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。

**各模块流程示意图：**

CPU

Instruction Memory

Data

Memory

Instruction

DataRead

Data Addr

DataWrite

Instruction Addr

单周期MIPS顶层模块结构图

Main decoder

ALUDecoder

Alu

Register

Instruction

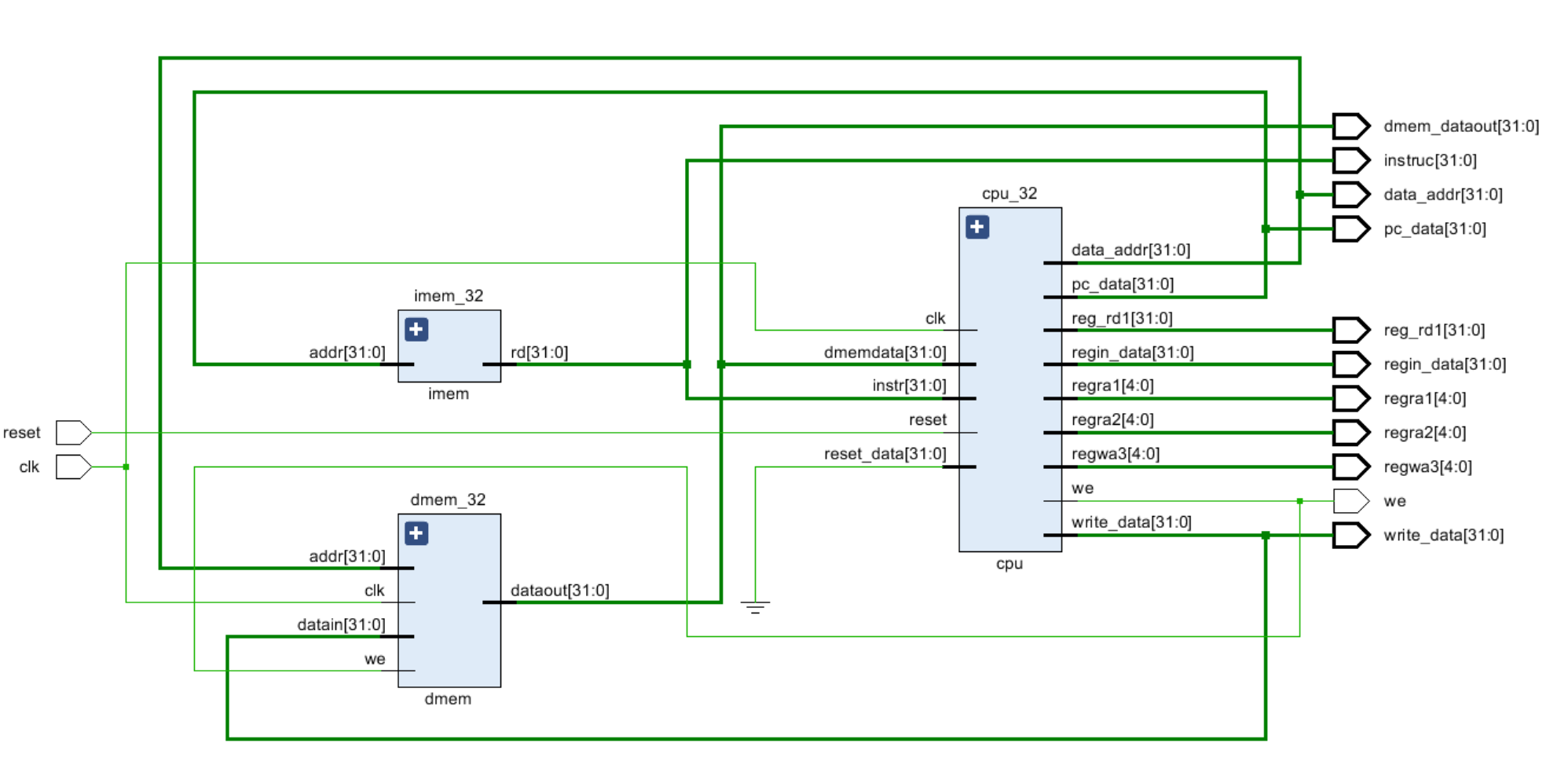
Next Instr Addr

PC

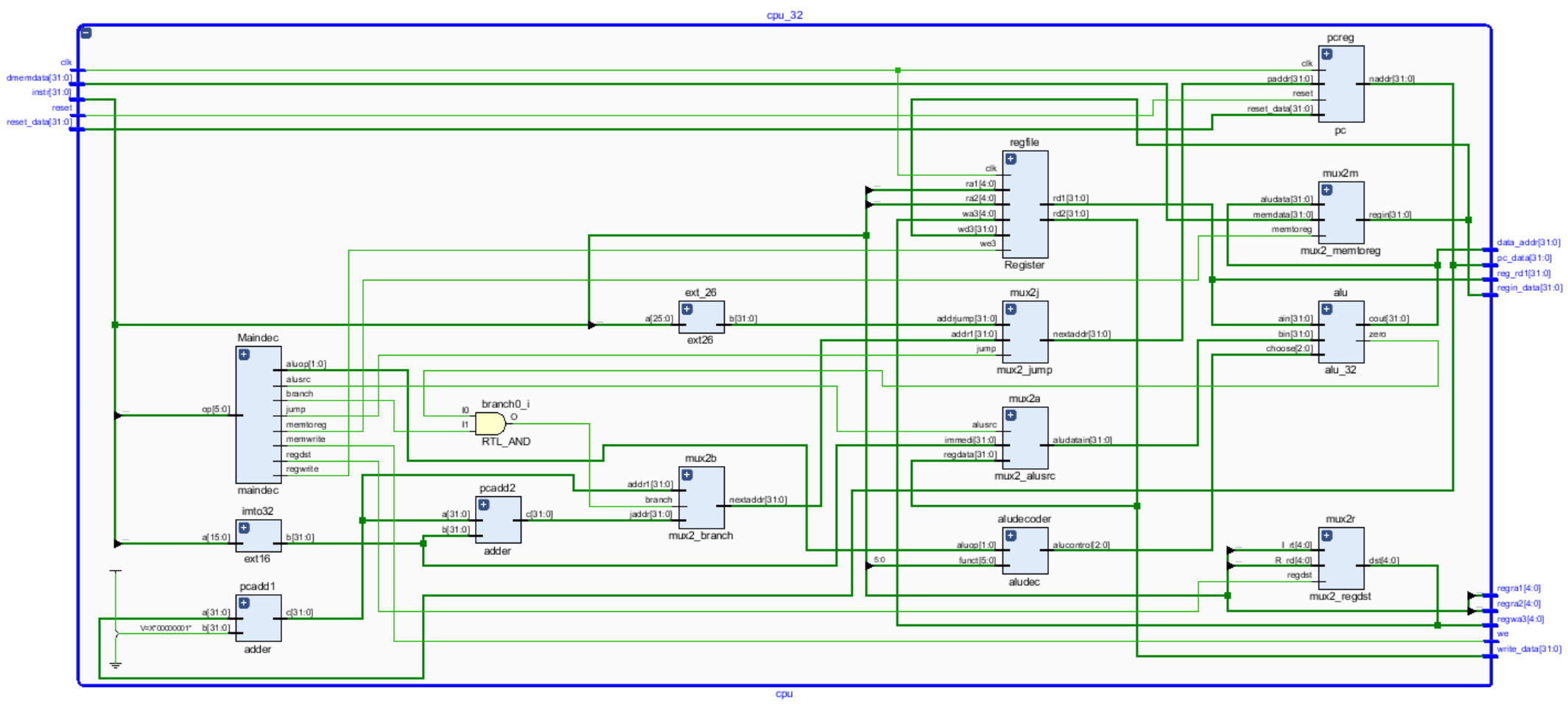
CPU模块结构图

