**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： 周明洋**

**报告人： 杜良衡 学号： 2022150255 班级： 01B**

**实 验 时 间： 2024.10.18**

**实验报告提交时间： 2024.10.29**

**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

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]

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）。

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

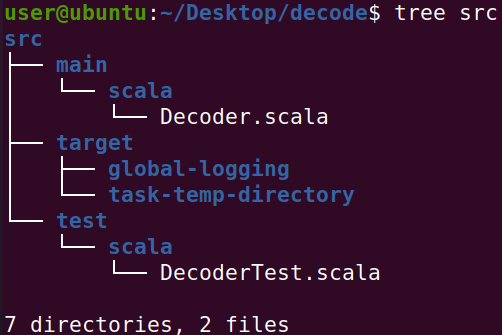
sw\_op

nop

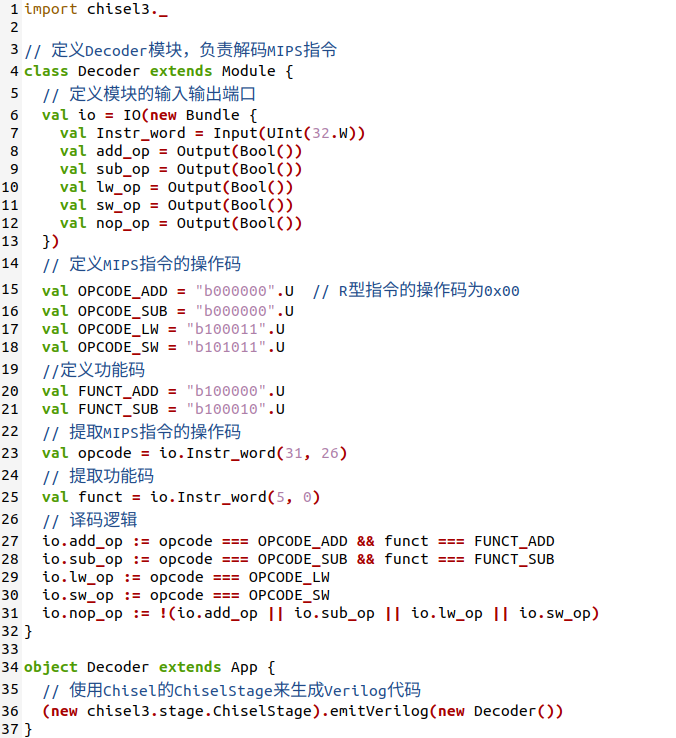
**五、实验结果**

**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；**

程序整体结构：



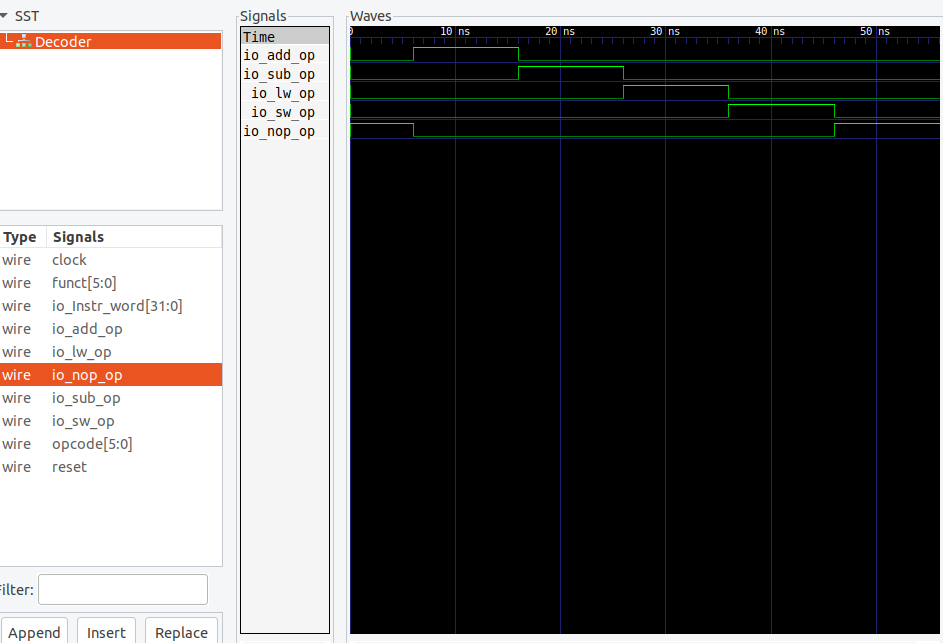
译码电路Decoder.scala用于解码MIPS指令。该模块接收一条32位的MIPS指令，并根据操作码和功能码来确定指令的类型，然后输出相应的布尔信号来表示指令类型。



