1. Introduction

Data representation is defined as the methods used to represent data in computers. In other words, it refers to the form in which data is stored and processed.

2. Binary code

There are three main binary coding systems:

2.1 Natural binary code

In this code, numbers are represented in straight binary (b=2). This code is simple and easy to implement, but it has the disadvantage of changing more binary digits between two consecutive numbers.

2.2 Gray code (reflected binary)

In Gray code two consecutive numbers differ from each other by only one bit. Gray code is not suitable for arithmetic operations but it is widely used in digital transmission systems to aid in error correction (minimizes the occurrence of errors), improves the signal's quality and consumes less power.

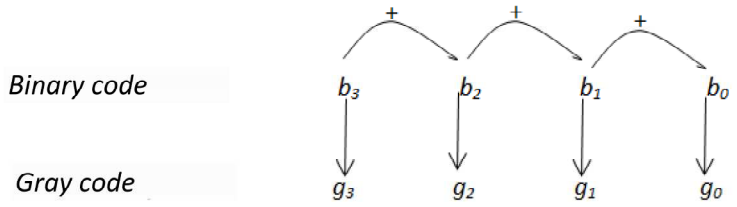
- **Conversion Binary → Gray code**

The conversion from natural binary to reflected binary (Gray code) is as follows:

- The MSB remains the same as in binary (unchanged)
- Starting from left to right, each bit is added to its neighbor on the right. The sum is carried over to the lower line which corresponds to the Gray code. Carries are neglected.

Note: the Gray code always has the same number of bits as the natural binary representation.

Example: consider a 4-digit number written *in natural binary* $(b_3b_2b_1b_0)_2$, the Gray code $(g_3g_2g_1g_0)_{gc}$ is obtained as follows :



Example: convert values 35 and 36 to straight binary and Gray code

Binary $35 = (1\ 0\ 0\ \ 0\ 1\ 1)_2$ $36 = (1\ 0\ 0\ \ 1\ 0\ 0)_2$

Gray code **$35 = (1\ 1\ 0\ \ 0\ 1\ 0)_{gc}$** **$36 = (1\ 1\ 0\ \ 1\ 1\ 0)_{gc}$**

Example: write numbers from 0 to 15 in natural binary and Gray code

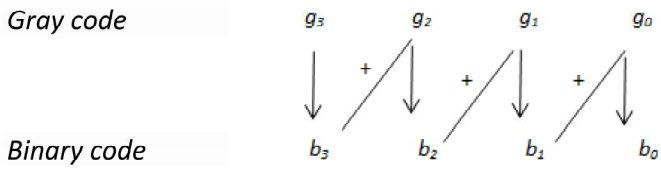
decimal	*Binary*	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Note: the Gray code is called reflected binary, because the **n-1** bits are generated by reflection (mirror).

- **Conversion Gray code → Binary**

- The MSB remains the same as in Gray code (unchanged)

- Starting from left to right, each bit of the binary code is added to its diagonal neighbor in Gray code. The sum is carried over to the lower line which corresponds to the binary code. Carries are neglected.



Example: convert the Gray $(1\ 1\ 0\ \ 0\ 1\ 0)_{gc}$ into decimal

$$(1\ 1\ 0\ \ 0\ 1\ 0)_{gc} = (1\ 0\ 0\ \ 0\ 1\ 1)_2 = \mathbf{35}$$

2.3 BCD Code (binary coded decimal)

In BCD, each decimal digit is replaced by its 4-bit binary equivalent.

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

Example: convert the number 275 into BCD

$2 \Rightarrow (0010)_2$

$7 \Rightarrow (0111)_2$

$5 \Rightarrow (0101)_2$

$$275 = (\mathbf{0010\ 0111\ 0101})_{BCD}$$

The main advantage of BCD is that it allows easy conversion between decimal and binary.

BCD has many important applications especially using digital displays (electronic calculators, digital clocks, etc.)

3. Data types

3.1 Numeric data

Is a data in the form of numbers that can be used in arithmetic calculations.

