4 연산(Arithmetic/Logic) 명령

1. **산술연산, 논리연산, 자리이동 연산으로 분류가 가능함**

2. **논리 연산: AND, OR, XOR**
   A. 마스킹(masking)

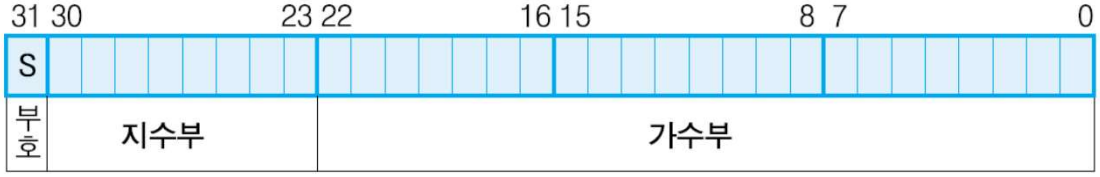
3. **회전(Rotate) 및 자리 이동(Shift):**
   A. 회전식 자리 이동(circular shift)
   B. 논리적 자리 이동(logical shift)
   C. 산술적 자리 이동(arithmetic shift)
   D. Shift 연산의 경우 x2 또는 /2 효과

4. **산술 연산: 덧셈, 뺄셈, 곱셈, 나눗셈**
   A. 정확한 동작은 값들의 인코딩 방식에 따라 달라진다 (2의 보수, 부동소수점).

4.1 부동소수점 수의 표현

1. 부동소수점 수의 표현

A. 소수점의 위치가 필요에 따라 이동하는 표현 방법을 의미하는 것



B. 지수부(Exponent part)
- 소수점의 위치를 나타냄

C. 가수부(Mantissa part)
- 유효 숫자를 나타냄 (숨은 1비트 방법으로 표현)

D. 일반적인 형태
- $N=(-1)^S \times M \times B^E$
- S: 수의 부호, M: 가수, B: 기수(Base), E: 지수

4.1 논리 연산

2. Bit 논리 연산

AND (&)

입력1	입력2	출력
0	0	0
0	1	0
1	0	0
1	1	1

OR (|)

입력1	입력2	출력
0	0	0
0	1	1
1	0	1
1	1	1

XOR (^)

입력1	입력2	출력
0	0	0
0	1	1
1	0	1
1	1	0

NOT (~)

입력1	입력2
0	1
1	0

```
    10011010
AND 11001001
------------
    10001000
```

```
    10011010
OR  11001001
------------
    11011011
```

```
    10011010
XOR 11001001
------------
    01010011
```

```
NOT 10011010
------------
    01100101
```

4.1 논리 연산 예

3. 논리 연산 응용

- AND 연산 (&)
  - 10101010의 상위 4비트를 모두 0으로 만드는 연산은? 10101010 & 00001111
  - 10101010의 최상위비트가 1인지 0인지 판단하는 연산은? 10101010 & 10000000
- OR 연산 (|)
  - 10101010의 상위 4비트를 모두 1로 만드는 연산은? 10101010 | 11110000
- XOR 연산 (^)
  - 10101010을 보수로 만드는 연산은? 11111111 ^ 10101010

4.1 논리 연산 예

3. 논리 연산 응용

1. AND 연산

- 10101010의 상위 4비트를 모두 0으로 만드는 연산은? 10101010 & 00001111
- 10101010의 최상위비트가 1인지 0인지 판단하는 연산은? 10101010 & 10000000
- 10101010의 최하위비트가 1인지 0인지 판단하는 연산은? 10101010 & 00000001

2. OR 연산

- 10101010의 상위 4비트를 모두 1로 만드는 연산은? 10101010 | 11110000
- 10101010의 최하위비트를 1로 만드는 연산은? 10101010 | 00000001

