CHUNGNAM NATIONAL UNIVERSITY

어셈블리어 프로그래밍

강의 2 : 2장. 정보의 표현 및 처리 I
2.1, 2.2, 2.3 정수의 표현 및 연산
2010년 9월 8일
김 형신
http://eslab.cnu.ac.kr

전달사항

데이터의 표현

C 언어에서의 데이터의 크기 (단위 바이트)

C Data Type	Alpha (RIP)	Typical 32-bit	Intel IA32
unsigned	4	4	4
int	4	4	4
long int	8	4	4
char	1	1	1
short	2	2	2
float	4	4	4
double	8	8	8
long double	8/16†	8	10/12
char *	8	4	4

바이트 저장 순서 Byte Ordering

여러바이트로 이루어진 데이터는 어떤 순서로 저장되는가의 문제

Sun, Mac : “Big Endian”

- LSB가 최대 주소의 위치에 기록된다

Alpha, PC : “Little Endian”

- LSB가 최소 주소의 위치에 기록된다

Byte Ordering의 실례

Byte Ordering 은 언제 문제가 될까?

Big Endian

- Least significant byte 가 최대 주소에 저장됨

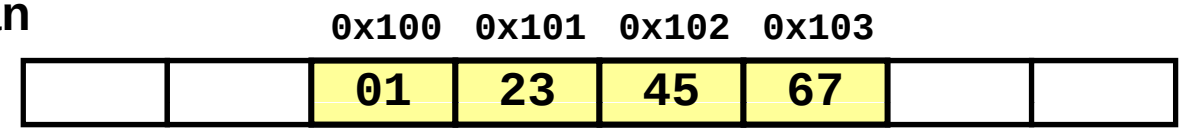
Little Endian

- Least significant byte 가 최소 주소에 저장됨

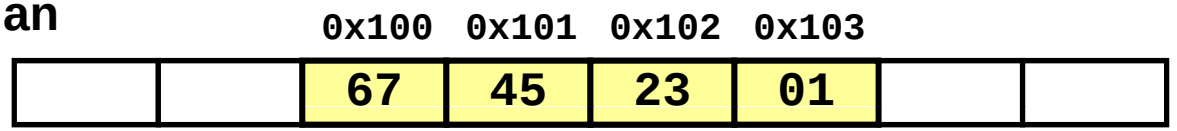
Example

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

Big Endian



Little Endian



기계어 해독하기 Disassembly

Disassembly 예제

Address	Instruction Code	Assembly Rendition
8048365:	5b	pop %ebx
8048366:	81 c3 ab 12 00 00	add $0x12ab,%ebx
804836c:	83 bb 28 00 00 00 00	cmpl $0x0,0x28(%ebx)

Little Endian의 해독과정

- 값: 0x12ab
- 4 바이트로 패딩 padding 하기 : 0x000012ab
- 바이트로 나누기 : 00 00 12 ab
- 뒤집기 (왜?): ab 12 00 00

리눅스 C 컴파일러에서의 구현

데이터의 바이트 표시를 위한 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):

```
int a = 15213;
0x11ffffcb8   0x6d
0x11ffffcb9   0x3b
0x11ffffcba   0x00
0x11ffffcbb   0x00
```

정수의 표현

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)

부호를 갖는 수의 표현 signed

- 부호-크기, 1의 보수, 2의 보수 로 표시 가능

부호-크기 : MSB에 의한 부호표시

- 예) $-22_{10} = 110110_2$

보수에 의한 표현

1의 보수 1's compliment

- n 비트 이진수 x 에 대하여 1의 보수는 $-x = (2^n-1) - x$
- 예) $22_{10} = 00010110_2$ 에 대하여 $-22_{10}=FF_{16}-x = 11101001_2$
- 덧셈시에 end-arround-carry가 발생하면 결과에 1을 더함

2의 보수 2's compliment

- n 비트 이진수 x 에 대하여 2의 보수는 $-x = 2^n - x$
- 예) $22_{10} = 00010110_2$ 에 대하여 $-22_{10}=FF_{16}-x + 1= 11101010_2$
- 덧셈시에 end-arround-carry가 발생하면 무시함

	장점	단점
2의 보수	덧셈이 단순함	부호변경시 +1 필요
1의 보수	부호변경 용이	덧셈시 end-around carry 처리 -0인지 검사: -0 -> 0 ('0'이 두개)
부호-크기	이해하기 쉬움	덧셈, 뺄셈이 복잡함

w 비트 정수 x의 표시

Unsigned

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

Two's Complement (2의 보수)

