**实验报告**

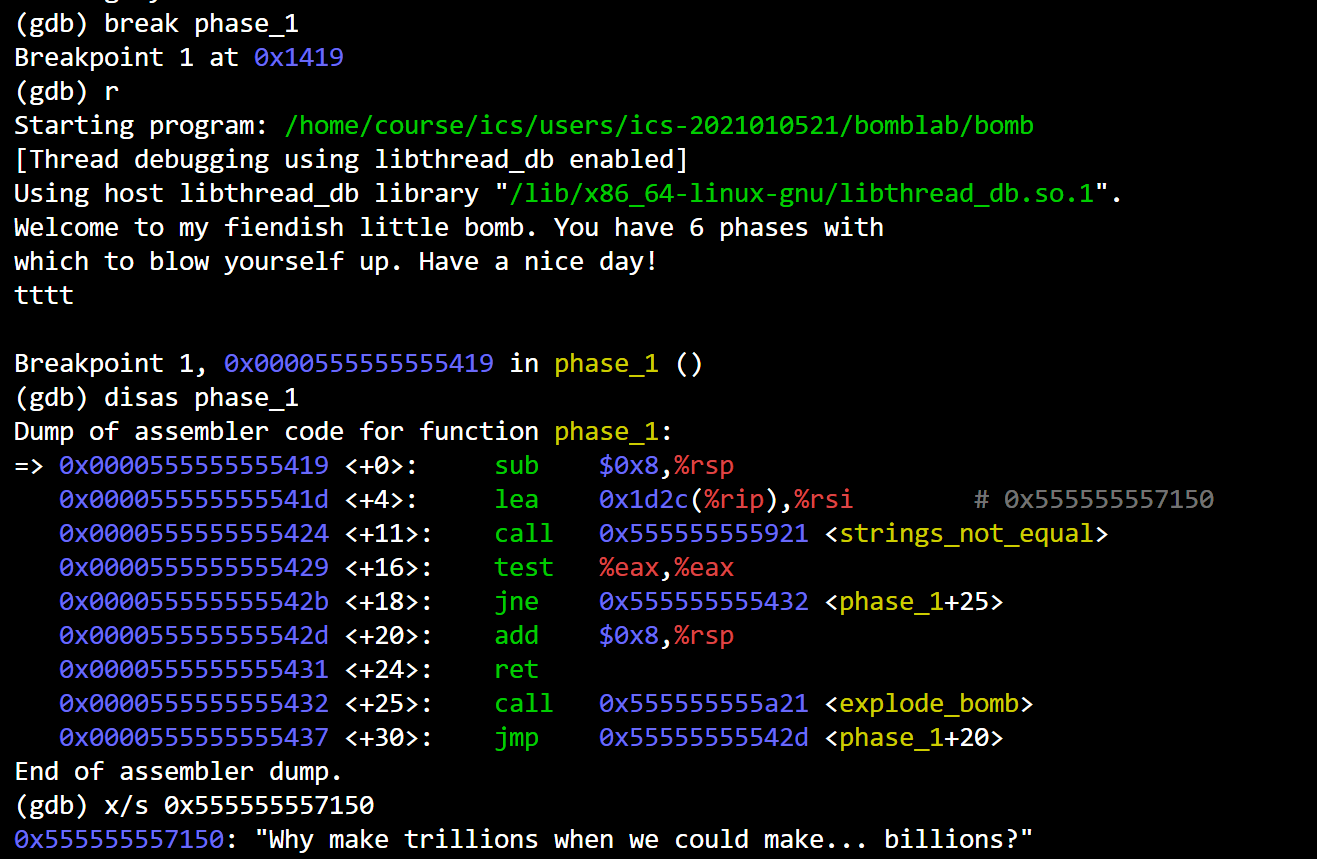
**刘雅迪**

**计26**

**Phase\_1**

考查的两字符串是否相等。

通过disas指令得到phase\_1的汇编代码，从代码中可以得知在<+11>行中call了strings\_not\_equal函数，猜测作用是判断输入的字符串是否等于目标字符串，而在call函数之前将一个地址的值存到了%rsi寄存器中，故使用x/s指令查看地址中的值，从而得到phase\_1的输入。



