计算机组成原理 实验报告

姓名：廖洲洲　学号：PB17081504　实验日期：2019-5-31

一、实验题目：

Lab06 综合实验

二、实验目的：

利用 MIPS-CPU、存储器和外设完成一个简单的计算机应用设计

三、实验平台：

Vivado

四、实验过程：

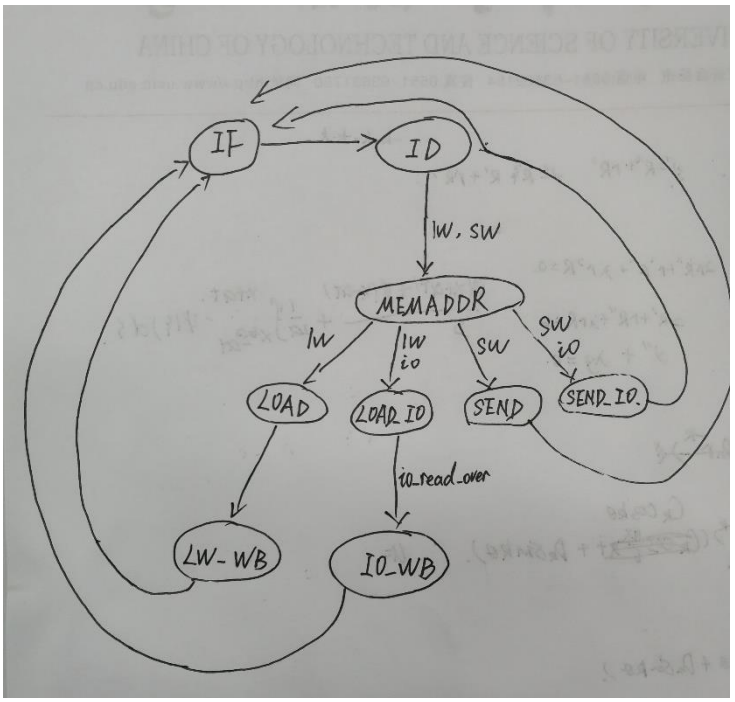
- 利用 lab5 实现的 MIPS-CPU 中的 load/store 指令对开发板上的外设（发光二极管、数码管、按键）进行读写操作
- 将外设和地址进行统一编址，将 32 位内存地址的后两位划给外设，即地址为 0000_0000 到 ffff_ff00 依旧作为普通内存地址，使用 Load 和 Store 指令对这些地址进行操作时是对内存进行读写。而地址 ffff_ff01 到 ffff_ffff 作为外设地址，使用 Load 和 Store 指令对这些地址进行操作时是对外设进行读写。
- 在这里，我们设置

外设	地址
LED(发光二极管)	ffff_ff10
SW(按键)	ffff_ff20
BTN(按键)	ffff_ff30
SEG(发光二极管)	ffff_ff40

verilog 代码：

```
parameter [31:0] LED=32'hffff_ff10,
                 SW=32'hffff_ff20,
                 BTN=32'hffff_ff30,
                 SEG=32'hffff_ff40;
```

- 增加 cpu 运行状态，在原来的基础上增加 3 个 IO 状态，分别为 LOAD_IO、IO_WB 和 SEND_IO,在执行 Load 和 Store 指令时，如果发现计算出的目标地址为外设地址，那么就会转入相应 IO 状态。修改部分的状态机如下图：



其中当 cpu 状态位于 LOAD_IO 时，只有当接收到外设读取完成 io_read_over 信号时，cpu 状态才会往下走，否则一直位于此状态。

修改部分 verilog 代码：

```
1.    MEMADRESS:begin //lw or sw 计算内存地址
            if(Op == 6'b100011)// lw3
            begin
                if(Instruction_15_0==16'hff20                          ||
Instruction_15_0==16'hff30) begin  //访问外设 sw 和 btn
                      next_state = LOAD_IO;
                      end
                else begin
                       next_state = LOAD;  //lw 下次进入加载阶段
                       end
            end
            else begin  //sw
                 if(Instruction_15_0==16'hff10                         ||
Instruction_15_0==16'hff40) begin //访问外设 led seg
                      next_state = SEND_IO;
                      end
                else begin
```

```
                    next_state = SEND;  //sw3 进入写阶段
                end
            end
        end
    2.  LOAD_IO:begin
            if(io_read_over)
                next_state = IO_WB;
            else
                next_state = LOAD_IO;
        end
        IO_WB:begin
            next_state = IF;
        end
    3.  SEND_IO:begin
            next_state = IF;
        end
```

- 增加两个 cpu 控制信号 io_read 和 io_write，io_read 信号始终为 1，即 io 始终是可读的，但是其读出的数据不一定会被写入寄存器堆，只有当 cpu 状态处于 IO_WB 时，寄存器堆写数据端口前的多选器才会选择 IO 送过来的数据。io_write 只有在 SEND_IO 状态有效，此时能对 IO 设备进行写操作。

  verilog 代码：

```
io_read =1;
if(next_state == SEND_IO)
    io_write = 1;
else begin
    io_write = 0;
end
if(next_state == LW_WB)
     MemtoReg = 1;
else if(next_state == IO_WB)
    MemtoReg = 2;
```

```
else begin
     MemtoReg = 0;
end
```

