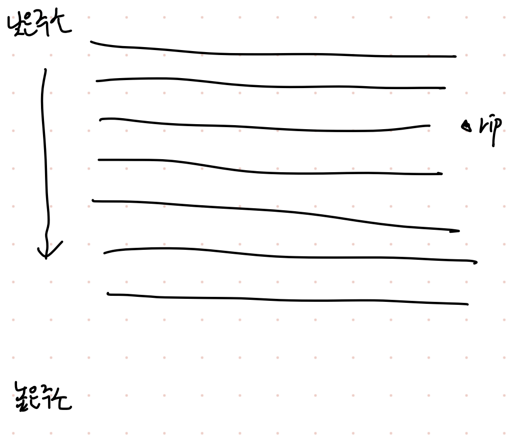
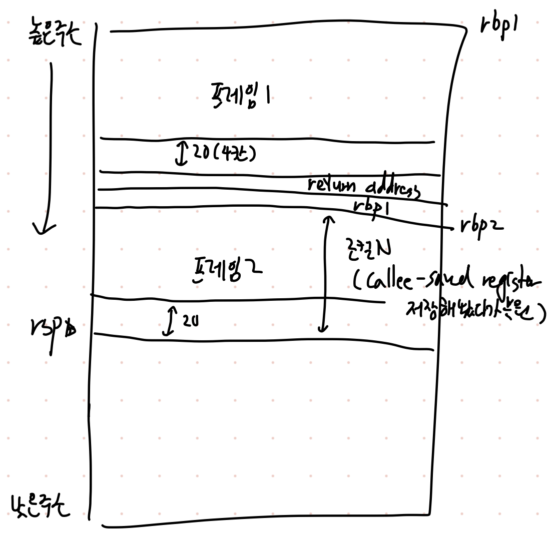
코드



스택



rax 반환값, 임시값

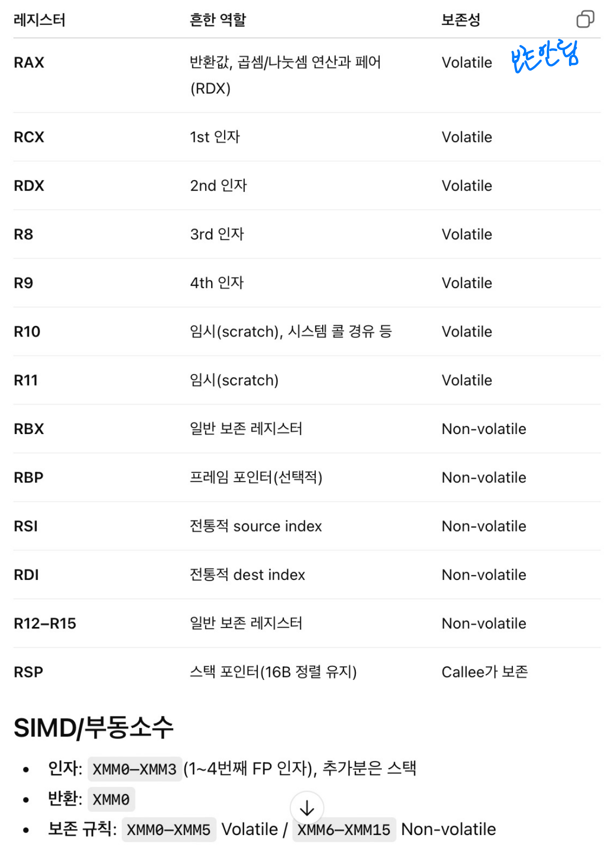
rcx
rdx
r8
r9 } 인자 4개

참고로 callee-saved 레지스터는
사용할 때만 스택에 넣어놨다가 복구

인자넣기

```
call XXXXX        # return address push가 포함된 명령어
가면 push rbp
mov rbp, rsp
sub rsp,N
  본문
add rsp, N or mov rsp,rbp
                  # rbp를 이전걸로 복원
pop rbp
                  # return address 얻어 원상 복구하는 명령어
ret
```

