c언어 - HW4

임베디드스쿨1기

Lv1과정

2021. 04. 09

이충재

push % X : rsp레지스터에 X레지스터의 주소값을 저장한다.
그리고 자신의 주소를 8바이트만큼 내린다.

Pop %X: rsp레지스터에 있는 내용을 X레지스터의 주소로 한다.
그리고 자신의 주소를 8바이트만큼 올린다.

mov %X, %Y : Y레지스터의 주소를 X레지스터의 주소로 한다.
mov $X, %Y : Y레지스터 주소를 X값으로한다.

mov +0xa(%X), %Y: X레지스터로부터 a바이트 큰 주소에 있는 값을 %Y레지스터의 주소로 한다.
mov %X, +0xa(%Y): X레지스터 주소를 Y레지스터 보다 a바이트 큰 주소에 있는 값으로 한다.

lea %X, %Y: X레지스터의 주소 값을 Y레지스터의 주소 값으로 옮긴다.

Callq 주소 <함수이름>: push + jump, rsp에 복귀주소 저장하고 함수주소로 이동한다.

Leaveq: 1. mov %rbp, %rsp
         2. Pop %rbp ==> 1동작과 2동작이 동시에 일어난다.

rsp의 주소를 rbp의 주소와 동일하게 만들고 rsp의 내용을 rbp의 주소로 한다.

retq: pop rip ==> rsp의 내용을 rip의 주소로 내보낸다.
(rip는 다음에 실행할 코드의 위치를 주소로 가지는 레지스터)

연산 명령어

Add a, b : a 와 b를 더한값을 b에 넣는다.

Sub a, b: a에서 b를 뺀값을 a에 넣는다.

Sar $0xa %X: a비트만큼 x레지스터의 주소를 오른쪽 쉬프트연산

1. 쉬프트연산 함수

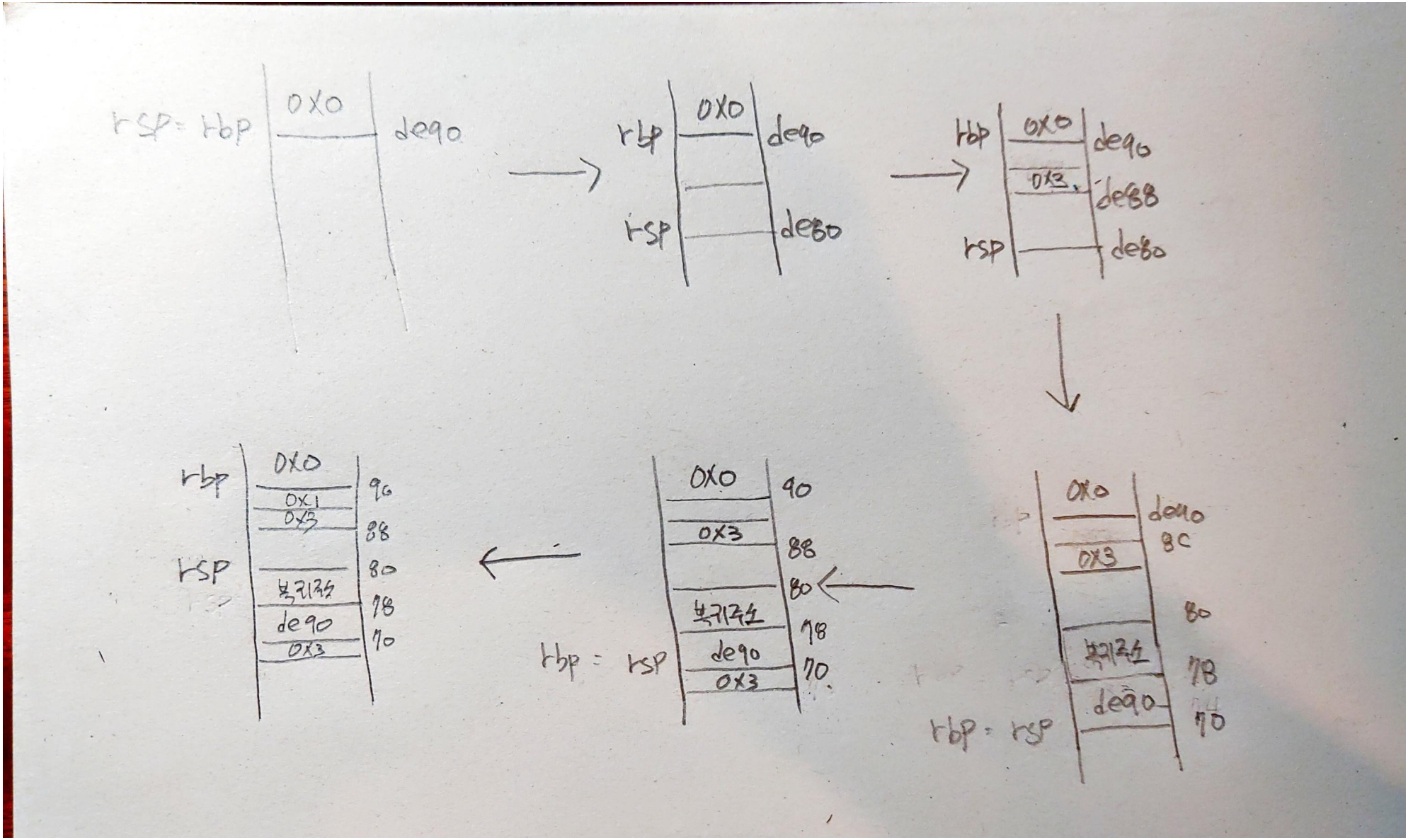
```
=> 0x000055555555515b <+0>:     endbr64
   0x000055555555515f <+4>:     push   %rbp
   0x0000555555555160 <+5>:     mov    %rsp,%rbp
   0x0000555555555163 <+8>:     sub    $0x10,%rsp
   0x0000555555555167 <+12>:    movl   $0x3,-0x8(%rbp)
   0x000055555555516e <+19>:    mov    -0x8(%rbp),%eax
   0x0000555555555171 <+22>:    mov    %eax,%edi
   0x0000555555555173 <+24>:    callq  0x555555555149 <my_func>
   0x0000555555555178 <+29>:    mov    %eax,-0x4(%rbp)
   0x000055555555517b <+32>:    mov    -0x4(%rbp),%eax
   0x000055555555517e <+35>:    mov    %eax,%esi
   0x0000555555555180 <+37>:    lea    0xe7d(%rip),%rdi        # 0x555555556004
   0x0000555555555187 <+44>:    mov    $0x0,%eax
   0x000055555555518c <+49>:    callq  0x555555555050 <printf@plt>
   0x0000555555555191 <+54>:    mov    $0x0,%eax
   0x0000555555555196 <+59>:    leaveq
   0x0000555555555197 <+60>:    retq
```

