实验六 CPU综合设计

**一、实验目的**

1. 综合运用Verilog进行复杂系统设计。
2. 深刻理解计算机系统硬件原理。

**二、实验内容**

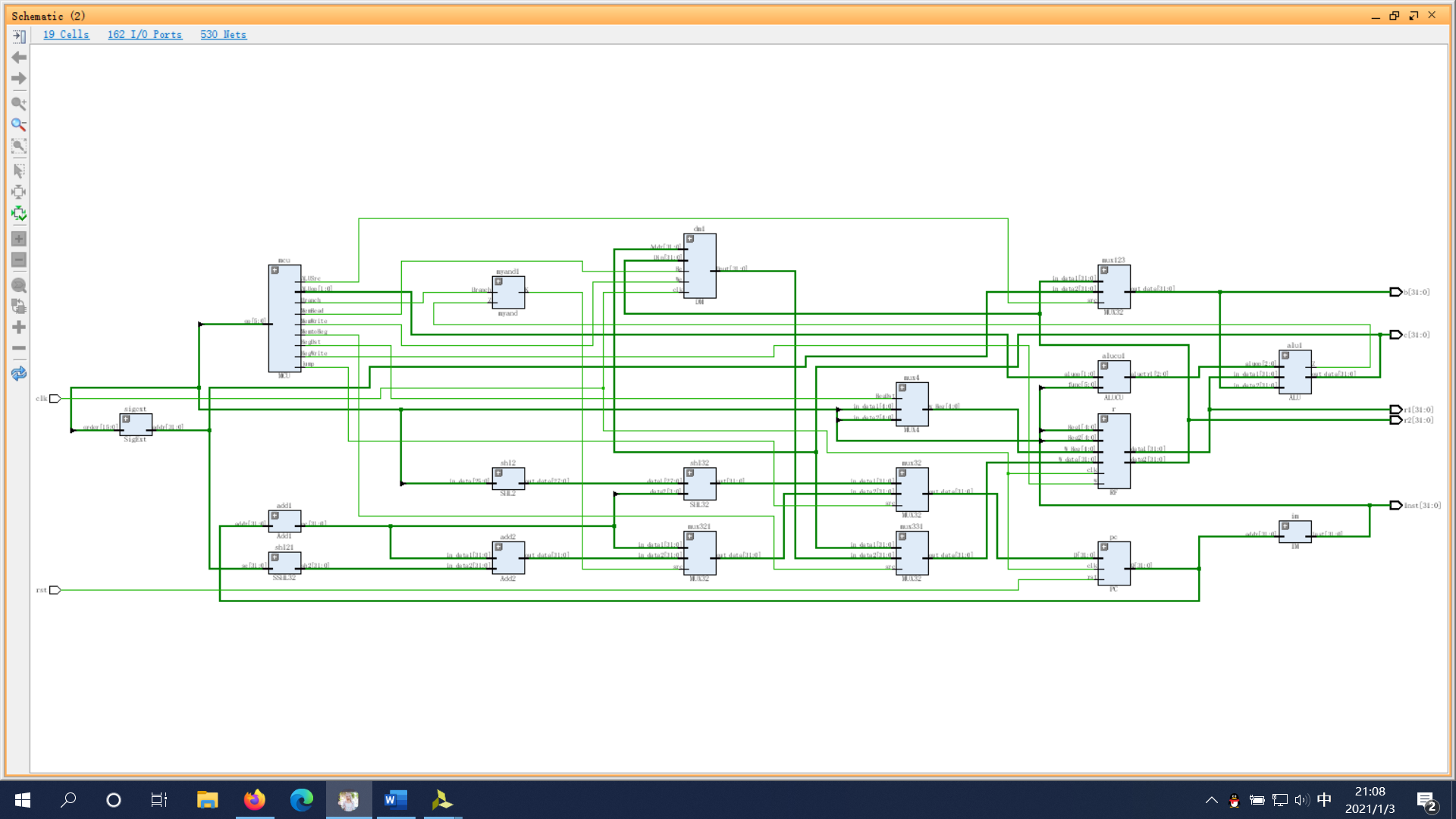
1. 设计一个基于MIPS/ARM/RISC-V指令集的CPU。
2. CPU需要包含寄存器组、RAM模块、ALU模块、指令译码模块。
3. 该CPU能运行基本的汇编指令。
4. 实现多核，流水线或其他现代CPU的高级功能（加分项）

