응용보안

2. 80×86 시스템의 이해 1

경기대학교 AI컴퓨터공학부 이재흥
jhlee@kyonggi.ac.kr

KGU 경기대학교

CONTENTS
PRESENTATION

- 80×86 시스템 CPU 구조와 레지스터
- 80×86 시스템 메모리의 구조와 동작

학습 목표

- 80×86 시스템 CPU의 구조를 이해한다.
- 80×86 시스템 메모리 구조와 동작 원리를 이해한다.

80×86 시스템 CPU 구조와 레지스터

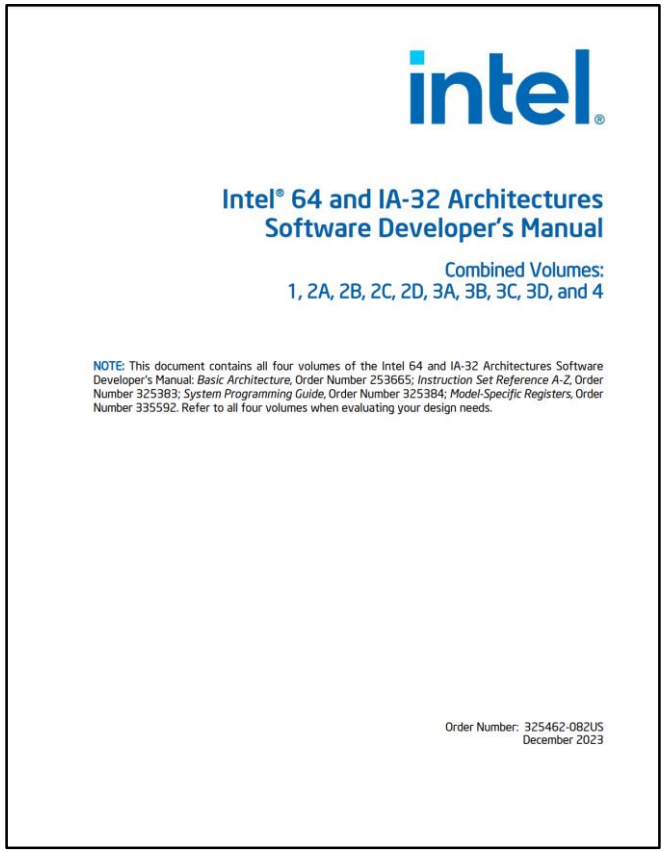
이 장에서 살펴볼 80x86 시스템 CPU

표 2-1 인텔 CPU 모델별 관련 정보

모델	8085	8086	8088	80286	80386	80486	펜티엄	펜티엄-프로
생산 시작 연도	1976년	1978년	1979년	1982년	1985년	1989년	1992년	1995년
클록 주파수(MHz)	3~8	5~0	5~8	6~16	16~33	25~50	60, 66	150
트랜지스터 수	6,500	29,000	29,000	130,000	275,000	1.2백 만	3.1백 만	5.5백 만
물리 메모리	64KB	1MB	1MB	16MB	4GB	4GB	4GB	64GB
내부 데이터 버스	8	16	16	16	32	32	32	32
외부 데이터 버스	8	16	8	16	32	32	64	64
주소 버스	16	20	20	24	32	32	32	36
데이터 크기(비트)	8	8, 16	8, 16	8, 16	8, 16, 32	8, 16, 32	8, 16, 32	8, 16, 32

이 장에서 살펴볼 80x86 시스템 CPU

- 자세한 내용을 알고 싶다면?
  - Intel® 64 and IA-32 Architectures Software Developer's Manual
    - 2023년 12월 버전 기준 5082 페이지



