第 3 章家庭作业

——201808010515　计科 1805 黄茂荣

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。

分析：第 1 句是将 z 的值存放在寄存器%edx 中（对应变量 temp1），第 2 句将 z 减去 y 再存放入%edx 中，第 3 句将%edx 中值移到到%eax 中(对应变量 temp2)，第 4,5 句是进行左右移位，移位后存储到寄存器%eax 中，6,7 句是对这些变量值进行最后的处理，x^(z-y)*temp2。

实现代码如下：

```
int decode2(int x ,int y, int z)
{
    int temp1 = z-y;
    int temp2 = temp1;
    temp2=(temp2<<15)>>15;
return (x^temp1)*temp2;
}
```

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。

分析：调用函数过程中，函数内传的参数会依次存放在%ebp+8,%ebp+12 的位置，由 1,2 句汇编可以知道 x,n 存放的存储器，3,4 句汇编是对函数内的局部变量初始化，从这里可以知道 result,mask 的初始值，.L2 是循环体内部执行的内容，第 6 句汇编是%eax=mask,第 7 句汇编将 mask&x 的结果存放在%eax 中，第 8 句执行 result^(x & mask),结果存放入%edi,第 11,12 句是要对 mask 的进行判定，根据 testl,jne 可以知道判定的条件是 mask 是否为 0。回到第 9,10 句，这里则是 对每次循环 mask 的进行修改，由于是 shrl,故进行逻辑右移操作，而 C 语言程序对 int 型默认为算术右移，故需要进行一个 int—>unsigned 的强制类型转换。最后一句汇编表示返回 result.

A. 哪个寄存器保存着程序值 x、n、result 和 mask?

寄存器%esi 存储 x,%ebx 存储 x,%edi 存储 reuslt,%edx 存储 mask。

B. result 和 mask 的初始值是什么?

result 的初始值为 1431655765；mask 的初始值为-2147483648。

C. mask 的测试条件是什么?

mask 的测试条件是 mask 不为 0.

D. mask 是如何被修改的?

mask 每次都是逻辑右移 n 位。

E. result 是如何被修改的?

result 每次与 mask 和 x 相与的结果异或，即 result^=(mask&x)。

F. 填写这段 C 代码中所有缺失的部分。

```
int loop(int x,int n)
{
   int result=1431655765;
   int mask;
   for(mask=-2147483648;mask!=0;mask=(unsigned)mask>>n)
   {
      result^=(mask&x);
   }
   return result;
}
```

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
```

分析：题目中给出的是 switch 语句的汇编代码，要求我们还原原 C 语言代码，根据每个跳转目的的情况标号，可以确定出每个 case 中的代码内容。这里需要注意的就是其余会落入.L19 的情况。题目中提到 result 会被初始化为-1，但整个汇编中没有对 result 的初始化，故考虑到最后的 default，当没有对 result 进行任何处理时，需要 result=-1，并返回给%eax，而进入 case 语句中处理过 result，直接将其返回即可，故 default 中代码为给 result 赋值为-1。

```
int switch3(int *p1, int *p2, mode_t action)
{
  int result=0;
  switch(action) {
  case MODE_A:
    result=*p1; *p1=*p2; break;
  case MODE_B:
    result=*p1+*p2; *p2=result; break;
  case MODE_C:
    *p2=15; result=*p1; break;
```

```
    case MODE_D:
      *p2=*p1; result=17; break;
    case MODE_E:
      result=17; break;
    default;
      result=-1;
    }
    return result;
}
```

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。

分析：由第 2 句代码可知，%ebx 中存储的是一个地址，表示将 A[i][j]->t(t 存储在寄存器%eax 中)，第 3 句代码是表示 A[j][i]->A[i][j], A[j][i]的索引采用比例变址寻址的方式，第 4 句是将 t->A[j][i],完成交换过程，第 5 句这里%ecx 加 1，对应的就是 j++,第 6 句将%ebx 对应的地址加上 76，表示是%ebx 存储的地址变为下一位置对应的地址，故 76=4*M，第 8,9 句汇编代码表示对内循环是否退出的条件判断，从这里可以确定%edi 中存储的是 i。

A. M 的值是多少？

M=76/4=19。

B. 哪个寄存器保存着程序值 i 和 j？

i 保存在寄存器%edi 中，j 保存在寄存器%ecx 中。

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。

```
int  transpose(int  M,  int  A[M][M])
{
    int  i,j;
    for(i=0;  i<M;  ++i)
    {
        int  *a =&A[i][0];//用指针 a 指向第 i 行
        int  *b =&A[0][i];//用指针 b 指向第 i 列
        for(j=0;  j<i;  ++j)
        {
                int  t=*a;
                *a =*b;
                *b = t;
                a++;
                b+= M;//跳转到下一列的位置
        }
    }
}
```

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。）

解答:

由汇编代码第 2 句可以知道调用的是 q->u,u 的首地址为 28（%eax）,第 3 句执行加法操作，将 q->t 与 q->u 相加之后存储到 q->u 中，这里可以知道 t 的首地址为 8（%eax）,两个地址相减得 t 所占字节和数组 s 所占字节为 20 个字节，故数组 s 所占 20-4=16 字节，即 2*B+C1=16,0<=C1<=4,且 B，C1 均为整数。

接着来看结构体 str1,str1 中最大占用字节数是 4，故是 4 字节对齐，根据汇编代码第 5 句得 A*（2B）+C2=44，0<=C2<=4，且 A,B,C2 为整数。

结合上述两个方程得 A=3，B=7。