



CHUNGNAM NATIONAL UNIVERSITY



# 어셈블리어 프로그래밍

강의 2 : 2장. 정보의 표현 및 처리 II

2.1, 2.2, 2.3 정수의 표현 및 연산

2014년 9월 11일

<http://eslab.cnu.ac.kr>

# 데이터 타입변환 casting 하기

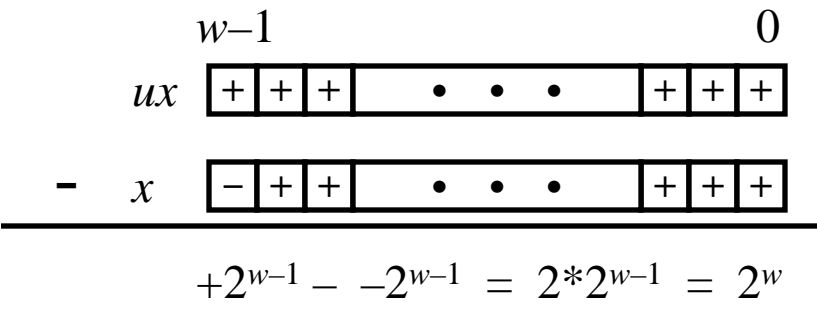
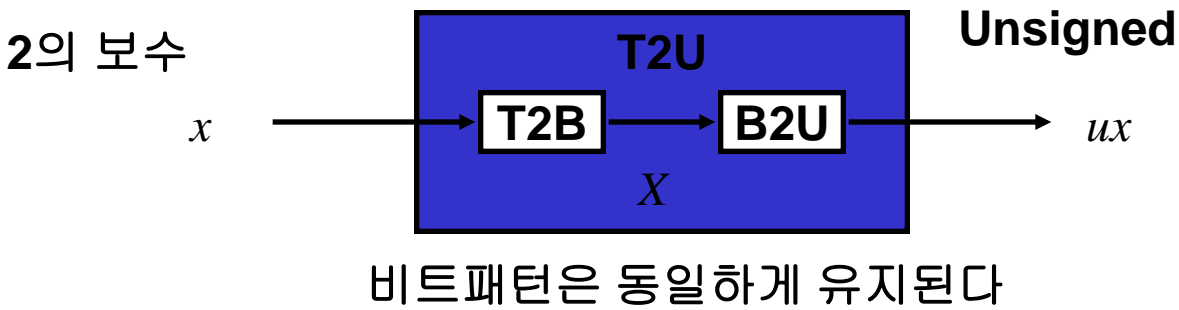
C 언어에서는 signed로부터 unsigned로의 변환을 허용한다

```
short int          x = 15213;
unsigned short int ux = (unsigned short) x;
short int          y = -15213;
unsigned short int uy = (unsigned short) y;
```

## 결과값

- 비트 표현에는 변화가 없다
- 양수는 변화가 없다 (당연)
  - ◆  $ux = 15213$
- 음수는 양수로 변환된다
  - ◆  $uy = 50323$


# Signed와 Unsigned와의 관계



$$ux = \begin{cases} x & x \geq 0 \\ x + 2^w & x < 0 \end{cases}$$

# Signed와 Unsigned와의 관계

Weight	-15213		50323	
1	1	1	1	1
2	1	2	1	2
4	0	0	0	0
8	0	0	0	0
16	1	16	1	16
32	0	0	0	0
64	0	0	0	0
128	1	128	1	128
256	0	0	0	0
512	0	0	0	0
1024	1	1024	1	1024
2048	0	0	0	0
4096	0	0	0	0
8192	0	0	0	0
16384	1	16384	1	16384
32768	1	-32768	1	32768

  $uy = y + 2 * 32768 = y + 65536$  (16비트의 경우)

# C 언어에서 signed, unsigned 변환

## 상수

- 아무 명시가 없으면 signed integers 임
- U를 숫자 끝에 붙이면 Unsigned  
0U, 4294967259U

## 타입변환 Casting

- 명시적으로 casting을 하는 경우

```
int tx, ty;
unsigned ux, uy;
tx = (int) ux;
uy = (unsigned) ty;
```
- 묵시적 캐스팅 Implicit casting 을 이용할 수도 있다

```
tx = ux; // unsigned를 signed로 변환
uy = ty; // signed를 unsigned로 변환
```

