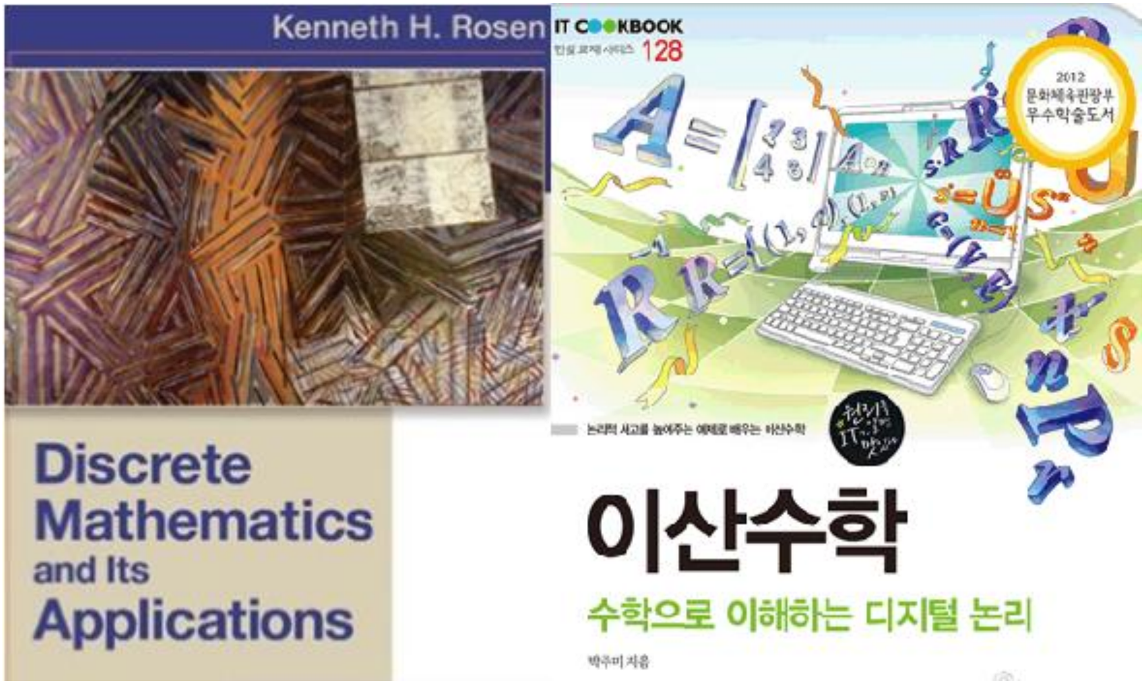
이산수학
Discrete Mathematics



Chapter 04:
수의 표현

Soongsil University : Kim Chang Wook

수의 연산

❖합이나 곱의 식에서 일정한 규칙을 가지고 연속적으로 나열된 수열을 계산할 경우에 특정 기호를 사용 : 합 $\sum$ 곱 $\prod$

❖ 합의 표시 $\sum$: 일정 규칙으로 나열된 값의 합

$$\sum_{k=1}^{n} k = 1+2+3+\cdots+n \qquad \sum_{i=1}^{n}(x_i+y_i) = \sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i$$

예제 4-8

$$\sum_{j=1}^{4} j^2 + 9 = (1^2+2^2+3^2+4^2)+9 = 1+4+9+16+9 = 39$$

$$\sum_{j=1}^{4} (j^2+9) = (1^2+9)+(2^2+9)+(3^2+9)+(4^2+9) = 10+13+18+25 = 66$$

❖곱의 표시 $\prod$: 일정 규칙으로 나열된 값의 곱

$$\prod_{k=1}^{15} k = 1\times 2\times 3\times\cdots\times 15$$

예제 4-9

$$\prod_{i=2}^{8} y_i = y_2\times y_3\times y_4\times y_5\times y_6\times y_7\times y_8$$

$$\prod_{i=0}^{4}(10-2i) = (10-2\times 0)\times(10-2\times 1)\times(10-2\times 2)\times(10-2\times 3)\times(10-2\times 4)$$

$$= 10\times 8\times 6\times 4\times 2 = 3840$$

수의 연산

❖나누기 연산 $d|n$: 정수 n을 d로 나누어 몫 q를 구하는 연산
또는 $n=dq$ 를 만족하는 정수 q를 구하는 연산

$d|n$: d로 n 을 나눈다. ($d \neq 0$)

