I/O 및 인터럽트
(제 13주 차)

서울사이버대학교

오 창 환

학습 목표

- I/O 장치의 접속을 설명할 수 있다.
- 인터럽트를 이용한 I/O를 설명할 수 있다.
- DMA를 이용한 I/O를 설명할 수 있다.

학습 내용

- I/O 장치의 접속
- 인터럽트를 이용한 I/O
- DMA를 이용한 I/O

I/O 장치의 접속 (1)

(1) I/O 제어

• I/O 장치들이 시스템 버스에 직접 접속되지 못하는 이유는 다음과 같음.

* I/O 장치들은 종류가 매우 다양하며, 동작을 제어하는 방법이 서로 다름.
  그러한 제어를 위한 회로들을 CPU 내부에 모두 포함시키는 것이 불가능하기 때문에 CPU가 그들을 직접 제어할 수가 없음.
* I/O 장치들의 데이터 전송 속도가 CPU의 데이터 처리 속도에 비해 훨씬 더 느리므로 고속의 시스템 버스와 I/O 장치들 사이에 직접 데이터를 교환하는 것은 불가능함.
* I/O 장치들과 CPU가 사용하는 데이터 형식의 길이가 서로 다른 경우가 많음.

• 상기와 같은 이유로 I/O 장치들은 시스템 버스에 직접 접속되지 못하며, 인터페이스가 필요한데 이러한 인터페이스 역할을 담당하는 장치를 I/O 제어기(I/O controller)라고 하며, 그것의 주요 기능은 다음과 같음.

* I/O 장치의 제어와 타이밍을 조정함.
* CPU와의 통신을 담당함.
* I/O 장치와의 통신을 담당함.

I/O 장치의 접속 (2)

* 데이터 버퍼링(data buffering) 기능을 수행함.

* 오류를 검출함.

• CPU가 프린터로 데이터를 출력하는 동작은 다음 순서에 따라 이루어짐.

① CPU가 프린터 제어기에게 프린터의 상태를 검사하도록 요청함.

② 제어기는 프린터의 상태를 검사하여 CPU에게 알려줌.

이때 프린터의 상태란 데이터를 받아 프린트 할 준비가 되어있는지,

혹은 다른 데이터를 프린트하는 중인지를 나타냄.

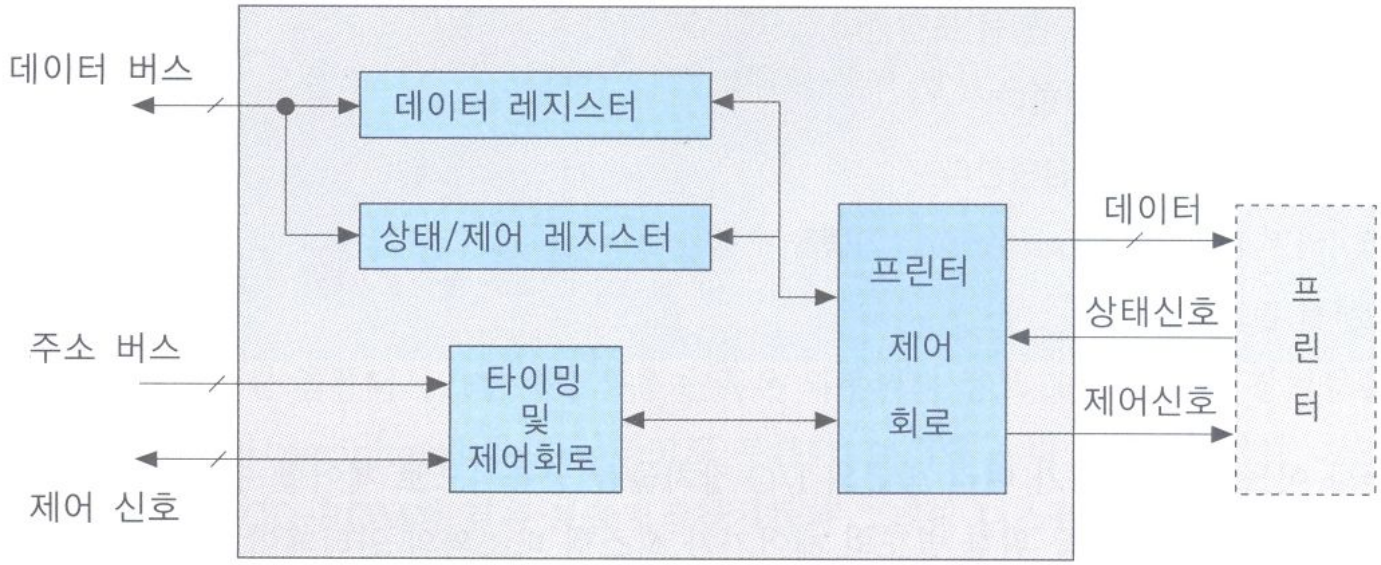
③ 프린터가 다음 데이터를 받을 준비가 된 상태라면,

CPU는 제어기에게 출력 명령과 데이터를 보냄.

④ 제어기는 프린트 동작을 위한 제어 신호와 함께 데이터를 프린터로 보냄.

I/O 장치의 접속 (3)

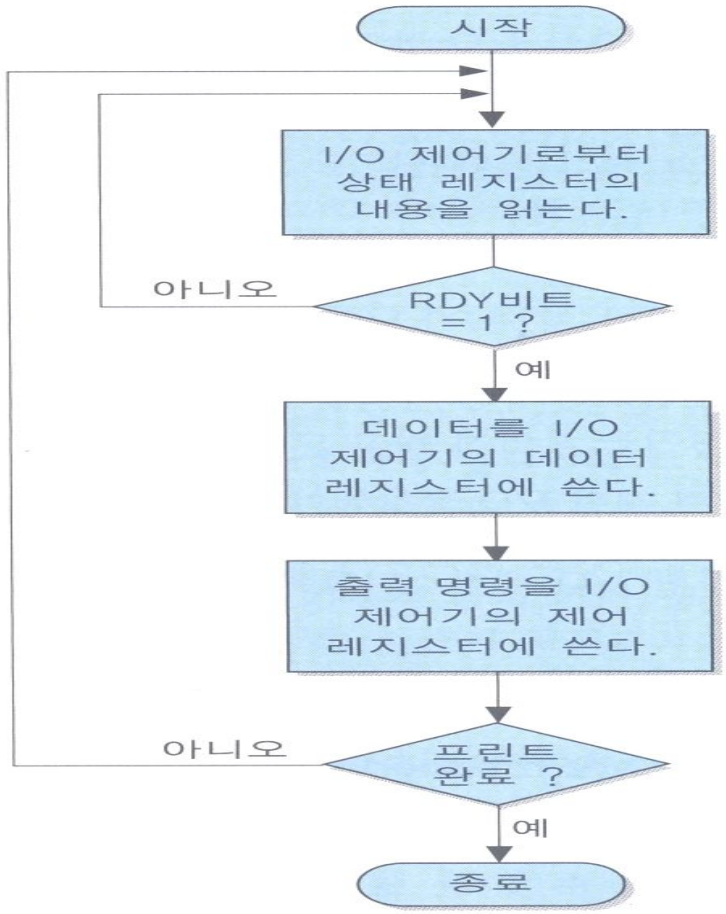
- 아래 그림에서 상태/제어 레지스터(status/control register)는 내부적으로 두 개의 레지스터들로 구성되지만 주소는 하나만 지정됨.
  상태 레지스터는 프린터의 상태와 오류 검사 결과 등을 나타내는 비트들로 구성되어 있어서, CPU가 프린터의 상태를 검사할 때는 그 주소를 이용하여 상태 레지스터의 내용을 읽게 됨.
  CPU가 제어기에게 출력 명령을 보낼 때는 제어 레지스터에 해당 명령 단어를 쓰면 됨.



