**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： 刘 刚**

**报告人： 何泽锋 学号： 2022150221 班级： 高性能特色班**

**实 验 时 间： 2024年10月17日 —— 2024年10月24日**

**实验报告提交时间： 2024年10月24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，输出信号均为1bit。

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 100时，对应的输出波形。

Instr\_decoder

Instr\_word[31:0]

add\_op

sub\_op

lw\_op

sw\_op

nop

图1 译码器

①根据IO输入获取当前的指令，输入指令为32bit，输出指令需要译码成add\_op 、sub\_op 、lw\_op 、sw\_op 、nop五种类型，具体代码如下，定义io接口，根据Instr\_word输入的值在map中寻找对应的输出，map每次都会返回一个列表，列表中存有每个哦op指令的取值，即0或1

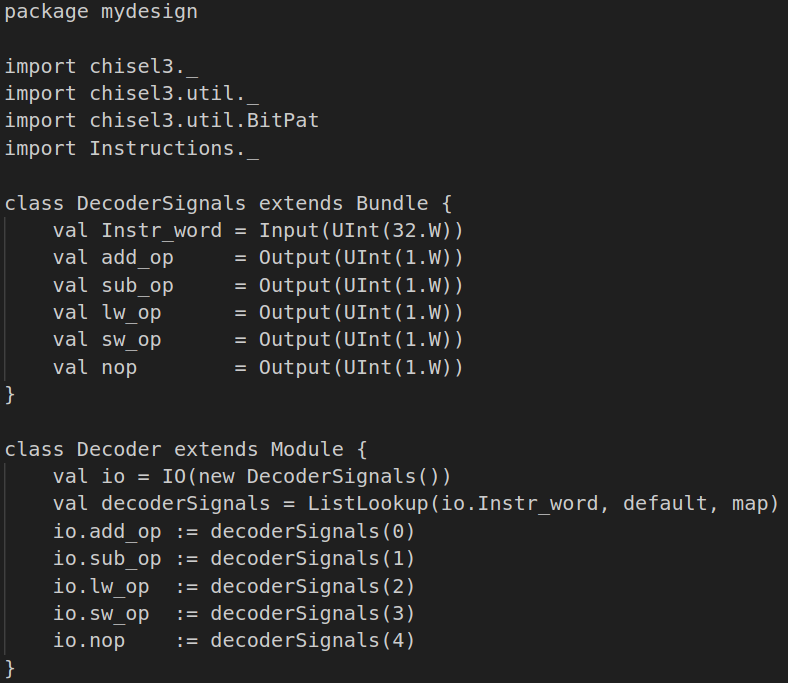


图2 译码器模块代码

②map实现，对于每种op，定义on和off两种类型，然后定义四种操作对于的32bit二进制码（除去nop操作），当输入非以上四种操作的二进制码时返回的是nop。对于map中的输入，根据类型选择对应op为on或off

