

**实验报告**



**题目： 拆解二进制炸弹**

**班 级： 2020211306**

**学 号： 2020211376**

**姓 名： 马天成**

**学 院： 计算机学院**

**2021年 11 月 06 日**

目录

[一、实验目的 3](#_Toc87901329)

[二、实验环境 3](#_Toc87901330)

[三、实验内容 3](#_Toc87901331)

[四、实验步骤及实验分析 3](#_Toc87901332)

[准备工作 3](#_Toc87901333)

[阶段一：尝试调试 4](#_Toc87901334)

[阶段二：拆phase\_1 -> phase\_6 4](#_Toc87901335)

[phase\_1 4](#_Toc87901336)

[phase\_2 5](#_Toc87901337)

[phase\_3 7](#_Toc87901338)

[phase\_4 8](#_Toc87901339)

[phase\_5 10](#_Toc87901340)

[phase\_6 12](#_Toc87901341)

[阶段三：secret\_phase 15](#_Toc87901342)

[找到进入方式 15](#_Toc87901343)

[寻找secret\_phase答案 16](#_Toc87901344)

[五、总结体会 17](#_Toc87901345)

[六、诚信声明（不签扣10分） 19](#_Toc87901346)

一、实验目的  
1.理解C语言程序的机器级表示。  
2.初步掌握GDB调试器的用法。  
3.阅读C编译器生成的x86-64机器代码，理解不同控制结构生成的基本指令模式，过程的实现。

# 二、实验环境

SecureCRT（10.120.11.12）

Linux

Objdump命令反汇编

GDB调试工具

积分榜（<http://10.120.11.13:19220/scoreboard>）

报告邮寄（X86版本最迟时间：2021年11月17日晚23：59；Arm版本最迟时间：2021年11月24日晚23：59）：大二班（5-8班）：[yangyyj98@bupt.edu.cn](mailto:yangyyj98@bupt.edu.cn)

# 三、实验内容

登录bupt1服务器，在home目录下可以找到Evil博士专门为你量身定制的一个bomb，当运行时，它会要求你输入一个字符串，如果正确，则进入下一关，继续要求你输入下一个字符串；否则，炸弹就会爆炸，输出一行提示信息并向计分服务器提交扣分信息。因此，本实验要求你必须通过反汇编和逆向工程对bomb执行文件进行分析，找到正确的字符串来解除这个的炸弹。

本实验通过要求使用课程所学知识拆除一个“binary bombs”来增强对程序的机器级表示、汇编语言、调试器和逆向工程等方面原理与技能的掌握。 “binary bombs”是一个Linux可执行程序，包含了5个阶段（或关卡）。炸弹运行的每个阶段要求你输入一个特定字符串，你的输入符合程序预期的输入，该阶段的炸弹就被拆除引信；否则炸弹“爆炸”，打印输出 “BOOM!!!”。炸弹的每个阶段考察了机器级程序语言的一个不同方面，难度逐级递增。

为完成二进制炸弹拆除任务，需要使用gdb调试器和objdump来反汇编bomb文件，可以单步跟踪调试每一阶段的机器代码，也可以阅读反汇编代码，从中理解每一汇编语言代码的行为或作用，进而设法推断拆除炸弹所需的目标字符串。实验2的具体内容见实验2说明。

# 四、实验步骤及实验分析

## 准备工作

先熟悉程序的机器级表示，如move指令集、跳转指令集、算术运算指令集、逻辑运算指令集；以及各种条件码和寄存器的固定使用方式；熟悉各类指令的基本组合，如test %rax %rax + jne address。

然后熟悉objdump的基本操作 ( objdump -d bomb > bomb.o …… )。

熟悉gdb的基本操作 ( break 地址/函数名 ; disassemble (函数名) ; stepi ; print ; x…… )。

## 阶段一：尝试调试

首先尝试上手gdb。从开始调试，设置断点，以及最后的逐步调试。

第一个实验显然就是练手的。我简要概括说一下遇到的难点：

1. 看不太懂汇编代码。一开始看真的很难受，一堆指令根本不知道在干啥，不了解组合，以及一看到 跳转就头疼。（当然现在也是）

2. 不会把握调试。一开始大刀阔斧让后边使劲continue；炸了之后又一步步stepi。

3. 不会看寄存器。一开始总是认为寄存器的值是胡乱的，是中间的计算量，无法跟踪。

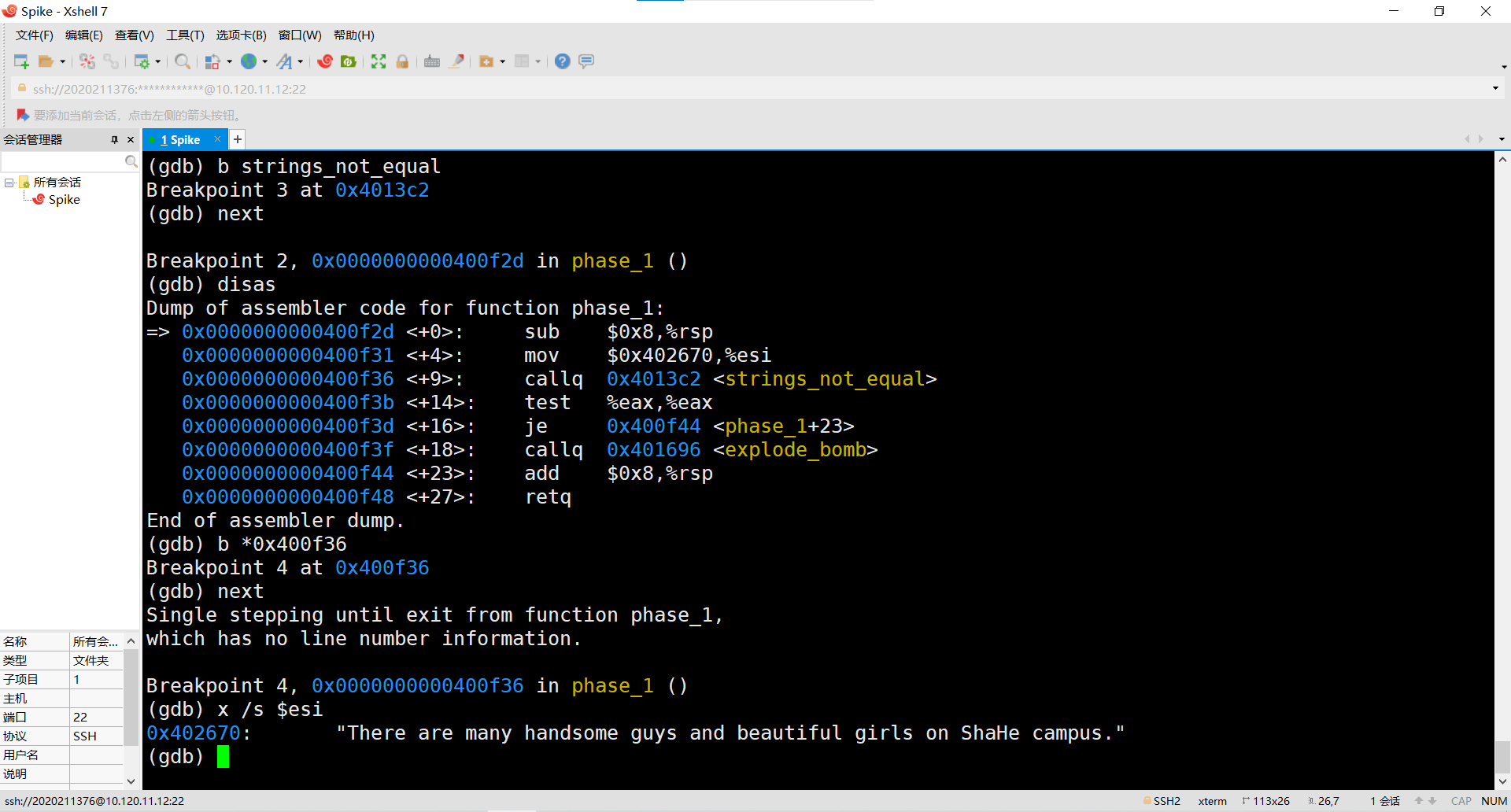
当然，这些问题我在拆7个炸弹的时候就慢慢解决了所谓的心理障碍和能力障碍。

## 阶段二：拆phase\_1 -> phase\_6

这就牵扯到详细的拆炸弹过程了。下面我详细介绍基础的6个炸弹是如何拆掉的。

难点主要从汇编代码反向理解源代码、调试并实时查看寄存器状态、理解寄存器和内部空间三个角度展开。

### phase\_1



phase\_1

**1. 初步理解程序**

在phase\_1处设了断点，并用disassemble指令观察了汇编代码。

可知大致意思为：调用strings\_not\_equal函数，判断返回值；如果返回值为0，则爆炸。

基于此，可以判断：我们要**输入正确的字符串与答案匹配**才能使程序跳过爆炸阶段。

**2. 寻找字符串**

****

根据寄存器状态可知：进入strings\_not\_equal后，我们的**寄存器参数esi**打印字符串可得:

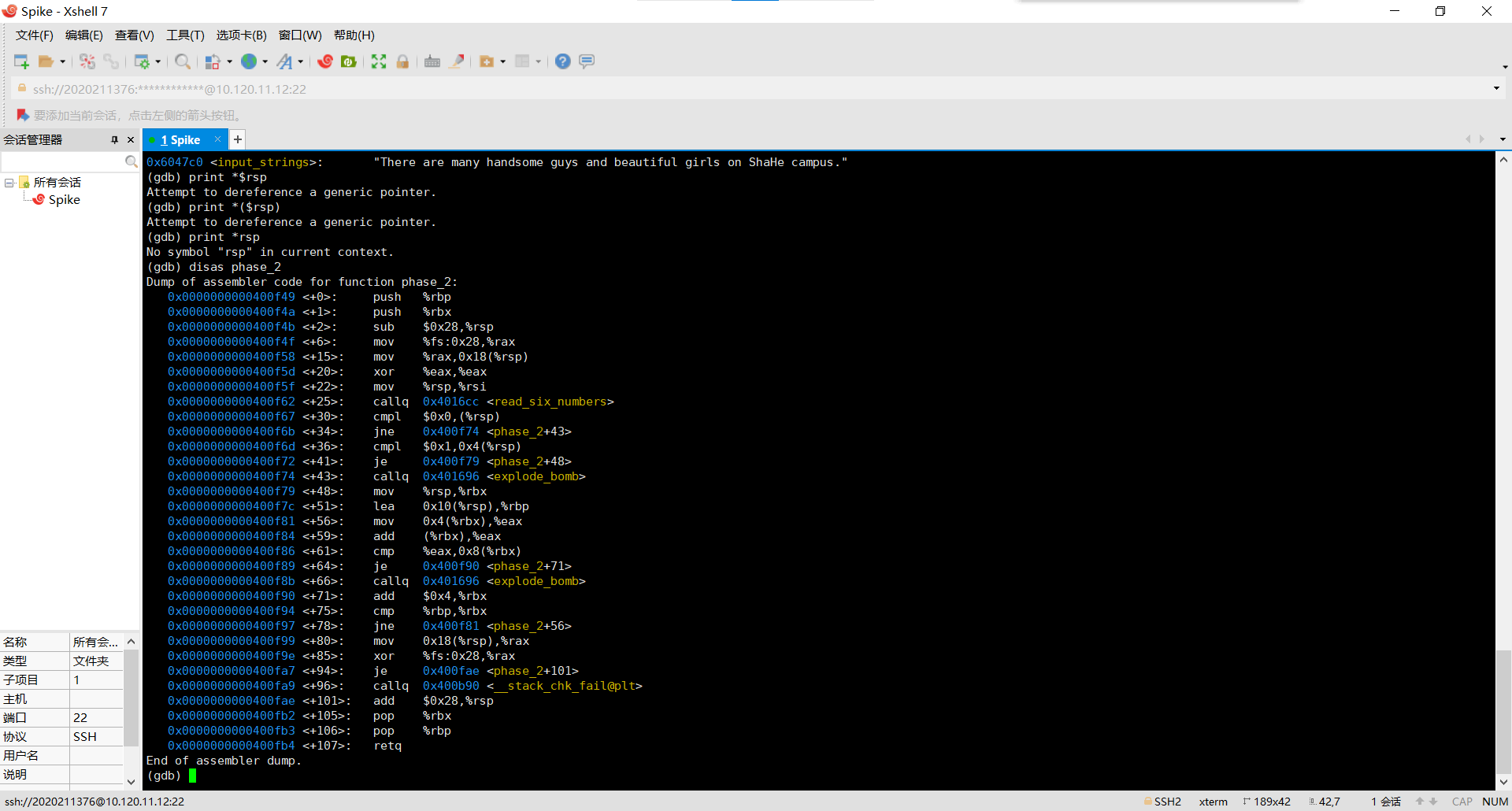
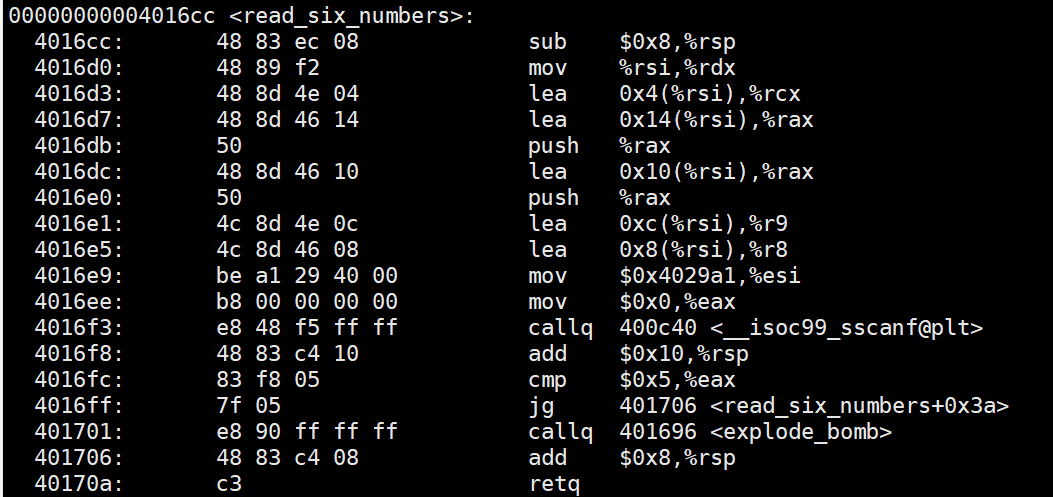
**“There are many handsome guys and beautiful girls on ShaHe campus.”**

