어셈블리 기초

https://youtu.be/Wpasc5KdnMk

어셈블리어란?

- Assembly
  - 프로그래밍 언어 중 하나.
  - 컴퓨터와 대화를 하기 위해서는 1과 0으로 대화를 나눠야 한다.(기계어)
    - 사람이 1과 0으로 대화하기 힘듬.
  - 이를 보완하기 위해 나온 언어가 어셈블리어 이다.
  - 기계어에서 한 단계 위의 언어이며 기계어와 함께 단 둘뿐인 Low level 언어.
    - High level 과 Low level은 컴퓨터가 이해하기 쉬운가로 구분한다.
    - High level : C, JAVA, Python … | Low level : 기계어, 어셈블리어

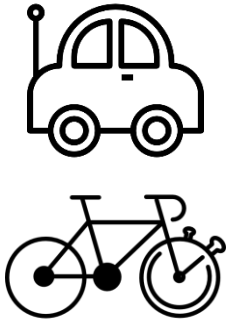
어셈블리어의 장점과 단점

- 장점
  - 작고 가볍다. -> 빠르다.
  - 가독성이 좋다. (ADD, SUB)
  - 성능이 좋다.

- 단점
  - 마스터하기 힘들다.
    - 통일된 규격이 없고, 문법이 아키텍처에 따라서 다르다)
    - 그래서 더 느릴 수도 있다.

High level language 와 Low level language

- High level language 와 Low level language

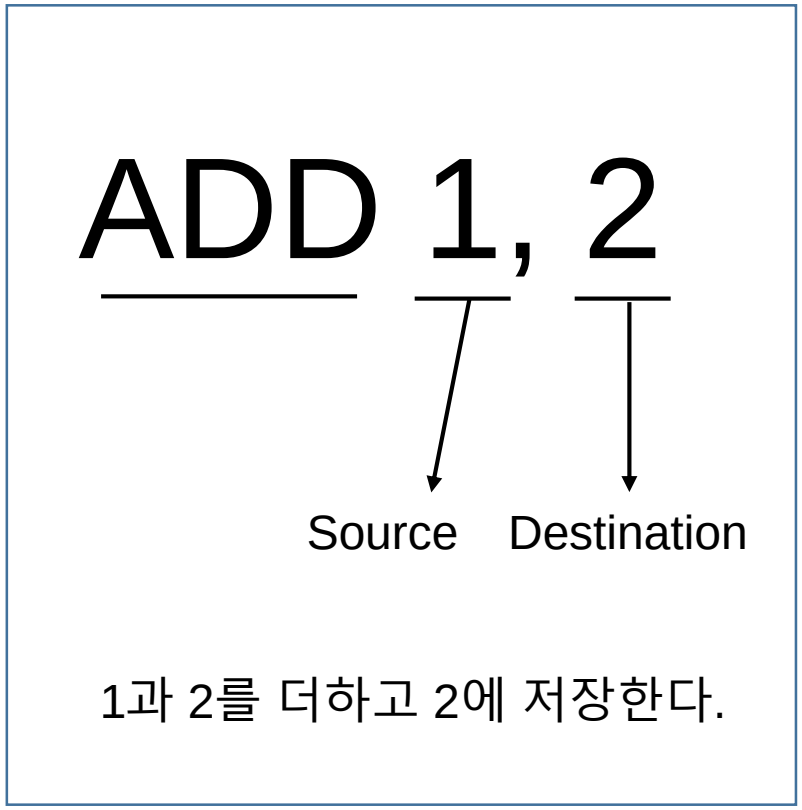


서울 → 강릉

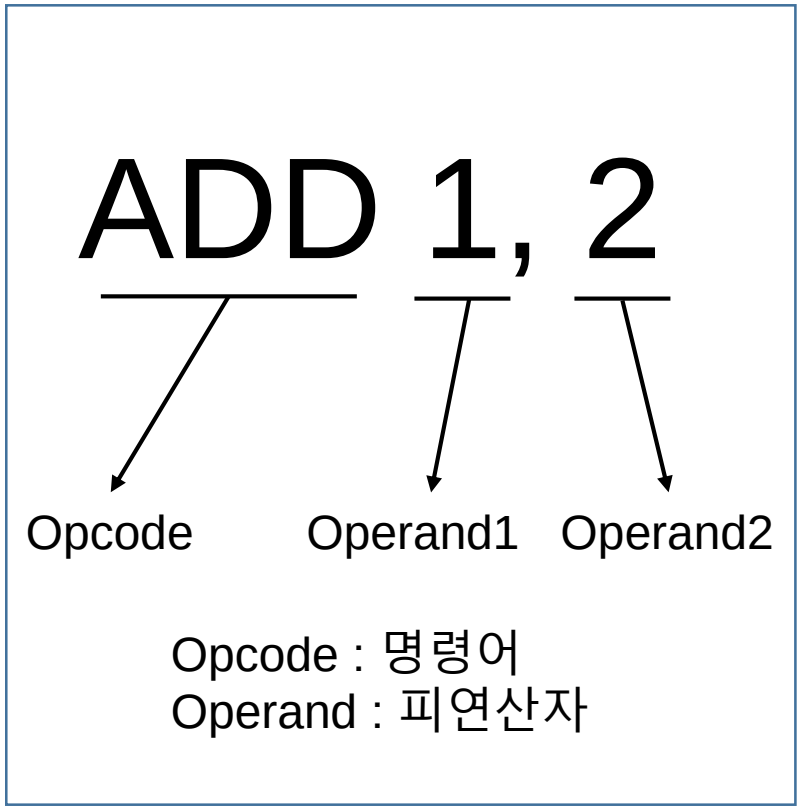
1. 자전거를 탄다.
2. 페달을 돌린다.
3. 몇번국도..
4. …..

