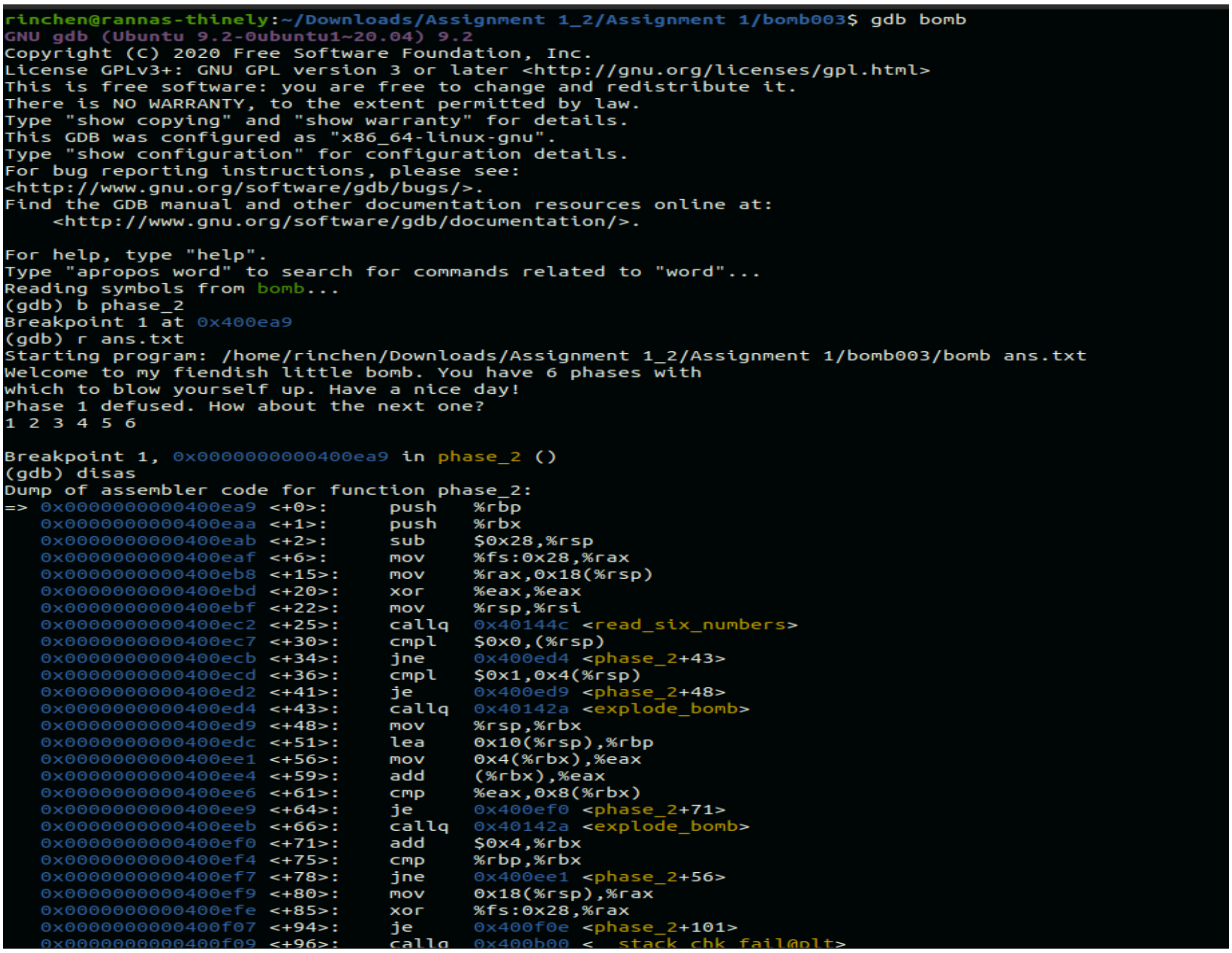
Bomb_3/Phase_2

second phase is about giving a set of six inputs. The inputs were found to be “0 1 1 2 3 5” which were the first 6 Fibonacci series numbers. After putting these six digit number in the phase_2 of bomb 003, the bomb diffuses. Before that let us saved our string answer of phase in a text file so that we need have to look back again and again.



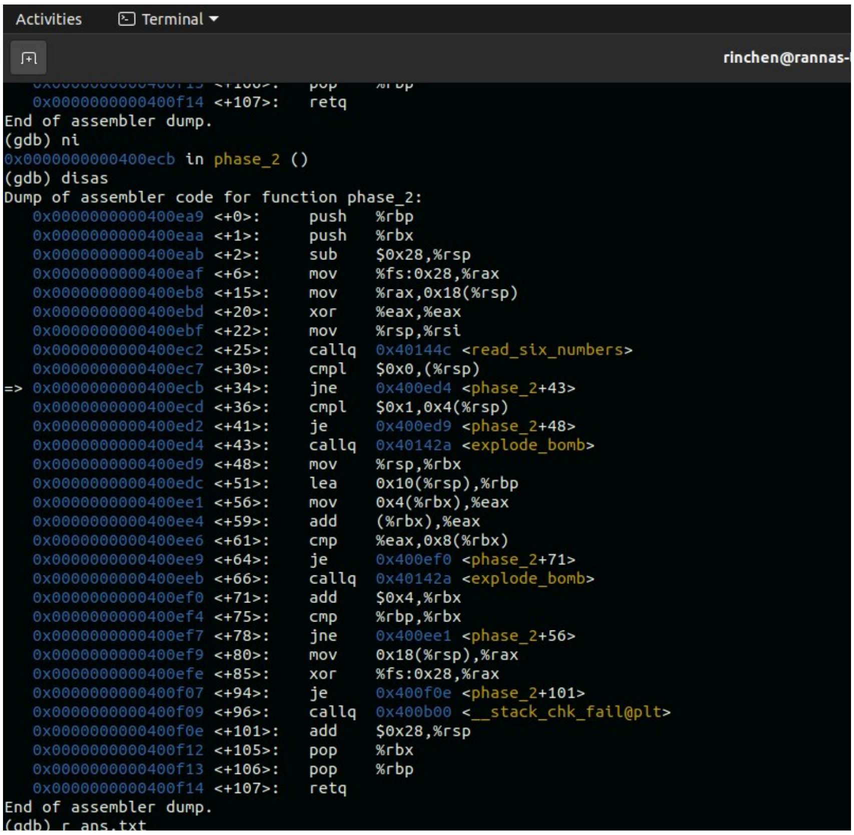
```
rinchen@rannas-thinely:~/Downloads/Assignment 1_2/Assignment 1/bomb003$ gdb bomb
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...
(gdb) b phase_2
Breakpoint 1 at 0x400ea9
(gdb) r ans.txt
Starting program: /home/rinchen/Downloads/Assignment 1_2/Assignment 1/bomb003/bomb ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
1 2 3 4 5 6

Breakpoint 1, 0x0000000000400ea9 in phase_2 ()
(gdb) disas
Dump of assembler code for function phase_2:
=> 0x0000000000400ea9 <+0>:     push   %rbp
   0x0000000000400eaa <+1>:     push   %rbx
   0x0000000000400eab <+2>:     sub    $0x28,%rsp
   0x0000000000400eaf <+6>:     mov    %fs:0x28,%rax
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    callq  0x40144c <read_six_numbers>
   0x0000000000400ec7 <+30>:    cmpl   $0x0,(%rsp)
   0x0000000000400ecb <+34>:    jne    0x400ed4 <phase_2+43>
   0x0000000000400ecd <+36>:    cmpl   $0x1,0x4(%rsp)
   0x0000000000400ed2 <+41>:    je     0x400ed9 <phase_2+48>
   0x0000000000400ed4 <+43>:    callq  0x40142a <explode_bomb>
   0x0000000000400ed9 <+48>:    mov    %rsp,%rbx
   0x0000000000400edc <+51>:    lea    0x10(%rsp),%rbp
   0x0000000000400ee1 <+56>:    mov    0x4(%rbx),%eax
   0x0000000000400ee4 <+59>:    add    (%rbx),%eax
   0x0000000000400ee6 <+61>:    cmp    %eax,0x8(%rbx)
   0x0000000000400ee9 <+64>:    je     0x400ef0 <phase_2+71>
   0x0000000000400eeb <+66>:    callq  0x40142a <explode_bomb>
   0x0000000000400ef0 <+71>:    add    $0x4,%rbx
   0x0000000000400ef4 <+75>:    cmp    %rbp,%rbx
   0x0000000000400ef7 <+78>:    jne    0x400ee1 <phase_2+56>
   0x0000000000400ef9 <+80>:    mov    0x18(%rsp),%rax
   0x0000000000400efe <+85>:    xor    %fs:0x28,%rax
   0x0000000000400f07 <+94>:    je     0x400f0e <phase_2+101>
   0x0000000000400f09 <+96>:    callq  0x400b00 <__stack_chk_fail@plt>
```