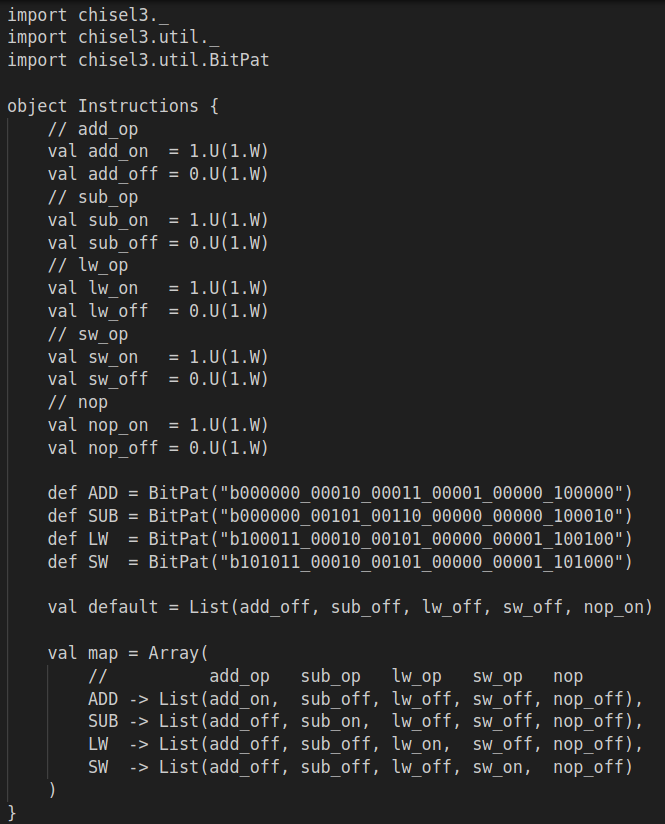


图3 各指令下的op状态

③测试代码，每个时钟周期测试一种类型的输入，在该输入下检测各个op的输出结果，若与expect的结果一致则通过测试

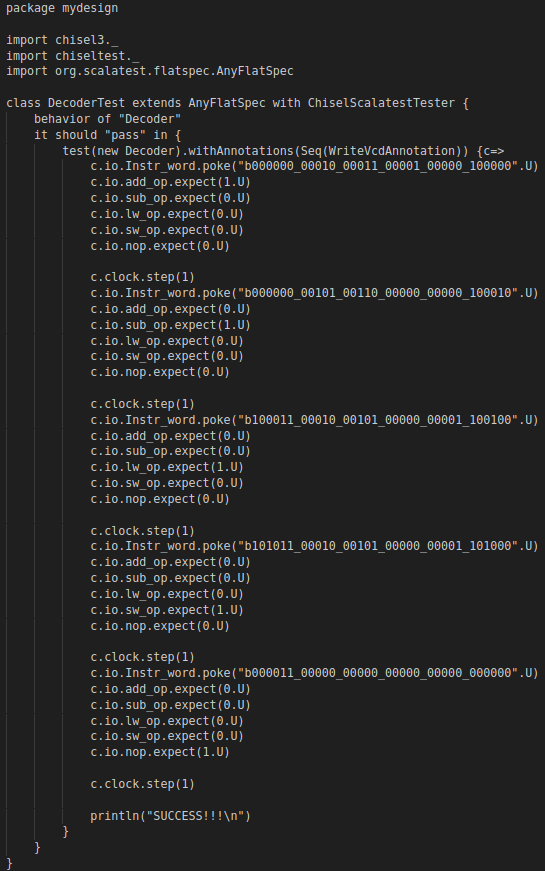


图4 测试译码器功能代码

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。五个输入信号为RS1、RS2、WB\_data、Reg\_WB、RF\_WrEn，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，RF\_WrEn=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[4:0]

RS1\_out[31:0]

RS2\_out[31:0]

RF\_WrEn

图5 寄存器堆

①定义一个名为RegFileIO的Bundle，包含寄存器文件的输入输出端口，定义一个名为RegFile的Module，实现寄存器文件的行为，初始时各个寄存器的值与其下标值对应，RS1和RS2是寄存器的地址，当不为0，则输出对应寄存器的值。当RF\_WrEn为1则将数据写入Reg\_WB指向的寄存器，代码如下：

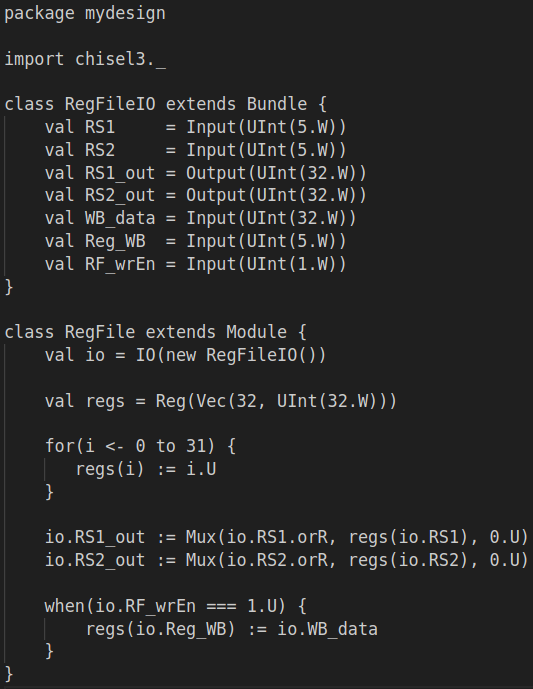


图6 寄存器堆模拟代码

②测试代码，先测试初始化之后的寄存器，检查寄存器地址与该寄存器的值是否是同一值。然后进行写入，选中一个寄存器写入数据，并在下一时钟周期检测该寄存器内的值是否与前面写入的值一致

