```
module CPU_Unity(
    input clk_100mhz,
    input rst,
    input clk_sel,
    input start,
    input [3:0] sel,
    output [15:0] IR,
    output [7:0] an,
    output [6:0] seg
    );

    wire clk5mhz;
    wire locked;
    clk_wiz_0 clk_5mhz0(.clk_in1(clk_100mhz),.locked(locked),.clk_out1(clk5mhz));

    wire clk_10hz;
    clk_5mhz_10hz clk_5mhz_10hz0 (clk5mhz,locked,clk_10hz);
    reg clk;
    wire [255:0] rf_data;
    wire [7:0] PC;
    cpu_mem uut (
        .clk(clk),
        .rst(rst),
        .start(start),
        .rf_data(rf_data),
        .PC(PC),
        .IR(IR)
    );

    always @(clk_sel)
        begin
        if(clk_sel)
            clk=clk5mhz;
        else
            clk=clk_10hz;
        end

    wire [15:0] in;
    Display dispaly(clk5mhz,locked,PC,in,an,seg);

    mux_16_1 mxu(sel,rf_data,in);

endmodule
```

```
module clk_5mhz_10hz(
        input clk5mhz,
        input locked,
        output clk_10hz
        );

        reg Q;
        reg [22:0] count;
        assign clk_10hz=Q;

            initial
                begin
                    count = 0;
                    Q = 0;
                end

            always @(posedge clk5mhz or    negedge locked)
                        begin
                            if(~locked)
                                count <= 0;
                            else if(count==249999)
                                begin
                                count <= 0;
                                Q <= ~Q;
                                end
                            else
                                count <= count + 1;
                         end
endmodule


module Display(
        input clk5mhz,
        input locked,
        input [7:0] PC,
        input [15:0] in,
        output [7:0] an,
        output [6:0] seg
        );
  //     wire [7:0] PC;
  //     assign PC=8'b0011_1111;
        wire [3:0] Tenthousands;
        wire [3:0] Thousands;
        wire [3:0] Hundreds;
```

```
    wire [3:0] Tens;
    wire [3:0] Ones;//16 位

    wire [3:0] hundreds;
    wire [3:0] tens;
    wire [3:0] ones; //8 位

    BCD_16binary binary16_BCD (in,Tenthousands,Thousands,Hundreds,Tens,Ones);
    BCD_8binary binary8_BCD(PC,hundreds,tens,ones);


    reg [7:0] seg_sel;
    reg [3:0] seg_din;

    seg_ctrl        seg_ctrl(
        .x              (seg_din),
        .sel            (seg_sel),
        .an             (an),
        .seg            (seg)
            );

//      wire clk5mhz;
//      wire locked;
//      clk_wiz_0 U1(.clk_in1(clk_100mhz),.locked(locked),.clk_out1(clk5mhz));




     reg [22:0] cnt;
     always@(posedge clk5mhz or negedge locked)
     begin
         if(~locked)
             cnt <= 23'h0;
         else if(cnt<23'd5000000)
             cnt <= cnt + 1;
         else
             cnt <= 23'h0;
     end

     always@(posedge clk5mhz or negedge locked)
     begin
      if(~locked)
     begin
```

```
        seg_sel <= 8'b0;
        seg_din <= 4'b0;
    end
    else case(cnt[14:12])
        3'b000:
        begin
            seg_sel <= 8'b1111_1110;
            seg_din <= Ones;
        end
        3'b001:
        begin
            seg_sel <= 8'b1111_1101;
            seg_din <= Tens;
        end
        3'b010:
        begin
            seg_sel <= 8'b1111_1011;
            seg_din <= Hundreds;
        end
        3'b011:
        begin
            seg_sel <= 8'b1111_0111;
            seg_din <= Thousands;
        end
        3'b100:
        begin
            seg_sel <= 8'b1110_1111;
            seg_din <= Tenthousands;
        end
         3'b101:
         begin
            seg_sel <= 8'b1011_1111;
            seg_din <= ones;
         end
         3'b110:
         begin
            seg_sel <= 8'b0111_1111;
            seg_din <= tens;
         end
        default
        begin
            seg_sel <= 8'b1111_1111;
            seg_din <= 4'b0;
        end
