연산자

2020학년도 1학기

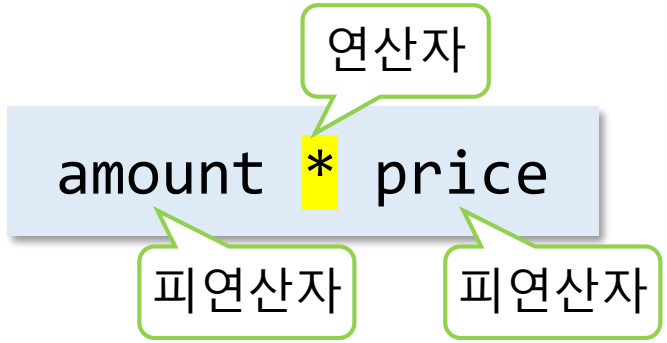
강원도립대학교 ICT드론과

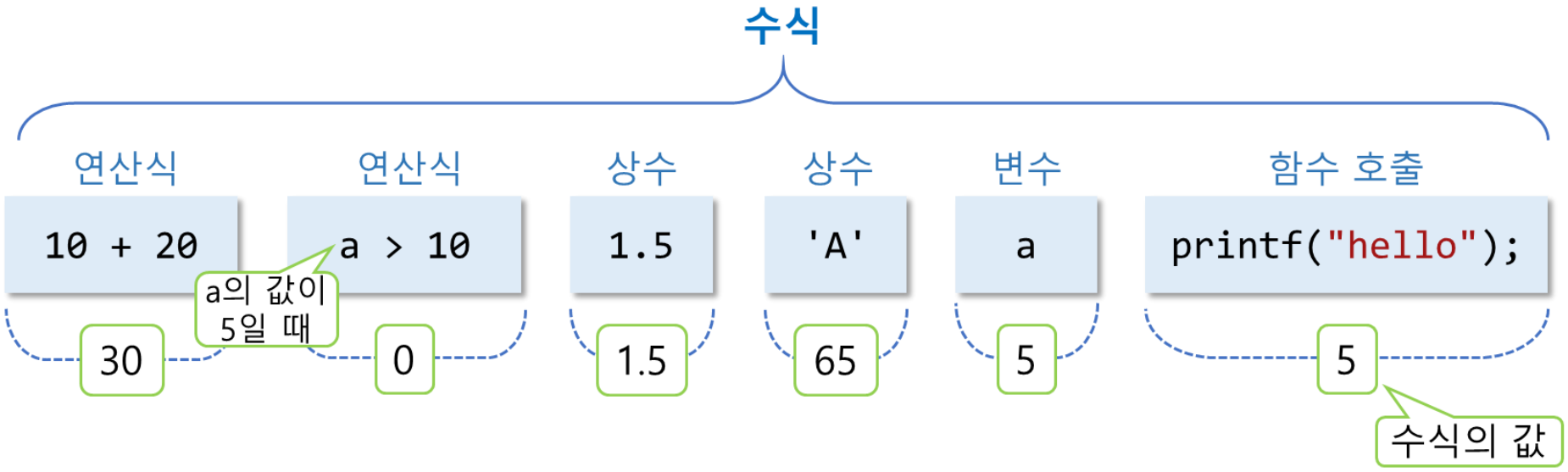
목차

- 연산자의 기본 개념
  - 수식
  - 연산자와 피연산자
- 연산자의 종류
  - 산술 연산자
  - 증감 연산자
  - 대입 연산자
  - 관계 연산자
  - 논리 연산자
  - 비트 연산자
  - 그 밖의 연산자
- 연산자의 우선순위와 결합 규칙

연산자의 기본 개념

- **수식(expression)** : 연산자와 피연산자의 조합
  - **연산자(operator)** : 연산에 사용되는 기호
  - **피연산자(operand)** : 연산의 대상이 되는 값
  - 모든 수식에는 반드시 값이 있다.
  - 수식의 평가 : 수식의 값을 구하는 것





피연산자의 개수로 분류한 연산자

종류	연산자의 의미	연산자	
단항 연산자	1개의 피연산자	+a -a a++ ++a a-- --a !a &a ~a sizeof a	
이항 연산자	2개의 피연산자	산술	a+b a-b a*b a/b a%b
		대입	a=b a+=b a-=b a*=b a/=b a%=b a>>=b a<<=b a&=b a|=b a^=b
		관계	a>b a<b a>=b a<=b a==b a!=b
		논리	a&&b a||b
		비트	a&b a|b a^b a<<b a>>b
삼항 연산자	3개의 피연산자	a?b:c	

연산자의 기능에 따라 분류한 연산자

연산자의 종류	연산자
산술 연산자	a+b a-b a*b a/b a%b +a -a
증감 연산자	a++ ++a a-- --a
관계 연산자	a>b a<b a>=b a<=b a==b a!=b
논리 연산자	a&&b a||b !a
비트 연산자	a&b a|b a^b ~a a<<b a>>b
대입 연산자	a=b a+=b a-=b a*=b a/=b a%=b a&=b a|=b a^=b a<<=b a>>=b
멤버 접근 연산자	*a &a a[b] a.b a->b
그 밖의 연산자	a?b:c a,b sizeof a (type)a

산술 연산자 (1/2)

산술 연산자	의미	사용 예	연산의 결과
+a	플러스(부호)	+10	10
-a	마이너스(부호)	-10	-10
a + b	더하기	10 + 3	13
a - b	빼기	10 - 3	7
a * b	곱하기	10 * 3	30
a / b	나누기	10 / 3	3
a % b	나머지 구하기	10 % 3	1

산술 연산자 (2/2)

- 부호 연산자 +, - : 단항 연산자

예제 4-1, 4-2

```
short a = 10;
printf("%d", -a);    // 수식의 값은 -10
```

- 나누기 연산자(/) : 피연산자가 둘 다 정수인 경우, 몫도 정수가 된다.

```
int result1 = 10 / 3;          // 수식의 값은 3
```

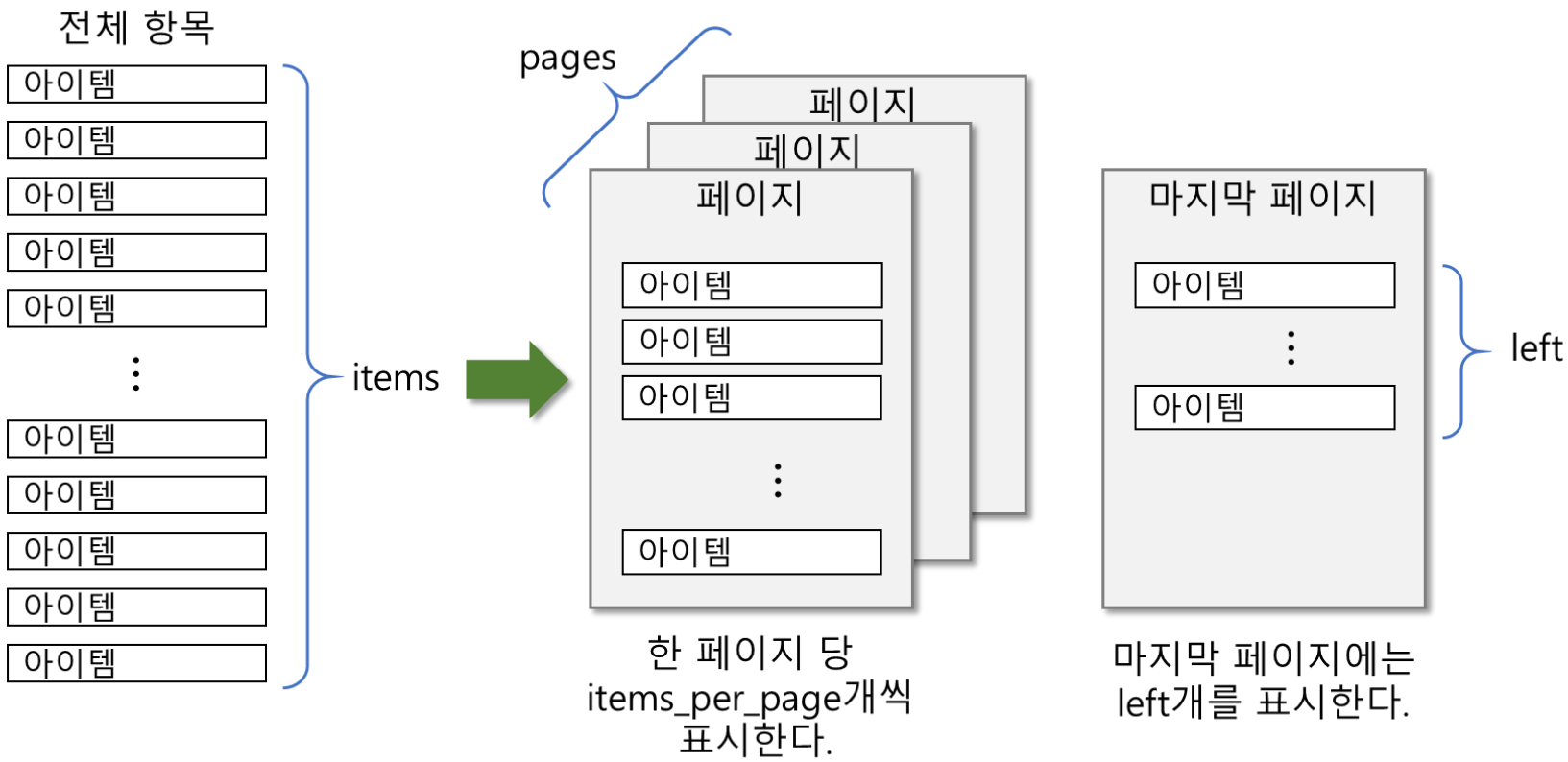
- 나머지 연산자(%) : 피연산자가 모두 정수인 경우에만 사용

```
int result2 = 10 % 3;          // 수식의 값은 1
```

