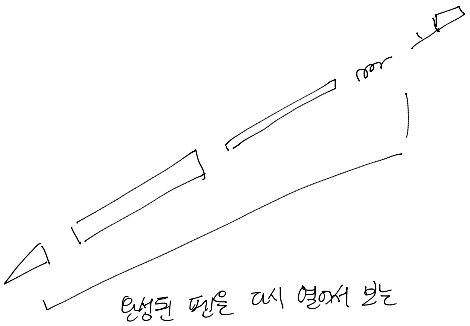
1번째 수업

2019년 12월 14일 토요일 오후 3:02

리버싱?



완성된 파일을 다시 열어서 보는

→ 완성된 코드를 다시 뜯어보는.

컴퓨터가 동작하기위해? ← 프로그램이 동작하기때문에

컴퓨터?

HardWare

- CPU
- Memory
- input device
- Output device

Software

- System Software
- 내가짜

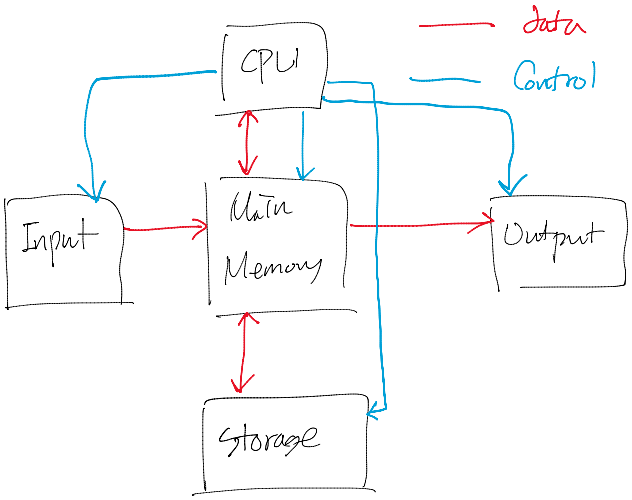
이 조합되어 있는 = 컴퓨터

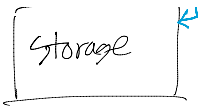
CPU가 Code가 실행되는 기준이 된다.

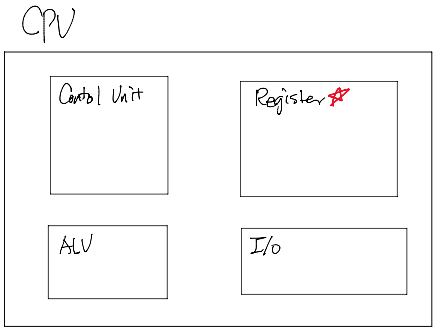
* IA-32 (Intel Architecture 32bit)

↳ 일반적 PC기준.

CPU : 연산, 제어, 저장

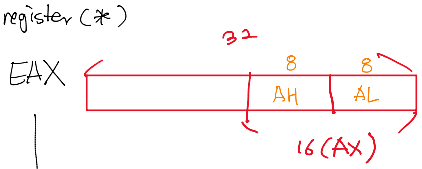






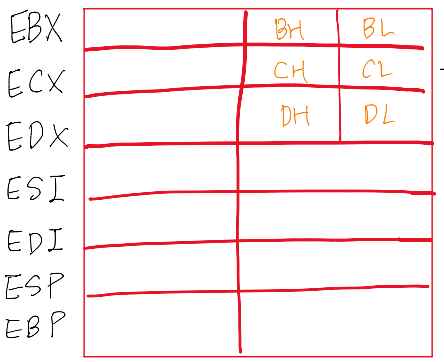
* Register : 32bit (4byte)

1. 범용 register (*)



→ 주로 산술연산 용도로 사용
(프로그래밍) 함수의 반환값 저장

* EBX ? EBP 도 가능하지만 주로 EAX



→ 주로 간접주소 참조용
→ 카운터의 반복문의 반복횟수를 저장 (특별)
→ 다른 Register들의 보조. ex) Double형 자료형이 8byte로 EAX+EDX 에서 저장
→ 대용량 복사시 사용
→ Stack Memory space의 top 값을 저장
→ 〃 Bottom 값을 저장

2. 명령 register (*)

EIP → 다음에 실행할 Code의 주소를 저장.

