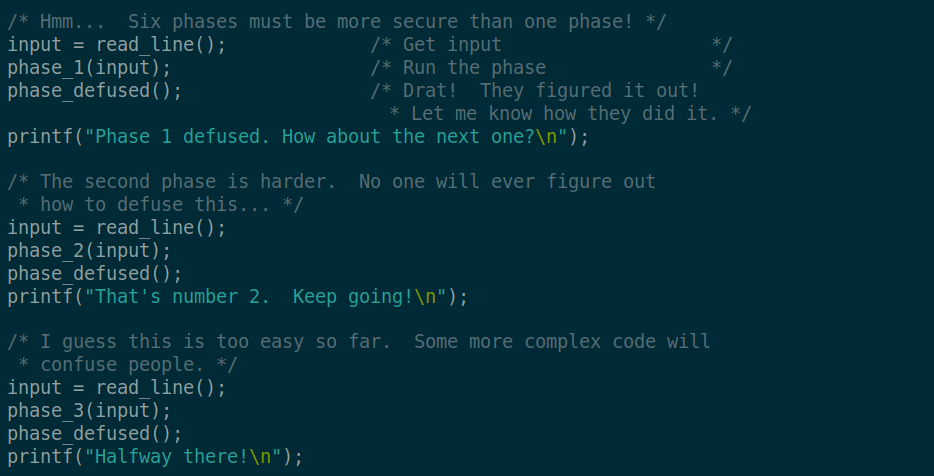
首先我们观察bomb.c文件的代码：

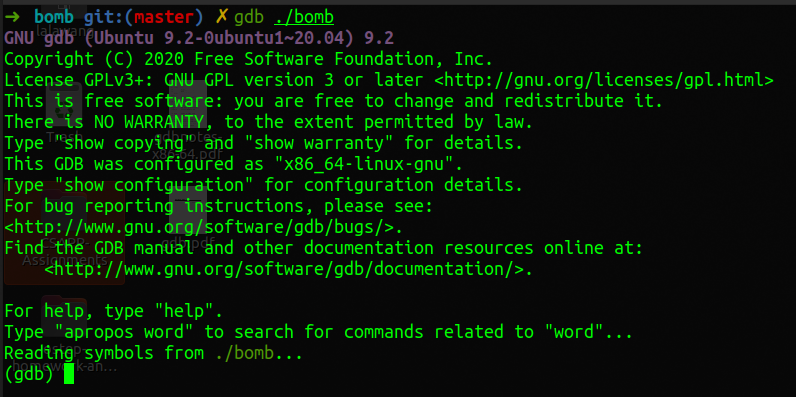


可以很明显的观察到，判断输入字符串是否是接触炸弹所需要的逻辑存在于phase\_1、phase\_2...phase\_6这几个函数中，我们只需要搞清楚这几个函数内部的逻辑，就可以获知解除各个炸弹所需要的字符串内容了。

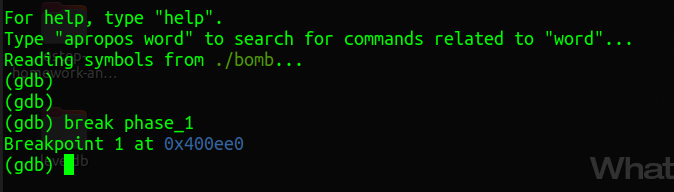
接下来分别讲述解除6个炸弹的过程：

1. phase\_1:

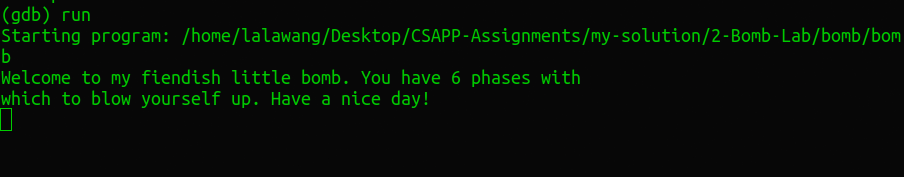
首先通过gdb调试./bomb可执行文件：



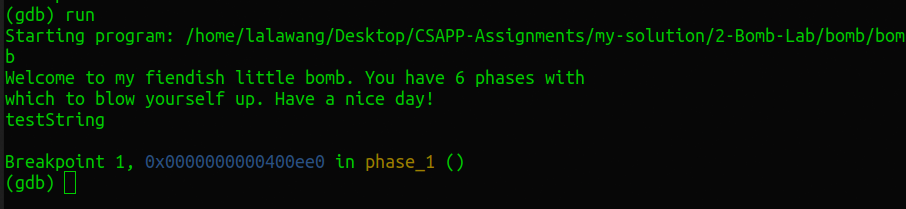
按照上面所述的逻辑，我们在phase\_1函数处打一个断点：



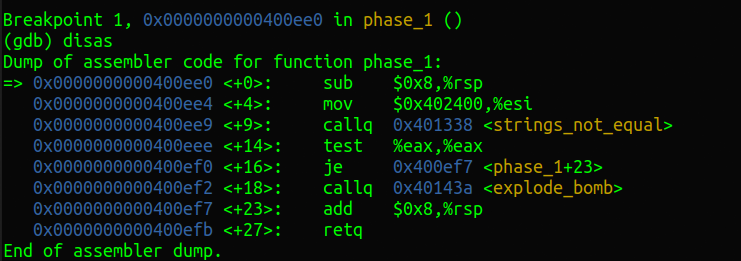
接下来输入run命令运行：



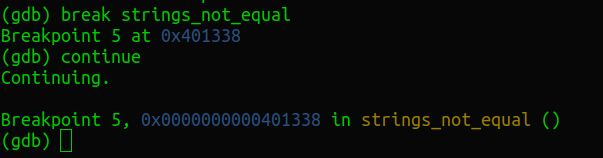
此时程序向我们索要第一个解除炸弹的字符串，这里先输入testString, 随后程序将在我们打的断点处停止：