80x86 시스템 CPU의 구조

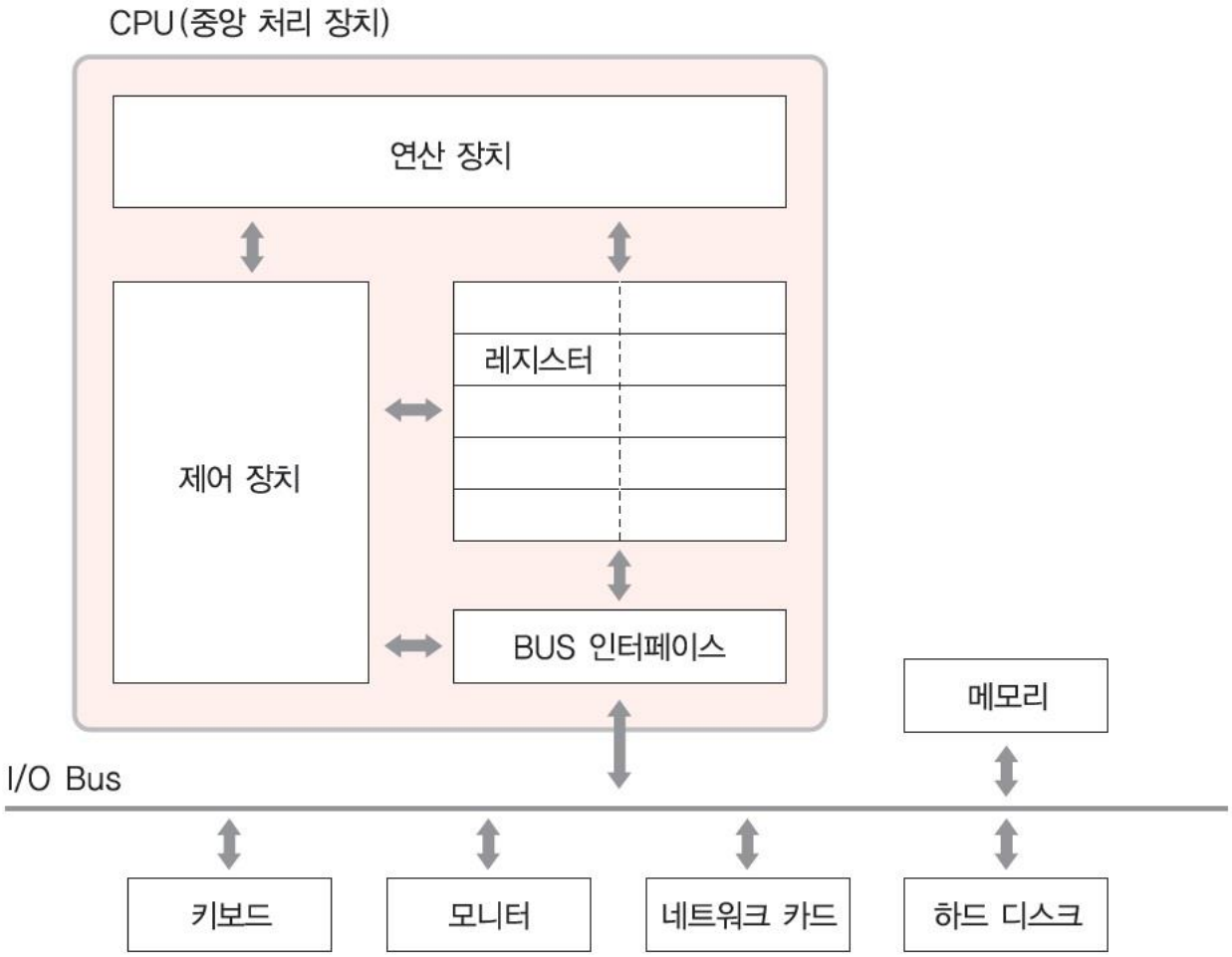


그림 2-1 80x86 시스템 구조

80x86 시스템 CPU의 구조

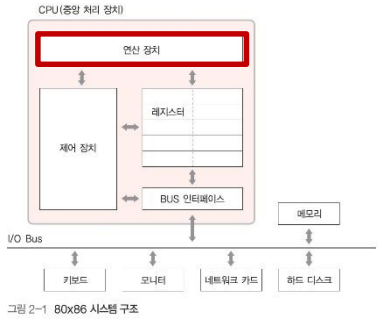


그림 2-1 80x86 시스템 구조

- 연산장치
  - 연산 장치(ALU : Arithmetic and Logic Unit)는 CPU(중앙 처리 장치)의 핵심 부분 중 하나로, 산술과 논리 등 연산을 수행하는 연산 회로 집합으로 구성
  - 연산 장치의 구성 요소

구성 요소		기능
내부 장치	가산기(Adder)	덧셈 연산 수행
	보수기(Complementer)	뺄셈 연산 수행. 1의 보수나 2의 보수 방식 이용
	시프터(Shifter)	비트를 오른쪽이나 왼쪽으로 이동하여 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 양수와 음수, 자리올림, 오버 플로우 상태 기억

80x86 시스템 CPU의 구조

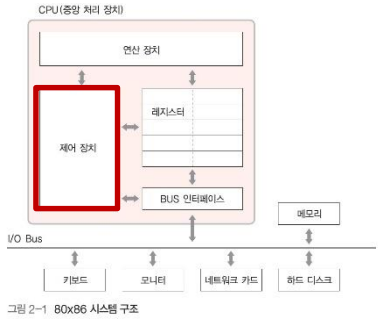


그림 2-1 80x86 시스템 구조

- 제어장치
  - 제어 장치(Control Unit)는 입력, 출력, 기억, 연산 장치를 제어하고 감시, 주기억 장치에 저장된 명령을 차례로 해독하여 연산 장치로 보내 처리하도록 지시
  - 제어 장치의 구성 요소

