

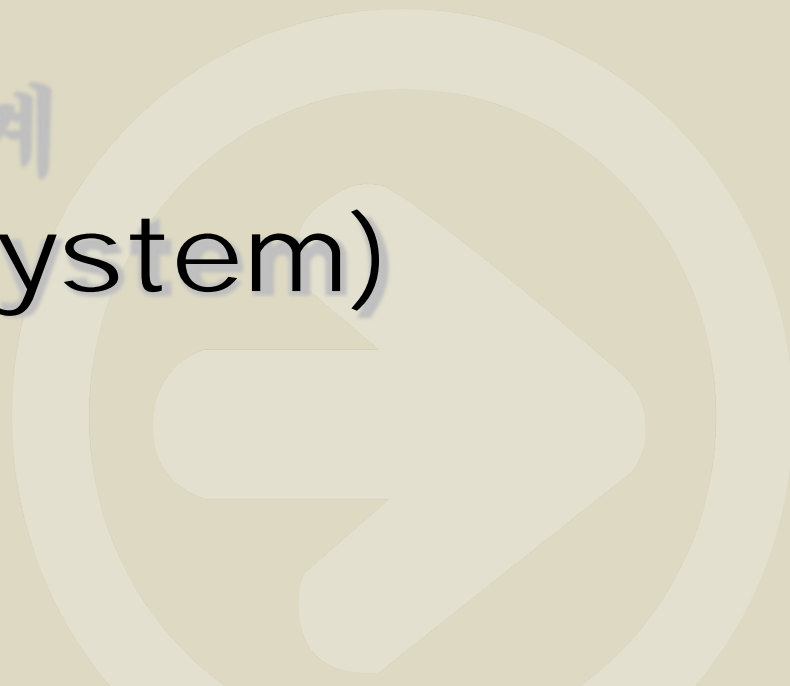


## 제 4 장

# 변수와 데이터 유형 2부

## 수 체계

# (Number System)





# 밑수(base) 변환

- ❑ 컴퓨터에서 모든 정보는 2진수로 표현된다
- ❑ C 프로그램에서는 10진수, 8진수, 16진수 사용 가능
- ❑ 밑수 변환이 필요 – Euclid 호제법
- ❑ 밑이 r인 수의 의미
  - ▣ 예를 들어,  $687_{10} = 6*10^2 + 8*10^1 + 7*10^0$



## 정수의 변환

□ 10진수를 다른 진수(예를 들면 2진수)로 변환 : Euclid 호제법

2	687	1
2	343	1
2	171	1
2	85	1
2	42	0
2	21	1
2	10	0
2	5	1
2	2	0
	1	

$$687 = b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \dots b_1 \times 2^1 + b_0 \times 2^0$$

에서  $b_{n-1}, b_{n-2}, \dots, b_1, b_0$  와  $n$ 을 구해야 한다

1) 양변을 2로 나눈다

2)  $687/2 =$

3)  $343 \times 2 + 1$  //  $b_0 = 1$

4)  $(171 \times 2 + 1) \times 2 + 1$  //  $b_1 = 1$

5)  $((85 \times 2 + 1) \times 2 + 1) \times 2 + 1$  //  $b_2 = 1$

6) .

7) .



# s 진수를 d 진수로 변환

- ❑ STEP-1: s진수를 먼저 10진 수로 변환
- ❑ STEP-2: 변환된 10진수를 d진수로 변환
- ❑ 예)  $687_9$  를 2진수로 변환 : 먼저 10진수로 변환하고, 2진수로 변환

$$687_9 \rightarrow D_{10} \rightarrow B_2$$

$$D_{10} = 6 \times 9^2 + 8 \times 9^1 + 7 \times 9^0 = 565_{10} \rightarrow 2\text{진수 변환은 앞에서 한 것과 동일한 방법}$$

9		565	←	7
9	X	62	←	8
	X	6		+

$$D_{10} = 6 \times 9^2 + 8 \times 9^1 + 7 \times 9^0$$

$$= \{6 \times 9 + 8\} \times 9 + 7$$

62

565



## 소수의 변환

- 소수의 진법 변환은 어떻게 ?
- 예)  $0.6875_{10}$ 을 2진수로 변환하면 ? 8진수로 변환하면?
- $0.1011_2$

2	x	0.6875	1
2	x	0.3750	0
2	x	0.7500	1
2	x	0.5000	1
		0	

- 1) 0.6875에 2를 곱하면 1.3750 --
  - 2) 정수 부분 1을 오른쪽 칸에, 나머지 0.3750을 아래칸에 쓰고
  - 3) 0.3750에 대해서 동일한 방법으로 2를 곱하여 정수 부분을 오른쪽 칸에, 나머지를 아래칸에 쓰고
- ·  
·
- 이러한 과정을 나머지 부분이 0 이 될 때까지 한다.

그리고 위부터 2진수를 소수점을 붙여 기술한다 →  $0.1011$



□ 그러면  $0.0101_2$ 를 10진수로 변환하면?

