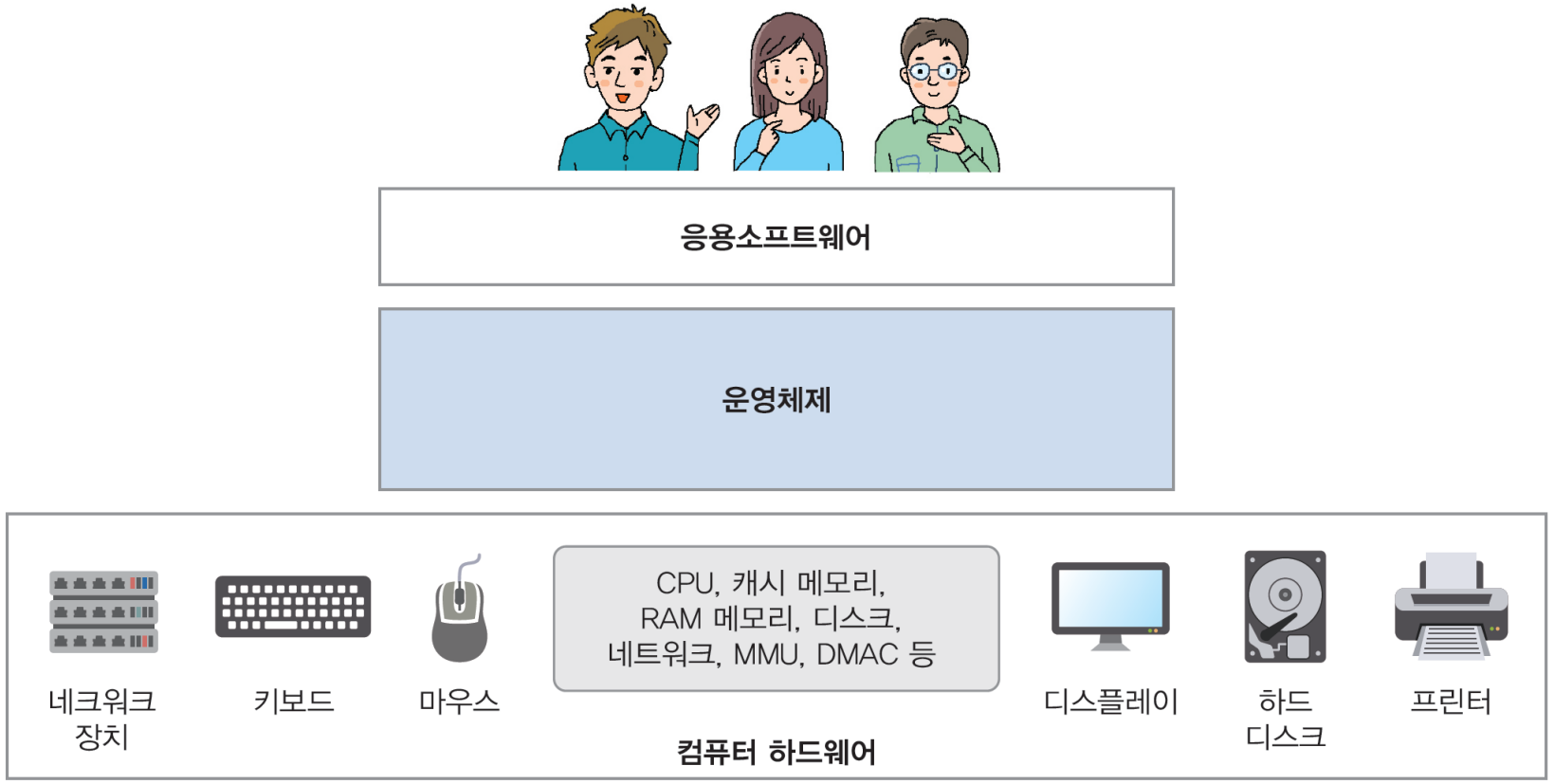
임베디드시스템설계

운영체제 기초2

Byungjin Ko
Department of Intelligent Robotics

2. 컴퓨터 시스템과 하드웨어

컴퓨터 시스템을 구성하는 계층



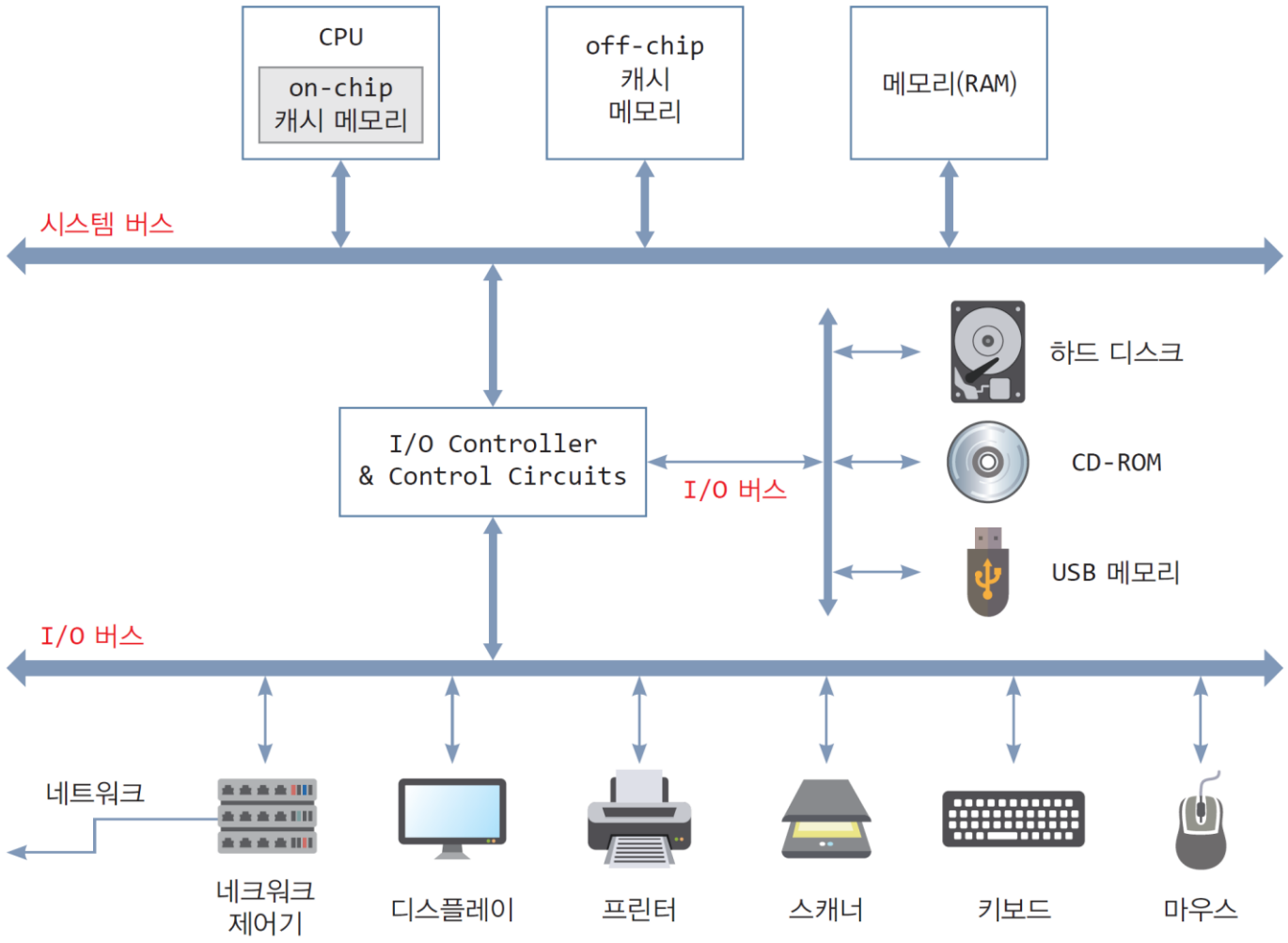
컴퓨터 시스템의 범위

- 컴퓨터 시스템의 계층
  - 응용프로그램 층
  - 운영체제 층
  - 컴퓨터 하드웨어 층

- 컴퓨터 시스템 계층 구조의 특징
  - 사용자는 응용프로그램과 GUI/도구프로그램(툴 /유틸리티)을 통해 컴퓨터 활용
  - 하드웨어는 모두 운영체제의 배타적 독점적 지배 받음
  - 사용자나 응용프로그램의 하드웨어에 대한 직접 접근 불허
    - 반드시 운영체제를 통해서만 접근 가능

