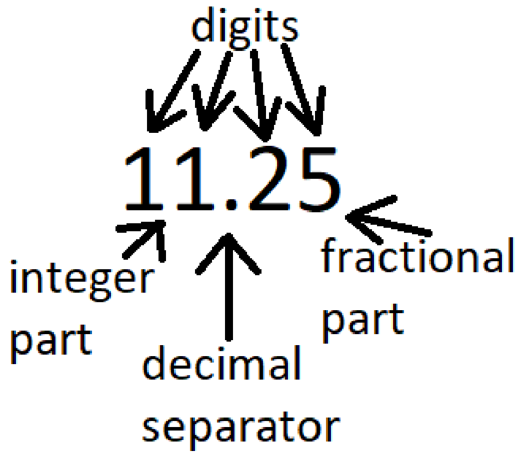
Budapest University of Technology and Economics
Faculty of Transportation Engineering and Vehicle Engineering
Department of Control for Transportation and Vehicle Systems

ENCODING, NUMBER REPRESENTATIONS IN COMPUTING, PART 1.

Lecture 3.

Encoding

- Generally used numeral systems:
  - binary,/base-2, e.g: 00011101
    - each digit is referred to as a bit,
    - used internally by almost all modern computers and computer-based devices,
    - it is a straightforward implementation in digital electronic circuitry using logical gates,
  - decimal/base10/denary, e.g: 156
    - most widely used,
    - fractional part can be:
      - finite,
      - infinite (or non terminating)
      - repeating sequence of digits
    - irrational numbers have infinite (non terminating) decimal representations,
      - irrational number is a real number, that cannot be expressed as a ratio of integers, e.g. π=3.14159……..
    - if we would like to use it for computing, we have to convert it – division/multiplication algorhitms



Encoding

Budapest University of Technology and Economics
Faculty of Transportation Engineering and Vehicle Engineering
Department of Control for Transportation and Vehicle Systems

- hexadecimal/base-16/hex, e.g: 1BE45A
  - it uses sixteen distinct symbols 0…9,A…F,
  - widely used by computer system designers and programmers
- General form of numbers:
  - $A \stackrel{\text{def}}{=} (\pm a_{-m} a_{-m+1} a_{-m+2} \ldots a_{-1} a_0, a_1 a_2 \ldots a_n)$, where
  - $a_{-m}, a_{-m+1}, \ldots, a_{-1} a_0, a_1, \ldots, a_n$ *are the values of the digits in each local value*
  - if the base of the numeral system is r (radix), the number A can be expressed as:
    - $A = \pm \sum_{i=-m}^{n} a_i r^{-i}, where\ 0 \le a_i < r, for\ \forall i$
  - e.g: A=7346
    - $B_{10} = 7346_{10} = 7 * 10^3 + 3 * 10^2 + 4 * 10^1 + 6 * 10^0 = 7000 + 300 + 40 + 6 = 7346_{10},$
    - $B_8 = 7346_8 = 7 * 8^3 + 3 * 8^2 + 4 * 8^1 + 6 * 8^0 = 3584 + 192 + 32 + 6 = 3814_{10},$

Encoding

- fixed-point representation:
  - $A = \pm a_{-m} a_{-m+1} a_{-m+2} \ldots a_{-1} a_0, a_1 a_2 \ldots a_n,$
  - where the integer part of the number is located to the left from the radix point, and the fractional part is located to the right from the radix point,
  - it is generally used to represent numbers with less digits,
- floating-point representation:
  - $A = \pm m * r^{\pm k}, where\ r^{-1} \leq |m| < r^0,$
  - m= mantissa (significand),
  - k=characteristic,
  - r=radix (base)
  - e.g: $-0.999*10^{+41}$, r=10
  - every number can be represented in this form!
  - nowdays, r is equal to 2 or 16 in modern computers!