2	0.3125	0
2	0.625	1
2	0.25	0
2	0.5	1
	0	

- 1) 먼저  $0.0101_2$ 를 위에서부터 소수부분을 기록한다
- 2) 맨 아래부터 시작하여....  
오른쪽 수와 아래쪽 수를 더해서 2로 나눈 결과를 위칸에 적는다 (즉,  $(1+0)/2 = 0.5$ )
- 3) 다시 오른쪽 수와 아래쪽 수를 더해 2로 나눈 결과를 위칸에 적는다 (즉,  $(0+0.5)/2 = 0.25$ )

..  
오른쪽 칸에 수가 없을 때까지 반복한다

- 1) 1과 0을 더해서 2로 나눈 나머지 0.5를 위칸에 적는다
- 2) 0과 0.5를 더해서 2로 나눈 결과 0.25를 위칸에 적는다
- 3) 1과 0.25를 더해서 2로 나눈 결과 0.625를 위칸에 적는다
- 4) 0과 0.625를 더해서 2로 나눈 결과 0.3125를 위칸에 적는다
- 5) 최종 결과는  $0.3125_{10}$  이다



# 양수와 음수의 표현 - 보수

- ❑ 보수(complement)의 개념을 이용하여 양수와 음수를 표현
- ❑  $r$  진법에서의 보수 유형은 2가지가 존재
  - ❑  $r$ 's complement ( $r$ 의 보수)
  - ❑  $(r-1)$ 's complement ( $(r-1)$ 의 보수)
  - ❑ 예) 2진법에서는 2's complement와 1's complement가 존재
  - ❑ 예) 10진법에서는 10's complement와 9's complement가 존재
- ❑  **$r$ 's complement =  $(r-1)$ 's complement + 1**
- ❑  **$(r-1)$ 's complement**
  - ❑ 같은 자리의 수끼리 합이  **$(r-1)$** 이 되는 두 수의 서로 관계
  - ❑ 예) 10진수 687의 9's complement인 687' 는 312이다
- ❑  **$r$ 's complement**
  - ❑  $(r-1)$ 's complement + 1
  - ❑ LSB로부터 시작하여 처음 0이 아닌 자리 수에 대해서만 합이  $r$ 이 되도록 하고 그 다음 자리 부터는 합이  $(r-1)$ 이 되도록 만든 수



## 2진 정수의 표현

- ❑ 부호와 절대값 (Signed-Magnitude Notation)
- ❑ 부호화 1의 보수 (Signed 1's Complement Notation)
- ❑ 부호화 2의 보수 (Signed 2's Complement Notation)
- ❑ 초과 표기 (Excess Notation) 등



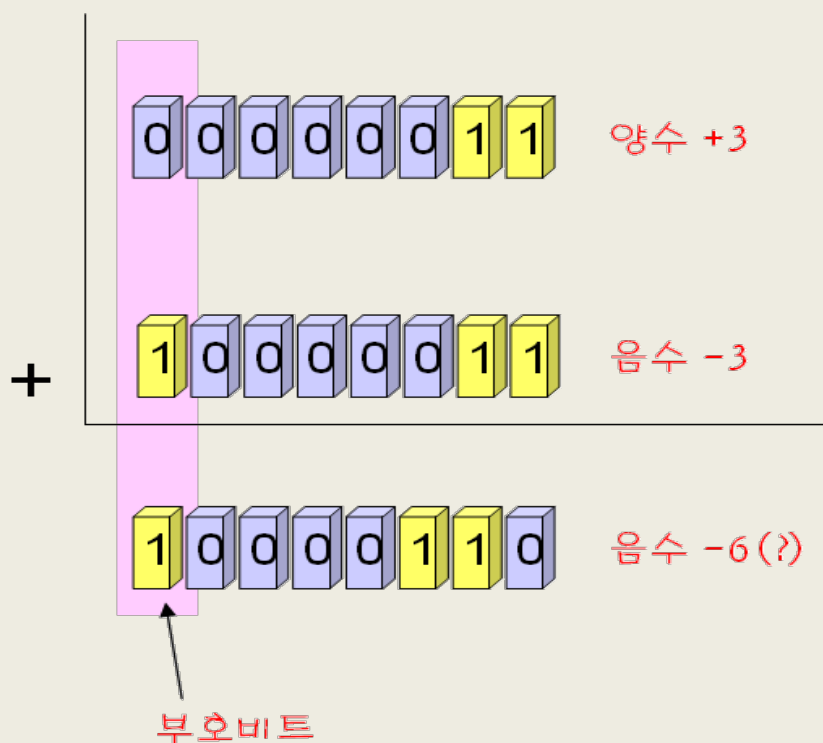


## (1) 부호와 절대값으로 정수 표현

- ❑ Most Significant Bit(MSB, 최상위 유효비트)를 부호(sign) 비트로 간주하고 나머지 비트들은 절대값으로 표시
  - ▣ MSB = 1 : 음수
  - ▣ MSB = 0 : 양수
- ❑ 예를 들어, 8-비트로 정수를 표현한다고 가정하면(보통 컴퓨터에서는 32-비트로 표현하지만 너무 길어서...)
  - ▣ 10001101 = ? 일단 음수다 => 10진수로 말하면 -11
  - ▣ 00000011 = ? 일단 양수다 => 10진수로 말하면 +3
- ❑ 8-비트로 정수를 표현한다고 가정할 때, 표현할 수 있는 정수의 범위는?
  - ▣ 최대 수 : 01111111 (10진수로 말하면 127)
  - ▣ 최소 수 : 11111111 (10진수로 말하면 -127)
  - ▣ **n 비트로 표현 가능한 정수의 범위는  $-(2^{n-1} - 1)$ 부터  $+(2^{n-1} - 1)$**
- ❑ 그러면 0은? 00000000 인가? 10000000 인가?
  - ▣ 둘 모두 0으로 0 표현이 2개가 존재
- ❑ 덧셈 결과가 부정확하다



