2 일차

은태영

기계어 분석

기계어 1 줄 해석 : push / rbp

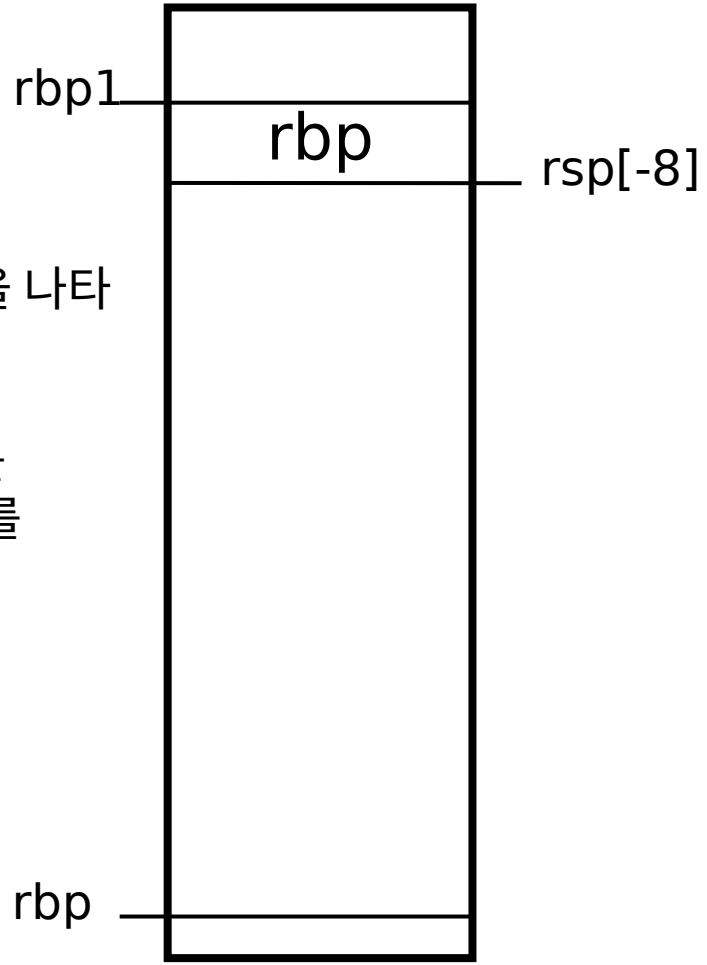
```
tewill@tewill-Z20NH-AS51B5U: ~/my_proj/Homework/sanghoonlee
Starting program: /home/tewill/my_proj/Homework/sanghoonlee/debug

Breakpoint 2, main () at func1.c:11
11      {
(gdb) disas
Dump of assembler code for function main:
=> 0x0000000000400535 <+0>:     push   %rbp
   0x0000000000400536 <+1>:     mov    %rsp,%rbp
   0x0000000000400539 <+4>:     sub    $0x10,%rsp
   0x000000000040053d <+8>:     movl   $0x3,-0x8(%rbp)
   0x0000000000400544 <+15>:    mov    -0x8(%rbp),%eax
   0x0000000000400547 <+18>:    mov    %eax,%edi
   0x0000000000400549 <+20>:    callq  0x400526 <myfunc>
   0x000000000040054e <+25>:    mov    %eax,-0x4(%rbp)
   0x0000000000400551 <+28>:    mov    -0x4(%rbp),%eax
   0x0000000000400554 <+31>:    mov    %eax,%esi
   0x0000000000400556 <+33>:    mov    $0x4005f4,%edi
   0x000000000040055b <+38>:    mov    $0x0,%eax
   0x0000000000400560 <+43>:    callq  0x400400 <printf@plt>
   0x0000000000400565 <+48>:    mov    $0x0,%eax
   0x000000000040056a <+53>:    leaveq
   0x000000000040056b <+54>:    retq
End of assembler dump.
(gdb)
```

명령어 : push
스택의 기준을 잡아 준다 .

