中国科学院大学计算机组成原理实验课

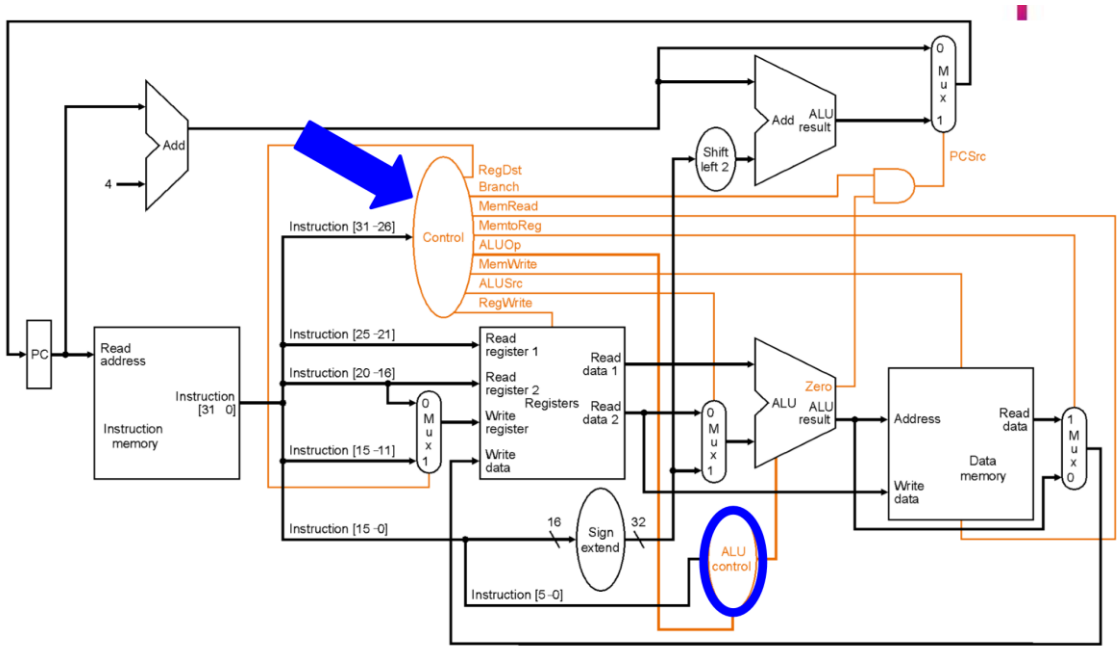
实 验 报 告

学号：2016K8009915009　姓名：钟赟　专业：计算机科学与技术

实验序号：__ 实验名称：单周期处理器设计

一、 逻辑电路结构与仿真波形的截图及说明

1.CPU 中各个功能部件的逻辑关系基本如下图（增加指令集后，实际控制信号数大于图中所示信号数）：



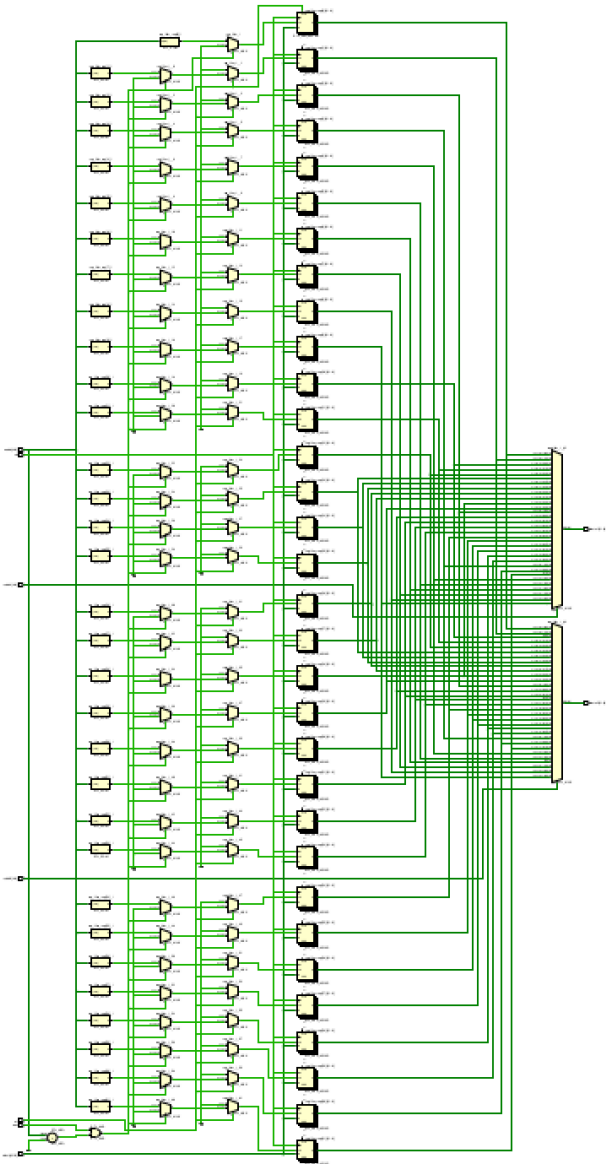
2.关键 RTL 代码段

（1）reg_file：储存寄存器组，并根据地址对寄存器进行读写

复用实验项目 2 的代码：

```
`define DATA_WIDTH 32
`define ADDR_WIDTH 5

module reg_file(
        input clk,
        input rst,
        input [`ADDR_WIDTH - 1:0] waddr,
        input [`ADDR_WIDTH - 1:0] raddr1,
        input [`ADDR_WIDTH - 1:0] raddr2,
        input wen,
        input [`DATA_WIDTH - 1:0] wdata,
        output [`DATA_WIDTH - 1:0] rdata1,
        output [`DATA_WIDTH - 1:0] rdata2
);

    reg [`DATA_WIDTH - 1:0] reg_files [0:`DATA_WIDTH - 1];//定义32个32位寄存器组

    //时钟逻辑实现同步写
    always @(posedge clk or posedge rst)//
    begin
        if(rst)//读到复位信号，将0号寄存器置为0
            begin
                reg_files[0] <= 32'b0;
            end
        else
        begin
            if(wen && waddr != 5'b0)//对于0号寄存器以外的寄存器，当写使能为1时，读入数据
                reg_files[waddr] <= wdata;
        end
    end

    //组合逻辑实现异步读
    assign rdata1 = reg_files[raddr1];//读口1读入数据
    assign rdata2 = reg_files[raddr2];//读口2读入数据
```

原理图：

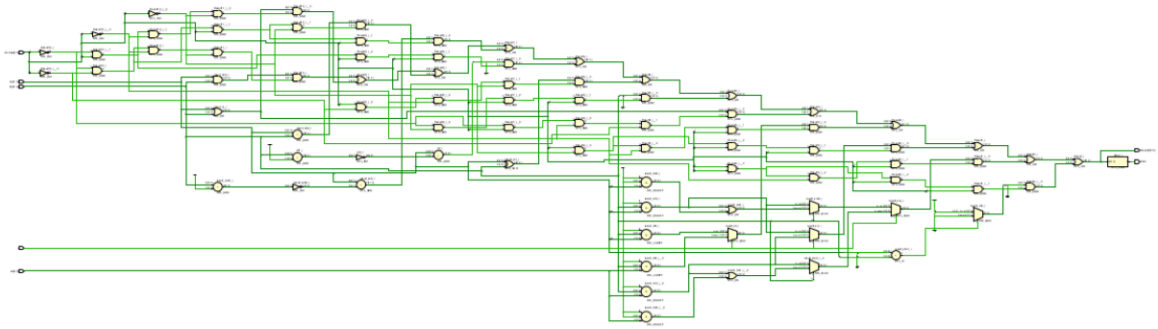


（2）alu 算数逻辑单元：用于逻辑计算指令和跳转指令的比较

在实验 2 的 alu 代码的基础上增删一些内容：

```
`timescale 10 ns / 1 ns

module alu(
  input [31:0] A,//第一个操作数
  input [31:0] B,//第二个操作数
  input [3:0] ALUop,//alu操作码
  input [4:0] sa,//指令的6~10位,作为移位操作时可能的操作数
  input v,//用来控制移位操作时操作数的选择
  output Zero, //结果为0时Zero为1
  output [31:0] Result//alu计算结果
);
  //每种操作对应的操作码ALUop
  wire [31:0] result_and;//0000
  wire [31:0] result_or;//0001
  wire [31:0] result_add;//0010
  wire [31:0] result_lui;//0011
  wire [31:0] result_nor;//0100
  wire [31:0] result_sub;//0110
  wire [31:0] result_slt;//0111
  wire [31:0] result_xor;//1000
