

**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 ：**

**专业班级 ：**

**学 号 ：**

**姓 名 ：**

**指导教师 ：**

**年 月 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**

**任务1** 二进制炸弹拆除

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

phases.h 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);**

要求输入两个整数数据，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。

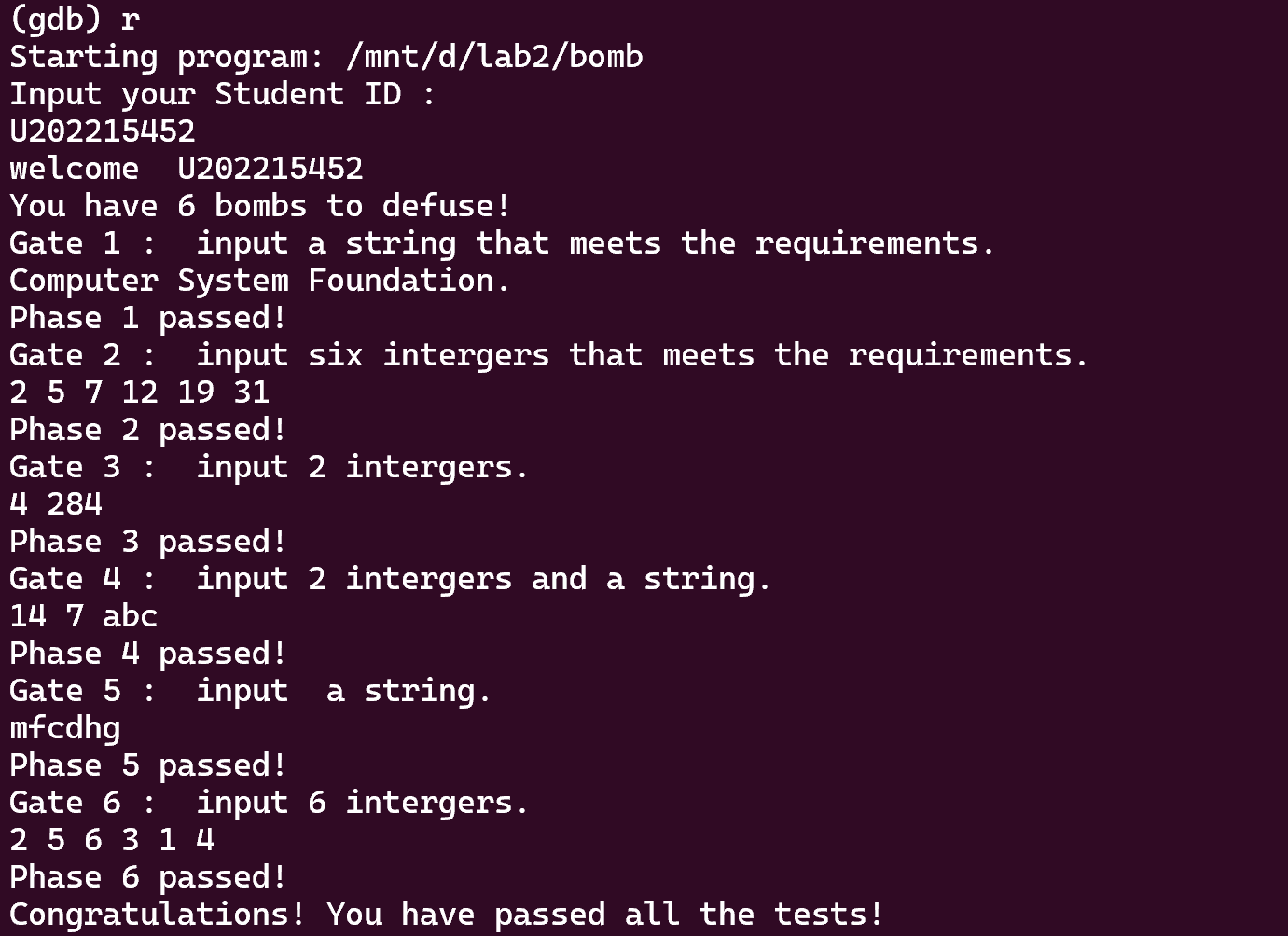
**阶段6：链表、结构、指针的访问 phase\_6(char \*input);**

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**

下图为最终完全拆解全部炸弹的测试截图：



**（2）拆除炸弹的过程中关键操作**

**阶段1：串比较（phase\_1）**

首先用gcc编译产生可执行文件bomb，然后用objdump对bomb文件进行反汇编生成bomb\_assembly.S的汇编语言源程序文件。对该文件的phase\_1函数部分进行观察。

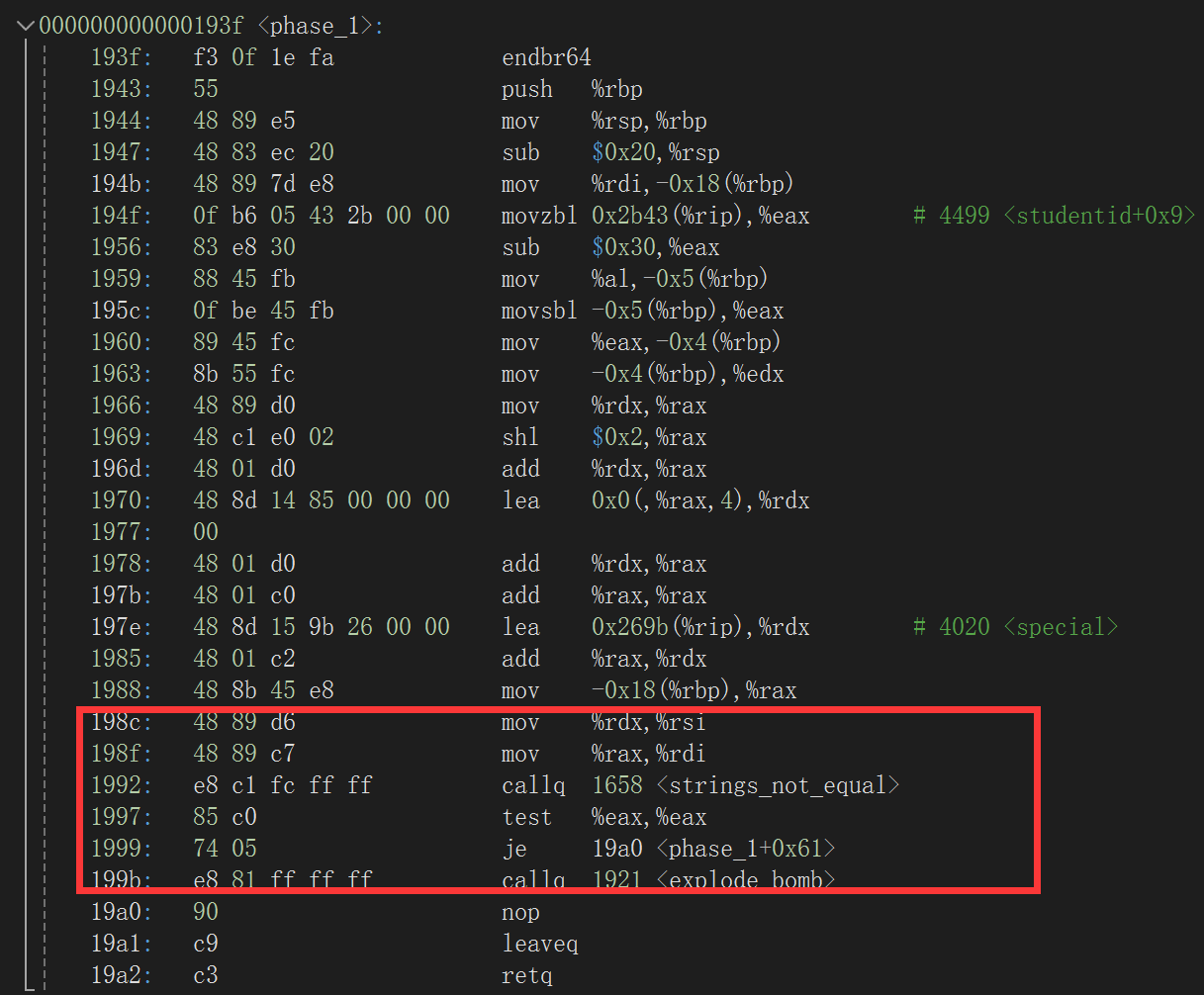


图1-1. phase\_1汇编语言源程序

图中红框所圈出的部分中，调用了<strings\_not\_equal>函数，并根据不同结果引发炸弹或结束函数，由此可以猜测这一步是比较了我们输入的字符串和答案的字符串，再结合前两步中mov %rdx, %rsi和 mov %rax, %rdi可以猜测%rdx或%rax中分别存放的是答案字符串和输入字符串。在198f处设置断点并运行，观察这两个寄存器的内容，就可以得到答案字符串“Computer System Foundation.”。

