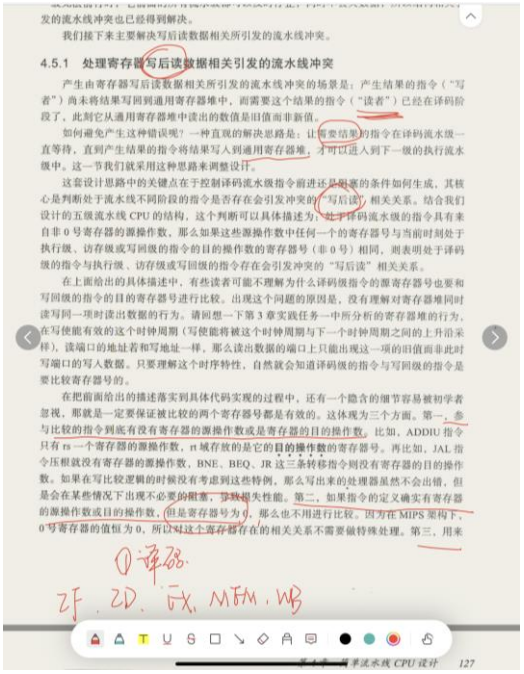
2022.11.28

李婷玉：

学习了 CPU 设计实战 4.1、4.5、4.6 相关知识。学习了如何设计一个简单的单周期 CPU（包含 19 条 MIPS 指令以及控制信号的生成、复位处理）以及在实际情况中如何利用阻塞和前递技术解决 RAW 的数据相关问题。前递技术会增加 CPU 关键路径的延迟，导致 CPU 主频下降。所以在实际情况中，如果只是在有些情况下将转移指令阻塞一拍。考虑到如果运行的程序中这种情况并不是频繁出现，那么程序在流水线上的整体执行效率并不会下降太多，同时由于 CPU 的主频得到提升，会使整体性能得到提升。



4.5.1 处理寄存器写后读数据相关引发的流水线冲突

产生由寄存器写后读数据相关所引发的流水线冲突的场景是：产生结果的指令（“写者”）尚未将结果写回到通用寄存器堆中，而需要这个结果的指令（“读者”）已经在译码阶段了，此刻它从通用寄存器堆中读出的数值是旧值而非新值。

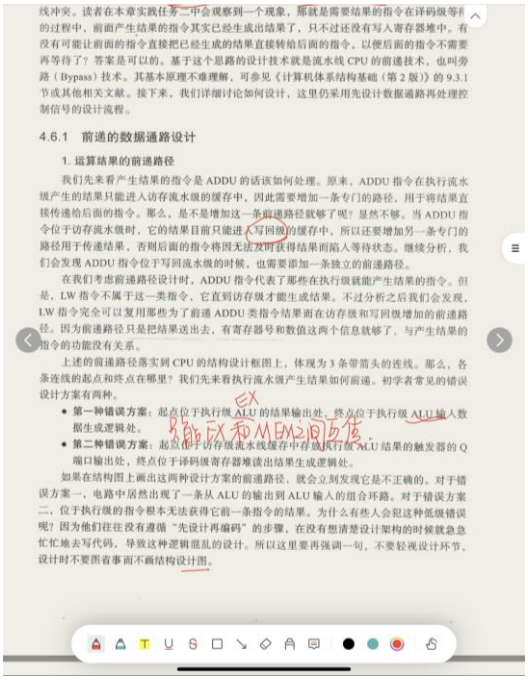
如何避免产生这种错误呢？一种直观的解决思路是：让需要结果的指令在译码流水级一直等待，直到产生结果的指令将结果写入到通用寄存器堆，才可以进入到下一级的执行流水级中。这一节我们就采用这种思路来调整设计。

