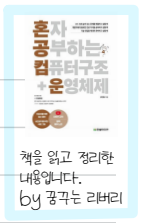
컴퓨터 구조



책을 읽고 정리한 내용입니다.
by 꿍꾸는 리버리

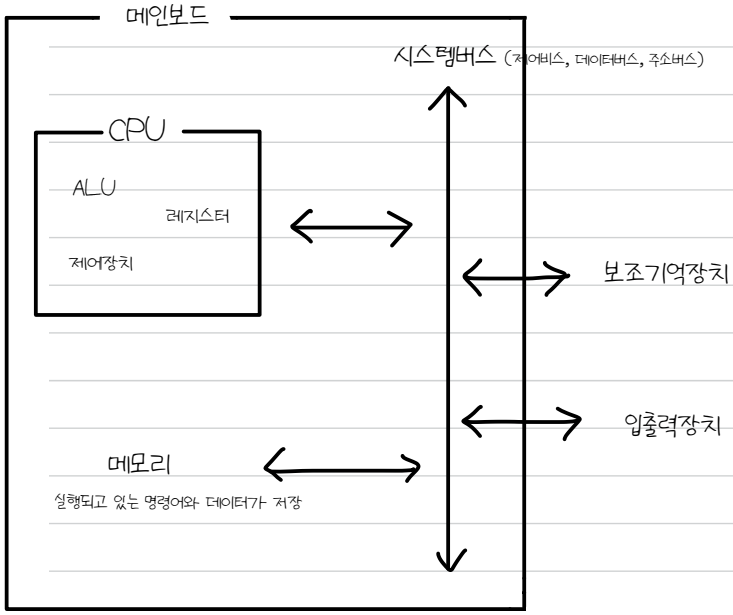
01. 컴구 시작하기

01. 공부의 이유

- 문제해결 능력 향상
- 성능, 용량, 비용의 이해

02. 컴구의 큰 그림

컴퓨터 = 데이터 + 명령어



Cpu가 프로그램을 처리하는 방식

1. 메모리에서 첫 명령어 읽기
CPU의 제어장치의 <메모리 읽기 + 주소>가 시스템 버스의 제어 버스, 주소 버스를 타고 메모리로 이동 -> 원하는 주소의 메모리 값을 읽음 (ex. "더하라. 3번지와 4번지를") 값이 데이터 버스를 타고 레지스터로 이동 -> 제어장치가 메모리의 3번지와 4번지 값을 불러옴 (<메모리 읽기 + 3번지 주소 & 4번지 주소>) -> 메모리가 <메모리 쓰기, 저장할 주소 2개>레지스터에 3번지와 4번지 값이 저장됨 -> ALU에서 불러드린 3,4번지 주소를 합하는 연산이 일어나고 해당 값이 레지스터에 저장됨
2. 메모리에서 첫 명령어를 읽은 이후 2번째 명령어 읽기
첫번째 일을 다 한 제어장치가 두번째 메모리에서 할 일을 읽어옴< 메모리 읽기 + 주소>가 메모리로 이동 -> 원하는 주소의 메모리 값을 읽음 (ex. "저장하라, 연산 결과를") -> 레지스터에 저장되어 있는 연산 결과 값을 제어장치가 읽어서 220을 5번지에 저장함 (<메모리쓰기, 5번지 주소>)

02. 데이터

01. 0과 1로 숫자를 표현하는 방법

bit byte = 8 bit

1 byte → 1kB → 1MB → 1GB → 1TB (1000씩 증가)

이진법

ex) 4(10) = 100(2) = 0b100
: 1/10/11/ 100

음수: 2의 보수 (부호 바꾸고 1 더하기)

ex) - 4= 011 + 1 = 100+ 플래그달기

십육진법: 2^4 = 16 (이진법과 변환이 쉬워서 사용)

ex) DA(16) = 0xDA -> 11011010(2) = 0b11101010
D -> 1101
A -> 1010

02. 0과 1로 문자를 표현하는 방법

문자 집합 : 컴퓨터가 인식하고 표현할 수 있는 문자의 모음

인코딩 : 문자 ------> 10010010010011010100100
디코딩 : 문자 <------ 10010010010011010100100

Cherish 개발할 때 "톱"과 같은 문자가 흰 글씨로 보였었는데, 이때의 issue가 우리가 import한 font가 EUC-KR 형식을 따랐던 거 같음 !!

