

입력과 출력

NASM에서의 입력과 출력을 알아보기 위해, 입력 받은 값을 그대로 출력하는 프로그램을 작성해 보자.

- System Call : 운영 체제의 커널이 제공하는 서비스에 대해, 응용 프로그램의 요청에 따라 커널에 접근하기 위한 인터페이스. 커널과 응용 프로그램은 CPU의 권한 수준이나, 하드웨어 접근 능력이 다름. 따라서 **응용 프로그램이 커널의 서비스를 사용하고 싶을 때 System Call을 사용함.**
 - Linux x86-64 System Call : 64bit 리눅스에서는 **RAX의 값에 따라 System Call의 종류가 달라지고, 그 실행을 위한 매개변수로 RDI, RSI, RDX, R10, R9, R8 등이 들어가게 됨.**
- XOR \oplus (eXclusive OR) : 상호 배제적인 OR로, INPUT A와 B가 서로 다를 경우에만 OUTPUT = 1이 됨. 이러한 특성에 의해, **A XOR A = 0**이 항상 성립함.
- MOV (MOV *destination source*) : source에 있는 데이터가 destination으로 **복사** 됨. 이 때, 두 연산자가 모두 메모리이면 안 되고, 크기 또한 항상 같아야 한다는 특징이 있음.

```

1 echo.asm
file: ehco.asm
1 ;입력받은 값을 그대로 출력하는 프로그램
2
3 section .text
4 global _start
5
6 _start:
7  xor rax, rax          ;자기 자신과의 xor연산 -> 0
8  mov rbx, rax          ;rbx 초기화
9  mov rcx, rax          ;rcx 초기화
10 mov rdx, rax          ;rdx 초기화
11
12 sub rsp, 64           ;rsp - 64를 통해 문자열이 담길 수 있는 공간 64를 확보
13 mov rdi, 0            ;rdi = 파일 서술자 (file descriptor), 무언가를 읽는 경우에는 0, 쓰는 경우에는 1
14 mov rsi, rsp          ;rsi = 문자열 포인터, 즉 문자열의 시작 주소를 가리킴
15 mov rdx, 63           ;rdx = 문자열을 얼마나 읽어올지 정함
16
17 syscall               ;rax = 0, sys_read 실행
18
19 mov rax, 1
20 mov rdi, 1            ;무언가를 쓰는 경우이므로 rdi = 1
21 mov rsi, rsp          ;위 syscall로 받아온 문자열 시작 주소 rsp
22 mov rdx, 63           ;rsi부터 63만큼 출력하기로 정함
23
24 syscall               ;rax = 1, sys_write 실행
25
26 mov rax, 60
27
28 syscall               ;rax = 60, sys_exit 실행. 즉, 프로그램 종료

```

line은 line1을 제외하여, 각 줄의 옆에 있는 숫자로 표기함

```

;line 7 - 10 : rax, rbx, rcx, rdx 초기화
xor rax, rax
mov rbx, rax
mov rcx, rax
mov rdx, rax

```

A **xor A = 0**이 항상 성립한다는 특징을 이용해 xor rax, rax를 통해 rax = 0을 저장한 뒤, mov를 통하여 rbx, rcx, rdx 모두 0으로 **초기화**했다.

```

;line 12 - 17 : 문자열 입력받을 공간 확보 및 sys_read를 위한 레지스터 설정
sub rsp, 64
mov rdi, 0
mov rsi, rsp
mov rdx, 63

syscall

```

rsp(스택 프레임의 끝 지점 주소) 에서 64를 빼줌으로써, **문자열이 들어갈 수 있는 공간 64를 확보**하였다.



현재 `rax = 0` 이므로 `syscall`에서는 `sys_read`가 실행되고, 이 때 `rdi`는 파일 서술자를, `rsi`는 입력받을 문자열의 포인터, `rdx`는 입력받을 문자열의 크기를 정하므로 이에 맞춰 값을 설정하였다. 또한 문자열의 끝에는 NULL문자가 들어가야 하므로 63 글자만 입력받을 수 있게 하여 Overflow를 막았다.

```
;line 19 - 24 : 문자열 출력 (sys_write)을 위한 레지스터 설정
mov rax, 1
mov rdi, 1
mov rsi, rsp
mov rdx, 63

syscall
```

`rax = 1`이므로 `syscall`에서는 `sys_write` 실행, `rdi = 1`로 파일 서술자를 지정해 주고 나머지는 `sys_read`와 동일하다.

```
;line 26 - 28 : sys_exit으로 프로그램 종료
mov rax, 60

syscall
```

`rax = 60`이므로 `sys_exit`을 실행하여 프로그램을 종료한다.

```
~/Desktop/Assembly  
> ./echo  
Isaac Toast  
Isaac Toast  
  
~/Desktop/Assembly 6s  
> ./echo  
AAAAAAAAAABBBBBBBBCCCCCCCCDDDDDDDDDEEEEEEEEEFFFFFFFFFFGGGG  
AAAAAAAAAABBBBBBBBCCCCCCCCDDDDDDDDDEEEEEEEEEFFFFFFFFFFGGG  
~/Desktop/Assembly 32s  
> G  
zsh: command not found: G  
  
~/Desktop/Assembly  
> █
```

실행 결과

63개 이상의 문자는 자동으로 잘리는 모습을 볼 수 있다.