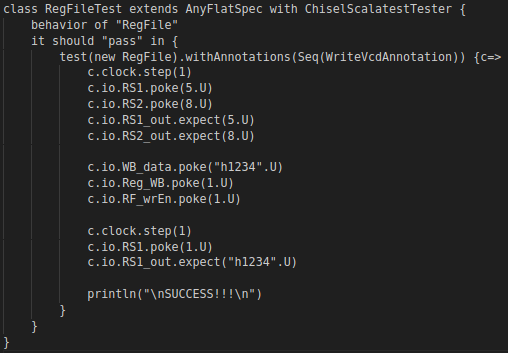


图7 寄存器堆测试代码

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），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。

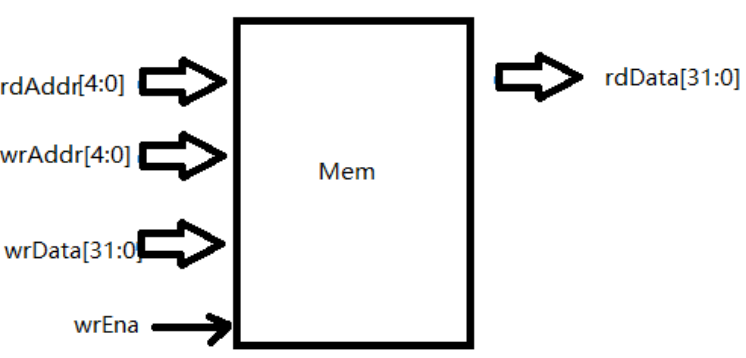


图8 指令寄存器

①定义了一个名为Mem的Chisel模块，读入地址并在地址处写入数据，使用 SyncReadMem() 定义一个32x32位的同步读内存，具有独立的读写地址和写使能信号。将add R1,R2,R3; sub R0,R5,R6，lw R5,100(R2), sw R5,104(R2)四条指令顺序存储，当识别到对应输入则输出二进制指令

