**单周期CPU设计**

李卓 pb19000064

**实验目的**

理解CPU的结构和工作原理

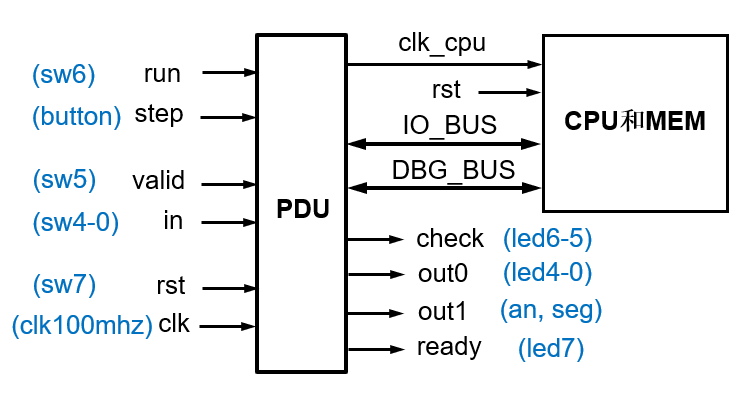
掌握单周期CPU的设计和调试方法

熟练掌握数据通路和控制器的设计和描述方法

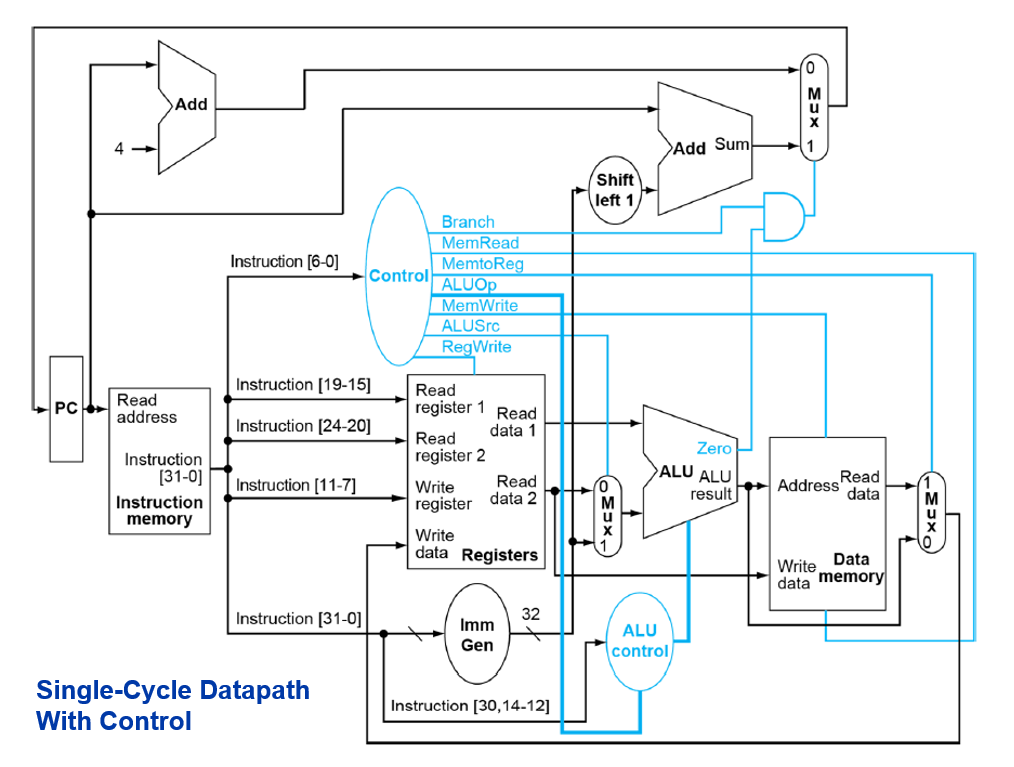
**实验原理**

设计实现单周期RISC-V CPU，可执行以下10条指令

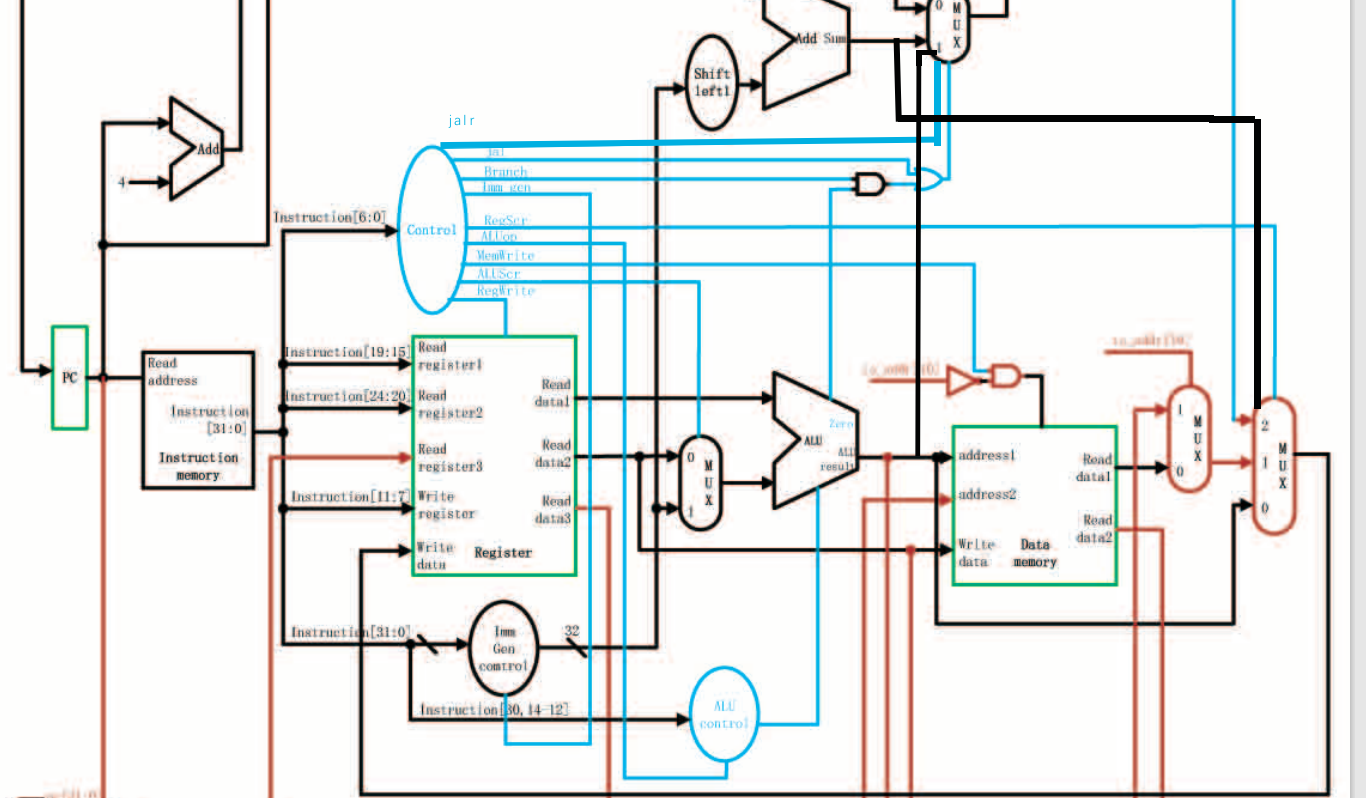
* + add, addi, sub, auipc, lw, sw, beq, blt, jal, jalr