测试类DecoderTest.scala使用 Chisel 测试框架 chiseltest 进行单元测试。先创建一个Decoder 模块的测试实例，并启用 VCD 输出以可视化波形，然后验证该模块是否能正确解码不同类型的MIPS指令，并输出相应的布尔信号。

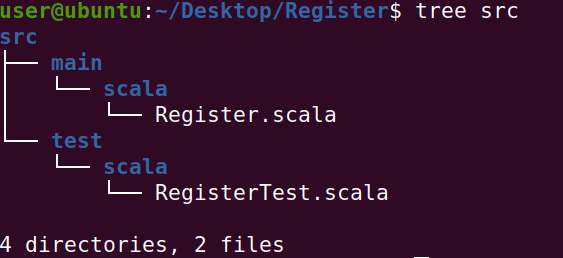


输入Instr\_word为add R1,R2,R3; sub R0,R5,R6，lw R5,100(R2), sw R5,104(R2)、JAL RA,100(R2)时，对应的输出波形。

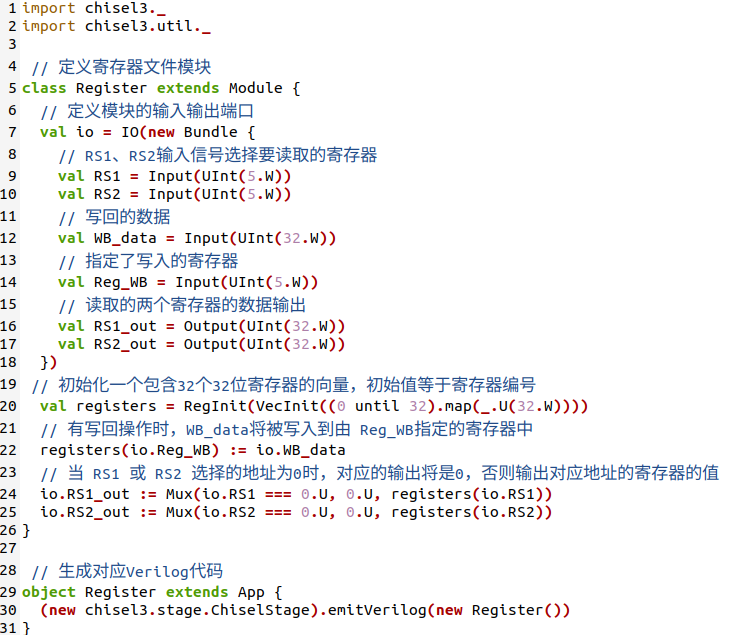


**2. 设计寄存器文件，共32个32bit寄存器，允许两读一写，且0号寄存器固定读出位0。四个输入信号为RS1、RS2、WB\_data、Reg\_WB，寄存器输出RS1\_out和RS2\_out；寄存器内部保存的初始数值等同于寄存器编号**