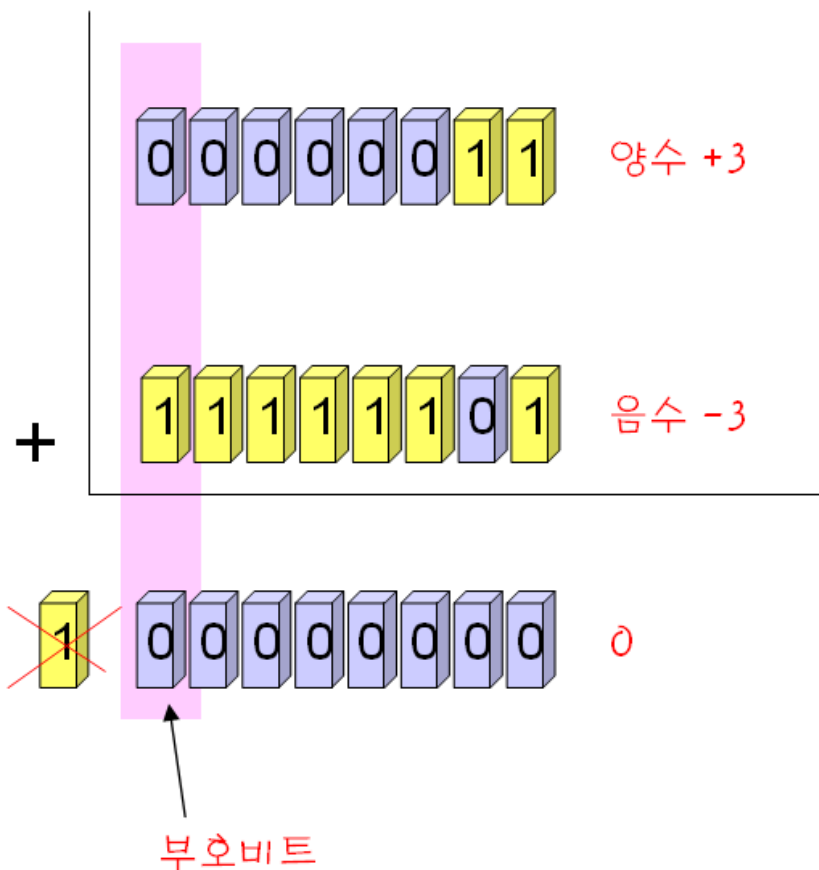
- ❑ 양수와 음수의 덧셈 연산을 하였을 경우, 결과가 부정확하다.
  - ❑ (예)  $+3 + (-3) = 0$  이 도출되어야 한다
  - ❑ 그런데...





## (2) 2's Complement로 정수 표현

- ❑ 양수이면 그대로 표현하고, **그 양수의 2의 보수를 음수로 정의**하는 표기법
  - ❑ 8-비트로 정수를 표현한다고 가정할 때,
  - ❑  $+5 = 00000101$  인데, 그러면  $-5$ 는 어떻게 표현하는가?
    - ❑  **$-5$ 는  $+5$ 의 2's complement**, 즉  $00000101(+5)$ 의 2's complement 인  $11111011$  이  $-5$ 가 되는 것이다
- ❑ 8-비트로 정수를 표현한다고 가정할 때, 표현할 수 있는 정수의 범위는?
  - ❑ 최대 수 :  $01111111$  (10진수로 말하면 127)
  - ❑ 최소 수 :  $10000000$  (10진수로 말하면 -128)
  - ❑ 그러므로  **$n$  비트로 표현 가능한 정수의 범위는  $-(2^{n-1})$ 부터  $+(2^{n-1} - 1)$**
- ❑ 0은  $00000000$  으로 하나만 존재, 그러면  $10000000$ 은?
  - ❑  $10000000$  은 -128로 간주 (왜? MSB가 1이므로...)
- ❑ 두수의 합 연산이 정확하게 도출된다



음수를 2의 보수로 표현하면 양수와 음수를 더할때 각각의 비트들을 더하면 됩니다.



# 예 제

```
/* 2의 보수 프로그램*/
```

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int x = 3;
```

```
    int y = -3;
```

음수가 2의 보수로  
표현되는지를 알아보자.

```
    printf("x = %08X\n", x);
```

// 8자리의 16진수로 출력한다.

```
    printf("y = %08X\n", y);
```

// 8자리의 16진수로 출력한다.

```
    printf("x+y = %08X\n", x+y);
```

// 8자리의 16진수로 출력한다.

