实验名称：五级流水线CPU设计

实验人：谢志康

学号：22307110187

实验时间：2024.5.3

参考学习——

[关于流水线的三种冒险 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/447682231)

[手把手代码实现五级流水线CPU——第三篇：流水线控制逻辑\_fwdb-CSDN博客](https://blog.csdn.net/Strive_LiJiaLe/article/details/128519372)

[从零开始写riscv处理器（五）数据冒险：停顿与前递\_load use数据冒险-CSDN博客](https://blog.csdn.net/weixin_45774715/article/details/132744848)

流水线三种冒险情况——

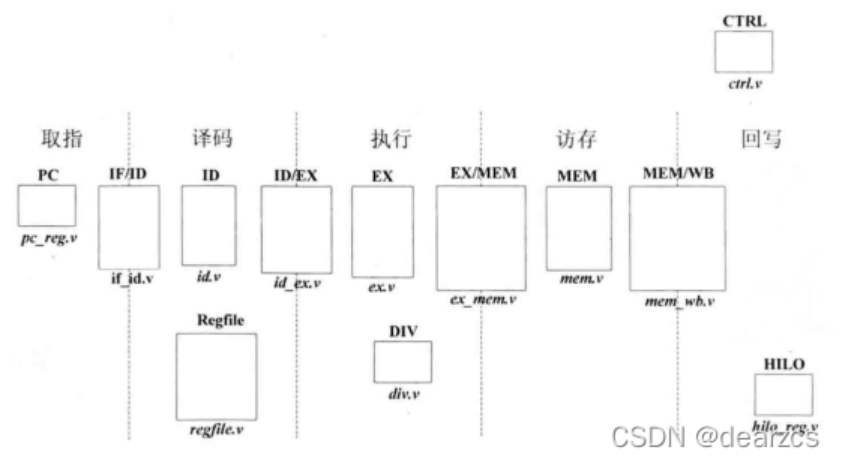
1. **数据冒险**：当指令在流水线中重叠执行时，后面的指令需要用到前面的指令的执行结果，而前面的指令尚未写回导致的冲突，称为数据冒险（也称为数据相关性）。
2. 结构冒险：当一条指令需要的硬件部件还在为之前的指令工作，而无法为这条指令提供服务，那就导致了结构冒险。（这里结构是指硬件当中的某个部件、也称为资源冲突）。

（我们架构ibus和dbus分开，不存在此问题）

1. **控制冒险**：如果现在想要执行哪条指令，是由之前指令的运行结果决定，而现在那条之前指令的结果还没产生，就导致了控制冒险（实际上就是riscv的跳转指令引起的）。

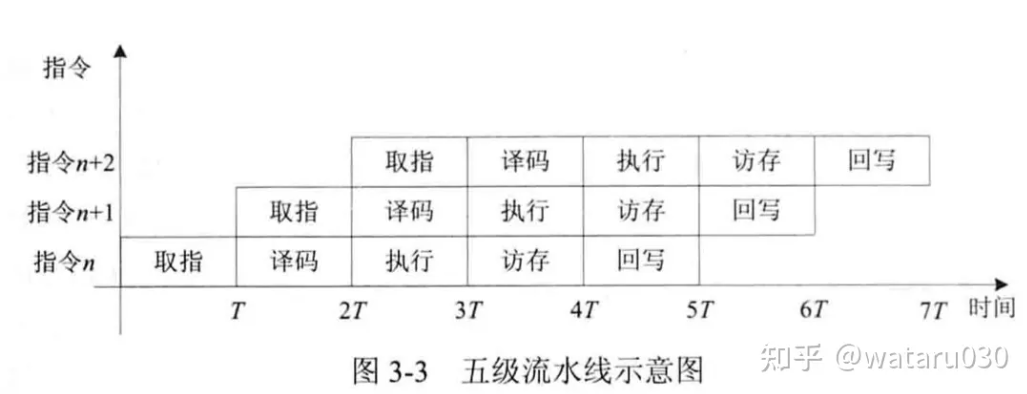
Div\_rem\_sig

整体上在之前多周期CPU代码上完成，基本功能实现代码与之前相同，将CPU执行方式改成五级流水线架构。参考以下结构设计——



**Figure 1**

在我原本的CPU中，ex阶段的乘除余数等指令是多周期，使用暂停处理（没算完不取下一条instr），mem阶段访存也是多周期，同样使用暂停处理（访存未结束stall住，不执行wb操作，等访存结束整个CPU继续跑），将这样的多周期CPU改成流水线，本质上和单周期到流水线相同。如下图所示——