- 增加 IO 控制模块，当 io 读写使能送到 IO 控制模块时，IO 控制模块即可以根据从 cpu 内送出来的地址对相应的 io 进行读写操作。

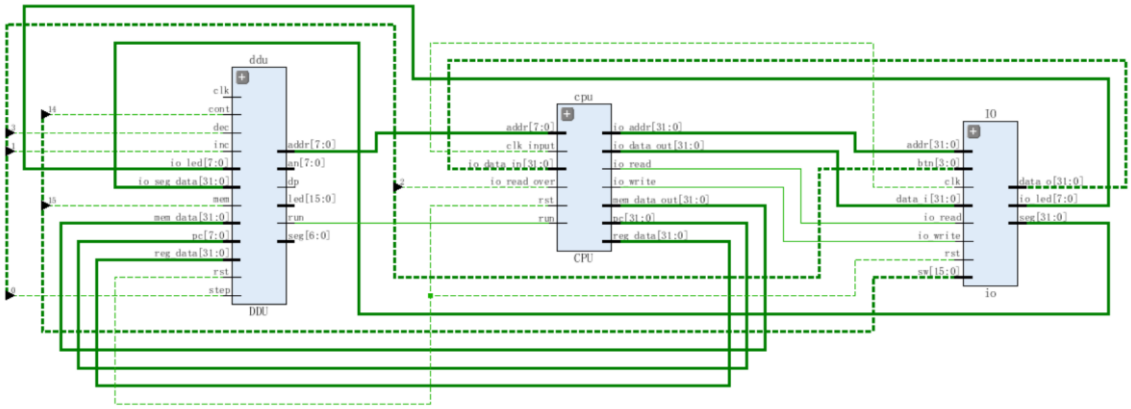
  Verilog 代码：

```
    always@(posedge clk or posedge rst)begin
        if(rst) begin
            io_led<=0;
            seg<=0;
            end
        else if(io_write) begin
           if(addr == LED)begin //led
               io_led <= data_i[7:0];//data_i[15:0];

           end
           else if(addr == SEG)begin
               seg <= data_i;
           end
           else ;

           end
       else
             ;
        ;
    end

    always @ (posedge clk or posedge rst) begin
        if(rst)
           data_o <= 0;
        else if(io_read) begin
             if(addr == SW)//sw
```

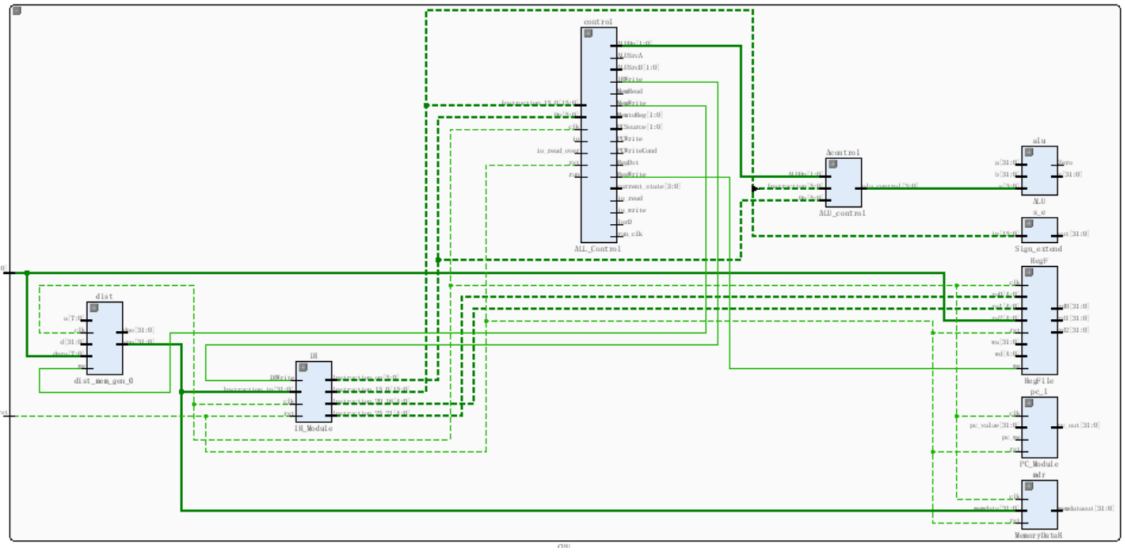
```
                data_o <= {18'b0,sw[13:0]};
            else if(addr == BTN)
                data_o <= {28'b0,btn[3:0]};
            else
                data_o <= 32'hffff_ffff;
        end
    else begin
            data_o <= 32'hafaf_afaf;
        end
endmodule
```

- 原理图

1. cpu 与 io 和 ddu 模块的连接图



2. cpu 主要模块内部连接图



五、应用

- cpu 发出外设输入指令从按键中依次输入两个数，然后 cpu 以这两个数为基础计算出后面的斐波那契数列，并且通过外设输出指令将它们打印出来。
- 汇编代码

```
j _chushihua

_chushihua:
addi $t0,$zero,35
addi $t1,$zero,1
lw $t2,0xff40($zero)
lw $t3,0xff40($zero)
j _zhixin

_zhixin:
beq $t0,$t1,_jieshu
addi $t1,$t1,1
add $t4,$t2,$t3
sw $t4,0xff40($zero)
sw $t4,0xff40($zero)
addi $t2,$t3,0
addi $t3,$t4,0
j _zhixin

_jieshu:
addi $t1,$t1,0
j _jieshu
```

- 机器码

```
memory_initialization_radix  = 16;
memory_initialization_vector =
08000001
20080023
```

