ICS Seminar Week4 Prep

余文凯 康子熙 赵廷昊 许珈铭

2023.10.11

Rules

remainder <- ordinal number in WeChat Group % 4

for all questions do

    if question number % 4 == remainder then

        you should work on it

    end

end

Q1

9. pushq %rbp 的行为等价于以下（）中的两条指令。

```
A. subq $8, %rsp     movq  %rbp,  (%rdx)
B. subq $8, %rsp     movq  %rbp,  (%rsp)
C. subq $8, %rsp     movq  %rax,  (%rsp)
D. subq $8, %rax     movq  %rbp,  (%rdx)
```

pushq指令的作用是向栈中压入数据

栈是从大地址向小地址拓展

1. 栈顶地址（%rsp）减8
2. 向栈顶地址对应的内存中存入数据。

B

Q2

6. 有如下代码段：

```
int func(int x, int y);
int (*p) (int a, int b);
p = func;
p(0,0);
```

对应的下列 x86-64 过程调用正确的是：

A. call *%rax　　B. call (%rax)

C. call *(%rax)　　D. call func

call Label

call *Operand

ACD均满足call的形式要求，本题中要写出p(0,0)的函数调用指令

由于p=func，函数指针p指向函数func，D选项正确

由于p=func语句使用func地址赋值指针p，所以指针p存放在寄存器中，调用函数p时通过寄存器中的值进行间接跳转而不是再次访存，A选项正确

A/D

Q3

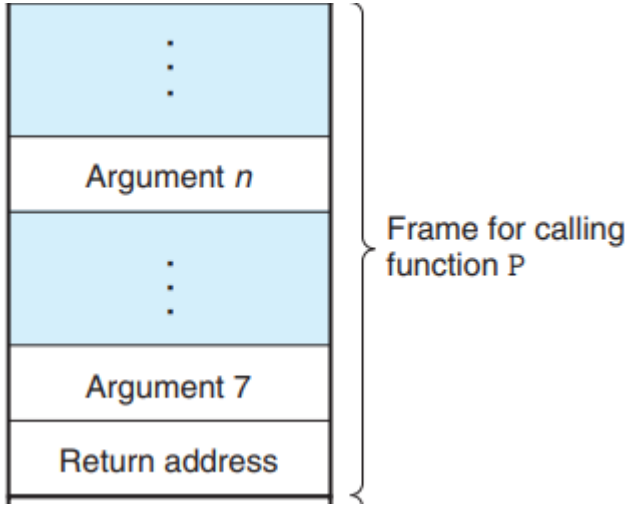
5. 已知函数 func 的参数超过 6 个。当 x86-64 机器执行完指令 call func 之后，%rsp 的值为 S。那么 func 的第 k(k > 6)个参数的存储地址是？

