

**实验报告**



**题目： 拆解二进制炸弹**

**班 级： 2022211313 \_\_\_**

**学 号： 2022211388 \_\_\_**

**姓 名： 陈韵涵 \_\_\_**

**学 院： 计算机学院**

**2023年 11月 14 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. Ubuntu(10.99.0.230)
3. Linux
4. Objdump命令反汇编
5. GDB调试工具
6. Tui

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。

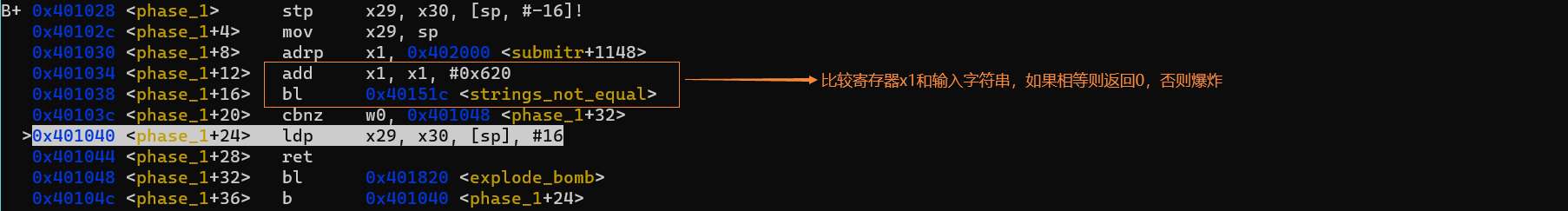
为完成二进制炸弹拆除任务，需要使用gdb调试器和objdump来反汇编bomb文件，可以单步跟踪调试每一阶段的机器代码，也可以阅读反汇编代码，从中理解每一汇编语言代码的行为或作用，进而设法推断拆除炸弹所需的目标字符串。实验2的具体内容见实验2说明。

