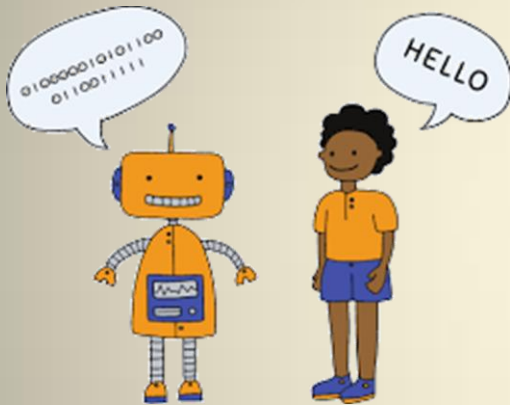


While the concept of number system was present in the 'Abacus ' considered as the first calculating machine of the world, it has progressed up to the computer of today.

Investigates how instructions and data are represented in computers and exploit them in arithmetic and logic operations

A/L ICT – Lesson 03



Competency 3: Investigates how instructions and data are represented in computers and exploit them in arithmetic and logic operations.

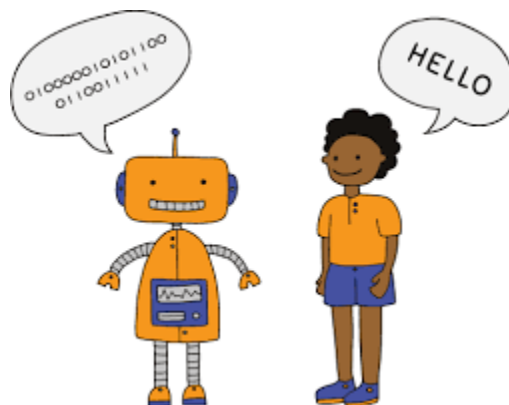
Competency level 3.1: Analyses how numbers are represented in computers.

Number System

When typing letters or words using the computer, these words or letters are represented by the computer as numbers it can understand. While this group of numbers that the computer can understand is called a 'Number System' the limited number of numerals in the number system called digits. The value of these numbers (numerals) depends on the position they occupy within the number. While the concept of number system was present in the 'Abacus ' considered as the first calculating machine of the world, it has progressed up to the computer of today. The number system used for the representation of data in the computer is as follows;

Number System	Base Value	Number and Alphabetic character used
Binary	2	0,1
Octal	8	0, 1, 2, 3, 4, 5, 6, 7
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Hexa Decimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Computer represents data in two signal states. There are two Voltage levels for these two symbols. One is named as the high voltage level and the other is named as low voltage level. "0" and "1" digits respectively represent these low and high voltage levels in a circuit. Thus, "1" and "0" status are equal to the “On” and “Off” states of an electronic circuit. Any data in the world can be represented on the computer using these two digits.



Most and Least Significant Positional Value of a Number.

There are two separate methods to find the most and least significant values of decimal numbers and whole numbers. When a whole number is read from left to right, the number in the right most end is the least significant positional value and the number in the left most end which is not 0 is the most significant positional value. In decimal numbers, the value in the right extreme after the decimal point which is not 0 becomes the least significant positional value and the number in the left extreme of the decimal point which is not 0 becomes the most significant positional value.

Most Significant Digit (MSD) and Least Significant Digit (LSD)

Number	MSD	LSD
329	3	9
1237.0	1	7
58.32	5	2
0.0975	9	5
0.4	4	4
0401.0020	4	2

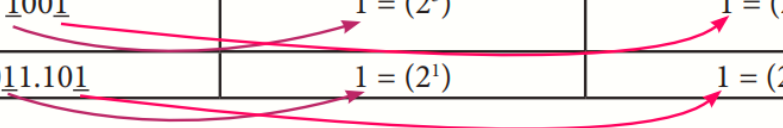
You can use the same method used for the decimal number system to find the most and least significant positional digits of binary, octal and hexadecimal numbers.

