

**实验报告**



**题目： 拆解二进制炸弹**

**班 级： 2022211305**

**学 号： 2022211119**

**姓 名： 赵宇鹏**

**学 院： 计算机学院（国家示范性软件学院）**

**2023年 11月 12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. 服务器
   1. 服务器：10.120.11.12（X86版本）：**2023年11月14日23:59分关机（三周）；提交作业截止时间2023年11月15日00：00，延迟扣分（每晚一天扣1.5分）**
   2. 服务器：（华为arm版本）：
   3. 服务器密码找回指南：<https://khbgo05wz2.feishu.cn/docx/doxcnioT927j2gDSTwHoAQLGVpg>
3. MobaXterm
4. Linux
5. Objdump命令反汇编
6. GDB调试工具
7. 积分榜（http://10.120.11.13:19220/scoreboard）

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。

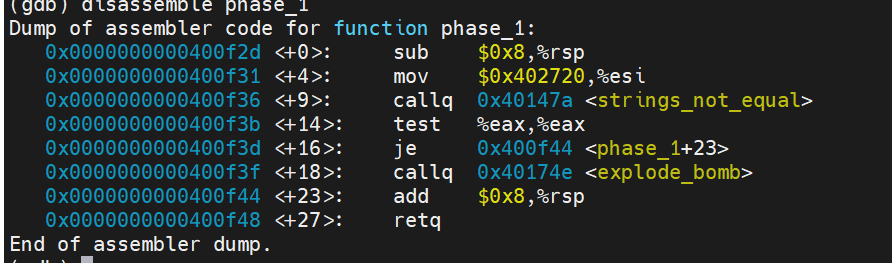
为完成二进制炸弹拆除任务，需要使用gdb调试器和objdump来反汇编bomb文件，可以单步跟踪调试每一阶段的机器代码，也可以阅读反汇编代码，从中理解每一汇编语言代码的行为或作用，进而设法推断拆除炸弹所需的目标字符串。实验2的具体内容见实验2说明。

1. 实验步骤及实验分析

### 普通lab2

**准备工作**：了解汇编语言的基本操作语句含义以及Linux操作系统相关操作，在GDB调试内给 explode\_bomb函数打上断点以防万一。

**Phase\_1 :**

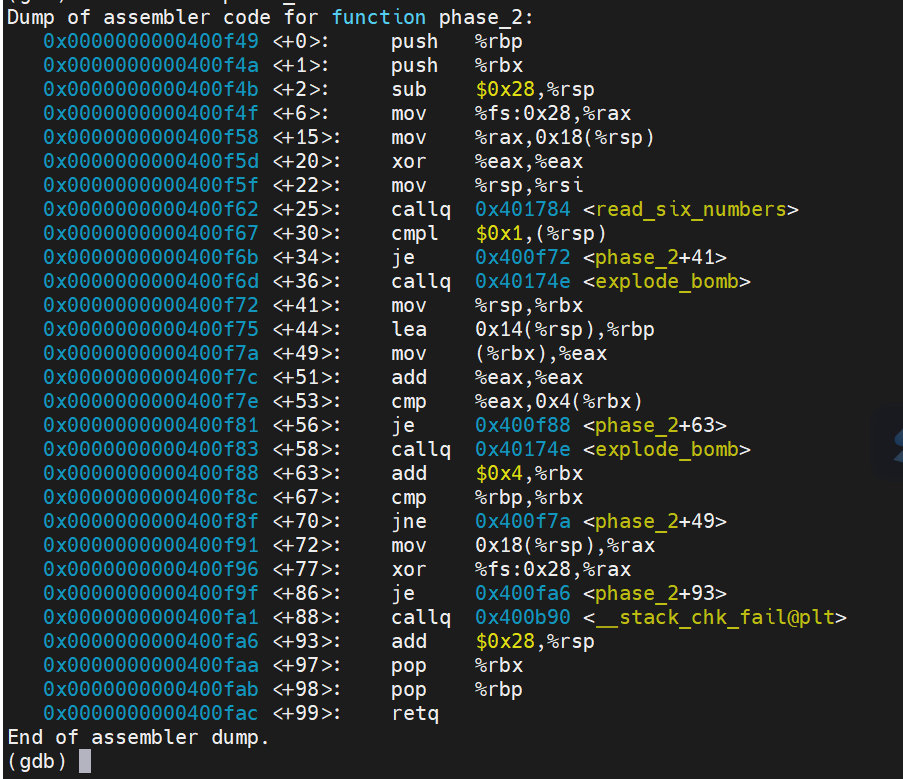


炸弹一结构比较简单，首先申请栈空间，然后将0x402720地址中所储存的字符串传给%esi寄存器，调用string\_not\_equal函数来比对输入字符串和%esi所储存字符串，查看该地址，得

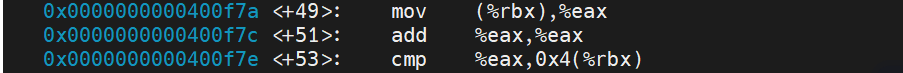
屏幕截图 2023-11-12 174358很明显phase\_1的答案即为 Joy shared with others are more enjoyed.

**Phase\_2:**

首先对其进行反汇编得到其汇编代码：



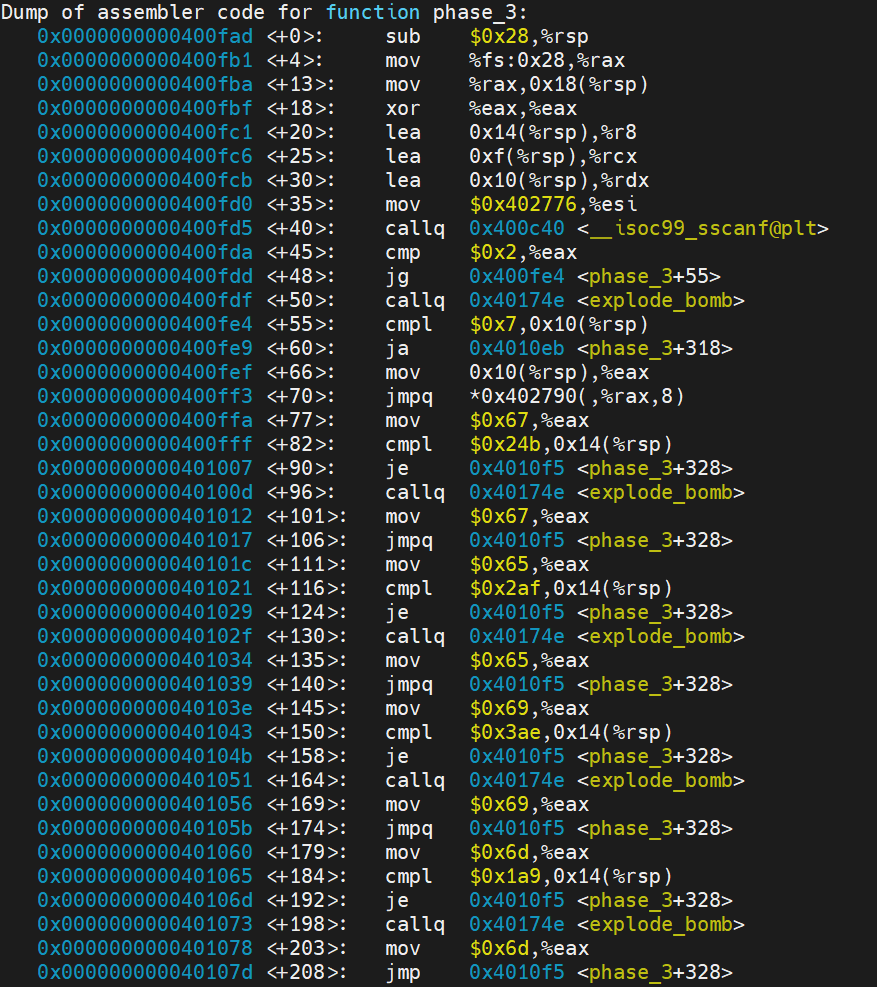
首先调用了read\_six\_numbers函数读取六个数字，即我们需要输入的数据为六个数并将其储存在%rsp

