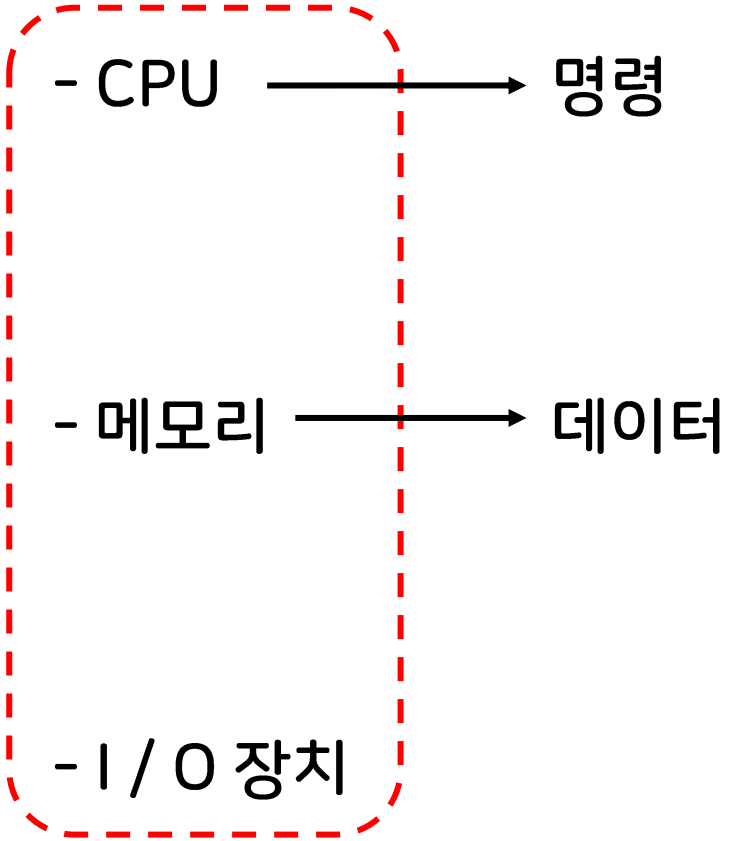
Computer Architecture (컴퓨터 구조)