나머지 연산자의 활용

```
pages = items / items_per_page;
left = items % items_per_page;
```

예제 4-3



피연산자의 형 변환 규칙

예제 4-4

① 피연산자 중에 double형이 있으면, 나머지 피연산자를 double형으로 변환한다.

```
1.25 * 3          // double * double로 처리
1.25 + 0.5F       // double + double로 처리
```

② 피연산자 중에 float형이 있으면, 나머지 피연산자(정수형)를 float 형으로 변환한다.

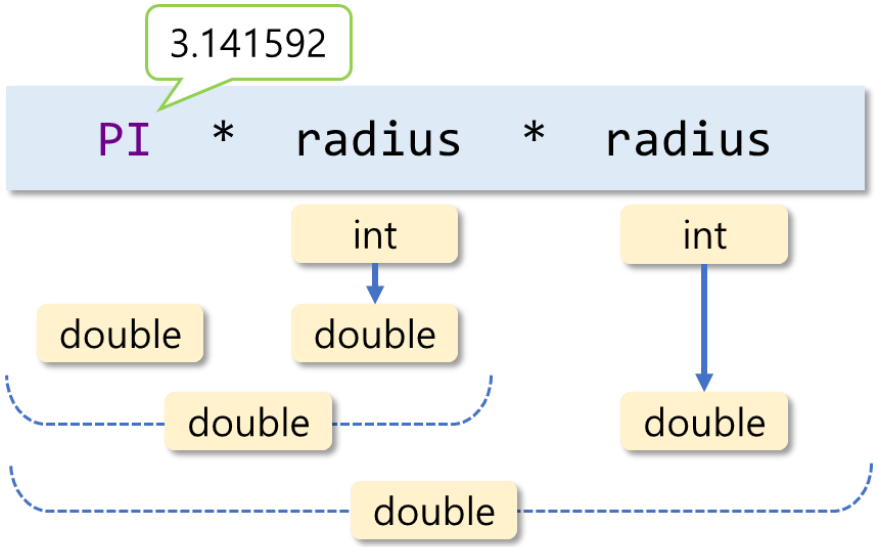
```
1.25F / 2         // float / float로 처리
```

③ 피연산자가 둘다 정수형이면 우선 승격시킨다.

```
short a = 1000, b = 2000;
a * b     // int * int로 처리
```

자동 형 변환

- **암시적인 형 변환**
  - 정수와 실수의 혼합 연산 시 수행되는 형 변환은 자동으로 처리



- 단항 연산자에서는 항상 정수의 승격이 일어난다.

```
short a = 10;
printf("%d", sizeof(-a)); // 4
```

int형

- int형보다 크기가 작은 경우에는 항상 int형으로 승격된다.

```
short a = 1000, b = 2000;
printf("%d", sizeof(a * b)); // 4
```

int형

증감 연산자 (1/2)

- 변수의 값을 1만큼 증가시키거나 감소시키는 단항 연산자
- 증감 연산자의 의미

구분	증감 연산자	수식의 값
전위형	++a	증가된 변수 a의 값
	--a	감소된 변수 a의 값
후위형	a++	증가되기 전 변수 a의 값
	a--	감소되기 전 변수 a의 값

증감 연산자 (2/2)

- 전위형과 후위형

예제 4-5

후위형

```
int index = 0;
index++;
```

전위형

```
int index = 0;
++index;
```

단일 문장인 경우 두 문장의 결과가 동일하다.

index가 1만큼 증가된다.

후위형

```
int index = 0;
int current = index++;
```

수식의 값은 증가 전 index의 값

0

전위형

```
int index = 0;
int current = ++index;
```

수식의 값은 증가 후 index의 값

1

대입 연산자

- 연산자의 좌변에 있는 변수(l-value)에 우변의 값(r-value)을 저장

```
int price = 3000, amount = 2;
int total = 0;
int count = 0;
price = 1000;                  // price 변수에 리터럴 상수의 값을 저장한다.
total = amount * price;        // total 변수에 amount * price 연산의 결과 값을 저장한다.
count = printf("hello");       // count 변수에 printf 함수의 리턴 값을 저장한다.
```

- 대입 연산자의 좌변에는 변수만 올 수 있다.

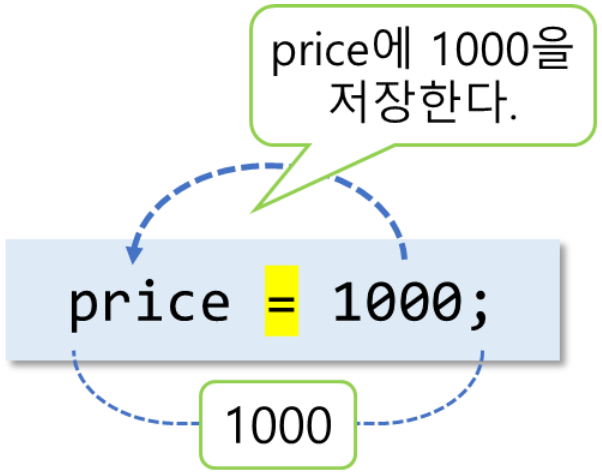
```
10 = x;              // 10은 변수가 아니므로 컴파일 에러
a + 1 = a;           // a + 1은 변수가 아니므로 컴파일 에러
printf("abc") = 10; // printf("abc")는 변수가 아니므로 컴파일 에러

const double PI = 3.141592;
PI = 3.14;           // PI는 const 변수이므로 컴파일 에러
```