Phase\_1具体操作

**Phase\_2**

考查循环。

由汇编代码中call了read\_six\_numbers函数可知，输入的格式是六个数字。

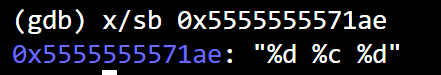
寄存器%ebx中存了当前数与前一个数的差值，%ebx初始值为1，而在循环中%ebx会每次加1。从第一个数0开始，通过%rbp指针的移动，来看输入的数是否满足程序要求的差值。当%ebx等于6时，说明6个数全部判断完毕，%rsp指针恢复，弹出%rbx和%rbp，程序结束。



Phase\_2中循环主体

**Phase\_3**

考查switch-case语句。

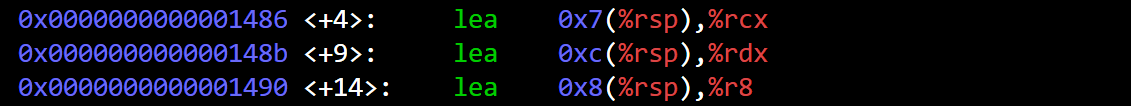
由于call了sscanf函数，所以通过查%rsi存的值可以得知phase\_3的输入格式，如下图：

查phase\_3的输入格式

sscanf会将参数str的字符串根据参数format字符串来转换格式并格式化数据，转换后的结果存于对应的参数内，成功则返回参数数目，失败则返回0，返回值保存在%eax中。

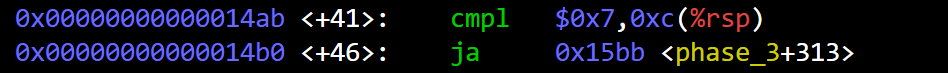
寄存器%rdi :存一开始的input，%rsi存输入格式“%d %c %d”，

所以输入的三个数分别依次由寄存器%rdx，%rcx，%r8储存，它们的地址分别为0xc(%rsp)、0x7(%rsp)、0x8(%rsp)，如下图：



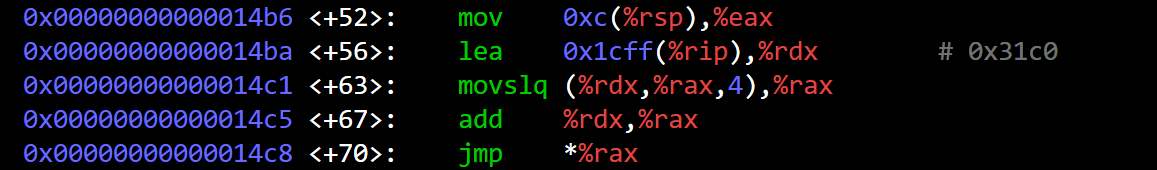
寄存器的地址分配

由第<+41>和<+46>行可知第一个数%rdx的值小于7，如下图：



%rdx的值范围

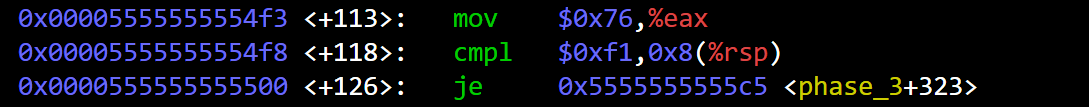
第<+52>~<+70>行是switch-case语句的条件判断部分，根据第一个数的不同跳转到不同的case语句，而跳转到的具体地址由%rax的地址决定，如下图：