- Unsigned Integers (positive numbers). ***E.g*** : age, number of students in class,…
- Signed Integers. ***E.g*** : temperature,…
- Real numbers. ***E.g*** score, weight, size…..

3.2 Alphanumeric data

Set of characters including letters, special characters and numerals that are not used in calculations. ***E.g*** : address, name ….

4. Integer representation

4.1 Unsigned Integers

Unsigned integers can represent zero and positive integers. Natural binary is used to represent these numbers.

The range of unsigned integers for **n** bits register is **$[0, 2^n-1]$**

Example: write the value 35 as an 8-bits unsigned integer.

$35 = \mathbf{(00100011)_2}$

4.2 Signed Integers

Signed integers can represent zero, positive and negative integers. Three representations had been proposed for signed integers:

- **Signed-Magnitude** representation
- **Ones' complement** representation (1's Complement)
- **Two's complement** representation (2's Complement)

4.2.1 Signed-Magnitude

- The most significant bit (MSB) is the sign bit, with value of 0 representing positive integer and 1 negative integer.

- The remaining **n-1** bits represent the magnitude (absolute value) of the integer.

The range of numbers that can be represented by n-bit signed-magnitude is

$$[-(2^{n-1}-1), +(2^{n-1}-1)]$$

***Example*:** convert the following values -15 and +20 into 8-bit Signed-Magnitude numbers.

- ***Conversion Signed-Magnitude →decimal***

Convert the **n-1** bits to decimal and introduce (+) if the MSB=0 or (-) otherwise.

Example: the following binary numbers are 8-bit Signed-magnitude numbers. What are the decimal values?

$(00001001)_2 = +9$ $(10000101)_2 = -5$

Question: express all represented Signed-Magnitude numbers, using 4 bits

Answer: see the table below.

Question: perform in 4-bit Signed-Magnitude representation the following operations and give the decimal results. 5+2 5-2

5+2

```
 0101
+0010
------
 0111
```

Correct result +7

5-2=5+(-2)

```
 0101
+1010
------
 1111
```

incorrect result -7

- **Signed-Magnitude drawbacks**

Signed-Magnitude method is very simple, but it has some drawbacks:

- There are two representations of zero (00……00) and (10……00) which could lead to inefficiency and confusion. (***E.g***. to test if a number is 0 or not, the CPU will need two tests).
- The difficulty of arithmetic operations which are complicated, because of the sign bit which must be treated separately (designing an appropriate circuit is difficult).
- The sign of both numbers have to be examined before the operation is determined (addition or subtraction).
- Two separate circuits are required to do the addition and subtraction operations (the ideal is to use a single adder-subtractor circuit that does both addition and subtraction.)

Signed-M	*decimal*
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	+0
1000	-0
1001	-1
1010	-2
1011	-3
1100	-4
1101	-5
1110	-6
1111	-7

4.2.2 One's complement

- Positive numbers are obtained by conversion to natural binary.

- Negative numbers are obtained by inverting each bit of the positive opposite (0 becomes 1 and 1 becomes 0).

Example: convert the following values to 8-bit 1's complement +12 and -23

- +12= **(00001100)2**
- Positive value of -23 is +23 =$(00010111)_2$, 1's complement ➔ **$(11101000)_2$**

Question: find the same values into 16-bit 1's complement

Answer:

- +12= **$(0000000000001100)_2$**
- Positive value of -23 is +23 =$(0000000000010111)_2$,
  1's complement ➔ **$(1111111111101000)_2$**

The range of numbers that can be represented by n-bit 1's complement is

$$[-(2^{n-1}-1), +(2^{n-1}-1)]$$

- ***Conversion 1's complement → decimal***

- Determine whether the number is positive or negative (look at the MSB)
- If the number is positive, convert to decimal
- If the number is negative, complement the number, convert to decimal and introduce the sign (-)

Example: convert the following 8-bit 1's complement numbers to decimal

$(00001111)_2$ $(11110011)_2$

$(00001111)_2$ = **+15** $(11110011)_2$ = **-12**

Example: convert the following 16-bit 1's complement numbers to decimal

$(1111111111100110)_2$ $(1111111111111111)_2$

$(1111111111100110)_2$ = **-25** $(1111111111111111)_2$ = **-0**

Question: express all represented 1's complement numbers, using 4 bits

Answer: see the table below.

