SVKM'S NMIMS, School of Technology Management & Engineering
Navi-Mumbai
B-Tech (A.Y. 2020-21)
Lecture Notes
Unit-3 Data representation

Content:

1) **Signed number representation**
2) **Fixed and Floating point representations**
3) **Character representation.**

1) Signed Number Representation:

In mathematics, positive numbers (including zero) are represented as unsigned numbers. That is we do not put the +ve sign in front of them to show that they are positive numbers.

However, when dealing with negative numbers we do use a -ve sign in front of the number to show that the number is negative in value and different from a positive unsigned value and the same is true with **signed binary numbers**.

However, in digital circuits there is no provision made to put a plus or even a minus sign to a number, since digital systems operate with binary numbers that are represented in terms of "0's" and "1's".

Mathematical numbers are generally made up of a sign and a value (magnitude) in which the sign indicates whether the number is positive, (+) or negative (–), with the value indicating the size of the number, for example 23, +156 or -274.

Presenting numbers is this fashion is called "sign-magnitude" representation since the left most digits can be used to indicate the sign and the remaining digits the magnitude or value of the number.

Sign-magnitude notation is the simplest and one of the most common methods of representing positive and negative numbers either side of zero, (0).

Thus negative numbers are obtained simply by changing the sign of the corresponding positive number as each positive or unsigned number will have a signed opposite, for example, +2 and -2, +10 and -10, etc.

But how do we represent signed binary numbers if all we have is a bunch of one's and zero's. We know that binary digits, or bits only have two values, either a "1" or a "0" and conveniently for us, a sign also has only two values, being a "+" or a "–".

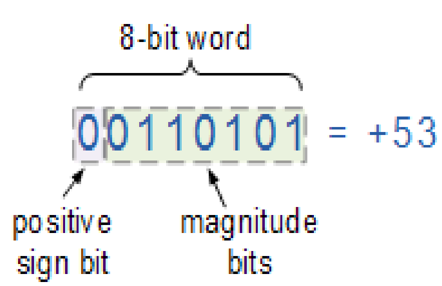
Then we can use a single bit to identify the sign of a signed binary number as being positive or negative in value. So to represent a positive binary number (+n) and a negative (-n) binary number, we can use them with the addition of a sign.

For signed binary numbers the most significant bit (MSB) is used as the sign bit. If the sign bit is "0", this means the number is positive in value. If the sign bit is "1", then the number is negative in value.

The remaining bits in the number are used to represent the magnitude of the binary number in the usual unsigned binary number format way.

➢ Positive Signed Binary Numbers:

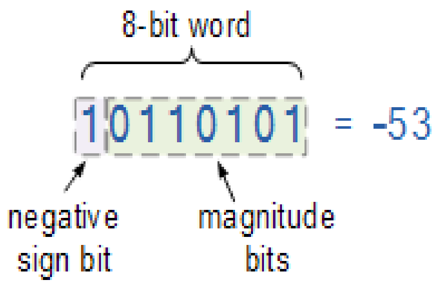
The binary numbers having their MSB 0 are called "Positive signed binary numbers".



Positive Number	Binary Equivalent
0	000
1	001
2	010
3	011

➢ Negative Signed Binary Numbers:

The binary numbers having their MSB 1 are called “Negative signed binary numbers”.



Negative Number	Binary Equivalent
1	1001
2	1010
3	1011
4	1100

Unsigned numbers can have a wide range of representation.

But whereas, in case of signed numbers, we can represent their range only from [$-(2^{(n-1)}-1)$ to $+(2^{(n-1)}-1)$].

Where n is the number of bits (including sign bit).

Ex: For a 5 bit signed binary number (including 4 magnitude bits & 1 sign bit), the range will be

$$-(2^{(5-1)}-1) \text{ to } +(2^{(5-1)}-1)$$

$$-(2^{(4)}-1) \text{ to } +(2^{(4)}-1)$$

$$-15 \text{ to } +15$$

Unsigned 8- bit binary numbers will have range from 0-255. The 8 – bit signed binary number will have maximum and minimum values as shown below.

