直接打开bomb.c文件，可以看到每个阶段中代码

```
input = read_line();
```

即通过用户输入正确格式的字符，符合函数要求就可以避免炸弹爆炸。

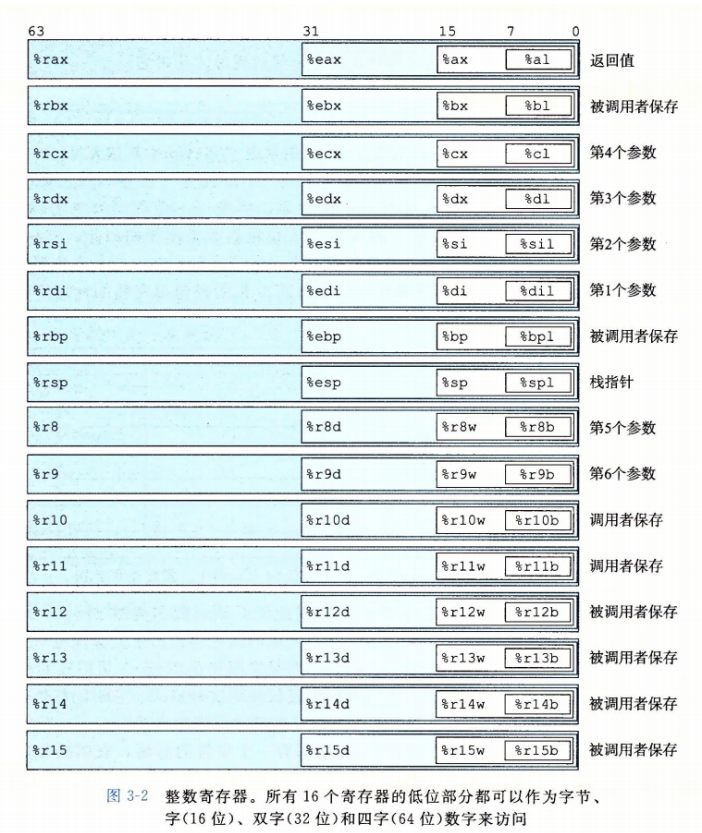


图 3-2 整数寄存器。所有 16 个寄存器的低位部分都可以作为字节、字(16 位)、双字(32 位)和四字(64 位)数字来访问

相关寄存器说明

```
./bomb      //发现无权限
chmod 777 bomb      //赋予权限
./bomb      //重新运行
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
xze

BOOM!!!
The bomb has blown up.
```

phase1

```
gdb bomb    //进入gdb调试
disas phase_1       //反汇编phase_1函数
//结果如下
Dump of assembler code for function phase_1:
   0x0000000000400ee0 <+0>:         sub    $0x8,%rsp
```

```
   0x0000000000400ee4 <+4>:     mov    $0x402400,%esi
   0x0000000000400ee9 <+9>:     callq  0x401338 <strings_not_equal>
   0x0000000000400eee <+14>:    test   %eax,%eax
   0x0000000000400ef0 <+16>:    je     0x400ef7 <phase_1+23>
   0x0000000000400ef2 <+18>:    callq  0x40143a <explode_bomb>
   0x0000000000400ef7 <+23>:    add    $0x8,%rsp
   0x0000000000400efb <+27>:    retq
End of assembler dump.
```

第一步：栈指针减0x8

第二步：将$0x402400地址中的数据移动到%esi

第三步：调用<strings_not_equal>函数

```
(gdb) disas strings_not_equal
Dump of assembler code for function strings_not_equal:
   0x0000000000401338 <+0>:     push   %r12
   0x000000000040133a <+2>:     push   %rbp
   0x000000000040133b <+3>:     push   %rbx
   0x000000000040133c <+4>:     mov    %rdi,%rbx
   0x000000000040133f <+7>:     mov    %rsi,%rbp
   //rdi与rsi为两个参数，假设为x，y，rbx=x，rbp=y
   0x0000000000401342 <+10>:    callq  0x40131b <string_length>
   0x0000000000401347 <+15>:    mov    %eax,%r12d
   //r12d=string_length(x)
   0x000000000040134a <+18>:    mov    %rbp,%rdi
   0x000000000040134d <+21>:    callq  0x40131b <string_length>
   0x0000000000401352 <+26>:    mov    $0x1,%edx
   0x0000000000401357 <+31>:    cmp    %eax,%r12d
   0x000000000040135a <+34>:    jne    0x40139b <strings_not_equal+99>
   //如果string_length(x)!=string_length(y),返回edx，值为1
   0x000000000040135c <+36>:    movzbl (%rbx),%eax
   0x000000000040135f <+39>:    test   %al,%al
   0x0000000000401361 <+41>:    je     0x401388 <strings_not_equal+80>
   //*x==0,返回0
   0x0000000000401363 <+43>:    cmp    0x0(%rbp),%al
   0x0000000000401366 <+46>:    je     0x401372 <strings_not_equal+58>
   //*x==*y，返回0
   0x0000000000401368 <+48>:    jmp    0x40138f <strings_not_equal+87>
   //else return 1
   0x000000000040136a <+50>:    cmp    0x0(%rbp),%al
```

```
   0x000000000040136d <+53>:    nopl   (%rax)

   0x0000000000401370 <+56>:    jne    0x401396 <strings_not_equal+94>
   0x0000000000401372 <+58>:    add    $0x1,%rbx
   0x0000000000401376 <+62>:    add    $0x1,%rbp
   //x++,y++
   0x000000000040137a <+66>:    movzbl (%rbx),%eax
   0x000000000040137d <+69>:    test   %al,%al
   0x000000000040137f <+71>:    jne    0x40136a <strings_not_equal+50>
   0x0000000000401381 <+73>:    mov    $0x0,%edx
   0x0000000000401386 <+78>:    jmp    0x40139b <strings_not_equal+99>
   0x0000000000401388 <+80>:    mov    $0x0,%edx
   0x000000000040138d <+85>:    jmp    0x40139b <strings_not_equal+99>
   0x000000000040138f <+87>:    mov    $0x1,%edx
   0x0000000000401394 <+92>:    jmp    0x40139b <strings_not_equal+99>
   0x0000000000401396 <+94>:    mov    $0x1,%edx
   0x000000000040139b <+99>:    mov    %edx,%eax
   0x000000000040139d <+101>:   pop    %rbx
   0x000000000040139e <+102>:   pop    %rbp
   0x000000000040139f <+103>:   pop    %r12
   0x00000000004013a1 <+105>:   retq
End of assembler dump.
```