```

```
            endcase
  end



endmodule

module seg_ctrl(
    input [3:0] x,
    input [7:0] sel,
    output [7:0] an,
    output reg [6:0] seg
    );
    assign an=sel;
    always @(x)
    begin
        case(x)
        4'b0000:seg=7'b0000001;
        4'b0001:seg=7'b1001111;
        4'b0010:seg=7'b0010010;
        4'b0011:seg=7'b0000110;
        4'b0100:seg=7'b1001100;
        4'b0101:seg=7'b0100100;
        4'b0110:seg=7'b0100000;
        4'b0111:seg=7'b0001111;
        4'b1000:seg=7'b0000000;
        4'b1001:seg=7'b0000100;
        default :seg=7'b0110000;
        endcase
        end

endmodule

module BCD_16binary(
    input [15:0] binary,
    output reg [3:0] Tenthousands,
    output reg [3:0] Thousands,
    output reg [3:0] Hundreds,
    output reg [3:0] Tens,
    output reg [3:0] Ones
    );
    integer i;
    always @(binary)
        begin
```

```
            Tenthousands=4'd0;
            Thousands=4'd0;
            Hundreds=4'd0;
            Tens=4'd0;
            Ones=4'd0;
            for(i=15;i>=0;i=i-1)
                begin
                if(Tenthousands>=5)
                    Tenthousands=Tenthousands+3;
                if(Thousands>=5)
                    Thousands=Thousands+3;
                if(Hundreds>=5)
                    Hundreds=Hundreds+3;
                if(Tens>=5)
                    Tens=Tens+3;
                if(Ones>=5)
                    Ones=Ones+3;
                Tenthousands=Tenthousands<<1;
                Tenthousands[0]=Thousands[3];
                Thousands=Thousands<<1;
                Thousands[0]=Hundreds[3];
                Hundreds=Hundreds<<1;
                Hundreds[0]=Tens[3];
                Tens=Tens<<1;
                Tens[0]=Ones[3];
                Ones=Ones<<1;
                Ones[0]=binary[i];
                end
            end
endmodule


module BCD_8binary(
        input [7:0] binary,
        output reg [3:0] Hundreds,
        output reg [3:0] Tens,
        output reg [3:0] Ones
        );
        integer i;
        always @(binary)
            begin
            Hundreds=4'd0;
            Tens=4'd0;
            Ones=4'd0;
```

```
        for(i=7;i>=0;i=i-1)
            begin
            if(Hundreds>=5)
                Hundreds=Hundreds+3;
            if(Tens>=5)
                Tens=Tens+3;
            if(Ones>=5)
                Ones=Ones+3;

            Hundreds=Hundreds<<1;
            Hundreds[0]=Tens[3];
            Tens=Tens<<1;
            Tens[0]=Ones[3];
            Ones=Ones<<1;
            Ones[0]=binary[i];
            end
        end
endmodule


module cpu_mem(input clk,
                input rst,
                input start,
                output [255:0] rf_data,     //寄存器,显示数据
                output[7:0] PC,               //PC 码，指令地址
                output [15:0] IR     );      //指令

    wire ROM_en;

    wire wr_ram,cs_ram;     //RAM 接口信号
    wire[7:0] addr_ram;     //RAM 地址
    wire[15:0] alu_out;      //运算模块输出

    wire clk_n;
    assign clk_n=~clk;
    Cpu Cpu(.clk(clk),
        .rst(rst),
        .start(start),
        .ROM_en(ROM_en),
        .IR(IR),
        .PC(PC),
        .rf_data(rf_data),
        .wr_ram(wr_ram),
        .cs_ram(cs_ram),
```

```
        .addr_ram(addr_ram),
        .alu_out(alu_out));

    rom rom_instruction(.clk(clk_n),
                        .rst(rst),
                        .rd(ROM_en),
                        .rom_data(IR),
                        .rom_addr(PC));

    ram ram_data(.clk(clk_n),
                 .wr(wr_ram),
                 .cs(cs_ram),
                 .addr(addr_ram),
                 .datain(alu_out)
                 );

endmodule

parameter M=16,N=8;   //有 8 根地址线，即 256*16 的只读存储器
//指令为 16 位:高 4 位为指令码,后面 16 位为 3 个寄存器 R1 R2 R3,或者整体为立即数，或者后八位为 ram 地址,PC 地址。
module   rom(input clk,
             input rst,
             input rd,
             output reg [M-1:0]rom_data,
