제3장

인텔 프로세서의 내부구조

학습내용

* 32비트 인텔 프로세서
* 16비트 인텔 프로세서
* 기억장치
* 레지스터
* 데이터 표현

인텔 프로세서의 내부구조

- 어셈블리어를 배우기 위한 준비 과정
- 인텔프로세서의 내부 구조와 기본 특성
- 32비트 인텔 프로세서와 16비트 프로세서에 대해서
- 두 프로세서의 차이점을 이해한 후
- 16비트 인텔 프로세서를 중심으로 설명

32비트 인텔 프로세서

펜티엄 프로세스의 개요

* 93년에 발표한 586 마이크로프로세서의 이름(75p참조)
* 내부적으로는
  - 32비트 데이터 버스 사용
* 외부적으로는
  - 64비트 데이터 버스로 메모리와 연결하여,

    단일 버스 사이클 당 전송될 수 있는 데이터 량이 486 프로세서의 2배
* 수퍼스칼라 구조
  - 한번의 클럭 동안에 두 개의 명령어 실행 구조

32비트 프로세스의 기본 실행 환경

- 프로세서에서 동작하고 있는 프로그램이 명령어를 실행하고 데이터 및 코드를 저장하기 위해서는 자원을 할 당 받아야 한다.
- 이러한 자원들은 32비트 프로세서에서의 기본 실행 환경을 구성한다.
- 응용 프로그램, 운영체제 등이 작동하게 된다
- 그림은 32비트 프로세서의 기본 실행 환경을 구성하는 자원을 나타냄

32비트 프로세스의 기본 실행 환경

기본 프로그램 실행 레지스터

8개의 32비트 레지스터	범용 레지스터
6개의 16비트 레지스터	세그먼트 레지스터
32 비트	EFLAGS 레지스터
32 비트	EIP (명령어 포인터 레지스터)

FPU 레지스터

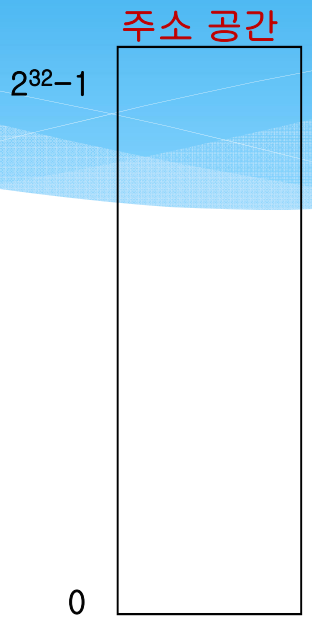
8개의 80비트 레지스터	실수 포인터 데이터 레지스터
16 비트	제어 레지스터
16 비트	상태 레지스터
16 비트	태그 레지스터
11 비트	연산 부호 레지스터
48 비트	FPU 데이터 연산 포인터 레지스터
48 비트	FPU 명령어 포인터 레지스터

MMX 레지스터

8개의 64비트 레지스터	MMX 레지스터 (Multimedia extension)

SSE와 SSE2 레지스터

32 비트	MXSCR 레지스터
8개의 128비트 레지스터	XMM 레지스터



32비트 프로세스의 기본 실행 환경

- 주소공간
  - 태스크나 프로그램 실행은 최고 **64GByte(2^36)**의 물리적 주소공간과 **4GByte(2^32)**의 선형주소 공간에 넣을 수 있다
- 기본 프로그램 실행 레지스터
  - 일반적인 명령어를 위해 실행되는 기본 실행 환경
    - **8**개의 **32**비트 범용레지스터
    - **6**개의 **16**비트 세그먼트 레지스터
    - **EFLAGS** 레지스터 ➔ **32**비트
    - **EIP**(명령어포인터)레지스터 ➔ **32**비트

32비트 프로세스의 기본 실행 환경

- **X87 FPU(Floating Point Unit) 레지스터들 :**
  - 8개의 **FPU** 데이터 레지스터
  - **FPU** 제어 레지스터
  - 상태 레지스터
  - **FPU** 명령어 포인터 레지스터
  - **FPU** 연산항 포인터 레지스터
  - **FPU** 태그 레지스터
  - **FPU** 명령어 레지스터 는 부동 소수점 연산을 위한 실행환경을 제공

32비트 프로세스의 기본 실행 환경

- **MMX 레지스터**
  - 64비트로 패킷된 바이트, 워드,더블워드 정수에 대한 SIMD(single-instruction,Multiple-data)연산을 8개의 MMX 레지스터가 지원
  - SIMD(Single Instruction Multiple Data)
  - 병렬 프로세서의 한 종류로, 하나의 명령어로 여러 개의 값을 동시에 계산하는 방식 ,벡터 프로세서에서 많이 사용되는 방식으로, 비디오 게임 콘솔이나 그래픽 카드와 같은 멀티미디어 분야에 자주 사용
- **XMM 레지스터**
  - 128비트로 패킷된 단정도,배정도 실수 포인터 값,바이트,워드,더블 워드등에 대한 SIMD연산을 8개 XMM 레지스터와 MXCSR레지스터가 지원

기본 프로그램 실행 레지스터(16개)

* 범용 레지스터(32비트 8개): 연산 항, 포인터 저장
  * EAX,EBX,ECX,EDX,ESI,EDI,EBP,ESP
* 세그먼트 레지스터(16비트 6개): 메모리 세그먼트 식별
  * CS,DS,SS,ES,FS,GS
* EFLAGS 레지스터(32비트의 프로그램 상태/제어):
  * 실행되고 있는 프로그램의 상태보고 및 애플리케이션 레벨로 제한된 프로세서 제어
* EIP(32비트 명령어 포인트)
  * 다음에 실행될 명령어를 가리킴

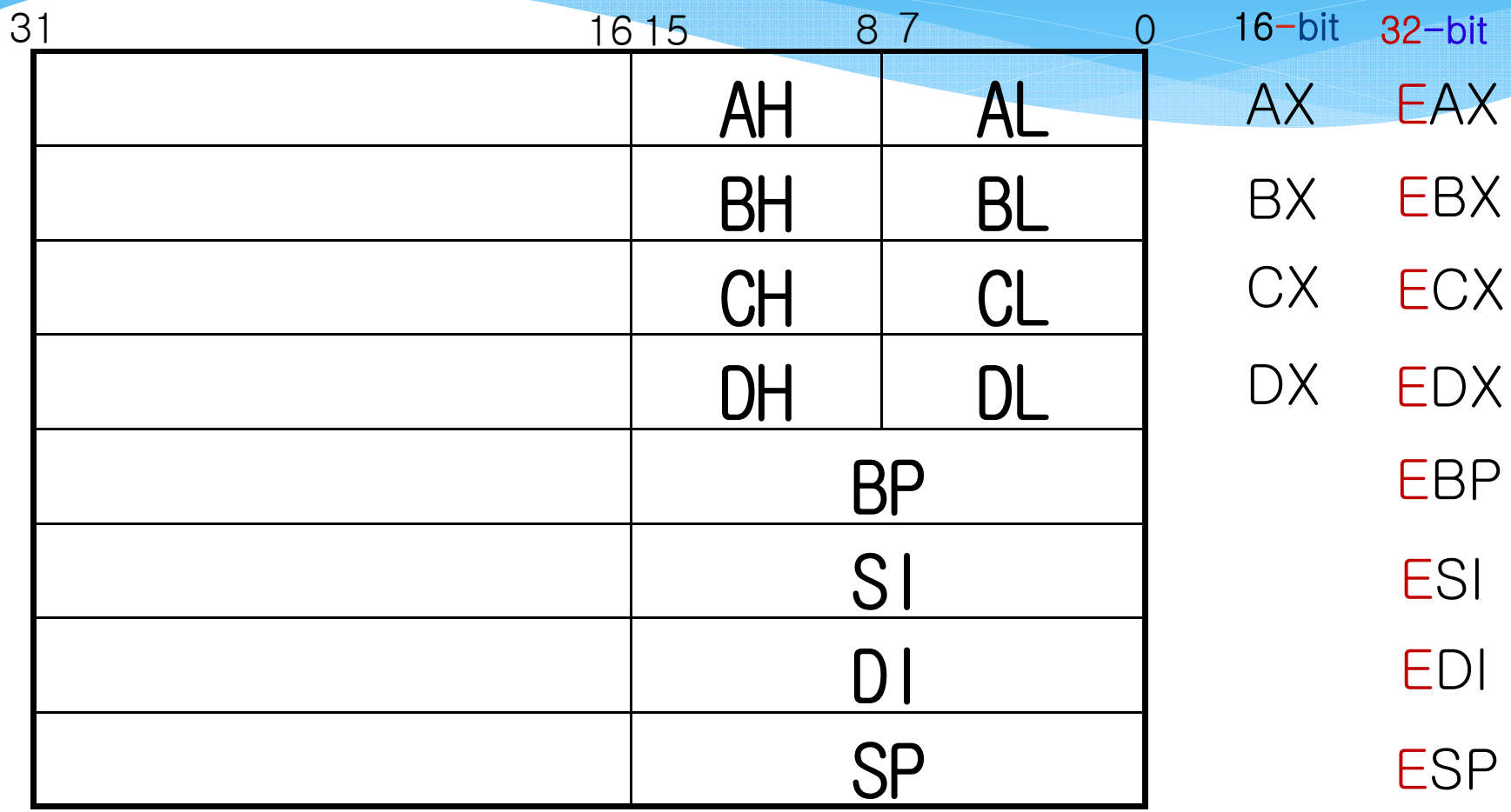
범용 레지스터

- 8086프로세서의
  - 데이터 레지스터, 포인터 레지스터, 인덱스 레지스터에 해당
- 각 레지스터의 길이는 16비트에서 32비트로 확장
- 범용 레지스터들은
  - 산술 논리 연산을 위한 연산 항 과 결과
  - 주소 계산을 위한 연산 항 과 결과
  - 메모리 포인터를 저장하기 위해 사용
- 4가지 방법으로 엑세스
  - EAX : 32비트 전체 엑세스
  - AX : EAX의 하위 16비트를 엑세스
  - AH : AX의 상위 8비트
  - AL : AX의 하위 8비트

범용 레지스터의 특별한 사용

* EAX : 연산항과 결과에 대한 누산기
* EBX : DS 세그먼트의 데이터에 대한 포인터
* ECX : 문자열과 반복문에 대한 계수기
* EDX : 입출력에 대한 포인터
* ESI : 문자열 연산에 대한 출발 항 또는 DS 레지스터가 가리키고 있는 세그먼트의 데이터에 대한 포인터
* EDI : 문자열 연산에 대한 목적 항 또는 ES 레지스터가 가리키고 있는 세그먼트의 데이터에 대한 포인터
* ESP : 스택 포인터
* EBP : 스택에 있는 데이터에 대한 포인터

