에디로봇아카데미 임베디드 마스터 Lv1 과정 Ch3.

> 제 3기 2022. 01. 13 여건영

## 핵심 규칙

1. 스택은 거꾸로 자란다.

rsp의 메모리 주소가 작아질 수록 스택의 사이즈는 커진다.

2. 메모리 단위 연산은 포인터 크기 단위로 이루어진다.

ALU(CPU – 64bit – 8byte)

메모리 한 칸의 크기는 1byte 이다.

64비트 운영체제는 메모리 한 칸의 주소를 64비트로 표현하며 이는 8byte와 같은 의미이고, 메모리 주소를 8byte로 표현하기 때문에 포인터(주소를 가리키는 변수)의 크기 또한 8byte이다.

3. ax의 용도 : 함수의 리턴값이 저장되며 연산용으로도 활용 가능

bp의 용도 : 현재 스택의 기준점 sp의 용도 : 현재 스택의 최상위

ip의 용도 : 다음에 실행 할 instruction(명령어)의 주소

2. push %rbp

push 명령어 : 현재 스택의 최상위 메모리(rsp) 값에 뒤 값(여기서 %rbp)을 저장하는 명령어

즉, 현재 스택의 최상위 메모리(rsp) 주소에서 %rbp의 크기만큼 push해서 그 값을 저장하라

주소 메모리

0x7fffffffdfa8 0xf7deb0b3

0x7fffffffdfa0 0x00000000

```
(gdb) si
0x0000555555555560 15 {
(gdb) x $rbp
0x0: Cannot access memory at address 0x0
(gdb) x $rsp
0x7fffffffdfa0: 0x00000000
(gdb) x/4wx $rsp
0x7fffffffdfa0: 0x00000000 0x00000000 0xf7deb0b3 0x00007fff
(gdb) ■
```

// rsp 주소 값이 push 되는 %rbp의 크기(8byte)만큼 작아지고, // 현재 rsp 메모리 값에 '0x0' 이 들어간 것을 확인할 수 있다.

1. 현재 스택의 최상위 주소에 위치한 rsp가 '0x7fffffffdfa8' 라는 가상의 주소 값을 가지며, rbp는 아직 메모리 값이 정해지지 않았으며, 주소 값은 '0x0'이다.

주소 메모리

0x7ffffffdfa8

0xf7deb0b3

```
(adb) b main
breakpoint r at exists: file function.c, line 15.
(qdb) r
Starting program: /nome/yeo/EmbeddedMasterLv1/37//GYY/c/ch3/function
Breakpoint 1, main () at function.c:15
(adb) si
                        15
(qdb) disas main
     of accombles code for function main:
  0x00005555555555515b <+0>:
                                 endbr64
                                 push
                                       %гьр
                                        %rsp,%rbp
   0x000005555555555160 <+5>:
                                 mov
                                        $0x10,%rsp
   0x00005555555555163 <+8>:
                                 sub
                                 movl
                                        $0x3,-0x8(%rbp)
   0x0000055555555555167 <+12>:
                                        -0x8(%rbp), %eax
  0x0000555555555516e <+19>:
                                 MOV
  0x000005555555555171 <+22>:
                                        %eax,%edi
                                 MOV
                                 callq 0x5555555555149 <test func>
   0x000055555555555173 <+24>:
  0x00005555555555178 <+29>:
                                 mov
                                        %eax,-0x4(%rbp)
  0x0000555555555517b <+32>:
                                        -0x4(%rbp),%eax
                                 MOV
                                        %eax.%esi
  0x00005555555555517e <+35>:
                                 mov
                                        0xe7d(%rip),%rdi
                                                                 # 0x55555556004
   0x00005555555555180 <+37>:
                                 lea
  0x00005555555555187 <+44>:
                                        $0x0, %eax
                                 MOV
  0x0000555555555518c <+49>:
                                 callq 0x55555555555050 <printf@plt>
   0x000055555555555191 <+54>:
                                        $0x0, %eax
                                 MOV
  0x000055555555555196 <+59>:
                                 leaveg
                                 retq
   0x000055555555555197 <+60>:
End of assembler dump
(gdb) x $rsp
0x7ffffffffdfa8: 0xf7deb0b3
(gdb) x $rbp
     Cannot access memory at address 0x0
```

3. mov %rsp,%rbp

mov : 좌측 값을 우측으로 복사 or 대 mov : 데이터 복사를 담당하는 명령어 레지스터 → 레지스터 메모리 < - > 레지스터 상수 → 레지스터 or 메모리 위 관계와 같이 데이터를 복사 or 대입 할 수 있다. 메모리 to 메모리는 불가

