2020 春第 3 章家庭作业

第二版教材，P202-212，家庭作业：3.54，3.56，3.58，3.62，3.65 题。

3.54 一个函数的原型为

int decode2 (int x, int y, int z);

将这个函数编译成 IA32 汇编代码。代码体如下：

```
    x at %ebp+8, y at %ebp+12, z at %ebp+16
1   movl    16(%ebp), %edx
2   subl    12(%ebp), %edx
3   movl    %edx, %eax
4   sall    $15, %eax
5   sarl    $15, %eax
6   xorl    8(%ebp), %edx
7   imull   %edx, %eax
```

参数 *x*、*y* 和 *z* 存放在存储器中相对于寄存器%ebp 中地址偏移量为 8、12 和 16 的地方。代码将返回值存放在寄存器%eax 中。

写出等价于上述汇编代码的 decode2 的 C 代码。

3.56 考虑下面的汇编代码：

```
    xat %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
```

以上代码是以下形式的 C 代码编译产生的：

```
1    int loop(int x, int n)
2    {
3        int   result = _____;
4        int 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，得到一个与上述汇编代码等价的完整 C 程序。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，并回答下述问题。

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。

3.58 下面代码是在一个开关语句中根据枚举类型值进行分支选择的例子。回忆一下，C 语言中枚举类型只是一种引入一组与整数值相对应的名字的方法。默认情况下，值是从 0 向上依次赋给名字的。在我们的代码中，省略了与各种情况标号相对应的动作。

```
/* Enumerated type creates set of constants numbered 0 and upward */
typedefenum {MODE_A, MODE_B, MODE_C, MODE_D, MODE_E} mode_t;

int switch3(int *p1, int *p2, mode_t action)
{
    int   result=0
    switch(action) {
    case      MODE_A:
    ……
```

```
    case    MODE_B:
    ……

    case    MODE_C:
    ……

    case    MODE_D:
    ……

    case    MODE_E:
    ……
    default;
    ……
    }
    return result
}
```

产生实现各个动作的汇编代码部分如下所示。这段代码实现了 switch 语句的各个分支，注释指明了参数位置，寄存器值，以及各个跳转目的地情况标号。寄存器%edx 对应于程序变量 *result*，并被初始化为-1。**填写 C 代码中缺失的部分**。注意那些会落入其他情况（default）中的情形。

```
    Arguments:  p1 at %ebp=8, p2 at %ebp=12, action at %ebp+16
    Registers:  result in %edx(initialized to -1)
    The jump targets:
1   .L17:                           MODE_E
2       movl    $17,%edx
3       jmp     .L19
4   .L13:                           MODE_A
5       movl    8(%ebp), %eax
6       movl    (%eax), %edx
7       movl    12(%ebpx), %ecx
8       movl    (%ecx), %eax
9       movl    8(%ebp), %ecx
10      movl    %eax, (%ecx)
11      jmp     .L19
12  .L14:                           MODE_B
13      movl    12(%ebp), %edx
14      movl    (%edx), eax
15      movl    %eax, %edx
16      movl    8(%ebp), %ecx
17      addl    (%ecx), %edx
18      movl    12(%ebp), %eax
19      movl    %edx, (%eax)
```

```
20      jmp     .L19
21  .L15                            MODE_C
22      movl    12(%ebp), %edx
23      movl    $15, (%edx)
24      movl    8(%ebp), %ecx
25      movl    (%ecx), %edx
26      jmp     .L19
27  .L16                            MODE_D
28      movl    8(%ebp), %edx
29      movl    (%edx), %eax
30      movl    12(%ebp), %ecx
31      movl    %eax, (%ecx)
32      movl    $17, %edx
33  .L19                            default
34      movl    %edx, %eax          set return value
```

3.62　下面的代码转置一个 $M \times M$ 矩阵的元素，这里 M 是一个用#define 定义的常数：

```
void transpose (int A[M][M]) {
    inti, j;
    for (i = 0; i< M; i++)
        for (j = 0; j <i; j++) {
            int t = A[i][j];
            A[i][j] = A[j][i];
            A[j][i] = t;
        }
}
```

当优化等级-O2 编译时，GCC 为这个函数的内循环产生下面的代码：

```
1   .L3
2   movl    (%ebx), %eax
3   movl    (%esi, %ecx, 4), %edx
4   movl    %eax, (%esi, %ecx, 4)
5   addl    $1, %ecx
6   movl    %edx, (%ebx)
7   addl    $76, %ebx
8   cmpl    %edi, %ecx
9   jl      .L3
```

A. *M* 的值是多少？

B. 哪个寄存器保存着程序值 *i* 和 *j*？

C. 重写 transpose 的一个 C 代码版本，使用在这个循环（上述汇编代码）中出现的优化思想。在你的代码中，使用参数 *M*，而不要用常数值。

3.65　在下面的代码中，*A* 和 *B* 是用#define 定义的常数：

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

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

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

GCC 为 setVal 的主体产生下面的代码：

```
1    movl    12(%ebp), %eax
2    movl    28(%eax), %edx
3    addl    8(%eax), %edx
4    movl    8(%ebp), %eax
5    movl    %edx, 44(%eax)
```

A 和 *B* 的值是多少？（答案是唯一的。）