**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： 刘 刚**

**报告人： 吴嘉楷 学号： 2022150168 班级： 国际班**

**实 验 时 间： 2024年12月29日**

**实验报告提交时间： 2024年12月30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。然后按照下面操作完成指令译码器的修改，以及数据通路的修改，

具体操作如下：按照参考文档完成comb指令的实现，自行设计新指令实现其功能并验证。

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。

**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：

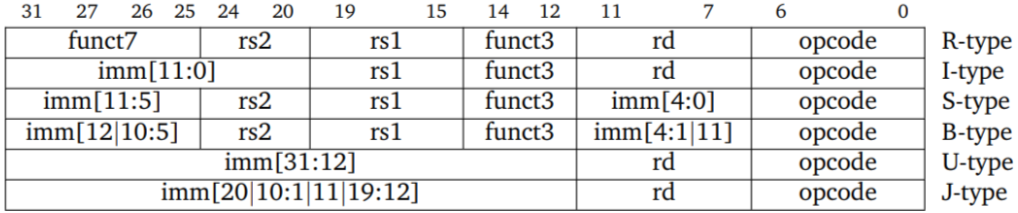


图 1 riscv指令格式

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。我们需要在Instrutcions.scala中添加comb指令比特模式串。

**在Instrutcions.scala中添加comb指令比特模式串，截图如下：**

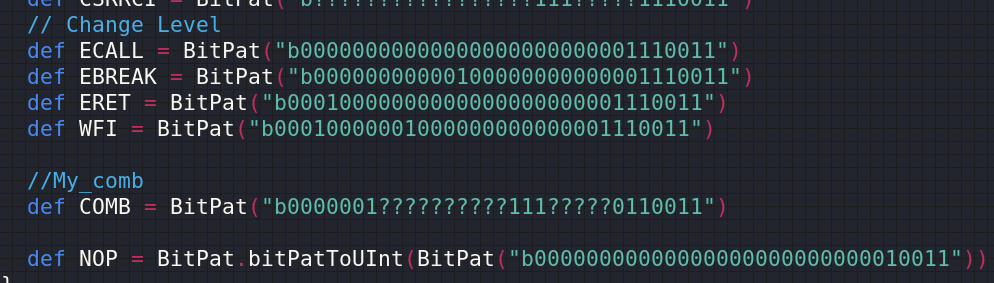


图 2 在Instructions.scala中添加comb指令比特模式串

**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。我们需要在Alu.scala中添加该常量。

**在Alu.scala中添加ALU\_COMB常量，截图如下：**

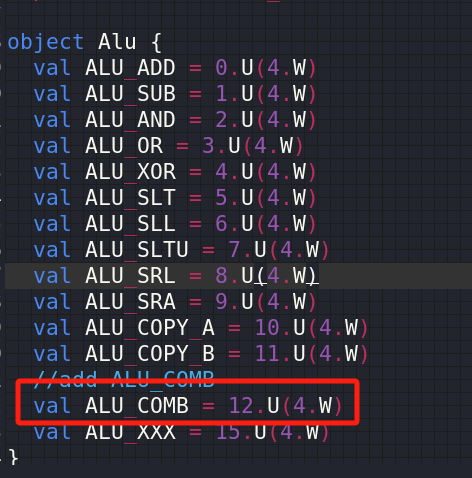


图 3 在Alu.scala中添加ALU\_COMB常量

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。我们需要在Control.scala中添加comb指令的译码映射。

