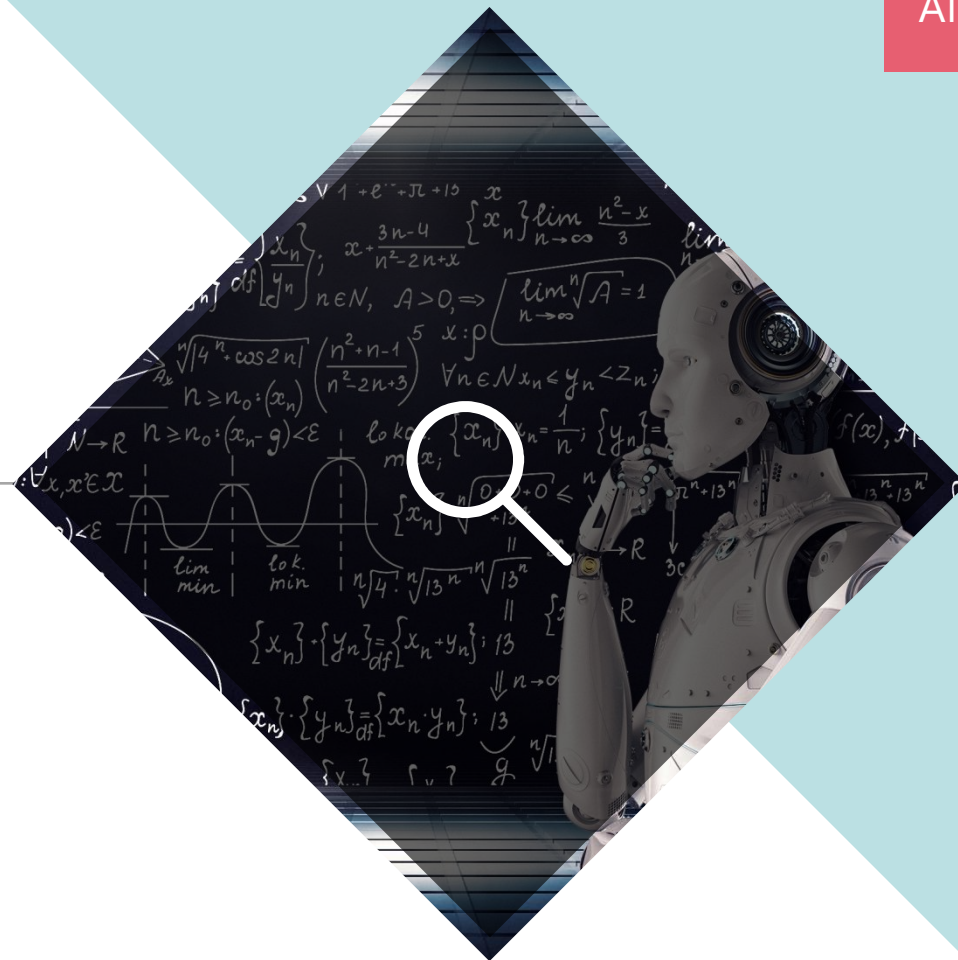
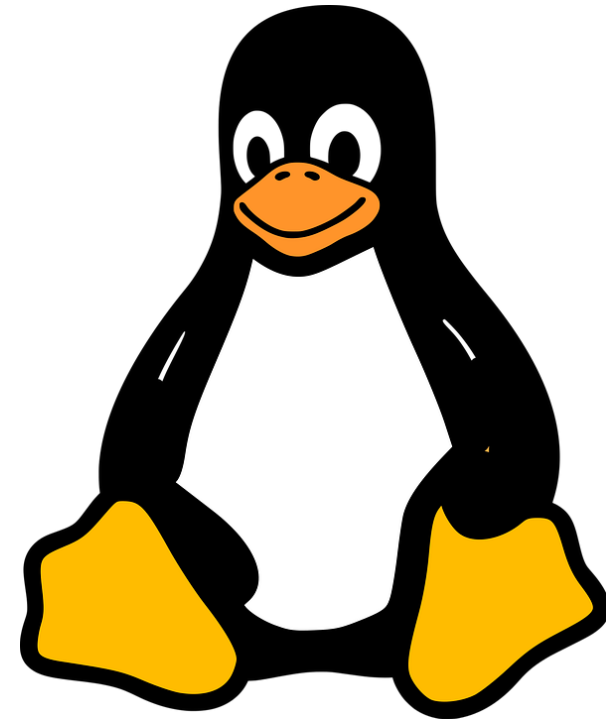


# Kernel - 001



# Contents

- I. 컴퓨터 구조
- II. 운영체제(OS)
- III. 커널이란?
- IV. Ubuntu Kernel 빌드/사용하기
- V. GRUB / DMESG



# I. 컴퓨터 구조

컴퓨터 구조를 알아야 하는 이유



프로그래밍 언어 뿐 아니라  
**컴퓨터의 근간**을 알아야 한다



- 문제 해결능력
- 성능 / 용량 / 비용

# I. 컴퓨터 구조

## 컴퓨터 구조를 알아야 하는 이유 (cont.)



문법에 맞는 소스 코드를  
컴퓨터에 입력만 하는 개발자

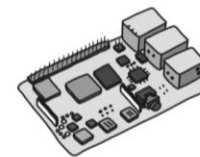


컴퓨터를 관조하며 문제를  
해결할 수 있는 개발자

# I. 컴퓨터 구조

## 컴퓨터 구조를 알아야 하는 이유 (cont.)

컴퓨터 구조는 결국  
**성능, 용량, 비용**에 대한 이야기



작은 컴퓨터



스마트폰



노트북



서버 컴퓨터

세상에 존재하는  
다양한 종류의 컴퓨터

<input type="radio"/>	r6a.large	2	x86_64	16	Up to 12.5 Gigabit	0.1134 USD per Hour	0.2054 USD per Hour
<input type="radio"/>	r6a.xlarge	4	x86_64	32	Up to 12.5 Gigabit	0.2268 USD per Hour	0.4108 USD per Hour
<input type="radio"/>	r6a.2xlarge	8	x86_64	64	Up to 12.5 Gigabit	0.4536 USD per Hour	0.8216 USD per Hour
<input type="radio"/>	r6a.4xlarge	16	x86_64	128	Up to 12.5 Gigabit	0.9072 USD per Hour	1.6432 USD per Hour
<input type="radio"/>	r6a.8xlarge	32	x86_64	256	12.5 Gigabit	1.8144 USD per Hour	3.2864 USD per Hour
<input type="radio"/>	r6a.12xlarge	48	x86_64	384	18.75 Gigabit	2.7216 USD per Hour	4.9296 USD per Hour

# I. 컴퓨터 구조

## 컴퓨터 구조 (4가지 핵심 부품)

```
00100101 10011101 10011001 10101011 00100100
11010101 00010001 00101001 10001010 01000111
11001001 10001111 10001001 11100000 00000000
11111101 01001011 01011100 01101000 10010110
00010111 00001110 00100010 00100010 10100010
00011011 01111000 10010101 10010000 11000110
10000101 01111100 11101111 11011100 00110000
01100111 11010010 10001100 00001110 01011110
11100100 00100010 10000110 10010111 10111001
01111000 10111111 11000100 01011001 01001111
10111111 01011001 00001001 11011001 11101111
10101000 01101000 11111100 10111110 00111100
1,1 All
```

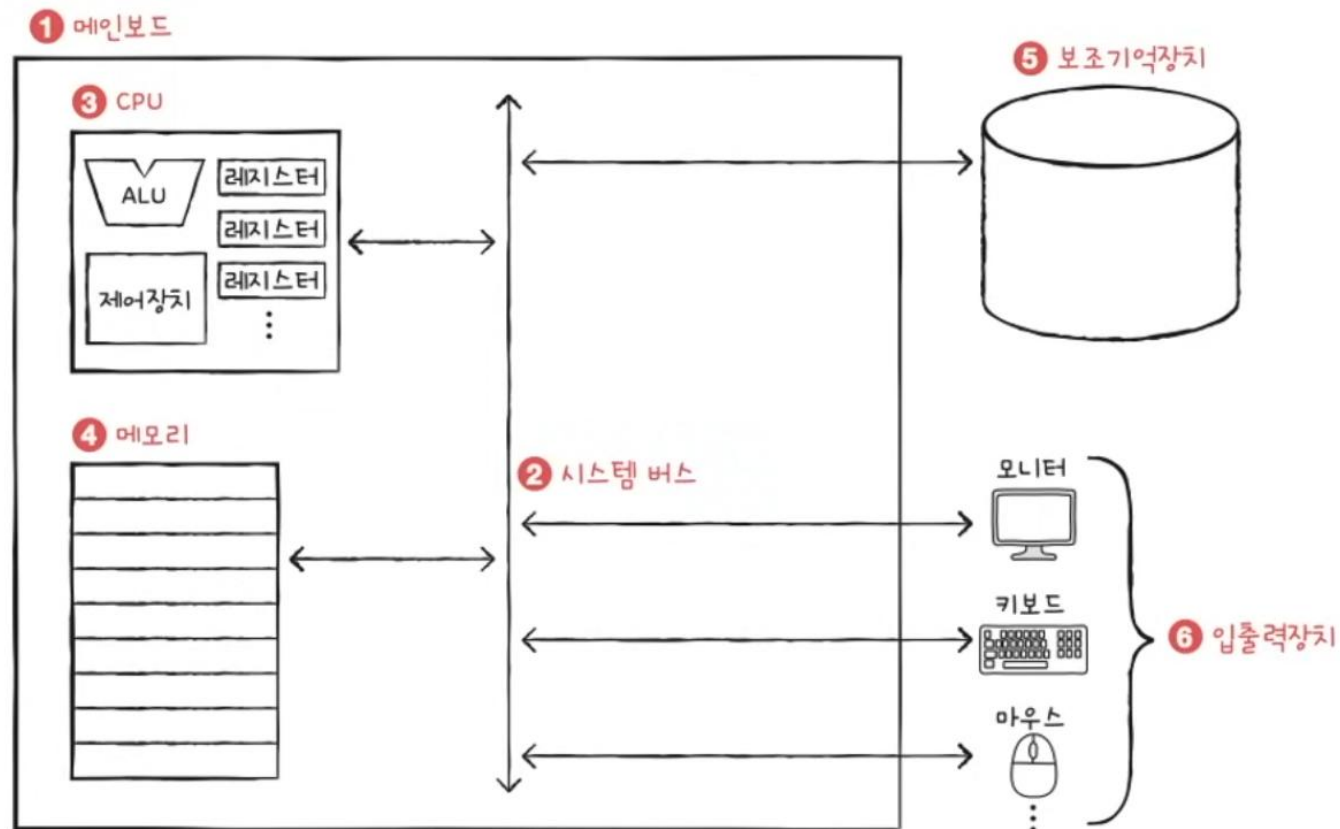
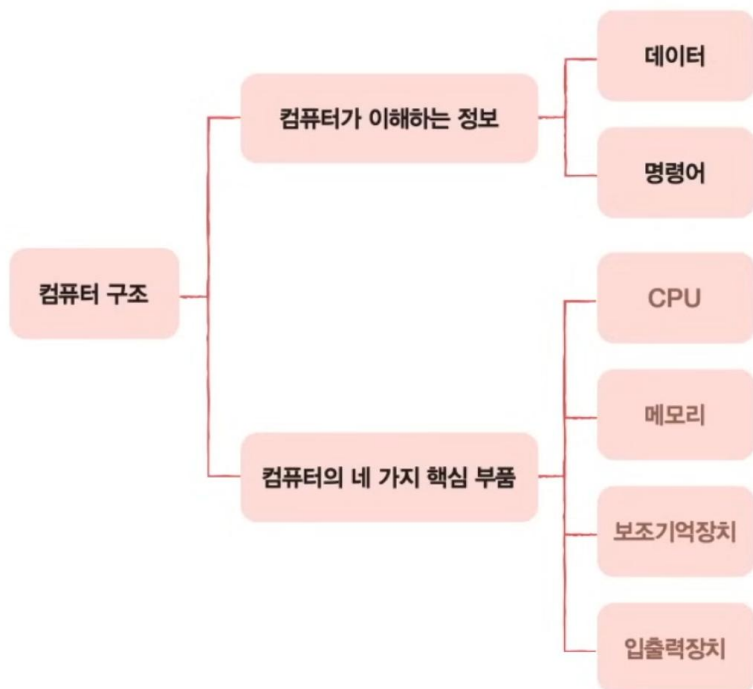
```
DOSSEG
.MODEL SMALL ;Let MASM handle the segment order
.STACK 400h ;Small model is adequate for this
;Set aside 1024 bytes for a stack
.DATA ;Start of the data segment
text DB "This is a simple MASM program"
DB 0Dh, 0Ah, 24h ;End with CR, LF and $ char
.CODE ;Start of code segment
go: mov ax,@DATA ;Load data segment location into DS
mov ds,ax
mov dx,OFFSET text ;Now DS:DX points to text
mov ah,09h ;DOS string display function number
int 21h ;Call DOS function
mov ax,4C00h ;Load DOS exit function number
int 21h ;Call DOS function (exit)

END go ;Start execution at label go
```



