Computer Science Study

Operating System

Subject 1. System Call
Subject 2. Interrupt
Subject 3. I/O

made by Seungtaek

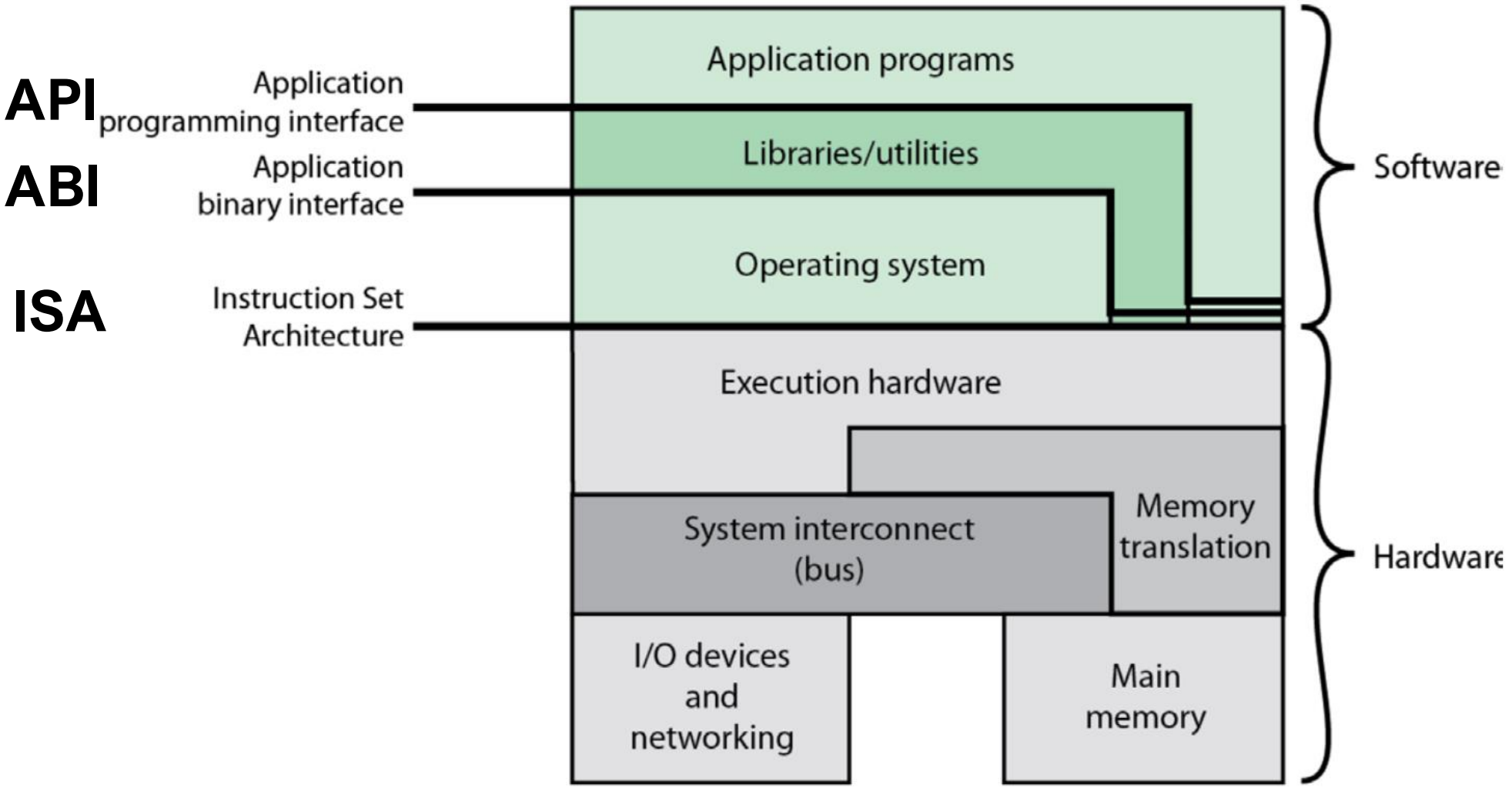
홍승택

2024. 09. 09. 19시

subject 1.

System Call

Hardware and Software Infrastructure



ISA (Instruction Set Architecture)

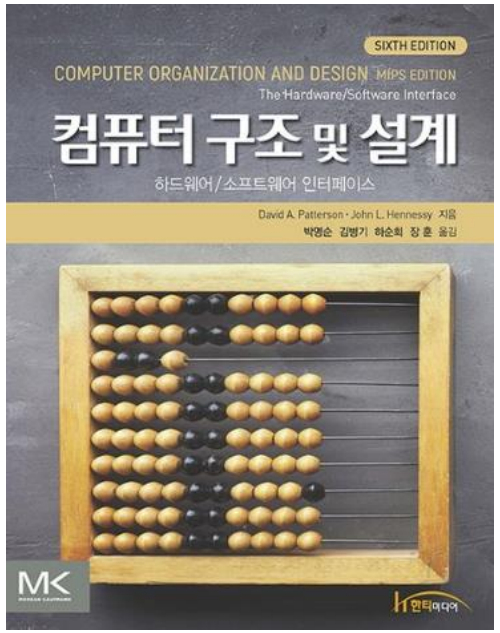
IA32(x86), x86-64(AMD64), IA64, ARM, MIPS, …

MIPS32 Add Immediate Instruction

001000	00001	00010	0000000101011110
OP Code	Addr 1	Addr 2	Immediate value

Equivalent mnemonic: **addi $r1, $r2, 350**

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7



David A. Patterson , John L. Hennessy

Microarchitecture

Pipelining, caches, branch prediction, buffers, …

x86, ARM

```
void add(){
    int a, b, c;
    a = 2;
    b = 2;
    c = a + b;
}
```

x86

Compile

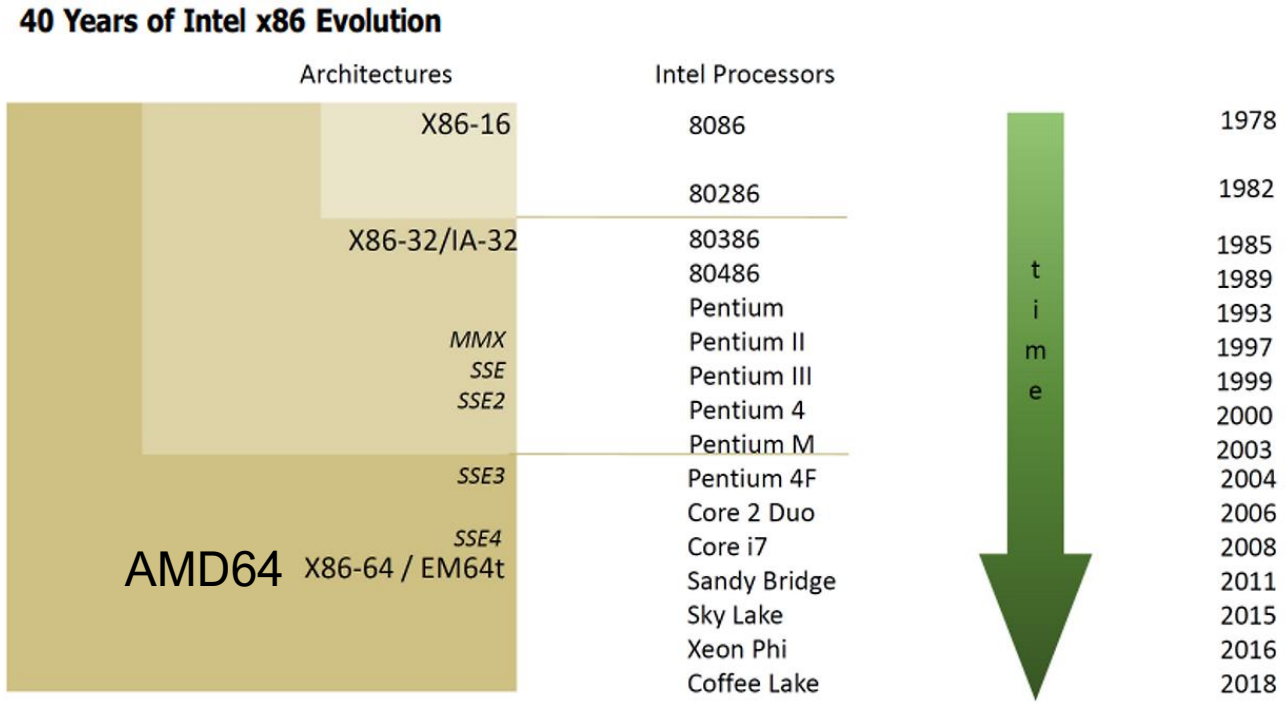
```
push    rbp
mov     rbp, rsp
mov     dword ptr [rbp - 4], 2
mov     dword ptr [rbp - 8], 2
mov     eax, dword ptr [rbp - 4]
add     eax, dword ptr [rbp - 8]
mov     dword ptr [rbp - 12], eax
pop     rbp
ret
```