string_length反汇编结果如下

```
(gdb) disas string_length
Dump of assembler code for function string_length:
   0x000000000040131b <+0>:     cmpb   $0x0,(%rdi)
   0x000000000040131e <+3>:     je     0x401332 <string_length+23>
    //rdi指向的字节，与0比较，如果为0，直接跳转到+23,返回0
   0x0000000000401320 <+5>:     mov    %rdi,%rdx
   0x0000000000401323 <+8>:     add    $0x1,%rdx
   0x0000000000401327 <+12>:    mov    %edx,%eax
   0x0000000000401329 <+14>:    sub    %edi,%eax
   0x000000000040132b <+16>:    cmpb   $0x0,(%rdx)
   0x000000000040132e <+19>:    jne    0x401323 <string_length+8>
   0x0000000000401330 <+21>:    repz retq
   0x0000000000401332 <+23>:    mov    $0x0,%eax
   0x0000000000401337 <+28>:    retq
   /*
```

```
16    int string_length(char *x)      //求字符串长度
17     {
18            if (*x==0)
19                    return 0;
20            x_begin=x;
21            else
22            {
23                    do
24                    {
25                            x++;
26                    }while(*x !=0)
27            }
28            return (x-xbegin);
29     }
30  */
31  End of assembler dump.
```

第四步：test用法

```
1  test %eax, %eax             //test是一个与运算，影响标志位，测试eax是否为0
2  jz    zeroLabel     ; jump if EAX is zero
3  js    negLabel      ; jump if EAX is negative
4  jns   posLabel      ; jump if EAX is positive
5  //je 指令Jump if Equals在ZF被置位时跳转。je 是 jz Jump if Zero的别名。
```

即此步骤是根据strings_not_equal的返回值决定跳转，如果为0，则进入下一阶段，否则爆炸

故尝试查看内存位置0x402400的内容，查询gdb手册可得

```
1  x/s 0xbffff890 //Examine a string stored at 0xbffff890
2  (gdb) x/s 0x402400
3  0x402400:  " "
```

输入此串，成功进入下一阶段：

```
[root@localhost bomb]# ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
```

phase2

反汇编phase_2

```
1  Dump of assembler code for function phase_2:
```

```
   0x0000000000400efc <+0>:     push   %rbp
   0x0000000000400efd <+1>:     push   %rbx
   0x0000000000400efe <+2>:     sub    $0x28,%rsp
   0x0000000000400f02 <+6>:     mov    %rsp,%rsi
   0x0000000000400f05 <+9>:     callq  0x40145c <read_six_numbers>
   0x0000000000400f0a <+14>:    cmpl   $0x1,(%rsp)//第一个数必须为1
   0x0000000000400f0e <+18>:    je     0x400f30 <phase_2+52>
   0x0000000000400f10 <+20>:    callq  0x40143a <explode_bomb>
   //rsp若为1，转到+52位置，不转则爆炸
   0x0000000000400f15 <+25>:    jmp    0x400f30 <phase_2+52>
   0x0000000000400f17 <+27>:    mov    -0x4(%rbx),%eax
   0x0000000000400f1a <+30>:    add    %eax,%eax//eax*=2;
   0x0000000000400f1c <+32>:    cmp    %eax,(%rbx)//判断接下来的是否等于二倍
   0x0000000000400f1e <+34>:    je     0x400f25 <phase_2+41>
   0x0000000000400f20 <+36>:    callq  0x40143a <explode_bomb>
   //rbx值若等于eax，转到+41位置，不转则爆炸
   0x0000000000400f25 <+41>:    add    $0x4,%rbx
   0x0000000000400f29 <+45>:    cmp    %rbp,%rbx
   0x0000000000400f2c <+48>:    jne    0x400f17 <phase_2+27>
   0x0000000000400f2e <+50>:    jmp    0x400f3c <phase_2+64>
   //如果rbx!=rbp,注意为jne，跳到+27，否则跳到+64
   0x0000000000400f30 <+52>:    lea    0x4(%rsp),%rbx
   0x0000000000400f35 <+57>:    lea    0x18(%rsp),%rbp
   //lea是将前面的偏移地址移动到后面
   0x0000000000400f3a <+62>:    jmp    0x400f17 <phase_2+27>
   //跳到+27
   0x0000000000400f3c <+64>:    add    $0x28,%rsp
   0x0000000000400f40 <+68>:    pop    %rbx
   0x0000000000400f41 <+69>:    pop    %rbp
   0x0000000000400f42 <+70>:    retq
End of assembler dump.
```