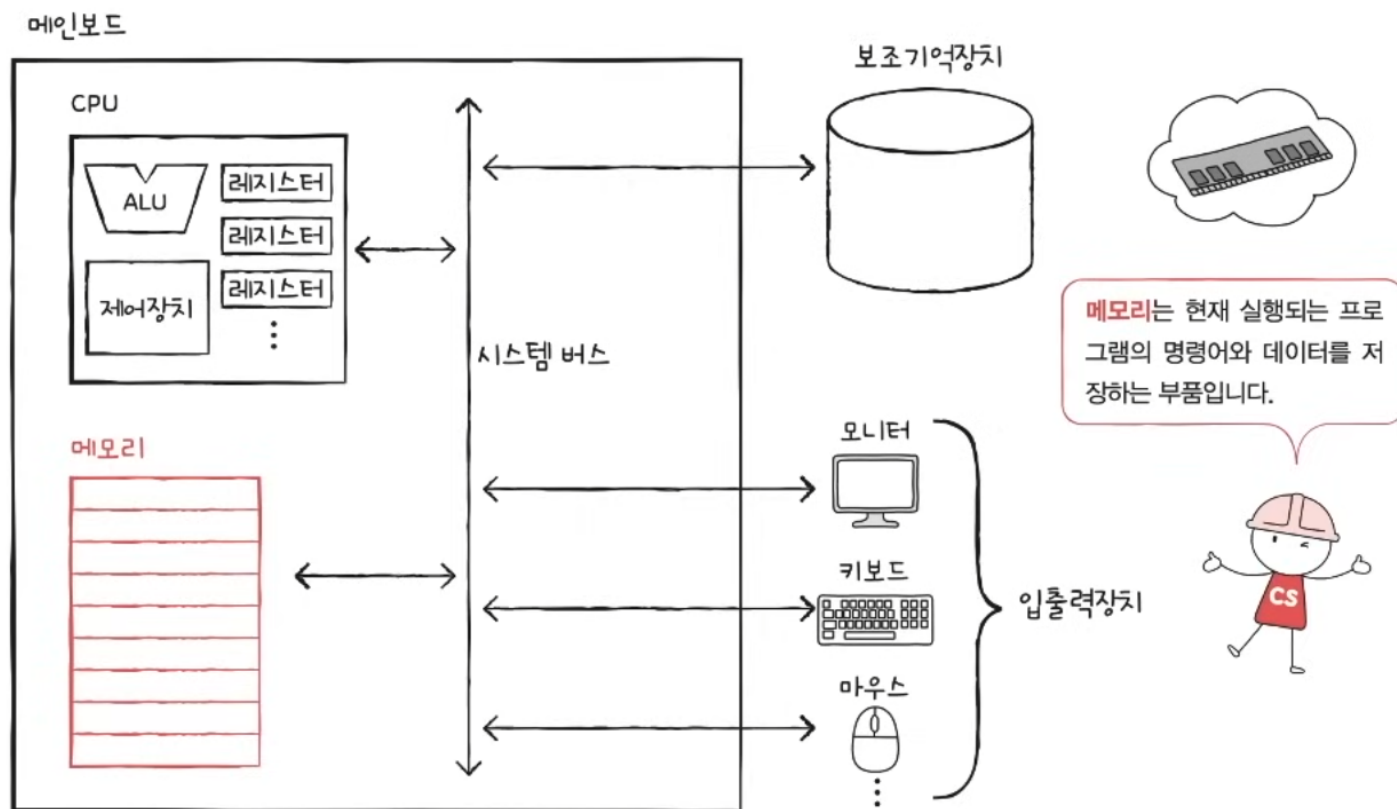
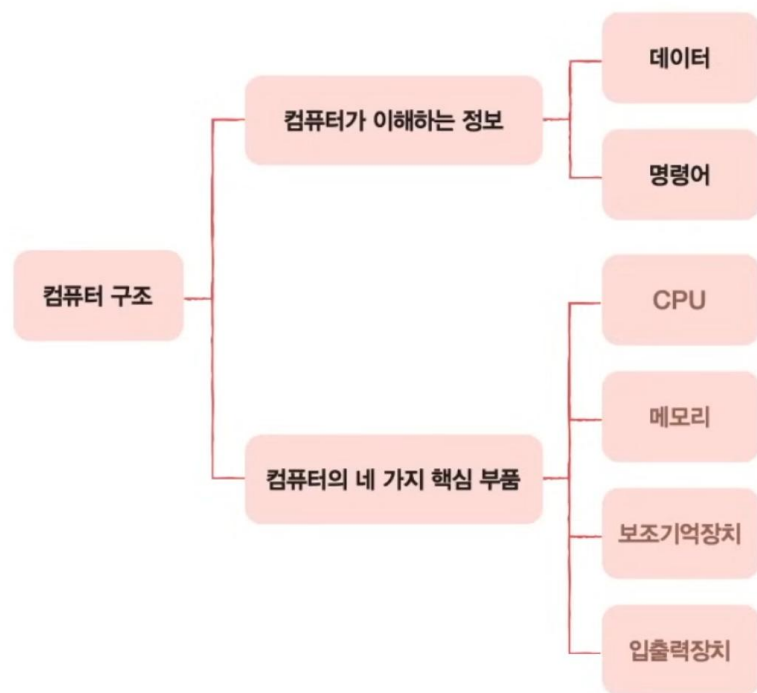
# I. 컴퓨터 구조

## 컴퓨터 구조 (4가지 핵심 부품)



# I. 컴퓨터 구조

## 컴퓨터 구조 (4가지 핵심 부품 - 메모리)

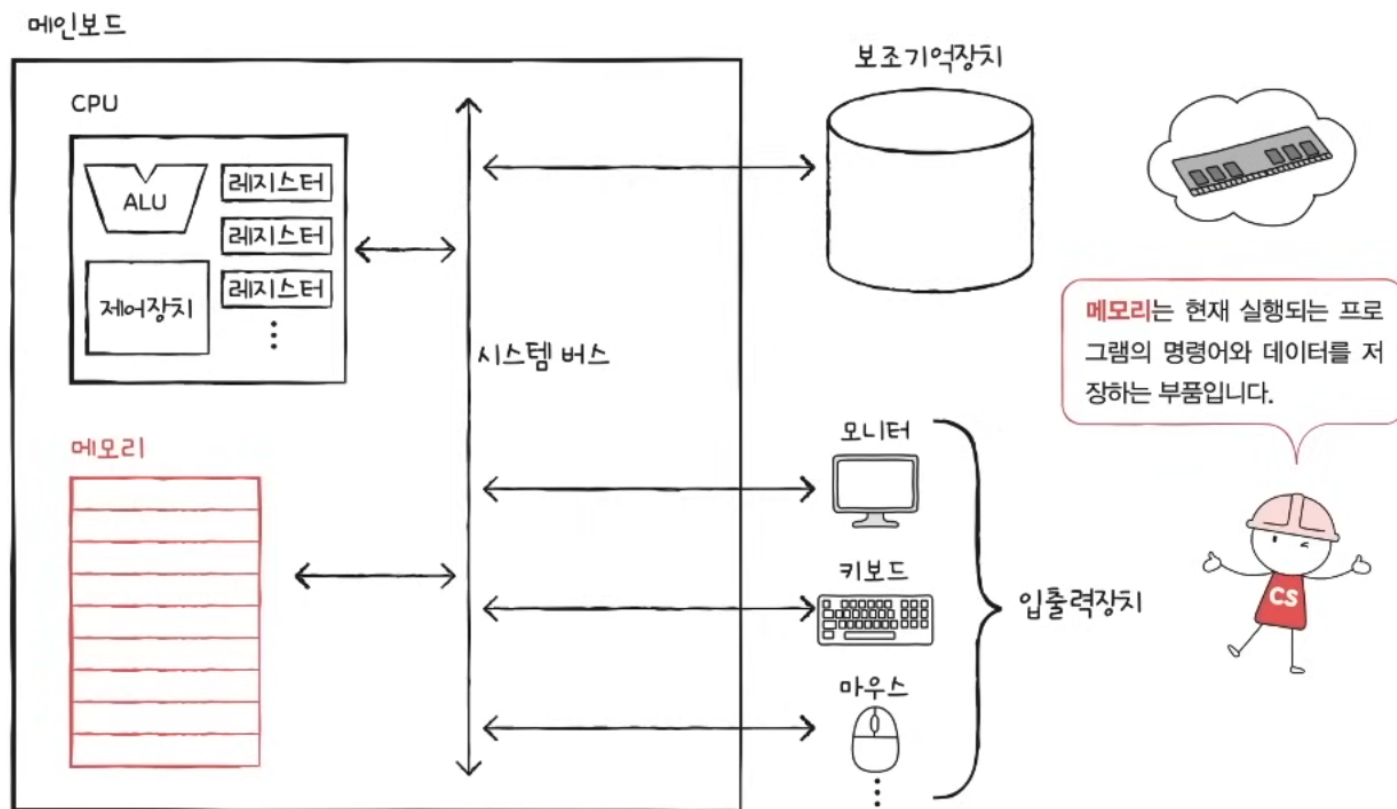
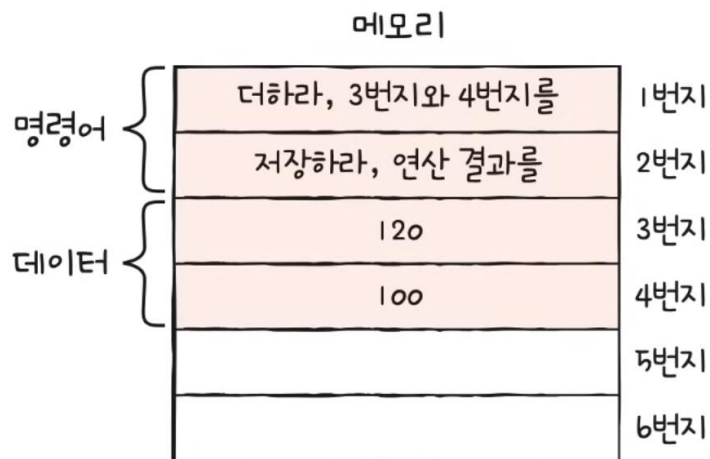




# I. 컴퓨터 구조

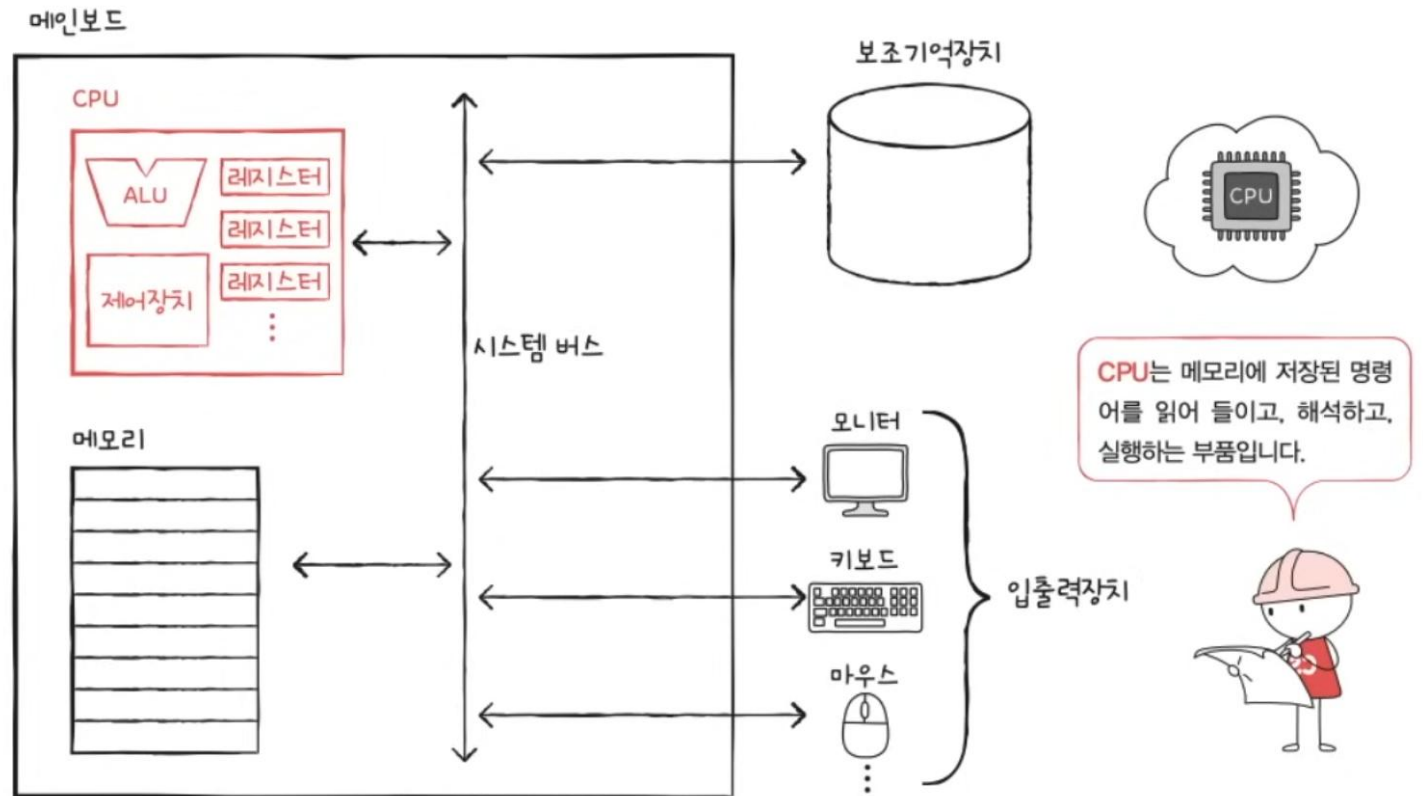
## 컴퓨터 구조 (4가지 핵심 부품 - 메모리)

