



CHUNGNAM NATIONAL UNIVERSITY



시스템 프로그래밍

강의 2 : 2장. 정보의 표현 및 처리 I
2.1, 2.2, 2.3 정수의 표현 및 연산
<http://eslab.cnu.ac.kr>

지난 시간

❖ 1장. 컴퓨터 시스템 소개

- ❖ 컴퓨터 시스템은 하드웨어, 시스템 소프트웨어, 응용프로그램으로 구성된다.
- ❖ 프로그램은 번역되어 변화한다.
- ❖ 프로그램은 시스템 프로그램의 도움으로 하드웨어에서 실행된다.

오늘의 주제 : 정수의 표현

컴퓨터에서 데이터의 표현

- 컴퓨터 내부(메모리, 레지스터)에 데이터가 표현되는 방식을 이해하는 것은 매우, 아~주 중요! 그 이유를 알 수 있을까?
- 바이트 저장 순서 : Byte Ordering
- 프로그램 소스코드
- 프로그램 실행코드
 - 문자 : char
 - 정수 : int, short int, long int
 - 실수 : float, double
- 정수의 타입변환 casting
- 정수의 연산

바이트 값의 인코딩

1바이트 Byte = 8 bits

- 이진수 00000000_2 to 11111111_2
- 십진수: 0_{10} to 255_{10}
- 16진수 00_{16} to FF_{16}
 - '0' to '9' and 'A' to 'F' 사용
 - $FA1D37B_{16}$ 는 C에서 다음과 같이 표시
 - $0xFA1D37B$
 - $0xfa1d37b$

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

컴퓨터의 워드길이 Machine Words

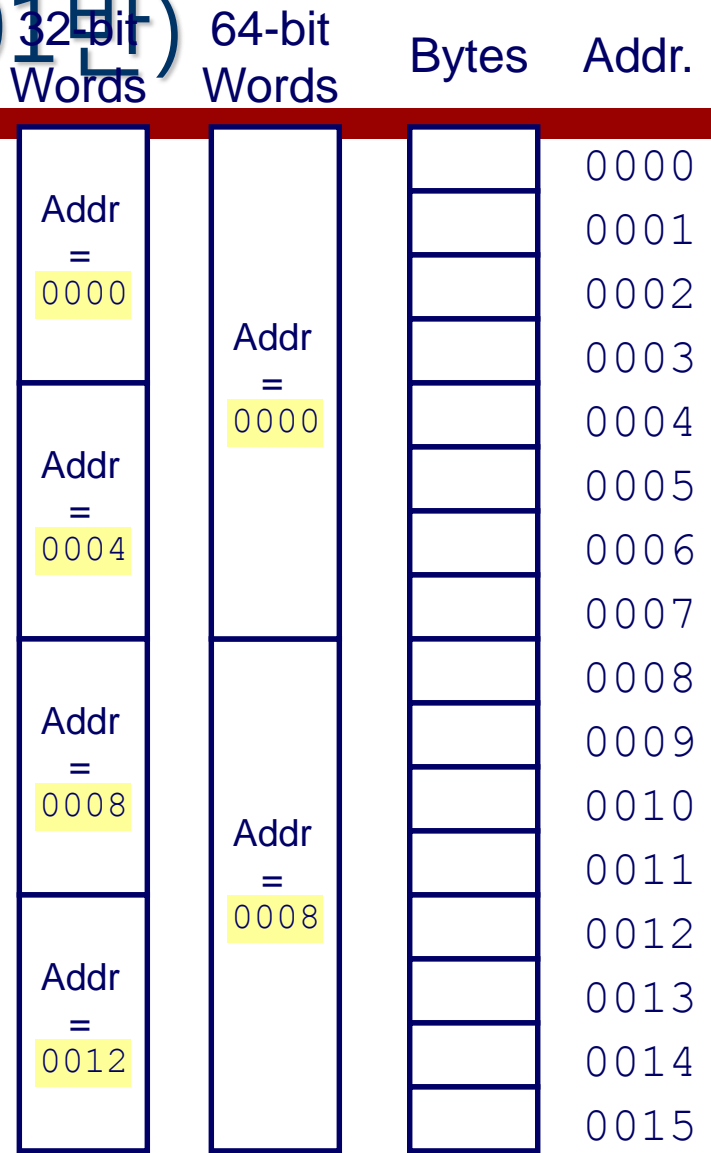
컴퓨터의 워드길이 Word Size

- 정수 값의 크기를 말한다
 - 주소의 길이가 되기도 한다
- 대부분의 요즘 컴퓨터는 32비트(4바이트) 워드이다
 - 이로 인해 주소 범위가 4GB로 제한된다.
 - 메모리가 많이 필요한 프로그램에서는 제약이 될 수 있다
- 최근 데스크 탑 컴퓨터는 64비트 워드를 사용한다
 - 가용 주소 공간 $\approx 1.8 \times 10^{19}$ bytes
 - x86-64 컴퓨터는 48비트 주소를 지원한다: 256 테라바이트
- 컴퓨터는 다양한 데이터 타입을 지원한다
 - 워드의 일부분 또는 여러 워드 길이의 데이터 타입
 - 모든 데이터 타입은 바이트의 배수를 길이로 갖는다.

