《计算机组成原理》实验报告

年级、专业、班级		**姓名**	
实验题目	实验三简易单周期 CPU 实验		
实验时间		**实验地点**	
实验成绩	优秀/良好/中等	**实验性质**	□验证性 ☑设计性 □综合性
教师评价： □算法/实验过程正确； □源程序/实验内容提交； □程序结构/实验步骤合理； □实验结果正确； □语法、语义正确； □报告规范； 其他: 评价教师:			
实验目的 (1)掌握不同类型指令在数据通路中的执行路径。 (2)掌握 Vivado 仿真方式。			

报告完成时间: 2024 年 7 月 21 日

1 实验内容

阅读实验原理实现以下模块：

(1) Datapath，其中主要包含 alu(实验一已完成)，PC(实验二已完成)，adder、mux2、signext、sl2(其中 adder、mux2 数字逻辑课程已实现，signext、sl2 参见实验原理)，
(2) Controller(实验二已完成)，其中包含两部分，分别为 main_decoder，alu_decoder。
(3) 指令存储器 inst_mem(Single Port Ram)，数据存储器 data_mem(Single Port Ram)；使用 Block Memory Generator IP 构造指令，注意考虑 PC 地址位数统一。(参考实验二)
(4) 参照实验原理，将上述模块依指令执行顺序连接。实验给出 top 文件，需兼容 top 文件端口设定。
(5) 实验给出仿真程序，最终以仿真输出结果判断是否成功实现要求指令。

2 实验设计

2.1 数据通路

2.1.1 功能描述

Datapath(数据通路) 模块综合了单周期 CPU 中除 Controller(控制器),Instruction Memory(指令存储器) 和 Data Memory(数据存储器) 以外的所有功能模块和总线，实现了 CPU(处理器) 中的算术运算和数据传输功能。

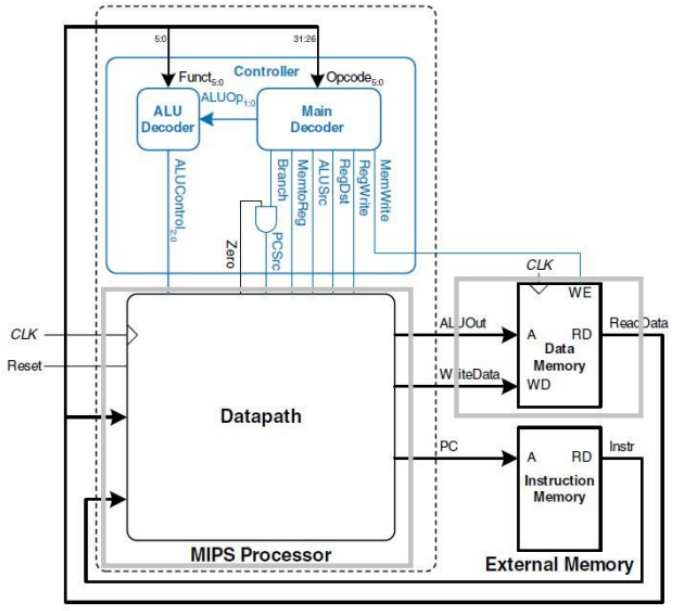

图 1: Datapath

2.1.2 接口定义

表 1: Datapath 接口定义

信号名	方向	位宽	功能描述
clk	Input	1-bit	时钟信号
rst	Input	1-bit	复位信号
Branch	Input	1-bit	分支控制信号
Jump	Input	1-bit	跳转控制信号
Inst	Input	26-bit	指令的 25-0 位
RegWrite	Input	1-bit	寄存器堆写使能信号
RegDst	Input	1-bit	写入目标寄存器选择信号.
MemtoReg	Input	1-bit	写入寄存器数据来源选择信号
ReadData	Input	32-bit	数据存储器读出的数据
AluSrc	Input	1-bit	ALU 第二操作数来源选择信号
ALUControl	Input	3-bit	ALU 控制信号
PC	Output	32-bit	当前指令地址信号
Inst_ce	Output	1-bit	指令存储器使能信号
ReadData2	Output	32-bit	寄存器堆的第二个输出
ALUResult	Output	32-bit	ALU 运算结果