这套设计思路中的关键点在于控制译码流水级指令前进还是阻塞的条件如何生成，其核心是判断处于流水线不同阶段的指令是否存在会引发冲突的“写后读”相关关系。结合我们设计的五级流水线 CPU 的结构，这个判断可以具体描述为：处于译码流水级的指令具有来自非 0 号寄存器的源操作数，那么如果这些源操作数中任何一个的寄存器号与当前时刻处于执行级、访存级或写回级的指令的目的操作数的寄存器号（非 0 号）相同，则表明处于译码级的指令与执行级、访存级或写回级的指令存在会引发冲突的“写后读”相关关系。

在上面给出的具体描述中，有些读者可能不理解为什么译码级指令的源寄存器号也要和写回级的指令的目的寄存器号进行比较。出现这个问题的原因是，没有理解对寄存器堆同时读写同一项时读出数据的行为。请回想一下第 3 章实践任务一中所分析的寄存器堆的行为，在写使能有效的这个时钟周期（写使能将被这个时钟周期与下一个时钟周期之间的上升沿采样），读端口的地址若和写地址一样，那么读出数据的端口上只能出现这一项的旧值而非此时写端口的写入数据。只要理解这个时序特性，自然就会知道译码级的指令与写回级的指令是要比较寄存器号的。

在把前面给出的描述落实到具体代码实现的过程中，还有一个隐含的细节容易被初学者忽视，那就是一定要保证被比较的两个寄存器号都是有效的。这体现为三个方面。第一，参与比较的指令到底有没有寄存器的源操作数或是寄存器的目的操作数。比如，ADDIU 指令只有 rs 一个寄存器的源操作数，rt 域存放的是它的目的操作数的寄存器号。再比如，JAL 指令压根就没有寄存器的源操作数，BNE、BEQ、JR 这三条转移指令则没有寄存器的目的操作数。如果在写比较逻辑的时候没有考虑到这些特例，那么写出来的处理器虽然不会出错，但是会在某些情况下出现不必要的阻塞，导致损失性能。第二，如果指令的定义确实有寄存器的源操作数或目的操作数，但是寄存器号为 0，那么也不用进行比较。因为在 MIPS 架构下，0 号寄存器的值恒为 0，所以对这个寄存器存在的相关关系不需要做特殊处理。第三，用来

图 1 4.5.1 数据相关处理



4.6.1 前递的数据通路设计

1. 运算结果的前递路径

我们先来看产生结果的指令是 ADDU 的话该如何处理。原来，ADDU 指令在执行流水级产生的结果只能进入访存流水级的缓存中，因此需要增加一条专门的路径，用于将结果直接传递给后面的指令。那么，是不是增加这一条前递路径就够了呢？显然不够。当 ADDU 指令位于访存流水级时，它的结果目前只能进入写回级的缓存中，所以还要增加另一条专门的路径用于传递结果，否则后面的指令将因无法及时获得结果而陷入等待状态。继续分析，我们会发现 ADDU 指令位于写回流水级的时候，也需要添加一条独立的前递路径。