```
    return 0;
```

```
}
```

*x = 00000003*

*y = FFFFFFFD*

*x+y = 00000000*



### (3) 초과표기법(Excess Notation)

- 표현 비트 길이가  $n$ 이라면 →  $n$ -bit excess notation 이라고 한다
- 0의 설정 : MSB=1이고 나머지는 모두 0인 수를 0으로 정의
  - 그러므로 4-비트 초과표기법으로 정수를 표현한다고 할 때, 1000 을 0으로 간주하고 그보다 1만큼 크면 +1, 작으면 -1, 2만큼 크면 +2, 작으면 -2, ... 이러한 방식의 정수 표현 방식

1111 → 7	0000 → -8	** 이 코드를 살펴보면 원래의 2진수 값보다 8만큼 초과한 코드를 사용하고 있음을 알 수 있다.
1110 → 6	0001 → -7	
1101 → 5	0010 → -6	** 그래서 이 표기 방법을 <u>4-bit 초과표기</u> (excess notation) 혹은 <u>8초과 코드</u> (Excess-8 code)라고 한다.
1100 → 4	0011 → -5	
1011 → 3	0100 → -4	** $n$ -bit를 사용하여 표현한다면, $n$ -bit excess notation 혹은 Excess- $2^{n-1}$ code 라고 한다
1010 → 2	0101 → -3	
1001 → 1	0110 → -2	
1000 → 0	0111 → -1	



- ❑ 2진수로 표현한 정수 1011 에 대하여,
  - ❑ 부호와 절대값으로 표현한 것이라면 = -3
  - ❑ 1의 보수 표기법으로 표현한 것이라면 = -4
  - ❑ 2의 보수 표기법으로 표현한 것이라면 = -5
  - ❑ 초과표기로 표현한 것이라면 = +3
  
- ❑ 8-bit 초과 표기를 사용한다고 할 때, 10110110 은 10진수로 얼마인가?



## (4) 4-bit BCD(Binary Coded Decimal) 코드

- ❑ 2진화 10진 코드 혹은 8421 **가중치** 코드
- ❑ 10진의 각 자리를 0000부터 1001까지의 4-비트 2진수로 표기
- ❑ 1010, 1011, 1100, 1101, 1110, 1111의 6개 코드는 사용하지 않음
- ❑ 그 외는 16진수 표기와 동일하다
- ❑ 예)  $953_{10} = 1001\ 0101\ 0011$
- ❑ 예)  $64.31_{10} = 0110\ 0100 . 0011\ 0001$
- ❑ 2진수보다 비트 사용이 비효율적인 반면 10진수로의 변환이 용이