- 프린터 제어기의 내부 구성도

I/O 장치의 접속 (4)

• 아래 그림과 같이 프로그램을 이용한 I/O 방식은 별도의 하드웨어가 필요하지 않다는 장점이 있지만, CPU가 I/O 동작을 수행하는 동안에 다른 일을 하지 못하는 단점이 있음.



• 프로그램을 이용한 I/O의 흐름도

I/O 장치의 접속 (5)

(2) I/O 주소 지정

(가) 기억장치-사상 I/O

• 기억장치-사상 I/O(memory-mapped I/O) 방식에서는 I/O 제어기 내의 레지스터들을 기억장치 내의 기억장소들과 동일하게 취급하고, 레지스터들의 주소도 기억장치 주소 영역의 일부분을 할당함.

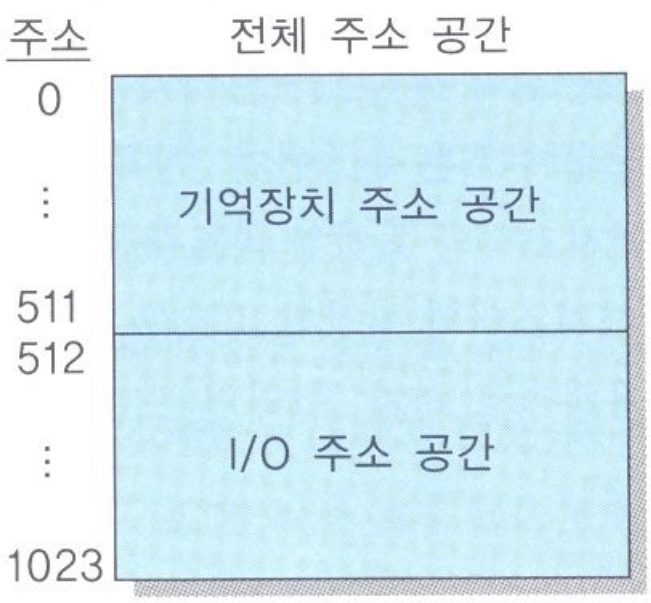
따라서, 기억장치와 I/O 레지스터들을 액세스할 때 동일한 기계 명령어들을 사용할 수 있으며, CPU로부터 발생되는 기억장치 읽기 신호와 쓰기 신호를 이용하여 I/O 장치에 대한 읽기와 쓰기도 수행할 수 있음.

```
TEST:   LOAD    413     ; 상태 레지스터의 내용을 읽는다.
        ANI     01      ; RDY 비트를 제외한 모든 비트들을 0으로 clear한다.
        JZ      TEST    ; 만약 RDY 비트가 0이라면, TEST로 점프한다.
        LOAD    M       ; 프린트할 데이터를 기억장치로부터 읽어온다.
        STOR    412     ; 프린트할 데이터를 데이터 레지스터에 쓴다.
        LOAD    80H     ; AC에 2진수 10000000을 적재한다 (start 비트 ← 1).
        STOR    413     ; 프린트 시작 명령을 보낸다.
```

• 기억장치-사상 I/O 방식의 예

I/O 장치의 접속 (6)

- 기억장치-사상 I/O 방식에서는 I/O 장치들에 대한 주소로서 기억장치 주소 공간의 일부를 할당하기 때문에 기억장치를 위한 주소 공간이 그만큼 감소하게 됨.
  예를 들어, 주소 비트들이 10비트인 시스템에서 주소 지정이 가능한 전체 기억 장소들의 수는 1024개이지만, 기억장치-사상 I/O 방식이 사용되면 그 수가 절반으로 줄어듦.



• 기억장치-사상 I/O에서의 주소 공간 할당

I/O 장치의 접속 (7)

(나) 분리형 I/O

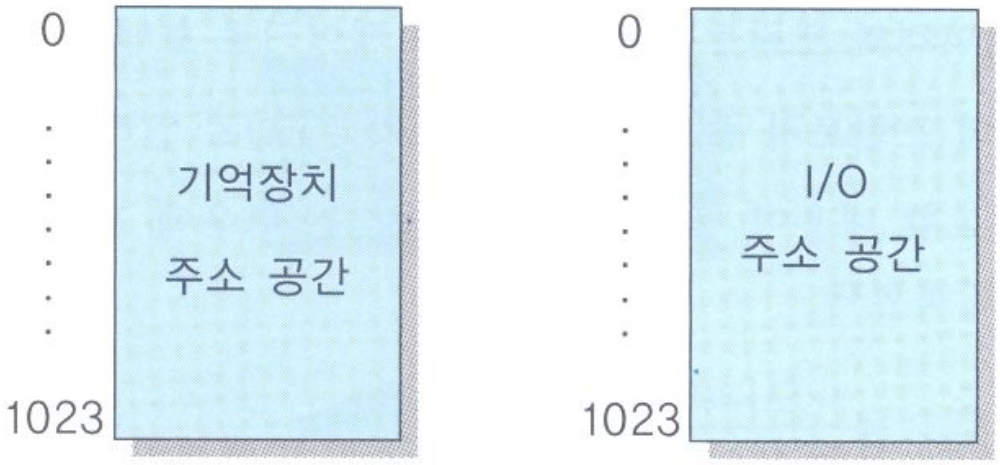
- 분리형 I/O 방식에서는 I/O 장치들의 주소 공간이 기억장치 주소 공간과는 별도로 할당된다는 점이 기억장치-사상 I/O와의 차이점임. 이 방식을 사용한 시스템에서는 I/O 레지스터들을 액세스할 때는 반드시 별도의 명령어들, 즉 I/O 명령어들을 사용해야 함. 그리고 이러한 I/O 장치에 대한 것인지를 구분할 수 있도록 I/O 읽기(I/O READ)와 I/O 쓰기(I/O WRITE) 신호가 CPU로부터 발생해야 함.
- 아래 예에서 I/O 장치의 데이터 레지스터의 주소는 0번지, 상태/제어 레지스터의 주소는 1번지로 가정함.

```
TEST :   IN      1     ; 상태 레지스터의 내용을 읽는다.
         ANI     01    ; RDY비트를 제외한 모든 비트들을 0으로 clear한다.
         JZ      TEST  ; 만약 RDY비트가 0이라면, TEST로 점프한다.
         LOAD    M     ; 프린트할 데이터를 기억장치로부터 읽어온다.
         OUT     0     ; 프린트할 데이터를 데이터 레지스터에 쓴다.
         LOAD    80H   ; AC에 2진수 10000000을 적재한다 (start 비트 ← 1).
         OUT     1     ; 프린트 시작 명령을 보낸다.
```

• 분리형 I/O 방식의 예

I/O 장치의 접속 (8)

- 이 예에서 CPU가 I/O 레지스터의 내용을 읽는 동작은 IN addr 명령어를 이용하여 수행하고, I/O 레지스터로 쓰는 동작은 OUT addr 명령어를 이용하여 수행하였음. 즉 I/O 제어를 위해서는 이 두 명령어들만 이용해야 하기 때문에 프로그래밍이 불편해지는 것이 이 방식의 단점임.
  그러나, 이 방식을 사용하는 경우에는 I/O 주소 공간이 기억장치 주소 공간과는 별도로 지정될 수 있는 장점이 있음.



