**Final Project Report**

Zhili Hu

The Pennsylvania State University

School of Electrical and Computer Science

CMPEN 331 Computer Organization and Design

Dr. Mohamed Almekkawy

December 16, 2022

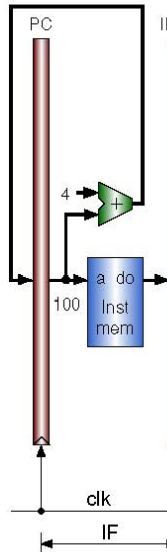
What I am doing:

I am trying to use vivado to complete a pipelining technique for building a fast CPU with five stages. I input 5 instructions for testing. They are ADD, SUB, OR, XOR, AND. The program stores and outputs the data after calculation. See more in introduction.

Introduction

The whole CPU is made by 15 units, separated as 5 stages:

IF: Instruction fetch

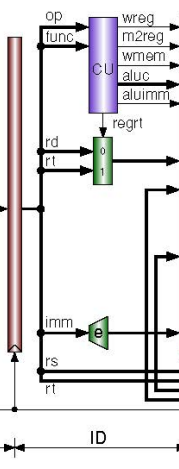
ID: Instruction decode and register file read

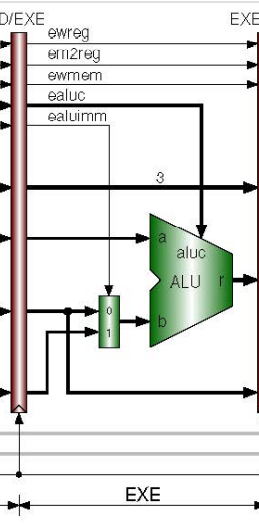
EX: Execution or address calculation

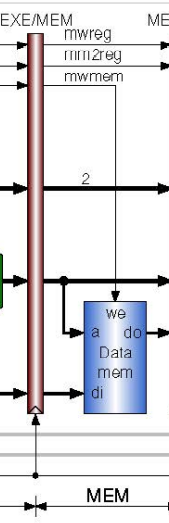
MEM: Data memory access

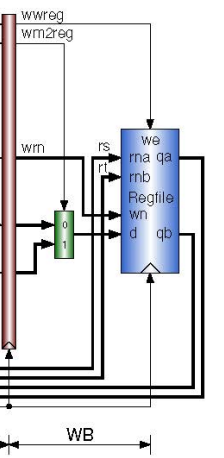
WB: Write back

The first stage is IF, made by Program Counter (PC), Instruction Memory (IM) and an adder. The purpose of this step is to prepare the computer for the next step of reading instructions, and to allocate the addresses required to accept instructions.

Next stage, ID includes 4 units. IF/ID (a pipe to pass information), Control Unit, Regrt Multiplexer and Immediate Extender. The main job of this part is to receive instruction binary code and decode it.

The third stage is EX, including units ID/EXE (pipe), ALU Multiplexer and Arithmetic Logic Unit (ALU). The main job of this part is to receive instruction binary code and decode it. After receiving the instruction, the CPU will start computing.

Then is MEM stage. There are EXE/MEM (pipe) and Data Memory two units. This stage writes to memory. Of course, not all instructions involve memory writes.

The final stage is write-back stage. Three unites are included. MEM/WB (pipe), write back Multiplexer and Register File. The purpose of this phase is to write some necessary data back to the registry for the next cycle.

Device: Zyboboard (XC7Z010- -1CLG400C)

CPUs:

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2022/11/08 11:49:51

// Design Name:

// Module Name: CPUs

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module ProgrameCounter(

input clk,

input [31:0] in\_PC,

output reg [31:0] out\_PC

);

initial begin

out\_PC = 100;

end

always@(posedge clk)

begin

out\_PC = in\_PC;

end

endmodule

module InstructionMemory(

input [31:0] PC,

output reg [31:0] InstOut

);

