```
module ifu(clk,reset,npc_sel,zero,insout,jpcnew);

  input clk,reset,zero;     //时钟，重置，0（用于比较）
  input [1:0]npc_sel;     //选择 t0(addu,subu,ori,lw,sw)还是 t1(beq)
  output [31:0]insout;      //输出的指令
  output [31:0]jpcnew;      //j 指令后的新地址

  reg [31:0]pc;     //pc 计数器
  wire [31:0] pcnew;     //pc 计数器的新值
  wire [31:0] t1;       //传输 beq 指令的地址码=(pc+4)接 imm16 经符号位扩展后
的 32 位后左移 2 位
  wire [31:0] t0;      //addu,subu,ori,lw,sw 均采用 pc+4 的地址码
  wire [31:0] extout;     //传输 imm16 经符号位扩展后的 32 位后左移 2 位后的值
(beq)
  wire [31:0] immext;     //传输 imm16 经符号位扩展后的 32 位的值(beq)
  wire [31:0] jbranch;      //传输 j 采用伪直接寻址即为 pc+4 后的值+imm16 左移
2 位变为 28 位数+00 所获得的 32 位地址码
  wire [31:0] jpcnew;     //执行 j 指令后 pc 的新值
  wire [15:0] imm16;      //16 位立即数
  wire [25:0] imm26;      //26 位立即数
  reg [7:0] im[1023:0];   //声明 1KB 的 im(命令存储器)，8 根线，包含 1024 个的
数组

  initial begin
     $readmemh("code.txt",im);     //读取指令
  end

  assign insout = {im[pc[9:0]], im[pc[9:0]+1], im[pc[9:0]+2],
im[pc[9:0]+3]};
       //取指令，一次取 8 位，连续去四次，合计 32 位为一条指令
  assign imm16 = insout[15:0];      //传输指令第 1 位至第 16 位，用于
lui,ori,sw,lw 的立即数和 beq 的扩展
  assign imm26 = insout[25:0];      //传输指令第 1 位至第 26 位，用于 j 指令的
26 位的地址码
  assign immext = {{16{imm16[15]}},imm16};
       //beq 指令符号扩展，{16{imm16[15]}意为将 imm16 的第 16 位(即符号位)扩展
16 次，{{16{imm16[15]}},imm16}最后在连接 imm16
  assign extout = immext<<2;      //beq,j 都需要左移 2 位
  assign jbranch = {t0[31:28],imm26,2'b0};
       //j 采用伪直接寻址即为 pc+4 后的值+imm16 左移 2 位变为 28 位数+00

  always@(posedge clk,posedge reset)
  /*
```

```
    1、always 语句有两种触发方式。第一种是电平触发，例如 always @(a or b or
c)，a、b、c 均为变量，当其中一个发生变化时，下方的语句将被执行。
    2、第二种是沿触发，例如 always @(posedge clk or negedge rstn)，即当时钟
处在上升沿或下降沿时，语句被执行。
  */
  begin
    if(reset) pc = 32'h0000_3000;     //当 Reset 信号为高电平时（同步），重
置 PC 为 0x00003000
    else pc = pcnew;      //当时钟上升沿到来时将 pcnew 写入 PC 内部，并且输出
  end

  assign pcnew =
(npc_sel==2'b10||npc_sel==2'b11)?jbranch:(npc_sel&&zero)?t1:t0;
assign jpcnew = pc+8;     //计算 j 指令后 pc 的新值
  assign t0 = pc+4;     //addu,subu,ori,lw,sw 均采用 pc+4 的顺序寻址
  assign t1 = t0+extout;      //计算 beq 指令的地址码=(pc+4)接 imm16 经符号位
扩展后的 32 位后左移 2 位
endmodule




module DataMemory(clk,MemWr,DataIn,Addr,DataOut);

  input clk,MemWr;
  input [31:0] DataIn,Addr;

  output reg [31:0]DataOut;

  reg [7:0]memory[0:1023];
  reg [31:0] address;

  integer i;
  initial begin
    for (i=0; i<1024; i=i+1)
      memory[i] <= 0;
  end
  always @(MemWr or DataIn or Addr or posedge clk)
    begin
    address = Addr << 2;
    DataOut =
(memory[address]<<24)+(memory[address+1]<<16)+(memory[address+2]<<8)+
memory[address+3];
    if (MemWr == 1)
```

```
      begin
        address = Addr << 2;
        memory[address] = DataIn[31:24];
        memory[address+1] = DataIn[23:16];
        memory[address+2] = DataIn[15:8];
        memory[address+3] = DataIn[7:0];
      end
    end
endmodule


module ext(imm,ExtOp,Extout);

  input [15:0] imm;
  input ExtOp;

  output [31:0]Extout;

    assign Extout[15:0] = imm;
    assign Extout[31:16] =
ExtOp?(imm[15]?16'hffff:16'h0000):16'h0000;
          //1 为符号扩展
endmodule



  module
Control(op,func,RegDst,ALUSrc,MemtoReg,RegWr,MemWr,nPC_sel,ExtOp,ALUc
tr,add8);

  input [5:0]op,func;     //界定具体指令

  output RegDst;      //仅有 aadu,subu 用到 rd 寄存器才会选择 1
  output ALUSrc;      //控制 ALU 的输入是来自寄存器堆还是扩展单元
  output MemtoReg;
  output RegWr;
  output MemWr;
  output [1:0] nPC_sel;
  output ExtOp,add8;
  output [2:0] ALUctr;

  wire [2:0]ADD,Or,Subtract;
```

```
  assign RegDst = (op==6'b000000)?1:0;
  assign ALUSrc = (op==6'b000000||op==6'b000100)?0:1;
//addu,subu,beq 选择 0
  assign MemtoReg = (op==6'b100011)?1:0;      //仅有 lw 选择 1
  assign RegWr = (op==6'b101011||op==6'b000100)?0:1;      //仅有 sw,beq
选择 0
