Operating Systems

# 2. Computer System

Hyunchan, Park

http://oslab.chonbuk.ac.kr

Division of Computer Science and Engineering

Chonbuk National University

# 학습 목표

- 운영체제의 동작 환경인 컴퓨터 시스템을 리뷰

- 운영체제와 연관된 컴퓨터 시스템의 이슈를 학습

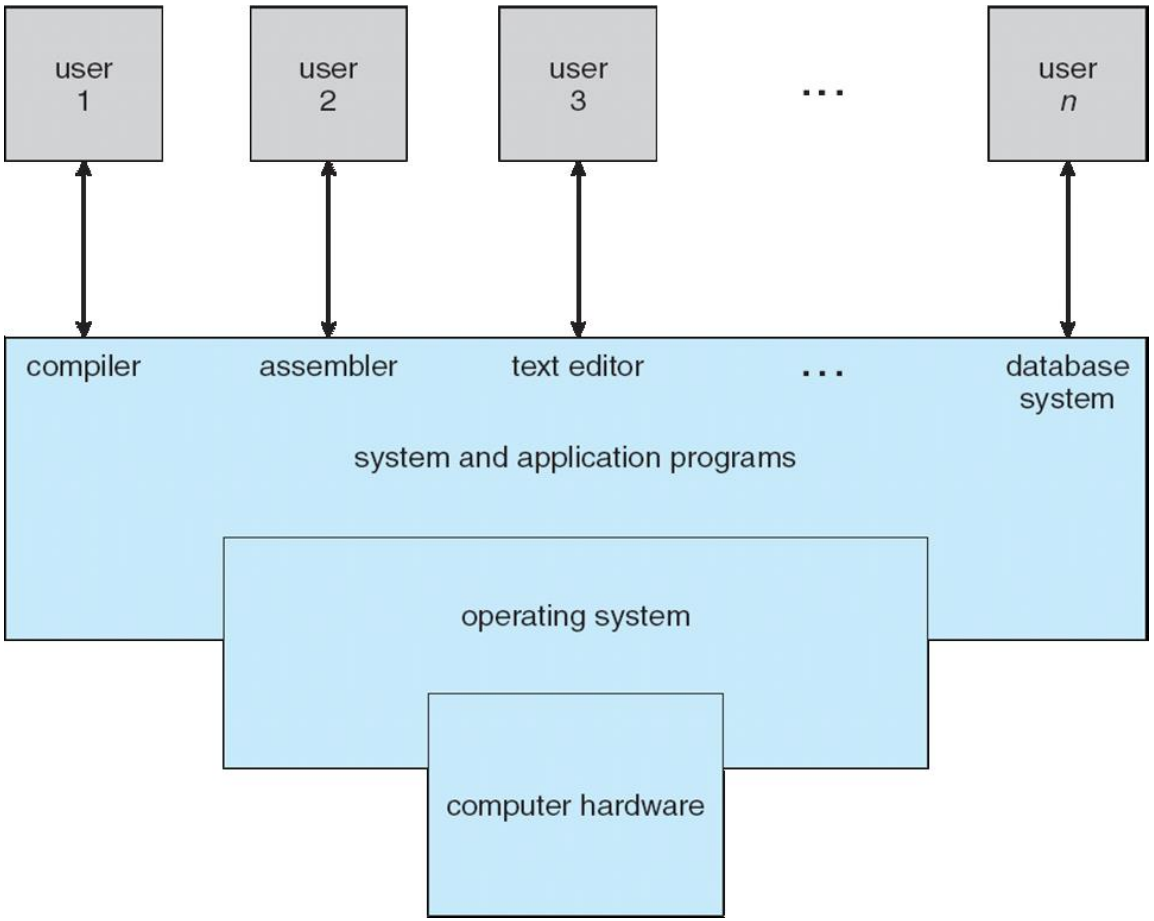
# Contents

- 컴퓨터 시스템의 구조

- OS와 연관된 컴퓨터 시스템 개념
  - User mode and Kernel mode
  - Event handling Mechanisms: Interrupt and Trap
  - I/O Device basic concept
  - I/O Device access 기법: I/O Instruction and Memory Mapped I/O
  - I/O 처리 기법: Interrupt, polling and DMA

- System boot!!

