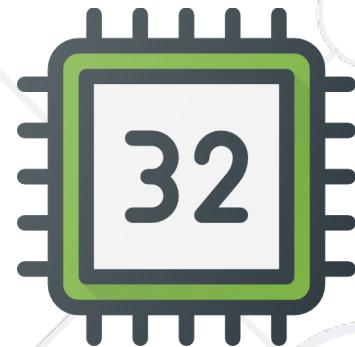


Bits and Bitwise Operations

Bits, Numerals Systems and Bitwise Operations



SoftUni Team
Technical Trainers



$$\begin{array}{r} x \ 0011 \\ y \ 0101 \\ \hline x \& y \ 0001 \end{array}$$

$$\begin{array}{r} x \ 0011 \\ y \ 0101 \\ \hline x | y \ 0111 \end{array}$$

$$\begin{array}{r} x \ 0011 \\ y \ 0101 \\ \hline x \wedge y \ 0110 \end{array}$$



SoftUni

Software University
<https://softuni.bg>

Table of Contents

- What is a **Bit, Byte, KB, MB?**
- **Numerals Systems**
 - Decimal, Binary, Hexadecimal
 - Conversion between Numeral Systems
- **Representation of Data** in Computer Memory
 - Representing Integers, Real Numbers and Text
- **Bitwise Operations: &, |, ^, ~**
 - Reading / Writing Bits from Integers

$$\begin{array}{r} x \boxed{0} \boxed{0} \boxed{1} \boxed{1} \\ y \boxed{0} \boxed{1} \boxed{0} \boxed{1} \\ \hline x \& y \boxed{0} \boxed{0} \boxed{0} \boxed{1} \end{array}$$

$$\begin{array}{r} x \boxed{0} \boxed{0} \boxed{1} \boxed{1} \\ y \boxed{0} \boxed{1} \boxed{0} \boxed{1} \\ \hline x | y \boxed{0} \boxed{1} \boxed{1} \boxed{1} \end{array}$$

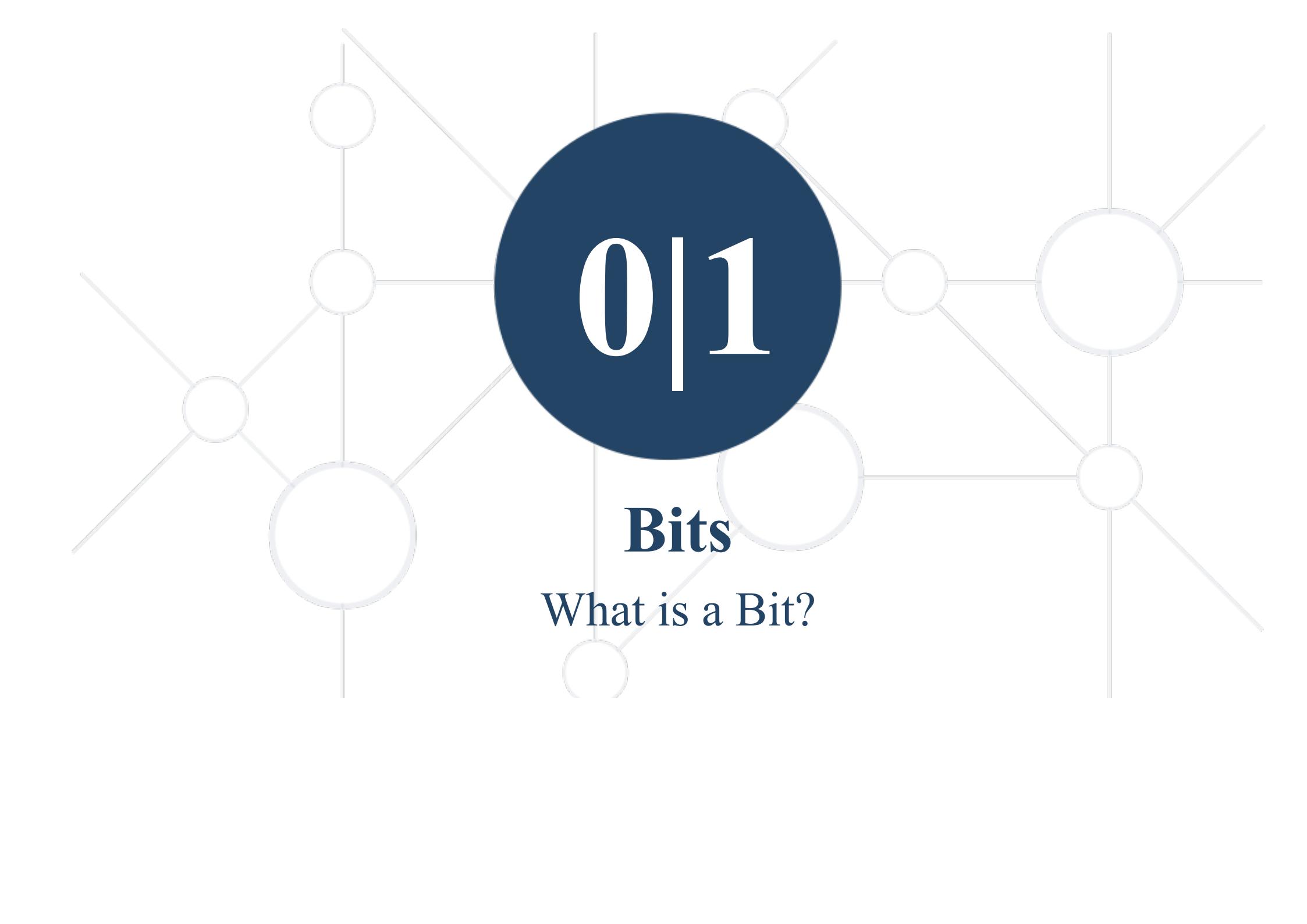
$$\begin{array}{r} x \boxed{0} \boxed{0} \boxed{1} \boxed{1} \\ y \boxed{0} \boxed{1} \boxed{0} \boxed{1} \\ \hline x \wedge y \boxed{0} \boxed{1} \boxed{1} \boxed{0} \end{array}$$