单周期CPU数据通路



该通路只能实现原有的六条指令，对于新加的四条指令(auipc, blt, jalr sum),需要修改通路。



sub：不需特殊处理

auipc：

连接regscr的mux 和pc\_adder。执行时pc\_adder将pc和立即数相加，RegSrc=11，mux选择pc adder 的结果sum

blt:

根据funct3位判断blt指令。alu\_control向alu发送操作符LT:101，alu将两数相减 最高位为一则说明小于，zero置为1，其他与beq指令完全相同

jalr：

control unit增加控制信号 jalr。连接alu的结果与pc\_mux。执行时，jalr为1，pc\_mux输入为3，选择alu相加的结果写入pc。RegSrc=10, 原pc+4的结果写入寄存器中

io\_bus信号

CPU运行时访问开关(sw)、指示灯(led)和数码管(an, seg）

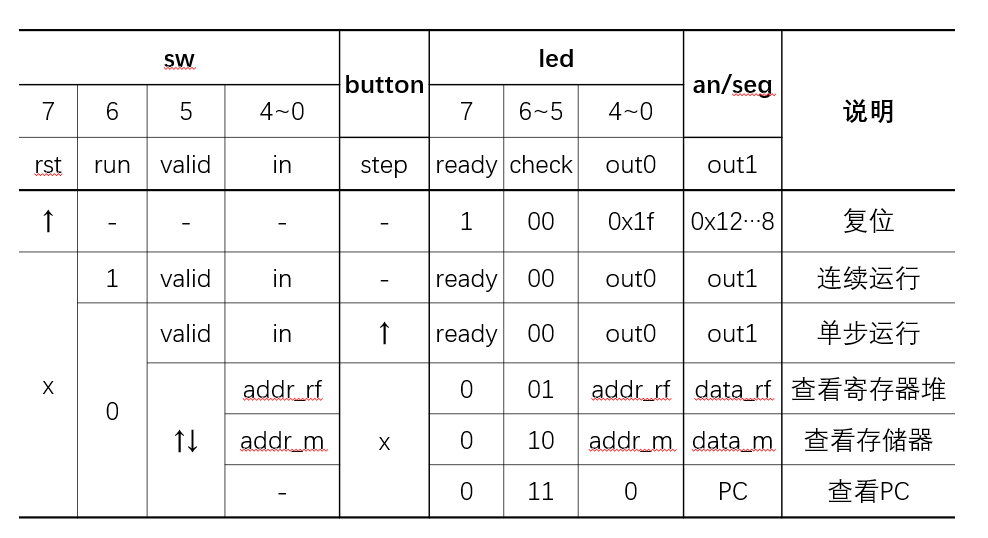
* + io\_addr：I/O外设的地址
  + io\_din：CPU接收来自输入缓冲寄存器（IBR）的sw输入数据
  + io\_dout：CPU向led和seg输出的数据
  + io\_we：CPU向led和seg输出时的使能信号，利用该信号将io\_dout存入输出缓冲寄存器（OBR），再经数码管显示电路将其显示在数码管（an，seg）

debug\_bus信号

调试时将存储器和寄存器堆内容，以及CPU数据通路状态信息导出显示

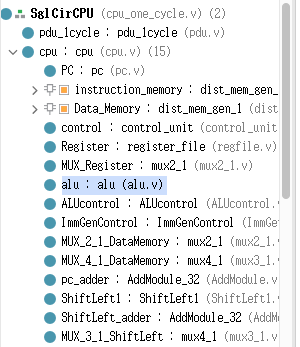
* + m\_rf\_addr：存储器(MEM)或寄存器堆(RF)的调试读口地址
  + rf\_data：从RF读取的数据
  + m\_data：从MEM读取的数据
  + pc：PC的内容

