Lab3 单周期无流水MIPS CPU设计

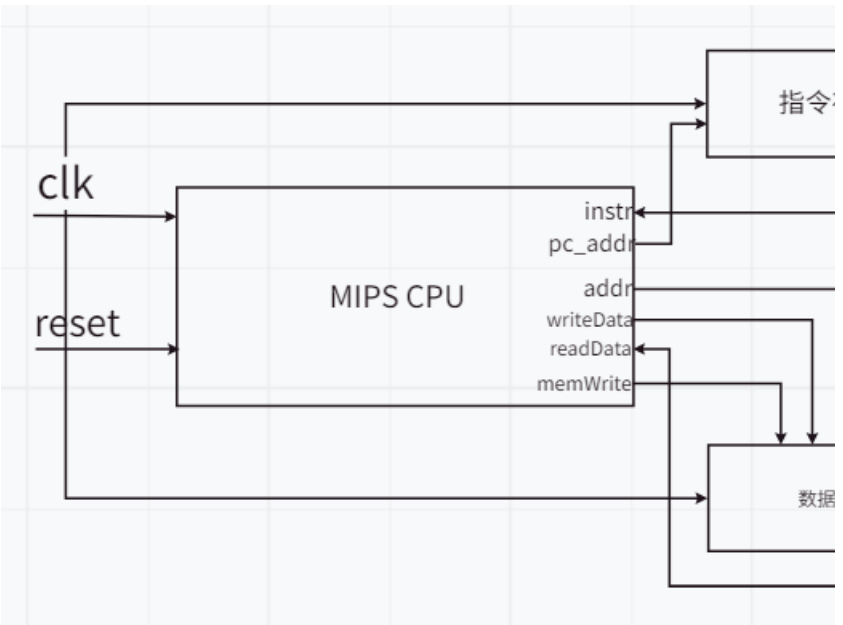
实验要求

实验文件树

├─top.v	设计顶层文件,已提供。
├──mips.v	MIPS 软核顶层文件,将 Controller 与 Datapath 连接,
├────controller.v	控制器模块,本次实验重点。
├────maindec.v	Main decoder 模块,负责译码得到各个组件的控制信
├────aludec.v	ALU Decoder 模块,负责译码得到 ALU 控制信号。
├────datapath.v	数据通路模块,自行实现
├────pc.v	PC 模块,使用实验二代码
├────alu.v	ALU 模块,使用实验一代码
├────sl2.v	移位模块,参考《其他组件实现.pdf》
├────signext.v	有符号扩展模块,参考《其他组件实现.pdf》
├────mux2.v	二选一选择器,自行实现
├────regfile.v	寄存器堆,已提供
├────adder.v	加法器,已提供
├──*inst_ram.ip*	*RAM IP*,通过 *Block memory generator* 进行实例化
├──*data_ram.ip*	*RAM IP*,通过 *Block memory generator* 进行实例化

