스마트시스템 운영체제
(LD01600)

김준철

정보시스템공학과

greensday@sungshin.ac.kr

2주차 강의

	주차	강의 목차
9.2	1	과목소개 / 운영체제 개요
9.9	2	컴퓨터 시스템 구조
9.16	3	프로세스와 스레드1
9.22	4	프로세스와 스레드2, CPU스케쥴링1
휴강(9.30) (추석)		
10.7	5	CPU스케쥴링2
10.14	6	프로세스 동기화
10.21	7	교착 상태
10.28	8	중간고사
11.4	9	물리 메모리 관리
11.11	10	가상메모리 기초
11.18	11	가상메모리 관리
11.25	12	입출력시스템1
12.2	13	입출력시스템2, 파일시스템1
12.9	14	파일시스템2
12.16	15	기말고사

Operating Systems

ch.02 컴퓨터 구조

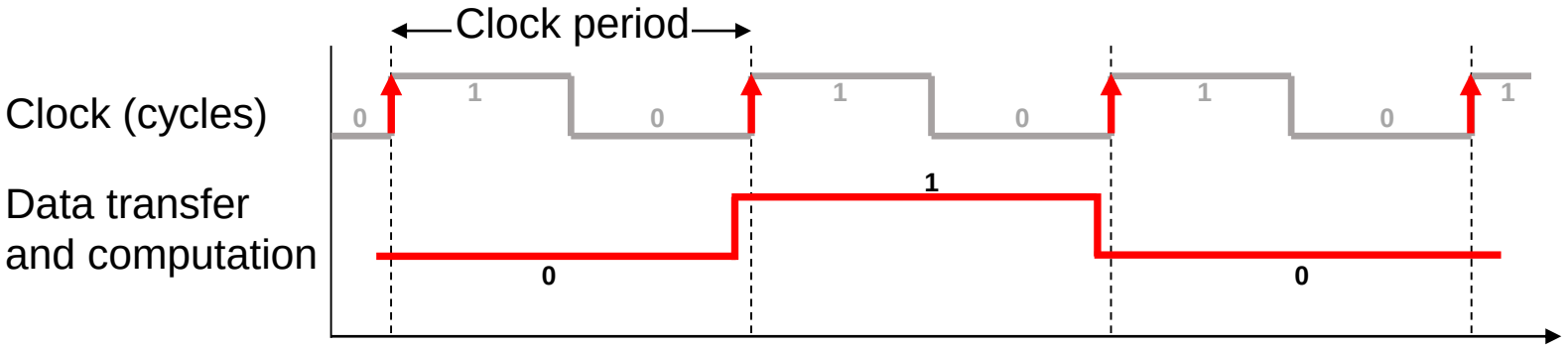
Basic Definitions

- SI(International System of Units) 단위와 IEC(International Electotechnical Commission) 단위

		SI(10진 단위)	IEC(2진 단위)	
기호	이름	값	값	
p	pico-	$(10^3)^{-4}=10^{-12}$		
n	nano-	$(10^3)^{-3}=10^{-9}$		
μ	micro-	$(10^3)^{-2}=10^{-6}$		
m	mili-	$(10^3)^{-1}=10^{-3}$		
k, K	kilo-	$(10^3)^{1}=10^{3}$	$(2^{10})^{1}=2^{10}$	Kibi
M	mega-	$(10^3)^{2}=10^{6}$	$(2^{10})^{2}=2^{20}$	Mebi
G	giga-	$(10^3)^{3}=10^{9}$	$(2^{10})^{3}=2^{30}$	Gibi
T	tera-	$(10^3)^{4}=10^{12}$	$(2^{10})^{4}=2^{40}$	Tebi

Clocking

- Digital hardware의 동작은 (constant-rate) **clock**에 지배됨



- Clock period (주기, 시간): duration of a clock cycle
  - 1000ps (= 1ns) = 1000×10^{-12}s = 1.0×10^{-9}s
  - 250ps (= 0.25ns) = 250×10^{-12}s = 0.25×10^{-9}s
- Clock frequency (주파수): cycles per second [**or** clock rate]
  - 1GHz (= 1000MHz) = 1.0×10^{9}Hz
  - 4.0GHz (= 4000MHz) = 4.0×10^{9}Hz

컴퓨터 하드웨어의 구성

- **컴퓨터의 구성**
  - 하드웨어는 **필수 장치**와 **주변장치**로 구성되고 **시스템 버스**로 연결 됨
    - 필수장치 : CPU, Main memory
    - 주변장치 : 입력장치, 출력장치, 저장장치
- **폰노이만 구조**
  - CPU, Main memory,입출력 장치의 전형적인 3단계 구조로 이루어진 프로그램 내장형 컴퓨터 구조

