With bomblab

2020년 리버싱 교육 1회차

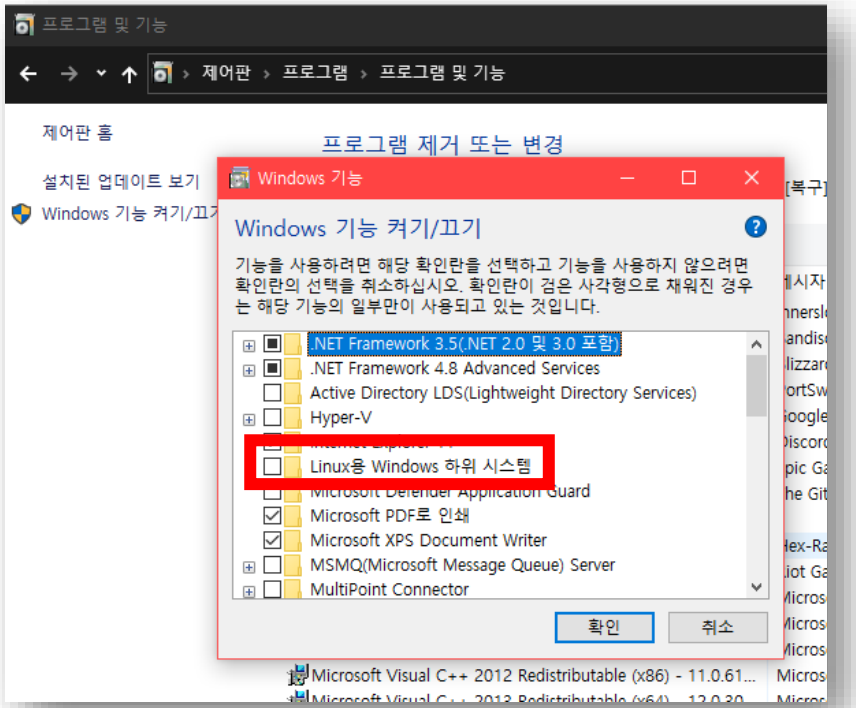
index

금일 교육 구성

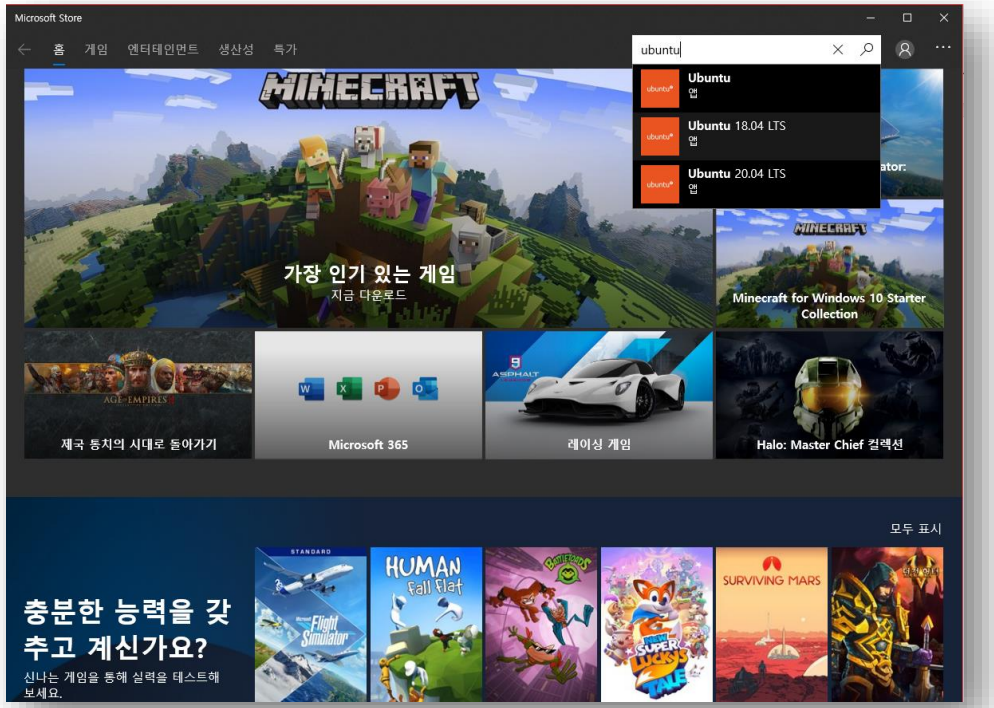
0. WSL 설치
1. 바이너리 리버싱에 대해서
2. 기초적인 컴퓨터 시스템
3. 어셈블리 기초
4. 밤랩 소개 및 1단계 해법

환경 구축

WSL 설치



제어판 -〉 프로그램 및 기능 -〉 좌측 탭의 Windows 기능 켜기/끄기
-〉 Linux용 Windows 하위 시스템 체크(다시시작 필요)



Microsoft Store에서 Ubuntu 18.04 LTS 설치

환경 구축

WSL 설치

아래 명령 차례대로 입력

sudo apt update

sudo apt install gdb gcc build-essential git

바이너리 리버싱에 대해서

바이너리가 무엇인가?

바이너리(binary)

0과 1로 이루어진 이진 상태를 일컫는 말이다.
컴퓨터의 실행 파일을 의미하기도 한다.

* 이 강의에서는 ELF(리눅스 실행파일)에 한정하여 바이너리라고 칭한다.

바이너리 리버싱에 대해서

왜 리버싱을 하는가?

리버싱(역공학, Reverse Engineering)
완성된 제품(SW/OS/HW)을 분석하여 설계 단계로 돌리는 기술
(설계 -> 개발 -> 제품화의 역과정)

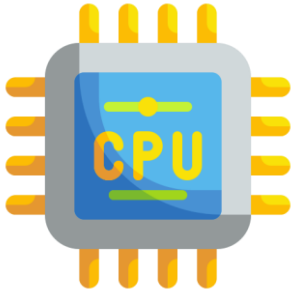
우리(학부생)의 관점
- 바이너리(실행파일)와 기본적인 컴퓨터 시스템에 대한 이해도가 올라간다.

개발자의 관점
- 프로그램에 존재하는 잠재적인 버그를 잡을 수 있다(코드를 한줄씩 실행하면서)

해커(or 보안 전문가)의 관점
- 컴퓨터 시스템에 대한 이해도가 있는 상태에서 프로그램에 존재하는 보안상 약점을 발견할 수 있다.

기초적인 컴퓨터 시스템

CPU 레지스터



1 + 2 = ?

CPU 레지스터

연산을 하기 위해서는 피연산자를 담을 공간이 필요

기초적인 컴퓨터 시스템

CPU 레지스터

rax rbx rcx rdx rsi rdi

rbp r8 r9 r10 r11 r12 r13 r14 r15

rsp

rip

범용 레지스터

용도가 특별하게 정해지지 않은 레지스터로, 변수와 같은 역할을 한다.
용도가 정해져 있지 않지만 때에 따라 그 쓰임새가 정해져 있는 경우도 존재

(예시 : rax는 함수 리턴 값, rsi는 함수 인자 값)

기초적인 컴퓨터 시스템

CPU 레지스터

rax rbx rcx rdx rsi rdi

rbp r8 r9 r10 r11 r12 r13 r14 r15

rsp

rip

함수 호출 인자

함수가 호출될 때 필요한 인자들을 넘겨주는 역할

rdi rsi rcx rdx …

기초적인 컴퓨터 시스템

CPU 레지스터

rax rbx rcx rdx rsi rdi

rbp r8 r9 r10 r11 r12 r13 r14 r15

rsp

rip

스택 포인터

스택 메모리의 가장 위쪽을 가르킴. 스택은 함수가 사용할 지역 변수들을 저장하기 위해 준비해놓은 공간임.