**在Control.scala中添加comb指令的译码映射，截图如下：**

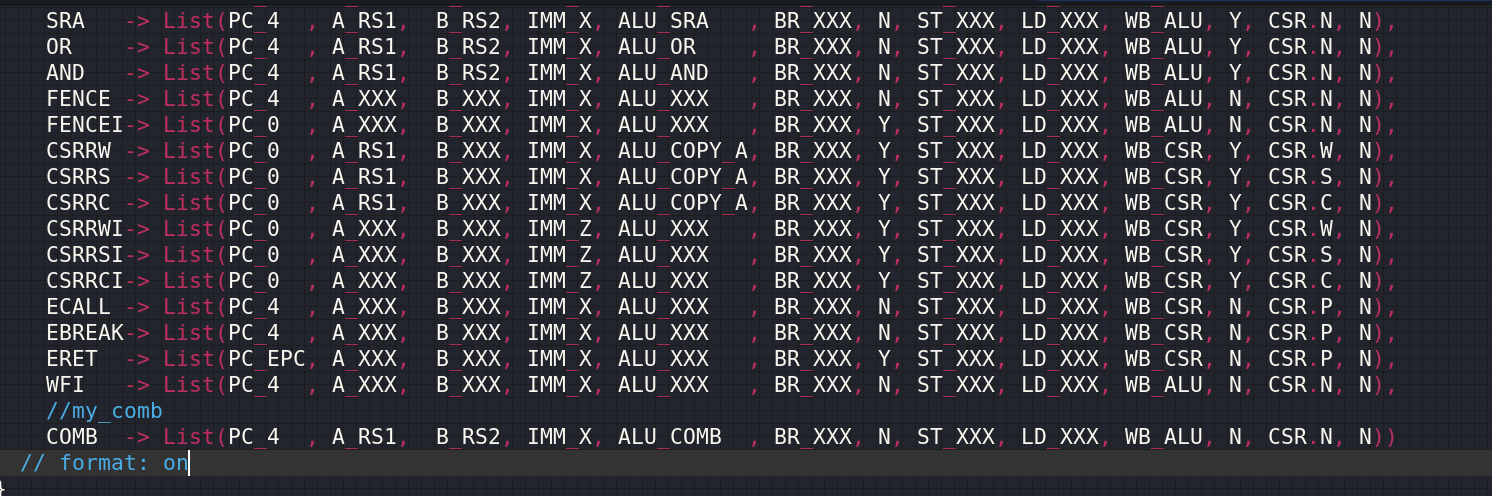


图 4 在Control.scala中添加comb指令的译码映射

**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。我们需要在Alu.scala中实现该操作。

**在Alu.scala中补全实现截图如下：**

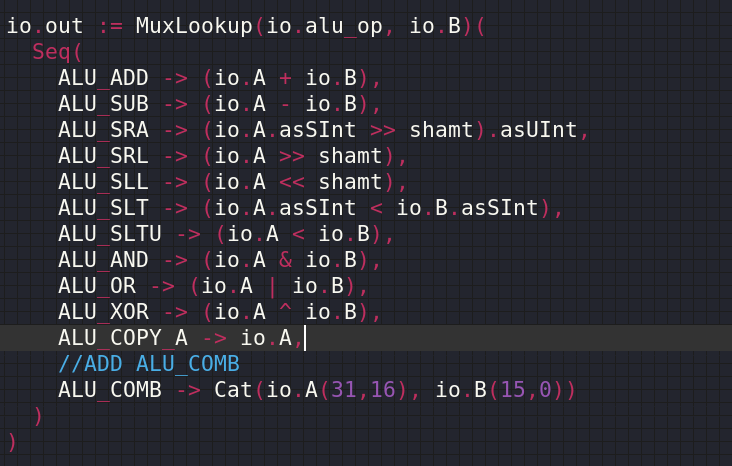


图 5 在Alu.scala中的AluSimple添加comb指令执行的逻辑

**在Alu.scala中将A(rs1)的高16位与B(rs2)的低16位拼接：**

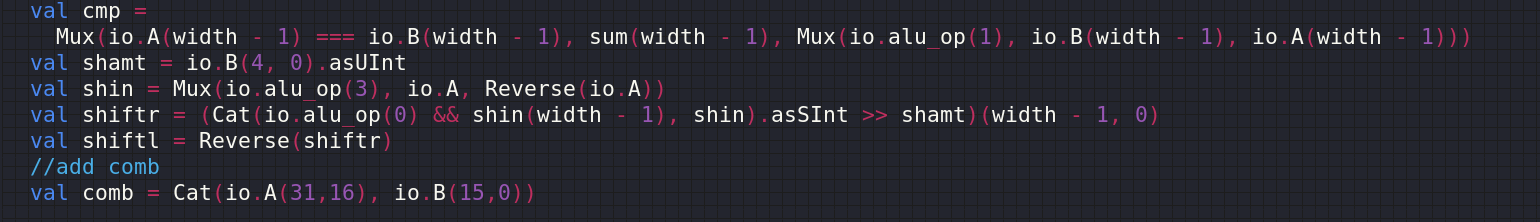


图 6 在Alu.scala中的AluArea将A(rs1)的高16位与B(rs2)的低16位拼接

**在Alu.scala中编写COMB指令执行代码：**

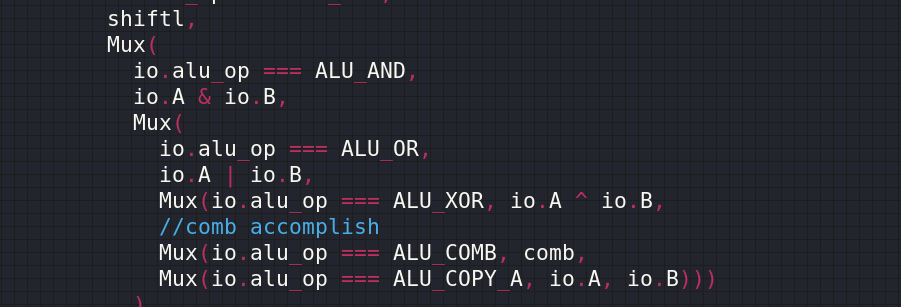


图 7 COMB指令执行

**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*

请注意，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：



图 8 gcc编译得到的comb二进制文件

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：