Have a Question?



sli.do

#fund-common



0|1

Bits

What is a Bit?

Bit

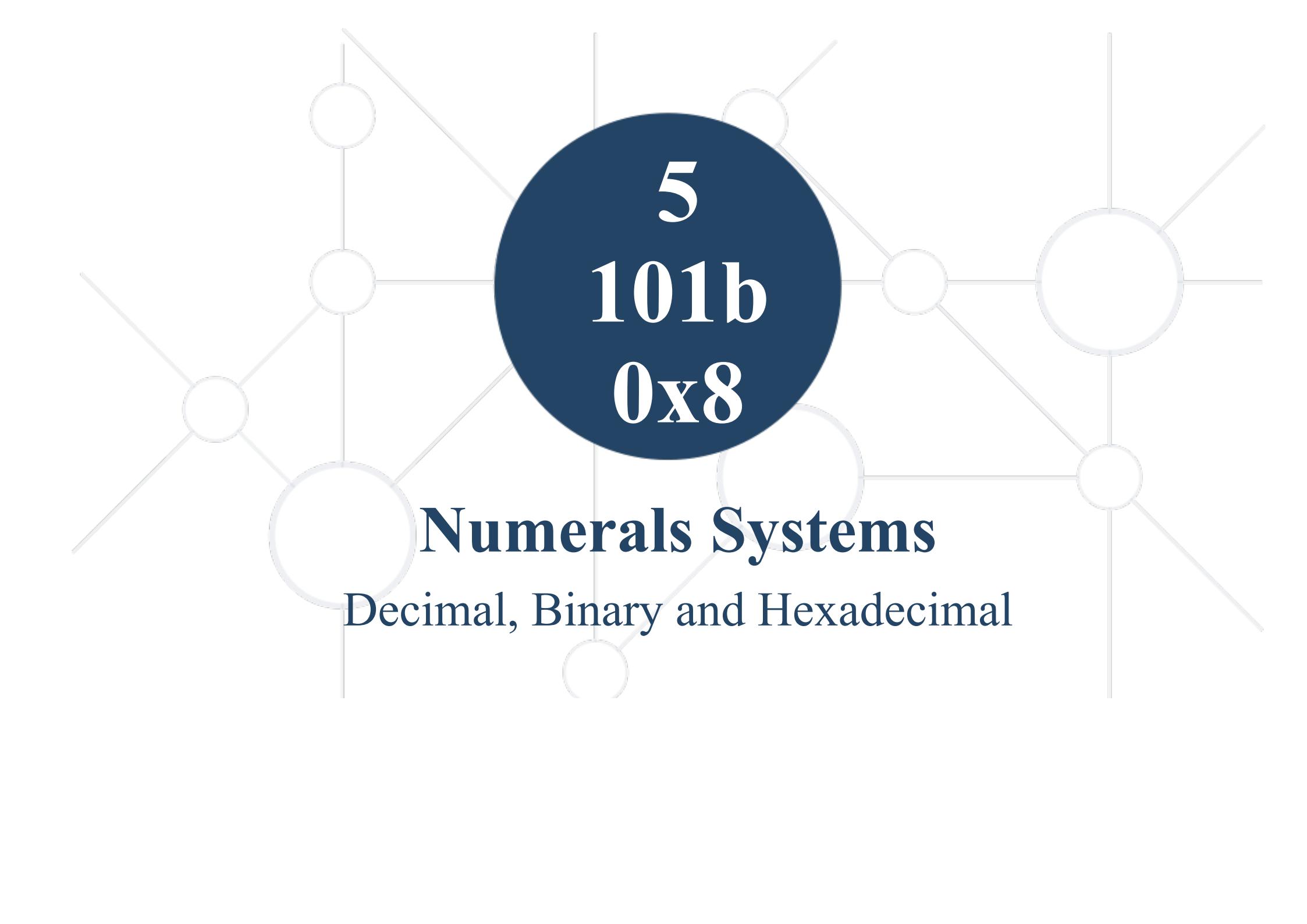


- Bit == the smallest **unit of data used in computing**
 - Takes only one of **two values**: either a **0** or **1**
- **1 bit can store anything with two separate states**
 - Logical values (true / false)
 - Algebraic signs (+ / -)
 - Activation states (on / off)
- Bits are organized in computer memory in sequences of **8 bits**, called **bytes (octets)**

Bit, Byte, KB, MB, GB, TB, PB



- **Bit** – single 0 or 1, representing a bit of data
- **Byte (octet)** == **8 bits** == the smallest addressable unit in the computer memory
- **KB** (kilobyte) == **1024 bytes** (sometimes 1000 bytes)
- **MB** (megabyte) == **1024 KB** == **1048576 bytes**
- **GB** (gigabyte) == **1024 MB** == **1073741824 bytes**
- **TB** (terabyte) == **1024 GB** == **1099511627776 bytes**
- **PB** (petabyte) == **1024 TB** == **1125899906842624 bytes**



5
101b
0x8

Numerals Systems

Decimal, Binary and Hexadecimal

Numerical Systems

- Numeral system == system for **representing numbers** in written form using sequence of **digits**