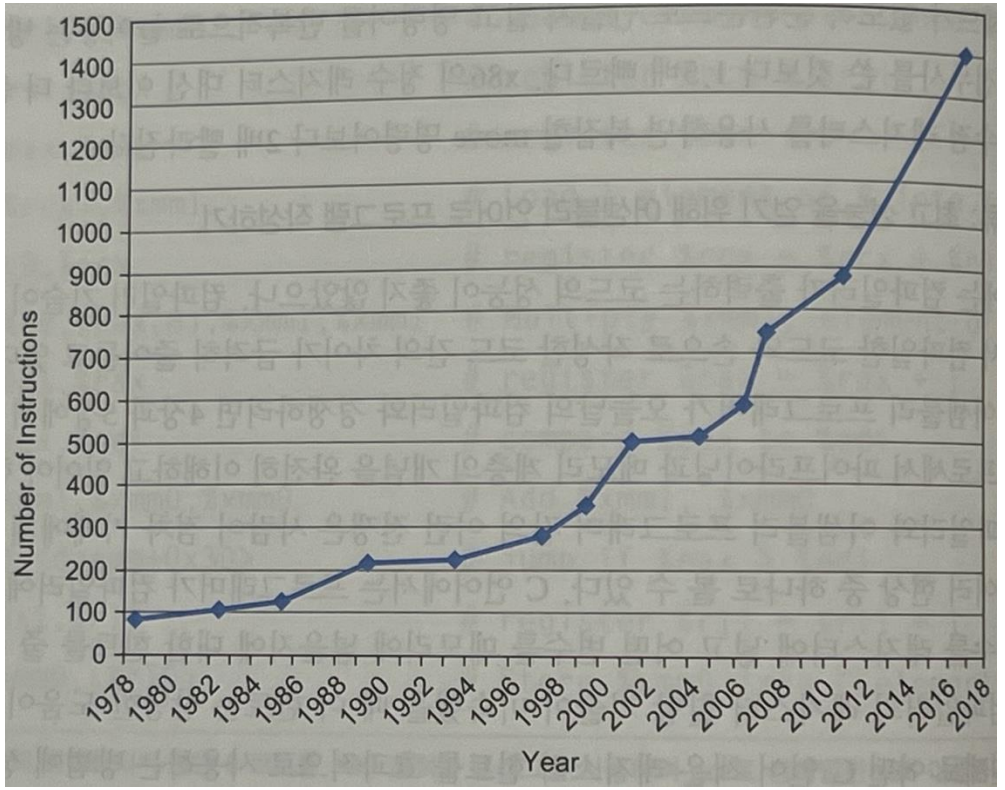
ARM

Compile

```
sub     sp, sp, #16
mov     w8, #2
str     w8, [sp, #12]
str     w8, [sp, #8]
ldr     w8, [sp, #12]
ldr     w9, [sp, #8]
add     w8, w8, w9
str     w8, [sp, #4]
add     sp, sp, #16
ret
```

