COMPUTER ARCHITECTURE

Code: CT173

Part II: Data Representation in Computer System

MSc. NGUYEN Huu Van LONG

Department of Computer Networking and Communication,

College of Information & Communication Technology,

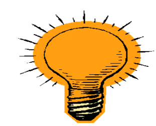
CanTho University

Agenda

- State of the art for Data binary representation
- Human perspective vs Computer perspective
- Simple Number vs Complex Number
- **Fixed point number**
  - Unsigned number: Dec, Bin, Octal, Hex theorem
  - Signed and Magnitude number
  - One's complement number
  - Two's complement number
- **Floating point number**
  - Scientific notation
  - Bias representation (Excess N)
  - IEEE 754 standard
- **Binary Coded Decimal – BCD**
- Character codes (ASCII, Unicode)

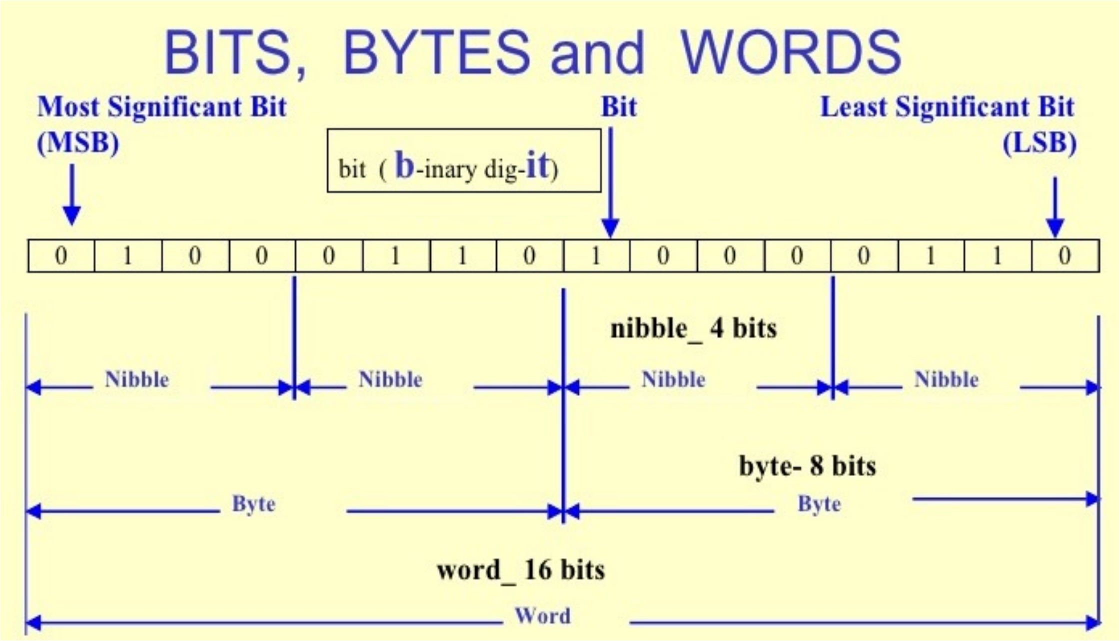
State of the art: Data Binary Representation

- Early computer (Mark I, ENIAC) designs were decimal and they soon met **several problems**:
  - Transistor operations are *not always stable*
  - Various of computer systems *hardly to agree the intense of 10 level of Voltage*
  - *Huge amount hardware* exists just for detecting which level of Voltage is
- In early 1945, John Von Neumann proposed binary data processing in his own named computer architecture - **John Von Neumann machine**
- **Advantages of Binary** data processing machine are
  - Detection of Voltage is *easy with 2 levels*: Presence of Absence
  - Make a *unique processing for Voltage in various computers*
  - *Natural relationship* between on/off switches and calculation using Boolean logic
  - *Remove unnecessary complex hardware*

	
On	Off
True	False
Yes	No
1	0

Binary Data Information

- A ***bit*** is the most basic unit of information
  - It is a state of On or Off in a digital circuit
  - Or **High** or **Low** voltage
- A ***byte*** is a group of eight bits, the smallest possible addressable unit of computer storage.
- A ***word*** is a contiguous group of bytes
  - Word sizes of 1 bytes, 2 bytes, 4 bytes, 8 bytes are most common