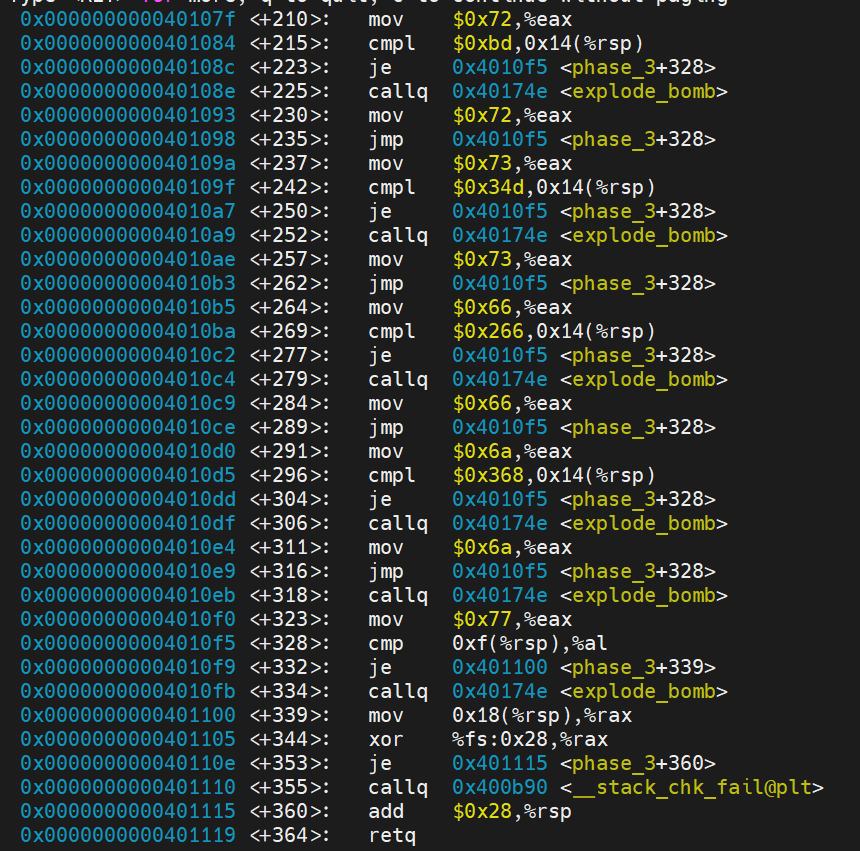
中，本炸弹的重点语句为以下三句：

这三句的意思为将读取的数字\*2并与下一个数字进行比较，若不等，则炸弹爆炸，若相等，则对读取 的下一个数字进行上面三句语句的判断，因此，我们输入的六个数字应该符合后面一个数字是前面一 个数字的两倍关系，即为以2为公比的等比数列，在read\_six\_numbers后有一比较语句，将%rsp中第 一个元素和1进行对比，若不等则炸弹爆炸，因此，我们可以知道第一个数字为1，所以正确答案应为： 1 2 4 8 16 32。

**Phase\_3:**

同样，首先进行反汇编获得汇编代码：





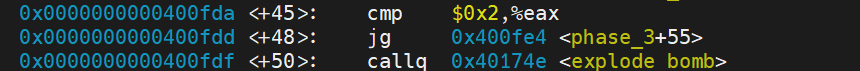
**第一阶段：**

屏幕截图 2023-11-12 192823在汇编代码的开头，可以看到

对这个地址进行读取查看

屏幕截图 2023-11-12 192815

可以看到本炸弹需要输入三个数据，%d %c %d，后依此进行判断。



此处也可以判断，若输入数据数量小于等于2的话不会跳转，则会执行炸弹爆炸部分。

**第二阶段：**

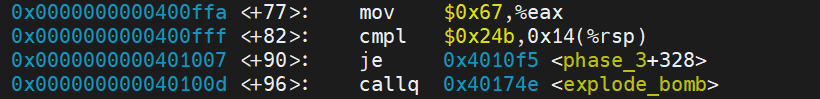
屏幕截图 2023-11-12 202302

此为将第一个输入的数字与7进行比较，若大于7则跳转至炸弹爆炸处，因此第一个输入的数据应该 小于7

屏幕截图 2023-11-12 204937

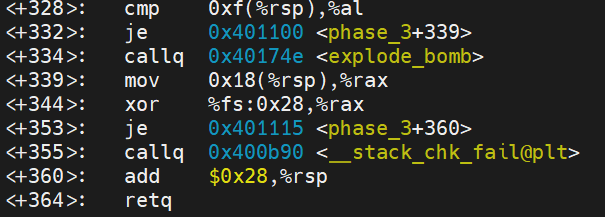
将读入第一个数传入%eax中，以%eax\*8为偏移量进行跳转，例如假设输入了0，则跳转到0x402790对 应地址，进行查看

屏幕截图 2023-11-12 205217跳转到0x400ffa对应地址：



将 0x67传给%eax，然后将0x14(%rsp)即输入第三个数据与0x24b进行比较，若相等则跳转到<+328>， 否则爆炸，这里要注意一点：由于是%d %c %d，整型4字节，字符1字节，所以入栈时不是按照输入顺 序进入栈的，而是把%c放到最上层

可以看到，<+328>处偏移量为0xf，之后便是0x10即输入第一个数地址，相差刚好一字节。

所以对输入0，字符对应ASCILL码低八位为%eax值即0x67，所以字符为g,第三个输入为整型，与0x24b 相等，所以为587

因此答案为 0 g 587

同理，所有答案为

1 e 687

2 i 942

3 m 425

4 r 189

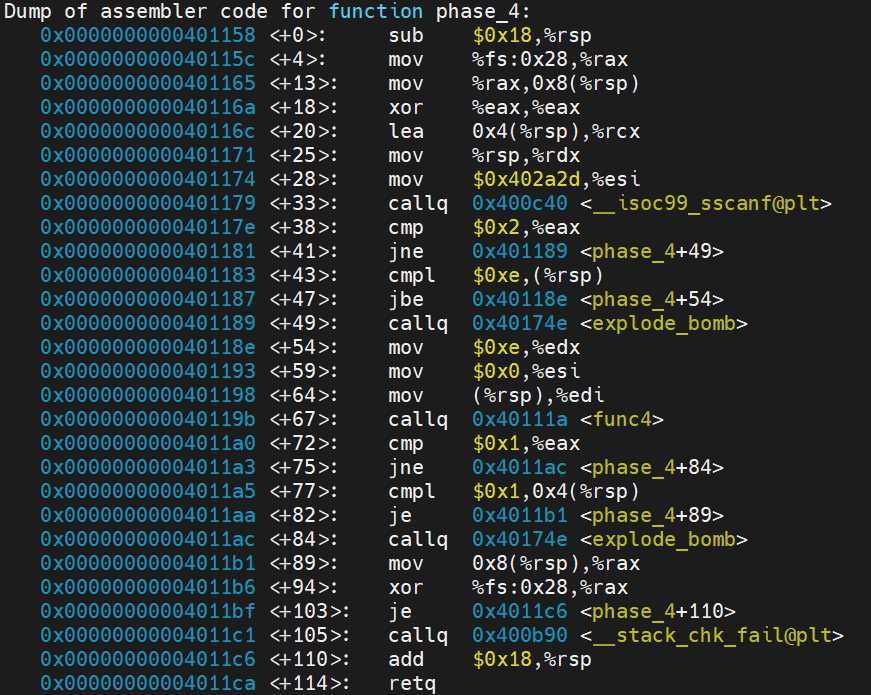
5 s 845

6 f 614

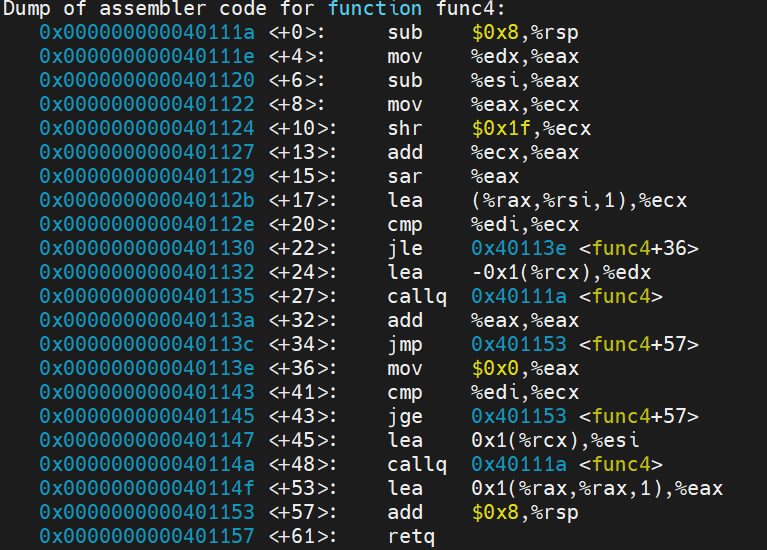
7 j 872

**Phase\_4:**

汇编代码为：

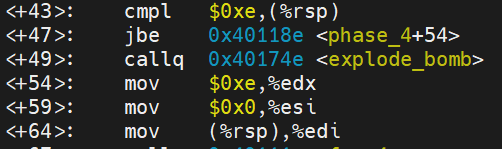


可以看见，<+67>调用了func4函数，其汇编代码如下：

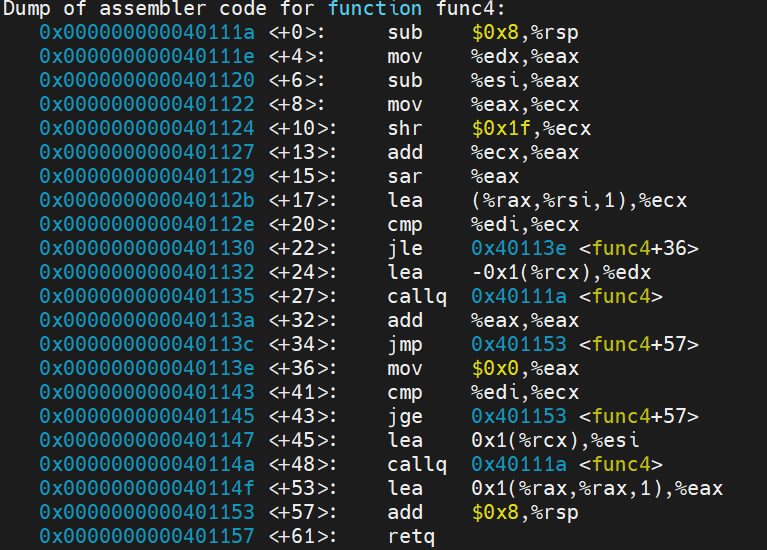


屏幕截图 2023-11-12 234934对<+28>出现地址0x402a2d，对其进行查看得

因此本炸弹的输入为两个整型。

由<+43>可以看出输入第一个数据小于14，然后将0xe传入%edx，将0传入%esi。

接下来进入func4:



可以看出这是一个递归程序，返回值在%eax中，在跳出func4后的<+72>有一判断条件，可以看出%eax 最终返回值需为1，不然炸弹爆炸，返回去代入func4中，在func4中，有<+34>、<+43>以及最终返 回三个返回语句，但是能让%eax为1的只有<+53>，所以最外层函数返回时需要过<+53>，即不能满 足前面的跳出条件。接下来看最外层递归：

<+4>: %eax = 0xe;

<+6>: %eax = %eax - %esi = 0xe

<+8>: %ecx = 0xe

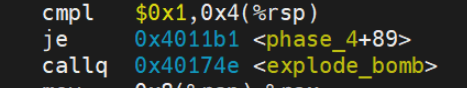
<+10>:%ecx = %ecx >> 31 = 0

<+13>:%eax = %eax + %ecx = %eax = 0xe

<+15>:%eax = %eax > 1 = 7

<+17>:%ecx = 7

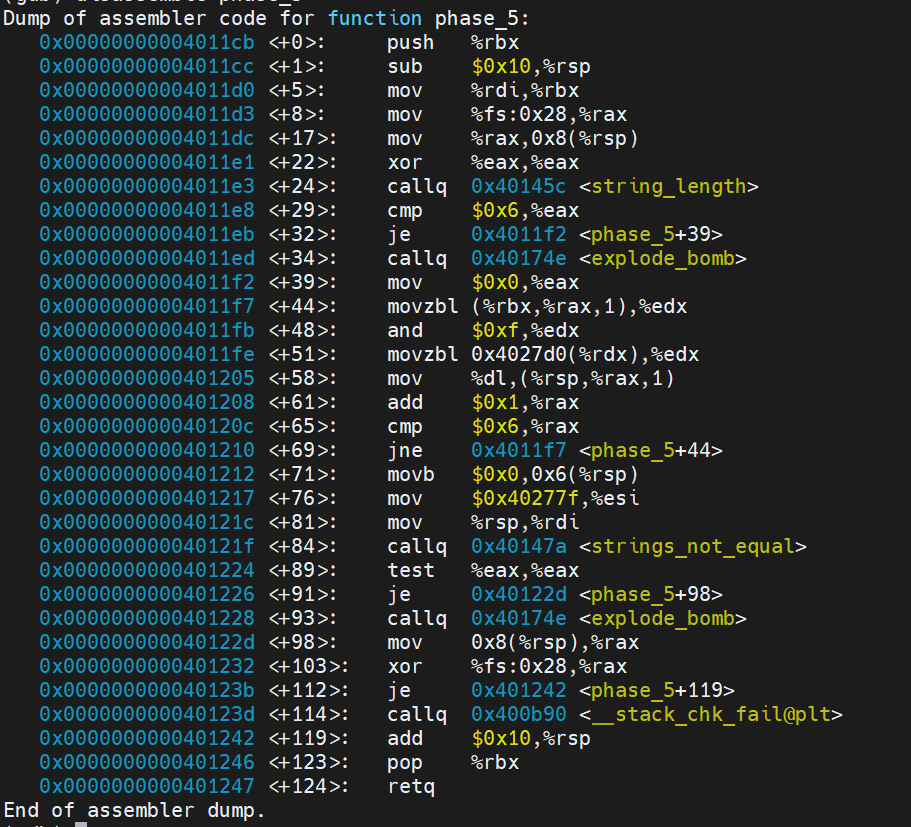
后续为根据传入数据的大小，若大于7，则往大递归遍历，若小于7，则往下递归遍历，当%edi = 7时，发现其返回值为0，不为1，要为1则需进入<+53>，当%edi = 8时，会执行<+36>到<+48>，这样当在深一层的递归中返回值为0时，最终返回值为1，通过。



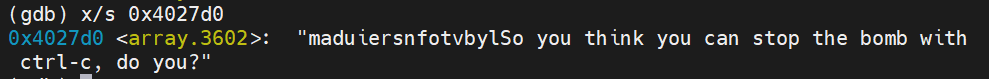
跳出func4后，由这三句语句可以看出第二个数字为1，所以答案为：8 1

**Phase\_5:**

**阶段一：**查看汇编代码

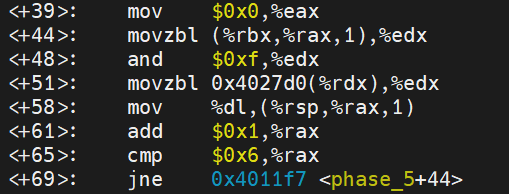


看到两个地址，对它们进行查看

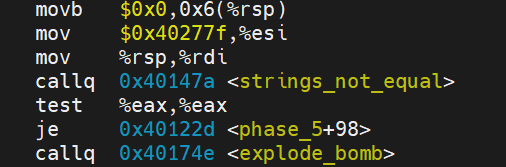


屏幕截图 2023-11-13 103432

**阶段二：**<+24>调用了string\_length函数统计输入字符串长度，<+29>将长度和6比较，若不等则炸 弹爆炸，所以输入字符长度应该为6。



这段汇编代码的逻辑为对输入的六个字符进行遍历，将他们的低四位和0xf进行与操作，以与操作结 果为偏移量对0x4027d0所存数据进行读取



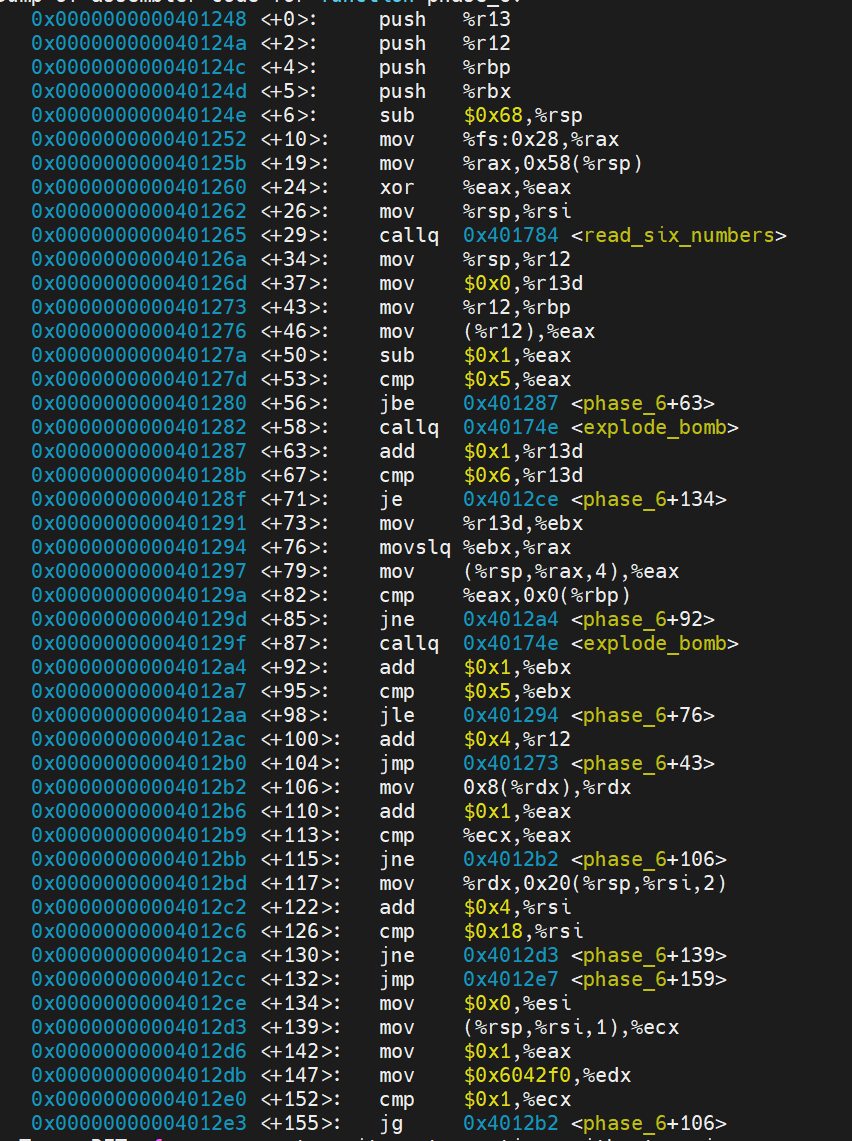
此处为将读取数据和0x40277f所存数据进行比较，若不等则炸弹爆炸，而0x40277f所存字符串为 oilers，在0x4027d0中位次分别为 10 4 15 5 6 7