• 분리형 I/O 에서의 주소 공간 할당

2 교시

인터럽트를 이용한 I/O (1)

- 인터럽트 메커니즘을 이용하면 I/O 동작이 I/O 제어기와 I/O 장치 사이에서 진행되는 동안에 CPU는 다른 작업을 처리할 수 있게 되므로 시간을 유용하게 활용할 수 있음.
- 인터럽트-구동 I/O(interrupt-driven I/O)의 동작 순서는 아래와 같음.

① CPU가 I/O 제어기에게 명령을 보냄. 그런 다음에 CPU는 다른 작업을 수행함.

② 제어기는 I/O 장치를 제어하여 I/O 명령을 수행함.

③ I/O 명령 수행이 완료되면, 제어기는 CPU로 인터럽트 신호를 보냄.

④ CPU는 인터럽트 신호를 받는 즉시 원래의 프로그램으로 돌아와서 수행을 계속함.

위의 과정에서 ②와 ③이 진행되는 동안에 CPU는 다른 프로그램을 수행할 수 있으므로 I/O 장치의 속도가 느린 경우에도 CPU 시간이 낭비되지 않음.

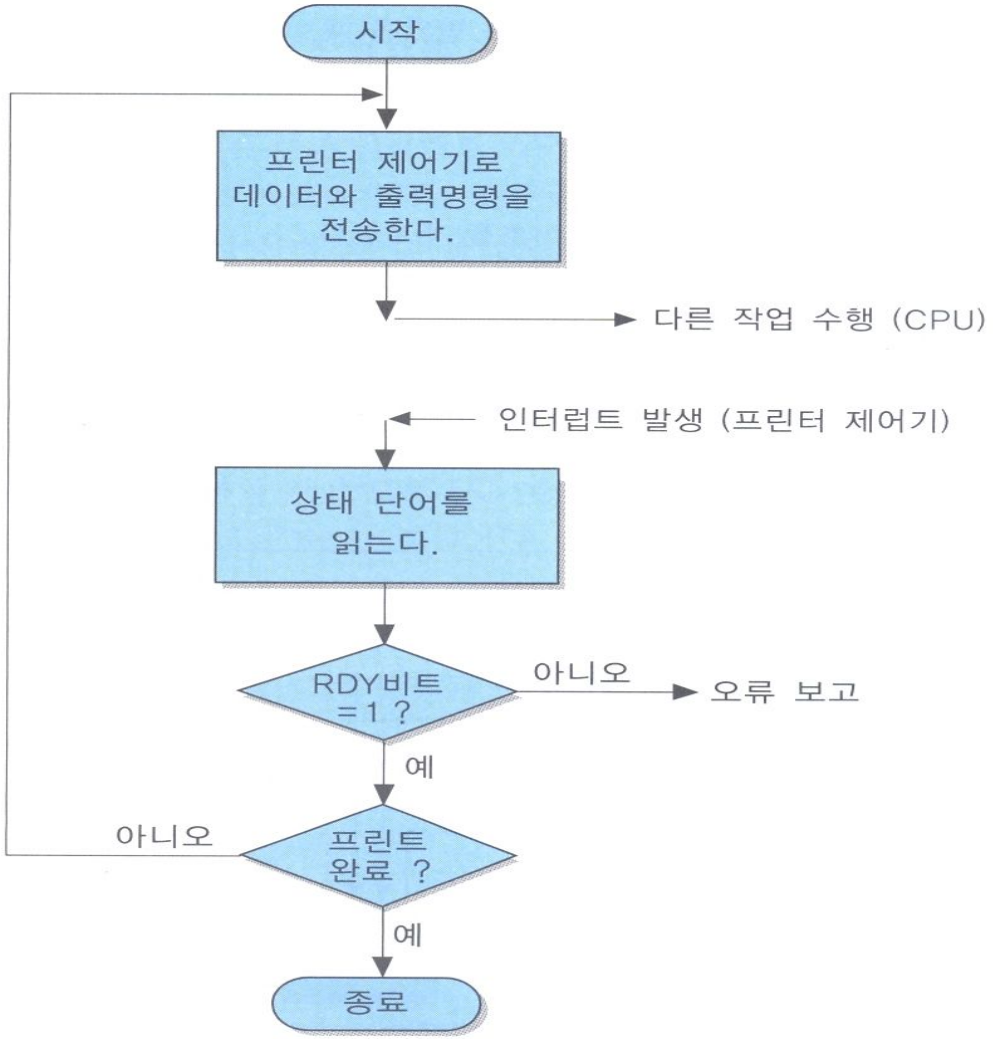
결과적으로 CPU가 프린터로 데이터를 출력하는 과정은 다음과 같음.

① CPU가 데이터와 프린트 명령을 프린터 제어기로 보냄.

② 그 데이터의 프린트가 종료되면, 제어기가 CPU로 인터럽트 요구 신호를 보냄.

③ 만약 프린트할 내용이 남아 있다면, CPU는 다음에 프린트할 데이터를 준비하여 ①번부터 반복함.

인터럽트를 이용한 I/O (2)



• 인터럽트-구동 I/O 방식을 이용한 프린터 출력 흐름도

인터럽트를 이용한 I/O (3)

(1) 다중-인터럽트 선들을 사용하는 방식

- 이 방식에서는 각 I/O 제어기와 CPU 사이에는 별도의 인터럽트 요구(interrupt request : INTR) 선과 인터럽트 확인(interrupt acknowledge: INTA) 선이 존재함.
- 아래 그림에서 I/O 제어기 2가 인터럽트를 요구하는 경우에는 다음과 같은 동작들이 순서대로 수행됨.

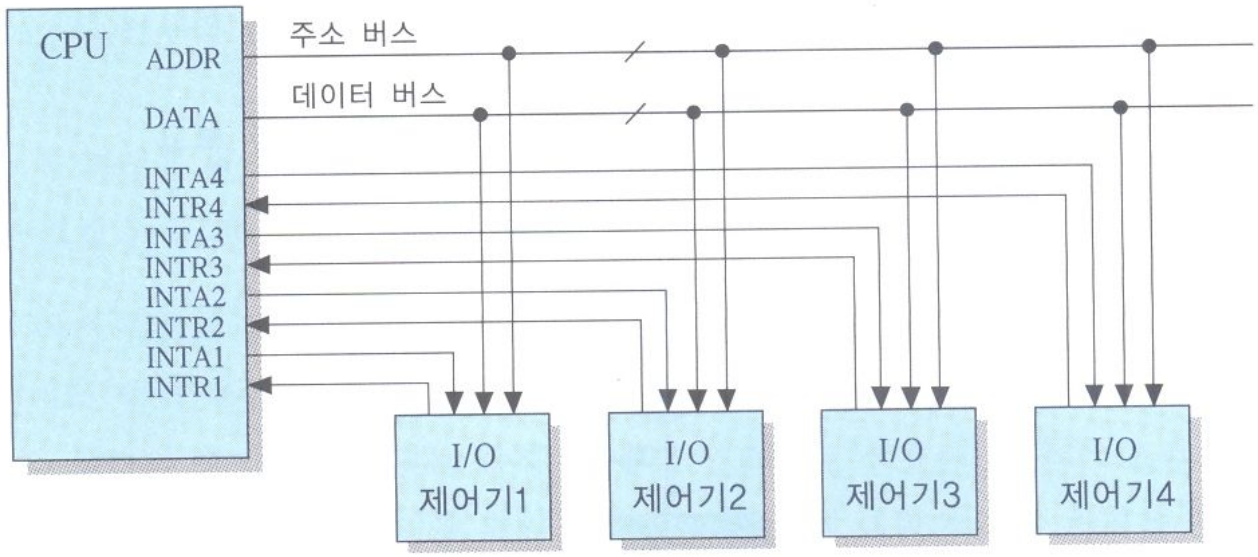
① I/O 제어기2가 INTR2 신호를 세트 함.