3 实验过程记录

3.1 Datapath

3.1.1 问题 1:lw 指令需要分别进行数据存储器读操作和寄存器堆写操作, 就时钟策略而言本应无法在一个时钟周期内完成

问题描述: 在共享同一个时钟信号的前提下, 数据存储器读和寄存器堆写应当需要经过两个连续的上升沿或下降沿才能完成 lw 操作。

解决方案: 经测试, 数据存储器在读使能信号有效时等效为组合逻辑, 在写使能信号有效时才为时序逻辑。

3.1.2 问题 2: 丢失指令指令存储器中的最后一条指令

问题描述: 仿真图中看不到 mips.asm 文件中的最后一条有效指令。

解决方案: 由于 PC 模块中包含有 PC+4 功能, 因此 PC 模块应当有两个输出 (PC/PC+4), 对

PC 模块进行对应修改即可。

3.1.3 问题 3:PC 寄存器在非分支指令写入了分支目标地址

问题描述: Mux(多路选择器) 在非分支指令时选择了分支地址加法器的输出结果。
解决方案:修改”Branch&Zero==0” 为”(Branch&Zero)==0”。

3.1.4 问题 4: 第一条指令的寄存器堆输出为 X

问题描述:第一条指令的寄存器堆输出为空, 即指定寄存器中没有数据。
解决方案:修改”assign ReadData1=Registers[ReadReg1]” 为”assign ReadData1=(ReadReg1==0)?0:Registers[ReadReg1]”, 即将 0 号寄存器的输出恒置为 0。

4 实验结果及分析

4.1 仿真图

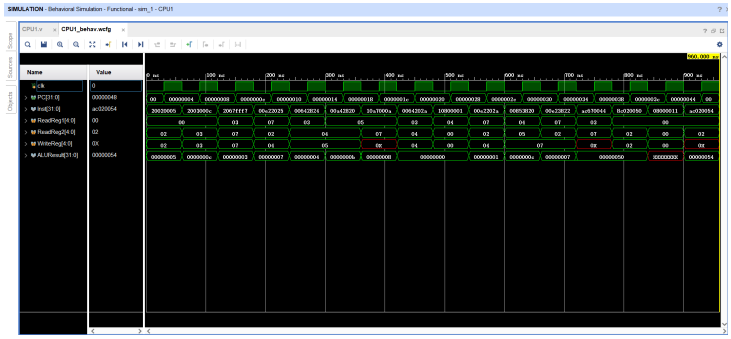

图 2: 仿真图

4.2 控制台输出

图 3: 控制台输出

A CPU1 代码

```
module CPU1();
    reg clk;
    reg rst;

    wire [31:0] Inst;
    wire [31:0] ReadData;
    wire [31:0] PC;
    wire Inst_ce;
    wire MemWrite;
    wire MemRead;
    wire [31:0] ALUResult;
    wire [31:0] ReadData2;

    always #30 clk=~clk;
    initial
    begin
        clk<=0;
        rst<=0;
    end

  always @(negedge clk) begin
    if(MemWrite) begin
      /* code */
      if(ALUResult== 84 & ReadData2 == 7) begin
        /* code */
        $display("Simulation succeeded");
        $stop;
      end else if(ALUResult !== 80) begin
        /* code */
        $display("Simulation Failed");
        $stop;
      end
    end
  end

    MIPS myMIPS(clk,rst,Inst,ReadData,PC,Inst_ce,MemWrite,MemRead,ALUResult,
        ReadData2);
    InstMem myInstMem(.clka(~clk),.ena(Inst_ce),.addra(PC),.douta(Inst));
    DataMem myDataMem(.clka(~clk),.ena(MemWrite|MemRead),.rsta(rst),.wea({4{
        MemWrite}}),.addra(ALUResult),.dina(ReadData2),.douta(ReadData));
endmodule
```

B MIPS 代码

```
module MIPS(clk,rst,Inst,ReadData,PC,Inst_ce,MemWrite,MemRead,ALUResult,ReadData2);
```

```
    input clk;
    input rst;//PC
    input [31:0] Inst;//InstMem
    input [31:0] ReadData;//DataMem

    output [31:0] PC;//PC
    output Inst_ce;
    output MemWrite;//DataMem
    output MemRead;
    output [31:0] ALUResult;
    output [31:0] ReadData2;

    wire [5:0] op,Funct;//Controller
    wire RegWrite;
    wire RegDst;
    wire AluSrc;
    wire Branch;
    wire MemtoReg;
    wire Jump;
    wire [2:0] ALUControl;

    assign op=Inst[31:26];//Controller
    assign Funct=Inst[5:0];

    Controller myController(op,Funct,RegWrite,RegDst,AluSrc,Branch,MemWrite,
        MemtoReg,MemRead,Jump,ALUControl);
    Datapath myDatapath(clk,rst,Branch,Jump,Inst[25:0],RegWrite,RegDst,MemtoReg,
        ReadData,AluSrc,ALUControl,PC,Inst_ce,ReadData2,ALUResult);
endmodule
```