# Casting 충격

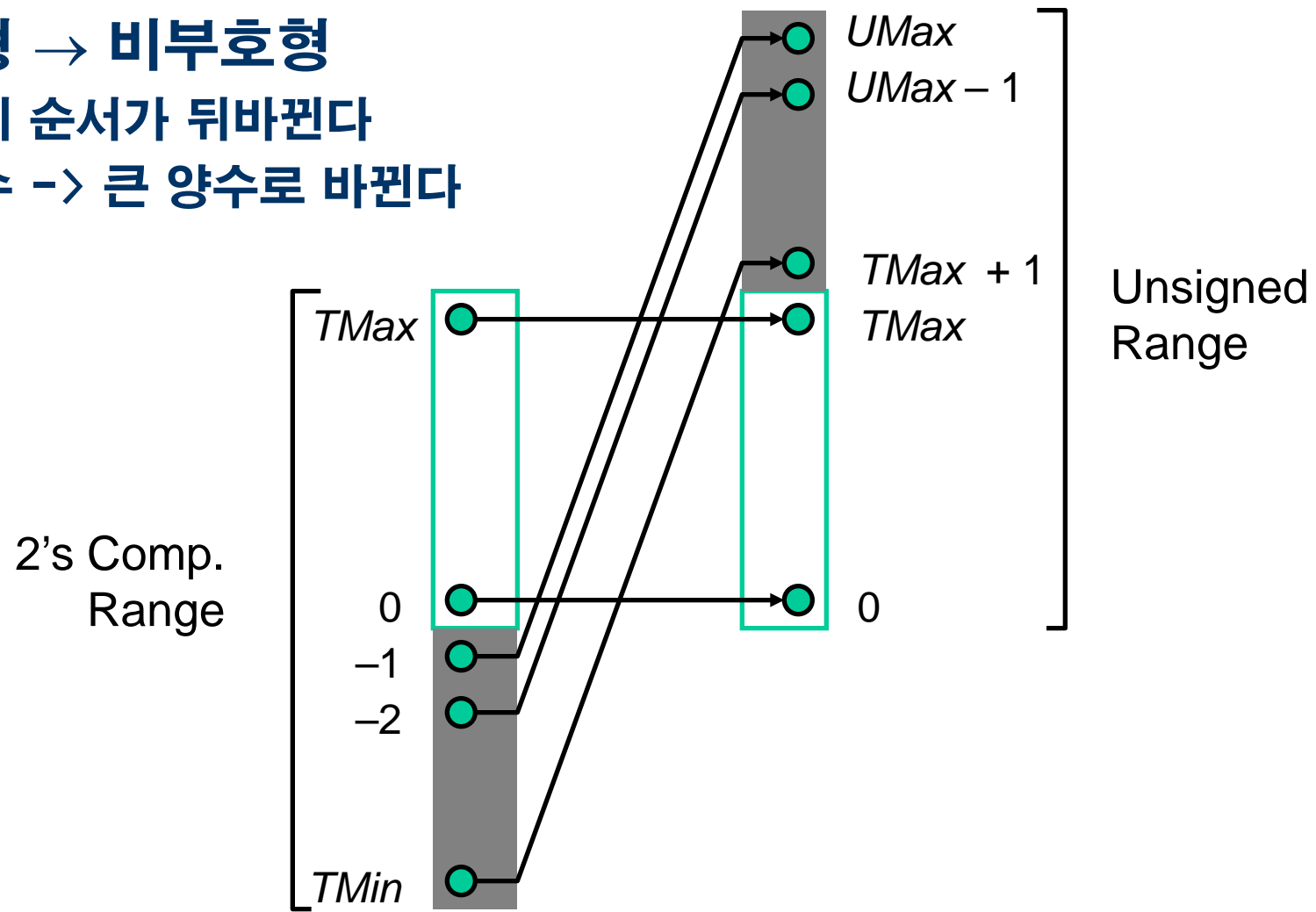
## 수식계산시

- signed와 unsigned 값들이 한 개의 수식 내에 섞여 있는 경우 implicit 하게 unsigned로 바뀌어 진다
- 비교연산자에서도 발생한다 <, >, ==, <=, >=
- Examples for  $W = 32$