由函数名易知，我们输入这个字符串，匹配后，就能使返回值为1，跳过该次爆炸。

**3. 增加的理解**

* + 用 **x /s address** 打印**字符串**
  + **rax ( eax )** 一般做**函数返回值**
  + **rdi rsi**一般做**函数参数**

### phase\_2

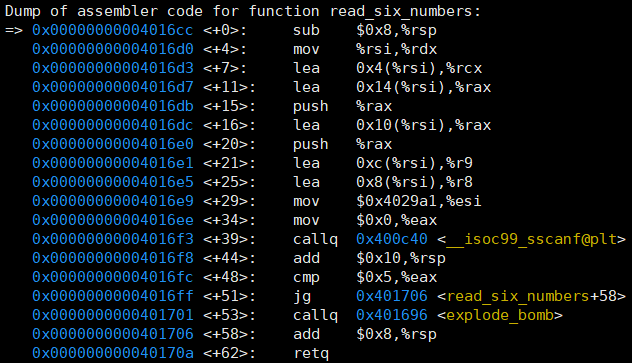


phase\_2 read\_six\_numbers

**1. 初步理解程序**

在汇编代码中，他告诉我 ”read\_six\_numbers”，所以我应该是要输留个数六个数进去。

但是这个**read\_six\_numbers也是有炸弹的**！

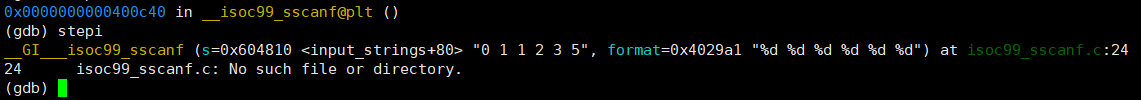


第一遍我在输入时只输入了一个0，导致在调用完这个函数后就炸了，**Boom！**

这个函数中**eax**是**记录读入数字的个数**。

按图中效果，如果输入数字个数小于等于**5**，则函数会调用explode\_bomb进行爆炸。

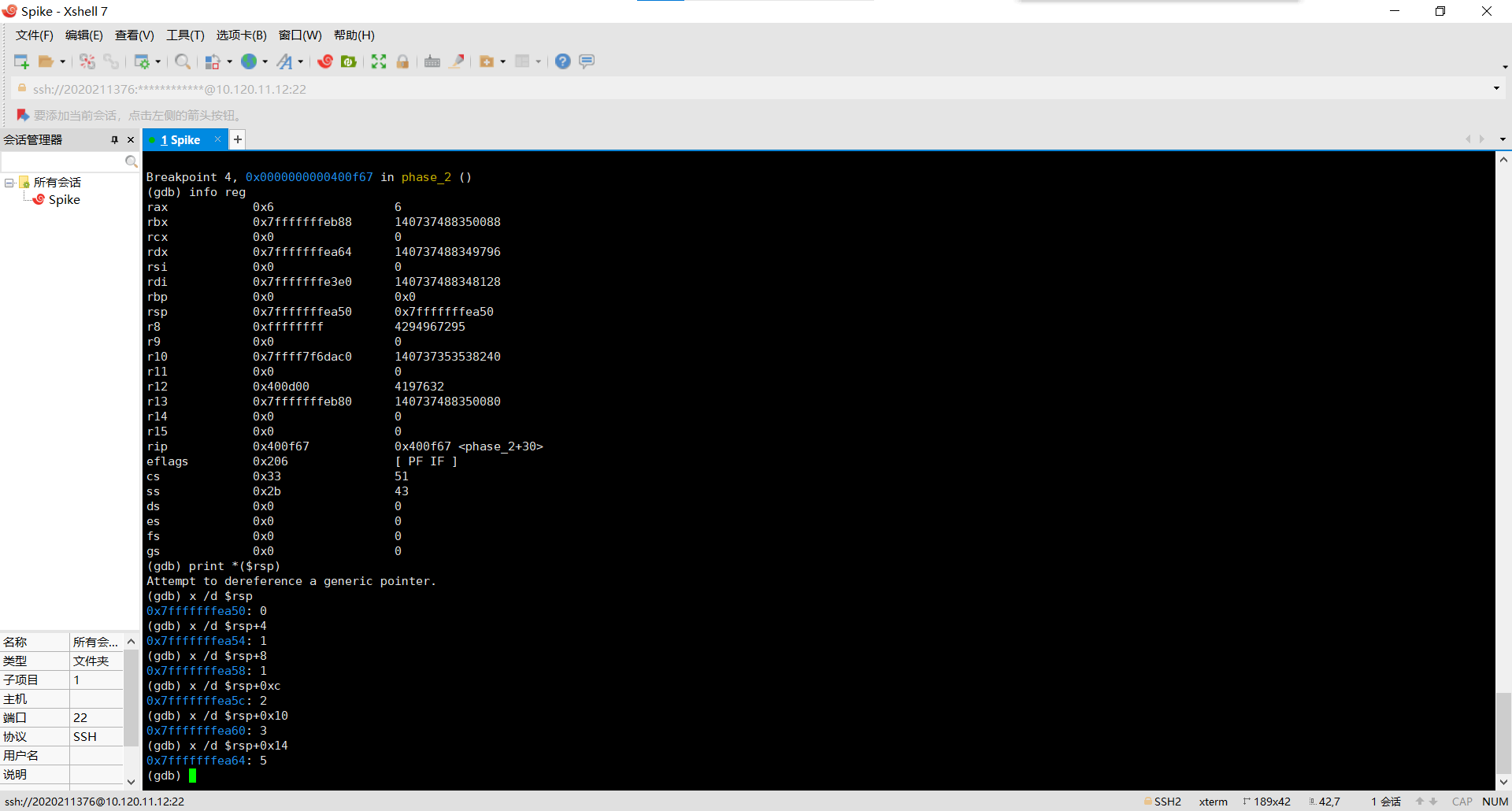
进这个函数后进了scanf，并出现以下提示：



scanf\_2

输入格式是**“%d %d %d %d %d %d”**，所以我需要的输入是6个数。

**2. 寻找六个数**

 根据观测寄存器可知，我输入的数是被保存在了**rsp（开栈）**的连续空间中。

查看源代码，它通过寄存器的来回赋值和逐步地址+0x4（**数组是连续空间，地址+4就是下标加一 取值**）移位操作，来达成逐步计算两项之和的功能，并且与第三项进行比较：

1. 匹配第一个输入是否为 **$0x0**；

2. 匹配第二个输入是否为 **$0x1**；

3. 循环 **$0x6** 次；

4. 循环算出前两个数之和并且与当前项比较，相等则继续循环；否则直接调用**explode\_bomb**；

5. 执行完循环则返回，并输出**phase\_defused**。

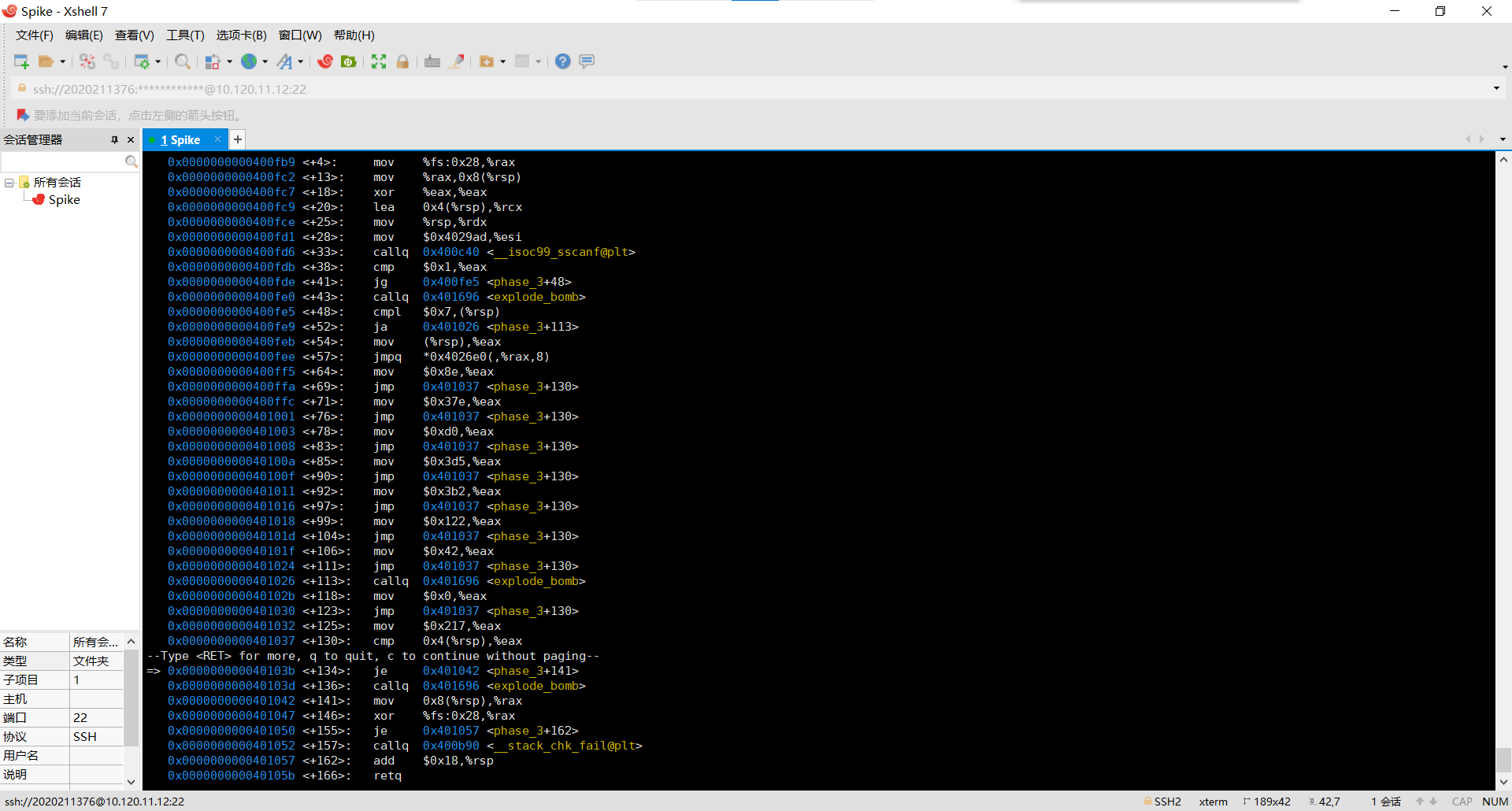
所以我只需要输入标准斐波那契数列前六项的数据，这个就能过了。

答案是 **( 0 , 1 , 1 , 2 , 3 , 5 )**

**3. 增加的理解**

* + 要理解机器代码实现**循环**：通过**cmp/test** 配合 **jump**，相当于**do-while** 或者 **goto**。
  + **栈帧rsp**很有用，是栈帧，保存当前函数地址，开栈存储输入的值。
  + 理解了**数组**的机器级结构，主要是操作地址进行下标的变化和取值。