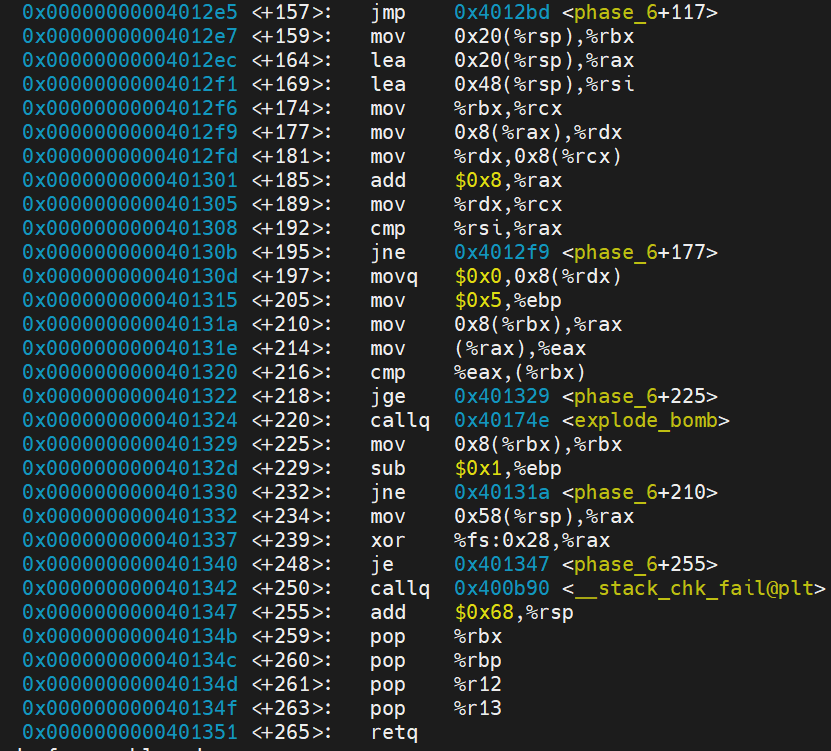
对应低四位为1010 0100 1111 0101 0110 0111

所以答案为低四位对应字符，这里可为jdoefg

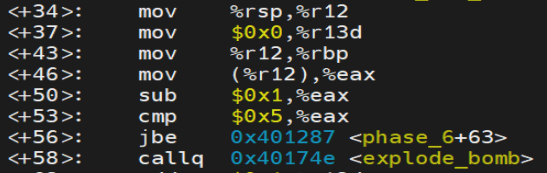
**Phase\_6:**

**阶段一：**汇编代码

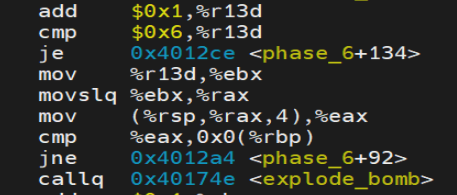




由read\_six\_numbers可知，本题需要输入6个整型

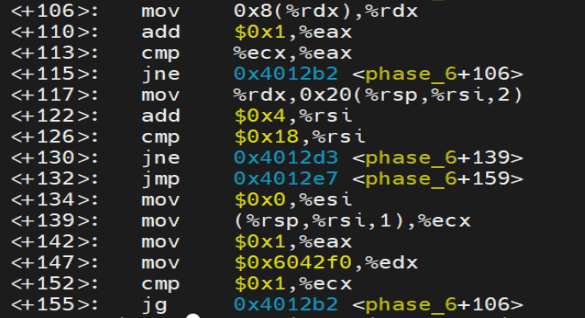


这段汇编代码意为将所有输入减一和5比较，若大于则炸弹爆炸，所以输入为6个小于等于6的数，

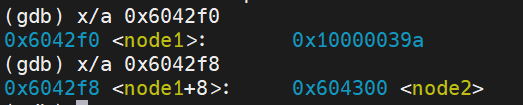


接着的这段代码则是通过循环将先输入的数字与后输入的数字比较，若不等则跳转，否则炸弹爆炸， 所以输入的六个数字应该为1 2 3 4 5 6按照一定的顺序进行排列。

**阶段二：**



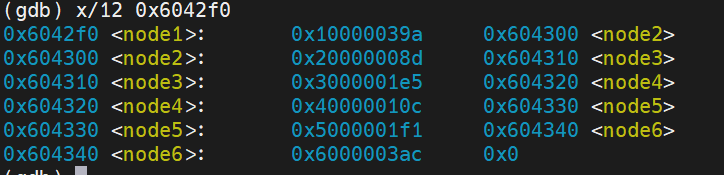
在<+147>中发现地址0x6042f0，并且跳转到<+106>时，%edx每次偏移8位，现查看该地址



可以看见储存了一个数据，地址偏移八位储存了一个新地址且命名为node2

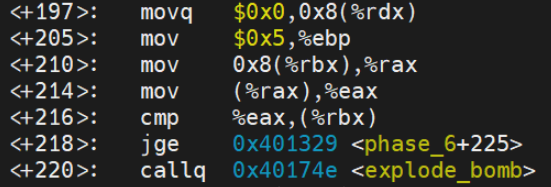
但是我们输入是整型常量，所以数据占八字节是储存放置自动对齐导致的，因此我们只需取最后四个 字节即可，为0x039a

读取后面的地址



很明显，这是一个链表，储存数据为 0x039a 0x008d 0x01e5 0x010c 0x01f1 0x03ac

**阶段三：**



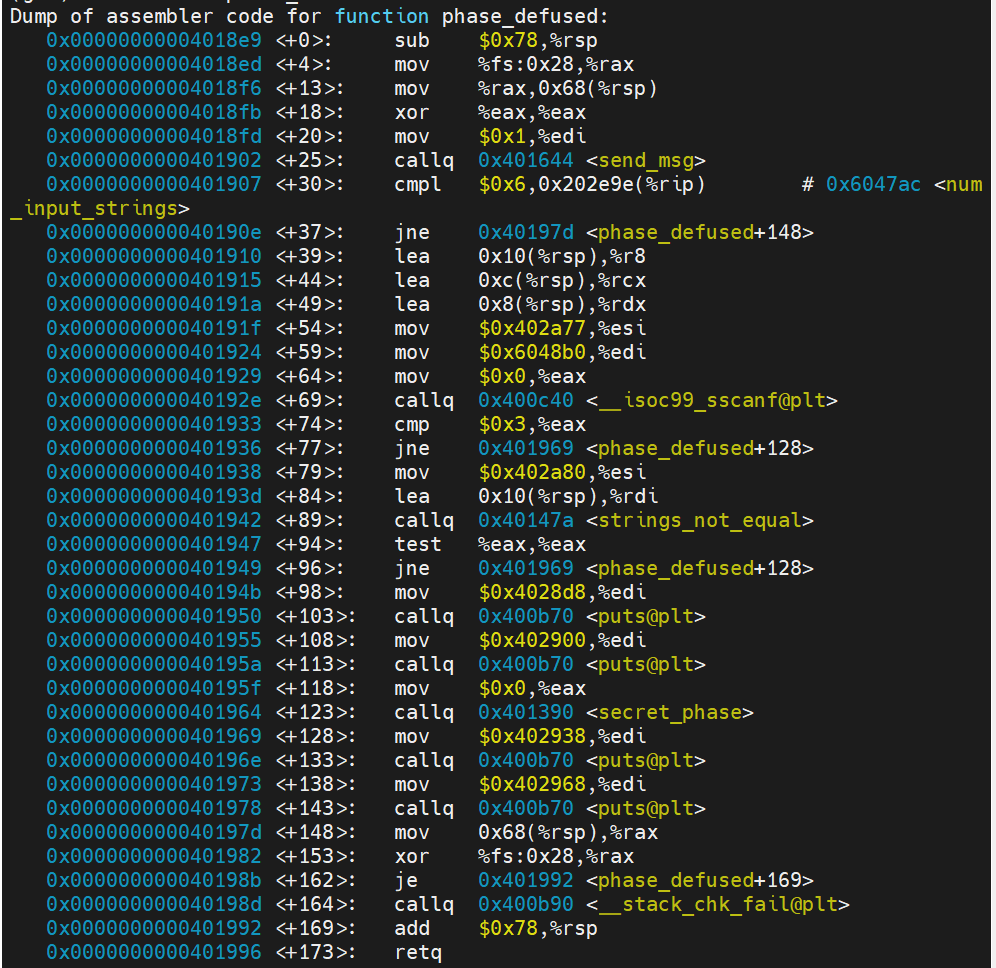
这段代码意思为将前一个输入所对应的链表数据和后一个输入的进行对比，若小于则爆炸，所以这是 一个降序排列，而0x03ac(node6) > 0x039a(node1) > 0x01f1(node5) > 0x01e5(node3) > 0x010c(node4) > 0x008d(node2)

