[Chap.2-1] Representing and Manipulating Information

Young Ik Eom (yieom@skku.edu, 031-290-7120)
Distributing Computing Laboratory
Sungkyunkwan University
http://dclab.skku.ac.kr

Contents

- Introduction
- Information storage
- Integer representations
- Integer arithmetic
- Floating point
- Summary

Introduction

- **Bit**
  - Binary digit
  - 0/1
- **Grouping bits together**
  - Can represent the elements of any finite set
  - n bits → 2^n representations
    (different meanings for each data type)
    - Integer
      - ✓ Unsigned, 2's-complement, …
    - Floating-point number
    - Character
    - Etc

- By studying the actual number representations, we can understand the ranges of values that can be represented and the properties of the different arithmetic operations

Information Storage

- **Byte** → char
  - Binary Term
  - 8-bits
  - Smallest addressable unit of memory ☆

Information Storage

Hexadecimal notation

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

- Single byte: 00_{16} ~ FF_{16}
- Converting between binary and hexadecimal
  - Straightforward
- Converting between decimal and hexadecimal
  - Requires some computation [Refer to text, Section 2.1.1]

Information Storage

Word

- Nominal size of integer and pointer data
  - Virtual address is encoded by a word
    - ✓ Maximum size of the virtual address space is determined by the word size
- Typical word size today
  - 32 / 64 -bits
    - ✓ 4GB virtual address space

Information Storage

Multiple data formats and data sizes

- Multiple data formats
  - Different ways to encode data, different lengths
  - Sizes of C numeric data types

C declarations		Bytes	
Signed	Unsigned	32-bit M	64-bit M
[signed] char	unsigned char	1	1
short	unsigned short	2	2
int	unsigned	4	4
long	unsigned long	4	8
int32_t	uint32_t	4	4
int64_t	uint64_t	8	8
char *		4	8
float		4	4
double		8	8

Information Storage

Byte ordering

- Whether to choose to store the object in memory ordered <u>from least significant byte to most</u> or <u>from most to least</u>

- Little endian
  - The least significant byte (LSB) comes first
  - Used in most Intel-compatible machines
- Big endian
  - The most significant byte (MSB) comes first
  - Used in most machines from IBM and SUN Microsystems
- Bi-endian
  - Can be configured to operate
    as either little- or big-endian machines

Information Storage

Byte ordering

- Example) Data 0x01234567 at address 0x100

Big endian

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

Little endian

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

Information Storage

Strings

- Array of characters terminated by a NULL character
- ASCII character code
  - Decimal digits
    - ✓ '0' ~ '9' → 0x30 ~ 0x39
  - Alphabets (lower case)
    - ✓ 'a' ~ 'z' → 0x61 ~ 0x7A
  - NULL character
    - ✓ NULL → 0x00

ASCII Code Table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source : www.LookupTables.com

** EBCDIC 코드 참고

ASCII Code Table

Friendly	Numeric		Description
‘		‘	left single quote
’		’	right single quote
‚		‚	single low-9 quote
“		“	left double quote
”		”	right double quote
„		„	double low-9 quote
†		†	dagger
‡		‡	double dagger
‰		‰	per mill sign
‹		‹	left angle quote
›		›	right angle quote
♠		♠	black spade suit
♣		♣	black club suit
♥		♥	black heart suit
♦		♦	black diamond suit
‾		‾	overline
←		←	leftward arrow
↑		↑	upward arrow
→		→	rightward arrow
↓		↓	downward arrow
™		™	trademark sign
			horizontal tab
	line feed
			space
	!	!	exclamation mark
"	"	"	double quotation
	#	#	number sign
	$	$	dollar sign
	%	%	percent sign
&	&	&	ampersand
	'	'	apostrophe
	(	(	left parenthesis
	)	)	right parenthesis
	*	*	asterisk
	+	+	plus sign
	,	,	comma
	-	-	hyphen
	.	.	period
⁄	/	/	slash
	0		0
	1		1
	2		2
	3		3
	4		4
	5		5
	6		6
	7		7
	8		8
	9		9
	:	:	colon
	;	;	semicolon
<	<	<	less-than sign
	=	=	equals sign
>	>	>	greater-than sign
	?	?	question mark
	@	@	at sign
	A	A	uppercase letter
	B	B	uppercase letter
	C	C	uppercase letter
	D	D	uppercase letter
	E	E	uppercase letter
	F	F	uppercase letter
	G	G	uppercase letter
	H	H	uppercase letter
	I	I	uppercase letter
	J	J	uppercase letter
	K	K	uppercase letter
	L	L	uppercase letter
	M	M	uppercase letter
	N	N	uppercase letter
	O	O	uppercase letter
	P	P	uppercase letter
	Q	Q	uppercase letter

