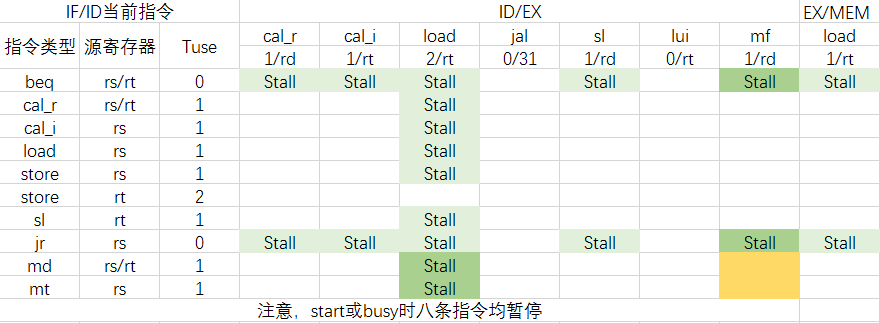
思考题

1. 为什么需要有单独的乘除法部件而不是整合进ALU？为何需要有独立的HI，LO寄存器?

答：因为乘除法和ALU中其他运算相比，速度慢很多，是几十倍的差距，因此一旦放ALU将导致流水线性能大大下降，E级延迟过高；而采用独立乘除模块，则避免了阻塞所有后续指令，一定程度上降低了延迟。同时设计独立的寄存器存储乘除法结果可以保证不会因为乘除运算而加大WB阶段延迟，单纯地读取HILO就是普通的流水线指令。

1. 参照你对延迟槽的理解，试解释“乘除槽”。

答：乘除槽即乘除指令之后紧跟着的一条指令，类比延迟槽，这条指令如果和上一条乘除类指令同属乘除类或者产生了数据相关比如下图：

则需要阻塞，但如果是其他不相关的流水级指令则可以继续执行。

1. 为何上文文末提到的lb等指令使用的数据扩展模块应在 MEM/WB 之后,而不能在 DM 之后?

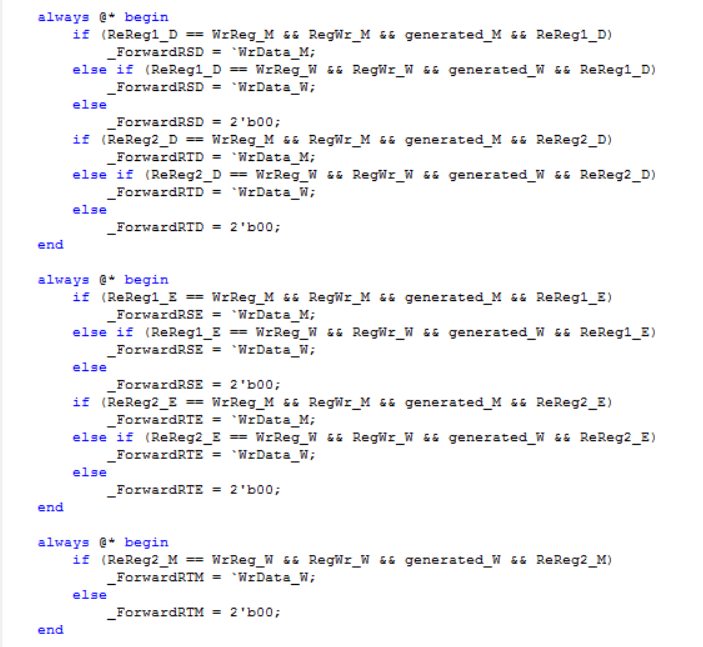
答：因为本身M级延迟已经很高了，如果再加上一个数据扩展模块，将进一步提高这一流水级的延迟，严重影响流水线性能，而如果放在本来就比较快的W阶段则较好地平衡了流水级之间的延迟，降低了对性能的影响。

1. 举例说明并分析何时按字节访问内存相对于按字访问内存性能上更有优势。（Hint： 考虑C语言中字符串的情况）

答：比如读写字符串时，每个char均为一个字节，如果此时按照字访问内存则灵活性大大降低且需要更多访问次数（每次写入字节扩展出来的一个字）；同样地还有在读写结构体或UNION时因为这两者更容易出现不同类型数据混杂在一起并导致数据存储在不同的机器字中的情况，此时如果继续按照字读取，则需要额外的处理，截断并重新拼接，相较于读取字节并直接拼接会慢许多。

1. 如何概括你所设计的CPU的设计风格？为了对抗复杂性你采取了哪些抽象和规范手段?

答：我的CPU在处理冲转发和暂停的时候采用了两种不同的思路，转发我使用了侦测者类型的架构，但是在指令分类上采取了PPT做法，如下图：



通过Control模块统一根据指令分类产生对应的读写寄存器需求和数据相关信号之后，直接判断并转发即可，并且可扩展性极强，对于新指令只需在control中按照需求分类即可，完全无需改动转发（相比较PPT还需要根据指令类型改转发）。暂停机制我采用了PPT的Tuse Tnew分析方法，这样在实现过程中也可以保证按照类别即可。（相比较mooc暂停和转发逻辑设计推导正确便无需debug）

为了对抗复杂性，我将指令按照需求分类抽象：将转发按照匹配流水级寄存器的方式实现；大量运用三目运算符，减少if类语句

为了提升规范性，我尽量减少了always实现的组合逻辑；将模块信号按照流水级后缀统一命名；将各个模块按照顺序在datapath中书写调用；大量采用宏定义确认指令信号。

1. 你对流水线CPU设计风格有何见解？

答：首先我觉得在设计一开始就要考虑到不仅是当前需求的实现，还需要给进一步扩展留下空间，比如在从十指令到五十指令的过程中，采用宏定义和assign而不是if/case就很好地提升了可扩展性；第二，设计风格没有最好，只有适合自己思路的风格，所以需要在学习和讨论的基础上形成适合自己思路的风格，比如我将mooc谈到的两种风格结合使用；第三，命名一定要规范，这有助于逻辑关系的形成和Debug；最后，如果可以重构，尽量减少代码数量，降低复杂度，达到两者平衡，是最好的。