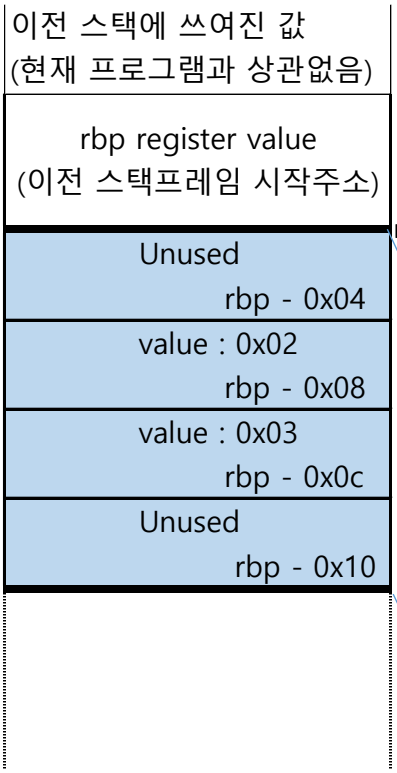
▣ GDB를 활용한 기계어 분석

2021/10/06 KKS

```
Dump of assembler code for function main:
   0x0000555555555160 <+0>:     endbr64
   0x0000555555555164 <+4>:     push   %rbp
   0x0000555555555165 <+5>:     mov    %rsp,%rbp
   0x0000555555555168 <+8>:     sub    $0x10,%rsp
   0x000055555555516c <+12>:    movl   $0x3,-0xc(%rbp)
   0x0000555555555173 <+19>:    movl   $0x2,-0x8(%rbp)
   0x000055555555517a <+26>:    mov    -0x8(%rbp),%edx
   0x000055555555517d <+29>:    mov    -0xc(%rbp),%eax
   0x0000555555555180 <+32>:    mov    %edx,%esi
   0x0000555555555182 <+34>:    mov    %eax,%edi
=> 0x0000555555555184 <+36>:    callq  0x555555555149 <mult>
   0x0000555555555189 <+41>:    mov    %eax,-0x4(%rbp)
   0x000055555555518c <+44>:    mov    -0x4(%rbp),%eax
   0x000055555555518f <+47>:    mov    %eax,%esi
   0x0000555555555191 <+49>:    lea    0xe6c(%rip),%rdi        # 0x555555556004
   0x0000555555555198 <+56>:    mov    $0x0,%eax
   0x000055555555519d <+61>:    callq  0x555555555050 <printf@plt>
   0x00005555555551a2 <+66>:    mov    $0x0,%eax
   0x00005555555551a7 <+71>:    leaveq
   0x00005555555551a8 <+72>:    retq
```



rbp = 이전 스택프레임의 베이스 메모리 주소

rsp = 스택 최상단 메모리 주소값

① push %rbp

rbp 레지스터에 쓰인 value가 최상위스택에 저장 됨

유의사항! push하면 자동으로 rsp 8byte아래로 자람

② mov %rsp, %rbp

rsp레지스터 값이 rbp값에 복사 됨

즉 진한 언더라인이 현재 rsp에 저장된 주소

rsp = rbp

rbp에 저장된 값은 새로운 스택프레임의 시작 주소

③ sub $0x10, %rsp

rsp 레지스터에서 0x10(지역변수 저장공간) 만큼 빼게 됨

즉, rsp에는 진한 언더라인 주소값이 쓰이게 됨

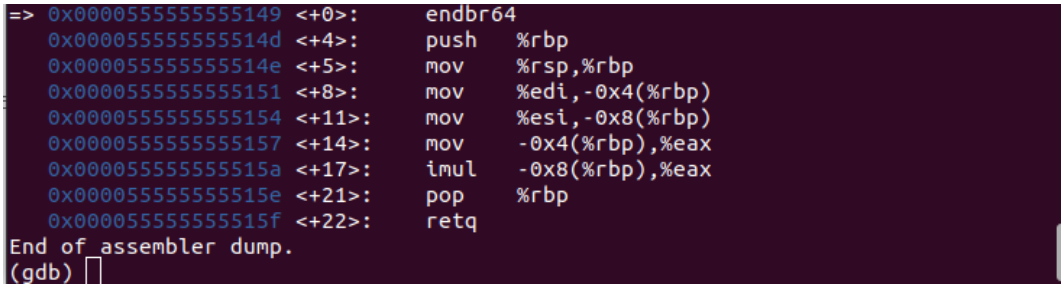
④ movl $0x3, -0xc(%rbp)