- 계층 구조로 보는 운영체제의 기능
  - 사용자가 하드웨어에 대해 몰라도 컴퓨터를 사용할 수 있도록 함
  - 응용프로그램과 하드웨어 사이의 중계
    - 위로는 응용프로그램과 아래로는 하드웨어와의 인터페이스

컴퓨터 하드웨어 구성


CPU
on-chip
캐시 메모리
off-chip
캐시
메모리
메모리(RAM)
시스템 버스
I/O Controller
& Control Circuits
I/O 버스
하드 디스크
CD-ROM
USB 메모리
I/O 버스
네트워크
네크워크
제어기
디스플레이
프린터
스캐너
키보드
마우스

컴퓨터 하드웨어 설명(1)

- CPU(Central Processing Unit)
  - 프로그램 코드(기계 명령)를 해석하여 실행하는 중앙처리장치
  - 컴퓨터의 가장 핵심 장치
  - 전원이 공급될 때 작동 시작, 메모리에 적재된 프로그램 실행

- 메모리
  - CPU에 의해 실행되는 프로그램 코드와 데이터가 적재되는 공간
  - 반도체 메모리 RAM
  - 프로그램은 실행되기 위해 반드시 메모리에 적재되어야 함

- 캐시 메모리(Cache Memory)
  - 배경
    - CPU 처리속도가 메모리 속도에 비해 빠르게 발전 -> 느린 메모리 때문에 CPU의 대기 시간이 늘게 되었음
    - CPU의 프로그램 실행 속도를 높이기 위해, CPU와 메모리 사이에 소량의 빠른 메모리(고가의 메모리)를 설치하게 되었음
  - 온칩 캐시(on-chip) – CPU 내부에 만들어진 캐시, 오늘날 대부분 온칩 캐시
  - 오프칩 캐시(off-chip) – CPU 외부에 설치되는 캐시
  - 캐시 메모리가 있는 경우 CPU는 캐시 메모리에서만 프로그램 실행
    - 실행하고자하는 프로그램과 데이터는 메모리에 먼저 적재되고 다시 캐시로 옮겨져야 함
    - 캐시는 용량이 작기 때문에 현재 실행할 코드와 데이터의 극히 일부분만 저장

컴퓨터 하드웨어 설명(2)

- 장치들
  - 키보드, 프린터, 스캐너 등

- 버스(bus)
  - 하드웨어들이 데이터를 주고받기 위해 0과 1의 디지털 신호가 지나가는 여러 가닥의 선을 다발로 묶어 부르는 용어

  - 비유
    - 컴퓨터의 버스 : 도로
    - 컴퓨터의 데이터 : 도로에 다니는 자동차

  - 버스의 종류(버스에 지나다니는 정보에 따라)
    - 주소 버스(address bus) - 주소 신호가 지나다니는 버스(도로)
    - 데이터 버스(data bus) - 데이터 신호가 지나다니는 버스(도로)
    - 제어 버스(control bus) - 제어 신호가 지나다니는 버스(도로)

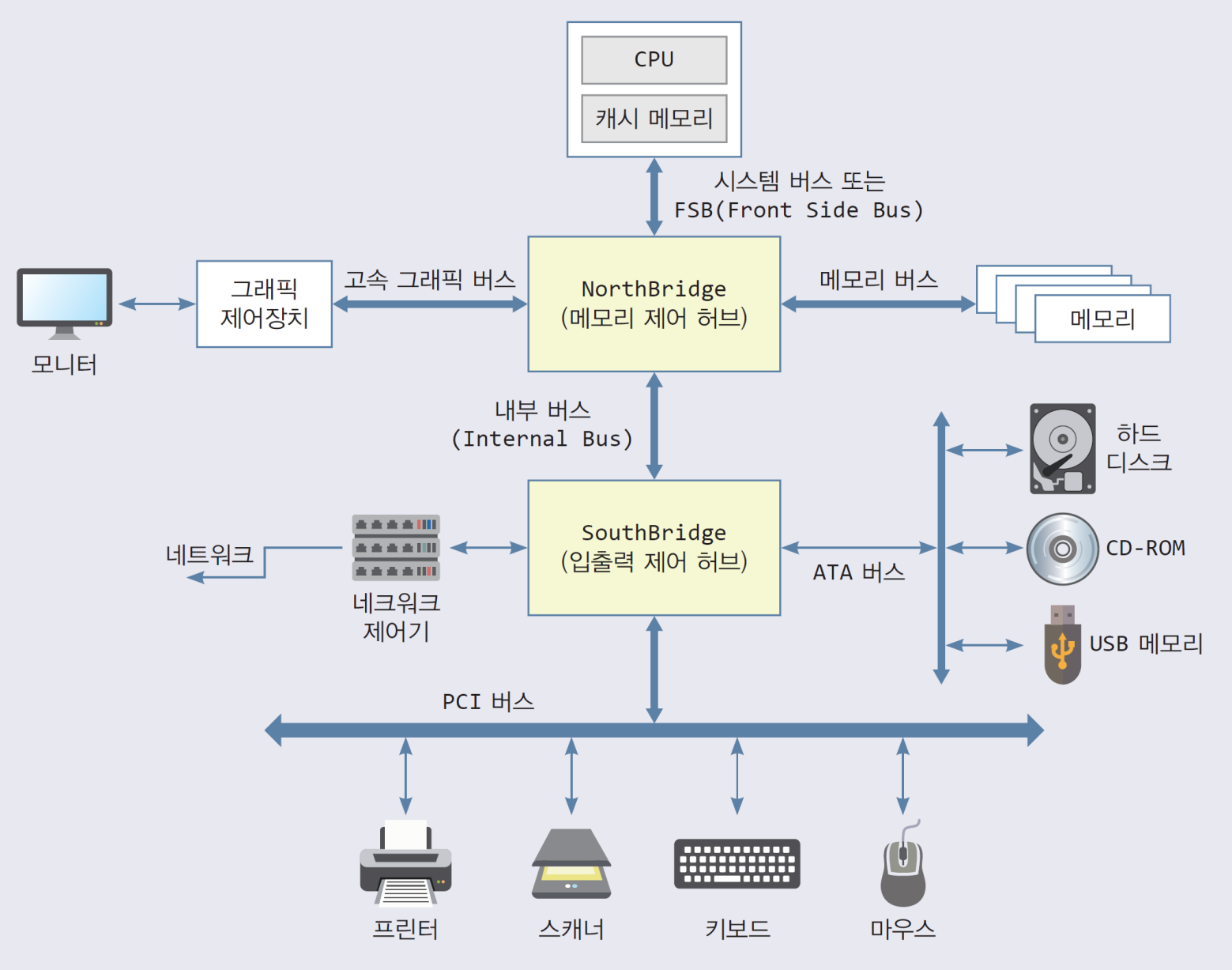
  - 주소
    - 메모리, 입출력 장치나 저장 장치 내에 있는 저장소(레지스터들)에 대한 번지
    - 주소 버스는 주소 값이 전달되는 여러 선의 다발
    - **CPU는** 메모리나 입출력 장치에 값을 쓰거나 읽을 때 반드시 **주소를 발생**시킴