# 컴퓨터 시스템의 네 가지 요소



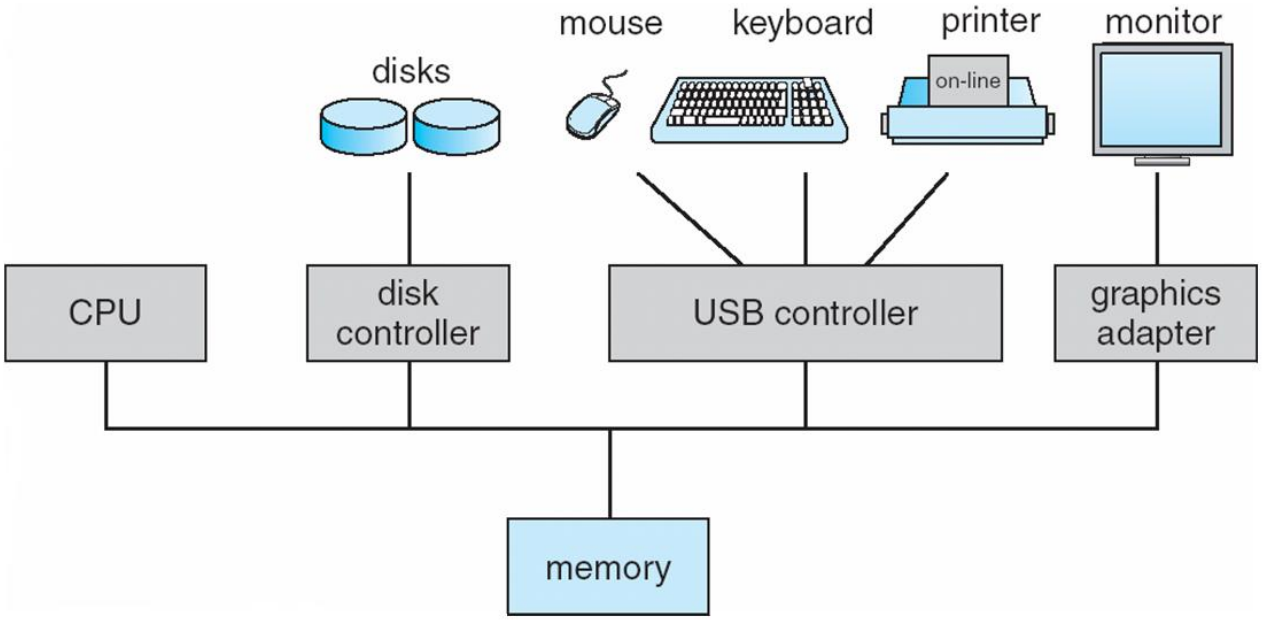
# 컴퓨터 시스템의 네 가지 요소

- Hardware – provides basic computing resources
  - CPU, memory, I/O devices
- Operating system
  - Controls and coordinates use of hardware among various applications and users
- Application programs – define the ways in which the system resources are used to solve the computing problems of the users
  - Word processors, compilers, web browsers, database systems, video games
- Users
  - People, machines, other computers

# 컴퓨터 시스템의 구조

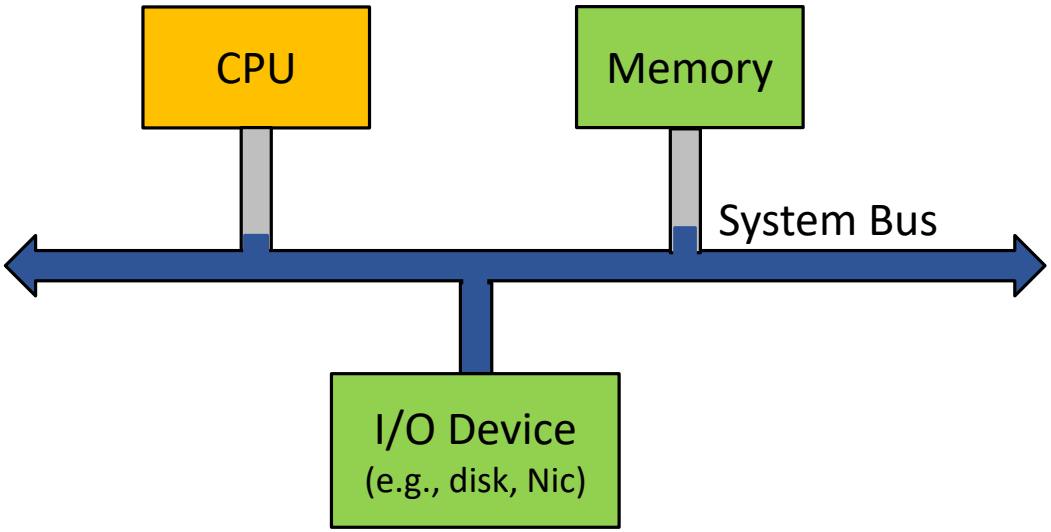
# 컴퓨터 시스템의 일반적인 구조

- 초창기: 단일 버스 사용



# 컴퓨터 시스템의 초창기 구조

- 단일 버스
  - 한 종류의 시스템 버스에 여러가지 모듈이 연결
  - CPU, Memory, I/O의 속도가 비슷했던 초창기에 사용
  - CPU, Memory, I/O의 속도 격차 증가 → 병목 현상 발생



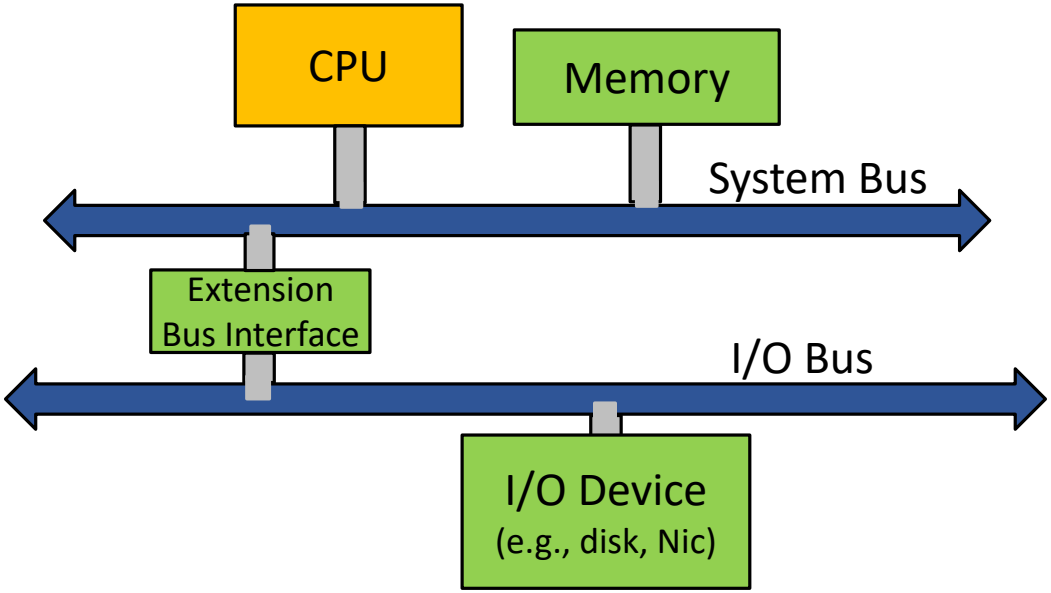

<단일 버스 구조>

# Bottleneck

- 병목현상
  - 같은 버스에 연결된 디바이스들 간의 속도 차이로 인해 발생
  - 빠른 디바이스가 처리하는 양 만큼을 느린 디바이스가 처리하지 못함 =>
    전체 시스템 속도는 느린 디바이스의 속도로 제한됨
  - 예) CPU는 초당 5 단위의 일을 처리할 수 있는데, 메모리가 초당 3 단위의 일만을 전달해 줄 수 있다고 할 때, 전체적인 시스템 속도는 메모리의 속도로 제한된다

