

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： 林宪亮 学号： 2022150130 班级： 国际班

实 验 时 间： 2024.12.19-2025.1.3

实验报告提交时间： 2024.12.26

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。

五、实验结果

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` 的高 16 位和 `rs2` 的低 16 位拼接成一个 32 位整数，并将结果保存到 `rd` 寄存器中。

分析： 添加新指令 `comb` 时，首先需要根据 RISC-V 指令格式设置该指令各个字段的值，并在相关文件中加入该指令的比特模式。然后，配置该指令的译码结果，并在 ALU 中实现指令的具体功能。最后，确保该指令能够在处理器上正确执行，并验证功能的正确性。

在 `Instrutcions.scala` 文件中添加 `comb` 指令的比特模式。

为了避免新指令与 RISC-V-mini 中已有指令的冲突，我将 `comb` 指令的 `opcode`、`funct3` 和 `funct7` 部分设置为 0110011、111 和 0000001。接着，使用 `BitPat()` 函数设置 `comb` 指令的比特模式。具体代码如下：

```

50 // Jump & Link
51 def JAL = BitPat("b????????????????????1101111")
52 def JALR = BitPat("b????????????????????000????1100111")
53 // Synch
54 def FENCE = BitPat("b0000????????00000000000000001111")
55 def FENCEI = BitPat("b00000000000000000000100000001111")
56 // CSR Access
57 def CSRRW = BitPat("b????????????????001????1110011")
58 def CSRRS = BitPat("b????????????????010????1110011")
59 def CSRRC = BitPat("b????????????????011????1110011")
60 def CSRRWI = BitPat("b????????????????101????1110011")
61 def CSRRSI = BitPat("b????????????????110????1110011")
62 def CSRRCI = BitPat("b????????????????111????1110011")
63 // Change Level
64 def ECALL = BitPat("b0000000000000000000000001110011")
65 def EBREAK = BitPat("b0000000000010000000000001110011")
66 def ERET = BitPat("b0001000000000000000000001110011")
67 def WFI = BitPat("b0001000000100000000000001110011")
68
69 def NOP = BitPat.bitPatToUInt(BitPat("b00000000000000000000000010011"))
70 def COMB = BitPat("b0000001????????111????0110011")
71 ]

```

图 1 修改数据通路

2. 添加 comb 指令的译码

comb 指令需要在 ALU 中将 rs1 的高 16 位和 rs2 的低 16 位拼接成一个 32 位整数，因此需要在 Alu.scala 文件中添加相应的常量，并生成正确的信号以实现该功能。具体代码如下：

```

8 object Alu {
9   val ALU_ADD = 0.U(4.W)
10  val ALU_SUB = 1.U(4.W)
11  val ALU_AND = 2.U(4.W)
12  val ALU_OR = 3.U(4.W)
13  val ALU_XOR = 4.U(4.W)
14  val ALU_SLT = 5.U(4.W)
15  val ALU_SLL = 6.U(4.W)
16  val ALU_SLTU = 7.U(4.W)
17  val ALU_SRL = 8.U(4.W)
18  val ALU_SRA = 9.U(4.W)
19  val ALU_COPY_A = 10.U(4.W)
20  val ALU_COPY_B = 11.U(4.W)
21  val ALU_COMB = 12.U(4.W)
22  val ALU_XXX = 15.U(4.W)
23 }

```

图 2 添加 comb 指令的译码

接下来，为 comb 指令添加对应的译码映射。comb 指令执行后，程序计数器（PC）需要加 4，并且需要将寄存器文件中读取的 rs1 和 rs2 数据进行拼接操作。然后，将 ALU 输出的拼接结果写回到寄存器文件中。

在 Control.scala 文件中添加的具体代码如下：

```

2  EBREAK-> List(PC_4, A_XXX, B_XXX, IMM_X, ALU_XXX, BR_XXX, N, ST_XXX, LD_XXX, WB_CSR, N, CSR.P, N),
3  ERET -> List(PC_EPC, A_XXX, B_XXX, IMM_X, ALU_XXX, BR_XXX, Y, ST_XXX, LD_XXX, WB_CSR, N, CSR.P, N),
4  WFI -> List(PC_4, A_XXX, B_XXX, IMM_X, ALU_XXX, BR_XXX, N, ST_XXX, LD_XXX, WB_ALU, N, CSR.N, N),
5  COMB -> List(PC_4, A_RS1, B_RS2, IMM_X, ALU_COMB, BR_XXX, N, ST_XXX, LD_XXX, WB_ALU, N, CSR.N, N))
6  // format on
7 }