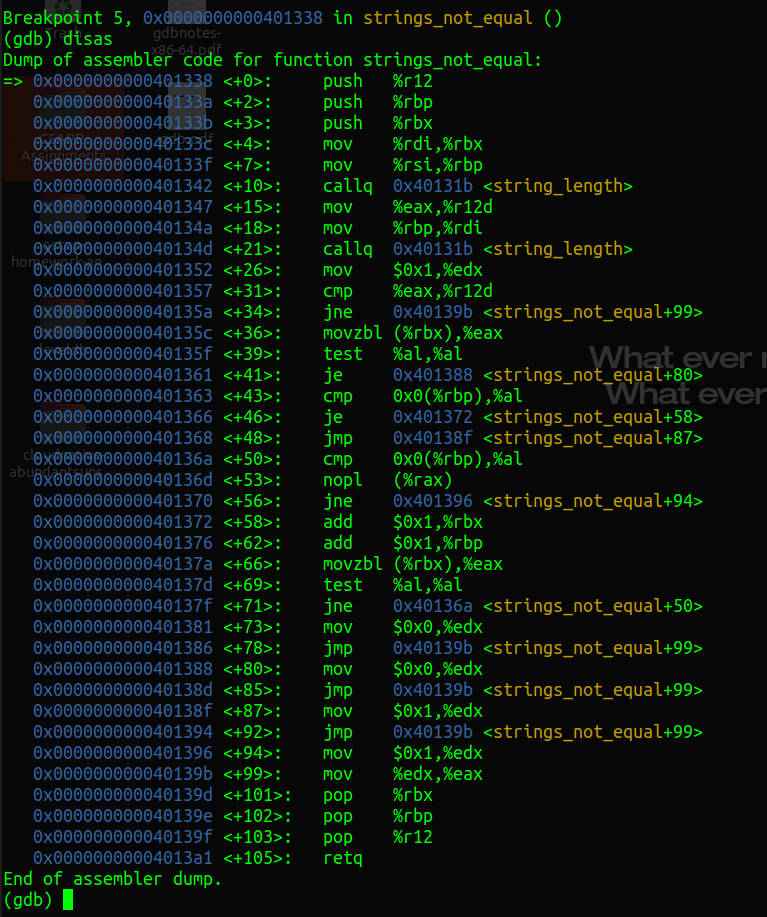
为了搞清楚phase\_1函数内部发生了什么，这里用disas命令将函数解汇编：



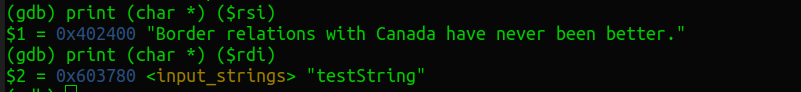
观察此处汇编代码的逻辑，实际上就是根据函数strings\_not\_equal的运行结果来决定是否调用explode\_bomb函数，如果调用该函数，则本次解除炸弹失败，因此我们需要了解函数strings\_not\_equal做了什么，在这个函数处也打一个断点, 同时让函数走到这个断点处：



再一次地，对strings\_not\_equal这个函数解汇编：



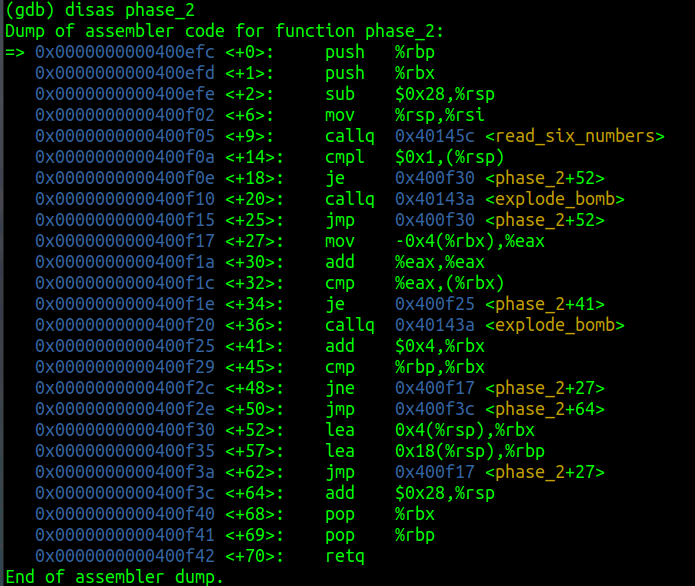
注意看<+10><+21>两处的代码，实际上是调用了两次string\_length函数来获得输入字符串和输出字符串的长度，并将结果存在%r12d和%eax寄存器里面，若果说长度都不相等，那么直接跳转至<+99>处，所做的事情就是将寄存器%eax的值置为0x1，而回过头来看phase\_1函数的汇编代码，可以发现如果%eax寄存器的值不为0，炸弹就会爆炸，因此我们可以推断两次调用string\_length的函数，是为了获得输入字符串和目标字符串的长度的，我们只需要看两次函数调用对应的参数即可，也就是参数寄存器%rsi和%rdi对应地址里存的值：



