Final Project report

Taihao Liu

12/21/2019

SECTION 2

`timescale 1ns / 1ps

module pipelinedcpu (clk, clrn,pc,inst,ealu,malu,wdi); // pipelined cpu

input clk, clrn; // clock and reset // plus inst mem

output [31:0] pc, inst, ealu, malu, wdi;

// signals in IF stage

wire [31:0] pc\_plus\_4, ins, npc;

// signals in ID stage

wire [31:0] dpc\_plus\_4, bpc, jpc, d\_a, d\_b, d\_imm;

wire [4:0] d\_rn;

wire [3:0] d\_aluc;

wire [1:0] pcsrc;

wire wpcir, dwreg, dm2reg, dwmem, daluimm, dshift, djal;

// signals in EXE stage

wire [31:0] epc\_plus\_4, ea, eb, eimm;

wire [4:0] ern0, ern1;

wire [3:0] e\_aluc;

wire ewreg, em2reg, ewmem, ealuimm, eshift, ejal;

// signals in MEM stage

wire [31:0] m\_b, m\_mo;

wire [4:0] mrn;

wire mwreg, mm2reg, mwmem;

// signals in WB stage

wire [31:0] w\_mo, w\_alu;

wire [4:0] wrn;

wire wwreg, wm2reg;

pipepc prog\_cnt (npc,wpcir,clk,clrn,pc);

pipeif if\_stage (pcsrc,pc,bpc,d\_a,jpc,npc,pc\_plus\_4,ins); // IF stage

// IF/ID pipeline register

pipeif\_id if\_id (pc\_plus\_4,ins,wpcir,clk,clrn,dpc\_plus\_4,inst);

pipeid id\_stage (mwreg,mrn,ern1,ewreg,em2reg,mm2reg,dpc\_plus\_4,inst,wrn,wdi,

ealu,malu,m\_mo,wwreg,clk,clrn,bpc,jpc,pcsrc,wpcir,

dwreg,dm2reg,dwmem,d\_aluc,daluimm,d\_a,d\_b,d\_imm,d\_rn,

dshift,djal); // ID stage

// ID/EXE pipeline register

pipeid\_exe id\_exe (dwreg,dm2reg,dwmem,d\_aluc,daluimm,d\_a,d\_b,d\_imm,d\_rn,

dshift,djal,dpc\_plus\_4,clk,clrn,ewreg,em2reg,ewmem,

e\_aluc,ealuimm,ea,eb,eimm,ern0,eshift,ejal,epc\_plus\_4);

pipeexe exe\_stage (e\_aluc,ealuimm,ea,eb,eimm,eshift,ern0,epc\_plus\_4,ejal,

ern1,ealu); // EXE stage

// EXE/MEM pipeline register

pipeexe\_mem exe\_mem (ewreg,em2reg,ewmem,ealu,eb,ern1,clk,clrn,mwreg,

mm2reg,mwmem,malu,m\_b,mrn);

pipemem mem\_stage (mwmem,malu,m\_b,clk,m\_mo); // MEM stage

// MEM/WB pipeline register

pipemem\_wb mem\_wb (mwreg,mm2reg,m\_mo,malu,mrn,clk,clrn,wwreg,wm2reg,

w\_mo,w\_alu,wrn);

pipewb wb\_stage (w\_alu,w\_mo,wm2reg,wdi); // WB stage

endmodule

module pipepc (npc,wpc,clk,clrn,pc); // program counter

input clk, clrn;

input wpc;

input [31:0] npc;

output [31:0] pc;

program\_counter\_cal prog\_cntr (npc,clk,clrn,wpc,pc); // program counter

endmodule

module program\_counter\_cal (d,clk,clrn,e,q);

input [31:0] d;

input e;

input clk, clrn;

output reg [31:0] q;

always @(negedge clrn or posedge clk)

if (!clrn) q <= 0;

else if (e) q <= d;

endmodule

module pipeif (pcsrc,pc,bpc,rpc,jpc,npc,pc4,ins); // IF stage

input [31:0] pc, bpc, rpc, jpc; // program counter, branch target, jump target of jr and jump target of j/jal

input [1:0] pcsrc; // select next pc

output [31:0] npc, pc4, ins;

mux4x32 next\_pc (pc4,bpc,rpc,jpc,pcsrc,npc);

cla32 pc\_plus4 (pc,32'h4,1'b0,pc4); // pc + 4

pl\_inst\_mem inst\_mem (pc,ins);

endmodule

module mux4x32 (a0,a1,a2,a3,s,y); // 4-to-1 multiplexer to choose the pc we want

input [31:0] a0, a1, a2, a3;//input

input [1:0] s; // selector 2 bits

output [31:0] y; // output

function [31:0] select;

input [31:0] a0,a1,a2,a3;

input [1:0] s;

case (s)

2'b00: select = a0;

2'b01: select = a1;

2'b10: select = a2;