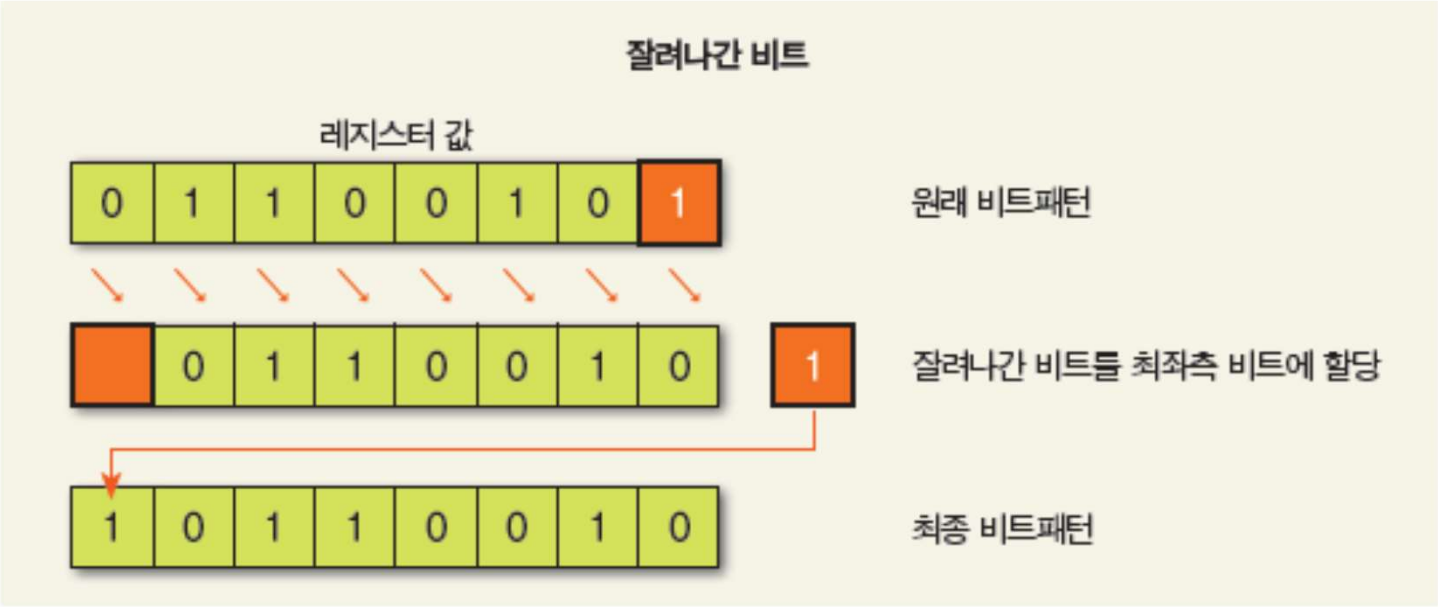
3. XOR 연산

- 10101010을 보수로 만드는 연산은? 11111111 ^ 10101010

4.2 자리이동(Shift) 연산

1. 회전식(Rotate Left/Right) 자리이동: 잘려나간 비트를 반대쪽에 삽입하는 방법

Shift & Rotate



Rotate Right

0110 0101

1011 0010

Rotate Left

0110 0101

11 0010 10

4.2 자리이동(Shift) 연산

2. 논리적 자리이동: 잘려나간 비트를 항상 0으로 채우는 방법

Logical Shift **Right**

0110 0101

0011 0010

Logical Shift **Left**

0110 0101

11 0010 10

4.2 자리이동(Shift) 연산-1

3. 산술적 자리이동: 부호 비트는 변경되지 않도록 하여 자리이동 방법

좌측 자리 이동

x2

00000100 – 4
00001000 – 8
00010000 – 16

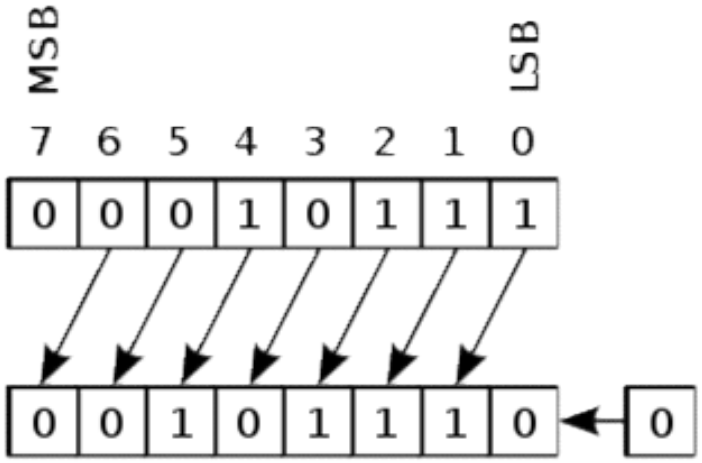
우측 자리 이동

/2

00000100 – 4
00000010 – 2
00000001 – 1

4.2 자리이동(Shift) 연산-2

3. 산술적 자리이동: 부호 비트는 변경되지 않도록 하여 자리이동 방법



23(10)(00010111(2)) - 좌측 산술 시프트의 예. 밀려서 새로 생기는 빈칸은 0으로 채워 넣는다.
연산의 결과는 **23**에 **2**를 곱한 수인 46(10) (00101110(2))이 됨.

(a<<3) + (a<<1)

Multiplication by left shift - The result of a Left Shift operation is a multiplication by 2.

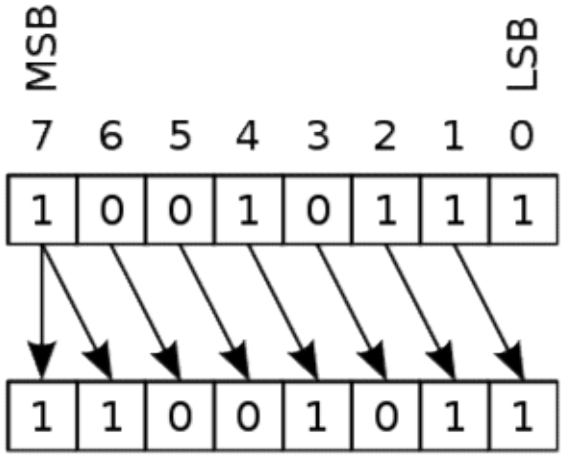
Example-1: the decimal number **2** represented as 4 bit binary number **0010**. By **shifting** in to the **left** with one position we get **0100** which is **4** in decimal. If we **shift it once more** we get binary value **1000** which is 8 in decimal representation.

Example-2: For unsigned representation, when the **first "1"** is **shifted out** of the left edge, the operation has **overflowed**. The result of the multiplication is larger than the largest possible.

Shifting left on signed values also works, but **overflow occurs** when the **most significant bit changes** values (from 0 to 1, or 1 to 0).

4.2 자리이동(Shift) 연산-3

3. 산술적 자리이동: 부호 비트는 변경되지 않도록 하여 자리이동 방법



23(10)(00010111(2)) - 우측 산술 시프트의 예. 부호가 정의된 2진법 정수(Signed Integer)에서 산술 시프트 경우 최상위 비트(MSB)이자 부호비트(Sign bit)는 그대로 유지(Copy)되며, 한 칸씩 우측으로 이동. **연산의 결과는 23을 2로 나누고** 소수점을 버린 결과인 **11(10)** (00001011(2))이 나옴. – 그림이 아님

Division by right shift - The result of a Right Shift operation is a division by 2.

Example-1: If we have the binary number 01110101 (117 decimal) and we perform arithmetic right shift by 1 bit. we get the binary number 00111010 (58 decimal). So we have divided the original number by 2.