```

图 3 添加的具体代码

3. 实现 comb 指令的执行操作

在 Alu.scala 文件中添加将 rs1 高 16 位和 rs2 低 16 位拼接成一个 32 位整数的操作。
具体代码如下：

```
40 class AluSimple(val width: Int) extends Alu {
41   val io = IO(new AluIO(width))
42
43   val shamt = io.B(4, 0).asUInt
44
45   io.out := MuxLookup(io.alu_op, io.B)(
46     Seq(
47       ALU_ADD -> (io.A + io.B),
48       ALU_SUB -> (io.A - io.B),
49       ALU_SRA -> (io.A.asSInt >> shamt).asUInt,
50       ALU_SRL -> (io.A >> shamt),
51       ALU_SLL -> (io.A << shamt),
52       ALU_SLT -> (io.A.asSInt < io.B.asSInt),
53       ALU_SLTU -> (io.A < io.B),
54       ALU_AND -> (io.A & io.B),
55       ALU_OR -> (io.A | io.B),
56       ALU_XOR -> (io.A ^ io.B),
57       ALU_COPY_A -> io.A,
58       ALU_COMB -> Cat(io.A(31,16), io.B(15,0))
59     )
60   )
61
62   io.sum := io.A + Mux(io.alu_op(0), -io.B, io.B)
63 }
64
65 class AluArea(val width: Int) extends Alu {
66   val io = IO(new AluIO(width))
67   val sum = io.A + Mux(io.alu_op(0), -io.B, io.B)
68   val cmp =
69     Mux(io.A(width - 1) === io.B(width - 1), sum(width - 1), Mux(io.alu_op(1), io.B(width - 1), io.A(width -
70   val shamt = io.B(4, 0).asUInt
71   val shin = Mux(io.alu_op(3), io.A, Reverse(io.A))
72   val shiftr = (Cat(io.alu_op(0) && shin(width - 1), shin).asSInt >> shamt)(width - 1, 0)
73   val shiftl = Reverse(shiftr)
74
75   val comb = Cat(io.A(31,16), io.B(15,0))
76
77   val out =
78     Mux(
```

图 4 执行操作

4. 对 comb 指令进行测试

首先创建 comb.s 文件，编写如下的汇编程序：

```
.text
.global _start
_start:
    lui x6, 1
    lui x7, 2
    # comb x5, x6, x7
exit:
    csrw mtohost, 1
    j exit
.end
```

图 5 汇编程序

5. 在处理器上执行该指令，观察仿真波形，验证功能是否正确。

编写完程序后，使用如下命令进行编译：

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

然后使用 elf2hx 命令将 comb 二进制文件转换成十六进制：


```
elf2hex 16 4096 comb > 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：

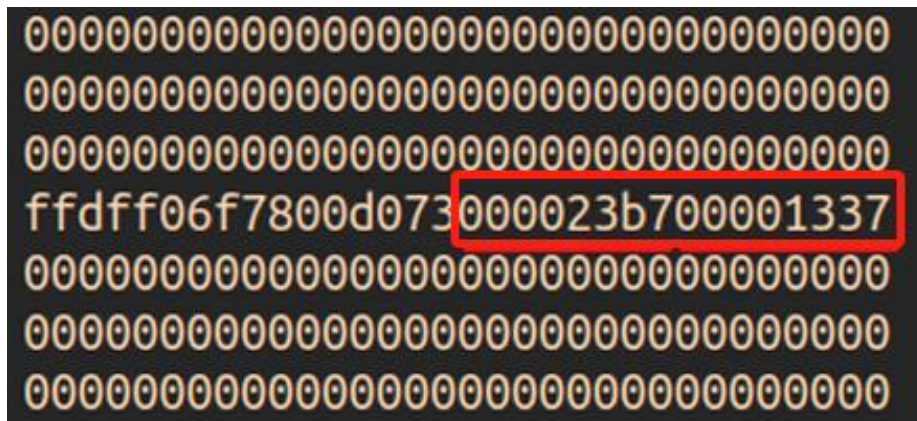


图 6 十六进制形式

comb x5, x6, x7 转换成机器码的十六进制形式为 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。修改后如下：

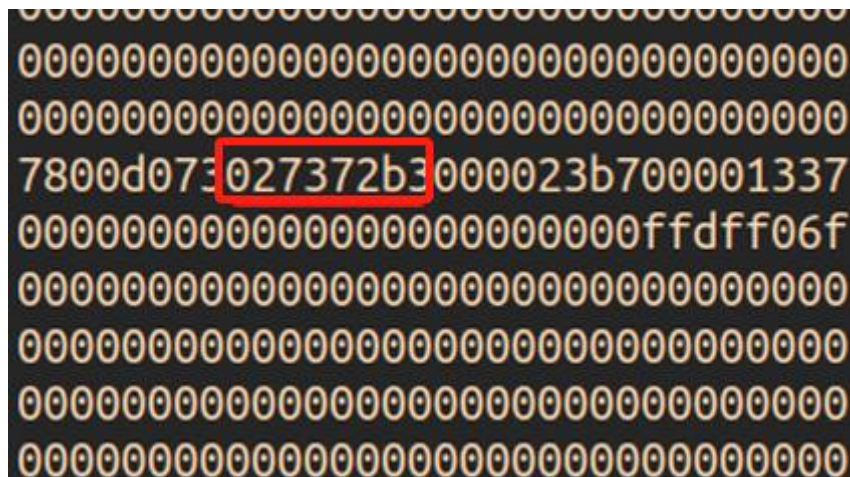


图 7 修改后

接着需要在主目录下执行 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 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。

