



作业 3

3.3、3.5、3.7、3.10、3.15、3.18、3.21、3.60、3.61

 **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)
```

1. 寄存器%ebx 不能用作地址寄存器，应使用%rbx
2. 指令后缀与寄存器 ID（长度）不匹配，应改为 movq
3. 目标操作数与源操作数不能都是内存引用
4. 没有名为%sl 的寄存器
5. 立即数不能作为目标地址
6. 目标操作数与源操作数的大小不匹配
7. 指令后缀与寄存器 ID（长度）不匹配，应改为 movq

 **练习题 3.5** 已知信息如下。将一个原型为

```
void decode1(long *xp, long *yp, long *zp);
```


的函数编译成汇编代码，得到如下代码：

```
void decode1(long *xp, long *yp, long *zp)
    xp in %rdi, yp in %rsi, zp in %rdx
decode1:
    movq    (%rdi), %r8
    movq    (%rsi), %rcx
    movq    (%rdx), %rax
    movq    %r8, (%rsi)
    movq    %rcx, (%rdx)
    movq    %rax, (%rdi)
    ret
```

参数 xp、yp 和 zp 分别存储在对应的寄存器%rdi、%rsi 和%rdx 中。

请写出等效于上面汇编代码的 decode1 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;
}
```

 练习题 3.7 考虑下面的代码，我们省略了被计算的表达式：

```
long scale2(long x, long y, long z) {  
    long t = _____;  
    return t;  
}
```

用 GCC 编译实际的函数得到如下的汇编代码：

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


填写出 C 代码中缺失的表达式。

1. $t = 5 * x;$

2. $t = t + 2 * y;$

3. $t = t + 8 * z;$

$t = 5 * x + 2 * y + 8 * z;$

 练习题 3.10 下面的函数是图 3-11a 中函数一个变种，其中有些表达式用空格替代：

```
long arith2(long x, long y, long z)  
{  
    long t1 = _____;  
    long t2 = _____;  
    long t3 = _____;  
    long t4 = _____;  
    return t4;  
}
```

实现这些表达式的汇编代码如下：

```
long arith2(long x, long y, long z)  
x in %rdi, y in %rsi, z in %rdx
```

```
arith2:  
    orq     %rsi, %rdi  
    sarq    $3, %rdi  
    notq    %rdi  
    movq    %rdx, %rax  
    subq    %rdi, %rax  
    ret
```


基于这些汇编代码，填写 C 语言代码中缺失的部分。

$\text{long } t1 = x \mid y;$

$\text{long } t2 = t1 \gg 3;$

$\text{long } t3 = \sim t2;$

$\text{long } t4 = z - t3;$

 **练习题 3.15** 在下面这些反汇编二进制代码节选中，有些信息被 X 代替了。回答下列关于这些指令的问题。

A. 下面 je 指令的目标是什么？（在此，你不需要知道任何有关 callq 指令的信息。）

```
4003fa: 74 02          je      XXXXXX
4003fc: ff d0         callq   *%rax
```

B. 下面 je 指令的目标是什么？

```
40042f: 74 f4          je      XXXXXX
400431: 5d            pop     %rbp
```

C. ja 和 pop 指令的地址是多少？

```
XXXXXX: 77 02          ja      400547
XXXXXX: 5d            pop     %rbp
```

D. 在下面的代码中，跳转目标的编码是 PC 相对的，且是一个 4 字节补码数。字节按照从最低位到最高位的顺序列出，反映出 x86-64 的小端法字节顺序。跳转目标的地址是什么？

```
4005e8: e9 73 ff ff    jmpq    XXXXXXXX
4005ed: 90            nop
```

跳转指令提供了一种实现条件执行(if)和几种不同循环结构的方式。

A. 目标偏移量是 0x02，即 2

je 指令的目标是 $4003fc + 2 = 4003fe$

B. 目标偏移量是 0xf4，即-12

jb 指令的目标是 $400431 - 12 = 400425$


C. 目标偏移量是 0x02，即 2

ja 指令的地址是 $400547 - 2 - 2 = 400543$

pop 指令的地址是 $400547 - 2 = 400545$

D. 目标偏移量是 0xfffff73，即-141

跳转目标的地址是 $4005ed - 141 = 400560$

 练习题 3.18 从如下形式的 C 语言代码开始：


```
long test(long x, long y, long z) {
    long val = _____;
    if (_____) {
        if (_____)
            val = _____;
        else
            val = _____;
    } else if (_____)
        val = _____;
    return val;
}
```

GCC 产生如下的汇编代码：

```
long test(long x, long y, long z)
x in %rdi, y in %rsi, z in %rdx
test:
    leaq    (%rdi,%rsi), %rax
    addq    %rdx, %rax
    cmpq    $-3, %rdi
    jge     .L2
    cmpq    %rdx, %rsi
    jge     .L3
    movq    %rdi, %rax
    imulq    %rsi, %rax
    ret
.L3:
    movq    %rsi, %rax
    imulq    %rdx, %rax
    ret
.L2:
    cmpq    $2, %rdi
    jle     .L4
    movq    %rdi, %rax
    imulq    %rdx, %rax
.L4:
    rep; ret
```

填写 C 代码中缺失的表达式。

```
long test(long x, long y, long z) {
    long val = x + y + z;
    if (val < -3) {
        if (y < z)
            val = x * y;
        else
            val = y * z;
    } else if (x > 2)
        val = x * z;
    return val;
}
```

 练习题 3.21 C 代码开始的形式如下:

```
long test(long x, long y) {
    long val = _____;
    if (_____) {
        if (_____)
            val = _____;
        else
            val = _____;
    } else if (_____)
        val = _____;
    return val;
}
```

GCC 会产生如下汇编代码:

```
long test(long x, long y)
x in %rdi, y in %rsi
test:
    leaq    0(,%rdi,8), %rax
    testq   %rsi, %rsi
    jle     .L2
    movq     %rsi, %rax
    subq     %rdi, %rax
    movq     %rdi, %rdx
    andq     %rsi, %rdx
    cmpq     %rsi, %rdi
    cmovge   %rdx, %rax
    ret
.L2:
    addq     %rsi, %rdi
    cmpq     $-2, %rsi
    cmovle   %rdi, %rax
    ret
```

填补 C 代码中缺失的表达式。

```
long test(long x, long y) {
    long val = 8 * x;
    if (y > 0) {
        if (x < y)
            val = y - x;
        else
            val = x & y;
    } else if (y <= -2)
        val = x + y;
    return val;
}
```

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
```

以上代码是编译以下整体形式的 C 代码产生的：

```
1  long loop(long x, int n)
2  {
3      long result = _____;
4      long mask;
5      for (mask = _____; mask _____; mask = _____) {
6          result |= _____;
7      }
8      return result;
9  }
```

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。

A. `x` 保存在 `%rdi` 中，`n` 保存在 `%esi` 中，`result` 保存在 `%rax` 中，`mask` 保存在 `%rdx` 中

B. `result` 的初始值是 0，`mask` 的初始值是 1

C. `mask != 0`

D. `mask = mask << n`

E. `result = result | (x & mask)`

F. `long loop2(long x, int n) {`
 `long result = 0;`
 `long mask;`
 `for (mask = 1; mask != 0; mask = mask << n) {`
 `result |= (x & mask);`
 `}`
 `return result;`
}

**** 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。

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