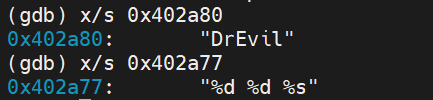
所以第六个炸弹输入为 6 1 5 3 4 2

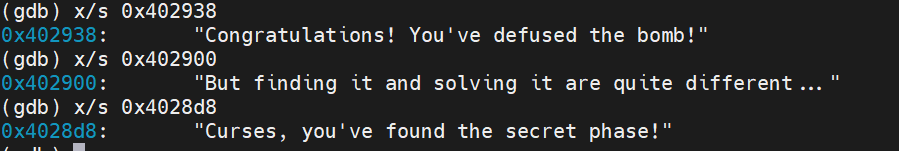
**Phase\_7:**

**准备阶段：**当我们拆完第六个炸弹时，没有出现隐藏炸弹，但是当我们查看题目所给的C语言程序时， 发现有一个secret\_phase函数，因此需要我们自己去寻找这个隐藏炸弹的入口。

线对各函数进行查看汇编代码，发现在phase\_defused函数中出现了secret\_phase





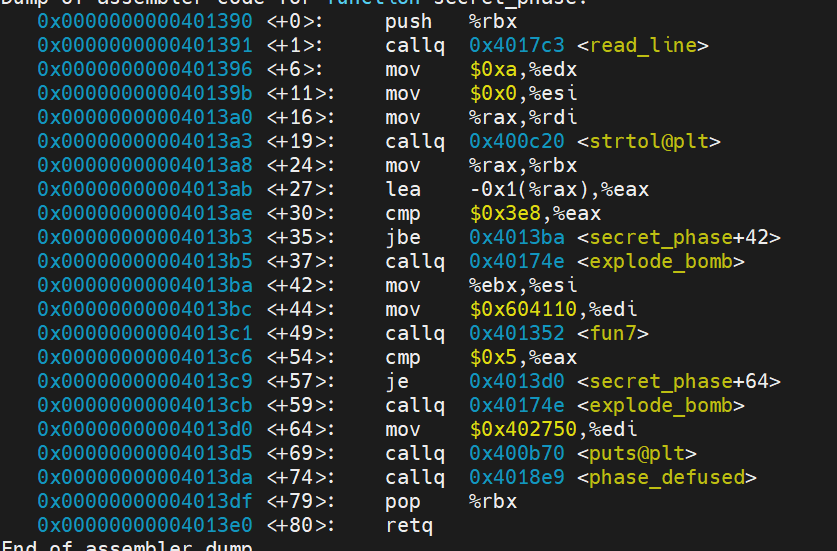


由上面储存的话可以看出隐藏炸弹触发条件是输入%d %d %s且%s为DrEvil

而前面的炸弹中只有炸弹四需要输入两个整型数据，因此进入隐藏炸弹的条件即为输入炸弹四答案后 输入DrEvil

**阶段一：**

开始查看隐藏炸弹汇编代码

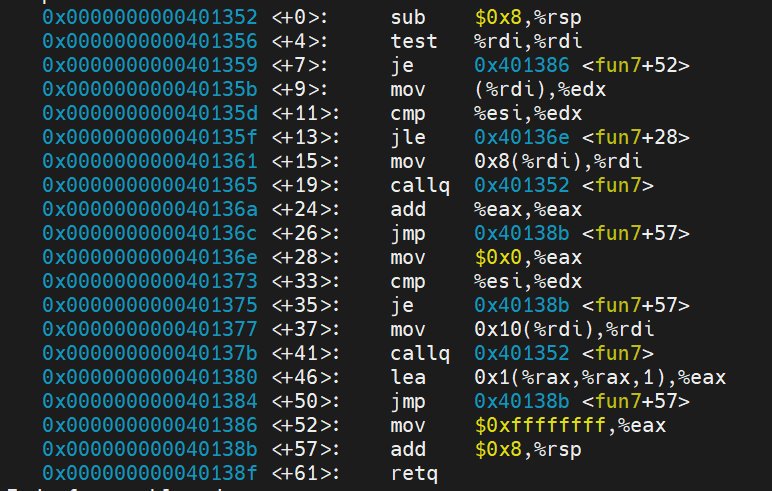


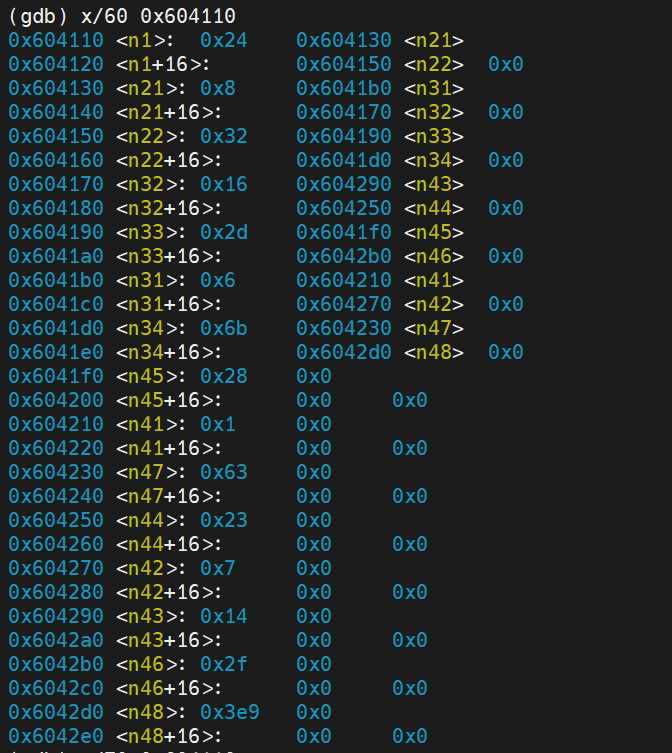
由<+27><+30>得%eax小于等于1001

之后进入fun7函数

**阶段二：**

fun7函数汇编代码为

  
 很明显fun7函数是一个递归函数，由<+15>得%rdi地址偏移量为8\*n，查看其地址得



可以看出这是一个二叉树，具体结构如下

0x24

0x8 0x32

0x6 0x16 0x2d 0x6b

0x1 0x7 0x14 0x23 0x28 0x2f 0x63 0x3e9

**阶段三：**

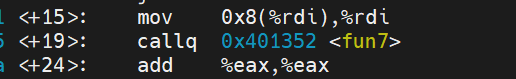
屏幕截图 2023-11-13 203942

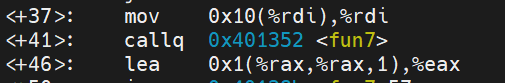
知fun7的最终返回值为5，再看二叉树的查找

屏幕截图 2023-11-13 204355

屏幕截图 2023-11-13 204403

很明显，偏移量为8时往左查找，为16时，往右查找，

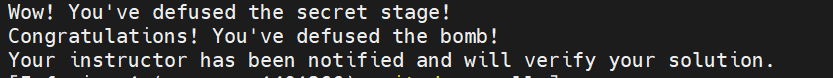




由此可以看出左查对应%eax\*2右查对应%eax\*2+1

而返回值需要是5，所以为向右一次，向左一次，再向右一次，最终返回值才会是5

所以对应为0x2f即47

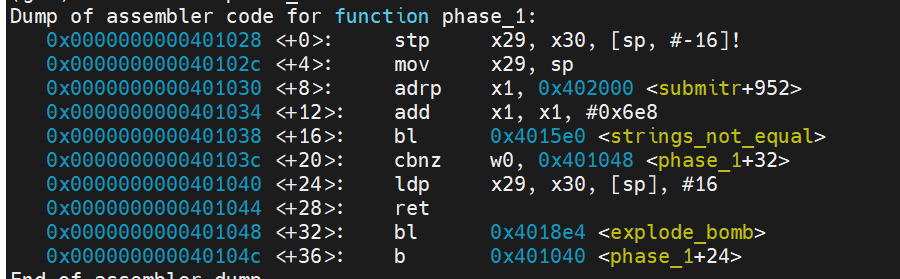


炸弹全部拆除完毕。

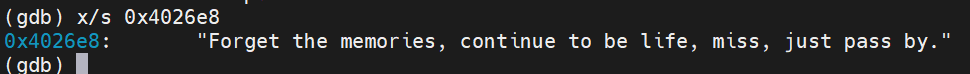
### 高阶lab2

**准备工作**:了解arm架构下汇编代码的相关语法，以及寄存器的名称和相关用法，由于之前已经完成了普通lba2实验，我们可以发现arm框架下的拆炸弹思路类似，只是汇编代码不同，因此部分细节不多做解释。

**Phase1:**



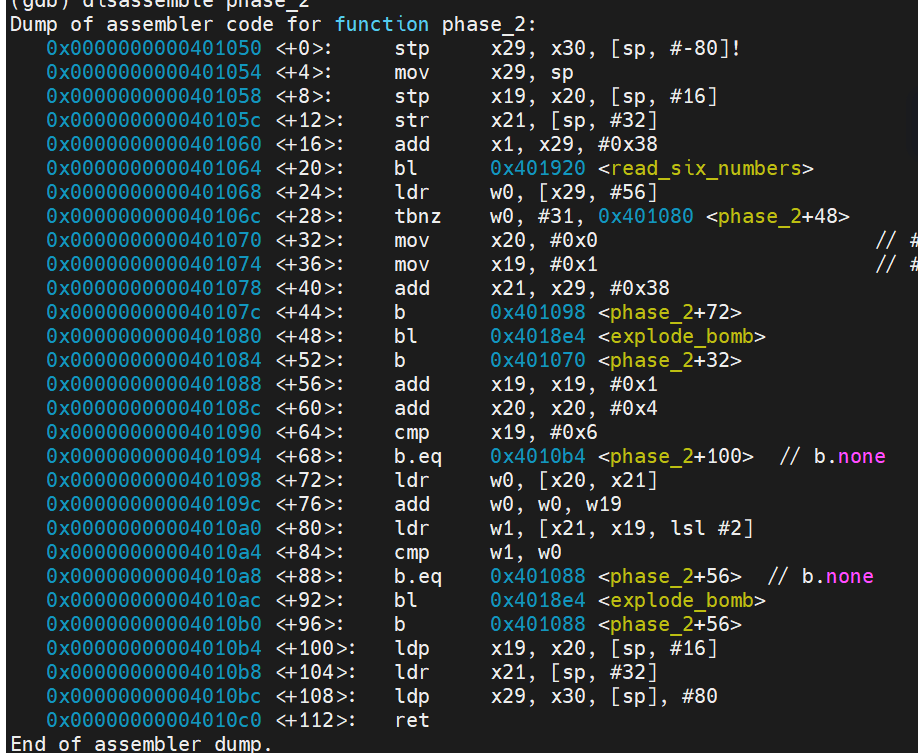
<+8><+12>操作为将0x402000地址赋给x1寄存器，然后再加上0x6e8的地址偏移量，为0x4026e8，我们来查看该地址所储存部分：



所以phase\_1答案为：Forget the memories, continue to be life, miss, just pass by.

**Phase2:**

首先来查看phase\_2的汇编代码