C Datapath 代码

```
module Datapath(clk,rst,Branch,Jump,Inst,RegWrite,RegDst,MemtoReg,ReadData,AluSrc,
    ALUControl,PC,Inst_ce,ReadData2,ALUResult);
    input clk;
    input rst;//PC
    input Branch;
    input Jump;
    input [25:0] Inst;//Instruction
    input RegWrite;//Regfile
    input RegDst;
    input MemtoReg;
    input [31:0] ReadData;
    input AluSrc;//ALU
    input [2:0] ALUControl;

    output [31:0] PC;//PC
```

```
    output Inst_ce;
    output [31:0] ReadData2;//Regfile
    output [31:0] ALUResult;//ALU

    wire [31:0] PC_in;//PC
    wire [31:0] PCa4;
    wire [31:0] Branch_Addr;
    wire [31:0] Mux_ifbranch;
    wire [4:0] ReadReg1,ReadReg2,WriteReg;//Regfile
    wire [31:0] WriteData;
    wire [31:0] ReadData1;
    wire [31:0] SignExtend;//SignExtend
    wire [31:0] A,B;//ALU
    wire Zero;

    assign Branch_Addr=PCa4+{SignExtend[30:0],2'b00};//PC
    assign Mux_ifbranch=((Branch&Zero)==0)?PCa4:Branch_Addr;
    assign PC_in=(Jump==0)?Mux_ifbranch:{PCa4[31:28],Inst,2'b00};
    assign ReadReg1=Inst[25:21];//Regfile
    assign ReadReg2=Inst[20:16];
    assign WriteReg=(RegDst==0)?Inst[20:16]:Inst[15:11];
    assign WriteData=(MemtoReg==0)?ALUResult:ReadData;
    assign SignExtend={{16{Inst[15]}},Inst[15:0]};//SignExtend
    assign A=ReadData1;//ALU
    assign B=(AluSrc==0)?ReadData2:SignExtend;

    PC myPC(clk,rst,PC_in,PC,PCa4,Inst_ce);
    Regfile myRegfile(~clk,RegWrite,ReadReg1,ReadReg2,WriteReg,WriteData,ReadData1,
        ReadData2);
    ALU myALU(ALUControl,A,B,ALUResult,Zero);
endmodule
```

D PC 代码

```
module PC(clk,rst,PC_in,PC,PCa4,Inst_ce);
    input clk;
    input rst;
    input [31:0] PC_in;
    output reg [31:0] PC=32'b0;
    output [31:0] PCa4;
    output Inst_ce;

    assign Inst_ce=1'b1;
    assign PCa4=PC+4;

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

```
        if(rst)
            PC<=32'b0;
        else PC<=PC_in;
    end
endmodule
```

E Regfile 代码

```
module Regfile(clk,RegWrite,ReadReg1,ReadReg2,WriteReg,WriteData,ReadData1,
    ReadData2);
    input clk;
    input RegWrite;
    input [4:0] ReadReg1,ReadReg2,WriteReg;
    input [31:0] WriteData;
    output [31:0] ReadData1,ReadData2;

    reg [31:0] Registers [31:0];
    always@(posedge clk)
    begin
        if(RegWrite)
            Registers[WriteReg]<=WriteData;
    end

    assign ReadData1=(ReadReg1==0)?0:Registers[ReadReg1];
    assign ReadData2=(ReadReg2==0)?0:Registers[ReadReg2];
endmodule
```

F ALU 代码

```
module ALU(ALUControl,A,B,ALUResult,Zero);
    input [2:0] ALUControl;
    input [31:0] A;
    input [31:0] B;
    output [31:0] ALUResult;
    output Zero;

    assign ALUResult=(ALUControl==3'b000)?A&B:
                     (ALUControl==3'b001)?A|B:
                     (ALUControl==3'b010)?A+B:
                     (ALUControl==3'b110)?A-B:
                     (ALUControl==3'b111)?A<B:32'b0;
    assign Zero=(ALUResult==32'b0);
endmodule
```