Week 3.

06 메모리와 캐시 메모리

RAM의 특징과 종류

1. RAM의 특징

RAM → 실행할 프로그램의 명령어와 데이터 저장

- **휘발성 저장 장치** : 전원 끄면 저장된 내용 삭제 (RAM)

  실행할 대상 을 저장

- **비휘발성 저장 장치** : 전원 꺼도 저장된 내용 유지 (보조기억장치)

  보관할 대상 을 저장

2. RAM의 용량과 성능

→ CPU가 실행하고 싶은 프로그램이 보조기억장치에 있다면 이를 **RAM으로 복사하여 저장 후 실행**

- RAM 용량에 따라 저장 가능 프로그램 양이 달라져 실행 속도에 영향을 미침
- **RAM이 클수록** 동시 실행 가능 프로그램이 많아져 **성능 향상**
- **but**, RAM 용량이 크다고 실행 속도가 비례하여 빨라지는 것은 X

3. RAM의 종류

[DRAM Dynamic RAM**]**

저장된 데이터가 동적으로 변하는 RAM

- 시간이 지나면 저장된 데이터가 **점차 사라짐**
  → 일정 주기로 데이터 재활성화 필수 (다시 저장)
- 일반적으로 가장 많이 사용되는 RAM

➡️ **소비 전력이 낮고, 저렴하고, 집적도가 높아서 대용량으로 설계하기 용이**

[SRAM Static RAM**]**

저장된 데이터가 변하지 않는 RAM

- 시간이 지나도 저장된 **데이터 유지**
- DRAM보다 속도가 빠름
- 대용량으로 만들어질 필요는 없지만 속도가 빨라야 하는 저장 장치'로 사용 (ex. 캐시 메모리)

➡️ **집적도가 낮고, 소비 전력이 크며, 가격이 비싸서 잘 사용 X**

✔️ SRAM도 전원이 공급되지 않으면 저장 내용 삭제 (휘발성 메모리)

[SDRAM Synchronous Dynamic RAM**]**

클럭 신호와 동기화된, 발전된 형태의 DRAM

→ 클럭에 맞춰 동작하며 **클럭마다 CPU와 정보를 주고받을 수** 있는 DRAM

[DDR SDRAM Double Data Rate SDRAM**]**

대역폭을 넓혀 속도를 빠르게 만든 SDRAM

** 대역폭 : 데이터를 주고받는 길의 너비

→ 한 클럭당 두 번씩 CPU와 데이터를 주고 받을 수 있음 (SDRAM의 2배)

> DDR2 SDRAM은 SDRAM의 4배의 대역폭
> DDR3 SDRAM은 SDRAM의 8배의 대역폭
> DDR4 SDRAM은 SDRAM의 16배의 대역폭

메모리의 주소 공간

1. 물리 주소와 논리 주소

- 물리 주소 : 메모리 하드웨어가 사용하는 주소
- 논리 주소 : CPU와 실행 중인 프로그램이 사용하는 주소

메모리에 저장된 정보가 **시시각각 변하기 때문에** CPU와 실행중인 프로그램이 메모리에 무엇이 저장되어 있는지 다 알 수 없음

[논리 주소와 물리 주소 간의 변환]

CPU와 주소 버스 사이에 위치한 **메모리 관리 장치(MMU)**라는 하드웨어에 의해 수행

MMU → CPU가 발생시킨 **논리 주소에 베이스 레지스터 값을 더하여** 물리 주소로 변환

> **베이스 레지스터** : 프로그램의 가장 작은 물리 주소, 첫 물리 주소를 저장
>
> **논리 주소** : 프로그램 시작점으로부터 떨어진 거리

2. 메모리 보호 기법

[한계 레지스터(limit Register)]

- 논리 주소 범위를 벗어나는 명령어 실행을 방지
- 실행 중인 프로그램이 다른 프로그램에 영향을 받지 않도록 보호하는 것을 담당
- **논리 주소의 최대 크기를 저장**

> **물리 주소 범위**
>
> 베이스 레지스터 값 이상 ~ 베이스 레지스터 값 + 한계 레지스터 값 미만
> ** CPU가 접근하려는 논리 주소는 한계 레지스터가 저장한 값보다 크면 안됨
> → 인터럽트(트랩)을 발생시켜 실행 중단

