

**实验报告**



**题目： 拆解二进制炸弹**

**班 级： 2022211313**

**学 号： 2022211388**

**姓 名： 陈韵涵**

**学 院： 计算机学院**

**202 年 10 月 日**

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。

1. 实验环境
2. Ubuntu 2204.2.33.0
3. Objdump命令反汇编
4. GDB调试工具

GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1

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。

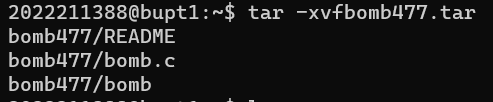
为完成二进制炸弹拆除任务，需要使用gdb调试器和objdump来反汇编bomb文件，可以单步跟踪调试每一阶段的机器代码，也可以阅读反汇编代码，从中理解每一汇编语言代码的行为或作用，进而设法推断拆除炸弹所需的目标字符串。实验2的具体内容见实验2说明。

四、实验步骤及实验分析

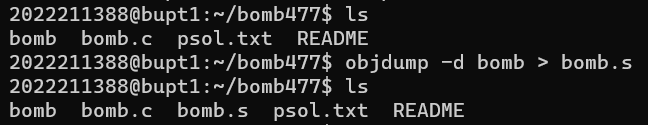
建议按照：准备工作、阶段1、阶段2、…等来组织内容

**准备工作：**

先获取炸弹

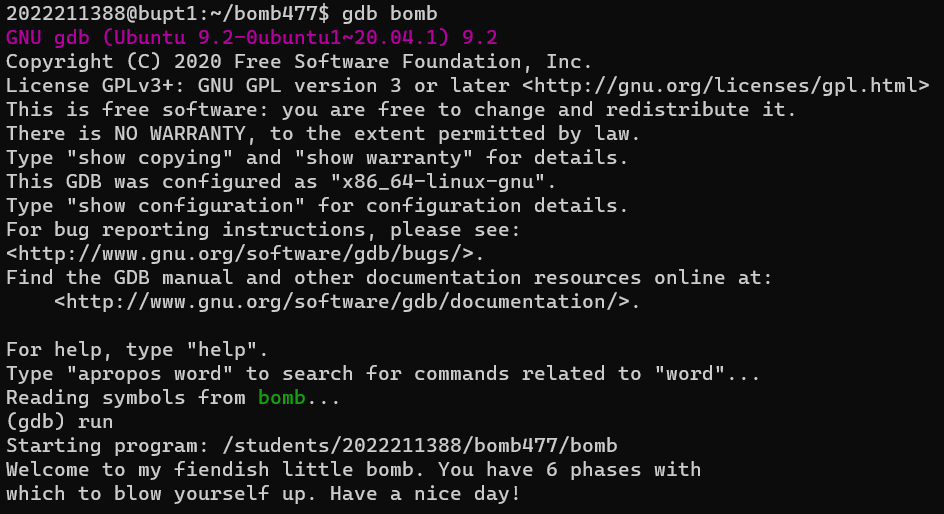


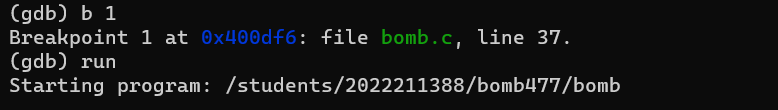
再对可执行文件bomb进行反汇编



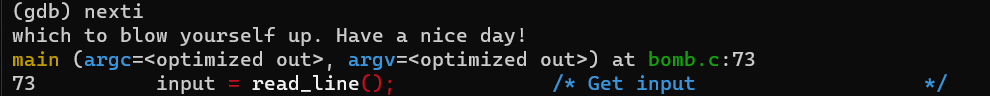
**阶段1：**

首先尝试运行了一下程序，发现要我输入一个东西才能继续运行，于是退出之后打断点





运行程序，发现input是函数read\_line的返回值



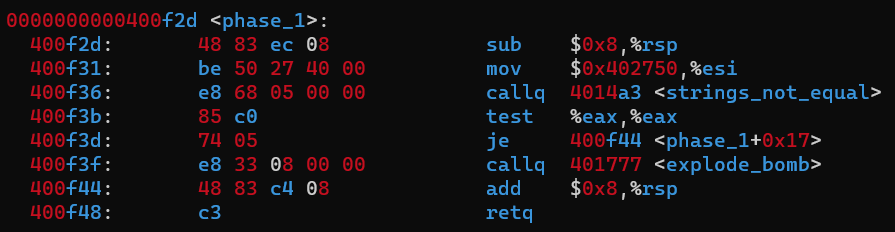
查看main函数，可以看出来我们输入的字符串是存在寄存器%rdi里面的：



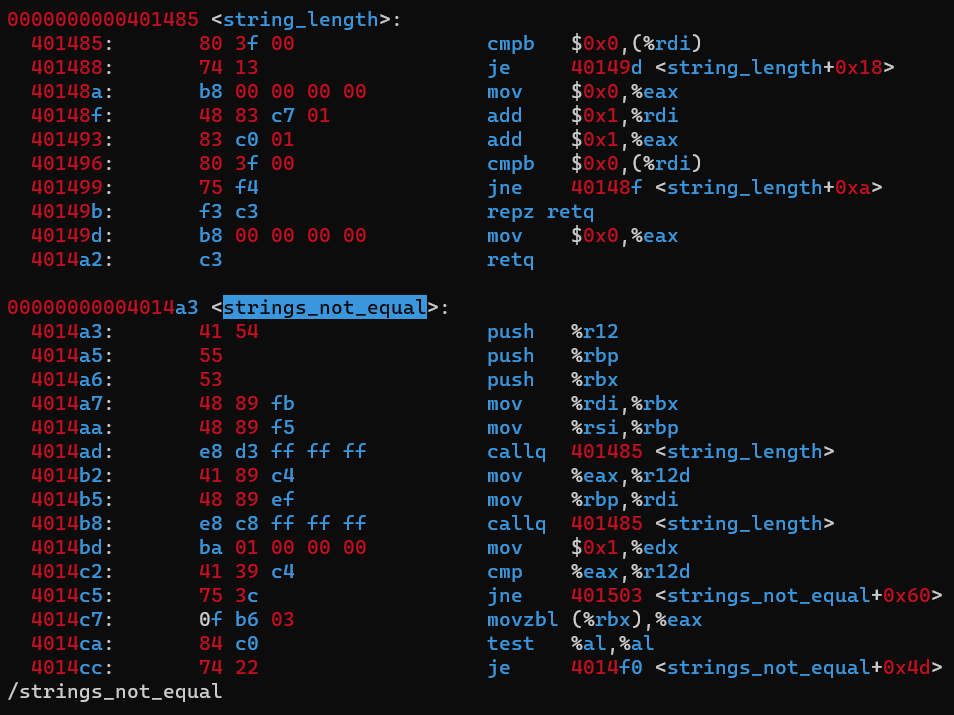
继续运行，发现输入的字符串是一个叫phase\_1的函数的参数：



查看phase\_1函数：

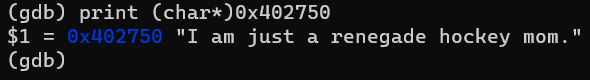


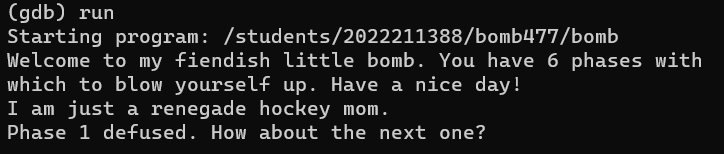
再查看phase\_1调用的函数strings\_not\_equal，可以看出这个函数是用来判断两个字符串是否相等的:



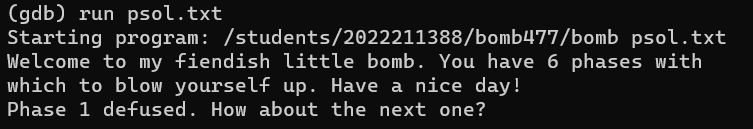
发现这个函数和string\_length函数主要就是用来看%rsi和%rdi里面的字符串是否相等。在phase\_1里面可以看出来，%esi(%rsi)中的字符串地址是0x402750，因此可以判断比较的正是这个字符串和我们输入的字符串是否相等。