忽略初始化过程，第六行调用read_six_numbers，猜测是输入6个数，反汇编结果如下

```
(gdb) disas read_six_numbers
Dump of assembler code for function read_six_numbers:
   0x000000000040145c <+0>:     sub    $0x18,%rsp
   0x0000000000401460 <+4>:     mov    %rsi,%rdx
   0x0000000000401463 <+7>:     lea    0x4(%rsi),%rcx
   0x0000000000401467 <+11>:    lea    0x14(%rsi),%rax
```

```
   0x000000000040146b <+15>:    mov    %rax,0x8(%rsp)
   0x0000000000401470 <+20>:    lea    0x10(%rsi),%rax
   0x0000000000401474 <+24>:    mov    %rax,(%rsp)
   0x0000000000401478 <+28>:    lea    0xc(%rsi),%r9
   0x000000000040147c <+32>:    lea    0x8(%rsi),%r8
   0x0000000000401480 <+36>:    mov    $0x4025c3,%esi
   0x0000000000401485 <+41>:    mov    $0x0,%eax
   0x000000000040148a <+46>:    callq  0x400bf0 <__isoc99_sscanf@plt>
   0x000000000040148f <+51>:    cmp    $0x5,%eax
   0x0000000000401492 <+54>:    jg     0x401499 <read_six_numbers+61>
   0x0000000000401494 <+56>:    callq  0x40143a <explode_bomb>
   0x0000000000401499 <+61>:    add    $0x18,%rsp
   0x000000000040149d <+65>:    retq
End of assembler dump.
```

所以第一个数字必须为1，之后保证下一个数字是前一个的2倍

```
[root@localhost bomb]# ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
```

phase_3

反汇编

```
(gdb) disas phase_3
Dump of assembler code for function phase_3:
   0x0000000000400f43 <+0>:     sub    $0x18,%rsp
   0x0000000000400f47 <+4>:     lea    0xc(%rsp),%rcx
   0x0000000000400f4c <+9>:     lea    0x8(%rsp),%rdx
   0x0000000000400f51 <+14>:    mov    $0x4025cf,%esi
   //把0x4025cf的值传到esi，查看esi内容
   //(gdb) x/1s 0x4025cf
   //0x4025cf:     "%d %d" 为两个int型

   0x0000000000400f56 <+19>:    mov    $0x0,%eax
```

```
   0x0000000000400f5b <+24>:    callq  0x400bf0 <__isoc99_sscanf@plt>
   0x0000000000400f60 <+29>:    cmp    $0x1,%eax
   0x0000000000400f63 <+32>:    jg     0x400f6a <phase_3+39>
   //sscanf的返回值大于1则跳到+39，否则接下来爆炸
   0x0000000000400f65 <+34>:    callq  0x40143a <explode_bomb>
   0x0000000000400f6a <+39>:    cmpl   $0x7,0x8(%rsp)
   0x0000000000400f6f <+44>:    ja     0x400fad <phase_3+106>
   //大于7跳转到+106，故第一个数字小于等于7
   0x0000000000400f71 <+46>:    mov    0x8(%rsp),%eax
   0x0000000000400f75 <+50>:    jmpq   *0x402470(,%rax,8)
   //(gdb) x/a 0x402470
   //0x402470:     0x400f7c <phase_3+57>
   //类似于case结构

   0x0000000000400f7c <+57>:    mov    $0xcf,%eax
   0x0000000000400f81 <+62>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f83 <+64>:    mov    $0x2c3,%eax
   0x0000000000400f88 <+69>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f8a <+71>:    mov    $0x100,%eax
   0x0000000000400f8f <+76>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f91 <+78>:    mov    $0x185,%eax
   0x0000000000400f96 <+83>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f98 <+85>:    mov    $0xce,%eax
   0x0000000000400f9d <+90>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f9f <+92>:    mov    $0x2aa,%eax
   0x0000000000400fa4 <+97>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400fa6 <+99>:    mov    $0x147,%eax
   0x0000000000400fab <+104>:   jmp    0x400fbe <phase_3+123>
   //对于0到6的case
   0x0000000000400fad <+106>:   callq  0x40143a <explode_bomb>
   0x0000000000400fb2 <+111>:   mov    $0x0,%eax
   0x0000000000400fb7 <+116>:   jmp    0x400fbe <phase_3+123>
   0x0000000000400fb9 <+118>:   mov    $0x137,%eax
   0x0000000000400fbe <+123>:   cmp    0xc(%rsp),%eax
   0x0000000000400fc2 <+127>:   je     0x400fc9 <phase_3+134>
   0x0000000000400fc4 <+129>:   callq  0x40143a <explode_bomb>
   0x0000000000400fc9 <+134>:   add    $0x18,%rsp
   0x0000000000400fcd <+138>:   retq
End of assembler dump.
```

