《计算机组成原理》实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| **年级、专业、班级** |  | **姓名** | 周梓蕙 |
| **实验题目** | 实验三简易单周期CPU 实验 | | |
| **实验时间** | 2021 年 6 月 7 日 | **实验地点** | ds1410 |
| **实验成绩** | 优秀/良好/中等 | **实验性质** | □验证性  □G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: 2021 年 6 月 11 日

# 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。

# 实验设计

这一节，主要描述各个模块的功能、接口、逻辑控制方法（状态机控制方法）等。(红字为内容说明，请删除)

## 数据通路

### 功能描述

数据通路是cpu 里除了controller、inst\_mem、data\_mem 之外的部件，其利用 pc 确定指令的存储地址，通过输入的控制信号完成寄存器的读和写等操作，以及 alu 的相关运算，输出指令存储地址和数据存储地址以及计算结果等。

* + 1. **接口定义**

# 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。subsubsec- tion 名称自行设定。

## 完成的工作

该实验基本完成了实验要求，即取指令并执行。

**表 1:** 接口定义模版

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **方向** | **位宽** | **功能描述** |
| clk | Input | 1-bit | 时钟周期信号 |
| rst | Input | 1-bit | 清零信号，为 1 时执行初始化操作。 |
| memtoreg | Input | 1-bit | 控制信号。为 1 时，选择存储器读出来的数据寄存器；为 0 时，将 alu 的运  算结果存入寄存器的写数据端口。 |
| pcsrc | Input | 1-bit | 控制信号。为 1 时 pc 选择为条件跳  转的地址；为 0 时pc 选择pc+1。 |
| alusrc | Input | 1-bit | 控制信号。为 0 时选择 rd2 作为 alu  的 srcB 端输入；为 1 时选择扩展后的立即数作为输入。 |
| regdst | Input | 1-bit | 控制信号。为 0 时选择指令的 17 21  位作为寄存器写数据的地址；为 1 时  选择指令的 12 16 位作为寄存器写数据的地址。 |
| regwrite | Input | 1-bit | 控制信号。为 1 时，将数据写入寄存  器。 |
| jump | Input | 1-bit | 控制信号。为 1 时，选择跳转的地址作为 pc 的值；为 0 时，选择条件跳转  的指令地址或pc+1 作为pc 的值。 |
| alucontrol | Input | 1-bit | alu 运算的控制信号。 |
| overflow | Output | 1-bit | 控制信号。 |
| zero | Output | 1-bit | 当 alu 的运算结果为 0 时，该输出为  1。 |
| pc | Output | 32-bit | pc 的值。 |
| rd1 | Output | 32-bit | 寄存器读出的数据。 |
| rd2 | Output | 32-bit | 寄存器读出的数据。 |
| result | Output | 32-bit | alu 的运算结果。 |

## 存在的问题及解决方案

### 部分信号变化有快慢差异

readdata 信号和 wd3 信号有快慢差异，readdata 信号比 wd3 信号的变化慢了半个周期，造成结果出错。解决方案是将 data\_mem 的时钟不取反。可见，当信号的变化有快慢差别时，可以尝试改变模块的时钟信号。

### 调整代码时再仿真，结果错误

程序运行正确后，将部分冗余的输出删除再进行调试，获取截图。此时，由于误删，导致代码无法运行。解决方案是将仿真中显示的输出罗列在纸上，再寻找不正确的信号或模块。最终发现模块内的中间连线的定义被删除，导致无法运行。

