REPRESENTING DATA

Fabio Stroppa

Machine language

Computers understand only one language: *machine language.*

Machine language consists of sets of instructions made of $\underbrace{\text{ones and zeros.}}_{\text{Binary Code}}$

Example of a single instruction: | 00000 | 10011110 |

Everything in the computer is stored as a binary number that codifies specific information.

For example, the values of the data processed by programs are stored as binary numbers.

Unsigned Integer Numbers

An **unsigned integer number** is a is a whole number $\in \mathbb{N}$ (not a fractional number) that does not have a sign (i.e., it can be positive or zero).

Example: representing the integer number 45 in 1 byte (8 bits):

0	0	1	0	1	1	0	1

Unsigned Integer Numbers – MIN and MAX

The minimum and maximum values you can represent depend on the number of bits you have at your disposal.

With 8 bits:

- the **minimum** representable unsigned integer number is `0`

0	0	0	0	0	0	0	0

- the **maximum** representable unsigned integer number is $2^8-1 = 255$

1	1	1	1	1	1	1	1

Signed Integer Numbers

A **signed integer number** is a is a whole number $\in \mathbb{N}$ (not a fractional number) that has a sign (i.e., it can be positive, negative, or zero).

Representing **positive** signed integer numbers is the same as representing unsigned integer numbers.

Example: representing the signed integer number +45 in 1 byte (8 bits):

0	0	1	0	1	1	0	1

There are two ways to represent a **negative** signed integer number:

1. Signed magnitude
2. 2's complement

Signed Integer Numbers – Signed Magnitude

Reserve the first bit as sign:

- `0` stands for `+`
- `1` stands for `-`

Example: representing the signed integer number +45 in 1 byte (8 bits):

0	0	1	0	1	1	0	1

Example: representing the signed integer number -45 in 1 byte (8 bits):

1	0	1	0	1	1	0	1

Signed Integer Numbers – 2’s complement

The 2’s complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Signed Integer Numbers – 2's complement

The 2's complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **`1`** to the entire inverted number.

Example: representing the signed integer number -45 in 1 byte (8 bits):

0	0	1	0	1	1	0	1

Signed Integer Numbers – 2's complement

The 2's complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **`1`** to the entire inverted number.

Example: representing the signed integer number -45 in 1 byte (8 bits):

`INVERT:`	`0`	`0`	`1`	`0`	`1`	`1`	`0`	`1`

Signed Integer Numbers – 2's complement

The 2's complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -45 in 1 byte (8 bits):

INVERT:	0	0	1	0	1	1	0	1
	1	1	0	1	0	0	1	0

Signed Integer Numbers – 2’s complement

The 2’s complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -45 in 1 byte (8 bits):

INVERT:	0	0	1	0	1	1	0	1	
ADD 1:	1	1	0	1	0	0	1	0	+
								1	=

Signed Integer Numbers – 2’s complement

The 2’s complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -45 in 1 byte (8 bits):

INVERT:	0	0	1	0	1	1	0	1	
ADD 1:	1	1	0	1	0	0	1	0	+
								1	=
								1	

Signed Integer Numbers – 2's complement

The 2's complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -45 in 1 byte (8 bits):

INVERT:	0	0	1	0	1	1	0	1	
ADD 1:	1	1	0	1	0	0	1	0	+
								1	=
							1	1	

Signed Integer Numbers – 2's complement

The 2's complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -45 in 1 byte (8 bits):

INVERT:	0	0	1	0	1	1	0	1	
ADD 1:	1	1	0	1	0	0	1	0	+
								1	=
						0	1	1	

Signed Integer Numbers – 2’s complement

The 2’s complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -45 in 1 byte (8 bits):

INVERT:	0	0	1	0	1	1	0	1	
ADD 1:	1	1	0	1	0	0	1	0	+
								1	=
					0	0	1	1	

Signed Integer Numbers – 2’s complement

The 2’s complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -45 in 1 byte (8 bits):

INVERT:	0	0	1	0	1	1	0	1	
ADD 1:	1	1	0	1	0	0	1	0	+
								1	=
				1	0	0	1	1	

Signed Integer Numbers – 2's complement

The 2's complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -45 in 1 byte (8 bits):