여기서는 main 이 시작되는 것을 나타낸다 .

운영체제가 잡아주던 rbp 주소를 저장한 후 , 64 비트 기준 8 비트를 스택에 쌓아준다 .[주소값 - 8]



기계어 분석

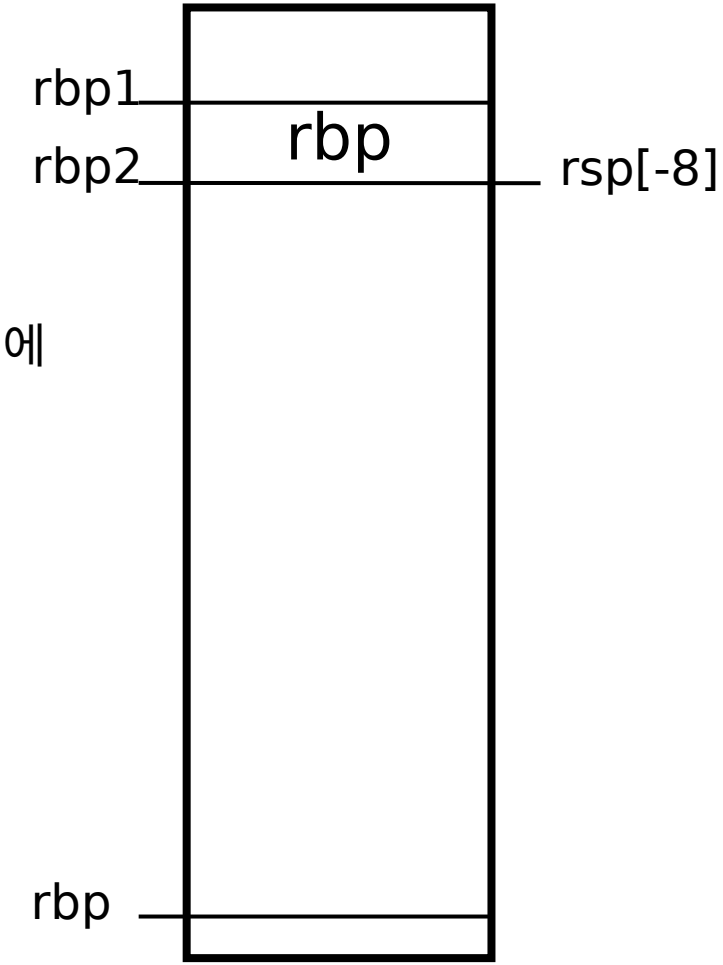
기계어 2 줄 해석 : mov / rsp, rbp

```
tewill@tewill-Z20NH-AS51B5U: ~/my_proj/Homework/sanghoonlee
Starting program: /home/tewill/my_proj/Homework/sanghoonlee/debug

Breakpoint 2, main () at func1.c:11
11      {
(gdb) disas
Dump of assembler code for function main:
=> 0x0000000000400535 <+0>:     push   %rbp
   0x0000000000400536 <+1>:     mov    %rsp,%rbp
   0x0000000000400539 <+4>:     sub    $0x10,%rsp
   0x000000000040053d <+8>:     movl   $0x3,-0x8(%rbp)
   0x0000000000400544 <+15>:    mov    -0x8(%rbp),%eax
   0x0000000000400547 <+18>:    mov    %eax,%edi
   0x0000000000400549 <+20>:    callq  0x400526 <myfunc>
   0x000000000040054e <+25>:    mov    %eax,-0x4(%rbp)
   0x0000000000400551 <+28>:    mov    -0x4(%rbp),%eax
   0x0000000000400554 <+31>:    mov    %eax,%esi
   0x0000000000400556 <+33>:    mov    $0x4005f4,%edi
   0x000000000040055b <+38>:    mov    $0x0,%eax
   0x0000000000400560 <+43>:    callq  0x400400 <printf@plt>
   0x0000000000400565 <+48>:    mov    $0x0,%eax
   0x000000000040056a <+53>:    leaveq
   0x000000000040056b <+54>:    retq
End of assembler dump.
(gdb)
```

명령어 : mov
값을 이동 혹은 복사한다 .