② CPU는 INTA2 신호를 세트 함으로써 그 제어기에게 인터럽트 요구를 인식하였음을 알리고, 인터럽트를 위한 서비스를 시작함.

③ I/O 제어기2는 INTR2 신호를 해제(0으로 리셋)함.

④ CPU도 INTA2 신호를 해제함.

인터럽트를 이용한 I/O (4)



• 다수의 인터럽트 선들을 사용하는 시스템의 구성도

- 만약 두 개 이상의 I/O 장치들이 동시에 인터럽트 요구 신호들을 보낸다면 각 I/O 장치에 대해 우선순위를 정하고, 더 높은 우선순위를 가진 장치의 인터럽트 요구부터 확인 신호를 보내고 서비스 함.

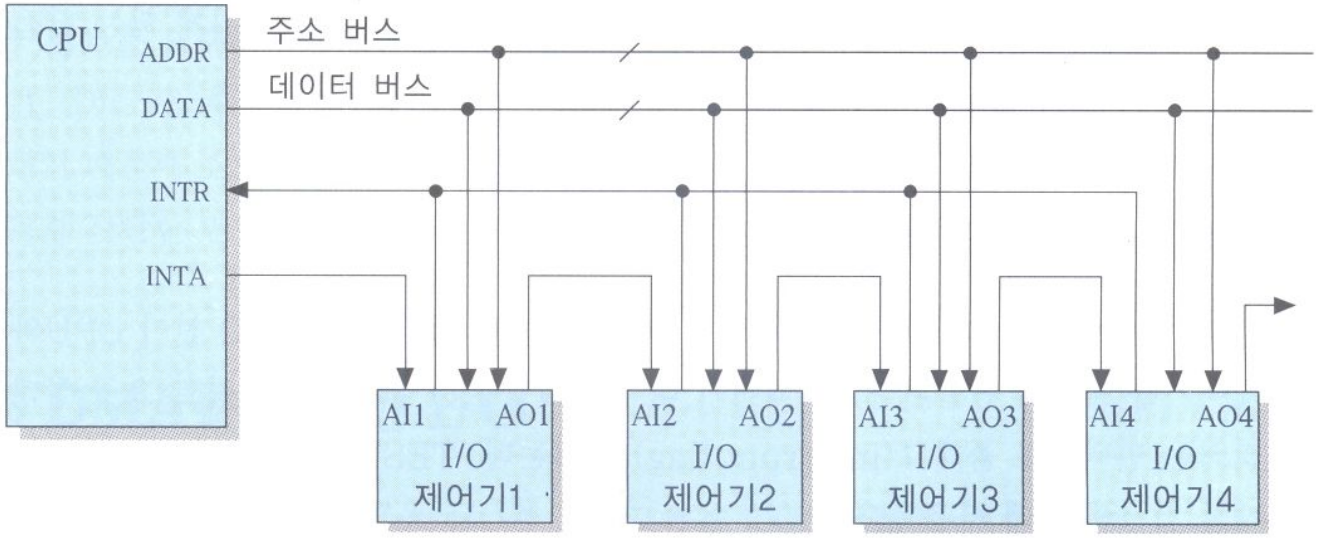
인터럽트를 이용한 I/O (5)

- 다중-인터럽트 선들을 사용하는 방식의 장점은 각 I/O 장치가 별도의 인터럽트 선을 가지고 있기 때문에 CPU가 인터럽트를 요구한 장치를 쉽게 찾아낼 수 있음.
  반면에, 이 방법은 각 I/O 장치에 대해 두 개씩의 신호 선들이 필요하므로 하드웨어가 복잡해지고, 접속 가능한 I/O 장치들의 수가 CPU의 인터럽트 요구 입력핀의 수에 의해 제한됨.
- CPU가 어떤 인터럽트 요구에 대한 서비스를 시작하는 순간에 인터럽트 플래그 (interrupt flag)를 인터럽트 불가능 상태로 세트 함으로써, 그 요구에 대한 서비스를 처리하는 동안에는 다른 인터럽트 요구가 들어오더라도 응답하지 않도록 할 수 있음.
  만약 그러한 상태로 세트 하지 않은 경우에는 더 높은 우선순위를 가진 인터럽트 요구가 들어온다면, 현재의 서비스 처리를 중단하고 새로운 인터럽트에 대해 응답해야 함.

인터럽트를 이용한 I/O (6)

(2) 데이지-체인 방식

• 이 방식에서는 모든 I/O 제어기들이 한 개의 INTR 선을 공유함.
그리고 CPU로부터 발생되는 INTA 출력 선은 가장 가까이 위치한 I/O 제어기 1의 인터럽트 확인 입력(AI1) 선으로 연결되고, 제어기1의 인터럽트 확인 출력(AO1) 선은 다음에 위치한 I/O 제어기의 AI2로 연결되며 AO2는 AI3로, AO3는 AI4로 데이지-체인 형태로 연결됨.



• 데이지-체인 인터럽트 방식의 구성도

인터럽트를 이용한 I/O (7)

- 어떤 순간에 하나 혹은 그 이상의 I/O 제어기가 인터럽트 요구를 보낸다면, INTR 선이 세트 됨.

  그러면 CPU는 일단 INTA 신호를 세트 하는데, 그 신호는 첫 번째 I/O 제어기로 보내짐.

  만약 그 제어기가 인터럽트를 요구한 상태라면,

  즉시 데이터 버스를 통해 자신의 고유 번호를 CPU로 보내게 되는데

  이 번호를 인터럽트 벡터(interrupt vector)라고 불리며,

  CPU에 의해 해당 I/O 장치를 위한 인터럽트 서비스 루틴의 시작 주소를

  결정하는 데 사용됨.

  그러나, 만약 첫 번째 I/O 제어기가 인터럽트를 요구한 상태가 아니라면,

  그 제어기는 확인 신호를 AO1을 통해 두 번째 제어기로 보냄.

  이러한 확인 신호의 전달 과정은 실제 인터럽트를 요구한 제어기에 도달할 때까지 계속됨.

  결과적으로 이 방식에서는 INTA 신호가 연결된 순서가 우선순위를 나타냄.

