2025计算机系统软件基础-第2次作业-张睿恒-2024302182001

$T_{3.58}$

```
long decode2(long x,long y,long z){//x in %rdi, y in %rsi, z in %rdx
    z-=y;
    z*=x;
    long ans=z;
    ans<<=63;
    ans>>=63;
    ans^=x;
    return ans;
}
```

$T_{3.59}$

先给出数学推导

> 不妨将x,y均以$64bit$的形式展开，即写成：
> $x=x_h\times 2^{64}+x_l,y=y_h\times 2^{64}+y_l$
> 乘开，有：
> $ans=(x_h\cdot y_h)\times 2^{128}+(x_h\cdot y_l+x_l\cdot y_H)\times 2^{64}+x_l\cdot y_l$
> 第一项溢出，故只需计算后两项

下面给出汇编代码的注释：

```
movq %rdx %rax # 将y拷贝到rax寄存器中
cqto # convert quatword to octoword ，即将rax寄存器的符号位copy到rdx中，使得rdx存y_h，
rax存y_l
movq %rsi %rcx # 将x拷贝到rcx寄存器中
sarq $63 %rcx # 提取出x的符号位，即x_h
imulq %rax %rcx # 将x_h和y_l相乘
imulq %rsi $rdx # 将x_l和y_h相乘
addq %rdx %rcx # 将刚才计算的两个数相加，即得到2^{64}前面的系数
movq %rax (%rdi) # 将运算结果的低64bit传到返回值（rdi寄存器存的地址）
movq %rdx 8(%rdi) # 将结果的高64bit传到返回值(rid寄存器的地址后面8个字节的地址)
```

$T_{3.61}$

```
long cread(long *xp){
    return (!xp?0:*(xp));
}
```

观察压缩包中的3.16.s文件，可以看到确实使用了条件赋值

$T_{3.62}$

这其实没啥好说的，直接上代码吧

```
typedef enum {MODE_A, MODE_B, MODE_C, MODE_D, MODE_E} mode_t;

long switch3(long *p1, long *p2, mode_t action){
    long result=0;
    switch (action){
       case MODE_A:
        result=*p2;
        *p2=*p1;
        break;
       case MODE_B:
        *p1=*p1+*p2;
        result=*p1;
        break;
       case MODE_C:
        *p1=59;
        result=*p2;
        break;
       case MODE_D:
        *p1=*p2;
        result=27;
        break;
       case MODE_E:
        result=27;
        break;
       default:
        result=12;
        break;
    }
    return result;
}
```

$T_{3.63}$

```
long switch_prob(long x, long n){
    long result=x;
    switch (n){
        case 60:
        case 62:
        result=8*x;
        break;
        case 63:
        result=x>>3;
        break;
        case 64:
        result=(x<<4)-x;
        case 65:
        result=x*x;
        default:
        result=x+0x4B;
    }
    return result;
}
```

$T_{3.65}$

A：由第六行可以看出：rdx寄存器每次加8（一个字节），所以是%rdx

B：同理，看第7行，一下挪120个bit，可以猜出是%rax

C：M=120/8=15

$T_{3.67}$

A：如下表所示：

相对于%rsp的偏移量	存储的值
+24	z
+16	&z
+8	y
+0(%rsp)	x

B：所有寄存器中的内容

C：通过访问栈顶指针%rsp加偏移量（字长）的方式寻址

D：通过访问寄存器%rdi加偏移量的方式。因为在eval函数里面%rdi已经被存为%rsp+64的一个值，相当于直接往栈帧的对应地址去写结果

E：如下表所示：

相对于%rsp的偏移量	存储的值
+80	z
+72	y
+64	x
+24	z
+16	&z
+8	y
+0(%rsp)	x

F：对于这种非标准数据类型，一般用某个寄存器存基址（如本题使用%rdi），然后用基址+偏移量的方式寻址，将结构体变量直接压在栈帧中进行传参or返回

$T_{3.69}$

A：0x120=288 可以看出，b_struct一共288个bits，除去first 和 last 一共还剩280个bits，是分配给a[CNT]的内存空间。后面%rax应该是用来访存的，看一下发现值是bp+40i，即(%rax)=*(bp+40i)。所以得出一个a_struct的大小是40个bits。综上所述，CNT=280/40=7

B：

```
typedef struct{
    long idx;
    long x[4];
} a_struct;
```

$T_{3.71}$

```
void good_echo(void){
    const int SIZE = 100;
    char buf[SIZE];
    while(1){
        char* p=fgets(buf,SIZE,stdin);
        if(p==NULL){
            break;
        }
        printf("%s",p);
    }
    return;
}
```