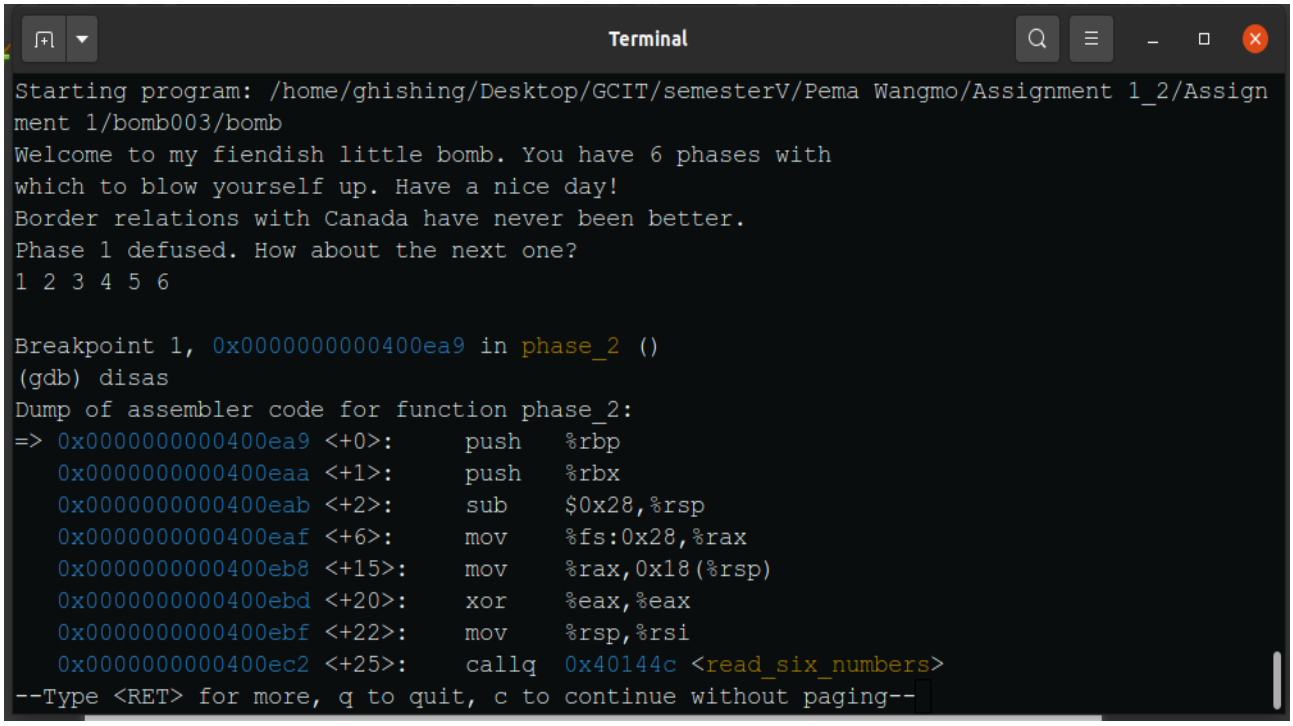
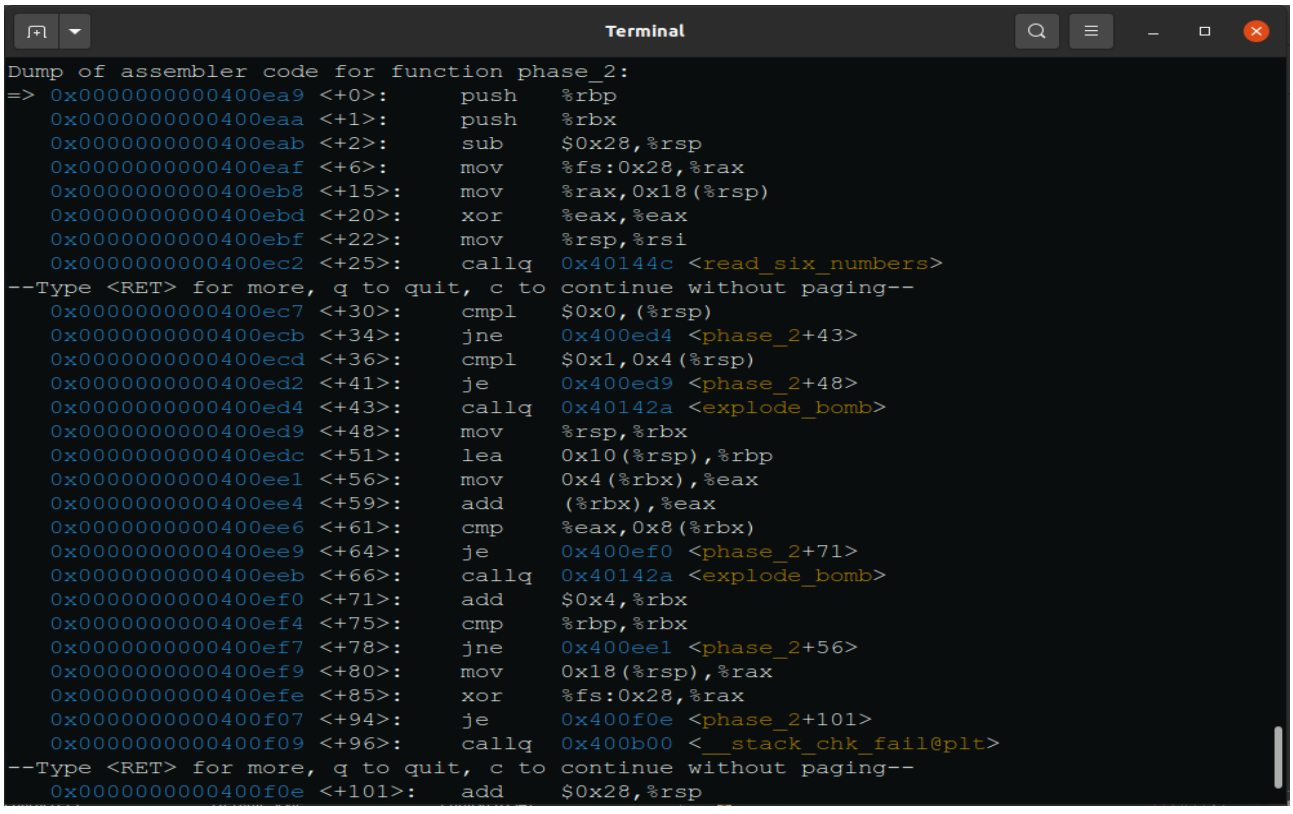
Phase_2

