

**实验报告**



**题目： 拆解二进制炸弹（高阶）**

**班 级： 2023211307**

**学 号： 2023211088**

**姓 名： 罗浩**

**学 院： 计算机学院**

**202 年 11 月 10 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. Windows PowerShell（10.120.11.12）
3. Linux
4. Objdump命令反汇编
5. GDB调试工具
6. 。。。。。

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。

四、实验步骤及实验分析

建议按照：准备工作、阶段1、阶段2、…等来组织内容

各阶段需要有操作步骤、运行截图、分析过程的内容

准备过程：

认识常用指令：



stp x29, x30, [sp, #-16]!——用于入栈，可以将两个寄存器x29x30同时入栈

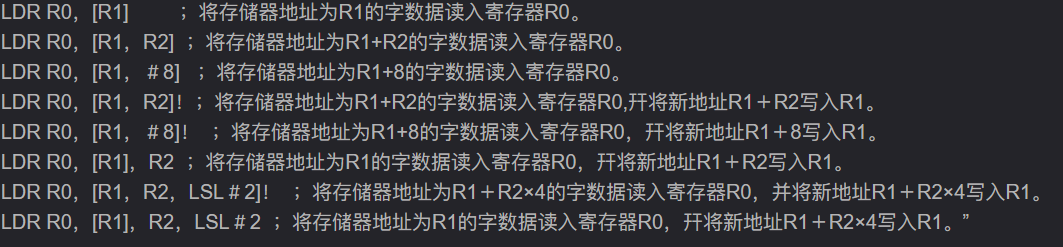


bl 0x401568 <strings\_not\_equal>bl指令用于分支跳转，似乎专门用于函数调用的指令跳转。

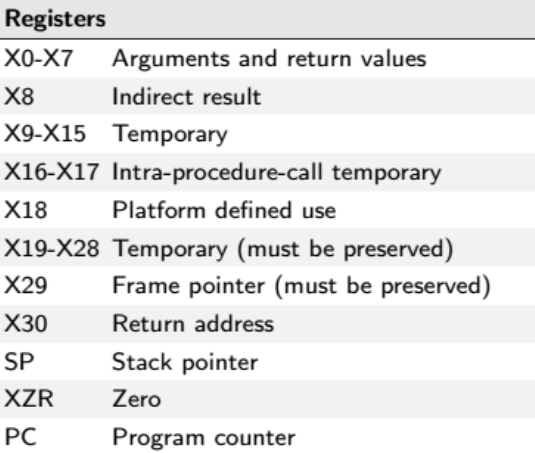


Ldr指令 LDR{条件}  目的[寄存器](https://so.csdn.net/so/search?q=%E5%AF%84%E5%AD%98%E5%99%A8&spm=1001.2101.3001.7020)，<存储器地址>，从存储器中将一个32位的字数据传送到目的寄存器中。该指令通常用于从存储器中读取32位的字数据到通用寄存器，然后对数据进行处理