2'b11: select = a3;

endcase

endfunction

assign y = select(a0,a1,a2,a3,s); // call the function

endmodule

module mux2x5(A0,A1,S,Y);// similar as above but for five-bits this time

input [4:0] A0,A1;

input S;

output [4:0] Y;

assign Y = S ? A1 : A0;

endmodule

module mux2x32 (a0,a1,s,y); // similar to above

input [31:0] a0, a1;

input s;

output [31:0] y;

assign y = s ? a1 : a0;

endmodule

module cla32 (a,b,ci,s);

input [31:0] a, b;

input ci;

output [31:0] s;

wire g\_out, p\_out;

cla\_32 cla (a, b, ci, g\_out, p\_out, s);

endmodule

module cla\_32 (a,b,c\_in,g\_out,p\_out,s);

input [31:0] a, b;

input c\_in;

output g\_out, p\_out;

output [31:0] s;

wire [1:0] g, p;

wire c\_out;

cla\_16 a0 (a[15:0], b[15:0], c\_in, g[0],p[0],s[15:0]);

cla\_16 a1 (a[31:16],b[31:16],c\_out,g[1],p[1],s[31:16]);

gp gp0 (g,p,c\_in,g\_out,p\_out,c\_out);

endmodule

module cla\_16 (a,b,c\_in,g\_out,p\_out,s);

input [15:0] a, b;

input c\_in;

output g\_out, p\_out;

output [15:0] s;

wire [1:0] g, p;

wire c\_out;

cla\_8 a0 (a[7:0], b[7:0], c\_in, g[0],p[0],s[7:0]);

cla\_8 a1 (a[15:8],b[15:8],c\_out,g[1],p[1],s[15:8]);

gp gp0 (g,p,c\_in, g\_out,p\_out,c\_out);

endmodule

module cla\_8 (a,b,c\_in,g\_out,p\_out,s);

input [7:0] a, b;

input c\_in;

output g\_out, p\_out;

output [7:0] s;

wire [1:0] g, p;

wire c\_out;

cla\_4 a0 (a[3:0],b[3:0],c\_in, g[0],p[0],s[3:0]);

cla\_4 a1 (a[7:4],b[7:4],c\_out,g[1],p[1],s[7:4]);

gp gp0 (g,p,c\_in, g\_out,p\_out,c\_out);

endmodule

module cla\_4 (a,b,c\_in,g\_out,p\_out,s);

input [3:0] a, b;

input c\_in;

output g\_out, p\_out;

output [3:0] s;

wire [1:0] g, p;

wire c\_out;

cla\_2 a0 (a[1:0],b[1:0],c\_in, g[0],p[0],s[1:0]);

cla\_2 a1 (a[3:2],b[3:2],c\_out,g[1],p[1],s[3:2]);

gp gp0 (g,p,c\_in, g\_out,p\_out,c\_out);

endmodule

module cla\_2 (a, b, c\_in, g\_out, p\_out, s);

input [1:0] a, b;

input c\_in;

output g\_out, p\_out;

output [1:0] s;

wire [1:0] g, p;

wire c\_out;

add a0 (a[0], b[0], c\_in, g[0], p[0], s[0]);

add a1 (a[1], b[1], c\_out, g[1], p[1], s[1]);

gp gp0 (g, p, c\_in, g\_out, p\_out, c\_out);

endmodule

module add (a, b, c, g, p, s);

input a, b, c;

output g, p, s;

assign s = a + b + c;

assign g = a & b;

assign p = a | b;

endmodule

module gp (g,p,c\_in,g\_out,p\_out,c\_out);

input [1:0] g, p;

input c\_in;

output g\_out,p\_out,c\_out;

assign g\_out = g[1] | p[1] & g[0];

assign p\_out = p[1] & p[0];

assign c\_out = g[0] | p[0] & c\_in;

endmodule

//this cla part reference : http://121.40.97.78/read/456362/cla\_32.v\_\_html

module pl\_inst\_mem (a,dins);

input [31:0] a; // pc address

output [31:0] dins;

wire [31:0] rom [0:63];

assign rom[6'h00] = 32'h3c010000; // (00) main: lui $1, 0

assign rom[6'h01] = 32'h34240050; // (04) ori $4, $1, 80

assign rom[6'h02] = 32'h0c00001b; // (08) call: jal sum

assign rom[6'h03] = 32'h20050004; // (0c) dslot1: addi $5, $0, 4

assign rom[6'h04] = 32'hac820000; // (10) return: sw $2, 0($4)

assign rom[6'h05] = 32'h8c890000; // (14) lw $9, 0($4)

assign rom[6'h06] = 32'h01244022; // (18) sub $8, $9, $4

assign rom[6'h07] = 32'h20050003; // (1c) addi $5, $0, 3

assign rom[6'h08] = 32'h20a5ffff; // (20) loop2: addi $5, $5, -1

assign rom[6'h09] = 32'h34a8ffff; // (24) ori $8, $5, 0xffff

assign rom[6'h0a] = 32'h39085555; // (28) xori $8, $8, 0x5555

assign rom[6'h0b] = 32'h2009ffff; // (2c) addi $9, $0, -1

assign rom[6'h0c] = 32'h312affff; // (30) andi $10,$9,0xffff

assign rom[6'h0d] = 32'h01493025; // (34) or $6, $10, $9

assign rom[6'h0e] = 32'h01494026; // (38) xor $8, $10, $9

assign rom[6'h0f] = 32'h01463824; // (3c) and $7, $10, $6

assign rom[6'h10] = 32'h10a00003; // (40) beq $5, $0, shift

assign rom[6'h11] = 32'h00000000; // (44) dslot2: nop

assign rom[6'h12] = 32'h08000008; // (48) j loop2

assign rom[6'h13] = 32'h00000000; // (4c) dslot3: nop

assign rom[6'h14] = 32'h2005ffff; // (50) shift: addi $5, $0, -1

assign rom[6'h15] = 32'h000543c0; // (54) sll $8, $5, 15

assign rom[6'h16] = 32'h00084400; // (58) sll $8, $8, 16

assign rom[6'h17] = 32'h00084403; // (5c) sra $8, $8, 16

assign rom[6'h18] = 32'h000843c2; // (60) srl $8, $8, 15

assign rom[6'h19] = 32'h08000019; // (64) finish: j finish

assign rom[6'h1a] = 32'h00000000; // (68) dslot4: nop

assign rom[6'h1b] = 32'h00004020; // (6c) sum: add $8, $0, $0

assign rom[6'h1c] = 32'h8c890000; // (70) loop: lw $9, 0($4)