Encoding

- Encoding: it is needed to convert an information into an appropriate form,
  - appropriate form: favorable form to data processing,
- Generally used encoding systems:
  - for numbers:
    - pure binary code,
    - complement code,
    - inverse binary code,
    - binary-coded decimal – BCD,
    - Stibitz code,
    - Gray code,
    - etc…

Encoding

- for characters:
  - telex-code:
    - started in the 1930's, it was a point-to point teleprinter system, it was last used in the United Kingdom in 2008
    - used 5 digits, worked with number-character changing characters

e.g. message: 3/x + 2 expression

1. number changer	11011
2. 3	00001
3. / – slash → „number"	11101
4. character changer	11111
5. x → „letter"	11101
6. number changer	11011
7. +	10001
8. 2	10011

the same code, with different meaning

Telestar 12x
source: http://www.cryptomuseum.com/telex/telefunken/telestar/index.htm

Encoding

- ASCII code:
  - American Standard Code for International Interchange,
  - earliest version: 7 digits + 1 specified bit,
  - 2^7=128 code words, 7. digit: parity bit, contained numbers from 0 to 9, lower case letters from a to z, uppercase letters from A to Z, punctuation sysmbols, control codes, space…

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d

source: http://www.asciitable.com/

- Extended ASCII:
  - 2^8=256 code words, ASCII+extensions,
  - ISO 8859-1: Latin 1. for Western European Languages, ANSI
  - ISO 8859-2: Latin 2. for Eastern European Languages,
  - ISO 8859-3 for Cyrillic Languages

128	Ç	144	É	160	á	176	░	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠

source: http://www.asciitable.com/

- Code page 1252:
  - it is a compatible subset of ISO 8859-1 with extra characters,
  - this is the standard character encoding of Western European language versions of Microsoft Windows, including English versions

Encoding

- UNICODE – ISO/IEC 104646:
  - most recent version: Unicode 11.0, contains 137439 characters, covering 146 modern and historic scripts ☺
  - 16 digits in 17 plains, in every plan 65535 code words,
  - 0. plain: Basic Multilingual Plane (Latin-1),
  - 1. plain. Supplementary Multilingual Plane,
  - 2. plain: Supplementary Ideographic Plane,
  - 3…13. plains: unassigned,
  - 14. plain: Supplementary Special Purpose Plane,
  - 15, 16. plains: Supplementary Private Use Area
  - $17*2^{16}$=1114112 pieses of characters (possibility)
- etc…

Encoding

- for images:
  - BMP,
  - JPEG,
  - etc…

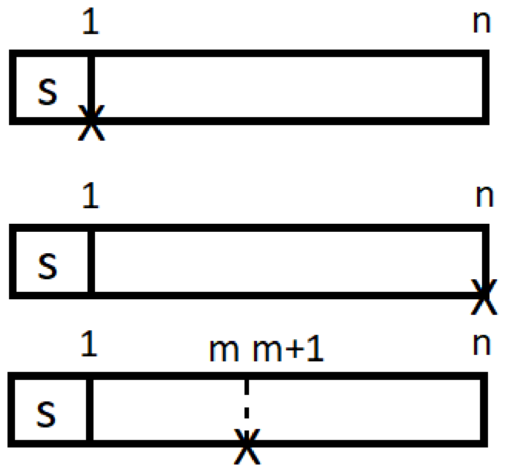
a	b	COL	R	G	B	HUE	H	S	V
0.0	0.0		255	0	0		0	1.00	1.00
0.0	0.0		255	0	0		0	1.00	1.00
0.0	0.0		255	0	0		0	1.00	1.00
0.3	0.0		255	48	48		0	0.81	1.00
0.3	0.0		255	48	48		0	0.81	1.00
0.6	0.0		255	87	87		0	0.66	1.00
0.9	0.0		255	118	118		0	0.54	1.00
1.2	0.0		238	134	134		0	0.44	0.93
1.5	0.0		214	139	139		0	0.35	0.84
1.5	0.0		214	139	139		0	0.35	0.84
1.8	0.0		193	138	138		0	0.29	0.76
1.8	0.0		193	138	138		0	0.29	0.76
0.0	0.3		255	48	48		0	0.81	1.00
0.3	0.3		65	255	255		180	0.75	1.00
0.3	0.3		65	255	255		180	0.75	1.00
0.6	0.3		175	95	255		270	0.63	1.00
0.9	0.3		255	123	255		300	0.52	1.00
1.2	0.3		235	135	210		315	0.42	0.92
1.5	0.3		212	139	183		324	0.35	0.83
1.8	0.3		192	138	165		330	0.28	0.75
0.0	0.6		255	87	87		0	0.66	1.00
0.3	0.6		175	255	95		90	0.63	1.00
0.6	0.6		113	255	255		180	0.56	1.00
0.9	0.6		131	131	248		240	0.47	0.97
1.2	0.6		182	137	227		270	0.39	0.89
1.5	0.6		193	139	206		288	0.33	0.81
1.8	0.6		187	137	187		300	0.27	0.73
0.0	0.9		255	118	118		0	0.54	1.00
0.3	0.9		255	255	123		60	0.52	1.00
0.6	0.9		131	248	131		120	0.47	0.97
0.6	0.9		131	248	131		120	0.47	0.97