아스키 코드 : 2^7 문자 집합 (2^8의 나머지는 오류 검출에 사용)

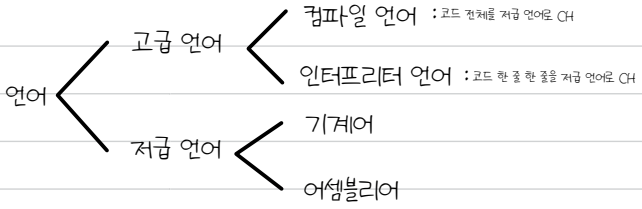
↓ 한글 없음

EUC-KR : 한글을 2바이트 크기로 인코딩하는 완성형 인코딩 방식

유니코드 : 여러 나라 문자를 광범위하게 표현할 수 있는 통일된 문자 집합
UTF-8, UTF-16, UTD-32 (유니코드 문자의 인코딩 방식)

03. 명령어

01. 소스 코드와 명령어



주의 !
- 컴파일 언어가 인터프리터 언어보다 빠름

02. 명령어의 구조

명령어	(연산자) 연산코드	(피연산자) 오퍼랜드(0~3개)

주소 지정방식

1) 즉시 주소 지정 방식
- 오퍼랜드에 연산에 사용할 데이터가 들어있음
- 표현할 수 있는 데이터의 크기가 작아짐
- 다른 주소로 접근하지 않기 때문에 속도는 빠름

2) 직접 주소 지정 방식
- 오퍼랜드에 유효 주소가 적혀 있고, 유효주소를 타고 가면 메모리에 사용할 데이터가 있음
- 오퍼랜드 필드의 길이가 연산 코드의 길이 만큼임

* 유효주소 : 연산의 대상이 되는 데이터가 저장된 위치

3) 간접 주소 지정 방식
- 오퍼랜드에 유효 주소의 주소가 적혀 있고, 유효주소의 주소를 타고 가면 메모리에 유효주소가, 유효주소를 타고 가면 메모리에 사용할 데이터가 있음
- 표현할 수 있는 유효주소의 범위가 넓어진 대신에, 메모리에 2번이나 접근해야 해서 느림

4) 레지스터 주소 지정 방식
- 오퍼랜드에 유효주소가 적혀 있고, 유효 주소를 타고 가면 레지스터에 사용할 데이터가 있음
- 오퍼랜드 필드의 길이가 레지스터의 크기에 제한이 생기지만, 레지스터에 접근하는 거기 때문에 직접 주소 지정 방식보다 빠름

5) 레지스터 간접 주소 지정 방식
- 오퍼랜드에 유효 주소의 주소가 적혀 있고, 유효주소의 주소를 타고 가면 레지스터에 유효주소가, 유효주소를 타고 가면 메모리에 사용할 데이터가 있음
- 레지스터에 접근하고 메모리에 접근하기 때문에 접 주소 지정 방식보다 따름

6) 스택 주소 지정 방식
: 스택 포인터에 따라 데이터를 읽는 방식

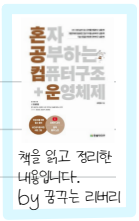
* 스택 영역 : 메모리에 스택처럼 사용하는 일부 영역

7) 변위 주소 지정 방식

- (1) 상대 주소 지정 방식
: 오퍼랜드와 프로그램 카운터의 값을 더하여 유효 주소를 얻는 방식
ex) 오퍼랜드 : -3, 프로그램 카운터: 200 -> 177
: 분기하여 특정 주소의 코드를 실행할 때 사용됨

- (2) 베이스 레지스터 주소 지정 방식
: 오퍼랜드와 베이스 레지스터의 값을 더하여 유효 주소를 얻는 방식
ex) 오퍼랜드: 50, 베이스 레지스터 : 300 -> 350