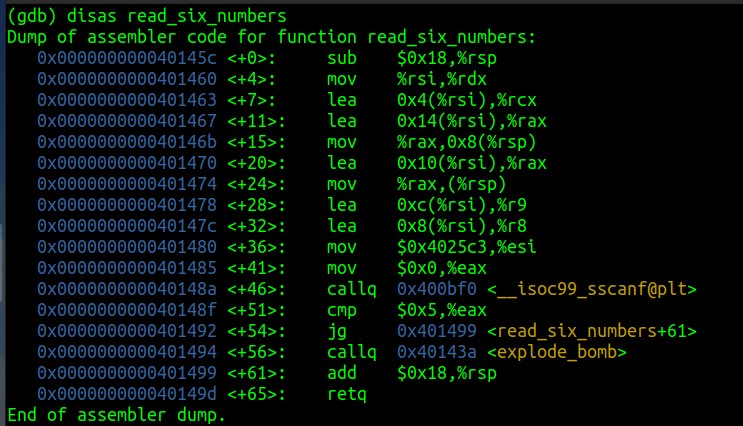
rdi是我们此前暂时输入的测试字符串testtring，这也验证了此前的猜想，那么另一个就是解除炸弹所需要的了。

1. phase\_2

和上述步骤一样，首先对phase\_2用disas命令解汇编：



阅读此处汇编代码,首先第一个关注点就在于read\_six\_numbers这个函数，从函数名上看推断，这个函数的功能应该是读入六个数字，对该函数解汇编：



可以看到，read\_six\_numbers这个函数的主体逻辑主要就是三步：

1. 给参数寄存器中传入相应参数，<+0>~<+41>
2. 调用\_\_isoc99\_sscanf@plt函数，<+46>
3. 跳转，若寄存器%eax中的值不大于五，则引爆炸弹，否则返回。<+51>~<+65>

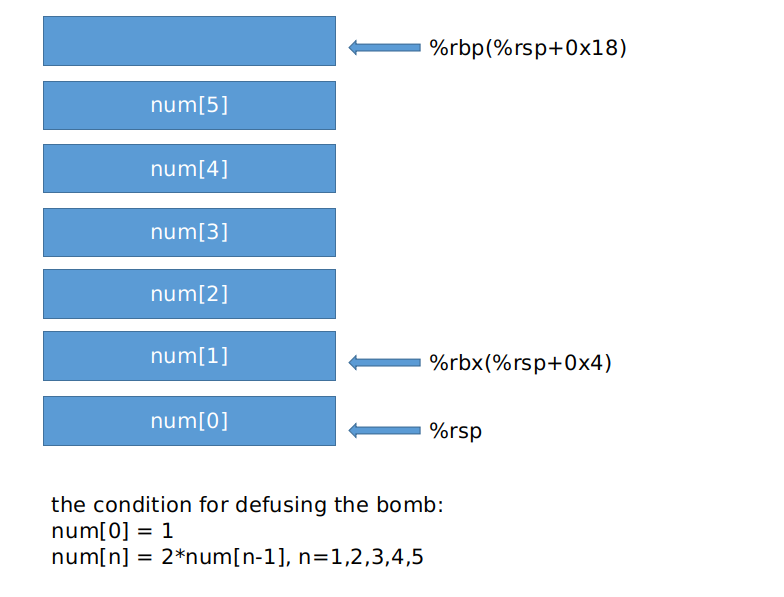
结合函数名和炸弹在此处起爆的逻辑，可以推测出此函数确实是在读取输入的数字，如果输入的数字数量小于6，那么炸弹将直接起爆，只有当数字数量大于等于6时才进行下一步的判断。

再观察phase\_2函数调用完read\_six\_numbers函数后的处理逻辑：

可以看出此处是一个循环处理的逻辑：

1. 首先判断%rsp内存的地址对应内存中数字是否为1，是则继续，否则爆炸
2. 将%rsp+0x4和%rsp+0x18分别赋值给%rbx和%rbp，刚好是20个字节，能够存储五个int类型的数据
3. 循环判断，%rbx指向当前判断的数字，%rbp指向循环的边界，每次判断%rbx指向的数字是否是(%rbp-0x4)指向的数字，也就是上一个内存单元中存储的数字的两倍，如果是，则继续循环，否则引爆炸弹。

