得分

第四题（10 分）

一个函数如下，其中部分代码被隐去，请通过gdb调试信息补全代码（4分）。

```
int f(int n, int m) {
    if (m > 0) {
        if (n > 1) {
            int r = f(n - 1, m);
            return (r - 1 + m) % n + 1;
        }
        else if (n == 1) {
            return 1;
        }
    }
    return 0;
}
```

{考察点：x86-64函数调用、参数传递及栈的使用。函数调用通过rdi和rsi传递第一和第二个参数，栈中只记录函数的返回地址。由于是递归调用，需要用栈保存递归过程中参数变量的值。同时考察xor、test、lea、idiv、sete等指令的使用。难点1：两个判断"n > 1"和"n == 1"在汇编代码中只有一次比较，第二次判断相等是通过sete使得n==1时返回值为1，否则返回值为0实现的。难点2：表达式"(r - 1 + m) % n + 1"比较复杂，需要综合多条语句的信息才能分析出来，并且变量r对应于f(n-1,m)的返回值即寄存器%eax，因而不存在相应的赋值指令。}

如下是通过"gcc -g -O2"命令编译后，在gdb中通过"disas f"命令得到的反汇编代码，其中有两个汇编指令不全，请补全这两条汇编指令（2分）。

```
0x00000000004004e0 <f+0>:     mov    %rbx,-0x10(%rsp)
0x00000000004004e5 <f+5>:     mov    %rbp,-0x8(%rsp)
0x00000000004004ea <f+10>:    xor    %eax,%eax
0x00000000004004ec <f+12>:    sub    $0x10,%rsp
0x00000000004004f0 <f+16>:    test   %esi,%esi
0x00000000004004f2 <f+18>:    mov    %edi,%ebp
0x00000000004004f4 <f+20>:    mov    %esi,%ebx
0x00000000004004f6 <f+22>:    jle    0x400513 <f+51>
0x00000000004004f8 <f+24>:    cmp    $0x1,%edi
0x00000000004004fb <f+27>:    jle    0x400521 <f+65>
0x00000000004004fd <f+29>:    lea    -0x1(%rbp),%edi
```

```
0x0000000000400500 <f+32>:     callq  0x4004e0 <f>
0x0000000000400505 <f+37>:     lea    -0x1(%rax,%rbx,1),%edx
0x0000000000400509 <f+41>:     mov    %edx,%eax
0x000000000040050b <f+43>:     sar    $0x1f,%edx
0x000000000040050e <f+46>:     idiv   %ebp
0x0000000000400510 <f+48>:     lea    0x1(%rdx),%eax
0x0000000000400513 <f+51>:     mov    (%rsp),%rbx
0x0000000000400517 <f+55>:     mov    0x8(%rsp),%rbp
0x000000000040051c <f+60>:     add    $0x10,%rsp
0x0000000000400520 <f+64>:     retq
0x0000000000400521 <f+65>:     sete   %al
0x0000000000400524 <f+68>:     movzbl %al,%eax
0x0000000000400527 <f+71>:     jmp    0x400513 <f+51>
```

{考察点：函数中使用到了%rbp和%rbx寄存器，两者都是callee保存的寄存器，使用前需要压栈，函数返回时需要弹栈恢复寄存器的值。通过前后汇编代码的对比，应该可以猜出两个空分别填写什么；但要注意，压栈和弹栈时，%rsp寄存器的值不同，因而对应的地址表示也不同。}

已知在调用函数f(4, 3)时，我们在函数f中指令retq处设置了断点，下面列出的是程序在第一次运行到断点处暂停时时，相关通用寄存器的值。请根据你对函数及其汇编代码的理解，填写当前栈中的内容。如果某些内存位置处内容不确定，请填写X。（4分）

```
rax             0x1
rbx             0x3
rcx             0x3
rdx             0x309c552970
rsi             0x3
rdi             0x1
rbp             0x2
rsp             0x7fffffffe340
rip             0x400520
```

{考察点：递归调用的返回地址共三处是明确的，并且相同，值可以从反汇编代码中确定（1 分）；三次递归调用程序栈中，压入的%rbx（m）的值不变，压入的%rbp（n）的值为每次减小 1（1 分）；注意 x86-64,栈中的数据都是 64 位的，但因为数值均比较小，所以这 9 个位置处的高 4 字节均为 0（1 分）；其余位置的内容均是不确定的（1 分）。}

0x7fffffffe38c	X
0x7fffffffe388	X
0x7fffffffe384	X
0x7fffffffe380	X
0x7fffffffe37c	X
0x7fffffffe378	X
0x7fffffffe374	0x0
0x7fffffffe370	0x00400505
0x7fffffffe36c	0x0
0x7fffffffe368	0x4
0x7fffffffe364	0x0
0x7fffffffe360	0x3
0x7fffffffe35c	0x0
0x7fffffffe358	0x00400505
0x7fffffffe354	0x0
0x7fffffffe350	0x3
0x7fffffffe34c	0x0
0x7fffffffe348	0x3
0x7fffffffe344	0x0
0x7fffffffe340	0x00400505
0x7fffffffe33c	0x0
0x7fffffffe338	0x2
0x7fffffffe334	0x0
0x7fffffffe330	0x3
0x7fffffffe32c	X
0x7fffffffe328	X
0x7fffffffe324	X
0x7fffffffe320	X