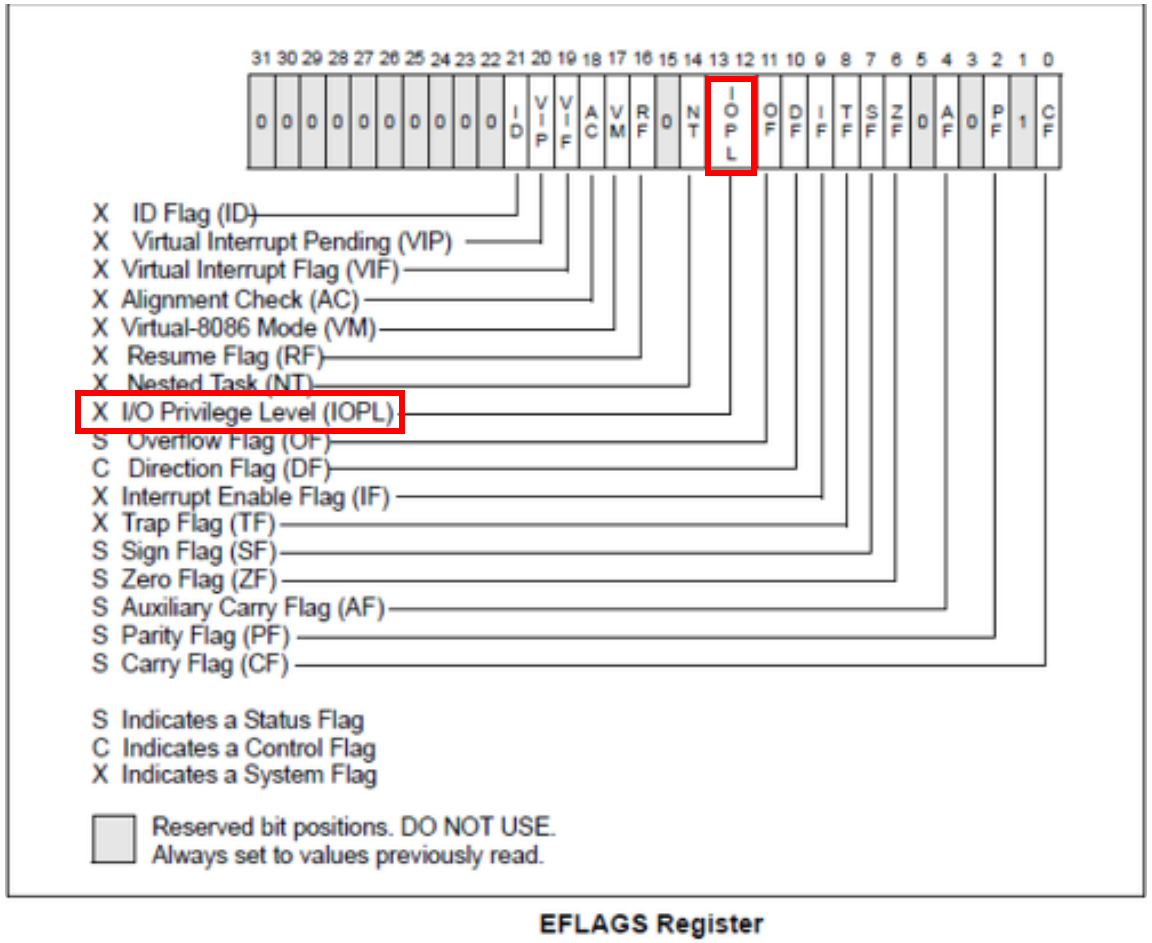
x86 Architecture



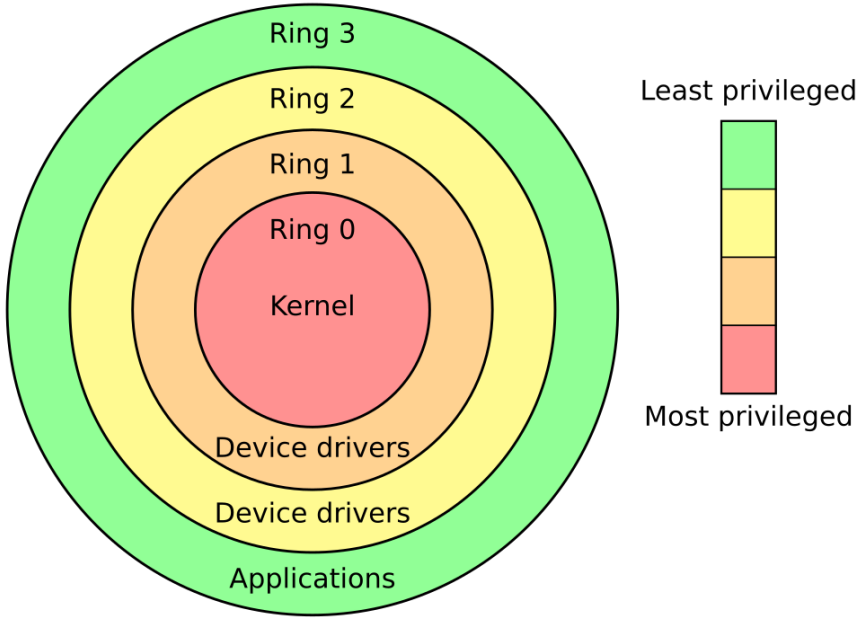


Privilege Level

Intel x86



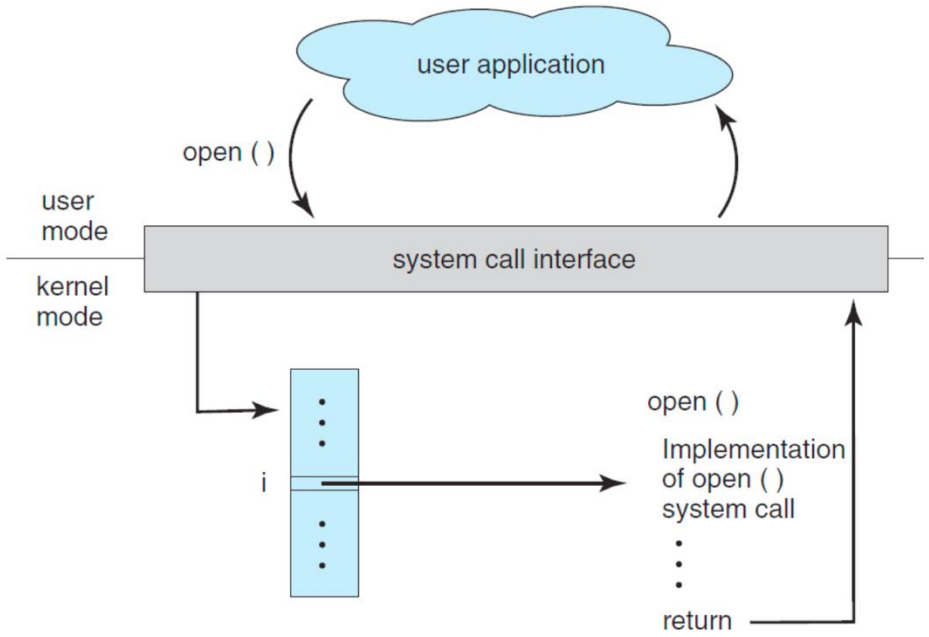
EFLAGS Register



system call, syscall

운영 체제의 커널이 제공하는 서비스에 대해,
응용 프로그램의 요청에 따라 커널에 접근하기 위한 인터페이스

프로세스 제어	– end, abort, load, execute, create, terminate – get/set process attribute	– wait for time, wait for event, wait for signal – allocate/free memory
파일 조작	–Create/delete file –Open/close	–Read, write, reposition –Get/set file attribute
주변장치 조작	–Request/release device –Read, write, reposition	–Get/set device attribute –Logically attach/detach
정보관리	–Get/set time or date –Get/set system data	– Get/set process, file, device attribute
통신	–create, delete connection –Send, receive message	–Transfer status info. –Attach/detach remote device
프로텍션(보호)	–user permission	–mode, mask, owner



open(2)

위치 : /fs/open.c
형태 : int open(const char *filename, int flags, umode_t mode)

인자 개수 3개

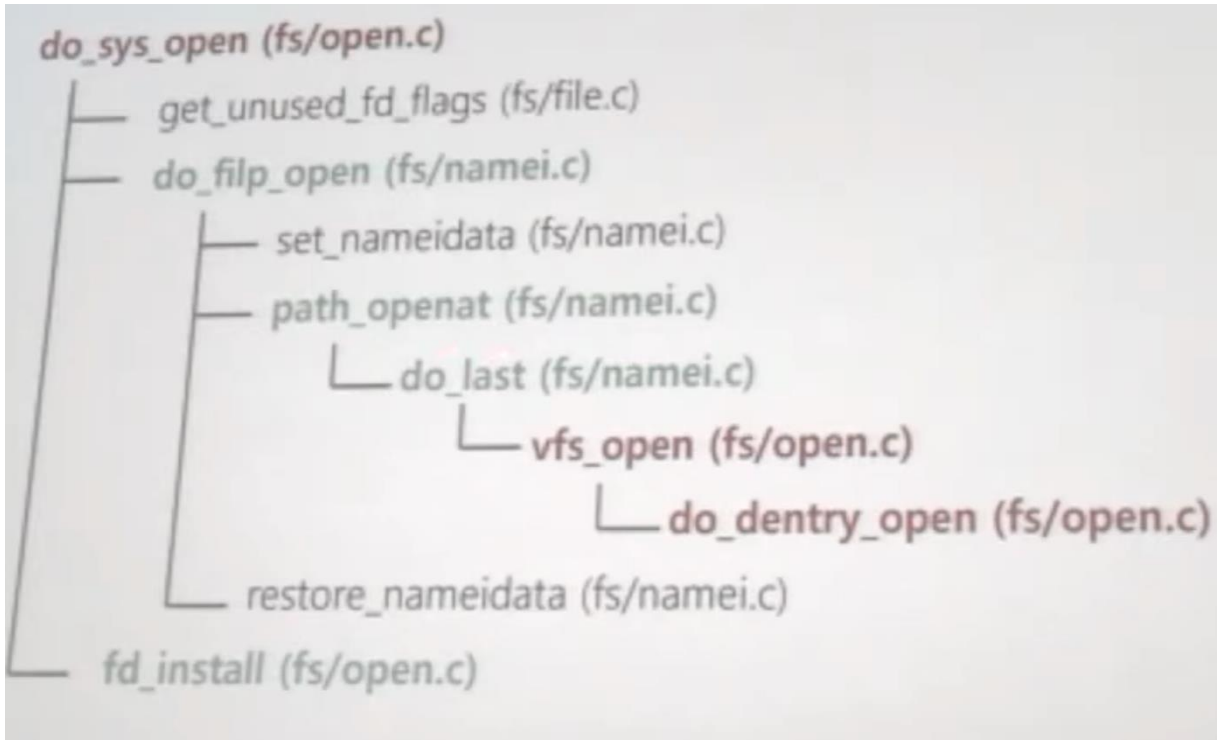
함수이름

```
SYSCALL_DEFINE3(open, const char __user *, filename, int, flags, umode_t, mode)
{
        if (force_o_largefile())
                flags |= O_LARGEFILE;

        return do_sys_open(AT_FDCWD, filename, flags, mode);
}
```