Most Significant Bit (MSB) and Least Significant Bit (LSB)

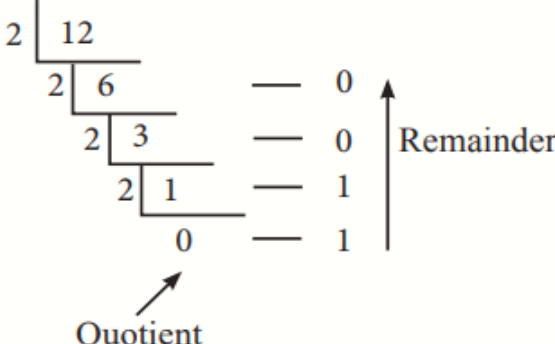
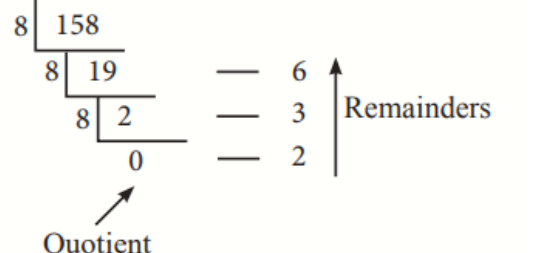
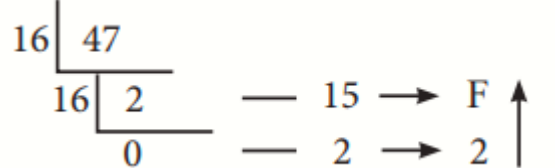
Only the Binary Number System is used to find the most significant bit (MSB) and the least significant bit (LSB). There are two ways to find this using decimal numbers and whole numbers. In a whole number, read from left to right, the value in the right extreme is the least significant bit and the value in the left extreme which is not 0 is the most significant bit. In binary decimal numbers, the value in the right extreme of the decimal point which is not 0 is the least significant bit and the value in the left extreme of the decimal point which is not 0 is the most significant bit.

Table 3.10 - The most significant bit and the least significant bit

Binary Number	MSB	LSB
<u>1</u> 00 <u>1</u>	1 = (2^3)	1 = (2^0)
0 <u>1</u> 1. <u>1</u> 0 <u>1</u>	1 = (2^1)	1 = (2^{-3})



Conversions between number systems.

