山东大学 计算机科学与技术 学院

计算机系统原理 课程实验报告

学号： 201700130011	姓名：刘建东	班级：计科 17 级菁英班
实验题目： 二进制炸弹拆除		
实验学时：8 学时	实验日期： 11.20-12.11	
实验目的： (1) 熟悉 MIPS 指令集 (2) 根据反汇编程序可以分析程序的功能和执行流程 (3) 熟悉 GDB 调试工具，帮助程序理解		
硬件环境： 2.3 GHz Intel Core i5 16 GB 2133 MHz LPDDR3、64 位操作系统		
软件环境： Sublime G++		
实验步骤与内容： 首先先列举一下常用的 gdb 调试命令： 1.gdb 直接进入 gdb 调试模式 2.disas function_name 查看函数反汇编代码 3.files file 打开文件 4.x /16c $a1 查看寄存器 a1 中的内容，显示 16 个字符 5.ni 单步调试 6.r 运行程序 7.b *0x00400d78 设置断点 实验步骤： (1)phase_1： 可以看到程序就是比较你输入的字符串和给定一个字符串是否相同，如果不相同则炸弹爆炸，因此是需要在进入比较函数之前的地方设一个断点，查看$a1 和$a2 的值即可，然后就得到了答案，为“Let’s begin now!”		

```
0x00400d88 <+28>:    addiu   a1,v0,10092
0x00400d8c <+32>:    jal 0x401cf8 <strings_not_equal>
0x00400d90 <+36>:    nop
0x00400d94 <+40>:    beqz    v0,0x400da4 <phase_1+56>
0x00400d98 <+44>:    nop
0x00400d9c <+48>:    jal 0x4021f0 <explode_bomb>
```

(2)phase_2:

一步步进行分析，首先是读入 6 个数字。

```
0x00400de4 <+40>: move  a1,v0
// $a1 = $v0
0x00400de8 <+44>: jal   0x401ba8 <read_six_numbers> //跳到地址0x401ba8，跳出函数
```

然后要求第一个数字必须为 1，否则炸弹会爆炸。

```
0x00400df8 <+60>: li v0,1                    //将1加载到vo中
0x00400dfc <+64>: beq   v1,v0,0x400e10 <phase_2+84>
//v1与v0内容相同时，跳转
```

接下来开始循环，将你输入的数字与答案依次比较，只要有一个不正确，就会直接爆炸。

```
//循环开始
0x00400e20 <+100>:   lw v0,24(s8)     //*****************************************循环开始
0x00400e24 <+104>:   nop
0x00400e28 <+108>:   addiu v0,v0,-1
```

在关键处直接加断电，查看每个数字即可得到答案，为 1 1 1 0 0 0。

```
0x00400e7c <+192>:   lw v0,4(v0)    // v0变成下一个输入的数
0x00400e80 <+196>:   nop
0x00400e84 <+200>:   beq    a0,v0,0x400e98 <phase_2+220>
//beq 当两个寄存器内容相同时，发生跳转
```

(3)phase_3:

一个 switch 结构，来依次进行分析。首先可以看出第一个数字必须小于 8。

```
0x00400f48 <+116>:   lw v0,44(s8)  //break
//v0为输入数字的第一个数，即s8保存了输入的数字，偏移44为第一个数字，输入的数组转移到了v0
**************************************
0x00400f4c <+120>:   nop
0x00400f50 <+124>:   sltiu v1,v0,8
//v0 < 8, v1 = 1; else v1 = 0;
0x00400f54 <+128>:   beqz  v1,0x401190 <phase_3+700>
//v1 == 0,跳转，所以v0 < 8，第一个输入的数字 < 8
```