a	b	COL	R	G	B	HUE	H	S	V
0.9	0.9		136	232	232		180	0.41	0.91
1.2	0.9		139	158	214		225	0.35	0.84
1.5	0.9		150	138	197		252	0.30	0.77
1.8	0.9		157	135	180		270	0.25	0.70
0.0	1.2		238	134	134		0	0.44	0.93
0.3	1.2		235	210	135		45	0.42	0.92
0.6	1.2		182	227	137		90	0.39	0.89
0.9	1.2		139	214	158		135	0.35	0.84
0.9	1.2		139	214	158		135	0.35	0.84
1.2	1.2		139	200	200		180	0.31	0.79
1.5	1.2		136	156	185		216	0.26	0.73
1.8	1.2		132	132	170		240	0.22	0.67
0.0	1.5		214	139	139		0	0.35	0.84
0.3	1.5		212	183	139		36	0.35	0.83
0.6	1.5		194	206	139		71	0.33	0.81
0.9	1.5		150	197	138		108	0.30	0.77
1.2	1.5		136	185	156		144	0.26	0.73
1.5	1.5		133	173	173		180	0.23	0.68
1.8	1.5		129	144	160		210	0.20	0.63
0.0	1.8		193	138	138		0	0.29	0.76
0.3	1.8		192	165	138		30	0.28	0.75
0.6	1.8		187	187	137		60	0.27	0.73
0.9	1.8		157	180	135		90	0.25	0.70
1.2	1.8		132	170	132		120	0.22	0.67
1.2	1.8		132	170	132		120	0.22	0.67
1.5	1.8		129	160	144		150	0.20	0.63
1.8	1.8		124	149	149		180	0.17	0.59

source:
http://mkweb.bcgsc.ca/tupleencode/?color_charts

Binary Encoding

- Fixed-point arithmetic
  - the radix point can be:
    - before the first data bit,
    - after the first data bit,
    - between those,
      - for real fractional numbers, range: $-1 \leq N \leq 1 - 2^{-n}$, $precision$: 2^{-n}
      - for integers (commonly used), range: $-2^{n} \leq N \leq 2^{n} - 1$, $precision$: 1
      - for other fractional numbers, range: $-2^{m} \leq N \leq 2^{m} - 2^{m-n}$, $precision$: 2^{m-n}
  - the signed bit is the first bit (usually):
    - it is 0, if the number is positive,
    - it is 1, if the number is negative



Binary Encoding

- Negative numbers in fixed-point arithmetic?
  - real numbers not exist in the registers!
  - the integers are represented in 2's complement code!
    - with the aim of this method, the substraction may originate in summation
    - $N_{2c} = \begin{cases} N, if\ N \geq 0 \\ 2^k - |N|, if\ N < 0, where\ k = number\ of\ the\ digits\ (sign + useful\ digits) \end{cases}$
    - e.g, if k=8:
      - 65 → 01000001
      - - 65 → 10111111 (256-65 = 191)

Binary Operations

- Addition in 2's complement code:
  - requisite: $A_{2c} + B_{2c} = (A+B)_{2c}$,
  - **instead of substraction, we have to realize addition in the case of the complement coded numbers!**
  - Case 1:
    - $A > 0$ and $B > 0$ and $A > B$
    - in this case: $A_{2c} = A_b$ and $B_{2c} = B_b$
    - then: $A_{2c}+B_{2c} = A_b + B_b = (A+B)_{2c}$