A. S + 8 * (k - 6)

B. S + 8 * (k - 7)

C. S - 8 * (k - 6)

D. S - 8 * (k - 7)



在call func后，压栈的不只有func所传的参数，同时还有函数的返回地址（call的下一条指令的地址）

参数传递是从右向左传递的，所以在栈上更靠前的参数应该存在地址更低的地方，且栈顶存储的是函数的返回地址

第7个参数应该储存在S + 8的地方，通过排除法，可以找到正确的答案。

A

Q4

4、下列关于 x86-64 过程调用的叙述中，哪一个是不正确的？

A．每次递归调用都会生成一个新的栈帧，空间开销大

B．当传递给被调用函数的参数少于 6 个时，可以通过通用寄存器传递

C．被调用函数要为局部变量分配空间，返回时无需释放这些空间

D．过程调用返回时，向程序计数器中推送的地址是调用函数中调用指令的下一条指令的地址

A正确，递归调用和普通函数调用行为一致，除了每次调用的都是同一个函数

B正确，前6个参数用寄存器，第7个参数以上通过栈传递

#扩展

寄存器传参不涉及地址读写，效率更高。寄存器只传递前6个参数是参数传送寄存器和 callee-saved 寄存器的 trade-off

C错误，局部变量作用域为函数内部，函数返回则作用域结束，函数栈帧释放

D正确，函数返回时，程序计数器中的地址是将要执行的指令，因此是call的下一条指令

C

Q5

7. 考虑以下 C 语言变量声明：

```
int *(*f[3])();
```

那么在一台 x86-64 机器上，sizeof(f)和 sizeof(*f)的值是多少？

A. 8 24

B. 24 8

C. 8 8

D. 8 不确定

该指针外层是函数指针，内层是指针数组
整个变量代表了一个函数指针的数组，且这些函数都不接受任何参数并返回一个int

sizeof(f)应当是指针数组的大小，即8 * 3 = 24

而sizeof(*f)是取数组的第一个元素，指的是一个函数指针的内存开销，也就是8个字节。

B

Q6

7. 有A的定义：int A[3][2] = {{1,2}, {3,3}, {2,1}};
那么A[2]的值为：

A. &A+16　　B. A+16　　C. *A+4　　D. *A+2

选项A，在C++中，引用的本质是数组，所以可以将&A理解成int (*)A[3][2],即是一个指向高维数组的指针。偏移单位是二维数组整体，&A + 16会偏移16 * 24个字节，对应地址应该为(A[3][2])[16]
选项B，A可以指代二维数组的第一个一维数组，偏移单位是一维数组， A + 16会偏移16 * 8个字节，对应地址应该为A[16]
选项C，首先判断优先级，*的优先级高于加法，*A等价于指向A[0][0]的指针，偏移单位是一个int，*A + 4会偏移4 * 4个字节，对应地址应该为A[2][0]的地址，与A[2]的值相同
选项D，与选项C同理，对应地址应该为A[1][0]的地址，与A[1]的值相同

C

Q7

3、假定静态 int 型二维数组 a 和指针数组 pa 的声明如下：

```
static int a[4][4]={ {3, 8, -2, 6}, {2, 1, -5, 3 }, {1, 18,
4, 10},{4, -2, 0, 8}};
static int *pa[4]={a[0], a[1], a[2], a[3]};
```

若 a 的首地址为 0x601080，则&pa[0]和 pa[1]分别是：

A. 0x6010c0、0x601090

B. 0x6010e0、0x601090

C. 0x6010c0、0x6010a0

D. 0x6010e0、0x6010a0

&pa[0] = pa 是指向pa[0]的指针，注意到pa是在a之后定义的变量，a的总长度为4 * 4 * 4字节，故pa的初始地址应当是0x6010c0

pa[1] = a[1] 是指针数组pa的第一个元素，存放的是一个指向int型一维数组的指针，所以pa[1]应当是a[1]对应的地址，也就是0x601090

A

Q8

7.阅读下列 C 代码和在 x86-64 机器上得到的汇编代码：

```
int a[__A__][__B__];
for (int i = 0; i < __C__; i++)
  a[i][__C__ - i] = 1;


  leaq 40(%rdi), %rax
  addq $440, %rdi
.L2:
  movl $1, (%rax)
  addq $40, %rax
  cmpq %rdi, %rax
  jne  .L2
```

假设 a 的地址初始时放在%rdi 中，假设程序正常运行且没有发生越界问题，则 C 代码中的 A、B、C 处应分别填：

A. 10、11、10

B. 9、11、9

C. 11、10、10

D. 11、11、11

由addq $40, %rax知a[i+1][c-i-1]地址比a[i][c-i]地址大40，即4 * (b – 1) = 40，b = 11

由leaq 40(%rdi), %rax及addq $40, %rax，可知在jne .L2控制跳转时%rax初始值为a + 80;
由addq $440, %rdi及cmpq %rdi, %rax，可知在jne .L2控制跳转时%rax终止值为a + 440。
共循环9次，每次+40，c=10

最后一次进入for循环i = 9，对应a[9][2]。由于没有发生越界问题因此A >= 9

A

Q9

6、有如下定义的结构，在 x86-64 下，下述结论中错误的是？

```
struct {
    char c;
    union {
        char vc;
        double value;
        int vi;
    } u;
    int i;
} sa;
```