주소

메모리

0x7ffffffdfa8 0x7ffffffdfa0 0xf7deb0b3

0x0000000

main함수의 rbp

```
// %rsp 값이 %rbp 값으로 대입되었다.
// stack frame이 생성
```

```
(qdb) disas main
Dump of assembler code for function main:
   0x000005555555555515b <+0>:
                                  endbr64
                                        %rbp
   0x0000555555555515f <+4>:
                                 push
                                         %rsp,%rbp
   0x00005555555555160 <+5>:
                                 MOV
                                         $0x10,%rsp
=> 0x00005555555555163 <+8>:
                                        $0x3,-0x8(%rbp)
                                 movl
                                         -0x8(%rbp), %eax
   0x00000555555555516e <+19>:
                                 mov
   0x00005555555555171 <+22>:
                                         %eax, %edi
                                 mov
   0x000055555555555173 <+24>:
                                 calla 0x555555555149 <test func>
                                         %eax,-0x4(%rbp)
                                 MOV
                                         -0x4(%rbp).%eax
                                 mov
   0x0000555555555517e <+35>:
                                         %eax.%esi
                                 mov
                                         0xe7d(%rip),%rdi
                                                                  # 0x55555556004
   0x00005555555555180 <+37>:
                                  lea
   0x0000055555555555187 <+44>:
                                 mov
                                         $0x0, %eax
                                  callq 0x5555555555050 <printf@plt>
   0x0000555555555518c <+49>:
   0x00005555555555191 <+54>:
                                         $0x0,%eax
                                  MOV
                                 leaveg
   0x00005555555555196 <+59>:
   0x00005555555555197 <+60>:
                                 retq
nu or asserbler durip.
(adb) x Srbp
    ffffffdfa0: 0x00000000
 adb) x $rsp
        ffdfa0: 0x00000000
```

4. sub %0x10,%rsp

뺄셈 명령, 현재 %rsp 값을 0x10(16byte) 만큼 빼겠다는 의미

주소

메모리

0x7fffffffdfa80xf7deb0b30x7fffffffdfa00x00000000스택 (지역 변수 공간)0x7fffffffdf900xffffe090

```
// %rsp 값이 0x10 만큼 작아졌다는 것 확인
// p $rbp-$rsp: rbp와 rsp의 차이가 16이라는 것을 확인
// 지역 변수 공간의 메모리와 그 값을 확인
```

```
Dump of assembler code for function main:
   0x00000555555555515b <+0>:
                                  endbr64
   0x000005555555555515f <+4>:
                                 push
                                         %гьр
                                         %rsp,%rbp
   0x00005555555555160 <+5>:
                                 mov
   0x00005555555555163 <+8>:
                                         $0x10,%rsp
=> 0x000005555555555167 <+12>:
                                 movl
                                         S0x3.-0x8(%rbp)
                                         -0x8(%rbp),%eax
                                 mov
                                         %eax,%edi
   0x00005555555555171 <+22>:
                                 mov
   0x00005555555555173 <+24>:
                                 calla 0x5555555555149 <test func>
                                         %eax,-0x4(%rbp)
                                 MOV
                                         -0x4(%rbp),%eax
   0x00000555555555517b <+32>:
                                 mov
   0x00000555555555517e <+35>:
                                         %eax.%esi
                                         0xe7d(%rip),%rdi
   0x00005555555555180 <+37>:
                                 lea
                                                                  # 0x55555556004
   0x000005555555555187 <+44>:
                                 MOV
                                         $0x0, %eax
                                 callo 0x55555555555050 <printf@plt>
   0x000005555555555518c <+49>:
                                 mov
                                         S0x0.%eax
   0x000005555555555191 <+54>:
   0x00005555555555196 <+59>:
                                 leaveg
   0x00005555555555197 <+60>:
                                 retq
End of assembler dump.
(gdb) x $rbp
ex7ffffffffdfa0: 0x00000000
(qdb) x $rsp
0x7fffffffffdf90: 0xffffe090
                                          rsp
(qdb) p $rbp-$rsp
52 = 16
(qdb) x/6wx $rsp
   fffffffdf90: 0xffffe090
                                  0x00007fff
                                                   0x00000000
                                                                   0x00000000
0x7ffffffffdfa0: 0x00000000
                                  0x00000000
(dbp)
```

rbp

5. movl \$0x3, -0x8(%rbp)

I 이 들어가면 4byte 처리를 의미 q 가 들어가면 8byte 처리를 하겠다는 의미 0x3을 %rbp에서 -0x8만큼 위치한 메모리 값에 대입한다.

