My risc-v core

risc-v作为当前非常火爆的开源risc指令集，受到了各方的关注。自己设计一款基于risc-v的处理器核，不仅能锻炼自己的设计能力，而且还能学习当前的risc-v的指令集

第一版

第一版的处理器核我打算只支持RV32I指令集，后续可以在此基础上不断添加新的扩展实现新的功能。第一版设计为5阶段流水线，只包括简单的处理器内核，并不包括外部的各种外设接口。存储器使用ITCM核DTCM。cpu为定序执行。

1. 取指(IF)

第一版的取指模块设计得非常简单，只包含PC程序计数器，并不包含其他任何的分支预测功能。由于没有使用RV的压缩指令集，所以取指的地址总是和4byte对齐的（地址最低两位始终为0），这种情况下不需要使用剩余缓存技术。

取指模块设计时需要考虑到分支跳转指令，PC所指向的下一个位置由多路选择选择上一次地址的自增还是跳转后的地址。

ITCM使用FPGA中的寄存器搭建，写入需要一个时钟周期，读出为组合逻辑，在同一个时钟周期即可完成。在本次设计中不允许在cpu运行过程中对ITCM进行修改，仅在初始化时允许外部接口向ITCM写入数据。

2. 译码(ID)

RV32I指令集规定两个源操作数寄存器索引和目的寄存器索引在指令中的位置固定不变，所以对于寄存器索引的译码比较简单。

RV32I规定了立即数的5种格式，译码过程中需要根据指令的类型生成32bit的操作数。

译码模块还需要生成一系列控制信号，包括多个多路选择器的选择信号，ALU的操作码等。

3. 执行(EX)

执行阶段使用ALU进行计算。

4. 访存(MEM)

访存使用的DTCM和ITCM构造一样，允许cpu进行读写。

5. 写回(WB)

向目的寄存器写回的数据有四个来源：ALU的计算结果（常规计算指令）、DTCM读出的数据（LW指令）、指令中的立即数（LUI、AUIPC）、PC+4（JAL、JALR）。

6. Hazard的处理

第一版由于设计的是定序处理器，所以数据阻滞只包括RAW。本次数据阻滞的处理借鉴了《Digital design and computer architecture》介绍的方法。当某条指令执行到EX阶段时，判断是否所需要的数据依赖于上两条指令写回的数据时。如果存在依赖那么直接将数据从MEM、WB阶段foward到EX阶段。但是forward的方法当MEM阶段执行的指令是lw时变得不适用，此时需要将ID、IF阶段stall一个时钟周期。

控制阻滞的处理没有使用复杂的方法，一旦发生分支跳转的时候，直接将ID、EX阶段flush。在这里也并没有对JAL、JALR这种无条件跳转指令做额外的处理，打算在下一版设计中将这部分加上。

Hazard的处理使用Hazard Unit来完成。

7. 整体结构图