

Binary representations of integers

- dimensions of the memory location: $n = 8, 16, 32, 64$ bits
- the structure of the memory location : bit 0 (the rightmost bit) is the least significant bit, bit $n-1$ (the leftmost) is the most significant one ;

bit $n-1$	bit $n-2$		bit 1	bit 0
		...		

Rules for binary operations with integers:

R1) Addition and subtraction on n bits: both terms and the result are represented on n bits.

R2) Multiplication on n bits: both factors represented on n bits and the product represented on $2 \cdot n$ bits.

R3) Division on n bits: dividend represented on $2 \cdot n$ bits, the divisor on n bits. This operation provides a quotient and a remainder represented both on n bits.

R4) In case of overflow the most significant bits are lost.

There are two classes of representations:

- *unsigned representation*: only for natural numbers;
- *signed representations*: for integers with sign.

Unsigned representation

- the number is converted into binary and then the bits are filled with the corresponding binary digits (aligned to the right), eventually the leftmost bits are filled with insignificant zeros.
- min: $0 \dots 0_{(2)} = 0$
- max: $1 \dots 1_{(2)} = 2^n - 1$
- intervals of values:
 $n = 8$ $[0, 255]$
 $n = 16$ $[0, 65535]$

$n = 32$ $[0, 4\,294\,967\,295]$
 $n = 64$ $[0, 18\,446\,824\,753\,389\,551\,615]$

Examples of representations and operations on 8 bits:

- 1)

18 ⁺	12 ₍₁₆₎ ⁺	00010010 ₍₂₎ ⁺
18	12 ₍₁₆₎	00010010 ₍₂₎
---	----	-----
36	24 ₍₁₆₎	00100100 ₍₂₎

- 2)

243 ⁻	F3 ₍₁₆₎ ⁻	11110011 ₍₂₎ ⁻
18	12 ₍₁₆₎	00010010 ₍₂₎
----	----	-----
225	E1 ₍₁₆₎	11100001 ₍₂₎

- 3)

243 ⁺	F3 ₍₁₆₎ ⁺	11110011 ₍₂₎ ⁺
18	12 ₍₁₆₎	00010010 ₍₂₎
---	--	-----
261	05 ₍₁₆₎	100000101 ₍₂₎ ! overflow at addition!

- 4)

18 ⁻	12 ₍₁₆₎ ⁻	00010010 ₍₂₎ ⁻
243	F3 ₍₁₆₎	11110011 ₍₂₎
---	--	-----
-225	1F ₍₁₆₎	00011111 ₍₂₎ ! overflow at subtraction !

Signed representations – codes

- dimensions of the memory location $n \in \{8, 16, 32, 64\}$ bits
- the most significant bit (bit $n-1$) is used for the sign ($S=0$ for positive numbers, $S=1$ for negative numbers), and for the magnitude of the number $n-1$ bits are used.

bit $n-1$	bit $n-2$		bit 1	bit 0
S		...		

Direct code

Let x be an integer, $|x| < 2^{n-1}$. The *direct code* of x is defined as follows:

$$[x]_{\text{dir}} = \begin{cases} x_{(2)} & , \text{if } x \geq 0 \\ 2^{n-1} + |x| & , \text{if } x < 0 \end{cases}$$

- for a positive number the bits from the memory location are filled with the digits from the binary representation, aligned to the right, the sign bit is 0;
- for negative number: its absolute value is represented and then the sign bit is set to 1 (equivalent with the operation: $10...0_{(2)} + |x|$)
- disadvantage: there are 2 different direct codes for 0:
 $[+0]_{\text{dir}}: |0|0...0|$ and $[-0]_{\text{dir}}: |1|0...0|$

Inverse code (complement to 1)

Let x be an integer, $|x| < 2^{n-1}$. The *inverse code* of x is defined as follows:

$$[x]_{\text{inv}} = \begin{cases} x_{(2)} & , \text{if } x \geq 0 \\ 2^n - 1 - |x| & , \text{if } x \leq 0 \end{cases}$$

- the inverse code of a positive number is the same as the direct code
- for a negative number: its absolute value is represented and then the values of all bits are inversed (complemented to 1): equivalent with the operation: $11...1_{(2)} - |x_{(2)}|$;
- disadvantage: there are 2 different inverse codes for 0:
 $[+0]_{\text{inv}}: |0|0...0|$ and $[-0]_{\text{inv}}: |1|1...1|$