1's comp	Decimal
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	+0
1111	- 0
1110	-1
1101	-2
1100	-3
1011	-4
1010	-5
1001	-6
1000	-7

- ***1's complement addition/ subtraction***

Addition in 1's complement is done as follows:

- Add the two numbers.
- If an end carry occurs, add the carry to the result.

Subtraction in 1's complement is done as follows:

- Transform the subtraction to the addition of the 1's complement.
- Use 1's complement addition rules.

Example: calculate the following operations using 1's complement form (4-bit)

5 + 2 -5-2 5 -2 -5 +2

5+2

```
 0101
+0010
------
 0111
= +7
```

(-5)+(-2)

```
  1010
+ 1101
------
 0111
+    1
------
 1000
= -7
```

5+(-2)

```
  0101
+ 1101
------
 0010
+    1
------
 0011
=+3
```

(-5)+2

```
 1010
+0010
------
 1100
=-3
```

Note :

The disadvantage of 1's complement method is the double representation of zero +0 and – 0.

Question: perform the following operations using 1's complement form (4-bit)

5+4 -5 -7

Answer:

5+4

```
  0101
+ 0100
------
 1001
= -6
```

incorrect result

(-5)+(-7)

```
  1010
+ 1000
------
 0010
+    1
------
 0011
= +3
```

incorrect

The results are incorrect because +9 and -12 are outside the range [-7, +7]

- **Overflow problem**

- The overflow occurs because the width of registers is finite.
- The overflow occurs when two numbers of **n** bits each are added and the result occupies **n+1** digits.
- An overflow can be detected when the sign of the result is different from the sign of the two numbers.
- The overflow cannot occur when the two numbers have different signs.

Note:

In unsigned integer addition, overflow occurs if there is end carry.

4.2.3 Two's complement

- Positive number is obtained by conversion to natural binary.
- The 2's complement of negative number equals its **1's complement + 1**

Example: convert the following values to 8-bit 2's complement +12 and -35

- +12= $\mathbf{(00001100)_2}$
- Positive value of -35 is +35 =$(00100011)_2$, 1's complement of -35 ➔ $(11011100)_2$

The 2's complement of -35= $(11011100)_2$ +1 = $\mathbf{(11011101)_2}$

Question*:* find the same values into 16-bit 2's complement

Answer:

- +12= $\mathbf{(0000000000001100)_2}$
- Positive value of -35 is +35 =$(0000000000100011)_2$,
  1's complement of -35 ➔ $(1111111111011100)_2$

The 2's complement of -35= $(1111111111011100)_2$ +1 = $\mathbf{(1111111111011101)_2}$

The range of numbers that can be represented by n-bit 2's complement is

$$\mathbf{[-2^{n-1}, +(2^{n-1}-1)]}$$

Note:

The easiest way to obtain the 2's complement of a negative number is by starting from LSB (the positive value), leaving all the 0s and the first 1 unchanged and complement all the remaining bits.