$d \nmid n$: d로 n 을 나누지 못한다

- q : 몫(quotient) d : n의 약수 (divisor) 또는 인수(factor) n : d의 배수

❖나머지 연산 n mod d

- 정수 n을 d로 나누어 나오는 몫 q와 나머지 r이 있을 때, r을 구하는 연산
  $n = dq + r$ 을 만족하는 정수 r을 구하는 연산 n mod $d = r$
- q : 몫 d : n의 약수 또는 인수 n : d의 배수 r : 나머지(remainder), $0 \leq r < d$
- 나머지 연산자는 나머지만을 결과로 갖는다.
  나머지 r이 0일 경우 나누기 연산 결과와 같다. n mod d=0 ⇔ $d|n$

예제 4-10

(1) 3|9 : 3은 9의 약수고, 9는 3의 배수므로 맞는 표현이다. ∴ 3|9 =3

(2) 7∤42 : 42는 7의 배수로 틀린 표현. 7|42로 표기를 고쳐야 한다. ∴ 7|42 =6

(3) 8|10 : 8은 10의 약수가 아니므로 틀린 표현. 그러므로 8∤10 로 고쳐야 한다.

(4) 6∤15 : 6은 15의 약수가 아니므로 맞는 표현.

(5) 10|100 : 100은 10의 배수로 맞는 표현. ∴ 10|100=10

예제 4-11

(1) 27 mod 4 = 3
(2) 52 mod 4 = 0 4|52
(3) 79 mod 10 =9
(4) 25 mod 5 = 0 5|25

수 체계

10진수(Decimal number)

- 기수를 10으로 하는 수 체계, 0에서 9 사이의 숫자를 이용해 수를 표현
  - 정수 n 에 대해 $(k > 0, a \geq 0)$

$$n_{10} = a_k a_{k-1} \ldots a_1 a_0 = a_k 10^k + a_{k-1} 10^{k-1} + \ldots a_1 10^1 + a_0 10^0$$

  - 실수 n 에 대해

$$n_{10} = a_k a_{k-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-l} a_{-(l+1)} \cdots$$
$$= a_k 10^k + a_{k-1} 10^{k-1} + \ldots a_1 10^1 + a_0 10^0$$
$$+ a_{-1} 10^{-1} + a_{-2} 10^{-2} + \ldots + a_{-l} 10^{-l} + a_{-(l+1)} 10^{-(l+1)} + \ldots$$

예제 4-12

다음 10진수를 기수와 자리수를 이용해 풀어써라.

(1) 1582_{10} (2) 523.1568_{10}

풀이

(1) $1582_{10} = 1 \times 10^3 + 5 \times 10^2 + 8 \times 10^1 + 2 \times 10^0$

(2) $523.6218_{10} = 5 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 + 6 \times 10^{-1} + 2 \times 10^{-2} + 1 \times 10^{-3} + 8 \times 10^{-4}$

2진수(Binary number)

- 기수를 2로 하는 수 체계, 0과 1을 이용해 수 표현

예제 4-13

다음 2진수 10001.001101_2 를 기수와 자리수를 이용해 풀어써라.

풀이 $10001.001101_2 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
$+ 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} + 0 \times 2^{-5} + 1 \times 2^{-6}$

수 체계

❖ 8진수(Octal number)

- 기수를 8로 하는 수 체계, 0부터 7사이의 숫자를 이용해 수를 표현
  - 실수 n에 대해 $(k, l > 0, 0 \le a \le 7)$

$$n_8 = a_k a_{k-1} \dots a_1 a_0 . a_{-1} a_{-2} \dots a_{-l} a_{-(l+1)} \cdots$$
$$= a_k 8^k + a_{k-1} 8^{k-1} + \dots + a_1 8^1 + a_0 8^0 + a_{-1} 8^{-1} + \cdots + a_{-l} 8^{-l} + \dots$$

예제 4-13

다음 8진수 712.3251_8 를 기수와 자리수를 이용해 풀어 써라.

