**深 圳 大 学 实 验 报 告**

**课 程 名 称： 计算机系统(3)**

**实验项目名称： 新增指令实验**

**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12月21日**

**实验报告提交时间： 2024年1月14日**

**教务处制**

**一、 实验目标**

了解RISC-V mini处理器架构，在其基础之上新增一个指令，完成设计并观察指令执⾏。

**二、实验内容**

1）修改数据通路，新增指令comb rs1,rs2,rd（采用R型指令格式），实现将rs1高16位和rs2低16位拼接成32位整数，并且保存到rd寄存器。

2）在处理器上执行该指令，观察仿真波形，验证功能是否正确。

3）自行设计其他功能指令，并验证设计是否正确。

**三、实验环境**

硬件：桌面PC

软件：Chisel开发环境

**四、****实验步骤及说明**

学习Chisel数据通路的Chisel描述，特别是指令译码部分和core核心代码。然后按照下面指引完成指令译码器的修改，以及数据通路的修改。

分析：添加新指令 comb ，首先需要根据riscv指令格式，设置该指令各个字段的值，并在相应文件中添加该指令的比特模式。然后设置该指令的译码结果，接着在ALU中实现该指令的功能。最后让该指令在处理器上执行，验证功能是否正确。以下操作均在实验六下载下来的riscv-mini项目工程上进行。

**一、根据知道完成新指令实现其功能并验证。**

**1. 添加 comb 指令比特模式串**

comb 为R型指令，riscv的R型指令格式如图1所示：

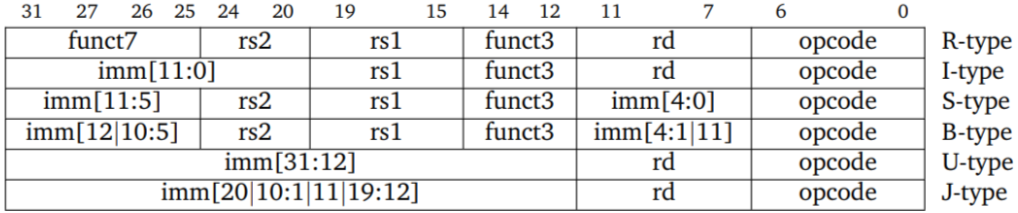


图 1 riscv指令格式

为了避免新加指令与riscv-mini已有指令冲突，这里我们将 comb 指令的opcode、funct3和funct7部分设置为0110011、111、0000001。然后使用 BitPat() 函数设置 comb 指令的比特模式。请在Instrutcions.scala中添加comb指令比特模式串，并截图。

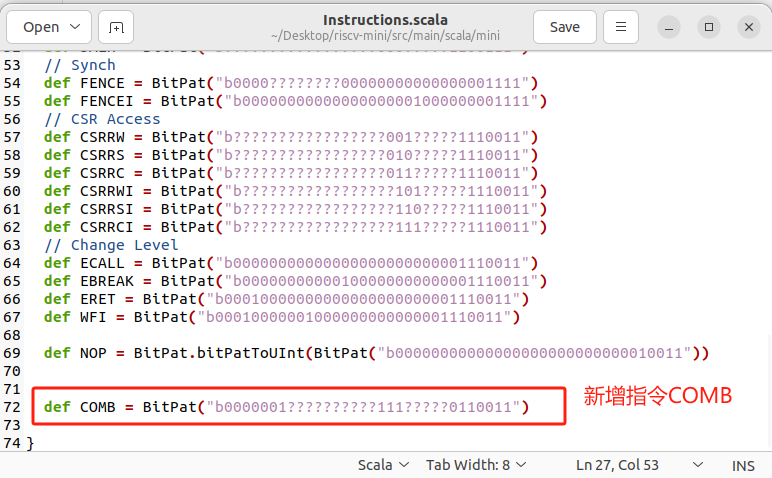


图 2 在Instructions.scala中添加comb指令比特模式串

**2. 添加 comb 指令的译码**

comb 指令需要在ALU中将rs1高16位和rs2低16位拼接成32位整数，因此需要在Alu.scala文件中添加常量 ALU\_COMB ，让译码器可以译码出正确的信号。请在Alu.scala中添加该常量，并截图。

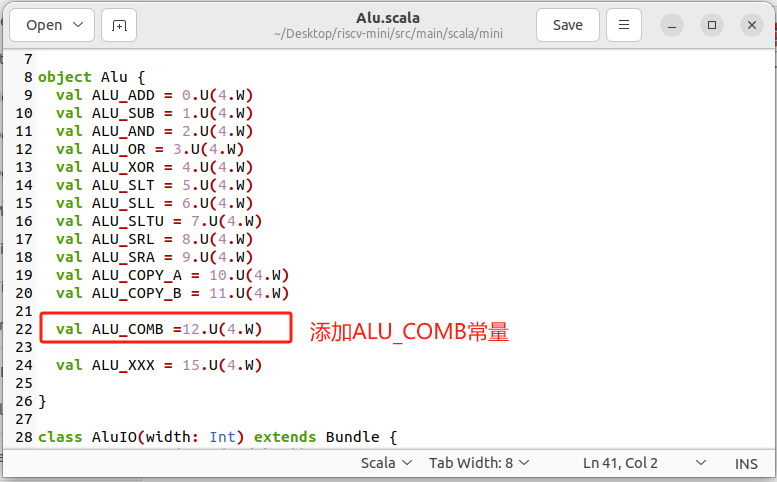


图 3 在Alu.scala中添加ALU\_COMB常量

接下来在Control.scala文件中为 comb 指令添加对应的译码映射。comb 指令执行后pc需要加4，并将从寄存器文件中读取的数据rs1和rs2进行拼接操作，然后将ALU输出的拼接结果写回到寄存器文件中。请在Control.scala中添加comb指令的译码映射，并截图。

