기수법(Numeral system)

1. 밑수(기수, radix)
    1) 10진수 : {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
    2) 2진수 : {0, 1}
    3) 16진수 : {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f}

| 진수 변환 1

10진수 에서 2진수로
1. 10진수 수를 2의 거듭 제곱의 조합으로 만든다.
2. 중간에 빠진 지수 부분은 0을 이용해 표현한다.
3. 1과 0의 조합만으로 표현한다.

예) 43을 2진수로 변환
1. 43 = 32 + 11 = 32 + 8 + 3 = 32 + 8 + 2 + 1
2. 1×2^5(32) + 0×2^4(16) + 1×2^3(8) + 0×2^2(4) + 1×2^1(2) + 1×2^0(1)
3. 101011_2

| 진수 변환 2

5 4 3 2 1 0

$$\mathbf{101011_2}$$

2진수에서 10진수로

1. 2진수 수 위에 오른쪽 부터 0~5까지 적습니다.
2. 이 수를 지수로 이용해 2의 거듭제곱 합으로 표현
3. $1 \times 2^5(32) + 1 \times 2^3(8) + 1 \times 2^1(2) + 1 \times 2^0(1)$
4. 32 + 8 + 2 + 1 = 43

진수 변환 3

10진수	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16진수	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f

16진수에서 2진수로

1. 0~9까지는 10진수와 같다.
   $7 = 1 \times 2^2(4) + 1 \times 2^1(2) + 1 \times 2^0(1)$
2. 16진수 a는 10진수로 10이므로
   $a = 1010_2$
3. $b = 1011_2$, $c = 1100_2$, $d = 1101_2$, $e = 1110_2$, $f = 1111_2$

진수 변환 4

2진수에서 16진수로

1. 4개 단위로 쪼갠다.
2. 4개씩 쪼갠 단위로 16진수로 변환한다.

예) 1011010_2

1. $0101\ \ 1010_2$
2. $5 \qquad\qquad a_{16}$
3. $5a_{16}$

| 정수 표현

정수(Integer)의 표현

1. 일반적으로 1 바이트, 2 바이트, 4 바이트, 8 바이트에 저장
2. 부호 있는 정수(signed)와 부호 없는 정수(unsigned)로 나뉜다
3. 부호가 있는 경우 첫 bit가 부호를 나타냄(0 : 양수, 1 : 음수)

| 정수 표현 범위

Data type	Size	Range
byte	1 byte	0 ~ 255
sbyte	1 byte	-128 ~ 127
short	2 byte	-32,768 ~ 32,767
ushort	2 byte	0 ~ 65,535
int	4 byte	-2,147,483,648 ~ 2,147,483,647
uint	4 byte	0 ~ 4,294,967,296

양의 정수 표현

양의 정수 표현

1. 부호 비트는 0
2. 정수의 2진수로 표현 나머지는 0으로

예) 43을 1 바이트로 표현

43 = $0010\ 1011_2$ 이므로

0010 1011

0x2b

| 음의 정수 표현

음의 정수 표현

1. 부호 비트는 1
2. 정수를 2의 보수(two's complement)로 저장

43 → $0010\ 1011_2$ → $1101\ 0100_2$(1의 보수)
→ $1101\ 0101_2$(2의 보수)

음의 정수 표현

$$0010\ 1011_2$$

모든 비트를 반전

$$1101\ 0100_2$$ 1의 보수

1을 더한다

$$1101\ 0101_2$$ 2의 보수

음의 정수를 2의 보수로 표현하는 이유

1. 2의 보수를 사용하지 않는다면

    0000 0000과 1000 0000 모두 0을 표현

    1) 1 비트 낭비

    2) if 문으로 비교할 때 엉뚱한 결과가 나옴

| 음의 정수 표현

음의 정수를 2의 보수로 표현하는 이유

2. 정수의 뺄셈 과정

예) 43 – 25 = 18

1. 25를 -25로 변환
2. 43 + (-25)로 연산

$$
\begin{array}{rr}
 \mathbf{0010\ 1011_2} & 43 \\
+\ \ \mathbf{1110\ 0111_2} & -25 \\
\hline
\boxed{\cancel{\mathbf{1}}}\ \mathbf{0001\ 0010_2} & 18
\end{array}
$$

