Badji-Mokhtar University. Annaba
Faculty of Technology
Department of Computer Science

Chapter 3: Data representation

Dr. Khelifi Hakima

S1- 2025/2026

Lessons Objectives

- ✓ Introduce the binary code (Natural, Gray, BCD)
- ✓ Understand how data is represented in a computer system
    - Integer representation (Unsigned, Signed)
    - Real numbers (Fixed-Point Representation, Floating-Point Representation)
    - Characters representation

Introduction

Introduction

- **Data Representation** refers to the methods used to represent data stored in a computer.
- The data stored in computer is knows as **Digital Data**.
- Computers use **binary code** to represent numbers, characters, text, computer processor instructions, or any other data.
- There are two possible states, off and on, usually symbolized by **0 and 1** from the **binary number system**.

Introduction

- The 0s and 1s used to represent digital data are referred to as binary digits — from this term we get the word **bit** that stands for binary digit.
- A **bit** is a 0 or 1 used in the digital representation of data.
- A group of eight **bits** is called a **byte.**
- The notion of **bits** and **bytes** is widely used to describe **storage capacity** and **network access speed**.
- **Kilo, mega, giga, tera,** and similar terms are used to quantify digital data.

Introduction

Bit	One binary digit
Byte	8 bits
Kilobit	1,024 or 2^{10} bits
Kilobyte	1,024 or 2^{10} bytes
Megabit	1,048,576 or 2^{20} bits
Megabyte	1,048,576 or 2^{20} bytes
Gigabit	2^{30} bits
Gigabyte	2^{30} bytes
Terabyte	2^{40} bytes
Petabyte	2^{50} bytes
Exabyte	2^{60} bytes

Introduction

- Use bits for network access speed:
  - ✓ **50 Mbps**: Megabit per second **(Mb or Mbit)** is used for faster data rates (Internet connection).
- Use bytes for storage capacities:
  - ✓ **16 GB**: Gigabyte (**GB or GByte**) is commonly used to refer to storage capacity

Binary code

Natural Binary code

DECIMAL (BASE 10)	BINARY (BASE 2)
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
1000	1111101000

Gray code

- **Gray code or reflected binary code:**
  - ✓ It is a type of binary coding system where two successive values differ in only one bit.
  - ✓ Example: 7 =$(0111)_2 = (0100)_{Gray}$ and 8 = $(1000)_2$ = $(1100)_{Gray}$
    - ✓ 7 and 8 are two successive values
    - ✓ Transition from 7 to 8: 0100 to 1100 only one bit changes
    - ✓ Transition from 7 to 8 in natural binary requires to change 4 bits

Gray code

- **Gray code or reflected binary code:**
- It is widely used in various applications especially where errors in data transmission can occur, as it minimizes the chance of multiple bit errors.

✓ **Example 1**: Send the sequence 5 → 6 → 7 using Gray code

Decimal	Binary	Gray Code
5	0101	**0111**
6	0110	**0101**
7	0111	**0100**

✓ The transmitter sends the sequence:

- 0111 (Gray code for 5)
- 0101 (Gray code for 6)
- 0100 (Gray code for 7)

Gray code

➢ **Gray code or reflected binary code:**

Example 1:

✓ The receiver receives: 0111 → 0101 → 0100

1. Compare Consecutive Gray Code Values:

   - 0111 → 0101: 1-bit change (Valid).
   - 0101 → 0100: 1-bit change (Valid).

2. Convert Gray Code Back to Binary: If no errors are detected, the receiver converts the Gray code back to binary. 0101 → 0110 → 0111 (5 →6 →7)

Gray code

➢ **Gray code or reflected binary code:**

Example 1: Suppose noise corrupts the second value (0101) into 1101.

✓ The receiver receives: 0111 → 1101 → 0100

1. Compare Consecutive Gray Code Values:

   - 0111 → 1101 : 2-bit change (Invalid, **error detected**).

2. The receiver flags an error, may request retransmission or discard the data.

Decimal	Binary	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Two successive values differ in only one bit.

