

## 20-面向流水线的指令设计（上）：一心多用的现代CPU

前面我们用了三讲，用一个个的电路组合，制作出了一个完整功能的CPU。这里面一下子给你引入了三个“周期”的概念，分别是指令周期、机器周期（或者CPU周期）以及时钟周期。

你可能会觉得有点摸不着头脑了，为什么小小一个CPU，有那么多的周期（Cycle）呢？我们在专栏一开始，不是把CPU的性能定义得非常清楚了吗？我们说程序的性能，是由三个因素相乘来衡量的，我们还专门说过“指令数×CPI×时钟周期”这个公式。这里面和周期相关的只有一个时钟周期，也就是我们CPU的主频倒数。当时讲的时候我们说，一个CPU的时钟周期可以认为是可以完成一条最简单的计算机指令的时间。

那么，为什么我们在构造CPU的时候，一下子出来了那么多个周期呢？这一讲，我就来为你说道说道，带你更深入地看看现代CPU是怎么一回事儿。

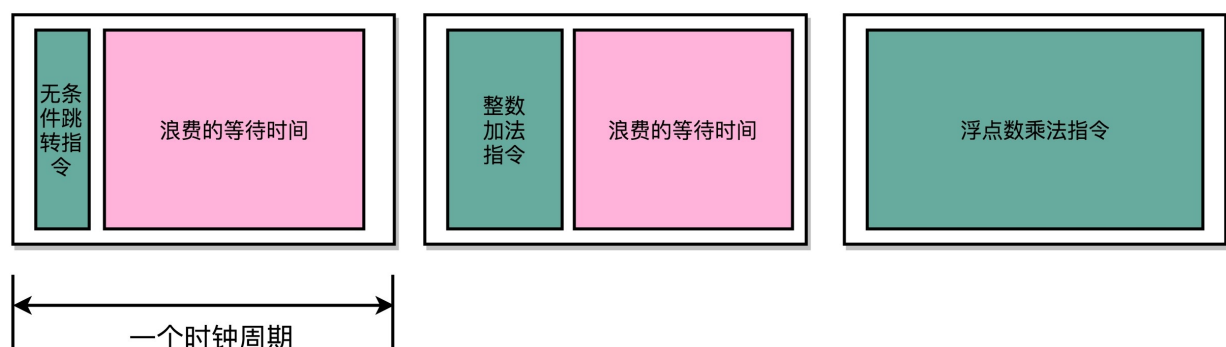
### 愿得一心人，白首不相离：单指令周期处理器

学过前面三讲，你现在应该知道，一条CPU指令的执行，是由“取得指令（Fetch）-指令译码（Decode）-执行指令（Execute）”这样三个步骤组成的。这个执行过程，至少需要花费一个时钟周期。因为在取指令的时候，我们需要通过时钟周期的信号，来决定计数器的自增。

那么，很自然地，我们希望能确保让这样一整条指令的执行，在一个时钟周期内完成。这样，我们一个时钟周期可以执行一条指令，CPI也就是1，看起来就比执行一条指令需要多个时钟周期性能要好。采用这种设计思路的处理器，就叫作单指令周期处理器（Single Cycle Processor），也就是在一个时钟周期内，处理器正好能处理一条指令。

不过，我们的时钟周期是固定的，但是指令的电路复杂程度是不同的，所以实际一条指令执行的时间是不同的。在[第13讲](#)和[第14讲](#)讲加法器和乘法器电路的时候，我给你看过，随着门电路层数的增加，由于门延迟的存在，位数多、计算复杂的指令需要的执行时间会更长。