- ❑ 정수의 표현 : **음수는 10의 보수를 BCD로 표현**
  - 양수 : 0000 (0)
  - 음수 : 1001 (9)
  - +375 : 0000 0011 0111 0101
  - -240 = 9760 : 1001 0111 0110 0000
  - (0240의 10의 보수는 9760이다)





- 예)
- $+375 - 240 =$
- $+375 + (-240) =$
- $0375 + 9760 =$
- $0135 = +135$
- 이때 발생하는 캐리는 무시한다

- 예)
- $-240 - 240 =$
- $9760 + 9760 =$
- $9520 = -480$
- 발생하는 캐리는 무시
- 9520은 -480 이다



## (5) BCD 코드의 변형

- ❑ 3초과 코드(Excess-3 code)
  - ❑ BCD 코드(8421 코드)로 표현된 값에 3을 더해준 값으로 나타내는 코드: 그러므로 가중치 코드가 아니다
  - ❑ 자기 보수(**self complementary**) 특성
  - ❑ **모든 비트가 0인 코드는 존재하지 않는다**

1's complement

10진 수	BCD코드	3초과코드
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100



## 다양한 10진 코드들

- ❑ BCD 코드로 정수 표현 → **packed decimal**
- ❑ 가장 우측에 4-bit를 추가하여 부호로 사용한다
  - ❑ 양수 C, F : 음수 D
  - ❑ +123 = 0001 0010 0011 **1100**
  - ❑ -456 = 0100 0101 0110 **1101**
- ❑ 기타 가중치 코드들
  - ❑ 2421 코드
  - ❑ 5421 코드
  - ❑ 84-2-1 코드
  - ❑ 51111 코드
  - ❑ Biquinary 코드 (5043210)
  - ❑ Ring Counter (9876543210)

## 기타 가중치 코드

10진 수	8421코드 (BCD)	2421 코드	5421 코드	84-2-1 코드	5111 코드	바이쿼너리코드 (Biquinary Code) 5043210	링 카운터 (ring counter) 9876543210
0	0000	0000	0000	0000	00000	0100001	0000000001
1	0001	0001	0001	0111	00001	0100010	0000000010
2	0010	0010	0010	0110	00011	0100100	0000000100
3	0011	0011	0011	0101	00111	0101000	0000001000