풀이 $712.3251_8 = 7 \times 8^2 + 1 \times 8^1 + 2 \times 8^0 + 3 \times 8^{-1} + 2 \times 8^{-2} + 5 \times 8^{-3} + 1 \times 8^{-4}$

❖ 16진수(Hexa number)

- **기수를 16으로 하는 수 체계, 0에서 9, A(10),B(11), C(12), D(13), E(14), F(15)**
  - 실수 n에 대해 $(k, l > 0, 0 \le a \le 9$ 또는 $A \le a \le F)$

$$n_{16} = a_k a_{k-1} \dots a_1 a_0 . a_{-1} a_{-2} \dots a_{-l} a_{-(l+1)} \cdots$$
$$= a_k 16^k + a_{k-1} 16^{k-1} + \dots + a_1 16^1 + a_0 16^0 + a_{-1} 16^{-1} + \dots + a_{-l} 16^{-l} + \dots$$

예제 4-15

16진수 (1) $A41C_{16}$ (2) $9B.FE3_{16}$ 를 기수와 자리수를 이용해 풀어써라.

풀이 (1) $A41C_{16} = A \times 16^3 + 4 \times 16^2 + 1 \times 16^1 + C \times 16^0$
$= 10 \times 16^3 + 4 \times 16^2 + 1 \times 16^1 + 12 \times 16^0$

(2) $9B.FE3_{16} = 9 \times 16^1 + B \times 16^0 + F \times 16^{-1} + E \times 16^{-2} + 3 \times 16^{-3}$
$= 9 \times 16^1 + 11 \times 16^0 + 15 \times 16^{-1} + 14 \times 16^{-2} + 3 \times 16^{-3}$

수 체계 변환

❖10진수 → 2진수 / 8진수 / 16진수

- **정수부**
  - 기수로 몫이 0이 나올 때까지 나누어 얻은 나머지를 나열
  - 가장 처음에 얻은 나머지는 최하위 자리(가장 오른쪽 자리)에 위치
  - 가장 나중에 얻은 나머지는 최상위 자리(가장 왼쪽 자리)에 위치
- **실수부**
  - 기수로 실수부가 0이 나올 때까지 곱해 얻은 정수부의 값을 나열
  - 가장 처음에 얻은 정수부는 소수점에 가장 가까운 자리에 위치
  - 가장 나중에 얻은 정수부는 소수점에 가장 멀리 있는 자리에 위치

수 체계 변환

예제 4-20

다음 10진수를 2진수로 변환하라.

(1) 274_{10} (2) 163.875_{10}

풀이

(1)

			최하위
2	274		
2	137	…	0
2	68	…	1
2	34	…	0
2	17	…	0
2	8	…	1
2	4	…	0
2	2	…	0
2	1	…	0
	0	…	1

최상위

최상위자리

$\therefore\ 274_{10} = 100010010_2$

최하위자리

(2)

			최하위
2	163		
2	81	…	1
2	40	…	1
2	20	…	0
2	10	…	0
2	5	…	0
2	2	…	1
2	1	…	0
	0	…	1

최상위

최상위자리

$\therefore\ 163_{10} = 10100011_2$

최하위자리

$$0.875 \times 2 = 1.750 \qquad 0.75 \times 2 = 1.50 \qquad 0.5 \times 2 = 1.0$$

$\therefore\ 0.875_{10} = 0.111_2$

$\therefore\ 163.875_{10} = 10100011.111_2$

10진수→ 2진수

(1) $274_{10} = 100010010_2$

(2) $163.875_{10} = 10100011.111_2$

수 체계 변환

예제 4-21

다음 10진수를 8진수로 변환하라.