由read\_six\_numbers可以看出第二个炸弹也是需要输入六个数字，<+28>可以看出要求输入的第一个数据第31位为0，之后是一个循环，每次将我们输入的数组a里面的数据进行比较，<+72-+76>代表w0 = a[m] + i

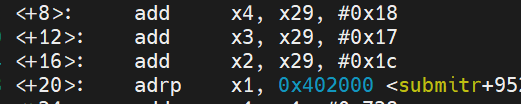
w1 = a[m + 1]，所以要求为a[m] + i = a[m + 1]

所以答案为： 1 2 4 7 11 16 22

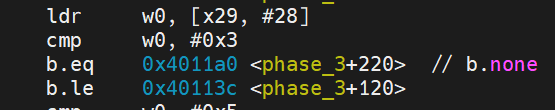
也可以为 0 1 3 6 10 15 21

**Phase3:**

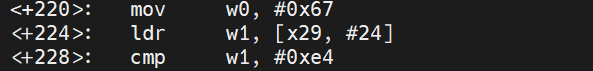
arm架构下的phase\_3与x86思路非常相似，可以有多个答案，此处就只取第一个来进行说明



首先看到这一段汇编代码，x29储存我们所输入的数据，可以发现x3储存了我们输入的第一个数据，然后是x4，再然后是x1，要注意这一点，以防止之后做题时输入答案顺序错误

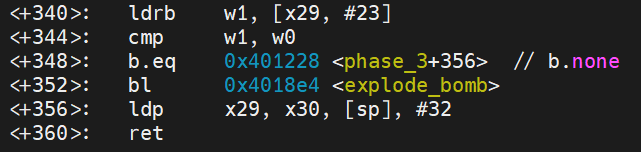


当我们输入的第一个数据为3时，跳转到<+220>：



这里给w0赋值0x64，并且将w1和0xe4进行比较，若相等则继续跳转至<+340>，所以x1值为0xe4，

这里注意，此时w1偏移量为24



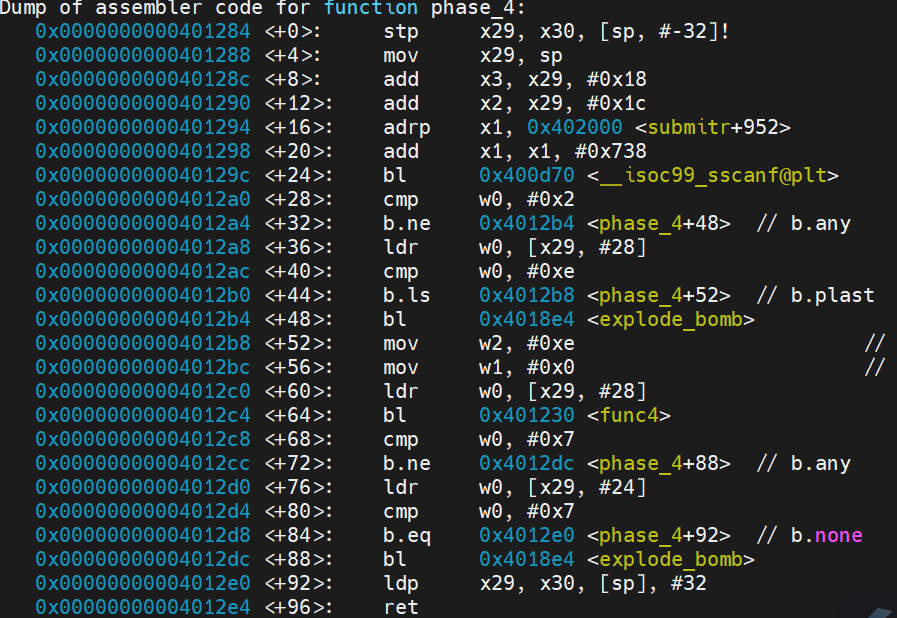
这里将x29偏移量为23地址的值储存进w1，所以这里比较的是我们输入的第二个数据，上面的才是我们输入的第三个数据，因此我们输入的第二个数据值为w0的值即为0x64

所以答案可以为 3 0x64 0xe4

由x86可知我们输入为%d %c %d 此处进行转换得 3 g 228

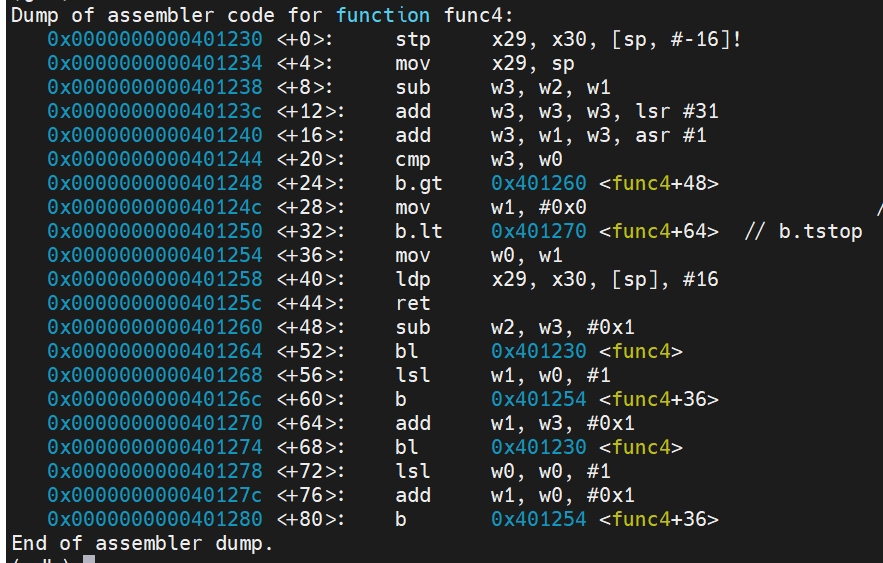
**Phase4:**

首先查看汇编代码



同x86，这里的phase\_4首先进入了func4递归函数进行比较，然后跳出时再比较返回值和第二个数据，<+68>易得我们输入的第二个数据为7

然后来查看func4



可以看到func4是一个递归函数，转换成C语言代码为

int func4(int w1, int w2,int w0)

{

    int w3 = w2 - w1;

    w3 = w3 + (w3 >> 31);

    w3 = w1 + (w3 >> 1);

    if (w3 > w0)

    {

        w2 = w3 - 1;

        w0 = func4(w1, w2,w0);

        w1 = w0 << 1;

        w0 = w1;

        return w0;

    }

    else{

        w1 = 0;

        if (w3 == w0){

            w0 = w1;

            return w0;

        }

        if (w3 < w0){

            w1 = w3 + 1;

            w0 = func4(w1, w2,w0);

            w0 = w0 << 1;

            w1 = w0 + 1;

            return w1;

