

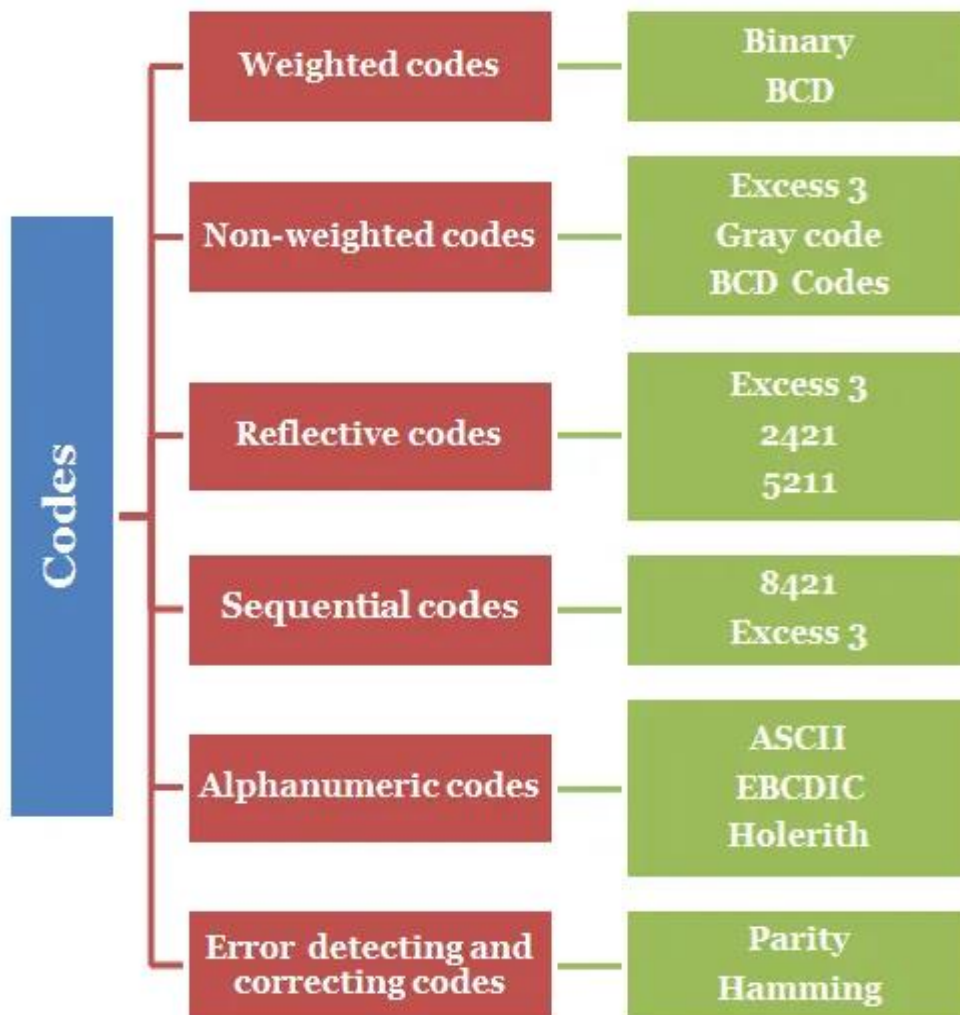
UNIT II

Binary Codes – 8421, 2421, Excess-3, Reflected Code – Error Detection Codes – Alphanumeric Code – Basic.

Logic Gates.

CLASSIFICATION OF BINARY CODES

The following figure shows the classification of binary codes.



Different Types of Binary Codes | BCD (8421), 2421, Excess-3, Gray

BCD Code:

- ✓ BCD code is an abbreviation for Binary coded Decimal codes.
- ✓ It is a numeric weighted code, in which each digit of a decimal number is represented by a separate group of 4-bits.
- ✓ There are several BCD codes like **8421**, **2421**, **3321**, **4221**, **5211**, **5311**, **5421**, etc.
- ✓ The most common and widely used BCD code is **8421** code.
- ✓ In 8421 code, the weights associated with 4 bits are 8, 4, 2, 1 from MSB to LSB.
- ✓ That is, the weight associated with 3rd bit is 8, the weight associated with 2nd bit is 4, the weight associated with 1st bit is 2 and the weight associated with 0th bit is 1.

The following table shows the 8421 code for 0-9 decimal numbers.

Decimal code	BCD Code			
	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

EXCESS-3 CODE:

- ✓ Excess-3 code is derived by simply adding 3 to each BCD number.
- ✓ It is a non-weighted code
- ✓ It is a sequential code and reflective code.

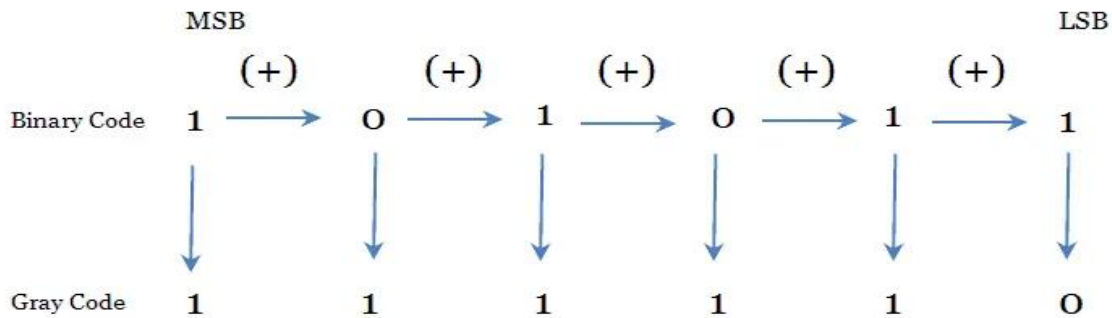
Decimal code	Excess-3 Code
0	0 0 1 1
1	0 1 0 0
2	0 1 0 1
3	0 1 1 0
4	0 1 1 1
5	1 0 0 0
6	1 0 0 1
7	1 0 1 0
8	1 0 1 1
9	1 1 0 0