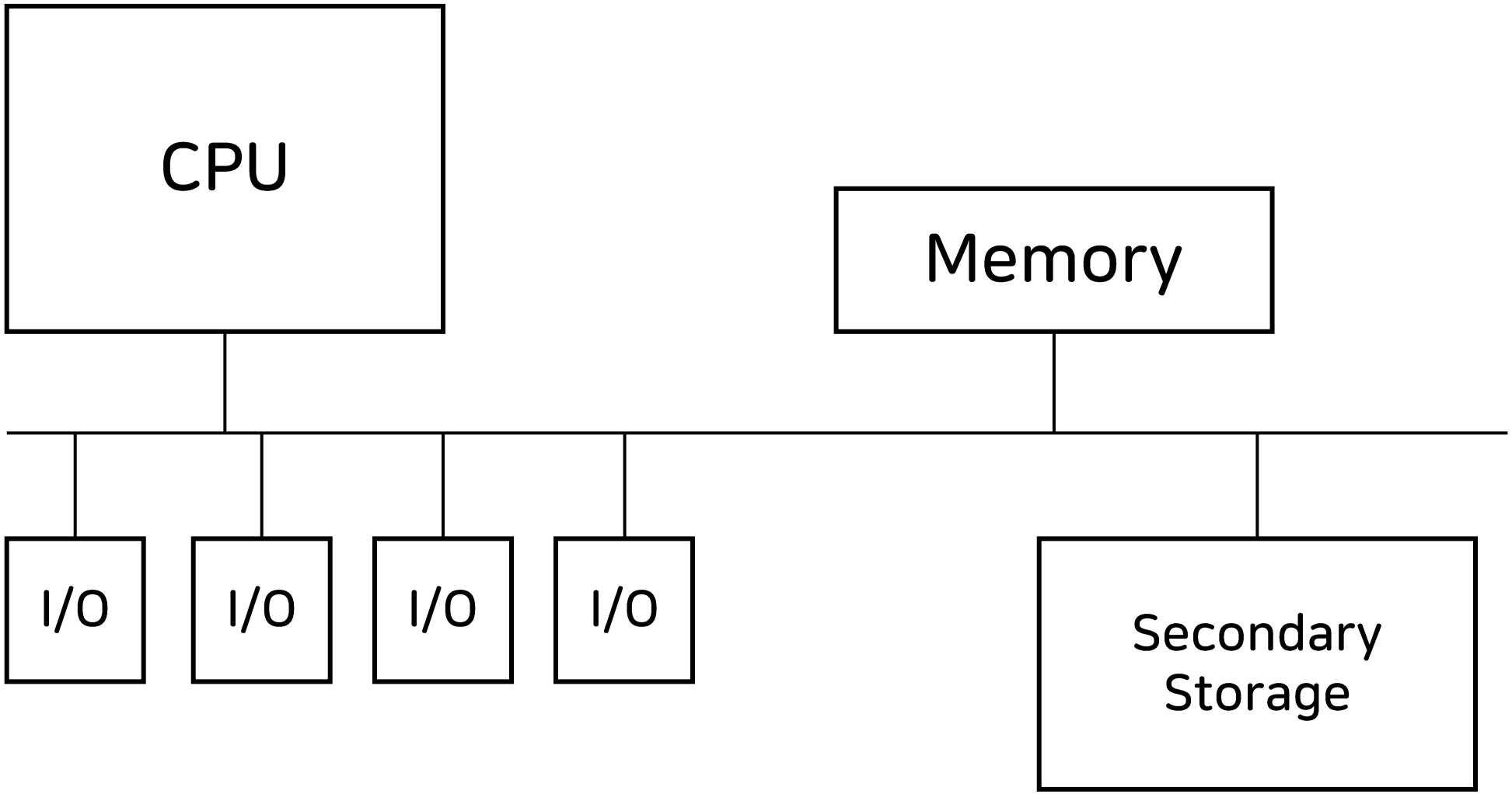
사전 발표 준비 및 전체적인 이론 학습

컴퓨터구조

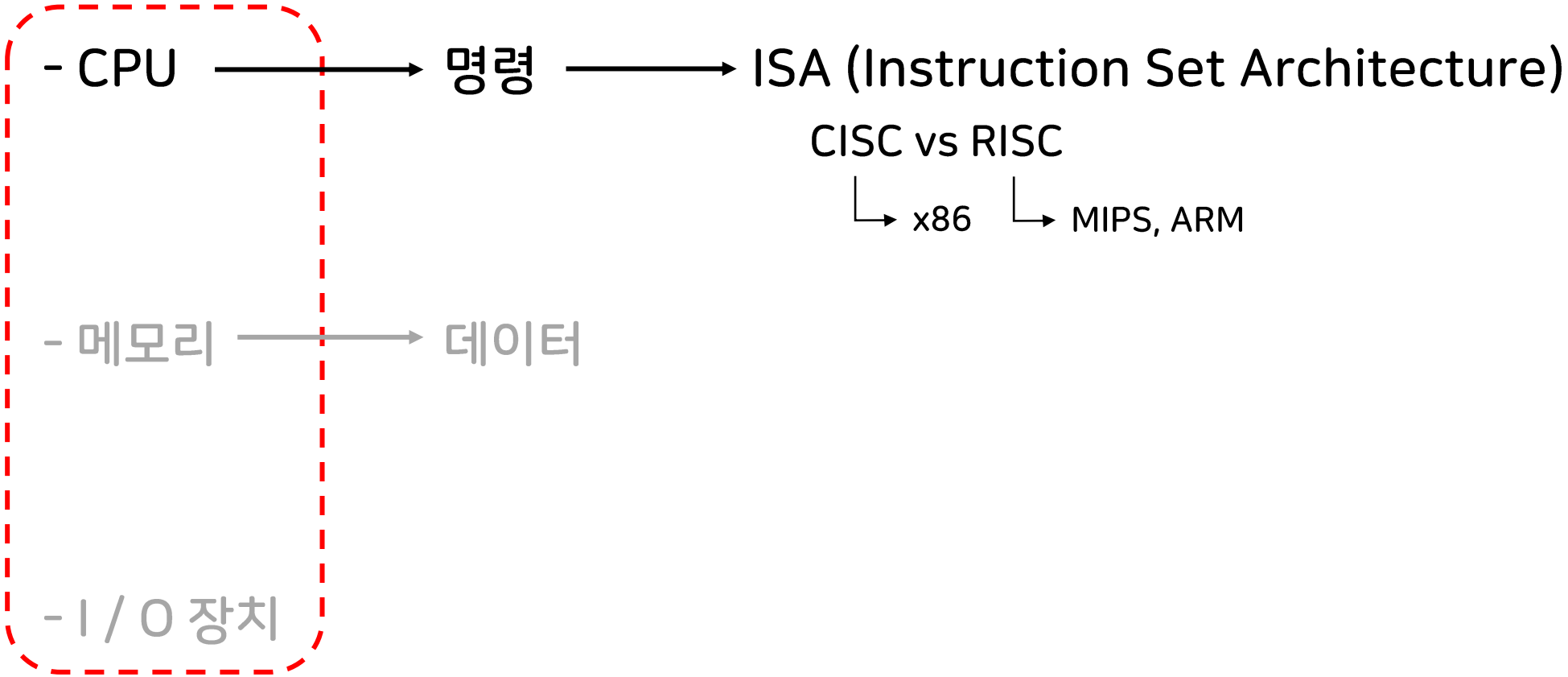
Computer

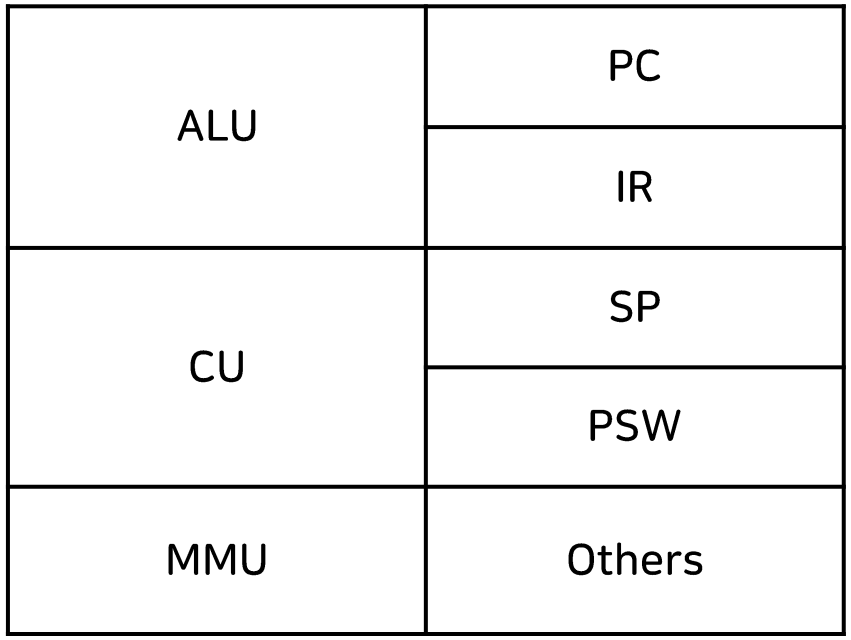


Computer Architecture – Von Neumann vs Harvard Architecture



Computer


- CPU
명령
ISA (Instruction Set Architecture)
CISC vs RISC