- **Positional numeral systems** == the value of each digit depends on its position
 - These numeral systems has a **base** (e.g. 2, 10, 16)

Decimal (base = 10)	Binary (base = 2)	Hexadecimal (base = 16)
30	111110	1E
45	101101	2D
60	111100	3C



Decimal Numbers

- Decimal numbers (**base 10**)
- Represented using 10 **digits**:
 - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Each position represents a **power of 10**

$$\begin{aligned}401 &= 4 \cdot 10^2 + 0 \cdot 10^1 + 1 \cdot 10^0 = \\&= 4 \cdot 100 + 0 \cdot 10 + 1 \cdot 1 = \\&= 400 + 0 + 1 = 401\end{aligned}$$

- A decimal number **$d_{n-1}d_{n-2}\dots d_1d_0$** =
 $d_0 \cdot 10^0 + d_1 \cdot 10^1 + d_2 \cdot 10^2 + \dots + d_{n-1} \cdot 10^{n-1}$

Binary Numbers

- The **binary system** is used in computer systems
- Binary numbers (**base 2**)
 - Represented by **sequence of 0 or 1**
- Each position represents a **power of 2**

$$5 == 101b$$

$$101b = 1*2^2 + 0*2^1 + 1*2^0 = 4 + 0 + 1 = 5$$

$$\begin{aligned}1010b &= 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 = \\&8 + 0 + 2 + 0 = 10\end{aligned}$$