정수/포인터용



레지스터	흔한 역할	보존성
RAX	반환값, 곱셈/나눗셈 연산과 페어 (RDX)	Volatile 보존안됨
RCX	1st 인자	Volatile
RDX	2nd 인자	Volatile
R8	3rd 인자	Volatile
R9	4th 인자	Volatile
R10	임시(scratch), 시스템 콜 경우 등	Volatile
R11	임시(scratch)	Volatile
RBX	일반 보존 레지스터	Non-volatile
RBP	프레임 포인터(선택적)	Non-volatile
RSI	전통적 source index	Non-volatile
RDI	전통적 dest index	Non-volatile
R12–R15	일반 보존 레지스터	Non-volatile
RSP	스택 포인터(16B 정렬 유지)	Callee가 보존

SIMD/부동소수

- 인자: `XMM0–XMM3` (1~4번째 FP 인자), 추가분은 스택
- 반환: `XMM0`
- 보존 규칙: `XMM0–XMM5` Volatile / `XMM6–XMM15` Non-volatile

보존된다 = 설령 때려쓴다해도 스택에 푸시해놓고 리턴 전에 [illegible] 복원
보존 안된다 : 값을 지키고 싶으면 네가 미리 저장해라

ret이란
rsp의 8바이트를 rip으로 넣어 리턴하는 명령어

cmp a b

je	jump equal	=	zf=1
jne	jump not equal	≠	zf=0
ja	jump above	>	cf=0 이고 zf=0
jae		≥	cf=0
jb	jump below	<	cf=1
jbe		≤	cf=1 이거나 zf=1

ㄴ zf=1 이어도
ㄴ 반대
ㄴ zf=1 이어도

```
rax(64)
 └ eax(32)
    └ ax(16)
       ├ ah 8..15
       └ al 0..7
```

```
Overwatch.exe+20DDBF0 - 48 8B 4B 08           - mov rcx,[rbx+08]
Overwatch.exe+20DDBF4 - 44 0FB7 4B FC         - movzx r9d,word ptr [rbx-04]
Overwatch.exe+20DDBF9 - 44 8B 40 08           - mov r8d,[rax+08]
Overwatch.exe+20DDBFD - 48 8B 10              - mov rdx,[rax]
Overwatch.exe+20DDC00 - 48 89 4C 24 78        - mov [rsp+78],rcx
Overwatch.exe+20DDC05 - 48 8B 4B 10           - mov rcx,[rbx+10]
Overwatch.exe+20DDC09 - 48 89 4D 80           - mov [rbp-80],rcx
Overwatch.exe+20DDC0D - 48 8D 4C 24 78        - lea rcx,[rsp+78]
Overwatch.exe+20DDC12 - 48 89 4C 24 20        - mov [rsp+20],rcx
Overwatch.exe+20DDC17 - 48 8D 4C 24 30        - lea rcx,[rsp+30]
Overwatch.exe+20DDC1C - E8 EF0BFEFF           - call Overwatch.exe+20BE810
Overwatch.exe+20DDC21 - 48 8D 4C 24 50        - lea rcx,[rsp+50]
Overwatch.exe+20DDC26 - E8 9548FCFF           - call Overwatch.exe+20A24C0
Overwatch.exe+20DDC2B - 48 85 C0              - test rax,rax
Overwatch.exe+20DDC2E - 75 C0                 - jne Overwatch.exe+20DDBF0
Overwatch.exe+20DDC30 - EB 73                 - jmp Overwatch.exe+20DDCA5
Overwatch.exe+2014F70 - 48 89 5C 24 08        - mov [rsp+08],rbx
Overwatch.exe+2014F75 - 48 89 6C 24 18        - mov [rsp+18],rbp
Overwatch.exe+2014F7A - 48 89 74 24 20        - mov [rsp+20],rsi
Overwatch.exe+2014F7F - 57                    - push rdi
Overwatch.exe+2014F80 - 48 83 EC 20           - sub rsp,20 { 32 }
Overwatch.exe+2014F84 - 48 8B 02              - mov rax,[rdx]
Overwatch.exe+2014F87 - 48 8B FA              - mov rdi,rdx
Overwatch.exe+2014F8A - 48 8B E9              - mov rbp,rcx
Overwatch.exe+2014F8D - 49 8B D8              - mov rbx,r8
Overwatch.exe+2014F90 - 48 8D 4C 24 38        - lea rcx,[rsp+38]
Overwatch.exe+2014F95 - 8B 10                 - mov edx,[rax]
Overwatch.exe+2014F97 - C1 EA 08              - shr edx,08 { 8 }
Overwatch.exe+2014F9A - E8 915FAC00           - call Overwatch.exe+2ADAF30
```

