디지털컨버전스 기반 UI UX Front 전문 개발자 양성과정

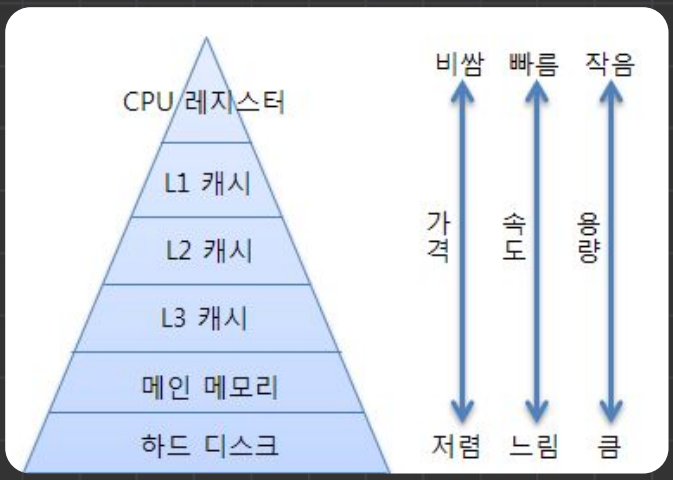
Lecturer silenc3502 Sanghoon Lee (이상훈)
gcccompil3r@gmail.com

Student may_hz HyeonJeong Choi (최현정)
hyeonjeong9943@gmail.com

Memory Hierarchy

Date 2021_01_25

용량이 크면 클 수록 동작 **속도**가 느리고
속도가 빠르면 빠를수록 **용량**이 작다.



CPU가 데이터를 요청하면 메인 메모리에서 해당 데이터뿐만아니라 근접한 데이터들과 함께 블록 단위 캐시로 가져오게 된다. — Spatial Locality

그다음 캐시에서 해당 데이터만 레지스터에 전달하게 된다. — << Caching

다음에 CPU가 데이터를 요청할 경우 바로 메인 메모리에서 데이터를 가져오는 것이 아니라 일단 캐시에 데이터가 있는지 요청을 하게 되고 만약 있다면 캐시에서 바로 해당 데이터를 가져오게 된다. — Temporal Locality / << Cache Hit