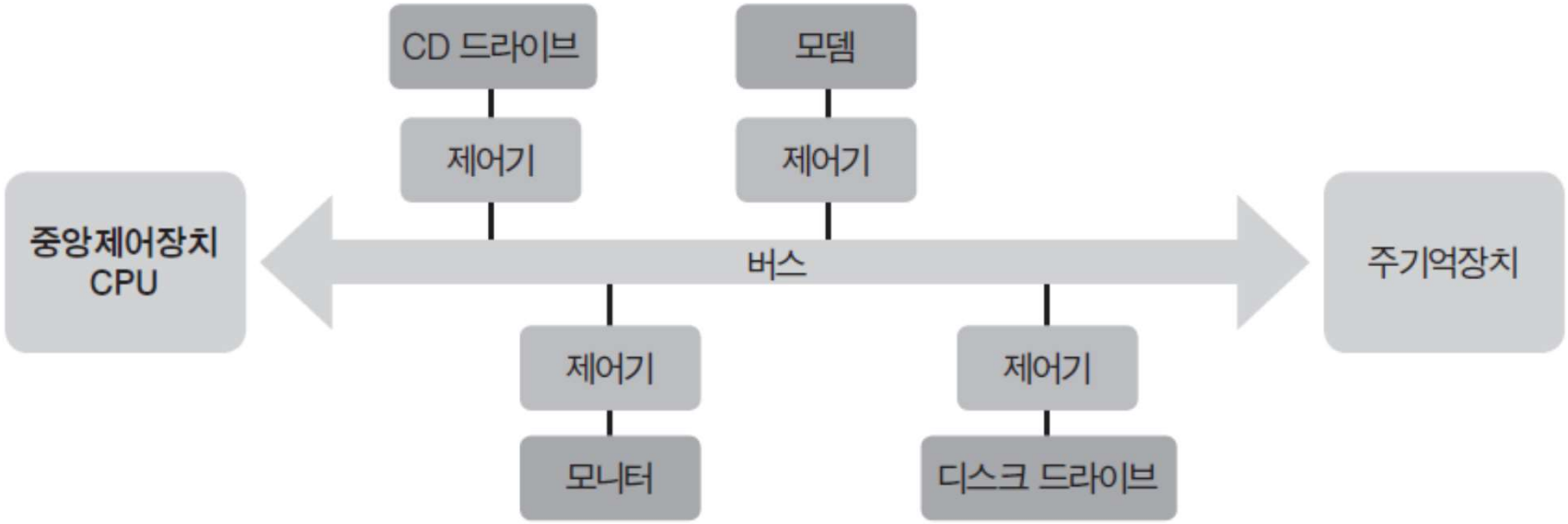
Example-2: If we have the binary number 1010 (-6 decimal) and we perform arithmetic right shift by 1 bit. we get the binary number 1101 (-3 decimal).
+6은 0110이며, **-6**은 0110(+6) -> 1's 보수 1001 -> +1, 2's 보수 **1010**을 **arithmetic right shif(shift right)**하면 2's 보수 1010(-6)을 1 비트 shift right **1101** -> MSB를 유지(복사)하며, 이 경우는 음수(-)를 나타내고, 2's 보수는 1101은 1's 보수 1100이며 이는 **-3**의 음수 표현 임.

5.1 다른 장치와의 통신

1. **제어기(Controller): 컴퓨터와 장치와의 통신을 처리하는 중개 장치**
   A. 각 장치유형마다 전용 제어기가 존재함
   B. 범용 제어기: USB와 파이어와이어(FireWire)