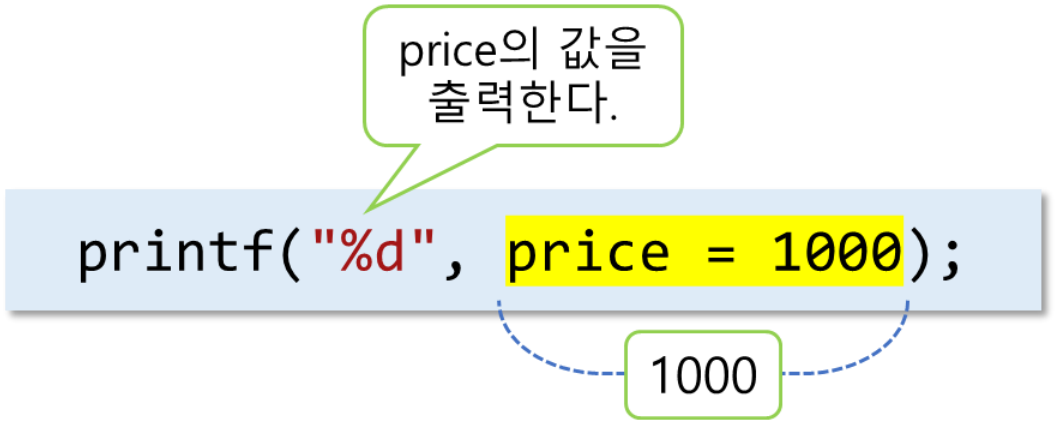
대입 연산식의 값

- 대입 연산자의 좌변에 있는 변수의 값이 대입 연산식의 값

예제 4-6



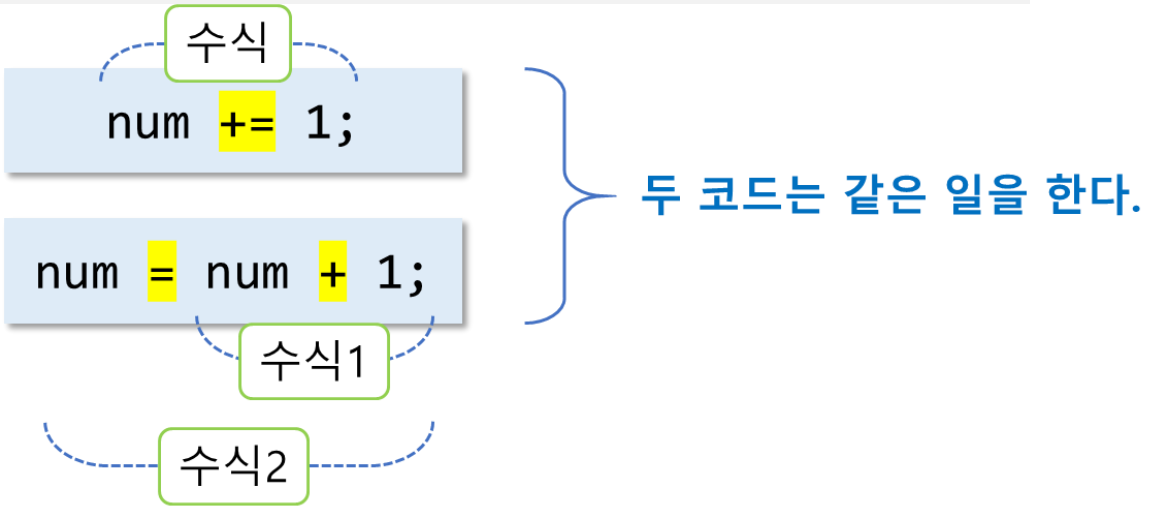
대입 연산식의 값



대입 연산식의 값을
다른 수식에 이용할 수 있다.

복합 대입 연산자

- 좌변의 변수를 피연산자로 이용해서 연산을 수행하고 연산의 결과를 다시 좌변의 변수에 대입한다.



복합 대입 연산자	의미	복합 대입 연산자	의미
a += b	a = a + b	a &= b	a = a & b
a -= b	a = a - b	a |= b	a = a | b
a *= b	a = a * b	a ^= b	a = a ^ b
a /= b	a = a / b	a <<= b	a = a << b
a %= b	a = a % b	a >>= b	a = a >> b

복합 대입 연산자의 활용

예제 4-7

[예제 4-3]

```
left = items % items_per_page;
```

남은 항목의 개수를
별도의 변수에 저장

items는 바뀌지 않는다

[예제 4-7]

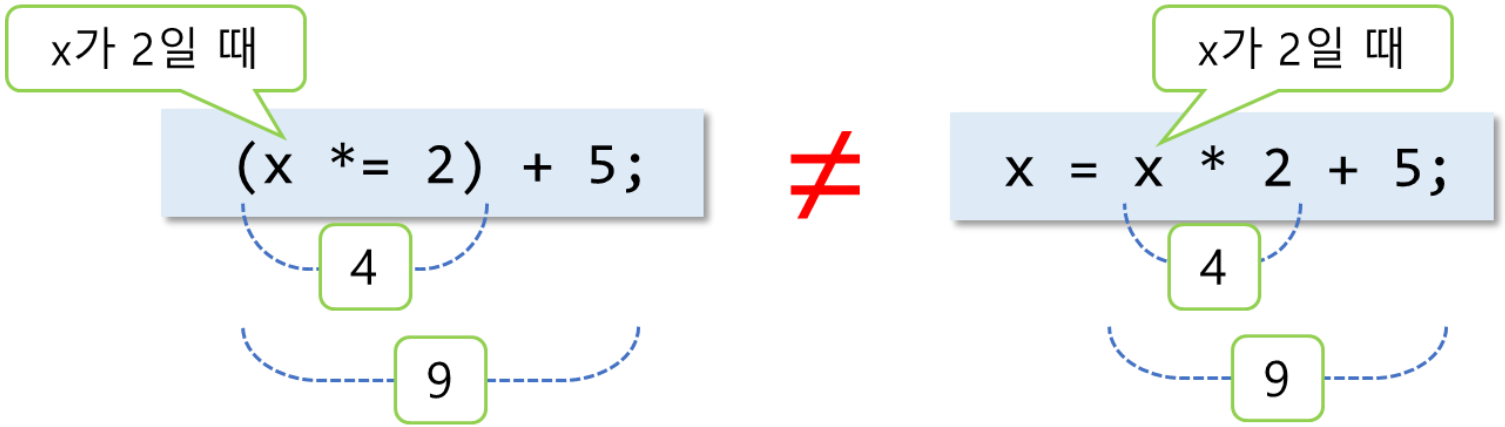
```
items %= items_per_page;
```

남은 항목의 개수를
items에 다시 저장

items가 바뀐다.

복합 대입 연산자의 우선순위

x *= 7로 처리된다.



연산 후 x는 4이다.

연산 후 x는 9이다.

관계 연산자

- 두 수의 값을 비교하기 위한 연산자
- 관계 연산식의 값은 항상 참 또는 거짓
  - C에서 참(true)은 1이고, 거짓(false)은 0이다.

예제 4-8

관계 연산자	의미	a = 1, b = 2일 때 연산의 결과
a > b	a가 b보다 큰가?	0
a >= b	a가 b보다 크거나 같은가?	0
a < b	a가 b보다 작은가?	1
a <= b	a가 b보다 작거나 같은가?	1
a == b	a가 b와 같은가?	0
a != b	a가 b와 다른가?	1

관계 연산자 사용 시 주의 사항

- 두 값이 같은지 비교할 때는 =가 아니라 ==를 이용해야 한다.

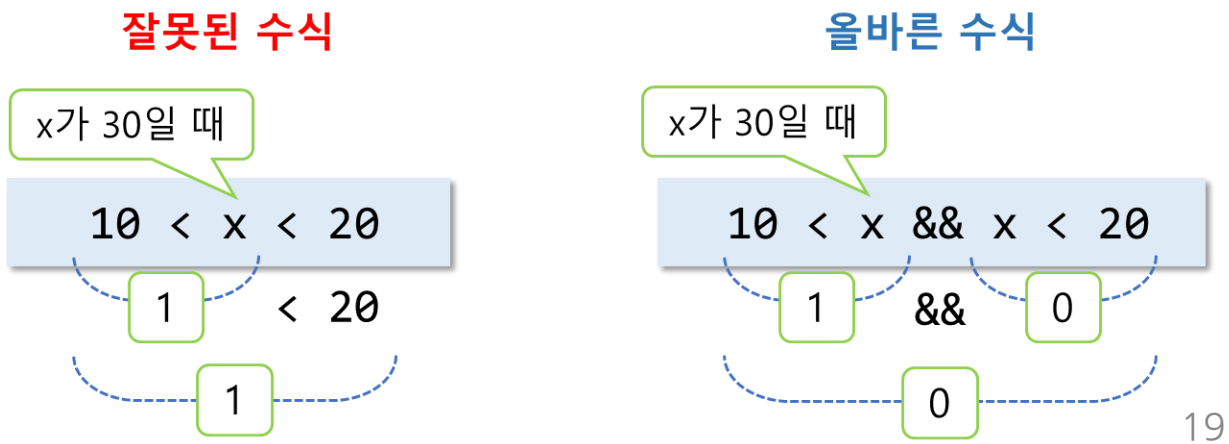
```
if (num = 1) // num에 1을 대입하므로 항상 참
    printf("이 문장은 항상 출력됩니다.");
```

- 실수를 비교할 때는 오차를 고려해야 한다.

```
if (fabs(result - expected) <= FLT_EPSILON)
    printf("두 수가 같습니다.\n");
```

- 10 < x < 20과 같은 수식을 사용해서는 안된다.

x가 10과 20 사이의 값인지 검사한다.