여기서는 main 의 스택 기준을 나타내기 위해 rsp 주소를 rbp 에 복사한다 .



기계어 분석

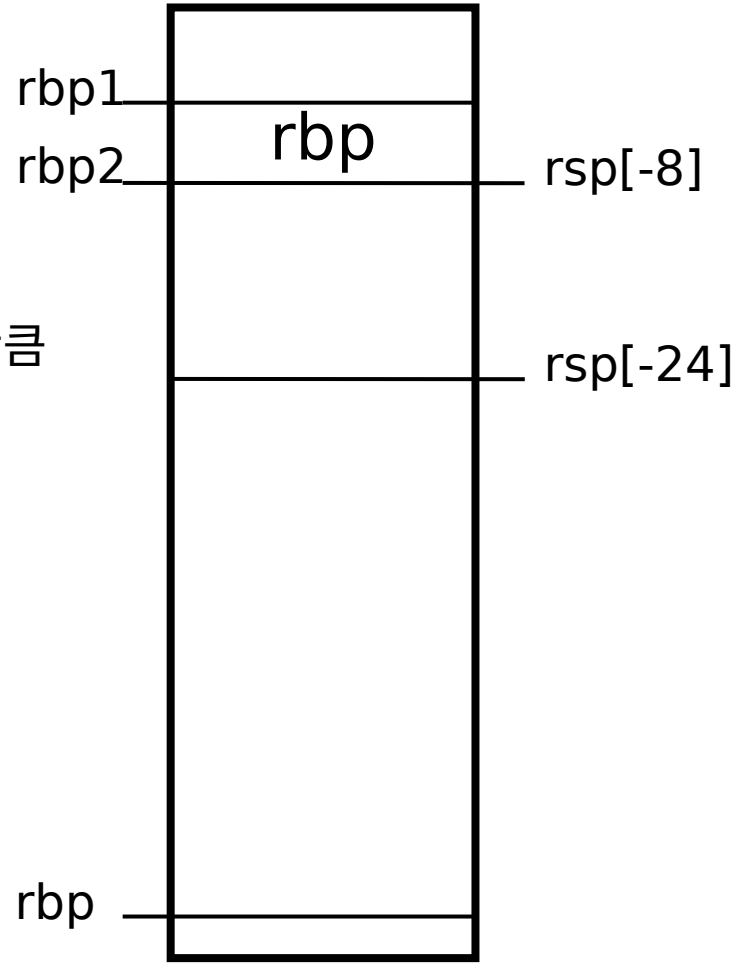
기계어 3 줄 해석 : sub / $0x10, rsp

```
tewill@tewill-Z20NH-AS51B5U: ~/my_proj/Homework/sanghoonlee
Starting program: /home/tewill/my_proj/Homework/sanghoonlee/debug

Breakpoint 2, main () at func1.c:11
11      {
(gdb) disas
Dump of assembler code for function main:
=> 0x0000000000400535 <+0>:     push   %rbp
   0x0000000000400536 <+1>:     mov    %rsp,%rbp
   0x0000000000400539 <+4>:     sub    $0x10,%rsp
   0x000000000040053d <+8>:     movl   $0x3,-0x8(%rbp)
   0x0000000000400544 <+15>:    mov    -0x8(%rbp),%eax
   0x0000000000400547 <+18>:    mov    %eax,%edi
   0x0000000000400549 <+20>:    callq  0x400526 <myfunc>
   0x000000000040054e <+25>:    mov    %eax,-0x4(%rbp)
   0x0000000000400551 <+28>:    mov    -0x4(%rbp),%eax
   0x0000000000400554 <+31>:    mov    %eax,%esi
   0x0000000000400556 <+33>:    mov    $0x4005f4,%edi
   0x000000000040055b <+38>:    mov    $0x0,%eax
   0x0000000000400560 <+43>:    callq  0x400400 <printf@plt>
   0x0000000000400565 <+48>:    mov    $0x0,%eax
   0x000000000040056a <+53>:    leaveq
   0x000000000040056b <+54>:    retq
End of assembler dump.
(gdb)
```

명령어 : sub
해당 함수 내 변수들의
메모리를 잡아준다 .