Human perspective to Computer perspective

- *Human uses Decimal system (Based 10)* to do their work but *Computer processes and calculates based on Binary system (Based 2)*
- Computer works under the command of Human. Hence, Computer must understand the Decimal system meanwhile still using Binary system to operate
- Modern computer works not only using Binary system but also several related binary system, such as: Octal system, Hexadecimal system
- Computer needs to learn how to maintain the relationship between these system → *A rule or standard in Number representing*
- Computer does not handle only the numbers but also the characters → *A rule or standard in Character representing*
- *Human needs to choose the ideal method to teach the computer maintains and processes the binary data* → SWOT (Strength, Weakness, Opportunities and Threats) in various standards

Simple number vs Complex number

- To represent the number in computer, *human classifies two types of number:*
  - Simple number, which be called Fixed point Number in scientific.
  - Complex number, which be called Floating point Number in scientific.
- ***Fixed point Number*** means *the point or the indicator is fixed* compare to the number of digits in real part (left part) and number of digits in fraction part (right part)
- ***Floating point Number*** means *the point or the indicator could be changed* compare to the number of digits in real part (left part) and number of digits in fraction part (right part).
- Most of the modern computer systems have a mechanism to maintain the Floating point Number to resolve complex calculations.
- Whatever the number is, the computer system must supply some resources to storage it.
  - This resource is measured by the number of bits
  - More bits were used, more accuracies in number representation

Fixed Point Number: Dec, Bin, Hex

- From human perspective, to convert from Decimal system to Binary system, **Division Method** is proposed for Unsigned Integer and Real number.
- Division method could be used to convert from Decimal system to not only Binary system but also any Radix system.
  - The calculation stops when the result of division is 0
  - For the Real part, the remainder (0,1) should be taken from the bottom to the top of calculation progress.
- **Multiply Method** could be used to convert the Fraction part of Real number
  - The calculation could be unstoppable meanwhile resource to represent number is limited → only take enough number of bits.
- Example:
  - Convert from 191 (10) to (2) system
  - Convert from 171.15 (10) to (2) system
  - Convert from 317.37 (10) to (2) system

Fixed Point Number: Dec, Bin, Hex

- *Any radix system could be represent easily into Decimal system*
- Example:
  - 6789 (10) → $6 \times 10^3 + 7 \times 10^2 + 8 \times 10^1 + 9 \times 10^0$
  - 10111 (2) → $1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
  - 1110.01 (2) → $1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$
  - 210012 (3) → $2 \times 3^5 + 1 \times 3^4 + 0 \times 3^3 + 0 \times 3^2 + 1 \times 3^1 + 2 \times 3^0$
- It is difficult to read long strings of binary number, i.e. 100101010111 → binary values are usually expressed using the hexadecimal system (16) by grouping the binary digits into groups of four
- Example: 11010100011011 (2) = 13595 (10)

0011	0101	0001	1011
3	5	1	B

Fixed point Number: Signed number

- To represent negative values, computer systems must have a way to identify the sign of a value. Normally, most of computer system use the **high-order bit** (left most bit, most significant bit) as the sign identifier of number.
- There are 4 ways in which signed binary numbers may be expressed:
  - Signed magnitude
  - One's complement
  - Two's complement
  - Exceeded

Fixed point Number: Signed Magnitude

- *How to convert from Dec to Signed Magnitude*
- *How to convert from Signed Magnitude to Dec*
- Example:
  - Using 4 bits to represent -7: 1111
  - Using 5 bits to represent -20: impossible
  - 10101 → ?, 01101 → ?
- Advantage: easy to understand and implement
- Disadvantage:
  - *Requires complicated computer hardware*
  - T*wo different representations for zero*: +0 and -0
  - Does not work well in computation, i.e. Adding
- Performing arithmetic operations: *ignore the signs of the operands while performing a calculation, applying the appropriate sign after the calculation is complete*