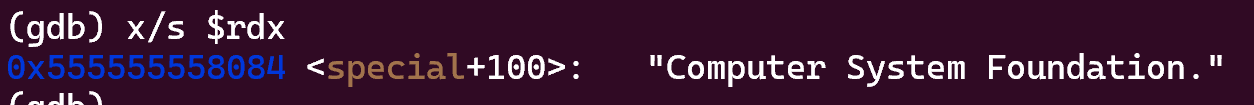


图1-2. phase\_1答案字符串

**阶段2：循环（phase\_2）**

依旧是先对bomb\_assembly.S文件中的phase\_2部分进行观察，发现在调用<read\_six\_numbers>后有三处比较可能会引发爆炸，如图所示。

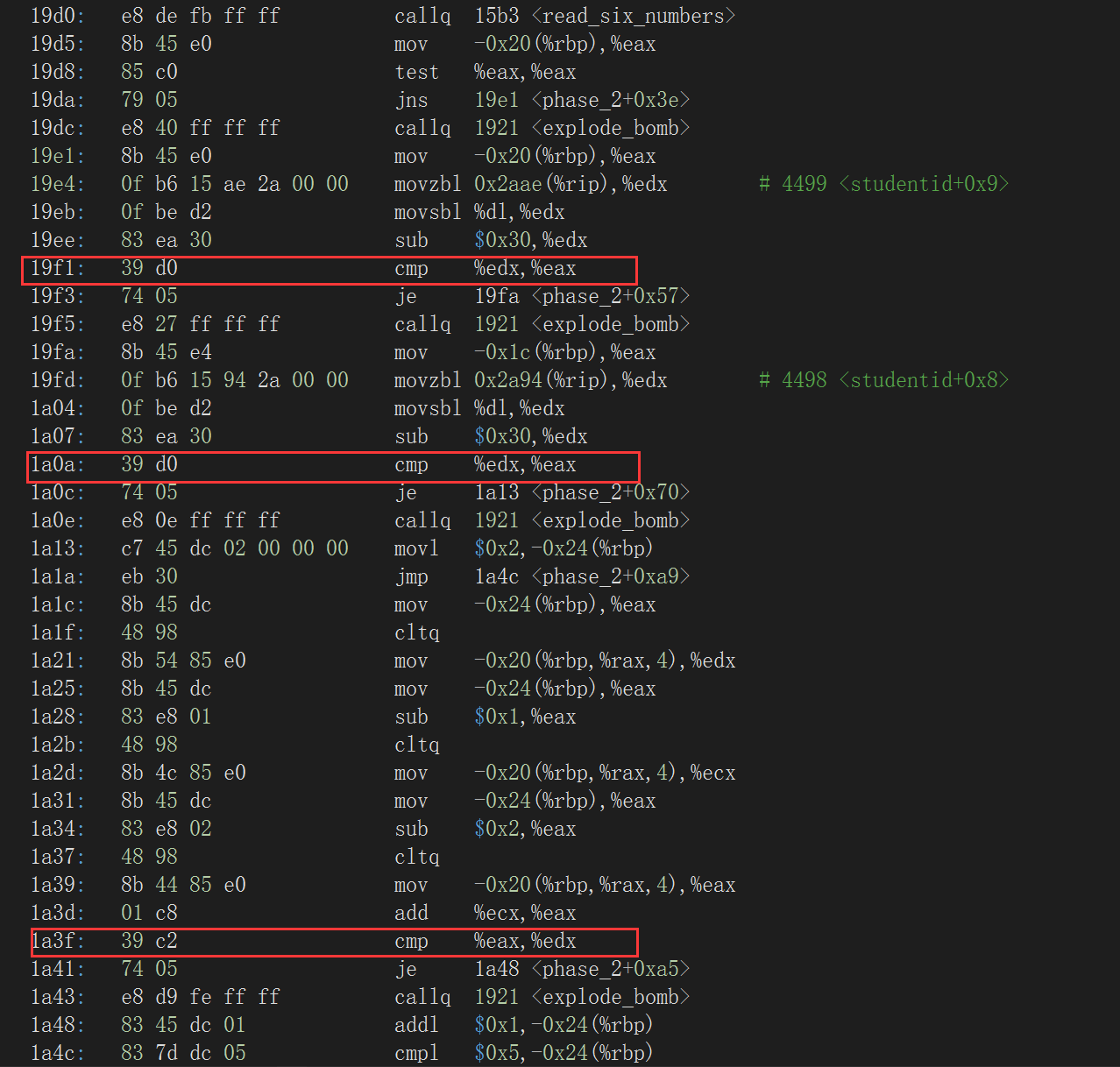


图2-1. phase\_2汇编语言源程序（部分）

经过进一步观察，最后一次比较在一个循环中，这个循环在总比较次数（这个次数位于存储6个数的数组的后一位）大于5后结束。

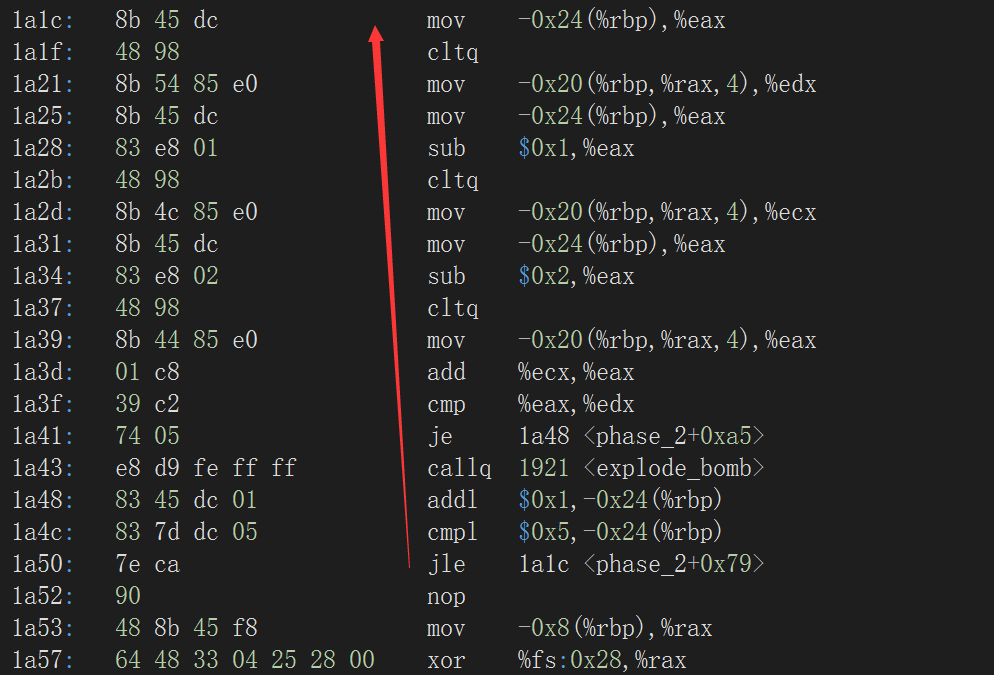


图2-2. phase\_2中的循环部分

在这几个比较的位置设置断点，并查看%edx和%eax的值，找出哪个是答案数值并记录下来即可。

如：输入2 5 7 12 19 1时，获得如下结果：

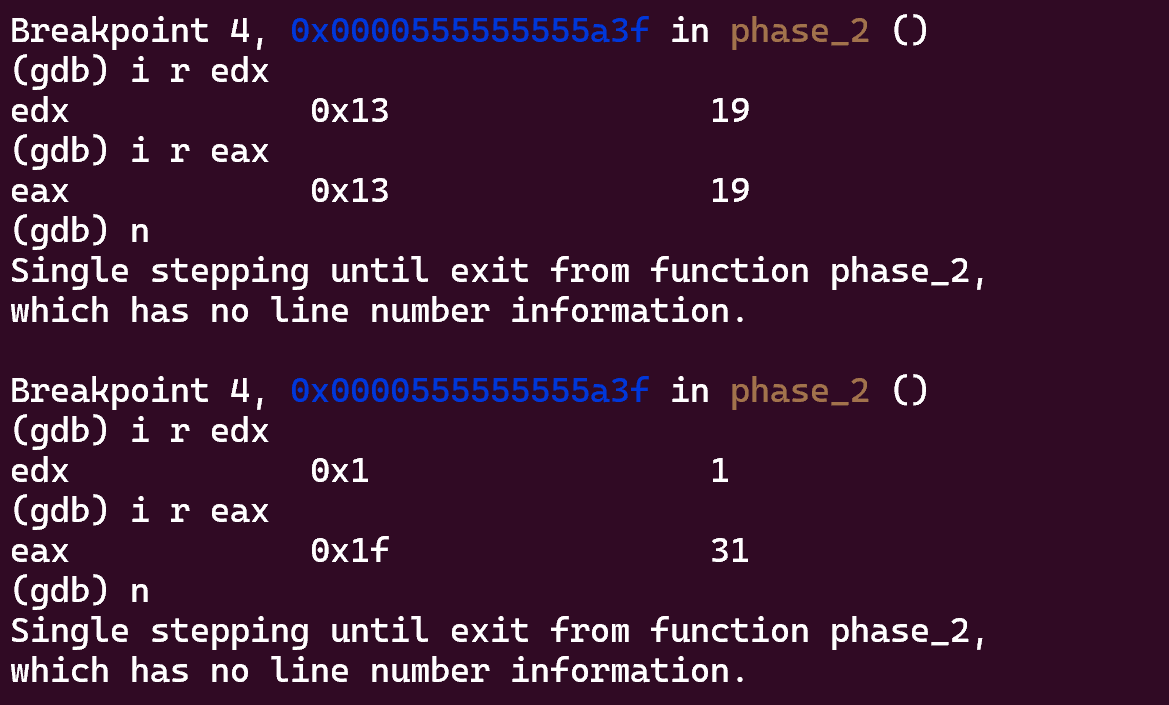


