《计算机系统》

BombLab 实验报告

班级：软件 2203

学号：202226010306

姓名：白旭

目录

1 实验项目二

1.1 项目名称

BombLab

1.2 实验目的

1) 破译所有问题

2) 熟练掌握汇编代码的阅读

1.3 实验资源

1) bomb：可执行文件，无法打开，我们主要研究的对象

2) bomb.c 文件

3) bomb-quiet：暂时不需要关注

4) README 文件

2 实验任务

2.1 实验任务 A

1) Phase_1

```
08048b50 <phase_1>:
 8048b50:       83 ec 1c                sub    $0x1c,%esp  #给栈分配28字节空间
 8048b53:       c7 44 24 04 ac a1 04    movl   $0x804a1ac,0x4(%esp)  #目标字符串
 8048b5a:       08
 8048b5b:       8b 44 24 20             mov    0x20(%esp),%eax  #输入的字符串
 8048b5f:       89 04 24                mov    %eax,(%esp)
 8048b62:       e8 7d 04 00 00          call   8048fe4 <strings_not_equal>  #比较字符串是否相等
 8048b67:       85 c0                   test   %eax,%eax  #返回值为0则跳转
 8048b69:       74 05                   je     8048b70 <phase_1+0x20>
 8048b6b:       e8 86 05 00 00          call   80490f6 <explode_bomb>
 8048b70:       83 c4 1c                add    $0x1c,%esp
 8048b73:       c3                      ret
```

观察得知 je 跳转时不会爆炸，也就是<strings_not_equal>的返回值为 0 时，即要求输入的字符串要与目标字符串相等；进行比较的两个字符串位置应为 0x4(%esp)和(%esp)，可以想到以字符串类型查看 0x4(%esp)存的地址中有什么；

```
(gdb) x/s 0x804a1ac
0x804a1ac:      "We have to stand with our North Korean allies."
```

此字符串应为进行比较的字符串之一，只要输入此字符串即可通过 phase_1;

2) Phase_2

```
08048b74 <phase_2>:
 8048b74:       56                      push   %esi
 8048b75:       53                      push   %ebx
 8048b76:       83 ec 34                sub    $0x34,%esp
 8048b79:       8d 44 24 18             lea    0x18(%esp),%eax
 8048b7d:       89 44 24 04             mov    %eax,0x4(%esp)
 8048b81:       8b 44 24 40             mov    0x40(%esp),%eax
 8048b85:       89 04 24                mov    %eax,(%esp)
 8048b88:       e8 9e 06 00 00          call   804922b <read_six_numbers>  #读入6个数字
 8048b8d:       83 7c 24 18 00          cmpl   $0x0,0x18(%esp)
 8048b92:       75 07                   jne    8048b9b <phase_2+0x27>  #若0x18(%esp)不等于0,跳到bomb
 8048b94:       83 7c 24 1c 01          cmpl   $0x1,0x1c(%esp)
 8048b99:       74 05                   je     8048ba0 <phase_2+0x2c>  #若0x1c(%esp)等于1,跳过bomb
 8048b9b:       e8 56 05 00 00          call   80490f6 <explode_bomb>
 8048ba0:       8d 5c 24 20             lea    0x20(%esp),%ebx
 8048ba4:       8d 74 24 30             lea    0x30(%esp),%esi
 8048ba8:       8b 43 f8                mov    -0x8(%ebx),%eax  #%ebx之前第二个数
 8048bab:       03 43 fc                add    -0x4(%ebx),%eax  #%ebx之前第二个数与之前第一个数之和
 8048bae:       39 03                   cmp    %eax,(%ebx)
 8048bb0:       74 05                   je     8048bb7 <phase_2+0x43>  #若%ebx等于其之前两个数之和，跳过bomb
 8048bb2:       e8 3f 05 00 00          call   80490f6 <explode_bomb>
 8048bb7:       83 c3 04                add    $0x4,%ebx
 8048bba:       39 f3                   cmp    %esi,%ebx
 8048bbc:       75 ea                   jne    8048ba8 <phase_2+0x34>  #若%esi不等于%ebx，跳转回8048ba8
 8048bbe:       83 c4 34                add    $0x34,%esp
 8048bc1:       5b                      pop    %ebx
 8048bc2:       5e                      pop    %esi
 8048bc3:       c3                      ret
```