https://elixir.bootlin.com/linux/v4.4/source/fs/open.c#L1038

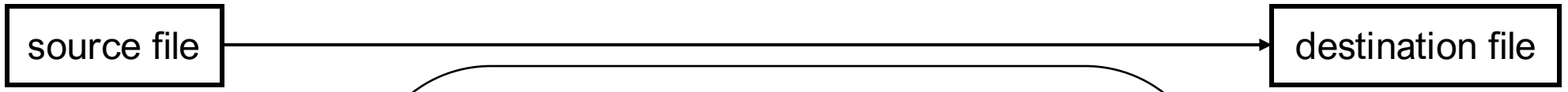
open(2)

function call chain




```c
long do_sys_open(int dfd, const char __user *filename, int flags, umode_t mode)
{
        struct open_flags op;
        int fd = build_open_flags(flags, mode, &op);
        struct filename *tmp;

        if (fd)
                return fd;

        tmp = getname(filename);
        if (IS_ERR(tmp))
                return PTR_ERR(tmp);

        fd = get_unused_fd_flags(flags);
        if (fd >= 0) {
                struct file *f = do_filp_open(dfd, tmp, &op);
                if (IS_ERR(f)) {
                        put_unused_fd(fd);
                        fd = PTR_ERR(f);
                } else {
                        fsnotify_open(f);
                        fd_install(fd, f);
                }
        }
        putname(tmp);
        return fd;
}
```

cp 명령어



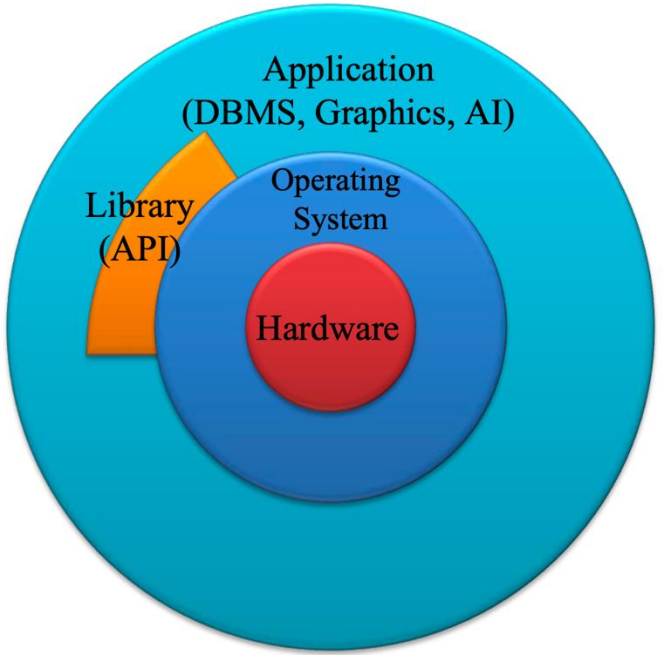
입력 파일 이름 획득
        화면에 프롬프트 출력
        입력 받아들임
출력 파일 이름 획득
        화면에 프롬프트 출력
        입력 받아들임
입력 파일 열기
        파일이 존재하지 않을 경우, 비정상 종료
출력 파일 생성
        파일이 존재할 경우, 비정상 종료
루프
        입력 파일로부터 읽어 들임
        출력 파일에 씀
읽기가 실패할 때까지
출력 파일 닫기
화면에 완료 메시지 출력
정상적으로 종료

API (Application Programming Interface)

Windows 시스템 : **Windows API**

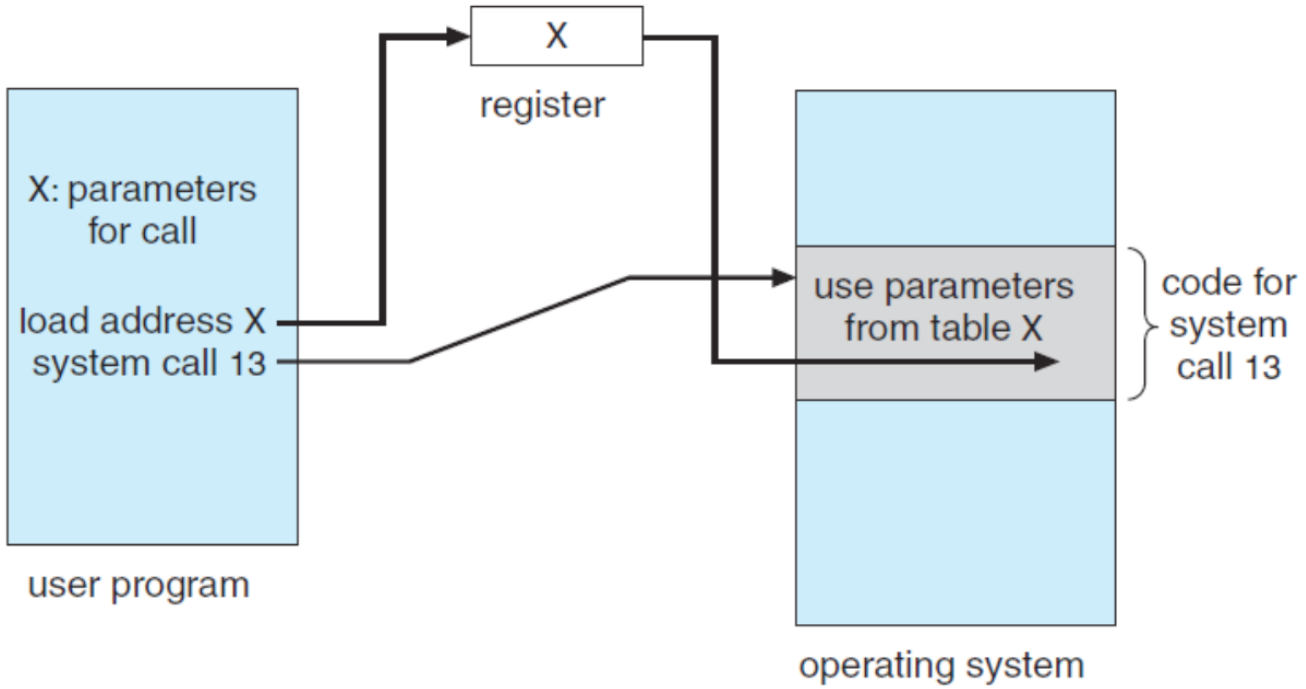
POSIX 기반 시스템(UNIX, Linux, MAC) : **POSIX API**

