一. 补充习题

1. 对于下列四个函数，假设 gcc 开了编译优化，判断 gcc 是否会将其编译为条件传送

```
long f1(long a, long b) {
    return (++a > --b) ? a : b;
}
```

```
long f2(long *a, long *b) {
    return (*a > *b) ? --(*a) : (*b)--;
}
```

```
long f3(long *a, long *b) {
    return a ? *a : (b ? *b : 0);
}
```

```
long f4(long a, long b) {
    return (a > b) ? a++ : ++b;
}
```

答： f1 由于比较前计算出的a 与b 就是条件传送的目标，因此会被编译成条件传送；f2 由于比较结果会导致a 与b 指向的元素发生不同的改变，因此会被编译成条件跳转；f3 由于指针a 可能无效，因此会被编译为条件跳转；f4 会被编译成条件传送，注意到a 和b 都是局部变量，return 的时候对a 和b 的操作都是没有用的。使用-O1 可以验证 gcc 的行为。

2. 根据汇编指令补充机器码中缺失的字节。

```
loop:
4004d0: 48 89 f8        mov %rdi, %rax
4004d3: eb __           jmp 4004d8 <loop+0x8>
4004d5: 48 d1 f8        sar %rax
4004d8: 48 85 c0        test %rax, %rax
4004db: 7f __           jg 4004d5 <loop+0x5>
4004dd: f3 c3           repz retq
```

答:03; f8. 程序进行到第五行时,开始跳转,跳转位置为0xf8(-8) + 0x4004dd = 0x4004d5,注意当程序走到第五行进行跳转之前，PC 指向该指令的下一条指令，即第六行 repz 的开头

二. 结构体与联合体

1. 在下面的代码中，A 和B 是用#define 定义的常数：

```
typedef struct {int x[A][B]; long y;} str1;
typedef struct {char array[B]; int t; short s[A]; long u;} str2;
void setVal(str1 *p, str2 *q) {
    long v1 = q->t; long v2 = q->u;
    p->y = v1+v2;
}
```

GCC 为setVal 产生下面的代码:

```
setVal:
movslq 8(%rsi), %rax
addq 32(%rsi), %rax
movq %rax, 184(%rdi)
ret
```

则A=_____，B=_____.

A. 8,6 B. 10,8 C. 10,5 D. 9.5

答: D. 4 < B <= 8，5 < A <= 10，44 < A*B <= 46，解得 A=9 B=5

2. 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) = _____

(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。

答: 56，40；CC1: 0；II1: 8，必须 4 字节对齐；LL1: 16，必须 8 字节对齐；CC2: 24；LL2: 40，必须 8 字节对齐；II2: 48。基本元素最大的为 long，因此 sizeof(A) 是 56。重排顺序: LL1 LL2 II1 II2 CC1 CC2，刚好没有空白空间，得到的大小为 8+8+4+4+10+6=40 字节。

3. 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) sizeof(struct_e) = _____字节。

(5) 若只将i 的类型改成short，那么sizeof(struct_e) = _____字节。

(6) 若只将 h 的类型改成int，那么sizeof(union_e) = _____字节。

(7) 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。

(8) 若只将short h 的定义删除，那么 (1)-(4) 问的答案分别是_____，_____，_____，_____。

答：

4; 8; 12; 16; 14; 8; 8, 16; 3, 7, 12, 16。具体解释如下：

(1)~(4)：

d1	d2	d3		h								i			
				c1	c2	c3	c4	c5	c6	c7					

(5)：

d1	d2	d3		h							i		
				c1	c2	c3	c4	c5	c6	c7			

(6)：

d1	d2	d3		h							i			
				c1	c2	c3	c4	c5	c6	c7					

(7)：

d1	d2	d3		h							i			
				c1	c2	c3	c4	c5	c6	c7					

对于(7)，虽然和(5)很像，并且 sizeof(union_e)也没变，但是实际上对齐要求的是所有基本元素都要对齐。所以在考虑 sizeof 的时候，不妨假设开辟一个包含两个元素的结构体数组，检查一下第二个结构体是否对其了所有的基本元素。

(8)：

d1	d2	d3	c1	c2	c3	c4	c5	c6	c7			i

三. 汇编大题 (2019 期中)

下面的 C 程序包含`main()`, `caller()`, `callee()` 三个函数。本题给出了该程序的部分 C 代码和 x86-64 汇编与机器代码。请分析给出的代码，补全空白处的内容，并回答问题。