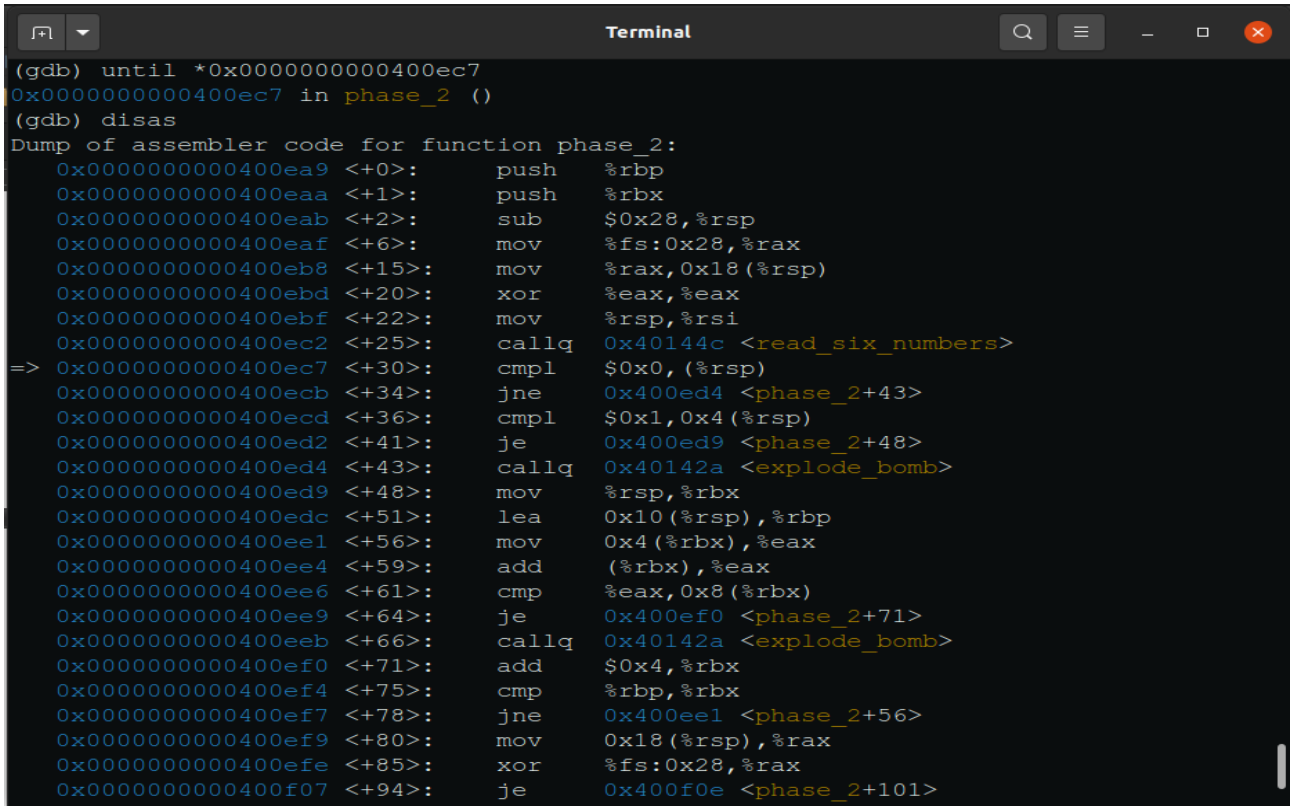
To continue the phase, now we have to set a break point at phase to avoid blowing up of bomb. And run the program and give a random trail for it. Again disassemble the program using “disas” command.