Java virtual machine : **Java API**



매개변수 전달

1. 매개변수를 레지스터에 **직접 전달**
2. 5개 초과 시에는 레지스터를 통한 **주소 전달**(블록, 스택)



example : linux/x86-64

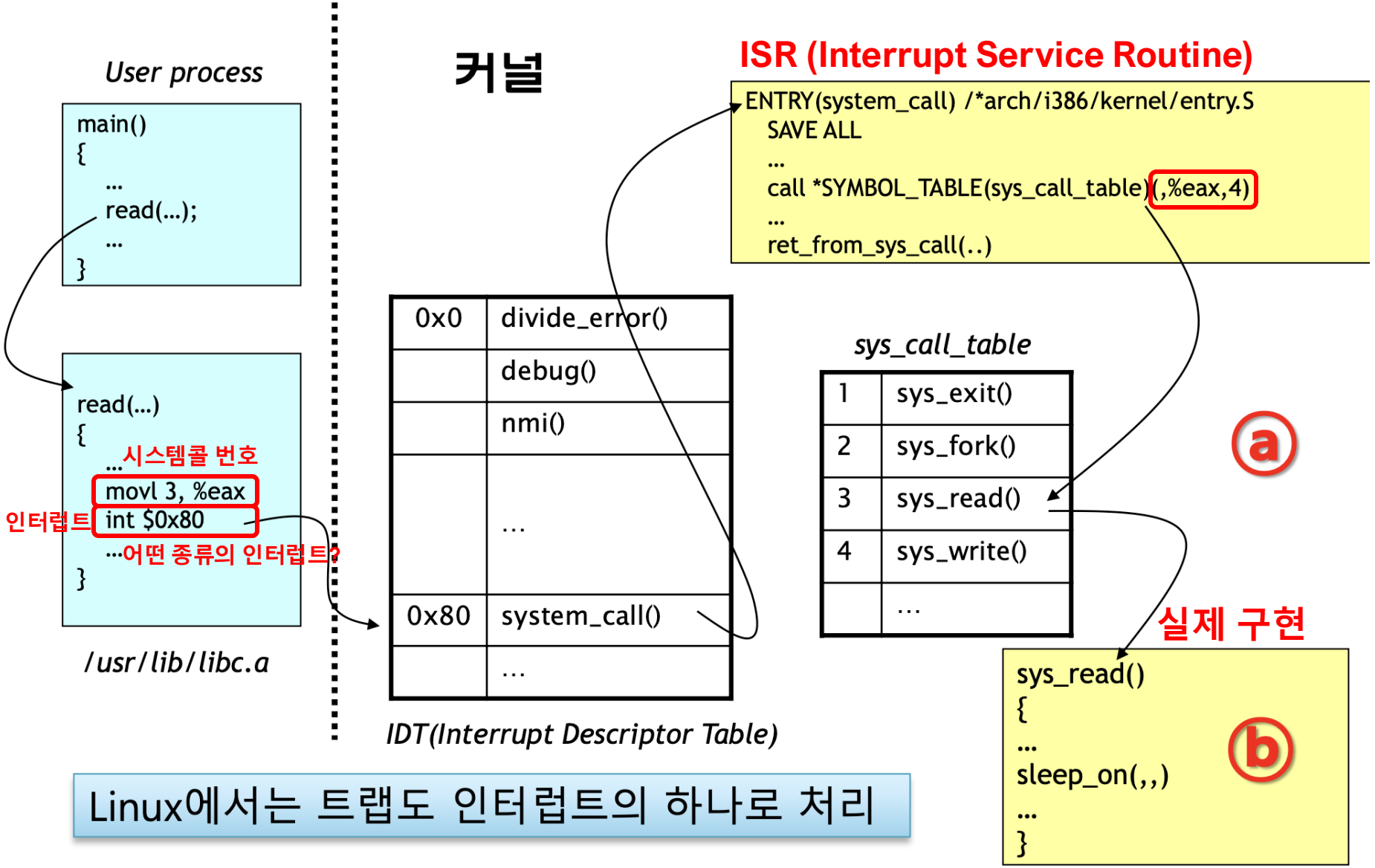
```
int main() {
        write(1, “hello, world\n”, 13);
        _exit(0);
}
```

x86_64 Linux Syscall Table

rax	System Call	rdi	rsi	rdx
0	sys_read	unsigned int fd	char* buf	size_t count
1	sys_write	unsigned int fd	const char* buf	size_t count
2	sys_open	const char* filename	int flags	int mode
59	sys_execve	const char* filename	const char* argv[]	const char* envp[]
60	sys_exit	int error_code		

```
.section .data
string:
        .ascii “hello, world\n”
string_end:
        .equ len, string_end – string
.section .text
.globl main
main
        movq $1, %rax
        movq $1, %rdi
        movq $string, %rsi
        movq $len, %rdx
        syscall

        movq $60, %rax
        movq $0, %rdi
        syscall
```

리눅스의 시스템 콜 처리 과정



Linux에서는 트랩도 인터럽트의 하나로 처리

x86 IVT (IDT)

IVT Offset	INT #	Description
0x0000	0x00	Divide by 0
0x0004	0x01	Reserved
0x0008	0x02	NMI Interrupt
0x000C	0x03	Breakpoint (INT3)
0x0010	0x04	Overflow (INTO)
0x0014	0x05	Bounds range exceeded (BOUND)
0x0018	0x06	Invalid opcode (UD2)
0x001C	0x07	Device not available (WAIT/FWAIT)
0x0020	0x08	Double fault
0x0024	0x09	Coprocessor segment overrun
0x0028	0x0A	Invalid TSS
0x002C	0x0B	Segment not present
0x0030	0x0C	Stack-segment fault
0x0034	0x0D	General protection fault
0x0038	0x0E	Page fault
0x003C	0x0F	Reserved
0x0040	0x10	x87 FPU error
0x0044	0x11	Alignment check
0x0048	0x12	Machine check
0x004C	0x13	SIMD Floating-Point Exception
0x00xx	0x14-0x1F	Reserved
0x0xxx	0x20-0xFF	User definable

subject 2.

Interrupt

Interrupt

비동기적 이벤트를 처리하기 위한 기법

예: 네트워크 패킷 도착, 마우스 이동, I/O 요청

CF) IRQ(Interrupt request)

인터럽트 처리 순서

1. 현재 실행 상태(state)를 저장
2. ISR(interrupt service routine)로 점프
3. 저장한 실행 상태(state)를 복원
4. 인터럽트로 중단된 지점부터 다시 시작