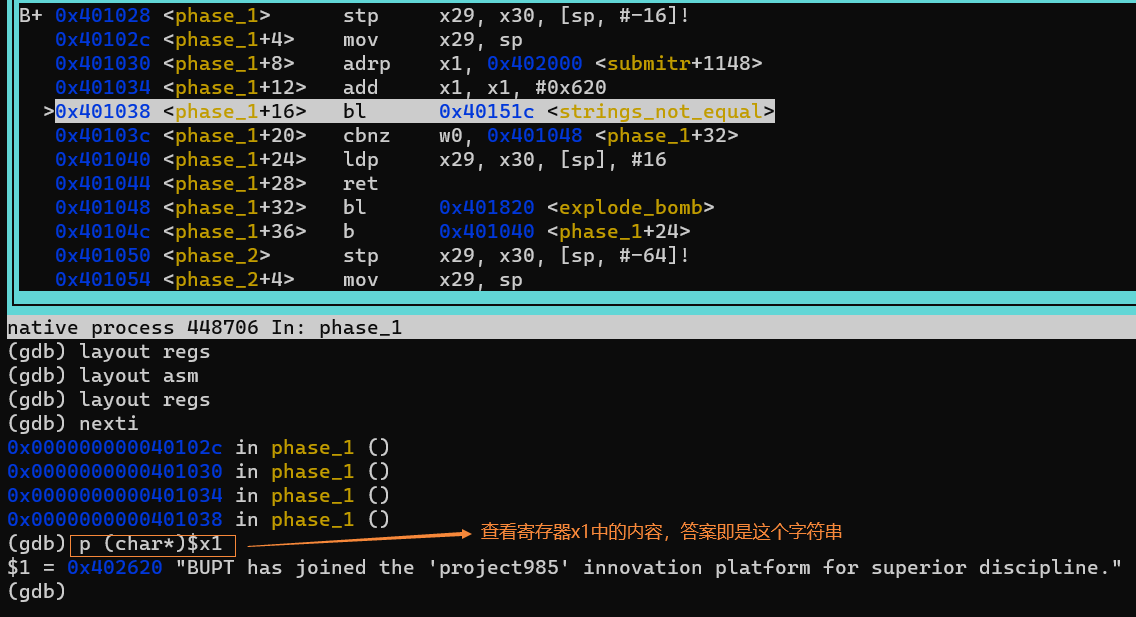
四、实验步骤及实验分析

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

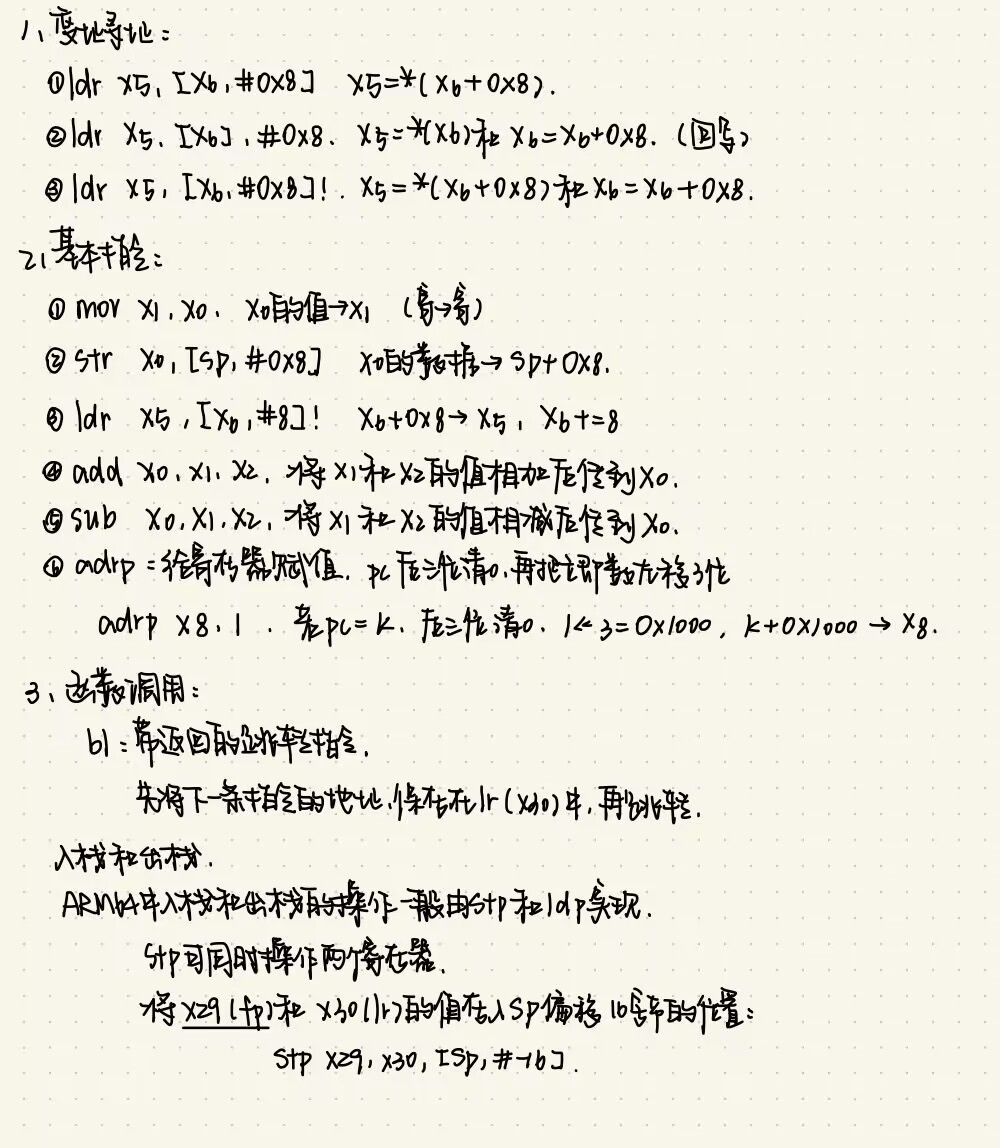
**阶段1：**







这一题学到的arm64指令：



**阶段2：**

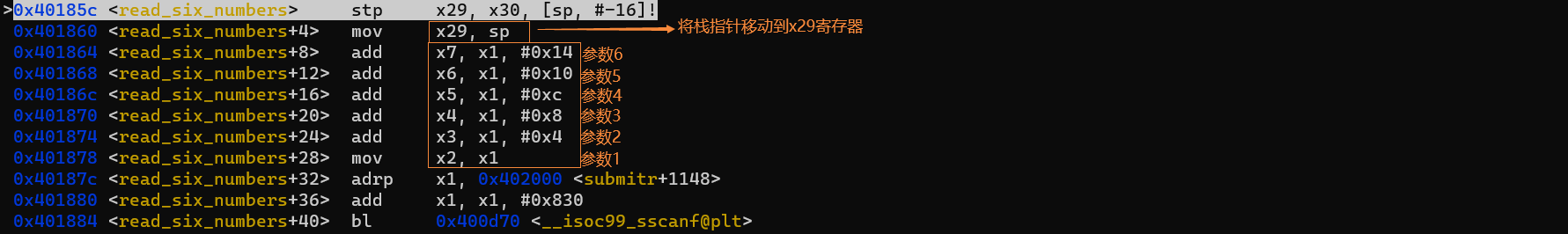


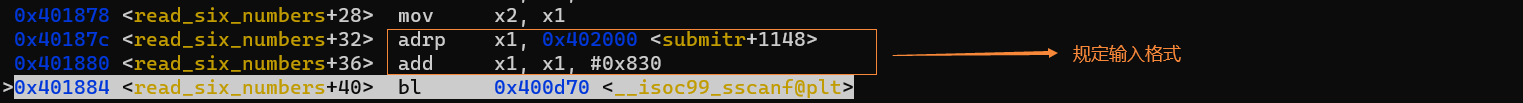
在这里需要知道：

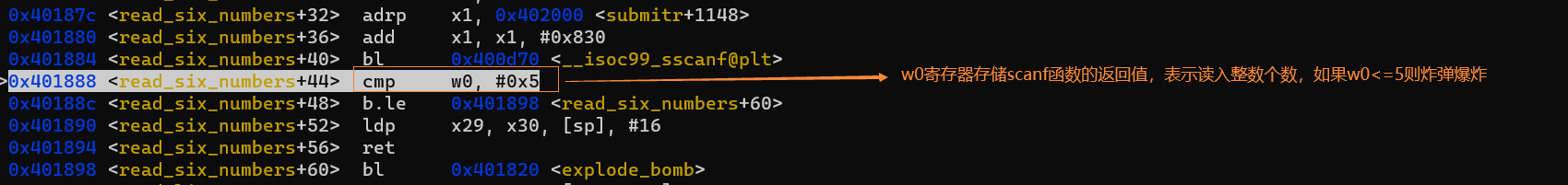
FP：栈顶指针，指向一个栈帧的顶部，当函数发生跳转时，会记录当时的栈的起始位置。

SP：栈指针（也称为栈底指针），指向栈当前的位置，

LR：链接寄存器，保存函数返回的地址。



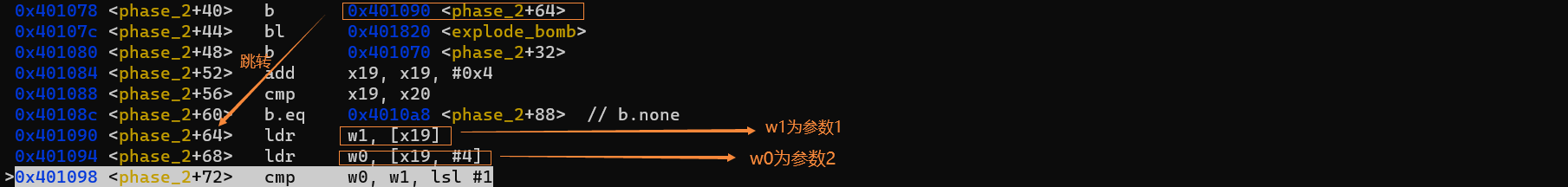




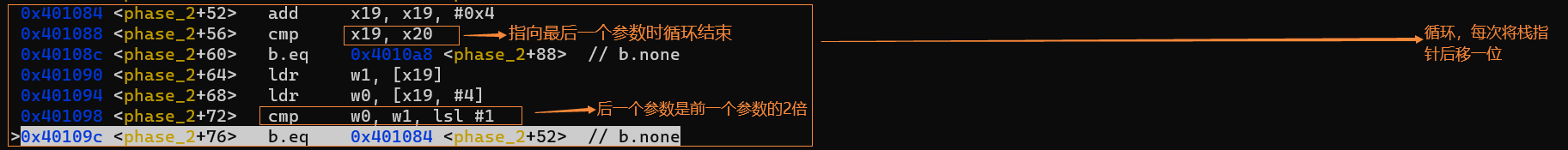
出这个函数过后再来看phase\_2剩下的内容：



得出：第一个参数是1



在ARM64汇编语言中，cmp w0, w1, lsl #1 表示比较两个寄存器中的值，其中 w0 和 w1 是32位的寄存器，而 lsl #1 是一个左移操作，表示将 w1 的值左移1位。具体来说，这条指令会将 w1 的值左移1位，然后将结果与 w0 的值进行比较，但不会修改任何寄存器的值，仅仅设置条件码（condition flags）以供后续的条件分支判断。



