IT Basics
3

Faculty of Cybernetics, Statistics and Economic Informatics – BUES

Prof. Răzvan ZOTA, Ph.D.

zota@ase.ro

https://zota.ase.ro/itb

Numbers' representation in a computer

Fixed point representation

- direct code (binary)
- inverse code (one's complement)
- complement code (two's complement)

Floating point representation

- Single precision (32 bits used)
- Double precision (64 bits used)
- Extended format (80 bits used)

Fixed point representation

Fixed point representation

- direct code (binary)
- inverse code (one’s complement)
- complement code (two’s complement)

Direct code

$$R = a_n * 2^n + \sum_{i=0}^{n-1} a_i 2^i$$

a_n – sign bit

a_n=0 if R>=0

a_n=1 if R<0

Inverse code (1's complement)

$$R^{CI} = \begin{cases} 0 \bullet 2^n + \sum_{i=-m}^{n-1} a_i \bullet 2^i \ , if \ \ R \geq 0 \\ 1 \bullet 2^n + \sum_{i=-m}^{n-1} \overline{a}_i \bullet 2^i , if \ \ R < 0 \end{cases}$$

$\overline{a}_i = 1 - a_i, \forall i = \overline{-m, n-1}$, where a_i are the binary digits of $|R|$ in direct code

Calculation method :

$$R^{CI} = 2^{n+1} - |R|_{CD} - 2^{-m}$$

Complementary code

$$R^{CC} = \begin{cases} 0 \bullet 2^n + \sum_{i=-m}^{n-1} a_i \bullet 2^i \ , if \ \ R \geq 0 \\ 1 \bullet 2^n + \sum_{i=-m}^{n-1} \overline{a}_i \bullet 2^i , if \ \ R < 0 \end{cases}$$

$\sum_{i=-m}^{n-1} \overline{a}_i \bullet 2^i = \sum_{i=-m}^{n-1} \overline{a}_i \bullet 2^i + 2^{-m}$, where $\overline{a}_i = a_i - 1$ and a_i are the binary digits of $|R|$ in direct code

Calculation method:

$$R^{CC} = 2^{n+1} - |R|_{CD}$$

Addition/subtraction in fixed point

- Addition in DC, IC and CC
- Subtraction in IC and CC (Ex. 93-27 in IC and CC)

Length, Precision and Range for Fixed-Point representation

Type	Length	Precision	Values range (binary)	Values range (decimal)
Word format	16	15	$-2^{15} – 2^{15}-1$	[-32768 ; 32767]
Short format	32	31	$-2^{31} – 2^{31}-1$	$[-2.14*10^{9}$; $2.14*10^{9}$
Long format	64	63	$-2^{63} – 2^{63}-1$	$-9.22*10^{18} – 9.22*10^{18}$

Floating Point Representation

- This representation has 3 parts:
  - Sign bit
  - Exponent (*characteristic* or *scale*)
  - Fraction (*mantissa* or *significand*)
- From the 1990's, the main used standard for FPR is represented by **IEEE 754** (published in 1985)
- **IEEE - Institute of Electrical and Electronics Engineers**

Normalized Numbers

- In most of the cases, the numbers are represented in *normalized form*. Except for zero, the significand is always made of an integer of 1 and the following fraction: 1.fffffff
- Numbers are represented as:

$$+/- 1.fff\ldots fff * 2^{exp}$$

S=0 or S=1

CAR = exp + K (K= biasing constant)

Fraction = fff…fff

S	CAR	Fraction

Numbers and special values

- **Signed Zeros** – Zero value can be represented as +0 or –0 depending on the sign bit. Both representations are equal as value. The sign of a zero result depends on the operation being performed and the rounding process being used.
- **Finite numbers - normalized and denormalized**.
- **+∞, -∞** are representing the maximum/minimum positive/negative value for real numbers for floating point representation. Infinite value is always represented by a fraction of zero and the maximum exponent for that format (255 for example, in single precision format). Exceptions are generated when an infinite value is used as a source operand and leads to an invalid operation.
- **NaN values** (Not a Number) – these are not real numbers. Their representation is made by using a maximum accepted exponent and a non-zero fraction; the sign bit is ignored.