컴퓨터 하드웨어 설명(3)

- 목적에 따라 버스 구분
  - 시스템 버스(system bus)
    - CPU, 캐시 메모리, 메모리 등 빠른 하드웨어들 사이에 데이터 전송
    - **고속도로**에 비유
  - 입출력 버스(I/O bus)
    - 상대적으로 느린 입출력 장치들로부터 입출력 데이터 전송
    - **일반 도로**에 비유

- I/O controllers & control circuit
  - 입 출력 장치 들을 제어하기 위한 여러 하드웨어
    - 입출력 장치에게 명령 하달
    - 메모리와 입출력 장치 사이에 혹은 CPU와 입출력 장치 사이에 데이터 전달 중계

현대 PC의 구조


CPU
캐시 메모리
시스템 버스 또는
FSB(Front Side Bus)
NorthBridge
(메모리 제어 허브)
그래픽
제어장치
고속 그래픽 버스
메모리 버스
메모리
모니터
내부 버스
(Internal Bus)
SouthBridge
(입출력 제어 허브)
하드
디스크
CD-ROM
USB 메모리
ATA 버스
네트워크
네크워크
제어기
PCI 버스
프린터
스캐너
키보드
마우스

CPU와 메모리의 관계

- CPU
  - 능동적 소자. 메모리 액세스 시 주소 발생

- 32비트 CPU, 32비트 운영체제, 32비트 컴퓨터란?

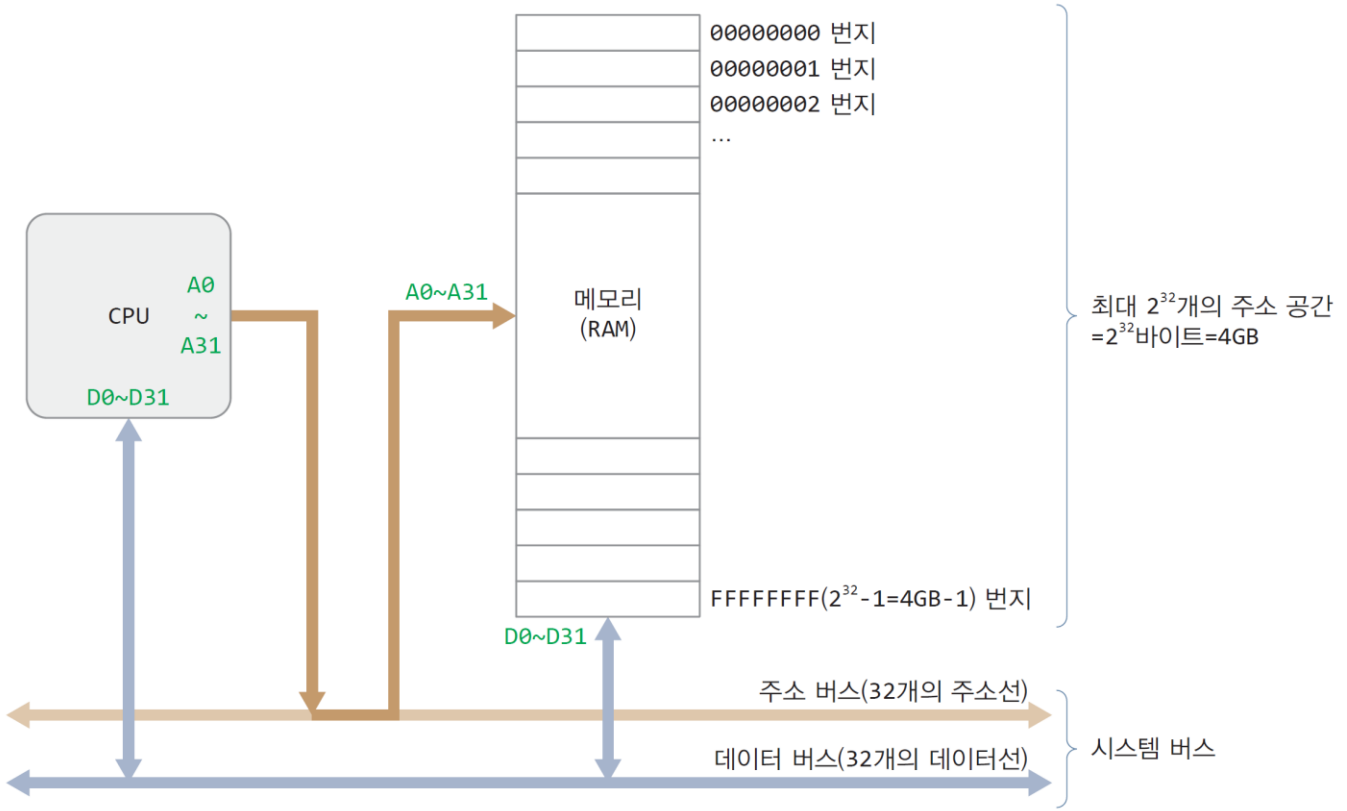
  1) CPU에 **32개의 주소선** 있음
     - CPU의 액세스 범위 : 2^{32}개의 서로 다른 주소(0 ~ 2^{32}-1 번지)
     - CPU가 최대 액세스할 수 있는 메모리의 크기 : 4GB
     - 한 번지의 공간이 1바이트이므로, **2^{32}개의 주소 = 2^{32} 바이트 = 4GB**

     - 2^{10} = 1KB
     - 2^{20} = 2^{10}x2^{10} = 1MB
     - 2^{30} = 2^{20}x2^{10} = 1GB
     - 2^{32} = 2^{2}x2^{30} = 4x1GB = 4GB

     - 32비트 CPU를 가진 컴퓨터에 4GB이상 메모리를 설치할 경우
       - 4GB를 넘어선 영역은 사용할 수 없음

  2) CPU에 입출력되는 **32개의 데이터 선** 있음
     - 한 번에 32비트 읽고 쓰기 가능(32비트는 4개의 주소)

32비트 CPU의 주소선, 데이터선과 메모리



※ CPU에서 A0~A31은 32개의 주소선을, D0~D31은 32개의 데이터선을 나타낸다. 이들은 시스템 버스와 연결되어 다른 하드웨어 장치들에게 주소를 보내거나 데이터를 주고받는데 사용된다.

※ 주소에 초점을 맞추기 위해 32비트 컴퓨터를 32개의 주소선을 가진 것으로 설명하였지만, 32비트 CPU는 32개 데이터선을 통해 32비트를 한 번에 메모리에서 읽고 쓰고 한 번에 32비트 더하기를 한다.