图 9 16进制文件comb.hex

在comb.hex文件中，可以找到 lui x6, 1 和 lui x7, 2的机器码对应的十六进制形式，如图10所示：

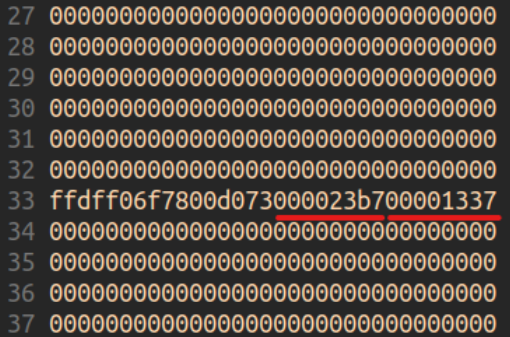


图 10 在comb.hex中找到lui x6,1和lui x7,2的十六进制机器码

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11所示：

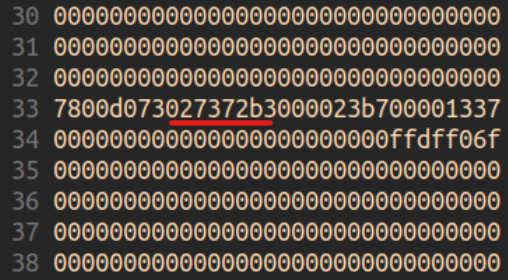


图 11 将comb x5, x6, x7汇编的十六进制机器码插入到comb.hex中

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

然后使用GTKWave打开comb.vcd文件，得到其波形图。

**在本机运行截图如下：**

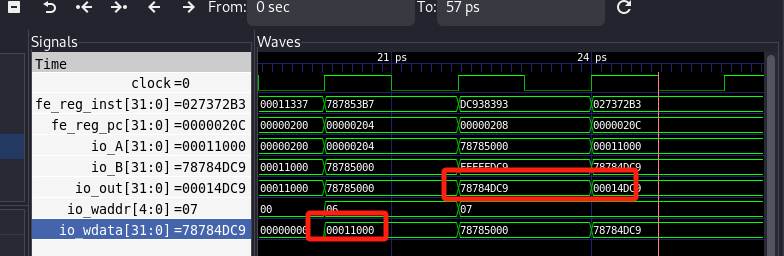


图 12 comb测试程序的波形图

如上图，首先执行lui x6, 17，向6号寄存器写入17，即0x00011000，然后执行li x7, 2021150153，其中学号转化为十六进制的值为78784DC9，那么在执行comb指令之后，会将x6的高32位0x0001和x7的低32位0x4DC9拼接，结果为0x00014DC9，可以看到上面的波形图中pc为027372b3，即comb指令位置的输出是正确的，说明指令添加成功且能正常运行。