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(Instruction) 레지스터	현재 실행 중인 명령 저장
	메모리 주소(Memory Address) 레지스터	주기억장치의 번지 저장
	메모리 버퍼(Memory Buffer) 레지스터	메모리 주소 레지스터에 저장된 주소의 실제 내용 저장

80x86 시스템 CPU의 구조

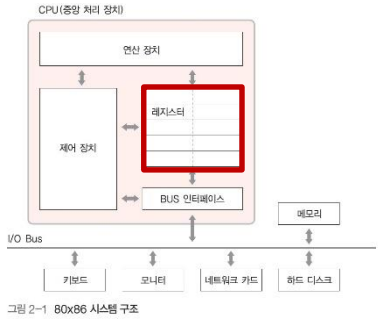


그림 2-1 80x86 시스템 구조

- 레지스터
  - 처리 중인 데이터나 처리 결과를 임시 보관하는 CPU 내의 기억 장치

레지스터의 종류와 기능

- 레지스터의 종류와 용도

범주	80386 레지스터	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 확대하기 위한 인덱스로 사용)
	ECX	카운트 레지스터 (Count Register)	32	반복적으로 실행되는 특정 명령에 사용(루프의 반복 횟수나 좌우 방향 시프트 비트 수 기억)
	EDX	데이터 레지스터 (Data Register)	32	일반 자료 저장(입출력 동작에 사용)
세그먼트 레지스터 (Segment Register)	CS	코드 세그먼트 레지스터 (Code Segment Register)	16	실행될 기계 명령어가 저장된 메모리 주소 지정
	DS	데이터 세그먼트 레지스터 (Data Segment Register)	16	프로그램에서 정의된 데이터, 상수, 작업 영역의 메모리 주소 지정
	SS	스택 세그먼트 레지스터 (Stack Segment Register)	16	프로그램이 임시로 저장해야 하는 데이터나 사용자의 피호출 서브 루틴(called subroutine)이 사용할 데이터와 주소 저장
	ES, FS, GS	엑스트라 세그먼트 레지스터 (Extra Segment Register)	16	문자 연산과 추가 메모리 지정을 위해 사용되는 여분의 레지스터

레지스터의 종류와 기능

- 레지스터의 종류와 용도

범주	80386 레지스터	이름	비트	용도
포인터 레지스터 (Pointer Register)	EBP	베이스 포인터 (Base Pointer)	32	SS 레지스터와 함께 사용되어 스택 내의 변수 값을 읽는데 사용
	ESP	스택 포인터 (Stack Pointer)	32	SS 레지스터와 함께 사용되며 스택의 가장 끝 주소를 가리킴
	EIP	명령 포인터 (Instruction Pointer)	32	다음 명령어의 오프셋(Offset, 상대 위치 주소)을 저장하며 CS 레지스터와 합쳐져 다음에 수행될 명령의 주소 형성
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
플래그 레지스터	EFLAGS	플래그 레지스터 (Flag Register)	32	연산 결과 및 시스템 상태와 관련된 여러 가지 플래그 값 저장

레지스터의 종류와 기능

- 레지스터의 종류와 용도

구분		맵핑 레지스터
연산장치 관련 레지스터	누산기(Accumulator)	EAX
	데이터 레지스터(Data Register)	EDX
	상태 레지스터(Status Register)	EFLAGS
제어장치 관련 레지스터	프로그램 카운터(Program Counter)	ECX
	명령 레지스터(Instruction Register)	EIP
	메모리 주소 레지스터(Memory Address Register)	EBX, EBP, ESP, EDI, ESI 등
	메모리 버퍼 레지스터(Memory Buffer Register)	프로그램 실행과 관련한 데이터를 저장하는 레지스터로 RAM의 역할을 한다.

레지스터의 종류와 기능

- 범용 레지스터
  - 연산 장치가 수행한 계산 결과의 임시 저장, 산술 및 논리 연산, 주소 색인 등 여러 목적으로 사용할 수 있는 레지스터로 EAX, EBX, ECX, EDX 등이 있음

