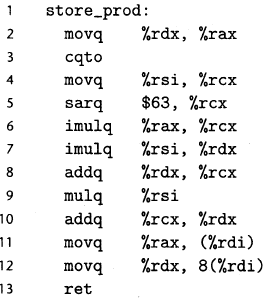
# 3.59

 1.

1. Rdx存的是y，所以Rax=yl
2. 使rdx和rax相关联，rdx成为rax的符号位扩展
3. Rsi存的是x，所以Rcx=xl
4. rcx向右算术移63位，得到x的高位，rcx = xh
5. rcx = yl \* xh
6. rdx = xl \* yh
7. rcx = yl \* xh + xl \* yh
8. 无符号全乘法，将存于rax的yl和存于rsi的xl相乘，得数的高位存rdx，低位存rax所以rdx:rax=zh:zl
9. Rdx=yl\*xh+yh\*xl+zh
10. 12. 将结果按次序存放

13．函数返回。

至此，返回rdx：rax，其中rdx等于yl\*xh+yh\*xl+zh，而rax等于xl和yl相乘的低位。至于xh和yh相乘必定溢出不用管。

# 3.61

long cread\_alt(long \*xp)

{

int t=0;

int \*p = xp ? xp : &t;

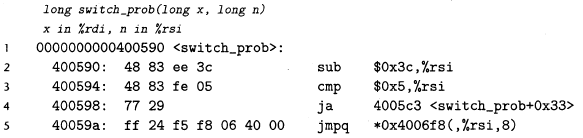
return \*p;

}

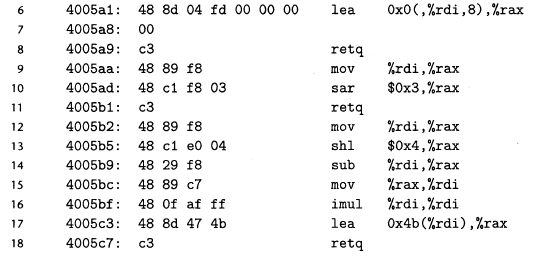
条件传送指令会对条件分别求值，如果没有判断是否为空指针就试图对其解引用，就会发生访问空指针的错误，所以在判断指针是否为空之前不能对其解引用。

于是我们直接对指针进行操作，并将其解引用后返回。将数值为0的变量取地址后赋值给指针，解引用后达到返回数字0的目的。

# 3.63



先将rsi减去3c也就是60，如果大于5就跳转到4005c3，然后根据0x4006f8+8\*（rsi-60）的值去跳转表中找地址就可以得到具体跳转的地方了





具体如下

当rsi=60或62，跳转表为4005a1，跳往第6行，返回8\*x

当rsi=63，跳转表为4005aa，跳往第10行，返回x>>3

当rsi=61或大于65，跳转表为4005c3，跳往第17行，返回x+0x4b

当rsi=65，跳转表为4005bf，跳往第16行，返回x平方

当rsi=64，跳转表为4005b2，跳往第12行，返回x<<4-x

switch(n):

{

case 60:

result = x \* 8;

break;

case 62:

result = x \* 8;

break;

case 63:

result = result >> 3;

break;

case 64:

result = (result << 4) - x;

x = result;

case 65:

x = x \* x;

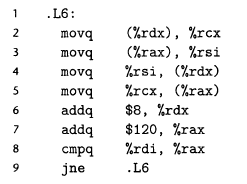
case 61:

default:

result = x + 0x4b;

}

# 3.65



1. 第六行中Rdx寄存器每次加8，数据又是long类型，A[i][j]指针每次移动一个，所以rdx指向的是A[i][j]
2. 第七行中Rax寄存器每次加120，A[j][i]指针每次移动一行，所以rax指向的是A[j][i]
3. 120/8=15所以M就是15

# 3.67

### A

|  |  |
| --- | --- |
| **对rsp的偏移量** | **大地址** |
| **Rsp+104** |  |
|  |  |
| **Rsp+64 rdi** |  |
|  |  |
| **Rsp+24** | **z** |
| **Rsp+16** | **&z** |
| **Rsp+8** | **y** |
| **Rsp** | **x** |
|  | **小地址** |

### B

传递的是64（%rsp）也就是%rsp+64，表示的栈空间

### C



如图，直接通过rsp+偏移量的方式来访问

### D



如图，process函数利用传入的%rsp+64作为结果的起始地址，从该处开始存放结构体r

### E

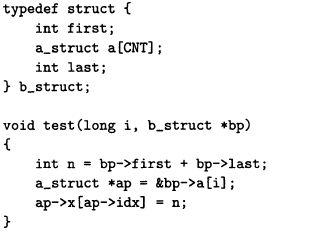
|  |  |
| --- | --- |
| **对rsp的偏移量** | **大地址** |
| **Rsp+80** | **z** |
| **Rsp+72** | **x** |
| **Rsp+64** | **y** |
|  |  |
| **Rsp+24** | **z** |
| **Rsp+16** | **&z** |
| **Rsp+8** | **y** |
| **Rsp** | **x** |
|  | **小地址** |

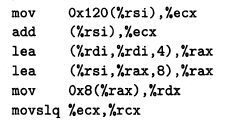
如图，还是直接通过rsp+偏移量的方式来访问

### F

调用者为被调用函数开辟栈空间，并将这一地址传给被调用函数，被调用函数执行操作后把数据从这一地址开始存放，执行结束后再返回该地址。

# 3.69



观察结构体我们发现，在int first成员和int last成员之间夹着一个结构体数组，大小未知，函数的第一句就出现了

bp->first和bp->last，这期间结构体指针bp的值是移动了0x120也就是288。说明first与a一共占了288个字节

这时n存于ecx

随后rax=i\*5 rax=40\*i+rsi，rdx=40\*i+rsi+8这是一个地址值所以不难猜出这是函数在访问&bp->a[i],这个+8是为了跳过第一个first成员，但是first成员是int类型只需要+4即可，为了对齐所以+8，所以我们知道a\_struct结构体第一个成员一定是8个字节，且总大小为40。

下一个movslq %ecx，%rcx 是对n进行类型转换，现在n是long类型了，所以结构体中的数组也是long类型数组。

显然0x10(%rax, %rdx, 8)是在访问ap->x[ap->idx]，

所以地址计算公式为bp + 16 + idx \* 8，此处+16是为了跳过first和idx变量，达到数组，证明了idx变量是结构体的第一个成员，且成员均为long类型。

### A

288个字节去掉first还有280，一个a\_struct结构体40,280/40=7

答案为7

### B

typedef struct {

long idx;

long x[4];

}