< Main 함수>

```
   0x0000555555555149 <+0>:     endbr64
=> 0x000055555555514d <+4>:     push   %rbp
   0x000055555555514e <+5>:     mov    %rsp,%rbp
   0x0000555555555151 <+8>:     mov    %edi,-0x4(%rbp)
   0x0000555555555154 <+11>:    mov    -0x4(%rbp),%eax
   0x0000555555555157 <+14>:    sar    %eax
   0x0000555555555159 <+16>:    pop    %rbp
   0x000055555555515a <+17>:    retq
```

<my_func 함수 >

쉬프트연산 동작 과정


rsp = rbp
0x0
de90
rbp
0x0
de90
rsp
de80
rbp
0x0
de90
0x3
de88
rsp
de80
rbp
0x0
de90
0x3
8C
80
복귀주소
78
de90
70
rbp = rsp
rbp
0x0
90
0x3
88
80
복귀주소
78
de90
70
0x3
rbp = rsp
rbp
0x0
90
0x1
0x3
88
rsp
80
복귀주소
78
de90
70
0x3

2. if문 분석

```c
int main(void)
{
        int num1 = 3;
        int num2 = 5;
        int num3;


        if(num1 > num2)
        {
                num3 = 0;
        }
        else
        {
                num3 = 1;
        }

        return 0;
}
```

c언어 코드

```
=> 0x0000555555555129 <+0>:     endbr64
   0x000055555555512d <+4>:     push   %rbp
   0x000055555555512e <+5>:     mov    %rsp,%rbp
   0x0000555555555131 <+8>:     movl   $0x3,-0xc(%rbp)
   0x0000555555555138 <+15>:    movl   $0x5,-0x8(%rbp)
   0x000055555555513f <+22>:    mov    -0xc(%rbp),%eax
   0x0000555555555142 <+25>:    cmp    -0x8(%rbp),%eax
   0x0000555555555145 <+28>:    jle    0x555555555150 <main+39>
   0x0000555555555147 <+30>:    movl   $0x0,-0x4(%rbp)
   0x000055555555514e <+37>:    jmp    0x555555555157 <main+46>
   0x0000555555555150 <+39>:    movl   $0x1,-0x4(%rbp)
   0x0000555555555157 <+46>:    mov    $0x0,%eax
   0x000055555555515c <+51>:    pop    %rbp
   0x000055555555515d <+52>:    retq
```

기계어 코드

<+8> movl 0x3, -0xc(%rbp) : rbp 12바이트 아래에 0x3을 넣는다.