참, 거짓의 판단

- C에서는 수식이 참인지 거짓인지 판단할 때, 0이 아닌 값은 참으로 간주한다.

y가 0이 아닌가?

```
if (y != 0)
    result = x / y;
```

=

y가 참인가?

```
if (y)
    result = x / y;
```

y가 0인가?

```
if (y == 0)
    printf("에러");
```

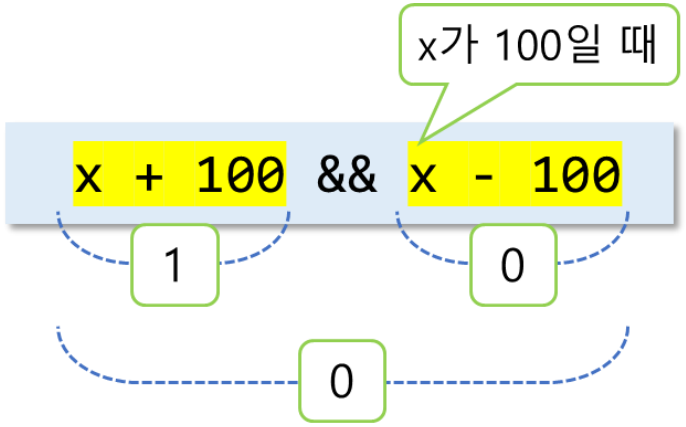
=

y가 거짓인가?

```
if (!y)
    printf("에러");
```

논리 연산자

- 참과 거짓을 이용한 논리 연산 기능
- 연산의 결과가 항상 참(1) 또는 거짓(0)
- 피연산자가 0이 아니면 참으로 간주한다.



논리 연산자	부울 대수	의미
a && b	논리 AND	a와 b가 둘 다 0이 아니면 1 a와 b중 하나만 0이면 0
a || b	논리 OR	a와 b중 하나만 0이 아니면 1 a와 b가 둘 다 0이면 0
! a	논리 NOT	a가 0이면 1, a가 0이 아니면 0

논리 연산의 결과

예제 4-9

a	b	a && b	a || b	!a	!b
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	0

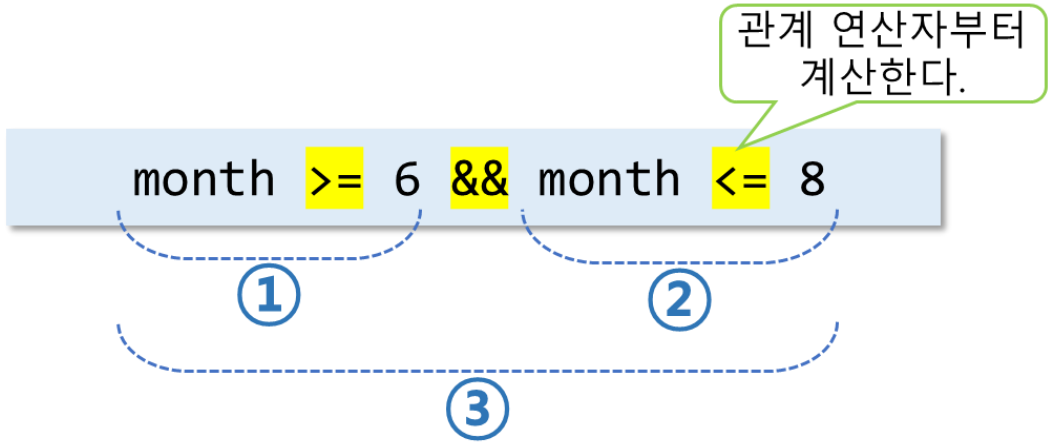
```
month >= 6 && month <= 8      // month가 성수기(6~8)에 해당하는지 검사하는 수식
```

```
month < 6 || month > 8        // month가 0보다 작은 경우, 12보다 큰 경우는 없다고 가정한다.
```

```
! (month >= 6 && month <= 8) // month < 6 || month > 8와 같다.
```

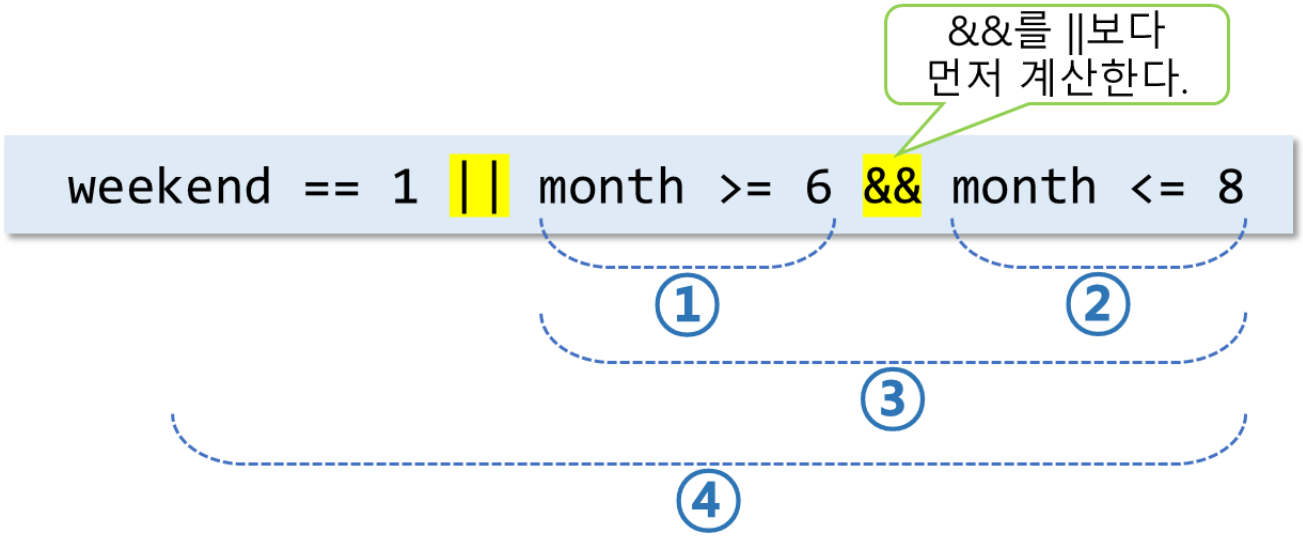
논리 연산자 사용시 주의 사항

- 관계 연산자과 함께 사용하면 관계 연산자부터 수행된다.




```
(month >= 6) && (month <= 8)
라는 의미
```

- && 연산자가 || 연산자보다 우선순위가 높다.




```
weekend == 1 || (month >= 6 && month <= 8)
라는 의미
```

논리 연산자의 단축 계산

- 'expr1 && expr2'에서 expr1이 거짓이면 expr2는 평가되지 않는다. 즉, expr2는 expr1이 참일 때만 평가된다.

```
a > 0 && printf("abc") == 3  // a <= 0이면 printf("abc") == 3은 수행되지 않는다.
```

- 'expr1 || expr2'에서 expr1이 참이면 expr2는 평가되지 않는다. 즉, expr2는 expr1이 거짓일 때만 평가된다.

```
a > 0 || --b < 0              // a > 0이면 --b < 0는 수행되지 않는다.
```

비트 연산자

- 피연산자의 각 비트 단위로 수행되는 연산자

구분	비트 연산자	의미
비트 논리 연산자	a & b	a와 b의 각 비트 단위로 논리 AND 연산
	a | b	a와 b의 각 비트 단위로 논리 OR 연산
	a ^ b	a와 b의 각 비트 단위로 논리 XOR 연산
	~a	a의 각 비트 단위로 논리 NOT 연산
비트 이동 연산자	a << b	a의 각 비트를 b개만큼 왼쪽으로 이동
	a >> b	a의 각 비트를 b개만큼 오른쪽으로 이동

비트 논리 연산자

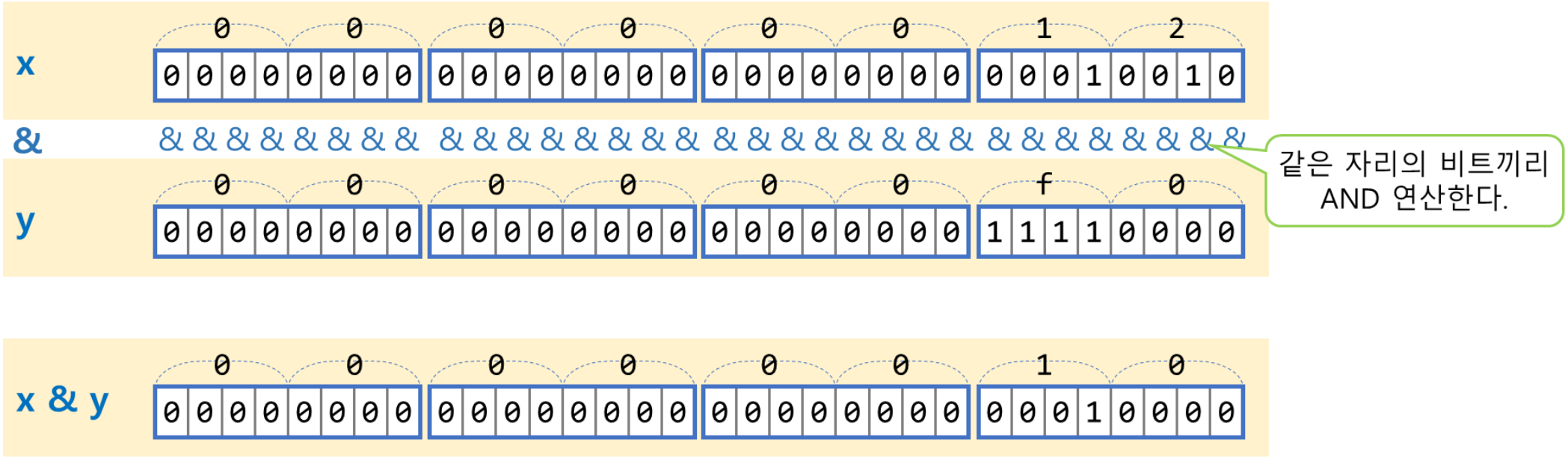
- 논리 연산자가 피연산자의 전체 값으로 연산을 수행하는 반면에, 비트 논리 연산자는 각 비트에 대하여 연산을 수행한다.
- &, |, ^ 연산자는 피연산자의 데이터형이 일치하지 않으면 형 변환을 수행하여 같은 형으로 만든 후, 피연산자의 각 비트에 대하여 비트 논리 연산을 수행한다.