  wire [31:0] result_sll;//1001
  wire [31:0] result_srl;//1011
  wire [31:0] result_sra;//1101
  wire [31:0] result_sltu;//1111

  wire [32:0] u_A, u_B;//最高位扩展一位0的操作数，用于无符号数计算
  wire [32:0] slt;//slt操作的中间变量

  assign result_and = A & B;
  assign result_or = A | B;
  assign result_add = A + B;
  assign result_lui = {B[15:0], 16'h0000};
  assign result_sub = A + ~(B+32'hffffffff);
  assign result_xor = A ^ B;
  assign result_nor = ~(A | B);
  assign result_sll = v ? (B << A[4:0]) : (B << sa);//v=1,执行sllv，下面执行对应操作
  assign result_srl = v ? (B >> A[4:0]) : (B >> sa);
  assign result_sra = v ? ((B[31]==0) ? B>>A[4:0] : (B>>A[4:0]|(~(32'hffffffff>>A
[4:0])))) : ((B[31]==0) ? B>>sa : (B>>sa|(~(32'hffffffff>>sa))));


  assign slt = {A[31],A} + ~({B[31],B}+33'h1ffffffff);
  assign result_slt = {31'd0, slt[32]};

  assign u_A = {1'b0, A};
  assign u_B = {1'b0, B};
  assign result_sltu = (u_A < u_B) ? 32'd1 : 32'd0;

  assign Result = (result_and & {32{(~ALUop[3]&~ALUop[0]&~ALUop[1]&~ALUop[2])}}) |
      (result_or & {32{(~ALUop[3]&ALUop[0]&~ALUop[1]&~ALUop[2])}}) |
      (result_add & {32{(~ALUop[3]&~ALUop[0]&ALUop[1]&~ALUop[2])}}) |
      (result_sub & {32{(~ALUop[3]&~ALUop[0]&ALUop[1]&ALUop[2])}}) |
      (result_slt & {32{(~ALUop[3]&ALUop[0]&ALUop[1]&ALUop[2])}}) |
      (result_xor & {32{(ALUop[3]&~ALUop[0]&~ALUop[1]&~ALUop[2])}}) |
      (result_lui & {32{(~ALUop[3]&~ALUop[2]&ALUop[1]&ALUop[0])}}) |
      (result_nor & {32{(~ALUop[3]&ALUop[2]&~ALUop[1]&~ALUop[0])}}) |
      (result_sll & {32{(ALUop[3]&~ALUop[2]&~ALUop[1]&ALUop[0])}}) |
      (result_srl & {32{(ALUop[3]&~ALUop[2]&ALUop[1]&ALUop[0])}}) |
      (result_sra & {32{(ALUop[3]&ALUop[2]&~ALUop[1]&ALUop[0])}}) |
      (result_sltu & {32{(ALUop[3]&ALUop[2]&ALUop[1]&ALUop[0])}});


  assign Zero = (Result == 32'd0) ? 32'd1 : 32'd0;

endmodule
```

