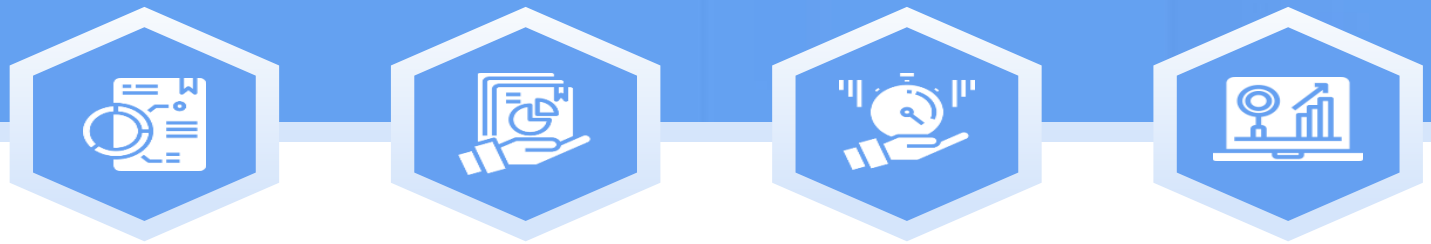
시스템 보안

#4 80x86 시스템 - 1



KOREA UNIVERSITY
SEJONG CAMPUS

- 목차

- 8bit 시스템
- x86? x64?
- x86 시스템의 구조

8bit 시스템

8bit CPU

- **8bit CPU 시대의 일체형 개인용 컴퓨터**
  - Apple II: 1977년 Apple에서 만든 세계 최초의 일체형 개인용 컴퓨터
    가장 초창기의 개인용 컴퓨터 중 하나이자 가장 성공한 제품으로 1970년대 말부터 80년대 전반에 걸쳐 개인용 컴퓨터 붐을 이끈 주역
    1976년에 만든 Apple I이 반제품 상태로 판매됐던 관계로 완제품으로 나온 것은 Apple II가 최초

- 1980년대에 나온 후발 주자인 코모도어 64, MSX 등과 비교하면 하드웨어 성능이 떨어지는 편이었지만 성공한 선발 주자였기 때문에 소프트웨어의 양과 질은 압도적이었으며 이를 밑천으로 떨어지는 성능에도 불구하고 경쟁 기종들에 뒤쳐지지 않는 수명을 자랑

- Apple 내부에서도 Apple II의 뒤처지는 성능을 인지하고 있어 여러 차례 업그레이드 모델을 내놓기도 했지만 1986년에 등장한 Apple IIGS를 마지막으로 1980년대 중반부터는 Macintosh 시리즈로 주력 사업이 바뀌면서 우선 순위가 밀려나게 됨

8bit CPU

- ## 8bit CPU 시대의 인텔(Intel)

  - 1974년 4월 인텔에서 출시한 8비트 마이크로프로세서

    1969년부터 인텔 4004 설계를 도운 시마 마사토시가 다니던 비지콤을 그만두고 1972년 인텔로 이직하면서 인텔 8080 설계를 주도

Intel 8080

Zilog Z80

- 인텔 8080은 컴퓨터 역사에서 가장 중요한 마이크로프로세서 중의 하나로 1980년대에 만든 8비트 기기들에 들어간 CPU들 중 가장 널리 사용된 Z80도 인텔 8080을 기반하고 있으며 당시 가장 널리 사용했던 8비트 운영 체제인 CP/M도 원래 인텔 8080 프로세서를 위해 제작

- 1976년에 출시된 CPU로 인텔 8080과 그 주변칩들을 설계한 시마 마사토시 등 11명의 인텔 직원이 1974년 인텔을 떠나 자일로그를 설립해 만들었으며, 8080에 기반한 CPU로 인텔 8080에 비해 레지스터, 명령어 추가, 높은 클럭 지원 등의 성능 향상을 했으며 1980년대 수 많은 PC와 게임기 등에서 사용되면서 가장 많은 8비트 기기에 사용된 CPU

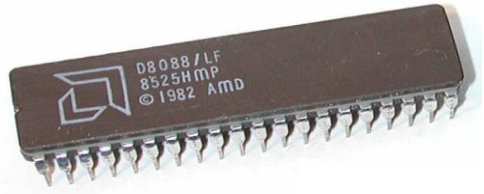
x86? x64?

x86? x64?

- ## x86(=80x86) 시스템이란?

  - x86(=80x86) 이라는 뜻은 인텔(Intel)의 CPU 시리즈 이름이자 그 CPU의 명령 체계 아키텍처 이름
    8086 CPU는 최초의 16bit CPU이자 이후 인텔의 후속모델로 연결되는 x86 아키텍처 역사의 시작
  - 80286 까지는 16bit CPU로 x86-IA16으로 명명(IA라는 약자는 Intel Architecture)

8086



8088

80286

- 1978년 인텔(Intel)이 최초의 16bit CPU인 8086 을 발표했고, 다음 해에 좀 더 저렴한 8088 CPU를 발표했는데, 미국 IBM社에서 공개형 표준 PC(Personal Computer)인 IBM-PC XT 의 메인 CPU로 인텔의 8088을 채택하면서 본격적인 PC의 역사가 시작

- 8086과 8088은 16비트 CPU였고 이때 처음 설계한 CPU 내부 8086 명령어 체계를 x86 아키텍처(x86-IA16)라고 부름
  IA-16 에서 IA라는 약자는 Intel Architecture 를 뜻하고 16은 16비트 라는 의미,
  이후 1982년에 차기 버전 80286 CPU를 출시하면서도 x86 아키텍처는 계속 호환 확장

※ 참고로 80268 CPU는 IBM의 2세대 개인용 컴퓨터인 IBM-PC AT에 사용됨

x86? x64?

- **x86 시스템 – 32bit CPU(x86-IA32)**

  - 80386 부터 32bit CPU로 일반적으로 386 PC라고 불렀으며, 80386 CPU 및 경쟁사 호환 CPU들을 통틀어 x86 아키텍처 CPU라고 부름

80386

- 1985년 인텔은 32비트 CPU인 80386을 발표했는데 80386 은 기존 16bit 인 체계와의 호환성을 버리고 32bit CPU 명령 세트 및 기초를 새롭게 만들어서 x86-IA32 (Intel Architecture-32) 라고 공식 명명하였으며, 일반 사용자들이 흔히 386 PC 라고 부르는 컴퓨터에 사용된 CPU의 탄생

- 그 후에 발표되는 인텔의 모든 후속 32bit CPU들 - 인텔 80486, 펜티엄(80586), 펜티엄 프로, 셀러론 등 뿐 아니라 경쟁사 CPU(AMD, Cyrix 등) 모두 역시 인텔의 x86-IA32 명령 체계와 호환되는 CPU를 제작

  ※ 참고로 8086, 80286, 80386, 80486 까지는 숫자로 부르다가, 숫자로 이루어진 제품명은 특허로 인정할 수 없다는 소송으로 인해 그 이후부터는 펜티엄 처럼 단어 호칭으로 바뀌게 됨

x86? x64?

- **x86 시스템 – 32bit CPU(x86-IA32)**
  - x86은 16bit CPU 부터 시작했지만 대중적으로 가장 잘 알려진 32bit CPU를 x86 시스템으로 부르게 됨

80386

- x86 아키텍처는 과거 x86-IA16 16비트 버전과 x86-IA32 32비트 버전이라고 뒷부분 IA16과 IA32를 구분하여 불러야 하겠지만, x86-IA16 (8088, 80286 16비트 CPU) 시절에는 PC라는 전자기기 자체가 회사, 학교 등 특수목적으로 사용되었기 때문에 일반 가정에는 PC를 보유한 사람이 거의 없었고

- 80386 CPU(x86-IA32 적용)가 출시된 후에야 점차 가정에 PC가 보급되기 시작하면서 사용자들에게 인지도가 생기기 시작해서 90년대 초부터 2004년 정도까지 15년~20년 이라는 오랜 시간 동안 32비트 CPU인 x86-IA32가 사용되었으므로 x86이라고 하면 16비트 시절은 무시하고 그냥 32비트 CPU의 대표명사처럼 불리게 됨

x86? x64?

- **x64 시스템 – 64bit CPU(IA64)**
  - 인텔은 차세대 64bit CPU에 대한 개발을 계속 진행하다가 최종적으로 기존 32bit x86 호환 체계를 버리고 새로운 설계를 하기로 결정했으며, 새로운 64bit CPU 구조를 IA-64 (Intel Architecture-64)라고 명명하고 아이태니엄(Itanium) CPU를 출시

Itanium

- x86이라는 시리즈 꼬리표를 버리고 기존 32bit 80386 CPU의 x86-IA32와 호환되지 않는 완전히 새로운 64비트 CPU 명령 세트 구조를 설계하여 IA-64 (Intel Architecture-64)라고 명명했으며 HP 휴렛팩커드와 함께 공동개발에 착수하여 IA-64 아키텍처를 적용한 최초 64비트 CPU로서 아이태니엄(Itanium)을 개발하여 2001년에 공식 발표

- 하지만 기존의 x86-IA32 즉 x86 32비트 아키텍처 호환성을 버렸기 때문에 운영체계를 포함한 모든 소프트웨어가 호환되지 않아 새로 다시 개발해야 했고, 생각보다 느린 성능과 비싼 가격 등의 이유로 시장에서 외면 받음

x86? x64?

