Lab 2 Report

PB20111623 马子睿

实验内容

- 完成RV32I五级流水线的数据通路
- 处理流水线的数据相关问题
- 实现CSR寄存器和流水线的适配

工作总结

在本次实验中，我将第一阶段和第二阶段合并完成，第三阶段单独完成。下面我来对我的工作一一介绍：

RV32I五级流水线补全

这是第一部分和第二部分的工作，我主要完成了：

- ALU:

  完成了ALU的基本运算操作。这一阶段我并没有理解NAND的运算是如何进行的，我将其实现为了与非运算，但RV32I指令集中是没有这个运算的。

- Branch：

  完成了Btype分支单元的设计。这个单元主要是通过比较来确定是否进行分支。对于有符号比较，我这里使用了一种简单的方法来进行实现：

  ```
  $signed(srr1) < $signed(sr2)
  ```

- NPC：

  NPC主要负责选择下一PC的值。这里有一些值得注意的问题：

  - JAL在ID阶段跳转，当JAL和前一条JALR或BRANCH同时跳转时，应该优先EX阶段的JALR或BRANCH跳转。
  - PC已经是加四之后的PC了，不需要再加四了。

- Decoder：

  Decoder只需要将对应的信号生成逻辑补全即可。其中的诸多信号都在Lab1中有了定义，因此不需要只需要将对应的信号补齐即可。

- DataExtend：

  DataExtend是将DCache中的读出的数据根据地址进行移位。但由于移位都是固定位数的，所以不需要使用移位器（因为桶型移位器的时延也会较大），只需要使用几种位拼接进行固定位宽移位即可。

  这里需要注意的是，对于有符号拓展，我们也应当使用补高位的方法进行为拓展连线。

- ImmExtend：

立即数生成器只需要根据不同类型的指令进行对应的立即数拓展即可。对于RISCV而言，不同类型的立即数会有一定的重叠，但这一部分我认为不必要特定进行考虑，因为那样会加大立即数生成的工程量。这一部分可以交给编译器来优化。

- Hazard：

  在流水线中，我们应该注意以下几种hazard：

  - 数据前递：当Mem段或WB段的写地址和EX段的读地址相同时，需要把对应的写数据前递，但有两种情况比较特殊：
    - 当写地址是0时，不可以前递
    - 当MEM和WB段的数据都可以前递时，应当优先MEM段数据前递
  - Load-use冲突：当EX段是加载指令且写地址与ID阶段的读地址相同时，需要在这里插入一个气泡（给PC和IF-ID寄存器一个停顿信号，给ID-EX寄存器一个冲刷信号）。这主要是因为，Load的结果需要在WB段才可以前递，因此ID阶段的运算指令无法通过前递得到对应的数据。
  - 跳转冲突：由于我们的分支预测方法就是一直预测不跳转，所以满足跳转阶段时，如果这个跳转由jalr或branch引起，那么需要给IF-ID寄存器和ID-EX寄存器一个冲刷信号。如果这个跳转由jal引起，那么需要给IF-ID寄存器一个冲刷信号。

CSR设计

在代码框架中给除了CSR的设计，CSR放在了EX段，这一部分的工作量不大，主要有四部分：

- CSR_EX段间寄存器：和其他段间寄存器逻辑完全一样，补齐即可
- CSR_Regfile：这个寄存器不应该是写优先的，因为它直接放在了EX段，是单周期更新的。其余与通用寄存器完全一致。只不过，CSR寄存器的地址空间为32位，实在是过大，因此最好采用对地址进行选择的方法进行访问。
- CSR有关的控制逻辑：
  - CSR_write_en：若当前指令是CSR相关指令，那么这个信号必然有效
  - CSR_zimm_or_reg：这个信号和CSR指令类型有关。如果本条指令和CSR立即数运算有关，那么应当设置为有效，否则无效。
- ALU的NAND运算：这个运算可以用来解决CSRRC指令的运算，由于CSR数据从sr2而来，因此应当实现为：

  ```
  ~op1 & op2
  ```

实验总结

遇到的问题

- NPC选择的PC问题：

  最开始时没有注意观察数据通路，这个PC是加四之后的PC，所以一开始测试时PC每次要+8，后来根据数据通路，修正了这个问题

- 算数右移问题：

  最开始时只使用了$signed标记来标识算数右移，但经过测试发现这样依然是错误的。查找相关资料后发现，如果要在Verilog中使用算数右移，必须使用">>>"三个右箭头才可以实现

实验收获

- 再一次重温了RISCV五级流水线，这一次又加入了CSR，使得整个流水线语法完整，让我逐步构建起了对于流水线拓展的知识图谱
- 回顾了Verilog和Vivado的使用方法
- 再次体会了流水线中冲突的解决方式，学习到了处理CSR冲突的简单方法

实验建议

- 本次实验的DCache复杂度太高，其中的地址空间太大导致了RTL不经特殊处理无法运行。但RTL是纠正代码错误的重要手段（例如找位宽不匹配、锁存器等）。希望之后的实验可以简化部件的设计
- 译码单元对于一些内容做了二次译码，例如br_type，本来就可以直接将funct3接到上面，没必要重新进行编码，因为EX阶段的跳转依然要译码
- 如果真实进行上板的话，不是非常建议在ID阶段就让jal跳转，因为ID阶段要读寄存器，这个组合延迟还要加上NPC选择的延迟，未免有些过大。在EX阶段跳转可以更好的节省时序。