练习题 3.3　当我们调用汇编器的时候，下面代码的每一行都会产生一个错误消息。解释每一行都是哪里出了错。

```
movb $0xF, (%ebx)
movl %rax, (%rsp)
movw (%rax),4(%rsp)
movb %al,%sl
movq %rax,$0x123
movl %eax,%rdx
movb %si, 8(%rbp)
```

3.4.3　数据传送示例

作为一个使用数据传送指令的代码示例，考虑图 3-7 中所示的数据交换函数，既有 C 代码，也有 GCC 产生的汇编代码。

```
long exchange(long *xp, long y)
{
    long x = *xp;
    *xp = y;
    return x;
}
```

a）C语言代码

```
    long exchange(long *xp, long y)
    xp in %rdi, y in %rsi
1   exchange:
2     movq    (%rdi), %rax    Get x at xp. Set as return value.
3     movq    %rsi, (%rdi)    Store y at xp.
4     ret                     Return.
```

b）汇编代码

图 3-7　exchange 函数的 C 语言和汇编代码。寄存器%rdi 和%rsi 分别存放参数 xp 和 y

如图 3-7b 所示，函数 exchange 由三条指令实现：两个数据传送(movq)，加上一条返回函数被调用点的指令(ret)。我们会在 3.7 节中讲述函数调用和返回的细节。在此之前，知道参数通过寄存器传递给函数就足够了。我们对汇编代码添加注释来加以说明。函数通过把值存储在寄存器%rax 或该寄存器的某个低位部分中返回。

当过程开始执行时，过程参数 xp 和 y 分别存储在寄存器%rdi 和%rsi 中。然后，指令 2 从内存中读出 x，把它存放到寄存器%rax 中，直接实现了 C 程序中的操作 x=*xp。稍后，用寄存器%rax 从这个函数返回一个值，因而返回值就是 x。指令 3 将 y 写入到寄存器%rdi 中的 xp 指向的内存位置，直接实现了操作*xp=y。这个例子说明了如何用 MOV 指令从内存中读值到寄存器(第 2 行)，如何从寄存器写到内存(第 3 行)。

关于这段汇编代码有两点值得注意。首先，我们看到 C 语言中所谓的“指针”其实就是地址。间接引用指针就是将该指针放在一个寄存器中，然后在内存引用中使用这个寄存器。其次，像 x 这样的局部变量通常是保存在寄存器中，而不是内存中。访问寄存器比访问内存要快得多。