Normalized and denormalized numbers

- Non-zero, finite numbers are divided into two classes: *normalized* and *denormalized*. The normalized finite numbers are all the non-zero finite values that can be encoded in a normalized real number format between 0 and ∞. This group includes all the numbers with biased exponents between 1 and 254 (unbiased, the exponent range is between –126 and 127)

- In the case when floating point numbers become very close to zero, the normalized number format cannot be used to represent the numbers, because the range of the exponent is not large enough to compensate for shifting the binary point to the right to eliminate leading zeros.

Normalized and denormalized numbers

- When the biased exponent is 0, smaller numbers can be represented by making the integer bit of the significand *zero*. The numbers from this range are called denormalized (*tiny*) numbers. This denormalization process leads to the loss of precision (the number of significand bits in the fraction is reduced by the leading zeros).

- When the normalized floating-points calculations are made, the IA-32 processor normally operates on normalized numbers and produces normalized results. Denormalized numbers are representing an *underflow* condition. A denormalized number is computed by a technique called *gradual underflow*.

Real Numbers and NaN

-0	1	0	0
+0	0	0	0
- Denormalized finite	1	0	0.fff
+ Denormalized finite	0	0	0.fff
- Normalized finite	1	1…254	Any value
+ Normalized finite	0	1…254	Any value

Real Numbers and NaN (cont.)

- ∞	1	255	0
+ ∞	0	255	0
- SNaN	x	255	1.0ff
+ SNaN	x	255	1.0ff
- QNaN	x	255	1.1ff
+ QNaN	x	255	1.1ff

Denormalization process

Operation	Sign	Exponent	Significand
True result	0	-129	1.010111000...000
Denormalize	0	-128	0.1010111000...000
Denormalize	0	-127	0.01010111000...000
Denormalize	0	-126	0.001010111000...000
Result in denormalized format	**0**	**-126**	**0.001010111000...000**

NaN values

- The IEEE standard defines two classes of NaNs:
  - QNaN (quiet NaN) – the MSB is set
  - SNaN (signaling NaN) – the MSB is zero.
- QNaNs are propagating through the arithmetic operations without indicating an exception
- SNaNs are signaling an exception (non-valid operation) when they are operand in arithmetic operations

Special operations

Operation	Result
n / ±∞	0
± ∞ * ± ∞	± ∞
±Non-zero val / 0	± ∞
∞ + ∞	∞
±0 / ±0	*NaN*
∞ - ∞	*NaN*
± ∞ / ± ∞	*NaN*
± ∞ * 0	*NaN*

Real Data Types

Single precision

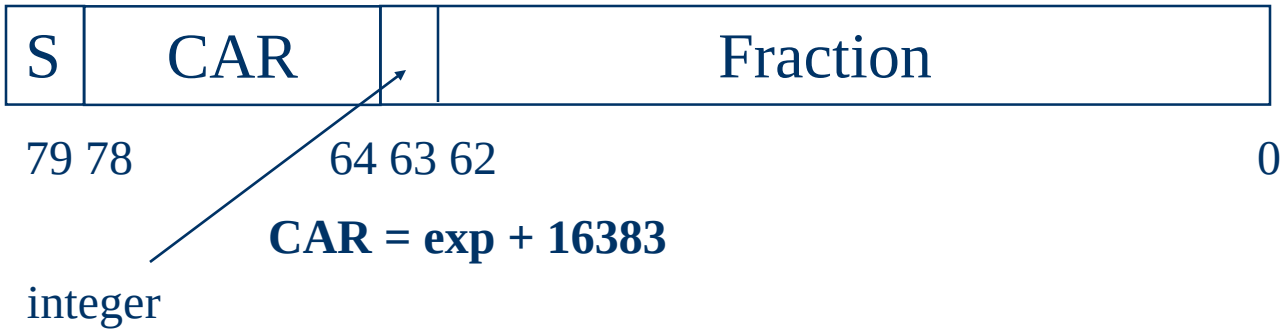
CAR = exp + 127

Double precision

CAR = exp + 1023

Real Data Types (cont.)