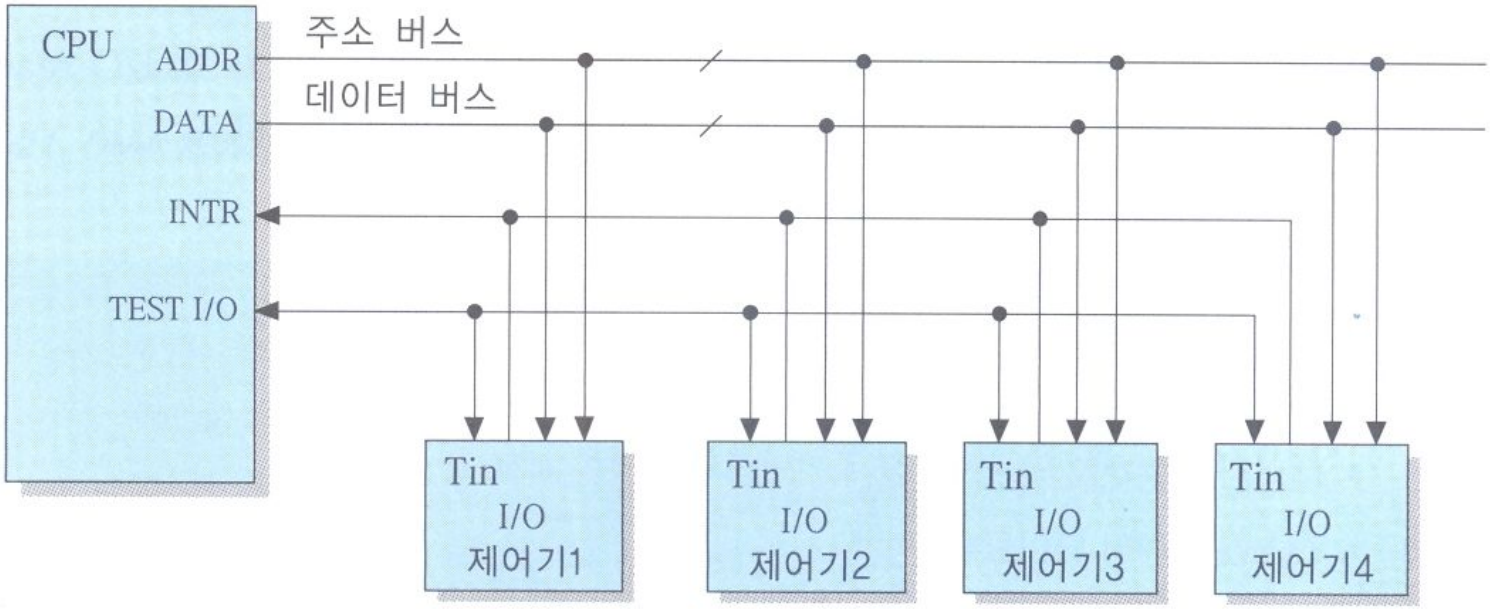
인터럽트를 이용한 I/O (8)

(3) 소프트웨어 폴링 방식

- 이 방식에서는 한 개의 TEST I/O선이 CPU와 모든 제어기들 사이에 연결됨.
  각 제어기 내에는 해당 I/O 장치의 인터럽트 요구 상태를 가리키는 인터럽트 플래그 (interrupt flag)가 있는데, TEST I/O 신호는 그 플래그의 상태를 검사하는 데 사용됨.
  이 방식에서도 인터럽트 요구(INTR) 선은 모두 I/O 제어기들에 의해 공통으로 사용됨.
  어떤 I/O 장치가 인터럽트 요구를 발생하면, 해당 제어기 내에 있는 인터럽트 플래그가 세트 됨과 동시에 공통 INTR 신호가 세트 됨.
  그러면 CPU는 즉시 인터럽트 서비스 루틴으로 분기하여, 먼저 TEST I/O 선을 이용해 첫 번째 I/O 제어기의 인터럽트 플래그가 세트 되어 있는 지를 검사함.
  만약 세트 되어 있다면, 인터럽트 검사 과정은 완료되지만, 그 플래그가 세트 되어 있지 않다면, CPU는 인터럽트를 요구한 장치를 찾을 때까지 검사 과정을 모든 제어기들에 대해 차례대로 수행함.

인터럽트를 이용한 I/O (9)

- 이 방법은 인터럽트를 요구한 장치를 프로그램을 이용하여 찾아내기 때문에 소프트웨어 폴링(software polling) 방식이라고 부름.
  이 방식은 하드웨어가 간단한 반면에, 시간이 많이 걸린다는 단점이 있음.



- 소프트웨어 폴링을 위한 시스템 구성도

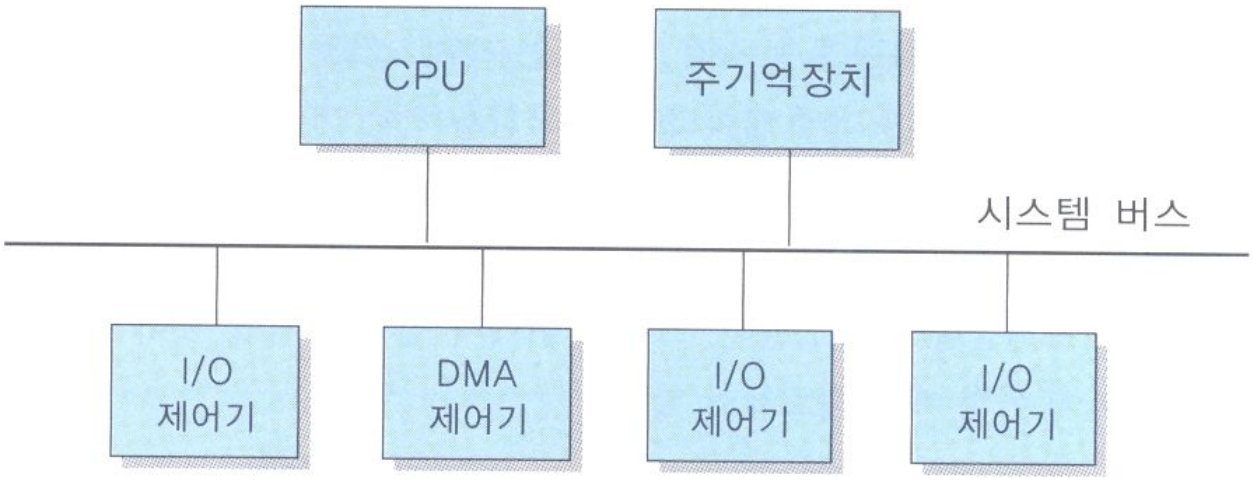
3 교시

DMA를 이용한 I/O (1)

- 인터럽트-구동 I/O가 프로그램된 I/O보다 더 효율적이기는 하지만,
  아직도 기억장치와 I/O 장치간의 데이터 이동에 CPU가 직접 개입해야 하며,
  이동되는 데이터들이 반드시 CPU를 경유해야 함.
  예를 들어, 주기억장치 내의 데이터를 I/O 장치로 이동시키는 I/O 쓰기인 경우에
  CPU가 주기억장치로부터 데이터를 한 개씩 읽어서 내부 레지스터에 적재한 다음에
  I/O 장치로 내보내고, I/O 명령도 보내야 함.
- 이러한 문제점을 해결하기 위해 직접기억장치액세스(Direct Memory Access: DMA)가
  널리 사용되고 있음.
  DMA란 CPU의 개입 없이 I/O 장치와 기억장치 사이에 데이터 전송을 수행하는 기술을
  말함.