범용 레지스터 이름


31
16 15
8 7
0
16-bit
32-bit
AH
AL
AX
EAX
BH
BL
BX
EBX
CH
CL
CX
ECX
DH
DL
DX
EDX
BP
EBP
SI
ESI
DI
EDI
SP
ESP

세그먼트 레지스터

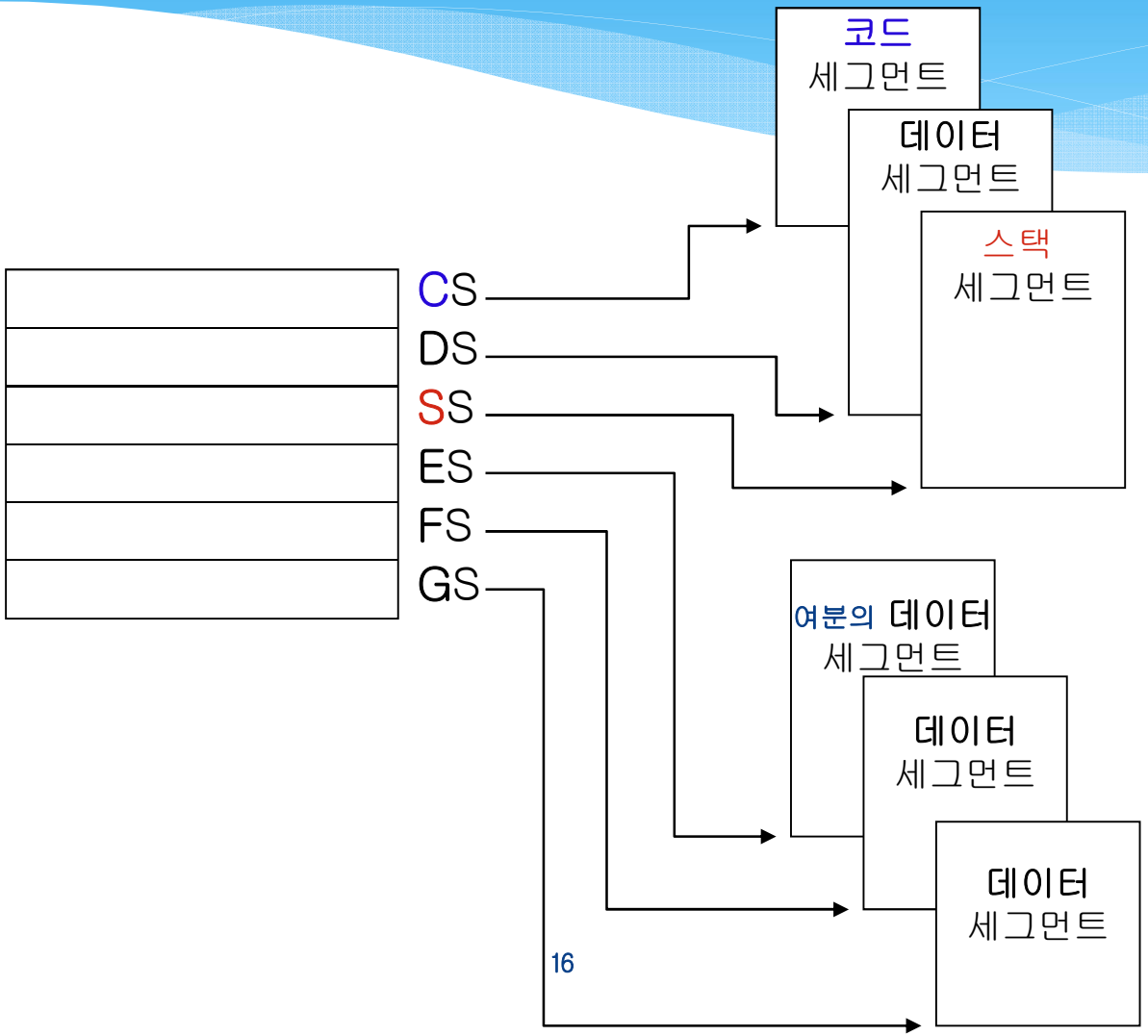
메모리 세그먼트를 식별 할 수 있게 한다.

16비트로 구성되었으며 FS와 GS 레지스터가 추가되어 한번에

서로 다른 6개의 다른 세그먼트가 활성화 될수있다.

* CS : 실행될 명령어가 저장되어 있는 세그먼트의 시작 주소
* DS, ES, FS, GS
  * 4개의 데이터 세그먼트를 가리킨다.
  * 4개의 데이터 세그먼트는 서로 다른 데이터 타입으로 이루어져 안정하고 효율적인 접근을 허가
* SS : 스택 세그먼트의 시작 위치

세그먼트 레지스터


코드
세그먼트
데이터
세그먼트
스택
세그먼트
CS
DS
SS
ES
FS
GS
여분의 데이터
세그먼트
데이터
세그먼트
데이터
세그먼트

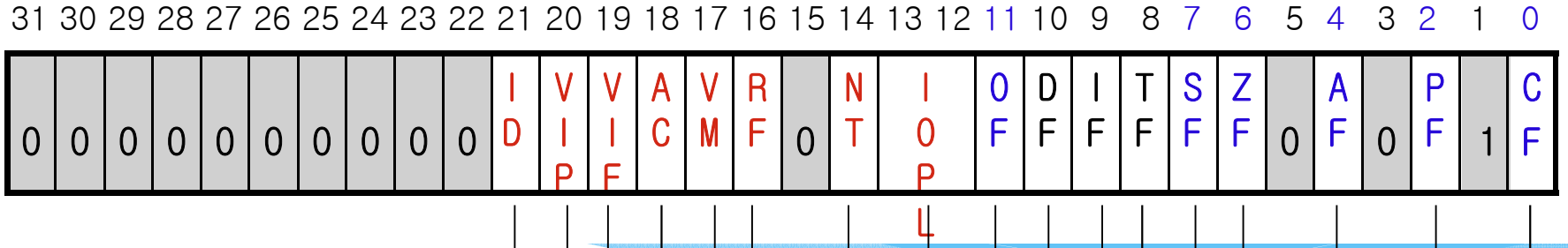
세그먼트 레지스터


Code Segment : CS
Data Segment : DS
Stack Segment : SS
Extra Segment : ES

EFLAGS 레지스터

* 32비트 EFLAGS 레지스터는
  * 여러 상태 플래그와 시스템플래그, 제어 플래그로 구성
* 1, 3, 5, 15, 22-31 번째 비트는 예약된 비트이다.
* 8086 프로세서에서 16비트 였던 플래그 레지스터에 8개의 새로운 플래그가 추가.(ID-IOPL)

EFLAGS 레지스터



X ID FLAG (ID)
X Virtual Interrupt Pending (VIP)
X Virtual Interrupt Flag (VIF)
X Alignment Check (AC)
X Virtual-8085 Mode (VM)
X Resume Flag (RF)
X Nested Task (NT)
X I/O Privilege Level (IOPL)
X Overflow Flag (OF)
C Direction Flag (DF)
X Interrupt Enable Flag (IF)
X Trap Flag (TF)
S Sign Flag (SF)
S Zero Flag (ZF)
S Auxiliary Carry Flag (AF)
S Parity Flag (PF)
S Carry Flag (CF)

S 상태 (Status) Flag
C 제어 (Control) Flag
X 시스템 (System) Flag

예약 비트
현재 사용하지는 않는다.

EFLAGS 레지스터

- 상태플래그
  - EFLAGS 레지스터의 상태 플래그(0,2,4,6,7,11번째 비트)는 ADD,SUB,MUL,DIV와 같은 산술 연산의 결과를 가르킨다

CF(Carry Flag)	최상위 비트에 자리올림 또는 내림이 발생하면 1로세팅.부호없는 정수에 대한 오버플로우를 나타낸다
PF(Parity Flag)	오퍼레이션 결과의 최하위 비트가 짝수패리티(1을 유지하고 있는 비트수가 짝수인경우)인 경우에 세트되고,홀수의경우는 클리어(0이된다).데이터 전송에 있어서 에러 검출에 사용한다.
AF(Auxiliary Carry Flag)	10진 산술연산(BCD)명령에 사용하고, 하위 니블(4비트)에서 상위 니블로의 자리올림 또는 상위에서 하위로 자리내림이 있을때 1로 세트된다.
ZF(Zero Flag)	연산결과가 0으로 되었을 때 1로 세트 된다.
SF(Sign Flag)	연산 결과의 최상위 비트가 1 일 때 1로 세트 된다.즉 최상위 비트가 1이면 음수, 0이면 양수를 나타낸다.
OF(Overflow Flag)	산술 연산 결과의 오버플로우 상태를 표시한다.

EFLAGS 레지스터

* DF 플래그
  * Direction Flag는 문자열 명령에서의 실행 방향을 제어한다.
  * DF플래그가 1로 세팅 되어 있으면 자동 감소 방향으로 문자열 명령이 실행된다.
  * DF플래그가 0인 경우엔 문자열 명령은 번지를 나타내는 레지스터 값이 자동 증가 방향으로 실행된다.
* 시스템 플래그와 IOPL 필드
  * 운영체제를 제어하는데 이용되며, 응용프로그램에 의해서 변경 되서는 안된다.

EFLAGS 레지스터

TF(Trap Flag)	프로세서를 프로그램 디버그용의 싱글 스텝모드가 가능하도록 세팅
IF(Interrupt-enable Flag)	외부 인터럽트가 가능또는 블가능 을 나타낸다
IOPL(I/O privilege level field)	현재 실행중인 프로그램이나 태스크의 입출력에 대한 우선 순위 레벨을 가리킨다.
NT(Nested task flag)	현재 실행하고 있는 작업이 이전에 실행된 작업과연결되어 있는경우 세팅된다.즉, 현재 태스크가 다른 태스크에 의해 호출되었음을 나타낸다.
RF(Resume flag)	디버깅 에외에 대한 프로세스의 응답 제어, 세팅되면 디버그 오류는 무시되고 다음 명령어가 정상적으로 실행된다
VM(Virtual-8086 mode flag)	세팅되면 가상8086 모드가 가능
AC(Aligement check flag)	세팅되면 메모리 레퍼런스 정렬검사가 가능
VIF(Vitual interrupt flag)	VIP 플래그와 같이 사용됨. 인터럽트 플래그 IF의 가상 이미지이다.
VIP(Vitual interrupt pending flag	인터럽트가 미결상태에 있음을 나타낸다
ID(Identification flag)	CPUID 명령어가 이 플래그를 테스트한다. 만약 어떤 프로그램이 ID 플래그를 세트 및 클리어 시킬수 있다면프로세서는 CPUID 명령어를 지원하게 된다.

16비트 인텔 프로세서

Ⅰ. 프로세서의 구조

1. 프로세서의 구조

1) 8086 프로세서의 구조

(1) Intel 프로세서의 발전사

- 인텔사에서 1978년 8086 출시
- 8088, 80186, 80286, ... , Pentium 등으로 발전
  (80i86 계열 프로세서)
- 데이터버스의 크기는 판독과 기록 사이클의 횟수를 결정
  32비트로 된 자료 : 16비트 데이터버스일 경우 2회 읽음
- 8086의 경우 20비트 주소버스, 16비트 데이터버스 사용
- 80386부터 32비트 주소버스 사용(4GB)
- 펜티엄의 경우 범용레지스터의 크기 : 64비트
  외부데이터버스 크기 : 64비트 (실제 32비트)

16 인텔 프로세서

* 펜티엄 프로세서의 강점
  * 기존 프로세서와의 소프트웨어의 호환성
  * 16비트 프로세서에서 사용하던 모든 명령어는 32비트에서도 사용가능
  * 기본적인 구조는 같다.
* 시스템 소프트웨어의 전반적인 구조를 보다 쉽게 이해하기 위해 16 비트 프로세서를 설명

16 인텔 프로세서의 구조

* 16비트 인텔 프로세서의 구조
  * 인텔사는 1978년 i8086을 내놓은 이후
  * 8088,80186,80286(16비트)
  * 80386,80486(32비트)
  * Pentium,p6등 계속 시판
* 80i86계열 프로세서
* 이 중 8088은 1981년 IBM PC에 채택되어 16비트 CPU의 표준이 됨