[그림] 컴퓨터를 구성하는 장치

소스코드의 번역과 실행

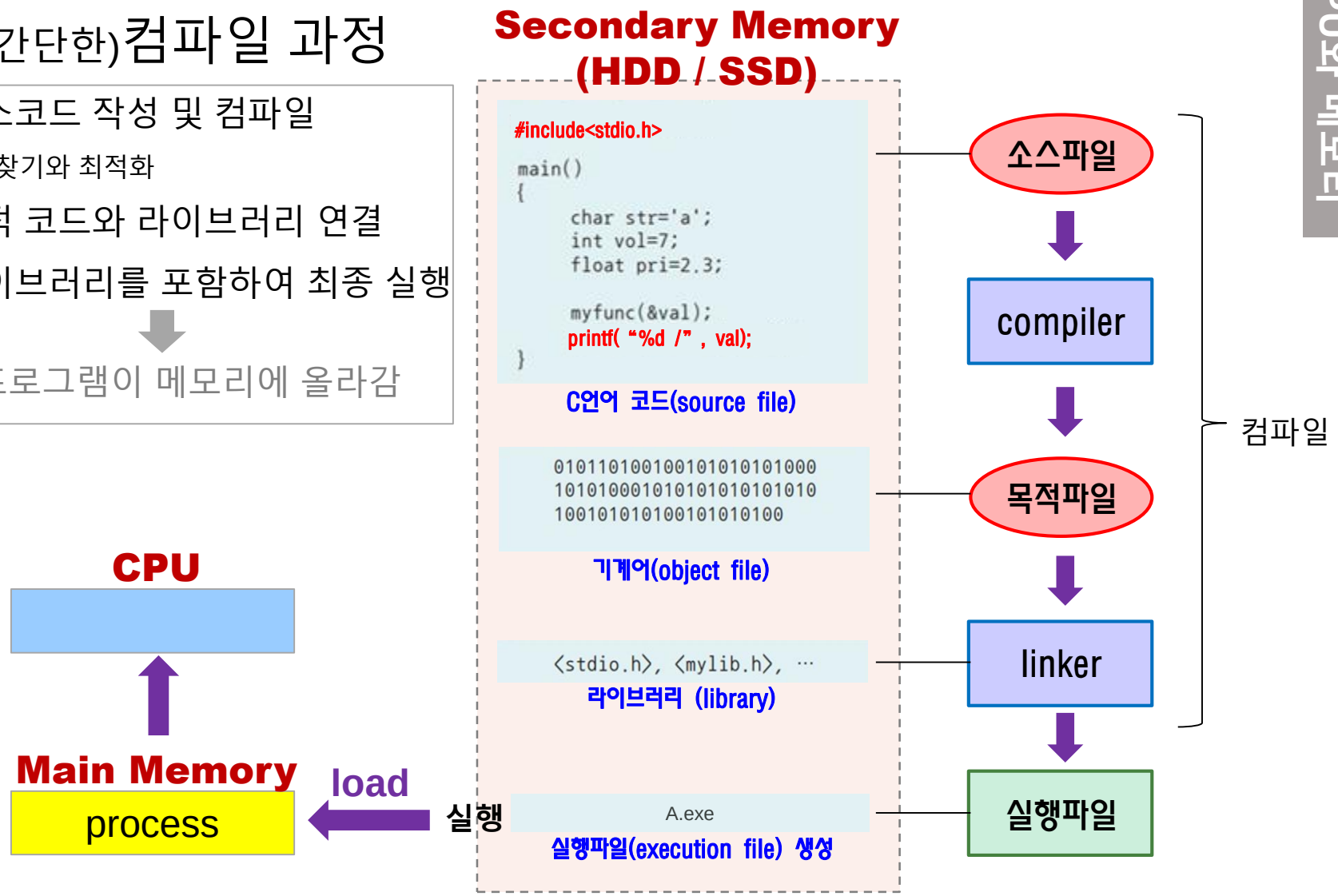
- (간단한)컴파일 과정

❶소스코드 작성 및 컴파일
- 에러찾기와 최적화

❷목적 코드와 라이브러리 연결

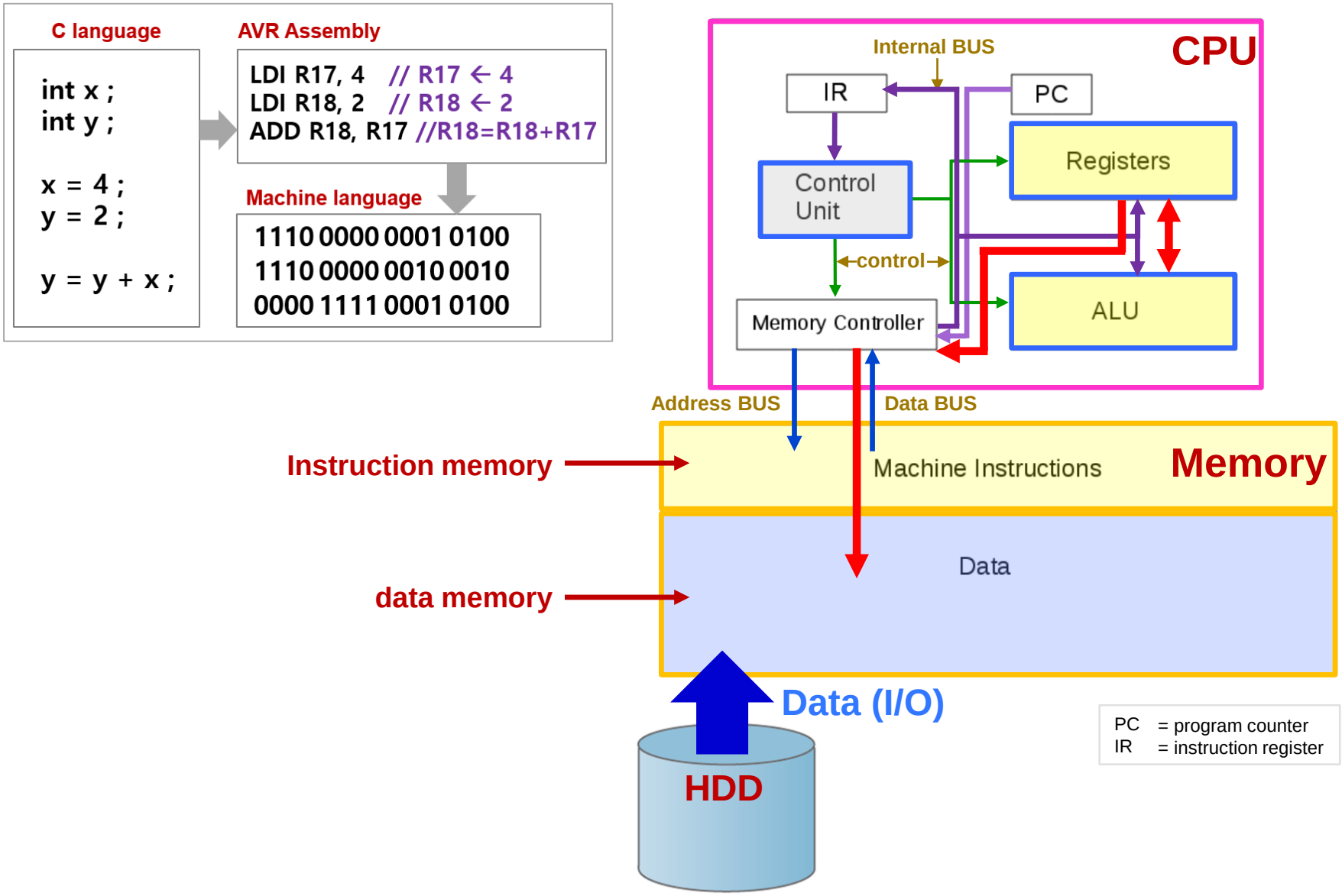
❸라이브러리를 포함하여 최종 실행

프로그램이 메모리에 올라감

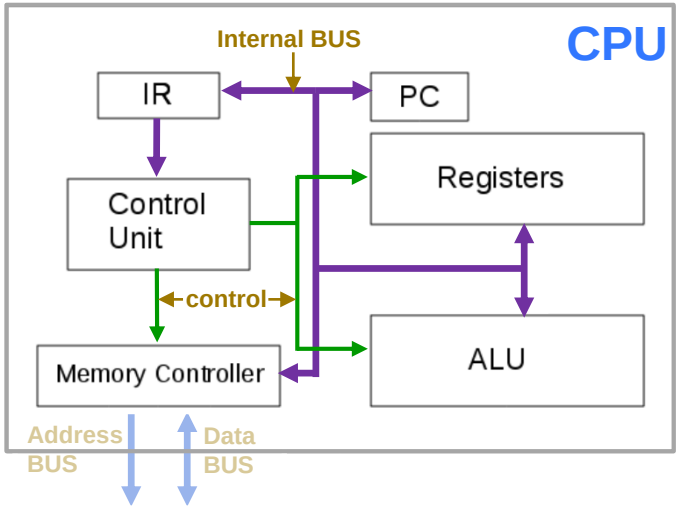


[그림] 컴파일 과정

Instruction 처리



프로세서 내부 구성



- **ALU , data path** : data 처리(**연산**)에 관계함
  arithmetic / logic unit

- **Control**
  : instruction(명령어) **해독**(decode), **제어 신호**발생
  CPU 내부 resource를 관리

- **Register** : very fast and expensive(relative to memory)
  - general purpose register (GPR) : 프로그램, 데이터 처리 등 일반적인 작업
  - special purpose register : 지정된 특별한 용도로 사용 (PC, IR, SP ...)