INVERT:	0	0	1	0	1	1	0	1	
ADD 1:	1	1	0	1	0	0	1	0	+
								1	=
			0	1	0	0	1	1	

Signed Integer Numbers – 2's complement

The 2's complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -45 in 1 byte (8 bits):

INVERT:	0	0	1	0	1	1	0	1	
ADD 1:	1	1	0	1	0	0	1	0	+
								1	=
		1	0	1	0	0	1	1	

Signed Integer Numbers – 2's complement

The 2's complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -45 in 1 byte (8 bits):

INVERT:	0	0	1	0	1	1	0	1	
ADD 1:	1	1	0	1	0	0	1	0	+
								1	=
	1	1	0	1	0	0	1	1	

Signed Integer Numbers – 2's complement

The 2's complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -45 in 1 byte (8 bits):

INVERT:	0	0	1	0	1	1	0	1	
ADD 1:	1	1	0	1	0	0	1	0	+
								1	=
2's complement:	1	1	0	1	0	0	1	1	

Signed Integer Numbers – 2's complement

The 2's complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -45 in 1 byte (8 bits):

INVERT:	0	0	1	0	1	1	0	1	
ADD 1:	1	1	0	1	0	0	1	0	+
								1	=
2's complement:	1	1	0	1	0	0	1	1	

The most significant bit must be equal to 1

Signed Integer Numbers – 2's complement

The 2's complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -6 in 1 byte (8 bits):

0	0	0	0	0	1	1	0

Signed Integer Numbers – 2's complement

The 2's complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -6 in 1 byte (8 bits):

INVERT:

0	0	0	0	0	1	1	0

1	1	1	1	1	0	0	1

Signed Integer Numbers – 2's complement

The 2's complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -6 in 1 byte (8 bits):

INVERT:	0	0	0	0	0	1	1	0	
ADD 1:	1	1	1	1	1	0	0	1	+
								1	=

Signed Integer Numbers – 2's complement

The 2's complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -6 in 1 byte (8 bits):

INVERT:	0	0	0	0	0	1	1	0	
ADD 1:	1	1	1	1	1	0	0	1	+
						carry →	1		=
								0	

Signed Integer Numbers – 2's complement

The 2's complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -6 in 1 byte (8 bits):

INVERT:	0	0	0	0	0	1	1	0	
ADD 1:	1	1	1	1	1	0	0	1	+
									=
							1	0	

Signed Integer Numbers – 2’s complement

The 2’s complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -6 in 1 byte (8 bits):

INVERT:	0	0	0	0	0	1	1	0	
ADD 1:	1	1	1	1	1	0	0	1	+
									=
2’s complement:	1	1	1	1	1	0	1	0	

Signed Integer Numbers – 2’s complement

The 2’s complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **`1`** to the entire inverted number.

Example: representing the signed integer number -1 in 1 byte (8 bits):

`0`	`0`	`0`	`0`	`0`	`0`	`0`	`1`

Signed Integer Numbers – 2's complement

The 2's complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -1 in 1 byte (8 bits):

INVERT:	0	0	0	0	0	0	0	1
	1	1	1	1	1	1	1	0

Signed Integer Numbers – 2's complement

The 2's complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -1 in 1 byte (8 bits):

INVERT:	0	0	0	0	0	0	0	1	
ADD 1:	1	1	1	1	1	1	1	0	+
								1	=

Signed Integer Numbers – 2’s complement

The 2’s complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -1 in 1 byte (8 bits):

`INVERT:`	0	0	0	0	0	0	0	1	
`ADD 1:`	1	1	1	1	1	1	1	0	+
								1	=
`2’s complement:`	1	1	1	1	1	1	1	1	

Signed Integer Numbers – 2's complement

The 2's complement of a number is calculated as follows:

- invert all bits (i.e., bitwise not), then
- add a place value of **1** to the entire inverted number.

Example: representing the signed integer number -1 in 1 b

INVERT:	0	0	0	0	0	0	0		
ADD 1:	1	1	1	1	1	1	1	0	
								1	=
2's complement:	1	1	1	1	1	1	1	1	

2'S COMPLEMENT IS USED IN C AND C++ TO REPRESENT SIGNED INTEGER NUMBERS

Signed Integer Numbers – MIN and MAX

The minimum and maximum values you can represent depend on the number of bits you have at your disposal.