<+15> movl 0x5, -0x8(%rbp): rbp 8바이트 아래에 0x5를 넣는다.

<+22> mov -0xc(%rbp), %eax: rbp 12바이트 아래에 있는 값(0x3)을 eax주소에 저장.

<+25> cmp -0x8(%rbp), %eax: rbp 8바이트 아래에 있는 값(0x5)를 eax주소(0x3)과 비교

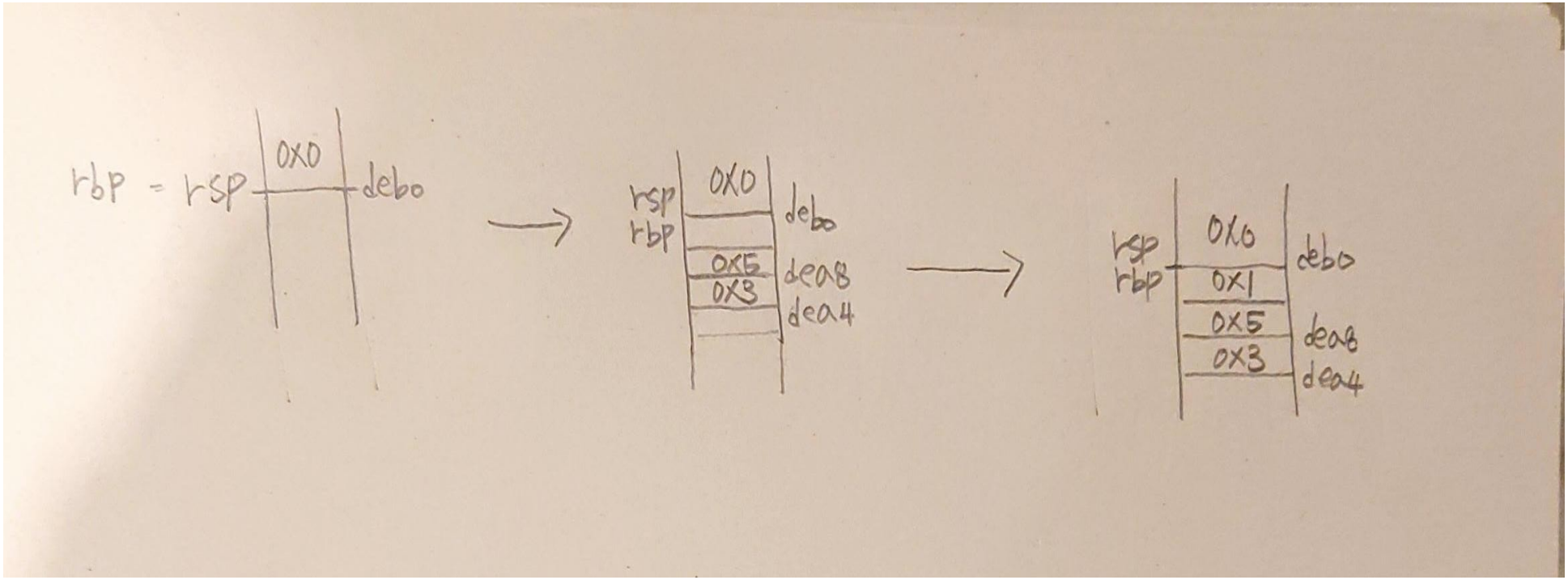
<+28> jle 0x55555555150 <main +39> : 비교한 값이 작거나 같으면 0x~~150주소로 이동

<+39> movl $0x1, -0x4(rbp): rbp 4바이트 아래에 0x1을 넣는다.

<+46> mov $0x0, %eax: eax의 주소를 0x0으로 한다.