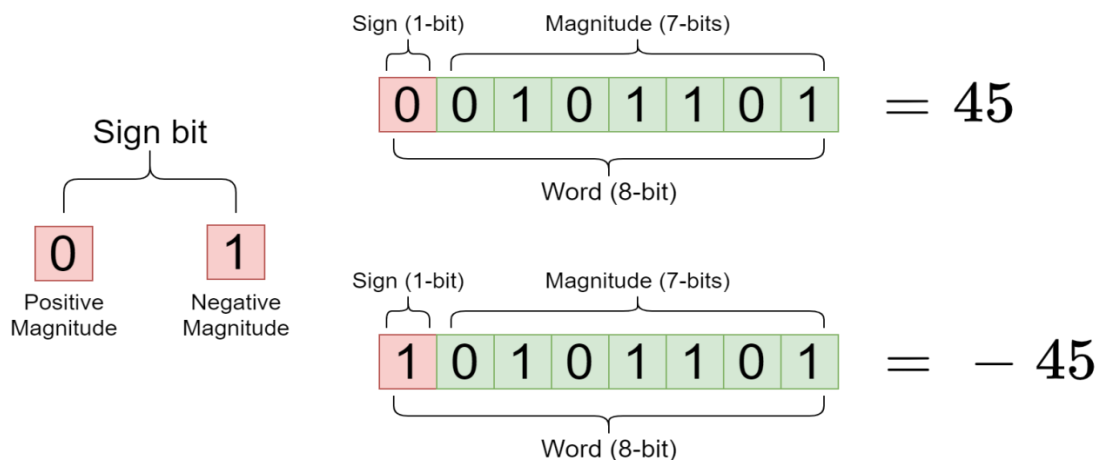
<p>Decimal Into Binary</p>	 <p>Quotient</p> <p>Remainder</p>	<p>E.g.:- convert 0.3125_{10} to binary</p> <table border="1" data-bbox="917 220 1128 535"> <tr> <td></td><td>0.3125</td><td>x2</td></tr> <tr> <td>0</td><td>.625</td><td>x2</td></tr> <tr> <td>1</td><td>.25</td><td>x2</td></tr> <tr> <td>0</td><td>.50</td><td>x2</td></tr> <tr> <td>1</td><td>.00</td><td></td></tr> </table> <p>$0.3125_{10} = 0.0101_2$</p>		0.3125	x2	0	.625	x2	1	.25	x2	0	.50	x2	1	.00	
	0.3125	x2															
0	.625	x2															
1	.25	x2															
0	.50	x2															
1	.00																
<p>Binary To Decimal</p>	<p> $1101_2 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0)$ $= (1 \times 8) + (1 \times 4) + (0 \times 2) + (1 \times 1)$ $= 8 + 4 + 0 + 1$ $1101_2 = 13_{10}$ </p>	<p>Zero Point</p> <p>0.1011</p> <p> $1 \times 2^{-4} = 0.0625$ $1 \times 2^{-3} = 0.125$ $0 \times 2^{-2} = 0$ $1 \times 2^{-1} = 0.5$ $\hline 0.6875_{10}$ </p>															
<p>Decimal To Octal</p>	 <p>Quotient</p> <p>Remainders</p>	<table border="1" data-bbox="901 1008 1112 1186"> <tr> <td>0</td><td>0.3125</td><td>x8</td></tr> <tr> <td>2</td><td>.50</td><td>x8</td></tr> <tr> <td>4</td><td>.0</td><td>x8</td></tr> </table> <p>$0.3125_{10} = 0.24_8$</p>	0	0.3125	x8	2	.50	x8	4	.0	x8						
0	0.3125	x8															
2	.50	x8															
4	.0	x8															
<p>Octal To Decimal</p>	<p> $1275_8 = (1 \times 8^3) + (2 \times 8^2) + (7 \times 8^1) + (5 \times 8^0)$ $= (1 \times 512) + (2 \times 64) + (7 \times 8) + (5 \times 1)$ $= 512 + 128 + 56 + 5$ $1275_8 = 701_{10}$ </p>																
<p>Decimal To Hexa Decimal</p>																	

<p>Hexa Decimal To Decimal</p>	<div> <div> <div>A</div> <div>B</div> <div>2</div> </div> <div> <div>↓</div> <div>↓</div> <div>↓</div> </div> <div> <div>16^2</div> <div>16^1</div> <div>16^0</div> </div> </div> $AB2_{16} = (A \times 16^2) + (B \times 16^1) + (2 \times 16^0)$ $= (10 \times 256) + (11 \times 16) + (2 \times 1)$ $= 2560 + 176 + 2$ $AB2_{16} = 2738_{10}$	
<p>Octal To Binary</p>	<div> <div>4</div> <div>5</div> <div>7</div> </div> <div> <div>↓</div> <div>↓</div> <div>↓</div> </div> <div> <div>100</div> <div>101</div> <div>111</div> </div> $457_8 = 100101111_2$	
<p>Binary To Octal</p>	<div> <div> <div>0</div><div>0</div><div>1</div><div>0</div><div>1</div><div>1</div><div>1</div><div>0</div><div>1</div> </div> <div>Binary number</div> </div> <div> <div> <div>0</div><div>0</div><div>1</div> </div> <div> <div>0</div><div>1</div><div>1</div> </div> <div> <div>1</div><div>0</div><div>1</div> </div> </div> <div>Cluster of 3</div> <div> <div>1</div> <div>3</div> <div>5</div> </div> $1011101_2 = 135_8$	
<p>Hexa decimal To Binary</p>	<div> <div>2</div> <div>A</div> <div>E</div> </div> <div> <div>↓</div> <div>↓</div> <div>↓</div> </div> <div> <div>0010</div> <div>1 010</div> <div>1110</div> </div> $2AE_{16} = 1010101110_2$	
<p>Binary To Hexa Decimal</p>	<div> <div> <div>0</div><div>1</div><div>0</div><div>1</div><div>1</div><div>1</div><div>0</div><div>1</div><div>1</div><div>1</div><div>0</div><div>0</div> </div> <div>Binary number</div> </div> <div> <div> <div>0</div><div>1</div><div>0</div><div>1</div> </div> <div> <div>1</div><div>1</div><div>0</div><div>1</div> </div> <div> <div>1</div><div>1</div><div>0</div><div>0</div> </div> </div> <div>Cluster of 4</div> <div> <div>5</div> <div>D</div> <div>C</div> </div> <div>Hexadecimal number</div> $10111011100_2 = 5DC_{16}$	

