下面给出了有错误解释的代码：

```
movb $0xF, (%ebx)          Cannot use %ebx as address register
movl %rax, (%rsp)          Mismatch between instruction suffix and register ID
movw (%rax),4(%rsp)        Cannot have both source and destination be memory references
movb %al,%sl               No register named %sl
movq %rax,$0x123           Cannot have immediate as destination
movl %eax,%rdx             Destination operand incorrect size
movb %si, 8(%rbp)          Mismatch between instruction suffix and register ID
```

3.4　这个练习给你更多经验，关于不同的数据传送指令，以及它们与 C 语言的数据类型和转换规则的关系。

src_t	dest_t	指令	注释
long	long	movq(%rdi),%rax	读 8 个字节
		movq %rax,(%rsi)	存 8 个字节
char	int	novsbl(%rdi),%eax	将 char 转换成 int
		movl %eax,(%rsi)	存 4 个字节
char	unsigned	movsbl(%rdi),%eax	将 char 转换成 int
		movl %eax,(%rsi)	存 4 个字节
unsigned char	long	movzbl(%rdi),%eax	读一个字节并零扩展
		movq %rax,(%rsi)	存 8 个字节
int	char	movl(%rdi),%eax	读 4 个字节
		movb %al,(%rsi)	存低位字节
unsigned	unsigned	movl(%rdi),%eax	读 4 个字节
	char	movb %al,(%rsi)	存低位字节
char	short	movsbw(%rdi),%ax	读一个字节并符号扩展
		movw %ax,(%rsi)	存 2 个字节

3.5　逆向工程是一种理解系统的好方法。在此，我们想要逆转 C 编译器的效果，来确定什么样的 C 代码会得到这样的汇编代码。最好的方法是进行“模拟”，从值 x、y 和 z 开始，它们分别在指针 xp、yp 和 zp 指定的位置。于是，我们可以得到下面这样的效果：

```
  void decode1(long *xp, long *yp, long *zp)
  xp in %rdi, yp in %rsi, zp in %rdx
decode1:
  movq    (%rdi), %r8     Get x = *xp
  movq    (%rsi), %rcx    Get y = *yp
  movq    (%rdx), %rax    Get z = *zp
  movq    %r8, (%rsi)     Store x at yp
  movq    %rcx, (%rdx)    Store y at zp
  movq    %rax, (%rdi)    Store z at xp
  ret
```

由此可以产生下面这样的 C 代码：

```
void decode1(long *xp, long *yp, long *zp)
{
    long x = *xp;
    long y = *yp;
    long z = *zp;

    *yp = x;
    *zp = y;
    *xp = z;
}
```