as we can see in the above picture, we need to get into the gdb and defuse the phase one bomb and after entering any random integers, now we can get into the assembly code of phase using disas command.

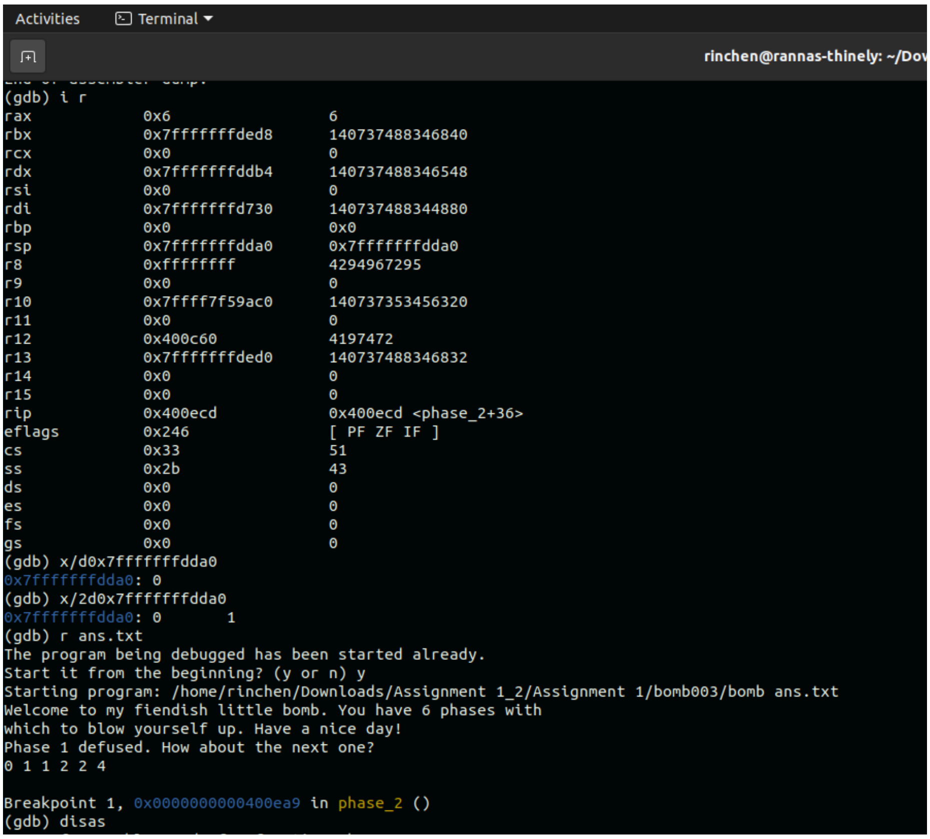
Bomb_3/Phase_2



```
   0x0000000000400f13 <+106>:   pop    %rbp
   0x0000000000400f14 <+107>:   retq
End of assembler dump.
(gdb) ni
0x0000000000400ecb in phase_2 ()
(gdb) disas
Dump of assembler code for function phase_2:
   0x0000000000400ea9 <+0>:     push   %rbp
   0x0000000000400eaa <+1>:     push   %rbx
   0x0000000000400eab <+2>:     sub    $0x28,%rsp
   0x0000000000400eaf <+6>:     mov    %fs:0x28,%rax
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    callq  0x40144c <read_six_numbers>
   0x0000000000400ec7 <+30>:    cmpl   $0x0,(%rsp)
=> 0x0000000000400ecb <+34>:    jne    0x400ed4 <phase_2+43>
   0x0000000000400ecd <+36>:    cmpl   $0x1,0x4(%rsp)
   0x0000000000400ed2 <+41>:    je     0x400ed9 <phase_2+48>
   0x0000000000400ed4 <+43>:    callq  0x40142a <explode_bomb>
   0x0000000000400ed9 <+48>:    mov    %rsp,%rbx
   0x0000000000400edc <+51>:    lea    0x10(%rsp),%rbp
   0x0000000000400ee1 <+56>:    mov    0x4(%rbx),%eax
   0x0000000000400ee4 <+59>:    add    (%rbx),%eax
   0x0000000000400ee6 <+61>:    cmp    %eax,0x8(%rbx)
   0x0000000000400ee9 <+64>:    je     0x400ef0 <phase_2+71>
   0x0000000000400eeb <+66>:    callq  0x40142a <explode_bomb>
   0x0000000000400ef0 <+71>:    add    $0x4,%rbx
   0x0000000000400ef4 <+75>:    cmp    %rbp,%rbx
   0x0000000000400ef7 <+78>:    jne    0x400ee1 <phase_2+56>
   0x0000000000400ef9 <+80>:    mov    0x18(%rsp),%rax
   0x0000000000400efe <+85>:    xor    %fs:0x28,%rax
   0x0000000000400f07 <+94>:    je     0x400f0e <phase_2+101>
   0x0000000000400f09 <+96>:    callq  0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0e <+101>:   add    $0x28,%rsp
   0x0000000000400f12 <+105>:   pop    %rbx
   0x0000000000400f13 <+106>:   pop    %rbp
   0x0000000000400f14 <+107>:   retq
End of assembler dump.
(gdb) r ans.txt
```

Here in the above picture we can see that <25> which has a callq function that says <read_six_numbers>. So we got a hint about the input for phase_2 is 6 integers. Below the callq function there is cmpl function which compares 0x0 and first integer(%rsp). Then lets directly jump into line <30> and compere the two integers. For directly jumping into the particular line we have to use (until *address present in that line) and disas. In this case we will use (until*0x0000000000400ec7), so as we can see we are in line <30> so lets compare the two integers. Right now we don’t know the value stored in the (rsp) register, in order to find the values of the register we can do “i r” command which is information register and displays the information/ values of every register.