In above program code for phase_2, it has a function named <read_six_numbers> and we get hint that it is six digit integer.

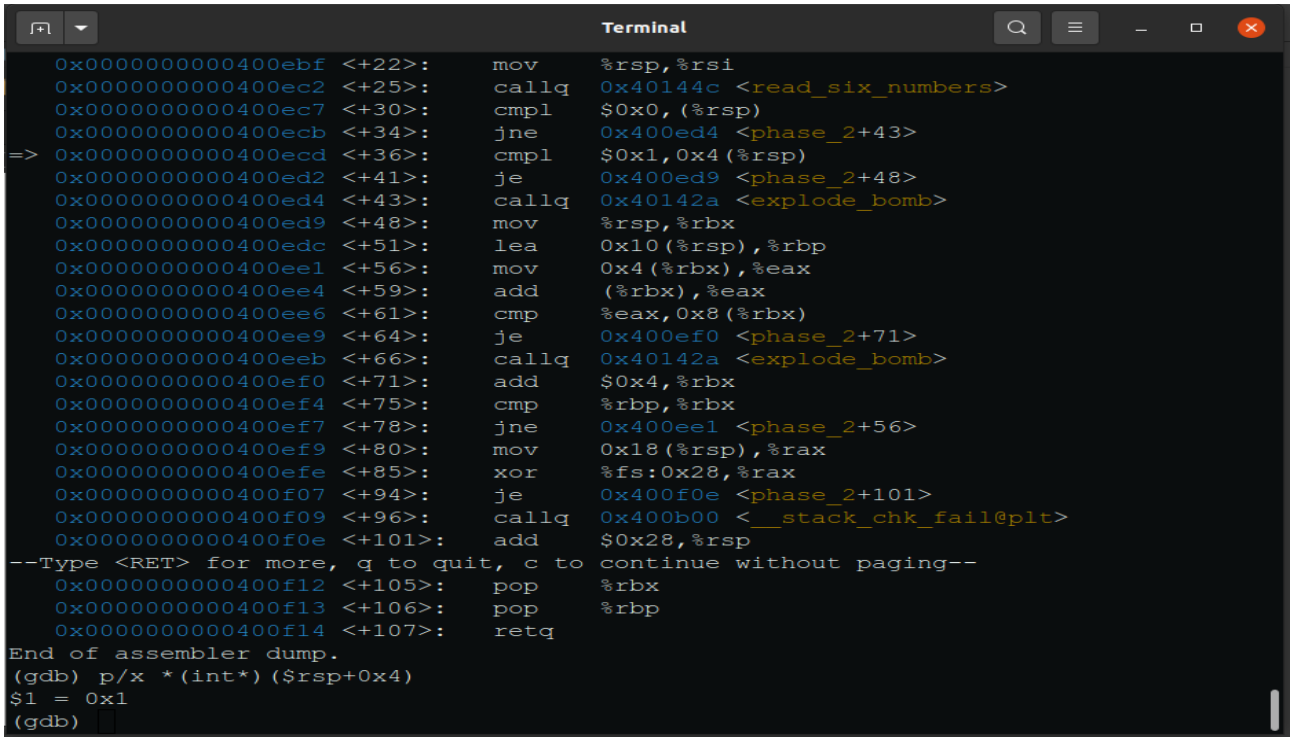


For a trail, random numbers(1,2,3,4,5,6) was given as input and as soon as the numbers are provided, program code makes first compare with the first digit i.e., 1. From the code “cmpl $0x0, (%rsp)” we can understand that 0 is being compared to 1 and the program logic states that if 0 is equal to 1 then jump to line number 48 and compare with next number else it calls <explode_bomb> function and bomb gets explode. After executing this line we can figure out that first digit is 0.