原理图：



（3）control_unit：控制信号模块，通过解析指令的操作码得到该指令的各种控制信号

各个控制信号的释义如下：

控制信号名	功能
RegDst	控制写寄存器组寄存器号
MemRead	内存读使能信号
MemWrite	内存写使能信号
Mem2Reg	控制写入寄存器的数据
ALUsrc	控制 alu 中第二个操作数
RegWrite	寄存器写使能信号
PCsrc	beq 和 bne 跳转指令 PC 控制信号
ExtSel	有符号扩展或无符号扩展控制信号
jump	jump 指令 PC 控制信号
jr	jr 指令 PC 控制信号
jal	jal 指令 PC 控制信号
jalr	jalr 指令 PC 控制信号
move	move 指令 alu 第一个操作数控制信号

v	移位指令中第一位操作数控制信号

```
`timescale 10 ns / 1 ns

module control_unit(
  input [5:0] op,//Instruction[31:26]
  inout [5:0] func,//Instruction[5:0]
  input [4:0] rt,//Instruction[20:16]
  input Zero,
  input [31:0] ea, rdata1,
  output reg RegDst, MemRead, Mem2Reg, MemWrite, ALUsrc, RegWrite, PCsrc, jump,
jr, jal, jalr, ExtSel, move, v,//各种控制信号
  output reg [3:0] ALUop,
  output reg [3:0] Write_strb
);

  parameter ADDIU = 6'b001001, LW = 6'b100011, SW = 6'b101011, BNE = 6'b000101,
NOP = 6'b000000, ADDU = 6'b100001, BEQ = 6'b000100, LUI = 6'b001111, SUBU =
6'b100011, SLL = 6'b000000, SLTI = 6'b001010, SLT = 6'b101010, SLTIU = 6'b001011,
OR = 6'b100101, ORI = 6'b001101, AND = 6'b100100, MOVE = 6'b100101, J = 6'b000010,
JAL = 6'b000011, JR = 6'b001000, ANDI = 6'b001100, BGEZ = 5'b00001, BLEZ =
6'b000110, BLTZ = 5'b00000, JALR = 6'b001001, LB = 6'b100000, LBU = 6'b100100, LH
= 6'b100001, LHU = 6'b100101, LWL = 6'b100010, LWR = 6'b100110, MOVN = 6'b001011,
MOVZ = 6'b001010, NOR = 6'b100111, SB = 6'b101000, SH = 6'b101001, SLLV =
6'b000100, SLTU = 6'b101011, SRA = 6'b000011, SRAV = 6'b000111, SRL = 6'b000010,
SRLV = 6'b000110, SWL = 6'b101010, SWR = 6'b101110, XOR = 6'b100110, XORI =
6'b001110, SPE = 6'b000000;

  //进行各种赋值
  initial begin
    RegDst = 0;
    MemRead = 0;
    Mem2Reg = 0;
    MemWrite = 0;
    ALUsrc = 0;
    RegWrite = 0;
    PCsrc = 0;
    ExtSel = 1;
```

```
    jump = 0;
    jr = 0;
    jal = 0;
    jalr = 0;
    move = 0;
    v = 0;
    ALUop = 0;
    Write_strb = 0;
  end

  always @ (*) begin
    case (op)
      LW, LWL, LWR, LB, LBU, LH, LHU:
      begin
        PCsrc = 0;
        RegDst = 0;
        MemRead = 1;
        Mem2Reg = 1;
        MemWrite = 0;
        ALUsrc = 1;
        RegWrite = 1;
        ExtSel = 1;
        jump = 0;
        jr = 0;
        jal = 0;
        jalr = 0;
        move = 0;
        ALUop = 4'b0010;
        Write_strb = 4'b0000;
      end

      SW, SWL, SWR, SB, SH:
      begin
        PCsrc = 0;
        RegDst = 0;
        MemRead = 0;
        Mem2Reg = 0;
        MemWrite = 1;
        ALUsrc = 1;
        RegWrite = 0;
        ExtSel = 1;
        jump = 0;
        jr = 0;
        jal = 0;
        jalr = 0;
        move = 0;
        ALUop = 4'b0010;
        if(op == SWL || op == SWR)  begin
          case(ea[1:0])
            2'b00:
              Write_strb = (op == SWL) ? 4'b0001 : 4'b1111;
            2'b01:
              Write_strb = (op == SWL) ? 4'b0011 : 4'b1110;
            2'b10:
              Write_strb = (op == SWL) ? 4'b0111 : 4'b1100;
            2'b11:
              Write_strb = (op == SWL) ? 4'b1111 : 4'b1000;
          endcase
        end
        else if(op == SB)   begin
          case(ea[1:0])
            2'b00:
              Write_strb = 4'b0001;
            2'b01:
              Write_strb = 4'b0010;
            2'b10:
              Write_strb = 4'b0100;
            2'b11:
              Write_strb = 4'b1000;
          endcase
        end
        else if(op == SH)   begin
          case(ea[1])
            1'b0:
```

```
        Write_strb = 4'b0011;
      1'b1:
        Write_strb = 4'b1100;
    endcase
  end
  else
    Write_strb = 4'b1111;
end
BNE: begin
  PCsrc = !Zero;
  RegDst = 0;
  MemRead = 0;
  Mem2Reg = 0;
  MemWrite = 0;
  ALUsrc = 0;
  RegWrite = 0;
  ExtSel = 1;
  jump = 0;
  jr = 0;
  jal = 0;
  jalr = 0;
  move = 0;
  ALUop = 3'b0110;
  Write_strb = 4'b0000;
end
ADDIU: begin
  PCsrc = 0;
  RegDst = 0;
  MemRead = 0;
  Mem2Reg = 0;
  MemWrite = 0;
  ALUsrc = 1;
  RegWrite = 1;
  ExtSel = 1;
  jump = 0;
  jr = 0;
  jal = 0;
  jalr = 0;
```

```
    move = 0;
    ALUop = 4'b0010;
    Write_strb = 4'b0000;
  end
  BEQ:  begin
    PCsrc = Zero ? 1 : 0;
    RegDst = 0;
    Mem2Reg = 0;
    MemWrite = 0;
    ALUsrc = 0;
    RegWrite = 0;
    ExtSel = 1;
    jump = 0;
    jr = 0;
    jal = 0;
    jalr = 0;
    move = 0;
    ALUop = 3'b0110;
    Write_strb = 4'b0000;
  end
  LUI:  begin
    PCsrc = 0;
    RegDst = 0;
    MemRead = 0;
    Mem2Reg = 0;
    MemWrite = 0;
    ALUsrc = 1;
    RegWrite = 1;
    ExtSel = 1;
    jump = 0;
    jr = 0;
    jal = 0;
    jalr = 0;
    move = 0;
    ALUop = 4'b0011;
    Write_strb = 4'b0000;
  end
  SLTI: begin
```