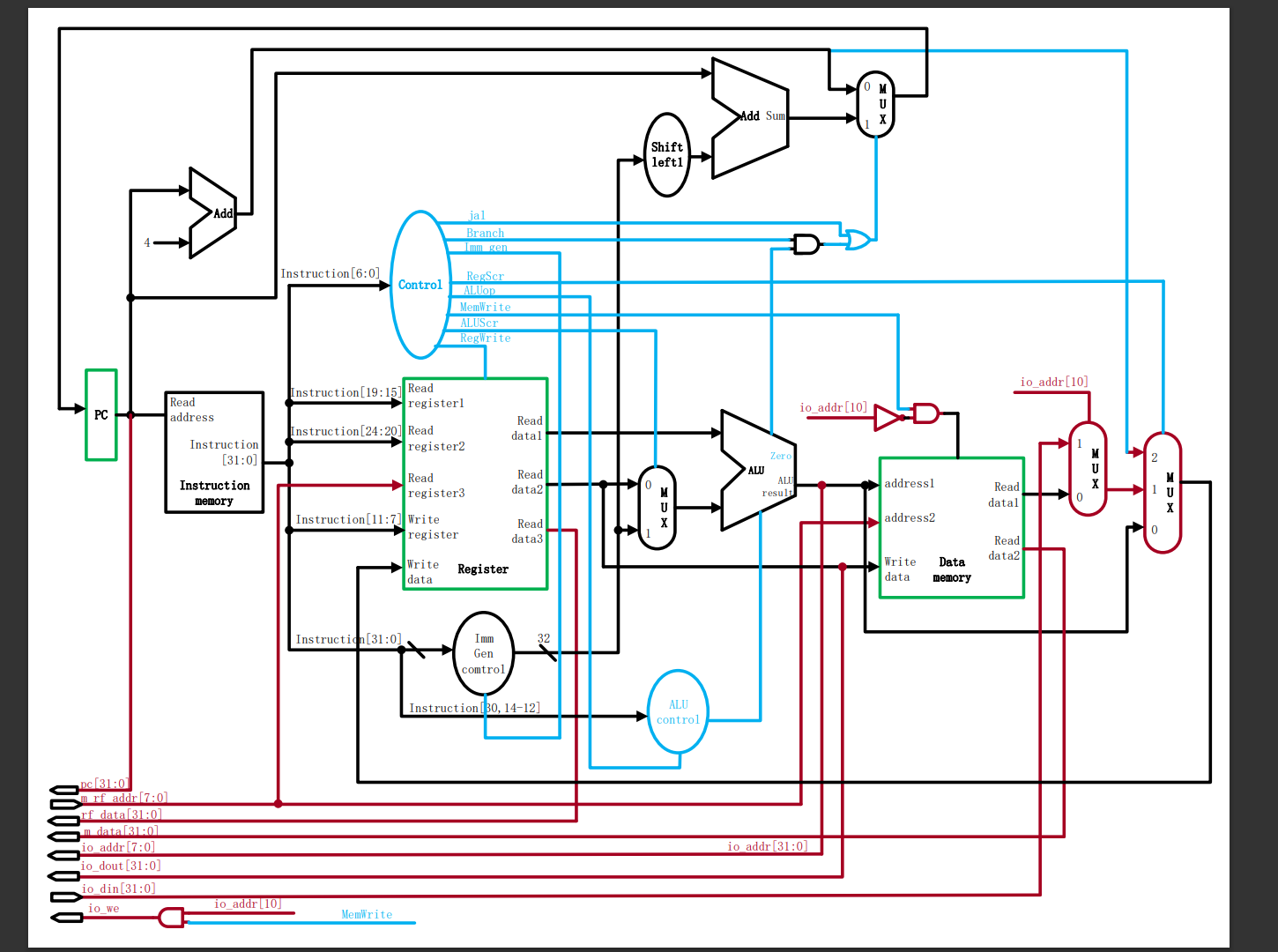
pdu 运行方式：



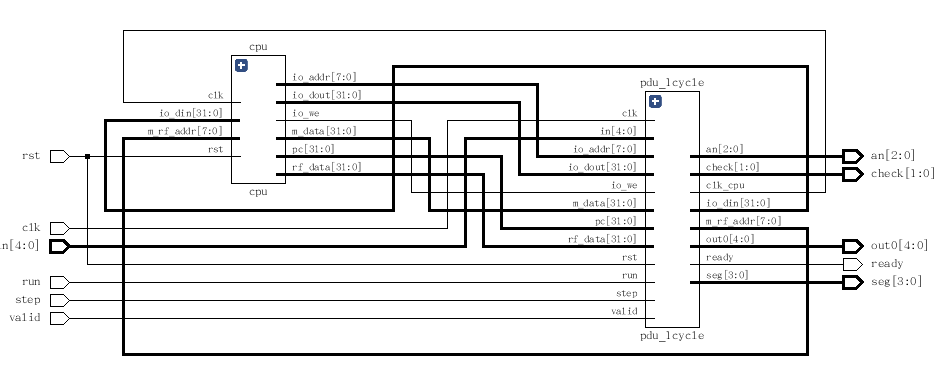
**实验过程**

源文件结构：



各个模块定义如下图  


电路



下载测试

***FibFrMem.s***

***FibFrMem\_IM.coe***

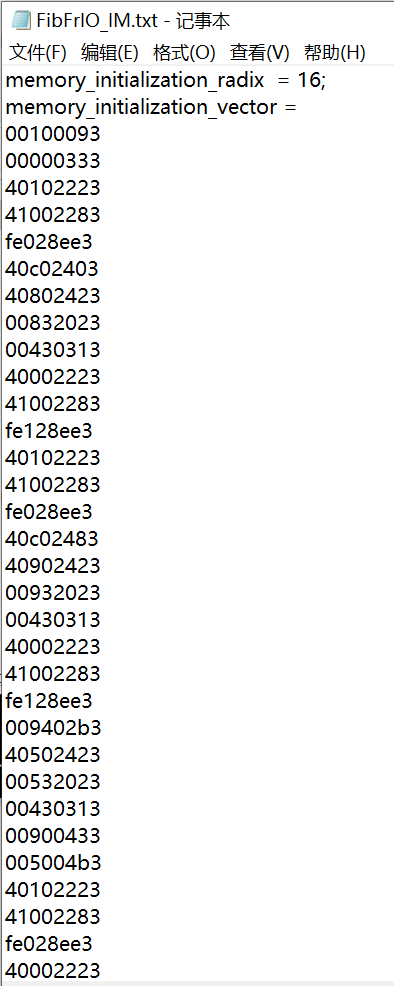