表 1: 实验文件树

左图的实验文件树就是Verilog自顶向下的设计思想的体现，如下：



无流水线单周期MIPS CPU框架图

- 顶层

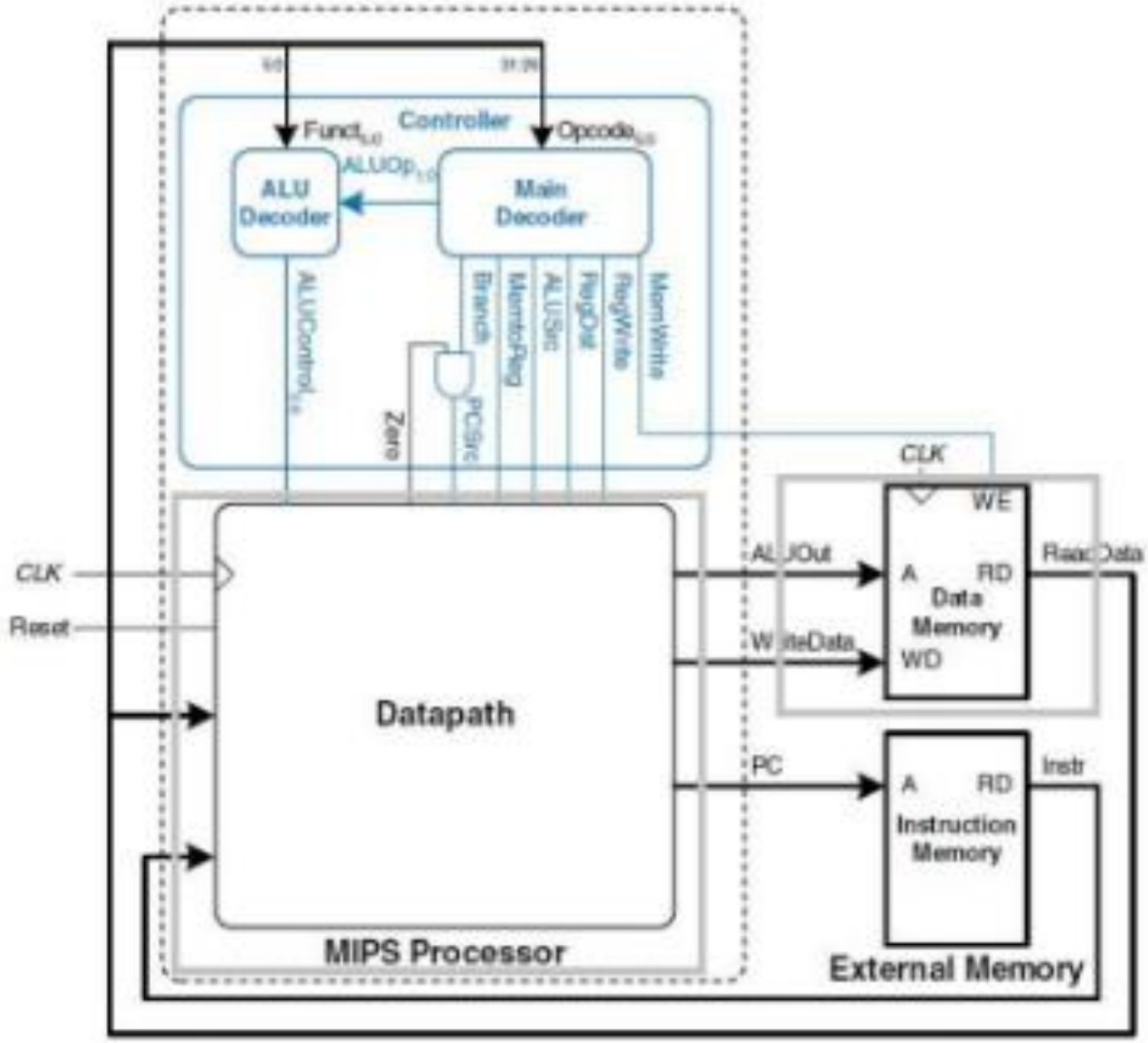
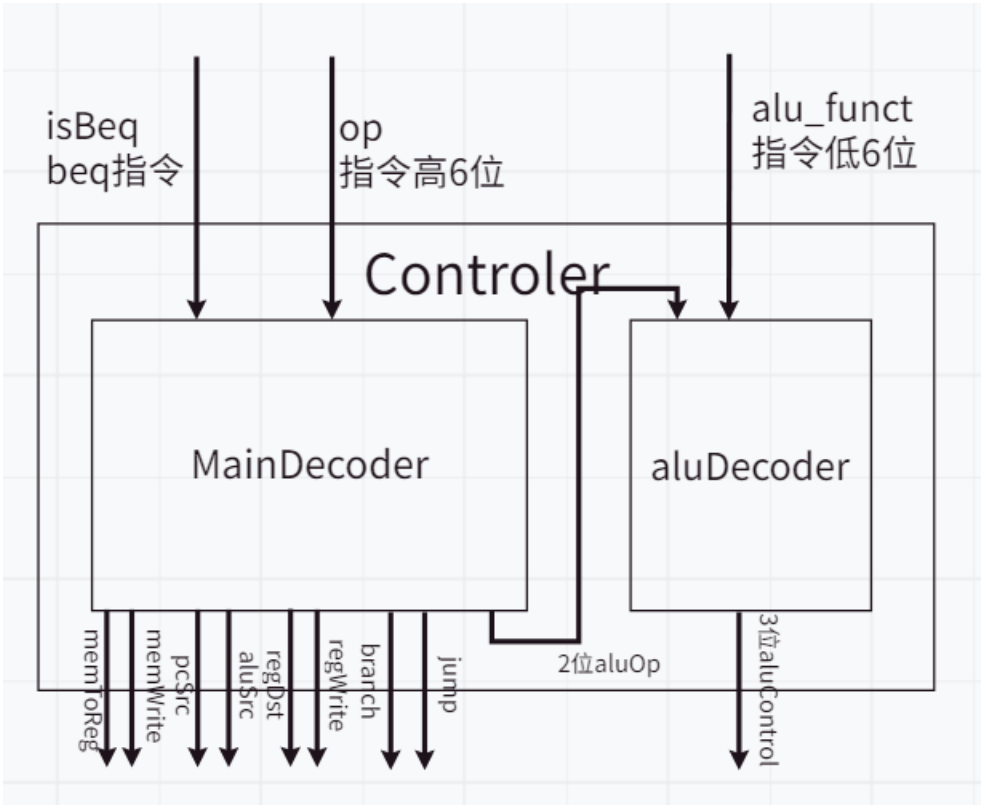