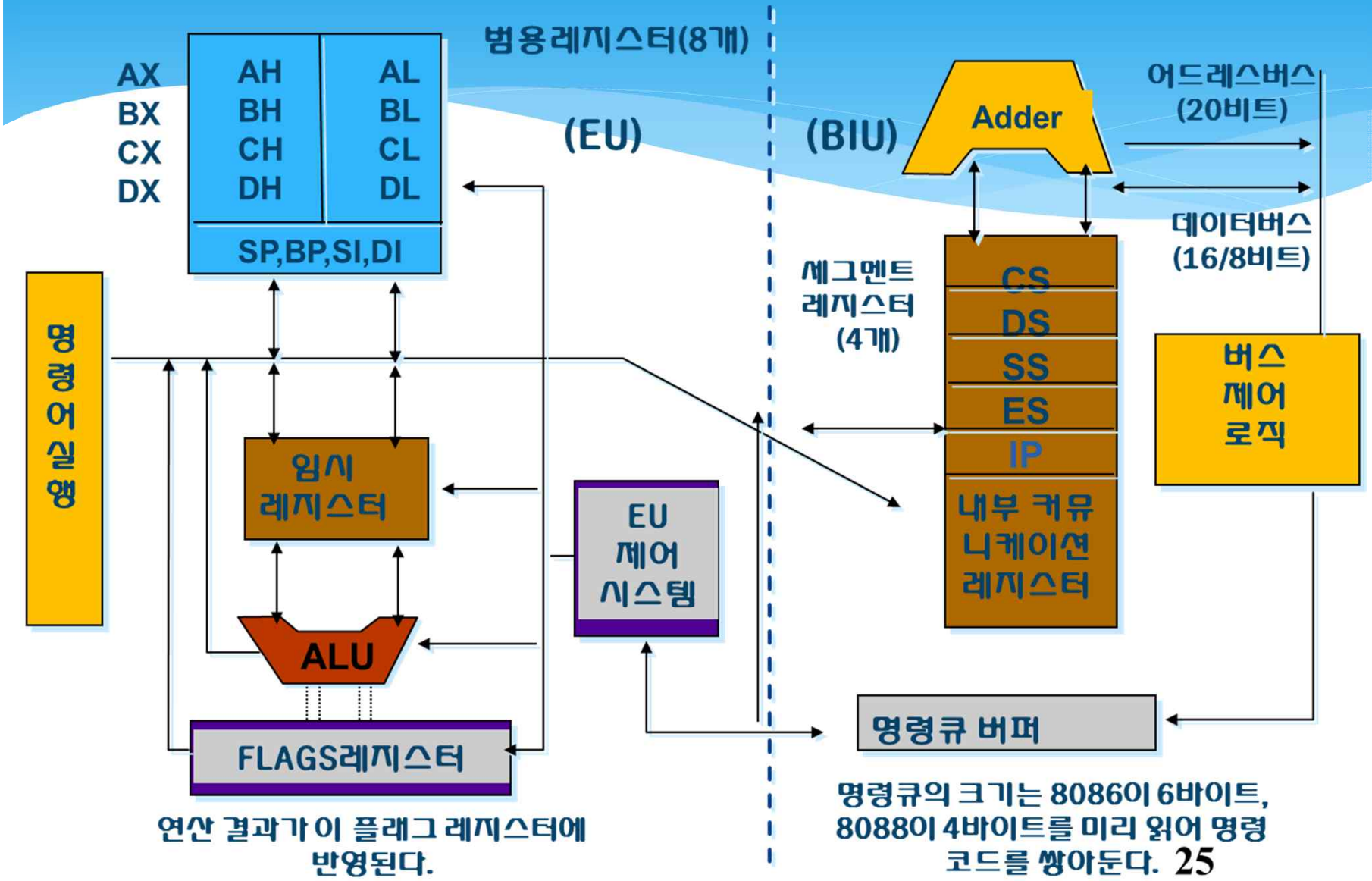
16 인텔 프로세서의 구조

* 8086CPU는 계산 능력이 8비트의 CPU보다 4000배 이상 빠름
  통신회사, 자동화 시스템 등 정보 사회의 모든 분야의 발전에 공헌
* 8088과 8086의 차이점
  * 8088은 데이터 버스가 8비트로 8비트 데이터를 처리
  * 8086은 16비트 데이터를 처리 할 수 있는 데이터 버스가 있다
* 8086중심으로 설명
  * 32비트로 된 자료를 읽기 위해서는 16비트 데이터 버스를 쓰는 8088/8086에서는 2회를 읽어야 한다.

16 인텔 프로세서의 구조

* 데이터 버스의 크기
  * 기억 장치에 있는 데이터를 이용하는데 판독과 기록사이클의 횟수를 결정
* 주소 버스는 최대 기억장치 주소를 결정
  * 8086에서는 20비트 주소를 이용
  * 2^20=1,048,576(1Mb)주소를 지정할 수 있다.
  * 8086/88시스템에서는 이 주소를 지정하기 위해 16비트 레지스터를 2개 사용해서 20비트 물리주소를 만들어 낸다.
  * 물리주소는 0에서 FFFFFh까지 1M바이트 메모리 공간에서 20비트 주소에 따라 표현
  * CPU와 메모리 사이에서 데이터 교환은 모두 이 물리 주소를 버스를 통해서 얻을수 있다

8086 프로세서 구조


범용레지스터(8개)
AX
BX
CX
DX
AH
BH
CH
DH
AL
BL
CL
DL
SP,BP,SI,DI
(EU)
(BIU)
Adder
어드레스버스
(20비트)
데이터버스
(16/8비트)
세그멘트
레지스터
(4개)
CS
DS
SS
ES
IP
내부 커뮤
니케이션
레지스터
버스
제어
로직
명령어실행
임시
레지스터
EU
제어
시스템
ALU
FLAGS레지스터
명령큐 버퍼
연산 결과가 이 플래그 레지스터에
반영된다.
명령큐의 크기는 8086이 6바이트,
8088이 4바이트를 미리 읽어 명령
코드를 쌓아둔다.

8086프로세서의 구조

- 명령어 실행장치(EU) : 명령을 실행하고 연산
  - 4개 범용레지스터
  - 산술 논리 연산장치(ALU)
  - 플래그 레지스터
- 버스인터페이스장치(BIU)

  명령어를 인출하고

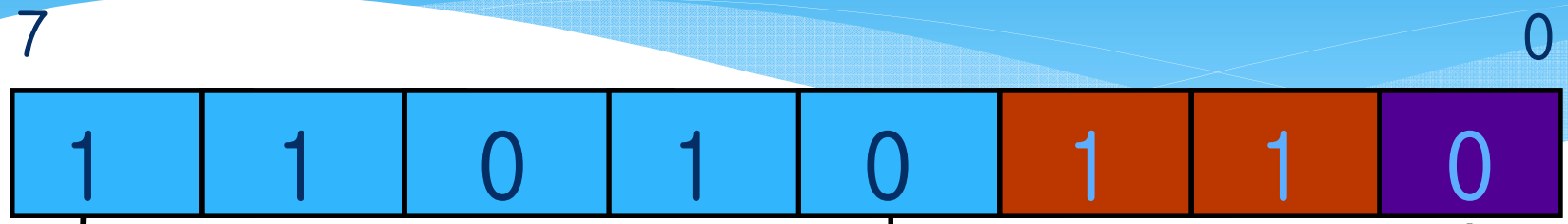
  기억장치나 입출력장치로 부터 데이터를 읽어 오거나 출력
  - 4개의 세그먼트 레지스터
  - 명령어 포인터(IP)
  - 가산기(adder) : 유효 주소 계산을 위한
  - 명령어 큐 : 실행할 명령어를 임시로 저장

명령어의 길이

* 8비트로된 명령어를 예
  * 2^8(256)개의 서로 다른 명령어를 만들 수 있다.
  * 명령어 : 연산항에 있는 주소의 지정방식과 레지스터를 명시
  * 주소 지정 방식이 4가지,레지스터가 2개가 있다면
  * 주소지정에 2비트,레지스터에 1비트가 필요
  * 명령어는 2^5(32)개만 만들수 있다.
* 명령어의 길이는
  * 레지스터의 개수,주소 지정 방식의 종류,명령어의 종류에 따라 결정
* 명령어의 길이를 길게 하면 기억 장치에서 읽어 오는데 많은 시간을 소비해야 한다.

명령어의 형태

8비트 명령어의 구성 예



명령어 코드(2^5=32)

00 = 직접
01 = 베이스
10 = 인덱스
11 = 베이스+인덱스

4가지의 주소지정 방식

2개의 레지스터

1 = 레지스터 A
0 = 레지스터 B

명령어의 길이는 레지스터의 개수,
주소지정 방식의 종류,명령어의 종류에 따라 결정

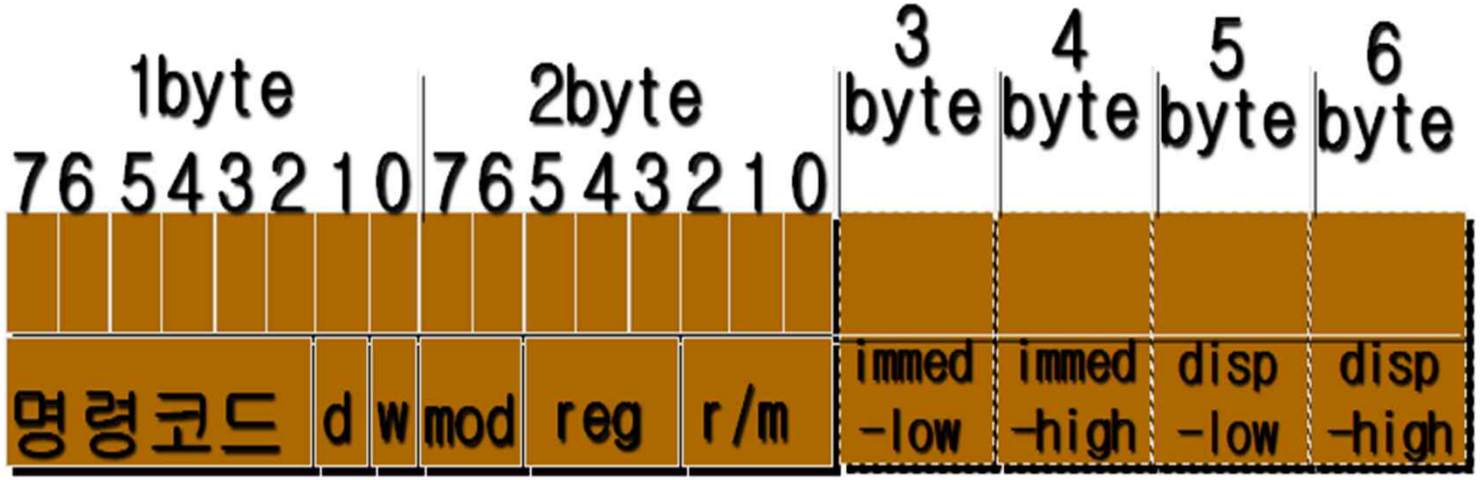
8086 명령어

* 14개의 기본 레지스터
  * 8개는 범용 레지스터
  * 4개의 세그먼트 레지스터
  * 명령어 포인터(IP)1개
  * 플래그 레지스터
* 명령에 따라 연산 항으로 범용레지스터를 사용 하는 것이 있고 3비트만 필요
* 데이터를 단어 단위로 전송하기 때문에 1바이트 데이터를 전송하는 데 1사이클의 시간이 필요

8086 명령어

* 데이터의 크기에 따라 연산 방법과 기억장치의 저장 방법이 달라져야 한다.
  * 2가지의 데이터 크기기 있어 명령어에 명시 : 1비트가 필요
    * (0 : 바이트 표시, 1 : 워드단위 표시)
  * 주소 지정 방식의 종류 명시 : 5비트 필요
  * 명령어가 기본적으로 1단어(2바이트)로 구성
    * 긴 명령어는 2단어(4바이트)
    * 최대 3단어(6바이트)까지 확장
* 프로세서가 명령어의 첫 단어를 읽어 온 후에 명령어가 여기서 끝났는지 다음 단어의 까지 확장 되어 있는지의 정보
  * 명령어의 길이 정보 : 첫 단어