assign rom[6'h1d] = 32'h01094020; // (74) stall: add $8, $8, $9

assign rom[6'h1e] = 32'h20a5ffff; // (78) addi $5, $5, -1

assign rom[6'h1f] = 32'h14a0fffc; // (7c) bne $5, $0, loop

assign rom[6'h20] = 32'h20840004; // (80) dslot5: addi $4, $4, 4

assign rom[6'h21] = 32'h03e00008; // (84) jr $31

assign rom[6'h22] = 32'h00081000; // (88) dslot6: sll $2, $8, 0

assign dins = rom[a[7:2]];

endmodule

module pipeif\_id (pc4,ins,wir,clk,clrn,dpc4,dins);

input clk, clrn;

input wir;

input [31:0] pc4, ins;

output [31:0] dpc4, dins;

program\_counter\_cal pc\_plus4 (pc4,clk,clrn,wir,dpc4); // pc+4 register

program\_counter\_cal instruction (ins,clk,clrn,wir,dins); // dins register

endmodule

module pipeid (mwreg,mrn,ern,ewreg,em2reg,mm2reg,dpc4,dins,wrn,wdi,ealu,

malu,mmo,wwreg,clk,clrn,bpc,jpc,pcsrc,nostall,wreg,m2reg,

wmem,aluc,aluimm,a,b,dimm,rn,shift,jal);// ID stage

input clk, clrn;

input [31:0] dpc4, dins,wdi,ealu,malu,mmo;

input [4:0] ern, mrn,wrn;

input ewreg, em2reg, mwreg, mm2reg, wwreg;

output [31:0] bpc, jpc, a, b, dimm;

output [4:0] rn;

output [3:0] aluc;

output [1:0] pcsrc;

output nostall, wreg, m2reg, wmem, aluimm, shift, jal;

wire [5:0] op = dins[31:26];

wire [4:0] rs = dins[25:21];

wire [4:0] rt = dins[20:16];

wire [4:0] rd = dins[15:11];

wire [5:0] func = dins[05:00];

wire [15:0] imm = dins[15:00];

wire [25:0] addr = dins[25:00];//assign the corresponding value into the wire

wire regrt;

wire sext;

wire [31:0] qa, qb;

wire [1:0] fwda, fwdb;

wire [15:0] s16 = {16{sext & dins[15]}};

wire [31:0] dis = {dimm[29:0],2'b00};

wire rsrtequ = ~|(a^b); // reg[rs] == reg[rt]

pipeidcu cu (mwreg,mrn,ern,ewreg,em2reg,mm2reg, // control unit

rsrtequ,func,op,rs,rt,wreg,m2reg,

wmem,aluc,regrt,aluimm,fwda,fwdb,

nostall,sext,pcsrc,shift,jal);

regfile r\_f (rs,rt,wdi,wrn,wwreg,~clk,clrn,qa,qb); // register file

mux2x5 d\_r (rd,rt,regrt,rn); // select dest reg #

mux4x32 s\_a (qa,ealu,malu,mmo,fwda,a); // forward for a

mux4x32 s\_b (qb,ealu,malu,mmo,fwdb,b); // forward for b

cla32 b\_adr (dpc4,dis,1'b0,bpc); // branch target

assign dimm = {s16,imm}; // 32-bit imm

assign jpc = {dpc4[31:28],addr,2'b00}; // jump target

endmodule

module regfile (rna,rnb,d,wn,we,clk,clrn,qa,qb);

input [31:0] d;

input [4:0] rna, rnb, wn;

input we;

input clk, clrn;

output [31:0] qa, qb;

reg [31:0] register [1:31];

// initial begin //initiallize register file

// register[0] = 0;

// register[1] = 0;

// register[2] = 0;

// register[3] = 0;

// register[4] = 0;

// register[5] = 0;

// register[6] = 0;

// register[7] = 0;

// register[8] = 0;

// register[9] = 0;

// register[10] = 0;

// register[11] = 0;

// register[12] = 0;

// register[13] = 0;

// register[14] = 0;

// register[15] = 0;

// end

assign qa = (rna == 0)? 0 : register[rna];

assign qb = (rnb == 0)? 0 : register[rnb];

always @(posedge clk or negedge clrn) // write port

if (!clrn)

register[1] <= 0;

else

if ((wn != 0) && we) // not reg[0] & enabled

register[wn] <= d; // write d to reg[wn]

endmodule

module pipeidcu (mwreg,mrn,ern,ewreg,em2reg,mm2reg,rsrtequ,func,op,rs,rt,

wreg,m2reg,wmem,aluc,regrt,aluimm,fwda,fwdb,nostall,sext,

pcsrc,shift,jal); // control unit in ID stage

input [5:0] op,func;

input [4:0] rs,rt, ern, mrn;

input ewreg, em2reg, mwreg, mm2reg, rsrtequ;

output [3:0] aluc;

output [1:0] pcsrc, fwda, fwdb;

output wreg, m2reg, wmem, aluimm, shift, jal, regrt, sext, nostall;

wire rtype,i\_add,i\_sub,i\_and,i\_or,i\_xor,i\_sll,i\_srl,i\_sra,i\_jr;