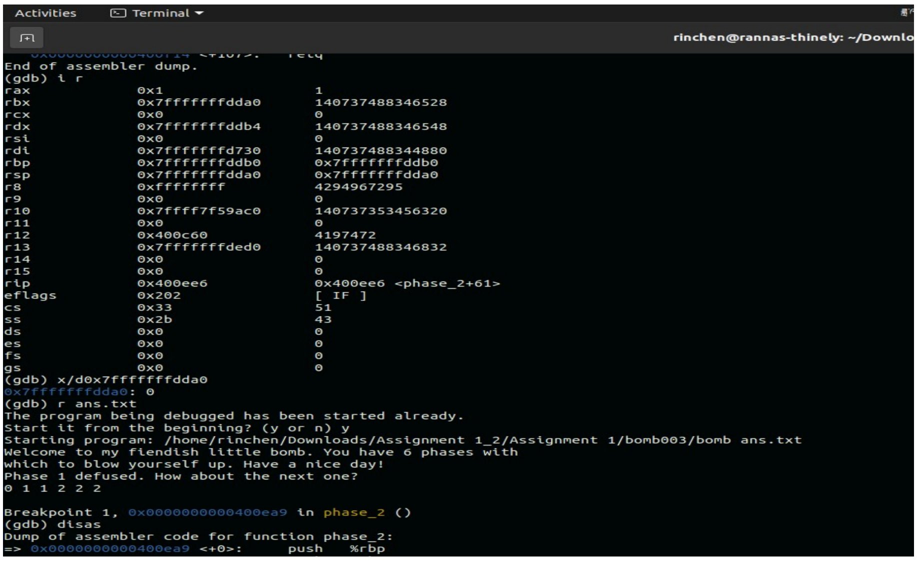
Bomb_3/Phase_2



```
(gdb) i r
rax            0x6                 6
rbx            0x7fffffffded8      140737488346840
rcx            0x0                 0
rdx            0x7fffffffddb4      140737488346548
rsi            0x0                 0
rdi            0x7fffffffd730      140737488344880
rbp            0x0                 0x0
rsp            0x7fffffffdda0      0x7fffffffdda0
r8             0xffffffff          4294967295
r9             0x0                 0
r10            0x7ffff7f59ac0      140737353456320
r11            0x0                 0
r12            0x400c60            4197472
r13            0x7fffffffded0      140737488346832
r14            0x0                 0
r15            0x0                 0
rip            0x400ecd            0x400ecd <phase_2+36>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb) x/d0x7fffffffdda0
0x7fffffffdda0: 0
(gdb) x/2d0x7fffffffdda0
0x7fffffffdda0: 0       1
(gdb) r ans.txt
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/rinchen/Downloads/Assignment 1_2/Assignment 1/bomb003/bomb ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
0 1 1 2 2 4

Breakpoint 1, 0x0000000000400ea9 in phase_2 ()
(gdb) disas
```

In above we are searching out the value for rsp and for that we use I r command and as we can see the value of rsp in hexadecimal form we can convert it into decimal using x/d and by giving the value of the rsp register and we got 1. so out here when we compare (0 and 1). As we know 0 and 1 are not equal so we move to next function <34> which is jne function (jump if not equal to) after going to the next function it will jump to <43> and the bomb will be get blast. But if the first input is 0 the function will move to line <36>.

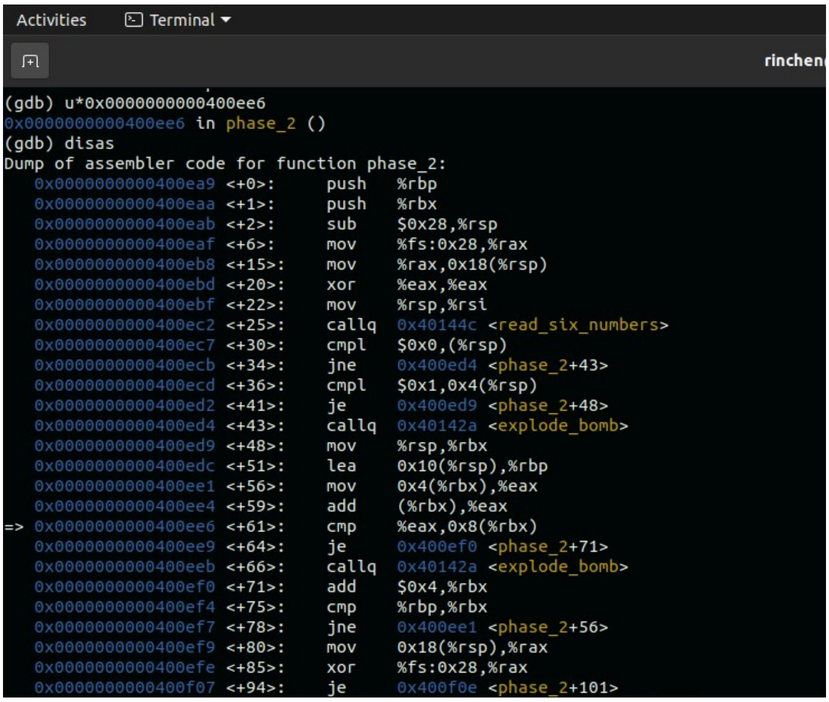


```
End of assembler dump.
(gdb) i r
rax            0x1                 1
rbx            0x7fffffffdda0      140737488346528
rcx            0x0                 0
rdx            0x7fffffffddb4      140737488346548
rsi            0x0                 0
rdi            0x7fffffffd730      140737488344880
rbp            0x7fffffffddb0      0x7fffffffddb0
rsp            0x7fffffffdda0      0x7fffffffdda0
r8             0xffffffff          4294967295
r9             0x0                 0
r10            0x7ffff7f59ac0      140737353456320
r11            0x0                 0
r12            0x400c60            4197472
r13            0x7fffffffded0      140737488346832
r14            0x0                 0
r15            0x0                 0
rip            0x400ee6            0x400ee6 <phase_2+61>
eflags         0x202               [ IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb) x/d0x7fffffffdda0
0x7fffffffdda0: 0
(gdb) r ans.txt
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/rinchen/Downloads/Assignment 1_2/Assignment 1/bomb003/bomb ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
0 1 1 2 2 2

Breakpoint 1, 0x0000000000400ea9 in phase_2 ()
(gdb) disas
Dump of assembler code for function phase_2:
=> 0x0000000000400ea9 <+0>:     push   %rbp
```