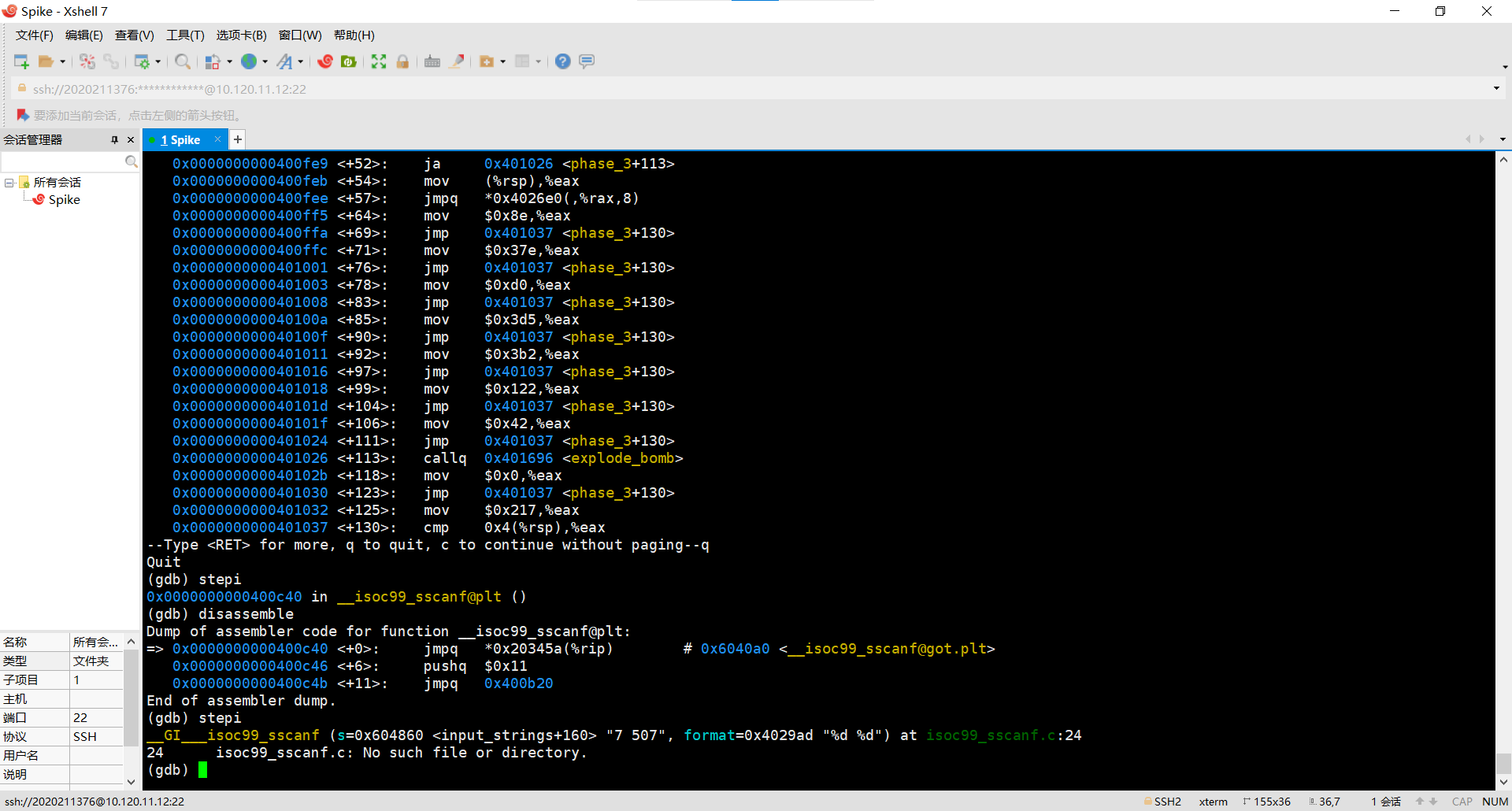
### phase\_3



phase\_3

**1. 初步理解程序**

从这个程序开始，就没有明确的提示输入的信息了。需要我们自己进入scanf来看。

**fomart=”%d %d”**，表示此时我们要输入两个数字来读入程序。

**2. 寻找二元组答案**

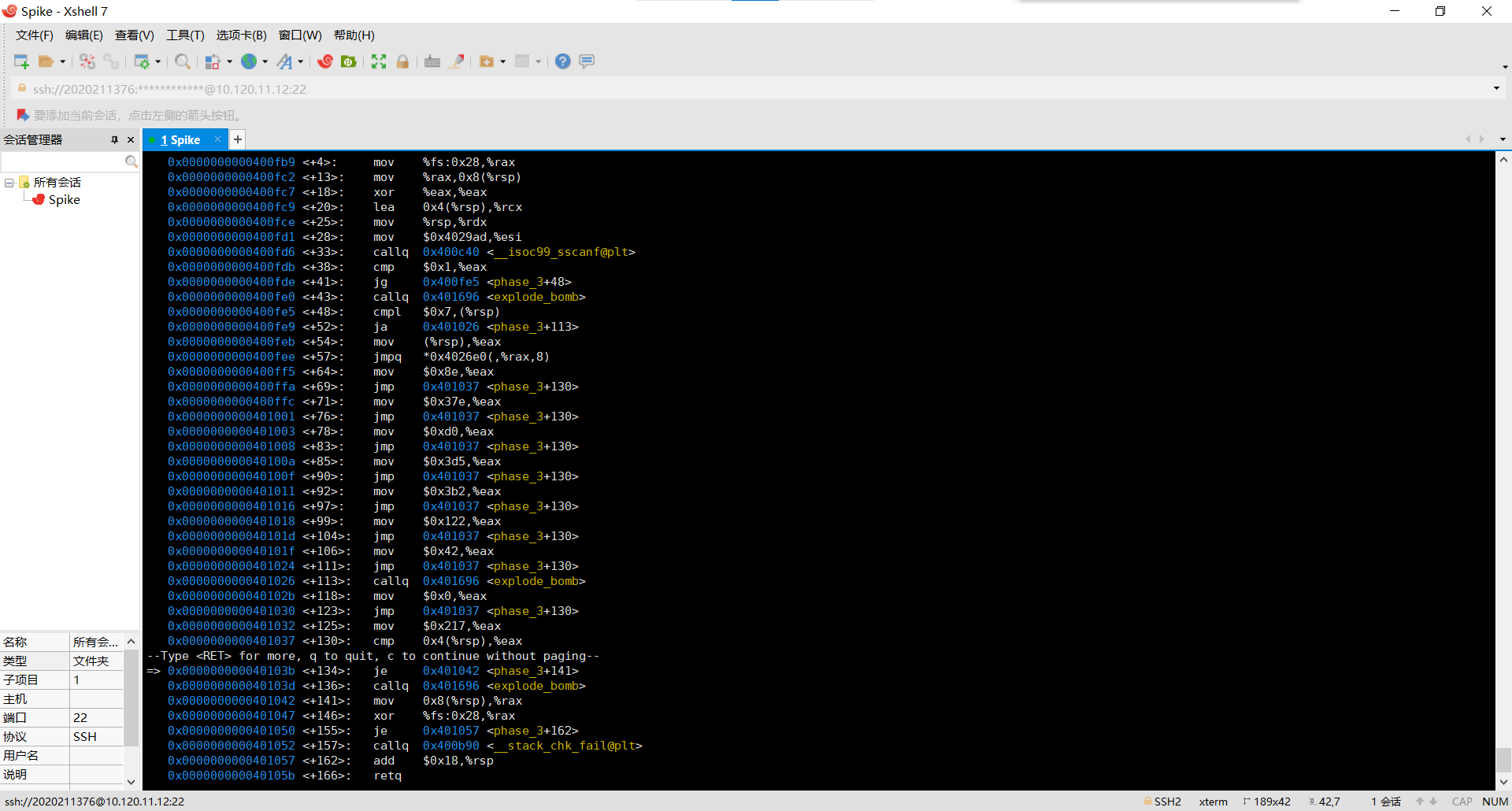
**info register**后，查看寄存器状态和地址取出来的值知道：

1. 我们存的第一个数是在rsp的地址中；

2. 第二个数就在它后面四个字节（数字占4个字节）。

所以我们要找出答案，用我们的输入来跳过相应的爆炸步骤。

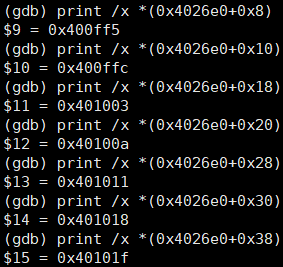
**第一个数：**



根据**cmpl $0x7 , (%rsp)** 知，第一个数要小于等于7才能跳过第一个炸弹。

**第二个数：**

根据**jmpq \*0x4026e0(,%rax,8)** 知：我们会根据第一次的输入跳转到相应行；而在执行完相应行之 后都会跳出。显然，这是一个**switch指令**。

根据打印相应的值，1-7会跳转到相应的行来比较我们输入的第二个数；而对于不同的行，要比较的数字也不一样，所以一共有**7个答案**。

**( 1 , 0x8e ) ( 2 , 0x37e ) ( 3 , oxd0 ) ( 4 , 0x3d5 ) ( 5 , 0x3b2 )**

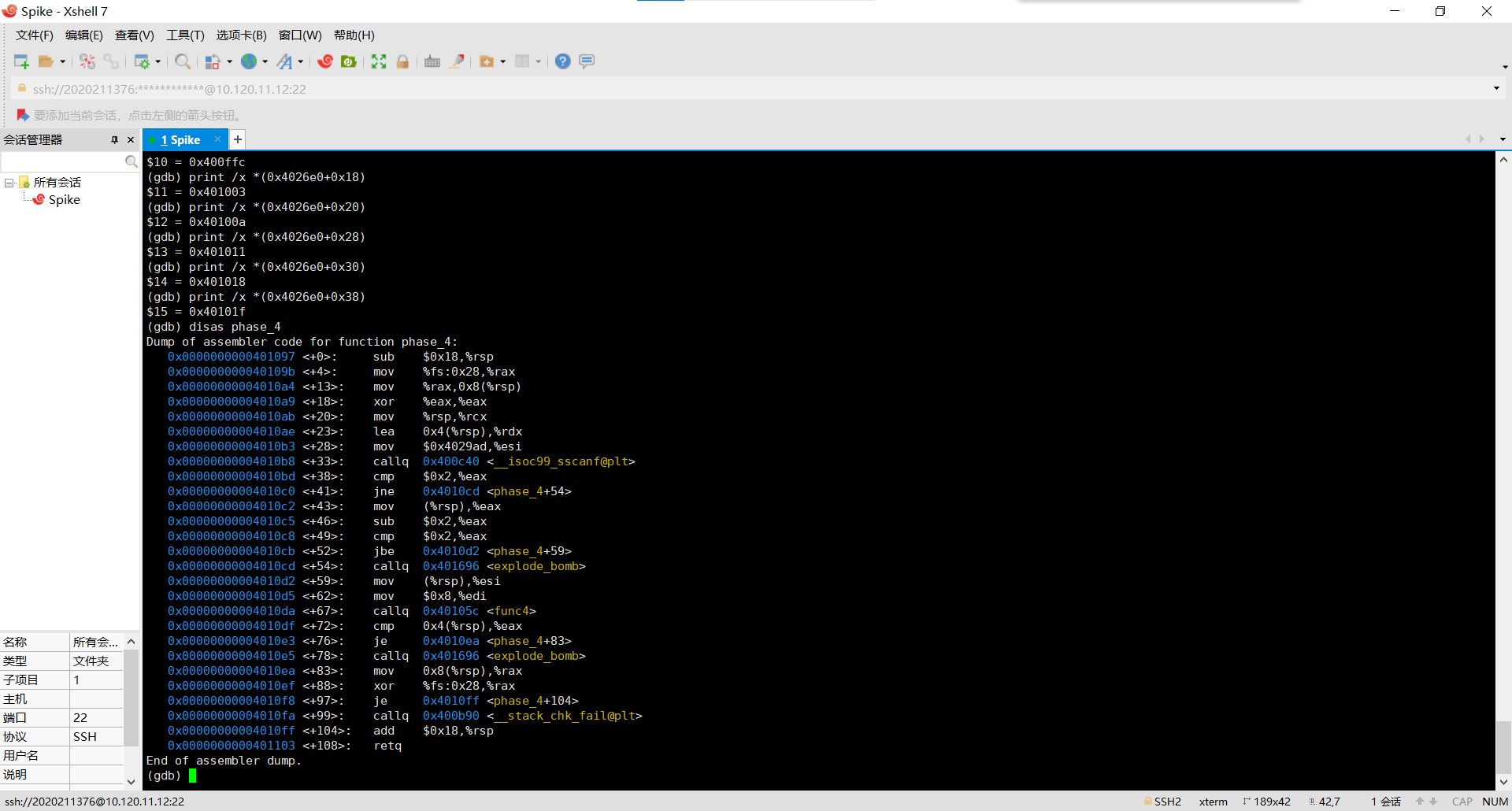
**( 6 , 0x122 ) ( 7 , 0x42 )**

我选择了一个比较小的值，就是 ( 7 , 66 ) -- 怕错 \*( ^ ~ ^ )\*

**3. 增加的理解**

* + 了解了**switch**的执行方式：通过参数来**选择要跳转的地址来执行相应的步骤**。
  + 懂得了拆这个炸弹要先学会尝试，现带入几个试试，然后逐个找到正确输入。
  + 熟悉了寄存器取地址和取值的操作。