输入第一个小于7整数，然后查看*0x402470(,%rax,8)处的值对应位置

```
(gdb) x/x 0x402470
0x402470:  0x00400f7c
(gdb) x/x 0x402478
0x402478:  0x00400fb9
(gdb) x/x 0x402480
0x402480:  0x00400f83
(gdb) x/x 0x402488
0x402488:  0x00400f8a
(gdb) x/x 0x402490
0x402490:  0x00400f91
(gdb) x/x 0x402498
0x402498:  0x00400f98
(gdb) x/x 0x4024a0
0x4024a0:  0x00400f9f
(gdb) x/x 0x4024a8
0x4024a8:  0x00400fa6
```

故答案如下：

(0，207)、(1，311)、(2，707)、(3，256)、(4，389)、(5，206)、(6，682)、(7，327)

```
[root@localhost bomb]# ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
2 707
Halfway there!
```

phase_4

```
(gdb) disas phase_4
Dump of assembler code for function phase_4:
   0x000000000040100c <+0>:     sub    $0x18,%rsp
```

```
   0x0000000000401010 <+4>:     lea    0xc(%rsp),%rcx
   0x0000000000401015 <+9>:     lea    0x8(%rsp),%rdx
   0x000000000040101a <+14>:    mov    $0x4025cf,%esi
   //int1=>rdx,int2=>rcx,esi=0x4025cf
   0x000000000040101f <+19>:    mov    $0x0,%eax
   0x0000000000401024 <+24>:    callq  0x400bf0 <__isoc99_sscanf@plt>
   0x0000000000401029 <+29>:    cmp    $0x2,%eax
   0x000000000040102c <+32>:    jne    0x401035 <phase_4+41>
   //如果sscanf返回值不为2，爆炸，
   0x000000000040102e <+34>:    cmpl   $0xe,0x8(%rsp)
   //第一个参数小于等于14则跳转到+46，否则爆炸
   0x0000000000401033 <+39>:    jbe    0x40103a <phase_4+46>
   0x0000000000401035 <+41>:    callq  0x40143a <explode_bomb>
   0x000000000040103a <+46>:    mov    $0xe,%edx
   0x000000000040103f <+51>:    mov    $0x0,%esi
   0x0000000000401044 <+56>:    mov    0x8(%rsp),%edi
   //edx=14，esi=0，edi=int1
   0x0000000000401048 <+60>:    callq  0x400fce <func4>
   //调用func4
   0x000000000040104d <+65>:    test   %eax,%eax
   0x000000000040104f <+67>:    jne    0x401058 <phase_4+76>//保证返回值为0
   0x0000000000401051 <+69>:    cmpl   $0x0,0xc(%rsp)
   0x0000000000401056 <+74>:    je     0x40105d <phase_4+81>
   0x0000000000401058 <+76>:    callq  0x40143a <explode_bomb>
   //保证func4的返回值为0，并且int2的值为0,可以直接穷举
   0x000000000040105d <+81>:    add    $0x18,%rsp
   0x0000000000401061 <+85>:    retq
End of assembler dump.
```