Fixed point Number: Signed Magnitude

Operation	ADD Magnitudes	SUBTRACT Magnitudes		
		A > B	A < B	A = B
(+A) + (+B)	+ (A + B)			
(+A) + (-B)		+ (A – B)	- (B – A)	+ (A – B)
(-A) + (+B)		- (A – B)	+ (B – A)	+ (A – B)
(-A) + (-B)	- (A + B)			
(+A) - (+B)		+ (A – B)	- (B – A)	+ (A – B)
(+A) - (-B)	+ (A + B)			
(-A) - (+B)	- (A + B)			
(-A) - (-B)		- (A – B)	+ (B – A)	+ (A – B)

- Normal case: find the sum of 75 and 46
- Abnormal case: find the sum of 107 and 46

Fixed point Number: 1's complement

- Signed and Magnitude to represent the fixed point number is used in earliest computer system, i.e. IBM 7090 (2nd generation).
- This method is still useful in representing of the significand in floating point number. The modern computer system uses diminished radix complement in which, a negative value is given by the difference between the absolute value of a number and one less than its base.
- ***One's complement***:
  - *Flipping the bits of a binary number = bitwise NOT*
  - *Sign is indicated by the left most bit*
  - T*wo different representations for zero*: +0 and -0
- Performing arithmetic operations: *using end around carry*
  - Example: Addition of -1 (8 bits) and +2 (8 bits)
  - *End around carry* is complex but still simpler than dedicated hardware used for arithmetic operations in signed magnitude.
  - Erroneous caused by overflow:
    - If the sum of two positive numbers yields a negative result, the sum has overflowed.
    - If the sum of two negative numbers yields a positive result, the sum has overflowed

Fixed point Number: 2's complement

- The problems of multiple representations of 0 and the need for the end around carry are circumvented by a system called **two's complement**
- **Two's complement**:
  - *Find the one's complement* of the number and *then add 1*.
  - *Sign is indicated by the left most bit*
  - Unique *representation for zero*
- Performing arithmetic operations:
  - Example: Addition of -1 (8 bits) and +2 (8 bits)
  - *End around carry is just discarded*
  - Erroneous caused by overflow:
    - If the sum of two positive numbers yields a negative result, the sum has overflowed.
    - If the sum of two negative numbers yields a positive result, the sum has overflowed
  - Another way to detect overflow: **When the "carry in" and the "carry out" of the sign bit differ, overflow has occurred.**
  - 2's complement is very useful but limited in multiplication ➔ **Booth's algorithm**

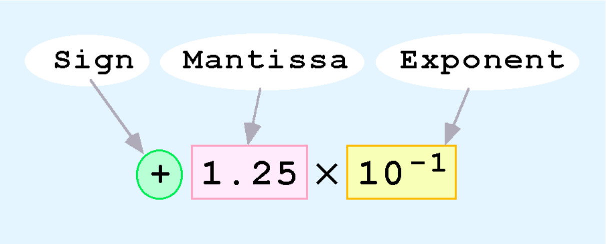
Fixed point vs Floating point

- The *signed magnitude*, *1's complement*, and *2's complement* representations are *fixed point representation* or *fixed point notation.*
- Fixed point number has radix point fixed

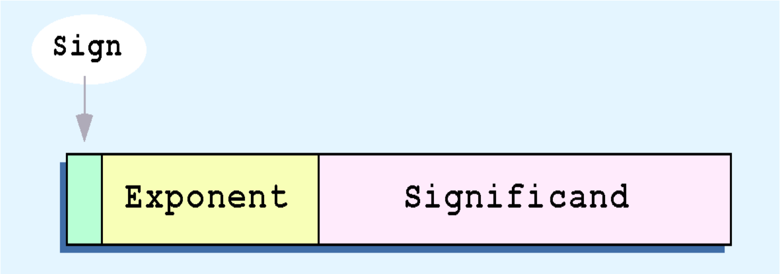
- Advantage: the performance in integer arithmetic operations.
- Disadvantage: limited range of values can be represented → not useful in scientific or business applications that deal with real number values over a wide range
- Floating point number takes advantage of scientific or business applications by using flexible the range, the accuracy and the precision.
  - The range: how difference between largest and smallest values that can express.
  - The accuracy: how closely a numeric representation approximates a true value.
  - The precision: how much information about a value

Floating point Number: Scientific notation

- Most modern computers use Floating point number when storing fractional numbers. It could represent very large or very small numbers precisely using *scientific notation* in binary.