(1) 274_{10} (2) 163.875_{10}

풀이

(1)

$$\begin{array}{r|r l r}
8 & 274 & & \text{최하위} \\ \hline
8 & 34 & \cdots & 2 \\ \hline
8 & 4 & \cdots & 2 \\ \hline
 & 0 & \cdots & 4 \\
 & & & \text{최상위}
\end{array}$$

$\therefore\ 274_{10} = 422_8$

(2)

$$\begin{array}{r|r l r}
8 & 163 & & \text{최하위} \\ \hline
8 & 20 & \cdots & 3 \\ \hline
8 & 2 & \cdots & 4 \\ \hline
 & 0 & \cdots & 2 \\
 & & & \text{최상위}
\end{array}$$

$\therefore\ 163_{10} = 243_8$

$$\begin{array}{r}
0.875 \\
\times \quad 8 \\ \hline
\boxed{7}.000
\end{array}$$

$\therefore\ 0.875_{10} = 0.7_8$

$\therefore\ 163.875_{10} = 243.7_8$

10진수 → 2진수 → 8진수

주어진 2진수를 소수점 기준으로 각 세 자리 단위의 블록을 8진수로 변환

(1) $274_{10} = 100,010,010_2 = 422_8$

(2) $163.875_{10} = 10,100,011.111_2 = 243.7_8$

수 체계 변환

예제 4-22

다음 10진수를 16진수로 변환하라.

(1) 274_{10} (2) 163.875_{10}

0:0
1:1
:
9: 9
10: A
11: B
12: C
13: D
14: E
15: F

풀이

(1)

$$\begin{array}{r|l l l}
16 & 274 & & \text{최하위} \\ \hline
16 & 17 & \cdots & 2 \\ \hline
16 & 1 & \cdots & 1 \\ \hline
 & 0 & \cdots & 1 \\
 & & & \text{최상위}
\end{array}$$

$\therefore\ 274_{10} = 112_{16}$

(2)

$$\begin{array}{r|l l l}
16 & 163 & & \text{최하위} \\ \hline
16 & 10 & \cdots & 3 \\ \hline
 & 0 & \cdots & 10 \\
 & & & \text{최상위}
\end{array}$$

$\therefore\ 163_{10} = A3_{16}$

$$\begin{array}{r}
0.875 \\
\times \quad 16 \\ \hline
14.000
\end{array}$$

$\therefore\ 0.875_{10} = 0.E_{16}$

$\therefore\ 163.875_{10} = A3.E_{16}$

10진수 → 2진수 → 16진수

주어진 2진수를 소수점 기준으로 각 4자리 단위의 블록을 16진수로 변환

(1) $274_{10} = 1,0001,0010_{2} = 112_{16}$

(2) $163.875_{10} = 1010,0011.1110_{2} = A3.E_{16}$

$(\ 10\ ,\ 3\ .\ 14\ )_{16}$

수 체계 변환

❖2진수 / 8진수 / 16진수 → 10진수

- 기수와 수를 구성하는 숫자의 자리수를 이용

예제 **4-23**

다음을 10진수로 변환하라.

(1) 10101_2 (2) 1101.001_2 (3) 724_8

(4) 365.114_8 (5) $3CA_{16}$ (6) $E1.F01_{16}$

풀이

(1) $10101_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 16 + 0 + 4 + 0 + 1 = 21_{10}$

(2) $1101.001_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$

$= 8 + 4 + 0 + 1 + 0 + 0 + \frac{1}{8} = 13.125_{10}$

(3) $724_8 = 7 \times 8^2 + 2 \times 8^1 + 4 \times 8^0 = 7 \times 64 + 2 \times 8 + 4 \times 1 = 448 + 16 + 4 = 468_{10}$

(4) $365.114_8 = 3 \times 8^2 + 6 \times 8^1 + 5 \times 8^0 + 1 \times 8^{-1} + 1 \times 8^{-2} + 4 \times 8^{-3}$

$= 3 \times 64 + 6 \times 8 + 5 \times 1 + 1 \times \frac{1}{8} + 1 \times \frac{1}{64} + 4 \times \frac{1}{512}$

$= 192 + 48 + 5 + 0.125 + 0.015625 + 0.0078125 = 245.1484375_{10}$

(5) $3CA_{16} = 3 \times 16^2 + C \times 16^1 + \mathrm{A} \times 16^0 = 3 \times 256 + 12 \times 16 + 10 \times 1$