注：汇编与机器码中的数字用 16 进制数填写

```
00000000004006cd <caller>:
4006cd:55                   push %rbp
4006ce:48 89 e5             mov %rsp, %rbp
4006d1:48 83 ec 50          sub $0x50, %rsp
4006d5:48 89 7d b8          mov %rdi, -0x48(%rbp)
4006d9:64 48 8b 04 25 28 00 mov %fs:0x28, %rax
4006e0:00 00
4006e2:48 89 45 f8          mov %rax, -0x8(%rbp)
4006e6:31 c0                xor %eax, %eax
4006e8:c6 45 d0 00          movb $0x0, -0x30(%rbp)
4006ec:c6 45 e0 00          movb $0x0, _(1)_
4006f0:48 8b 45 b8          mov _(2)_ , %rax
4006f4:48 89 c7             mov %rax, %rdi
4006f7:                     callq 400510 <strlen@plt>
4006fc:89 45 cc             mov _(3)_ , -0x34(%rbp)
4006ff:83 7d cc 0e          cmpl $0xe, -0x34(%rbp)
400703:7f _(4)_             jg 400752 <caller+0x85>
400705:83 7d cc 09          cmpl $0x9, -0x34(%rbp)
400709:                     jg 400720 <caller+0x53>
40070b:48 8b 55 b8          mov -0x48(%rbp), %rdx
40070f:48 8d 45 d0          lea _(5)_ , %rax
400713:48 89 d6             mov %rdx, %rsi
400716:48 89 c7             mov %rax, %rdi
400719:                     callq 400500 <strcpy@plt>
40071e:                     jmp 40073b <caller+0x6e>
400720:48 8b 45 b8          mov -0x48(%rbp), %rax
400724:48 8d 50 0a          lea 0xa(%rax), %rdx
400728:48 8d 45 d0          lea -0x30(%rbp), %rax
40072c:48 83 c0 10          add _(6)_ , %rax
400730:48 89 d6             mov %rdx, %rsi
400733:48 89 c7             mov %rax, %rdi
400736:                     callq 400500 <strcpy@plt>
40073b:ff 75 e8             pushq -0x18(%rbp)
40073e:ff 75 e0             pushq -0x20(%rbp)
400741:ff 75 d8             pushq -0x28(%rbp)
400744:ff 75 d0             pushq -0x30(%rbp)
400747:e8 _(7)_             callq 400666 <callee>
40074c:48 83 c4 20          add $0x20, %rsp
```

```
400750:                    jmp 400753 <caller+0x86>
400752:90                  nop
400753:48 8b 45 f8         mov _(8)_ , %rax
400757:64 48 33 04 25 28 00 xor %fs:0x28, %rax
40075e:00 00
400760:                    je 400767 <caller+0x9a>
400762:                    callq 400520 <__stack_chk_fail@plt>
400767:c9                  leaveq
400768:c3                  retq
```

```
#include <stdio.h>
#include "string.h"
#define N _(9)_
#define M _(10)_
typedef union {
    char str_u[N];
    long l;
} union_e;
typedef struct {
    char str_s[M];
    union_e u;
    long c;
} struct_e;
void callee(struct_e s) {
    char buf[M+N];
    strcpy(buf, s.str_s);
    strcat(buf, s.u.str_u);
    printf("%s␣\n", buf);
}
```

```
void caller(char *str) {
    struct_e s;
    s.str_s[0] = '\0';
    s.u.str_u[0] = '\0';
    int len = strlen(str);
    if (len >= M+N)
        _(11)_;
    else if (len < N) {
        strcpy(s.str_s, _(12)_);
    }
    else {
        strcpy(s.u.str_u, _(13)_);
    }
    callee(s);
}
int main(int argc, char *argv[]){
    caller("0123456789abcd");
    return 0;
}
```

(1) _____ (6) _____ (11) _____

(2) _____ (7) _____ (12) _____

(3) _____ (8) _____ (13) _____

(4) _____ (9) _____

(5) _____ (10) _____

`caller` 函数中，变量`s` 所占的内存空间为：(14)_____.

该程序运行后，`printf` 函数是否有输出？输出结果为：(15)_____.

答: (1) `-0x28(%rbp)` (2) `-0x48(%rbp)` (3) `%eax` (4) `4d` (5) `-0x30(%rbp)` (6) `$0x8` (7) `1a ff ff ff` (8) `-0x8(%rbp)` (9) `10` (10) `5` (11) `return` (12) `str` (13) `str+M` (14) 32bytes (15) `abcd`