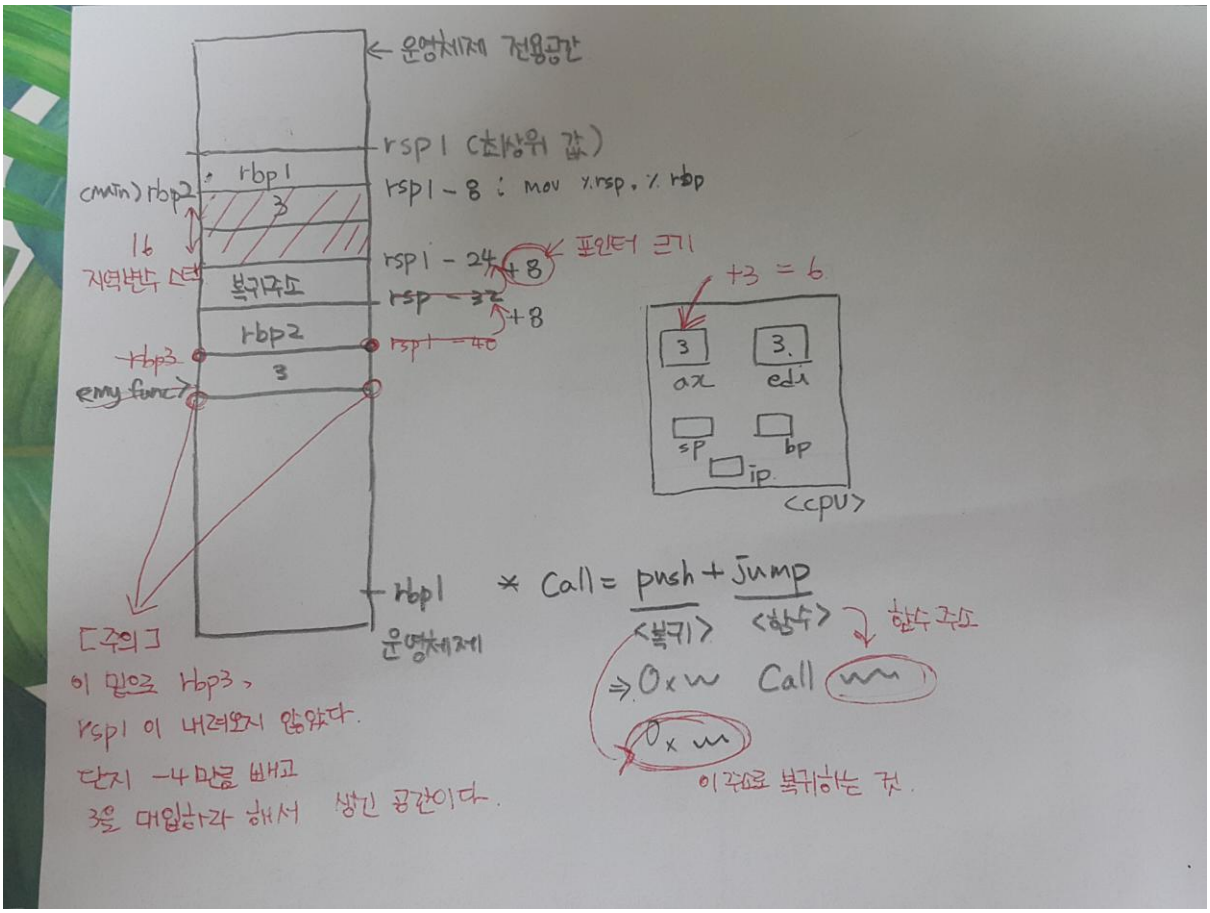
*기계어 분석

rsp1(최상위 값)이 지정되는 이유는 스택의 특성 때문이다. 스택은 값을 주면 밑으로 정보를 쌓는 특성이 있는데 이는 다른 것과 다른 구조로, 운영체제 전용구간이 지정되지 않을 경우, 메모리가 깨질 수 있기 때문이다.

rbp2가 main이 되는데, 이는 bp가 스택의 기준점이라 rbp의 값은 변동되지 않는다. rsp 는 값이 변동 된다.

Si : 기계어 단위로 한 줄만 실행할 것을 명령한다.