Rbp 를 기준으로 하여 16 비트만큼
Rsp 를 쌓아준다 .[rsp-24]



기계어 분석

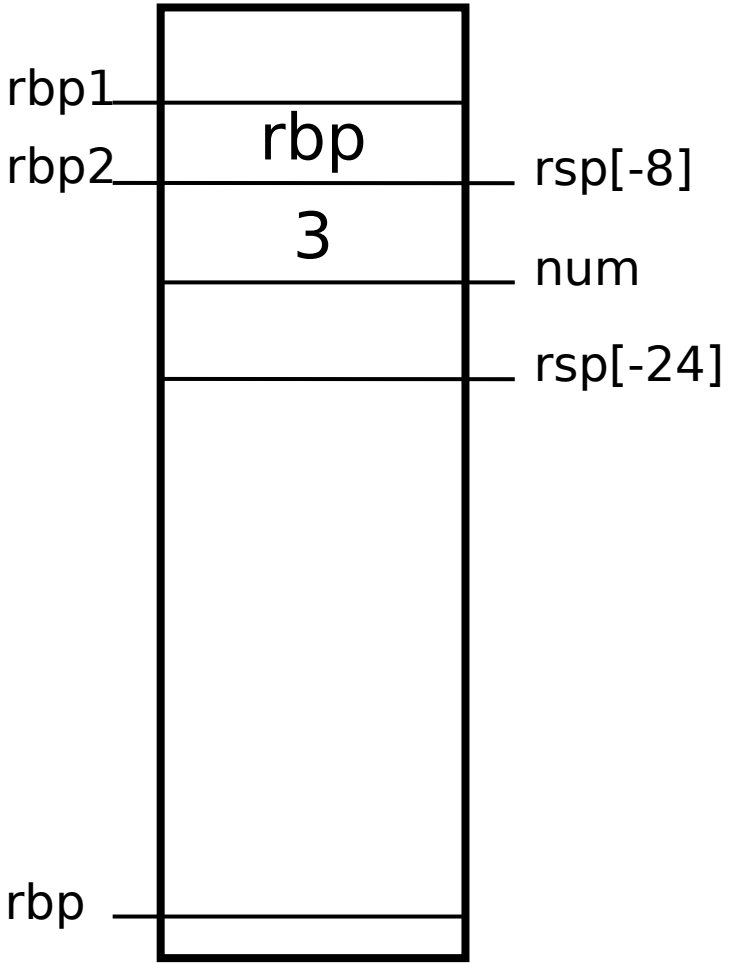
기계어 4 줄 해석 : movl / $0x3, -0x8(%rbp)

```
tewill@tewill-Z20NH-AS51B5U: ~/my_proj/Homework/sanghoonlee
Starting program: /home/tewill/my_proj/Homework/sanghoonlee/debug

Breakpoint 2, main () at func1.c:11
11      {
(gdb) disas
Dump of assembler code for function main:
=> 0x0000000000400535 <+0>:     push   %rbp
   0x0000000000400536 <+1>:     mov    %rsp,%rbp
   0x0000000000400539 <+4>:     sub    $0x10,%rsp
   0x000000000040053d <+8>:     movl   $0x3,-0x8(%rbp)
   0x0000000000400544 <+15>:    mov    -0x8(%rbp),%eax
   0x0000000000400547 <+18>:    mov    %eax,%edi
   0x0000000000400549 <+20>:    callq  0x400526 <myfunc>
   0x000000000040054e <+25>:    mov    %eax,-0x4(%rbp)
   0x0000000000400551 <+28>:    mov    -0x4(%rbp),%eax
   0x0000000000400554 <+31>:    mov    %eax,%esi
   0x0000000000400556 <+33>:    mov    $0x4005f4,%edi
   0x000000000040055b <+38>:    mov    $0x0,%eax
   0x0000000000400560 <+43>:    callq  0x400400 <printf@plt>
   0x0000000000400565 <+48>:    mov    $0x0,%eax
   0x000000000040056a <+53>:    leaveq
   0x000000000040056b <+54>:    retq
End of assembler dump.
(gdb)
```