图2-3. phase\_2调试过程截图

即可知最后一位答案数值存放在eax中，输入值存放在edx中，由于前面几个数均正确并通过断点，所以只需要改正最后一个数即可。这六个数分别为：2 5 7 12 19 31。

也可以通过观察反汇编代码直接得出密码。如图，两个方框内分别取出了学号最后一位和倒数第二位与输入的前两个数字进行比较，不相等则直接引爆炸弹，可知密码前两位就是学号的倒数两位。

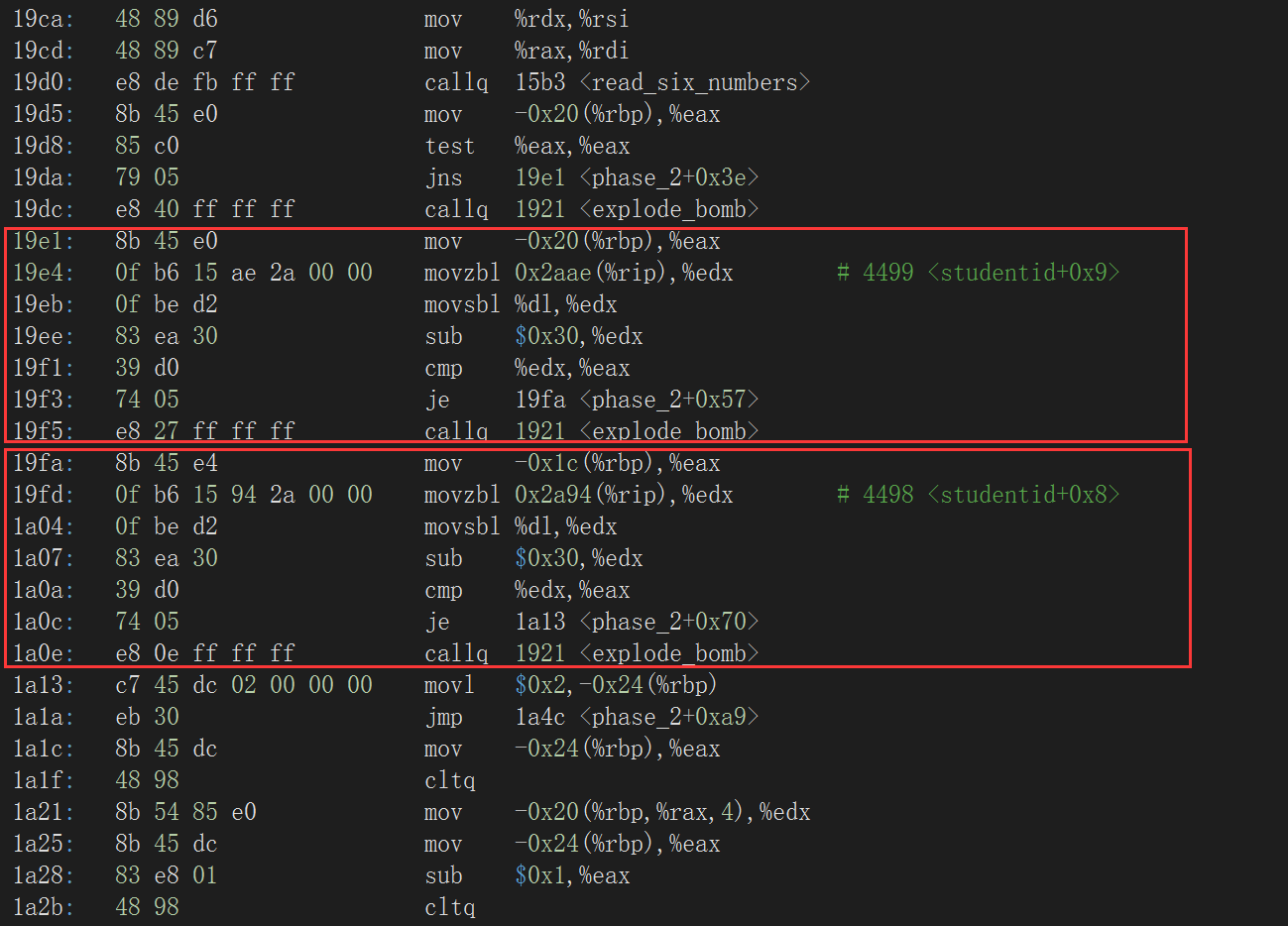


图2-4. phase\_2反汇编代码分析

接着继续观察，由于本关主题为循环，可以找到一个循环体，该循环体的内容翻译成C语言大致为：

int val = 2;

do {

int a = arr[val];

int b = arr[val - 1];

int c = arr[val - 2];

if (b + c != a) {

explode\_bomb();

}

val++;

} while (val <= 5);