***FibFrMem\_DM.coe***

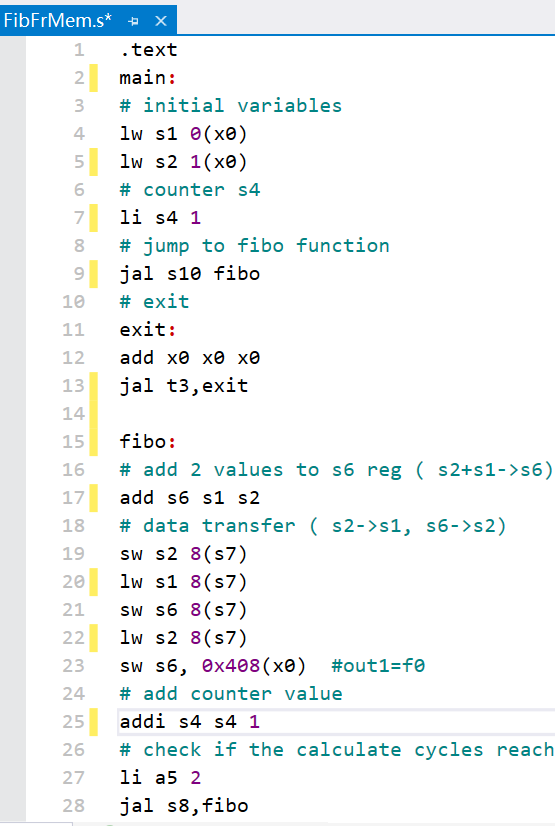
***FibFrMem.bit***

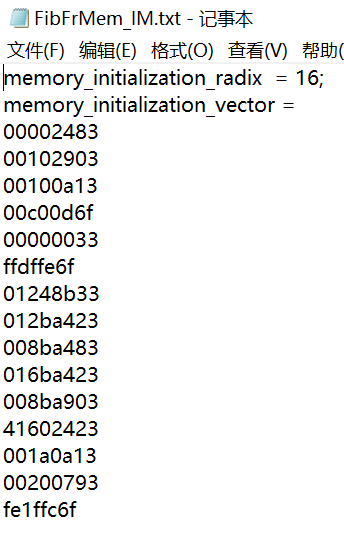
由Mem生成fib数列, 分别为 汇编源文件，指令段coe，数据段coe，下载结果