A. sizeof(sa) == 24

B. (&sa.i - &sa.u.vi) == 8

C. (&sa.u.vc - &sa.c) == 8

D. 优化成员变量的顺序，可以做到"sizeof(sa) == 16"

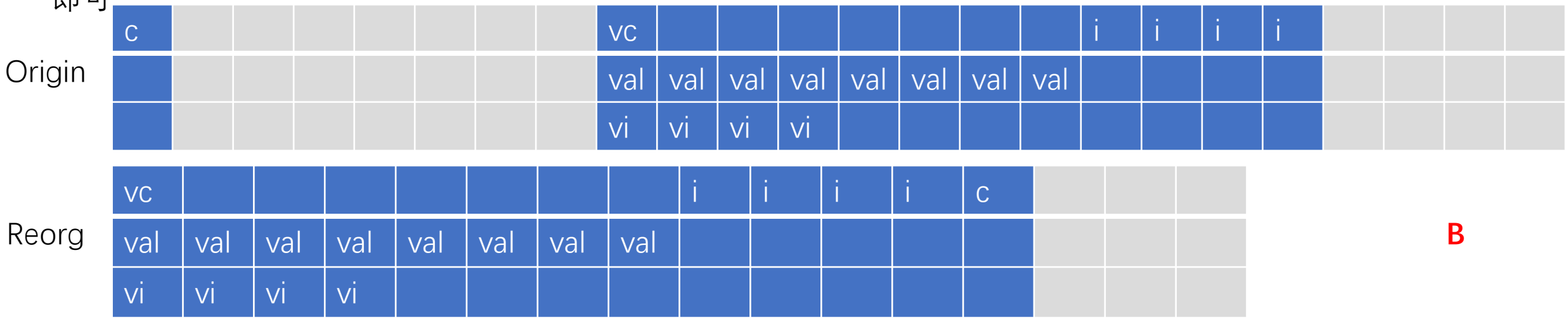
答：（　　）

A：如图Origin，sizeof（sa）=24，因为必须满足double value的对齐要求,最后必须有四字节空位（考虑声明连续两个sa的情形）

B：两个同类型指针相减，是地址之差除以变量大小，答案为2

C：同B知答案正确

D：将成员按对齐要求从大到小/从小到大排序可以使得结构体size最小，本题以u、i、c顺序排列即可



B

Q10

5、假设结构体类型 student_info 的声明如下：

```
struct student_info {
        char id[8];
        char name[16];
        unsigned zip;
        char address[50];
        char phone[20];
}x;
```

若 x 的首地址在%rdx 中，则“unsigned xzip=x.zip;”所对应的汇编指令为：

A. movl 0x24(%rdx), %rax

B. movl 0x18(%rdx), %rax

C. leal 0x24(%rdx), %rax

D. leal 0x18(%rdx), %rax

x.zip相对于x首地址偏移量为24（char型1字节对齐），注意24在16进制中是0x18

%rdx中存的是x的首地址，0x18（%rdx）存的是x.zip地址

应从内存中读取数据，所以使用movl而非leal。

B

Q11

对齐如图所示，注意第一问最后留下四个空字节且无法被使用，以保证声明多个A的时候第二个A内部的long能够对齐

14. 在 x86-64、Linux 操作系统下有如下 C 定义：

```
struct A {
    char CC1[6];
    int II1;
    long LL1;
    char CC2[10];
    long LL2;
    int II2;
};
```

(1) sizeof(A) = _____字节。

(2) 将 A 重排后，令结构体尽可能小，那么得到的新的结构体大小为_____字节。

Origin

cc10	cc11	cc12	cc13	cc14	cc15			ii1	ii1	ii1	ii1		
		II1	II1	II1	II1	II1	II1	II1	II1	cc20	cc21	cc22	cc23
cc24	cc25	cc26	cc27	cc28	cc29							II2	II2
II2	II2	II2	II2	II2	II2	ii2	ii2	ii2	ii2				