***Example*:** convert the following values to 8-bit 2's complement -24 and -19

+24= (00011000) +19=$(00010011)_2$

-24= $\mathbf{(11101000)_2}$ -19= $\mathbf{(11101101)_2}$

- ***Conversion 2's complement →decimal***

- Determine whether the number is positive or negative (look at the MSB).
- If the number is positive, convert to decimal.
- If the number is negative, flip all the bits and add 1 then convert to decimal (introduce the sign -)

Note:

Another method to obtain the decimal value of 2's complement negative number is to leave (from the LSB) all the 0s and the first 1 unchanged, flip all the remaining bits then convert to decimal (introduce the sign -)

Example: convert the following 8-bit 2's complement numbers to decimal

$(00001110)_2$ $(10001000)_2$ $(11101110)_2$

- $(00001110)_2 =$ **+14**
- $(10001000)_2 = -(01111000)_2 =$ **- 120**
- $(11101110)_2 = -(00010010)_2 =$ **-18**

Example: convert the following 16-bit 2's complement numbers to decimal

$(1111111111101111)_2$ $(1111111111111000)_2$

- $(1111111111101111)_2 = -(0000000000010001)_2 =$ **-17**
- $(1111111111111000)_2 = -(0000000000001000)_2 =$ **- 8**

Question: express all represented 2's complement numbers, using 4 bits

Answer: see the table below.

2's comp	Decimal
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	+0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

- ***2's complement addition/ subtraction***

Addition / subtraction in 2's complement is similar to 1's complement with the following difference:

o If an end carry occurs it'll be dropped rather than added to the result

Example: Calculate the following operations using 2's complement form (4-bit)

5 + 2	-5-2	5 -2	-5 +2
5+2	-5+(-2)	5+(-2)	-5+2
0101 + 0010 --- 0111 = +7	1011 + 1110 --- 1001 = -7	0101 + 1110 --- 0011 = +3	1011 + 0010 --- 1101 = -3

Question: perform the following operations using 2's complement form (4-bit)

5+4 -5 -7

Answer:

5+4

^{1}0101
+ 0100

1001
= - 7
incorrect result

(-5)+(-7)

10^{1}11
+ 1001

0100
= +4 incorrect result

Overflow problem!!!

The results are incorrect because +9 and -12 are outside the range [-8,+7]

- **Advantages of 2's complement**

- The most used representation for negative numbers in computers
- One representation of zero
- One additional number - $\mathbf{2^{n-1}}$

Note :

In 1's complement or 2's complement, arithmetic operations are advantageous. The subtraction of a number is reduced to the addition of its complement. This allows the machine to use a single adder-subtractor circuit that does both addition and subtraction.

Exercise 3.1

Compare the signed integers types in C++ and java

Solution:

C++	JAVA	size	Range
---------	byte	8 bits	[-128 , +127]
short	short	16 bits	[-32768 , +32767]
int	int	32 bits	[-2147483648 , +2147483647]
long	long	64 bits	[-9223372036854775808 , +9223372036854775807]

Exercise 3.2

1. The program below is written in C++

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    short A=20000,B=30000, C=A+B;
    cout<<C;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

What is the result after execution of the code? Explain the problem and find a solution.

5. Real numbers

A fractional number has two parts: integer part and decimal part these parts are separated by a dot (**.**) called the decimal point. The problem in fractional number representation is how to indicate to the machine the position of the decimal point? There are two ways to represent real numbers:

5.1 Fixed-Point Representation

In this representation, the most significant bit represents the sign (+/-), the integer part is represented on a fixed *n* bits and the fractional part on a fixed *p* bits.

Sign	integer part	Fractional part
1bit	**n bits**	**p bits**

Example: assume a 6-bit fractional number with 3 bits for integer part and 2 bits for the

- Find the representation of the following values +5.75 -0. 25 -4.125
- Calculate the range of the possible values

Answer:

- +5.75 $= (010111)_2$

  -0. 25 $= (100001)_2$

  -4.125 impossible

- The range is [-7.75, +7.75]

[-7.75, -7.50, -7.25,……+0.00……..,+7.25, +7.50, +7.75]

- **Fixed-Point advantages**

- Fixed point representation is easy to implement.
- Arithmetic operations are simple and can be performed faster (like integers) .

- **Fixed-Point drawbacks**

- In fixed point representation, range of representable numbers is limited.
- Loss of precision.
- There is no standard for fixed point representation.