Binary Operations

- Case 1, e.g:
  - A =17, B=9, k=8
  - in this case: $A_{2c} = A_b$=00010001 and $B_{2c} = B_b$ =00001001, if k=8
  - then: $A_{2c}+B_{2c} = A_b + B_b = (A+B)_{2c} \equiv 26$ =00011010

CY	sign	useful bits	remarks
0	0	0010001	A_b (17)
0	0	0001001	B_b (9)
0	0	0011010	sum (26)

  - CY = carry

Binary Operations

- Case 1, - problem, the sum is bigger, then the number range, e.g:
  - A =90, B=56, k=8
  - in this case: $A_{2c} = A_b$=01011010 and $B_{2c} = B_b$ =00111000, if k=8
  - then: $A_{2c}+B_{2c} = A_b + B_b = (A+B)_{2c}$ =10010010!!!

CY	sign	useful digits	remark
0	0	1011010	A_b (90)
0	0	0111000	B_b (56)
0	1	0010010	sum (-18)

  - the result is wrong, but CY=0! solution: e.g: OV – overflow bit is 1, if the result is not in the range: -128…127
    - eg. in this case 10010010 = 146 in denary numeral system

Binary Operations

- Case 2:
  - A > 0 and B < 0 and $|A| > |B|$
  - in this case: $A_{2c} = A_b$ and $B_{2c} = 256 - B_b$, if k=8
  - then: $A_{2c}+B_{2c} = A_b + 256 - B_b = (A_b - B_b) + 256$
  - considering, that $(A_b - B_b) > 0$, there is an unnecessary bit – carry – in the result

- Case 2, e.g:
  - A =17, B=-9, k=8
  - in this case: $A_{2c} = A_b$=00010001 and $B_{2c} = 256 - B_b$ =11110111, if k=8
  - then: $A_{2c}+B_{2c} = (A_b - B_b)_{2c} + 256 = \equiv 8 =$100001000 with an unnecessary bit

Binary Operations

Budapest University of Technology and Economics
Faculty of Transportation Engineering and Vehicle Engineering
Department of Control for Transportation and Vehicle Systems

CY	sign	useful digits	remarks
0	0	0010001	A_b (17)
0	1	1110111	B_{2c} (-9)
1	0	0001000	sum (8)

- the result is good, but CY=1, that is the unnecessary bit!

Binary Operations

- Case 3:
  - $A < 0$ and $B > 0$ and $|A| > |B|$
  - in this case: $A_{2c} = 256 - A_b$ and $B_{2c} = B_b$, if $k=8$
  - then: $A_{2c} + B_{2c} = 256 - A_b + B_b = 256 - (A_b - B_b)$ considering, that $(A_b - B_b) > 0$, the result will be a negative number in 2's complement code!

- Case 3, e.g:
  - $A = -17$, $B = 9$, $k=8$
  - in this case: $A_{2c} = 256 - A_b = 11101111$ and $B_{2c} = B_b = 00001001$, if $k=8$
  - then: $A_{2c} + B_{2c} = (A_b - B_b)_{2c} + 256 = \equiv -8 = 11111000$
  - the result is good, because the 2's complement code of $-8 = 256 - 8 = 11111000$

Binary Operations

CY	sign	useful digits	remarks
0	1	1101111	A_b (-17)
0	0	0001001	B_{2c} (9)
0	1	1111000	sum (-8)

- the result is good, the signed bit =1!

Binary Operations

- Case 4:
  - A < 0 and B < 0 and $|A| > |B|$
  - in this case: A_{2c} =256- A_b and B_{2c} =256-B_b, if k=8
  - then: A_{2c}+B_{2c} =256-A_b + 256-B_b =256- (A_b+B_b)+256considering, that (A_b - B_b) > 0, the result will be a negative number in 2's complement code, and also will be in the result an unnecessary bit!

- Case 4, e.g:
  - A =-17, B=-9, k=8
  - in this case: A_{2c} = 256-A_b=11101111 and B_{2c} = 256-B_b =11110111, if k=8
  - then: A_{2c}+B_{2c} = $(A_b\text{-}B_b)_{2c}$+256=$\equiv -26 =$111100110 with an unnecessary bit
  - the result is good, because the 2's complement code of -26 =256-8=11100110

Binary Operations

CY	sign	useful digits	remarks
0	1	1101111	A_{2c} (-17)
0	1	1110111	B_{2c} (-9)
1	1	1100110	sum(-26)

- the result is good, the signed bit =1!, but CY=1, that is the unnecessary bit!