Binary and Decimal Conversion

- Binary to decimal

- Multiply each digit to its magnitude (power of 2)

$$\begin{aligned}1011_b &= 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = \\&= 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = \\&= 8 + 0 + 2 + 1 = \\&= 11\end{aligned}$$

- Decimal to binary

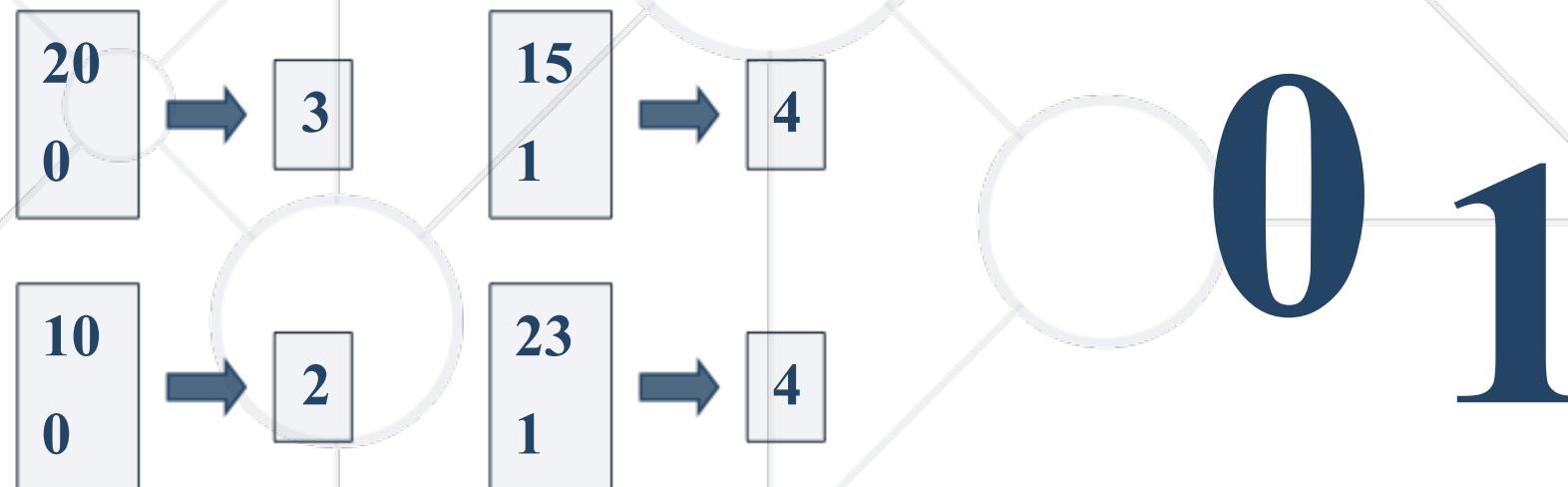
- Divide to the base (2) until 0 is reached and take the reminders in reversed order

$$\begin{aligned}11 / 2 &= 5 (1) // \text{last digit} \\5 / 2 &= 2 (1) // \text{previous digit} \\2 / 2 &= 1 (0) // \text{previous digit} \\1 / 2 &= 0 (1) // \text{first digit}\end{aligned}$$

Result: 1011

Problem: Binary Digits Count

- You are given a positive integer **n** and a binary digit **b** (0 or 1)
- Write a program that finds the count of **b** digits in the binary representation of **n**



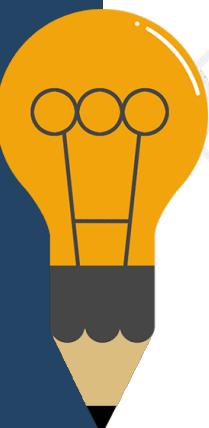
Solution: Binary Digits Count

- . Read the **input** from the user: **n** and **b**
 - . Convert the **input to binary system**
(collect the reminders of division by 2)
 - . Count the digits **b** in the reminders of **n**
 - . Print the **count**
-
- Another solution is to use **bitwise operations**
(think how later)