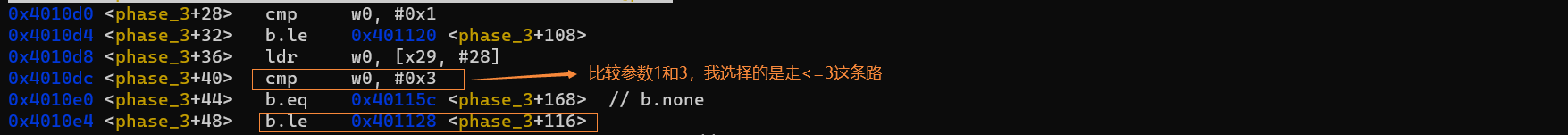
因此最后的答案是1 2 4 8 16 32

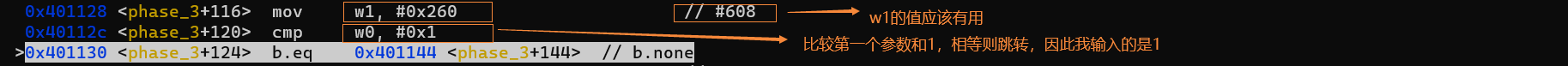


**阶段3：**

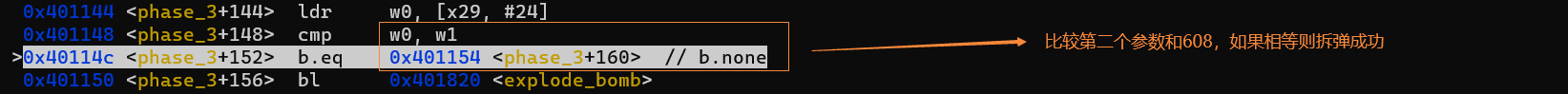


这次要求输入两个整数





目测估计这道题有多组答案，但由于我尝试的时候第一个输入的1，因此先确定第一个参数是1



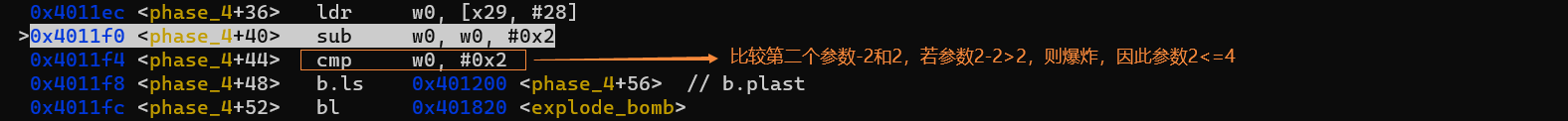
因此这个题答案之一是1 608（有时间再看看别的）



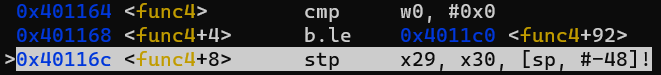
**阶段4：**



这次要输入的是两个整数



进入func4:



这条指令的作用是将当前帧的帧指针 x29 和链接寄存器 x30 的值存储到栈上，同时将栈指针 sp 向下偏移48字节。这通常发生在函数调用的开头，用于保存调用者的上下文，以便在函数结束时能够正确地恢复。

发现func4其实就是一个参数为输入的第二个数和6的递归参数，只要让输入的第二个数经过这个函数出来之后的值符合题目条件就可以了，因此先随便输入一个小于4的数，我输入的是2，再看出来后w0里面的数变成了什么。

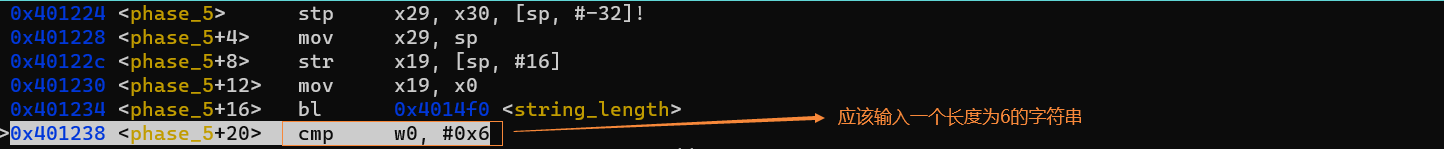


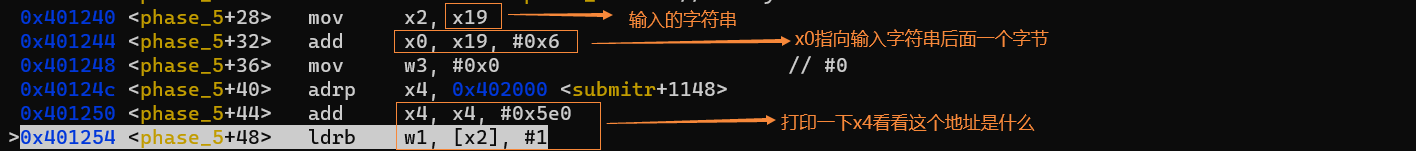
查看寄存器发现w0变成了40，因此第一个参数是40，第二个参数是2.





