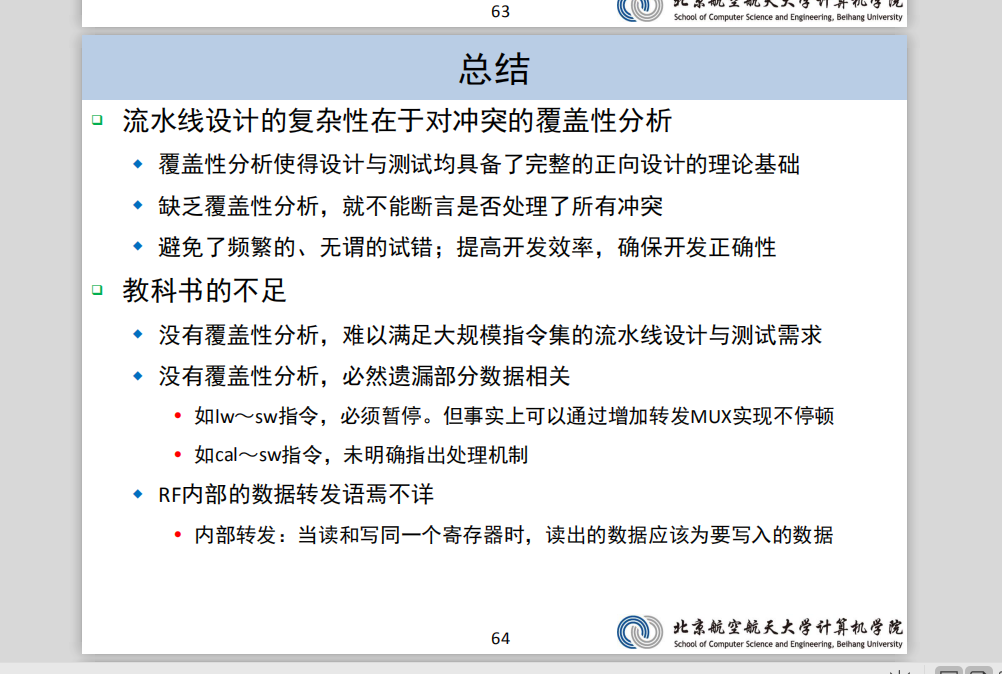
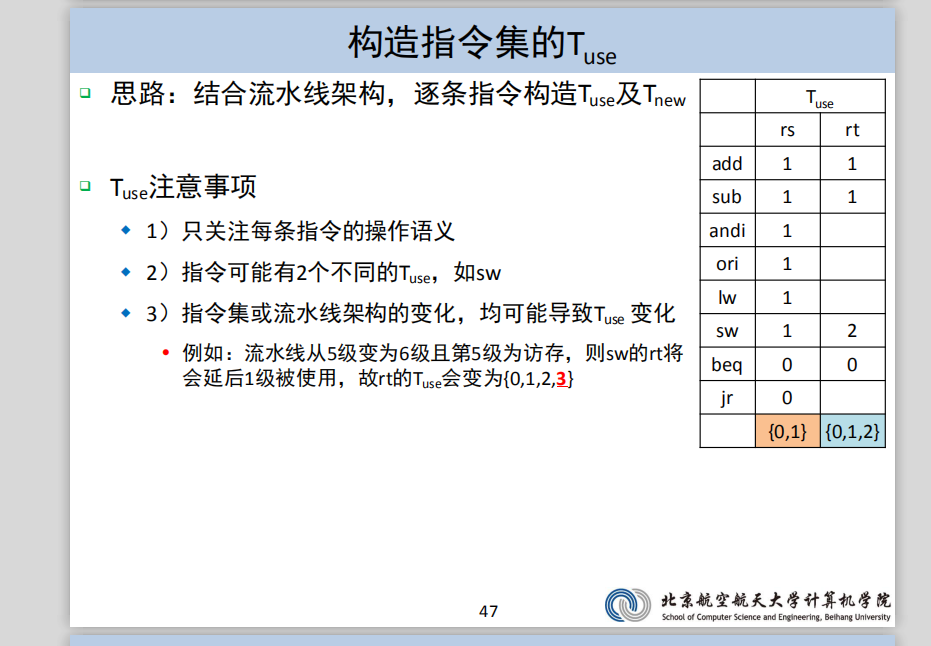