$= 768 + 192 + 10 = 970_{10}$

(6) $E1.F01_{16} = \mathrm{E} \times 16^1 + 1 \times 16^0 + \mathrm{F} \times 16^{-1} + 0 \times 16^{-2} + 1 \times 16^{-3} +$

$= 14 \times 16 + 1 \times 1 + 15 \times \frac{1}{16} + 0 \times \frac{1}{256} + 1 \times \frac{1}{4096}$

$= 224 + 1 + 0.9375 + 0 + 0.000244140625 = 225.937744140625_{10}$

수 체계 변환

❖8진수 → 2진수

- 8진수의 각 자리를 세 자리의 2진수 (4,2,1) 로 변환

❖16진수 [0,…,9, A(10),B(11), C(12), D(13), E(14), F(15)] → 2진수

- 16진수의 각 자리를 네 자리의 2진수 (8,4,2,1) 로 변환

예제 4-24 4-25

2진수를 8진수로 변환 : $11\ 100\ 101\ .\ 010\ 011\ 110\ 1_2 = 345.2364_8$

8진수 345.2364_8를 2진수로 변환하라.

풀이 8진수의 각 자리를 세 자리의 2진수로 표현한다.

① $3_8 = 011_2$ ② $4_8 = 100_2$ ③ $5_8 = 101_2$ ④ $2_8 = 010_2$

⑤ $3_8 = 011_2$ ⑥ $6_8 = 110_2$ ⑦ $4_8 = 100_2$

$\therefore 345.2364_8 = 011100101.010011110100_2 = 11100101.0100111101_2$

예제 4-26 4-27

2진수를 16진수로 변환 $1110\ 0101.0100\ 1111\ 01_2 = E5.4F4_{16}$

16진수 $E5.4F4_{16}$를 2진수로 변환하라.

풀이 16진수의 각 자리를 네 자리의 2진수로 표현한다.

① $E_{16} = 14 = 1110_2$ ② $5_{16} = 0101_2$ ③ $4_{16} = 0100_2$ ④ $F_{16} = 15 = 1111_2$ ⑤ $4_{16} = 0100_2$

$\therefore E5.4F4_{16} = 11100101.010011110100_2 = 11100101.0100111101_2$

보수의 표현

❖보수

- 보충해 주는 수
  - 1에 대한 10의 보수 : $1 + n = 10$ ∴ $n = 9$
  - 3에 대한 10의 보수 : $3 + n = 10$ ∴ $n = 7$

❖1의 보수(2진수에서 : 1의 보수는 1은 0으로, 0은 1로)

- 어떤 수 n과의 합이 1이 되는 수
- ex) 110101_2 (53_{10})에 대한 1의 보수 : 001010 (0 → 1, 1 → 0)

  110101+ 001010=111111

❖2의 보수(2진수에서 : 2의 보수 = 1의 보수+1)

- 어떤 수 n과의 합이 2가 되는 수
- ex) 110101_2 (53_{10}) 에 대한 2의 보수 =1의 보수+1= 001010+1= 001011

  110101+001011=1000000

Representing Negative Numbers: Two's Complement

- In many systems, results may be negative,
  Input values may even negative numbers.
  Negative numbers common.   Need to represent negative numbers
  - How represent in binary?

- Signed-magnitude
  - Use leftmost bit for sign bit
    - Ex: –5: four bits : **1**101 : -5   **0**101 : 5
      eight bits : **1**0000101 : -5   **0**0000101: 5

MSB : Most Significant Bit

최상위 비트	…	…	…	…	…	…	최하위 비트
부호 비트	데이터 비트						

LSB : Least Significant Bit

- Better way: Two's complement
  - Big advantage : Allows us to perform subtraction using addition
  - Thus, only need adder component, no need for separate subtractor component

Ten's Complement(10의 보수)

- Before introducing two's complement, let's consider ten's complement
  - But, be aware that computers DO NOT USE TEN'S COMPLEMENT. Introduced for intuition only.
  - Complements for each base ten number shown to right. Complement is the number that when added results in 10

