3.59

store\_prod:

movq %rdx, %rax # %rax = y0.

cqto #cqto把%rax中的符号位扩展到%rdx中

movq %rsi, %rcx # %rcx = x0.

sarq $63, %rcx #将%rcx向右移63位,要么是-1,要么是0, %rcx = x1.

imulq %rax, %rcx # %rcx = y0 \* x1

imulq %rsi, %rdx # %rdx = x0 \* y1

addq %rdx, %rcx # %rcx = y0 \* x1 + x0 \* y1

mulq %rsi #无符号 x0\*y0,并将x0\*y0的128位结果的高位放在%rdx,低位放在%rax,因此这里%rdx = z1, %rax = z0.

addq %rcx, %rdx # %rdx = y0\*x1+x0\*y1+z1

movq %rax, (%rdi) #将%rax的值放到结果的低位

movq %rdx, 8(%rdi) #将%rdx的值放到结果的高位

ret

3.61

假如xp为空指针，那么这里产生对空指针读数据的操作，显然是不可以的。于是这里不能存在\*xp,可以用指针来代替，最后判断出xp的值之后，再进行读取数据，因此这里0也必须赋予一个地址，于是需要加个变量来存储0这个数字。

long cread\_alt(long \*xp)

{

int a=0;

int \*b=&a;

int \*p = xp ? xp : b;

return \*p;

}

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 = result >> 3;

break;

case 64:

result = (result << 4) - x;

x = result;

case 65:

x = x \* x;

case 61:

default:

result = x + 0x4b;

}

}

3.65

A：A[i][j]每次只移动一个单位，所以每次+8的寄存器%rdx就是指的A[i][j]。

B：另一个寄存器%rax是A[j][i]。

C：A[j][i]每次移动一行的距离，8 \* M = 120,显然，M=15。

3.67

A：

|  |  |
| --- | --- |
| 偏移量 | 值 |
| %rsp+24 | z |
| %rsp+16 | &z |
| %rsp+8 | y |
| %rsp | x |

B：%rsp+64表示的栈地址

C：通过%rsp+偏移量的栈地址来访问的s的值。

D：通过所传的表示栈地址的参数,来间接存储在栈上。

E：

|  |  |
| --- | --- |
| 相对于%rsp的偏移量 | 存储的值 |
| %rsp+80 | z |
| %rsp+72 | x |
| %rsp+64 | y |
| %rsp+32 |  |
| %rsp+24 | z |
| %rsp+16 | &z |
| %rsp+8 | y |
| %rsp | x |

F：在涉及结构体这种无法用一个寄存器存储的参数时,不管是传入还是返回,都是直接通过在栈上的存储来进行访问的。

3.69

A：CNT = (288 - 8) / 40 = 7

B：

typedef struct {

long idx;

long x[4];

}

3.71

void good\_echo()

{

char str[32];

int i;

char\* st = fgets(str, 32, stdin);

while(st != NULL)

{

for(i=0; str[i]; i++)

putchar(str[i]);

if(i < 31)

break;

st = fgets(str, 32, stdin);

}

return;

}