《计算机组成原理》实验报告

年级、专业、班级	2021 级计算机科学与技术 04、05 班	**姓名**	胡鑫, 冯宇馨
实验题目	实验三简易单周期 CPU 实验		
实验时间	2023 年 4 月 23 日	**实验地点**	第一实验楼 DS1410
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: 冯永

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: 2023 年 5 月 18 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

简易单周期 CPU，可实现整数算术逻辑运算，存取指令和跳转指令的在数据通路中的执行。

2.1.2 接口定义

表 1: 接口定义

信号名	方向	位宽	功能描述
memtoreg	input	1-bit	高电平时将取数指令的取数地址对应的存储器数据写入到寄存器堆 rt 寄存器,否则低电平
regdst	input	1-bit	高电平时写寄存器的目标寄存器号来自 rd（15:11），低电平时来自 rt（20:16）
regwrite	input	1-bit	高电平时指令写入寄存器堆,否则低电平
alusrc	input	1-bit	高电平时 ALU 第二个操作数来自指令低 16 位符号扩展，低电平时来自寄存器堆第二个输出
branch	input	1-bit	条件分支指令执行后,条件成立则执行 PC 相对寻址,信号为 1 否则为 0
jump	input	1-bit	J 型指令(无条件跳转)时为 1
alucontrol	input	3-bit	ALU 控制信号，由 funct(5:0) 和 opcode(31:26)共同决定
instr	input	32-bit	从 inst_ram 中读取的 32 位 MIPS 指令
readdata	input	32-bit	c 用数据存储器读出，输入 datapath 读入 regfile
pc	output	32-bit	程序计数器,每个时钟上升沿自增 4,指向下一条指令
aluout	output	32-bit	数值为 ALU 的结果,当 memwrite 有效时为写入存储器地址
writedata	output	32-bit	regfile 取出的 RD@ 操作数，写入数据存储器,只用于 sw 指令

3 实验过程记录

3.1 实验设计

3.1.1 datapath 设计

1. datapath 主要分为以下部分:pc 模块, 寄存器堆, 控制块, alu 和多路选择器、移位器、符号扩展器件等。
2. 由于 pc 涉及到分支跳转指令的实现, 需要对多个模块进行组合, 同时使用中间寄存器变量的形式来组合多个模块, 从而完成分支跳转功能。
3. controllor, alu, pc 使用前两次实验的成果即可, 但是需要对 pc 进行细微的修改, 去掉其自增功能, 这里的 pc+4 由自定义的加法器实现; 除此之外还需要添加 pcsrc 作为中间信号表示是否为 branch 指令。
4. 将 controller 和设计好的 datapath 封装为一个 mips 模块, 最后将 mips 模块封装到 top 顶部模块中, 最后再处理模块内部的 wire 连线。

3.1.2 仿真与调试

1. 利用文件包中提供的 testbench 模块进行仿真测试, 需要把部分未显示的信号从左侧拖动到波形图中, 然后重新进行仿真
2. 调试完成后, 若过程正确, 则会执行最后一条 sw 指令, 将寄存器 2 内的数据 7 存到数据存储器地址为 84 的地方。

