Number Representation

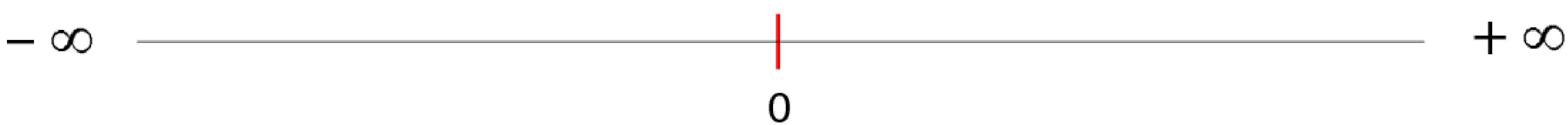
- Integer representation
  - Unsigned Integers
  - Signed integers
    - Signed-Magnitude
    - 2's complement
- Floating point representation

INTEGER REPRESENTATION

Integer Representation

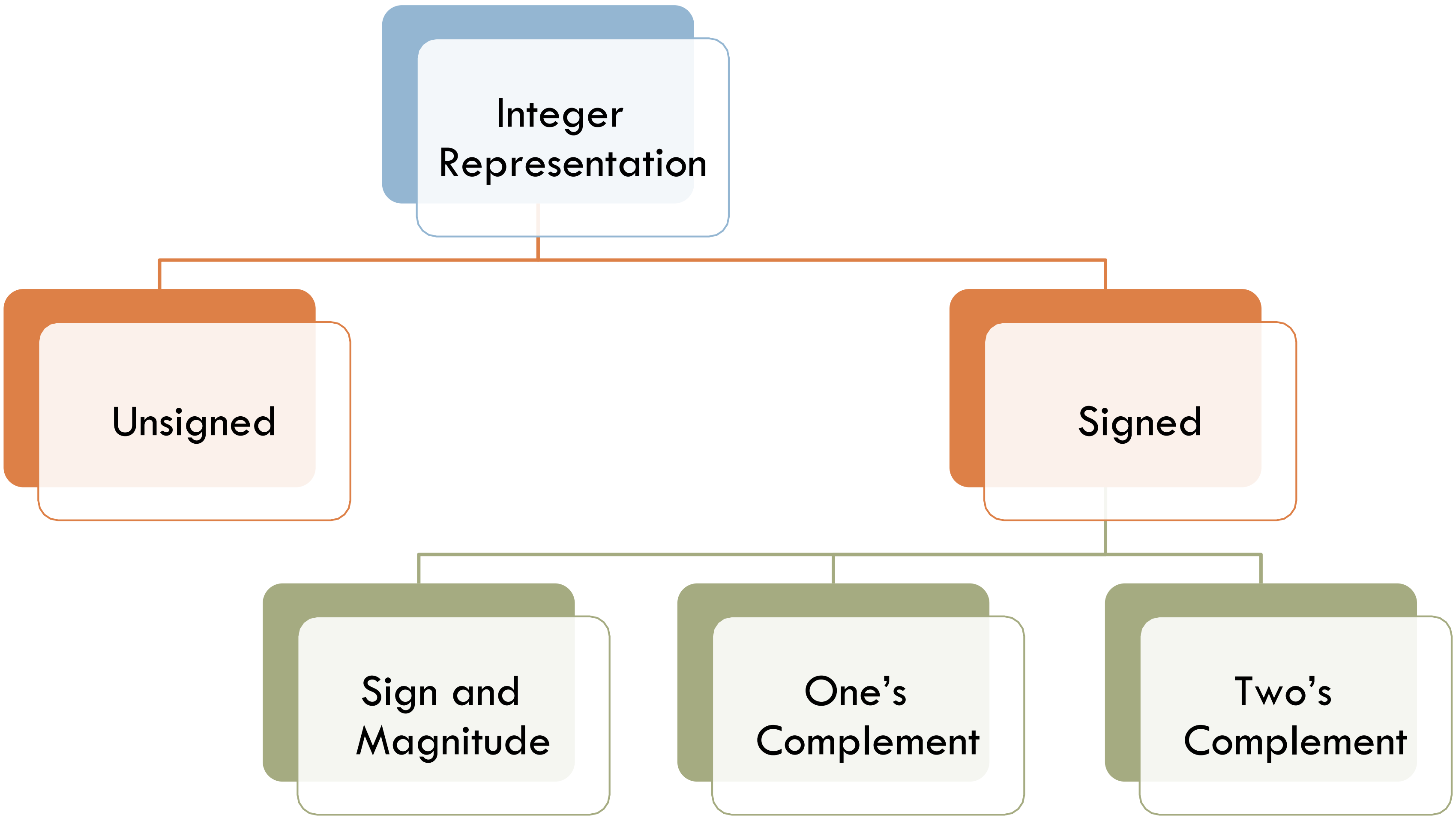
Integer Number: is a whole number without fractions, it can be positive or negative

