

# Bits, Bytes and Words

Troels Henriksen

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

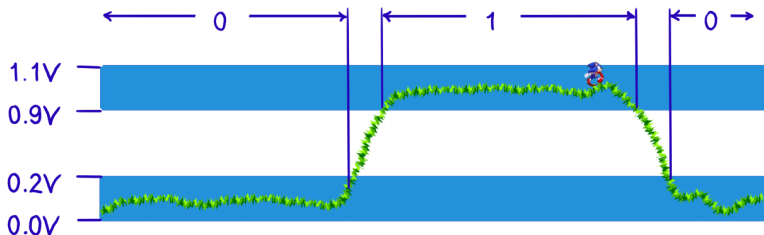
- Each bit is 0 or 1
- By interpreting sets of bits in various ways...
  - ▶ ...computers determine what to do.
  - ▶ ...represent and manipulate numbers, sets, strings—*data*.

**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.

# Everything is bit vectors

A sequence of bits is called a *bit vector*

$$\langle x_{w-1}, \dots, x_0 \rangle$$

- Number bits from 0 to  $w - 1$ .
- Bit  $x_0$  typically called *least significant* and  $x_{w-1}$  *most significant*.
  - ▶ Due to how bit vectors can be interpreted as binary numbers.
- **Bit vectors are not numbers.**
  - ▶ Can represent many kinds of objects.
  - ▶ ...but we will mostly focus on number representations.

# Binary numbers

- **Base 2 numbers.**

- ▶ Represent  $15213_{10}$  as  $11101101101101_2$ 
  - ▶  $\langle 0011\ 1011\ 0110\ 1101 \rangle$  (with  $w = 16$ )
- ▶ Represent  $1.5213 \times 10^4$  as  $1.1101101101101_2 \times 2^{13}$ 
  - ▶  $\langle 0000\ 0000\ 0000\ 0001\ 1101\ 1011\ 0110\ 1000 \rangle$  ( $w = 32$ )
  - ▶ 16 bits before and after *binial point*.

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

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

# Encoding byte values

## Byte = 8 bit word

- (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	Bits
0	0	$\langle 0000 \rangle$
1	1	$\langle 0001 \rangle$
2	2	$\langle 0010 \rangle$
3	3	$\langle 0011 \rangle$
4	4	$\langle 0100 \rangle$
5	5	$\langle 0101 \rangle$
6	6	$\langle 0110 \rangle$
7	7	$\langle 0111 \rangle$
8	8	$\langle 1000 \rangle$
9	9	$\langle 1001 \rangle$
A	10	$\langle 1010 \rangle$
B	11	$\langle 1011 \rangle$
C	12	$\langle 1100 \rangle$
D	13	$\langle 1101 \rangle$
E	14	$\langle 1110 \rangle$
F	15	$\langle 1111 \rangle$



# 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**

$\wedge$	0	1
0	0	0
1	0	1

**Not**

$\neg$	
0	1
1	0

**Or**

$\vee$	0	1
0	0	1
1	1	1

**Exclusive-or**

$\oplus$	0	1
0	0	1
1	1	0

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

# Boolean algebra on bit vectors

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

$$\begin{array}{cccc} \begin{array}{c} \langle 01101001 \rangle \\ \wedge \langle 01010101 \rangle \\ \hline \langle 01000001 \rangle \end{array} & \begin{array}{c} \langle 01101001 \rangle \\ \vee \langle 01010101 \rangle \\ \hline \langle 01111101 \rangle \end{array} & \begin{array}{c} \langle 01101001 \rangle \\ \oplus \langle 01010101 \rangle \\ \hline \langle 00111100 \rangle \end{array} & \begin{array}{c} \neg \langle 01101001 \rangle \\ \hline \langle 10010110 \rangle \end{array} \end{array}$$

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

# Bit-level operations in C

## Operators $\&$ ( $\wedge$ ), $|$ ( $\vee$ ), $\wedge$ ( $\oplus$ ), $\sim$ ( $\neg$ ) available in C.

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

## Examples

- $\sim 0x41 = 0xBE$ 
  - ▶  $\neg \langle 01000001 \rangle = \langle 10111110 \rangle$
- $\sim 0x00 = 0xFF$ 
  - ▶  $\langle 00000000 \rangle = \langle 11111111 \rangle$
- $0x69 \ \& \ 0x55 = 0x41$ 
  - ▶  $\langle 01101001 \rangle \wedge \langle 01010101 \rangle = \langle 01000001 \rangle$
- $0x69 \ \& \ 0x55 = 0x7D$ 
  - ▶  $\langle 01101001 \rangle \vee \langle 01010101 \rangle = \langle 01111101 \rangle$