wire i\_addi,i\_andi,i\_ori,i\_xori,i\_lw,i\_sw,i\_beq,i\_bne,i\_lui,i\_j,i\_jal;

and (rtype,~op[5],~op[4],~op[3],~op[2],~op[1],~op[0]); // r format

and (i\_add,rtype, func[5],~func[4],~func[3],~func[2],~func[1],~func[0]);

and (i\_sub,rtype, func[5],~func[4],~func[3],~func[2], func[1],~func[0]);

and (i\_and,rtype, func[5],~func[4],~func[3], func[2],~func[1],~func[0]);

and (i\_or, rtype, func[5],~func[4],~func[3], func[2],~func[1], func[0]);

and (i\_xor,rtype, func[5],~func[4],~func[3], func[2], func[1],~func[0]);

and (i\_sll,rtype,~func[5],~func[4],~func[3],~func[2],~func[1],~func[0]);

and (i\_srl,rtype,~func[5],~func[4],~func[3],~func[2], func[1],~func[0]);

and (i\_sra,rtype,~func[5],~func[4],~func[3],~func[2], func[1], func[0]);

and (i\_jr, rtype,~func[5],~func[4], func[3],~func[2],~func[1],~func[0]);

and (i\_addi,~op[5],~op[4], op[3],~op[2],~op[1],~op[0]); // i format

and (i\_andi,~op[5],~op[4], op[3], op[2],~op[1],~op[0]);

and (i\_ori, ~op[5],~op[4], op[3], op[2],~op[1], op[0]);

and (i\_xori,~op[5],~op[4], op[3], op[2], op[1],~op[0]);

and (i\_lw, op[5],~op[4],~op[3],~op[2], op[1], op[0]);

and (i\_sw, op[5],~op[4], op[3],~op[2], op[1], op[0]);

and (i\_beq, ~op[5],~op[4],~op[3], op[2],~op[1],~op[0]);

and (i\_bne, ~op[5],~op[4],~op[3], op[2],~op[1], op[0]);

and (i\_lui, ~op[5],~op[4], op[3], op[2], op[1], op[0]);

and (i\_j, ~op[5],~op[4],~op[3],~op[2], op[1],~op[0]); // j format

and (i\_jal, ~op[5],~op[4],~op[3],~op[2], op[1], op[0]);

wire i\_rs = i\_add | i\_sub | i\_and | i\_or | i\_xor | i\_jr | i\_addi |

i\_andi | i\_ori | i\_xori | i\_lw | i\_sw | i\_beq | i\_bne;

wire i\_rt = i\_add | i\_sub | i\_and | i\_or | i\_xor | i\_sll | i\_srl |

i\_sra | i\_sw | i\_beq | i\_bne;

// pipeline stall caused by data dependency with lw instruction

assign nostall = ~(ewreg & em2reg & (ern != 0) & (i\_rs & (ern == rs) |

i\_rt & (ern == rt)));

reg [1:0] fwda, fwdb;//forwarding

always @ (ewreg, mwreg, ern, mrn, em2reg, mm2reg, rs, rt) begin

fwda = 2'b00; // default: no hazards

if (ewreg & (ern != 0) & (ern == rs) & ~em2reg) begin

fwda = 2'b01; // select exe\_alu

end else begin

if (mwreg & (mrn != 0) & (mrn == rs) & ~mm2reg) begin

fwda = 2'b10; // select mem\_alu

end else begin

if (mwreg & (mrn != 0) & (mrn == rs) & mm2reg) begin

fwda = 2'b11; // select mem\_lw

end

end

end

// forward control signal for alu input b

fwdb = 2'b00; // default: no hazards

if (ewreg & (ern != 0) & (ern == rt) & ~em2reg) begin

fwdb = 2'b01; // select exe\_alu

end else begin

if (mwreg & (mrn != 0) & (mrn == rt) & ~mm2reg) begin

fwdb = 2'b10; // select mem\_alu

end else begin

if (mwreg & (mrn != 0) & (mrn == rt) & mm2reg) begin

fwdb = 2'b11; // select mem\_lw

end

end

end

end

// control signals

assign wreg =(i\_add |i\_sub |i\_and |i\_or |i\_xor |i\_sll |i\_srl |

i\_sra |i\_addi|i\_andi|i\_ori |i\_xori|i\_lw |i\_lui |

i\_jal)& nostall; // prevent from executing twice

assign regrt = i\_addi|i\_andi|i\_ori |i\_xori|i\_lw |i\_lui;

assign jal = i\_jal;

assign m2reg = i\_lw;

assign shift = i\_sll |i\_srl |i\_sra;

assign aluimm = i\_addi|i\_andi|i\_ori |i\_xori|i\_lw |i\_lui |i\_sw;

assign sext = i\_addi|i\_lw |i\_sw |i\_beq |i\_bne;

assign aluc[3] = i\_sra;

assign aluc[2] = i\_sub |i\_or |i\_srl |i\_sra |i\_ori |i\_lui;

assign aluc[1] = i\_xor |i\_sll |i\_srl |i\_sra |i\_xori|i\_beq |i\_bne|i\_lui;

assign aluc[0] = i\_and |i\_or |i\_sll |i\_srl |i\_sra |i\_andi|i\_ori;

assign wmem = i\_sw & nostall; // prevent from executing twice

assign pcsrc[1] = i\_jr |i\_j |i\_jal;

assign pcsrc[0] = i\_beq & rsrtequ | i\_bne & ~rsrtequ | i\_j | i\_jal;