Integers range between negative infinity ($-\infty$) and positive infinity ($+\infty$)



But can a computer store all the integers in between?

Integer Representation



Unsigned Integer

- **Unsigned Integer: is an integer without a sign and ranges between 0 and + ∞**
- The maximum unsigned integer depends on the number of bits ***(N)*** allocated to represent the unsigned integer in a computer

Range: 0 → $(2^N - 1)$

No. of bits	Range
8	0 to 255
16	0 to 65535
32	0 to ?

Unsigned Integer

While storing unsigned integer, If the number of bits is *less than* *N*, 0s are added to the *left* of the binary number so that there is a total of *N* bits.

Example 1

Store 7 in an 8-bit memory location using unsigned representation.

Solution

1. First change the integer to binary, $(111)_2$.
2. Add five 0s to make a total of ***N*** (8) bits, $(00000111)_2$.
3. The integer is stored in the memory location.

Change 7 to binary	→							1	1	1
Add five bits at the left	→		0	0	0	0	0	1	1	1

Example 2

Store 258 in an 16-bit memory location using unsigned representation.

Solution

1. First change the integer to binary $(100000010)_2$.
2. Add seven 0s to make a total of ***N*** (16) bits, $(0000000100000010)_2$.
3. The integer is stored in the memory location.

Change 258 to binary	→								1	0	0	0	0	0	0	1	0
Add seven bits at the left	→	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0

What will happen if you Try to store an unsigned integer Such as 256 in an 8-bit memory Location?

Overflow

occurs if the decimal is out of range (if binary bits > n)