func4

易得这是一个递归函数

```
(gdb) disas func4
Dump of assembler code for function func4:
//func4(a=int1,b=0,c=0xe)
//a,b,c分别存于edi,esi,edx
   0x0000000000400fce <+0>:     sub    $0x8,%rsp//初始化
   0x0000000000400fd2 <+4>:     mov    %edx,%eax//
   0x0000000000400fd4 <+6>:     sub    %esi,%eax//eax=eax-esi p129和通常汇编不一样？
   0x0000000000400fd6 <+8>:     mov    %eax,%ecx//ecx=c
```

```
   0x0000000000400fd8 <+10>:    shr    $0x1f,%ecx//操作数逻辑右移31位，ecx储存符号位
   0x0000000000400fdb <+13>:    add    %ecx,%eax//ecx+=eax>>0x1f  if (eax < 0) eax -= 1;
   0x0000000000400fdd <+15>:    sar    %eax//算数右移，即除以2，eax/=2
   0x0000000000400fdf <+17>:    lea    (%rax,%rsi,1),%ecx//ecx=eax+esi
   0x0000000000400fe2 <+20>:    cmp    %edi,%ecx//比较int1和ecx的值
   0x0000000000400fe4 <+22>:    jle    0x400ff2 <func4+36>//
   //int1<=ecx则跳转，
   0x0000000000400fe6 <+24>:    lea    -0x1(%rcx),%edx
   0x0000000000400fe9 <+27>:    callq  0x400fce <func4>//如果大于
   0x0000000000400fee <+32>:    add    %eax,%eax
   0x0000000000400ff0 <+34>:    jmp    0x401007 <func4+57>
   0x0000000000400ff2 <+36>:    mov    $0x0,%eax
   0x0000000000400ff7 <+41>:    cmp    %edi,%ecx
   0x0000000000400ff9 <+43>:    jge    0x401007 <func4+57>
   0x0000000000400ffb <+45>:    lea    0x1(%rcx),%esi
   0x0000000000400ffe <+48>:    callq  0x400fce <func4>
   0x0000000000401003 <+53>:    lea    0x1(%rax,%rax,1),%eax
   0x0000000000401007 <+57>:    add    $0x8,%rsp
   0x000000000040100b <+61>:    retq
End of assembler dump.
```

```
int func4(int edi, int esi, int edx)
{int1 0 14
    int eax = edx - esi;
    if (eax < 0)
      eax -= 1;
    eax /= 2;

    int ecx = eax + esi;
    if (ecx > edi)
        return func4(edi, esi, ecx - 1) * 2;
    else if (ecx < edi)
        return func4(edi, ecx + 1, edx) * 2 + 1;
    else
        return 0;
}
```

```
[root@localhost bomb]# ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
1 2 4 8 16 32
2 707
7 0
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
```

phase_5

```
(gdb) disas phase_5
Dump of assembler code for function phase_5:
   0x0000000000401062 <+0>:     push   %rbx
   0x0000000000401063 <+1>:     sub    $0x20,%rsp
   0x0000000000401067 <+5>:     mov    %rdi,%rbx
   0x000000000040106a <+8>:     mov    %fs:0x28,%rax
   0x0000000000401073 <+17>:    mov    %rax,0x18(%rsp)
   0x0000000000401078 <+22>:    xor    %eax,%eax
   0x000000000040107a <+24>:    callq  0x40131b <string_length>
   0x000000000040107f <+29>:    cmp    $0x6,%eax
   0x0000000000401082 <+32>:    je     0x4010d2 <phase_5+112>
   0x0000000000401084 <+34>:    callq  0x40143a <explode_bomb>
   //调用string_length并要求返回值为6，否则爆炸，应该是长度为6的字符
   0x0000000000401089 <+39>:    jmp    0x4010d2 <phase_5+112>
   //此处开始循环 rax=0,rbx被rdi初始化
   0x000000000040108b <+41>:    movzbl (%rbx,%rax,1),%ecx//字符串首字符传入ecx
   0x000000000040108f <+45>:    mov    %cl,(%rsp)//刚刚的字符压入栈顶
   0x0000000000401092 <+48>:    mov    (%rsp),%rdx//rdx=rsp
   0x0000000000401096 <+52>:    and    $0xf,%edx//edx处字符高四位设为0
   0x0000000000401099 <+55>:    movzbl 0x4024b0(%rdx),%edx
   // 以0x4024b0这一特定地址为基址，以传入的字符为下标找到该地址下的1字节
   0x00000000004010a0 <+62>:    mov    %dl,0x10(%rsp,%rax,1)
```