**三、实验要求**

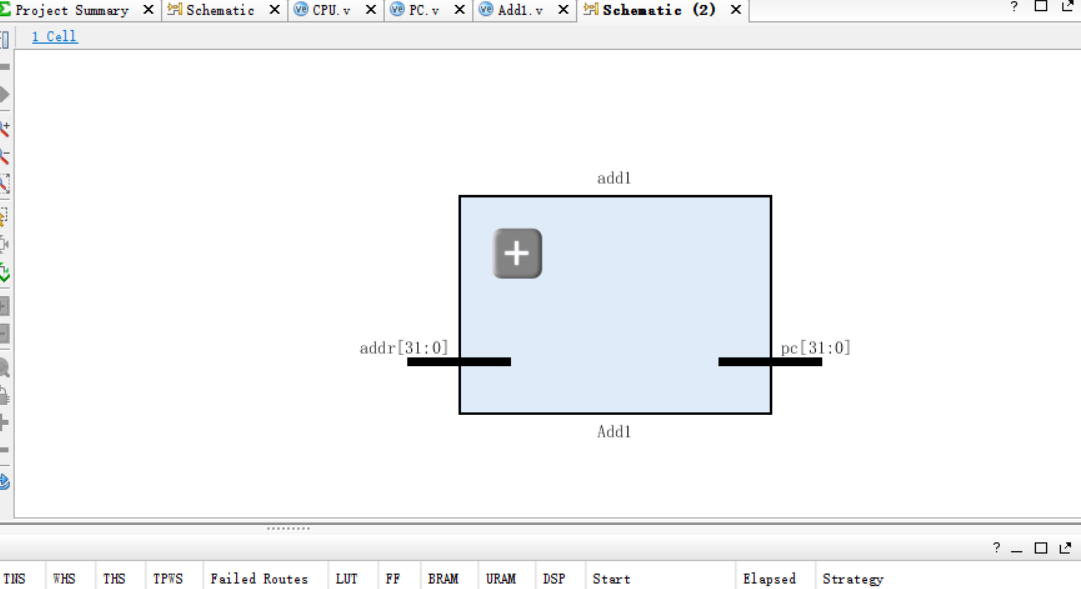
1. 分析各模块的的程序结构，画出其流程图。
2. 画出模块的电路图。
3. 分析电路的仿真波形，标出关键的数值。
4. 记录设计和调试过程。

