21강. 다양한 입출력 방법

컴퓨터에서 CPU와 입출력장치 간의 데이터 교환은 여러 방식으로 이루어집니다. 대표적인 방법으로 다음 세 가지가 있습니다.

- 프로그램 입출력
- 인터럽트 기반 입출력
- DMA(Direct Memory Access) 입출력

이 방식들은 CPU가 입출력장치와 데이터를 주고 받는 과정에서 얼마나 개입하는지에 따라 구분됩니다.

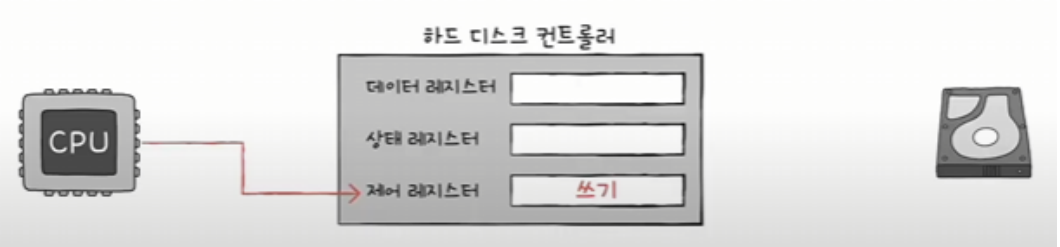
프로그램 입출력

프로그램 입출력은 프로그램 내부의 명령어를 통해 입출력장치를 직접 제어하는 방식입니다.

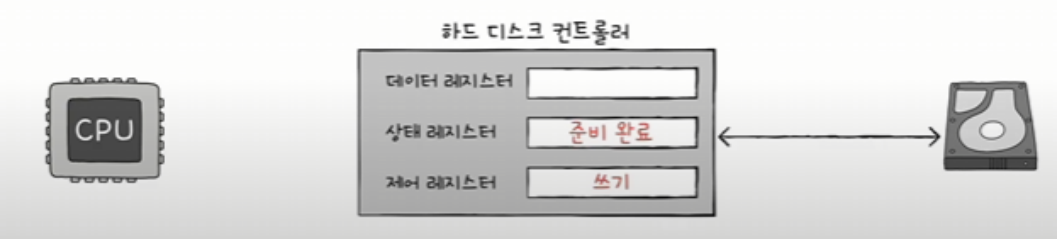
즉, CPU가 장치 컨트롤러의 레지스터에 직접 접근해 데이터를 읽고 쓰는 방식으로 입출력을 수행합니다.

예를 들어, 메모리에 저장된 데이터를 하드디스크에 백업한다고 합시다.

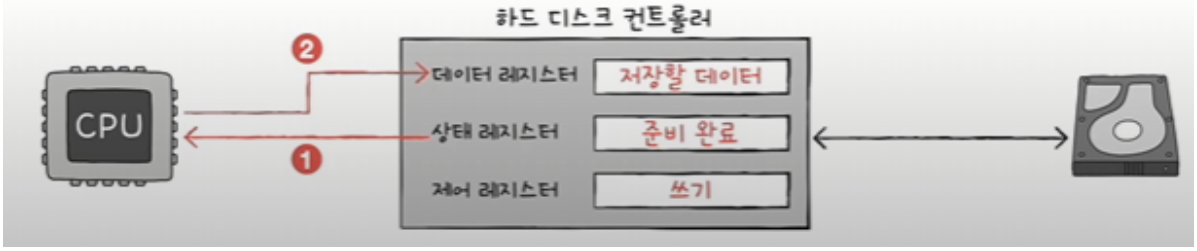
1. CPU는 먼저 하드디스크 컨트롤러의 제어 레지스터에 “쓰기(write)” 명령을 보냅니다.



2. 하드디스크 컨트롤러는 장치의 상태를 확인하고, 준비가 완료되면 상태 레지스터에 이를 표시합니다.



3. CPU는 주기적으로 상태 레지스터를 읽어보며 하드디스크가 준비되었는지 확인합니다.
4. 준비가 완료되면 CPU는 백업할 데이터를 데이터 레지스터에 써서 하드디스크로 전송합니다.



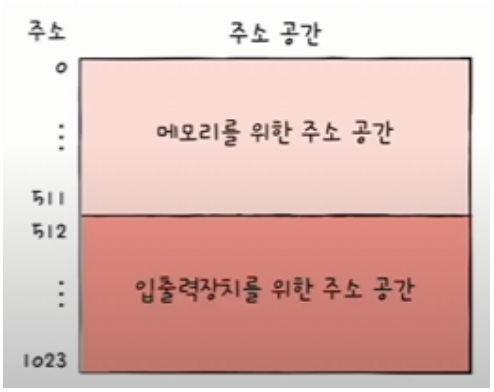
- 만약 작업이 아직 끝나지 않았다면 이 과정을 반복합니다.