```
./VTile comb.hex comb.vcd
```

使用 GTKWave 打开 comb.vcd 文件，其波形图如下，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

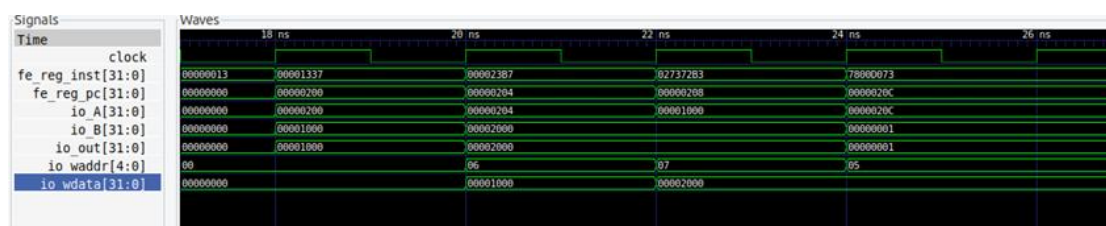


图 8 波形图

这里以新增了一个"MULT"指令，用于执行寄存器之间的乘法操作：
在 Instructions.scala 中定义新指令的比特模式：

[illegible]

在 `Control.scala` 中添加新指令的译码映射：

```

123 ERET -> List(PC_EPC, A_XXX, B_XXX, IMM_X, ALU_XXX, BR_XXX, Y, ST_XXX, LD_XXX, WB_CSR, N, CSR.P,
124 WFI -> List(PC_4, A_XXX, B_XXX, IMM_X, ALU_XXX, BR_XXX, N, ST_XXX, LD_XXX, WB_ALU, N, CSR.N,
125 COMB -> List(PC_4, A_RS1,B_RS2,IMM_X,ALU_COMB,BR_XXX,N,ST_XXX,LD_XXX,WB_ALU,N,CSR.N,N))
126 MULT -> List(PC_4, A_RS1,B_RS2,IMM_X,ALU_MULT,BR_XXX,N,ST_XXX,LD_XXX,WB_ALU,N,CSR.N,N))
127 // format: on
128 }

```

在 Alu.scala 中添加新指令的执行逻辑:

```

7
8 object Alu {
9   val ALU_ADD = 0.U(4.W)
10  val ALU_SUB = 1.U(4.W)
11  val ALU_AND = 2.U(4.W)
12  val ALU_OR = 3.U(4.W)
13  val ALU_XOR = 4.U(4.W)
14  val ALU_SLT = 5.U(4.W)
15  val ALU_SLL = 6.U(4.W)
16  val ALU_SLTU = 7.U(4.W)
17  val ALU_SRL = 8.U(4.W)
18  val ALU_SRA = 9.U(4.W)
19  val ALU_COPY_A = 10.U(4.W)
20  val ALU_COPY_B = 11.U(4.W)
21  val ALU_COMB=12.U(4.W)
22  val ALU_MULT=13.U(4.W)
23  val ALU_XXX = 15.U(4.W)
24 }
25

```

在 Control.scala 文件中添加 mult.s 的具体代码如下

```

1          .text
2
3          .global _start
4      _start:
5          lui x6, 1
6          lui x7, 2
7          # mult x5, x6, x7
8      exit:
9          csrw mtohost, 1
10         j exit
11         .end

```

深圳大学学生实验报告用纸

编写完程序后，使用如下命令进行编译

```
riscv32-unknown-elf-gcc -nostdlib -Text=0x200 -o mult mult.s
```

然后使用 elf2hx 命令将 mult 二进制文件转换成十六进制：