x86
MIPS, ARM
- 메모리
데이터
- I / O 장치



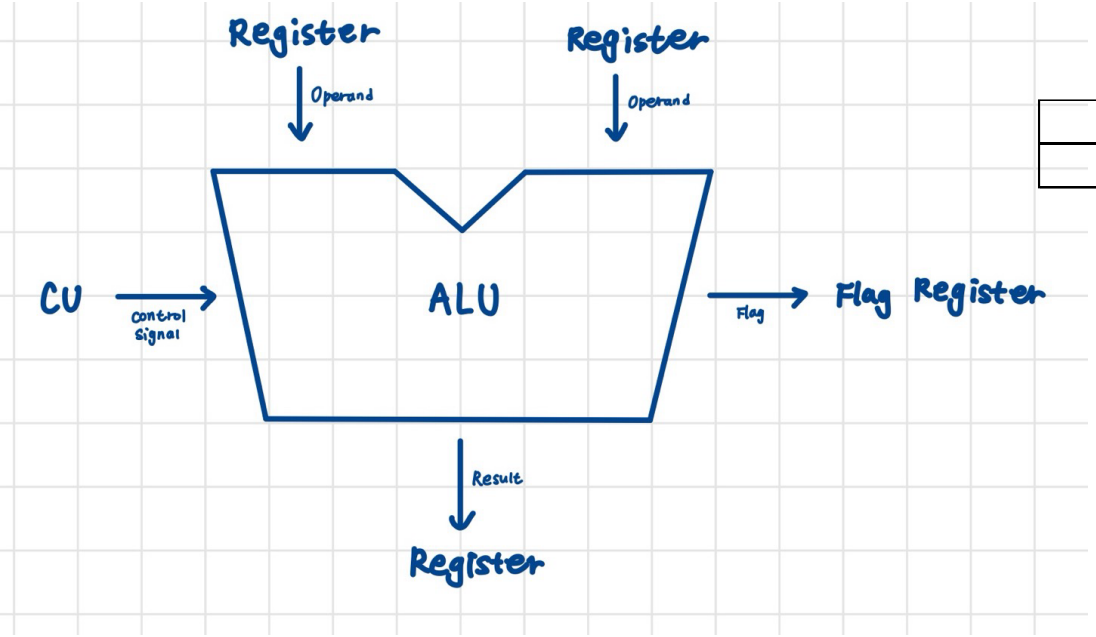
CPU Architecture

Pipelining
: Fetch, Decode, Execute, Write Back

CPU :: 주 기억장치인 메모리에서 명령어를 읽어 들이고 이를 해석하여 수행하는 작업을 함

ALU

- 다양한 산술 및 논리 연산을 처리
- 산술 연산은 덧셈, 뺄셈, 곱셈, 나눗셈 등을 포함하며, 논리 연산은 AND, OR, NOT 등의 비트 수준 연산을 포함