- 메모리는 실행되는 프로그램의 명령어와 데이터를 저장한다
- 프로그램이 실행되려면 메모리에 저장되어 있어야 한다
- 메모리에 저장된 값의 위치는 주소로 알 수 있다



# I. 컴퓨터 구조

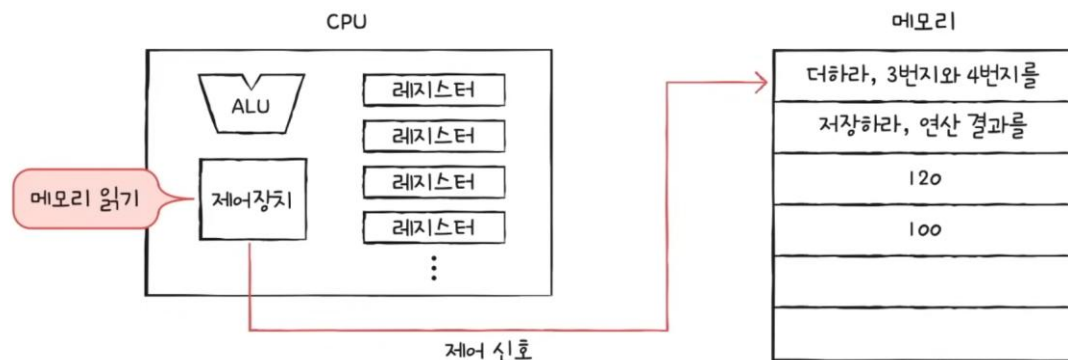
## 컴퓨터 구조 (4가지 핵심 부품 - CPU)



# I. 컴퓨터 구조

## 컴퓨터 구조 (4가지 핵심 부품 - CPU)

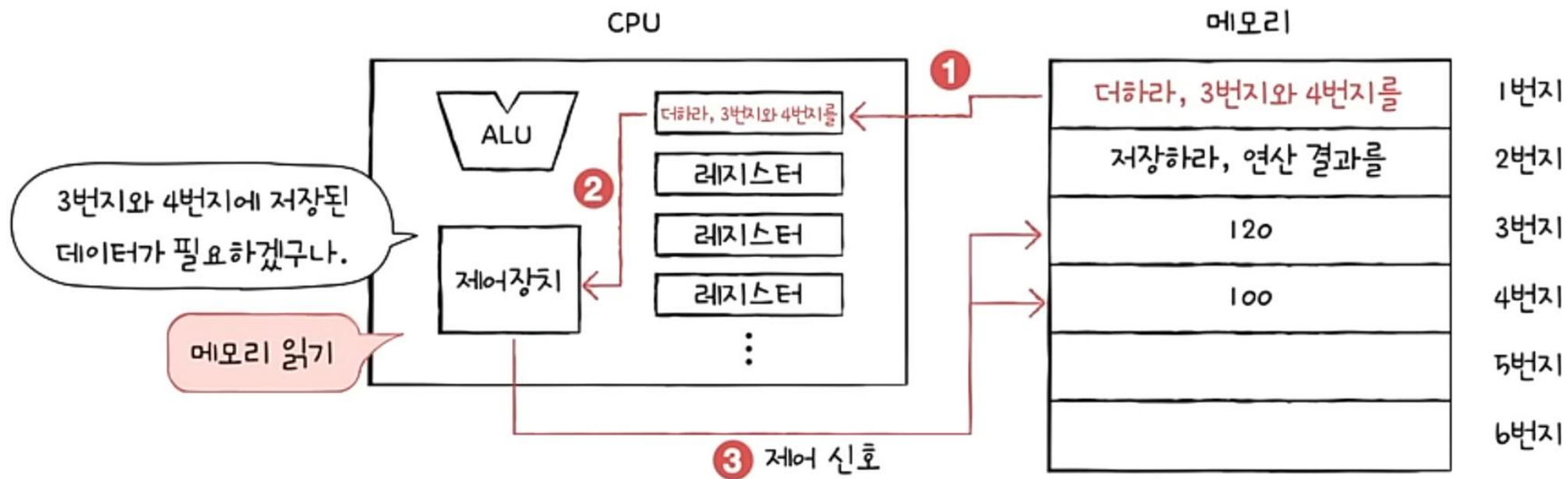
- CPU는 메모리에 저장된 값을 읽어 들이고, 해석하고, 실행하는 장치다
- CPU 내부에는 ALU, 레지스터, 제어장치가 있다
  - ALU는 계산하는 장치
  - 레지스터는 임시 저장 장치
  - 제어장치는 제어 신호를 발생시키고 명령어를 해석하는 장치



# I. 컴퓨터 구조

## 컴퓨터 구조 (4가지 핵심 부품 - CPU)

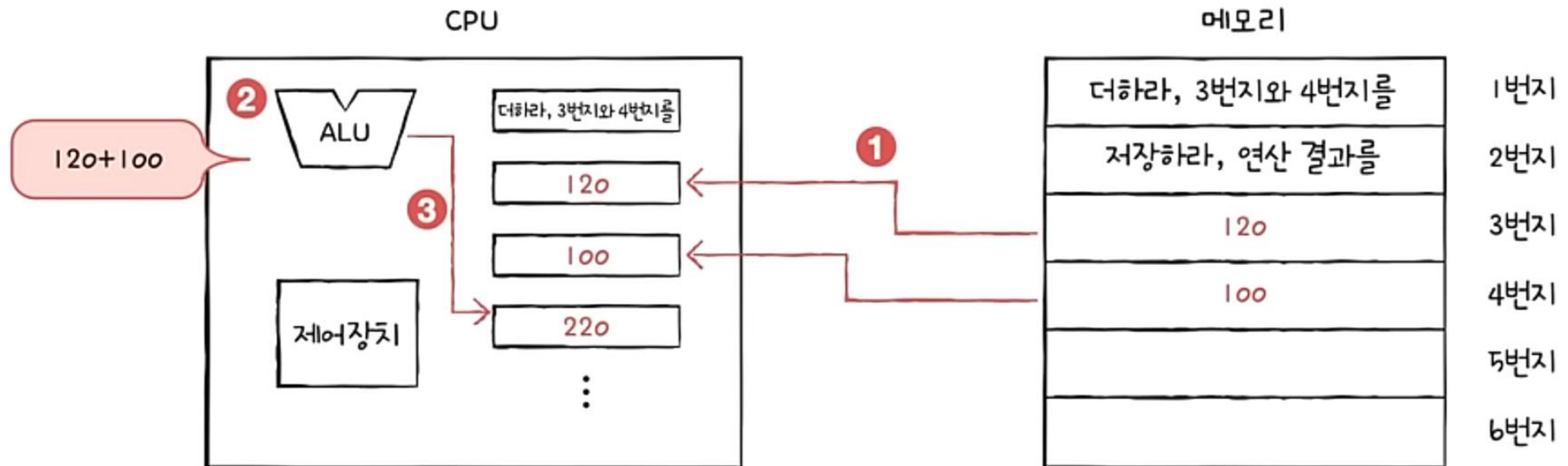
“120 + 100 = ?” 이 CPU 에서 처리되는 과정



# I. 컴퓨터 구조

## 컴퓨터 구조 (4가지 핵심 부품 - CPU)

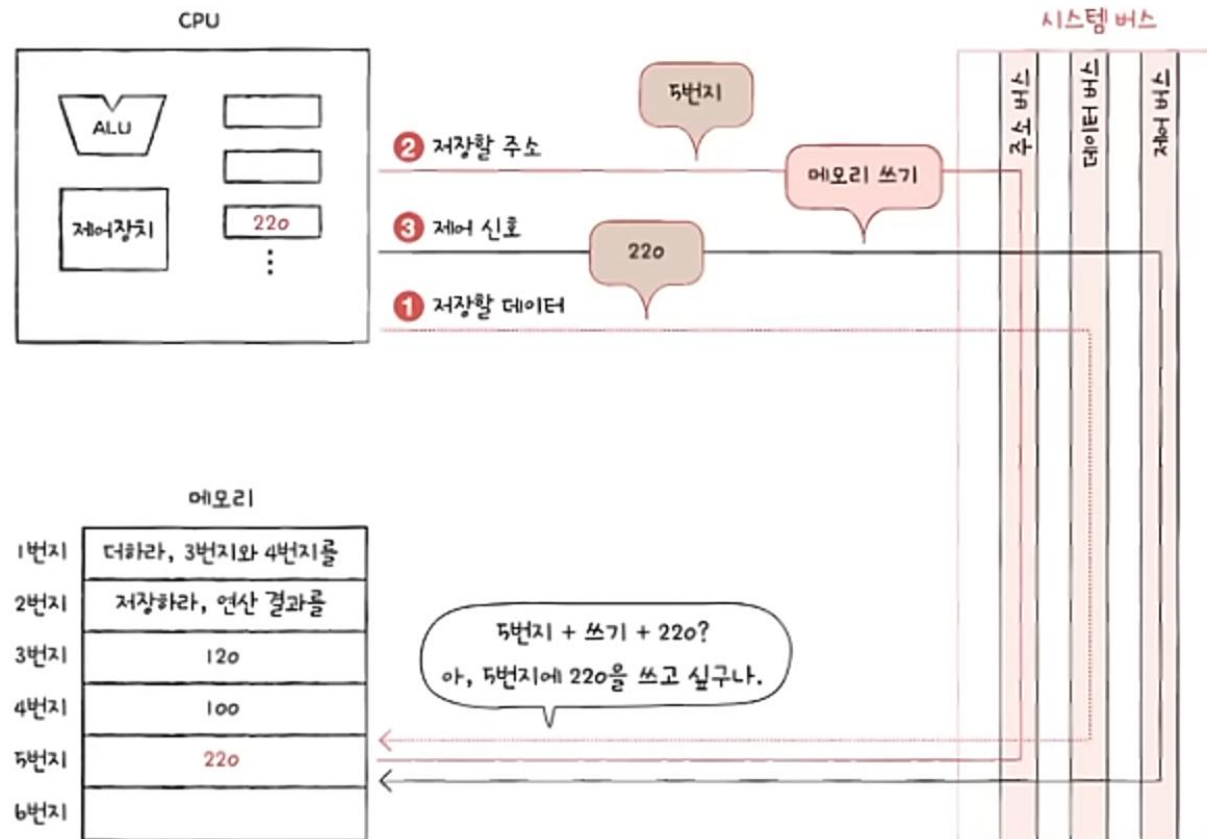
“ $120 + 100 = ?$ ” 이 CPU 에서 처리되는 과정



# I. 컴퓨터 구조

## 컴퓨터 구조 (4가지 핵심 부품 - CPU)

“120 + 100 = ?” 이 CPU 에서 처리되는 과정



# I. 컴퓨터 구조

## 컴퓨터 구조 (4가지 핵심 부품 - CPU)

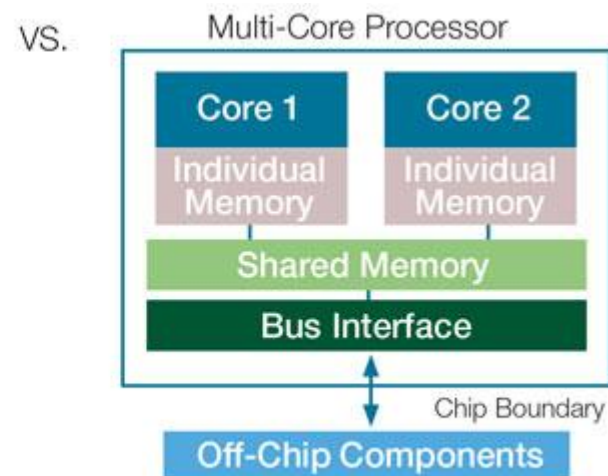
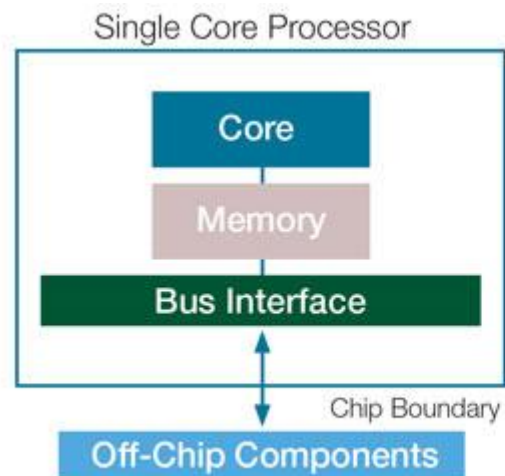
**Multi-Core?** - 하나의 CPU 칩 안에 2개 이상의 코어(연산 장치)를 내장한 구조

싱글 코어 CPU가 하나의 코어로 작업을 처리하는 반면, 멀티 코어 CPU는 여러 코어가 병렬로 작업을 처리하여 동시에 여러 작업을 수행하거나 더 복잡한 작업을 효율적으로 처리함

듀얼 코어 (Dual-core): 2개의 코어를 가진 CPU.