然后会根据你输入的数字进入不同的分支，此处我选择的是 5。

进入分支后，观察代码可以发现，程序读出了输入的第三个数字，并将第三个数字与 513 进行比较，如果不相同，则炸弹爆炸，由此得到了第三个数字。

```
0x004010cc <+504>:    li v0,116                 //第一个数为5
*********************************************
// v0 = 116
0x004010d0 <+508>:    sb v0,32(s8)    //将一个字节的数据从寄存器中存储到存储器中
0x004010d4 <+512>:    lw v0,-32660(gp)
0x004010d8 <+516>:    nop
0x004010dc <+520>:    lw v1,44(v0) //break
//v1变成1,和输入数字无关
0x004010e0 <+524>:    lw v0,36(s8) //break
//v0变成输入的第三个数字
*****************************************************************************************
0x004010e4 <+528>:    nop
0x004010e8 <+532>:    mult   v1,v0
//v1*v0结果放入寄存器lo中，还有个寄存器hi
0x004010ec <+536>:    mflo   v1
//从寄存器lo中取出数据
0x004010f0 <+540>:    li v0,513
//v0 = 513
0x004010f4 <+544>:    beq    v1,v0,0x4011e8 <phase_3+788>**************************
v1为输入的第三个数字
```

查看程序跳转位置处的代码，可以看到它取了你输入的第二个数与答案进行比较，可以发现是 ASCII 码形式，于是第二个数为 t，由此得到答案 5 t 513。

```
0x004011e8 <+788>:    nop ********************************** 直接跳转到了这里
0x004011ec <+792>:    b  0x4011f8 <phase_3+804>
0x004011f0 <+796>:    nop
0x004011f4 <+800>:    nop
0x004011f8 <+804>:    lb v0,40(s8) //v0 = 57 — 9, v0 = 49 — 1, v0 = 50 — 2
********************** 断点
0x004011fc <+808>:    lb v1,32(s8) //v1 = 116
0x00401200 <+812>:    nop
0x00401204 <+816>:    beq    v1,v0,0x401218 <phase_3+836>
//v1 = v0则跳转
```

(4)phase_4：

这是一个斐波那契数列，会根据你输入的数字得到斐波那契数列中对应的值，然后将这个值与答案进行比较，不相同则爆炸。

```
0x0040136c <+176>:    jal    0x401230 <func4> ********************************** break
   求feb[n],n为第一个数
0x00401370 <+180>:    nop  //1 1 2 3 5 8
0x00401374 <+184>:    lw gp,16(s8)
0x00401378 <+188>:    move   v1,v0
0x0040137c <+192>:    li v0,8
0x00401380 <+196>:    beq    v1,v0,0x4013d0 <phase_4+276> //相同跳转
```

先根据学号末尾最后一个数字判断跳入哪个分支，跳入分支后输出 feb[n]，n 为你输入的数，此处和 8 进行比较，因此 1 1 2 3 5 8 可知，答案为 5。

(5)phase_5：

要求输入一个六位长度的字符串，然后将每一位读出来，进行&运算，取 ASCII 码的最后四位。然后将这四位表示的 ASCII 数值对应到程序设定的 0-15 位数组中，将对应位置的字符取出和预设的字符进行比较，如果不相同，则爆炸。

合适位置设置断点后，可以查看到预设字符为 giants，多次尝试之后可以得到对应数组为 i s r v e a w h o b p n u t f g，于是对应 ASCII 表找到答案即可，答案不唯一，OPEKMA 为可行的答案。

```
0x004014a0 <+184>:    addiu v0,v0,1
0x004014a4 <+188>:    sw v0,24(s8)
0x004014a8 <+192>:    lw v0,24(s8) //v0 = 0
0x004014ac <+196>:    nop
0x004014b0 <+200>:    slti   v0,v0,6 //循环6次**********************************break
```

(6)phase_6：

将你输入的六个数对应到预设的数组中，得到答案数组，然后依次比较答案数组的相邻位，要求构成一个递减数组。

首先一个循环读入 6 个数字。

```
0x00401618 <+280>:   lw v0,24(s8)        //**********************跳到这里
0x0040161c <+284>:   nop
0x00401620 <+288>:   slti  v0,v0,6
//v0 < 6, v0 = 1; else v0 = 0;
0x00401624 <+292>:   bnez  v0,0x4015c0 <phase_6+192> //****** v0 < 6,则跳转
```

然后一个二重循环先查看输入数字是否大于 7，然后再循环之后的每一位数字，判断有没有相等的数字。如果有相等数字则爆炸。

```
/*************************************************************************
第一个二重循环先判断这个数是否大于7，再判断有没有相等的数字
*************************************************************************/
0x0040164c <+332>:   nop
0x00401650 <+336>:   sw zero,28(s8)
0x00401654 <+340>:   b  0x4016f8 <phase_6+504> //************************* 二重循环
0x00401658 <+344>:   nop
0x0040165c <+348>:   lui   v0,0x41
0x00401660 <+352>:   addiu v0,v0,12592
```