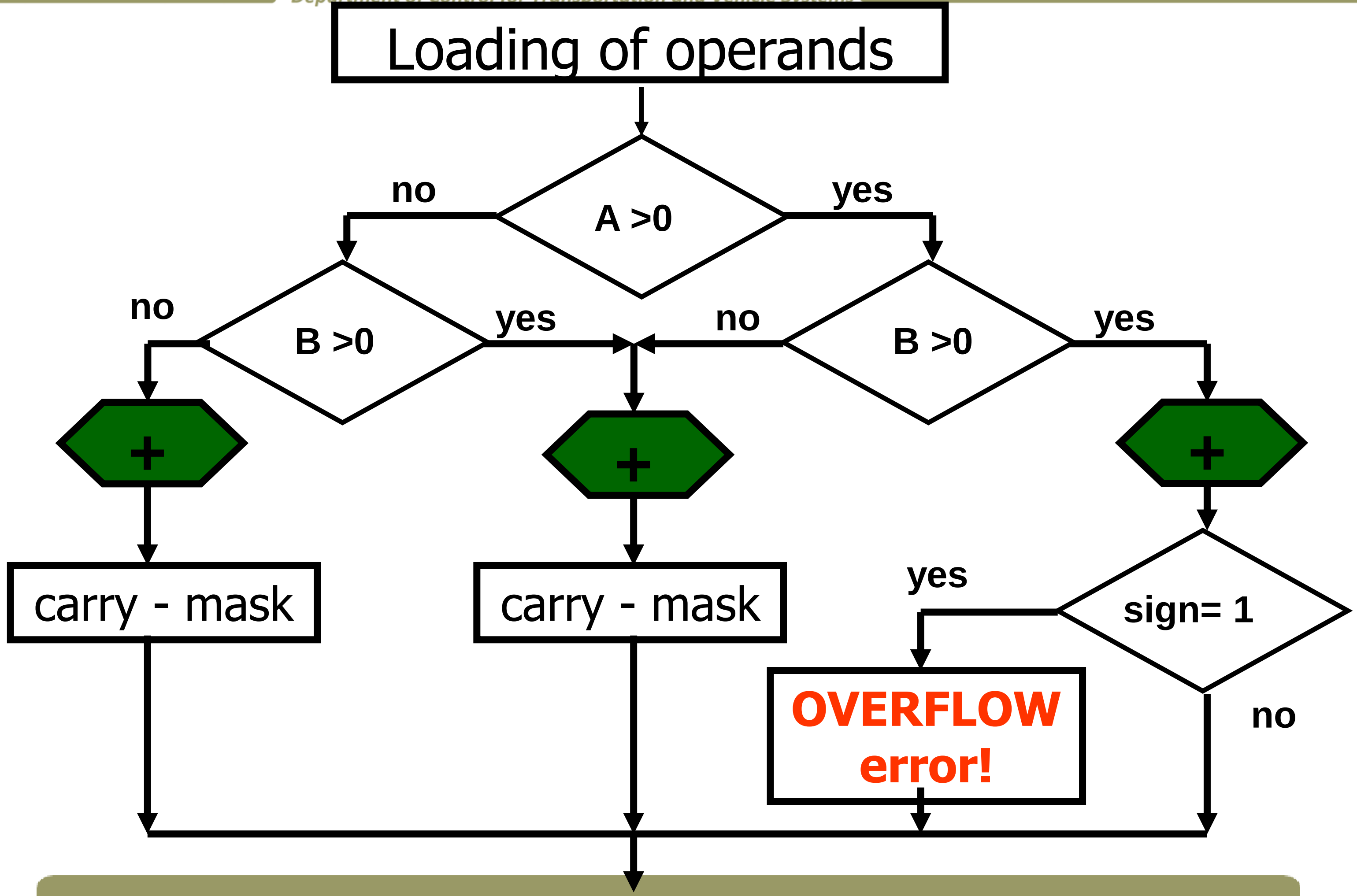
Binary Operations

- Summary of addition (substraction) in 2's complement code:

Case	A	B	Carry	Remark
1	>0	>0	-	Result is good, (sign= 0)
2	>0	<0	exist	CY, result is good
3	<0	>0	-	Result is good (in 2'c)
4	<0	<0	exist	CY, (result in 2'c)

Binary Operations

- Logical scheme of a binary adder (e.g. in an ALU):



Binary Operations

- Fractional numbers in 2's complement code:
  - $N_{2c} = \begin{cases} N, & if\ 1 > N \geq 0 \\ 2 - |N|, & if\ -1 < N < 0 \end{cases}$
  - e.g:
    - N=-0.75
    - in binary form: $-0.75_{10}=-0.11_2$
    - $2_{10}=10_2$, then:

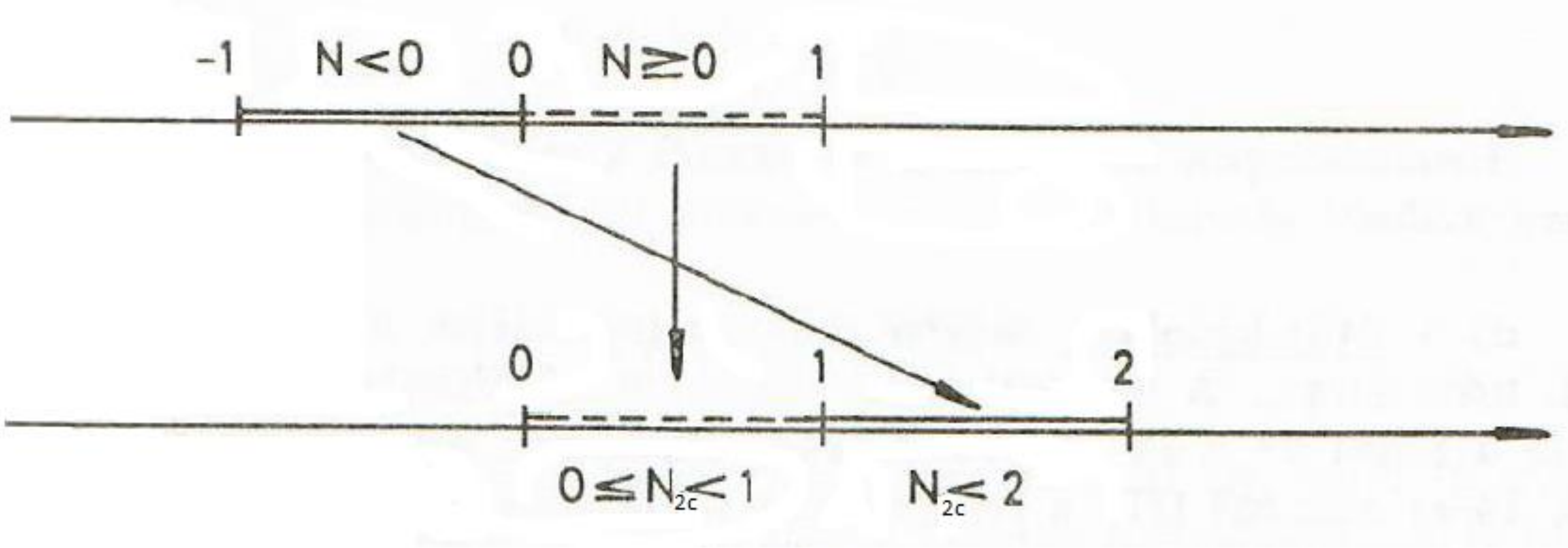
	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	
	1	0.	0	0	0	(2)
-		0.	1	1	0	(-|N|)
		1.	0	1	0	(N2c)

    - the signed bit is the bit located at the local value 2^0

Binary Operations

Budapest University of Technology and Economics
Faculty of Transportation Engineering and Vehicle Engineering
Department of Control for Transportation and Vehicle Systems

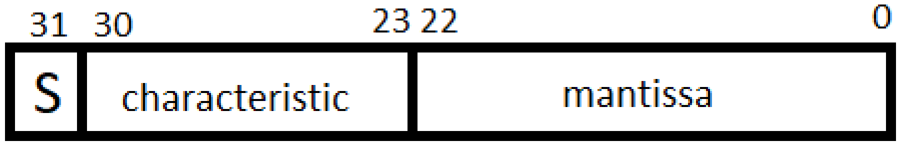
- Fractional numbers in 2's complement code:
  - with other words, it is a transformation, shown on the next picture, if the fractional number is: betwen -1…1