20090001

8c0aff20

8c0bff20

08000006

11090005

21290001

014b6020

ac0cff40

ac0cff10

216a0000

218b0000

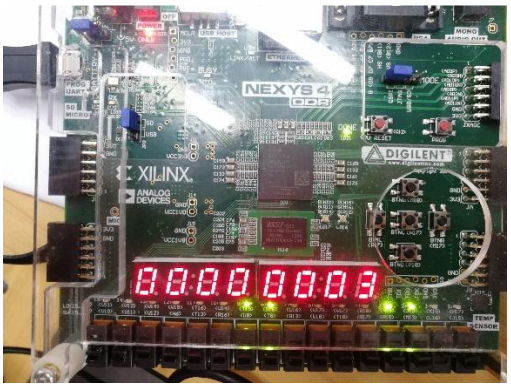
08000006

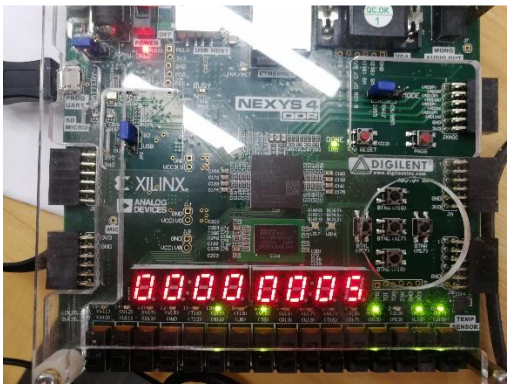
21290000

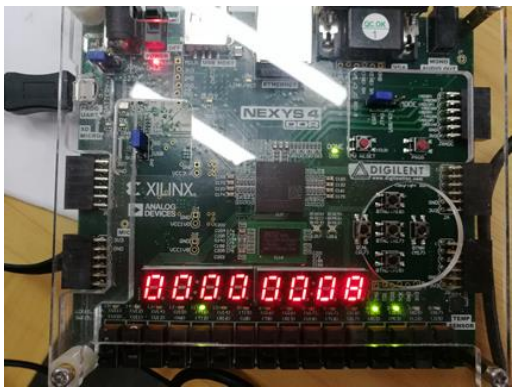
0800000c

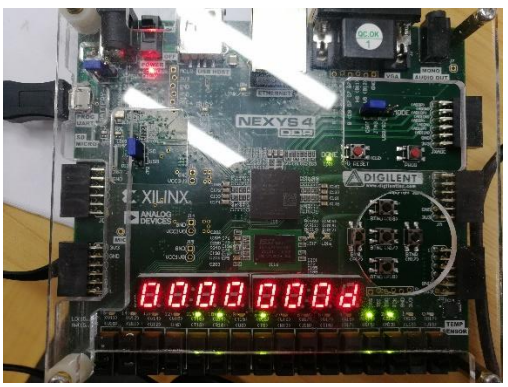
六、实验结果：

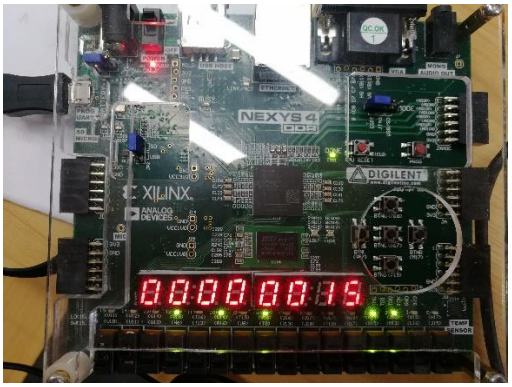
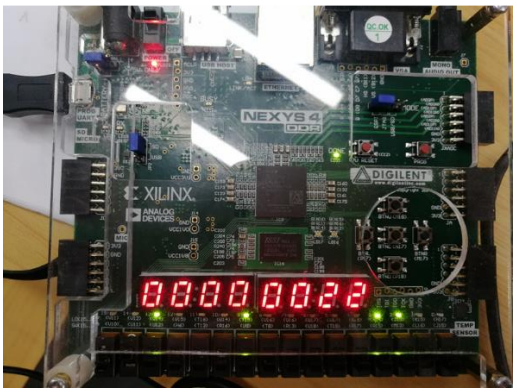
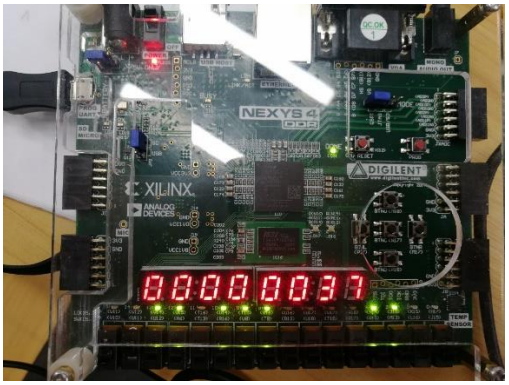
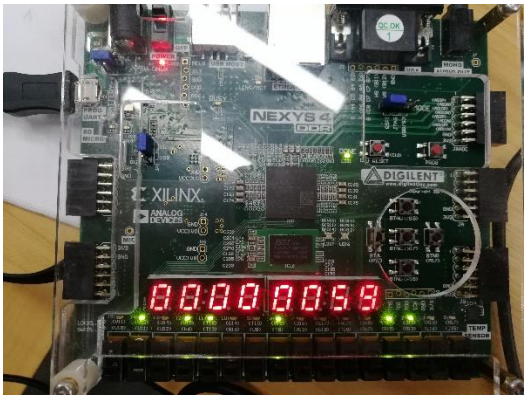
输入 1 和 2 两个数，输出的斐波那契数列









七、心得体会：

- 通过这次实验我利用 MISP_CPU 对外设进行了简单的读写，实现了一个求斐波那契数列的简单应用。
- 这次实验中由于没有写仿真文件进行调试，所以调试得特别麻烦，每次出现 bug 修改代码后都需要重新生成比特流，因此浪费了许多时间。