    stack pointer
  - 기타 OS / assemble 등에 예약된 register들도 있음
  - 피연산자(operand) 및 연산결과의 임시저장
  - instruction을 표현하는 bit들을 저장

- **BUS** : system, component 간의 data 전달 역할
  (data bus, Control bus, address bus)

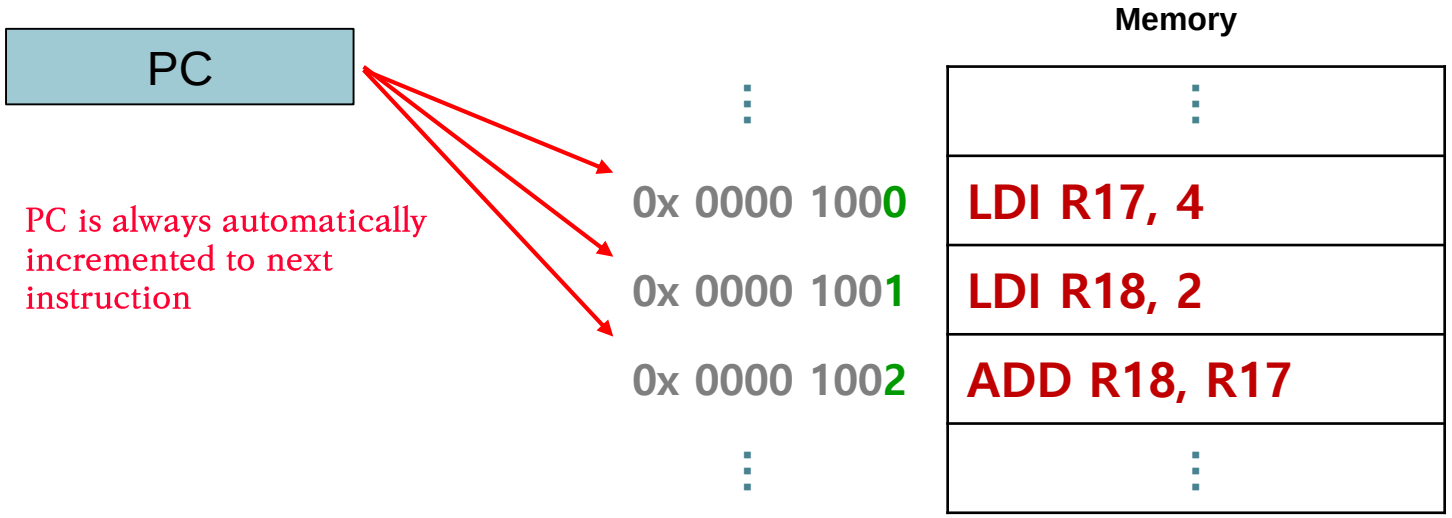
PC (program counter)

- **PC(program counter)**
  - 현재 수행하고 있는 instruction의 memory 주소를 저장
  - Next sequential instruction의 주소 계산

    * PC는 항상 자동적으로 1word 증가하여 다음 실행할 instruction의 주소를 가리킴

- **Example**

```
int x = 4 ;     // LDI R17, 4
int y = 2 ;     // LDI R18, 2
y = y + x ;     // ADD R18, R17
```



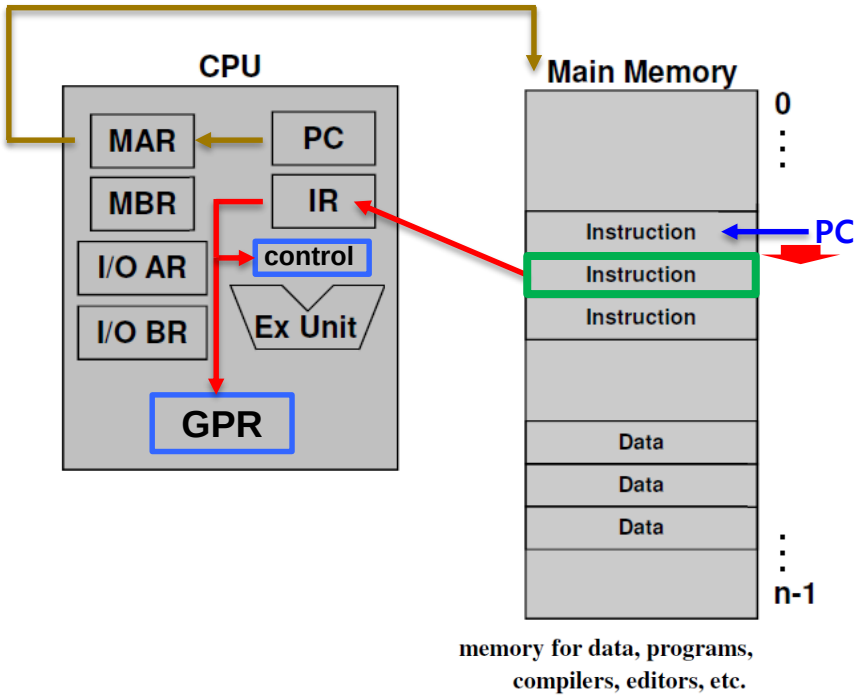
Register Operands(피연산자)