주소

메모리

```
      0x7fffffffdfa8
      0xf7deb0b3

      0x7fffffffdfa0
      0x00000000

      ...
      0x7fffffffdf98
      0x00000003

      ...
      0x7ffffffffdf90
      0xffffe090
```

// \$rbp-0x8의 메모리 주소와 값 변화를 확인 // int type으로 선언한 변수에 3이 들어

```
0x0000555555555163 <+8>: sub $0x10,%rsp
0x0000555555555167 <+12>: movl $0x3,-0x8(%rbp)
=> 0x000055555555516e <+19>: mov -0x8(%rbp),%eax
0x00005555555555171 <+22>: mov %eax,%edi
```

```
End of assembler dump.
(gdb) x $rbp-0x8
0x7fffffffdf98: 0x00000003
(gdb) x/6wx $rsp
0x7fffffffdf90: 0xffffe090 0x00007fff 0x00000003 0x00000000
0x7ffffffffdfa0: 0x00000000 0x00000000
(gdb)
```

6. mov -0x8(%rbp),%eax

eax 레지스터(4byte 레지스터) 주소에 %rbp-0x8에 위치한 메모리 값, 0x3을 %eax에 대입 이것은 연산에 사용되며, 연산 이후 ax 레지스터의 변경된 값을 획

주소	메모리	
0x7ffffffdfa8	0xf7deb0b3	
0x7ffffffdfa0	0x00000000	
0x7ffffffdf98	0x00000003	
0x7ffffffdf90	0xffffe090	

// eax 레지스터의 주소 값 변화 확인

```
(adb) x Seax
                Cannot access memory at address 0x5555515b
                        17
                                         int result = test func(num);
(odb) disas main
Dump of assembler code for function main:
   0x0000555555555515b <+0>:
                                 endbr64
   0x000055555555555515f <+4>:
                                 push
                                       %гьр
   0x00005555555555160 <+5>:
                                        %rsp.%rbp
                                 MOV
                                        $0x10,%rsp
   0x00005555555555163 <+8>:
                                 sub
                                 movl
                                        $0x3,-0x8(%rbp)
   0x0000555555555516e <+19>:
                                        -0x8(%rbp),%eax
                                 MOV
                                        %eax, %edi
  0x000005555555555171 <+22>:
                                 MOV
   0x000005555555555173 <+24>:
                                 callq 0x555555555149 <test func>
  0x00005555555555178 <+29>:
                                        %eax.-0x4(%rbp)
                                 MOV
                                        -0x4(%rbp),%eax
   0x0000555555555517b <+32>:
                                 MOV
   0x0000555555555517e <+35>:
                                 mov
                                        %eax, %esi
                                        0xe7d(%rip).%rdi
   0x00005555555555180 <+37>:
                                 lea
                                                                 # 0x55555556004
                                        $0x0, %eax
   0x00005555555555187 <+44>:
                                 callo 0x55555555555050 <printf@plt>
   0x0000555555555518c <+49>:
   0x000005555555555191 <+54>:
                                 MOV
                                        S0x0.%eax
                                 leaveg
  0x00005555555555196 <+59>:
   0x000055555555555197 <+60>:
                                 retq
End of assembler dump.
(qdb) x Seax
        Cannot access memory at address 0x3
```

7. mov %eax,%edi

%eax 값을 %edi 값에 대입한다.