2. **포트(Port): 장치를 컴퓨터에 연결하는 지점**

3. **장치간 연결 도식**

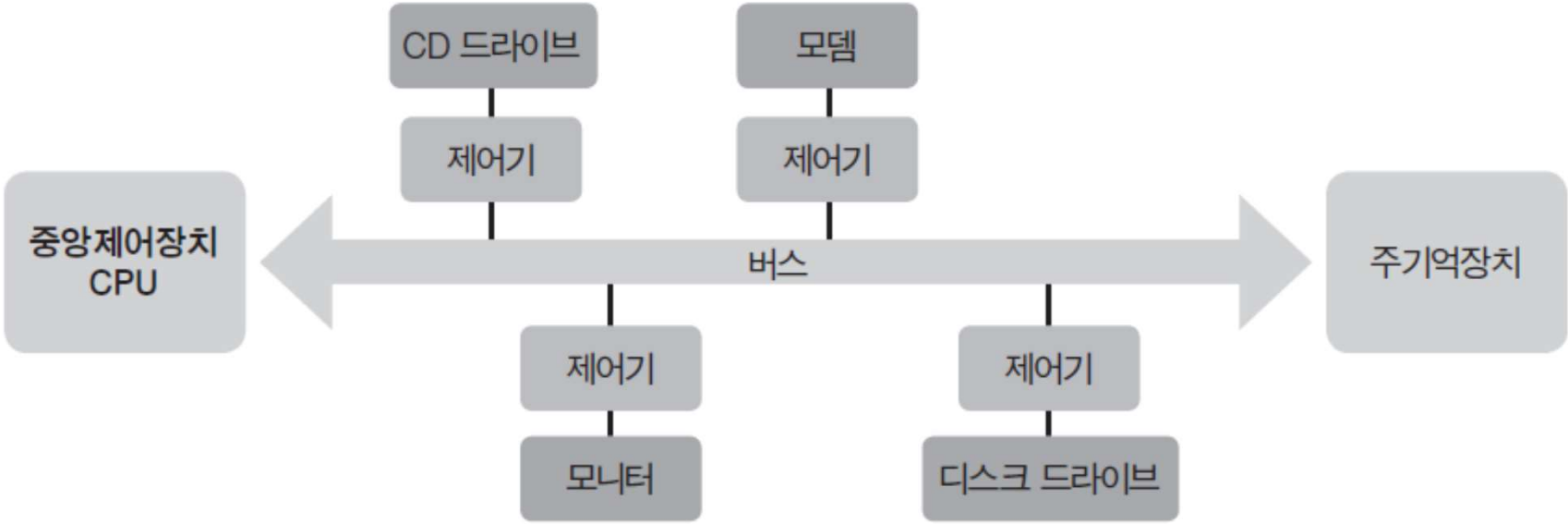


5.2 CPU와 제어기 사이의 통신 방법-1

1. 제어기 전용명령을 사용하는 방법

A. 각 제어기마다 비트패턴을 지정하여 통신하는 방법

B. 단점?
제어기 전용 명령이 필요
제어기가 다수인 경우 **CPU 설계가 복잡해지는 한계**가 있음



5.2 CPU와 제어기 사이의 통신 방법-2

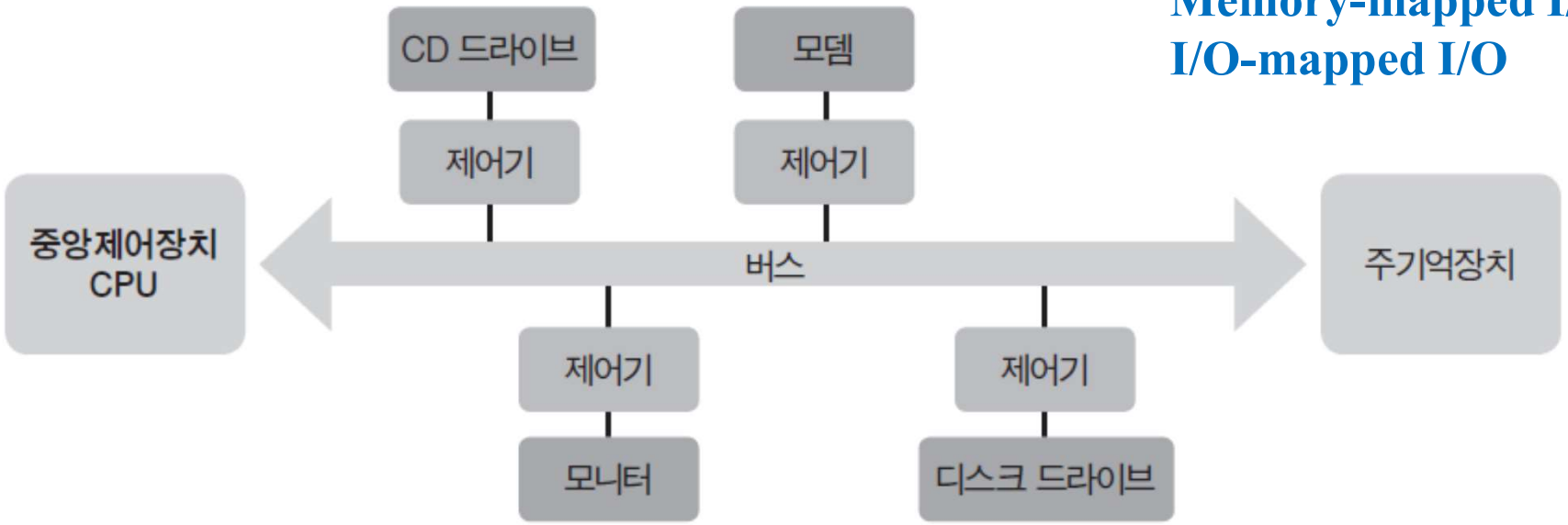
2. 주기억장치 제어명령을 사용하는 방법

A. 주기억장치와의 통신 명령을 이용하여 각 제어기와 통신하는 방법으로, 특정 주소를 지정하여 활용함
(주기억장치는 해당 주소를 무시하도록 설계함)

B. 입출력 장치들이 각기 다른 메모리 위치에 나타나는 것과 같이 동작하므로 **메모리 사상 입출력(Memory-mapped I/O)**라고 함

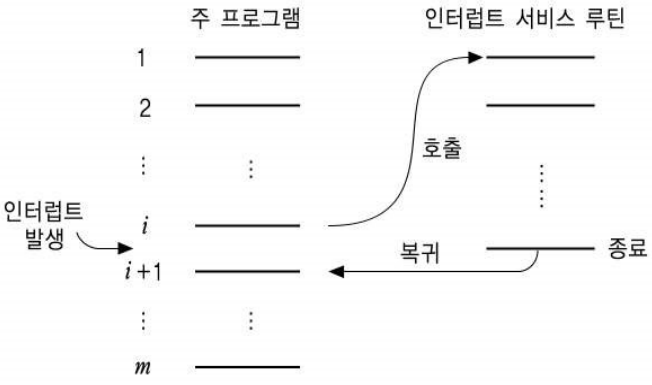
C. 장점? 기계어 명령이 간단하게 표현됨

Memory-mapped I/O
I/O-mapped I/O



5.3 다른 장치와의 통신-1

1. **DMA(Direct Memory Access):**
   제어기가 버스를 경유하여 직접 주기억장치에 접근 -> 뒤 Section 6.2에서
2. **폰노이만 병목현상:**
   A. 불충분한 버스 속도로 인한 성능 저하 현상
   B. 메모리 접근 vs 디스크 접근 속도: CPU 활용률 저하
3. **핸드셰이킹(Handshaking):**
   컴퓨터 구성요소 사이의 데이터 전송 조정 과
4. **인터럽트(Interrupt):**
   시스템의 성능향상을 위하여 필요함
   예: 운영체제의 준비 – 실행 – 대기과정을 통한 멀티테스킹



처리 방식

1. **Interrupt** 방식 – CPU 외부의 H/W S/W적인 요구에 의해서 정상적인 프로그램의 실행순서 변경해 보다 시급한 작업을 먼저 수행하고 후에 원래의 Pgm으로 복귀하는 방식.
2. **폴링(Polling)** 방식 - 컴퓨터 또는 단말 제어 장치 등에서 여러 개의 단말 장치에 순차적으로 송신의 유무를 문의하고, 송신이 있을 경우에는 그 장치에 송신을 하도록 하며, 없을 때에는 다음의 장치에 대하여 문의하게 되는 전송 처리(제어) 방식.

5.3 다른 장치와의 통신-2

1. 병렬 통신(Parallel communication):
   다중 통신 경로를 사용하여 비트들을 동시에 전송한다.
   예) 컴퓨터의 프린터 포트

2. 직렬 통신(Serial communication):
   단일 통신 경로를 사용하여 한 번에 한 비트씩 전송한다.
   예) 컴퓨터의 시리얼 포트 (휴대폰 USB 연결 등으로 확인 가능)