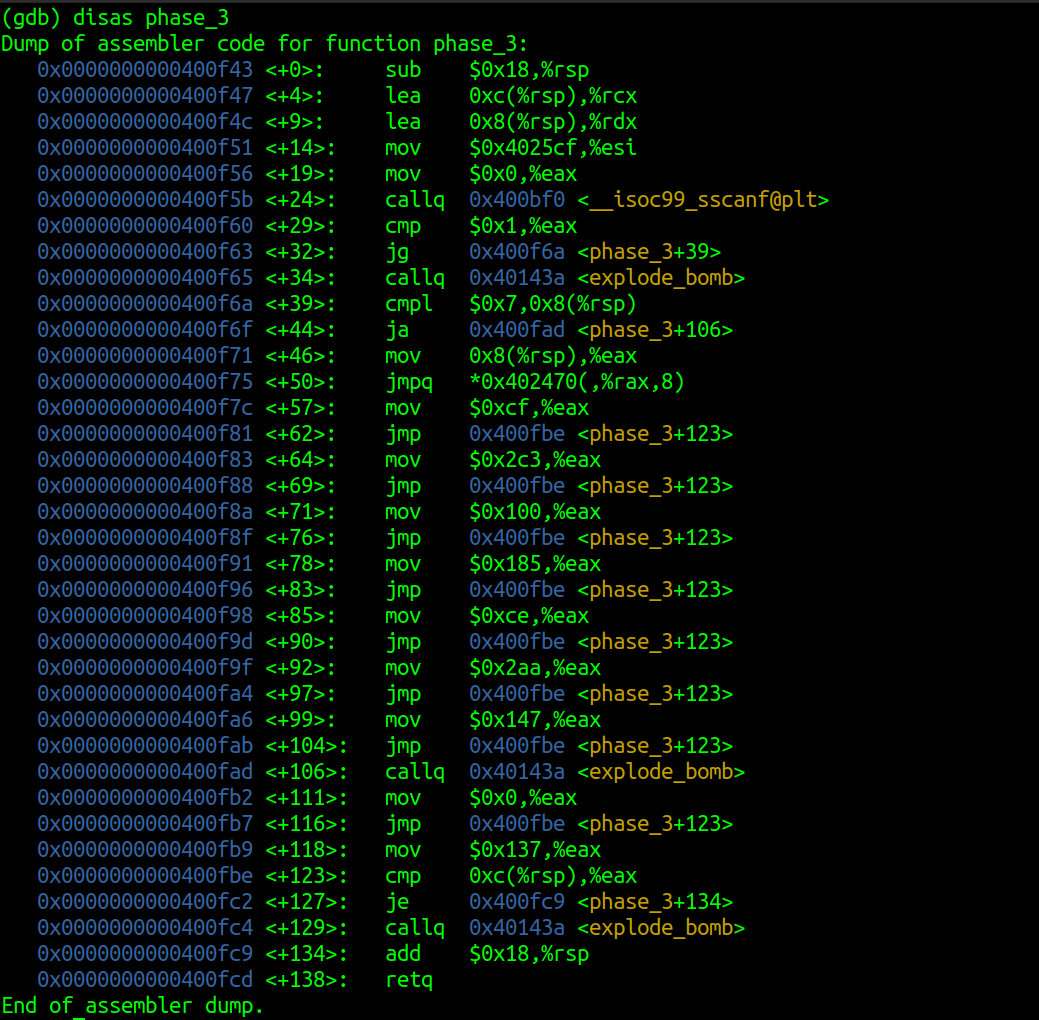
对应的内存布局如图所示：



因此这六个数字应该是1 2 4 8 16 32

phase\_3:

首先对phase\_3函数解汇编：



这一段代码比较简单，主要逻辑为：

<+0>~<+44>: 调用\_\_isoc99\_sscanf@plt函数读取输入，输入参数需要大于一个，否则爆炸；根据之后的代码可以判断是读入两个参数，分别存于%rsp+0x8和%rsp+0xc所指向的内存单元之中，第一个数字若大于7则爆炸；

<+46>~<+118>:这里是一段类似于switch语句的逻辑，根据第一个参数的值来确定跳转到的地址，而跳转之后的逻辑就是将一个固定的值赋值给%eax寄存器

<+123>：比较第二个输入的值是否和此前根据第一个数字赋给%eax的值一样，若不一样则爆炸。

那么只需要保证输入的两个数字有汇编中的对应关系即可，即

0 207

1 311

2 707

3 256

4 389

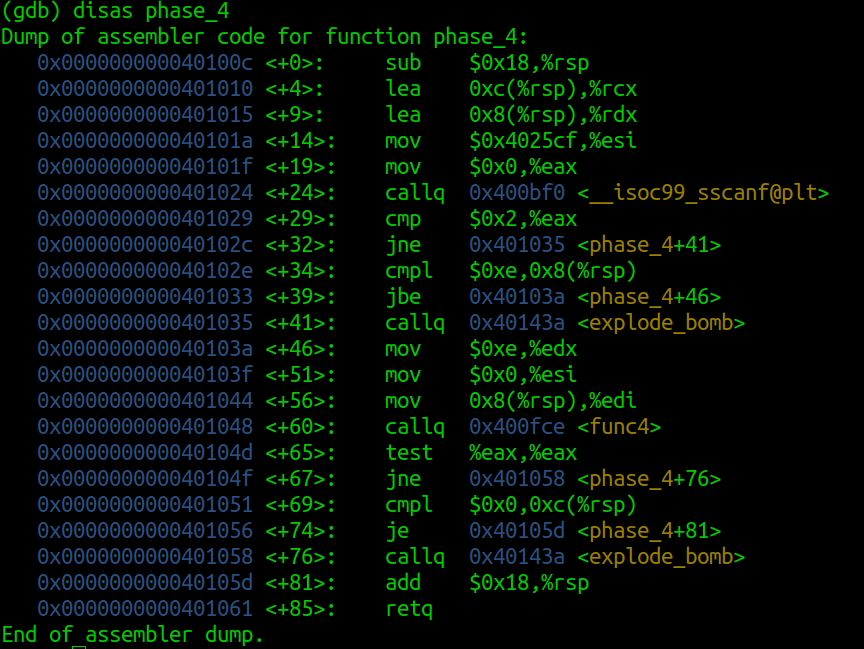
5 206

6 682

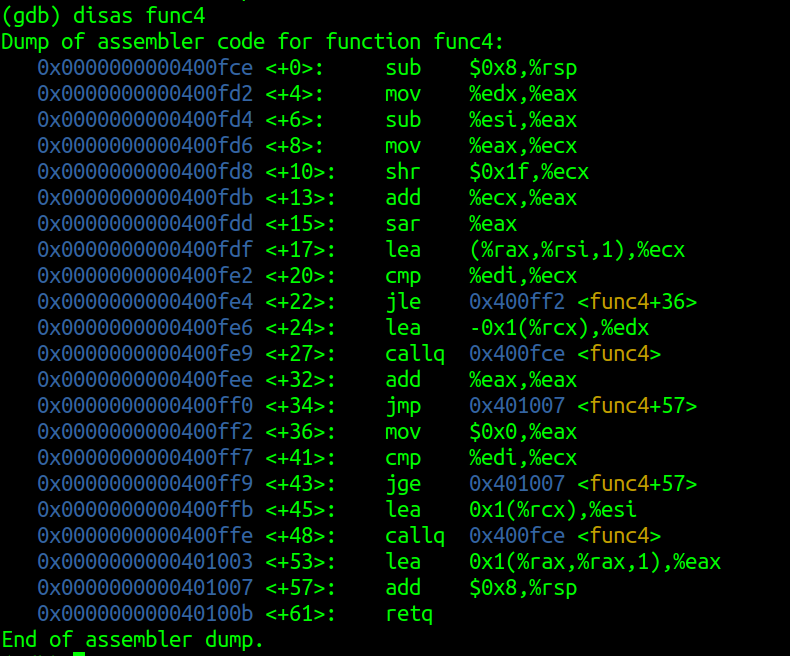
7 327

phase\_4:

对phase\_4解汇编：



需要输入两个数字，第一个数需要小于等于15，第二个需要是0

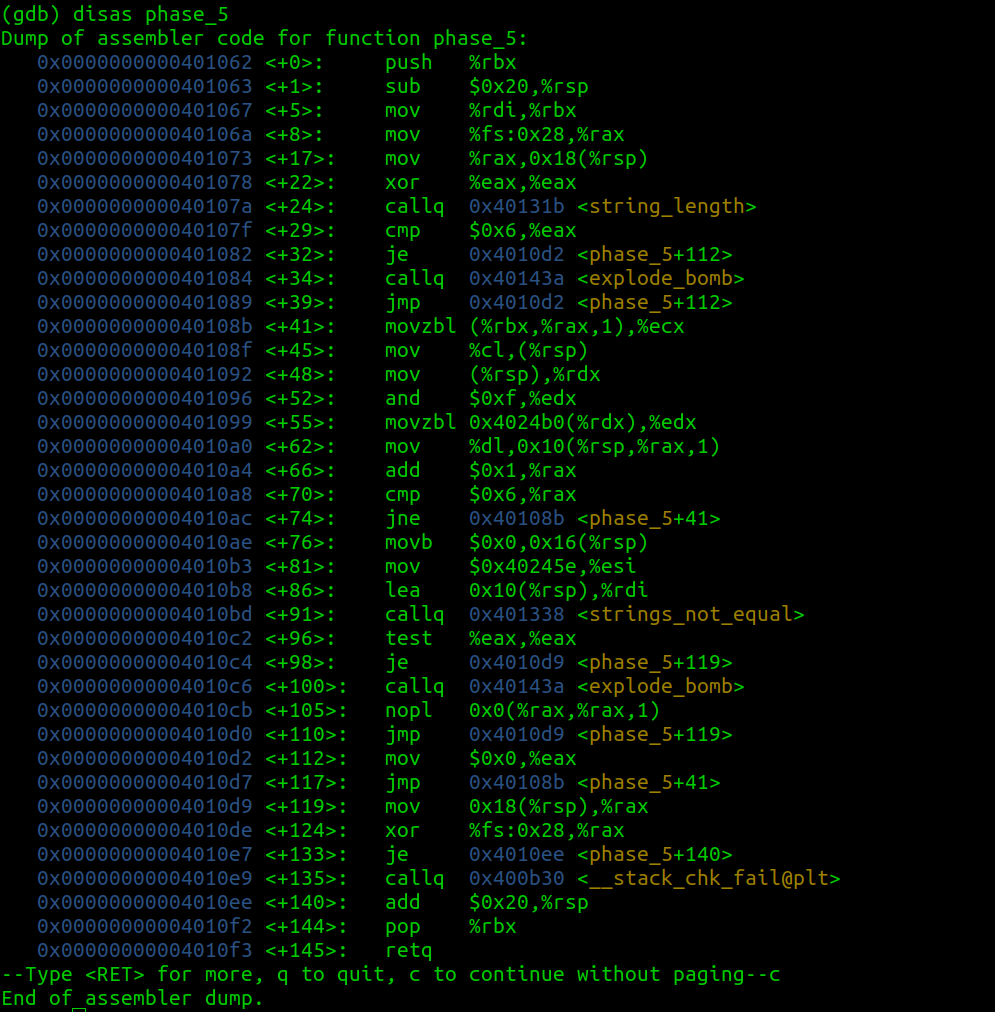


func4是一个递归程序，通过对参数的分析可以看出是二分查找,返回值需要为0，需要注意的是递归有两个分支，分别是向左递归和向右递归，当找到目标数字之后，%eax为0，但在层层返回的过程中，向左递归返回前会将%eax乘二，而向右递归的返回前会将%eax乘2加一，一旦发生乘二加一，那么最终%eax的值就肯定不为0，因此目标值必须是在向左递归的过程里找到的值，所以只能是7或3或1或0

最终两个数字就是7 0或3 0或1 0或0 0

phase\_5:

对phase\_5解汇编：



代码中出现了%fs:0x28这个此前从未出现过的代码，查阅文档说明如下：

On x86\_64, segmented addressing is no longer used, but the both the FS and GS registers can be used as base-pointer addresses in order to access special operating system data-structures. So what you're seeing is a value loaded at an offset from the value held in the FS register, and not bit manipulation of the contents of the FS register.

Specifically what's taking place, is that FS:0x28 on Linux is storing a special sentinel stack-guard value, and the code is performing a stack-guard check. For instance, if you look further in your code, you'll see that the value at FS:0x28 is stored on the stack, and then the contents of the stack are recalled and an XOR is performed with the original value at FS:0x28. If the two values are equal, which means that the zero-bit has been set because XOR'ing two of the same values results in a zero-value, then we jump to the test routine, otherwise we jump to a special function that indicates that the stack was somehow corrupted, and the sentinel value stored on the stack was changed.