The maximum positive number is 0111 1111 (+127)

The maximum negative number is 1000 0000 (–127)

2) Fixed and Floating point representations:

Digital Computers use Binary number system to represent all types of information inside the computers.

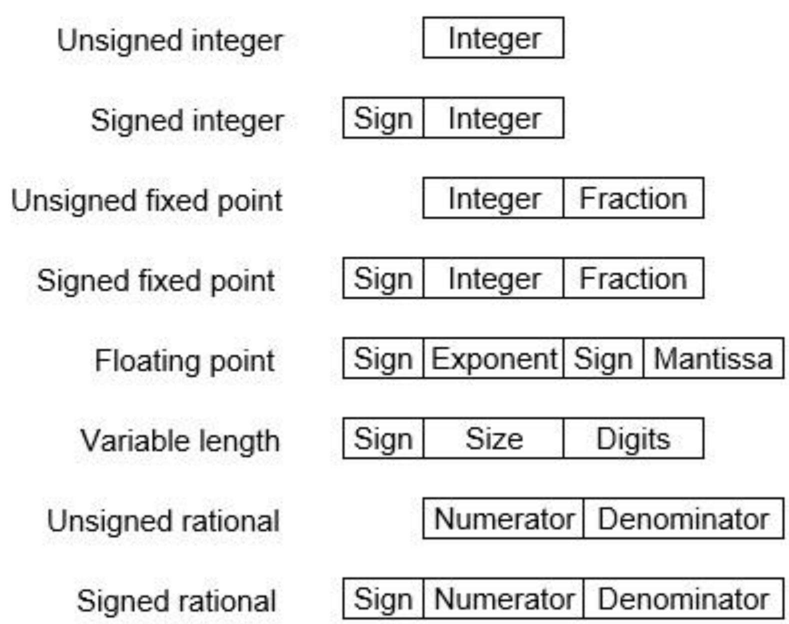
Alphanumeric characters are represented using binary bits (i.e., 0 and 1). Digital representations are easier to design, storage is easy, accuracy and precision are greater.

There are various types of number representation techniques for digital number representation, for example: Binary number system, octal number system, decimal number system, and hexadecimal number system etc.

But Binary number system is most relevant and popular for representing numbers in digital computer system.

Storing Real Number

These are structures as following below –



There are two major approaches to store real numbers (i.e., numbers with fractional component) in modern computing. These are (i) Fixed Point Notation and (ii) Floating Point Notation. In fixed point notation, there are a fixed number of digits after the decimal point, whereas floating point number allows for a varying number of digits after the decimal point.

Fixed-Point Representation –

This representation has fixed number of bits for integer part and for fractional part. For example, if given fixed-point representation is IIII.FFFF, then you can store minimum value is 0000.0001 and maximum value is 9999.9999.

There are three parts of a fixed-point number representation: the sign field, integer field, and fractional field.

Unsigned fixed point		Integer	Fraction
Signed fixed point	Sign	Integer	Fraction

We can represent these numbers using:

- Signed representation: range from - $(2^{(k-1)}-1)$ to $(2^{(k-1)}-1)$, for k bits.
- 1's complement representation: range from - $(2^{(k-1)}-1)$ to $(2^{(k-1)}-1)$, for k bits.
- 2's complementation representation: range from - $(2^{(k-1)})$ to $(2^{(k-1)}-1)$, for k bits.

Example –Assume number is using 32-bit format which reserve 1 bit for the sign, 15 bits for the integer part and 16 bits for the fractional part.