# 계층적 이중 버스 구조

- 계층적 이중 버스
  - 속도 격차로 인한 병목 현상 해결
  - 접근 빈도가 적고, 처리 속도가 느린 장치들은 시스템 버스에 직접 연결하지 않음(I/O 버스를 거쳐 연결됨) → 병목 현상 방지

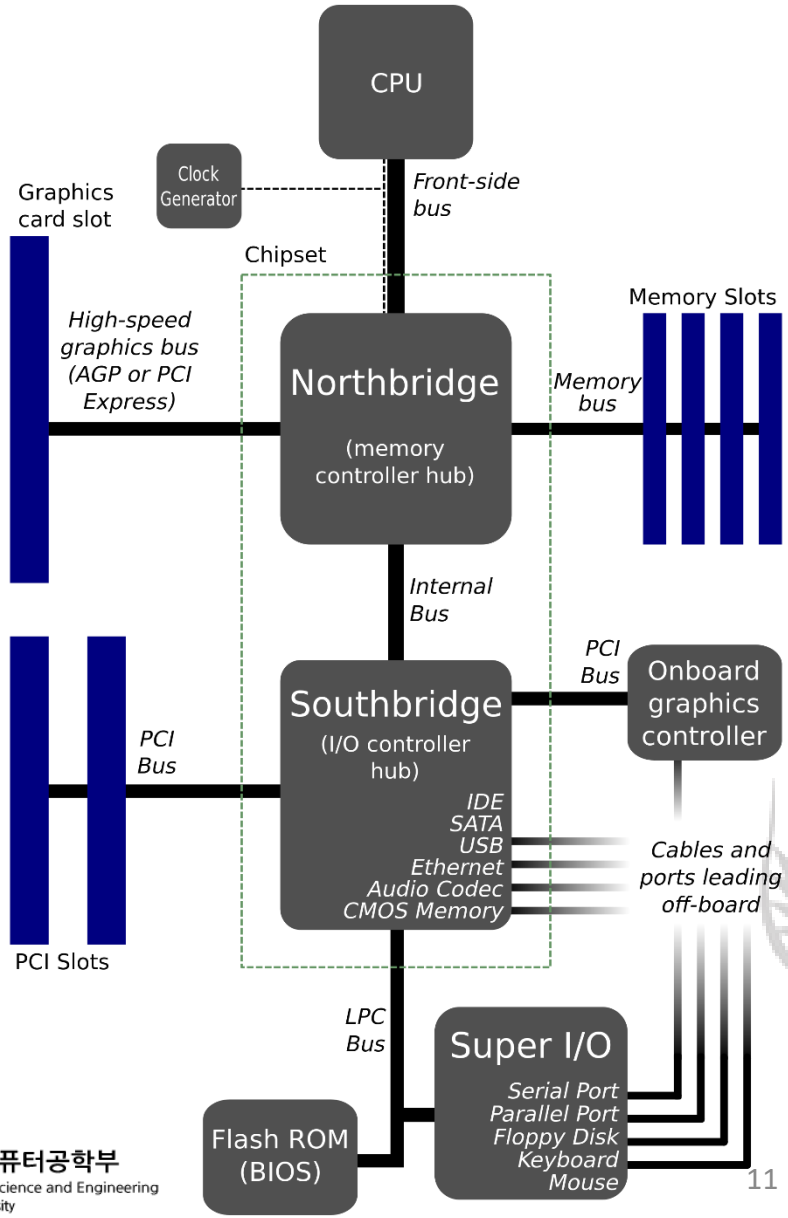



<이중 버스 구조>

# North and South-bridges

- Northbridge
  - 고속 장치를 제어하는 버스 컨트롤러
    - CPU, 메모리, PCI-E 등
- Southbridge
  - 비교적 저속의 장치를 제어하는 버스 컨트롤러
    - IDE, SATA, USB 등
  - Northbridge에 연결됨

❖ 참고: I/O 컨트롤러, I/O 허브 등으로 불리기도 함
  ❖ 너무 일반적인 용어라 불명확함



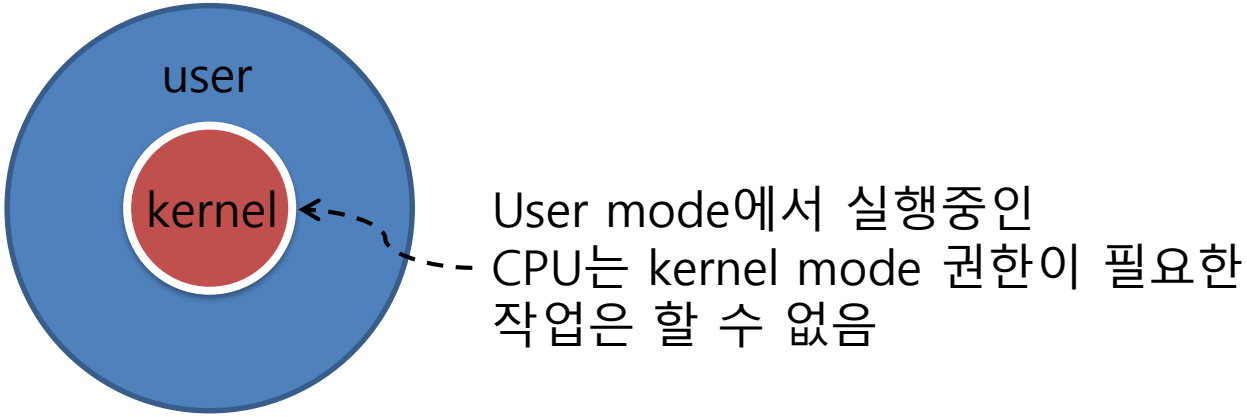
# OS와 연관된 컴퓨터 시스템 개념

# OS와 연관된 컴퓨터 시스템 개념

- User mode and Kernel mode
- Event handling Mechanisms: Interrupt
- I/O Device basic concept
- I/O Device access 기법: I/O Instruction and Memory Mapped I/O
- I/O 처리 기법: Polling and DMA