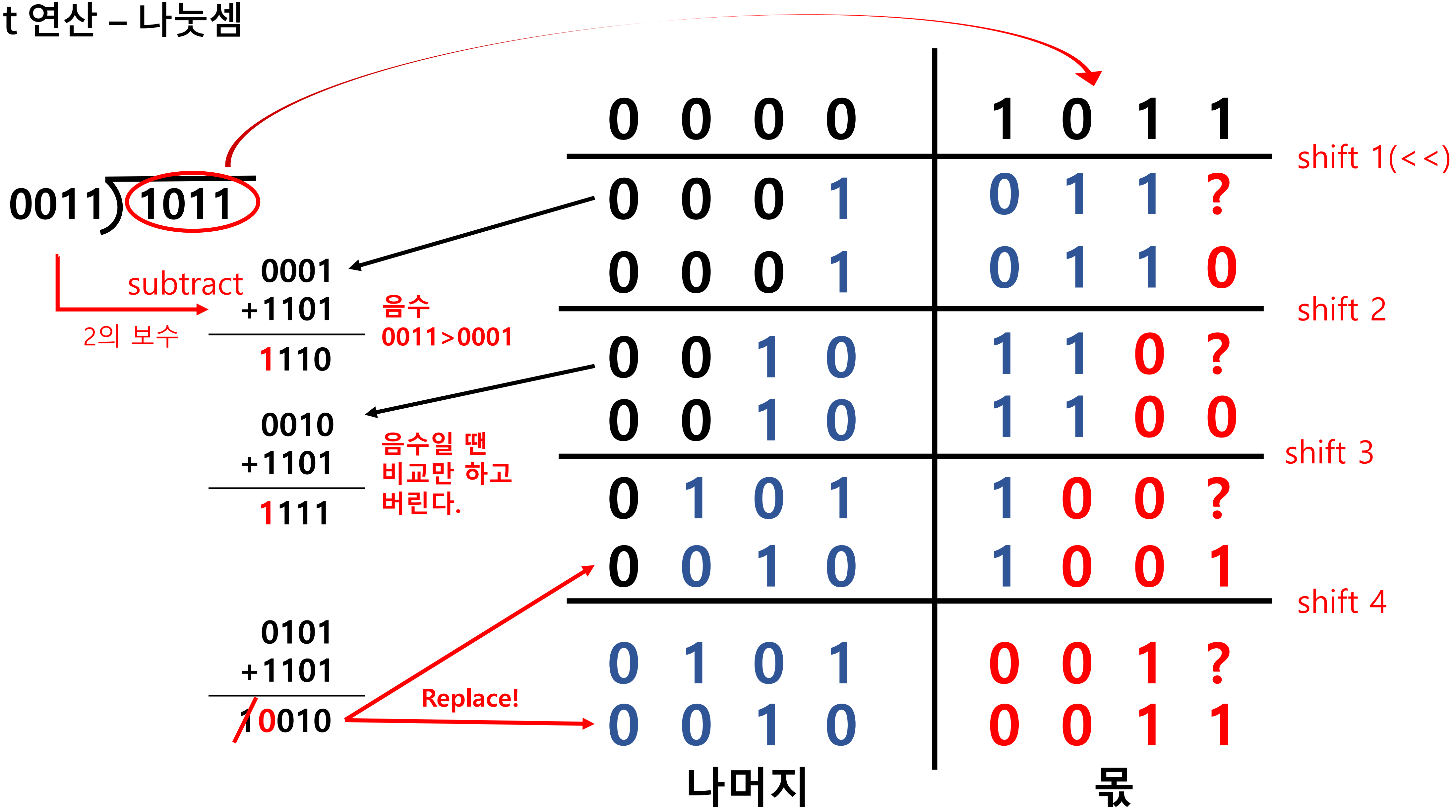
Carry는 버린다

4bit 연산 – 곱셈

```
  1010
x 0101
------
110010
```

	0 0 0 0	0 1 0 1	
+ 1010	1 0 1 0		shift 1(>>)
	0 1 0 1	0 0 1 0	shift 2
	0 0 1 0	1 0 0 1	
+ 1010	1 1 0 0		shift 3
	0 1 1 0	0 1 0 0	shift 4
	0 0 1 1	0 0 1 0	

| 4bit 연산 – 나눗셈


0011)1011
subtract
2의 보수
0001
+1101
1110
음수
0011>0001
0010
+1101
1111
음수일 땐
비교만 하고
버린다.
0101
+1101
10010
Replace!
0 0 0 0 | 1 0 1 1
shift 1(<<)
0 0 0 1 | 0 1 1 ?
0 0 0 1 | 0 1 1 0
shift 2
0 0 1 0 | 1 1 0 ?
0 0 1 0 | 1 1 0 0
shift 3
0 1 0 1 | 1 0 0 ?
0 0 1 0 | 1 0 0 1
shift 4
0 1 0 1 | 0 0 1 ?
0 0 1 0 | 0 0 1 1
나머지
몫

부동 소수점(floating-point)

부동 소수점(floating-point)

1. 단정도(single precision)
   1) 32 bit
   2) 부호(1 bit) + 지수(8 bit) + 가수(23 bit)
2. 배정도(double precision)
   1) 64 bit
   2) 부호(1 bit) + 지수(11 bit) + 가수(52 bit)

| 부동 소수점(floating-point)

	sign	exponent	mantissa
float	1 bit	8 bit	23 bit
double	1 bit	11 bit	52 bit

$$\pm 1.\mathrm{man} \times 2^{Exp\,-bias}$$

| 10진수 에서 2진수로 : 실수의 경우

1 0 -1 -2 -3

10.625

$$
\begin{aligned}
10.625 &= 8 + 2 + 0.5 + 0.125 \\
&= 2^3 + 2^1 + 2^{-1} + 2^{-3} \\
&= 1010.101_2
\end{aligned}
$$

$$\mathbf{1010.101_2}$$

| 정규화(Normalization)