결국 프로그램 입출력은 CPU가 장치 컨트롤러의 레지스터를 직접 읽고 쓰는 과정으로 이루어집니다.

이때 CPU가 장치 컨트롤러의 주소를 인식해야 하는데, 이를 위해 두 가지 접근 방식이 존재합니다.

- 메모리 맵 입출력
- 고립형 입출력

메모리 맵 입출력



메모리 맵 입출력(Memory-Mapped I/O)은 메모리와 입출력장치의 주소 공간을 하나로 통합해 사용하는 방식입니다.

즉, 메모리의 특정 주소를 입출력장치의 레지스터로 간주하고, CPU는 일반적인 메모리 접근 명령어로 입출력장치에 접근합니다.

예를 들어,

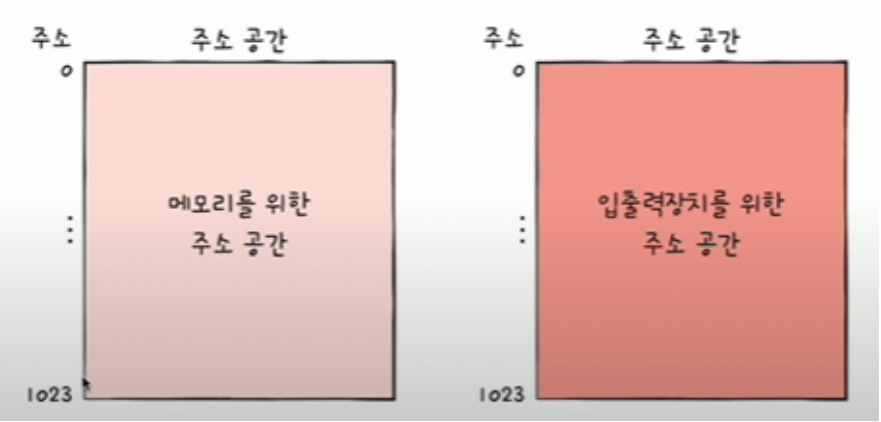
- 516번지 : 프린터 컨트롤러의 데이터 레지스터
- 517번지 : 프린터 컨트롤러의 상태 레지스터
- 518번지 : 하드 디스크 컨트롤러의 데이터 레지스터
- 519번지 : 하드 디스크 컨트롤러의 상태 레지스터

이때 CPU가 `517번지를 읽어라` 라고 명령하면 이는 곧 프린터의 상태를 읽는 것이 됩니다.

`518번지에 a를 써라` 라고 명령하면 하드디스크에 데이터를 쓰는 것이 됩니다.

메모리 접근 명령어와 입출력 명령어가 동일하게 사용되며, 주소 공간이 통합되어 있다는 것이 특징입니다.

고립형 입출력



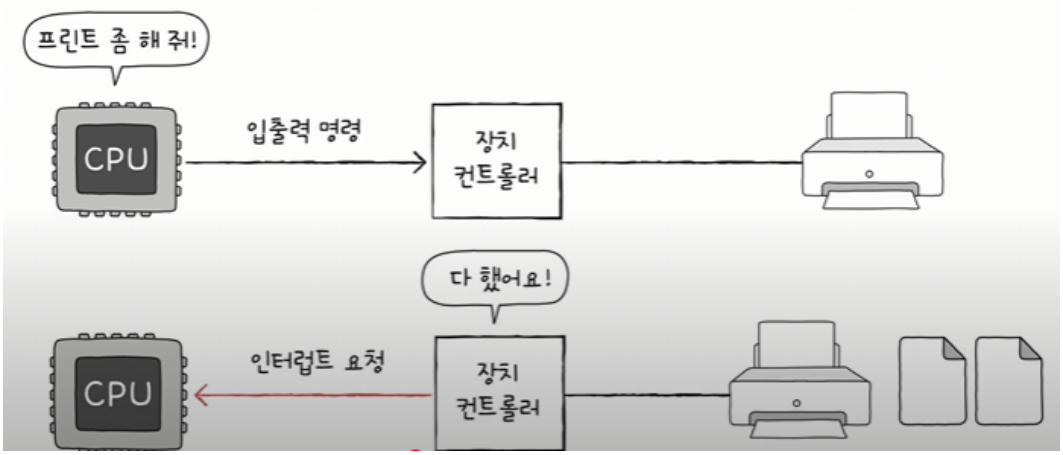
고립형 입출력(Isolated I/O)은 메모리와 입출력장치의 주소 공간을 완전히 분리하는 방식입니다. 이 방식에서는 메모리 접근용 명령어와 입출력 전용 명령어가 구분되어 있으며, CPU는 입출력 전용 제어 신호를 사용해 장치에 접근합니다.