CPU 기계 명령(instruction)

- CPU 명령
  - CPU가 해석하고 실행할 수 있는 기계 명령(machine instruction)
    - C 언어나 자바의 프로그램 소스 코드와 다름
    - CPU를 설계하는 기업이 명령어들, 명령어 개수, 형태 등을 결정
    - CPU의 명령 개수는 수십개~수백개
    - CPU마다 명령 이름, 기계어 코드, 크기, 개수 등이 다름
    - CPU 사이에 명령들의 호환성 없음
      - C 프로그램을 어떤 CPU를 대상으로 컴파일하였는지에 따라 기계어가 달라지므로, 컴파일된 코드는 CPU 사이에 호환성 없음
  - CPU 명령 사례

```
어셈블리어 명령    기계 명령
mov ecx, 51      ; b9 33 00 00 00     ecx 레지스터에 51 저장
add ax, 8        ; 83 c4 08           ax 레지스터에 8 더하기
push ebp         ; 55                 ebp 레지스터의 값을 스택에 저장
call _printf       ; e8 00 00 00 00     _printf 함수 호출
ret 0            ; c3                 이 함수를 호출한 곳으로 리턴(0 값과 함께)
```

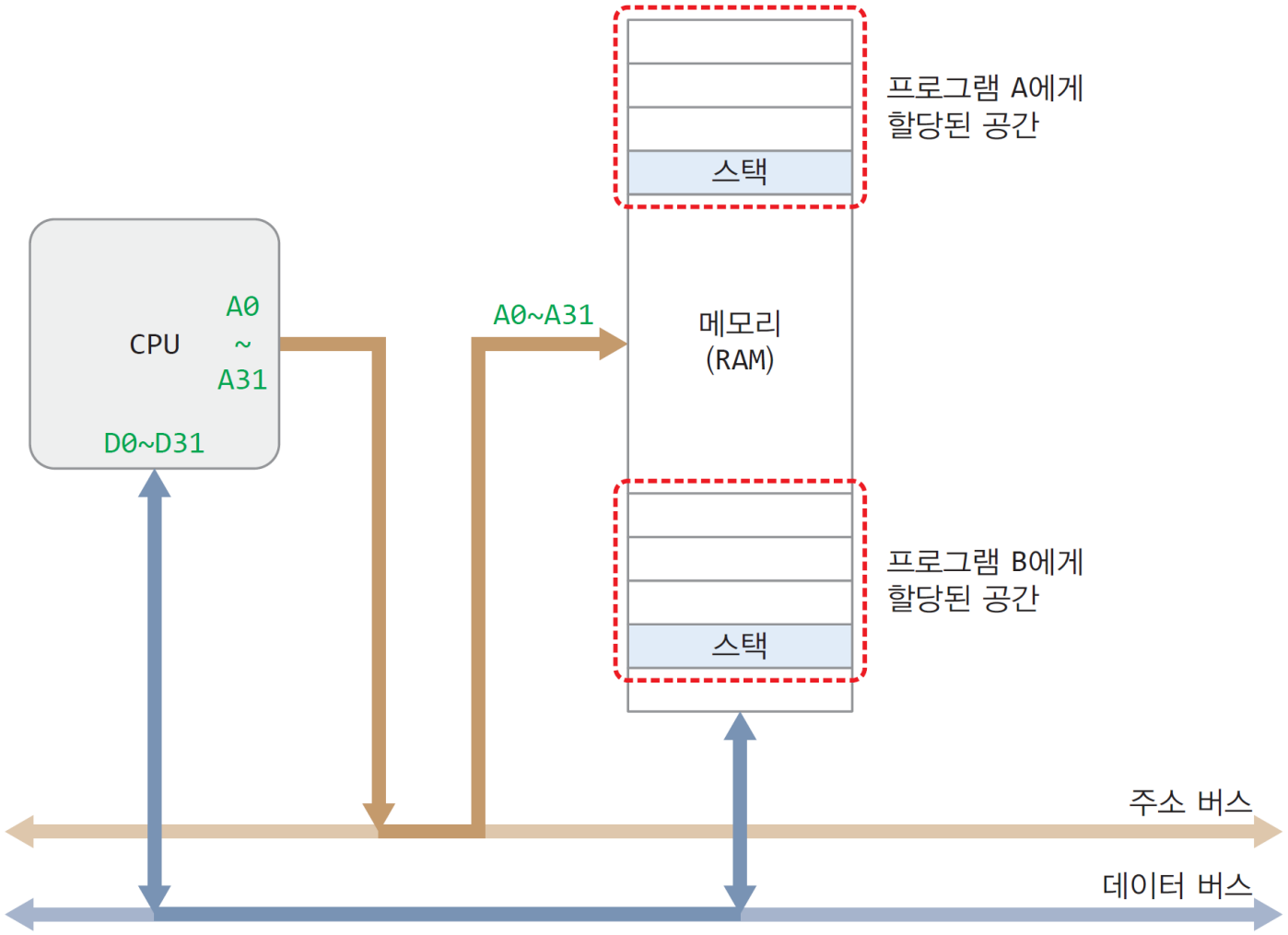
스택(Stack)은 어디에 있는가?

- 프로그램이 실행되기 위해 운영체제에 의해 할당되는 4공간
  - **코드**(code) 공간 – 프로그램 코드 적재
  - **데이터**(data) 공간 - 전역 변수들이 적재되는 공간
  - **힙**(heap) 공간 – 프로그램에서 동적 할당받는 공간
  - **스택**(stack) 공간 – 함수가 호출될 때 매개변수, 지역변수 등 저장

- 스택
  - 메모리의 일부를 스택으로 사용하도록 할당된 공간
    - 스택이라는 별도의 하드웨어 메모리가 있는 것 아님
  - 운영체제는 각 프로그램에게 스택 공간 할당
  - CPU의 SP (stack pointer) 레지스터는 현재 CPU가 실행중인 프로그램의 스택 꼭대기 주소를 가리킴
  - 스택에 저장되는 내용
    - 함수의 지역 변수들
    - 함수가 호출될 때 전달받은 매개변수 값들
    - 함수가 실행된 후 돌아갈 주소
    - 함수가 의도적으로 저장해 두기 위한 값