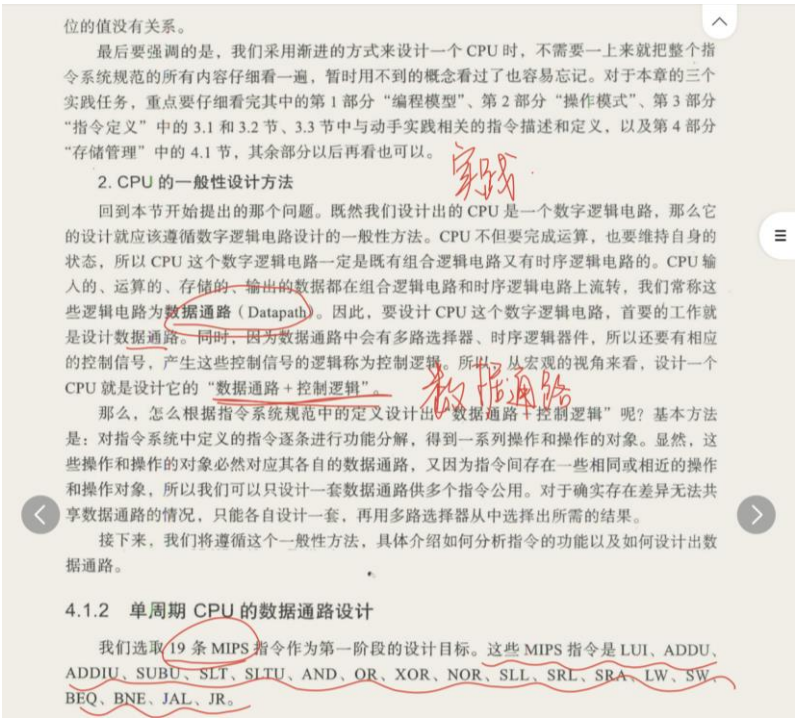
在我们考虑前递路径设计时，ADDU 指令代表了那些在执行级就能产生结果的指令。但是，LW 指令不属于这一类指令，它直到访存级才能生成结果。不过分析之后我们会发现，LW 指令完全可以复用那些为了前递 ADDU 类指令结果而在访存级和写回级增加的前递路径。因为前递路径只是把结果送出去，有寄存器号和数值这两个信息就够了，与产生结果的指令的功能没有关系。

上述的前递路径落实到 CPU 的结构设计框图上，体现为 3 条带箭头的连线。那么，各条连线的起点和终点在哪里？我们先来看执行流水级产生结果如何前递。初学者常见的错误设计方案有两种。

- **第一种错误方案**：起点位于执行级 ALU 的结果输出处，终点位于执行级 ALU 输入数据生成逻辑处。
- **第二种错误方案**：起点位于访存级流水线缓存中存放执行级 ALU 结果的触发器的 Q 端口输出处，终点位于译码级寄存器堆读出结果生成逻辑处。

如果在结构图上画出这两种设计方案的前递路径，就会立刻发现它是不正确的。对于错误方案一，电路中居然出现了一条从 ALU 的输出到 ALU 输入的组合环路。对于错误方案二，位于执行级的指令根本无法获得它前一条指令的结果。为什么有些人会犯这种低级错误呢？因为他们往往没有遵循“先设计再编码”的步骤，在没有想清楚设计架构的时候就急急忙忙地去写代码，导致这种逻辑混乱的设计。所以这里要再强调一句，不要轻视设计环节，设计时不要图省事而不画结构设计图。

图 2 前递的数据通路设计



位的值没有关系。

最后要强调的是，我们采用渐进的方式来设计一个 CPU 时，不需要一上来就把整个指令系统规范的所有内容仔细看一遍，暂时用不到的概念看过了也容易忘记。对于本章的三个实践任务，重点要仔细看完其中的第 1 部分“编程模型”、第 2 部分“操作模式”、第 3 部分“指令定义”中的 3.1 和 3.2 节、3.3 节中与动手实践相关的指令描述和定义，以及第 4 部分“存储管理”中的 4.1 节，其余部分以后再看也可以。

2. CPU 的一般性设计方法

回到本节开始提出的那个问题。既然我们设计出的 CPU 是一个数字逻辑电路，那么它的设计就应该遵循数字逻辑电路设计的一般性方法。CPU 不但要完成运算，也要维持自身的状态，所以 CPU 这个数字逻辑电路一定是既有组合逻辑电路又有时序逻辑电路的。CPU 输入的、运算的、存储的、输出的数据都在组合逻辑电路和时序逻辑电路上流转，我们常称这些逻辑电路为数据通路（Datapath）。因此，要设计 CPU 这个数字逻辑电路，首要的工作就是设计数据通路。同时，因为数据通路中会有多路选择器、时序逻辑器件，所以还要有相应的控制信号，产生这些控制信号的逻辑称为控制逻辑。所以，从宏观的视角来看，设计一个 CPU 就是设计它的“数据通路 + 控制逻辑”。