따라서 고립형 입출력은 메모리 공간을 침범하지 않지만, 별도의 입출력 명령이 필요해 하드웨어 설계가 조금 더 복잡해집니다.

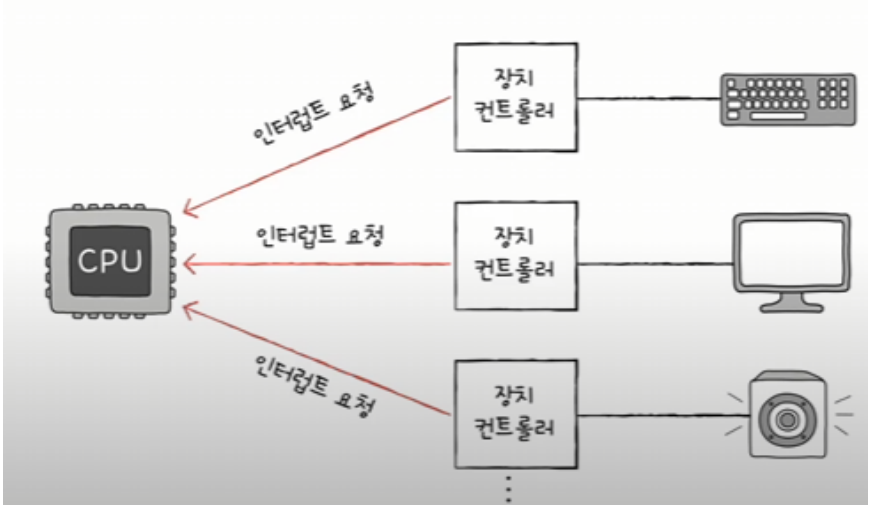
메모리 맵 입출력	고립형 입출력
메모리와 입출력장치는 같은 주소 공간 사용	메모리와 입출력장치는 분리된 주소 공간 사용
메모리 주소 공간이 축소됨	메모리 주소 공간이 축소되지 않음
메모리와 입출력장치에 같은 명령어 사용 가능	입출력 전용 명령어 사용

인터럽트 기반 입출력

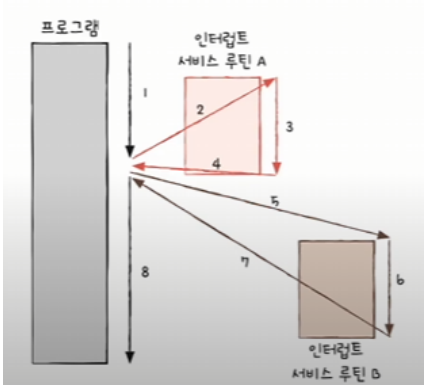
인터럽트 기반 입출력(Interrupt-Driven I/O)은 장치가 준비될 때 CPU를 직접 호출하는 방식입니다.

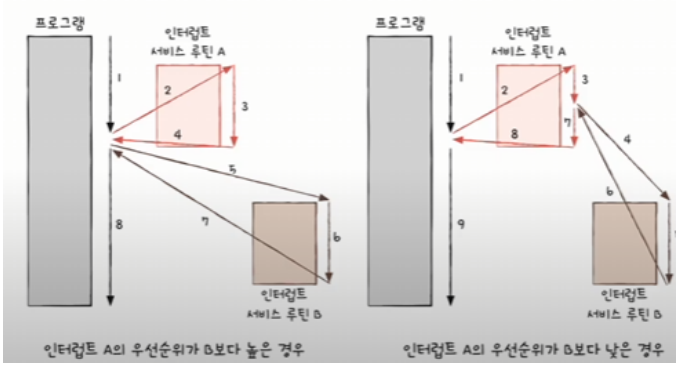


기존의 프로그램 입출력 방식에서는 CPU가 장치의 상태를 계속 확인해야 했기 때문에 CPU 자원이 낭비되었습니다. 하지만 인터럽트 기반 방식에서는 장치 컨트롤러가 하드웨어 인터럽트를 발생시켜 CPU에 알립니다.



