

**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**实验名称： 二进制程序分析**

**院 系 ：计算机科学与技术**

**专业班级 ： 计算机202201班**

**学 号 ： U202215365**

**姓 名 ： 叶俊江**

**指导教师 ： 朱虹**

**2024 年 4 月 12 日**

**一、实验目的与要求**

通过逆向分析一个二进制程序（称为“二进制炸弹”）的构成和运行逻辑，加深对理论课中关于程序的机器级表示各方面知识点的理解，增强反汇编、跟踪、分析、调试等能力。

实验环境：Ubuntu，GCC，GDB等

**二、实验内容**

作为实验目标的二进制炸弹（binary bombs）可执行程序由多个“关”组成。每一个“关”（阶段）要求输入一个特定字符串，如果输入满足程序代码的要求，该阶段即通过，否则程序输出失败。实验的目标是设法得到得出解除尽可能多阶段的字符串。

为了完成二进制炸弹的拆除任务，需要通过反汇编和分析跟踪程序每一阶段的机器代码，从中定位和理解程序的主要执行逻辑，包括关键指令、控制结构和相关数据变量等等，进而推断拆除炸弹所需要的目标字符串。

实验源程序及相关文件 bomb.rar

bomb.c 主程序

phases.o 各个阶段的目标程序

support.c 完成辅助功能的目标程序

support.h 公共头文件

**阶段1： 串比较 phase\_1(char \*input);**

要求输出的字符串(input) 与程序中内置的某一特定字符串相同。提示：找到与input串相比较的特定串的地址，查看相应单元中的内容，从而确定input 应输入的串。

**阶段2：循环 phase\_2(char \*input);**

要求在一行上输入 6个整数数据，与程序自动产生的 6个数据进行比较，若一致，则过关。提示：将输入串input拆分成 6个数据由函数 read\_six\_numbers(input, numbers) 完成。之后是各个数据与自动产生的数据的比较，在比较中使用了循环语句。

**阶段3：条件分支 phase\_3(char \*input);**

要求输入一个整数数据，该数据与程序自动生成的 一个数据比较，相等则过关。提示：在自动生成数据时，使用了 switch … case 语句。

**阶段4：递归调用和栈 phase\_4(char \*input);**

要求在一行中输入两个数，第一个数表示在一个有序的数组（或者binary search tree）中需要搜索到的数，该数是在一定范围之内的；第二个数表示找到搜索数的路径（在树的左边搜索编码为二进制位0，在树的有边搜索编码为二进制位1）。

**阶段5：指针和数组访问 phase\_5(char \*input);**

要求在一行中输入一个串，该串与程序自动生成的串相同。在生成串和比较串时，使用了数组和指针。

**阶段6：链表、结构、指针的访问 phase\_6(char \*input);**

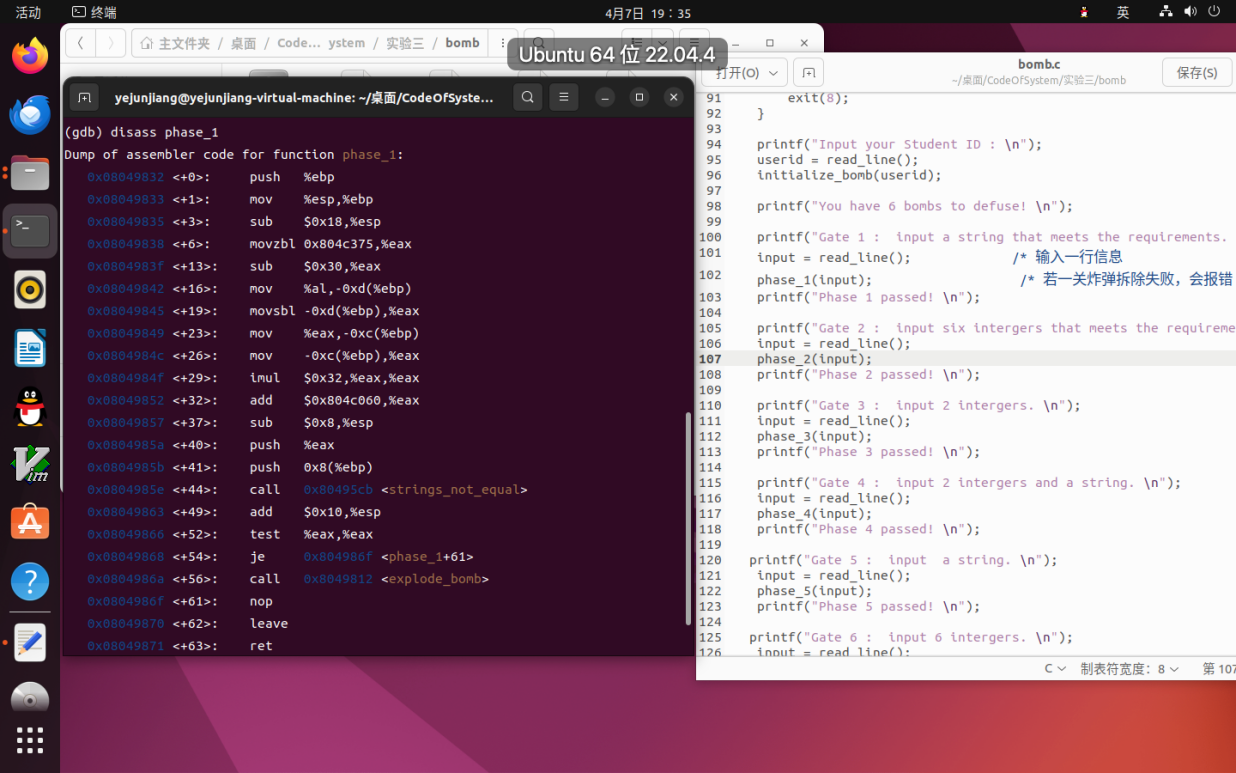
要求在一行中输入6个数，这6个数是一个链表中结点的顺序号（从 1 到6）。按照输入的顺序号，将对应链表结点中的值形成一个数组。若该数组是按照降序排列的，则过关。

**三、实验记录及问题回答**

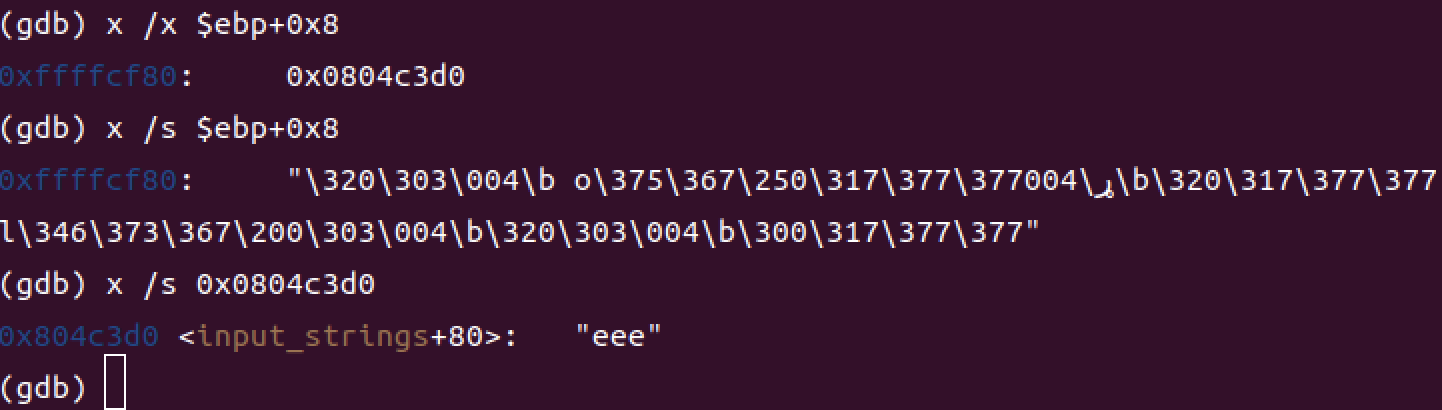
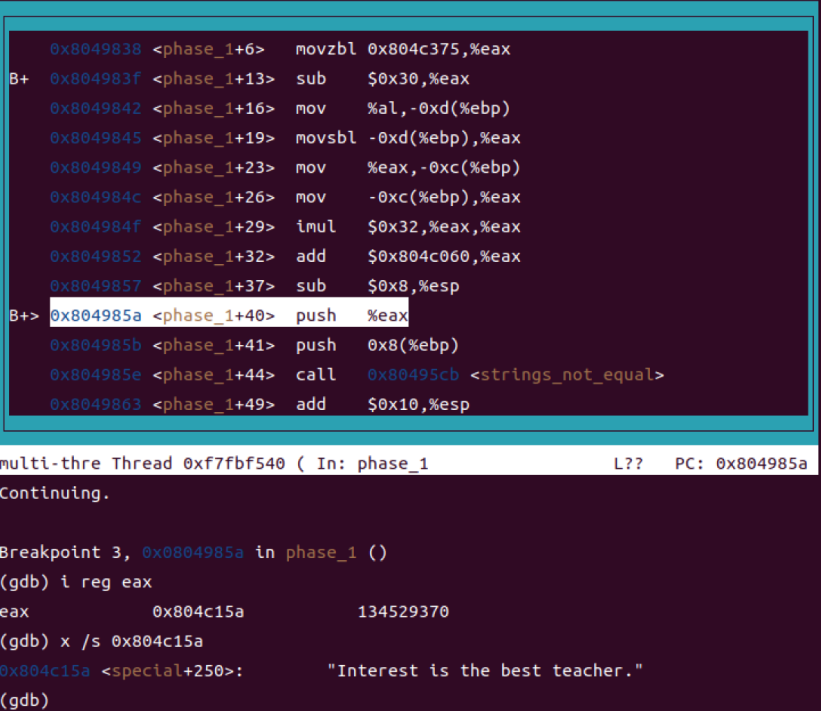
**（1）实验任务1 的实验记录**

**phase\_1:**

先在bomb.c执行phase\_1处设置断点，然后输入指令“r”运行，输入学号ID后程序到断点处。我们试输入字符串“eee”进入phase\_1子程序。输入指令“disass phase\_1”显示程序汇编代码。