这句代码并不影响理解phase\_5的意图。

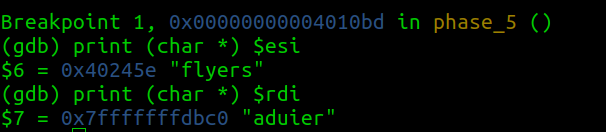
可以看到这段代码中出现了此前出现过的string\_length和strings\_not\_equal两个函数

<+0>~<+34>：输入字符串的长度必须为6否则炸弹爆炸。

<+41>~<+74>:对输入字符串的每一个字符进行处理再存回原位

<+76>~<91>：%esi和%rdi指向的字符串必须相等

暂且输入abcdef进行测试，在0x00000000004010bd处打一个断点，首先来看%esi和%rdi指向的字符串是什么：



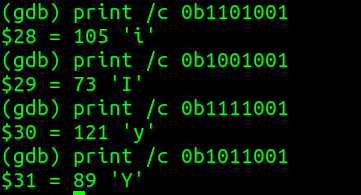
可以看到，abcdef被处理成了aduier，而目标字符串是flyers。那么着重需要理解的地方就是看明白<+41>~<+74>这段代码对输入的字符串做了怎样的处理。阅读此处代码实际上就是将字符串当作数字处理，对该数字与0xf做与运算后得到一个0-15的数字，以这个数字作为偏移，以0x4024b0作为基址去取字符放到指定的位置中去之后再进行比较，那么我们需要看一下0x4024b0存储了什么东西：



由于在经过和0xf进行与运算后，%edx的取值范围就是0-15，因此我们只需要关注这段字符串的前16个，也就是：maduiersnfotvbyl

p.s. 之后的字符串是用于当./bomb运行时键盘按下ctrl+c时会出现的内容

这样就可以推断出，在六次循环中，程序分别希望%edx的值依次为9 15 14 5 6 7



以9为例，那么第一个字母就应该是i/I/y/Y中的一个，同理可得出另外五个字符：

1st char: i/I/y/Y

2nd char:o/O/\_

3rd char:n/N/^/~

4th char:e/E/u/U

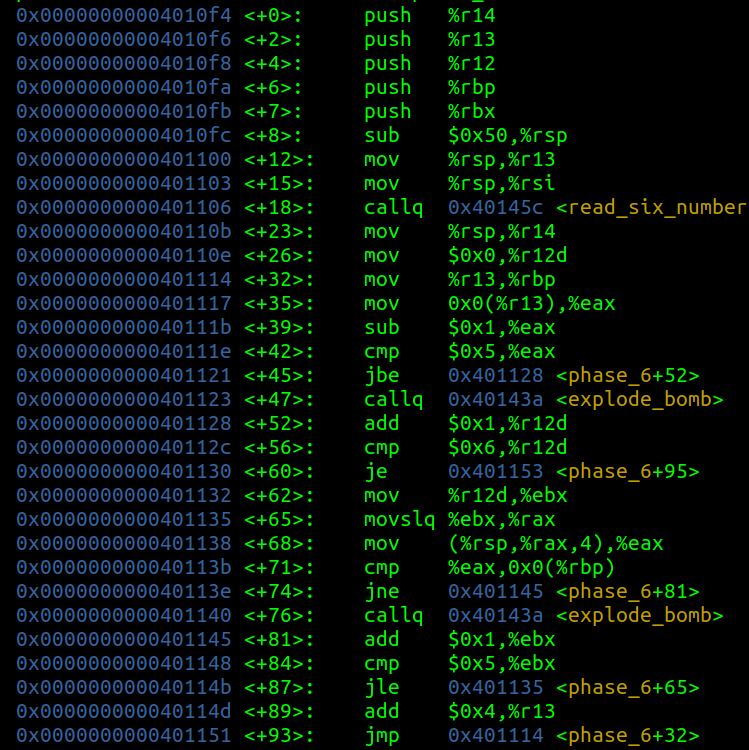
5th char:f/F/v/V

6th char:g/G/w/W

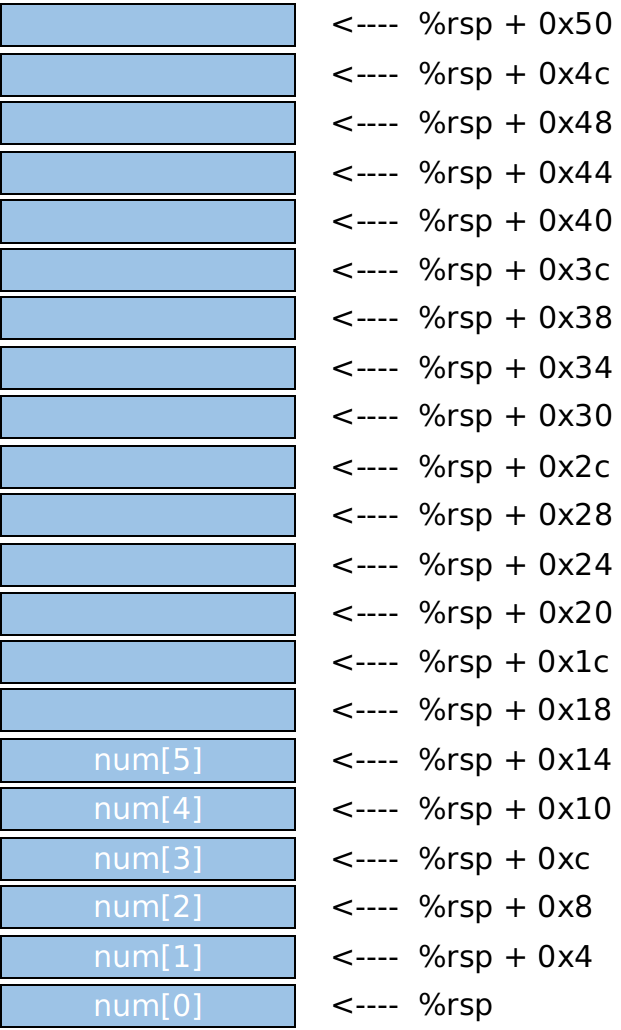
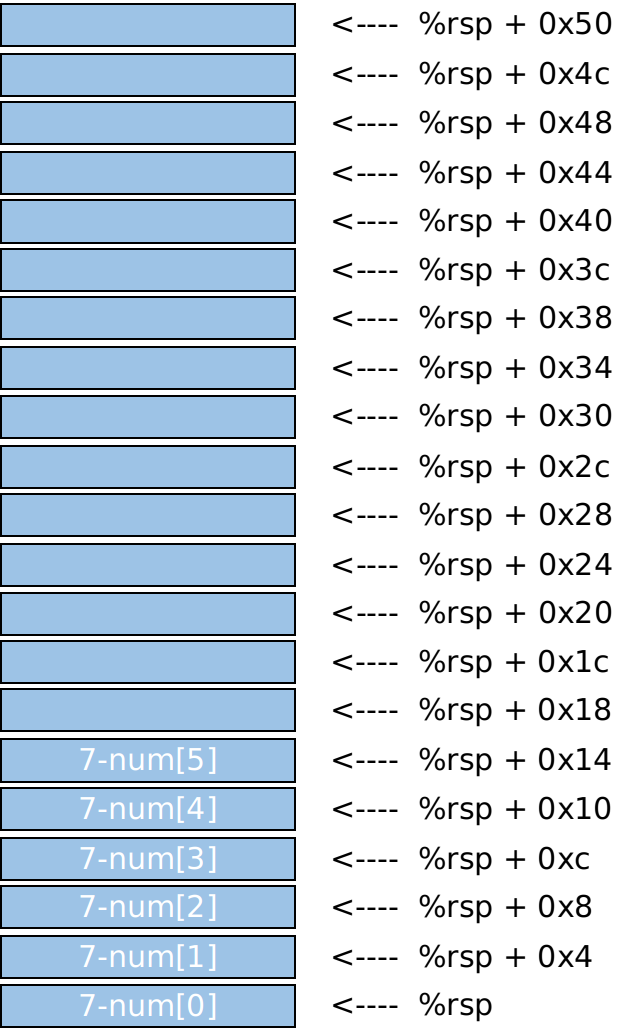
phase\_6:

首先解汇编，在对代码进行阅读后，可以将代码分为以下几个部分单独理解：

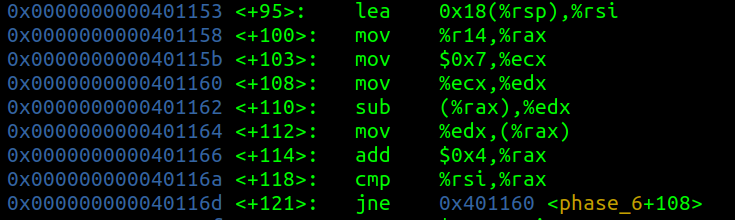
part1：



这一段表明需要输入6个数字，这六个数字必须要大于0小于6(注意<+39><+42>，是先减一再和5比较，因此不能为0)，且互相不相等，所以这六个数字就是123456的排列组合中的一个。此时的内存布局如下图左：

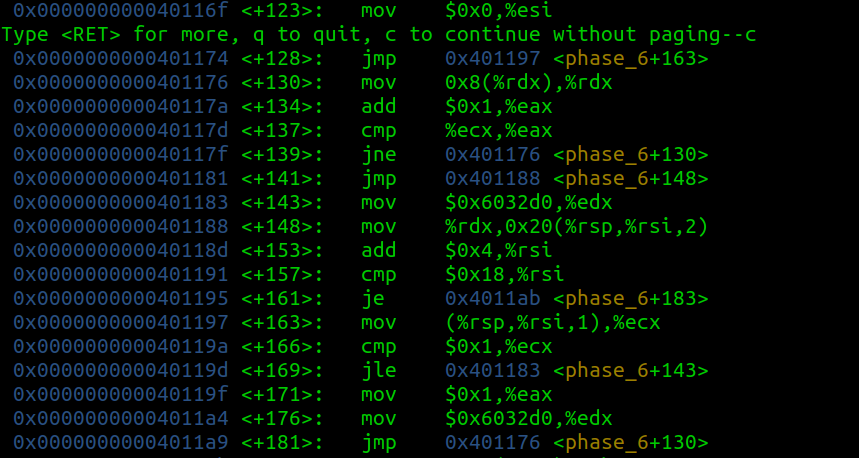
 

part2：

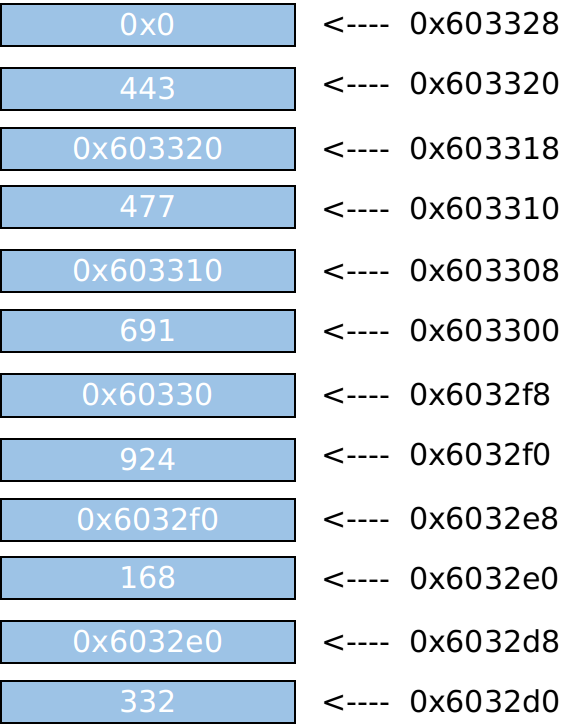