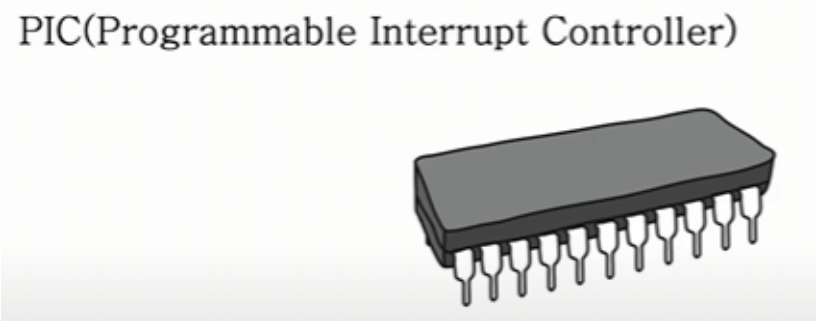
이 방식에서는 여러 장치가 동시에 인터럽트를 발생시킬 수도 있는데, 이때 단순히 “발생한 순서대로” 처리하는 것은 비효율적입니다. 그 이유는 인터럽트마다 중요도(우선순위)가 다르기 때문입니다.



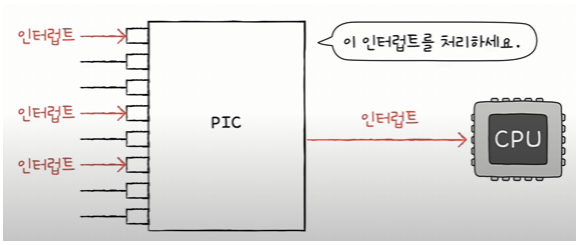


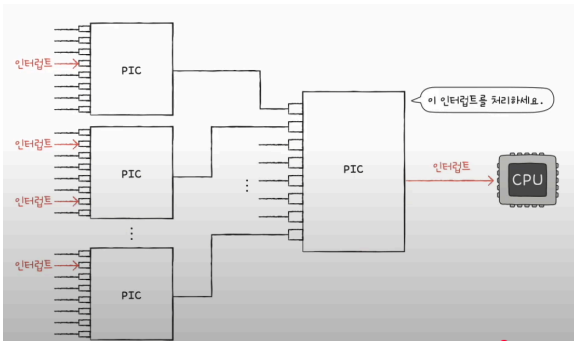
이를 해결하기 위해 우선순위 기반 인터럽트 처리 (Priority Interrupt Handling)가 도입되었습니다.

현대 시스템에서는 PIC(Programmable Interrupt Controller)라는 하드웨어가 이를 담당합니다.



PIC는 여러 장치 컨트롤러의 인터럽트 요청을 받아, 각 요청의 우선순위를 판단하고 가장 먼저 처리해야 할 인터럽트를 CPU에 전달합니다.



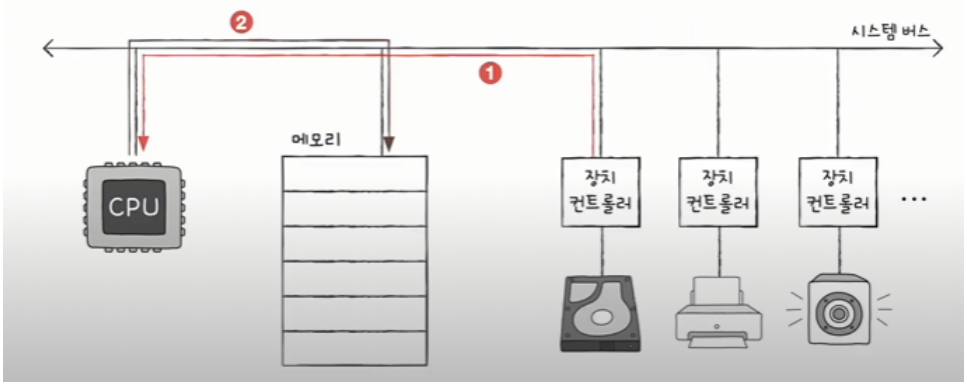


이 덕분에 중요한 작업(e.g. 전원 이상, 디스크 오류 등)은 먼저 처리되고, 덜 중요한 작업은 나중에 처리됩니다.