스택과 메모리

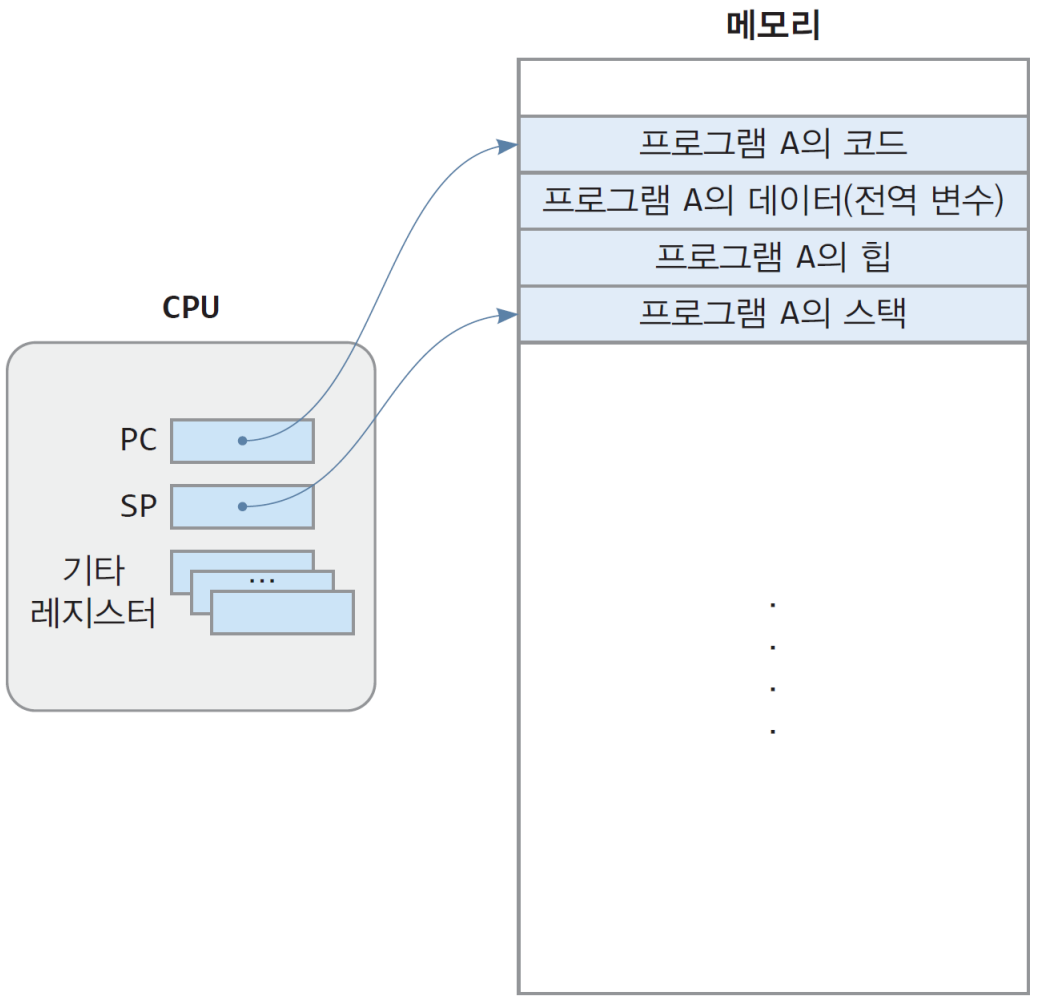
스택은 운영체제에 의해 프로그램마다 할당된 메모리의 일부 영역



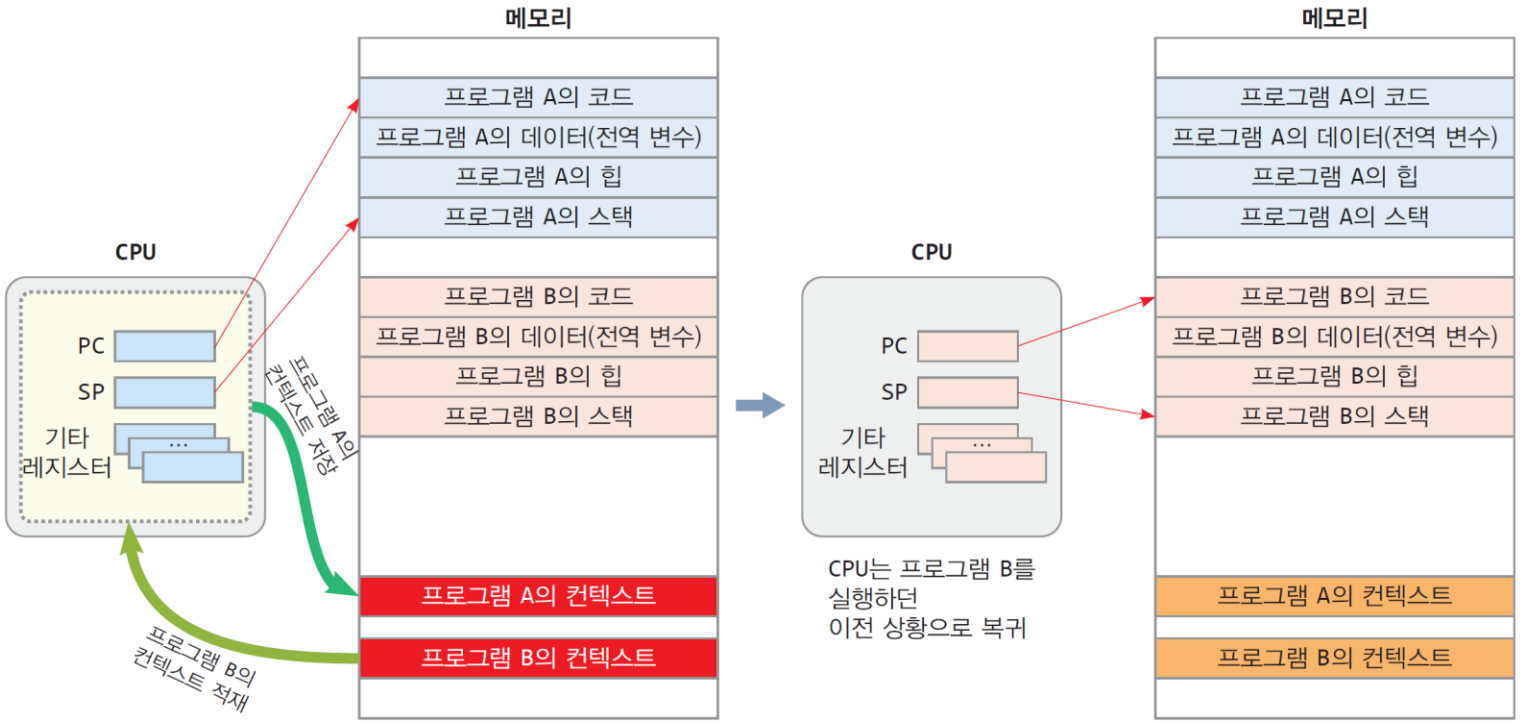
컨텍스트(Context)

- 컨텍스트(문맥이라고도 표현)
  - 한 프로그램이 실행 중인 일체의 상황 혹은 상황 정보
    - 메모리
      - 프로그램 코드와 데이터, 스택, 동적할당 받아 저장한 값
    - CPU 레지스터들의 값
      - PC에는 코드의 주소
      - SP에는 스택의 주소
      - 다른 레지스터는 이전의 실행 결과나 현재 실행에 사용되는 데이터 들
  - **축소 정의**
    - 현재 CPU에 들어 있는 레지스터의 값들
      - 메모리에 저장된 상황 정보는 그대로 있으니까
- 컨텍스트 스위칭
  - 발생
    - CPU가 현재 프로그램의 실행을 중지하고 다른 프로그램을 실행할 때
  - 과정
    - 현재 실행중인 프로그램의 컨텍스트(CPU레지스터들의 값)를 메모리에 저장
    - 새로 실행시킬 프로그램의 저장된 컨텍스트(CPU레지스터들의 값)를 CPU에 복귀

프로그램 A에서 프로그램 B로 컨텍스트 스위칭



프로그램 A의 컨텍스트 사례

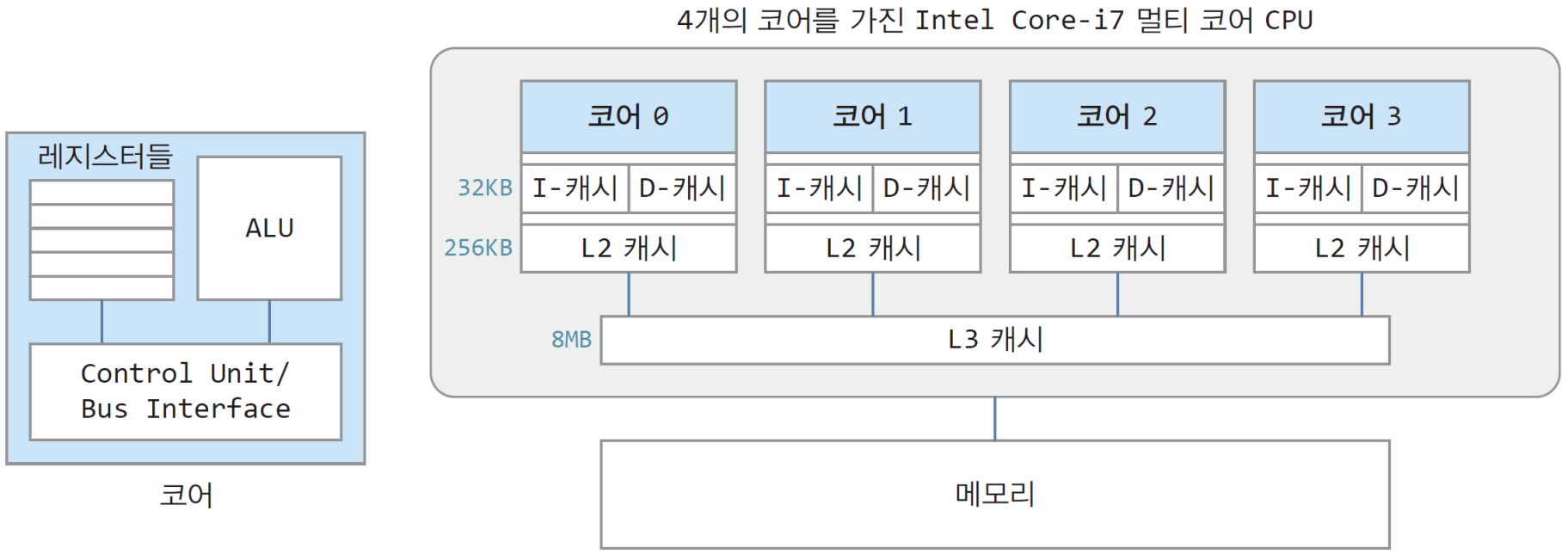


(a) 프로그램 A에서 프로그램 B로 컨텍스트 스위칭

(b) 컨텍스트 스위칭 후 프로그램 B 실행 재개

멀티 코어 CPU(Multi-core CPU)