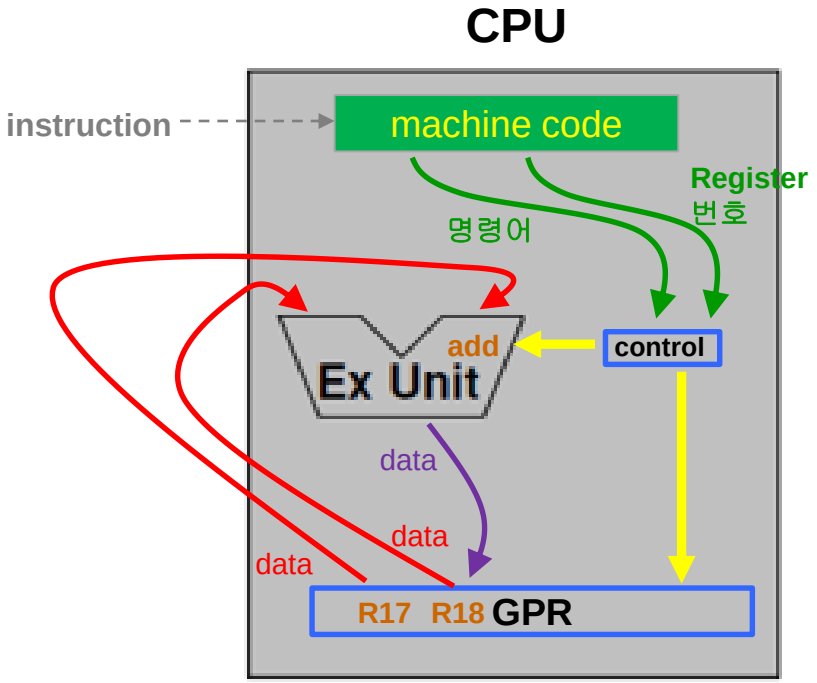
Memory의 operand를 CPU에서 바로 연산하는 것은 불가능
→operand를 register에 저장하고 연산

EX

```
⋮
ADD R18, R17
⋮
```

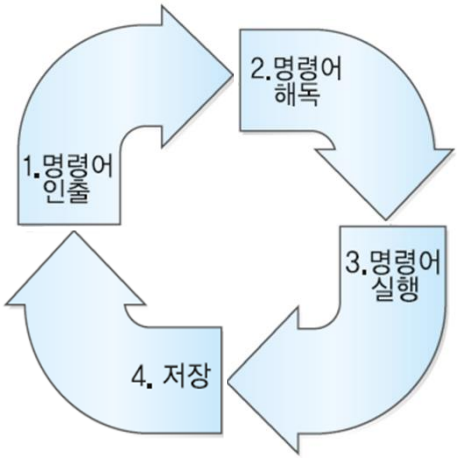
Fetch / Decode / Execution



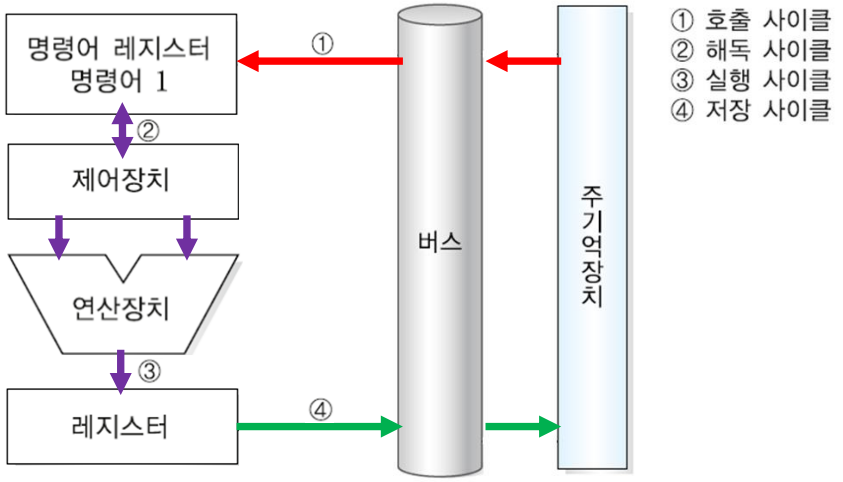


명령어 사이클

- **명령어 사이클 (Instruction Cycle)**
  - 프로그램을 구성하는 명령어는 4단계의 과정을 통해서 수행
- **각 단계별 사이클의 역할**
  - **인출(Fetch)** 사이클 : 필요한 명령어를 주기억장치에서 불러오는 사이클
  - **해독(Decode)** 사이클 : 호출된 명령어를 해석하는 사이클
  - **실행(Execute)** 사이클 : 해석된 명령어를 산술논리연산장치를 통하여서 실행
  - **저장(Store)** 사이클 : 수행결과를 주기억장치에 저장하는 사이클



명령어 사이클



명령어 수행과정

메모리의 종류

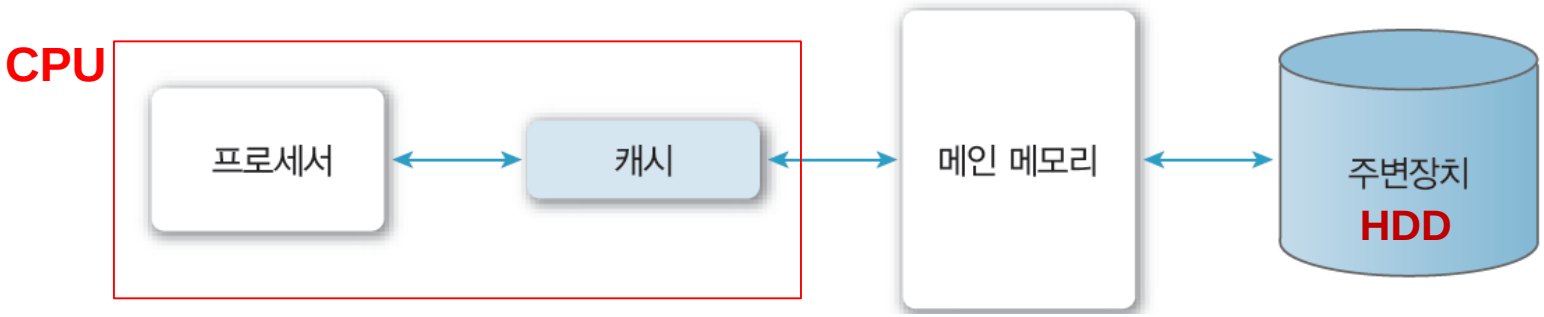
- **휘발성 메모리**
  - DRAM(Dynamic RAM)
    - 저장된 0과 1의 데이터가 일정 시간이 지나면 사라짐

