水线的阶段数加倍，我们将性能提高了 14.29/8.33＝1.71。虽然我们将每个计算时钟的时间缩短了两倍，但是由于通过流水线寄存器的延迟，吞吐量并没有加倍。这个延迟成了流水线吞吐量的一个制约因素。在我们的新设计中，这个延迟占到了整个时钟周期的 28.6%。

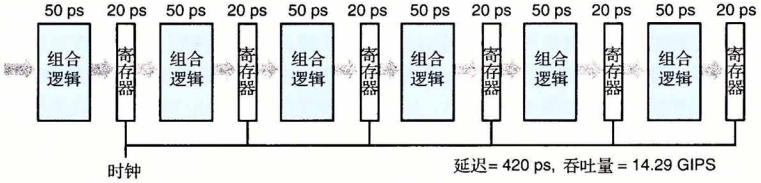

图 4-37　由开销造成的流水线技术的局限性。在组合逻辑被分成较小的块时，由寄存器更新引起的延迟就成为了一个限制因素

为了提高时钟频率，现代处理器采用了很深的(15 或更多的阶段)流水线。处理器架构师将指令的执行划分成很多非常简单的步骤，这样一来每个阶段的延迟就很小。电路设计者小心地设计流水线寄存器，使其延迟尽可能得小。芯片设计者也必须小心地设计时钟传播网络，以保证时钟在整个芯片上同时改变。所有这些都是设计高速微处理器面临的挑战。

练习题 4.29　让我们来看看图 4-32 中的系统，假设将它划分成任意数量的流水线阶段 k，每个阶段有相同的延迟 $300/k$，每个流水线寄存器的延迟为 20ps。

A. 系统的延迟和吞吐量写成 k 的函数是什么？

B. 吞吐量的上限等于多少？

4.4.4　带反馈的流水线系统

到目前为止，我们只考虑一种系统，其中传过流水线的对象，无论是汽车、人或者指令，相互都是完全独立的。但是，对于像 x86-64 或 Y86-64 这样执行机器程序的系统来说，相邻指令之间很可能是相关的。例如，考虑下面这个 Y86-64 指令序列：

```
1    irmovq $50, %rax
2    addq %rax, %rbx
3    mrmovq 100(%rbx), %rdx
```

在这个包含三条指令的序列中，每对相邻的指令之间都有数据相关(data dependency)，用带圈的寄存器名字和它们之间的箭头来表示。irmovq 指令(第 1 行)将它的结果存放在%rax 中，然后 addq 指令(第 2 行)要读这个值；而 addq 指令将它的结果存放在%rbx 中，mrmovq 指令(第 3 行)要读这个值。

另一种相关是由于指令控制流造成的顺序相关。来看看下面这个 Y86-64 指令序列：

```
1    loop:
2        subq %rdx,%rbx
3        jne targ
4        irmovq $10,%rdx
5        jmp loop
6    targ:
7        halt
```