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

Sign Bit

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

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

2의 보수방식에서 MSB는 부호-크기 방식처럼 부호를 나타낸다

- **0 이면 양수**
- **1 이면 음수**

정수 표시의 예

```
x =      15213: 00111011 01101101
y =     -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

- Unsigned Values
  - *UMin* = 0
    000…0
  - *UMax* = $2^w - 1$
    111…1
- 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	7F FF	01111111 11111111
TMin	-32768	80 00	10000000 00000000
-1	-1	FF FF	11111111 11111111
0	0	00 00	00000000 00000000

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

풀어보기

1. 프로세서의 명령주기 Instruction cycle을 쓰시오.
2. hello.c의 번역단계를 설명하시오.
3. 0x100 번지 부터 4바이트 정수 0x1234 를 저장하려고 할 때, Little Endian의 경우 0x100 번지에 저장되는 한 바이트를 쓰시오.
4. 이진수 10001001 을 Arith >> 2 한 결과를 쓰시오.
5. 십진수 -7 을 8비트 2의 보수를 이용하여 나타내시오.

casting 하기

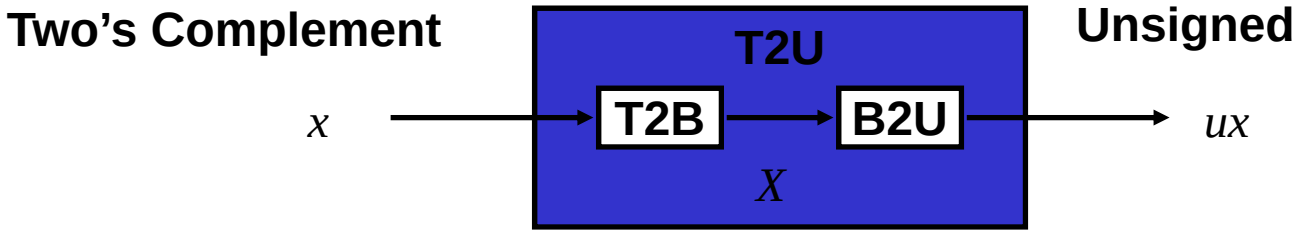
C 언어에서는 signed로부터 unsigned로의 변환을 허용한다

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

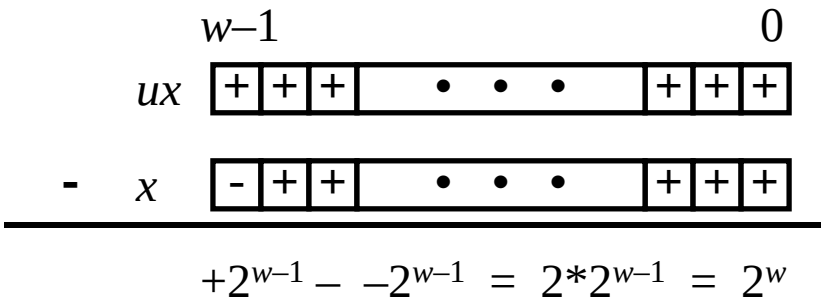
결과값

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

Signed와 Unsigned와의 관계



비트패턴은 동일하게 유지된다



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

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*
Sum		**-15213**		**50323**

- ***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
```

Casting

- 명시적으로 casting을 하는 경우

```
int tx, ty;
unsigned ux, uy;
tx = (int) ux;
uy = (unsigned) ty;
```