### phase\_4



phase\_4 func4

从这一题开始，对于不完全熟悉汇编代码的我，题目变得抽象起来了。

**1. 初步理解程序**

本题是递归函数。首先，通读汇编代码，根据上一题的经验，**rsp是栈帧**，保存我的输入地址。

通过 **callq <func4>** 可知，我们需要运行**func4**，然后函数的返回值是 **%eax**。

通过 **cmp 0x4(%rsp) %eax** 可知，是我输入的第二个int与答案作比较。

初步解读：

1. 我的**第一个输入要合法**，跳过第一个炸弹步骤；

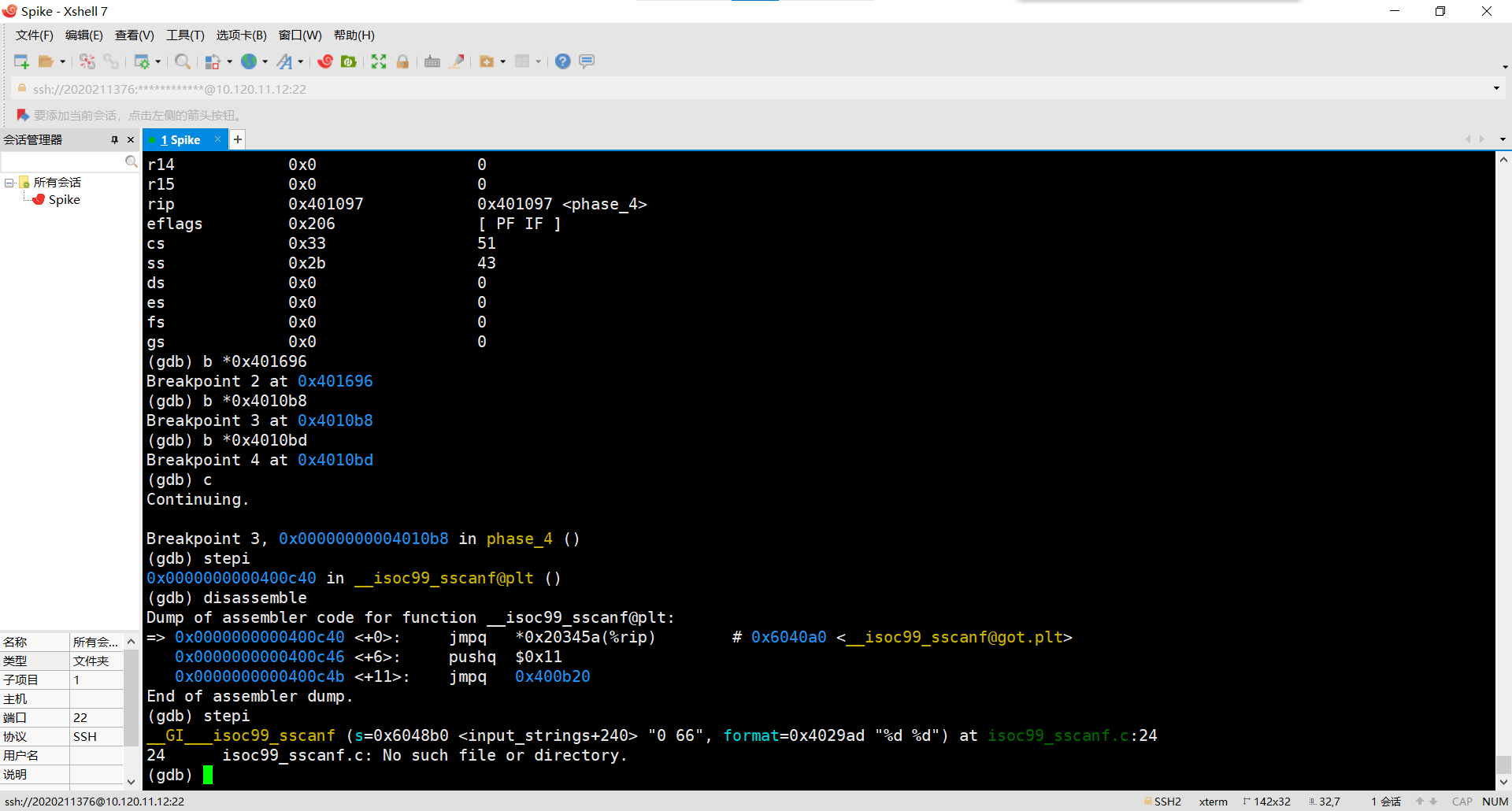
2. 将我的第一个输入传参进入func4，然后将**跑出来的答案与我的第二个输入匹配**；

3. 如果第二次匹配正确，则输出**phase\_defused**；否调用**explode\_bomb**。

所以我完全可以先往里面跑一遍，然后获得func4的运行结果，就可以得到答案了！

当然这是拆炸弹的思路。这是练习，还是得看懂汇编代码。

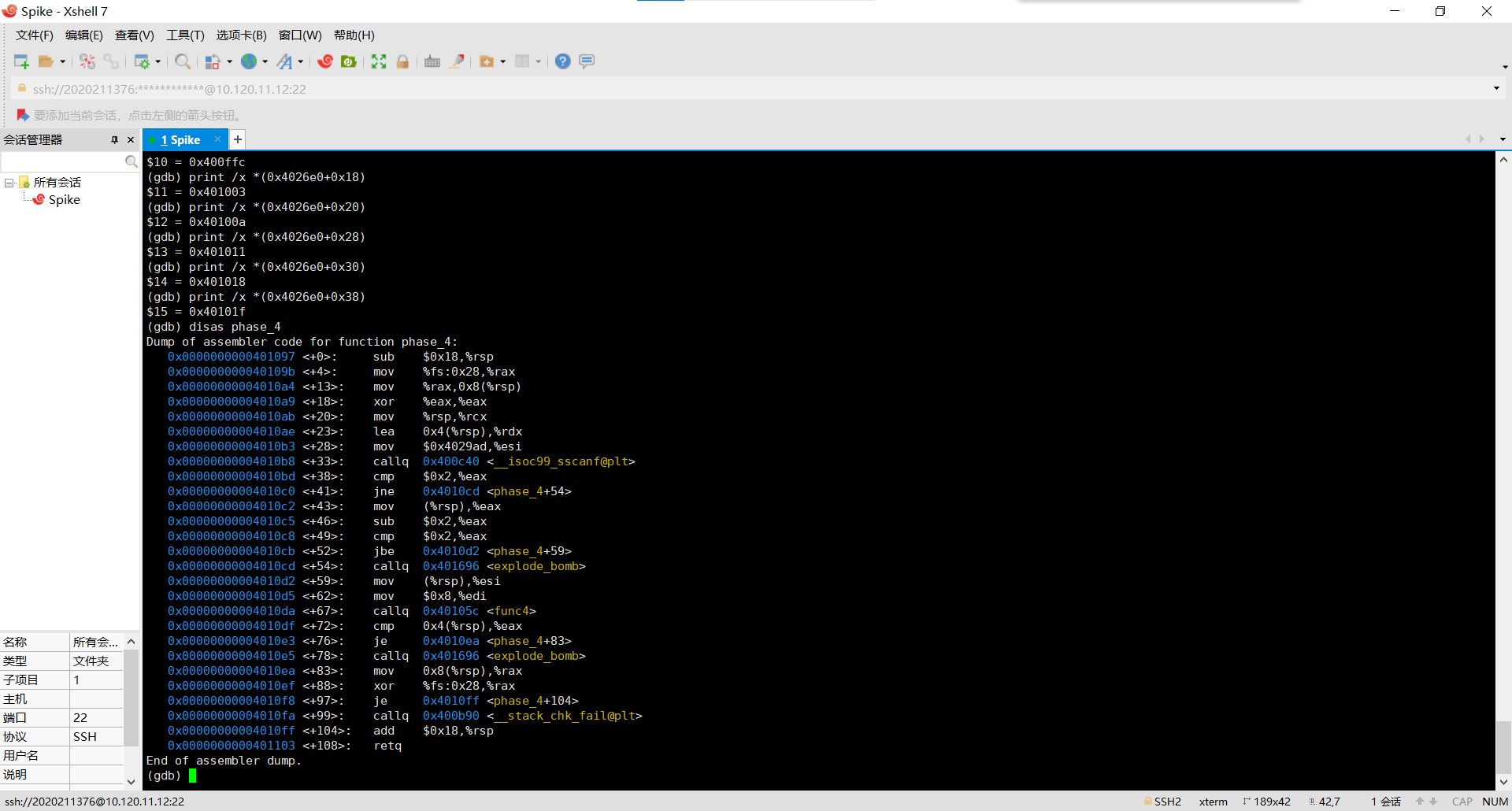
通过两次比较，我大概猜测我的输入是两个int；跑进scanf后也印证了我的猜测：



scanf\_4

**2. 寻找答案**

**第一个数：**

****

1. **mov (%rsp) , %eax** ：将我的第一个数赋给 **%eax** ；

2. **sub $0x2 , %eax** ：将 **%eax** 减2；

3. **cmp $0x2 , %eax** ：将 **%eax** 与2比较；

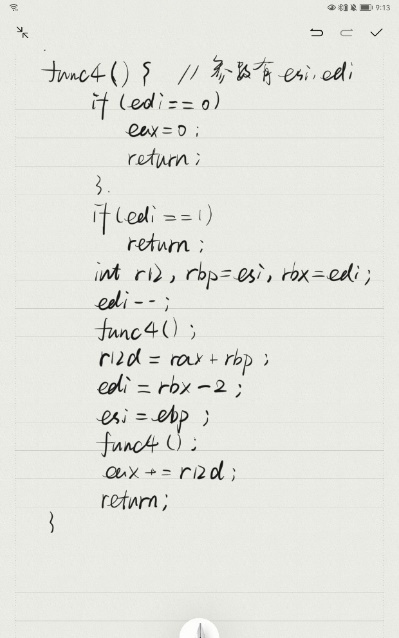
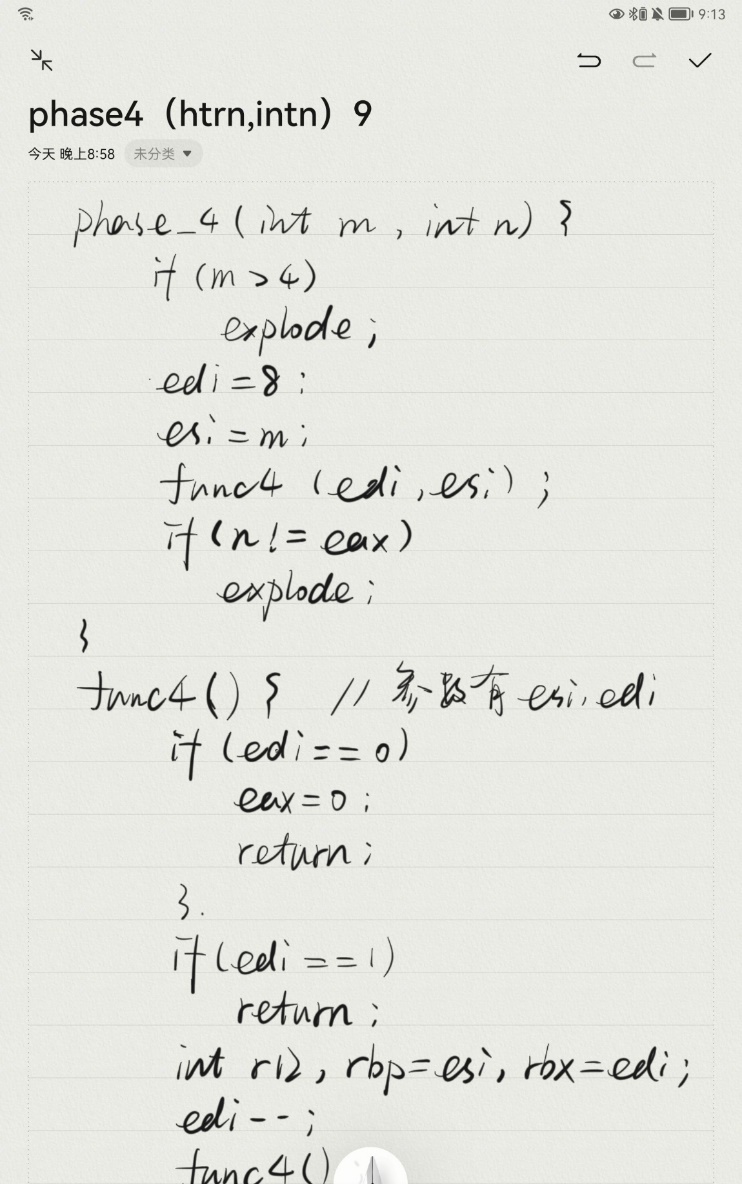
4. 如果小于等于4则继续进行；否则调用explode\_bomb；

