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 ⊕ (eXclusive OR) : 상호 배제적인 OR로, INPUT A와 B가 서로 다를 경우에만 OUTPUT = 1이 됨. 이러한 특성에 의해, A XOR A = 0이 항상 성립함.
- MOV (MOV *destination source*) : source에 있는 데이터가 destination으로 복사됨. 이 때, 두 연산자가 모두 메모리이면 안 되고, 크기 또한 항상 같아야 한다는 특징이 있음.

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

```
;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 확보하였다.

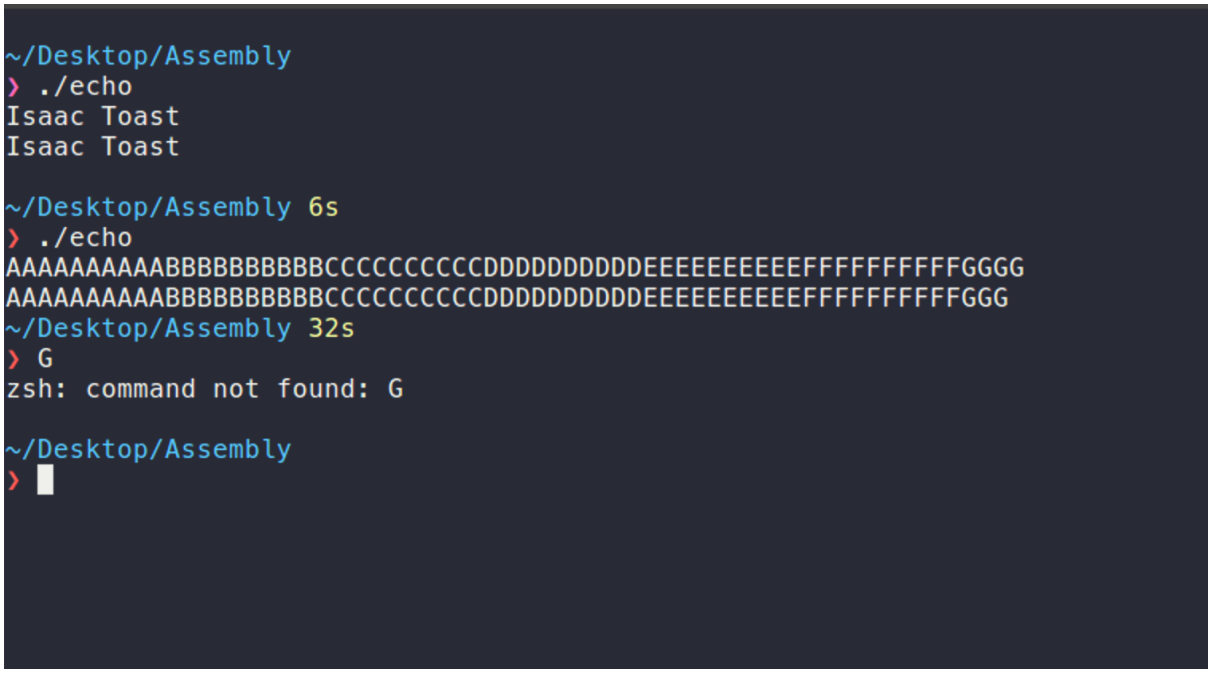
현재 rax = 0 이므로 syscall에서는 sys_read가 실행되고, 이 때 rdi는 파일 서술자를, rsi는 입력받을 문자열의 포인터, rdx는 입력받을 문자열의 크기를 정하므로 이에 맞춰 값을 설정하였다. 또한 문자열의 끝에는 NULL문자가 들어가야 하므로 63 글자만 입력받을 수 있게 하여 Overflow를 막았다.

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

syscall
```

rax = 1이므로 syscall에서는 sys_write 실행, rdi = 1로 파일 서술자를 지정해 주고 나머지는 sys_read와 동일하다.

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

syscall
```

rax = 60이므로 sys_exit을 실행하여 프로그램을 종료한다.



실행 결과

63개 이상의 문자는 자동으로 잘리는 모습을 볼 수 있다.