**四、实验代码及结果、**

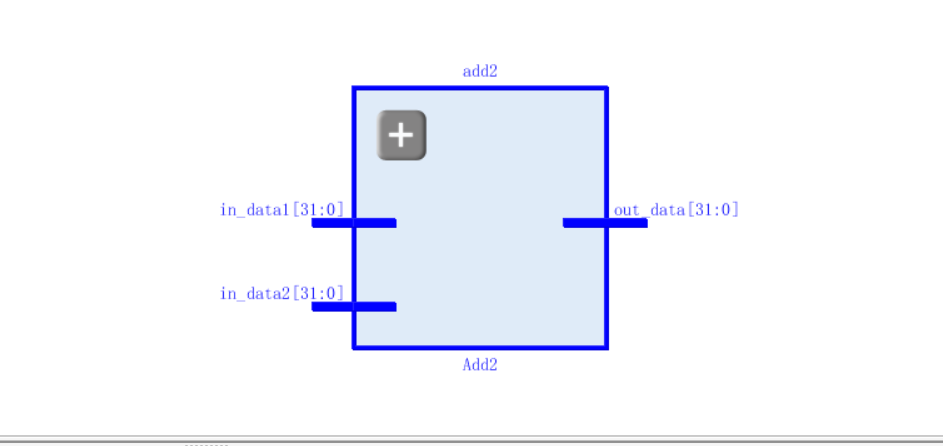
全局CPU电路图



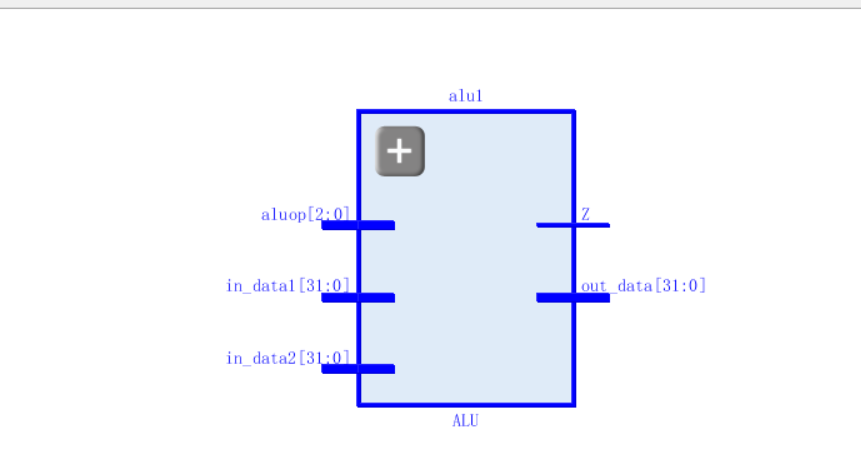
子模块：



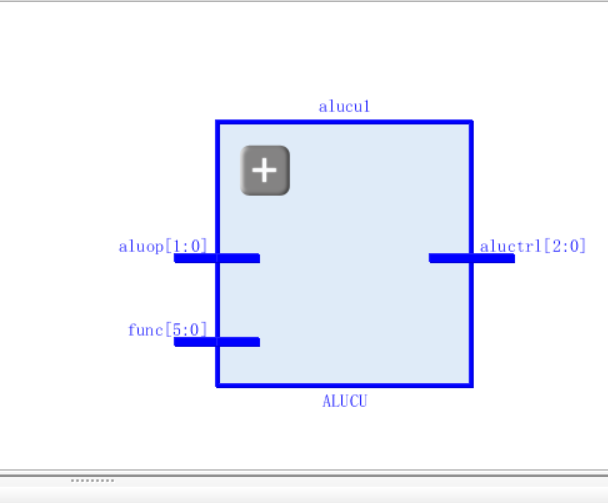
PC自加器（PC=(PC)+1）



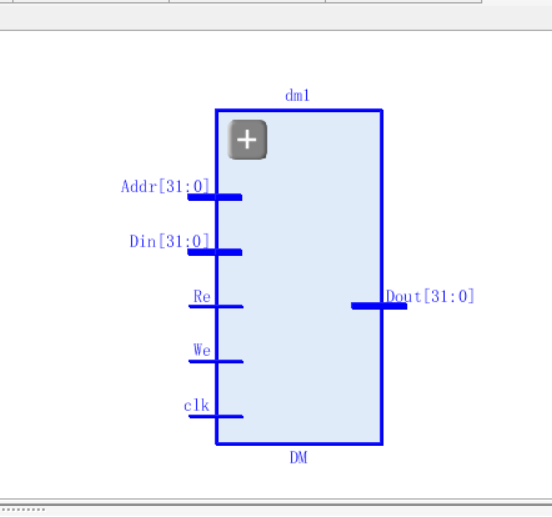
32位加法器



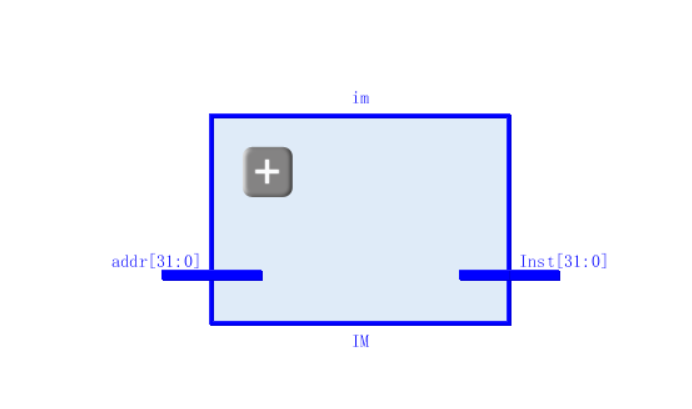
算术逻辑单元



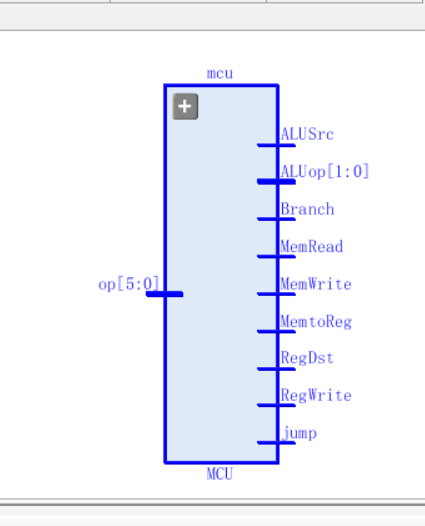
算术逻辑单元的控制单元



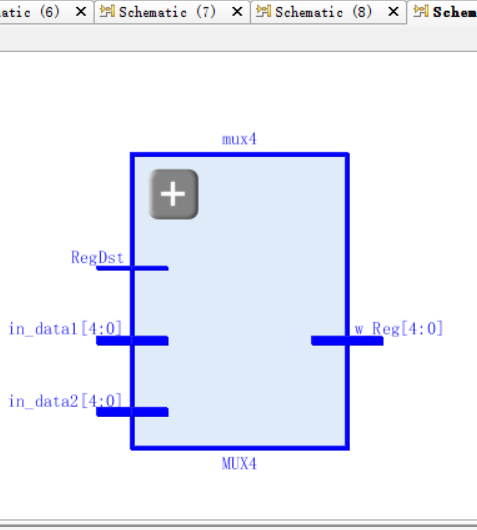
数据存储器(data memory)



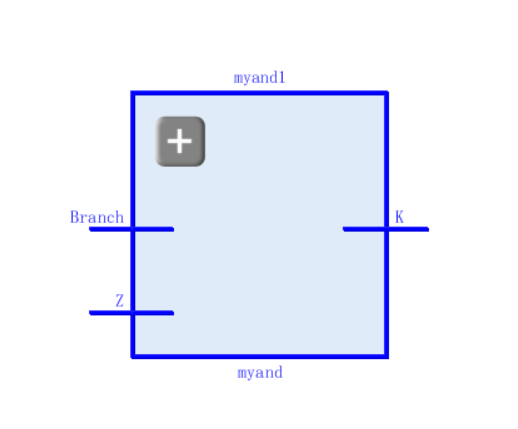
指令存储器(用于存储指令)