# User mode와 Kernel mode

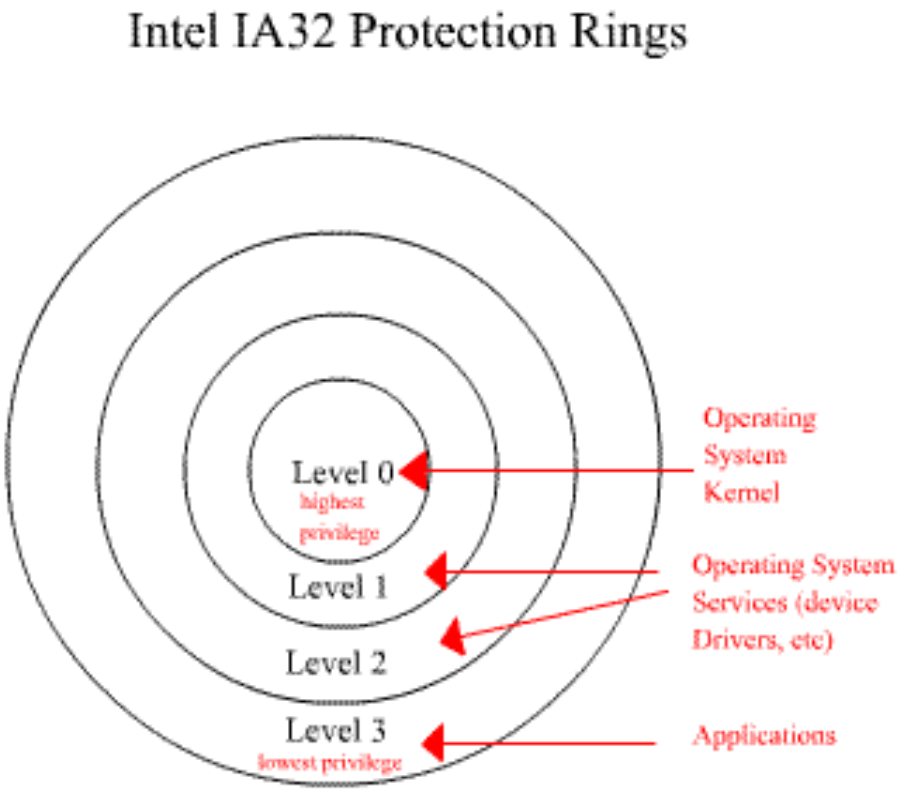
- CPU의 2가지 이상의 실행 모드
  - System protection을 위해서 필요
    - 실행 모드의 권한에 따라 접근할 수 있는 메모리, 실행 가능한 명령어가 제한됨
  - 각각의 모드 별로 권한(privilege)이 설정 됨
    - Intel – ring 0~3, 4개의 모드 제공
    - ARM 등 – 2개의 모드 제공

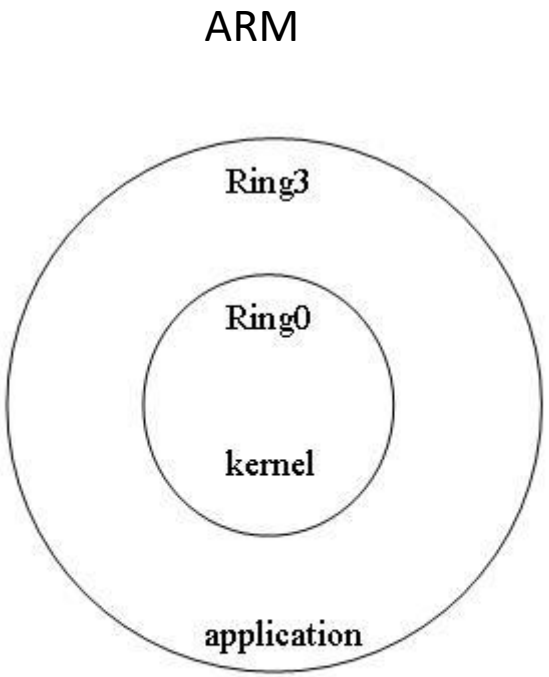


# Privileged instructions

- HLT (Halts the processor)
- CLTS (Clear task switch flag)
- LGDT,LIDT,LLDT (Loads GDT,IDT,LDT register)
- LTR (Load task register)
- LMSW (Load machine status word)
- Mov CRn,…. (moves to control register)
- Mov DRn,.. (moved to debug registers)
- Mov TRn,…. (moves to test registers)