Note

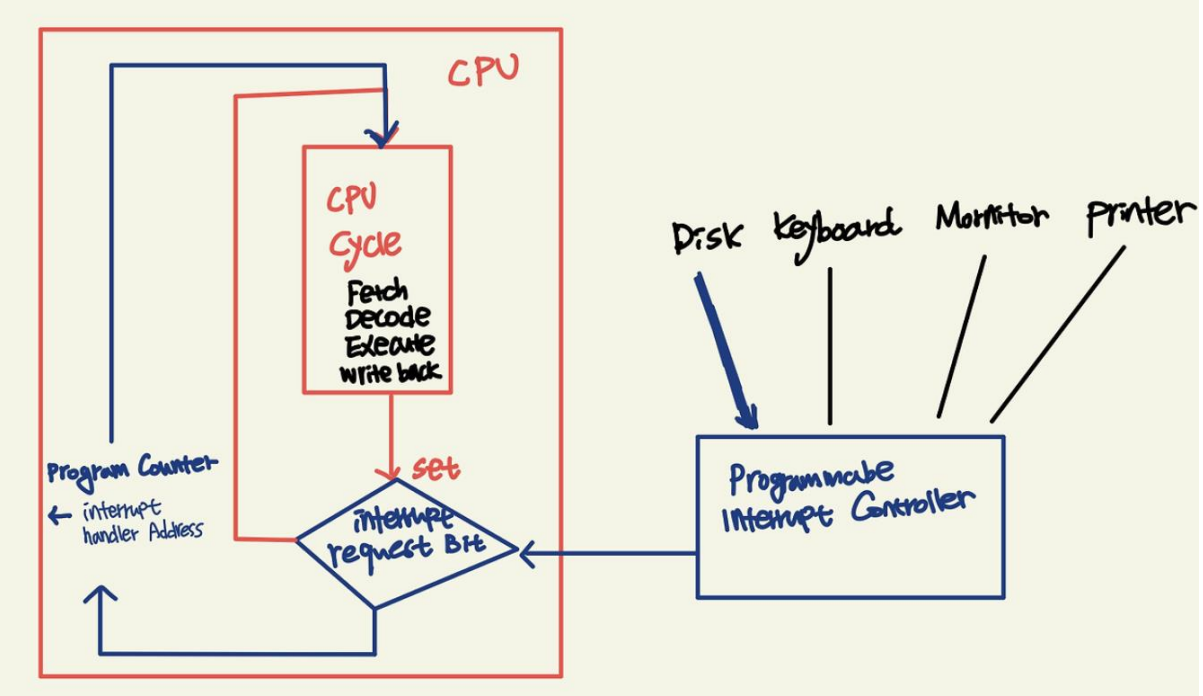
ISR은 짧아야 함

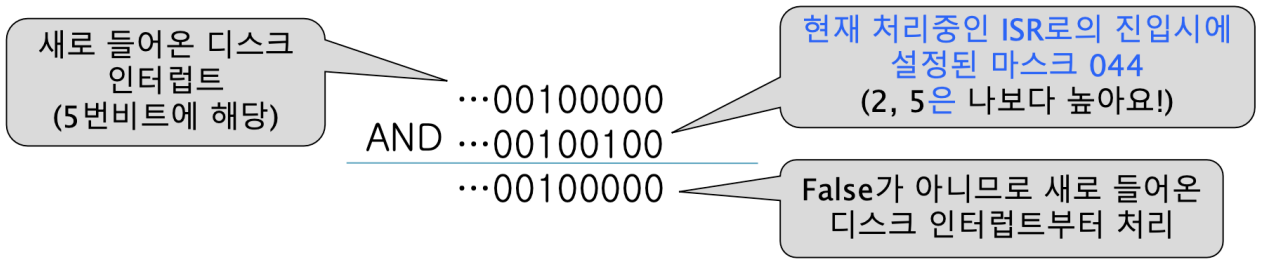
너무 길면 다른 인터럽트들이 제시간에 처리되지 못함

Timesharing역시 timer interrupt의 도움으로 가능하게 됨

multilevel interrupt

1. **(중단)** 현재 진행중인 프로세스 또는 하위의 ISR(Interrupt Service Routine) 수행을 즉시 중단
2. **(문맥보존)** 프로그램 카운터 (PC) 및 CPU 레지스터 값들을 보존
3. **(마스크설정)** 현재의 인터럽트에 해당하는 마스크를 설정하여 자신보다 하위 인터럽트가 먼저 처리되지 않도록 함
4. **(ISR진입)** 현재의 인터럽트에 해당하는 ISR으로 제어를 넘김(즉, 프로그램 카운터를 해당 ISR의 첫주소로 셋팅)
   인터럽트 백터 테이블에 장치번호 순서대로 ISR의 시작주소가 기록되어 있음





CF) PIC, APIC

Trap

동기적인 이벤트를 처리하기 위한 기법
예: 시스템 콜, divide by zero
동기적 : 현재 수행하고 있는 프로그램에 의해 발생

Trap handler에 의해 처리

하드웨어 인터럽트	소프트웨어 인터럽트
네트워크 패킷 도착	시스템콜
디바이스 인터럽트	0으로 나누기
타이머 인터럽트	오버플로우
컴퓨터 파워 다운, 정전	exception

macOS latency

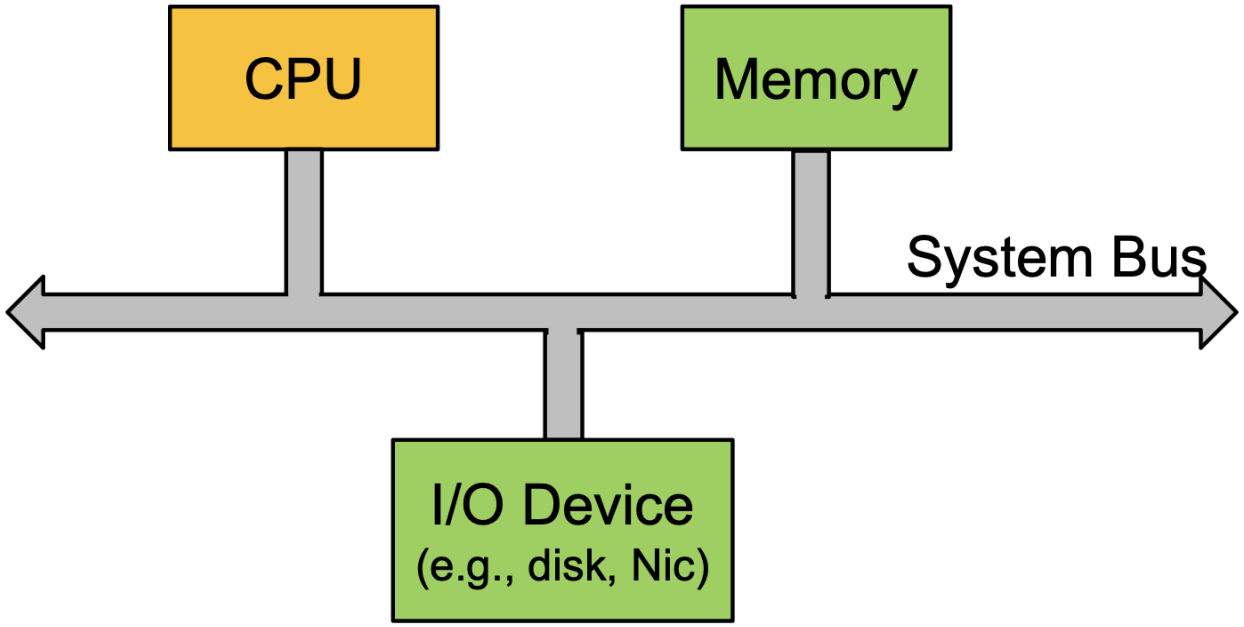
```
Mon Sep  9 17:53:10                                        0:00:10
                      SCHEDULER     INTERRUPTS
-------------------------------------------
total_samples              2549          13280

delays <  10 usecs         2126          13222
delays <  20 usecs          394             36
delays <  30 usecs           28              9
delays <  40 usecs            1             13
delays <  50 usecs            0              0
delays <  60 usecs            0              0
delays <  70 usecs            0              0
delays <  80 usecs            0              0
delays <  90 usecs            0              0
delays < 100 usecs            0              0
total  < 100 usecs         2549          13280
```

subject 3.