캐시에 데이터가 없어서 메인메모리에 데이터를 요청하게 되는 것은? — << Cache Miss

<< 하드디스크에 있는 프로그램 내용을 실행 할때 >>

메인 메모리로 이동 -> L3 캐시로 이동 -> L2 캐시로 이동 -> L1 캐시로 이동 -> 연산에 필요한 데이터는 레지스터 이동

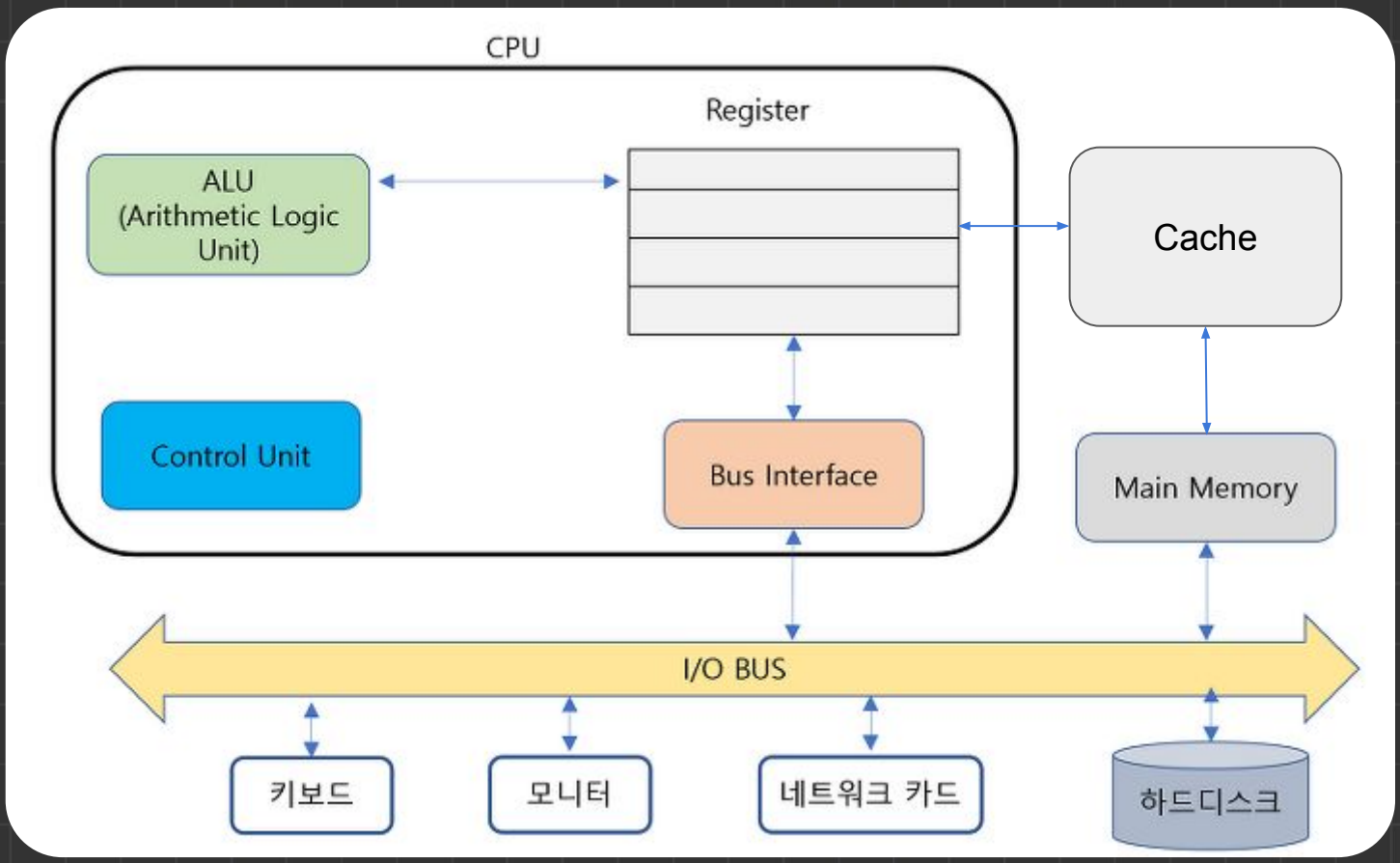
<< 연산에 필요한 데이터가 레지스터에 없을때 >> 이경우 극심한 속도저하가 발생한다.

L1 캐시를 살펴봄 -> 없으면 L2캐시 -> 없으면 메인 메모리 -> 없으면 하드디스크 참조 -> 하드디스크에서 데이터를 찾은 후 -> 메인 메모리 -> L2 캐시 -> L1 캐시 -> 레지스터로 데이터가 들어오게 됨

캐시를 없애 중간단계를 줄이는 것이 속도가 빠르지 않냐 생각할수 있는데
L1 캐시와 L2 캐시에 연산에 필요한 데이터가 존재할 확률이 90% 이상이다.
따라서 캐시는 속도향상에 도움을 준다

I/O [Input / Output]

Date 2021_01_25



메모리와 다른 입출력 장치와 통신을 하는 Subsystem을 I/O Bus 라고한다.

컴퓨터 구조적으로 보았을때 I/O를 실행한다는 것은 CPU 외부로

나갔다가 다시 들어오는 것으로 속도가 떨어질 수 밖에 없을 것 이다.

결국 모니터에 출력하는 행위 자체가 I/O 를 빈번하게 발생시키는 것이므로

I/O 를 최소화하면 어떤 프로그램이던 속도가 엄청나게 빨라진다.