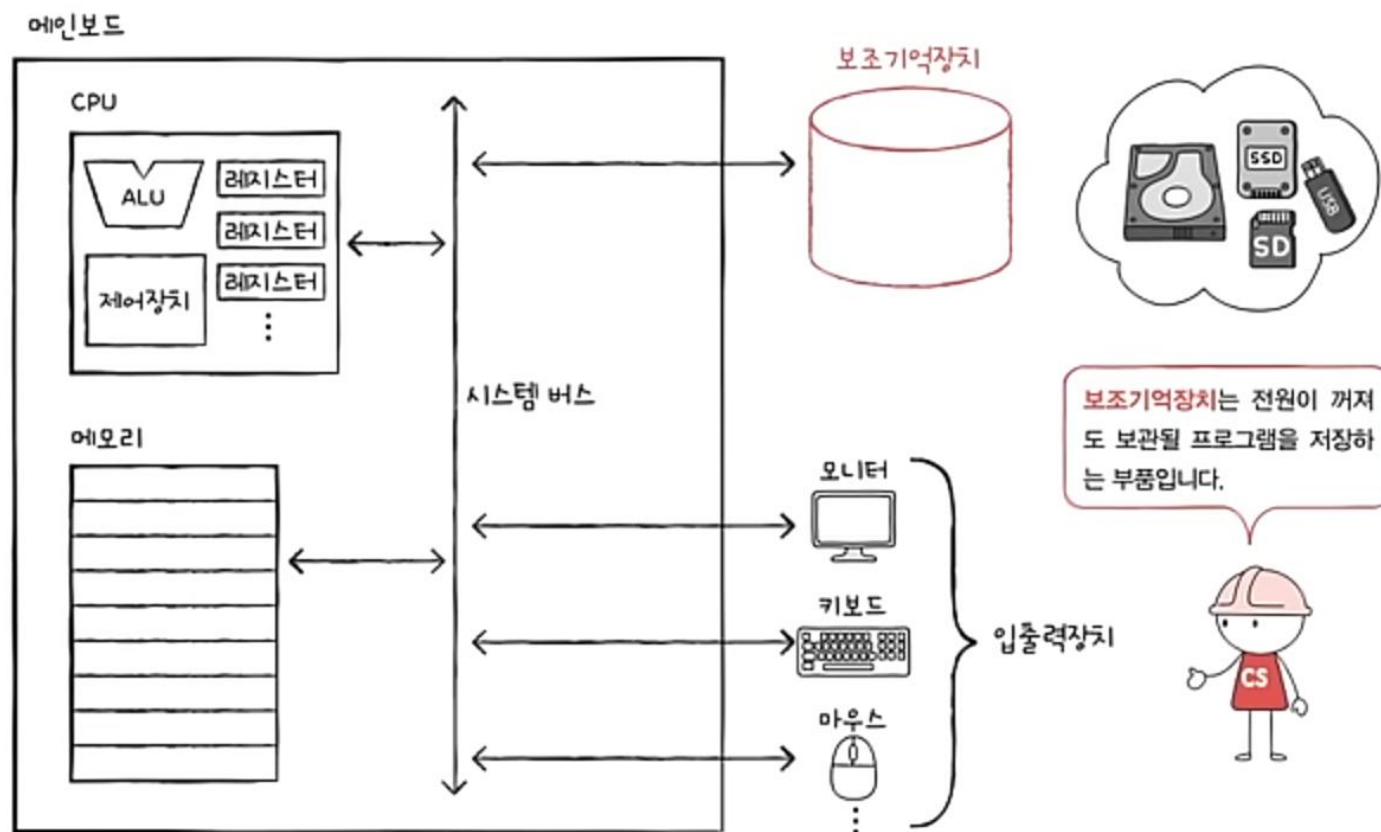
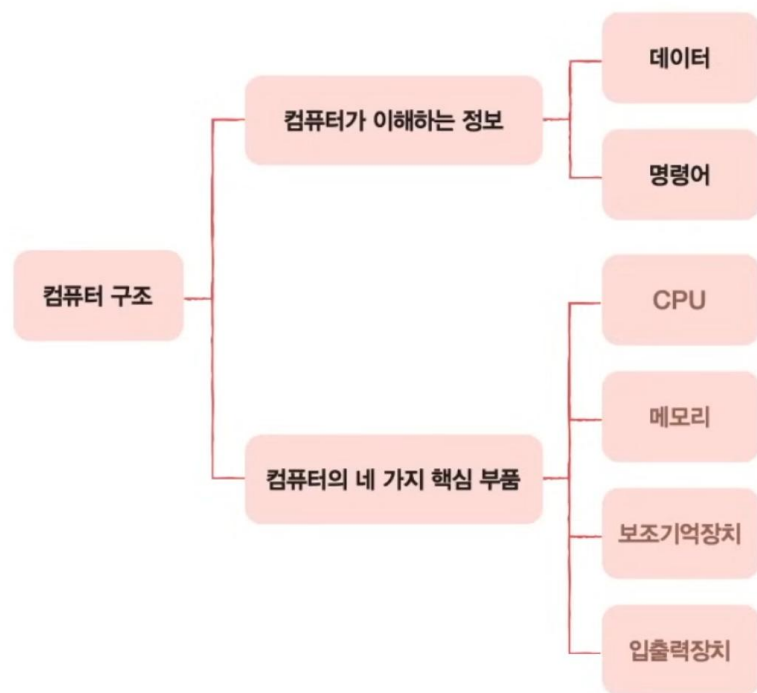
쿼드 코어 (Quad-core): 4개의 코어를 가진 CPU.

옥타 코어 (Octa-core): 8개의 코어를 가진 CPU.



# I. 컴퓨터 구조

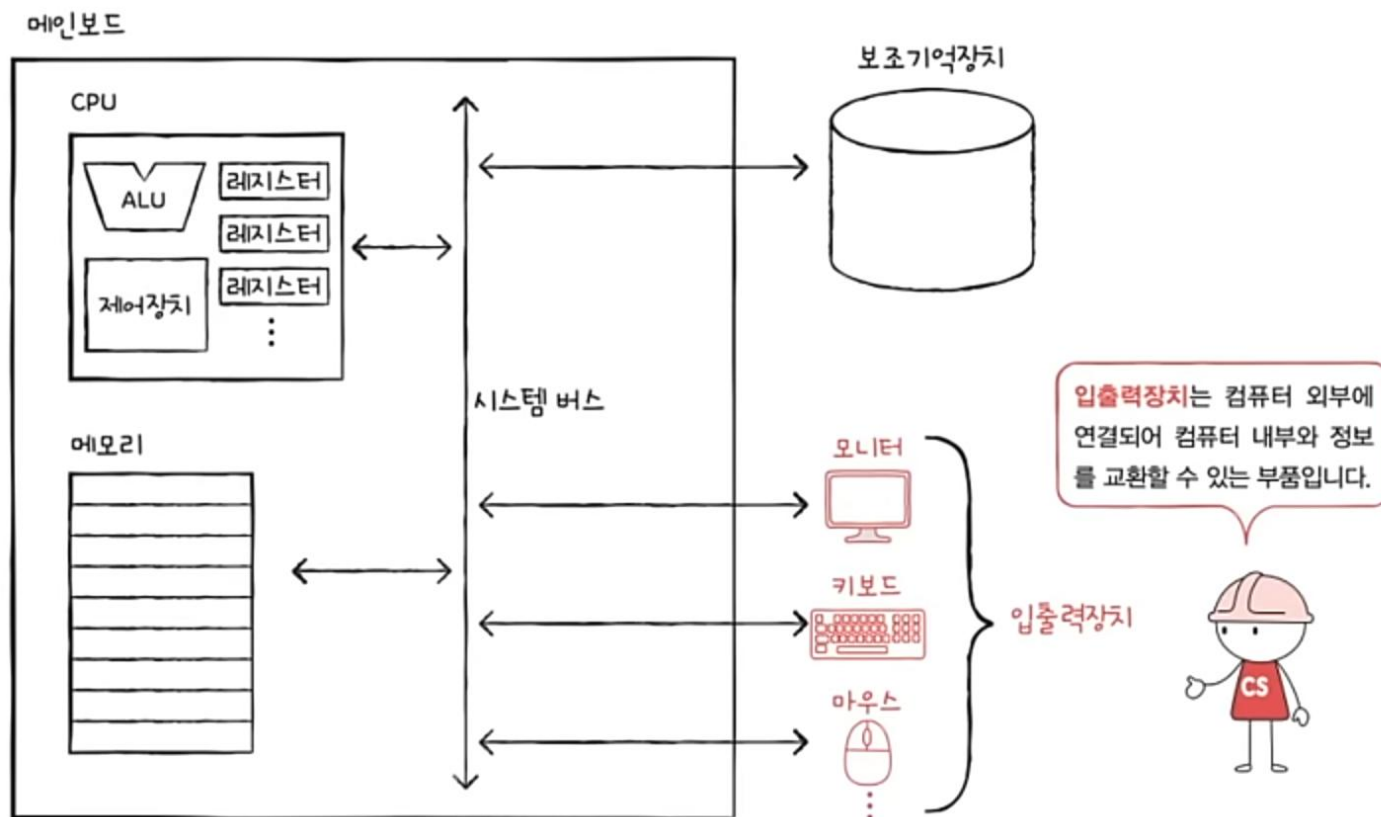
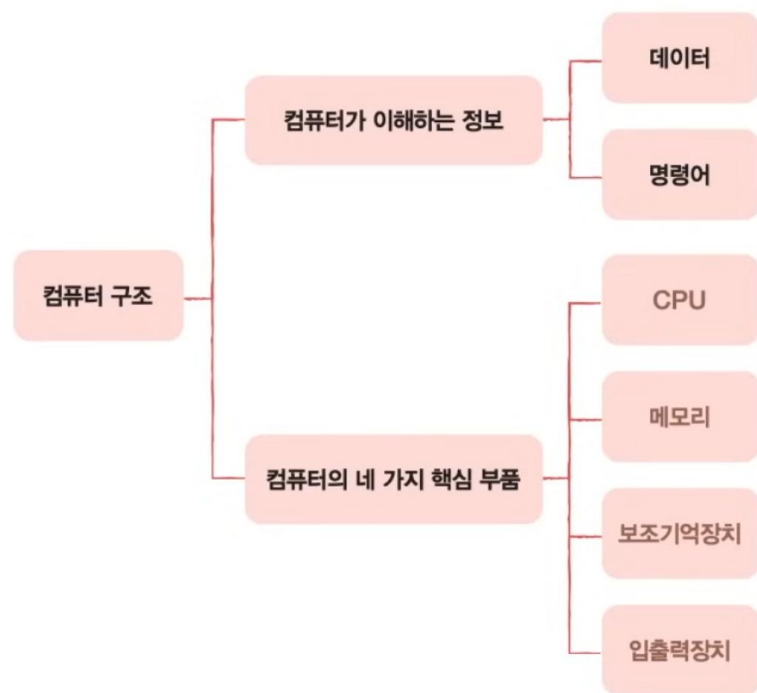
## 컴퓨터 구조 (4가지 핵심 부품 - 보조기억장치)