기초적인 컴퓨터 시스템

CPU 레지스터

rax rbx rcx rdx rsi rdi

rbp r8 r9 r10 r11 r12 r13 r14 r15

rsp

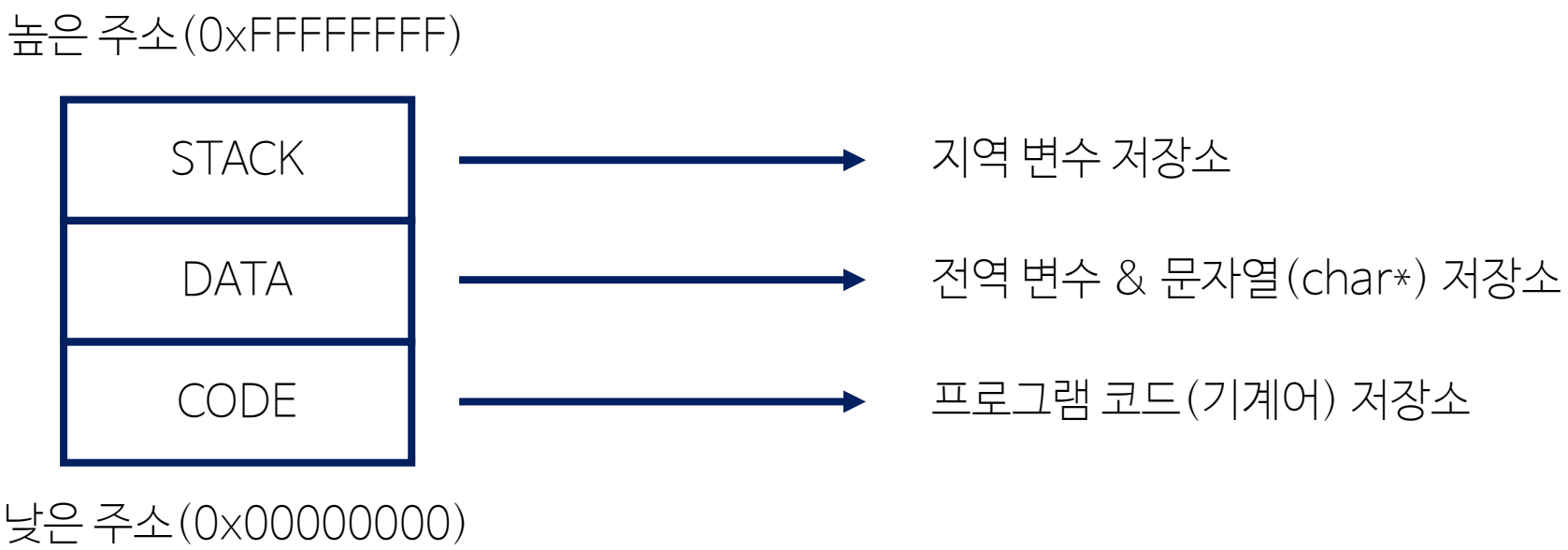
rip

프로그램 카운터

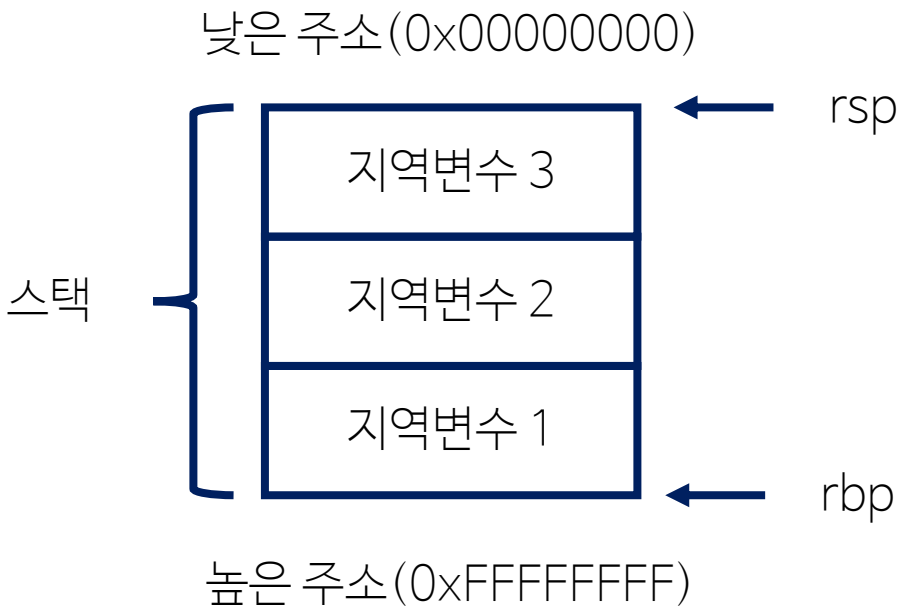
rip는 프로그램 카운터(Program Counter)의 역할을 한다. 프로그램 카운터는 다음에 실행할 명령어가 위치한 주소를 가르킨다.

기초적인 컴퓨터 시스템

메모리

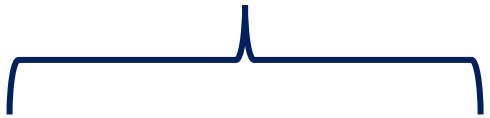


메모리


낮은 주소(0x00000000)
rsp
지역변수 3
스택
지역변수 2
지역변수 1
rbp
높은 주소(0xFFFFFFFF)

어셈블리 형태

상수, 레지스터, 주소 등..



[연산자] [피연산자1] [피연산자2]

*예시 : add rax, rbx

대표적인 연산자

mov a, b	b를 a에 복사한다 (a = b)
lea a, b	b의 주소에 있는 값을 a에 복사한다 (a = *b)
cmp a, b	a와 b를 비교한다.