那么，怎么根据指令系统规范中的定义设计出“数据通路 + 控制逻辑”呢？基本方法是：对指令系统中定义的指令逐条进行功能分解，得到一系列操作和操作的对象。显然，这些操作和操作的对象必然对应其各自的数据通路，又因为指令间存在一些相同或相近的操作和操作对象，所以我们可以只设计一套数据通路供多个指令公用。对于确实存在差异无法共享数据通路的情况，只能各自设计一套，再用多路选择器从中选择出所需的结果。

接下来，我们将遵循这个一般性方法，具体介绍如何分析指令的功能以及如何设计出数据通路。

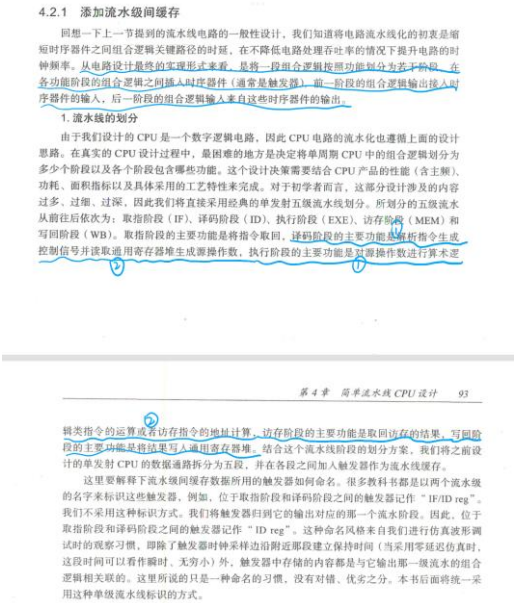
4.1.2 单周期 CPU 的数据通路设计

我们选取 19 条 MIPS 指令作为第一阶段的设计目标。这些 MIPS 指令是 LUI、ADDU、ADDIU、SUBU、SLT、SLTU、AND、OR、XOR、NOR、SLL、SRL、SRA、LW、SW、BEQ、BNE、JAL、JR。

图 3 4.1CPU 一般设计

王小可：

CPU 设计实战 4.1 节设计单周期的 CPU，单周期 CPU 的数据通路设计（其中涉及到的 MIPS 指令主要有 LUI、STL、SLTU、AND、OR、XOR、NOR、SLL、SRL、SRA），了解 ALU 模块的两个 32 为输入 alu_src1 和 alu_src2 以及一个 32 位输出 alu_res 和控制信号输入 alu_op。译码阶段的主要功能为解析指令生成控制信号并读取通用寄存器堆生成源操作数；执行阶段主要功能是对源操作数进行算术逻辑的运算或访存指令的地址计算；访存阶段主要是取回访存的结果；写回阶段将结果写入通用寄存器堆。



4.2.1 添加流水级间缓存

回想一下上一节提到的流水线电路的一般性设计，我们知道将电路流水线化的初衷是缩短时序器件之间组合逻辑关键路径的时延，在不降低电路处理吞吐率的情况下提升电路的时钟频率。从电路设计最终的实现形式来看，是将一段组合逻辑按照功能划分为若干阶段，在各功能阶段的组合逻辑之间插入时序器件（通常是触发器），前一阶段的组合逻辑输出接入时序器件的输入，后一阶段的组合逻辑输入来自这些时序器件的输出。