명령어 : movl
mov 와 동일하다 .

rbp 기준 -8 만큼 위치한 공간에
3 의 값을 저장한다 .



기계어 분석

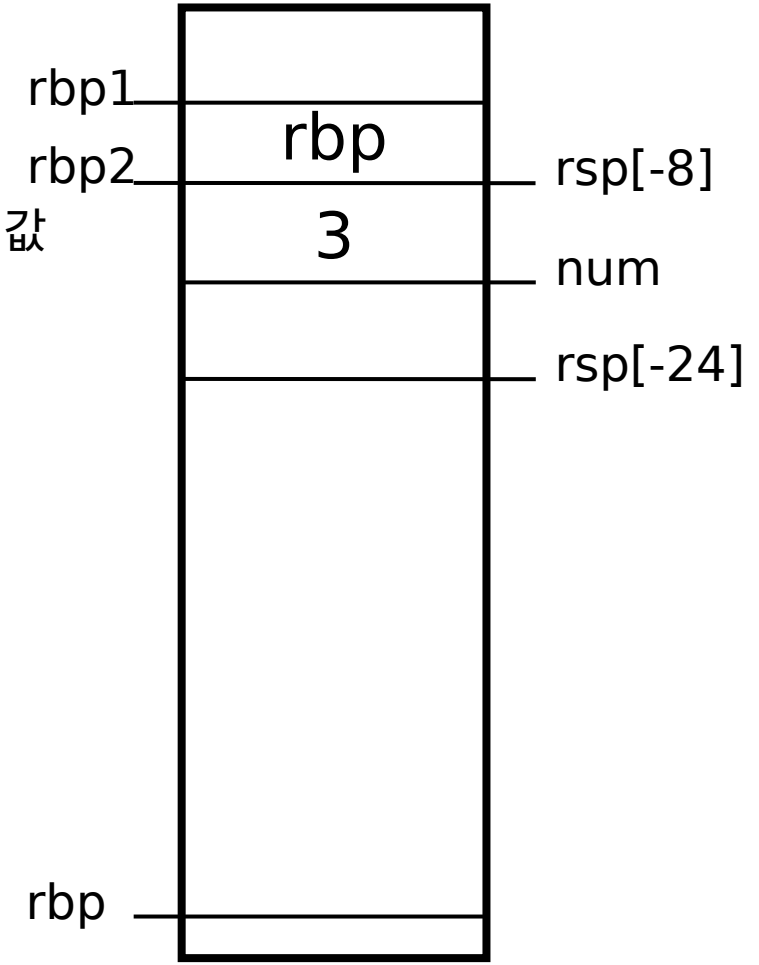
기계어 5 줄 해석 : mov / -0x8(%rbp), %eax

```
tewill@tewill-Z20NH-AS51B5U: ~/my_proj/Homework/sanghoonlee
Starting program: /home/tewill/my_proj/Homework/sanghoonlee/debug

Breakpoint 2, main () at func1.c:11
11      {
(gdb) disas
Dump of assembler code for function main:
=> 0x0000000000400535 <+0>:     push   %rbp
   0x0000000000400536 <+1>:     mov    %rsp,%rbp
   0x0000000000400539 <+4>:     sub    $0x10,%rsp
   0x000000000040053d <+8>:     movl   $0x3,-0x8(%rbp)
   0x0000000000400544 <+15>:    mov    -0x8(%rbp),%eax
   0x0000000000400547 <+18>:    mov    %eax,%edi
   0x0000000000400549 <+20>:    callq  0x400526 <myfunc>
   0x000000000040054e <+25>:    mov    %eax,-0x4(%rbp)
   0x0000000000400551 <+28>:    mov    -0x4(%rbp),%eax
   0x0000000000400554 <+31>:    mov    %eax,%esi
   0x0000000000400556 <+33>:    mov    $0x4005f4,%edi
   0x000000000040055b <+38>:    mov    $0x0,%eax
   0x0000000000400560 <+43>:    callq  0x400400 <printf@plt>
   0x0000000000400565 <+48>:    mov    $0x0,%eax
   0x000000000040056a <+53>:    leaveq
   0x000000000040056b <+54>:    retq
End of assembler dump.
(gdb)
```

