实验3 简单计算机系统-系统设计A-代码

电 25 吴晨聪 2022010311

# 1. mux2.v

module mux2(d0,d1,sel,y);

input [7:0]d0;

input [7:0]d1;

input sel;

output reg [7:0]y;

always@(\*) begin

if(!sel) begin

y=d0;

end

else begin

y=d1;

end

end

endmodule

# 2. mux2\_tb.v

`timescale 1ns/1ps

module mux2\_tb;

reg [7:0]d0;

reg [7:0]d1;

reg sel;

wire [7:0]y;

initial begin

d0 = 1;

d1 = 2;

sel = 0;

#100 sel = 1;

#100;

end

mux2 mux2(

.d0(d0),

.d1(d1),

.sel(sel),

.y(y)

);

endmodule

# 3. mux1.v

module mux1(d0,d1,sel,y);

input [1:0]d0;

input [1:0]d1;

input sel;

output reg [1:0]y;

always@(\*) begin

if(!sel) begin

y=d0;

end

else begin

y=d1;

end

end

endmodule

# 4. cpuB.v

module cpuB(Rs,Rt,Rd,selscrB,regdes,memtoreg,memData,imm,alucs,clk,rst\_n,regwrite,flagwrite,s,zeroout);

wire [7:0]scrA;

wire [7:0]scrB;

input [1:0]Rs;

input [1:0]Rt;

input [1:0]Rd;

input [7:0]memData;

input selscrB;

input regdes;

input memtoreg;

input [7:0]imm;

input regwrite;

input [2:0]alucs;

input clk;

input rst\_n;

input flagwrite;

wire [1:0]nd;

wire [7:0]di;

wire [7:0]q2;

wire carry\_out;

wire carry\_in;

wire zeroin;

output [7:0]s;

output zeroout;

always@(posedge clk or negedge rst\_n) begin

if(!rst\_n) begin

end

else begin

end

end

mux2 muxscrB(

.d0(q2),

.d1(imm),

.sel(selscrB),

.y(scrB)

);

mux1 muxnd(

.d0(Rt),

.d1(Rd),

.sel(regdes),

.y(nd)

);

mux2 muxdi(

.d0(s),

.d1(memData),

.sel(memtoreg),

.y(di)

);

regfile regfile(

.clk(clk),

.rst\_n(rst\_n),

.n1(Rs),

.n2(Rt),

.nd(nd),

.di(di),

.reg\_we(regwrite),

.q1(scrA),

.q2(q2)

);

alu alu(

.data\_a(scrA),

.data\_b(scrB),

.s(s),

.zero(zeroin),

.cs(alucs),

.carry\_in(carry\_in),

.carry\_out(carry\_out)

);

flag flag(

.clk(clk),

.rst\_n(rst\_n),

.zeroin(zeroin),

.flagwrite(flagwrite),

.flagin(carry\_out),

.flagout(carry\_in),

.zeroout(zeroout)

);

endmodule

# 5. cpuB\_tb.v

`timescale 1ns/1ps

module cpuB\_tb;

reg [1:0]Rs;

reg [1:0]Rt;

reg [1:0]Rd;

reg [7:0]memData;

reg selscrB;

reg regdes;

reg memtoreg;

reg [7:0]imm;

reg regwrite;

reg [2:0]alucs;

reg clk;

reg rst\_n;

reg flagwrite;

wire [7:0]s;

wire zeroout;

parameter AND =3'b000,

OR =3'b001,

ADD =3'b010,

SUB =3'b011,

SLT =3'b100,

SUBC=3'b101,

ADDC=3'b110;

initial begin

clk=1;

rst\_n=1;

Rs=0;

Rt=0;

Rd=0;

memData=0;

selscrB=0;

regdes=0;

memtoreg=0;

imm=0;

regwrite=0;

alucs=0;

#20 rst\_n=0;

#20 rst\_n=1;

#20 alucs=0;

flagwrite=0;

Rs=0;

Rt=1;

Rd=2;

memtoreg=0;

regdes=1;

selscrB=0;

regwrite=1;

#20 alucs=1;

flagwrite=0;

Rs=0;

Rt=1;

Rd=3;

memtoreg=0;

regdes=1;

selscrB=0;

regwrite=1;

#20 alucs=2;

flagwrite=1;

Rs=2;

Rt=3;

Rd=2;

memtoreg=0;

regdes=1;

selscrB=0;

regwrite=1;

#20 alucs=6;

flagwrite=1;

Rs=2;

Rt=3;

Rd=2;

memtoreg=0;

regdes=0;

selscrB=1;

imm=255;

regwrite=1;

#20 alucs=6;

flagwrite=1;

Rs=2;

Rt=3;

Rd=2;

memtoreg=1;

memData=112;

regdes=0;

selscrB=0;

regwrite=1;

#20 alucs=3;

flagwrite=1;

Rs=3;

Rt=0;

