计算机系统第二次作业

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。

练习题 3.34 一个具有通用结构的 C 函数如下：

```
int rfun(unsigned x) {
    if ( ____x==0____ )
        return ____0____;
    unsigned nx = ____x>>1____;
    int rv = rfun(nx);
    return (x&0x1)+rv;
}
```

A. 保存的是传入的参数 x

C. 这段代码是用递规计算无符号数 x 的二进制表示中 1 的个数

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   x>>1
7     movl    %eax, (%esp)
8     call    rfun
9     movl    %ebx, %edx
10    andl    $1, %edx
11    leal    (%edx,%eax), %eax   rv
12  .L3:
```

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
11    testl   %edx, %edx
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;
5      for (mask = -2147483648; mask  !=0  ; mask = (unsigned) mask>>n ) {
6          result ^= x&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。

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. %esi保存x； %ebx保存n； %edi保存result； %edx保存mask

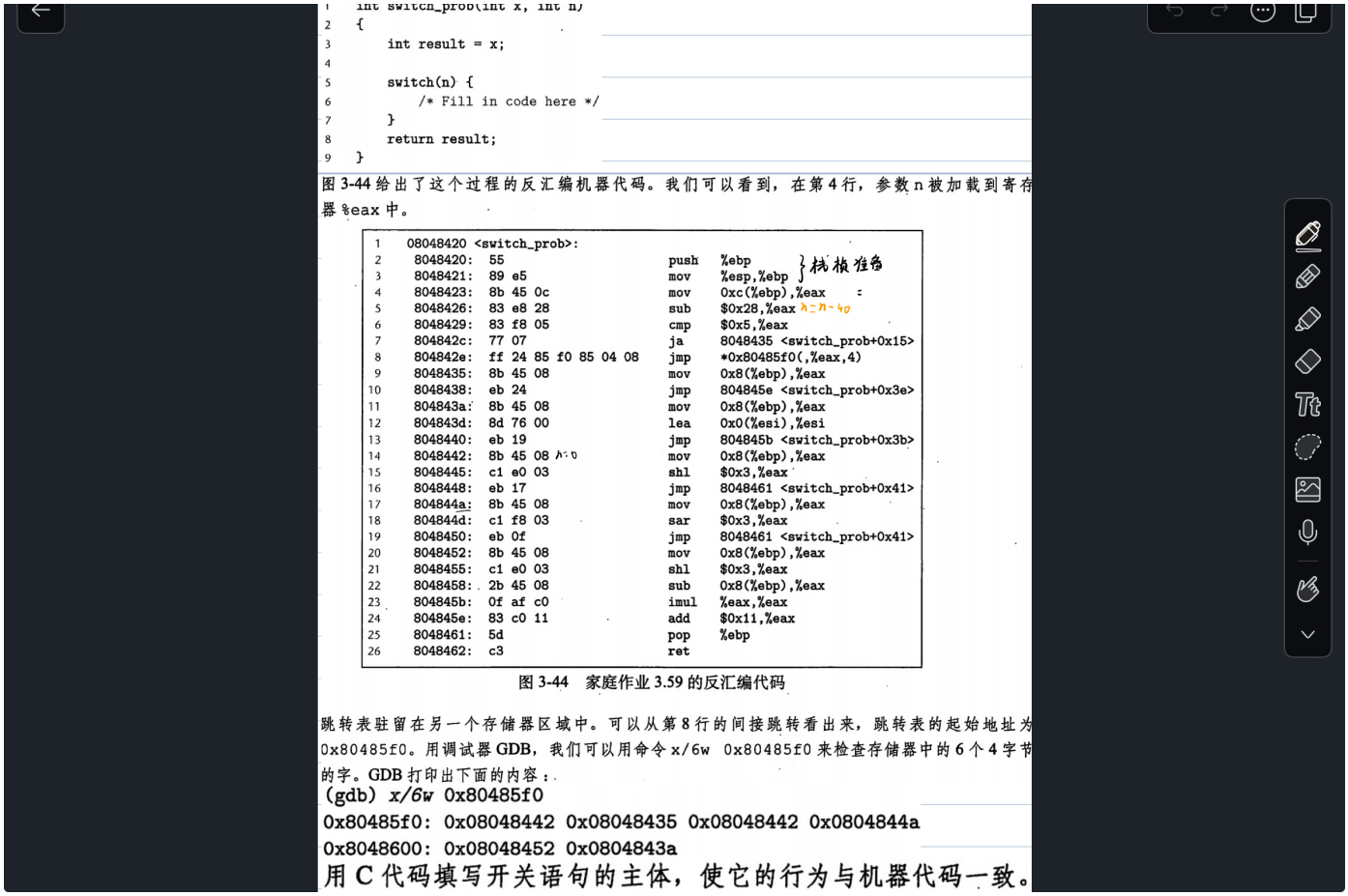
B. result的初始值为1431655765；mask的初始值为-2147483648

C. mask的测试条件为mask≠0

D. 每次循环mask都右移n位

E. 每次循环result与(x & mask)的结果异或

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
```