명령어 : mov

범용 레지스터 eax 에 num 의 값 3 을 저장한다 .



기계어 분석

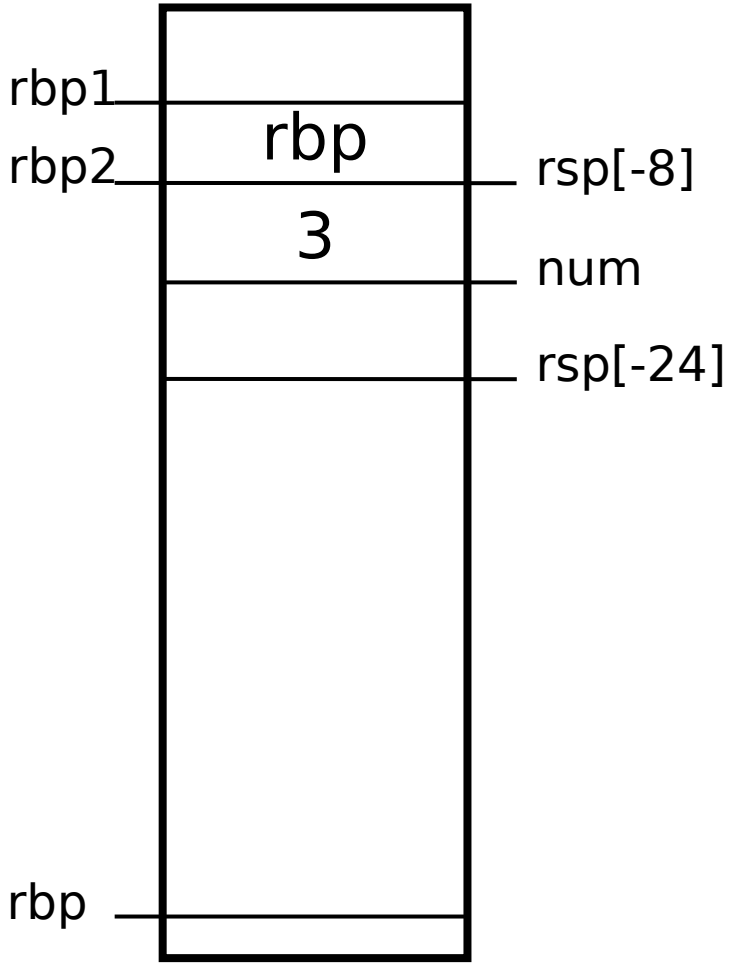
기계어 6 줄 해석 : mov / %eax, % edi

```
tewill@tewill-Z20NH-AS51B5U: ~/my_proj/Homework/sanghoonlee
Starting program: /home/tewill/my_proj/Homework/sanghoonlee/debug

Breakpoint 2, main () at func1.c:11
11      {
(gdb) disas
Dump of assembler code for function main:
=> 0x0000000000400535 <+0>:     push   %rbp
   0x0000000000400536 <+1>:     mov    %rsp,%rbp
   0x0000000000400539 <+4>:     sub    $0x10,%rsp
   0x000000000040053d <+8>:     movl   $0x3,-0x8(%rbp)
   0x0000000000400544 <+15>:    mov    -0x8(%rbp),%eax
   0x0000000000400547 <+18>:    mov    %eax,%edi
   0x0000000000400549 <+20>:    callq  0x400526 <myfunc>
   0x000000000040054e <+25>:    mov    %eax,-0x4(%rbp)
   0x0000000000400551 <+28>:    mov    -0x4(%rbp),%eax
   0x0000000000400554 <+31>:    mov    %eax,%esi
   0x0000000000400556 <+33>:    mov    $0x4005f4,%edi
   0x000000000040055b <+38>:    mov    $0x0,%eax
   0x0000000000400560 <+43>:    callq  0x400400 <printf@plt>
   0x0000000000400565 <+48>:    mov    $0x0,%eax
   0x000000000040056a <+53>:    leaveq
   0x000000000040056b <+54>:    retq
End of assembler dump.
(gdb)
```