강릉으로 간다.

AT&T와 Intel의 차이



AT&T



Intel

AT&T와 Intel의 차이

숫자
$1, $2, $3, $4, $5, $6, $7 , $8 , $9 , $0

Register(레지스터)
%rax, %rbx, %rcx, %rdx

메모리주소 참조(eax 레지스터 메모리 주소)
(EAX)

Offset(기준이 되는 주소에서 얼만큼 떨어져 있는지 표기하는 상대 주소)
4(EAX)

AT&T

숫자 표기
1, 2, 3, 4, 5, 6, 7, 8, 9, 0

Register(레지스터)
rax, rbx, rcx, rdx

메모리주소 참조(eax 레지스터 메모리 주소)
[EAX]

Offset(기준이 되는 주소에서 얼만큼 떨어져 있는지 표기하는 상대 주소)
[EAX+4]

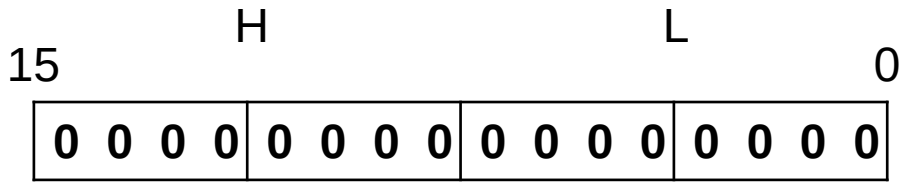
Intel

Register (레지스터)

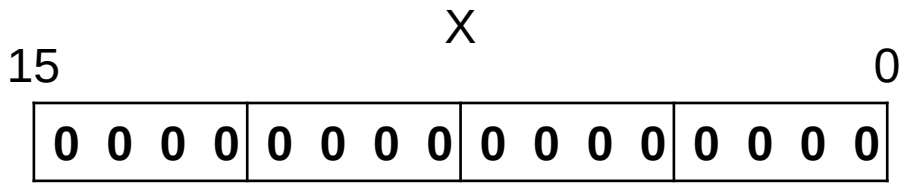
- Register : CPU에서 사용하는 변수 같은 것. 변수와 개념은 다르지만 비슷한 역할을 한다.
  - EAX (Extended Accumulator Register) – 가장 많이 사용, + - 같은 연산에 사용 return 값이 저장됨
  - EDX (Extended Data Register) – + - 와 같은 연산에 사용, return 값이 저장 안됨.
  - ECX (Extended Counter Register) – 카운터하는 레지스터
    - for(int i=0; i<10; i++) 에서 i와 같은 역할.
  - EBX (Extended Base Register) – eax, edx, ecx 가 부족할 때 사용되는 여분의 레지스터
  - ESI (Extended Source Index) – 데이터를 복사할 때 복사할 데이터의 주소 저장
  - EDI (Extended Destination Index) – 데이터를 복사할 때 복사할 곳의 주소 저장
  - ESP (Extended Stack Pointer) – 스텍프레임의 끝 지점 주소를 저장
  - EBP (Extended Base Pointer) – 스텍프렘임의 시작 지점 주소를 저장

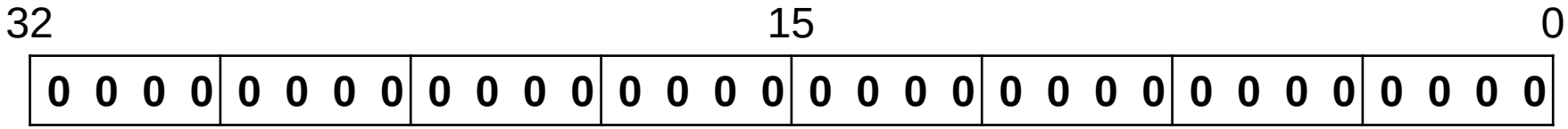
Register (레지스터)

AH, BH, CH, DH // AL, BL, CL, DL



AX, BX, CX, DX





EAX, EBX, ECX, EDX

Opcode

- PUSH – 스택에 값을 넣는다.
- POP – 스택의 값을 가져온다.
- MOV – 단순하게 값을 넣는 명령어 (MOV EAX, ECX)
- LEA - MOV와 같은 역할이지만 MOV는 값을 넣고, LEA는 주소를 넣는다.
- ADD, SUB – 더하기와 빼기
- INC, DEC – increase, decrease c언어에서 ++, -- 같은 역할
- CMP – operand를 비교하는 명령어
- CALL – 함수를 호출하는 명령어
- RET - CALL로 호출된 함수를 종료하고 CALL다음의 명령줄로 이동하는 명령어
- NOP – 아무것도 하지 않는 명령어

Q & A