3. 세그먼트 register

4. flag register....

기계어 : Assembly = 1 : 1

수의 표현 (16진수) ☆

2진수
8진수
10진수
16진수 (*) => 1byte 공간을 효과적으로 표현하기 위해

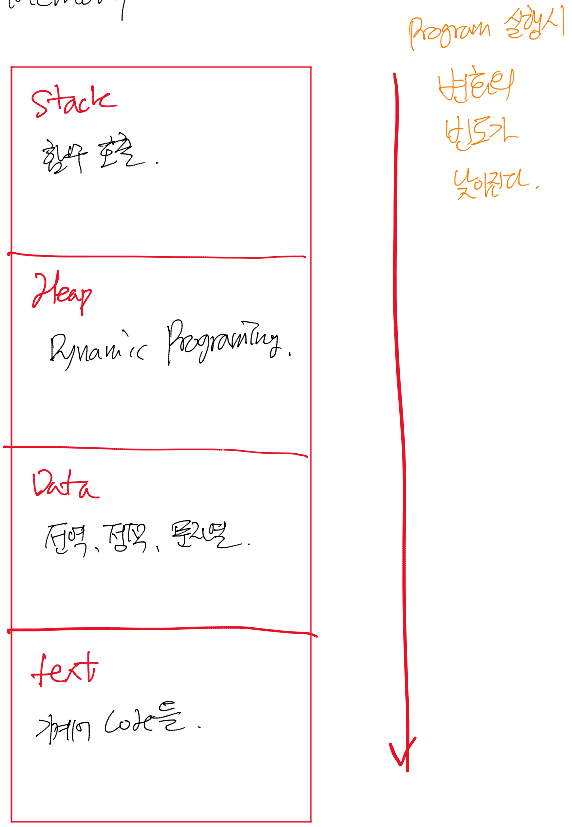
데이터 표현 → bit
데이터 저장 → Byte

영문자(1byte)는 ASCII 상 7bit로 표현된다
00 ~ 127 <= 1byte로 사용가능.

0000 0000 = 0 = 0x00
0 0
1111 1111 = 255 = 0xFF
15
F F

↳ 16진수 2개 = 1byte

Memory



Byte Ordering

- Big Endian
- Little Endian

2byte 이상 표현할때 뒤쪽에서 저장 <= Little endian
(단, 문자열은 두번째 X, Network 사용시 X)