워드 기반 메모리 구조(01번)

주소는 메모리에서 바이트의 위치를 지정

- 워드의 첫번째 바이트의 위치를 지정
- 연속된 워드의 주소는 4(32비트 머신) 또는 8씩(64비트머신) 증가한다



데이터의 표시

C Data Type	Typical 32-bit	Typical 64-bit	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	8	8
float	4	4	4
double	8	8	8
long double	-	-	10/16
pointer	4	8	8

바이트 저장 순서 Byte Ordering

여러 바이트로 이루어진 데이터는 어떤 순서로 저장되는가의 문제

Sun, Mac, 인터넷 : "Big Endian"

- LSB가 최대 주소의 위치에 기록된다

x86, 안드로이드와 iOS, Windows를 실행하는 ARM프로세서 : "Little Endian"

- LSB가 최소 주소의 위치에 기록된다

Byte Ordering 예제

Byte Ordering
은 언제
문제가 될까?

Big Endian

- Least significant byte 가 최대 주소에 저장됨

Little Endian

- Least significant byte 가 최소 주소에 저장됨

Example

- 변수 `x` 는 다음과 같은 4 바이트의 워드이다 `0x01234567`
- `x`의 주소 `&x` 는 현재 `0x100` 이다

Big Endian

		0x100	0x101	0x102	0x103		
		01	23	45	67		

Little Endian

		0x100	0x101	0x102	0x103		
		67	45	23	01		

리눅스에서 바이트의 출력

데이터를 바이트로 출력해주는 프로그램

- `unsigned char *` 는 바이트 배열을 만든다

```
typedef unsigned char *pointer;

void show_bytes(pointer start, int len)
{
    int i;
    for (i = 0; i < len; i++)
        printf("0x%p\t0x%.2x\n",
               start+i, start[i]);
    printf("\n");
}
```

Printf directives:

%p: Print pointer

%x: Print Hexadecimal

show_bytes 실행결과

```
int a = 15213;  
printf("int a = 15213;\n");  
show_bytes((pointer) &a, sizeof(int));
```

Result (Linux x86-64):

```
int a = 15213;  
0x7ffffb7f71dbc      0x6d  
0x7ffffb7f71dbd      0x3b  
0x7ffffb7f71dbe      0x00  
0x7ffffb7f71dbf      0x00
```

C언어에서 비트수준 연산

비트연산자 $\&$, $|$, \sim , \wedge

- 정수형 연산자에 적용가능
- 인자들을 비트 벡터로 처리
- 인자들은 비트들 끼리 연산

C 언어에서 논리 연산

비트수준 연산자와는 다르다

⇒ && 와 &, ||와 |와의 차이에 유의

&&, ||, !

- 0은 거짓으로 처리
- 0이 아닌 값들은 모두 참으로 처리
- 연산의 결과는 0 또는 1
- 수식의 결과가 첫번째 인자를 계산해서 결정될 수 있으면 두 번째 인자는 계산하지 않는다
 - 예) `a && a/5, p && *p++`

Practice 1: 2진수, 16진수, 10진수

다음의 숫자들을 지시한 대로 변환하십시오.

- A. 0x39A7F8 => 2진수로
- B. 2진수 1100 0011 0101 0001 을 16진수로 => 0x
- C. 0xD5E4C 를 32비트 이진수로
- D. 십진수 255를 이진수로
- E. 16진수 0x010E 를 십진수로

Practice 2 : Boolean bit operation in C

Operate on bit vectors

01101001	01101001	01101001	
& 01010101	01010101	^ 01010101	~ 01010101
<u> </u>	<u> </u>	<u> </u>	<u> </u>
0100	0111		

쉬프트 연산

Left Shift: $x \ll y$

- 비트벡터 x 를 왼쪽으로 y 위치만큼 이동
 - 왼쪽에 있던 비트들은 없어진다
 - ➔ 오른쪽에는 0으로 채운다

Argument x	01100010
$\ll 3$	00010000
Log. $\gg 2$	00011000
Arith. $\gg 2$	00011000