             input [N-1:0] rom_addr);

    reg[M-1:0] memory[0:2**N-1];     //8 根地址线,16 位数据的存储器

    always @(posedge clk,posedge rst)
    if(rst)
    begin: init       //指令码初始化
        integer i;
        memory[0]=16'b1011_0000_0000_0000;
        memory[1]=16'b0000_0000_0000_0000;
        memory[2]=16'b1011_0001_0000_0000;
        memory[3]=16'b0000_0000_0000_0001;
        memory[4]=16'b1011_0010_0000_0000;
        memory[5]=16'b0000_0000_0000_0000;
        memory[6]=16'b1011_0011_0000_0000;
        memory[7]=16'b0000_0000_0000_0001;
        memory[8]=16'b1011_0100_0000_0000;
        memory[9]=16'b0000_0000_0001_0110;
        memory[10]=16'b1000_0101_0000_0100;
```

```
            memory[11]=16'b1101_0101_0001_0011;
            memory[12]=16'b0000_0001_0010_0011;
            memory[13]=16'b1010_0010_0011_0000;
            memory[14]=16'b1010_0011_0001_0000;
            memory[15]=16'b1011_0110_0000_0000;
            memory[16]=16'b0000_0000_0000_0001;
            memory[17]=16'b0000_0000_0000_0110;
            memory[18]=16'b1101_0110_0000_1010;
            memory[19]=16'b1111_0000_0000_0000;
            for(i=20;i<(2**N);i=i+1)          //存储器其余地址存放 0
                memory[i] = 0;
        end
        else
        begin                                          //读取 ROM 值
            if(rd)
              begin
              rom_data=memory[rom_addr];
              end
        end
endmodule

parameter H=16,P=8; //8 根地址线，256*16 的随机存储器，可读可写
module ram(input rd,
                input wr,
                input cs,
                input clk,
                input[P-1:0] addr,
                input[H-1:0] datain,
                output reg[H-1:0] dataout);


        reg[H-1:0] memory[0:2**P-1];
        always @(posedge clk)
        begin
            if(cs)
                if(rd) dataout<=memory[addr];
                else if(wr) memory[addr]<=datain;
            else dataout<='bz;
        end
endmodule



module Cpu(input clk,
```

```
                input rst,
                input start,
                input[15:0] IR,                 //指令寄存器内容
                output[7:0] PC,                  //PC 内容
                output ROM_en,
                output wr_ram,
                output cs_ram,   //RAM 接口信号
                output[7:0] addr_ram,
                output[15:0] alu_out,
                output [255:0] rf_data     );   //寄存器堆内容

wire[15:0] imm;
wire[3:0] sel_rf;
wire[3:0] sel_alu;
wire sel_mux;
wire r_wf,en_rf,en_reg,en_alu,en_imm,alu_zero;
wire clk_n;
   assign clk_n=~clk;
   datapath datapath1(.rst(rst),
                .clk(clk_n),
                .r_wf(r_wf),
                .en_rf(en_rf),
                .en_reg(en_reg),
                .en_alu(en_alu),
                .en_imm(en_imm),
                .sel_rf(sel_rf),
                .sel_alu(sel_alu),
                .sel_mux(sel_mux),
                .imm(imm),
                .alu_zero(alu_zero),
                .alu_out(alu_out),
                .rf_data(rf_data));
   ctrl controller(.rst(rst),
                .start(start),
                .clk(clk),
                .alu_zero(alu_zero),
                .r_wf(r_wf),
                .en_rf(en_rf),
                .en_reg(en_reg),
                .en_alu(en_alu),
                .en_imm(en_imm),
                .sel_rf(sel_rf),
                .sel_alu(sel_alu),
                .sel_mux(sel_mux),
```

```
                .imm(imm),
                .PC(PC),
                .IR(IR),
                .ROM_en(ROM_en),
                .wr_ram(wr_ram),
                .cs_ram(cs_ram),
                .addr_ram(addr_ram));

endmodule

module datapath(input rst,clk,r_wf,en_rf,en_reg,en_alu,en_imm,
                input[15:0] imm,
                input[3:0] sel_alu,
                input[3:0] sel_rf,
                input sel_mux,
                output alu_zero,
                output[255:0] rf_data,
                output [15:0] alu_out);