Tnew逐级减1

Tuse < Tnew暂停

Tuse >= Tnew 转发

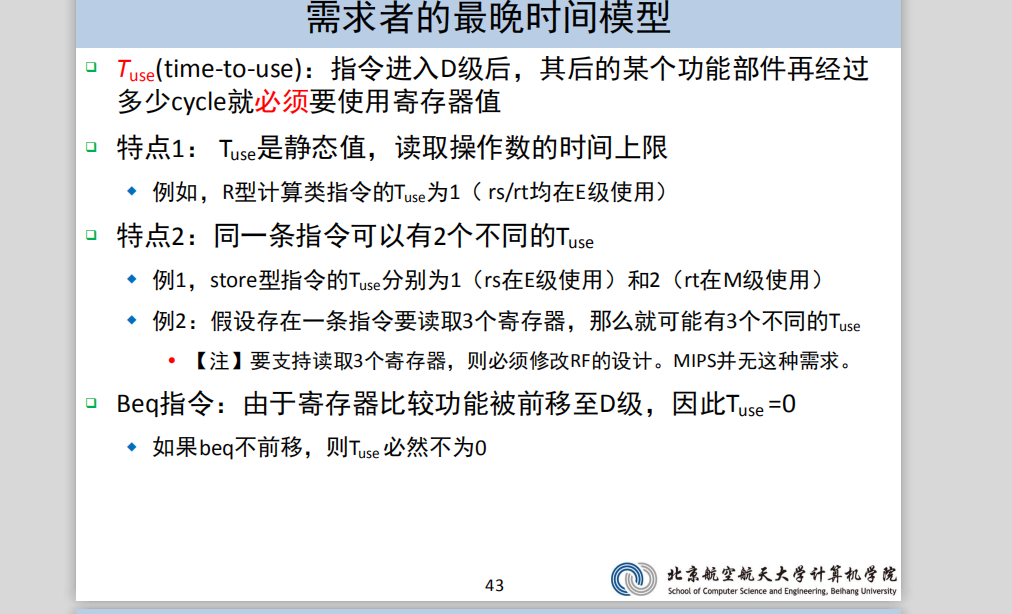
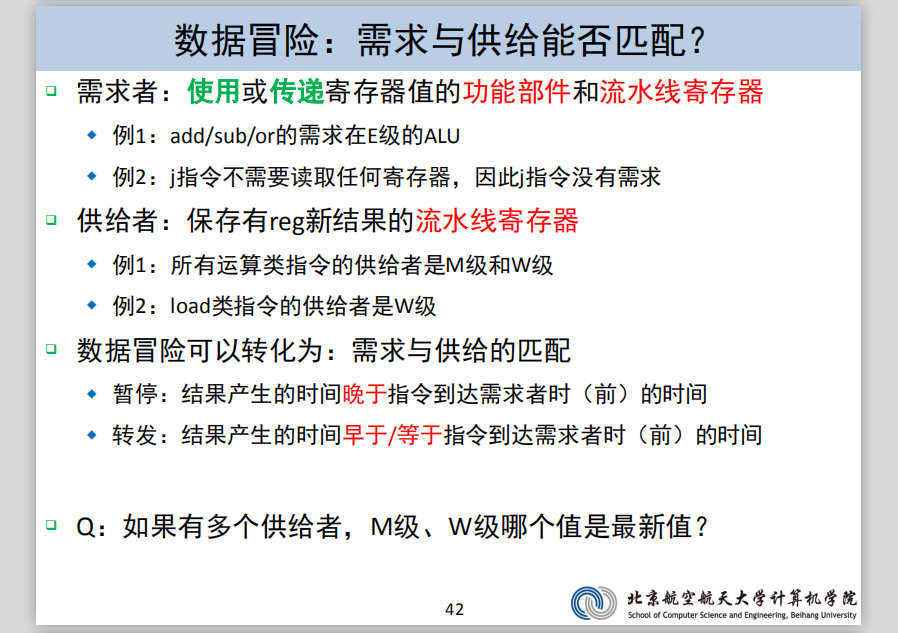


韦学长中的timer是算Tuse的吧

数据冒险是针对寄存器操作的指令

Tuse还可以理解成是流过多少级，毕竟和时钟上升一个样

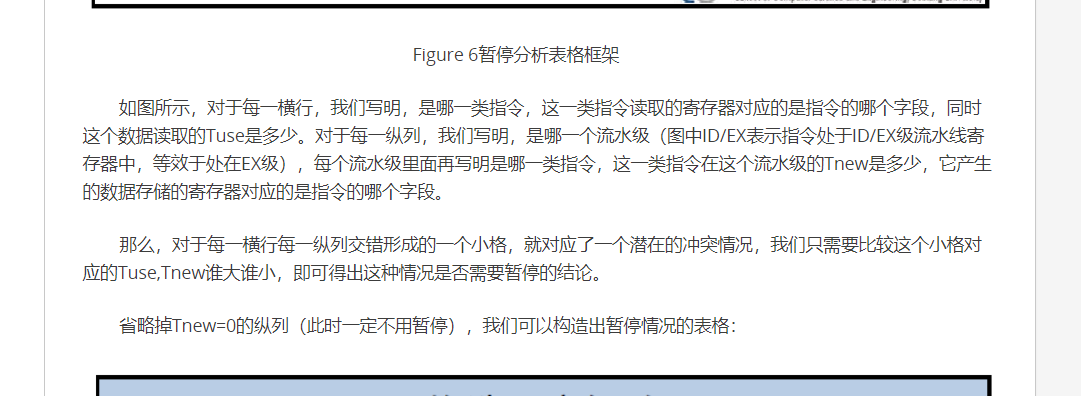
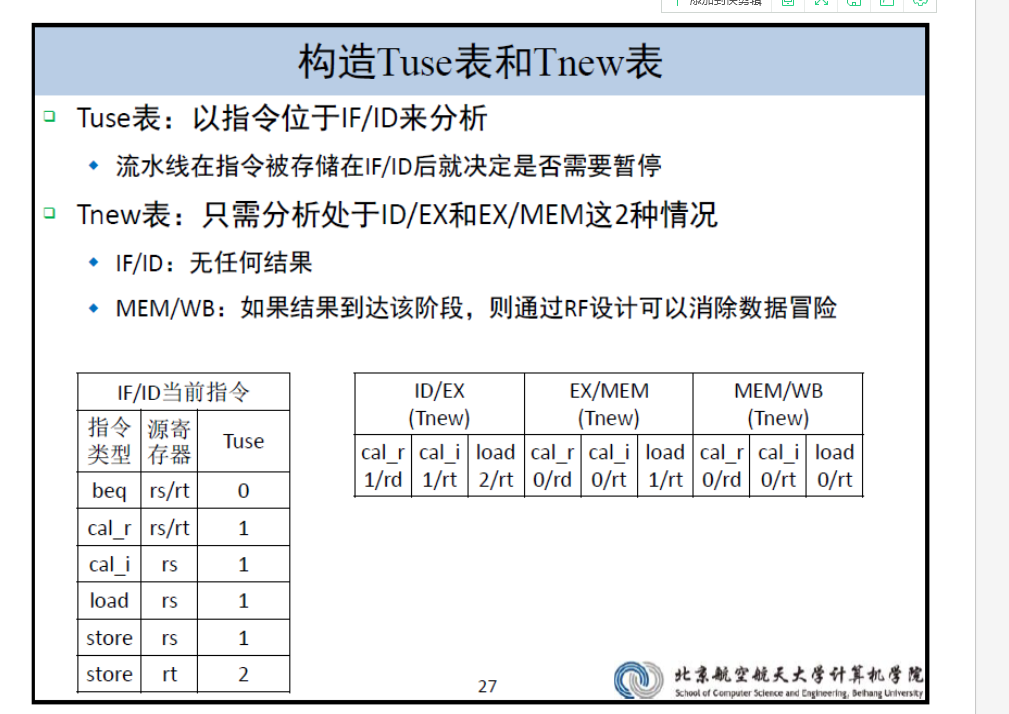
指还要多少个周期就得使用寄存器的值