取值规则：



寄存器功能：



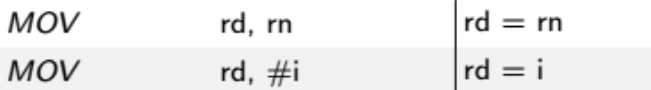
 **32 位寄存器（w0 到 w30）**

 **16 位寄存器（h0 到 h30）**

 **8 位寄存器（b0 到 b30）**

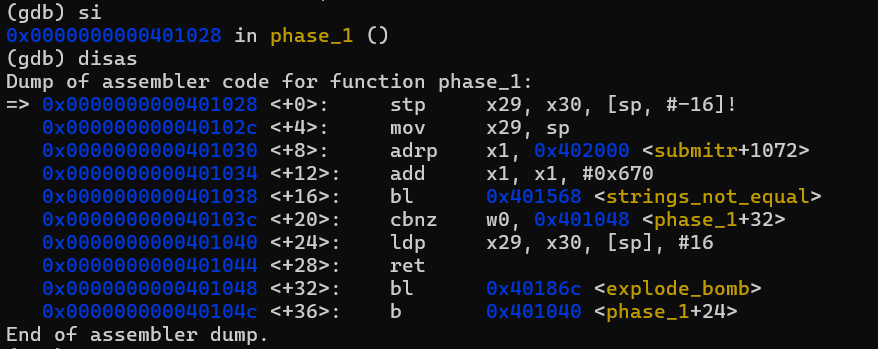
TBNZ 指令是 "Test Bit and Branch if Not Zero" 的缩写，通常用于条件跳转。它的功能是对寄存器中的特定位进行测试，如果该位的值不为零，则执行跳转。TBNZ <Xt>, #<bit>, <label>

Mov指令：



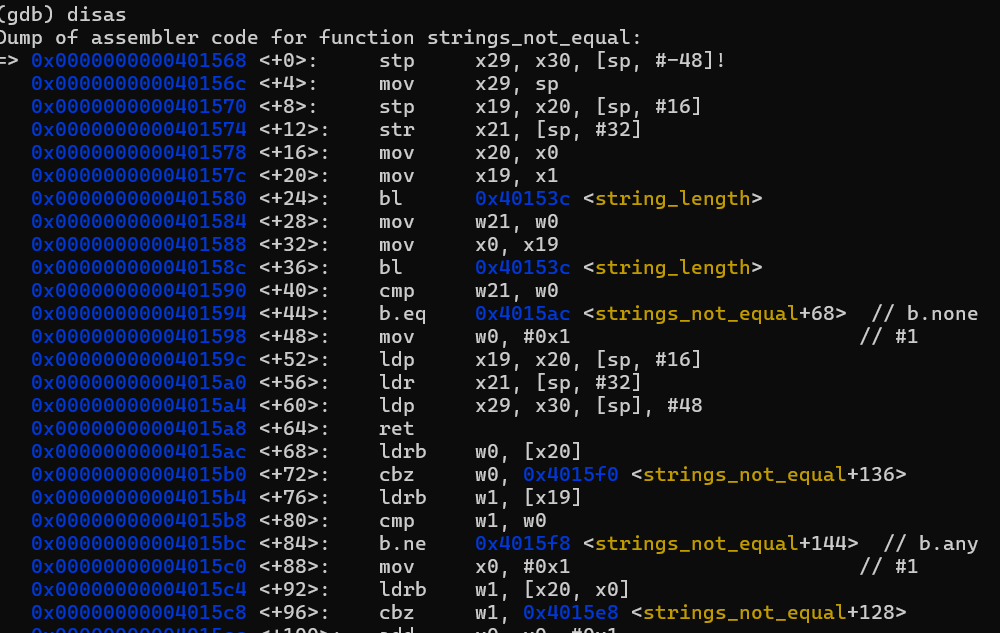
***Phase1：***

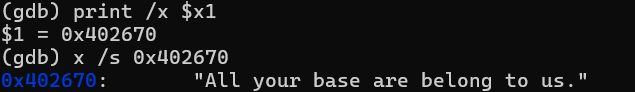
1.根据之前的lab2实验做法，我们使用gdb一路来到第一关



单步进入函数strings\_not\_equal

2.来到strings\_not\_equal函数内部，反汇编观察一下，打印出寄存器中的值





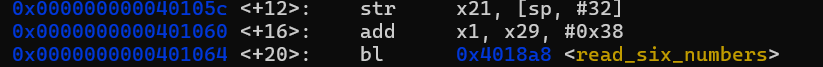
至此，我们找到第一关的答案——All your base are belong to us.