메모리		
0xf7deb0b3		
0x00000000		
0x00000003		
0xffffe090		

// edi 레지스터의 주소 값 변화 확인

```
(qdb) x $edi
0x1: Cannot access emory at address 0x1
(qdb) si
                        17
                                         int result = test func(num);
(adb) disas main
Dump of assembler code for function main:
   0x000005555555555515b <+0>:
                                 endbr64
   0x000005555555555515f <+4>:
                                 push
                                       %rbp
                                        %rsp,%rbp
   0x000005555555555160 <+5>:
                                        $0x10,%rsp
   0x000055555555555163 <+8>:
                                        $0x3,-0x8(%rbp)
   0x000005555555555167 <+12>:
                                 movl
   0x0000555555555516e <+19>:
                                        -0x8(%rbp),%eax
   0x00005555555555171 <+22>:
                                        %eax.%edi
=> 0x000005555555555173 <+24>:
                                 callq 0x5555555555149 <test func>
   0x00005555555555178 <+29>:
                                        %eax,-0x4(%rbp)
   0x0000555555555517b <+32>:
                                        -0x4(%rbp),%eax
                                 MOV
   0x0000555555555517e <+35>:
                                        %eax.%esi
                                 MOV
                                        0xe7d(%rip),%rdi
   0x00005555555555180 <+37>:
                                                                 # 0x55555556004
   0x000055555555555187 <+44>:
                                        $0x0, %eax
                                 callq 0x55555555555050 <printf@plt>
   0x0000555555555518c <+49>:
   0x00005555555555191 <+54>:
                                        $0x0,%eax
                                 MOV
   0x000055555555555196 <+59>:
                                 leaveg
End of assembler dump.
(adb) x Seax
        Cannot access memory at address 0x3
(adb) x Sedi
        Cannot access memory at address 0x3
(qdb) x $rsp
     ffffffdf90: 0xffffe090
```

8. callq 0x555555555149 <test\_func> call은 push+jump로 구성되어 있음 함수 호출이 끝난 이후에 실행해야 할 어셈블리 명령어의 주소 값을 push 하며 저장한다. 이후 함수 호출을 수행 하기 위해 jmp 한다.

주소	메모리	
0x7ffffffdfa8	0xf7deb0b3	
0x7ffffffdfa0	0x00000000	
0x7ffffffdf98	0x00000003	
0x7ffffffdf90	0xffffe090	
0x7ffffffdf9c	0x00005555	
0x7ffffffdf88	0x55555178	

```
(qdb) si
test func (num=21845) at function.c:10
10
(adb) disas main
Dump of assembler code for function main:
   0x00005555555555515b <+0>:
                                 endbr64
   0x000005555555555515f <+4>:
                                 push
                                         %rbp
   0x00005555555555160 <+5>:
                                 MOV
                                         %rsp.%rbp
                                         $0x10,%rsp
   0x000055555555555163 <+8>:
                                 sub
                                 mov1
                                         $0x3,-0x8(%rbp)
   0x00005555555555167 <+12>:
                                         -0x8(%rbp),%eax
   0x0000555555555516e <+19>:
                                 mov
                                         %eax.%edi
   0x00005555555555171 <+22>:
                                 MOV
                                 callq 0x5555555555149 <test func>
   UXUUUUDDDDDDDDDD1/3 <+24>:
                        +29>:
                                         %eax,-0x4(%rbp)
                                 MOV
                                         -0x4(%rbp),%eax
                                 MOV
                                         %eax.%esi
   0x0000555555555517e <+35>:
                                 MOV
   0x00005555555555180 <+37>:
                                 lea
                                         0xe7d(%rip),%rdi
                                                                  # 0x555555556004
   0x00005555555555187 <+44>:
                                 mov
                                         S0x0.%eax
                                 calla 0x55555555555050 <printf@plt>
   0x0000555555555518c <+49>:
   0x00005555555555191 <+54>:
                                 MOV
                                         $0x0, %eax
                                 leaveg
   0x000055555555555196 <+59>:
  0x00005555555555197 <+60>:
                                 retq
ind of assembler dump.
 qdb) x $rsp
 x7ffffffffdf88: 0x55555178
 qdb) x $rbp
 x7fffffffdfa0: 0x00000000
 qdb) x/8wx $rsp
                                                  0xffffe090
 x7ffffffffff88: 0x55555178
                                 0x00005555
                                                                   0x00007fff
 x7ffffffffdf98: 0x000000003
                                 0x00000000
                                                  0x00000000
                                                                   0x00000000
(dab)
```

// 복귀주소의 크기만큼 push 된 것을 확인 (8byte) // rsp 메모리 값에 복귀 주소 들어간 것 확인

9. push %rbp

현재 최상위 메모리(rsp)에 rbp 레지스터의 주 값을 push 한다.