<+51> pop %rbp: rsp에 있는값을 rbp주소로 이동.
rsp에 0x0이 저장되어 있었기에 rbp주소는 0x0이 된다.

if문 동작과정


rbp = rsp
0x0
debo
rsp
rbp
0x0
debo
0x5
0x3
dea8
dea4
rsp
rbp
0x0
debo
0x1
0x5
0x3
dea8
dea4

```c
#include <stdio.h>

int main(void)
{
        int i, num1 = 0;

        for(i = 0; i < 5; i++)
        {
                num1 += 1;
        }

        return 0;
}
```

```
=> 0x0000555555555129 <+0>:     endbr64
   0x000055555555512d <+4>:     push   %rbp
   0x000055555555512e <+5>:     mov    %rsp,%rbp
   0x0000555555555131 <+8>:     movl   $0x0,-0x4(%rbp)
   0x0000555555555138 <+15>:    movl   $0x0,-0x8(%rbp)
   0x000055555555513f <+22>:    jmp    0x555555555149 <main+32>
   0x0000555555555141 <+24>:    addl   $0x1,-0x4(%rbp)
   0x0000555555555145 <+28>:    addl   $0x1,-0x8(%rbp)
   0x0000555555555149 <+32>:    cmpl   $0x4,-0x8(%rbp)
   0x000055555555514d <+36>:    jle    0x555555555141 <main+24>
   0x000055555555514f <+38>:    mov    $0x0,%eax
   0x0000555555555154 <+43>:    pop    %rbp
   0x0000555555555155 <+44>:    retq
```

<+8> movl $0x0, -0x4(%rbp): rbp 4바이트 아래에 0x0을 넣는다.

<+15> $0x0, -0x8(%rbp): rbp 8바이트 아래에 0x0을 넣는다.

<+22> jmp 0x~~149 <main + 32> : 0x~149주소로 이동한다.

<+28> addl $0x1, -0x8(%rbp) : rbp 8바이트 아래에 0x1을 넣는다.

<+32> cmpl $0x4, -0x8(%rbp) : 0x4와 rbp 8바이트 아래에 있는 값(0x1)을 비교한다.

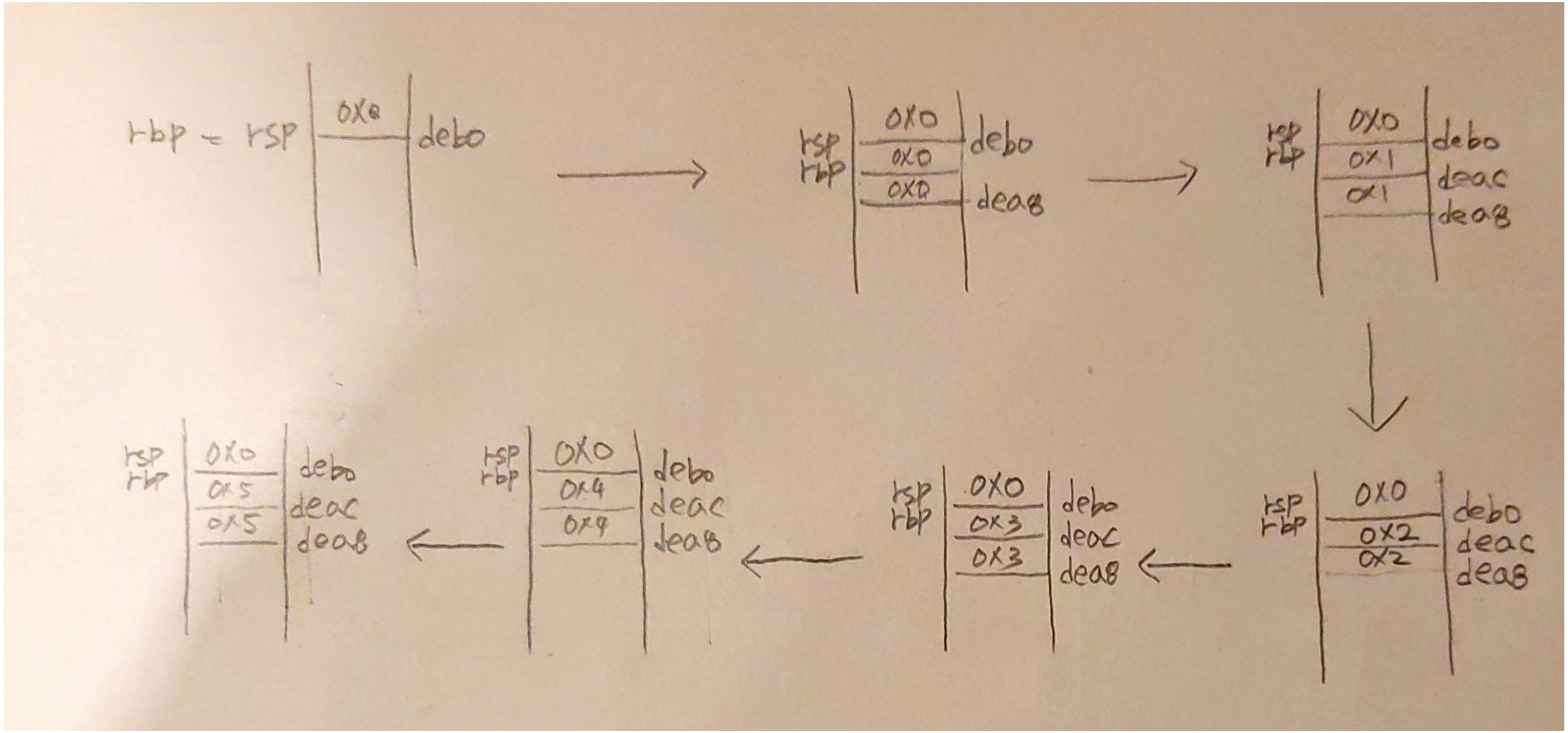
<+36> jle 0x~141 <main+24> : 비교한값이 작거나 같으면 ~141주소로 이동한다.
0x1은 0x4보다 작기때문에 적힌주소로 이동한다.