利用gdb打印输出该地址的值：



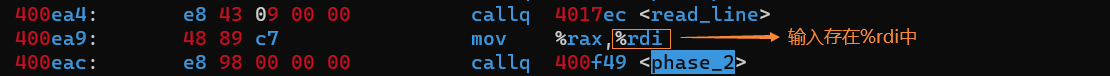


至此phase\_1完成，把答案存在psol.txt里面就不用再重复提交了

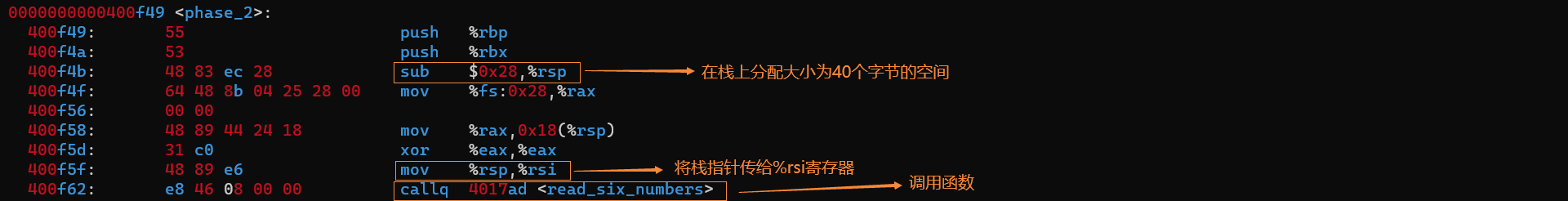


**阶段2：**

先查看main：

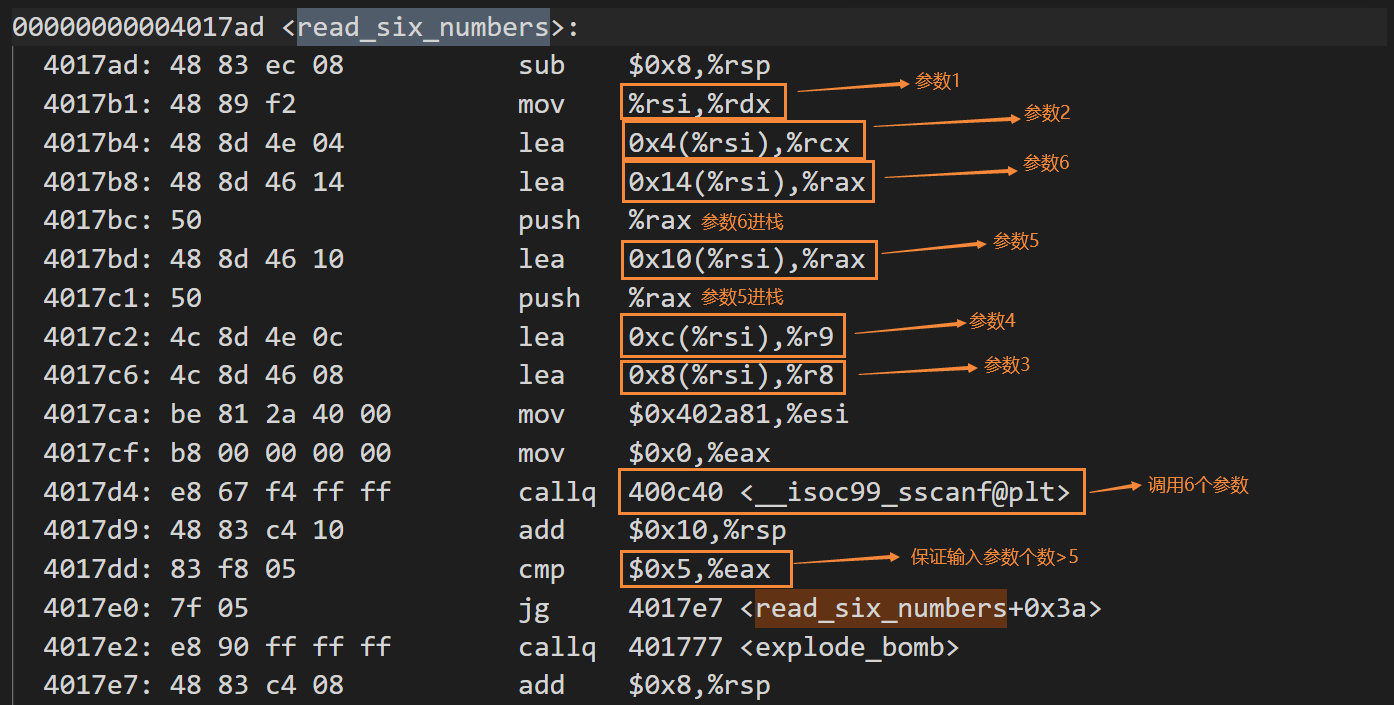


再查看phase\_2和其调用的函数：