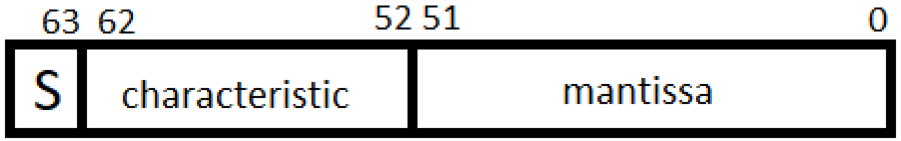
  - not used in computer technology…

Binary Encoding

- Floating-point arithmetic – standard IEEE 754 -1985, nowdays: ISO/IEC/IEEE 60559:2011:
  - $A = \pm m * 2^{\pm k}$, - every number can be written in this form,
  - two main types are (other types also exist):
    - single-precision floating-point number – number representation in 32 digits, called also binary32



    - double-precision floating-point number – number representation in 64 digits, called also binary64

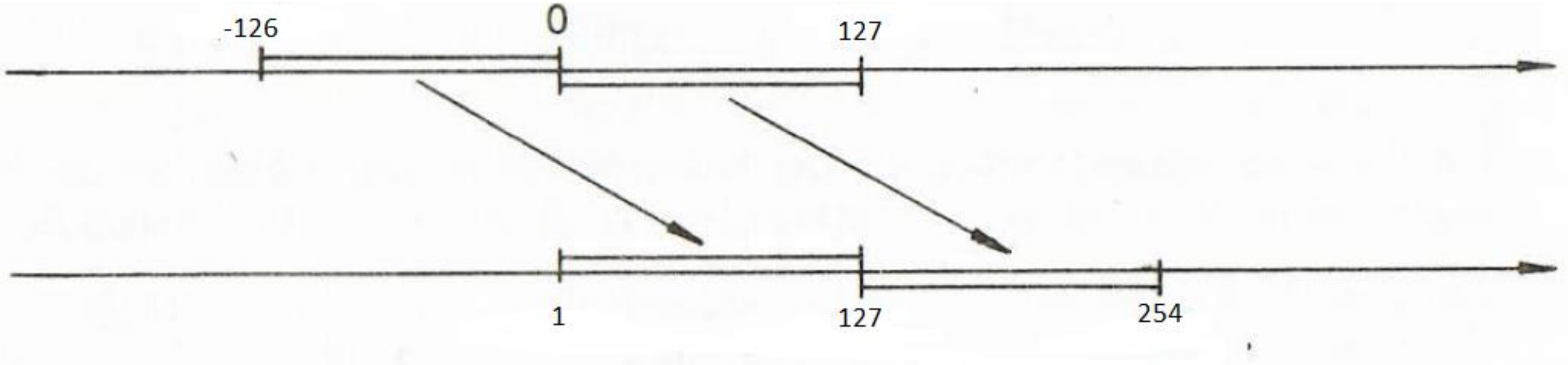


  - signed bit:
    - 0, if the number is positive
    - 1, if the number is negative