Then, -43.625 are represented as following:

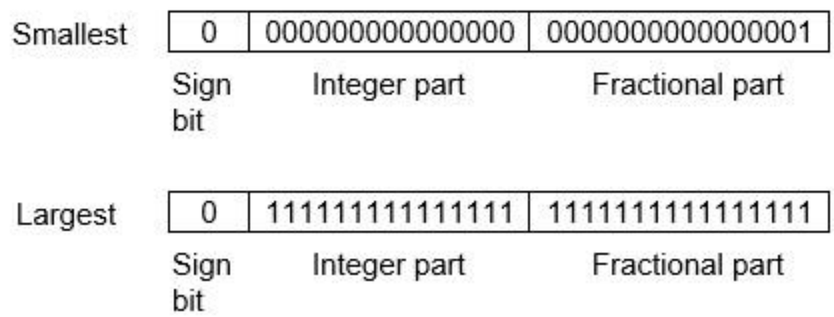
1	000000000101011	1010000000000000
Sign bit	Integer part	Fractional part

Where, (0) is used to represent (+) and (1) is used to represent (-).

000000000101011 is 15 bit binary value for decimal 43 and 1010000000000000 is 16 bit binary value for fractional 0.625.

The advantage of using a fixed-point representation is performance and disadvantage is relatively limited range of values that they can represent.

So, it is usually inadequate for numerical analysis as it does not allow enough numbers and accuracy. A number whose representation exceeds 32 bits would have to be stored inexactly.



These are above smallest positive number and largest positive number which can be store in 32-bit representation as given above format.

Therefore, the smallest positive number is $2^{-16} \approx 0.000015$ approximate and the largest positive number is $(2^{15}-1) + (1-2^{-16}) = 2^{15}(1-2^{-16}) = 32768$, and gap between these numbers is 2^{-16}.

We can move the radix point either left or right with the help of only integer field is 1.

Floating-Point Representation –

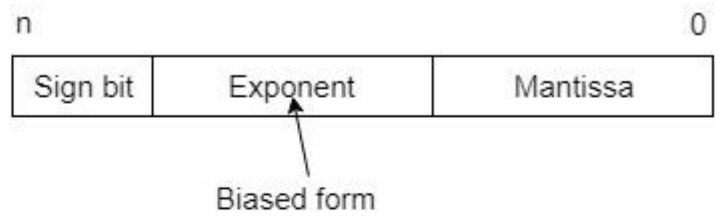
This representation does not reserve a specific number of bits for the integer part or the fractional part. Instead it reserves a certain number of bits for the number (called the mantissa or significant) and a certain number of bits to say where within that number the decimal place sits (called the exponent).

The floating number representation of a number has two parts: the first part represents a signed fixed point number called mantissa. The second part of designates the position of the decimal (or binary) point and is called the exponent. The fixed point mantissa may be fraction or an integer.

Only the mantissa m and the exponent e are physically represented in the register (including their sign).

A floating-point binary number is represented in a similar manner except that is uses base 2 for the exponent.

A floating-point number is said to be normalized if the most significant digit of the mantissa is 1.



So, actual number is $(-1)^{s}\ (1+m) \times 2^{(e-Bias)}$, where *s* is the sign bit, *m* is the mantissa, *e* is the exponent value, and *Bias* is the bias number.

Note that signed integers and exponent are represented by either sign representation, or one's complement representation, or two's complement representation.

The floating point representation is more flexible. Any non-zero number can be represented in the normalized form of $\pm\ (1.b_1b_2b_3\ \ldots)_2 \times 2^n$. This is normalized form of a number x.

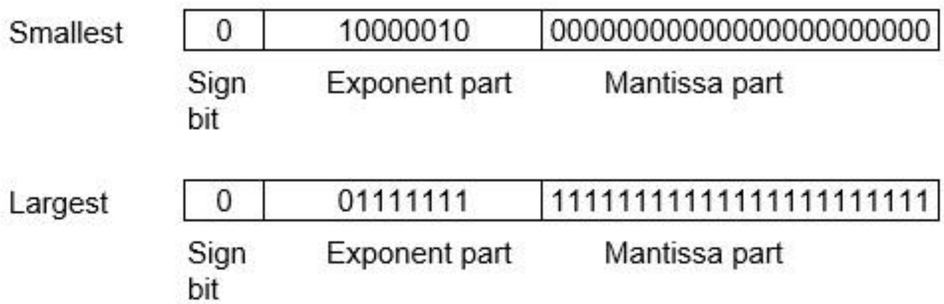
Example –Suppose number is using 32-bit format: the 1 bit sign bit, 8 bits for signed exponent, and 23 bits for the fractional part. The leading bit 1 is not stored (as it is always 1 for a normalized number) and is referred to as a *"hidden bit"*.