Right Shift: $x \gg y$

- 비트벡터 x 를 오른쪽으로 y 위치 이동
 - ➔ 오른쪽에 있던 비트들이 없어진다
- 논리 쉬프트
 - ➔ 왼쪽에 0으로 채운다
- 산술 쉬프트
 - ➔ 왼쪽의 가장중요한 비트를 복제한다

Argument x	10100010
$\ll 3$	00010000
Log. $\gg 2$	00101000
Arith. $\gg 2$	11101000

Undefined Behavior

- 쉬프트의 크기가 음수이거나 워드길이보다 큰 경우

Practice 3 : Shift operation in C

8비트로 표시한 값 x 에 대해 다음과 같이 left 또는 right shift를 수행한 결과를 빈 칸에 쓰시오.

연산	값	
x	0110 0011	1001 0101
x << 4	0011 0000	
x >> 4(논리)	0000 0110	
x >> 4(산술)		1111 1001

Practice 4 : Byte ordering

다음과 같은 값이 주어졌다. show_bytes 함수를 호출할 때, 실행하는 컴퓨터의 바이트 정렬법에 따라 화면에 출력되는 값을 쓰시오.(주소 제외)

```
int val = 0x87654321;  
byte_pointer valp = (byte_pointer) &val;  
show_bytes(valp, 1); /* A */  
show_bytes(valp, 2); /* B */
```

A. Little endian :

Big endian :

B. Little endian :

Big endian :

2.2 정수의 표현방법

부호가 없는 정수 unsigned int

- 주어진 저장장소에 이진수로 표현

BCD 코드

- 네 비트로 십진수 0~9 를 표시
- 예) $2351_{10} = 0010\ 0011\ 0101\ 0001$ (BCD)

부호-크기 : MSB에 의한 부호표시

- 예) $-22_{10} = 110110_2$

부호를 갖는 수의 표현 signed

- 부호-크기, 1의 보수, 2의 보수 로 표시 가능

정수의 인코딩

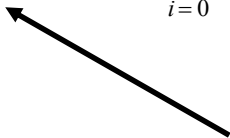
비부호형 정수

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

```
short int x = 15213;
short int y = -15213;
```

부호형 정수 (2의 보수)

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$



MSB는
부호비트

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101
y	-15213	C4 93	11000100 10010011

2의 보수방식에서 MSB는 부호를 나타낸다

- 0 이면 양수
- 1 이면 음수

정수 표시의 예

x =

y =

15213: 00111011 01101101

-15213: 11000100 10010011

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

표현 가능한 정수의 범위

■ 비부호형 Unsigned Values

- **UMin** = 0
000...0
- **UMax** = $2^w - 1$
111...1

■ 2의 보수 Two's Complement Values

- **TMin** = -2^{w-1}
100...0
- **TMax** = $2^{w-1} - 1$
011...1

■ Other Values

- **Minus 1**
111...1

Values for $W = 16$

	Decimal	Hex	Binary
UMax	65535	FF FF	11111111 11111111
TMax	32767		01111111 11111111
TMin	-32768	80 00	
-1	-1	FF FF	11111111 11111111
0	0	00 00	00000000 00000000

여러가지 워드길이의 표현 값

	W			
	8	16	32	64
UMax	255	65,535	4,294,967,295	18,446,744,073,709,551,615
TMax	127	32,767	2,147,483,647	9,223,372,036,854,775,807
TMin	-128	-32,768	-2,147,483,648	-9,223,372,036,854,775,808

- C 프로그램
 - #include <limits.h>
 - 정의되어 있는 값들,
 - ULONG_MAX
 - LONG_MAX
 - LONG_MIN
 - 이 값들은 머신에 따라 달라진다

Signed 와 Unsigned 수의 비교

X	B2U(X)	B2T(X)
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

관찰

- $|TMin| = TMax + 1$
 ▶ 범위가 대칭이 아니다
- $UMax = 2 * TMax + 1$

동일성

- 양수부분에 있어서는 signed와 unsigned의 표현은 동일하다

Practice 5 : 정수의 표현

8비트로 표시된 16진수 x를 다음의 표에 맞게 비부호형 및 부호형 값으로 계산해서 채우시오

x		정수값	
16진수	2진수	비부호형	부호형
0x 03	0000 0011	3	3
0x 51			
0x 8A	1000 1010	$2^7+2^3+2^1 = 128+8+2 = 138$	$-2^7+2^3 +2^1 = -128+8+2 = -118$
0x D9			