저장

백업

rcx: 매니저
rdx: 설명자
r8: 엔티티

-5 +7 +2칸 주소를 rcx에 복사

8비트 오른쪽 shift

4칸	rsi
3칸	rbp
2칸	해당값
1칸	rbx
시작점 0칸	rdi
1칸	
2칸	
3칸	
4칸	
5칸	

```
Overwatch.exe+2ADAF30 - 89 11                 - mov [rcx],edx
Overwatch.exe+2ADAF32 - 48 8B C1              - mov rax,rcx
Overwatch.exe+2ADAF35 - C3                    - ret
```

+2칸에 해당값 넣어주기
+2칸 주소를 rax에 백업

```
Overwatch.exe+2014F9F - 8B 08                 - mov ecx,[rax]
Overwatch.exe+2014FA1 - E8 2A6E0600           - call Overwatch.exe+207BDD0
```

+2칸에서 꺼내기

```
Overwatch.exe+207BDD0 - 89 4C 24 08           - mov [rsp+08],ecx
Overwatch.exe+207BDD4 - 48 83 EC 28           - sub rsp,28 { 40 }
Overwatch.exe+207BDD8 - 48 8D 4C 24 30        - lea rcx,[rsp+30]
Overwatch.exe+207BDDD - E8 5EF1A500           - call Overwatch.exe+2ADAF40
```

-6 -5칸에 해당값 넣기
rsp= -11칸
-5칸 주소 rcx에

```
Overwatch.exe+2ADAF40 - 8B 01                 - mov eax,[rcx]
Overwatch.exe+2ADAF42 - C1 E8 10              - shr eax,10 { 16 }
Overwatch.exe+2ADAF45 - 0FBD C8               - bsr ecx,eax
Overwatch.exe+2ADAF48 - B8 00000000           - mov eax,00000000 { 0 }
Overwatch.exe+2ADAF4D - 0F95 C0               - setne al
Overwatch.exe+2ADAF50 - 0FAF C8               - imul ecx,eax
Overwatch.exe+2ADAF53 - D1 E9                 - shr ecx,1
Overwatch.exe+2ADAF55 - 03 C8                 - add ecx,eax
Overwatch.exe+2ADAF57 - 0F95 C0               - setne al
Overwatch.exe+2ADAF5A - C3                    - ret
```

꺼내면 해당값

16비트 오른쪽 shift

최상위비트 어딘지

eax=0 → zf=1 → al=0
al=0 이면 ecx=0

eax≠0 → zf=0 → al=1
al=1 이면 ecx=그대로

1비트 밀기

(ecx/2)+1

add=0 → zf=1 → al=0

add≠0 → zf=0 → al=1

```
Overwatch.exe+207BDE2 - 84 C0                 - test al,al
Overwatch.exe+207BDE4 - 75 0A                 - jne Overwatch.exe+207BDF0
```

al=1 → and≠0 → zf=0 → jne 감

16비트 이하나 처리해야할 일 없으면 패스스루
↓

```
Overwatch.exe+207BDE6 - 0FB7 44 24 30         - movzx eax,word ptr [rsp+30]
Overwatch.exe+207BDEB - 48 83 C4 28           - add rsp,28 { 40 }
Overwatch.exe+207BDEF - C3                    - ret
```

하위16비트 -11 +6

```
Overwatch.exe+2014FA6 - 0FB7 F0               - movzx esi,ax
Overwatch.exe+2014FA9 - 48 85 DB              - test rbx,rbx
Overwatch.exe+2014FAC - 74 32                 - je Overwatch.exe+2014FE0
```

esi에 백업

rbx가 존재하면 zf=0 → je 안감

rbx값 있어서 처리해야 할 일 없으면 패스스루
↓

```
Overwatch.exe+2014FAE - B8 FFFF0000           - mov eax,0000FFFF { 65535 }
Overwatch.exe+2014FB3 - 66 3B F0              - cmp si,ax
Overwatch.exe+2014FB6 - 74 28                 - je Overwatch.exe+2014FE0
```

원래값 vs FFFF

같으면 에러. 처리해야 할 일 없으면 패스스루
↓

```
Overwatch.exe+2014FB8 - 0FB7 CE               - movzx ecx,si
Overwatch.exe+2014FBB - E8 20D30500           - call Overwatch.exe+20722E0
```