图 1: 单周期 CPU 框架图

- 控制层



- 数据通路

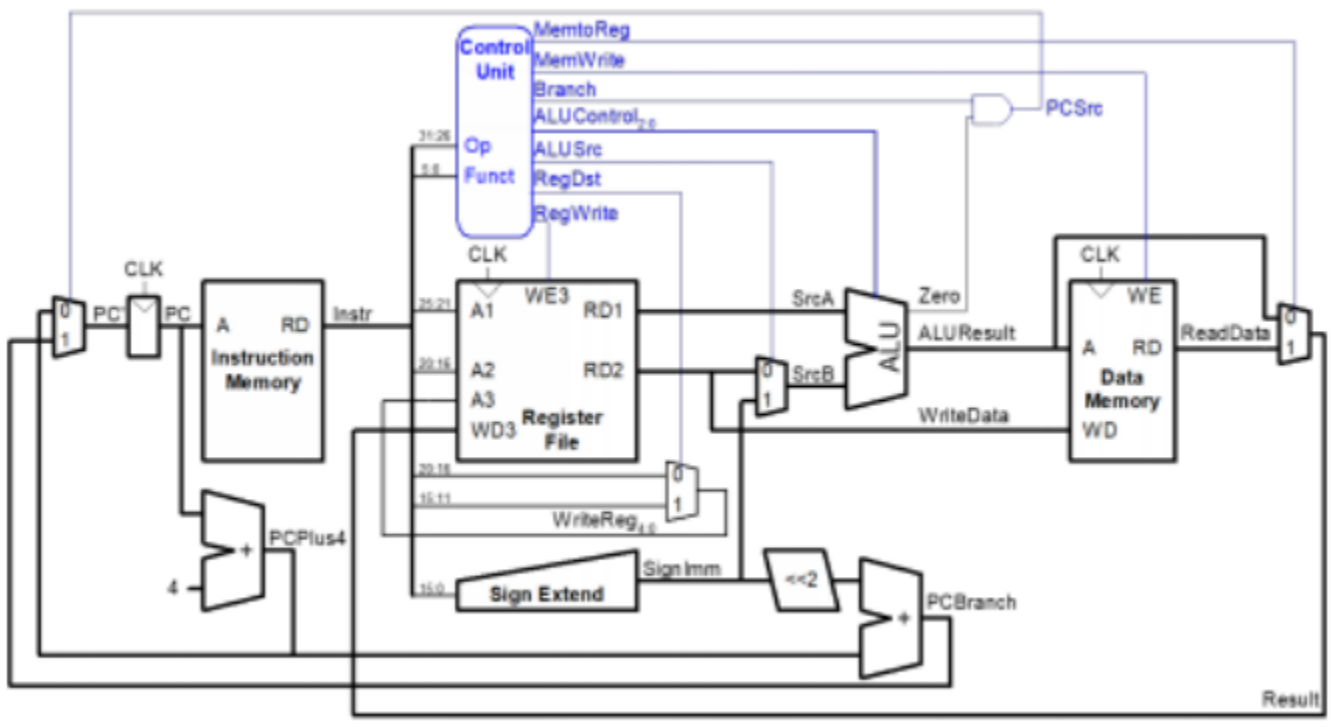


图 2: 单周期 CPU 框架图

上图是无Jump指令的，结合jump指令的数据通路更改如下图：

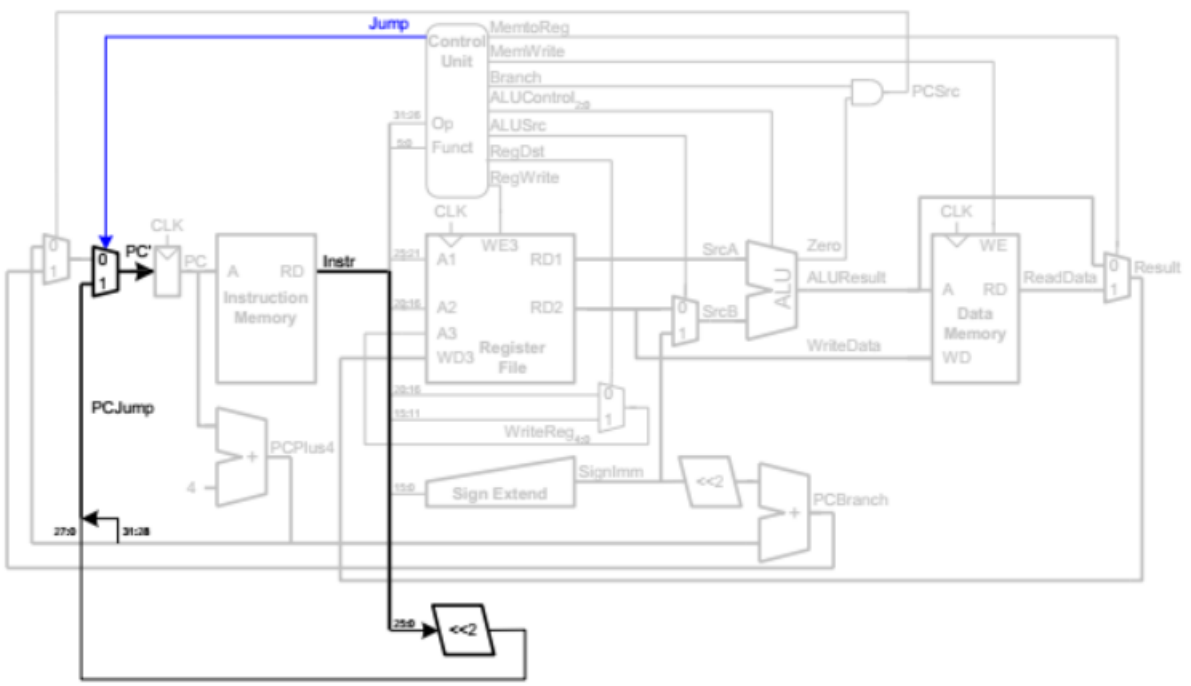


图 13

由“无流水线单周期MIPS CPU框架图”可得，采用**自顶向下**的方法进行设计：

top顶层模块包括：**mips模块、insMem、dataMem模块**

mips模块是**controler和datapath的集成**

controler模块之前已实现，是**mainDecoder和aluDecoder的集成**

datapath模块包括了**pc、signExtend、regFile、alu、mux2to1、shl2、adder**模块

数据通路的分析

首先是一个*二选一多选器*（输入pcPlus4和pcBranch，在pcsrc选择➡0plus1branch），然后选择的结果作为相连的*二选一多选器*的0端输入，1端接{pcPlus4[31:28],inst[25:0],2'b00}，在jump信号下的选择的结果经*pc*传送给指令存储器*insRom*，*pc*的输出结果也通过*adder*加4得到pcPlus4。inst[25:21],inst[20:16]则作为寄存器堆*regFile*的读端口，写端口是*二选一选择器*的结果（输入inst[20:16]和inst[15:11]，选择信号regdst➡0前1后），写寄存器的结果也是一个*二选一选择器*的结果（输入是aluResult和memReadResult，选择信号memToReg➡0前1后）。而inst[15:0]则送*signExtend*进行立即数扩展，扩展结果SignImm先通过*shl2*左移两位，再通过*adder*加上pcPlus4得到pcBranch。寄存器读端口[25:21]对应的立即数直接送*ALU*端口A，而B端口的操作数则是*二选一选择器*的结果（输入是[20:16]读出的结果、signImm，选择信号alusrc➡0前1后），然后在aluControl的控制下进行对应的alu运算。（这里若是beq指令，则对操作数执行减法，判断结果是否为0的zero送控制器产生pcsrc）（若是load/store指令，则ALU运算得到的aluResult是访存地址送dataRAM，store指令的写结果来自[20:16]读到的结果）