switch-case条件判断部分

假设第一个数%rdx为1，由于我不知道程序运行中%rax的具体地址，所以我通过gdb调试工具来辅助我判断程序跳转到哪个case语句。

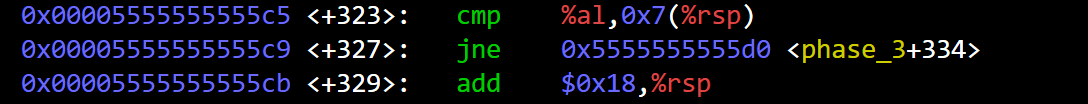
给phase\_3函数和explode\_bomb函数设置断点，输入第一个数为1的三个数，在函数explode\_bomb执行前通过info reg指令查看寄存器%rax的地址，发现为0x76，说明程序跳转到了第<+113>~<+126>行，如下图：



%rdx为1时对应的case语句

此时第三个数%r8 = 0xf1 = 241

而由汇编代码第<+323>~<+329>行可知%rcx的ACSII码为0x76即118，对应的字符是v，如下图：



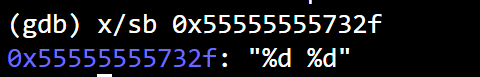
%rcx的值

所以phase\_3的输入为1 v 241，这只是其中的一种情况，改变第一个数的值，后两个数的值也会跟着变化。

**Phase\_4**

考查了递归函数。

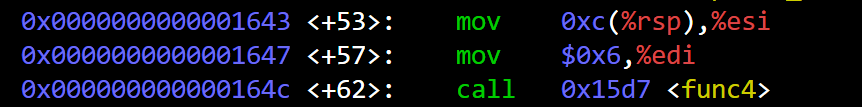
输入格式的判断如phase\_3，如下图：



查phase\_4的输入格式

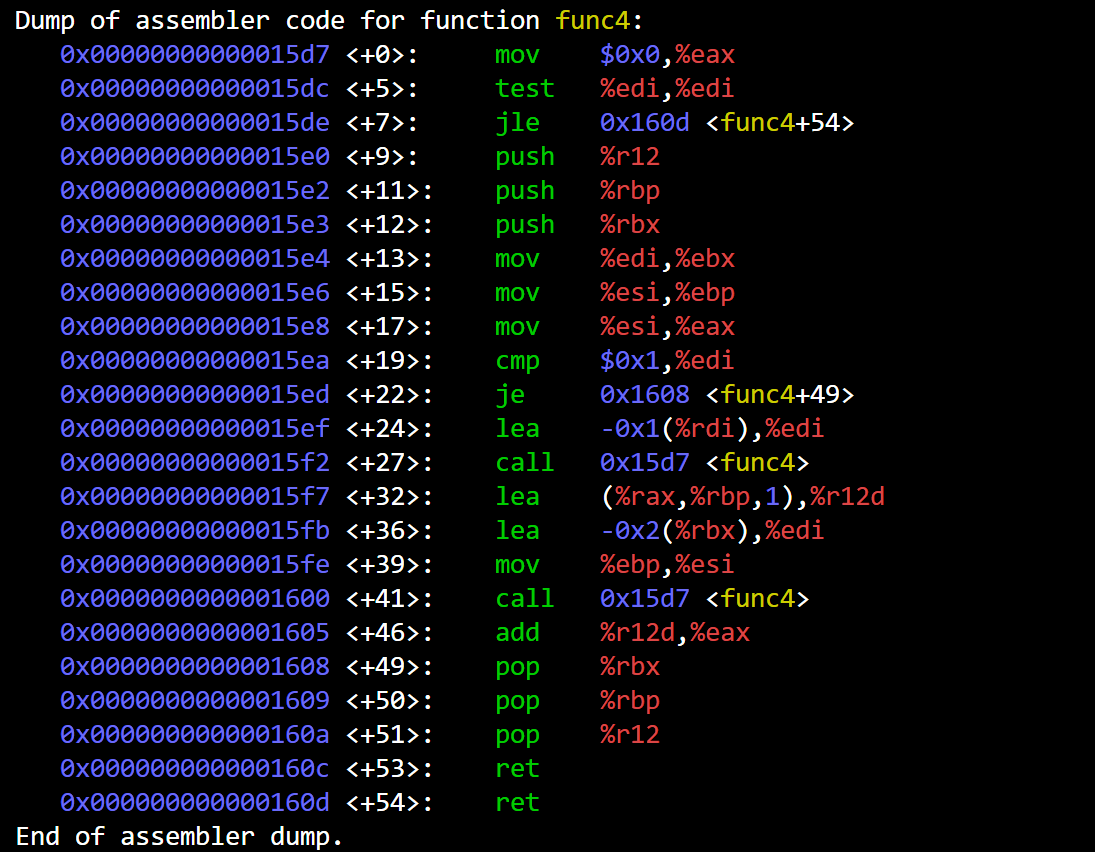
说明输入的是两个数字，分别储存在寄存器%rdx和%rcx中，其地址分别为0x8(%rsp)和0xc(%rsp)。