4	0100	0100	0100	0100	01111	0110000	0000010000
5	0101	1011	1000	1011	10000	1000001	0000100000
6	0110	1100	1001	1010	11000	1000010	0001000000
7	0111	1101	1010	1001	11100	1000100	0010000000
8	1000	1110	1011	1000	11110	1001000	0100000000
9	1001	1111	1100	1111	11111	1010000	1000000000



# Gray Code

- ❑ 비가중치 코드로 연산에는 부적당하지만
- ❑ 이웃하는 코드 간에 하나의 비트만 다르다는 특성을 가지므로 여러가지 하드웨어 제어 코드 등으로 유용하게 사용된다

10진수	2진코드	Gray code	10진수	2진코드	Gray code
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000



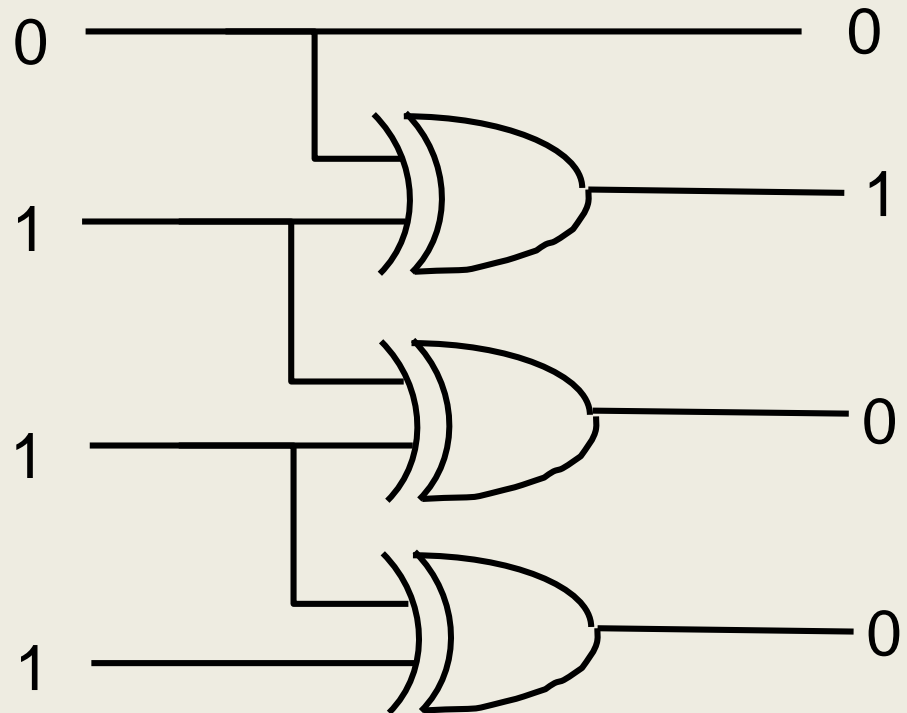
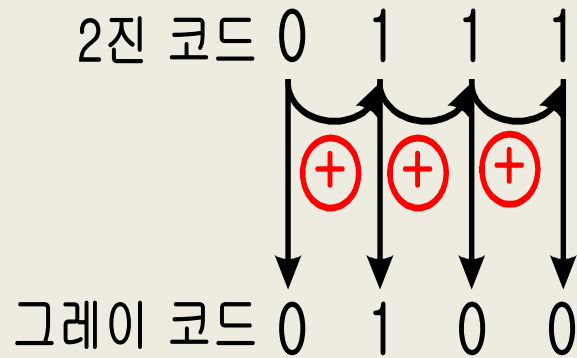
## 2진 코드 → Gray code 생성

- ❑ 2진 코드의 MSB는 변경 없이 유지한다
- ❑ MSB부터 시작하여 오른쪽으로 이웃한 2비트 끼리 XOR 연산을 하여 Gray code의 나머지 비트를 생성
- ❑ 예) 2진 코드 0 1 1 1에 해당하는 Gray code를 생성하면
  - ❑ MSB 0은 그대로 유지 **0**
  - ❑ **0**과 그 다음 비트 1과 XOR 하면 **1**
  - ❑ **1**과 그 다음 비트 1을 XOR 하면 **0**
  - ❑ **1**과 그 다음 비트 1을 XOR 하면 **0**
  - ❑ 그러므로 2진 코드 0 1 1 1에 대한 Gray code는 0 1 0 0이 된다



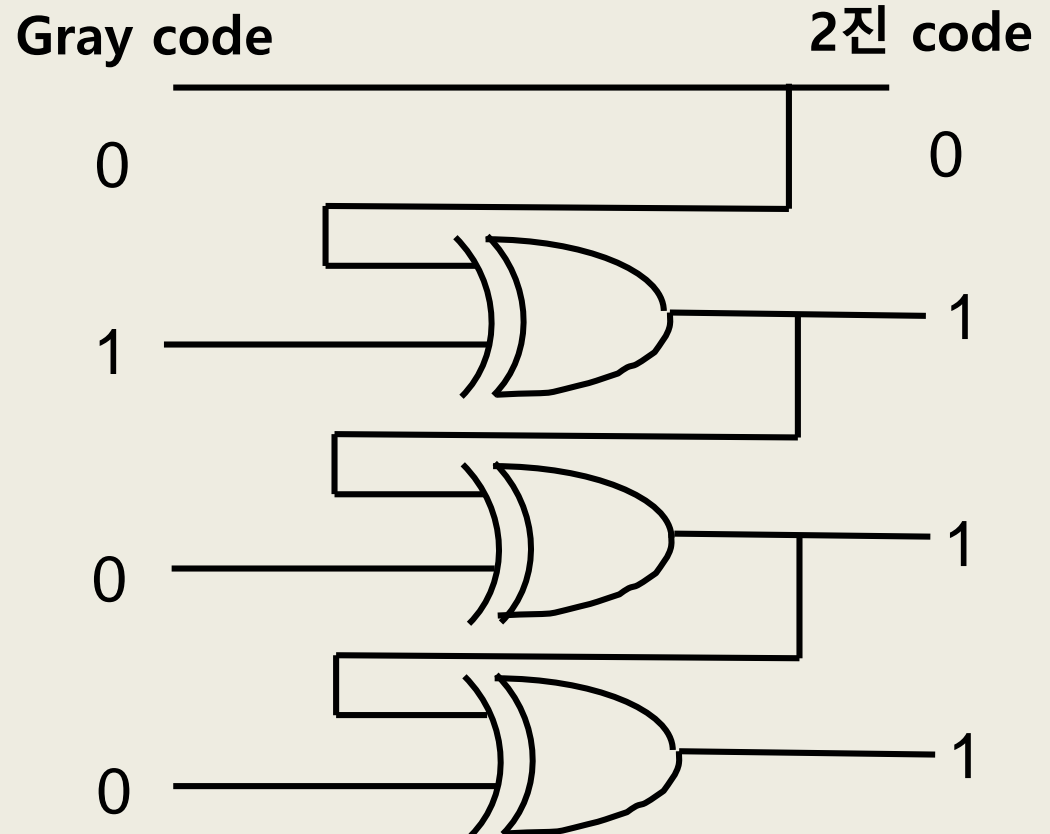
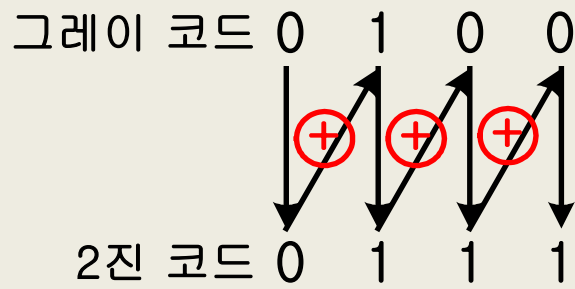
2진코드

Gray code





## Gray code → 2진 코드





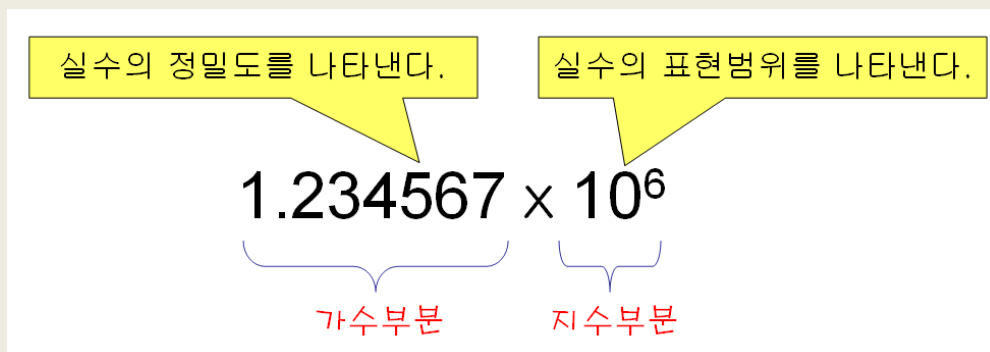
## 기타 비가중치 코드

10진수	3-초과 코드	5중 2코드 (2-out-of-5)	shift counter	그레이코드
0	0011	11000	00000	0000
1	0100	00011	00001	0001
2	0101	00101	00011	0011
3	0110	00110	00111	0010