a의 비트	b의 비트	a & b의 비트	a | b의 비트	a ^ b의 비트	~a의 비트
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

비트 AND 연산 (1/2)

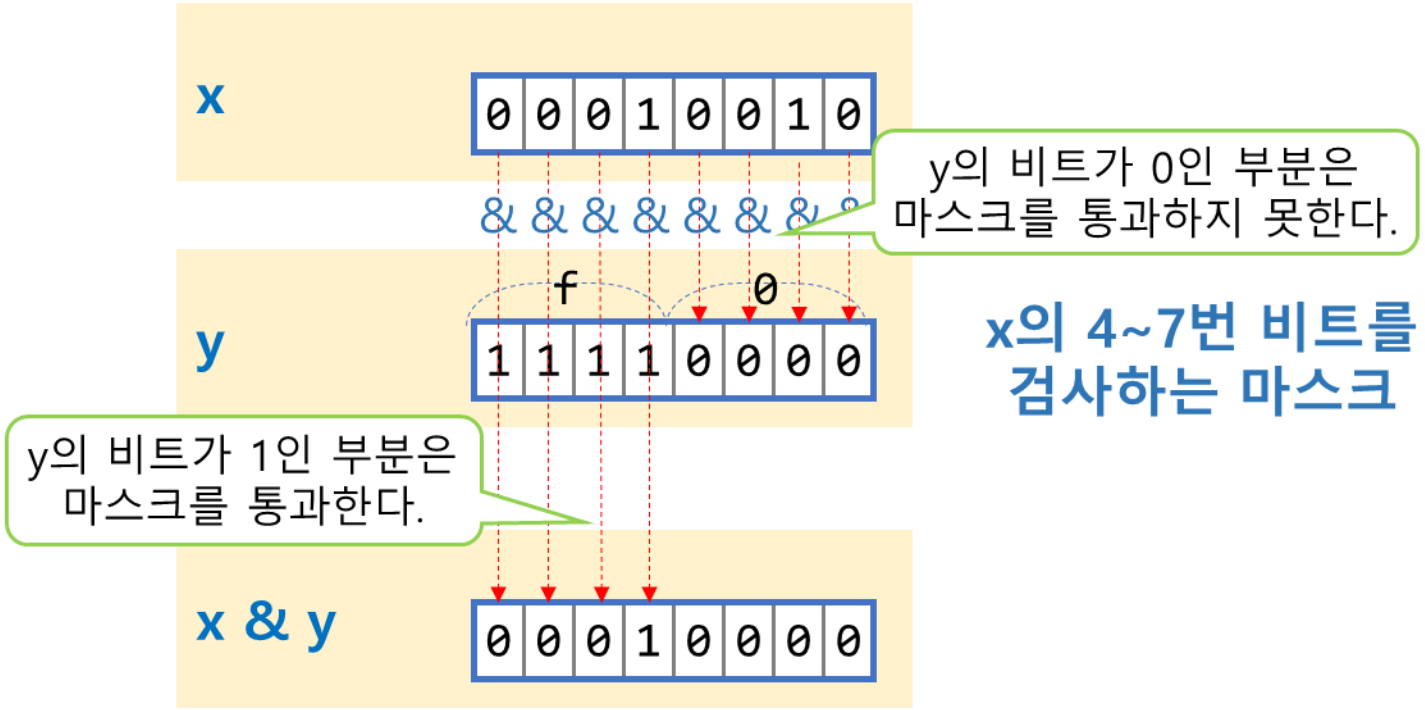
```
unsigned short x = 0x12;
unsigned short y = 0xf0;
x & y
```

수식의 값은
0x00000010



비트 AND 연산 (2/2)

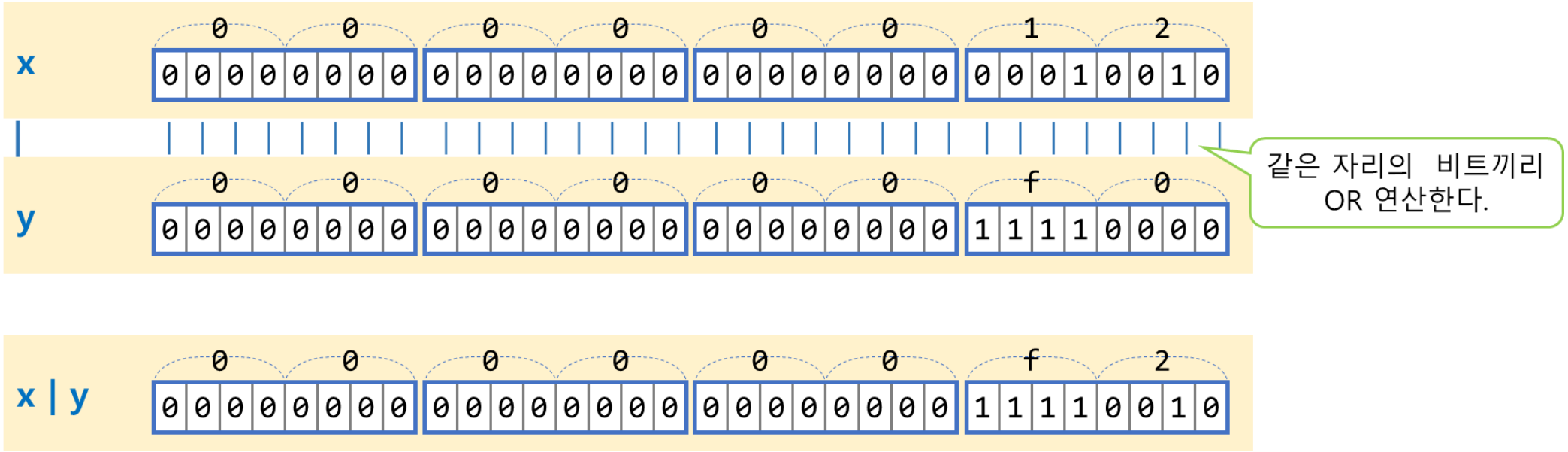
- **비트마스크(bitmask)** 또는 **마스크(mask)**
  - 비트 논리 연산에서 이용되어 특정 비트 값을 조작하기 위한 목적의 데이터
- x와 y를 비트 AND 연산하면 x의 값 중에 y의 비트가 1인 부분의 값만 유지되고, 나머지 비트는 모두 0이 된다.