DMA 입출력

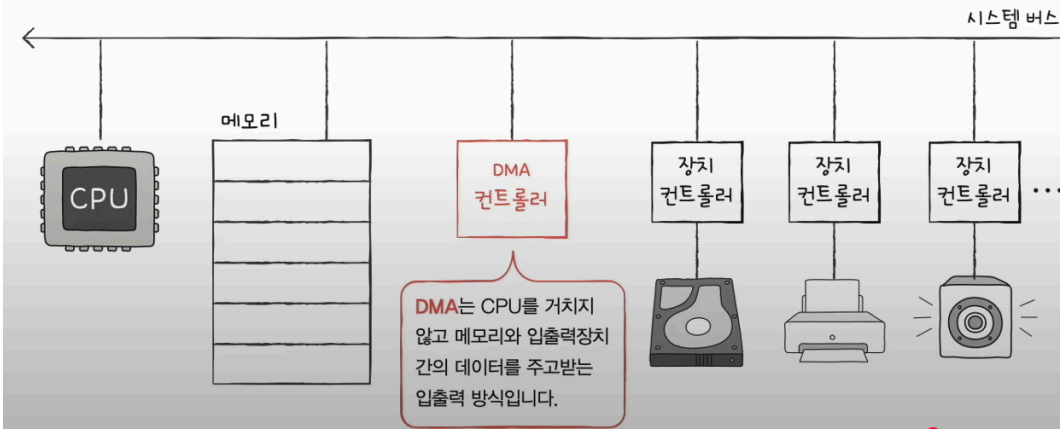
DMA(Direct Memory Access)는 프로그램 입출력과 인터럽트 기반 입출력의 한계를 개선한 방식입니다.

앞선 두 방식 (프로그램, 인터럽트 기반 입출력)의 공통점은 입출력장치와 메모리 간의 모든 데이터 이동이 CPU를 거친다는 점입니다.



이로 인해 CPU는 대용량 데이터 전송 중에도 개입해야 하므로, 처리 효율이 떨어지고 부담이 커집니다.

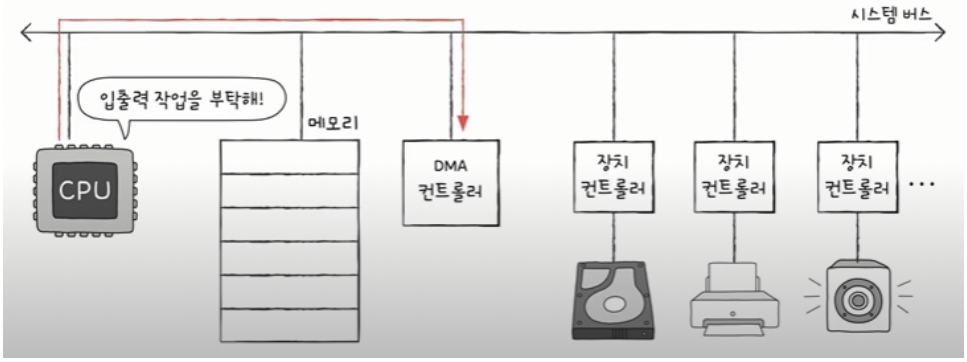
DMA 방식은 이러한 문제를 해결하기 위해 CPU를 거치지 않고 입출력장치가 메모리에 직접 접근할 수 있도록 하는 기능을 제공합니다.