Reorg

II1	II1	II1	II1	II1	II1	II1	II1	II2	II2
II2	II2	II2	II2	II2	II2	ii1	ii1	ii1	ii1
ii2	ii2	ii2	ii2	cc10	cc11	cc12	cc13	cc14	cc15
cc20	cc21	cc22	cc23	cc24	cc25	cc26	cc27	cc28	cc29

56
40

Q12

5. 以下代码的输出结果是

```
union {
    double d;
    struct {
        int i;
        char c[4];
    } s;
} u;
u.d = 1;
printf("%d\n", u.s.c[2]);
```

A) 0　　B) -16　　C) 240　　D) 191

在小端法机器中，低地址存储u.d低位，高地址存储高位

double共64位，c[2]存储的应该是从高向低9~16位，（最高位算第一位），而1的双精度浮点表示为0x3ff0000000000000，c[2]表示0xf0，即-16。

B

Q13

1. 考虑如下代码在 x86-64 处理器上的运行情况：

```
union U{
  char x[12];
  char *p;
}u;

int main() {
  strcpy(u.x, "I love ICS!");
  printf("%p\n", u.p);
  return 0;
}
```

提示：

1) "I love ICS!"对应的字节序列是 49 20 6c 6f 76 65 20 49 43 53 21

2) %p 用于输出一个指针的值

程序运行的输出是：

A. 0x49206c6f76652049435321

B. 0x2153434920657666c2049

C. 0x49206c6f76652049

D. 0x492065766f6c2049

char*是8个字节，A、B显然不对

由于字符串存储方式是先出现的字符在较低字节，也就是x[0]~x[7]依次对应49、20、6c、6f、76、65、20、49，由于小端法存储，x[7]~x[0]依次表示p的高位到低位，故答案选D

D

Q14

2. 函数 g 定义如下：

```
int g(float f){
  union {
    float f;
    int i;
  } x;
  x.f = f;
  return x.i ^ ((1u << (x.i < 0 ? 31 : 0)) - 1);
}
```

则对于两个 float 型变量 a,b，有 a<b 是 g(a)<g(b) 的：

A. 充分必要条件

B. 充分不必要条件

C. 必要不充分条件

D. 不必要不充分条件

最后一句是说如果x.i非负，直接返回x.i，反之符号位不变，其余位取反。即非负数返回非负数，负数返回负数

若a<b，a、b同非负或a为负b为正，由上述分析f(a)<f(b)

下考虑两者同为负，对于float型变量，a<b说明除去符号位剩下部分a比b大，取反之后除去符号位部分g(a)比g(b)小，对于整型的g(a)、g(b),这说明g(a)<g(b),因此是充分条件

反之考虑a=-0.0，b=+0.0，有g(a)<g(b)但是a=b，因此不为必要条件，选B

B

Q15

6. 下列关于C语言中的结构体(struct)以及联合(union)的说法中，正确的是：

A) 对于任意 struct，将其成员按照其实际占用内存大小从小到大的顺序进行排列不一定会使之内存占用最小

B) 对于任意 struct，将其成员按照其实际占用内存大小从小到大的顺序进行排列一定不会使之内存占用最大

C) 对于任意 union，将其成员按照其实际占用内存大小从小到大的顺序进行排列不一定会使之内存占用最小

D) 对于任意 union，将其成员按照其实际占用内存大小从小到大的顺序进行排列一定不会使之内存占用最大

对于union来说，其大小确定，与成员排列顺序无关，c、d错误。

考虑只有一种数据类型成员的结构体，无论其怎么排列，大小不变，即达到了最大\最小值，故b错误。

对于struct，应当将成员按照其对其要求从小到大(或从大到小) 的顺序进行排列才能使内存占用最小，a正确

A

Q16

1. (10 分) 在 x86-64、LINUX 操作系统下，考虑如下的 C 定义：

```
typedef union {
    char c[7];
    short h;
} union_e;
typedef struct {
    char d[3];
    union_e u;
    int i;
} struct_e;
struct_e s;
```