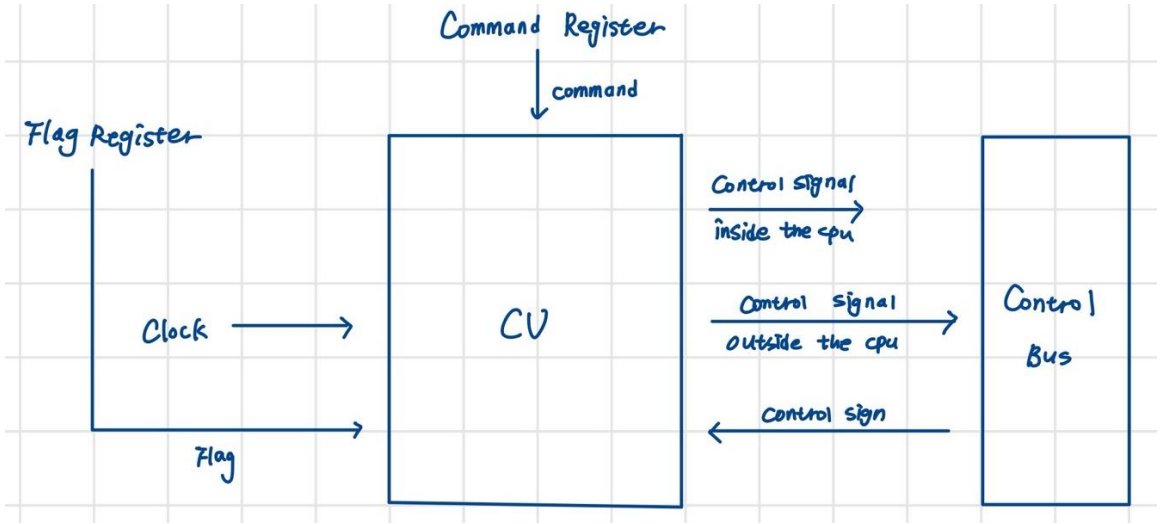


sign	zero	carry	overflow	interrupt	supervisor
1	0	1	0	0	0

ㄴ제가 만듦

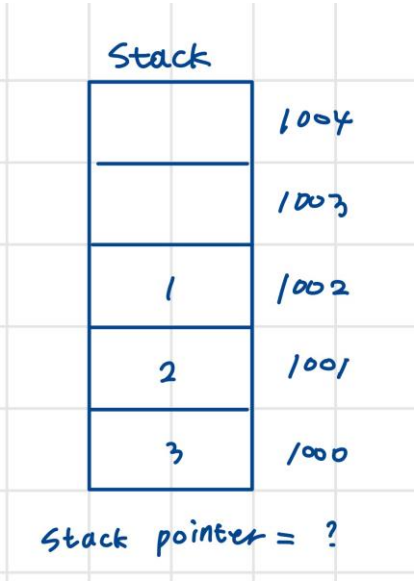
CU

- 다양한 하드웨어 구성 요소들을 조율하고 관리하는 역할
- 시스템 전체의 효율적인 작동을 위해 전기적 제어 신호를 발생시키고, 이를 통해 다른 컴퓨터 부품들의 동작을 제어
- CU가 받아들이는 정보
    1. 클럭 신호를 받아들인다.
    2. 명령어 레지스터에서 명령어와, 플래그 레지스터에서 플래그를 받아들인다.
    3. 마지막으로 시스템 버스, 그 중 제어 버스로 전달된 제어 신호를 받아들인다.
- CU가 내보내는 정보
    1. CPU 외부에 위치한 여러 하드웨어 구성 요소들에게 제어 신호를 보낸다.
    2. CPU 내부에 위치한 ALU나 레지스터에게 내부 제어 신호를 생성한다.



Register

- 프로그램 속 명령어와 데이터는 실행 전후로 반드시 레지스터에 저장
- 프로그램 카운터(Program Counter, PC): 메모리에서 읽어 들일 명령어의 주소를 저장하는 레지스터
  - CPU가 명령어를 실행할 때마다 프로그램 카운터는 자동적으로 증가하여 다음 명령어의 위치를 가리키는 방식으로 작동
- 명령어 레지스터(Command Register, Instruction Register, IR)는: 현재 CPU가 해석하고 있는 명령어를 저장하는 레지스터
  - 명령어가 메모리에서 읽히면, 먼저 이 레지스터에 저장되고 CPU는 이곳에서 명령어를 읽어 해석하고 실행
- 메모리 주소 레지스터(Memory Address Register, MAR): 메모리의 주소를 저장하는 레지스터
- 메모리 버퍼 레지스터(Memory Buffer Register, MBR): 메모리와 주고받을 데이터와 명령어를 저장하는 레지스터
- 범용 레지스터(General Purpose Register): 다양한 목적에 맞게 사용할 수 있는 레지스터
- PSW = 플래그 레지스터(Flag Register): 프로세스의 상태를 나타내는 여러 플래그를 저장하는 레지스터
  - CPU가 복잡한 연산을 수행하거나 프로그램 흐름을 제어하는 데 사용
- 스택 포인터(Stack pointer): 스택의 가장 위를 가리키는 레지스터



MMU

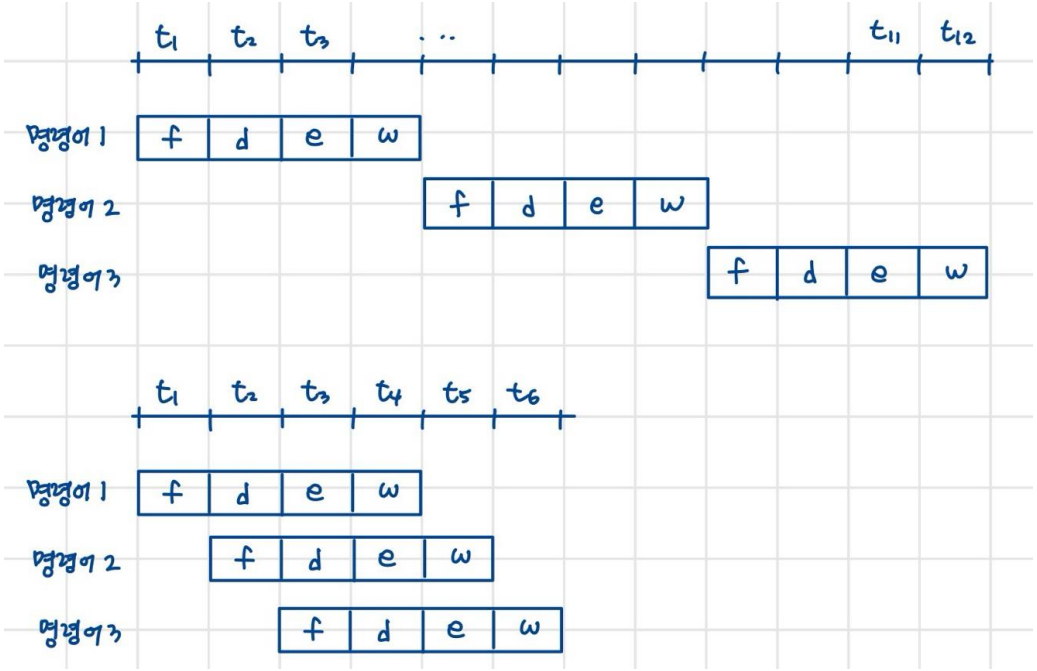
- 논리 주소를 물리 주소로 변환해주며 메모리 보호나 캐시 관리 등 CPU가 메모리에 접근하는 것을 총 관리해주는 하드웨어