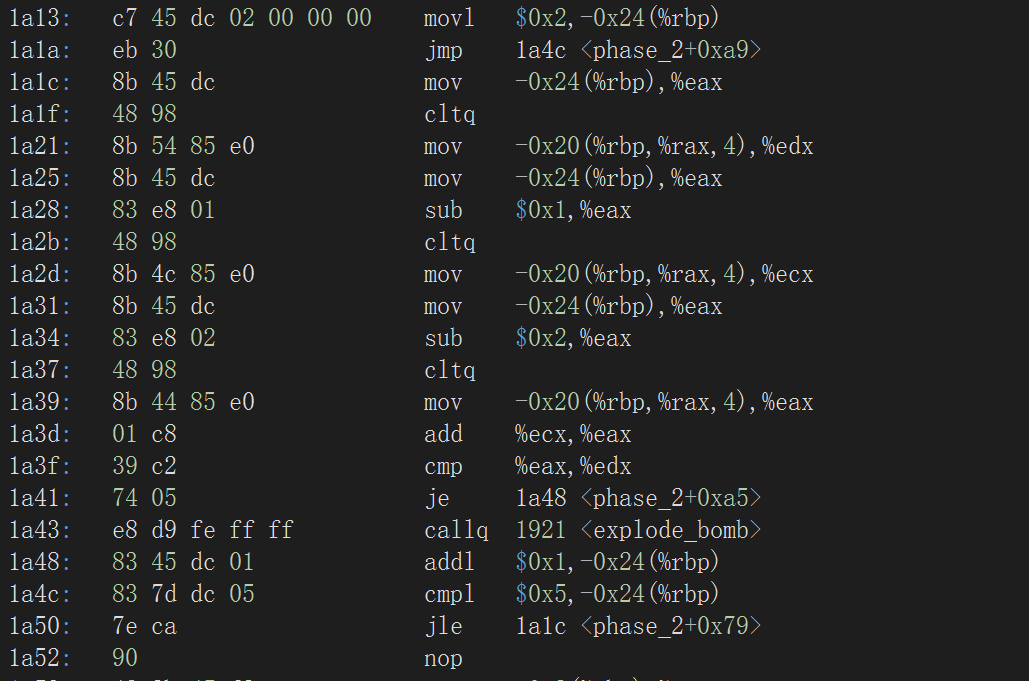


图2-4. phase\_2循环体反汇编代码分析

即后续四个数字均由前两位相加所得，这样就可以得到最终的密码了。

**阶段3：条件分支（phase\_3）**

根据前两个阶段的经验，首先重点寻找汇编语言源程序中的cmp语句，并分析。

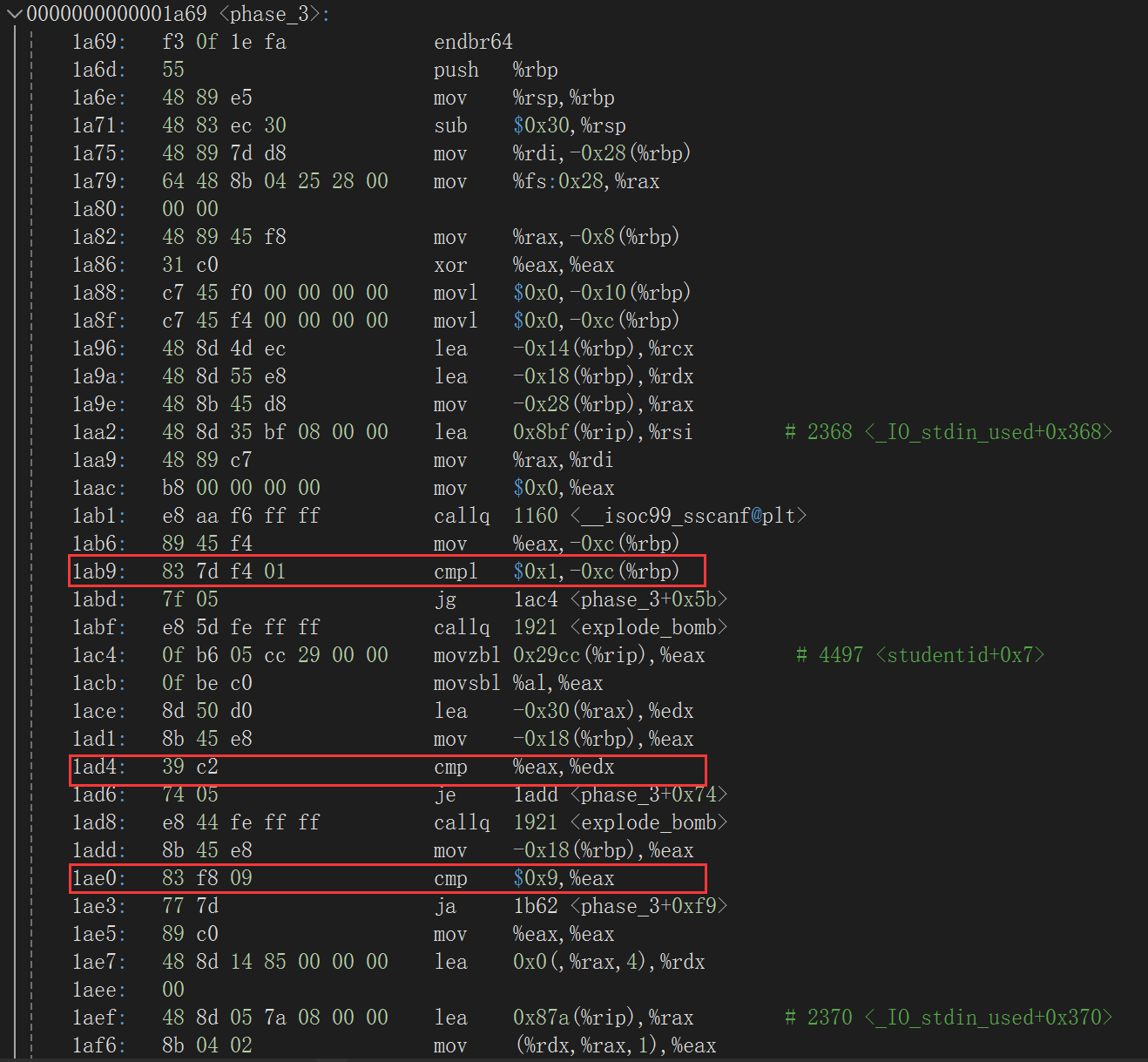


图3-1. phase\_3汇编语言源程序（部分）

经过对前后代码的理解和分析，1ab9处比较应该是保证输入数值的个数应该大于1，否则炸弹直接爆炸；1ad4处比较的是输入的第一个数和第一个数的答案的比较，所以只需要在这里设置断点并查看%eax和%edx的值即可；1ae0处将%eax中内容与9进行比较，可以猜测应该不是炸弹的答案。