5.2 Floating-Point Representation

Floating point representation is similar to scientific notation. The proper format for **normalized** scientific notation of a number N is $\pm a \times 10^b$ where $1 \leq a < 10$ and b is the power of 10. In binary $N=\pm a \times 2^b$ where $1 \leq a < 2$ and b is the power of 2.

Question: express the following numbers in **normalized** scientific notation

637.8 -0.0475 89 $-(10101)_2$ $(1101.011)_2$

$(0.01101)_2$ $(33.476)_8$ $(279.DE3)_{16}$

Answer :

$637.8 = +6.378 \times 10^{+2}$ $-0.0475 = -4.75 \times 10^{-2}$ $89 = +8.9 \times 10^{+1}$

$-(10101)_2 = -(1.0101)_2 \times 2^{+4}$ $(1101.011)_2 = +(1.101011)_2 \times 2^{+3}$

$(0.01101)_2 = +(1.101)_2 \times 2^{-2}$ $(33.476)_8 = (3.3476)_8 \times 8^{+1}$

$(279.DE3)_{16} = (2.79DE3)_{16} \times 16^{+2}$

In the early days of computing, each computer constructor (brand) had its own floating point format. This had the unfortunate effect that a code that worked perfectly well on one machine could crash on another one (portability limited).

- **IEEE 754 Floating Point representation standard**

IEEE 754 is a technical standard for floating-point representation which was established in 1985 by the **Institute of Electrical and Electronics Engineers (IEEE)**. IEEE 754 has 3 basic components:

1. The Sign:
2. The Biased exponent
3. The Mantissa

- **IEEE 754 single precision**

The number is expressed as follows:

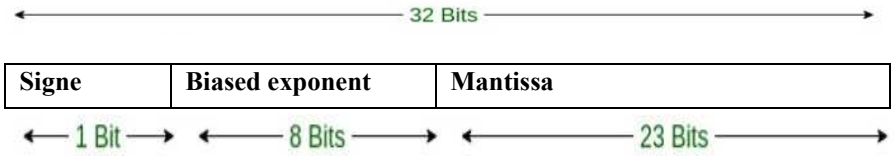
• 1 bit for sign

• 8 bits for the biased (shifted) exponent. **Biased exponent = Real exponent +127**. The exponent is shifted by $2^{8-1} - 1 = 127$. This shift is useful because the exponent can be positive or negative.

• 23 bits for the mantissa

The following steps provide the method to convert a real number to floating point format:

1. Convert the number to Binary
2. Normalize the number under the form **$N = (+/-)(1.m)_2.2^{ER}$**
3. Calculate the **Biased exponent, BE=RE+ bias= RE+127**
4. Store the **sign**, **BE** and the **mantissa** in 32 bits



Note:
Since the first bit of a normalized binary floating point number is always 1 (always exists), we don't need to store it explicitly in the memory. This bit is called *"hidden bit"* .

Examples: convert the following numbers to IEEE 754 single-precision.
-3.625 +65.75 -0.125

- N=-3.625

1. Convert to binary $N = -(11.101)_2$
2. Normalize $N = -(1.1101)_2 \times 2^{+1}$
3. Calculate the **Biased exponent , BE= RE+127**= 1+127 = 128
4. Store S, BE and M in 32 bits

1 10000000 11010000000000000000000

- N=+65.75

1. Convert to binary $N = +(1000001.11)_2$
2. Normalize $N = +(1.\ 00000111)_2 \times 2^{+6}$
3. Calculate the **Biased exponent ,** **BE= RE+127**= 6+127 = 133
4. Store S, BE and M in 32 bits

0 10000101 00000111000000000000000

- N= -0.125

1. Convert to binary $N = -(0.001)_2$
2. Normalize $N = +(1.\ 0)_2 \times 2^{-3}$
3. Calculate the **Biased exponent ,** **BE= RE+127**= -3+127 = 124
4. Store S, BE and M in 32 bits

1 0111100 00000000000000000000000

- **IEEE 754 single precision → decimal**

To convert a number written in IEEE754 single precision to decimal:

1. Calculate the **Real exponent, RE= BE-127**
2. Calculate value = sign $\times$(1, mantissa)$_2 \times 2^{\mathbf{ER}}$, with sign = ± 1
3. Convert the value to decimal (polynomial form)

Examples: convert to decimal the following IEEE 754 single-precision numbers.

