EMBEDDED SCHOOL

파이썬 - HW4

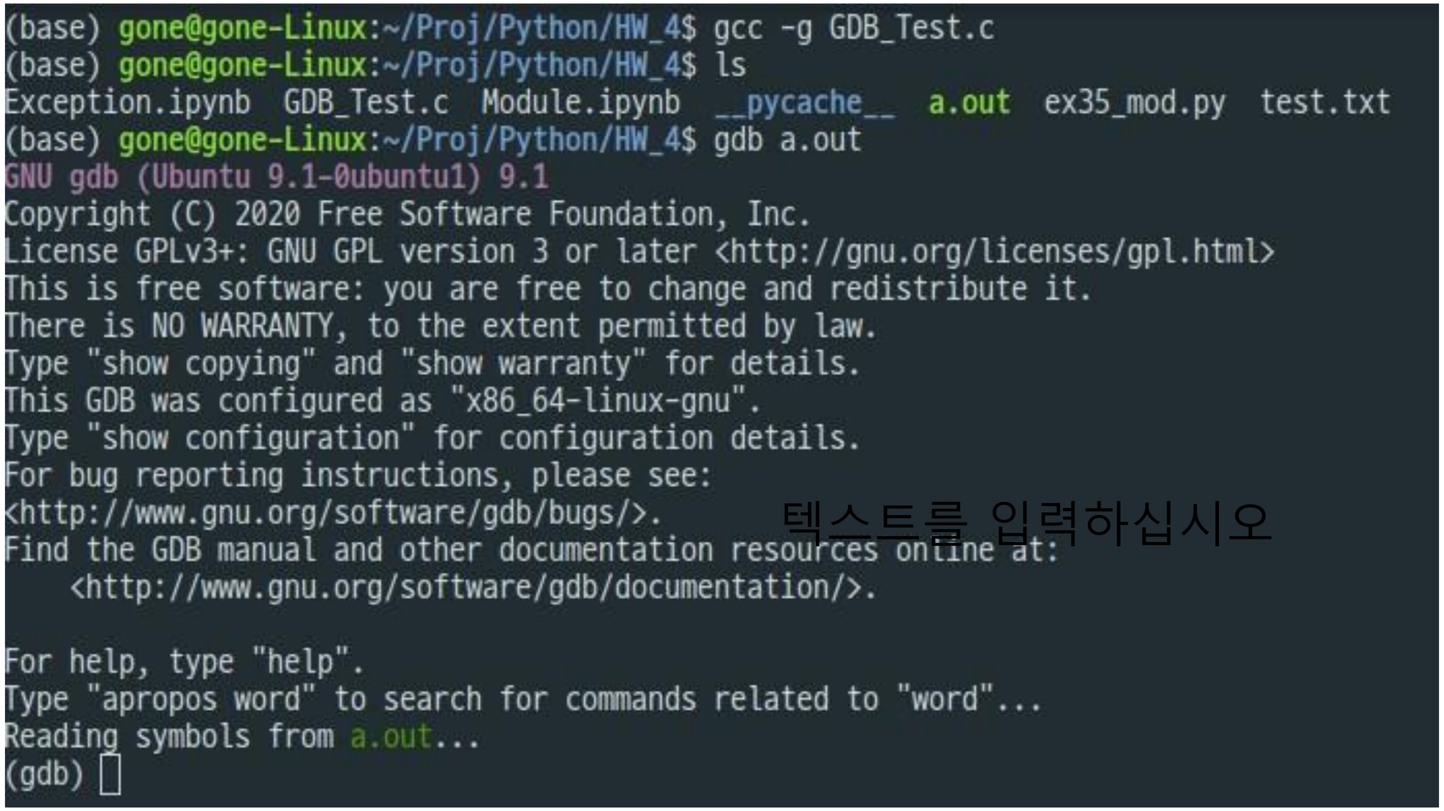
임베디드스쿨1기

Lv1과정

2020. 08. 15

박성환

1. GDB 디버깅



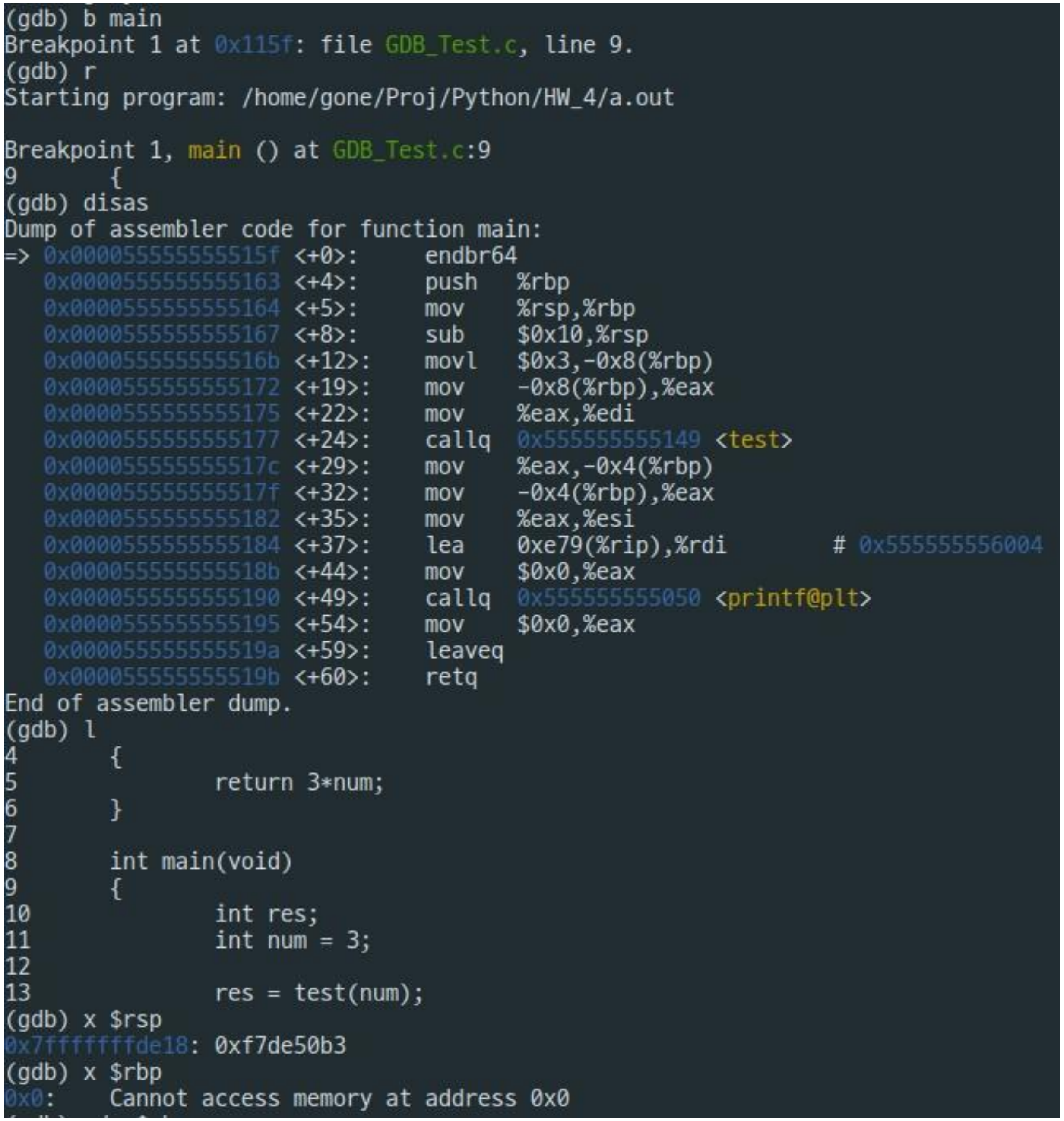
```
(base) gone@gone-Linux:~/Proj/Python/HW_4$ gcc -g GDB_Test.c
(base) gone@gone-Linux:~/Proj/Python/HW_4$ ls
Exception.ipynb  GDB_Test.c  Module.ipynb  __pycache__  a.out  ex35_mod.py  test.txt
(base) gone@gone-Linux:~/Proj/Python/HW_4$ gdb a.out
GNU gdb (Ubuntu 9.1-0ubuntu1) 9.1
Copyright (C) 2020 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 a.out...
(gdb) 
```