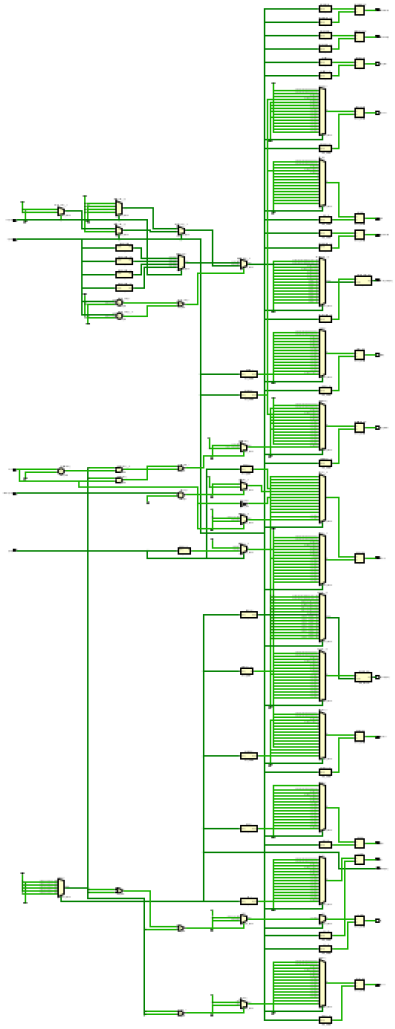
```
        PCsrc = 0;
        RegDst = 0;
        MemRead = 0;
        Mem2Reg = 0;
        MemWrite = 0;
        ALUsrc = 1;
        RegWrite = 1;
        ExtSel = 1;
        jump = 0;
        jr = 0;
        jal = 0;
        jalr = 0;
        move = 0;
        ALUop = 4'b0111;
        Write_strb = 4'b0000;
      end
      SLTIU:  begin
        PCsrc = 0;
        RegDst = 0;
        MemRead = 0;
        Mem2Reg = 0;
        MemWrite = 0;
        ALUsrc = 1;
        RegWrite = 1;
        ExtSel = 1;
        jump = 0;
        jr = 0;
        jal = 0;
        jalr = 0;
        move = 0;
        ALUop = 4'b0111;
        Write_strb = 4'b0000;
      end
      ANDI: begin
        PCsrc = 0;
        RegDst = 0;
        MemRead = 0;
        Mem2Reg = 0;
```

```
  MemWrite = 0;
  ALUsrc = 1;
  RegWrite = 1;
  ExtSel = 0;
  jump = 0;
  jr = 0;
  jal = 0;
  jalr = 0;
  move = 0;
  ALUop = 4'b0000;
  Write_strb = 4'b0000;
end
ORI:  begin
  PCsrc = 0;
  RegDst = 0;
  MemRead = 0;
  Mem2Reg = 0;
  MemWrite = 0;
  ALUsrc = 1;
  RegWrite = 1;
  ExtSel = 0;
  jump = 0;
  jr = 0;
  jal = 0;
  jalr = 0;
  move = 0;
  ALUop = 4'b0001;
  Write_strb = 4'b0000;
end
XORI:   begin
  PCsrc = 0;
  RegDst = 0;
  MemRead = 0;
  Mem2Reg = 0;
  MemWrite = 0;
  ALUsrc = 1;
  RegWrite = 1;
  ExtSel = 0:
```

```
    jump = 0;
    jr = 0;
    jal = 0;
    jalr = 0;
    move = 0;
    ALUop = 4'b1000;
    Write_strb = 4'b0000;
  end
  J, JAL: begin
    PCsrc = 0;
    RegDst = 0;
    MemRead = 0;
    Mem2Reg = 0;
    MemWrite = 0;
    ALUsrc = (op == JAL) ? 1 : 0;
    RegWrite = (op == JAL) ? 1 : 0;
    ExtSel = 1;
    jump = (op == J) ? 1 : 0;
    jr = 0;
    jal = (op == JAL) ? 1 : 0;
    jalr = 0;
    move = 0;
    ALUop = 4'b0000;
    Write_strb = 4'b0000;
  end
  6'b000001: begin //BGEZ,BLTZ
    RegDst = 0;
    MemRead = 0;
    Mem2Reg = 0;
    MemWrite = 0;
    ALUsrc = 0;
    RegWrite = 0;
    ExtSel = 1;
    jump = 0;
    jr = 0;
    jal = 0;
    jalr = 0;
    move = 0;
```

```
        ALUop = 4'b0111;
        Write_strb = 4'b0000;
        if(rt == BGEZ)
          PCsrc = (rdata1 < 0) ? 0 : 1;
        else if(rt == BLTZ)
          PCsrc = (rdata1 < 0) ? 1 : 0;
      end
      BLEZ: begin
        PCsrc = (rdata1 > 0) ? 0 : 1;
        RegDst = 0;
        MemRead = 0;
        Mem2Reg = 0;
        MemWrite = 0;
        ALUsrc = 0;
        RegWrite = 0;
        ExtSel = 1;
        jump = 0;
        jr = 0;
        jal = 0;
        jalr = 0;
        move = 0;
        ALUop = 4'b0111;
        Write_strb = 4'b0000;
      end
      SPE: begin//R-type instructions
        PCsrc = 0;
        RegDst = 1;
        MemRead = 0;
        Mem2Reg = 0;
        MemWrite = 0;
        ALUsrc = 0;
        RegWrite = ((func==MOVN && Zero==1)||(func==MOVZ && Zero==0)) ? 0 : 1;
        jump = 0;
        jr = (func == JR) ? 1 : 0;
        jal = 0;
        jalr = (func == JALR) ? 1 : 0;
        ExtSel = (func == SLTU) ? 0 : 1;
        move = (func==MOVN || func==MOVZ) ? 1 : 0;
        v = (func==SLLV||func==SRLV||func==SRAV) ? 1 : 0;
        Write_strb = 4'b0000;
        case(func)
          ADDU:          ALUop = 4'b0010;
          SUBU:          ALUop = 4'b0110;
          SLL:           ALUop = 4'b1001;
          SLT:           ALUop = 4'b0111;
          SLTU:          ALUop = 4'b1111;
          AND:           ALUop = 4'b0000;
          OR:            ALUop = 4'b0001;
          NOR:           ALUop = 4'b0100;
          XOR:           ALUop = 4'b1000;
          SLL, SLLV:     ALUop = 4'b1001;
          SRL, SRLV:     ALUop = 4'b1011;
          SRA, SRAV:     ALUop = 4'b1101;
          MOVZ, MOVN:    ALUop = 4'b0001;
        endcase
      end
    endcase
  end
endmodule
```