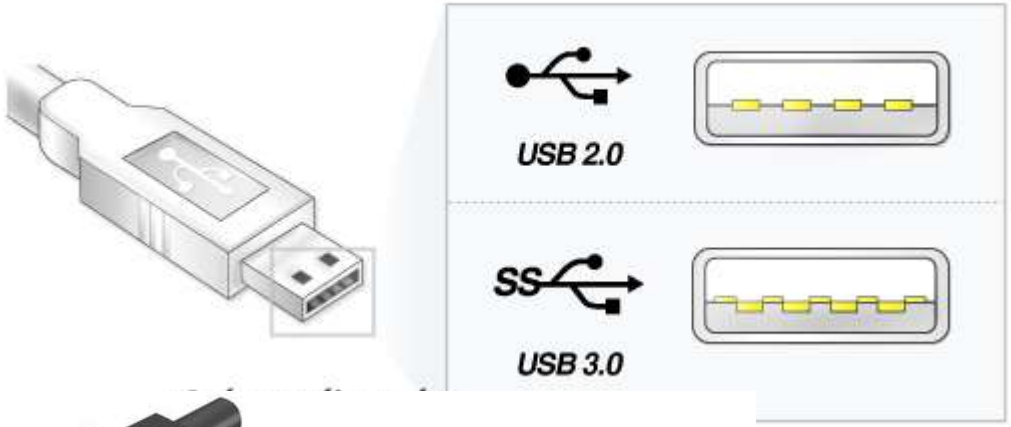
USB - Universal Serial Bus

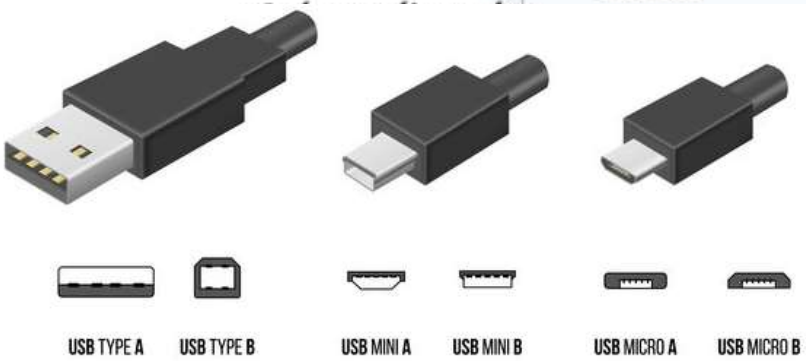
5.3 다른 장치와의 통신-3

USB - Universal Serial Bus

	USB 3.0	USB 2.0	USB 1.1
최고 속도	5000 Mbps	480 Mbps	12 Mbps
전송 속도	빠름 ◀ ▶ 느림		
호환 여부	다른 버전의 기기끼리 호환은 가능하지만 하위 버전과 연결시 속도는 하위버전에 맞춰짐		

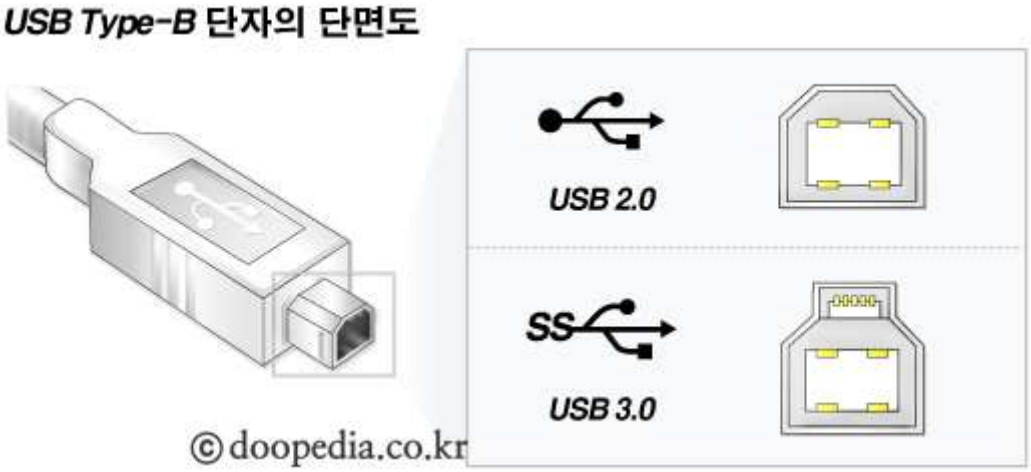
USB Type-A 단자의 단면도





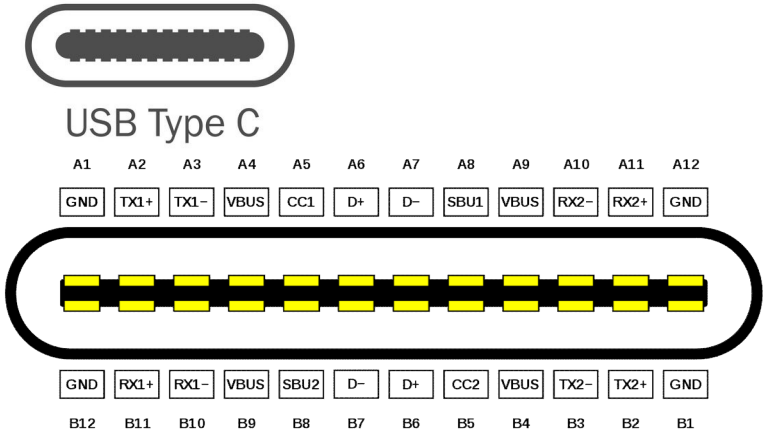
Type-A - 좌우로 긴 납작한 직사각형 형태를 갖고 있으며 전력과 데이터 모두 전송이 가능