04. CPU의 동작 원리

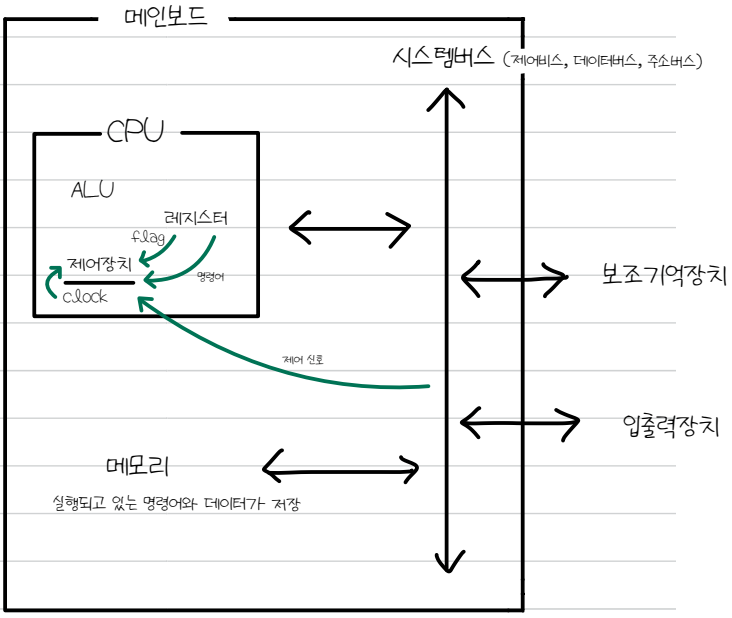


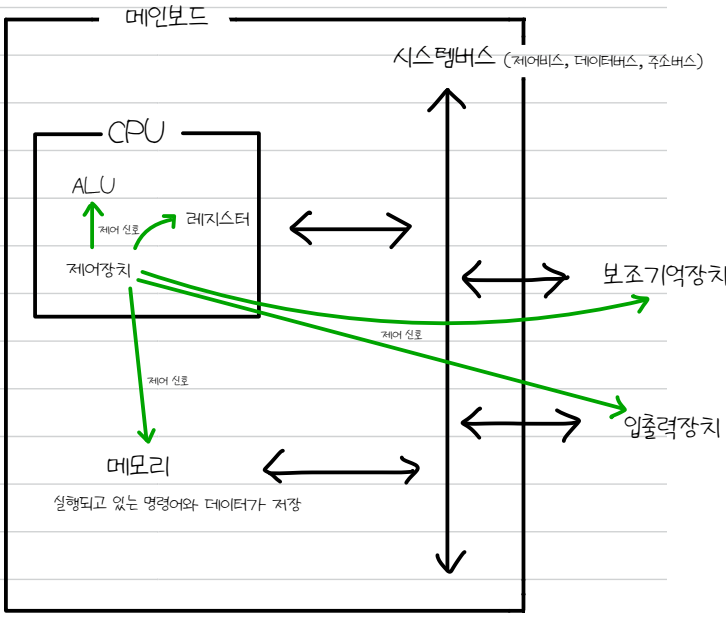
책을 읽고 정리한 내용입니다.
by 꿀꺽는 리버리

01. ALU와 제어장치

- ALU: 계산하는 부품

- 제어장치 : 명령어를 해석하고 제어신호를 내보내는 부품
ex. 메모리를 읽고 쓰는..!





02. 레지스터

레지스터의 종류

1) 프로그램 카운터 (== 명령어 포인터)
: 메모리에서 가져올 명령어의 주소를 저장하는 레지스터

2) 명령어 레지스터
: 방금 메모리에서 읽어들인 명령어를 저장하는 레지스터

3) 메모리 주소 레지스터
: CPU가 읽으려고하는 명령어의 주소 값을 저장하는 레지스터로, 주소 버스로 보낼 때 거치게 됨

4) 메모리 버퍼 레지스터 (== 메모리 데이터 레지스터)
: 메모리와 주고 받을 데이터와 명령어를 저장하는 레지스터로, 데이터 버스로 보낼 때 거치게 됨