观察有<read_six_numbers>得知要读入 6 个数字；分别储存在从 0x18(%esp)到 0x2c(%esp)六个地址中；第一个为 0，第二个为 1，之后读取循环内容得知每个数为其之前两个数之和，即斐波那契数列；前六项为 0 1 1 2 3 5，即为答案；

3) Phase_3

```
8048bc7:    8d 44 24 1c          lea    0x1c(%esp),%eax  #第二个参数位置
8048bcb:    89 44 24 0c          mov    %eax,0xc(%esp)
8048bcf:    8d 44 24 18          lea    0x18(%esp),%eax  #第一个参数位置
8048bd3:    89 44 24 08          mov    %eax,0x8(%esp)
8048bd7:    c7 44 24 04 9f a3 04 movl   $0x804a39f,0x4(%esp)  #格式为"%d %d"
8048bde:    08
8048bdf:    8b 44 24 30          mov    0x30(%esp),%eax
8048be3:    89 04 24             mov    %eax,(%esp)
8048be6:    e8 85 fc ff ff       call   8048870 <__isoc99_sscanf@plt>
8048beb:    83 f8 01             cmp    $0x1,%eax
8048bee:    7f 05                jg     8048bf5 <phase_3+0x31>  #要求输入参数数量>1
8048bf0:    e8 01 05 00 00       call   80490f6 <explode_bomb>
8048bf5:    83 7c 24 18 07       cmpl   $0x7,0x18(%esp)
8048bfa:    77 3c                ja     8048c38 <phase_3+0x74>  #若第一个参数>7，跳到bomb
8048bfc:    8b 44 24 18          mov    0x18(%esp),%eax
8048c00:    ff 24 85 0c a2 04 08 jmp    *0x804a20c(,%eax,4)  #根据第一个参数值跳转不同位置
8048c07:    b8 68 03 00 00       mov    $0x368,%eax              #%eax=0,x=872
8048c0c:    eb 3b                jmp    8048c49 <phase_3+0x85>
8048c0e:    b8 af 01 00 00       mov    $0x1af,%eax              #%eax=2,x=431
8048c13:    eb 34                jmp    8048c49 <phase_3+0x85>
8048c15:    b8 87 01 00 00       mov    $0x187,%eax              #%eax=3,x=391
8048c1a:    eb 2d                jmp    8048c49 <phase_3+0x85>
8048c1c:    b8 1b 03 00 00       mov    $0x31b,%eax              #%eax=4,x=795
8048c21:    eb 26                jmp    8048c49 <phase_3+0x85>
8048c23:    b8 59 01 00 00       mov    $0x159,%eax              #%eax=5,x=345
8048c28:    eb 1f                jmp    8048c49 <phase_3+0x85>
8048c2a:    b8 e2 03 00 00       mov    $0x3e2,%eax              #%eax=6,x=994
8048c2f:    eb 18                jmp    8048c49 <phase_3+0x85>
8048c31:    b8 77 00 00 00       mov    $0x77,%eax               #%eax=7,x=119
8048c36:    eb 11                jmp    8048c49 <phase_3+0x85>
8048c38:    e8 b9 04 00 00       call   80490f6 <explode_bomb>
8048c3d:    b8 00 00 00 00       mov    $0x0,%eax
8048c42:    eb 05                jmp    8048c49 <phase_3+0x85>
8048c44:    b8 1e 03 00 00       mov    $0x31e,%eax              #%eax=1,x=798
```

输入两个参数，读取 0x804a39f 处内容得知格式为"%d %d"

```
(gdb) x/s 0x804a39f
0x804a39f:      "%d %d"
```