所以我们知道，输入必须小于等于4；我这里选择**4**。

**第二个数：**

首先找到传入的参数 **%edi = 8 , %esi = (%rsp)**；直观观察func4可知，我们的edi是控制深度的量。

函数手写代码如下：

 phase\_4手稿 func4手稿

根据深度控制条件反推，我这里代码的功能是：

**Step 1**. 一共最多开**8**层栈，从初到末记为**1-8**；

**Step 2**. 对每一层，**eax**，**esi**，**edi**都可以**视为全局变量**；

**Step 3**. 在每一层，**eax加上计算前eax+我输入的第一个值**；

**Step 4**. 每次阶段一结束，**edi-1**；每次阶段二结束，**edi-2**；

**求出答案：**

注意：我们的输入是反过来的：

**\*rsp存的是第二个数，\*(rsp+4)存的是第一个数**；

最终答案计算为 **( 214 , 4 )**

**3. 增加的理解**

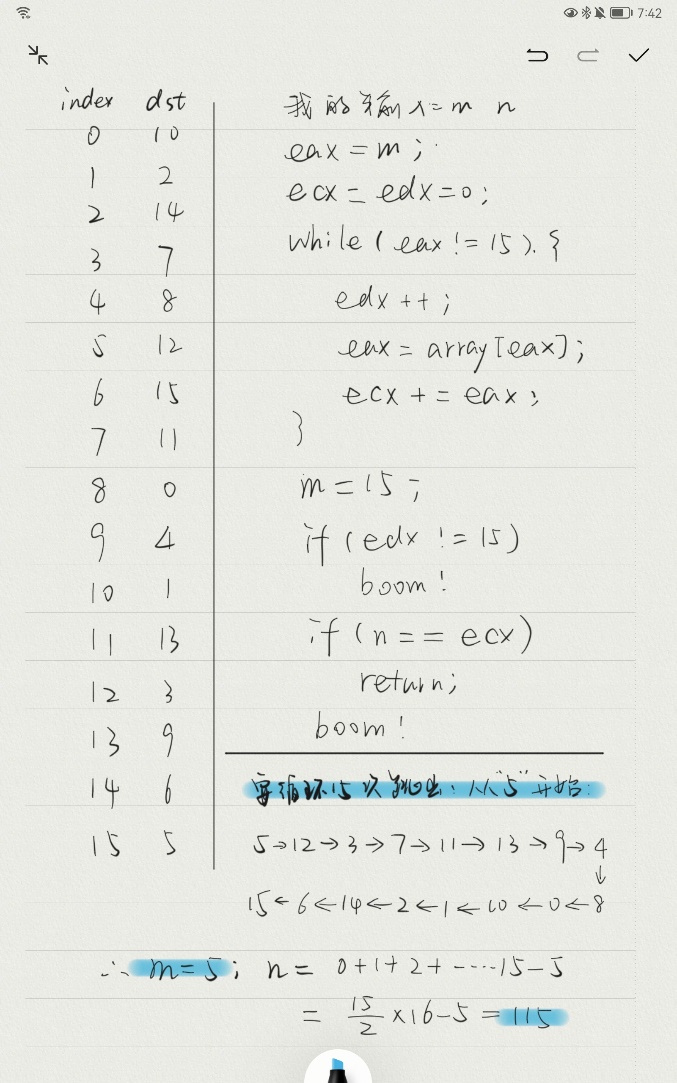
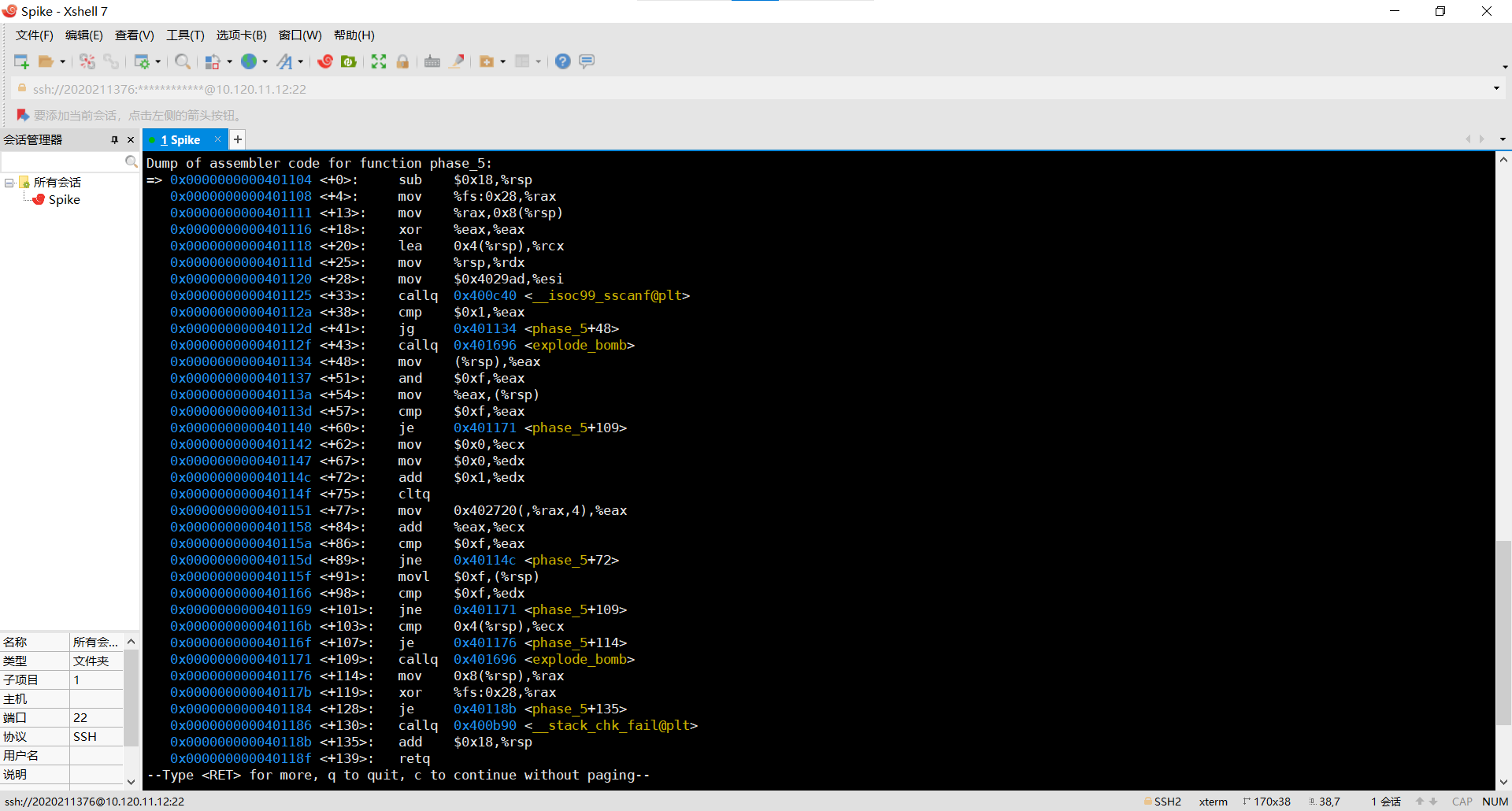
* + 见识了**递归函数的恶心程度**，以后再也不写递归了！！！（快改了，但有时候转递推是真的有点难想）
  + 同时也加深了**传入参数**和**返回参数**的理解，也重新认识了如何解决临时变量：

**push 参数寄存器（加入临时变量）**

**pop 参数寄存器（释放临时变量）**

* 认识到传入参数和返回值寄存器就是类似于全局变量的存在，而其他就是临时变量了。

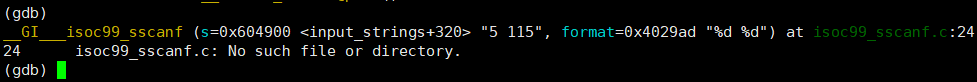
### phase\_5



phase\_5 phase\_5手稿

**1. 初步理解程序**

显然这次也是要输入几个数。不猜了，直接进去看：



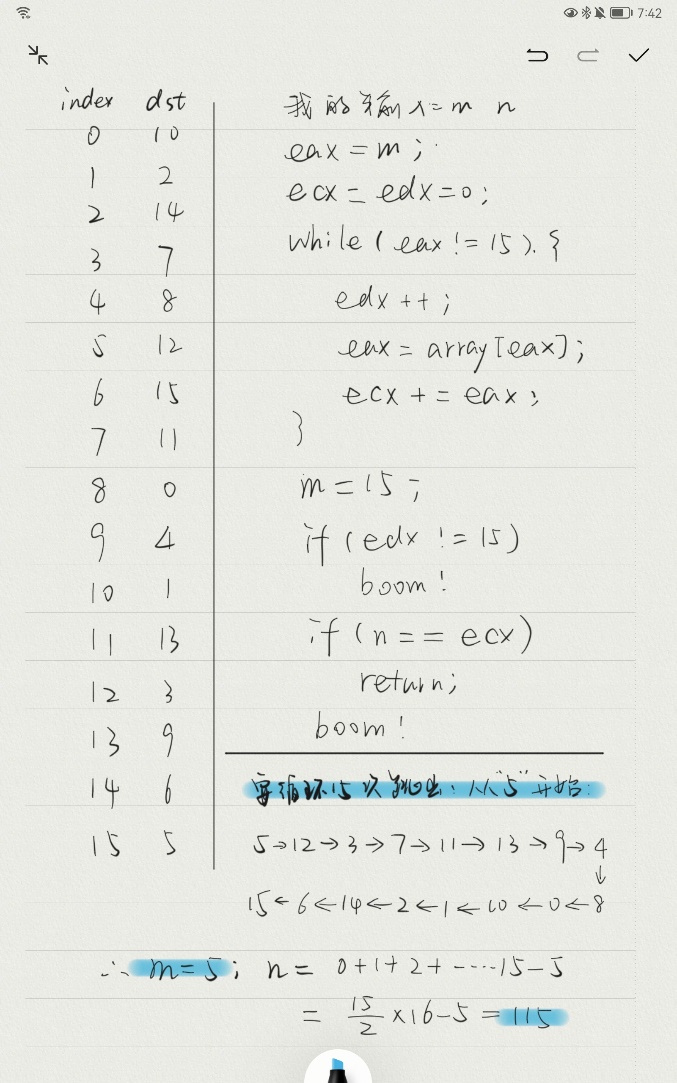
**“%d %d”**，显然是要求输入两个数。

了解到这次是输入两个数，那么先来找找线索。

本次好像只需要比较我的第二个数？那么，找第一个数是干什么的就很重要了。

**2. 寻找答案**

根据我的反汇编至源程序的代码可知：

**Step 1**. 我的输入是**m**，**n**；

**Step 2**. 有计数器**edx**初始值为**0**；每次循环**+1**；

**Step 3**. 初始下标**eax**的值为我的输入**m**。( **eax** = **m** )

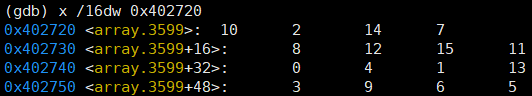
**Step 4**. 循环中，在数组中根据**eax**作为下标找到对应内容，并将其给**eax**；