DMA를 이용한 I/O (2)

• DMA 기술을 사용하기 위해서는 아래 그림과 같이 시스템 버스에 DMA 제어기(DMA controller)가 추가되어야 함.



• DMA 제어기가 포함된 시스템 구성도

DMA를 이용한 I/O (3)

- 주기억장치 내의 데이터를 I/O 장치로 보내거나(I/O write),
  I/O 장치로부터 데이터를 주기억장치로 읽어 들이고자 할(I/O read) 때,
  CPU는 DMA 제어기로 다음과 같은 정보들이 포함된 명령을 보냄.
  * I/O 장치의 주소
  * 연산(쓰기 혹은 읽기) 지정자
  * 데이터가 읽혀지거나 쓰여질 주기억장치 영역의 시작 주소
  * 전송될 데이터 단어들의 수
- 여기서 I/O 장치가 디스크인 경우에는 주소가 디스크 드라이브 번호, 트랙(혹은 실린더) 번호 및 섹터 번호를 모두 포함해야 함.
  CPU는 이러한 정보들이 포함된 명령을 DMA 제어기로 보낸 다음에는 다른 일을 계속함.
  즉, CPU는 I/O 동작을 DMA 제어기에 맡기고, 모든 데이터 전송 동작이 완료될 때까지 전혀 개입하지 않음.

DMA를 이용한 I/O (4)

- DMA 제어기가 DMA 동작을 수행하기 위해서는 시스템 버스를 통하여 주기억장치를 액세스 하게 됨.

  즉, CPU와 DMA 제어기가 시스템 버스를 공유하는 것임.

  그런데 DMA 제어기는 가능한 한 CPU의 정상적인 동작을 방해하지 않으면서 시스템 버스를 사용함.

  CPU가 주기억장치를 액세스 하지 않는 시간(예를 들면 CPU가 내부적으로 명령어를 해독하거나 ALU 연산을 수행하는 시간) 동안에 시스템 버스를 사용하기 때문에 DMA의 이러한 동작을 사이클 훔침(cycle stealing)이라고도 함.
- 그런데 큰 데이터 블록을 전송하는 경우에 그러한 사이클만 이용하게 되면 I/O 시간이 너무 길어지게 되므로 일반적으로는 DMA 제어기가 시스템 버스 사용을 요구하면, CPU는 현재 사이클이 끝나는 즉시 허가를 해주어서 DMA 동작이 신속하게 일어날 수 있게 해주며, 그 동안에 CPU는 기다림.

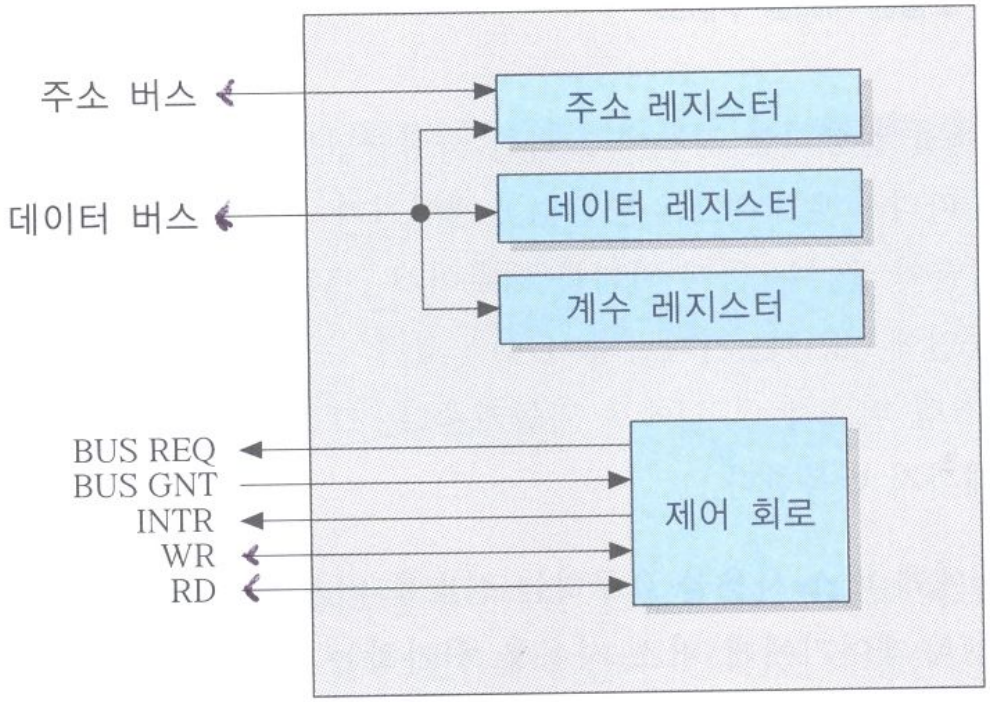
  DMA 제어기는 데이터를 한 번에 한 개씩 이동시킨 다음에 시스템 버스를 CPU에게 되돌려 줌. 그리고 그러한 동작을 모든 데이터 블록을 전송할 때까지 반복함.

DMA를 이용한 I/O (5)

- DMA 제어기가 CPU에게 버스 사용을 요구하는 신호를
  버스 요구(BUS REQ) 신호라고 하며,
  CPU가 DMA 제어기에게 버스 사용을 허가하는 신호를 버스 승인(BUS GRANT)이라고 함.
  그리고 DMA 동작이 완료되면 DMA 제어기는 CPU로 인터럽트 요구(INTR) 신호를
  보내어 그 사실을 알리며, CPU는 그에 따라 원래 프로그램을 계속 수행하게 됨.
- 예로서, 주기억장치 내의 한 데이터 블록을 디스크에 저장하기 위한 DMA 과정은
  아래와 같음.
  ① CPU가 DMA 제어기에게 명령을 보냄
  ② DMA 제어기는 CPU로 BUS REQ 신호를 보냄.
  ③ CPU가 DMA 제어기로 BUS GRANT 신호를 보냄.
  ④ DMA 제어기가 주기억장치로부터 데이터를 읽어서 디스크에 저장함.
  ⑤ 전송할 데이터가 남아 있으면, ②번부터 ④번까지를 다시 반복함.
  ⑥ 모든 데이터들의 전송이 완료되면 CPU로 INTR 신호를 보냄.

DMA를 이용한 I/O (6)

• 아래 그림은 DMA 제어기의 내부 구조를 보여주고 있는데, 먼저 DMA 명령에 포함되는 내용들을 각각 저장하는 레지스터들이 있음. 즉, I/O 장치의 주소를 저장하는 주소 레지스터와, 데이터 버퍼 역할을 하는 데이터 레지스터, 그리고 전송될 데이터 수를 저장하는 계수 레지스터(count register) 등이 있음.