- **x64 시스템 – 64bit CPU(IA64)**
  - Microsoft가 아이태니엄 아키텍처인 64비트 IA-64를 그냥 간편하게 x64 라고 부르자고 제안하면서 64bit CPU를 x64로 부르기 시작

Itanium

- 인텔은 개인 PC 시장이 아니라 고성능 컴퓨터가 필요한 기업용 Enterprise 서버 시장에 관심이 많았고, 따라서 IA-64 CPU를 기업용 유닉스(UNIX) 서버 장비, 즉 HP PA-RISC, SUN Spark, IBM Power CPU 등에 대항하기 위한 고성능 서버용 CPU로서 마케팅을 진행했기 때문에 일반 가정용, 사무용 PC 시장에서는 외면을 받음

- 이 즈음에 Microsoft가 아이태니엄 아키텍처인 64비트 IA-64를 그냥 간편하게 x64 라고 부르자고 제안하면서 Windows OS버전을 발표할 때 Windows XP Professional x64 Edition, Windows Server 2003 x64 Edition 처럼 x64라는 단어를 사용하기 시작

x86? x64?

- **x64 시스템 – 64bit CPU(x86-64, AMD64)**
  - AMD의 반격 : 32비트 호환되는 64비트 CPU x86-64, AMD64

Opteron

Athlon

- AMD社는 1999년에 기존의 인텔 32비트 x86 IA-32와 호환되는 확장 64비트 명령 세트를 설계하여 x86-64로 발표했으며, 몇 년 후인 2003년에 해당 x86-64 명령어 세트를 적용한 K8 마이크로아키텍쳐 CPU 설계를 완성하고 64비트 CPU인 옵테론(서버용)과, 애슬론64(가정용)를 발표
  (이름도 x86-64 에서 AMD64 라고 변경)

- INTEL이 기존 32bit 호환성을 포기하고 64bit로 전환한 것에 비해, AMD의 x86-64 CPU들은 기존의 32bit인 x86-IA32와 호환되면서 64bit 처리가 가능하므로 폭발적으로 관심을 끌게 됨

- 당시 MS의 windows OS는 IA64 CPU만 지원하고 AMD64 CPU의 지원은 늦어지고 있었지만, 리눅스가 한발 먼저 AMD64의 64비트를 전격 지원하면서 리눅스가 빠르게 확산되기 시작

x86? x64?

- **x64 시스템 – 64bit CPU(x86-64, AMD64)**
  - AMD의 반격 : 32비트 호환되는 64비트 CPU x86-64, AMD64

Prescott

Core i7

- 기존 x86 32bit 체계와 호환되는 x86-64(AMD64) CPU를 사용하게 되면 기존의 32bit 프로그램을 완전 재개발할 필요 없이 컴파일만 64bit로 다시 해주면 바로 사용 가능했고, 동시에 메모리 처리는 64비트로 해결할 수 있으니 32bit 의 대표적인 문제점인 4GB 메모리 제약(2^32제곱=4,294,967,296) 한계가 단번에 해결되면서 소프트웨어 제작사들은 앞다투어 64bit 버전 소프트웨어를 발표한데다 가격도 무료인 리눅스 OS가 기업용 시장에서 인기를 끌게 됨

- 인텔은 기존 x86-IA32 과 호환되지 않는 새로운 64bit인 IA-64를 고집했고 x86-64를 계속 외면했지만, AMD의 제품이 너무 폭발적 인기를 끌자 결국에는 다음 해인 2004년에 경쟁사인 AMD사와 라이선스를 협약해서 x86-64 INTEL64 라는 똑같은 아키텍처를 선보이고 프레스캇(Prescott) CPU를 발표 즉, x86-64의 원천기술은 AMD가 개발하여 가지고 있는 것이고, INTEL64는 AMD64의 카피본

- 이후에 출시한 AMD사의 애슬론, 셈프론, 페넘, 카바리, A시리즈 등 및 인텔의 셀러론D, 펜티엄D, 코어 i3, i5, i7 등은 모두 x86-64 (AMD64, INTEL64)에 기반하는 CPU

x86? x64?

- ## x86과 x64 시스템

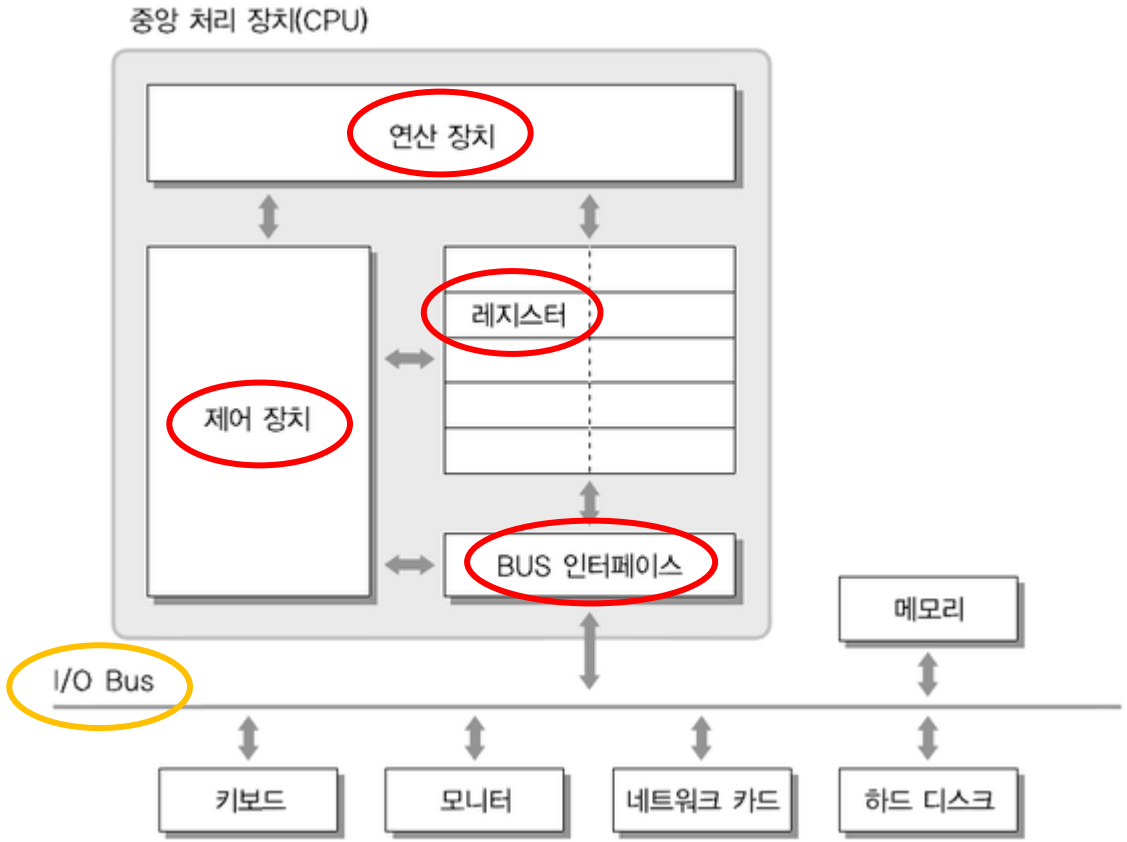
  - INTEL 이 만든 x86은 CPU의 내부 명령 체계 아키텍처를 뜻하고, x86 아키텍처는 계속 진화하면서 16bit에서 32bit를 거쳐 64bit 로 발전
  - x86 하위에는 x86-IA32(32bit) 와 x86-64 (64bit) 두가지가 있지만, 32bit가 오랜 기간 사용되었기에 x86은 통상 32bit CPU로 인식
  - INTEL에서 64bit CPU 전용 아키텍처인 IA-64를 만들었고 x64라고 명명
  - INTEL의 32bit 전용 x86-IA32 가 있고, AMD에서 개발한 32bit/64bit 동시지원하는 x86-64(AMD64)
  - 현재 출시하는 거의 모든 PC CPU는 x86-64(AMD64)로 제작하므로 64bit(32bit 하위호환 가능)
  - x64는 원래 인텔의 IA-64 64bit CPU를 말하는 것이었지만 시장 점유에 실패한 CPU가 되었고,
    이후 AMD의 x86-64가 대히트를 치면서 x64라고 하면 64bit CPU를 의미하며 통상 x86-64(AMD64)를 지칭

x86 시스템의 구조

x86 시스템의 구조

- **x86 시스템 구조**

  - x86은 통상 32bit CPU(Central Processing Unit)를 의미



- **CPU의 네 가지 기본 구성 요소**
  - 연산 장치(ALU)
  - 제어 장치(Control Unit)
  - 레지스터(Register)
  - 버스 인터페이스(Bus Interface)