1. 流水线的划分

由于我们设计的 CPU 是一个数字逻辑电路，因此 CPU 电路的流水化也遵循上面的设计思路。在真实的 CPU 设计过程中，最困难的地方是决定将单周期 CPU 中的组合逻辑划分为多少个阶段以及各个阶段包含哪些功能。这个设计决策需要结合 CPU 产品的性能（含主频）、功耗、面积指标以及具体采用的工艺特性来完成。对于初学者而言，这部分设计涉及的内容过多、过细、过深，因此我们将直接采用经典的单发射五级流水线划分。所划分的五级流水从前往后依次为：取指阶段（IF）、译码阶段（ID）、执行阶段（EXE）、访存阶段（MEM）和写回阶段（WB）。取指阶段的主要功能是将指令取回，译码阶段的主要功能是解析指令生成控制信号并读取通用寄存器堆生成源操作数，执行阶段的主要功能是对源操作数进行算术逻

第 4 章 简单流水线 CPU 设计 93

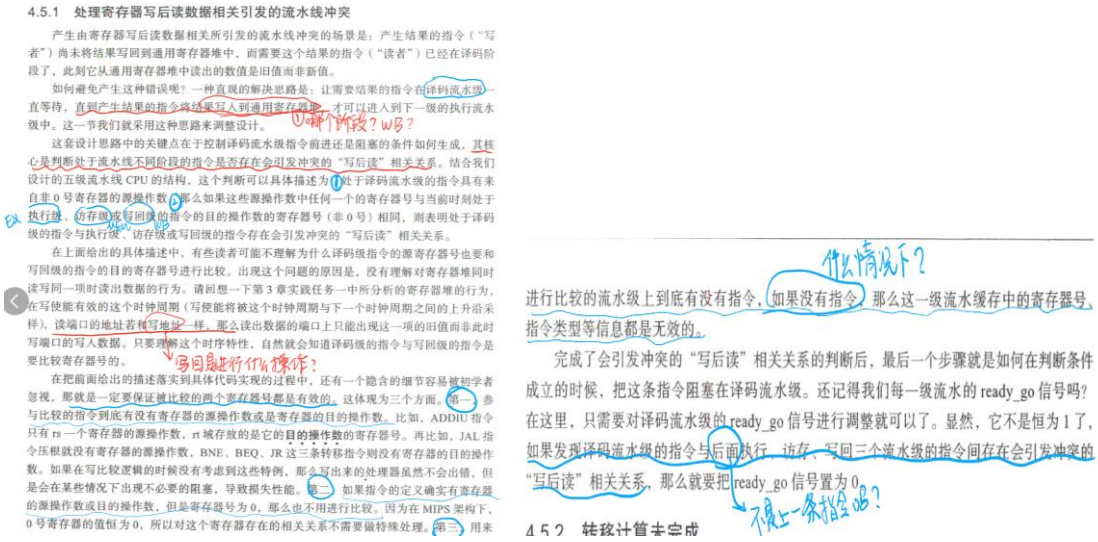
辑类指令的运算或者访存指令的地址计算，访存阶段的主要功能是取回访存的结果，写回阶段的主要功能是将结果写入通用寄存器堆。结合这个流水线阶段的划分方案，我们将之前设计的单发射 CPU 的数据通路拆分为五段，并在各段之间加入触发器作为流水线缓存。

这里要解释下流水级间缓存数据所用的触发器如何命名。很多教科书都是以两个流水级的名字来标识这些触发器，例如，位于取指阶段和译码阶段之间的触发器记作“IF/ID reg”。我们不采用这种标识方式。我们将触发器归到它的输出对应的那一个流水阶段。因此，位于取指阶段和译码阶段之间的触发器记作“ID reg”。这种命名风格来自我们进行仿真波形调试时的观察习惯，即除了触发器时钟采样边沿附近段建立保持时间（当采用零延迟仿真时，这段时间可以看作瞬时、无穷小）外，触发器中存储的内容都是与它输出那一级流水的组合逻辑相关联的。这里所说的只是一种命名的习惯，没有对错、优劣之分。本书后面将统一采用这种单级流水线标识的方式。