THE GRAY CODE:

BINARY-TO-GRAY CODE CONVERSION

- ✓ The MSB in the Gray code is the same as corresponding MSB in the binary number.
- ✓ Going from left to right, add each adjacent pair of binary code bits to get the next Gray code bit. **Discard carries.**

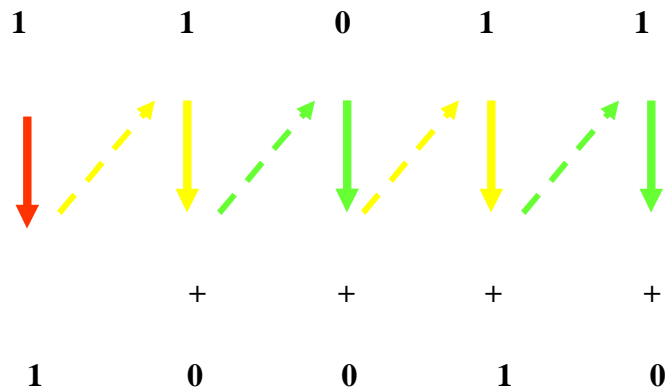
Ex: Convert $(101011)_2$ to Gray code



GRAY-TO-BINARY CONVERSION

- ✓ The MSB in the binary code is the same as the corresponding bit in the Gray code.
- ✓ Add each binary code bit generated to the Gray code bit in the next adjacent position.
Discard carries.

Ex: Convert the Gray code word 11011 to binary



APPLICATION OF GRAY CODE:

The gray code is used in a few specific applications.

- ✓ The main applications include being used in analog to digital converters, as well as being used for error correction in digital communication.
- ✓ Gray code is used to minimize errors in converting analog signals to digital signals.

ADVANTAGES OF GRAY CODE:

- ✓ Better for error minimization in converting analog signals to digital signals.
- ✓ Reduces the occurrence of “Hamming Walls” (an undesirable state) when used in genetic algorithms.
- ✓ Can be used to in to minimize a logic circuit.
- ✓ Useful in clock domain crossing.

DISADVANTAGES OF GRAY CODE:

- ✓ Not suitable for arithmetic operations.
- ✓ Limited practical use outside of a few specific applications.

2421 CODE:

This code also a 4 bit application code where the binary weights carry 2, 4, 2, 1 from left to right.

DECIMAL NUMBER	BINARY NUMBER	2421 CODE
0	0	0000
1	1	0001
2	10	0010
3	11	0011
4	100	0100
5	101	1011
6	110	1100
7	111	1101
8	1000	1110
9	1001	1111

ERROR DETECTION CODE:

- ✓ The error detection codes are the code used for detecting the error in the received data **bit stream**.
- ✓ In these codes, some bits are included appended to the original **bit stream**.
- ✓ Error detecting codes encode the message before sending it over the noisy channels.
- ✓ The encoding scheme is performed in such a way that the decoder at the receiving can find the errors easily in the receiving data with a higher chance of success.

Example – Parity code, Hamming code.

1. PARITY CODE:

- ✓ It is easy to include append one parity bit either to the left of MSB or to the right of LSB of original bit stream.
- ✓ There are two types of parity codes, namely
 - ◆ even parity code and
 - ◆ odd parity code based on the type of parity being chosen.
- ◆ **Even Parity Code**
 - ✓ The value of even parity bit should be zero, if even number of ones present in the binary code. Otherwise, it should be one.
 - ✓ So that, even number of ones present in **even parity code**.
 - ✓ Even parity code contains the data bits and even parity bit.
 - ✓ The following table shows the **even parity codes** corresponding to each 3-bit binary code.
 - ✓ Here, the even parity bit is included to the right of LSB of binary code.

Binary Code	Even Parity bit	Even Parity Code
000	0	0000
001	1	0011
010	1	0101
011	0	0110
100	1	1001
101	0	1010
110	0	1100
111	1	1111

Here, the number of bits present in the even parity codes is 4. So, the possible even number of ones in these even parity codes are 0, 2 & 4.

- ✓ If the other system receives one of these even parity codes, then there is no error in the received data. The bits other than even parity bit are same as that of binary code.
- ✓ If the other system receives other than even parity codes, then there will be an errors in the received data. In this case, we can't predict the original binary code because we don't know the bit positions of error.

Therefore, even parity bit is useful only for detection of error in the received parity code. But, it is not sufficient to correct the error.

◆ **Odd Parity Code**

- ✓ The value of odd parity bit should be zero, if odd number of ones present in the binary code. Otherwise, it should be one. So that, odd number of ones present in **odd parity code**. Odd parity code contains the data bits and odd parity bit.
- ✓ The following table shows the **odd parity codes** corresponding to each 3-bit binary code. Here, the odd parity bit is included to the right of LSB of binary code.

Binary Code	Odd Parity bit	Odd Parity Code
000	1	0001
001	0	0010
010	0	0100
011	1	0111
100	0	1000
101	1	1011
110	1	1101
111	0	1110

Here, the number of bits present in the odd parity codes is 4. So, the possible odd number of ones in these odd parity codes is 1 & 3.