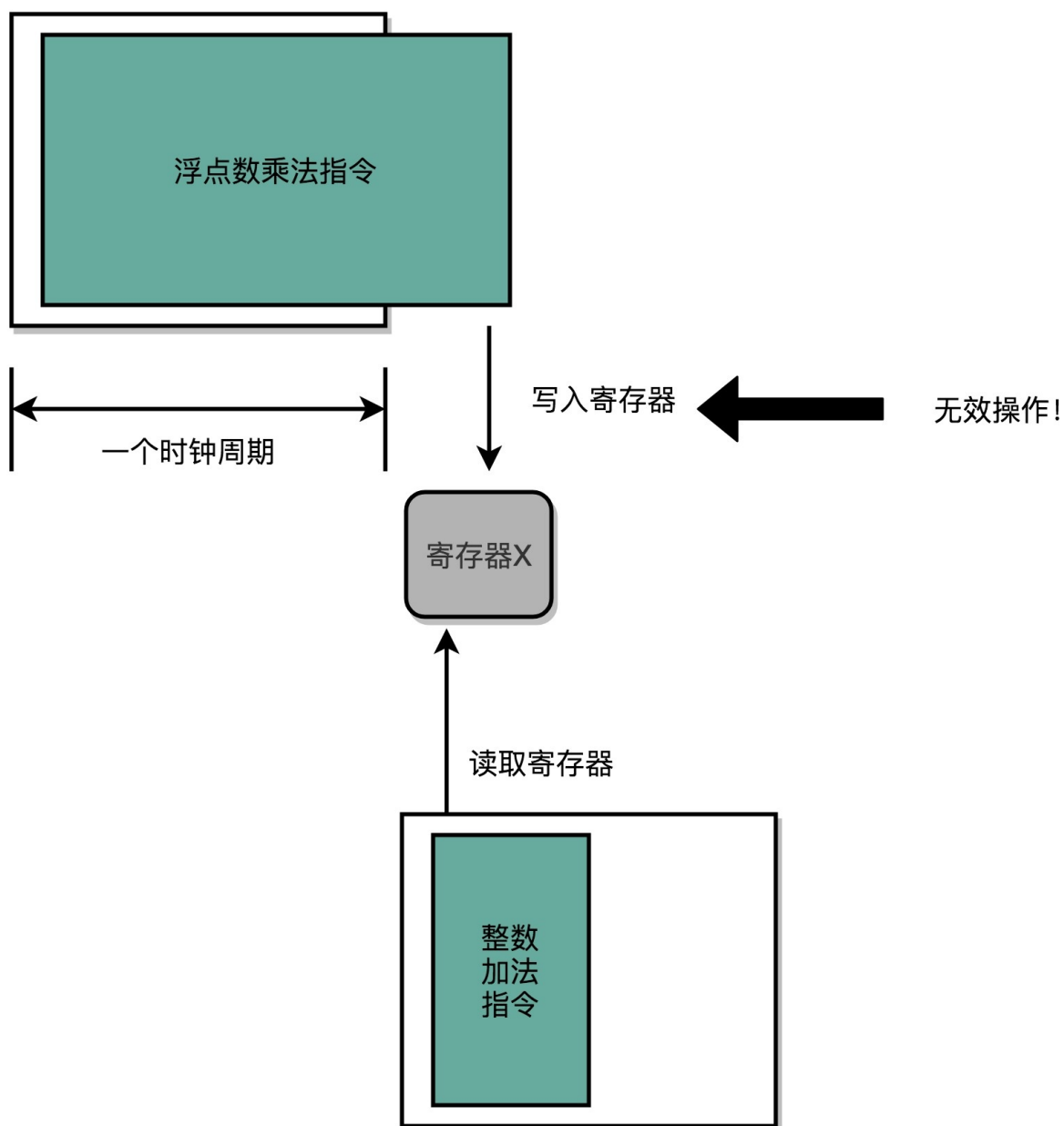
不同指令的执行时间不同，但是我们需要让所有指令都在一个时钟周期内完成，那就只好把时钟周期和执行时间最长的那个指令设成一样。这就好比学校体育课1000米考试，我们要给这场考试预留的时间，肯定得和跑得最慢的那个同学一样。因为就算其他同学先跑完，也要等最慢的同学跑完，我们才能进行下一项活动。



快速执行完成的指令，需要等待满一个时钟周期，才能执行下一条指令

所以，在单指令周期处理器里面，无论是执行一条用不到ALU的无条件跳转指令，还是一条计算起来电路特别复杂的浮点数乘法运算，我们都得等满一个时钟周期。在这个情况下，虽然CPI能够保持在1，但是我们的时钟频率却没法太高。因为太高的话，有些复杂指令没有办法在一个时钟周期内运行完成。那么在下一

一个时钟周期到来，开始执行下一条指令的时候，前一条指令的执行结果可能还没有写入到寄存器里面。那下一条指令读取的数据就是不准确的，就会出现错误。



前一条指令的写入，在后一条指令的读取之前

到这里你会发现，这和我们之前[第3讲](#)和[第4讲](#)讲时钟频率时候的说法不太一样。当时我们说，一个CPU时钟周期，可以认为是完成一条简单指令的时间。为什么到了这里，单指令周期处理器，反而变成了执行一条最复杂的指令的时间呢？

