《计算机组成原理》实验报告

年级、专业、班级	2020 级计算机科学与技术 01 班 2020 级计算机科学与技术 02 班	**姓名**	陈鹏宇 徐小龙
实验题目	实验三简易单周期 CPU 实验		
实验时间	2022 年 05 月 07 日	**实验地点**	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: 肖春华

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: 2022 年 5 月 28 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，支持整数算术逻辑运算，存取，跳转等指令。

2.1.2 接口定义

表 1: 接口定义

信号名	方向	位宽	功能描述
writedata	Output	32-bit	regfile 取出的 RD2 操作数, 写入数据存储器
dataadr	Output	32-bit	数值为 ALU 的结果，当 memwrite 有效时为写入存储器地址，无效时为读出存储器地址
instr	Output	32-bit	从 inst_ram 中读取的 32 位 MIPS 指令
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
readdata	Output	32-bit	从数据存储器读出的数据

2.1.3 逻辑控制

信号名	位宽	功能描述
memtoreg	1-bit	多选器控制信号，高电平代表写入寄存器堆的数据为从数据存储器中读取的数据，低电平代表 alu 结果写入寄存器堆
memtwrite	1-bit	数据存储器写使能信号，高电平有效
pcsrc	1-bit	多选器控制信号，高电平代表条件跳转指令生效，低电平代表 pc+4
branch	1-bit	高电平代表该指令为条件跳转指令
alusrc	1-bit	多选器控制信号，高电平代表 alu 的第二个操作数来自符号拓展后的立即数，低电平代表 RD2
regdst	1-bit	多选器控制信号，高电平代表寄存器堆的写地址为 ra3，低电平代表 ra2
regwrite	1-bit	寄存器堆写使能信号
jump	1-bit	多选器控制信号，高电平代表无条件跳转指令生效，低电平代表不生效

3 实验过程记录

3.1 datapath 的设计与仿真

1. datapath 主要由 pc，adder，alu，regfile 等模块构成，均已完成或以提供，仅需要添加 mux 将各个端口连接即可
2. controller 模块已在实验二实现，将各个控制信号输入 datapath，为 mux，regfile 提供控制信号
3. inst_ram，data_ram 直接调用 ip 核，由 datapath 输入 pc，dataadr 和 writedata 后，将 readdata 和 inst 输入 datapath
4. 在 datapath 内部，根据单周期的完整通路图，一一连接 add、and、sub、or、slt、beq、j、lw、sw、addi 等指令的数据通路即可
5. 采用提供的 testbench.v，把需要观察的信号从左侧添加到波形图中，然后重新进行仿真
6. 调试完后无误则会执行最后一条 sw 指令，将 $2 的 7 存到数据存储器的 [84], 并在控制台得到’Simulation succeeded’