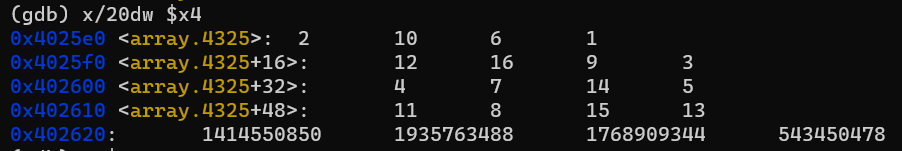
**阶段5：**



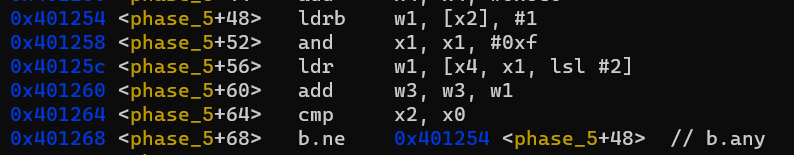


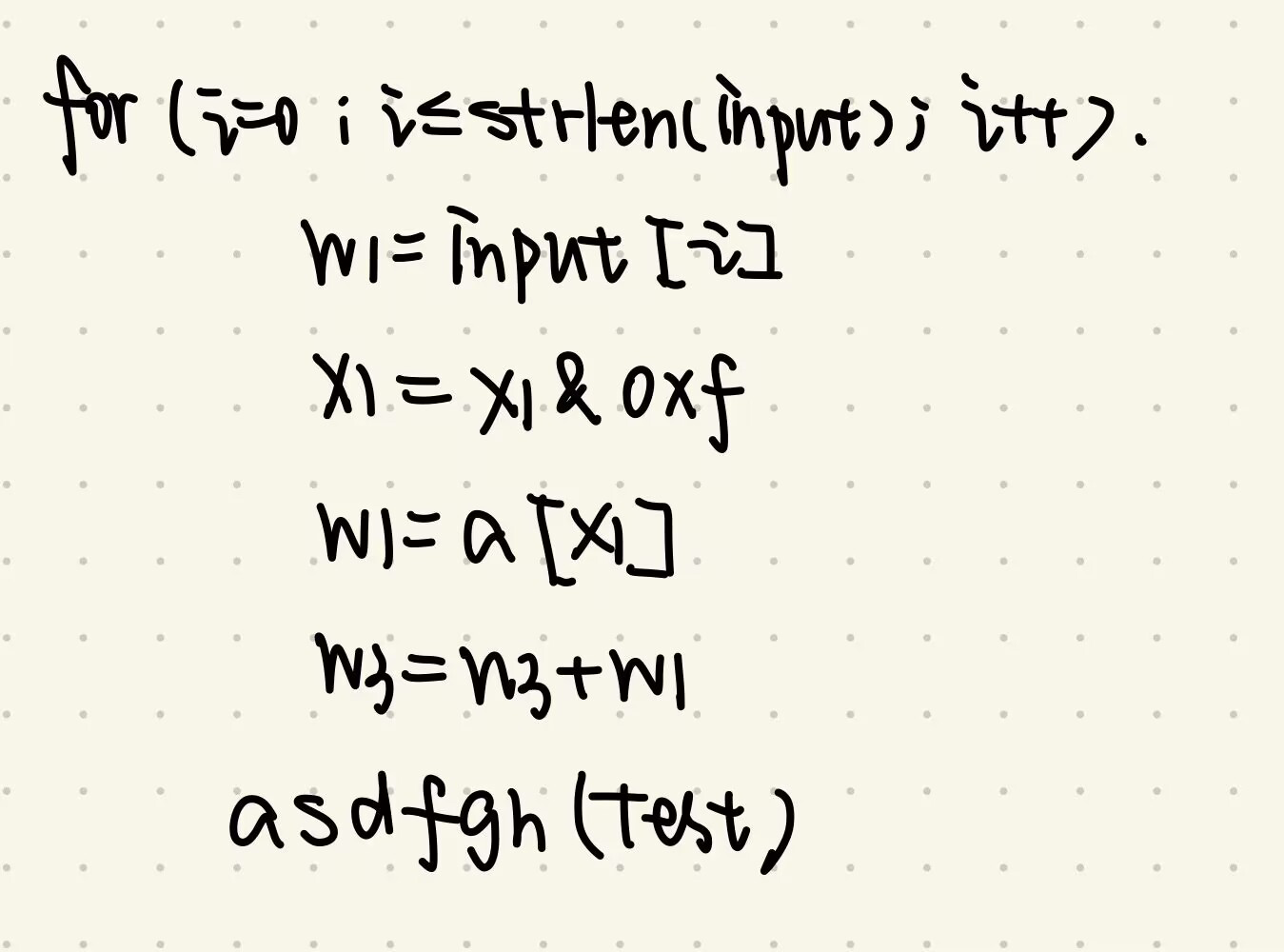


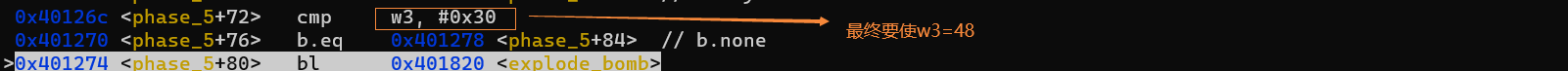
打印数组内容，发现数组有16个元素：

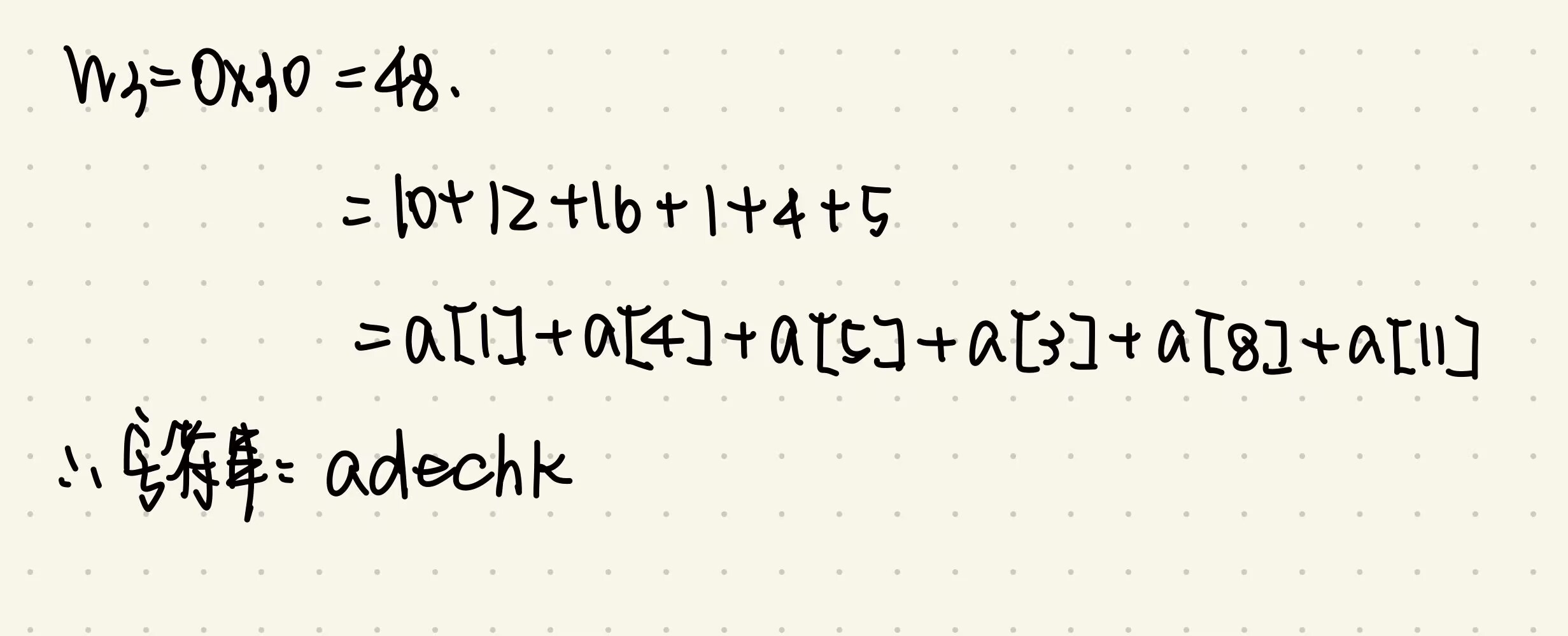


简单描述这段代码：



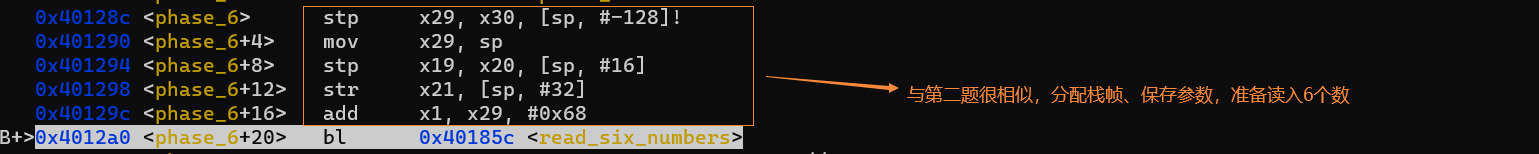








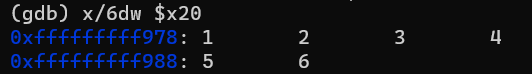