명령어 : mov

범용 레지스터 edi 에 eax 의 값 3 을 저장한다 .



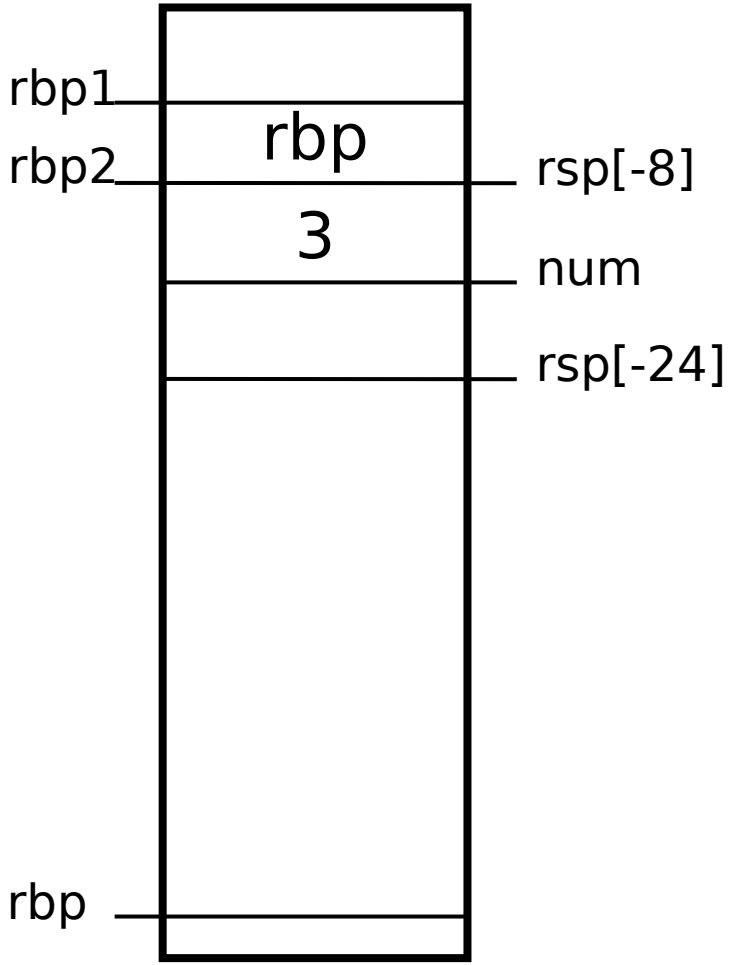
기계어 분석

기계어 7 줄 해석 : callq / 0x400526, <my-func>

```
tewill@tewill-Z20NH-AS51B5U: ~/my_proj/Homework/sanghoonlee
Starting program: /home/tewill/my_proj/Homework/sanghoonlee/debug

Breakpoint 2, main () at func1.c:11
11      {
(gdb) disas
Dump of assembler code for function main:
=> 0x0000000000400535 <+0>:     push   %rbp
   0x0000000000400536 <+1>:     mov    %rsp,%rbp
   0x0000000000400539 <+4>:     sub    $0x10,%rsp
   0x000000000040053d <+8>:     movl   $0x3,-0x8(%rbp)
   0x0000000000400544 <+15>:    mov    -0x8(%rbp),%eax
   0x0000000000400547 <+18>:    mov    %eax,%edi
   0x0000000000400549 <+20>:    callq  0x400526 <myfunc>
   0x000000000040054e <+25>:    mov    %eax,-0x4(%rbp)
   0x0000000000400551 <+28>:    mov    -0x4(%rbp),%eax
   0x0000000000400554 <+31>:    mov    %eax,%esi
   0x0000000000400556 <+33>:    mov    $0x4005f4,%edi
   0x000000000040055b <+38>:    mov    $0x0,%eax
   0x0000000000400560 <+43>:    callq  0x400400 <printf@plt>
   0x0000000000400565 <+48>:    mov    $0x0,%eax
   0x000000000040056a <+53>:    leaveq