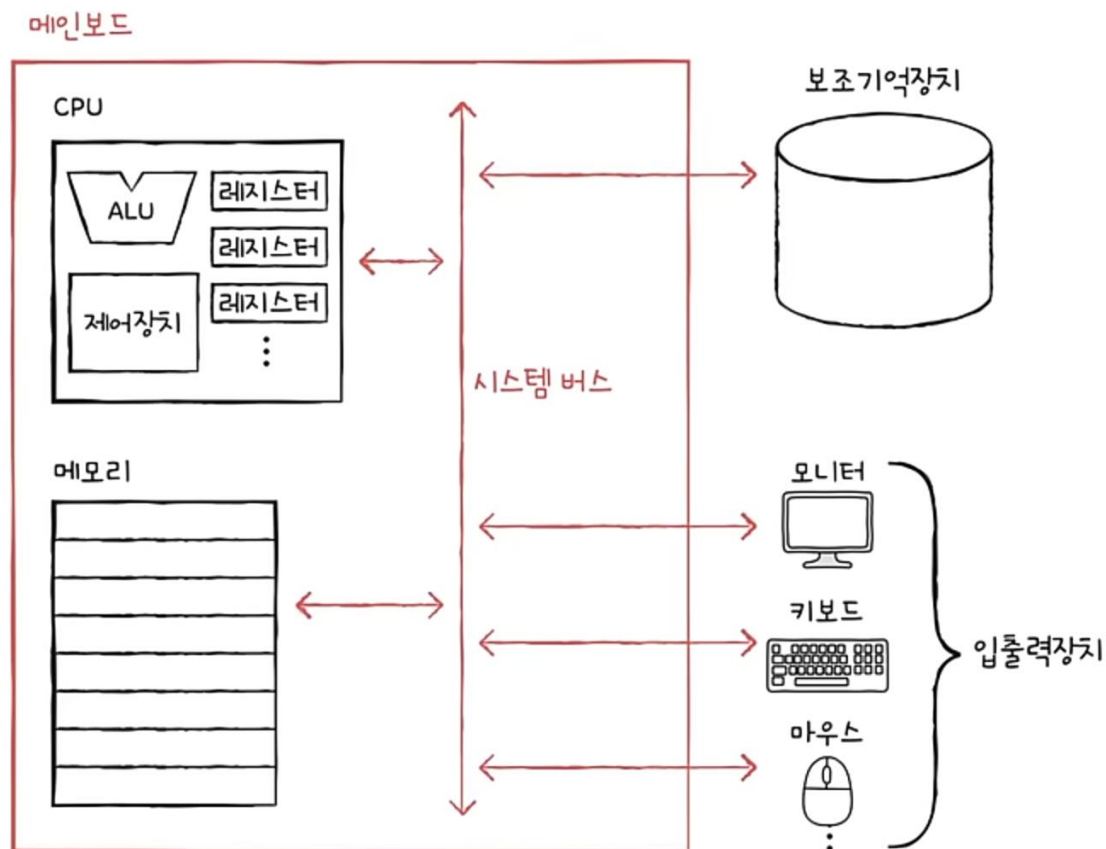
# I. 컴퓨터 구조

## 컴퓨터 구조 (4가지 핵심 부품 - 입출력 장치)



# I. 컴퓨터 구조

## 컴퓨터 구조 (메인 보드 & 시스템 버스)



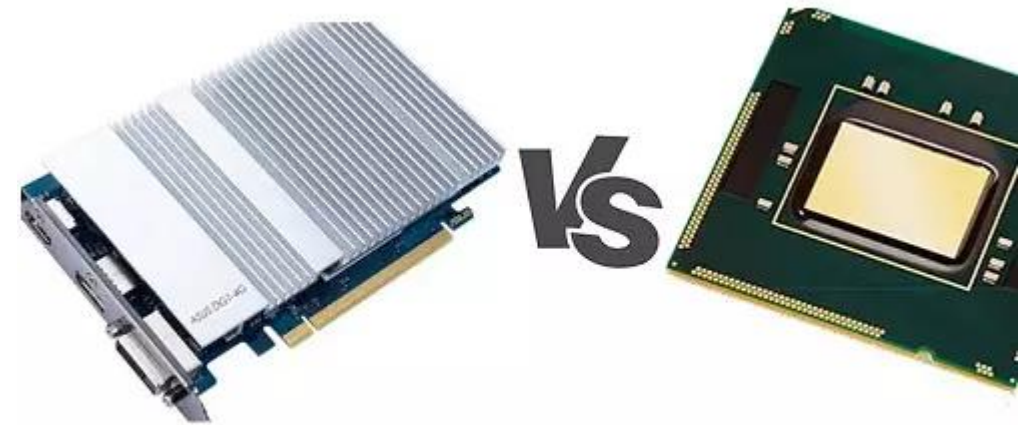
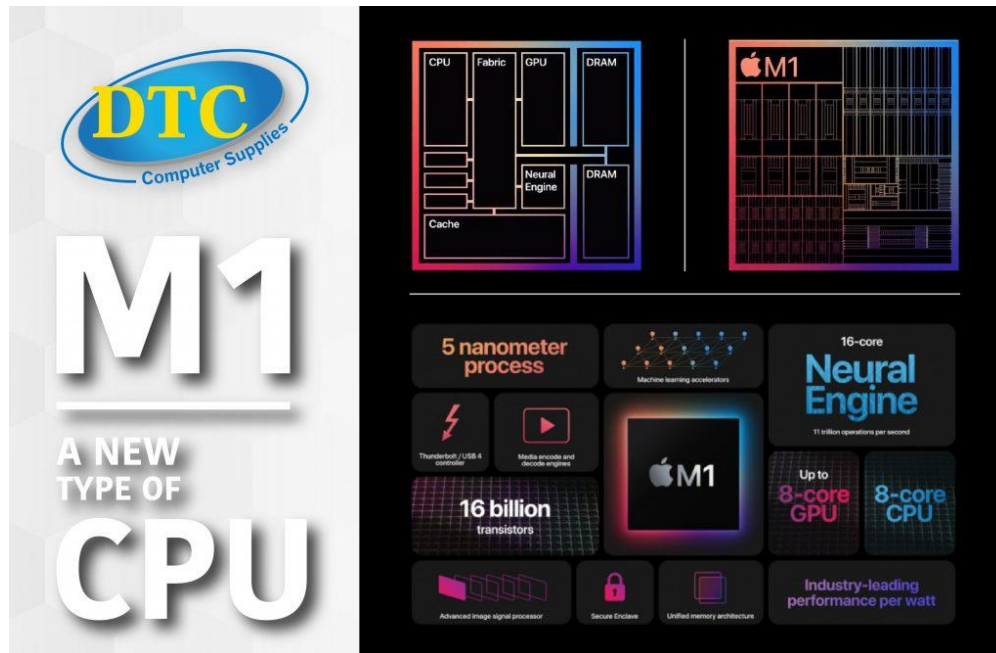
### 메인보드 & (시스템) 버스

- 메인보드에 연결된 부품은 **버스**를 통해 정보를 주고 받음
- 버스는 컴퓨터의 부품끼리 정보를 주고받는 일종의 **통로**
- 다양한 종류의 버스가 있음
- 컴퓨터의 핵심 부품을 연결하는 버스는 **시스템 버스**

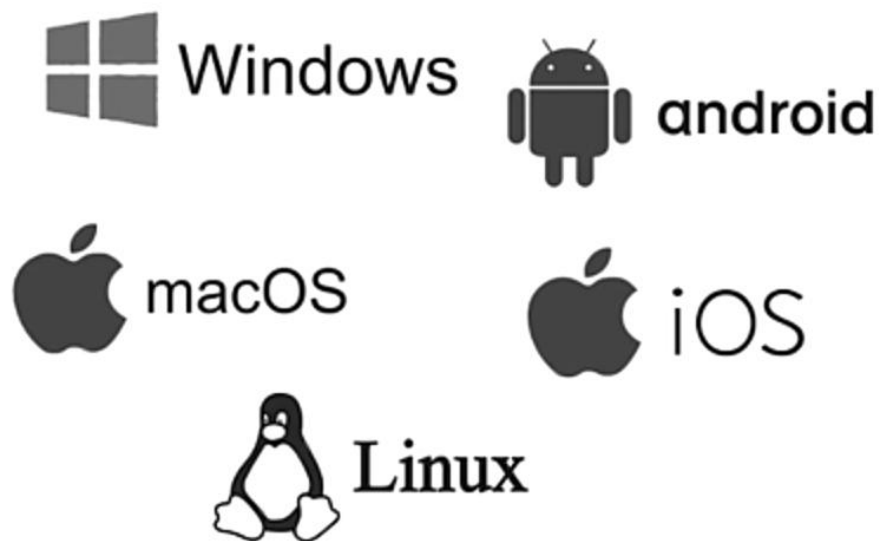
# I. 컴퓨터 구조

## 컴퓨터 구조 (GPU)

**Graphics Processing Unit** - 다양한 장치에서 컴퓨터 그래픽 및 이미지 처리 속도를 높이도록 설계된 전자 회로, 내장 GPU 와 외장 GPU (discrete GPU) 로 크게 나뉨



## II. 운영체제 (OS)



운영체제는 무엇이고,  
개발자는 왜 운영체제를  
알아야 할까?



## II. 운영체제 (OS)

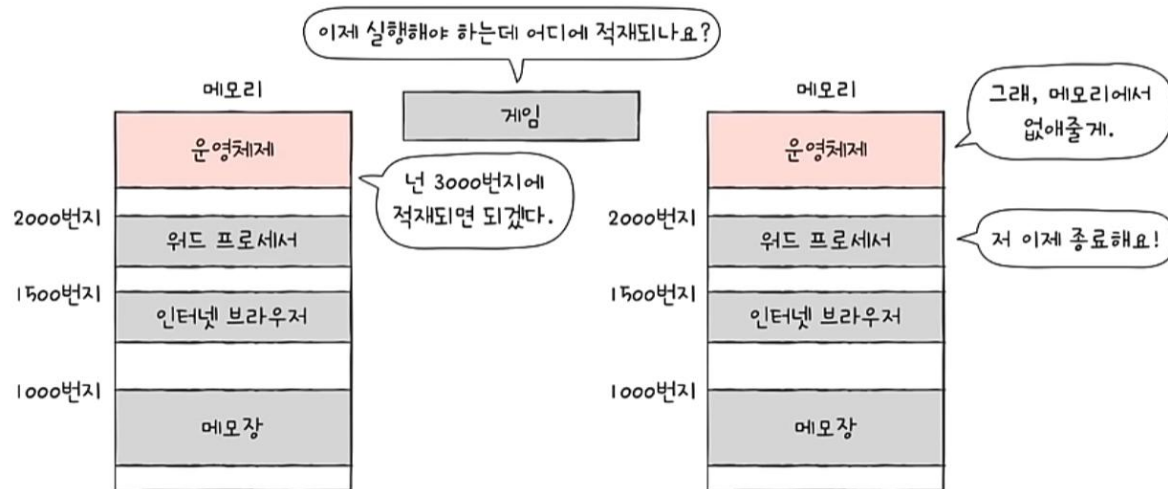
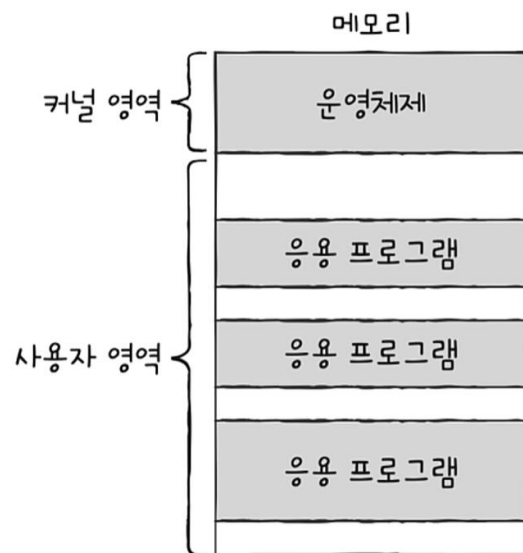
자원을 관리하는 특별한 프로그램

프로세스를 관리하는 특별한 프로그램

(시스템) 자원

- 프로그램이 실행되기 위해 마땅히 필요한 요소
- 컴퓨터의 네 가지 핵심 부품 포함

### 운영체제의 자원(메모리) 관리

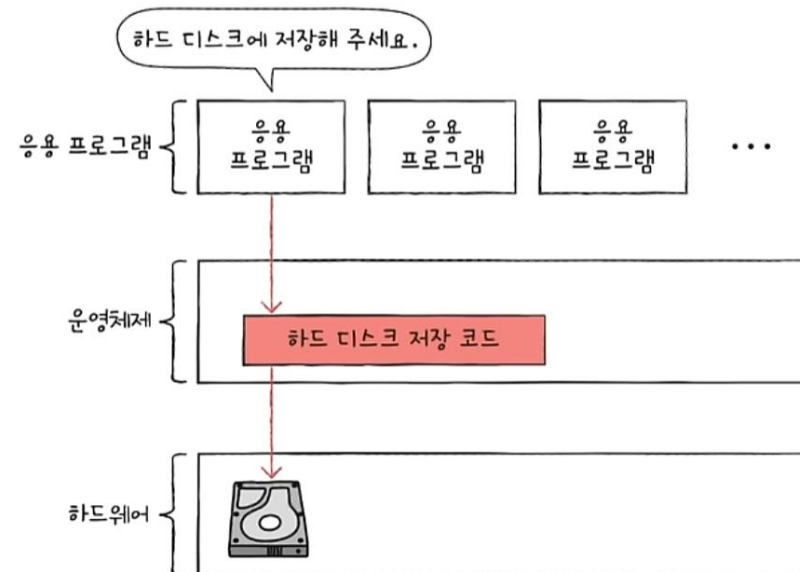
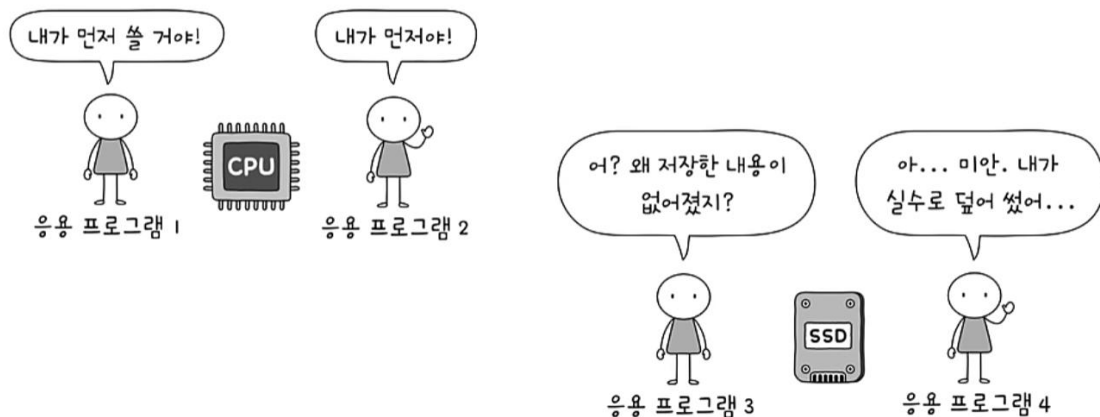