- 주소 변환
  - 가상 메모리(Virtual Memory)가 사용되는데, 가상 메모리 시스템에서는 프로세스가 사용하는 메모리 주소(가상 주소)와 실제 메모리의 물리적 주소가 다름
  - 1. 프로세스가 가상 주소를 참조하면, 이 주소가 MMU로 전달
  - 2. TLB(Translation Lookaside Buffer)를 먼저 확인하여 최근 참조된 주소의 변환이 캐시에 있는지 확인하고, 있으면 빠르게 주소를 변환
  - 3. TLB 미스가 발생하면, 페이지 테이블을 참조하여 가상 주소를 물리적 주소로 변환
  - 4. 가상 주소가 페이지 테이블에 존재하지 않거나, 해당 페이지가 메모리에 로드되지 않은 경우 MMU는 페이지 폴트(Page Fault)를 발생
  - 4. 물리적 주소가 확인되면, 해당 메모리 주소로 접근하여 데이터를 처리

- 메모리 보호
  - 한 프로세스가 다른 프로세스의 메모리에 접근하지 못하도록 막음 -> 잘못된 접근이 오면 trap을 발생시키며 보호
  - base 레지스터 : 메모리상의 프로세스 시작주소를 물리 주소로 저장
  - limit 레지스터 : 프로세스의 사이즈를 저장
  - 프로세스의 접근 가능한 합법적인 메모리 영역(x) base <= x < base+limit -> 따라서 이 영역 밖에서 접근을 요구하면 trap을 발생시킴