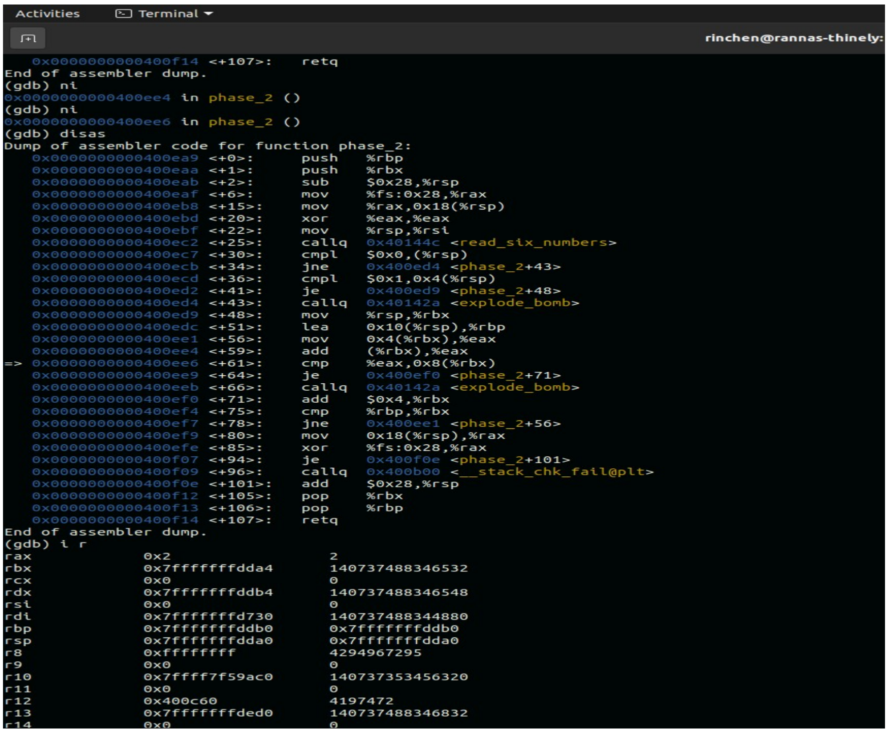
Bomb_3/Phase_2

now we will again go inside the disassemble and look for next value. Now we will go to second compare function which compares (1 and %rsp). For that we have to find the value of rsp and as we can see in the picture above that the its value is 0 and when we compare it with 1 which is not equal which can cause the bomb to get blast so therefore in order to not let that happen we must consider the second digit as 1 and move further, now we got first and second digit and it time for third one.



```
(gdb) u*0x0000000000400ee6
0x0000000000400ee6 in phase_2 ()
(gdb) disas
Dump of assembler code for function phase_2:
   0x0000000000400ea9 <+0>:     push   %rbp
   0x0000000000400eaa <+1>:     push   %rbx
   0x0000000000400eab <+2>:     sub    $0x28,%rsp
   0x0000000000400eaf <+6>:     mov    %fs:0x28,%rax
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    callq  0x40144c <read_six_numbers>
   0x0000000000400ec7 <+30>:    cmpl   $0x0,(%rsp)
   0x0000000000400ecb <+34>:    jne    0x400ed4 <phase_2+43>
   0x0000000000400ecd <+36>:    cmpl   $0x1,0x4(%rsp)
   0x0000000000400ed2 <+41>:    je     0x400ed9 <phase_2+48>
   0x0000000000400ed4 <+43>:    callq  0x40142a <explode_bomb>
   0x0000000000400ed9 <+48>:    mov    %rsp,%rbx
   0x0000000000400edc <+51>:    lea    0x10(%rsp),%rbp
   0x0000000000400ee1 <+56>:    mov    0x4(%rbx),%eax
   0x0000000000400ee4 <+59>:    add    (%rbx),%eax
=> 0x0000000000400ee6 <+61>:    cmp    %eax,0x8(%rbx)
   0x0000000000400ee9 <+64>:    je     0x400ef0 <phase_2+71>
   0x0000000000400eeb <+66>:    callq  0x40142a <explode_bomb>
   0x0000000000400ef0 <+71>:    add    $0x4,%rbx
   0x0000000000400ef4 <+75>:    cmp    %rbp,%rbx
   0x0000000000400ef7 <+78>:    jne    0x400ee1 <phase_2+56>
   0x0000000000400ef9 <+80>:    mov    0x18(%rsp),%rax
   0x0000000000400efe <+85>:    xor    %fs:0x28,%rax
   0x0000000000400f07 <+94>:    je     0x400f0e <phase_2+101>
```

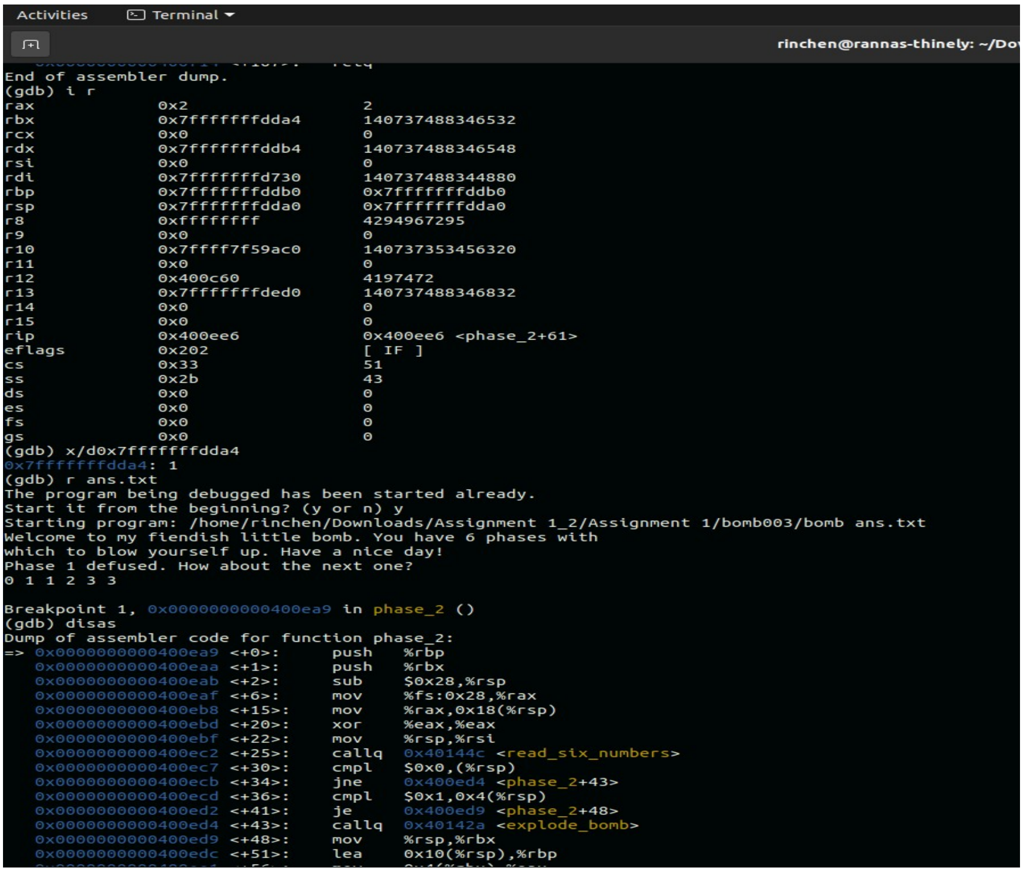
Here the procedure is same as that of before. We have to run the program and insert first two input that is 0 and 1 and after that run dissembler and go to next line where the function calls. Right now it is at line <48>. There are few operations such as mov, add and etc. But we will directly go to the compare function by commanding “ni” which means next line.



```
   0x0000000000400f14 <+107>:   retq
End of assembler dump.
(gdb) ni
0x0000000000400ee4 in phase_2 ()
(gdb) ni
0x0000000000400ee6 in phase_2 ()
(gdb) disas
Dump of assembler code for function phase_2:
   0x0000000000400ea9 <+0>:     push   %rbp
   0x0000000000400eaa <+1>:     push   %rbx
   0x0000000000400eab <+2>:     sub    $0x28,%rsp
   0x0000000000400eaf <+6>:     mov    %fs:0x28,%rax
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    callq  0x40144c <read_six_numbers>
   0x0000000000400ec7 <+30>:    cmpl   $0x0,(%rsp)
   0x0000000000400ecb <+34>:    jne    0x400ed4 <phase_2+43>
   0x0000000000400ecd <+36>:    cmpl   $0x1,0x4(%rsp)
   0x0000000000400ed2 <+41>:    je     0x400ed9 <phase_2+48>
   0x0000000000400ed4 <+43>:    callq  0x40142a <explode_bomb>
   0x0000000000400ed9 <+48>:    mov    %rsp,%rbx
   0x0000000000400edc <+51>:    lea    0x10(%rsp),%rbp
   0x0000000000400ee1 <+56>:    mov    0x4(%rbx),%eax
   0x0000000000400ee4 <+59>:    add    (%rbx),%eax
=> 0x0000000000400ee6 <+61>:    cmp    %eax,0x8(%rbx)
   0x0000000000400ee9 <+64>:    je     0x400ef0 <phase_2+71>
   0x0000000000400eeb <+66>:    callq  0x40142a <explode_bomb>
   0x0000000000400ef0 <+71>:    add    $0x4,%rbx
   0x0000000000400ef4 <+75>:    cmp    %rbp,%rbx
   0x0000000000400ef7 <+78>:    jne    0x400ee1 <phase_2+56>
   0x0000000000400ef9 <+80>:    mov    0x18(%rsp),%rax
   0x0000000000400efe <+85>:    xor    %fs:0x28,%rax
   0x0000000000400f07 <+94>:    je     0x400f0e <phase_2+101>
   0x0000000000400f09 <+96>:    callq  0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0e <+101>:   add    $0x28,%rsp
   0x0000000000400f12 <+105>:   pop    %rbx
   0x0000000000400f13 <+106>:   pop    %rbp
   0x0000000000400f14 <+107>:   retq
End of assembler dump.
(gdb) i r
rax            0x2                 2
rbx            0x7fffffffdda4      140737488346532
rcx            0x0                 0
rdx            0x7fffffffddb4      140737488346548
rsi            0x0                 0
rdi            0x7fffffffd730      140737488344880
rbp            0x7fffffffddb0      0x7fffffffddb0
rsp            0x7fffffffdda0      0x7fffffffdda0
r8             0xffffffff          4294967295
r9             0x0                 0
r10            0x7ffff7f59ac0      140737353456320
r11            0x0                 0
r12            0x400c60            4197472
r13            0x7fffffffded0      140737488346832
r14            0x0                 0
```

Bomb_3/Phase_2

After some looping, moving ,adding we have reach to line 61 which compares(%eax and %rbx). We have to find the decimal values stored in these registers. For that we will follow the same procedure by going into the information registers and getting the value of them.