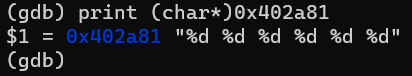


查看read\_six\_numbers函数：

函数首先构造了6个参数：

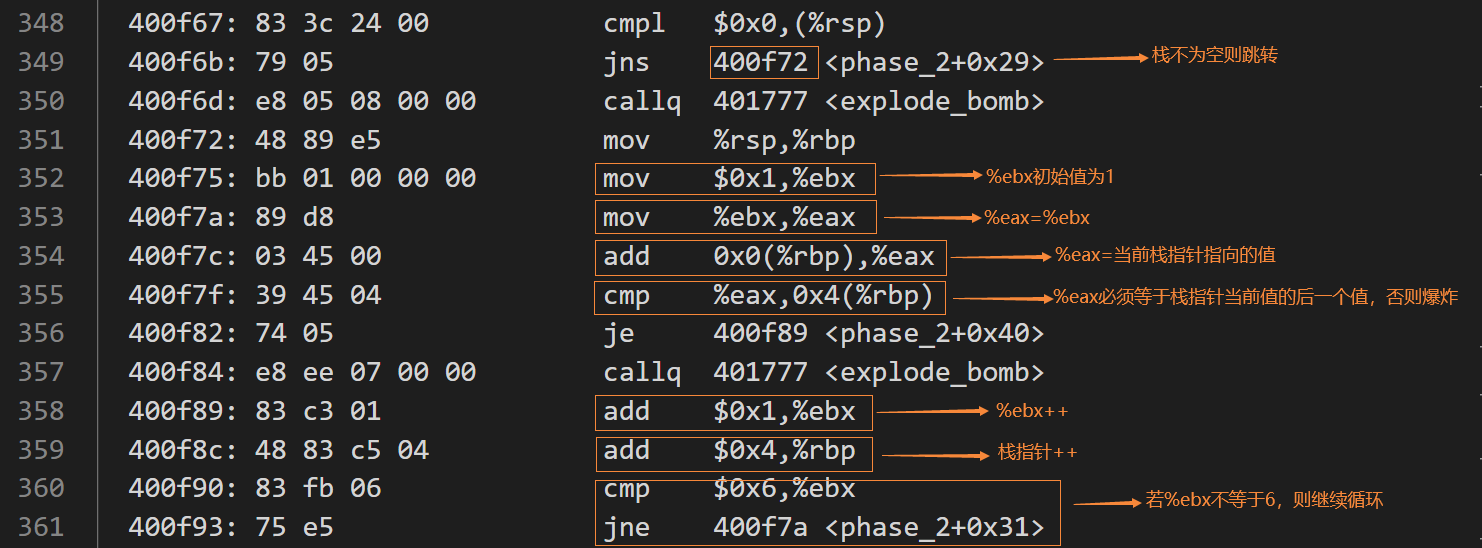


查看0x402a81可知，题目要求输入6个整数：

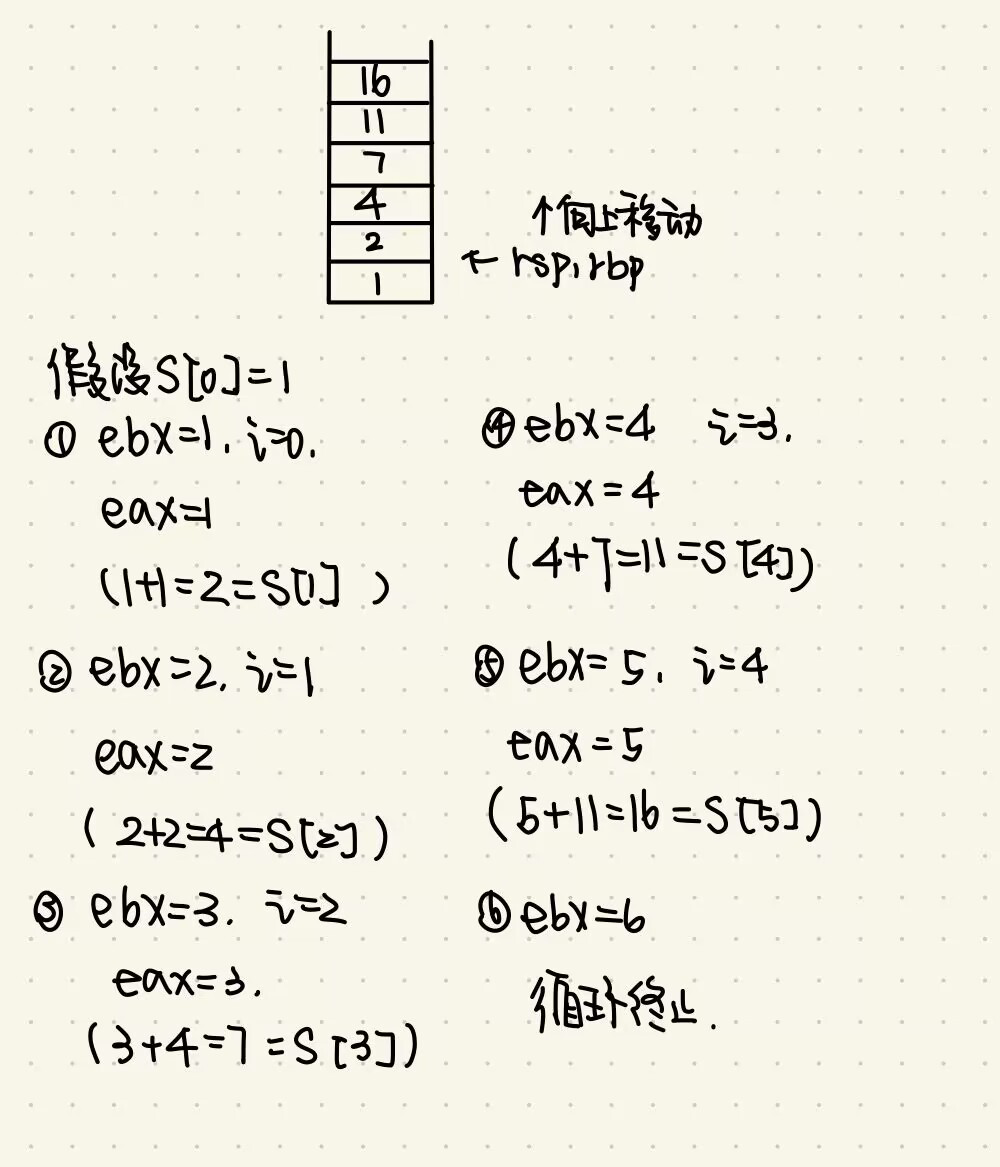
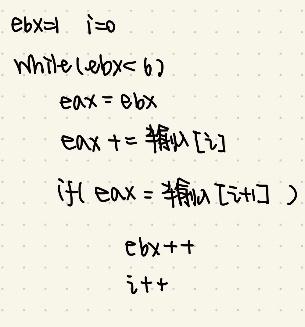