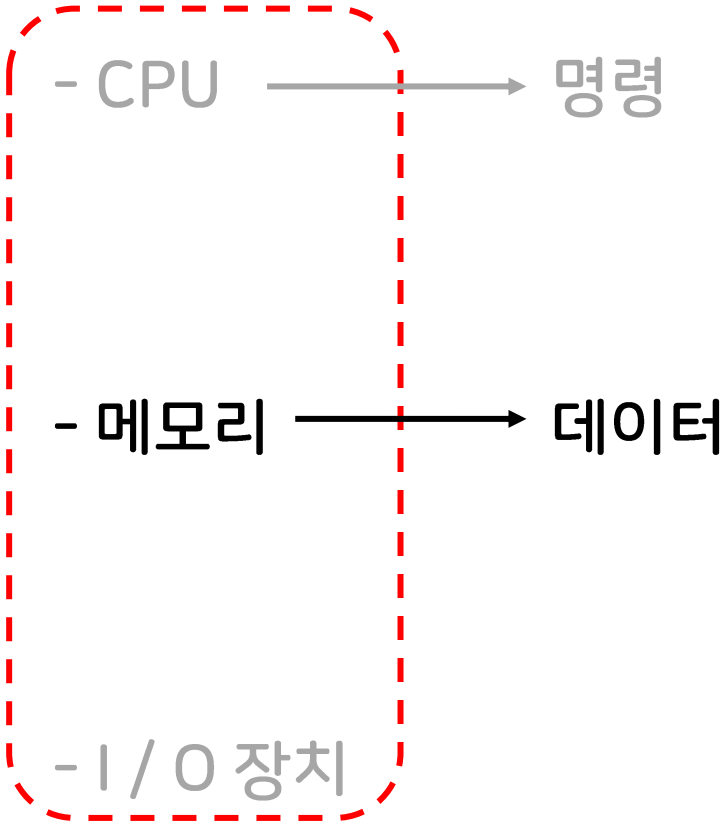
Pipelining

- CPU의 성능을 향상시키기 위해 명령어 처리를 여러 단계로 나누어 동시에 실행하는 명령어 병렬 처리 기법



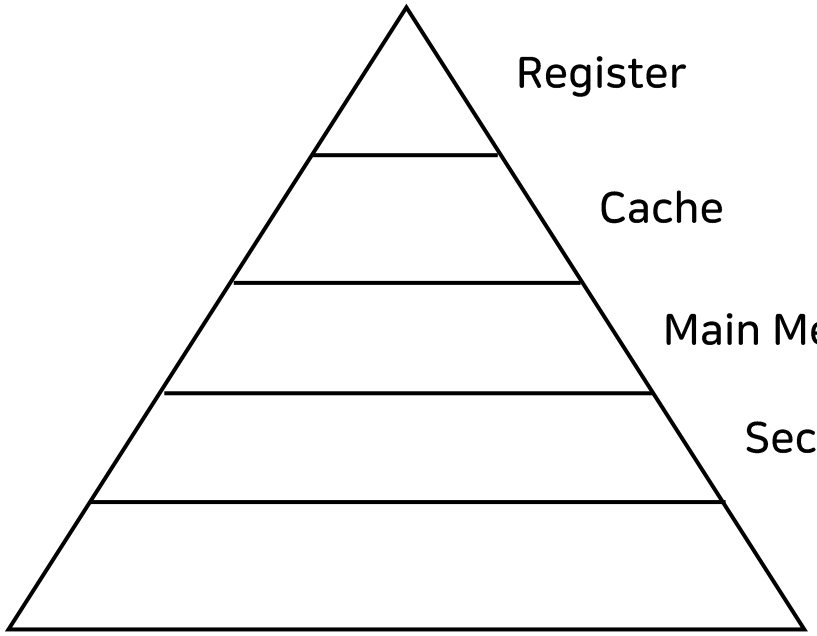
- Fetch : 주기억장치로부터 명령어를 읽어옵니다. 이때 프로그램 카운터의 값이 명령어의 메모리 주소를 가리키며, 이 값을 다음 명령어 주소로 업데이트
- Decode : 읽어온 명령어를 해석하여 명령어와 필요한 데이터를 구분합니다. 이때, 명령어의 종류와 수행할 작업을 파악
- Execute : 해석된 명령어에 따라 필요한 연산을 수행합니다. 예를 들어 산술, 논리, 비트 연산 등 다양한 작업을 실행할 수 있습니다. 이 작업은 ALU에서 수행
- Write back : 실행 결과를 레지스터에 저장합니다. 이 과정에서 연산 결과 레지스터를 업데이트

Computer



Memory

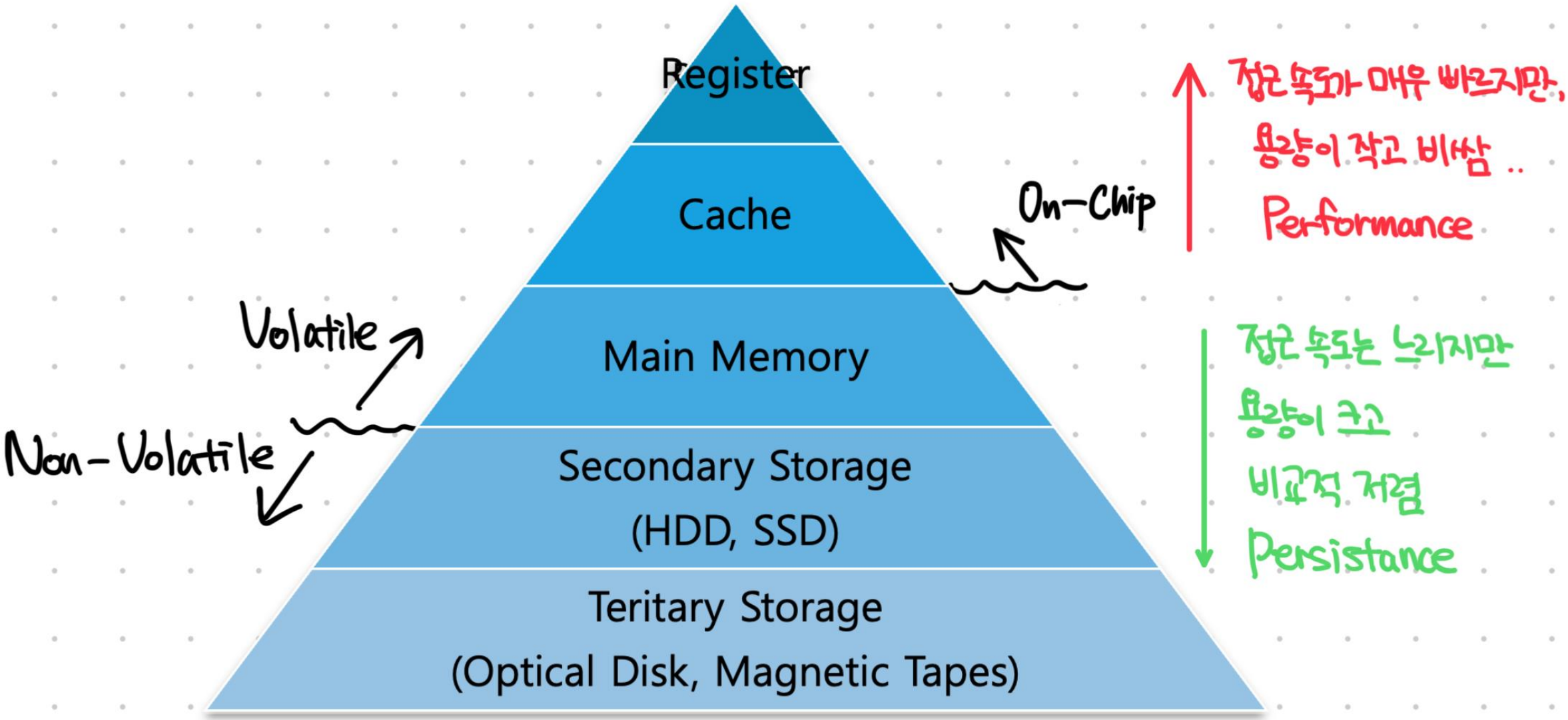
김소명



Memory Architecture

SRAM vs DRAM

ROM vs RAM


Register
Cache
Main Memory
Secondary Storage
(HDD, SSD)
Teritary Storage
(Optical Disk, Magnetic Tapes)
On-Chip
Volatile
Non-Volatile
접근 속도가 매우 빠르지만,
용량이 작고 비쌈 ..
Performance
접근 속도는 느리지만
용량이 크고
비교적 저렴
Persistance

