UNIT 2. DATA AND INFORMATION

Data is encoded information, ready to be processed by a computer. In other words, data is just a way to represent information. We consider information to be data once it has been processed and interpreted, and its result is displayed in an intelligible manner.

2.1 Number Systems

The so-called machine language of computers, as well as the information stored by them is digital and can be represented with the binary numbering system. However, hexadecimal system is also used in some cases, therefore it is necessary to be familiar with it as well. In our daily lives, we use the decimal system, which uses 10 different digits to represent a certain quantity: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. It is a positional system as well, because the value of each digit depends on its position.

For example:

$2165 = 2 \cdot 10^3 + 1 \cdot 10^2 + 6 \cdot 10^1 + 5 \cdot 10^0 = 2000 + 100 + 60 + 5$

a. Binary System

In this system, we only have two digits: 0 and 1. For example, 10100 could be a binary number. To avoid confusion with the decimal system, the base is shown in between brackets at the right side of the number, as seen in the following example: $10100_{(2)}$.

To find the decimal value of a binary number, simply multiply each of its digits by the corresponding power of 2 according to its position (starting from 2^0), in a similar way to what we did in the previous example with decimal numbers.

Let's look at an example:

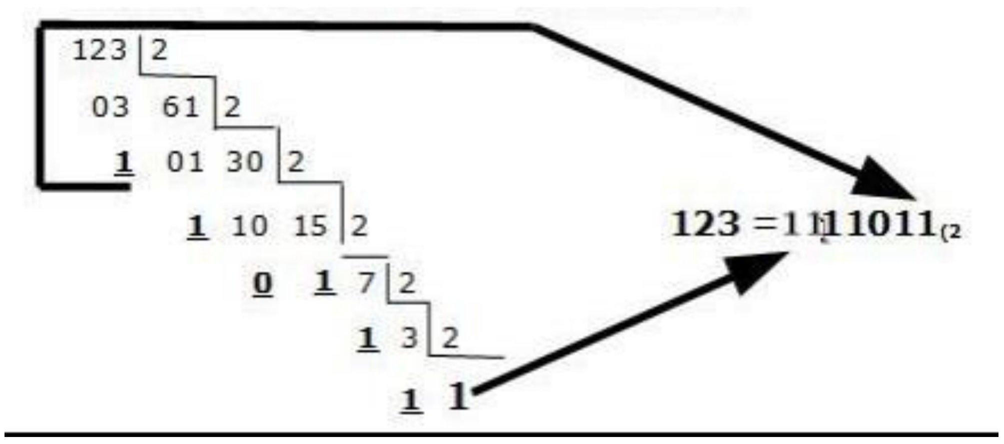
$10101001_{(2)} = 1x2^7 + 0x2^6 + 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0$

$= 1x128 + 0x64 + 1x32 + 0x16 + 1x8 + 0x4 + 0x2 + 1x1 = 169$

To find the binary expression of a decimal number, we perform successive divisions by two until we get a quotient of 1. The binary number equivalent to the given decimal will consist of the digits of the last quotient, the last remainder, ..., up to the first remainder.

Let’s look at an example:

Convert the decimal 123 to binary:



b. Hexadecimal System

This system has a base of 16 (which is equal to 2^4) and, therefore, uses 16 symbols or digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. The letters represent the following decimal numbers: A = 10, B = 11, C = 12, D = 13, E = 14, and F = 15.

This system is one of the most widely used nowadays in data processing because it simplifies the writing of decimal numbers and because converting hexadecimal to binary (and vice versa) is very simple (each hexadecimal digit is represented by four binary digits).