最后一个一重循环，查看当前数字是否比前一个数字小，如果大的话就爆炸。合适位置加断点，即可读出每一个数字对应的数字，然后排成降序依次输入即可。由此可得答案为 4 2 6 3 1 5。

```
0x0040183c <+828>:   slt   v0,v0,v1 //************************************break
/*
判断输入的六个数中，是否前一个数所对应的值比当前数字所对应的值大，如果小，则爆炸
1 —— 253，2 —— 725，3 —— 301，4 —— 997，5 —— 212，6 —— 432
排成降序就是 4 2 6 3 1 5
*/
0x00401840 <+832>:   beqz  v0,0x401854 <phase_6+852>
0x00401844 <+836>:   nop
0x00401848 <+840>:   jal   0x4021f0 <explode_bomb>
0x0040184c <+844>:   nop
0x00401850 <+848>:   lw gp,16(s8)
0x00401854 <+852>:   lw v0,32(s8)
0x00401858 <+856>:   nop
0x0040185c <+860>:   lw v0,8(v0)
0x00401860 <+864>:   nop
0x00401864 <+868>:   sw v0,32(s8)
0x00401868 <+872>:   lw v0,28(s8)
0x0040186c <+876>:   nop
0x00401870 <+880>:   addiu v0,v0,1
0x00401874 <+884>:   sw v0,28(s8)
0x00401878 <+888>:   lw v0,28(s8)
0x0040187c <+892>:   nop
0x00401880 <+896>:   slti  v0,v0,5
0x00401884 <+900>:   bnez  v0,0x4017b4 <phase_6+692> //******************* 一重循环
依次判断每个数是否符合要求
```

(7)隐藏炸弹：

首先要找到隐藏炸弹的入口，只能通过 main 函数来找了。发现 phase_defused 函数。

```
0x00400ca0 <+928>:   jal   0x4012bc <phase_4>
0x00400ca4 <+932>:   nop
0x00400ca8 <+936>:   lw gp,16(s8) ******************************************break
0x00400cac <+940>:   jal   0x402264 <phase_defused>
```

在适当位置加断点，找到入口。

```
0x004022ac <+72>: move  a1,v1       //*********************************在这里x /16c 察看v1、v0
//发现v1里面存的是%d %s austinpowers,
发现每一关答案里面只有第四关是输入单独一个5，于是考虑在那一关后面输入这个字符串，于是进入到隐藏炸弹中
```

进入 serect_phase 函数。发现 func7 是一个二叉搜索树，会根据你输入的数字在二叉搜索树内进行查找。如果输入数字比当前数字大则往右走，如果比当前数字小则往左走，直到无法再走程序结束。程序记录一个变量 i，初值为 0，往右则*2+1，往左则*2，采取递归计算，即无法走以后，再一步步递归回来计算。

程序初始值为 36，比 36 小是 8 6 1，比 36 大是 50 107 1001，由此输入 1001，i 即可变为 7，即可拆除炸弹。

```
0x00401a18 <+136>:   lw a1,24(s8)     //a1为输入数字*****************************************
0x00401a1c <+140>:   jal   0x4018a4 <fun7>
0x00401a20 <+144>:   nop
0x00401a24 <+148>:   lw gp,16(s8)
0x00401a28 <+152>:   move  v1,v0
0x00401a2c <+156>:   li v0,7 //*************************************************与7比较
0x00401a30 <+160>:   beq   v1,v0,0x401a44 <secret_phase+180>
```

结论分析与体会：

1.反汇编调试首先要看出代码的整体架构，把握循环、条件、递归结构，将程序进行分块阅读。

2.整理出整体架构后，需要在关键位置加断点，查看寄存器内部信息，可以更快的把握程序的目的与作用，理解整个程序。

3.需要用好常用的 gdb 调试技巧，display、ni、x /16c、x /16x 用好这些语句可以减少大量的调试时间。

附件：程序代码、调试过程及运行结果（截图）

备注：Lab1 需要提交源码