**作业二**

**1190201215 冯开来**

**3.59**

(\*dest in %rdi, x in %rsi, y in %rdx)

movq %rdx, %rax // 将y传给%rax

cqto // 将y有符号扩展到128位，并将低位给%rax，符号位高位存放在%rdx

movq %rsi, %rcx // 将x传给%rcx

sarq $63, %rcx // 将x算数右移63位，若x为正，%rcx为0，若x为负，%rcx为-1

imulq %rax, %rcx // 将y的低位与x的符号位相乘，即%rcx = xh \* yl

imulq %rsi, %rdx // 类似上一步，%rdx = xl \* yh

addq %rdx, %rcx // %rcx = xl \* yh + xh \* yl

mulq %rsi // 将%rax与%rsi相乘送到%rdx和%rax中，即yl \* xl

addq %rcx, %rdx // 将%rcx累加到%rdx中，可以理解为（xl\*yh+xh\*yl）\* 264 + yl\*xl的高位

movq %rax, (%rdi) // 将yl\*xl低64位送到（%rdi）即dest的低位

movq %rdx, 8(%rdi) // 将xl\*yh+xh\*yl送到dest的高位

ret // 返回

因为x = 264 \* xh + xl，y = 264 \* yh + yl，则x \* y =（264 \* xh + xl）\*（264 \* yh + yl）

= xhyh \* 2128 + （xh\*yl + xl\*yh）\* 264 + xl\*yl

这里面xhyh \* 2128必定溢出，截断之后全是0不用管。

所以我们最后要算的就是（xh\*yl + xl\*yh）\* 264 + xl\*yl，而这里264的意思就是这个二进制数的高64位，存在%rdx中，整个值的高64位是由xh\*yl + xl\*yh + xl\*yl>>64决定的。所以整个值的低位pl = xl \* xh的低64位，ph = xh\*yl + xl\*yh + xl\*yl>>64（xl\*yl高出64位的部分）。

**3.63**

long switch\_prob(long x, long n)

long result = x;

switch(n){

case 60:

case 62:

result = x \* 8;

break;

case 63:

result = x / 8;

break;

case 64:

result = 16 \* x – x;

x = result;

case 65:

x = x \* x;

default:

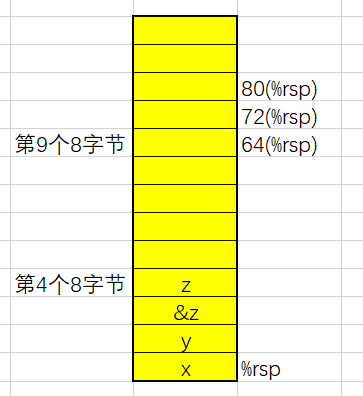
result = x + 75;

}

}

**3.67**

**A:**



**B:**

传递了64（%rsp），即栈指针rsp开始的第9个8字节的开始地址

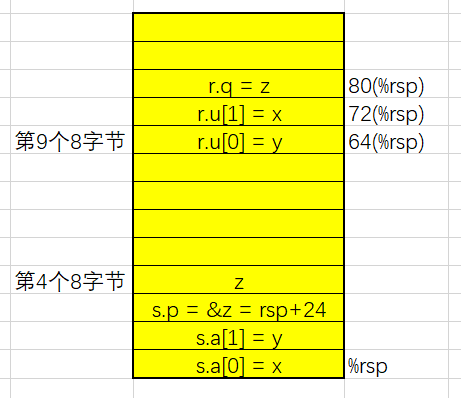
**C：**

因为这是通过堆栈传参，因为call指令导致返回地址压栈，rsp-8个字节，所以process是通过栈指针%rsp+8+偏移量来访问的

**D：**

process过程通过rdi寻址返回值结构体r。rdi寄存器中存放了返回值结构体r的起始地址，通过rdi寄存器加上偏移即可寻址返回值结构体的各个元素。

**E：**调用process后的栈帧



Process的返回值结构体r放在了rsp+64开始的内存处，通过这个地址+偏移量访存。

**F：**

结构体无论是作为函数参数还是作为返回值，都是通过内存传参，而不是通过寄存器。作为函数参数时，在子过程中通过rsp+偏移访问结构体参数的各个元素，作为返回值时，在调用子过程时通过rdi寄存器将返回值结构体在主过程中的存放的起始地址传给子过程。

**3.71**

#define SIZE 10

void good\_echo()

{

char str[SIZE];

while (1)

{

fgets(str, SIZE, stdin);

printf("%s", str);

if (str[strlen(str) - 1] == '\n')

{

break;

}

}

}

**3.75**

**A.**

对于第n个参数,则imag部分传%xmm(2n-1),real部分传%xmm(2n-2)

**B:**

imag部分返回值%xmm1, real部分返回值%xmm0.