1. 假设下列 int 和 unsigned 数均为 32 位：

```
int x = 0x80000000;
unsigned y = 0x00000001;
int z = 0x80000001;
```

以下表达式正确的是（ABC）

A. (-x) < 0 ✓

B. (-1) > y ✓

C. (z << 3) == (z * 8) ✓

D. y * 24 == z << 5 - z << 3 ✗

2. x86 体系结构中，下面哪些选项是错误的？（ACD）

A. leal 指令只能够用来计算内存地址 ✗

B. x86_64 机器可以使用栈来给函数传递参数 ✓

C. 在一个函数内，改变任一寄存器的值之前必须先将其原始数据保存在栈内 ✗

D. 判断两个寄存器中值大小关系，只需要 SF（符号）和 ZF（零）两个 conditional code

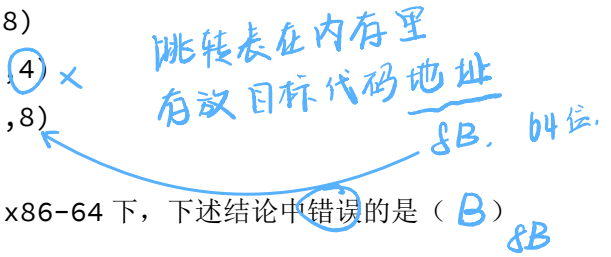
3. 对简单的 switch 语句常采用跳转表的方式实现，在 x86-64 下，下述指令中最有可能正确实现 switch 分支跳转的汇编指令是（D）

A. jmp .L3(,%eax,4)

B. jmp .L3(,%eax,8)

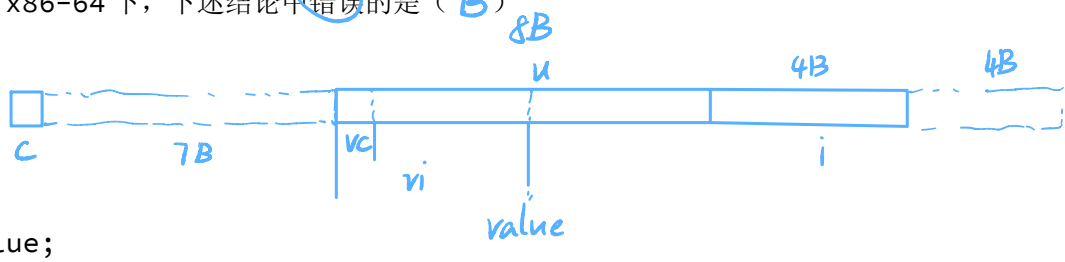
C. jmp *.L3(,%eax,4) ✗

D. jmp *.L3(,%eax,8)



4. 有如下定义的结构，在 x86-64 下，下述结论中错误的是（B）

```
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 ✗ 2 } 指针相减.

C. (&sa.u.vc - &sa.c) == 8 ✓

D. 优化成员变量的顺序，可以做到“sizeof(sa) == 16” ✓

8 + 4 + 1 + 3 = 16.

5. 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) = ____56____ 6+2+4+4+8+10+6+8+4+4

(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___40___字节 8+8+4+4+10+1

以下为 2020 年期中第三大题，2021 年期中第三大题

得分

第三题（20 分）

请分析下面的 C 语言程序和对应的 x86-64 汇编代码。

1.其中，有一部分缺失的代码（用标号标出），请在标号对应的横线上填写缺失的内容。注:汇编与机器码中的数字用 16 进制数填写。

C 语言代码如下：

```
typedef struct _parameters {
    int n;
    int product;
} parameters;
int bar(parameters *params, int x) {
    params->product *= x;
}
void foo (parameters *params) {
    if (params->n <= 1)
        ___ (1) ___
    bar(params, ___ (2) ___);
    params->n--;
    foo(params);
}
```

(1) return;

(2) params->n

x86-64 汇编代码如下（为简单起见，函数内指令地址只给出后四位，需要时可补全）：

```
0x0000555555555189 <bar>:
    5189: f3 0f 1e fa    endbr64
    518d: 55             push   %rbp
    518e: 48 89 e5       mov    %rsp,%rbp
    5191: 48 89 7d f8    mov    _(3)_,-0x8(%rbp)  (3)___%rdi___
    5195: 89 75 f4       mov    %esi,-0xc(%rbp)
    5198: 48 8b 45 f8    mov    -0x8(%rbp),%rax
    519c: 8b 40 04       mov    0x4(%rax),%eax
    519f: 0f af 45 f4    imul   _(4)_(%rbp),%eax  (4)___-0xc___
    51a3: 89 c2          mov    %eax,%edx
    51a5: 48 8b 45 f8    mov    -0x8(%rbp),%rax
```

rax = params

eax = params -> product

eax *= x

edx = eax

rax = params