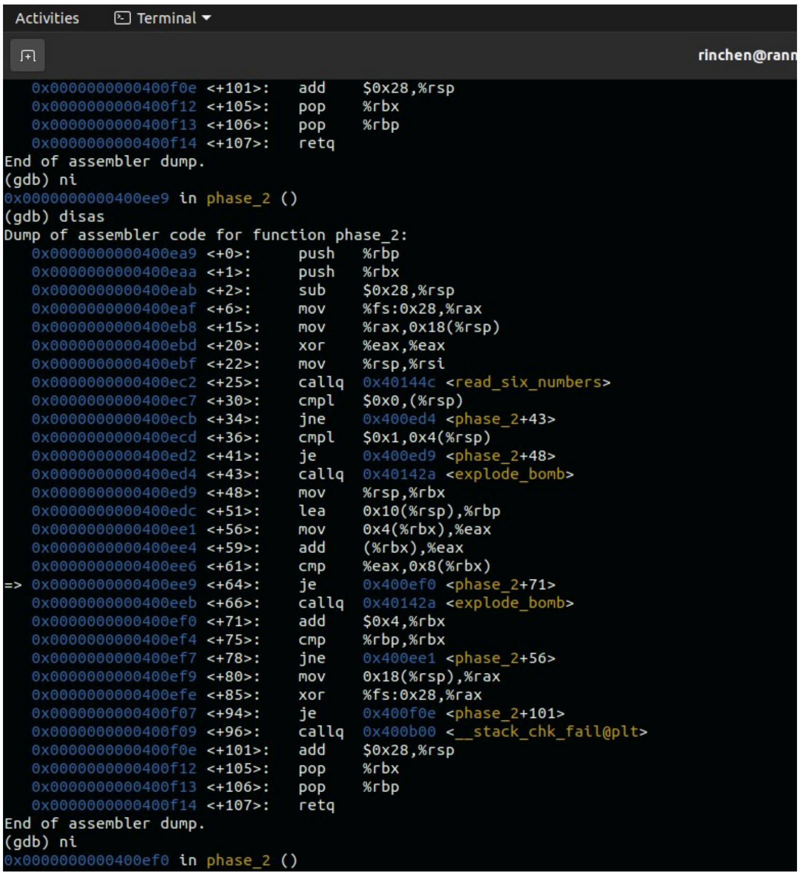


```
End of assembler dump.
(gdb) i r
rax            0x2                 2
rbx            0x7fffffffdda4      140737488346532
rcx            0x0                 0
rdx            0x7fffffffddb4      140737488346548
rsi            0x0                 0
rdi            0x7fffffffd730      140737488344880
rbp            0x7fffffffddb0      0x7fffffffddb0
rsp            0x7fffffffdda0      0x7fffffffdda0
r8             0xffffffff          4294967295
r9             0x0                 0
r10            0x7ffff7f59ac0      140737353456320
r11            0x0                 0
r12            0x400c60            4197472
r13            0x7fffffffded0      140737488346832
r14            0x0                 0
r15            0x0                 0
rip            0x400ee6            0x400ee6 <phase_2+61>
eflags         0x202               [ IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb) x/d0x7fffffffdda4
0x7fffffffdda4: 1
(gdb) r ans.txt
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/rinchen/Downloads/Assignment 1_2/Assignment 1/bomb003/bomb ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
0 1 1 2 3 3

Breakpoint 1, 0x0000000000400ea9 in phase_2 ()
(gdb) disas
Dump of assembler code for function phase_2:
=> 0x0000000000400ea9 <+0>:     push   %rbp
   0x0000000000400eaa <+1>:     push   %rbx
   0x0000000000400eab <+2>:     sub    $0x28,%rsp
   0x0000000000400eaf <+6>:     mov    %fs:0x28,%rax
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    callq  0x40144c <read_six_numbers>
   0x0000000000400ec7 <+30>:    cmpl   $0x0,(%rsp)
   0x0000000000400ecb <+34>:    jne    0x400ed4 <phase_2+43>
   0x0000000000400ecd <+36>:    cmpl   $0x1,0x4(%rsp)
   0x0000000000400ed2 <+41>:    je     0x400ed9 <phase_2+48>
   0x0000000000400ed4 <+43>:    callq  0x40142a <explode_bomb>
   0x0000000000400ed9 <+48>:    mov    %rsp,%rbx
   0x0000000000400edc <+51>:    lea    0x10(%rsp),%rbp
```

In the above picture we can see that we we compare 2 and 1, they are not equal to each other which can cause the bomb to blast so therefore to get into the safer side we can put of third digit as 1 and run the program to whether it will work or not. As the we have given the our input as one the comparison becomes equal and it will jump into line <64>.