> 绿色表示器件、黄色表示数据通路与外部的输入输出、蓝色表示数据通路内部之间器件的数据传递

模块实现

- pc

```Verilog
`timescale 1ns / 1ps

module pc(
    input rst,
    input clk,

    input [31:0]i_addr,
    output reg [31:0]o_addr
);
    always @(posedge clk) begin
        if (~rst) begin
            o_addr=0;
        end else begin
            o_addr=i_addr;
        end
    end
endmodule
```

- adder

```Verilog
`timescale 1ns / 1ps

module adder(
    input [31:0]A,
    input [31:0]B,

    output [31:0]res
);
```

```
    assign res=A+B;
endmodule
```

Verilog

- mainDecoder

```
`timescale 1ns/1ps

module mainDecoder(
    input [5:0]op,
    input zero,

    output  memtoreg,//写寄存器的结果来自mem还是alu
    output  memwrite,//是否写存储
    output  pcsrc,//下一个PC的值是PC+4还是跳转后的地址0表示src, 1表示跳转——由branch
    output  alusrc,//aluB的源操作数是立即数还是寄存器读出的0表示寄存器1表示立即数
    output  regdst,//写入寄存器堆的地址是rt还是rd,0是rt, 1是rd
    output  regwrite,//是否写寄存器
    output  branch,//是否是分支指令, 且满足分支条件
    output  jump,//是否是无跳转指令
    output  [1:0] aluop//alu控制
);
    wire [8:0]control;
    assign control = (op==6'b000000)?9'b000110010:
                     (op==6'b100011)?9'b101010000:
                     (op==6'b101011)?9'bx11x00000:
                     (op==6'b000100)?9'bx00x01001:
                     (op==6'b001000)?9'b001010000:
                     (op==6'b000010)?9'bx0xx0x1xx:9'bxxxxxxxxx;
```

```
    assign {memtoreg,memwrite,alusrc,regdst,regwrite,branch,jump}=control[8:2];
    assign aluop=control[1:0];
    assign pcsrc=branch&zero;

endmodule
```

Verilog

- aluDecoder

```
`timescale 1ns / 1ps

module alu_control(
    input [1:0]aluop,
    input [5:0]alu_func,

    output reg[2:0]control
);

    always @(*) begin
        case (aluop)
            2'b00:control=3'b010;
            2'b01:control=3'b110;
            2'b10:begin
                case (alu_func)
                    6'b100000: control=3'b010;//ADD
                    6'b100010: control=3'b110;//SUB
                    6'b100100: control=3'b000;//AND
                    6'b100101: control=3'b001;//OR
                    6'b101010: control=3'b111;//SLT
                    default: control=3'bxxx;
                endcase
            end
            default:  control=3'bxxx;
```

```
        endcase
    end
endmodule
```

Verilog

- insMem、dataMem

insMem:

```
memory_initialization_radix = 16;
memory_initialization_vector =20020005,2003000c,2067fff7,00e22025,
00642824,00a42820,10a7000c,0064202a,10800002,00000000,20050000,00e2202a,00853820,00e23822,ac670044,8c020050,08000013,00000000,20020001,ac020054,
```

纯文本

dataMem:

```
memory_initialization_radix=16;
memory_initialization_vector=1,2,3,4,5,6,7,8,9,a,b,c,d,e,f;
```

text

- signExtend

```
`timescale 1ns/1ps

module signExtend(
    input [15:0] num,
    output [31:0] o_num
);
    assign o_num={{16{num[15]}},num};
endmodule
```

Verilog

regFile

```
`timescale 1ns/1ps

module regFile(
    input         clk,
    // READ PORT 1
    input  [ 4:0] raddr1,
    output [31:0] rdata1,
    // READ PORT 2
    input  [ 4:0] raddr2,
    output [31:0] rdata2,
    // WRITE PORT
    input         we,       //write enable, HIGH valid
    input  [ 4:0] waddr,
    input  [31:0] wdata
);
    reg [31:0] rf[31:0];

    //WRITE
    always @(posedge clk) begin
        if (we) begin
            rf[waddr]=wdata;
        end
    end

    //READ OUT 1
    assign rdata1 = (raddr1==5'b0) ? 32'b0 : rf[raddr1];//x0为0

    //READ OUT 2
```

```
    assign rdata2 = (raddr2==5'b0) ? 32'b0 : rf[raddr2];//x0为0
endmodule
```