- Although there are several ways to represent floating point number, a floating-point number consists of three fixed-size fields:



The one-bit **sign field** is *the sign* of the stored value.

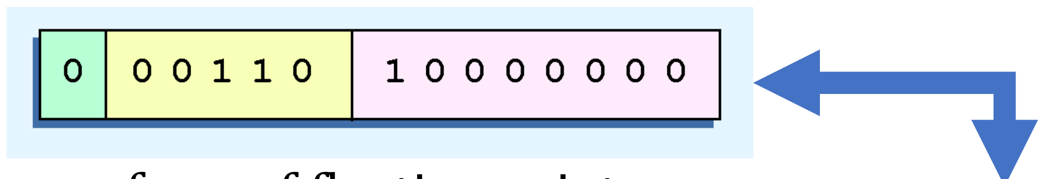
The size of the **exponent field**, determines *the range* of values that can be represented.

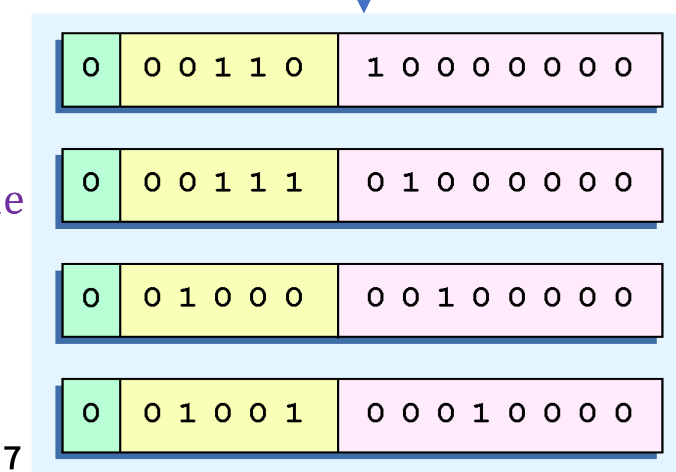
The size of the **significand field** determines *the precision* of the representation.

Floating point Number: A typical model

- Example: Express 32_{10} in the simplified 14-bit floating-point model
  - Unique form to represent: $0.f \times 2^n$
  - The significand is preceded by *an implied bit 0*
  - The exponent is 5 bits; the fraction is 8 bits
  - Scientific notation of 32: $1.0 \times 2^5 = 0.1 \times 2^6$
  - Put 110 (= 6_{10}) in the exponent field and 1 in the significand as shown

32





- Problem 1: Synonymous form of floating point representations.
  - Solution: A unique pattern (a rule) to represent floating point number and method to normalized the number, i.e. IEEE 754
- Problem 2: Negative exponents: allowed or not allowed
  - Solution: Using biased representation, i.e. bias 127

Floating point Number: Bias representation

- Bias representation also referred to as **excess K**, **excess *N***, **excess code or Offset binary** is a digital coding scheme where the minimal value is 0 → *Bias representation is unsigned*
- The exponent is stored as an unsigned value suitable for comparison floating point number → *Exponent bias*
- Value of unbiased (actual) exponent could be calculated from the value of the bias exponent
- Example:
  - A floating point number uses an actual exponent with 5 bits → Range to represent the value: [-16;+15] in 2's complement
  - Then, we choose 16 for our new (bias) exponent → *Excess-16* representation
  - With this bias exponent, range of floating point number changed to [0;+31]
- *There is no standard for bias representation, but most often the bias for an n-bit binary word is 2^{n-1}*

Floating point Number: Bias representation

- **Example**: Express 32_{10} in the revised 14-bit floating-point model using 16 bias exponent (excess 16)

0	1 0 1 1 0	1 0 0 0 0 0 0 0

  - $32 = 1.0 \times 2^5 = 0.1 \times 2^6$
  - Unbiased exponent 6 → Biased exponent: $16 + 6 = 22_{10} = 10110_2$

- **Example**: Express 0.0625_{10} in the revised 14-bit floating-point model using 16 bias exponent.

