实验三：LC3 系统-寄存器堆模块设计

1. 实验目的

1.1. 掌握 LC3 系统中寄存器堆模块的设计

1.2. 学习设计 Regfile 的前置 Chisel 语法。

1.3. 掌握使用单元测试验证 LC3 系统中的寄存器堆模块。

2. 实验内容

2.1. 阅读本实验前置 Chisel 语法。

2.2. 根据实验指导设计 ALU 模块

2.3. 根据实验指导对 ALU 模块进行单元测试。

3. 实验步骤

3.1. Chisel 语法学习

（1）Reg

时序电路中的关键组件是寄存器，寄存器的值在时钟上升沿到来时变化。Chisel 中定义寄存器的语法为 Reg,另外 RegInit 除了定义一个寄存器，还可以赋一个初始值。

示例代码（src/test/scala/example/Reg.scala）如图所示：

```scala
// 示例Reg模块
class Reg extends Module {
  val io = IO(new Bundle {
    val in  = Input(UInt(2.W))
    val out = Output(UInt(2.W))
  })

  val aReg = RegInit(1.U(2.W))

  aReg := io.in
  io.out := aReg

}
```

```scala
// 示例测试
class RegTest extends AnyFlatSpec
  with ChiselScalatestTester {

  it should "test reg" in {
    test(new Reg) { c =>
      c.io.in.poke(2.U)
      println(s"时钟到来前输出：${c.io.out.peek}")
      c.clock.step()
      println(s"时钟到来后输出：${c.io.out.peek}")
    }
  }
}
```

键入 `mill chisel_lc3.test.testOnly example.RegTest` 其测试结果为：

```
RegTest:
时钟到来前输出： UInt<2>(1)
时钟到来后输出： UInt<2>(2)
- should test reg
```

从结果可见，只有当时钟上升沿到来时，寄存器的值才会发生变化。

寄存器值改变除了依赖时钟，还可以认为的制造寄存器值更新的条件，代码如图所示：

```scala
class CondReg extends Module {
  val io = IO(new Bundle {
    val in   = Input(UInt(2.W))
    val cond = Input(Bool())
    val out  = Output(UInt(2.W))
  })

  val aReg = RegInit(1.U(2.W))

  when(io.cond) {
    aReg := io.in
  }
  io.out := aReg

}
```

```scala
// 示例测试
class CondRegTest extends AnyFlatSpec
  with ChiselScalatestTester {

  it should "test condreg" in {
    test(new CondReg) { c =>
      c.io.in.poke(2.U)
      c.io.cond.poke(false.B)
      c.clock.step()
      println(s"cond==false输出: ${c.io.out.peek}")

      c.io.in.poke(2.U)
      c.io.cond.poke(true.B)
      c.clock.step()
      println(s"cond==true输出: ${c.io.out.peek}")
    }
  }
}
```

该模块比上一个模块多了一个 cond 条件，只有当 cond 为 true 时寄存器的值才会改变。键入 `mill chisel_lc3.test.testOnly example.RegTest` 其测试结果为：

```
CondRegTest:
cond==false输出: UInt<2>(1)
cond==true输出: UInt<2>(2)
- should test condreg
```

在实际电路设计过程中寄存器的定义有时候需要多个一起定义，如寄存器堆和各种存储部件，因此在 Chisel 中可以通过 Array 来建立一个数组，再用 VecInit 包装这个数组，最后用 RegInit 来定义寄存器数组。定义十个寄存器，每个寄存器初值为 0.U，宽度为 1024 比特，语法如下：

```
val tenReg = RegInit(VecInit(Array.fill(10)(0.U(1024.W))))
```

3.2. 寄存器堆的设计

对一个寄存器堆需要做的操作主要有读和写。在 LC3 的指令中最多需要读取两个数据，写入一个数据。 因此需要两组读端口一组写端口，每组端口需要一个地址和数据。此外还需要一个信号来判断是否为写信号，只有该信号为真才可以改变寄存器里的数据。因此，寄存器堆的模块端口如下表：

端口名称	方向	位宽
读地址 1	输入	3 bits
读数据 1	输出	16 bits
读地址 2	输入	3 bits
读数据 2	输出	16 bits
写地址	输入	3 bits
写数据	输入	16 bits
写使能	输入	1 bits

由于 LC3 寄存器堆中有 8 个寄存器，因此读写地址可以用 3bits 索引。其次，寄存器宽度为 16 位，所以读写数据位宽为 16。

对寄存器堆进行读操作时，模块输入一个读地址，输出一个读数据。对寄存器堆进行写操作时，模块输入一个写地址和一个写数据且写使能为 1，此时寄存器的值被改变。

根据以上模块端口和功能描述，寄存器堆模块的 Chisel 代码如下，代码中已经给出端口描述，需要大家补充内部逻辑。

（src/main/scala/LC3/Regfile.scala）

```
class Regfile extends Module{
  val io = IO(new Bundle {
    val wen = Input(Bool())
    val wAddr = Input(UInt(3.W))
    val r1Addr = Input(UInt(3.W))
    val r2Addr = Input(UInt(3.W))
    val wData = Input(UInt(16.W))
    val r1Data = Output(UInt(16.W))
    val r2Data = Output(UInt(16.W))
  })

  io.r1Data := DontCare
  io.r2Data := DontCare

  // 实验四 任务一
  // 在此编写寄存器堆逻辑（定义寄存器堆、写逻辑、读逻辑）

}
```

实验四-任务一：在 RegFile.scala 编写寄存器堆逻辑，包括定义寄存器堆、写逻辑、读逻辑

3.3. 寄存器堆的单元测试

自此我们已经验证过控制器和 ALU 模块，请自己尝试搭建验证框架对 Regfile 模块进行验证。

验证思路：对寄存器堆中每个寄存器进行读写测试，即对一个寄存器写一个数值，然后读出来判断是否是写入的值

实验四-任务二：在 RegFileTest.scala 文件中编写寄存器堆测试用例

提示：输入一个读操作需要输入 wen, raddr, rdata，输入一个写操作需要输入 wen, waddr, wdata

完成设计和验证后输入如下命令对 Regfile 进行验证。

```
mill chisel_lc3.test.testOnly LC3.RegfileTest
```