- Implicit casting 을 이용할 수도 있다

```
tx = ux;
uy = ty;
```

Casting 충격

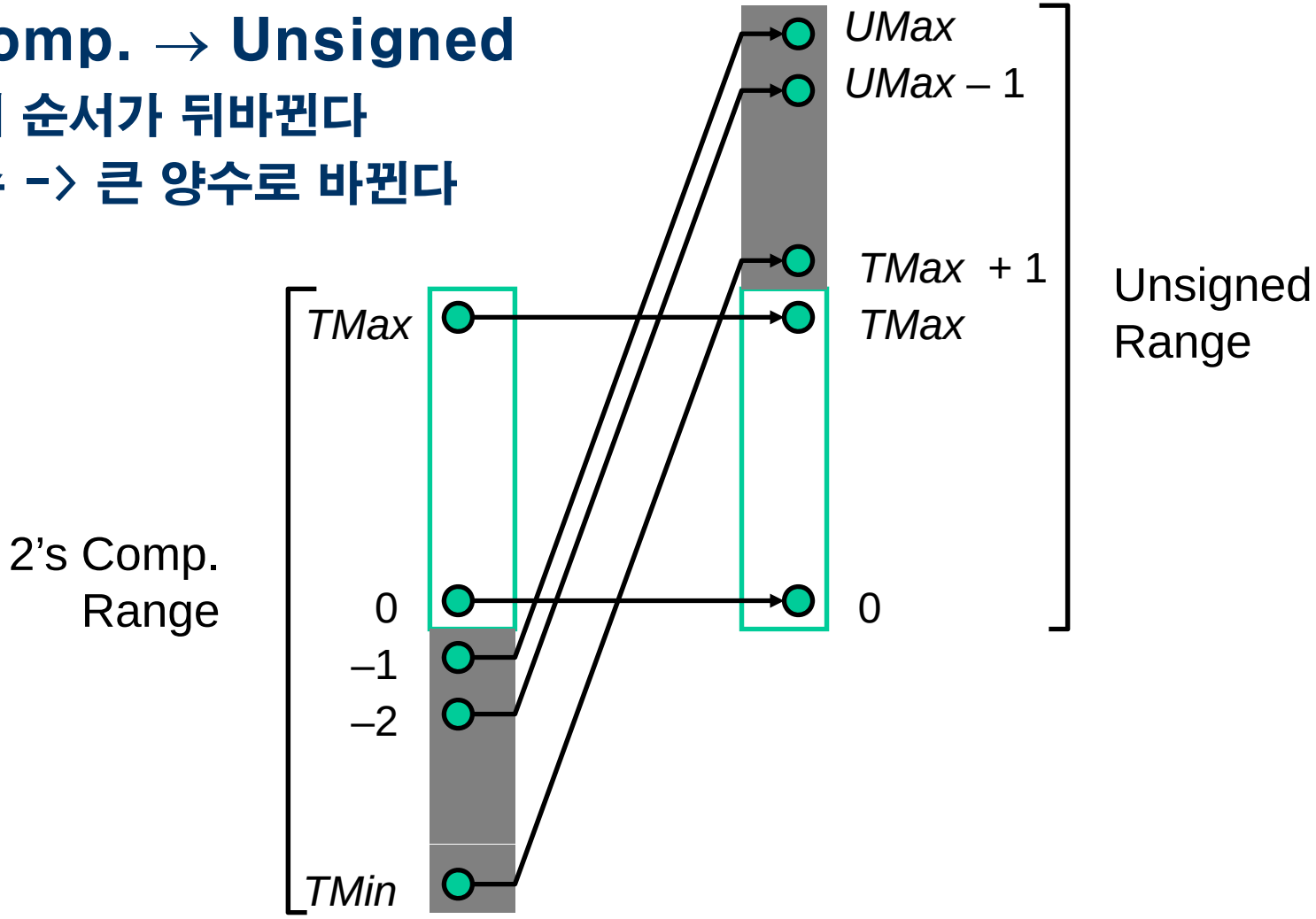
수식계산시

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

Constant$_1$	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

- 2's Comp. → Unsigned
  - 크기 순서가 뒤바뀐다
  - 음수 -> 큰 양수로 바뀐다



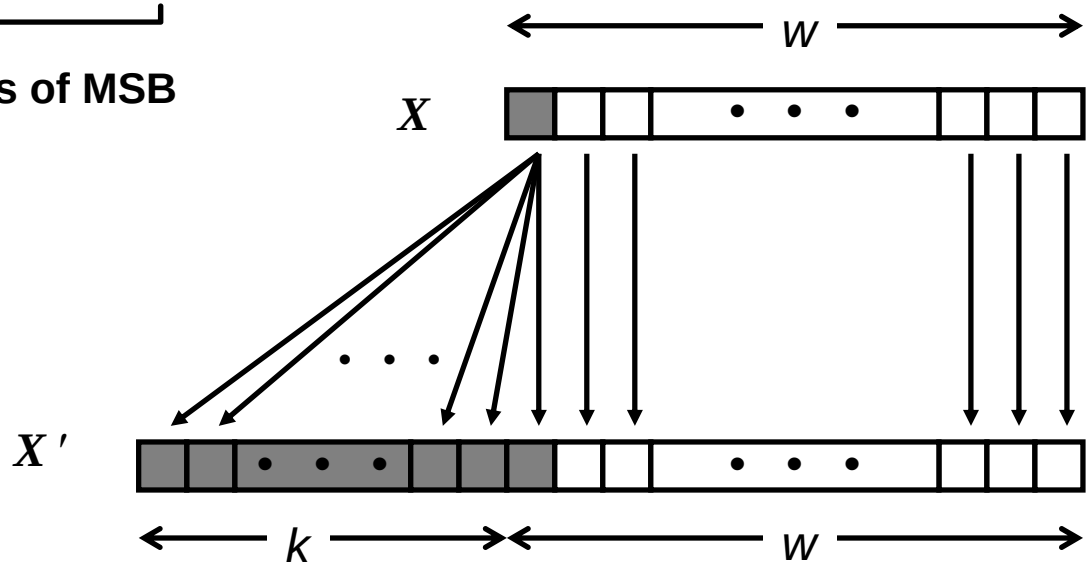
부호 확장 sign extension

목적

- w 비트의 부호형 정수 x가 주어질 때 x를 w+k 비트의 보다 길이가 긴 정수로 변환시킨다

규칙

- x의 부호비트를 k 개 복사한다
- $X' = \underbrace{x_{w-1}, \cdots, x_{w-1}}_{k \text{ copies of MSB}}, x_{w-1}, x_{w-2}, \cdots, x_0$