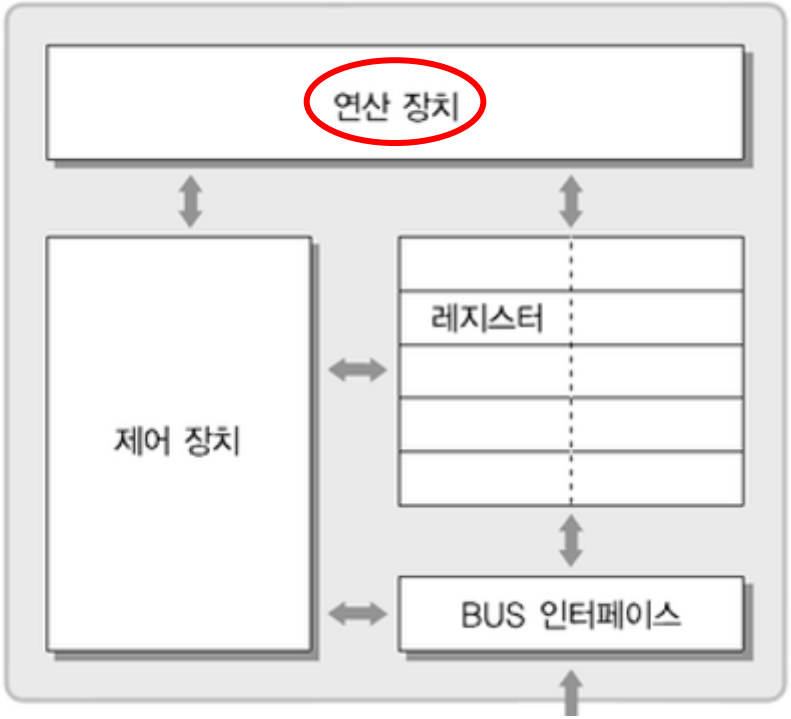
- **CPU의 동작**
  - Fetch : 명령어 가져오기
  - Decode : 가져온 명령어 해독하기
  - Execute : 해독한 명령어를 실행
  - Write : 실행한 명령대로(필요하다면) 메모리에 쓰기

x86 시스템의 구조

- ## x86 시스템 구조 - CPU

  - 연산 장치(산술논리연산장치, ALU, Arithmetic Logic Unit) : 제어 장치의 명령에 따라 실제로 연산을 수행하는 장치
    산술연산(+ , - , × , ÷), 논리연산(AND, OR, NOT, XOR), 관계연산, 비트 이동(Shift) 등을 수행
    연산장치는 연산에 필요한 데이터를 레지스터에서 가져오고, 연산 결과를 다시 레지스터로 보내 저장

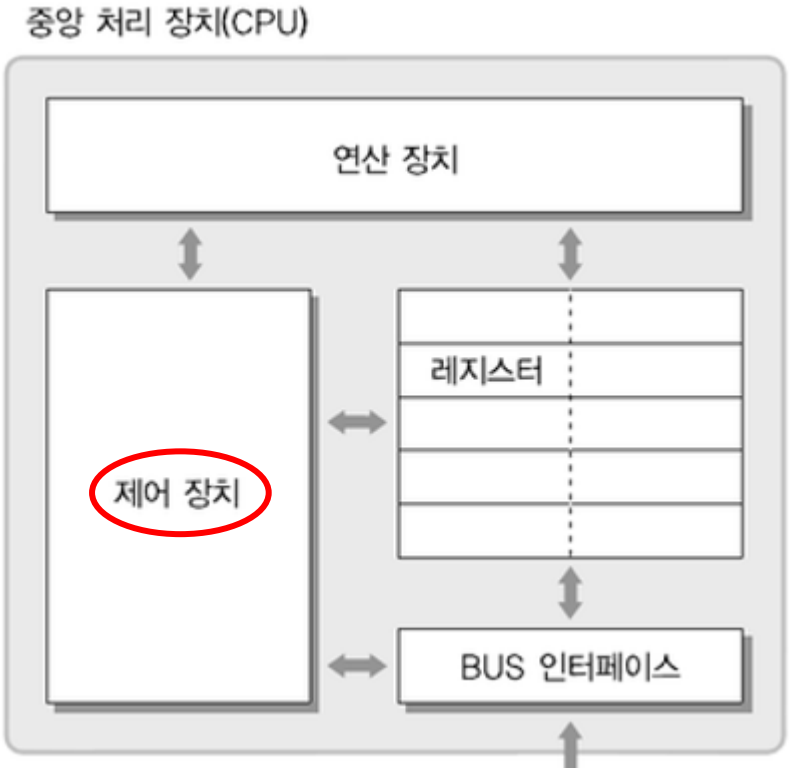
중앙 처리 장치(CPU)



구성 요소		기능
내부 장치	가산기(Adder)	덧셈 연산 수행
	보수기(Complementor)	뺄셈 연산 수행, 1의 보수나 2의 보수 방식 이용
	시프터(Shifter)	비트를 오른쪽이나 왼쪽으로 이동, 나눗셈과 곱셈 연산 수행
관련 레지스터	누산기(Accumulator)	연산의 중간 결과 저장
	데이터 레지스터(Data Register)	연산에 사용할 데이터 저장
	상태 레지스터(Status Register)	연산 실행 결과로 나타나는 양수/음수, 자리 올림, 오버플로우의 상태 기억

x86 시스템의 구조

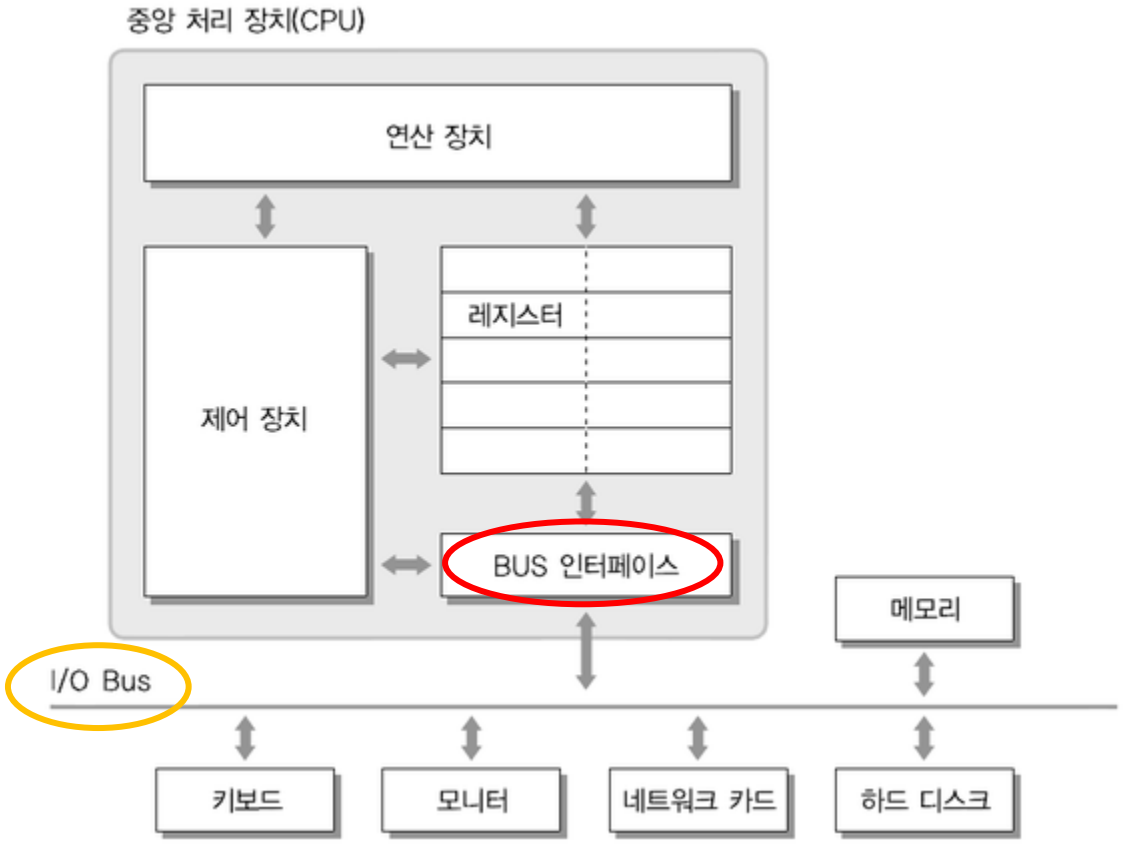
- **x86 시스템 구조 - CPU**
  - 제어 장치(CU, Control Unit) : 명령어를 순서대로 실행할 수 있도록 제어하는 장치
    주 기억장치에서 명령어를 꺼내 해독한 다음, 해독한 결과에 따라 명령어 실행에 필요한 제어 신호를 기억장치, 연산장치, 입출력장치로 전송
    또한 이들 장치가 보낸 신호를 받아 다음에 수행할 동작을 결정