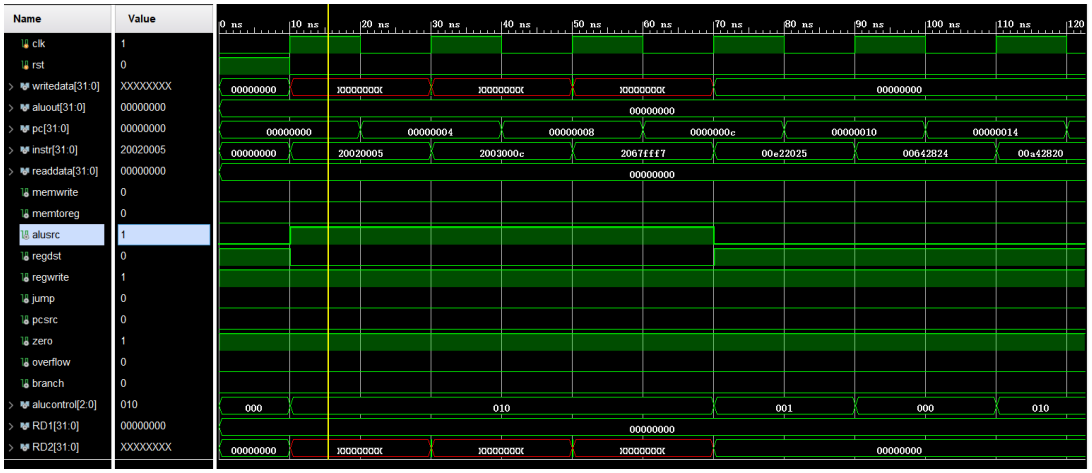
3.2 问题 1:inst 比 pc 信号延迟

问题描述：仿真后发现 inst 比 pc 慢了两个周期

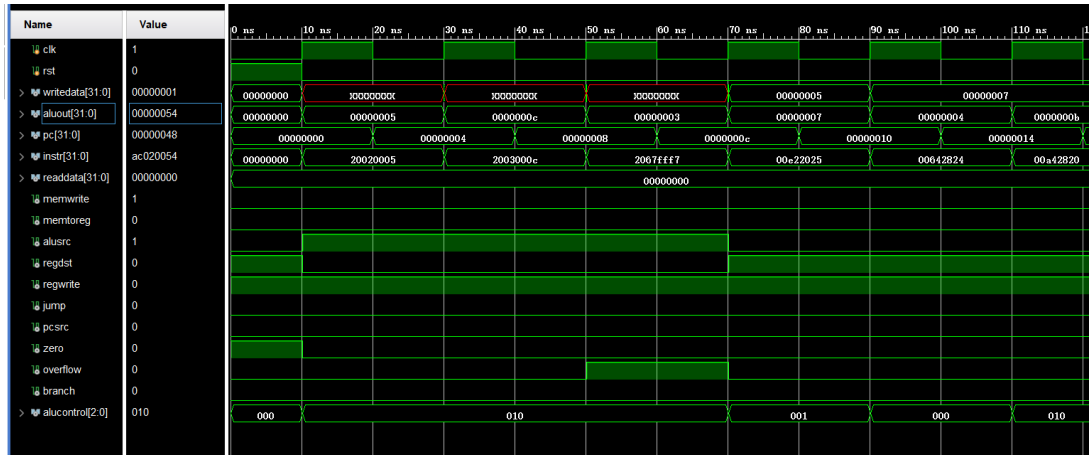
解决方案：了解 ip 后发现选中 Primitives Output Register 会增加一个寄存器导致延时，但删除后还是延时了一个周期，推测 pc 的时钟比指令内存和数据内存的时钟慢得多，将原来的 posedge clk 改为 negedge clk 后变为正常

3.3 问题 2: 数据读出异常

问题描述：对指令的译码没有问题，但读不出 RD 的数据



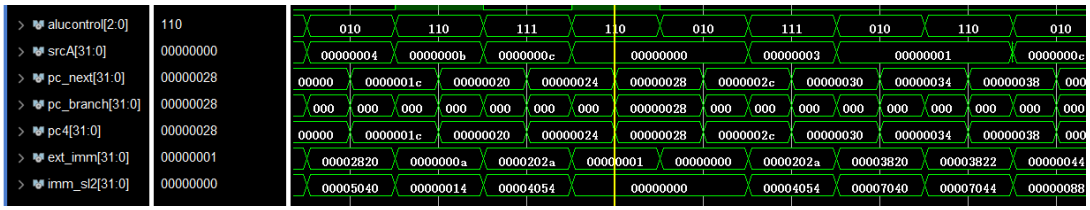
解决方案：发现 datapath 中一些名称过于相似混淆，如 regwrite 和 writereg，或者将写入 dataram 的值和写入 regfile 的值搞混淆。另外源代码添加了过多无必要导线导致冗杂不方便阅读，删除后结果如下