      → 일정 시간마다 다시 refresh
  - SRAM(Static RAM)
    - 전력이 공급되는 동안에는 데이터를 보관할 수 있어서 refresh가 필요가 없음

- **비휘발성 메모리**
  - 플래시 메모리(flash memory)
    - 디지털카메라, MP3 플레이어, USB 드라이버같이 전력이 없어도 데이터를 보관하는 저장장치
  - SSD
    - 가격이 비싸지만 빠른 데이터 접근 속도, 저전력, 내구성 때문에 많이 사용
    - Hard Disk Drive(HDD)를 대체하고 있음

저장장치의 계층 구조

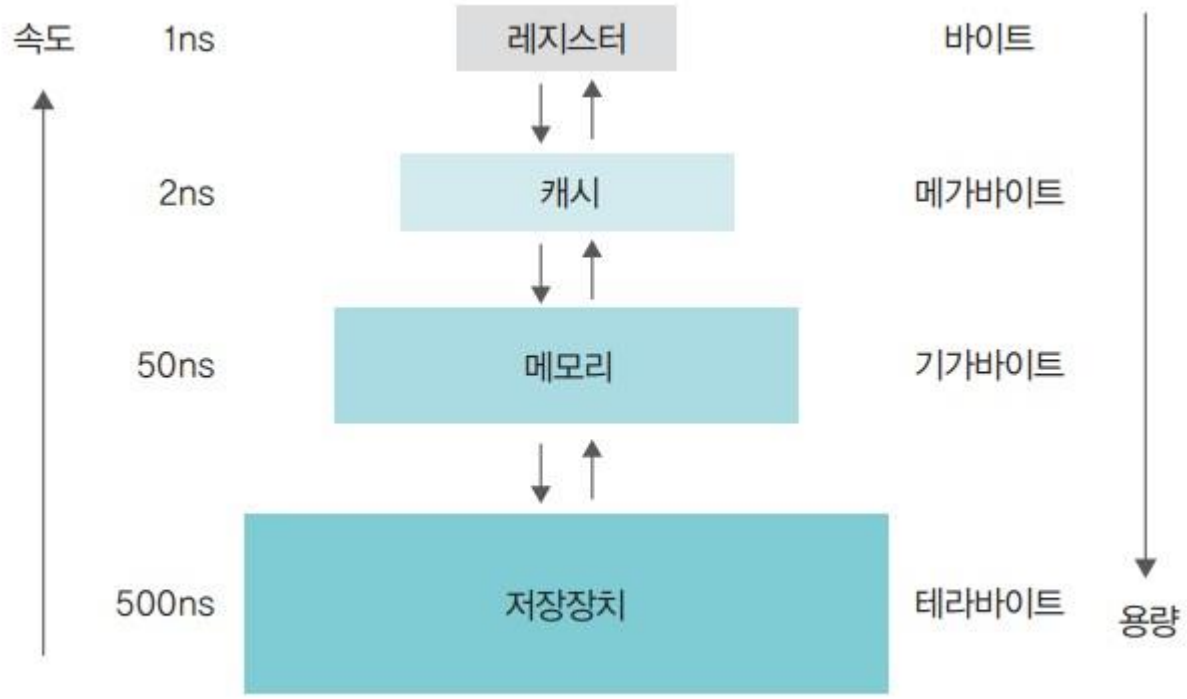
- **레지스터**
  - 프로세서 내부에 있으며, 프로세서가 사용할 데이터를 보관하는 가장 빠른 메모리
- **캐시(cache)**
  - 프로세서와 메인 메모리 간에 속도 차이의 부담을 줄이려고 프로세서 내부나 외부에 캐시를 구현하기도 함
  - SRAM(Static RAM) 사용 (고가 메모리)
- **메인 메모리**
  - 프로세서 외부에 있으면서 프로세서에서 수행할 프로그램과 데이터를 저장하거나 프로세서에서 처리한 결과 저장
  - DRAM(Dynamic RAM) 사용 (저장 밀도가 높고 저가격)



