Rachael Wilson

Final Project Compen 331

Section One

Due 5/8/2025

Abstract:

As detailed in the Project instructions provided this project puts work from labs 4 and 5 together. The project uses pipelining, circuits of the instruction fetch stage, circuits of the instruction decode stage, circuits of the execution stage, circuits of the memory access stage, circuits of the write back stage, and circuits for forwarding. These are all detailed in labs 4 and 5 with the forwarding being detailed in lab 5 honors option. To update my work from lab 5 I changed the control unit, Alu, all multiplexers, added two multiplexers for forwarding, added registers to register file, and added instructions to the instruction memory. To update the control unit I needed to add how forwarding works I did this by watching the help video and learning to update forward A when rs is changed and update forward B when rt is changed. I decided to add the two multiplexors because the video told me to also and put what the video provided for which stages do what using case statements. For the ALU I realized that I had never added subtract, or, xor, store word, and and, so I added the alus for that. I also updated the control unit to do subtract, or, xor, store word, and and. I added the provided registers to register file and instructions to instruction memory. I updated the multiplexors to a one line within the always loops because I learned of a simpler way to do multiplexers when reviewing the slides. I also updated the datapath to have an instance of the two multiplexers I added and updated testbench’s wires and inputs so that I could check the waveform provided. To check my work, I looked at the provided waveform.

Verilog design code:

`timescale 1ns / 1ps

# module datapath(

input wire clock,

output wire [31:0] pc,

output wire ewreg,

output wire em2reg,

output wire ewmem,

output wire [3:0] ealuc,

output wire ealuimm,

output wire [4:0] edestReg,

output wire [31:0] eqa,

output wire [31:0] eqb,

output wire [31:0] eimm32,

output wire [31:0] dinstOut,

output wire wwreg,

output wire wm2reg,

output wire [4:0] wdestReg,

output wire [31:0] wr,

output wire [31:0] wdo,

output wire mwreg,

output wire mm2reg,

output wire mwmem,

output wire [4:0] mdestReg,

output wire [31:0] mr,

output wire [31:0] mqb,

output wire [31:0] qa,

output wire [31:0] qb,

output wire [1:0] fwda,

output wire [1:0] fwdb,

output wire [31:0] regFwda,

output wire [31:0] regFwdb

//output wire [31:0] mdo,

//output wire [31:0] b,

//output wire [31:0] r,

);

// wires for the inputs of modules (commented the module it came from):

wire [31:0] nextPc; // program\_counter

wire [31:0] instOut; // ifid\_pipeline\_register

wire [5:0] op; // control\_unit

wire [5:0] func; // countrol\_unit

wire [4:0] rt; // regrt\_multiplexer, register\_file

wire [4:0] rd; // regrt\_multiplexer

wire [4:0] rs; // register\_file

wire [15:0] imm; // immediate\_extender

wire wreg; // idexe\_pipeline\_register

wire m2reg; // idexe\_pipeline\_register

wire wmem; // idexe\_pipeline\_register

wire [3:0] aluc; // idexe\_pipeline\_register

wire aluimm; // idexe\_pipeline\_register

wire [4:0] destReg; // idexe\_pipeline\_register

//wire [31:0] qa; // idexe\_pipeline\_register

//wire [31:0] qb; // idexe\_pipeline\_register

wire [31:0] imm32; // idexe\_pipeline\_register

//wire [31:0] wbData; // register\_file

wire [31:0] mdo;

wire [31:0] b;

wire [31:0] r;

wire regrt;

//wire fwda;

//wire fwdb;

wire [31:0] wbData;