구성 요소		기능
내부 장치	명령 해독기(Instruction Decoder)	명령 레지스터에 있는 명령을 해독하여 부호기로 전송
	부호기(Decoder)	명령 해독기가 전송한 명령을 신호로 만들어 각 장치 전송
	주소 해독기(Address Decoder)	명령 레지스터에 있는 주소를 해독하여 메모리의 실제주소로 변환한 후, 이를 데이터 레지스터에 저장
관련 레지스터	프로그램 카운터(Program Counter)	다음에 실행할 명령의 주소 저장
	명령 레지스터(Instruction Register)	현재 실행 중인 명령 저장
	메모리 주소 레지스터 (Memory Address Register)	주 기억 장치의 번지 저장
	메모리 버퍼 레지스터 (Memory Buffer Register)	메모리 주소 레지스터에 저장된 주소의 실제 내용 저장

x86 시스템의 구조

- **x86 시스템 구조 - CPU**
  - 버스 인터페이스(Bus Interface) : 외부의 입출력 버스와 연결되어 데이터를 주고 받는 역할을 담당
  - 입출력 버스(Input / Output Bus): 다른 구성요소의 사이에서 데이터를 교환하기 위해 거치는 통로



- **버스 인터페이스(Bus Interface)**

  CPU 외부의 입출력 버스(I/O Bus)와 데이터를 주고 받기 위해 필요한 인터페이스

- **입출력 버스(Input / Output Bus)**

  CPU와 메모리, 그리고 I/O 장치(키보드, 마우스 등..) 사이의 데이터 입출력을 담당

  전송되는 데이터에 따라

  번지 버스(Address Bus),

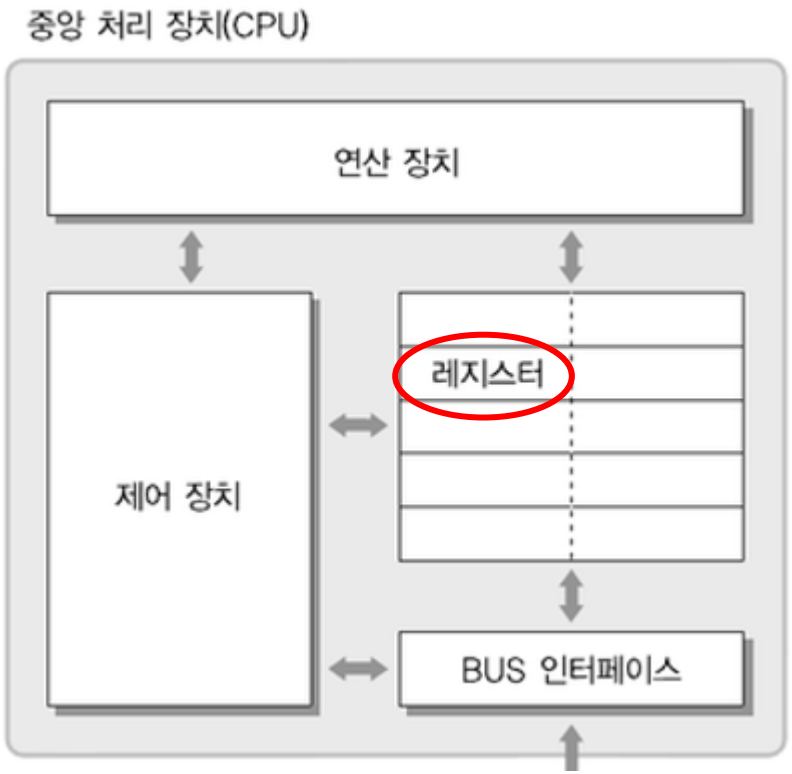
  자료 버스(Data Bus),

  제어 버스(Control Bus)로 분류

x86 시스템의 구조

- **x86 시스템 구조 - CPU**

  - 레지스터(Register) : 중앙처리장치의 속도와 비슷한 고속의 기억장치로 CPU에 따라서 사용할 수 있는 레지스터의 크기가 개수가 다름
    명령어 주소, 명령어 코드, 연산에 필요한 데이터, 연산 결과 등을 임시로 저장하며 용도에 따라 범용 레지스터와 특수 목적 레지스터로 구분



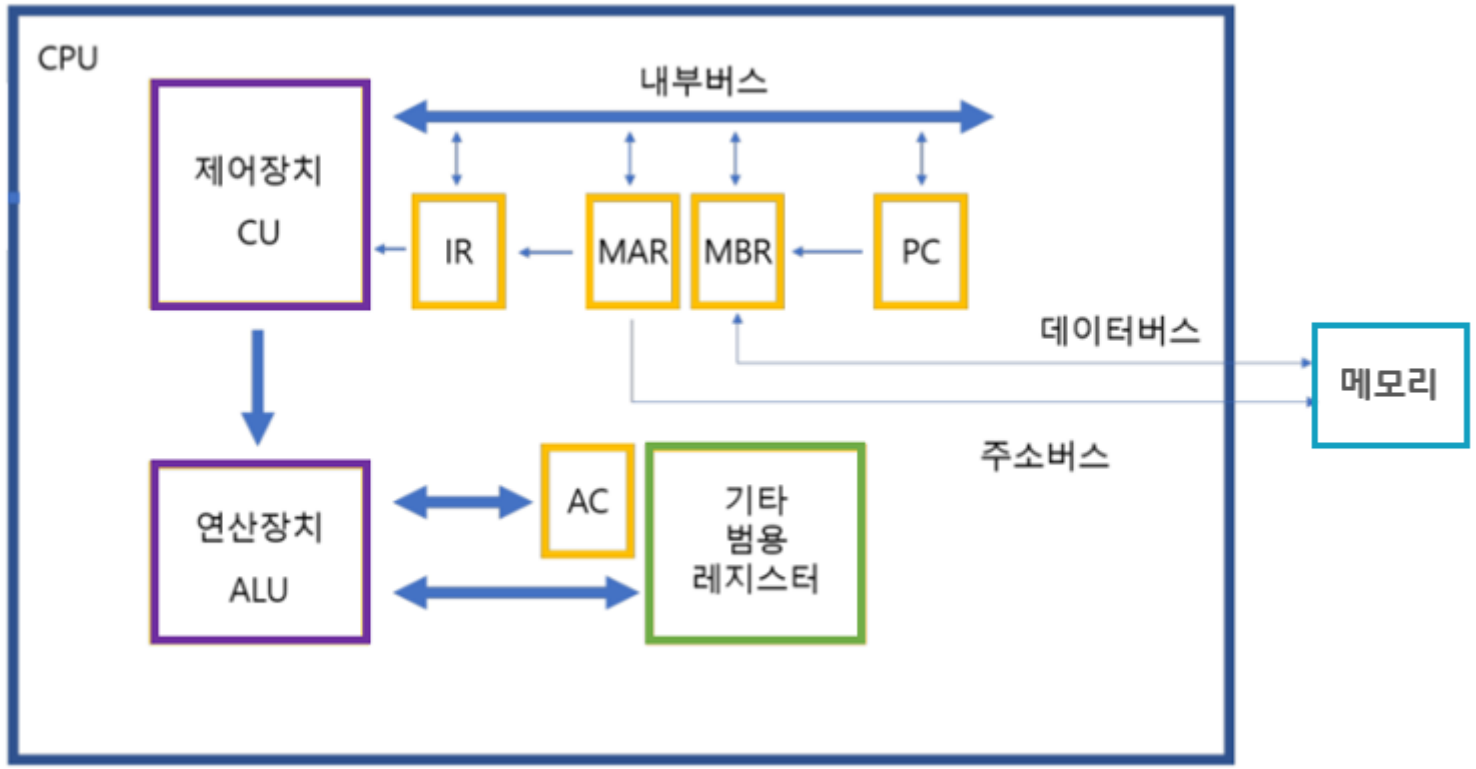
- 범용 레지스터 : 연산에 필요한 데이터나 연산 결과를 임시로 저장하는 레지스터

- 특수 목적 레지스터 : 특별한 용도로 사용하며, 용도와 기능에 따라 구분되는 레지스터
  AC(누산기) : 연산 결과 임시 저장
  PC(프로그램 카운터) : 다음에 수행할 명령어 주소 저장
  IR(명령어 레지스터) : 현재 실행 중인 명령어 저장
  MAR(메모리 주소 레지스터) : 읽기와 쓰기 연산을 수행할 주기억장치 주소 저장
  MBR(메모리 버퍼 레지스터) : 주 기억장치에서 읽어온 데이터나
  주 기억 장치에 저장할 데이터를 임시로 저장

x86 시스템의 구조

- **x86 시스템 구조 - CPU**

  - 레지스터(Register) 종류와 기능



- 범용 레지스터

  연산에 필요한 데이터나 연산 결과를 임시로 저장

- 특수 목적 레지스터

  AC(누산기) : 연산 결과 임시 저장

  PC(프로그램 카운터) : 다음에 수행할 명령어 주소 저장

  IR(명령어 레지스터) : 현재 실행 중인 명령어 저장

  MAR(메모리 주소 레지스터) :

  읽기와 쓰기 연산을 수행할 주기억장치 주소 저장

  MBR(메모리 버퍼 레지스터) :