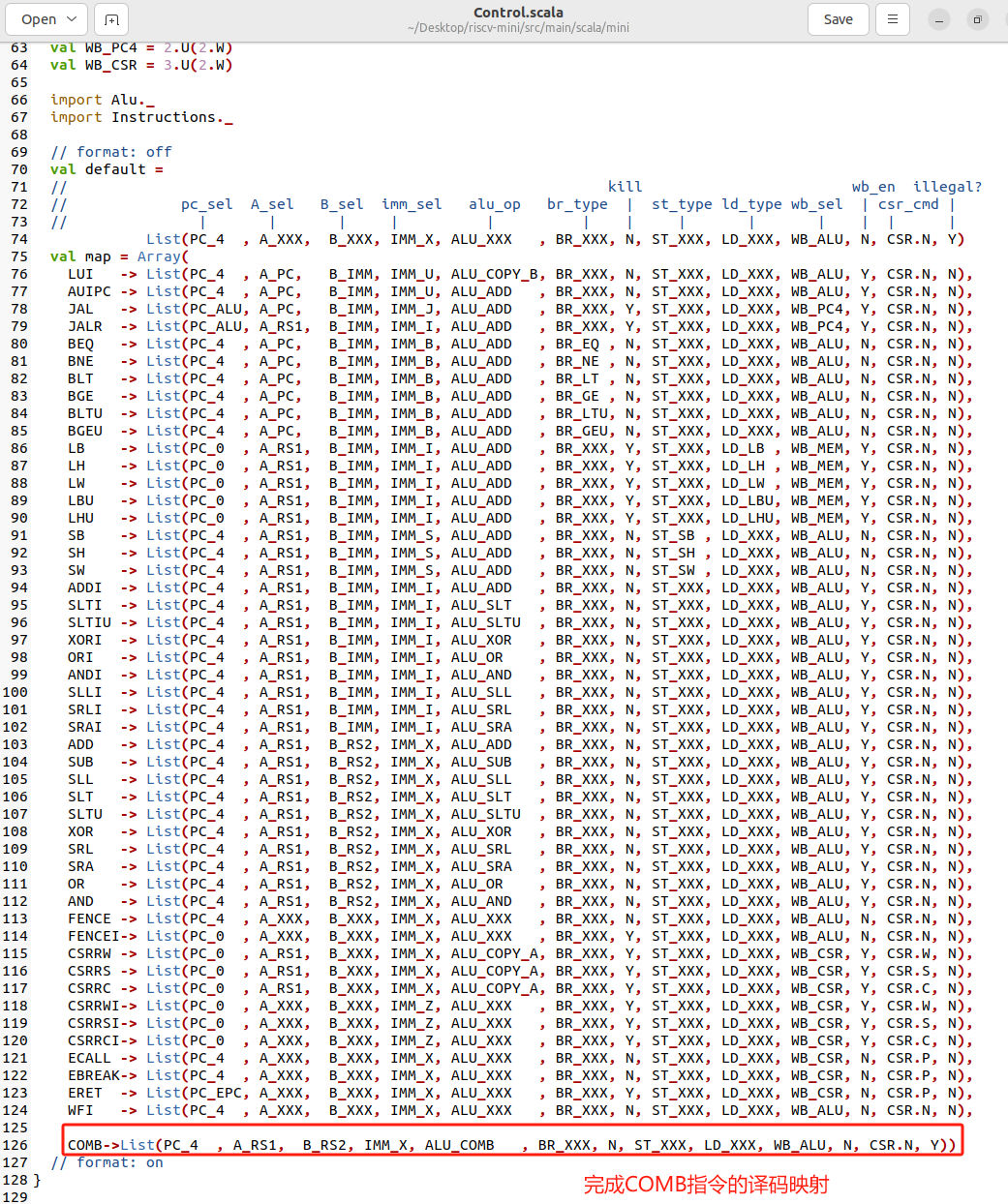


图 4 在Control.scala中添加comb指令的译码映射

**3. 实现 comb 指令的执行操作**

在Alu.scala文件添加将rs1高16位和rs2低16位拼接成32位整数的操作。请在Alu.scala中实现该操作，并截图。

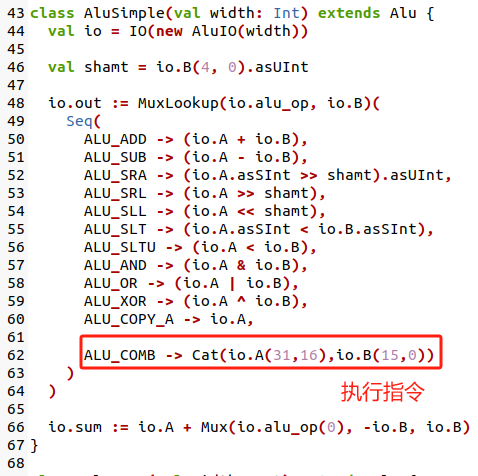
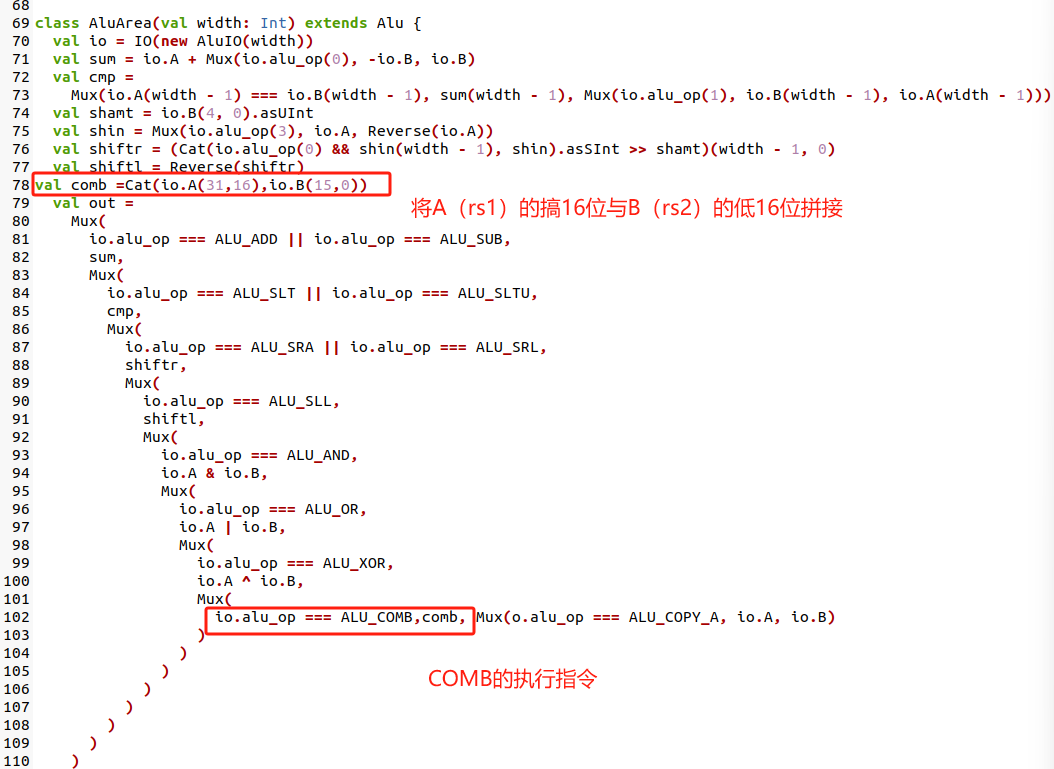