0x 12 34 56 78

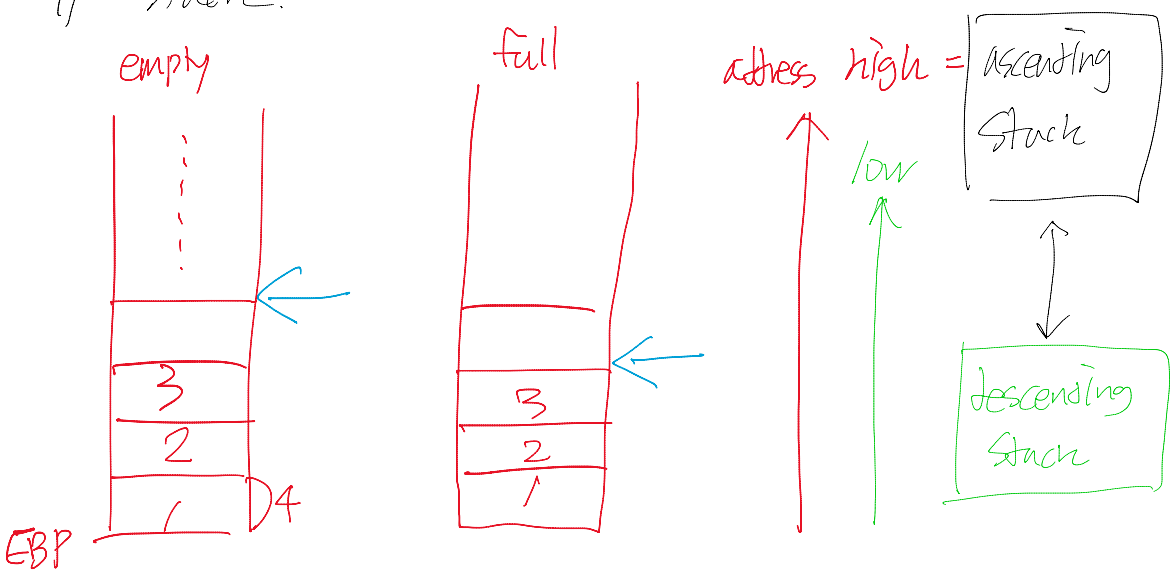
↓

78	56	34	12

char형으로 %x 출력시 => 78

short형으로 %x 출력시 => 56 78

Stack.



총 4종류

{ full / empty } + { ascending / descending } Stack

* intel은 full descending Stack.

0000 0000 (2) = 0 (10)

1111 1111 (2) = 255, -1 (10)

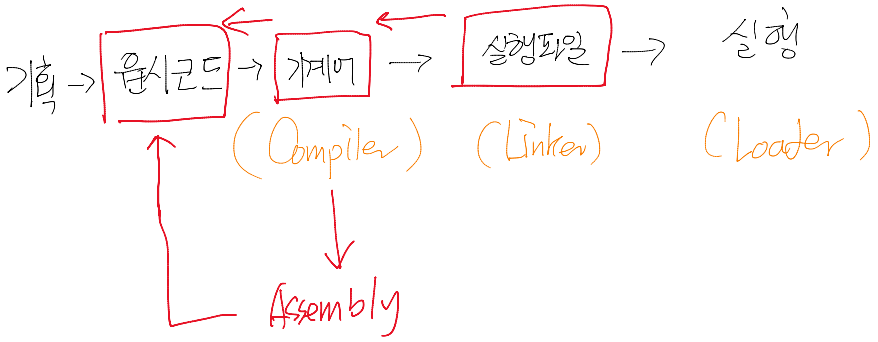
dddd dddd = Unsigned

데이터 표현

Sddd dddd = Signed

↑
부호 비트

Assembly



Assembly language

- 기계어 1 : 1 대응관계에 있는 언어.

OPCODE	OPERAND
(연산자)	(피연산자)

BYTE : 1 Byte
WORD : 2 Byte
DWORD : 4 Byte
QWORD : 8 Byte

n : 상수
[n] : n은 주소, []는 주소를 찾아감.

1. 데이터 이동

Push Pop mov movZX movSX movs lea

2. 연산

산술 : add sub mul imul div idiv inc dec

논리 : not and or xor neg

비트이동 : shl shr sal sar rol ror

비교 : test cmp

3. 분기

Jmp rep call ret

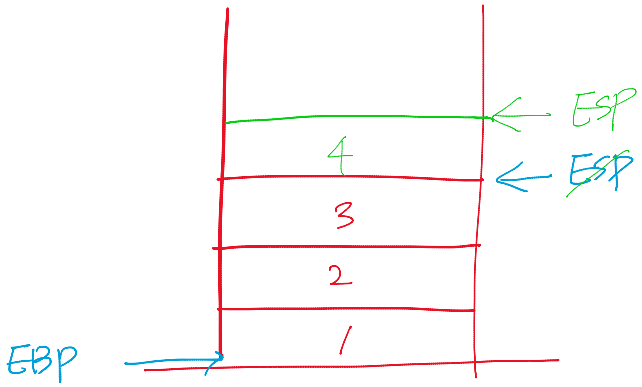
Push : stack에 data를 집어넣는다.