  주 기억장치에서 읽어온 데이터나 주 기억 장치에 저장할 데이터를 임시로 저장

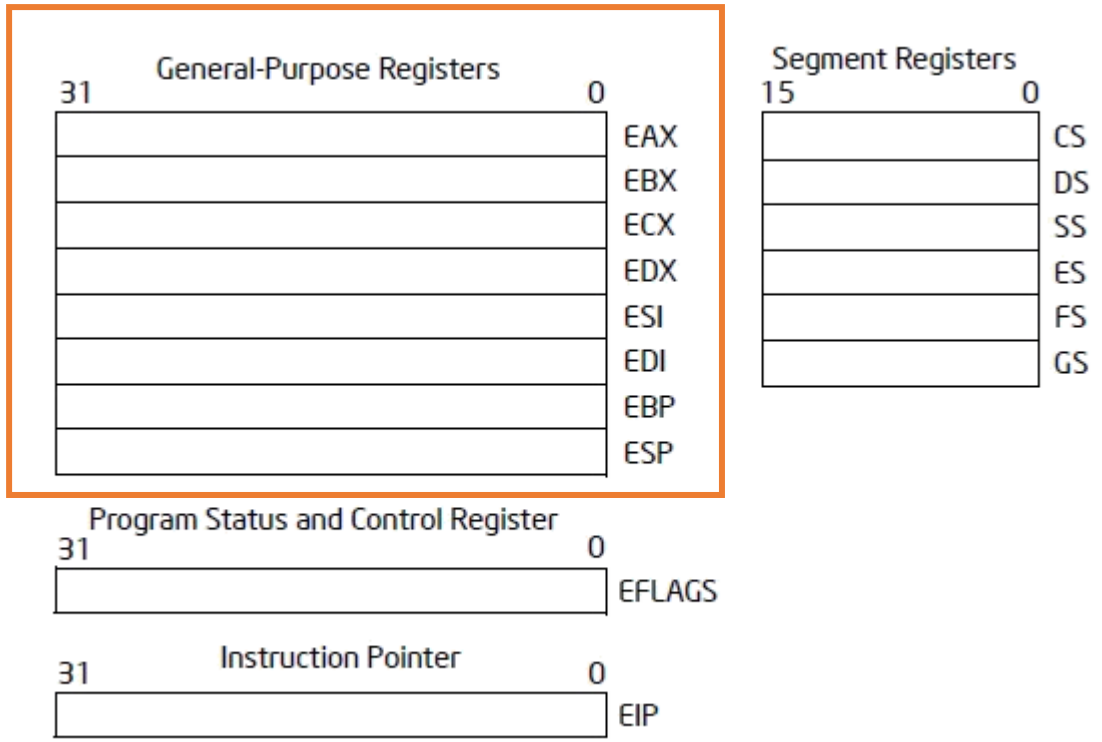
x86 시스템의 구조

- x86 시스템 구조 – CPU : 레지스터(Register) 종류와 기능
  - 사용 목적 별 레지스터 맵핑

구분		맵핑 레지스터
연산 장치 관련 레지스터	누산기(AC, Accumulator)	EAX
	데이터 레지스터(DR, Data Register)	ECX, EDX
	상태 레지스터(SR, Status Register)	EFLAGS
제어 장치 관련 레지스터	프로그램 카운터(PC, Program Counter)	EIP
	명령 레지스터(IR, Instruction Register)	현재 실행 중인 명령 저장
	메모리 주소 레지스터(MAR, Memory Address Register)	EBX, EBP, ESP, EDI, ESI 등
	메모리 버퍼 레지스터(MBR, Memory Buffer Register)	주 기억장치에서 읽어오거나 저장할 데이터를 임시로 저장

x86 시스템의 구조

- **x86 시스템 구조 – CPU : 레지스터(Register) 종류와 기능**
  - 범용 레지스터(General-Purpose Registers, General Register)



- x86 시스템에서 **범용 레지스터(General-Purpose Registers)**는
  EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP 8개로 구성되어 있고
  더 세부적으로는 구분하면
  범용 레지스터(General Register) : EAX, EBX, ECX, EDX
  인덱스 레지스터(Index Register) : ESI, EDI
  포인터 레지스터(Pointer Register) : EBP, ESP

- X64 시스템에서는 R8~R15까지의 레지스터가 추가
  기존 x86의 레지스터는 64비트를 나타낼 때 'E' -> 'R'로 변경 EAX -> RAX

x86 시스템의 구조

- **x86 시스템 구조 – CPU : 레지스터(Register) 종류와 기능**
  - 범용 레지스터(General Register)

- 연산 장치가 수행한 계산 결과의 임시 저장, 산술 및 논리 연산, 주소 색인 등의 목적으로 사용될 수 있는 레지스터

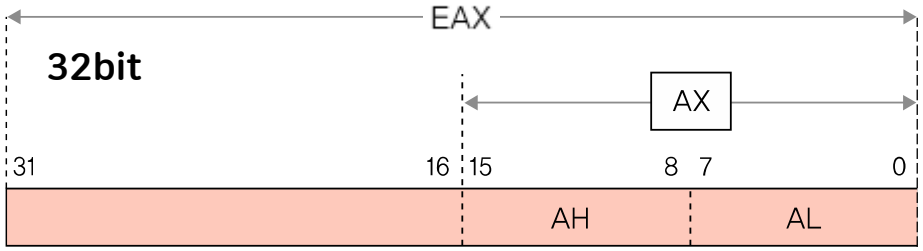
범주	이름		비트	용도
범용 레지스터 (General Register)	EAX	누산기(Accumulator)	32	주로 산술 연산에 사용(함수의 결과 값 저장)
	EBX	베이스 레지스터(Base Register)	32	특정 주소 저장(주소 지정을 위한 용도로 사용)
	ECX	카운트 레지스터(Count Register)	32	반복적으로 실행되는 명령에 사용(루프의 반복 횟수나 시프트 횟수 저장)
	EDX	데이터 레지스터(Data Register)	32	일반 자료 저장(입출력 동작에 사용)

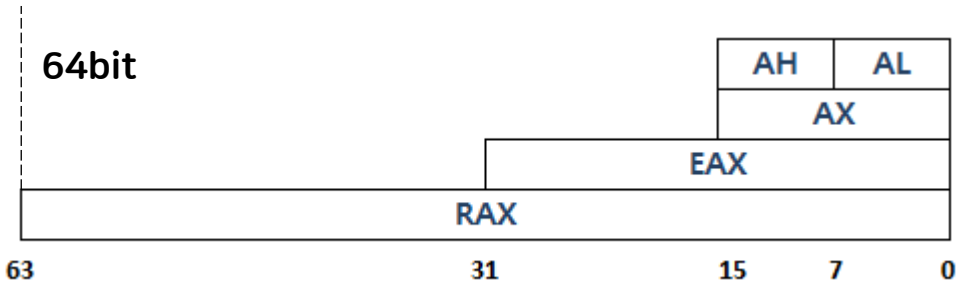
x86 시스템의 구조

- **x86 시스템 구조 – CPU : 레지스터(Register) 종류와 기능**
  - 범용 레지스터(General Register)

- EAX, EBX, ECX, EDX는 32비트 레지스터로 앞의 E는 '확장된(Extended)'을 의미
  64비트 레지스터는 RAX, RBX, RCX, RDX 등으로 앞의 R은 'Register'를 의미하는 것으로 알려져 있음

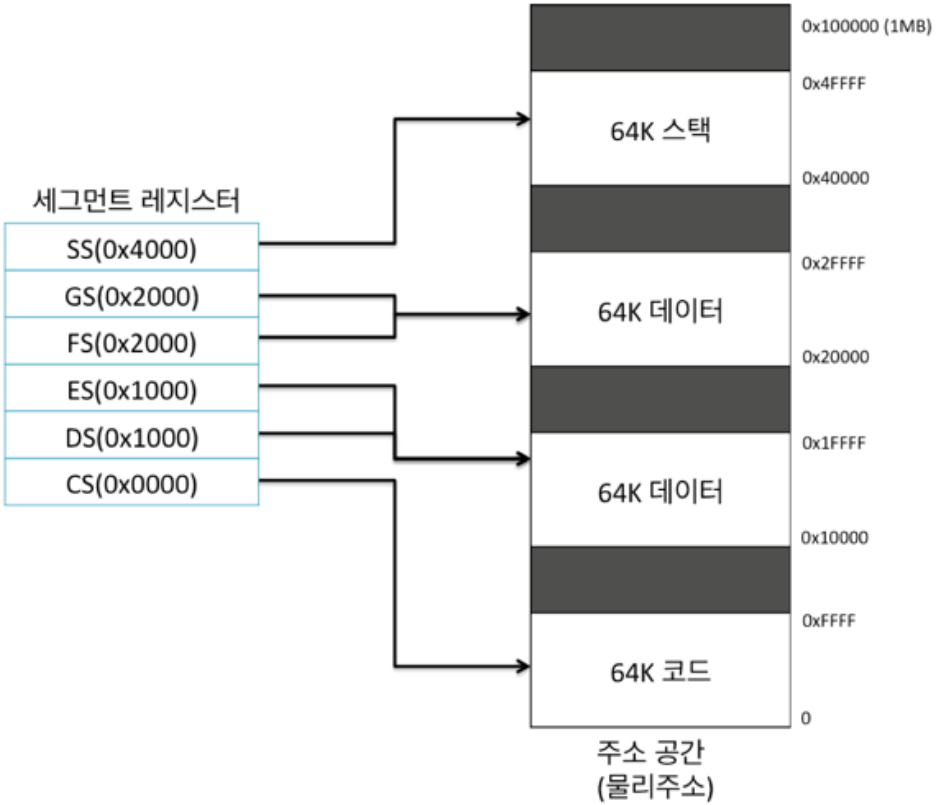
- 이 레지스터의 오른쪽 16비트를 각각 AX, BX, CX, DX라 부르고, 이 부분은 다시 둘로 나뉨
  예를 들어, AX는 왼쪽 8비트 상위(high) 부분을 AH, 오른쪽 8비트 하위(low) 부분을 AL





x86 시스템의 구조

- **x86 시스템 구조 – CPU : 레지스터(Register) 종류와 기능**
  - 세그먼트(Segment) : 프로그램에 정의된 메모리의 특정 영역으로 코드, 데이터, 스택 등을 포함