Synchronized

Date 2021_01_25

```
class A extends Thread {
    public void run() {
        WhyThreadMutex.fb.plusMoney(3000);
        System.out.println("plusMoney(3000): " + FailedBank.getMoney());
    }
}

class B extends Thread {
    public void run() {
        WhyThreadMutex.fb.minusMoney(1000);
        System.out.println("minusMoney(1000): " + FailedBank.getMoney());
    }
}
```

Race Condition >>

↑↑↑ Critical Section ↑↑↑

두 개 이상의 프로세스나 스레드가 **하나**의 데이터를 공유할 때
데이터가 **동기화** 되지 않는 상황 (하나의 자원을 갖기 위해 싸우는 것)

Critical Section 문제를 해결하기 위해

Synchronized (동기화) 하여 여러 테스크가

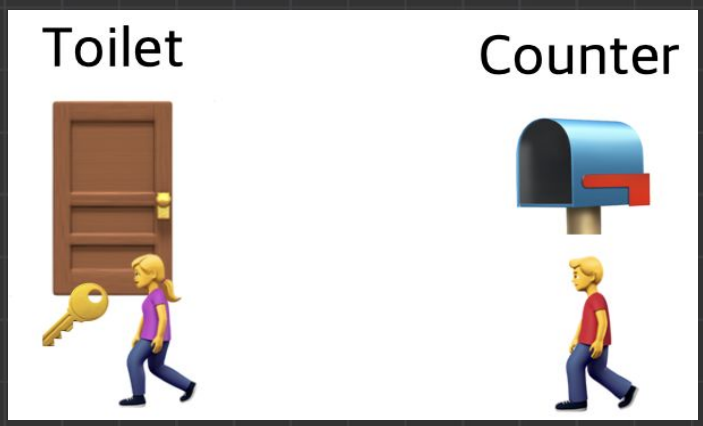
동시에 접근하는 것을 보호하고

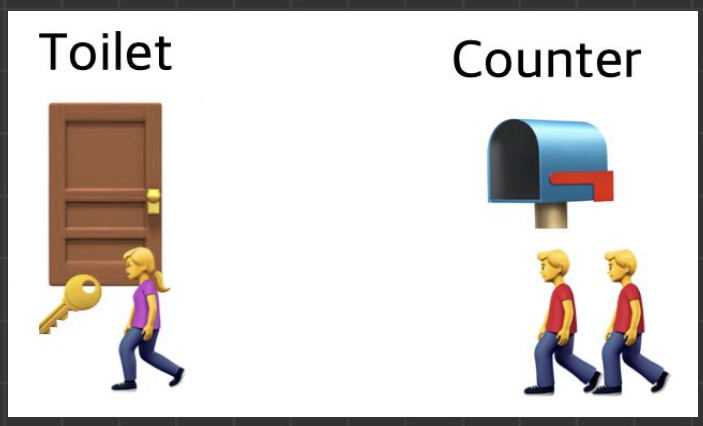
한개의 테스크만이 자원을 공유할 수 있도록 한다.

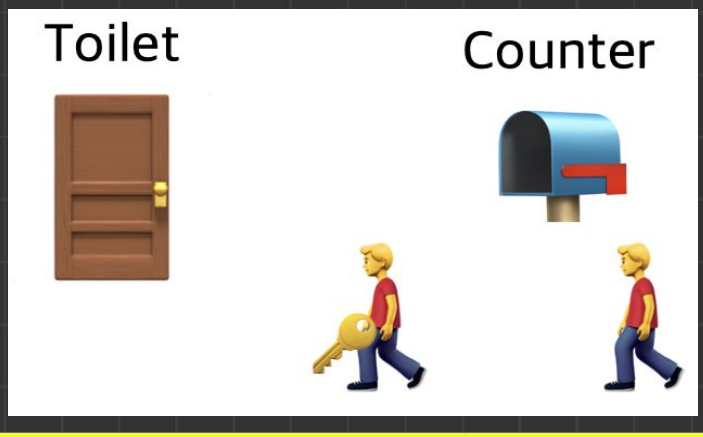
Mutex / Semaphore

Date 2021_01_25

Mutex







Semaphore