# II. 운영체제 (OS)

자원을 관리하는 특별한 프로그램

프로세스를 관리하는 특별한 프로그램

문지기 역할(시스템 호출)을 통한 자원 보호

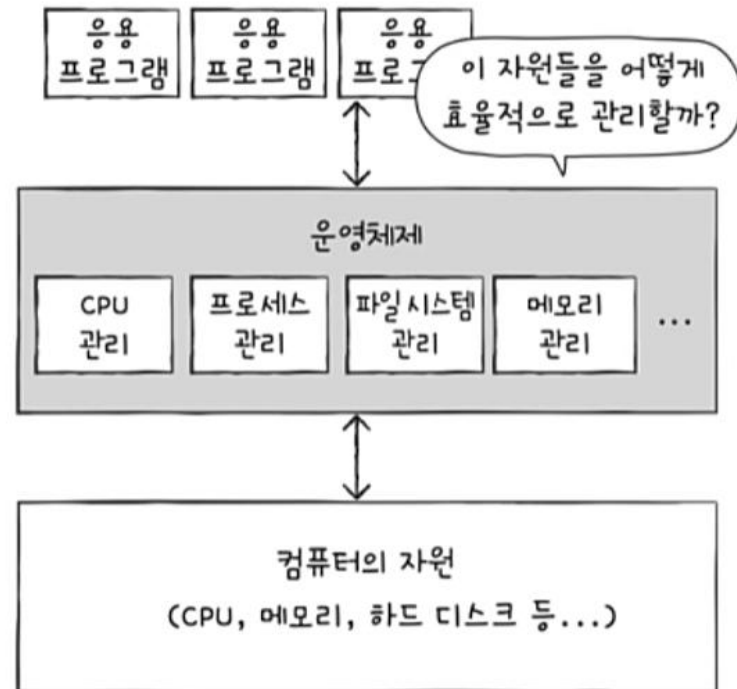
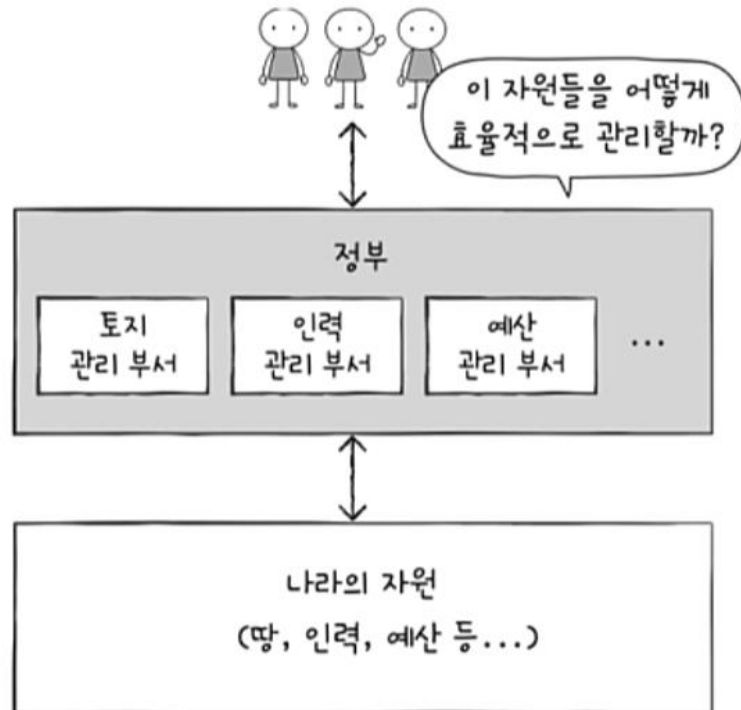


## II. 운영체제 (OS)

자원을 관리하는 특별한 프로그램

프로세스를 관리하는 특별한 프로그램

마치 정부와도 같은..

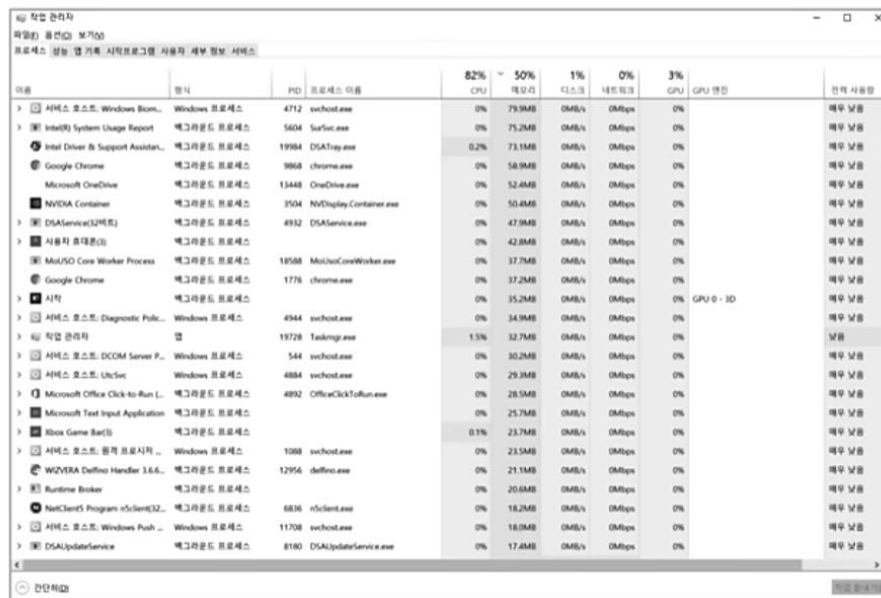


# II. 운영체제 (OS)

자원을 관리하는 특별한 프로그램

프로세스를 관리하는 특별한 프로그램

Q. 이렇게 많은 프로그램들이 동시에 실행되는데,  
누가 일목요연하게 실행을 관리해주지?



작업 관리자 (Task Manager) 화면 캡처. 현재 실행 중인 프로세스 목록과 CPU, 메모리, 디스크, 네트워크, GPU 사용량을 보여줍니다.

이름	원격	PID	프로세스 이름	CPU	메모리	디스크	네트워크	GPU	GPU 엔진	작업 사용량
서비스 호스트: Windows B...	Windows 프로세스	4712	svchost.exe	0%	79.9MB	0MB/s	0Mbps	0%		매우 낮음
Intel(R) System Usage Report	백그라운드 프로세스	5604	SurSvc.exe	0%	75.2MB	0MB/s	0Mbps	0%		매우 낮음
Intel Driver & Support Assistan...	백그라운드 프로세스	19984	DSATray.exe	0.2%	71.1MB	0MB/s	0Mbps	0%		매우 낮음
Google Chrome	백그라운드 프로세스	9868	chrome.exe	0%	58.9MB	0MB/s	0Mbps	0%		매우 낮음
Microsoft OneDrive	백그라운드 프로세스	13488	OneDrive.exe	0%	52.8MB	0MB/s	0Mbps	0%		매우 낮음
NVIDIA Container	백그라운드 프로세스	3704	NVDisplayContainer.exe	0%	50.8MB	0MB/s	0Mbps	0%		매우 낮음
DSAService(32비트)	백그라운드 프로세스	4932	DSAService.exe	0%	47.9MB	0MB/s	0Mbps	0%		매우 낮음
사용자 휴대전화	백그라운드 프로세스			0%	42.8MB	0MB/s	0Mbps	0%		매우 낮음
Malware Core Worker Process	백그라운드 프로세스	18588	MalwareCoreWorker.exe	0%	37.7MB	0MB/s	0Mbps	0%		매우 낮음
Google Chrome	백그라운드 프로세스	1776	chrome.exe	0%	37.2MB	0MB/s	0Mbps	0%		매우 낮음
시작	백그라운드 프로세스			0%	35.2MB	0MB/s	0Mbps	0%	GPU 0 - 3D	매우 낮음
서비스 호스트: Diagnostic Polic...	Windows 프로세스	4944	svchost.exe	0%	34.9MB	0MB/s	0Mbps	0%		매우 낮음
작업 관리자	임	19728	Taskmgr.exe	1.5%	32.7MB	0MB/s	0Mbps	0%		낮음
서비스 호스트: DCOM Server P...	Windows 프로세스	544	svchost.exe	0%	30.2MB	0MB/s	0Mbps	0%		매우 낮음
서비스 호스트: UtcSvc	Windows 프로세스	4884	svchost.exe	0%	29.3MB	0MB/s	0Mbps	0%		매우 낮음
Microsoft Office Click-to-Run L...	백그라운드 프로세스	4892	OfficeClickToRun.exe	0%	28.5MB	0MB/s	0Mbps	0%		매우 낮음
Microsoft Text Input Application	백그라운드 프로세스			0%	25.7MB	0MB/s	0Mbps	0%		매우 낮음
Xbox Game Bar	백그라운드 프로세스			0.1%	23.7MB	0MB/s	0Mbps	0%		매우 낮음
서비스 호스트: 원격 프로시저 ...	Windows 프로세스	1088	svchost.exe	0%	23.5MB	0MB/s	0Mbps	0%		매우 낮음
WIZVERA Dellino Handler 3.6.6...	백그라운드 프로세스	12956	dellino.exe	0%	21.1MB	0MB/s	0Mbps	0%		매우 낮음
Runtime Broker	백그라운드 프로세스			0%	20.6MB	0MB/s	0Mbps	0%		매우 낮음
NetClient5 Program xSClient32...	백그라운드 프로세스	6836	xSClient.exe	0%	18.2MB	0MB/s	0Mbps	0%		매우 낮음
서비스 호스트: Windows Push ...	Windows 프로세스	11708	svchost.exe	0%	18.0MB	0MB/s	0Mbps	0%		매우 낮음
DSAUpdateService	백그라운드 프로세스	8180	DSAUpdateService.exe	0%	17.6MB	0MB/s	0Mbps	0%		매우 낮음

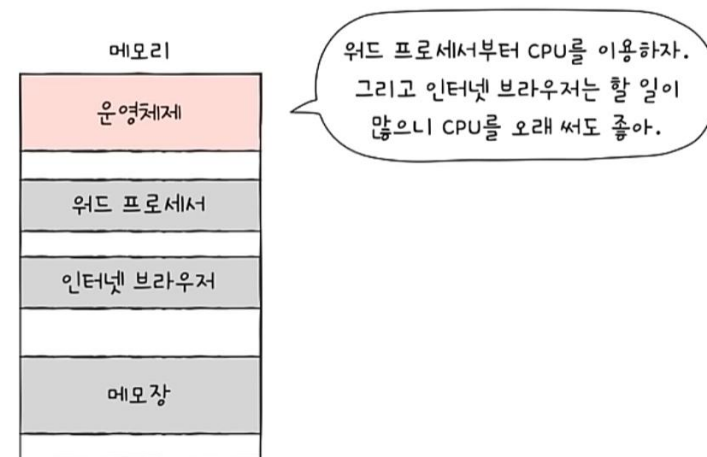
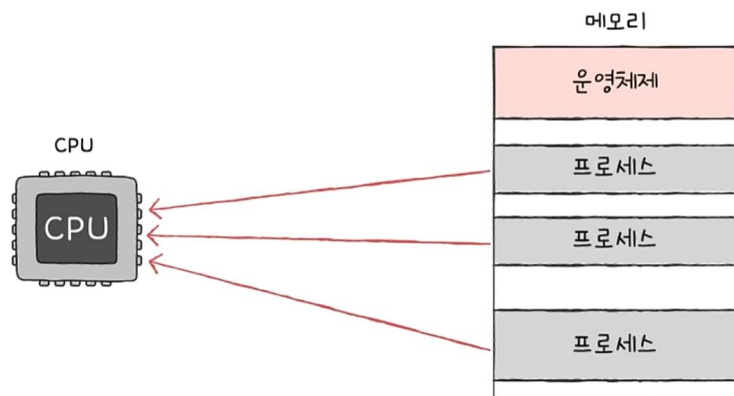


# II. 운영체제 (OS)

자원을 관리하는 특별한 프로그램

**프로세스**를 관리하는 특별한 프로그램

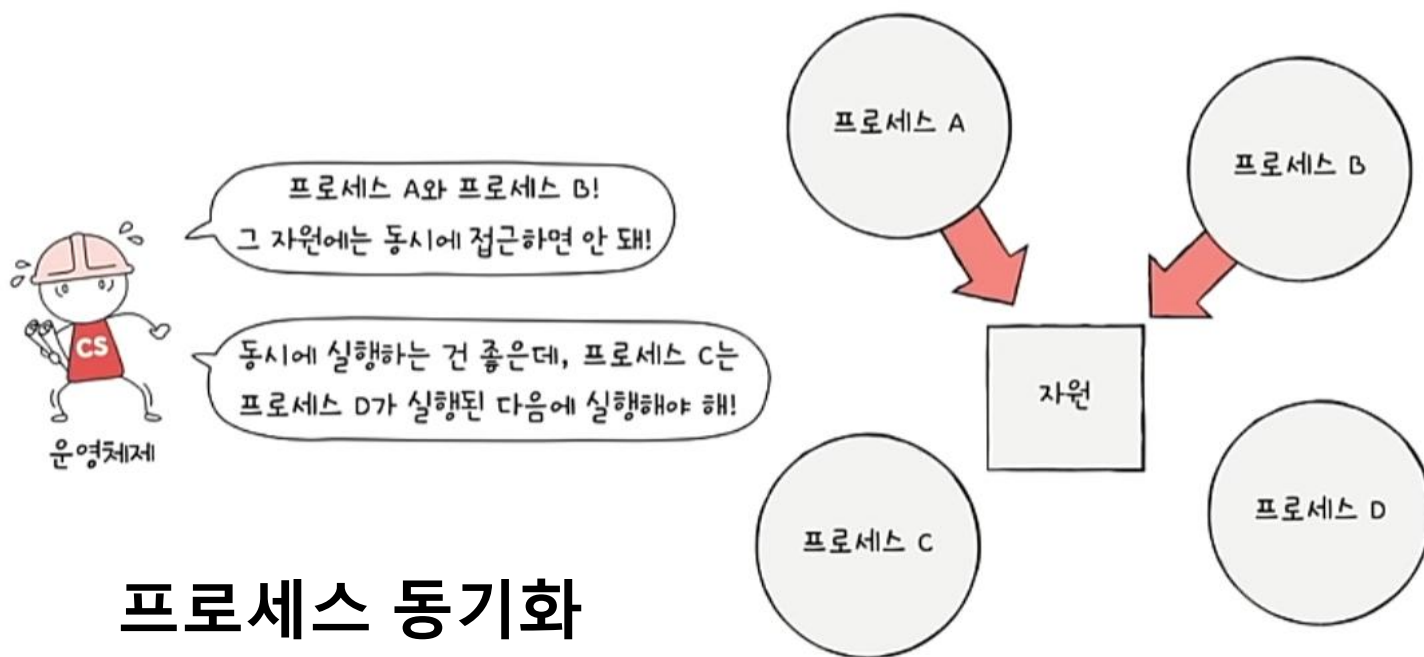
Q. 어떤 프로세스를 먼저, 얼마나 오래 실행할까?