<+24> addl $0x1, -0x4(%rbp): rbp 4바이트 아래에 있는값(0x0)에 0x1을 더한다.

<+28> addl $0x1, -0x8(%rbp): rbp 8바이트 아래에 있는값(0x0)에 0x1을 더한다.

<+32> <+36>: 0x4와 rbp -8바이트에 있는 값과 비교하여 작거나 같으면 적힌주소로 이동한다.
rbp -8 이 5가 될때 까지 24 ~ 36과정을 계속 반복한다.

for문 동작과정


rbp = rsp
0x0
debo
rsp
rbp
0x0
debo
0x0
deac
0x0
deab
rsp
rbp
0x0
debo
0x1
deac
0x1
deab
rsp
rbp
0x0
debo
0x2
deac
0x2
deab
rsp
rbp
0x0
debo
0x3
deac
0x3
deab
rsp
rbp
0x0
debo
0x4
deac
0x4
deab
rsp
rbp
0x0
debo
0x5
deac
0x5
deab

```c
#include <stdio.h>

int main(void)
{
        int num1 = 1;
        int num2 = 5;

        while(num1 < num2)
        {
                num1++;
        }

        return 0;
}
```

```
=> 0x0000555555555129 <+0>:     endbr64
   0x000055555555512d <+4>:     push   %rbp
   0x000055555555512e <+5>:     mov    %rsp,%rbp
   0x0000555555555131 <+8>:     movl   $0x1,-0x8(%rbp)
   0x0000555555555138 <+15>:    movl   $0x5,-0x4(%rbp)
   0x000055555555513f <+22>:    jmp    0x555555555145 <main+28>
   0x0000555555555141 <+24>:    addl   $0x1,-0x8(%rbp)
   0x0000555555555145 <+28>:    mov    -0x8(%rbp),%eax
   0x0000555555555148 <+31>:    cmp    -0x4(%rbp),%eax
   0x000055555555514b <+34>:    jl     0x555555555141 <main+24>
   0x000055555555514d <+36>:    mov    $0x0,%eax
   0x0000555555555152 <+41>:    pop    %rbp
   0x0000555555555153 <+42>:    retq
End of assembler dump.
```

<+8> movl $0x1, -0x8(%rbp): 0x1을 rbp -8바이트에 넣는다.

<+15> movl $0x5, -0x4(%rbp): 0x5를 rbp -4바이트에 넣는다.