Constant <sub>1</sub>	Constant <sub>2</sub>	Relation	Evaluation
0	0U	==	<b>unsigned</b>
-1	0	<	<b>signed</b>
-1	0U	>	<b>unsigned</b>
2147483647	-2147483648	>	<b>signed</b>
2147483647U	-2147483648	<	<b>unsigned</b>
-1	-2	>	<b>signed</b>
(unsigned)-1	-2	>	<b>unsigned</b>
2147483647	2147483648U	<	<b>unsigned</b>
2147483647	(int) 2147483648U	>	<b>signed</b>

# Casting 충격에 대한 설명

- 부호형 → 비부호형
  - 크기 순서가 뒤바뀐다
  - 음수 → 큰 양수로 바뀐다



# 부호 확장 sign extension

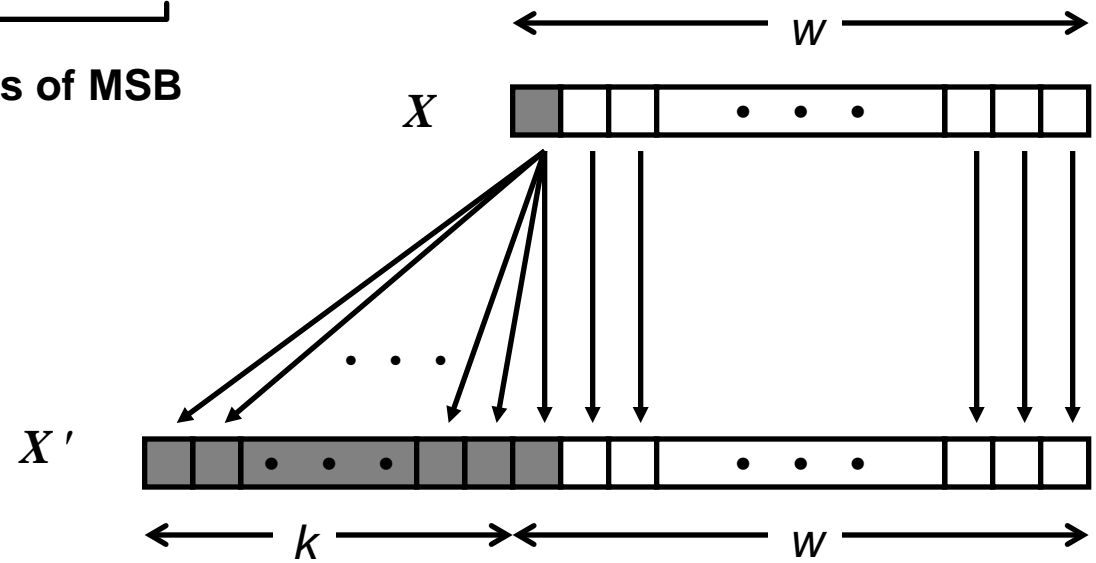
## 목적

- $w$  비트의 부호형 정수  $x$ 가 주어질 때  $x$ 를  $w+k$  비트의 보다 길이가 긴 정수로 변환시킨다

## 규칙

- $x$ 의 부호비트를  $k$  개 복사한다
- $X' = \underbrace{X_{w-1}, \dots, X_{w-1}}_{k \text{ copies of MSB}}, X_{w-1}, X_{w-2}, \dots, X_0$

$k$  copies of MSB





# 부호 확장 예제

```
short int x = 15213;
int      ix = (int) x;
short int y = -15213;
int      iy = (int) y;
```

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101
ix	15213	00 00 3B 6D	00000000 00000000 00111011 01101101
y	-15213	C4 93	11000100 10010011
iy	-15213	FF FF C4 93	11111111 11111111 11000100 10010011

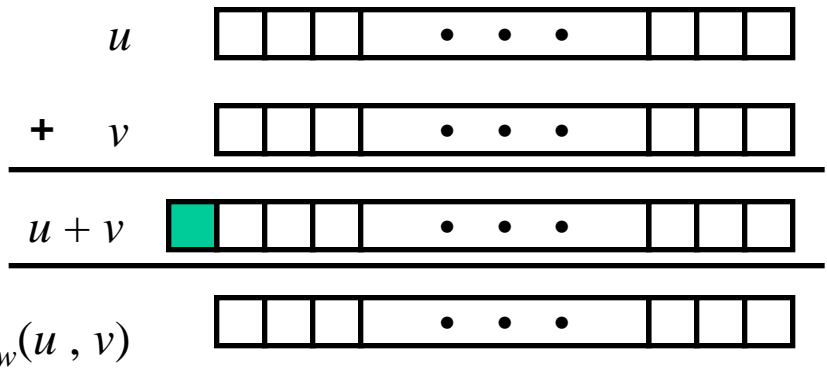
C 언어에서는 부호확장을 자동으로 해준다

# 정수의 연산

## 비 부호형의 덧셈

- 일반적인 덧셈연산과 동일
- Carry 는 무시
- mod 함수로 표시가능
- $s = \text{UAdd}_w(u, v) = (u + v) \bmod 2^w$