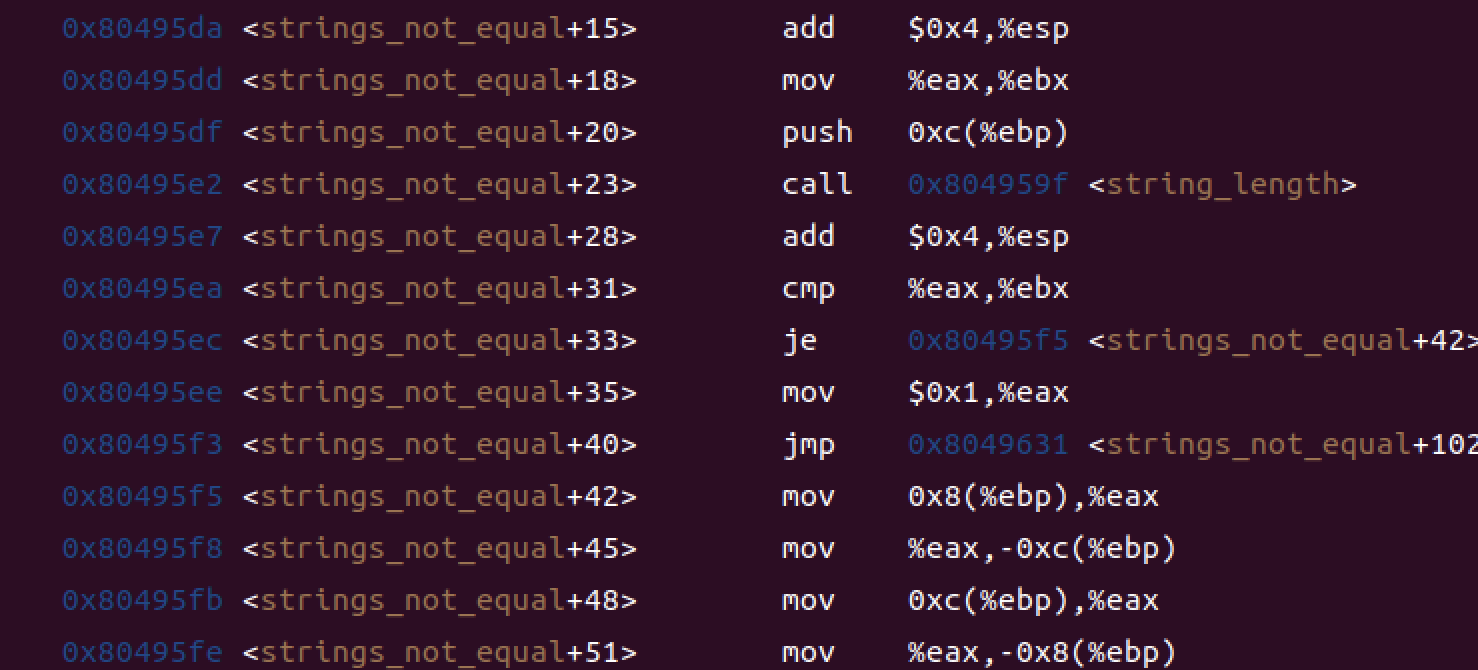


上图中，注意到红框部分代码调用了<strings\_not\_equal>程序，并且传入了两个参数，我们猜想这两个参数与答案相关，我们打印输出这两个参数值。



我们发现%eax中存放的地址指向字符串“Interest is the best teacher.”，0x8(%ebp)中存放的地址指向我们输入的字符串“eee”。我们便可知phase\_1的答案为“Interest is the best teacher.”。

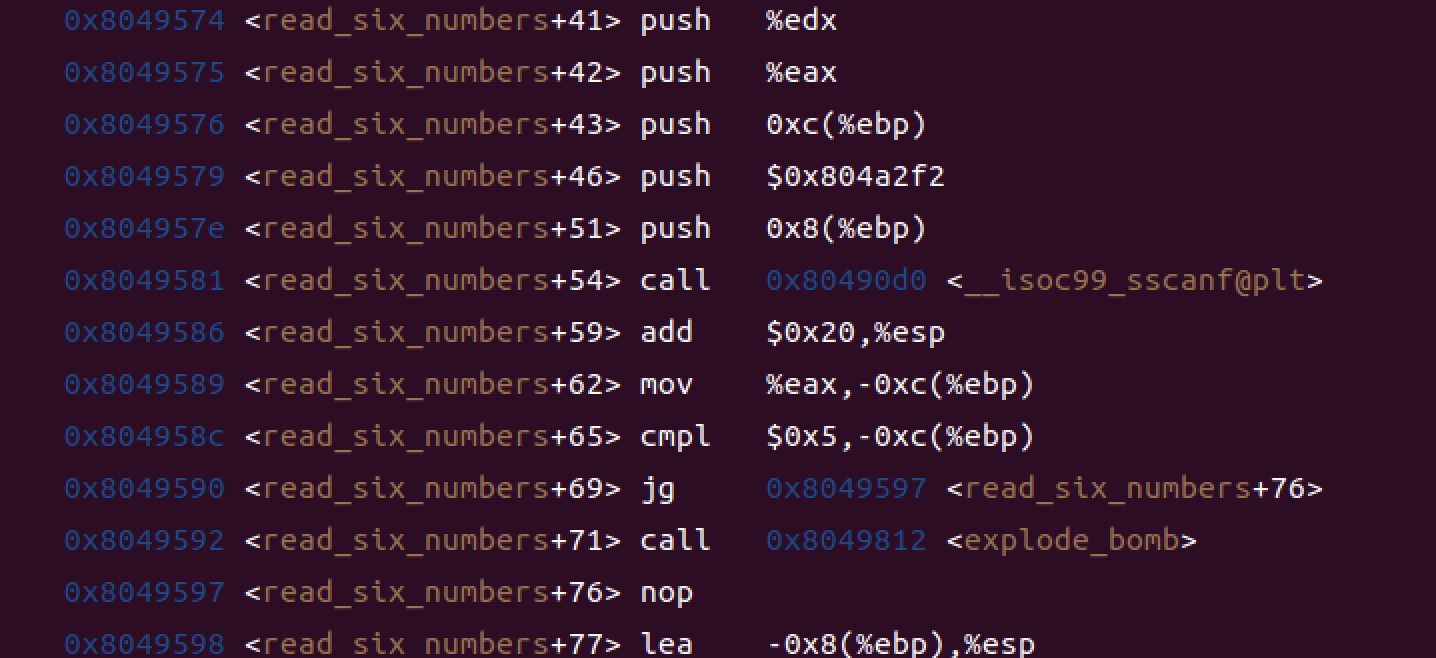
此外，我们可以进一步验证，进入<sting\_not\_equal>程序中发现，程序将我们输入的参数与%eax中存放的字符串进行长度比较。如果长度不相等则跳转到<sting\_not\_equal+102>返回；如果长度相等则在进行下一步比较。分析得phase\_1的答案为“Interest is the best teacher.”。



**phase\_2:**

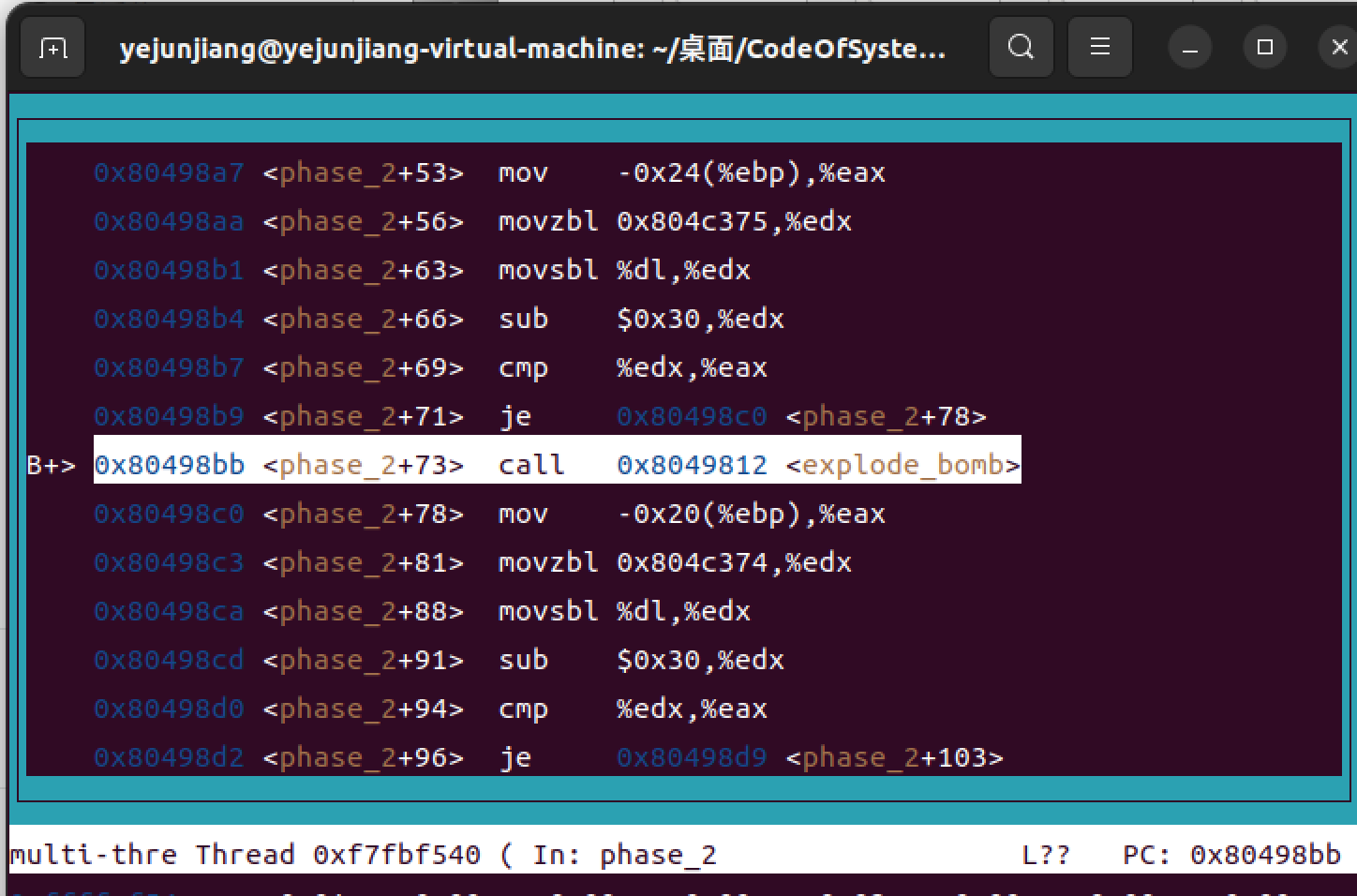
类似地，我们试输入“1 2 3 4 5 6”六个数字进入phase\_2程序。

显示汇编代码页面，我们发现进入phase\_2程序后便调用<read\_six\_numbers>程序，我们查看<read\_six\_numbers>程序。



在上图红框部分中可见，程序将$ebp-0xc与数字5进行比较，根据前后代码可知这里的功能是检测输入框中输入了多少个数字，如果数字小于等于5则调用<explode\_bomb>程序，炸弹爆炸。