텍스트를 입력하십시오

- gcc –g GDB_Test.c :
디버깅할 c파일(a.out default로 생성)
- gdb a.out :
a.out 디버깅 실행

1. GDB 디버깅

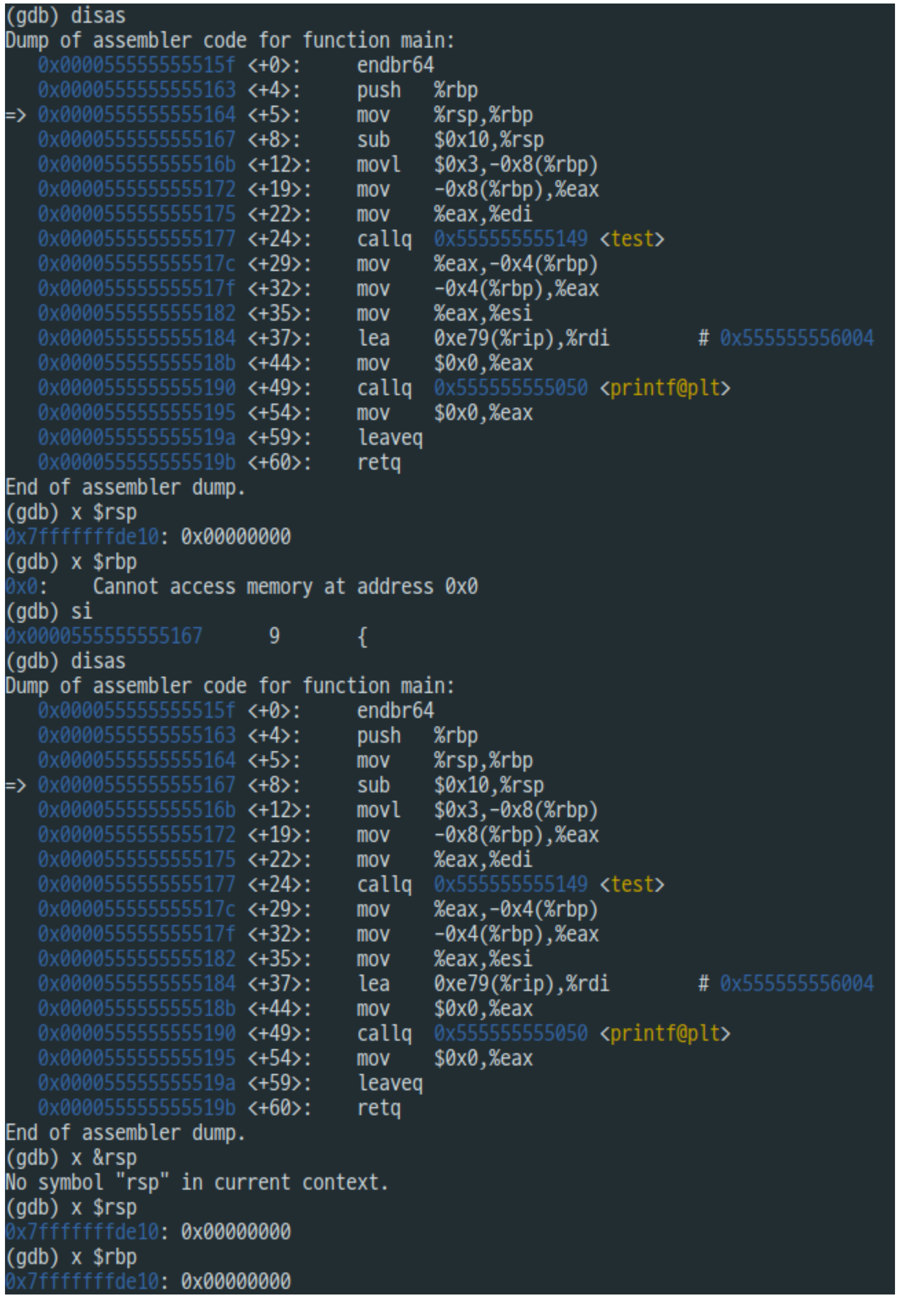


```
(gdb) b main
Breakpoint 1 at 0x115f: file GDB_Test.c, line 9.
(gdb) r
Starting program: /home/gone/Proj/Python/HW_4/a.out

Breakpoint 1, main () at GDB_Test.c:9
9       {
(gdb) disas
Dump of assembler code for function main:
=> 0x000055555555515f <+0>:     endbr64
   0x0000555555555163 <+4>:     push   %rbp
   0x0000555555555164 <+5>:     mov    %rsp,%rbp
   0x0000555555555167 <+8>:     sub    $0x10,%rsp
   0x000055555555516b <+12>:    movl   $0x3,-0x8(%rbp)
   0x0000555555555172 <+19>:    mov    -0x8(%rbp),%eax
   0x0000555555555175 <+22>:    mov    %eax,%edi
   0x0000555555555177 <+24>:    callq  0x555555555149 <test>
   0x000055555555517c <+29>:    mov    %eax,-0x4(%rbp)
   0x000055555555517f <+32>:    mov    -0x4(%rbp),%eax
   0x0000555555555182 <+35>:    mov    %eax,%esi
   0x0000555555555184 <+37>:    lea    0xe79(%rip),%rdi        # 0x555555556004
   0x000055555555518b <+44>:    mov    $0x0,%eax
   0x0000555555555190 <+49>:    callq  0x555555555050 <printf@plt>
   0x0000555555555195 <+54>:    mov    $0x0,%eax
   0x000055555555519a <+59>:    leaveq
   0x000055555555519b <+60>:    retq
End of assembler dump.
(gdb) l
4       {
5               return 3*num;
6       }
7
8       int main(void)
9       {
10              int res;
11              int num = 3;
12
13              res = test(num);
(gdb) x $rsp
0x7fffffffde18: 0xf7de50b3
(gdb) x $rbp
0x0:    Cannot access memory at address 0x0
```