因此这个函数的作用就是接收6个参数并将它们存在栈里面返回给phase\_2进行后续操作。

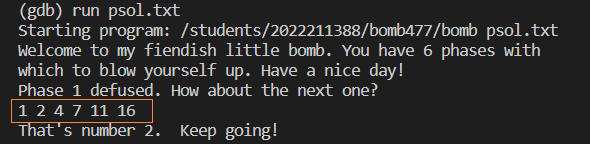
再回到phase\_2，可知后面的代码是一个循环:



简单分析该循环：

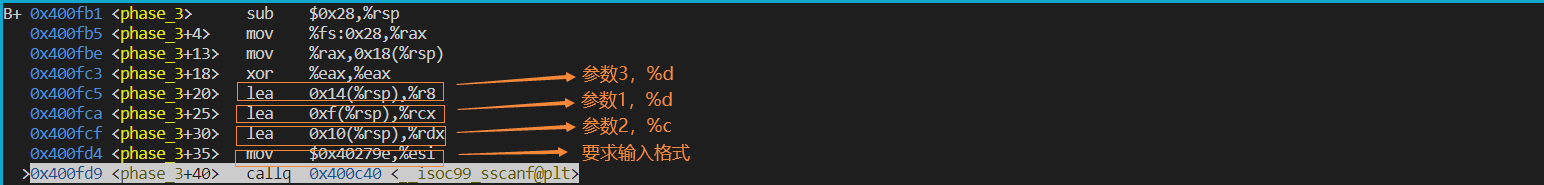


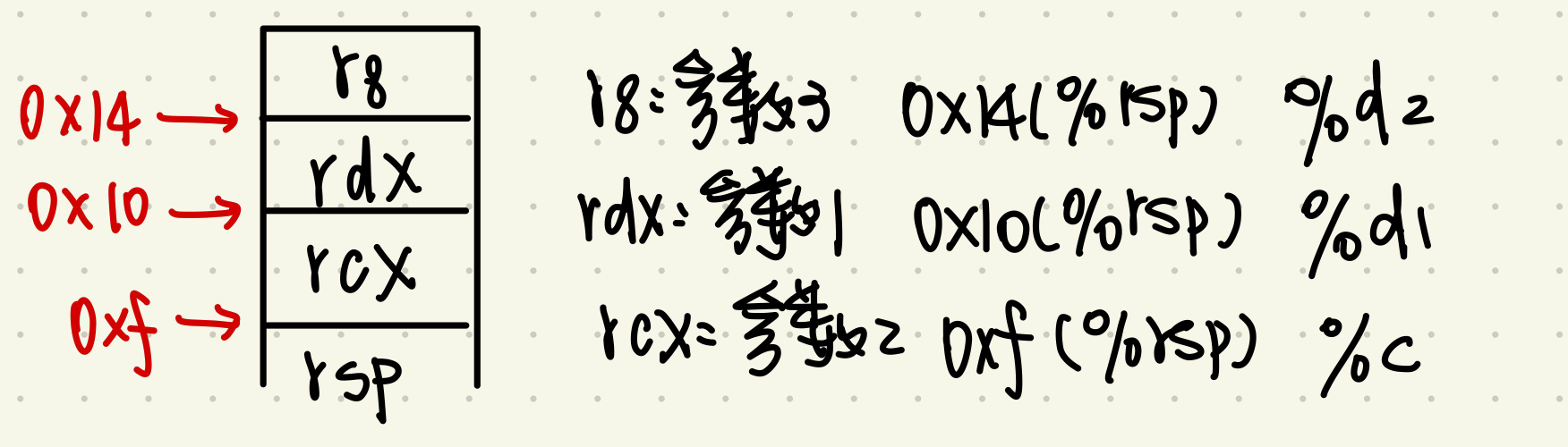
输入答案，答案正确：



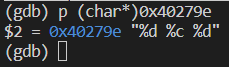
**阶段3：**

分析phase\_3函数：

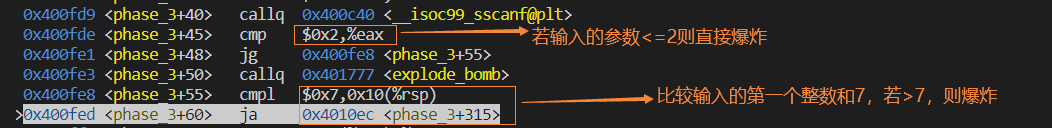




输入格式：



让程序继续执行：



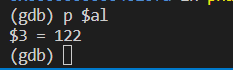
这里就可以得出第一个输入值：小于等于7即可，因此我输入了1。

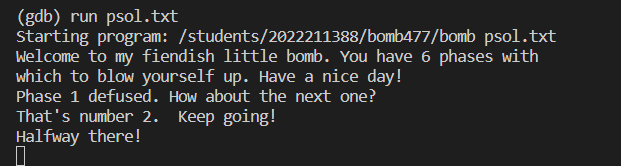
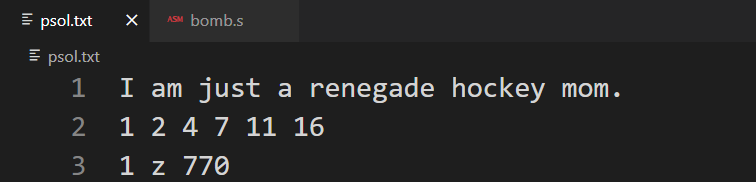


