## 实验十 实现转移指令的理想流水线设计实验

### 实验目的

1. 掌握转移指令的数据通路。
2. 掌握在执行级中添加转移控制部件 BRU 的方法。
3. 掌握在五级流水线中实现转移指令的方法。

### 实验原理与实验内容

RV64I中转移指令共计8条，包括6条分支跳转指令和2条跳转链接指令，详细内容请参考第1章的1.4小节的转移指令部分。

程序的基本结构包括：顺序结构、分支结构、循环结构。在实现转移指令前，MyCPU只能运行仅有顺序结构的程序，当完成本实验后，处理器才可以执行一般性的程序。下面以beq指令和jalr指令为例，分析一下如何设计实现转移指令的数据通路。

这次数据通路的分析顺序有所不同，前端和后端数据通路的设计会相互穿插说明。

#### 转移类指令的数据通路的设计

结合指令执行的一般流程以及转移指令功能的说明可以分析得到：指令在译码单元产生合适的控制信号，在执行单元进行转移指令的执行将转移目标地址发送至取指单元用于更新PC，对于链接指令还需在写回单元将数据写回寄存器堆。

##### 指令队列和译码单元

指令队列无需修改。

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。

beq指令格式和功能

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **位数**  **指令** | **31 25** | **24 20** | **19 15** | **14 12** | **11 7** | **6 0** | **B型格式分支跳转指令的功能** |
| **offset[12|10:5]** | **rs2** | **rs1** | **funct3** | **offset[4:1|11]** | **opcode** |
| beq rs1,rs2,offset12 | offset[12|10:5] | rs2 | rs1 | 000 | offset[4:1|11] | 1100011 | 相等时跳转：if(rs1=rs2) then  PC+SEXT64({offset<<1})→PC |

jalr指令格式和功能

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 位数  指令 | 31 20 | 19 15 | 14 12 | 11 7 | 6 0 | I 型格式的跳转和链接指令功能 |
| **offset12** | **rs1** | **funct3** | **rd** | **opcode** |
| jalr rd,offset12(rs1) | offset[11:0] | rs1 | 000 | rd | 1100111 | 寄存器相对跳转并链接，12位偏移量经符号扩展后与rs1相加，将得到的地址最低位置0后置入PC：PC+4→rd,  (rs1+SEXT64(offset12))&~1→PC |

由表7-1展示的beq指令定义可以得知，beq为B型指令。对于B型指令而言：两个源操作数来自寄存器堆，需要生成正确的src1\_ren和src2\_ren信号；同时还需要使用立即数字段，因此还需正确产生imm。对于jalr指令而言，其为I型指令，数据通路已经设计完毕无需修改，jalr指令的格式和功能如表7-2所示。

fusel信号需要修改，只需修改FuType中的值即可：将num修改为4，定义bru为3；对于转移类指令，其译码得到的fusel信号的值应当为FuType.bru。下面的代码展示了FuType的定义：

object FuType {

  def num     = 4

  def alu     = 0.U // arithmetic logic unit

  def mdu     = 1.U // multiplication division unit

  def lsu     = 2.U // load store unit

  def bru     = 3.U // branch unit

  def apply() = UInt(log2Up(num).W)

}

op信号也需要进行修改，观察1.4小节的转移指令格式，可以通过opcode分辨B型、J型和I型这三种转移指令，通过func3字段对于B型转移指令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表示条件转移指令，1表示直接跳转指令。因此beq指令的op应该为0000，jalr指令的op为1000。jal指令不存在func3字段，那么只要将其op设置为1001~1111中的一种，不与jalr相同即可。

将转移相关指令的op定义于BRUOpType中，便于统一管理，代码如下：

object BRUOpType {

def jal  = "b1000".U

  def jalr = "b1010".U

  def beq  = "b0000".U

  def isBranch(func: UInt) = !func(3)

  def isJump(func:   UInt) = !isBranch(func)