Hexadecimal Numbers

- Hexadecimal numbers (**base 16**)
 - Represented using **16 literals** (hex digits)
 - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F
 - Usually **prefixed with 0x** in computer science
 - Each position represents a **power of 16**

$$\begin{aligned}0xB7F6 &= B*16^3 + 7*16^2 + F*16^1 + 6*16^0 = \\&= 11*4096 + 7*256 + 15*16 + 6*1 = \\&= 45056 + 1792 + 240 + 6 = 47094\end{aligned}$$



Hex \leftrightarrow Decimal Conversions

- Hexadecimal to decimal

- Multiply each digit to its weight (power of 16)

$$\begin{aligned}0x1F4 &= 1*16^2 + 15*16^1 + 4*16^0 = \\&= 1*256 + 15*16 + 4*1 = \\&= 256 + 240 + 4 = \\&= 500\end{aligned}$$

- Decimal to hexadecimal

- Divide by 16 and take the reminders in reversed order

$$\begin{aligned}500 / 16 &= 31 \text{ (4)} \\31 / 16 &= 1 \text{ (F)} \\1 / 16 &= 0 \text{ (1)} \\ \text{Result: } 0x1F4\end{aligned}$$



Hex \leftrightarrow Binary Conversions

- The conversion from **binary** to **hexadecimal** (and back) is straightforward
 - Each hex digit corresponds to a **sequence of 4 binary digits**

A2E3F = 1010 0010 1110 0011 1111

A = 1010

2 = 0010

E = 1110

3 = 0011

F = 1111

1010 0010 1110 0011 1111 = A2E3F

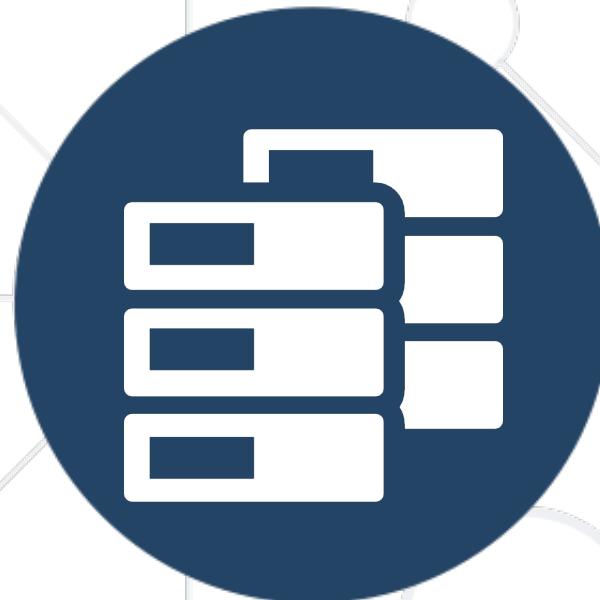
1010b = 10dec = Ahex

0010b = 2dec = 2hex

1110b = 14dec = Ehex

0011b = 3dec = 3hex

1111b = 15dec = Fhex



Representation of Data

Integers, Floating-Point Numbers and Text

Representing Integers in Memory

- Integer numbers are sequences of bits
- Can be signed (in most cases) or unsigned
 - The sign == the **Most Significant Bit (MSB)**
 - Leading 0 □ **positive number**
 - Leading 1 □ **negative number**
- Example (8-bit signed integers)

```
0XXXXXXXb > 0 // 00010010b = 18
```

```
00000000b = 0
```

```
1XXXXXXXb < 0 // 10010010b = -110
```



Representation of Signed Integers

- Positive 8-bit numbers have the format **0XXXXXXX**
 - The value is the decimal value of their last **7 bits (XXXXXXX)**
- Negative 8-bit numbers have the format **1YYYYYYY**
 - The value is **-128 (-27) + the decimal value of YYYYYYYY**

$$\begin{aligned}10010010b &= -27 + 0010010b = \\&= -128 + 18 = \\&= -110\end{aligned}$$