原理图：



（4）mips_cpu：顶层连接模块，以及 PC、寄存器写数据和内存写数据的赋值

```
`timescale 10ns / 1ns

module mips_cpu(
  input  rst,
  input  clk,
  input  [31:0] Instruction,
  input  [31:0] Read_data,
  output reg [31:0] PC,
  output [31:0] Address,
  output MemWrite,
  output reg [31:0] Write_data,
  output [3:0] Write_strb,
  output MemRead
);

  wire [5:0] op, func;
  wire [4:0] rs, rt, rd, sa, waddr;
  wire [15:0] immediate;
  wire [31:0] immediate_32, rdata1, rdata2, Result, ea, result_slt, A, B;
  wire [3:0] ALUop;
  wire Zero, MemWrite, MemRead, RegDst, Mem2Reg, ALUsrc, RegWrite, PCsrc, jump,
ExtSel, jr, jal, jalr, move, v;

  reg [31:0] wdata;

  //alu operands
  assign A = move ? 32'd0 : rdata1;
  assign B = ALUsrc ? immediate_32 : rdata2;

  assign op = Instruction[31:26];
  assign rs = Instruction[25:21];
  assign rt = Instruction[20:16];
  assign rd = Instruction[15:11];
  assign sa = Instruction[10:6];
  assign immediate = Instruction[15:0];
  assign func = Instruction[5:0];
  assign ea = Result;
  assign Address = {ea[31:2], 2'b00};
```