继续向后看，可以发现在1b6a处又一次进行了比较，此处我们也可以设置断点。

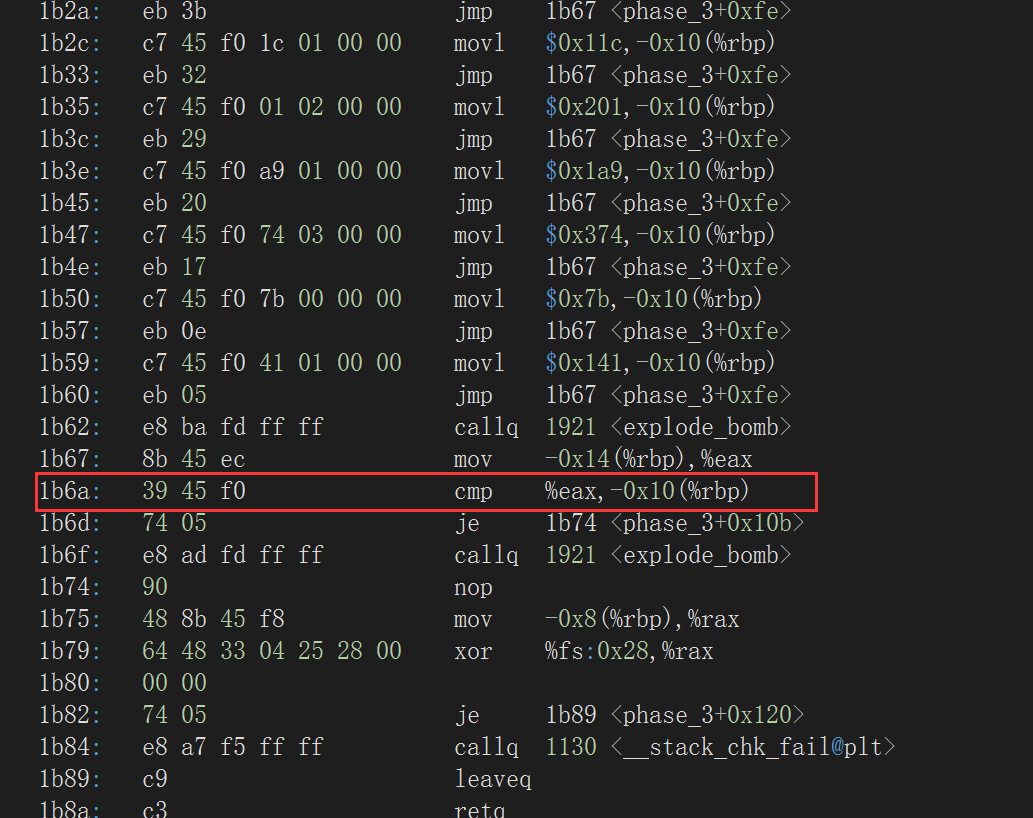


图3-2. phase\_3汇编语言源程序（部分）

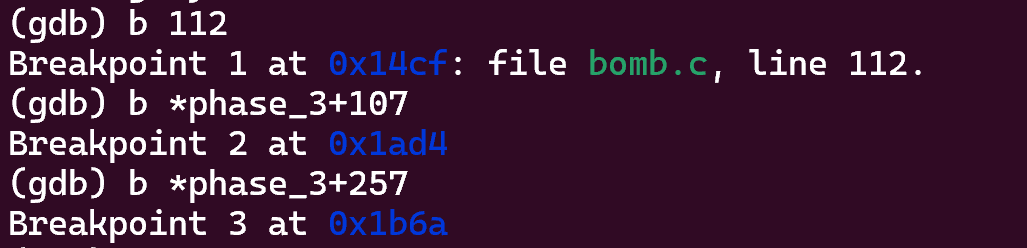


图3-3. phase\_3设置断点

在设置好断点后，就可以开始逐步调试，测试输入“3 8”两个整数，然后逐步调试，查看比较的两个数，判断哪个是答案数值即可。如下图所示，在获得第一个数为4后，由于第一次输入的答案错误，会引爆炸弹，第二次调试时可以尝试输入“4 8”，方法类似，从而获得第二个答案数值为284。

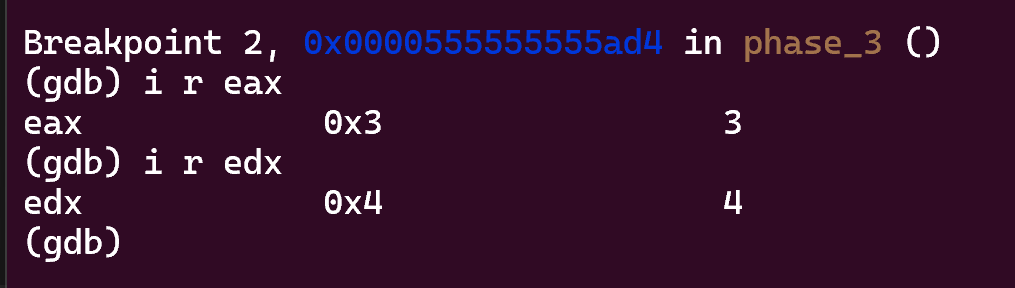


图3-4. phase\_3调试过程截图





图3-5. phase\_3调试过程截图2

综上，需要输入的两个整数为4 284。

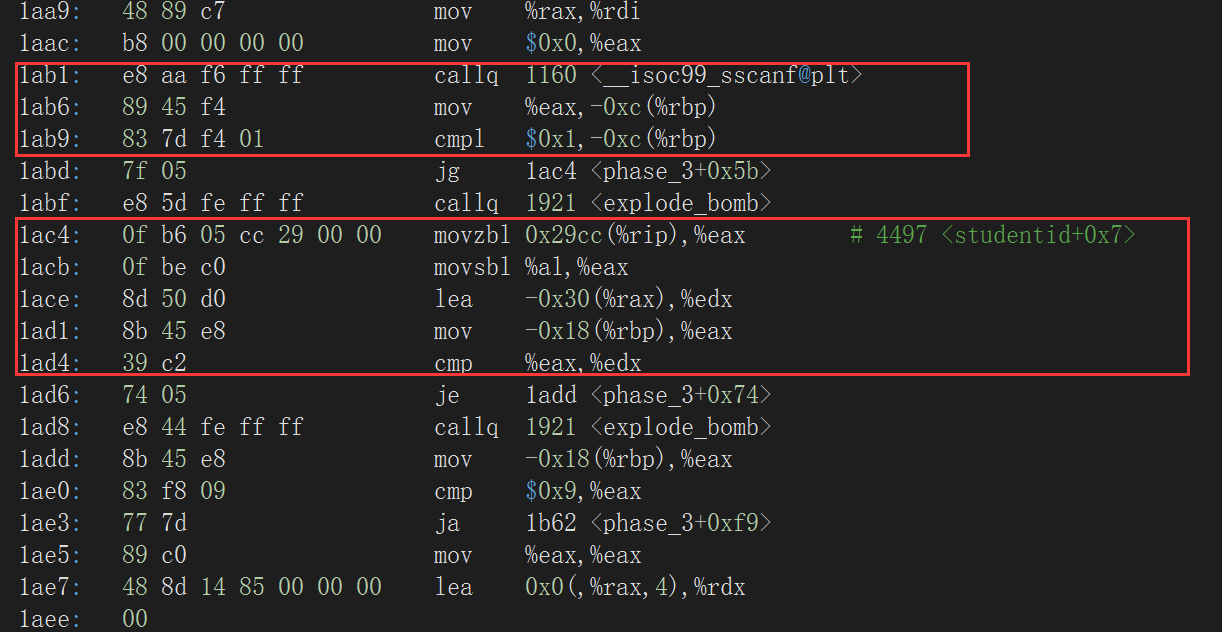


图3-6. phase\_3反汇编代码分析

本关也可以通过观察反汇编代码直接得到最终的密码。如图，第一个红色方框得到将输入的数字与1比较，小于等于1则爆炸，所以至少输入两个数字。由第二个红色方框可知取了学号的第7位数字存入%edx中，输入的第一位数字存于%eax中，将其比较，若相等则跳转至该数字所在的分支，由下图可得该分支对应的第二位密码是0x11c，转化为十进制为284。第三关的密码为4 284。