3.4 问题 3:beq 指令无法跳转

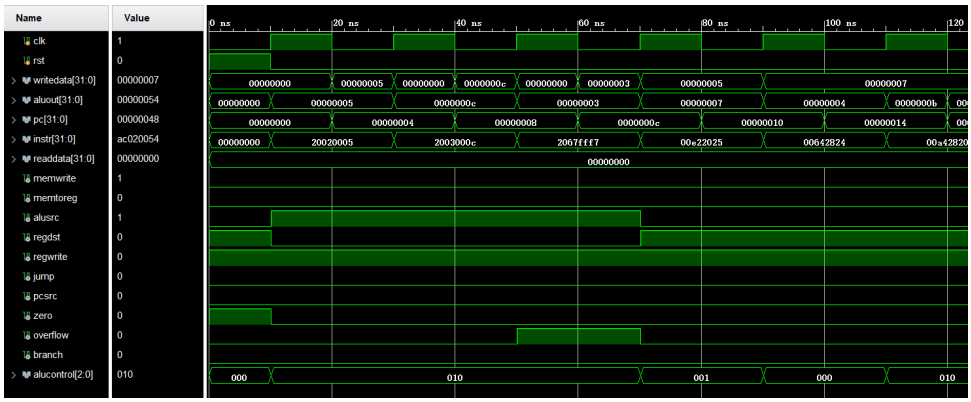
问题描述：branch 信号有效却无法跳转

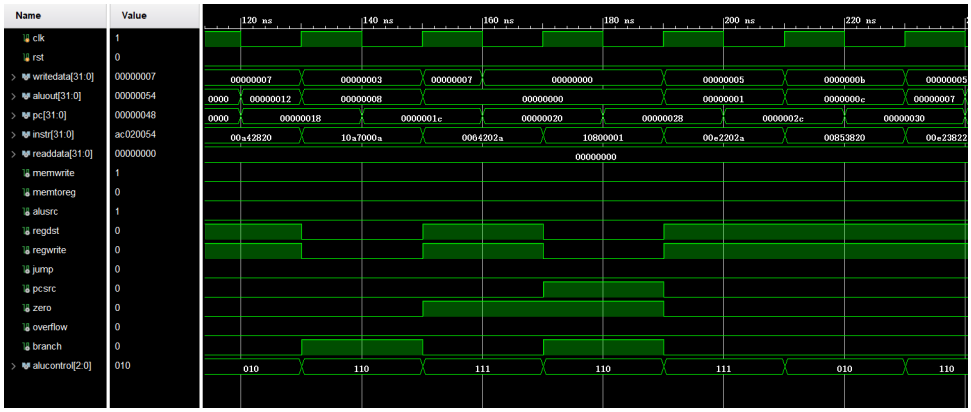
解决方案：为了解情况，将所有与 pc 有关信号加入 wave 中，发现 pc，branch 并没有在 pc4 的基础上加偏移，发现是 sl2 失效的原因，具体原因是 sl2 中的拼接位跳过了第一位，修改后正常