```
elf2hex 16 4096 mult > mult.hex
```

在 mult.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：

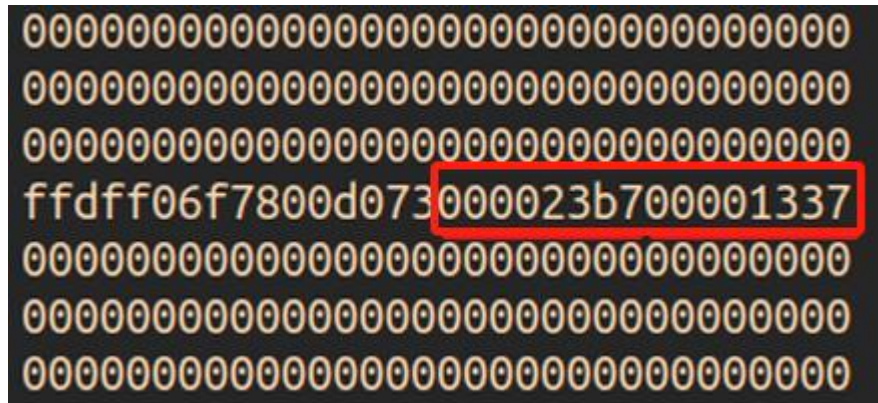


图 13mult.hex

mult x5, x6, x7 转换成机器码的十六进制形式为 0x00020367。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。修改后如下：



图 14 mult x5, x6, x7

产生 VTile 可执行文件后执行下面命令，使 mini 处理器执行新建指令并产生波形文件。

```
./VTile mult.hex mult.vcd
```

使用 GTKWave 打开 comb.vcd 文件，其波形图如下：

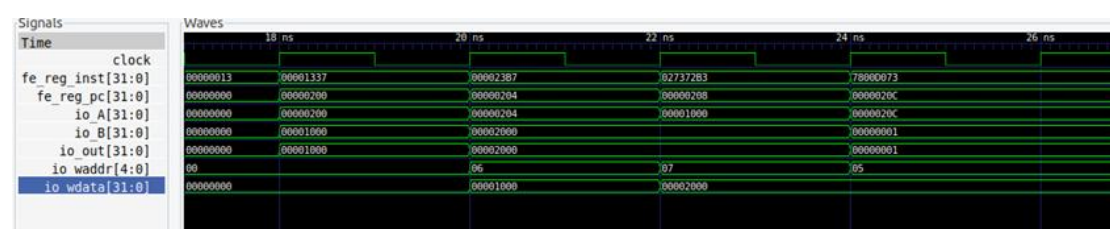


图 15 波形图

六、实验总结与体会

通过本次新增指令实验，我对 RISC-V mini 处理器架构有了更深入的理解，同时在指令设计与实现方面积累了宝贵的经验。在实验过程中，我深刻体会到了硬件描述语言 Chisel 的强大功能。它能够以简洁且直观的方式描述处理器的数据通路和控制逻辑，使得复杂的硬件设计变得相对容易理解和实现。通过修改数据通路来新增指令，我明白了处理器各个组件之间的协同工作原理，以及指令在不同阶段的处理流程。新增“comb”指令时，从设置指令字段值、添加比特模式，到在译码器和 ALU 中实现相应功能，每一步都需要仔细考虑，确保与现有指令集的兼容性以及新指令功能的正确性。在这个过程中，遇到了一些问题，比如指令格式的设置错误导致编译不通过，以及信号连接错误导致功能无法实现。但通过仔细查阅文档、分析代码和调试，最终成功解决了这些问题。这让我认识到在硬件设计中，细节至关重要，一个小的错误可能会导致整个系统的故障。自行设计“MULT”指令进一步加深了我对指令设计的理解。在这个过程中，我学会了如何根据需求定义指令的功能、比特模式和译码映射，并且能够在 ALU 中实现具体的执行逻辑。通过编写测试程序并观察仿真波形，验证了指令的正确性，这让我感受到了设计成功后的成就感。实验也让我意识到处理器设计的复杂性和严谨性。每一个指令的添加都需要考虑到对整个系统的影响，包括与其他指令的交互、数据通路的变化以及控制逻辑的调整。同时，测试和验证环节也不可或缺，只有通过充分的测试，才能确保指令在各种情况下都能正确执行。总的来说，本次实验不仅提高了我的硬件设计能力，还培养了我的问题解决能力和耐心。在今后的学习和研究中，我将继续深入学习计算机系统相关知识，探索更多关于处理器设计和指令集扩展的内容。

<p>指导教师批阅意见:</p> 	
<p>成绩评定:</p> 	

指导教师签字:

 年 月 日

指导教师批阅意见:

成绩评定:

指导教师签字:
年 月 日

<p>指导教师批阅意见:</p> <p>成绩评定:</p>	<p>指导教师签字:</p> <p>年 月 日</p>
--	---------------------------------

指导教师批阅意见：

成绩评定：

指导教师签字：
年 月 日

备注:	
-----	--

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。