```
    51a9: 89 50 04       mov    %edx,0x4(%rax)
    51ac: 90             nop
    51ad: 5d             pop    _(5)_
    51ae: c3             retq

000055555555551af <foo>:
    51af: f3 0f 1e fa    endbr64
    51b3: 55             push   %rbp
    51b4: 48 89 e5       mov    %rsp,%rbp
    51b7: 48 83 ec 10    _(6)_  $0x10,%rsp
    51bb: 48 89 7d f8    mov    %rdi,-0x8(%rbp)
    51bf: 48 8b 45 f8    mov    -0x8(%rbp),%rax
    51c3: 8b 00          mov    (%rax),%eax
    51c5: 83 f8 01       cmp    $0x1,%eax
    51c8: 7e 31          _(7)_  51fb <foo+0x4c>
    51ca: 48 8b 45 f8    mov    -0x8(%rbp),%rax
    51ce: 8b 10          mov    (%rax),%edx
    51d0: 48 8b 45 f8    mov    -0x8(%rbp),%rax
    51d4: 89 d6          mov    %edx,%esi
    51d6: 48 89 c7       mov    %rax,%rdi
    51d9: e8 ab ff ff ff    callq  0x0000555555555189 <bar>
    51de: 48 8b 45 f8    mov    -0x8(%rbp),%rax
    51e2: 8b 00          mov    (%rax),%eax
    51e4: 8d 50 ff       lea    -0x1(_(8)_),%edx
    51e7: 48 8b 45 f8    mov    -0x8(%rbp),%rax
    51eb: 89 10          mov    _(9)_,(%rax)
    51ed: 48 8b 45 f8    mov    _(10)_ ,%rax
    51f1: 48 89 c7       mov    %rax,%rdi
    51f4: e8 b6 ff ff ff    callq  _(11)_
    51f9: eb 01          jmp    51fc <foo+0x4d>
    51fb: 90             nop
    51fc: c9             leaveq
    51fd: c3             retq
```

params->x = edx.

(5) %rbp

(6) sub

rax = params

ecx = params->n

ecx : 1

(7) jle

rax = params

edx = params->n

rax = params

esi = edx

rdi = params

rax = params

ecx = params->n

(8) %rax edx = rax - 1

rax = params

(9) %edx

(10) -0x8(%rbp)

(11) 51af <foo>

2.在程序执行到 0x000055555555518e 时(该指令还未执行),此时的栈帧如下,请填写空格中对应的值。

地址	值
0x7fffffffe308	0xffffe340
0x7fffffffe304	0x00000000
0x7fffffffe300	0x00000000
0x7fffffffe2fc	0x00005555
0x7fffffffe2f8	(12) 0x555551ff
0x7fffffffe2f4	0x00007fff
0x7fffffffe2f0	0xffffe310
0x7fffffffe2ec	0x00007fff
0x7fffffffe2e8	0xffffe340
0x7fffffffe2e4	0x00000004
0x7fffffffe2e0	0xffffe350
0x7fffffffe2dc	0x00005555
0x7fffffffe2d8	(13) 0x555551de
0x7fffffffe2d4	0x00007fff
0x7fffffffe2d0	(14) 0xffffe2f0

3.当 params={n,1}时, foo(&params)函数的功能是什么?

计算 n!, 结果存在 params.product 中.

考察内容：

1. 对 51c8 及 51fb 的理解，由 51c8 和 51fb 可以知道这是跳转指令，跳转到 51fb，阅读汇编代码可知这里就结束了。结合源代码这里是 return 退出
2. 对 51ca 和 51ce 的理解，由 51ca 可知此时 rax 保存的是 param 的地址，51ce 直接从地址取值，所以取出的是第一个数 n，即 params->n
3. 51d4 和 51d6 通过两个寄存器传参，分别是%esi 和%rdi，5195 使用了%esi，推出此处使用%rdi
4. 对源程序及 5195 的理解，由 519c 可知 eax 保存的是 param->product，所以这里是执行乘法操作，另一个操作数是参数 x，5195 处已经把 x 保存在-0xc(%rbp)中，所以就从这里读取数据
5. Callee 保存的参数，518d 保存，这里读取
6. 进入函数后首先分配栈帧，通过减%rsp 实现
7. 源程序 if (params->n <= 1)的判断
8. 源程序 params->n--;，上一句已经把 params->n 读到%eax 中，这里进行修改
9. 对源程序 params->n--;的理解，51e4 已经把计算结果保存在%edx 中，这里把%edx 的结果进行保存
10. 与 51e7 相同，与 51d0 的内容也相同，考察调用函数的传参方法
11. 函数递归调用，由 foo 函数的内容可得
12. 函数调用保存返回地址，首先进入 foo 函数，所以返回的是调用 foo 函数后的下一条指令地址
13. 函数调用保存返回地址，在 foo 函数中进入 bar 函数，所以返回的是调用 bar 后的下一条指令
14. Push 之后栈顶的内容，推测出此时的 rbp 是在上一次函数调用，进入 callee 时设置的。上一次函数调用保存的返回地址在 0x7fffffffe2f8，所以上一次 rbp 的值是 0x7fffffffe2f0

得分

第三题（15 分）请阅读并分析下面的 C 语言程序和对应的 x86-64 汇编代码。

1．其中，有一部分缺失的代码（用标号标出），请在标号对应的横线上填写缺失的内容。注:汇编与机器码中的数字用 16 进制数填写。

C 代码如下：