캐시 메모리

CPU가 메모리에 접근하는 시간과 연산 속도의 차이를 극복하기 위해 만들어진 저장 장치

1. 저장 장치 계층 구조

✔️ 빠른 저장 장치 와 용량이 큰 저장 장치 양립 어려움

> **저장 장치**
>
> - CPU와 가까운 저장 장치는 빠르고, 멀리 있는 저장 장치는 느림
> - 속도가 빠른 저장 장치는 용량이 작고 가격이 비쌈

→컴퓨터가 사용하는 저장 장치들은 **CPU에 얼마나 가까운가**를 기준으로 계층적 표현

→ 각기 다른 용량과 성능의 저장 장치들을 계층화하여 표현한 구조

** 위로 올라갈수록 빠르고 아래로 내려갈수록 느림

2. 캐시 메모리

CPU와 메모리 사이에 위치

레지스터보다 용량이 크고 메모리보다 빠른 SRAM 기반의 저장 장치

** CPU에 가까운 순으로 L1 캐시, L2 캐시, L3 캐시

> 용량 : L3 > L2 > L1
> 속도 가격 : L1 > L2 > L3

3. 참조 지역성 원리

캐시 메모리는 CPU가 사용할 법한 대상을 **예측하여 저장**

- **캐시 히트**

  자주 사용될 것으로 예측한 데이터가 실제로 들어맞아 캐시 메모리 내 데이터가 CPU에서 활용되는 경우

- **캐시 미스**

  자주 사용될 것으로 예측하여 캐시 메모리에 저장했지만, 예측이 틀려 메모리에서 필요한 데이터를 직접 가져와야 하는 경우

- **캐시 적중률**

  캐시가 히트되는 비율

> **캐시 적중률 = 캐시 히트 횟수 / (캐시 히트 횟수 + 캐시 미스 횟수)**
>
> ** 대체적으로 캐시 적중률은 85~95%

[참조 지역성 원리]

CPU가 메모리에 접근할 때의 주된 경향을 바탕으로 만들어진 원리

> **시간 지역성** : 최근에 접근했던 메모리 공간에 다시 접근하려는 경향
>
> **공간 지역성** : 접근한 메모리 공간 근처를 접근하려는 경향