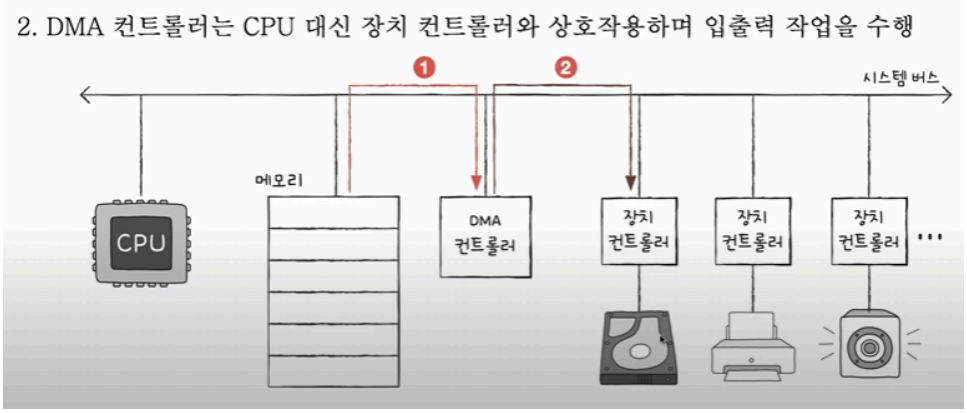
이때 실제 제어는 DMA 컨트롤러 (DMA Controller)라는 하드웨어가 담당합니다.

DMA 입출력 과정

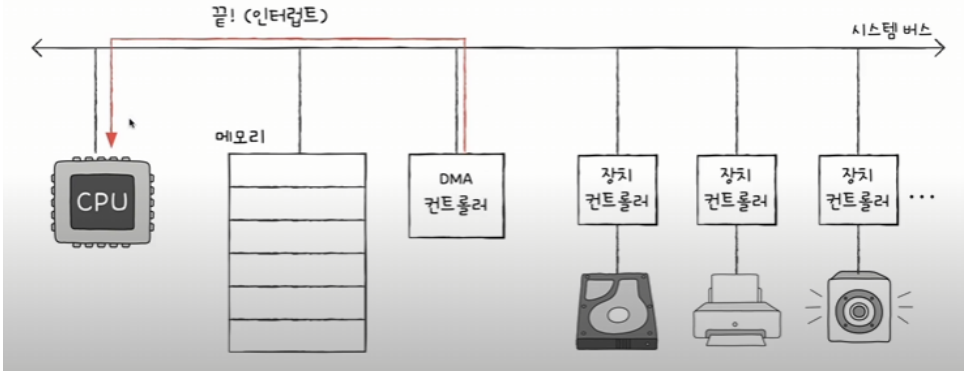
1. CPU가 DMA 컨트롤러에 입출력 작업을 명령합니다.



2. DMA 컨트롤러는 CPU 대신 장치 컨트롤러와 통신하며 데이터를 메모리로 전송합니다.

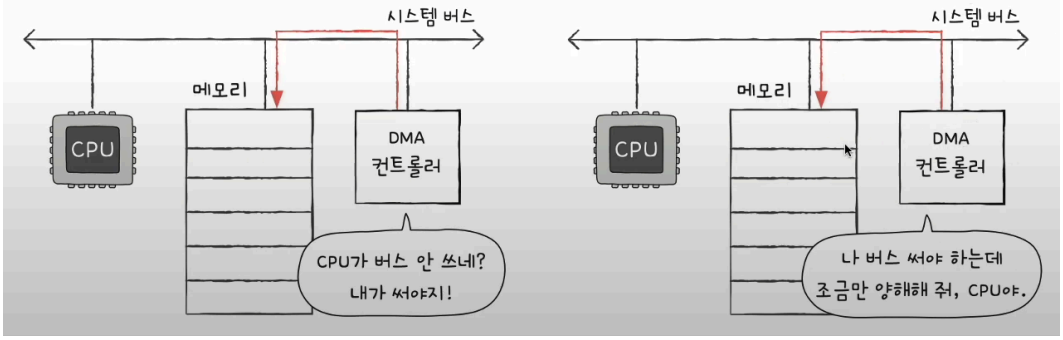


3. 작업이 완료되면 DMA 컨트롤러는 인터럽트를 통해 CPU에 완료 사실을 알립니다.



이 과정에서 DMA 컨트롤러 역시 시스템 버스를 사용합니다. 그러나 시스템 버스는 CPU와 공유되는 공용 자원이기 때문에, DMA는 CPU가 버스를 사용하지 않을 때 조금씩 사용하거나 CPU의 사용을 잠시 양보받아 데이터를 전송합니다.