$(C0980000)_{16}$ $(42484000)_{16}$

- $N=(C0980000)_{16}$

Convert to binary $N = (1\ 10000001\ 00110000000000000000000)_2$

1. Calculate the **Real exponent, RE= BE-127 =129-127=+2**
2. $N=-(1.0011)_2 \times 2^{+2}$
3. Convert the value to decimal $N=-(1.0011)_2 \times 2^{+2} == -(100.11)_2 = -4.75$

- $(42484000)_{16}$ = +50.0625

- **Special Values**

***Zero** :

SIGNE = 0/1	EXPOSANT = 000…..0	MANTISSA =000…..0

***Infinity** (+infinity, -infinity): result of overflow or division by 0

SIGNE = 0/1	EXPOSANT = 111…..1	MANTISSA =000…..0

***NAN (Not A Number):** undefined values such as:

$(\pm 0) / (\pm 0)$ $(\pm\infty) / (\pm\infty)$ $(\pm 0) \times (\pm\infty)$ $0 * (\pm\infty)$

$\infty - \infty$ $-\infty + \infty$ square root of a negative number

SIGNE = 0/1	EXPOSANT = 111…..1	MANTISSA $\neq$ 000…..0

Note:

There is a compromise between the size of the mantissa (23 bits) representing the accuracy and the size of the exponent (8 bits) representing the range.

- More digits assigned for the mantissa ➔ higher precision and lower range
- More digits assigned for the exponent➔ higher range and lower precision

Note:

The IEEE754 standard defines three formats for representing floating point numbers:

. Single precision on 32 bits (1bit 8bits 23 bits Bias= 127)

. Double precision on 64 bits (1 bit 11bits 52 bits Bias= 1023)

. Extended precision on 80 bits (1 bit 15 bits 64 bit Bias= 16383)

- **IEEE 754 Addition/subtraction operations**

Floating point arithmetic is more complicated than the fixed point. To calculate A+B (or A-B) we have to follow these steps:

1- Write the two numbers in normalized form

2- Align the exponents (smaller exponent aligned to the larger)

3- Add/subtract the mantissas

4- Renormalize if necessary

Examples

Assume A and B two IEEE 754 single –precision numbers. Calculate A+B , A-B and B-A

A= 0 10000010 10101000000000000000000

B= 0 10000001 00100000000000000000000

- A+B

  $A+B= +(1,10101)_2 \ x2^3 \ + (1,001)_2 \ x2^2$

  $= \ + \ ((1,10101)_2 \ +(0,1001)_2 \ )x2^3 = (10,00111)_2 \ x2^3 =$ **$(1 ,000111)_2 \ x2^4$**

  A+B= 0 10000011 00011100000000000000000

- A-B

  $A-B= +(1,10101)_2 \ x2^3 \ - (1,001)_2 \ x2^2$

  $= \ + \ ((1,10101)_2 \ - (0,1001)_2 \ )x2^3 = (1,00011)_2 \ x2^3$

  A-B= 0 10000010 00011000000000000000000

- B-A

  $B-A= (1,001)_2 \ x2^2 - (1,10101)_2 \ x2^3$

  $= \ ((0,1001)_2 - (1,10101)_2 \ )x2^3 = - \ ((1,10101)_2 \ - (0,1001)_2 \ )x2^3 = - (1,00011)_2 \ x2^3$

  B-A= 1 10000010 00011000000000000000000

Note :

B-A can be obtained directly by flipping the sign of the operation A-B.

6. Characters representation

The set of alphanumeric characters includes letters 'a'……'z', 'A…….'Z', numerals 0…..9, and special characters +,-,*, ? #. The character representation is done by a correspondence table specific to each code used.