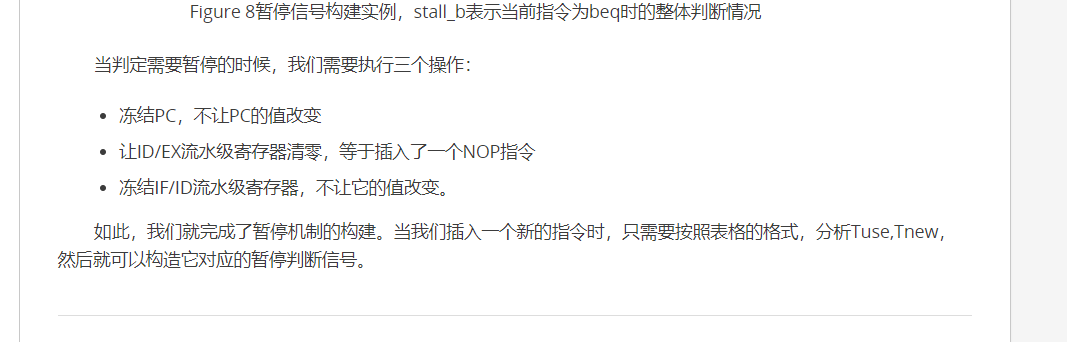
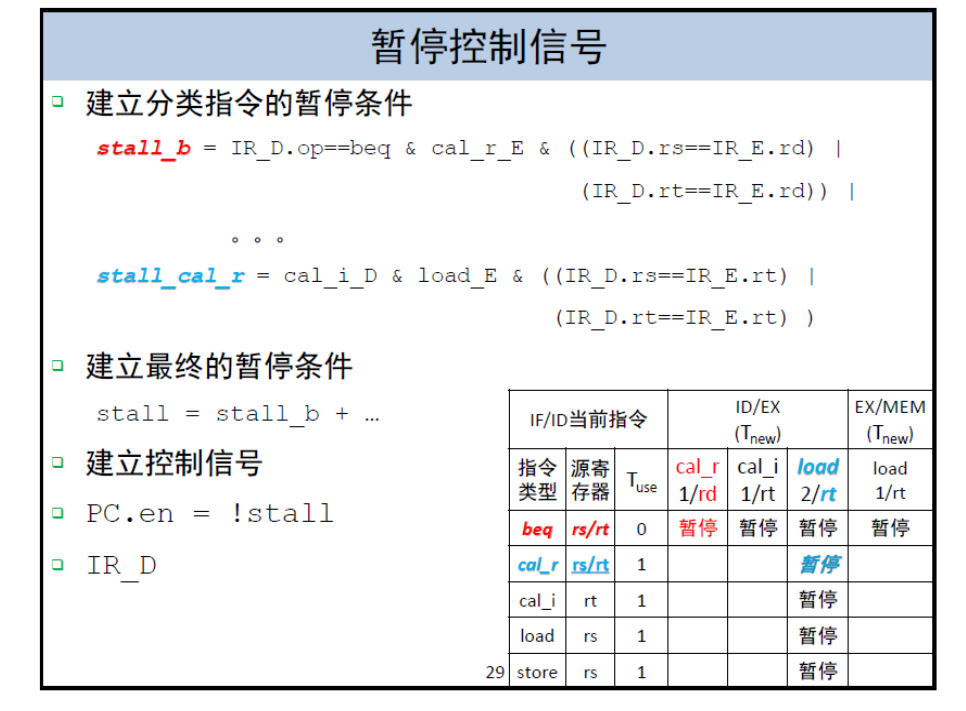