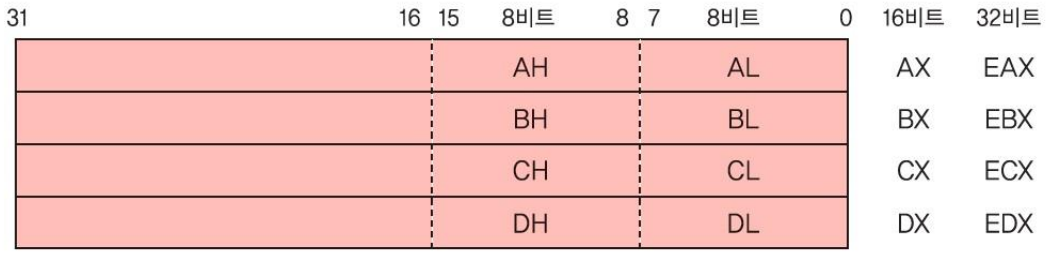


그림 2-2 범용 레지스터의 종류와 구분

  - EAX, EBX, ECX, EDX는 32비트 레지스터로 앞의 E는 확장된(Extended)을 의미
  - 이 레지스터의 오른쪽 16비트를 각각 AX, BX, CX, DX라 부르고, 이 부분은 다시 둘로 나뉨 (ex. AX = AH + AL)

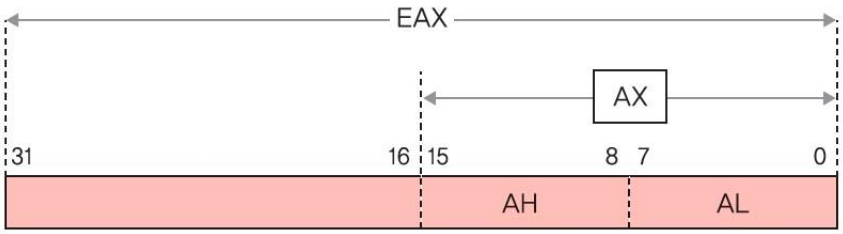


그림 2-3 32비트 레지스터의 사용 구분

레지스터의 종류와 기능

- 범용 레지스터
  - EAX 레지스터
    - 누산기, 입출력과 대부분의 산술 연산에서 사용
  - EBX 레지스터
    - DS 세그먼트에 대한 포인터를 주로 저장
    - ESI나 EDI와 결합하여 인덱스에 사용
  - ECX 레지스터
    - 루프가 반복되는 횟수를 제어하는 값
    - 왼쪽이나 오른쪽으로 이동되는 비트 수 등을 포함
  - EDX 레지스터
    - 입출력 연산에 사용
    - 큰 수의 곱셈과 나눗셈 연산에서 EAX와 함께 사용

레지스터의 종류와 기능

- 세그먼트 레지스터
  - 프로그램에 정의한 메모리상의 특정 영역
  - 코드, 데이터, 스택 등을 포함
  - CS, DS, SS 3개의 레지스터가 기본
  - ES, FS, GS 레지스터는 여분

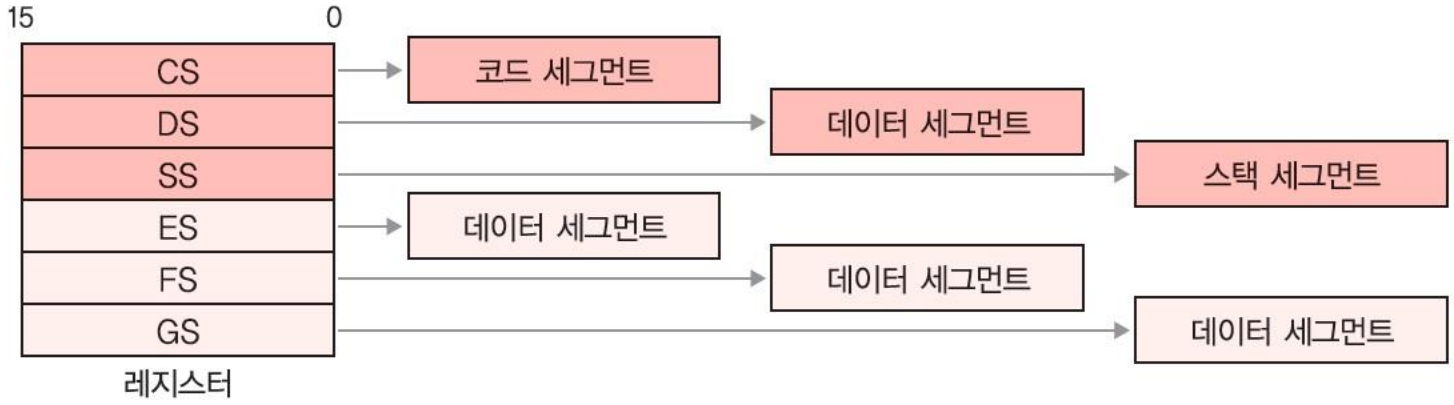


그림 2-4 세그먼트 레지스터와 세그먼트 맵핑

레지스터의 종류와 기능

- 세그먼트 레지스터
  - CS 레지스터
    - 실행될 기계 명령을 포함
    - 코드 세그먼트의 시작 주소를 가리킴
  - DS 레지스터
    - 프로그램에 정의된 데이터, 상수, 작업 영역을 포함
    - 데이터 세그먼트의 시작 주소를 가리킴
    - 프로그램은 참조하려는 데이터의 오프셋을 DS 레지스터에 저장된 주소 값에 더해 데이터 세그먼트 안의 데이터를 참조