3.2 问题以及解决方案

问题描述:最初设计 datapath 模块时, 针对每种类型的指令都进行了具体设计, 综合仿真的时候存在同一模块重复调用的错误

解决方案:删除多余的模块实例化, 让各种指令自调用所需的模块

问题描述:仿真结果失败, 各模块之间的连线存在问题, 位数或者端口名不正确

解决方案:查找不正确的端口名, 修改连线, 保证各个模块的端口正确对应

4 实验结果及分析

4.1 仿真图

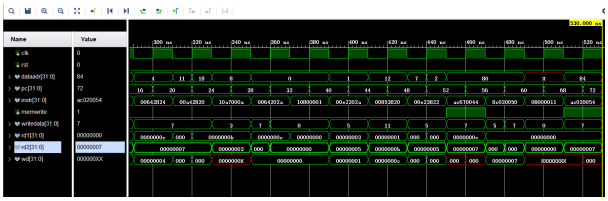

图 1: 含 pc,instr,rs,rt,rd,result 等信号的仿真图

4.2 结果

图 2: 控制台打印输出 Simulation succeeded

4.3 结果分析

1. 450ns 时 instr=ac670044,执行 sw 指令,存入 dataram 为 RD2 值为 7;
2. 470ns 时 instr=8c020050,执行 lw 指令,存入 regfilw 为 WD3 值为 7;
3. 490ns 时 instr=08000011,执行 j 指令,跳转到 00000044 处;
4. 510ns 时 instr=ac020054,执行 sw 指令,writedata 为 7,dataaddr 为 00000054(2d'84)。

A Datapath 代码

```
module datapath(
 input clk,
 input rst,
 input memtoreg,//回写的数据来自于 ALU 计算的结果0/存储器读取的数据1
 input regdst,//写入寄存器堆的地址是 rt 还是 rd,0 为 rt,1 为 rd
 input regwrite,//是否需要写寄存器堆
 input alusrc,//送入 ALU B 端口的值是立即数的 32 位扩展1/寄存器堆读取的值0
 input branch,//是否为 branch 指令, 且满足 branch 的条件
 input jump,//是否为 jump 指令
 input [2:0]alucontrol,
 input [31:0]instr,
 input  [31:0]readdata,
 output [31:0]pc,
 output [31:0]aluout,
 output [31:0]writedata,
```

```
output inst_ce
   );
wire [31:0]rd1,rd2;
wire pcsrc,zero;
wire overflow;
 assign writedata=rd2;//写入数据存储器
 assign pcsrc=zero&branch;

 //立即数符号扩展
 wire [31:0]signimm;
 signext ex(.a(instr[15:0]),.y(signimm));

 //alusrc ,alu的第二个输入
 wire [31:0]srcB;
 mux2 #(32) src1(.s(alusrc),.a(rd2),.b(signimm),.res(srcB));

 //alu
 alu alu1(.num1(rd1),.num2(srcB),.op(alucontrol),.res(aluout),.overflow(overflow)
     ,.zero(zero));
 // regdst,0 为 rt,1 为 rd
 wire [4:0]writeR;
 mux2 #(5)WR(.s(regdst),.a(instr[20:16]),.b(instr[15:11]),.res(writeR));

 //memtoreg
 wire [31:0]wd;//回写寄存器
 mux2 #(32)memdata(.s(memtoreg),.a(aluout),.b(readdata),.res(wd));

 //指令左移2位
 wire [31:0]instr_sl2;
 sl2 sl21(.a(instr),.y(instr_sl2));
 //立即数左移2位
 wire [31:0]signimm_sl2;
 sl2 sl22(.a(signimm),.y(signimm_sl2));

 //pc
 wire [31:0]pc_add4,pc_next;
 wire [31:0]pc_tmp,pc_branch,pc_j;
 pc pc0(.clk(clk),.rst(rst),.pc_next(pc_next),.pc(pc),.inst_ce(inst_ce)); //存储下
     一条地址
 adder1 pc1(.a(pc),.b(32'h00000004),.y(pc_add4));
 adder1 pc2(.a(pc_add4),.b(signimm_sl2),.y(pc_branch));
 assign pc_j={pc_add4[31:28],instr_sl2[27:0]};
 //branch,0为pc+4, 1为pc_branch
 mux2 #(32) branch0(.a(pc_add4),.b(pc_branch),.s(pcsrc),.res(pc_tmp));
 //
 mux2 #(32) jump0(.a(pc_tmp),.b(pc_j),.s(jump),.res(pc_next));//得到下一条地址

 //寄存器
```

```
    regfile regfile0(.clk(clk),.we3(regwrite),.ra1(instr[25:21]),.ra2(instr[20:16]),.
        wa3(writeR),.wd3(wd),.rd1(rd1),.rd2(rd2));

endmodule
```