```
long decode2(long x, long y, long z);
```

GCC 产生如下汇编代码：

```
1   decode2:
2     subq    %rdx, %rsi
3     imulq   %rsi, %rdi
4     movq    %rsi, %rax
5     salq    $63, %rax
6     sarq    $63, %rax
7     xorq    %rdi, %rax
8     ret
```

参数 x、y 和 z 通过寄存器%rdi、%rsi 和%rdx 传递。代码将返回值存放在寄存器%rax 中。

写出等价于上述汇编代码的 decode2 的 C 代码。

**3.59　下面的代码计算两个 64 位有符号值 x 和 y 的 128 位乘积，并将结果存储在内存中：

```
1   typedef __int128 int128_t;
2
3   void store_prod(int128_t *dest, int64_t x, int64_t y) {
4       *dest = x * (int128_t) y;
5   }
```

GCC 产出下面的汇编代码来实现计算：

```
1   store_prod:
2     movq    %rdx, %rax
3     cqto
4     movq    %rsi, %rcx
5     sarq    $63, %rcx
6     imulq   %rax, %rcx
7     imulq   %rsi, %rdx
8     addq    %rdx, %rcx
9     mulq    %rsi
10    addq    %rcx, %rdx
11    movq    %rax, (%rdi)
12    movq    %rdx, 8(%rdi)
13    ret
```

为了满足在 64 位机器上实现 128 位运算所需的多精度计算，这段代码用了三个乘法。描述用来计算乘积的算法，对汇编代码加注释，说明它是如何实现你的算法的。**提示**：在把参数 x 和 y 扩展到 128 位时，它们可以重写为 $x=2^{64}\cdot x_h+x_l$ 和 $y=2^{64}\cdot y_h+y_l$，这里 x_h，x_l，y_h 和 y_l 都是 64 位值。类似地，128 位的乘积可以写成 $p=2^{64}\cdot p_h+p_l$，这里 p_h 和 p_l 是 64 位值。请解释这段代码是如何用 x_h，x_l，y_h 和 y_l 来计算 p_h 和 p_l 的。

**3.60　考虑下面的汇编代码：

```
    long loop(long x, int n)
    x in %rdi, n in %esi
1   loop:
2     movl    %esi, %ecx
3     movl    $1, %edx
4     movl    $0, %eax
5     jmp     .L2
6   .L3:
7     movq    %rdi, %r8
8     andq    %rdx, %r8
9     orq     %r8, %rax
10    salq    %cl, %rdx
11  .L2:
12    testq   %rdx, %rdx
13    jne     .L3
14    rep; ret
```