[그림] 메인 메모리의 역할

저장장치의 계층 구조

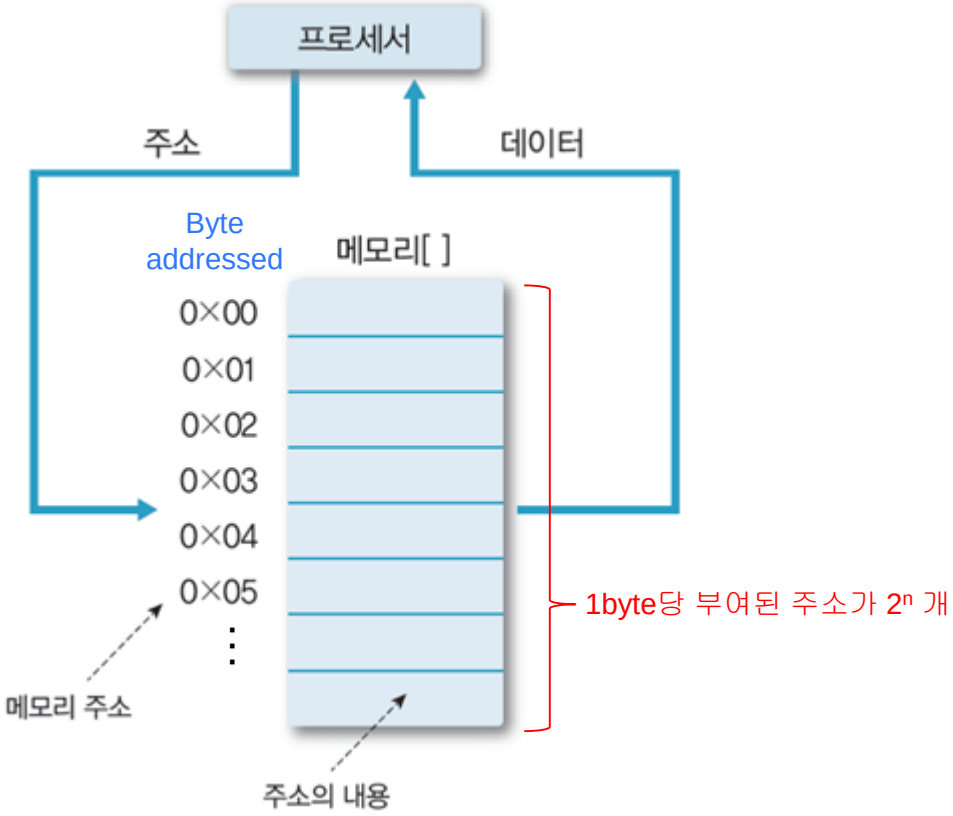
- **저장장치의 계층 구조**



[그림] 저장장치의 계층 구조

메인 메모리 및 주소

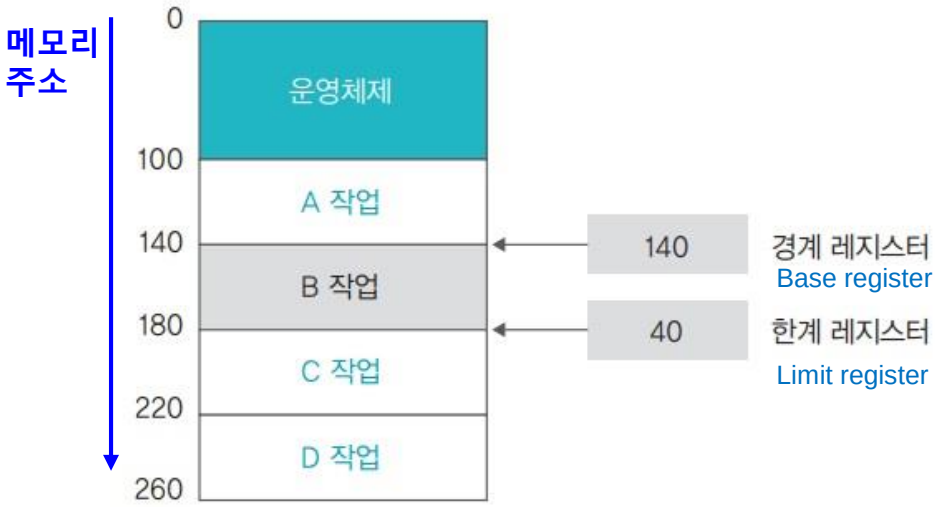
- **메인 메모리**
  - 메모리에는 byte 단위로 주소가 부여 되어 있음



[그림] 메인 메모리의 주소 지정

메모리 보호

- **메모리 보호의 필요성**
  - 현대의 운영체제는 **시분할 기법**을 사용하여 여러 프로그램을 동시에 실행하므로 **사용자 영역이 여러 개의 작업 공간으로 나뉘어 있음**
  - 메모리가 보호되지 않으면 어떤 작업이 **다른 작업의 영역을 침범**하여 프로그램을 파괴하거나 데이터를 지울 수도 있으며, 최악의 경우 운영체제 영역을 침범하면 시스템이 멈출 수도 있음



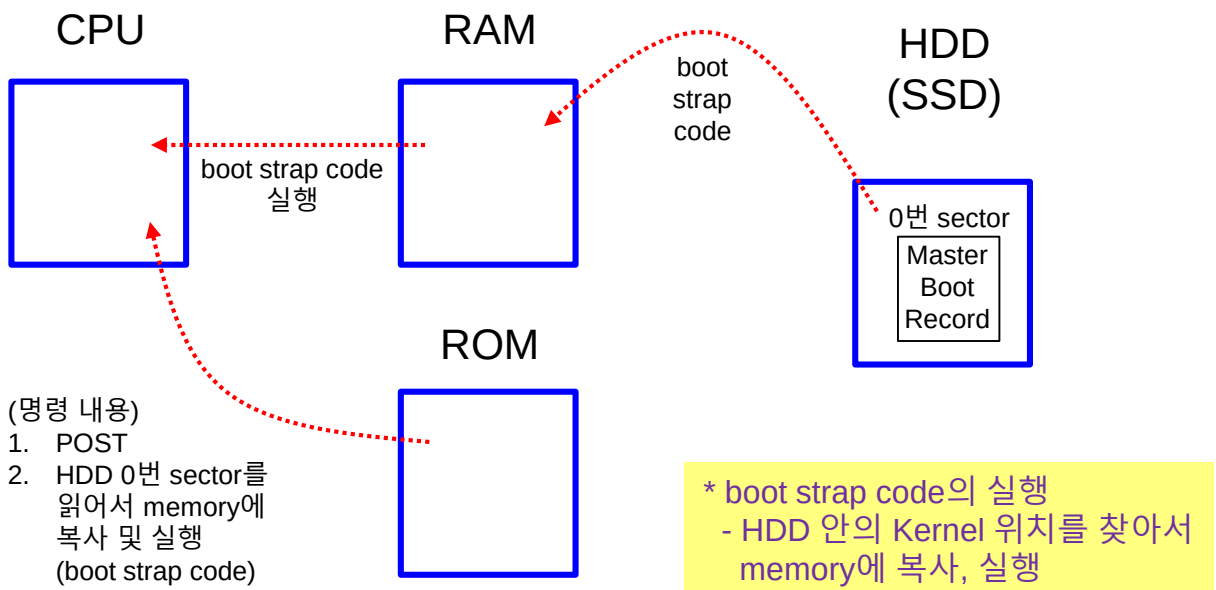
[그림] 메모리의 보호

두 레지스터의 값을 벗어 나면 메모리 오류와 관련된 인터럽트가 발생

메모리 영역을 벗어나서 발생한 인터럽트의 경우 운영체제가 해당 프로그램을 강제 종료

부팅

- **부팅**
  - 컴퓨터를 켰을 때 운영체제를 메모리에 올리는 과정



[그림] 부팅 과정

이중 모드(dual mode)