## 心得体会

### 信号变化延迟的解决办法

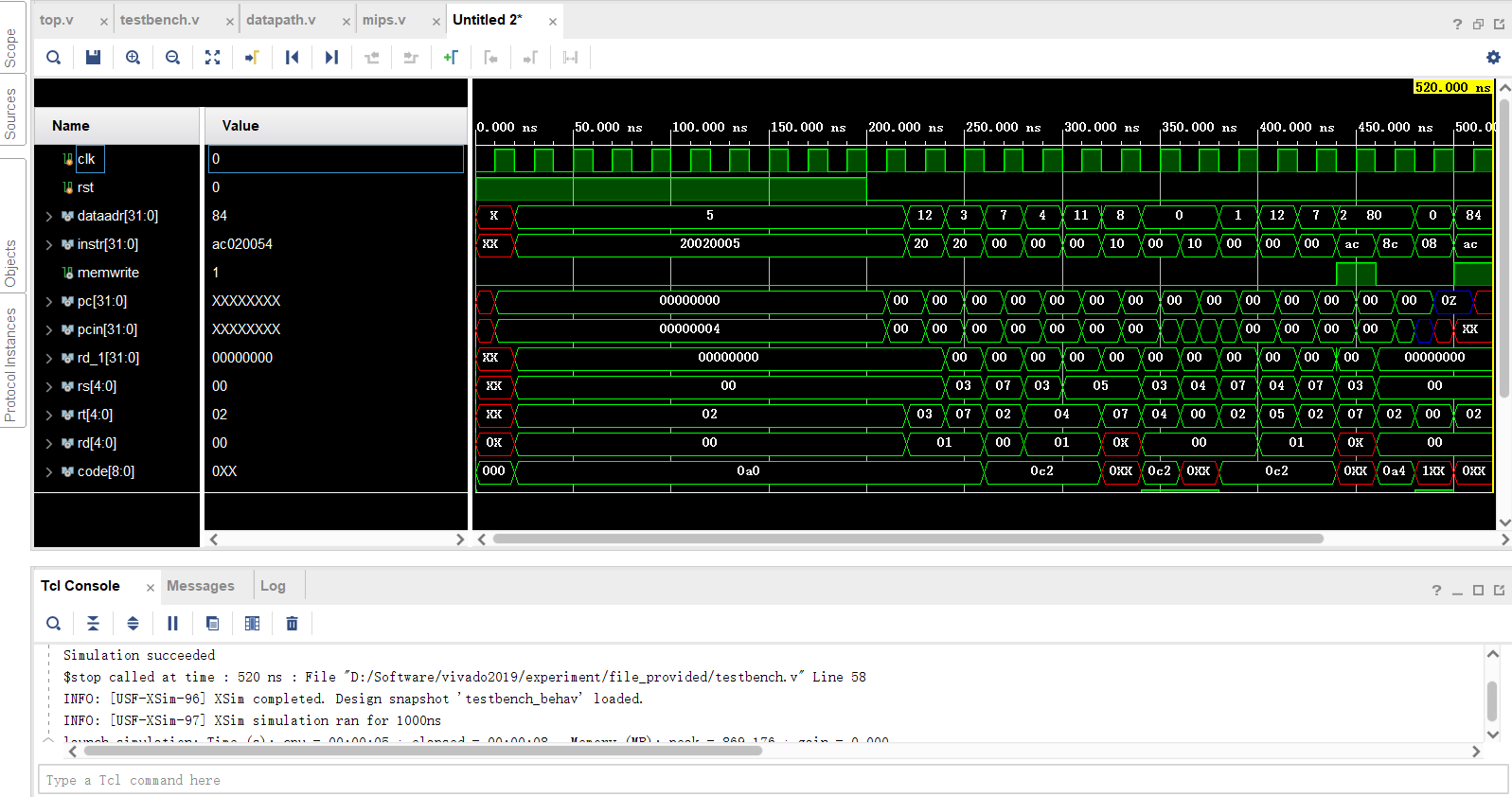
当信号的变化有快慢差别时，可以尝试改变模块的时钟信号。

### 删除代码要谨慎

当程序运行出现了阶段性的成果时，应当备份。

# 实验结果及分析

## 仿真图



**图 1**

仿真结果如图。其中，dataadr 为alu 的运算结果，即“result”。

# A Datapath 代码

module datapath ( clk , rst , memtoreg , pcsrc , alusrc ,

regdst , regwrite , jump , alucontrol , overflow , zero , pc , instr , aluout , writedata , readdata , rd\_1 , pcin , src\_b , wa

);

parameter WIDTH = 32;

input clk , rst , memtoreg , pcsrc , alusrc , regdst , regwrite , jump , overflow ; input[ WIDTH - 1 : 0] instr;

input[ WIDTH - 1 : 0] readdata; input [2 : 0] alucontrol; output zero ;

output[ WIDTH - 1 : 0] aluout;

output[ WIDTH - 1 : 0] writedata ; output[ WIDTH - 1 : 0] pc , pcin ; output[ WIDTH - 1 : 0] rd\_1 , src\_b ; output [4 : 0] wa;

wire[ WIDTH - 1 : 0] pc\_plus , pc\_branch\_1 , pc\_branch\_2 , pc\_jump\_1 , pcin , sign\_imm , pc\_sl , rd\_1 , rd\_2 , src\_b , wa , wd\_3 , pc\_branch ;

wire [27 : 0] pc\_jump ;

mux2 mux\_branch (. a( pc\_plus), . b( pc\_branch ), . select( pcsrc), . result( pc\_branch\_ 2 ));

mux2 mux\_jump (. a( pc\_branch\_ 2 ), . b( pc\_jump\_ 1 ), . select( jump ), . result( pcin ))

;

PC p(. clk ( clk ), . rst( rst), . pc( pc), . pcin ( pcin ));

adder add\_pc\_plus (. a( pc), . b(32 ’ b100 ), . result( pc\_plus)); signext sign\_ext (. a( instr [15 : 0]), . result( sign\_imm )); sl2 sl\_pc\_plus (. a( sign\_imm ), . result( pc\_sl));

adder add\_pc\_branch (. a( pc\_sl), . b( pc\_plus), . result( pc\_branch )); sl2 #(26) sl\_pc\_jump (. a( instr [25 : 0]), . result( pc\_jump ));

regfile reg\_file (. clk (~ clk ), . we3 ( regwrite), . ra1 ( instr [25 : 21]) , . ra2 ( instr [20 : 16]) , . wa3 ( wa), . wd3 ( wd\_3 ), . rd1 ( rd\_1 ), . rd2 ( rd\_2 ));

mux2 #(5) mux\_wa (. a( instr [20 : 16]) , . b( instr [15 : 11]) , . select( regdst), . result( wa));

mux2 #(32) mux\_src\_b (. a( rd\_2 ), . b( sign\_imm ), . select( alusrc), . result( src\_b

));

alu a(. a( rd\_1 ), . b( src\_b ), . clk (~ clk ), . reset( rst), . op( alucontrol), . result( aluout), . zero ( zero ));

mux2 #(32) mux\_result (. a( aluout), . b( readdata), . select( memtoreg ), . result( wd\_3 ));

assign pc\_jump\_ 1 = { pc\_plus [31 : 28], pc\_jump }; assign writedata = rd\_2 ;

endmodule