4	0111	01001	01111	0110
5	1000	01010	11111	0111
6	1001	01100	11110	0101
7	1010	10001	11100	0100
8	1011	10010	11000	1100
9	1100	10100	10000	1101



# 부동소수점형(Floating Point Expression)

- 3개의 부분으로 이루어진다
  - 부호(sign) 부분
  - 정규화된 가수부분 (normalized fraction part)
  - 지수(exponent) 부분



실수	과학적 표기법	지수 표기법
123.45	$1.2345 \times 10^2$	1.2345e2
12345.0	$1.2345 \times 10^5$	1.2345e5
0.000023	$2.3 \times 10^{-5}$	2.3e-5
2,000,000,000	$2.0 \times 10^9$	2.0e9



## ❑ 고정 소수점 방식(Fixed Point Expression)

- ❑ 정수 부분을 위하여 일정 비트를 할당하고 소수 부분을 위하여 일정 비트를 할당
- ❑ 예를 들어, 전체가 32비트이며, 정수 부분 16비트, 소수 부분 16비트 할당
- ❑ (예) 3.14

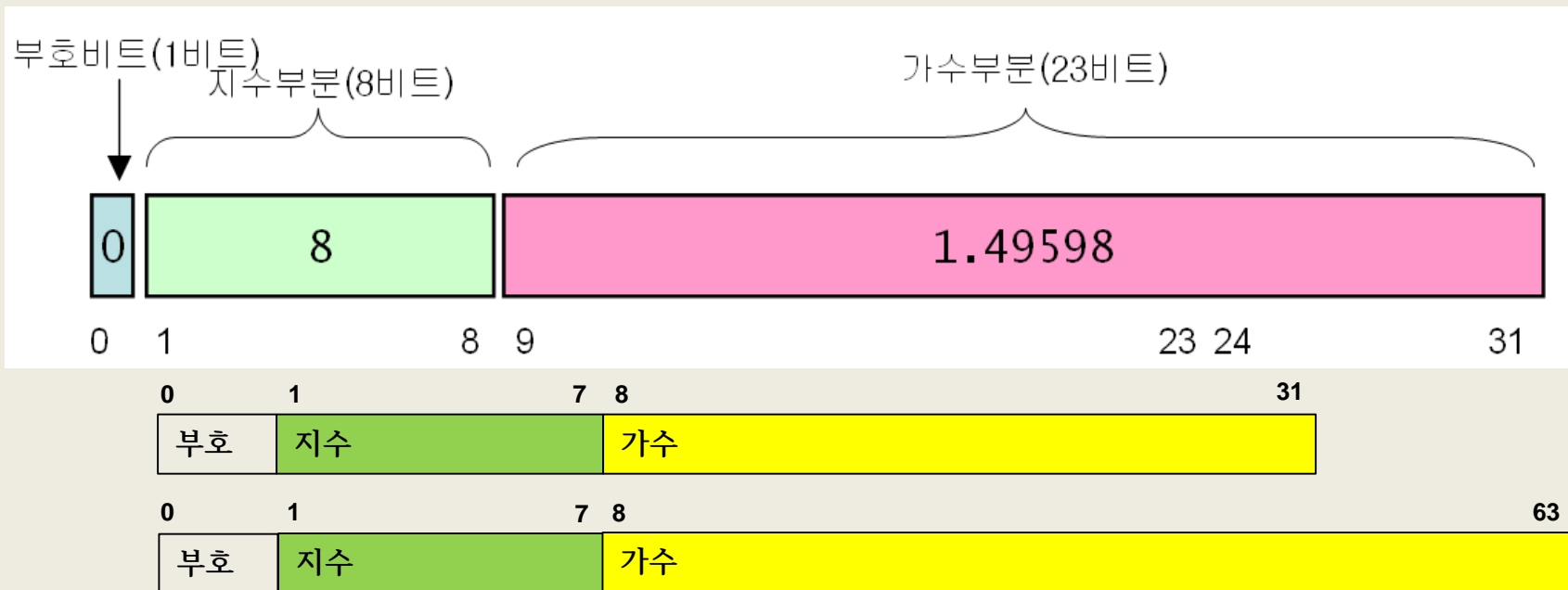
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

- ❑ 과학과 공학에서 필요한 아주 큰 수를 표현할 수 없다

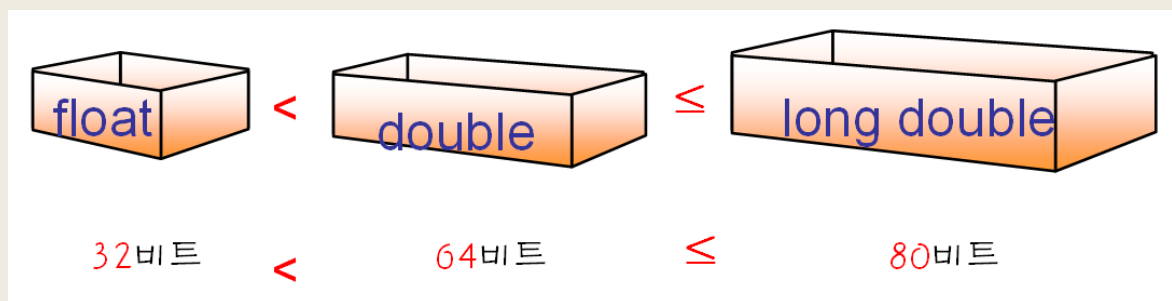


## □ 부동 소수점 방식 비트 할당 - 표현 비트 수가 32 비트일 경우

- 부호 1비트, 지수 7비트, 가수 24비트라면



- 표현할 수 있는 범위가 대폭 늘어난다.
- $10^{-38}$  에서  $10^{+38}$



자료형	명칭	크기	범위
float	단일정밀도(single-precision) 부동소수점	32비트	$\pm 1.17549 \times 10^{-38} \sim \pm 3.40282 \times 10^{+38}$
double	두배정밀도(double-precision) 부동소수점	64비트	$\pm 2.22507 \times 10^{-308} \sim \pm 1.79769 \times 10^{+308}$
long double	두배확장정밀도 (double-extension-precision) 부동소수점	64비트 또는 80비트	$\pm 2.22507 \times 10^{-308} \sim \pm 1.79769 \times 10^{+308}$



## 예제