**四、实验代码及结果**

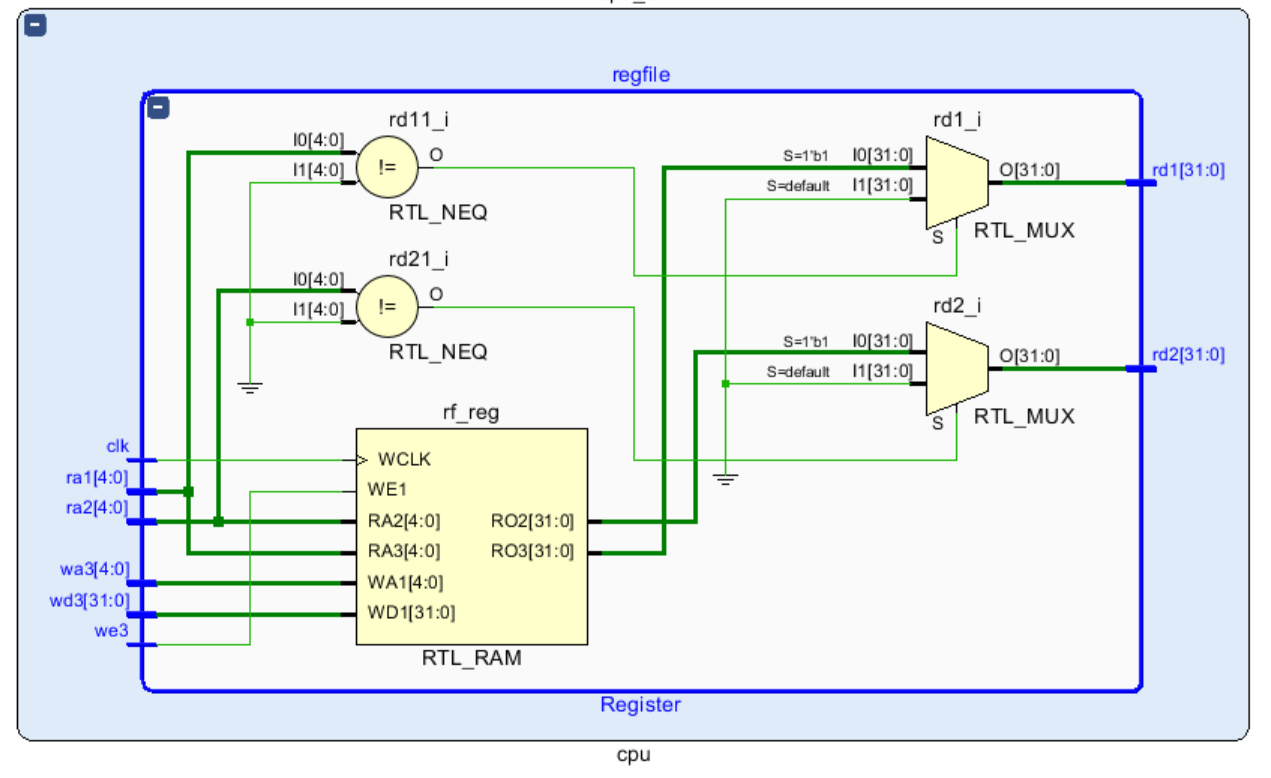
**各模块实现：**



顶层模块



CPU模块

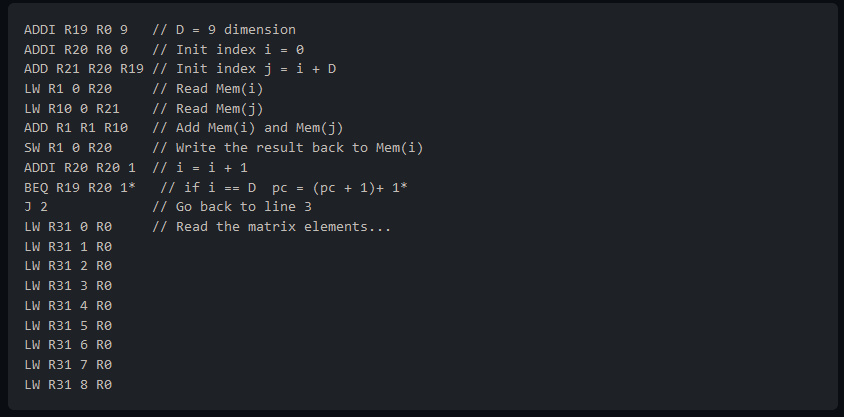


寄存器模块

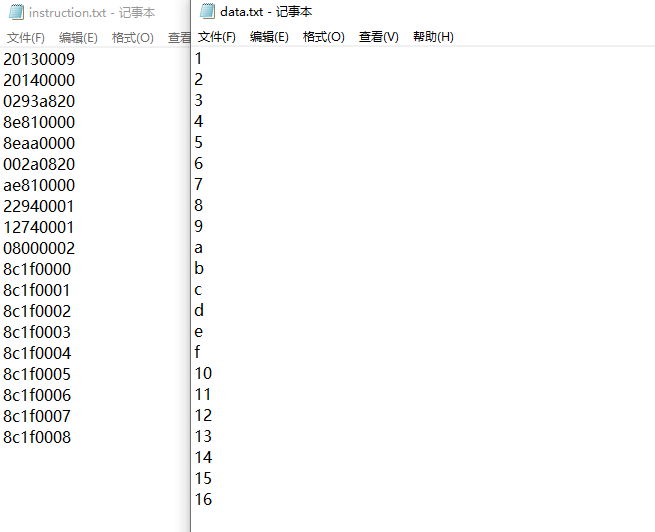
**仿真所用指令及波形：**

**实现了从内存中读取数据，九维矩阵各对应元素相加循环跳转。将结果写回内存中。**

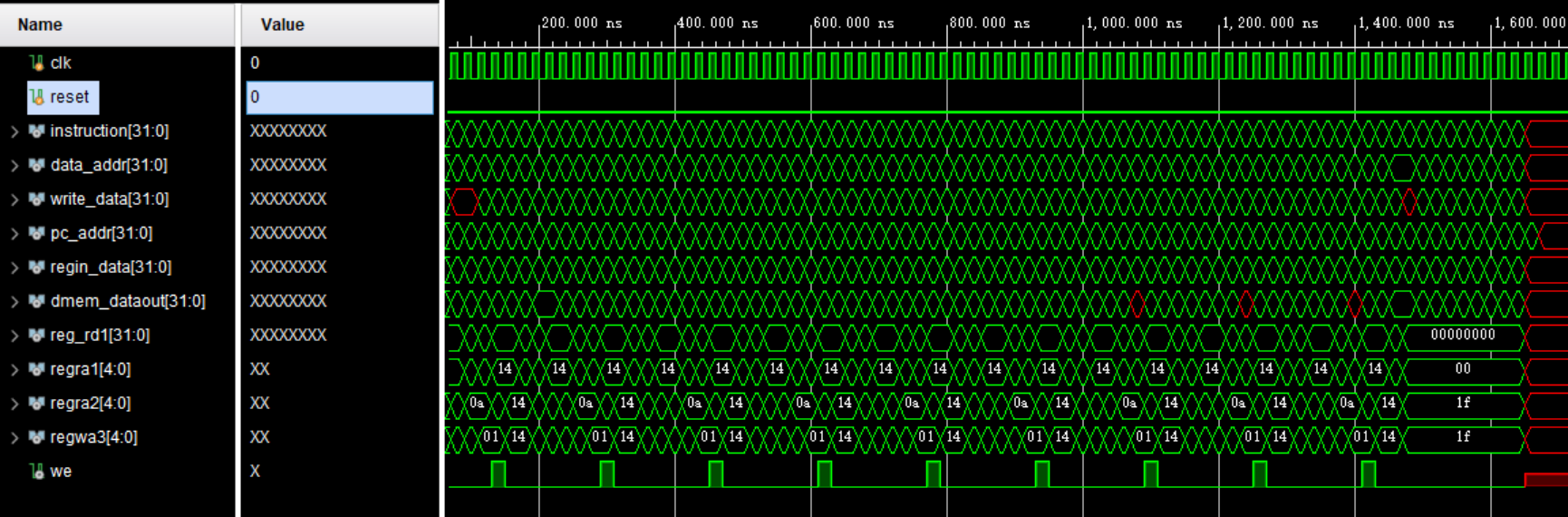
**各指令含义已注释在指令旁边，如下图所示。**





****

**通过$readmemh 在Memory中初始化**



仿真总览

**关键部分Top，CPU，Maindecoder，Register, ALUDecoder, ALU 代码如下**

**Top:**

**module top(**

**input clk,**

**input reset,**

**output [31:0]instruc,**

**output [31:0]data\_addr,**

**output [31:0]write\_data,**

**output [31:0]pc\_data,**

**output [31:0]regin\_data,reg\_rd1,**

**output [4:0]regra1,regra2,regwa3,**

**output [31:0]dmem\_dataout,**

**output we**

**);**

**wire [31:0]pc\_addr,d\_addr,w\_data;**

**wire [31:0]wd,da,instr,readdata;**

**wire memwrite,mem\_toreg;**

**assign data\_addr=d\_addr;**

**assign write\_data=w\_data;**

**assign we=memwrite;**

**assign instruc=instr;**

**assign pc\_data=pc\_addr;**

**assign dmem\_dataout=readdata;**

**cpu cpu (clk,instr, reset,32'h00000000,readdata,pc\_addr,d\_addr,w\_data,regin\_data,reg\_rd1,regra1,regra2,regwa3,memwrite);**