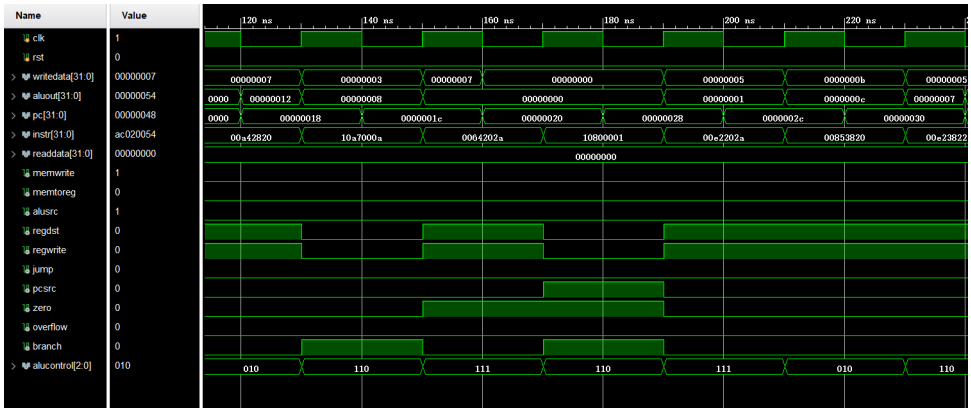


4 实验结果及分析

4.1 仿真图







4.2 控制台输出

```
Block Memory Generator module loading initial data...
Block Memory Generator data initialization complete.
Block Memory Generator module testbench.dut.imem.inst.native_mem_module.blk_mem_gen_v8_4_2_inst is using a behavioral model for simulation which will not precisely model memory collisior
Block Memory Generator module testbench.dut.dmem.inst.native_mem_module.blk_mem_gen_v8_4_2_inst is using a behavioral model for simulation which will not precisely model memory collisior
Simulation succeeded
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 = 00:00:05 ; elapsed = 00:00:10 . Memory (MB): peak = 824.371 ; gain = 26.738
```

4.3 结果分析

1. instr = 20020005 时为 addi 指令，将二号寄存器赋初值 5，在下降沿 regfile 有效时，成功写入
2. instr = 10800001 时为 beq 指令，由于四号寄存器为 0，所以 branch 信号有效，跳转到 pc=28 位置
3. instr = ac670044 时为 sw 指令，将七号寄存器保存到 80 的位置上，aluout 为写 data_mem 的地址 80，写入的值为 writedata(RD2) 7
4. instr = 8c020050 时为 lw 指令，将 80 位置上的树赋给二号寄存器，aluout 为读 data_mem 的地址 80，readdata 的值为 7，与上一指令一致，而 writedata 在下降沿由原来的值 5 变为 7
5. instr = 08000011 时为 jump 指令，jump 信号有效，直接跳转到 pc=44 的位置

A Datapath 代码

```
module datapath(
    input clk,
    input reset,
    input memtoreg,
    input pcsrc,
    input branch,
    input alusrc,
    input regdst,
    input regwrite,
    input jump,
    input [2:0]alucontrol,
    output overflow,
    output zero,
    output [31:0] pc, //output pc to inst_ram
    input  [31:0] inst,
    output [31:0] aluout,
    output [31:0] writedata,//write for data_mem
    input  [31:0] readdata //read from data_mem
    );

    wire [31:0] pc4;  //pc + 4
    wire [31:0] pc_branch; //branch pc
    wire [31:0] pc_next; // real next pc
    wire [31:0] real_pc; //branch and jump

    wire [31:0] ext_imm; //after extension
    wire [31:0] imm_sl2; //after left 2
```

```
    wire [31:0] result; //real writedata
    wire [31:0] srcB;  //alu operand B
    wire [31:0] srcA;  //alu operand A == RD1
    wire [4:0]  writereg;    //write address

    assign pcsrc = branch & zero;

    ////////////////////mux//////////////////////////
    mux2 a3_mux(inst[20:16], inst[15:11], regdst, writereg);
    mux2 wd_mux(aluout, readdata, memtoreg, result);
    mux2 srcB_mux(writedata, ext_imm, alusrc, srcB);
    mux2 pc1_mux(pc4, pc_branch, pcsrc, pc_next);
    mux2 pc2_mux(pc_next, {pc4[31:28] , inst[25:0] , 2'b00}, jump, real_pc);
    ////////////////////pc//////////////////////////
    pc counter(clk, reset, real_pc, pc);
    adder pcadder(pc, 32'b100, pc4);   //default add 4
    adder branchadder(imm_sl2, pc4, pc_branch);
    ////////////////register file//////////////////
    regfile regs(clk, regwrite, inst[25:21], inst[20:16], writereg, result,
        srcA, writedata);
    ////////////////signext_imm////////////////////
    signext sign(inst[15:0], ext_imm);
    ////////////////////sl2//////////////////////////
    sl2 s1(ext_imm, imm_sl2);
    ////////////////////alu//////////////////////////
    alu al(srcA, srcB, alucontrol, zero, overflow, aluout);  //input aluout to
        data_ram to get readdata
endmodule
```