REPORT:
x86어셈블리어

COLONY
1824275 정상지

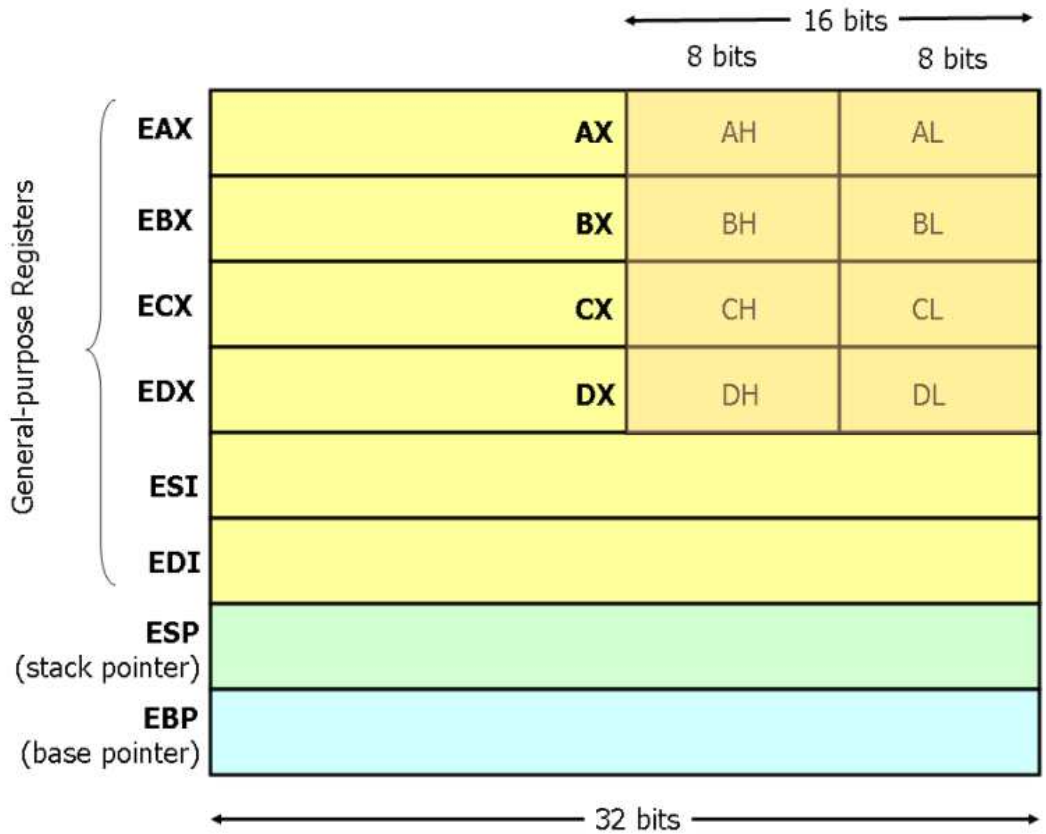
2021.04.07

(1) 어셈블리어

- 어셈블리어 언어는 기계어와 일대일 대응이 되는 컴퓨터 프로그래밍의 저급 언어

(2) 범용 레지스터

1) x86 기반 CPU는 8개의 범용 레지스터를 가지고 있으며, 각각 4byte(32bit)의 크기를 저장



2) 각 레지스터의 사용

- EAX(Extended Accumulator Register)
  : EAX 레지스터는 산술, 논리 연산을 수행하며 함수의 반환값이 레지스터에 저장
- EBX(Extended Base address Register)
  : ESI 또는 EDI 레지스터와 결합이 가능하며, EBX 레지스터는 메모리 주소를 저장
- ECX(Extended Counter Register)
  : 반복 명령어 사용 시 반복 카운터로 사용되는 레지스터로 ECX 레지스터에서 반복 횟수를 지정하고 반복 작업을 수행
- EDX(Extended Data Register)
  : EAX와 같이 쓰이고 부호 확장 명령 등에 사용
- ESI(Extended Source Index)
  : 데이터 복사/조작 등의 과정에서 자료형 변수의 시작점 포인터로 사용

- EDI(Extended Destination Index)
  : ESI 레지스터와 비슷하나, EDI 레지스터에는 복사 시 목적지의 주소가 저장
- EBP(Extended Base Pointer)
  : 스택의 시작 지점 주소가 저장. 스택이 소멸되지 않는 이상 레지스터 값은 변하지 않음
- ESP(Extended Stack Pointer)
  : 스택의 마지막 지점의 주소가 저장. PUSH, POP 명령에 따라 ESP 값이 4바이트씩 변함