- 2001년 IBM에 의해 PowerPC라는 멀티코어 CPU 개발
  - CPU 내부에 2개의 프로세서(processor) 포함
  - 2개의 프로그램을 동시에 실행
  - 코어는 완벽한 처리기(과거 개념의 CPU)



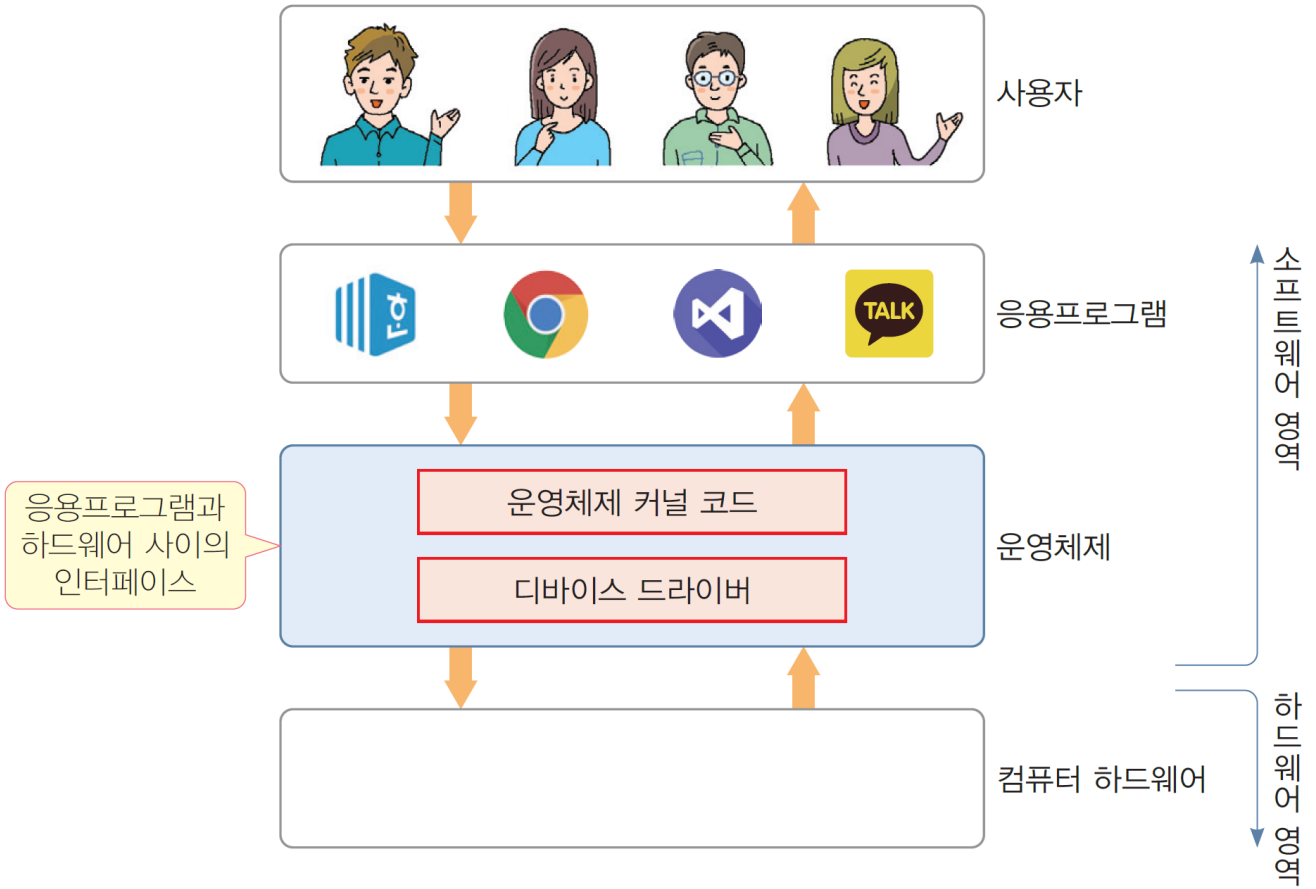
(a) 코어의 내부 구조

(b) 4개의 코어를 가진 Intel Core–i7 멀티 코어 CPU 사례

3. 컴퓨터 시스템의 계층 구조와 운영체제 인터페이스

컴퓨터 시스템 계층 구조

- 컴퓨터 시스템 계층
  - 사용자
  - 응용프로그램
  - 운영체제
    - 커널 코드
    - 디바이스 드라이버
  - 하드웨어



컴퓨터 시스템이 계층 구조로 설계된 이유

- 계층간 독립성 확보(칸막이가 있다고 생각)를 위해
  - 사용자
    - 운영체제나 하드웨어에 대해 몰라도 응용프로그램으로 컴퓨터 활용
  - 응용프로그램
    - 컴퓨터 하드웨어의 타입이나 구조, 제어 방법을 몰라도 개발 가능
    - 예)
      - CPU 크기, 메모리 크기가 얼마인지 모르고 프로그램 작성
      - 저장 장치가 하드디스크인지 SSD인지, 저장 장치의 크기는 얼마인지, 디스크 헤드는 몇개 있는지 몰라도 파일 입출력 프로그램 작성
    - 운영체제에게 요청하여 해결
    - 컴퓨터 하드웨어가 바뀌어도 응용프로그램을 다시 작성할 필요 없음
  - 운영체제
    - 운영체제는 장치 관련된 모든 작업을 디바이스 드라이버에게 요청
    - 응용프로그램과 하드웨어 사이의 인터페이스

왜 운영체제가 필요한가?

- 운영체제가 없다면
  - 응용프로그램이나 사용자가 직접 하드웨어를 제어해야 함
  - 하드웨어에 대한 지식이 필요하며, 충돌, 관리, 보안의 문제 발생

- 실례
  - 2 명이 동시에 프로그램을 실행시키고자 할 때, 이미 어떤 프로그램이 실행되고 있는 상황에서, 다른 사람이 프로그램을 실행시키고자 하면, 2개의 응용프로그램에 대한 스케줄링은 누가 할 것인가? **프로세스 관리**

  - 응용프로그램이 메모리가 필요할 때, 사용 중이지 않는 메모리와 사용중인 메모리로 나누어 관리하고, 응용 프로그램이 종료하면 사용한 메모리 회수하는 등 메모리 관리는 누가 할 것인가? **메모리관리**