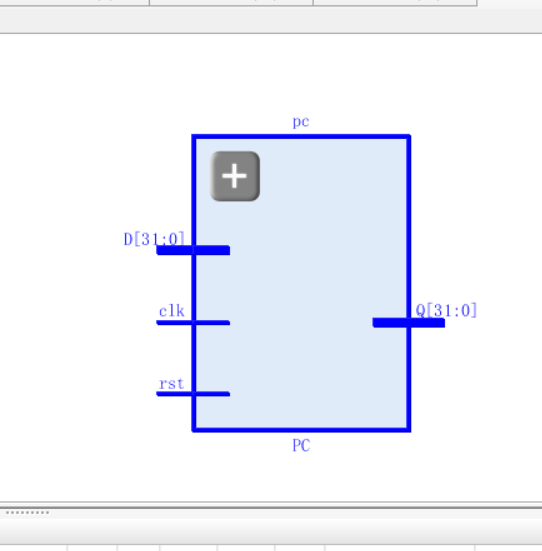
主控制单元



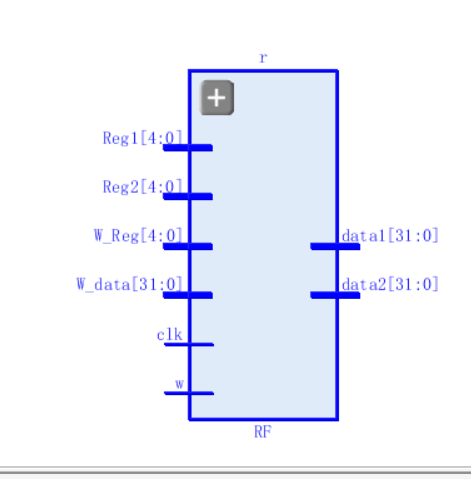
5位2路选择器



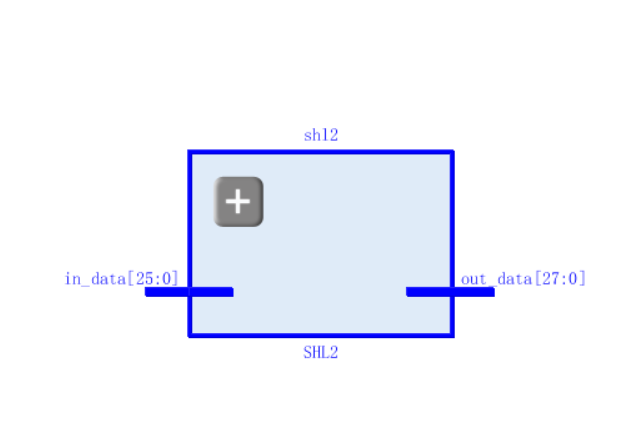
与操作



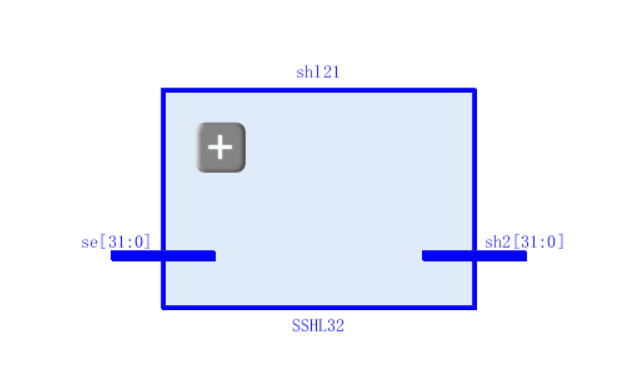
程序计数器



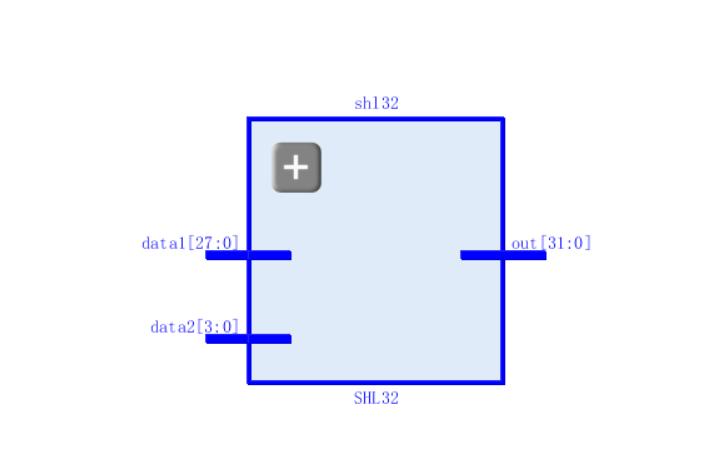
Regfile(寄存器组)