(3) 명령어

1) PUSH, POP
- PUSH: 스택에 데이터를 삽입하는 명령어
  > push word/dword --> push 수행 시 esp -4h를 수행한 후 해당 공간에 데이터 삽입
- POP: 스택에 데이터를 꺼내와 지정한 레지스터에 삽입하는 명령어
  > push와 반대로 pop 수행 시 데이터를 가져온 후 esp +4h를 수행

2) MOV
- MOV: 데이터를 복사하는 명령어
  > 레지스터와 레지스터 같은 사이즈의 데이터끼리만 복사
  > 메모리에서 메모리로 복사 불가능
- Syntax
  > mov <reg>,<reg>
  > mov <reg>,<mem>
  > mov <mem>,<reg>
  > mov <reg>,<const>
  > mov <mem>,<const>

3) LEA(복사, Load Effective Address - 유효주소로드)
- LEA: 레지스터 연산 결과를 레지스터에 저장
- mov와 다른 점 : mov는 주소가 가리키는 값을 로드하지만 lea는 해당 주소를 로드

4) ADD, SUB, INC, DEC
- ADD: add val1 val2 --> val2에 val1과 val2의 합을 저장
- SUB: sub val1 val2 --> val1에서 val2를 뺀 값을 val1에 저장
- INC: inc val --> val의 값을 1 증가
- DEC: dec val --> val의 값을 1 감소

5) IMUL, IDIV
- IMUL: 정수 곱셈을 계산할 때 사용하는 명령어
  > imul <reg32>,<reg32/mem> --> 첫 번째 인자에 첫 번째 인자와 두 번째 인자의 곱을 저장
  > imul <reg32>,<reg32/mem>,<const> --> 첫 번째 인자에 두 번째 인자와 세 번째 인자의

곱을 저장

> 첫 번째 인자는 레지스터여야만 하며, 세 번째 인자가 올 경우 반드시 const 형태의 정수만 사용가능

- IDIV: 정수 나눗셈을 계산할 때 사용하는 명령어

> idiv ebx --> EDX:EAX를 EBX로 나눈 후 몫을 EAX에 나머지를 EDX에 저장

6) AND, OR, XOR

- 3개의 명령어 모두 비트 논리 연산자로 AND는 두 비트가 1일 경우, OR은 두 비트중 하나가 1일 경우, XOR은 두 비트가 모두 0일 경우를 1로 연산

7) CMP

- CMP: 비교 연산 명령어로 두 값을 빼서 비교. 따로 값을 저장하지않고 jmp와 mov와 같이 사용

8) TEST

- TEST: 두 값을 AND 연산하여 결과값을 CMP와 마찬가지로 저장하지않고 분기문에 영향을 준다.

> test eax,eax --> eax가 0인지 판단하는 것으로 사용가능

9) JMP

- JMP: 말 그대로 다른 주소로 이동하는 명령어

> JMP는 조건 없이 이동하지만 밑에 표의 명령어의 경우 조건을 만족시킬 때 이동한다

명령어	명령어 의미	부등호	플래그 조건
JA	Jump if (unsigned) above	>	CF = 0 and ZF = 0
JAE	Jump if (unsigned) above or equal	>=	CF = 0
JB	Jump if (unsigned) below	<	CF = 1
JBE	Jump if (unsigned) below or equal	<=	CF = 1 or ZF = 1
JC	Jump if carry flag set		CF = 1
JCXZ	Jump if CX is 0		CX = 0
JE	Jump if equal	==	ZF = 1
JECXZ	Jump if ECX is 0		ECX = 0
JG	Jump if (signed) greater	>	ZF = 0 and SF == OF
JZ	Jump if zero	==	ZF = 1

10) CALL, RET

- CALL: 함수 호출 명령으로 현재 주소를 EIP에 저장하고 함수를 호출한 주소로 이동
- RET: 호출한 함수 바로 다음 지점으로 이동. pop EIP

(4) 어셈블리어 핸드레이

- 어셈블리어 핸드레이를 통해 위의 명령어에 익숙해지기
- c언어의 코드를 어셈블리어로 변환해보기

```c
#include<stdio.h>

int main()
{
    int a, b;
    a = 3;
    b = 5;

    int c = a + b;

    return 0;
}
```

<어셈블리어 핸드레이>

```
push ebp
mov ebp,esp
sub esp,12
mov DWORD PTR [ebp-4],3
mov DWORD PTR [ebp-8],5
mov eax, DWORD PTR[ebp-4]
add eax, DWORD PTR[ebp-8]
mov DWORD PTR[ebp-12], eax
```