Gray code

➢ **Gray code or reflected binary code:**

Example 2: Gray Code in a Digital State Machine

✓ Design a state machine to represent a 4-state sequential process using Gray code

- State A: 000, State B: 001, State C: 011, D:010

✓ Using Gray code ensures that transitions between states involve only one bit change:

- A → B: Only the last bit changes (000 → 001).
- B → C: Only the second bit changes (001 → 011).
- C → D: Only the third bit changes (011 → 010).
- D → A: Only the second bit changes (010 → 000).

✓ Only One bit changes between states. This avoid errors during state transitions.

Gray code

- **Convert Binary code to Gray Code:**
  - ✓ Let B be a number written in pure natural binary on m bits

$$B_{(2)} = B_m \ldots\ldots B_3 B_2 B_1 B_0;\ B_m \text{ is the most significant bit (MSB)}$$

  - ✓ G is the Gray code equivalent of the number B also written on m bits

$$G_{(Gray)} = G_m \ldots\ldots G_3 G_2 G_1 G_0$$

Gray code

- **Convert Binary code to Gray Code:**

- To convert a binary number to Gray code, follow these steps:

  1. The first bit of the Gray code is the same as the first bit of the binary number.

$$G_m = B_m$$

  2. Each subsequent bit in the Gray code is found by performing an XOR (Exclusively-OR) operation between the current binary bit and the previous binary bit.

$$\text{If } B_m = B_{m-1} \text{ then } G_{m-1} = 0$$

$$\text{If } B_m \neq B_{m-1} \text{ then } G_{m-1} = 1$$

Gray code

- **Convert Binary code to Gray Code:**

- **Example**: Convert 0110 (6 in decimal) to Gray code
- B = 0110 = $B_3B_2B_1B_0$

  1. G_3= B_3

  2. The second step :

$$B_3 \neq B_2 \text{ then } G_2 = 1$$

$$B_2 = B_1 \text{ then } G_1 = 0$$

$$B_1 \neq B_0 \text{ then } G_0 = 1$$

- **G= 0101**

Gray code

- **Convert code Binary to Gray Code:**

- ✓ **Example**: What is the equivalent of the binary number = 1101101 in Gray code?

- ✓ **G = 1011011**

Gray code

- **Convert Gray Code to Binary Code:**

  - Let G be a number written in Gray code on m bits

$$G_{(Gray)} = G_m \ldots\ldots G_3G_2G_1G_0,\ G_m \text{ is the most significant bit (MSB)}$$

  - B is the equivalent in pure or natural binary code of the number G, B also written on m bits.

$$B_{(2)} = B_m \ldots\ldots B_3B_2B_1B_0$$

Gray code

➢ **Convert Gray Code to Binary Code :**

✓ To convert a binary number to Gray code, follow these steps:

1. The first bit of the Gray code is the same as the first bit of the binary number.

$$B_m = G_m$$

2. Now we compare the pairs:

If G_{m-1} = B_m then B_{m-1} = 0

If G_{m-1} ≠ B_m then B_{m-1} = 1

Gray code

- **Convert Gray Code to Binary Code :**

- **Example**: Convert 0100 to Binary code

- G = 0100 =$G_3 G_2 G_1 G_0$

  1. G_3= B_3 $= 0$

  2. The second step :

$G_2 \neq B_3$ then B_2 = 1

$G_1 \neq B_2$ then B_1 = 1

$G_0 \neq B_1$ then B_0 = 1

- **B= 0111**

Gray code

- **Convert Gray Code to Binary Code :**

- ✓ **Example**: What is the equivalent of the Gray Code= 110010 in pure Binary code?

Gray code

- **Convert Gray Code to Binary Code :**

- **Example**: What is the equivalent of the Gray Code= 110010 in pure Binary code?

