

《计算机组成原理》实验报告

年级、专业、班级	2023 级计算机科学与技术 06 班 2023 级计算机科学与技术 04 班 2023 级计算机科学与技术 04 班	姓名	饶格奇 刘雨霜 李隆征
实验题目	实验三简单周期 CPU 实验		
实验时间	2025 年 5 月 11 日	实验地点	DS1410
实验成绩	优秀/良好/中等	实验性质	<input type="checkbox"/> 验证性 <input checked="" type="checkbox"/> 设计性 <input type="checkbox"/> 综合性
教师评价： <input type="checkbox"/> 算法/实验过程正确； <input type="checkbox"/> 源程序/实验内容提交； <input type="checkbox"/> 程序结构/实验步骤合理； <input type="checkbox"/> 实验结果正确； <input type="checkbox"/> 语法、语义正确； <input type="checkbox"/> 报告规范； 其他： <div>评价教师：任骛</div>			
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: 2025 年 5 月 9 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 Controller

2.1.1 功能描述

Controller 模块负责将取来的指令中的 OP 和 FUNC 字段翻译成各类控制信号, 驱动下游的数据通路 (Datapath) 完成正确的指令执行。它由 Maindec 和 ALUdec 两级译码器组成通过这两级译码, Controller 将复杂的指令语义分解为一组简单的布尔控制信号, 确保数据通路各单元在每个时钟周期内协同工作。

2.1.2 接口定义

表 1: Controller 接口

信号名	方向	位宽	功能描述
op	Input	6-bit	指令的 opcode 字段
func	Input	6-bit	R-type 指令的 function 字段
jump	Output	1-bit	跳转控制信号, 高电平表示执行 J-type
branch	Output	1-bit	分支判断使能, 与 zero 相与成 PCsrc 控制信号
alusrc	Output	1-bit	ALU 第二操作数来源: 0= 寄存器, 1= 立即数
alucontrol	Output	3-bit	传输给 ALU 的具体运算控制码
memwrite	Output	1-bit	数据存储写使能, 高电平写
memtoreg	Output	1-bit	写回寄存器的数据来源: 0=ALU, 1=Mem
regdst	Output	1-bit	写回寄存器来源: 0=rt, 1=rd
regwrite	Output	1-bit	寄存器写使能, 高电平写

2.2 数据通路

2.2.1 功能描述

Datapath 模块实现指令的取指、译码、执行、访存与回写五大阶段的数据流与运算逻辑。

2.2.2 接口定义

表 2: datapath 接口

信号名	方向	位宽	功能描述
clka	Input	1-bit	时钟信号,所有时序逻辑同步上升沿
rst	Input	1-bit	异步复位,高电平清零 PC 和寄存器
jump	Input	1-bit	来自 controller 的跳转控制
branch	Input	1-bit	来自 controller 的分支控制
alusrc	Input	1-bit	来自 controller 的 ALU 源选择
memtoreg	Input	1-bit	来自 controller 的写回数据源选择
regwrite	Input	1-bit	来自 controller 的寄存器写使能
alucontrol	Input	1-bit	来自 controller 的 ALU 运算控制信号
instr	Input	32-bit	取指存储器输出的 32 位指令
readdata	Input	32-bit	数据存储器读出的 32 位数据
aluout	Output	32-bit	ALU 运算结果
pc	Output	32-bit	当前程序计数器值
writedata	Output	32-bit	要写入数据总线的 32 位数据

