


逆向工程像图 3-19c 中那样的汇编代码，需要确定哪个寄存器对应的是哪个程序值。本例中，这个对应关系很容易确定：我们知道 n 在寄存器 `%rdi` 中传递给函数。可以看到寄存器 `%rax` 初始化为 1 (第 2 行)。(注意，虽然指令的目的寄存器是 `%eax`，它实际上还会把 `%rax` 的高 4 字节设置为 0。)还可以看到这个寄存器还会在第 4 行被乘法改变值。此外，`%rax` 用来返回函数值，所以通常会用来存放需要返回的程序值。因此我们断定 `%rax` 对应程序值 `result`。

 **练习题 3.23** 已知 C 代码如下：

```
long dw_loop(long x) {
    long y = x*x;
    long *p = &x;
    long n = 2*x;
    do {
        x += y;
        (*p)++;
        n--;
    } while (n > 0);
    return x;
}
```

GCC 产生的汇编代码如下：

```
long dw_loop(long x)
x initially in %rdi
dw_loop:
1      movq    %rdi, %rax
2      movq    %rdi, %rcx
3      imulq   %rdi, %rcx
4      leaq    (%rdi,%rdi), %rdx
5      .L2:
6      leaq    1(%rcx,%rax), %rax
7      subq    $1, %rdx
8      testq   %rdx, %rdx
9      jg      .L2
10     rep; ret
```

- 哪些寄存器用来存放程序值 x 、 y 和 n ?
- 编译器如何消除对指针变量 p 和表达式 $(*p)++$ 隐含的指针间接引用的需求?
- 对汇编代码添加一些注释，描述程序的操作，类似于图 3-19c 中所示的那样。

旁注 逆向工程循环

理解产生的汇编代码与原始源代码之间的关系，关键是找到程序值和寄存器之间的映射关系。对于图 3-19 的循环来说，这个任务非常简单，但是对于更复杂的程序来说，就可能是更具挑战性的任务。C 语言编译器常常会重组计算，因此有些 C 代码中的变量在机器代码中没有对应的值；而有时，机器代码中又会引入源代码中不存在的新值。此外，编译器还常常试图将多个程序值映射到一个寄存器上，来最小化寄存器的使用率。

我们描述 `fact_do` 的过程对于逆向工程循环来说，是一个通用的策略。看看在循环之前如何初始化寄存器，在循环中如何更新和测试寄存器，以及在循环之后又如何使用寄存器。这些步骤中的每一步都提供了一个线索，组合起来就可以解开谜团。做好准