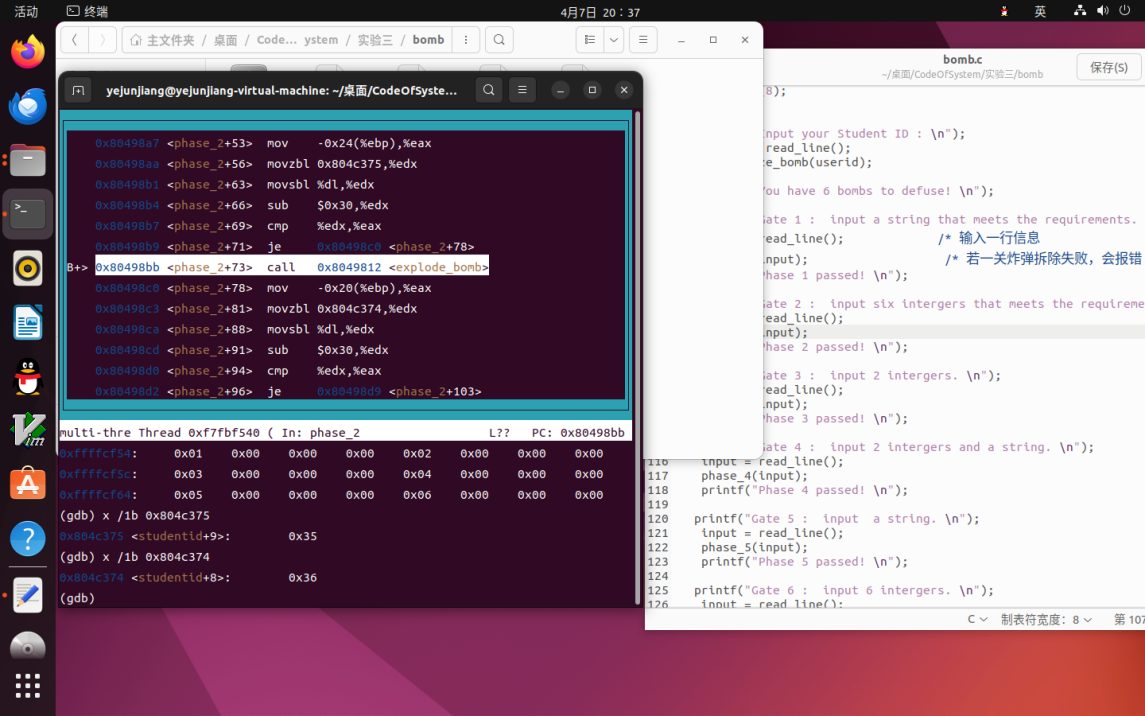
我们接着进行调试，发现当调试到下图所示位置时炸弹就要爆炸了。



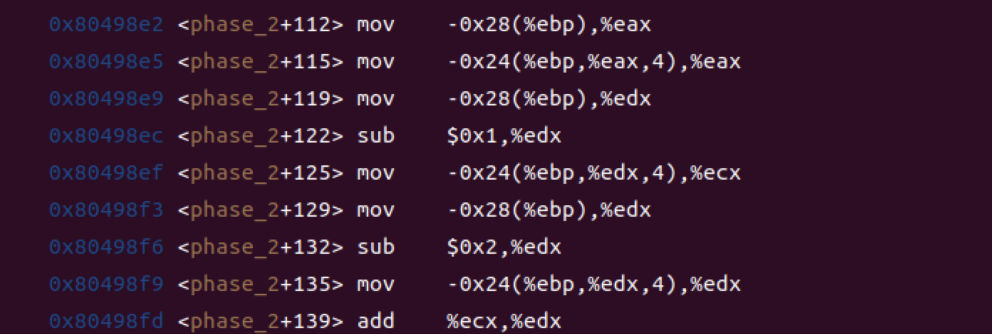
我们这时候输出一下$ebp-0x24，即%eax中的值，发现里面存储的便是我们输入的六个数字。

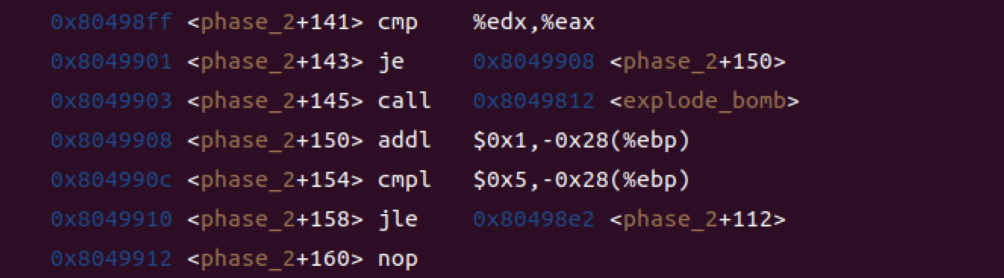


发现当%edx与%eax中存放的值不同时则会调用<explode\_bomb>程序。我们这时候查看%edx中存放的值。下图中打印的值为0x804c375以及后续比较第二个数字用到的0x804c374的值，因为比较前有一个sub $30,%edx指令，故而%edx中存放的为0x804c375存放的值减去30以及0x804c374存放的值减去30，可知%edx前后存放的值分别为0x5，0x6。



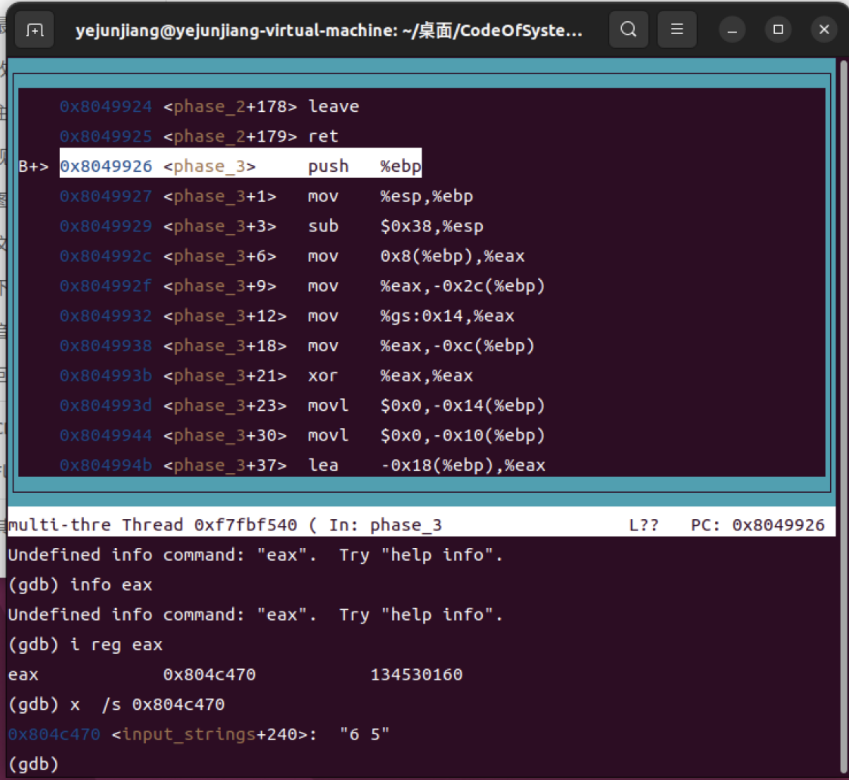
这时我们知道答案前两个数字为5，6，我们重新输入进入到phase\_2，当到后续比较时发现是一个循环代码（下图红框部分），而绿线部分-0x24(%ebp,%edx,4)则是访问我们输入的数字地址，在第一次循环中程序将刚刚提到的数字5，6相加得到11存入%edx与我们输入的数字%eax比较，即后续比较的数字为斐波那契数列的形式。由起始值5，6可知答案为5，6，11，17，28，45。



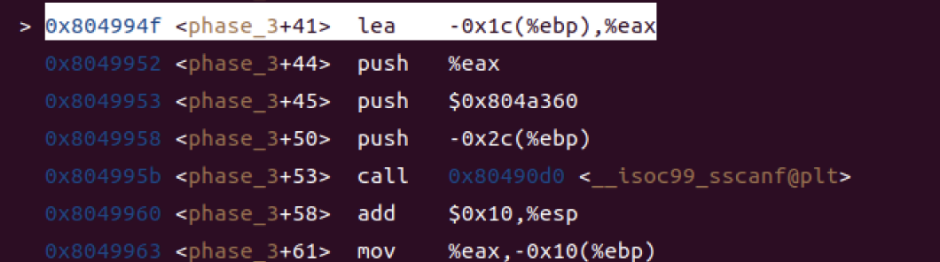


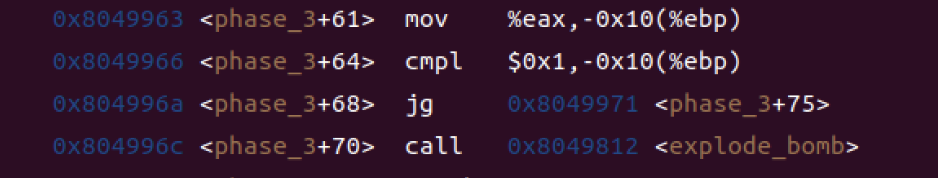
**phase\_3:**

首先试输入两个int型数字“6 5”，然后进入phase\_3程序调试窗口。



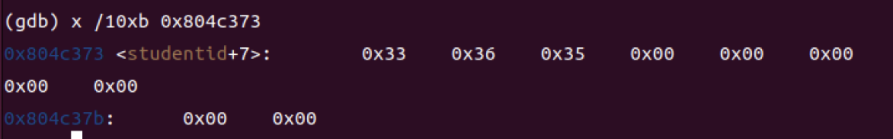
经查看，发现%eax中存放的地址指向我们输入的数字，并且是以字符串的形式存储。我们查看后续代码时发现，程序调用的sscanf程序将我们输入的字符串转换成了数字。再继续调试时发现程序将转换后得到的数字将0x1进行比较，即程序识别我们是否输入了两个数字，如果不能正常识别出两个数字则会调用<explode\_bomb>程序，炸弹爆炸。