주소	메모리	
0x7ffffffdfa8	0xf7deb0b3	
0x7ffffffdfa0	0x00000000	
0x7ffffffdf98	0x00000003	
0x7ffffffdf90	0xffffe090	
0x7ffffffdf9c	0x00005555	
0x7ffffffdf88	0x55555178	
0x7ffffffdf84	0x00007fff	
0x7ffffffdf80	0xffffdfa0	

```
// rsp가 8byte 만큼 push 되고,
// 이전 함수의 rbp 주소 값이 들어간 것 확인
```

```
Dump of assembler code for function test func:
   0x00005555555555149 <+0>:
                               endbr64
  0x0000555555555514d <+4>:
                               push
                                     %гьр
=> 0x000055555555514e <+5>:
                                     %rsp.%rbp
                              MOV
  0x000005555555555151 <+8>:
                                     %edi,-0x4(%rbp)
                               MOV
  mov
                                     -0x4(%rbp),%eax
  0x0000055555555555157 <+14>:
                               add
                                     %eax, %eax
  pop
                                     %rbp
   0x00005555555555515a <+17>:
                               reta
Ella of assembler damp.
(gdb) x $rsp
 x7fffffffdf80: 0xffffdfa0
(adb) x Srbp
 x7ffffffffdfa0: 0x00000000
(gdb) x/10wx $rsp
 x7ffffffffdf80: 0xffffdfa0
                               0x00007fff
                                              0x55555178
                                                             0x00005555
     fffffdf90: 0xffffe090
                               0x00007fff
                                              0x00000003
                                                             0x00000000
 x7ffffffffdfa0: 0x00000000
                               0x00000000
(gdb)
```

10. mov %rsp,%rbp

rsp의 메모리 주소 값을 rbp의 메모리 주소 값에 대입

메모리 주소 0x7fffffffdfa8 0xf7deb0b3 0x7fffffffdfa0 0x00000000 0x0000003 0x7ffffffdf98 0x7ffffffdf90 0xffffe090 0x7ffffffdf9c 0x00005555 0x7ffffffdf88 0x55555178 0x7fffffffdf84 0x00007fff 0x7fffffffdf80 0xffffdfa0

```
(adb) x Srbp
         fdfa0: 0x00000000
(qdb) x $rsp
     fffffdf80: 0xffffdfa0
(adb) si
                        10
(adb) disas
Dump of assembler code for function test func:
   0x00005555555555149 <+0>:
                                endbr64
   0x000055555555514d <+4>:
                                push
                                       %rbp
   0x0000555555555514e <+5>:
                                       %rsp.%rbp
                                MOV
                                       %edi,-0x4(%rbp)
=> 0x000055555555555151 <+8>:
                                MOV
                                       -0x4(%rbp),%eax
                                MOV
                                add
                                       %eax, %eax
   0x000055555555555157 <+14>:
                                       %rbp
   DOD
   0x0000555555555515a <+17>:
                                reta
End of assembler dump.
(gdb) x $rbp
       fffdf80: 0xffffdfa0
(adb) x Srsp
     fffffdf80: 0xffffdfa0
(dbp)
```

// 현재 rsp 메모리 주소는 test\_func 함수의 rbp

// test\_func의 rbp와 같으며, 안에 값은 이전 함수의 rbp임을 알 수 있다.

11. mov %edi, -0x4(%rbp)

edi 주소 값을 rbp-0x4 위치에 대입 한다.

주소	메모리	
0x7ffffffdfa8	0xf7deb0b3	
0x7ffffffdfa0	0x00000000	
0x7ffffffdf98	0x00000003	
0x7ffffffdf90	0xffffe090	
0x7ffffffdf9c	0x00005555	
0x7ffffffdf88	0x55555178	
0x7ffffffdf84	0x00007fff	
0x7ffffffdf80	0xffffdfa0	
0x7ffffffdf7c	0x00000003	

```
(qdb) x Sedi
        Cannot access memory at address 0x3
(gdb) x $rbp-0x4
0x7fffffffffffc: 0x00005555
(gdb) si
11
                return num * 2:
(qdb) x $rbp-0x4
0x7ffffffffffc: 0x00000003
(gdb) disas
Dump of assembler code for function test func:
                                 endbr64
   0x00005555555555149 <+0>:
   0x0000555555555514d <+4>:
                                 push
                                        %гьр
   0x0000555555555514e <+5>:
                                 MOV
                                        %rsp,%rbp
                                        %edi,-0x4(%rbp)
   0x0000555555555555151 <+8>:
                                 MOV
=> 0x000055555555555154 <+11>:
                                        -0x4(%rbp), %eax
                                 MOV
   0x000055555555555157 <+14>:
                                 add
                                        %eax, %eax
   0x000055555555555159 <+16>:
                                        %гьр
                                 pop
   0x00005555555555515a <+17>:
                                 retq
End of assembler dump.
(gdb)
```