데이터 타입변환 casting 하기

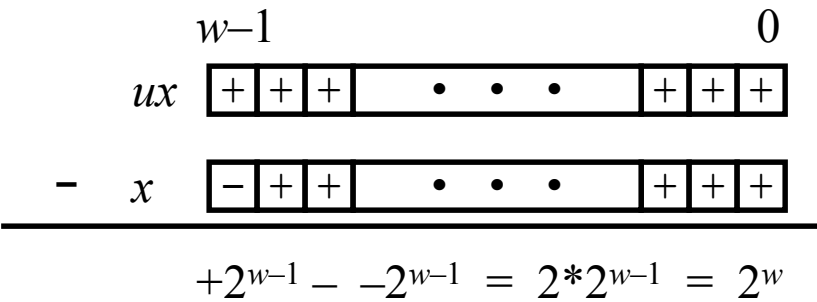
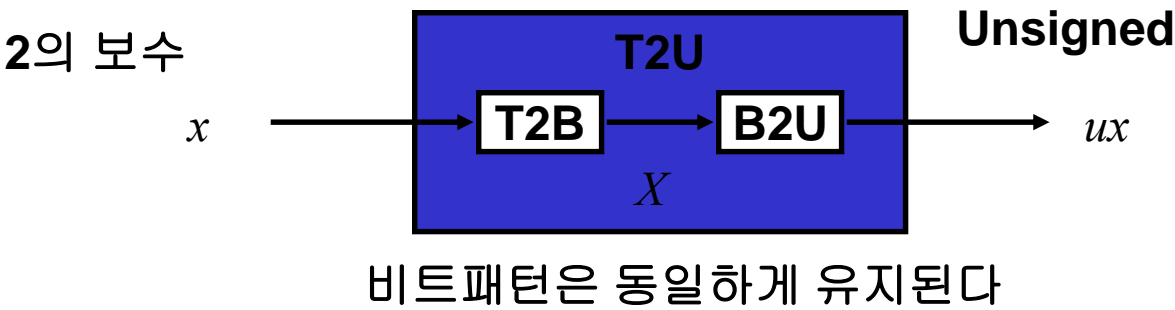
C 언어에서는 signed로부터 unsigned로의 변환을 허용한다

```
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

 $uy = \text{Sum} \quad -15213 \quad 50323$
 $y + 2 * 32768 = y + 65536$ (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

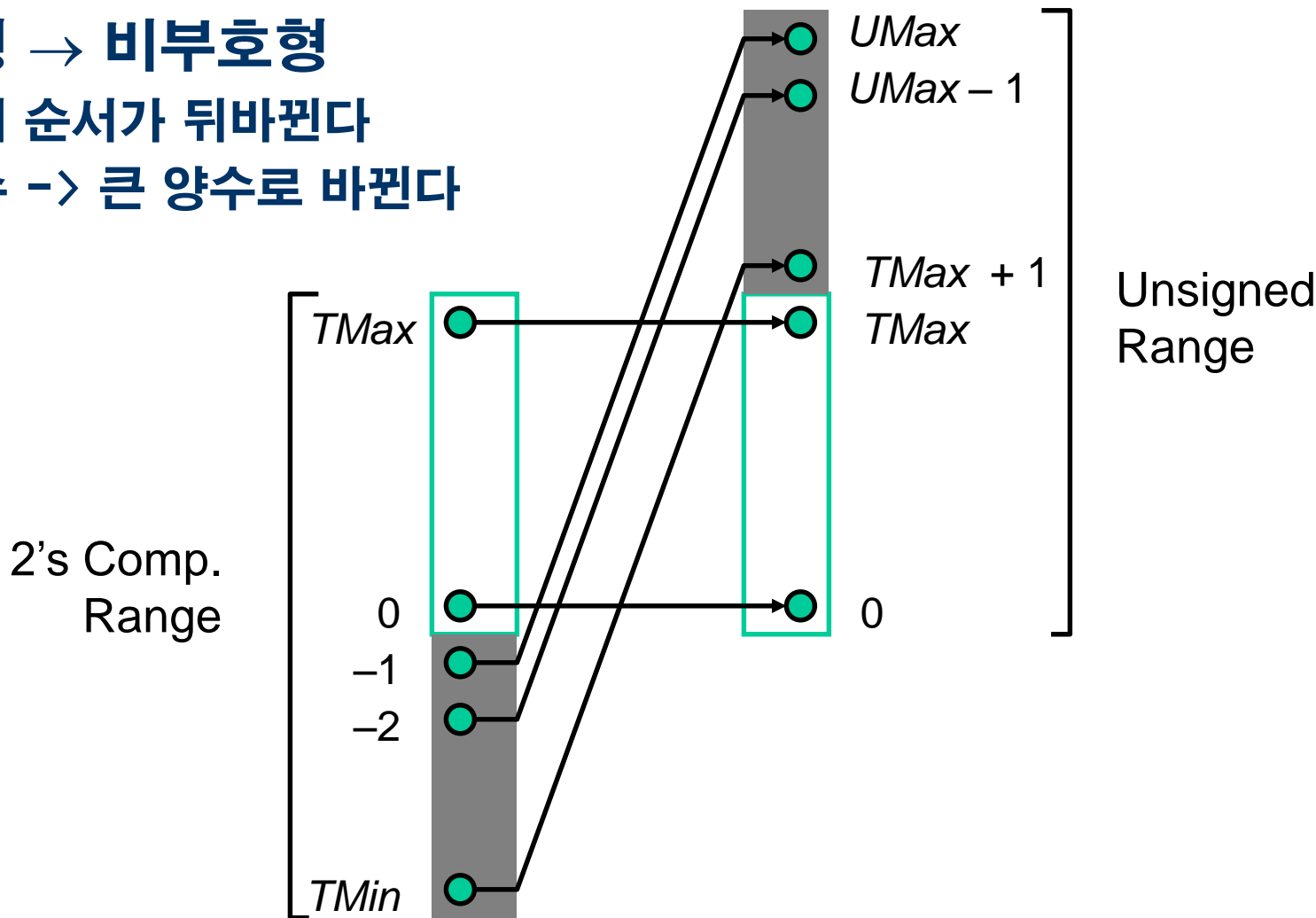
수식계산시

- signed와 unsigned 값들이 한 개의 수식 내에 섞여 있는 경우 implicit 하게 unsigned로 바뀌어 진다
- 비교연산자에서도 발생한다 <, >, ==, <=, >=