回答如下问题：

(1) s.u.c 的首地址相对于 s 的首地址的偏移量是_____字节。

(2) sizeof(union_e) = _____字节。

(3) s.i 的首地址相对于 s 的首地址的偏移量是_____字节。

(4) 若将i 的类型改成short、将h 的类型改成int，那么sizeof(union_e) = _____字节，sizeof(struct_e) = _____字节。

struct_e 的结构如下图Origin：

(1) 4. union_e中成员为char[]和short，2字节对齐

(2) 8. sizeof(short) = 2，实际大小7字节，2字节对齐

(3) 12. 由(1) s.u偏移4字节，sizeof(union_e) = 8

(4) 8, 16. 如下图(4)，分析方法同上

Origin

d0	d1	d2		c0	c1	c2	c3	c4	c5	c6		i	i	i	i
				h	h										

(4)

d0	d1	d2		c0	c1	c2	c3	c4	c5	c6		i	i		
				h	h	h	h								

Q17

第三题（20 分）

1.考虑下面的union的声明，回答后面的问题。

```
union ELE {
    struct {
        int x;
        int *p;
    }e1;
    struct {
        union ELE * next;
        int y;
    }e2;
};
```

(注：32位机器)

这个union总共大小为多少____字节。

2.假设编译器为process的主体产生了如下了代码，请补充完整下面的过程：（只有一个不需要任何强制类型转换且不违反任何类型限制的答案）

```
    movl 8(%ebp),%eax
    movl (%eax),%ecx
    movl 4(%ecx),%edx
    movl (%edx),%edx
    subl 4(%eax),%edx
    movl %edx,(%ecx)
```

```
void process(union ELE * up)
{
    up->_____=_____- _____;
}
```

Q17

第三题（20 分）

1.考虑下面的union的声明，回答后面的问题。

```
union ELE {
    struct {
        int x;
        int *p;
    }e1;
    struct {
        union ELE * next;
        int y;
    }e2;
};
```

(注：32位机器)

这个union总共大小为多少____字节。

2.假设编译器为process的主体产生了如下了代码，请补充完整下面的过程：（只有一个不需要任何强制类型转换且不违反任何类型限制的答案）

```
    movl 8(%ebp),%eax
    movl (%eax),%ecx
    movl 4(%ecx),%edx
    movl (%edx),%edx
    subl 4(%eax),%edx
    movl %edx,(%ecx)

void process(union ELE * up)
{
    up->_____=_____- _____;
}
```

8

up->next->x=*(up->next->p) - (up->y)

e1	x	x	x	x	p	p	p	p
e2	next	next	next	next	y	y	y	y

- 32位机器，指针大小位32bit即4字节
- 对齐规则，x字节的变量存储地址必然为x的倍数
- Int为4字节，故sizeof(e1)=sizeof(e2)=4+4=8 不需要额外对齐
- union大小为元素中最大的，即8字节

- 32位机中bp(Base Point)为基指针寄存器，访问栈中数据
- 第一行为从栈中调出参数，eax%中存储的即为指针up
- 第二行对%eax存储的地址的前4字节取值，那么前4字节为一个指针，*up内容是e2，%ecx内容为next指针
- 第三行对(%ecx+4)进行取值到%edx，结合第四行对%edx取值判断(%ecx+4)为一个地址，于是推断next指针的ELE内容是e1，%edx最终为p指向元素的值
- 将各个寄存器中的内容带入即可得出表达式

Q18

第三题（20 分）

1.考虑下面的union的声明，回答后面的问题。

```
union ELE {
    struct {
        int x;
        int *p;
    }e1;
    struct {
        union ELE * next;
        int y;
    }e2;
};
```

(注：32位机器)

这个union总共大小为多少____字节。

4.阅读下列代码，回答后面的问题

```
typedef struct {
    short x[A][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;
}
```

(short 以 2 字节计算)