비트 OR 연산 (1/2)

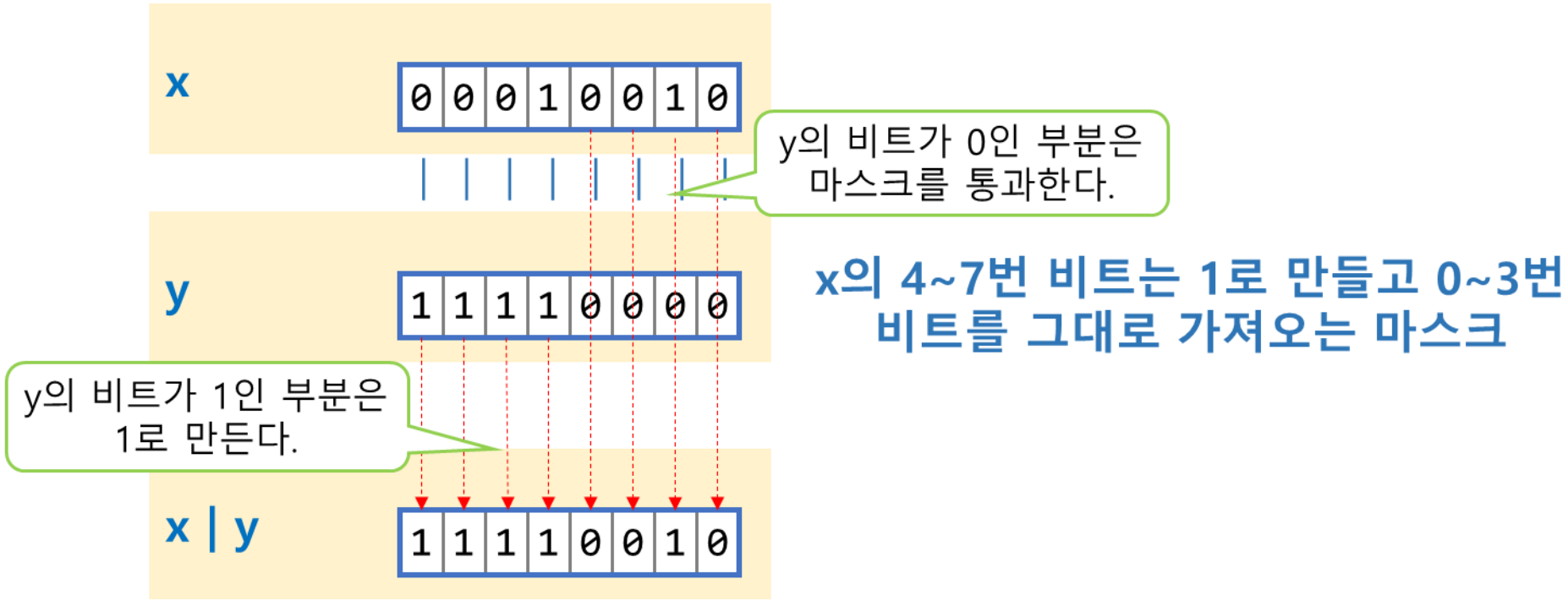
```
unsigned short x = 0x12;
unsigned short y = 0xf0;
x | y
```

수식의 값은 0x000000f2



비트 OR 연산 (2/2)

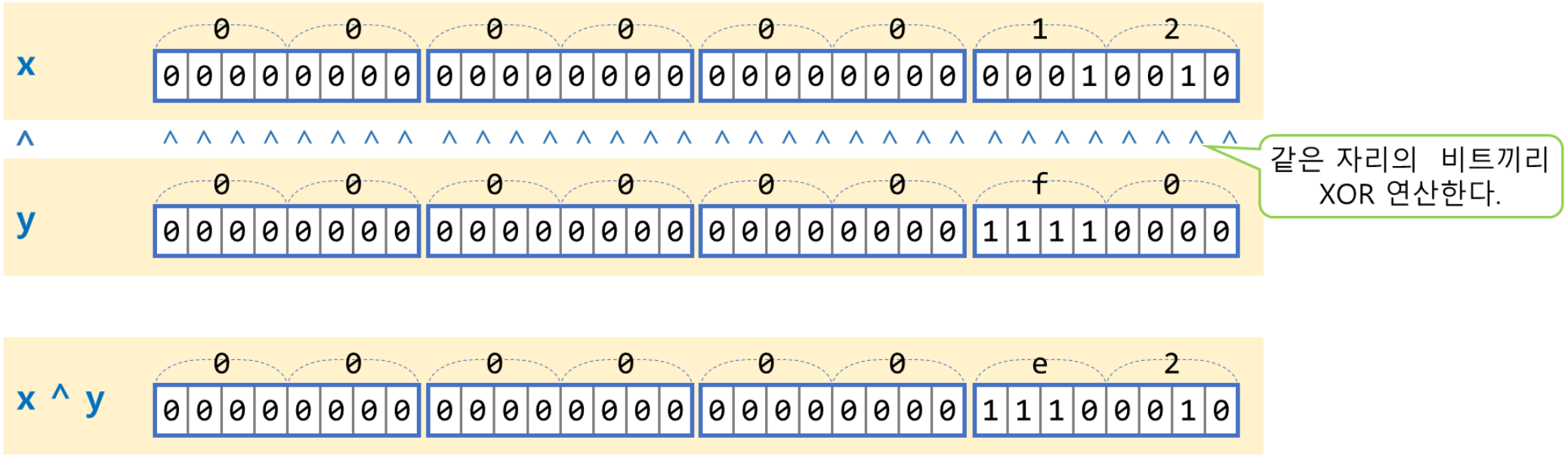
- x와 y를 비트 OR 연산하면 x의 값 중에 y의 비트가 0인 부분의 값만 유지되고, y의 비트가 1인 부분은 모두 1이 된다.



비트 XOR 연산 (1/2)

```
unsigned short x = 0x12;
unsigned short y = 0xf0;
x ^ y
```

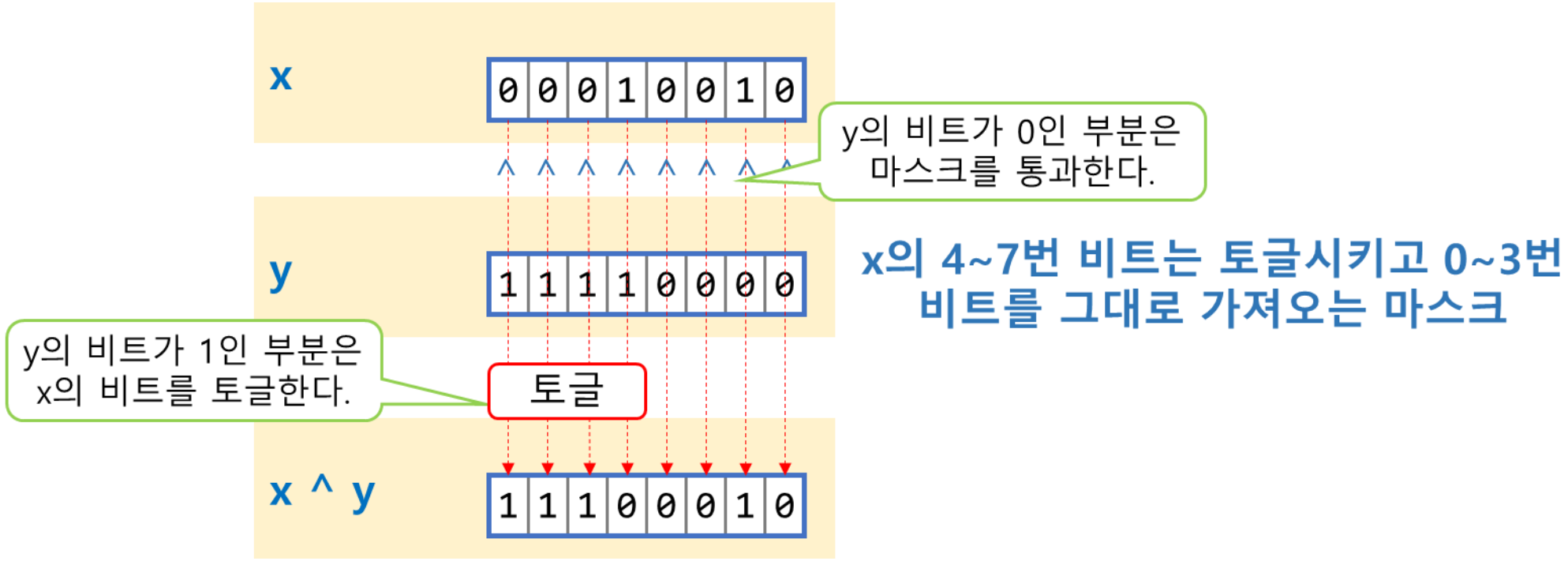
수식의 값은 0x000000e2



같은 자리의 비트끼리 XOR 연산한다.

비트 XOR 연산 (2/2)

- x와 y를 비트 XOR 연산하면 x의 값 중에 y의 비트가 1인 부분은 토글되고, y의 비트가 0인 부분의 값은 유지된다.