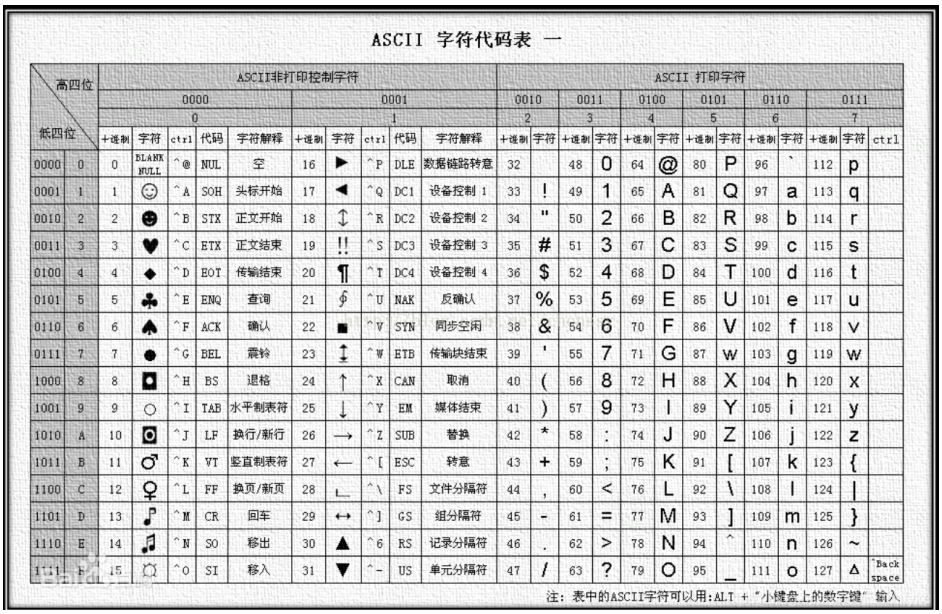
```
   //将该字符存于 rsp+0x10+rax 地址中
   0x00000000004010a4 <+66>:       add    $0x1,%rax
   //rax++
   0x00000000004010a8 <+70>:       cmp    $0x6,%rax
   0x00000000004010ac <+74>:       jne    0x40108b <phase_5+41>
   //如果rax<6，重复此循环
   // 这段循环用C代码表示为:
   //for (int i=0;i<6;i++){
   //      a[i] = src(input[i] & 0xf);
   //}


   //其中a是rsp+0x10为地址的字符串，src=0x4024b0，input为phase_5的输入参数，
   //取输入字母的低4位当索引

   0x00000000004010ae <+76>:       movb   $0x0,0x16(%rsp)//(rsp+24)=0
   0x00000000004010b3 <+81>:       mov    $0x40245e,%esi//参数2esi=0x40245e，转换成上面字
   0x00000000004010b8 <+86>:       lea    0x10(%rsp),%rdi//参数1=数组a首元素地址
   0x00000000004010bd <+91>:       callq  0x401338 <strings_not_equal>

   0x00000000004010c2 <+96>:       test   %eax,%eax
   0x00000000004010c4 <+98>:       je     0x4010d9 <phase_5+119>
   0x00000000004010c6 <+100>:      callq  0x40143a <explode_bomb>//这两个字符串必须相等
   0x00000000004010cb <+105>:      nopl   0x0(%rax,%rax,1)
   0x00000000004010d0 <+110>:      jmp    0x4010d9 <phase_5+119>

   0x00000000004010d2 <+112>:      mov    $0x0,%eax
   0x00000000004010d7 <+117>:      jmp    0x40108b <phase_5+41>

   0x00000000004010d9 <+119>:      mov    0x18(%rsp),%rax
   0x00000000004010de <+124>:      xor    %fs:0x28,%rax
   0x00000000004010e7 <+133>:      je     0x4010ee <phase_5+140>
   0x00000000004010e9 <+135>:      callq  0x400b30 <__stack_chk_fail@plt>
   0x00000000004010ee <+140>:      add    $0x20,%rsp
   0x00000000004010f2 <+144>:      pop    %rbx
   0x00000000004010f3 <+145>:      retq
End of assembler dump.
```

查询两个特殊地址的值

```
(gdb) x/1s 0x4024b0
0x4024b0 <array.3449>:     "maduiersnfotvbylSo you think you can stop the bomb with ctrl
(gdb) x/1s 0x40245e
0x40245e:  "flyers"
```

字符串a等于flyers，所以 输入字符的低四位必须为0x09、0x0f，0x0e，0x05，0x06和0x07，以对应第一个字符

IONEFG



ASCII 字符代码表 一

高四位 / 低四位		ASCII非打印控制字符 0000 0					ASCII非打印控制字符 0001 1					ASCII 打印字符 0010 2		ASCII 打印字符 0011 3		ASCII 打印字符 0100 4		ASCII 打印字符 0101 5		ASCII 打印字符 0110 6		ASCII 打印字符 0111 7		
		十进制	字符	ctrl	代码	字符解释	十进制	字符	ctrl	代码	字符解释	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	ctrl
0000	0	0	BLANK NULL	^@	NUL	空	16	▶	^P	DLE	数据链路转意	32		48	0	64	@	80	P	96	`	112	p	
0001	1	1	☺	^A	SOH	头标开始	17	◀	^Q	DC1	设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q	
0010	2	2	☻	^B	STX	正文开始	18	↕	^R	DC2	设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r	
0011	3	3	♥	^C	ETX	正文结束	19	!!	^S	DC3	设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s	
0100	4	4	◆	^D	EOT	传输结束	20	¶	^T	DC4	设备控制 4	36	$	52	4	68	D	84	T	100	d	116	t	
0101	5	5	♣	^E	ENQ	查询	21	§	^U	NAK	反确认	37	%	53	5	69	E	85	U	101	e	117	u	
0110	6	6	♠	^F	ACK	确认	22	■	^V	SYN	同步空闲	38	&	54	6	70	F	86	V	102	f	118	v	
0111	7	7	•	^G	BEL	震铃	23	↨	^W	ETB	传输块结束	39	'	55	7	71	G	87	W	103	g	119	w	
1000	8	8	◘	^H	BS	退格	24	↑	^X	CAN	取消	40	(	56	8	72	H	88	X	104	h	120	x	
1001	9	9	○	^I	TAB	水平制表符	25	↓	^Y	EM	媒体结束	41	)	57	9	73	I	89	Y	105	i	121	y	
1010	A	10	◙	^J	LF	换行/新行	26	→	^Z	SUB	替换	42	*	58	:	74	J	90	Z	106	j	122	z	
1011	B	11	♂	^K	VT	竖直制表符	27	←	^[	ESC	转意	43	+	59	;	75	K	91	[	107	k	123	{	
1100	C	12	♀	^L	FF	换页/新页	28	∟	^\	FS	文件分隔符	44	,	60	<	76	L	92	\	108	l	124	|	
1101	D	13	♪	^M	CR	回车	29	↔	^]	GS	组分隔符	45	-	61	=	77	M	93	]	109	m	125	}	
1110	E	14	♫	^N	SO	移出	30	▲	^6	RS	记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~	
1111	F	15	☼	^O	SI	移入	31	▼	^-	US	单元分隔符	47	/	63	?	79	O	95	_	111	o	127	Δ	Back space

