《计算机组成原理》实验报告

年级、专业、班级	2022级计算机科学与技术03班/01班/06班	姓名	叶旭航，解吴雪，李佳玲
实验题目	实验三简易单周期CPU实验		
实验时间	2024 年 4 月 16 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: 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: 2024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(datapath)

2.1.1 功能描述

计算机中用于执行指令的路径，它包含了各种功能模块（如寄存器、ALU、存储器等）以及它们之间的连接，实现了指令的操作流程和数据的传输。

2.1.2 接口定义

2.1.3 逻辑控制

Datapath将pc,mux2,pc_inst,branch,reg,alu等结合起来，代码的思路是构建一个能够执行MIPS指令的数据路径。数据路径包括指令的取值、解码、执行和写回阶段。PC用于跟踪下一条指令的地址，寄存器堆用于存储操作数和结果，ALU执行计算，多路选择器用于选择数据来源和写回的目的地。整个数据路径由多个组合逻辑和时序逻辑组件组成，它们在时钟信号的控制下协同工作以执行指令。

2.2 寄存器堆(Regfile)

2.2.1 功能描述

一个小型、高速存储单元集合，用于在处理器执行指令过程中临时存储操作数和中间结果。

表 1: 数据通路接口定义

信号名	方向	宽度	含义
clk	in	1	时钟信号
rst	in	1	复位信号
memstoreg	in	1	回写的数据来自于 ALU 或者存储器
memwrite	in	1	是否要书写数据存储器
pcsrc	in	1	下一个 PC 是 CP+4 还是新地址
alusrc	in	1	送入的是立即数的 32 位扩展还是寄存器堆读取的值
regdst	in	1	写入寄存器堆的是 rt 还是 rd
regwrite	in	1	是否书写寄存器堆
jump	in	1	跳转指令
alucontrol	in	3	指令结果
instr	in	32	指令码
readdata	in	32	从存储器读取的数据
pc	out	32	程序计数器的地址
aluout	out	32	算术逻辑单元的计算结果
writedata	out	32	要写入存储器的数据
zero	out	1	是否为 0

2.2.2 接口定义

表 2: 寄存器堆接口定义

接口信号名	方向	宽度	含义
clk	输入	1	时钟信号，用于同步寄存器文件的操作
we3	输入	1	写使能信号，为1时允许写操作
ra1	输入	5	第一个读地址，用于选择第一个源寄存器
ra2	输入	5	第二个读地址，用于选择第二个源寄存器
wa3	输入	5	写地址，用于选择写回的寄存器
wd3	输入	32	写数据，将要写入寄存器的数据
rd1	输出	32	第一个读数据，来自第一个源寄存器的值
rd2	输出	32	第二个读数据，来自第二个源寄存器的值

2.2.3 逻辑控制

在 `regfile` 模块的逻辑控制部分，重点控制逻辑如下：

1. 写控制：当写使能信号 `we3` 为高时，在时钟信号 `clk` 的上升沿，将输入数据 `wd3` 写入由写地址 `wa3` 指定的寄存器中。
2. 读控制：读操作是组合逻辑，不受时钟控制。当读地址 `ra1` 或 `ra2` 不为 0 时，分别从寄存器文件 `rf` 中读取对应地址的寄存器值输出到 `rd1` 或 `rd2`；如果读地址为 0，则输出 0，这是因为寄存器 0（`$zero`）在 MIPS 架构中是硬连线为 0 的寄存器。

3 实验过程记录

3.1 问题1: 设计 Datapath 模块连接

问题描述：由给出的参考书得知 Datapath 模块的各个组成部分及其连接关系，因为 datapath 内的模块大多已给出或前述实验已完成，所以重点在需要详细分析各部分的功能和数据流向，确保模块之间的正确连接。

解决方案：

1. **PC 模块连接：**
   - `adderPC_inst` 实例化了一个加法器模块，用于将当前 PC 地址加上一个偏移量，得到下一个 PC 地址。
   - `mux2 PC_select` 实例化了一个多路选择器模块，根据 `pcsrc` 信号选择下一个 PC 地址。

- `PCnext_inst` 实例化了一个多路选择器模块，根据 `jump` 信号选择是否跳转到指定地址。
- `PC_inst` 实例化了 PC 模块，用于存储并更新 PC 地址。