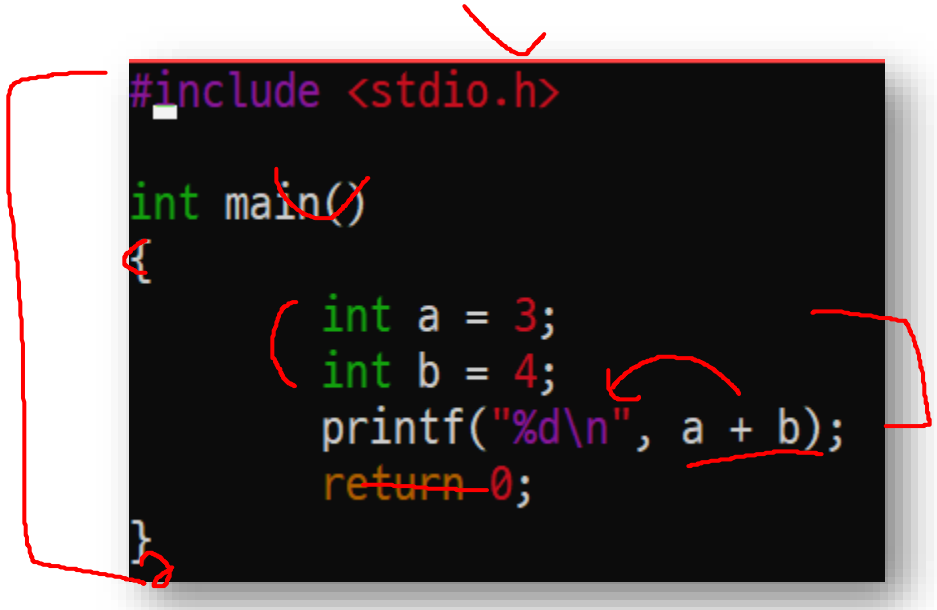
대표적인 연산자

add a, b	a와 b를 더해서 a에 결과를 넣는다 (a += b)
sub a, b	a와 b를 뺀 결과를 a에 넣는다 (a -= b)
imul a, b	a와 b를 곱한 결과를 a에 너허는다 (a *=b)
xor a, b	a와 b를 xor 한 결과를 a 에 넣는다 (a^=b)

대표적인 연산자

jmp	해당 코드로 점프
je	cmp 결과가 같으면 점프
jne	cmp 결과가 다르면 점프
call	함수 호출

실습 1 – 계산기



```c
#include <stdio.h>

int main()
{
    int a = 3;
    int b = 4;
    printf("%d\n", a + b);
    return 0;
}
```

코드 작성 후 컴파일

실습 1 – 계산기

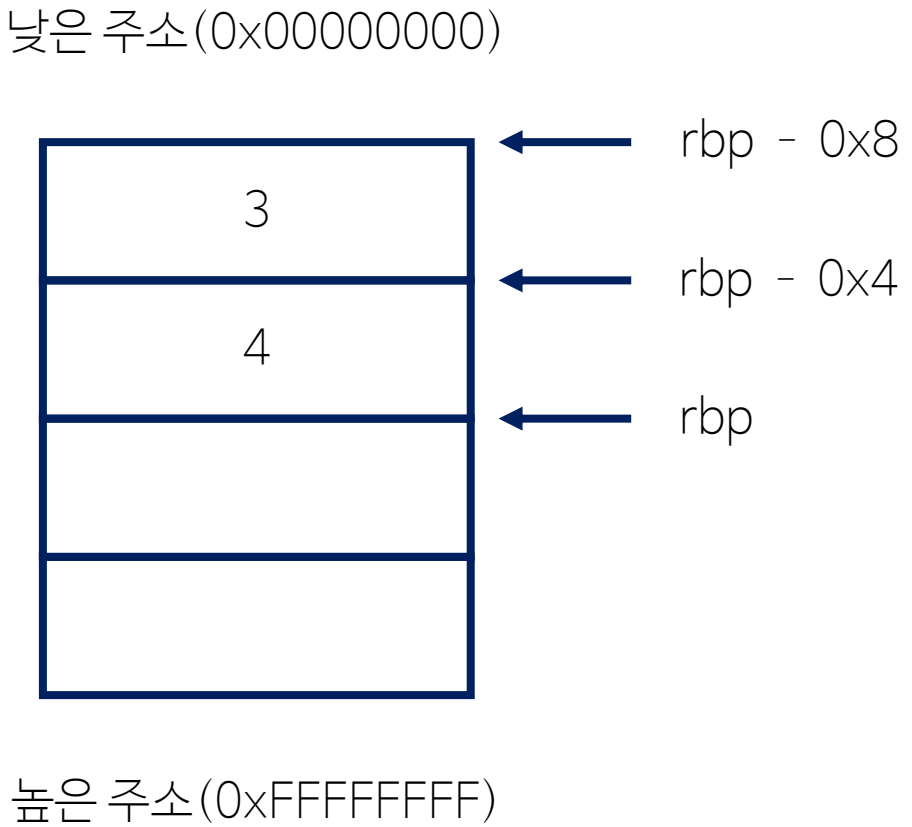
```
bomblab-edu@DESKTOP-0JLISUU:~$ gdb prac1
```

```
(gdb) set disassembly-flavor intel
```

```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    DWORD PTR [rbp-0x4],0x4
   0x0000000000000660 <+22>:    mov    edx,DWORD PTR [rbp-0x8]
   0x0000000000000663 <+25>:    mov    eax,DWORD PTR [rbp-0x4]
   0x0000000000000666 <+28>:    add    eax,edx
   0x0000000000000668 <+30>:    mov    esi,eax
   0x000000000000066a <+32>:    lea    rdi,[rip+0xa3]        # 0x714
   0x0000000000000671 <+39>:    mov    eax,0x0
   0x0000000000000676 <+44>:    call   0x520 <printf@plt>
   0x000000000000067b <+49>:    mov    eax,0x0
   0x0000000000000680 <+54>:    leave
   0x0000000000000681 <+55>:    ret
End of assembler dump.
(gdb)
```

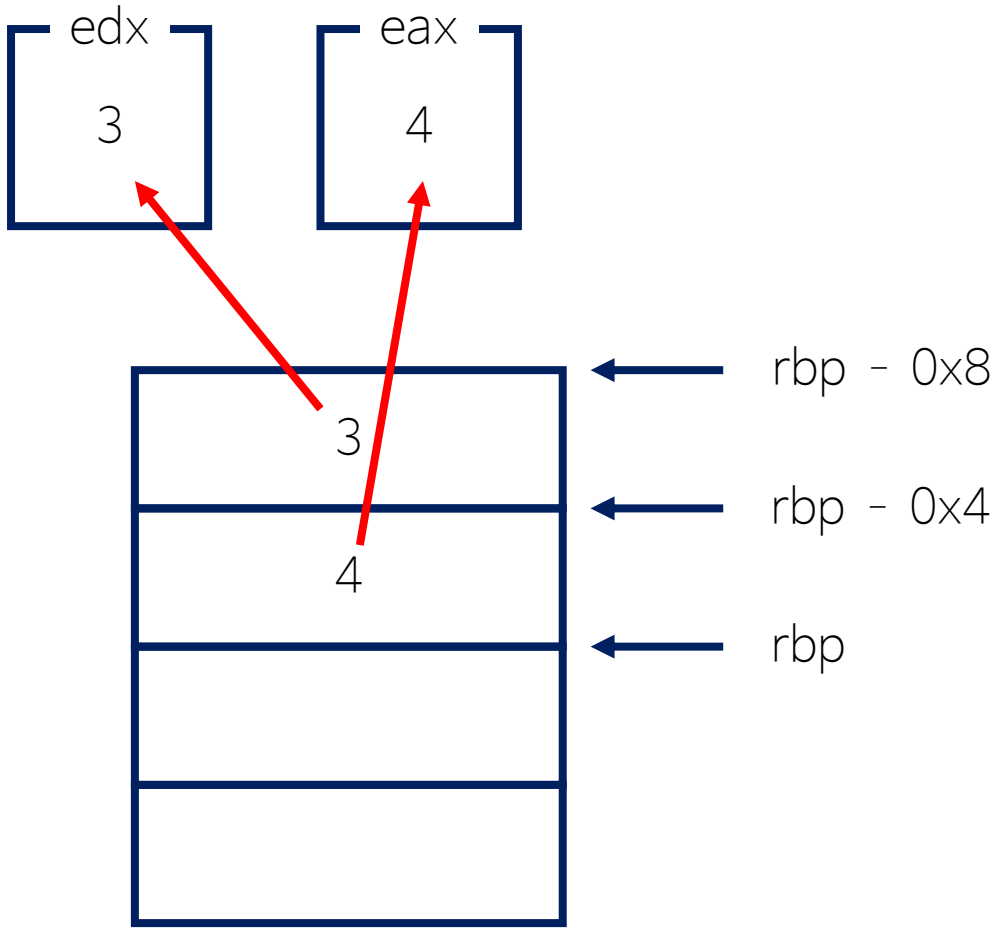
실습 1 – 계산기

```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    DWORD PTR [rbp-0x4],0x4
   0x0000000000000660 <+22>:    mov    edx,DWORD PTR [rbp-0x8]
   0x0000000000000663 <+25>:    mov    eax,DWORD PTR [rbp-0x4]
   0x0000000000000666 <+28>:    add    eax,edx
   0x0000000000000668 <+30>:    mov    esi,eax
   0x000000000000066a <+32>:    lea    rdi,[rip+0xa3]        # 0x714
   0x0000000000000671 <+39>:    mov    eax,0x0
   0x0000000000000676 <+44>:    call   0x520 <printf@plt>
   0x000000000000067b <+49>:    mov    eax,0x0
   0x0000000000000680 <+54>:    leave
   0x0000000000000681 <+55>:    ret
End of assembler dump.
(gdb)
```