图 5 在Alu.scala中的AluSimple添加comb指令执行的逻辑

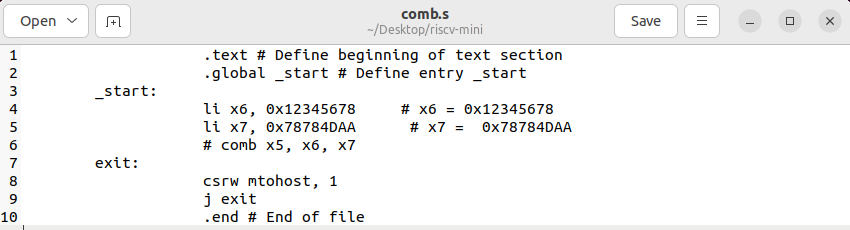
图 6 在Alu.scala中的AluArea添加comb指令执行的逻辑和多选器逻辑

**4. 对comb指令进行测试**

参照实验六第二部分第一节，尝试编写一个汇编程序comb.s，对comb指令进行测试。这里给出一个样例代码。在下面代码中，x6寄存器中的数据为0x00001000，x7寄存器中的数据为0x00002000，comb命令将x6的高16位（即0x0000）和x7的低16位（即0x2000）拼接成一个32位的整数（即0x00002000）。

1. .text *# Define beginning of text section*
2. .global \_start *# Define entry \_start*
3. \_start:
4. lui x6, 1 *# x6 = 0x00001000*
5. lui x7, 2 *# x7 = 0x00002000*
6. *# comb x5, x6, x7*
7. exit:
8. csrw mtohost, 1
9. j exit
10. .end *# End of file*

自定义的数0x12345678加载进x6寄存器,将我的学号2021150122转换为16进制，得到0x78784DAA，装载进x7寄存器。最终使用comb命令得到的结果应该是0x1234DAA。自己编写的程序代码如下。因为lui指令用于将一个20位的立即数加载到目标寄存器的高20位，同时将低12位清零。所以只能将0x12345装载至前20位。li并不是一个独立的指令，而是一种伪指令，用于加载一个立即数到寄存器中。它实际上会被汇编器转化为适当的指令序列。译器或汇编器会将这个指令转换为适当的指令序列，通常是lui（加载高位）和addi（加载低位）的组合。



其中，rd是目标寄存器，immediate是要加载的立即数。编译器或汇编器会将这个指令转换为适当的指令序列，通常是lui（加载高位）和addi（加载低位）的组合。

因为 comb 为自己加入的指令，不能被汇编器汇编，所以这里先将其注释掉，到后面生成的comb.hex文件中再将 comb x5, x6, x7 的二进制添加进去。

编写完程序后，使用riscv32-unknown-elf-gcc编译得到comb二进制文件：

riscv32-unknown-elf-gcc -nostdlib -Ttext=0x200 -o comb comb.s

 然后使用elf2hx将编译得到的comb二进制文件转换成16进制文件comb.hex：

elf2hex 16 4096 comb > comb.hex



在comb.hex文件中，可以找到li x6, 0x12345678和li x7, 0x78784DAA的机器码对应的十六进制形式，如图7所示：

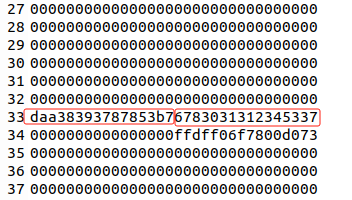


图 7 在comb.hex中找到li x6, 0x12345678和li x7, 0x78784DAA的十六进制机器码

comb x5, x6, x7 转换成机器码的十六机制形式为 027372b3。因此处指令存储为小端模式，故我们需要将十六进制数插入到第一个红线的前面。修改后如图8所示：

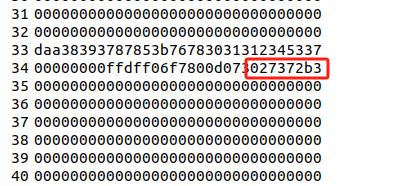


图 8 将comb x5, x6, x7汇编的十六进制机器码插入到comb.hex中

接着需要在主目录下一次执行 make 和 make verilator 命令（若之前已经执行过，则在此次操作之前需要执行 make clean），执行后会产生VTile可执行文件。然后执行下面命令，使mini处理器执行新建指令并产生波形文件。