八、代码：

```
module MainCPU(
input [15:0] sw,
input [3:0] btn,
input clk_in,
input rst,
output [15:0]led,
output [7:0]an,
output [6:0]seg,
output dp
```

```
    );
wire cont=sw[14];
wire mem=sw[15];
wire step=btn[0];
wire inc=btn[1];
wire dec=btn[3];
wire io_read_over=btn[2];

wire clk_50mhz;
wire [7:0]pc;
wire [31:0]mem_data;
wire [31:0]reg_data;
wire run;
wire [7:0]addr;
wire locked;
wire clk_10hz;
clk_wiz_0
c(.clk_out1(clk_50mhz),.reset(rst),.locked(locked),.clk_in1(clk_in));
cout_10hz c2(clk_50mhz,rst,clk_10hz);

wire  [31:0] io_data_in;
wire  [31:0] io_data_out;
reg  [31:0] io_data_out_test;
wire [31:0] io_addr;
wire io_read;
wire io_write;
wire [7:0] io_led;
wire [31:0] io_seg_data;
wire [31:0] io_data_outReg;
always@(*) begin
    io_data_out_test = io_data_out;
    end
//CPU cpu(run,addr,clk_10hz,rst,pc,mem_data,reg_data,
```

```
//
io_data_outReg,io_data_in,io_addr,io_read,io_write,io_read_over);
CPU cpu(run,addr,clk_10hz,rst,pc,mem_data,reg_data,

io_data_out_test,io_data_in,io_addr,io_read,io_write,io_read_over);
io IO(rst,clk_10hz,io_read,io_write,io_addr,io_data_in,
      io_data_out,sw,btn,io_led,io_seg_data
           );

IoDataR io_data_reg(
           io_data_out,
           rst,
           clk_10hz,
           io_data_outReg
                );

DDU ddu(cont,step,mem,inc,dec,pc,mem_data,reg_data,
        clk_50mhz,rst,run,addr,led,an,seg,dp,
        io_led,io_seg_data);
endmodule



module cout_10hz(
    input clk,
    input rst_n,
    output reg Q
    );
    reg     [24:0] cnt;
    always@(posedge clk or negedge rst_n)
    begin
        if(rst_n)
            cnt    <= 25'd0;
        else if(cnt >= 25'd5000000)
```

```
            cnt    <= 25'd0;
        else
            cnt    <= cnt + 25'd1;
    Q =  (cnt >= 25'd2500000) ? 1'b1 : 1'b0;
    end
endmodule



module cout_5hz(
    input clk,
    input rst_n,
    output reg Q
    );
    reg     [24:0] cnt;
    always@(posedge clk or negedge rst_n)
    begin
        if(rst_n)
            cnt    <= 25'd0;
        else if(cnt >= 25'd10000000)
            cnt    <= 25'd0;
        else
            cnt    <= cnt + 25'd1;
    Q =  (cnt >= 25'd5000000) ? 1'b1 : 1'b0;
    end
endmodule

module io(
    input wire rst,
    input wire clk,
      input wire io_read,
      input wire io_write,
      input wire[31:0] addr,
      input wire[31:0] data_i,
```

```
  output reg[31:0] data_o,
  input wire[15:0] sw,
  input wire[3:0] btn,
  output reg[7:0] io_led,
  output reg[31:0] seg
);
parameter [31:0] LED=32'hffff_ff10,
                 SW=32'hffff_ff20,
                 BTN=32'hffff_ff30,
                 SEG=32'hffff_ff40;
always@(posedge clk or posedge rst)begin
     if(rst) begin
         io_led<=0;
         seg<=0;
         end
     else if(io_write) begin
        if(addr == LED)begin //led
            io_led <= data_i[7:0];//data_i[15:0];

        end
        else if(addr == SEG)begin
            seg <= data_i;
        end
        else ;

        end
    else
           ;
     ;
end

always @ (posedge clk or posedge rst) begin
     if(rst)
```

```
            data_o <= 0;
         else if(io_read) begin
                if(addr == SW)//sw
                    data_o <= {18'b0,sw[13:0]};
                else if(addr == BTN)
                    data_o <= {28'b0,btn[3:0]};
                else
                    data_o <= 32'hffff_ffff;
            end
          else begin
                    data_o <= 32'hafaf_afaf;
            end
        end
endmodule

module IoDataR(
input [31:0]iodata,
input rst,
input clk,
output [31:0]iodataout
     );
reg [31:0]memd;
assign iodataout = memd;
always @(posedge clk or posedge rst) begin
     if(rst)
            memd <= 0;
     else begin
            memd <= iodata;
     end
end
endmodule

module DDU(
```

```
cont,step,mem,inc,dec,pc,mem_data,reg_data,clk,rst,run,addr,led,an,seg,dp
,
io_led,io_seg_data
    );
input cont;
input step;//always@step
input mem;
input inc;
input dec;
input [7:0]pc;
input [31:0]mem_data;
input [31:0]reg_data;
input clk;
input rst;
output run;
output [7:0]addr;
output [15:0]led;
output [7:0]an;
output [6:0]seg;
output dp;

input [7:0] io_led;
input [31:0] io_seg_data;

wire [31:0]data;
wire clk_10hz;
//assign data = mem == 1?mem_data:reg_data;
assign data = io_seg_data;
RUN r(cont,step,clk,rst,run);
cout_5hz c1(clk,rst,clk_10hz);
Address a(inc,dec,clk_10hz,rst,addr);
//Display d(pc,data,addr,clk,rst,led,an,seg,dp);
Display d(pc,data,io_led,clk,rst,led,an,seg,dp);
```

```
endmodule


//module about run
module RUN(
input cont,
input step,
input clk,
input rst,
output run
     );
wire clk_out;
//run = 1/run = 0;
cout_1ms c3(clk,rst,clk_out);  //1000HZ prevent the cycle too short
reg real_run;
reg flag;
always @(posedge clk_out or posedge rst) begin
     if (rst) begin
            // reset
            real_run <= 0;
            flag <= 0;
     end
     else begin
            if(step == 1 && flag == 0)
            begin
                  real_run <= 1;
                  flag <= 1;
            end
            else if(step == 0 && flag == 1)
            begin
                  flag <= 0;
                  real_run <= 0;
            end
            else begin
```

```
                real_run <= 0;
            end
        end
end