由第<+53>~<+62>行可知phase\_4调用了函数func4，传入的两个参数分别是6和%rcx，即调用函数func4(6, %rcx)，如下图：



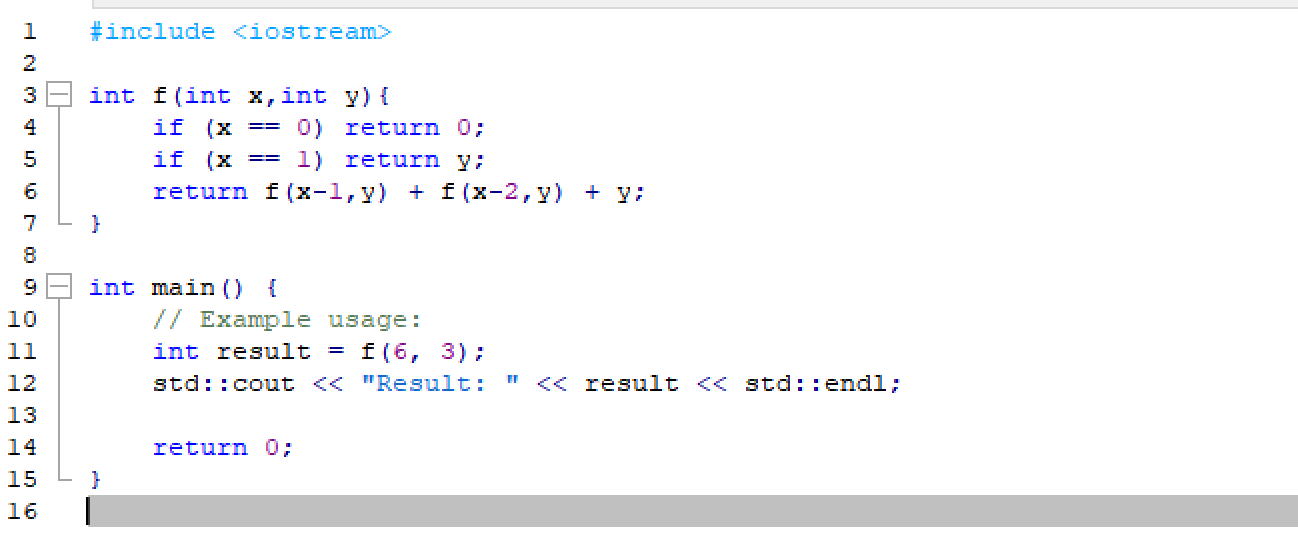
func4函数调用

查看func4函数的汇编代码，如下图，发现它在函数内部分不同情况调用了自身，说明这是个递归函数，%edi和%esi的初始值分别为6和%rcx。

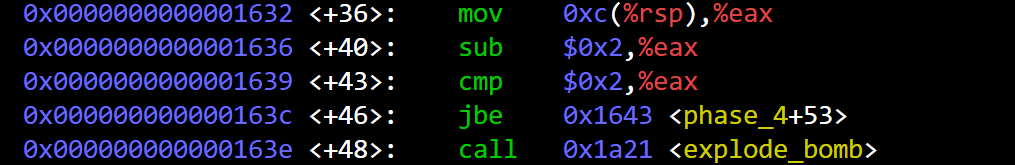


递归函数func4的汇编代码

由于phase\_4使用了递归函数，故我根据汇编代码写出了它的C++函数，方便计算func(6, %rcx)的值，如下图：



func4函数的C++代码



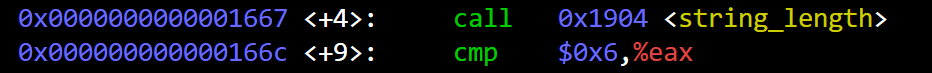
%rcx取值范围

从第<+36>~<+48>行可知第二个数%rcx的取值应小于等于4，不妨令其等于3，算出第一个数%rdx为60。这只是其中的一种情况，改变第二个数的值，第一个数也会跟着发生变化。

**Phase\_5**

考查数组。

通过call了string\_length函数可知输入的是字符串且长度为6，如下图：



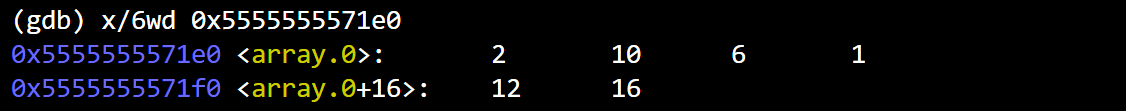
phase\_6输入格式

由第<+51>行cmp $0x32,%ecx可知最后%ecx要等于0x32即50，分析汇编代码可知%ecx初始值为0，每次会加上数组0~5位的值。



Phase\_5汇编代码

而从<+26>行可知%rsi储存的是数组的起始地址，通过gdb查看数组中的数，如下图：



数组中的数

可知数组array[0]~array[5]的值分别为2、10、6、1、12、16。

在这六个数里面凑六个相加等于50的数，如50=16+16+12+2+2+2，对应的数组下标偏移量为5 5 4 0 0 0。

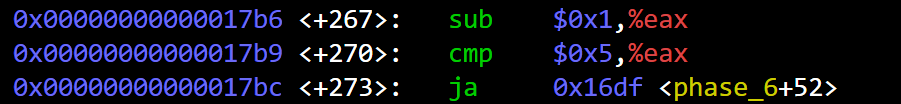
而由汇编代码第<+36>and 0xf %edx可知，只取原字符的低四位，如字符’A’，ASCII码为0x41，取低四位后得到0x01，所以查ACSII后可知，输入的字符串可以是eedppp。