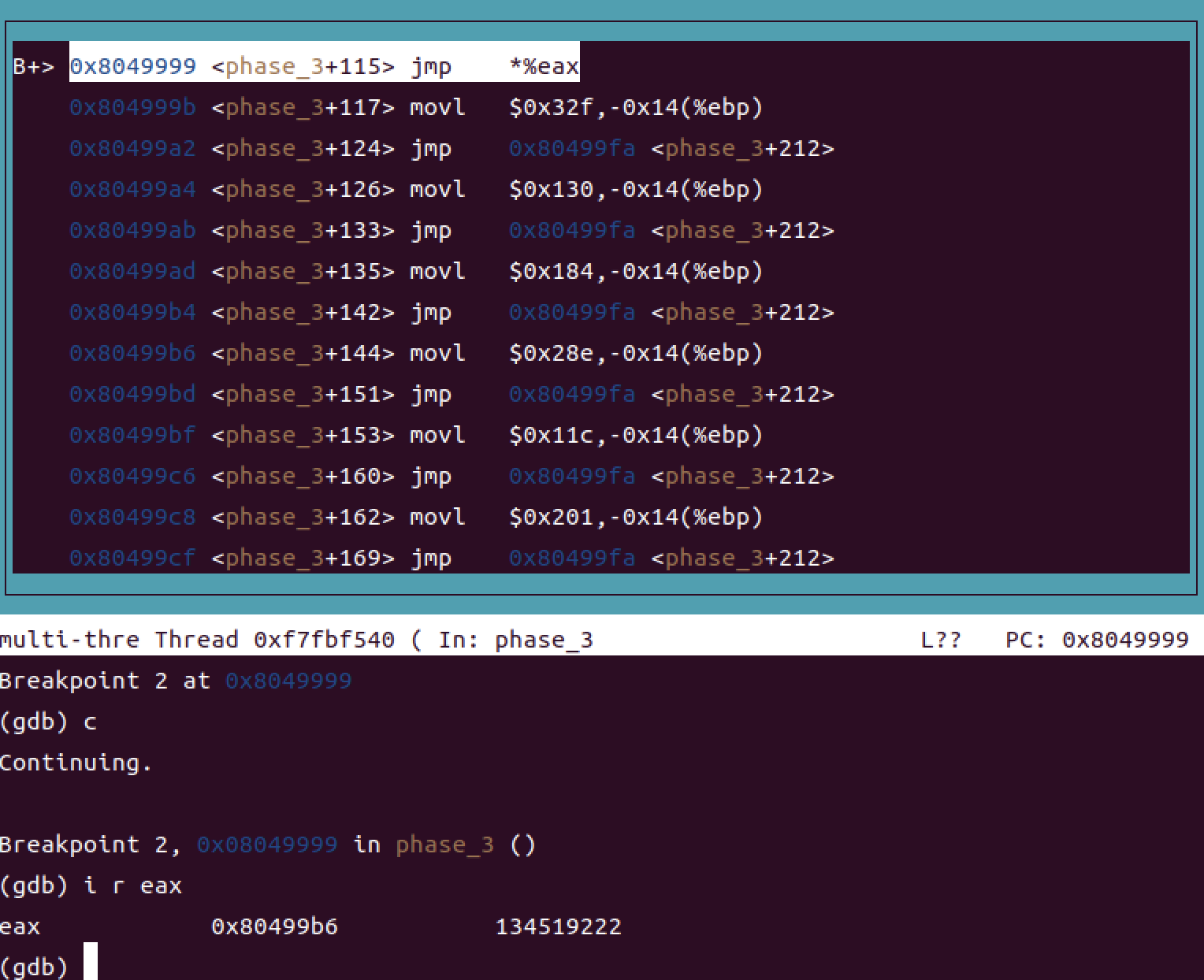


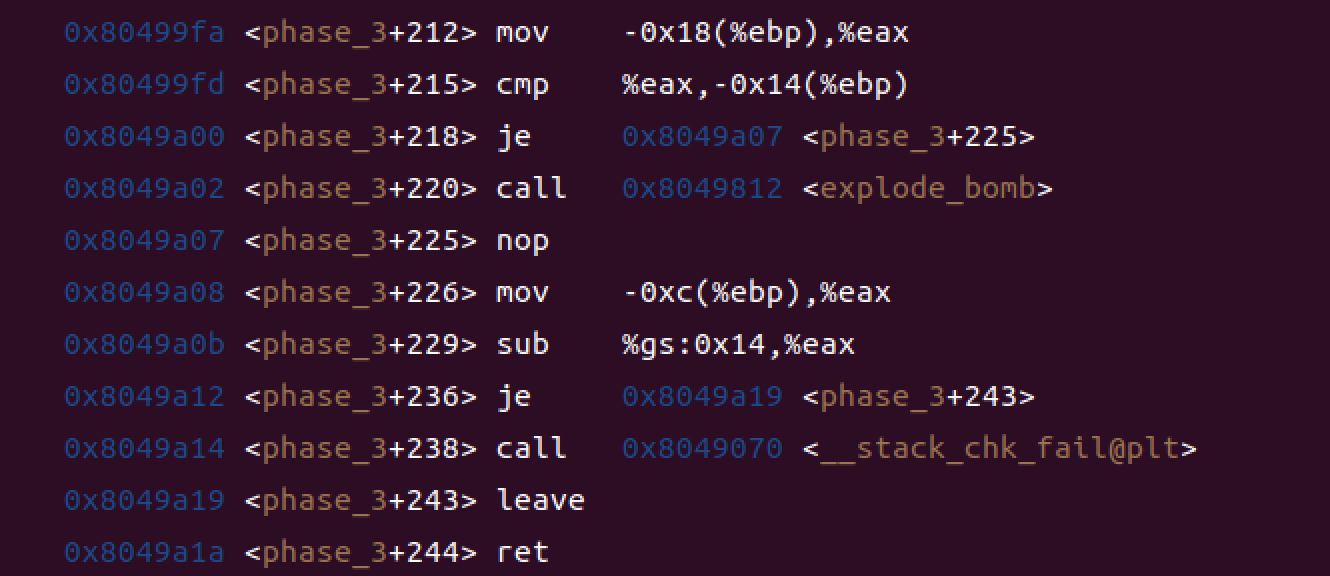
然后我们发现在后续代码中程序传入了一个全局变量0x804c373的地址，并且用其值进行比较（下图红框部分）。我们打印其值，发现里面存储着我们学号后三位加上30的数，即答案第一个数为3（学号第8位）。





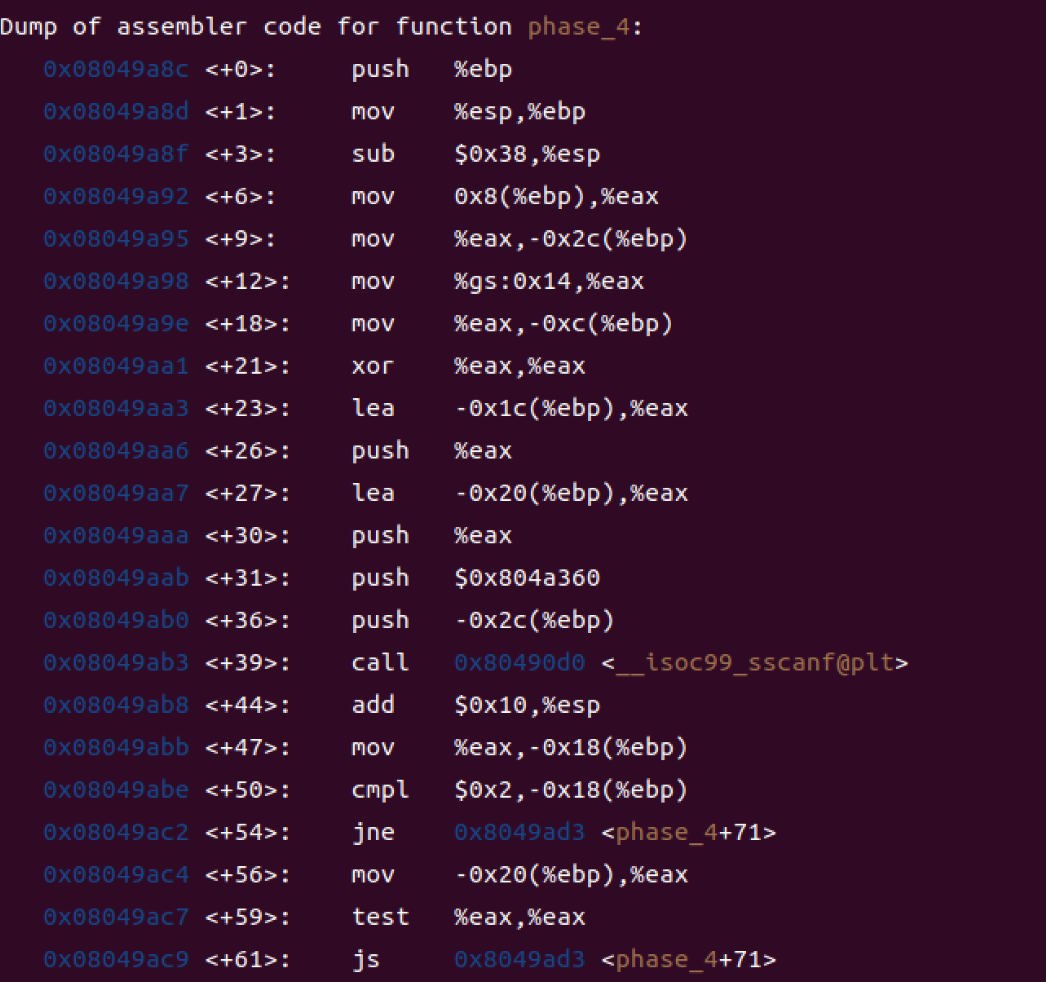
在比较完第一位数字后，程序将跳转到地址为0x804a368的数加上4倍%eax的和作为地址的程序语句。这时候的%eax存放的地址为0x80499b6，也就是此时程序跳转到下图红框部分的语句，后续程序将我们输入的第二个数字与0x28e（654）进行比较，若相等则返回，不相等则调用<explode\_bomb>，故而可知phase\_3的答案为“3 654”。



