作业 4

3.24、3.25、3.28、3.30、3.31

3.64、3.67、3.68

练习题 3.24　对于如下 C 代码：

```
long loop_while(long a, long b)
{
    long result = _____;
    while (_____) {
        result = _____;
        a = _____;
    }
    return result;
}
```

以命令行选项-Og 运行 GCC 产生如下代码：

```
    long loop_while(long a, long b)
    a in %rdi, b in %rsi
1   loop_while:
2     movl    $1, %eax
3     jmp     .L2
4   .L3:
5     leaq    (%rdi,%rsi), %rdx
6     imulq   %rdx, %rax
7     addq    $1, %rdi
8   .L2:
9     cmpq    %rsi, %rdi
10    jl      .L3
11    rep; ret
```

```
long loop_while(long a, long b)
{
    long result = 1;
    while (a < b) {
        result = result * (a + b);
        a = a + 1;
    }
    return result;
}
```

练习题 3.25 对于如下 C 代码：

```
long loop_while2(long a, long b)
{
    long result = _____;
    while (_____) {
        result = _____;
        b = _____;
    }
    return result;
}
```

以命令行选项-O1 运行 GCC，产生如下代码：

```
    a in %rdi, b in %rsi
1   loop_while2:
2     testq   %rsi, %rsi
3     jle     .L8
4     movq    %rsi, %rax
5   .L7:
6     imulq   %rdi, %rax
7     subq    %rdi, %rsi
8     testq   %rsi, %rsi
9     jg      .L7
10    rep; ret
11  .L8:
12    movq    %rsi, %rax
13    ret
```

```
long loop_while2(long a, long b)
{
    long result = b;
    while (b > 0) {
        result = result * a;
        b = b - a;
    }
    return result;
}
```

练习题 3.28　函数 fun_b 有如下整体结构：

```
long fun_b(unsigned long x) {
    long val = 0;
    long i;
    for ( ... ; ... ; ... ) {
        .
        .
        .
    }
    return val;
}
```

GCC C 编译器产生如下汇编代码：

```
    long fun_b(unsigned long x)
    x in %rdi
1   fun_b:
2     movl    $64, %edx
3     movl    $0, %eax
4   .L10:
5     movq    %rdi, %rcx
6     andl    $1, %ecx
7     addq    %rax, %rax
8     orq     %rcx, %rax
9     shrq    %rdi            Shift right by 1
10    subq    $1, %rdx
11    jne     .L10
12    rep; ret
```

逆向工程这段代码的操作，然后完成下面的工作：

A. 根据汇编代码版本填写 C 代码中缺失的部分。

B. 解释循环前为什么没有初始测试也没有初始跳转到循环内部的测试部分。

C. 用自然语言描述这个函数是计算什么的。

A.

```
long fun_b(unsigned long x) {
    long val = 0;
    long I;
    for (i = 64; i != 0; i--) {
        val = (val << 1) | (x & 0x1);
        x >>= 1;
    }
    return val;
}
```

B. 这段代码用 guarded-do 生成，i 初始化为 64，i != 0 必然成立，不需要初始测试。

C. 将 x 中的位反过来，得到一个完全对称的位串。

练习题 3.30 下面的C函数省略了switch语句的主体。在C代码中，情况标号是不连续的，而有些情况有多个标号。

```
void switch2(long x, long *dest) {
    long val = 0;
    switch (x) {
        :
        Body of switch statement omitted
    }
    *dest = val;
}
```

在编译该函数时，GCC为程序的初始部分生成了以下汇编代码，变量x在寄存器%rdi中：

```
    void switch2(long x, long *dest)
    x in %rdi
1   switch2:
2     addq    $1, %%rdi
3     cmpq    $8, %rdi
4     ja      .L2
5     jmp     *.L4(,%rdi,8)
```

为跳转表生成以下代码：

```
1   .L4:
2     .quad   .L9
3     .quad   .L5
4     .quad   .L6
5     .quad   .L7
6     .quad   .L2
7     .quad   .L7
8     .quad   .L8
9     .quad   .L2
10    .quad   .L5
```

根据上述信息回答下列问题：

A. switch语句内情况标号的值分别是多少？

B. C代码中哪些情况有多个标号？

A. (x+1)超过（无符号）8 时跳转到 default，即 x+1 在-1 和 8 之间，又因为跳转表中 x+1 = 4 与 x+1 = 7 也跳转到 default，即 switch 语句内标号的值分别是-1,0,1,2,4,5 和 7。

B. 目标为.L5 的情况有标号 0 和 7，目标为.L7 的情况有标号 2 和 4。

练习题 3.31 对于一个通用结构的 C 函数 switcher：

```
void switcher(long a, long b, long c, long *dest)
{
    long val;
    switch(a) {
    case _________:       /* Case A */
        c = _________;
        /* Fall through */
    case _________:       /* Case B */
        val = _________;
        break;
    case _________:       /* Case C */
    case _________:       /* Case D */
        val = _________;
        break;
    case _________:       /* Case E */
        val = _________;
        break;
    default:
        val = _________;
    }
    *dest = val;
}
```

