**深 圳 大 学 实 验 报 告**

**课 程 名 称： 计算机系统(3)**

**实验项目名称： 取指和指令译码设计**

**学 院： 计算机与软件学院**

**专 业： 计算机与软件学院所有专业**

**指 导 教 师： 王毅**

**报告人： 郑雨婷 学号： 2021150122 班级： 高性能**

**实 验 时 间： 2023年10月12日**

**实验报告提交时间： 2023年11月5日**

**教务处制**

**一、 实验目标：**

设计完成一个连续取指令并进行指令译码的电路，从而掌握设计简单数据通路的基本方法。

**二、实验内容**

本实验分成三周（三次）完成：1）首先完成一个译码器（30分）；2）接着实现一个寄存器文件（30分）；3）最后添加指令存储器和地址部件等将这些部件组合成一个数据通路原型（40分）。

**三、实验环境**

硬件：桌面PC

软件：Linux Chisel开发环境

**四、****实验步骤及说明**

本次试验分为三个部分：

1. 设计译码电路，输入位32bit的一个机器字，按照课本MIPS 指令格式，完成add、sub、lw、sw指令译码，其他指令一律译码成nop指令。输入信号名为Instr\_word，对上述四条指令义译码输出信号名为add\_op、sub\_op、lw\_op和sw\_op，其余指令一律译码为nop；

给出Chisel设计代码和仿真测试波形，观察输入Instr\_word为add R1,R2,R3; sub R0,R5,R6，lw R5,100(R2), sw R5,104(R2)、JAL RA,100(R2)时，对应的输出波形

Instr\_decoder

Instr\_word[31:0]

add\_op

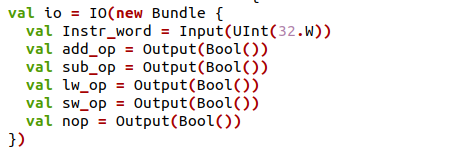
sub\_op

lw\_op

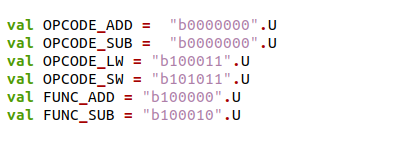
sw\_op

nop

目标是设计一个译码器（Decoder），其输入为一条指令，是32位的信号。输出为五个1位信号，一位信号也可以用布尔型（Bool）来表示。因此，Decoder.scala的输入输出部分的定义如下：



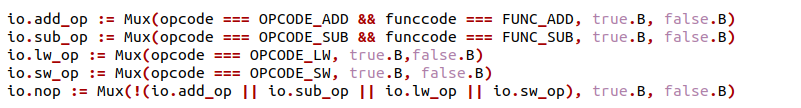
题目要求完成add、sub、lw、sw指令译码，其他指令一律译码成nop指令。在MIPS指令中，add指令和sub指令的操作码相同都是000000,lw指令和sw指令的操作码是100011和101011。为了区分add指令和sub指令，还需要把add的功能码和aub的功能码定义出，分别是100000和100010。