// continuous assignments

assign op = dinstOut[31:26]; // first 6 bits of dinstOut

assign rs = dinstOut[25:21]; // next 5 bits after op

assign rt = dinstOut[20:16]; // next 5 bits after rt

assign rd = dinstOut[15:11]; // next 5 bits after rt

assign func = dinstOut[5:0]; // last 6 bits of dinstOut

assign imm = dinstOut[15:0]; // last 16 bits of dinstOut

// initiate instance of all the modules:

program\_counter program\_counter\_inst (.nextPc(nextPc), .clock(clock), .pc(pc));

instruction\_memory instruction\_memory\_inst (.pc(pc), .instOut(instOut));

pc\_adder pc\_adder\_inst (.pc(pc), .nextPc(nextPc));

ifid\_pipeline\_register ifid\_pipeline\_register\_inst (.instOut(instOut), .clock(clock), .dinstOut(dinstOut));

control\_unit control\_unit\_inst (.op(op), .func(func), .rs(rs), .rt(rt), .mdestReg(mdestReg), .mm2reg(mm2reg),

.mwreg(mwreg), .edestReg(edestReg), .em2reg(em2reg), .ewreg(ewreg), .wreg(wreg),

.m2reg(m2reg), .wmem(wmem), .aluc(aluc), .aluimm(aluimm), .fwda(fwda), .fwdb(fwdb),

.regrt(regrt));

regrt\_multiplexer regrt\_multiplexer\_inst (.rt(rt), .rd(rd), .regrt(regrt), .destReg(destReg));

fwda\_multiplexer fwda\_multiplexer\_inst (.fwda(fwda), .qa(qa), .r(r), .mr(mr), .mdo(mdo), .regFwda(regFwda));

fwdb\_multiplexer fwdb\_multiplexer\_inst (.fwdb(fwdb), .qb(qb), .r(r), .mr(mr), .mdo(mdo), .regFwdb(regFwdb));

register\_file register\_file\_inst (.rs(rs), .rt(rt), .wdestReg(wdestReg), .wbData(wbData), .wwreg(wwreg), .clock(clock),

.qa(qa), .qb(qb));

immediate\_extender immediate\_extender\_inst (.imm(imm), .imm32(imm32));

idexe\_pipeline\_register idexe\_pipeline\_register\_inst (.wreg(wreg), .m2reg(m2reg), .wmem(wmem), .aluc(aluc),

.aluimm(aluimm), .destReg(destReg), .regFwda(regFwda), .regFwdb(regFwdb), .imm32(imm32), .clock(clock),

.ewreg(ewreg), .em2reg(em2reg), .ewmem(ewmem), .ealuc(ealuc), .ealuimm(ealuimm),

.edestReg(edestReg), .eqa(eqa), .eqb(eqb), .eimm32(eimm32));

alu\_multiplexer alu\_multiplexer\_inst (.eqb(eqb), .eimm32(eimm32), .ealuimm(ealuimm), .b(b));

alu alu\_inst (.eqa(eqa), .b(b), .ealuc(ealuc), .r(r));

exe\_mem\_pipeline\_register exe\_mem\_pipeline\_register\_inst (.ewreg(ewreg), .em2reg(em2reg), .ewmem(ewmem), .edestReg(edestReg),

.r(r), .eqb(eqb), .clock(clock), .mwreg(mwreg), .mm2reg(mm2reg), .mwmem(mwmem), .mdestReg(mdestReg),

.mr(mr), .mqb(mqb));

data\_memory data\_memory\_inst (.mr(mr), .mqb(mqb), .mwmem(mwmem), .clock(clock), .mdo(mdo));

mem\_wb\_pipeline\_register mem\_wb\_pipeline\_register\_inst (.mwreg(mwreg), .mm2reg(mm2reg), .mdestReg(mdestReg), .mr(mr), .mdo(mdo), .clock(clock),

.wwreg(wwreg), .wm2reg(wm2reg), .wdestReg(wdestReg), .wr(wr), .wdo(wdo));

writeback\_multiplexer writeback\_multiplexer\_inst (.wr(wr), .wdo(wdo), .wm2reg(wm2reg), .wbData(wbData));

endmodule

# module program\_counter(

input wire [31:0] nextPc,

input wire clock,

output reg [31:0] pc

);

initial pc <= 32'd100;