0	0 1 1 0 1	1 0 0 0 0 0 0 0

  - $0.0625 = 1.0 \times 2^{-4} = 0.1 \times 2^{-3}$
  - Unbiased exponent -3 → Biased exponent: $16 + (-3) = 13_{10} = 01101_2$.

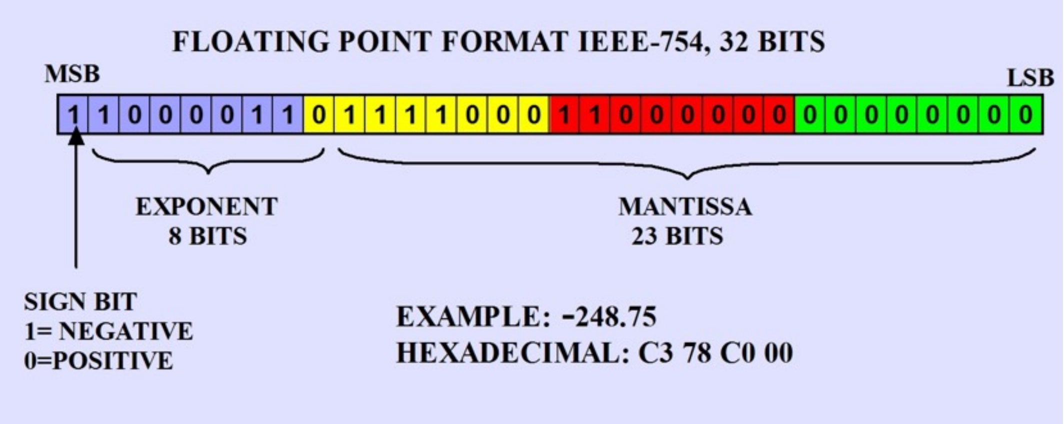
- **Example**: Express -26.625_{10} in the revised 14-bit floating-point model using 16 bias exponent.

1	1 0 1 0 1	1 1 0 1 0 1 0 1

  - $26.625_{10} = 11010.101_2 = 0.11010101 \times 2^5$.
  - Unbiased exponent 5 → Biased exponent: $16 + 5 = 21_{10} = 10101_2$

Floating point Number: IEEE 754

- The IEEE 754 **single precision** has 8 bits for bias exponent (K=127) and 23-bit for significand (mantissa)



- The IEEE 754 **double precision** has 11 bits for bias exponent (K=1023) and 52 bits for significand (mantissa)

Floating Point Range

	Denormalized	Normalized	Approximate Decimal
Single Precision	$\pm 2^{-149}$ to $(1-2^{-23})\times 2^{-126}$	$\pm 2^{-126}$ to $(2-2^{-23})\times 2^{127}$	$\pm \approx 10^{-44.85}$ to $\approx 10^{38.53}$
Double Precision	$\pm 2^{-1074}$ to $(1-2^{-52})\times 2^{-1022}$	$\pm 2^{-1022}$ to $(2-2^{-52})\times 2^{1023}$	$\pm \approx 10^{-323.3}$ to $\approx 10^{308.3}$

Floating point Number: IEEE 754

- The biased exponent is 255 (2047 for double precision)
  - the significand is 0 → the value is $\pm \infty$
  - the significand is not 0 → the value is NaN - Not A Number, used to flag an error condition
- Two represented Zero (-0 and +0) which is indicated by all zeros in the exponent and the significand.
  - Avoid a floating-point value = 0
  - +0 is not equal -0
- The models for floating point representation has finite number of bits
  - Problem 1: the model can give only an approximation of a real value
  - Problem 2: errors could appear in calculations --> Solutions: using greater number of bits, aware of the possible magnitude of error in calculations (Arithmetic Operation)
  - Problem 3: overflow/underflow → crashed program
  - Problem 4: arithmetic operations could be NOT distributive → https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html