Bomb_3/Phase_2

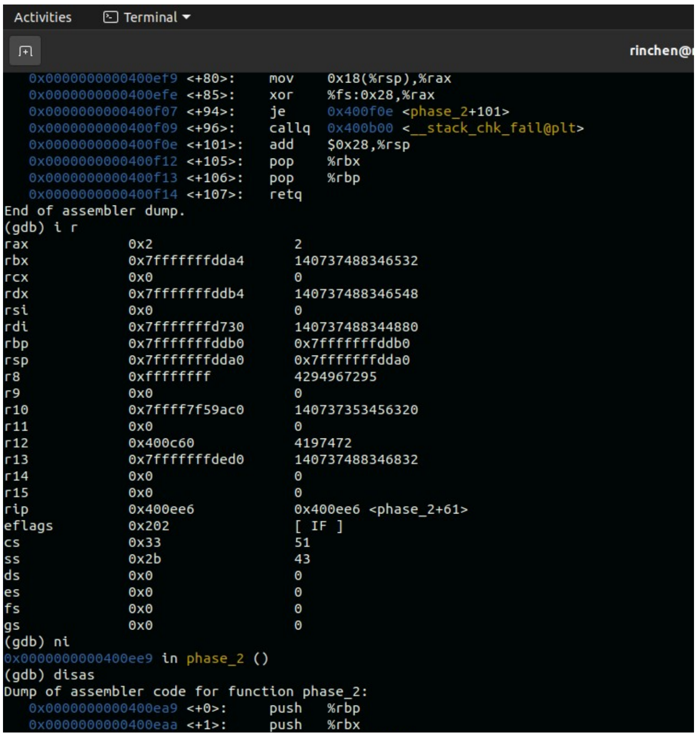


```
   0x0000000000400f0e <+101>:   add    $0x28,%rsp
   0x0000000000400f12 <+105>:   pop    %rbx
   0x0000000000400f13 <+106>:   pop    %rbp
   0x0000000000400f14 <+107>:   retq
End of assembler dump.
(gdb) ni
0x0000000000400ee9 in phase_2 ()
(gdb) disas
Dump of assembler code for function phase_2:
   0x0000000000400ea9 <+0>:     push   %rbp
   0x0000000000400eaa <+1>:     push   %rbx
   0x0000000000400eab <+2>:     sub    $0x28,%rsp
   0x0000000000400eaf <+6>:     mov    %fs:0x28,%rax
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    callq  0x40144c <read_six_numbers>
   0x0000000000400ec7 <+30>:    cmpl   $0x0,(%rsp)
   0x0000000000400ecb <+34>:    jne    0x400ed4 <phase_2+43>
   0x0000000000400ecd <+36>:    cmpl   $0x1,0x4(%rsp)
   0x0000000000400ed2 <+41>:    je     0x400ed9 <phase_2+48>
   0x0000000000400ed4 <+43>:    callq  0x40142a <explode_bomb>
   0x0000000000400ed9 <+48>:    mov    %rsp,%rbx
   0x0000000000400edc <+51>:    lea    0x10(%rsp),%rbp
   0x0000000000400ee1 <+56>:    mov    0x4(%rbx),%eax
   0x0000000000400ee4 <+59>:    add    (%rbx),%eax
   0x0000000000400ee6 <+61>:    cmp    %eax,0x8(%rbx)
=> 0x0000000000400ee9 <+64>:    je     0x400ef0 <phase_2+71>
   0x0000000000400eeb <+66>:    callq  0x40142a <explode_bomb>
   0x0000000000400ef0 <+71>:    add    $0x4,%rbx
   0x0000000000400ef4 <+75>:    cmp    %rbp,%rbx
   0x0000000000400ef7 <+78>:    jne    0x400ee1 <phase_2+56>
   0x0000000000400ef9 <+80>:    mov    0x18(%rsp),%rax
   0x0000000000400efe <+85>:    xor    %fs:0x28,%rax
   0x0000000000400f07 <+94>:    je     0x400f0e <phase_2+101>
   0x0000000000400f09 <+96>:    callq  0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0e <+101>:   add    $0x28,%rsp
   0x0000000000400f12 <+105>:   pop    %rbx
   0x0000000000400f13 <+106>:   pop    %rbp
   0x0000000000400f14 <+107>:   retq
End of assembler dump.
(gdb) ni
0x0000000000400ef0 in phase_2 ()
```

for the fourth one as we are in line <64 >, so from there with the function je(jump if equals to) , it will jump into the line <71> where it added two registers and compared in the next step. we have compare function in line <75> that is (%rbp and rbx) . Now we have got the value those functions in decimal that is (3and1) .

The two numbers are not equal so it will go to next function which says jne(jump if not equal to) ,so it will go in the jne function and get inside jne function it takes us to line <56>. In line <56> we have some mov and add functions and after that in line <61>, we have the compare function which compares(eax and rbx). So after opening the information register we got the value as (2 and 1). So as we know that if the two number did not match than the bomb will get blast.

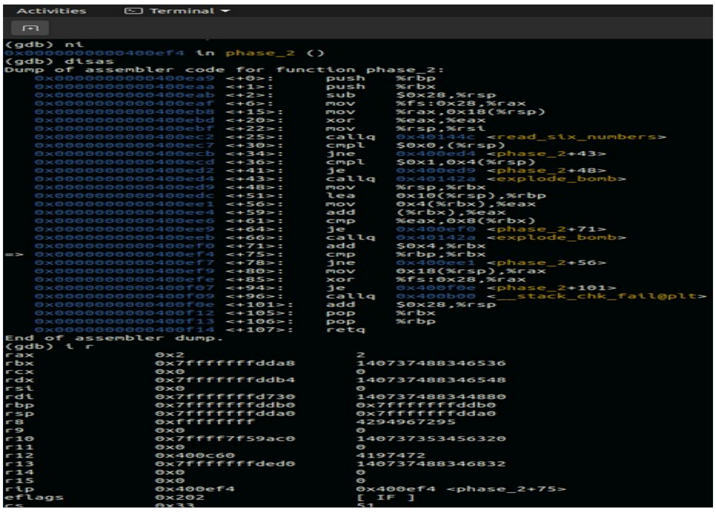
Bomb_3/Phase_2



```
    0x0000000000400ef9 <+80>:    mov    0x18(%rsp),%rax
    0x0000000000400efe <+85>:    xor    %fs:0x28,%rax
    0x0000000000400f07 <+94>:    je     0x400f0e <phase_2+101>
    0x0000000000400f09 <+96>:    callq  0x400b00 <__stack_chk_fail@plt>
    0x0000000000400f0e <+101>:   add    $0x28,%rsp
    0x0000000000400f12 <+105>:   pop    %rbx
    0x0000000000400f13 <+106>:   pop    %rbp
    0x0000000000400f14 <+107>:   retq
End of assembler dump.
(gdb) i r
rax            0x2                 2
rbx            0x7fffffffdda4      140737488346532
rcx            0x0                 0
rdx            0x7fffffffddb4      140737488346548
rsi            0x0                 0
rdi            0x7fffffffd730      140737488344880
rbp            0x7fffffffddb0      0x7fffffffddb0
rsp            0x7fffffffdda0      0x7fffffffdda0
r8             0xffffffff          4294967295
r9             0x0                 0
r10            0x7ffff7f59ac0      140737353456320
r11            0x0                 0
r12            0x400c60            4197472
r13            0x7fffffffded0      140737488346832
r14            0x0                 0
r15            0x0                 0
rip            0x400ee6            0x400ee6 <phase_2+61>
eflags         0x202               [ IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb) ni
0x0000000000400ee9 in phase_2 ()
(gdb) disas
Dump of assembler code for function phase_2:
    0x0000000000400ea9 <+0>:     push   %rbp
    0x0000000000400eaa <+1>:     push   %rbx
```

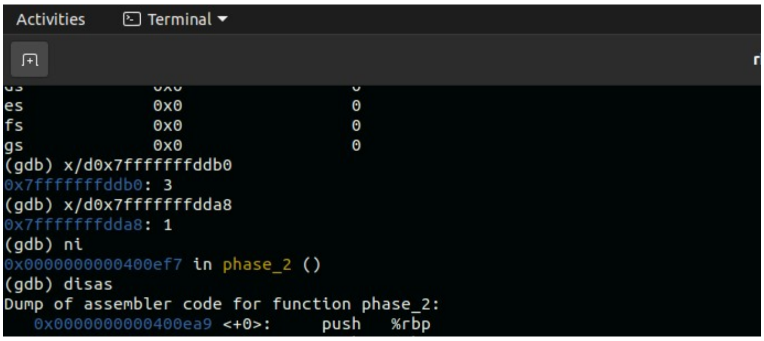
In order to not let that happen we should keep out fourth digit 2.

For fifth digit, we have to follow the same procedure like what we did before and here we have reached to line<75> now it will compare the two registers. The values were found to be (3 and 1) and since they are not equal it will jump to next line which says jne (jump if not equals to).