   0x000000000040056b <+54>:    retq
End of assembler dump.
(gdb)
```

명령어 : callq



포인터 크기

포인터의 란 ?

- 메모리의 주소값을 나타내는 변수이며 , 포인터의 크기는 운영체제에 따라 정해진 값을 갖고 있다 .

포인터의 크기가 정해진 이유 .

- ALU 의 연산이 범용 레지스터에 종속적으로 되어 있으며 , 이 범용 레지스터들은 운영체제에 따라 비트가 구성되어 있기 때문이다 .
- 주소 값을 저장하는데 있어서 , 최소값에서 최대값까지 모두 저장이 가능해야 하므로 최대값을 기준으로 포인터 크기가 정해진다 .
- 운영체제가 64 비트의 경우 8 바이트 , 32 비트의 경우 4 바이트가 포인터의 크기이다 .

2 진수 , 16 진수 변환 정리

2 진수 란 ?

- 컴퓨터에서 사용하는 방식이며 , 0 과 1 로 숫자를 표시한다 .

16 진수 란 ?

- 컴퓨터에서 사용하는 방식이며 , 10 에서 15 까지의 숫자를 알파벳 a 에서 f 까지로 표시한다 .
- 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f 로 숫자를 표시한다 .

10 진수의 진수 변환 정리

- 해당 진수의 제곱을 통하여 계산할 수 있다 .
- 구하고자 하는 값에서 진수의 제곱한 값을 지워 나가며 , 자릿수를 채워 나가는 방식이다 .
- 예를들어 65 를 2 진수로 구하고자 하는 경우 , 2^6 값인 64 를 65 에서 뺀 후 , 남은 1 의 값인 2^0 에 들어가게 된다 . 이 계산을 통해 65 는 2 진수 값으로 100 0001 이 된다 .

2 진수 , 16 진수 변환 정리

2 진수와 16 진수

- 컴퓨터가 사용하는 2 진수의 경우 , 0 과 1 만 존재하여 숫자의 값에 따라 가독성이 떨어지기 때문에 , 이 부분을 개선하고자 16 진수로 변환하여 표시한다 .

2 진수와 16 진수의 변환 정리

- 4 자리의 2 진수는 총 16 의 값을 표현할 수 있다 .
- 이를 통하여 , 진수의 변환을 할 때 , 2 진수의 4 자리 씩 계산을 하면 편하게 변환을 할 수 있다 .
- 예를들어 1100 1001 이라는 2 진수를 변환 할 경우 16 진수의 2 번째 자리에는 1100 의 값인 C 가 , 1 번째 자리인 1001 의 값인 9 가 들어가 C9 의 값이 들어간다 .