Operands:  $w$  bits

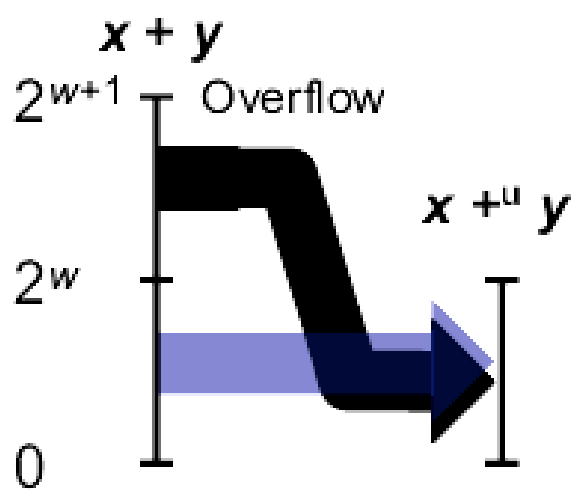


참값:  $w+1$  bits

캐리 버림:  $w$  bits

$\text{UAdd}_w(u, v)$

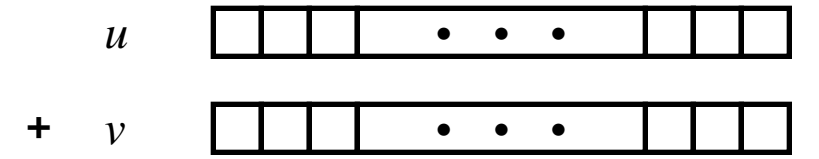
# 비부호형 정수의 덧셈



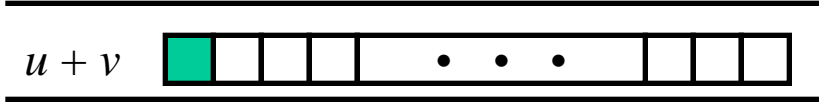
$x+y$ 의 값이  $2^w - 1$ 보다 크면 Overflow가 발생한다

# 부호형(2의 보수)에서의 덧셈

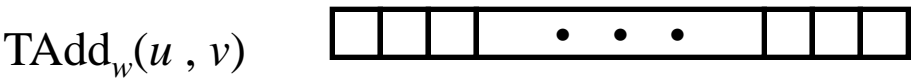
Operands:  $w$  bits



참값:  $w+1$  bits



Discard Carry:  $w$  bits



## 비부호형에서의 덧셈과 동일하게 수행

- C에서 부호형과 비부호형의 덧셈 Signed vs. unsigned  
`int s, t, u, v;`  
`s = (int) ((unsigned) u + (unsigned) v);`  
`t = u + v`
- `s == t` 동일한 결과를 얻는다