1. $ ./VTile comb.hex comb.vcd



使用GTKWave打开comb.vcd文件，其波形图如图9所示：

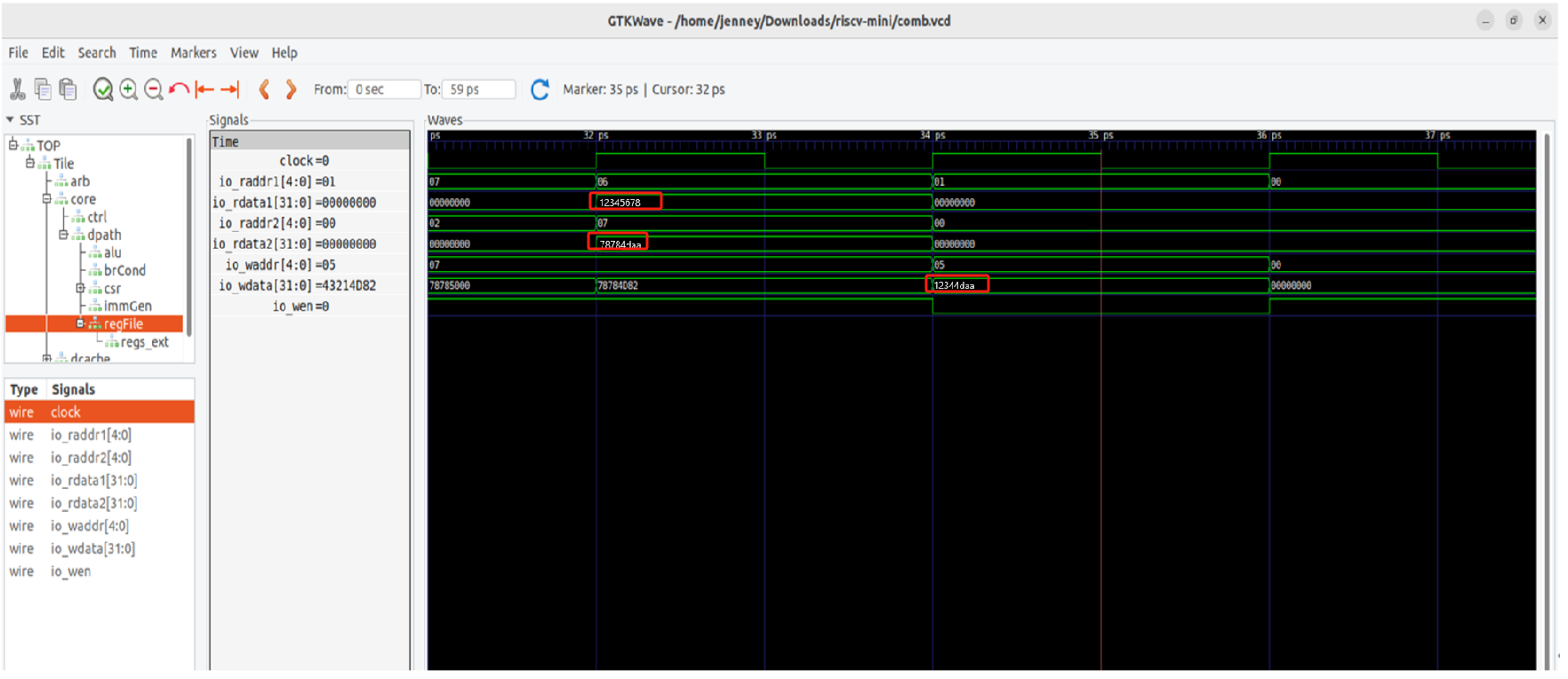


图 9 comb程序的波形图

指令对应的十六进制形式见表1所示：

表 1 核心指令对应的十六进制形式

|  |  |  |
| --- | --- | --- |
| **指令** | **十六进制表示** | **说明** |
| li x6, 0x12345678 | 6783031312345337 | x6 = 0x12345678 |
| li x7, 0x78784DAA | daa38393787853b7 | x7 = 0x78784DAA |
| comb x5, x6, x7 | 027372b3 | x5 = cat(x6(31:16), x7(15:0)) |

从波形图中可以看出，li指令成功将x6寄存器赋值为0x12345678，x7寄存器赋值为0x78784DAA，comb 指令将拼接后的结果0x12344DAA写回到了5号寄存器中，故该指令执行正常。

