

### 练习题 3.34 一个具有通用结构的 C 函数如下：

```
int rfun(unsigned x) {
    if (_____)
        return _____;
    unsigned nx = _____;
    int rv = rfun(nx);
    return _____;
}
```

GCC 产生如下汇编代码 (省略了建立和完成代码)：

```
1    movl    8(%ebp), %ebx
2    movl    $0, %eax
3    testl   %ebx, %ebx
4    je      .L3
5    movl    %ebx, %eax
6    shrl    %eax                Shift right by 1
7    movl    %eax, (%esp)
8    call    rfun
9    movl    %ebx, %edx
10   andl    $1, %edx
11   leal    (%edx,%eax), %eax
12   .L3:
```

A. rfun 存储在被调用者保存寄存器 %ebx 中的值是什么？

B. 填写上述 C 代码中缺失的表达式。

C. 用自然语言描述这段代码计算的功能。

A. 参数  $x$

B.  $\text{if } (x == 0)$

$\text{return } 0;$

$\text{unsigned } nx = x \gg 1;$

$\text{int } rv = \text{rfun}(nx);$

$\text{return } (x \& 0x1) + rv;$

C. 递归计算  $x$  中 1 的个数

	初始	1			
%ebx	$x$	$x$			
%eax	0	$x$	$\left[ \frac{x}{2} \right]$	...	
%edx		( $x \& 1$ )			

如果 %ebx 即  $x$  为 0, 跳过嵌套调用

最后-层 %ebx: 0. %eax: 0.

上一次 %ebx: 1. %edx: 1. %eax: 1.

再上一次 %ebx: 2. %edx:  $2 \& 1$  %eax:  $1 + \text{eax}$

### \*\*3.56 考虑下面的汇编代码：

```
x at %ebp+8, n at %ebp+12
1    movl    8(%ebp), %esi
2    movl    12(%ebp), %ebx
3    movl    $1431655765, %edi
4    movl    $-2147483648, %edx
5    .L2:
6    movl    %edx, %eax
7    andl    %esi, %eax
8    xorl    %eax, %edi
9    movl    %ebx, %ecx
10   shrl    %cl, %edx
```

%esi:  $x$

%ebx:  $n$

%edi:  $\$1431655765$  result

%edx:  $\$-2147483648$  mask

%eax:  $\$-2147483648$

%ecx:  $n$

%edx:  $\gg n \text{ 次}$

```

11 testl %edx, %edx    %edx == 0
12 jne .L2
13 movl %edi, %eax

```

以上代码是以下整体形式的 C 代码编译产生的：

```

1 int loop(int x, int n)
2 {
3     int result = 1431655765;
4     int mask; -2147483648
5     for (mask = 70; mask = mask > (n & 0xff)
6         result ^= mask
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 %eax 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。

- 哪个寄存器保存着程序值 x、n、result 和 mask？
- result 和 mask 的初始值是什么？
- mask 的测试条件是什么？
- mask 是如何被修改的？
- result 是如何被修改的？
- 填写这段 C 代码中所有缺失的部分。

- 最后-步)  
%esi: x    %ebx: n    %edi: result    %edx: mask
- result: 1431655765    mask: -2147483648
- 判断是否 0
- 向右移 (n & 0xff) 位
- result ^= x & mask

**\*\*3.59** 这个程序给你一个机会，逆向工程一个 switch 语句。在下面这个过程中，去掉了 switch 语句的主体：