这是因为，无论是PC上使用的Intel CPU，还是手机上使用的ARM CPU，都不是单指令周期处理器，而是采用了一种叫作**指令流水线**（Instruction Pipeline）的技术。

## 无可奈何花落去，似曾相识燕归来：现代处理器的流水线设计

其实，CPU执行一条指令的过程和我们开发软件功能的过程很像。

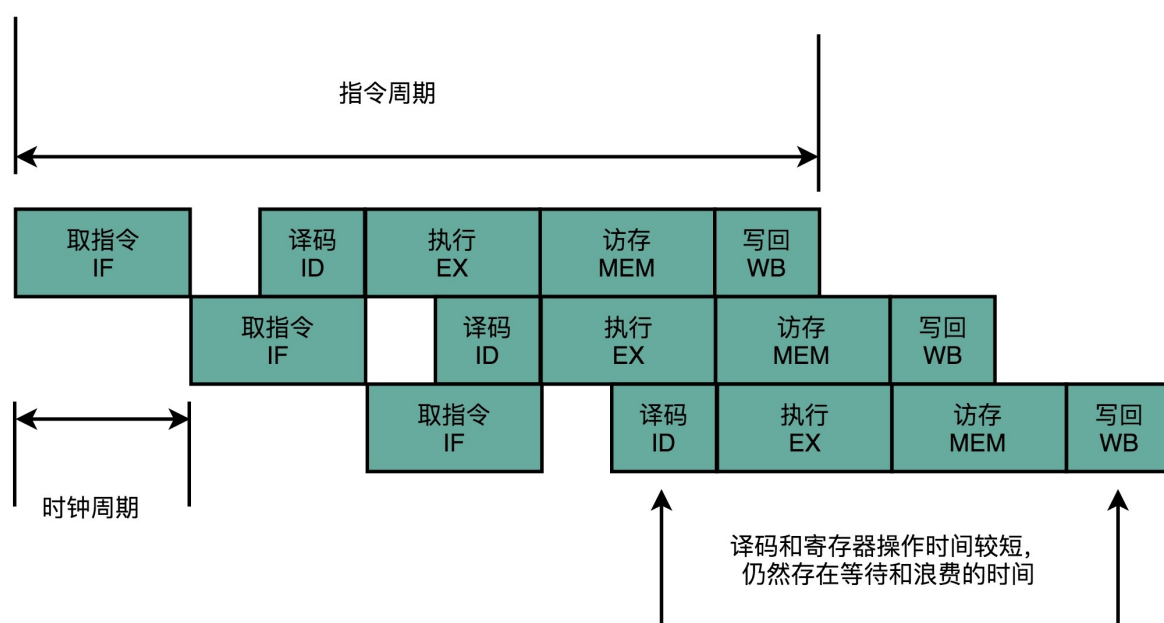
如果我们想开发一个手机App上的功能，并不是找来一个工程师，告诉他“你把这个功能开发出来”，然后他就吭哧吭哧把功能开发出来。真实的情况是，无论只有一个工程师，还是有一个开发团队，我们都需要先

对开发功能的过程进行切分，把这个过程变成“撰写需求文档、开发后台API、开发客户端App、测试、发布上线”这样多个独立的过程。每一个后面的步骤，都要依赖前面的步骤。

我们的指令执行过程也是一样的，它会拆分成“取指令、译码、执行”这样三大步骤。更细分一点的话，执行的过程，其实还包含从寄存器或者内存中读取数据，通过ALU进行运算，把结果写回到寄存器或者内存中。

如果我们有一个开发团队，我们不会让后端工程师开发完API之后，就歇着等待前台App的开发、测试乃至发布，而是会在客户端App开发的同时，着手下一个需求的后端API开发。那么，同样的思路我们可以一样应用在CPU执行指令的过程中。

通过过去三讲，你应该已经知道了，CPU的指令执行过程，其实也是由各个电路模块组成的。我们在取指令的时候，需要一个译码器把数据从内存里面取出来，写入到寄存器中；在指令译码的时候，我们需要另外一个译码器，把指令解析成对应的控制信号、内存地址和数据；到了指令执行的时候，我们需要的则是一个完成计算工作的ALU。这些都是一个一个独立的组合逻辑电路，我们可以把它们看作一个团队里面的产品经理、后端工程师和客户端工程师，共同协作来完成任务。