        }

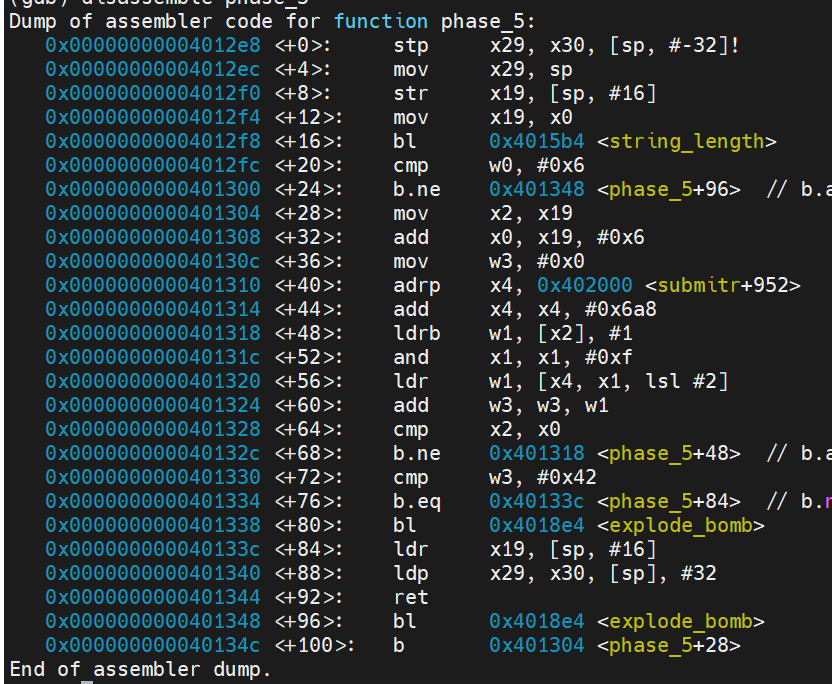
    }

}

现在其返回值要为7，w1 = 0，w2 = 14，易得w0为14，所以第四题答案为14 7

**Phase5:**

首先查看汇编代码



<+16><+20>表示第五题要求输入长度为6的字符串<+40><+44>中x4储存地址为0x4026a8，现在进行查看，得：

0x4026a8 <array.4328>: 0x02 0x00 0x00 0x00 0x0a 0x00 0x00 0x00

0x4026b0 <array.4328+8>: 0x06 0x00 0x00 0x00 0x01 0x00 0x00 0x00

0x4026b8 <array.4328+16>: 0x0c 0x00 0x00 0x00 0x10 0x00 0x00 0x00

0x4026c0 <array.4328+24>: 0x09 0x00 0x00 0x00 0x03 0x00 0x00 0x00

0x4026c8 <array.4328+32>: 0x04 0x00 0x00 0x00 0x07 0x00 0x00 0x00

0x4026d0 <array.4328+40>: 0x0e 0x00 0x00 0x00 0x05 0x00 0x00 0x00

0x4026d8 <array.4328+48>: 0x0b 0x00 0x00 0x00 0x08 0x00 0x00 0x00

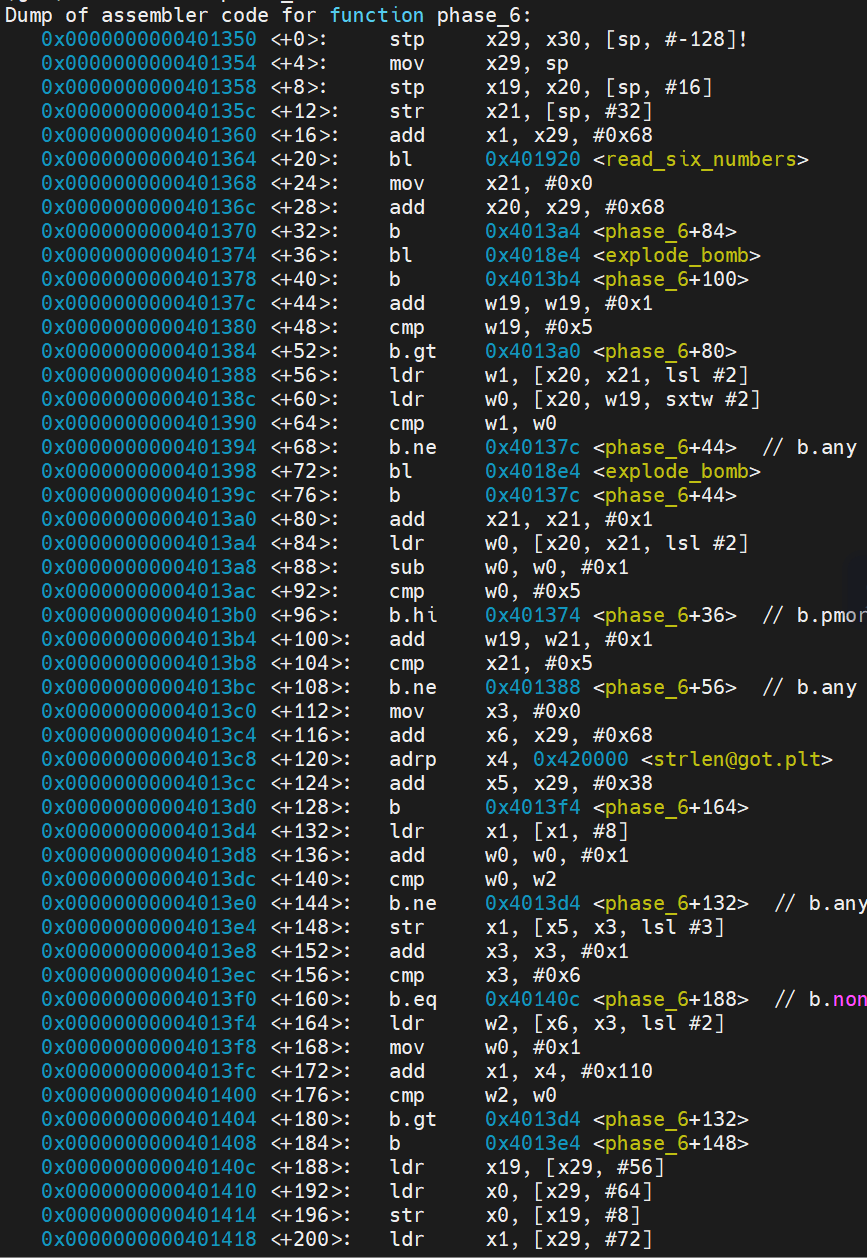
0x4026e0 <array.4328+56>: 0x0f 0x00 0x00 0x00 0x0d 0x00 0x00 0x00

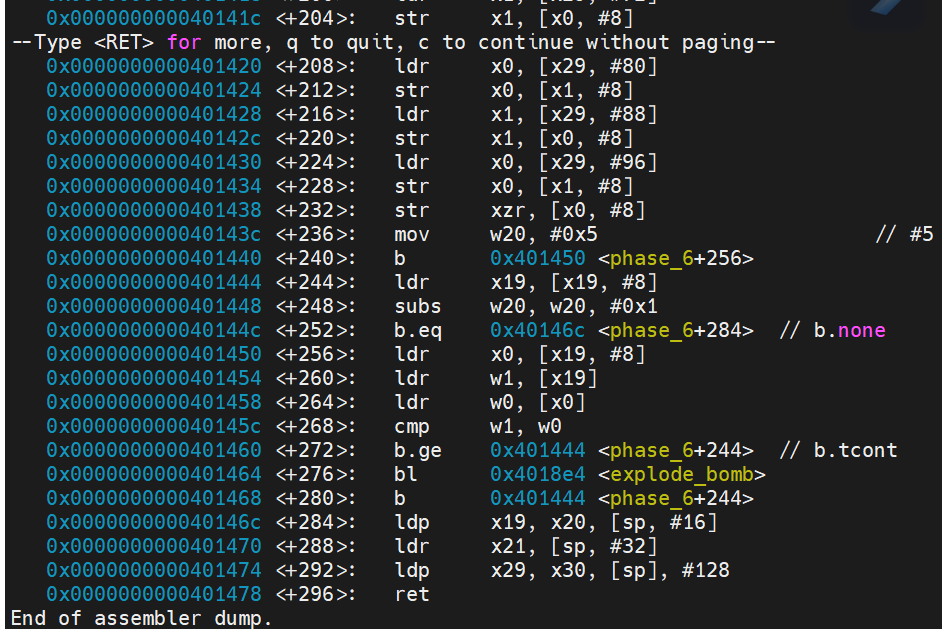
再看<+44>-<+56>，将输入的字符后四位作为偏移量将x4中的值赋给w1，然后w3 = w3 +w1，当w3 = 0x42时跳出循环，我们可以让w3 = 6 + 12 + 9 + 14 + 11 + 14 = 66