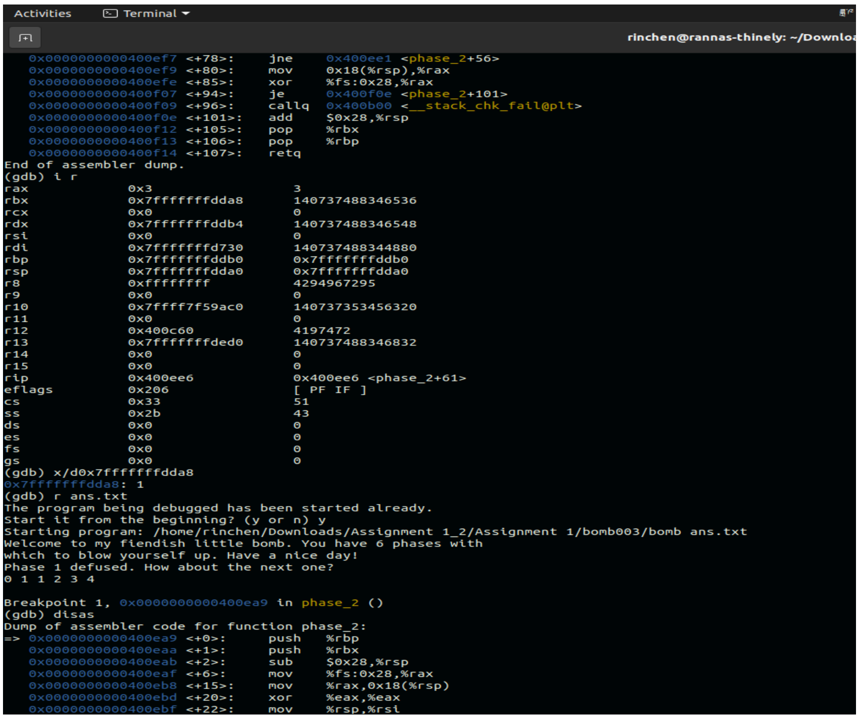


```
(gdb) ni
0x0000000000400ef4 in phase_2 ()
(gdb) disas
Dump of assembler code for function phase_2:
    0x0000000000400ea9 <+0>:     push   %rbp
    0x0000000000400eaa <+1>:     push   %rbx
    0x0000000000400eab <+2>:     sub    $0x28,%rsp
    0x0000000000400eaf <+6>:     mov    %fs:0x28,%rax
    0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
    0x0000000000400ebd <+20>:    xor    %eax,%eax
    0x0000000000400ebf <+22>:    mov    %rsp,%rsi
    0x0000000000400ec2 <+25>:    callq  0x40144c <read_six_numbers>
    0x0000000000400ec7 <+30>:    cmpl   $0x0,(%rsp)
    0x0000000000400ecb <+34>:    jne    0x400ed4 <phase_2+43>
    0x0000000000400ecd <+36>:    cmpl   $0x1,0x4(%rsp)
    0x0000000000400ed2 <+41>:    je     0x400ed9 <phase_2+48>
    0x0000000000400ed4 <+43>:    callq  0x40142a <explode_bomb>
    0x0000000000400ed9 <+48>:    mov    %rsp,%rbx
    0x0000000000400edc <+51>:    lea    0x10(%rsp),%rbp
    0x0000000000400ee1 <+56>:    mov    0x4(%rbx),%eax
    0x0000000000400ee4 <+59>:    add    (%rbx),%eax
    0x0000000000400ee6 <+61>:    cmp    %eax,0x8(%rbx)
    0x0000000000400ee9 <+64>:    je     0x400ef0 <phase_2+71>
    0x0000000000400eeb <+66>:    callq  0x40142a <explode_bomb>
    0x0000000000400ef0 <+71>:    add    $0x4,%rbx
=>  0x0000000000400ef4 <+75>:    cmp    %rbp,%rbx
    0x0000000000400ef7 <+78>:    jne    0x400ee1 <phase_2+56>
    0x0000000000400ef9 <+80>:    mov    0x18(%rsp),%rax
    0x0000000000400efe <+85>:    xor    %fs:0x28,%rax
    0x0000000000400f07 <+94>:    je     0x400f0e <phase_2+101>
    0x0000000000400f09 <+96>:    callq  0x400b00 <__stack_chk_fail@plt>
    0x0000000000400f0e <+101>:   add    $0x28,%rsp
    0x0000000000400f12 <+105>:   pop    %rbx
    0x0000000000400f13 <+106>:   pop    %rbp
    0x0000000000400f14 <+107>:   retq
End of assembler dump.
(gdb) i r
rax            0x2                 2
rbx            0x7fffffffdda8      140737488346536
rcx            0x0                 0
rdx            0x7fffffffddb4      140737488346548
rsi            0x0                 0
rdi            0x7fffffffd730      140737488344880
rbp            0x7fffffffddb0      0x7fffffffddb0
rsp            0x7fffffffdda0      0x7fffffffdda0
r8             0xffffffff          4294967295
r9             0x0                 0
r10            0x7ffff7f59ac0      140737353456320
r11            0x0                 0
r12            0x400c60            4197472
r13            0x7fffffffded0      140737488346832
r14            0x0                 0
r15            0x0                 0
rip            0x400ef4            0x400ef4 <phase_2+75>
eflags         0x202               [ IF ]
cs             0x33                51
```

Bomb_3/Phase_2



```
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb) x/d0x7fffffffddb0
0x7fffffffddb0: 3
(gdb) x/d0x7fffffffdda8
0x7fffffffdda8: 1
(gdb) ni
0x0000000000400ef7 in phase_2 ()
(gdb) disas
Dump of assembler code for function phase_2:
   0x0000000000400ea9 <+0>:     push   %rbp
```



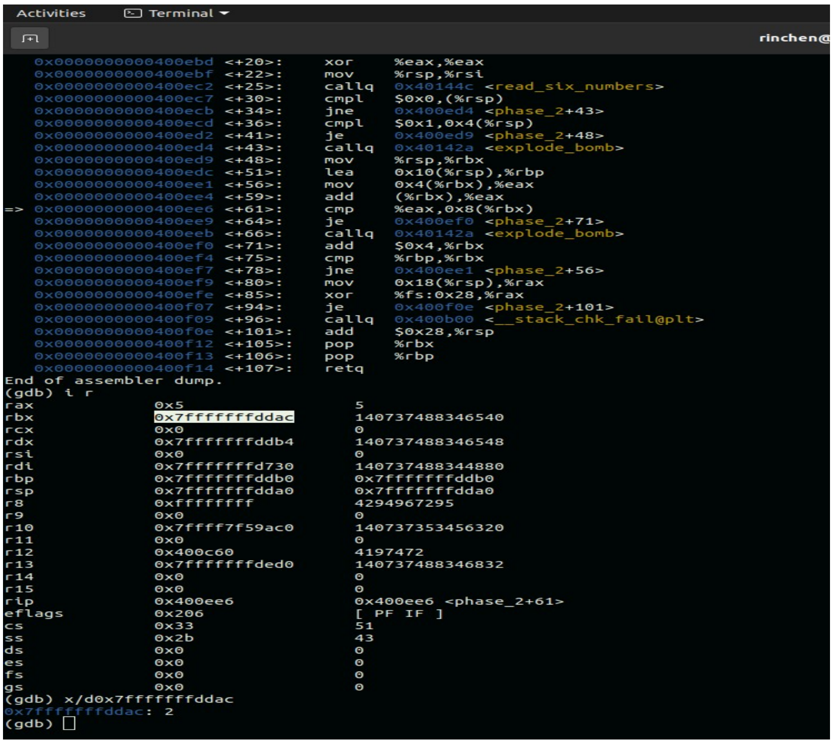
```
   0x0000000000400ef7 <+78>:    jne    0x400ee1 <phase_2+56>
   0x0000000000400ef9 <+80>:    mov    0x18(%rsp),%rax
   0x0000000000400efe <+85>:    xor    %fs:0x28,%rax
   0x0000000000400f07 <+94>:    je     0x400f0e <phase_2+101>
   0x0000000000400f09 <+96>:    callq  0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0e <+101>:   add    $0x28,%rsp
   0x0000000000400f12 <+105>:   pop    %rbx
   0x0000000000400f13 <+106>:   pop    %rbp
   0x0000000000400f14 <+107>:   retq
End of assembler dump.
(gdb) i r
rax            0x3                 3
rbx            0x7fffffffdda8      140737488346536
rcx            0x0                 0
rdx            0x7fffffffddb4      140737488346548
rsi            0x0                 0
rdi            0x7fffffffd730      140737488344880
rbp            0x7fffffffddb0      0x7fffffffddb0
rsp            0x7fffffffdda0      0x7fffffffdda0
r8             0xffffffff          4294967295
r9             0x0                 0
r10            0x7ffff7f59ac0      140737353456320
r11            0x0                 0
r12            0x400c60            4197472
r13            0x7fffffffded0      140737488346832
r14            0x0                 0
r15            0x0                 0
rip            0x400ee6            0x400ee6 <phase_2+61>
eflags         0x206               [ PF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb) x/d0x7fffffffdda8
0x7fffffffdda8: 1
(gdb) r ans.txt
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/rinchen/Downloads/Assignment 1_2/Assignment 1/bomb003/bomb ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
0 1 1 2 3 4

Breakpoint 1, 0x0000000000400ea9 in phase_2 ()
(gdb) disas
Dump of assembler code for function phase_2:
=> 0x0000000000400ea9 <+0>:     push   %rbp
   0x0000000000400eaa <+1>:     push   %rbx
   0x0000000000400eab <+2>:     sub    $0x28,%rsp
   0x0000000000400eaf <+6>:     mov    %fs:0x28,%rax
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
```