3 实验过程记录

记录实验的过程,完成了什么样的工作,存在的问题包括哪些,解决方案如何等。subsubsection 名称自行设定。

4 实验结果及分析

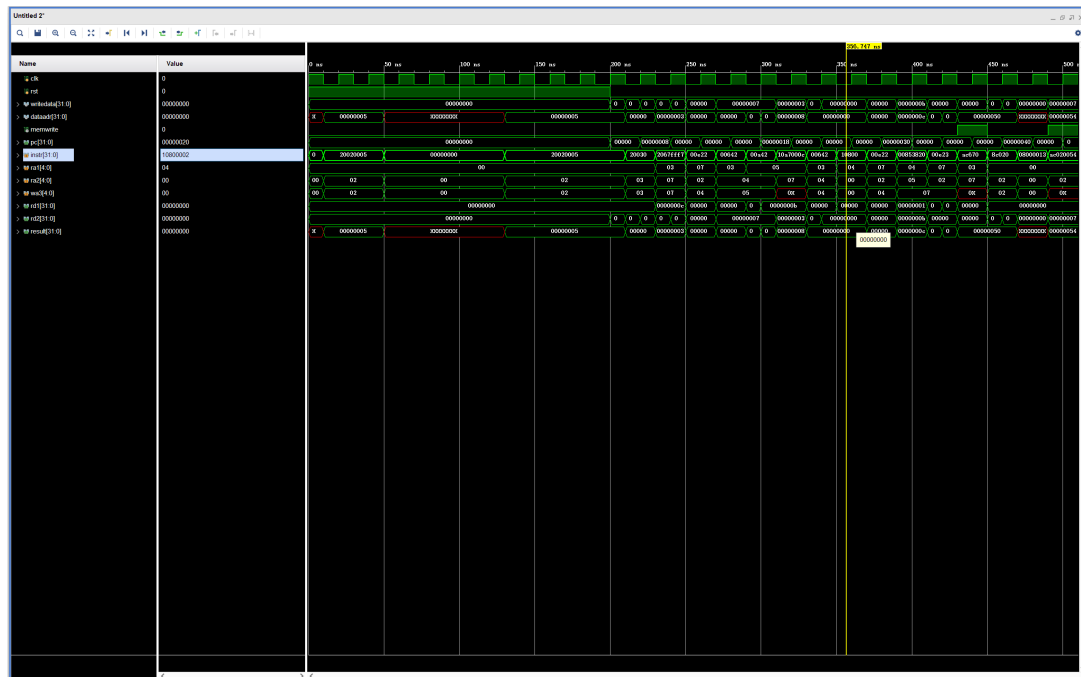
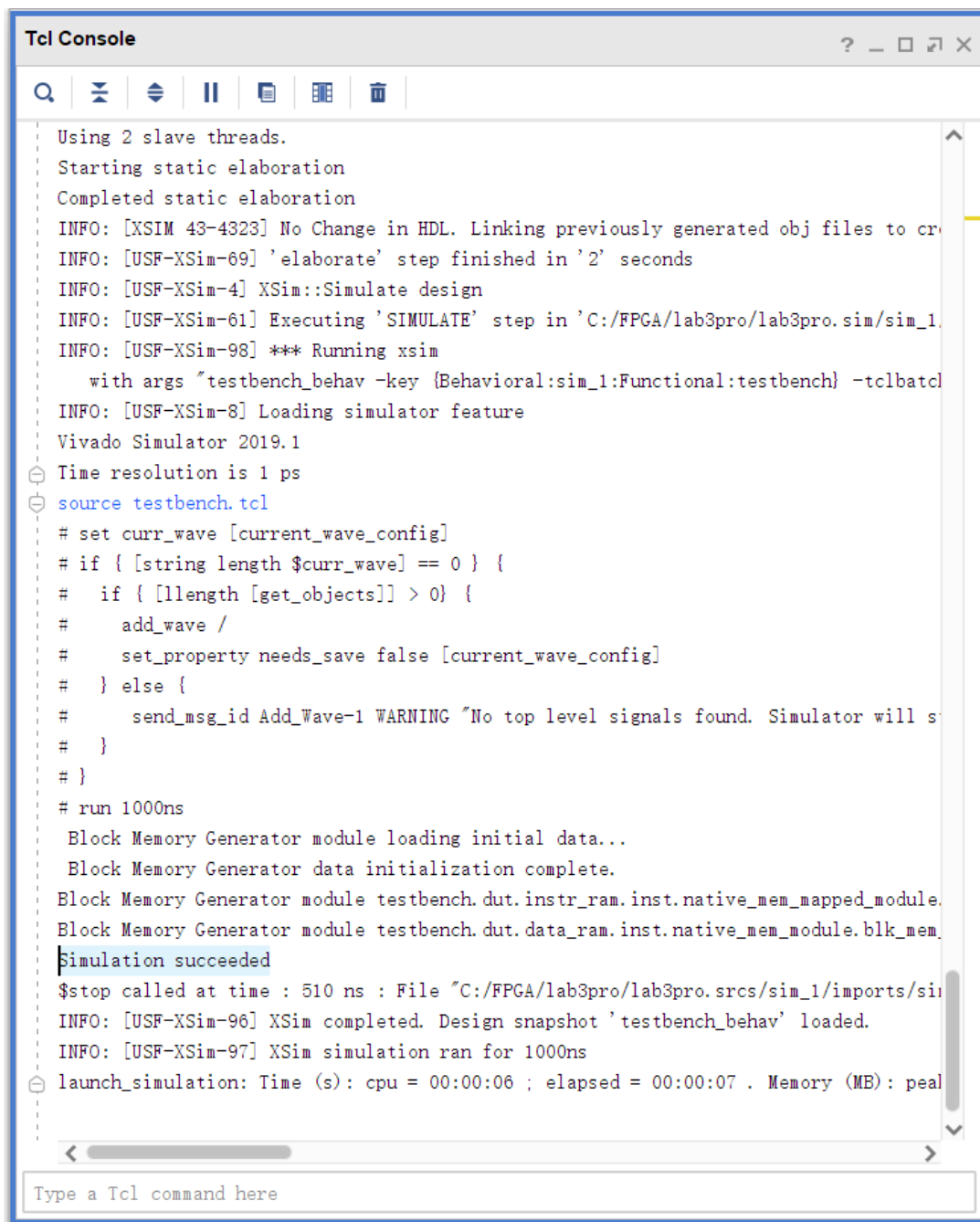