앞의 주소는 빼고 뒤의 명령어만 기입하였습니다.



push %rbp : 현재 스택에 쌓는 것을 말한다. 스택에 값을 넣는 것을 push라 하며, 위 그림에서 메모리에 rbp2 를 main으로 잡는 역할을 한다.

mov %rsp,%rbp : mov는 이동으로 값을 넣는 역할을 한다. 즉, rsp 에 rbp 값을 넣어주게 됩니다. 위 그림에서처럼 8만큼의 값이 빠지게 됩니다(8은 포인터 베이스기 때문이다.)

이 동작 후에 p/x $ rsp 로 8값이 빠졌는지 확인이 가능하다. 또 x $ rsp 로 rbp 값과 같은지 확인이 가능하다.

sub $0x10,% rsp : sub는 뺄셈으로 rsp에서 $0x10의 값을 빼주면 되는데 $0x10는 16진법으로 10진법으로 바꾸면 16이다. 그 만큼의 스택을 빼준다. 그럼 그림에서처럼 rsp1 -24로 잡히게 된다. 여기서 16은 지역변수 스택이 된다.

movl $0x3,-0x8(%rbp) : movl은 mov와 비슷한 함수로 길이만 차이가 진다. 즉 -0x8(%rbp)에 $0x3를 넣어주면 된다. 여기서 -0x8(%rbp)는 ( )안 값을 0x8만큼 빼라는 뜻으로 rbp2 값에서 8만큼 뺀 스택을 잡아준다. 여기에 $0x3은 3이므로, 그림에서처럼 잡아준 자리에 3을 넣어주면 된다.

mov -0x8(%rbp),%eax : -0x8(%rbp)의 값을 eax에 넣어주면 된다. 위의 실행된 내용에서 -0x8(%rbp)는 3이었으므로 cpu에 위치하는 eax 값은 3이 된다.

mov %eax,%edi : eax 값을 edi에 넣으라는 것으로 cpu에 있는 edi 값 또한 3이 된다.

callq 0x400526 <myfunc> : call = push + jump 로 된 명령어로 push는 다음 연산을 위해 화살표 밑 주소로 복귀하라는 것이 된다. Jump는 함수 주소로 가라는 것이다. 보통 명령어 다음에 주소가 있고 여기서는 0x400526 <myfunc>의 함수로 가라는 것이다. 이 함수를 완료하면 push로 복귀가 된다.

이 작업까지 하고 disas를 하면 함수 기계어가 보인다.

push %rbp

mov %rsp,%rbp : 이 두 줄은 시작부분에 항상 있는 것으로 Stack Frame을 생성한다. rsp 와 rbp의 값이 같아지면서 새로운 기준점이 생성된다.

mov %edi,-0x4(%rbp) : -0x4(%rbp)는( )안 값을 -0x4만큼 빼라는 것으로 4를 빼준 스택을 잡는다. 여기에 edi 값을 넣어주면 된다. 그림에서처럼 3이 잡힌다.

mov -0x4(%rbp),%eax : 위 작동에서 잡힌 3을 eax에 대입해주면 된다.

add $0x3,%eax : add라는 명령어처럼 값을 더해주면 된다. 0x3는 10진법으로 3이고 eax의 값이 3이었으므로 6이 된다.

pop %rbp : pop은 빼내는 것으로 현재 sp에서 값을 빼서 뒤로 넘겨주는 것을 말한다. 따라서, rsp1 – 40이 rsp1 – 32가 됩니다.

retq : 복귀하라는 명령어 입니다.

pop %rbp

retq : 이 두 명령어 또한 항상 끝에 존재하며 stack frame 해제를 뜻합니다.

이렇게 아까 callq 명령어처럼 복귀주소로 다시 복귀하게 되고 rsp1 -24위치로 이동되는데 이는 포인트 크기만큼 이동이 됩니다. 레지스터가 cpu에 있어서 연산을 위해 값의 이동이 이루어진 것입니다. My func가 끝났기 때문에 그 위치 값 또한 지워집니다. 이후

```
  0x0000000000400549 <+20>:    callq  0x400526 <myfunc>
=> 0x000000000040054e <+25>:    mov    %eax,-0x4(%rbp)
```

식의 복귀가 이루어집니다.

* 포인터 크기

컴퓨터에서 변수(Variable)는 메모리에 존재하는 정보를 저장할 수 있는 공간이다. 이때 **정보가 주소가 되면** 포인터라고 한다(즉, C언어에서 포인터는 주소가 된다.) 따라서, 몇 비트의 시스템을 쓰느냐에 따라 크기가 달라진다(예를 들면 8 비트 시스템의 경우 1 byte, 16 비트는 2 byte, 32 비트는 4 byte, **64 비트는 8 byte** 가 된다.)