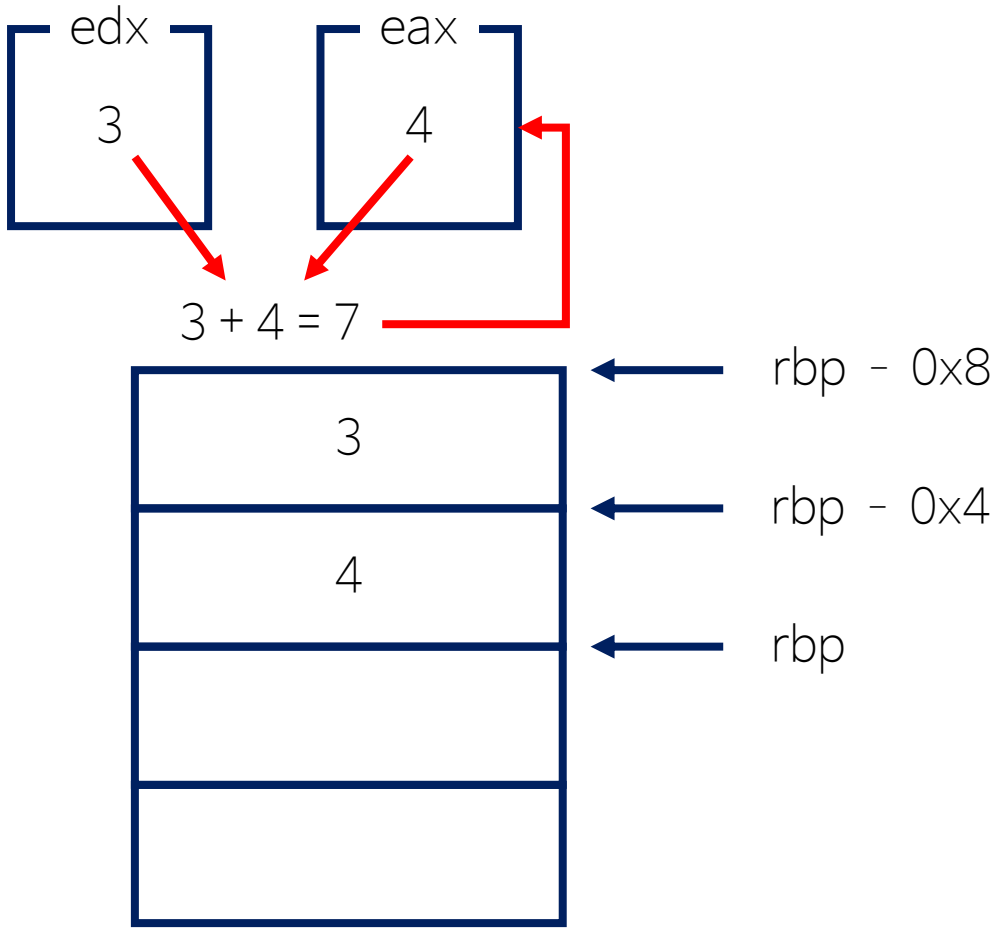
실습 1 – 계산기

```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    DWORD PTR [rbp-0x4],0x4
   0x0000000000000660 <+22>:    mov    edx,DWORD PTR [rbp-0x8]
   0x0000000000000663 <+25>:    mov    eax,DWORD PTR [rbp-0x4]
   0x0000000000000666 <+28>:    add    eax,edx
   0x0000000000000668 <+30>:    mov    esi,eax
   0x000000000000066a <+32>:    lea    rdi,[rip+0xa3]        # 0x714
   0x0000000000000671 <+39>:    mov    eax,0x0
   0x0000000000000676 <+44>:    call   0x520 <printf@plt>
   0x000000000000067b <+49>:    mov    eax,0x0
   0x0000000000000680 <+54>:    leave
   0x0000000000000681 <+55>:    ret
End of assembler dump.
(gdb)
```



실습 1 – 계산기

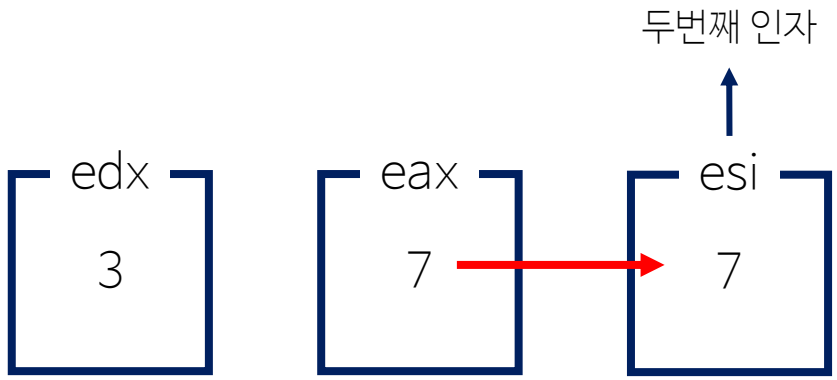
```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    DWORD PTR [rbp-0x4],0x4
   0x0000000000000660 <+22>:    mov    edx,DWORD PTR [rbp-0x8]
   0x0000000000000663 <+25>:    mov    eax,DWORD PTR [rbp-0x4]
   0x0000000000000666 <+28>:    add    eax,edx
   0x0000000000000668 <+30>:    mov    esi,eax
   0x000000000000066a <+32>:    lea    rdi,[rip+0xa3]        # 0x714
   0x0000000000000671 <+39>:    mov    eax,0x0
   0x0000000000000676 <+44>:    call   0x520 <printf@plt>
   0x000000000000067b <+49>:    mov    eax,0x0
   0x0000000000000680 <+54>:    leave
   0x0000000000000681 <+55>:    ret
End of assembler dump.
(gdb)
```