<p>Octal To Hexa Decimal</p>	<div> <div> 1057 001000101111 </div> <div> 001000101111 </div> <div> 2215 22F </div> <div> 1057₈ = 22F₁₆ </div> </div>
<p>Hexa Decimal To Octal</p>	<div> <div> 23A 001000111010 </div> <div> 001000111010 </div> <div> 1072 1072 </div> <div> 23A₁₆ = 1072₈ </div> </div>

Sign-Magnitude

The sign and magnitude method is commonly an 8 bit system that uses the most significant bit (MSB) to indicate a positive or a negative value. By convention, a '0' in this position indicates that the number given by the remaining 7 bits is positive, and a most significant bit of '1' indicates that the number is negative. This interpretation makes it possible to create a value of negative zero.



One's Complement

In one's complement, positive numbers are represented as usual in regular binary. However, negative numbers are represented differently. To negate a number, replace all zeros with ones, and ones with zeros - flip the bits. Thus, 12 would be 00001100, and -12 would be 11110011. As in signed magnitude, the leftmost bit (most significant bit-MSB) indicates the sign (1 is negative, 0 is positive). To compute the value of a negative number, flip the bits and translate as before.

When representing positive and negative numbers in 8-bit ones complement binary form, the positive numbers are the same as in signed binary notation.

- -120 is represented in one's complement form as 10000111_2 and
- -60 is represented in one's complement form as 11000011_2

0 1 1 0 1 1 1 0 ← Original binary value

1 0 0 1 0 0 0 1 ← 1's complement

The ones complement system still has two ways of writing 0_{10} ($00000000_2 = +0_{10}$ and $11111111_2 = -0_{10}$);

Two's Complement.

A single set of bits is used. To form a negative number, start with a positive number, complement each bit and add one. This interpretation includes one more negative value than positive values (to accommodate zero).

0 1 1 0 1 1 1 0 ← Original binary value

1 0 0 1 0 0 0 1 ← 1's complement

1 0 0 1 0 0 0 1
+ 1
1 0 0 1 0 0 1 0

← 2's complement

Addition using 2's complement

1. Show how the computation **15+(-5)** is done in 8-bit two's complement arithmetic. Explain how you deal with the carry generated in the most significant bit.

15 -> 00001111₂

5-> 00000101₂

-5-> 11111010₂+00000001₂ ->11111011₂

00001111₂(15)

+11111011₂(-5)

00001010₂

- **Ignore the carry bit**

2. Explain how the positive and negative numbers in two's complement can be converted into decimal numbers.

Identify the sign of the final decimal number by MSB

If MSB is 0 - >positive

- **Convert to decimal**

If MSB is 1 ->negative

- **Take the sign as negative**
- **Get binary number**
- **Invert bit values**
- **Add 1 to LSB**
- **Convert the number into decimal**

Representation Method	Usage
Sign Magnitude	Used only when we do not add or subtract the data. They are used in analog to digital conversions. They have limited use as they require complicated arithmetic circuits.
One's Complement	Simpler design in hardware due to simpler concept.
Two's Complement	Makes it possible to build low-cost, high-speed hardware to perform arithmetic operations.