因此第三个输入值是0x302，即770。



打印输出%al的值：

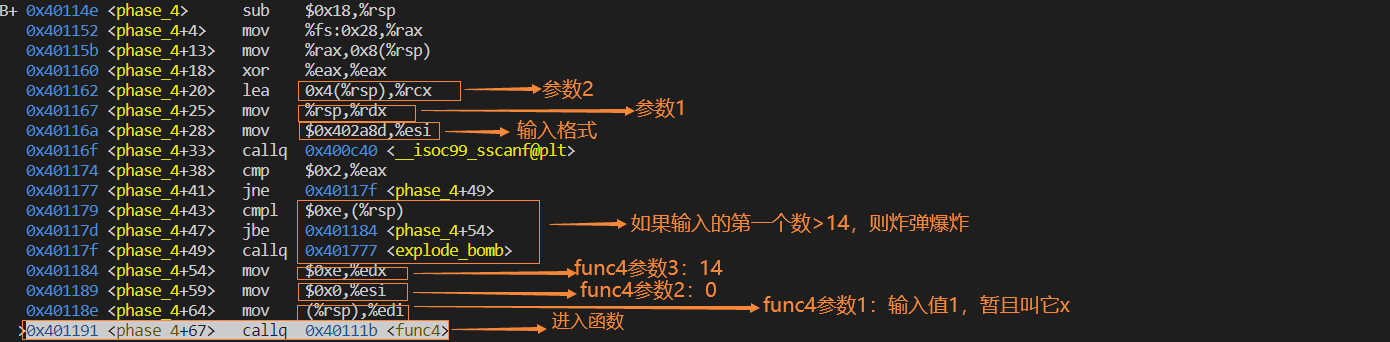
发现是122，查询ascii表可知第二个输入的值是z。

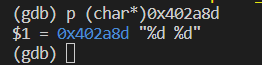


第三个炸弹拆除完毕。

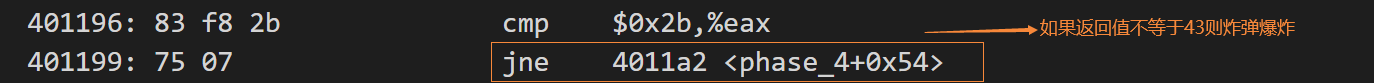
**阶段4：**

分析phase\_4，可以看出这次要输入的是两个整型：

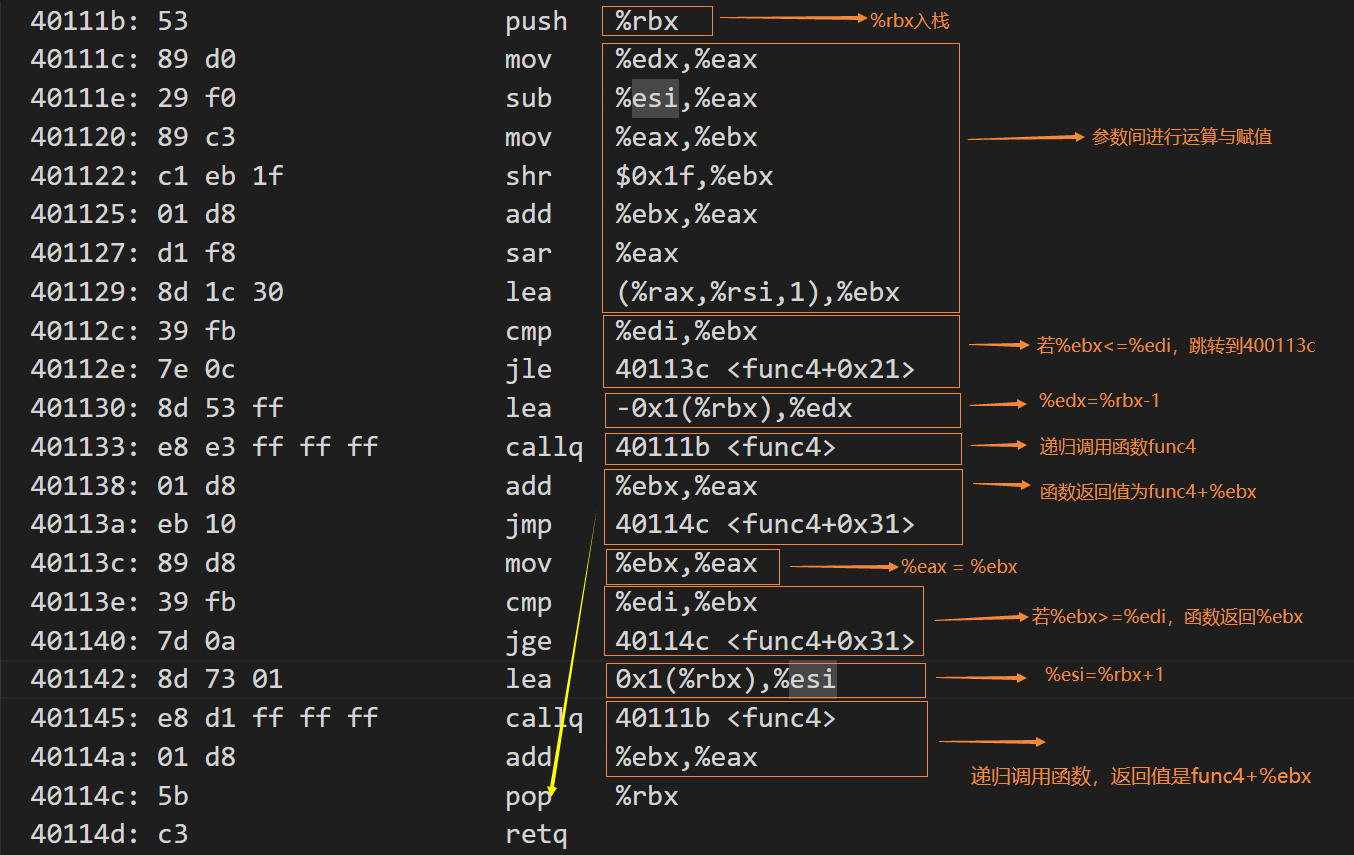