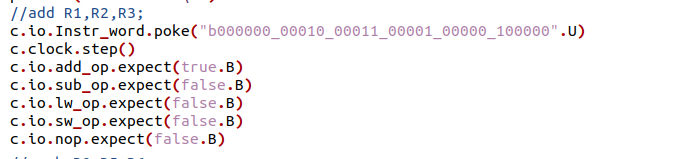
定义好这些之后，从输入的32位指令中提取前六位作为操作码以及后六位作为功能码。



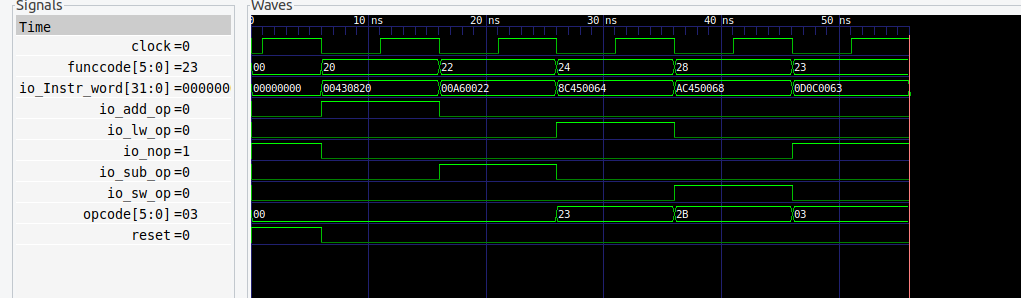
利用复选器Mux来确定输出信号的值，根据不同指令类型的匹配条件来指定输出值。ADD和SUB指令需要满足操作码和功能码都匹配才能赋值位真，LW和SW只需要满足操作码匹配即可。若这四种指令都不是，那么就将nop设置为真。



测试文件将五条指令依次输入(只给出add指令，其余相似)，看输出是否符合预期，sbt test通过，编写成功。



生成的波形图如下，可以看到当输入add指令时，add\_op为1，其余全为0；当输入sub指令时，sub\_op为1，其余全为0；当输入lw指令时，lw\_op为1，其余全为0；当输入sw指令时，sw\_op为1，其余全为0；当输入jal指令时，nop为1，其余全为0。



1. 设计寄存器文件，共32个32bit寄存器，允许两读一写，且0号寄存器固定读出位0。四个输入信号为RS1、RS2、WB\_data、Reg\_WB，寄存器输出RS1\_out和RS2\_out；寄存器内部保存的初始数值等同于寄存器编号

给出Chisel设计代码和仿真测试波形，观察RS1=5,RS2=8，WB\_data=0x1234,Reg\_WB=1的输出波形和受影响寄存器的值。

Reg\_file

RS1[4:0]

RS2[4:0]

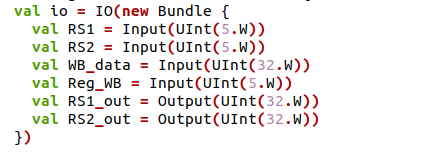
WB\_data[31:0]

Reg\_WB

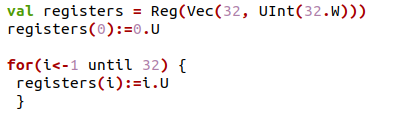
RS1\_out[31:0]

RS2\_out[31:0]

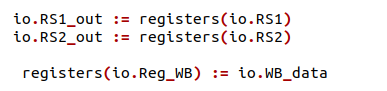
目标是设计一个寄存器文件，其输入为RS1、RS2、WB\_data、Reg\_WB，输出RS1\_out和RS2\_out。其中RS1、RS2、Reg\_WB是寄存器的编号，寄存器共32个，需要五位。因此，WB\_data、RS1\_out、RS2\_out是寄存器的数据，每个寄存器都是32位的。 Reg.scala的输入输出部分的定义如下：



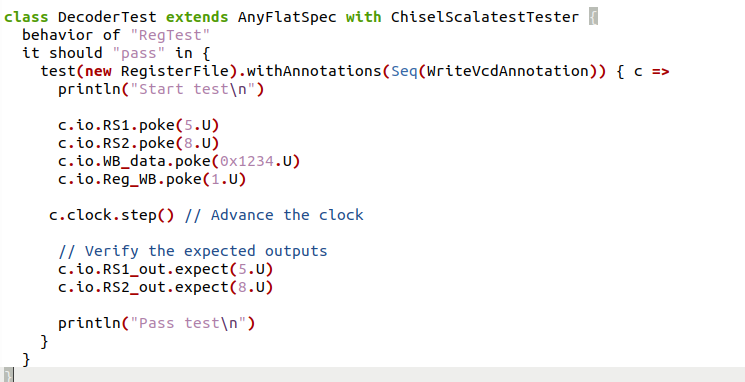
定义一个时序逻辑信号，初始化为一个包含32个元素的向量，每个元素都是一个32位宽，对应着MIPS中的32个32位寄存器。并且使用for语句令每个寄存器内部保存的初始数值等同于寄存器编号。