Rd=3;

memtoreg=0;

regdes=1;

selscrB=0;

regwrite=1;

#20;

end

always #10 clk = ~clk;

cpuB cpuB(

.Rs(Rs),

.Rt(Rt),

.Rd(Rd),

.selscrB(selscrB),

.regdes(regdes),

.memtoreg(memtoreg),

.memData(memData),

.imm(imm),

.alucs(alucs),

.clk(clk),

.rst\_n(rst\_n),

.regwrite(regwrite),

.flagwrite(flagwrite),

.s(s),

.zeroout(zeroout)

);

endmodule

# 6. controller.v

module controller(Op,alucs,flagwrite,regwrite,selscrB,redges,memtoreg);

input [3:0]Op;

output reg [2:0]alucs;

output reg flagwrite;

output reg regwrite;

output reg selscrB;

output reg redges;

output reg memtoreg;

always@(\*) begin

if(Op>=0 && Op<=6) begin

regwrite=1;

selscrB=0;

redges=1;

memtoreg=0;

alucs=Op[2:0];

if(Op==2 || Op==3 || Op==5 || Op==6) begin

flagwrite=1;

end

else begin

flagwrite=0;

end

end

end

endmodule

# 7. cpuC.v

module cpuC(memData,clk,rst\_n,s,zeroout);

wire [7:0]scrA;

wire [7:0]scrB;

wire [1:0]Rs;

wire [1:0]Rt;

wire [1:0]Rd;

wire [3:0]Op;

input [7:0]memData;

wire selscrB;

wire regdes;

wire memtoreg;

wire [7:0]imm;

wire regwrite;

wire [2:0]alucs;

input clk;

input rst\_n;

wire flagwrite;

wire [1:0]nd;

wire [7:0]di;

wire [7:0]q2;

wire carry\_out;

wire carry\_in;

wire zeroin;

output [7:0]s;

output zeroout;

always@(posedge clk or negedge rst\_n) begin

if(!rst\_n) begin

end

else begin

end

end

mux2 muxscrB(

.d0(q2),

.d1(imm),

.sel(selscrB),

.y(scrB)

);

mux1 muxnd(

.d0(Rt),

.d1(Rd),

.sel(regdes),

.y(nd)

);

mux2 muxdi(

.d0(s),

.d1(memData),

.sel(memtoreg),

.y(di)

);

regfile regfile(

.clk(clk),

.rst\_n(rst\_n),

.n1(Rs),

.n2(Rt),

.nd(nd),

.di(di),

.reg\_we(regwrite),

.q1(scrA),

.q2(q2)

);

alu alu(

.data\_a(scrA),

.data\_b(scrB),

.s(s),

.zero(zeroin),

.cs(alucs),

.carry\_in(carry\_in),

.carry\_out(carry\_out)

);

flag flag(

.clk(clk),

.rst\_n(rst\_n),

.zeroin(zeroin),

.flagwrite(flagwrite),

.flagin(carry\_out),

.flagout(carry\_in),

.zeroout(zeroout)

);

ROM ROM(

.clk(clk),

.rst\_n(rst\_n),

.Op(Op),

.Rs(Rs),

.Rt(Rt),

.Rd(Rd),

.imm(imm)

);

controller controller(

.Op(Op),

.alucs(alucs),

.flagwrite(flagwrite),

.regwrite(regwrite),

.selscrB(selscrB),

.redges(redges),

.memtoreg(memtoreg)

);

endmodule

# 8. cpuC\_tb.v

`timescale 1ns/1ps

module cpuC\_tb;

reg clk;

reg rst\_n;

reg [7:0]memData;

wire [7:0]s;

wire zeroout;

initial begin

memData=0;

clk=1;

rst\_n=0;

#20 rst\_n=1;

end

always #10 clk = ~clk;

cpuC cpuC(

.memData(memData),

.clk(clk),

.rst\_n(rst\_n),

.s(s),

.zeroout(zeroout)

);

endmodule 9. cpuA.v

module cpuA(scrA,scrB,alucs,clk,rst\_n,flagwrite,s,zeroout);

input [7:0]scrA;

input [7:0]scrB;

input [2:0]alucs;

input clk;

input rst\_n;

input flagwrite;

wire carry\_out;

wire carry\_in;

wire zeroin;

output [7:0]s;

output zeroout;

always@(posedge clk or negedge rst\_n) begin

if(!rst\_n) begin

end

else begin

end

end

alu alu(

.data\_a(scrA),

.data\_b(scrB),

.s(s),

.zero(zeroin),

.cs(alucs),

.carry\_in(carry\_in),

.carry\_out(carry\_out)

);

flag flag(

.clk(clk),

.rst\_n(rst\_n),

.zeroin(zeroin),

.flagwrite(flagwrite),

.flagin(carry\_out),

.flagout(carry\_in),

.zeroout(zeroout)

);

endmodule