***Phase2：***

1.单步调试来到phase2

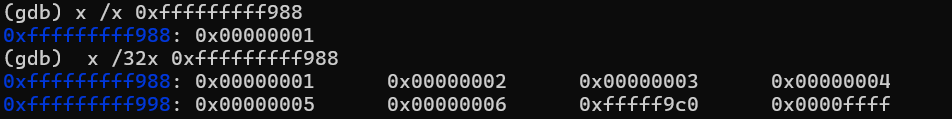
******

2.发现又要读六个数



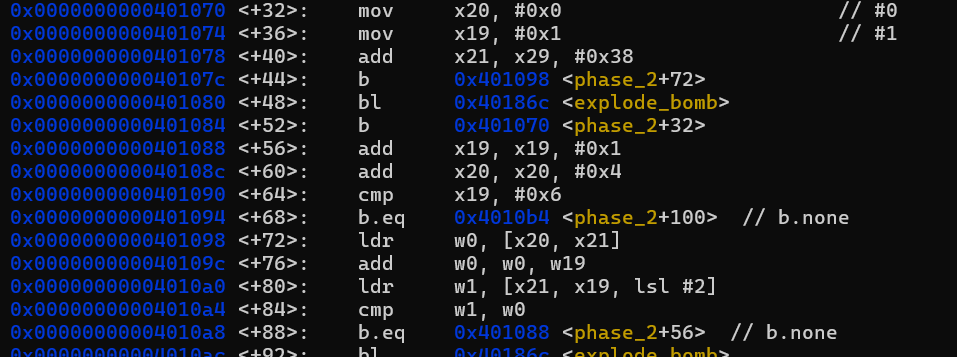
因此注意输入六个数，以防炸弹提前爆炸。

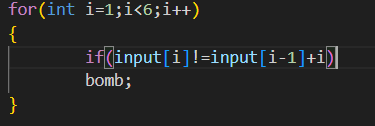
3.观察



我们发现这个地址存贮了我们的输入

4.将以下汇编代码转译为如下C源码：即可得到答案：1 2 4 7 11 16



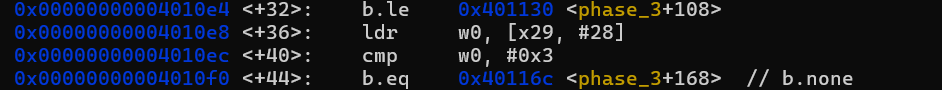


***Phase3：***

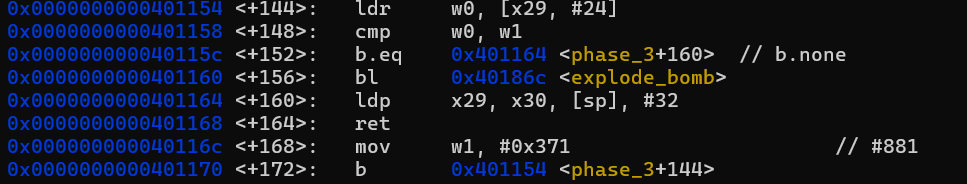
1. 函数只接受两个整数，



1. 进入后有一个条件判断可以跳转到很后面，所以我们直接满足这个条件，也即我们输入的第一个数为3

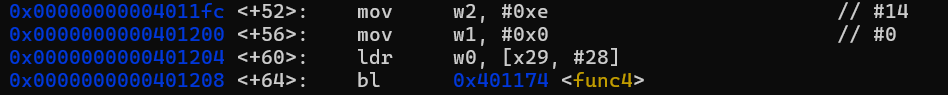


1. 跳转后我们发现又要将我们输入的第二个数与881作比较，因此使第二个数为881即可满足条件完成当前阶段的解码。

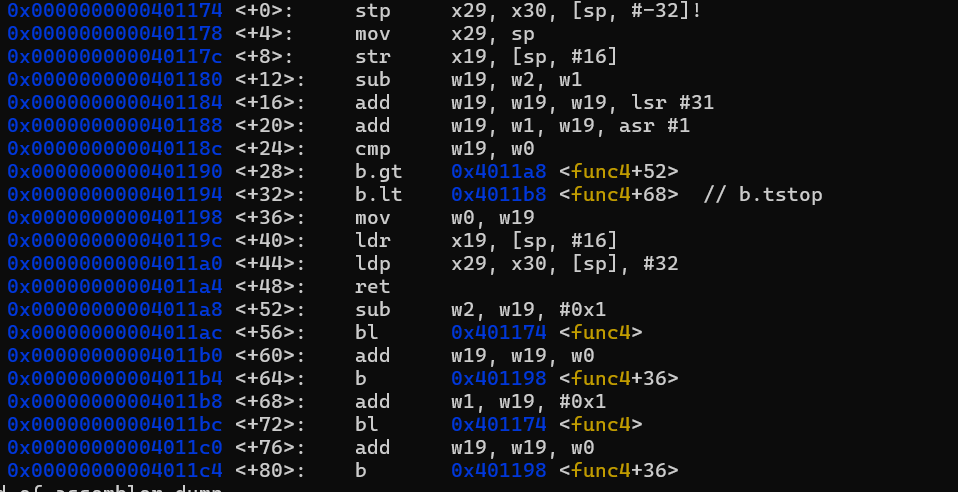


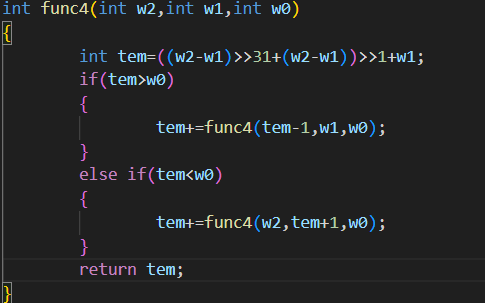
***Phase4：***

1. 在进入fun之前，先观察参数：



1. phase4的唯一难点为fun4，这是一个递归调用，因此用C源码翻译会比较直观，返回值为w0,要求返回值为0xf，也即15.