- ✓ If the other system receives one of these odd parity codes, then there is no error in the received data. The bits other than odd parity bit are same as that of binary code.
- ✓ If the other system receives other than odd parity codes, then there is an errors in the received data. In this case, we can't predict the original binary code because we don't know the bit positions of error.

Therefore, odd parity bit is useful only for detection of error in the received parity code. But, it is not sufficient to correct the error.

ASCII CODE

- ✓ The ASCII stands for **American Standard Code for Information Interchange**.
- ✓ The ASCII code is an alphanumeric code used for data communication in digital computers.
- ✓ The ASCII is a 7-bit code capable of representing 2^7 or 128 number of different characters.
- ✓ The ASCII code is made up of a three-bit group, which is followed by a four-bit code.

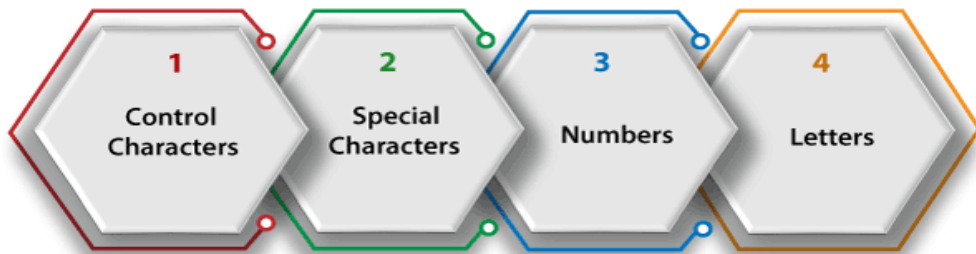
Representation of ASCII Code



- ✓ The ASCII Code is a 7 or 8-bit alphanumeric code.
- ✓ This code can represent 127 unique characters.
- ✓ The ASCII code starts from 00h to 7Fh. In this, the code from 00h to 1Fh is used for control characters, and the code from 20h to 7Fh is used for graphic symbols.
- ✓ The 8-bit code holds ASCII, which supports 256 symbols where math and graphic symbols are added.
- ✓ The range of the extended ASCII is 80h to FFh.

The ASCII characters are classified into the following groups:

ASCII Characters



1. CONTROL CHARACTERS

- ✓ The non-printable characters used for sending commands to the PC or printer are known as control characters.
- ✓ We can set tabs, and line breaks functionality by this code.
- ✓ The control characters are based on **telex** technology. Nowadays, it's not so much popular in use.
- ✓ The character from 0 to 31 and 127 comes under control characters.

2. SPECIAL CHARACTERS

- ✓ All printable characters that are neither numbers nor letters come under the special characters.
- ✓ These characters contain technical, punctuation, and mathematical characters with space also.
- ✓ The character from 32 to 47, 58 to 64, 91 to 96, and 123 to 126 comes under this category.

3. NUMBERS CHARACTERS

This category of ASCII code contains **Ten Arabic numerals from 0 to 9.**

4. LETTERS CHARACTERS

- ✓ In this category, two groups of letters are contained, i.e., the group of uppercase letters and the group of lowercase letters.
- ✓ The range from 65 to 90 and 97 to 122 comes under this category.

ASCII Table

The values are typically represented in ASCII code tables in **Decimal, Binary, And Hexadecimal Form.**

Binary	Hexa decimal	Decimal	ASCII Symbol	Description	Group
0000000	0	0	NUL	The null character encourage the device to do nothing	Control Character
0000001	1	1	SOH	The symbol SOH(Starts of heading) Initiates the header.	Control Character
0000010	2	2	STX	The symbol STX(Start of Text) ends the header and marks the beginning of a message.	Control Character
0000011	3	3	ETX	The symbol ETX(End of Text) indicates the end of the message.	Control Character
0000100	4	4	EOT	The EOT(end of text) symbol marks the end of a completes transmission	Control Character
0000101	5	5	ENQ	The ENQ(Enquiry) symbol is a request that requires a response	Control Character
0000110	6	6	ACK	The ACK(Acknowledge) symbol is a positive answer to the request.	Control Character
0000111	7	7	BEL	The BEL(Bell) symbol triggers a beep.	Control Character
0001000	8	8	BS	Lets the cursor move back one step (Backspace)	Control Character
0001001	9	9	TAB (HT)	A horizontal tab that moves the cursor within a row to the next predefined position (Horizontal Tab)	Control Character
0001010	A	10	LF	Causes the cursor to jump to the next line (Line Feed)	Control Character
0001011	B	11	VT	The vertical tab lets the cursor jump to a predefined line (Vertical Tab)	Control Character
0001100	C	12	FF	Requests a page break (Form Feed)	Control Character
0001101	D	13	CR	Moves the cursor back to the first position of the line (Carriage Return)	Control Character
0001110	E	14	SO	Switches to a special presentation (Shift Out)	Control Character
0001111	F	15	SI	Switches the display back to the normal state (Shift In)	Control Character
0010000	10	16	DLE	Changes the meaning of the following characters (Data Link Escape)	Control Character
0010001	11	17	DC1	Control characters assigned depending on the device used (Device Control)	Control Character
0010010	12	18	DC2	Control characters assigned depending on the device used (Device Control)	Control Character

