어셈블리어

유튜브 주소:
https://www.youtube.com/watch?v=g9PZ6os2ncg

Contents

CryptoCraft LAB

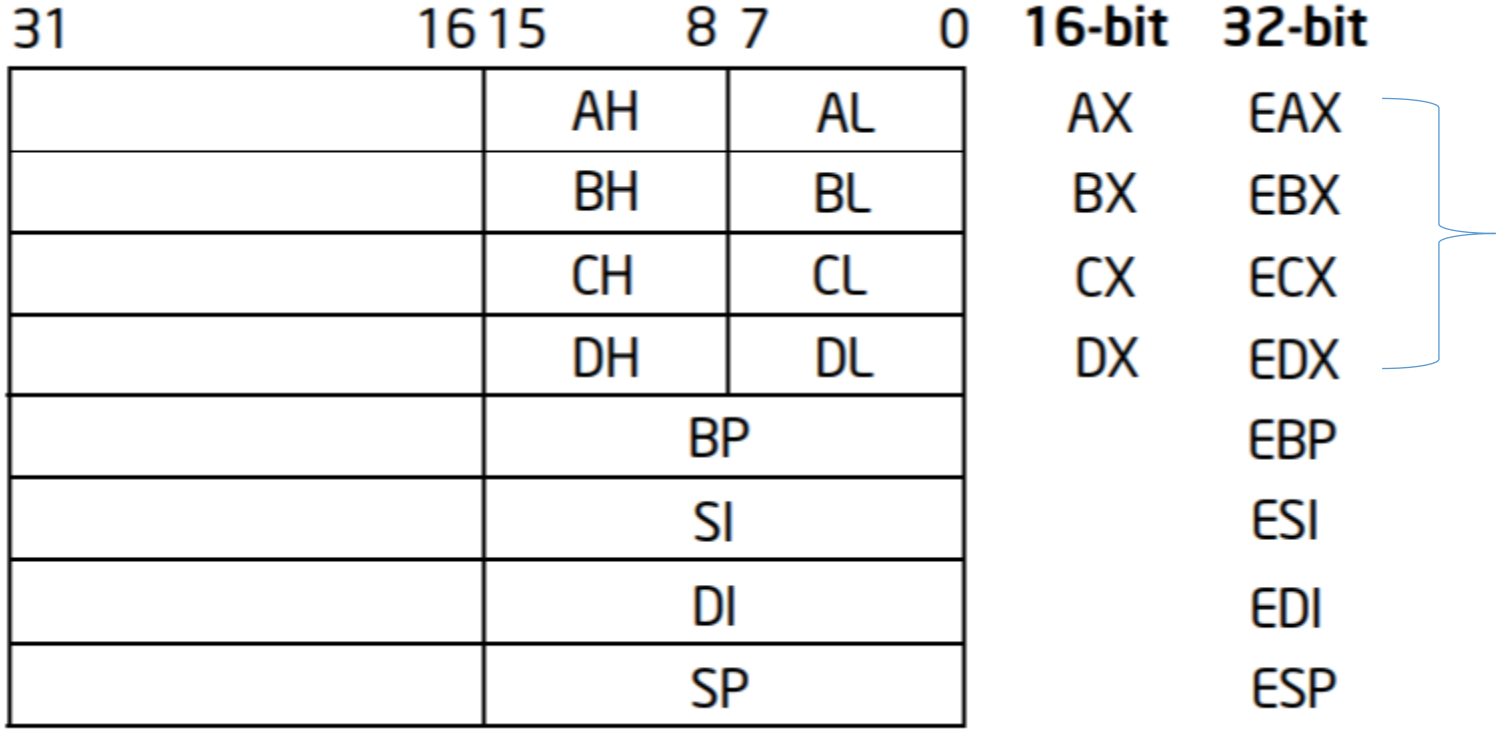
어셈블리어

기계어: 컴퓨터가 읽을 수 있는 2진 숫자로 표현
어셈블리어: 기계어를 사람이 보기 쉽게 표현
기계어와 1대 1 대응
컴파일 시 실행 속도가 빠름.

장점
매우빠름, 프로세서의 자원을 직접 접근
단점
개발속도는 느림 매우 많은 버그 발생
유지 보수가 힘듬(코드 수정이 어려움)