Unsigned Integer

- **Example of storing unsigned integers in two different computers**

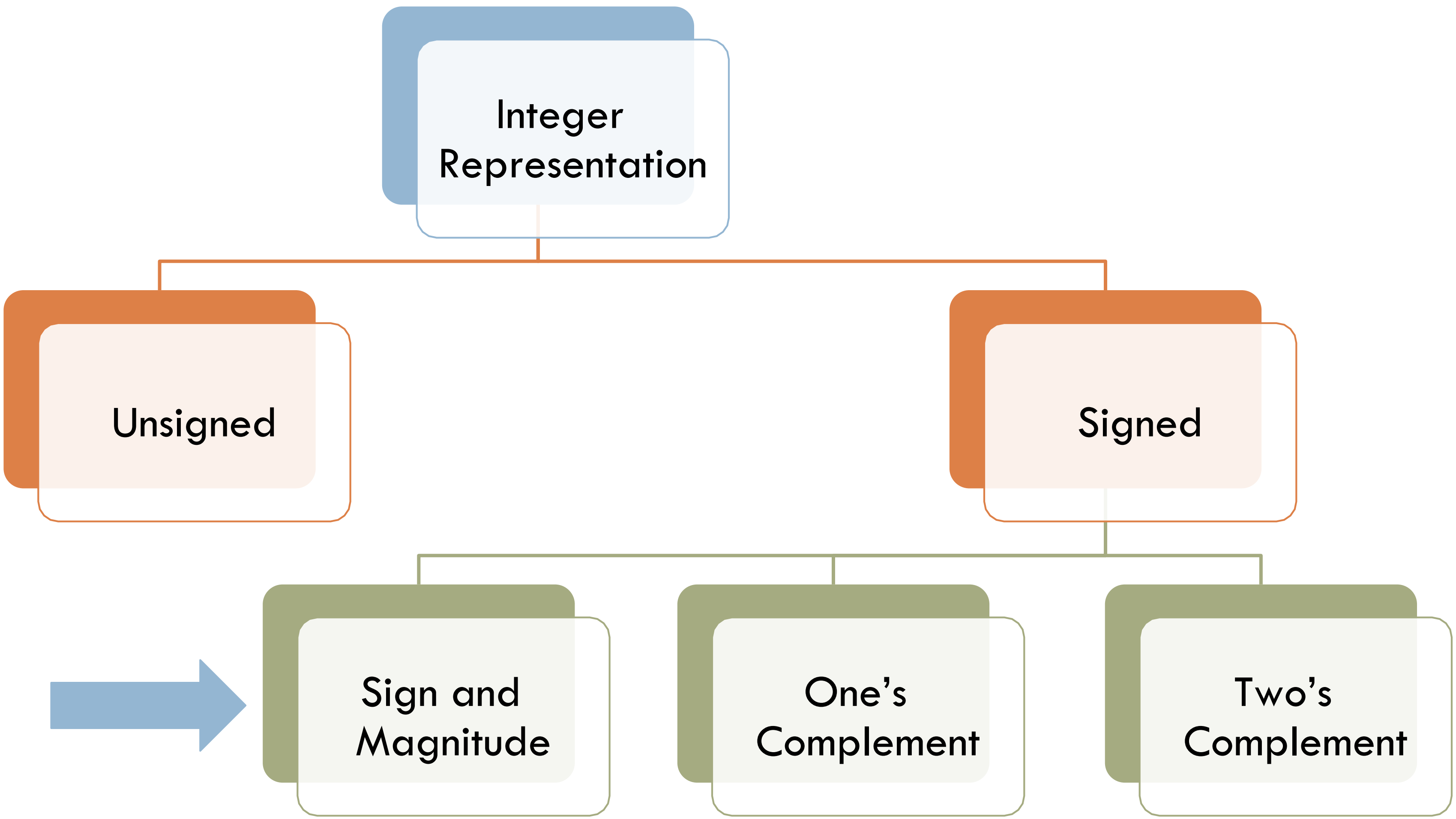
Decimal	*8-bit allocation*	*16-bit allocation*
7	00000111	0000000000000111
234	11101010	0000000011101010
258	overflow	0000000100000010
24,760	overflow	0110000010111000
1,245,678	overflow	overflow

Unsigned Integer Applications

- Counting : you don't need negative numbers to count and usually start from 0 or 1 going up

- Addressing: sometimes computers store the address of a memory location inside another memory location, addresses are positive numbers starting from 0

Integer Representation



SIGNED NUMBER REPRESENTATION

SIGNED MAGNITUDE REPRESENTATION

Sign-and-magnitude representation

- Signed Integer is an integer with a sign either + or -
- Storing an integer in a sign-and-magnitude format requires 1(the leftmost) bit to represent the sign (0 for positive and 1 for negative) and rest of the bits to represent madnitude

Range : $-(2^{N-1} -1) \ldots +(2^{N-1} -1)$

Sign-and-magnitude representation

- Range of Sign and Magnitude representation

No. of bits	Range
8	-127 ………………….-0 +0 ……….. +127
16	-32767 ……………-0 +0 ………..+32767
32	–2,147,483,647 ……-0+0………..+2147483647

There are two 0s in sign-and-magnitude epresentation:

positive and negative.

Sign-and-magnitude representation

- Storing sign-and-magnitude signed integer process:

1. The integer is changed to binary, (the sign is ignored).

2. If the number of bits is *less* than ***N-1***, 0s are added to the left of the number so that there will be a total of ***N-1*** bits .

3. If the number is ***positive***, 0 is added to the left (to make it ***N*** bits). But if the number is ***negative***, ***1*** is added to the left (to make it ***N*** bits)

Example 4

- Store +7 in an 8-bit memory location using sign-and-magnitude representation.

Solution

- The integer is changed to binary (111).
- Add 4 0s to make a total of ***N-1*** (7) bits, 0000111 .
- Add an extra **0** (in bold) to represent the ***positive*** sign

0 0000111

Example 5

- Store –258 in a 16-bit memory location using sign-and-magnitude representation

Solution

- First change the number to binary 100000010
- Add six 0s to make a total of ***N-1*** (15) bits, 000000100000010
  - Add an extra ***1*** because the number is ***negative***.