assign run = real_run | cont;

endmodule

//address
module Address(
input inc,
input dec,
input clk,
input rst,
output reg [7:0]addr
    );

always@(posedge clk or posedge rst)
begin
    if(rst == 1)
            addr = 8'b0;
    else begin
            if(inc == 1)
                addr = addr + 1;
            else if(dec == 1)begin
                addr = addr - 1;
            end
            else begin
                addr = addr;
            end
    end
end
```

```
endmodule



module Display(
input [7:0]pc,
input [31:0]data,
input [7:0]io_led,
input clk,
input rst,
output [15:0]led,
output reg[7:0]an,
output [6:0]seg,
output dp
     );
wire clk_1ms;
cout_1ms t(clk,rst,clk_1ms);
assign led[15:8] = io_led;
assign led[7:0] = pc;
reg [3:0]in;
reg [2:0]count;
Change change(in,seg,dp);
always @(posedge clk_1ms or posedge rst) begin
     if (rst) begin
           // reset
           count <= 0;
           an <= 8'b11111111;
     end
     else begin
         an[0] = 1;
         an[1] = 1;
         an[2] = 1;
         an[3] = 1;
         an[4] = 1;
```

```
            an[5] = 1;
            an[6] = 1;
            an[7] = 1;
               an[count] = 0;
               count = count + 1;
      end
end

always@(count)
begin
      if(rst)
               in = 0;
      else begin
               case(count)  //set in for the pre cycle
               3'b000:begin
                        in = data[31:28];
               end
               3'b111:begin
                        in = data[27:24];
               end
               3'b110:begin
                        in = data[23:20];
               end
               3'b101:begin
                        in = data[19:16];
               end
               3'b100:begin
                        in = data[15:12];
               end
               3'b011:begin
                        in = data[11:8];
               end
               3'b010:begin
```

```
                    in = data[7:4];
            end
            3'b001:begin
                    in = data[3:0];
            end
            endcase
      end
end



endmodule

module cout_1ms(  //the time of one cycle doesn't be confirmed,so it may need to change to follow the time of one period
    input clk,
    input rst_n,
    output reg Q
    );
    reg     [24:0] cnt;
    always@(posedge clk or posedge rst_n)
    begin
        if(rst_n)
            cnt    <= 25'd0;
        else if(cnt >= 25'd50000)
            cnt    <= 25'd0;
        else
            cnt    <= cnt + 25'd1;
    Q =  (cnt >= 25'd25000) ? 1'b1 : 1'b0;
    end
endmodule

module Change(
input [3:0]in,
```

```
output reg[6:0]seg,
output reg dp
      );

always@(*)
begin
      case(in)
      4'b0:{dp,seg} = 8'hC0;
      4'b1:{dp,seg} = 8'hF9;
      4'b10:{dp,seg} = 8'hA4;
      4'b11:{dp,seg} = 8'hB0;
      4'b100:{dp,seg} = 8'h99;
      4'b101:{dp,seg} = 8'h92;
      4'b110:{dp,seg} = 8'h82;
      4'b111:{dp,seg} = 8'hF8;
      4'b1000:{dp,seg} = 8'h80;
      4'b1001:{dp,seg} = 8'h90;
      4'b1010:{dp,seg} = 8'h88;
      4'b1011:{dp,seg} = 8'h83;
      4'b1100:{dp,seg} = 8'hC6;
      4'b1101:{dp,seg} = 8'hA1;
      4'b1110:{dp,seg} = 8'h84;
      4'b1111:{dp,seg} = 8'h8E;
      endcase
end


endmodule

module CPU(
input run,
input [7:0]addr,
input clk_input,
```

```
input rst,
output [31:0] pc,
output [31:0] mem_data_out,
output [31:0] reg_data,
input  [31:0] io_data_in,
output [31:0] io_data_out,
output [31:0] io_addr,
output io_read,
output io_write,
input io_read_over
/*output PCWriteCond,
output Zero,
output PCWrite,
output lorD,
output MemWrite,
output IRWrite,
output RegDst,
output RegWrite,
output ALUSrcA,
output MemtoReg,
output current_state,
output [5:0]Instruction_op,
output [1:0]PCSource,
output [1:0]ALUSrcB*/
    );
//declaration
wire
[31:0]pc_src,pc_out,B_out,Mem_data,Write_data_regf,RD0,RD1,alu_o,A_out,A_
out_1,B_out_1,ALU_1,memdataout,I,I_shift,ALU_2,alu_result;
wire [31:0]Jaddress;
wire [31:0]ALU_out,address;
wire
PCWriteCond,Zero,PCWrite,lorD,MemWrite,IRWrite,RegDst,RegWrite,ALUSrcA;
```

```
wire [1:0] MemtoReg;
wire clk,MemRead;
wire [3:0]current_state;
wire [5:0]Instruction_op;
wire [4:0]Instruction_25_21,Instruction_20_16,Write_register;
wire [15:0]Instruction_15_0;
wire [3:0] alu_c;
wire [1:0] ALUOp;
wire [1:0] ALUSrcB;
wire [1:0]PCSource;
wire run_clk;
wire io_read,io_write;
wire io;
//
wire pc_we;
assign clk = run_clk & clk_input;
assign  pc = pc_out;
assign io_addr = ALU_out;
assign io= (ALU_out <= 32'hffff_ff00) ? 0:1;  //io使能的判断
assign io_data_out = B_out;
PC_Module pc_1(pc_src,pc_we,rst,clk,pc_out);
PCWE pc_2(Instruction_op,PCWrite,PCWriteCond,Zero,pc_we);
MUX_2_32 MUX1(pc_out,(ALU_out),lorD,address);

dist_mem_gen_0
dist(address[7:0],B_out,addr,clk,MemWrite,Mem_data,mem_data_out);//read

IR_Module
iR(Mem_data,IRWrite,clk,rst,Instruction_op,Instruction_25_21,Instruction_
20_16,Instruction_15_0);
MUX_2_5
MUX2(Instruction_20_16,Instruction_15_0[15:11],RegDst,Write_register);
```

```
RegFile
RegF(Instruction_25_21,Instruction_20_16,addr[4:0],Write_data_regf,Write_
register,RegWrite,rst,clk,RD0,RD1,reg_data);
ALU_A a(RD0,clk,rst,A_out);
ALU_B b(RD1,clk,rst,B_out);
//DataLate A(A_out_1,clk,A_out);
//DataLate B(B_out_1,clk,B_out);
MUX_2_32 MUX3(pc_out,A_out,ALUSrcA,ALU_1);