2. **Branch 计算和存储：**
   - `sign_extend_inst` 实例化了一个符号扩展模块，将指令中的 16 位符号扩展成 32 位。
   - `adderBranch_inst` 实例化了一个加法器模块，用于计算分支目标地址。
3. **寄存器文件和数据路径选择：**
   - `regfile_inst` 实例化了寄存器文件模块，用于读取和写入寄存器数据。
   - `mux2 select_dst` 实例化了一个多路选择器模块，根据 `regdst` 信号选择写入的寄存器地址。
4. **ALU 计算和数据选择：**
   - `select_srcB` 实例化了一个多路选择器模块，根据 `alusrc` 信号选择 ALU 的第二个操作数。
   - `ALU_inst` 实例化了一个 ALU 模块，用于执行算术逻辑运算。
   - `mux2 select_res` 实例化了一个多路选择器模块，根据 `memtoreg` 信号选择 ALU 的输出或数据存储器的读取数据作为最终结果。
5. **其他连接：**连接各个模块之间的数据和控制信号，确保正常的数据流和操作顺序。

3.2 问题2: 两个存储器实现

问题描述：使用 Block Memory Generator IP 构造指令存储器。

解决方案：使用 IP 实例化数据存储器和指令存储器，配置不同的宽度和深度参数。指令存储器需要导入指定的 coe 文件以支持所需操作。

3.3 问题3: 端口适配

问题描述：使其余模块端口适配顶层文件。

解决方案：由于 MIPS 软核顶层文件和设计顶层文件 `top` 均已给出，故修改了底层的各模块的输入输出端口，以适配顶层文件的调用需求。

3.4 问题4: 仿真调试

问题描述：调用 `top` 模块进行仿真。

解决方案：根据已给出的仿真文件对实现的单周期 CPU 进行仿真，通过仿真图和控制台打印的输出判断仿真是否成功。

4 实验结果及分析

4.1 仿真图

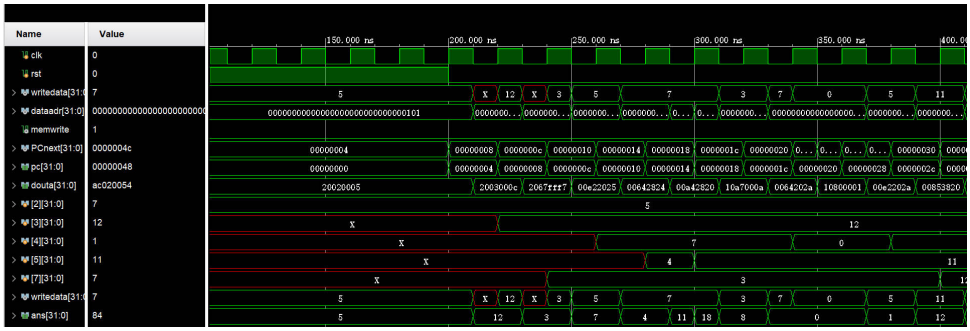
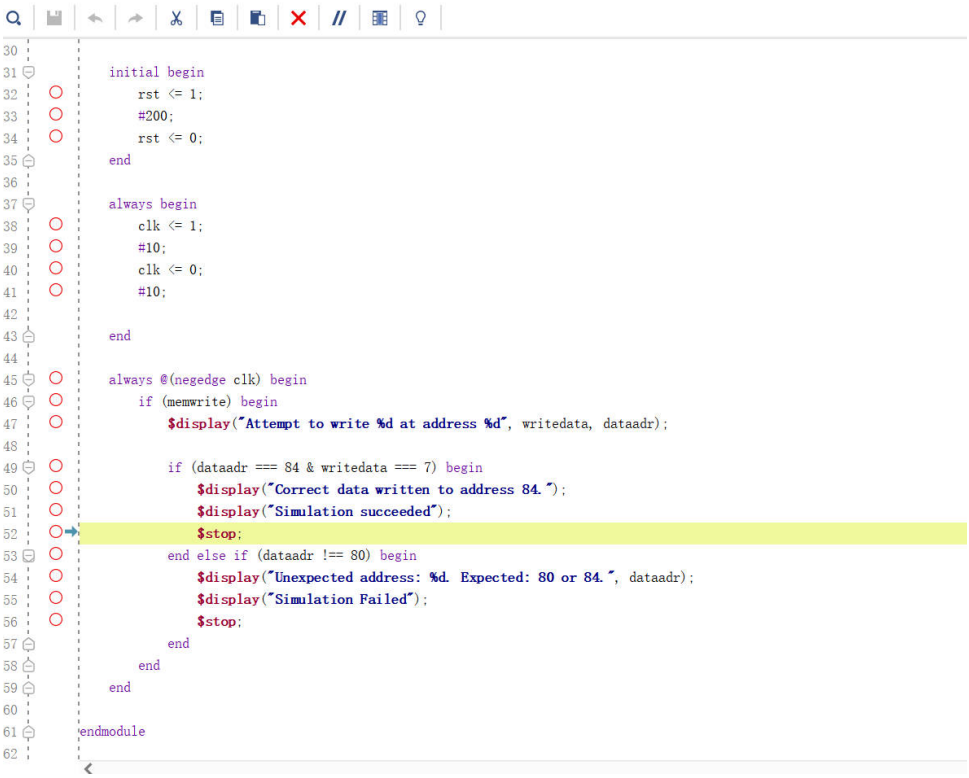


