## 实验十三 实现CSR指令的流水线设计实验

### 实验目的

1. 掌握CSR指令的数据通路。
2. 掌握在执行级中添加控制状态寄存器部件CSR的方法。
3. 掌握在五级流水线中实现CSR指令的方法。

### 实验原理与实验内容

CSR指令被定义在Zicsr指令集中，共计6条指令，详细内容请参考第6章的6.4小节的Zicsr指令集部分。CSR寄存器是在处理器核内部用于配置或记录程序性能和状态信息的寄存器。CSR指令用于设置相应的控制信息或读取状态信息。因此本实验不仅需要实现CSR指令还需要实现部分CSR寄存器。特权模式和CSR寄存器的相关介绍可以参考第6章的6.3小节。

#### 需要实现的CSR寄存器

本实验要求实现的U模式和M模式的CSR寄存器如表7-1所示。权限列中，URO表示最低在U模式下只读；MRO表示最低在M模式下只读；MRW表示最低在M模式下可读写。权限的大小为U<S<M，高权限可以访问低权限的CSR，反之则不行。

CSR寄存器列表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **非特权计数器/定时器** | | | | | |
| **地址** | **权限** | **名称** | **描述** | **是否要求实现** | **简单实现** |
| 0xC00 | URO | cycle | RDCYCLE指令的周期计数器 | 是 |  |
| **机器信息寄存器** | | | | | |
| **地址** | **权限** | **名称** | **描述** | **是否要求实现** | **简单实现** |
| 0xF11 | MRO | mvendorid | 供应商ID | 是 | 只读0 |
| 0xF12 | MRO | marchid | 架构ID | 是 | 只读0 |
| 0xF13 | MRO | mimpid | 实现ID | 是 | 只读0 |
| 0xF15 | MRO | mhartid | 硬件线程ID | 是 | 只读0 |
| **机器陷阱设置寄存器** | | | | | |
| **地址** | **权限** | **名称** | **描述** | **是否要求实现** | **简单实现** |
| 0x300 | MRW | mstatus | 机器状态寄存器 | 是 |  |
| 0x301 | MRW | misa | 指令集架构和拓展 | 是 | 只读定值 |
| 0x302 | MRW | medeleg | 机器异常委托寄存器 | 否 |  |
| 0x303 | MRW | mideleg | 机器中断委托寄存器 | 否 |  |
| 0x304 | MRW | mie | 机器中断使能寄存器 | 是 |  |
| 0x305 | MRW | mtvec | 机器陷阱处理程序基地址 | 是 |  |
| 0x306 | MRW | mcounteren | 机器计数器启用 | 是 | 只读0 |
| **机器陷阱处理寄存器** | | | | | |
| **地址** | **权限** | **名称** | **描述** | **是否要求实现** | **简单实现** |
| 0x340 | MRW | mscratch | 机器陷阱处理程序的草稿寄存器 | 是 |  |
| 0x341 | MRW | mepc | 机器异常程序计数器 | 是 |  |
| 0x342 | MRW | mcause | 机器陷阱原因 | 是 |  |
| 0x343 | MRW | mtval | 机器错误的地址或指令 | 是 |  |
| 0x344 | MRW | mip | 机器中断待处理 | 是 |  |

#### CSR指令的数据通路的后端设计

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。下面以csrrs指令和csrrwi指令为例，介绍如何设计实现CSR指令的数据通路。

##### 译码单元

CSR指令均为I型格式的指令，对于准备I型指令格式的源操作数的数据通路在之前的实验中就已经设计完毕，因此无需修改。对于csrrwi指令而言，其rs1字段应该为立即数，但是为了保证I型指令在译码级别解码的一致性，将CSR指令的特殊解码移到后续的执行单元CSR部件中去。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 位数  指令 | 31 20 | 19 15 | 14 12 | 11 7 | 6 0 | I 型格式的csr指令功能 |
| **csr** | **rs1** | **funct3** | **rd** | **opcode** |
| csrrs rd,csr,rs1 | csr | rs1 | 010 | rd | 1110011 | 读后置位csr: csr→rd, rs1 | csr→csr |
| csrrwi rd,csr,rs1 | csr | imm5 | 101 | rd | 1110011 | 立即数读后写csr，把5位的零扩展立即数zimm写入csr：  csr→rd, UEXT64 (imm5)→csr |

fusel信号需要修改，所有的CSR指令的fusel应该为FuType.csr，具体的代码定义如下：

object FuType {

  def num     = 5

  def alu     = 0.U // arithmetic logic unit

  def lsu     = 1.U // load store unit

  def mdu     = 2.U // multiplication division unit

  def bru     = 3.U // branch unit

  def csr     = 4.U // control status register

  def apply() = UInt(log2Up(num).W)

}

op信号也需要修改，通过func3字段分辨各种操作。csrrw、csrrs、csrrc、csrrwi、csrrsi和csrrci中的r表示读，各指令的区别在于后半段操作：w表示write，这里简写为wrt；s表示set；c表示clear，这里简写为clr；i表示immediate。将CSR指令的op定义于CSROpType中，便于统一管理，代码如下：

object CSROpType {

  def wrt  = "b001".U

  def set  = "b010".U

  def clr  = "b011".U

  def wrti = "b101".U

  def seti = "b110".U

  def clri = "b111".U

}

I型指令的reg\_wen均为1，因此reg\_wen信号也无需修改。

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。

在FU中增加新的功能单元CSR，如图7-1所示。CSR内定义了相关的CSR寄存器以及CSR指令操作的逻辑。CSR寄存器有点类似通用寄存器堆，也是一系列寄存器的集合，也拥有寄存器地址也能保存64位数据。区别在于CSR寄存器的地址是离散的，CSR寄存器的每个位的意义都是不同的，需要根据手册定义赋值，并且有些位可能是只读、只写、可读写或无关等。