wire[15:0] op1,op2,out_imm,out_rf;
register register0(.clk(clk),
                .en(en_reg),
                .in(op1),
                .out(op2));
register register1(.clk(clk),
                .en(en_imm),
                .in(imm),
                .out(out_imm));
mux21 mux0(.sel(sel_mux),
                .in1(out_imm),
                .in2(out_rf),
                .out(op1));
alu alu0(.clk(clk),
            .en(en_alu),
            .sel(sel_alu),
            .in1(op1),
            .in2(op2),
            .out(alu_out),
            .alu_zero(alu_zero));
registerfile rf0(.rst(rst),
        .clk(clk),
        .r_w(r_wf),
        .enb(en_rf),
        .in(alu_out),
```

```
        .sel(sel_rf),
        .out(out_rf),
        .rf_data(rf_data));
endmodule


module register(inout clk,
                        input en,
                        input [15:0] in,
                        output reg [15:0] out);

reg[15:0] val;
always @(posedge clk)
   val<=in;
always @(en,val)
        if(en == 1'b1) out <= val;
          else ;
endmodule


module mux21(input sel,
                    input[15:0] in1,
                    input[15:0]    in2,
                    output[15:0]    out);

assign out=(sel)?in2:in1;
endmodule

//算术逻辑单元
module alu(    input en,clk,
                    input[3:0] sel,
                    input[15:0] in1,in2,
                    output reg[15:0] out,
                    output reg alu_zero);


        always @(posedge clk) begin
            if(en)
                case(sel)
                    4'b0000: out=in1;
                    4'b0001: if(in1==0) alu_zero=1; else alu_zero=0;
                    4'b0010: out=in1+in2;
                    4'b0011: out=in1-in2;
                    4'b0100: out=in1<<in2;
```

```
                4'b0101: if(in1>in2) out=16'b1; else out=16'b0;
                4'b0110: if(in1==in2) out=16'b1; else out=16'b0;
                4'b0111: if(in1>in2) out=16'b1; else out=16'b0;
                4'b1000: if(in1==0 & in2==0) out=16'b0; else out=16'b1;
                4'b1001: if(in1>0 & in2>0) out=16'b1; else out=16'b0;
                4'b1010: out=in1>>in2;
                4'b1011: out=in1*in2;
                4'b1100: out=in1/in2;
                default: ;
            endcase
        end
endmodule


//从通用寄存器中读数据到 out;写数据 in 到通用寄存器中
module registerfile( input rst,clk,enb,r_w,
                input[15:0] in,
                input[3:0] sel,
                output reg[15:0] out,
                output[255:0] rf_data);

reg[15:0] reg_file[0:15];
integer i;
//将寄存器文件数据读出
assign rf_data={reg_file[15],reg_file[14],reg_file[13],reg_file[12],reg_file[11],
                        reg_file[10],reg_file[9],reg_file[8],reg_file[7],reg_file[6],
                        reg_file[5],reg_file[4],reg_file[3],reg_file[2],reg_file[1],
                        reg_file[0]};
always @(posedge rst, posedge clk) begin
   if(rst) begin
            for(i=0;i<15;i=i+1)
                reg_file[i]<=0;
   end
   else if(enb == 1)       begin
                if(r_w==0) reg_file[sel] <= in; //写 register
                else    out <= reg_file[sel];         //读 register
            end
end
endmodule

parameter div=4'b0011,
                loadi=4'b1011,
                add=4'b0000,
                sub=4'b0001,
```

```
                jz0=4'b1100,
                jz1=4'b1101,
                store=4'b1110,
                shiftL=4'b0100,
                reg2reg=4'b1010,
                multi=4'b0010,
                shiftR=4'b0101,
                And=4'b0110,
                Or=4'b0111,
                equal=4'b1001,
                greater=4'b1000,
                halt=4'b1111;
module ctrl(input rst,start,clk,alu_zero,
                  input[15:0] IR,
                  output reg r_wf,en_rf,en_reg,en_alu,en_imm,
                  output reg[3:0] sel_rf,
                  output reg[3:0] sel_alu,