8086 명령어


1byte
2byte
3 byte
4 byte
5 byte
6 byte
7 6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
명령코드
d
w
mod
reg
r/m
immed -low
immed -high
disp -low
disp -high

8086 명령어

* **14개의 기본레지스터**
  * 8개의 범용레지스터
  * 4개의 세그먼트 레지스터
  * 1개의 명령어 포인터
  * 1개의 플래그 레지스터
* **레지스터 크기 : 16비트 혹은 8비트**
* **데이터버스 : 16비트, 주소버스 : 20비트**
* **8086 계열 어셈블리어 명령어 크기**
  * • 1단어(2바이트)로 구성되며 최대 3단어(6바이트)까지 확장

기억장치

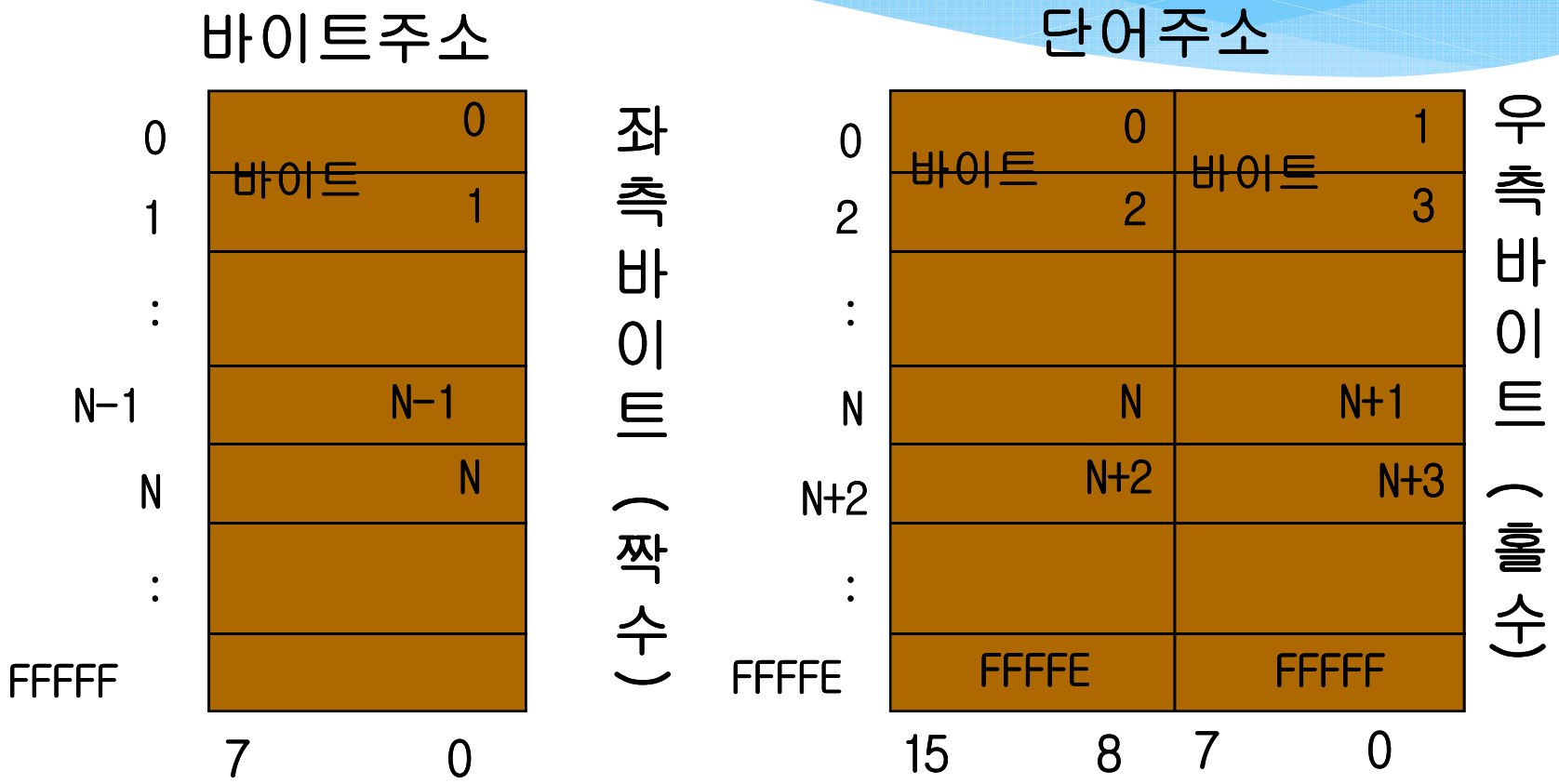
- 명령어의 작업:
  - 기억 장치의 데이터를 기록하거나 읽어오는 일

기억 장치의 모델

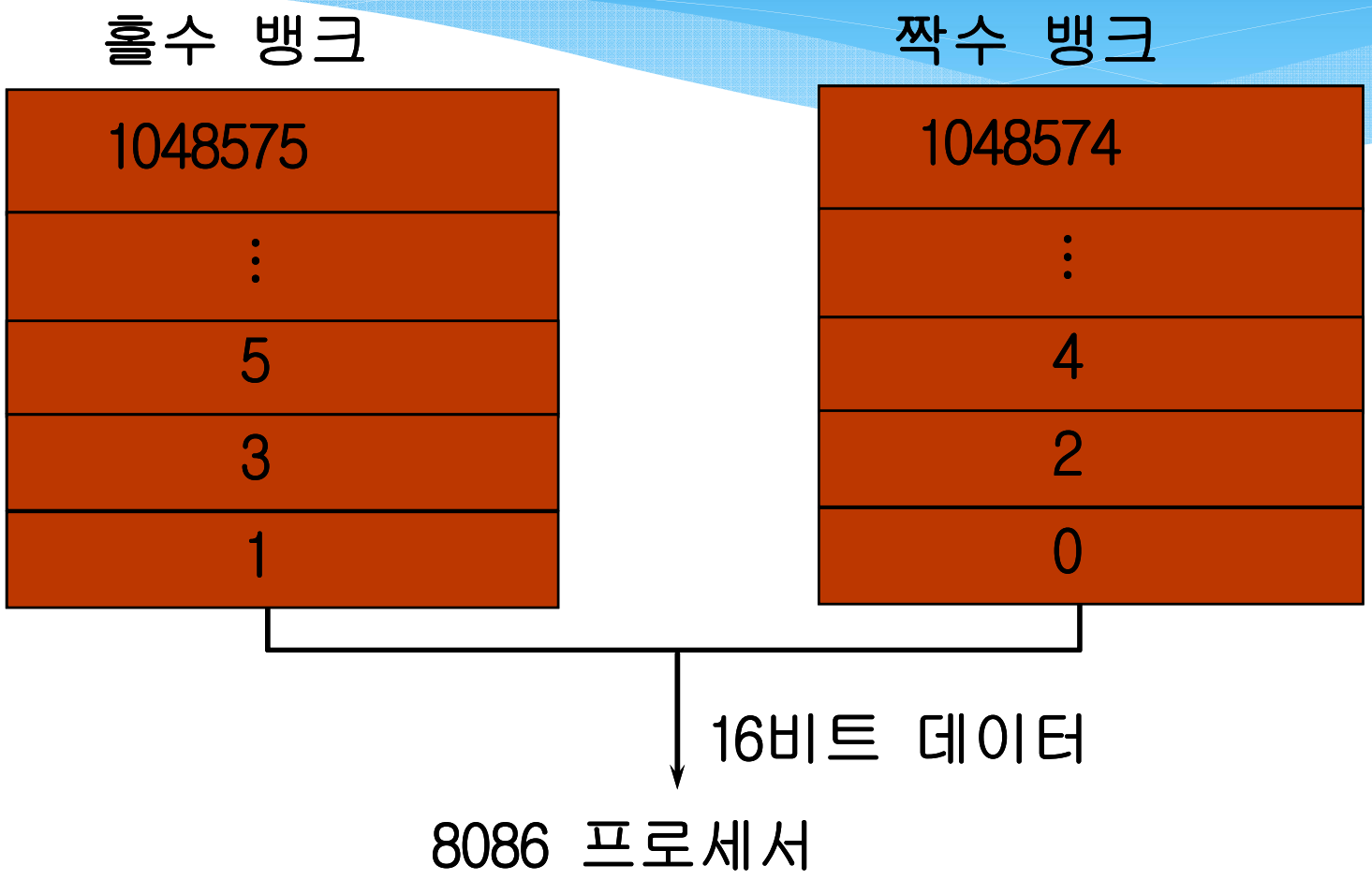
* 8086 프로세서와 함께 동작되는 기억장치
* 주소가 물리 주소로 붙여짐
* 20비트의 주소 버스를 사용
  * 1M바이트의 주소를 가짐
  * 이때 마지막 주소
  * 1111 1111 1111 1111 1111
  * FFFFFh
* 기억 장치를 바이트 단위로 사용

기억장치의 모델

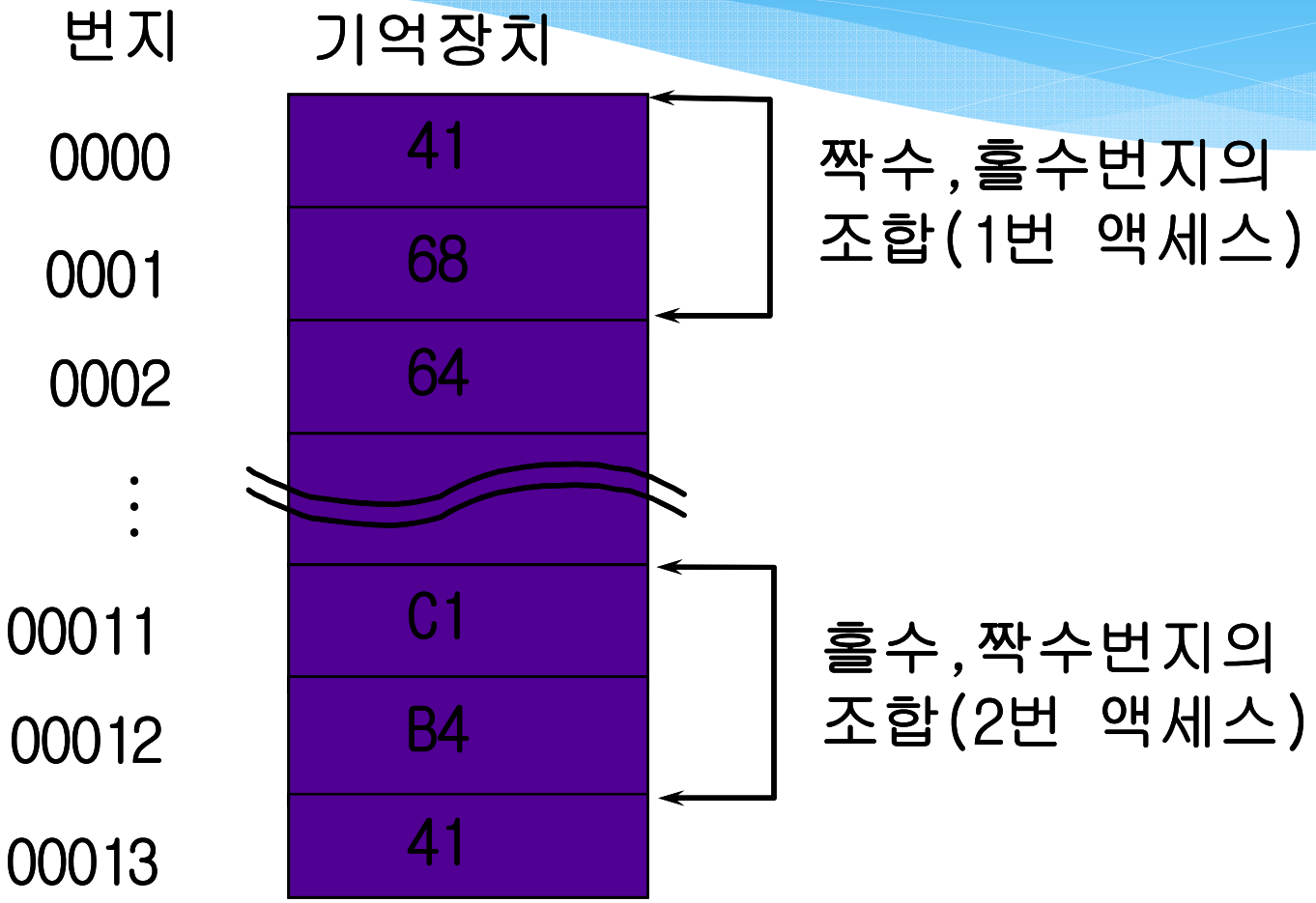
* 8086 프로세서의 예(20비트의 주소버스사용:1M)