- Examples for $W = 32$, **TMIN = -2,147,483,648**, **TMAX = 2,147,483,647**

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

Casting 충격에 대한 설명

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



부호 확장 sign extension

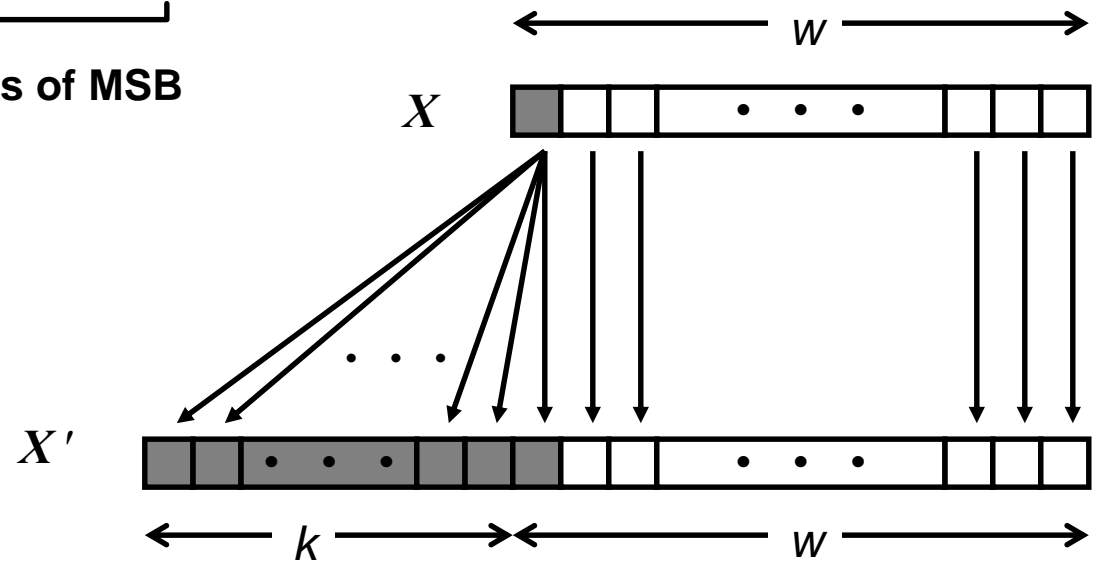
목적

- 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

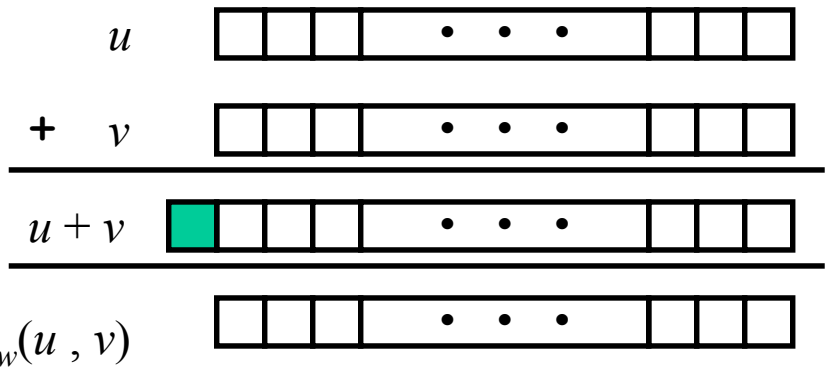
C 언어에서는 부호확장을 자동으로 해준다

정수의 연산

비 부호형의 덧셈

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

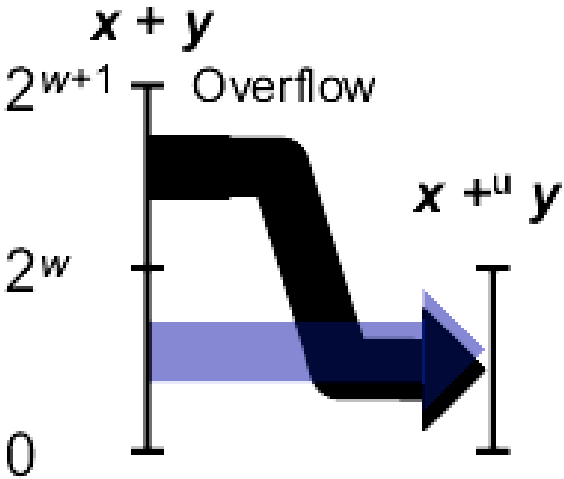
Operands: w bits