Extended real format



If *n* represents the number of bits for the characteristic the we may use the formula:

CAR = exp + 2^{n-1} - 1

Length, precision and range for real numbers

Type	Length	Precision	Value range (binary)	Value range (decimal)
Single precision	32	24	2^{-126} - 2^{127}	$1.18*10^{-38}$ – $3.40*10^{38}$
Double precision	64	53	2^{-1022} - 2^{1023}	$2.23*10^{-308}$ – $1.79*10^{308}$
Extended real	80	64	2^{-16382} - 2^{16383}	$3.37*10^{-4932}$ – $1.18*10^{4932}$

Examples

1. Which is the decimal value for the number (represented in single precision format):

1 10000001 01000000000000000000000

The characteristic is 129, so the real exponent is 129 – 127 = 2

Fraction is $.01_2$ = .25, so will have the value of 1.25.

The number is negative (the sign bit is 1)

The result: $-1.25 \times 2^2 = -5$

Examples

2. Which is the representation of 16.625 in single precision format ?

We represent the number in normalized format:

$16 = 10000_2$

$.625 = .101_2$

$16.625 = 10000.101 = 1.0000101 * 2^4$

CAR = 4 + 127 = 131

In conclusion, the representation is the following:

0 10000011 00001010000000000000000

BCD format representation

BCD (Binary Coded Decimal) Format:

- Packed BCD
- Unpacked BCD

In packed BCD there are representing two decimal digits using a byte (the LSD on 0-3 bits and the MSD on 4-7 bits):

96 = 1 0 0 1 0 1 1 0

7 4 3 0

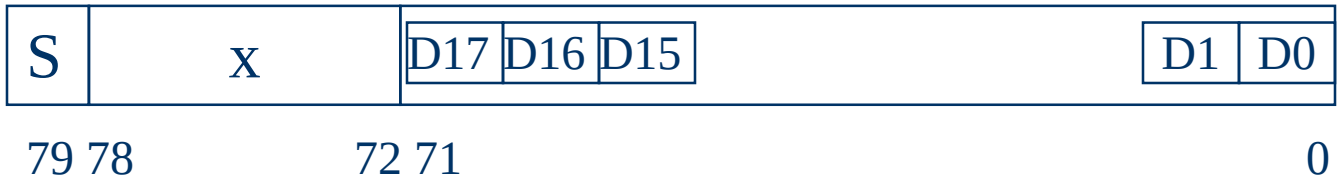
In unpacked BCD a digit is represented using a byte in bits 0-3, and the bits 4-7 are containing the value F_h:

6 = 1 1 1 1 0 1 1 0

7 4 3 0

BCD representation for Intel

Type	Length	Precision	Value domain (decimal)
Packed BCD	80	18 (decimal digits)	$(-10^{18}+1) – (10^{18}-1)$



Addition in BCD

- Addition in BCD – normally addition in binary, for each group of 4 binary digits, considering the following cases. If a and b are the two decimal digits coded in binary, the result is:
  - Correct, if 0000 < c <=1001
  - Wrong, and we add **0110** in both these two cases:
    - **1010 <= c <=1111** – it doesn't match to a decimal digit (addition of 0110 will determine a transport to the next level)
    - **0000 <= c < 1001**, with the appearance of the 5th digit, 1, which represents a transport for the next group of 4 binary digits

Addition example in BCD

Exemplu: Să se efectueze în BCD suma 5683 + 2794.

5683 +	0101 +	0110 +	1000 +	0011 +
2794	0010	0111	1001	0100
8477	0001 ←	0001 ←	← 1 0001 +	0111
	1000	1110 +	0110	7
	8	0110	0111	
		← 1 0100	7	
		4		

Mai sus transportul apărut din tetrada anterioară este evidențiat prin „← ”.

Subtraction in BCD

- Subtraction in BCD – normally subtraction in binary, for each group of 4 binary digits, considering the following cases:
- If a and b are the two decimal digits coded in binary, the result **c = a - b** is:
  - correct, if **a >= b**
  - if **a < b,** we have to borrow 1 from the next group of 4 binary digits, we make the subtraction, then we subtract the correction value of **0110**.