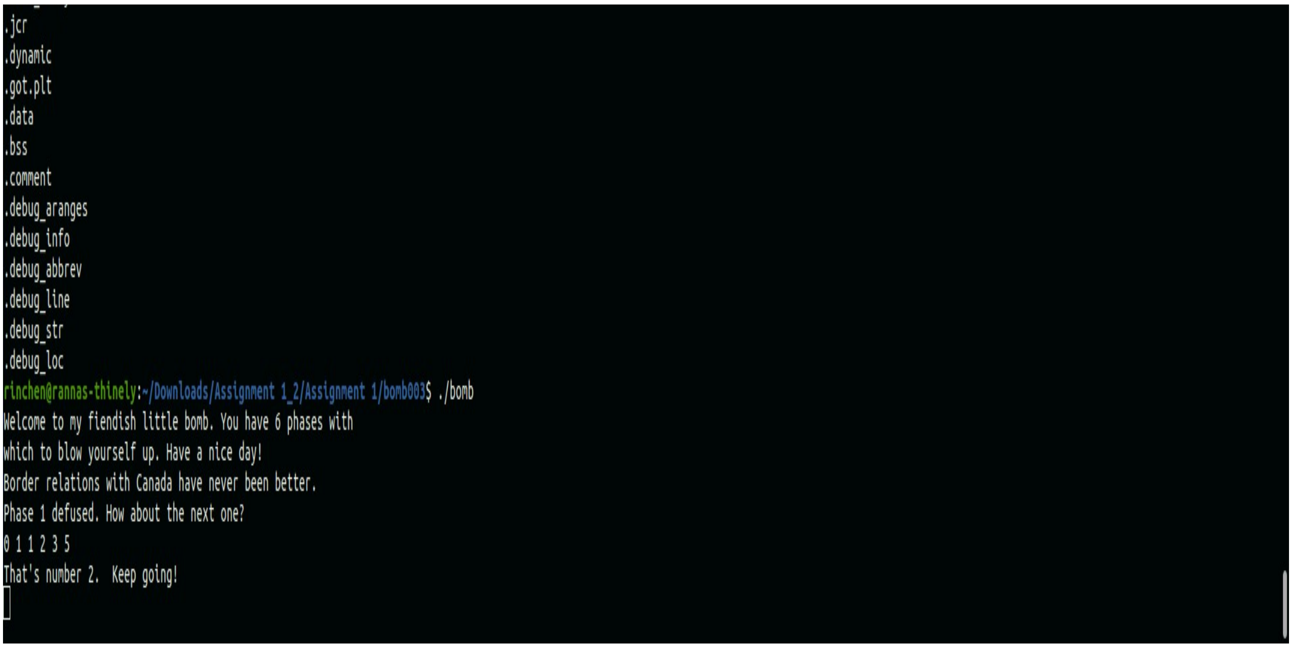
In line <78> the function will move to line <56> where it will perform mov and add functions and it will compare (%eax and %rbx) where there values were (3 and 1). so as the fifth input is not equal to 3, the bomb will get blast, in order to not let that happen the correct fifth input is 3.

Bomb_3/Phase_2

In fifth we have to go inside<65> then we have to compare (eax and rbx). We got the(5 and 2) as they are not equal the bomb will be exploded and in order not the bomb to get blast we must input the last digit as 5.



```
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    callq  0x40144c <read_six_numbers>
   0x0000000000400ec7 <+30>:    cmpl   $0x0,(%rsp)
   0x0000000000400ecb <+34>:    jne    0x400ed4 <phase_2+43>
   0x0000000000400ecd <+36>:    cmpl   $0x1,0x4(%rsp)
   0x0000000000400ed2 <+41>:    je     0x400ed9 <phase_2+48>
   0x0000000000400ed4 <+43>:    callq  0x40142a <explode_bomb>
   0x0000000000400ed9 <+48>:    mov    %rsp,%rbx
   0x0000000000400edc <+51>:    lea    0x10(%rsp),%rbp
   0x0000000000400ee1 <+56>:    mov    0x4(%rbx),%eax
   0x0000000000400ee4 <+59>:    add    (%rbx),%eax
=> 0x0000000000400ee6 <+61>:    cmp    %eax,0x8(%rbx)
   0x0000000000400ee9 <+64>:    je     0x400ef0 <phase_2+71>
   0x0000000000400eeb <+66>:    callq  0x40142a <explode_bomb>
   0x0000000000400ef0 <+71>:    add    $0x4,%rbx
   0x0000000000400ef4 <+75>:    cmp    %rbp,%rbx
   0x0000000000400ef7 <+78>:    jne    0x400ee1 <phase_2+56>
   0x0000000000400ef9 <+80>:    mov    0x18(%rsp),%rax
   0x0000000000400efe <+85>:    xor    %fs:0x28,%rax
   0x0000000000400f07 <+94>:    je     0x400f0e <phase_2+101>
   0x0000000000400f09 <+96>:    callq  0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0e <+101>:   add    $0x28,%rsp
   0x0000000000400f12 <+105>:   pop    %rbx
   0x0000000000400f13 <+106>:   pop    %rbp
   0x0000000000400f14 <+107>:   retq
End of assembler dump.
(gdb) i r
rax            0x5                 5
rbx            0x7fffffffddac      140737488346540
rcx            0x0                 0
rdx            0x7fffffffddb4      140737488346548
rsi            0x0                 0
rdi            0x7fffffffd730      140737488344880
rbp            0x7fffffffddb0      0x7fffffffddb0
rsp            0x7fffffffdda0      0x7fffffffdda0
r8             0xffffffff          4294967295
r9             0x0                 0
r10            0x7ffff7f59ac0      140737353456320
r11            0x0                 0
r12            0x400c60            4197472
r13            0x7fffffffded0      140737488346832
r14            0x0                 0
r15            0x0                 0
rip            0x400ee6            0x400ee6 <phase_2+61>
eflags         0x206               [ PF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb) x/d0x7fffffffddac
0x7fffffffddac: 2
(gdb) 
```



```
.jcr
.dynamic
.got.plt
.data
.bss
.comment
.debug_aranges
.debug_info
.debug_abbrev
.debug_line
.debug_str
.debug_loc
rinchen@rannas-thinely:~/Downloads/Assignment 1_2/Assignment 1/bomb003$ ./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?
0 1 1 2 3 5
That's number 2.  Keep going!
```

Finally we got the phase_2 answer too which was the Fibonacci six digit numbers.
Now lets go to the last Phase.