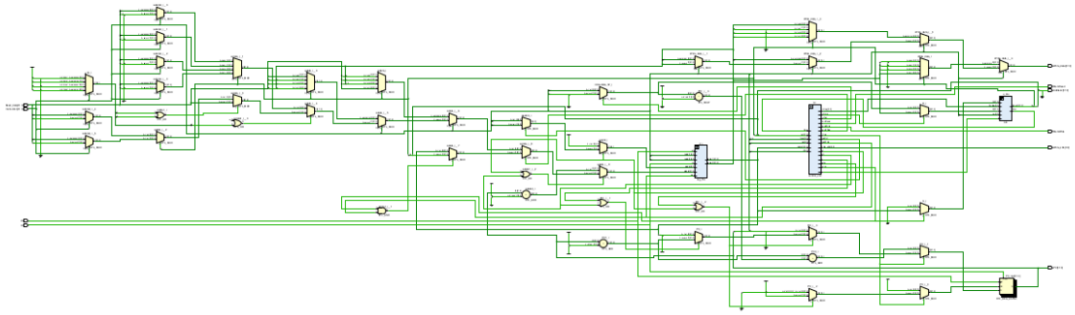
```
//pc
  always @(posedge clk or posedge rst)  begin
  if(rst)
    PC <= 32'd0;
  else  begin
    if(PCsrc)
      PC <= PC + 4 + immediate_32*4;
    else if(jump | jal)
      PC <= {PC[31:28], Instruction[25:0], 2'b00};
    else if(jr | jalr)
      PC <= {rdata1[31:1], 1'b0};
    else
      PC <= PC + 4;
  end
end

always @(*) begin
  if(op == 6'b101010) begin//SWL操作时，ea[1:0]控制内存写数据
    case(ea[1:0])
      2'b00:
        Write_data = {{24{1'b0}}, rdata2[31:24]};
      2'b01:
        Write_data = {{16{1'b0}}, rdata2[31:16]};
      2'b10:
        Write_data = {{8{1'b0}}, rdata2[31:8]};
      2'b11:
        Write_data = rdata2;
    endcase
  end
  else if(op == 6'b101110) begin //SWR操作时，ea[1:0]控制内存写数据
    case(ea[1:0])
      2'b00:
        Write_data = rdata2;
      2'b01:
        Write_data = {rdata2[23:0], {8{1'b0}}};
      2'b10:
```

```
          Write_data = {rdata2[15:0], {16{1'b0}}};
        2'b11:
          Write_data = {rdata2[7:0], {24{1'b0}}};
      endcase
    end
    else if (op==6'b101000)//SB操作时，写数据对齐
      Write_data = {rdata2[7:0], rdata2[7:0], rdata2[7:0], rdata2[7:0]};
    else if(op==6'b101001)//SH操作时，写数据对齐
      Write_data = {rdata2[15:0], rdata2[15:0]};
    else
     Write_data = rdata2;
  end

  //sign extend 1:signed 0:unsigned
  assign immediate_32 = ExtSel ? {{16{immediate[15]}}, immediate[15:0]} :
{16'h0000, immediate[15:0]};

  //regiser file写地址和写数据的赋值
  assign waddr = jal ? 5'b11111 : (RegDst ? rd : rt);

  always @ (*) begin
    if(jal||jalr)
      wdata = PC + 8;
    else if(Mem2Reg)  begin
      if(op == 6'b100010) begin //LWL操作时，ea[1:0]控制寄存器写数据
        case(ea[1:0])
          2'b00:
            wdata = {Read_data[7:0], rdata2[23:0]};
          2'b01:
            wdata = {Read_data[15:0], rdata2[15:0]};
          2'b10:
            wdata = {Read_data[23:0], rdata2[7:0]};
          2'b11:
            wdata = Read_data;
        endcase
```

```
    end
    else if(op == 6'b100110)  begin//LWR操作时，ea[1:0]控制寄存器写数据
      case(ea[1:0])
        2'b00:
          wdata = Read_data;
        2'b01:
          wdata = {rdata2[31:24], Read_data[31:8]};
        2'b10:
          wdata = {rdata2[31:16], Read_data[31:16]};
        2'b11:
          wdata = {rdata2[31:8], Read_data[31:24]};
      endcase
    end
    else if(op == 6'b100000 || op == 6'b100100) begin//LB,LBU操作时，ea[1:0]控制
寄存器写数据
      case(ea[1:0])
        2'b00:
          wdata = (op==6'b100000)?{{24{Read_data[7]}},Read_data[7:0]}:{{24
{1'b0}},Read_data[7:0]};
        2'b01:
          wdata = (op==6'b100000)?{{24{Read_data[15]}},Read_data[15:8]}:{{24
{1'b0}},Read_data[15:8]};
        2'b10:
          wdata = (op==6'b100000)?{{24{Read_data[23]}},Read_data[23:16]}:{{24
{1'b0}},Read_data[23:16]};
        2'b11:
          wdata = (op==6'b100000)?{{24{Read_data[31]}},Read_data[31:24]}:{{24
{1'b0}},Read_data[31:24]};
      endcase
    end
    else if(op==6'b100001||op==6'b100101) begin//LH,LHU操作时，ea[1]控制寄存器写数
据
      case(ea[1])
        1'b0:
          wdata = (op==6'b100001)?{{16{Read_data[15]}},Read_data[15:0]}:{{16
```