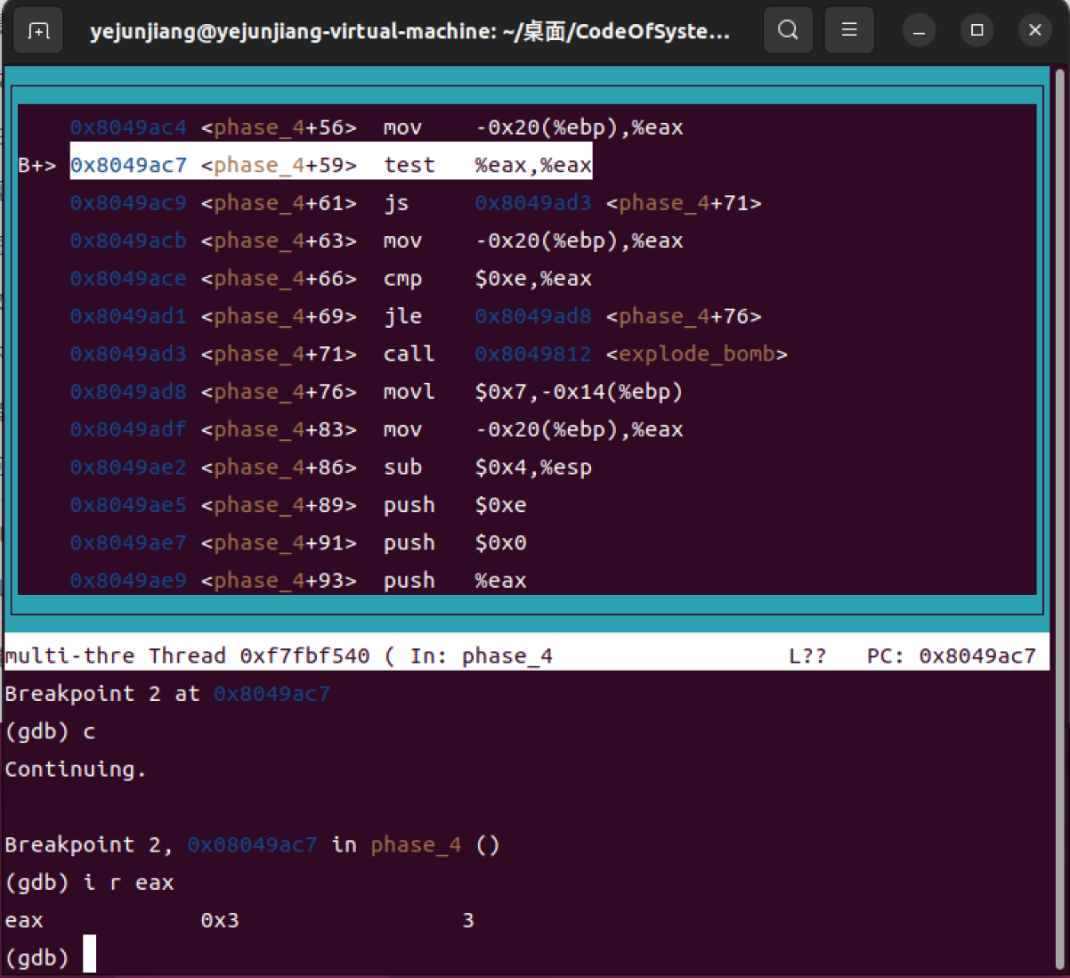


**phase\_4:**

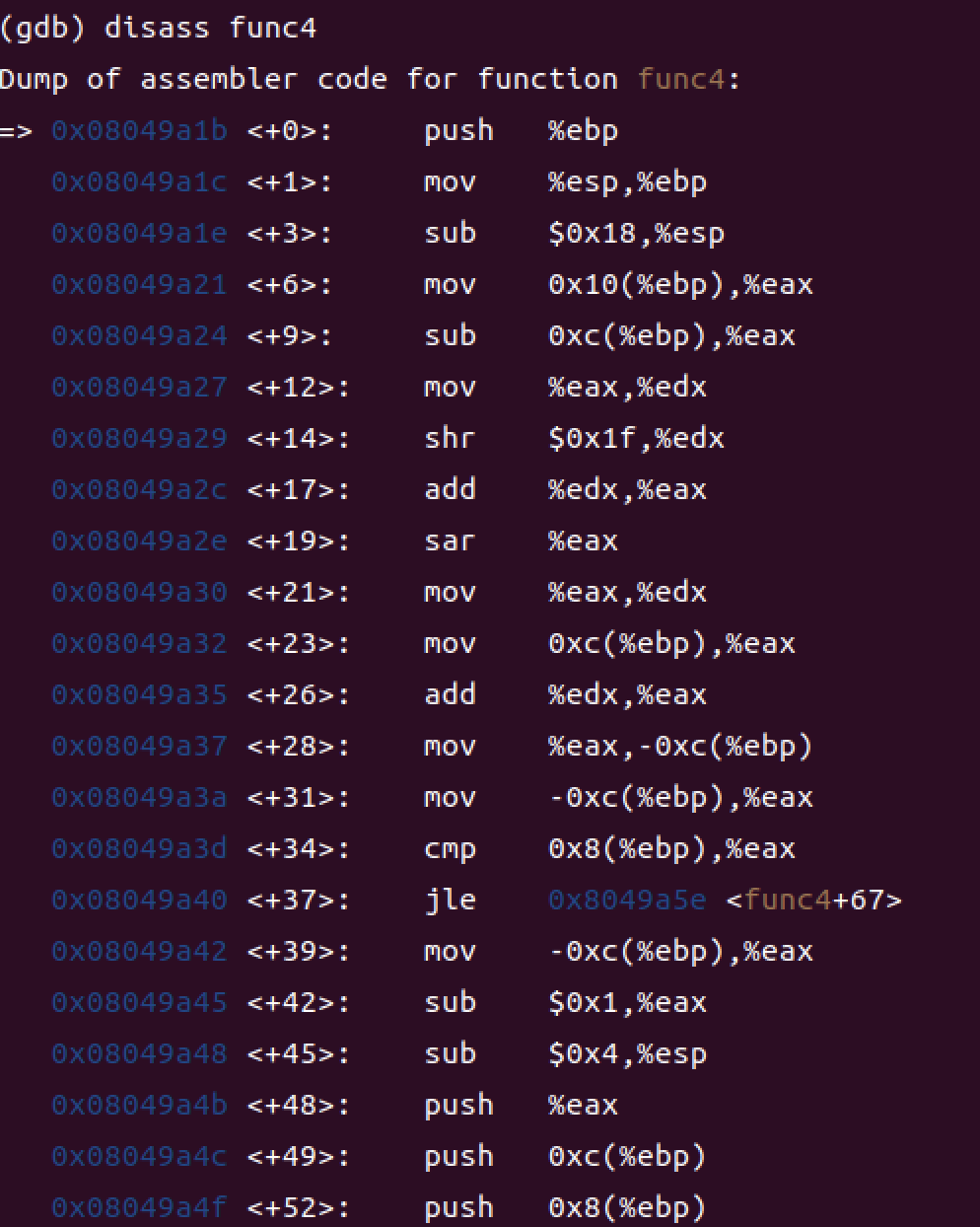
类似地，我们试输入数字以及字符串“3 654 i”进入phase\_4调试界面。

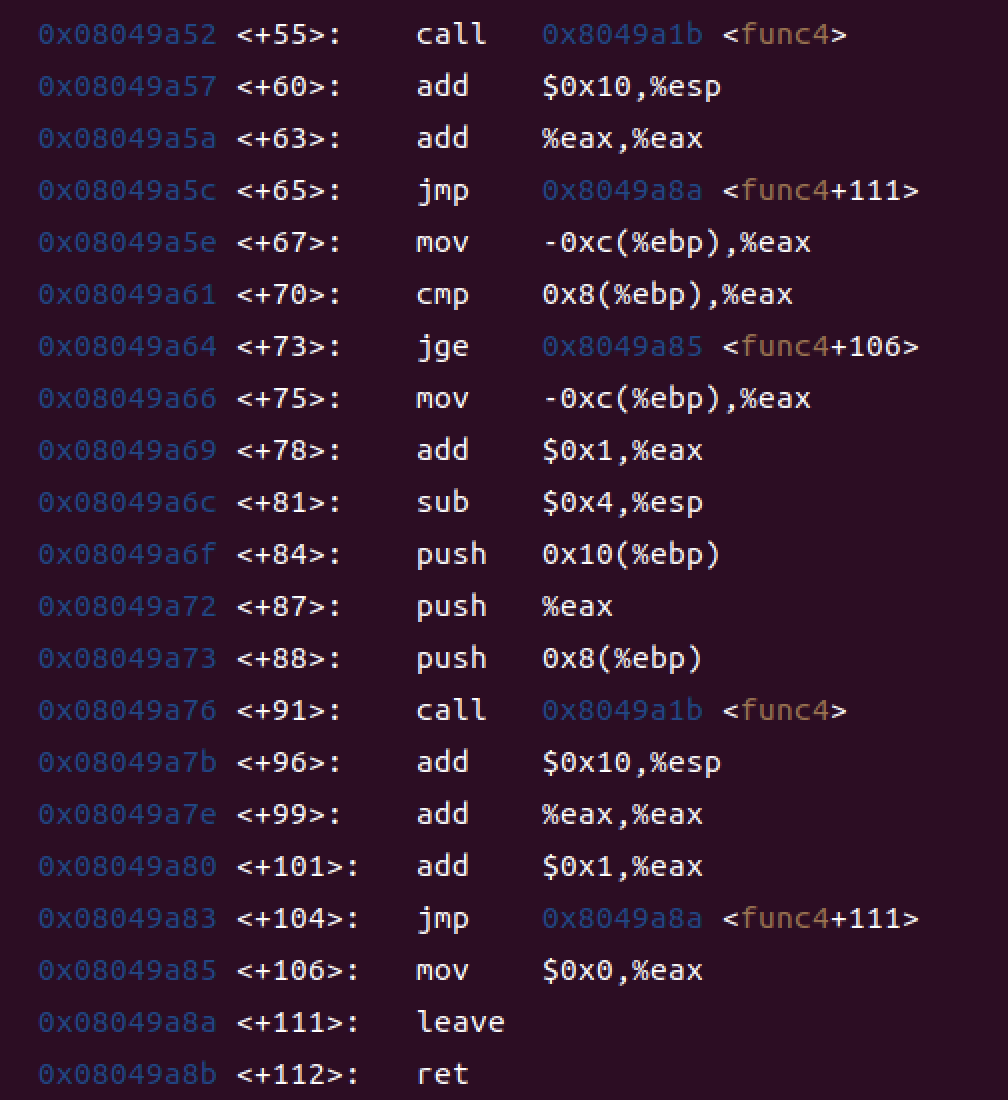


输入发现-20（%ebp）存储的是我们第一个数字3，并且发现这个数字输入时有范围限制[0,14]，传入的三个参数分别为：我们输入的第一个数字3，0，14。

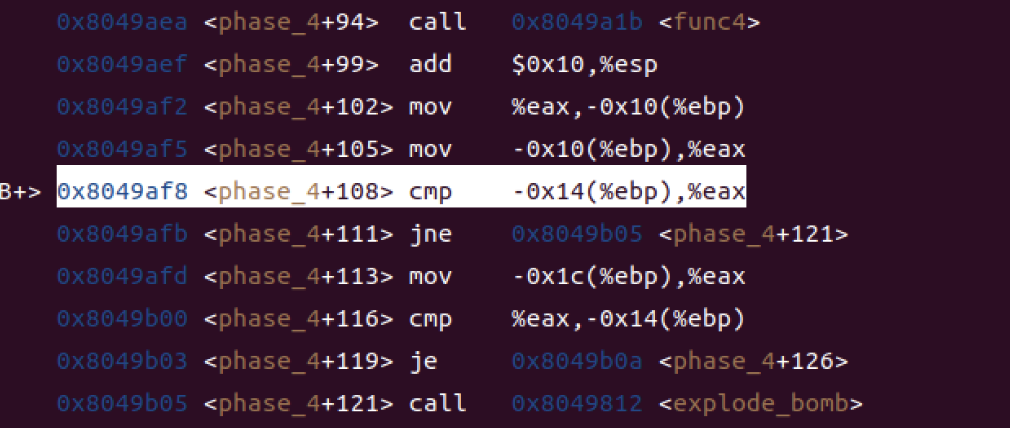


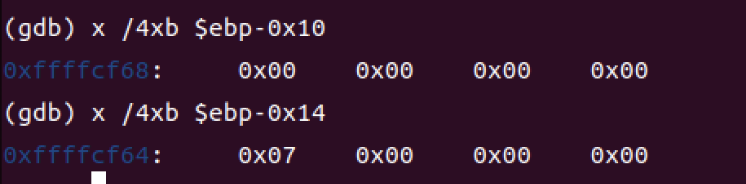
接着程序进入fun4函数，然后0x8(%ebp),0xc(%ebp),0x10(%ebp)分别对应上述三个数字，然后分析fun4函数可以发现fun4函数实际上是二分查找函数，我们将fun4函数定义为fun4（num,l,r），首先fun4函数先计算偏移（r-l）/2（下图紫框部分），随后计算mid = l + (r-l)/2 (下图橙框部分)，接着比较mid和num（下图绿框部分），得到三种情况（下图蓝框部分），第一种返回eax\*2，第二种返回eax\*2+1，第三种返回0。第一种情况为mid大于num的情况，这时会调用func(num,1,mid)递归，返回eax\*2；第二种为mid小于num的情况，这时会调用func（num,mid,14）递归，返回eax\*2+1;第三种情况为mid等于num，这时返回0。