Complementary code (complement to 2)

Let x be an integer, $|x| < 2^{n-1}$. the *complementary code* of x is defined as follows:

$$[x]_{\text{compl}} = \begin{cases} x_{(2)} & , \text{if } x \geq 0 \\ 2^n - |x_{(2)}| & , \text{if } x < 0 \end{cases}$$

- for a positive number x , $[x]_{\text{compl}} = [x]_{\text{inv}} = [x]_{\text{dir}}$
- for a negative number: its absolute value is represented, then from right to left, beginning with bit 0, all the bits with value 0 and the first bit 1 remain unchanged and all the other bits to the left are inversed.
- if $x < 0$, then $[x]_{\text{compl}} = [x]_{\text{inv}} + 1$
- advantage: there is a unique code for 0: $[0]_{\text{compl}}: |0|0...0|$
- there is a configuration of bits which does not correspond to a number: $|1|0...0|$

Intervals of values for integers:

n= 8	[-127 , 127]
n=16	[-32767 , 32767]
n=32	[-2 147 483 647 , 2 147 483 647]
n=64	[-9 223 412 376 694 775 807, +9 223 412 376 694 775 807]

Examples: the codes on 8 bits:

$$100 = 1100100_{(2)}$$

$$[100]_{\text{dir}} = [100]_{\text{inv}} = [100]_{\text{compl}} = |0|1100100|$$

$$[-100]_{\text{dir}} = |1|1100100|$$

$$[-100]_{\text{inv}} = |1|0011011|$$

$$[-100]_{\text{compl}} = |1|0011100|$$

$$40 = 101000_{(2)}$$

$$[40]_{\text{dir}} = [40]_{\text{inv}} = [40]_{\text{compl}} = |0|0101000|$$

$$[-40]_{\text{dir}} = |1|0101000|$$

$$[-40]_{\text{inv}} = |1|1010111|$$

$$[-40]_{\text{compl}} = |1|1011000|$$

Addition and subtraction in complementary code

$[x]_{\text{compl}}$ is considered an array of n bits, representing a positive number in base 2.

The complementary codes of the sum and difference of the integers x, $y \geq 0$:

$$[x+y]_{\text{compl}} = [x]_{\text{compl}} \oplus [y]_{\text{compl}}$$

$$[x-y]_{\text{compl}} = [x]_{\text{compl}} \oplus [-y]_{\text{compl}}$$

The algebraic sum: \oplus , is defined as follows:

$$\forall x, y \in [0, 2^n), \quad x \oplus y = \begin{cases} x + y, & \text{if } x + y < 2^n \\ x + y - 2^n, & \text{if } x + y \geq 2^n \end{cases}$$

Rules

r1: if x, y have the same sign, but $x \oplus y$ has the opposite sign \Rightarrow overflow.

r2: if in the result of $x \oplus y$, there is a carry digit outside the representation space this will be eliminated (the second branch of the above definition).

Example:

$$\begin{array}{rcl}
 [40+40]_{\text{compl}} & = & [40]_{\text{compl}} \oplus 0|0101000 \oplus \\
 & & [40]_{\text{compl}} \quad \quad \quad 0|0101000 \\
 \hline
 [80]_{\text{compl}} & = & 0|1010000 \text{ - correct result}
 \end{array}$$

$$\begin{array}{rcl}
 [100+100]_{\text{compl}} & = & [100]_{\text{compl}} \oplus 0|1100100 \oplus \\
 & & [100]_{\text{compl}} \quad \quad \quad 0|1100100 \\
 \hline
 [200]_{\text{compl}} & = & 1|1001000 \text{ - overflow (r1)}
 \end{array}$$

$$\begin{array}{rcl}
 [100-40]_{\text{compl}} & = & [100]_{\text{compl}} \oplus 0|1100100 \oplus \\
 & & [-40]_{\text{compl}} \quad \quad \quad 1|1011000 \\
 \hline
 [60]_{\text{compl}} & = & \textcolor{red}{1}0|0111100 \text{ - correct result (r2)}
 \end{array}$$

Subunitary convention:

bit n-1 , bit n-2			bit 1	bit 0
S		...		

Let $x \in \mathbb{R}$, $|x| < 1$, with at most **n-1** binary digits after the decimal point.