```
long f(long n, long m)
{
  if (n == 0 || (1) m==1)
  return m;
  if ((2) m&1!=0)
  {
    long ret = (3) f(n-1,3*m+1);
    return ret;
  }
  else
  {
    long ret = f(n - 1, m >> 1);
    return ret;
  }
}
```

x86-64 汇编代码如下（为简单起见，函数内指令地址只给出后四位，需要时可补全）：

```
0x0000555555555149 <f>:
    5149:  f3 0f 1e fa          endbr64
    514d:  55                   push   %rbp
    514e:  48 89 e5             mov    (4) %rsp ,%rbp
    5151:  48 83 ec 20          sub    $0x20,%rsp
    5155:  48 89 7d e8          mov    %rdi,-0x18(%rbp)
    5159:  48 89 75 e0          mov    %rsi,-0x20(%rbp)
    515d:  48 83 7d e8 00       cmpq   $0x0,-0x18(%rbp) n : 0
    5162:  74 (5) 07            je     (6) 5166 <f+0x22>
```

```
5164:  48 83 7d e0 01        cmpq   $0x1,-0x20(%rbp)
5169:  75 06                 jne    5171 <f+0x28>
516b:  48 8b 45 e0           mov    (7) -0x20(%rbp),%rax
516f:  eb 5f                 jmp    51d0 <f+0x87>
5171:  48 8b 45 e0           mov    -0x20(%rbp),%rax
5175:  83 e0 01              and    $0x1,%eax
5178:  48 85 c0              test   %rax,%rax
517b:  74 ??                 je     (8) 51ab <f+0x122>
517d:  48 8b 55 e0           mov    -0x20(%rbp),%rdx
5181:  48 89 d0              mov    %rdx,%rax
5184:  48 01 c0              add    %rax,%rax
5187:  48 01 d0              add    %rdx,%rax
518a:  48 8d 50 01           lea    0x1(%rax),%rdx
518e:  48 8b 45 e8           mov    -0x18(%rbp),%rax
5192:  48 83 e8 01           sub    $0x1,%rax
5196:  48 89 d6              mov    (9) %rdx,%rsi
5199:  48 89 c7              mov    %rax,%rdi
519c:  e8 a8 ff ff ff        callq  5149 <f>
51a1:  48 89 45 f8           mov    %rax,-0x8(%rbp)
51a5:  48 8b 45 f8           mov    -0x8(%rbp),%rax
51a9:  eb 25                 jmp    51d0 <f+0x87>
51ab:  48 8b 45 e0           mov    -0x20(%rbp),%rax
51af:  48 d1 f8              (10) sarq   %rax
51b2:  48 89 c2              mov    %rax,%rdx
51b5:  48 8b 45 e8           mov    -0x18(%rbp),%rax
51b9:  48 83 e8 01           sub    $0x1,%rax
51bd:  48 89 d6              mov    (9) %rdx,%rsi
51c0:  48 89 c7              mov    %rax,%rdi
51c3:  e8 81 ff ff ff        callq  5149 <f>
51c8:  48 89 45 f0           mov    %rax,-0x10(%rbp)
51cc:  48 8b 45 f0           mov    -0x10(%rbp),%rax
51d0:  c9                    leaveq
51d1:  c3                    retq
```

m:1
rax=m
rax&=1
rdx=m
rax=2m
rdx=2m+1
m
右移1位

把栈帧划分一下

2. 已知在调用函数 f(7,6)时，我们在 gdb 中使用 b f 指令在函数 f 处加上了断点，下面是程序某一次运行到断点时从栈顶开始的栈的内容，请在空格中填入相应的值。（U 表示不要求填写）

0x7fffffffe558	0x00005555555551c8
0x7fffffffe550	(11) 0x00007fffffffe580 %rbp
0x7fffffffe548	U
0x7fffffffe540	U
0x7fffffffe538	U
0x7fffffffe530	(12) 0x0000000000000005
0x7fffffffe528	(13) 0x00005555555551a1
0x7fffffffe520	0x00007fffffffe550
0x7fffffffe518	U
0x7fffffffe510	U
0x7fffffffe508	(14) 0x0000000000000003
0x7fffffffe500	0x0000000000000010
0x7fffffffe4f8	0x00005555555551c8

3. 运行函数 f(7,6)后得到的值是多少？ (15) 2