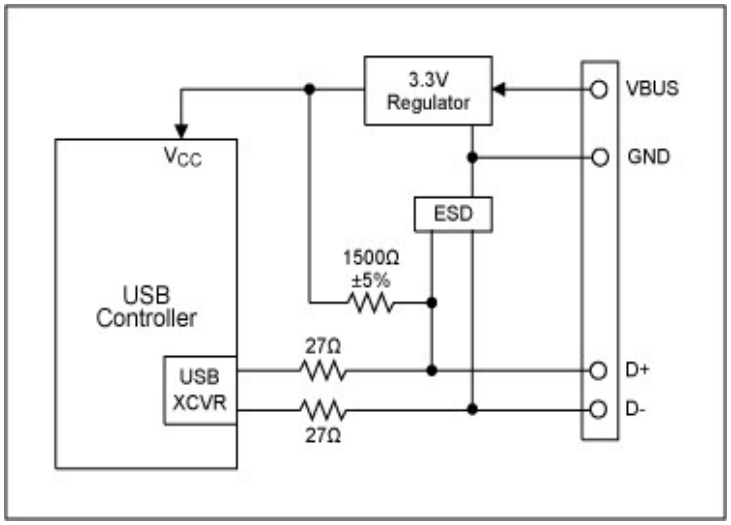
5.3 다른 장치와의 통신-4



Type-B - 정사각형 형태로 한쪽 모서리가 경사진 모양

Type-C - Type-A에 비해 크기가 작으며, 앞뒤의 구분이 없어 연결이 쉽다





USB peripheral controller

5.3 다른 장치와의 통신-5

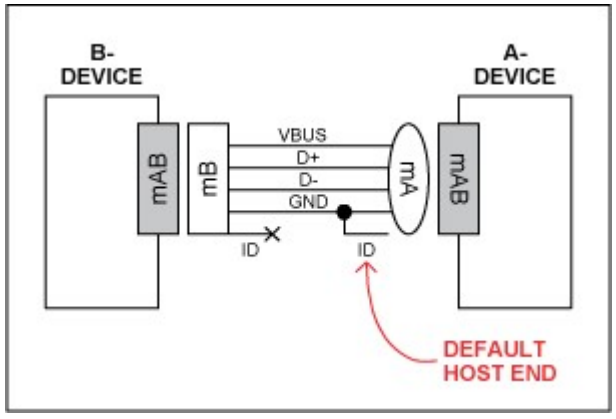
USB OTG

1. USB On-the-Go (OTG) allows two USB devices to talk to each other without requiring the services of a personal computer.

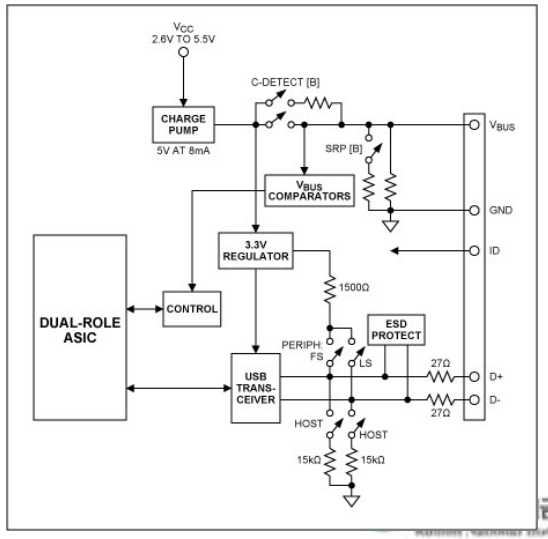
2. Although OTG appears to add "**peer to peer**“ connections to USB, it does not. USB OTG retains **the standard USB host/peripheral model-a single host talks to USB peripherals**.

3. OTG introduces **the dual-role device(DRD) capable of functioning as either host or peripheral**. The magic of OTG is that a host and peripheral can exchange roles if necessary.

4. The OTG device receiving the **grounded ID pin** is the default A-Device(host); the device with the floating ID pin is the default B-Device(peripheral).



Fifth ID pin determines default host



USB OTG Transceiver

5.4 데이터 전송 속도

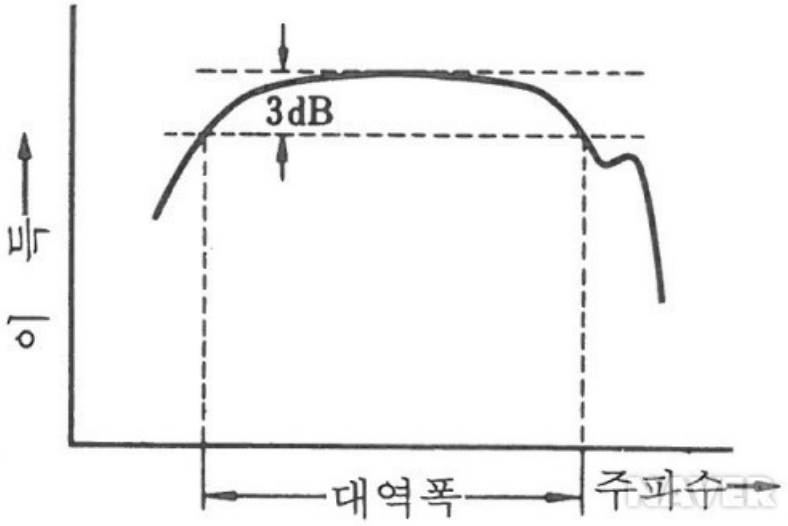
1. 측정 단위

A. Bps: Bits per second
B. Kbps: Kilo-bps (1,000 bps)
C. Mbps: Mega-bps (1,000,000 bps)
D. Gbps: Giga-bps (1,000,000,000 bps)

2. 대역폭(bandwidth): 최대 전송 속도

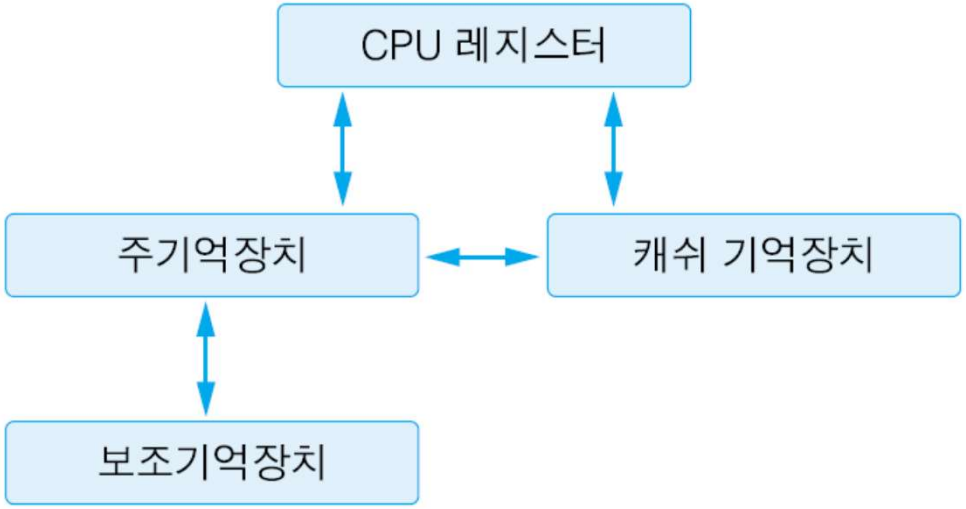
Bandwidth(Band-width)