I/O

Bus

CPU, Memory, I/O의 속도 격차 증가 → **Bottleneck !**



<단일 버스 구조>

Bottleneck

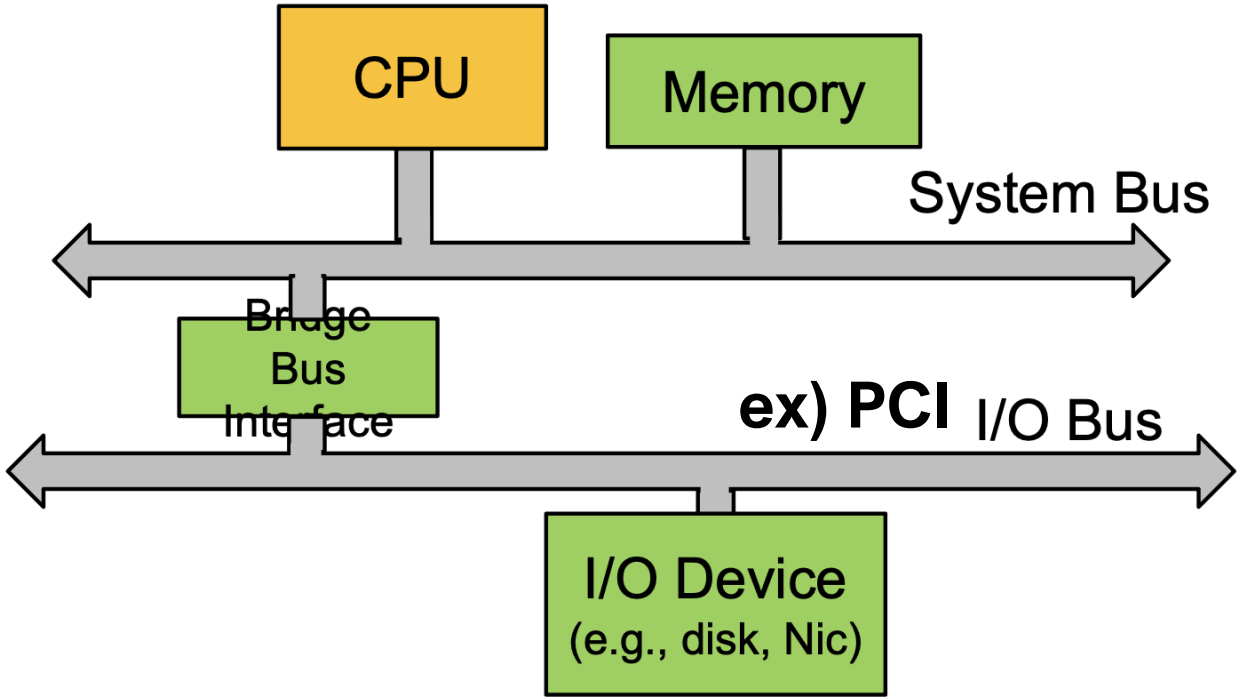
CPU > Memory > I/O

CPU : 3.4GHz, 0.29 ns (intel i7-14700K)

RAM : 5600MHz(MT/s), 44800MB/s (DDR5-5600)

SSD : 7400MB/s, 6900MB/s (Samsung 990 PRO)

전체 시스템 속도는 느린 디바이스 속도로 제한



<이중 버스 구조>

HW interface

I/O device access 기법

격리형(isolated I/O)
메모리 사상형(memory-mapped I/O)

제어 방법에 의한 분류(제어기 상태 변화 전달 방식)

폴링(polling)
인터럽트

자료이동 방식에 따른 분류

직접 입출력 방식
DMA

I/O device access 기법

Device registers

하드웨어 장치는 장치를 제어하는 controller 가 있음

Controller 는 대부분 4종류의 레지스터를 가짐 (Control, Status, Input, Output)

레지스터들은 메인 메모리의 일부 영역에 매핑 → **memory-mapped I/O**

매핑 된 영역의 주소만 알면, CPU에서 접근 가능

제어 방법에 의한 분류 : Polling

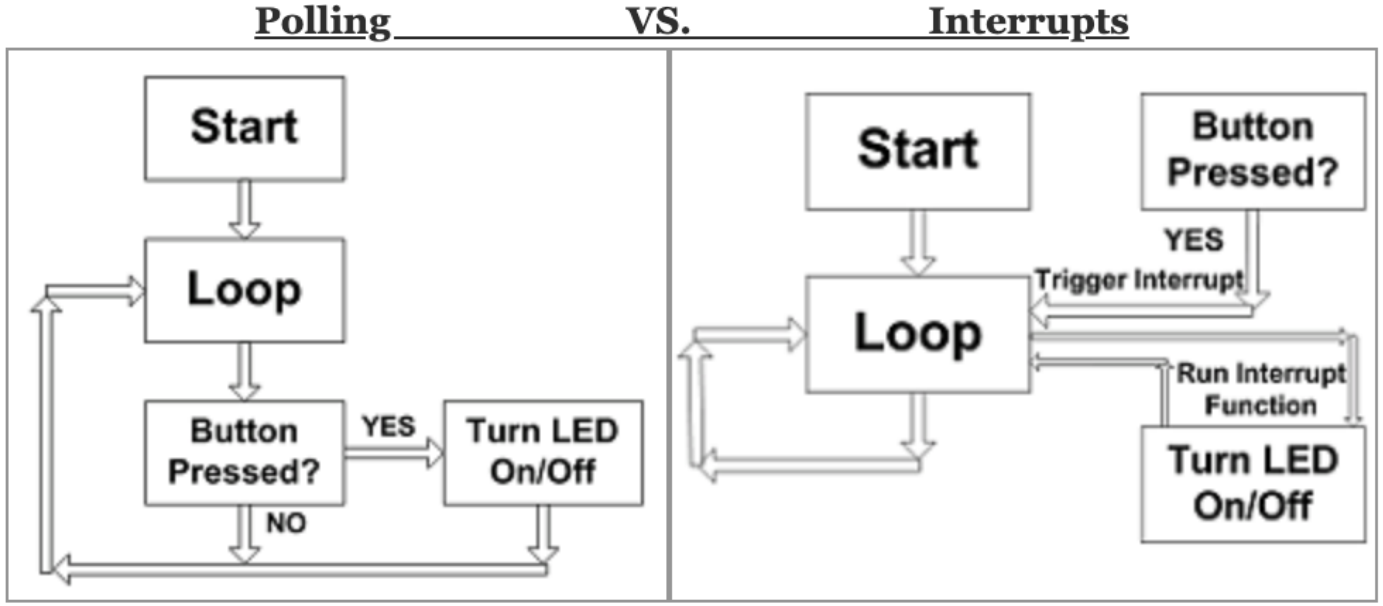
Loop안에서 특정 이벤트의 도착 여부를 계속 확인하는 방법

이벤트 도착 시간이 길 경우, 폴링은 CPU time을 낭비

일부 장치 드라이버는 I/O 발생률이 낮으면 인터럽트 사용하고,
폴링이 더 빠르고 효율적인 수준까지 발생률이 증가하면 폴링으로 전환

폴링 뒤에 programmed I/O(PIO) 수행

CPU가 상태 비트를 반복적으로 검사하면서 1바이트씩 옮기는 입출력 방식



1. **CPU Cycle 낭비**
2. **다소 부정확한 타이밍**

자료이동 방식에 따른 분류 : 직접 입출력

Programmed I/O(PIO)

CPU가 상태 비트를 반복적으로 검사하면서 1바이트씩 옮기는 입출력 방식

Interrupt I/O

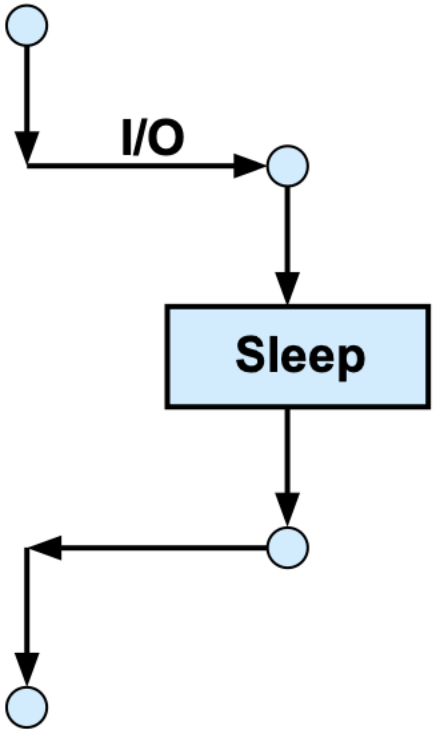
입출력 발생의 시간간격이 비교적 큰 문자장치(character device)는 별 문제가 되지 않지만 디스크와 같이 블록장치(block device)의 경우에는 문제가 됨