The *direct code* of x is:

$$[x]_{\text{dir}} = \begin{cases} x_{(2)} & , \text{if } x \geq 0 \\ 1 + |x_{(2)}| & , \text{if } x \leq 0 \end{cases}$$

The *inverse code* (complement to 1) of x is:

$$[x]_{\text{inv}} = \begin{cases} x_{(2)} & , \text{if } x \geq 0 \\ 2 - 2^{-n+1} - |x_{(2)}| & , \text{if } x \leq 0 \end{cases}$$

The *complementary code* (complement to 2) of x is:

$$[x]_{\text{compl}} = \begin{cases} x(2) & , \text{if } x \geq 0 \\ 2 - |x| & , \text{if } x < 0 \end{cases}$$

The complementary codes for the sum and difference of two subunitary positive numbers x, y:

$$[x+y]_{\text{compl}} = [x]_{\text{compl}} \oplus [y]_{\text{compl}}$$

$$[x-y]_{\text{compl}} = [x]_{\text{compl}} \oplus [-y]_{\text{compl}}$$

The algebraic sum \oplus on n bits, is defined as follows:

$$\forall x, y \in [0,1), \quad x \oplus y = \begin{cases} x + y & , \text{if } x + y < 2 \\ x + y - 2 & , \text{if } x + y \geq 2 \end{cases}$$

Remarks:

- **for a positive number: the number is converted into binary, the sign bit is 0, beginning with the bit n-2, from left to the right, the bits are filled with the corresponding binary digits.**
- **the same practical rules for obtaining the codes for negative numbers and for the addition and subtraction in complementary code as for integers are applied.**
- **if $x \geq 0$, then $[x]_{\text{compl}} = [x]_{\text{inv}} = [x]_{\text{dir}}$**
- **if $x < 0$, then $[x]_{\text{compl}} = [x]_{\text{inv}} + 2^{-n+1}$**

Examples:

$$0,25 = 0,01_{(2)}$$

$$13/16 = 0,8125 = 0,1101_{(2)}$$

$$[13/16]_{\text{dir}} = [13/16]_{\text{inv}} = [13/16]_{\text{compl}} = |0|1101000|$$

$$[-13/16]_{\text{dir}} = |1|1101000|$$

$$[-13/16]_{\text{inv}} = |1|0010111|$$

$$[-13/16]_{\text{compl}} = |1|0011000|$$

$$[0,25]_{\text{compl}} = |0|0100000|$$

$$[-0,25]_{\text{compl}} = |1|1100000|$$

$$\begin{array}{rcl}
[0,25+0,25]_{\text{compl}} & = & [0,25]_{\text{compl}} \oplus 0|0100000 \oplus \\
& & [0,25]_{\text{compl}} \quad 0|0100000 \\
\hline
[0,5]_{\text{compl}} & = & 0|1000000 \text{ correct result}
\end{array}$$

$$\begin{array}{rcl}
[0,25+13/16]_{\text{compl}} & = & [0,25]_{\text{compl}} \oplus 0|0100000 \oplus \\
& & [13/16]_{\text{compl}} \quad 0|1101000 \\
\hline
& = & 1|0001000 - \text{overflow (r1)} \\
& & \text{!!positive operands, negative sum !!}
\end{array}$$

$$\begin{array}{rcl}
[13/16-0,25]_{\text{compl}} & = & [13/16]_{\text{compl}} \oplus 0|1101000 \oplus \\
& & [-0,25=-4/16]_{\text{compl}} \quad 1|1100000 \\
\hline
[0,5625=9/16]_{\text{compl}} & = & 1|001000 - \text{correct result (r2)}
\end{array}$$

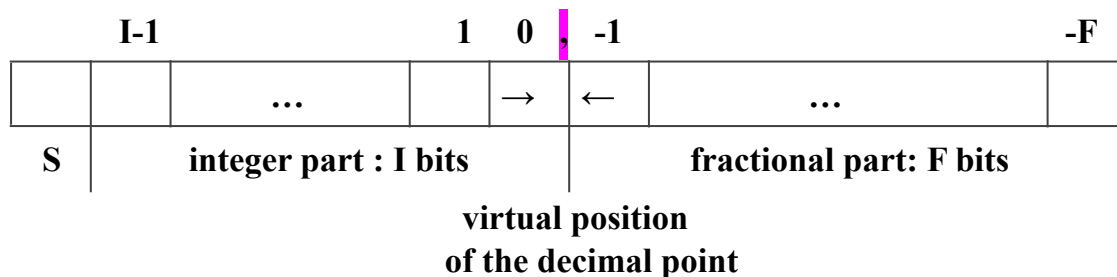
Binary representations for real numbers

❖ fixed point representation and floating point representation

Fixed point representation

- dimension of memory location: n bits (n=16,32,64)
- 3 zones of the memory location with predefined dimensions (1,I,F):
 - the most significant bit (S), position n-1, is the sign bit with the values: 0 for positive numbers and 1 for negative numbers;
 - the decimal point has a fixed position, a virtual one, separating the integer part from the fractional one;
 - $1+I+F = n$ bits (n=16,32,64)
 - the integer part (I bits)
 - memorizes (aligned to the right relative to the virtual position of the decimal point) the digits of the absolute value of the number converted in binary;