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

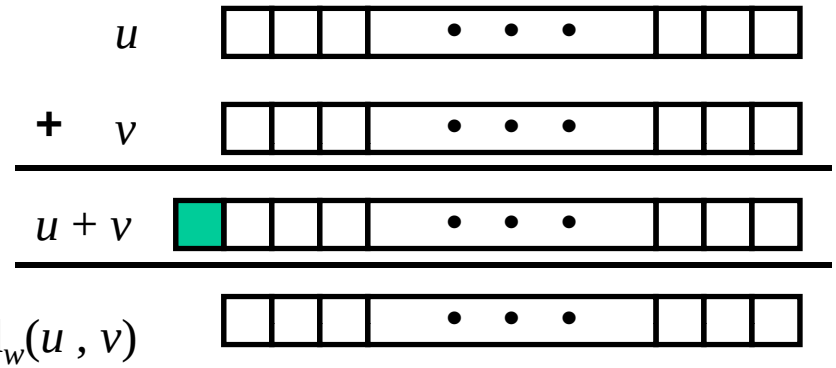
비 부호형의 덧셈

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

Operands: *w* bits

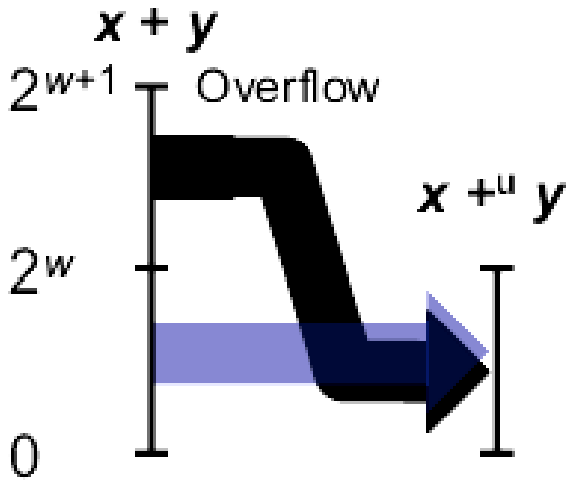
참값: *w*+1 bits

Discard Carry: *w* bits



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

비부호형 정수의 덧셈



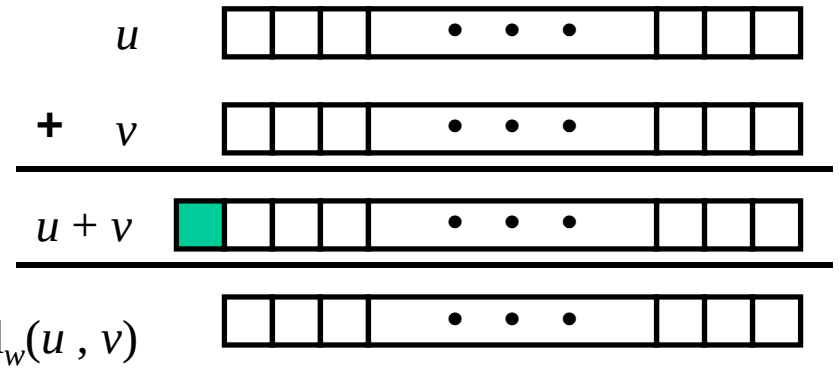
x+y의 값이 $2^w - 1$보다 크면 Overflow가 발생한다

2의 보수에서의 덧셈

Operands: *w* bits

참값: *w*+1 bits

Discard Carry: *w* bits



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

비부호형에서의 덧셈과 동일하게 수행

- **Signed vs. unsigned addition in C:**

```
int s, t, u, v;
s = (int) ((unsigned) u + (unsigned) v);
t = u + v
```