**阶段6：**



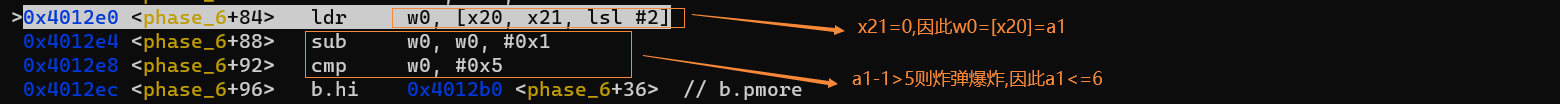
在这里设置了x21=0:



测试用的输入数据为1 2 3 4 5 6，可以看到输入数据存在x20指针指向的一块内存区域：



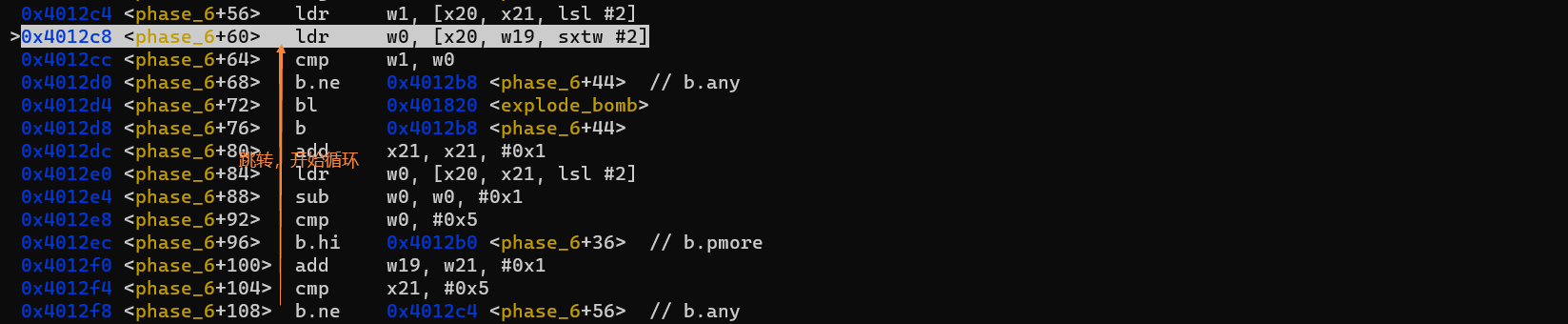
然后跳转：用a1-a6表示输入的六个整数



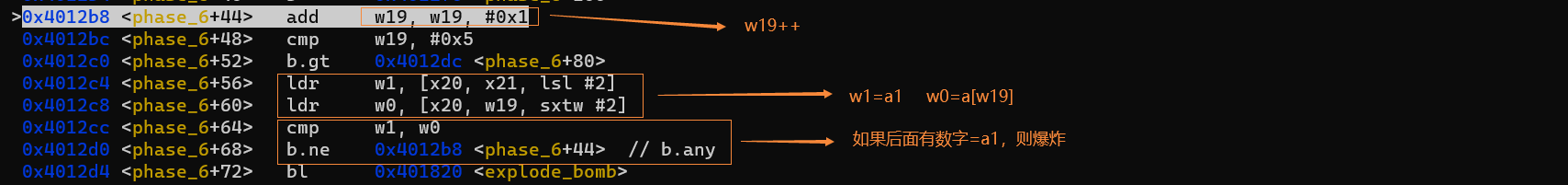
因此a1<=6

陌生指令：ldr w0, [x20, w19, sxtw #2]

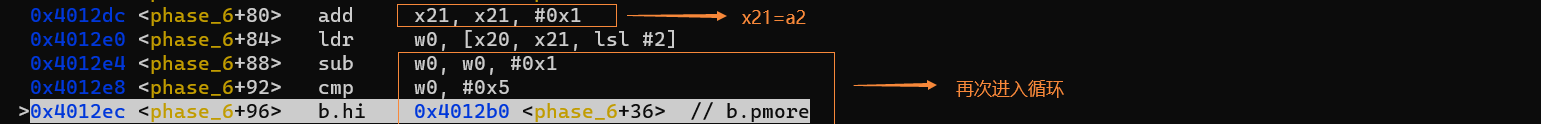
这条指令的整体意义是从内存地址 [x20 + (w19 \* 4)] 处加载一个32位的字，并将其存储在寄存器 w0 中。