기억장치의 실제구성



주소지정에 의한 액세스 효율



데이터 기록 시 일반적인 규칙

* 8086에서는 기억 장치에 세그먼트 단위로
  * 필요한 데이터나 프로그램을 저장

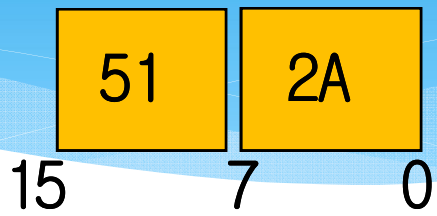
* 16비트 단위로 작업하고 처리할 데이터의 크기에 따라서 소요되는 사이클 시간이 다르다고 했다

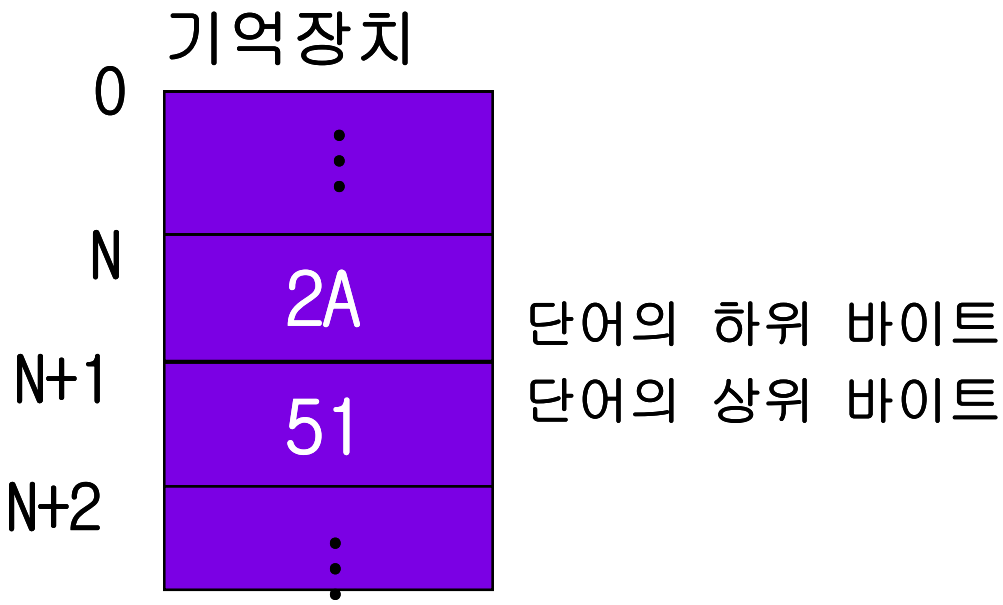
* 기억 장치에 데이터를 기록하고 읽을때 규칙

  * 기억장소 주소 : 바이트 단위로 되어 있음
    * 바이트 주소는 : 1씩 증가,짝수,홀수의값을 가짐
  * 단어 주소는 : 2씩 증가,짝수의 값을 가짐
  * 프로세서가 N번지의 1단어에 접근하면 자동으로
    → N, N+1 바이트에 접근
  * N번지의 긴 단어(Long word)에 접근하면 4개의 바이트에 접근하게 된다.
    → N, N+1, N+2, N+3에 접근

기억장치에서 단어 배열

N번지에 하나의 단어
512A가 들어 있을때





단어의 하위 바이트
단어의 상위 바이트

레지스터에서 기억장치로 데이터를 쓰거나 읽어올때는 그 단어의 하위 바이트를 먼저 기록하고 난 후에 상위 바이트를 가록하게 된다

레지스터

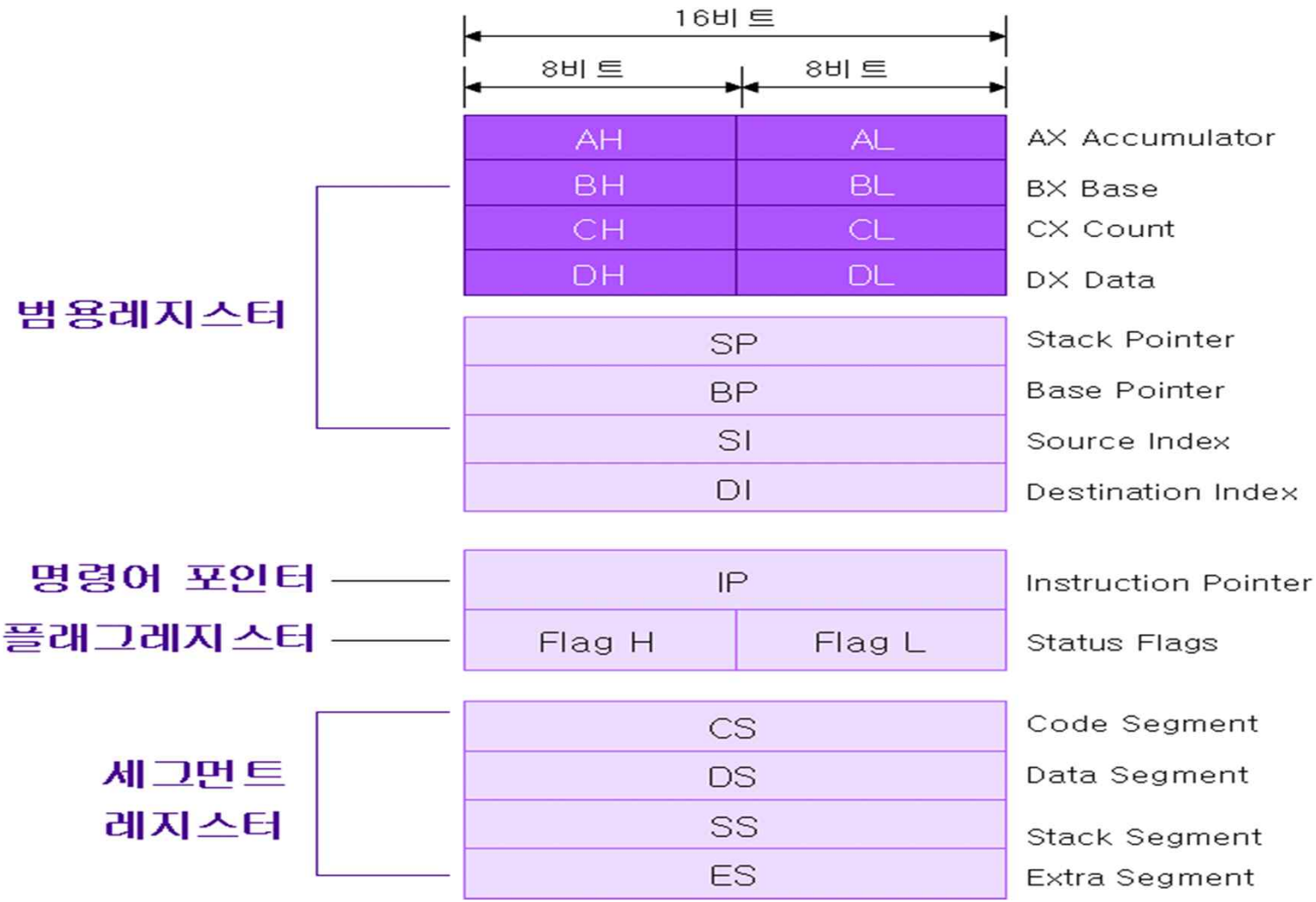
1. 레지스터에 대한 이해

1) 레지스터 종류와 역할

(1) 레지스터 개념

- 데이터를 빠르게 읽고 기록할 수 있는 기억장치
- 중앙처리장치 내에 있는 임시 기억장치
- 명령어의 임시보관이나 데이터의 일시적인 중간 결과를 보관
- 8086에는 16비트 레지스터 14개 존재
  (4개의 데이터 레지스터, 2개의 인덱스 레지스터, 3개의 포인트 레지스터, 4개의 세그먼트 레지스터, 1개의 플래그 레지스터)
- 데이터 레지스터는 2개로 분할 사용 가능
- 포인터, 인덱스, 세그먼트 레지스터는 16비트로만 사용 가능
- 레지스터의 크기는 점점 증가하고 있으며 현재 펜티엄의 경우 64비트

2 레지스터 구성도


16비트
8비트
8비트
AH
AL
BH
BL
CH
CL
DH
DL
AX Accumulator
BX Base
CX Count
DX Data
범용레지스터
SP
BP
SI
DI
Stack Pointer
Base Pointer
Source Index
Destination Index
명령어 포인터
IP
Instruction Pointer
플래그레지스터
Flag H
Flag L
Status Flags
세그먼트
레지스터
CS
DS
SS
ES
Code Segment
Data Segment
Stack Segment
Extra Segment