Verilog

- alu

```
`timescale 1ns/1ps

module alu(
    input [2:0]alu_control,

    input [31:0]A,
    input [31:0]B,

    output zero,
    output [31:0]aluResult
);
    assign aluResult=(alu_control==3'b010)?A+B:
                     (alu_control==3'b110)?A-B:
                     (alu_control==3'b000)?A&B:
                     (alu_control==3'b001)?A|B:
                     (alu_control==3'b111)?A<B:32'bx;
    assign zero=(aluResult==0)?1:0;
endmodule
```

Verilog

- mux2to1

```
`timescale 1ns / 1ps

module mux2to1(
    input [31:0]A,
    input [31:0]B,
```

```
    input sel,

    output[31:0]res
);
    assign res=(sel==0)?A:B;//0选A, 1选B
endmodule
```

Verilog

- shl2

```
`timescale 1ns/1ps

module shl2(
    input [31:0]A,
    output [31:0]res
);
    assign res={A[29:0],2'b0};
endmodule
```

Verilog

- datapath

```
`timescale 1ns/1ps
module datapath(
    input clk,
    input rst,

    input [31:0]readData,
    input [31:0]instr,

    input  memtoreg,//写寄存器的结果来自mem还是alu
    input  pcsrc,//下一个PC的值是PC+4还是跳转后的地址0表示src, 1表示跳转——由branch
```

```
    input  alusrc,//aluB的源操作数是立即数还是寄存器读出的0表示寄存器1表示立即数
    input  regdst,//写入寄存器堆的地址是rt还是rd,0是rt, 1是rd
    input  regwrite,//是否写寄存器
    input  jump,//是否是无跳转指令
    input  [2:0]control,

    output zero,
    output [31:0]addr,
    output [31:0]writeData,
    output [31:0]aluOut
);
    wire [31:0]i_addr;//输入给PC的地址
    wire [31:0]pc_plus4;//PC+4后的地址
    wire [31:0]j_addr;
    wire [31:0]branch_mux_addr;
    wire [31:0]pc_branch;
    mux2to1  mux2to1_branch_mux_addr (
        .A(pc_plus4),
        .B(pc_branch),
        .sel(pcsrc),
        .res(branch_mux_addr)
    );

    mux2to1 mux2to1_pcNext(
        .A(branch_mux_addr),
        .B({pc_plus4[31:28],instr[25:0],2'b00}),
        .sel(jump),
        .res(i_addr)
    );
    pc  pc_inst (
```

```
        .rst(rst),
        .clk(clk),
        .i_addr(i_addr),
        .o_addr(addr)
    );
    adder  adder_inst_pcPlus4 (
        .A(addr),
        .B(32'h4),
        .res(pc_plus4)
    );

    wire [5:0]w_regAddr;
    wire [31:0]srcA;
    wire [31:0]tempB;
    wire [31:0]imm32;
    wire [31:0]srcB;
    wire [31:0]result;
    wire [31:0]pc_imm;
    mux2to1  mux2to1_regWAddr (
        .A(instr[20:16]),
        .B(instr[15:11]),
        .sel(regdst),
        .res(w_regAddr)
    );
    regFile  regFile_inst (
        .clk(clk),
        .raddr1(instr[25:21]),
        .rdata1(srcA),
        .raddr2(instr[20:16]),
        .rdata2(tempB),
        .we(regwrite),
        .waddr(w_regAddr),
        .wdata(result)
```

```
    );
    signExtend  signExtend_inst (
        .num(instr[15:0]),
        .o_num(imm32)
    );
    mux2to1  mux2to1_SrcB (
        .A(tempB),
        .B(imm32),
        .sel(alusrc),
        .res(srcB)
    );
    shl2  shl2_inst (
        .A(imm32),
        .res(pc_imm)
    );
    adder  adder_inst_pcBranch (
        .A(pc_imm),
        .B(pc_plus4),
        .res(pc_branch)
    );

    alu  alu_inst (
        .alu_control(control),
        .A(srcA),
        .B(srcB),
        .zero(zero),
        .aluResult(aluOut)
    );
    mux2to1  mux2to1_regWData (
        .A(aluOut),
        .B(readData),
        .sel(memtoreg),
        .res(result)
```

```
    );
    assign writeData=tempB;
endmodule
```