# 2의 보수 덧셈에서 Overflow 찾아내기

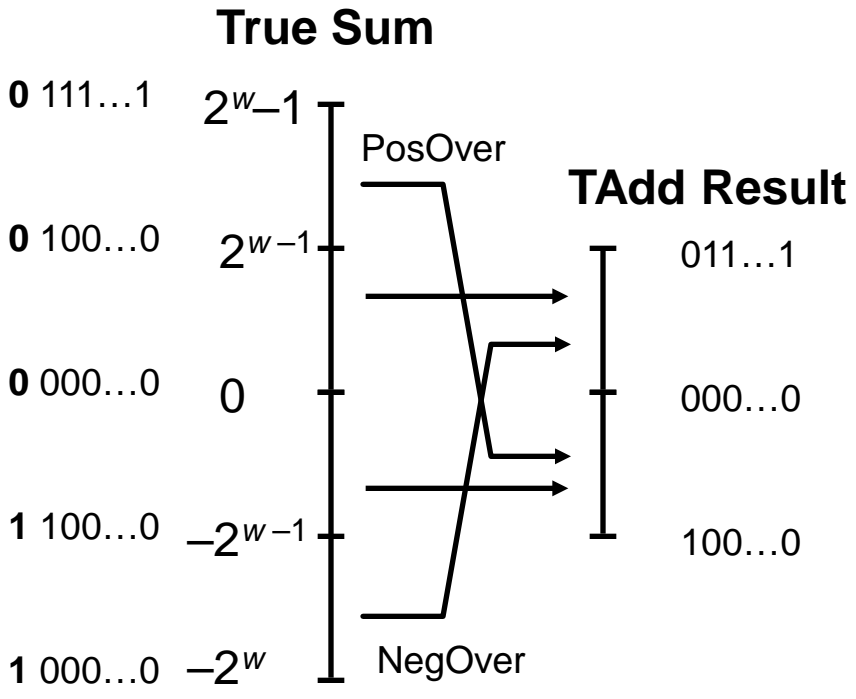
## 목표

- $s = \text{TAdd}_w(u, v)$  일때
- $s = \text{Add}_w(u, v)$  성립여부 체크
- Example

```
int s, u, v;
s = u + v;
```

## 판단방법

- Overflow iff either:
  - $u, v < 0, s \geq 0$  (NegOver)
  - $u, v \geq 0, s < 0$  (PosOver)



$$\text{TAdd}_w(u, v) = \begin{cases} u + v + 2^{w-1} & u + v < \mathcal{M}_w \quad (\text{NegOver}) \\ u + v & \mathcal{M}_w \leq u + v \leq \mathcal{M}_w \\ u + v - 2^{w-1} & \mathcal{M}_w < u + v \quad (\text{PosOver}) \end{cases}$$

# Practice 6 : Overflow

두 정수의 덧셈의 결과 오버플로우가 발생하지 않으면 1을 리턴하는 함수 `tadd_ok(int x, int y)`를 작성하시오.

```
int tadd_ok(int x, int y) {
```

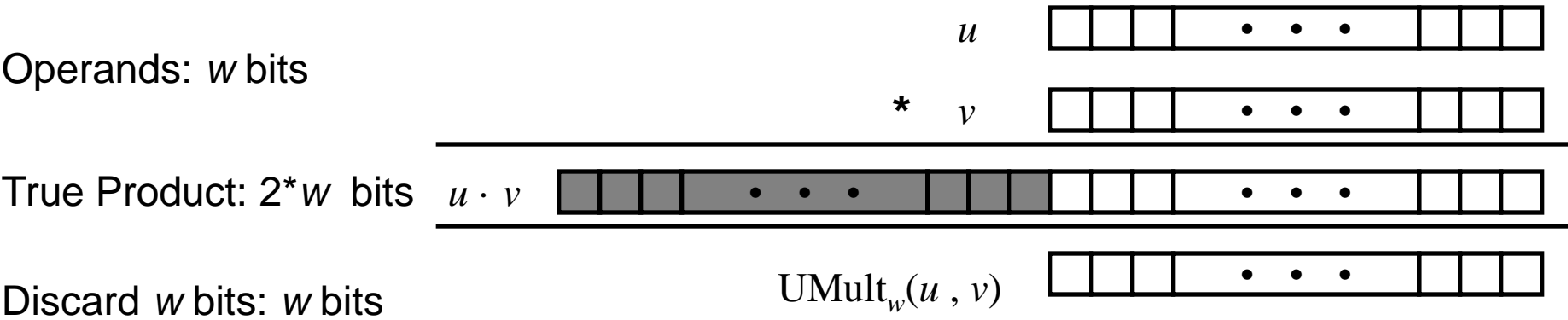
```
}
```

# Practice 7 : Casting

다음 표의 식들이 32비트 머신에서 2의 보수를 사용하는 연산을 수행한다고 할 때, 비교연산의 결과(Y/N)과 사용되는 정수의 타입을 채우시오.