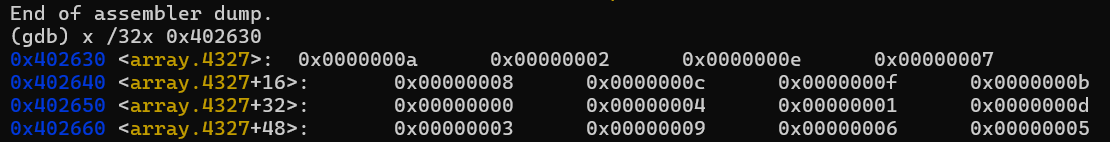


1. 会发现最少两次递归调用就能拿出答案，递归条件为w0=5；

可使得返回值为15。

***Phase5：***

1.有一个取数操作来源来自一个数组：

******

w1取出sp+28存储的值，并和0xf比较。如果相等，结束这个循环。

w2每次加一；ldr指令：将w1 signed-extended，然后左移两位，并将此作为偏移量加上x0的基地址，将得到的地址中的数据加载到w0中。

str指令：将w0的内容存储到栈指针sp+28中。

w3：累加，每次都将w0的值累加到w3中。

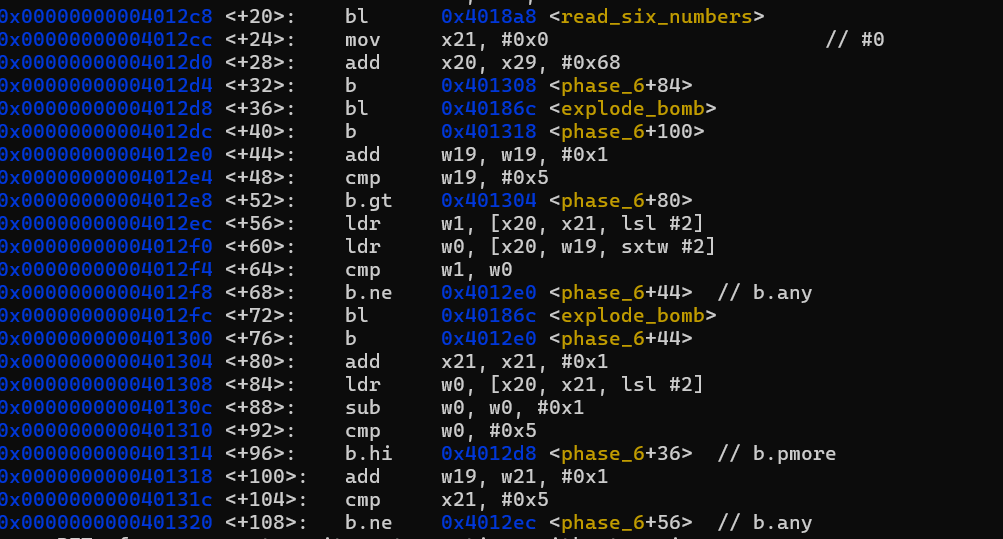
至此，我们大致可以非形式化的描述这个过程：首先，取栈指针中的值，并据此计算偏移量，存储数组中的值到栈指针中，直到取到的值为0xf。在这个循环中，w2相当于循环计数器，程序期待其最终值为：w2 = 0xf，所以要求循环次数必须为15；w3为记录每次取到数组中的值的累加和。