- **Will give** `s == t`

2의 보수 덧셈에서 Overflow 찾아내기

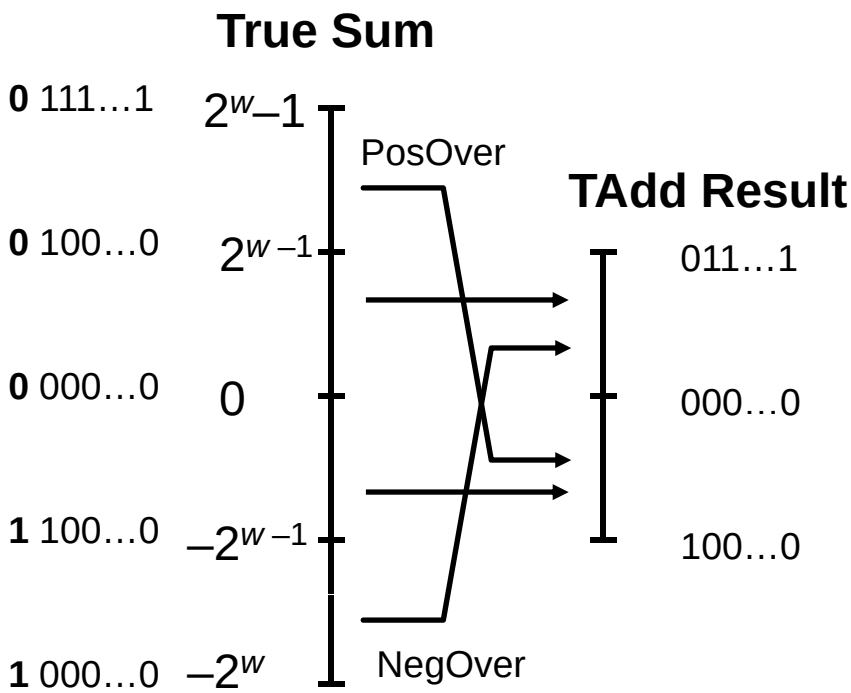
목표

- Given $s = TAdd_w(u, v)$
- Determine if $s = Add_w(u, v)$
- Example

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

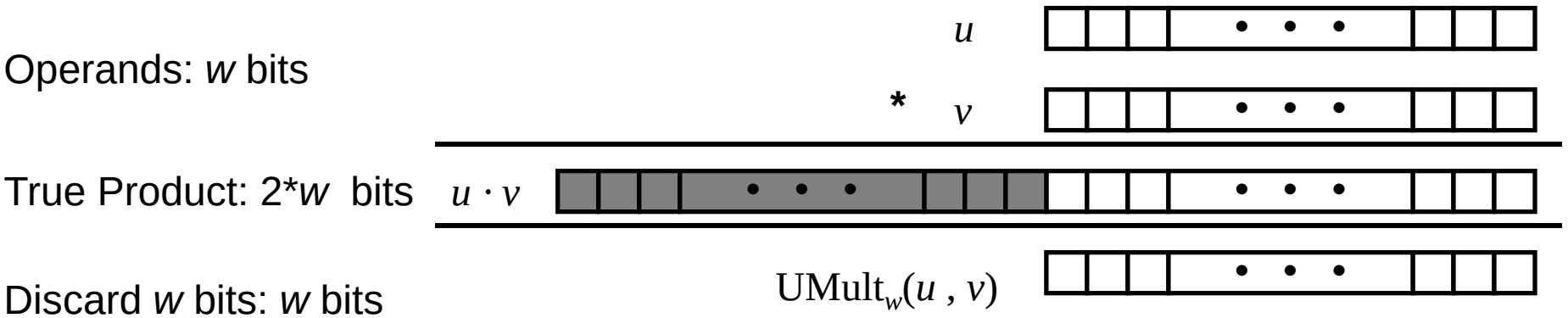
판단방법

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