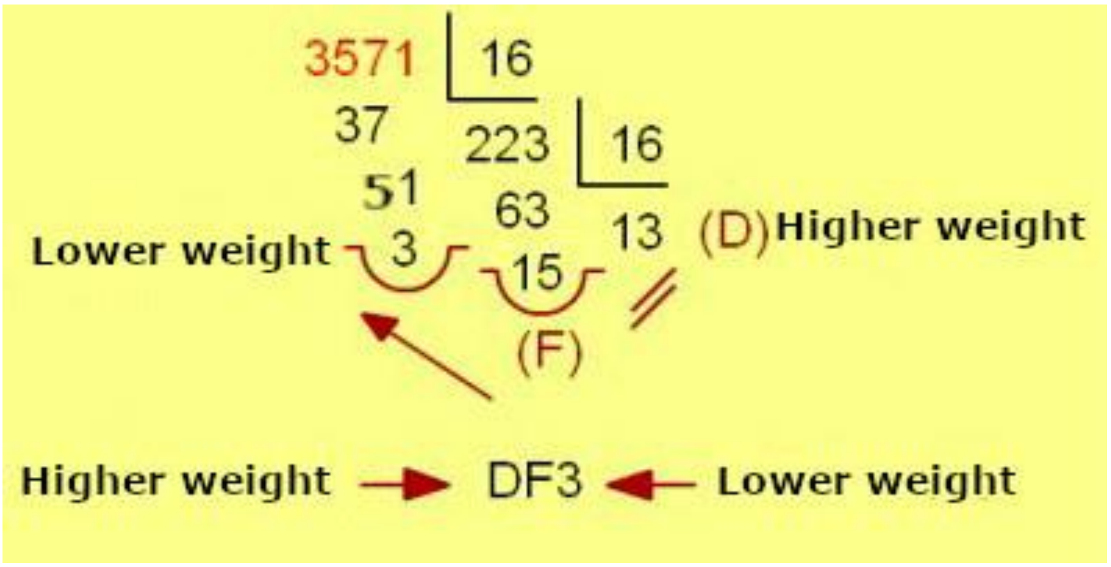
The conversion of a hexadecimal number to decimal is done in the same way as converting a binary number to its decimal equivalent, but this time, instead of multiplying by successive powers of 2, we multiply by successive powers of 16, as shown in the following example:

$2B7E_{(16)} = 2\cdot16^3 + B\cdot16^2 + 7\cdot16^1 + E\cdot16^0$

$= 2\cdot4096 + 11\cdot256 + 7\cdot16 + 15$

$= 8192 + 2816 + 112 + 15 = 11135$

To convert a decimal number to its hexadecimal equivalent, we perform successive divisions by 16, just as we did to convert from decimal to binary. Let's look at an example: we'll calculate the hexadecimal equivalent of decimal 3571.



Thus, 3571 = DF3($_{16}$).

If we wanted to obtain the binary equivalent of a hexadecimal number, we need to take the following equivalencies into account:

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

An example: 7A2($_{16}$) = 0111 1010 0010($_{2}$).

Number system exercises part 1:

1.- Convert the following numbers from binary to decimal:

a) **$11010_{(2)}$**
b) **$111000_{(2)}$**
c) **$101111_{(2)}$**
d) **$100011_{(2)}$**
e) **$11101010_{(2)}$**

2.- Convert the following numbers from decimal to binary:

a) **$28_{(10)}$**
b) **$125_{(10)}$**
c) **$234_{(10)}$**
d) **$101_{(10)}$**
e) **$255_{(10)}$**

3.- Convert the following numbers from decimal to hexadecimal:

a) **$29_{(10)}$**
b) **$55_{(10)}$**
c) **$1048576_{(10)}$**
d) **$2457_{(10)}$**
e) **$5496_{(10)}$**

4.- Convert the hexadecimal numbers of the previous exercise to binary.