5) 플래그 레지스터
: 연산 결과나 CPU의 상태에 대한 부가적인 정보를 저장하는 레지스터

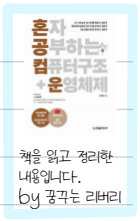
6) 범용 레지스터
: 데이터와 주소를 모두 저장할 수 있는 레지스터
(메모리 주소 레지스터 + 메모리 버퍼 레지스터의 역할을 수행)

7) 스택 포인터
: 스택의 최상단을 가리키는 레지스터

8) 베이스 레지스터
: 기준 주소를 저장하는 레지스터

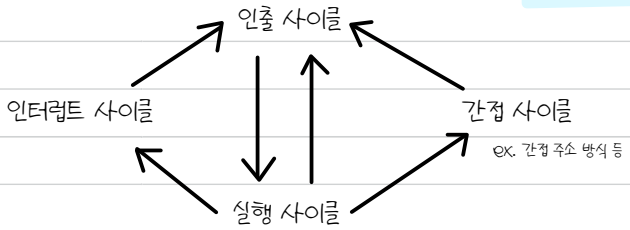
Q. 메모리에 저장된 프로그램을 실행하는 과정
A. 프로그램 카운터의 값이 1000 -> 1000번지라는 주소를 메모리 주소 레지스터에 저장 -> 제어장치는 제어 버스를 타고 <메모리 읽기>를, 메모리 주소 레지스터의 값은 주소 버스를 타고 1000번지 메모리로 이동 -> 1000번지 메모리의 값인 0X123456값을 읽음 -> 0X123456값은 데이터 버스를 타고 메모리 버퍼 레지스터에 저장되면서 프로그램 카운터는 1 증가함 -> 메모리 버퍼에 저장된 값은 명령어 레지스터에 저장됨 -> 제어장치는 명령어 레지스터의 명령어를 해석하고 제어 신호 발생 -> 프로그램 카운터 값을 읽어서 위의 과정을 반복함

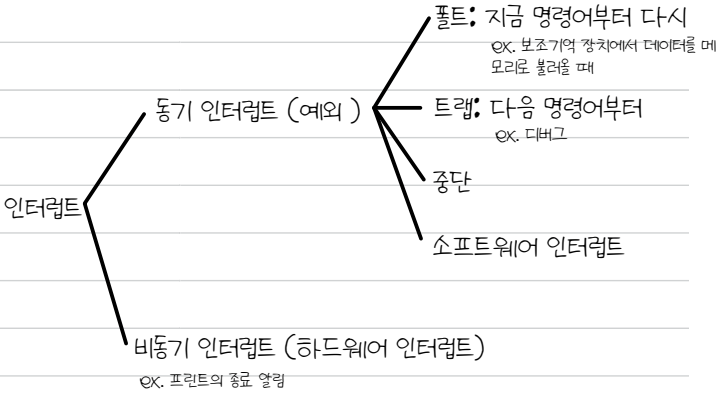
03. 명령어 사이클과 인터럽트



책을 읽고 정리한 내용입니다.
by 꿈꾸는 리버리

명령어 사이클



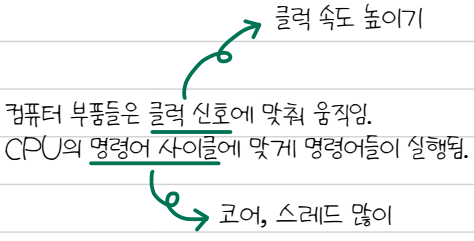


하드웨어 인터럽트의 절차

1. 입출력 장치가 CPU에 인터럽트 요청 신호를 보냄
2. Cpu는 실행 사이클이 돈 후 인터럽트 사이클이 있는 지 확인
3. 만약, 인터럽트 사이클이 있다면, 인터럽트 플래그를 통해서 현재 플래그를 받아들일 수 있는 지 확인
4. 인터럽트 플래그가 1이라면, 프로그램 카운터 등 프로그램 재개시 필요한 내용들을 스택에 백업해둠
5. 인터럽트 벡터를 참조해서 인터럽트 서비스 루틴(인터럽트 핸들러)을 시작
6. 인터럽트 서비스 루틴이 끝난 후 4에서 백업 해둔 작업을 복구해서 실행을 재개