There are a number of standards, we can cite examples:

- **EBCDIC** (Extended Binary Coded Decimal Interchange Code) : 8-bit encoding, mainly used by IBM.
- **ASCII** (American Standard Code for Information Interchange): each character on 7 bits → 128 characters
- **Extended ASCII**: each character on 8 bits → 256 characters
- **Unicode :** it is the coding of most alphabets: Arabic, Chinese, Hebrew, Cyrillic …….. It includes several standards: UTF-8, UTF-16, UTF-32…...

Example*:* use the Extended ASCII table to code the following characters *'A' 'a' ' ?'*

'A' → *code* $(41)_{16} = (01000001)_2$

'a' → *code* $(61)_{16} = (01100001)$

' ?' → *code* $(3F)_{16} = (00111111)_2$

Left / Right		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	NUL	DLE	SP	0	@	P	`	p	Ç	É	á	▒	└	ð	Ó	-
1	0001	SOH	DC1	!	1	A	Q	a	q	ü	æ	í	▒	┴	Ð	ß	±
2	0010	STX	DC2	"	2	B	R	b	r	é	Æ	Ó	▓	┬	Ê	ô	‗
3	0011	ETX	DC3	#	3	C	S	c	s	â	ô	ú	│	├	Ë	ò	¾
4	0100	EOT	DC4	$	4	D	T	d	t	ä	ö	ñ	┤	─	È	ő	¶
5	0101	ENQ	NAK	%	5	E	U	e	u	à	ò	Ñ	Á	┼	ı	Ő	§
6	0110	ACK	SYN	&	6	F	V	f	v	å	û	ª	Â	ã	í	µ	÷
7	0111	BEL	ETB	'	7	G	W	g	w	ç	ù	º	À	Ã	î	þ	¸
8	1000	BS	CAN	(	8	H	X	h	x	ê	ÿ	¿	©	╚	ï	Þ	°
9	1001	HT	EM	)	9	I	Y	i	y	ë	Ö	®	╣	╔	┘	Ú	¨
A	1010	LF	SUB	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Û	·
B	1011	VT	ESC	+	;	K	[	k	{	ï	ø	½	╗	╦	█	Ù	¹
C	1100	FF	FS	,	<	L	\	l	|	î	£	¼	╝	╠	▄	ý	³
D	1101	CR	GS	-	=	M	]	m	}	ì	Ø	¡	¢	=	¦	Ý	²
E	1110	SO	RS	.	>	N	^	n	~	Ä	×	«	¥	╬	ì	¯	■
F	1111	SI	US	/	?	O	_	o	DEL	Å	ƒ	»	┐	¤	▀	´	

Example1 : 'A'= $(41)_{16}$ =$(01000001)_2$

Example2 : 'A B C'= 41 20 42 20 43

Uppercase letters are represented by codes 41 to 5A and lowercase letters by codes 61 to 7A. By modifying the 6th bit we change from uppercase to lowercase.

The ASCII table defines 34 control characters:

Abbrev.	Full name
NUL	Null
SOH	Start of Header
STX	Start of Text
ETX	End of Text
EOT	End of Transmission
ENQ	Enquiry
ACK	Acknowledge
BEL	Bell
BS	BackSpace
HT	Horizontal Tabulation
LF	Line Feed/New Line
VT	Vertical Tabulation
FF	Form Feed/New Page
CR	Carriage Return
SO	Shift Out
SI	Shift In
DLE	Data Link Escape

Abbrev.	Full name
DC1	Device Control 1
DC2	Device Control 2
DC3	Device Control 3
DC4	Device Control 4
NAK	Negative Acknowledge
SYN	Synchronous Idle
ETB	End of Transmission Block
CAN	Cancel
EM	End of Medium
SUB	Substitute
ESC	Escape
FS	File Separator
GS	Group Separator
RS	Record Separator
US	Unit Separator
SP	Space
DEL	Delete