// when clock edge is positive assign pc value to nextPc

always @ (posedge clock) begin

pc <= nextPc; // assign pc to nextPc

end

endmodule

# module instruction\_memory(

input wire [31:0] pc,

output reg [31:0] instOut

);

reg [31:0] memory [63:0]; // create memory register

initial begin // assume $1 has a value of 0

memory[25] = {// add $3, $1, $2

6'b000000, // op = r type = 000000

5'b00001, // rs = 00001

5'b00010, // rt = 00010

5'b00011, // rd = $3 = 00011

5'b00000, // shamt = 00000

6'b100000 // funct = add = 100000

};

memory[26] = {// sub $4, $9, $3

6'b000000, // op = r type = 000000

5'b01001, // rs = 01001

5'b00011, // rt = 00011

5'b00100, // rd = $4 = 00100

5'b00000, // shamt = 00000

6'b100010 // funct = sub = 100010

};

memory[27] = {// or $5, $3, $9

6'b000000, // op = r type = 000000

5'b00011, // rs = 00001

5'b01001, // rt = 01001

5'b00101, // rd = $5 = 00101

5'b00000, // shamt = 00000

6'b100101 // funct = or = 100101

};

memory[28] = {// xor $6, $3, $9

6'b000000, // op = r type = 000000

5'b00011, // rs = 00011

5'b01001, // rt = 01001

5'b00110, // rd = $5 = 00110

5'b00000, // shamt = 00000

6'b100110 // funct = xor = 100110

};

memory[29] = { // and $7, $3, $9

6'b000000, // op = r type = 000000

5'b00011, // rs = 00011

5'b01001, // rt = 01001

5'b00111, // rd = $7 = 000111

5'b00000, // shamt = 00000

6'b100100 // funct = and = 100100

};

end

always @(\*) begin

// update value of instOut with any signal change to value of memory at pc location

instOut = memory[pc[7:2]];

end

endmodule

# module pc\_adder(

input wire [31:0] pc,

output reg [31:0] nextPc

);

parameter four = 4;

always @(\*) begin

nextPc = pc + four; // update nextPc to pc + 4

end

endmodule

# module ifid\_pipeline\_register(

input wire [31:0] instOut,

input wire clock,

output reg [31:0] dinstOut

);

always @ (posedge clock) begin // at positive edge of clock

dinstOut <= instOut; // assign instOut to dinstOut

end

endmodule

# module control\_unit(

input wire [5:0] op, // input vector

input wire [5:0] func, //input vector

input wire [4:0] rs,

input wire [4:0] rt,

input wire [4:0] mdestReg,

input wire mm2reg,

input wire mwreg,

input wire [4:0] edestReg,

input wire em2reg,

input wire ewreg,

output reg wreg, // write register

output reg m2reg, // memory to register

output reg wmem, // write memory

output reg [3:0] aluc, // ALU control

output reg aluimm, // ALU immediate

output reg [31:0] fwda,

output reg [31:0] fwdb,

output reg regrt // register rt

);

// always loop from hint video for lab 3

always @(\*) begin

case (op)

6'b000000: // r-types

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluimm <= 0;

regrt <= 0;

case (func) // add function

6'b100000: // ADD instruction

begin // setting values of control

aluc <= 4'b0010;

end // signals for ADD instruction

endcase

end

6'b100011: // load word

begin // setting values of control signals

wreg <= 1;

m2reg <= 1;

wmem <= 0;

aluc <= 4'b0010;

aluimm <= 1;

regrt <= 1;

end // for load word instructions

endcase

case (rs)

edestReg: fwda <= 2'b01;

mdestReg: fwda <= 2'b10;

default: fwda <= 2'b00;

endcase

case (rt)

edestReg: fwdb <= 2'b01;

mdestReg: fwdb <= 2'b10;

default: fwdb <= 2'b00;

endcase

end

endmodule

# module regrt\_multiplexer(

input wire [4:0] rt,

input wire [4:0] rd,

input wire regrt,

output reg [4:0] destReg

);

always @(\*) begin

destReg <= regrt ? rt : rd;

end

endmodule

# module fwda\_multiplexer(

input wire [1:0] fwda,

input wire [31:0] qa,

input wire [31:0] r,

input wire [31:0] mr,

input wire [31:0] mdo,

output reg [31:0] regFwda

);

always @ (\*) begin

case (fwda)

2'b00: regFwda <= qa;