- B=100011

Binary Coded Decimal

- **BCD or Binary Coded Decimal code :**
  - ✓ It is a binary encoding for decimal numbers where each digit of a decimal number is represented by its own binary sequence.
  - ✓ Instead of encoding the entire number in binary, BCD treats each decimal digit independently and encodes it in a 4-bit binary form.
- **Example**: To encode the number 856, each digit of the number will be encoded separately in binary on 4 bits. $(856)_{10} = (1000\ 0101\ 0110)_{BCD}$

Binary Coded Decimal

- Decimal 0 to 9 are represented as 0000 to 1001 in BCD:

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Binary Coded Decimal

- **Advantages:**
  - ✓ Easy conversion between binary and decimal systems.
  - ✓ Suitable for applications that require decimal representation such as digital clocks, calculators, and financial applications.
- **Disadvantages:**
  - ✓ It requires more bits than pure binary to represent the same number.
  - ✓ Arithmetic operations are more complex.

Binary Coded Decimal

- **Example: Addition in BCD code**

- ✓ In BCD, only 10 digits are allowed (from 0 to 9).
- ✓ Add each 4-bit block of the first number, with its equivalent of the second number.
- ✓ If the result of one of the blocks exceeds 9, (its value is between 10 and 15 - 1010 to 1111), the number 6 (0110) is added to this block.
- ✓ 126+32=158

	0001	0010	0110
+	0000	0011	0010
	0001	0101	1000
	1	5	8

Binary Coded Decimal

- **Example: Addition in BCD code**

✓ 789+123=

	1 1 1		1 1
	0111	1000	1001
+	0001	0010	0011
	1000	1011	1100
		11	12

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Binary Coded Decimal

- **Example: Addition in BCD code**

✓ 789+123=

✓ Add the number 6 (0110) to the block that exceeds 9

	1 1 1 1	1	1 1
	0111	1000	1001
+	0001	0010	0011
		1 1	1
	1001	1011	1100
		0110	0110
	1001	0001	0010
	9	1	2

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

BCD+3

- The **BCD+3** or **excess 3** code
- Where 3 is added to each BCD digit
- Simplify the arithmetic operations like adding or subtracting decimal numbers.
- Reduce the need for complex corrections when performing arithmetic operations.
- Improve the speed and efficiency of arithmetic operations.

Decimal	Decimal+3	BCD+3
0	3	0011
1	4	0100
2	5	0101
3	6	0110
4	7	0111
5	8	1000
6	9	1001
7	10	1010
8	11	1011
9	12	1100

Binary Coded Decimal

- **Example: Addition in BCD+3 code**
- ✓ If carry occurs, add 3 (0011)
- ✓ If carry does not occur, subtract 3.
- ✓ 6+3=

```
    1001
+   0110
  ------
    1111   (No carry)
-   0011
  ------
    1100
```

Decimal	Decimal+3	BCD+3
0	3	0011
1	4	0100
2	5	0101
3	6	0110
4	7	0111
5	8	1000
6	9	1001
7	10	1010
8	11	1011
9	12	1100

Binary Coded Decimal

- **Example: Addition in BCD+3 code**

✓ If carry occurs, add 3 (0011)

✓ If carry does not occur, subtract 3.

✓ 7+3= 10

		1 1 1		
		0011	1010	
	+	0011	0110	
(no carry)		0111	0000	(carry)
	-	0011	+ 0011	
		0100	0011	

Decimal	Decimal+3	BCD+3
0	3	0011
1	4	0100
2	5	0101
3	6	0110
4	7	0111
5	8	1000
6	9	1001
7	10	1010
8	11	1011
9	12	1100

Data types

Integer representation

- **Unsigned Integer Representation**:
- ✓ Integers are represented in binary form, either as signed or unsigned numbers.
- ✓ An unsigned integer can only represent non-negative numbers (zero and positive).
- ✓ The representation of integers depends on the range of numbers to be used.
- ✓ If an integer uses n bits, an unsigned integer can represent values from 0 to $2^n - 1$.
- ✓ Example: for 8 bits
  - 0 to 255 ($2^8 - 1$)
  - 00000000 to 11111111

Integer representation

- **Signed Integer Representation**:
- ✓ A signed integer can represent both positive and negative numbers.
- ✓ The most common methods to represent negative numbers:
- ✓ Sign and Absolute Value (S/VA)
- ✓ One’s Complement
- ✓ The two-way complement

Integer representation

- **Signed Integer Representation**:

✓ **Sign and Absolute Value (S/VA):**

- The most significant bit (**MSB**) is used to represent the sign of the number (1: negative sign, 0: positive sign) , and the remaining bits represent the absolute value of the number.