With 8 bits and 2's complement representation:

- the **minimum** representable unsigned integer number is:

1	0	0	0	0	0	0	1

- the **maximum** representable unsigned integer number is:

0	1	1	1	1	1	1	1

Which numbers are these?
Let's make the math

Signed Integer Numbers – MIN and MAX

Calculations:

With 8 bits:

- MIN = ???
- MAX = ???

Signed Integer Numbers – MIN and MAX

Calculations:

With 8 bits:

- MIN = ???
- MAX = ???

domain of a signed integer number with 8 bits

Signed Integer Numbers – MIN and MAX

Calculations:

With 8 bits:

- MIN = ???
- MAX = ???

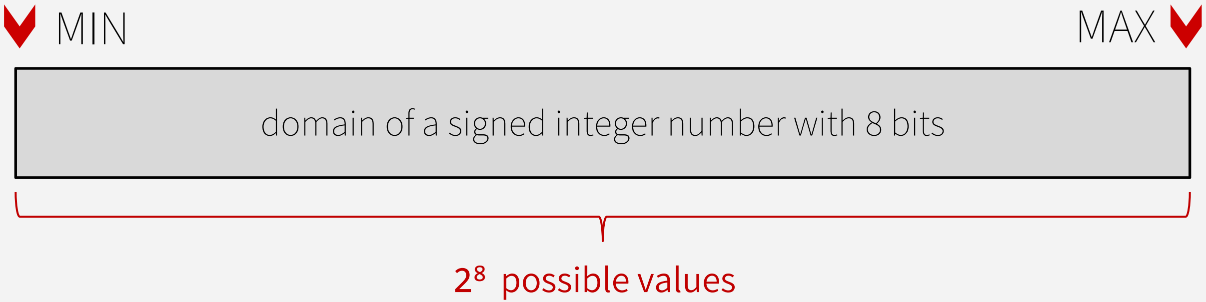
$$\underbrace{\boxed{\text{domain of a signed integer number with 8 bits}}}_{2^8\ \text{possible values}}$$

Signed Integer Numbers – MIN and MAX

<u>Calculations:</u>

With 8 bits:

- MIN = ???
- MAX = ???

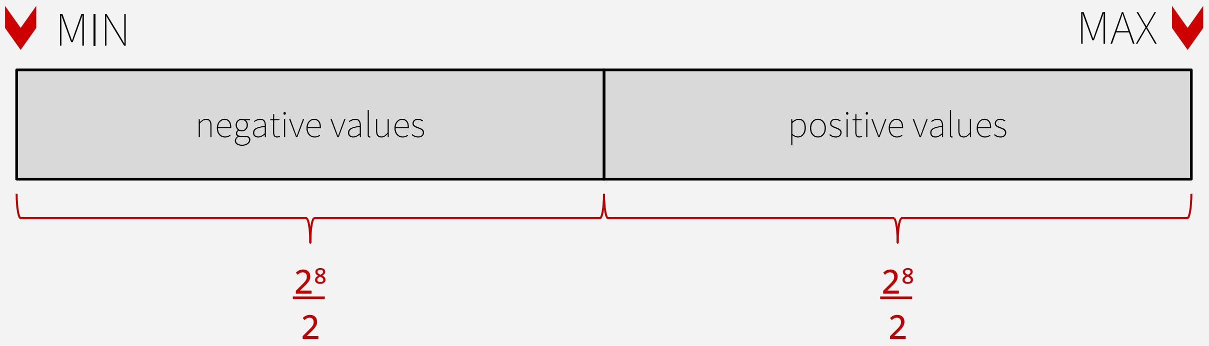


Signed Integer Numbers – MIN and MAX

Calculations:

With 8 bits:

- MIN = $-2^8/2$
- MAX = $+2^8/2$

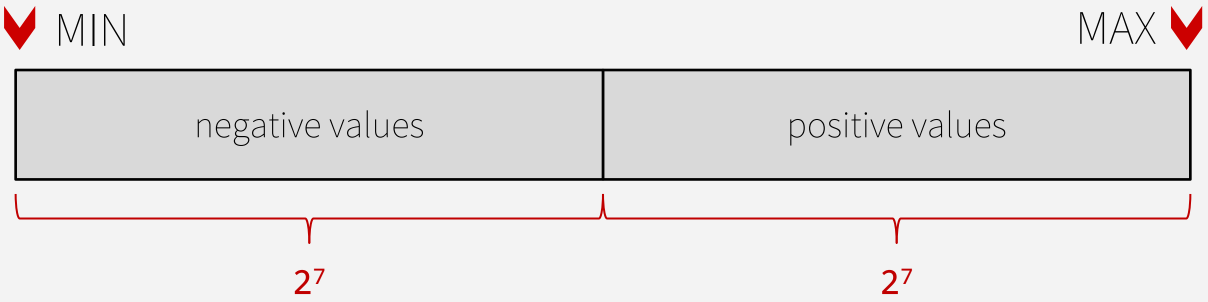


Signed Integer Numbers – MIN and MAX

Calculations:

With 8 bits:

- MIN = -2^7
- MAX = $+2^7$

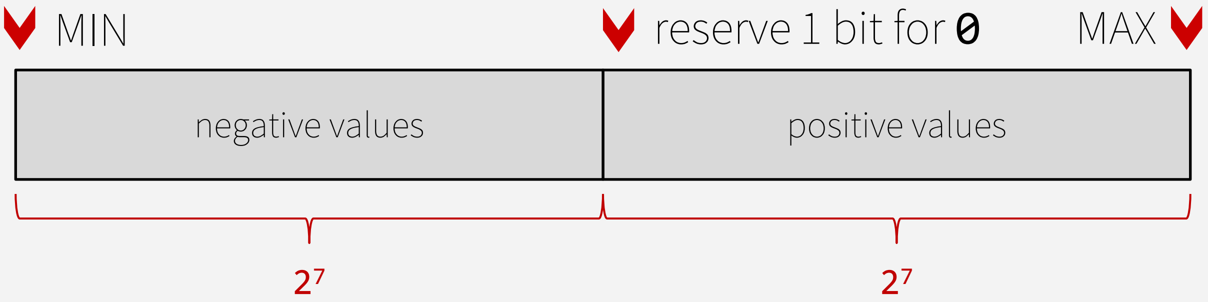


Signed Integer Numbers – MIN and MAX

Calculations:

With 8 bits:

- MIN = -2^7
- MAX = $+2^7-1$

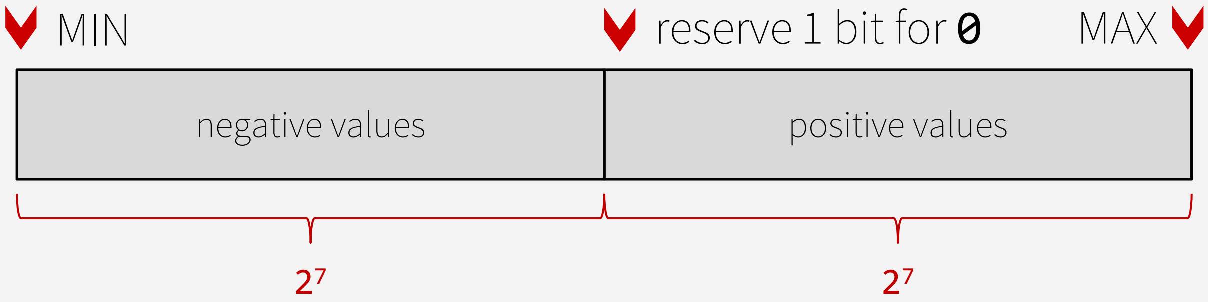


Signed Integer Numbers – MIN and MAX

Calculations:

With 8 bits:

- MIN = $-2^7 = -128$
- MAX = $+2^7 - 1 = 128 - 1 = 127$



Signed Integer Numbers – MIN and MAX

The minimum and maximum values you can represent depend on the number of bits you have at your disposal.

With 8 bits and 2's complement representation:

- the **minimum** representable unsigned integer number is:

1	0	0	0	0	0	0	1	⟶	$(-128)_{10}$

- the **maximum** representable unsigned integer number is:

0	1	1	1	1	1	1	1	⟶	$(+127)_{10}$

Floating-Point Numbers

A **floating-point number** is a is a fractional/real number $\in \mathbb{R}$. It has a sign (i.e., it can be positive, negative, or zero).

They have their own protocol for representation (i.e., a set of specific rules that allow for codification and de-codification of the information) .