Binary Encoding

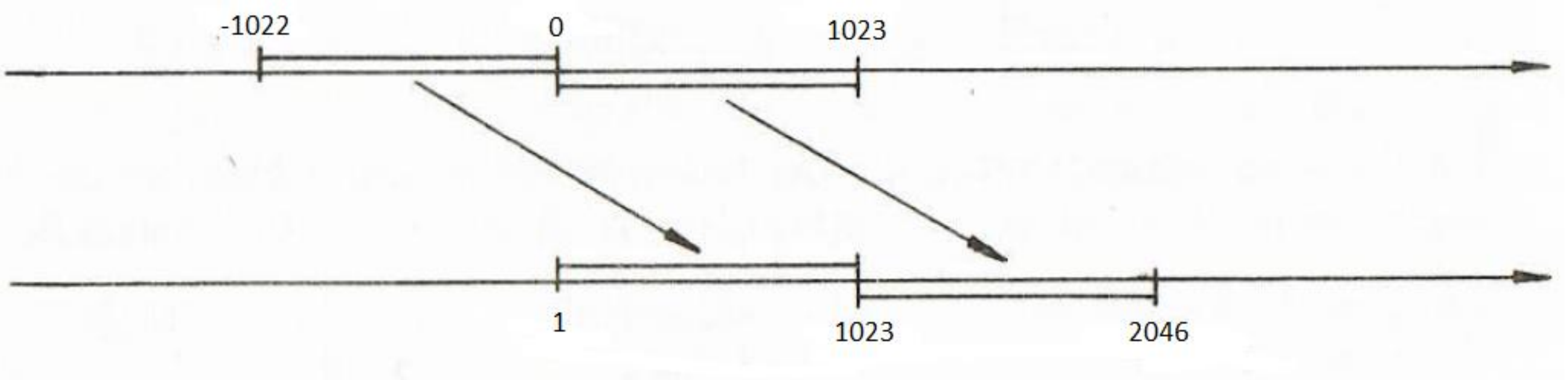
Budapest University of Technology and Economics
Faculty of Transportation Engineering and Vehicle Engineering
Department of Control for Transportation and Vehicle Systems

- characteristic – single-precision:
  - $-2^{-7} + 2 \leq k < 2^7 - 1$
  - $-126 \leq k < 127$ offset zero-point representation:
  - -126=00000001
  - -125=00000010
  - -124=00000011
  - …
  - 0=01111111
  - 1=10000000
  - 2=10000010
  - …
  - 127=11111110



Binary Encoding

- characteristic – double-precision:
  - $-2^{10} + 2 \leq k < 2^{10} - 1$
  - $-1022 \leq k < 1023$ offset zero-point representation:
  - -1022=00000000001
  - -1021=00000000010
  - -1020=00000000011
  - …
  - 0=01111111111
  - 1=10000000000
  - 2=10000000001
  - …
  - 1023=11111111110



Binary Encoding

- range of floating-point numbers:
  - single precision floating point numbers: $-(1-2^{-23}) * 2^{127} \leq N \leq (1-2^{-23}) * 2^{127}$
  - double precision floating point numbers: $-(1-2^{-52}) * 2^{1023} \leq N \leq (1-2^{-52}) * 2^{1023}$
  - $N_{max} \approx 2^{1023} \approx 9 * 10^{307}$
- precision of floating-point numbers:
  - single precision floating point numbers: $2^{-23} * 2^{127} = 2^{104}$
  - double precision floating point numbers: $2^{-52} * 2^{1023} = 2^{971}$
  - conversion from denary numeral system to floating-point arithmetic:

1. convert to binary form,
2. convert to normalized binary form,
3. calculation of the characteristic,
4. writing in the single/double precision floating point representation

Binary Encoding

- conversion from denary numeral system to single precision floating-point representation, e.g:
  1. convert to binary form,
     - by using the division and multiplication algorithmes:
     - $635{,}015625_{10}=1001111011{,}000001_2$
  2. convert to normalized binary form,
     - $=1001111011{,}000001=1001111011{,}000001*2^0$
     - $=1001111011{,}000001*2^0=1{,}001111011000001*2^9$
  3. calculation of the characteristic,
     - by using the offset zero point representation, c=127+k=127+9=136
     - $136_{10}=10001000_2$ by using the division algorithm
  4. writing in the single precision representation
     - in binary fom:
     - in hexadecimal form: 441EC100

S 30 23 22 0
01000100000111101100000100000000

Binary Encoding

- conversion from single precision floating-point representation to denary numeral system, e.g:
  1. writing in the single precision representation
     - in hexadecimal form: C4F9AB10
     - in binary fom:

```
S 30      23 22                             0
1 1000100 1 111 1001 1010 1011 0001 0000
```

  2. calculation of the characteristic,
     - by using the offset zero point representation, $c=10001001_2=137_{10}$
     - k=c-127=137-127=10
  3. convert from normalized binary form to binary form,
     - $=-1,11110011010101100001*2^{10}=-11111001101,010110001*2^0$
  4. convert to decimal form,
     - $-11111001101,010110001_2=-1997,345703125_{10}$

End of Lecture 3.

Thank you for your attention!