1.(주파수의) 대역폭
2.대역폭, 띠 너비(컴퓨터 네트워크나 인터넷이 특정 시간 내에 보낼 수 있는 정보량. 흔히 초당 비트로 측정됨)

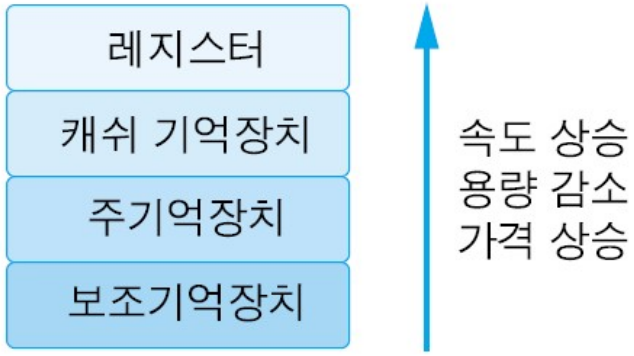


6.1 기타 컴퓨터 구조-1

1. 기억장치의 구성



2. 기억장치 계층



6.2 기타 컴퓨터 구조-2

1. 캐쉬(Cache, Cache Memory)

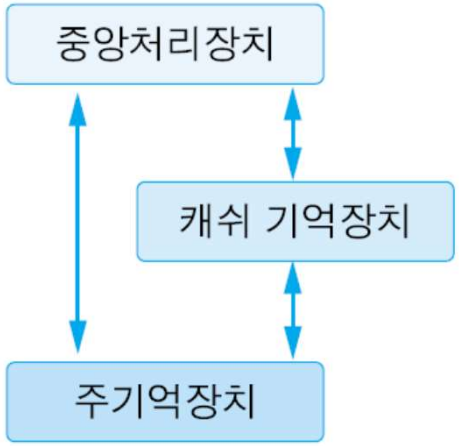
A – (고속) 기억장치, CPU와 주기억장치 사이에 위치한 접근속도(Access Time)가 빠른 소규모 기억장치.

B – 주기억장치 내용의 일부를 복사하여 가지고 있음.

CPU에서 메모리 접근 속도 향상을 위해서 먼저 Cache에 접근해서 Cache에 있으면 이를 이용하고,
없으면(Cache 접근 후에) 다시 메모리에 접근함.

적중률(hit ratio)

$$\text{적중률} = \frac{\text{캐쉬에 적중된 횟수}}{\text{기억장치 총 접근 횟수}}$$



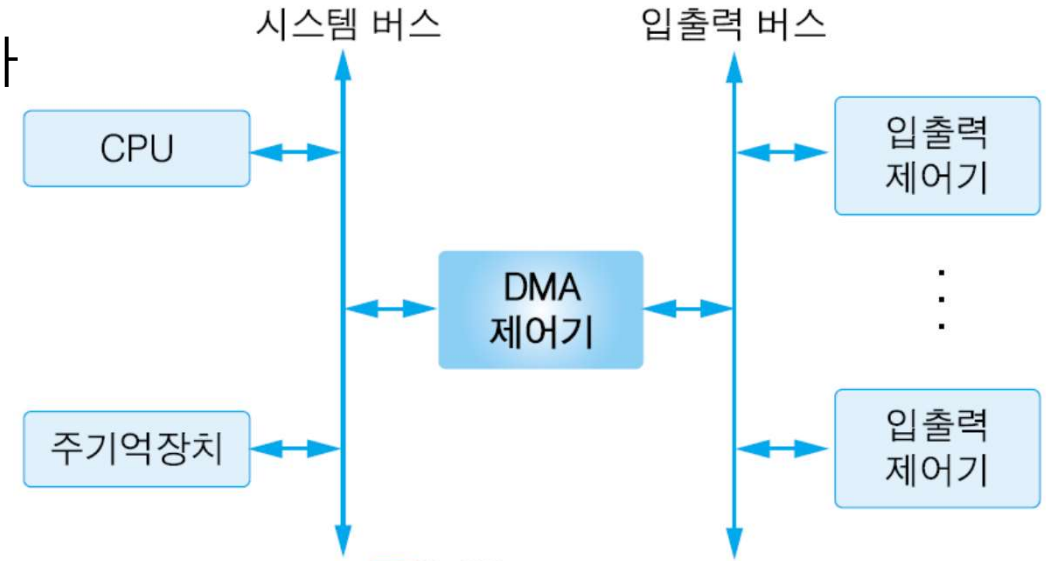
6.2 기타 컴퓨터 구조-3

1. DMA(Direct Memory Access)에 의한 입출력 방식

A. 프로그램에 의한 입출력 방식과 인터럽트에 의한 입출력 방식의 단점을 보완하기 위한 것

B. CPU의 레지스터를 **거치지 않고** 직접 주기억장치와 입출력장치 사이에서 데이터 전송이 이루어짐 – **DMA 제어기**로 I/O장치의 주소, 연산 지정자, 데이터가 읽혀지거나 쓰여질 주기억장치 영역의 주소, 전송될 데이터의 수량에 대한 명령을 전송하여 **DMA 제어기가 직접 처리하는 방식**.