## II. 운영체제 (OS)

자원을 관리하는 특별한 프로그램

프로세스를 관리하는 특별한 프로그램

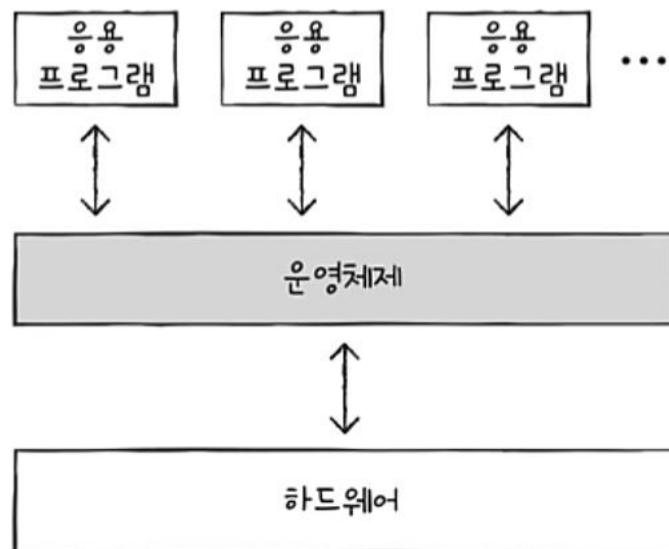


프로세스 동기화

# II. 운영체제 (OS)

## 정리

- 운영체제는 컴퓨터를 동작하는데 도움을 주는 SW
- HW 와 SW 사이의 매개체 역할
- 운영체제의 역할
  - 사용자를 위한 역할: 사용자에게 컴퓨터를 손쉽게 이용할 수 있는 인터페이스 제공
  - 하드웨어를 위한 역할:
    - 프로세스 관리 - 컴퓨터에서 프로세스들을 연속적으로 실행 가능하게
    - 메모리 관리 - 한정된 메모리 공간에 프로그램 메모리 할당
    - HW 관리 - 사용자의 직접 접근 저지, 파일 효율적으로 관리
- 운영체제의 구조
  - Kernel: 운영체제의 가장 핵심적인 부분으로 프로세스, 메모리, 저장장치 관리로 커널 자체를 좁은 의미에서의 OS 라고 부르기도 함
  - 인터페이스: 사용자가 커널에 직접 접근 할 수 없어 인터페이스(System Call) 를 통해 커널에 명령을 전달하고 결과를 전달받음



# II. 운영체제 (OS)

## Playground

- 프로세스 관리

```
ps aux # process status(프로세스 상태)를 보여주는 명령어 (a:모든, u: 자세히, x: 터미널 비종속 프로세스도 표시)
htop   # 실시간으로 프로세스/시스템 자원(CPU, 메모리 등) 사용 현황을 그래픽(터미널 GUI) 형태로 보여주는 도구
```

```
sleep 100 &
ps aux | grep sleep
kill <PID>
```

- 메모리 관리

```
free -h # 메모리 사용량 확인
vmstat 1 # "Virtual Memory Statistics - 메모리, 프로세스, 스왑, IO, 시스템, CPU 사용 현황을 요약, 1초단위
```

# III. Kernel

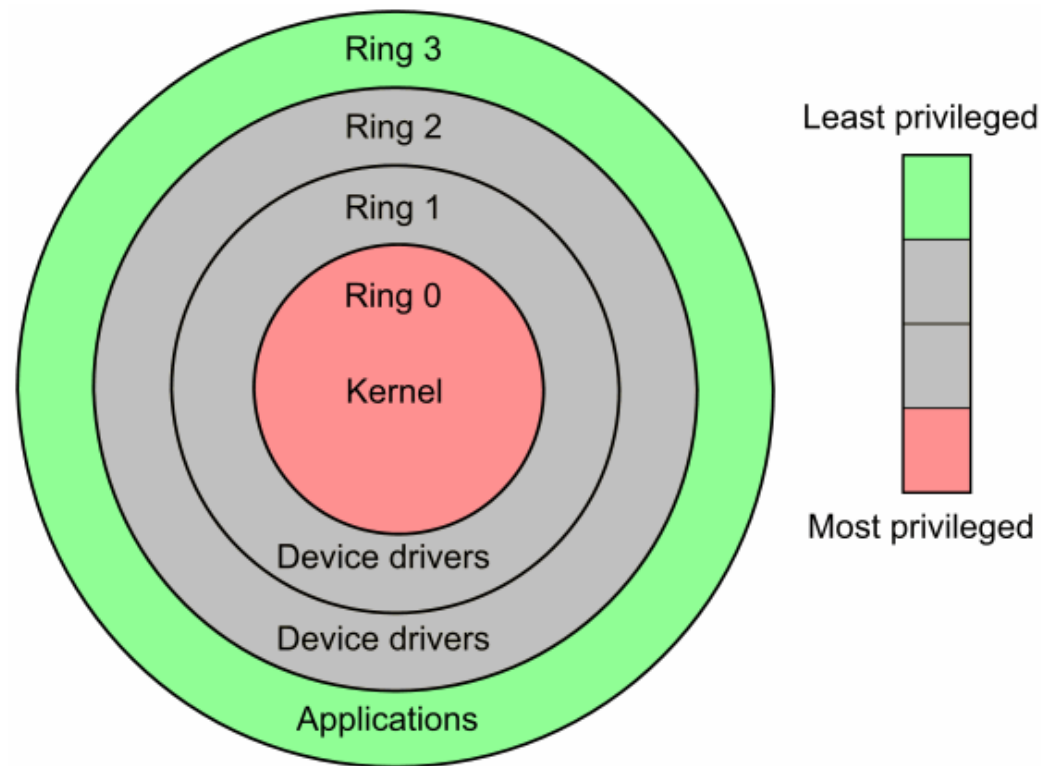
## Kernel 이란?

- 커널(Kernel)은 운영체제의 핵심 부분으로, 하드웨어와 직접 상호작용하며 OS의 주요 기능들을 담당하는 중앙 관리 프로그램
- 운영체제 전체가 집이라면 커널은 그 토대(핵심)에 해당하며, 응용프로그램이 요청하는 저 수준 작업 들을 수행
- 커널은 일반적으로 커널 모드에서 동작하며, 사용자 프로그램은 유저 모드에서 동작하여 커널에 직접 접근하지 못하게 격리
- 커널의 역할:
  - **메모리 관리**: 시스템의 메모리 사용 현황을 추적하고, 프로세스들에게 메모리를 할당/회수하며, 페이지 교체 등 의 메커니즘을 통해 효율적으로 메모리를 관리
  - **프로세스 관리**: 프로세스 생성과 종료, CPU 스케줄링, 프로세스 간 통신(IPC) 등을 수행하여 여러 프로그램이 원활하게 CPU를 공유
  - **장치 드라이버 제어**: 각종 하드웨어 장치에 대한 디바이스 드라이버를 관리하고, 응용프로그램의 입출력 요청을 해당 드라이버를 통해 하드웨어에 전달. 커널은 하드웨어와 프로세스 간의 중재자로서 드라이버를 통해 장치를 제어
  - **시스템 호출 처리 및 보안**: 사용자 프로세스가 커널 기능을 요청할 수 있도록 시스템 콜 인터페이스를 제공하고, 이 때 권한 체크 및 보안을 적용. 커널 모드에서만 수행가능한 작업과 자원 접근을 통제하여 시스템 전체의 안정성을 유지

# III. Kernel

## Kernel mode vs User mode

- **유저모드:**  
일반 프로그램이 동작하는 안전한 영역.  
→ 시스템의 핵심 자원(메모리, 하드웨어 등)에 직접 접근 못함.
- **커널모드:**  
운영체제(커널) 코드가 실행되는 특권 영역.  
→ 하드웨어와 시스템 자원에 직접 접근/제어 가능.



# III. Kernel

## System Call 실습

- System Call 이란?
  - 유저 모드에서 커널 모드로 진입하는 공식 통로
  - 파일 입출력, 프로세스 관리, 메모리 할당 등 주요 OS 기능은 반드시 system call 을 통해 요청
  - strace 는 리눅스에서 프로그램이 어떤 시스템콜(system call)을 호출하는지 실시간으로 보여주는 명령어/도구로 이를 통해 System Call 이 어떻게 호출되는지 확인 가능함
- strace 를 활용해서 리눅스 명령어 따라가 보기

```
strace ls
```

```
strace ps aux
```

`strace -c ls` : 시스템콜별 호출 횟수/시간 요약

`strace -e trace=open,read,write ls` : 파일 열기/읽기/쓰기만 추적

# III. Kernel

## System Call 실습

- strace 를 활용해서 리눅스 명령어 따라가 보기

```
strace cat /etc/hostname
```

```
# openat/write/close system call 만 추적하기, hello 란 text 를 test.txt 에 write 하는 명령어  
strace -e trace=openat,write,close echo "hello" > test.txt
```

```
# network system call 만 추적하기, 해당 주소에 HTTP GET 명령어를 보냄  
strace -e trace=network curl http://seojeong.club:8090/upload
```

```
# sleep 을 100초간 수행하는 process 에 kill signal 을 보내 어떤 system call 이 발생되는지 확인  
strace sleep 100  
# 다른 터미널에서  
kill -SIGINT <sleep의 PID>
```



# III. Kernel

## Ubuntu 에서 커널 정보 확인하기

- 커널 정보 출력하는 명령어 – uname

```
uname -r # 실행 중인 커널의 버전 번호를 출력  
uname -a # 커널 버전뿐 아니라 시스템의 전체 정보를 출력  
uname -v # 커널 컴파일 시각 및 빌드 정보
```

Tip: uname -r 로 출력된 버전과 일치하는 설정 파일이 /boot 디렉토리에 존재함 (예: /boot/config-5.15.0-75-generic ), 이게 무엇인지 확인해 보기

- 가상 파일 시스템 /proc 아래 명령어

```
/proc/version # 커널 버전과 빌드 정보  
/proc/meminfo # 메모리 용량 및 사용 현황  
/proc/cpuinfo # CPU 모델, 코어 수 등 프로세서 정보  
/proc/sys/kernel/hostname # Host 이름  
/proc/sys/kernel/osrelease # OS 릴리스 버전
```

# IV. Ubuntu Kernel 빌드/사용

## Kernel Source 다운로드 받는 방법

### 1. apt 소스 패키지 이용 (Ubuntu 공식 커널 소스)

- 가장 Ubuntu 친화적이며, 배포된 커널의 정확한 소스를 받기 적합한 방식이나 최신 버전은 아님

```
sudo apt install linux-source # 커널 소스 받기
cp /usr/src/linux-source-*.tar.bz2 ./ # 현재 디렉토리로 다운로드 된 소스 복사
tar xjf linux-source-*.tar.bz2 && cd linux-source-* # 압축 해제 및 커널 소스로 이동
```

### 2. [git.kernel.org](https://git.kernel.org) (메인라인 커널)

- 리눅스 커널의 공식 저장소에서 직접 clone 하여 다운로드하는 방식으로 최신 커널 혹은 특정 버전이 필요할 때 사용

```
git clone https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
cd linux && git checkout v6.1 # 커널 소스 디렉토리로 이동 후 특정 버전으로 checkout
```

### 3. Ubuntu 패치된 커널 git 저장소 이용 (Canonical/Ubuntu 자체 커널)

- Ubuntu 에서 직접 관리하는 git 저장소를 이용하는 방식으로 정확히 Ubuntu 에서 사용하는 커널(패치/모듈 포함)로 작업 하고 싶을 때

```
git clone https://git.launchpad.net/~ubuntu-kernel/ubuntu/+source/linux/+git/jammy
cd jammy
git tag | grep 6.8
git checkout Ubuntu-hwe-6.8-6.8.0-62.65_22.04.1
```