***FibFrIO.bit***

由IO生成fib数列 的下载结果, 指令段coe用老师给的

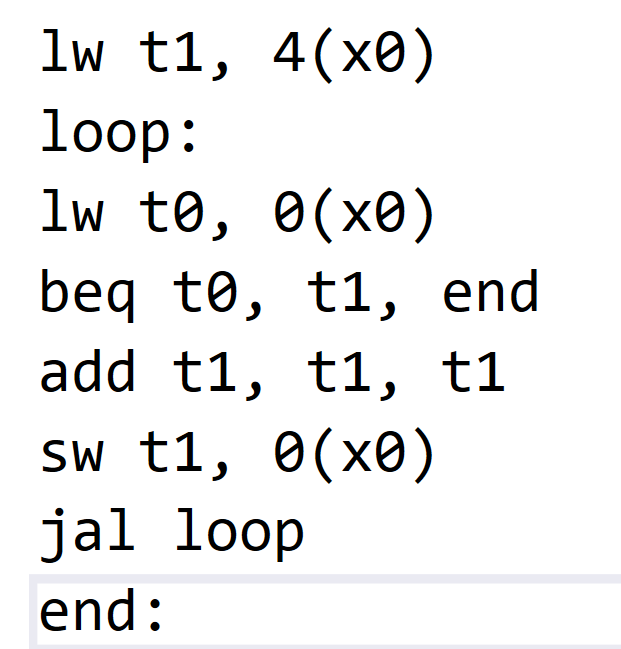




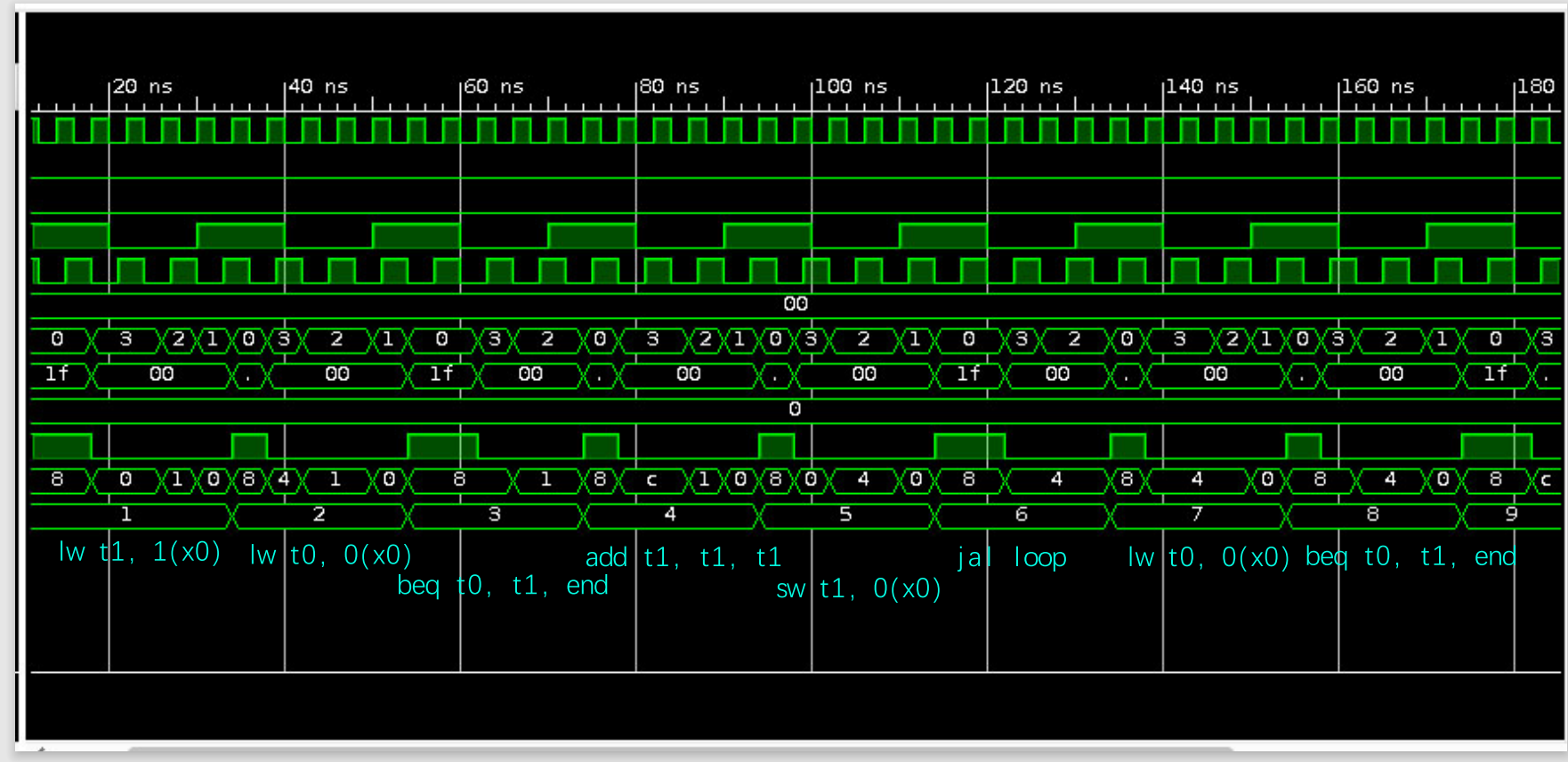


仿真测试：

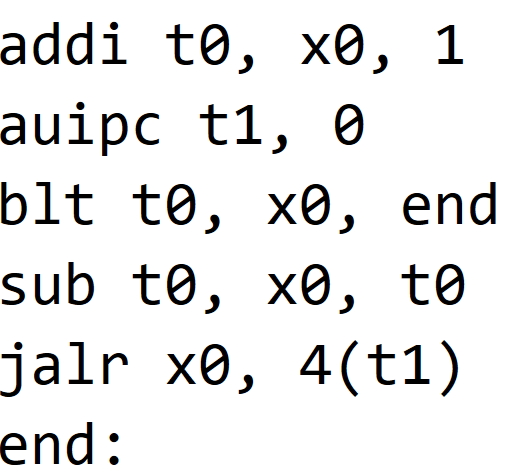
***test1.s***



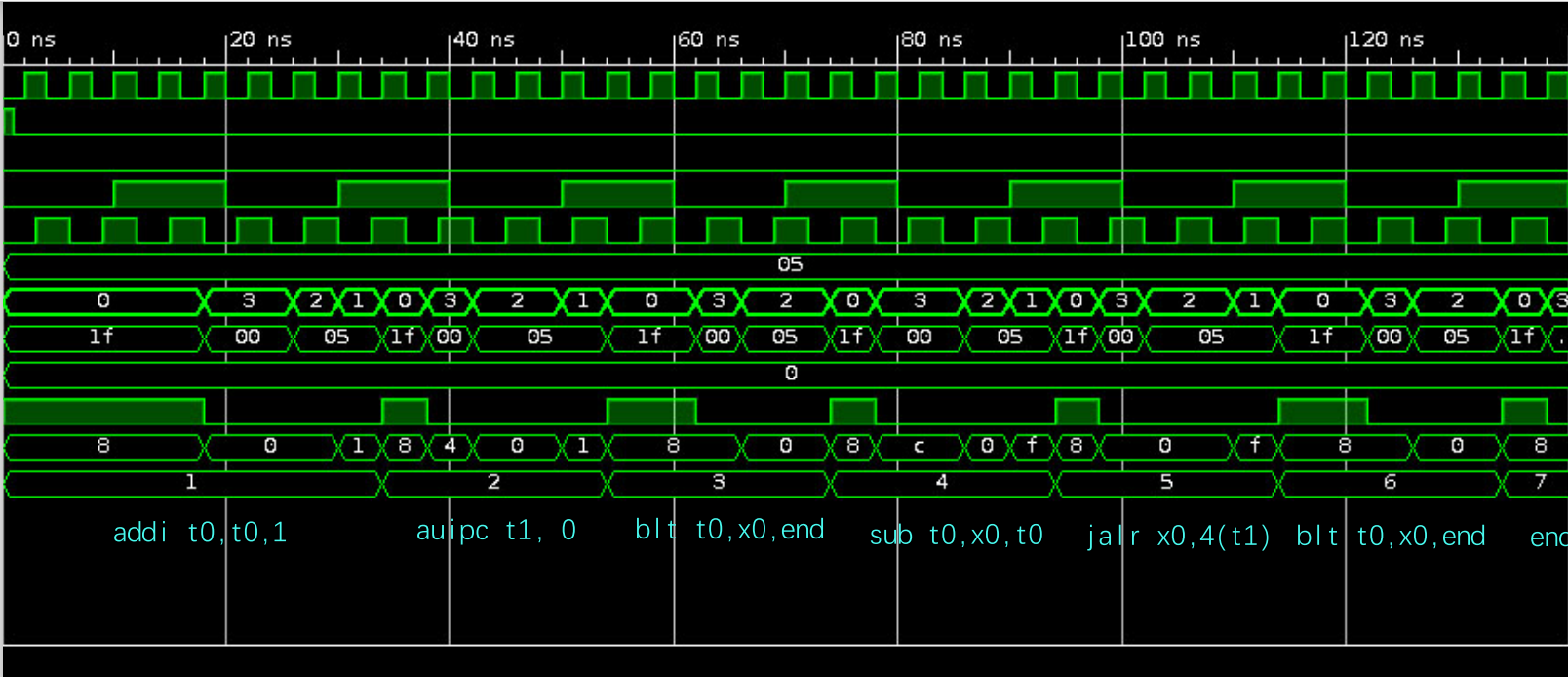
生成的波形图 ***wave1.pdf***



***test2.s***



生成的波形图 ***wave2.pdf***



**实验代码**

SglCirCpu.v

*module SglCirCPU(*

*input run, //sw6*

*input step, //button*

*input valid,//sw5*

*input [4:0]in,//sw4-0*

*input rst, //sw7*

*input clk, //clk100mhz*

*output [1:0]check, //led6-5*

*output [4:0] out0, //led4-0*

*output [2:0] an, //8个数码管*

*output [3:0] seg,*

*output ready //led7*

*);*

*wire clk\_cpu,io\_we;*

*wire[7:0]io\_addr,m\_rf\_addr;*

*wire[31:0]io\_dout,io\_din,rf\_data,m\_data,pc;*

*pdu\_1cycle pdu\_1cycle(*

*.clk(clk),*

*.rst(rst),*

*//选择CPU工作方式;*

*.run(run),*

*.step(step),*

*.clk\_cpu(clk\_cpu),*

*//输入switch的端口*

*.valid(valid),*

*.in(in),*

*//输出led和seg的端口*

*.check(check), //led6-5:查看类型*

*.out0(out0), //led4-0*

*.an(an), //8个数码管*

*.seg(seg),*

*.ready(ready), //led7*

*//IO\_BUS*

*.io\_addr(io\_addr),*

*.io\_dout(io\_dout),*

*.io\_we(io\_we),*

*.io\_din(io\_din),*

*//Debug\_BUS*

*.m\_rf\_addr(m\_rf\_addr),*

*.rf\_data(rf\_data),*

*.m\_data(m\_data),*

*.pc(pc)*

*);*

*cpu cpu (*

*.clk(clk\_cpu),*

*.rst(rst),*

*//IO\_BUS*

*.io\_addr(io\_addr), //led和seg的地址*

*.io\_dout(io\_dout), //输出led和seg的数据*

*.io\_we(io\_we), //输出led和seg数据时的使能信号*

*.io\_din(io\_din), //来自sw的输入数据*

*//Debug\_BUS*

*.m\_rf\_addr(m\_rf\_addr), //存储器(MEM)或寄存器堆(RF)的调试读口地址*

*.rf\_data(rf\_data), //从RF读取的数据*

*.m\_data(m\_data), //从MEM读取的数据*

*.pc(pc) //PC的内容*

*);*

*endmodule*

pdu.v

*module pdu\_1cycle(*

*input clk,*

*input rst,*

*//选择CPU工作方式;*

*input run,*

*input step,*

*output clk\_cpu,*

*//输入switch的端口*

*input valid,*

*input [4:0] in,*

*//输出led和seg的端口*

*output [1:0] check, //led6-5:查看类型*

*output [4:0] out0, //led4-0*

*output [2:0] an, //8个数码管*

*output [3:0] seg,*

*output ready, //led7*

*//IO\_BUS*