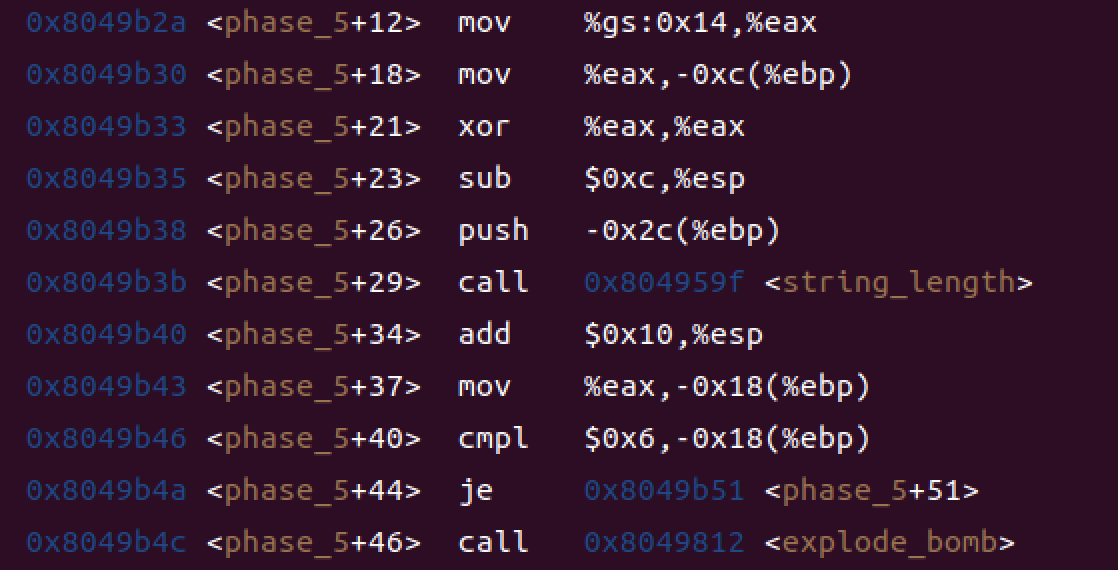
func4函数返回后，回到phase\_4程序，这时程序会将func4函数返回值与-0x14(%ebp)中的值进行比较，经查看，-0x14(%ebp)中的值为7，我们的返回值为0。所以针对func4函数我们要设置第一个数字以至func4函数返回值为7，故而第一个数字应为14。第二次比较则是将我们输入的第二个数与-0x14(%ebp)的值进行比较，故而第二个数字为7。



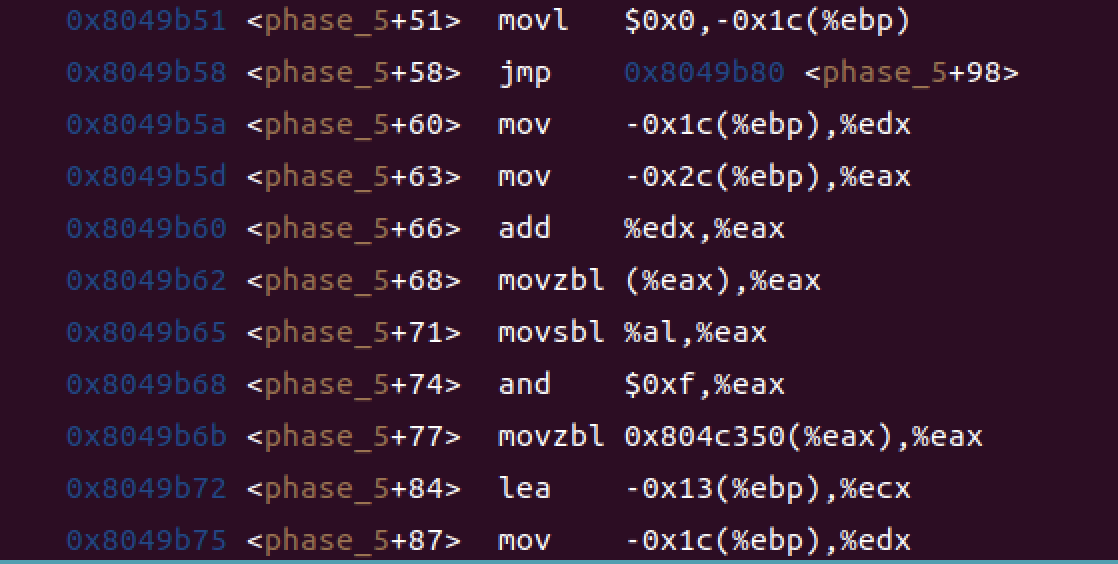


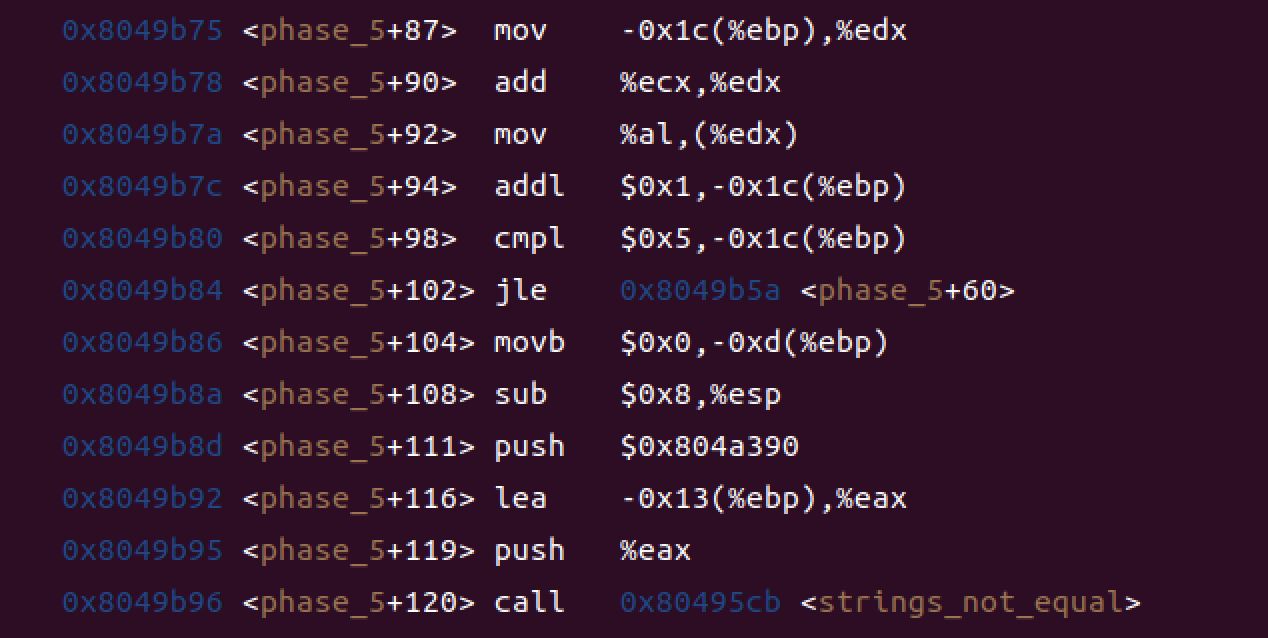
**phase\_5:**

类似地，我们试输入“eeeeee”字符串进入phase\_5程序调试。

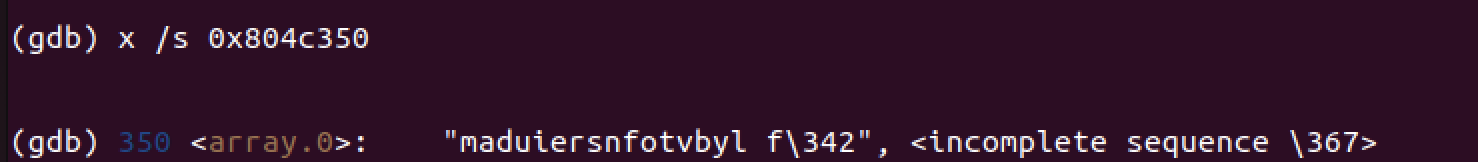


发现进入phase\_5程序后传入一个参数调用<string\_length>函数，由前面分析可得这个函数的功能为判断我们输入的字符串长度，计算我们输入的字符串长度得到函数返回值并与0x6进行比较，若不符合则调用<explode\_bomb>函数，所以分析可得答案为六位字符串。

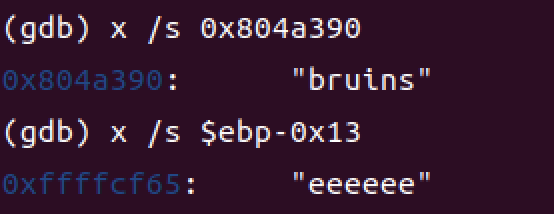




接着分析发现，后续函数使用-0x1c(%ebp)进行计数，用于表示访问的数据到起始值的偏移，而-0x2c(%ebp)则为我们输入的字符串的首地址。并且发现函数后访问了0x804c350地址，我们查看该地址的存储内容发现其值为一组字符串“maduiersnfotvby”。



然后在调用<string\_not\_equal>函数之前程序传入了两个参数，打印出来是两个字符串，据此猜想进入<string\_not\_equal>函数进行两个字符串的比较，若相等则拆除炸弹。