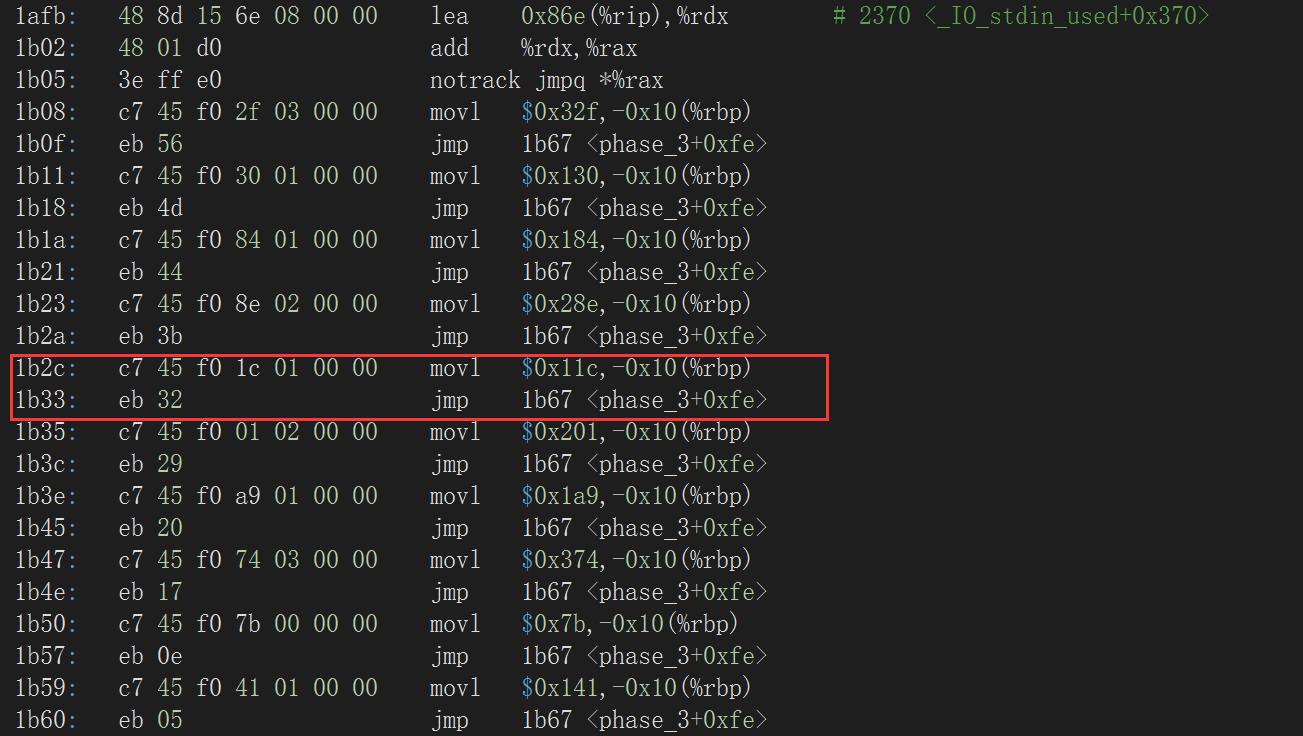


图3-6. phase\_3反汇编代码分支分析

**阶段4：递归调用和栈（phase\_4）**

首先依旧使用layout asm指令查看phase\_4部分的反汇编代码，如下图所示。首先分析上方的方框部分内容，首先判断输入的值的数量是否大于等于2，如果小于2直接引爆炸弹，然后将输入的某一个数存入了-0x1c(%rbp)并进行了一系列比较，要求该数在0~14之内，否则会引爆炸弹，又根据实验内容中对这一部分的叙述，猜测是要在二叉树里寻找的数值，即第一个数。

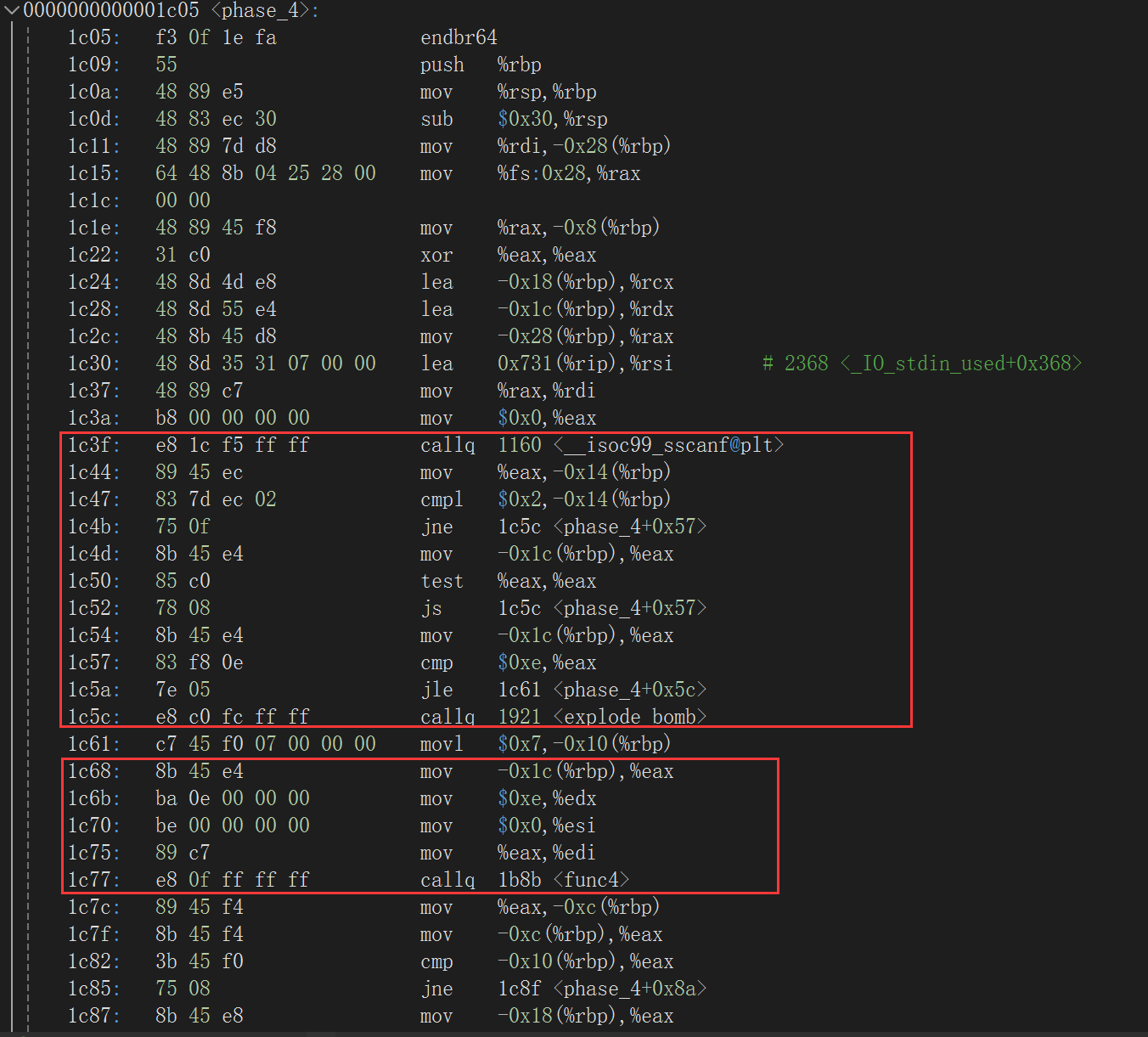


图4-1. phase\_4汇编语言源程序（部分）

接着分析下面一个方框，可以看见将14，0以及输入的第一个数分别移入了%edx，%esi和%edi，作为调用func4的参数，猜测func4是一个在二叉树里寻找指定范围内的某一个数，并返回由路径的二进制表达构成的数，并与-0x10(%rbp)中存储的数进行比较，如果不相同则引爆炸弹。