*input [7:0] io\_addr,*

*input [31:0] io\_dout,*

*input io\_we,*

*output [31:0] io\_din,*

*//Debug\_BUS*

*output [7:0] m\_rf\_addr,*

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

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

*input [31:0] pc*

*);*

*reg [4:0] in\_r; //同步外部输入用*

*reg run\_r, step\_r, step\_2r, valid\_r, valid\_2r;*

*wire step\_p, valid\_pn; //取边沿信号*

*reg clk\_cpu\_r; //寄存器输出CPU时钟*

*reg [4:0] out0\_r; //输出外设端口*

*reg [31:0] out1\_r;*

*reg ready\_r;*

*reg [19:0] cnt; //刷新计数器，刷新频率约为95Hz*

*reg [1:0] check\_r; //查看信息类型, 00-运行结果，01-寄存器堆，10-存储器，11-PC*

*reg [7:0] io\_din\_a; //\_a表示为满足组合always描述要求定义的，下同*

*reg ready\_a;*

*reg [4:0] out0\_a;*

*reg [31:0] out1\_a;*

*reg [3:0] seg\_a;*

*assign clk\_cpu = clk\_cpu\_r;*

*assign io\_din = io\_din\_a;*

*assign check = check\_r;*

*assign out0 = out0\_a;*

*assign ready = ready\_a;*

*assign seg = seg\_a;*

*assign an = cnt[19:17];*

*assign step\_p = step\_r & ~step\_2r; //取上升沿*

*assign valid\_pn = valid\_r ^ valid\_2r; //取上升沿或下降沿*

*assign m\_rf\_addr = {{3{1'b0}}, in\_r};*

*//同步输入信号*

*always @(posedge clk) begin*

*run\_r <= run;*

*step\_r <= step;*

*step\_2r <= step\_r;*

*valid\_r <= valid;*

*valid\_2r <= valid\_r;*

*in\_r <= in;*

*end*

*//CPU工作方式*

*always @(posedge clk, posedge rst) begin*

*if(rst)*

*clk\_cpu\_r <= 0;*

*else if (run\_r)*

*clk\_cpu\_r <= ~clk\_cpu\_r;*

*else*

*clk\_cpu\_r <= step\_p;*

*end*

*//读外设端口*

*always @\* begin*

*case (io\_addr)*

*8'h0c: io\_din\_a = {{27{1'b0}}, in\_r};*

*8'h10: io\_din\_a = {{31{1'b0}}, valid\_r};*

*default: io\_din\_a = 32'h0000\_0000;*

*endcase*

*end*

*//写外设端口*

*always @(posedge clk, posedge rst) begin*

*if (rst) begin*

*out0\_r <= 5'h1f;*

*out1\_r <= 32'h1234\_5678;*

*ready\_r <= 1'b1;*

*end*

*else if (io\_we)*

*case (io\_addr)*

*8'h00: out0\_r <= io\_dout[4:0];*

*8'h04: ready\_r <= io\_dout[0];*

*8'h08: out1\_r <= io\_dout;*

*default: ;*

*endcase*

*end*

*//LED和数码管查看类型*