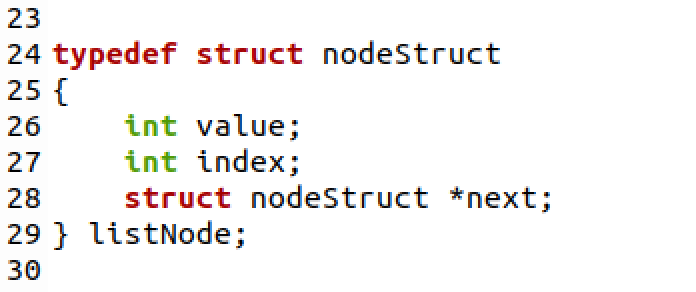


但是程序并不是直接用我们输入的字符串进行比较，而是在比较之前会进行一个计算，再与字符串“bruins”进行比较。而这个计算过程由上图可以分析得到，首先函数每次都会根据计数的-0x1c(%ebp)作为字符串的index来取出我们输入的字符串的指定字符，然后通过无符号扩展成四个字节，接着选取低字节进行有符号扩展成四个字节，然后与0xf进行与运算将结果存至eax中，说明答案与我们输入的字符串低4位ASCII码有关，答案并不唯一。

并且函数将刚刚计算得到的值%eax作为偏移量，在刚刚提到的字符串“maduiersnfotvby”中提取字符，组合成字符串，来与“bruins”进行比较，若相等则通关。这时候就根据字母的ASCII码进行对照就可得出答案，其中“MFCDHG”便为其中一种答案，这时候恰好可以在0x804c350地址的字符串中取出“bruins”。

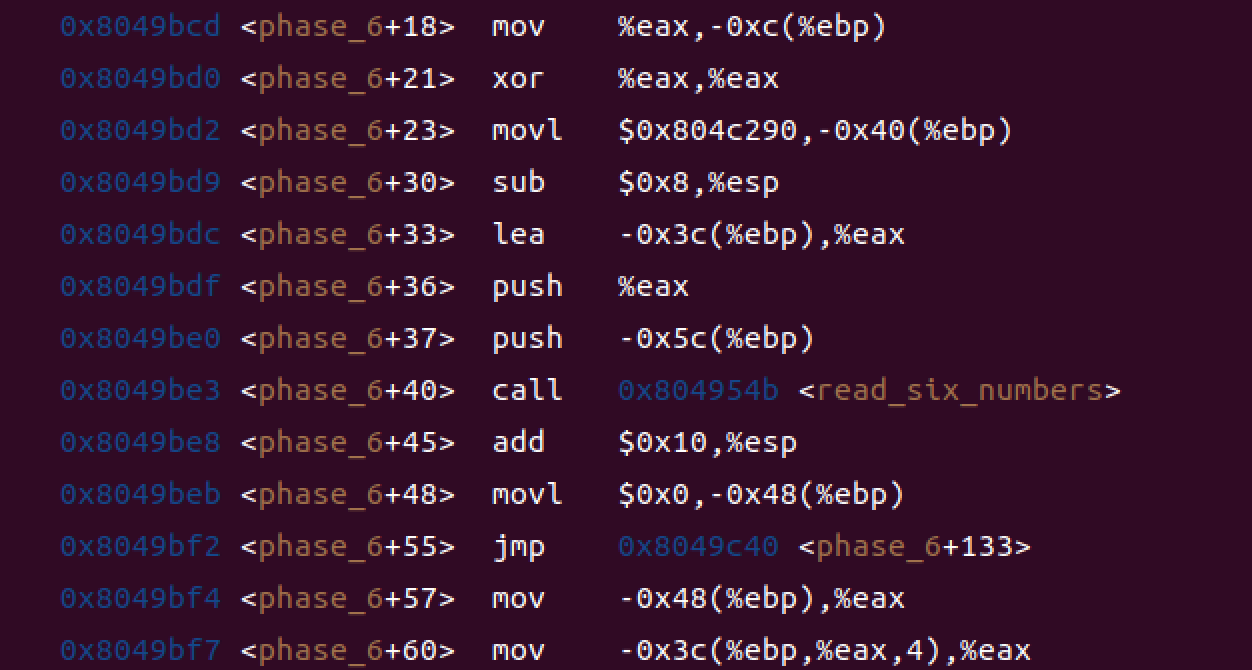
**phase\_6:**

首先我们先从support.h文件中查看节点的定义：

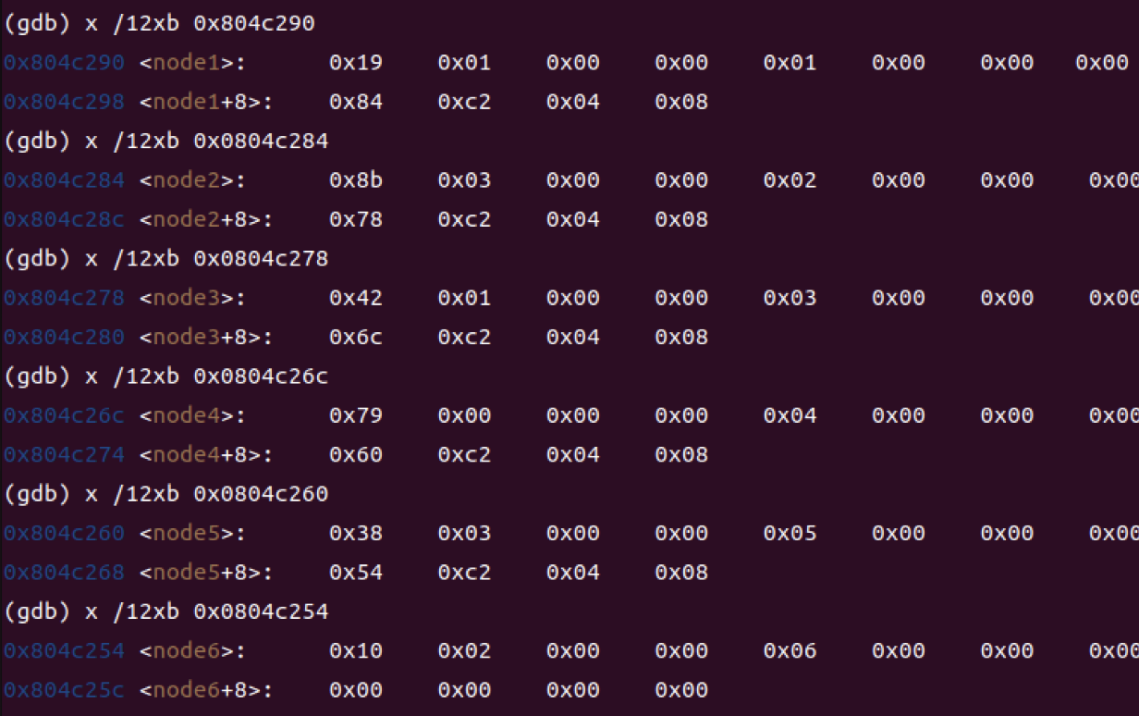


可知listNode为一个单向链表，链表节点nodeStruct有value、index两个属性并存有指向下一节点的指针。

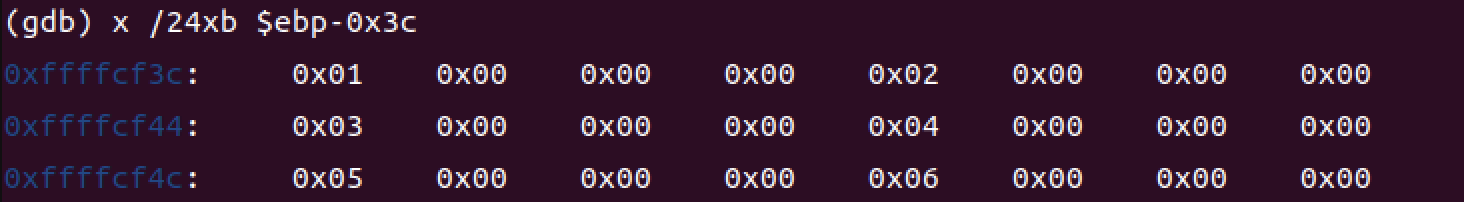
我们再次试输入六个数字“1 2 3 4 5 6”进入phase\_6函数调试。

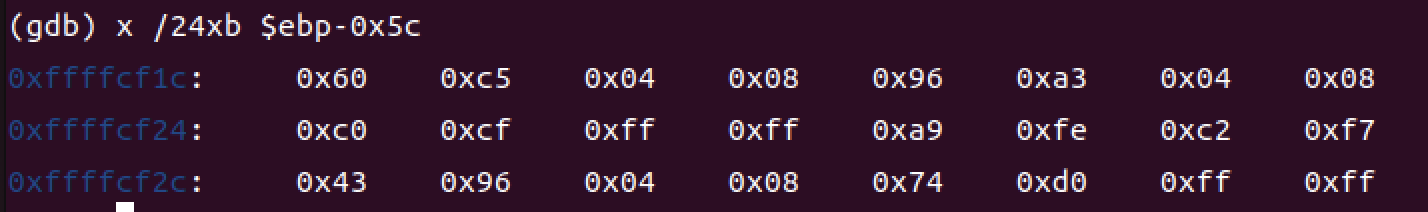


进入调试后，我们注意到程序传入地址0x804c290，我们打印出这个地址的内容。

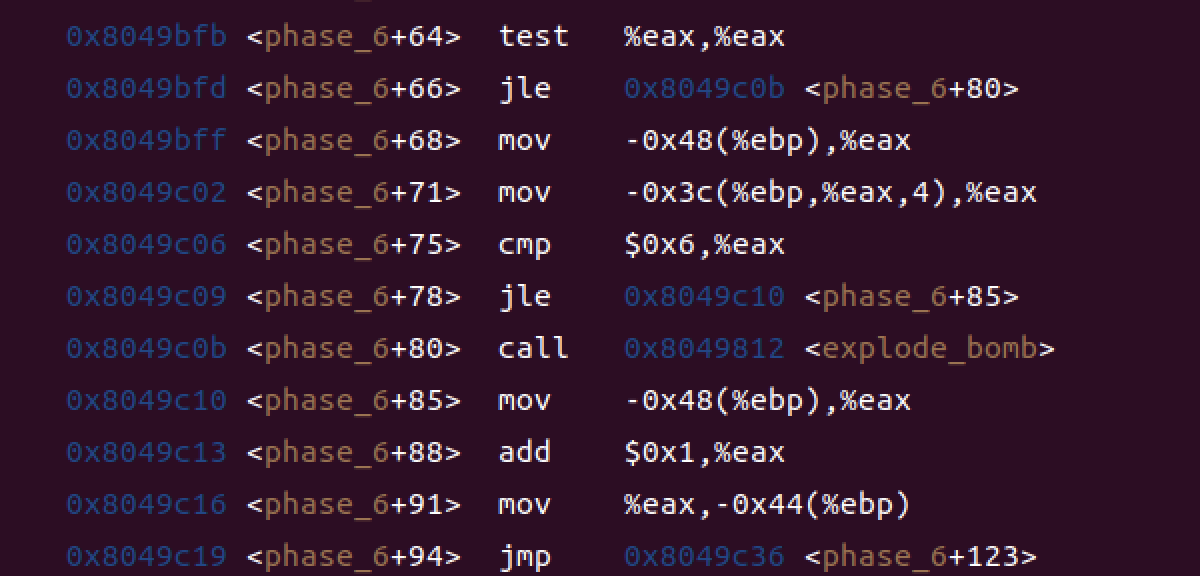


根据打印出的内容，我们猜测第一个数字存储value，第二个数字存储index，第三个数字存储指针地址。并且函数将该链表的第一个节点元素存储到-0x40(%ebp)中，然后程序传入两个参数进入<read\_six\_numbers>函数，这两个参数分别为-0x3c(%ebp)和-0x5c(%ebp)。我们打印这两个参数，发现其中一个参数存储的是我们输入的字符串，另一个存储字符串中的数字。

