CMPEN 331 Lab5 Report – Yubo Jing

# Verilog code:

## IFID\_IDEXE\_Module.v

`timescale 1ns / 1ps

module IFID\_IDEXE\_Module(

input wire clock,

output wire [31:0] pc,dinstOut,eqa, eqb, eimm32,

output wire ewreg, em2reg,ewmem,ealuimm,

output wire [3:0] ealuc,

output wire [4:0] edestReg,

output wire mwreg, mm2reg, mwmem,

output wire [4:0] mdestReg,

output wire [31:0] mqb,

output wire [31:0] mr,

output wire wwreg, wm2reg,

output wire [4:0] wdestReg,

output wire [31:0] wr, wdo

);

wire [31:0] nextPc;

wire [31:0] instOut;

wire wreg, m2reg,wmem,aluimm,regrt;

wire [3:0] aluc;

wire [4:0] destReg;

wire [31:0] qa,qb,imm32;

wire [5:0] op, func;

wire [4:0] rs,rt,rd;

wire [15:0] imm;

wire [31:0] b;

wire [3:0] ealuc;

wire [31:0] r;

wire ealuimm;

wire [31:0] mdo;

wire [4:0] wdestReg;

wire [31:0] wbData;

wire wwreg;

PC PC\_tb(nextPc,clock,pc);

IM IM\_tb(pc, instOut);

Pc\_Adder Pc\_Adder\_tb(pc,nextPc);

IFID\_Pipeline\_Register IFID\_Pipeline\_Register\_tb(clock, instOut, dinstOut, op, func, rd, rs, rt, imm);

Control\_Unit Control\_Unit\_tb(op,func,wreg,m2reg,wmem,aluc,aluimm,regrt);

regDst\_mux regDst\_mux\_tb(rt,rd,regDst,destReg);

Register\_File Register\_File\_tb(rs,rt,wdestReg,wbData,wwreg,clock,qa,qb);

imm\_Extender imm\_Extender\_tb(imm,imm32);

IDEXE\_Pipeline\_Register IDEXE\_Pipeline\_Register\_tb(clock,wreg, m2reg, wmem, aluc, aluimm,destReg,qa,qb,imm32,ewreg,em2reg,ewmem,ealuc,ealuimm, edestReg, eqa, eqb,eimm32);

ALU ALU\_tb(eqa,b,ealuc,r);

ALUMux ALUMux\_tb(eqb,eimm,ealuimm,b);

DataMem DataMem\_tb(mr,mqb,mwmem,mdo);

EXEMEM\_Pipeline\_Register EXEMEM\_Pipeline\_Register\_tb(clock, ewreg, em2reg, ewmem,edestReg,eqb,r,mwreg,mm2reg,mwmem,mdestReg,mqb,mr);

MEMWB\_Pipeline\_Register MEMWB\_Pipeline\_Register\_tb(clock,mreg,mm2reg,mdestReg,mr,mdo,wwreg,wm2reg,wdestReg,wr,wdo);

wbMux wbMux\_tb(wr,wdo,wm2reg,wbData);

endmodule

## PC.v

`timescale 1ns / 1ps

module PC(

input[31:0] nextPc,

input clock,

output reg[31:0] pc

);

initial begin

pc = 100;

end

always@(posedge clock) begin

pc = nextPc;

end

endmodule

## IM.v

`timescale 1ns / 1ps

module IM(

input[31:0] pc,

output reg[31:0] instOut

);

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

initial begin

// memory[25] = { //lw I-type

// 6'b100011, //lw op

// 5'b00010, //rs -> $v0 -> 2

// 5'b00001, //rt -> $at -> 1

// 16'b0000000000000000 //offset = 0

// };

memory[25] = 32'h8C220000;

memory[26] = 32'h8C230004;

memory[27] = 32'h8C240008;

memory[28] = 32'h8C25000C;

memory[29] = 32'h004A3020;

// memory[26] = { //lw I-type

// 6'b100011, //lw op

// 5'b00011, //rs -> $v1 -> 3

// 5'b00001, //rt -> $at -> 1

// 16'b0000000000000100 //offset = 4

// };

end

always@(\*)begin

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

end

endmodule

## Pc\_Adder.v

`timescale 1ns / 1ps

module Pc\_Adder(

input[31:0] pc,

output reg[31:0] nextPc

);

always@(\*)begin

nextPc <= pc + 32'b100;

end

endmodule

## IFID\_Pipeline\_Register.v

`timescale 1ns / 1ps

module IFID\_Pipeline\_Register(

input clock,

input [31:0] instOut,

output reg [31:0] dinstOut,

output reg [5:0] op,

output reg [5:0] func,

output reg [4:0] rd, rs, rt,

output reg [15:0] imm

);

always@(posedge clock) begin

dinstOut <= instOut;

op = instOut[31:26];

func = instOut[5:0];

rs = instOut[25:21];

rt = instOut[20:16];

rd = instOut[15:11];

imm = instOut[15:0];

end

endmodule

## Control\_Unit.v

`timescale 1ns / 1ps

module Control\_Unit(

input[5:0] op,

input[5:0] func,

output reg wreg, //wreg

output reg m2reg, //m2reg

output reg wmem, //wmem

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

output reg aluimm, //aluimm

output reg regrt //regrt

);

always@(\*) begin

case (op)

6'b000000: // r-types

begin

case (func)