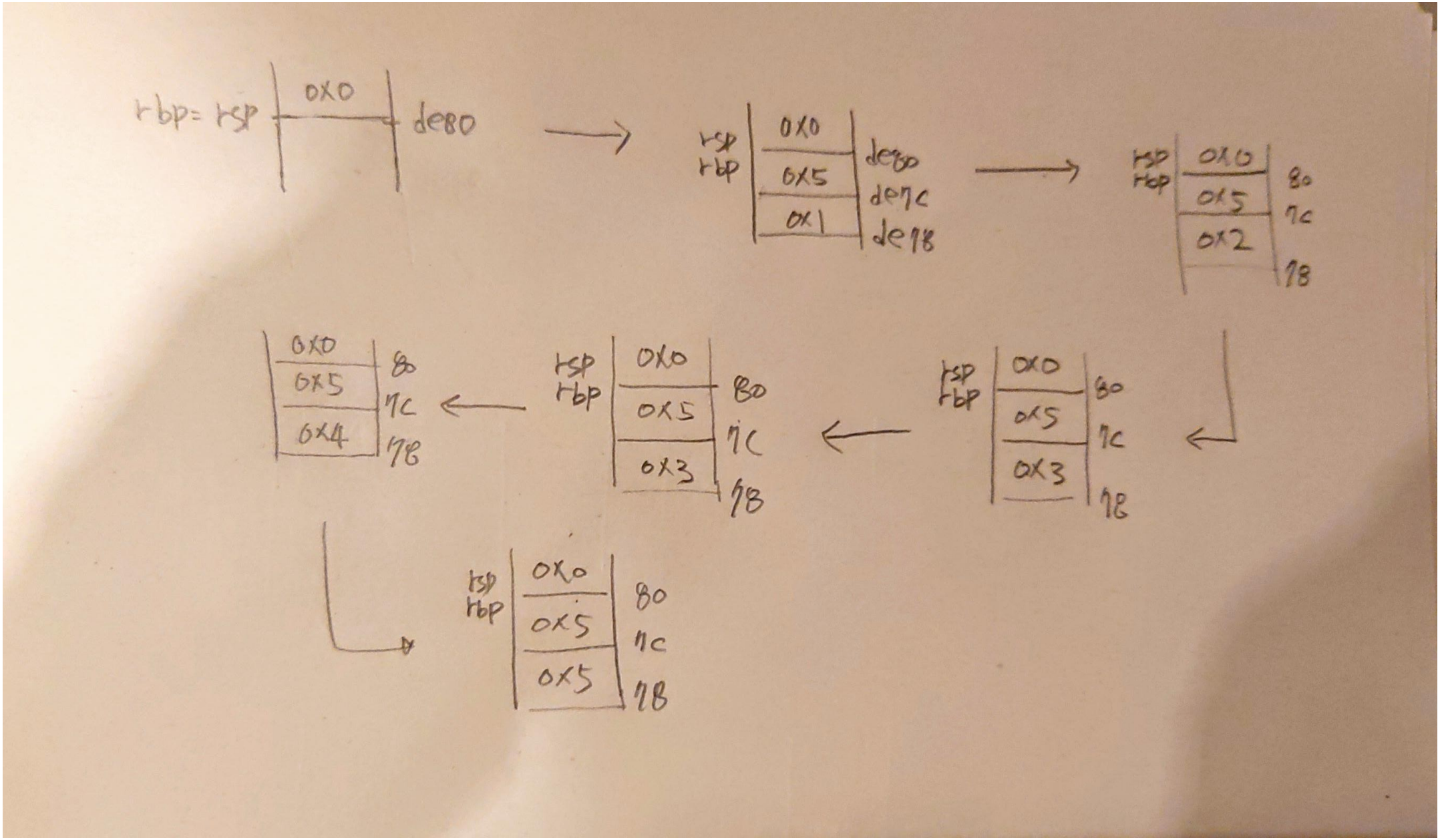
<+22> jmp 0x~~145: ~~145주소로 이동한다.