- b main:
main함수 에 breakpoint
- r :
실행
- disas
어셈블리 보기
- l
c코드 해당 위치 확인
- X $rsp
  X $rbp
해당 메모리 주소 : 값

1. GDB 디버깅



```
(gdb) disas
Dump of assembler code for function main:
   0x000055555555515f <+0>:     endbr64
   0x0000555555555163 <+4>:     push   %rbp
=> 0x0000555555555164 <+5>:     mov    %rsp,%rbp
   0x0000555555555167 <+8>:     sub    $0x10,%rsp
   0x000055555555516b <+12>:    movl   $0x3,-0x8(%rbp)
   0x0000555555555172 <+19>:    mov    -0x8(%rbp),%eax
   0x0000555555555175 <+22>:    mov    %eax,%edi
   0x0000555555555177 <+24>:    callq  0x555555555149 <test>
   0x000055555555517c <+29>:    mov    %eax,-0x4(%rbp)
   0x000055555555517f <+32>:    mov    -0x4(%rbp),%eax
   0x0000555555555182 <+35>:    mov    %eax,%esi
   0x0000555555555184 <+37>:    lea    0xe79(%rip),%rdi        # 0x555555556004
   0x000055555555518b <+44>:    mov    $0x0,%eax
   0x0000555555555190 <+49>:    callq  0x555555555050 <printf@plt>
   0x0000555555555195 <+54>:    mov    $0x0,%eax
   0x000055555555519a <+59>:    leaveq
   0x000055555555519b <+60>:    retq
End of assembler dump.
(gdb) x $rsp
0x7fffffffde10: 0x00000000
(gdb) x $rbp
0x0:    Cannot access memory at address 0x0
(gdb) si
0x0000555555555167      9       {
(gdb) disas
Dump of assembler code for function main:
   0x000055555555515f <+0>:     endbr64
   0x0000555555555163 <+4>:     push   %rbp
   0x0000555555555164 <+5>:     mov    %rsp,%rbp
=> 0x0000555555555167 <+8>:     sub    $0x10,%rsp
   0x000055555555516b <+12>:    movl   $0x3,-0x8(%rbp)
   0x0000555555555172 <+19>:    mov    -0x8(%rbp),%eax
   0x0000555555555175 <+22>:    mov    %eax,%edi
   0x0000555555555177 <+24>:    callq  0x555555555149 <test>
   0x000055555555517c <+29>:    mov    %eax,-0x4(%rbp)
   0x000055555555517f <+32>:    mov    -0x4(%rbp),%eax
   0x0000555555555182 <+35>:    mov    %eax,%esi
   0x0000555555555184 <+37>:    lea    0xe79(%rip),%rdi        # 0x555555556004
   0x000055555555518b <+44>:    mov    $0x0,%eax
   0x0000555555555190 <+49>:    callq  0x555555555050 <printf@plt>
   0x0000555555555195 <+54>:    mov    $0x0,%eax
   0x000055555555519a <+59>:    leaveq
   0x000055555555519b <+60>:    retq
End of assembler dump.
(gdb) x &rsp
No symbol "rsp" in current context.
(gdb) x $rsp
0x7fffffffde10: 0x00000000
(gdb) x $rbp
0x7fffffffde10: 0x00000000
```

Rsp : stack pointer
Rbp: break pointer(pc)

1. push 동작을 통해
RSP Base Address에 RBP Address값이 저장됨(64bit 포인터 크기 8Byte만큼)