这里开始进入循环：

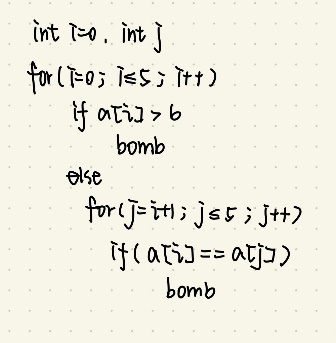


循环结束后跳出：



因此这两段代码组成了一个二重循环，循环的目的是：每个数字各不相同，每个数字<=6

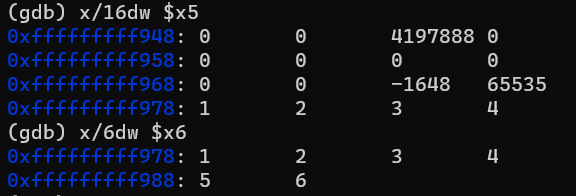
转换成C语言代码：

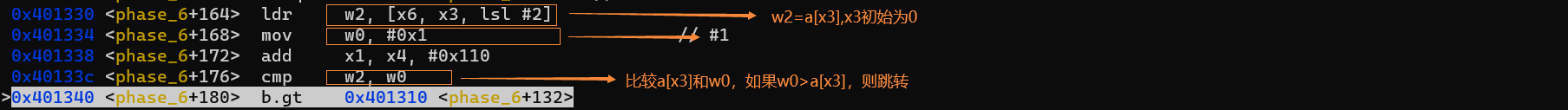


出循环后：

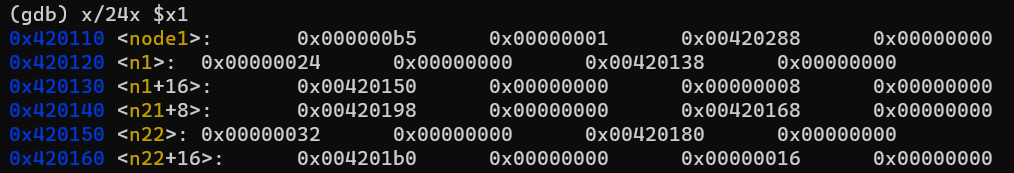


可以看出x5的指针指向输入6个数之前的12字节的空间：

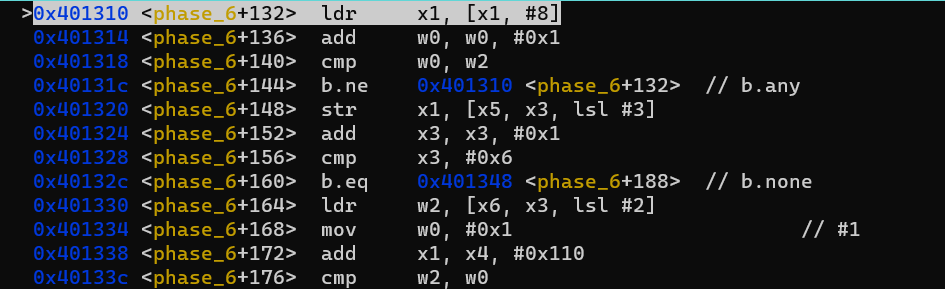


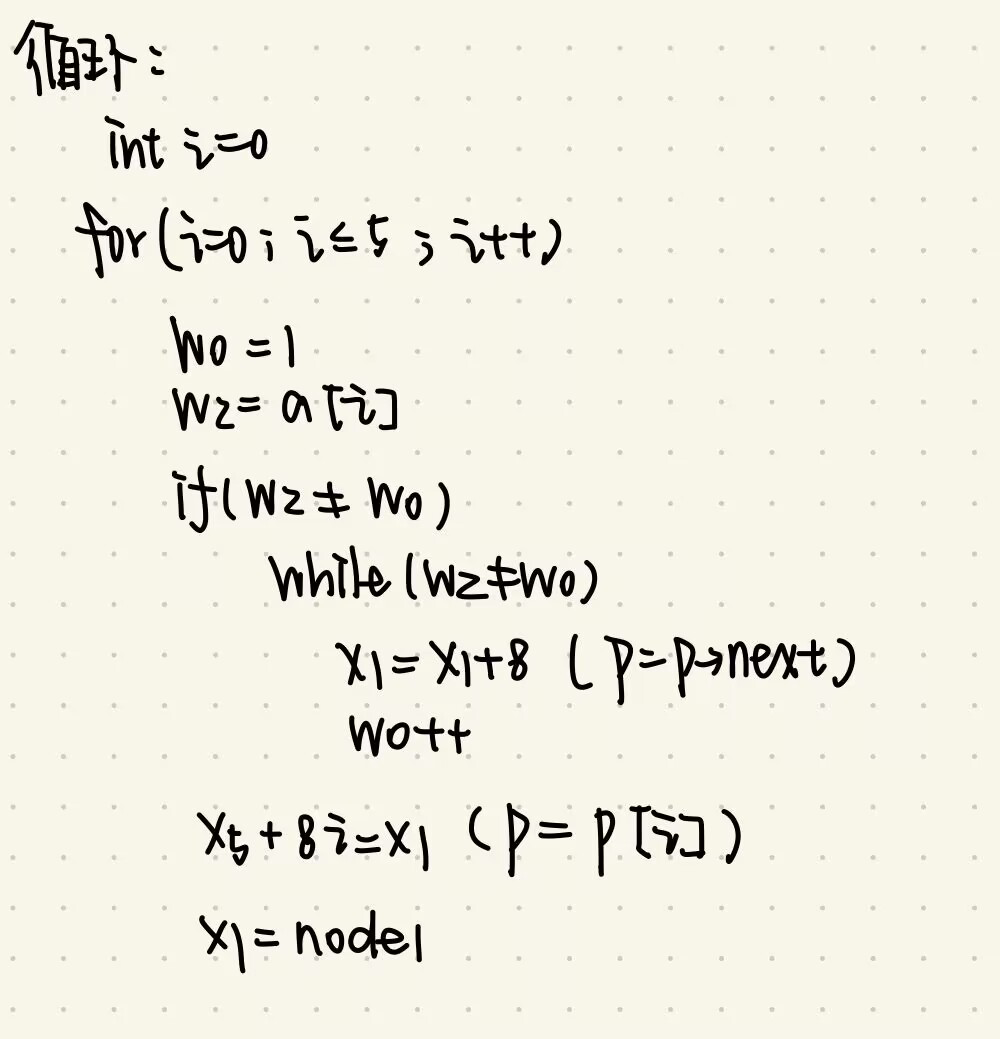


可以看到这里往x1里面存了一个地址，猜测跟6个数相关，因此打印x1的东西：x1应该存了个链表，但是现在还看不出来。

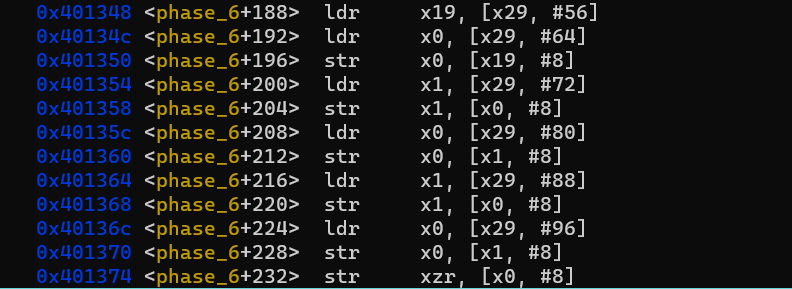


这一块是一个循环，翻译成c语言：

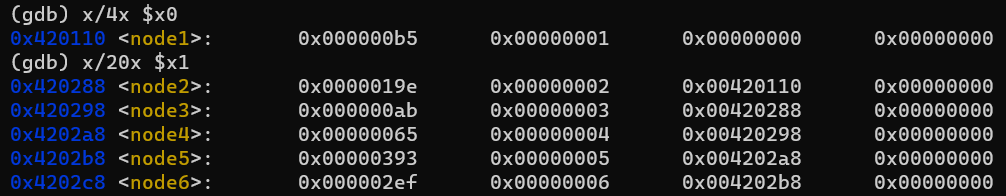


此循环的目的是把x1指向的6个节点按照输入数字的顺序重排。

把各节点地址存入寄存器x0和寄存器x1，一个节点16字节：



可以看出：每个节点存的对应的输入的数字，而第三列存储了指向后一个节点的指针



跳转：

若w1>=w0则不爆炸，w1存的是w0里面的节点的后一个节点