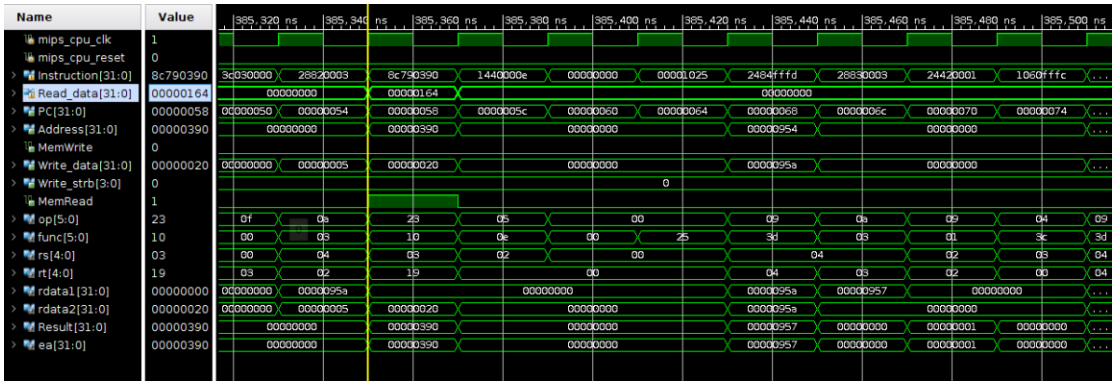
```
{1'b0}},Read_data[15:0]};
          1'b1:
            wdata = (op==6'b100001)?{{16{Read_data[31]}},Read_data[31:16]}:{{16
{1'b0}},Read_data[31:16]};
        endcase
      end
      else
        wdata = Read_data;
    end
    else if(move & RegWrite)
      wdata = rdata1;
    else if(!Mem2Reg)
      wdata = Result;
  end

  //模块之间的连接
  control_unit cu(
      .op(op),
      .func(func),
      .rt(rt),
      .Zero(Zero),
      .ea(ea),
      .rdata1(rdata1),
      .RegDst(RegDst),
      .MemRead(MemRead),
      .Mem2Reg(Mem2Reg),
      .MemWrite(MemWrite),
      .ALUsrc(ALUsrc),
      .RegWrite(RegWrite),
      .PCsrc(PCsrc),
      .jump(jump),
      .jr(jr),
      .jal(jal),
      .jalr(jalr),
      .ExtSel(ExtSel),
      .move(move),
      .v(v),
      .ALUop(ALUop),
      .Write_strb(Write_strb)
      );

  alu u_alu(
      .A(A),
      .B(B),
      .ALUop(ALUop),
      .sa(sa),
      .v(v),
      .Zero(Zero),
      .Result(Result)
      );

  reg_file rf_i(
      .clk(clk),
      .rst(rst),
      .waddr(waddr),
      .raddr1(rs),
      .raddr2(rt),
      .wen(RegWrite),
      .wdata(wdata),
      .rdata1(rdata1),
      .rdata2(rdata2)
      );

endmodule
```

原理图:



3.仿真波形

以 benchmark 的 advanced:04 recursion 的部分仿真波形为例：



recursion 的对应反汇编代码如下：

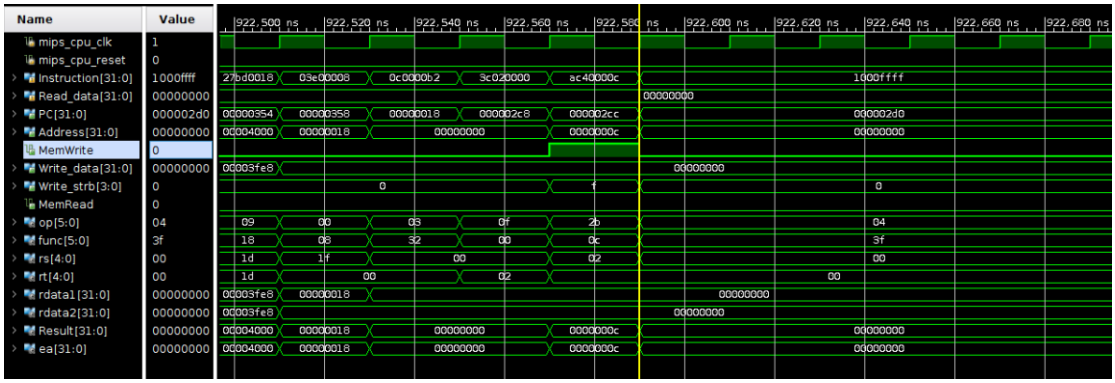
```
00000020 <f0>:
  20:   3c030000        lui     v1,0x0
  24:   8c620394        lw      v0,916(v1)
  28:   0045102a        slt     v0,v0,a1
  2c:   10400002        beqz    v0,38 <f0+0x18>
  30:   00000000        nop
  34:   ac650394        sw      a1,916(v1)
  38:   3c030000        lui     v1,0x0
  3c:   8c620398        lw      v0,920(v1)
  40:   24420001        addiu   v0,v0,1
  44:   ac620398        sw      v0,920(v1)
  48:   18800010        blez    a0,8c <f0+0x6c>
  4c:   00000000        nop
  50:   3c030000        lui     v1,0x0
  54:   28820003        slti    v0,a0,3
  58:   8c790390        lw      t9,912(v1)
  5c:   1440000e        bnez    v0,98 <f0+0x78>
  60:   00000000        nop
  64:   00001025        move    v0,zero
  68:   2484fffd        addiu   a0,a0,-3
  6c:   28830003        slti    v1,a0,3
  70:   24420001        addiu   v0,v0,1
  74:   1060fffc        beqz    v1,68 <f0+0x48>
  78:   00000000        nop
  7c:   24a50001        addiu   a1,a1,1
  80:   00402025        move    a0,v0
  84:   03200008        jr      t9
  88:   00000000        nop
  8c:   24020001        li      v0,1
  90:   03e00008        jr      ra
  94:   00000000        nop
```

从上波形图可以看出 :PC=0x54 时 ,执行 slti 指令 ,A=0x95a, B=0x05, A>B, 故 Result=0;

PC=0x58 时，执行 lw 指令，读使能信号 MemRead=1, 读入数据 ReadData=0x164...