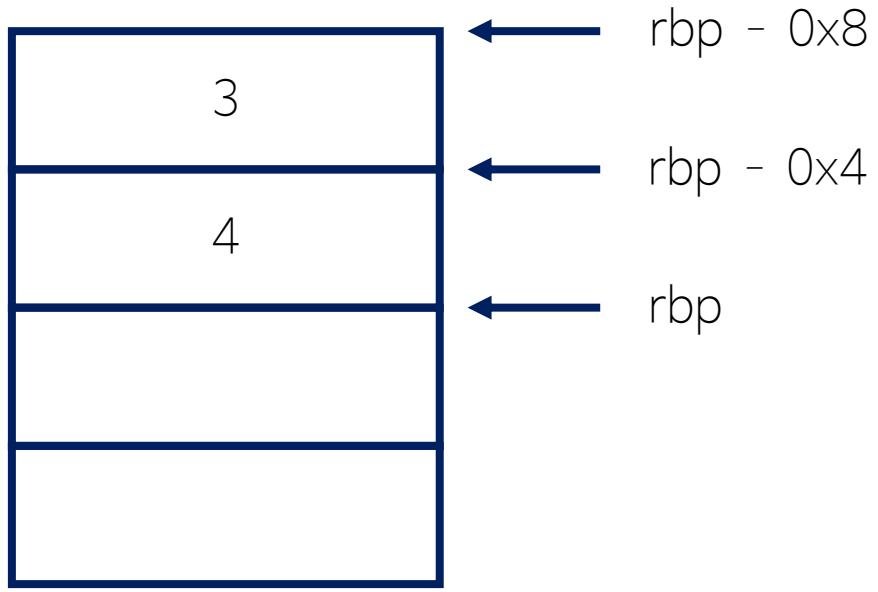


어셈블리 기초

실습 1 – 계산기

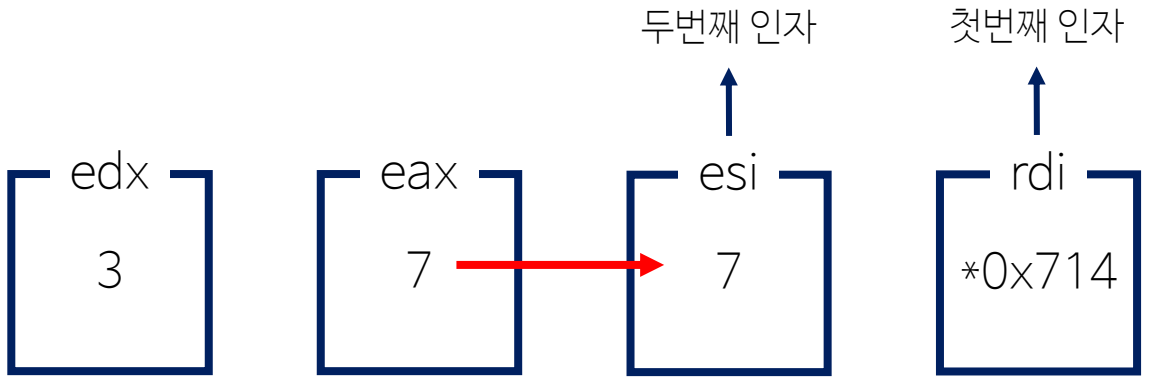
```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    DWORD PTR [rbp-0x4],0x4
   0x0000000000000660 <+22>:    mov    edx,DWORD PTR [rbp-0x8]
   0x0000000000000663 <+25>:    mov    eax,DWORD PTR [rbp-0x4]
   0x0000000000000666 <+28>:    add    eax,edx
   0x0000000000000668 <+30>:    mov    esi,eax
   0x000000000000066a <+32>:    lea    rdi,[rip+0xa3]        # 0x714
   0x0000000000000671 <+39>:    mov    eax,0x0
   0x0000000000000676 <+44>:    call   0x520 <printf@plt>
   0x000000000000067b <+49>:    mov    eax,0x0
   0x0000000000000680 <+54>:    leave
   0x0000000000000681 <+55>:    ret
End of assembler dump.
(gdb)
```