레지스터의 종류와 기능

- 세그먼트 레지스터
  - SS 레지스터
    - 스택 세그먼트는 프로그램을 실행할 때, 실행 과정에서 필요한 데이터나 연산 결과 등을 임시로 저장하거나 삭제하는 등의 목적으로 사용
    - 스택 세그먼트의 시작 주소를 가리킴
  - ES, FS, GS 레지스터
    - ES 레지스터는 추가로 사용된 데이터 세그먼트의 주소를 가리킴
    - FS와 GS 레지스터도 목적은 비슷하나, 실제로는 거의 사용하지 않음

레지스터의 종류와 기능

- 포인터 레지스터
  - 프로그램의 실행 과정에서 사용하는 주요 메모리 주소 값을 저장하며 EBP, ESP, EIP가 있음
  - EBP 레지스터
    - 스택 세그먼트에서 현재 호출해서 사용하는 함수의 시작 주소 값을 저장
    - 함수로 전달되는 지역변수 등을 참조할 때 기준이 됨
    - ESP 레지스터와 함께 사용하여 스택 프레임(stack frame)을 형성
    - 실제 메모리상의 주소를 참조할 때 SS(Stack Segment) 레지스터와 함께 사용
  - ESP 레지스터
    - 현재 스택 영역에서 가장 하위 주소를 저장
    - EBP와 마찬가지로 실제 메모리상의 주소를 참조할 때 SS(Stack Segment) 레지스터와 함께 사용

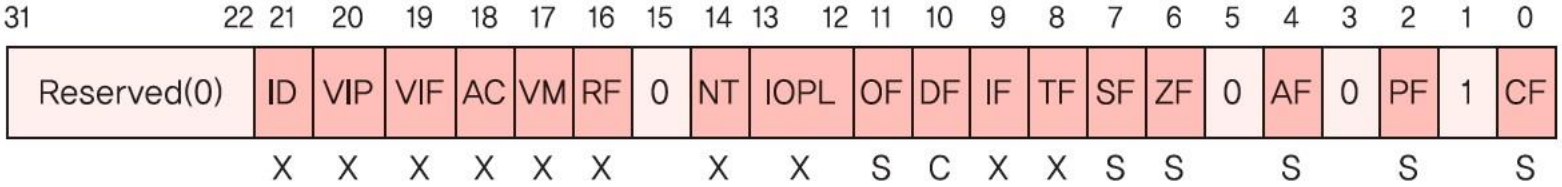
레지스터의 종류와 기능

- 포인터 레지스터
  - EIP 레지스터
    - 다음에 실행될 명령의 오프셋을 포함
    - CS (Code Segment) 레지스터와 함께 사용

- 인덱스 레지스터
  - ESI & EDI 레지스터
    - 메모리의 한 영역 (Source) 에서 다른 영역 (Destination) 으로 데이터를 연속적으로 복사할 때 사용

레지스터의 종류와 기능

- 플래그 레지스터
  - EFLAGS 레지스터
    - 크기가 32비트로, 컴퓨터의 다양한 상태를 나타내는 비트를 포함
    - 상태 플래그, 제어 플래그, 시스템 플래그로 구성



S: 상태 플래그 C: 제어 플래그 X: 시스템 플래그 * 음영 처리된 부분은 사용하지 않는 비트이다.

- 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

그림 2-5 EFLAGS 레지스터의 플래그 맵핑

레지스터의 종류와 기능

- 플래그 레지스터
  - 상태 플래그(Status Flag)
    - 산술 명령(ADD, SUB, MUL, DIV) 결과를 반영
    - CF(Carry Flag, 비트 0)
      - 산술 연산 결과로 자리올림이나 자리내림이 발생할 때 세트(1)
    - ZF(Zero Flag, 비트 6)
      - 산술 연산 결과 0이면 세트(1), 이외에는 클리어(0)
    - OF(Overflow Flag, 비트 11)
      - 부호가 있는 수의 오버플로우가 발생하거나 MSB(Most Significant Bit)가 변경되었을 때 세트

레지스터의 종류와 기능

- 플래그 레지스터
  - 제어 플래그(Control Flag)
    - 스트링 명령(MOVS, CMPS, SCAS, LODS, STOS)을 제어
    - DF가 1이면 스트링 명령 자동 감소, 0이면 자동 증가
    - STD/CLD 명령은 각각 DF 플래그를 세트(1), 클리어(0)