*always @(posedge clk, posedge rst) begin*

*if(rst)*

*check\_r <= 2'b00;*

*else if(run\_r)*

*check\_r <= 2'b00;*

*else if (step\_p)*

*check\_r <= 2'b00;*

*else if (valid\_pn)*

*check\_r <= check - 2'b01;*

*end*

*//LED和数码管显示内容*

*always @\* begin*

*ready\_a = 1'b0;*

*case (check\_r)*

*2'b00: begin*

*out0\_a = out0\_r;*

*out1\_a = out1\_r;*

*ready\_a = ready\_r;*

*end*

*2'b01: begin*

*out0\_a = in\_r;*

*out1\_a = rf\_data;*

*end*

*2'b10: begin*

*out0\_a = in\_r;*

*out1\_a = m\_data;*

*end*

*2'b11: begin*

*out0\_a = 5'b00000;*

*out1\_a = pc;*

*end*

*endcase*

*end*

*//扫描数码管*

*always @(posedge clk, posedge rst) begin*

*if (rst) cnt <= 20'h0\_0000;*

*else cnt <= cnt + 20'h0\_0001;*

*end*

*always @\* begin*

*case (an)*

*3'd0: seg\_a = out1\_a[3:0];*

*3'd1: seg\_a = out1\_a[7:4];*

*3'd2: seg\_a = out1\_a[11:8];*

*3'd3: seg\_a = out1\_a[15:12];*

*3'd4: seg\_a = out1\_a[19:16];*

*3'd5: seg\_a = out1\_a[23:20];*

*3'd6: seg\_a = out1\_a[27:24];*

*3'd7: seg\_a = out1\_a[31:28];*

*default: ;*

*endcase*

*end*

*endmodule*

cpu.v

*module cpu (*

*input clk,*

*input rst,*

*//IO\_BUS*

*output [7:0] io\_addr, //led和seg的地址*

*output [31:0] io\_dout, //输出led和seg的数据*

*output io\_we, //输出led和seg数据时的使能信号*

*input [31:0] io\_din, //来自sw的输入数据*

*//Debug\_BUS*

*input [7:0] m\_rf\_addr, //存储器(MEM)或寄存器堆(RF)的调试读口地址*

*output [31:0] rf\_data, //从RF读取的数据*

*output [31:0] m\_data, //从MEM读取的数据*

*output [31:0] pc //PC的内容*

*);*

*wire [31:0] PC\_in, PC\_out, PC\_plus\_4\_new,PC\_plus\_4, PC\_NotJump, WriteData, MemData, MDR; // PC\_plus\_4=PC+4； PC\_out 当前地址 PC\_in 下一个指令*

*wire zf, RegWrite, ALUSrc, PC\_en, RegDst, Jump, Branch, MemRead, MemtoReg, MemWrite,ALU\_zero;*

*wire [31:0] ReadData1\_Data\_Memory,ShiftLeft\_Output,ShiftLeft\_plus\_pc,MUX\_3\_1\_out\_DataMemory,MUXout\_DataMemory;*

*wire [31:0] INS,ImmgenControl\_Output,JumpAddr, BranchAddr, ALU\_result, ReadData1, ReadData2, ALUsrcB, Imm,ReadData3; // INS 指令 wire [4:0] WriteReg; wire [2:0] ALUop;*