-27

Largest and Smallest Signed Integers

- The largest signed 8-bit integer is:

$$127 = (2^7 - 1) = \text{0111111b}$$

$2^7 - 1$

- The smallest negative 8-bit integer is:

$$-128 = -(2^7) = \text{10000000b}$$

-2^7

- The largest signed 32-bit integer is:

$$2147483647 = (2^{31} - 1) = \text{0111...111b}$$

$2^{31} - 1$

- The smallest negative 32-bit integer is:

$$-2147483648 = -(2^{31}) = \text{1000...000b}$$

-2^{31}

Integers and Their Ranges in Programming

Bits	Sign	Range	Data Types
8-bit	signed	-128 ... 127 (-2 ⁷ ... 2 ⁷ -1)	sbyte in C#, byte in Java
8-bit	unsigned	0 ... 255 (2 ⁰ ... 2 ⁸ -1)	byte in C#
16-bit	signed	-32768 ... 32767 (-2 ¹⁵ ... 2 ¹⁵ -1)	short in C#, short in Java
32-bit	signed	-2,147,483,648 ... 2,147,483,647 (-2 ³¹ ... 2 ³¹ -1)	int in C#, int in Java

Representing Real Numbers



- Computers use the **floating-point number** format, defined by the **IEEE 754 technical standard**
- The **IEEE-754** standard defines:
 - Arithmetic and exchange formats – representations of the binary and decimal floating-point data
 - Rounding rules for floating-point numbers
 - Operations – arithmetic and other operations
 - Special numbers – such as **infinity** and **NaN**

Storing Floating-Point Numbers

- Floating-point numbers are stored as sequence of bits:
sign bit, exponent and mantissa

32
Bits

1 10000011 010100101000000000000000 ≈ -21.15625

Sig Exponen

Mantiss

Play with the **IEEE-754 converter**

online:

<https://h-schmidt.net/FloatConverter/IEEE754.html>

- Note: **errors in calculations** and **precision** may occur
 - Some numbers (e.g. 0.3) cannot be represented in the above format without rounding (as a sum of negative powers of 2)

Representing Text

- Computers represent **text characters** as unsigned integer numbers (i.e. as sequence of bits)
 - Letters, digits, punctuation chars, etc.
- The **ASCII** standard represent chars as **8-bit integers**
- Defines the **ASCII code** for 127 chars, e.g.

Binary	Dec	Hex	Char
0b01000001	65	0x41	A
0b01000010	66	0x42	B
0b00101011	43	0x2B	+

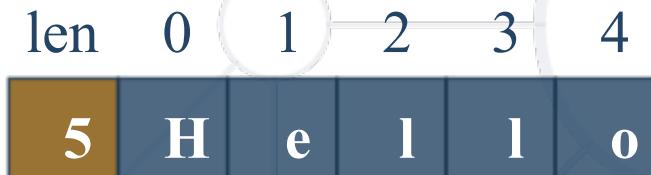
Representing Unicode Text

- The **Unicode** standard represents 100,000+ text characters as **16-bit integers** (see unicode.org)
 - Supports many alphabets, e.g. Latin, Cyrillic, Arabic

Decimal	Hex	Char	Explanation
65	0x0041	A	Latin "A"
1097	0x0449	Щ	Cyrillic letter "Sht"
1576	0x0628	ٻ	Arabic letter "Beh"
127928	0x1F3B8		Emoji "Guitar"

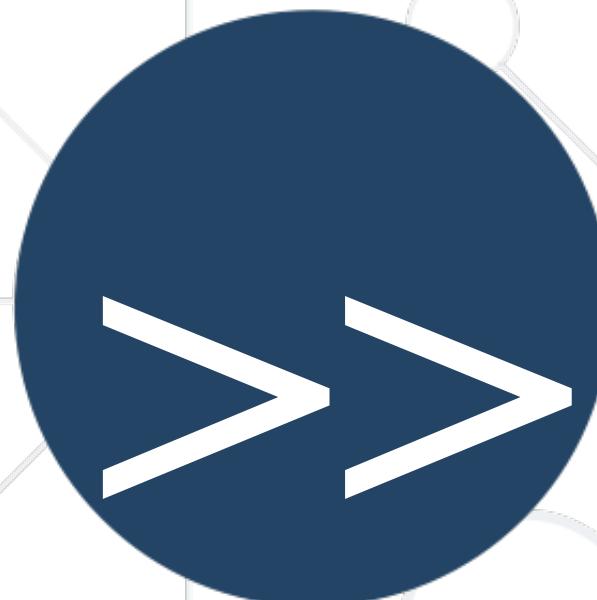
- UTF-16 uses 2 bytes (16 bits) for each char
- UTF-8 uses 1, 2, 3 or 4 bytes for each char

Sequences of Characters

- **Strings** represent **text data** in programming
 - **Strings** are **arrays of characters**, typically represented like this:

The diagram illustrates the memory representation of the string "Hello". It shows a horizontal array of six cells. The first cell, labeled "len", contains the value 5. The subsequent five cells contain the characters H, e, l, l, and o respectively. Above the array, indices 0 through 4 are shown, with arrows pointing from each index to its corresponding cell. The cell for index 5 is empty.

