以上代码是编译以下整体形式的C代码产生的：

```
long loop(long x, int n)
{
    long result = _____;
    long mask;
    for (mask = _____; mask _____ ; mask = _____ ) {
        result |= _____ ;
    }
    return result;
}
```

你的任务是填写这个C代码中缺失的部分，得到一个程序等价于产生的汇编代码。回想一下，这个函数的结果是在寄存器%rax中返回的。你会发现以下工作很有帮助：检查循环之前、之中和之后的汇编代码，形成一个寄存器和程序变量之间一致的映射。

A. 哪个寄存器保存着程序值x、n、result和mask?

B. result和mask的初始值是什么?

C. mask的测试条件是什么?

D. mask是如何被修改的?

E. result是如何被修改的

F. 填写这段C代码中所有缺失的部分。

** 3.61　在3.6.6节，我们查看了下面的代码，作为使用条件数据传送的一种选择：

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

我们给出了使用条件传送指令的一个尝试实现，但是认为它是不合法的，因为它试图从一个空地址读数据。

写一个C函数cread_alt，它与cread有一样的行为，除了它可以被编译成使用条件数据传送。当编译时，产生的代码应该使用条件传送指令而不是某种跳转指令。

** 3.62　下面的代码给出了一个开关语句中根据枚举类型值进行分支选择的例子。回忆一下，C语言中枚举类型只是一种引入一组与整数值相对应的名字的方法。默认情况下，值是从0向上依次赋给名字的。在我们的代码中，省略了与各种情况标号相对应的动作。

```
/* Enumerated type creates set of constants numbered 0 and upward */
typedef enum {MODE_A, MODE_B, MODE_C, MODE_D, MODE_E} mode_t;

long switch3(long *p1, long *p2, mode_t action)
{
    long result = 0;
    switch(action) {
    case MODE_A:

    case MODE_B:

    case MODE_C:

    case MODE_D:

    case MODE_E:

    default:

    }
    return result;
}
```