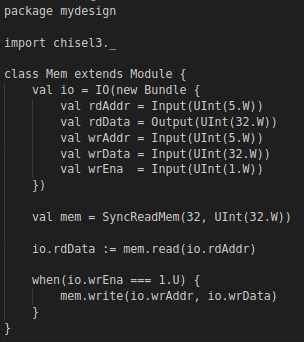


图9 指令寄存器代码

②测试代码，每个时钟周期测试一种类型，例如输入0则测试指令是否为add R1,R2,R3对应的32bit二进制码，具体代码如下：

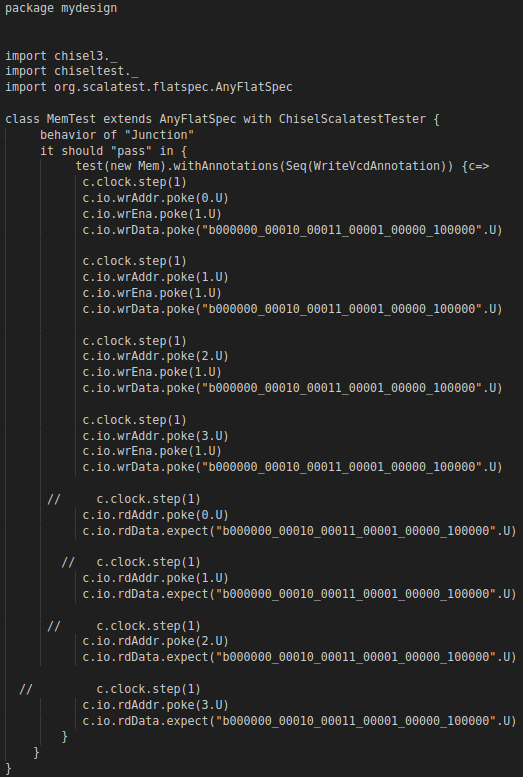


图10 指令寄存器验证代码

1. 各个模块的连接，将上方的译码器、寄存器文件、指令存储器相连接，最终让电路能逐条指令取出、译码

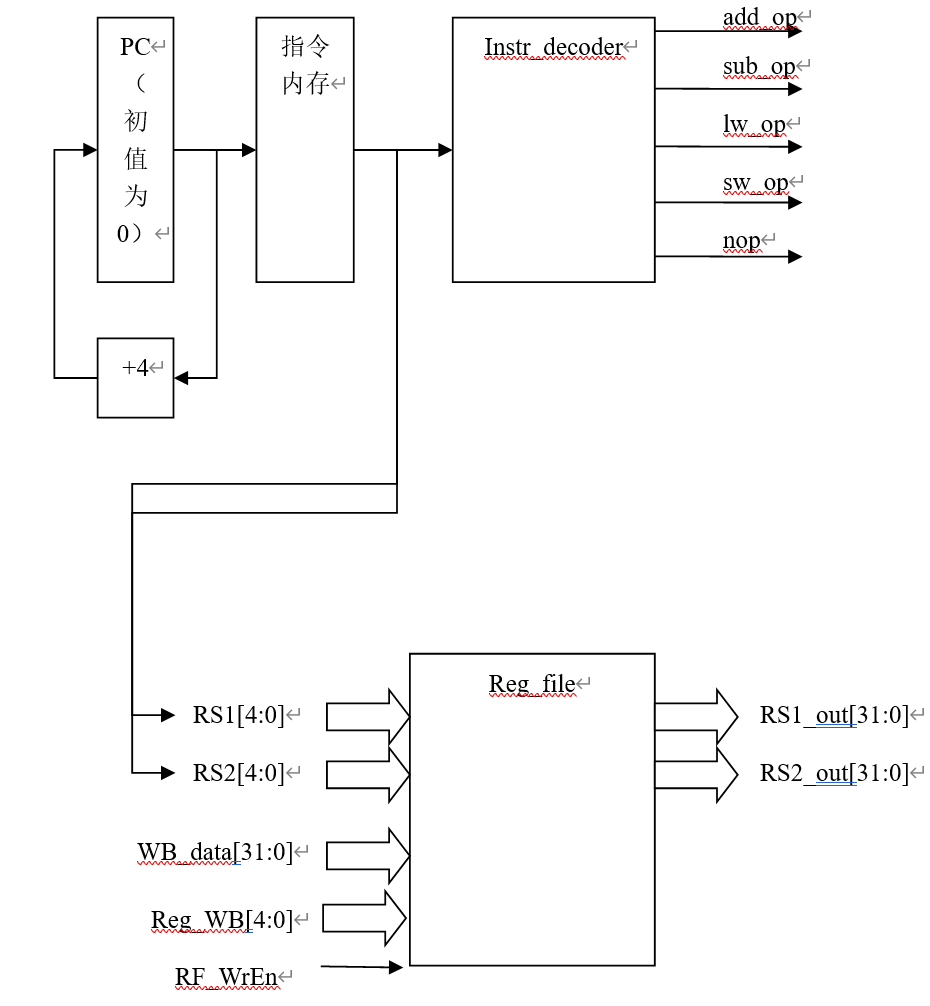


图11 模块连接

①实例化了Decoder、RegFile、Mem和PC四个模块。该模块协调指令的读取、解码。它实现了部分基础的处理器核心，可以取出部分指令集，例如加法、减法、读写和存储操作。

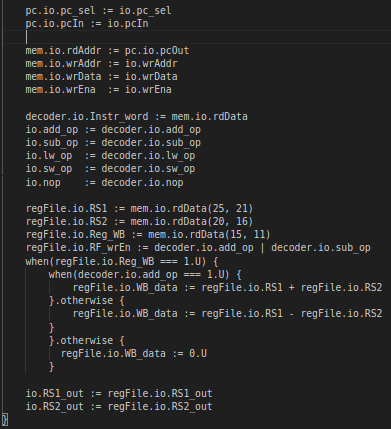
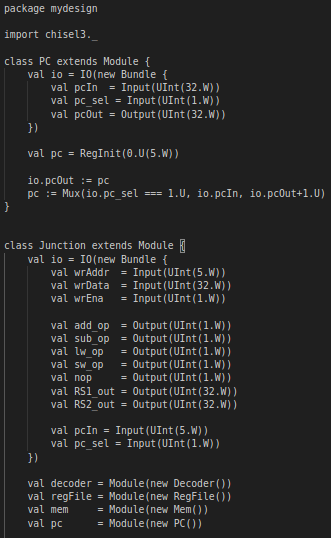