2'b01: regFwda <= r;

2'b10: regFwda <= mr;

2'b11: regFwda <= mdo;

endcase

end

endmodule

# module fwdb\_multiplexer(

input wire [1:0] fwdb,

input wire [31:0] qb,

input wire [31:0] r,

input wire [31:0] mr,

input wire [31:0] mdo,

output reg [31:0] regFwdb

);

always @ (\*) begin

case (fwdb)

2'b00: regFwdb <= qb;

2'b01: regFwdb <= r;

2'b10: regFwdb <= mr;

2'b11: regFwdb <= mdo;

endcase

end

endmoduleinput wire [1:0] fwdb,

input wire [31:0] qb,

input wire [31:0] r,

input wire [31:0] mr,

input wire [31:0] mdo,

output reg [31:0] regFwdb

);

always @ (\*) begin

case (fwdb)

2'b00: regFwdb <= qb;

2'b01: regFwdb <= r;

2'b10: regFwdb <= mr;

2'b11: regFwdb <= mdo;

endcase

end

endmodule

# module register\_file(

input wire [4:0] rs,

input wire [4:0] rt,

input wire [4:0] wdestReg,

input wire [31:0] wbData,

input wire wwreg,

input wire clock,

output reg [31:0] qa,

output reg [31:0] qb

);

// create register register

reg [31:0] register [0:63];

integer i;

initial begin // initalize all integers to zero

for (i = 0; i < 32; i = i + 1) begin

register[i] = 0;

end

register[0] = 32'h00000000;

register[1] = 32'hA00000AA;

register[2] = 32'h10000011;

register[3] = 32'h20000022;

register[4] = 32'h30000033;

register[5] = 32'h40000044;

register[6] = 32'h50000055;

register[7] = 32'h60000066;

register[8] = 32'h70000077;

register[9] = 32'h80000088;

register[10] = 32'h90000099;

end

always @(negedge clock) begin

if (wwreg == 1) begin

register[wdestReg] = wbData;

end

// assign rs to qa and rt to qb by expanding rs and rt

qa <= register[rs];

qb <= register[rt];

end

endmodule

# module immediate\_extender(

input wire [15:0] imm,

output reg [31:0] imm32

);

always @(\*) begin

imm32 <= {{16{imm[15]}}, imm}; // assign imm32 to imm using concatenation

end

endmodule

# module idexe\_pipeline\_register(

module idexe\_pipeline\_register(

input wire wreg,

input wire m2reg,

input wire wmem,

input wire [3:0] aluc,

input wire aluimm,

input wire [4:0] destReg,

input wire [31:0] regFwda,

input wire [31:0] regFwdb,

input wire [31:0] imm32,

input wire clock,

output reg ewreg,

output reg em2reg,

output reg ewmem,

output reg [3:0] ealuc,

output reg ealuimm,

output reg [4:0] edestReg,

output reg [31:0] eqa,

output reg [31:0] eqb,

output reg [31:0] eimm32

);

always @ (posedge clock) begin

ewreg <= wreg;

em2reg <= m2reg;

ewmem <= wmem;

ealuc <= aluc;

ealuimm <= aluimm;

edestReg <= destReg;

eqa <= regFwda;

eqb <= regFwdb;

eimm32 <= imm32;

end

endmodule

# module alu\_multiplexer(

input wire [31:0] eqb,

input wire [31:0] eimm32,

input wire ealuimm,

output reg [31:0] b

);

always @ (\*) begin

b <= ealuimm ? eimm32 : eqb;

end

endmodule

# module alu(

input wire [31:0] eqa,

input wire [31:0] b,

input wire [3:0] ealuc,

output reg [31:0] r

);

always @ (\*) begin

case(ealuc)

4'b0010: // ADD

r <= (eqa + b);

4'b0110: // SUB

r <= (eqa - b);

4'b0001: // OR

r <= (eqa | b);

4'b1000: // XOR

r <= (eqa ^ b);