则相应偏移量为 8 16 24 40 48 40 对应二进制为 0010 0100 0110 1010 1100 1010

对应字符串为 b d f j l j为答案

**Phase6:**





第六个炸弹思路同样和x86类似，我们可以通过<+120>和<+172>找到链表

0x420110 <node1>: 0x1000003a4 0x420288 <node2> 1

(gdb) x/2 0x420288

0x420288 <node2>: 0x2000003a2 0x420298 <node3> 2

(gdb) x/2 0x420298

0x420298 <node3>: 0x3000000d0 0x4202a8 <node4> 6

(gdb) x/2 0x4202a8

0x4202a8 <node4>: 0x400000144 0x4202b8 <node5> 5

(gdb) x/2 0x4202b8

0x4202b8 <node5>: 0x500000196 0x4202c8 <node6> 4

(gdb) x/2 0x4202c8

0x4202c8 <node6>: 0x600000370 0x0 3

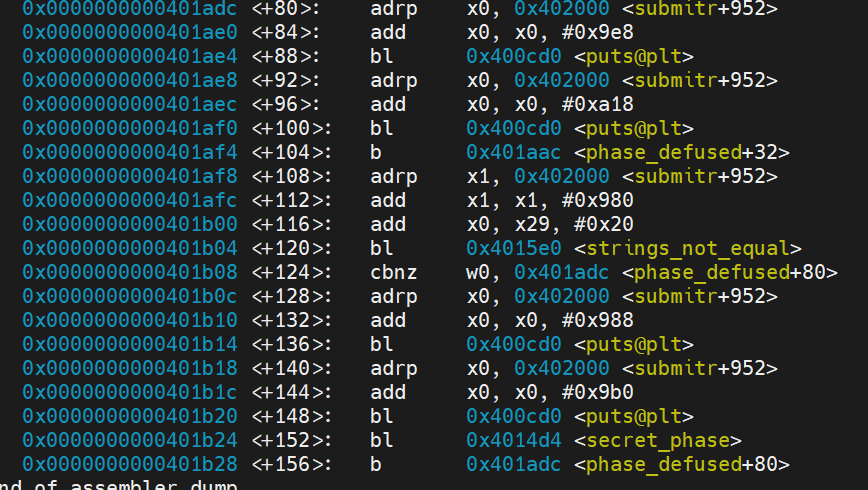
因为储存的是int类型，所以只取后四位，得0x3a4, 0x3a2, 0xd0, 0x144, 0x196, 0x370

<+272>要求是大于，我们可以判断需要将链表元素进行降序排序，由大到小的顺序为 1 2 6 5 4 3

所以答案为1 2 6 5 4 3

**Phase\_7:**

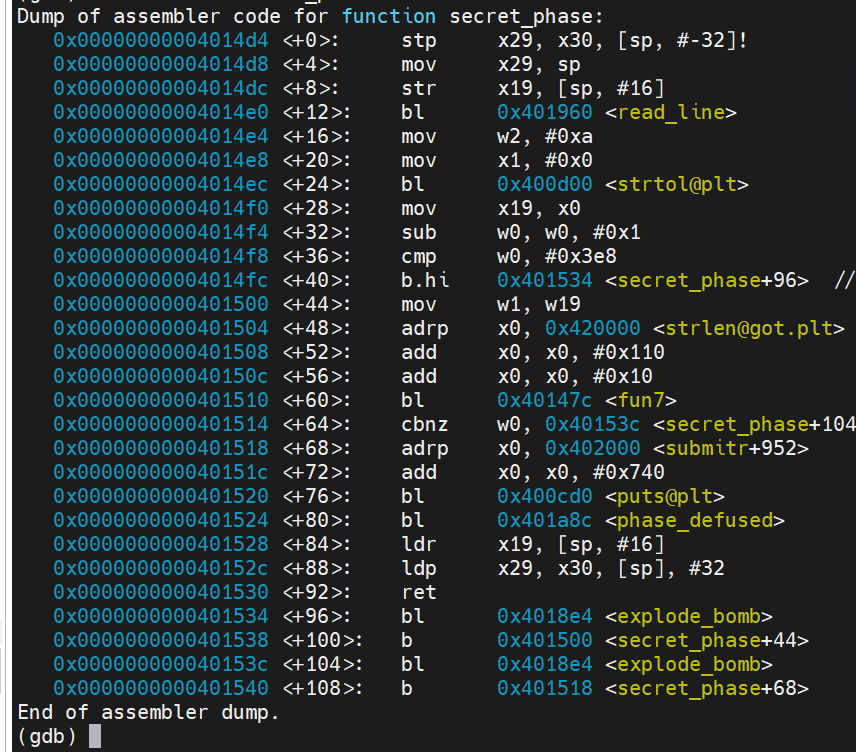
首先需要进入炸弹，虽然x86我们已知进入方式，但在这里还是先检验一下，



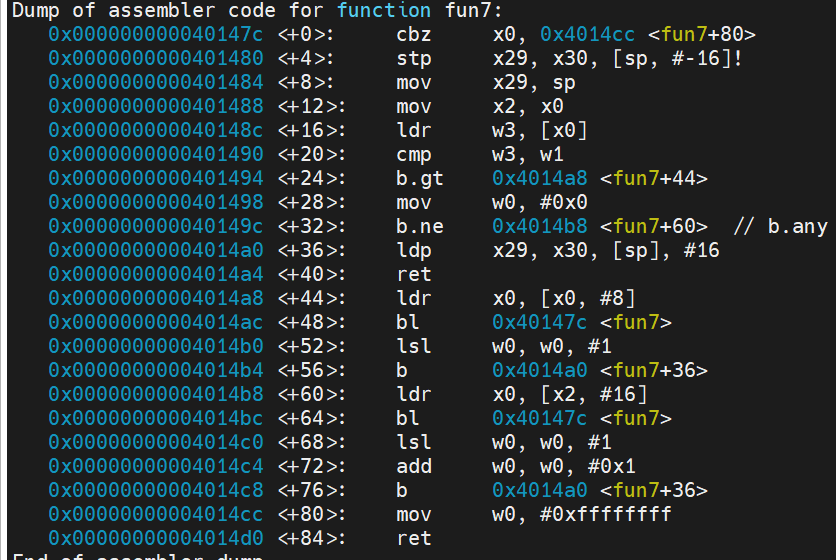
<+112>有个地址，进行查看



所以进入秘密炸弹方式不变，然后查看秘密炸弹汇编代码



查看fun7



通过查看0x420110我们发现这是一个二叉树，结构如下

0x24

0x8 0x32

0x6 0x16 0x2d 0x6b

0x1 0x7 0x14 0x23 0x28 0x2f 0x63 0x3e9

由fun7代码得出向左查找w0 = w0\*2 向右则为w0 = w0\*2 + 1，由秘密炸弹<+64>得出返回值必须为0，所以可以进行的操作为向左查找n次，n >= 0

答案可为 0x24, 0x8, 0x6, 0x1

由此，拆弹结束

五、总结体会

本次实验总计花费了一周时间来完成，其中炸弹一炸弹二共计花费1小时，炸弹三花费2小时，炸弹四花费1.5小时，炸弹五炸弹六花费4小时，隐藏炸弹花费0.5小时，本人在本次实验中投入了较大精力，查询了大量汇编指令的含义，过关时常常感觉松了一口气，而拆弹失败时常常感到疑惑是哪里理解错误，由于汇编代码行数较多，为防止忘记相关内容，本人常常为理解整段代码并做好笔记而拆炸弹至凌晨一点，但拆弹成功后的成就感也让人着迷。

在本次实验中本人遇到了许多问题，比如x打印出地址所储存的数据或者字符串时常常会因为设置参数错误而导致打印出的内容不符合预期，后来每次打印都会尝试多个参数以防万一。

拆除第四个炸弹时，对跳转判定条件不熟悉，加上递归的复杂性，导致我晕头转向，在看完第四个炸弹的汇编代码后发现其第一次递归时直接跳出条件为输入为7，完全没有注意若直接输入7的话返回值不符合炸弹四函数内的判定条件，导致拆弹一直失败，后面一步一步进行模拟发现需要先递归一次使%eax值为0才可。

第五个炸弹刚开始时数字符位置时从1开始数，导致答案错误，后发现失误从0开始数起，纠正了该错误。

拆除第六个炸弹时，开始只输出了地址0x6042f0中内容，没有发现是一个链表，在后面看到地址偏移量进行多次打印后面的地址时才发现后面仍然还有内容，打印完成后，输出为0x10000039a，忘记了int类型占四个字节，采取截断方式，导致排序错误。

拆除隐藏炸弹时，不确定如何进入隐藏炸弹，因此对所有模块的汇编代码进行了搜索，最终发现在每个炸弹都会调用的phase\_defused函数中存在隐藏炸弹的入口。

拆除arm炸弹时，不熟悉arm框架下寄存器相关知识，在拆第三个炸弹时把输入数据的后两个顺序写反，导致长时间错误，后发现偏移量不是按照一般的从小到大或者从大到小给出的，由此发现问题并改正。s

建议：在本次实验前，希望课程能够先多布置与汇编代码相关的作业，以此增加同学对汇编代码的熟悉，方便拆弹。

六、诚信声明（不签扣10分）

需要填写如下声明，并在底部给出手写签名的电子版。

在完成本次实验过程中，我曾分别与以下各位同学就以下方面做过交流：

1. 来自唐浦瑞同学的建议，采用x/数字的方法，一次性查看链表所存地址和数据。
2. 来自高天涛同学的看法，关于对齐和int型占4字节等的提醒

此外，我还参考了以下资料：

1. bing.com
2. chat.openai.com通过询问大语言模型来了解汇编指令的跳转条件等

在我提交的程序中，还在对应的位置以注释形式记录了具体的参考内容。

我独立完成了本次实验除以上方面之外的所有工作，包括分析、设计、编码、调试与测试。

我清楚地知道，从以上方面获得的信息在一定程度上降低了实验的难度，可能影响起评分。

我从未使用他人代码，不管是原封不动地复制，还是经过某些等价转换。

我未曾也不会向同一课程（包括此后各届）的同学复制或公开我这份程序的代码，我有义务妥善保管好它们。

我编写这个程序无意于破坏或妨碍任何计算机系统的正常运行。

我清楚地知道，以上情况均为本课程纪律所禁止，若违反，对应的实验成绩将按照0分计。



（签名）