  - 2명의 사용자가 동시에 프린터에 프린팅을 시킬 때 등, 장치 사용에 대한 충돌이 생기지 않게 하려면 누가 할 것인가? **장치 관리**

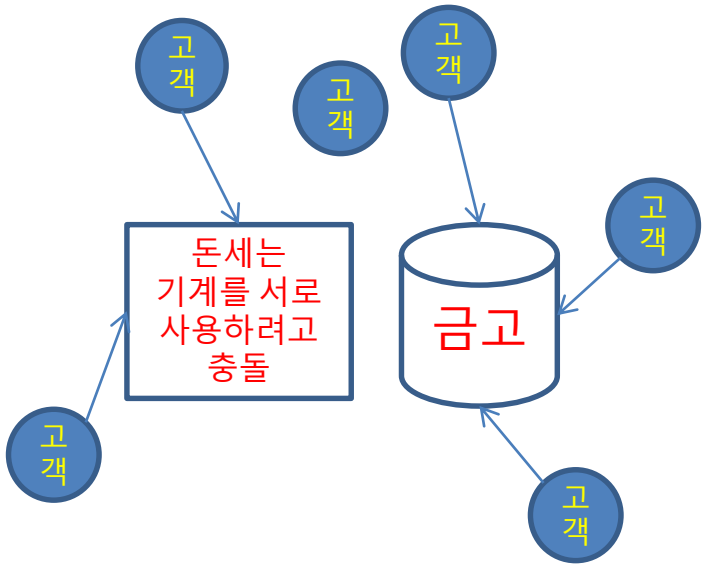
  - 데이터를 파일에 기록하려는데, 하드 디스크의 어느 위치에 기록해야 할지 결정해야 하고, 내가 기록한 디스크 위치에 다른 응용프로그램이 덮어쓰지 못하게 하려면 누가 할 것인가? **파일 시스템 관리**

  - 키보드에서 키가 입력되면 실행 중인 여러 응용프로그램 중 어떤 응용프로그램에게 키를 전달할지 누가 결정할 것인가? **입출력 관리**

- 운영체제의 필요성
  - 자원에 대한 충돌 해결, 성능 최적화, 사용자의 시스템 사용 효율화

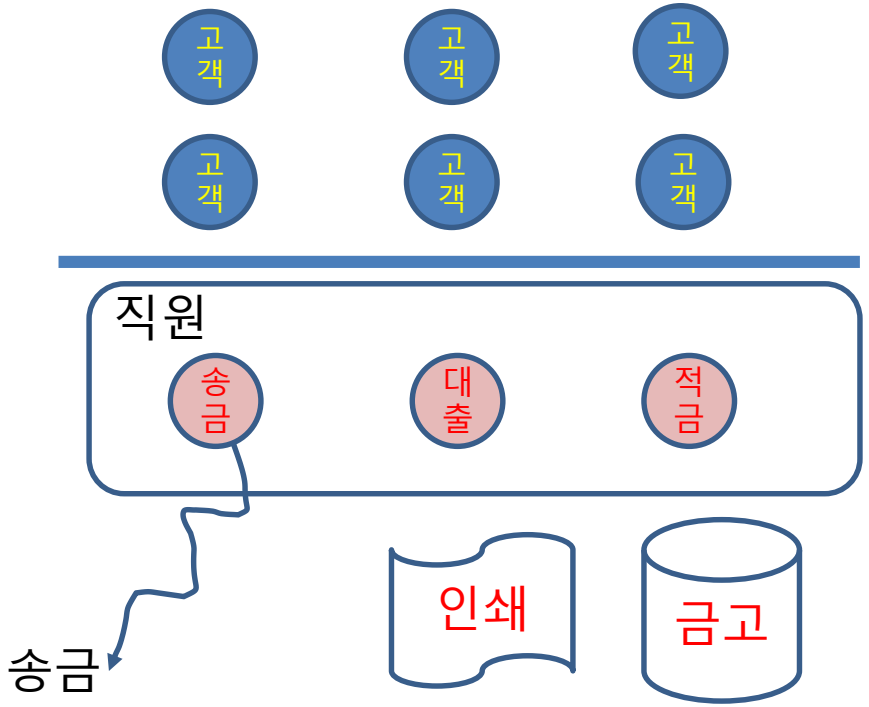
직원이 없는 은행과 직원 있는 은행 비교

은행직원이 없는 경우



비효율성, 충돌로 인한 오류, 지연,
보안 취약, 무질서 !!!

은행직원이 있는 경우



- 고객 : 응용프로그램
- 직원 : 운영체제의 여러 기능들(커널 함수들)

효율적, 충돌 해소, 보안 강화, 질서

- 사용자 모드 : 고객이 활동하는 영역
- 커널 모드 : 직원이 활동하는 영역

운영체제와 응용프로그램 사이의 관계

- 응용프로그램
  - 워드, 웹브라우저 등, 사용자가 컴퓨터를 활용하도록 작성된 다양한 프로그램들

- 응용프로그램에 대한 운영체제의 역할
  - 응용프로그램이 직접 하드웨어를 다루지 못하도록 차단
    - 운영체제가 하드웨어를 완벽히 독점 장악
    - 이유 : 응용프로그램들 사이의 하드웨어 사용 충돌을 막기 위함
  - 응용프로그램이 하드웨어를 사용하고자 할 때
    - 반드시 운영체제에게 요청 -> 운영체제가 대신하여 하드웨어 조작
    - 유일한 요청 방법 – **시스템 호출(system call)**
  - 응용프로그램과 하드웨어 사이의 인터페이스
  - 응용프로그램들의 실행 순서 제어
  - 응용프로그램들 사이의 통신 중계

운영체제와 사용자의 관계

- 사용자는 응용프로그램을 통해 컴퓨터 활용
  - 탐색기, 메모장 등

- 사용자에 대한 운영체제의 역할
  - 사용자가 하드웨어에 관한 지식이 없어도 컴퓨터 다루기 용이
  - 사용자가 하드웨어를 설치하거나 변경하는 것에 도움
  - 사용자에게 컴퓨터 시스템을 사용할 편리한 인터페이스 제공
    - UI, 마우스, 음성 명령 등
  - 컴퓨터의 사용을 돕는 여러 도구 응용프로그램(유틸리티) 제공
    - Windows의 탐색기와 작업 관리자
    - 리눅스의 쉘
  - 사용자 계정 관리
  - 사용자의 컴퓨터 사용 시간 계산, 과금 처리 등