Binary	Hexa decimal	Decimal	ASCII Symbol	Description	Group
0010011	13	19	DC3	Control characters assigned depending on the device used (Device Control)	Control Character
0010100	14	20	DC4	Control characters assigned depending on the device used (Device Control)	Control Character
0010101	15	21	NAK	The negative response to a request (Negative Acknowledge)	Control Character
0010110	16	22	SYN	Synchronizes a data transfer, even if no signals are transmitted (Synchronous Idle)	Control Character
0010111	17	23	ETB	Marks the end of a transmission block (End of Transmission Block)	Control Character
0011000	18	24	CAN	Makes it clear that transmission was faulty and the data must be discarded (Cancel)	Control Character
0011001	19	25	EM	Indicates the end of the storage medium (End of Medium)	Control Character
0011010	1A	26	SUB	Replacement for a faulty sign (Substitute)	Control Character
0011011	1B	27	ESC	Initiates an escape sequence and thus gives the following characters a special meaning (Escape)	Control Character
0011100	1C	28	FS	File separator.	Control Character
0011101	1D	29	GS	Group separator.	Control Character
0011110	1E	30	RS	Record separator.	Control Character
0011111	1F	31	US	Unit separator.	Control Character
0100000	20	32	SP	Blank space	Special Character
0100001	21	33	!	Exclamation mark	Special Character
0100010	22	34	“	Only quotes above	Special Character
0100011	23	35	#	Pound sign	Special Character
0100100	24	36	\$	Dollar sign	Special Character
0100101	25	37	%	Percentage sign	Special Character
0100110	26	38	&	Commercial and	Special Character
0100111	27	39	‘	Apostrophe	Special Character

Binary	Hexadecimal	Decimal	ASCII Symbol	Description	Group
0101000	28	40	(Left bracket	Special Character
0101001	29	41)	Right bracket	Special Character
0101010	2A	42	*	Asterisk	Special Character
0101011	2B	43	+	Plus symbol	Special Character
0101100	2C	44	,	Comma	Special Character
0101101	2D	45	-	Dash	Special Character
0101110	2E	46	.	Full stop	Special Character
0101111	2F	47	/	Forward slash	Special Character
0110000	30	48	0		Numbers
0110001	31	49	1		Numbers
0110010	32	50	2		Numbers
0110011	33	51	3		Numbers
0110100	34	52	4		Numbers
0110101	35	53	5		Numbers
0110110	36	54	6		Numbers
0110111	37	55	7		Numbers
0111000	38	56	8		Numbers
0111001	39	57	9		Numbers
0111010	3A	58	:	Colon	Special characters
0111011	3B	59	;	Semicolon	Special characters
0111100	3C	60	<	Small than bracket	Special characters
0111101	3D	61	=	Equals sign	Special characters
0111110	3E	62	>	Bigger than symbol	Special characters
0111111	3F	63	?	Question mark	Special characters
1000000	40	64	@	At symbol	Special characters
1000001	41	65	A		Capital letters

Binary	Hexadecimal	Decimal	ASCII Symbol	Description	Group
1000010	42	66	B		Capital letters
1000011	43	67	C		Capital letters
1000100	44	68	D		Capital letters
1000101	45	69	E		Capital letters
1000110	46	70	F		Capital letters
1000111	47	71	G		Capital letters
1001000	48	72	H		Capital letters
1001001	49	73	I		Capital letters
1001010	4A	74	J		Capital letters
1001011	4B	75	K		Capital letters
1001100	4C	76	L		Capital letters
1001101	4D	77	M		Capital letters
1001110	4E	78	N		Capital letters
1001111	4F	79	O		Capital letters
1010000	50	80	P		Capital letters
1010001	51	81	Q		Capital letters
1010010	52	82	R		Capital letters
1010011	53	83	S		Capital letters
1010100	54	84	T		Capital letters
1010101	55	85	U		Capital letters
1010110	56	86	V		Capital letters
1010111	57	87	W		Capital letters

Binary	Hexa decimal	Decimal	ASCII Symbol	Description	Group
1011000	58	88	X		Capital letters
1011001	59	89	Y		Capital letters
1011010	5A	90	Z		Capital letters
1011011	5B	91	[Left square bracket	Special character
1011100	5C	92	\	Inverse/backward slash	Special character
1011101	5D	93]	Right square bracket	Special character
1011110	5E	94	^	Circumflex	Special character
1011111	5F	95	_	Underscore	Special character
1100000	60	96	`	Gravis (backtick)	Special character
1100001	61	97	a		Lowercase letters
1100010	62	98	b		Lowercase letters
1100011	63	99	c		Lowercase letters
1100100	64	100	d		Lowercase letters
1100101	65	101	e		Lowercase letters
1100110	66	102	f		Lowercase letters
1100111	67	103	g		Lowercase letters
1101000	68	104	h		Lowercase letters
1101001	69	105	i		Lowercase letters
1101010	6A	106	j		Lowercase letters
1101011	6B	107	k		Lowercase letters
1101100	6C	108	l		Lowercase letters
1101101	6D	109	m		Lowercase letters

Binary	Hexa decimal	Decimal	ASCII Symbol	Description	Group
1101110	6E	110	n		Lowercase letters
1101111	6F	111	o		Lowercase letters
1110000	70	112	p		Lowercase letters
1110001	71	113	q		Lowercase letters
1110010	72	114	r		Lowercase letters
1110011	73	115	s		Lowercase letters
1110100	74	116	t		Lowercase letters
1110101	75	117	u		Lowercase letters
1110110	76	118	v		Lowercase letters
1110111	77	119	w		Lowercase letters
1111000	78	120	x		Lowercase letters
1111001	79	121	y		Lowercase letters
1111010	7A	122	z		Lowercase letters
1111011	7B	123	{	Left curly bracket	Special characters
1111100	7C	124		Vertical line	Special characters
1111101	7D	125	}	Right curly brackets	Special characters
1111110	7E	126	~	Tilde	Special characters
1111111	7F	127	DEL	The DEL (Delete) symbol deletes a character. This is a control character that consists of the same number in all positions.	Control characters

Example 1: (10010101100001111011011000011010100111000011011111101001 1101110 11101
001000000011000101100100110011)₂

Step 1: In the first step, we we make the groups of 7-bits because the ASCII code is 7 bit.