GCC 为 setVal 的主体产生下面的代码：

```
    movl 12(%ebp),%eax
    movl 28(%eax),%edx
    addl 8(%eax),%edx
    movl 8(%ebp),%eax
    movl %edx,44(%eax)
```

请直接写出A和B的值各是多少？

Q18

第三题（20 分）

1.考虑下面的union的声明，回答后面的问题。

```
union ELE {
    struct {
        int x;
        int *p;
    }e1;
    struct {
        union ELE * next;
        int y;
    }e2;
};
```

(注：32位机器)

这个union总共大小为多少____字节。

8

4.阅读下列代码，回答后面的问题

```
typedef struct {
    short x[A][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;
}
```

(short 以 2 字节计算)

GCC 为 setVal 的主体产生下面的代码：

```
    movl 12(%ebp),%eax
    movl 28(%eax),%edx
    addl 8(%eax),%edx
    movl 8(%ebp),%eax
    movl %edx,44(%eax)
```

请直接写出A和B的值各是多少？

A=3, B=7

e1	x	x	x	x	p	p	p	p
e2	next	next	next	next	y	y	y	y

- 32位机器，指针大小位32bit即4字节
- 对齐规则，x字节的变量存储地址必然为x的倍数
- Int为4字节，故sizeof(e1)=sizeof(e2)=4+4=8 不需要额外对齐
- union大小为元素中最大的，即8字节

- 传入参数为指针，参数p与q存储在栈上，且p比q更靠近栈顶
- 由addl可判断8(%eax)与%edx（即28(%eax)）存储的是t和u，可推断%eax存储的是指针q
- t与u起始地址相差20字节，相隔一个int与B个short，且最终对齐为int4字节，可推断B为7或8
- 第四行8(%ebp)推断为取出栈中参数p
- 由最后一行判断44(%eax)存储的是y，short s[A][B]共44字节，且按照int对齐，A*B=21或22
- 故B为7，A为3

Q19

第三题（15 分）

数据结构定义如下：（测试用机：64 位 Linux 机器）

```
typedef struct s1 {
        char cc[N];
        int ii[N];
        int *ip;
} S1;

S1 t1[N];
```

1、计算该数据结构的空间：

当 N=3 时，sizeof(S1) = (1)_____，sizeof(t1) = (2)_____

当 N=4 时，sizeof(S1) = (3)_____，sizeof(t1) = (4)_____

当 N=5 时，sizeof(S1) = (5)_____，sizeof(t1) = (6)_____

3、当 N=3 时，函数 fun 的汇编代码如下：

```
fun:
        movslq  %esi, %rax
        movslq  %edi, %rdi
        leaq    (%rdi,%rdi), %rdx
        leaq    (%rdx,%rdi), %r8
        leaq    (%r8,%r8), %rcx
        addq    %rcx, %rax
        movl    %esi, t1+4(,%rax,4)
        addq    %rdx, %rdi
        leaq    0(,%rdi,8), %rax
        movq    t1+16(%rax), %rax
        movl    %esi, (%rax)
        ret
```

根据上述代码，填写函数 fun 的 C 语言代码：

```
void fun(int x, int y)
{
        (1)_____ = (2)_____;
        (3)_____ = (4)_____;
}
```

Q19

第三题（15 分）

数据结构定义如下：（测试用机：64 位 Linux 机器）

```
typedef struct s1 {
        char cc[N];
        int ii[N];
        int *ip;
} S1;

S1 t1[N];
```

1、计算该数据结构的空间：

当 N=3 时，sizeof(S1) = (1)_____，sizeof(t1) = (2)_____

当 N=4 时，sizeof(S1) = (3)_____，sizeof(t1) = (4)_____

当 N=5 时，sizeof(S1) = (5)_____，sizeof(t1) = (6)_____

3、当 N=3 时，函数 fun 的汇编代码如下：

```
fun:
        movslq  %esi, %rax
        movslq  %edi, %rdi
        leaq    (%rdi,%rdi), %rdx
        leaq    (%rdx,%rdi), %r8
        leaq    (%r8,%r8), %rcx
        addq    %rcx, %rax
        movl    %esi, t1+4(,%rax,4)
        addq    %rdx, %rdi
        leaq    0(,%rdi,8), %rax
        movq    t1+16(%rax), %rax
        movl    %esi, (%rax)
        ret
```