Fixed point numbers

In calculations involving fixed point numbers that have a fixed number of digits after the decimal point.

E.g. :- 763.2135 (The decimal point is located in the same position in each number)
 179.4821

 942.6956

Floating point

The floating point number is used to represent

- Numbers with fractions, e.g., 3.1416
- Very small numbers, e.g., 0.000000001
- Very large numbers, e.g., 3.15576×10^9

There are several ways to represent floating point number but IEEE 754 is the most efficient in most cases. IEEE 754 has 3 basic components:

1. **The Sign of Mantissa –**

This is as simple as the name. 0 represents a positive number while 1 represents a negative number.

2. **The Biased exponent –**

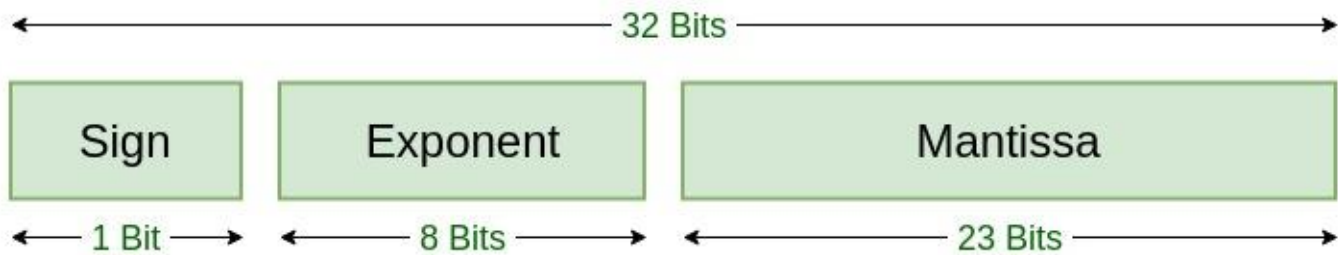
The exponent field needs to represent both positive and negative exponents. A bias is added to the actual exponent in order to get the stored exponent.

3. **The Normalised Mantissa –**

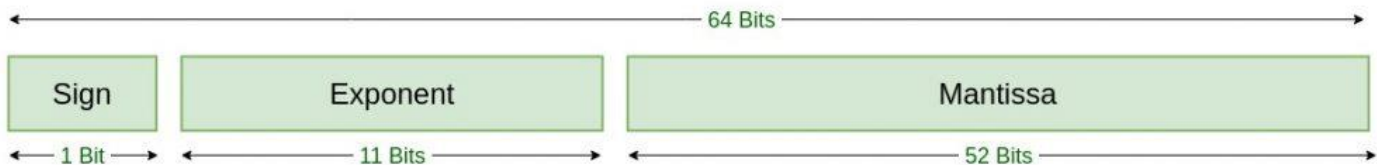
The mantissa is part of a number in scientific notation or a floating-point number, consisting of its significant digits. Here we have only 2 digits, i.e. 0 and 1. So a normalised mantissa is one with only one 1 to the left of the decimal.

IEEE 754 numbers are divided into two based on the above three components: single precision and double precision.

single precision IEEE 754 floating point standard



double precision IEEE 754 floating point standard



Ex1: 85.125 is represented in single precision floating point as below,

- First convert it to binary

$$85.125 = 1010101.001 = 1.010101001 \times 2^6$$

- sign = 0
- biased exponent

$$127 + 6 = 133$$

- Normalised mantisa = 010101001 , we will add 0's to complete the 23 bits

The IEEE 754 Single precision is:

sign	exponent	mantisa
0	10000101	01010100100000000000000 we added 0's to complete the 23 bits

Ex2 : -10.625 is represented in single precision floating point as below.