Then −53.5 is normalized as $-53.5 = (-110101.1)_2 = (-1.101011) \times 2^5$, which is represented as following below,

1	00000101	10101100000000000000000
Sign bit	Exponent part	Mantissa part

Where, 00000101 is the 8-bit binary value of exponent value +5. Note that 8-bit exponent is used to store integer exponents $-126 \le n \le 127$.

The smallest normalized positive number that fits into 32 bits is $(1.00000000000000000000000)_2 \times 2^{-126} = 2^{-126} \approx 1.18\times10^{-38}$, and largest normalized positive number that fits into 32 bits is $(1.11111111111111111111111)_2 \times 2^{127} = (2^{24}-1) \times 2^{104} \approx 3.40 \times 10^{38}$.



The precision of a floating point format is the number of positions reserved for binary digits plus one (for the hidden bit). In the examples considered here the precision is 23+1 = 24.

The gap between 1 and the next normalized floating-point number is known as machine epsilon. the gap is $(1+2^{-23})-1=2^{-23}$ for above example, but this is same as the smallest positive floating-point number because of non-uniform spacing unlike in the fixed point scenario.

Note that non-terminating binary numbers can be represented in floating point representation, e.g., $1/3 = (0.010101 \ldots)_2$ cannot be a floating-point number as its binary representation is non-terminating.

Characters Representation in Computers-

Computers are designed to work internally with numbers. In order to handle characters, we need to choose a number for each character. There are many ways to do this.

Alphanumeric codes are basically binary codes which are used to represent the alphanumeric data. As these codes represent data by characters, alphanumeric codes are also called “Character codes”.

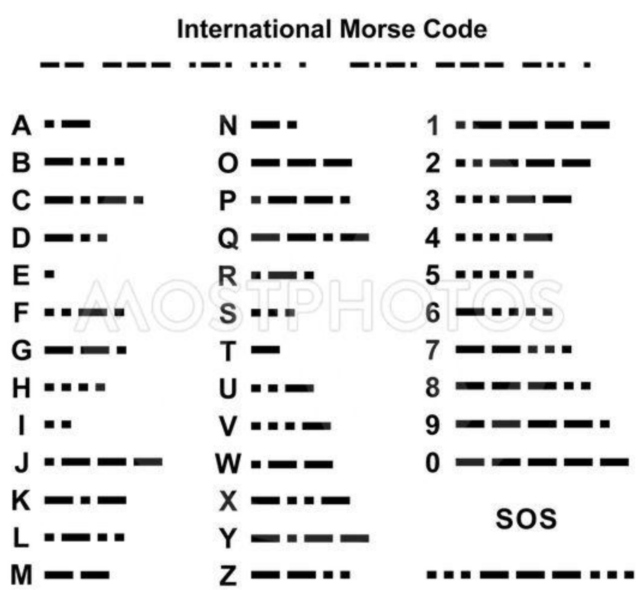
These codes can represent all types of data including alphabets, numbers, punctuation marks and mathematical symbols in the acceptable form by computers. These codes are implemented in I/O devices like key boards, monitors, printers etc.

In earlier days, punch cards are used to represent the alphanumeric codes. They are listed below as:

- MORSE code
- BAUDOT code
- HOLLERITH code
- ASCII code
- EBCDI code
- UNICODE

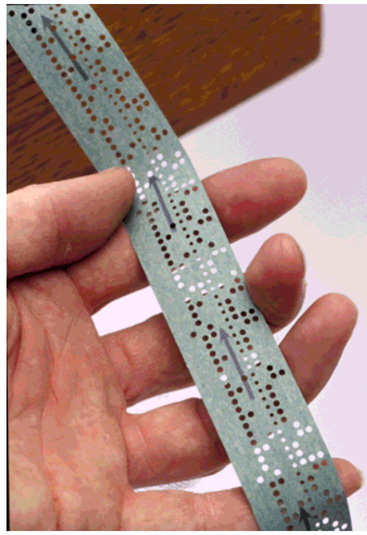
MORSE Code:

At the starting stage of computer and digital electronics era, Morse code is very popular and most used code. This was invented by Samuel F.B.Morse, in 1837. It was the first ever telegraphic code used in telecommunication. It is mainly used in Telegraph channels, Radio channels and in air traffic control units.