GCC 产生如图 3-24 所示的汇编代码和跳转表。

```
    void switcher(long a, long b, long c, long *dest)
    a in %rdi, b in %rsi, c in %rdx, dest in %rcx
 1  switcher:
 2    cmpq    $7, %rdi
 3    ja      .L2
 4    jmp     *.L4(,%rdi,8)
 5    .section        .rodata
 6  .L7:
 7    xorq    $15, %rsi
 8    movq    %rsi, %rdx
 9  .L3:
10    leaq    112(%rdx), %rdi
11    jmp     .L6
12  .L5:
13    leaq    (%rdx,%rsi), %rdi
14    salq    $2, %rdi
15    jmp     .L6
16  .L2:
17    movq    %rsi, %rdi
18  .L6:
19    movq    %rdi, (%rcx)
20    ret
```

a）代码

```
1  .L4:
2    .quad   .L3
3    .quad   .L2
4    .quad   .L5
5    .quad   .L2
6    .quad   .L6
7    .quad   .L7
8    .quad   .L2
9    .quad   .L5
```

b）跳转表

图 3-24 练习题 3.31 的汇编代码和跳转表

填写 C 代码中缺失的部分。除了情况标号 C 和 D 的顺序之外，将不同情况填入这个模板的方式是唯一的。

case 5: c = b ^ 15; case 0: val = c + 112; break; case 2: case 7: val = val = (b + c) << 2;
case 4: val = a; break; default: val = b;

** 3.64 考虑下面的源代码，这里 R、S 和 T 都是用#define声明的常数：

```
long A[R][S][T];

long store_ele(long i, long j, long k, long *dest)
{
    *dest = A[i][j][k];
    return sizeof(A);
}
```

在编译这个程序中，GCC产生下面的汇编代码：

```
long store_ele(long i, long j, long k, long *dest)
i in %rdi, j in %rsi, k in %rdx, dest in %rcx
store_ele:
  leaq    (%rsi,%rsi,2), %rax
  leaq    (%rsi,%rax,4), %rax
  movq    %rdi, %rsi
  salq    $6, %rsi
  addq    %rsi, %rdi
  addq    %rax, %rdi
  addq    %rdi, %rdx
  movq    A(,%rdx,8), %rax
  movq    %rax, (%rcx)
  movl    $3640, %eax
  ret
```

A. 将等式(3.1)从二维扩展到三维，提供数组元素 A[i][j][k]的位置的公式。

B. 运用你的逆向工程技术，根据汇编代码，确定 R、S 和 T 的值。

A. A[i][j][k]的地址为 A + (S * T * i + T * j + k) * 8

B. 观察汇编代码可得地址偏移量为(65 * i + 13 * j + k) * 8，
则 S * T = 65，T = 13，sizeof(A) = R * S * T * 8 = 3640，即 R = 7，S = 5，T = 13。

3.67 这个作业要查看 GCC 为参数和返回值中有结构的函数产生的代码，由此可以看到这些语言特性通常是如何实现的。

下面的 C 代码中有一个函数 process，它用结构作为参数和返回值，还有一个函数 eval，它调用 process：

```
typedef struct {
    long a[2];
    long *p;
} strA;

typedef struct {
    long u[2];
    long q;
} strB;

strB process(strA s) {
    strB r;
    r.u[0] = s.a[1];
    r.u[1] = s.a[0];
    r.q =    *s.p;
    return r;
}

long eval(long x, long y, long z) {
    strA s;
    s.a[0] = x;
    s.a[1] = y;
    s.p = &z;
    strB r = process(s);
    return r.u[0] + r.u[1] + r.q;
}
```

GCC 为这两个函数产生下面的代码：

```
strB process(strA s)
process:
  movq    %rdi, %rax
  movq    24(%rsp), %rdx
  movq    (%rdx), %rdx
  movq    16(%rsp), %rcx
  movq    %rcx, (%rdi)
  movq    8(%rsp), %rcx
  movq    %rcx, 8(%rdi)
  movq    %rdx, 16(%rdi)
  ret
```

```
long eval(long x, long y, long z)
x in %rdi, y in %rsi, z in %rdx
eval:
  subq    $104, %rsp
  movq    %rdx, 24(%rsp)
  leaq    24(%rsp), %rax
  movq    %rdi, (%rsp)
  movq    %rsi, 8(%rsp)
  movq    %rax, 16(%rsp)
  leaq    64(%rsp), %rdi
  call    process
  movq    72(%rsp), %rax
  addq    64(%rsp), %rax
  addq    80(%rsp), %rax
  addq    $104, %rsp
  ret
```