1 000000100000010

Signed-Magnitude Representation - Example

Decimal Number	Signed Magnitude representation in 8 bits	Signed Magnitude Representation in 16 bits
-7	10000111	1000000000000111
-124	11111100	1000000001111100
+124	01111100	0000000001111100
+258	Overflow	0000000100000010
-24760	overflow	1110000010111000

Sign-and-magnitude Interpretation

Q: How do you interpret a signed binary representation in decimal?

1. Ignore the first (leftmost) bit for a moment
2. Change the remaining ***N -1*** bits from binary to decimal
3. Attach a + or – sign to the number based on the leftmost bit.

Example 6

- Interpret 10111011 to decimal if the number was stored as a sign-and-magnitude integer.

Solution

- Ignoring the leftmost bit for a moment, the remaining bit are 0111011.
- This number in decimal is 59.
- the leftmost bit is 1 so the number is – 59.

Signed Magnitude representation Applications

The sign-and-magnitude representation is ***not used*** now by computers because:

- **Operations:** such as subtraction and addition is not straightforward for this representation.

- **Uncomfortable in programming:** because there are two 0s in this representation

Signed Magnitude Representation Applications

However..

The ***advantage*** of this representation is:

- ***Transformation:*** from decimal to binary and vice versa which makes it convenient for applications that don't need operations on numbers

- Ex: Converting Audio (analog signals) to digital signals.

2'S COMPLEMENT REPRESENTATION

Complement of a number

- **(R-1)'s complement**
- **R's complement = [(R-1's complement) + 1]**
  - Where is called radix (or base)

R = 10	**(R-1)'s complement** **9's complement**	**R's complement** **(10's complement)**
473	999-473 = 526	526 + 1= 527
8437	9999 – 8437 = 1562	1562 + 1 = 1563

R = 2	**(R-1)'s complement** **1's complement**	**R's complement** **(2's complement)**
1011	1111-1011 = 0100	0100 + 1= 0101
0011101	1111111 – 0011101 = 1100010	1100010 + 1 = 1100011

Complement of a number

- **Exercise 7**
  - Write down the 1's complement and 2's complement of following binary numbers in 8 bits
    a) 11001
    b) 10001101
  - Write down the 1's complement and 2's complement of following binary numbers in 16 bits
    c) 11001
    d) 000000110101

1's complement representation

- The most significant bit (msb) is the *sign bit*, with value of 0 representing positive integers and 1 representing negative integers.
- The remaining *n*-1 bits represents the magnitude of the integer, as follows:
  - for positive integers, the absolute value of the integer is equal to "the magnitude of the (*n*-1)-bit binary pattern".
  - for negative integers, the absolute value of the integer is equal to "the magnitude of the *complement* (*inverse*) of the (*n* -1)-bit binary pattern" (hence called 1's complement).

1's complement representation

- **Example 1**: Suppose that n=8 and the binary representation 0 100 0001.
  Sign bit is 0 ⇒ positive
  Absolute value is 100 0001 = 65
  Hence, the integer is +65
- **Example 2**: Suppose that n=8 and the binary representation 1 000 0001.
  Sign bit is 1 ⇒ negative
  Absolute value is the complement of 000 0001, i.e., 111 1110 = 126
  Hence, the integer is -126

1's complement representation

- **Example 3**: Suppose that n=8 and the binary representation 0 000 0000.
  Sign bit is 0 ⇒ positive
  Absolute value is 000 0000 = 0
  Hence, the integer is **+0**
- **Example 4**: Suppose that n=8 and the binary representation 1 111 1111.
  Sign bit is 1 ⇒ negative
  Absolute value is the complement of 111 1111, i.e., 000 0000 = 0
  Hence, the integer is **-0**

1's complement representation

- **Drawbacks** of 1's complement representation for signed numbers :
  - There are two representations (0000 0000 and 1111 1111) for zero.
  - The positive integers and negative integers need to be processed separately.