图12 实例化代码

②测试代码，测试模块功能

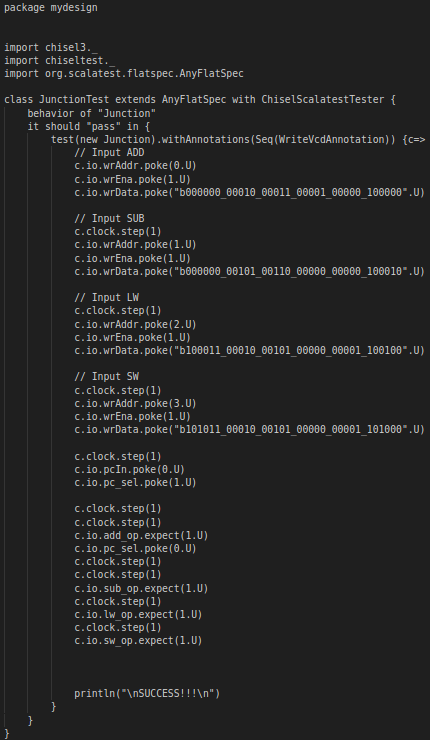


图13 功能测试代码

**五、实验结果**

使用sbt test命令运行测试代码，测试结果如下，可以看到通过了所有测试，说明模块设置正确

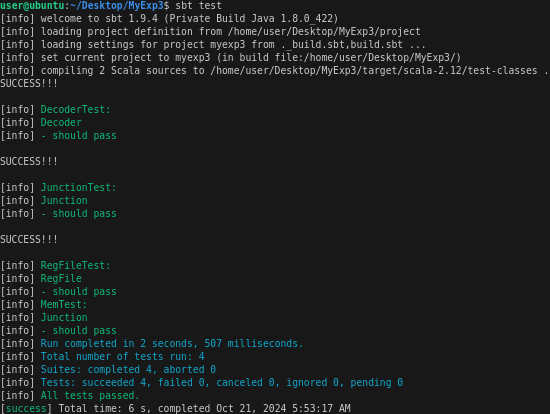


图14 验证正确代码

1）观察输入Instr\_word为add R1,R2,R3; sub R0,R5,R6，lw R5,100(R2), sw R5,104(R2)、JAL 100时，对应的输出波形。如图所示，将以上指令写成16进制形式即为图中Instr\_word的取值，可以看到，在对应情况下，只有对应的op的取值为高。

对应关系如下：

add R1, R2, R3 的十六进制为 00430820

sub R0, R5, R6 的十六进制为 00A60022

lw R5, 100(R2) 的十六进制为 8C450064

sw R5, 104(R2) 的十六进制为 AC450068

jal 100 非上述指令统一归类为nop，十六进制为 0C000000

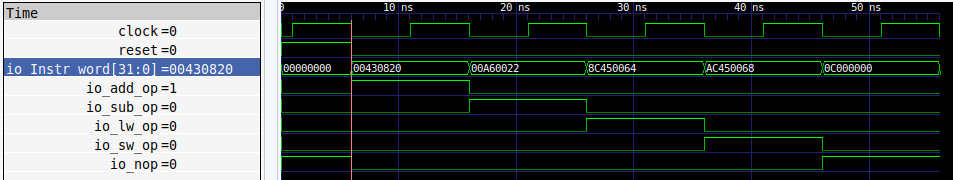


图15 输出波形

2) 观察RS1=5，RS2=8，WB\_data=0x1234，Reg\_WB=1，RF\_WrEn=1的输出波形和受影响寄存器的值，如图所示，分析可知，在这个时候寄存器为可写入状态，写入的寄存器为1号寄存器，写入的数据为0x1234，在图中，数据会在下一个时钟周期时进行写入，因此可以看到在下一个上升沿到来时寄存器1中的值为0x1234



图16 寄存器波形

3) 观察四条指令的执行过程波形，记录并解释其含义。

①add\_op：在reset之后io\_wrAddr被初始化为0，此处默认选择的是add\_op指令，因此在解码后io\_wrData的值为00430820，即指令对应的16进制数，当下一上升沿时，add\_op指令被置为高，即当前的指令是add\_op