                  output reg sel_mux,
                  output reg[15:0] imm,
                  output reg[7:0] PC,
                  output reg ROM_en,
                  output reg wr_ram,cs_ram,
                  output reg[7:0] addr_ram);



parameter s0=6'b000000,s1=6'b000001,s2=6'b000010,s3=6'b000011,s4=6'b000100,
            s5=6'b000101,s5_2=6'b000110,s5_3=6'b000111,
            s6=6'b001000,s6_2=6'b001001,s6_3=6'b001010,
            s6_4=6'b001011,s6_5=6'b001100,
            s7=6'b001101,s7_2=6'b001110,s7_3=6'b001111,
            s7_4=6'b010000,s7_5=6'b010001,
            s8=6'b010010,s8_2=6'b010011,s8_3=6'b010100,
            s9=6'b010101,s9_2=6'b010110,s9_3=6'b010111,
            s10=6'b100000,s10_2=6'b100001,s10_3=6'b100010,
            s11=6'b100011,s11_2=6'b100100,s11_3=6'b100101,
            s11_4=6'b100110,s11_5=6'b100111,s12=6'b101000,
            done=6'b101001,s5_0=6'b101010;
reg[5:0] state;

reg[3:0] OPCODE;
reg[7:0] address;
reg[3:0] register;
always @(posedge rst,posedge clk)    begin
```

```
      sel_mux<=1'b1;                    //每次若其没改，都选择 in2,第一个数据都会先进入寄存
器，然后第二个数据直接进入 alu
       en_rf<=1'b0;                     //使能全为 0
       en_reg<=1'b0;
       en_alu<=1'b0;
       en_imm<=1'b0;
      ROM_en<=1'b0;                     //ROM 输出控制信号
      wr_ram<=1'b0;                    //写
       cs_ram<=1'b0;                   //RAM 接口信号
//       addr_ram<=0;
      if(rst)      begin
              state<=s0;
              PC<=0;
           end
      else begin
         case(state)
           s0: begin              //  开始
                    PC <= 0;
                    state <= s1;
                end
           s1: begin              //  取指令，每次指令操作完成后，PC 会加 1，然后时钟下降沿
时会将指令重新取出
                    if(start == 1'b1) begin    //start 控制单步执行,可由按键控制继续
                       ROM_en<=1;
                       state <= s2;
                    end
                    else state <= s1;//start=0,留在此
                 end
           s2:    begin             //  拆分指令
                     OPCODE <= IR[15:12];   //操作码
                     register<=IR[11:8];     //第一寄存器
                     address<= IR[7:0];      //第二，三寄存器或 ram 地址
                     state <= s3;
                end
           s3:   begin                  //PC++
              PC <= PC + 8'b1;
              state <= s4;
             end
           s4:   begin                  //  解指令码
                    case(OPCODE)
                    loadi:      state <= s5_0;
                    add:        state <= s6;
                    sub:        state <= s6;
                    div:       state<= s6;
```

```
            multi:    state<= s6;
            And:      state<= s6;
            Or:       state<=s6;
            greater: state<=s6;
            equal:    state<=s6;
            jz0:  state <= s8;
            jz1:      state <= s8;
            store:     state <= s9;
            reg2reg: state <= s10;
            shiftL:   state <= s11;
            shiftR:   state<=s11;
            halt:      state <= done;
            default: state <= s1;
            endcase
 end
s5_0:begin   //如果是赋值，继续取出指令码，作为立即数
      ROM_en<=1;
      state<=s5;
      end
s5: begin                 // loadi，赋值,立即数进入 imm 寄存器
            imm<=IR;    //赋值的话，下一条指令为立即数，PC 再加 1
            en_imm<=1;
            PC<=PC+8'b1;//然后 PC 在加 1
            state <= s5_2;
       end
s5_2: begin       //选择 in1,上升沿 mux21 马上输出立即数，下降沿 alu 直通
            sel_mux<=0;
            en_alu<=1;
            sel_alu<=4'b0000;
            state <= s5_3;
       end
 s5_3: begin                                          //写入寄存器堆
            en_rf<=1;
            r_wf<=0;
            sel_rf<=register;
            state <= s12;
       end
s6: begin                 // 加,减，乘，除，与，或，取出 R3
            sel_rf<=IR[3:0];
            en_rf<=1;
            r_wf<=1;
            state <= s6_2;
       end
s6_2: begin
```

```