참값: $w+1$ bits

캐리 버림: w bits

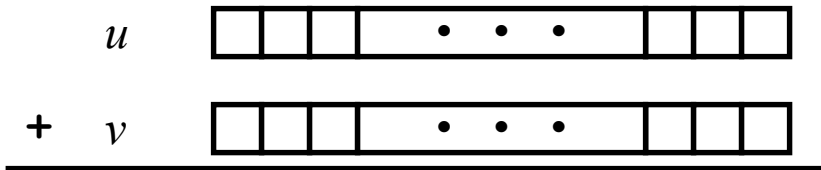
비부호형 정수의 덧셈



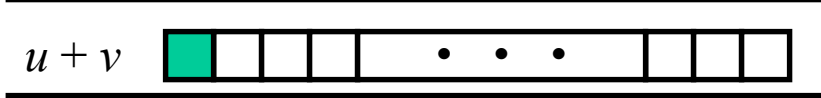
$x+y$ 의 값이 $2^w - 1$ 보다 크면 Overflow가 발생한다

부호형(2의 보수)에서의 덧셈

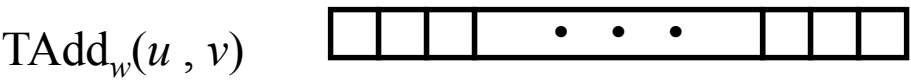
Operands: w bits



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

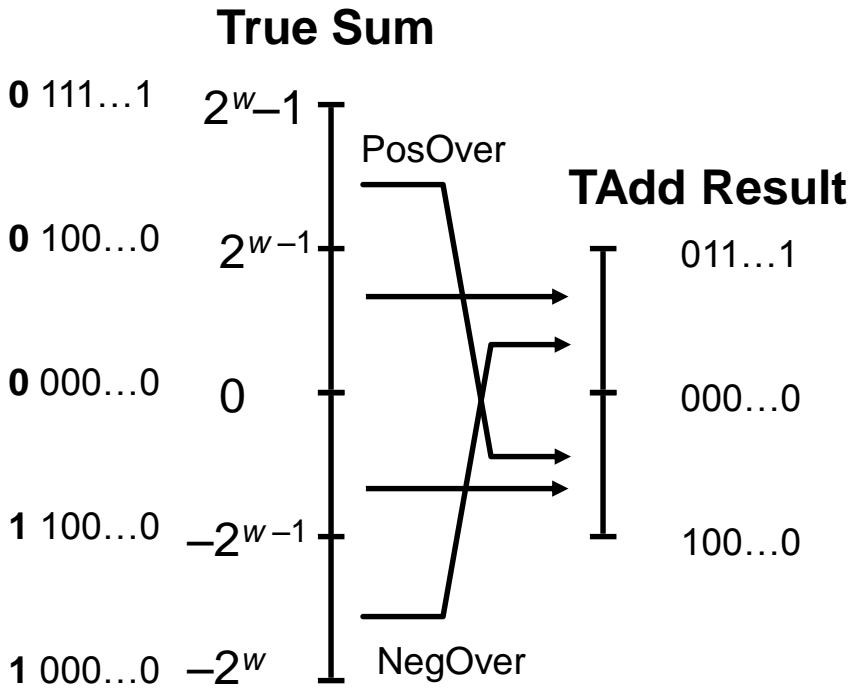
목표

- $s = \text{TAdd}_w(u, v)$ 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)



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) {
```

