# 实验十二 实现访存指令的理想流水线设计实验

## 实验目的

1. 掌握访存指令的数据通路。
2. 掌握在访存级中添加访存部件 LSU 的方法。
3. 掌握在五级流水线中实现访存指令的方法。

## 实验原理与实验内容

### 访存指令列表

RV64I中访存指令共计11条，包括7条加载（Load）指令和4条存储（Store）指令，指令格式如表1-1所示。

* + - * 1. RV64I访存指令

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 |  | 25 | 24 | 20 | 19 | 15 | 14 | 12 | 11 | 7 | 6 | 0 |  |
| imm[11:0] | | | | | rs1 | | 000 | | rd | | 0000011 | | LB |
| imm[11:0] | | | | | rs1 | | 001 | | rd | | 0000011 | | LH |
| imm[11:0] | | | | | rs1 | | 010 | | rd | | 0000011 | | LW |
| imm[11:0] | | | | | rs1 | | 011 | | rd | | 0000011 | | LD |
| imm[11:0] | | | | | rs1 | | 100 | | rd | | 0000011 | | LBU |
| imm[11:0] | | | | | rs1 | | 101 | | rd | | 0000011 | | LHU |
| imm[11:0] | | | | | rs1 | | 110 | | rd | | 0000011 | | LWU |
| imm[11:5] | | | rs2 | | rs1 | | 000 | | imm[4:0] | | 0100011 | | SB |
| imm[11:5] | | | rs2 | | rs1 | | 001 | | imm[4:0] | | 0100011 | | SH |
| imm[11:5] | | | rs2 | | rs1 | | 010 | | imm[4:0] | | 0100011 | | SW |
| imm[11:5] | | | rs2 | | rs1 | | 011 | | imm[4:0] | | 0100011 | | SD |

其中B表示Byte，1字节，即8bit；H表示Half word，半字，即16bit；W表示Word，1字，即32bit；D表示Double word，双字，即64bit。

观察指令格式可以知道，加载指令为I型指令，存储指令为S型指令。

### 数据通路设计

在之前的实验中，访存单元内部空空如也是毫无功能的，在本实验中终于可使访存单元发挥其应有的作用。下面以LH指令和SW指令为例，分析一下如何设计实现访存指令的数据通路。

前端

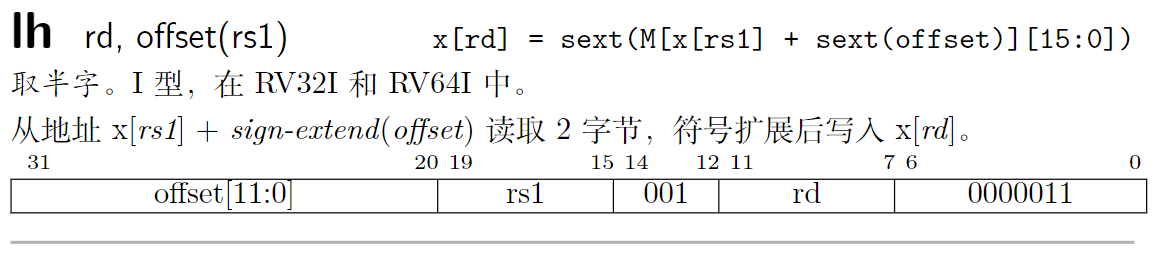
前端只负责指令的准备，指令的执行细节都在后端部分，因此无需修改前端的数据通路。

后端

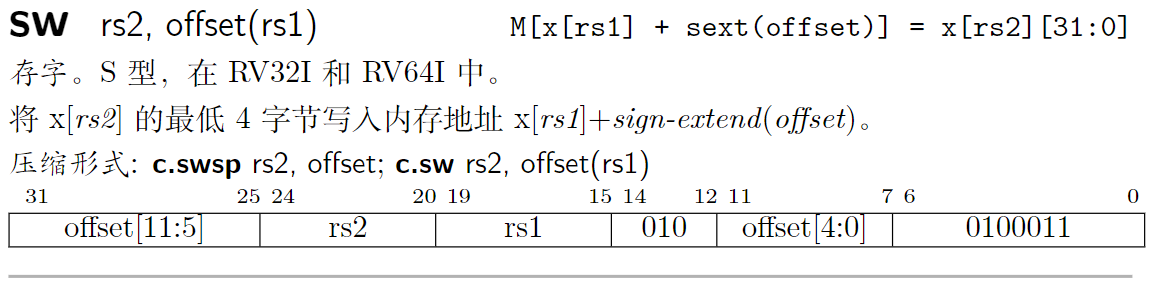
前端部分已经成功取得指令，按照指令执行的一般流程可以分析得到：在译码单元产生合适的控制信号，在访存单元进行访存操作，在写回单元将数据写回寄存器堆。

##### 译码单元

译码单元要完成指令译码产生控制信号以及准备源操作数这两个操作。



* + - 1. LH指令定义



* + - 1. SW指令定义

由图1-1展示的LH指令定义可以得知，LH为I型指令，在之前的实验中，已经实现了I型运算指令的数据通路，因此准备LH指令源操作数的数据通路已经实现完毕，无需进行修改。对于Store指令而言，虽然其为S型指令，但其源操作数均来自通用寄存器堆。至于S型指令的立即数字段，只需修改译码器内imm信号的生成逻辑生成正确的立即数即可，该信号被包含于info数据包内向后传递并在后续流水级可以随时使用，因此Store指令也无需修改数据通路。