# Protection Rings



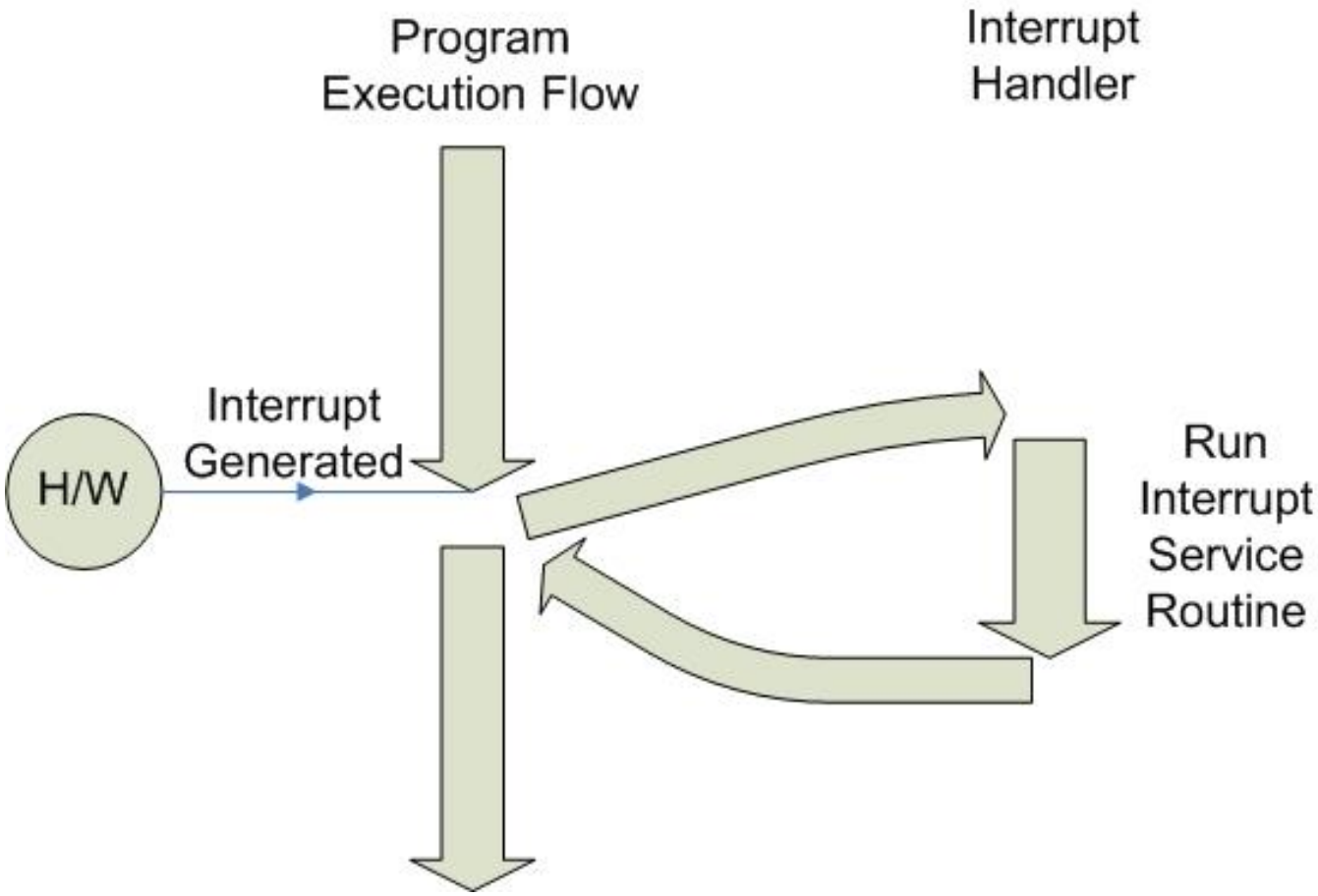
# User mode와 Kernel mode (cont.)

- Kernel mode
  - 모든 권한을 가진 실행 모드
  - 운영체제가 실행되는 모드
  - Privilege 명령어 실행 및 레지스터 접근 가능
    - 예) I/O 장치 제어 명령어, memory management register – CR3
- User mode
  - Kernel 모드에 비해 낮은 권한의 실행 모드
  - 어플리케이션이 실행되는 모드
  - Privilege 명령어 실행은 불가능
- 실행 모드 전환 (execution mode switch)
  - CPU의 실행 모드 설정은 시스템 보호가 목적
  - User mode에서 실행중인 어플리케이션이 kernel mode의 권한이 필요한 서비스를 받기 위한 방법이 필요!

# CPU의 이벤트 처리 기법: Interrupt

- HW Interrupt: 비동기적 이벤트를 처리하기 위한 기법
  - 예) 네트워크 패킷 도착 이벤트, I/O 요청
- 인터럽트 처리 순서
  - 현재 실행 상태(state)를 저장
  - ISR(interrupt service routine)로 점프
  - 저장한 실행 상태(state)를 복원
  - 인터럽트로 중단된 지점부터 다시 시작
- Note
  - 인터럽트에는 우선순위가 있으며, H/W 장치마다 다르게 설정됨
  - ISR은 짧아야 함: 너무 길면 다른 인터럽트들의 대기 지연
  - Timesharing역시 timer interrupt의 도움으로 가능하게 된 기술