05. CPU 성능 향상 기법

01. 빠른 CPU를 위한 설계 기법

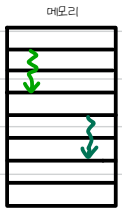


클럭 : 명령어 사이클이 도는 신호 처리 주기 단위
ex. 100Hz = 1초당 100번

- 오버클럭킹 : 최대 클럭 속도를 끌어올림

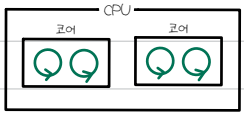
멀티 코어 CPU (멀티 코어 프로세서)
: 여러개의 코어를 가지고 있는 CPU

쓰레드
- 소프트웨어적 쓰레드
  : 하나의 프로그램에서 독립적으로 실행되는 단위
- 하드웨어적 쓰레드 (논리 프로세서)
  : 하나의 코어가 동시에 처리하는 명령어 단위



멀티쓰레드

- 멀티쓰레드 프로세서 (멀티쓰레드 CPU)
  : 레지스터 세트를 여러 개 가지고 있음



02) 명령어 병렬 처리 기법

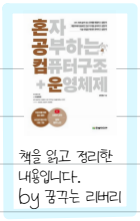
1) 명령어 파이프라인
: 동시에 병렬적을 여러개의 명령어를 처리하는 기법

파이프라인 위험
- 데이터 위험
  ex) 결과값이 도출되지 않은 데이터를 접근하려 함
- 제어 위험
  ex) 분기가 발생해서 실행하고 있는 작업이 쓸모 없게 됨
  -> 분기 예측으로 해결
- 구조적 위험
  ex) 동시에 같은 레지스터에 접근하려 함

2) 슈퍼스칼라
: CPU 내부에 여러 파이프라인을 두는 기법
ex) 멀티 쓰레드

3) 비순차적 명령어 처리
: 파이프라인의 중단을 방지하기 위해서 명령어를 순차적으로 처리하지 않는 기법
-> 파이프라인 위험 중 제어 위험을 해결하기 위한

03) CISC와 RISC



명령어 집합 구조 (ISA)
: 소프트웨어를 해석하는 하드웨어의 언어

Swift -- 다른 ISA인 컴퓨터에서 컴파일 --> 다른 결과

1) CISC
복잡하고 다양한 명령어
가변 길이 명령어
다양한 주소 지정 방식
프로그램을 이루는 명령어의 수가 적음
여러 클럭에 거쳐 명령어 수행
파이프라이닝하기 어려움

2) RISC
단순하고 적은 명령어
고정 길이 명령어
load-store 구조 / 레지스터 적극 활용
프로그램을 이루는 명령어의 수가 많음
1클럭 내외로 명령어 수행
파이프라이닝하기 쉬움

06) 메모리와 캐시 메모리

01) RAM의 특징과 종류

저장 장치
- 휘발성 저장 장치 : 실행할 대상 저장
  ex) RAM(메모리)
- 비휘발성 저장 장치 : 보관할 대상 저장
  ex) SSD, USB 등 보조기억 장치

RAM의 종류

1) DRAM
: 데이터가 동적으로 사라지기 때문에 일정 주기로 데이터를 다시 저장해야 하는 RAM

2) SRAM
: 저장된 데이터가 변하지 않는 RAM

	DRAM	SRAM
재충전	필요	불필요
속도	느림	빠름
가격	저렴함	비쌈
집적도	높음	낮음
소비 전력	낮음	높음
사용 용도	주기억장치	캐시 메모리

* 집적도가 높다 == 더 작고 빽빽하게 만들 수 있다.