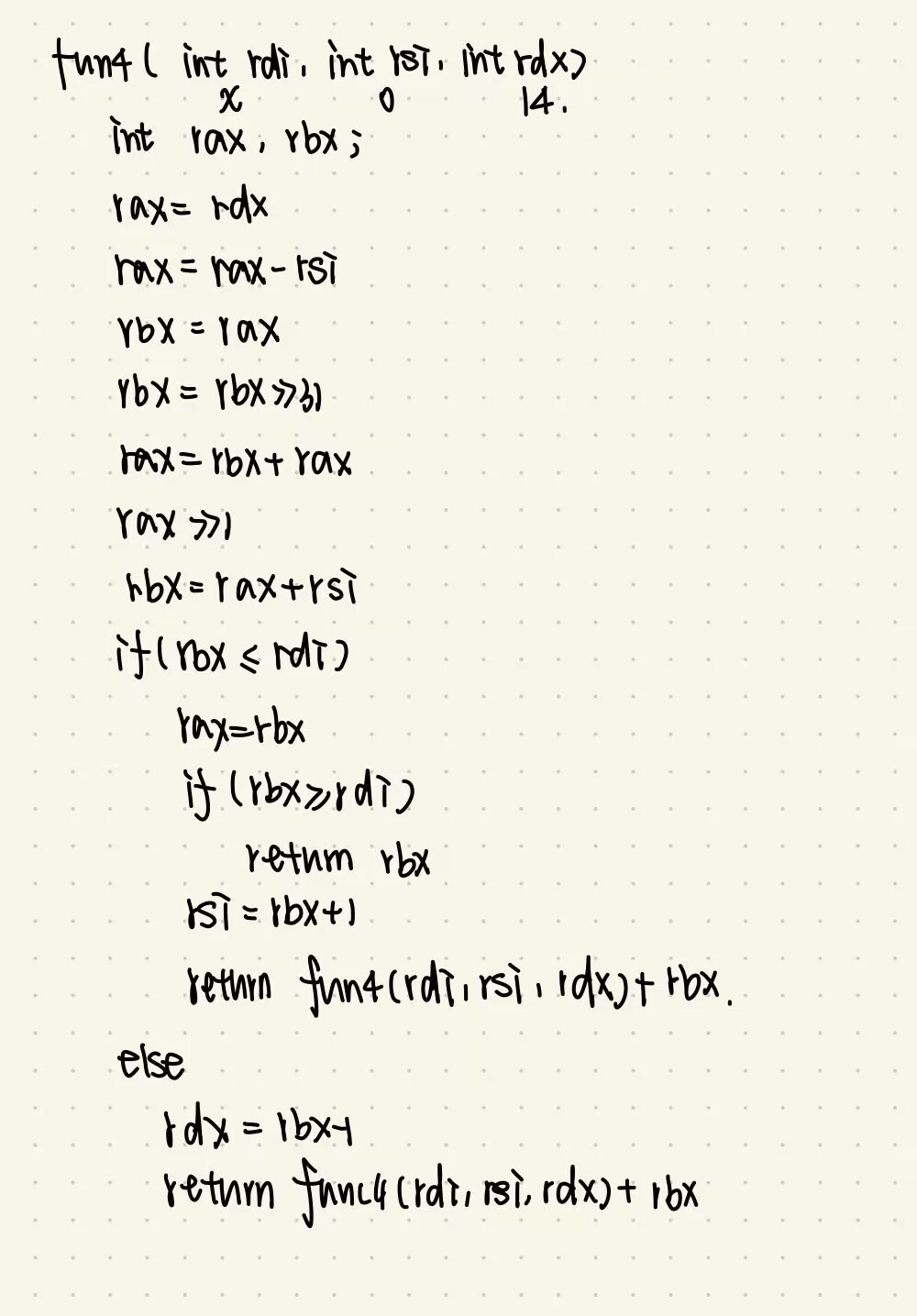
可以得到输入值1的大致范围是<=14

函数返回值为%rax，查看phase\_4后面的一行代码：

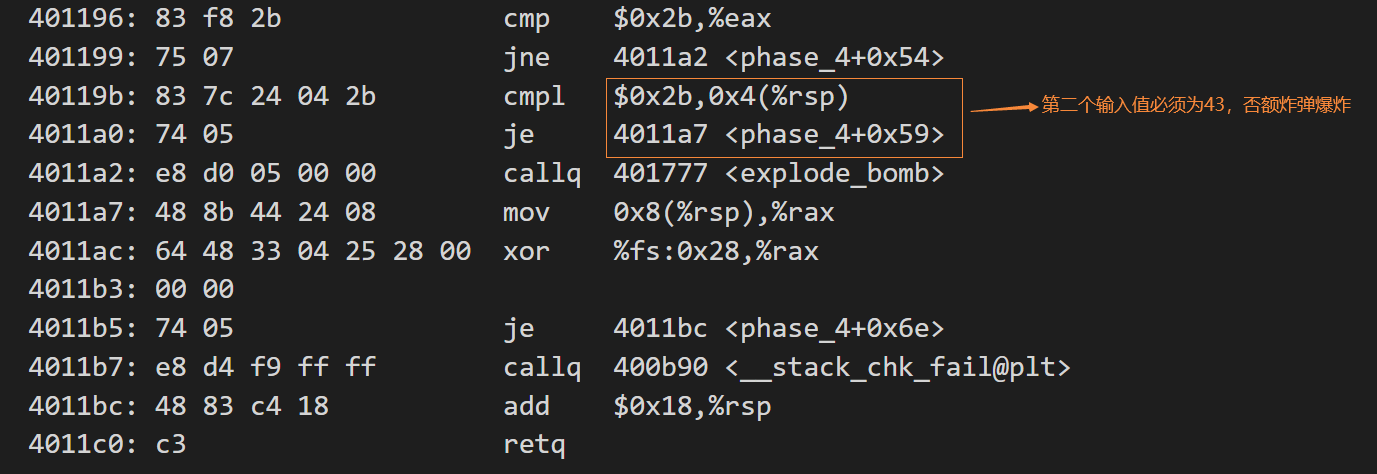


进入func4分析输入值1的具体值：

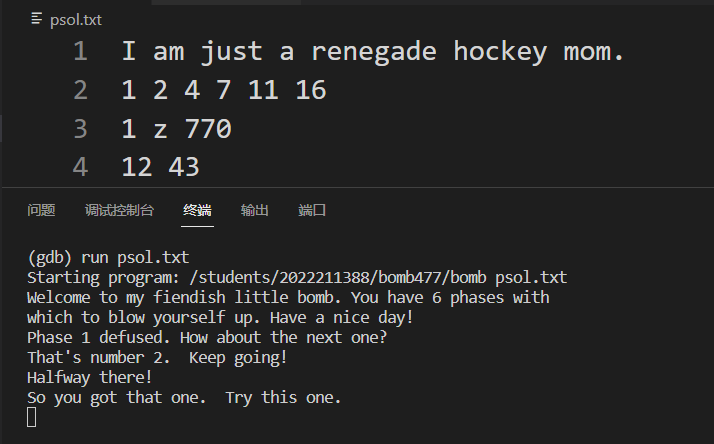


从14开始枚举可知第一个输入的数为12

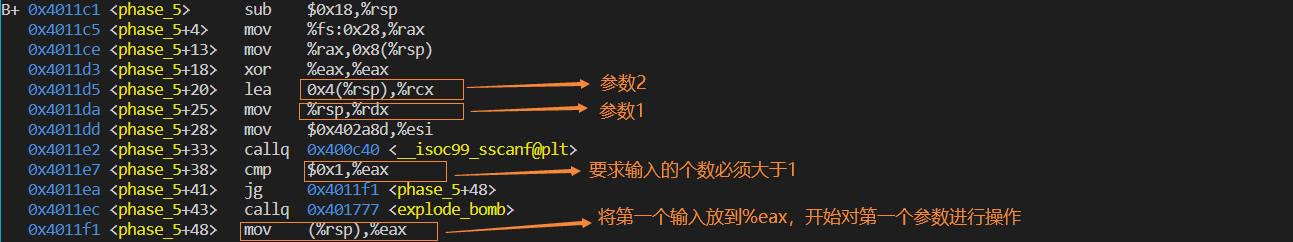
继续看phase\_4:

