COA2023-programming08

1 实验要求

1.1 实验概述

本次实验我们要求完成 cpu 中的控制器部分，模拟每个时钟周期取指、间址、执行、中断四种操作。我们使用 RISC-V 指令来完成本次实验。本次实验仅涉及到 add, addi, lw, lui, jalr, ecall 几种指令，在以下实验指导中均会给出具体格式。

1.2 实验内容

1. 完成 tick 方法，在每个时钟周期内根据 ICC 的状态进行对应操作

```
public void tick()
```

2. 完成取指、间址、执行、中断四种操作

```
//取指
private void getInstruct()
//间址
private void findOperand()
//执行
private void operate()
//中断
private void interrupt()
```

2 实验攻略

2.1 代码结构

在完成了前面的实验后，我们加入了模拟控制器部分，整体代码结构如下所示。本次实验需要使用到 Controller 文件。

```
.
|  pom.xml # Maven Config File
|  README.md # This File
|
├─.idea
|
└─src
   ├─main
   |  └─java
   |      └─cpu
   |        |
   |        |
   |        ├─controller
   |        |     Controller.java # 控制器类，需要修改
   |        |
   |        └─memory
   |           |  Memory.java    # 主存类，不需要修改
   |           |
   |           └─cache
   └─test
       └─java
           └─controller
                ControllerTest.java
```

2.2 实验指导

2.2.1 时钟周期的实现

本次实验我们使用一个 tick 函数来模拟每个时钟周期。在每个时钟周期中，我们需要完成三件事：

1. 判断 ICC 中内容，得到当前处于哪个时钟周期
2. 执行对应周期的指令微操作序列
3. 根据指令执行情况判断 ICC 的下一个状态

ICC 是一个 2 位的寄存器，请参考课堂讲解内容完成对应状态的判断。

在判断 ICC 下一个状态时请注意，我们在框架代码中提供了一个中断控制器，其中包含了表示中断是否发生的标志，请参考代码中 InterruptController 部分。是否进入间址周期的判断请阅读 2.2.3 节间址周期的实现。（你可以在完成间址周期的处理和中断的处理后再回到时钟周期的实现）

2.2.2 取指微操作序列

在取指过程中，我们要进行以下操作：

1. 将 PC 中的内容加载到 MAR（PC 中的内容会被初始化为全 0，即我们默认程序开始位置为主存开始位置，此处我们忽略了系统程序，请阅读测试用例对整个流程进行理解）
2. 根据 MAR 中保存的地址，读取 memory 中对应内容到 MBR（请注意 memory 中读出的数据是 byte 数组类型，而寄存器类型是 char 数组）
3. 增加 PC 到下一条指令的位置（此时 PC 应该加上多少？为什么？考虑指令的长度）
4. 将 MBR 中的内容装载到 IR 中

取出的指令格式请参考：reference-card.请注意我们只使用在 1.1 节提到的指令。

2.2.3 间址周期的实现

由于 risc-v 并不具备间址类型的指令，因此我们额外规定一种间址指令 addc。根据 reference-card 我们可以得到正常的 add 指令 opcode 为 1100110，此处规定 addc 的 opcode 为 1101110.只有寄存器 rs2 中保存的是操作数的地址。所以如何判断是否需要进入间址周期的方式就交给聪明的你了！

在间址过程中，我们规定了以下操作：

1. 将 rs2 中的内容加载到 MAR 中
2. 根据 MAR 中的地址读出内存中对应数据存回 rs2 中

在完成间址周期后，我们就可以和正常的 add 指令一样进入执行周期了。

2.2.4 执行周期的实现

执行周期需要根据不同的 opcode 进行不同的操作。此处 add 指令可以调用 ALU 中已经实现好的加法进行。请将对应结果存到相应的位置中，在测试中，我们将对寄存器和主存进行检查。本次实验我们不设置隐藏用例，请同学们认真阅读测试用例中的 memory 部分进行 debug 工作。

有两个指令在执行阶段需要我们特殊关注：

- jalr：保存并跳转指令。在改变 PC 之前，我们要先将返回的位置保存到 ra 寄存器中，我们规定 GPR 的第 1 个寄存器是返回地址寄存器（第 0 个 GPR 寄存器保存 0）
- ecall：系统调用中断指令。同样要保存返回位置，同时要设置中断控制器。

请注意，寄存器和立即数的下标在指令中为了方便处理采用大端存储的方式，即从低到高直接截取转化为十进制即可。不明白的同学请参考测试用例。

2.2.5 中断

本次实验中，我们使用 ecall 指令来模拟中断操作。在中断发生时，系统要保存程序的返回位置（是多少？），以便完成中断处理程序后返回原有程序。

此处我们使用 handleInterrupt 来模拟中断程序的实现。更多有关中断的内容同学们将在操作系统中学到，此处我们不作过多说明。执行完中断操作后，不要忘记将允许中断位置为 false。至此，你就完成了本次实验的所有工作！