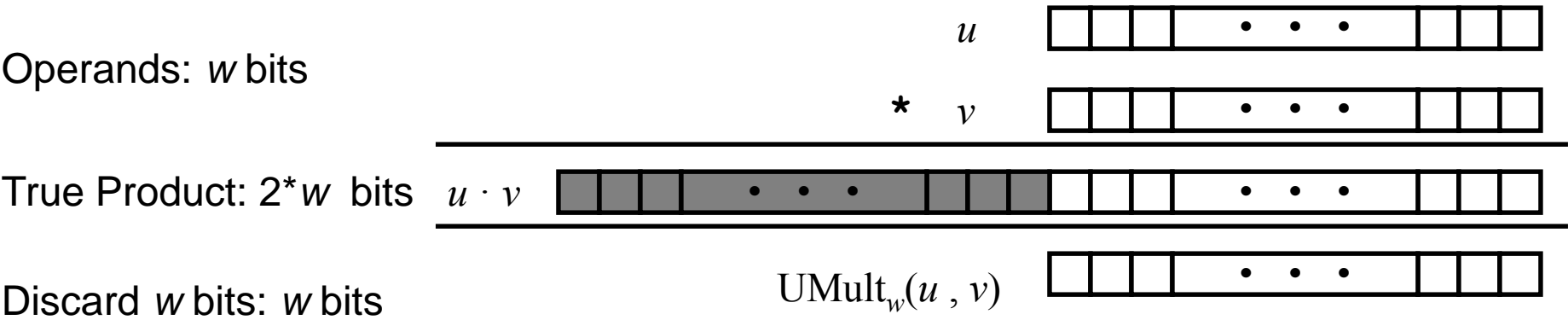
```
}
```

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

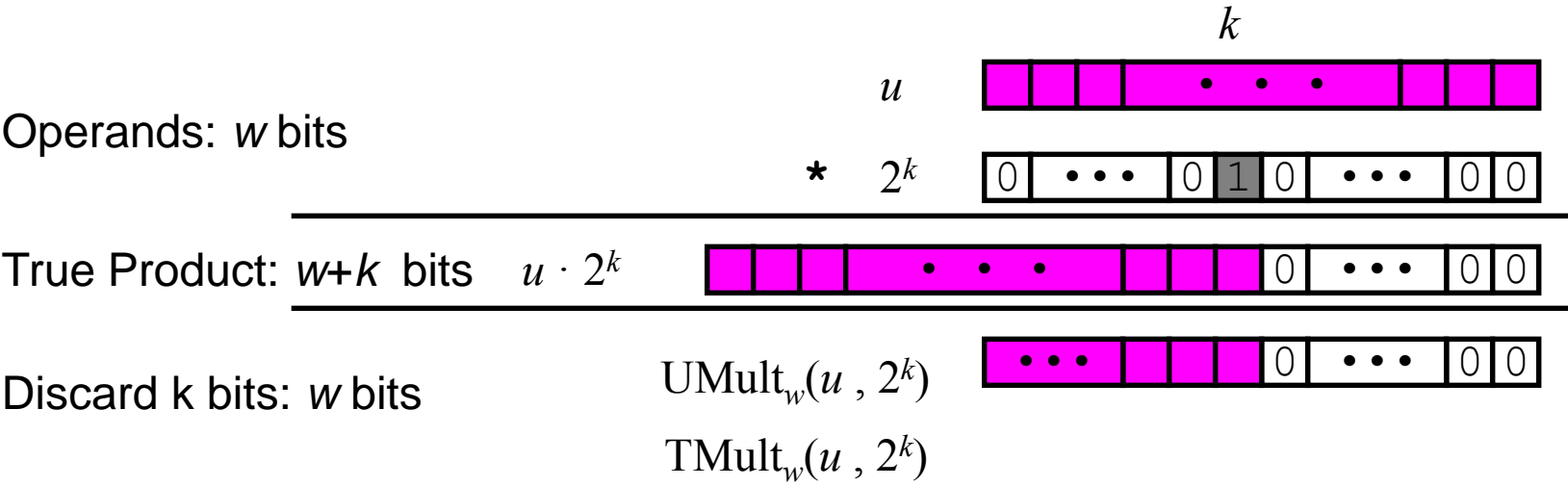


표준 곱셈함수와 동일

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

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
```

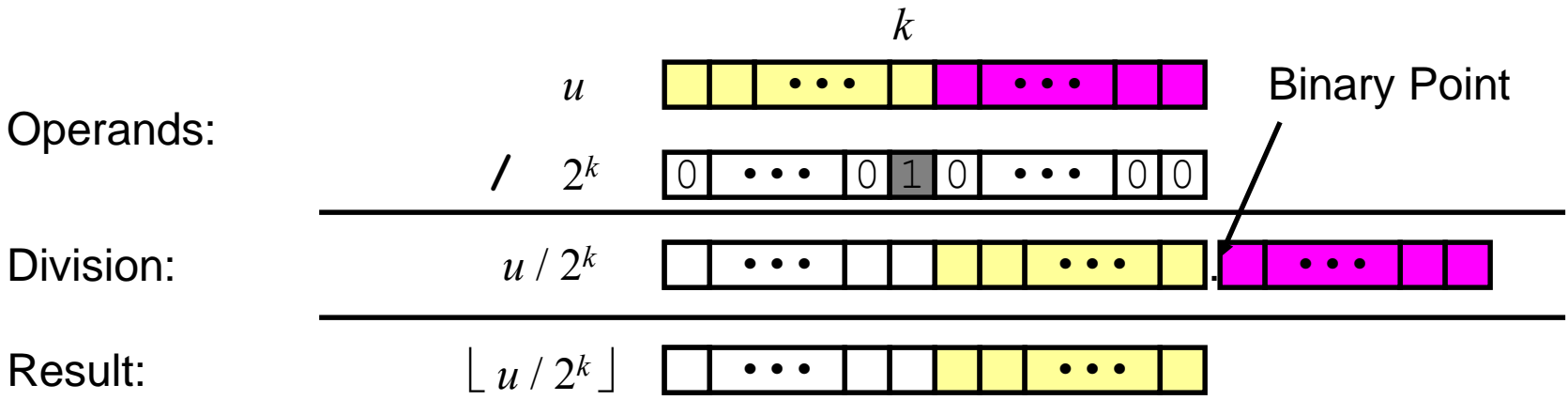