Register

- CPU 내부에 위치 -> 접근, 읽기, 쓰기가 매우 빠름
- 용량이 작음(8bit~64bit) – Architecture에 따라 다름
- CPU가 즉시 처리해야 할 데이터와 명령어 저장, 작업 중간 결과를 저장

- 범용 레지스터
- AX (Accumulator): 산술 및 논리 연산에 주로 사용
- BX (Base): 주소 계산에 사용
- CX (Count): 반복 연산의 카운터로 사용
- DX (Data): 데이터 저장에 사용

- 특수 목적 레지스터
- 프로그램 카운터(PC) : 다음에 실행할 명령어의 주소를 저장
- 명령어 레지스터(IR): 현재 실행 중인 명령어를 저장
- 메모리 주소 레지스터(MAR): 읽기/쓰기 연산을 수행할 주기억장치의 주소를 저장
- 메모리 버퍼 레지스터(MBR): 주기억장치와 데이터를 주고받을 때 임시로 데이터를 저장
- 상태 레지스터(SR): CPU의 현재 상태를 나타내는 플래그들을 저장
- 누산기(AC): 연산 결과를 임시로 저장

Cache

- 자주 사용되는 데이터의 복사본을 저장
- CPU칩 내부에 위치함
- L1 캐시가 가장 빠르고 작으며, L2, L3로 갈수록 용량은 커지지만 속도는 조금 느려짐

- 캐시의 종류
- CPU 캐시: CPU와 주 메모리 사이에 위치한 고속 메모리
- 디스크 캐시: 하드 디스크의 데이터 접근 속도를 개선하기 위한 캐시
- 웹 브라우저 캐시: 웹 페이지, 이미지 등을 로컬에 저장하여 로딩 속도 개선
- 데이터베이스 캐시: 자주 요청되는 쿼리 결과를 저장

- 캐시 작동 원리

1. 캐시 검색: 데이터 요청 시 먼저 캐시를 확인함
2. 캐시 히트(Cache Hit): 요청된 데이터가 캐시에 있는 경우
3. 캐시 미스(Cache Miss): 요청된 데이터가 캐시에 없는 경우
4. 데이터 갱신: 캐시 미스 시 주 저장장치에서 데이터를 가져와 캐시에 저장

Main Memory (주기억장치)

RAM(Random Access Memory)

- 현재 실행중인 프로그램의 데이터를 저장
- 어떤 위치의 데이터도 동일한 속도로 접근 (Random Access)
- 휘발성 : 전원이 꺼지면 데이터가 날아감
- 멀티태스킹 지원 : 여러 프로그램 동시에 실행 가능
- 충분한 RAM 용량은 시스템의 전반적인 성능을 향상시킴

ROM(Read-Only Memory)

- 비휘발성 메모리 : 데이터가 영구적으로 저장됨
- 읽기 전용 : 변경되면 큰일이 날 수 있음
- 주로 부팅 시 필요한 펌웨어 등을 저장함

- SRAM(Cache)
- 빠름
- 전력 소비 높음
- 리프레시 불필요
- 낮은 밀도
- 비쌈

- DRAM(Main Memory)
- 느림
- 전력 소비 낮음
- 주기적인 리프레시 필요
- 높은 밀도 (더 작은 크기)
- 상대적으로 저렴

Secondary Storage(보조기억장치)

- 사용할 거의 모든 데이터를 저장 (운영체제, 응용 프로그램, DB, 문서, 사진, etc..)
- 메인 메모리가 부족할 때 가상 메모리로 사용할 수 있음