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器区域中。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。

由汇编代码可知0x80485f0到0x8048600地址存放的是switch语句的跳转表

```
sub $0x28,%eax
cmp $0x5,%eax
```

这两行代码是被测试的值(n - 40)，再用(n - 40) - 5与5比较，可知比较的值为40—45，再根据跳转表和汇编代码可得到switch语句主体如下：

```
int switch_prob (int x, int n){
  int result = x;
  switch (n){
    case 40:
      result <<= 3;
      break;
    case 41:
      result += 17;
      break;
    case 42:
      result <<= 3;
      break;
    case 43:
      result >>= 3;
      break;
    case 44:
      result = (result << 3) -x;
      result *= result;
```

```
      result += 17;
      break;
    case 45:
      result *= result;
      result += 17;
      break;
    default:
      result += 17;
      break;
  }
  return result;
}
```

3.66 你负责维护一个大型的C程序时，遇到下面这样的代码：

```
typedef struct {
    int left;
    a_struct a[CNT];
    int right;
} b_struct;

void test(int i, b_struct *bp)
{
    int n = bp->left + bp->right;
    a_struct *ap = &bp->a[i];
    ap->x[ap->idx] = n;
}
```

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。

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

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。

首先来看2-4行汇编代码：

```
push %ebp
mov %esp,%ebp
push %ebx
```

这三行是test函数的栈帧准备：保存调用函数的帧指针，建立函数栈帧，保存调用函数中%ebx中的值。

接下来第5行从%ebp偏移8个字节的地址中的值存入%eax中，根据函数调用过程，此处存放的值应是第一个参数i

```
mov 0x8(%ebp),%eax  /*  %eax = i */
```

第6行是将第二个参数bp保存到%ecx中

```
mov 0xc(%ebp),%ecx  /*  %exc = bp */
```

第7行将%eax中的值与立即数0xc1(28)相乘存入%ebx中

```
imul $0xc1,%eax,%ebx  /*  %ebx = %eax * 28 = 28i  */
```

8-9行计算 8 *%eax + 0值并存入到%edx中，再计算%edx - %eax将结果在保存到%edx中

```
lea 0x0(,%eax,8),%edx /*  %edx = 8 * %eax = 8i  */
sub %eax,%edx /*  %edx = %edx - %eax = 7i */
```

第10行计算(%ecx + %ebx + 4)地址中的值与%edx相加并将结果保存到edx中，%ecx保存的bp是结构体的起始地址，所以bp 是结构体中第一个变量left的地址，则bp + 4 + 28i 应是bp->a[i]的地址,可判断a_struct占28个字节，+7是为了后面得到ap->x[ap->idx]的地址

```
add 0x4(%ecx,%ebx,1),%edx /*  %edx = %ecx + %ebx + 4  + %edx = (bp + 28i + 4) + 7i  */
```

第11-12行将%exc + 200地址处的值保存到%eax中，即(bp + 200)地址处的值，然后将%ecx地址处的值，即结构体中第一个参数bp->left与%eax值相加保存到%eax中，可推断是计算n = bp->left + bp->right，所以(bp + 200)地址处保存的是 bp->right，则a[CNT]的字节大小为200 - 4 =196，**CNT = 196 / 28 = 7**。

```
mov 0xc8(%ecx),%eax
add (%exc),%eax   /* int n = bp->left + bp->right */
```

根据上述分析，%eax中保存的是n,%ecx中保存的*dp，%edx中保存的是*ap，是所以第13行执行的语句是ap->x[ap->idx] = n，汇编指令是将%eax中的值保存到地址 bp + 4 *((bp + 28i + 4) + 7i) + 8 中

```
mov %eax,0x8(%ecx,%edx,4)   /* bp + 4 *((bp + 28i + 4) + 7i) + 8 =
                               bp + 4 + 28i + 4 + 4*(bp + 28i + 4)
                                     ap->x[ap->idx] = n;
```

bp + 4 + 28i是ap结构体的起始地址，+4后应是跳过第一个变量idx的地址，此时是ap->x[0]的地址，说明idx是int型变量，再+4*(bp + 28i + 4)就是x[ap->idx]的地址，*4 说明 x[CNT] 是int型数组，而a_struct的大小为28字节，所以整型数组x的大小为24，即有7个元素

综上，a_struct的声明如下：

```
typedef struct {
  int idx;
  int x[6];
} a_struct
```