**二、自行设计新指令实现其功能并验证。**

设计一条乘法指令，计算两数的乘积。

1. 在scala目录下的mini文件夹中创建MDU.scala，并在其中定义操作码。

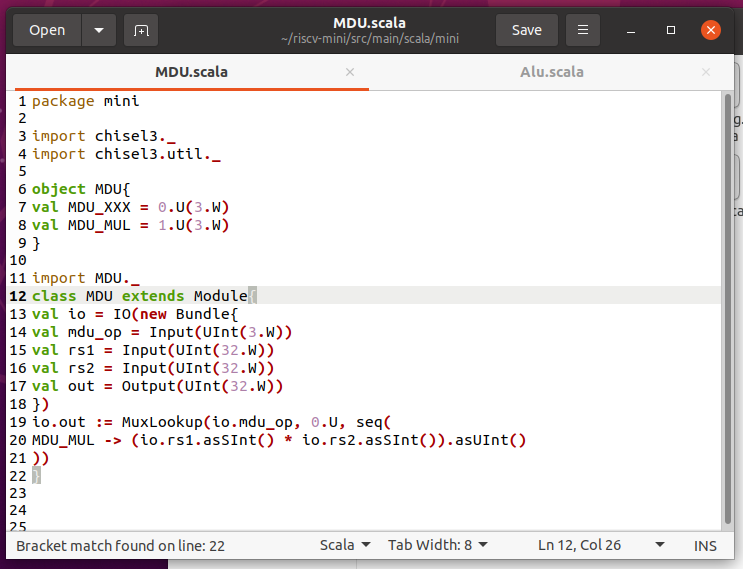


图10 自己设计MDU.scala

2. 在Instructions.scala文件中添加如下BitPat。

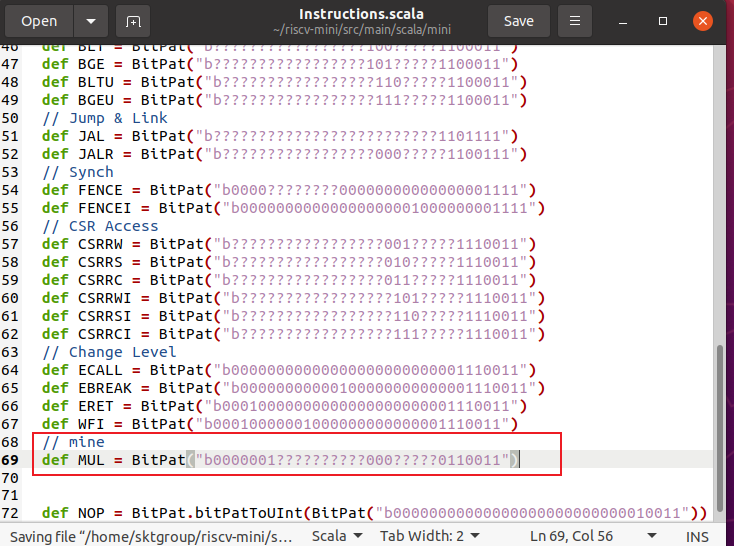
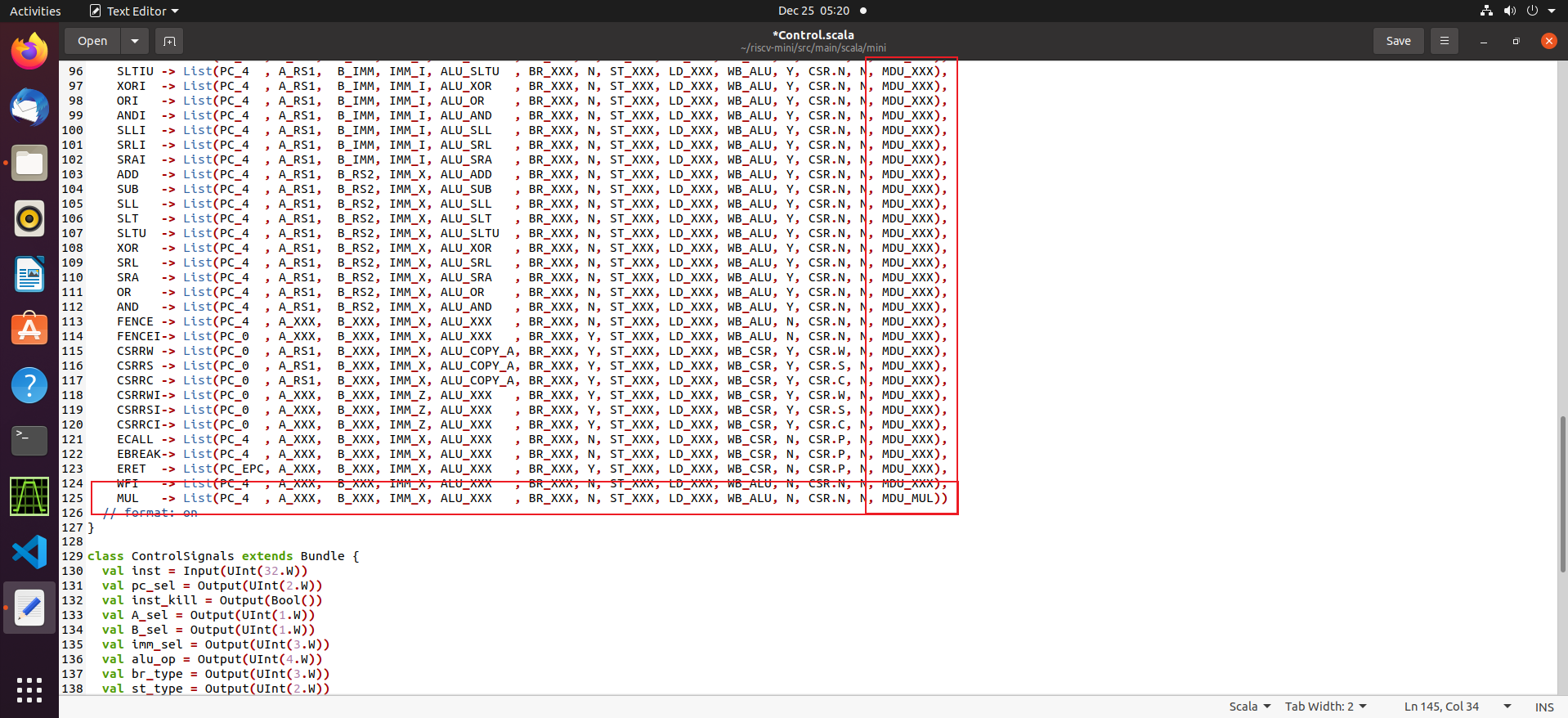
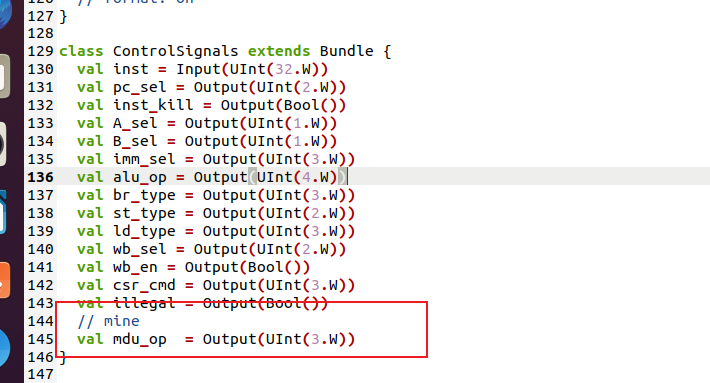