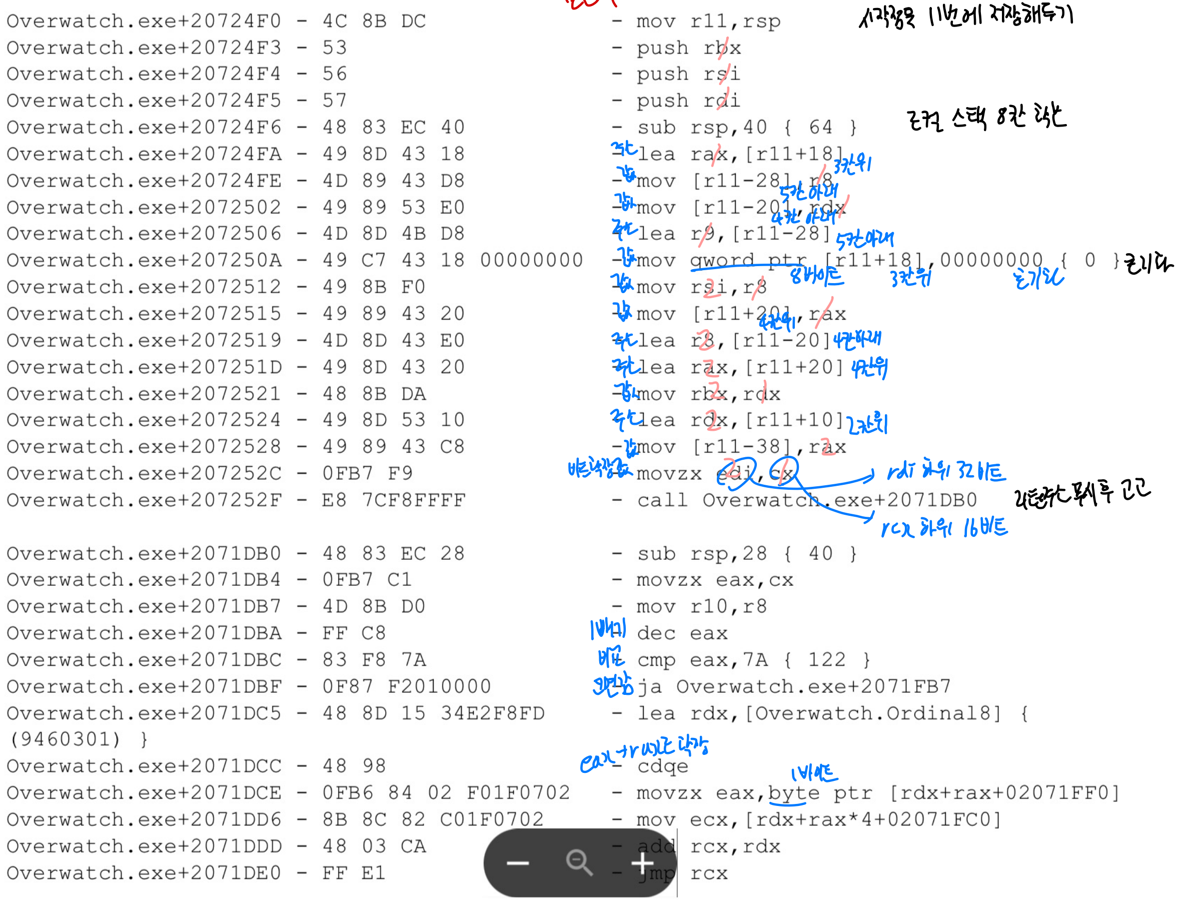
ecx에 넣고 호출

```
Overwatch.exe+20722E0 - 48 83 EC 28            - sub rsp,28 { 40 }
Overwatch.exe+20722E4 - 0FB7 C1                - movzx eax,cx   그 값
Overwatch.exe+20722E7 - FF C8                  - dec eax          -1
Overwatch.exe+20722E9 - 83 F8 7A               - cmp eax,7A { 122 }
Overwatch.exe+20722EC - 77 26                  - ja Overwatch.exe+2072314    122보다 크면 처리하기

Overwatch.exe+20722EE - 4C 8D 05 0BDDF8FD      - lea r8,[Overwatch.Ordinal8] { (9460301)
}
Overwatch.exe+20722F5 - 48 98                  - cdqe 확장
Overwatch.exe+20722F7 - 41 0FB6 84 00 3C230702  - movzx eax,byte ptr [r8+rax+0207233C]
Overwatch.exe+2072300 - 41 8B 94 80 34230702  - mov edx,[r8+rax*4+02072334]
Overwatch.exe+2072308 - 49 03 D0               - add rdx,r8
Overwatch.exe+207230B - FF E2                  - jmp rdx
```