MemoryDataR mdr(Mem_data,rst,clk,memdataout);
//*****************
MUX_3_32 MUX4(ALU_out,memdataout,io_data_in,MemtoReg,Write_data_regf);
//*************
Sign_extend s_e(Instruction_15_0,I);
//Shift_left_2 s_l_1(I,I_shift);
assign I_shift = I;
//MUX_4_32 MUX5(B_out,4,I,I_shift,ALUSrcB,ALU_2);
MUX_4_32
MUX5(B_out,I,I_shift,ALUSrcB,ALU_2);//1?0?6?0?5?0?6?0?2° ü?0?2?0?1?0?8?0?
3?0?8?0?7?0?1?0?3?0?9?0?9
ALU alu(ALU_1,ALU_2,alu_c,alu_result,Zero);
ALU_OUT aluout1(alu_result,clk,rst,ALU_out);
//DataLate ALUOut(alu_o,clk,ALU_out);
ALU_control Acontrol(Instruction_op,Instruction_15_0[5:0],ALUOp,alu_c);
//Shift_left_2
s_l_2({6'b000000,Instruction_25_21,Instruction_20_16,Instruction_15_0},Ja
ddress);
assign Jaddress =
{6'b000000,Instruction_25_21,Instruction_20_16,Instruction_15_0};
MUX_3_32 MUX6(alu_result,ALU_out,Jaddress,PCSource,pc_src);
ALL_Control
control(Instruction_op,clk,rst,PCWriteCond,PCWrite,IorD,MemRead,MemWrite,
MemtoReg,
```

```
IRWrite,PCSource,ALUOp,ALUSrcB,ALUSrcA,RegWrite,RegDst,current_state,run,
run_clk,
                    io,io_read,io_write,Instruction_15_0,io_read_over);
endmodule




/*module MUX_2_8(
input [7:0]value_0,
input [7:0]value_1,
input choose,
output reg[7:0]value
    );
always@(*)
begin
    case(choose)
    1'b1:
            value = value_1;
    1'b0:
            value = value_0;
    endcase
end
endmodule
*/
module PC_Module(
input [31:0]pc_value,
input pc_we,
input rst,
input clk,
output reg[31:0]pc_out
    );
```

```
always @(posedge clk or posedge rst) begin
      if (rst) begin
            // reset
            pc_out = 0;
      end
      else begin
            if(pc_we==1)
                  pc_out = pc_value;
            else begin
                  pc_out = pc_out;
             end
      end
end
endmodule



module IR_Module(
input [31:0]Instruction_in,
input IRWrite,
input clk,
input rst,
output [5:0]Instruction_op,
output [4:0]Instruction_25_21,
output [4:0]Instruction_20_16,
output [15:0]Instruction_15_0
);
reg [31:0]Instruction;
assign Instruction_op = Instruction[31:26];
assign Instruction_25_21 = Instruction[25:21];
assign Instruction_20_16 = Instruction[20:16];
assign Instruction_15_0 = Instruction[15:0];
always @(negedge clk or posedge rst) begin//decrease
      if(rst)
```

```
            Instruction <= 0;
      else
      begin
            if(IRWrite==1)
                  Instruction <= Instruction_in;
            else begin
                  Instruction <= Instruction;
            end
      end
end
endmodule

module RegFile(ra0,ra1,ra2,wa,wd,we,rst,clk,rd0,rd1,rd2
    );
parameter ADDR = 5;
parameter NUM = 32;
parameter SIZE = 32;
input [ADDR-1:0]ra0;
input [ADDR-1:0]ra1;
input [ADDR-1:0]ra2;
input [SIZE-1:0]wa;
input [ADDR-1:0]wd;
input we;
input rst;
input clk;
output [SIZE-1:0]rd0;
output [SIZE-1:0]rd1;
output [SIZE-1:0]rd2;
//output [SIZE-1:0]rd3;
reg [SIZE-1:0]register[0:NUM-1];
integer i;

always @(posedge clk or posedge rst) begin
```

```
	if (rst) begin
		// reset
		for(i = 0;i<NUM;i=i+1)
			register[i] <= 0;  //初始化操作
	end
	else if (!rst) begin
		if(we)begin
			if(wd!=0)
				register[wd] <= wa;  //写操作
		end
	end
end

assign rd0 = register[ra0];  //读操作
assign rd1 = register[ra1];
assign rd2 = register[ra2];
//assign rd3 = 0;
endmodule



module MemoryDataR(
input [31:0]memdata,
input rst,
input clk,
output [31:0]memdataout
	);
reg [31:0]memd;
assign memdataout = memd;
always @(posedge clk or posedge rst) begin
	if(rst)
		memd <= 0;
	else begin
		memd <= memdata;
```

```
	end
end
endmodule

module Sign_extend(
input [15:0]in,
output [31:0]out
	);
assign out = in[15]==0?{16'h0000,in}:{16'hffff,in};
endmodule



module ALU_A(
input [31:0]in,
input clk,
input rst,
output reg[31:0]out
	);
always @(posedge clk or posedge rst) begin
	if(rst == 1)
		out = 0;
	else begin
		out = in;
	end
end
endmodule

module ALU_B(
input [31:0]in,
input clk,
input rst,
output reg[31:0]out
	);
```

```
always @(posedge clk or posedge rst) begin
      if(rst == 1)
            out = 0;
      else begin
            out = in;
      end
end
endmodule

module ALU(
    input [31:0]a,
    input [31:0]b,
    input [3:0]s,
    output reg [31:0]y,
    output Zero

    );
      assign Zero = (y==0)?1:0;  //判0
    reg [32:0] temp;
    always @(a,b,s)
    begin
    case(s)
        4'b0000:   //&
        begin
            y<=a&b;
        end
        4'b0001:    //|
        begin
            y<=a|b;
        end
        4'b0010:  //xor
        begin
```

```
    y<=a^b;
end
4'b0011:    //nor
begin
        y<=~(a|b);
end
4'b0100:    //add
begin
    /*temp={1'b0,a}+{1'b0,b};
    y=temp[31:0];*/
    y <= a+b;
end
4'b0101:    //sub
begin
    /*temp={1'b0,a}-{1'b0,b};
    y=temp[31:0];*/
    y <= a-b;
end
4'b0110:    //slt
begin
    if(a[31]==1&&b[31]==1)
            if(a[30:0]>b[30:0])
                    y <= 1;
            else begin
                    y <= 0;
            end
    else if(a[31]==1&&b[31]==0)
    begin
            y <= 1;
    end
    else if(a[31]==0&&b[31]==1)
    begin
            y <= 0;
```

```
            end
            else begin
                if(a[30:0]<b[30:0])
                    y <= 1;
                else begin
                    y <= 0;
                end
            end
        end
        default:begin