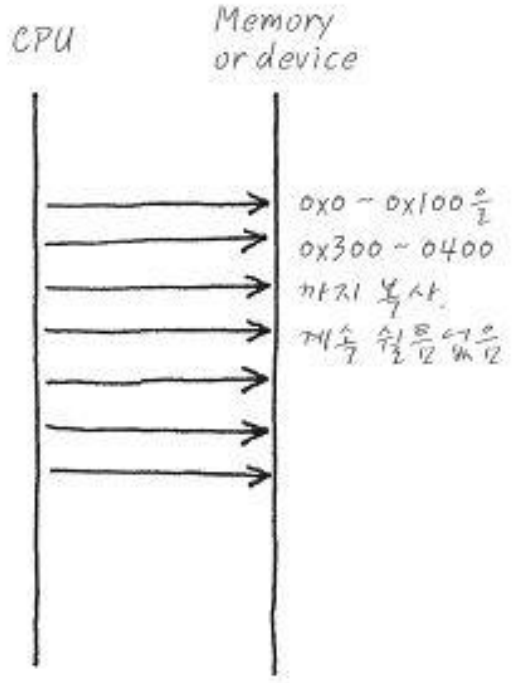
DMA (Direct Memory Access)

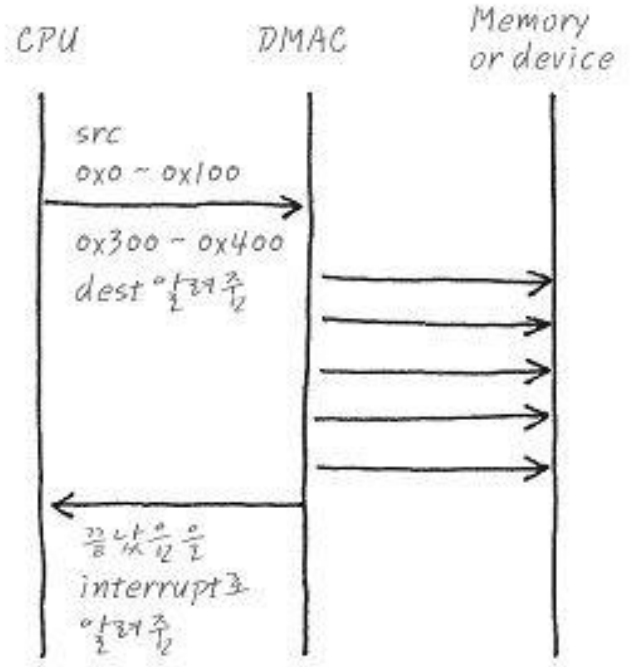
CPU를 장치의 상태 확인 및 데이터 이동에 사용하지 않고

I/O를 위한 별도의 장치(DMA) 사용

DMA 를 사용하는 경우, I/O 를 호출한 프로세스는 sleep 함







cycle stealing

DMA (Direct Memory Access)

1. CPU는 DMA controller 초기화하고 전송 모드를 DMA_MODE_READ로 설정

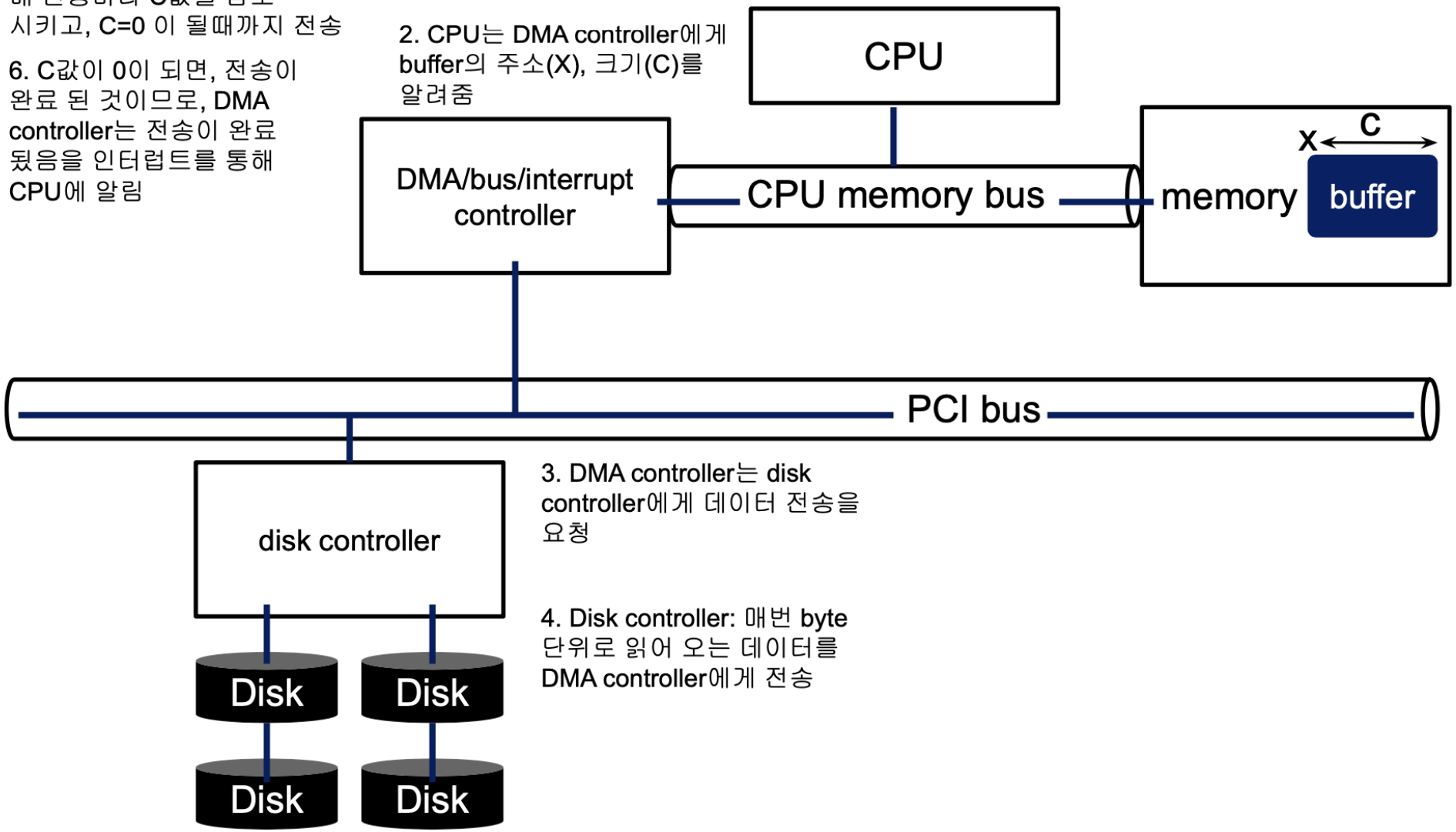
2. CPU는 DMA controller에게 buffer의 주소(X), 크기(C)를 알려줌

3. DMA controller는 disk controller에게 데이터 전송을 요청

4. Disk controller: 매번 byte 단위로 읽어 오는 데이터를 DMA controller에게 전송

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



Reference

https://www.youtube.com/watch?v=a05IvPxRLWA

http://www.pyroelectro.com/tutorials/pic_interrupts_vs_polling/theory.html

https://wiki.osdev.org/Interrupt_Vector_Table

https://www.youtube.com/watch?v=6Q5Gb1fxNMk&t=860s

https://hasensprung.tistory.com/179

Operating System Concepts (10/E, Silberschatz)

Computer Systems A Programmer's Perspective (3/E, Randal E. Bryan)

Computer Organization And Design (6/E, David A. Patterson)