Opcode	Operand1	Operand2
ADD	EAX	EBX
	↑	↑
	destination	source

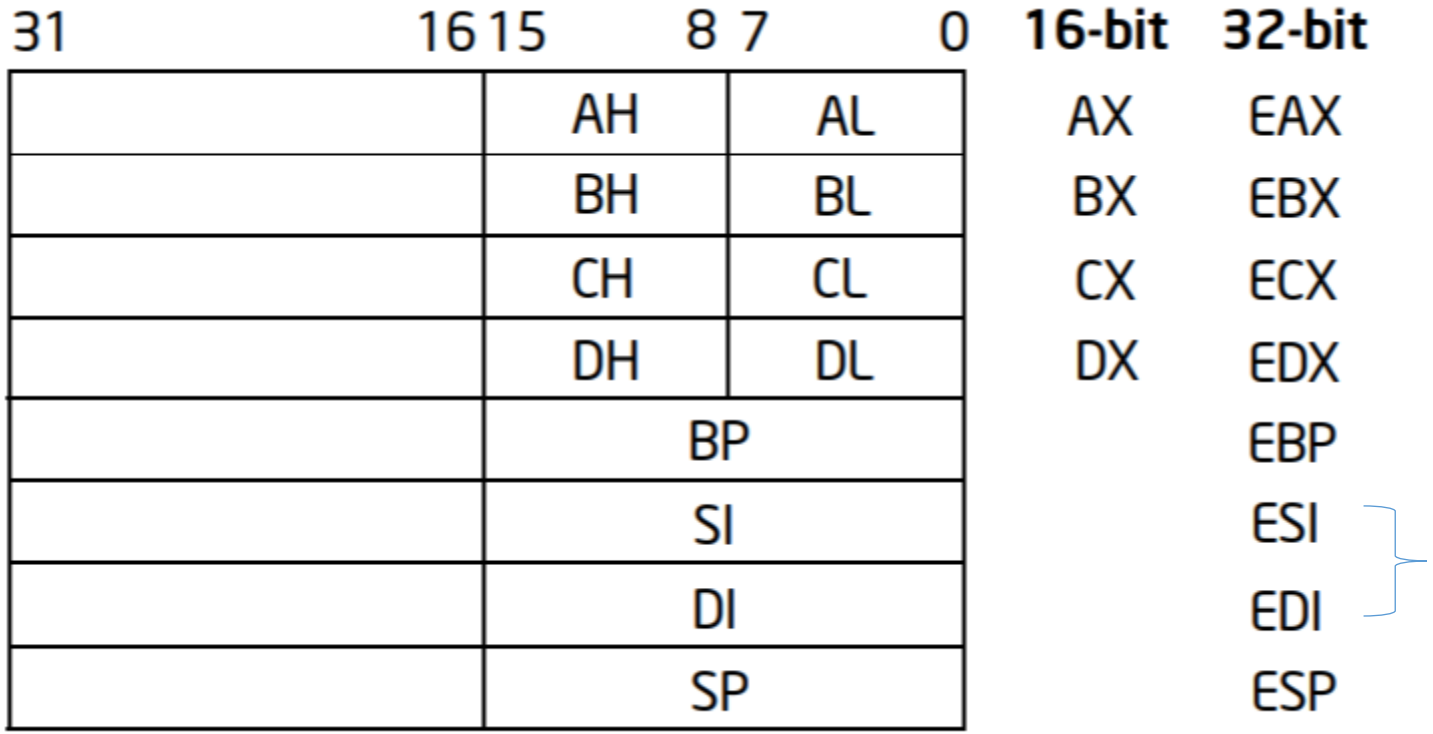
레지스터



범용 레지스터

EAX : 사칙연산 등 산술 연산에 자동으로 사용, 함수의 반환 값을 처리할 때도 사용
EBX : 간접 번지 지정(메모리 주소 지정) 에 사용, 산수, 변수를 저장합니다.
ECX : 반복(Loop)에서 반복 Count 역할을 수행
EDX : EAX를 보조,
Ex) 나누기 계산 시 몫은 EAX에 나머지는 EDX에 저장

레지스터



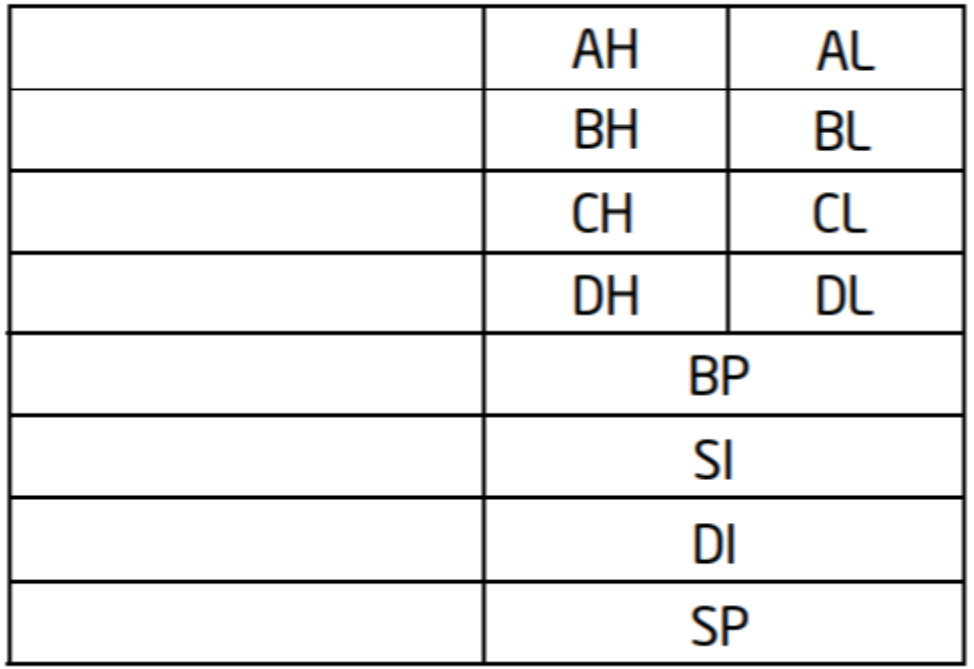
16-bit	32-bit
AX	EAX
BX	EBX
CX	ECX
DX	EDX
	EBP
	ESI
	EDI
	ESP