指令到哪一级应该是理解成没存进去，得上升后才存进去吧？？

**但是理解很别扭，不如理解成得时钟再上升才能更新吧，这是因为供给者必须是寄存器，需要一个时钟周期更新从而转发**

定义模板MUX，就像韦学长那样

可以看到，通过这种方式，我们简单明了的分析出了什么情况需要暂停。当我们需要新加指令的时候，如果指令属于已经分析过的指令类，则此表格完全不用修正。如果指令属于新的指令类，则我们可以依照工程化的分析法：如果指令有输出，则加入新的纵列；如果指令有输入，则加入新的横行；然后完善Tnew,Tuse和目标寄存器信息；接下来即可非常简单的对这个表格做出补充，得出加入新指令之后的暂停表格。



  在流水线CPU设计中，不可避免地会出现大量的端口和用来在端口中传输数据的wire变量，以及由各流水级部件产生的大量数据端口，如何给它 们命名是一个重要的问题。虽然命名方式各有习惯，但是有一些原则是可以明确的。对于数据信号，明确指出信号名、阶段名、方向， 如”ALUout\_E\_out”，代表E阶段ALU的计算结果的输出端口，传送到EM级寄存器。总之，无论采取何种命名原则，务求意义明确，避免歧义，从 一而终。在代码输入上多花费些时间来避免大量的潜在风险，是完全值得的。

   几乎每一条指令，都需要获取一定的数据输入，然后某些指令还会产生数据输出。流水线之所以会产生冒险，就是因为后面指令需求的数据，正好就是前面指令供给的数据，而后面指令在需要使用数据时，前面供给的数据还没有存入寄存器堆，从而导致后面的指令不能正常地读取到正确的数据。因此我们从需求数据和供给数据的行为来入手分析暂停、转发情况。

****需求者****：对于某条指令，实际上需求寄存器数据的是某些硬件部件。如，对于addu指令，需要数据的是位于流水线E级的ALU，对于BEQ指令，需要数据的是位于流水线D级的比较器CMP。而对于SW指令，需要数据的有EX级的ALU（这个数据用来计算存储地址），还有MEM级的DM（这里需要存入的具体的值）。

****供给者****：所有的供给者，都是存储了上一级传来的各种数据的流水级寄存器。至于为什么一定要由流水级寄存器来提供数据而不是由ALU或者DM来提供数据，超出本文讨论范围，各位不妨暂时理解为这是一项硬性的设计要求，具体原因有兴趣可以查阅相关资料或者与老师同学讨论。

分析清楚了数据的需求者和供给者，我们就可以理清处理数据冒险的策略了。假设当前我需要的数据，其实已经计算出来，只是还没有进入寄存器堆，那么我们可以用****转发****(Forwarding)来解决，即不引用寄存器堆的值，而是直接从后面的流水级的供给者把计算结果发送到前面流水级的需求者来引用。如果我们需要的数据还没有算出来。则我们就只能****暂停****(Stall)，让流水线停止工作，等到我们需要的数据计算完毕，再开始下面的工作。

**对于某一个指令的某一个数据需求，我们定义需求时间Tuse为：这条指令位于D级的时候，再经过多少个时钟周期就必须要使用相应的数据。**

        例如，对于BEQ指令，立刻就要使用数据，所以Tuse=0。

        对于addu指令，等待下一个时钟周期它进入EX级才要使用数据，所以Tuse=1。

        而对于sw指令，在EX级它需要GPR[rs]的数据来计算地址，在MEM级需要GPR[rt]来存入值，所以对于rs数据，它的Tuse\_rs=1,对于rt数据，它的Tuse\_rt=2。

* 特点1：是一个定值，每个指令的Tuse是一定的
* 特点2：一个指令可以有两个Tuse值

**对于某个指令的数据产出，我们定义供给时间Tnew为：位于某个流水级的某个指令，它经过多少个时钟周期可以算出结果并且存储到流水级寄存器里。**

        例如，对于addu指令，当它处于EX级，此时结果还没有存储到流水级寄存器里，所以此时它的Tnew=1，而当它处于MEM或者WB级，此时结果已经写入了流水级寄存器，所以此时Tnew=0。

* 特点1：是一个动态值，每个指令处于流水线不同阶段有不同的Tnew值
* 特点2：一个指令在一个时刻只会有一个Tnew值（一个指令只有一个结果）

        当两条指令发生数据冲突（前面指令的写入寄存器，等于后面指令的读取寄存器），我们就可以根据Tnew和Tuse值来判断策略。

1. Tnew=0，说明结果已经算出，如果指令处于WB级，则可以通过寄存器的内部转发设计解决（关于内部转发请自行查阅资料思考或与同学讨论），不需要任何操作。如果指令不处于WB级，则可以通过转发结果来解决。
2. Tnew<=Tuse，说明需要的数据可以及时算出，可以通过转发结果来解决。（与1有何区别？）
3. Tnew>Tuse，说明需要的数据不能及时算出，必须暂停流水线解决。

        我们发现，对于暂停，只需要把处在ID级的指令与后续指令进行比较即可判断，并且简单的处理。而转发，我们需要比较各级指令，并且在各级之间加入转发数据通路，比较复杂。因此，我们先讨论暂停机制。