레지스터의 종류와 기능

- 플래그 레지스터
  - 시스템 플래그(System Flag)
    - TF(Trap Flag, 비트 8)
      - 디버깅 시 ‘Single Step Mode’ 모드를 활성화하면 세트
    - IF(Interrupt enable Flag, 비트 9)
      - 프로세서의 인터럽트 처리 여부를 제어
    - IOPL(I/O Privilege Level, 비트 12/13)
      - 현재 실행하는 프로그램이나 태스크의 입출력 특권 레벨을 지시
    - NT(Nested Task Flag, 비트 14)
      - 인터럽트하거나 호출된 태스크를 제어
    - RF(Resume Flag, 비트 16)
      - 프로세서의 디버그 예외 반응을 제어

레지스터의 종류와 기능

- 플래그 레지스터
  - 시스템 플래그(System Flag)
    - VM(Virtual 8086 Mode flag, 비트 17)
      - V86 모드를 활성화하면 세트
    - AC(Alignment Check, 비트 18)
      - 메모리를 참조할 때 정렬 기능을 활성화하면 세트
    - VIF(Virtual Interrupt Flag, 비트 19), VIP(Virtual Interrupt Pending, 비트 20)
      - 가상 모드 확장과 관련하여 사용
    - ID(IDentification, 비트 21)
      - CPUID 명령의 지원 유무를 결정

80×86 시스템 메모리의 구조와 동작

80x86 시스템의 메모리

- 메모리의 기본 구조

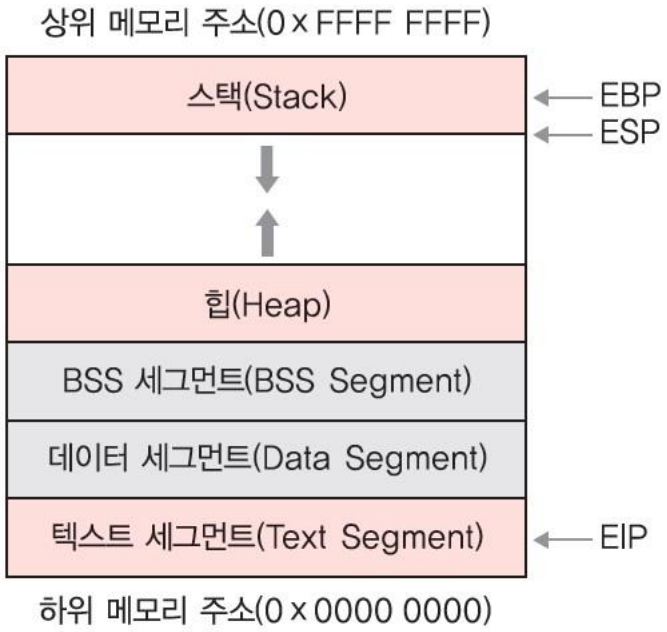


그림 2-6 80x86 시스템 메모리 구조

80x86 시스템의 메모리

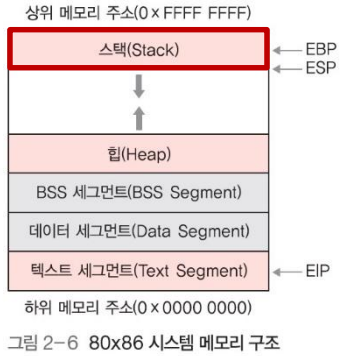


그림 2-6 80x86 시스템 메모리 구조

- 메모리의 기본 구조
  - 스택
    - 데이터 구조로서의 스택
      - 후입선출(LIFO : Last-In, First Out) 방식에 의해 정보를 관리하는 데이터 구조
      - Top이라고 불리는 스택의 끝부분에서 데이터의 삽입과 삭제가 발생
      - 가장 나중에 삽입된 정보가 가장 먼저 읽히는 특징이 있음
    - 컴퓨터 메모리 상의 스택
      - 프로그램 함수 내부에서 정의
      - 로컬 변수 등을 저장
      - 함수를 실행하는 동안만 존재하며, 함수 실행을 종료하면 사라짐
      - ESP 레지스터는 항상 스택의 가장 끝을 가리킴

80x86 시스템의 메모리

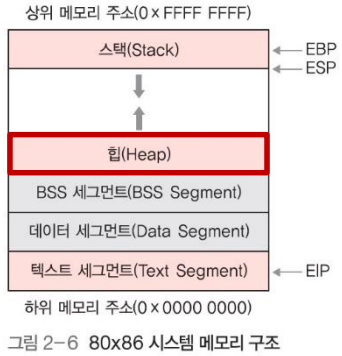


그림 2-6 80x86 시스템 메모리 구조

- 메모리의 기본 구조
  - 힙
    - 프로그램의 실행 중 필요한 기억 장소를 할당하기 위해 운영체제에 예약되어 있는 기억 장소영역
    - 데이터를 저장하기 위해 기억 장소를 요청하면 운영체제는 힙에 존재하는 기억 장소를 프로그램에 할당
    - 기억 장치가 더 이상 필요 없으면 할당 받았던 기억 장소를 운영체제에 반납, 운영체제에서는 반납된 기억 장소를 다시 힙에 돌려줌
    - 힙에 대한 기억 장소는 포인터를 통해 동적으로 할당되거나 반환
    - 연결 리스트, 트리, 그래프처럼 동적인 특성이 있는 데이터 구조에서 널리 사용

```
char *p = new char[1000];
```