레지스터의 종류

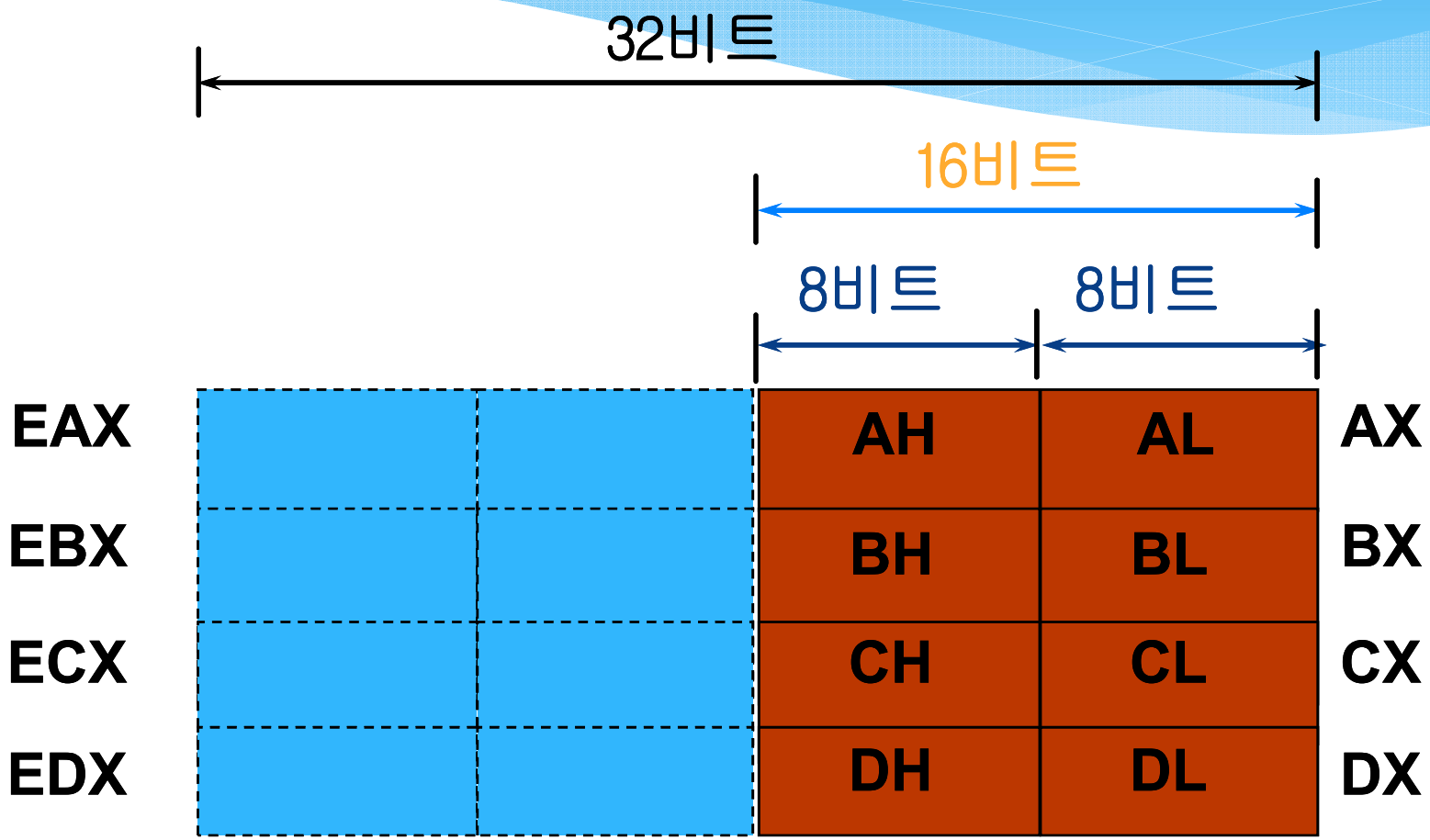
* 데이터 레지스터(4개)
  - 일시적인 결과 기록
* 포인터 레지스터(3개)
  - 스택 포인터와 베이스 포인터
* 인덱스 레지스터(2개)
  - 데이터의 주소 저장
* 플래그 레지스터(1개)
  - 연산 결과의 정보 저장
* 세그먼트 레지스터(4개)
  - 세그먼트의 시작주소 저장

데이터 레지스터

- 각종 데이터 처리시 일시적인 결과를 기록하기 위해 사용하는 16비트 레지스터 및 8비트 레지스터
- 프로그래머가 명령중에 자유롭게 지정 가능한 범용 레지스터
- 상위,하위 레지스터를 각각 엑세스 할 수 있다.

- AX(AL, AH) : Accumulator
  - 산술 논리 연산의 결과를 저장
- BX(BL, BH) : base
  - 간접 주소 지정시 베이스 주소를 가리키는 레지스터로 사용
- CX(CL, CH) : Counter
  - 스트링이나 루프에서 반복 횟수를 카운트하는 레지스터
- DX(DL, DH) : 데이터
  - 입출력 주소 저장에 사용되며 상위 워드용 데이터 레지스터로 사용되는 보조 어큐물레이터

데이터 레지스터


32비트
16비트
8비트
8비트
EAX
EBX
ECX
EDX
AH
AL
BH
BL
CH
CL
DH
DL
AX
BX
CX
DX

포인터와 인덱스 레지스터

1) 포인터 레지스터

- 스택 포인터(**SP**) : 스택 조작을 위해서 사용하는 레지스터로 프로그램 실행 중에 데이터의 저장 주소를 기억하고 있는 레지스터
- 베이스 레지스터(**BP**) : 기본적으로 스택 영역내의 주소를 지시하지만 스택 세그먼트 **SS**영역 내에 배치한 데이터에 대한 베이스 주소

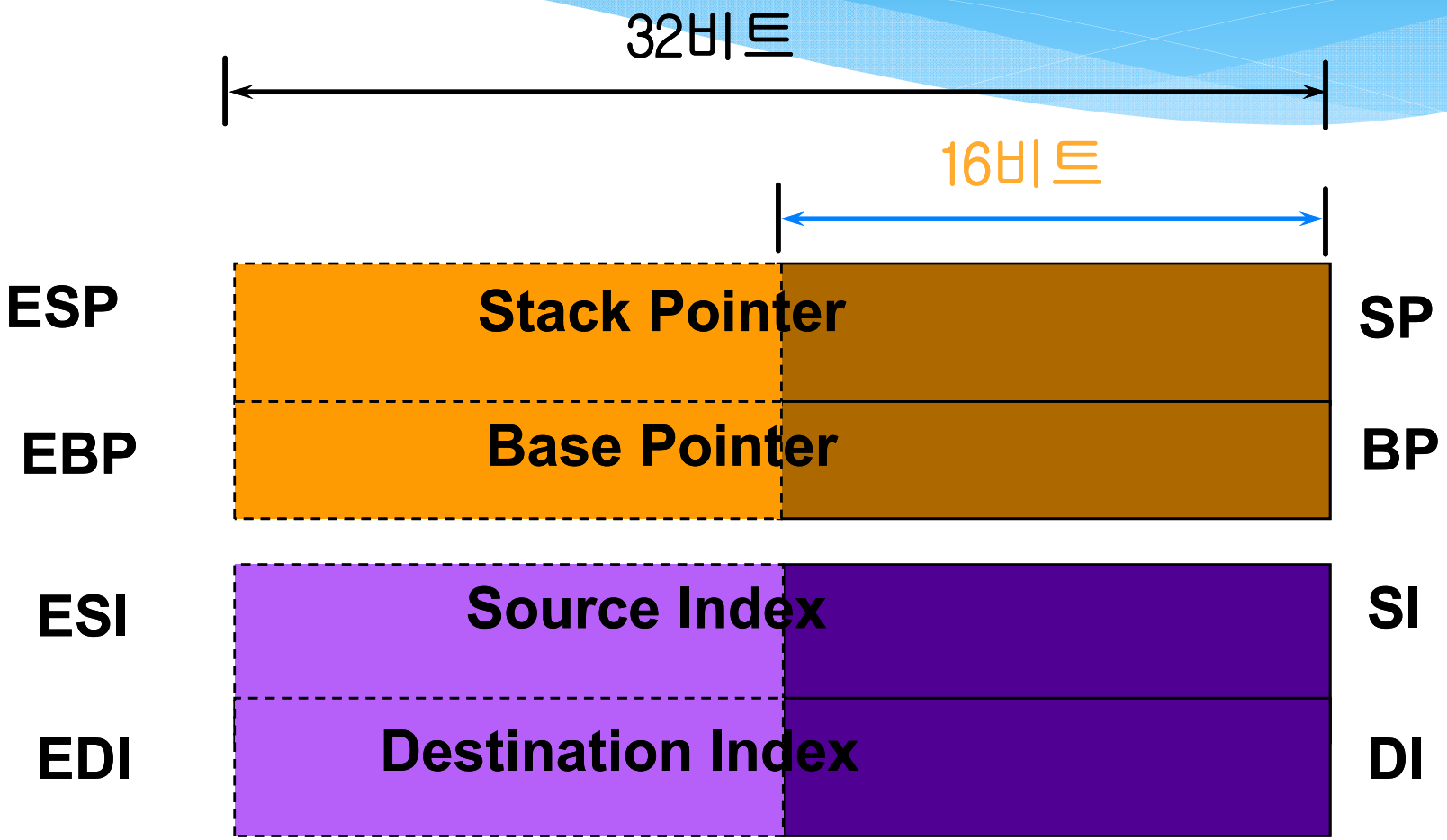
포인터와 인덱스 레지스터

2) 인덱스 레지스터

- 데이터의 주소를 나타내기 위해 사용하며 두가지 레지스터 **SI, DI**가 있다.

  - **SI(Source index register)**
    - 연산 항 소스, 소스 데이터 지정에 사용
  - **DI(Destination index register)**
    - 연산항의 처리 대상, 또는 목적 항 데이터를 나타내는데 사용

포인터와 인덱스 레지스터

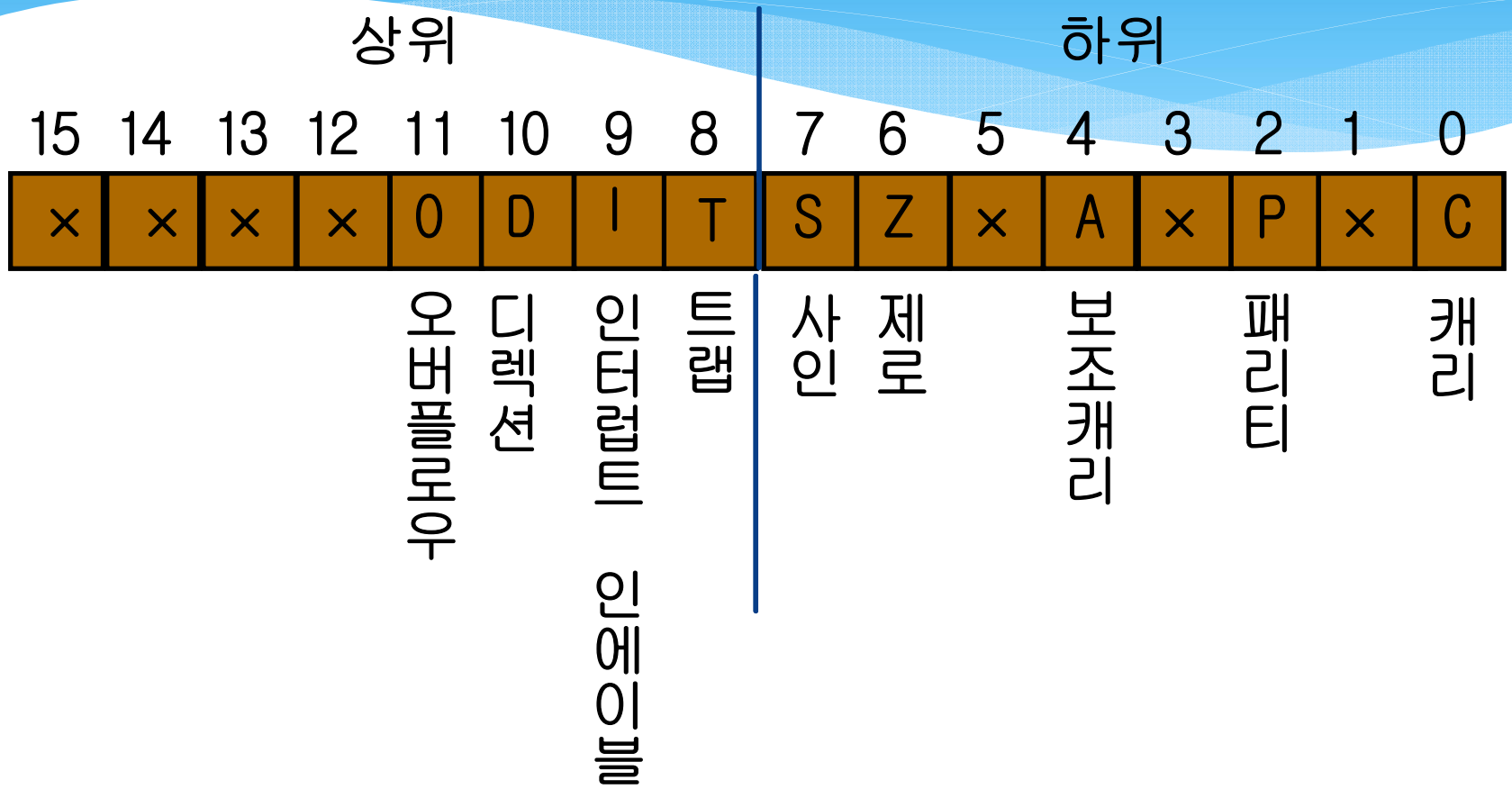


플래그 레지스터의 구성

플래그 레지스터

- 프로그램을 실행 할때 앞의 연산 결과가 음수인지 양수인지, 컴퓨터 사용자가 일반인지 관리자인지등의 정보를 16비트 크기의 상태 레지스터에 기록되고 이것에 기록되는 각 비트글을 플래그라고 부른다.
- 8086에서는 9종류의 플래그가 있다
- **CF, PF, AF, ZF, SF, OF :** 수치 연산과 논리 연산의 결과에 의한 CPU 의 상태를 나타내는 상태 플래그
- **TF, IF, DF ; CPU**동작을 변화하기 위한 제어 플래그