endmodule

//Above module reference: http://121.40.97.78/article/456362

module pipeid\_exe (dwreg,dm2reg,dwmem,daluc,daluimm,da,db,dimm,drn,dshift,

djal,dpc4,clk,clrn,ewreg,em2reg,ewmem,ealuc,ealuimm,ea,

eb,eimm,ern,eshift,ejal,epc4); // ID/EXE pipeline register

input clk, clrn; // clock and reset

input [31:0] da, db, dimm, dpc4;

input [4:0] drn;

input [3:0] daluc;

input dwreg,dm2reg,dwmem,daluimm,dshift,djal;

output [31:0] ea, eb, eimm, epc4;

output [4:0] ern;

output [3:0] ealuc;

output ewreg,em2reg,ewmem,ealuimm,eshift,ejal;

reg [31:0] ea, eb, eimm, epc4;

reg [4:0] ern;

reg [3:0] ealuc;

reg ewreg,em2reg,ewmem,ealuimm,eshift,ejal;

always @(posedge clk) begin

ewreg <= dwreg;

em2reg <= dm2reg;

ewmem <= dwmem;

ealuc <= daluc;

ealuimm <= daluimm;

ea <= da;

eb <= db;

eimm <= dimm;

ern <= drn;

eshift <= dshift;

ejal <= djal;

epc4 <= dpc4;

end

endmodule

module pipeexe (ealuc,ealuimm,ea,eb,eimm,eshift,ern0,epc4,ejal,ern,ealu);

input [31:0] ea, eb, eimm, epc4;

input [4:0] ern0;

input [3:0] ealuc;

input ealuimm, eshift, ejal;

output [31:0] ealu;

output [4:0] ern;

wire [31:0] alua, alub, ealu0, epc8;

wire [31:0] esa = {eimm[5:0],eimm[31:6]};

cla32 ret\_addr (epc4,32'h4,1'b0,epc8);

mux2x32 alu\_in\_a (ea,esa,eshift,alua);

mux2x32 alu\_in\_b (eb,eimm,ealuimm,alub);

mux2x32 save\_pc8 (ealu0,epc8,ejal,ealu);

assign ern = ern0 | {5{ejal}};

alu al\_unit (alua,alub,ealuc,ealu0);

endmodule

module alu (a,b,aluc,r);

input [31:0] a, b;

input [3:0] aluc;

output [31:0] r;

wire [31:0] d\_and = a & b; //AND

wire [31:0] d\_xor = a ^ b; //XOR

wire [31:0] d\_or = a | b; //OR

wire [31:0] d\_xor\_lui = aluc[2]? d\_lui : d\_xor; //SRL

wire [31:0] d\_as, d\_sh; //SRA

wire [31:0] d\_and\_or = aluc[2]? d\_or : d\_and; //SLL

wire [31:0] d\_lui = {b[15:0],16'h0};

addsub32 as32 (a,b,aluc[2],d\_as);

shift shifter (b,a[4:0],aluc[2],aluc[3],d\_sh);

mux4x32 res (d\_as,d\_and\_or,d\_xor\_lui,d\_sh,aluc[1:0],r);

endmodule

// alu reference: http://121.40.97.78/read/456362/alu.v\_\_html

module addsub32 (a,b,sub,s);

input [31:0] a, b;

input sub;

output [31:0] s;

wire [31:0] b\_xor\_sub = b ^ {32{sub}};

cla32 as32 (a, b\_xor\_sub, sub, s);

endmodule

module shift (d,sa,right,arith,sh);

input [31:0] d;

input [4:0] sa;

input right, arith;

output [31:0] sh;

reg [31:0] sh;

always @\* begin

if (!right) begin

sh = d << sa;

end else if (!arith) begin

sh = d >> sa;

end else begin

sh = $signed(d) >>> sa;

end

end

endmodule

module pipeexe\_mem (ewreg,em2reg,ewmem,ealu,eb,ern,clk,clrn,mwreg,mm2reg,

mwmem,malu,mb,mrn); // EXE/MEM pipeline register

input clk, clrn;

input [31:0] ealu, eb;

input [4:0] ern;

input ewreg,em2reg,ewmem;

output [31:0] malu, mb;

output [4:0] mrn;

output mwreg,mm2reg,mwmem;

reg [31:0] malu,mb;

reg [4:0] mrn;

reg mwreg,mm2reg,mwmem;

always @(posedge clk) begin

mwreg <= ewreg;

mm2reg <= em2reg;

mwmem <= ewmem;

malu <= ealu;

mb <= eb;

mrn <= ern;

end

endmodule

module pipemem (we,addr,datain,clk,dataout); // MEM stage

input clk; // clock

input [31:0] addr, datain;

input we; // memory write

output [31:0] dataout; // data out (from mem)

pl\_data\_mem dmem (clk,dataout,datain,addr,we);

endmodule

module pl\_data\_mem (clk,dataout,datain,addr,we);

input clk; // clock

input [31:0] addr; // ram address

input [31:0] datain; // data in (to memory)

input we; // write enable

output [31:0] dataout; // data out (from memory)

reg [31:0] ram [0:31]; // ram cells: 32 words \* 32 bits

assign dataout = ram[addr[6:2]]; // use 5-bit word address

always @ (posedge clk) begin

if (we) ram[addr[6:2]] = datain; // write ram

end

integer i;

initial begin // ram initialization

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

ram[i] = 0;