5.- Convert the following numbers from binary to hexadecimal:

a) **$10001010_{(2)}$**
b) **$111000110101110_{(2)}$**
c) **$1010101100110011_{(2)}$**
d) **$11110001001001000110110011001 0_{(2)}$**
e) **$1111001010100111101100110111 0111_{(2)}$**

6.-Convert the following IP_{v4} addresses into binary:

a) **206.253.208.100**
b) **172.16.0.0**
c) **192.168.31.255**
d) **176.195.16.0**

IPv4 and IPv6 addresses formats

Octets or segments, or a combination of both, make up Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6) addresses.
An IPv4 address has the format x.x.x.x, where *x* is called an *octet* and must be a decimal value between 0 and 255. Octets are separated by periods. An IPv4 address must contain three periods and four octets. The following examples are valid IPv4 addresses:

- 1.2.3.4
- 01.102.103.104

An IPv6 address can have either of the following two formats:

- Normal - Pure IPv6 format
- Dual - IPv6 plus IPv4 formats

An IPv6 (normal) address has the format y:y:y:y:y:y:y:y, where *y* is called a *segment* and can be any hexadecimal value between 0 and FFFF. The segments are separated by colons, not periods. An IPv6 normal address must have eight segments; however, a short form notation can be used for segments that are zero, or those that have leading zeros.
The following are examples of valid IPv6 (normal) addresses:

- 2001:db8:3333:4444:5555:6666:7777:8888
- 2001:db8:3333:4444:CCCC:DDDD:EEEE:FFFF
- :: (implies all 8 segments are zero)
- 2001:db8:: (implies that the last six segments are zero)
- ::1234:5678 (implies that the first six segments are zero)
- 2001:db8::1234:5678 (implies that the middle four segments are zero)
- 2001:0db8:0001:0000:0000:0ab9:C0A8:0102 (This can be compressed to eliminate leading zeros, as follows: 2001:db8:1::ab9:C0A8:102)

An IPv6 (dual) address combines an IPv6 and an IPv4 address and has the following format: y:y:y:y:y:y:x.x.x.x. The IPv6 portion of the address (indicated with y's) is always at the beginning, followed by the IPv4 portion (indicated with x's).

- In the IPv6 portion of the address, *y* is called a *segment* and can be any hexadecimal value between 0 and FFFF. The segments are separated by colons, not periods. The IPv6 portion

of the address must have six segments but there is a short form notation for segments that are zero.
- In the IPv4 portion of the address *x* is called an *octet* and must be a decimal value between 0 and 255. The octets are separated by periods. The IPv4 portion of the address must contain three periods and four octets.

The following are examples of valid IPv6 (dual) addresses:

- 2001:db8:3333:4444:5555:6666:1.2.3.4
- ::11.22.33.44 (implies all six IPv6 segments are zero)
- 2001:db8::123.123.123.123 (implies that the last four IPv6 segments are zero)
- ::1234:5678:91.123.4.56 (implies that the first four IPv6 segments are zero)
- ::1234:5678:1.2.3.4 (implies that the first four IPv6 segments are zero)
- 2001:db8::1234:5678:5.6.7.8 (implies that the middle two IPv6 segments are zero)

Number system exercises part 2:

Tip to complete these exercises read and understand the previous content.

7.- Convert the following IP_{v6} addresses from hexadecimal into binary:

a) **2001:DB8:3333:4444:CCCC:DDDD:EEEE:FFFF**
b) **2050:0DB9:0001::0AB9:C0A8:0102**
c) **2A0C:5A84::2::1**
d) **2A0C:5A80:F200:BD00:19F6:88D6:396B:1BC5**

8.-Convert the following mac numbers from hexadecimal into binary:

a) **AC-22-0B-0C-67-1B**
b) **6C-71-D9-7D-5A-66**
c) **6C-71-D9-7D-34-12**
d) **00-09-0F-FE-00-01**

9.- Convert the following IP_{v6} (dual) addresses from hexadecimal/decimal into binary

a) **2001:DB8:122:344::192.0.2.33**
b) **2001:DB8::138.46.55.53**
c) **2001:DB8:ABCD:: 0.66.131.41**
d) **2001:DB8::66.115.54.2**