- if $I >$ the number of digits of the binary representation of the absolute value of the number, the remaining bits to the left are filled with 0.
- if $I <$ the number of digits of the binary representation of the absolute value of the number, then the most significant digits of the integer part are lost (!! disadvantage).
- the fractional part (F bits)
 - memorizes (aligned to the left relative to the virtual position of the decimal point) the digits of the fractional part
 - if $F >$ the number of binary digits of the fractional part then the remaining digits to the right are filled with 0.
 - if $F <$ the number of binary digits of the fractional part then the least significant digits of the fractional part are lost.



A nonzero real number x is represented in fixed point convention on $n=1+I+F$ bits if: $2^{-F} \leq |x| \leq 2^I - 2^{-F}$

The representation of the minimum absolute value 2^{-F} is:

S	I bits	,	F bits
0	0 ... 0		0 ... 0 1

The representation of the maximum absolute value $2^I - 2^{-F}$ is:

S	I bits	,	F bits
0	1 ... 1		1 ... 1 1

Remark: $=10...0,0...00_{(2)} -$ (2^I)
 $0,0...01_{(2)}$ (2^{-F})

0	0	0	0	1	1	0	0	0	1	1	0	1	0	1	1	=	0C6B
----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	-------------

|
location's content in hexa

Floating point representation

- used to represent very large and very small numbers with a high precision
- if there is an overflow the least significant digits are lost

Any real number x can be written as: $x = \pm 0, m \cdot b^e$ where:

m - mantissa ;

b - numeration base;

e - exponent

Example :

$$1234,5678 = 0,12345678 * 10^4$$

$$0,004371 = 0,4371 * 10^{-2}$$

$$1101,0011_{(2)} = 0,11010011_{(2)} * 2^4$$

Def 1: A real number x is written with a subunitary mantissa and an exponent of base b if $x = \pm 0, m \cdot b^e$

Def 2: A real number x , $x \neq 0$, is written with a subunitary normalized mantissa, if x is written with a subunitary mantissa and an exponent of base b

$$\frac{1}{b} \leq m < 1$$

and

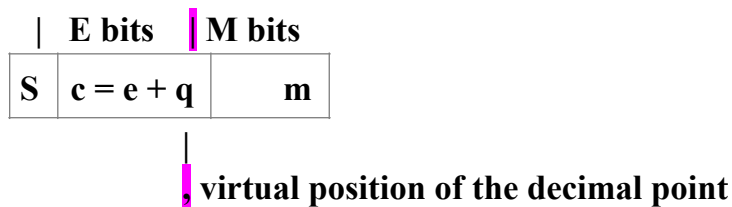
Ex: $0,12345678 * 10^4$ - subunitary normalized mantissa
 $0,004371 * 10^{-4}$ - mantissa is not normalized

Def 3 : A real number x , $x \neq 0$, is written with a $1 < \text{mantissa} < 2$,
if $x = \pm 1, m \cdot 2^e$

Def 4: A real number x is represented in floating point notation if it is used the binary representation with exponent and subunitary mantissa or $1 < \text{mantissa} < 2$.

Remarks:

1. If $1 < \text{mantissa} < 2$ the digit 1 from the integer part is not represented (is hidden) but will be used in the operations with the representations.
2. Computers use now $1 < \text{mantissa} < 2$.



where:

S – sign bit

c - e (exponent) + q(bias)

q – bias (a constant of the computer)

e - the exponent from the binary representation with mantissa and exponent

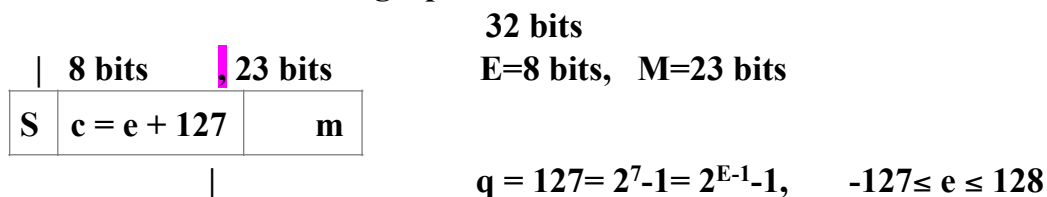
m - mantissa from the binary representation with mantissa and exponent

The precision of representation is provided by the min and max values of the exponent:

$$0 \leq c = (e + q) \leq 2^E - 1 \quad \Rightarrow \quad -q \leq e \leq 2^E - 1 - q$$

Standards IEEE $1 < \text{mantissa} < 2$.

Standard IEEE 754 Single precision



Standard IEEE 754 Double precision



S	c = e + 1023	m
---	--------------	---

$$q = 1023 = 2^{10} - 1 = 2^{E-1} - 1, \quad -1023 \leq e \leq 1024$$

Special values

Value	Representation		
	S	c	m
0+	0	0...0	0...0
0-	1	0...0	0...0
-inf	1	1...1	0...0
+inf	0	1...1	0...0
NaN (not a number)	1 sau 0	1...1	Nonzero value

Remarks

1. $c = 11..1_{(2)}$ is used only for the special values: +inf, -inf, NaN
2. The smallest positive number which can be represented in single precision has the representation:

S	c	,	m
0	00000001		000000000000000000000000

$$c=1 \Rightarrow e + 127(q) = 1 \Rightarrow e = -126$$

Using the hidden bit the value of the mantissa is 1.0 and:

$$\min = 1.0 * 2^{-126} = 2^{-126}$$

2. The largest positive number which can be represented in single precision has the representation:

S	c	,	m
0	11111110		111111111111111111111111

$$c = 11111110_{(2)} = 2^8 - 2 = (\text{the biggest value of } c \text{ for a valid number})$$

$$\Rightarrow e + 127(q) = 254 \Rightarrow e = 127$$

Using the hidden bit, the value of the mantissa is:

$$\begin{aligned} & \textbf{1},\textbf{11111111111111111111}_{(2)}=\textbf{2-0,0000000000000000000000001}_{(2)}=\textbf{2-2^{-23}} \\ & \quad \textbf{\max}=(\textbf{2-2^{-23}})*~\textbf{2^{127}} \end{aligned}$$

Precision	Binary absolute value	Decimal absolute value
Single	$\min = 2^{-126}$ $\max = (2-2^{-23}) \cdot 2^{127}$	$\min \approx 10^{-38}$ $\max \approx 10^{38}$
Double	$\min = 2^{-1022}$ $\max = (2-2^{-52}) \cdot 2^{1023}$	$\min \approx 10^{-308}$ $\max \approx 10^{308}$

Example 1:

Represent in floating point notation, single precision the number 2530,41.

8 is used as an intermediate base:

- **Conversion of the integer part: successive divisions by 8**

$$2530: 8 = 316 \text{ remainder } \underline{2}$$

$$316:8 = 39 \text{ remainder } \underline{4}$$

$$39: 8 = 4 \text{ remainder } \underline{7}$$

4: 8 = 0 remainder 4

$$2530 = 4742_{(8)} = 100\ 111\ 100\ 010_{(2)} = 1,00111100010_{(2)} * 2^{11}$$

The number is written with exponent and a mantissa >1

$$\Rightarrow e=11 \quad \text{şi} \quad c=127+11=138=128+8+2=2^7+2^3+2^1=10001010_{(2)}$$

- **Conversion of the fractional part: successive multiplications by 8**

- there are 11 digits in mantissa from the integer part

- we need 12 more binary digits obtained from the fractional part:

12 binary digits are covered by 4 octal digits: 4 multiplications by 8

$$0,41 \cdot 8 = \underline{3,28}$$

$$0,28 \cdot 8 = \underline{2,24}$$

$$0,24 \cdot 8 = \underline{1,92}$$

$$0,92 \cdot 8 = \underline{7,36}$$

0,41=0,3217₍₈₎=0,011 010 001 111₍₂₎ (rapid conversion)

$$2530,41 = 1,00111100010011010001111_{(2)} * 2^{11}$$

Representation on 32 bits:

S	c	,	m	memory content in hexa	
0	10001010		00111100010011010001111	=	451E268F