# IV. Ubuntu Kernel 빌드/사용

## Kernel 빌드하기 (3번째 Ubuntu 패치된 커널 git 저장소 이용하는 경우)

- 커널 빌드를 위한 필수 package 설치하기

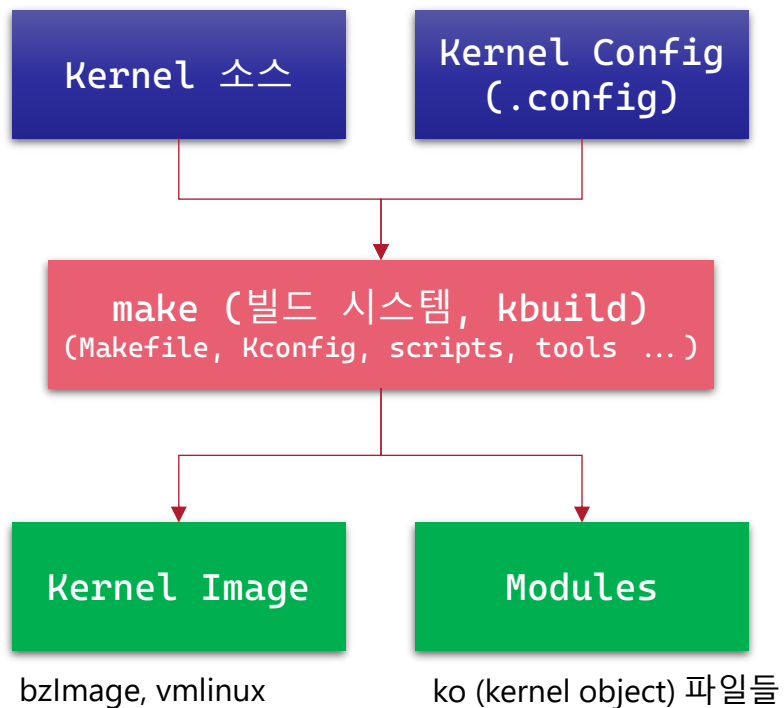
```
sudo apt update
sudo apt install -y git build-essential libncurses-dev bison flex libssl-dev libelf-dev
sudo apt install -y fakeroot bc cpio rsync debhelper-compat zstd
```

- 기존 kernel config 복사

```
cd ../jammy
cp /boot/config-$(uname -r) my-config
```

# IV. Ubuntu Kernel 빌드/사용

## Kernel Source 빌드 Flow



- 커널 소스: 리눅스의 전체 소스 코드
- 커널 설정 파일: 빌드할 때 어떤 기능을 켜고/끄는지 결정하는 파일 (예: CONFIG\_USB=y)
- make 시스템: 설정 파일에 따라 실제로 필요한 코드만 빌드
- 최종 산출물:
  - 커널 이미지 (부팅용 바이너리)
  - 커널 모듈 (필요시 로드하는 .ko 파일)

# IV. Ubuntu Kernel 빌드/사용

## Kernel Configuration (커널 설정파일, .config) 이란?

- 커널 빌드 시 필요한 기능/드라이버/옵션의 활성화 여부를 결정하는 설정 파일
- 파일명은 보통 `.config` 으로 빌드 디렉토리에 생성됨
- 각 항목은 `CONFIG_` 로 시작됨
- 설정 방법
  - `menuconfig` `xconfig`, `gconfig`, `nconfig`
  - Example: `make menuconfig`
  - 커널 설정 (.config) 의 문법

표기	의미
<code>=y</code>	해당 기능을 커널에 내장 (built-in)
<code>=m</code>	해당 기능을 모듈로 빌드 (.ko 파일)
<code>=n</code>	해당 기능을 비활성화 (불포함)

# IV. Ubuntu Kernel 빌드/사용

## Kernel Configuration 수정할 부분

CONFIG\_MODULE\_SIG=n

CONFIG\_MODULE\_SIG\_ALL=n

CONFIG\_MODULE\_SIG\_SHA52=n

CONFIG\_MODULE\_SIG\_KEY=""

CONFIG\_SYSTEM\_TRUSTED\_KEYS=""

CONFIG\_SYSTEM\_REVOCATION\_KEYS=""

CONFIG\_DEBUG\_INFO=n

CONFIG\_DEBUG\_INFO\_DWARF5=n

# IV. Ubuntu Kernel 빌드/사용

## Kernel 빌드하기 (3번째 Ubuntu 패치된 커널 git 저장소 이용하는 경우)

- 빌드 스크립트 작성 (kernel\_build.sh)

```
#!/bin/bash

KERNEL_VERSION=${1:-"6.1.80"}           # 1번 인자: 커널 버전 (기본값 6.1.80)
SRC_DIR=${2:-"$PWD/jammy"}              # 2번 인자: 소스 디렉토리
BUILD_DIR=${3:-"$PWD/linux-build-$KERNEL_VERSION"} # 3번 인자: 빌드 디렉토리
CONFIG_FILE=${4:-"$PWD/my-config"}      # 4번 인자: config 파일 경로
MAKE_OPTS=${5:-"-j$(nproc) KCFLAGS=-Wno-error deb-pkg"} # 5번 인자: make 옵션(기본: 모든 CPU 사용)

mkdir -p "$BUILD_DIR"
export KBUILD_OUTPUT="$BUILD_DIR"

if [ -f "$CONFIG_FILE" ]; then
    cp "$CONFIG_FILE" "$BUILD_DIR/.config"
    cd "$SRC_DIR"
    make olddefconfig
else
    echo "Config 파일이 없습니다."
    exit
fi

make $MAKE_OPTS 2>&1 | tee build.log
cd ../
mkdir -p deb_pkg && mv *.deb deb_pkg/
rm -f *.buildinfo *.changes *.dsc *.debian.tar.gz *.orig.tar.gz *.tar.bz2
```

# IV. Ubuntu Kernel 빌드/사용

## Kernel Source 빌드 산출물

파일명/확장자	의미 및 용도
<b>vmlinux</b>	디버깅용 <b>커널 전체 ELF 바이너리</b> (압축x, 심볼 포함). 디버깅/심볼 분석용
<b>bzImage</b>	부팅에 사용하는 <b>커널 부트 이미지</b> . 압축된 커널 실행파일 (bootloader가 읽음)
zImage	bzImage와 유사하나, 주로 임베디드/ARM 환경에서 사용되는 압축 커널 이미지
System.map	커널 함수/변수 등의 <b>심볼 테이블 목록</b> . 커널 디버깅/오류 분석에 활용
<b>.config</b>	실제 빌드에 사용된 <b>커널 설정 파일</b> (enable된 기능/드라이버 등)
initrd, initramfs	초기 램디스크 이미지. 부팅 시 root filesystem 준비용
<b>.ko</b> (예: e1000.ko)	개별 <b>커널 모듈 파일</b> . 드라이버 등, 필요한 경우에만 로딩되는 동적 모듈
arch/x86/boot/bzImage	x86용 bzImage 파일의 실제 경로
include/, arch/, drivers/, ...	빌드 중 생성되는 각종 임시/오브젝트/헤더 파일들



# IV. Ubuntu Kernel 빌드/사용

## Kernel Source 빌드 후 설치하기 (deb install or manual)

### deb 설치

빌드 완료 후 아래와 같은 deb 파일들이 생성:

- linux-image-\*.deb : 커널 이미지
- linux-headers-\*.deb : 헤더
- linux-libc-dev-\*.deb : 라이브러리 헤더
- (Optional) linux-image-\*.dbgsym.deb : 디버그 심볼

장점: GRUB 부트메뉴가 자동 업데이트 되며, dpkg가 커널 이미지(/boot/vmlinuz-\*), initramfs, System.map, 모듈(/lib/modules/\*) 자동 설치, 관리 용이

```
sudo dpkg -i linux-image-*.deb linux-headers-*.deb
sudo reboot
```

### Manual 설치

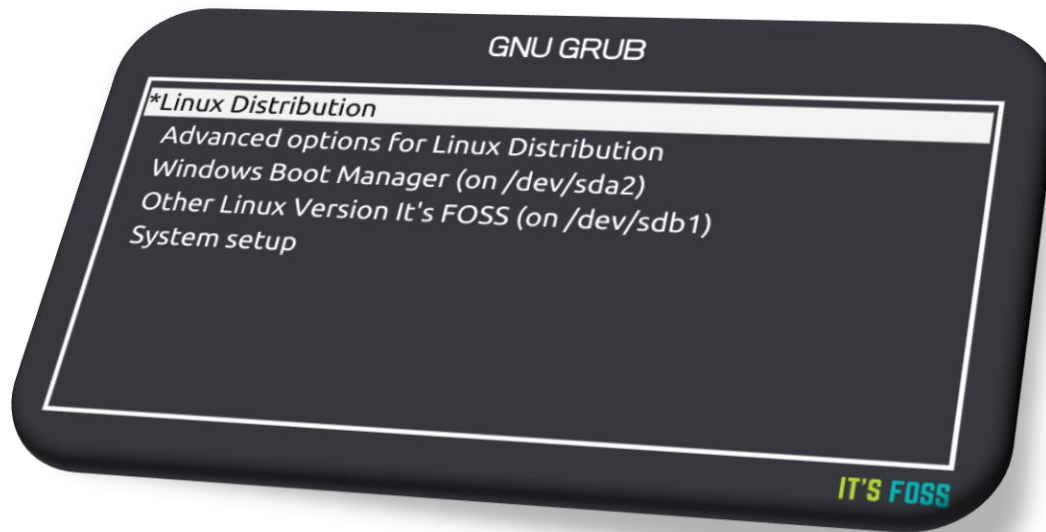
배포 패키지 없이 커널 이미지와 모듈만 직접 복사해서 설치  
패키지 관리 시스템에 등록되지 않아 자동 관리/제거가 어렵고, 실수 시 복구가 번거로움

```
sudo cp arch/x86/boot/bzImage /boot/vmlinuz-<custom-version>
sudo make modules_install # 자동으로 /lib/modules/<custom-version>/ 아래로 복사
sudo cp System.map /boot/System.map-<custom-version>
sudo cp .config /boot/config-<custom-version>
sudo update-initramfs -c -k <custom-version> # initramfs (초기 램 디스크 이미지) 생성
sudo update-grub
sudo reboot
```

# V. GRUB

## GRUB 이란?

- GRUB (Grand Unified Bootloader)
  - 리눅스, 윈도우 등 여러 OS의 부팅을 관리하는 범용 부트 로더
  - 다중 부트, 커널 파라미터 전달, 복구 모드 진입 등에 사용



# V. GRUB

## GRUB menu 로 진입하는 방법

- (Option #1) 부팅 시 GRUB 메뉴가 자동으로 나오지 않을 경우
  - 부팅 중 Shift 또는 Esc 키를 반복 입력하여 GRUB 메뉴로 진입
  - 타이밍이 잘 맞으면 GRUB 메뉴 진입 성공
- (Option #2) /etc/default/grub 수정으로 항상 GRUB 메뉴 진입

```
sudo vi /etc/default/grub
```

```
# GRUB_TIMEOUT_STYLE=hidden → comment 처리
```

```
GRUB_TIMEOUT=5 # 5 초간 GRUB menu 활성화 후 별도 입력 없으면 default 설정으로 부팅
```

```
sudo update-grub
```

# V. GRUB

## GRUB 주요 옵션

옵션 명-+	-
GRUB_TIMEOUT	메뉴 노출 대기 시간(초)
GRUB_DEFAULT	기본 선택 메뉴(0=첫번째, saved=마지막 부팅)
GRUB_CMDLINE_LINUX	모든 커널에 추가 전달할 커널 파라미터
GRUB_DISABLE_RECOVERY	복구(recovery) 메뉴 비활성화

```
GRUB_DEFAULT=saved
GRUB_SAVEDEFAULT="true"
GRUB_TIMEOUT=3
#GRUB_TIMEOUT_STYLE=hidden
```



## 2. 커널이 사용하는 메모리 영역 확인법

### A. 메모리 맵(boot-time)

시스템 부팅 시 커널이 어느 영역을 차지하는지는 `/proc/iomem` 에서 일부 확인할 수 있습니다.

```
bash
cat /proc/iomem
```

- 출력에서 `Kernel code`, `Kernel data`, `Kernel bss` 같은 라인이 보일 수 있음(최근 커널은 이 정보 : 한적).

### B. `/proc/kallsyms`

커널 심볼 주소를 확인할 수 있습니다.

```
bash
cat /proc/kallsyms | grep ' T _stext'
cat /proc/kallsyms | grep ' T _etext'
```

- `_stext` : 커널 코드 시작 주소
- `_etext` : 커널 코드 끝 주소

비슷하게,

```
bash
cat /proc/kallsyms | grep ' T _sinittext'
cat /proc/kallsyms | grep ' T _einittext'
```

- init text 영역(초기화 코드) 범위도 알 수 있음

예시

```
r
ffffff81000000 T _stext
ffffff81c49000 T _etext
```

이렇게 나오면,

- 커널 코드 : `0xffffffff81000000 ~ 0xffffffff81c49000`

### C. `/boot/System.map-$(uname -r)`

심볼맵 파일로 동일 정보 확인 가능

```
bash
grep _stext /boot/System.map-$(uname -r)
grep _etext /boot/System.map-$(uname -r)
```

## 3. 실제 메모리 매핑 확인

- 유저 프로세스 메모리 맵:  
`/proc/[pid]/maps`  
하지만 커널 영역은 이 파일에 표시되지 않음(유저영역만).
- 전체 가상 메모리 맵:  
커널 소스 분석 및 위 `/proc/kallsyms` 활용

## 5. 정리

- 커널은 유저와 "완전히 분리된" 상위 가상 주소 영역(예: `0xffffffff81000000` 등)에 로드됨
- 실제 커널의 코드 시작/끝 주소는 `/proc/kallsyms` 또는 `/boot/System.map-<커널버전>` 에서 `_stext ~ _etext` 로 확인
- 물리 메모리 내에서 커널이 차지하는 부분은 `/proc/iomem` 참고(단, 최근엔 상세 정보 비공개).

## 요약

- 커널 코드 주소: `/proc/kallsyms` 에서 `_stext`, `_etext`
- 물리 메모리: `/proc/iomem` 에서 "System RAM", "Kernel code" 등
- 가상 주소 맵 구조: x86\_64 기준 유저 0~28TB, 커널 `0xffffffff8000000000000000` 등