// (gdb) si 실행 전 후로 \$rbp-0x4의 메모리 값 변화 확인

```
-0x4(%rbp),%eax
12. mov
rbp-0x4에 위치한 메모리 값을
eax 레지스터 주소 값에 대입
    주소
                   메모리
0x7fffffffdfa8
                 0xf7deb0b3
0x7fffffffdfa0
                0x00000000
                0x0000003
0x7ffffffdf98
                  0xffffe090
0x7ffffffdf90
0x7fffffffdf9c
                0x00005555
0x7fffffffff88
                0x55555178
0x7fffffffffdf84
                 0x00007fff
0x7ffffffdf80
                  0xffffdfa0
0x7ffffffffffc
                0x00000003
```

```
(gdb) x $eax

0x3: Cannot access memory at address 0x3
(gdb) si

0x000005555555555557 11 return num * 2;
(gdb) x $eax

0x3: Cannot access memory at address 0x3
(gdb)
```

```
13. add %eax,%eax
```

%eax = %eax + %eax의 의미

주소	메모리	
0x7ffffffdfa8	0xf7deb0b3	
0x7ffffffdfa0	0x00000000	
0x7ffffffdf98	0x00000003	
0x7ffffffdf90	0xffffe090	
0x7ffffffdf9c	0x00005555	
0x7ffffffdf88	0x55555178	
0x7ffffffdf84	0x00007fff	
0x7ffffffdf80	0xffffdfa0	
0x7ffffffdf7c	0x00000003	

```
(qdb) x Seax
       Cannot access memory at address 0x3
(qdb) si
12
(gdb) disas
Dump of assembler code for function test func:
   0x00005555555555149 <+0>:
                              endbr64
   0x0000555555555514d <+4>:
                              push
                                     %rbp
   0x0000555555555514e <+5>:
                                     %rsp,%rbp
                              mov
   0x000055555555555151 <+8>:
                                     %edi,-0x4(%rbp)
                              mov
   0x0000055555555555154 <+11>:
                                     -0x4(%rbp),%eax
                              mov
   0x000055555555555157 <+14>:
                              add
                                     %eax,%eax
pop
                                     %rbp
   0x00005555555555515a <+17>:
                              retq
End of assembler dump
(qdb) x $eax
       Cannot access memory at address 0x6
(gab)
```

// (gdb) si 실행 전 후로 eax 레지스터 주소 값 변화 확

```
14. pop
          %rbp
현재 스택의 최상위(rsp)에서 메모리 값을 빼서
뒤쪽에 메모리나 레지스터에 값을 넘겨 준다.
                  메모리
    주소
0x7fffffffdfa8
                0xf7deb0b3
0x7fffffffdfa0
                0x00000000
0x7ffffffdf98
                0x0000003
0x7ffffffdf90
                 0xffffe090
0x7fffffffdf9c
                0x00005555
0x7ffffffdf88
                0x55555178
0x7fffffffdf84
                 0x00007fff
0x7ffffffdf80
                 0xffffdfa0
0x7fffffffffffc
                0x0000003
```

```
(gdb) x $rbp
    fffffdf80: 0xffffdfa0
(qdb) x $rsp
0x7ffffffffdf80: 0xffffdfa0
(adb) si
(adb) x Srbp
      fffdfa0: 0x00000000
adb) x Srsp
         fdf88: 0x55555178
(qdb) x/10wx 0x7ffffffffdf80
       ffdf80: 0xffffdfa0
                                 0x00007fff
                                                  0x55555178
                                                                   0x00005555
        fdf90: 0xffffe090
                                 0x00007fff
                                                  0x00000003
                                                                   0x00000000
    fffffdfa0: 0x00000000
                                 0x00000000
(adb)
```

// (gdb) si 실행 전, 후의 rbp, rsp 메모리 주소와 값을 확인
// 값을 빼냈기 때문에 rsp 값은 빠진 메모리 값 크기(8byte)만큼 올라가고
// 내부에 있는 값은 rbp 레지스터 주소 값으로 들어가는 것 확인
// 이전 함수의 rbp 주소 값으로 복원되었다.

15. retq

pop \$rip 에 해당하는 연산 현재 스택의 최상위(rsp)에서 값을 빼서 rip에 넘겨 준다.