图11 修改Instructions.scala

3. 修改控制单元以产生乘法指令执行时的控制信号，并为 MDU 指令添加对应的译码映射。

对Control.scala文件进行修改





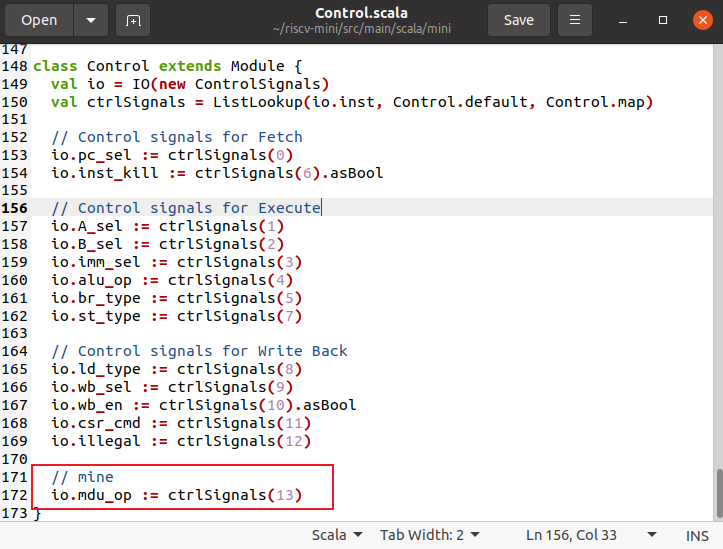
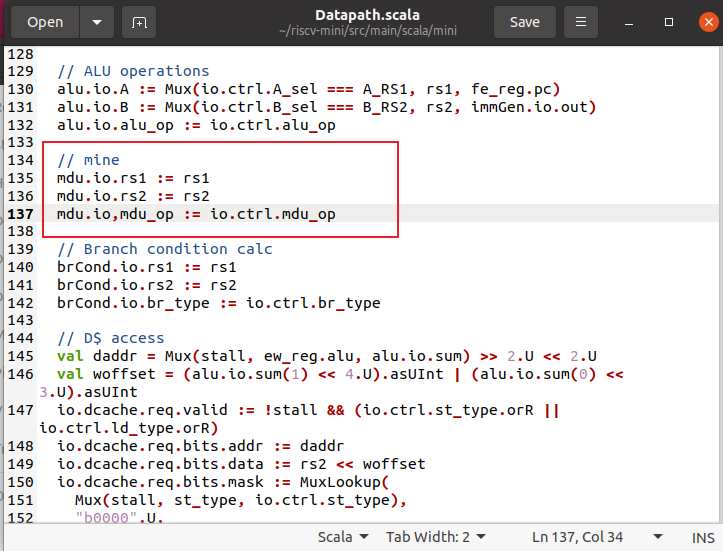