- If an integer uses n bits, a signed integer can represent values from

$$-2^{n-1}-1 \text{ to } 2^{n-1}-1$$

- **Example**: for 8 bits
  - -127 to +127 (-2^7-1 to 2^7-1) with two zeros

Integer representation

- **Signed Integer Representation**:

✓ **Sign and Absolute Value (S/VA):**

- **Example**: For a 4-bit representation:

  +4: 0100 (MSB is 0, positive).

  −4: 1100 (MSB is 1, negative).

- **Issues**: The zero has two representations -0 (1000) and +0 (0000). This leads to complications in certain calculations.

Integer representation

- **Signed Integer Representation**:

✓ **One's Complement :**

- One's complement represents negative numbers by inverting all the bits of the corresponding positive number (change 0 to 1 and 1 to 0).

- If an integer uses n bits, a signed integer can represent values from

$$-2^{n-1} - 1 \text{ to } 2^{n-1} - 1$$

- **Example**: for 8 bits
  - -127 to +127 ($-2^{7} - 1$ to $2^{7} - 1$) with two zeros

Integer representation

- **Signed Integer Representation**:

✓ **One's Complement :**

- **Example**: For a 4-bit representation:

  +4: 0100

  −4: 1011 (flip all bits of 0100).

- **Issues**: The zero in this method also has two representations -0 (1111) and +0 (0000). This leads to complications in certain calculations.

Integer representation

- **Signed Integer Representation**:

✓ **The two-way complement:** the most widely used method

- Negative numbers are represented by inverting all the bits of the number and then adding 1 to the result.

- If an integer uses n bits, a signed integer can represent values from -2^{n-1} to $2^{n-1} - 1$ (Range: -128 to 127, there is only one representation for zero)

- **Example**: For a 4-bit representation:

  +4: 0100

  −4: Invert the bits 1011 then add 1 = 1100

Real numbers

- Real numbers in computers can be represented using **fixed-point representation** or **floating-point representation**.

- **Fixed-Point Representation**: is a way of representing real numbers where the position of the decimal point is fixed at a certain place. This means a certain number of bits are allocated to the integer part and a fixed number of bits are allocated to the fractional part.

Real numbers

- Real numbers in computers can be represented using **fixed-point representation** or **floating-point representation**.
- **Fixed-Point Representation**:

- **Example**: Suppose we want to represent the number 3.75 in 8 bits using a fixed-point representation with 4 bits for the integer part and 4 bits for the fractional part:
  - The number 3.75 can be written as 3 + 0.75.
  - In binary, 3 is 0011 and 0.75 is 1100 (because $0.75 \times 2^4 = 12$, which is 1100 in binary).
  - So, the 8-bit fixed-point representation would be 00111100.

Real numbers

- **Floating-point representation:** is a more flexible and widely used way. It can handle a much wider range of values compared to fixed-point representation by using an exponent to scale the value.

- ✓ The most common format used in computers for the representation of floating-point numbers in binary is the **IEEE 754 standard.**
- ✓ In the IEEE 754 standard, a floating point number is always represented by a triplet (S,E,M):
  1. Sign bit: Indicates whether the number is positive or negative.
  2. Exponent: Determines the scale (magnitude) of the number.
  3. Mantissa (or significand): Represents the precision or actual digits of the number.