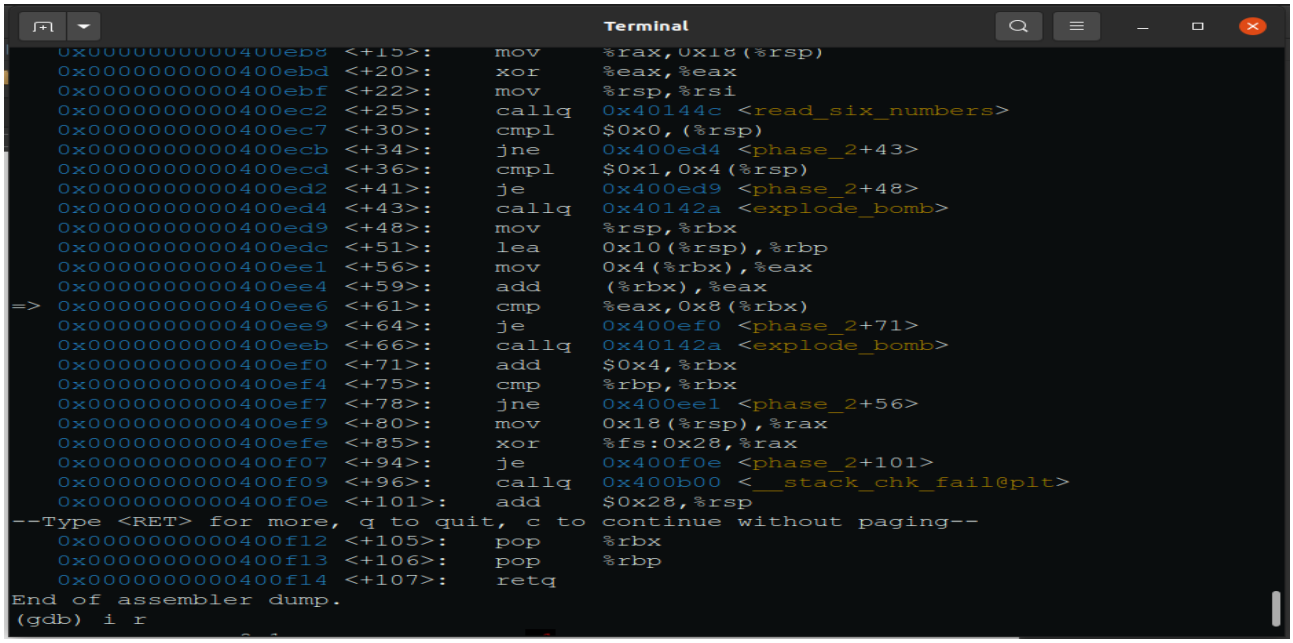


Now we know that first digit is 0, lets try running the program again and insert new value with first digit 0 and rest we will keep the same.

We have inputted second digit as 1, the program code compares it with value $0x1, and the result turned it be true and we can declare that second digit is 1. As it resulted as true it jumps line 48 and performs certain operation like mov lea and add.



lets compare the value at line 63 " cmp %eax,0x8(%rbx)", this time the value of %eax is 1 and value of 0x8(%rbx) is 2 . It doesn't match, thus the bomb will blow up. From this comparasion we can figure out 3rd digit is also 1 as shown below.



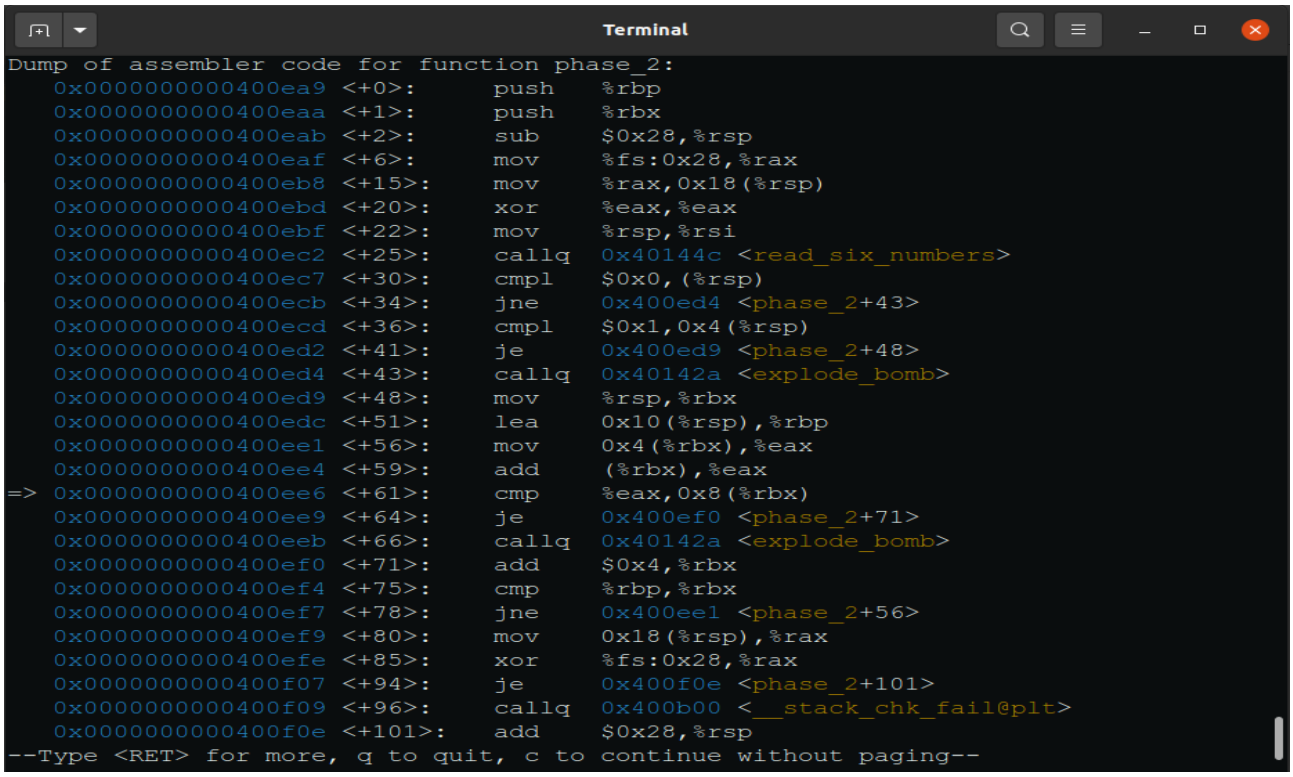
```
Terminal
   0x0000000000400f14 <+107>:    retq
End of assembler dump.
(gdb) i r
rax            0x1                 1
rbx            0x7fffffffe2c0      140737488347840
rcx            0x0                 0
rdx            0x7fffffffe2d4      140737488347860
rsi            0x0                 0
rdi            0x7fffffffdc50      140737488346192
rbp            0x7fffffffe2d0      0x7fffffffe2d0
rsp            0x7fffffffe2c0      0x7fffffffe2c0
r8             0xffffffff          4294967295
r9             0x0                 0
r10            0x7ffff7f5aac0      140737353460416
r11            0x0                 0
r12            0x400c60            4197472
r13            0x7fffffffe3f0      140737488348144
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) p/x *(int*)($rbx+0x8)
$2 = 0x2
(gdb)
```