플래그 레지스터의 구성


상위
하위
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
× × × × O D I T S Z × A × P × C
오버플로우
디렉션
인터럽트 인에이블
트랩
사인
제로
보조캐리
패리티
캐리

세그먼트 레지스터

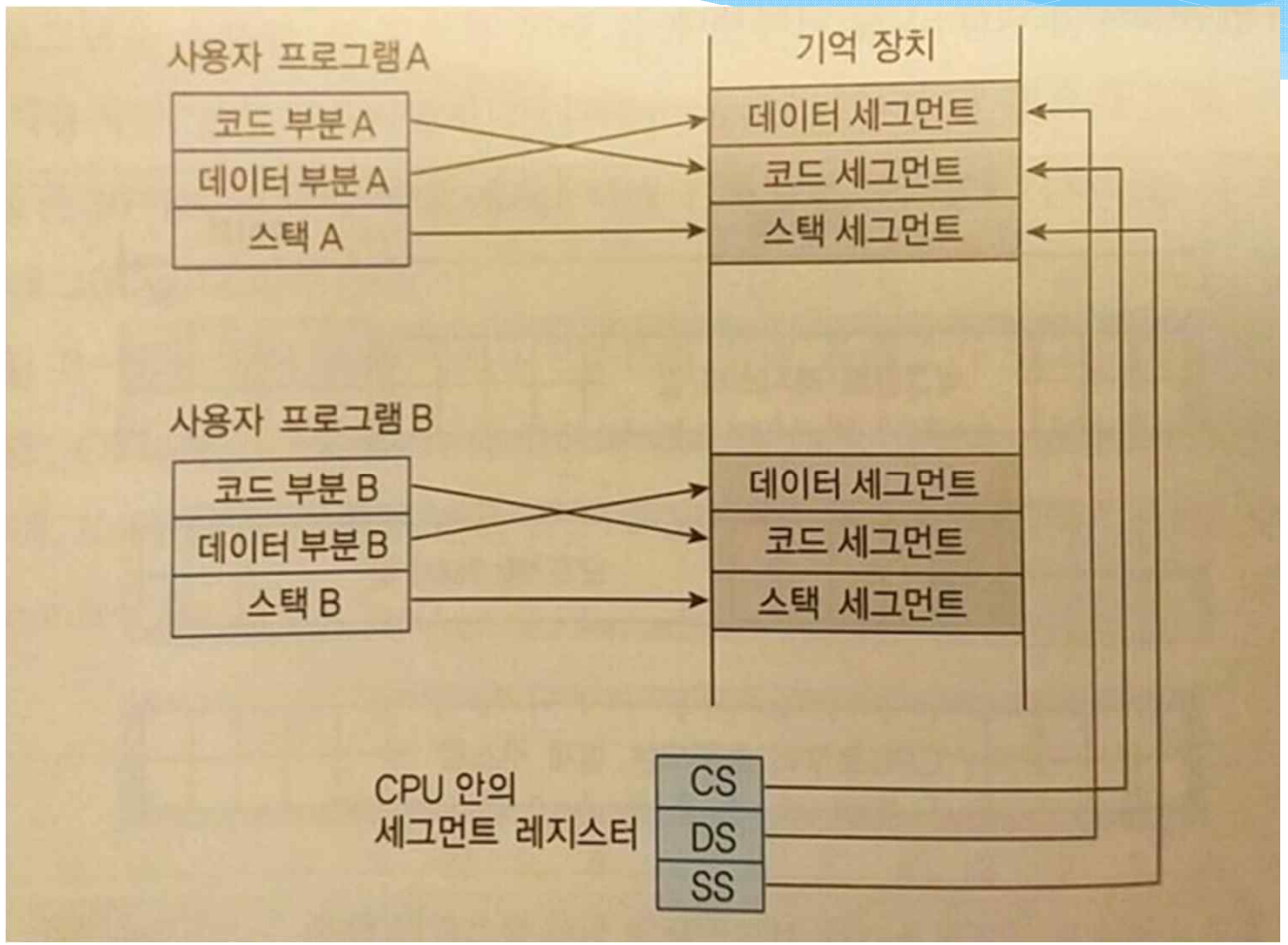
1).세그먼트 레지스터

- **CS**(코드세그먼트) : 프로그램의 명령 코드를 포함하고 **CS**레지스터는 이 세그먼트의 선두
- **DS**(데이터세그먼트) : 프로그램을 위한 데이터를 저장하고 있으며 **DS**레지스터는 이 세그먼트의 선두 주소를 가지고 있다.
- **ES**(엑스트라 세그먼트) : 여분의 데이터 세그먼트이며 주로 스트링 명령을 수행 할때 사용. **ES**레지스터는 이 세그먼트의 선두 주소
- **SS**(스택 세그먼트) : 인터럽트와 서브루틴의 반환 주소를 저장하는데 사용. **SS**레지스터는 스택 세그먼트의 선두 주소