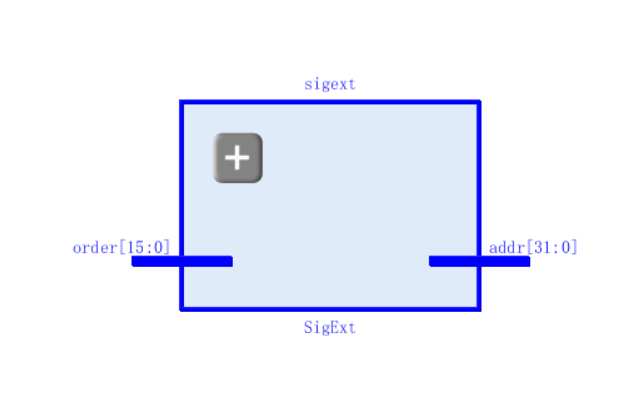
左移



32位左移



28位左移



16/32偏移器

**代码**

module PC(

clk,D,Q,rst

);

input rst;

input clk;

input [31:0]D;

output reg[31:0]Q;

always@(posedge clk)begin

if(!rst)

Q<=D;

else Q<=32'b0;

end

endmodule

module IM(

addr,Inst,

);

input [31:0]addr;

output [31:0]Inst;

reg [31:0]Im[31:0];

integer i=0;

initial begin

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

Im[i]=32'h2003000c;

Im[0]=32'h00210822;

Im[1]=32'h2003000c;

Im[2]=32'h2067fff7;

Im[3]=32'h00e22025;

Im[4]=32'h00642824;

Im[5]=32'h00a42820;

Im[6]=32'h10a7000a;

Im[7]=32'h0064202a;

Im[8]=32'h10800001;

Im[9]=32'h20050000;

Im[10]=32'h00e2202a;

Im[11]=32'h00853820;

Im[12]=32'h00e23822;

Im[13]=32'hac670044;

Im[14]=32'h8c020050;

Im[15]=32'h08000011;

Im[16]=32'h20020001;

Im[17]=32'hac020054;

end

assign Inst=Im[addr[6:2]];

endmodule

module MCU(op,RegDst,jump,Branch,MemRead,

MemtoReg,ALUop,MemWrite,ALUSrc,RegWrite);

input [5:0] op;

output reg RegDst,jump,Branch,MemtoReg,ALUSrc,RegWrite,MemWrite,MemRead;

output reg [1:0] ALUop;

always@(\*)

begin

case(op)

//R型指令

6'b000000:begin

RegDst <= 1; jump <= 0; ALUop <= 2'b10;

Branch <= 0; MemtoReg <= 0; RegWrite <= 1;

MemWrite <= 0; MemRead <= 0; ALUSrc <= 0;end

//I型lw存指令

6'b100011:begin

RegDst <= 0; jump <= 0; ALUop <= 2'b00;

Branch <= 0; MemtoReg <= 1; RegWrite <= 1;

MemWrite <= 0; MemRead <= 1; ALUSrc <= 1;end

//I型sw取指令

6'b101011:begin

RegDst <= 0; jump <= 0; ALUop <= 2'b00;

Branch <= 0; MemtoReg <= 0; RegWrite <= 0;

MemWrite <= 1; MemRead <= 0; ALUSrc <= 1;end

//I型beq指令分支反指令

6'b000100:begin

RegDst <= 0; jump <= 0; ALUop <= 2'b01;

Branch <= 1; MemtoReg <= 0; RegWrite <= 0;

MemWrite <= 0; MemRead <= 0; ALUSrc <= 0;end

//J型指令

6'b000010:begin

RegDst <= 0; jump <= 1; ALUop <= 0;

Branch <= 0; MemtoReg <= 0; RegWrite <= 0;

MemWrite <= 0; MemRead <= 0; ALUSrc <= 0;end

default:begin

RegDst <= 0; jump <= 0; ALUop <= 0;

Branch <= 0; MemtoReg <= 0; RegWrite <= 0;

MemWrite <= 0; MemRead <= 0; ALUSrc <= 0;end

endcase

end

endmodule

module MUX4(

in\_data1,in\_data2,RegDst,w\_Reg

);

input [4:0]in\_data1,in\_data2;

input RegDst;

output [4:0]w\_Reg;

assign w\_Reg=(RegDst==0)?in\_data1:in\_data2;

endmodule

module Add1(

addr,pc

);

input [31:0]addr;

output [31:0]pc;

assign pc=addr+32'd4;

endmodule

module SHL2(

in\_data,out\_data

);

input [25:0]in\_data;

output reg[27:0]out\_data;

always@(\*)begin

out\_data[27:2]<=in\_data;

out\_data[1:0]<=2'b00;

end

endmodule

module SHL32(

data1,data2,out

);

input [3:0]data2;

input [27:0]data1;

output reg[31:0]out;

always@(\*)begin

out[31:28]<=data2;

out[27:0]<=data1;

end

endmodule

module MUX32(

in\_data1,in\_data2,src,out\_data,

);

input [31:0]in\_data1,in\_data2;

input src;

output [31:0]out\_data;

assign out\_data=(src==0)?in\_data1:in\_data2;

endmodule

module Add2(

in\_data1,in\_data2,out\_data,

);

input [31:0]in\_data1,in\_data2;

output [31:0]out\_data;

assign out\_data=in\_data1+in\_data2;

endmodule

module myand(

Branch,Z,K

);

input Branch,Z;

output K;

assign K=Branch&Z;

endmodule

module RF(

W\_data,Reg1,Reg2,W\_Reg,data1,data2,w,clk

);