ram[5'h14] = 32'h000000a3; // (50) data[0] 0 + a3 = a3

ram[5'h15] = 32'h00000027; // (54) data[1] a3 + 27 = ca

ram[5'h16] = 32'h00000079; // (58) data[2] ca + 79 = 143

ram[5'h17] = 32'h00000115; // (5c) data[3] 143 + 115 = 258

end

endmodule

module pipemem\_wb (mwreg,mm2reg,mmo,malu,mrn,clk,clrn,wwreg,wm2reg,wmo,walu,

wrn); // MEM/WB pipeline register

input clk, clrn; // clock and reset

input [31:0] mmo, malu;

input [4:0] mrn;

input mwreg, mm2reg;

output [31:0] wmo, walu;

output [4:0] wrn;

output wwreg, wm2reg;

reg [31:0] wmo, walu;

reg [4:0] wrn;

reg wwreg,wm2reg;

always @(posedge clk) begin

wwreg <= mwreg;

wm2reg <= mm2reg;

wmo <= mmo;

walu <= malu;

wrn <= mrn;

end

endmodule

module pipewb (walu,wmo,wm2reg,wdi); // WB stage

input [31:0] walu, wmo;

input wm2reg;

output [31:0] wdi;

mux2x32 select\_wb (walu,wmo,wm2reg,wdi);

endmodule

Test\_bench:

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 12/19/2019 04:43:21 PM

// Design Name:

// Module Name: final\_tb

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module final\_tb();

reg clk, clrn;

wire [31:0] pc; // program counter // and data mem

wire [31:0] inst; // instruction in ID stage

wire [31:0] ealu; // alu result in EXE stage

wire [31:0] malu; // alu result in MEM stage

wire [31:0] wdi; // data to be written into register file

// signals in IF stage

wire [31:0] pc\_plus\_4, ins, npc;

// signals in ID stage

wire [31:0] dpc\_plus\_4, bpc, jpc, d\_a, d\_b, d\_imm;

wire [4:0] d\_rn;

wire [3:0] d\_aluc;

wire [1:0] pcsrc;

wire wpcir, dwreg, dm2reg, dwmem, daluimm, dshift, djal;

// signals in EXE stage

wire [31:0] epc\_plus\_4, ea, eb, eimm;

wire [4:0] ern0, ern1;

wire [3:0] e\_aluc;

wire ewreg, em2reg, ewmem, ealuimm, eshift, ejal;

// signals in MEM stage

wire [31:0] m\_b, m\_mo;

wire [4:0] mrn;

wire mwreg, mm2reg, mwmem;

// signals in WB stage

wire [31:0] w\_mo, w\_alu;

wire [4:0] wrn;

wire wwreg, wm2reg;

pipelinedcpu tb(clk,clrn,pc,inst,ealu,malu,wdi);

pipepc prog\_cnt (npc,wpcir,clk,clrn,pc);

pipeif if\_stage (pcsrc,pc,bpc,d\_a,jpc,npc,pc\_plus\_4,ins); // IF stage

// IF/ID pipeline register

pipeif\_id if\_id (pc\_plus\_4,ins,wpcir,clk,clrn,dpc\_plus\_4,inst);

pipeid id\_stage (mwreg,mrn,ern1,ewreg,em2reg,mm2reg,dpc\_plus\_4,inst,wrn,wdi,

ealu,malu,m\_mo,wwreg,clk,clrn,bpc,jpc,pcsrc,wpcir,

dwreg,dm2reg,dwmem,d\_aluc,daluimm,d\_a,d\_b,d\_imm,d\_rn,

dshift,djal); // ID stage

// ID/EXE pipeline register

pipeid\_exe id\_exe (dwreg,dm2reg,dwmem,d\_aluc,daluimm,d\_a,d\_b,d\_imm,d\_rn,

dshift,djal,dpc\_plus\_4,clk,clrn,ewreg,em2reg,ewmem,

e\_aluc,ealuimm,ea,eb,eimm,ern0,eshift,ejal,epc\_plus\_4);

pipeexe exe\_stage (e\_aluc,ealuimm,ea,eb,eimm,eshift,ern0,epc\_plus\_4,ejal,

ern1,ealu); // EXE stage

// EXE/MEM pipeline register

pipeexe\_mem exe\_mem (ewreg,em2reg,ewmem,ealu,eb,ern1,clk,clrn,mwreg,

mm2reg,mwmem,malu,m\_b,mrn);

pipemem mem\_stage (mwmem,malu,m\_b,clk,m\_mo); // MEM stage

// MEM/WB pipeline register

pipemem\_wb mem\_wb (mwreg,mm2reg,m\_mo,malu,mrn,clk,clrn,wwreg,wm2reg,

w\_mo,w\_alu,wrn);

pipewb wb\_stage (w\_alu,w\_mo,wm2reg,wdi); // WB stage

initial

begin

clk = 0;

clrn = 0;