图 1: 仿真结果



The screenshot shows a 'Tcl Console' window with a toolbar at the top containing icons for search, zoom, run, pause, copy, paste, and delete. The main area displays the following text:

```
Using 2 slave threads.
Starting static elaboration
Completed static elaboration
INFO: [XSIM 43-4323] No Change in HDL. Linking previously generated obj files to cr
INFO: [USF-XSim-69] 'elaborate' step finished in '2' seconds
INFO: [USF-XSim-4] XSim::Simulate design
INFO: [USF-XSim-61] Executing 'SIMULATE' step in 'C:/FPGA/lab3pro/lab3pro.sim/sim_1.
INFO: [USF-XSim-98] *** Running xsim
    with args "testbench_behav -key {Behavioral:sim_1:Functional:testbench} -tclbatch
INFO: [USF-XSim-8] Loading simulator feature
Vivado Simulator 2019.1
Time resolution is 1 ps
source testbench.tcl
# set curr_wave [current_wave_config]
# if { [string length $curr_wave] == 0 } {
#   if { [llength [get_objects]] > 0 } {
#     add_wave /
#     set_property needs_save false [current_wave_config]
#   } else {
#     send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will s
#   }
# }
# run 1000ns
Block Memory Generator module loading initial data...
Block Memory Generator data initialization complete.
Block Memory Generator module testbench.dut.instr_ram.inst.native_mem_mapped_module.
Block Memory Generator module testbench.dut.data_ram.inst.native_mem_module.blk_mem.
Simulation succeeded
$stop called at time : 510 ns : File "C:/FPGA/lab3pro/lab3pro.srscs/sim_1/imports/si
INFO: [USF-XSim-96] XSim completed. Design snapshot 'testbench_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:06 ; elapsed = 00:00:07 . Memory (MB): peal
```

At the bottom, there is a text input field with the placeholder text 'Type a Tcl command here'.

图 2: 控制台结果

```
timescale 1ns / 1ps
//
//
// Company:
// Engineer:
//
// Create Date: 2025/05/06 19:46:00
// Design Name:
// Module Name: datapath
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//
//
module datapath(
    input wire clka,          // 时钟
    input wire rst,           // 复位信号
    input wire jump,          // 跳转控制
    input wire branch,        // 分支控制
    input wire alusrc,         // ALU源选择
    input wire mentoreg,       // 数据选择: 内存到寄存器
    input wire regwrite,       // 寄存器写控制
    input wire regdst,         // 寄存器目的选择
    input wire [2:0] alucontrol, // ALU控制信号
    output wire [31:0] aluout, // ALU结果
    output wire [31:0] pc,     // 当前PC地址
    output wire [31:0] writedata, // 写入数据
    input wire [31:0] instr,   // 指令
    input wire [31:0] readdata // 数据存储器读出数据
);

wire [31:0] readdata1; // 寄存器读数据1
wire [31:0] readdata2; // 寄存器读数据2
wire [31:0] addrext;   // 扩展后的地址
wire [31:0] pcnew;     // 新的PC值
wire [31:0] pcplus4;   // PC+4
```

```

wire [31:0] jumpaddr;      // 跳转地址
wire [27:0] jumpleft;      // 左移后的跳转地址
wire [31:0] branchleft;    // 左移后的分支地址
wire [31:0] objectaddr;    // 目标地址
wire pcsrc;                // PC源选择
wire zero;                 // 零标志

// 寄存器模块
regfile regfile(
    .clk(clka),
    .rst(rst),
    .we3(regwrite),        // 修正：使用regwrite而不是memtoreg
    .ra1(instr[25:21]),
    .ra2(instr[20:16]),    // 修正：ra2应该是instr[20:16]而不是instr[15:11]
    .wa3((regdst==1) ? instr[15:11] : instr[20:16]),
    .wd3((memtoreg==1) ? readdata : aluout),
    .rd1(readdata1),
    .rd2(readdata2)
);

// 将readdata2赋给writedata，用于存储到内存
assign writedata = readdata2;

// 符号扩展模块
signext sext(
    .addr(instr[15:0]),
    .y(addrext)
);

// ALU模块
alu alu1(
    .num1(readdata1),
    .num2((alusrc==1) ? addrext : readdata2),
    .op(alucontrol),
    .result(aluout),
    .zero(zero)
);

// 分支条件 = branch & zero
assign pcsrc = branch & zero;

// PC模块
pc pc1(
    .clk(clka),
    .rst(rst),
    .pcnew(pcnew),
    .pc(pc)
);

```



```

// 加法器1: 计算PC+4
adder ad1(
    .a(pc),
    .b(32'd4),
    .y(pcplus4)
);

// 加法器2: 计算分支目标地址
adder ad2(
    .a(pcplus4),
    .b(branchleft),
    .y(objectaddr)
);

// 26位左移2位模块
bits26sl2 s1(
    .addr(instr[25:0]),
    .y(jumpleft)
);

// 计算跳转地址
assign jumpaddr = {pcplus4[31:28], jumpleft};

// 左移2位模块
sl2 s2(
    .addr(addrext),
    .y(branchleft)
);

// 计算新的PC值
// 如果jump=1, 使用jumpaddr
// 如果jump=0但pcsrc=1(branch & zero=1), 使用objectaddr
// 否则使用pcplus4
assign pcnew = (jump==1'b1) ? jumpaddr :
                (pcsrc==1'b1) ? objectaddr : pcplus4;

endmodule

```