80x86 시스템의 메모리

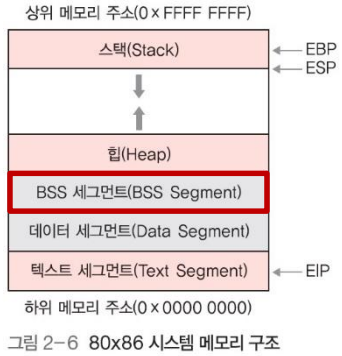


그림 2-6 80x86 시스템 메모리 구조

- 메모리의 기본 구조
  - BSS 세그먼트
    - 초기화되지 않은 데이터 세그먼트(Uninitalized data segment)라고 불리며, 프로그램이 실행될 때 0이나 NULL 포인터로 초기화되는 영역
    - 외부 변수나 static 변수 중 초기화 되지 않은 변수들을 저장

```
static int a;
```

80x86 시스템의 메모리

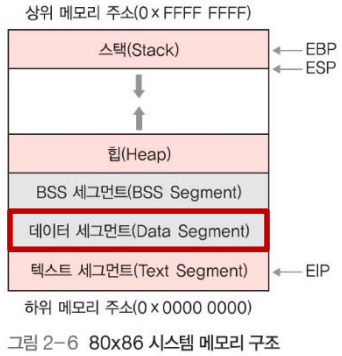


그림 2-6 80x86 시스템 메모리 구조

- 메모리의 기본 구조
  - 데이터 세그먼트
    - 초기화된 외부 변수나 static 변수 등을 저장하는 영역
    - 보통 텍스트 세그먼트(Text segment)와 데이터 세그먼트 영역을 합쳐 프로그램이라 함

```
static int a = 1;
```

80x86 시스템의 메모리

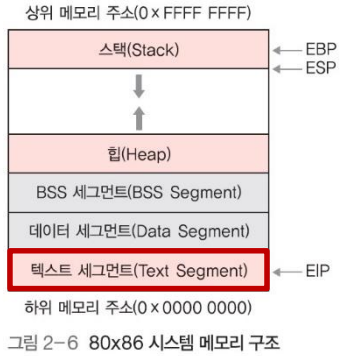


그림 2-6 80x86 시스템 메모리 구조

- 메모리의 기본 구조
  - 텍스트 세그먼트
    - CPU로 실행되는 머신 코드가 있는 영역으로, EIP가 다음에 실행하는 명령을 가리킴

메모리 접근 모드와 동작

- 초기 8086 CPU를 사용한 IBM-PC의 주소 공간
  - 1MB
    - 640KB(00000~0x9FFFF)
      - 프로그램이 사용
    - 384KB(0xA0000~0xFFFFF)
      - BIOS와 ISA(Industry Standard Architecture) 장치용으로 사용

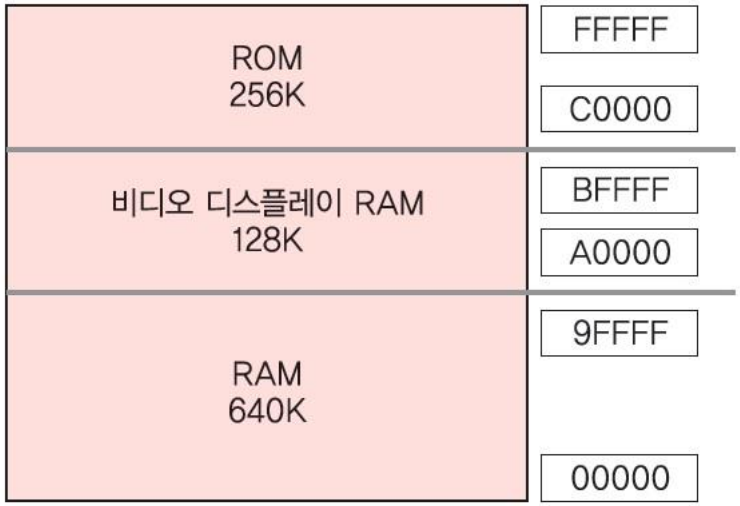


그림 2-7 초기 8086 시스템 메모리 구조

메모리 접근 모드와 동작

- 실제 모드
  - 8086 CPU에서 사용하던 동작 모드
  - 20비트 주소 버스 사용 위해 16비트 레지스터 사용
  - 총 1MB(2^{20} =1,048,567)의 메모리 사용 가능
  - 20비트 주소를 나타내기 위해 세그먼트 레지스터를 도입
  - 16비트 세그먼트 레지스터와 16비트 오프셋을 중첩시켜 20비트 물리 주소를 생성