- **Because of the above drawbacks 1's complement is not the preferred choice for representing signed numbers**

2’s complement representation

- Most significant bit (msb) is the *sign bit*, with value of 0 representing positive integers and 1 representing negative integers.
- The remaining *n*-1 bits represents the magnitude of the integer, as follows:
  - for positive integers, the absolute value of the integer is equal to "the magnitude of the (*n*-1)-bit binary pattern".
  - for negative integers, the absolute value of the integer is equal to "the magnitude of the *complement* of the (*n*-1)-bit binary pattern *plus one*" (hence called 2's complement).

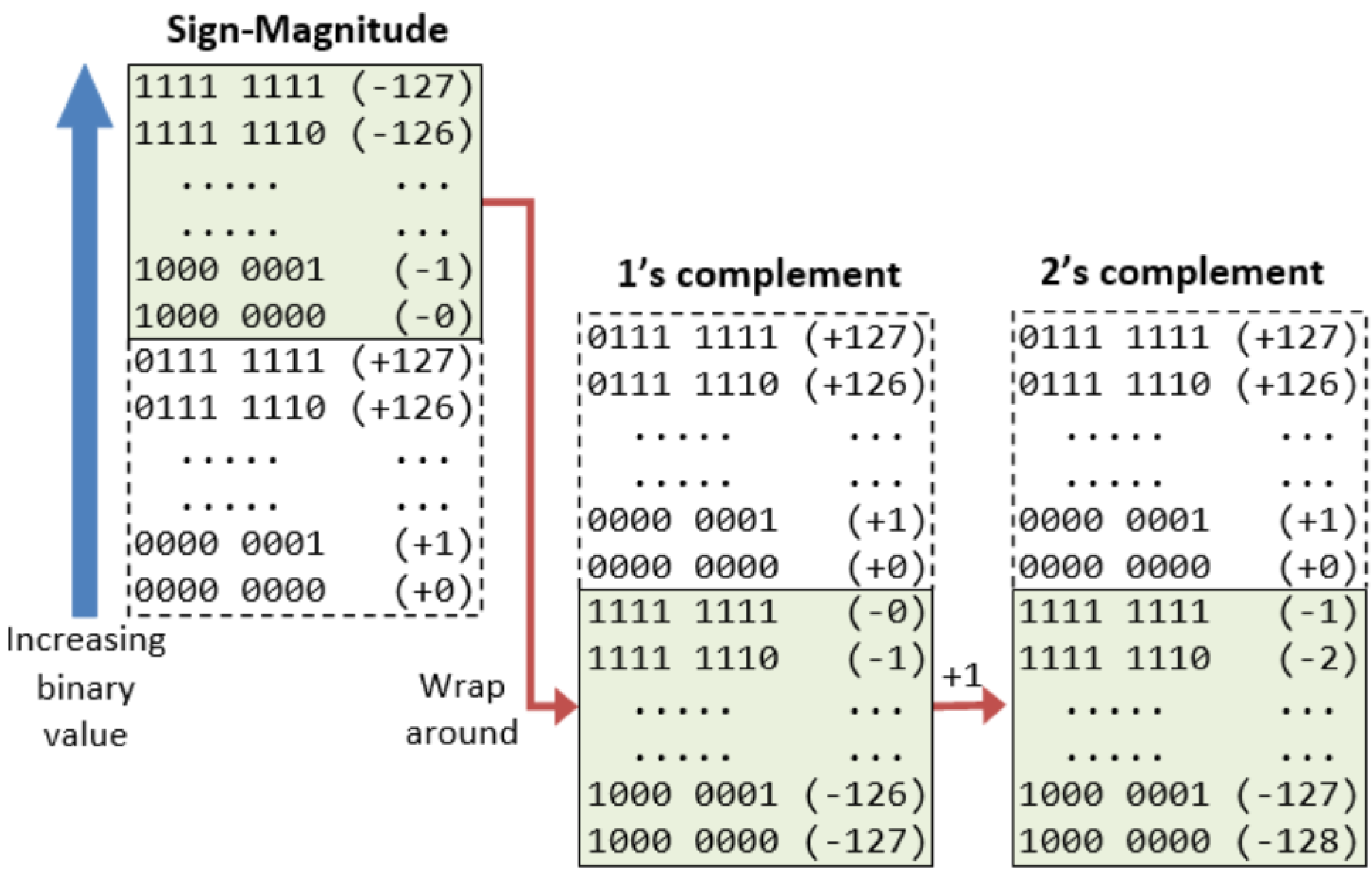
2's complement representation

- **Example 1**: Suppose that n=8 and the binary representation 0 100 0001.
  Sign bit is 0 $\Rightarrow$ positive
  Absolute value is 100 0001 = 65
  Hence, the integer is +65
- **Example 2**: Suppose that n=8 and the binary representation 1 000 0001.
  Sign bit is 1 $\Rightarrow$ negative
  Absolute value is the complement of 000 0001 plus 1, i.e., 111 1110 + 1 = 127
  Hence, the integer is -127

2's complement representation

- **Example 3**: Suppose that n=8 and the binary representation 0 000 0000B.
  Sign bit is 0 $\Rightarrow$ positive
  Absolute value is 000 0000B = 0D
  Hence, the integer is +0D
- **Example 4**: Suppose that n=8 and the binary representation 1 111 1111B.
  Sign bit is 1 $\Rightarrow$ negative
  Absolute value is the complement of 111 1111B plus 1, i.e., 000 0000B + 1B = 1D
  Hence, the integer is -1D

Signed Integer Representation



For 8 bits

Range