세그먼트 레지스터(Segment Register) :

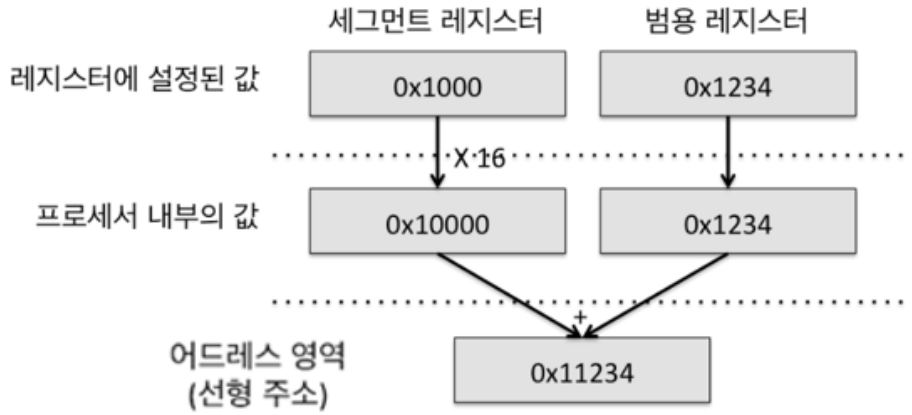
- 메모리를 다른 세그먼트로 나누어 관리하고,
  각 세그먼트에 대한 기준 주소를 저장

- 세그먼트를 이용함으로써, 물리 메모리를 효율적으로 사용하고
  프로그램 간 메모리 격리를 가능하게 함

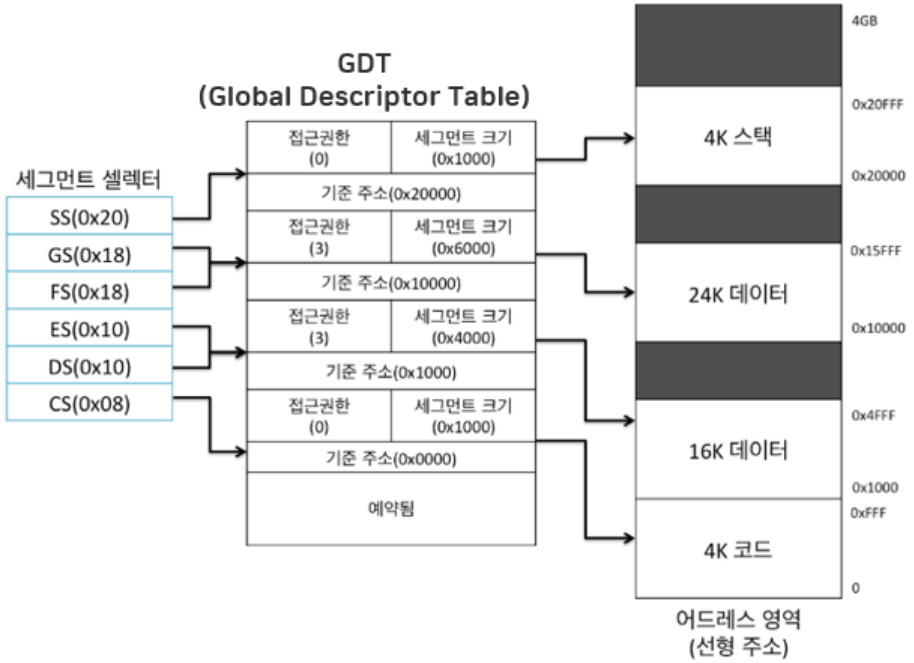
- CPU 운영 모드에 따라서 세그먼트 레지스터의 역할에 차이가 있음

x86 시스템의 구조

- **x86 시스템 구조 – CPU : 레지스터(Register) 종류와 기능**
  - x86 CPU 운영 모드(리얼 모드, 보호 모드)와 운영 모드에 따른 세그먼트 레지스터(Segment Register) 역할



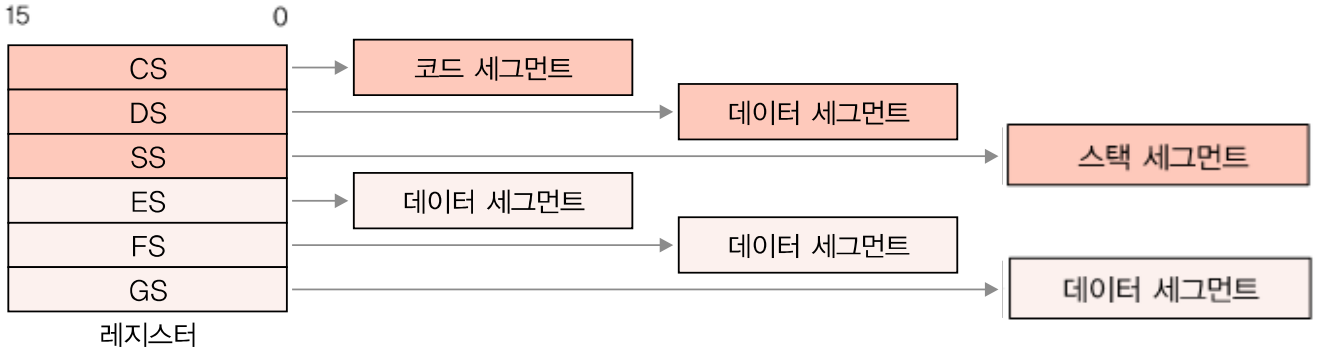
리얼 모드의 세그먼테이션



보호 모드의 세그먼테이션

x86 시스템의 구조

- **x86 시스템 구조 – CPU : 레지스터(Register) 종류와 기능**
  - 세그먼트 레지스터(Segment Register) : 각 세그먼트(Segment)에 대한 기준(시작) 주소를 저장
    포인터 레지스터(Pointer Register), 인덱스 레지스터(Index Register)와 결합하여 실제 주소 지정



범주	이름		비트	용도
세그먼트 레지스터 (Segment Register)	CS	코드 세그먼트 레지스터 (Code Segment Register)	16	프로그램의 코드 세그먼트의 시작 주소 EIP 레지스터의 오프셋 값을 더하면 메모리로부터 가져올 명령어의 주소
	DS	데이터 세그먼트 레지스터 (Data Segment Register)	16	프로그램의 데이터 세그먼트의 시작 주소 프로그램에서 정의된 데이터, 상수, 작업 영역의 메모리 주소 지정
	SS	스택 세그먼트 레지스터 (Stack Segment Register)	16	프로그램의 스택 세그먼트의 시작 주소 프로그램은 주소와 데이터의 임시 저장 목적으로 스택을 사용
	ES, FS, GS	엑스트라 세그먼트 레지스터 (Extra Segment Register)	16	문자열 연산과 추가 메모리 지정을 위해 사용되는 여분의 레지스터

x86 시스템의 구조

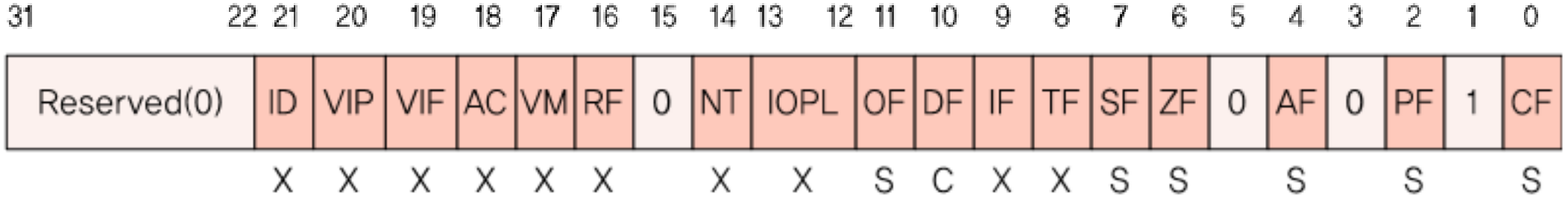
- **x86 시스템 구조 – CPU : 레지스터(Register) 종류와 기능**
  - 포인터 레지스터(Pointer Register) : 프로그램 실행 과정에서 사용되는 주요 메모리 주소 값을 저장하는 레지스터
  - 인덱스 레지스터(Index Register) : 인덱스 레지스터는 데이터를 복사할 때 출발지와 목적지 주소를 각각 가리키는 레지스터