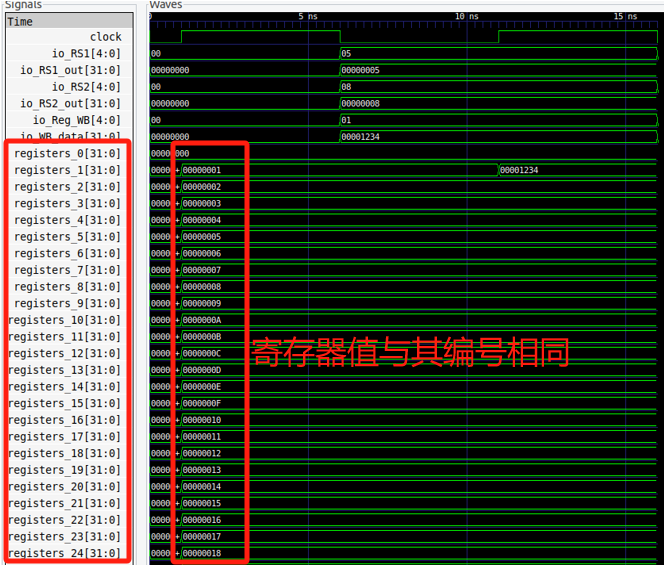
输出为RS1寄存器的数据和RS2寄存器的数据，还需要将WB\_data写入Reg\_WB寄存器。



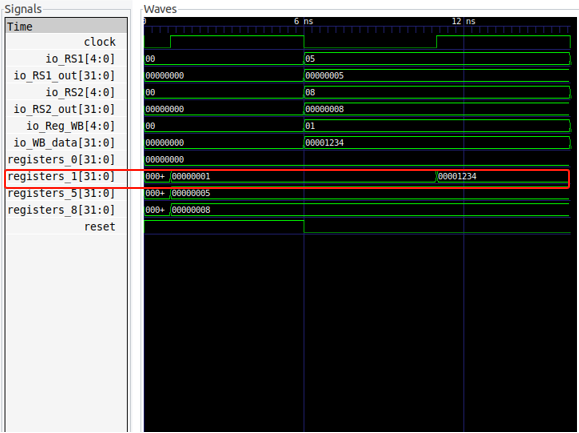
测试文件中令RS1=5,RS2=8，WB\_data=0x1234,Reg\_WB=1，测试通过。



输出波形如下，可以看到大多寄存器的值与其编号相同，说明初始赋值成功。

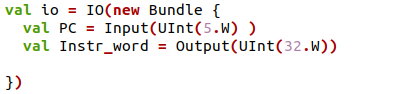


将无关的寄存器删除，只留下R1、R5、R8,可以观察到，R5的值为5，R8的值为8，R1的值初始为1然后再经过一个时钟周期之后，被赋值成了）0x00001234,验证寄存器文件设计正确。