流水线执行示意图

这样一来，我们就不用把时钟周期设置成整条指令执行的时间，而是拆分成完成这样的一个小步骤需要的时间。同时，每一个阶段的电路在完成对应的任务之后，也不需要等待整个指令执行完成，而是可以直接执行下一条指令的对应阶段。

这就好像我们的后端程序员不需要等待功能上线，就会从产品经理手中拿到下一个需求，开始开发API。这样的协作模式，就是我们所说的**指令流水线**。这里面每一个独立的步骤，我们就称之为**流水线阶段**或者**流水线级**（Pipeline Stage）。

如果我们把一个指令拆分成“取指令-指令译码-执行指令”这样三个部分，那这就是一个三级的流水线。如果我们进一步把“执行指令”拆分成“ALU计算（指令执行）-内存访问-数据写回”，那么它就会变成一个五级的流水线。

五级的流水线，就表示我们在同一个时钟周期里面，同时运行五条指令的不同阶段。这个时候，虽然执行一条指令的时钟周期变成了5，但是我们可以把CPU的主频提得更高了。**我们不需要确保最复杂的那条指令在时钟周期里面执行完成，而只要保障一个最复杂的流水线级的操作，在一个时钟周期内完成就好了。**

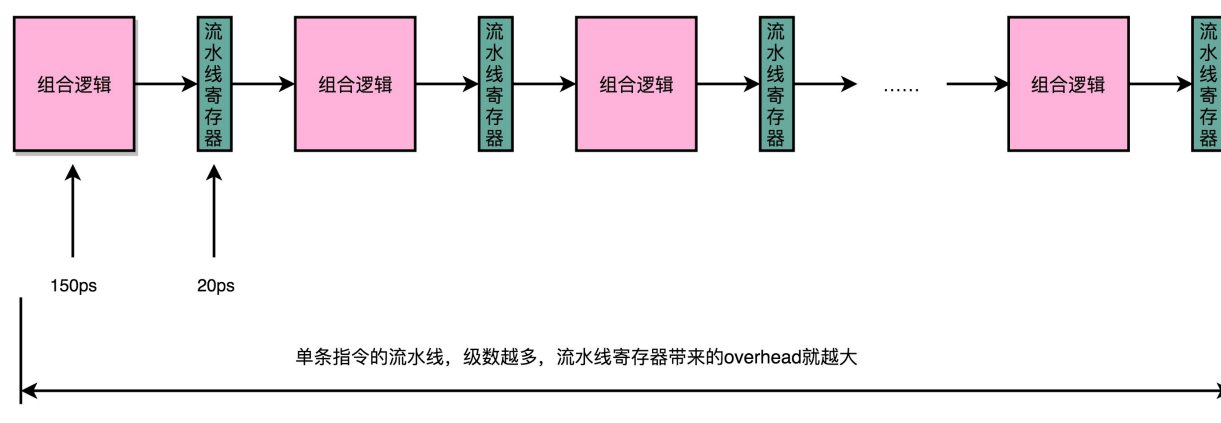
如果某一个操作步骤的时间太长，我们就可以考虑把这个步骤，拆分成更多的步骤，让所有步骤需要执行的时间尽量都差不多长。这样，也就可以解决我们在单指令周期处理器中遇到的，性能瓶颈来自于最复杂的指令的问题。像我们现代的ARM或者Intel的CPU，流水线级数都已经到了14级。

虽然我们不能通过流水线，来减少单条指令执行的“延时”这个性能指标，但是，通过同时在执行多条指令的不同阶段，我们提升了CPU的“吞吐率”。在外部看来，我们的CPU好像是“一心多用”，在同一时间，同时执行5条不同指令的不同阶段。在CPU内部，其实它就像生产线一样，不同分工的组件不断处理上游传递下来的内容，而不需要等待单件商品生产完成之后，再启动下一件商品的生产过程。

## 超长流水线的性能瓶颈

既然流水线可以增加我们的吞吐率，你可能要问了，为什么我们不把流水线级数做得更深呢？为什么不做成20级，乃至40级呢？这个其实有很多原因，我在之后几讲里面会详细讲解。这里，我先讲一个最基本的原因，就是增加流水线深度，其实是有性能成本的。