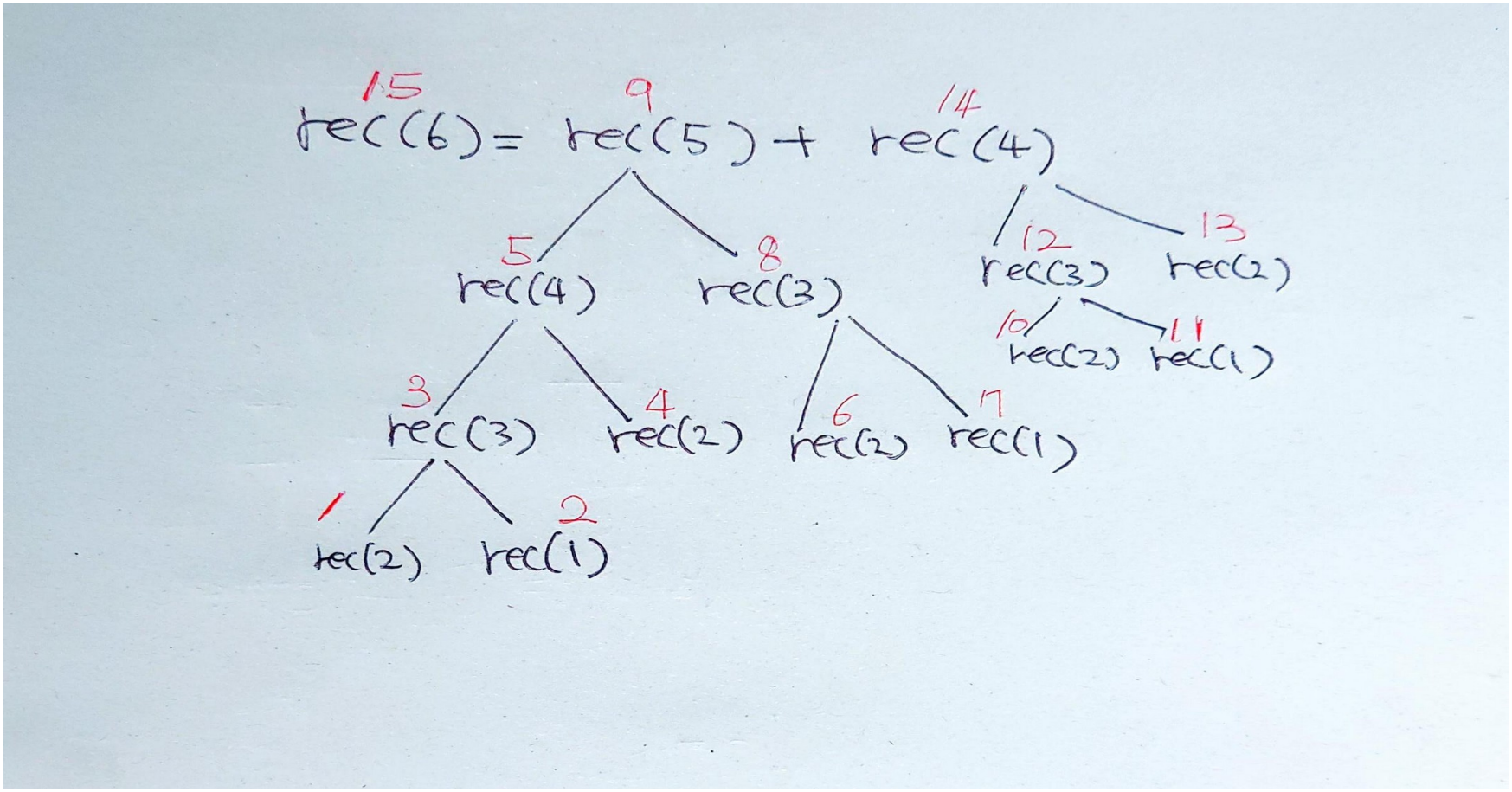
<+28> mov -0x8(%rbp), %eax: eax주소를 rbp - 8바이트에 있는 값으로 한다.

<+31> cmp -0x4(%rbp), %eax: rbp -4바이트에 있는 값(0x5)와 eax주소(0x1)를 비교한다.
<+34> jl 0x~~141: 비교한 값이 작으면 적힌 주소로 이동한다.

<+24> addl $0x1, -0x8(%rbp): rbp – 8바이트에 있는 값에 0x1을 더한다.

rbp -8바이트에 있는 값이 0x5가 될 때 까지 24 ~ 34과정을 반복한다.


rbp= rsp
0x0
de80
rsp
rbp
0x0
0x5
0x1
de80
de7c
de78
0x0
0x5
0x2
80
7c
78
0x0
0x5
0x3
0x0
0x5
0x4
0x0
0x5
0x5



빨간색 숫자는 함수가 실행되는 순서입니다.
앞 단계에서 구한 값을 뒷 단계에서는 모른다는 것을 주의해야합니다.
많은 횟수를 반복하는 재귀함수에서는 메모리 사용량이 많아지고 속도도 느려질 것으로 생각됩니다.