2.2 Encoding Information

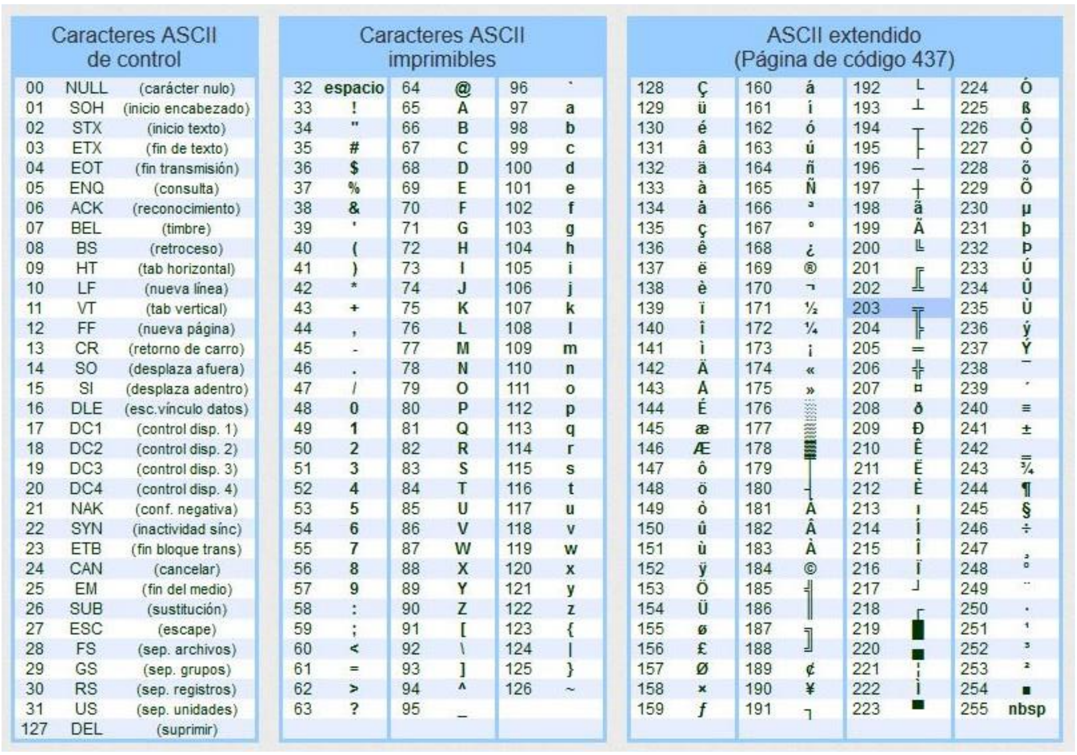
Alphanumeric codes are used in computers to represent digits and letters of the alphabet. Additional codes are needed to represent control characters and symbols like punctuation marks, parentheses, etc. The most well-known are ASCII and, more recently, UNICODE.

The ASCII code (American Standard Code for Information Interchange) was created in the U.S. to allow all computers to "speak" the same language, so any program made to run on one computer could be run on any other. Since this code became the standard for all computers and spread to non-English-speaking countries, it was necessary to extend it (extended ASCII), which increased the number of encoded characters from 128 to 256, represented with eight bits (one byte or octet) instead of the original 6 or 7 bits. However, this code was not fully standardized, so the same character could have different encodings (as happened, for example, with the letter ñ). For this reason, the

ANSI code was created to solve this problem. Although similar to ASCII, it is not universally used.

To achieve real internationalization of character encoding, UNICODE was created. The first 256 characters in UNICODE coincide with the 256 characters in the ANSI code. This code assigns a unique number to each character and allows for the encoding of 65,536 different characters or symbols using 16 bits per character, which is enough for today's languages and independent of the computing platform. This is the code currently becoming widespread.