**Phase\_6**

考查链表和结构体。

Phase\_6的汇编代码很长，我看了好久之后才大致看懂。可以分几个部分来分析：

首先由call read\_six\_numbers函数可知phase\_6输入的也是六个数字。



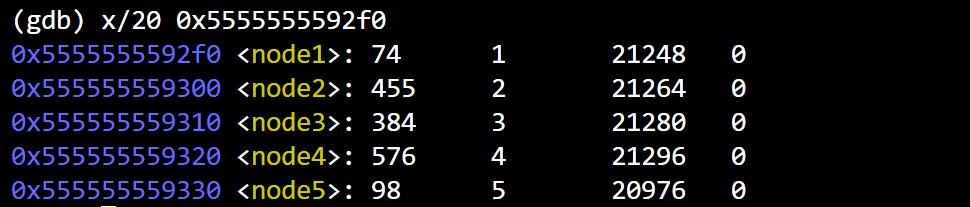
输入数字的范围

从<+270>行得知%eax减1后≤5，所以输入的数均≤6。

第一部分：

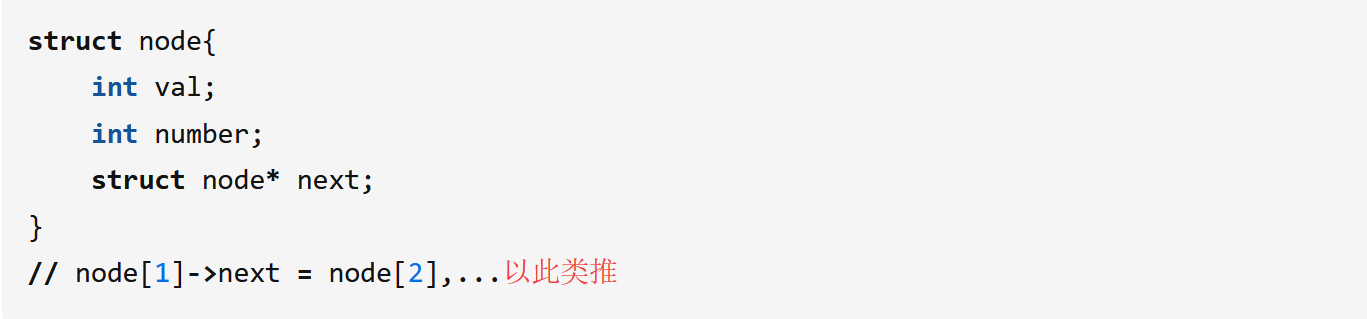


通过<+114>行我们可以得到nodes的值和地址，如下图：



nodes的值和地址

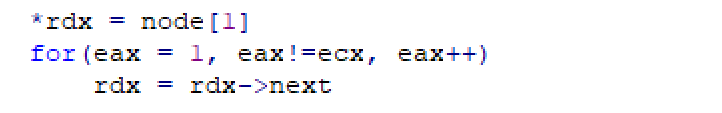
这是个结构如下图的链表：



链表结构

而上图这段汇编代码包含了内外两个循环，设输入的第i个数字为n[i-1]，第i个结构体为node[i]。第<+137>行可知0x10(%rsp)是node[1]的地址，而<+105>行可知%ecx = n[i]。