C. CPU가 **유휴(Idle)** 상태가 될 수 있음

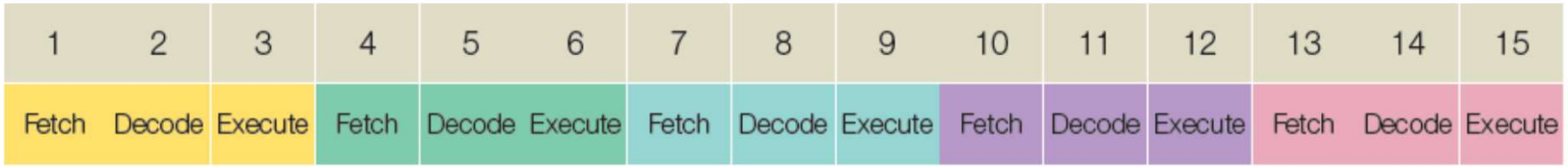


6.3 기타 컴퓨터 구조 : 파이프라이닝

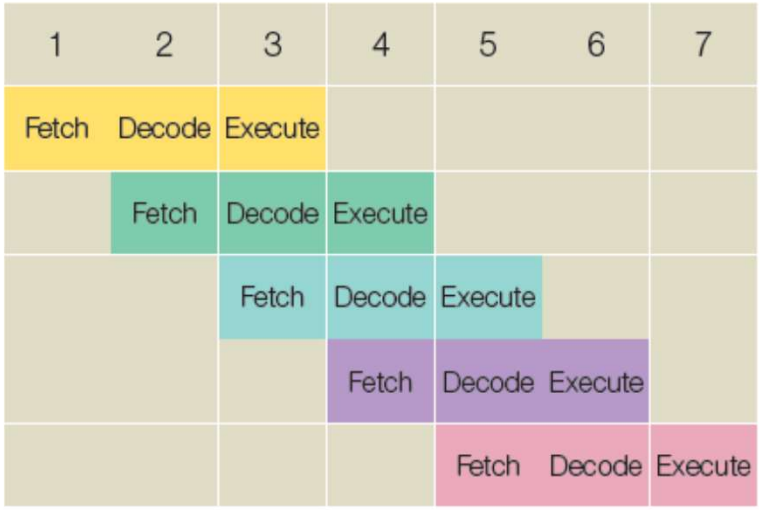
1. 처리율(Throughput) 개선을 위한 기술(1)

A. 파이프라이닝(**pipelining**): 기계 주기의 단계들을 중첩하여 성능 향상
분기 명령시 효율성 없음

without pipelining (15 clocks)



with pipelining (7 clocks)



6.3 기타 컴퓨터 구조 : 플린 분류

2. 처리율(Throughput) 개선을 위한 기술(2)

A. 병렬 처리(parallel processing): 여러 개의 프로세서를 동시에 사용한다.

- SISD: 병렬 처리하지 않음
- MIMD: 여러 개의 프로그램이 각자 다른 데이터를 사용하여 수행됨
- SIMD: 동일한 프로그램이 여러 데이터에 적용됨

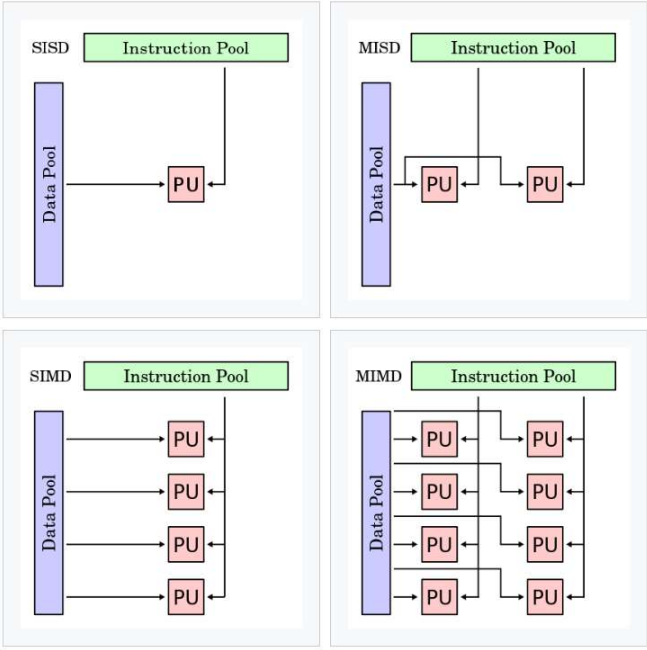
3. 멀티프로세서(Multi-processor) vs 멀티프로세싱 (Multi-processing)

A. 최근 싱글코어 → 듀얼코어 → 쿼드코어

B. 도스 vs 윈도우/유닉스

플린 분류 **(Flynn's taxonomy)** - 마이클 플린이 1966년에 제안한 컴퓨터 아키텍처 분류

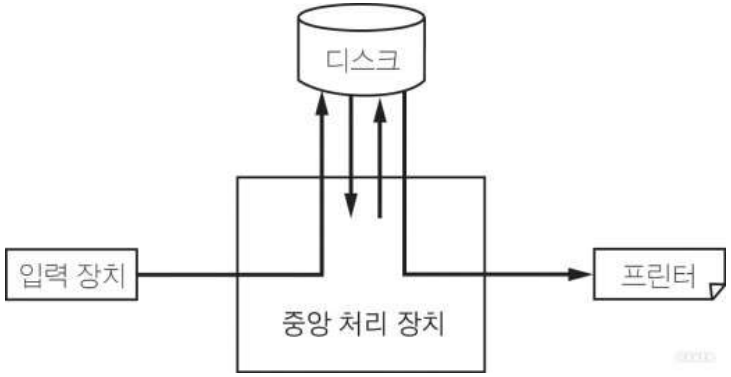
SISD (Single Instruction, Single Data stream)
SIMD (Single Instruction, Multiple Data streams)
MISD (Multiple Instruction, Single Data stream)
MIMD (Multiple Instruction, Multiple Data streams)



6.4 기타 질문들

1. 시스템 성능 향상을 위한 방법들

A. 캐쉬 (Cache) 사용
B. DMA (Direct Memory Access) 사용
C. MMX (MultiMedia Extension) Instructions
D. 스풀링 (Spooling)
E. 인터럽트 활용
F. **가상메모리 (Virtual Memory)**



스풀링 [Spooling]
주변 장치와 컴퓨터 처리 장치 간에 데이터를 전송할 때 **처리 지연을 단축**하기 위해 **보조 기억 장치를 완충 기억 장치**로서 사용하는 것.
Simultaneous Peripheral Operation Online을 의미

→ **구체적인 내용은 컴퓨터 구조 강의에서 학습**