答案为5 115

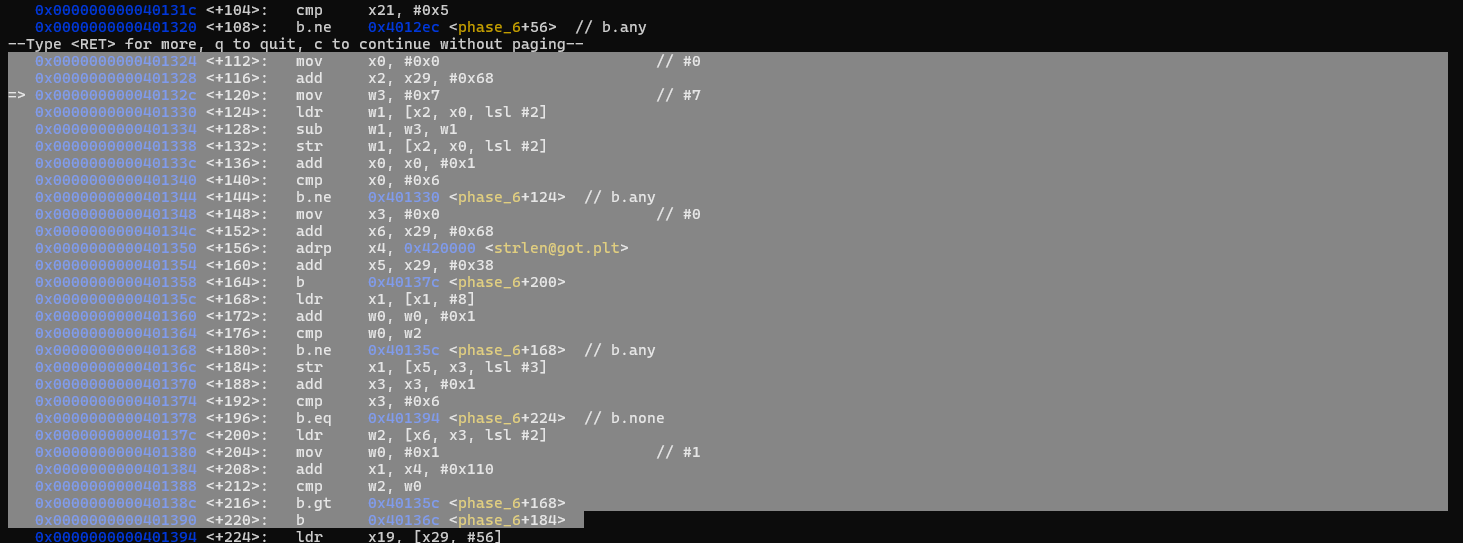
***Phase6：***

我们将第六阶段的代码分为几段来看待

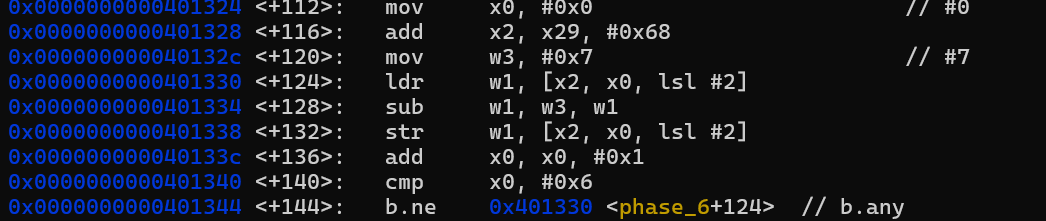
1.判断输入的六个数是否互不相等：



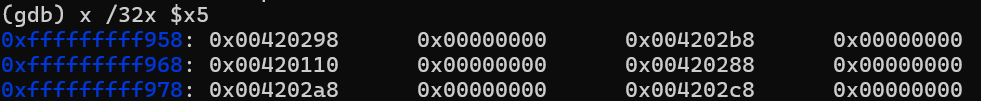
2.第二段代码（白色选中的一部分）：



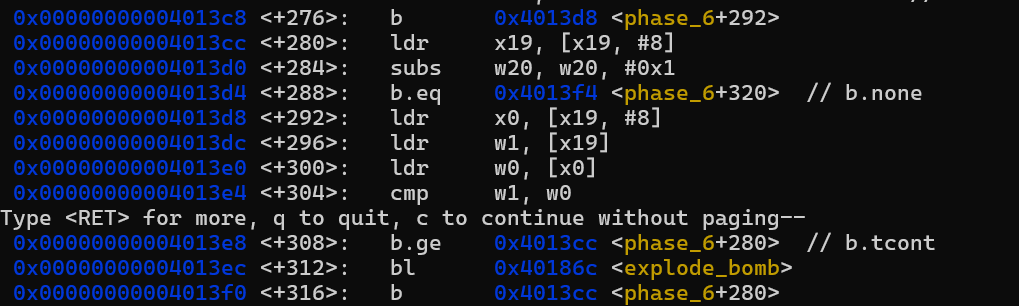
3.将输入补码化：如1 2 3 4 5 6转化为6 5 4 3 2 1