1001010 1100001 1110110 1100001 1010100 1110000 1101111 1101001 1101110 1110100 1000000
0110001 0110010 0110011

Step 2: Then, we find the equivalent decimal number of the binary digits either from the **ASCII** table or **64 32 16 8 4 2 1** scheme.

BINARY	DECIMAL
64 32 16 8 4 2 1 1 0 0 1 0 1 0	64+8+2=74
64 32 16 8 4 2 1 1 1 0 0 0 0 1	64+32+1=94
64 32 16 8 4 2 1 1 1 1 0 1 1 0	64+32+16+4+2=118
64 32 16 8 4 2 1 1 1 0 0 0 0 1	64+32+1=97
64 32 16 8 4 2 1 1 0 1 0 1 0 0	64+16+4=84
64 32 16 8 4 2 1 1 1 1 0 0 0 0	64+32+16=112
64 32 16 8 4 2 1 1 1 0 1 1 1 1	64+32+8+4+2+1=111
64 32 16 8 4 2 1 1 1 0 1 0 0 1	64+32+8+1=105
64 32 16 8 4 2 1 1 1 0 1 1 1 0	64+32+8+4+2=110
64 32 16 8 4 2 1 1 1 1 0 1 0 0	64+32+16+4=116
64 32 16 8 4 2 1 1 0 0 0 0 0 0	64
64 32 16 8 4 2 1 0 1 1 0 0 0 1	32+16+1=49
64 32 16 8 4 2 1 0 1 1 0 0 1 0	32+16+2=50
64 32 16 8 4 2 1 0 1 1 0 0 1 1	32+16+2+1=51

Step 3: Last, we find the equivalent symbol of the decimal number from the ASCII table.

S.NO.	DECIMAL	SYMBOL	S.NO.	DECIMAL	SYMBOL
1	74	J	8	105	i
2	94	a	9	110	n
3	118	v	10	116	t
4	97	a	11	64	@
5	84	T	12	49	1
6	112	p	13	50	2
7	111	o	14	51	3

LOGIC GATES:

Logic Gates may be defined as

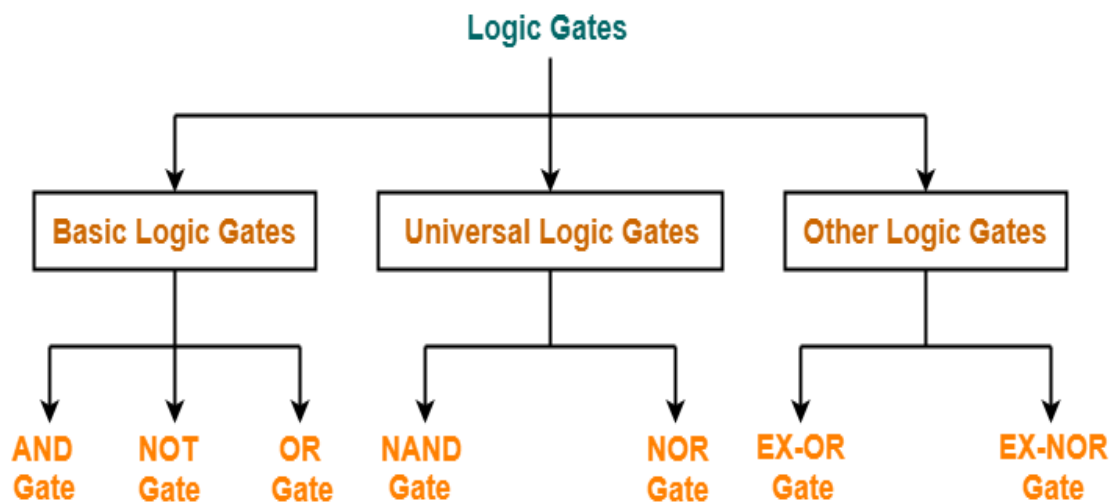
Logic gates are the digital circuits capable of performing a particular logic function by operating on a number of binary inputs.

OR

Logic gates are the basic building blocks of any digital circuit.

Types Of Logic Gates:

Logic gates can be broadly classified as



Types of Logic Gates

1. BASIC LOGIC GATES:

1. AND Gate

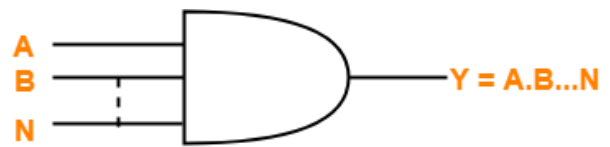
- ✓ The output of AND gate is high ('1') if all of its inputs are high ('1').
- ✓ The output of AND gate is low ('0') if any one of its inputs is low ('0').

Logic Symbol

The logic symbol for AND Gate is as shown below



2-Input AND Gate



N-Input AND Gate

Truth Table

The truth table for AND Gate is as shown below

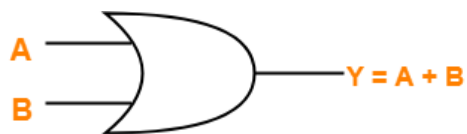
A	B	Y = A.B
0	0	0
0	1	0
1	0	0
1	1	1

2. OR Gate

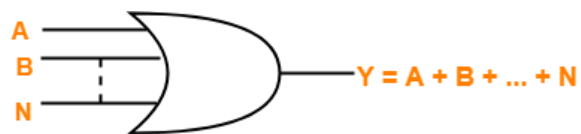
- ✓ The output of OR gate is high ('1') if any one of its inputs is high ('1').
- ✓ The output of OR gate is low ('0') if all of its inputs are low ('0').

