备，你会看到令人惊奇的变换，其中有些情况很明显是编译器能够优化代码，而有些情况很难解释编译器为什么要选用那些奇怪的策略。根据我们的经验，GCC 常常做的一些变换，非但不能带来性能好处，反而甚至可能降低代码性能。

2. while 循环

while 语句的通用形式如下：

```
while (test-expr)
    body-statement
```

与 do-while 的不同之处在于，在第一次执行 body-statement 之前，它会对 test-expr 求值，循环有可能就中止了。有很多种方法将 while 循环翻译成机器代码，GCC 在代码生成中使用其中的两种方法。这两种方法使用同样的循环结构，与 do-while 一样，不过它们实现初始测试的方法不同。

第一种翻译方法，我们称之为跳转到中间(jump to middle)，它执行一个无条件跳转跳到循环结尾处的测试，以此来执行初始的测试。可以用以下模板来表达这种方法，这个模板把通用的 while 循环格式翻译到 goto 代码：

```
    goto test;
loop:
    body-statement
test:
    t = test-expr;
    if (t)
        goto loop;
```

作为一个示例，图 3-20a 给出了使用 while 循环的阶乘函数的实现。这个函数能够正确地计算 0!=1。它旁边的函数 fact_while_jm_goto(图 3-20b)是 GCC 带优化命令行选项-Og 时产生的汇编代码的 C 语言翻译。比较 fact_while(图 3-20b)和 fact_do(图 3-19b)的代码，可以看到它们非常相似，区别仅在于循环前的 goto test 语句使得程序在修改 result 或 n 的值之前，先执行对 n 的测试。图的最下面(图 3-20c)给出的是实际产生的汇编代码。

练习题 3.24　对于如下 C 代码：

```
long loop_while(long a, long b)
{
    long result = _____;
    while (_____) {
        result = _____;
        a = _____;
    }
    return result;
}
```

以命令行选项-Og 运行 GCC 产生如下代码：

```
    long loop_while(long a, long b)
    a in %rdi, b in %rsi
1   loop_while:
2     movl    $1, %eax
3     jmp     .L2
```