Push 1

Push 2

Push 3

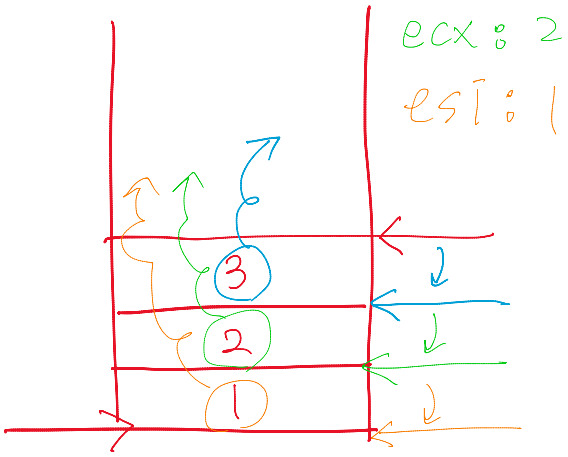
Push 4



Pop oper ①

eax : 3

```
push 1
push 2
push 3
pop eax
pop ecx
pop esi
```



| mov oper①, oper② : ① ← ②

size : oper① = oper②

ex)
mov eax, 1 ← eax에 1 대입.

segment register (생략 X)

변형 mov DWORD ptr ss:[100], 1

4byte 할당을 / 100번지 찾아서 / 1로 대입

⇒ "100번지 찾아서 4byte 할당을 1로 대입"