图 4 4.2.1 添加流水线间缓存

在处理寄存器写后读相关主要学习如何实现插入气泡法处理冲突。在实现过程中需要注意

的三个方面主要是什么，同时在学习过程中随时记录遇见的问题。



4.5.1 处理寄存器写后读数据相关引发的流水线冲突

产生由寄存器写后读数据相关所引发的流水线冲突的场景是：产生结果的指令（“写者”）尚未将结果写回到通用寄存器堆中，而需要这个结果的指令（“读者”）已经在译码阶段了，此刻它从通用寄存器堆中读出的数值是旧值而非新值。

如何避免产生这种错误呢？一种直观的解决思路是：让需要结果的指令在译码流水级一直等待，直到产生结果的指令将结果写入到通用寄存器堆，才可以进入到下一级的执行流水级中。这一节我们就采用这种思路来调整设计。

这套设计思路中的关键点在于控制译码流水级指令前进还是阻塞的条件如何生成，其核心是判断处于流水线不同阶段的指令是否存在会引发冲突的“写后读”相关关系。结合我们设计的五级流水线 CPU 的结构，这个判断可以具体描述为：处于译码流水级的指令具有来自非 0 号寄存器的源操作数，那么如果这些源操作数中任何一个的寄存器号与当前时刻处于执行级、访存级或写回级的指令的目的操作数的寄存器号（非 0 号）相同，则表明处于译码级的指令与执行级、访存级或写回级的指令存在会引发冲突的“写后读”相关关系。

在上面给出的具体描述中，有些读者可能不理解为什么译码级指令的源寄存器号也要和写回级的指令的目的寄存器号进行比较。出现这个问题的原因是，没有理解对寄存器堆同时读写同一项时读出数据的行为。请回想一下第 3 章实践任务一中所分析的寄存器堆的行为，在写使能有效的这个时钟周期（写使能将被这个时钟周期与下一个时钟周期之间的上升沿采样），读端口的地址若和写地址一样，那么读出数据的端口上只能出现这一项的旧值而非此时写端口的写入数据。只要理解这个时序特性，自然就会知道译码级的指令与写回级的指令是要比较寄存器号的。

在把前面给出的描述落实到具体代码实现的过程中，还有一个隐含的细节容易被初学者忽视，那就是一定要保证被比较的两个寄存器号都是有效的。这体现为三个方面。第一，参与比较的指令到底有没有寄存器的源操作数或是寄存器的目的操作数。比如，ADDIU 指令只有 rs 一个寄存器的源操作数，rt 域存放的是它的目的操作数的寄存器号。再比如，JAL 指令压根就没有寄存器的源操作数，BNE、BEQ、JR 这三条转移指令则没有寄存器的目的操作数。如果在写比较逻辑的时候没有考虑到这些特例，那么写出来的处理器虽然不会出错，但是会在某些情况下出现不必要的阻塞，导致损失性能。第二，如果指令的定义确实有寄存器的源操作数或目的操作数，但是寄存器号为 0，那么也不用进行比较。因为在 MIPS 架构下，0 号寄存器的值恒为 0，所以对这个寄存器存在的相关关系不需要做特殊处理。第三，用来

进行比较的流水级上到底有没有指令，如果没有指令，那么这一级流水缓存中的寄存器号、指令类型等信息都是无效的。

完成了会引发冲突的“写后读”相关关系的判断后，最后一个步骤就是如何在判断条件成立的时候，把这条指令阻塞在译码流水级。还记得我们每一级流水的 ready_go 信号吗？在这里，只需要对译码流水级的 ready_go 信号进行调整就可以了。显然，它不是恒为 1 了，如果发现译码流水级的指令与后面执行、访存、写回三个流水级的指令间存在会引发冲突的“写后读”相关关系，那么就要把 ready_go 信号置为 0。

4.5.2 转移计算与完成

图 5 4.5.1 处理写后读数据相关引发的冲突

雷传澳：学习了 CPU 设计实战 4.5、4.6，学习 id.v 代码