비트 NOT 연산

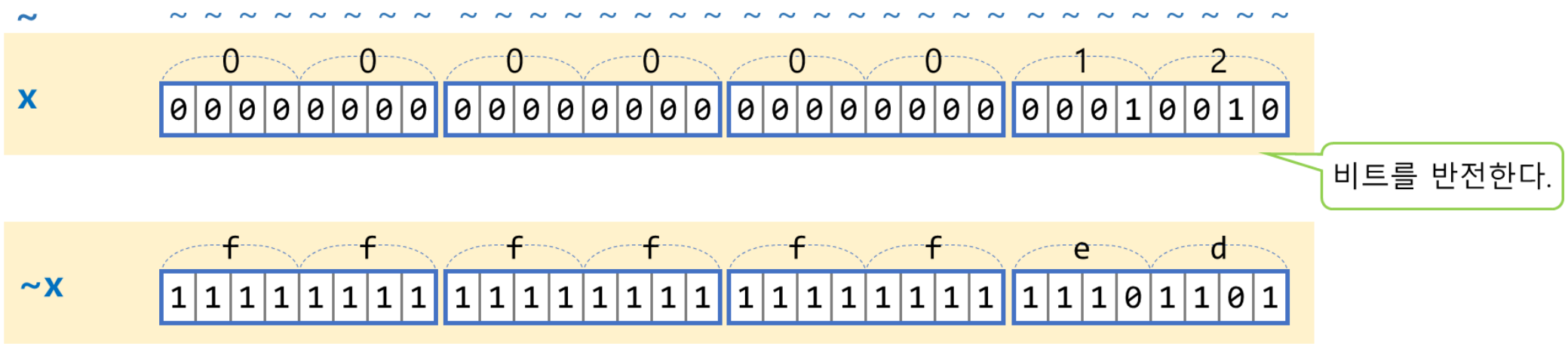
- 0은 1로, 1은 0으로 반전한다.

예제 4-10
비트논리 연산자의 사용 예

```
unsigned short x = 0x12;

~x
```

수식의 값은 0xffffffed



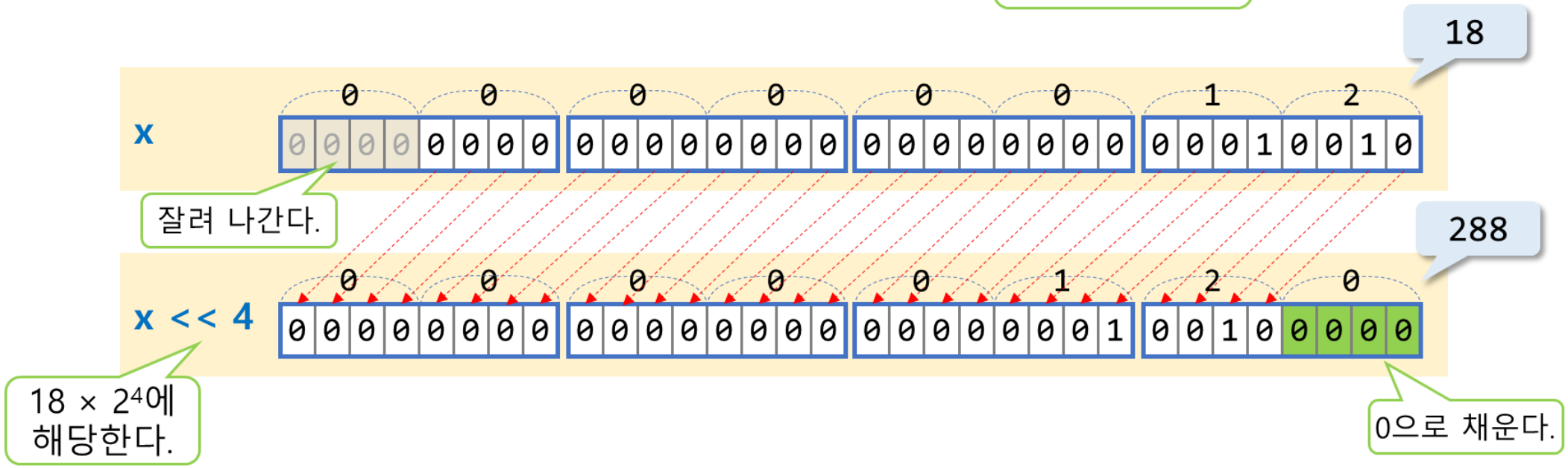
비트 이동 연산자

- 좌변에 있는 피연산자의 비트들을 우변의 피연산자가 지정하는 만큼 왼쪽으로 또는 오른쪽으로 이동(shift)한다.
- **비트 왼쪽 이동(<<) 연산자**
  - 비트들을 왼쪽으로 이동
  - 왼쪽으로 밀려난 비트는 사라지고 오른쪽 빈 자리에 0을 채운다.
  - n 비트 왼쪽 이동은 2^n을 곱하는 것과 같다.
- **비트 오른쪽 이동(>>) 연산자**
  - 비트들을 오른쪽으로 이동
  - 오른쪽으로 밀려난 비트는 사라지고 왼쪽 빈 자리에 부호 비트를 채운다.
  - 연산자의 좌변이 부호 없는 정수형이면 왼쪽 빈 자리를 0으로 채운다.
  - n 비트 오른쪽 이동은 2^n 으로 나누는 것과 같다.