图 1: 仿真1



```
30
31      initial begin
32          rst <= 1;
33          #200;
34          rst <= 0;
35      end
36
37      always begin
38          clk <= 1;
39          #10;
40          clk <= 0;
41          #10;
42
43      end
44
45      always @(negedge clk) begin
46          if (memwrite) begin
47              $display("Attempt to write %d at address %d", writedata, dataadr);
48
49              if (dataadr === 84 & writedata === 7) begin
50                  $display("Correct data written to address 84.");
51                  $display("Simulation succeeded");
52                  $stop;
53              end else if (dataadr !== 80) begin
54                  $display("Unexpected address: %d. Expected: 80 or 84.", dataadr);
55                  $display("Simulation Failed");
56                  $stop;
57              end
58          end
59      end
60
61  endmodule
62
```

图 2: 仿真2

A Datapath代码

```
module datapath(
    input clk, rst,
    input memtoreg, memwrite, pcsrc, alusrc, regdst, regwrite, jump,
    input [2:0] alucontrol,
    input [31:0] instr,
    input [31:0] readdata,
    output [31:0] pc,
```

```
    output [31:0] aluout, writedata,
    output wire zero
);

wire [31:0] adderPC, branchPC, PCnext_temp;
adder adderPC_inst(.a(pc), .b(32'h4), .y(adderPC));
mux2 PC_select(.option1(adderPC), .option2(branchPC), .select(pcsrc), .data
    (PCnext_temp));
wire [31:0] PCjump, PCnext;
assign PCjump = {adderPC[31:28], instr[25:0], 2'b00};
mux2 PCnext_inst(.option1(PCnext_temp), .option2(PCjump), .select(jump), .
    data(PCnext));
pc PC_inst(.clk(clk), .rst(rst), .newPC(PCnext), .pc(pc));
wire [31:0] sign_extend;
sign_extend sign_extend_inst(.sign(instr[15:0]), .sign_extend(sign_extend))
    ;
adder adderBranch_inst(.a(adderPC), .b(sign_extend<<2), .y(branchPC));
wire [31:0] srcA, res;
wire [4:0] wa3;
regfile regfile_inst(.clk(clk), .we3(regwrite), .ra1(instr[25:21]), .ra2(
    instr[20:16]), .wa3(wa3), .wd3(res), .rd1(srcA), .rd2(writedata));
mux2 select_dst(.option1({27'b0, instr[20:16]}), .option2({27'b0, instr
    [15:11]}), .select(regdst), .data(wa3));
wire [31:0] srcB;
mux2 select_srcB(.option1(writedata), .option2(sign_extend), .select(alusrc
    ), .data(srcB));
ALU ALU_inst(.num1(srcA), .num2(srcB), .op(alucontrol), .ans(aluout), .zero
    (zero));
mux2 select_res(.option1(aluout), .option2(readdata), .select(memtoreg), .
    data(res));

endmodule
```