我们用来同步时钟周期的，不再是指令级别的，而是流水线阶段级别的。每一级流水线对应的输出，都要放到流水线寄存器（Pipeline Register）里面，然后在下一个时钟周期，交给下一个流水线级去处理。所以，每增加一级的流水线，就要多一级写入到流水线寄存器的操作。虽然流水线寄存器非常快，比如只有20皮秒（ps， $10^{-12}$ 秒）。



但是，如果我们不断加深流水线，这些操作占整个指令的执行时间的比例就会不断增加。最后，我们的性能瓶颈就会出现在这些overhead上。如果我们指令的执行有3纳秒，也就是3000皮秒。我们需要20级的流水线，那流水线寄存器的写入就需要花费400皮秒，占了超过10%。如果我们需要50级流水线，就要多花费1纳秒在流水线寄存器上，占到25%。这也就意味着，单纯地增加流水线级数，不仅不能提升性能，反而会有更多的overhead的开销。所以，设计合理的流水线级数也是现代CPU中非常重要的一点。

## 总结延伸

讲到这里，相信你已经能够理解，为什么我们的CPU需要流水线设计了，也能把每一个流水线阶段在干什么，和上一讲的整个CPU的数据通路的连接过程对上了。

可以看到，为了能够不浪费CPU的性能，我们通过把指令的执行过程，切分成一个一个流水线级，来提升

CPU的吞吐率。而我们本身的CPU的设计，又是由一个个独立的组合逻辑电路串接起来形成的，天然能够适合这样采用流水线“专业分工”的工作方式。

因为每一级的overhead，一味地增加流水线深度，并不能无限地提高性能。同样地，因为指令的执行不再是顺序地一条条执行，而是在上一条执行到一半的时候，下一条就已经启动了，所以也给我们的程序带来了许多挑战。这些挑战和对应的解决方案，就要请你持续关注后面的几讲，我们一起来揭开答案了。

## 推荐阅读

想要了解CPU的流水线设计，可以参看《深入理解计算机系统》的4.4章节，以及《计算机组成与设计 硬件/软件接口》的4.5章节。

## 课后思考

我们在前面讲过，一个CPU的时钟周期，可以认为是完成一条简单指令的时间。在这一讲之后，你觉得这句话正确吗？为什么？在了解了CPU的流水线设计之后，你是怎么理解这句话的呢？

欢迎留言和我分享你的疑惑和见解。你也可以把今天的内容，分享给你的朋友，和他一起学习和进步。



# 深入浅出计算机组成原理

## 带你掌握计算机体系全貌



徐文浩 bothub 创始人

新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

## 精选留言：

- 喜欢吃鱼 2019-06-10 12:06:05  
每个流水线级的执行时间应该不一定相同吧，所以一个CPU执行周期是不是执行时间最久的流水线级所需要的时间？ [1赞]
- 易儿易 2019-06-10 20:53:23  
一个 CPU 的时钟周期，可以认为是完成一条简单指令的时间。  
两个错误：  
1简单：不是简单是复杂（耗时最长）  
2指令：不是指令时间，而是指令拆分后线级执行时间  
简单线级需要等待以补齐时间然后与上下流水线一起牵手步入下一线级

线级分的多的话，这个时钟周期应该就可以固定是IF取指令的时间了

如果把IF取指令看做是最简单的指令的话，这句话就没毛病了

但一般说指令应该是指令集的一个命令吧？

但是指令集是提供给开发者视角的，在CPU内部IF取指令才是一个原子化操作，从CPU视角来看又觉得这句话是靠谱的……

- 陈华应 2019-06-10 12:43:17

无条件跳转这样的简单指令三个流水级应该可以在一个时钟周期完成，换句话说就是一个指令在一个时钟周期完成。但是稍微复杂的指令往往是超过一个时钟周期的。

这里有一点疑惑:在流水线设计的CPU中，一个时钟周期只会执行一个流水级吗？对于很简单的流水级，可能执行时间只占时钟周期的一半，那另一半就是什么也不干吗？

- Only now 2019-06-10 10:11:37

一个 CPU 的时钟周期，可以认为是完成一条简单指令的时间。这个应该是依据于吞吐量来理解的吧。

实际指令不能在一个时钟周期完成,但是流水线的引入使吞吐量更高。

- cc 2019-06-10 08:37:48

随着流水线设计的引入，一个指令被拆分为14个子流程。一个CPU的时钟时间，应该是14个子流程中最长的那条的耗时时间