**Step 5**. 一共要循环**15**次，要不然直接调用**explode\_bomb**。

图中的**Array数组**哪里来？观察到：**mov 0x402720(,%rax,4) eax**

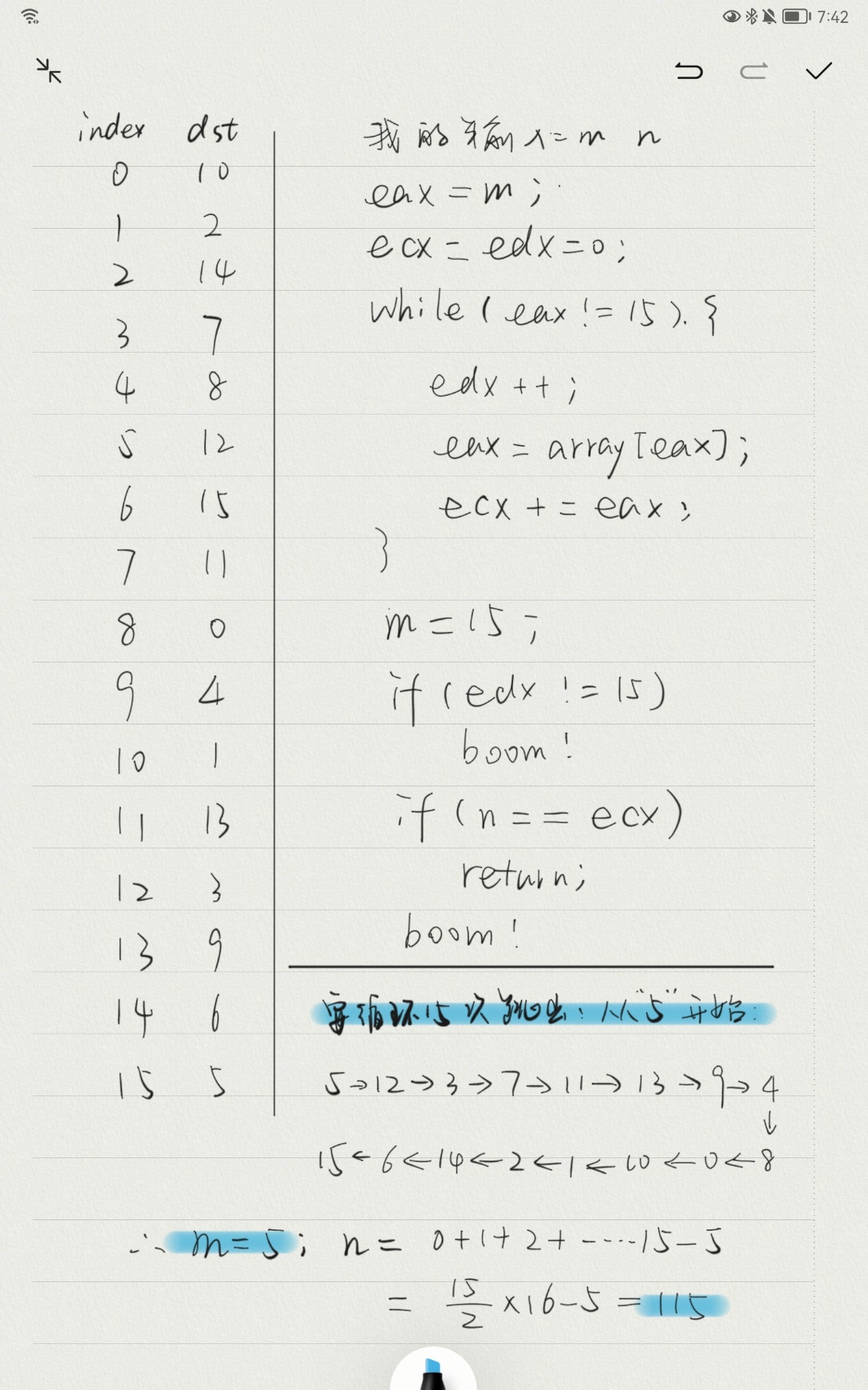
这就是我的数组，且起始地址是0x402720。

打印出来如下：



总共有16个，整理关系就是我在上面完整手稿写的那样。

根据**反推原则**，**最后一步计算是找到值为15的数组单元**



所以答案是：

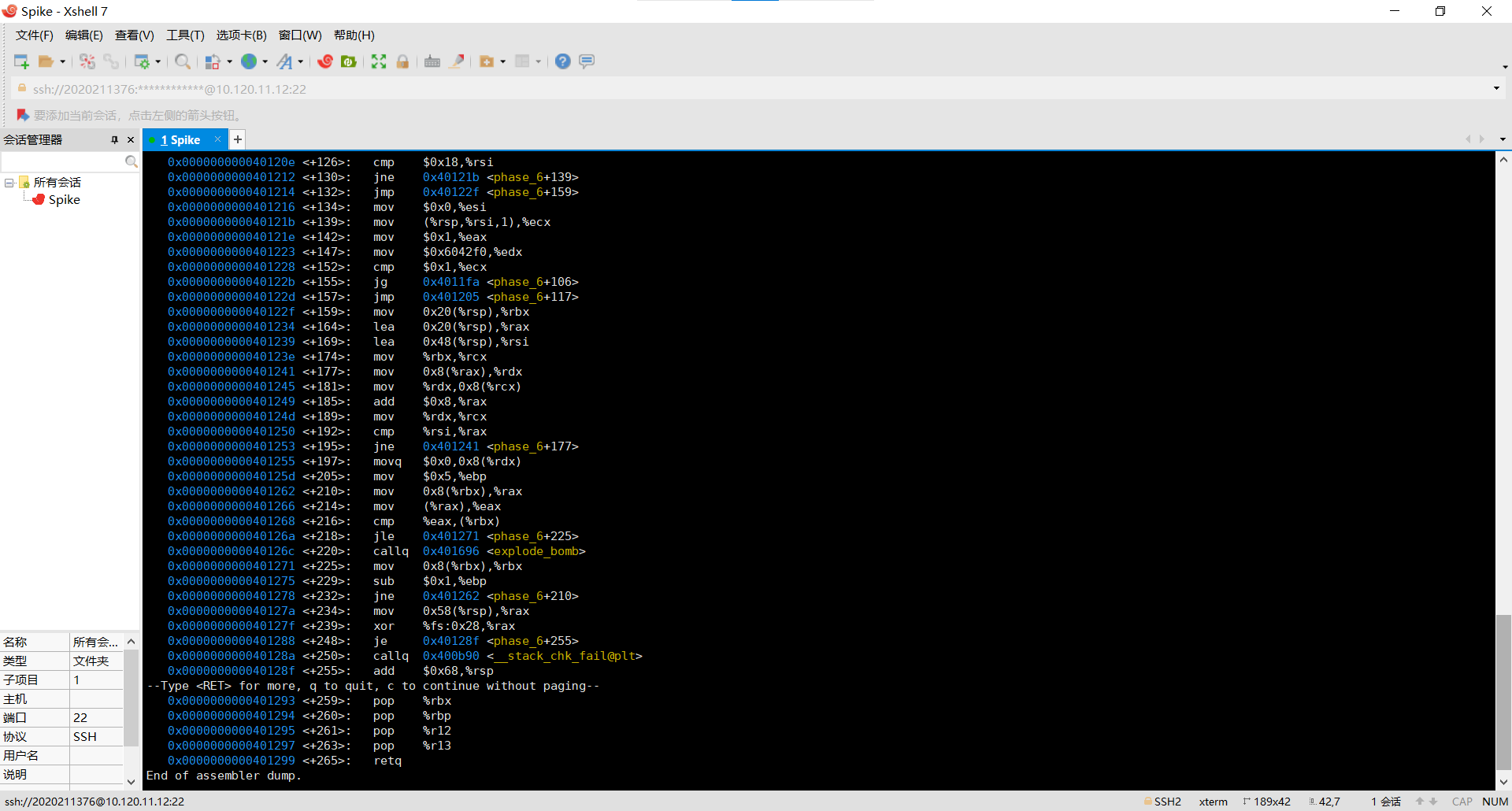
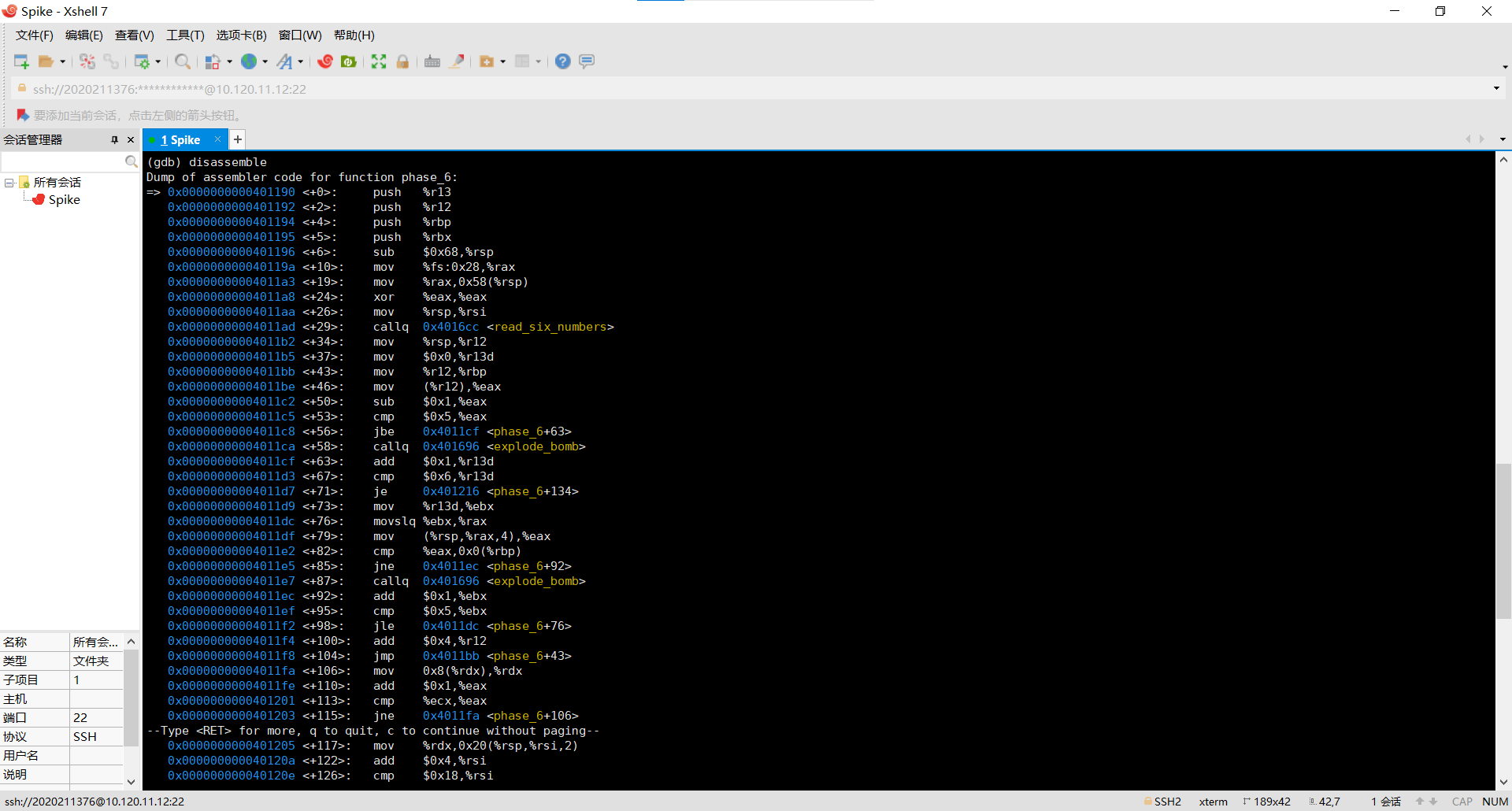
**第一个数**：起始值5，标志着要从5开始找；

**第二个数**：把路过的数组值相加之和，为115。

**3. 增加的理解**

* + 认识到数组在机器中的存储方式，以及**lea**指令对数组的重要性。
  + 加强了对数组的理解。
  + 加强了对循环的理解。

### phase\_6

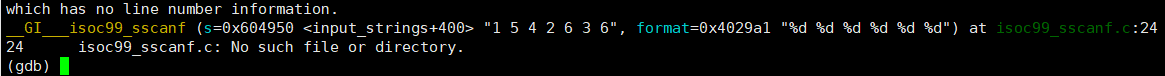


phase\_6\_1 phase\_6\_2

**1. 初步理解**

phase\_6显然是个难题。一看，这么长！

先看输入数据：



**“%d %d %d %d %d %d”**，显然，他让我输入六个数字。

初步观察代码：

**sub $0x1 %eax**

**cmp $0x5 %eax**

意思显然是输入的第一个数不能大于6。

不然而然想到我们的输入应该是小于等于6的整数。

**2. 寻找答案**

第六题分三个部分：

**section\_1**：

检验所有数据 <6且不相等；

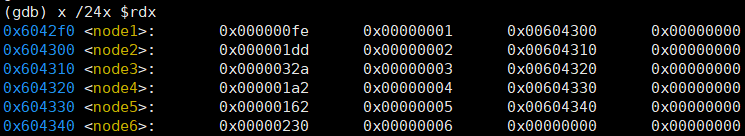
**section\_2**：

初始化地址和数组，用我输入的数据**将地址存到新的空间里**；

**section\_3**：

通过新数组比较大小，来判断新数组里**地址对应的值是否是从小到大排序**；即检验排序结果是否与 输入匹配。

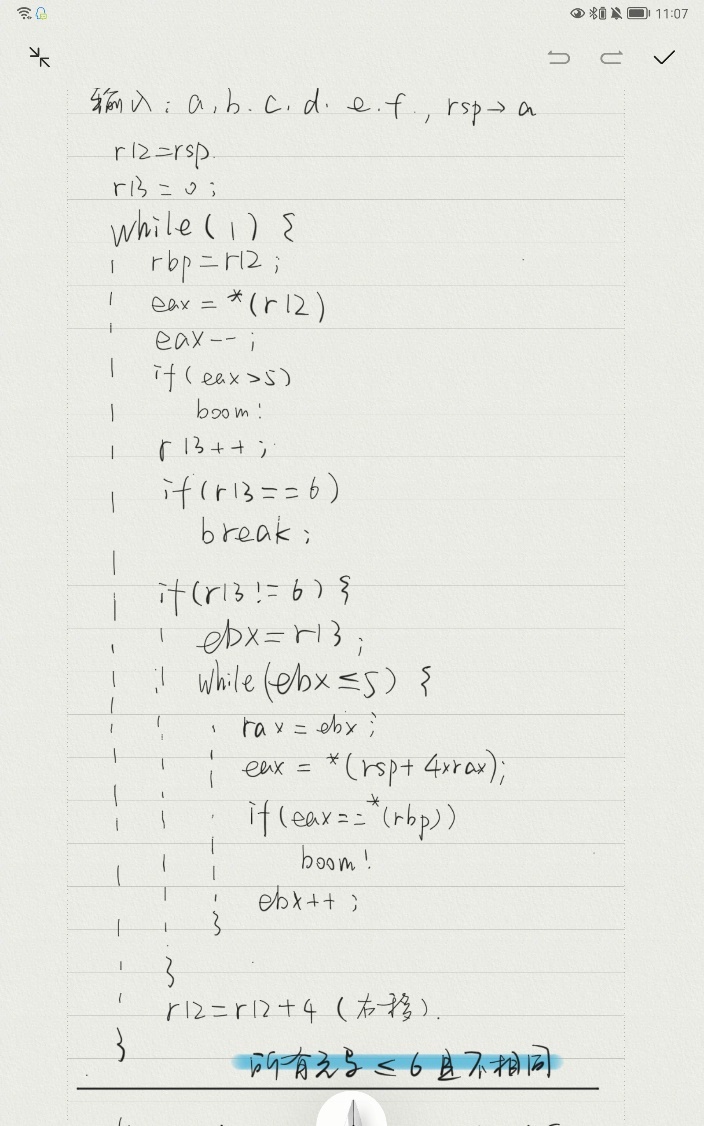
此外我将已经弄好的但没有排序的数组**数据**记录了下来，具体如下：



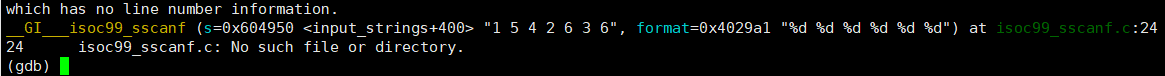
显然，虽然存储空间相邻，但这并不是一个严格的数组；因为每个对应的空间后面会跟着一个地址。