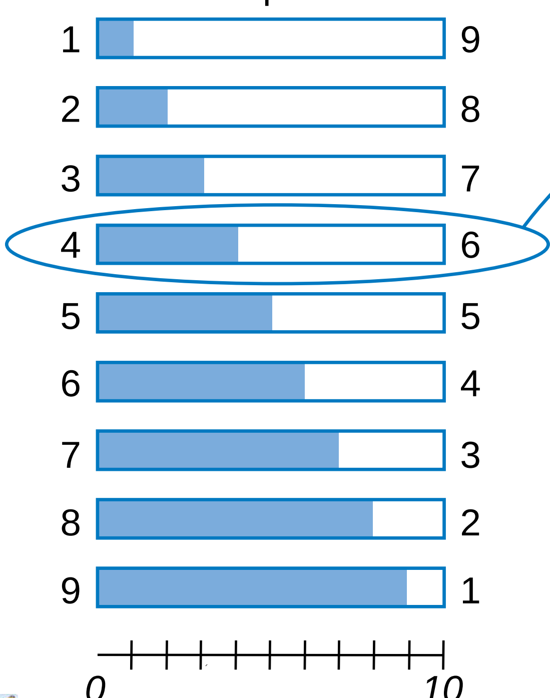
❖Nice feature of ten's complement

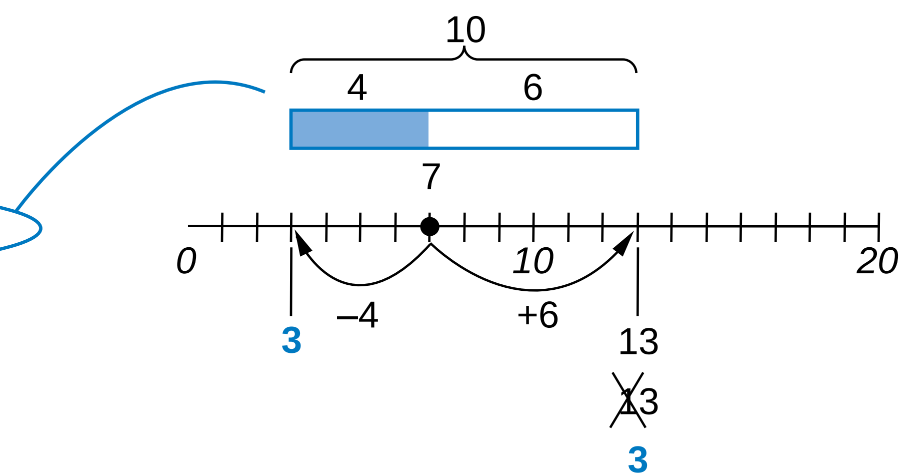
  - Instead of subtracting a number, adding its complement results in answer exactly 10 too much   7– 4→7+6(4 Ten's complement) = $13 \rightarrow \cancel{1}3 = 3$
  - So just drop the 1 : results in subtracting using addition only

Ten's complement

1 → 9
2 → 8
3 → 7
4 → 6
5 → 5
6 → 4
7 → 3
8 → 2
9 → 1

10's complements





7–4 = 7+(10-4)-10 = 7+6(4 10's complement) -10 = 3

Adding the complement results in an answer that is exactly 10 too much – dropping the tens column gives the right answer.

Two's Complement is Easy to Compute: Just Invert Bits and Add 1

❖ Hold on!

- Sure, adding the ten's complement achieves subtraction using addition only
- But don't we have to perform *subtraction* to have determined the complement in the first place?
  E.g., we only know that the complement of 4 is 6 by subtracting 10-4=6 in the first place.

❖ True. But in binary, it turns out that the two's complement can be computed **easily**

- Two's complement of 011 is 101, because 011 + 101 is 1000
- Could compute complement of 011 as 1000 – 011 = 101
- **Easier method: Just invert all the bits, and add 1** a
- The complement of 011 is 100+1 = 101. It works!