**dmem dmem\_32(clk,memwrite,d\_addr,w\_data,readdata);**

**imem imem\_32(pc\_addr,instr);**

**endmodule**

**Cpu:**

**module cpu(**

**input clk,**

**input [31:0] instr,**

**input reset,**

**input [31:0]reset\_data,**

**input [31:0]dmemdata,**

**output [31:0]pc\_data,**

**output [31:0]data\_addr,**

**output [31:0]write\_data,**

**output [31:0]regin\_data,reg\_rd1,**

**output [4:0]regra1,regra2,regwa3,**

**output we**

**);**

**wire regwrite,regdst,alusrc,branch,memwrite,memtoreg,jump;**

**wire zero;**

**wire [5:0]op\_code,funct;**

**wire [27:0]in\_join;**

**assign in\_join=instr[25:0]<<2;**

**assign op\_code=instr[31:26];**

**assign funct=instr[5:0];**

**wire [1:0]aluop;**

**wire [2:0]alucontrol;**

**wire [4:0]ra1,ra2,wa3;**

**wire [31:0]wd3,rd1,rd2,alu\_out;**

**assign ra1=instr[25:21];**

**assign ra2=instr[20:16];**

**assign regra1=ra1;**

**assign regra2=ra2;**

**assign regwa3=wa3;**

**assign regin\_data=wd3;**

**wire [31:0]im32,alu\_in;**

**wire [31:0]pc\_addr,pc\_nojump,pc\_jumpbeq,pc\_jump,pc\_afterb,pc\_afterj,pc1;**

**assign pc\_data=pc\_addr;**

**assign we=memwrite;**

**assign data\_addr=alu\_out;**

**assign write\_data=rd2;**

**assign reg\_rd1=rd1;**

**Maindec：**

**module maindec(input [5:0] op,**

**output regwrite,regdst,alusrc,branch,memwrite,memtoreg,jump,**

**output [1:0] aluop);**

**reg [8:0]controls;**

**assign {**

**regwrite,**

**regdst,**

**alusrc,**

**branch,**

**memwrite,**

**memtoreg,**

**jump,**

**aluop**

**}=controls;**

**always @(\*)**

**begin**

**case(op)**

**6'b000000: controls <=9'b110000010;//Rtype**

**6'b100011: controls <=9'b101001000;//LW**

**6'b101011: controls <=9'b001010000;//SW**

**6'b000100: controls <=9'b000100001;//BEQ**

**6'b001000: controls <=9'b101000000;//Addi**

**6'b000010: controls <=9'b000000100;//J**

**default: controls <=9'bxxxxxxxxx;**

**endcase**

**end**

**endmodule**

**Register:**

module Register(input clk,

input we3,

input [4:0] ra1,ra2,wa3,

input [31:0] wd3,

output [31:0] rd1,rd2);

reg [31:0] rf[31:0];

always @ (posedge clk)

if (we3) rf[wa3] <=wd3;

assign rd1=(ra1!=0)?rf[ra1]:0;

assign rd2=(ra2!=0)?rf[ra2]:0;

endmodule

**Aludec:**

module aludec(input [5:0] funct,

input [1:0] aluop,

output reg [2:0] alucontrol);

always @ (\*)

case (aluop)

2'b00: alucontrol <=3'b010;//add

2'b01: alucontrol <=3'b110;//sub

default:case(funct)

6'b100000:alucontrol <=3'b010;//ad

6'b100010:alucontrol <=3'b110;//sub

6'b100100:alucontrol <=3'b000;//and

6'b100101:alucontrol <=3'b001;//or

6'b101010:alucontrol <=3'b100;

endcase

endcase

endmodule

**ALU:**

module alu(

ain,bin,cout,choose,zero

);

input [31:0] ain,bin;

input [2:0]choose;

output reg [31:0] cout;

output reg zero;

always@(\*)

begin

case(choose)

3'b010:

begin

cout=ain+bin;

end

3'b110:

begin

cout=ain-bin;

end

3'b000:cout=ain&bin;

3'b001:cout=ain|bin;

3'b100:

begin

if(ain<bin)cout=32'b1;

else cout=32'h00000000;

end

endcase

if(cout!=0) zero =0;

else zero=1;

end

endmodule

五、调试和心得体会

调试比较辛苦，对cpu理解更深刻了。