주소	메모리	
0x7ffffffdfa8	0xf7deb0b3	
0x7ffffffdfa0	0x00000000	
0x7ffffffdf98	0x00000003	
0x7ffffffdf90	0xffffe090	
0x7ffffffdf9c	0x00005555	
0x7ffffffdf88	0x55555178	

```
Dump of assembler code for function test func:
                               endbr64
   0x00005555555555149 <+0>:
   0x0000555555555514d <+4>:
                               push
                                      %rbp
   0x0000555555555514e <+5>:
                               MOV
                                      %rsp.%rbp
                                      %edi,-0x4(%rbp)
   0x00005555555555151 <+8>:
                               mov
                                      -0x4(%rbp),%eax
  0x000055555555555154 <+11>:
                               MOV
                               add
  0x0000055555555555557 <+14>:
                                      %eax.%eax
                                      %rbp
   pop
=> 0x0000555555555515a <+17>:
                               reta
End of assembler dump.
(adb) x $rip
0x55555555515a <test func+17>: 0x1e0ff3c3
(gdb) x/2wx $rsp
0x7ffffffffffdf88: 0x55555178
                               0x00005555
(adb) si
0x00005555555555178 in main () at function.c:17
               int result = test func(num);
17
(adb) x Srip
0x5555555555178 <main+29>:
                               0x8bfc4589
(gdb) x $rsp
  ffffffffdf90: 0xffffe090
```

```
// (gdb) si 실행 전, rsp에는 8byte의 복귀주소가 있으며,
// 그 값이 빠지며 rsp는 8byte만큼 올라가고
// 내부에 있는 값(복귀주소)는 rip 메로리 주소 값으로 들어간다.
// (gdb) si 실행 전, 후 rsp, rip의 값 변화를 확인
```

16. mov %eax, -0x4(%rbp)

%eax의 값을 %rbp-0x4에 위치한 메모리에 대입한다.

주소	메모리	
0x7ffffffdfa8	0xf7deb0b3	
0x7ffffffdfa0	0x00000000	
0x7ffffffdf9c	0x00000006	
0x7ffffffdf98	0x0000003	
0x7ffffffdf90	0xffffe090	

```
(qdb) x $eax
        Cannot access memory at address 0x6
(qdb) x Srbp-0x4
 x7ffffffffffgc: 0x00000000
(qdb) x/6wx $rsp
      fffffdf90: 0xffffe090
                                 0x00007fff
                                                 0x00000003
                                                                  0x00000000
0x7fffffffdfa0: 0x00000000
                                 0x00000000
(gdb) si
                printf("result = %d\n", result);
(gdb) x/6wx $rsp
         ffdf90: 0xffffe090
                                 0x00007fff
                                                 0x00000003
                                                                  0x00000006
 x7fffffffdfa0: 0x00000000
                                 0x00000000
```

```
// eax 메모리 값은 0x6,
// $rbp-0x4의 메모리 값은 0x0 이며,
// si 실행 전, 후로 해당 메모리 값의 변화를 확인
```

17. mov -0x4(%rbp),%eax

%rbp-0x4에 위치한 메모리를 %eax에 값으로 대입한다.

주소	메모리
0x7ffffffdfa8	0xf7deb0b3
0x7ffffffdfa0	0x00000000
0x7ffffffdf9c	0x00000006
0x7ffffffdf98	0x00000003
0x7ffffffdf90	0xffffe090

18. mov %eax,%esi

%eax값을 %esi값에 대입한다.

주소	메모리
0x7ffffffdfa8	0xf7deb0b3
0x7ffffffdfa0	0x00000000
0x7ffffffdf9c	0x00000006
0x7ffffffdf98	0x00000003
0x7ffffffdf90	0xffffe090

```
(gdb) x $eax

0x6: Cannot access memory at address 0x6
(gdb) x $esi

0xfffffffffffffe098: Cannot access memory at address 0xffffffffffffe098
(gdb) si
0x0000555555555180 19 printf("result = %d\n", result);
(gdb) x $eax
0x6: Cannot access memory at address 0x6
(gdb) x $esi
0x6: Cannot access memory at address 0x6
```