input clk;

input [4:0]Reg1,Reg2,W\_Reg;

input w;//读写控制,1写0读

input [31:0]W\_data;

output [31:0]data1,data2;

reg [31:0]ram[31:0];//寄存器组

integer i=0;

initial begin

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

ram[i]=32'b0;

ram[0]=10;

ram[1]=20;

ram[2]=30;

end

assign data1=ram[Reg1];

assign data2=ram[Reg2];

always@(posedge clk)begin

if(w)ram[W\_Reg]<=W\_data;

end

endmodule

module SigExt(order,addr);

input [15:0] order;

output reg [31:0] addr;

always @(\*)

begin

if(order[15]==1) addr[31:16]=16'b1111111111111111;

else addr[31:16]=16'b0000000000000000;

addr[15:0]=order[15:0];

end

endmodule

module SSHL32(

se,sh2

);

input [31:0]se;

output reg[31:0]sh2;

always@(\*)begin

sh2[31:2]<=se[29:0];

sh2[1:0]<=2'b00;

end

endmodule

module ALU(

in\_data1,in\_data2,out\_data,aluop,Z

);

input [31:0]in\_data1,in\_data2;

input [2:0]aluop;

output reg[31:0]out\_data;

output Z;//判断两个数是否相等

always@(\*)begin

case(aluop)

3'b100:out\_data<=in\_data1+in\_data2;

3'b101:out\_data<=in\_data1+in\_data2;

3'b110:out\_data<=in\_data1-in\_data2;

3'b000:out\_data<=in\_data1&in\_data2;

3'b001:out\_data<=in\_data1|in\_data2;

3'b011:out\_data<=in\_data1^in\_data2;

default:

out\_data<=32'b0;

endcase

end

assign Z=(out\_data==0);//相等为1，相异为0

endmodule

module ALUCU(

func,aluop,aluctrl,

);

input [5:0]func;

input [1:0]aluop;

output reg[2:0]aluctrl;

always@(\*)begin

case(aluop)

2'b00:aluctrl<=3'b100;//加

2'b01:aluctrl<=3'b110;//减

default:

case(func)

6'b100000:aluctrl<=3'b100;//加

6'b100001:aluctrl<=3'b101;

6'b100010:aluctrl<=3'b110;//减

6'b100100:aluctrl<=3'b000;//与

6'b100101:aluctrl<=3'b001;//或

6'b101010:aluctrl<=3'b011;//异或

default:aluctrl<=3'bxxx;

endcase

endcase

end

endmodule

module DM(Addr,Din,We,Dout,Re,clk);

input[31:0]Addr,Din;

input We,Re,clk;

output[31:0]Dout;

reg[31:0]Ram[31:0];

integer i=0;

initial begin

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

Ram[i]<=32'b0;

Ram[0]=20;

Ram[1]=10;

Ram[2]=30;

Ram[3]=5;