- An n-bit 2's complement signed integer can represent integers from

Range : $-(2^{n-1})$ to $+(2^{n-1} - 1)$

n	minimum	maximum
8	-(2^7) (=-128)	+(2^7)-1 (=+127)
16	-(2^15) (=-32,768)	+(2^15)-1 (=+32,767)
32	-(2^31) (=-2,147,483,648)	+(2^31)-1 (=+2,147,483,647)(9+ digits)
64	-(2^63) (=-9,223,372,036,854,775,808)	+(2^63)-1 (=+9,223,372,036,854,775,807)(18+ digits) + digits)

2's complement representation

- There is only one representation of 0 which makes 2's complement representation a preferred choice for representing signed numbers

- Computers also use 2's complement representation for representing signed numbers

2's complement representation

- **Exercise 8**
  - Write down the following numbers in binary using 2's complement representation for signed numbers in 8 bits
    - -58
    - +58
    - -102
  - Figure out the decimal numbers (including sign) from the following binary numbers represented using 2's complement.
    - 00100010
    - 10111001
    - 11000110

FLOATING POINT REPRESENTATION

Floating Point Numbers

- A floating-point number is typically expressed in the scientific notation, with a *fraction* (F), and an *exponent* (E) of a certain *radix* (r), in the form of **F×r^E.**
- Decimal numbers use radix of 10 → (**F×10^E**)

$$
\begin{aligned}
547.32 &= 547.32 \times 10^0 \\
&= 54.732 \times 10^1 \\
&= 5.4732 \times 10^2 \\
&= 0.54732 \times 10^3
\end{aligned}
$$

- Binary numbers use radix of 2 → (**F×2^E**)

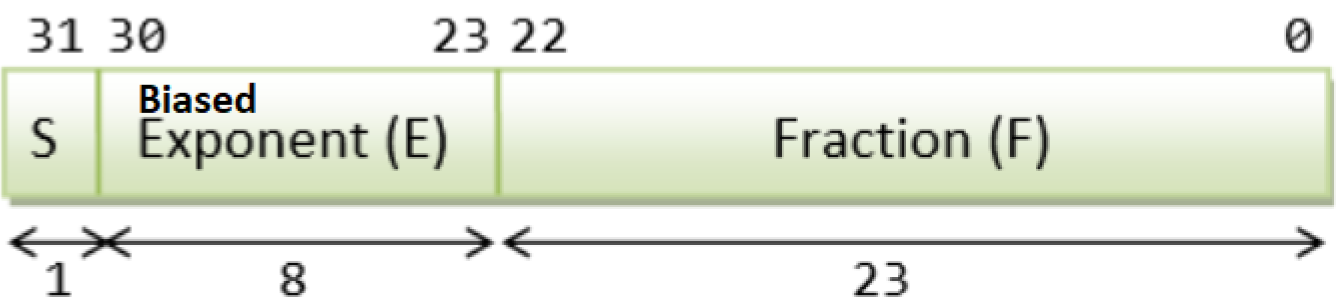
$$
\begin{aligned}
0110.101 &= 0110.101 \times 2^0 \\
&= 011.0101 \times 2^1 \\
&= 01.10101 \times 2^2 \\
&= 0.110101 \times 2^3
\end{aligned}
$$

