冒险的解决

阻塞

这是最为容易的解决方法。冲突的本质是由于指令间存在相关性或依赖相同的部件，导致两条指令无法在相邻的时钟周期内相继执行。也就是说，只要把发生冲突的两条指令之间“分隔开”，就可以有效的解决冒险的问题。

阻塞是指当发生数据依赖时，只让前一条指令执行，而后一条指令被阻塞在流水线的某个阶段，并不向下执行，等待前一条指令执行完成（或者执行到没有冲突的时候），再解除后一条的阻塞状态。在我们的 CPU 中，我们将会发生冒险的指令阻塞在 D 流水级上。

在 D 级阻塞的时候，像下一流水级传递的指令不应当是 D 级指令，否则 D 级指令还是会向下一流水级传递。所以我们应当插入“指令气泡（bubble）”，也就是 nop 空指令来避免这种情况。实现 CPU 流水级的“空转”。

提前分支判断

在 P4 中，我们的分支判断是利用 ALU 实现的，也就对应我们现在的 E 级中实现。在不考虑阻塞的情况下，当我们在 E 级得到分支结果后，若跳转，则此时 F、D 级的指令都需要作废，这无疑是一种低效的方式。

如果我们将分支判断提前到 D 级，那么即使发生跳转，需要作废的指令只有 F 级。其本质是尽早产生结果可以缩短因不确定而带来的开销。

延迟槽

在前面我们提到即使将分支判断提到 D 级，发生跳转的时候，F 级指令依然是需要作废的。但是我们如果约定 F 级指令不作废呢？也就是说，不论判断结果如何，我们都将执行分支或跳转指令的下一条指令。这也就是所谓的“**延迟槽**”。那么指令执行的效率就提高了。

再次强调，延迟槽是基本上不需要实现的，也就是说，只要不考虑 F 级指令的作废问题，就是实现了延迟槽。唯一需要变更的是，**对于 jal 指令，应当向 31 号寄存器写入 D_PC + 8 或者 F_PC + 4（当 jal 指令在 D 级时）。**

延迟槽中的指令到底是什么，是由编译器决定的（一个把高级语言翻译成汇编语言的软件），并不需要 CPU 的设计者（也就是我们）操心。编译器保证**延迟槽中的指令不能为分支或跳转指令。**

转发

虽然阻塞可以解决全部冒险问题（阻塞的极限情况就是单周期 CPU），但是这无疑会降低 CPU 的并行度，使 CPU 的吞吐量减少。阻塞的本质是说，因为后一条指令需要前一条指令的执行结果，只有让后一条指令等到前一条指令写入寄存器，才可以继续执行。

但是我们考虑数据并非一定要等到写入寄存器堆中才可以被使用，以 add 指令为例，其结果在 E 级时就已经计算完成。那么就没必要再让后面被阻塞的指令多等一个周期，我们现在可以在 M 级就把这个结果传递回去，让 D 级指令解除阻塞状态，这样提高了 CPU 的并行度。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，这个过程就叫做**转发**，它可以用于处理**部分的数据冒险**情况。

A 模型

A 模型描述的是一个很显然的事情，就是需求者和供给者转发的数据必须是同一个寄存器的值。如果需求者需要 5 号寄存器中的值，而供给者只能提供 8 号寄存器中的值，那么显然转发是不能发生的。

我们的 A 指 Address，也就是**寄存器的地址**（编号）。在转发的时候需要检验需求者和供给者的对应的 A 值是否相等，且不为 0。

T 模型

可以看到，转发需要考虑的因素有很多，稍有不慎就会犯下错误。所以我们需要一个**数学模型**来描述转发，即**需求时间-供给时间模型。**

对于某一个指令的某一个数据需求，我们定义需求时间 T_{use} 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 $T_{use}=0$。对于 add 指令，等待下一个时钟周期它进入 E 级才要使用数据，所以 $T_{use}=1$。而对于 sw 指令，在 E 级它需要 GPR[rs] 的数据来计算地址，在 M 级需要 GPR[rt] 来存入值，所以 $rs\_T_{use}=1$，$rt\_T_{use}=2$。

我们可以归纳出 T_{use} 的两条性质：

- T_{use} 是一个定值，每个指令的 T_{use} 是一定的；
- 一个指令可以有两个 T_{use} 值。

对于某个指令的数据产出，我们定义供给时间 T_{new} 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 add 指令，当它处于 E 级，此时结果还没有存储到流水级寄存器里，所以此时它的 $T_{new}=1$，而当它处于 M 或者 W 级，此时结果已经写入了流水级寄存器，所以此时 $T_{new}=0$。

我们可以归纳出 T_{new} 的两条性质：

- T_{new} 是一个动态值，每个指令处于流水线不同阶段有不同的 T_{new} 值；
- 一个指令在一个时刻至多有一个 T_{new} 值（一个指令至多写一个寄存器）。

在阐述完所有的数学概念后，我们就可以利用这些概念来描述转发的条件：

- 当 $T_{use} \geq T_{new}$，说明需要的数据可以及时算出，可以通过**转发**来解决。
- 当 $T_{use} < T_{new}$，说明需要的数据不能及时算出，必须**阻塞**流水线解决。

至此，我们的 T 模型就已经介绍完毕了。A 模型和 T 模型结合就形成了 **AT 法**的理论基础。

思考题

我们使用提前分支判断的方法尽早产生结果来减少因不确定而带来的开销，但实际上这种方法并非总能提高效率，请从流水线冒险的角度思考其原因并给出一个指令序列的例子。

思考题

因为延迟槽的存在，对于 jal 等需要将指令地址写入寄存器的指令，要写回 PC + 8，请思考为什么这样设计？

思考题

我们要求大家所有转发数据都来源于流水寄存器而不能是功能部件（如 DM、ALU），请思考为什么？

思考题

[P5 选做] 在冒险的解决中，我们引入了 AT 法，如果你有其他的解决方案，请简述你的思路，并给出一段指令序列，简单说明你是如何做到尽力转发的。