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** |
| lui x6, 1 | 00001337 | x6 = 0x00001000 |
| lui x7, 2 | 000023b7 | x7 = 0x00002000 |
| comb x5, x6, x7 | 027372b3 | x5 = cat(x6(31:16), x7(15:0)) |

从波形图中可以看出， comb 指令将拼接后的结果0x00002000写回到了5号寄存器中，故该指令执行正常。

**五、实验结果**

1. **添加 comb 指令比特模式串**

如图13所示，在Instrutcions.scala中添加comb指令比特模式串：

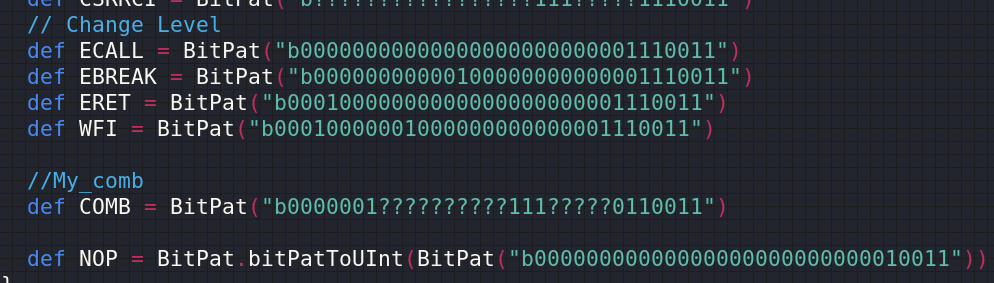


图 13 在Instructions.scala中添加comb指令比特模式串

1. **添加 comb 指令的译码**