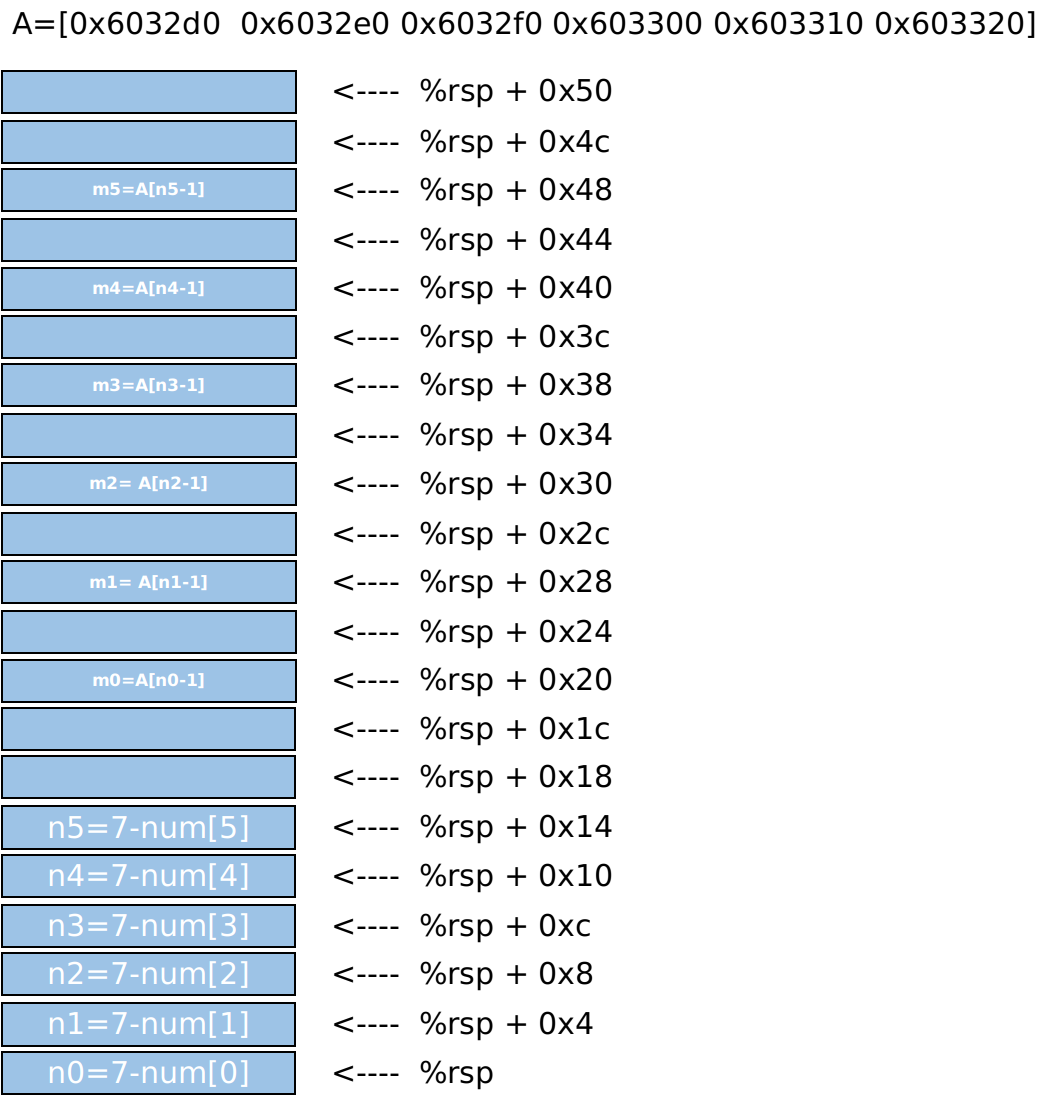


将输入的六个数变成7与该数的差，如上图右

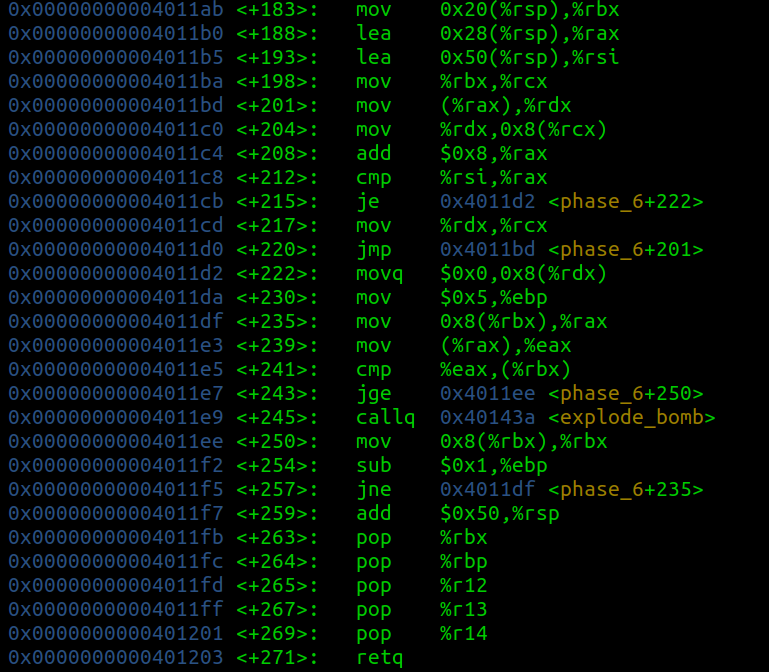
part3：



对应的对内存的改动为：



part4：



需要保证串成的链表按照大小排列，根据数组NUM=[332，168，924，691，477，443]这几个数字可以得知，应该输入4 3 2 1 6 5，被7减去后就是3 4 5 6 1 2，作为NUM索引刚好是降序排列