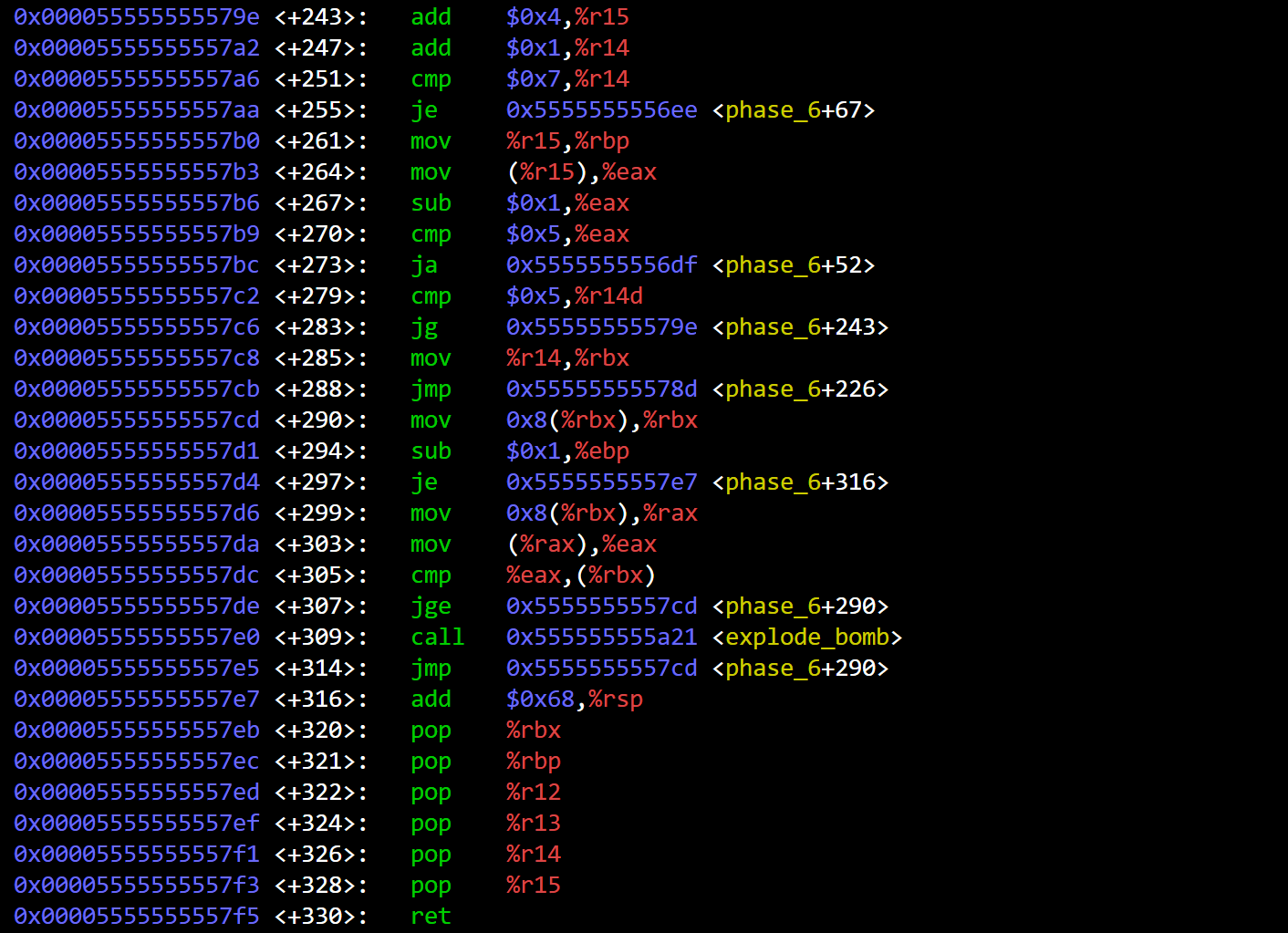
第<+109>~<+135>是内循环，可以写成如下for循环：



内部for循环

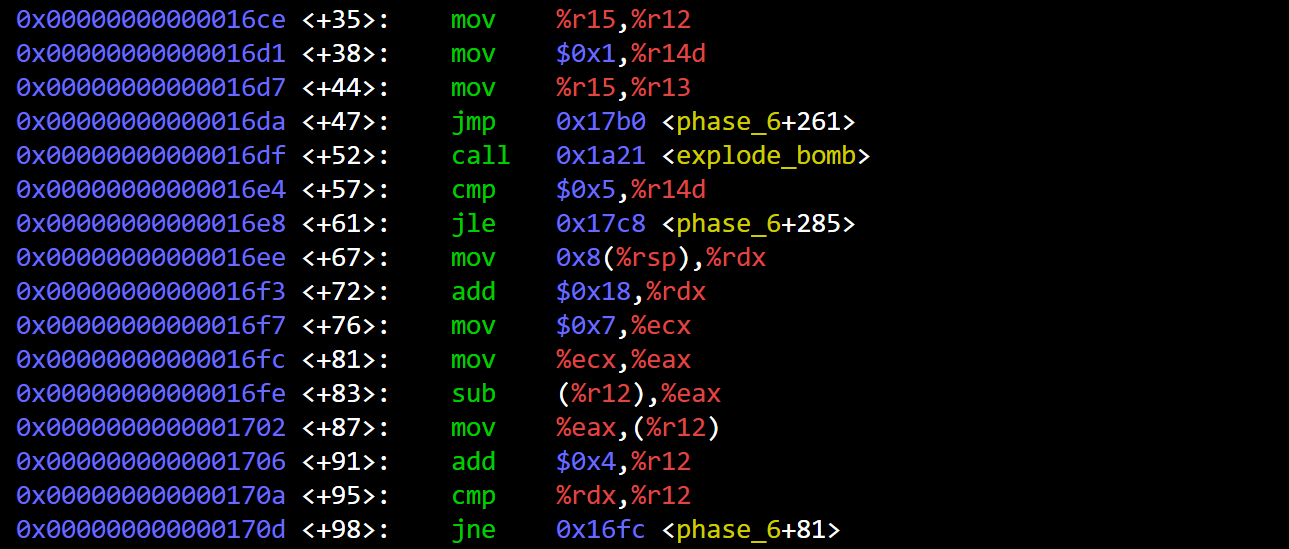
表明指针移动了n[i] - 1位，到了node[n[i]]结点，将这个结点压入栈，外循环则是说明另5个n[i]也这样做。所以从这段汇编代码我们可以知道程序想要将node结点们重新排序。