Q: What is the two's complement of 0101? A: 1010+1=1011 (check: 0101+1011=10000)

Q: What is the two's complement of 0011? A: 1100+1=1101 (check: 0011+1101=10000)

Two's Complement

❖ Two's complement can represent negative numbers

- Suppose have 4 bits
- Positive numbers 0 to 7 : 0000 to 0111
- Negative numbers -1 to -8 : Take two's complement of num
  - – 1: 0001 → 1110+1 = 1111
  - – 2: 0010 → 1101+1 = 1110
  - … … …
  - – 7: 0111 → 1000+1 = 1001
  - – 8: 1000 → 0111+1 = 1000

  So –1 to – 8 : 1111 to 1000
- Leftmost bit indicates sign of number, known as *sign bit.* 1 means negative.

❖ Signed vs. unsigned N-bit number

- Unsigned: 0 to 2^N-1
  - Ex. Unsigned 8-bit : 0(00000000) to 255(11111111)
- Signed (two's complement) : – 2^{N-1} to 2^{N-1}–1
  - Ex. Signed 8-bit : –128 (10000000) to 127 (01111111)

    -128(10000000), ··· , -1(11111111), 0(00000000), 1(00000001), ··· ,127 (01111111)

4bit
2의 보수
0:0000
1:0001
2:0010
3:0011
4:0100
5: 0101
6: 0110
7: 0111
-8: 1000
-7: 1001
-6: 1010
-5: 1011
-4: 1100
-3: 1101
-2: 1110
-1: 1111

컴퓨터에서의 데이터 표현

예제 4-28

10진수 $+53_{10}$과 -53_{10}을 8bits의 Signed-magnitude (부호화-절대치)로 나타내라.

풀이

$|53_{10}|$을 2진수로 변환하면 110101_2 . 8bits의 가장 왼쪽에 있는 최상위 비트(MSB) 자리에 부호를 의미하는 0(양수) 또는 1(음수)가 입력된다.
나머지 자리에는 절댓값이 오른쪽부터 채워진다.

· $+53_{10}$의 경우

0	0	1	1	0	1	0	1
부호 (양수)	$\lvert 53_{10} \rvert$						

$$\therefore +53_{10} = 00110101$$

· -53_{10}의 경우

1	0	1	1	0	1	0	1
부호 (음수)	$\lvert 53_{10} \rvert$						

$$\therefore -53_{10} = 10110101$$

컴퓨터에서의 데이터 표현

예제 4-30

10진수 $+53_{10}$과 -53_{10}을 8bits 2의 보수 표현으로 나타내라.

풀이

· 보수는 음수 표현에만 사용, $+53_{10}$에 대한 2의 보수 표현은 부호화-절대치와 같다.

$\therefore +53_{10} = 00110101$

· -53_{10}의 부호화-절대치 표현은 $-53_{10} = 10110101$.

-53_{10}은 음수, 2의 보수 표현으로 바꿀 수 있다. $-53_{10} = 11001010+1=11001011$

$\therefore -53_{10}$ 에 대한 2의 보수 표현은 11001011

예제 4-31

10진수 $\pm123_{10}$에 대한 부호화 - 절대치, 2의 보수 구하라.

풀이 $|123_{10}| = 1111011$

· 부호화 - 절대치 표현 : $+123_{10} = 01111011$ $-123_{10} = 11111011$

· 2의 보수 표현 : $+123_{10} = 01111011$

$-123_{10} = 10000100+1=10000101$

보수 연산

예제 4-36

· 2의 보수 연산 $+13_{10}$ -72_{10}

$+13_{10}$ 의 2의 보수 표현은 00001101. -72_{10}의 2의 보수 표현은 10111000.

$$
\begin{array}{rl}
111 & \\
00001101 & \quad +13_{10}\text{ 의 2의 보수} \\
+\ 10111000 & \quad -72_{10}\text{ 의 2의 보수} \\
\hline
11000101 &
\end{array}
$$

2의 보수로 연산했으므로 얻은 결과는 2의 보수가 된다.

다음 방식으로 10진수로 변환할 수 있다.

결과 11000101의 2의 보수 00111010+1=00111011, MSB 1은 음수이므로

$$-00111011 = -(0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0)$$

$$= -(0 + 32 + 16 + 8 + 0 + 2 + 1) = -59$$

보수 연산

예제 4-38

8bits 컴퓨터에서 다음을 2의 보수를 이용해 연산하라.