# Interrupt (cont.)

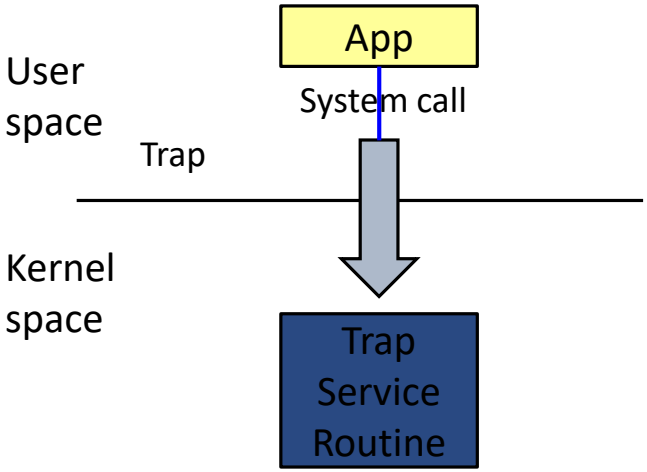


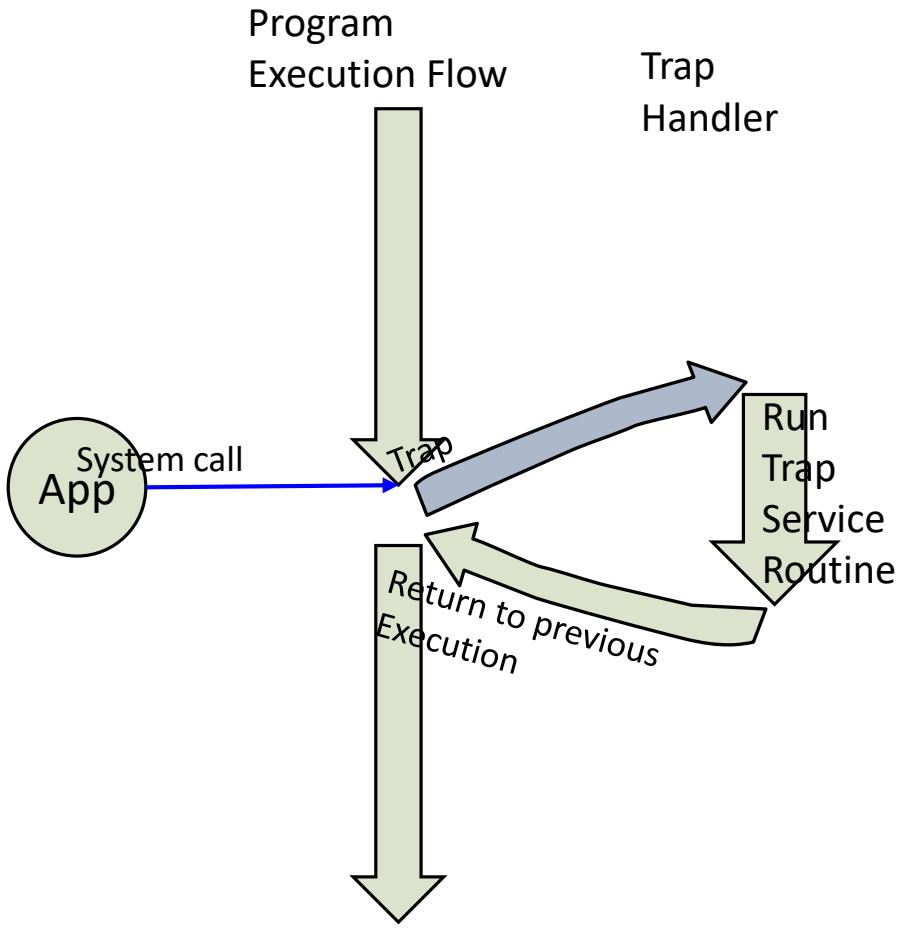

< Flow of handling interrupt >

# 이벤트 처리 기법: Trap (SW Interrupt)

- 동기적인 이벤트를 처리하기 위한 기법
  - 예) divide by zero와 같은 프로그램 에러에 의해 발생, Trap handler에 의해 처리
  - 현재 동작 중인 작업에 의해 발생하므로, HW 인터럽트와 달리 실행 상태(state)를 저장/복원 하지 않음

# Trap (cont.)




< Flow of handling trap>

# I/O Device Basic Concepts

- Bus
  - CPU, RAM, I/O장치 간 데이터가 전송되는 통로
  - 3가지의 버스로 구성됨
    - Data, Address, Control
- Device Registers
  - 보통 하드웨어 장치는 4종류의 레지스터를 가짐
  - Control register, Status register, Input register, Output register
  - 레지스터들은 메인 메모리의 일부 영역에 매핑
    - 매핑된 영역의 주소만 알면, CPU에서 접근 가능
- Controller
  - 고수준의 I/O요청을 저수준의 장치 명령어로 해석하는 디지털 회로
  - 장치와 직접 상호작용

# I/O device access 기법: I/O instruction

- Controller는 보통 1개 혹은 그 이상의 레지스터를 가짐
  - Data, control signal을 처리 하기 위함

- CPU는 controller의 레지스터의 bit패턴을 읽고 씀으로써 장치와 통신함
  - 이러한 기능을 수행하기 위한 명령어들이 제공됨
  - 예) Intel CPU의 I/O 명령어
    - in, out, ins, outs …
    - HW Control register 마다 할당된 port 를 지정하여 데이터를 주고 받음

# I/O device access 기법: Memory Mapped I/O

- 장치 레지스터들을 시스템 메모리 공간에 매핑하여 사용
  - 장치 레지스터가 메모리 주소에 매핑이 되면, 장치 레지스터들은 주소 공간의 일부로 여겨짐

- CPU는 일반적인 명령어를 사용하여 I/O 작업을 수행
  - 예) mov, and, or, xor …

# I/O 처리 기법: Interrupt

- IO를 요청하고, 그 종료를 확인하는 방법
  - Interrupt: CPU는 다른 작업을 수행하다가, 장치로부터 종료 메시지를 인터럽트로 수신하면, ISR을 수행하여 IO 종료 작업을 수행함
  - Polling: CPU가 작업의 종료를 직접 계속해서 확인하는 방식

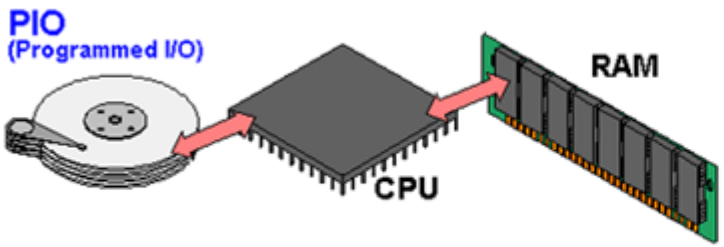
- Interrupt 방식
  - 장점: CPU가 IO 작업 종료를 대기하지 않고, 다른 작업을 수행하여 CPU 활용률을 높일 수 있음
  - 단점
    - 다른 작업으로 전환하는데 드는 비용이 클 수 있음
    - IO 작업의 종료를 즉각 확인할 수 없음

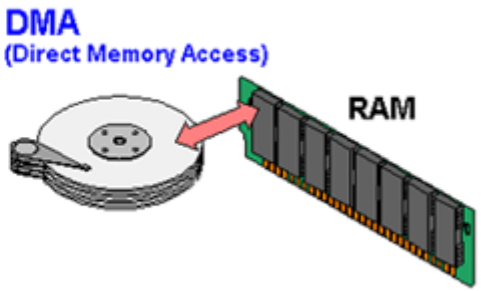
# I/O 처리 기법: Polling

- Loop이나 time-delayed loop안에서 특정 이벤트의 도착 여부를 확인하면서 기다리는 방법
- 인터럽트 핸들러를 등록하는 방식과 반대되는 개념
  - Polling 은 매 순간 이벤트의 발생 여부를 확인
- 장점
  - Controller나 장치가 충분히 빠른 경우, IO 작업의 종료를 즉각 확인하고 처리 가능
- 단점
  - 이벤트 도착 시간이 길 경우, Polling 은 CPU time을 낭비
    - 컴퓨터 시스템에서 CPU time은 매우 귀중한 자원!!
- Polling 은 흔히 programmed I/O(PIO)로 알려진 방식

# I/O 처리 기법: Direct Memory Access (DMA)

- Interrupt와 polling의 단점
  - Polling 을 사용할 경우, 모든 I/O 연산은 CPU에 의해 진행
  - Interrupt 를 사용하더라도 데이터의 송수신은 CPU가 담당
- 전송할 데이터가 클 경우, CPU를 장치의 상태 확인 및 버스에 데이터를 쓰는 행위에 사용하는 것은 낭비
  - I/O를 위한 별도의 프로세서가 있다면?