1. **Tnew=0，说明结果已经算出，如果指令处于WB级，则可以通过寄存器的内部转发设计解决（关于内部转发请自行查阅资料思考或与同学讨论），不需要任何操作。如果指令不处于WB级，则可以通过转发结果来解决。**
2. Tnew<=Tuse，说明需要的数据可以及时算出，可以通过转发结果来解决。（与1有何区别？）
3. Tnew>Tuse，说明需要的数据不能及时算出，必须暂停流水线解决。

当判定需要暂停的时候，我们需要执行三个操作：

* **冻结PC，不让PC的值改变**
* **让ID/EX流水级寄存器清零，等于插入了一个NOP指令**
* **冻结IF/ID流水级寄存器，不让它的值改变。**

        如此，我们就完成了暂停机制的构建。当我们插入一个新的指令时，只需要按照表格的格式，分析Tuse,Tnew，然后就可以构造它对应的暂停判断信号。

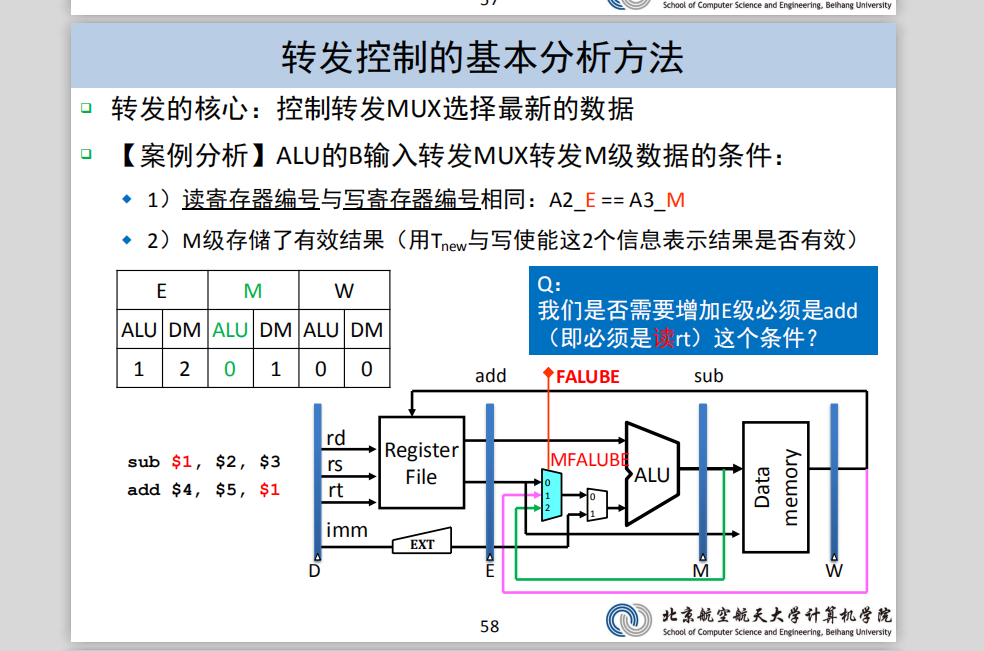
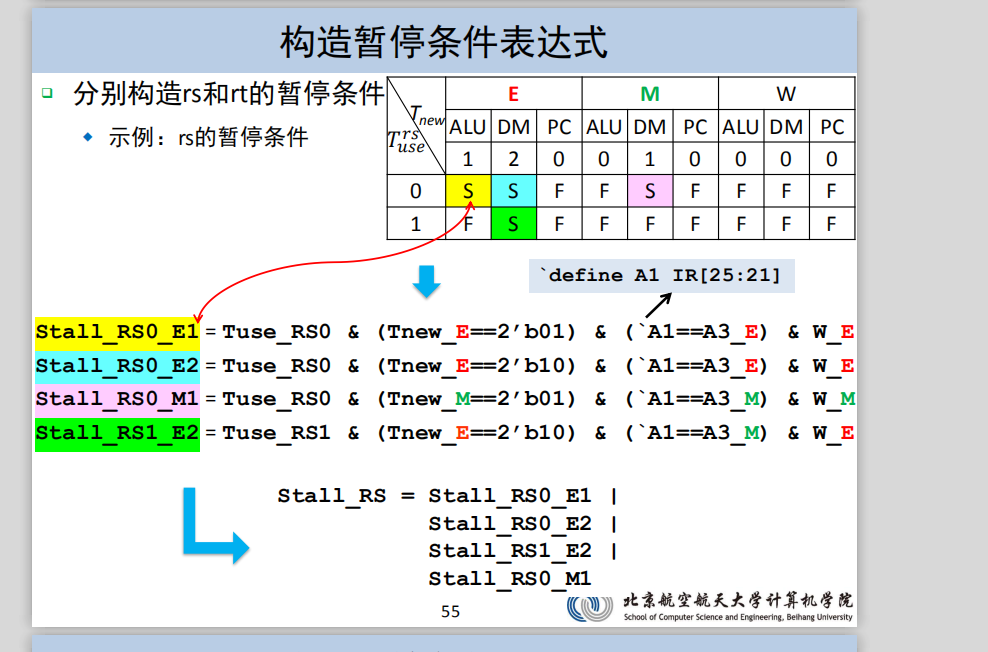
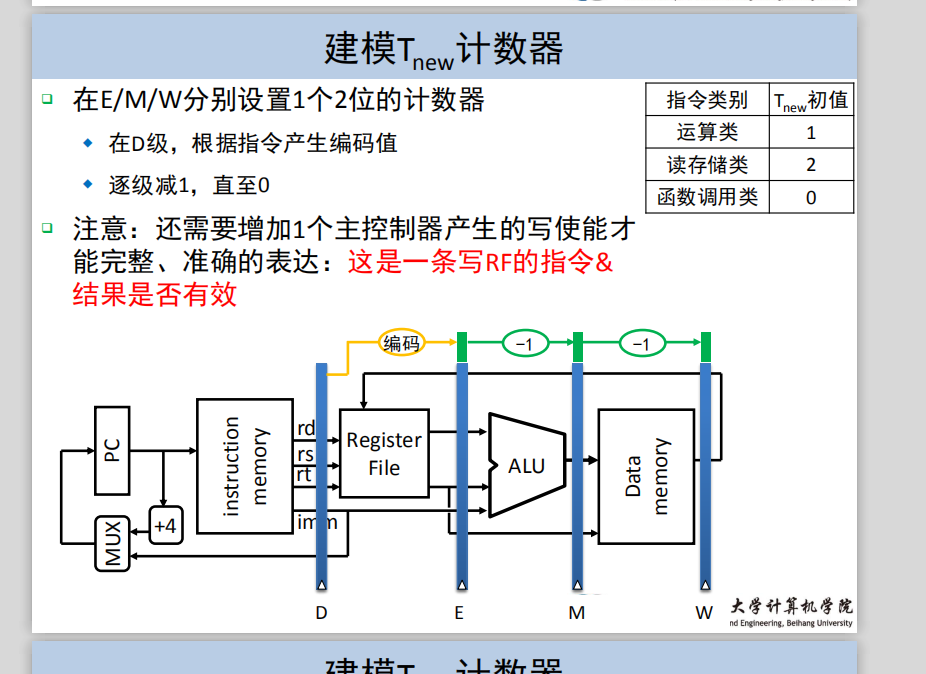
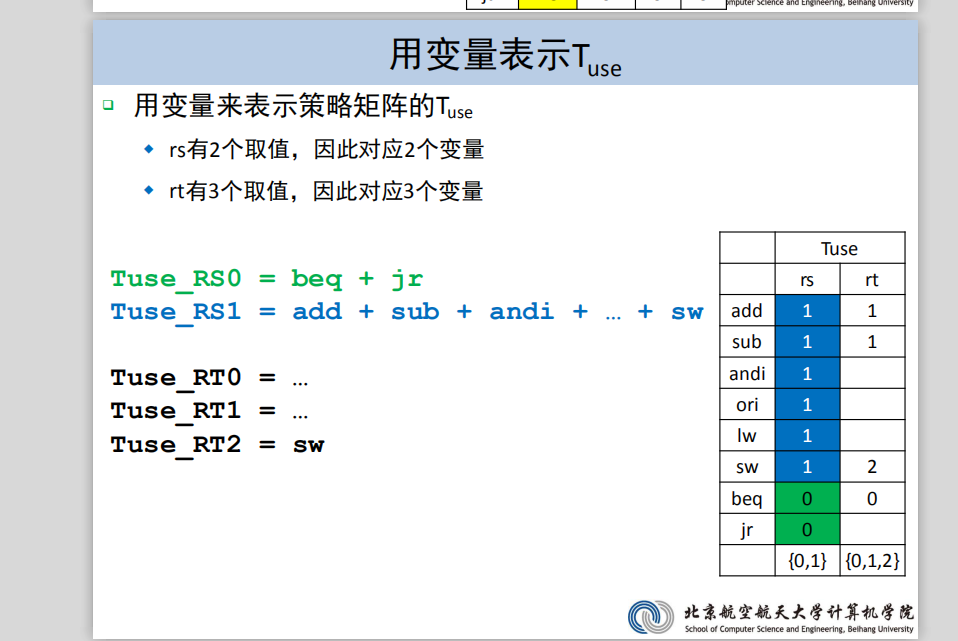
对于任何一个指令，**引用数据本质都是引用寄存器堆的值**，而通过对指令集的解析，我们知道引用寄存器的值，必然是**引用指令的rs字段或者rt字段对应的寄存器**。所以对于每个阶段，我们需求的数据其实只有两种，即需求rs寄存器，或者需求rt寄存器。而对于MEM阶段则不会再需求rs寄存器，所以对数据的需求一共只有五种情况：