이는 컴퓨터의 산술 연산이 ALU에 의존적이기 때문이다. ALU의 연산은 범용 레지스터에 종속적

이고 컴퓨터가 64비트라는 의미는 이들이 64비트로 구성되었기 때문이다. 포인터는 메모리에 주소를 저장하는 공간이므로 64비트로 표현할 수 있는 최대값을 저장할 수 있어야 한다. 만약 포인터의 크기가 작다면, 그 주소를 다 저장(표현)할 수 없기 때문이다. 따라서, 최대치인 64비트(8 byte)가 포인터의 크기가 된 것이다.

이는 리눅스에서 Terminal 창으로 확인할 수 있다.

vi pointer_size.c

```
#include <stdio.h>

int main(void) {
        printf("sizeof(int *) = %lu₩n", sizeof(int *));
        printf("sizeof(double *) = %lu₩n", sizeof(double *));
        printf("sizeof(float *) = %lu₩n", sizeof(float *));
        return 0;
}
```

결과가 전부 8 이 나오는 것을 볼 수 있을 것이다. 이는 스택의 동작과정이 포인터 베이스이기 때문이다. 또한, 모든 컴퓨터의 동작과정이 이 포인터 베이스로 동작된다.

* 2진수, 16진수 변환 정리

2진수 : 0, 1의 2종류 숫자로 나타내는 수. 컴퓨터가 사용하는 방식이다.
사람은 10진수를 기본으로 하기 때문에, 컴퓨터에 인식시키기 위해서는 2진법으로의 변환이 필요하다. 방법은 10진수를 2로 0이 될 때까지 계속 나눈 법과 더하는 방법이 있다.
10진법 129를 2진법으로 바꾼다고 가정해본다.
129를 10진수에서 분해 한다면, 1 x 10^2 + 2 x 10^1 + 9 x 10^0 으로 10이 계속 곱해지며 달라지는 부분은 승수부분이다. 2진법도 비슷하게 생각하면 된다.
2진수의 자릿수를 쭉 적어본다,

	7	6	5	4	3	2	1	0
128	64	32	16	8	4	2	1	
=	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

129 = 128 + 1 이므로 위의 7번째와 0번째에 1 값을 주고 나머지는 0을 주면 된다.
즉, 129를 2진법으로 나타내면 10000001(2)가 된다.
이를 10진법으로 다시 바꾼다면 1이 있는 자릿수에 -1를 하고 승수로 해서 더해주면 10진법으로 나타낼 수 있다.

16진법 : 0~f의 16개로 숫자를 나타내는 방법이다. 컴퓨터가 주로 사용하는 방식이며, 컴퓨터는 2진법이 기본이나 수가 커질 겨우, 2진법도 커지게 되기 때문에 가독성을 좋게 하기 위해 인간이 채택한 방식이다. 즉, 컴퓨터를 배운 인간과 기계의 혼용어라고 생각하면 된다.

0~9까지는 10진법과 같은 방식으로 표기하나 10진법에서 자리 올림이 된 10부터 15까지는 한 글자인 영문알파벳으로 치환된다,

10 = a 11 = b 12 = c 13 = d 14 = e 15 = f

로 표기한다. 이런 식으로 16진법은 1자리에 16개가 오게 된다. 컴퓨터가 사용하는 진법이기에 10진법과의 변환에서 2진법을 사용하면 더 쉽게 변환할 수 있다. 앞서 말한, 16진법이 1자리에 16개가 오는 것을 2진법과 연결해서 생각하면 쉬워진다. 2진법은 1자리에 2개가 있을 수 있어, 4자릿수가 되면 16개를 표현할 수 있다. 따라서, 16진법 변환에서 2진법을 4자릿수로 끊어서 계산하면 쉽게 변환할 수 있다.

예를 들어, 10100001(2)을 16진법으로 변환한다고 가정해본다.

8 4 2 1 8 4 2 1 1이 입력된 곳의 수를 더 해준다. > 8+2 = 10 = a 1 = 1

1 0 1 0 0 0 0 1 0xa1 로 표시할 수 있다.

이를 이용해서 10진법을 16진법으로 변환시키면 된다.

예를 들어, 10진법 54를 16진법으로 변환한다고 가정해본다.

54는 110110(2)로 변환된다. 이를 4자릿수로 나눈다고 가정하면

8 4 2 1 8 4 2 1 1이 입력된 곳의 수를 더 해준다. > 2+1 = 3 4+2 = 6

0 0 1 1 0 1 1 0 0x36 으로 표시할 수 있다.