以此类推，其他指令的执行也可以从波形中体现。

下图为行为仿真正确结束的标志：向 0xc 号寄存器写入 0。



下图为三组 benchmark 的上板通过情况：

```
cod@cod-VirtualBox:~/prj2-kyrie2333$ make USER=zhongyun HW_VAL="basic" cloud_run
[sudo] password for cod: Stopping xl2tpd (via systemctl): xl2tpd.service.
/usr/bin/poff: No pppd is running.  None stopped.
Starting xl2tpd (via systemctl): xl2tpd.service.
Remote target: root@172.16.15.66
Warning: Permanently added '172.16.15.66' (ECDSA) to the list of known hosts.
Completed FPGA configuration
Evaluating basic benchmark suite...
Launching memcpy benchmark...
Hit good trap
pass 1 / 1
cod@cod-VirtualBox:~/prj2-kyrie2333$ make USER=zhongyun HW_VAL="medium" cloud_ru
n
Starting xl2tpd (via systemctl): xl2tpd.service.
Remote target: root@172.16.15.66
Warning: Permanently added '172.16.15.66' (ECDSA) to the list of known hosts.
Completed FPGA configuration
Evaluating medium benchmark suite...
Launching sum benchmark...
Hit good trap
Launching mov-c benchmark...
Hit good trap
Launching fib benchmark...
Hit good trap
Launching add benchmark...
Hit good trap
Launching if-else benchmark...
Hit good trap
Launching pascal benchmark...
Hit good trap
Launching quick-sort benchmark...
Hit good trap
Launching select-sort benchmark...
Hit good trap
Launching max benchmark...
Hit good trap
Launching min3 benchmark...
Hit good trap
Launching switch benchmark...
Hit good trap
Launching bubble-sort benchmark...
Hit good trap
pass 12 / 12
```

```
cod@cod-VirtualBox:~/prj2-kyrie2333$ make USER=zhongyun HW_VAL="advanced" cloud_
run
Starting xl2tpd (via systemctl): xl2tpd.service.
Remote target: root@172.16.15.66
Warning: Permanently added '172.16.15.66' (ECDSA) to the list of known hosts.
Completed FPGA configuration
Evaluating advanced benchmark suite...
Launching shuixianhua benchmark...
Hit good trap
Launching sub-longlong benchmark...
Hit good trap
Launching bit benchmark...
Hit good trap
Launching recursion benchmark...
Hit good trap
Launching fact benchmark...
Hit good trap
Launching add-longlong benchmark...
Hit good trap
Launching shift benchmark...
Hit good trap
Launching wanshu benchmark...
Hit good trap
Launching goldbach benchmark...
Hit good trap
Launching leap-year benchmark...
Hit good trap
Launching prime benchmark...
Hit good trap
Launching mul-longlong benchmark...
Hit good trap
Launching load-store benchmark...
Hit good trap
Launching to-lower-case benchmark...
Hit good trap
Launching movsx benchmark...
Hit good trap
Launching matrix-mul benchmark...
Hit good trap
Launching unalign benchmark...
Hit good trap
pass 17 / 17
```

二、 实验过程中遇到的问题、对问题的思考过程及解决方法

1. 由于对单周期 cpu 的结构不是十分熟悉，前期的许多 bug 出现在控制单元信号赋值错误上。Debug 过程很艰难，主要通过逐条检查指令的控制信号，以及和同学比对仿真波形。但是后期在助教张旭给了仿真测试环境后，debug 过程变高效了许多。
2. Debug 过程中遇到的一些错误集锦：在写 medium 部分的 jal 指令时，向往 31 号寄存器回写跳转前地址时，只把地址赋给了 wdata，没有写赋给哪个寄存器，应该同时把 31 赋给 waddr；对于 addi, ori, addiu 等指令，想当然地把符号扩展写成无符号的（ExtSel 置 0），实际上应扩展为有符号的（ExtSel 置 1），应该仔细阅读

mips 手册；SB，SH 指令操作需要字节对齐；srlv, sllv 指令第一个操作数 A 只取低五位…

3. 在写 advanced 组指令时，生成了组合电路环。由于第三阶段增加了一些指令，需要对 wdata 和 alu 的第二个操作数增加更多的控制信号，于是产生了 ALUop→Result→wdata→ALUop 的电路环，解决方法是将原本写在 alu 中的对操作数 B 的控制信号移到 mips_cpu 模块中，同时减少一些指令对 alu 通路的使用，如：将原本利用 alu 通路的 bgez, bltz, blez 指令改成通过直接判断 rdata1 来控制 PC 的跳转，这样就减少了一个控制 alu 中操作数 B 的信号。

三、 对于此次实验的心得、感受和建议

本次实验在思考层面上并不是十分困难，需要代码部分需要做到结构严谨，语法规范。但是在实现层次上有一定难度，主要在于内容繁多，debug 过程比较艰难，最后助教张旭给出的测试环境有很大的帮助。同时感谢同学侯承轩、张林隽和杨依涵在共同讨论中给予的帮助！