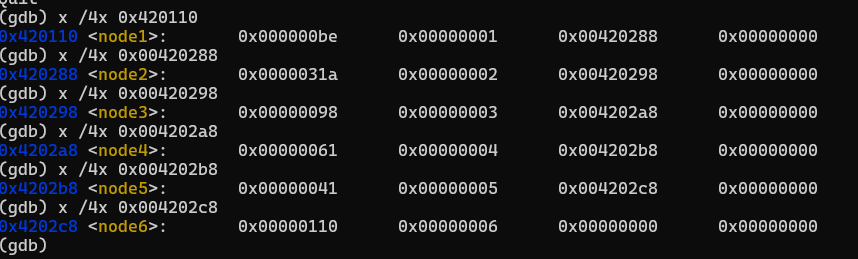
4.将指向节点的指针按照以上补码的大小进行排序得到的存储：



5.要求每个结点的值大于等于链表中下一个节点的值，否则炸弹爆炸。



6.原链表中每个结点的值：



7.由要求可知因该如此排序：

0x31a>0x110>0xbe>0x98>0x61>0x41

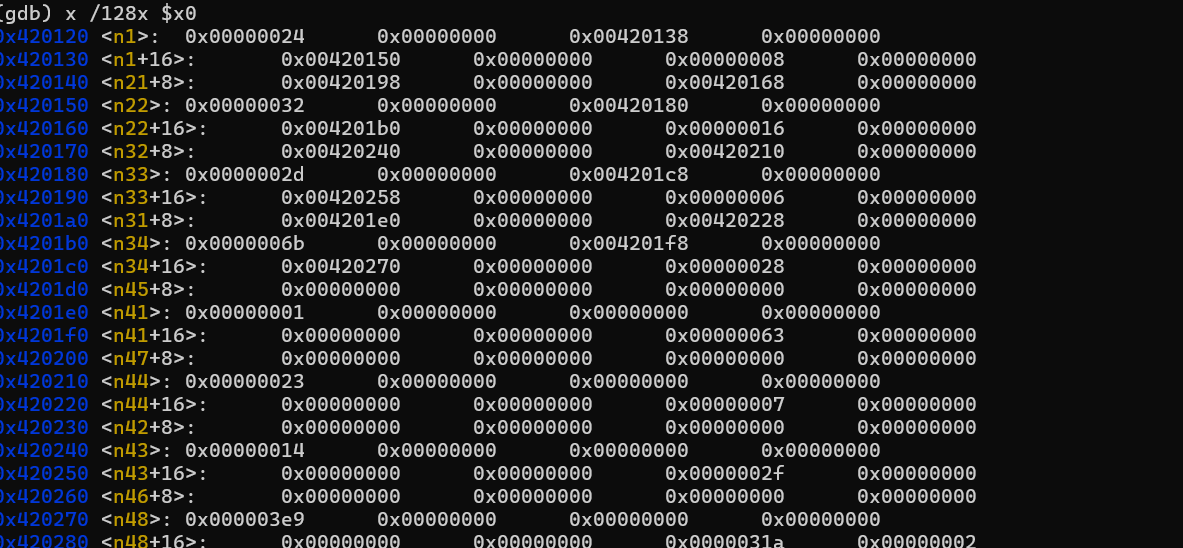
Node2> node6> node1> node3> node4> node5

根据7的补码操作可知输入为：