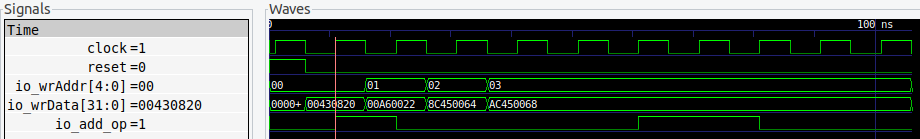


图17 add指令波形

②sub\_op：在add\_op指令后，io\_wrAddr设置为1，在顺序表中1存储对应的是00A60022，即sub R0, R5, R6。因此io\_wrData取出的值是00A60022。当下一上升沿到来时io\_sup\_op被置为高。

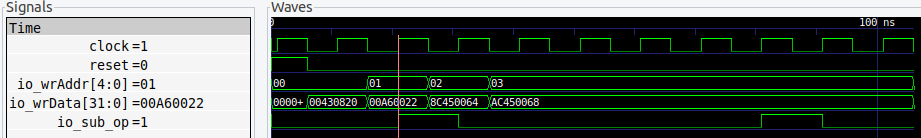


图18 sub指令波形

③lw\_op：在sub\_op指令后，io\_wrAddr设置为2，在顺序表中2存储对应的是8C450064，即lw R5, 100(R2)。因此io\_wrData取出的值是8C450064。当下一上升沿到来时io\_lw\_op被置为高。

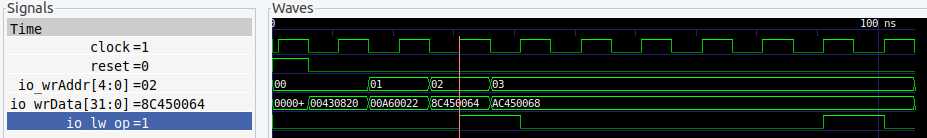


图19 lw指令波形

④sw\_op：在lw\_op指令后，io\_wrAddr设置为3，在顺序表中3存储对应的是AC450068，即sw R5, 104(R2)。因此io\_wrData取出的值是AC450068。当下一上升沿到来时io\_sw\_op被置为高。

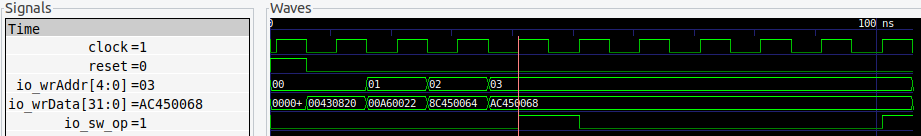


图20 sw指令波形

④完整过程展示，可以看到随着wrAddr取值的变化，在各个时钟周期下op指令都出现了变化，并且每个时钟周期最多只有一个op指令为高，此时高电平的指令即为选中的指令。

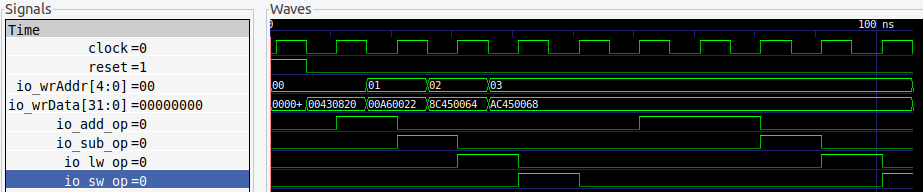


图21 四种指令完整运行周期波形

**六、实验总结与体会**

通过本次实验学习如何使用Chisel模拟电路，成功设计了一个连续读取指令的电路。理解了Scala中的IO输入，对于数据类型有了更多了解，对硬件层面也初步进行了学习。

实验主要包括三个部分，分别是设计译码器、寄存器文件、指令寄存器。首先是设计译码器，根据32bit输入数据选择对应的op指令，通过一个map来进行选择，结合硬件的io输入即可实现。

然后是寄存器文件，实现了两读取一写入的寄存器堆。接着实现了指令寄存器，模拟地址与指令的存储关系，即取指过程。将这三个部分结合在一起即可实现cpu的取指、译码的过程。这模拟的是数据通路，是硬件的实现过程。

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