而链表的值：

递增：node4<node3<node1<node2<node6<node5

按这个顺序排布，则满足要求：

答案为5 6 2 1 3 4

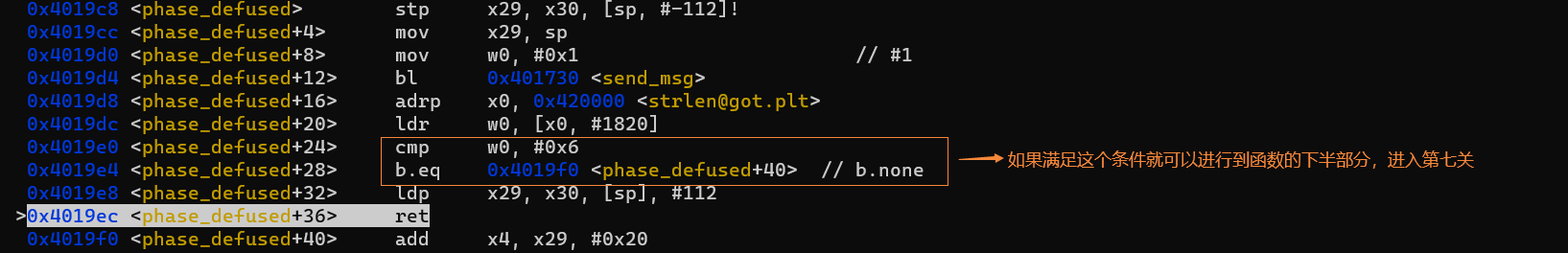


**阶段7：**

之前就有注意到:函数phase\_defused里面有这么一个函数：



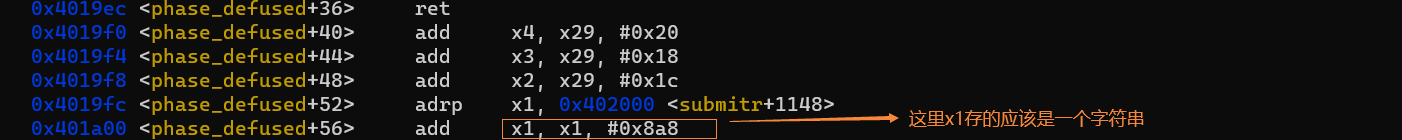
这个应该就是传说中的第七关



推测这个数字应该是表示进行到了第几关



在第六个炸弹解决后才有资格跳到后面，因此设置断点，在第六个炸弹以后进入phase\_defused



查看发现果然又是输入格式：

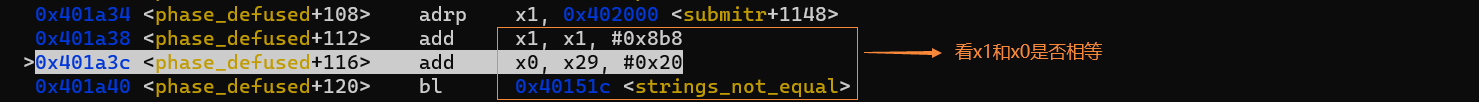


因为只有第四关可以输三个数也不影响，于是在第四关后面的答案后加上test



查看x0:发现果然这里要读的是第四关输入的东西

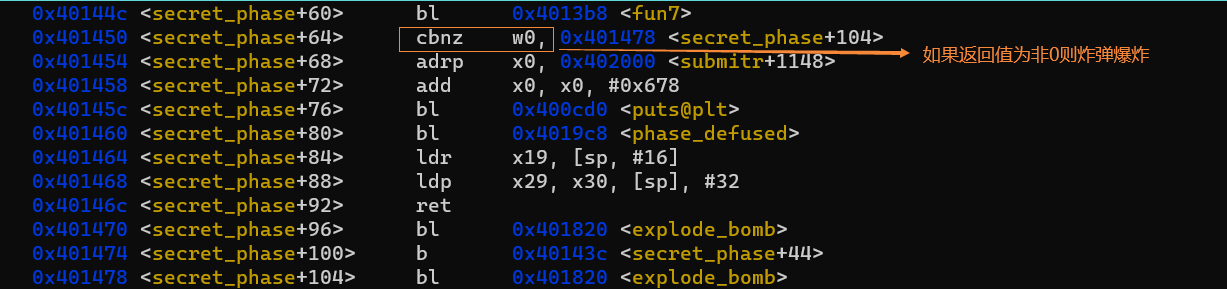




因此x1里面存的就是要输入的东西：

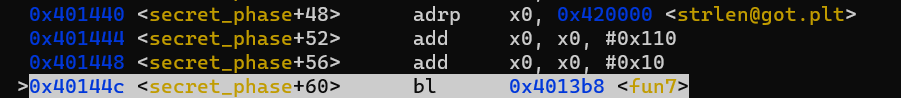


修改phase\_4答案后进入第七关：

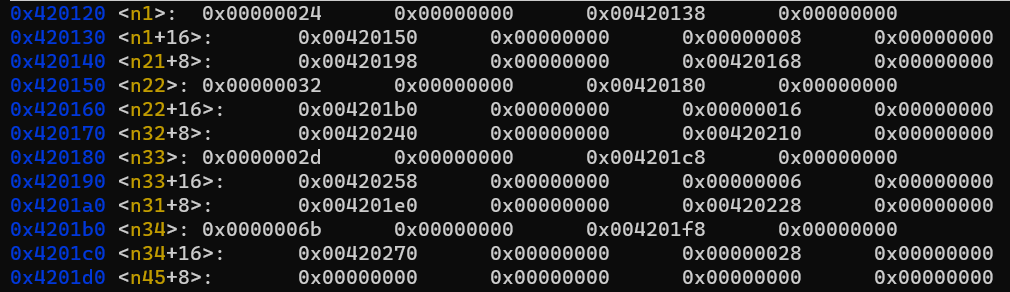


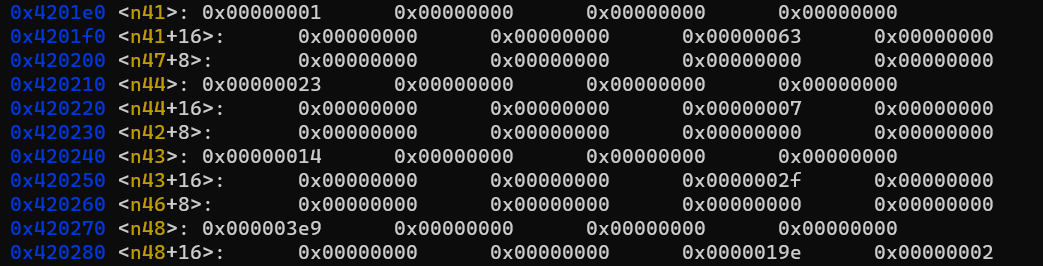
进入func7:

发现进入之前打印了一个参数：

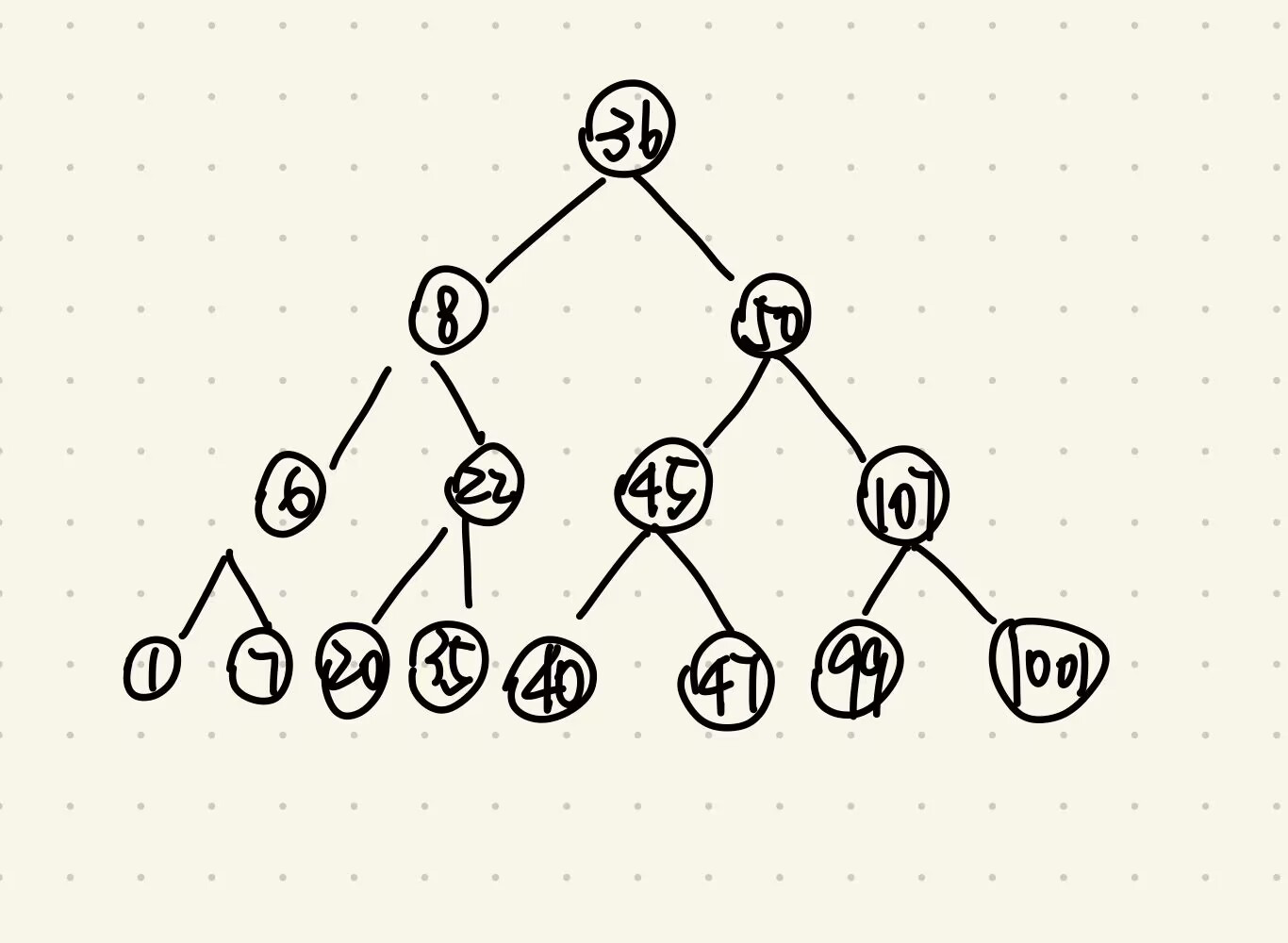


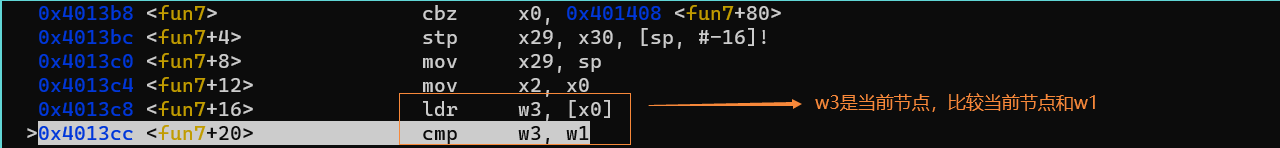
查看该内存地址的东西：





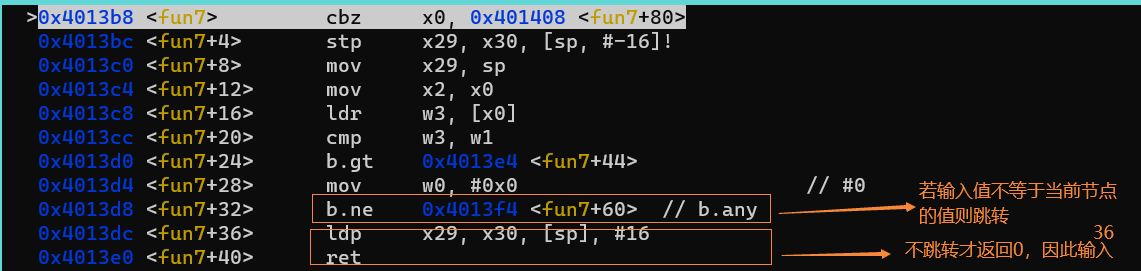
是个每个元素8字节的二叉树，每24个字节是下一个节点，画出二叉树:





第七关的任务是使func7返回0，而返回值在w0里面

查看func7:





五、总结体会

总结心得（包括实验过程中遇到的问题、如何解决的、过关或挫败的感受、实验投入的时间和精力、意见和建议等）

1. 各处理器其实大同小异，真正要提升性能的话还要走很长的路。

2. 感受：只要善用gdb和断点是不会爆炸的，心情从最开始的战战兢兢变到逐渐放松，但炸弹爆炸了心里还是会起起落落一下，觉得有点伤心。

3. 投入精力：感觉投入精力挺多的，其实有些关卡也不用完全搞懂就可以做出来，但要是不把每个指令搞懂就试出来的话意义不大。在第四关耗了很久，因为仅仅看机器代码很难看出递归是怎么进行的。

4. 对于数据结构的理解也很重要，最后一关没能一眼看出那个是二叉树，不得已才上网搜索，知道是二叉树之后就能比较方便地做出来了。