19. lea 0xe7d(%rip),%rdi #055555556004

주소	메모리
0x7ffffffdfa8	0xf7deb0b3
0x7ffffffdfa0	0x00000000
0x7ffffffdf9c	0x00000006
0x7ffffffdf98	0x00000003
0x7ffffffdf90	0xffffe090

```
(gdb) disas
Dump of assembler code for function main:
   0x00000555555555515b <+0>:
                                endbr64
   0x000055555555555 <+4>:
                                push %rbp
   0X00005555555555100 <+5>:
                                       Mrsp.Wrbp
                                nov
   0x00008555555555103 <+8>:
                                       S0x10,%rsp
                                sub
   0x0000555555555107 <+12>1
                                movl $0x3,-8x8(%rbp)
   0x00008555555555516E <+19>:
                                       -0x8(%rbp),%eax
                                nov
   0x00005555555555171 <+22>1
                                MOV
                                       %eax,%edt
   @X8000055555555551/F <+24>:
                                callo 0%555555555549 <test func>
   0x00005555555555178 <+29>:
                                       %eax,-8x4(%rbp)
   0x000055555555551/h <+32>1
                                       -0x4(%rbp),%eax
                                NOV
   8x888855555555517E <+35>1
                                MOV
                                       %eax,%est
   0x00005555555555100 <+37>:
                                 lea
                                       0xe7d(%rtp),%rdt
                                                                # 0x55555556004
=> 0x0000555555555107 <+44>:
                                       S0x0, %eax
                                MOV
                                callq 0x555555555550 <printf@plt>
   0x0000555555555518c <+49>;
   0x0000555555555591 <+54>1
                                nov
                                        S0x0.Keax
   0X00005555:55:5190-<+59>1
                                leaveg
   0x00005555555555107 <+60>:
                                retq
End of assembles dumo
(qdb) x/c 0x55555556604
  555555555000=: 114 'r'
 (gdb)
  SSSSSSSSSSSSS: 101 'e'
 0x5555555550000: 115 's'
(dbp)
 #8-5555550007: 117 'u'
 MS555555550000: 108 'l'
 (dbp)
 #5555555556009: 116 't'
```

20. mov \$0x0,%eax

0x0을 %eax값에 대입한다.

주소	메모리	
0x7ffffffdfa8	0xf7deb0b3	
0x7ffffffdfa0	0x00000000	
0x7ffffffdf9c	0x00000006	
0x7ffffffdf98	0x00000003	
0x7ffffffdf90	0xffffe090	

```
21.
(gdb) ni
// callq printf 함수 수행
(gdb) si
// mov $0x0,%eax
(gdb) si
// leaveq 스택 해제 명령어
// leaveq 전, 후의 stack frame 확인
```

주소	메모리
0x7ffffffdfa8	0xf7deb0b3
0x7ffffffdfa0	0x00000000
0x7ffffffdf9c	0x00000006
0x7ffffffdf98	0x00000003
0x7ffffffdf90	0xffffe090

```
(adb) disas
Dump of assembler code for function main:
   endbr64
   0x00005555555555515f <+4>:
                                 push
                                        %rbp
   0x00005555555555160 <+5>:
                                        %rsp.%rbp
                                 MOV
   0x000055555555555163 <+8>:
                                 sub
                                        $0x10,%rsp
                                        $0x3,-0x8(%rbp)
   0x00005555555555167 <+12>:
                                 movl
   0x00000555555555516e <+19>:
                                        -0x8(%rbp),%eax
                                 MOV
   0x000005555555555171 <+22>:
                                        %eax.%edi
                                 MOV
                                        0x5555555555149 <test func>
   0x0000055555555555173 <+24>:
                                 calla
   0x00005555555555178 <+29>:
                                        %eax,-0x4(%rbp)
                                 mov
   0x0000555555555517b <+32>:
                                        -0x4(%rbp), %eax
                                 MOV
   0x000005555555555517e <+35>:
                                        %eax, %esi
                                 MOV
                                 lea
                                        0xe7d(%rip),%rdi
                                                                 # 0x55555556004
   0x000005555555555180 <+37>:
   0x00005555555555187 <+44>:
                                        S0x0.%eax
                                 MOV
                                 callq 0x55555555555050 <printf@plt>
=> 0x0000555555555518c <+49>:
   0x000005555555555191 <+54>:
                                        S0x0.%eax
                                 MOV
                                 leavea
   0x00005555555555196 <+59>:
   0x00005555555555197 <+60>:
                                 reta
End of assembler dump.
(qdb) ni
result = 6
20
                return 0:
(gdb) si
(adb) x/6wx $rsp
0x7ffffffffdf90: 0xffffe090
                                                 0x00000003
                                 0x00007fff
                                                                  0x00000006
0x7ffffffffdfa0: 0x00000000
                                 0x00000000
(adb) si
                        21
(gdb) x/6wx $rsp
0x7ffffffffdfa8: 0xf7deb0b3
                                 0x00007fff
                                                 0xf7ffc620
                                                                  0x00007fff
0x7fffffffdfb8: 0xffffe098
                                 0x00007fff
```