3.6.6　用条件传送来实现条件分支

实现条件操作的传统方法是通过使用控制的条件转移。当条件满足时，程序沿着一条执行路径执行，而当条件不满足时，就走另一条路径。这种机制简单而通用，但是在现代处理器上，它可能会非常低效。

一种替代的策略是使用数据的条件转移。这种方法计算一个条件操作的两种结果，然后再根据条件是否满足从中选取一个。只有在一些受限制的情况中，这种策略才可行，但是如果可行，就可以用一条简单的条件传送指令来实现它，条件传送指令更符合现代处理器的性能特性。我们将介绍这一策略，以及它在 x86-64 上的实现。

图 3-17a 给出了一个可以用条件传送编译的示例代码。这个函数计算参数 x 和 y 差的绝对值，和前面的例子一样(图 3-16)。不过前面的例子中，分支里有副作用，会修改 lt_cnt 或 ge_cnt 的值，而这个版本只是简单地计算函数要返回的值。

GCC 为该函数产生的汇编代码如图 3-17c 所示，它与图 3-17b 中所示的 C 函数 cmovdiff 有相似的形式。研究这个 C 版本，我们可以看到它既计算了 y-x，也计算了 x-y，分别命名为 rval 和 eval。然后它再测试 x 是否大于等于 y，如果是，就在函数返回 rval 前，将 eval 复制到 rval 中。图 3-17c 中的汇编代码有相同的逻辑。关键就在于汇编代码的那条 cmovge 指令(第 7 行)实现了 cmovdiff 的条件赋值(第 8 行)。只有当第 6 行的 cmpq 指令表明一个值大于等于另一个值(正如后缀 ge 表明的那样)时，才会把数据源寄存器传送到目的。

```
long absdiff(long x, long y)
{
    long result;
    if (x < y)
        result = y - x;
    else
        result = x - y;
    return result;
}
```

a) 原始的C语言代码

```
1   long cmovdiff(long x, long y)
2   {
3       long rval = y-x;
4       long eval = x-y;
5       long ntest = x >= y;
6       /* Line below requires
7          single instruction: */
8       if (ntest) rval = eval;
9       return rval;
10  }
```

b) 使用条件赋值的实现

```
    long absdiff(long x, long y)
    x in %rdi, y in %rsi
1   absdiff:
2     movq    %rsi, %rax
3     subq    %rdi, %rax        rval = y-x
4     movq    %rdi, %rdx
5     subq    %rsi, %rdx        eval = x-y
6     cmpq    %rsi, %rdi        Compare x:y
7     cmovge  %rdx, %rax        If >=, rval = eval
8     ret                       Return tval
```

c) 产生的汇编代码

图 3-17　使用条件赋值的条件语句的编译。a)C 函数 absdiff 包含一个条件表达式；b)C 函数 cmovdiff 模拟汇编代码操作；c)给出产生的汇编代码