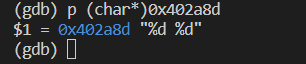


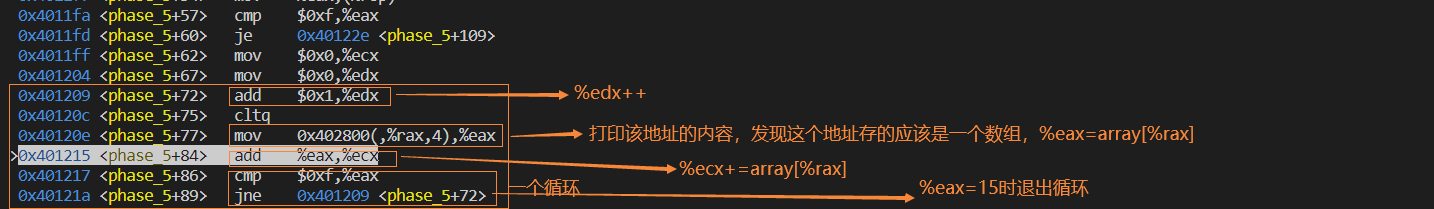
因此输入的数为12 43

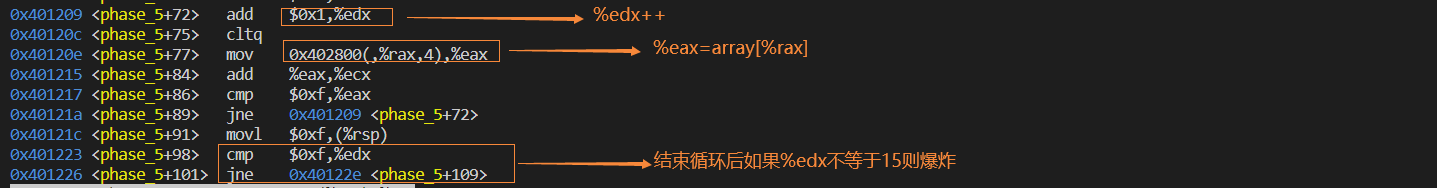
输入正确。

**阶段5：**

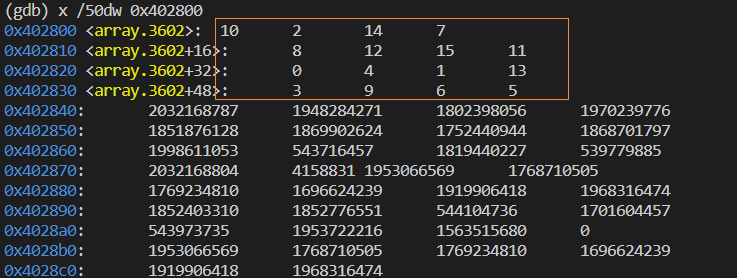
****

可以看出phase\_5要求输入两个整型变量

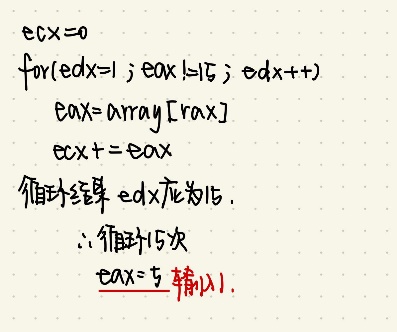
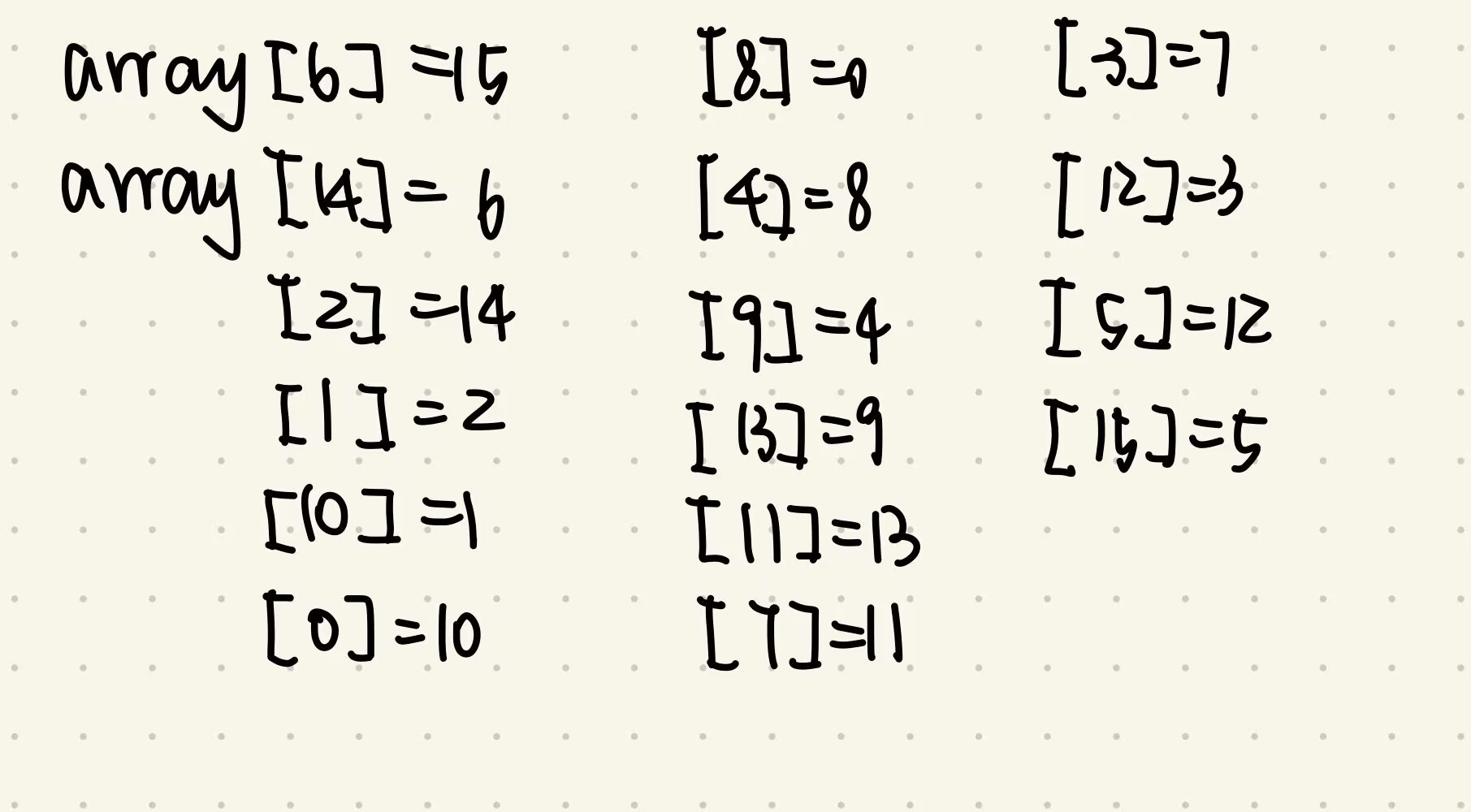




打印出该地址的数组内容：

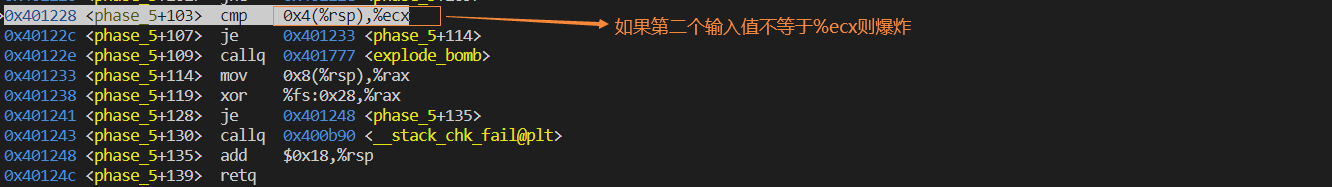
发现应该只有前16个有用

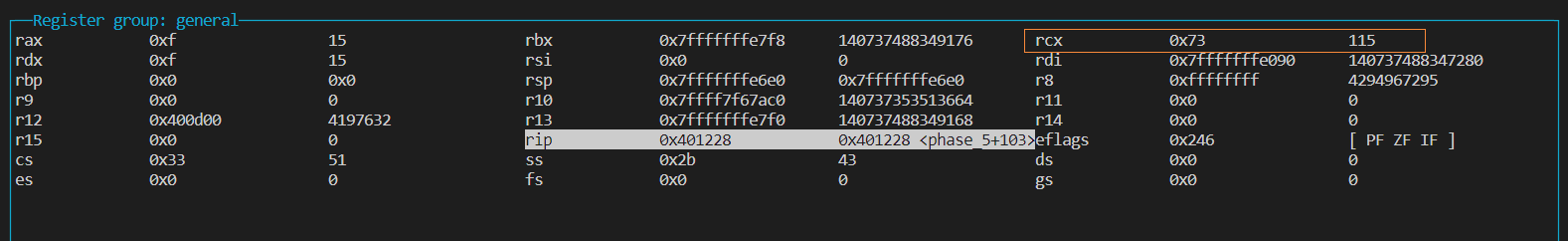