A. 从 eval 函数的第 2 行我们可以看到，它在栈上分配了 104 个字节。画出 eval 的栈帧，给出它在调用 process 前存储在栈上的值。

B. eval 调用 process 时传递了什么值？

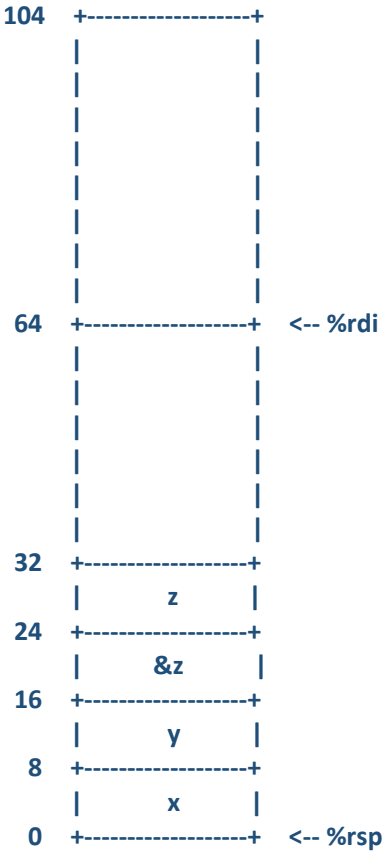
C. process 的代码是如何访问结构参数 s 的元素的？

D. process 的代码是如何设置结果结构 r 的字段的？

E. 完成 eval 的栈帧图，给出在从 process 返回后 eval 是如何访问结构 r 的元素的。

F. 就如何传递作为函数参数的结构以及如何返回作为函数结果的结构值，你可以看出什么通用的原则？

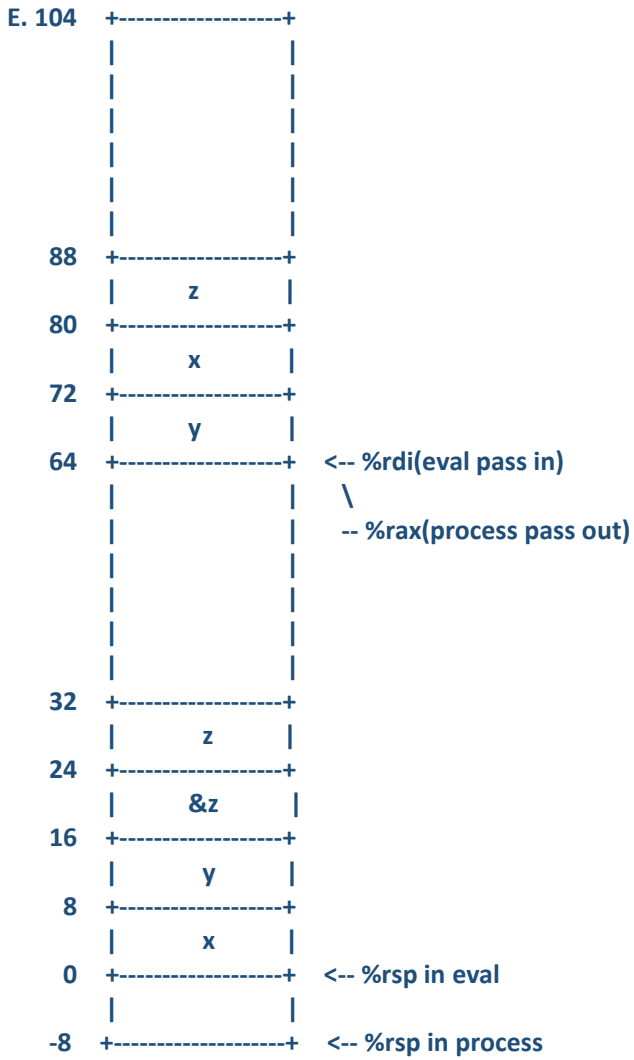
A. 104



B. eval 调用 process 时传递了地址%rsp + 64

C. process 的代码使用%rsp + 偏移量访问结构参数 s 的元素

D. eval 将地址%rsp + 64 传给 process，process 从该处开始存储数据，并返回这个地址



F. 函数被调用时，调用者将一个地址传给被调用者，被调用者从该处开始存储数据，并在函数调用结束时返回这个地址。

**3.68 在下面的代码中，A 和 B 是用# define定义的常数：

```
typedef struct {
    int x[A][B]; /* Unknown constants A and B */
    long y;
} str1;

typedef struct {
    char array[B];
    int t;
    short s[A];
    long u;
} str2;

void setVal(str1 *p, str2 *q) {
    long v1 = q->t;
    long v2 = q->u;
    p->y = v1+v2;
}
```

GCC 为 setVal 产生下面的代码：

```
void setVal(str1 *p, str2 *q)
p in %rdi, q in %rsi
setVal:
  movslq  8(%rsi), %rax
  addq    32(%rsi), %rax

  movq    %rax, 184(%rdi)
  ret
```

A 和 B 的值是多少？（答案是唯一的。）

movslq 8(%rsi), %rax，又 int 的长度为 4，则有 4 < B <= 8
addq 32(%rsi), %rax，又 long 的长度为 8，则 24 < 12 + A*2 <= 32
movq %rax, 184(%rdi)，又 long 的长度为 8，则 176 < A*B*4 <= 184
由 4 < B <= 8，5 < A <= 10，44 < A*B <= 46 解得 A = 9，B = 5。