如图14所示，在Alu.scala中添加ALU\_COMB常量：

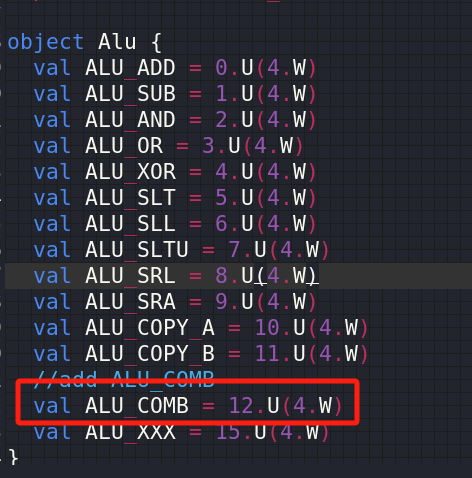


图 14 ALU\_COMB常量

在Control.scala文件中为 comb 指令添加对应的译码映射，如图15所示：

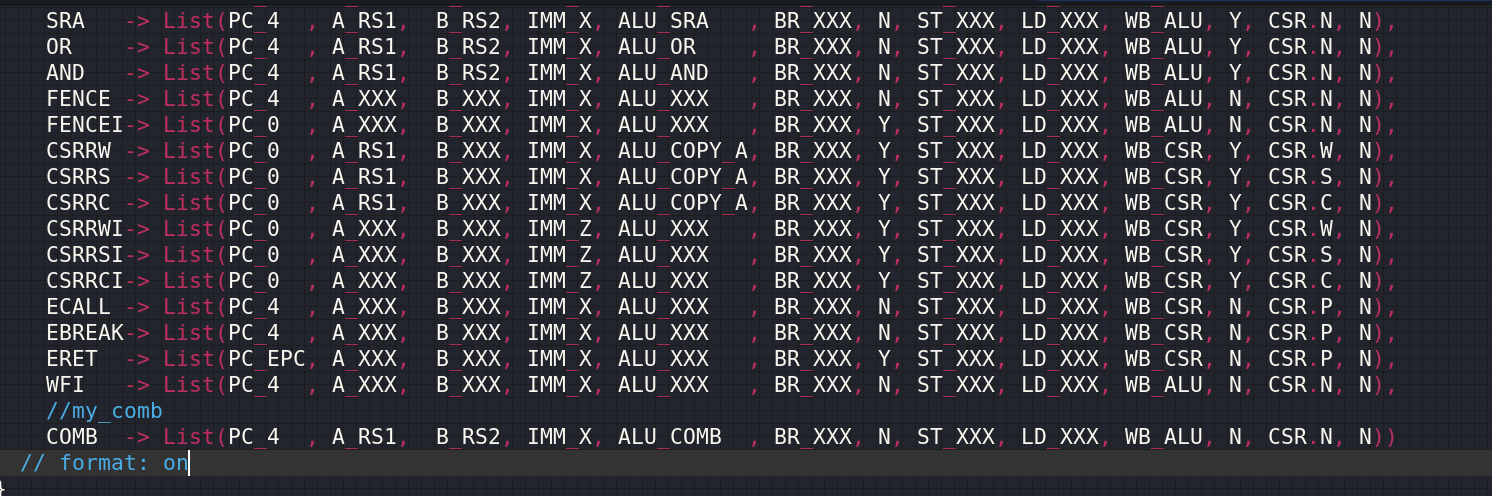


图 15 添加对应的译码映射

1. **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，如图16，17所示：

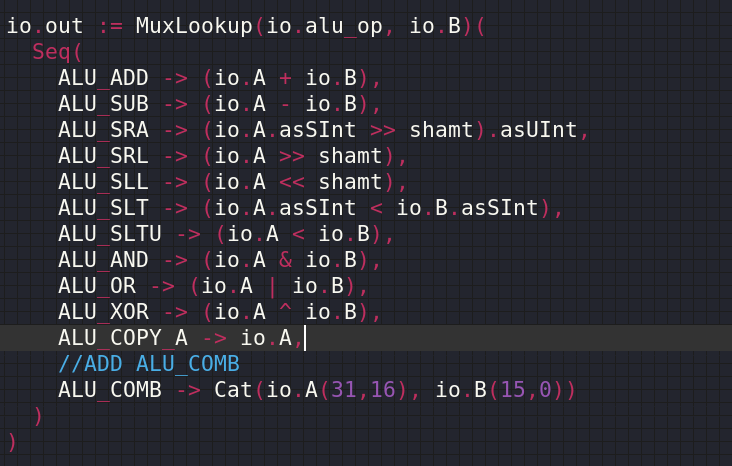


图 16 添加将rs1高16位和rs2低16位拼接成32位整数的操作

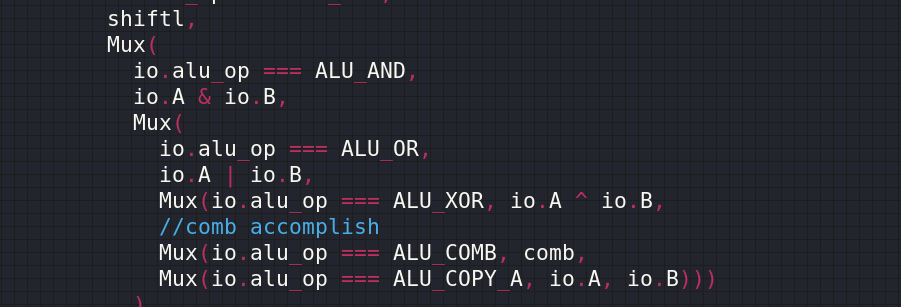
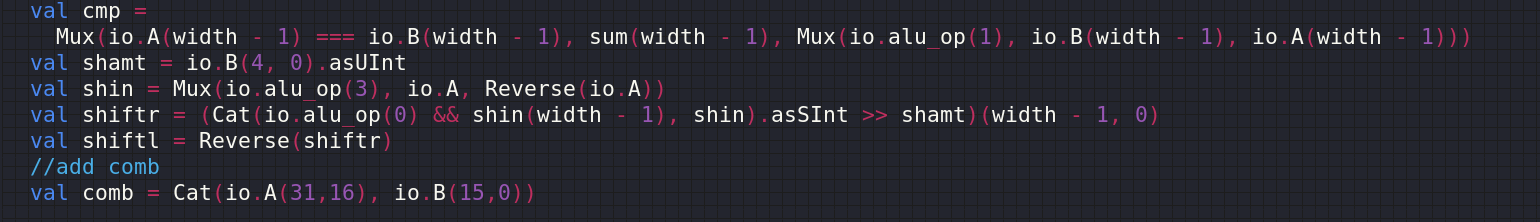


