

# 어셈블리어 공부

권도윤

# 목차

- 어셈블리어란?
- Intel과 AT&T
- 레지스터란?
- 자주 사용하는 명령어

# 어셈블리어란?

- 기계어와 일대일 대응이 되는 컴퓨터 프로그래밍의 저급 언어
- **장점** : 컴파일을 해도 간단한 명령으로 실행돼서 실행 속도가 빠르다.
- **단점** : 배우기가 어렵고 유지보수가 힘들다

# Intel과 AT&T

- 어셈블리어에는 Intel문법과 AT&T문법이 존재
- ex) Intel의 문법
- EBX의 값을 EAX에 더한다.

Opcode	Operand1	Operand2
ADD	EAX	EBX

# Intel과 AT&T

- 숫자 표기 방식 :

Intel - 1, 2, 3, 4...

AT&T - \$1, \$2, \$3

- 레지스터 표기 방식 :

Intel - EAX, EBX, EBP

AT&T - %EAX, %EBX, %EBP

# 레지스터란?

- CPU의 요청을 처리하는 데이터의 임시공간
- 32비트 : 레지스터의 처음이 E로 시작
- 64비트 : 레지스터의 처음이 R로 시작

# 범용 레지스터

- RAX : 사칙연산 등 산술 연산에 자동으로 사용되며, 함수의 반환 값을 처리할 때도 사용.
- RBX : 간접 번지 지정에 사용. 산수, 변수를 저장.
- RCX : 반복(Loop)에서 반복 Count 역할을 수행.
- RDX : RAX를 보조하는 역할을 함. 예를 들어 나누기를 진행할 경우 몫은 RAX에 나머지는 EDX에 저장.

# 인덱스 레지스터

- RSI : 복사나 비교를 할 경우 출발지 주소를 저장하는 레지스터.
- RDI : 복사나 비교를 할 경우 목적지 주소를 저장하는 레지스터.



# 포인터 레지스터

- **RIP** : 다음에 실행할 명령어의 주소를 가지고 있는 레지스터. 현재 실행하고 있는 명령어가 종료되면 이 레지스터에 있는 명령어를 실행.
- **RSP** : **Stack Pointer**의 가장 최근에 저장된 공간의 주소를 저장하는 레지스터.
- **RBP** : **Stack Pointer**의 기준점(바닥 부분)을 저장하는 레지스터.

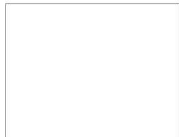
# 자주 사용하는 명령어

명령어	예제	설명	분류
push	push eax	eax의 값을 스택에 저장	스택 조작
pop	pop eax	스택 가장 상위에 있는 값을 꺼내서 eax에 저장	스택 조작
mov	mov eax, ebx	메모리나 레지스터의 값을 옮길때 사용	데이터 이동
inc	inc eax	eax의 값을 1증가시킨다 (++)	데이터 조작
dec	dec eax	eax의 값을 1감소시킨다 (--)	데이터 조작
add	add eax, ebx	레지스터나 메모리의 값을 덧셈할때 쓰인다.	논리, 연산
sub	sub eax, ebx	레지스터나 메모리의 값을 뺄셈할때 쓰인다.	논리, 연산
call	call proc	프로시저를 호출한다.	프로시저
ret	ret	호출했던 바로 다음 지점으로 이동	프로시저
cmp	cmp eax, ebx	레지스터와 레지스터의 값을 비교	비교
jmp	jmp proc	특정한 곳으로 분기	분기
int	int \$0x80	OS에 할당된 인터럽트 영역을 system call	인터럽트
nop	nop	아무 동작도 하지 않는다. (No Operation)	

# 간단실습-SASM

```
1  %include "io64.inc"
2  section .bss ;크기와 변수의 개수 지정
3
4  section .data ;크기와 초기값이 설정된 변수를 선언
5      a db 10
6  section .text ;코드작성
7  global CMAIN
8  CMAIN:
9      mov eax, [a] ;대괄호는 메모리 주소를 가리킬 때
10     PRINT_DEC 1, eax
11     ret
```

Input



Output

10