3) SDRAM
: 클럭 신호와 동기화된 발전된 형태의 DRAM

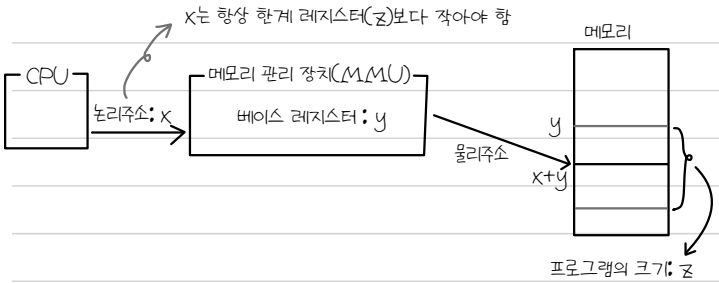
4) DDR SDRAM
: 대역폭을 넓혀 속도를 빠르게 만든 SDRAM

* 대역폭 : 데이터를 주고받는 길의 너비

- DDR SDRAM은 SDR SDRAM보다 2배 넓은 대역폭을 가짐
- DDR3 SDRAM은 DDR SDRAM보다 2^3배 넓은 대역폭을 가짐

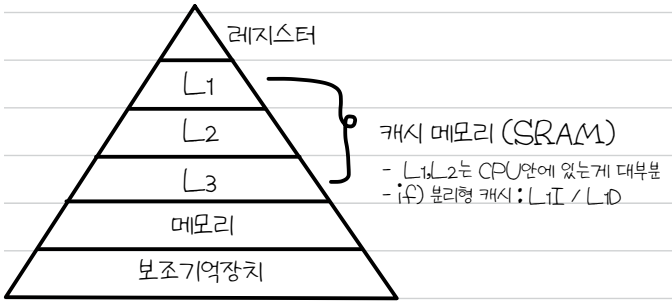
02) 메모리의 주소 공간

주소 < 물리주소: 메모리 하드웨어 상의 주소
논리주소: CPU와 실행중인 프로그램이 사용하는 주소



03) 캐시 메모리

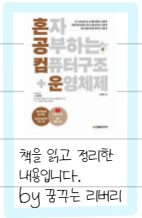
저장장치 계층구조 : 상위 저장 장치일 수록, 코어와 가까이 위치하기 때문에 메모리 접근 속도가 빠르다.