범주	이름		비트	용도
포인터 레지스터 (Pointer Register)	EBP	베이스 포인터(Base Pointer)	32	SS 레지스터와 함께 사용되어 스택의 베이스 주소를 저장 ESP 레지스터와 함께 사용되어 스택 프레임(stack frame)을 형성
	ESP	스택 포인터(Stack Pointer)	32	SS 레지스터와 함께 사용되며 현재 스택 영역에서 가장 하위 주소를 저장
	EIP	명령 포인터(Instruction Pointer)	32	다음 명령어의 오프셋을 저장하며 CS 레지스터와 합쳐져 다음에 수행될 명령의 주소 저장
인덱스 레지스터 (Index Register)	EDI	목적지 인덱스(Destination Index)	32	목적지 주소에 대한 값 저장
	ESI	출발지 인덱스(Source Index)	32	출발지 주소에 대한 값 저장

x86 시스템의 구조

- **x86 시스템 구조 – CPU : 레지스터(Register) 종류와 기능**
  - 플래그 레지스터(Flag Register) : EFLAGS 레지스터

    연산 결과 및 시스템 상태와 관련된 여러 가지 플래그 값 저장, 상태 플래그, 제어 플래그, 시스템 플래그로 구성



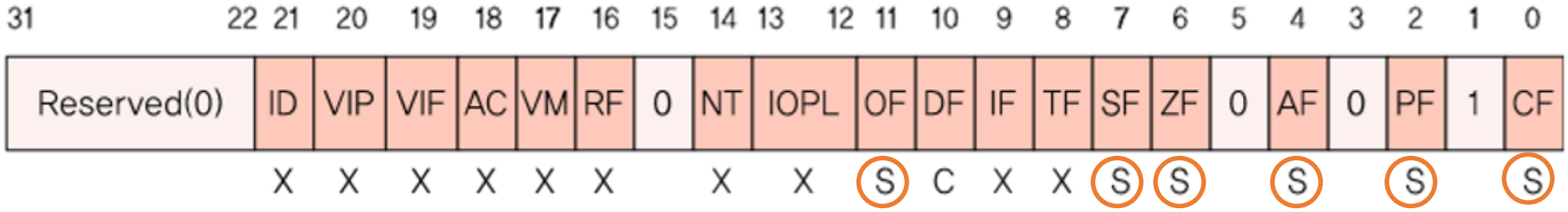
S : 상태 플래그 C : 제어 플래그 X : 시스템 플래그

* 흰색 부분은 사용되지 않는 비트

- CF : Carry Flag
- PF : Parity Flag
- AF : Adjust Flag
- ZF : Zero Flag
- SF : Sign Flag
- TF : Trap Flag
- IF : Interrupt enable Flag
- DF : Direction Flag
- OF : Overflow Flag
- IOPL : I/O Privilege Level
- NT : Nested Task flag
- RF : Resume Flag
- VM : Virtual 8086 Mode flag
- AC : Alignment Check
- VIF : Virtual Interrupt Flag
- VIP : Virtual Interrupt Pending
- ID : IDentification

x86 시스템의 구조

- **x86 시스템 구조 – CPU : 레지스터(Register) 종류와 기능**
  - 플래그 레지스터(Flag Register) : EFLAGS 레지스터
  - 상태 플래그(Status Flag) : 산술 명령(ADD, SUB, MUL, DIV)의 연산 결과를 반영



- ZF(Zero Flag) : 산술 연산 결과 0이면 세트(1), 이외에는 클리어(0)
- SF(Sign Flag) : 산술 논리 연산의 결과가 음수일 경우 발생하는 플래그, 결과가 음수라면 세트(1)
- CF(Carry Flag) : 부호 없는 숫자의 연산 결과가 비트 범위를 넘어섰을 때(자리 올림이나 자리 내림) 세트(1)
- OF(Overflow Flag) : 부호 있는 숫자의 연산에서 연산 결과가 최상위 비트(Sign-bit, MSB)는 제외한 비트 범위를 넘어섰을 때 세트(1)
- PF(Parity Flag) : 산술 연산 수행 결과 하위 1byte(8bit) 중 '1'의 개수가 짝수면 세트(1)
- AF(Adjust Flag) : 하위 4비트의 가장 높은 자리 수 비트에서 올림이 일어날 경우 세트(1), 이진화 십진수(BCD) 연산에 사용되는 플래그

x86 시스템의 구조

- **x86 시스템 구조 – CPU : 레지스터(Register) 종류와 기능**
  - 플래그 레지스터(Flag Register) : EFLAGS 레지스터
  - 상태 플래그(Status Flag) : CF(Carry Flag)와 OF(Overflow Flag)는 독립적으로 세트 됨

- Unsigned 4-bit 연산

  1000 + 1000 = 10000은 자리 올림이 발생한 것이기 때문에 CF(Carry Flag) 세트(1)

  Unsigned 연산은 부호 개념이 없어서 Overflow는 의미 없음

- Signed 4-bit 연산

  0111 + 0001 = 1000의 경우에는 MSB(Most Significant Bit), 즉 Sign-bit가 0에서 1이 되었으니 OF(Overflow) 세트(1)

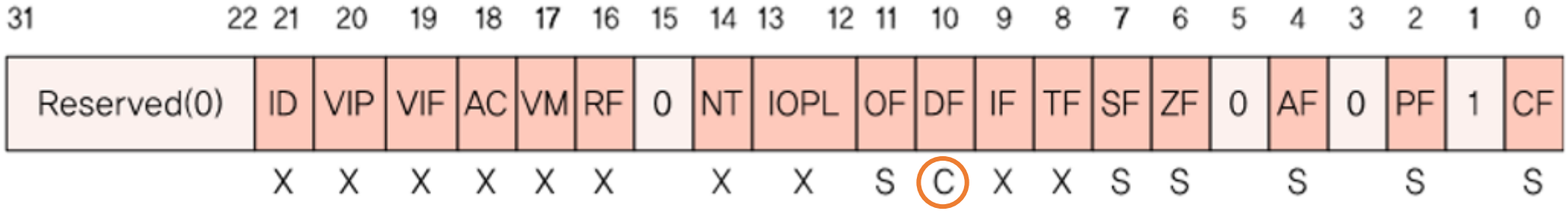
- CPU는 signed인지 unsigned인지 구분이 불가능하니 CF와 OF는 독립적으로 세트 됨

  1000 + 1000 = 10000 같은 경우는

  자리 올림이 발생했으니 CF가 세트(1) 되고, MSB가 바뀌었으니 OF가 세트(1)

x86 시스템의 구조

- **x86 시스템 구조 – CPU : 레지스터(Register) 종류와 기능**
  - 플래그 레지스터(Flag Register) : EFLAGS 레지스터
  - 제어 플래그(Control Flag) : 스트링 명령(MOVS, CMPS, SCAS, LODS, STOS)을 제어



- 스트링 조작을 할 때에 DF(Direction Flag)가

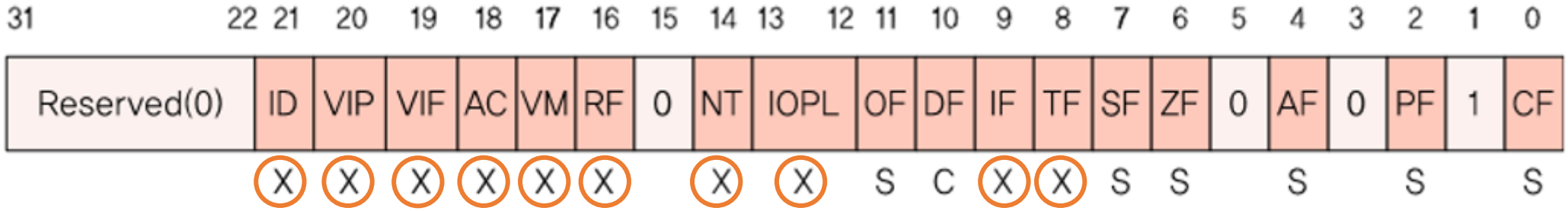
  '0'이면 번지를 나타내는 레지스터 값이 자동적으로 증가(낮은 주소에서 높은 주소로 처리)

  '1'이면 레지스터 값은 자동적으로 감소(높은 주소에서 낮은 주소로 처리)

- STD/CLD 명령은 각각 DF 플래그를 세트(1), 클리어(0)

x86 시스템의 구조

- **x86 시스템 구조 – CPU : 레지스터(Register) 종류와 기능**
  - 플래그 레지스터(Flag Register) : EFLAGS 레지스터
  - 시스템 플래그(System Flag)