정수 부분을 0이 아닌 자연수로 만드는 것

$\boxed{10}.625$ ➡ 1.0625×10^{1}

$\boxed{1062}.5$ ➡ 1.0625×10^{3}

정규화(Normalization)

2진수의 경우 0이 아닌 자연수는 1 밖에 없으므로
정규화를 하면 정수 부분은 반드시 1이다

$$\boxed{1010}.101_2 \rightarrow \boxed{1}.010101_2 \times 2^3$$

| 부동 소수점(floating-point)

$$\pm 1.\text{man} \times 2^{Exp-bias}$$

$$+1.010101_2 \times 2^3$$

1. 부호 : 0
2. Exp – bias = 3
3. man = 010101

bias

$$\text{bias} = 2^{n-1} - 1$$

n : 지수부의 비트 수

float의 경우 $2^{8-1} - 1 = 127$

bias

$$E_{re} = E_{mem} - \mathbf{bias}$$

bias

$$+1.010101_2 \times \mathbf{2^3}$$

$$\mathbf{3 = E_{mem} - 127}$$

$$\mathbf{E_{mem} = 130}$$

부동 소수점(floating-point)

$$\pm 1.\text{man} \times 2^{Exp-bias}$$

$$+1.010101_2 \times 2^3$$

1. sign : 0
2. Exp : 10000010
3. man : 010101...(나머지 비트는 0)

부동 소수점(floating-point)

float : 4 byte(32 bit)

$$+1.010101_2 \times 2^3$$

sign	exponent	mantissa
0	10000010	010101(나머지 비트는 0)
1 bit	8 bit	23 bit

| 부동 소수점(floating-point)

0.01은 가수부 23 bit로 온전히 표현할 수 없다.
→ 근사치로 표현하므로
0.01을 100번 더한 값은 정확히 1.0이 아니다.

실수 1.0 $+1.0_2 \times 2^0$

sign	exponent	mantissa
0	01111111	0000…(0이 23개)
1 bit	8 bit	23 bit

1.0 다음으로 표현할 수 있는 수는?

엡실론(epsilon)

$$+1.00\ldots\ldots.001_2 \times 2^0$$

sign	exponent	mantissa
0	01111111	0000… … 1
1 bit	8 bit	23 bit

22 bit까지 모두 0 마지막 bit만 1

$$+1.0_2 \times 2^0 + \mathbf{2^{-23}}$$

엡실론(epsilon)

엡실론(Epsilon)

- 1.0과 그 다음 표현할 수 있는 수 사이의 차이

$$2^{-23} = 1.192092896\text{e-}07$$

| 엡실론(epsilon)의 쓰임

실수 10.5와 그 다음 표현 가능한 수 사이의 차이는 얼마일까?

$10.5 = 8 + 2 + 0.5 = 2^3 + 2^1 + 2^{-1} = 1.0101_2 \times 2^3$

$$\boldsymbol{diff = 2^E \times epsilon}$$

$$= 2^E \times 2^{-23}$$

$$= \boldsymbol{2^{-20}}$$

= 9.53674e-07

| 엡실론(epsilon)의 쓰임

$$1.0 \le \frac{|num|}{2^E} < 2.0$$

$$2^E \le |num| < 2^{E+1}$$

$|num|$이 2^E로 근사하면

$$diff = 2^E \times epsilon$$
$$\approx |num| \times \text{epsilon}$$

Ex) $\text{num} = 1.0101_2 \times 2^3$이

$$1.0 \le \frac{1.0101_2 \times 2^3}{2^3} < 2.0$$

$$2^3 \le 1.0101_2 \times 2^3 < 2^4$$

$$8 \le 10.5 < 16$$

$$diff \approx |num| \times \text{epsilon}$$
$$\approx 10.5 \times 2^{-23}$$
$$\approx 1.25169\text{e-}06$$

엡실론(epsilon)의 쓰임

실수 비교
- 실수는 if(a==b)처럼 직접 비교하면 안된다.
- 그러므로 a, b를 비교하는 함수를 작성해 비교한다.

1. Absolute comparison
2. Relative comparison

| 엡실론(epsilon)의 쓰임

1. Absolute comparison

def Is_equal1(a, b):
return |a - b| <= 1e-10

두 수의 차이가 1e-10보다 작다면 같다고 생각해도 무방
여기서 1e-10은 임의로 정한 충분히 작은 수

| 엡실론(epsilon)의 쓰임

2. Relative comparison

def Is_equal2(a, b, allowed=0):

    ep ← 2^{-23}

    diff = |a – b|

    return diff <= max(|a|, |b|) * ep * $2^{allowed}$

$$diff \approx |num| \times \mathbf{epsilon}$$

→ 두 수 중 큰 수에 epsilon을 곱하면

→ 그 수와 그 이웃하는 수 차이의 근사값