**Figure 2**

执行时间和访存时间不止一个T，但每一次操作都取总周期数最大的一个作为当前操作的周期数。譬如，译码、执行、访存同时进行，译码所需1周期，执行除法所需64周期，访存所需10周期，总的这一次操作，CPU会在stall\_this\_alu\_div信号下停止64个周期（代码实现上就是空转），等计算完成后继续往后跑，到下一个流水线时间段。

模仿figure1的方式，在原本的架构上新增了IF\_ID\_PIPE, ID\_EX\_PIPE, EX\_MEM\_PIPE, MEM\_WB\_PIPE四个中间过渡模块，即用来实现流水寄存器。新增hazard模块处理各种冒险情况。其余如fetch.sv, control.sv进行了微小改动（新增接口用于初始化if\_id的某些信号），其余模块无需改动。

**IF阶段**：Fetch.sv NPC.sv PC.sv

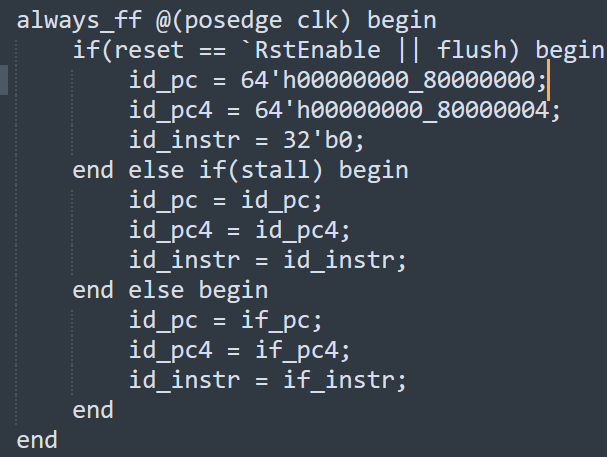
Fetch.sv取指，若一切准备就绪（指当前周期访存完成，下周期指令已经准备好），从ibus中取出下一周期的指令if\_instr。同时，对id\_instr进行初始化，取if\_instr时将id\_instr链接过去（这里是debug试错试出来的），如果在本周期if\_instr取指成功，下个周期就应该执行本条instr，而若没有初始化，id\_instr为空值（起始状态），整个CPU没法跑起来，因此将他们先行链接，保证CPU往下跑，而id\_instr并不会一直同步为if\_instr的值，在IF\_ID\_PIPE模块会根据flush以及stall\_hazard传入值情况将其修改为正确的要执行的id\_instr。达到了instr在IF\_ID过渡阶段的正确传递功能。

NPC.sv利用ex阶段算得的各个值决定下一个PC。新增了一个接口，output一个信号HD\_br，告诉hazard模块遇到了分支指令要进行处理（不止b型指令，还有j型）