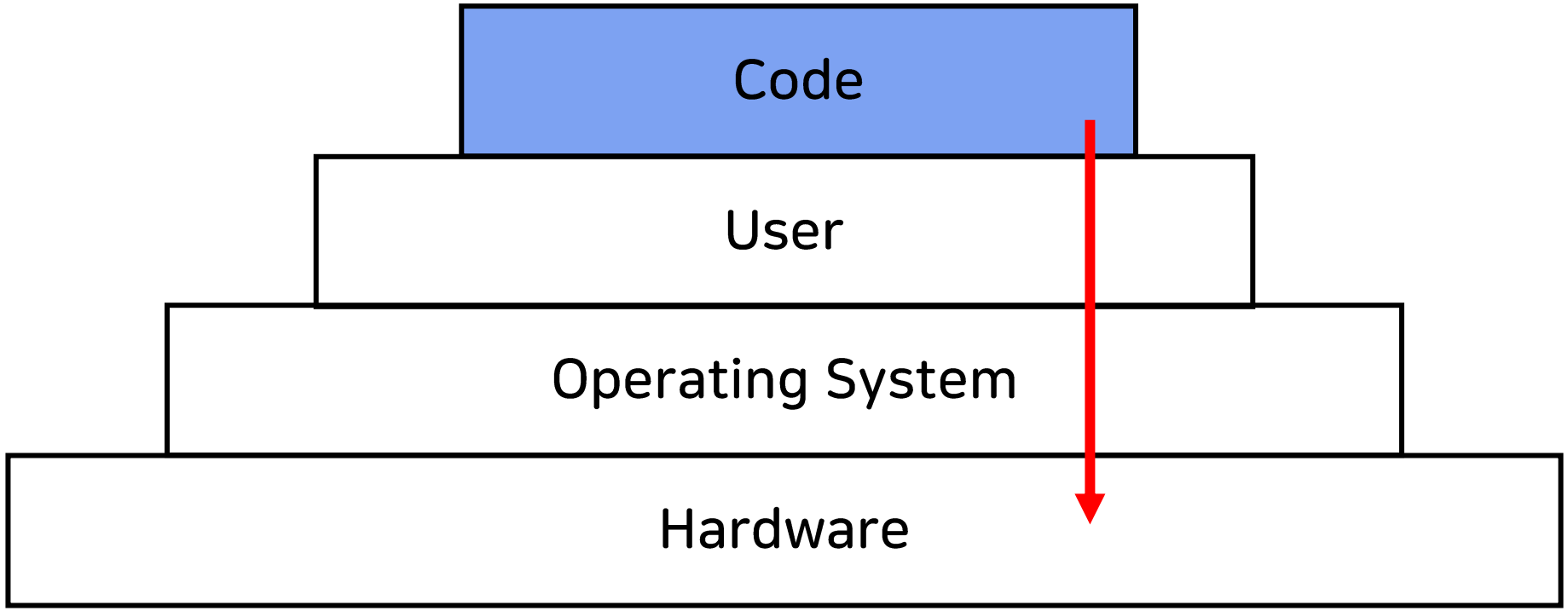
- Secondary Storage의 종류
- HDD, SSD, USB, SD카드, ,,,
- CD나 DVD 등의 저장소도 어떻게 사용하느냐에 따라 Secondary Storage로 분류할 수 있다.

Tertiary Storage (3차 저장소)

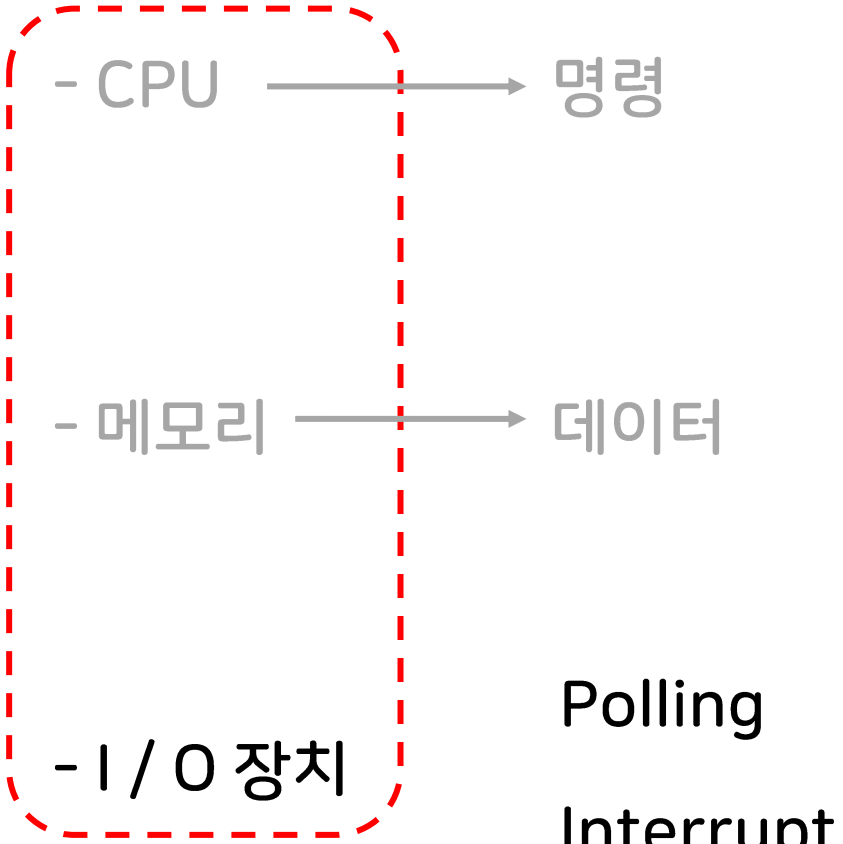
- 백업, 아카이브(기록 보관소), 재해 복구 등, 자료의 초-장기 보관을 목적으로 사용
- 자주 접근하지 않기 때문에 읽기/쓰기 속도가 느림
- 사용할 때만 컴퓨터에 연결
- 데이터 중복성 : 데이터 손실 방지를 위해 같은 정보를 물리적으로 다른 여러 곳에 저장
- 테이프(or CD,DVD) 라이브러리, Archive계층의 클라우드 저장소 ..

Code → 명령

메모리 → 데이터



Computer



Polling

Interrupt

Polling	Synchronous (동기)	scanf (C) std::cin (C++) input (python)
Interrupt	Aynchronous (비동기)	AJAX (javascript)

Task

T/F

T/F

서술형

Curriculum

1주차 : Orientation	(9/4)
2주차 : Data Structure	(9/11)
3주차 : Algorithm 1	(9/25)
4주차 : Algorithm 2	(10/2)
5주차 : Computer Architecture	(10/16)
6주차 : Operating System	(11/6)
7주차 : Computer Network	(11/13)
8주차 : DataBase	(11/20)
9주차 : Design Pattern	(11/27)
10주차 : Security	(12/4)