3.6　这个练习说明了 leaq 指令的多样性，同时也让你更多地练习解读各种操作数形式。虽然在图 3-3 中有的操作数格式被划分为“内存”类型，但是并没有访存发生。

指令	结果
leaq 6(%rax),%rdx	$6+x$
leaq(%rax,%rcx),%rdx	$x+y$
leaq(%rax,%rcx,4),%rdx	$x+4y$
leaq 7(%rax,%rax,8),%rdx	$7+9x$
leaq 0xA(,%rcx,4),%rdx	$10+4y$
leaq 9(%rax,%rcx,2),%rdx	$9+x+2y$

3.7　逆向工程再次被证明是学习 C 代码和生成的汇编代码之间关系的有用方式。

解决此类型问题的最好方式是为汇编代码行加注释，说明正在执行的操作信息。下面是一个例子：

```
  long scale2(long x, long y, long z)
  x in %rdi, y in %rsi, z in %rdx
scale2:
  leaq    (%rdi,%rdi,4), %rax    5 * x
  leaq    (%rax,%rsi,2), %rax    5 * x + 2 * y
  leaq    (%rax,%rdx,8), %rax    5 * x + 2 * y + 8 * z
  ret
```

由此很容易得到缺失的表达式：

```
long t = 5 * x + 2 * y + 8 * z;
```

3.8　这个练习使你有机会检验对操作数和算术指令的理解。指令序列被设计成每条指令的结果都不会影响后续指令的行为。

指令	目的	值
addq %rcx,(%rax)	0x100	0x100
subq %rdx,8(%rax)	0x108	0xA8
imulq $16,(%rax,%rdx,8)	0x118	0x110
incq 16(%rax)	0x110	0x14
decq %rcx	%rcx	0x0
subq %rdx,%rax	%rax	OXFD

3.9　这个练习使你有机会生成一点汇编代码。答案的代码由 GCC 生成。将参数 n 加载到寄存器%ecx 中，它可以用字节寄存器%cl 来指定 sarl 指令的移位量。使用 movl 指令看上去有点儿奇怪，因为 n 的长度是 8 字节，但是要记住只有最低位的那个字节才指示着移位量。

```
  long shift_left4_rightn(long x, long n)
  x in %rdi, n in %rsi
shift_left4_rightn:
  movq    %rdi, %rax     Get x
  salq    $4, %rax       x <<= 4
  movl    %esi, %ecx     Get n (4 bytes)
  sarq    %cl, %rax      x >>= n
```

3.10　这个练习比较简单，因为汇编代码基本上沿用了 C 代码的结构。

```
long t1 = x | y;
long t2 = t1 >> 3;
long t3 = ~t2;
long t4 = z-t3;
```