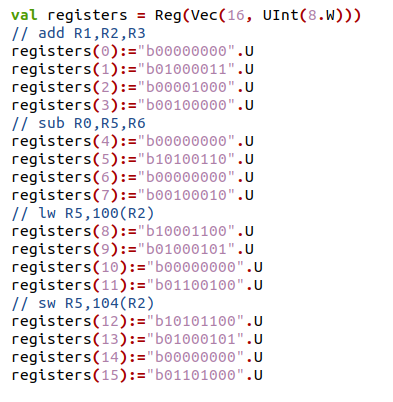


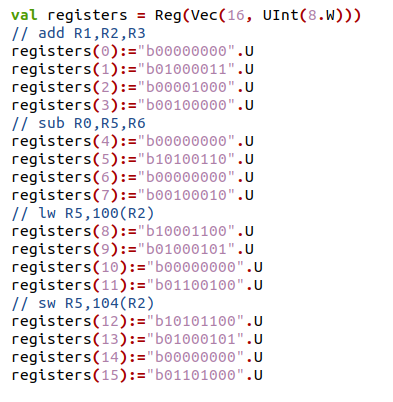
1. 实现一个32个字的指令存储器，从0地址分别存储4条指令add R1,R2,R3; sub R0,R5,R6，lw R5,100(R2), sw R5,104(R2)。然后组合指令存储器、寄存器文件、译码电路，并结合PC更新电路（PC初值为0）、WB\_data和Reg\_WB信号产生电路，最终让电路能逐条指令取出、译码（不需要完成指令执行）。

目标是设计一个指令存储器，从0地址分别存储4条指令add R1,R2,R3; sub R0,R5,R6，lw R5,100(R2), sw R5,104(R2)。根据PC值的不同可以输出不同的指令。因此输入是PC，输出是32位宽的指令。