Binary Coded Decimal - BCD

- Comparing to the binary positional system, BCD′s main virtue is its more accurate representation and rounding of decimal quantities as well as an ease of conversion into human-readable representations.
- **BCD** is a class of binary encodings of decimal numbers
  - Each decimal digit is represented by a fixed number of bits, i.e. 4 or 8 bits
  - Special bit patterns are sometimes used for a sign or for other indications (e.g., error or overflow)
- Main advantages:
  - Easy to encode and decode decimals into BCD and vice versa
  - Simple to implement a hardware algorithm for the BCD converter
  - Useful in digital systems whenever decimal information is given either as inputs or displayed as outputs
  - Digital voltmeters, frequency converters and digital clocks all use BCD to display output information in decimal

Binary Coded Decimal - BCD

- Disadvantages:
  - BCD code for a given decimal number requires more bits than the straight binary code
  - Limited to represent the BCD form in high speed digital computers
  - The arithmetic operations require a complex design
  - The speed of the arithmetic operations is naturally slow due
- There are two main types of BCD used in modern computer (byte oriented system):
  - Unpacked BCD: implies full byte for each digit, including a sign
  - Packed BCD: encodes two decimal digits within a single byte
- In the past, BCD was used in the instruction set of early decimal computer: IBM System/360, Digital Equipment Corporation's VAX, Motorola 68000-series processors
- Recently, BCD is still used in financial, commercial, and industrial computing, where subtle conversion and fractional rounding errors that are inherent in floating point binary representations cannot be tolerated

Binary Coded Decimal - Packed BCD

- How to represent a decimal number using BCD
  - Use BCD table to find binary pattern for each decimal digits
  - Adding 4 bits pattern for the sign of the number
    - Positive (+): 0000
    - Negative (-): 1001 (9's complement of +)
  - Negative decimal number is 10's complement of Positive decimal number
  - 10's complement = 9's complement + 1
- Example: 1357, -1357

Dec digit	BCD converting				Dec digit	BCD converting			
	d3	d2	d1	d0		d3	d2	d1	d0
0	0	0	0	0	8	1	0	0	0
1	0	0	0	1	9	1	0	0	1
2	0	0	1	0	A	1	0	1	0
3	0	0	1	1	B	1	0	1	1
4	0	1	0	0	C	1	1	0	0
5	0	1	0	1	D	1	1	0	1
6	0	1	1	0	E	1	1	1	0
7	0	1	1	1	F	1	1	1	1

Character Codes

- Early computer manufacturers chose the 7-bit ASCII (American Standard Code for Information Interchange)
  - While BCD and EBCDIC were based upon punched card codes, ASCII was based upon telecommunications (Telex) codes.
  - Until recently, ASCII was the dominant character code outside the IBM mainframe world.
- Many of today's systems embrace Unicode, a 16-bit system that can encode the characters of every language in the world.
- According to evolution of computers, character codes have also evolved and richer, more in: https://en.wikipedia.org/wiki/Character_encoding#Common_character_encodings

Character Types	Language	Number of Characters	Hexadecimal Values
Alphabets	Latin, Greek, Cyrillic, etc.	8192	0000 to 1FFF
Symbols	Dingbats, Mathematical, etc.	4096	2000 to 2FFF
CJK	Chinese, Japanese, and Korean phonetic symbols and punctuation.	4096	3000 to 3FFF
Han	Unified Chinese, Japanese, and Korean	40,960	4000 to DFFF
	Han Expansion	4096	E000 to EFFF
User Defined		4095	F000 to FFFE

Conclusion

- Computer systems have used *binary for data representation* based on binary advantages
- Whatever the *methods used in binary representation*, we should analyze the *SWOT each of them* carefully to find the appropriate method for modern computer system
- The number representation could be divided into 2 parts: *Simple number (Fixed point number) and Complex number (Floating point number)*
- For simple number, there are several ways to maintain the unsigned value: *Sign and Magnitude, 1's complement, 2's complement, Excess N*
- For complex number, *IEEE 754* is the best standard for arithmetic operation in high speed modern computer
- *Binary coded Decimal BCD* is an another way to represent decimal number. This model has a stand in financial, commercial, and industrial computing
- Besides number representation, computer system always implemented vary *character codes*, such as: ASCII, Unicode

Question?