Takes **14 bytes** in memory:
 $4 \text{ bytes (length)} + 5 * 2 \text{ bytes}$
 - The string can have its **size as prefix** (used in most languages) or can end with **\0** (null-terminated string – used in C)
 - Characters in the string can be:
 - **16-bit (UTF-16)** – default in C#, Java, JS, Python
 - **8-bit (ASCII / windows-1251)** – default in C, C++

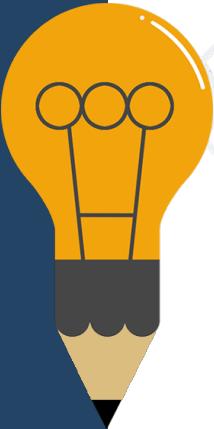


Bitwise Operations

Bitwise Operators and Bit Shifts

Bitwise Operators

- Bitwise operators works with the binary representations of the numbers, applying bit by bit calculations
- The operator `~` turns all `0` to `1` and all `1` to `0` (like `!` for boolean expressions but bit by bit)
- The operators `|`, `&` and `^` behave like `||`, `&&` and `^` for boolean expressions but bit by bit



Operator					&	&	&	&	^	^	^	^
Operand 1	0	0	1	1	0	0	1	1	0	0	1	1
Operand2	0	1	0	1	0	1	0	1	0	1	0	1
Result	0	1	1	1	0	0	0	1	0	1	1	0

Bitwise Operators – Examples

- Bitwise NOT (\sim)

```
5 // 0101  
 $\sim$ 5 // 1010
```

- Bitwise AND

```
( $\&$ ) 5 // 0101  
3 // 0011  
5 & 3 // 0001
```

- Bitwise OR ($\|$)

```
5 // 0101  
3 // 0011  
5 | 3 // 0111
```

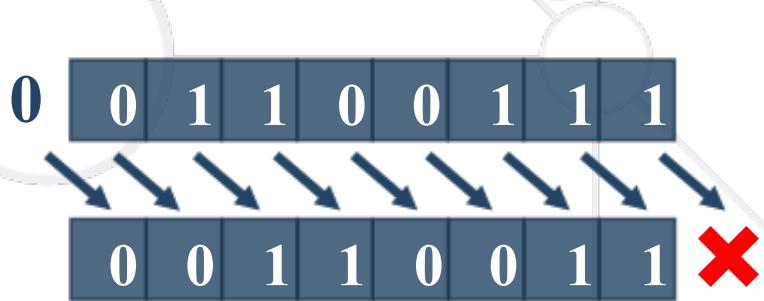
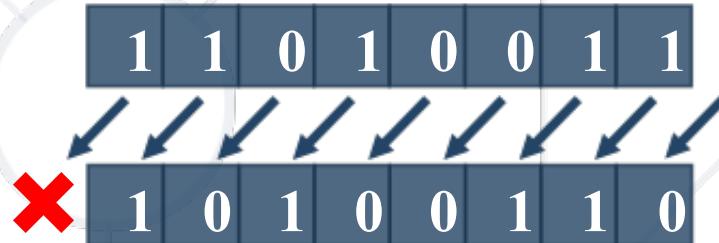
- Bitwise XOR ($^$)

```
5 // 0101  
3 // 0011  
5 ^ 3 // 0110
```

Bit Shifts



- Bit shifts are bitwise operations, where
 - Bits are moved (**shifted**) to the **left** or **right**
 - The bits that fall outside the number are **lost** and **replaced by 0**
- Left shift (<< operator)
- Right shift (>> operator)