Expression	Type	Evaluation
$-2147483647-1 == 2147483648U$	_____	_____
$-2147483647-1 < 2147483647$	_____	_____
$-2147483647-1U < 2147483647$	_____	_____
$-2147483647-1 < -2147483647$	_____	_____
$-2147483647-1U < -2147483647$	_____	_____

# 비부호형 곱셈



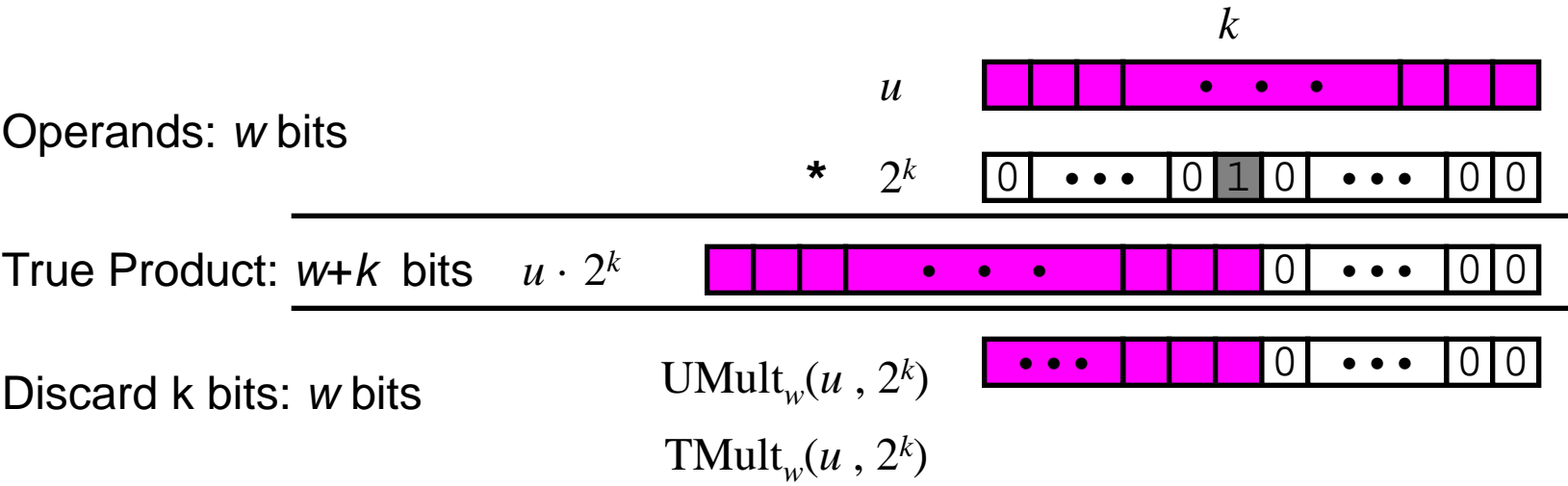
## 표준 곱셈함수와 동일

- 상위  $w$  비트는 무시
- mod 로 표시할 수 있음
- $\text{UMult}_w(u, v) = (u \cdot v) \bmod 2^w$

부호형 곱셈은 비부호형과 동일하게 수행



# Shift 연산을 이용한 곱셈



## 연산

- $u \ll k$  gives  $u * 2^k$
- signed 와 unsigned 모두 적용

# 컴파일러의 곱셈번역

## C Function

```
int mul12(int x)
{
    return x*12;
}
```

### 컴파일한 산술연산

```
leal (%eax,%eax,2), %eax
sall $2, %eax
```

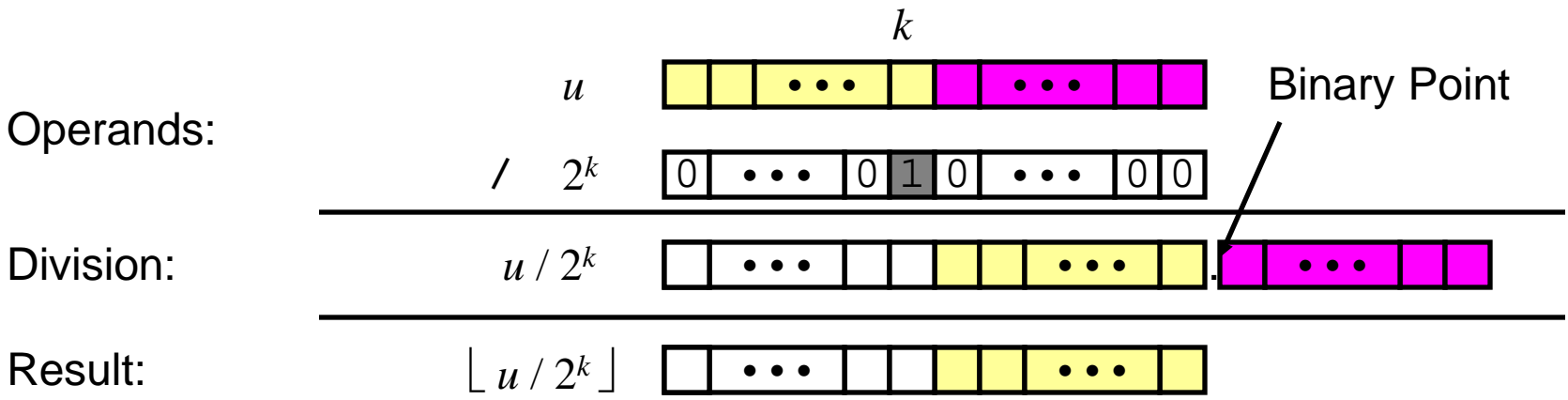
### 해석

```
t <- x+x*2
return t << 2;
```