2. mov 동작을 통해
RBP Address가 RSP Address와 동일 위치를 가리킴

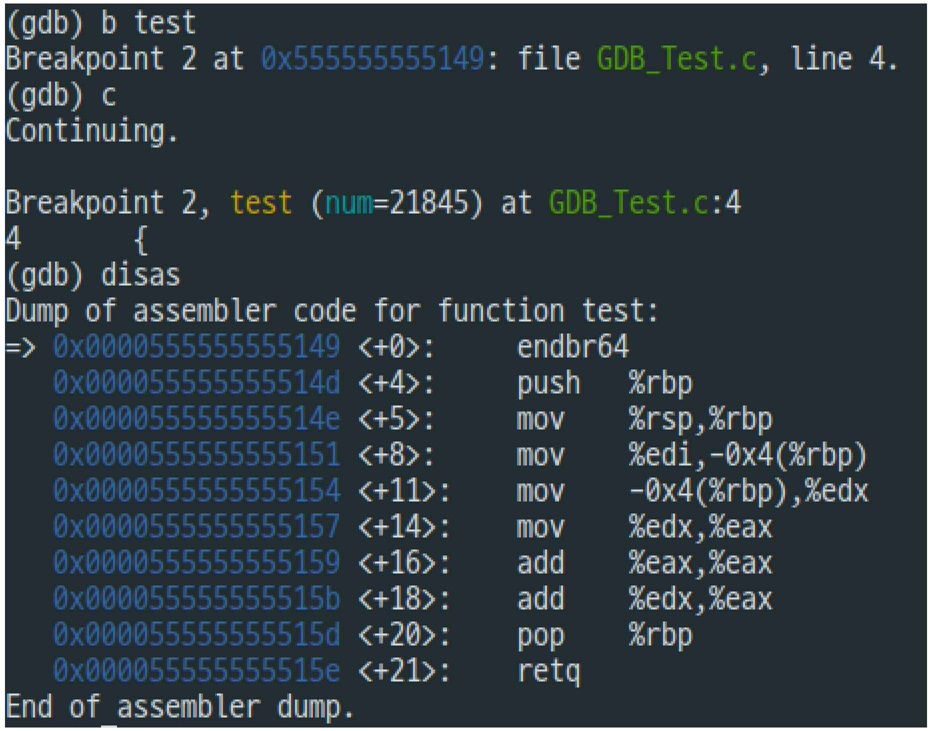
3. sub 동작을 통해
컴파일러에 의해 할당된 main 함수 stack 공간을 할당함(여기서는 0x10만큼)

4. RSP가 주소가 감소하면서 Stack에 Data 쌓임

5. POP하면 RSP 처음에 저장했던 RBP 주소를 현재 RBP값으로 불러옴(복귀)

##함수 호출시 메모리에 경계선의 의미는 호출시 RSP와 RBP가 바뀌고 그게 같아지면서 경계선과 같은 의미

1. GDB 디버깅



```
(gdb) b test
Breakpoint 2 at 0x555555555149: file GDB_Test.c, line 4.
(gdb) c
Continuing.

Breakpoint 2, test (num=21845) at GDB_Test.c:4
4       {
(gdb) disas
Dump of assembler code for function test:
=> 0x0000555555555149 <+0>:     endbr64
   0x000055555555514d <+4>:     push   %rbp
   0x000055555555514e <+5>:     mov    %rsp,%rbp
   0x0000555555555151 <+8>:     mov    %edi,-0x4(%rbp)
   0x0000555555555154 <+11>:    mov    -0x4(%rbp),%edx
   0x0000555555555157 <+14>:    mov    %edx,%eax
   0x0000555555555159 <+16>:    add    %eax,%eax
   0x000055555555515b <+18>:    add    %edx,%eax
   0x000055555555515d <+20>:    pop    %rbp
   0x000055555555515e <+21>:    retq
End of assembler dump.
```