비트 왼쪽 이동

```
int x = 0x12;
int y = x << 4;
```

수식의 값은
0x00000120

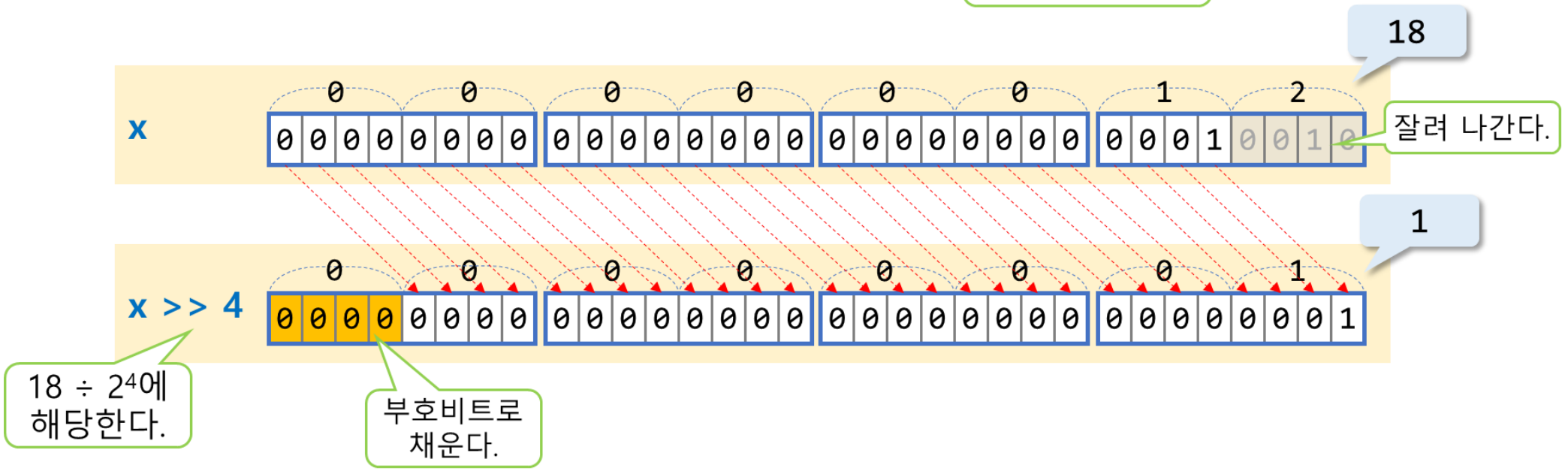


비트 오른쪽 이동

예제 4-11

```
int x = 0x12;
int z = x >> 4;
```

수식의 값은 0x00000001



조건 연산자

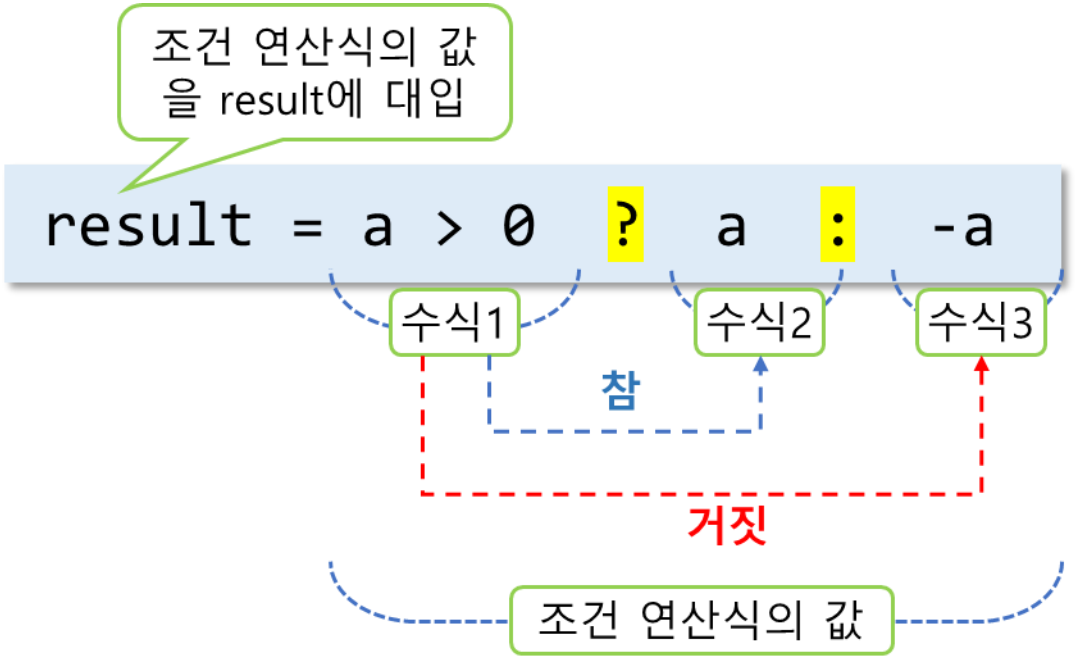
- 피연산자가 3개인 삼항 연산자

형식 수식1 ? 수식2 : 수식3

사용예

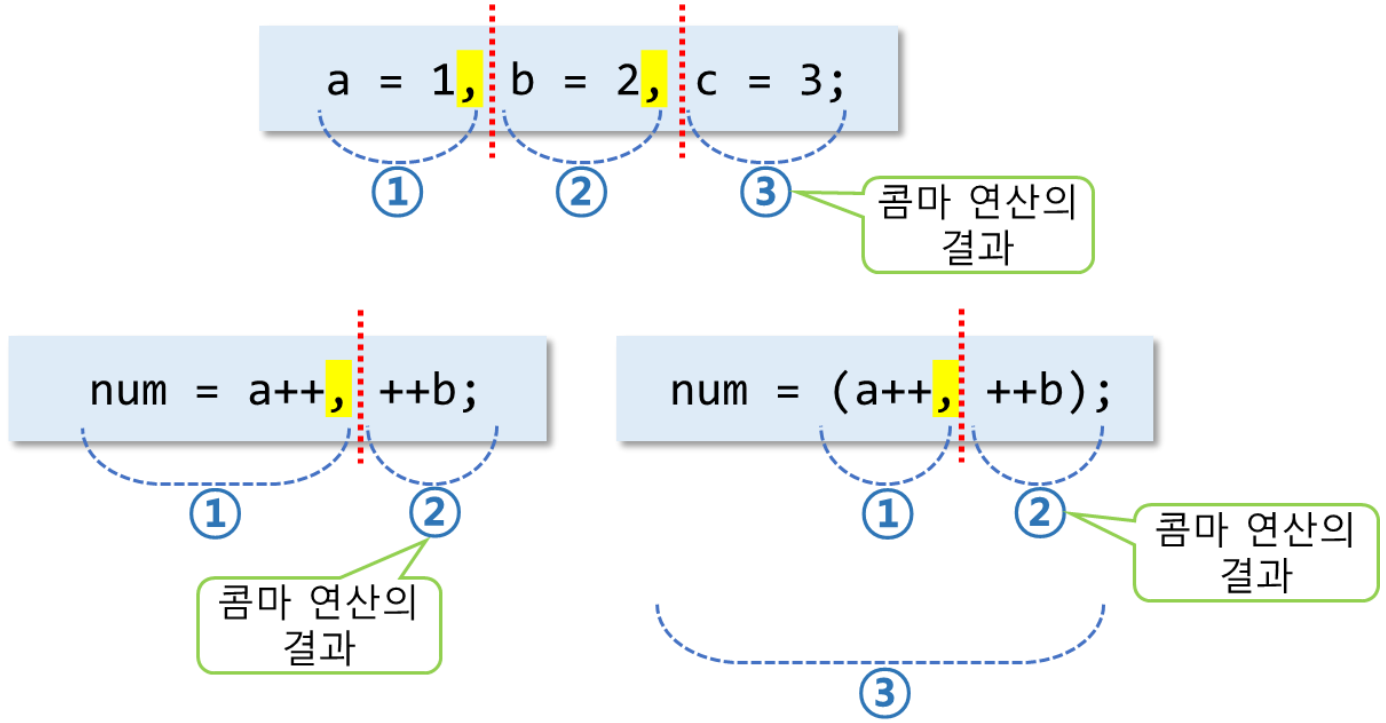
```
a > 0 ? a : -a
a > b ? a : b
num % 2 ? printf("odd"):printf("even")
```

예제 4-12



콤마 연산자

- 수식의 값을 왼쪽부터 계산해서 마지막으로 계산한 오른쪽 수식의 값이 연산의 결과가 된다.
- 여러 수식을 한 문장으로 연결할 때 주로 사용된다.



형 변환 연산자

- 명시적인 형 변환

예제 4-15

형식	(데이터형) 수식
사용예	(double) 0 (int) price * rate (double) 10 / (double) 20

```
int result1, result2;
result1 = (int)(1.5 + 3.8);          // 1.5 + 3.8을 먼저 수행한 다음 (int) 5.3
result2 = (int)1.5 + (int)3.8;       // (int)1.5와 (int)3.8을 먼저 수행한 다음 1 + 3
```

연산자의 우선순위

단항 연산자 > 산술 연산자 > 관계 연산자 > 논리 연산자 > 대입 연산자 > 콤마 연산자

- '변수 = 수식'의 형태인 경우 항상 수식의 값을 먼저 계산한다.

```
result = a + b * c;          // =의 우변에 있는 a + b * c를 먼저 계산한다.
```

- 관계 연산자는 논리 연산자보다 우선순위가 높다.

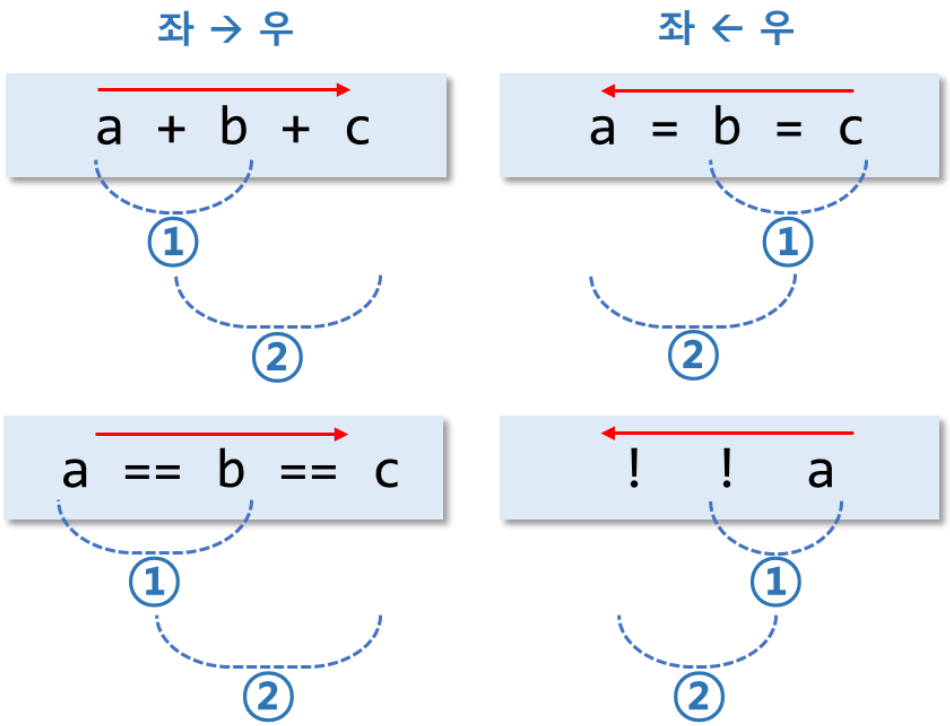
```
a > b && a < c               // (a > b) && (a < c)의 의미
```

- 산술 연산자는 관계 연산자나 논리 연산자보다 우선순위가 높다.

```
a + 100 == b * 3             // (a + 100) == (b * 3)의 의미
```

연산자의 결합 규칙

- 대부분의 연산자는 좌 → 우 방향으로 결합하고 단항 연산자와 대입 연산자는 우 → 좌 방향으로 결합한다.



예제 4-16

연산자

- 학습과제
  - 4장 연산자 교재 내용 이해하고 연습문제 풀이하여 그 결과를 메일로 보내기
  - Exercise 4
    - 페이지 174~177페이지

  - 메일 주소 : wykim@gw.ac.kr
  - 메일 제목 : 학번_이름_5차~6차 과제