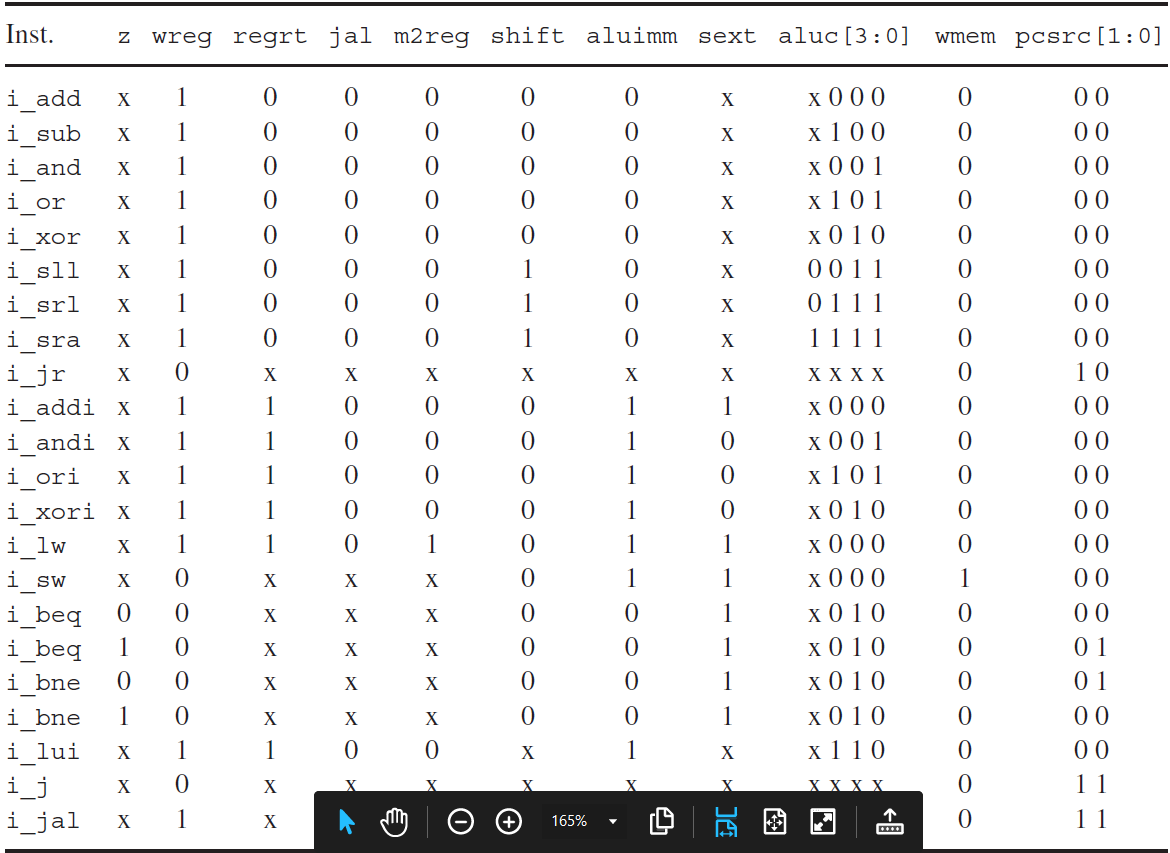
#5 clrn = !clrn;

end

always

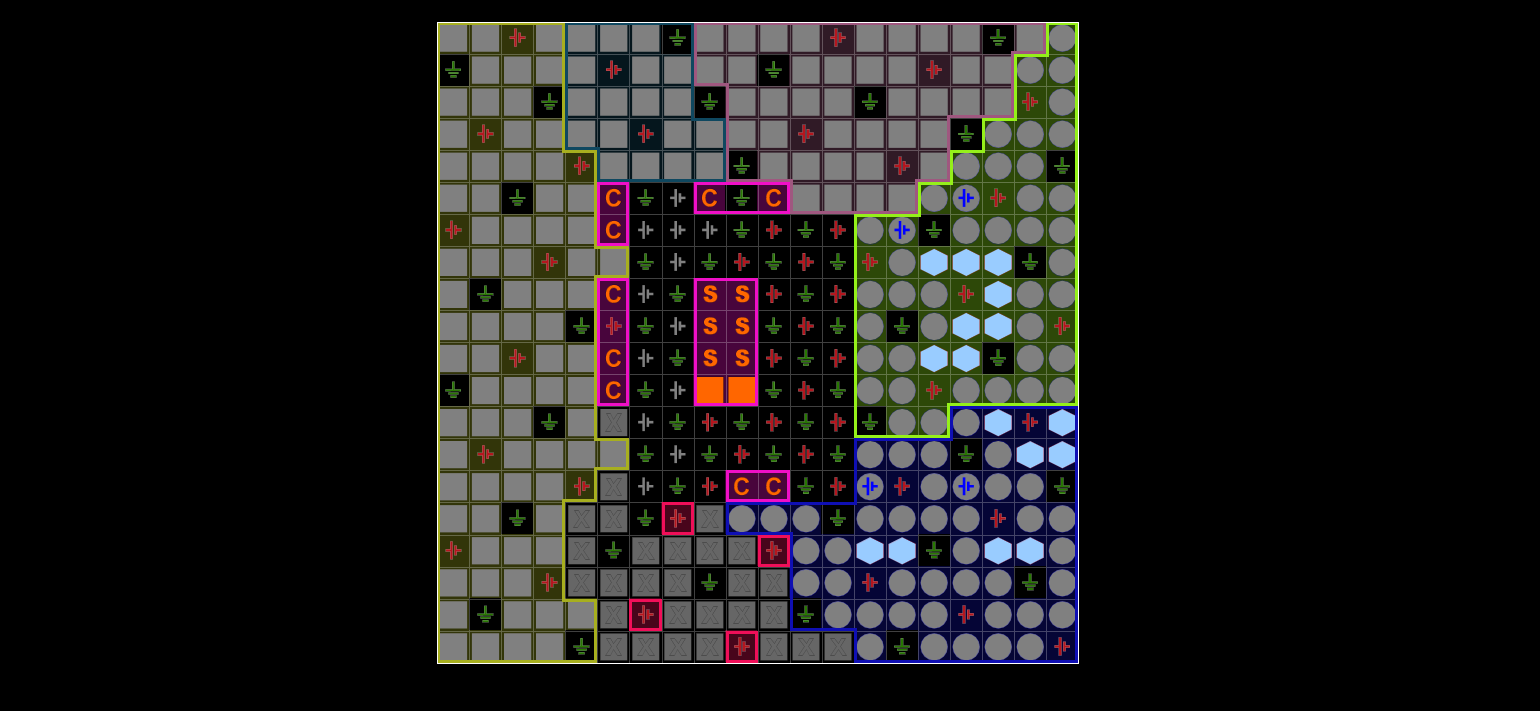
#5 clk = !clk;