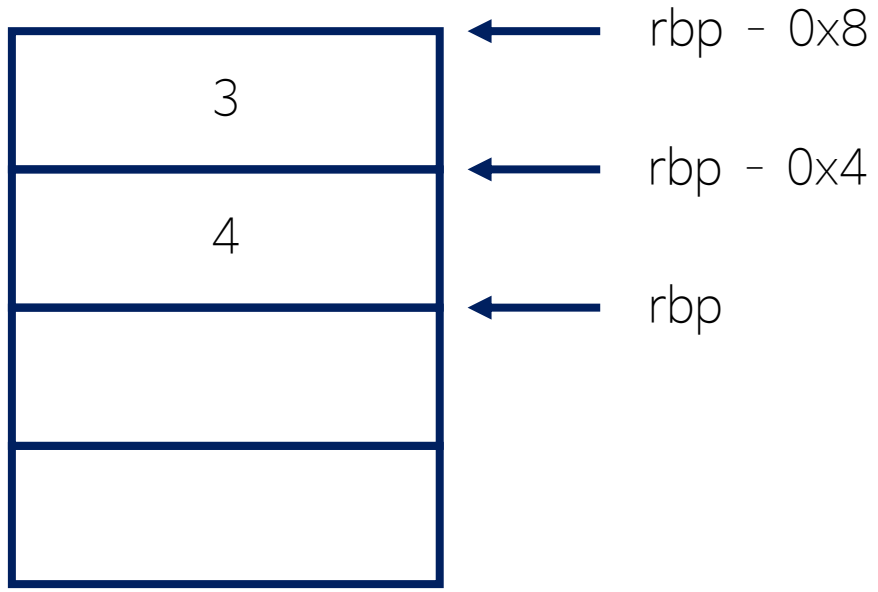


어셈블리 기초

실습 1 – 계산기



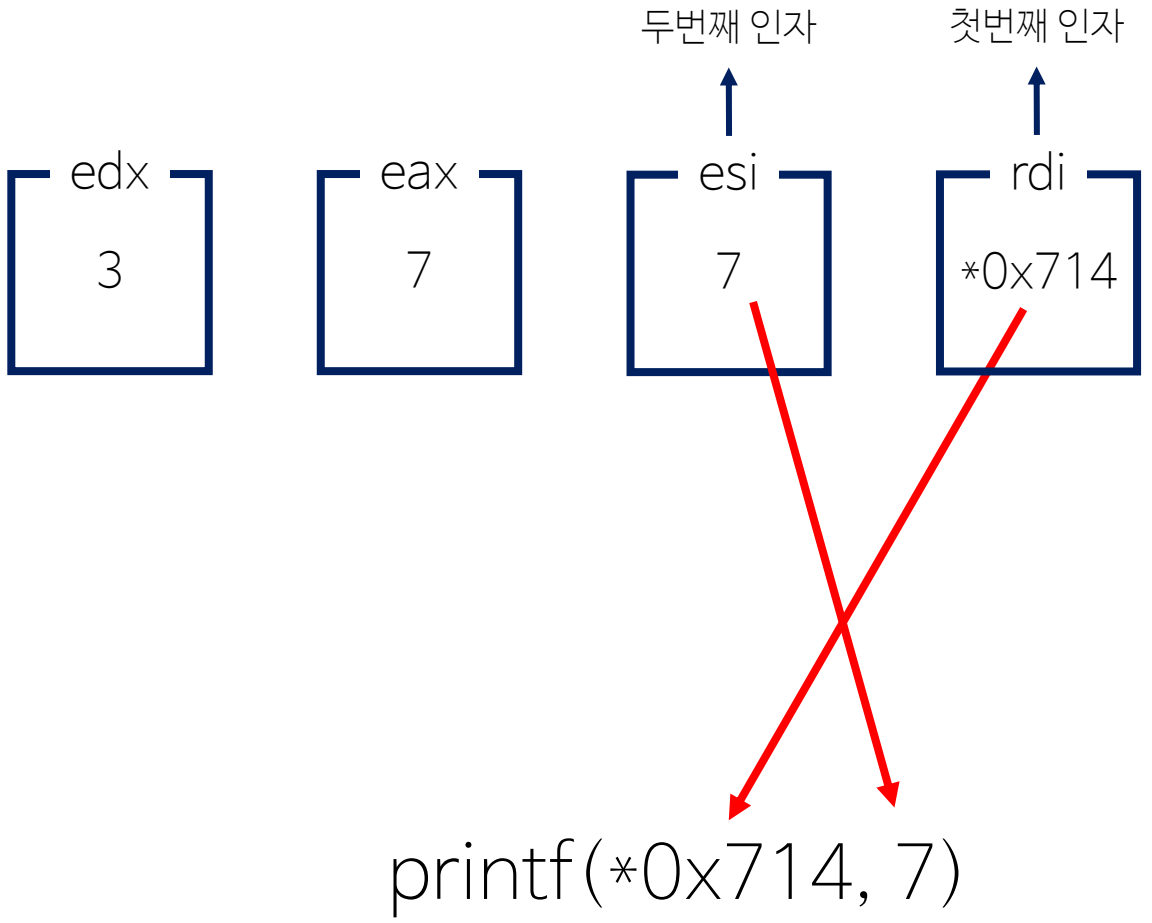

```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    DWORD PTR [rbp-0x4],0x4
   0x0000000000000660 <+22>:    mov    edx,DWORD PTR [rbp-0x8]
   0x0000000000000663 <+25>:    mov    eax,DWORD PTR [rbp-0x4]
   0x0000000000000666 <+28>:    add    eax,edx
   0x0000000000000668 <+30>:    mov    esi,eax
   0x000000000000066a <+32>:    lea    rdi,[rip+0xa3]        # 0x714
   0x0000000000000671 <+39>:    mov    eax,0x0
   0x0000000000000676 <+44>:    call   0x520 <printf@plt>
   0x000000000000067b <+49>:    mov    eax,0x0
   0x0000000000000680 <+54>:    leave
   0x0000000000000681 <+55>:    ret
End of assembler dump.
(gdb)
```



어셈블리 기초

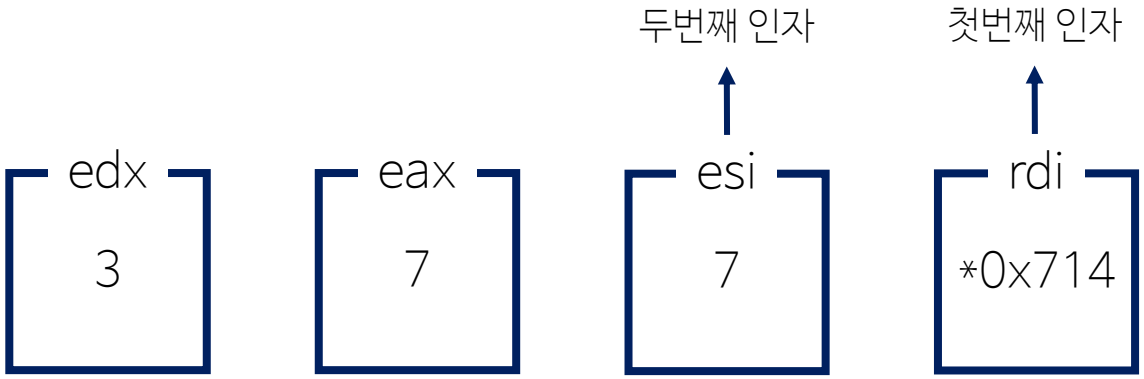
실습 1 – 계산기

```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    DWORD PTR [rbp-0x4],0x4
   0x0000000000000660 <+22>:    mov    edx,DWORD PTR [rbp-0x8]
   0x0000000000000663 <+25>:    mov    eax,DWORD PTR [rbp-0x4]
   0x0000000000000666 <+28>:    add    eax,edx
   0x0000000000000668 <+30>:    mov    esi,eax
   0x000000000000066a <+32>:    lea    rdi,[rip+0xa3]        # 0x714
   0x0000000000000671 <+39>:    mov    eax,0x0
   0x0000000000000676 <+44>:    call   0x520 <printf@plt>
   0x000000000000067b <+49>:    mov    eax,0x0
   0x0000000000000680 <+54>:    leave
   0x0000000000000681 <+55>:    ret
End of assembler dump.
(gdb)
```



어셈블리 기초

실습 1 – 계산기

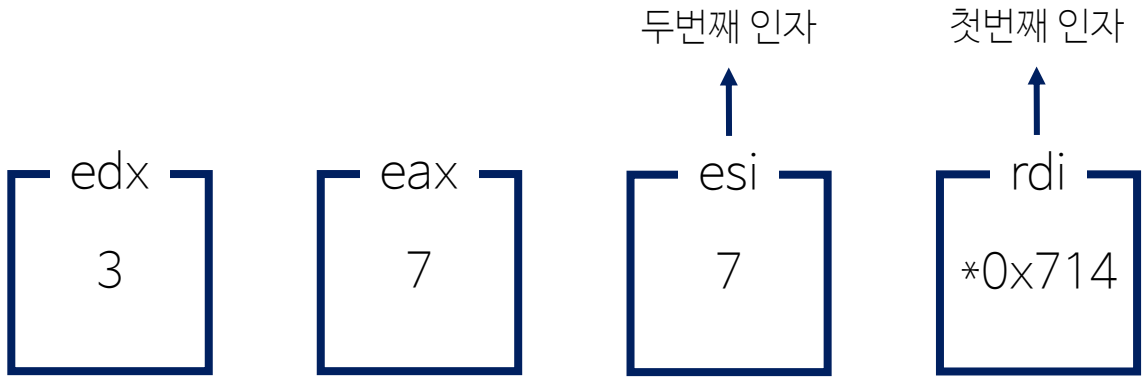



```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    DWORD PTR [rbp-0x4],0x4
   0x0000000000000660 <+22>:    mov    edx,DWORD PTR [rbp-0x8]
   0x0000000000000663 <+25>:    mov    eax,DWORD PTR [rbp-0x4]
   0x0000000000000666 <+28>:    add    eax,edx
   0x0000000000000668 <+30>:    mov    esi,eax
   0x000000000000066a <+32>:    lea    rdi,[rip+0xa3]        # 0x714
   0x0000000000000671 <+39>:    mov    eax,0x0
   0x0000000000000676 <+44>:    call   0x520 <printf@plt>
   0x000000000000067b <+49>:    mov    eax,0x0
   0x0000000000000680 <+54>:    leave
   0x0000000000000681 <+55>:    ret
End of assembler dump.
(gdb)
```

printf(*0x714, 7)
return 0;