Friendly	Numeric		Description
	R	R	uppercase letter
	S	S	uppercase letter
	T	T	uppercase letter
	U	U	uppercase letter
	V	V	uppercase letter
	W	W	uppercase letter
	X	X	uppercase letter
	Y	Y	uppercase letter
	Z	Z	uppercase letter
	[	[	left square bracket
	\	\	backslash
	]	]	right square bracket
	^	^	caret
	_	_	underscore
	`	`	grave accent
	a	a	lowercase letter
	b	b	lowercase letter
	c	c	lowercase letter
	d	d	lowercase letter
	e	e	lowercase letter
	f	f	lowercase letter
	g	g	lowercase letter
	h	h	lowercase letter
	i	i	lowercase letter
	j	j	lowercase letter
	k	k	lowercase letter
	l	l	lowercase letter
	m	m	lowercase letter
	n	n	lowercase letter
	o	o	lowercase letter
	p	p	lowercase letter
	q	q	lowercase letter
	r	r	lowercase letter
	s	s	lowercase letter
	t	t	lowercase letter
	u	u	lowercase letter
	v	v	lowercase letter
	w	w	lowercase letter
	x	x	lowercase letter
	y	y	lowercase letter
	z	z	lowercase letter
	{	{	left curly brace
			|	vertical bar
	}	}	right curly brace
	~	~	tilde
–	–	–	en dash
—	—	—	em dash
			nonbreaking space
¡	¡	¡	inverted exclamation
¢	¢	¢	cent sign
£	£	£	pound sterling
¤	¤	¤	general currency sign
¥	¥	¥	yen sign
¦ &brkbar;	¦	¦	broken vertical bar
§	§	§	section sign
¨ ¨	¨	¨	umlaut
©	©	©	copyright
ª	ª	ª	feminine ordinal
«	«	«	left angle quote
¬	¬	¬	not sign
		-	soft hyphen
®	®	®	registered trademark
¯ &hibar;	¯	¯	macron accent
°	°	°	degree sign
±	±	±	plus or minus
²	²	²	superscript two
³	³	³	superscript three
´	´	´	acute accent
µ	µ	µ	micro sign
¶	¶	¶	paragraph sign

Friendly	Numeric		Description
·	·	·	middle dot
¸	¸	¸	cedilla
¹	¹	¹	superscript one
º	º	º	masculine
»	»	»	right angle
¼	¼	¼	one-fourth
½	½	½	one-half
¾	¾	¾	three-fourths
¿	¿	¿	inverted
À	À	À	A, grave
Á	Á	Á	A, acute accent
Â	Â	Â	A, circumflex
Ã	Ã	Ã	A, tilde
Ä	Ä	Ä	A, umlaut
Å	Å	Å	A, ring
Æ	Æ	Æ	AE
Ç	Ç	Ç	C, cedilla
È	È	È	E, grave accent
É	É	É	E, acute accent
Ê	Ê	Ê	E, circumflex
Ë	Ë	Ë	E, umlaut
Ì	Ì	Ì	I, grave accent
Í	Í	Í	I, acute accent
Î	Î	Î	I, circumflex
Ï	Ï	Ï	I, umlaut
Ð	Ð	Ð	Eth, Icelandic
Ñ	Ñ	Ñ	N, tilde
Ò	Ò	Ò	O, grave
Ó	Ó	Ó	O, acute accent
Ô	Ô	Ô	O, circumflex
Õ	Õ	Õ	O, tilde
Ö	Ö	Ö	O, umlaut
×	×	×	multiplication
Ø	Ø	Ø	O, slash
Ù	Ù	Ù	U, grave accent
Ú	Ú	Ú	U, acute accent
Û	Û	Û	U, circumflex
Ü	Ü	Ü	U, umlaut
Ý	Ý	Ý	Y, acute accent
Þ	Þ	Þ	THORN,
ß	ß	ß	sharps,
à	à	à	a, grave accent
á	á	á	a, acute accent
â	â	â	a, circumflex
ã	ã	ã	a, tilde
ä	ä	ä	a, umlaut
å	å	å	a, ring
æ	æ	æ	ae
ç	ç	ç	c, cedilla
è	è	è	e, grave accent
é	é	é	e, acute accent
ê	ê	ê	e, circumflex
ë	ë	ë	e, umlaut
ì	ì	ì	i, grave accent
í	í	í	i, acute accent
î	î	î	i, circumflex
ï	ï	ï	i, umlaut
ð	ð	ð	eth, Icelandic
ñ	ñ	ñ	n, tilde
ò	ò	ò	o, grave accent
ó	ó	ó	o, acute accent
ô	ô	ô	o, circumflex
õ	õ	õ	o, tilde
ö	ö	ö	o, umlaut
÷	÷	÷	division sign
ø	ø	ø	o, slash
ù	ù	ù	u, grave accent
ú	ú	ú	u, acute accent
û	û	û	u, circumflex
ü	ü	ü	u, umlaut
ý	ý	ý	y, acute accent
þ	þ	þ	thorn, Icelandic
ÿ	ÿ	ÿ	y, umlaut