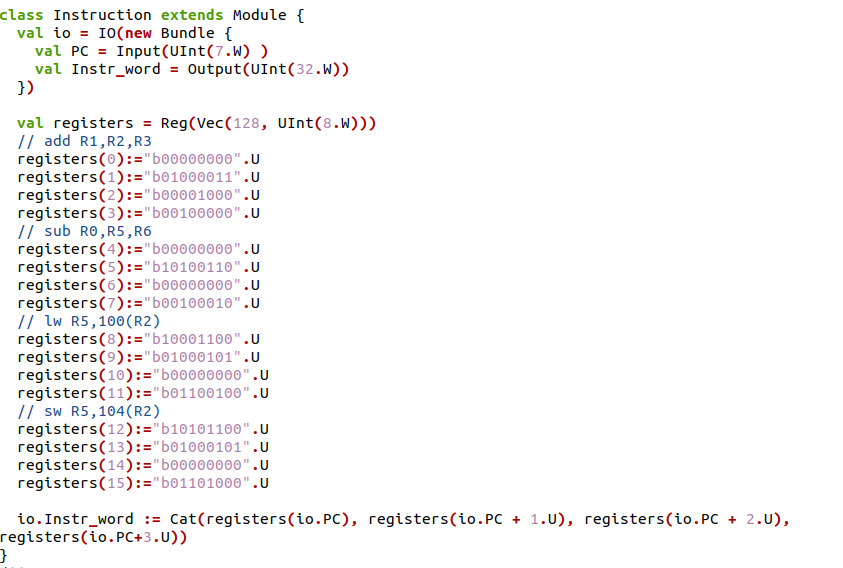


PC值每次加4，因此一条指令占四个地址，也就是说一个地址可以存8 bits。于是可以定义一个含16个元素，每个元素占8位宽的向量。并将四条指令逐条写入。

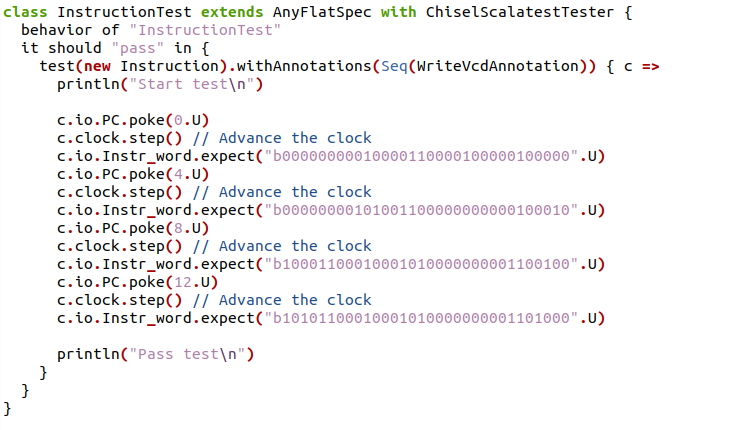




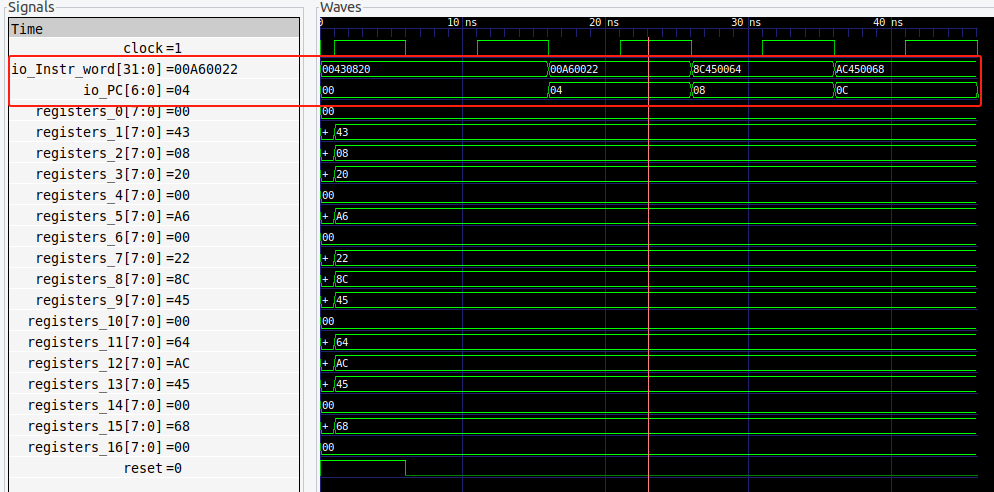
根据输入的PC，将四部分用Cat函数拼接成完整的一条指令输出。



在编写的测试程序中，令PC为0，4，8，12.输出也与四条指令相符，通过测试，编写正确。



查看输出的波形文件，可以看到PC值不同时Instr\_word的值也不同，正好是add R1,R2,R3; sub R0,R5,R6，lw R5,100(R2), sw R5,104(R2)这四条指令的二进制表示。



给出Chisel设计代码和仿真测试波形，观察四条指令的执行过程波形，记录并解释其含义。

PC

（初值为0）

+4

指令内存

Instr\_decoder

Reg\_file

RS1[4:0]

RS2[4:0]

WB\_data[31:0]

Reg\_WB

RS1\_out[31:0]

RS2\_out[31:0]

add\_op

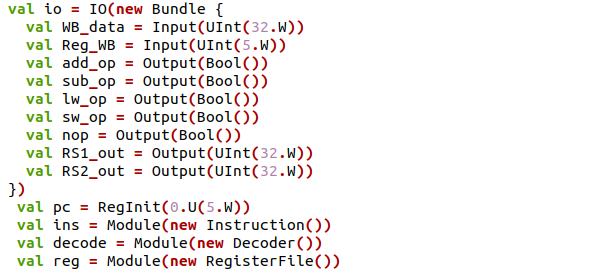
sub\_op

lw\_op

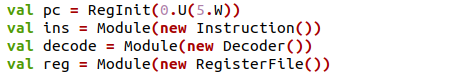
sw\_op

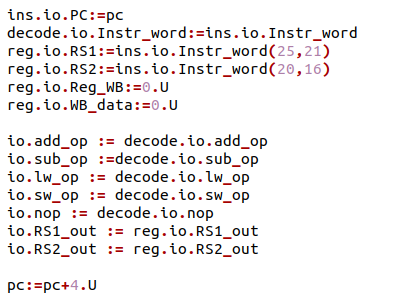
nop

现在我们有了译码器、寄存器文件和指令存储器这三个模块，将这三个模块拼接成一个数据通路大模块，按照上图的方式连接，整个数据通路有两个输入（WB\_data、Reg\_WB）和七个输出（add\_op、sub\_op、lw\_op、sw\_op、nop、RS1\_out、RS2\_out）。



