Rachael Wilson

Lab 3 Compen 331

Due 3/23/2025

# Verilog Design Code:

`timescale 1ns / 1ps

module program\_counter(

input wire [31:0] nextPc,

input wire clock,

output reg [31:0] pc

);

initial pc <= 100;

// 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

memory[25] = {

6'b100011, // op

5'b00001, // rs

5'b00010, // rt

16'b0000000000000000 // offset

}; // first instruction

memory[26] = {

6'b100011, // op

5'b00001, // rs

5'b00011, // rt

16'b0000000000000100 // offset

}; // second instruction

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

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 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

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

if (regrt == 0) begin

destReg <= rd;

end

else begin

destReg <= rt;

end

end

endmodule

module register\_file(

input wire [4:0] rs,

input wire [4:0] rt,

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

end

always @(\*) begin

// 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(

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] qa,

input wire [31:0] qb,

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 <= qa;

eqb <= qb;

eimm32 <= imm32;

end

endmodule

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

);

// 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

// 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), .wreg(wreg), .m2reg(m2reg), .wmem(wmem), .aluc(aluc),

.aluimm(aluimm), .regrt(regrt));

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

register\_file register\_file\_inst (.rs(rs), .rt(rt), .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), .qa(qa), .qb(qb), .imm32(imm32), .clock(clock),

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

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

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;

//initialize clock to zero

initial clk = 0;

//increment clock

always @(\*) begin

#1 clk = ~clk;

end

// datapath instance

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

//increment clock

always @(\*) begin

#1 clk = ~clk;

end

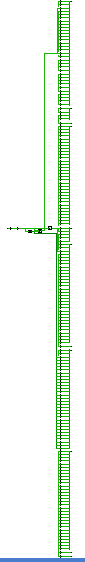
endmodule

# Waveforms:

A screen shot of a computer

AI-generated content may be incorrect.

# Design Schematics from Xilinx Sythnesis:



# Snapshot from I/O Planning:

A screenshot of a computer

AI-generated content may be incorrect.

# Snapshot from Floor Planning:

A screenshot of a computer

AI-generated content may be incorrect.