Source : www.LookupTables.com

Information Storage

Code

- Different machine types use different and incompatible instructions and encodings
- Even identical processors running different OSs have differences in their coding conventions and hence are not binary compatible

- Example) C code and its corresponding machine codes

```
int sum(int x, int y) {
    return x + y;
}
```

Linux 32:	55 89 e5 8b 45 0c 03 45 08 c9 c3
Windows:	55 89 e5 8b 45 0c 03 45 08 5d c3
Sun:	81 c3 e0 08 90 02 00 09
Linux 64:	55 48 89 e5 89 7d fc 89 75 f8 03 45 fc c9 c3

- [Refer to text, Chap. 3]

Information Storage

- **Boolean algebra**
  - Defined over 2-element set {0, 1}
  - Basic operations
    - ~ (NOT)
    - & (AND)
    - | (OR)
    - ^ (XOR)

Information Storage

Bit-level operations in C

- ~, &, |, ^
- Example)

C expression	Binary expression	Binary result	Hexadecimal result
~0x41	~[0100 0001]	[1011 1110]	0xBE
~0x00	~[0000 0000]	[1111 1111]	0xFF
0x69 & 0x55	[0110 1001] & [0101 0101]	[0100 0001]	0x41
0x69 | 0x55	[0110 1001] | [0101 0101]	[0111 1101]	0x7D

- Can be used for masking
  - Eg) x & 0xFF (low-order byte of a word x)
    when x = 0x1234ABCD, x & 0xFF == 0x000000CD

Information Storage

Logical operations in C

- !, &&, ||
- Treat any non-zero argument as representing TRUE and argument 0 as representing FALSE
- Return 1 or 0 (TRUE for 1, FALSE for 0)
- Example)

Expression	Result
!0x41	0
!0x00	1
!!0x41	1
0x69 && 0x55	1
0x69 || 0x55	1

$\boxed{\text{* Early Evaluation}}$

- Logical operators do not evaluate their second argument if the result of the expression can be determined by evaluating the first argument
  Eg) a && 5/a, p && *p++

Information Storage

Shift operations in C

- <<, >>
- x << k (left shift)
  - x is shifted k bits to the left
  - $[x_{n-1}, x_{n-2}, \ldots, x_0] \rightarrow [x_{n-k-1}, x_{n-k-2}, \ldots, x_0, 0, \ldots, 0]$
- x >> k (logical right shift)
  - $[x_{n-1}, x_{n-2}, \ldots, x_0] \rightarrow [0, \ldots, 0, x_{n-1}, x_{n-2}, \ldots, x_k]$
- x >> k (arithmetic right shift, default for signed data in typical C environments)
  - $[x_{n-1}, x_{n-2}, \ldots, x_0] \rightarrow [x_{n-1}, \ldots, x_{n-1}, x_{n-1}, x_{n-2}, \ldots, x_k]$
- Example)

Operation	Values	
Argument x	[01100011]	[10010101]
x << 4	[00110000]	[01010000]
x >> 4 (logical)	[00000110]	[00001001]
x >> 4 (arithmetic)	[00000110]	[11111001]

Integer Representations

- **Encoding unsigned integers**
- **Encoding signed integers**
  - Signed-magnitude encoding
  - 1's-complement encoding
  - 2's-complement encoding