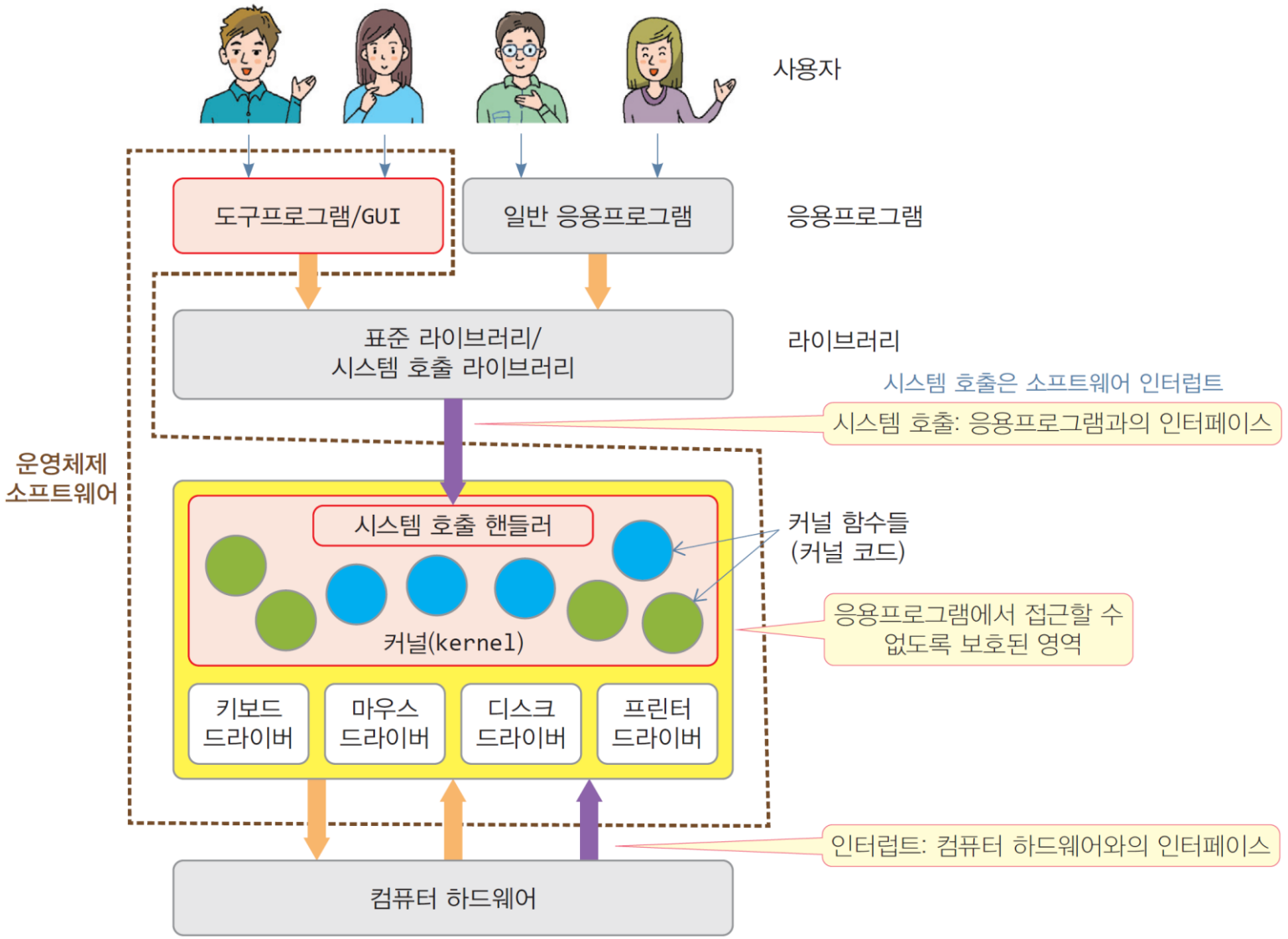
운영체제와 하드웨어의 관계

- 하드웨어를 제어하는 것은 전적으로 운영체제의 몫
  - 응용프로그램에서 printf(“hello”)
    - 디스플레이 장치에 “hello” 를 출력하는 일은 운영체제가 함
  - 응용프로그램에서 scanf()
    - 키보드로부터 문자를 입력받는 일은 운영체제가 함
- 운영체제
  - 사용자/응용프로그램과 하드웨어 사이의 중계자
  - 하드웨어 제어는 전적으로 운영체제의 기능
    - 하드디스크에서 파일을 읽거나 쓰기
    - 마우스의 클릭
    - 키보드 입력 받기
    - 네트워크를 통한 데이터 전송 혹은 수신
    - 디스플레이에 텍스트나 이미지, 그래픽 등 출력

운영체제의 전체 기능

- 프로세스와 스레드 관리
  - 프로세스/스레드의 실행, 일시 중단, 종료, 스케줄링, 컨텍스트 스위칭, 동기화
- 메모리 관리
  - 프로세스나 스레드에게 메모리 할당, 메모리 반환, 다른 프로세스/스레드로부터의 메모리 보호
  - 메모리를 하드 디스크의 영역까지 확장하는 가상 메모리 기술
- 파일 관리 혹은 파일 시스템 관리
  - 파일 생성, 저장, 읽기, 복사, 삭제, 이동, 파일 보호
- 장치 관리
  - 키보드, 마우스, 프린터 등 입출력 장치, 하드 디스크 등 저장 장치 제어
  - 입출력
- 사용자 인터페이스
  - 라인 기반 명령 입출력 창, 마우스와 그래픽 사용 GUI 인터페이스 제공
- 네트워킹
  - 네트워크 인지, 연결, 닫기, 데이터 송수신
- 보호 및 보안
  - 바이러스나 웜, 멀웨어(malware), 해킹 등의 외부 공격이나 무단 침입으로부터 보호

운영체제의 구성 요소와 커널



운영체제 구성(1/2)

- 운영체제 = 커널 + 툴 + 디바이스 드라이버
  - 커널(kernel)
    - 운영체제의 핵심 부분, 좁은 의미의 운영체제
    - 부팅 후 메모리에 상주하는 코드와 데이터
    - 운영체제의 핵심 기능 모두 구현
    - CPU, Memory, MMU(memory management unit) 등 컴퓨터 자원을 직접 제어하고 관리하는 코드와 자료 구조들
      - MMU: CPU가 메모리에 접근하는 것을 관리하는 컴퓨터 하드웨어 부품
    - 커널 코드는 함수들의 집합
    - 커널 기능을 이용하려면 응용프로그램은 반드시 시스템 호출 사용

운영체제 구성 (2/2)

- 운영체제 = 커널 + 툴 + 디바이스 드라이버
  - 도구(tool) 소프트웨어와 GUI
    - 사용자가 컴퓨터를 편리하게 사용할 수 있도록 제공하는 툴 소프트웨어 혹은 툴 응용프로그램
    - Windows 경우, 바탕화면 GUI, 탐색기, 명령창, 작업 관리자, 제어판 등
    - 리눅스 경우, 쉘, 로그인 프로그램
  - 디바이스 드라이버(device driver)
    - 장치를 직접 제어하고 입출력하는 소프트웨어
    - 장치마다 전담 디바이스 드라이버 있음
    - 일반적으로 장치 제작자에 의해 작성되어 배포됨
    - 사례
      - 키보드 드라이버, 디스크 드라이버, SCSI 드라이버, 마우스 드라이버, 그래픽 드라이버, 네트워크 드라이버, usb 드라이버 등

운영체제 커널 인터페이스 : 시스템 호출과 인터럽트

- 커널이 제공하는 2개 인터페이스 : 시스템 호출과 인터럽트
  - 응용프로그램과 하드웨어 사이의 중계 역할

- **시스템 호출**(system call)
  - 커널과 응용프로그램 사이의 인터페이스
  - 응용프로그램에서 커널 기능을 사용할 수 있는 유일한 방법
  - 시스템 호출 라이브러리를 통해 다양한 시스템 호출 함수 제공
    - 시스템 호출 라이브러리는 운영체제 패키지에 포함됨
    - 예) 파일 읽기, 메모리 할당, 프로세스 정보 보기, 프로세스 생성 등
      - open(), close(), read(), write(), fork(), exit(), wait() 등의 시스템 호출 함수

표준 라이브러리(standard library)란?
- 응용프로그램을 쉽게 작성할 수 있도록 복잡한 기능이 미리 작성된 코드
  - 예) printf(), abs(), strcmp() 등
- 운영체제 커널의 기능과 무관한 작업
- 응용프로그램은 표준 라이브러리 함수를 함수 호출(function call) 방법으로 사용

운영체제 커널 인터페이스 - 인터럽트

- **인터럽트(interrupt)**
  - 커널과 하드웨어 장치 사이의 인터페이스
  - 장치들이 입출력 완료, 타이머 완료 등을 CPU에게 알리는 하드웨어적 방법
    - 인터럽트를 알리는 하드웨어 신호가 직접 CPU에 전달
  - 인터럽트가 발생하면
    - CPU는 하는 일을 중단하고 인터럽트 서비스 루틴 실행
    - 인터럽트 서비스 루틴은 대부분 디바이스 드라이버 내에 있음
      - 예) 키를 입력하면 커널의 키보드 인터럽트 서비스 루틴 실행, 키를 읽어 커널 버퍼에 저장
    - 인터럽트 서비스 루틴은 커널 영역에 적재
    - 인터럽트 서비스 루틴의 실행을 마치면 하던 작업 계속
  - 인터럽트 활용
    - 운영체제가 장치에게 지시한 입출력 작업의 완료, 예고 없는 네트워크 데이터의 도착, 키보드나 마우스의 입력, 부족한 배터리 경고 등 장치와 관련된 모든 이벤트 처리