*wire[1:0]ALUop;*

*wire[2:0]ALUfunc;*

*wire jal,jalr;*

*wire Imm\_gen;*

*wire [1:0]RegSrc;*

*assign PC\_plus\_4\_new = pc + 32'd4;*

*// wire MemWrite;*

*// wire ALUSrc;*

*// wire RegWrite;*

*assign io\_we = MemWrite&&(~ALU\_result[10]);*

*pc PC(*

*.input\_data(PC\_in),*

*.en(1),*

*.rst(rst),*

*.clk(clk),*

*.output\_data(pc)*

*);*

*dist\_mem\_gen\_0 instruction\_memory(*

*.a(pc[9:2]),*

*.spo(INS)*

*);*

*dist\_mem\_gen\_1 Data\_Memory(*

*.a(ALU\_result[9:2]),*

*.d(ReadData2),*

*.dpra(m\_rf\_addr[7:0]), //second address*

*.dpo(m\_data[31:0]), //ReadData2*

*.clk(clk),*

*.we(MemWrite&&(~ALU\_result[10])),*

*.spo(ReadData1\_Data\_Memory) //ReadData1*

*);*

*assign io\_dout = ReadData2;*

*control\_unit control(*

*.instruction(INS[6:0]), // aka. op code*

*.jal(jal),*

*.jalr(jalr),*

*.Branch(Branch),*

*// .Imm\_gen(Imm\_gen),*

*.RegSrc(RegSrc),*

*.MemWrite(MemWrite),*

*.ALUSrc(ALUSrc),*

*.RegWrite(RegWrite),*

*.ALUop(ALUop)*

*);*

*register\_file Register(*

*.clk(clk),*

*.ra0(INS[19:15]),*

*.rd0(ReadData1),*

*.ra1(INS[24:20]),*

*.rd1(ReadData2),*

*.wa(INS[11:7]),*

*.we(RegWrite),*

*.wd(MUX\_3\_1\_out\_DataMemory),//here comes the bug 15:52*

*.ra2(m\_rf\_addr[7:0]),*

*.rd\_debug(rf\_data[31:0])*

*);*

*mux2\_1 MUX\_Register(*

*.a(ReadData2),*

*.b(ImmgenControl\_Output),*

*.sel(ALUSrc),*

*.o(ALUsrcB)*

*);*

*alu alu(*

*.a(ReadData1),*

*.b(ALUsrcB),*

*.f(ALUfunc),*

*.y\_r(ALU\_result),*

*.z\_r(ALU\_zero)*

*);*

*ALUcontrol ALUcontrol(*

*.funct3(INS[14:12]),*

*.ALUop(ALUop),*

*.ALUout(ALUfunc)*

*);*

*ImmGenControl ImmGenControl(*

*.instruction(INS),*

*.immediate(ImmgenControl\_Output)*

*);*

*mux2\_1 MUX\_2\_1\_DataMemory(*

*.a(ReadData1\_Data\_Memory),*

*.b(io\_din[31:0]),*

*.sel(ALU\_result[10]),*

*.o(MUXout\_DataMemory)*

*);*

*assign io\_addr[7:0] = ALU\_result[7:0];*

*mux4\_1 MUX\_4\_1\_DataMemory(*

*.d(ShiftLeft\_plus\_pc),//3*

*.c(PC\_plus\_4), // 2?*

*.b(MUXout\_DataMemory), //1?*

*.a(ALU\_result), // 0?*

*.sel(RegSrc),*

*.o(MUX\_3\_1\_out\_DataMemory)*

*); // may cause a bug here*

*AddModule\_32 pc\_adder(*

*.a(32'd4),*

*.b(pc), //两操作数*

*.y\_r(PC\_plus\_4) //运算结果*

*);*

*ShiftLeft1 ShiftLeft1(*

*.immediate(ImmgenControl\_Output),*

*.immediate\_out(ShiftLeft\_Output)*

*);*

*AddModule\_32 ShiftLeft\_adder(*

*.clk(clk),*

*.a(ShiftLeft\_Output),*

*.b(pc), //两操作数*

*.y\_r(ShiftLeft\_plus\_pc) //运算结果*

*);*

*mux4\_1 MUX\_3\_1\_ShiftLeft(*

*.a(PC\_plus\_4\_new),*

*.b(ShiftLeft\_plus\_pc),*

*.c(ALU\_result),*

*.d(32'b0),*

*.sel({jalr,{jal||(Branch && ALU\_zero)}}), // notice the logic computing here may cause a bug. remember to check out*

*.o(PC\_in)*

*);*

*endmodule*

control\_unit.v register\_file.v

*module control\_unit(*

*input [6:0] instruction, // aka. op code*

*// output reg jal, Branch, reg Imm\_gen, reg [1:0]RegSrc, reg MemWrite, reg ALUSrc,reg RegWrite,*

*// remove imm\_gen*

*output reg jalr,jal, Branch, reg [1:0]RegSrc, reg MemWrite, reg ALUSrc,reg RegWrite,*

*output reg [1:0] ALUop*

*);*

*always@(\*)*

*case (instruction)*

*7'b0110011://add sub*

*begin*

*jalr=0;*

*jal=0;*

*Branch=0;*

*RegSrc=2'b00;*

*ALUop=2'b00;*

*MemWrite=0;*

*ALUSrc=0;*

*RegWrite=1;*

*end*

*7'b0010011://addi*

*begin*

*jalr=0;*

*jal=0;*

*Branch=0;*

*RegSrc=2'b00;*

*ALUop=2'b00;*

*MemWrite=0;*

*ALUSrc=1;*

*RegWrite=1;*

*end*

*7'b1101111://jal*

*begin*

*jalr=0;*

*jal=1;*

*Branch=0;*

*RegSrc=2'b10;*

*ALUop=2'b00;*

*MemWrite=0;*

*ALUSrc=0;*

*RegWrite=1;*

*end*

*7'b1100011://beq blt*

*begin*

*jalr=0;*

*jal=0;*

*Branch=1;*

*RegSrc=2'b00;*

*ALUop=2'b01;*

*MemWrite=0;*

*ALUSrc=0;*

*RegWrite=0;*

*end*

*7'b0000011://lw*

*begin*

*jalr=0;*

*jal=0;*

*Branch=0;*

*RegSrc=2'b01;*

*ALUop=2'b00;*

*MemWrite=0;*

*ALUSrc=1;*

*RegWrite=1;*

*end*

*7'b0100011://sw*

*begin*

*jalr=0;*

*jal=0;*

*Branch=0;*

*RegSrc=2'b00;*

*ALUop=2'b00;*

*MemWrite=1;*

*ALUSrc=1;*

*RegWrite=0;*

*end*

*7'b0010111://auipc*

*begin*

*jalr=0;*

*jal=0;*

*Branch=0;*

*RegSrc=2'b11;*

*ALUop=2'b00;*

*MemWrite=0;*

*ALUSrc=0;*

*RegWrite=1;*

*end*

*7'b1100111://jalr*

*begin*

*jalr=1;*

*jal=0;*

*Branch=0;*

*RegSrc=2'b10;*

*ALUop=2'b00;*

*MemWrite=0;*

*ALUSrc=1;*

*RegWrite=1;*

*end*

*default: ;*

*endcase*

*endmodule*

*module register\_file #(parameter WIDTH = 32)(*

*input clk,*

*input [4:0]ra0,*

*output[WIDTH-1:0]rd0,*

*input[4:0]ra1,*

*output[WIDTH-1:0]rd1,*

*input[4:0]wa,*

*input we,*

*input[WIDTH-1:0]wd,*

*input[7:0]ra2,// didn't connect anything*

*output[WIDTH-1:0]rd\_debug*

*);*

*// reg [3:0] regfile[0:7];*

*reg [WIDTH-1:0] regfile[31:0];*

*assign rd0 = regfile[ra0],*

*rd1 = regfile[ra1];*

*assign rd\_debug = regfile[ra2];*

*always @ (posedge clk) begin*

*if (we) begin*

*if(wa == 0)begin*

*regfile[wa]<=0;*

*end*

*else begin*

*regfile[wa] <= wd;*

*end*

*end*

*end*

*endmodule*