BOUDOT Code:

This code is invented by a French Engineer Emile Baudot, in 1870. It is a 5 unit code, means it uses 5 elements to represent an alphabet. It is also used in Telegraph networks to transfer Roman numeric.



HOLLERITH Code:

This code is developed by a company founded by Herman Hollerith in 1896. The 12 bit code used to punch cards according to the transmitting information is called “Hollerith code”.

TABLE 6.1 HOLLERITH CODE

Character	Punch at Rows	Character	Punch at Rows
1	1	O	11,6
2	2	P	11,7
3	3	Q	11,8
4	4	R	11,9
5	5	S	0,2
6	6	T	0,3
7	7	U	0,4
8	8	V	0,5
9	9	W	0,6
A	12,1	X	0,7
B	12,2	Y	0,8
C	12,3	Z	0,9
D	12,4	+	12
E	12,5	−	11
F	12,6	*	11,4,8
G	12,7	/	0,1
H	12,8	=	3,8
I	12,9	(	0,4,8
J	11,1	)	12,4,8
K	11,2		12,3,8
L	11,3	, comma	0,3,8
M	11,4	' quote	4,8
N	11,5	$	11,3,8

ASCII CODE:

ASCII means American Standard Code for Information Interchange. It is the world's most popular and widely used alphanumeric code. This code was developed and first published in 1967. ASCII code is a 7 bit code that means this code uses 27 = 128 characters.

This includes 26 lower case letters (a – z), 26 upper case letters (A – Z), 33 special characters and symbols (! @ # $ etc), 33 control characters (* – + / and %) and 10 digits (0 – 9).

Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char
0	00	000	0000000	NUL (null character)	32	20	040	0100000	space	64	40	100	1000000	@	96	60	140	1100000	`
1	01	001	0000001	SOH (start of header)	33	21	041	0100001	!	65	41	101	1000001	A	97	61	141	1100001	a
2	02	002	0000010	STX (start of text)	34	22	042	0100010	"	66	42	102	1000010	B	98	62	142	1100010	b
3	03	003	0000011	ETX (end of text)	35	23	043	0100011	#	67	43	103	1000011	C	99	63	143	1100011	c
4	04	004	0000100	EOT (end of transmission)	36	24	044	0100100	$	68	44	104	1000100	D	100	64	144	1100100	d
5	05	005	0000101	ENQ (enquiry)	37	25	045	0100101	%	69	45	105	1000101	E	101	65	145	1100101	e
6	06	006	0000110	ACK (acknowledge)	38	26	046	0100110	&	70	46	106	1000110	F	102	66	146	1100110	f
7	07	007	0000111	BEL (bell (ring))	39	27	047	0100111	'	71	47	107	1000111	G	103	67	147	1100111	g
8	08	010	0001000	BS (backspace)	40	28	050	0101000	(	72	48	110	1001000	H	104	68	150	1101000	h
9	09	011	0001001	HT (horizontal tab)	41	29	051	0101001	)	73	49	111	1001001	I	105	69	151	1101001	i
10	0A	012	0001010	LF (line feed)	42	2A	052	0101010	*	74	4A	112	1001010	J	106	6A	152	1101010	j
11	0B	013	0001011	VT (vertical tab)	43	2B	053	0101011	+	75	4B	113	1001011	K	107	6B	153	1101011	k
12	0C	014	0001100	FF (form feed)	44	2C	054	0101100	,	76	4C	114	1001100	L	108	6C	154	1101100	l
13	0D	015	0001101	CR (carriage return)	45	2D	055	0101101	-	77	4D	115	1001101	M	109	6D	155	1101101	m
14	0E	016	0001110	SO (shift out)	46	2E	056	0101110	.	78	4E	116	1001110	N	110	6E	156	1101110	n
15	0F	017	0001111	SI (shift in)	47	2F	057	0101111	/	79	4F	117	1001111	O	111	6F	157	1101111	o
16	10	020	0010000	DLE (data link escape)	48	30	060	0110000	0	80	50	120	1010000	P	112	70	160	1110000	p
17	11	021	0010001	DC1 (device control 1)	49	31	061	0110001	1	81	51	121	1010001	Q	113	71	161	1110001	q
18	12	022	0010010	DC2 (device control 2)	50	32	062	0110010	2	82	52	122	1010010	R	114	72	162	1110010	r
19	13	023	0010011	DC3 (device control 3)	51	33	063	0110011	3	83	53	123	1010011	S	115	73	163	1110011	s
20	14	024	0010100	DC4 (device control 4)	52	34	064	0110100	4	84	54	124	1010100	T	116	74	164	1110100	t
21	15	025	0010101	NAK (negative acknowledge)	53	35	065	0110101	5	85	55	125	1010101	U	117	75	165	1110101	u
22	16	026	0010110	SYN (synchronize)	54	36	066	0110110	6	86	56	126	1010110	V	118	76	166	1110110	v
23	17	027	0010111	ETB (end transmission block)	55	37	067	0110111	7	87	57	127	1010111	W	119	77	167	1110111	w
24	18	030	0011000	CAN (cancel)	56	38	070	0111000	8	88	58	130	1011000	X	120	78	170	1111000	x
25	19	031	0011001	EM (end of medium)	57	39	071	0111001	9	89	59	131	1011001	Y	121	79	171	1111001	y
26	1A	032	0011010	SUB (substitute)	58	3A	072	0111010	:	90	5A	132	1011010	Z	122	7A	172	1111010	z
27	1B	033	0011011	ESC (escape)	59	3B	073	0111011	;	91	5B	133	1011011	[	123	7B	173	1111011	{
28	1C	034	0011100	FS (file separator)	60	3C	074	0111100	<	92	5C	134	1011100	\	124	7C	174	1111100	|
29	1D	035	0011101	GS (group separator)	61	3D	075	0111101	=	93	5D	135	1011101	]	125	7D	175	1111101	}
30	1E	036	0011110	RS (record separator)	62	3E	076	0111110	>	94	5E	136	1011110	^	126	7E	176	1111110	~
31	1F	037	0011111	US (unit separator)	63	3F	077	0111111	?	95	5F	137	1011111	_	127	7F	177	1111111	DEL

Example:

If we want to print the name LONDAN, the ASCII code is?

The ASCII-7 equivalent of L = 100 1100

The ASCII-7 equivalent of O = 100 1111

The ASCII-7 equivalent of N = 100 1110

The ASCII-7 equivalent of D = 100 0100

The ASCII-7 equivalent of A = 100 0001

The ASCII-7 equivalent of N = 100 1110

The output of LONDAN in ASCII code is 1 0 0 1 1 0 0 1 0 0 1 1 1 1 1 0 0 1 1 1 0 1 0 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 1 1 1 0.

EBCDI CODE:

EBCDI stands for Extended Binary Coded Decimal Interchange code. This code is developed by IBM Inc Company.

It is an 8 bit code, so we can represent 28 = 256 characters by using EBCDI code.

This include all the letters and symbols like 26 lower case letters (a – z), 26 upper case letters (A – Z), 33 special characters and symbols (! @ # $ etc), 33 control characters (* – + / and % etc) and 10 digits (0 – 9).

In the EBCDI code, the 8 bit code the numbers are represented by 8421 BCD code preceded by 1111.