第一个参数的范围为 0-7，根据第一个参数值不同跳转到跳转表的不同位置，

一共有八个不同答案；读取 0x804a20c 后八个地址，进而得知具体跳转到哪；

```
(gdb) x/10w 0x804a20c
0x804a20c:      0x08048c07      0x08048c44      0x08048c0e      0x08048c15
0x804a21c:      0x08048c1c      0x08048c23      0x08048c2a      0x08048c31
```

八个答案分别为：

```
#%eax=0,x=872

#%eax=2,x=431

#%eax=3,x=391

#%eax=4,x=795

#%eax=5,x=345

#%eax=6,x=994

#%eax=7,x=119



#%eax=1,x=798
```

4) Phase_4

```
08048cc1 <phase_4>:
 8048cc1:       83 ec 2c                sub    $0x2c,%esp
 8048cc4:       8d 44 24 1c             lea    0x1c(%esp),%eax  #第二个参数y
 8048cc8:       89 44 24 0c             mov    %eax,0xc(%esp)
 8048ccc:       8d 44 24 18             lea    0x18(%esp),%eax  #第一个参数x
 8048cd0:       89 44 24 08             mov    %eax,0x8(%esp)
 8048cd4:       c7 44 24 04 9f a3 04    movl   $0x804a39f,0x4(%esp)  #scanf,与phase_3相同
 8048cdb:       08
 8048cdc:       8b 44 24 30             mov    0x30(%esp),%eax
 8048ce0:       89 04 24                mov    %eax,(%esp)
 8048ce3:       e8 88 fb ff ff          call   8048870 <__isoc99_sscanf@plt>
 8048ce8:       83 f8 02                cmp    $0x2,%eax
 8048ceb:       75 0d                   jne    8048cfa <phase_4+0x39>  #要求输入参数数量等于2
 8048ced:       8b 44 24 18             mov    0x18(%esp),%eax
 8048cf1:       85 c0                   test   %eax,%eax
 8048cf3:       78 05                   js     8048cfa <phase_4+0x39>  #若x<0,bomb
 8048cf5:       83 f8 0e                cmp    $0xe,%eax
 8048cf8:       7e 05                   jle    8048cff <phase_4+0x3e>  若x>14,bomb
 8048cfa:       e8 f7 03 00 00          call   80490f6 <explode_bomb>
```

第一部分说明 phase_4 传入两个参数 x 和 y，格式为"%d %d"，且 x 大于等于 0，小于等于 14

```
8048cff:       c7 44 24 08 0e 00 00    movl   $0xe,0x8(%esp)  #func4的第三个参数
8048d06:       00
8048d07:       c7 44 24 04 00 00 00    movl   $0x0,0x4(%esp)  #func4的第二个参数
8048d0e:       00
8048d0f:       8b 44 24 18             mov    0x18(%esp),%eax
8048d13:       89 04 24                mov    %eax,(%esp)  #把第一个参数x传进func4
8048d16:       e8 3d ff ff ff          call   8048c58 <func4>  #func4(x,0,14)
```

调用 func4（x，0，14）开始递归

```
08048c58 <func4>:                                       #二叉搜索树
 8048c58:       83 ec 1c                sub    $0x1c,%esp
 8048c5b:       89 5c 24 14             mov    %ebx,0x14(%esp)
 8048c5f:       89 74 24 18             mov    %esi,0x18(%esp)
 8048c63:       8b 44 24 20             mov    0x20(%esp),%eax  #x
 8048c67:       8b 54 24 24             mov    0x24(%esp),%edx  #y=0
 8048c6b:       8b 74 24 28             mov    0x28(%esp),%esi  #f=14
 8048c6f:       89 f1                   mov    %esi,%ecx
 8048c71:       29 d1                   sub    %edx,%ecx  #f-y
 8048c73:       89 cb                   mov    %ecx,%ebx
 8048c75:       c1 eb 1f                shr    $0x1f,%ebx  #逻辑右移31位，取符号位
 8048c78:       01 d9                   add    %ebx,%ecx  #f = f + (f >> 31)
 8048c7a:       d1 f9                   sar    %ecx  #除以2，实现向0取整
 8048c7c:       8d 1c 11                lea    (%ecx,%edx,1),%ebx  #m=y+(f-y)/2
 8048c7f:       39 c3                   cmp    %eax,%ebx              #分支跳转
 8048c81:       7e 17                   jle    8048c9a <func4+0x42>  #递归终点
```