$$TAdd_w(u,v) = \begin{cases} u+v+2^{w-1} & u+v < TMin_w \quad \text{(NegOver)} \\ u+v & TMin_w \leq u+v \leq TMax_w \\ u+v-2^{w-1} & TMax_w < u+v \quad \text{(PosOver)} \end{cases}$$

비부호형 곱셈

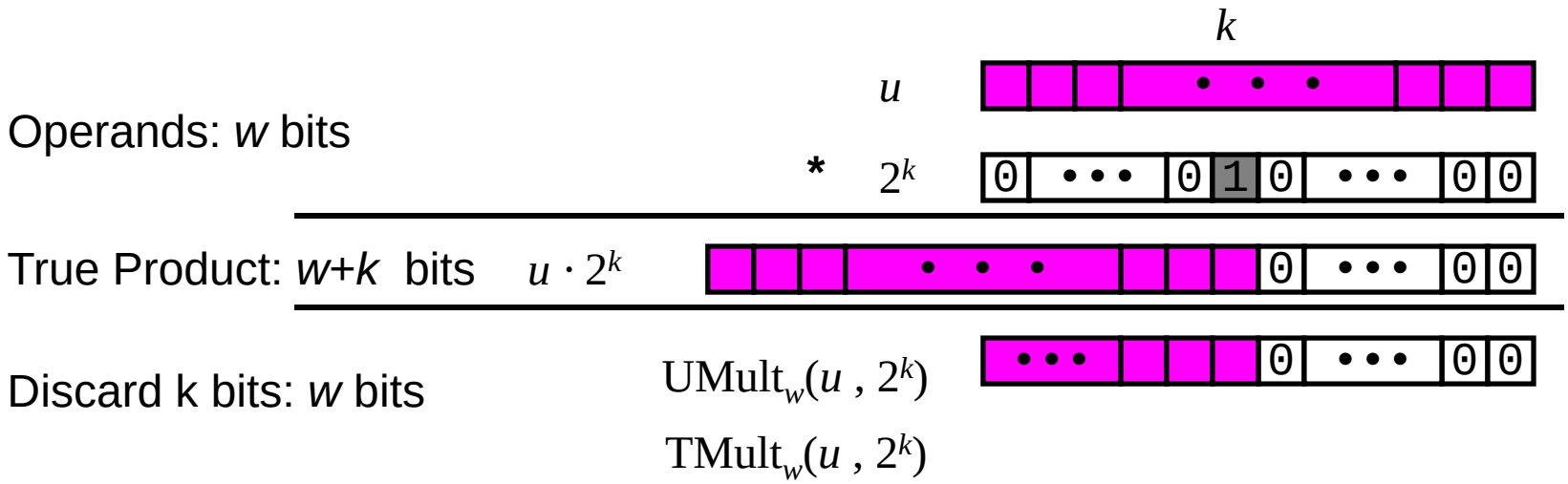


표준 곱셈함수와 동일

- 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 << k **gives** u * ***2^k***
- **signed 와 unsigned 모두 적용**