Floating Point Representation

- In computers, floating-point numbers are represented in scientific notation of *fraction* (F) and *exponent* (E) with a *radix* of 2, in the form of F×2^E.
- Both E and F can be positive as well as negative.
- Modern computers adopt IEEE 754 standard for representing floating-point numbers.
- There are two representation schemes: 32-bit single -precision and 64-bit double-precision.

IEEE-754 32-bit Single-Precision Floating-Point Number Representation

- In 32-bit single-precision floating-point representation:
  - The most significant bit is the *sign bit* (S), with 0 for positive numbers and 1 for negative numbers.
  - The following 8 bits represent **biased *exponent*** **(E).**
  - The remaining 23 bits represents ***fraction*** **(F).**



32-bit Single-Precision Floating-point Number

IEEE-754 32-bit Single-Precision Floating-Point Number Representation

- Example 1
  - Represent -13.8 using IEEE 754 32 bit single precision floating point representation

1. (13.8) → (1101.11001)
2. 1101.1100 = 1.10111001x 2^3
3. Actual exponent = 3
4. Biased exponent = 3 + 127 = 130 = (10000010)
5. Sign of Fraction/Mantissa (s) = -ve = 1

Bias of 127 is to be added to the actual exponent so that sign of exponent is taken care of

S	Biased Exponent	Matissa/Fraction
1	10000010	10111001000000000000000

IEEE-754 32-bit Single-Precision Floating-Point Number Representation

- **Example 2**
- Let's illustrate with an example, suppose that the 32-bit pattern is

1 1000 0001 011 0000 0000 0000 0000 0000

S = 1

Biased Exponent = 1000 0001 (Actual Exponent = 10000001 – 127 =2)

F = 011 0000 0000 0000 0000 0000

In the *normalized form*, the actual fraction is normalized with an implicit leading 1 in the form of 1.F

In this example, the actual fraction is

1.011 0000 0000 0000 0000 0000 = 1 + 1×2^-2 + 1×2^-3 = **1.375**

The sign bit represents the sign of the number, with S=0 for positive and S=1 for negative number.

In this example with S=1, this is a negative number, i.e., **-1.375**

IEEE-754 32-bit Single-Precision Floating-Point Number Representation

- The actual exponent is (biased exponent -127). This is because we need to represent both positive and negative exponent.
- With an 8-bit for exponent, ranging from 0 to 255, the bias(127) scheme could provide actual exponent of -127 to 128.
- In this example, actual exponent is $=129-127=2$
- Hence, the number represented is $-1.375\times2^2=-5.5$

IEEE-754 32-bit Single-Precision Floating-Point Number Representation

- Example 2
  - Figure out the floating point number

  110000010 10111000000000000000000 which is represented by IEEE 754 -32 bit

- Solution

1	10000010	10111001000000000000000

  - S= 1 (number is –ve)
  - Biased Exponent = 10000010 = 130
  - Actual Exponent = Biased Exponent -127 = 130 -127 = 3
  - Fraction = 1.10111001000000000000000

  $= 1 + (1x\ 2^{-1}) + (0x\ 2^{-2}) + (1x2^{-3}) + (1x\ 2^{-4}) + (1x\ 2^{-5}) + 0 + 0 + (1x2^{-6}) = 1.734375$

  - $1.734375 \times 2^{3} = 13.875$

IEEE-754 32-bit Single-Precision Floating-Point Number Representation

- **Exercise 9**
  - **Represent -102.27 using IEEE 754 32 bit single precision floating point representation**

- **Exercise 10**
- **Figure out the floating point number which has been represented by IEEE 754 -32 bit**

a) 0 10000000 110 0000 0000 0000 0000 0000

b) 1 01111110 100 0000 0000 0000 0000 0000.

Thank you

Reference

- https://www.ntu.edu.sg/home/ehchua/programming/java/datarepresentation.html