**ID阶段对rs,rt寄存器的需求，**

**EX阶段对rs,rt寄存器的需求，**

**MEM阶段对rt寄存器的需求。**

只不过，指令只对应其中的一个或两个需求而已。如beq指令需求ID阶段的rs,rt寄存器，addu指令需求EX阶段的rs,rt寄存器。



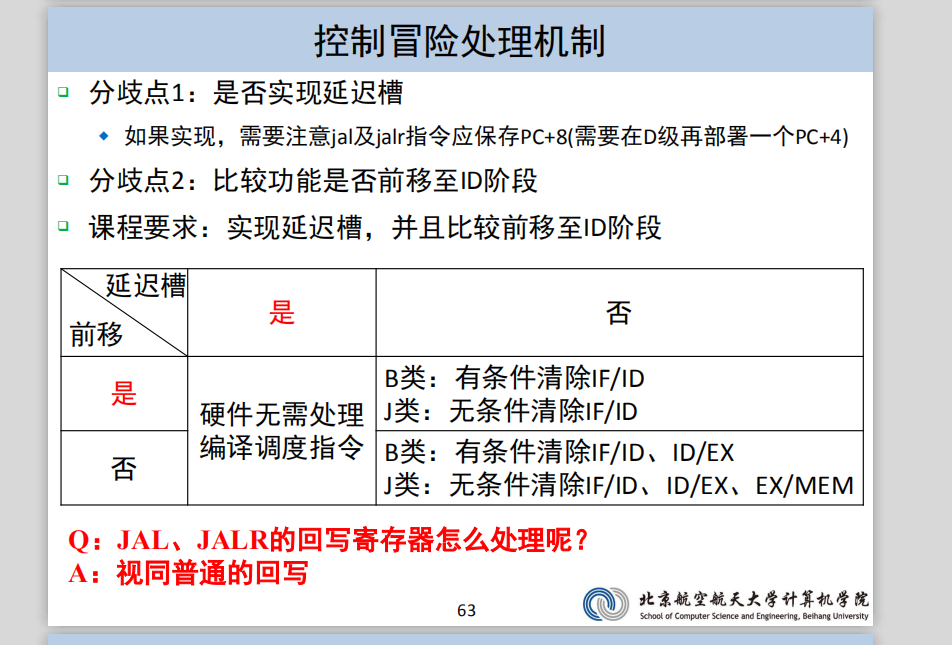
MUX的三目运算符自带优先级

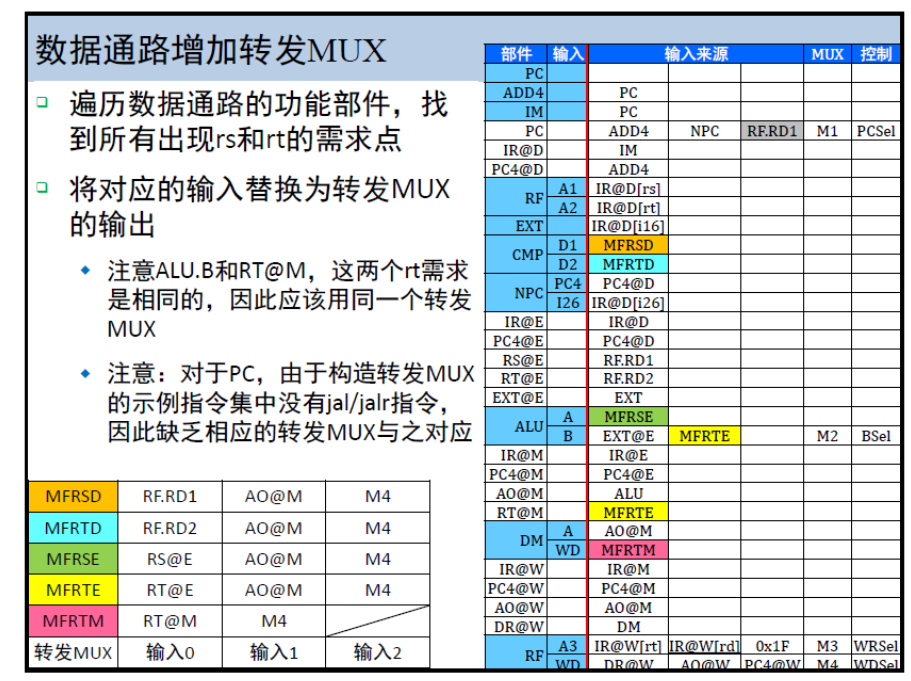
    此时如果我们需要添加指令，则首先考虑：它需不需要使用数据？如果需要，根据使用寄存器的情况，在横行里添加涉及指令。

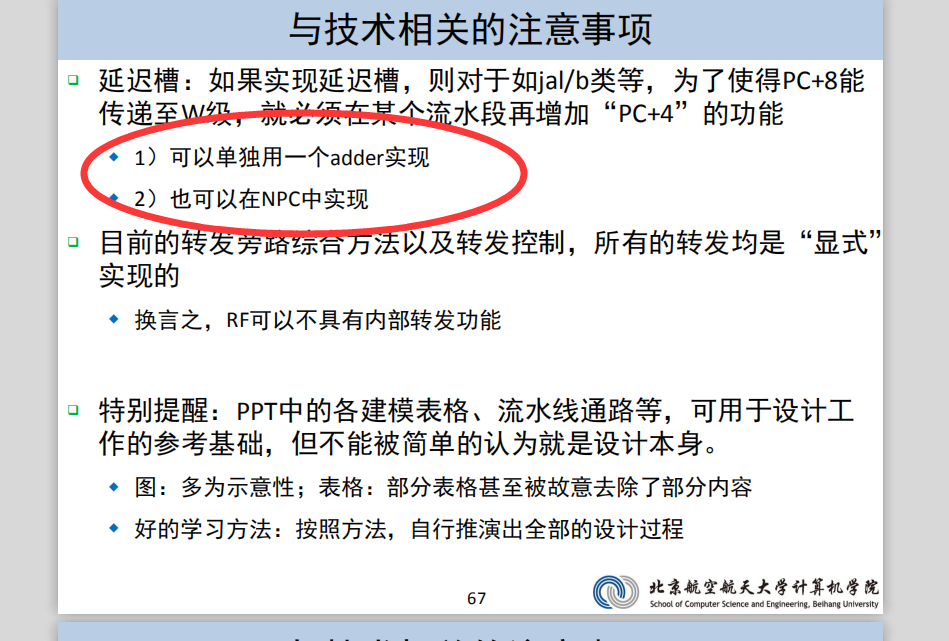
        其次，它是否提供数据结果？如果提供，则需要从它能计算出结果的那一级开始，在每一级都增加一个新的纵列（如jal，jal指令，从ID级就产生结果，则表格需要增加一个大列），同时在对应的新空格里，填写这种情况下的转发数据来源。