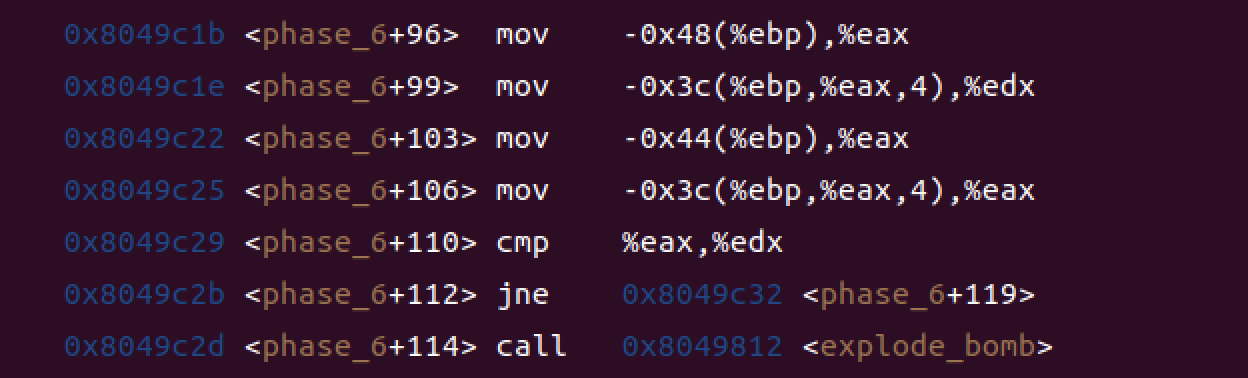




接着程序将0存入-0x48(%ebp)中与0x5进行比较，据此猜想这存储的是数组开头的偏移。

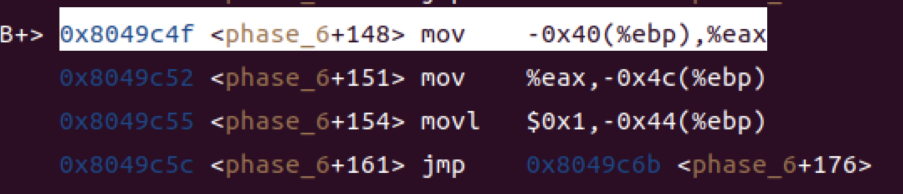


然后程序采用基址+变址寻址来取出第一个值来和6进行比较，这个值如果大于6则调用<explode\_bomb>程序，炸弹爆炸。然后程序将这个偏移量+1存入到-0x44(%ebp)中。

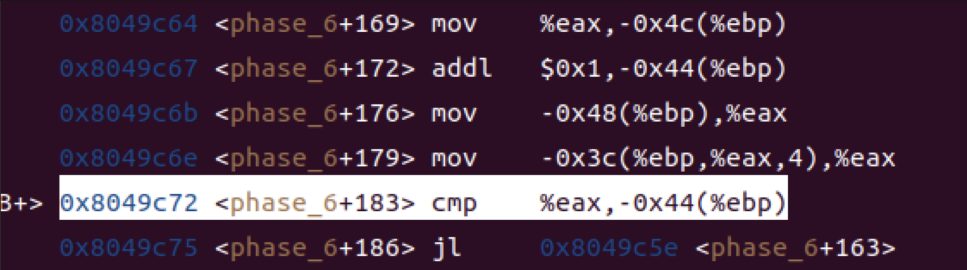


程序再取出第一个数和第二个数进行比较，如果这两个数相等则调用<explode\_bomb>程序。比较之后又将-0x44(%ebp)中的值+1，再重复以上进程。完成第一个数和后面五个数比较，第一个数不能与后面五个数相等，比较一轮后，再将-0x48(%ebp)的值+1，然后再取出这个偏移量的数，判断这个数是否在0～6的范围中，然后接着循环以上比较。简而言之就是六个数字要互不相等。

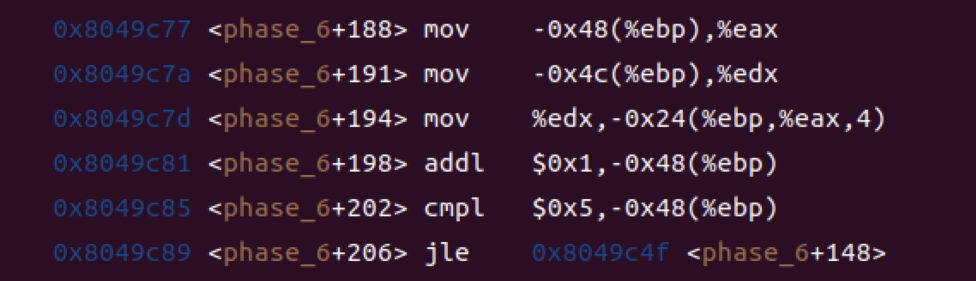
然后把0放入-0x48(%ebp)，把链表第一个节点的地址存入-0x4c(%ebp)，将1存入-0x44(%ebp)。



然后把我们输入的第一个数字1放到%eax和-0x44(%ebp)中的1进行比较，若小于1则跳转，否则进行下一步。

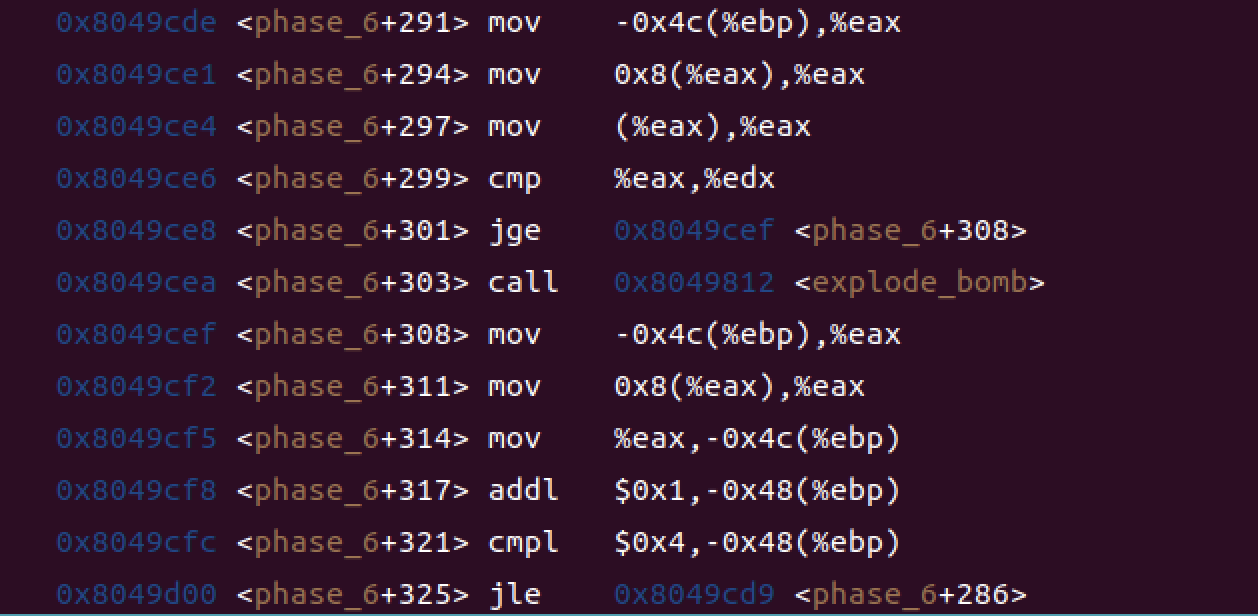


然后程序将第一个节点的地址存入-0x24(%ebp,%eax,4)。



接着程序进行循环操作，按照我们输入的顺序将对应节点的地址按序存储到-0x24(%ebp,%eax,4)中，并且继续循环操作，即按照我们输入的序列来对链表进行排序，每一个节点的next指针指向下一个节点的地址。

最后一部进行比较，将当前节点的地址-0x4c(%ebp)与下一个节点的地址0x8(%eax)来进行比较，若符合则令-0x4c(%ebp)等于next指针，并且循环以上操作。其中-0x48(%ebp)存储我们循环的次数。



总而言之，phase\_6程序将我们输入的序列对链表进行排序并进行比较，因此我们得到正确的序列应该为“2 5 6 3 1 4”，得到答案。

**（2）拆除炸弹的过程中关键操作**

1. 在查看汇编代码时，最好不要从始至尾一步一步地去看，这种方式查看代码时容易给自己造成混乱。在本次实验中，我采取一种寻找关键操作的方法来分析代码，譬如当程序执行到比较，取出全局变量，调用函数，调用<explode\_bomb>函数之前的语句等，在这些语句前后入手可能可以比较快的了解到代码语句的意思意图。

2. 每次分析关卡时，最好先看一下这个关卡函数push了几个变量，也就是这个函数参数的个数，我们也可以事先打印这些参数，对执行函数有一个大概的认知。

3. 分析代码时，最好执行到哪一关键步骤就将此时打印的参数信息给截图记录下来，否则执行后续操作时容易忘记之前的内容。

**四、体会**

在完成本次实验后，我深刻体会到了汇编知识的重要性以及汇编知识对于解决问题的帮助。以下是我在这次实验中的一些心得体会：

1. 深入理解程序逻辑：通过分析二进制炸弹程序的汇编代码，我深入理解了程序的执行逻辑，包括条件判断、循环结构、函数调用等。这让我意识到计算机程序背后的复杂性，以及编写高效、安全程序的重要性。只有深入理解程序的运行原理，才能更好地进行调试和优化。
2. 要加强反汇编技能：通过使用gdb调试工具，我学会了如何逆向分析程序，从汇编代码层面理解程序的运行过程。这种技能不仅对于解决逆向工程问题有帮助，也有助于理解程序的性能瓶颈和安全隐患，提高编程水平。
3. 注重细节和逻辑推理：在分析每个阶段的过程中，我学会了注重细节，比如函数参数的传递、变量的存储和比较等。同时，我也学会运用逻辑推理，根据程序的行为推断出正确的解答。这种细致入微的分析能力在解决实际问题时至关重要。

总结与反思：通过这次实验，我不仅掌握了分析汇编代码的知识，还提高了自己解决和分析问题的能力。同时，我也意识到自己在学习过程中还有许多不足之处，比如在分析问题时可能存在思维定式，缺乏全面的思考。因此，我将继续努力学习，提高自己的技能水平，不断完善自己。