* + - * 1. 译码信号

| 信号名 | 含义 |
| --- | --- |
| inst | 32位指令 |
| src1\_ren  src1\_raddr  src2\_ren  src2\_raddr  op  reg\_wen  reg\_waddr | src1是否需要读通用寄存器堆  src1的通用寄存器堆读地址  src2是否需要读通用寄存器堆  src2的通用寄存器堆读地址  指令的操作类型  是否需要写回通用寄存器堆  通用寄存器堆的写地址 |
| imm | 立即数 |
| fusel | FU选择信号 |

表1-2展示了译码器生成的信号。imm信号均为立即数字段符号拓展至64位的结果。fusel信号需要修改：只需修改FuType中的值即可：将num修改为3，定义lsu为2；这时只要fusel信号是由apply方法进行实例化，则其宽度会自动拓展为2。对于访存类指令，其fusel信号的值应当为FuType.lsu。下面的代码展示了FuType的定义。

object FuType {

  def num     = 3

  def alu     = 0.U // arithmetic logic unit

  def mdu     = 1.U // multiplication division unit

  def lsu     = 2.U // load store unit

  def apply() = UInt(log2Up(num).W)

}

op信号也需要进行修改，观察表1-1，可以通过opcode分辨加载和存储指令，通过func3字段进行不同操作的细分。可以使用func3字段作为op的低3位，在高位使用0表示加载，1表示存储，因此LH指令的op应该为0001，SW指令的op为1010。将访存相关指令的op定义于LSUOpType中，便于统一管理，代码如下：

object LSUOpType {

  def lh  = "b0001".U

  def sw  = "b1010".U

  def isStore(func: UInt): Bool = func(3)

  def isLoad(func:  UInt): Bool = !isStore(func)

}



* + - 1. 译码单元结构

综上所述，译码单元只需改动译码器内部逻辑代码即可，整体译码单元的结构无调整，结构图如图1-3所示，与上一实验译码单元结构一致。这里再次强调，所有的译码信号都在info数据包中向后续流水级传递，chisel在生成verilog时会自动删去在后续流水级中未使用到的信号，因此无需担心信号冗余。

##### 执行级缓存

无需修改。

##### 执行单元

指令RAM是使用SRAM实现的，对于读数据而言其时序是第一个周期发送读信号和读地址，第二个周期获得数据RAM的返回结果；而存数据则不必考虑时序，只要发送写信号、写地址和写数据即可，因为不需要得到数据RAM的回应。为了在访存级可以读到正确的数据，应当在执行单元向数据RAM发起读请求，这样在下一个周期，指令运行到访存单元时，正好可以读到正确的数据。

## 实验要求

1. 根据本实验提供的五级流水线编程框架，在流水线 CPU 中添加以下指令：ADD、SUB、SLL、SLT、SLTU、XOR、SRL、SRA、OR、AND、ADDW、SUBW、SLLW、SRLW、SRLW、SRAW。
2. 通过本实验提供的所有仿真验证测试用例。
3. 通过本实验提供的所有板级验证测试用例。
4. 撰写实验报告。撰写报告时要求叙述以下内容，以及你对本实验的思考与探索。
   1. 选择需要实现的指令中的一条，按照你自己的理解，逐步介绍其数据通路设计的思路以及实现过程
   2. 尝试自己绘制一幅属于自己的数据通路图。（注意：以后数据通路的添加都需要在该图上继续增加，因此打一个好的地基很重要，现在偷懒之后还是需要补的！）
   3. TODO：增加更多内容

## 实验步骤

1. 实验步骤参见实验九的实验步骤部分
2. 提醒与建议
   1. 实验的关键在于正确的在译码单元对源操作数进行选择。需要注意的是AUIPC指令的两个操作数分别是当前指令的PC和立即数；LUI指令只有一个源操作数，但也相当于是0与立即数进行运算。
   2. 在chisel中有多种数据选择器类型，如Mux、MuxCase、MuxLookup、Mux1H等，请参考chisel教程章节选择适合的数据选择器。
   3. 当reg\_ren为0时，建议将reg\_raddr置为0，防止CPU出现意想不到的状况。

## 思考与探索

1. RISC-V 指令集是定长指令集吗？
2. RV64 和 RV32 的 R 型运算指令是否有区别？
3. SLT 和 SLTU 这类比较指令的实现是为了什么目的，比如是为了实现什么样的目标才有了这类指令？
4. SLL、SRL 和 SRA 这三条指令在 src2 高 63 至 6 位不全为 0 的时候，指令的执行结果是什么？
5. RISC-V 的运算指令有进行运算结果的溢出判断吗，为什么要这样设计？可以对比 MIPS 指令集进行说明。
6. 为什么并不是所有的R型计算指令都有对应的字指令？
7. 请问差分测试框架只有这些debug信号就够了吗？假如有的指令不写回通用寄存器堆呢，这时框架又该如何发现问题？
8. 当前处理器采用的是哈佛结构还是冯诺依曼结构？
9. 谈谈你在实验中碰到了哪些问题？又是如何解决的？