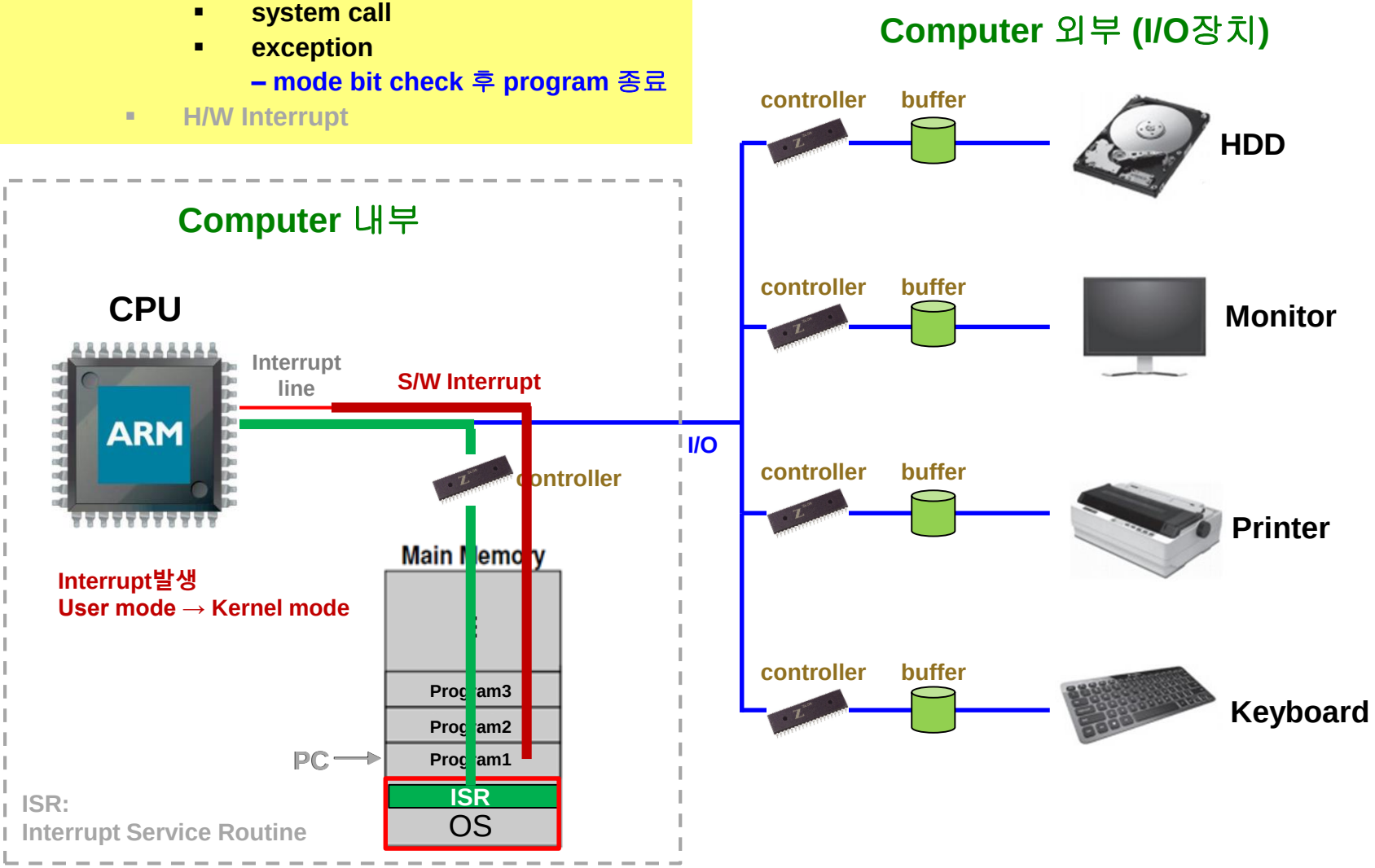
- **이중 모드(dual mode) – CPU는 두 가지 모드로 동작함**
  - 운영체제가 **커널 모드**와 **사용자 모드**를 전환하며 일 처리를 하는 것
    Kernel mode User mode
  - 궁극적인 목적은 자원 보호에 있음
- 커널 모드(Kernel mode)
  - 운영체제와 관련된 커널 프로세스가 실행되는 상태
  - Mode bit = 0
- 사용자 모드(User mode)
  - 사용자 프로세스가 실행되는 상태
  - Mode bit = 1

이중모드의 구현 – flag register 사용

OS가 있는 시스템에서의 Interrupt

μP에게 즉시 특정 작업을 처리하도록 요구하는 (비정상적)사건

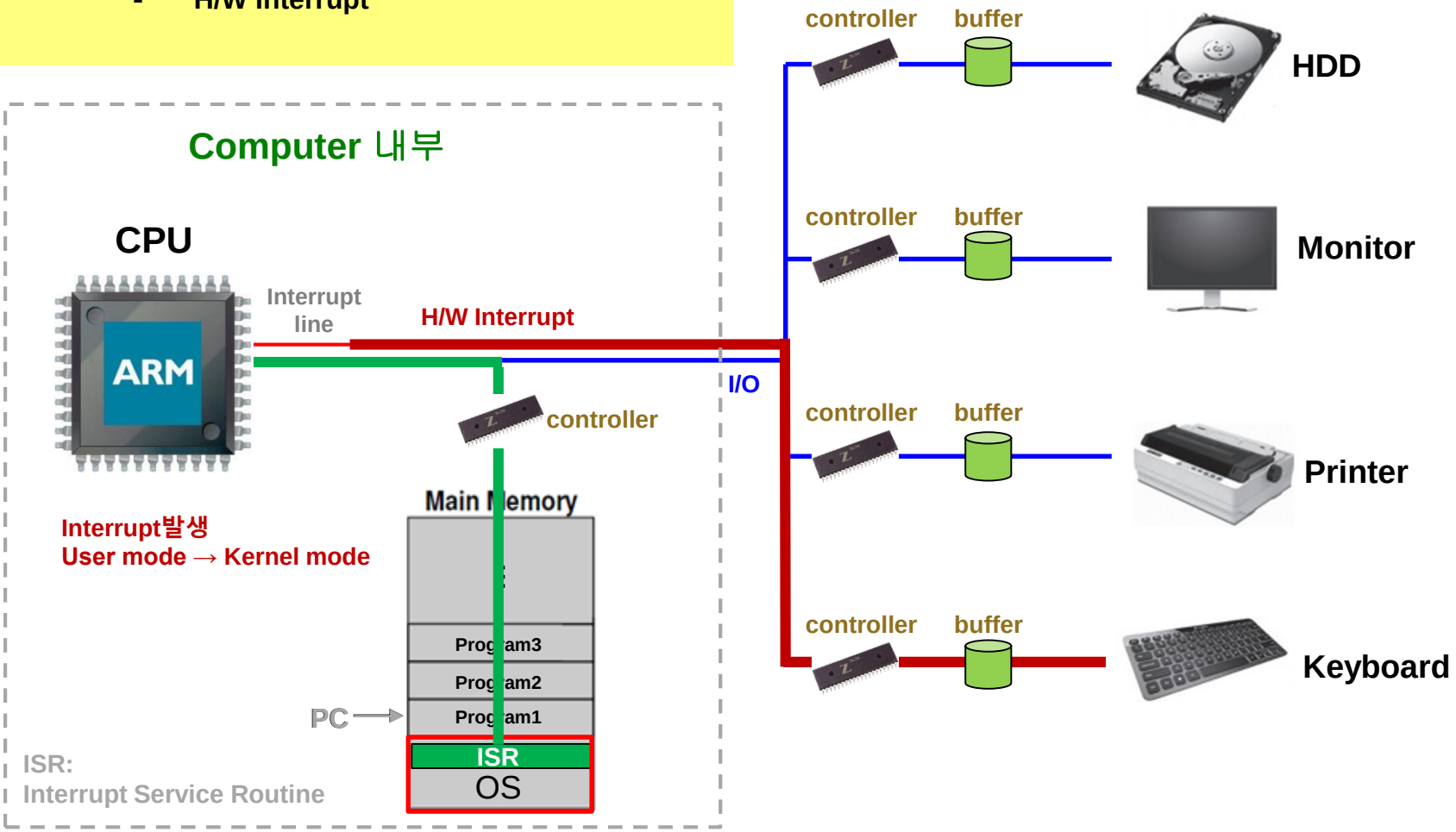
- **Interrupt : CPU 사용 권한이 OS로 넘어감**
  - **S/W Interrupt (=Trap)**
    - **system call**
    - **exception**
      - **– mode bit check 후 program 종료**
  - **H/W Interrupt**