EBCDIC Format

Bits	5	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	6	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	7	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	8	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1 2 3 4																	
0 0 0 0		NUL	SOH	STX	ETX	PF	HT	LC	DEL			SMM	VT	FF	CR	SO	SI
0 0 0 1		DLE	DC1	DC2	DC3	RES	NL	BS	IL	CAN	EM	CC		IFS	IGS	IRS	IUS
0 0 1 0		DS	SOS	FS		BYP	LF	EOB	PRE			SM			ENQ	ACK	BEL
0 0 1 1				SYN		PN	RS	US	EOT					DC4	NAK		SUB
0 1 0 0		SP										¢		<	(	+	|
0 1 0 1		&										!	$	*	)	;	¬
0 1 1 0		–	/										,	%	_	>	?
0 1 1 1												:	#	@	'	=	"
1 0 0 0			a	b	c	d	e	f	g	h	i						
1 0 0 1			j	k	l	m	n	o	p	q	r						
1 0 1 0				s	t	u	v	w	x	y	z						
1 0 1 1																	
1 1 0 0			A	B	C	D	E	F	G	H	I						
1 1 0 1			J	K	L	M	N	O	P	Q	R						
1 1 1 0				S	T	U	V	W	X	Y	Z						
1 1 1 1		0	1	2	3	4	5	6	7	8	9						

PF	Punch off
HT	Horizontal tab
LC	Lower case
DEL	Delete
SP	Space
UC	Upper case
RES	Restore
NL	New line
BS	Backspace
IL	Idle
PN	Punch on
EOT	End of transmission
BYP	Bypass
LF	Line feed
EOB	End of block
PRE	Prefix (ESC)
RS	Reader stop
SM	Start message
DS	Digit select
SOS	Start of significance
IFS	Interchange file separator
IGS	Interchange group separator
IRS	Interchange record separator
IUS	Interchange unit separator
Others	Same as ASCII

UNICODE:

The draw backs in ASCII code and EBCDI code are that they are not compatible to all languages and they do not have sufficient set of characters to represent all types of data.

To overcome these drawback this UNICODE is developed.

UNICODE is the new concept of all digital coding techniques. In this we have a different character to represent every number.

It is the most advanced and sophisticated language with the ability to represent any type of data.

SO this is known as “Universal code”. It is a 16 bit code, with which we can represent 216 = 65536 different characters.

UNICODE is developed by the combined effort of UNICODE consortium and ISO (International organization for Standardization).

Graphic character symbol	Hexadecimal character value

0020	0 0030	@ 0040	P 0050	` 0060	p 0070	00A0	° 00B0	À 00C0	Ð 00D0	à 00E0	ð 00F0
! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	¡ 00A1	± 00B1	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
" 0022	2 0032	B 0042	R 0052	b 0062	r 0072	¢ 00A2	² 00B2	Â 00C2	Ò 00D2	â 00E2	ò 00F2
# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	£ 00A3	³ 00B3	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	¤ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	¥ 00A5	µ 00B5	Å 00C5	Õ 00D5	å 00E5	õ 00F5
& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	¦ 00A6	¶ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
' 0027	7 0037	G 0047	W 0057	g 0067	w 0077	§ 00A7	· 00B7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
(0028	8 0038	H 0048	X 0058	h 0068	x 0078	¨ 00A8	¸ 00B8	È 00C8	Ø 00D8	è 00E8	ø 00F8
) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	© 00A9	¹ 00B9	É 00C9	Ù 00D9	é 00E9	ù 00F9
* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	ª 00AA	º 00BA	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	« 00AB	» 00BB	Ë 00CB	Û 00DB	ë 00EB	û 00FB
, 002C	< 003C	L 004C	\ 005C	l 006C	| 007C	¬ 00AC	¼ 00BC	Ì 00CC	Ü 00DC	ì 00EC	ü 00FC
- 002D	= 003D	M 004D	] 005D	m 006D	} 007D	- 00AD	½ 00BD	Í 00CD	Ý 00DD	í 00ED	ý 00FD
. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	® 00AE	¾ 00BE	Î 00CE	Þ 00DE	î 00EE	þ 00FE
/ 002F	? 003F	O 004F	_ 005F	o 006F	007F	¯ 00AF	¿ 00BF	Ï 00CF	ß 00DF	ï 00EF	ÿ 00FF