把循环写成代码：

因此第一个输入值为5

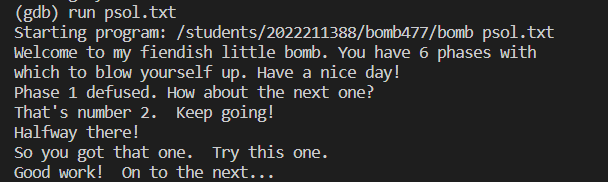
接着看关于第二个输入值的代码：





查看寄存器发现%ecx=115，因此第二个输入值是115。

因此这一题答案是5 115



五、总结体会

总结心得（包括实验过程中遇到的问题、如何解决的、过关或挫败的感受、实验投入的时间和精力、意见和建议等）

1. 把答案写进文件，然后炸弹莫名其妙爆炸多次。

猜测是因为没有换行，而读文件时是一行一行读的，因此会爆炸。

2. 用vscode进行ssh连接

之前一直在终端里面做题，知道这个方法之后做题方便快捷了很多。

3. gdb tui界面

之前一直在gdb和汇编文件里面反复横跳，知道gdb的这个功能之后可以更直观地看出程序进行状态，及时在炸弹爆炸之前终止程序。

4. 善用断点

其实只要断点用得好，炸弹永远都不会爆炸。