* https://www.pcmag.com/encyclopedia/term/41601/dma

# Direct Memory Access (DMA) (cont.)

- DMA controller: 특정 목적을 위한 프로세서 추가
  - CPU와 DMA controller간의 통신으로 I/O를 수행
  - CPU가 DMA controller에게 I/O를 요청하면
    DMA controller는 CPU를 대신하여 I/O장치와 메인 메모리 사이의 데이터 전송을 수행
    - CPU는 I/O 기간 동안 다른 일을 수행

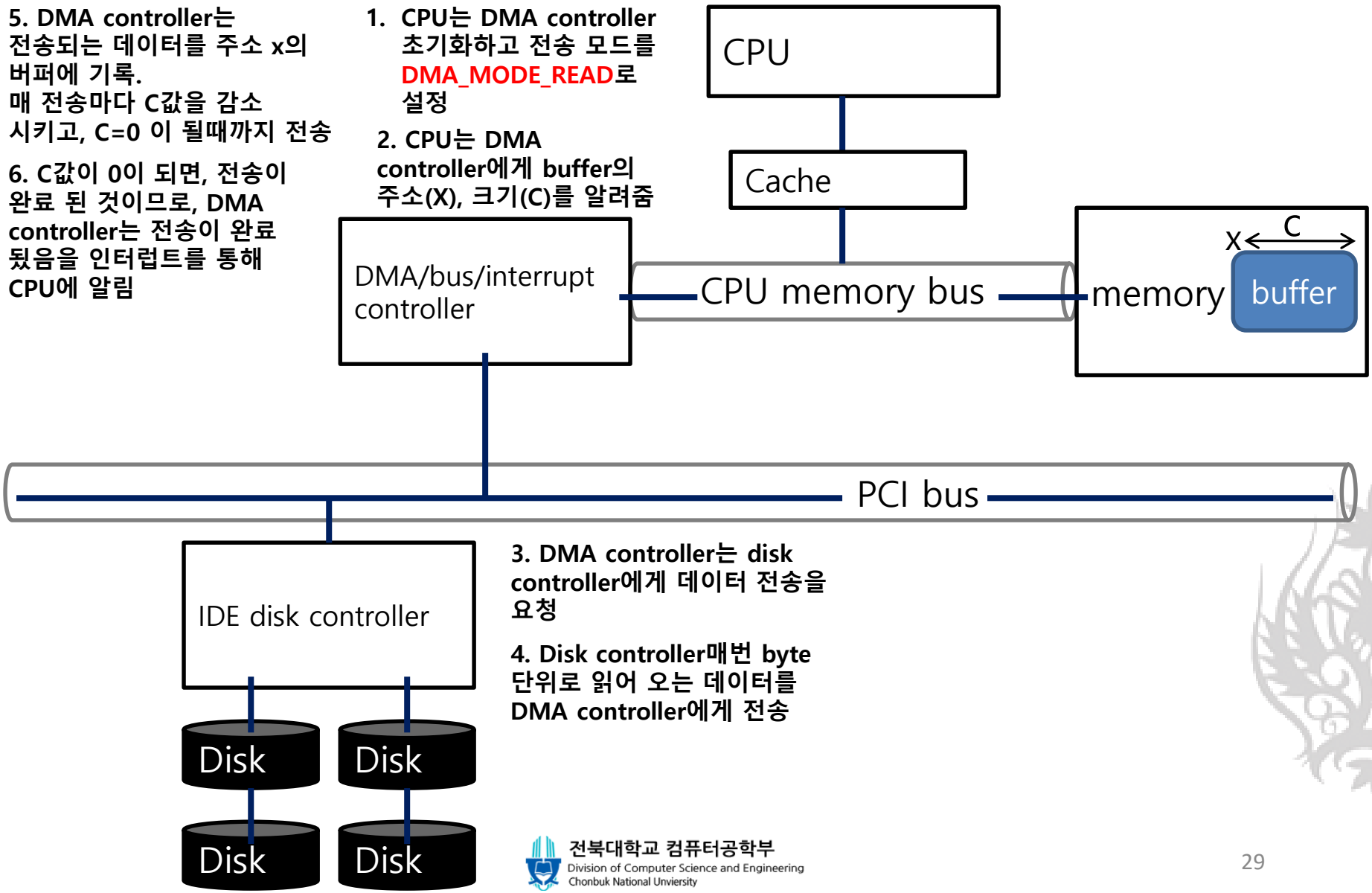
# DMA-Read

5. DMA controller는 전송되는 데이터를 주소 x의 버퍼에 기록.
매 전송마다 C값을 감소 시키고, C=0 이 될때까지 전송

6. C값이 0이 되면, 전송이 완료 된 것이므로, DMA controller는 전송이 완료 됐음을 인터럽트를 통해 CPU에 알림

1. CPU는 DMA controller 초기화하고 전송 모드를 DMA_MODE_READ로 설정

2. CPU는 DMA controller에게 buffer의 주소(X), 크기(C)를 알려줌




3. DMA controller는 disk controller에게 데이터 전송을 요청

4. Disk controller매번 byte 단위로 읽어 오는 데이터를 DMA controller에게 전송

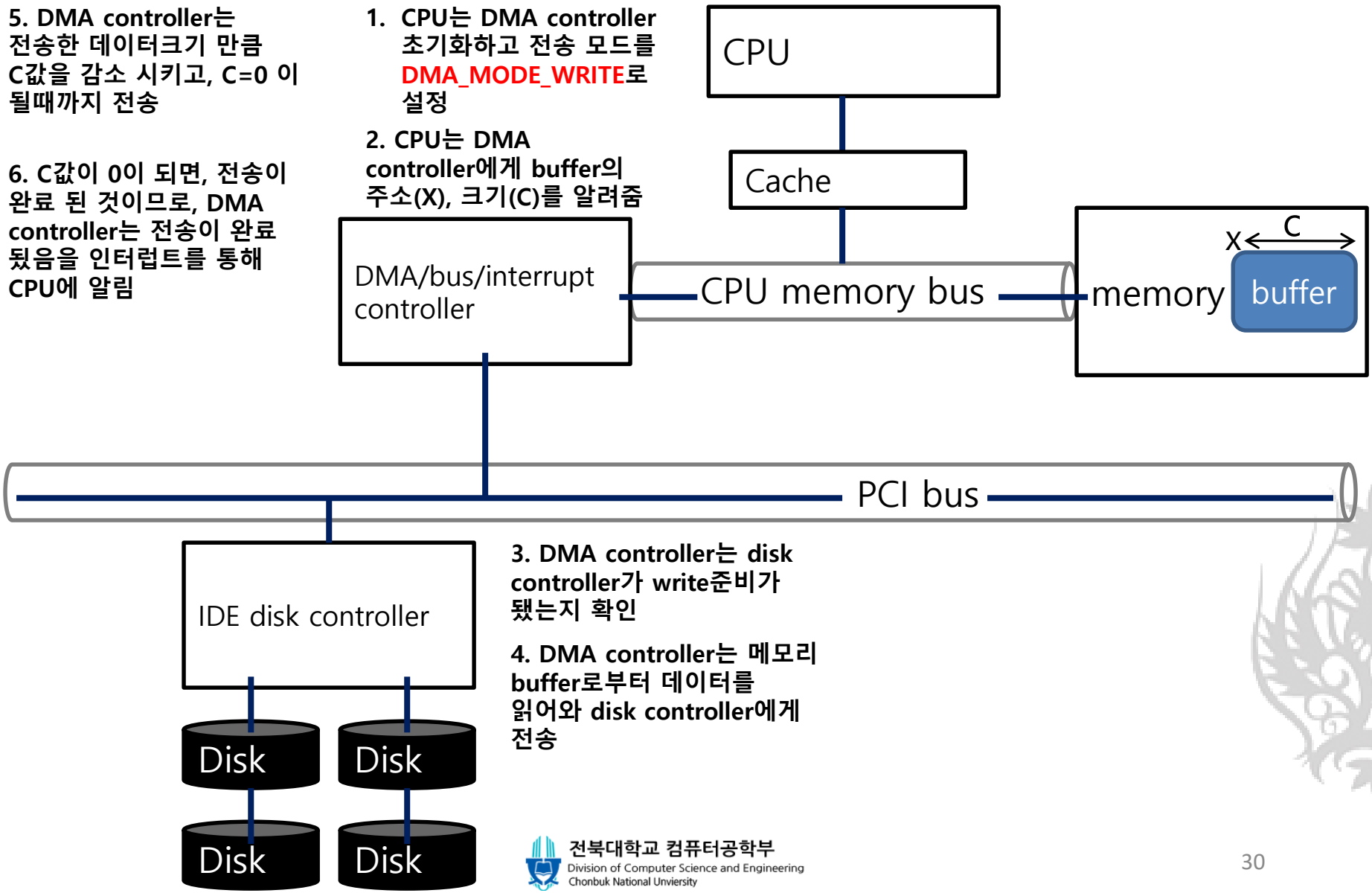
# DMA-Write

5. DMA controller는 전송한 데이터크기 만큼 C값을 감소 시키고, C=0 이 될때까지 전송

6. C값이 0이 되면, 전송이 완료 된 것이므로, DMA controller는 전송이 완료 됐음을 인터럽트를 통해 CPU에 알림

1. CPU는 DMA controller 초기화하고 전송 모드를 DMA_MODE_WRITE로 설정