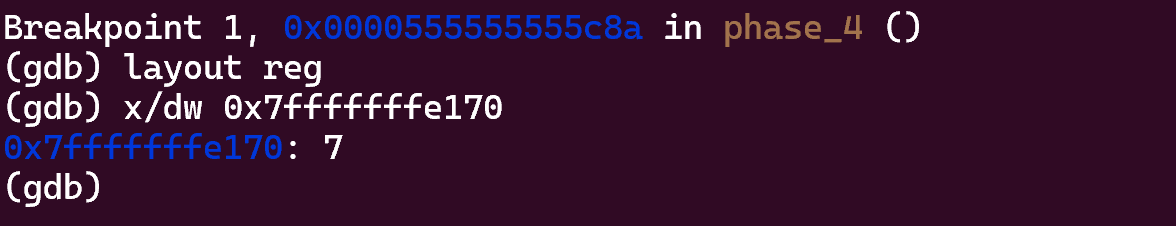


图4-2. phase\_4调试过程截图

设置断点并调试，查看此处-0x10(%rbp)中存储的为7，即搜索路径为111，倒推出第一个数为14。又由前文所述的：输入的数的数量要大于等于2可知，这一关要我们输入的一个字符串可能是无效信息。

综上，本关的解为：14 7 任意字符串。

**阶段5：指针和数组访问（phase\_5）**

依然先观察反汇编代码：可以得到主要由两个部分组成。首先观察第一个部分，如下图所示，这里主要是先开栈存储输入的字符串，然后调用<string\_length>得到字符串的长度，当长度不等于6时引爆炸弹，由此我们可知，这一关应该是让我们输入一个由6个字符构成的字符串。

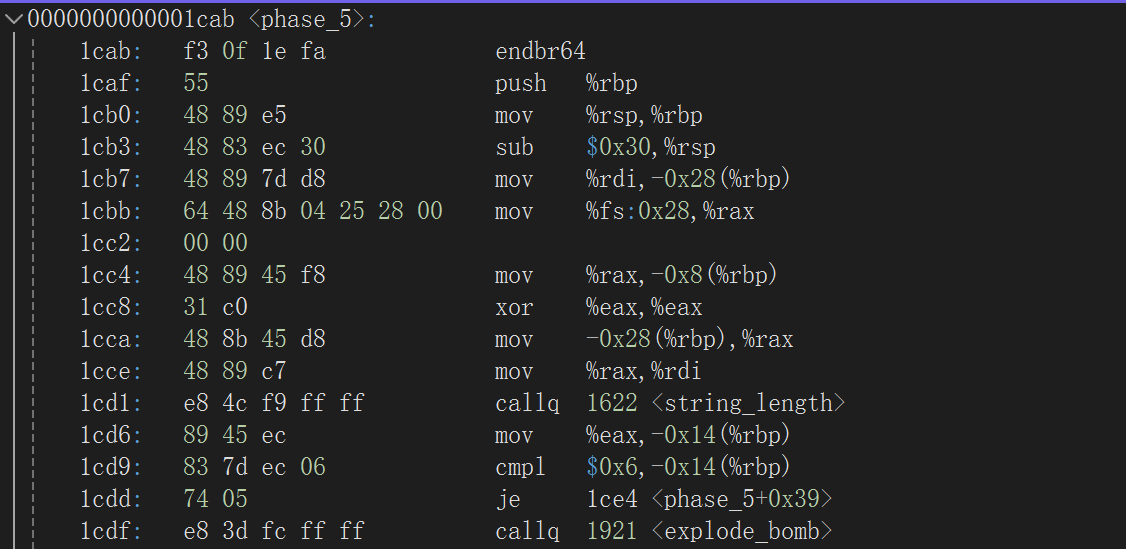


图5-1. phase\_5汇编语言源程序（第1部分）

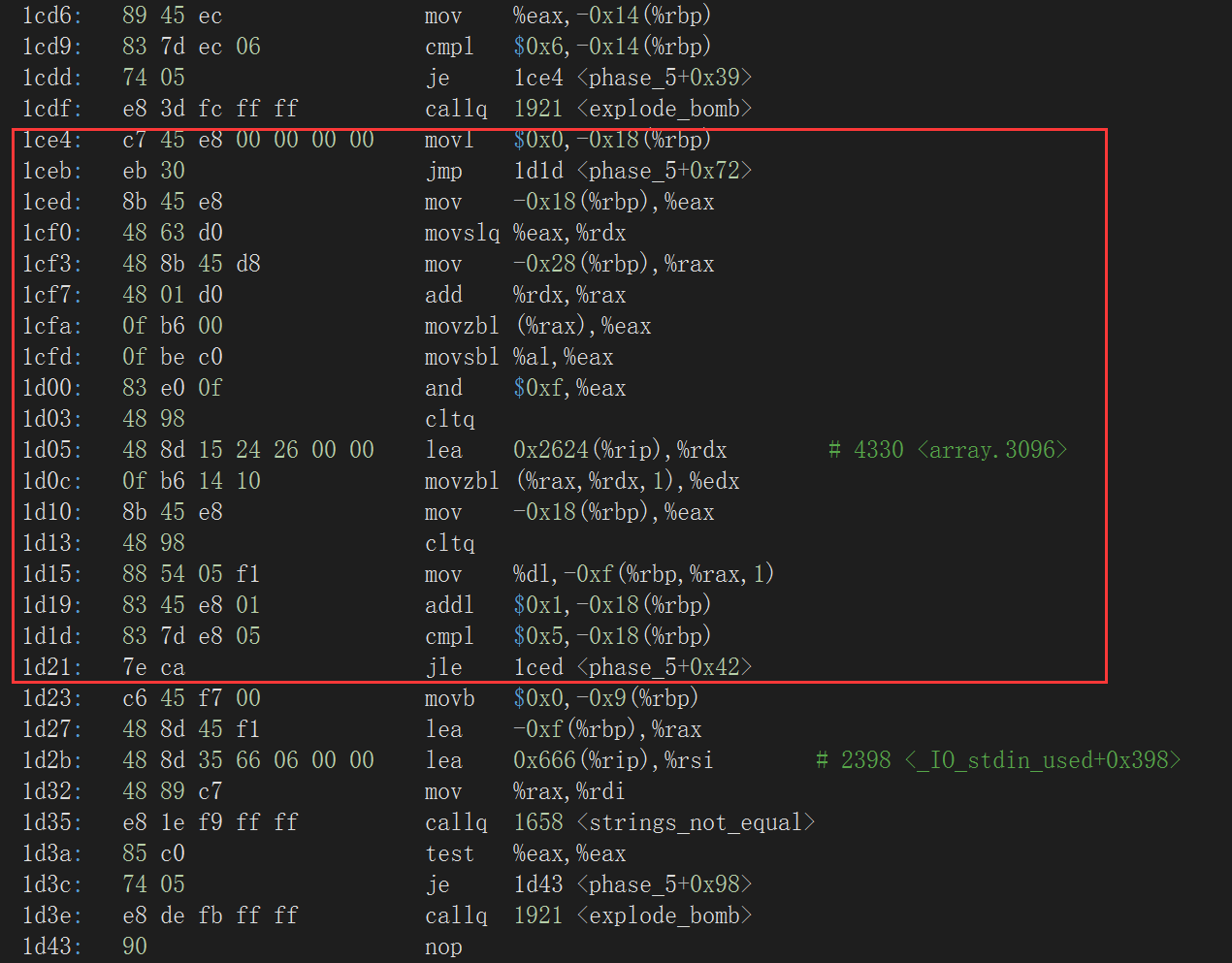


图5-2. phase\_5汇编语言源程序（第2部分）

接着观察第2部分，其中由红框圈出的部分是一个循环过程，里面的内容是对输入字符串依次进行处理，然后生成一个新的字符串，并将这个新的字符串与已有字符串进行比较。

第一个cltq之前的操作是根据循环的次数i获取字符串的第i个字符的后四位，第二个cltq之前，将<array.3096>的数组首地址存入了%rdx，调用该处地址（第一个红圈所示）可以查看这个数组内容，为“maduiersnfotvbyl8\003”，然后以前面取出的后四位，设为k，作为索引，在数组中找到第k个字符存入新的字符串中，最后将新生成的字符串与另一个字符串（第二个红圈所示地址）进行比较，调用后得到该字符串为“bruins”。