07 보조기억장치

다양한 보조기억장치

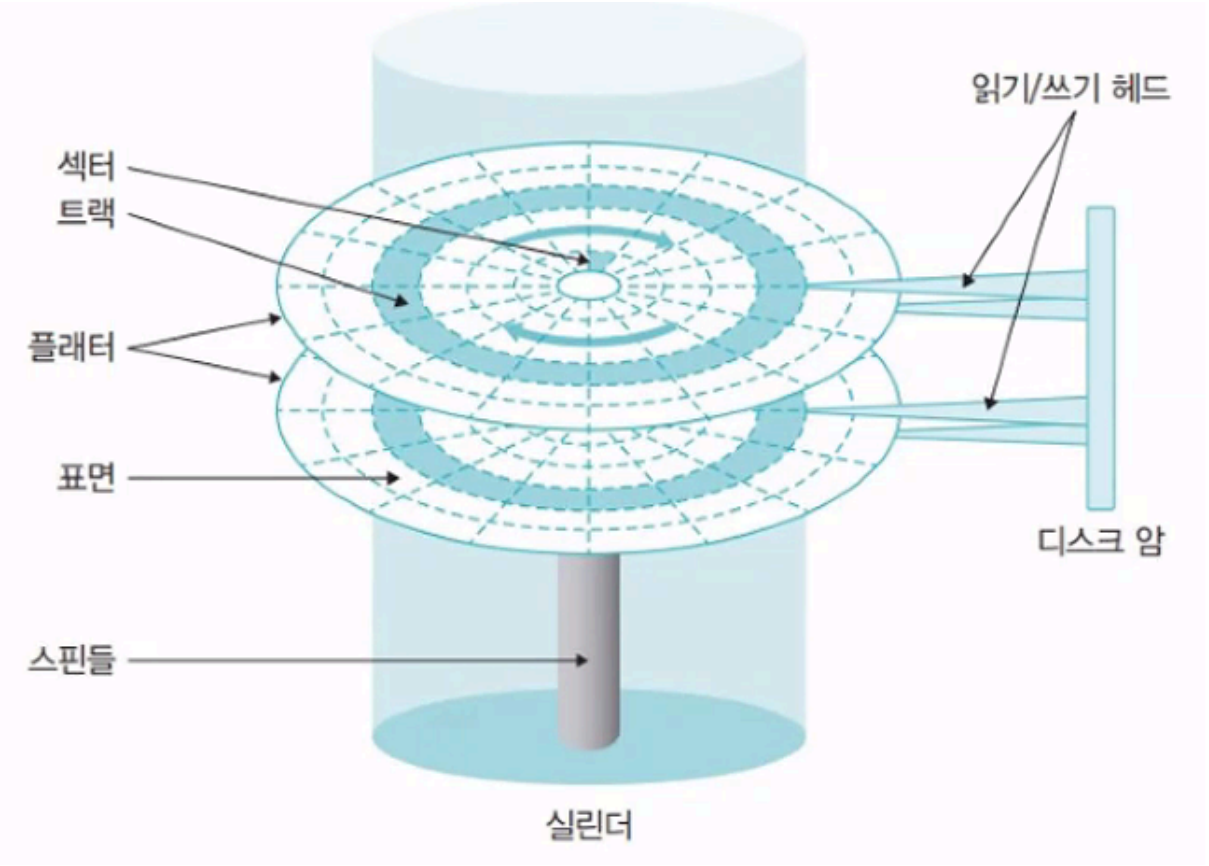
1. 하드 디스크

자기적인 방식으로 데이터를 저장하는 보조기억장치(자기디스크 magnetic disk)

- **플래터 :** 실질적으로 데이터가 저장되는 곳
- **스핀들** : 플래터를 회전시키는 구성 요소
- **RPM(분당 회전수)** : 스핀들이 플래터를 돌리는 속도
- **헤드** : 플래터를 대상으로 데이터를 읽고 쓰는 구성 요소
- **디스크암** : 헤드를 원하는 위치로 이동시키는 요소

💡 많은 양의 데이터를 저장해야 하므로 일반적으로 **여러 겹의 플래터**로 이루어져 있고, 플래터 **양면 모두 사용**

[플래터에 데이터가 저장되는 과정]



- **트랙**

  플래터를 여러 동심원으로 나누었을 때 그 중 하나의 원

- **섹터**

  트랙이 피자처럼 여러 조각으로 나누어진 것 중 한 조각

- **블록**

  하나 이상의 섹터를 묶어 표현

  ** 섹터는 하드 디스크의 **가장 작은 전송 단위**
  **
  하나의 섹터는 일반적으로 512바이트 정도의 크기 (하드 디스크에 따라 차이 O)

- **실린더**

  여러 겹의 플래터 상에서 같은 트랙이 위치한 곳을 모아 연결한 논리적 단위

✔️ 연속된 정보는 한 실린더에 기록 → 디스크 암을 움직이지 않고도 바로 데이터에 접근할 수 있기 때문

[하드 디스크가 저장된 데이터에 접근하는 시간]

- **탐색 시간**

  접근하려는 데이터가 저장된 트랙까지 헤드를 이동시키는 시간

- **회전 지연**

  헤드가 있는 곳으로 플래터를 회전시키는 시간

- **전송 시간**

  하드 디스크와 컴퓨터 간에 데이터를 전송하는 시간

➡️ 하드 디스크에서 다량의 데이터를 탐색하고 읽어 들이는 시간은 생각보다 어마어마

💡 탐색 시간과 회전 지연을 단축시키기 위해서는 **플래터를 빨리 돌려 RPM을 높이는 것도 중요**

but 데이터가 플래터 혹은 헤드를 **조금만 옮겨도 접근할 수 있는 곳에 위치해 있는 것도 중요**

2. 플래시 메모리

전기적으로 데이터를 읽고 쓸 수 있는 반도체 기반의 저장 장치

플래시 메모리는 보조기억장치 범주에만 속한다기보다는 다양한 곳에서 널리 사용하는 저장 장치로 보는 것이 O

ex. USB 메모리, SD 카드, SSD 등..

> **NAND 플래시 메모리** : NAND 연산을 수행하는 회로
> → 대용량 저장 장치로 많이 사용, 대부분의 플래시 메모리
>
> **NOR 플래시 메모리** : NOR 연산을 수행하는 회로

셀 cell

플래시 메모리에서 데이터를 저장하는 가장 작은 단위
셀이 모여서 MB, GB, TB 용량을 갖는 저장 장치가 되는 것

SLC 타입

- 한 셀로 두 개의 정보를 표현
- 비트의 빠른 입출력
- 수명도 MLC나 TLC보다 길어서 수만에서 수십만 번 가까이 데이터를 쓰고 지우고를 반복
- 용량 대비 가격이 높음