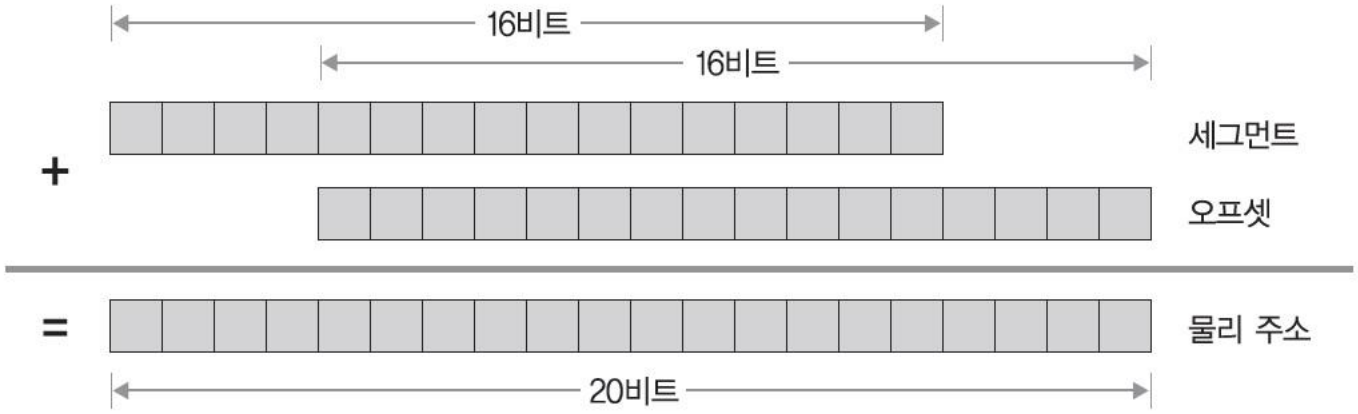


그림 2-8 20비트 메모리 주소 구성 방법

메모리 접근 모드와 동작

- 실제 모드
  - 세그먼트 주소인 CS 레지스터가 0x2525h, 오프셋인 IP가 0x95F3h면 0x2525h 뒤에 한 자리의 0x0h를 붙여 95F3h를 더한 2E843h가 실제 가리키는 물리 주소가 됨
  - 2525h : 95F3h, 또는 [CS]:96F3h로 표현
  - 하나의 세그먼트에 64KB(2^{16} =65,536)의 메모리 사용
  - 세그먼트 레지스터별 기본 오프셋 레지스터

세그먼트 레지스터	오프셋 레지스터
CS	IP
DS	SI, DI, BX
SS	SP, BP
ES	SI, DI, BX

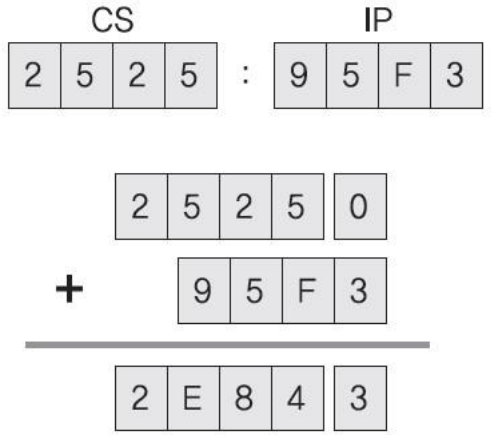


그림 2-9 20비트 메모리 주소 구성 예

메모리 접근 모드와 동작

- 보호 모드
  - 80286부터 도입한 보호 모드(Protected Mode)는 32비트 CPU 80386에 이르러 완성된 모습을 보이기 시작
  - 32비트 주소 버스를 해 4GB의 메모리 사용 가능
  - 세그먼테이션(Segmentation)과 페이징(Paging)을 이용하여 메모리를 관리
    - 세그먼테이션은 4GB의 메모리를 세그먼트 단위로 쪼갠 것으로, 여기서는 16비트 셀렉터와 32비트 오프셋을 이용해 4GB 범위의 32비트 선형 주소를 생성
    - 선형 주소는 메모리를 4KB 단위로 쪼개서 관리하는 페이징을 이용하여 물리 주소로 변환

그림 2-10 보호 모드에서 메모리 변환 과정

THANK YOU!