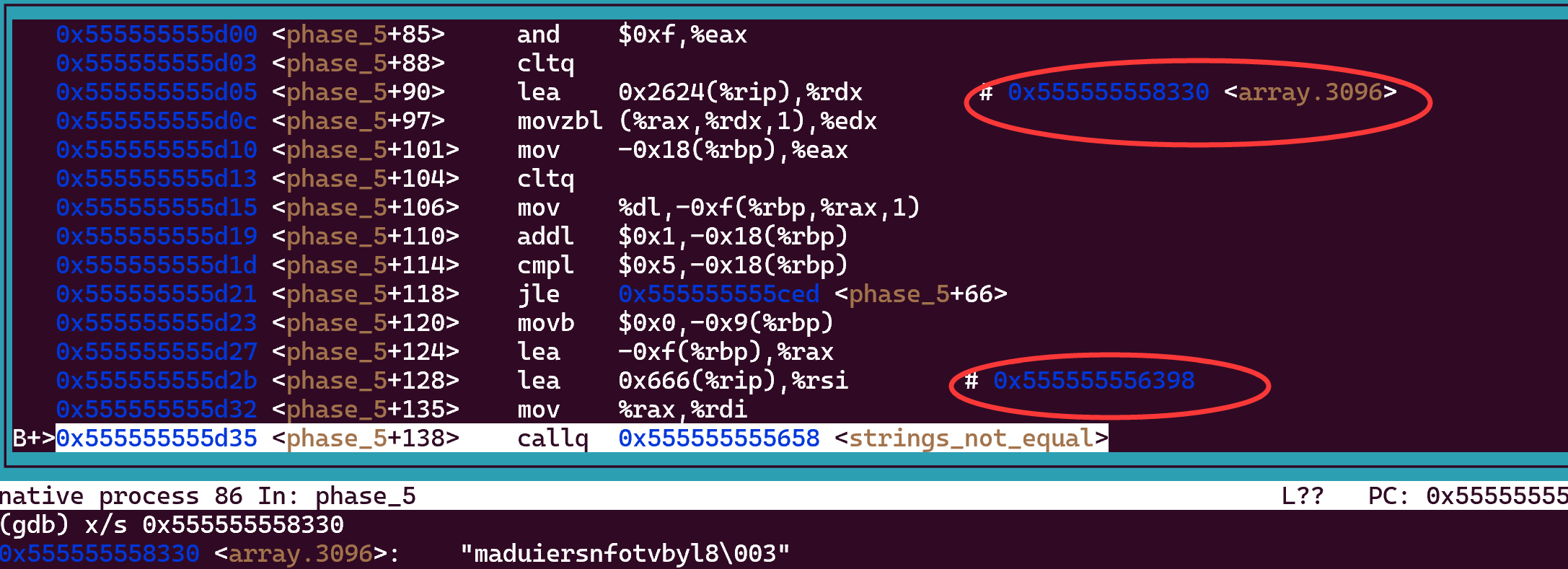


图5-3. phase\_5查看字符串

第一个字符‘b’在数组中下标为13，所以我们要输入的字符串的第一个字符应为ASCII码最后4位为13，如‘m’，同理可以获得其它字符，最终得到字符串“mfcdhg”，为最终答案。

**阶段6：链表、结构、指针的访问（phase\_6）**

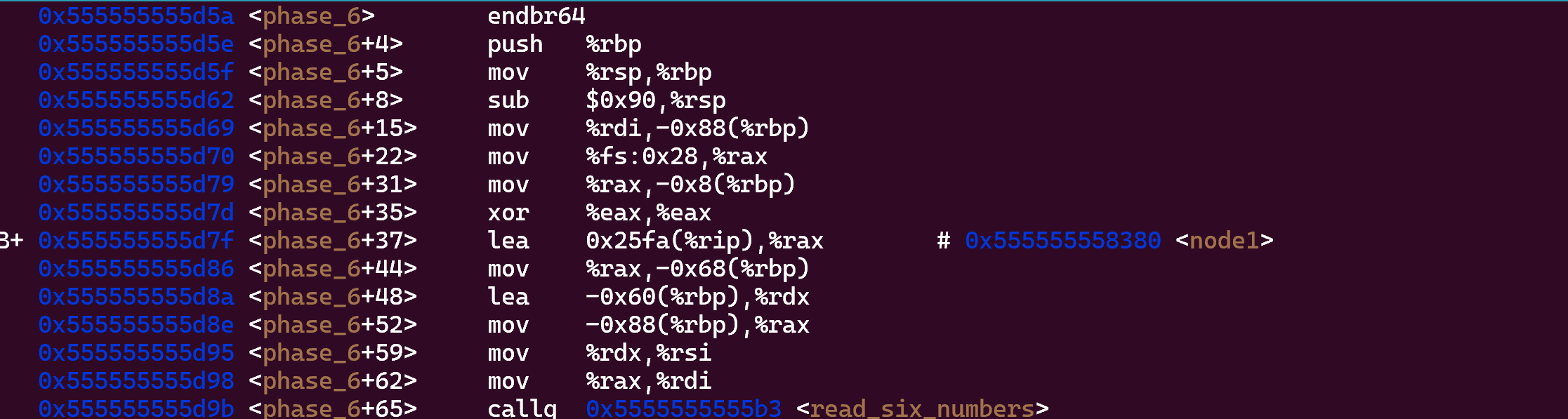


图6-1. phase\_6汇编语言源程序（部分）

在反汇编代码中可以找到传入了一个名为<node1>的地址，存储的是第一个结点的内容，查看这个地址，可以看到第1个字储存了值为0x119的数，第2个字存储了1，最后2个字内存储的为0x555555558370，与node1的地址十分接近，猜测为node2的地址，于是再次查看0x555555558370处存储的东西，同样，第1个字存储的值为0x38b，第2个字存储的值为2，最后两个字存储的值应该是node3的地址。

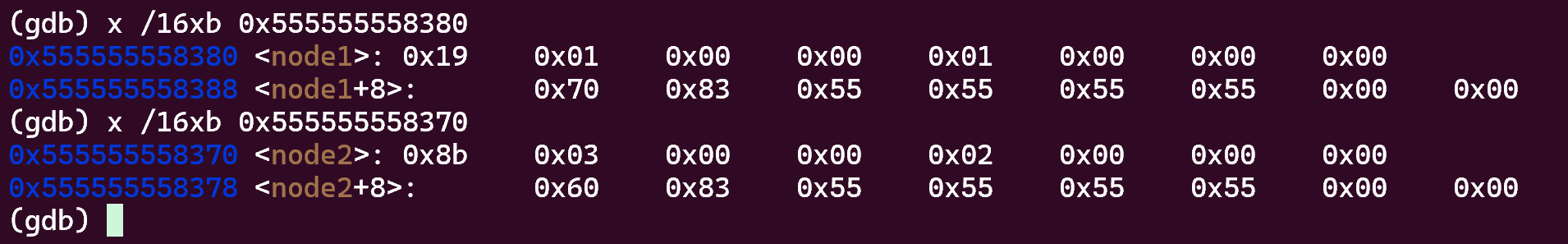


图6-2. 地址内容查看

依次类推，可以得到6个结点中的内容，格式大致为：值（4个字节），编号（4个字节），下一个结点的地址（4个字节）。依次查看各结点内容，汇总如下表所示。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| node1 | node2 | node3 | node4 | node5 | node6 |
| 0x119 | 0x38b | 0x142 | 0x79 | 0x338 | 0x210 |
| 0x555555558370 | 0x555555558360 | 0x555555558350 | 0x555555558340 | 0x555555558220 | 0 |

由于需要降序排列，所以本关答案为2 5 6 3 1 4。

1. **体会**

通过本次实验，我基本了解了gdb、gcc等工具的使用，加强了反汇编代码的阅读能力，对于循环调用、条件分支、数组指针等类型的代码的反汇编形式以及链表、指针、数组等在内存中的存储形式有了更深的理解。同时，通过应用所学的知识解决一些简单的问题，也进一步加深了我对于课堂知识的理解。