어셈블리 기초

실습 1 – 계산기



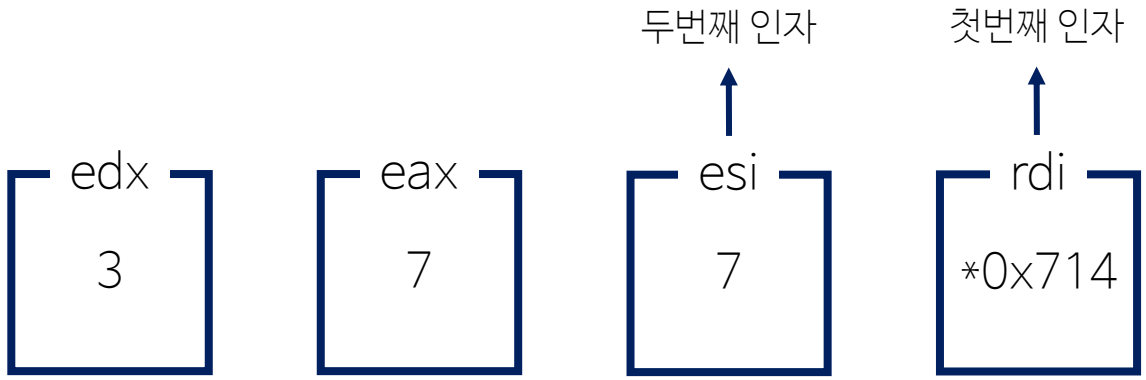

```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    DWORD PTR [rbp-0x4],0x4
   0x0000000000000660 <+22>:    mov    edx,DWORD PTR [rbp-0x8]
   0x0000000000000663 <+25>:    mov    eax,DWORD PTR [rbp-0x4]
   0x0000000000000666 <+28>:    add    eax,edx
   0x0000000000000668 <+30>:    mov    esi,eax
   0x000000000000066a <+32>:    lea    rdi,[rip+0xa3]        # 0x714
   0x0000000000000671 <+39>:    mov    eax,0x0
   0x0000000000000676 <+44>:    call   0x520 <printf@plt>
   0x000000000000067b <+49>:    mov    eax,0x0
   0x0000000000000680 <+54>:    leave
   0x0000000000000681 <+55>:    ret
End of assembler dump.
(gdb)
```

```
(gdb) x/s 0x714
0x714:  "%d\n"
```

printf(“%d₩n”, 7)
return 0;

어셈블리 기초

실습 1 – 계산기

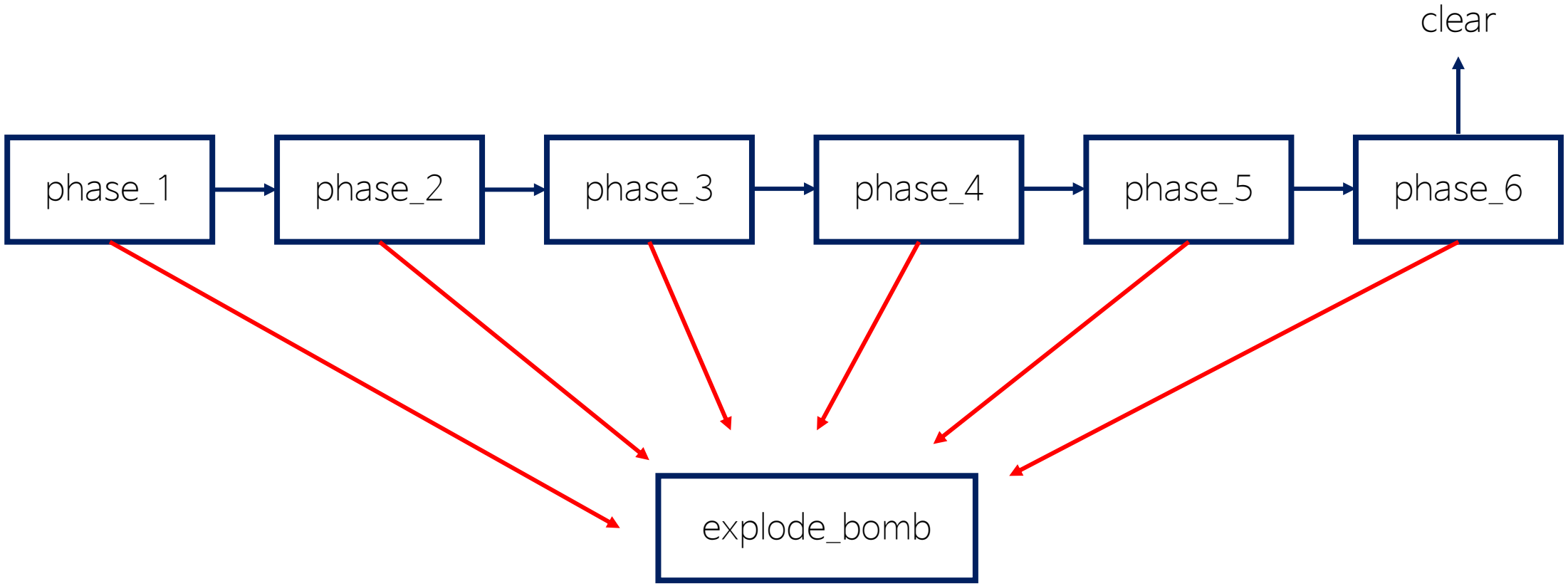



```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    DWORD PTR [rbp-0x4],0x4
   0x0000000000000660 <+22>:    mov    edx,DWORD PTR [rbp-0x8]
   0x0000000000000663 <+25>:    mov    eax,DWORD PTR [rbp-0x4]
   0x0000000000000666 <+28>:    add    eax,edx
   0x0000000000000668 <+30>:    mov    esi,eax
   0x000000000000066a <+32>:    lea    rdi,[rip+0xa3]        # 0x714
   0x0000000000000671 <+39>:    mov    eax,0x0
   0x0000000000000676 <+44>:    call   0x520 <printf@plt>
   0x000000000000067b <+49>:    mov    eax,0x0
   0x0000000000000680 <+54>:    leave
   0x0000000000000681 <+55>:    ret
End of assembler dump.
(gdb)
```

```
bomblab-edu@DESKTOP-0JLISUU:~$ ./prac1
7
```