컴파일러의 곱셈번역

C Function

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

Compiled Arithmetic Operations

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

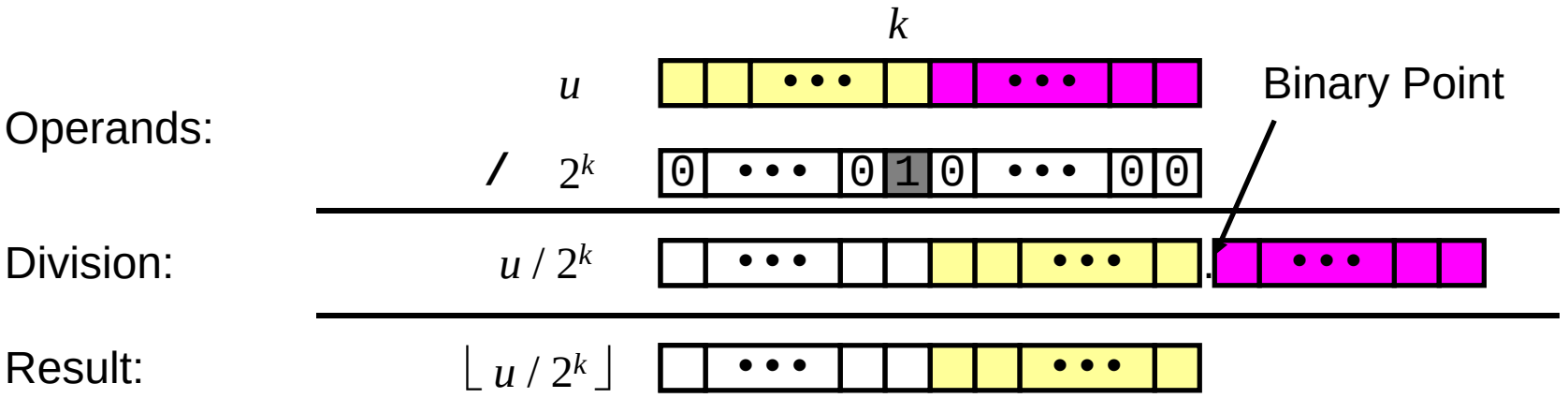
Explanation

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

C 컴파일러는 상수의 곱셈을 자동으로 shift와 덧셈으로 번역한다

비부호형 정수의 shift를 이용한 나눗셈

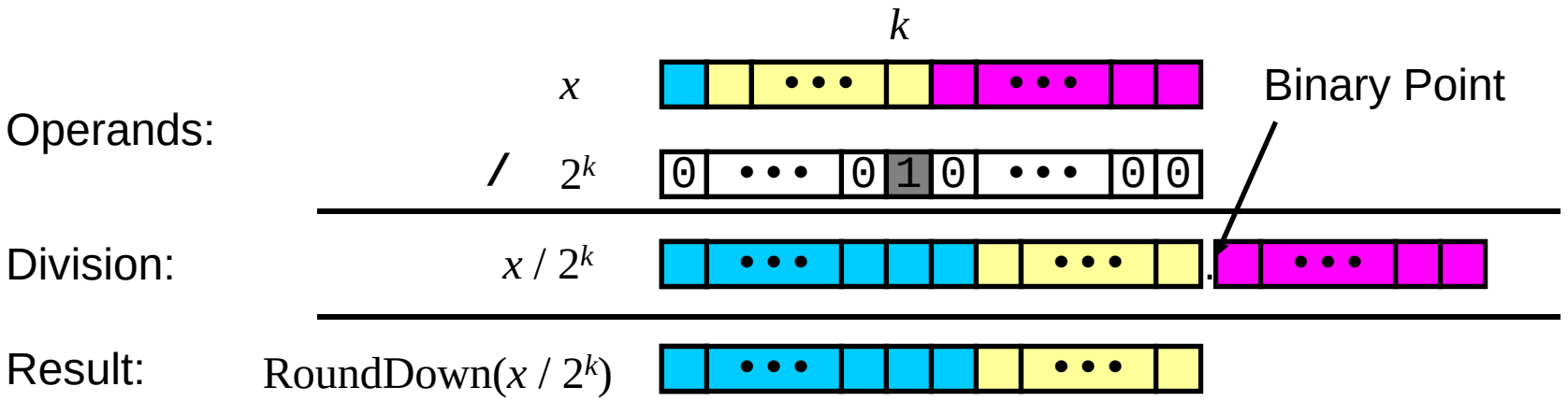
- u >> k **gives** ⌊ u / 2^k ⌋
- **Uses logical shift**



	Division	Computed	Hex	Binary
x	15213	15213	3B 6D	00111011 01101101
x >> 1	7606.5	7606	1D B6	**0**0011101 10110110
x >> 4	950.8125	950	03 B6	**0000**0011 10110110
x >> 8	59.4257813	59	00 3B	**00000000** 00111011

부호형 정수의 shift를 이용한 나눗셈

- x >> k gives ⌊ x / 2^k ⌋
- **Uses arithmetic shift**
- **Rounds wrong direction when** u < 0



	Division	Computed	Hex	Binary
y	-15213	-15213	C4 93	11000100 10010011
y >> 1	-7606.5	-7607	E2 49	**1**1100010 01001001
y >> 4	-950.8125	-951	FC 49	**1111**1100 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.