- First convert it to binary

1010.101₂ (the whole and the fractional part separately)

- Put it in standard form

1.010101x2³

- Biased value for exponent

3+127 = 130₁₀

- exponent part(130) in binary

10000010₂

- Sign = 1

The IEEE 754 Single precision is:

sign	exponent	mantisa
1	10000010	010101000000000000000000

	Advantage	Disadvantage
Fixed Point Representation	Performance good. No need to rely on additional hardware or software logic	Limited range of values can represent
Floating point representation	Greater range of numbers is represented. Varying degrees of precision.	More storage space needed. Slower processing times. Lack of precision

Competency level 3.2: Analyses how characters are represented in computers

Following are different coding systems used.

BCD - Binary Coded Decimal

This coding system was used in the early stages of computing. In this system one digit is represented by 4 bits.

This is used only to represent decimal numbers. Sixteen symbols ($2^4 = 16$) can be represented in this system.

The table 3.15 shows the BCD codes for the 10 digits from 0 to 9.

Table 3.15 – Decimal Numbers and BCD Values

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

Example

Indicating number 37_{10} in BCD codes.

$$\begin{array}{cc} 3 & 7_{10} \\ 0011 & 0111 \\ 37_{10} = & 00110111 \end{array}$$

ASCII – American Standard Code for Information Interchange

Normally uses 8 bits (1 byte) to store each character. However, the 8th bit is used as a check digit, meaning that only 7 bits are available to store each character. This gives ASCII the ability to store a total of $2^7 = 128$ different values. The 7 bit ASCII code was originally proposed by the American National Standard Institute (ANSI). (IBM personal computers use ASCII).

Example

- Text

When the word 'School' is entered into the computer through the keyboard, write down how it is understood by the computer.

⌚ First, write the decimal numbers for the symbols.

S - 83 c - 99 h - 104 o - 111 l - 108

⌚ Write binary numbers for each value.

S - 1010011 c - 1100011 h - 1101000 o - 1101111
l - 1101100

⌚ Write the associated code

S c h o o l
101001111000111101000110111111011111101100

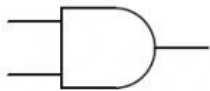
EBCDIC – Extended Binary Coded Decimal Interchange Code

We can write only 128 characters using ASCII system, but the EBCDIC code system allows the use of 256 characters. Here, one symbol can be written with a binary number which consists of 8 bits. Hence, 256 characters can be represented using this system. This system was used in IBM main frame computers. The table below shows that there are different EBCDIC codes for the 26 different capital letters and 26 different EBCDIC codes for the 26 simple letters in this system.

UNICODE – Though 128 characters can be used in the ASCII system and 256 characters can be used in the EBCDIC system for data representation. For example, these systems cannot be used for Sinhala, Japanese, Chinese and Tamil languages as there are more than 256 characters. Hence Unicode system was designed according to a standard to represent 65536 different symbols of 16 bits.

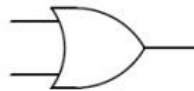
Code System	Number of Bits Used
BCD - Binary Coded Decimal	4
ASCII - American Standard Code for Information Interchange Code	7
EBCDIC- Extended Binary Coded Decimal Interchange Code	8
Unicode	16

Competency level 3.3: Uses basic arithmetic and logic operations on binary numbers



AND

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1



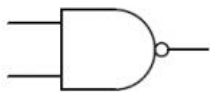
OR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1



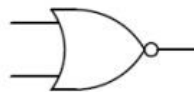
XOR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0



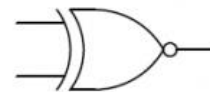
NAND

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0



NOR

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0



XNOR

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

Example :- Find 10101_2 **AND** 11100_2 , 10101_2 **OR** 11100_2 , 10101_2 **XOR** 11100_2

Number 1	1	0	1	0	1
Number 2	1	1	1	0	0
<hr/>					
AND	1	0	1	0	0
OR	1	1	1	0	1
XOR	0	1	0	0	1