为了理解为什么基于条件数据传送的代码会比基于条件控制转移的代码(如图 3-16 中那样)性能要好，我们必须了解一些关于现代处理器如何运行的知识。正如我们将在第 4 章和第 5 章中看到的，处理器通过使用流水线(pipelining)来获得高性能，在流水线中，一条指令的处理要经过一系列的阶段，每个阶段执行所需操作的一小部分(例如，从内存取指令、确定指令类型、从内存读数据、执行算术运算、向内存写数据，以及更新程序计数器)。这种方法通过重叠连续指令的步骤来获得高性能，例如，在取一条指令的同时，执行它前面一条指令的算术运算。要做到这一点，要求能够事先确定要执行的指令序列，这样才能保持流水线中充满了待执行的指令。当机器遇到条件跳转(也称为“分支”)时，只有当分支条件求值完成之后，才能决定分支往哪边走。处理器采用非常精密的分支预测逻辑来猜测每条跳转指令是否会执行。只要它的猜测还比较可靠(现代微处理器设计试图达到 90%以上的成功率)，指令流水线中就会充满着指令。另一方面，错误预测一个跳转，要求处理器丢掉它为该跳转指令后所有指令已做的工作，然后再开始用从正确位置处起始的指令去填充流水线。正如我们会看到的，这样一个错误预测会招致很严重的惩罚，浪费大约 15～30 个时钟周期，导致程序性能严重下降。

作为一个示例，我们在 Intel Haswell 处理器上运行 absdiff 函数，用两种方法来实现条件操作。在一个典型的应用中，x< y 的结果非常地不可预测，因此即使是最精密的分支预测硬件也只能有大约 50%的概率猜对。此外，两个代码序列中的计算执行都只需要一个时钟周期。因此，分支预测错误处罚主导着这个函数的性能。对于包含条件跳转的 x86-64 代码，我们发现当分支行为模式很容易预测时，每次调用函数需要大约 8 个时钟周期；而分支行为模式是随机的时候，每次调用需要大约 17.50 个时钟周期。由此我们可以推断出分支预测错误的处罚是大约 19 个时钟周期。这就意味着函数需要的时间范围大约在 8 到 27 个周期之间，这依赖于分支预测是否正确。

旁注　如何确定分支预测错误的处罚

假设预测错误的概率是 p，如果没有预测错误，执行代码的时间是 T_{OK}，而预测错误的处罚是 T_{MP}。那么，作为 p 的一个函数，执行代码的平均时间是 $T_{avg}(p)=(1-p)T_{OK}+p(T_{OK}+T_{MP})=T_{OK}+pT_{MP}$。如果已知 T_{OK} 和 T_{ran}(当 $p=0.5$ 时的平均时间)，要确定 T_{MP}。将参数代入等式，我们有 $T_{ran}=T_{avg}(0.5)=T_{OK}+0.5T_{MP}$，所以有 $T_{MP}=2(T_{ran}-T_{OK})$。因此，对于 $T_{OK}=8$ 和 $T_{ran}=17.5$，我们有 $T_{MP}=19$。

另一方面，无论测试的数据是什么，编译出来使用条件传送的代码所需的时间都是大约 8 个时钟周期。控制流不依赖于数据，这使得处理器更容易保持流水线是满的。

练习题 3.19　在一个比较旧的处理器模型上运行，当分支行为模式非常可预测时，我们的代码需要大约 16 个时钟周期，而当模式是随机的时候，需要大约 31 个时钟周期。

A. 预测错误处罚大约是多少？

B. 当分支预测错误时，这个函数需要多少个时钟周期？

图 3-18 列举了 x86-64 上一些可用的条件传送指令。每条指令都有两个操作数：源寄存器或者内存地址 S，和目的寄存器 R。与不同的 SET(3.6.2 节)和跳转指令(3.6.3 节)一样，这些指令的结果取决于条件码的值。源值可以从内存或者源寄存器中读取，但是只有在指定的条件满足时，才会被复制到目的寄存器中。

源和目的的值可以是 16 位、32 位或 64 位长。不支持单字节的条件传送。无条件指令的操作数的长度显式地编码在指令名中(例如 movw 和 movl)，汇编器可以从目标寄存器的名字推断