# 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.

$x$	$\langle 01100010 \rangle$
<hr/>	
$x \ll 3$	$\langle 00010000 \rangle$
$x \gg 2$	$\langle 00011000 \rangle$
$x \gg^a 2$	$\langle 00011000 \rangle$

## ▪ 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.

$x$	$\langle 10100010 \rangle$
<hr/>	
$x \ll 3$	$\langle 00010000 \rangle$
$x \gg 2$	$\langle 00101000 \rangle$
$x \gg^a 2$	$\langle 11101000 \rangle$

## ▪ Undefined behaviour in C

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



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

$$\text{Bits2N}(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

## Two's complement

$$\text{TC2Int}(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

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

	Decimal	Hex	Bits
x	15213	3 B 5 D	$\langle 0011\ 1011\ 0110\ 1101 \rangle$
y	-15213	C 4 9 3	$\langle 1100\ 0100\ 1001\ 0011 \rangle$

## 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

$$U_{\min} = 0 = 0 \dots 0_2$$

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

## Two's complement

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

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

Values for  $w = 16$ :

	Decimal	Hex	Binary
U <sub>Max</sub>	65535	F F F F	⟨1111 1111 1111 1111⟩
T <sub>Max</sub>	32767	7 F F F	⟨0111 1111 1111 1111⟩
T <sub>Min</sub>	-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 $w = 4$ )

$x$	Bits2N( $x$ )	TC2Int( $x$ )
$\langle 0000 \rangle$	0	0
$\langle 0001 \rangle$	1	1
$\langle 0010 \rangle$	2	2
$\langle 0011 \rangle$	3	3
$\langle 0100 \rangle$	4	4
$\langle 0101 \rangle$	5	5
$\langle 0110 \rangle$	6	6
$\langle 0111 \rangle$	7	7
$\langle 1000 \rangle$	8	-8
$\langle 1001 \rangle$	9	-7
$\langle 1010 \rangle$	10	-6
$\langle 1011 \rangle$	11	-5
$\langle 1100 \rangle$	12	-4
$\langle 1101 \rangle$	13	-3
$\langle 1110 \rangle$	14	-2
$\langle 1111 \rangle$	15	-1

## ■ Equivalence

- ▶ Same encoding for non-negative values.

## ■ Uniqueness

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

## ■ Can invert mappings

- ▶  $N2Bits(x) = Bits2N^{-1}(x)$ 
  - ▶ Bit vector for unsigned integer.
- ▶  $Int2TC(x) = TC2Int^{-1}(x)$ 
  - ▶ Bit vector for Two's Complement integer.

Representing information as bits

Bit-level manipulation

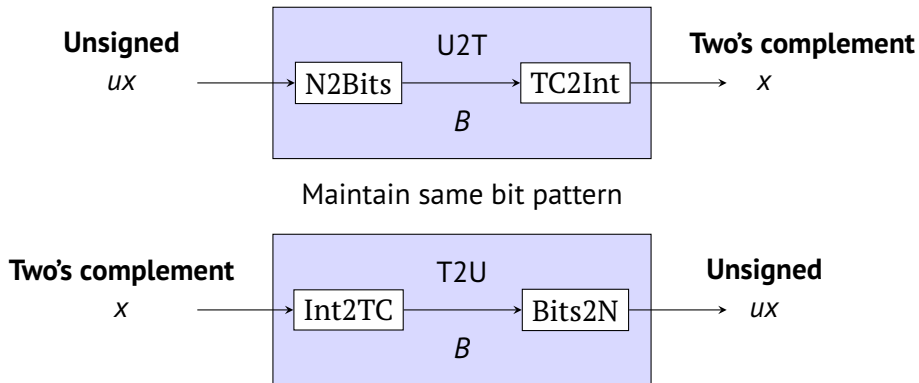
Integers

Representation: unsigned and signed

Conversion, casting

Expanding, truncating

# Mapping between signed and unsigned



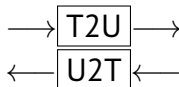
Mapping between unsigned and two's complement numbers:  
**Keep bit representations and reinterpret.**