→ 이전에 있은 값 rbx(1), rsi(1), rdi(1), rdx(1), r8(1), rcx(1)

rdx(1): 목적지 베이스 포인터 · r8(1): 소스 베이스 포인터 · rcx(1): 타입 코드



```
Overwatch.exe+20724F0 - 4C 8B DC                  - mov r11,rsp
Overwatch.exe+20724F3 - 53                        - push rbx
Overwatch.exe+20724F4 - 56                        - push rsi
Overwatch.exe+20724F5 - 57                        - push rdi
Overwatch.exe+20724F6 - 48 83 EC 40               - sub rsp,40 { 64 }
Overwatch.exe+20724FA - 49 8D 43 18               - lea rax,[r11+18]
Overwatch.exe+20724FE - 4D 89 43 D8               - mov [r11-28],r8
Overwatch.exe+2072502 - 49 89 53 E0               - mov [r11-20],rdx
Overwatch.exe+2072506 - 4D 8D 4B D8               - lea r9,[r11-28]
Overwatch.exe+207250A - 49 C7 43 18 00000000      - mov qword ptr [r11+18],00000000 { 0 }
Overwatch.exe+2072512 - 49 8B F0                  - mov rsi,r8
Overwatch.exe+2072515 - 49 89 43 20               - mov [r11+20],rax
Overwatch.exe+2072519 - 4D 8D 43 E0               - lea r8,[r11-20]
Overwatch.exe+207251D - 49 8D 43 20               - lea rax,[r11+20]
Overwatch.exe+2072521 - 48 8B DA                  - mov rbx,rdx
Overwatch.exe+2072524 - 49 8D 53 10               - lea rdx,[r11+10]
Overwatch.exe+2072528 - 49 89 43 C8               - mov [r11-38],rax
Overwatch.exe+207252C - 0FB7 F9                   - movzx edi,cx
Overwatch.exe+207252F - E8 7CF8FFFF               - call Overwatch.exe+2071DB0

Overwatch.exe+2071DB0 - 48 83 EC 28               - sub rsp,28 { 40 }
Overwatch.exe+2071DB4 - 0FB7 C1                   - movzx eax,cx
Overwatch.exe+2071DB7 - 4D 8B D0                  - mov r10,r8
Overwatch.exe+2071DBA - FF C8                     - dec eax
Overwatch.exe+2071DBC - 83 F8 7A                  - cmp eax,7A { 122 }
Overwatch.exe+2071DBF - 0F87 F2010000             - ja Overwatch.exe+2071FB7
Overwatch.exe+2071DC5 - 48 8D 15 34E2F8FD         - lea rdx,[Overwatch.Ordinal8] {
(9460301) }
Overwatch.exe+2071DCC - 48 98                     - cdqe
Overwatch.exe+2071DCE - 0FB6 84 02 F01F0702       - movzx eax,byte ptr [rdx+rax+02071FF0]
Overwatch.exe+2071DD6 - 8B 8C 82 C01F0702         - mov ecx,[rdx+rax*4+02071FC0]
Overwatch.exe+2071DDD - 48 03 CA                  - add rcx,rdx
Overwatch.exe+2071DE0 - FF E1                     - jmp rcx
```