图12 修改Control.scala

4. 对Datapath.scala进行修改添加MDU模块以及相关电路。



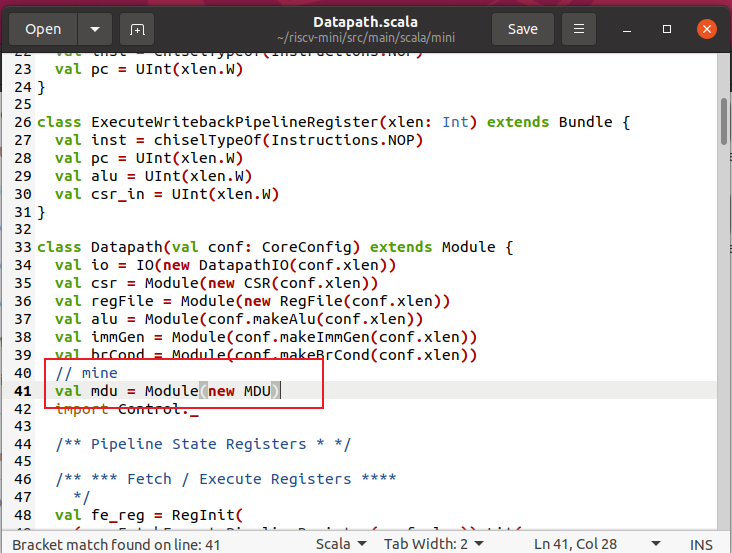


图13 修改Datapath.scala

5. 编写测试程序mul.s。



图14 mul.s代码

6. 生成hex和vcd文件

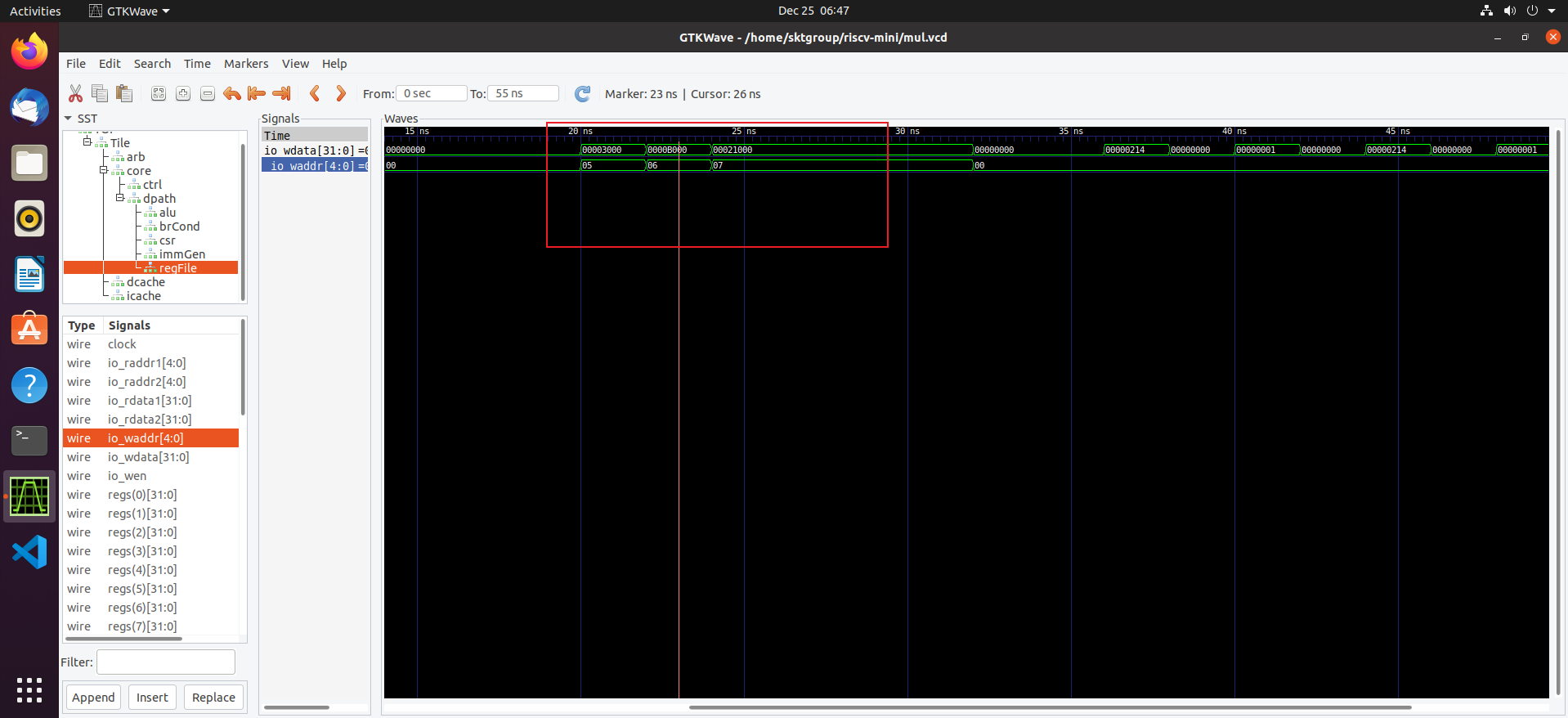






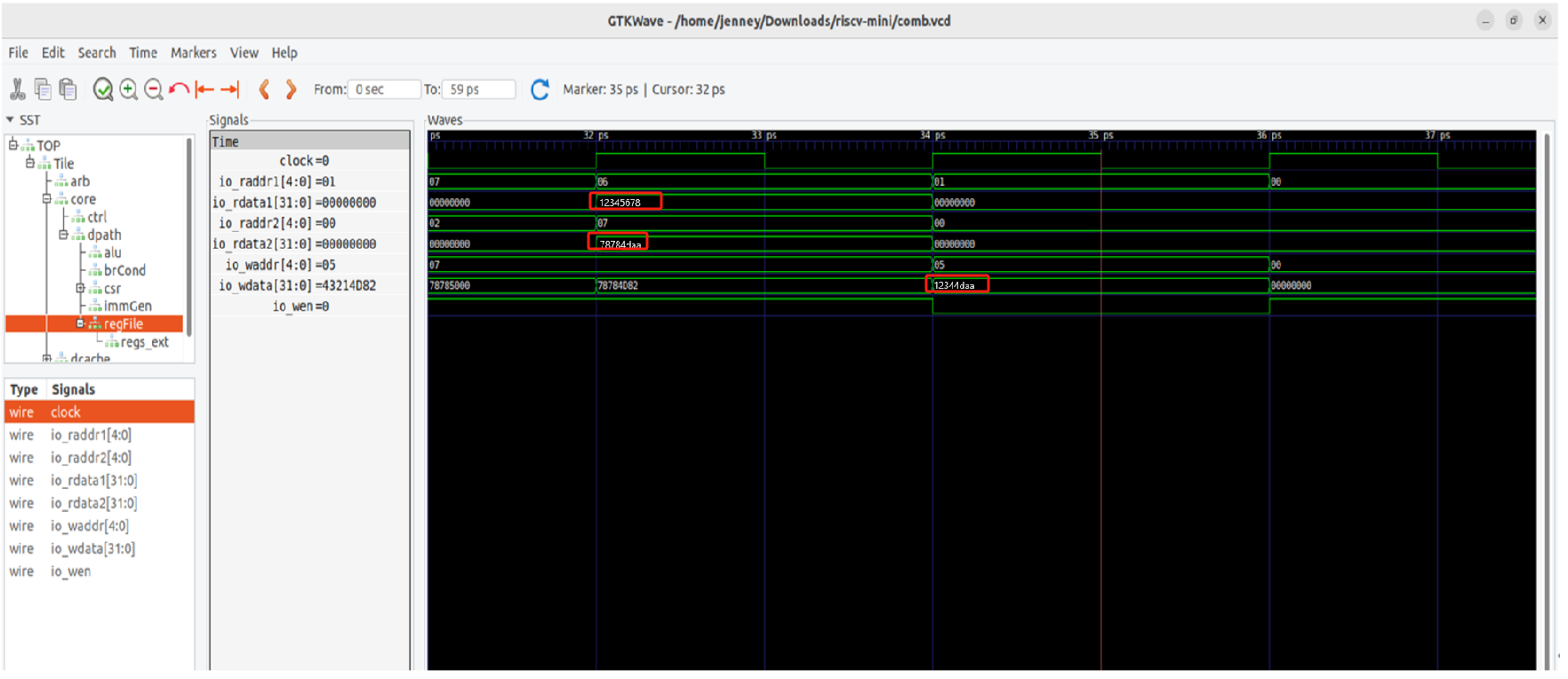
7. 验证

程序中x5 = 3，x6 = 11，结果应该为33(0x00000021)。

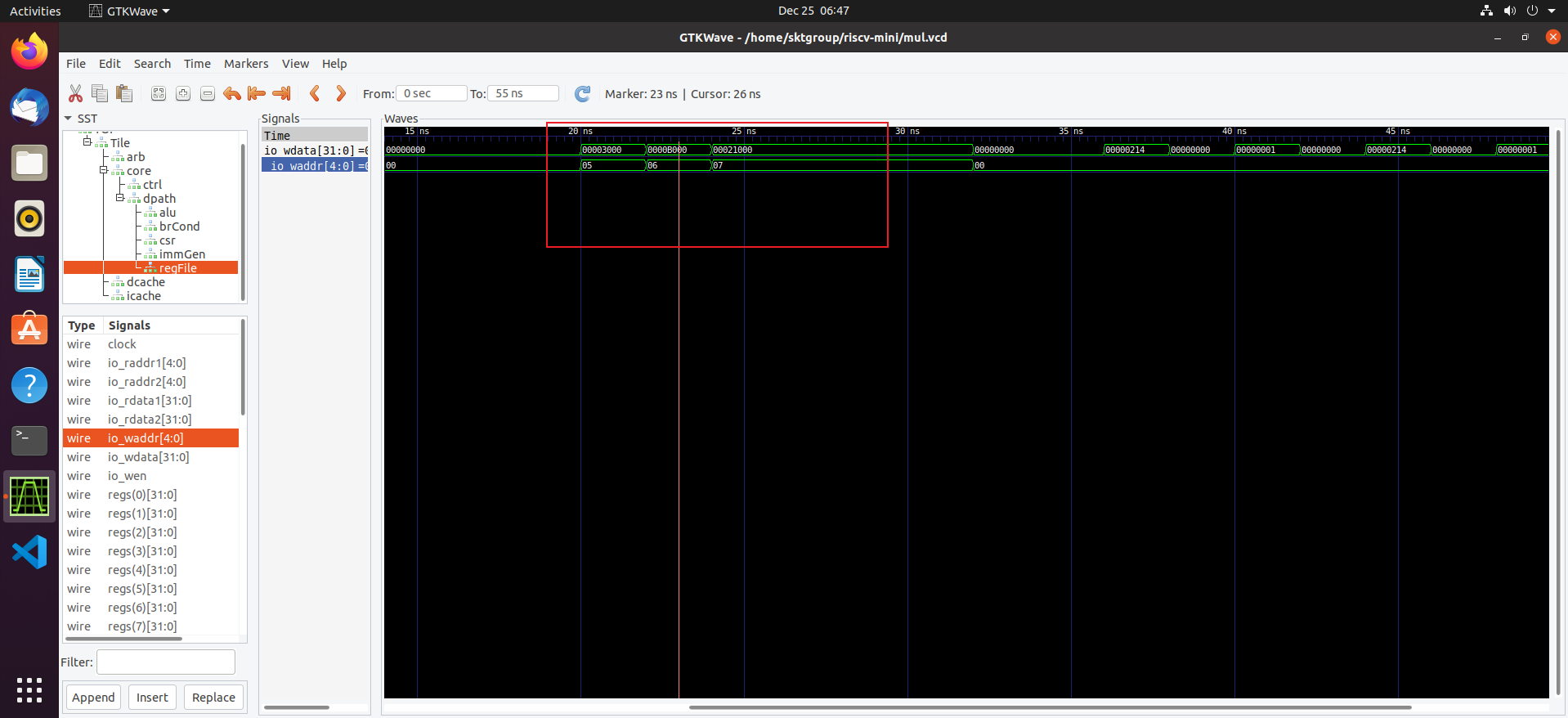


**五、实验结果**

在第一部分中，我按给出的实验步骤，成功完成了comb指令的设计，并对其功能进行了验证。li指令成功将x6寄存器赋值为0x12345678，x7寄存器赋值为0x78784DAA，comb 指令将拼接后的结果0x12344DAA写回到了5号寄存器中，增加指令成功。



在第二部分中，我仿照第一部分的步骤，自己设计了一个乘法指令mul, 并对其功能进行了验证。程序中x5 = 3，x6 = 11，结果为33(0x00000021)，设计成功。