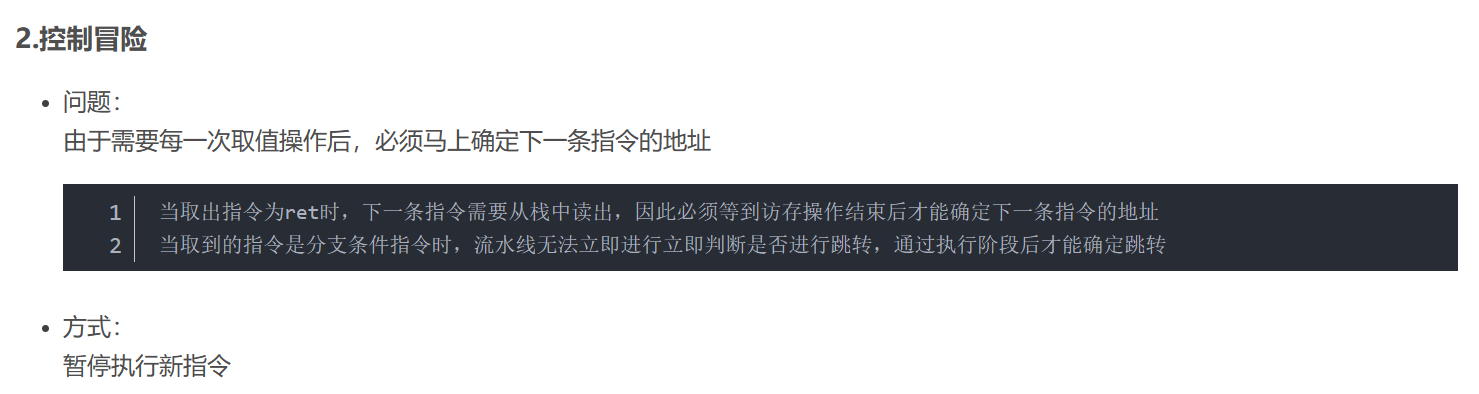
PC.sv根据NPC值给出下一个周期的PC值。

**IF\_ID阶段**：IF\_ID\_PIPE.sv

IF\_ID\_PIPE作用将取指阶段的结果（if\_pc, if\_instr）在下个时钟周期传到译码阶段，特别的，输入由hazard模块判断出的stall\_hazard和if\_id\_flush信号（后面hazard模块详述）



Flush信号表示遇到了分支指令。即，上一个周期解析的id\_instr为分支指令，上一个周期flush被置为1，此时应该清空IF\_ID流水线寄存器，达到正确处理跳转指令的目的。具体功能是，将id\_instr设置为0，即相当于插入了一个周期的bubble，其它器件先跑空指令（啥都不做），等到本周期处理完是否跳转再取到正确的NPC。



Stall指令表示出现数据冒险（相关性），采用暂停的策略，空跑CPU。

**ID阶段**：SEXT.sv Control.sv ALU\_input\_MUX.sv ALU\_SIGN.sv

这几个模块较之前多周期架构没有改变。唯一ALU\_input\_MUX中先一步将ex\_oprand与id\_oprand进行了链接，也是为了处理初始化的问题。即，第一次id\_oprand已经有值，ex\_oprand还未定义，下一个周期ALU开始计算ex\_oprand会出现问题，因此提前将其链接，若链接错误，下一个周期ex\_oprand会根据前递信号、顺序执行信号和rst信号重新得到正确的值，若链接正确则就是直接进行计算（选择顺序执行信号），不会产生问题。

SIGN模块有一个地方bug搞了很久，是ex\_signed\_A ex\_signed\_AW ex\_signed\_B ex\_signed\_BW四个信号的阻塞赋值或是非阻塞赋值的问题，若非阻塞，他们会在下个周期才被修改，而下个周期已经要靠这四个信号计算divu，mulu等指令了，就会乱掉。因此最后采用阻塞赋值，在当前周期就被顺序执行掉，相当于下个周期ex要执行时，所有原操作数已经准备好，就可以直接开始计算不会出错。

**ID\_EX阶段**：ID\_EX\_PIPE.sv

ID\_EX阶段的流水寄存器。Id\_ex\_flush则清空流水线寄存器（相当于加入bubble，该信号由hazard模块产生），特别的，两个源操作数由forward信号给出（hazard模块产生），用于处理数据冒险（采用前递的方式处理数据冒险，具体在hazard模块解释）