Integer Representations

Encoding unsigned integers

$$X = [x_{w-1}, x_{w-2}, \ldots, x_0]$$

X = 0000 0111 1101 0011$_2$

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

B2U(X) = $2^{10} + 2^9 + 2^8 + 2^7$
$+ 2^6 + 2^4 + 2^1 + 2^0$
= 1024 + 512 + 256 + 128
+ 64 + 16 + 2 + 1
= 2003

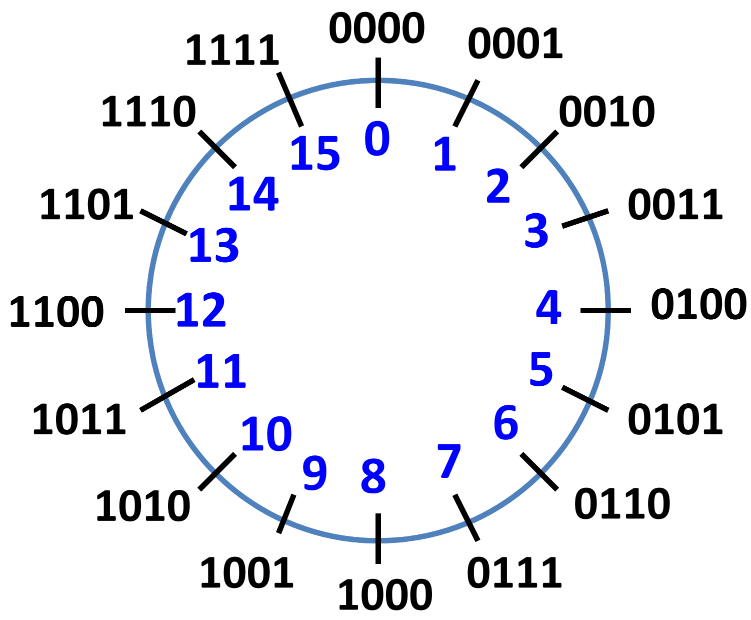
- What is the range for unsigned values with w bits?
  - $0 \sim 2^w - 1$

Integer Representations

- **Encoding unsigned integers**

$$X = [x_{w-1}, x_{w-2}, \ldots, x_0]$$

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



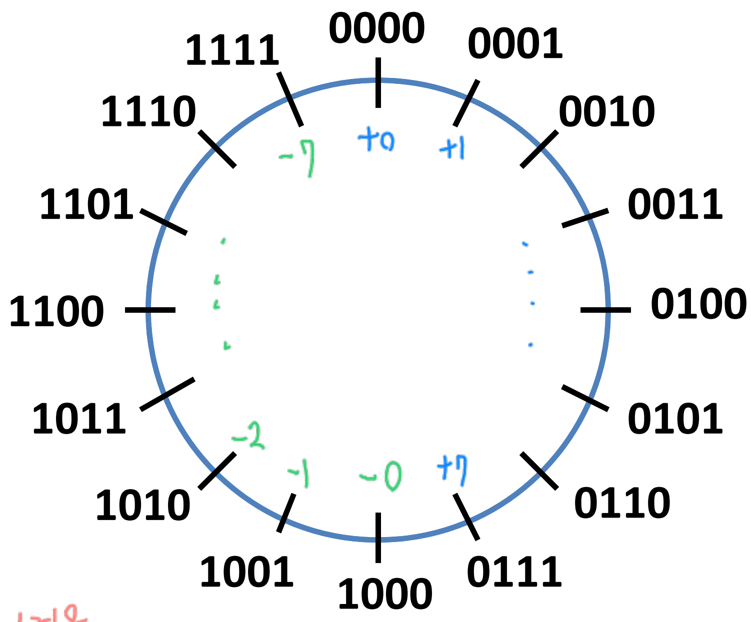
Integer Representations

■ Encoding signed integers: sign-magnitude

x_{w-1}	x_{w-2}	…	…	…	…	x_1	x_0

Sign bit　　　　Magnitude

$$B2S(X) = (-1)^{x_{w-1}} \cdot \left(\sum_{i=0}^{w-2} x_i \cdot 2^i \right)$$



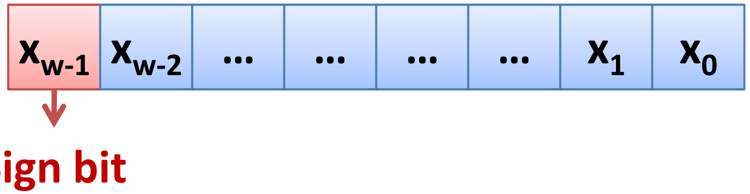
- Two zeros
  - [000…00], [100…00]

+ 사칙연산에 어려움 ⇒ 정수 표현에 잘 이용X

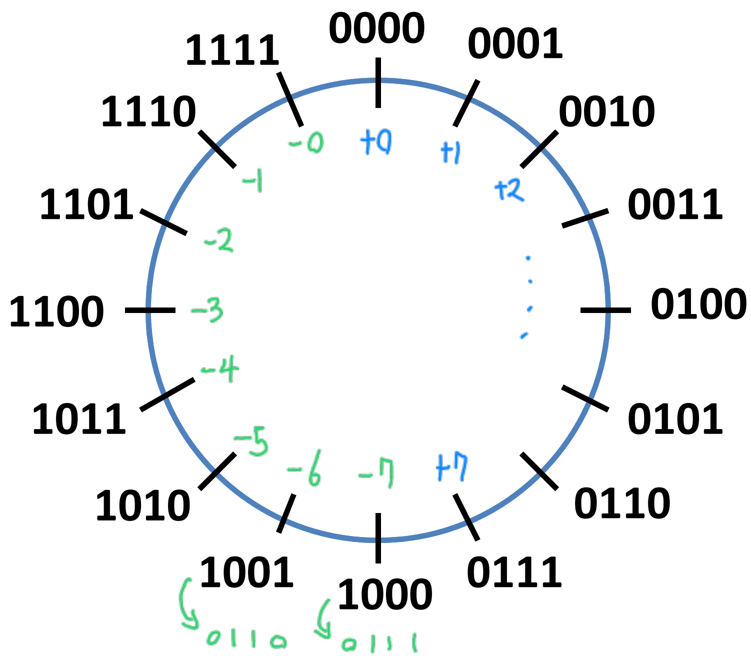
- Used for floating-point numbers

Integer Representations

■ Encoding signed integers: 1's-complement



$$B2O(X) = -x_{w-1}(2^{w-1} - 1) + \left(\sum_{i=0}^{w-2} x_i \cdot 2^i\right)$$



- Easy to find ***−n***
- Two zeros
  - [000…00], [111…11]
- No longer used

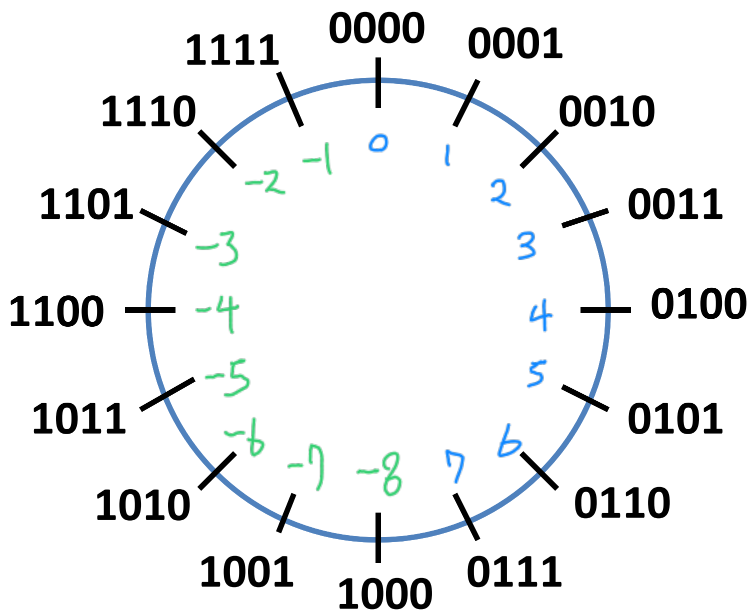
Integer Representations

- **Encoding signed integers: 2's-complement**

x_{w-1}	x_{w-2}	…	…	…	…	x_1	x_0

Sign bit

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



- Unique zero
- Easy for hardware
  - leading 0 ≥ 0, leading 1 < 0
- Used by almost all modern machines

Integer Representations

- **Notes) 2's-complement representation**
  - Following holds for 2's-complement

$$\sim x + 1 == -x$$

  - Observation
    - $\sim x + x == 1111\ldots11_2 == -1$
    - $\sim x + x + (-x + 1) == -1 + (-x + 1)$
    - $\sim x + 1 == -x$

Integer Representations

- **Value ranges of w-bit integers**
  - Unsigned values
    - UMin = 0 [000…00]
    - UMax = $2^w - 1$ [111…11]
  - 2's-complement values
    - TMin = -2^{w-1} [100…00]
    - TMax = $2^{w-1} - 1$ [011…11]

(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

Integer Representations

Value ranges for different word sizes

	w = 8	w = 16	w = 32	w = 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

Observations

- |TMin| = TMax + 1
  (Asymmetric range)
- UMax = 2 * TMax + 1

In C programming

- `#include <limits.h>`
- `INT_MIN, INT_MAX, LONG_MIN, LONG_MAX, UINT_MAX, …`
- Platform-specific values

Integer Representations

- **Value ranges of w-bit integers**
  - Unsigned integer
    - $0 \sim 2^{w} - 1$
  - Signed-magnitude
    - $-(2^{w-1} - 1) \sim (2^{w-1} - 1)$
  - 1's-complement
    - $-(2^{w-1} - 1) \sim (2^{w-1} - 1)$
  - 2's-complement
    - $-2^{w-1} \sim (2^{w-1} - 1)$

Integer Representations

■ Type conversion (signed ↔ unsigned)

- Guess the output

```
short    int     v   = -12345;
unsigned short uv = (unsigned short) v;
printf("v = %d, uv = %u\n", v, uv);
```

⇒ v = -12345 , uv = 53191

```
unsigned u = 4294967295u;   /* UMax_32 */
int      tu = (int) u;
printf("u = %u, tu = %d\n", u, tu);
```

⇒ u= 4294967295 , tu = -1

Integer Representations

- **Type conversion (signed ↔ unsigned)**
  - Principle
    - The effect of casting is to keep the bit values identical but change how these bits are interpreted
      - ✓ Underlying bit representation stays the same

2진 표현을 동일하게 가져가지만
그 해석이 달라진다.

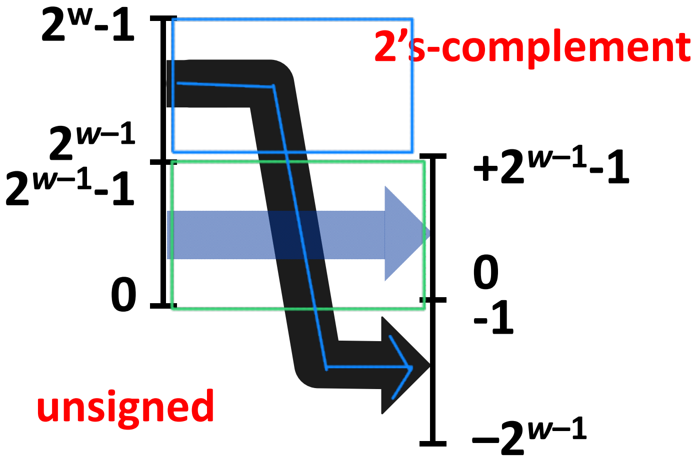
Integer Representations

Type conversion (unsigned → signed)

- The same bit pattern is interpreted as a signed number
- Mathematical analyses

$$U2T_w(x) = \begin{cases} \boxed{x, \quad x < 2^{w-1}} \\ \boxed{x - 2^w, \quad x \geq 2^{w-1}} \end{cases}$$

```
unsigned short    x = 2003;
short             ix = (short) x;
unsigned short    y = 0xbabe;
short             iy = (short) y;
```



	Decimal	Hex	Binary
x	2003	07 D3	00000111 11010011
ix	2003	07 D3	00000111 11010011
y	47806	BA BE	10111010 10111110
iy	-17730	BA BE	10111010 10111110

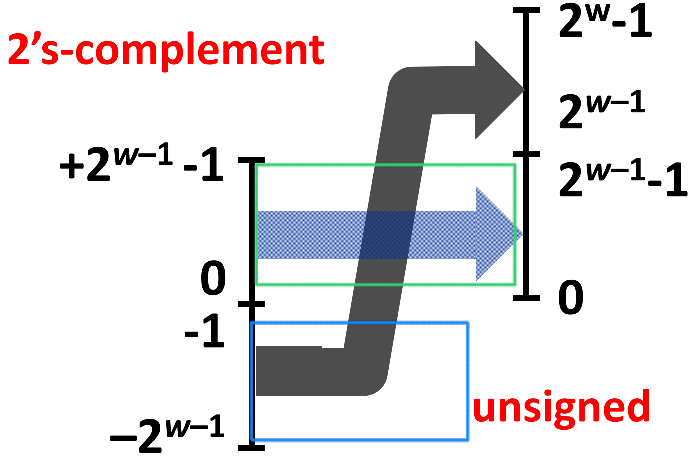
Integer Representations

Type conversion (signed → unsigned)

- Ordering inversion (negative → big positive)
- Mathematical analyses

$$T2U_w(x) = \begin{cases} \boxed{x + 2^w, \quad x < 0} \\ \boxed{x, \quad x \geq 0} \end{cases}$$

```
short           ix = 2003;
Unsigned short  x = (unsigned short) ix;
short           iy = -2003;
unsigned short  y = (unsigned short) iy;
```



	Decimal	Hex	Binary
ix	2003	07 D3	00000111 11010011
x ~~ix~~	2003	07 D3	00000111 11010011
iy	-2003	F8 2D	11111000 00101101
y ~~iy~~	63533	F8 2D	11111000 00101101

Integer Representations

- **Type conversion (signed ↔ unsigned) in C**
  - Same as $U2T_w/T2U_w$
  - C constants
    - By default, most numbers are considered to be signed
    - Use "U" or "u" as a suffix for unsigned constants
      - ✓ 0U, 12345U, 0x1A2Bu
  - When conversion occurs?
    - Explicit casting
    - Implicit casting
      - ✓ In assignments and procedure calls
    - Printing numeric values with printf()

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

```
int         f(unsigned);
tx = ux;
f(ty);
```

Integer Representations

- **Type conversion (signed ↔ unsigned) in C**
  - C rules
    - When an operation is performed where one operand is signed and the other is unsigned
      - ✓ C implicitly casts the signed argument to unsigned and performs the operations assuming the numbers are nonnegative
        - Little difference for standard arithmetic operations
        - But non-intuitive results for relational operators (<, >, ==, <=, >=, !=)

Integer Representations

■ Type conversion (signed ↔ unsigned) in C

- Example) subtleties in computation with relational operators

Expression	Type	Evaluation
0 == 0U	unsigned	1
-1 < 0	signed	1
UINT_MAX ← -1 < 0U	unsigned	0 (False)
-1 > -2	signed	1
(unsigned) -1 > -2	unsigned	1
2147483647 > -2147483647-1	signed	1
2147483647U > -2147483647-1	unsigned	0
2147483647 > (int) 2147483648U	signed	1

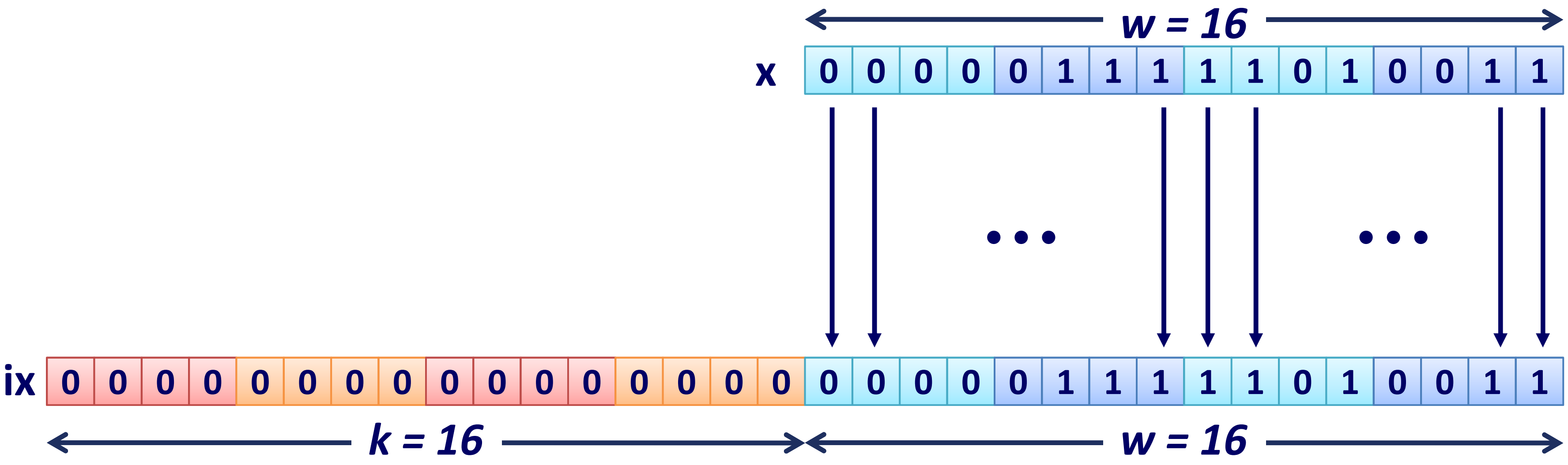
Integer Representations

- **Expanding the bit representation**
  - Zero extension
  - Sign extension

Integer Representations

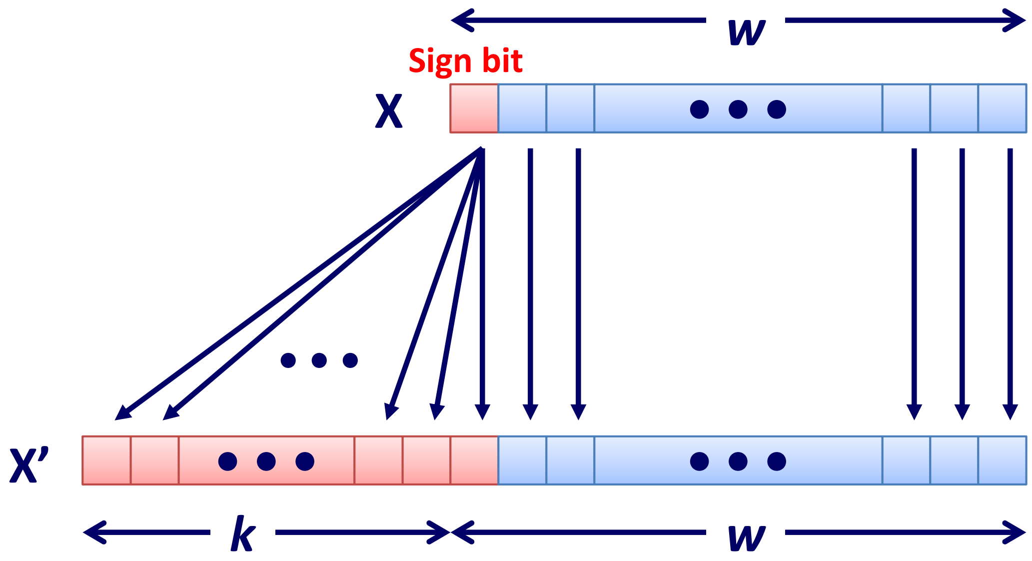
- **Expansion for unsigned: *w* bits → *w+k* bits**
  - **Zero extension**
    - Just fill left ***k***-bits with 0's

```
unsigned short   x   =  2003;
unsigned         ix  =  (unsigned) x;
```



Integer Representations

- **Expansion for signed: *w* bits → *w+k* bits**
  - **Sign extension**
    - Add copies of most significant bit (sign bit)
    - Given ***w***-bit signed integer **x**,
      convert it to (***w+k***)-bit integer with the same value



Integer Representations

- **Expansion for signed: Example**
  - Converting from smaller to larger integer type
  - C automatically performs sign extension

w	Decimal	Binary	Description
4	5	0101	4 + 1 = 5
5	5	0 0101	4 + 1 = 5
8	5	0000 0101	4 + 1 = 5
4	-5	1011	-8 + 2 + 1 = -5
5	-5	1 1011	-16 + 8 + 2 + 1 = -5
8	-5	1111 1011	-128 + 64 + 32 + 16 + 8 + 2 + 1 = -5

Integer Representations

Expansion for signed: Example

- Converting from smaller to larger integer type
- C automatically performs sign extension

```
short int   x  = 2003;
int         ix = (int) x;
short int   y  = -2003;
int         iy = (int) y;
```

	Decimal	Hex	Binary
x	2003	07 D3	00000111 11010011
ix	2003	00 00 07 D3	00000000 00000000 00000111 11010011
y	-2003	F8 2D	11111000 00101101
iy	-2003	FF FF F8 2D	11111111 11111111 11111000 00101101

Integer Representations

- **Truncating for unsigned & signed: *w* bits → *k* bits**
  - Just drop the high order (***w-k***)-bits
  - Equivalent to computing x mod 2^k for unsigned and **U2T**$_k$(x mod 2^k) for signed

```
unsigned int    x  = 0xcafebabe;
unsigned short  sx = (unsigned short) x;
int             y  = 0x2003beef;
short           sy = (short) y;
```

	Decimal	Hex	Binary
x	3405691582	CA FE BA BE	11001010 11111110 10111010 10111110
sx	47806	BA BE	10111010 10111110
y	537116399	20 03 BE EF	00100000 00000011 10111110 11101111
sy	-16657	BE EF	10111110 11101111

Integer Representations

Example-1) Type conversion in C

```
int main ()
{
  unsigned  i;
  for (i = 10; i > 0; i--)
    printf("%u\n", i);
}
```

10
9
8
7
6
5
4
3
2
1

```
int main ()
{
  unsigned  i;
  for (i = 10; i >= 0; i--)
    printf("%u\n", i);
}
```

i=0 일때
마지막으로 i--를 수행하면
i= UINT_MAX

10
9
8
7
6
5
4
3
2
1
0

Integer Representations

■ Example-2) Type conversion in C

```
int sum_array(int a[], unsigned len)
{
  int i;
  int result = 0;

  for (i = 0; i <= len - 1; i++)
    result += a[i];

  return result;
}
```

when len == 0

문제 존재

OU

⇒ OU +(-1) = UINT_MAX

-1 → 111...111 → UINT_MAX

Computes the sum of *len* elements of array *a*

Integer Representations

- **Example-3) Type conversion in C**

```
int strlonger(char *s, char *t)
{
  return strlen(s) – strlen(t) > 0;
}
```

0U - 1U

Compares the length of two strings

```
int strlonger(char *s, char *t)
{
  return strlen(s) > strlen(t);
}
```

Compares the length of two strings

Integer Representations

Example-4) Type conversion in C

```
#include <stdio.h>
int main ()
{
  unsigned char c;

  while ((c = getchar()) != EOF)
    putchar(c);
}
```

Copy standard input (or a file) onto standard output (or another file)

Integer Representations

- **Advices on signed vs. unsigned**
  - There are many tricky situations when you use unsigned integers (hard to debug)
  - Do not use unsigned just because numbers are nonnegative
  - Use unsigned only when you need collections of bits with no numeric interpretation ("flags")
  - Few languages other than C support unsigned integers

Summary