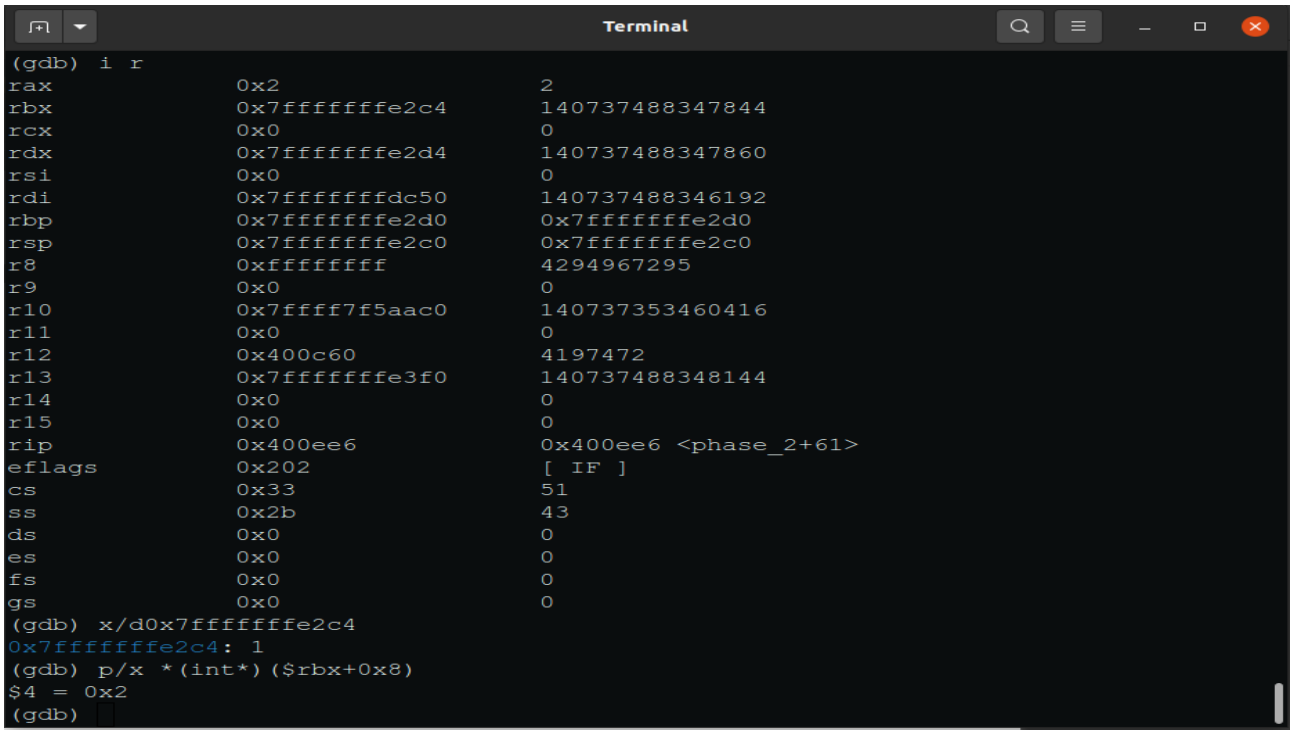
```
Terminal
   0x0000000000400ecb <+34>:     jne    0x400ed4 <phase_2+43>
   0x0000000000400ecd <+36>:     cmpl   $0x1,0x4(%rsp)
   0x0000000000400ed2 <+41>:     je     0x400ed9 <phase_2+48>
   0x0000000000400ed4 <+43>:     callq  0x40142a <explode_bomb>
   0x0000000000400ed9 <+48>:     mov    %rsp,%rbx
   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
--Type <RET> for more, q to quit, c to continue without paging--
   0x0000000000400f12 <+105>:    pop    %rbx
   0x0000000000400f13 <+106>:    pop    %rbp
   0x0000000000400f14 <+107>:    retq
End of assembler dump.
(gdb) ni

BOOM!!!
The bomb has blown up.
[Inferior 1 (process 69283) exited with code 010]
(gdb)
```

lets run the program again and input respective values 0, 1, 1, 2, 3, 4 and check the program code. Again lets compare at line number 61 and this time %eax and 0x8(%rbx) is 1 and it resulted be true as show below.