程序整体结构



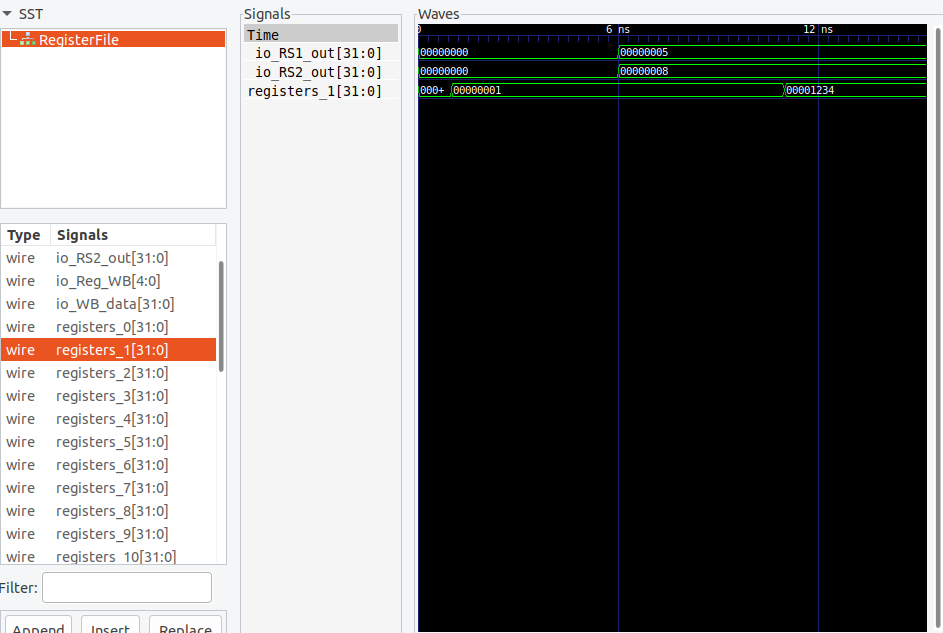
寄存器文件模块Register.scala包含了32个32位的寄存器，支持寄存器的读取与写回操作。WB\_data会被作为写回的数据写入Reg\_WB指定的寄存器中。



测试类DecoderTest.scala使用 Chisel 测试框架 chiseltest 进行单元测试，来验证一个时钟周期内寄存器内容是否被正确更新。

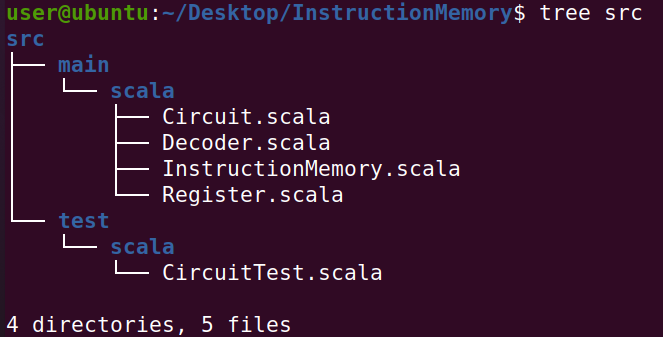


RS1=5,RS2=8，WB\_data=0x1234,Reg\_WB=1对应的输出波形以及受影响的寄存器值。



**3. 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）。**

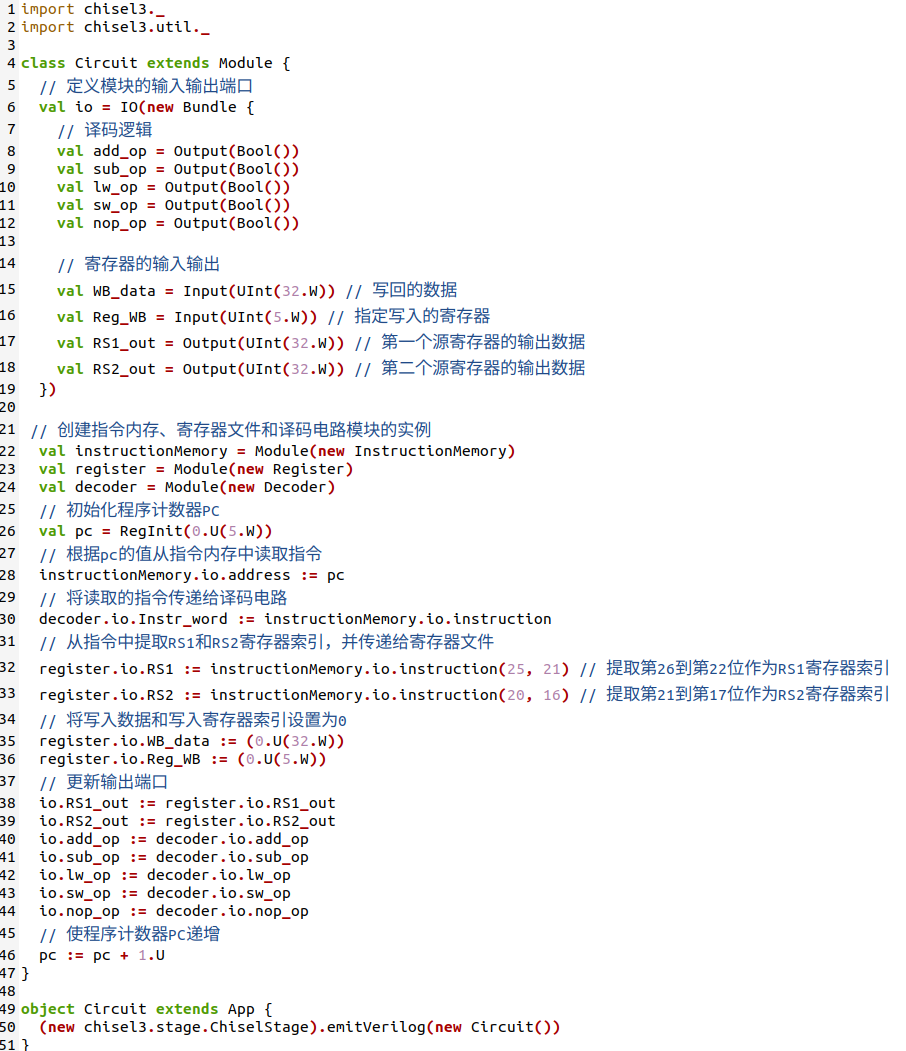
程序整体结构，复用之前定义好的译码电路Decoder以及寄存器文件Register



