Port Wires  
 wire [31:0] IncrOut;  
   
// Register Declarations  
 reg [31:0] A;  
   
 incrementer incr (  
 .pcin(A),  
 .pcout(IncrOut)  
 ); // instantiate the incrementer  
   
 initial begin  
 #10;  
 A = 3;  
 #10;  
 A = 15;  
 #10;  
 A = 64;  
 #5;  
 end  
   
 always @(A)  
 #1 $display("Time = %0d\tA=%0d\tIncrOut=%0d",$time, A, IncrOut);  
  
endmodule // test

**INCREMENTER:**

// Class: CSE401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
// Lab #1: Instruction Fetch Stage  
// -\*- Mode: Verilog -\*-  
// Filename: incr.v

module incrementer (  
 input wire [31:0] pcin, // Input of incrementer  
 output wire [31:0] pcout // Output of incrementer  
 );  
   
 assign pcout = pcin + 1; // Increment PC by 1  
endmodule // incrementer

**TEST\_MEM:**

`timescale 1ns / 1ps  
// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
// Mode: test\_mem   
// include "registerfile.v"  
  
module test\_mem();  
 wire [31:0] data\_out;  
  
 reg [31:0] address;  
 reg [31:0] data;  
 reg mem\_read, mem\_write;  
  
 initial  
 begin  
 // $dumpfile("reg\_test.vcd");  
 // $dumpvars(0, reg\_test);  
  
 mem\_read = 1'b1;  
 mem\_write = 1'b0;  
 address = 32'b00000001;  
  
 #1;  
 mem\_read = 1'b0;  
 mem\_write = 1'b1;  
 address = 32'b00000001;  
 data = ~address;  
  
 #1;  
 mem\_read = 1;  
 mem\_write = 0;  
 address = 32'b00000010;  
  
 #1;  
 mem\_read = 1;  
 mem\_write = 1;  
 address = 32'b00000010;  
 data = ~address;  
  
 #1;  
 mem\_read = 1;  
 mem\_write = 0;  
 address = 32'b00000100;  
  
 #1;  
 mem\_read = 0;  
 mem\_write = 1;  
 address = 32'b00000100;  
 data = ~address;  
   
 #1;  
 mem\_read = 1;  
 mem\_write = 0;  
 address = 32'b00001000;  
  
 #1;  
 mem\_read = 1;  
 mem\_write = 1;  
 address = 32'b00001000;  
 data = ~address;  
   
 #1;  
 $finish;  
 end  
 data\_memory data\_memory1(  
 .address(address),  
 .write\_data(data),  
 .memread(mem\_read),  
 .memwrite(mem\_write),  
 .read\_data(data\_out));  
endmodule

**DATA\_MEMORY:**

// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
/\* data\_memory.v \*/

module data\_memory (  
 output reg [31:0] read\_data, // Output of Memory Address Contents  
 input wire [31:0] addr, // Memory Address  
 input wire [31:0] write\_data, // Memory Address Contents  
 input wire memwrite, memread  
 );  
  
 // Register Declaration  
 reg [31:0] DMEM[0:255]; // 256 words of 32-bit memory  
  
 integer i;  
   
 initial begin  
 read\_data <= 0;  
   
 // Initialize DMEM[0-5] from data.txt  
 $readmemb("etc/data.txt",DMEM);  
   
 end  
   
 always@(addr)begin  
 if ((memread == 1) && (memwrite == 0))  
 // Read data  
 read\_data <= DMEM[addr];  
 if ((memwrite == 1) && (memwrite == 0))  
 // Write data  
 DMEM[addr] <= write\_data;  
 end  
endmodule // data\_memory

**TEST\_MUX:**

`timescale 1ns / 1ps  
// Class: CSE 401 Computer Architecture  
// Term: WINTER2018  
// Name(s): Michael Smith  
// Hon Huynh  
// Mode: test\_mux   
module test\_mux();  
  
// Wire Ports  
 wire [31:0] Y;  
   
// Register Declarations  
 reg [31:0] A, B;  
 reg sel;  
   
 mux mux1 (  
 .a(A),  
 .b(B),  
 .sel(sel),  
 .y(Y)  
 ); //instantiate the mux  
   
 initial begin  
 A = 32'hAAAAAAAA;  
 B = 32'h55555555;  
 sel = 1'b1;  
 #10;  
 A = 32'h00000000;  
 #10;  
 sel = 1'b1;  
 #10;  
 B = 32'hFFFFFFFF;  
 #5;  
 A = 32'hA5A5A5A5;  
 #5;  
 sel = 1'b0;  
 B = 32'hDDDDDDDD;  
 #5;  
 sel = 1'bx;  
 end  
   
 always @(A or B or sel)  
 #1 $display("At t = %0d sel = %b A = %h B = %h Y = %h", $time, sel, A, B, Y);  
   
endmodule // test