밤랩 소개 및 1단계 해법

BOMBLAB



밤랩 소개 및 1단계 해법

BOMBLAB

git clone https://github.com/MINIBEEF/2020-bomblab-edu.git

```
bomblab-edu@DESKTOP-0JLISUU:~$ git clone https://github.com/MINIBEEF/2020-bomblab-edu.git
Cloning into '2020-bomblab-edu'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (7/7), done.
```

cd 2020-bomblab-edu ; chmod +x bomb

밤랩 소개 및 1단계 해법

BOMBLAB

b* 위치 : 브레이크 포인트

ni : 다음 스텝

disas : 현재 위치

disas 함수이름 : 함수 코드 보기

i r : 현재 레지스터 상태

r : 시작

밤랩 소개 및 1단계 해법

BOMBLAB

```
Dump of assembler code for function phase_1:
   0x0000000000400ee0 <+0>:     sub    rsp,0x8
   0x0000000000400ee4 <+4>:     mov    esi,0x402400
   0x0000000000400ee9 <+9>:     call   0x401338 <strings_not_equal>
   0x0000000000400eee <+14>:    test   eax,eax
   0x0000000000400ef0 <+16>:    je     0x400ef7 <phase_1+23>
   0x0000000000400ef2 <+18>:    call   0x40143a <explode_bomb>
   0x0000000000400ef7 <+23>:    add    rsp,0x8
   0x0000000000400efb <+27>:    ret
End of assembler dump.
```

인자로 들어온 두 문자열이 같음을 확인 -〉 같으면 통과 -〉 아니면 폭탄

밤랩 소개 및 1단계 해법

BOMBLAB

```
Dump of assembler code for function phase_1:
   0x0000000000400ee0 <+0>:     sub    rsp,0x8
   0x0000000000400ee4 <+4>:     mov    esi,0x402400
   0x0000000000400ee9 <+9>:     call   0x401338 <strings_not_equal>
   0x0000000000400eee <+14>:    test   eax,eax
   0x0000000000400ef0 <+16>:    je     0x400ef7 <phase_1+23>
   0x0000000000400ef2 <+18>:    call   0x40143a <explode_bomb>
   0x0000000000400ef7 <+23>:    add    rsp,0x8
   0x0000000000400efb <+27>:    ret
End of assembler dump.
```

```
(gdb) x/s 0x402400
0x402400:       "Border relations with Canada have never been better."
```

두번째 인자 획득, 첫번째 인자는?

밤랩 소개 및 1단계 해법

BOMBLAB

```
(gdb) b *phase_1
Breakpoint 1 at 0x400ee0
(gdb) r
Starting program: /home/bomblab-edu/2020-bomblab-edu/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
hello this is argos education
```

브레이크 포인트 설치 후(b *phase_1) -〉 실행(r)

-〉 우리가 알아볼 수 있는 문자열 입력

밤랩 소개 및 1단계 해법

BOMBLAB

```
(gdb) i r
rax            0x603780 6305664
rbx            0x0      0
rcx            0x1d     29
rdx            0x1      1
rsi            0x402400 4203520
rdi            0x603780 6305664
rbp            0x402210 0x402210 <__libc_csu_init>
rsp            0x7fffffffee180   0x7fffffffee180
r8             0x60448e 6309006
r9             0x7ffff16ed40   140737473080640
r10            0x3      3
r11            0x7ffff030920   140737471777056
r12            0x400c90 4197520
r13            0x7fffffffee270   140737488282224
r14            0x0      0
r15            0x0      0
rip            0x400ee9 0x400ee9 <phase_1+9>
eflags         0x202    [ IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
```

```
(gdb) x/s 0x603780
0x603780 <input_strings>:       "hello this is argos education"
```

첫번째 인자는 우리가 입력한 값…

즉, 우리가 입력한 값이 두번째 문자열과 같아야함

밤랩 소개 및 1단계 해법

BOMBLAB

```
(gdb) c
Continuing.

BOOM!!!
The bomb has blown up.
[Inferior 1 (process 5551) exited with code 010]
(gdb)
```

역시나 다른 문자열을 넣으면 폭탄 터짐

```
(gdb) x/s 0x402400
0x402400:       "Border relations with Canada have never been better."
(gdb) r
Starting program: /home/bomblab-edu/2020-bomblab-edu/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
```

```
(gdb) c
Continuing.
Phase 1 defused. How about the next one?
```

1단계 해결

밤랩 소개 및 1단계 해법

BOMBLAB

```
(gdb) c
Continuing.

BOOM!!!
The bomb has blown up.
[Inferior 1 (process 5551) exited with code 010]
(gdb)
```

역시나 다른 문자열을 넣으면 폭탄 터짐

```
(gdb) x/s 0x402400
0x402400:       "Border relations with Canada have never been better."
(gdb) r
Starting program: /home/bomblab-edu/2020-bomblab-edu/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
```

```
(gdb) c
Continuing.
Phase 1 defused. How about the next one?
```

1단계 해결

간단한 리버싱

과제

```
Dump of assembler code for function main:
   0x0000000000000664 <+0>:     push   rbp
   0x0000000000000665 <+1>:     mov    rbp,rsp
   0x0000000000000668 <+4>:     sub    rsp,0x10
   0x000000000000066c <+8>:     mov    DWORD PTR [rbp-0x8],0x2
   0x0000000000000673 <+15>:    mov    DWORD PTR [rbp-0x4],0x7
   0x000000000000067a <+22>:    mov    edx,DWORD PTR [rbp-0x4]
   0x000000000000067d <+25>:    mov    eax,DWORD PTR [rbp-0x8]
   0x0000000000000680 <+28>:    mov    esi,edx
   0x0000000000000682 <+30>:    mov    edi,eax
   0x0000000000000684 <+32>:    call   0x64a <sum>
   0x0000000000000689 <+37>:    mov    esi,eax
   0x000000000000068b <+39>:    lea    rdi,[rip+0xa2]        # 0x734
   0x0000000000000692 <+46>:    mov    eax,0x0
   0x0000000000000697 <+51>:    call   0x520 <printf@plt>
   0x000000000000069c <+56>:    mov    eax,0x0
   0x00000000000006a1 <+61>:    leave
   0x00000000000006a2 <+62>:    ret
End of assembler dump.
```

```
Dump of assembler code for function sum:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     mov    DWORD PTR [rbp-0x14],edi
   0x0000000000000651 <+7>:     mov    DWORD PTR [rbp-0x18],esi
   0x0000000000000654 <+10>:    mov    edx,DWORD PTR [rbp-0x14]
   0x0000000000000657 <+13>:    mov    eax,DWORD PTR [rbp-0x18]
   0x000000000000065a <+16>:    add    eax,edx
   0x000000000000065c <+18>:    mov    DWORD PTR [rbp-0x4],eax
   0x000000000000065f <+21>:    mov    eax,DWORD PTR [rbp-0x4]
   0x0000000000000662 <+24>:    pop    rbp
   0x0000000000000663 <+25>:    ret
End of assembler dump.
```

```
(gdb) x/s 0x734
0x734:  "%d\n"
```

다음시간까지 해당 프로그램의 결과 값 알아오기

(심화) 해당 프로그램 C 코드로 복원해보기