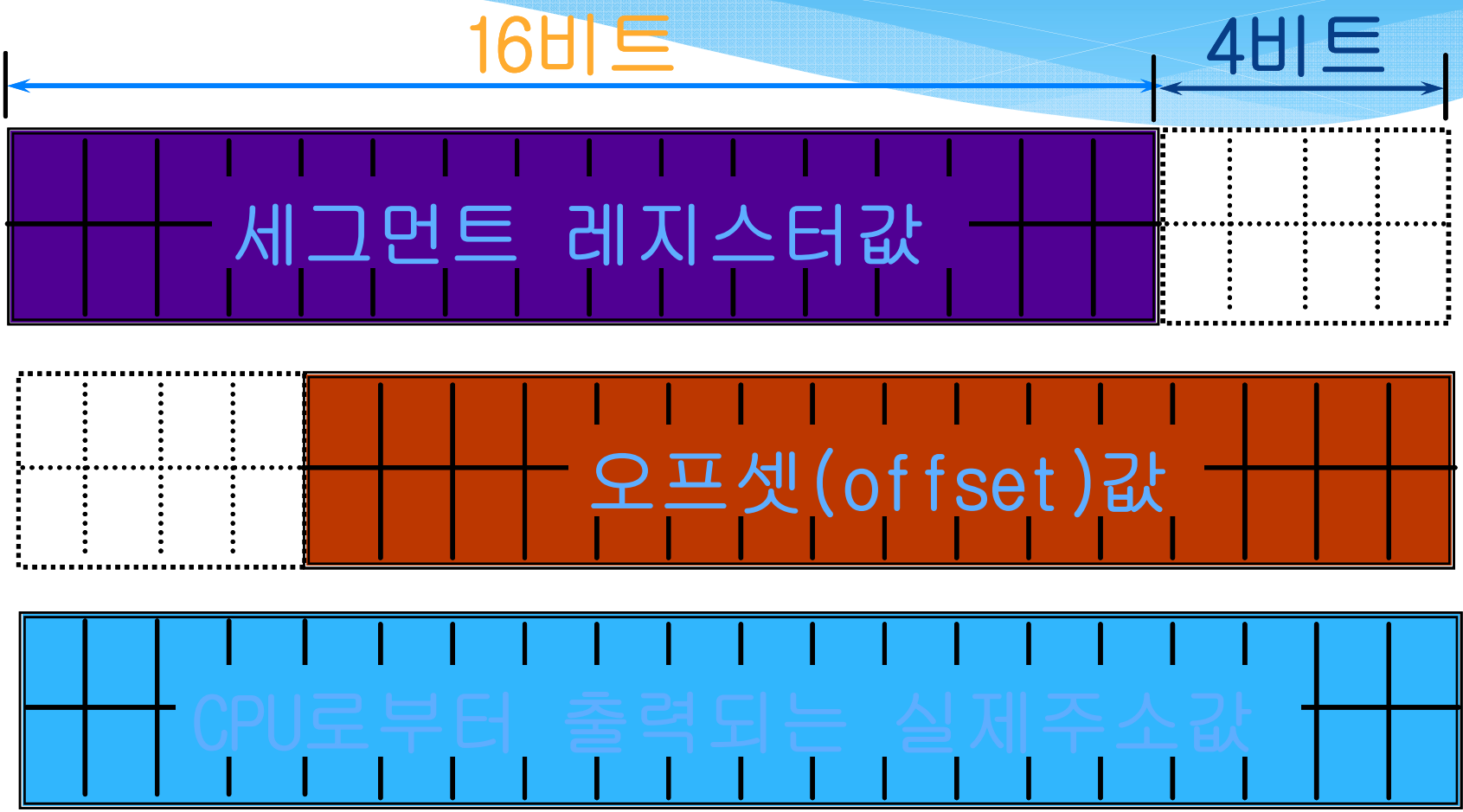
세그먼트 레지스터


Code Segment : CS
Data Segment : DS
Stack Segment : SS
Extra Segment : ES

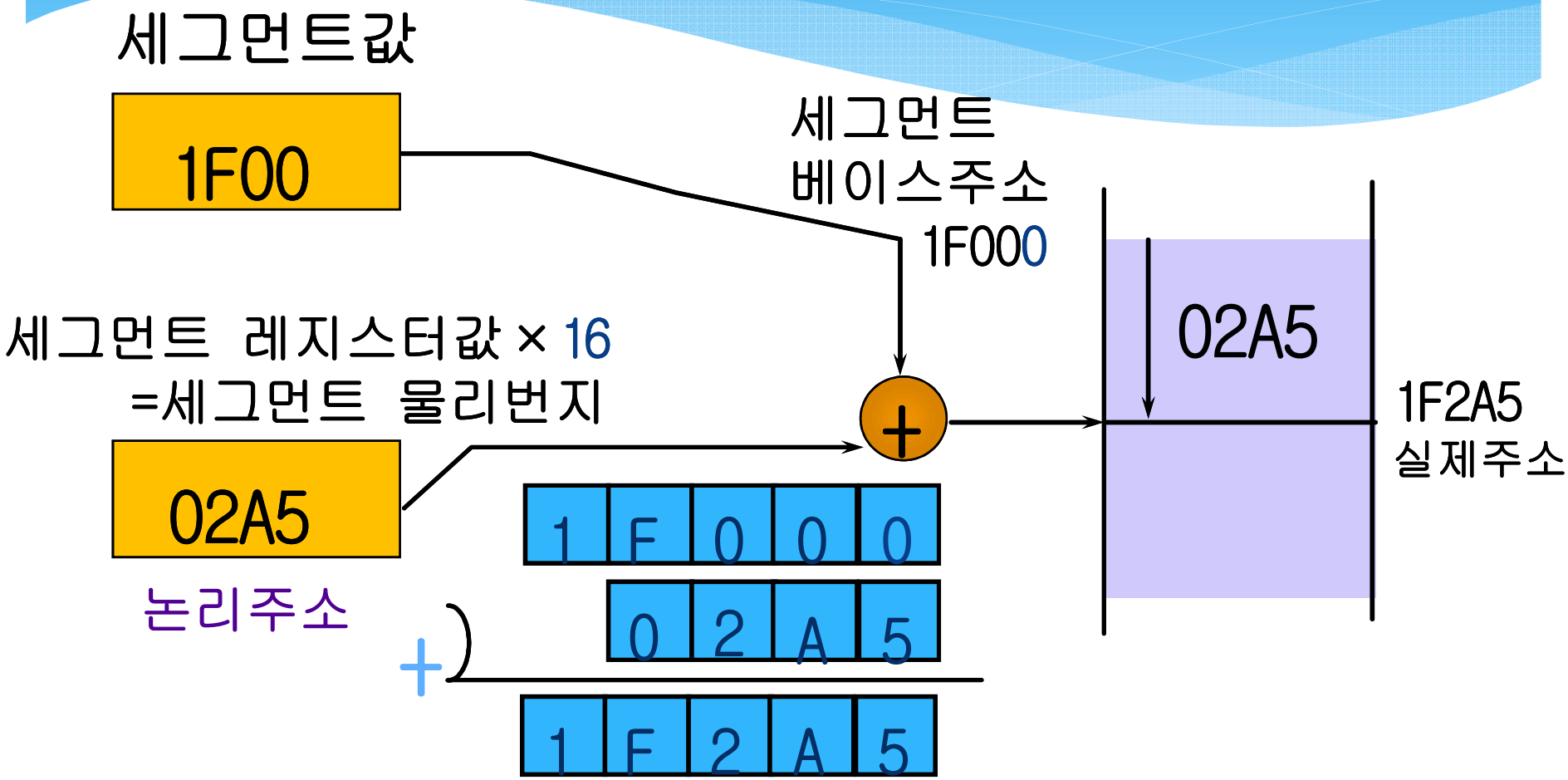
세그먼트 레지스터


사용자 프로그램A
코드 부분A
데이터 부분A
스택 A
기억 장치
데이터 세그먼트
코드 세그먼트
스택 세그먼트
사용자 프로그램B
코드 부분B
데이터 부분B
스택 B
데이터 세그먼트
코드 세그먼트
스택 세그먼트
CPU 안의
세그먼트 레지스터
CS
DS
SS

세그먼트레지스터와 물리주소

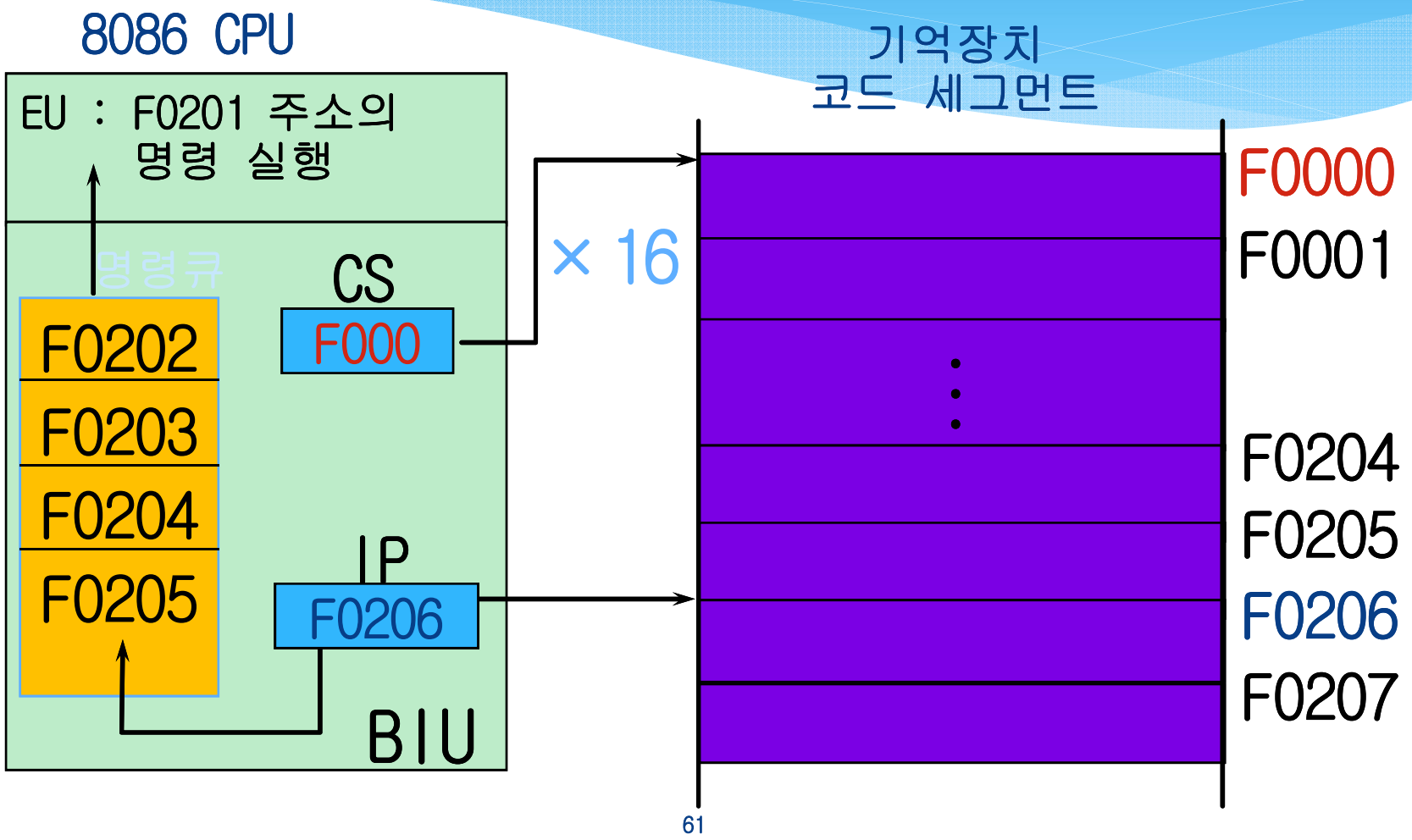


논리주소와 물리주소의 관계



명령어 포인터의 동작

명령어 포인터는 프로그램의 실행 순서를 조절하기 위해 명령어의 주소를 기억하는 레지스터로 CS와 쌍이 되어 실제 주소를 만듬



레지스터의 특성

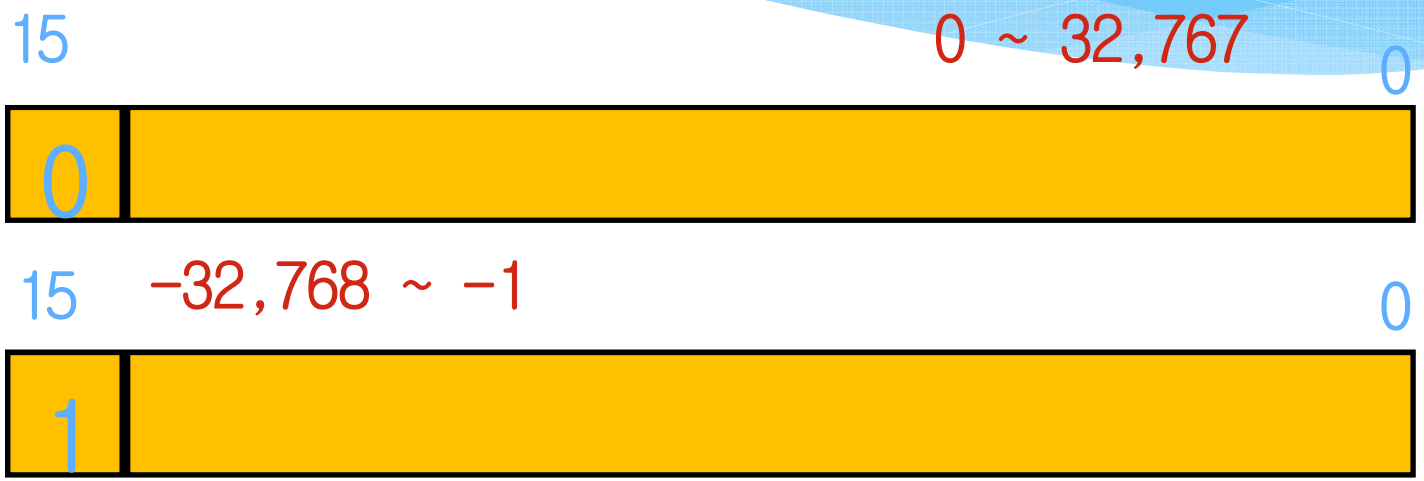
* 직접 엑세스 가능한 레지스터
  - AX, BX, CX, DX
  - AH, BH, CH, DH
  - AL, BL, CL, DL
  - SI, DI
  - DS, ES, SS
  - BP, SP

* 직접 수정이 가능한 플래그
  - CF, DF, IF

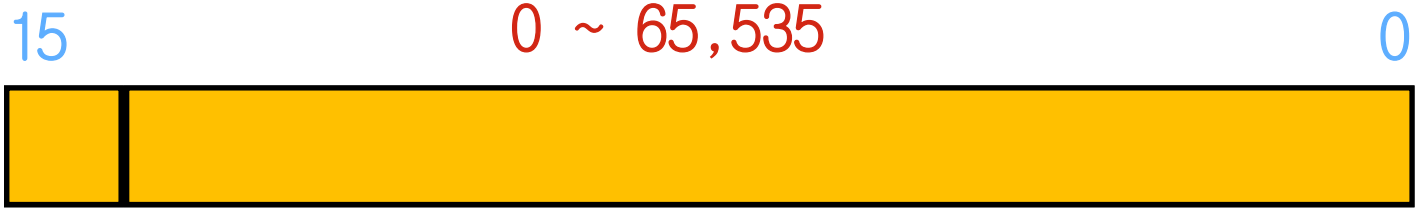
데이터의 표현

데이터의 표현

* 부호있는 숫자의 표현



- 부호없는 숫자의 표현



제 3 장 요약

* 32비트 인텔 프로세서 이해
* 8086 프로세서구조 이해, 명령어길이
* 명령어의 길이와 구성형태 이해
* 기억장치 모델과 구성형태 이해
* 주소지정과 데이터 기록을 이해 데이터, 포인터, 인덱스,세그먼트, 플래그레지스터 이해
* 논리주소는 물리주소로 변환됨
* 데이터 표현 시 부호비트 사용 유무

연습문제

연습문제

1. 레지스터의 역할은 무엇인가? 그리고 그 종류는 어떤 것이 있는가?
2. CS의 값이 1B00h이며, IP의 값은 00FFh이다. 이때 실제 물리주소 번지는 어디인지 구하시오.
3. 8비트 데이터가 다음과 같이 저장되어 있다.

   10100111

   이것을 부호 있는 정수(signed)와 양의 정수(unsigned)로 해석할 때 어떻게 되는가?

연습문제

4.다음 8비트에 데이터가 저장되어 있다.

01000110

1)이 데이터를 부호있는 정수(signed)로 해석하라

2)이 데이터를 양의정수(unsigned)로 해석하라

3)이 데이터를 문자로 해석하라(ASCII표 참조)

5.다음과 같은 조건을 만족하는 명령어를 만들고자 한다. 명령어의 길이가 최소한 몇 비트가 되어야 하겠는가?

1)명령어 코드의 종류 64개

2)주소 지정 방식 8개

3)레지스터 16개

4)데이터 크기 코드 3개

연습문제

6. 명령어를 이용하여 연산한 결과가 음수인지 양수인지 알고 싶다. 어떻게 하면 되겠는가?
7. 명령어에 데이터 크기 코드를 표시하는 이유는 무엇인가?
8. 스택 포인터의 역할은 무엇인가?