movl $0x2, -0x8(%rbp)

mov %edx, %esi

mov %eax, %edi

지역변수 공간에 값을 쓰는 과정



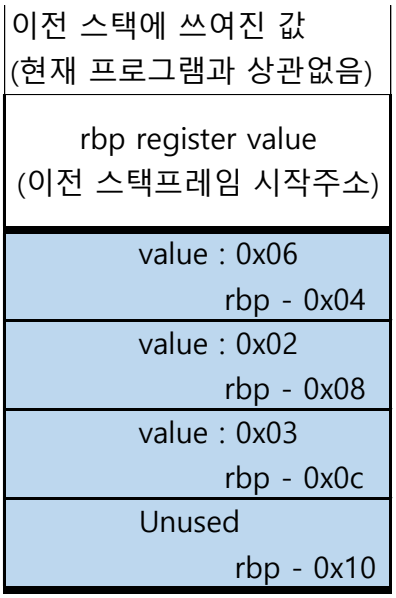
함수 호출전 스택 쓰여진값
rbp register value (이전 스택프레임 시작주소)
%edi value : 0x02 %rbp -0x4
%esi value : 0x03 %rbp -0x8

⑤ callq를 통해서 0x5555555..149에 저장된 mult 함수가 호출되게 된다.
이때 push %rbp하여 이전과 똑같이 스택프레임이 형성 되는것을 알 수 있다.
따라서 함수를 호출할 때 마다 스택은 아래로 증가하며 새로운 스택프레임이 계속 해서 생겨남을 알 수 있다.

⑥ 최종적으로 imul 곱셈연산을 통해 eax레지스터에 연산된 값이 저장된다.

⑦ pop %rbp 명령어 동작시 rbp레지스터가 미리 저장한 이전 스택프레임의 시작주소를 rbp 레지스터에 쓴다.
이때, rsp는 +0x8돼어 이전 push전의 주소값이 씌어진 것을 볼 수 있다.

```
Dump of assembler code for function main:
   0x0000555555555160 <+0>:     endbr64
   0x0000555555555164 <+4>:     push   %rbp
   0x0000555555555165 <+5>:     mov    %rsp,%rbp
   0x0000555555555168 <+8>:     sub    $0x10,%rsp
   0x000055555555516c <+12>:    movl   $0x3,-0xc(%rbp)
   0x0000555555555173 <+19>:    movl   $0x2,-0x8(%rbp)
   0x000055555555517a <+26>:    mov    -0x8(%rbp),%edx
   0x000055555555517d <+29>:    mov    -0xc(%rbp),%eax
   0x0000555555555180 <+32>:    mov    %edx,%esi
   0x0000555555555182 <+34>:    mov    %eax,%edi
=> 0x0000555555555184 <+36>:    callq  0x555555555149 <mult>
   0x0000555555555189 <+41>:    mov    %eax,-0x4(%rbp)
   0x000055555555518c <+44>:    mov    -0x4(%rbp),%eax
   0x000055555555518f <+47>:    mov    %eax,%esi
   0x0000555555555191 <+49>:    lea    0xe6c(%rip),%rdi        # 0x555555556004
   0x0000555555555198 <+56>:    mov    $0x0,%eax
   0x000055555555519d <+61>:    callq  0x555555555050 <printf@plt>
   0x00005555555551a2 <+66>:    mov    $0x0,%eax
   0x00005555555551a7 <+71>:    leaveq
   0x00005555555551a8 <+72>:    retq
```



⑧ %eax, -0x4(%rbp) eax레지스터에 저장된 값이 rbp-0x04 메모리 공간에 씌어 진다.

⑨ printf 출력과정의 bnd jmpq 에 관해서는 더 공부가 필요하다.

```
0x0000555555555191 <+49>:    lea    0xe6c(%rip),%rdi
```

이 부분에 대해서 공부 및 질문 할 것

⑩ leaveq는 push %rsp, %rbp
pop %rbp 를 의미한다.
즉, 사용한 스택프레임을 완전히 없애는 것이다.