注：表中的ASCII字符可以用：ALT + "小键盘上的数字键" 输入

```
[root@localhost bomb]# ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
1 2 4 8 16 32
2 707
7 0
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
IONEFG
Good work!  On to the next...
```

测试通过

phase_6

```
Dump of assembler code for function phase_6:
   0x00000000004010f4 <+0>:     push   %r14
   0x00000000004010f6 <+2>:     push   %r13
   0x00000000004010f8 <+4>:     push   %r12
   0x00000000004010fa <+6>:     push   %rbp
   0x00000000004010fb <+7>:     push   %rbx
   0x00000000004010fc <+8>:     sub    $0x50,%rsp
   0x0000000000401100 <+12>:    mov    %rsp,%r13
   0x0000000000401103 <+15>:    mov    %rsp,%rsi
   0x0000000000401106 <+18>:    callq  0x40145c <read_six_numbers>
   //读入6个数字

   0x000000000040110b <+23>:    mov    %rsp,%r14
   0x000000000040110e <+26>:    mov    $0x0,%r12d
   //令r12d=i,rbx=j,j=0
   0x0000000000401114 <+32>:    mov    %r13,%rbp
   0x0000000000401117 <+35>:    mov    0x0(%r13),%eax//eax=a[i]
   0x000000000040111b <+39>:    sub    $0x1,%eax//eax-=1
   0x000000000040111e <+42>:    cmp    $0x5,%eax
   0x0000000000401121 <+45>:    jbe    0x401128 <phase_6+52>
   0x0000000000401123 <+47>:    callq  0x40143a <explode_bomb>
   //如果i<=5，则继续执行，否则爆炸
   0x0000000000401128 <+52>:    add    $0x1,%r12d//i++
   0x000000000040112c <+56>:    cmp    $0x6,%r12d
   0x0000000000401130 <+60>:    je     0x401153 <phase_6+95>//i=6时循环结束
   0x0000000000401132 <+62>:    mov    %r12d,%ebx//否则j=i

   0x0000000000401135 <+65>:    movslq %ebx,%rax
   0x0000000000401138 <+68>:    mov    (%rsp,%rax,4),%eax//取出下一个数保存到eax
   0x000000000040113b <+71>:    cmp    %eax,0x0(%rbp)
   0x000000000040113e <+74>:    jne    0x401145 <phase_6+81>
   0x0000000000401140 <+76>:    callq  0x40143a <explode_bomb>
   //如果相等，则爆炸，不等则继续
   0x0000000000401145 <+81>:    add    $0x1,%ebx//ebx++
   0x0000000000401148 <+84>:    cmp    $0x5,%ebx
   0x000000000040114b <+87>:    jle    0x401135 <phase_6+65>
   //如果ebx<=5,则跳转到+65，继续循环

   //因为eax-1<=5，故a[i]去小于等于6的不重复值
   0x000000000040114d <+89>:    add    $0x4,%r13
```

```
0x0000000000401151 <+93>:    jmp    0x401114 <phase_6+32>
//上面的大循环结束

0x0000000000401153 <+95>:    lea    0x18(%rsp),%rsi//取a[6]地址作为下面循环结束条件
0x0000000000401158 <+100>:   mov    %r14,%rax//rax=&a[0]
0x000000000040115b <+103>:   mov    $0x7,%ecx//ecx=7

0x0000000000401160 <+108>:   mov    %ecx,%edx
0x0000000000401162 <+110>:   sub    (%rax),%edx//edx=7-a[k]
0x0000000000401164 <+112>:   mov    %edx,(%rax)//a[k]=7-a[k]
0x0000000000401166 <+114>:   add    $0x4,%rax//++k
0x000000000040116a <+118>:   cmp    %rsi,%rax//如果没有到a[6]，则继续
0x000000000040116d <+121>:   jne    0x401160 <phase_6+108>
//即此循环将a[i]=7-a[i]

0x000000000040116f <+123>:   mov    $0x0,%esi//esi=0;
0x0000000000401174 <+128>:   jmp    0x401197 <phase_6+163>


0x0000000000401176 <+130>:   mov    0x8(%rdx),%rdx
//应该是偏移0x8来移动到下一个位置
0x000000000040117a <+134>:   add    $0x1,%eax//eax++
0x000000000040117d <+137>:   cmp    %ecx,%eax//ecx初始值为7
0x000000000040117f <+139>:   jne    0x401176 <phase_6+130>//不为7则循环到+130
0x0000000000401181 <+141>:   jmp    0x401188 <phase_6+148>
//接下来看6032d0
0x0000000000401183 <+143>:   mov    $0x6032d0,%edx
0x0000000000401188 <+148>:   mov    %rdx,0x20(%rsp,%rsi,2)//即将链表的地址赋给rsp+2r
0x000000000040118d <+153>:   add    $0x4,%rsi//rsi+4
0x0000000000401191 <+157>:   cmp    $0x18,%rsi//24,保证循环6次
0x0000000000401195 <+161>:   je     0x4011ab <phase_6+183>//6次后跳到+183
0x0000000000401197 <+163>:   mov    (%rsp,%rsi,1),%ecx//获取链表中6个数
0x000000000040119a <+166>:   cmp    $0x1,%ecx
0x000000000040119d <+169>:   jle    0x401183 <phase_6+143>//ecx<=1跳到开头
0x000000000040119f <+171>:   mov    $0x1,%eax
0x00000000004011a4 <+176>:   mov    $0x6032d0,%edx
0x00000000004011a9 <+181>:   jmp    0x401176 <phase_6+130>
//即将node的6个数据放入sp+0x20的指针数组中
0x00000000004011ab <+183>:   mov    0x20(%rsp),%rbx//rbx=b[0]
0x00000000004011b0 <+188>:   lea    0x28(%rsp),%rax//rax=&b[1]
```

```
   0x00000000004011b5 <+193>:   lea    0x50(%rsp),%rsi//记录结尾b[6]位置，作为判断条件
   0x00000000004011ba <+198>:   mov    %rbx,%rcx//rcx=rbx=b[0]
   0x00000000004011bd <+201>:   mov    (%rax),%rdx//rdx=b[1]
   0x00000000004011c0 <+204>:   mov    %rdx,0x8(%rcx)//b[0]->next=b[1]
   0x00000000004011c4 <+208>:   add    $0x8,%rax//rax++,故rax应该是rcx的下一个值
   0x00000000004011c8 <+212>:   cmp    %rsi,%rax
   0x00000000004011cb <+215>:   je     0x4011d2 <phase_6+222>
   //rax到6退出循环，否则rcx=rdx,回到循环
   0x00000000004011cd <+217>:   mov    %rdx,%rcx
   0x00000000004011d0 <+220>:   jmp    0x4011bd <phase_6+201>
   //此部分应该是将链表串起来
   0x00000000004011d2 <+222>:   movq   $0x0,0x8(%rdx)
   0x00000000004011da <+230>:   mov    $0x5,%ebp
   0x00000000004011df <+235>:   mov    0x8(%rbx),%rax//rax指向rbx下一指针
   0x00000000004011e3 <+239>:   mov    (%rax),%eax//取出rax的值与eax比较
   0x00000000004011e5 <+241>:   cmp    %eax,(%rbx)
   0x00000000004011e7 <+243>:   jge    0x4011ee <phase_6+250>
   0x00000000004011e9 <+245>:   callq  0x40143a <explode_bomb>
   //比较相邻两个结点值的大小，前面的<后面的，爆炸
   0x00000000004011ee <+250>:   mov    0x8(%rbx),%rbx
   0x00000000004011f2 <+254>:   sub    $0x1,%ebp
   //循环
   0x00000000004011f5 <+257>:   jne    0x4011df <phase_6+235>

   0x00000000004011f7 <+259>:   add    $0x50,%rsp
   0x00000000004011fb <+263>:   pop    %rbx
   0x00000000004011fc <+264>:   pop    %rbp
   0x00000000004011fd <+265>:   pop    %r12
   0x00000000004011ff <+267>:   pop    %r13
   0x0000000000401201 <+269>:   pop    %r14
   0x0000000000401203 <+271>:   retq
End of assembler dump.
```