载入三个模块以及PC值，设PC的初始值为0，在程序最后让PC+4。并且按照上图将输入输出连接，PC作为输入给到指令存储器，指令存储器的输出作为输入给到寄存器文件和译码器。





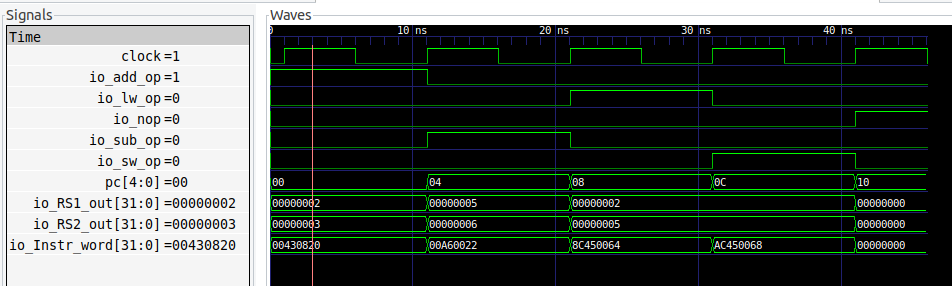
编写测试文件通过之后，查看仿真测试波形如下。可以观察到：

第一个时钟周期时PC值为0，Instr-word为0x00430820,是add R1,R2,R3对应的十六进制表示，指令的rs为R2,rt为R3,add\_op为1。

第二个时钟周期时PC值为4，Instr-word为0x00A60022,是sub R0,R5,R6对应的十六进制表示，指令的rs为R5,rt为R6,sub\_op为1。

第三个时钟周期时PC值为8，Instr-word为0x8C450064,是lw R5,100(R2)对应的十六进制表示，指令的rs为R2,rt为R5,lw\_op为1。

第四个时钟周期时PC值为12，Instr-word为0xAC450068,是sw R5,104(R2)对应的十六进制表示，指令的rs为R2,rt为R5,sw\_op为1。



**五、实验结果**

1）用Chisel语言设计了一个译码器，实现了对输入的指令进行分解，以识别出指令的类型，是一个较为简陋的译码器，只能识别出add,sub,lw,sw这四种指令，其余指令全部视为nop。

2）用Chisel语言设计一个寄存器文件,实现了一个32个32位的寄存器文件，并且将寄存器的初值初始化位寄存器的编号。可以进行两读一写，能够读取输入的两个寄存器的数据，也可以向特定的寄存器写数据。

3）用Chisel语言设计了一个指令存储器，实现根据不同PC值取出不同的指令。

4）将上述三个部件组合成一个简单的数据通路原型，实现每个周期PC值加4，指令存储器根据PC值取出指令，把指令传入译码器和寄存器文件，最终输出指令的类型以及rs、rt寄存器中的数据。

**五、实验总结与体会**

通过本次实验，我学会了使用Chisel语言进行硬件描述，还学会了如何设计和实现硬件电路。这是一个从未接触过的语言，但因为有过C语言和数字电路的基础，学习Chisel的过程并没有想象中困难。这个过程帮助我更好地理解了计算机硬件是如何工作的，也提升了我的硬件设计和描述能力。

通过本次实验，我更深入地理解了计算机体系结构的基本组成部分，如指令译码、寄存器文件和指令存储器。本实验分成三个阶段，逐步构建数据通路原型，从译码器、寄存器文件到指令存储器和地址部件的组合。这个分阶段的设计方法让我们能够更好地掌握每个组件的功能和原理，也更好地体会到了分层设计的原则。

|  |
| --- |
| **指导教师批阅意见：**  **成绩评定：**  指导教师签字：  年 月 日 |
| 备注： |