➡️ 보통 기업에서 데이터를 읽고 쓰기가 매우 많이 반복되며 고성능의 빠른 저장 장치가 필요한 경우에 사용

MLC 타입

- 한 셀로 네 개의 정보 표현
- SLC보다 일반적으로 속도와 수명이 떨어지지만, 한 셀에 두 비트씩 저장할 수 있다는 점에서 **대용량화가 유리**
- 용량 대비 가격이 저렴

TLC 타입

- 한 셀당 3비트씩 저장
- 한 셀로 여덟 개의 정보를 표현
- 대용화 하기 유리
- SLC나 MLC보다 수명과 속도가 떨어지지만 용량 대비 가격이 저렴

> 썼다 지우기를 자주 반복 or 높은 성능 → **고가의 SLC**
>
> 저가의 대용량 저장 장치 → **TLC**
>
> 그 중간 → **MLC**

페이지 page

셀들이 모여 만들어진 단위, 읽기와 쓰기가 이루어지는 단위

Free, Valid, Invalid 상태

> **Free** : 어떠한 데이터도 저장하고 있지 않아 새로운 데이터를 저장할 수 있는 상태
>
> **Valid** : 이미 유효한 데이터를 저장하고 있는 상태
>
> **Invalid** : 쓰레기값이라 부르는 유효하지 않은 데이터를 저장하고 있는 상태
>
> ** 플래시 메모리는 하드 디스크와 달리 덮어쓰기가 불가능
> → Valid 상태인 페이지에는 새 데이터 저장 불가능

블록 block

페이지가 모여 만들어진 단위

플레인 plane

블록이 모여 만들어진 단위

다이 die

플레인이 모여 만들어진 단위

** **가비지 컬렉션** : 유효한 페이지들만을 새로운 블록으로 복사 → 기존의 블록을 삭제하는 기능

RAID의 정의와 종류

1. RAID의 정의

주로 하드 디스크와 SSD를 사용하는 기술

데이터의 안전성 혹은 높은 성능을 위해 **여러 개의 물리적 보조기억장치를 마치 하나의 논리적 보조기억장치처럼 사용하는 기술**

2. RAID의 종류

[RAID 레벨]

RAID 구성 방법 → **RAID 0, RAID 1, RAID 2, RAID 3, RAID 4, RAID 5, RAID 6**

** RAID 2, RAID 3은 현재 활용 X

- **RAID 0**

  여러 개의 보조기억장치에 데이터를 단순히 나누어 저장하는 구성 방식
  저장되는 데이터가 하드 디스크 개수만큼 나뉘어 저장

> **스트라입** : 줄무늬처럼 분산되어 저장된 데이터
>
> **스트라이핑** : 분산하여 저장하는 것
>
> **[장점]**
>
> 하나의 대용량 저장 장치를 이용했다면 여러 번에 걸쳐 읽고 썼을 데이터를 **동시에 읽고 쓸 수 있기** 때문에 저장된 데이터를 읽고 쓰는 속도 빨라짐
>
> **[단점]**
>
> 저장된 정보가 안전 X
>
> 하나의 하드 디스크에 문제가 생기면 다른 모든 하드 디스크의 정보를 읽는데 문제 발생 가능성 O

- **RAID 1**
  복사본
  을 만드는 방식

> **미러링** : 거울처럼 완전한 복사본을 만드는 구성
>
> 원본과 복사본을 만드는 구조이기 때문에 쓰기 속도가 **RAID 0보다 느림**
> **[장점]**

> 복구가 매우 간단
>
> **[단점]**
>
> 하드 디스크 개수가 한정되었을 때 사용 가능한 용량이 적어짐

- **RAID 4**

  완전한 복사본을 만드는 대신 오류를 검출하고 복구하기 위한 정보를 저장한 장치를 두는 구성 방식

> **패리티 비트** : 오류를 검출하고 복구하기 위한 정보
>
> 패리티를 저장한 장치를 이용해 다른 장치들의 오류를 검출하고, 오류가 있다면 복구
>
> RAID 1보다 적은 하드 디스크로도 데이터를 **안전하게 보관**

✔️ 어떤 새로운 데이터가 저장될 때마다 패리티를 저장하는 디스크에도 데이터를 쓰게되므로 패리티를 저장하는 장치에 병목 현상 이 발생한다는 문제 생김

- **RAID 5**

  패리티 정보를 분산하여 저장하는 방식으로 RAID 4의 병목 현상 해소

- **RAID 6**

  기본적으로 RAID 5와 같으나, **서로 다른 두 개의 패리티**를 두는 방식
  오류를 검출하고 복구할 수 있는 수단이 2개
  → RAID 4, 5보다 안전한 구성
  새로운 정보를 저장할 때마다 함께 저장할 패리티가 두 개이므로 쓰기 속도는 RAID 5보다 느림