根据上述代码，填写函数 fun 的 C 语言代码：

```
void fun(int x, int y)
{
        (1)_____ = (2)_____;
        (3)_____ = (4)_____;
}
```

24 72

32 128

40 200

t1[x].ii[y] = y

*(t1[x].ip) = y

- 64位指针为8字节
- N=3

cc0	cc1	cc3		ii0	ii0	ii0	ii0
ii1	ii1	ii1	ii1	ii2	ii2	ii2	ii2
ip	ip	ip	ip	ip	ip	ip	ip

- N=5

cc0	cc1	cc3	cc4	cc5			
ii0	ii0	ii0	ii0	ii1	ii1	ii1	ii1
ii2	ii2	ii2	ii2	ii3	ii3	ii3	ii3
ii4	ii4	ii4	ii4				
ip	ip	ip	ip	ip	ip	ip	ip

N=4

cc0	cc1	cc3	cc4	ii0	ii0	ii0	ii0
ii1	ii1	ii1	ii1	ii2	ii2	ii2	ii2
ii3	ii3	ii3	ii3				
ip	ip	ip	ip	ip	ip	ip	ip

- %edi值为x，%esi值为y
- 第6行addq后%rax中的值为6x + y，第7行(t1 + 24x + 4 + 4y)即为t1[x].ii[y]的值，将其赋值为y
- 第9行leaq后%rax中的值为24x，第10行(t1 + 24x + 16)即为t1[x].ip的值，第11行将y存入t1[x].ip取地址的空间中

Q20

第一段代码：

```
#include<stdio.h>

typedef struct
{
    int x;
    int y;
}struct_e;

typedef union
{
    struct_e s;
    double d;
}union_e;

int main () {
    union_e test;
    test.s.x = 0;
    test.s.y = -1;
    printf("%lf\n", test.d);

    test.s.y = 0x3ff00000;
    printf("%lf\n", test.d);
}
```

（1）以上程序的两次输出分别是 _____， _____。

（2） sizeof(union_e) = _____。

（3）如果某次程序输出的 test.d 是最大的负非规格化数，则

此时从 test 的起始地址开始的 8 个字节（用 16 进制表示）依次是：

此时

test.s.x = _____

test.s.y = _____

Q20

第一段代码：

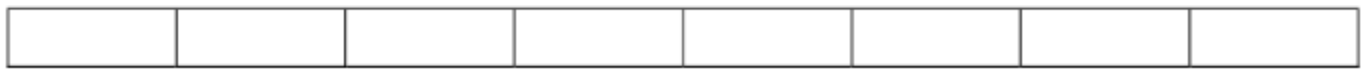
```
#include<stdio.h>

typedef struct
{
    int x;
    int y;
}struct_e;

typedef union
{
    struct_e s;
    double d;
}union_e;

int main () {
    union_e test;
    test.s.x = 0;
    test.s.y = -1;
    printf("%lf\n", test.d);

    test.s.y = 0x3ff00000;
    printf("%lf\n", test.d);
}
```

（1）以上程序的两次输出分别是 _____， _____。

（2） sizeof(union_e) = _____。

（3）如果某次程序输出的 test.d 是最大的负非规格化数，则

此时从 test 的起始地址开始的 8 个字节（用 16 进制表示）依次是：



此时

test.s.x = _____

test.s.y = _____

NAN 1

8

01 00 00 00 00 00 00 80

1

-2147483647-1 (-2^31)

union_e

x	x	x	x	y	y	y	y
d	d	d	d	d	d	d	d

- 小端法机器，在读double的x与y顺序应该反过来
- 第一次输出double符号位均为1但frac位不为0
- 最大的非规格化数为0b1000…0001，注意小端法机器的表示