Inmediately below you can see the tables of the **ASCII code:**



Caracteres ASCII de control		
00	NULL	(carácter nulo)
01	SOH	(inicio encabezado)
02	STX	(inicio texto)
03	ETX	(fin de texto)
04	EOT	(fin transmisión)
05	ENQ	(consulta)
06	ACK	(reconocimiento)
07	BEL	(timbre)
08	BS	(retroceso)
09	HT	(tab horizontal)
10	LF	(nueva línea)
11	VT	(tab vertical)
12	FF	(nueva página)
13	CR	(retorno de carro)
14	SO	(desplaza afuera)
15	SI	(desplaza adentro)
16	DLE	(esc.vínculo datos)
17	DC1	(control disp. 1)
18	DC2	(control disp. 2)
19	DC3	(control disp. 3)
20	DC4	(control disp. 4)
21	NAK	(conf. negativa)
22	SYN	(inactividad sínc)
23	ETB	(fin bloque trans)
24	CAN	(cancelar)
25	EM	(fin del medio)
26	SUB	(sustitución)
27	ESC	(escape)
28	FS	(sep. archivos)
29	GS	(sep. grupos)
30	RS	(sep. registros)
31	US	(sep. unidades)
127	DEL	(suprimir)

Caracteres ASCII imprimibles					
32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

ASCII extendido (Página de código 437)							
128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

Numbers from 0 to 31 in the ASCII table are assigned to control characters used to manage peripheral devices like printers. For example, number 12 represents the paper feed/new page function, instructing the printer to move directly to the top of the next page. These are non-printable characters. Numbers 32 to 126 are assigned to characters included in the keyboard and appear when viewing or printing a document. Number 127 is the DELETE command. Extended ASCII characters address the need for additional characters. The extended ASCII code includes the 128 characters from the original ASCII and adds another 128 characters, bringing the total to 256. Even with these additional characters, many languages have symbols that cannot be condensed into 256 characters. For this reason, there are ASCII variants to cover the characters and symbols of specific languages of many countries.

2.3 Information Measurement Units

The **bit** is the smallest unit of information used in computing, in any digital device, or in information theory. With it, we can represent any two values, such as true or false, open or closed, black or white, north or south, male or female, red or blue, etc. One of these values is assigned to the "off" state (0), and the other to the "on" state (1). The word bit comes from binary digit.

A **byte** or **octet** is a sequence of contiguous bits, whose size depends on the information code or character set it is defined in. It is commonly used as the basic unit of data storage, often combined with quantity prefixes. Originally, the byte was chosen as a sub-multiple of the computer's word size, ranging from five to twelve bits. The term "octet" is widely used as a precise synonym where ambiguity is undesirable (for example, in protocol definitions). Therefore, we have 1 byte = 8 bits.

Because binary numbers use powers of 2 rather than powers of 10, the units of information in computer science were considered a little bit weird by users. As you can see below the units do not coincide exactly with the units of the International System of units:

One kilobyte (kB) = 1024 bytes. (2^{10} bytes instead of 10^3 bytes)
One megabyte (MB) = 1024 kB (2^{20} bytes)
One gigabyte (GB) = 1024 MB (2^{30} bytes)
One terabyte (TB) = 1024 GB (2^{40} bytes)
One petabyte (PB) = 1024 TB (2^{50} bytes)
One exabyte (EB) = 1024 PB (2^{60} bytes)

For this reason, in December 1998, the International Electrotechnical Commission (IEC) stepped in. (http://physics.nist.gov/cuu/Units/binary.html). The following list includes the IEC standardized units:

 One kilobyte (kB) = 1000 bytes
 One megabyte (MB) = 1000 kB = 1,000,000 bytes
 One gigabyte (GB) = 1000 MB = 1,000,000,000 bytes
 One terabyte (TB) = 1000 GB = 1,000,000,000,000 bytes
 One petabyte (PB) = 1000 TB = 1,000,000,000,000,000 bytes.
 One exabyte (EB) = 1000 PB = 1,000,000,000,000,000,000 bytes.

However, both lists coexist nowadays because some programs (and even some operating systems) still use the pre-1998 notation.

Note the use of the uppercase "B" to distinguish between **Byte** and **bit**.

The above units are always used to "measure" information. In the case of data transmission, the unit is bit per second or its multiples (kilobits per second, megabits per second, etc.)