            y <= 0;
        end
    endcase
    end
endmodule

module ALU_OUT(
input [31:0]ALU_in,
input clk,
input rst,
output reg[31:0]ALU_out
    );
always @(posedge clk or posedge rst) begin
    if(rst == 1)
        ALU_out <= 0;
    else begin
        ALU_out <= ALU_in;
    end
end
endmodule

module MUX_2_32(
input [31:0]value_0,
```

```
input [31:0]value_1,
input choose,
output reg[31:0]value
     );
always@(*)
begin
     case(choose)
     1'b1:
            value = value_1;
     1'b0:
            value = value_0;
     endcase
end
endmodule

module MUX_2_5(
input [4:0]value_0,
input [4:0]value_1,
input choose,
output reg[4:0]value
     );
always@(*)
begin
     case(choose)
     1'b1:
            value = value_1;
     1'b0:
            value = value_0;
     endcase
end
endmodule

module MUX_3_32(
```

```
input [31:0]value_0,
input [31:0]value_1,
input [31:0]value_2,
input [1:0]choose,
output reg[31:0]value
      );
always@(*)
begin
      case(choose)
      2'b10:
            value = value_2;
      2'b01:
            value = value_1;
      2'b00:
            value = value_0;
      default:begin end
      endcase
end
endmodule

module MUX_4_32(
input [31:0]value_0,
input [31:0]value_2,
input [31:0]value_3,
input [1:0]choose,
output reg[31:0]value
      );
always@(*)
begin
      case(choose)
      2'b11:
            value = value_3;
```

```
	2'b10:
		value = value_2;
	2'b01:
		value = 32'b001;
	2'b00:
		value = value_0;
	endcase
end
endmodule

//ALUOP: 00 01 10 11
module ALU_control(
input [5:0] Op,
input [5:0] Instruction,
input [1:0] ALUOp,
output reg[3:0]alu_control
	);
parameter [3:0]add = 4'b0100,
sub = 4'b0101,
and_ = 4'b0000,
or_ = 4'b0001,
xor_ = 4'b0010,
nor_ = 4'b0011,
slt = 4'b0110,
wrong = 4'b1000;
always @(*) begin
	case(ALUOp)
	2'b00:alu_control <= 4'b0100;//add
	2'b10:begin
		case(Instruction)//funct
		6'b100000://add
		begin
			alu_control <= 4'b0100;
```

```
        end
        6'b100010://sub
        begin
                alu_control <= 4'b0101;
        end
        6'b100100://and
        begin
                alu_control <= 4'b0000;
        end
        6'b100101://or
        begin
                alu_control <= 4'b0001;
        end
        6'b100110://xor
        begin
                alu_control <= 4'b0010;
        end
        6'b100111://nor
        begin
                alu_control <= 4'b0011;
        end
        6'b101010://slt
        begin
                alu_control <= 4'b0110;
        end
        default:begin
                alu_control <= 4'b1000;
        end
        endcase
end
2'b01:begin
        alu_control <= 4'b0101;
end
```

```
        2'b11:begin
                case(Op)
                6'b001000:
                begin
                        alu_control <= 4'b0100;
                end
                6'b001100:
                begin
                        alu_control <= 4'b0000;
                end
                6'b001101:
                begin
                        alu_control <= 4'b0001;
                end
                6'b001110:
                begin
                        alu_control <= 4'b0010;
                end
                6'b001010:
                begin
                        alu_control <= 4'b0110;
                end
                default:begin
                        alu_control <= 4'b0100;
                end
                endcase
        end
        endcase
end

endmodule

module PCWE(Op,pc_we,PCWriteCond,Zero,pcwe);
```

```
input [5:0]Op;
input pc_we;
input PCWriteCond;
input Zero;
output reg pcwe;

always@(*)
begin
	if(Op == 6'b000100)//BEQ
		pcwe <= pc_we|(PCWriteCond&Zero);
	else if(Op == 6'b000101)//bne
		pcwe <= pc_we|(PCWriteCond&(~Zero));
	else begin
		pcwe <= pc_we;
	end
end
endmodule

module ALL_Control(
Op,clk,rst,PCWriteCond,PCWrite,lorD,MemRead,MemWrite,MemtoReg,IRWrite,PCS
ource,ALUOp,ALUSrcB,ALUSrcA,RegWrite,RegDst,current_state,run,run_clk,
io,io_read,io_write,Instruction_15_0,io_read_over
	);
input io_read_over;
input [15:0] Instruction_15_0;
input clk;
input rst;
input [5:0]Op;
output reg PCWriteCond;
output reg PCWrite;
output reg lorD;
output reg MemRead;
output reg MemWrite;
```

```
output reg [1:0] MemtoReg;
output reg RegDst;
output reg RegWrite;
output reg IRWrite;
output reg [1:0]PCSource;
output reg[1:0]ALUOp;
output reg[1:0]ALUSrcB;
output reg ALUSrcA;
input run;
output reg run_clk;

input io;
output reg io_write,io_read;

parameter [3:0]IF = 4'b0000,
                          ID = 4'b0001,
                          MEMADRESS = 4'b0010,
                          LOAD = 4'b0011,
                          LW_WB = 4'b0100,
                          SEND = 4'b0101,
                          EXE = 4'b0110,
                          R_WB = 4'b0111,
                          BEQC = 4'b1000,
                          JMC = 4'b1001,
                          EXE_I = 4'b1010,
                          I_WB = 4'b1011,

                          LOAD_IO = 4'b1100,
                          SEND_IO = 4'b1101,
                          IO_WB = 4'b1110,

                          INVALID = 4'b1111;
```

```
output reg[3:0] current_state;
reg [3:0] next_state;

always @(posedge run or posedge clk or posedge rst) begin
      if(rst)
      begin
            run_clk <= 0;