Floating-Point Numbers

A **floating-point number** is a is a fractional/real number $\in \mathbb{R}$. It has a sign (i.e., it can be positive, negative, or zero).

For example, the *IEEE 754 double-precision binary floating-point format*, which represents a real number with 64 bits:

Example:

64 bits

Floating-Point Numbers

A **floating-point number** is a is a fractional/real number $\in \mathbb{R}$. It has a sign (i.e., it can be positive, negative, or zero).

For example, the *IEEE 754 double-precision binary floating-point format*, which represents a real number with 64 bits:

- `1 bit` for sign → $2^1 = 2$ possibilities for the sign
-
-

sign

$\underbrace{\hspace{6em}}_{\texttt{64 bits}}$

Example:

–

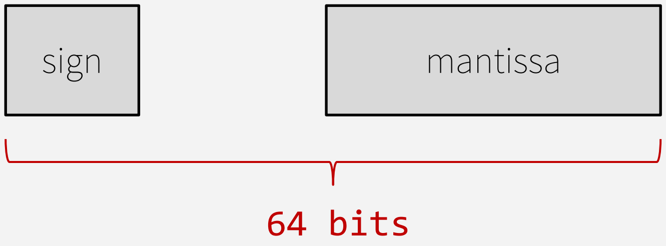
Sign

Floating-Point Numbers

A **floating-point number** is a is a fractional/real number $\in \mathbb{R}$. It has a sign (i.e., it can be positive, negative, or zero).

For example, the *IEEE 754 double-precision binary floating-point format*, which represents a real number with 64 bits:

- `1 bit` for sign → 2^1 `= 2` possibilities for the sign
-
- `52 bits` for mantissa → 2^{52} `= 4.5036E+15` possibilities for mantissa



Example:

$$\underset{\text{Sign}}{-}\ 0.\underbrace{3213242}_{\text{Mantissa}}$$

Floating-Point Numbers

A **floating-point number** is a is a fractional/real number $\in \mathbb{R}$. It has a sign (i.e., it can be positive, negative, or zero).

For example, the *IEEE 754 double-precision binary floating-point format*, which represents a real number with 64 bits:

- `1 bit` for sign → `2`^{1} `= 2` possibilities for the sign
- `11 bits` for exponent → `2`^{11} `= 2048` possibilities for exponent
- `52 bits` for mantissa → `2`^{52} `= 4.5036E+15` possibilities for mantissa

sign	exponent	mantissa

`64 bits`

Example:

$$\underbrace{-}_{\text{Sign}}\ 0.\underbrace{3213242}_{\text{Mantissa}} \cdot 10^{\underbrace{345}_{\text{Exponent}}}$$

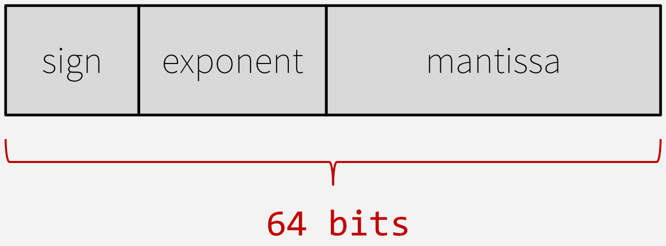
Floating-Point Numbers

A **floating-point number** is a is a fractional/real number $\in \mathbb{R}$. It has a sign (i.e., it can be positive, negative, or zero).

For example, the *IEEE 754 double-precision binary floating-point format*, which represents a real number with 64 bits:

- `1 bit` for sign → $2^1 = 2$ possibilities for the sign
- `11 bits` for exponent → $2^{11} = 2048$ possibilities for exponent
- `52 bits` for mantissa → $2^{52} = 4.5036E+15$ possibilities for mantissa

If you make the math, this is $2^1 \cdot 2^{52} \cdot (2^{11} - 2) = 1.84287E+19$ discrete values!



Example:

$$\underset{\text{Sign}}{-}\ 0.\underbrace{3213242}_{\text{Mantissa}} \cdot 10^{\underbrace{345}_{\text{Exponent}}}$$

Floating-Point Numbers – MIN and MAX

The minimum and maximum values you can represent depend on the number of bits you have at your disposal.

<u>EXERCISE FOR HOME:</u>

Calculate the minimum and maximum representable numbers with the *IEEE 754 double-precision binary floating-point* format.