해석

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

C 컴파일러는 상수의 곱셈을 자동으로 shift와 덧셈으로 번역한다

비부호형 정수의 shift를 이용한 나눗셈

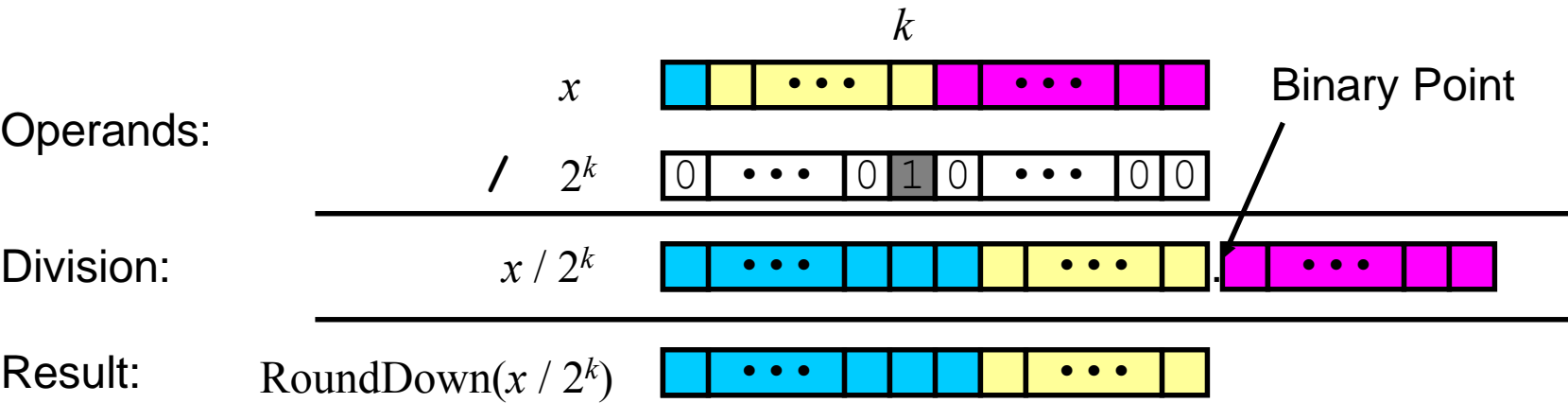
- $u \gg k$ 하면 $\lfloor u / 2^k \rfloor$ 가 된다
- 논리 쉬프트 연산을 사용 logical shift



	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