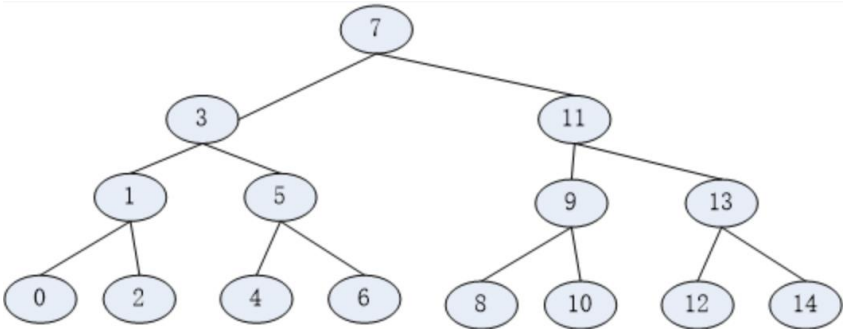
传入三个参数 x，y，f；并求出 m=y+(f-y)/2，其中(f-y)/2 是向零取整；即 m 是 y 和 f 的中间值；之后将 m 与 x 比较，根据不同情况进行递归；

```
                                        #m>x
8048c83:       8d 4b ff                 lea    -0x1(%ebx),%ecx
8048c86:       89 4c 24 08              mov    %ecx,0x8(%esp)  #m-1
8048c8a:       89 54 24 04              mov    %edx,0x4(%esp)  #y
8048c8e:       89 04 24                 mov    %eax,(%esp)     #x
8048c91:       e8 c2 ff ff ff           call   8048c58 <func4> #fun4(x,y,m-1)
8048c96:       01 c3                    add    %eax,%ebx       #返回值与m相加
8048c98:       eb 19                    jmp    8048cb3 <func4+0x5b>

                                        #m<=x
8048c9a:       39 c3                    cmp    %eax,%ebx
8048c9c:       7d 15                    jge    8048cb3 <func4+0x5b>
8048c9e:       89 74 24 08              mov    %esi,0x8(%esp)  #f
8048ca2:       8d 53 01                 lea    0x1(%ebx),%edx
8048ca5:       89 54 24 04              mov    %edx,0x4(%esp)  #m+1
8048ca9:       89 04 24                 mov    %eax,(%esp)     #x
8048cac:       e8 a7 ff ff ff           call   8048c58 <func4> #fun4(x,m+1,f)
8048cb1:       01 c3                    add    %eax,%ebx       #返回值与m相加

8048cb3:       89 d8                    mov    %ebx,%eax
8048cb5:       8b 5c 24 14              mov    0x14(%esp),%ebx
8048cb9:       8b 74 24 18              mov    0x18(%esp),%esi
8048cbd:       83 c4 1c                 add    $0x1c,%esp
8048cc0:       c3                       ret
```

m<=x 时，递归调用 fun4(x,m+1,f)；m>x 时，递归调用 fun4(x,y,m-1)；相当于一个二叉搜索树，返回结果是搜索路径上的所有值之和；

```
8048d1b:       83 f8 0f              cmp    $0xf,%eax
8048d1e:       75 07                 jne    8048d27 <phase_4+0x66>  #要求返回值等于0xf
8048d20:       83 7c 24 1c 0f        cmpl   $0xf,0x1c(%esp)
8048d25:       74 05                 je     8048d2c <phase_4+0x6b>  #要求y=0xf
8048d27:       e8 ca 03 00 00        call   80490f6 <explode_bomb>
8048d2c:       83 c4 2c              add    $0x2c,%esp
8048d2f:       c3                    ret
```