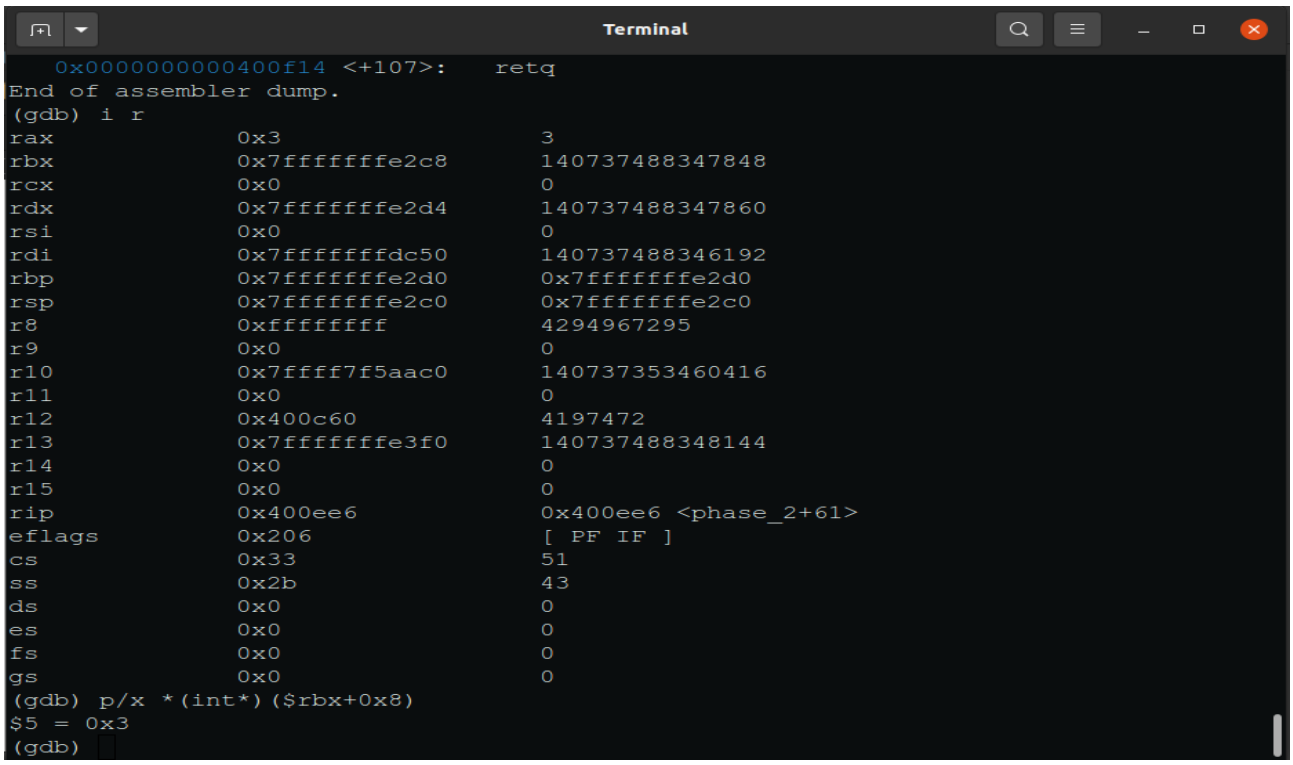
```
Terminal
   0x0000000000400f14 <+107>:    retq
End of assembler dump.
(gdb) i r
rax            0x1                 1
rbx            0x7fffffffe2c0      140737488347840
rcx            0x0                 0
rdx            0x7fffffffe2d4      140737488347860
rsi            0x0                 0
rdi            0x7fffffffdc50      140737488346192
rbp            0x7fffffffe2d0      0x7fffffffe2d0
rsp            0x7fffffffe2c0      0x7fffffffe2c0
r8             0xffffffff          4294967295
r9             0x0                 0
r10            0x7ffff7f5aac0      140737353460416
r11            0x0                 0
r12            0x400c60            4197472
r13            0x7fffffffe3f0      140737488348144
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) p/x *(int*)($rbx+0x8)
$3 = 0x1
(gdb)
```

As it resulted to be true, it jumps on line 73 and performs add operation with $0x4 and %rbx and followed by compare. Now lets check the value of %rbp and%rbx and figure out next 4th digit. User input for 4th digit was 2 and this was compare to value of %eax after a iteration of code. Again result happen to be true and 4th digit is identified as 2 as show below.