```
/* 부동 소수점 자료형의 크기 계산*/
#include <stdio.h>

int main(void)
{
    float x = 1.234567890123456789;
    double y = 1.234567890123456789;

    printf("float의 크기=%d\n", sizeof(float));
    printf("double의 크기=%d\n", sizeof(double));
    printf("long double의 크기=%d\n", sizeof(long double));

    printf("x = %30.25f\n", x);
    printf("y = %30.25f\n", y);
    return 0;
}
```

float의 크기=4

double의 크기=8

long double의 크기=8

x = 1.23456788063049320000000000

y = 1.23456789012345670000000000



- ❑ 예제
- ❑ 전체 길이는 16비트
  - ❑ 부호 부분 : MSB 1 비트
  - ❑ 지수 부분 : 5 비트 (15초과 표기법으로 표현)
  - ❑ 가수 부분 : 10 비트
  - ❑ 여기서 부동소수점으로 표현된 0 10010 1011110000 은 10진수로 얼마인가?
    - ❑ 1) 부호는 0-이므로 양수
    - ❑ 2) 지수부분 10010 은 15초과표기를 사용하므로 10진수로 3
    - ❑ 3) 가수부분은 정규화되어 있다는 전제하에서
  - ❑ 그러므로 이 수는  $0.1011110000 \times 2^3 = 101.1110000 = 5.875$



- ❑ 8 비트로 부동소수점을 표현한다고 할 때, 01101011 은 얼마인가?  
2진수로 답하라. 단 지수부분 3비트 가수부분 4비트이며, 지수부분  
은 3-비트 초과코드를 사용한다
  - 0 110 1011 이므로 부호는 양수, 지수부분은 110 이므로 2이며 가수부분은  
1011 이므로 2진수로 표현하면  $0.1011 \times 2^2 = 10.11$  이된다
- ❑  $13.25_{10}$ 을 32비트 부동소수점 방식으로 표현하면 얼마인가? 2진수  
로 표현하라. 단 지수부분은 7-비트 초과표기법을 사용한다
  - 일단 양수이므로 부호 비트 = 0
  - 그리고 13.25를 2진수로 변환하여야 하는데  $13.25_{10} = 1101.01_2$  가되므로 가수  
부분은 0.110101 이 되고, 지수부분은 4가 되어야 한다.
  - 그러므로 컴퓨터 내부에서  $13.25_{10}$  은
  - 0 1000100 110101000000000000000000 이렇게 표현된다





## 부동 소수점 상수

- 일반적인 실수 표기법
  - 3.141592(double형)
  - 3.141592F(float형)
- 지수 표기법
  - $1.23456e4 = 12345.6$
  - $1.23456e-3 = 0.00123456$
- 유효한 표기법의 예
  - 1.23456
  - 2.       // 소수점만 붙여도 된다.
  - .28       // 정수부가 없어도 된다.
  - 0e0
  - $2e+10$     // +나 -기호를 지수부에 붙일 수 있다.
  - 9.26E3    //
  - 9.26e3    //



## 부동 소수점 오버플로우

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
float x = 1e39;
```

```
printf("x = %e\n",x);
```

```
}
```

숫자가 커서 오버플로우  
발생

```
C:\CPROGRAM\test\test.c(5) : warning C4056: overflow in floating-point  
constant arithmetic
```



## 부동 소수점 언더플로우

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    float x = 1.23456e-38;
```

```
    float y = 1.23456e-40;
```

```
    float z = 1.23456e-46;
```

```
    printf("x = %e\n", x);
```

```
    printf("y = %e\n", y);
```

```
    printf("z = %e\n", z);
```

```
}
```

숫자가 작아서  
언더플로우 발생

```
x = 1.234560e-038
```

```
y = 1.234558e-040
```

```
z = 0.000000e+000
```



# 부동소수점 연산 오차(truncation error)

- ❑ 오차가 있을 수 있다!

```
#include <stdio.h>
```

```
int main(void)  
{
```

```
    double x;
```

```
    x = (1.0e20 + 5.0) - 1.0e20;
```

```
    printf("%f \n", x);
```

```
    return 0;
```

```
}
```

5.0은 부동 소수점수가  
저장할 수 있는  
한계때문에 저장되지  
않는다.


0.000000



## 절단 오차(Truncation Error)

- ❑ 부동소수점 실수를 표현하는 비트 길이에 따라 달라진다
- ❑ 예) 8비트 부동 소수점 표현
  - 부호 1비트 지수 3비트 초과표기 사용 가수 4비트
  - $2.625 = 10.101_2$  를 표현하려고 하면,...
  - $10.101 = 0.10101 \times 2^2$  : 부호 0 지수 110 가수 1010 이 된다
- ❑ 즉, 0 110 1010 이 되어 맨 마지막 1이 잘려 나가게 된다
- ❑ 이러한 오차를 절단 오차라고 한다
  - 그러므로 2.625는 2.5로 되어 연산이 이루어지게 되는 것이다
- ❑ 2.625의 정확한 표현을 위해서는 5비트의 가수 부분을 필요로 함



- 앞에서 언급한 8비트 부동 소수점 표현방식을 사용한다고 가정하고, 다음과 같은 연산 ( $2.5 + 0.125 + 0.125$ )을 생각해 보자
- $2.5 + 0.125 + 0.125 = 2.5 + 0.125 = 2.5$  (절단 오차 발생)  

- 그러나 이를 ( $0.125 + 0.125 + 2.5$ ) 처럼 연산 순서를 바꾸면
- $0.125 + 0.125 = 0.25 \rightarrow 0.25 + 2.5 = 2.75$ 가 되어 정확한 값을 도출한다