查看涉及地址内容

```
(gdb) x 0x6032d0
0x6032d0 <node1>:  0x0000014c//推测此处应该是一个struct结构

 (gdb) x/32w 0x6032d0
0x6032d0 <node1>:  0x0000014c      0x00000001      0x006032e0      0x00000000
```

```
0x6032e0 <node2>:  0x000000a8      0x00000002      0x006032f0      0x00000000
0x6032f0 <node3>:  0x0000039c      0x00000003      0x00603300      0x00000000
0x603300 <node4>:  0x000002b3      0x00000004      0x00603310      0x00000000
0x603310 <node5>:  0x000001dd      0x00000005      0x00603320      0x00000000
0x603320 <node6>:  0x000001bb      0x00000006      0x00000000      0x00000000
0x603330:  0x00000000      0x00000000      0x00000000      0x00000000//应该从此处停止
0x603340 <host_table>:     0x00402629      0x00000000      0x00402643      0x00000000
//观察可得第二块为编号，第三块为下一块的地址，类似于链表
```

所以是将上述结构按照降序排列，3，4，5，6，1，2，但是这个结果是7-a[i]，所以是4 3 2 1 6 5

```
[xze@localhost bomb]$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
2 707
Halfway there!
7 0
So you got that one.  Try this one.
IONEFG
Good work!  On to the next...
4 3 2 1 6 5
Congratulations! You've defused the bomb!
```