这是一个**链表排序题**；不过他的方法比较特殊。下面详细分析上述三个过程。

**section\_1：检验输入数据**



首先，输入格式如下：

****

所以我们的输入应该是6个数字。

第一个部分的主要功能有两个：

**1**. 第一个输入要小于等于6；

**sub $0x1 %eax**

**cmp $0x5 %eax**

所以第一个输入不可能大于6，否则直接爆炸；

**2**. 所有输入不能相等。

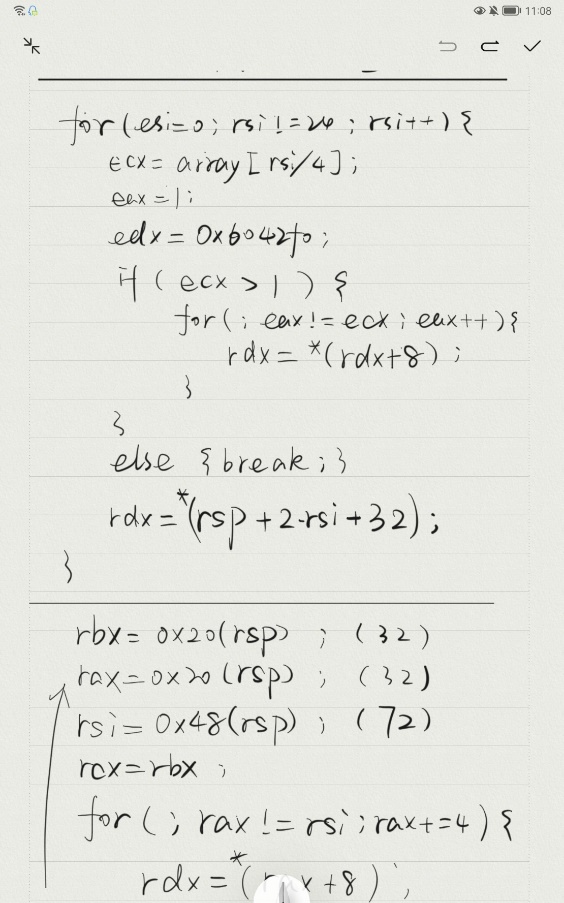
循环计数6次，每个循环内与后面几个数逐个匹配， 若相等则爆炸。

根据这两个限制，我们很容易想到：

我们的输入是123456六个数的排列。

至于正确性，我们得知道其中真正的含义之后才知道。

**section\_2：创建新的地址数组**



首先有个指令是**将0x6042f0赋值给edx寄存器**。

这是干什么呢？这是为排序开栈，**0x6042f0是排序首地址**。

**前提**：



这是我的六个输入，被存在了**rsp**为首的连续空间中。

这就是我在这个循环中操作地址转移的关键。

（注，最后一行赋值代码写反了）

**实现过程：**

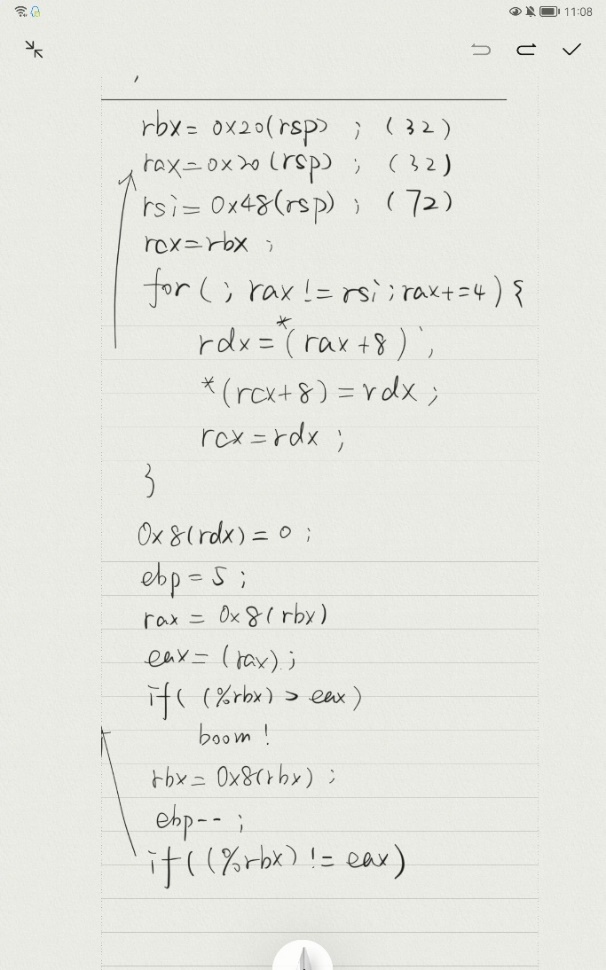
**Step 1**. 开辟新的临时变量以便于遍历我的输入数组；

**Step 2**. 每次循环找到我对应的输入数字，并将其赋值给**ecx**；

**Step 3**. 通过**ecx**为位移量找到原来对应的链表的相应地址；

**Step 4**. 将**得到的地址挨个存入到开辟空间中**。

**section\_3：根据地址数组重新修改指针域**



这段代码主要功能如下：

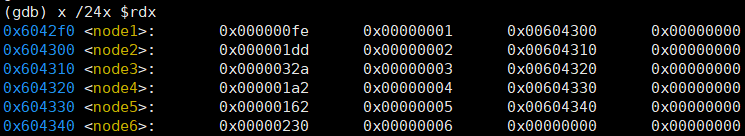
**1**. 通过双重循环：第一个循环找到对应要修改的指针域；

**2**. 将对应指针赋值回给原来的链表；

**3**. 判断当前元素是否小于指针域指向的元素（即下一个元素）；若 大于，则直接爆炸，调用explode\_bomb函数。

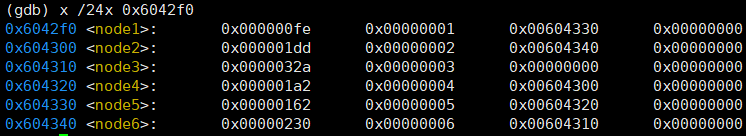
经过观察代码，知道是从小到大左右排序。

我们的输入是正确的元素顺序对应的原来链表中元素的位置



所以正确的答案是 **( 1 , 5 , 4 , 2 , 6 , 3 )**。

**运行结果**：（显然已经排列好了，**修改的是指针域**，且进行的是类冒泡排序）



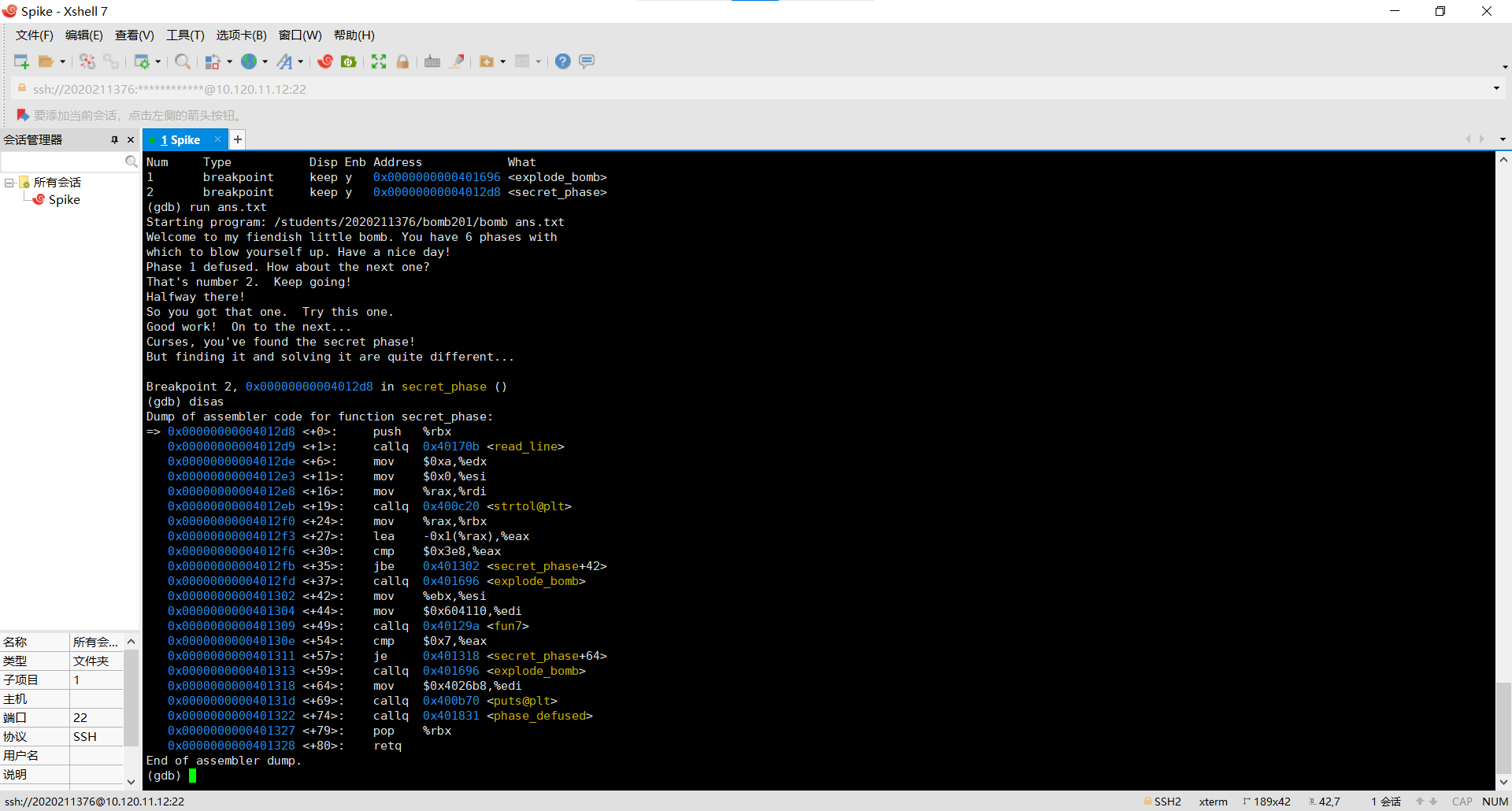
**3. 增加的理解**

* + 了解了链表的机器级存储结构，并且理解了链表的机器级相关操作；
  + 熟悉了各类跳转操作，熟悉了循环操作。这题三个大循环，看循环的能力直接上了一档次；
  + 理解了rsp的操作，即临时开栈，存储返回地址等。