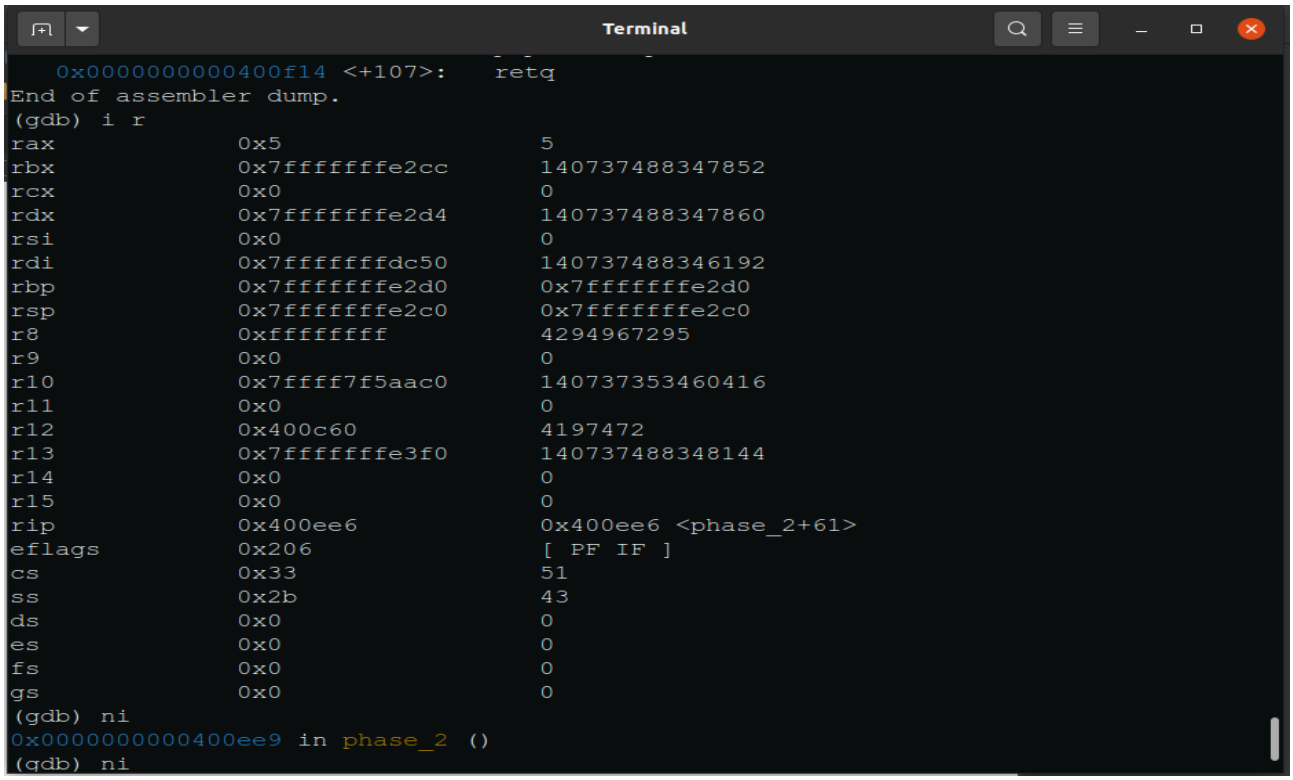


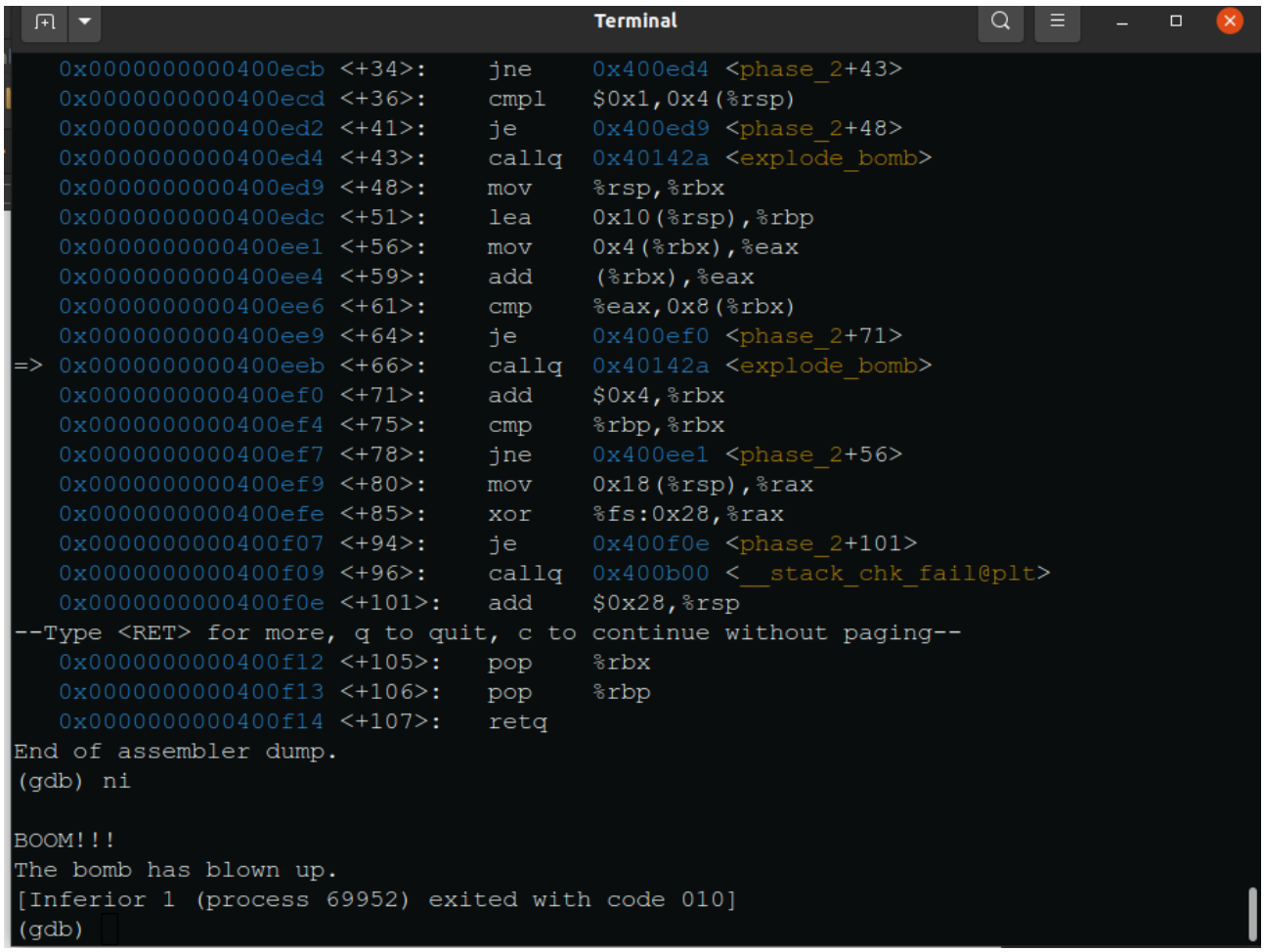


The user input for 5th digit was 3 and next compare was executed and and after looping through the loop the value of %eax and user input matched as shown below.

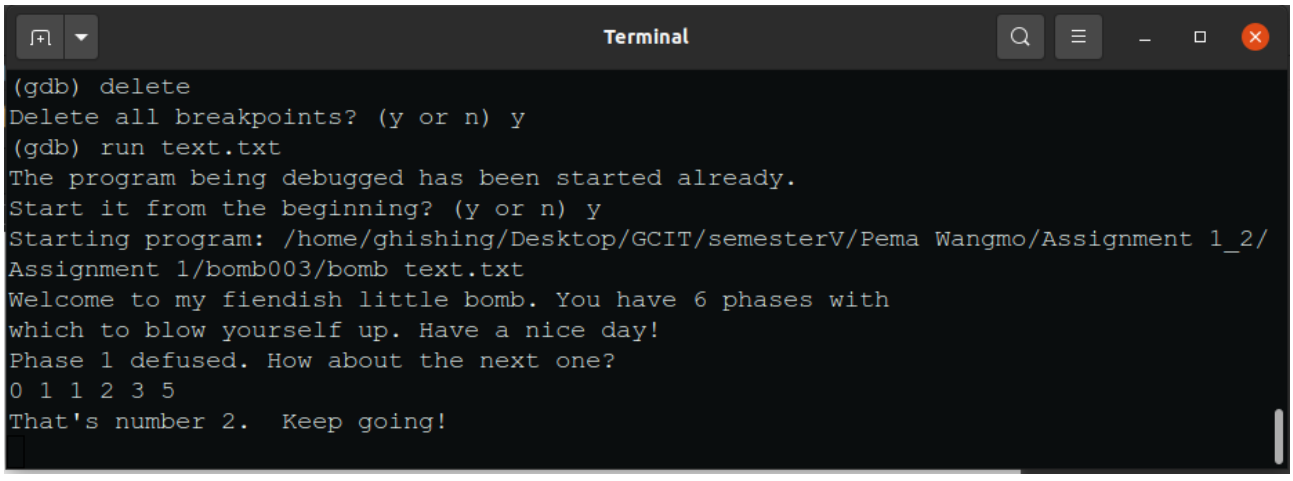


As it happens to be true, it jumps on line 71 and perform the add operation and iterate in the loop for a while. After several iteration, the value of eax is 5 and user input value was 4, thus the user input and predefined valve didnt match. As the result bomb has blow up but good new is that we have found out last digit i.e., 5.





As the bomb has exploded, now lets try to run the program again and input the values that was extracted i.e., 0,1,1,2,3,5. All the break point was deleted as we were sure that bomb wouldn't explode anymore.



After providing the extracted number , the bomb was defuse and phase 2 was over.