Bitwise Operations: Get the Last Bit

- How to get the last bit from a number **n**?
 - The bits are numbered from 0, from right to the left
 - The position of the last (rightmost) bit is 0

```
n = 125 // 01111101
```

```
mask = 1 // & 00000001
```

```
n & mask // 00000001 = 1
```

n =

7	6	5	4	3	2	1	0
0	1	1	1	1	1	0	1

lastBit = 1

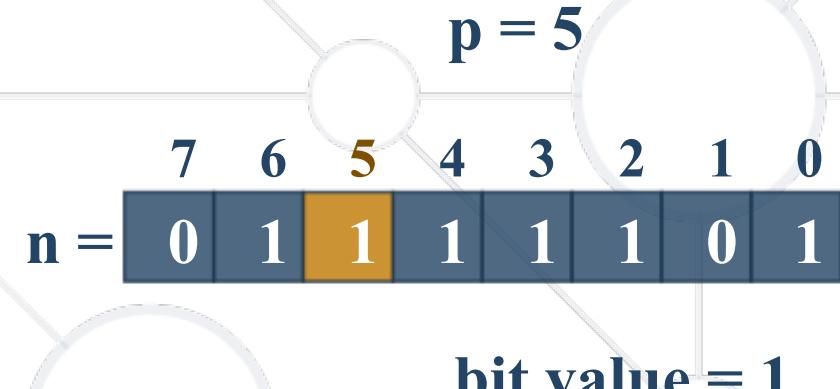
- Last bit – formula:

```
lastBit = n & 1
```

Bitwise Operations: Get Bit at Position

- How to get the bit at position **p** from a number **n**?

```
n = 125 // 01111101  
p = 5   // 5th position  
125 >> p // 00000011 = 3  
3 & 1   // 00000001 = 1
```



- Bit at position – formula:

```
bit = (n >> p) & 1
```

Bitwise Operations: Set Bit at Position

- How to set the bit at given position **p** to 0 or 1?

- Clear a bit (0) at position **p**

```
p = 5          // 5th position  
n = 125        // 01111101  
mask = ~(1 << p) // 11011111  
result = n & mask // 01011101
```

- Set a bit (1) at position **p**

```
p = 5          // 5th position  
n = 125        // 01111101  
mask = 1 << p  // 00100000  
result = n | mask // 01111101
```

- Assign a bit **b** (0 or 1) at position **p** – formula:

```
n = n & ~(1 << p) | (b << p)
```

Why We Need Bitwise Operations?

- **Networking protocols**
 - Many devices communicate using bit-level protocols
 - E.g. the SYN flag in the **TCP protocol** header is the bit #1 from the 14th byte in the TCP packets
 - Web browsers use bitwise operations to connect to a Web site
- Many **binary file formats** use bits to save space
 - E.g. PNG images use 3 bits to specify the color format used
- **Data compression** replaces byte sequences with bit sequences
 - E.g. the DEFLATE algorithm in **ZIP files**

Problem: Bit #1 (the Bit Before the Last)

- Write a program that prints the bit at position 1 of an integer

51

1

51 == 0 0 1 1 0 0 1 1

13

0

13 == 0 0 0 0 1 1 0 1

24

0

24 == 0 0 0 1 1 0 0 0

2

1

2 == 0 0 0 0 0 0 1 0

- Solution:

```
p = 1      // 1st position  
n = 51     // 00110011  
n = n >> p // 00011001 = 25  
n & 1      // 1
```



Live Exercises

Computers store data using **bits**

Signed **integers** (leftmost bit == sign)

IEEE-754 – floating point numbers

Text is stored using ASCII / Unicode / other

Binary and **hexadecimal numeral systems**

play a key role in computing

Developers manipulate **bits** in integers using
bitwise operators and **bit masks**



Questions?



© SoftUni – <https://softuni.org>. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

SoftUni Diamond Partners



SoftUni Organizational Partners



Trainings @ Software University (SoftUni)



- Software University – High-Quality Education, Profession and Job for Software Developers
 - softuni.bg, softuni.org
- Software University Foundation
 - softuni.foundation
- Software University @ Facebook
 - facebook.com/SoftwareUniversity
- Software University Forums
 - forum.softuni.bg



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://softuni.org>
- © Software University – <https://softuni.bg>