end

always@(posedge clk)begin

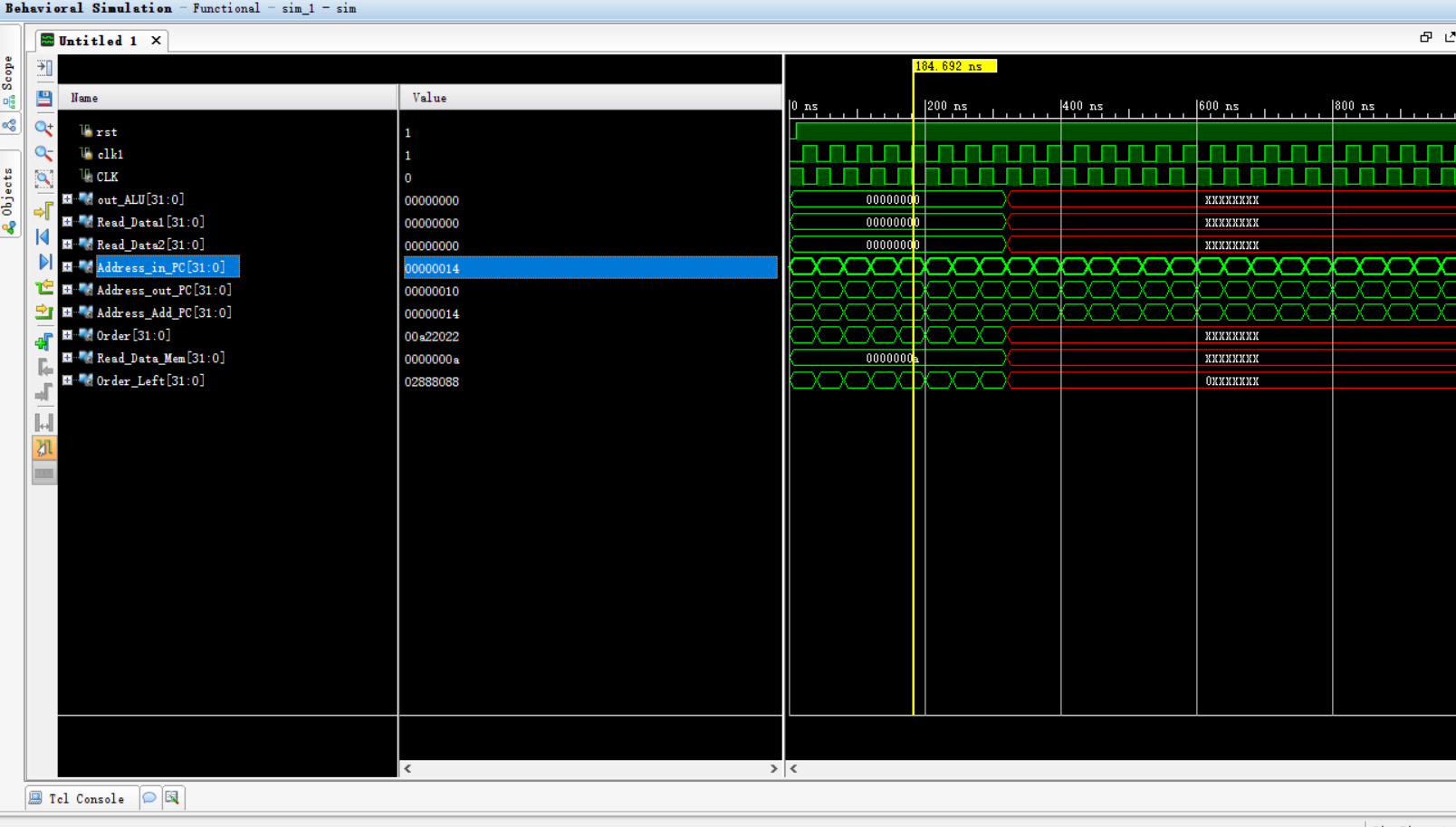
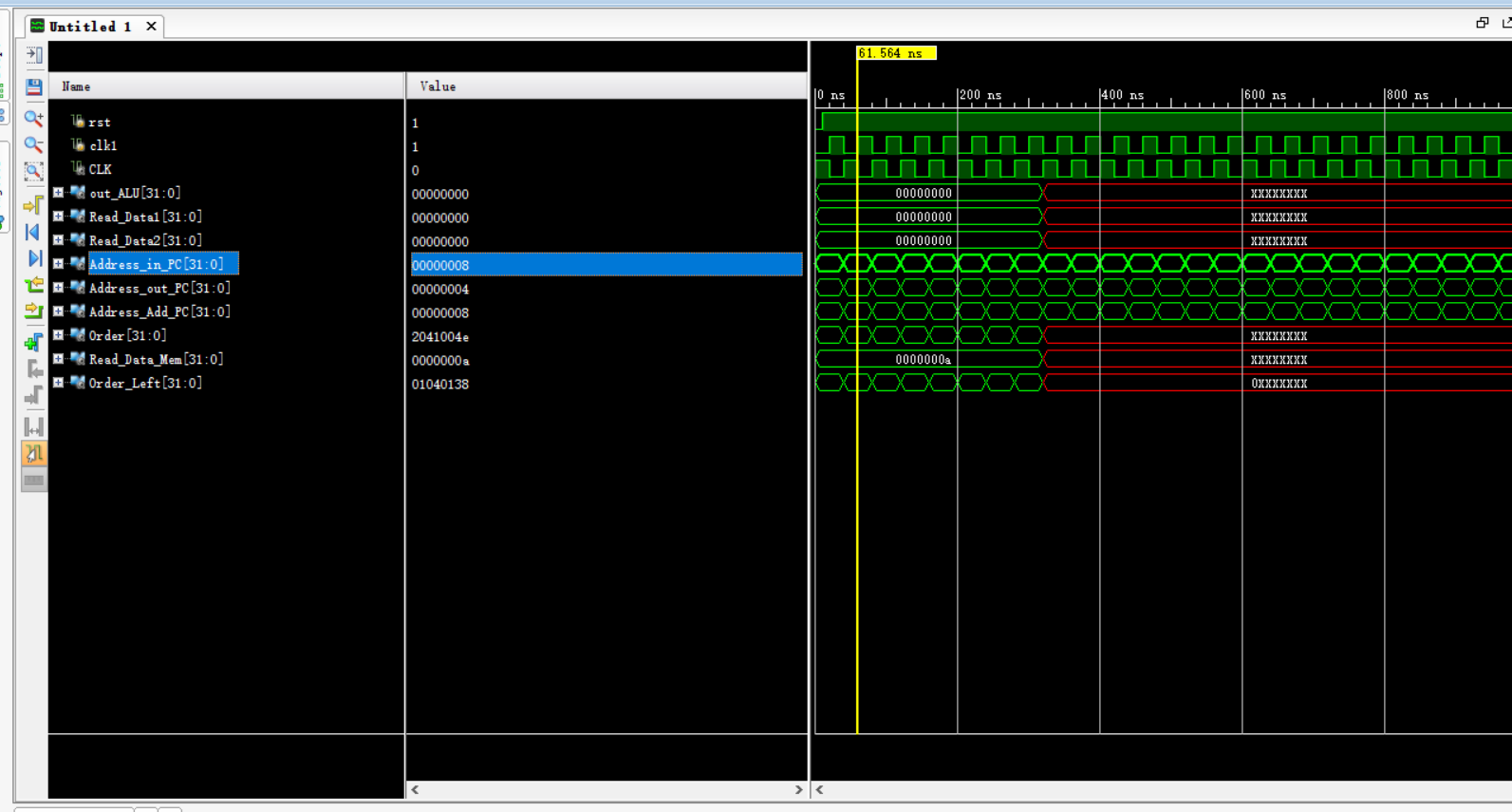
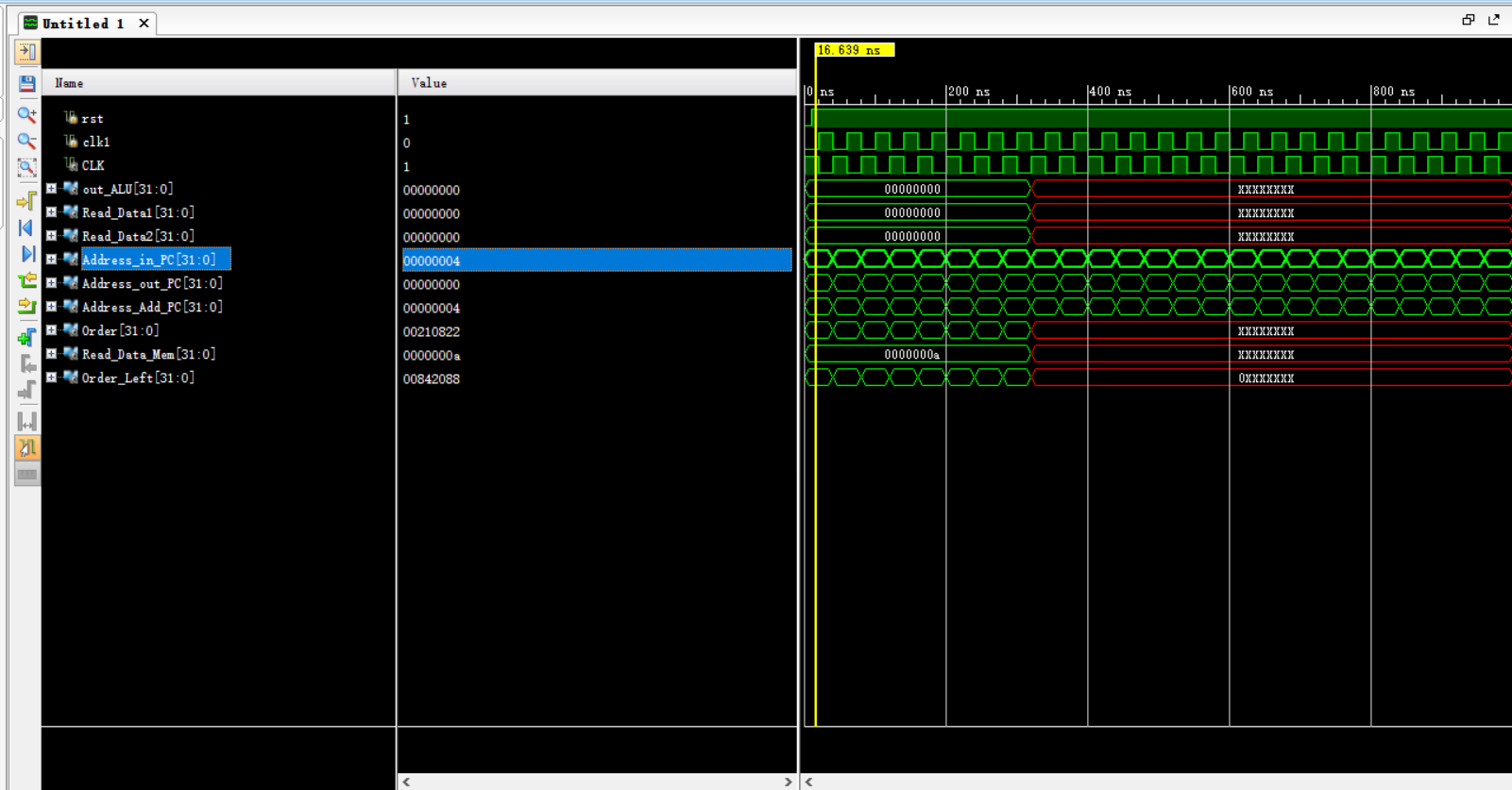
if(We)Ram[Addr]<=Din;

end

assign Dout=Ram[Addr];

endmodule

仿真截图



**五、调试和心得体会**

这次CPU的仿真实验是所有实验中最难的而且最复杂的。在接到任务时我们还没有学到CPU，对CPU的概念只有最基本的控制器、算术器和一些寄存器总线等。为此甚至想过放弃，但是后来还是选择翻教科书，没想到教科书给了很多详细的解释和指令的使用方法，终于花了一天半的时间仿真＋调试完成了本次实验。