Real numbers

- **Floating-point representation:**

- The general formula for a floating-point number is:

$$(-1)^S . 2^{(E-127)} . 1, M$$

- Two main types:

  - **IEEE 754 Single Accuracy (32 bits)**

Sign (1 bit)	**Exponent (8 bit)**	**Mantissa (23 bit)**

  - **IEEE 754 Double Precision: (64 bits)**

Sign (1 bit)	**Exhibitor (11 bit)**	**Mantissa (52 bit)**

Real numbers

- **Floating-point representation:**

✓ **Example 1:** Find the IEEE 754 single-precision representation of the number $(27.5)_{10}$

- The number is positive S=0
- $(27.5)_{10}$ = $(11011.1)_2$……. Fixed-point = $1.10111 * 2^4$ Floating point (M= 10111)
- Exhibitor: E-127 = 4, E= 127+4= 131 = $(10000011)_2$

0	10000011	10111000000000000000000
S	E	M

Real numbers

- **Floating-point representation:**

✓ **Example 2:** Find the IEEE 754 single-precision representation of the number $(-620.25)_{10}$

- The number is positive S=1
- $(-620.25)_{10}$ = $(1001101100.01)_2$ ……. Fixed-point = 1.00110110001 * 2^9 Floating point (M= 00110110001)
- Exhibitor: E-127 = 9, E= 127+9= 136 = $(10001000)_2$

1	10001000	00110110001000000000000
S	E	M

Real numbers

- **Floating-point representation:**

- ✓ **Example 3:** Find the IEEE 754 single-precision representation of the number $(-0.0625)_{10}$

  - $(-0.0625)_{10}$ The number is positive S=1
  - $(-0.0625)_{10}$ = $(0.0001)_2$……. Fixed-point = 1. * 2^{-4} Floating point (M= 0)
  - Exhibitor: E-127 = -4, E= 127-4= 123 = $(1111011)_2$

1	1111011	00000000000000000000000
S	E	M

Real numbers

- **Floating-point representation:**

✓ **Example 4:** Find the floating number with the following IEEE754 representation

1	10000010	01000100000000000000000
S	E	M

- $S = 1$ Number is negative
- E= $(10000010)_2$=130, E-127= 3
- $1.M = 1.010001$
- $1.010001 * 2^3$ = $(1010.001)_2$= $(10.125)_{10}$

Characters representation

- In computers, characters (such as letters, digits, punctuation marks, etc.) are represented using character encoding schemes, which map each character to a specific binary value.
- The most common methods for character representation are:
  - ✓ **ASCII**
  - ✓ **Unicode**
  - ✓ **EBCDIC.**

Characters representation

- **ASCII (American Standard Code for Information Interchange):**

✓ It is one of the most widely used character encoding standards.

✓ It represents characters using 7 bits (or 8 bits in extended versions) and maps each character to a unique binary number.

✓ 7 bits to represent 128 characters and 8 bits to represent 256 characters.

- **Example**:
  - 'A' = 65 (in decimal) = 01000001 (in binary)
  - 'a' = 97 (in decimal) = 01100001 (in binary)

- **Limitations**: Limited to 128 or 256 characters.

Characters representation

➢ **ASCII table:**

DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR
0	00	NUL	32	20	SP	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Characters representation

- **Unicode:** Unicode is a more comprehensive character encoding standard designed to support characters from all languages, as well as symbols, emojis, and even ancient scripts. It aims to provide a unique number for every character in the world, regardless of the platform, program, or language.

- **Example**:

  - 'A' = U+0041
  - '□' (musical symbol) = U+1D11E

Characters representation

- **Unicode:**
- Unicode can be represented using different encoding schemes:
  - ✓ **UTF-8**: A variable-length encoding that uses 1 to 4 bytes to represent characters. It's widely used for web pages and file formats.
  - ✓ **UTF-16**: Uses 2 bytes for most characters, but some characters may use 4 bytes
  - ✓ **UTF-32**: Uses 4 bytes for all characters, ensuring fixed-length encoding.
- **Limitations**: More memory-intensive than ASCII, and more complex.

Characters representation

- **EBCDIC (Extended Binary Coded Decimal Interchange Code):**

- ✓ EBCDIC is an encoding system primarily used on IBM mainframe and midrange computer systems.
- ✓ It is different from ASCII in that it is a 8-bit encoding system (256 unique characters) with a unique character mapping.
- ✓ Example: ‘A’
  - **ASCII**: 65 in decimal, 41 in Hexa, 01000001 in binary.
  - **EBCDIC**: C1 in Hexa, 11000001 in binary.

- **Limitations**: Not widely used, and less standard than ASCII and Unicode.

The end