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·n bits.

R3) Division on n bits: dividend represented on 2·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:** $0...0_{(2)} = 0$
- **max:** $1…1_{(2)} = 2^n – 1$
- **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:

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

- **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:**
  $[+0]_{dir}$: |0|0....0| and $[-0]_{dir}$: |1|0....0|

<u>Inverse code (complement to 1)</u>

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

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

- **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: $11\ldots1_{(2)} - |x_{(2)}|$;**
- **disadvantage: there are 2 different inverse codes for 0:**
  $[+0]_{inv}$: |0|0....0| and $[-0]_{inv}$: |1|1...1|

<u>Complementary code (complement to 2)</u>

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

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

- **for a positive number x, $[x]_{compl} = [x]_{inv} = [x]_{dir}$**
- **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 $[x]_{compl} = [x]_{inv} + 1$**
- **advantage: there is a unique code for 0: $[0]_{compl}$: |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:

$100 = 1100100_{(2)}$

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

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

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

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

$40 = 101000_{(2)}$

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

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

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

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

Addition and subtraction in complementary code

$[x]_{compl}$ 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>=0:

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

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

The algebraic sum: ⊕, is defined as follows:

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

Rules

r1: if x, y have the same sign, but x⊕y has the opposite sign=>overflow.

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

Example:

$[40+40]_{compl} = [40]_{compl} \oplus$		$0\|0101000 \oplus$
$[40]_{compl}$		$0\|0101000$
--		
$[80]_{compl}$	=	$0\|1010000$ - correct result

$[100+100]_{compl} = [100]_{compl} \oplus$ $\qquad$ $0|1100100 \oplus$

$[100]_{compl}$ 0|1100100

$[200]_{compl}$ = 1|1001000 - overflow (r1)

$[100\text{-}40]_{compl}$ = $[100]_{compl} \oplus$ 0|1100100$\oplus$

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

$[60]_{compl}$ = ~~1~~0|0111100 - correct result (r2)

<u>Subunitary convention:</u>

bit n-1 ,	bit n-2		bit 1	bit 0
S		**…**		

Let $x \in R$, $|x| < 1$, with at most <u>n-1</u> binary digits after the decimal point.
The *direct code* of x is:

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

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

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

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

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

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

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

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

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

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

<u>Remarks:</u>

- **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 $x >= 0$, then $[x]_{compl} = [x]_{inv} = [x]_{dir}$**
- **if $x < 0$, then $[x]_{compl} = [x]_{inv} + 2^{-n+1}$**

<u>Examples:</u>

$0,25 = 0,01_{(2)}$ $\qquad$ $13/16 = 0,D_{(16)} = 0,1101_{(2)}$

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

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

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

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

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

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

$[0,25+0,25]_{compl} = [0,25]_{compl} \oplus$ $\qquad$ $0|0100000 \oplus$

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

$[0,5]_{compl}$ = $\qquad$ $0|1000000$ correct result

$[0,25+13/16]_{compl} = [0,25]_{compl} \oplus$ $\qquad$ $0|0100000 \oplus$

$[13/16]_{compl}$ $\qquad$ $0|1101000$

= $\qquad$ $1|0001000$ - overflow (r1)

!!positive operands, negative sum !!

$[13/16-0,25]_{compl} = [13/16]_{compl} \oplus$ $\qquad$ $0|1101000 \oplus$

$[-0,25=-4/16]_{compl}$ $\qquad$ $1|1100000$

$[0,5625=9/16]_{compl}$ = $\qquad$ $\cancel{1}0|1001000$ - correct result (r2)

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):**
  - **<u>the most significant bit (S)</u>, 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)**
  - **<u>the integer part</u> (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).**
  - **t<u>he fractional part</u> (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.**

	I-1		1	0 ,	-1		-F
		…		→	←	…	
S	integer part : I bits				fractional part: F bits		

virtual position
of the decimal point

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:

$$\begin{array}{rl} =10...0,0...00_{(2)} - & (2^{I}) \\ 0,0...01_{(2)} & (2^{-F}) \\ \hline 1...1,1...11_{(2)} & \end{array}$$

<u>Example 1:</u>

Represent in fixed point convention on 16 bits, I=6 bits, F=9 bits the number: – 29,21.

<u>Conversion into binary:</u>

- integer part: the number is written as a sum of powers of 2:

$$29 = 16+8+4+1= 2^4+2^3+2^2+2^0= 11101_{(2)}$$

- fractional part: successive multiplications by the destination base 2:

9 multiplications by 2 (F=9 bits)

0,21*2=<u>0</u>,42 0,42*2=<u>0</u>,84 0,84*2=<u>1</u>,68 0,68*2=<u>1</u>,36

0,36*2=<u>0</u>,72 0,72*2=<u>1</u>,44 0,44*2=<u>0</u>,88 0,88*2=<u>1</u>,76

0,76*2=<u>1</u>,52

$$0,21=0,001101011_{(2)}$$

$$-29,21=-11101,001101011_{(2)}$$

<u>Representation</u>:

S | 6 bits , 9 bits

1	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**=**	**BA6B**

location's content in hexa

<u>Example 2.</u>

Represent in fixed point convention on 16 bits, I=6 bits, F=9 bits, the number: + 70,21.

70 = 64+4+2= $2^6+2^2+2^1$= $1000110_{(2)}$,

70,21= $1000110,001101011_{(2)}$

<u>Remark</u>:

The integer part does not fit in 6 bits, the most significant bit is lost: 1.

S | 6 bits , 9 bits

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

location’s content in hexa

<u>Floating point representation</u>

- **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

<u>Example :</u>

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

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

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

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

<u>Def 2</u>: 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 and $\frac{1}{b} \leq m < 1$.

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<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<mantissa<2.

Remarks:

1. **If 1<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< mantissa <2.**

	E bits	M bits
S	**c = e + q**	**m**

, virtual position of the decimal point

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<mantissa< 2.

Standard IEEE 754 Single precision

32 bits

	8 bits	23 bits
S	**c = e + 127**	**m**

E=8 bits, M=23 bits

$$q = 127 = 2^7 - 1 = 2^{E-1} - 1, \quad -127 \leq e \leq 128$$

Standard IEEE 754 Double precision

64 bits

| 11 bits , 52 bits   E=11bits,   M=52 bits

S	c = e + 1023	m

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

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	00000000000000000000000

c=1 => e +127(q)=1 => 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	11111111111111111111111

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

⇨ e +127(q)= 254 => e= 127

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

$1,11111111111111111111111_{(2)}=2-0,00000000000000000000001_{(2)}=2-2^{-23}$

max $=(2-2^{-23})* 2^{127}$

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

  316: 8 = 39 remainder 4

  39: 8 = 4 remainder 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

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

- **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*8 = 3,28

  0,28*8 = 2,24

  0,24*8 = 1,92

  0,92*8 = 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