        在完成了转发多选器的构造之后，我们就需要把转发多选器加入数据通路中。我们只需要牢记一点：**数据通路表格中任何引用寄存器数据的地方，都要改成引用对应的转发多选器输出数据！**

        所以，我们应当根据刚刚的转发分析，修改我们之前的数据通路表格。



再设计一个PC+4，其实可以丢到NPC中搞



D级暂停，E级清空，同步清空

想一下要不要加后缀表明信号的所属

注意讨论区！！！

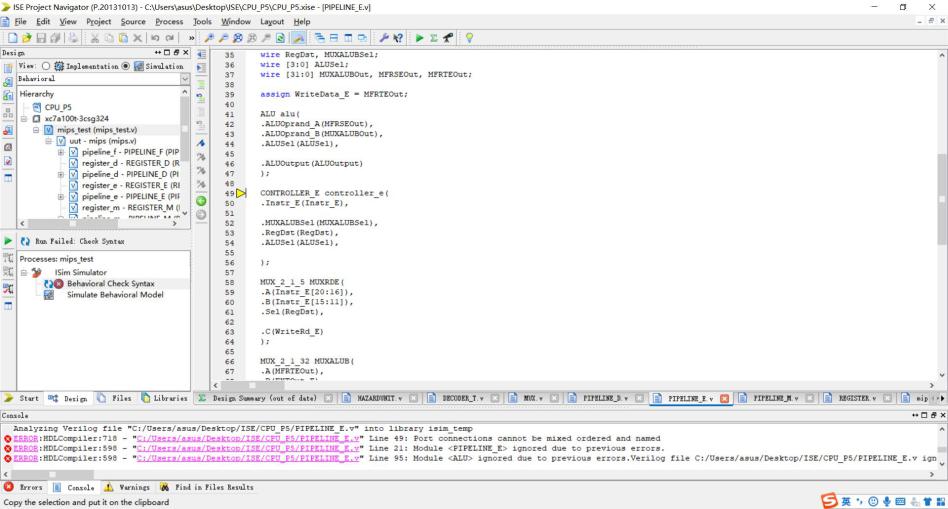
实际上每一级的RS RT可以由指令读出来

RF的rd和regwrite都是从W级传回来

命名的统一性问题！！！

还是有点乱，先写完再说

如果流水级寄存器有些东西比如RT不一开始就产生流水的话，可能会由于非阻塞赋值而产生BUG

define的东西记得加`不然不报错。。。

多加个逗号

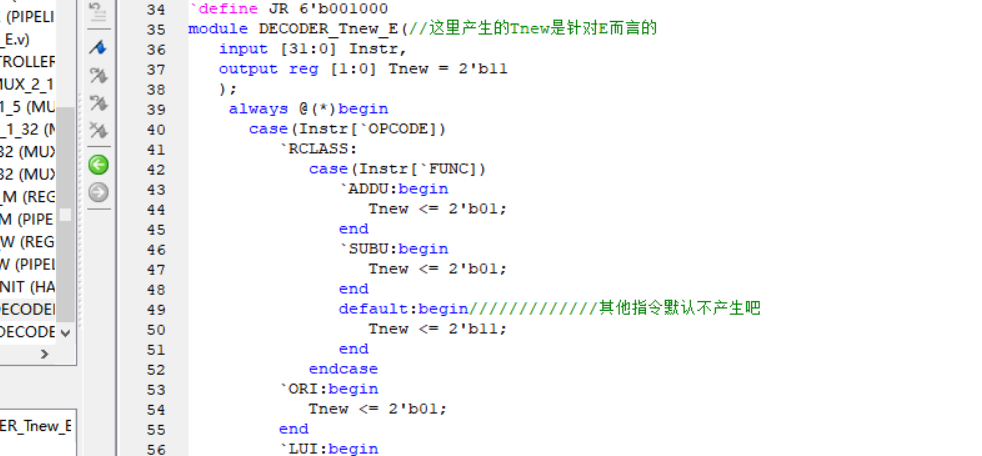
Port connections cannot be mixed ordered and named

就产生这种怀疑人生的BUG

应该输出WRITEBACK的PC！！！

还要注意ADDU等R型指令的判断





其他指令默认不转发，很重要

组合逻辑块用ALWAYS和CASE用阻塞赋值

注意Tnew -1 >= 0 是有符号还是无符号

CMP判断大小需要有符号！！！！

把JAL的结果在E级通过一个多选器选择了，然后直接转发为ALUOutput

0号寄存器不转发

最后一句是跳转指令的话必须加nop，不然D级传进去xxxx

考虑延迟槽

ALU输出只和写入寄存器的指令有关

复制粘贴记得改E D M 之类的东西，尤其是不同级别的指令要改

CMP有毛病，应该是传CMPOut，而不是传输信号！！！这样会有BUG

凡是跨过流水线寄存器传输的信号都要通过寄存器传，不然有BUG！！！

第一步，分析指令属于哪种类型，