  → 저장 속도를 희생하더라도 **데이터를 더욱 안전하게 보관하고 싶을 때 사용**하는 방식

08 입출력 장치

장치 컨트롤러와 장치 드라이버

1. 장치 컨트롤러

[입출력장치가 다루기 까다로운 이유]

- 입출력장치의 종류가 매우 많아 **규격화 하기 어렵기 때문**
- 일반적으로 CPU와 메모리의 데이터 전송률은 높지만, 입출력 장치의 데이터 전송률은 낮음

  ** 전송률 : 데이터를 얼마나 빨리 교환할 수 있는지를 나타내는 지표

→ 입출력장치는 컴퓨터에 직접 연결되지 않고 **장치 컨트롤러라는 하드웨어를 통해 연결**

[장치 컨트롤러의 역할]

- CPU와 입출력장치 간의 통신 중개
- 오류 검출
- 데이터 버퍼링
  → 종류의 다양성으로 인해 정보 규격화가 어려웠던 문제는 장치 컨트롤러가 일종의 **번역가 역할**을 함으로써 해결

> **데이터 버퍼링**
>
> **버퍼링** : 전송률이 높은 장치와 낮은 장치 사이에 주고받는 데이터를 `버퍼`라는 임시 저장 공간에 저장하여 전송률을 비슷하게 맞추는 방법
>
> 버퍼에 데이터를 조금씩 모았다가 한꺼번에 내보내거나, 데이터를 한 번에 많이 받아 조금씩 내보내는 방법
>
> 장치 컨트롤러는 일반적으로 전송률이 높은 CPU와 일반적으로 전송률이 낮은 입출력장치와의 **전송률 차이를 데이터 버퍼링으로 완화**

[장치 컨트롤러의 내부 구조]

- **데이터 레지스터**

  CPU와 입출력장치 사이에 주고받을 데이터가 담기는 레지스터
  데이터 레지스터가
  버퍼링 과정에서의 버퍼 역할

  ** 최근 주고받는 데이터가 많은 입출력장치에서는 레지스터 대신 RAM을 사용

- **상태 레지스터**

  입출력장치가 입출력 작업을 할 준비가 되었는지, 입출력 작업이 완료되었는지, 입출력 장치에 오류는 없는지 등의 **상태 정보 저장**

- **제어 레지스터**

  입출력장치가 수행할 내용에 대한 제어 정보와 명령을 저장

2. 장치 드라이버

장치 컨트롤러의 동작을 감지하고 제어함으로써 장치 컨트롤러가 컴퓨터 내부와 정보를 주고받을 수 있게 하는 프로그램

- 장치 컨트롤러 : 입출력 장치를 연결하기 위한 **하드웨어적인** 통로
- 장치 드라이버 : 입출력장치를 연결하기 위한 **소프트웨어적인** 통로

다양한 입출력 방법

1. 프로그램 입출력

프로그램 속 명령어로 입출력장치를 제어하는 방법

→ CPU가 장치 컨트롤러의 레지스터 값을 읽고 씀으로써 이루어짐

[CPU가 장치 컨트롤러의 레지스터를 아는 방법(입출력장치의 주소)]

- **메모리 맵 입출력**

  메모리에 접근하기 위한 주소 공간과 입출력장치에 접근하기 위한 주소 공간을 하나의 주소 공간으로 간주하는 방법

CPU는 메모리의 주소들이나 장치 컨트롤러의 레지스터들이나 모두 똑같이 메모리 주소를 대하듯 하면 됨!

→ 메모리에 접근하는 명령어와 입출력장치에 접근하는 **명령어가 다를 필요 X**

- **고립형 입출력**

  메모리를 위한 주소 공간과 입출력장치를 위한 주소 공간을 분리하는 방법

  CPU는 입출력장치에 접근하기 위해 메모리에 접근하는 명령어와는 **다른 입출력 명령어 사용**

2. 인터럽트 기반 입출력

CPU가 입출력장치에 처리할 내용을 명령하면 입출력장치가 명령어를 수행하는 동안 CPU는 다른 일 수행

하드웨어 인터럽트는 입출력장치가 아닌 **장치 컨트롤러에 의해 발생**

> **[인터럽트 실행 순서]**
>
> 1. CPU는 장치 컨트롤러에 입출력 작업 명령
> 2. 장치 컨트롤러가 입출력 장치를 제어 및 입출력 수행
> 3. CPU 다른 일 수행 가능
> 4. 장치 컨트롤러가 입출력 작업 끝낸 뒤 CPU에게 인터럽트 요청 신호 전송
> 5. CPU는 하던 일 백업 후 인터럽트 서비스 루틴 실행