第二部分：



这段代码通过比较(%rbx)和0x8(%rbx)的值，告诉我们新的nodes应该是个降序。而node[4]>node[2]>node[3]>node[5]>node[1]，故顺序为4 2 3 5 1。

第三部分：



由<+76>~<+87>可知，程序将原始的数据进行了处理，%r12 = 7 - %r12，故最后还需用7减去得到的数字，才是我们输入的原始数据。

用7分别减去上述的4 2 3 5 1，得到3 5 4 2 6，而第一个数为1，故输入的数据为1 3 5 4 2 6。

**实验感想**

汇编的学习对我来说和之前学习微积分/线代等数学的感觉完全不一样，我经常在课堂上听的一头雾水，或者以为自己听懂的部分实际上没有完全听懂，导致作业下不了笔。写这次实验前我是十分恐惧的，因为觉得自己很多知识还没有搞懂，不知道能不能把实验写完，于是拖了很久才开始打开实验的文件。

在写phase\_2时我就被难住了，我又回去把C语言-3的课件以及回放看了一遍，才理解了一点栈指针%rsp和帧指针%rbp的使用，也读懂了phase\_2其实考查的是一个循环，而在编输入数字的过程中我又因为搞错了地址的加和值的加而卡在了这一关，最后还是求助同学才知道我错在哪。

平心而论，这次实验对我来说还是很有难度的，由于每个人的代码都不一样，我也无法在网上找到参考资料，只能根据自己的理解编写输入的数据。又因为对汇编知识的不太熟悉，导致对别人来说可能一天就能写完的实验我写了好几天。但是在看到完成最后一个phase后终端出现的”congratulations”的时候，还是很高兴很有成就感的。我完成了一个对我来说有点难的任务，并且在这个过程中我又回顾理解了好几遍课件内容，还因为课件部分内容写的有点简略而又去阅读了CSAPP课本。随着对汇编知识点和实验的熟悉，明显感觉到解决后面的phase比解决前面的更得心应手，虽然后面的phase更难更复杂。

通过本次实验我学到了很多处理汇编语言的技巧，也巩固/重新学习了课堂上没有搞懂的知识，收获很大，也非常感谢助教和老师将本次实验的截止时间设置的那么晚，让我有机会完成本次实验。

就我来看，bomb实验本身已经很精巧了，重要的知识点都考查到了，我暂时想不到什么创新点。但是我觉得可以给每个phase的输入排除一些“随便猜就可以猜中的答案”，要求同学们必须认真阅读理解汇编代码或者使用gdb调试工具后才能得到phase的答案。因为我有朋友在写phase\_6时一开始就随便输入了6 5 4 3 2 1，结果就过了，感觉对于phase\_6这个比较复杂的phase来说有点荒谬与不公平。