2. CPU는 DMA controller에게 buffer의 주소(X), 크기(C)를 알려줌




3. DMA controller는 disk controller가 write준비가 됐는지 확인

4. DMA controller는 메모리 buffer로부터 데이터를 읽어와 disk controller에게 전송

# DMA vs Others: 어느 것이 나은가

- DMA
  - 추가적인 hardware가 필요하다: 비용 증가
  - ISR을 거쳐 완료를 인지하게 되어 처리 경로 증가: 약간의 속도 저하

- 성능 – DMA를 최대로 활용하기 위해서는
  - 적당한 parallelism이 필요 (병렬성)
  - DMA가 일하는 동안 CPU가 놀면 소용이 없음

# 운영체제에서 알아야 할 HW이슈 정리

- 컴퓨터 시스템의 전체적인 구성
- HW의 성능상 특성
- I/O의 동작원리

# 시스템 부트!

- **부트스트랩 (Bootstrap) 프로그램이 전원 인가 즉시 시작**
  - ROM, EPROM에 적재 (firmware)
  - 시스템 초기화 (BIOS 수행)
  - 약속된 위치에 저장된 운영체제 커널로 수행을 넘기고 종료

# Appendix

# RISC vs CISC

- RISC (Reduced Instruction Set Computing)
  - 적은 수의 명령어만으로 명령어 집합(instruction set)을 구성
  - 복잡한 연산은 명령어를 조합하여 수행
  - 고정 길이 명령어 사용, pipeline이 가능해짐
  - Compiler가 복잡해짐
  - Ex) Sparc, Arm processor
- CISC (Complex Instruction Set Computing)
  - 프로그래밍에 필요한 모든 명령어들을 제공하는 명령어 집합
  - 명령어 해석(decoding)이 오래 걸림
    - 회로 복잡도 증가
  - 명령어 실행 cycle수가 다르므로, pipeline이 어려움
  - Ex) Intel Pentium processor