C 컴파일러는 상수의 곱셈을 자동으로 shift와 덧셈으로 번역한다

# 비부호형 정수의 shift를 이용한 나눗셈

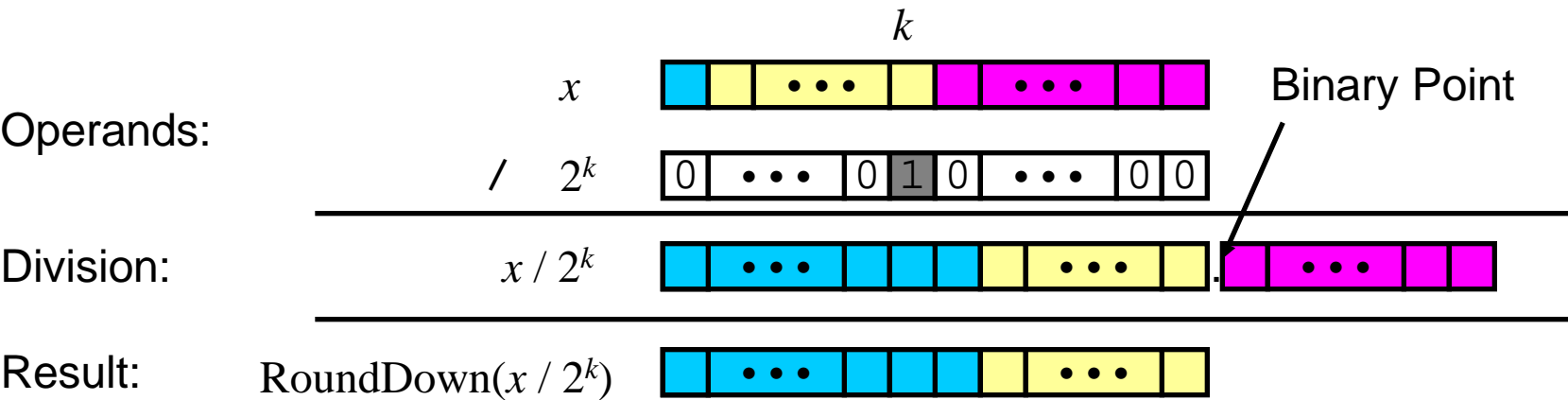
- $u \gg k$  하면  $\lfloor u / 2^k \rfloor$  가 된다
- 논리 쉬프트 연산을 사용 logical shift



	Division	Computed	Hex	Binary
x	15213	15213	3B 6D	00111011 01101101
x >> 1	7606.5	7606	1D B6	00011101 10110110
x >> 4	950.8125	950	03 B6	00000011 10110110
x >> 8	59.4257813	59	00 3B	00000000 00111011

# 부호형 정수의 shift를 이용한 나눗셈

- $x \gg k$  gives  $\lfloor x / 2^k \rfloor$
- 산술 쉬프트 연산을 사용 arithmetic shift
- $x < 0$  이면, 잘못된 방향으로 절삭이 일어난다



	Division	Computed	Hex	Binary
y	-15213	-15213	C4 93	11000100 10010011
y >> 1	-7606.5	-7607	E2 49	11100010 01001001
y >> 4	-950.8125	-951	FC 49	11111100 01001001
y >> 8	-59.4257813	-60	FF C4	11111111 11000100

# 요약

정수에는 unsigned와 signed 가 있다  
signed 정수는 2의 보수로 표현된다  
signed, unsigned 정수들 간에 다양한 덧셈, 곱셈, 나눗셈을  
알아보았다.

다음주는 소수의 표현

\* 예습은 pp.137-144, 2.4.2 와 2.4.3