**EX阶段**：RegFile\_wD\_MUX.sv ALU.sv ALU\_DIV.sv ALU\_DIVW.sv ALU\_MUL.sv

ALU\_output\_MUX.sv

与多周期架构相比没啥变化，算不完的还是stall，不会影响啥。

**EX\_MEM阶段**：EX\_MEM\_PIPE.sv

与之前相同。

**MEM阶段**：MEM.sv

访存模块，与之前相同。

**MEM\_WB阶段**：MEM\_WB\_PIPE.sv

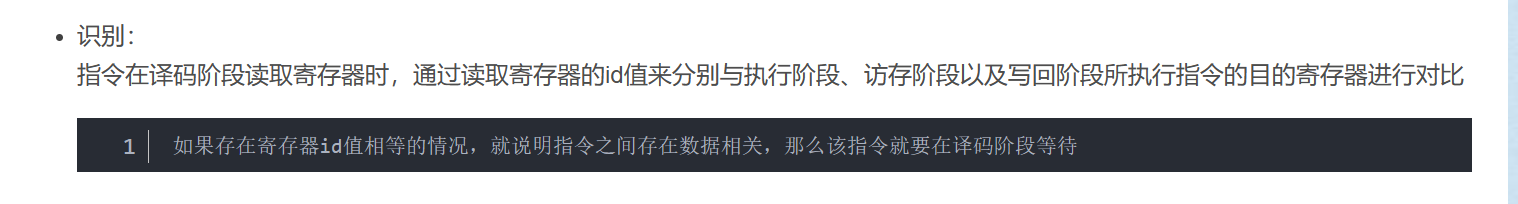
与之前相同。

**WB阶段**：RegFile.sv

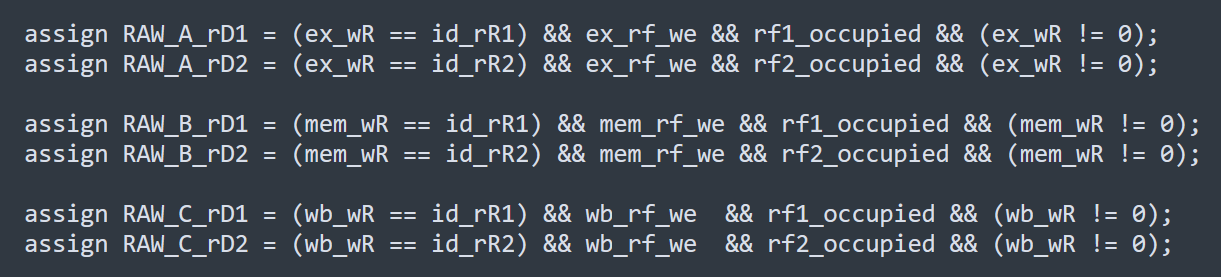
寄存器写回模块，与之前相同。内存写回在MEM阶段做了。

**冲突冒险处理：Hazard.sv**

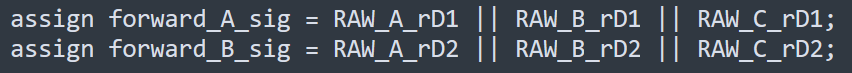
数据冒险处理——



通过取寄存器id值分别于ex mem wb阶段的write\_register相比来依次检测相邻间隔、间隔一条指令、间隔两条指令的冒险。

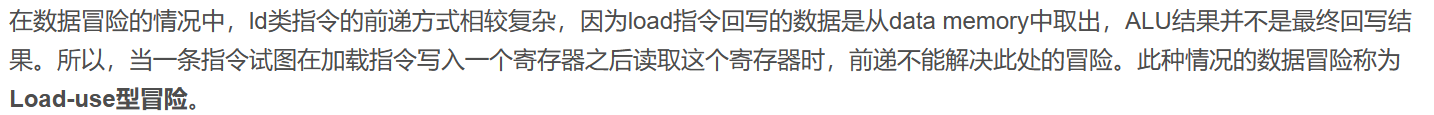


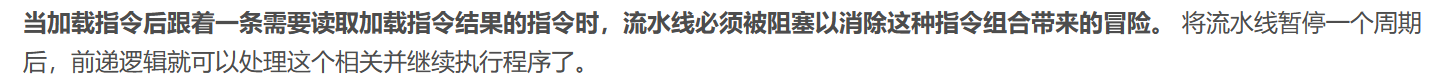
使用前递的方式处理冒险，若有任意冒险，将前递使能置为1。



接下来确定前递oprand值，考虑到可能同时发生多次冒险，按照A>B>C的优先级将操作数前递（前递具有优先性，应先处理EX冒险）。详见hazard.sv代码。

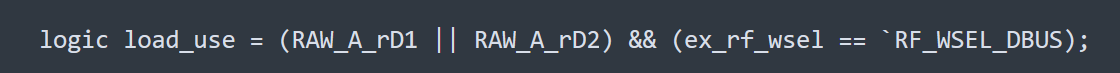
最后处理特殊的load\_use型冒险：







解决方法对应代码——



如果load\_use型数据冒险存在，那么PC和IF\_ID流水线寄存器停顿一个周期(stall\_hazard)，清空ID\_EX流水线寄存器(id\_ex\_flush)。

在静态分支预测部分中，接受NPC传来的跳转信号，如果发生跳转则清空if\_id id\_ex这两个流水线寄存器。

整体五级流水线CPU架构及各组件解释如上

通过运行截图——