Verilog

controler

```
`timescale 1ns/1ps

module controler(
    input [5:0]op,alu_funct,
    input isZero,

    output  memtoreg,//写寄存器的结果来自mem还是alu
    output  memwrite,//是否写存储
    output  pcsrc,//下一个PC的值是PC+4还是跳转后的地址0表示src, 1表示跳转——由branch
    output  alusrc,//aluB的源操作数是立即数还是寄存器读出的0表示寄存器1表示立即数
    output  regdst,//写入寄存器堆的地址是rt还是rd,0是rt, 1是rd
    output  regwrite,//是否写寄存器
    output  jump,//是否是无跳转指令
    output  [2:0]control
);
    wire [1:0]aluop;
    wire branch;

    mainDecoder  mainDecoder_inst (
        .op(op),
        .zero(isZero),
        .memtoreg(memtoreg),
        .memwrite(memwrite),
```

```
        .pcsrc(pcsrc),
        .alusrc(alusrc),
        .regdst(regdst),
        .regwrite(regwrite),
        .branch(branch),
        .jump(jump),
        .aluop(aluop)
    );

    aluDecoder  aluDecoder_inst (
        .aluop(aluop),
        .alu_func(alu_funct),
        .control(control)
    );
endmodule
```

Verilog

▿ mips

```
`timescale 1ns/1ps

module mips(
    input clk,
    input rst,

    input [31:0]readData,
    input [31:0]instr,

    output memwrite,
    output [31:0]addr,
    output [31:0]aluOut,
    output [31:0]writeData
```

```
);
    wire  memtoreg;//写寄存器的结果来自mem还是alu
    wire  pcsrc;//下一个PC的值是PC+4还是跳转后的地址0表示src, 1表示跳转——由branch
    wire  alusrc;//aluB的源操作数是立即数还是寄存器读出的0表示寄存器1表示立即数
    wire  regdst;//写入寄存器堆的地址是rt还是rd,0是rt, 1是rd
    wire  regwrite;//是否写寄存器
    wire  jump;//是否是无跳转指令
    wire  [2:0]control;
    wire  zero;

    datapath  datapath_inst (
        .clk(clk),
        .rst(rst),
        .readData(readData),
        .instr(instr),
        .memtoreg(memtoreg),
        .pcsrc(pcsrc),
        .alusrc(alusrc),
        .regdst(regdst),
        .regwrite(regwrite),
        .jump(jump),
        .control(control),
        .zero(zero),
        .addr(addr),
        .writeData(writeData),
        .aluOut(aluOut)
    );
    controler  controler_inst (
        .op(instr[31:26]),
        .alu_funct(instr[5:0]),
```

```
        .isZero(zero),
        .memtoreg(memtoreg),
        .memwrite(memwrite),
        .pcsrc(pcsrc),
        .alusrc(alusrc),
        .regdst(regdst),
        .regwrite(regwrite),
        .jump(jump),
        .control(control)
    );
endmodule
```

Verilog

▿ top

```
`timescale 1ns/1ps
module top(
    input clk,
    input rst
);
    wire memwrite;
    wire [31:0]addr,aluOut,writeData,readData,instr;

    mips  mips_inst (
        .clk(clk),
        .rst(rst),
        .readData(readData),
        .instr(instr),
        .memwrite(memwrite),
        .addr(addr),
        .aluOut(aluOut),
        .writeData(writeData)
    );
```

```
    blk_mem_gen_0 insRom(
        .clka(clk),
        .addra(addr),
        .douta(instr)
    );
    blk_mem_gen_1 dataRam(
        .clka(clk),
        .wea(memwrite),
        .addra(aluOut),
        .dina(writeData),
        .douta(readData)
    );
endmodule
```

Verilog

模块测试

```
`timescale 1ns / 1ps


module top_tb;

    // Parameters

    //Ports
    reg  clk;
    reg  rst;

    initial begin
        clk=0;
        rst=0;
        #100;
        rst=1;
```

```
    end

    top  top_inst (
    .clk(clk),
    .rst(rst)
    );

    always #10 clk = ! clk ;

endmodule
```

Verilog

波形图文件如下：