要求返回值是 15，即要求路径上的和为 15，根据传入参数，二叉搜索树如下：



根据计算，x=5 时和为 15，即 7+3+5

5) Phase_5

```
08048d30 <phase_5>:
 8048d30:       53                    push   %ebx
 8048d31:       83 ec 28              sub    $0x28,%esp
 8048d34:       8b 5c 24 30           mov    0x30(%esp),%ebx  #输入的第一个字符位置
 8048d38:       65 a1 14 00 00 00     mov    %gs:0x14,%eax  #gs是一个段寄存器
 8048d3e:       89 44 24 1c           mov    %eax,0x1c(%esp)  #防止数组越界
 8048d42:       31 c0                 xor    %eax,%eax  #ZF=1
 8048d44:       89 1c 24              mov    %ebx,(%esp)
 8048d47:       e8 7f 02 00 00        call   8048fcb <string_length>  #获得字符串长度
 8048d4c:       83 f8 06              cmp    $0x6,%eax
 8048d4f:       74 05                 je     8048d56 <phase_5+0x26>  #要求字符串长度为6
 8048d51:       e8 a0 03 00 00        call   80490f6 <explode_bomb>
```

第一部分要求输入一个长度为 6 的字符串；gs 是一个段寄存器，用于防止数组越界，避免栈溢出；

```
8048d56:       b8 00 00 00 00        mov    $0x0,%eax               #循环begin
8048d5b:       0f be 14 03           movsbl (%ebx,%eax,1),%edx  #第一个字符移至edx
8048d5f:       83 e2 0f              and    $0xf,%edx  #保留末四位
8048d62:       0f b6 92 2c a2 04 08  movzbl 0x804a22c(%edx),%edx  #a[edx]输入的是索引，查此处地址得索引对应值
8048d69:       88 54 04 15           mov    %dl,0x15(%esp,%eax,1)  #将dl的内容送入到esp+eax*1+0x15的地方
8048d6d:       83 c0 01              add    $0x1,%eax
8048d70:       83 f8 06              cmp    $0x6,%eax
8048d73:       75 e6                 jne    8048d5b <phase_5+0x2b>  #循环end
8048d75:       c6 44 24 1b 00        movb   $0x0,0x1b(%esp)
8048d7a:       c7 44 24 04 02 a2 04  movl   $0x804a202,0x4(%esp)  #要得到oidlers，索引为1010 0100 1111 0101
                                                                  0110 0111，即jdoefg
```

开始循环，阅读代码可知输入的 edx 的末四位是一个以 0x804a22c 为首地址的数组的索引；查看后面的 16 个值可知每个索引对应的值：

```
(gdb) x/16cb 0x804a22c
0x804a22c <array.2956>: 109 'm' 97 'a'  100 'd' 117 'u' 105 'i' 101 'e' 114 'r'115 's'
0x804a234 <array.2956+8>:       110 'n' 102 'f' 111 'o' 116 't' 118 'v' 98 'b' 121 'y'  108 'l'
```

```
8048d75:       c6 44 24 1b 00          movb   $0x0,0x1b(%esp)
8048d7a:       c7 44 24 04 02 a2 04    movl   $0x804a202,0x4(%esp)  #要得到oidlers，索引为1010 0100 1111 0101
                                                                    0110 0111，即jdoefg
8048d81:       08
8048d82:       8d 44 24 15             lea    0x15(%esp),%eax  #输入的字符串
8048d86:       89 04 24                mov    %eax,(%esp)
8048d89:       e8 56 02 00 00          call   8048fe4 <strings_not_equal>
8048d8e:       85 c0                   test   %eax,%eax
8048d90:       74 05                   je     8048d97 <phase_5+0x67>  #若字符串相等，跳过bomb
8048d92:       e8 5f 03 00 00          call   80490f6 <explode_bomb>
8048d97:       8b 44 24 1c             mov    0x1c(%esp),%eax
8048d9b:       65 33 05 14 00 00 00    xor    %gs:0x14,%eax
8048da2:       74 05                   je     8048da9 <phase_5+0x79>
8048da4:       e8 27 fa ff ff          call   80487d0 <__stack_chk_fail@plt>  #栈检查失败
8048da9:       83 c4 28                add    $0x28,%esp
8048dac:       5b                      pop    %ebx
8048dad:       c3                      ret
```