인덱스 레지스터

ESI : 복사나 비교를 할 경우 출발지 주소를 저장하는 레지스터

EDI : 복사나 비교를 할 경우 목적지 주소를 저장하는 레지스터

레지스터



16-bit	32-bit
AX	EAX
BX	EBX
CX	ECX
DX	EDX
	EBP
	ESI
	EDI
	ESP

포인터 레지스터

ESP : Stack Pointer, 가장 최근에 저장된 공간의 주소(스택의 최종점)를 저장하는 레지스터입니다.
EBP : 베이스 포인터. Stack Pointer의 기준점(바닥 부분)을 저장하는 레지스터,
Ex) 함수 호출시 그 순간의 esp를 저장하고 반환 전 ESP에 값을 되돌려줌.

EIP

상태 레지스터

EIP : 다음에 실행할 명령어의 주소를 가지고 있는 레지스터
현재 실행하고 있는 명령어가 종료되면 이 레지스터에 있는 명령어를 실행

기초 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
lea	leal(%esi), %ecx	%esi의 주소값을 %ecx에 옮긴다.	주소 이동
inc	inc %eax	%eax의 값을 1 증가시킨다.	데이터 조작
dec	dec %eax	%eax의 값을 1 감소시킨다.	데이터 조작
add	add %eax, %ebx	레지스터나 메모리의 값을 덧셈할 때 쓰인다.	논리, 연산
sub	sub $0x8, %esp	레지스터나 메모리의 값을 뺄셈할 때 쓰인다.	논리, 연산
call	call proc	프로시져를 호출한다.	프로시져
ret	ret	호출했던 바로 다음 지점으로 이동.	프로시져
cmp	cmp %eax, %ebx	레지스터와 레지스터값을 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)	

어셈블리어 예시

```
#include <stdio.h>
void main()
{
    int a,b,sum,sub,mul,div;
    scanf("%d %d",&a,&b);
    _asm{
        mov eax,a;
        add eax,b;
        mov sum,eax;
        mov eax,a;
        sub eax,b;
        mov sub,eax;
        mov eax,a;
        mul b;
        mov mul,eax;
        mov eax,a;
        div b;
        mov div,eax;

    }
    printf("%d %d %d %d",sum,sub,mul,div);
}
```

int a,b,sum,sub,mul,div;
scanf(“%d %d”, &a,&b);

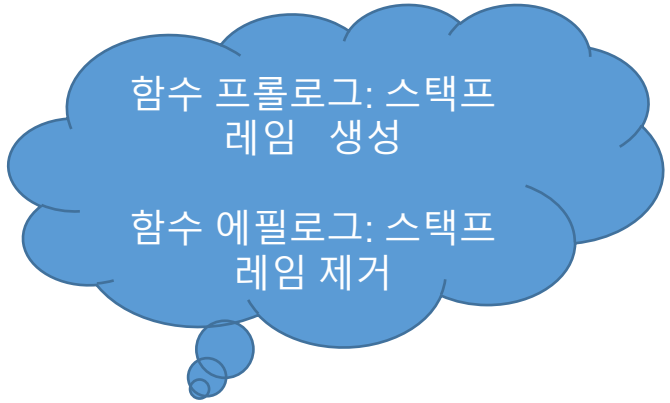
sum= a+b;

sub = a – b;

mul =a x b;

div = a / b;

함수호출 규약- cdecl



- 스택을 사용해서 push,
- Caller 에서 스택 정리! * add esp, n 사용
- print()함수와 같이 가변 길이 파라미터를 전달 가능

stdcall, fastcall에서는 가변 길이 파라미터를 전달하는 기능을 구현하기 어렵다.

```
int function(int a,int b){
     return a+b;
}
int main(){
    int x=function(1,2);

    return 0;
}
```

```
push  2
push  1
call    function
add   esp, 8
```

함수호출 규약- stdcall

- 스택을 이용해서 push
- Callee 가 스택 정리!
- 보통 dll 파일내 api들이 stdcall 방식
- 장점 : Callee에 Stack 정리 코드가 있어서 Caller의 크기가 cdecl방식에 비해 작아진다.

```
int function(int a,int b){
     return a+b;
}
int main(){
    int x=function(1,2);

    return 0;
}
```

```
push  2
push  1
call    function
```

함수호출 규약- fastcall

- 모든 인자 스택에 push x, 레지스터 이용(edx, ecx)
- 인자가 여러 개면, 나머지는 스택 사용, callee에서 스택정리
- 레지스터를 이용하여 속도가 빠름

```
• int function(int a,int b,int c,int d){
    return a+b+c+d;
}
• int main(){
   int x=function(1,2,3,4);

   return 0;
}
```

```
push  4
push  3
mov  edx,2
mov  ecx,1
call   function
```

Q & A