指令内存模块InstructionMemery.scala, 通过32位二进制的形式存储MIPS指令，并根据输入的地址输出相应的指令。



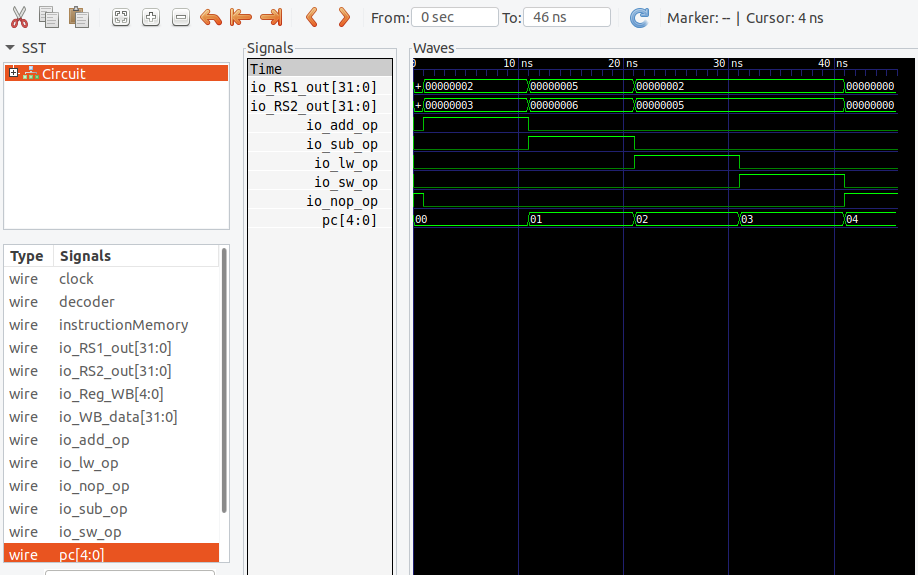
Circuit模块集成了指令内存、寄存器文件和译码电路，用于从指令内存中读取指令，解码指令，并更新寄存器文件的输出。在每次执行指令后，会使程序计数器PC递增。



测试类Cricuit模拟了四个时钟周期程序的变化，对应了程序计数器PC从0地址开始读取4条指令。



输出波形如下图所示，可以看到4个时钟周期内指令存储器内存储的4条指令add R1,R2,R3; sub R0,R5,R6，lw R5,100(R2), sw R5,104(R2)被逐条取出并进行译码。



**五、实验总结与体会**

本次实验我学会了Chisel语言的基本语法并实现了一个简单的数据通路。由于是在Linux环境下进行操作，加上初次接触Chisel语言，一开始环境的搭建与代码编译测试遇到了不少问题，不过通过网上查询Chisel学习的相关教程与视频，最终也熟悉了这门语言的使用，同时实验过程中也对之前学习到的MIPS指令操作码和控制码，时钟周期等概念有了更加深刻的认识。

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