## 阶段三：secret\_phase

### 找到进入方式

首先，我知道有这个难题，但是找不到进入方法。它在**phase\_defused**中可以被调用。



**思考过程：**

**Step 1:**

我想过在**第三题换着七个答案输入**，想着**有一个能触发**，但是都没有进入。

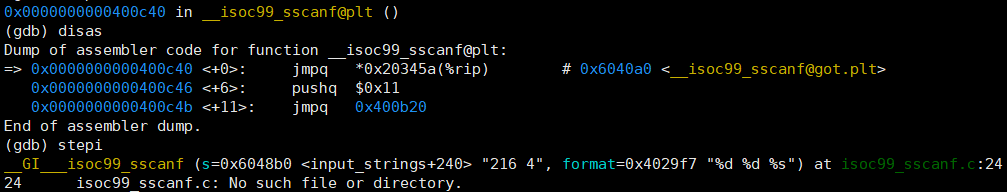
那我还有什么方法可以进入呢？

看见**secret\_phase**中有**read\_line**。那我在第六题后面输入，会有什么发生吗？

我输入一个6，就过了一个判断！

**Step 2:**

然后跟着找，在寄存器里找了个字符串，并且提醒我在第四题后边加上：



scanf\_secret\_phase

 **“邪恶先生”**

根据这个，再联系我们要求的输入”%d %d %s”，就可以得到答案了：

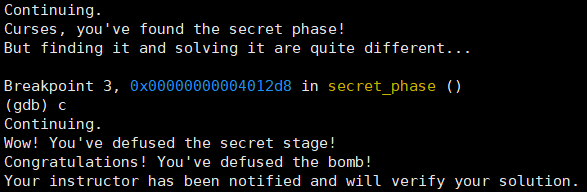
它**要求我们在第四题之后加上DrEvil**。

于是，过了第二个关卡。

**Step 3:** 来到最后的输入secret\_phase答案了。

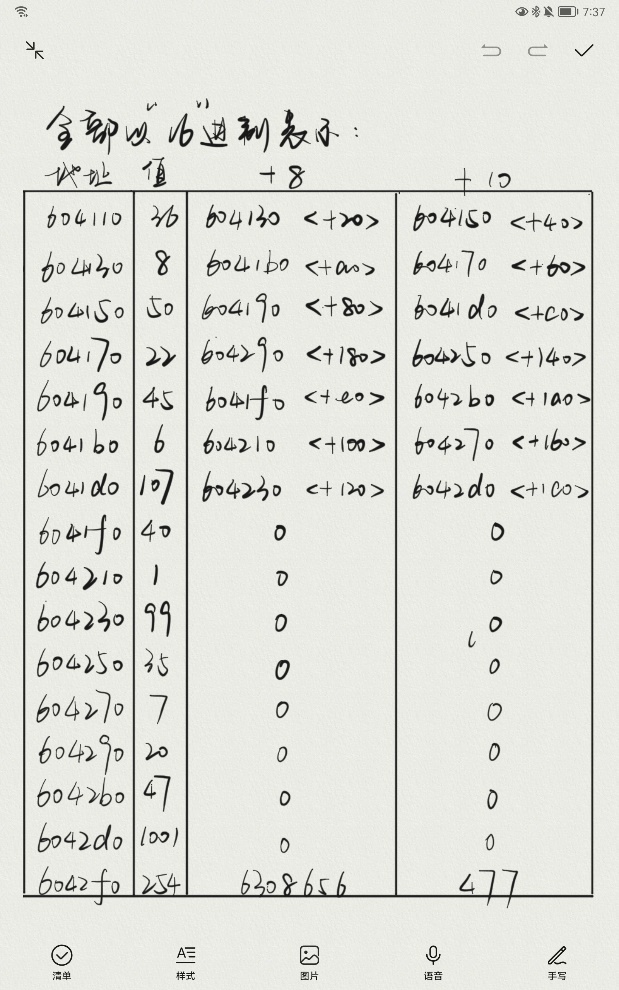
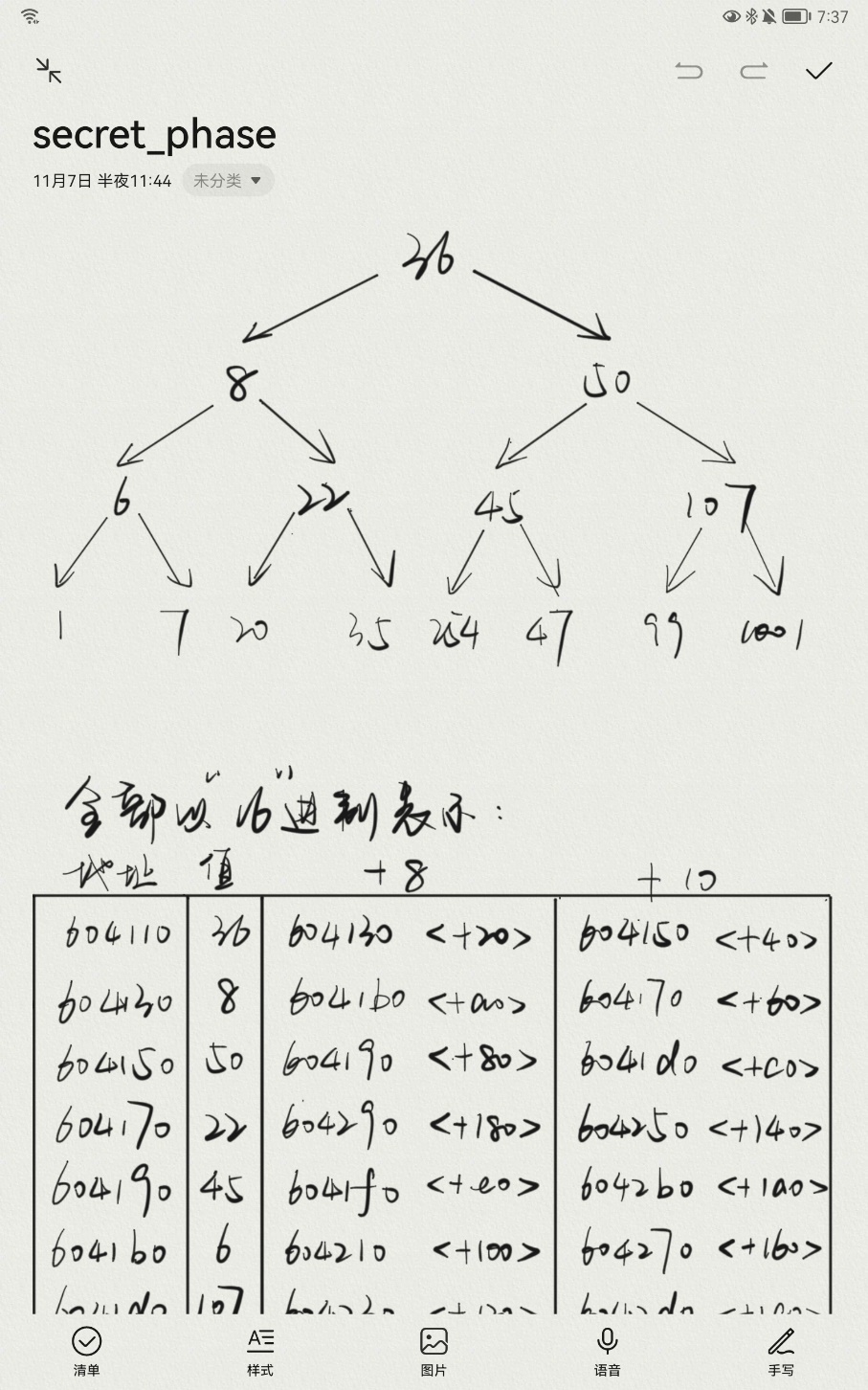
### 寻找secret\_phase答案

直接放图，这题我做了蛮久的：



这是我们正确输入后会出现的输出！

然后这是手写稿：

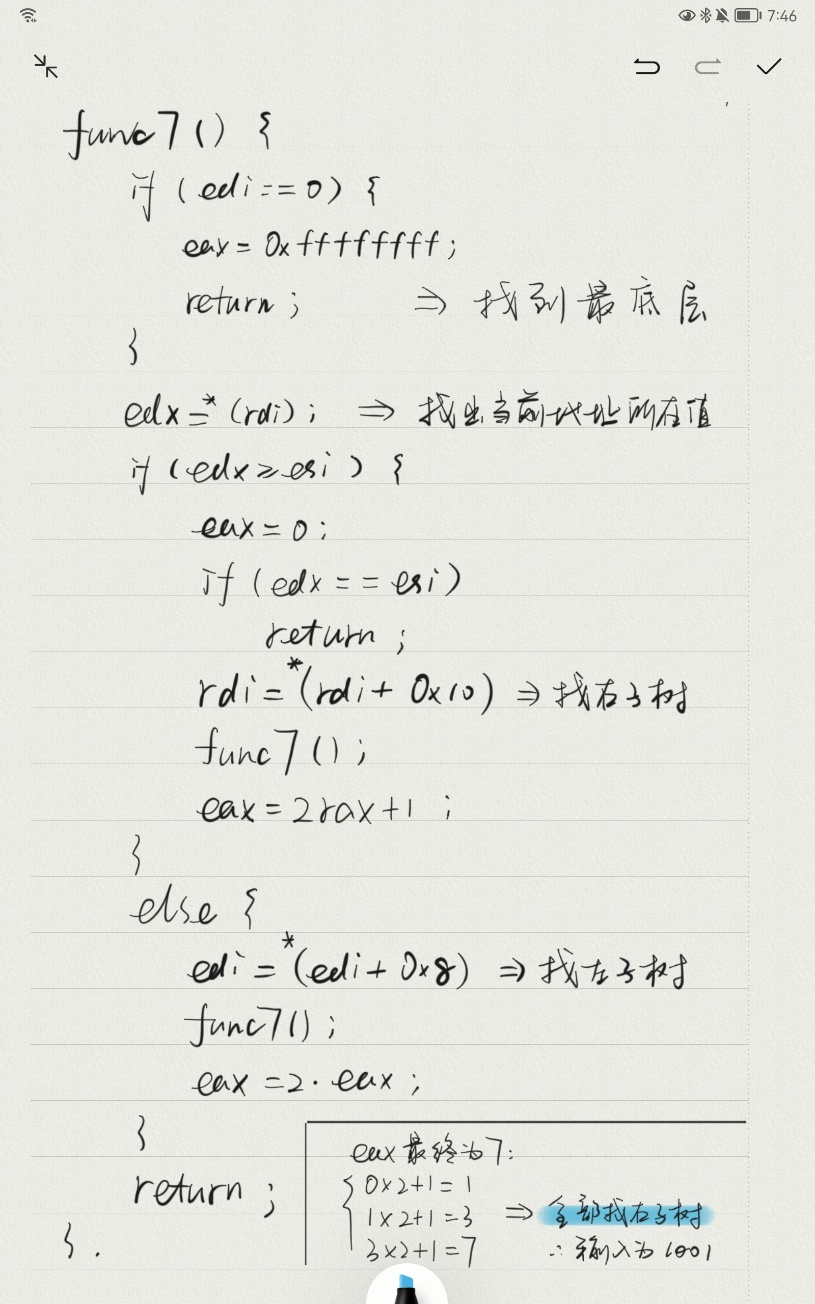
 secret\_phase\_二叉树

这是我在寄存器中发现的一个很奇怪的空间。

然后我把它抄下来后，经过链表的知识发现：

**它是一个二叉树！**

那他这是叫我干什么呢？接下来就是阅读代码了。



阅读代码，我马上接理解了代码的意思：

这个程序的意思就是，**要找我们输入的数**。

但会有一个初始值为0的变量：

1. 往**左子树**找一次：**\*2**

2. 往**右子树**找一次：**\*2+1**

而我要找的**立即数是7**，但二叉树只有四层，能有三 次计算的机会，我们需要把变量变成7：

**( ( 2\*0+1 ) \*2 +1 ) \*2 +1 = 7**

所以我要一直找三次右子树，找到第四层。

**找的步骤：**

**1.** 根据我画的二叉树，我要找到1001；

**2.** 如果数字匹配，则返回；

**3.** 数字比要找的数小，则找左子树；

**4.** 数字比要找的数字大，则找右子树。

**所以我答案是1001。**

secret\_phase

、

# 五、总结体会

**遇到的问题&解决方法**

**Q1**：一开始不会阅读汇编代码。

**A1**：经过2\*7题的打磨，应该已经对机器及的指令表示比较熟悉。

**Q2**：不知道对应的寄存器的功能，以为只是和一般的临时变量一般。

**A1**：**rax**为**返回值**；

**rdi rsi r12 r13 r14 r15** 为**被调用者寄存器**，可作为临时变量的作用参与子函数运行。

**rsp**为**栈帧**，记录函数的返回地址，并且可以在上边开辟空间存储临时变量。

**Q3**：**gdb**工具用不熟练，不知道该如何合理设置断点。

**A3**：失败是成功之母，**BOOM！！！**一次之后就小心翼翼，慢慢就会了；

并且要学会将**break**和**continue**合起来用，这样能进行快速的调试。

**Q4**：不知道如何观测想要的值，打印想要的值。

**A4**：方法一：**print value/\*(address)**；

方法二：**x /d(/x/24d) address**；

方法三：**watch varible**。

**Q5**：不清楚循环结构，导致一旦看到跳转指令就头疼。

**A5**：啃书，了解了循环体结构在机器级指令的基本实现形式。

**Q6**：不能很好的理解递归函数，尤其是关于传参和变量。

**A6**：做第四题：**eax**是**返回值**；

**pop/push的寄存器**在该次函数中可看作**临时变量**；

**esi edi等寄存器**可看作**默认传入参数**。

**Q7**：不熟悉数组、链表、二叉树等数据结构的机器级表示；

**A7**：在做题中逐渐熟悉：

第五题：熟悉了**数组**的存储方式和使用方式

第六题：熟悉了**链表**的存储方式和使用方式，尤其是如何修改指针域；

第七题：熟悉了**二叉树**的存储方式和遍历方式（与链表如出一辙）。

**Q8**：不了解地址的妙用，尤其是在遇到lea指令的时候；

**A8**：一般对数据的间接操作都是用地址实现，除非必须用mov修改值或者做下标参数，否则不会轻易 修改地址里面的值（在**机器级可以把取值看作间接操作，操作地址才是直接操作**）。

**Q9**：不清楚如何找到各个题目的正确输入形式；

**A9**：方法一：进入读取函数看**format**是什么；

方法二：打印相应寄存器（如**esi/edi**）存的**字符串**，一般scanf会根据相应字符串形式读取数据。

**补充：我们的输入都会以字符串形式存在相应位置，所谓的读取都是操作字符串。**

**挫败的感受**：

我连着三四天拆炸弹拆到1、2点；面对**不熟悉的指令**和**操作组合**会感到很烦躁，但又很无力。

**过关的感受**：

感觉自己又行了，直到翻开下一个炸弹。

**实验投入的精力**：

**64h+**；总共2\*7题加两个实验报告。

以x86为例：

1. **前三题**平均**1h**；

2. **后三题**带完全理解所有指令**6h**一题；

3. **进入secret\_phase** **1.5h**；

4. **解决secret\_phase** **4h**；

5. **实验报告12h**。

**建议**：最好把时间延长一点，因为这题需要消化的东西还挺多的。

# 六、诚信声明（不签扣10分）

需要填写如下声明，并在底部给出手写签名的电子版。

我参考了以下资料：

1. 搜索如何打印地址里的字符串；

2. 搜索如何打印链表；

3. 《深入理解计算机系统》书籍中关于机器指令功能的表格。

在我提交的程序中，还在对应的位置以注释形式记录了具体的参考内容。

我独立完成了本次实验除以上方面之外的所有工作，包括分析、设计、编码、调试与测试。

我清楚地知道，从以上方面获得的信息在一定程度上降低了实验的难度，可能影响起评分。

我从未使用他人代码，不管是原封不动地复制，还是经过某些等价转换。

我未曾也不会向同一课程（包括此后各届）的同学复制或公开我这份程序的代码，我有义务妥善保管好它们。

我编写这个程序无意于破坏或妨碍任何计算机系统的正常运行。

我清楚地知道，以上情况均为本课程纪律所禁止，若违反，对应的实验成绩将按照0分计。

（签名）