mov r11,rsp: 시작점을 11번에 저장해두기

sub rsp,40: 로컬 스택 8칸 확보

lea rax,[r11+18]: 주소, 3칸위

mov [r11-28],r8: 값, 5칸아래

mov [r11-20],rdx: 값, 4칸 아래

lea r9,[r11-28]: 주소, 5칸아래

mov qword ptr [r11+18],00000000 { 0 }: 값, 8바이트, 3칸위, 초기화, 비운다

mov rsi,r8: 값

mov [r11+20],rax: 값, 4칸위

lea r8,[r11-20]: 주소, 4칸아래

lea rax,[r11+20]: 주소, 4칸위

mov rbx,rdx: 값

lea rdx,[r11+10]: 주소, 2칸위

mov [r11-38],rax: 값

movzx edi,cx: 비트확장값, rdi 하위 32비트, rcx 하위 16비트

call Overwatch.exe+2071DB0: 리턴주소 푸시 후 ㄱㄱ

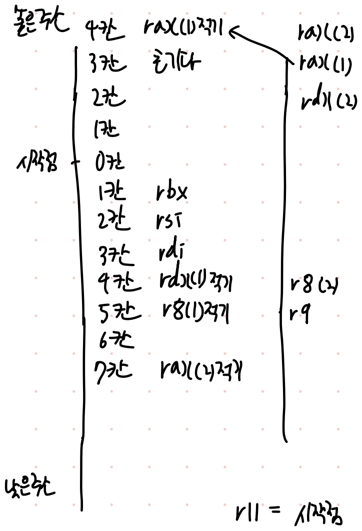
dec eax: 1빼기

cmp eax,7A: 비교

ja Overwatch.exe+2071FB7: 크면감

cdqe: eax가 rax로 확장

movzx eax,byte ptr [rdx+rax+02071FF0]: 1바이트



r11 = 시작점

rsi(2) = r8(1)로 업뎃

rbx(2) = rdx(1)로 업뎃

rax 3칸위주소 4칸위에적기, 4칸위주소 7칸 아래적기 업데이트 → 확장 찾은값

rbx 원가공 스택에푸시, 업데이트

rcx 원가공

rdx 원가공 4칸 아래적용, 2칸위주소 Ordinal8로 업뎃

r8 원가공 5칸아래적용, 4칸아래주소 업데이트

r9 5칸아래주소

r10

r11 시작점

rsi 원가공 스택에푸시 업데이트

rdi 원가공 스택에푸시 업데이트

rsp

찾은값

처음에 넣던 rdx 값, 목적지 포인터

rsp는 최초에서 push세번, sub 40, call한번, sub 28
즉 17칸만큼 내려갔다. 다시 10칸 올리면 7칸이다.

해당 위치에 있는건 +4칸 주소.

```
Overwatch.exe+2071F89 - 49 8B 08          - mov rcx,[r8]
Overwatch.exe+2071F8C - 48 8B 44 24 50    - mov rax,[rsp+50]
Overwatch.exe+2071F91 - 48 83 C1 03       - add rcx,03 { 3 }
Overwatch.exe+2071F95 - 48 83 E1 FC       - and rcx,-04 { 252 }
Overwatch.exe+2071F99 - 48 8B 10          - mov rdx,[rax]
Overwatch.exe+2071F9C - 49 8B 01          - mov rax,[r9]
Overwatch.exe+2071F9F - F2 0F10 00        - movsd xmm0,[rax]
Overwatch.exe+2071FA3 - F2 0F11 01        - movsd [rcx],xmm0
Overwatch.exe+2071FA7 - 8B 40 08          - mov eax,[rax+08]
Overwatch.exe+2071FAA - 89 41 08          - mov [rcx+08],eax
Overwatch.exe+2071FAD - B0 01             - mov al,01 { 1 }
Overwatch.exe+2071FAF - 48 89 0A          - mov [rdx],rcx
Overwatch.exe+2071FB2 - 48 83 C4 28       - add rsp,28 { 40 }
Overwatch.exe+2071FB6 - C3                - ret
```

4의 배수로 올림

한번 더 꺼내면 +3칸 주소

해당 위치에 있는건 최초의 r8, 소스포인터

거기서 꺼낸 소스값

소스값을, 목적지 포인터에 저장

소스값 뒷부분도 읽고

저장

성공 플래그

스택 위치에 목적지 포인터 저장

🟦 : 목적지

🟩 : 소스값

🟨 : 스택위치

```
Overwatch.exe+2071E71 - 49 8B 01          - mov rax,[r9]
Overwatch.exe+2071E74 - 49 8B 10          - mov rdx,[r8]
Overwatch.exe+2071E77 - 48 83 C2 03       - add rdx,03 { 3 }
Overwatch.exe+2071E7B - 48 83 E2 FC       - and rdx,-04 { 252 }
Overwatch.exe+2071E7F - 8B 08             - mov ecx,[rax]
Overwatch.exe+2071E81 - 48 8B 44 24 50    - mov rax,[rsp+50]
Overwatch.exe+2071E86 - 89 0A             - mov [rdx],ecx
Overwatch.exe+2071E88 - 48 8B 08          - mov rcx,[rax]
Overwatch.exe+2071E8B - B0 01             - mov al,01 { 1 }
Overwatch.exe+2071E8D - 48 89 11          - mov [rcx],rdx
Overwatch.exe+2071E90 - 48 83 C4 28       - add rsp,28 { 40 }
Overwatch.exe+2071E94 - C3                - ret
```