Logic Symbol

The logic symbol for OR Gate is as shown below



2-Input OR Gate



N-Input OR Gate

Truth Table

The truth table for OR Gate is as shown below

A	B	Y = A + B
0	0	0
0	1	1
1	0	1
1	1	1

3. NOT Gate

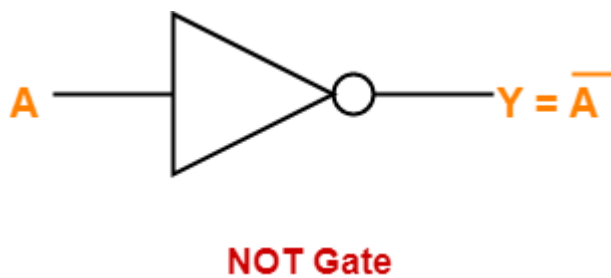
- ✓ The output of NOT gate is high ('1') if its input is low ('0').
- ✓ The output of NOT gate is low ('0') if its input is high ('1').

From here

- ✓ It is clear that NOT gate simply inverts the given input.
- ✓ Since NOT gate simply inverts the given input, therefore it is also known as **Inverter Gate**.

Logic Symbol

The logic symbol for NOT Gate is as shown below



Truth Table

The truth table for NOT Gate is as shown below

A	Y = A'
0	1
1	0

2. UNIVERSAL LOGIC GATES:

Universal logic gates are the logic gates that are capable of implementing any Boolean function without requiring any other type of gate.

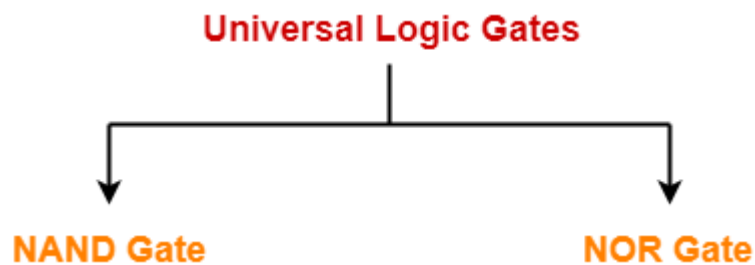
They are called as “**Universal Gates**” because-

- ✓ They can realize all the binary operations.
- ✓ All the basic logic gates can be derived from them.

They have the following properties-

- ✓ Universal gates are not associative in nature.
- ✓ Universal gates are commutative in nature.

There are following two universal logic gates-



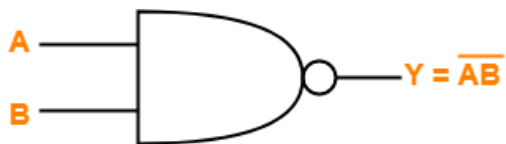
1. NAND Gate
2. NOR Gate

1. NAND Gate

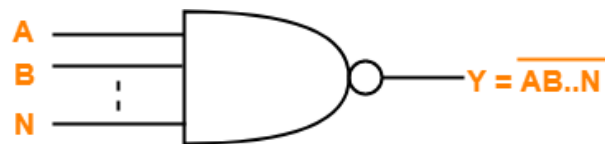
- ✓ A NAND Gate is constructed by connecting a NOT Gate at the output terminal of the AND Gate.
- ✓ The output of NAND gate is high ('1') if at least one of its inputs is low ('0').
- ✓ The output of NAND gate is low ('0') if all of its inputs are high ('1').

Logic Symbol

The logic symbol for NAND Gate is as shown below-



2-Input NAND Gate



N-Input NAND Gate

Truth Table

The truth table for NAND Gate is as shown below

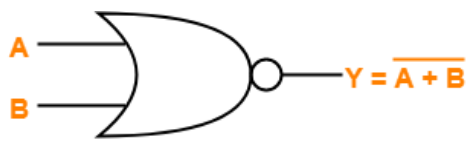
A	B	$Y = (A.B)'$ OR $Y = \overline{(A.B)}$
0	0	1
0	1	1
1	0	1
1	1	0

2. NOR Gate

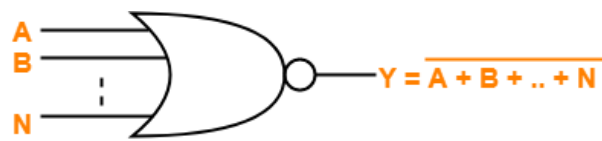
- ✓ A NOR Gate is constructed by connecting a NOT Gate at the output terminal of the OR Gate.
- ✓ The output of OR gate is high ('1') if all of its inputs are low ('0').
- ✓ The output of OR gate is low ('0') if any of its inputs is high ('1').

Logic Symbol

The logic symbol for NOR Gate is as shown below



2-Input NOR Gate



N-Input NOR Gate

Truth Table

The truth table for NOR Gate is as shown below

A	B	$Y = (A+B)'$ OR $Y = \overline{(A+B)}$
0	0	1
0	1	0
1	0	0
1	1	0

3. OTHER LOGIC GATES:

1. XOR Gate

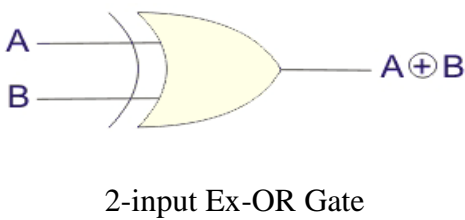
The Exclusive-OR or 'Ex-OR' gate is a digital logic gate with more than two inputs and gives only one output.

- ✓ The output of XOR Gate is 'High' if either input is 'High'.
- ✓ The output is 'Low' if both the inputs are 'High' or if both the inputs are 'Low'.

The symbol and truth table of the XOR gate can be shown as:

Logical Symbol of XOR Gate

An XOR gate is logically represented as,

Symbol	Truth Table		
 <p>2-input Ex-OR Gate</p>	A	B	Y
	0	0	0
	1	0	1
	0	1	1
	1	1	0
Boolean Expression $Y = A \oplus B$	A OR B but NOT BOTH gives Y		

2. XNOR Gate

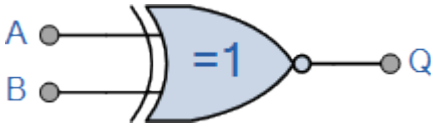
The Exclusive-NOR or 'EX-NOR' gate is a digital logic gate with more than two inputs and gives only one output.

- ✓ The output of XNOR Gate is 'High' if both the inputs are 'High' or if both the inputs are 'Low.'
- ✓ The output is 'Low' if either of the input is 'Low'.

The symbol and truth table of an XNOR gate can be given as:

Logical Symbol of XNOR Gate

An XNOR gate is logically represented as

Symbol	Truth Table		
 <p style="text-align: center;">2-input Ex-NOR Gate</p>	A	B	Y
	0	0	1
	0	1	0
	1	0	0
	1	1	1
Boolean Expression $Y = \overline{A \oplus B}$	If A AND B the SAME gives Y		

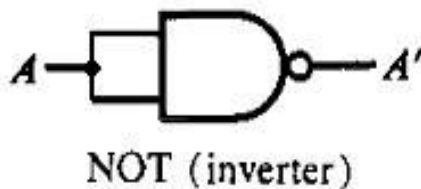
REALIZATION OF LOGIC FUNCTIONS WITH THE HELP OF UNIVERSAL GATES- NAND GATE

- ✓ NAND gate is actually a combination of two logic gates: **AND** gate followed by **NOT** gate.
- ✓ So its output is complement of the output of an AND gate.
- ✓ This gate can have minimum two inputs, output is always one. By using only NAND gates, we can realize all logic functions: AND, OR, NOT, X-OR, X-NOR, NOR. So this gate is also called universal gate.

1. NAND gates as NOT gate:

A NOT produces complement of the input. It can have only one input, tie the inputs of a NAND gate together. Now it will work as a NOT gate. Its output is

$$Y = (A.A)'$$



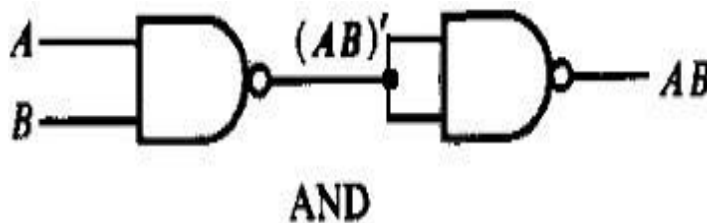
=> $Y = (A)'$

2. NAND gates as AND gate

A NAND produces complement of AND gate. So, if the output of a NAND gate is inverted, overall output will be that of an AND gate.

$$Y = ((A.B)')'$$

=> $Y = (A.B)$

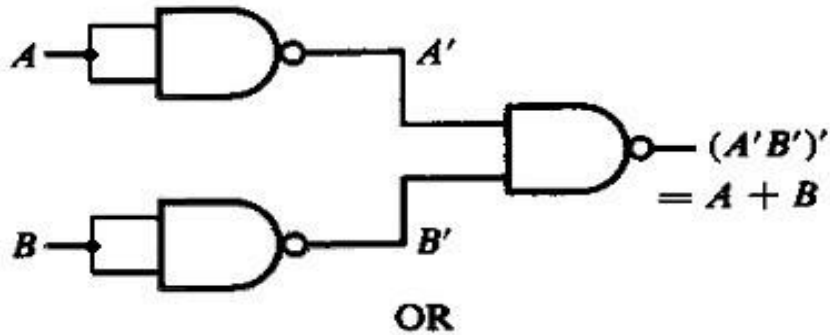


3. NAND gates as OR gate

From DeMorgan's theorems: $(A.B)' = A' + B'$

$$\Rightarrow (A'.B')' = A'' + B'' = A + B$$

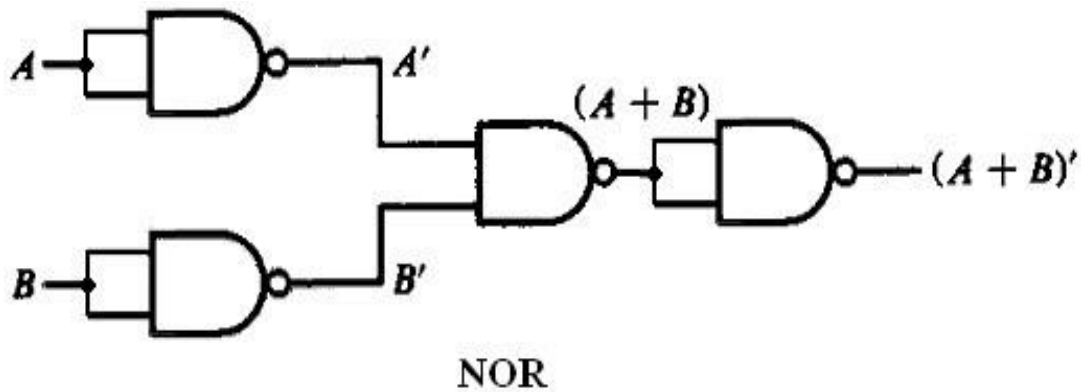
So, give the inverted inputs to a NAND gate, obtain OR operation at output.