OS가 있는 시스템에서의 Interrupt

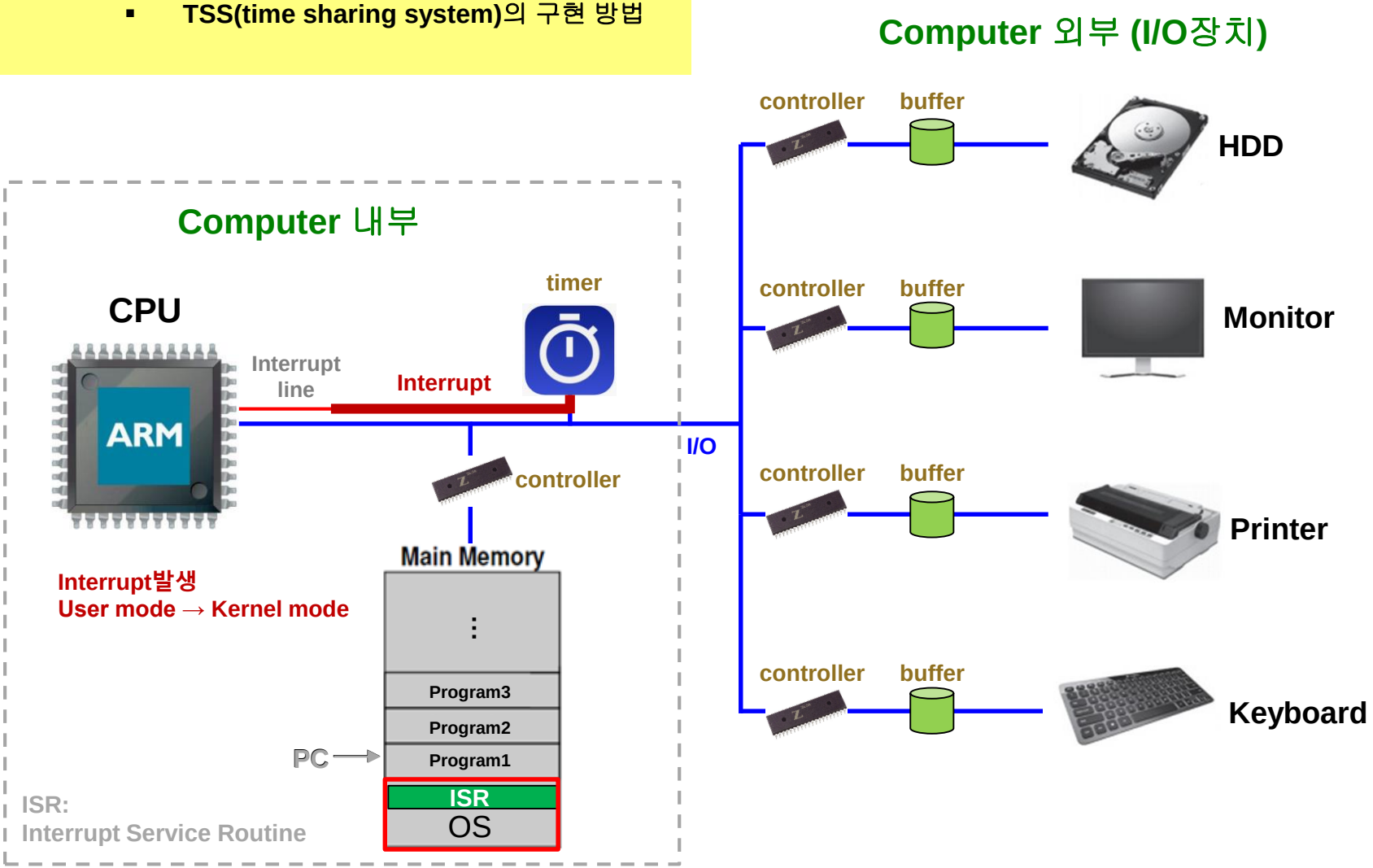
μP에게 즉시 특정 작업을 처리하도록 요구하는 (비정상적)사건

- **Interrupt : CPU 사용 권한이 OS로 넘어감**
  - S/W Interrupt (=Trap)
    - system call
    - exception
  - **H/W Interrupt**



타이머 인터럽트

- **Timer Interrupt**
  - 특정 program의 CPU 독점 방지
  - TSS(time sharing system)의 구현 방법



Polling 방식과 Interrupt 방식

- **Polling 방식 : 우선 순위가 없는 순차적 처리**
  - 입출력을 요청하면 주기적으로 입출력장치를 직접 확인해서 처리하는 방식

- **Interrupt 방식 : 우선 순위가 높은 코드를 먼저 처리**
  - Interrupt :
    - μP에게 즉시 특정 작업을 처리하도록 요구하는 (비정상적)사건
      micro processor, CPU
    - 정상적인 실행 순서를 바꿀 만큼 중요한 사건
    - 우선 처리를 위한 사건을 정의하고, 우선 순위에 따라 실행 순서를 변경
    - 하드웨어 vs. 소프트웨어 Interrupt
      - HW Interrupt : 하드웨어에 의해 발생, μC의 Interrupt
        micro controller
      - SW Interrupt : 운영체제의 커널을 통해 발생

Polling 방식과 Interrupt 방식 차이 예

```
void main(void) {
      while(1) {
         if(button_pressed)                    // 소프트웨어에 의해 사건(event) 검사
            do_something();
      }

      return 0;
}
```

Polling 방식

```
ISR(BUTTON_PRESS_vect) {
      do_something();
}

void main(void) {
      setup_button_press_interrupt();     // 하드웨어에 의해 사건(event) 검사 설정
      while(1) {
      }

      return 0;
}
```

Interrupt 방식

감사합니다.