(1) $33_{10} - 15_{10}$ (2) $-38_{10} - 70_{10}$

풀이

(1) $33_{10} - 15_{10} = 33_{10} + (-15_{10})$

$|33_{10}| = 100001 = 00100001$, $|15_{10}| = 00001111$: 1의 보수 표현 $= 11110000$

2의 보수 표현은 $+33_{10} = 00100001$, $-15_{10} = $1의 보수 표현$+1 = 11110001$

$$\begin{array}{r} 11 \\ 00100001 \\ +\ 11110001 \\ \hline \boxed{1}00010010 \end{array}$$

무시

⇒ 결과가 양수이므로 2의 보수도 같은 값이다.

$\therefore 33_{10} - 15_{10} = 33_{10} + (-15_{10}) = 00010010$

$= +(0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0)$

$= +(0 + 0 + 16 + 0 + 0 + 2 + 0) = 18$

보수 연산

예제 4-38

(2) $-38_{10} - 70_{10} = (-38_{10}) + (-70_{10})$

$|38_{10}| = 00100110$, $|70_{10}| = 01000110$

2의 보수 표현 = 1의 보수 표현 + 1

2의 보수 표현은 -38_{10}= 11011010, -70_{10}= 10111010 이 된다.

$$\begin{array}{r} 1111\ \ 1 \\ 11011010 \\ +\ 10111010 \\ \hline \boxed{1}10010100 \end{array}$$

⇒ 결과가 음수이므로 2의 보수로 변환하여 구한다.

무시

$\therefore -38_{10} - 70_{10} = (-38_{10}) + (-70_{10}) = 10010100 \Rightarrow$ 2의 보수 : 01101100

$= -(1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0)$

$= -(64 + 32 + 0 + 8 + 4 + 0 + 0) = -108$

Overflow

❖ Sometimes result can't be represented with given number of bits

- Either too large magnitude of positive or negative performing arithmetic using fixed-width binary numbers, result is wider than the fixed bitwidth : overflow
- Adding two 4-bit regular binary numbers
  adding 1111+0001=10000 : 15+1=16 requires 5-bits
  easily detect overflow by looking carry-out bit of the adder
- Using two's complement numbers, detecting overflow is more complicated.
  Ex. 4-bit two's complement addition of 0111+0001 (7+1=8).
  But 4-bit two's complement can't represent number >7
  - 0111+0001 = 1000 WRONG answer, 1000 in two's complement is -8, not +8
- Adder/subtractor should indicate when overflow has occurred, so result can be discarded

Detecting Overflow: Method 1

❖ For two's complement numbers,

- overflow occurs when the two numbers' sign bits are the same but differ from the result's sign bit
- If the two numbers' sign bits are initially different, overflow is impossible
  Adding positive and negative can't exceed largest magnitude positive or negative

❖ (a) adding two positive number: 0111+0001=1000 incorrect: 7+1=8 but 1000 is –8 in two's complement, adding two positive numbers : overflow MSB=1
(b) adding two negative number :1111+10000=0111 carry-out 1 : incorrect : -1+-8=-9 but 0111 is +7, adding two negative numbers, overflow can be detected by checking whether MSB is 0 in the result
(c) adding a positive with a negative or negative with positive, can never result in overflow. Result: always be less negative than the most negative number or less positive than the most positive number

❖ Simple overflow detection circuit for 4-bit adder

- overflow = $a_3'b_3's_3 + a_3b_3s_3'$ Include "overflow" output bit on adder/subtractor

sign bits

	(a)	(b)	(c)
	0 1 1 1	1 1 1 1	1 0 0 0
+	0 0 0 1	1 0 0 0	0 1 1 1
	1 0 0 0	0 1 1 1	1 1 1 1
	overflow	overflow	no overflow
	(a) -8 ≠ 7+1=8	(b)7 ≠ -1+-8=-9	(c) -1 = -8 + 7= -1

If the numbers' sign bits have the same value, which differs from the result's sign bit, overflow has occurred.