6'b100000: //ADD Instruction

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0010;

aluimm <= 0; //no sign extend

regrt <= 0; //write in rd

end

endcase

end

6'b100011: //LW

begin

wreg <= 1;

m2reg <= 1;

wmem <= 0;

aluc <= 4'b0010;

aluimm <= 1; //sign extend

regrt <= 1; //write in rt

end

endcase

end

endmodule

## regDst\_mux.v

`timescale 1ns / 1ps

module regDst\_mux(

input[4:0] rt,

input[4:0] rd,

input regDst,

output reg[4:0] destReg

);

always@(\*)begin

if(!regDst)begin

destReg <= rd;

end

else begin

destReg <= rt;

end

end

endmodule

## Register\_File.v

`timescale 1ns / 1ps

module Register\_File(

input[4:0] rs,

input[4:0] rt,

input[4:0] wdestReg,

input[31:0] wbData,

input wwreg,

input clock,

output reg[31:0] qa,

output reg[31:0] qb

);

reg [31:0] registers [0:31];

integer i;

initial begin

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

registers[i]=0;

end

end

always@(\*) begin

qa = registers[rs];

qb = registers[rt];

end

always@(negedge clock) begin

if(wwreg == 1) begin

registers[wdestReg] = wbData;

end

end

endmodule

## imm\_Extender.v

`timescale 1ns / 1ps

module imm\_Extender(

input[15:0] imm,

output reg[31:0] imm32

);

always@(\*) begin

imm32 = {{16{imm[15]}},imm};

end

endmodule

## IDEXE\_Pipeline\_Register.v

`timescale 1ns / 1ps

module IDEXE\_Pipeline\_Register(

input clock,

input wreg, //wreg

input m2reg, //m2reg

input wmem, //wmem

input[3:0] aluc, //aluc

input aluimm, //aluimm

input[4:0] destReg,

input[31:0] qa,

input[31:0] qb,

input[31:0] imm32,

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

## ALU.v

`timescale 1ns / 1ps

module ALU(

input [31:0] eqa,

input [31:0] b,

input [3:0] ealuc,

output reg [31:0] r

);

always@(eqa, b, ealuc) begin

case(ealuc)

4'b0010: begin

r = eqa + b;

end

default: begin

r = 0;

end

endcase

end

endmodule

## ALUMux.v

`timescale 1ns / 1ps

module ALUMux(

input [31:0] eqb,

input [31:0] eimm,

input ealuimm,

output [31:0] b

);

assign b = ealuimm?eimm:eqb;

endmodule

## DataMem.v

`timescale 1ns / 1ps

module DataMem(

input [31:0] mr,

input [31:0] mqb,

input mwmem,

output reg [31:0] mdo

);

reg [31:0] Data\_memory[0:127];

initial begin

Data\_memory[0]=32'hA00000AA;

Data\_memory[1]=32'h10000011;

Data\_memory[2]=32'h20000022;

Data\_memory[3]=32'h30000033;

Data\_memory[4]=32'h40000044;

Data\_memory[5]=32'h50000055;

Data\_memory[6]=32'h60000066;

Data\_memory[7]=32'h70000077;

Data\_memory[8]=32'h80000088;

Data\_memory[9]=32'h90000099;

end

always@(\*) begin

mdo = Data\_memory[mr];

if(mwmem == 1)begin

Data\_memory[mr] = mqb;

end

end

endmodule

## EXEMEM\_Pipeline\_Register.v

`timescale 1ns / 1ps

module EXEMEM\_Pipeline\_Register(

input clock, ewreg, em2reg, ewmem,

input [4:0] edestReg,

input [31:0] eqb,

input [31:0] r,

output reg mwreg, mm2reg, mwmem,

output reg [4:0] mdestReg,

output reg [31:0] mqb,

output reg [31:0] mr

);

always@(posedge clock) begin

mwreg = ewreg;

mm2reg = em2reg;

mwmem = ewmem;

mdestReg = edestReg;

mqb = eqb;

mr = r;

end

endmodule

## MEMWB\_Pipeline\_Register.v

`timescale 1ns / 1ps

module MEMWB\_Pipeline\_Register(

input clock, mwreg, mm2reg,

input [4:0] mdestReg,

input [31:0] mr, mdo,

output reg wwreg, wm2reg,

output reg [4:0] wdestReg,

output reg [31:0] wr, wdo

);

always@(posedge clock) begin

wwreg = mwreg;

wm2reg = mm2reg;

wdestReg = mdestReg;

wr = mr;

wdo =mdo;

end

endmodule

## wbMux.v:

`timescale 1ns / 1ps

module wbMux(

input [31:0] wr, wdo,

input wm2reg,

output [31:0] wbData

);

assign wbData = wm2reg ? wdo : wr;

endmodule

# Your Verilog® Test Bench design code.

`timescale 1ns / 1ps

module testbench();

reg clock;

wire [31:0] pc,dinstOut,eqa, eqb, eimm32;

wire ewreg, em2reg,ewmem,ealuimm;

wire [3:0] ealuc;

wire [4:0] edestReg;

wire mwreg, mm2reg, mwmem;

wire [4:0] mdestReg;

wire [31:0] mqb;

wire [31:0] mr;

wire wwreg, wm2reg;

wire [4:0] wdestReg;

wire [31:0] wr, wdo;

IFID\_IDEXE\_Module IFID\_IDEXE\_Module\_tb(clock,pc,dinstOut,eqa,eqb,eimm32,ewreg,em2reg,eewmem,ealuimm,ealuc,edestReg, mwreg,mm2reg,mwmem,mdestReg,mqb,mr,wwreg,wm2reg,wdestReg,wr, wdo);

initial begin

clock = 0;

end

always#5 clock = ~clock;

endmodule

# Waveform

Timeline

Description automatically generated with medium confidence

# Schematic:

Diagram, schematic

Description automatically generated

Diagram, schematic

Description automatically generated

# I/O Planning:

A screenshot of a computer

Description automatically generated with medium confidence

# floor planning

Graphical user interface

Description automatically generated