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

initial begin

imemory[25] <= 32'b00000000001000100001100000100000; //add $3, $1, $2

imemory[26] <= 32'b00000001001000110010000000100010; //sub $4, $9, $3

imemory[27] <= 32'b00000000011010010010100000100101; //or $5, $3, $9

imemory[28] <= 32'b00000000011010010011000000100110; //xor $6, $3, $9

imemory[29] <= 32'b00000000011010010011100000100100; //and $7, $3, $9

end

always @(\*)

begin

InstOut = imemory[PC[7:2]];

end

endmodule

module PcAdder(

input[31:0] pc,

output reg[31:0] nextpc

);

wire[31:0] v = 32'd4;

always @(\*)

begin

nextpc = pc + v;

end

endmodule

module IFID(

input [31:0]instOut,

input clk,

output reg [31:0]dinstOut,

output reg[5:0] op,

output reg[5:0] func,

output reg[4:0] rd,

output reg[4:0] rt,

output reg[4:0] rs,

output reg[15:0] imm

);

always @(posedge clk)

begin

dinstOut = instOut;

op = instOut[31:26];

func = instOut[5:0];

rd = instOut[15:11];

rt = instOut[20:16];

rs = instOut[25:21];

imm = instOut[15:0];

end

endmodule

module ControlUnit(

input [5:0]op,

input [5:0]func,

output reg wreg,

output reg m2reg,

output reg wmem,

output reg [3:0]aluc,

output reg aluimm,

output reg regrt

);

always @(\*)

begin

case(op)

6'b000000: //r-type

begin

if (func == 6'b100000) //add

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0010;

aluimm <= 0;

regrt <= 0;

end

if (func == 6'b100000) //sub

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0101;

aluimm <= 0;

regrt <= 0;

end

if (func == 6'b100000) //or

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0001;

aluimm <= 0;

regrt <= 0;

end

if (func == 6'b100000) //xor

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0011;

aluimm <= 0;

regrt <= 0;

end

if (func == 6'b100000) //and

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0000;

aluimm <= 0;

regrt <= 0;

end

end

6'b100011: // LW

begin

wreg <= 1;

m2reg <= 1;

wmem <= 0;

aluc <= 4'b0010;

aluimm <= 1;

regrt <= 1;

end

default:

begin

wreg = 0;

m2reg = 0;

wmem = 0;

aluc = 4'b0000;

aluimm = 0;

regrt = 1;

end

endcase

end

endmodule

module RegrtMultiplexer(

input [4:0]rt,

input [4:0]rd,

input regrt,

output reg [4:0] destReg

);

always @(\*)

begin

destReg = regrt?rt:rd;

end

endmodule

module RegisterFile(

input [4:0]rs,

input [4:0]rt,

input [4:0]wdestReg,

input [31:0]wbData,

input wwreg,

input clk,

output reg[31:0] qa,

output reg[31:0] qb

);

reg [31:0] rmemory[0:31];

integer i;

initial begin

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

begin

rmemory[i] <= 32'b00000000000000000000000000000000;

end

end

always @(\*)

begin

qa = rmemory[rs];

qb = rmemory[rt];

end

always @(negedge clk)

begin

if (wwreg == 1)

begin

rmemory[wdestReg] = wbData;

end

end

endmodule

module ImmediateExtender(

input [15:0] imm,

output reg [31:0] imm32

);

always @(\*)

begin

imm32[31:0] <= { {16{imm[15]}}, imm [15:0] };

end

endmodule

module IDEXE(

input wreg,

input m2reg,

input wmem,

input [3:0]aluc,

input aluimm,

input [4:0]destReg,

input [31:0]qa,

input [31:0]qb,

input [31:0]imm32,

input clk,

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 clk )

begin

ewreg = wreg;

em2reg = m2reg;

ewmem = wmem;

ealuc = aluc;

ealuimm = aluimm;

edestReg = destReg;

eqa = qa;

eqb = qb;

eimm32 = imm32;

end

endmodule

module AluMux(

input [31:0] eqb,

input [31:0] eimm32,

input ealuimm,

output reg [31:0] b

);

always @(\*)

begin

if (ealuimm == 1'b0)

begin

b = eqb;

end

else

begin

b = eimm32;

end

end

endmodule

module Alu(

input [31:0] eqa,

input [31:0] b,

input [3:0] ealuc,

output reg [31:0] r

);

always @(\*)

begin

case(ealuc)