# Mapping signed $\Leftrightarrow$ unsigned

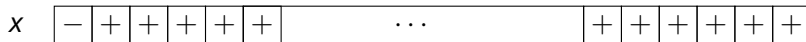
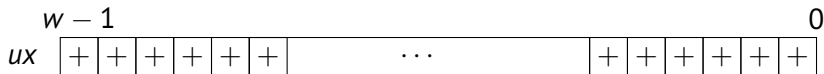
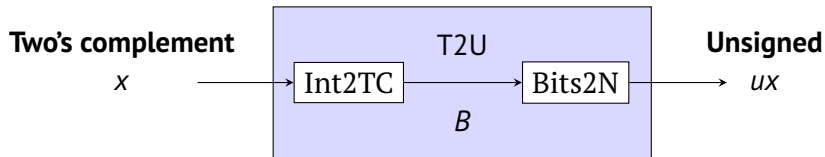
Bits
$\langle 0000 \rangle$
$\langle 0001 \rangle$
$\langle 0010 \rangle$
$\langle 0011 \rangle$
$\langle 0100 \rangle$
$\langle 0101 \rangle$
$\langle 0110 \rangle$
$\langle 0111 \rangle$
$\langle 1000 \rangle$
$\langle 1001 \rangle$
$\langle 1010 \rangle$
$\langle 1011 \rangle$
$\langle 1100 \rangle$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

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

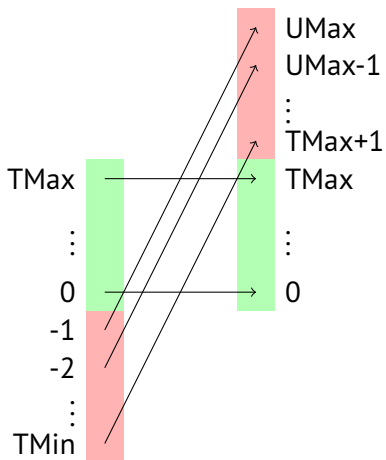


Large negative weight becomes large positive weight.

# 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.**

- |                  |   |
|------------------|---|
| <b>Types</b>     | ▪ Signedness part of type: <code>unsigned int</code> , <code>int32_t</code> , <code>uint32_t</code> .   |
| <b>Constants</b> | ▪ By default are considered signed integers.<br>▪ Unsigned with <code>U</code> suffix: <code>0U</code> , <code>4294967259U</code>   |
| <b>Casting</b>   | ▪ Explicit casting between signed and unsigned:<br><br><pre>int tx, ty;<br/>unsigned int ux, uy;<br/>tx = (int) ux;<br/>uy = (unsigned int) ty;</pre><br>▪ Implicit casting due to assignments and other expressions:<br><br><pre>tx = ux;<br/>uy = ty;</pre> |

# 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 representation 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 representation 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 representation 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:** Avoid 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`). C is always ready to stab you in the back.

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  $x' = x \bmod 2^w$ .
  - Numerical change if number has no representation in  $w$  bits.
  - Otherwise safe.

$w$	$x$	$TC2Int(x)$
8	$\langle 11111111 \rangle$	-1
4	$\langle 1111 \rangle$	-1
8	$\langle 10000000 \rangle$	-128
4	$\langle 0000 \rangle$	0

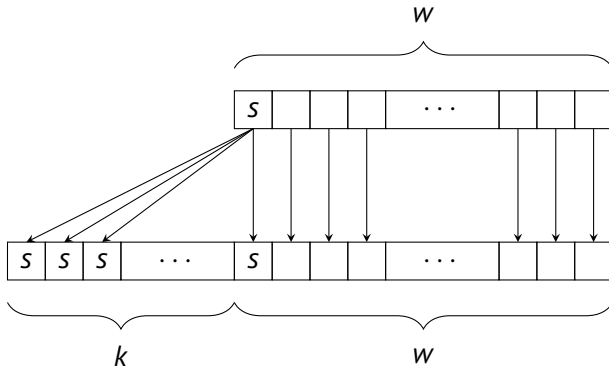
# Sign extension

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



## Sign extension example

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

	Decimal	Hex	Bits
x	15213 <sub>10</sub>	3B 6D	⟨0011 1011 0110 1101⟩
ix	15213 <sub>10</sub>	00 00 3B 6D	⟨0000 0000 0000 0000 0011 1011 0110 1101⟩
y	-15213 <sub>10</sub>	C4 93	⟨1100 0100 1001 0011⟩
iy	-15213 <sub>10</sub>	FF FF C4 93	⟨1111 1111 1111 1111 1100 0100 1001 0011⟩

# 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.