  assign MemWr = (op==6'b101011)?1:0;     //仅有 sw 选择 0
  assign nPC_sel =
(op==6'b000011)?2'b11:(op==6'b000010)?2'b10:(op==6'b000100)?2'b01:2'b
00;
       //依次对应 jal,j,beq

  assign ExtOp = (op==6'b001101)?0:1;     //ori 指令选择 0

  assign ADD = 3'b000;
  assign Subtract = 3'b001;
  assign Or = 3'b100;
  assign ALUctr =
(op==6'b001111)?3'b011:(op==6'b001101)?Or:(func==6'b100011||op==6'b00
0100)?Subtract:ADD;
     //依次对应 lui,ori,subu,addu

  endmodule



module MUX2X5(A,B,Sel,Out);
          //insout[20:16],insout[15:11],RegDst,rw
    input [4:0]A,B;
    input Sel;
    output [4:0]Out;
    function [4:0]select;
        input [4:0]A,B;
        input Sel;
        case(Sel)
            1'b0:select = A;
            1'b1:select = B;
        endcase
    endfunction
    assign Out = select(A,B,Sel);
endmodule
```

```
module MIPS(clk,reset);

  input reset,clk;

  wire [31:0]insout,jpcnew;
  wire [4:0]rw,rt;
  wire [2:0]ALUctr;
  wire [1:0]nPC_sel;

  wire RegWr,RegDst,ExtOp,ALUSrc,MemWr,MemtoReg,zero,add8;
  wire [31:0]busA,busB,busO,busE,busB1,DataO,busW;

  assign rt = (nPC_sel==2'b11)?5'b11111:insout[20:16];

  Control
cu0(insout[31:26],insout[5:0],RegDst,ALUSrc,MemtoReg,RegWr,MemWr,nPC_
sel,ExtOp,ALUctr,add8);
  ifu i1(clk,reset,nPC_sel,zero,insout,jpcnew);
  MUX2X5 M1(insout[20:16],insout[15:11],RegDst,rw);
  RegFile R1(clk,RegWr,busA,busB,busW,insout[25:21],rt,rw,jpcnew);
  ext E1(insout[15:0],ExtOp,busE);
  MUX2X32 M2(busB,busE,ALUSrc,busB1);
  ALU A1(busA,busB1,ALUctr,zero,busO);
  DataMemory D1(clk,MemWr,busB,busO,DataO);
  MUX2X32 M3(busO,DataO,MemtoReg,busW);

endmodule




module MUX2X32(A,B,Sel,Out);
          //M2: busB,busE,ALUSrc,busB1
          //M3: busO,DataO,MemtoReg,busW
    input [31:0]A,B;
    input Sel;
    output [31:0]Out;
    function [31:0]select;
       input [31:0]A,B;
       input Sel;
       case(Sel)
          1'b0:select = A;
```

```
            1'b1:select = B;
        endcase
    endfunction
    assign Out = select(A,B,Sel);
endmodule



module test;
  reg clk,reset;

  MIPS ct1(clk,reset);
  initial
    begin
      clk = 1;
      reset = 0;
      #5 reset = 1;
      #5 reset = 0;
    end
    always
    begin
      #30 clk=~clk;
    end
endmodule



module RegFile(clk,RegWr,busA,busB,busW,RA,RB,RW,jpcnew);

  input [4:0] RA;     //相当于 rs
  input [4:0] RB;     //相当于 rt
  input [4:0] RW;     //rd 或 rt
  input [31:0] busW;
  input [31:0] jpcnew;      //连接自 IFU
  input clk;
  input RegWr;

  output [31:0] busA;
  output [31:0] busB;

  reg [31:0] register[0:31];
    integer i;
    initial begin
```

```
      for (i = 0; i < 32; i = i+1)
        register[i] <= 0;
    end

  assign busA = register[RA];
  assign busB = register[RB];

  always @(RegWr or RW or busW or posedge clk)
    begin
      if (RegWr && RW!=5'b0)      //addu,subu 需要取 rd,并且满足不能是零寄存
器
        register[RW] = busW;      //rd = rs +/- rt
      if(RB==5'b11111)
        register[RB] = jpcnew;      //rt 写入 j 采用伪直接寻址即为 pc+4 后的值
+imm16 左移 2 位变为 28 位数+00 所获得的 32 位地址码
    end
endmodule



module ALU(busA,busB,ALUctr,zero,Out);

  input [31:0] busA,busB;
  input [2:0] ALUctr;     //可控制 lui,ori,subu,addu
  output reg zero;
  output reg [31:0]Out;

  wire [31:0]A;
  wire [31:0]B;

  assign B = busB;
  assign A = busA;

  always@(busA,busB,ALUctr,A,B)
    begin
      case(ALUctr)      //addu
      3'b000:begin
        Out = A+B;
        zero = (Out==0)?1:0;
      end
      3'b001:begin      //subu
        Out = A-B;
        zero = (Out==0)?1:0;
      end
```

```
      3'b100:begin      //ori 进行按位或运算
        Out = A|B;
        zero = (Out==0)?1:0;
      end
      3'b011:begin      //lui 把一个 16 位的立即数填入到寄存器的高 16 位，写入 rt
        Out = B<<16;
        zero = (Out==0)?1:0;
      end
    endcase
  end
endmodule
```