4'b0010:

begin

r = eqa + b;

end

default:

begin

r = 0;

end

endcase

end

endmodule

module Exemem(

input ewreg,

input em2reg,

input ewmem,

input [4:0] edestReg,

input [31:0] r,

input [31:0] eqb,

input clk,

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 clk)

begin

mwreg = ewreg;

mm2reg = em2reg;

mwmem = ewmem;

mdestReg = edestReg;

mr = r;

mqb = eqb;

end

endmodule

module DataMemory(

input [31:0] mr,

input [31:0] mqb,

input mwmem,

input clk,

output reg [31:0] mdo);

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

initial begin

Data[0]=32'h00000000;

Data[1]=32'hA00000AA;

Data[2]=32'h10000011;

Data[3]=32'h20000022;

Data[4]=32'h30000033;

Data[5]=32'h40000044;

Data[6]=32'h50000055;

Data[7]=32'h60000066;

Data[8]=32'h70000077;

Data[9]=32'h80000088;

Data[10]=32'h90000099;

end

always @(\*)

begin

mdo = Data[mr];

end

always @(negedge clk)

begin

if(mwmem == 1)

begin

Data[mr] = mqb;

end

end

endmodule

module Memwb(

input mwreg,

input mm2reg,

input [4:0] mdestReg,

input [31:0] mr,

input [31:0] mdo,

input clk,

output reg wwreg,

output reg wm2reg,

output reg [4:0] wdestReg,

output reg [31:0] wr,

output reg [31:0] wdo

);

always@(posedge clk)

begin

wwreg = mwreg;

wm2reg = mm2reg;

wdestReg = mdestReg;

wr = mr;

wdo = mdo;

end

endmodule

module WbMux(

input [31:0]wr,

input [31:0]wdo,

input wm2reg,

output reg [31:0]wbData

);

always @(\*)

begin

if(wm2reg == 0)

begin

wbData = wr;

end

else begin

wbData = wdo;

end

end

endmodule

testbench:

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2022/11/08 11:50:26

// Design Name:

// Module Name: testbench

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module testbench();

reg clk;

wire [31:0] in\_pc;

wire [31:0] out\_pc;

wire [31:0] Imout;

wire [31:0] DinstOut;

wire [5:0] op;

wire [5:0] func;

wire [4:0] rd;

wire [4:0] rt;

wire [4:0] rs;

wire [15:0] imm;

wire wreg;

wire m2reg;

wire wmem;

wire [3:0] aluc;

wire aluimm;

wire regrt;

wire [4:0]destReg;

wire [31:0] qa;

wire [31:0] qb;

wire [31:0]imm32;

wire ewreg, em2reg,ewmem, ealuimm;

wire [3:0] ealuc;

wire [4:0] edestReg;

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

wire [31:0] b;

wire [31:0] r;

wire mwreg, mm2reg, mwmem;

wire [4:0] mdestReg;

wire [31:0] mr, mqb;

wire [31:0] mdo;

wire wwreg, wm2reg;

wire [4:0] wdestReg;

wire [31:0] wr,wdo;

wire [31:0]wbData;

initial begin

clk = 0;

end

ProgrameCounter PC (clk, in\_pc, out\_pc);

PcAdder Adder(out\_pc, in\_pc);

InstructionMemory IM (out\_pc, Imout);

IFID Ifid (Imout, clk, DinstOut, op, func, rd, rt, rs, imm);

ControlUnit Control(op, func, wreg, m2reg, wmem, aluc, aluimm, regrt);

RegrtMultiplexer RM (rt, rd, regrt, destReg);

ImmediateExtender IE(imm,imm32);

IDEXE Indexe(wreg, m2reg, wmem, aluc, aluimm, destReg, qa, qb, imm32, clk, ewreg, em2reg, ewmem, ealuc, ealuimm, edestReg, eqa, eqb, eimm32);

AluMux AL(eqb, eimm32, ealuimm, b);

Alu alu\_out(eqa, b, ealuc, r);

Exemem exe(ewreg, em2reg, ewmem, edestReg, r, eqb, clk, mwreg, mm2reg, mwmem, mdestReg, mr, mqb);

DataMemory dm(mr, mqb, mwmem, clk, mdo);

Memwb mem(mwreg, mm2reg, mdestReg, mr, mdo, clk, wwreg, wm2reg, wdestReg, wr, wdo);

WbMux wbmux(wr, wdo, wm2reg, wbData);

RegisterFile RF(rs, rt, wdestReg, wbData, wwreg, clk, qa, qb);

always begin

#5

clk = ~clk;

end

endmodule