}

对于链接类指令而言，在写回级需要将返回地址写入寄存器堆，因此其reg\_wen应当为1；同理得到非链接类指令reg\_wen为0。

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，与上一实验译码单元结构一致。

##### 执行级缓存和执行单元

执行级缓存无需修改。

通过在执行单元增加BRU来实现转移指令的执行。BRU需要产生3个信号：

1. branch：是否发生跳转。
2. target：跳转地址。
3. rd\_info.wdata(FuType.bru)：链接地址。

对于直接跳转指令jal和jalr而言：

1. branch信号在下面3种情况均满足时为真：
2. info.valid为真，即当前指令有效。
3. info.fusel的值为FuType.bru，即当前指令为转移类指令。
4. isJump(op)为真，即当前指令为直接跳转指令。
5. target的设定逻辑如下：
   1. 对于jal指令而言，target的值等于src\_info.src1\_data加上src\_info.src1\_data。
   2. 对于 jalr指令而言，target的值等于src\_info.src1\_data加上src\_info.src1\_data的和与上~1.U(XLEN.W)[[1]](#footnote-1)，也就是将最低位置为0。
6. rd\_info.wdata(FuType.bru)设置为PC+4。

对于条件分支指令而言:

1. branch信号在下面4种情况均满足时为真：
2. info.valid为真，即当前指令有效。
3. info.fusel的值为FuType.bru，即当前指令为转移类指令。
4. isBranch(op)为真，即当前指令为条件分支指令。
5. src\_info.src1\_data和src\_info.src1\_data的值满足指令要求，如对于beq指令而言，两者的值需要相同。
6. target被赋值为PC加上info.imm。
7. rd\_info.wdata(FuType.bru)的设置可以忽略，因为条件分支指令在写回单元无需写回寄存器堆。

完成BRU的逻辑后，执行单元需要将branch和target发送至取指单元。执行单元的结构如图7-1所示。



执行单元结构

##### 取指单元



取指单元结构

取指单元结构修改为如图7-2所示，pc\_next的来源增加为PC+4和target，使用branch进行选择。branch为0时，pc\_next等于PC+4，与顺序执行的情况一致；当branch为1时，pc\_next等于target，当前发送至InstMEM的地址为跳转地址target，下一拍可以取到跳转地址对应的指令，PC被更新成跳转地址。

##### 访存级缓存和访存单元

转移指令与访存无关，因此访存级缓存和访存单元无需修改。

##### 写回级缓存和写回单元

写回级缓存和写回单元无需修改。

### 实验要求

1. 在上一个实验的基础上继续添加第1章的1.4小节的所有转移指令共计8条指令。
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. 线上提交实验作品，获得系统评分。
5. 撰写实验报告。报告中应包含以下内容和至少3道本实验思考题。
6. 仿照6.1实验七中的取指令访问时序波形图，绘制发生跳转前后的取指令访问时序波形图。
7. 选择第1章的1.4小节转移指令中的一条指令（非beq、jalr），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。
8. 修改MyCPU内部数据通路图，增加转移功能。

### 实验步骤

1. 实验步骤参见6.4节实验十的实验步骤部分。
2. 提醒与建议：
   1. 对于J型指令而言，其源操作数来自PC和imm，只需正确设置src1\_ren和src2\_ren信号即可。

### 思考与探索

1. 为什么jalr的target需要与上~1.U(XLEN.W)？其它转移指令为什么不需要这样的操作？
2. 能否在译码单元实现转移指令？如果可以，应该怎样实现，需要考虑哪些因素？如果不可以，又是什么原因导致的？
3. 观察绘制出的取指令时序波形图，你是否发现了什么问题？
4. 谈谈你在实验中碰到了哪些问题？又是如何解决的？

1. ~1.U(XLEN.W)的值等于FFFF\_FFFF\_FFFF\_FFFE。 [↑](#footnote-ref-1)