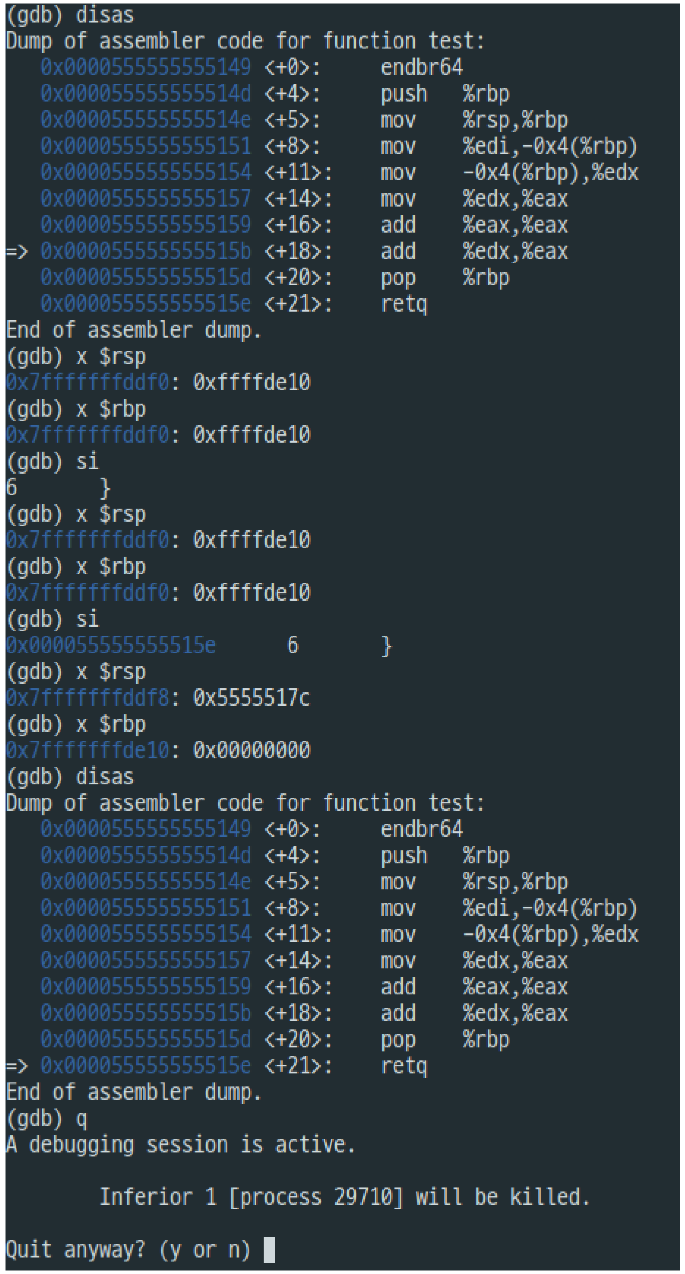
- b test:

test함수 에 breakpoint

- c:

c는 이미 한번 r(run)으로 실행된 상태에서 다음 break point 까지 실행시 사용

1. GDB 디버깅



```
(gdb) disas
Dump of assembler code for function test:
   0x0000555555555149 <+0>:     endbr64
   0x000055555555514d <+4>:     push   %rbp
   0x000055555555514e <+5>:     mov    %rsp,%rbp
   0x0000555555555151 <+8>:     mov    %edi,-0x4(%rbp)
   0x0000555555555154 <+11>:    mov    -0x4(%rbp),%edx
   0x0000555555555157 <+14>:    mov    %edx,%eax
   0x0000555555555159 <+16>:    add    %eax,%eax
=> 0x000055555555515b <+18>:    add    %edx,%eax
   0x000055555555515d <+20>:    pop    %rbp
   0x000055555555515e <+21>:    retq
End of assembler dump.
(gdb) x $rsp
0x7fffffffddf0: 0xffffde10
(gdb) x $rbp
0x7fffffffddf0: 0xffffde10
(gdb) si
6       }
(gdb) x $rsp
0x7fffffffddf0: 0xffffde10
(gdb) x $rbp
0x7fffffffddf0: 0xffffde10
(gdb) si
0x000055555555515e      6       }
(gdb) x $rsp
0x7fffffffddf8: 0x5555517c
(gdb) x $rbp
0x7fffffffde10: 0x00000000
(gdb) disas
Dump of assembler code for function test:
   0x0000555555555149 <+0>:     endbr64
   0x000055555555514d <+4>:     push   %rbp
   0x000055555555514e <+5>:     mov    %rsp,%rbp
   0x0000555555555151 <+8>:     mov    %edi,-0x4(%rbp)
   0x0000555555555154 <+11>:    mov    -0x4(%rbp),%edx
   0x0000555555555157 <+14>:    mov    %edx,%eax
   0x0000555555555159 <+16>:    add    %eax,%eax
   0x000055555555515b <+18>:    add    %edx,%eax
   0x000055555555515d <+20>:    pop    %rbp
=> 0x000055555555515e <+21>:    retq
End of assembler dump.
(gdb) q
A debugging session is active.

        Inferior 1 [process 29710] will be killed.

Quit anyway? (y or n)
```

- q:
gdb 종료 가능

감사합니다.