      end
      else begin
            if(run == 1)
                  run_clk <= 1;
            else begin
                  if(next_state == IF)
                        run_clk <= 0;//一条指令运行完毕，又到了取指阶
段，则让时钟为0
            end
      end
end

always @(posedge clk or posedge rst) begin
      if (rst) begin
            // reset
            current_state <= INVALID;//IF state
      end
      else begin
            current_state <= next_state;
      end
end

always@(*) begin
      next_state = 4'b0000;
      case(current_state)
      INVALID:begin
```

```
            next_state = IF;
        end
        IF:begin
            next_state = ID;
        end
        ID:begin
            if(Op == 6'b100011||Op == 6'b101011)//lw or sw
            begin
                next_state = MEMADRESS;  //计算内存地址
            end
            else if(Op==6'b000000) begin  //R 型
                next_state = EXE;
            end
            else if(Op == 6'b000100||Op == 6'b000101) begin //beq and
bne
                next_state = BEQC;
            end
            else if(Op == 6'b000010) begin  //j 型 跳转
                next_state = JMC;
            end
            else if(Op[5:3] == 3'b001) begin  //I 型
                next_state = EXE_I;
            end
        end
        MEMADRESS:begin //lw or sw 计算内存地址
            if(Op == 6'b100011)// lw3
            begin
                if(Instruction_15_0==16'hff20 ||
Instruction_15_0==16'hff30) begin  //访问外设 sw 和 btn
                    next_state = LOAD_IO;
                    end
                else begin
                    next_state = LOAD;  //lw 下次进入加载阶段
```

```
                    end
            end
            else begin  //sw
                if(Instruction_15_0==16'hff10 ||
Instruction_15_0==16'hff40) begin //访问外设 led seg
                    next_state = SEND_IO;
                    end
                else begin
                      next_state = SEND;  //sw3 进入写阶段
                   end
            end
      end
      LOAD:begin
                next_state = LW_WB;  //lw4 进入写寄存器阶段
      end
      LOAD_IO:begin
         if(io_read_over)
               next_state = IO_WB;
          else
               next_state = LOAD_IO;
      end
      IO_WB:begin
         next_state = IF;
      end
      LW_WB:begin
            next_state = IF;   //lw5 结束,再次取指
      end
      SEND:begin     //sw4 写回，结束了，下一阶段取指
            next_state = IF;
      end
      SEND_IO:begin
         next_state = IF;
      end
```

```
        EXE:begin //R3 执行，下一阶段写回
                next_state = R_WB;
        end
        R_WB:begin //R4 R 结束，下一阶段取指
                next_state = IF;
        end
        BEQC:begin //beq3 bne3 结束，下一阶段取值
                next_state = IF;
        end
        JMC:begin   //JMC3 结束
                next_state = IF;
        end
        EXE_I:begin  //I3 i 型执行，下一阶段写回
                next_state = I_WB;
        end
        I_WB:begin  //I4 写回，结束
                next_state = IF;
        end
        default:begin
                next_state = INVALID;
        end
        endcase
end

always @(posedge clk) begin  //下一状态的控制信号使用的是上一个时钟中修改的
        //PCWriteCond
if(next_state == BEQC)
        PCWriteCond = 1;  //PCwrite 只有在 beq bne 中有效
else begin
        PCWriteCond = 0;
end
```

```
//PCWRITE  只在取值和跳转时有效
if(next_state == IF || next_state == JMC)
      PCWrite = 1;
else begin
      PCWrite = 0;
end

//lord 只有在 load 和 send 阶段有效，lord 为 1 说明是对数据存储器读写
if(next_state == LOAD ||next_state == SEND)
      lorD = 1;
else begin
      lorD = 0;
end
//MemRead 想读都可以读
MemRead = 1;
//io 想读都可以读 但是写入寄存器时要进行选择
io_read =1;
//memwrite 在 sw 的写回有效
if(next_state == SEND)
      MemWrite = 1;
else begin
      MemWrite = 0;
end
//IO_write 只在 sw 的对外设的写回有效
//******************
if(next_state == SEND_IO)
    io_write = 1;
else begin
    io_write = 0;
end

//memtoreg lw_wb 为 1，即内存中数据写入，0 则 alu 结果写入
if(next_state == LW_WB)
```

```
      MemtoReg = 1;
else if(next_state == IO_WB)
    MemtoReg = 2;
else begin
      MemtoReg = 0;
end
//regdst
if(next_state == R_WB)//IWB is zero
      RegDst = 1;
else begin
      RegDst = 0;
end

//RegWrite
if(next_state == LW_WB|| next_state == R_WB||next_state==I_WB
    || next_state == IO_WB  )
      RegWrite = 1;
else begin
      RegWrite = 0;
end

//IRWrite
if(next_state == IF)
      IRWrite = 1;
else begin
      IRWrite = 0;
end

//pcsource
if(next_state == IF)
      PCSource = 2'b00;
else if(next_state == BEQC)
      PCSource = 2'b01;
```

```
else if(next_state == JMC)
      PCSource = 2'b10;
else begin
      PCSource = 2'b11;
end

//ALUOP
if(next_state == IF || next_state == MEMADRESS || next_state == ID)
      ALUOp = 2'b00;
else if(next_state == EXE)begin
      ALUOp = 2'b10;
end
else if(next_state == BEQC)
      ALUOp = 2'b01;  //减
else if(next_state == EXE_I)begin
      ALUOp = 2'b11;
end
else begin
      ALUOp = 2'b00;
end

//ALUSrcB
if(next_state == MEMADRESS || next_state == EXE_I || next_state==LOAD_IO
|| next_state == IO_WB )
      ALUSrcB = 2'b10;
else if(next_state == IF)
      ALUSrcB = 2'b01;
else if(next_state == ID)
      ALUSrcB = 2'b11;
else begin
      ALUSrcB = 2'b00;
end
```

```
//ALUSRCA
if(next_state == ID||next_state == IF)
      ALUSrcA = 0;
else begin
      ALUSrcA = 1;
end
//MemRead==1


end
endmodule
```