4'b0000: // AND

r <= (eqa & b);

default:

r <= 0;

endcase

end

endmodule

# module exe\_mem\_pipeline\_register(

input wire ewreg,

input wire em2reg,

input wire ewmem,

input wire [4:0] edestReg,

input wire [31:0] r,

input wire [31:0] eqb,

input wire clock,

output reg mwreg,

output reg mm2reg,

output reg mwmem,

output reg [4:0] mdestReg,

output reg [31:0] mr,

output reg [31:0] mqb

);

always @ (posedge clock) begin

mwreg <= ewreg;

mm2reg <= em2reg;

mwmem <= ewmem;

mdestReg <= edestReg;

mr <= r;

mqb <= eqb;

end

endmodule

# module data\_memory(

input wire [31:0] mr,

input wire [31:0] mqb,

input wire mwmem,

input wire clock,

output reg [31:0] mdo

);

reg [31:0] memory [63:0];

initial begin

memory[0] = 32'hA00000AA;

memory[4] = 32'h10000011;

memory[8] = 32'h20000022;

memory[12] = 32'h30000033;

memory[16] = 32'h40000044;

memory[20] = 32'h50000055;

memory[24] = 32'h60000066;

memory[28] = 32'h70000077;

memory[32] = 32'h80000088;

memory[36] = 32'h90000099;

end

always @ (\*) begin

mdo <= memory[mr];

end

always @ (negedge clock) begin

if (mwmem == 1) begin

memory[mr] <= mqb;

end

end

endmodule

# module mem\_wb\_pipeline\_register(

input wire mwreg,

input wire mm2reg,

input wire [4:0] mdestReg,

input wire [31:0] mr,

input wire [31:0] mdo,

input wire clock,

output reg wwreg,

output reg wm2reg,

output reg [4:0] wdestReg,

output reg [31:0] wr,

output reg [31:0] wdo

);

always @ (posedge clock) begin

wwreg <= mwreg;

wm2reg <= mm2reg;

wdestReg <= mdestReg;

wr <= mr;

wdo <= mdo;

end

endmodule

# module writeback\_multiplexer(

input wire [31:0] wr,

input wire [31:0] wdo,

input wire wm2reg,

output reg [31:0] wbData

);

always @ (\*) begin

wbData <= wm2reg ? wdo : wr;

end

endmodule

Verilog Testbench code:

`timescale 1ns / 1ps

# module testbench();

reg clk;

// add wires from help video

wire [31:0] pc;

wire [31:0] dinstOut;

wire ewreg;

wire em2reg;

wire ewmem;

wire [3:0] ealuc;

wire ealuimm;

wire [4:0] edestReg;

wire [31:0] eqa;

wire [31:0] eqb;

wire [31:0] eimm32;

wire [31:0] dinstOut;

wire wwreg;

wire wm2reg;

wire [4:0] wdestReg;

wire [31:0] wr;

wire [31:0] wdo;

wire mwreg;

wire mm2reg;

wire mwmem;

wire [4:0] mdestReg;

wire [31:0] mr;

wire [31:0] mqb;

//wire [31:0] wbData;

wire [31:0] qa;

wire [31:0] qb;

wire [1:0] fwda;

wire [1:0] fwdb;

wire [31:0] regFwda;

wire [31:0] regFwdb;

//wire [31:0] mdo;

//wire [31:0] b;

//wire [31:0] r;

// datapath instance

datapath datapath\_instance (clk, pc, ewreg, em2reg, ewmem, ealuc, ealuimm, edestReg, eqa, eqb, eimm32, dinstOut,

wwreg, wm2reg, wdestReg, wr, wdo, mwreg, mm2reg, mwmem, mdestReg, mr, mqb,

qa, qb, fwda, fwdb, regFwda, regFwdb);

//initialize clock to zero

initial clk = 0;

//increment clock

always begin

#1;

clk = ~clk;

end

endmodule

Waveform:

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

design schematics from synthesis  
A screenshot of a computer

AI-generated content may be incorrect.

A green lines on a white background

AI-generated content may be incorrect.

I/O Planning Screenshot

A screenshot of a computer

AI-generated content may be incorrect.

Floor Planning Screenshot

A screenshot of a computer

AI-generated content may be incorrect.