endmodule

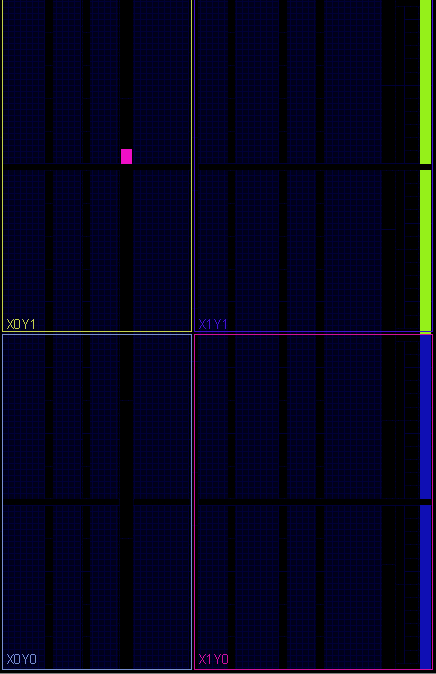


Using it as reference to implement the module

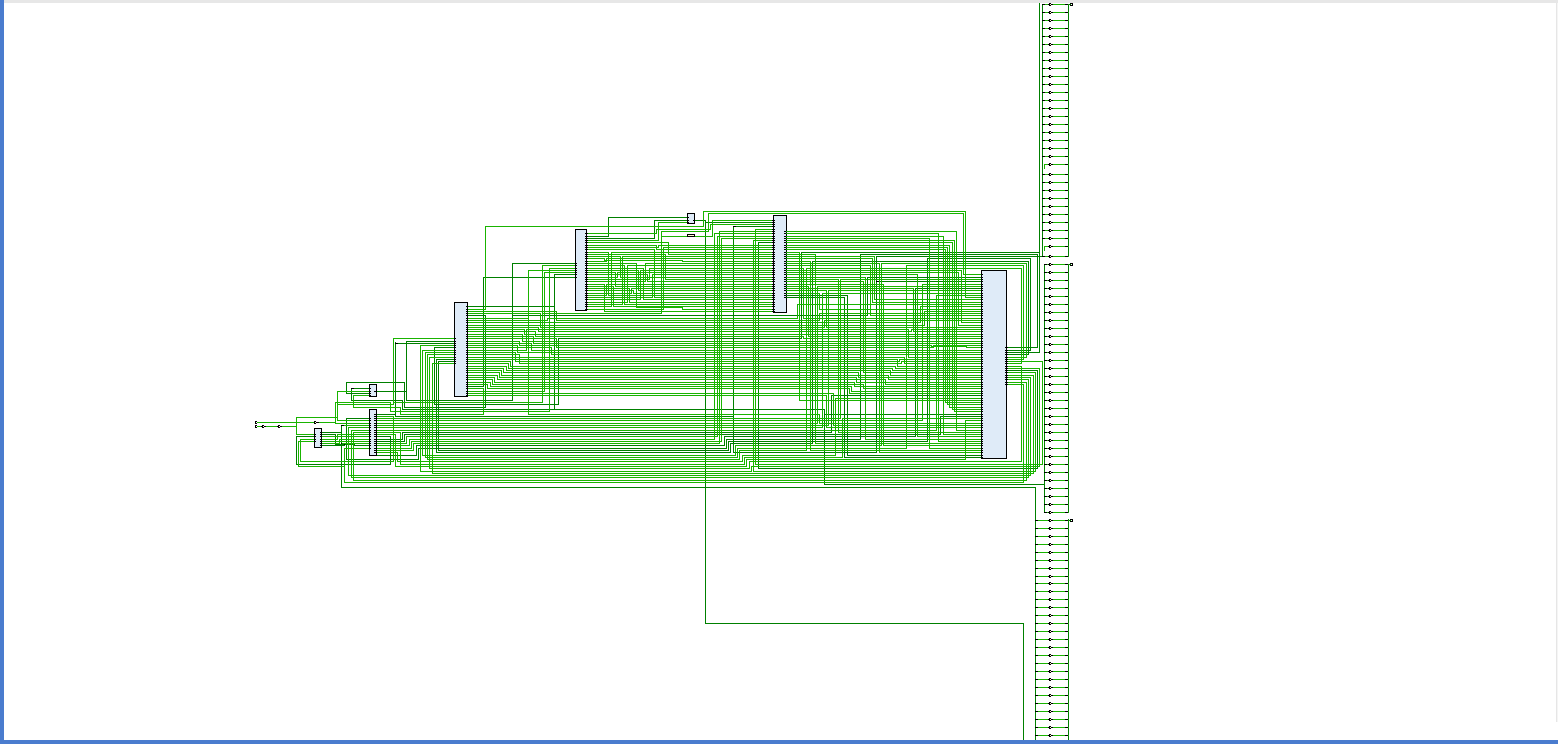
Device: XC7Z010-1CLG400C

I/O Planning: 

Floor Planning:



Schematic:



Waveform:

