

Bits, Bytes and Words

Troels Henriksen

25-11-2021

Based on slides by Randal E. Bryant and David R. O'Hallaron

Agenda

Representing information as bits

Bit-level manipulation

Integers

- Representation: unsigned and signed

- Conversion, casting

- Expanding, truncating

Representing information as bits

Bit-level manipulation

Integers

- Representation: unsigned and signed

- Conversion, casting

- Expanding, truncating

Everything is bits

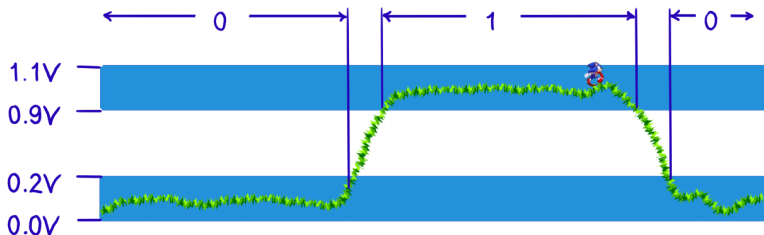
- Each bit is 0 or 1
- By interpreting sets of bits in various ways...
 - ▶ ...computers determine what to do.
 - ▶ ...represent and manipulate numbers, sets, strings—*data*.

Why bits? Why not decimals? Could it have been some other way?

Everything is bits

- **Why bits? Electronic implementation.**

- ▶ Easy to store with bistable elements.
- ▶ Reliably transmitted on noisy and inaccurate wires (error correction).



- **... But there exist models that do not use bits.**

- ▶ The Soviet Setun computer used ternary *trits*.
- ▶ Quantum computers use *qubits* that are in a superposition of the two states.
 - ▶ ...error correction is the main challenge here.

Binary numbers

- **Base 2 number representation.**

- ▶ Represent 15213_{10} as 11101101101101_2
- ▶ Represent 1.20_{10} as $1.0011001100110011[0011] \dots_2$
- ▶ Represent 1.5213×10^4 as $1.1101101101101_2 \times 2^{13}$

- **Machine numbers are of some finite size.**

- ▶ If we use k bits to represent a number, only 2^k distinct values are possible.
- ▶ How we interpret those bits can vary.
- ▶ **Why do we use finite-sized numbers?**

Binary numbers

- **Base 2 number representation.**

- ▶ Represent 15213_{10} as 11101101101101_2
- ▶ Represent 1.20_{10} as $1.0011001100110011[0011] \dots_2$
- ▶ Represent 1.5213×10^4 as $1.1101101101101_2 \times 2^{13}$

- **Machine numbers are of some finite size.**

- ▶ If we use k bits to represent a number, only 2^k distinct values are possible.
- ▶ How we interpret those bits can vary.
- ▶ **Why do we use finite-sized numbers?**
- ▶ A “ k -bit machine” handles numbers of up to k bits “natively” (meaning fast).

Encoding byte values

Byte = 8 bits

- (Machine-specific, but is true for all mainstream machines.)
- 256 different values.
- Binary 00000000_2 to 11111111_2 .
- Decimal 0_{10} to 255_{10} .
- Hexadecimal 00_{16} to FF_{16} .
 - ▶ Base 16 number representation.
 - ▶ Uses characters 0–9 and A–F.
 - ▶ In C we write $FA1D37B_{16}$ as
 - ▶ $0xFA1D37B$
 - ▶ $0xfa1d37b$ (case does not matter)

Hex	Dec	Bin
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

Let's play a game

<http://topps.diku.dk/compsys/integers.html>

Example data representations

C data type	Typical 16-bit	Typical 32-bit	Typical 64-bit	x86-64
char	1	1	1	1
short	1	2	2	2
int	2	4	4	4
long	4	4	8	8
int32_t	4	4	4	4
int64_t	8	8	8	8
float	4	4	4	4
double	8	8	8	8
long double	-	-	-	10
pointer	2	4	8	8

Representing information as bits

Bit-level manipulation

Integers

- Representation: unsigned and signed

- Conversion, casting

- Expanding, truncating

Boolean algebra

Developed by George Boole in 19th century

- Algebraic representation of logic (“truth values”).
- Encode *true* as 1 and *false* as 0.

And

&	0	1
0	0	0
1	0	1

Or

	0	1
0	0	1
1	1	1

Not

~	
0	1
1	0

Exclusive-or

^	0	1
0	0	1
1	1	0

- These operations can be implemented with tiny electronic *gates*.

General boolean algebras

- The truth tables generalise to operate on *bit vectors*, applied elementwise.

01101001	01101001	01101001	
& 01010101	01010101	^ 01010101	~ 01101001
<hr/>	<hr/>	<hr/>	<hr/>
00100001	01111101	00111100	10010110

- This is the form they take when available in programming languages such as C.

Bit-level operations in C

Operations $\&$, $|$, \sim , \wedge available in C.

- Apply to any integral type.
 - ▶ E.g. long, int, short, char...
- Interpret operands as bit vectors.
- Applied bit-wise.

Examples

- $\sim 0x41 = 0xBE$
 - ▶ $\sim 01000001_2 = 10111110_2$
- $\sim 0x00 = 0xFF$
 - ▶ $\sim 00000000_2 = 11111111_2$
- $0x69 \& 0x55 = 0x41$
 - ▶ $01101001_2 \& 01010101_2 = 01000001_2$
- $0x69 \& 0x55 = 0x7D$
 - ▶ $01101001_2 \& 01010101_2 = 01111101_2$

Contrast: logical operators in C

The logical operators interpret numbers as *single boolean values*, not as bit vectors!

- **&&, ||, !**

- ▶ View 0 as false.
- ▶ Anything nonzero as true.
- ▶ Always produce 0 or 1.
- ▶ **Early termination:** `1 || (0/0)` is safe.

- **Examples**

- ▶ `!0x41 = 0x00`
- ▶ `!0x00 = 0x01`
- ▶ `!!0x41 = 0x01`
- ▶ `0x69 && 0x55 = 0x01`
- ▶ `0x69 || 0x55 = 0x01`

- **Do not confuse the logical and bitwise operators!**

Shift operations

▪ Left shift $x \ll y$

- ▶ Shift bit-vector x left by y positions.
 - ▶ Throws away excess bits on the left.
 - ▶ Fills with zeroes on right.

▪ Right shift $x \gg y$

- ▶ Shift bit-vector x right by y positions.
 - ▶ Throws away excess bits on the left.
- ▶ Logical shift: Fill with 0s on left.
- ▶ Arithmetic shift: Replicate most significant bit on left.

▪ Undefined behaviour

- ▶ Shifting a negative amount or by the vector size or more.

x		01100010
<hr/>		
$x \ll 3$		00010000
$x \gg 2$ (log)		00011000
$x \gg 2$ (arith)		00011000

x		10100010
<hr/>		
$x \ll 3$		00010000
$x \gg 2$ (log)		00101000
$x \gg 2$ (arith)		11101000

Representing information as bits

Bit-level manipulation

Integers

- Representation: unsigned and signed

- Conversion, casting

- Expanding, truncating

Encoding integers

Suppose x_i is the i th bit of a w -bit word (with x_0 being the least significant bit).

Unsigned

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

Two's complement

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

```
int16_t x = 15213;  
int16_t y = -15213;
```

	Decimal	Hex	Binary
x	15213	3 B 5 D	0011 1011 0110 1101
y	-15213	C 4 9 3	1100 0100 1001 0011

Sign bit

- For 2's complement, most significant bit (x_{w-1}) indicates sign.
 - 0 for non-negative.
 - 1 for negative.

Two's complement encoding example

```
int16_t x = 15213; // 0011 1011 0110 1101  
int16_t y = -15213; // 1100 0100 1001 0011
```

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	2047	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	

Numeric ranges

Unsigned

$$UMin = 0 = 0 \dots 0_2$$

$$UMax = 2^{w-1} = 1 \dots 1_2$$

Two's complement

$$TMin = -2^{w-1} = 10 \dots 0_2$$

$$TMax = 2^{w-1} - 1 = 01 \dots 1_2$$
$$-1 = 1 \dots 1_2$$

Values for $w = 16$:

	Decimal	Hex	Binary
UMax	65535	F F F F	1111 1111 1111 1111
TMax	32767	7 F F F	0111 1111 1111 1111
TMin	-32768	8 0 0 0	1000 0000 0000 0000
-1	-1	F F F F	1111 1111 1111 1111
0	0	0 0 0 0	0000 0000 0000 0000

Values for different word sizes

	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

Observations

$$|TMin| = TMax + 1$$

$$|UMax| = 2 \cdot TMax + 1$$

Note the assymetric range.

C Programming

- `#include <limits.h>`
- Declares constants, e.g:
 - ▶ `ULONG_MAX`
 - ▶ `LONG_MAX`
 - ▶ `LONG_MIN`
- Values are platform-specific.

Unsigned and signed numeric values (here $w = 4$)

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

- **Equivalence**

- ▶ Same encoding for non-negative values.

- **Uniqueness**

- ▶ Every bit pattern represents distinct integer value.
- ▶ Each representable integer has unique bit encoding.
- ▶ The representation is **bijective**.

- **Can invert mappings**

- ▶ $U2B(x) = B2U^{-1}(x)$
 - ▶ Bit pattern for unsigned integer.
- ▶ $T2B(x) = B2T^{-1}(x)$
 - ▶ Bit pattern for two's complement integer.

Representing information as bits

Bit-level manipulation

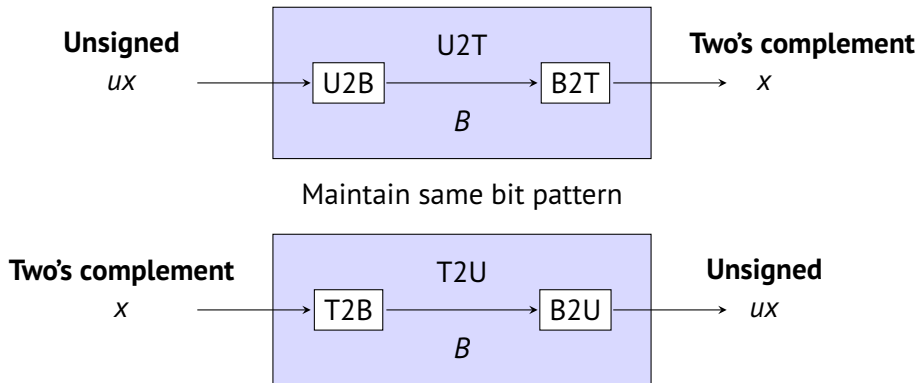
Integers

Representation: unsigned and signed

Conversion, casting

Expanding, truncating

Mapping between signed and unsigned

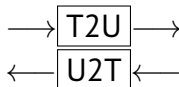


Mapping between unsigned and two's complement numbers:
Keep bit representations and reinterpret.

Mapping signed \Leftrightarrow unsigned

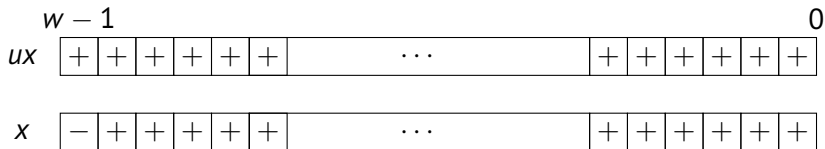
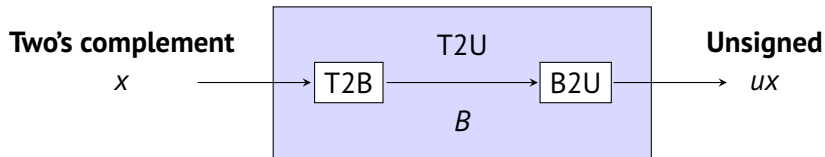
Bits
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

Signed
0
1
2
3
4
5
6
7
-8
-7
-6
-5
-4
-3
-2
-1



Unsigned
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Relation between signed and unsigned

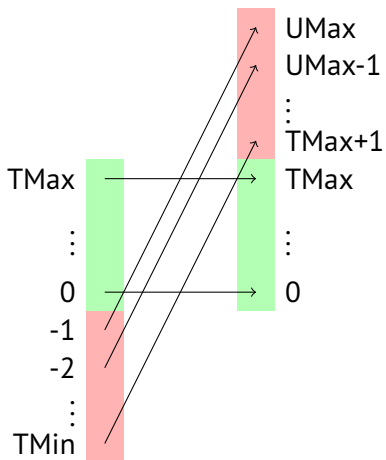


Large negative weight becomes **large positive weight**.

Conversion (that is, *reinterpretation*) visualized

Two's complement to unsigned

- Ordering inversion.
- Negative numbers become large positive numbers.



Signed versus unsigned in C

C makes working with this more error-prone than it should be.

Types ■ Signedness part of type: `unsigned int`, `int32_t`, `uint32_t`.

Constants ■ By default are considered signed integers.

■ Unsigned with U suffix: `0U`, `4294967259U`

Casting ■ Explicit casting between signed and unsigned:

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

■ Implicit casting due to assignments and other expressions:

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

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	$==$	0U	

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	==	0U	unsigned

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	$==$	0U	unsigned
-1	$<$	0	

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	$==$	0U	unsigned
-1	$<$	0	signed

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	$==$	0U	unsigned
-1	$<$	0	signed
-1	$>$	0U	

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	$==$	0U	unsigned
-1	$<$	0	signed
-1	$>$	0U	unsigned

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	$==$	0U	unsigned
-1	$<$	0	signed
-1	$>$	0U	unsigned
2147483647	$>$	-2147483647-1	

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	$==$	0U	unsigned
-1	$<$	0	signed
-1	$>$	0U	unsigned
2147483647	$>$	-2147483647-1	signed

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	$==$	0U	unsigned
-1	$<$	0	signed
-1	$>$	0U	unsigned
2147483647	$>$	-2147483647-1	signed
2147483647U	$<$	-2147483647-1	

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	$==$	0U	unsigned
-1	$<$	0	signed
-1	$>$	0U	unsigned
2147483647	$>$	-2147483647-1	signed
2147483647U	$<$	-2147483647-1	unsigned

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	$==$	0U	unsigned
-1	$<$	0	signed
-1	$>$	0U	unsigned
2147483647	$>$	-2147483647-1	signed
2147483647U	$<$	-2147483647-1	unsigned
-1	$>$	-2	

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	$==$	0U	unsigned
-1	$<$	0	signed
-1	$>$	0U	unsigned
2147483647	$>$	-2147483647-1	signed
2147483647U	$<$	-2147483647-1	unsigned
-1	$>$	-2	signed

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	$==$	0U	unsigned
-1	$<$	0	signed
-1	$>$	0U	unsigned
2147483647	$>$	-2147483647-1	signed
2147483647U	$<$	-2147483647-1	unsigned
-1	$>$	-2	signed
(unsigned int)-1	$>$	-2	

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	$==$	0U	unsigned
-1	$<$	0	signed
-1	$>$	0U	unsigned
2147483647	$>$	-2147483647-1	signed
2147483647U	$<$	-2147483647-1	unsigned
-1	$>$	-2	signed
(unsigned int)-1	$>$	-2	unsigned

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	$==$	0U	unsigned
-1	$<$	0	signed
-1	$>$	0U	unsigned
2147483647	$>$	-2147483647-1	signed
2147483647U	$<$	-2147483647-1	unsigned
-1	$>$	-2	signed
(unsigned int)-1	$>$	-2	unsigned
2147483647	$<$	2147483648U	

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	$==$	0U	unsigned
-1	$<$	0	signed
-1	$>$	0U	unsigned
2147483647	$>$	-2147483647-1	signed
2147483647U	$<$	-2147483647-1	unsigned
-1	$>$	-2	signed
(unsigned int)-1	$>$	-2	unsigned
2147483647	$<$	2147483648U	unsigned

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	$==$	0U	unsigned
-1	$<$	0	signed
-1	$>$	0U	unsigned
2147483647	$>$	-2147483647-1	signed
2147483647U	$<$	-2147483647-1	unsigned
-1	$>$	-2	signed
(unsigned int)-1	$>$	-2	unsigned
2147483647	$<$	2147483648U	unsigned
2147483647	$>$	(int) 2147483648U	

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	$==$	0U	unsigned
-1	$<$	0	signed
-1	$>$	0U	unsigned
2147483647	$>$	-2147483647-1	signed
2147483647U	$<$	-2147483647-1	unsigned
-1	$>$	-2	signed
(unsigned int)-1	$>$	-2	unsigned
2147483647	$<$	2147483648U	unsigned
2147483647	$>$	(int) 2147483648U	signed

Casting surprises

- Evaluation**
- If there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*.
 - Including comparison operations $<$, $>$, $==$, $<=$, $>=$.
 - Examples for
 $w = 32$: $TMIN = -2,147,483,648$, $TMAX = 2,147,483,647$:

Const LHS	Relation	Const RHS	Evaluation
0	$==$	0U	unsigned
-1	$<$	0	signed
-1	$>$	0U	unsigned
2147483647	$>$	-2147483647-1	signed
2147483647U	$<$	-2147483647-1	unsigned
-1	$>$	-2	signed
(unsigned int)-1	$>$	-2	unsigned
2147483647	$<$	2147483648U	unsigned
2147483647	$>$	(int) 2147483648U	signed

Casting between signed and unsigned: basic rules

- Bit pattern is maintained.
- ...but reinterpreted.
- Can have unexpected effects: adding or subtracting 2^w .
- Expression containing signed and unsigned int:
 - ▶ `int` is cast to unsigned `int`!
 - ▶ **When can this go bad?**

Casting between signed and unsigned: basic rules

- Bit pattern is maintained.
- ...but reinterpreted.
- Can have unexpected effects: adding or subtracting 2^w .
- Expression containing signed and unsigned int:
 - ▶ int is cast to unsigned int!
 - ▶ **When can this go bad?**

```
for (unsigned int i = n-1; i >= 0; i--) {  
    // do something with x[i]  
}
```

Casting between signed and unsigned: basic rules

- Bit pattern is maintained.
- ...but reinterpreted.
- Can have unexpected effects: adding or subtracting 2^w .
- Expression containing signed and unsigned int:
 - ▶ `int` is cast to unsigned `int`!
 - ▶ **When can this go bad?**

```
for (unsigned int i = n-1; i >= 0; i--) {  
    // do something with x[i]  
}
```

Advice: Never do arithmetic on unsigned types—only use them for bit operations.

But: Some C operators (`sizeof`) and many functions return unsigned types (e.g. `size_t`).

Representing information as bits

Bit-level manipulation

Integers

Representation: unsigned and signed

Conversion, casting

Expanding, truncating

Truncation

- Task**
- Given $k + w$ -bit signed integer x .
 - Convert it to w -bit integer x' with same value if possible.

- Approach**
- Remove the k most significant bits.
 - Equivalent to computing $x' = x \bmod 2^w$.
 - Can cause numerical change if number has no representation in w bits.
 - Otherwise safe.

w	Bits	Two's complement
8	11111111_2	-1_{10}
4	1111_2	-1_{10}
8	10000000_2	-128_{10}
4	0000_2	0_{10}

Sign extension

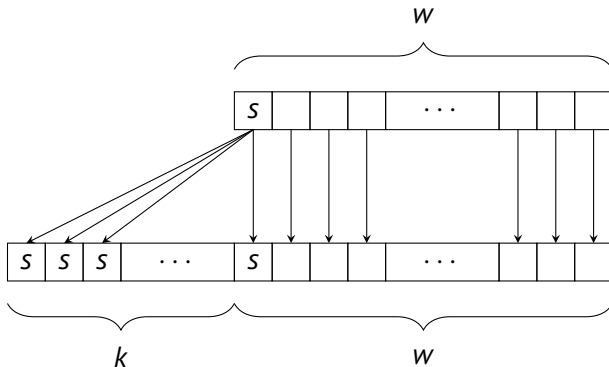
Task

- Given w -bit signed integer x .
- Convert it to $w + k$ -bit integer x' with same value.

Approach

- Make k copies of sign bit (most significant bit):

$$x' = \underbrace{x_{w-1}, \dots, x_{w-1}}_{k \text{ copies of sign bit.}}, x_{w-1}, \dots, x_0$$



Sign extension example

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

	Decimal	Hex	Binary
x	15213	3B 6D	0011 1011 0110 1101
ix	15213	00 00 3B 6D	0000 0000 0000 0000 0011 1011 0110 1101
y	-15213	C4 93	1100 0100 1001 0011
iy	-15213	FF FF 3B 6D	1111 1111 1111 1111 0011 1011 0110 1101

Summary: basic rules for expanding and truncating

Expanding (e.g. `short` to `int`)

- Unsigned: zeros added.
- Signed: sign extension.
- Both yield expected result.

Truncating (e.g. `unsigned int` to `unsigned short`)

- Bits are truncated.
- Result reinterpreted.
- Unsigned: modulo operation.
- Signed: similar to a modulo operation.
- For small numbers yield expected behaviour.