```

1 int switch_prob(int x, int n)
2 {
3     int result = x;
4
5     switch(n) {
6         /* Fill in code here */
7     }
8     return result;
9 }

```

图 3-44 给出了这个过程的反汇编机器代码。我们可以看到，在第 4 行，参数  $n$  被加载到寄存器  $\%eax$  中。

1	08048420 <switch_prob>:	
2	8048420: 55	push %ebp
3	8048421: 89 e5	mov %esp,%ebp
4	8048423: 8b 45 0c	mov 0xc(%ebp),%eax
5	8048426: 83 e8 28	sub \$0x28,%eax
6	8048429: 83 f8 05	cmp \$0x5,%eax
7	804842c: 77 07	ja 8048435 <switch_prob+0x15>
8	804842e: ff 24 85 f0 85 04 08	jmp *0x80485f0(,%eax,4)
9	8048435: 8b 45 08	mov 0x8(%ebp),%eax
10	8048438: eb 24	jmp 804845e <switch_prob+0x3e>
11	804843a: 8b 45 08	mov 0x8(%ebp),%eax
12	804843d: 8d 76 00	lea 0x0(%esi),%esi
13	8048440: eb 19	jmp 804845b <switch_prob+0x3b>
14	8048442: 8b 45 08	mov 0x8(%ebp),%eax
15	8048445: c1 e0 03	shl \$0x3,%eax
16	8048448: eb 17	jmp 8048461 <switch_prob+0x41>
17	804844a: 8b 45 08	mov 0x8(%ebp),%eax
18	804844d: c1 f8 03	sar \$0x3,%eax
19	8048450: eb 0f	jmp 8048461 <switch_prob+0x41>
20	8048452: 8b 45 08	mov 0x8(%ebp),%eax
21	8048455: c1 e0 03	shl \$0x3,%eax
22	8048458: 2b 45 08	sub 0x8(%ebp),%eax
23	804845b: 0f af c0	imul %eax,%eax
24	804845e: 83 c0 11	add \$0x11,%eax
25	8048461: 5d	pop %ebp
26	8048462: c3	ret

$\%eax = n$        $n - 0x28$   
 $> 5$

图 3-44 家庭作业 3.59 的反汇编代码

跳转表驻留在另一个存储区域中。可以从第 8 行的间接跳转看出来，跳转表的起始地址为  $0x80485f0$ 。用调试器 GDB，我们可以用命令  $x/6w \ 0x80485f0$  来检查存储器中的 6 个 4 字节的字。GDB 打印出下面的内容：

```

(gdb) x/6w 0x80485f0
0x80485f0: 0x08048442 0x08048435 0x08048442 0x0804844a
0x8048600: 0x08048452 0x0804843a

```

用 C 代码填写开关语句的主体，使它的行为与机器代码一致。

case 40:  $! \text{break}$ ;

case 42:  $\text{result} = \text{result} \ll 3$ ;  $\text{break}$ ;

case 43:  $\text{result} = \text{result} \gg 3$ ;  $\text{break}$ ;

case 44:  $\text{result} = \text{result} \ll 3$ ;  $\text{result} -= x$ ;  
 $! \text{break}$ ;

case 45: result \*= result; (break)

default: result += 17;

3.36

你负责维护一个大型的 C 程序时，遇到下面这样的代码：

```
1 typedef struct {
2     int left;
3     a_struct a[CNT];
4     int right;
5 } b_struct;
6
7 void test(int i, b_struct *bp)
8 {
9     int n = bp->left + bp->right;
10    a_struct *ap = &bp->a[i];
11    ap->x[ap->idx] = n;
12 }
```

a\_struct {

x  
idx

$16 \times 12 + 8$

编译时常数 CNT 和结构 a\_struct 的声明在一个你没有访问权限的文件中。幸好，你有代码的 '.o' 版本，可以用 OBJDUMP 程序来反汇编这些文件，得到如图 3-45 所示的反汇编代码。

\*bp + c8

1	00000000	<test>	
2	0:	55	push %ebp
3	1:	89 e5	mov %esp, %ebp
4	3:	53	push %ebx
5	4:	8b 45 08	mov 0x8(%ebp), %eax
6	7:	8b 4d 0c	mov 0xc(%ebp), %ecx
7	a:	6b d8 1c	imul \$0x1c, %eax, %ebx
8	d:	8d 14 c5 00 00 00 00	lea 0x0(, %eax, 8), %edx
9	14:	29 c2	sub %eax, %edx
10	16:	03 54 19 04	add 0x4(%ecx, %ebx, 1), %edx
11	1a:	8b 81 c8 00 00 00	mov 0xc8(%ecx), %eax
12	20:	03 01	add (%ecx), %eax
13	22:	89 44 91 08	mov %eax, 0x8(%ecx, %edx, 4)
14	26:	5b	pop %ebx
15	27:	5d	pop %ebp
16	28:	c3	ret

%eax i  
%ecx \*bp  
%ebx 28i  
%edx 8i - i  
+ (\*bp + 28i + 4)

图 3-45 家庭作业 3.66 的反汇编代码

运用你的逆向工程技术，推断出下列内容：

A. CNT 的值。

B. 结构 a\_struct 的完整声明。假设这个结构中只有字段 idx 和 x。

$196 = 28 \times 7$

imul \$0x1c, %eax, %ebx  
%ebx \* 28 → %ebx

$bp + 4( ) + 28$   
 $+ 28i + 4 + 4$

A. CNT = 7

B. typedef struct {  
 int idx;  
 int x[6];  
}