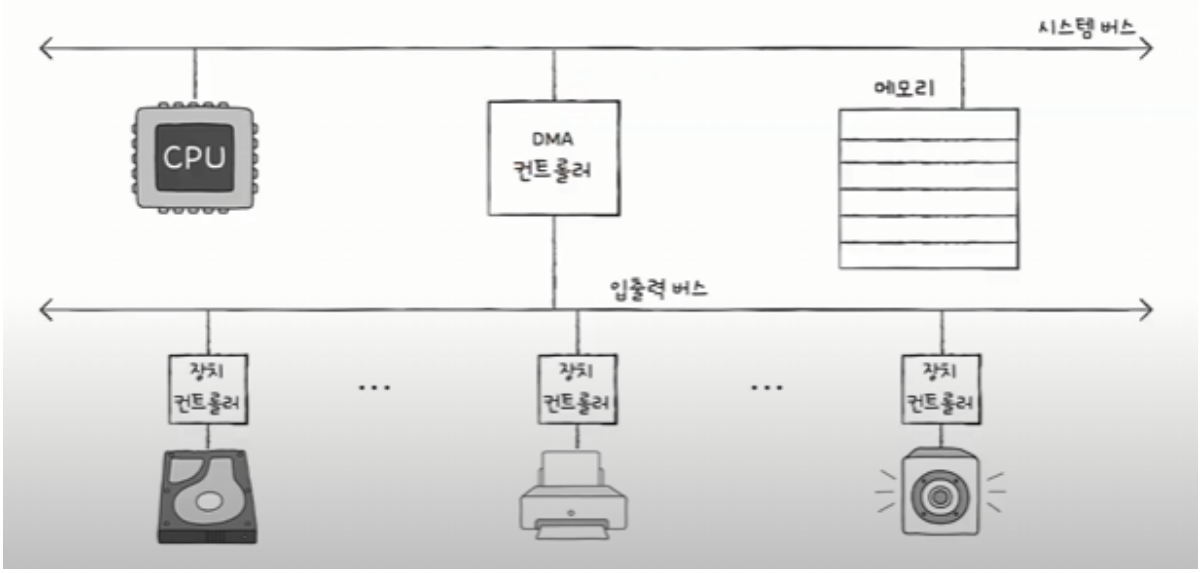
이를 사이클 스틸링(Cycle Stealing) 또는 버스 허락(Bus Arbitration) 방식이라고 합니다.



입출력 버스

마지막으로, 장치 컨트롤러가 직접 시스템 버스에 연결될 경우 버스 충돌이 발생할 수 있습니다.

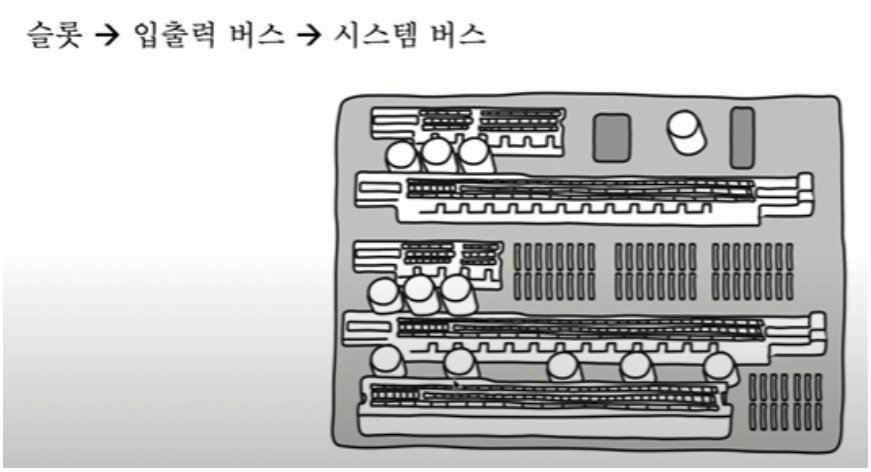
이를 방지하고 효율적으로 입출력장치를 관리하기 위해 입출력 버스 (I/O Bus)라는 별도의 통로가 사용됩니다.



입출력 버스는 장치 컨트롤러와 시스템 버스 사이에 위치하며, CPU가 모든 장치와 직접 통신하는 대신 입출력 버스 → 시스템 버스 경로를 통해 데이터를 전달합니다.

이로써 시스템 버스의 사용 빈도를 줄이고, 여러 장치가 동시에 연결되어도 효율적인 통신이 가능해집니다.

대표적인 입출력 버스로는 PCI(Peripheral Component Interconnect) 버스와 그 이후 발전된 PCI Express(PCIe) 버스가 있습니다.



이들은 메인보드 확장 슬롯을 통해 그래픽카드, 네트워크 카드, SSD 등의 장치를 연결하는 데 사용됩니다.