$$\text{캐시 적중률} = \frac{\text{캐시 히트 횟수}}{\text{캐시 히트 횟수} + \text{캐시 미스 횟수}}$$

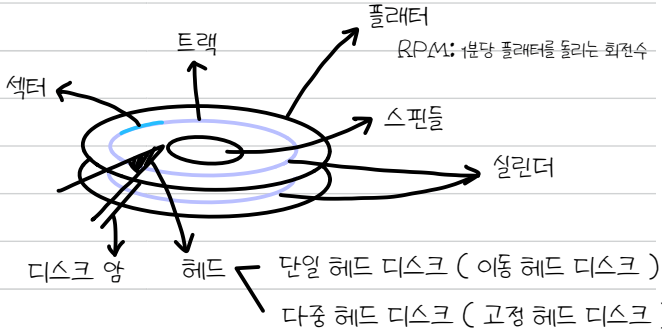
참조 지역성의 원리 < 시간 지역성
공간 지역성

07. 보조기억장치



01. 다양한 보조기억장치

1) 하드 디스크



- 디스크가 저장된 데이터에 접근하는 시간
= 탐색 시간 , 회전 지연, 전송 시간

2) 플래시 메모리

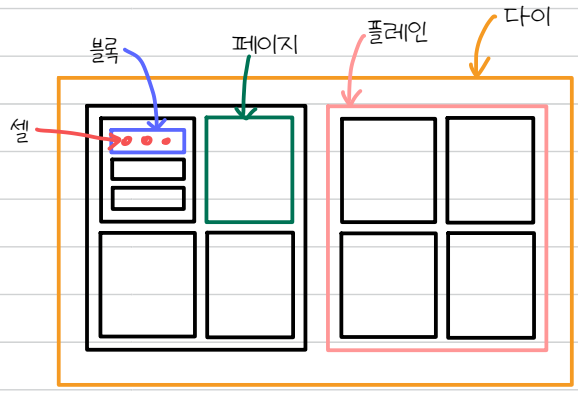
ex. usb, sd카드, ssd

< NAND 플래시 메모리
NOR 플래시 메모리

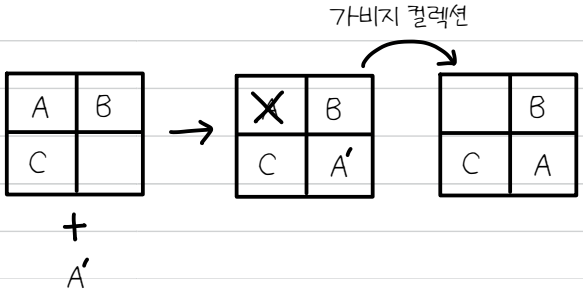
셀 : 플래시 메모리에서 데이터를 저장하는 가장 작은 단위

수명
읽기/쓰기 속도
용량대비 가격

셀당 bit (1) (2) (3)
SLC > MLC > TRC



페이지 상태 < Free 상태
Valid 상태
Invalid 상태

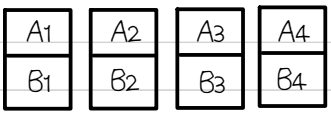


02) RAID의 정의와 종류

RAID
: 데이터의 안전성, 높은 성능을 위해 여러개의 물리적 보조기억장치를 마치 하나의 논리적 보조기억장치처럼 사용하는 기술

RAID 레벨
: RAID 구성 방식

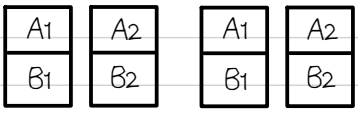
1) RAID 0



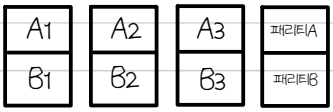
스트라입

스트라이핑

2) RAID 1



3) RAID 2



패리티 비트
: 오류를 검출하고 복구하기 위한 정보

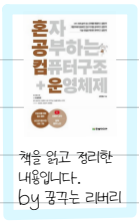
4) RAID 5

5) RAID 6

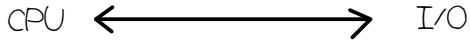
6) Nested RAID

ex. RAID 10, RAID 50

08) 입출력장치



01) 장치 컨트롤러와 장치 드라이버

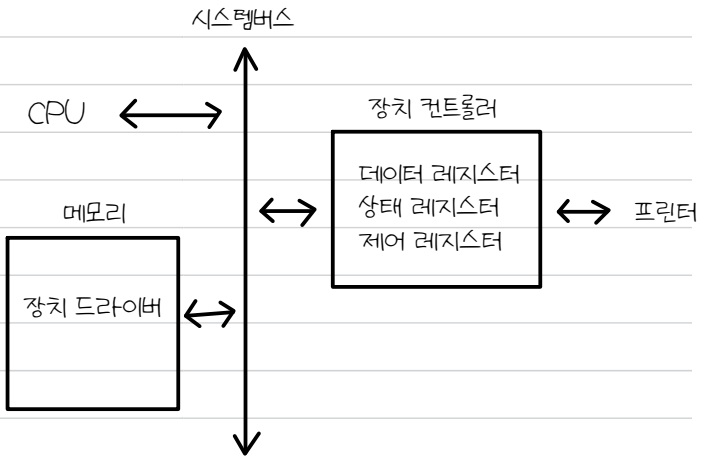


문제점
1) I/O 장치의 종류가 많음
2) 전송률 차이

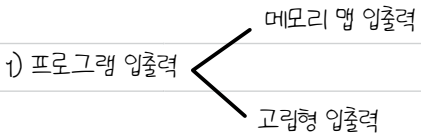
장치 컨트롤러 (입출력 제어기, 입출력 모듈)

기능: CPU와 I/O 사이의 통신 중계, 오류 검출, 데이터 버퍼링

장치 드라이버 : 장치 컨트롤러가 컴퓨터 내부와 정보를 주고 받을 수 있게 하는 프로그램



02) 다양한 입출력 방법



2) 인터럽트 기반 입출력

* 폴링 : 입출력장치의 상태, 처리할 데이터의 유무 등을 주기적으로 확인하는 방식

* PIC (프로그래머블 인터럽트 컨트롤러)

3) DMA 입출력

+)
- 사이클링 스틸링 문제를 입출력 버스로 해결
- 입출력 프로세서 (입출력 채널)