图 17 添加comb指令执行的逻辑和多选器逻辑

1. **对comb指令进行测试**

在comb.hex文件中，可以找到 lui x6, 1 和 lui x7, 2的机器码对应的十六进制形式，如图18所示。

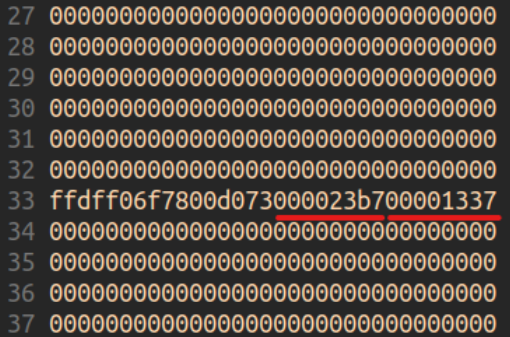


图 18 在comb.hex中找到lui x6,1和lui x7,2的十六进制机器码

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19所示：

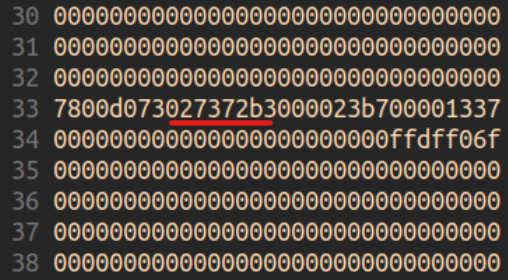


图 19 将comb x5, x6, x7汇编的十六进制机器码插入到comb.hex中

使用GTKWave打开comb.vcd文件，其波形图如图20所示：

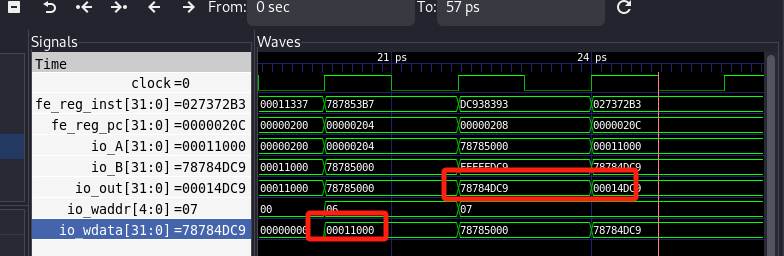


图 20 波形图

**六、实验总结与体会**

**实验总结：**

在本次实验中，我成功地在RISC-V mini处理器架构中新增了一个名为“comb”的自定义指令。该指令能够将两个寄存器rs1的高16位和rs2的低16位拼接成一个32位整数，并将其存储到目标寄存器rd中。通过修改数据通路和指令译码器，我实现了这一功能，并在处理器上执行了该指令。仿真波形的观察结果验证了comb指令的正确性，拼接后的值正确地写入了目标寄存器，表明我的实现是成功的。

通过完成本次实验，我加深了对 RISC-V mini 处理器架构的理解，学会了如何通过修改数据通路来新增自定义指令。我对 RISC-V mini 中各组件的功能有了更清晰的认识，并理解了部分代码的结构。通过波形图的调试，我提高了对 RISC-V 代码的分析能力。

**实验难点：**

1. **指令格式与冲突避免**：

在新增指令时，我需要确保comb指令的opcode、funct3和funct7字段值不会与现有的RISC-V指令冲突。这要求我深入了解RISC-V指令集的编码规则，并进行细致的规划以避免潜在的冲突。

1. **数据通路的修改**：

修改数据通路是一个复杂的过程，涉及到多个组件的协同工作。我需要精确地控制数据流，确保rs1和rs2的特定位段能够正确地拼接。这不仅需要对硬件描述语言有深入的理解，还需要对RISC-V架构的细节有充分的掌握。

1. **代码规范与调试**：

在实验过程中，我遇到了因代码编写不规范而导致的错误。这要求我严格遵守编码规范，并且在调试过程中保持耐心和细致，以确保每一行代码都能正确执行。

1. **仿真波形分析**：

仿真波形的分析对于验证指令的正确性至关重要。我需要能够解读波形图，并从中提取出关键信息，以确认指令执行的每一个步骤是否符合预期。这一过程需要较强的逻辑思维和分析能力。

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