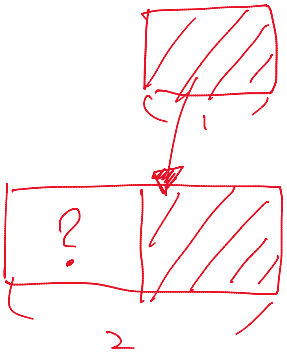
zero

movzx oper①, oper② ⇒ 남은공간 '0'으로 채움

signed bit

movsx oper①, oper② ⇒ 남은공간 '부호비트'로 채움

size : oper① > oper②



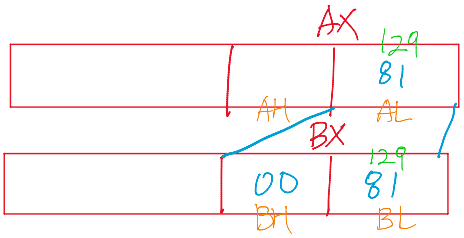
* 따져줘야 하는 Data가

① Unsigned data 면 ⇒ movzx

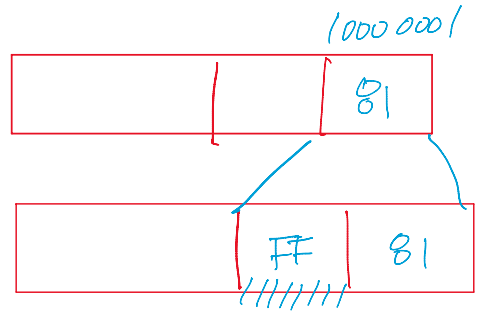
② signed data ⇒ movsx

ex) — — — — — — — —

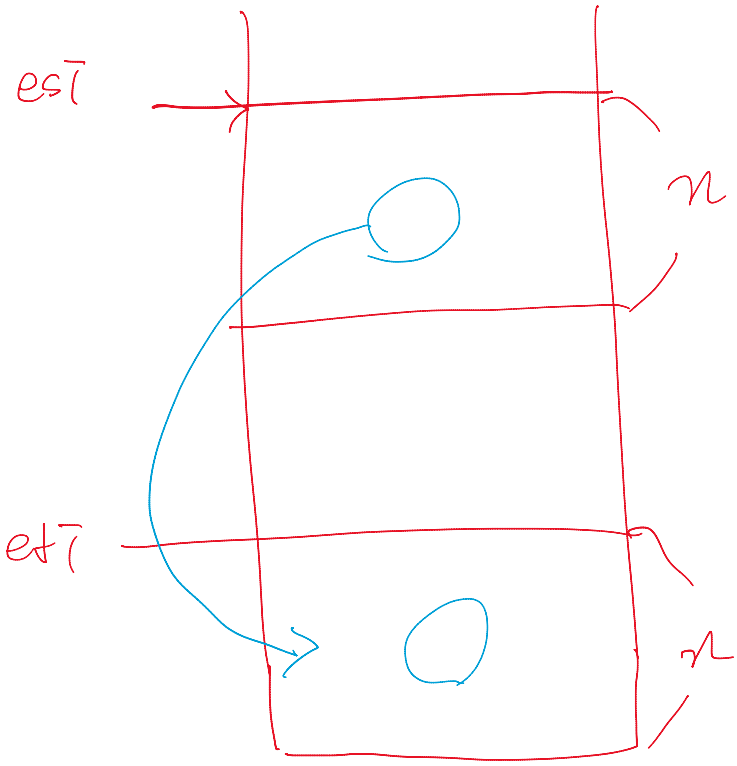
mov al, 81(16)
movzx bx, al



mov al, 81(16)
movsx bx, al



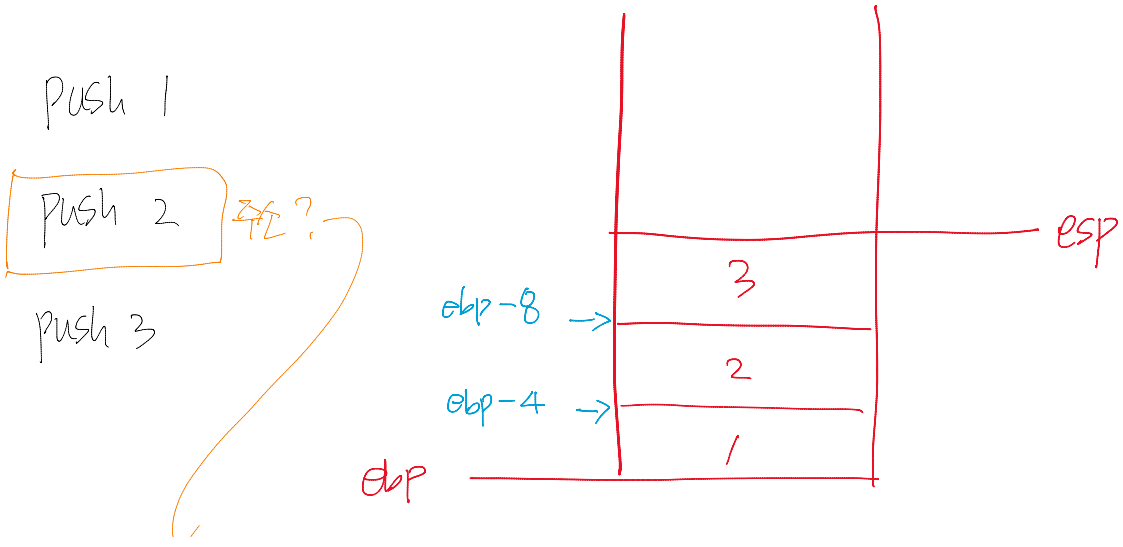
movs oper①(edi), oper②(esi) ⇒ esi에서 edi로 한개씩 복사



lea oper①, oper② ⇒ oper②의 "주소"를 oper①에 대입.

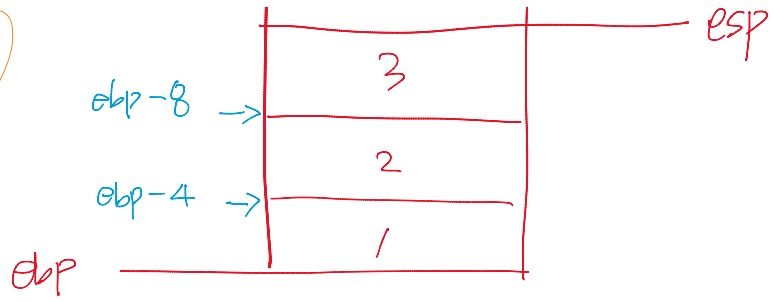
↳ oper① = &oper②

ex) - - - - - - - - - - - - - - - - - -



push 2

push 3



lea ecx, DWORD ptr ss:[ebp-8]

ebp-8의 값을 ecx에 저장.