소스값을 목적지 포인터에 저장

스택 위치에 목적지 포인터저장 .

```
Overwatch.exe+2071EB9 - 49 8B 08          - mov rcx,[r8]
Overwatch.exe+2071EBC - 48 8B 44 24 50    - mov rax,[rsp+50]
Overwatch.exe+2071EC1 - 48 83 C1 0F       - add rcx,0F { 15 }
Overwatch.exe+2071EC5 - 48 83 E1 F0       - and rcx,-10 { 240 }
Overwatch.exe+2071EC9 - 48 8B 10          - mov rdx,[rax]
Overwatch.exe+2071ECC - 49 8B 01          - mov rax,[r9]
Overwatch.exe+2071ECF - 0F10 00           - movups xmm0,[rax]
Overwatch.exe+2071ED2 - 0F11 01           - movups [rcx],xmm0
Overwatch.exe+2071ED5 - 0F10 48 10        - movups xmm1,[rax+10]
Overwatch.exe+2071ED9 - 0F11 49 10        - movups [rcx+10],xmm1
Overwatch.exe+2071EDD - 0F10 40 20        - movups xmm0,[rax+20]
Overwatch.exe+2071EE1 - 0F11 41 20        - movups [rcx+20],xmm0
Overwatch.exe+2071EE5 - 0F10 48 30        - movups xmm1,[rax+30]
Overwatch.exe+2071EE9 - B0 01             - mov al,01 { 1 }
Overwatch.exe+2071EEB - 0F11 49 30        - movups [rcx+30],xmm1
Overwatch.exe+2071EEF - 48 89 0A          - mov [rdx],rcx
Overwatch.exe+2071EF2 - 48 83 C4 28       - add rsp,28 { 40 }
Overwatch.exe+2071EF6 - C3                - ret
```

16의 배수로 올림

128비트 xmm은 정렬없이
소스값을, 목적지 포인터에) 저장

```
Overwatch.exe+2072534 - 84 C0                 - test al,al
Overwatch.exe+2072536 - 75 4C                 - jne Overwatch.exe+2072584
Overwatch.exe+2072538 - 0FB7 CF               - movzx ecx,di
Overwatch.exe+207253B - E8 E063F9FF           - call Overwatch.exe+2008920
Overwatch.exe+2072540 - 48 8D 15 798BCB01     - lea rdx,[Overwatch.exe+3D2B0C0] {
(7FF6F09DB0C0) }
Overwatch.exe+2072547 - 48 8B C8              - mov rcx,rax
Overwatch.exe+207254A - 48 8B F8              - mov rdi,rax
Overwatch.exe+207254D - 4C 8B 00              - mov r8,[rax]
Overwatch.exe+2072550 - 41 FF 50 10           - call qword ptr [r8+10]
Overwatch.exe+2072554 - 84 C0                 - test al,al
Overwatch.exe+2072556 - 74 2C                 - je Overwatch.exe+2072584
Overwatch.exe+2072558 - 83 7F 08 02           - cmp dword ptr [rdi+08],02 { 2 }
Overwatch.exe+207255C - 75 16                 - jne Overwatch.exe+2072574
Overwatch.exe+207255E - 0FB7 0E               - movzx ecx,word ptr [rsi]
Overwatch.exe+2072561 - 48 8D 43 01           - lea rax,[rbx+01]
Overwatch.exe+2072565 - 48 83 E0 FE           - and rax,-02 { 254 }
Overwatch.exe+2072569 - 66 89 08              - mov [rax],cx
Overwatch.exe+207256C - 48 83 C4 40           - add rsp,40 { 64 }
Overwatch.exe+2072570 - 5F                    - pop rdi
Overwatch.exe+2072571 - 5E                    - pop rsi
Overwatch.exe+2072572 - 5B                    - pop rbx
Overwatch.exe+2072573 - C3                    - ret
```

and한 결과. 즉 0(오류플래그)이 있나 테스트
오류없으면 점프