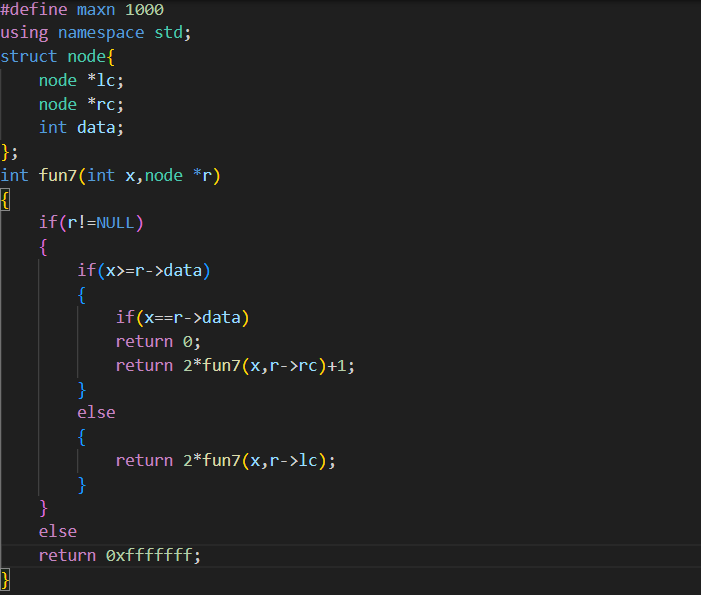
5 1 6 4 3 2

***Phase7：***

得到二叉树：



与普通lab实验相同的套路，反解出func7的C源程序



发现当输入为0x14时，也就是与节点n43值相同时，得到0x2的返回值，符合当前phase要求，解出secretphase

五、总结体会

解高阶lab实验的感觉是，当我们把arm64指令及搞清楚以后，基本上是换汤不换药，由此也熟悉了arm精简指令集的操作，属实精简，能把复杂的命令用常见的指令就表达了出来，而且表现的也很直观。在完成时间上远远快于第一个lab2实验。