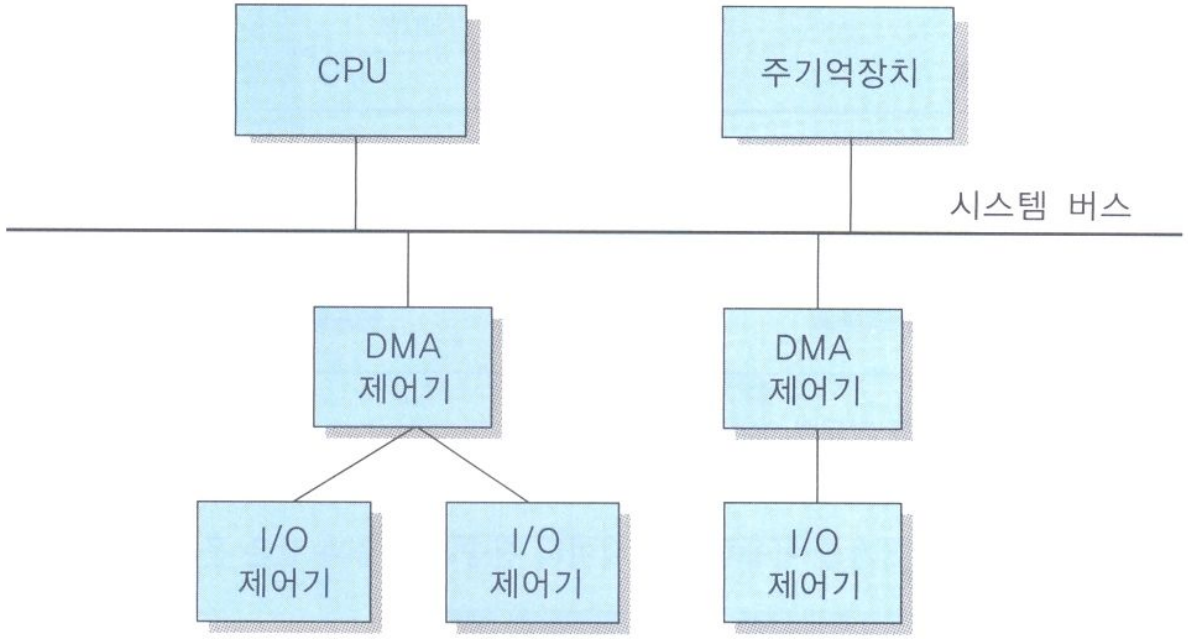
• DMA 제어기의 내부 구조

DMA를 이용한 I/O (7)

- DMA 동작은 주기억장치와 I/O 제어기 사이에 데이터를 한 번에 한 개씩 전송시키는데, DMA와 I/O가 시스템 버스에 연결되어 있으면 그때마다 시스템 버스를 두 번씩 사용함. 예를 들어, 디스크 쓰기 동작을 위하여 주기억장치의 데이터를 디스크 제어기로 전송하는 경우에, 먼저 데이터를 주기억장치로부터 DMA 제어기로 이동시키기 위하여 시스템 버스를 한 번 사용하며, DMA 제어기로부터 데이터를 디스크 제어기로 이동시키기 위하여 다시 시스템 버스를 사용하므로 1024바이트 크기의 데이터 블록을 저장한다면, 시스템 버스를 2048번 사용하게 됨.

DMA를 이용한 I/O (8)

- 그러한 문제를 해결하기 위하여 아래 구조를 고려할 수 있음. 즉, I/O 제어기를 시스템 버스에 직접 접속하지 않고 DMA 제어기에 접속하는 것임. 이럴 경우에는 1024 바이트 블록을 DMA로 전송하는 데 시스템 버스를 1024번만 사용하면 됨. 그러나, 이 구조에서는 각 DMA 제어기에 직접 접속할 수 있는 I/O 제어기의 수가 제한됨으로 다양한 종류의 많은 I/O 장치들이 포함되는 시스템에서는 여러 개의 DMA 제어기들이 시스템 버스에 접속되어야 함.



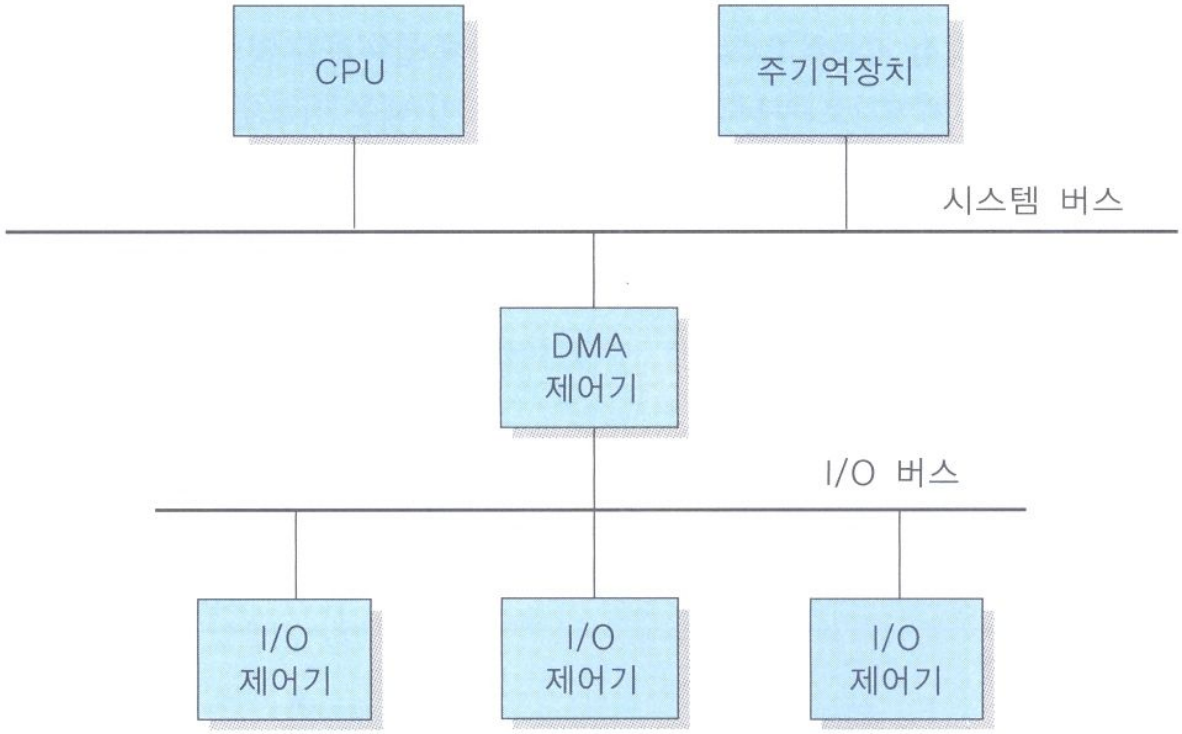
• DMA 제어기를 이용한 I/O 접속 방법

DMA를 이용한 I/O (9)

- 아래 그림은 DMA 제어기의 한 쪽은 시스템 버스와 접속되고 다른 한 쪽은 I/O 버스와 접속됨.

  그리고 I/O 버스에는 여러 개의 각종 I/O 장치들이 제어기들을 통하여 접속됨.

  이 경우에는 DMA 제어기의 내부에 I/O 인터페이스 회로가 추가되어야 함.