→ CPU는 인터럽트 간에 우선순위를 고려하여 **우선순위가 높은 인터럽트 순**으로 여러 인터럽트를 처리

[프로그래머블 인터럽트 컨트롤러(PIC)]

우선순위를 반영하여 다중 인터럽트를 처리하는 방법 중 많이 사용하는 방법

여러 장치 컨트롤러에 연결되어 장치 컨트롤러에서 보낸 하드웨어 인터럽트 요청들의 우선순위를 판별,

CPU에 지금 처리해야 할 하드웨어 인터럽트가 무엇인지 알려주는 장치

PIC의 다중 인터럽트 처리 과정

- PIC가 장치 컨트롤러에서 **인터럽트 요청 신호 수신**
- PIC는 인터럽트 **우선순위 판단** 후, CPU에 처리해야 할 **인터럽트 요청 신호 전송**
- CPU는 PIC에 **인터럽트 확인 신호 전송**
- PIC는 데이터 버스를 통해 CPU에 **인터럽트 벡터 전송**
- CPU는 인터럽트 벡터를 통해 인터럽트 **요청의 주체**를 알게 되고, 해당 장치의 **인터럽트 서비스 루틴 실행**일반적으로 더 많고 복잡한 장치들의 인터럽트를 관리하기 위해 PIC를 **두 개 이상 계층적으로 구성**

DMA 입출력

입출력장치와 메모리 사이에 전송되는 모든 데이터는 CPU를 반드시 거친다는 공통점

→ CPU는 입출력장치를 위한 연산 등 때문에 시간을 뺏기게 됨

[DMA(Direct Memory Access)]

입출력장치와 메모리가 CPU를 거치지 않고도 상호작용할 수 있는 입출력 방식

DMA를 입출력하기 위해서는 시스템 버스에 연결된 `DMA 컨트롤러` 라는 하드웨어 필요

> ### DMA 입출력 과정
>
> - CPU는 DMA 컨트롤러에 입출력장치의 주소, 수행할 연산, 읽거나 쓸 메모리의 주소 등과 같은 정보로 **입출력 작업 명령**
> - DMA 컨트롤러는 CPU 대신 **장치 컨트롤러와 상호작용하며 입출력 작업을 수행**(필요한 경우 메모리에 직접 접근)
> - 입출력 작업이 끝나면 DMA 컨트롤러는 CPU에 인터럽트를 걸어 작업 종료 알림

> **메모리 내의 정보를 하드디스크에 백업하는 작업 (DMA 입출력)**
>
> - CPU는 DMA 컨트롤러에 하드 디스크 주소, 수행할 연산, 백업할 내용이 저장된 메모리의 주소 등의 정보와 함께 **입출력 작업 명령**
> - DMA 컨트롤러는 CPU를 거치지 않고 **메모리와 직접 상호작용**하며 백업할 정보를 읽어옴
> - 하드 디스크의 장치 컨트롤러에 전송
> - 백업이 끝나면 DMA 컨트롤러는 CPU에게 인터럽트 걸어 작업 종료 알림

→ DMA 컨트롤러는 시스템 버스로 메모리에 직접 접근 가능 but 시스템 버스는 **동시 사용 불가능**

✔️ CPU가 시스템 버스를 사용할 때 DMA 컨트롤러는 사용 불가

** DMA 컨트롤러가 시스템 버스를 사용하는 것을 `사이클 스틸링`이라고 함

[입출력 버스]

CPU, 메모리, DMA 컨트롤러, 장치 컨트롤러가 모두 같은 버스를 공유하는 구성에서는 DMA를 위해 한 번 메모리에 접근할 때마다 **시스템 버스를 두 번 사용하게 되는 부작용 발생**

→ DMA 컨트롤러와 장치 컨트롤러들을 `입출력 버스`라는 별도의 버스에 연결하여 해결

> **입출력 버스**
>
> - PCI(Peripheral Component Interconnect) 버스
> - PCI Expres (PCIe) 버스
> - PCIe 슬롯 : 여러 입출력장치들을 PCIe 버스와 연결해 주는 통로