执行单元结构

在CSR单元中需要实现表7-2中的控制信号。

CSR单元内部需要生成的控制信号

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **含义** | **信号名** | **含义** |
| addr | 要访问的CSR的地址 | csri | CSR立即数指令的立即数 |
| wdata | 写入CSR寄存器的值 | rdata | 从CSR寄存器中读到的值 |
| write | 是否写CSR寄存器 |  |  |

addr的值为info.inst[31:20]。csri的生成逻辑为UEXT64(info.inst[19:15])。对于csrrs而言wdata等于rdata|src\_info.src1\_rdata；csrrwi的wdata为csri；目前write值恒为1。

CSR单元内部的寄存器定义

| **地址** | **权限** | **名称** | **初始值** | **读掩码** | **写掩码** |
| --- | --- | --- | --- | --- | --- |
| 0xC00 | URO | cycle | 00000000\_00000000 | FFFFFFFF\_FFFFFFFF | 00000000\_00000000 |
| 0xF11 | MRO | mvendorid | 00000000\_00000000 | FFFFFFFF\_FFFFFFFF | 00000000\_00000000 |
| 0xF12 | MRO | marchid | 00000000\_00000000 | FFFFFFFF\_FFFFFFFF | 00000000\_00000000 |
| 0xF13 | MRO | mimpid | 00000000\_00000000 | FFFFFFFF\_FFFFFFFF | 00000000\_00000000 |
| 0xF15 | MRO | mhartid | 00000000\_00000000 | FFFFFFFF\_FFFFFFFF | 00000000\_00000000 |
| 0x300 | MRW | mstatus | 00000000\_00000000 | FFFFFFFF\_FFFFFFFF | FFFFFFFF\_FFFFFFFF |
| 0x301 | MRW | misa | 00000000\_00001100 | FFFFFFFF\_FFFFFFFF | 00000000\_00000000 |
| 0x304 | MRW | mie | 00000000\_00000000 | FFFFFFFF\_FFFFFFFF | FFFFFFFF\_FFFFFFFF |
| 0x305 | MRW | mtvec | 00000000\_00000000 | FFFFFFFF\_FFFFFFFF | FFFFFFFF\_FFFFFFFF |
| 0x306 | MRW | mcounteren | 00000000\_00000000 | FFFFFFFF\_FFFFFFFF | FFFFFFFF\_FFFFFFFF |
| 0x340 | MRW | mscratch | 00000000\_00000000 | FFFFFFFF\_FFFFFFFF | FFFFFFFF\_FFFFFFFF |
| 0x341 | MRW | mepc | 00000000\_00000000 | FFFFFFFF\_FFFFFFFF | FFFFFFFF\_FFFFFFFF |
| 0x342 | MRW | mcause | 00000000\_00000000 | FFFFFFFF\_FFFFFFFF | FFFFFFFF\_FFFFFFFF |
| 0x343 | MRW | mtval | 00000000\_00000000 | FFFFFFFF\_FFFFFFFF | FFFFFFFF\_FFFFFFFF |
| 0x344 | MRW | mip | 00000000\_00000000 | FFFFFFFF\_FFFFFFFF | 00000000\_00000888 |

对于CSR寄存器的读写有两种处理方法可以自行选择：

①利用switch语句，根据addr选择寄存器，寄存器中的数据与上读掩码赋值给rdata，如果write为1则将wdata与上写掩码写入寄存器。这种方式容易实现，但是会有许多相似的重复语句，整体代码行数会很多。

②设计这样两个函数：一个函数用于组织寄存器组，输入是CSR地址、CSR寄存器、读掩码、写掩码，输出是一个地址映射正确的CSR寄存器组（利用scala的Map函数）；另一个函数用于读写CSR寄存器组，输入是addr、write、wdata，输出是rdata。这种方式实现起来更优雅，但是难度较高。

方法①适合verilog以及chisel，方法②只适合chisel。

最后将rdata写入rd\_info.wdata(FuType.csr)中，用于在写回单元写入寄存器堆。

##### 其他后端缓存和单元

均无需修改。

### 实验要求

1. 在上一个实验的基础上继续添加第6章的6.4小节的所有CSR指令共计6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. 选择一个CSR寄存器，介绍各字段的含义。
7. 选择第6章的6.4小节转移指令中的一条指令（非csrrc、csrrwi），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，支持CSR指令。

### 实验步骤

1. 实验步骤参见7.8节实验十二的实验步骤部分。
2. 提醒与建议：
   1. 在实现CSR部件时可以先使用方法①实现，之后再修改为方法②，先实现后完美。

### 思考与探索

1. 查阅资料，哪些CSR寄存器常用CSR立即数指令进行读写操作？
2. CSR指令的写（write）、置位（set）、清除（clear）分别在什么情况下使用？请结合例子说明。
3. OS是怎么知道硬件的信息的呢？
4. 谈谈你在实验中碰到了哪些问题？又是如何解决的？