**六、实验总结与体会**

在本次实验中，我学习了使用 Chisel 描述数据通路，并重点关注了指令译码部分和核心代码。根据参考文档的指引，我成功完成了 comb 指令的实现，并对其功能进行了验证。此外，我还设计并实现了一个新的指令，并对其功能进行了验证。

通过本次实验可以总结出添加新指令的步骤：

首先，根据 RISC-V 指令格式，设置新指令各个字段的值，并在相应的文件中添加该指令的比特模式。这样可以确保新指令在编码过程中能够被正确识别和解析。

接下来，设置新指令的译码结果。通过在译码阶段将新指令与相应的操作码进行匹配，可以确保在执行阶段能够正确地识别并处理新指令。

然后，在 ALU（算术逻辑单元）中实现新指令的功能。根据新指令的操作码和操作数，在 ALU 中编写逻辑代码，以实现新指令所定义的操作和计算逻辑。

最后，让新指令在处理器上执行，并验证其功能是否正确。通过编写相应的测试用例，并在处理器上运行这些测试用例，可以确保新指令在实际运行中的行为与预期一致。

本次实验让我更深入地理解了 Chisel 数据通路的描述方法，并提升了我的硬件设计和编码能力。我认识到在设计新指令时的各个环节都非常重要，尤其是在译码阶段和功能实现中需要仔细

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