```
4   .L3:
5     leaq    (%rdi,%rsi), %rdx
6     imulq   %rdx, %rax
7     addq    $1, %rdi
8   .L2:
9     cmpq    %rsi, %rdi
10    jl      .L3
11    rep; ret
```

可以看到编译器使用了跳转到中间的翻译方法，在第 3 行用 jmp 跳转到以标号 .L2开始的测试。填写 C 代码中缺失的部分。

```
long fact_while(long n)
{
    long result = 1;
    while (n > 1) {
        result *= n;
        n = n-1;
    }
    return result;
}
```

a）C代码

```
long fact_while_jm_goto(long n)
{
    long result = 1;
    goto test;
 loop:
    result *= n;
    n = n-1;
 test:
    if (n > 1)
        goto loop;
    return result;
}
```

b）等价的goto版本

```
  long fact_while(long n)
  n in %rdi
fact_while:
  movl    $1, %eax        Set result = 1
  jmp     .L5             Goto test
.L6:                    loop:
  imulq   %rdi, %rax      Compute result *= n
  subq    $1, %rdi        Decrement n
.L5:                    test:
  cmpq    $1, %rdi        Compare n:1
  jg      .L6             If >, goto loop
  rep; ret                Return
```

c）对应的汇编代码

图 3-20　使用跳转到中间翻译方法的阶乘算法的 while 版本的 C 代码和汇编代码。C 函数 fact_while_jm_goto 说明了汇编代码版本的操作

第二种翻译方法，我们称之为 guarded-do，首先用条件分支，如果初始条件不成立就跳过循环，把代码变换为 do-while 循环。当使用较高优化等级编译时，例如使用命令行选项-O1，GCC 会采用这种策略。可以用如下模板来表达这种方法，把通用的 while 循环