

我们给出的抽象代码会对 *then-expr* 和 *else-expr* 都求值。如果这两个表达式中的任意一个可能产生错误条件或者副作用，就会导致非法的行为。前面的一个例子(图 3-16)就是这种情况。实际上，我们在该例中引入副作用就是为了强制 GCC 用条件转移来实现这个函数。

作为说明，考虑下面这个 C 函数：

```
long cread(long *xp) {
    return (xp ? *xp : 0);
}
```


乍一看，这段代码似乎很适合被编译成使用条件传送，当指针为空时将结果设置为 0，如下面的汇编代码所示：

```
long cread(long *xp)
Invalid implementation of function cread
xp in register %rdi
1  cread:
2      movq    (%rdi), %rax    v = *xp
3      testq   %rdi, %rdi     Test x
4      movl    $0, %edx       Set ve = 0
5      cmovbe  %rdx, %rax     If x==0, v = ve
6      ret                     Return v
```

不过，这个实现是非法的，因为即使当测试为假时，`movq` 指令(第 2 行)对 `xp` 的间接引用还是发生了，导致一个间接引用空指针的错误。所以，必须用分支代码来编译这段代码。

使用条件传送也不总是会提高代码的效率。例如，如果 *then-expr* 或者 *else-expr* 的求值需要大量的计算，那么当相对应的条件不满足时，这些工作就白费了。编译器必须考虑浪费的计算和由于分支预测错误所造成的性能处罚之间的相对性能。说实话，编译器并不具有足够的信息来做出可靠的决定；例如，它们不知道分支会多好地遵循可预测的模式。我们对 GCC 的实验表明，只有当两个表达式都很容易计算时，例如表达式分别都只是一条加法指令，它才会使用条件传送。根据我们的经验，即使许多分支预测错误的开销会超过更复杂的计算，GCC 还是会使用条件控制转移。

所以，总的来说，条件数据传送提供了一种用条件控制转移来实现条件操作的替代策略。它们只能用于非常受限制的情况，但是这些情况还是相当常见的，而且与现代处理器的运行方式更契合。

 **练习题 3.20** 在下面的 C 函数中，我们对 `OP` 操作的定义是不完整的：

```
#define OP _____ /* Unknown operator */

long arith(long x) {
    return x OP 8;
}
```

当编译时，GCC 会产生如下汇编代码：

```
long arith(long x)
x in %rdi
arith:
    leaq    7(%rdi), %rax
    testq   %rdi, %rdi
    cmovns  %rdi, %rax
    sarq    $3, %rax
    ret
```