                en_reg<=1;
                state <= s6_3;
            end
        s6_3: begin     //取 R2
                sel_rf<=IR[7:4];
                en_rf<=1;
                r_wf<=1;
                state <= s6_4;
            end
        s6_4: begin
                en_alu<=1;
                case(OPCODE)
                add:          sel_alu<=4'b0010;
                sub:          sel_alu<=4'b0011;
                div:          sel_alu<=4'b1100;
                multi:        sel_alu<=4'b1011;
                And:          sel_alu<=4'b1001;
                Or:           sel_alu<=4'b1000;
                greater:    sel_alu<=4'b0101;
                equal:        sel_alu<=4'b0110;
                default: ;
                endcase
                state <= s6_5;
            end
        s6_5: begin     //存到 R1
                sel_rf<=register;
                en_rf<=1;
                r_wf<=0;
                state <= s12;
            end
/*          s7: begin                       // sub,减
                sel_rf<=IR[3:0];
                en_rf<=1;
                r_wf<=1;
                state <= s7_2;
            end
        s7_2: begin
                en_reg<=1;
                state <= s7_3;
            end
        s7_3: begin
                sel_rf<=IR[7:4];
                en_rf<=1;
                r_wf<=1;
```

```
                state <= s7_4;
        end
s7_4: begin
                en_alu<=1;

                state <= s7_5;
        end
s7_5: begin
                sel_rf<=register;
                en_rf<=1;
                r_wf<=0;
                state <= s12;
        end             */
s8: begin                       // jz0,当 R1 为 0，跳转   jz1,当 R1 为 1，跳转
          en_rf<=1;
          r_wf<=1;
          sel_rf<=register;
          state<=s8_2;
        end
s8_2: begin
          en_alu<=1;
          sel_alu<=4'b0001;
          state <= s8_3;
        end
s8_3: begin
       case(OPCODE)
       jz0:begin
              if(alu_zero==1)
                  PC <= address;
            end
       jz1:begin
                if(alu_zero==0)
                    PC <= address;
            end
       endcase
       state <= s12;
        end
  s9:   begin        // store,将 R1 地址寄存器存储的数存储到 ram
       sel_rf<=register;
       en_rf<=1;
       r_wf<=1;
       state <= s9_2;
          end
  s9_2: begin
```

```
        en_alu<=1;
        sel_alu<=4'b0000;
        state <= s9_3;
        end
s9_3:   begin
        cs_ram<=1;   //选中 RAM
        wr_ram<=1;   //写入 RAM
        addr_ram<=address;
        state <= s12;
         end
s10:    begin                  //reg2reg1，将 R2 值赋给 R1
       sel_rf<=IR[7:4];
       en_rf<=1;
       r_wf<=1;
       state <= s10_2;
        end
s10_2: begin
       en_alu<=1;
       sel_alu<=4'b0000;
       state <= s10_3;
        end
s10_3: begin
       sel_rf<=register;
       en_rf<=1;
       r_wf<=0;
       state <= s12;
        end
s11: begin                    //shift left,shitf right,R1 左移，R1 右移，后八位是左移位数

       imm<={8'b0000_0000,address};
       en_imm<=1;
       state <= s11_2;
        end
s11_2: begin
       sel_mux<=0;
       en_reg<=1;
       state <= s11_3;
        end
s11_3: begin
       sel_rf<=register;
       en_rf<=1;
       r_wf<=1;
       state <= s11_4;
        end
```

```
            s11_4: begin
                  en_alu<=1;
                  case(OPCODE)
                  shiftL:          sel_alu<=4'b0100;
                  shiftR:          sel_alu<=4'b1010;
                  endcase
                  state <= s11_5;
                    end
            s11_5: begin
                  sel_rf<=register;
                  en_rf<=1;
                  r_wf<=0;
                  state <= s12;
                    end
            s12:    state <= s1;       //返回下一条指令
            done:    state <= done; //  结束
            default:
                  PC<=0;
          endcase
        end
    end
endmodule
```