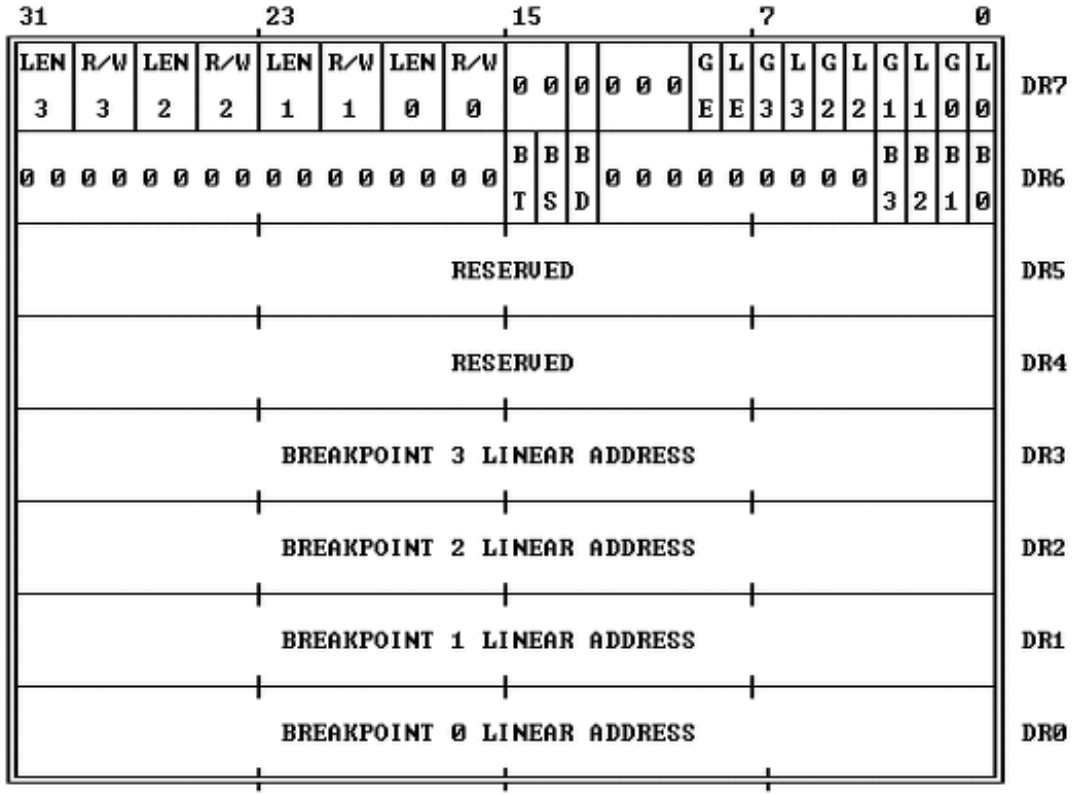
- TF(Trap Flag, 비트 8) : 디버깅 시'Single Step Mode'모드를 활성화하면 세트
- IF(Interrupt enable Flag, 비트 9) : 프로세서로부터 인터럽트가 발생했을 때 처리할지 여부를 제어
- IOPL(I/O Privilege Level, 비트 12/13) : 현재 실행되는 프로그램, 태스크의 입출력 특권 레벨 지시
- NT(Nested Task flag, 비트 14) : 인터럽트 되거나 호출된 태스크를 제어, 1로 설정 시 현재 태스크가 아전에 실행된 태스크와 연결되어 있음을 의미
- RF(Resume Flag, 비트 16) : 프로세서의 디버그 예외 반응을 제어, 1로 설정 시 디버그 오류를 무시하고 다음 명령어를 수행
- VM(Virtual 8086 Mode flag, 비트 17) : 가상 8086 모드로 전환 시키는데 사용되는 보호모드 플래그, 1로 설정 시 가상 8086 모드가 활성화되어 있음을 의미
- AC(Alignment Check, 비트 18) : 메모리 참조 시 정렬 기능을 활성화하면 세트
- VIF(Virtual Interrupt Flag, 비트 19), VIP(Virtual Interrupt Pending, 비트 20) : 가상 8086 모드 확장과 관련해 사용
- ID(Identification, 비트 21) : CPUID 명령의 지원 유무를 결정

x86 시스템의 구조

- **x86 시스템 구조 – CPU : 레지스터(Register) 종류와 기능**
  - 디버그 레지스터(Debug Register) :

    디버그 레지스터는 프로그램 디버깅을 목적으로 프로세서에 의해 사용되는 레지스터로써 DR0~DR7까지 총 8개

    그 중 DR0~DR3까지 4개의 레지스터만 직접 사용이 가능(보통 하드웨어 브레이크 포인트를 4개 까지만 설정이 가능한 이유)



- DR0~3(Hardware Break Point Address) :

  하드웨어 브레이크 포인터의 주소,

  해당주소 접근/쓰기/실행 시 INT1발생

- DR6(Debug Status Register) :

  INT1이 발생한 이유를 알려주는 레지스터로 설정은 프로세서가 하지만

  클리어는 사용자가 직접 해줘야 함

- DR7(Debug Control Register) :

  4개의 하드웨어 브레이크 포인터 발동조건을 지정(R/W0~3)

  (하드웨어 브레이크 포인터가 설정된 주소에 대해 발동 조건 지정)

  00 : 명령어 실행 시

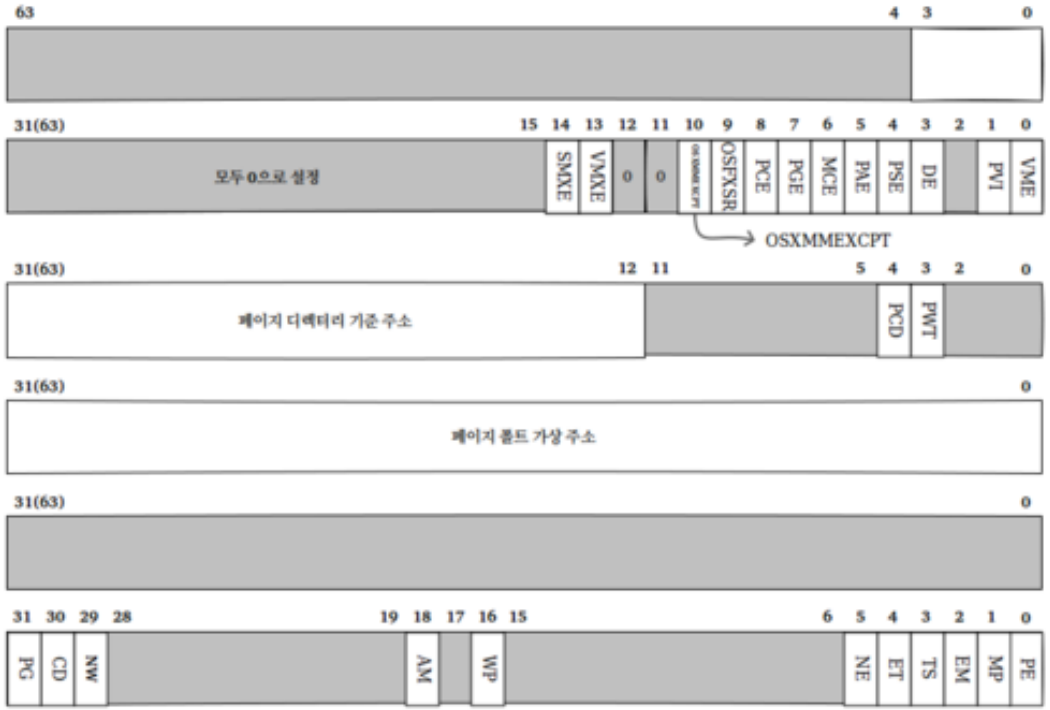
  01 : 쓰기 작업 발생 시

  10 : CR4 레지스터에 있는 DE 플래그가 1인 경우 I/O 읽기/쓰기 작업 발생 시

  11 : 모든 접근 시

x86 시스템의 구조

- **x86 시스템 구조 – CPU : 레지스터(Register) 종류와 기능**
  - 컨트롤 레지스터(Control Register) : 운영 모드를 변경, 현재 운영 중인 모드의 특정 기능을 제어하는 레지스터
    x86 프로세서에는 CR0 ~ CR4 5개의 컨트롤 레지스터, x86-64 프로세서에는 CR8 추가



- CR8 :
  - x86-64 프로세서에서 추가된 레지스터로 IA-32e 모드에서만 접근 가능
  - 태스크 우선 순위 레지스터의 값을 제어, 프로세스 외부에서 발생하는 인터럽트 필터링
- CR4
  - 프로세서에서 지원하는 각종 확장된 기능을 제어
  - 페이지 크기 확장, 메모리 영역 확장 등의 기능을 활성화
- CR3
  - 페이지 디렉터리의 물리 주소와 페이지 캐시에 관련된 기능을 설정하는 레지스터
- CR2
  - 페이지 폴트를 유발시킨 선형 주소를 담고 있는 레지스터
  - 페이징 기법을 활성화한 후에는 페이지 폴트 발생 시만 유효한 값을 가짐
- CR1
  - 프로세서에 의해 예약 됨
- CR0
  - 운영 모드를 제어, 리얼 모드에서 보호 모드로 전환, 캐시 및 페이징 기능 등을 활성화

Q A