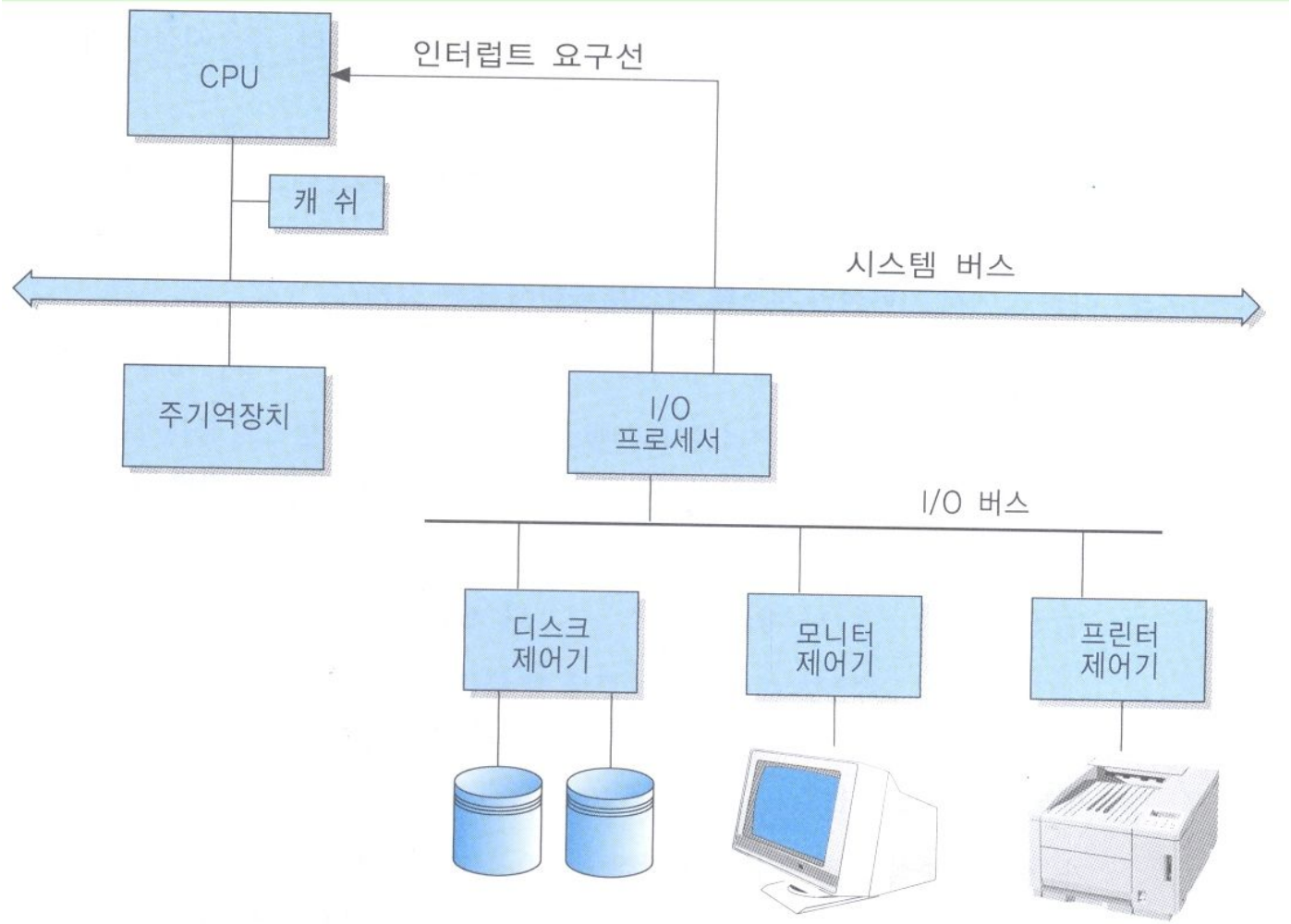


• I/O 버스를 이용한 DMA 구성도

DMA를 이용한 I/O (10)

- DMA 제어기를 이용한 I/O 데이터 전송은 다음과 같은 문제점들을 가지고 있음.
  * I/O 장치들은 종류와 속도가 다양하고 제어 방법도 복잡하기 때문에,
    간단한 구조를 가진 DMA 제어기로 지원하는 데는 한계가 있음.
  * 디스크 쓰기 혹은 읽기 동작의 경우에는 데이터 블록의 크기가
    512바이트 이상이기 때문에 그 데이터들을 버퍼링 하기 위한 내부 기억장치가 필요함.
- 이러한 문제점들을 해결하기 위해 최근의 고성능 컴퓨터시스템들에서는
  DMA 제어기를 확장시킨 I/O 프로세서(I/O processor: IOP)를 사용하고 있음.
  IOP는 I/O 채널(I/O channel)이라고도 부르는데, 다음과 같은 하드웨어들을 가짐.
  * I/O 명령어들을 실행할 수 있는 프로세서
  * 데이터 블록들을 임시 저장할 수 있는 용량의 지역 기억장치(local memory)
  * 시스템 버스에 대한 인터페이스 및 버스 마스터 회로
  * I/O 버스 중재 회로

DMA를 이용한 I/O (11)



• I/O 프로세서가 사용된 시스템의 구성도

셀프 테스트

• I/O 장치들에 대한 주소로서 기억장치 주소 공간의 일부를 할당하기 때문에 기억장치를 위한 주소 공간이 그만큼 감소하게 되는 I/O 주소 지정 방식은 무엇인가?

* 기억장치-사상 I/O 방식

해설) 기억장치-사상 I/O방식에서는 I/O를 위한 별도의 주소 공간이 없으므로 기억장치 주소 공간을 그만큼 사용함에 따라 기억장치 주소 공간이 감소하게 됨.

• 한 개의 TEST I/O선이 CPU와 모든 제어기들 사이에 연결되는 다중 인터럽트 제어 방식은 무엇인가?

* 소프트웨어 폴링 방식

해설) TEST I/O선은 I/O제어기가 인터럽트를 발생시켰는지를 소프트웨어적으로 시험할 수 있게 해줌

• CPU가 주기억장치를 액세스 하지 않는 시간 동안에 시스템 버스를 사용하는 DMA의 동작을 무엇이라고 하는가?

* 사이클 훔침

해설) DMA는 CPU가 시스템버스를 사용하지 않는 기간 동안에 시스템버스를 사용하는데 이를 사이클 훔침이라고 함.

요점 정리

- I/O 장치들은 시스템 버스에 직접 접속되지 못하며, 인터페이스가 필요한데 이러한 인터페이스 역할을 담당하는 장치를 I/O 제어기(I/O controller)라고 함.
- I/O 주소 지정 방식에는 기억장치-사상 I/O 방식과 분리형 I/O 방식이 있음.
- 인터럽트 메커니즘을 이용하면 I/O 동작이 I/O 제어기와 I/O 장치 사이에서 진행되는 동안에 CPU는 다른 작업을 처리할 수 있게 되므로 시간을 유용하게 활용할 수 있음.
- 다중 인터럽트 제어 방식에는 다중-인터럽트 선들을 사용하는 방식, 데이지-체인 방식, 소프트웨어 폴링 방식 등이 있음.
- 인터럽트-구동 I/O가 프로그램된 I/O보다 더 효율적이기는 하지만, 아직도 기억장치와 I/O 장치간의 데이터 이동에 CPU가 직접 개입해야 하며, 이동되는 데이터들이 반드시 CPU를 경유해야 하는 문제점을 해결하기 위하여 직접기억장치액세스(Direct Memory Access: DMA)가 널리 사용되고 있음.
- DMA 사용의 문제점들을 해결하기 위해 최근의 고성능 컴퓨터시스템들에서는 DMA 제어기를 확장시킨 I/O 프로세서(I/O processor: IOP)를 사용하고 있으며 IOP는 I/O 채널(I/O channel)이라고도 부름.