读取 0x804a202 处内容得到目标字符串为 oidlers，对应索引为 1010 0100 1111 0101 0110 0111，只要输入的字符二进制末四位对应即可，其中一组为 jdoefg

6) Phase_6

```
08048dae <phase_6>:
 8048dae:       56                      push   %esi
 8048daf:       53                      push   %ebx
 8048db0:       83 ec 44                sub    $0x44,%esp
 8048db3:       8d 44 24 10             lea    0x10(%esp),%eax
 8048db7:       89 44 24 04             mov    %eax,0x4(%esp)
 8048dbb:       8b 44 24 50             mov    0x50(%esp),%eax  #输入的首地址
 8048dbf:       89 04 24                mov    %eax,(%esp)
 8048dc2:       e8 64 04 00 00          call   804922b <read_six_numbers>
 8048dc7:       be 00 00 00 00          mov    $0x0,%esi
 8048dcc:       8b 44 b4 10             mov    0x10(%esp,%esi,4),%eax  #eax=(esp+esi*4+0x10)
 8048dd0:       83 e8 01                sub    $0x1,%eax
 8048dd3:       83 f8 05                cmp    $0x5,%eax
 8048dd6:       76 05                   jbe    8048ddd <phase_6+0x2f>  #判断(esp+esi*4+0x10)是否<=6以及>=1
 8048dd8:       e8 19 03 00 00          call   80490f6 <explode_bomb>
 8048ddd:       83 c6 01                add    $0x1,%esi
 8048de0:       83 fe 06                cmp    $0x6,%esi
 8048de3:       74 35                   je     8048e1a <phase_6+0x6c>  #esi=6跳转
 8048de5:       89 f3                   mov    %esi,%ebx
 8048de7:       8b 44 9c 10             mov    0x10(%esp,%ebx,4),%eax
 8048deb:       39 44 b4 0c             cmp    %eax,0xc(%esp,%esi,4)  #num[ebx]与num[esi+1]如果相等，则引爆炸弹
 8048def:       75 05                   jne    8048df6 <phase_6+0x48>
 8048df1:       e8 00 03 00 00          call   80490f6 <explode_bomb>
 8048df6:       83 c3 01                add    $0x1,%ebx
 8048df9:       83 fb 05                cmp    $0x5,%ebx
 8048dfc:       7e e9                   jle    8048de7 <phase_6+0x39>  #二重循环
```

输入六个数，要求大于 0 小于等于 6，且每个不同，伪代码如下：

```
for (i = 0; i < 6; i++){
    if ((num[i] < 1) || (num[i] > 6)) {
        explode_bomb();
    }
    for (j = i + 1; j < 6; j++){
        if (num[i] == num[j]){
            explode_bomb();
 }}}
```

之后为一个循环，将每个数压入栈中；观察压入栈的内容，每个内容地址实际上是指向 12 字节的一段数据，该数据的末尾又是指向一个地址，因此，可以判断 0x804c174 开始的地方指向的是一个链表，每个节点包括 12 个字节，一个编号值，一个内容值，一个是指向下一个结点的指针；

按照 num[i]的值重新排列链表；判断以上链表是否降序排列，如果是，则拆弹成功；

3 总结

3.1 实验中出现的问题

1) 第一次遇到 gs 指令不知道是什么意思，上网查询后得知是一个段寄存器，一般用来避免栈溢出；
2) 虚拟机无法开机，导致已完成成果全部消失，后来发现是内存不足，只能重新装一个虚拟机从头开始，浪费了很多时间；

3.2 心得体会

1) 对各种代码结构在汇编语言中的表示形式有了直观认识，阅读速度大大加快；
2) 对汇编语言有了更深入的认识，可以熟练掌握运用；
3) 虚拟机容易出问题，要多备份虚拟机；