4. NAND gates as NOR gate

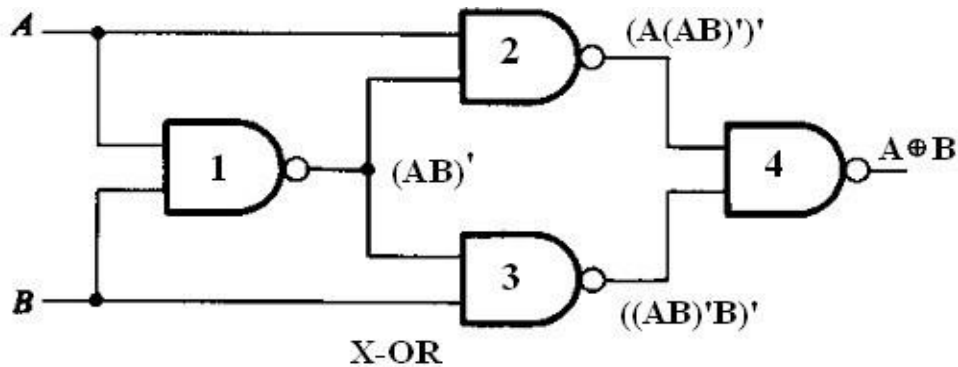
- ✓ A NOR gate is an OR gate followed by NOT gate.
- ✓ So connect the output of OR gate to a NOT gate, overall output is that of a NOR gate.

$$Y = (A + B)'$$



5. NAND gates as X-OR gate

- ✓ The output of a two input X-OR gate is shown by: $Y = A'B + AB'$.
- ✓ This can be achieved with the logic diagram shown in the left side.



Gate No.	Inputs	Output
1	A, B	$(AB)'$
2	A, $(AB)'$	$(A(AB)')'$
3	$(AB)'$, B	$(B(AB)')'$
4	$(A(AB)')'$, $(B(AB)')'$	$A'B + AB'$

Now the output from gate no. 4 is the overall output of the configuration.

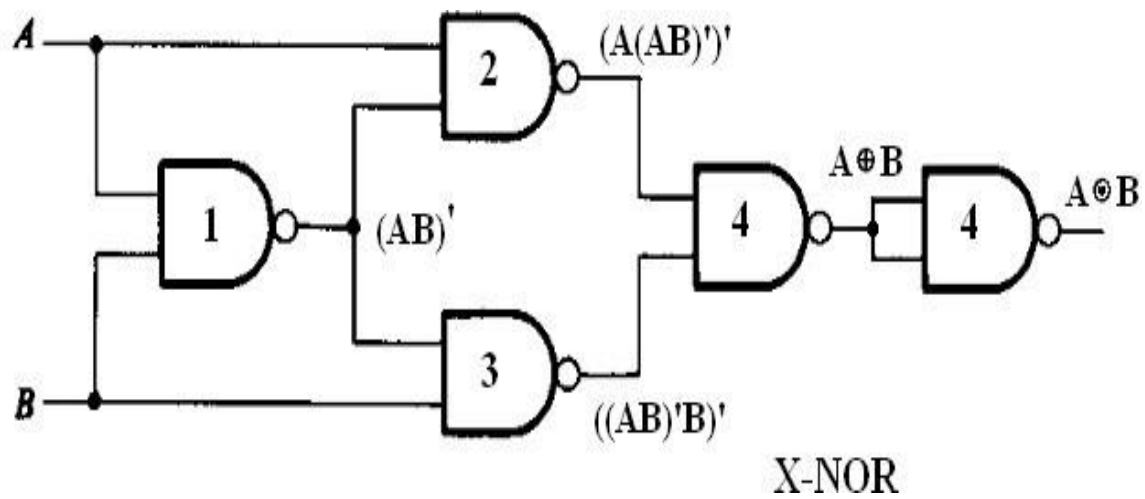
$$\begin{aligned}
 Y &= ((A(AB)')' (B(AB)')')' \\
 &= (A(AB)')'' + (B(AB)')'' \\
 &= (A(AB)') + (B(AB)') \\
 &= (A(A' + B)') + (B(A' + B)) \\
 &= (AA' + AB') + (BA' + BB') \\
 &= (0 + AB' + BA' + 0) \\
 &= AB' + BA'
 \end{aligned}$$

$$\Rightarrow \quad Y = AB' + A'B$$

6. NAND gates as X-NOR gate

- ✓ X-NOR gate is actually X-OR gate followed by NOT gate.
- ✓ So give the output of X-OR gate to a NOT gate, overall output is that of an X-NOR gate.

$$Y = AB + A'B'$$



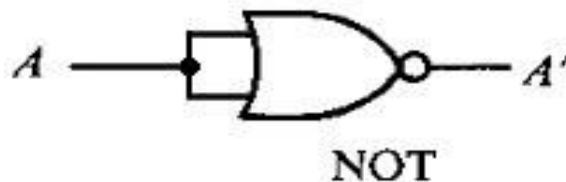
REALIZATION OF LOGIC FUNCTIONS WITH THE HELP OF UNIVERSAL GATES-

NOR GATE

1. NOR gates as NOT gate

- ✓ A NOT produces complement of the input.
- ✓ It can have only one input, tie the inputs of a NOR gate together.
- ✓ Now it will work as a NOT gate. Its output is

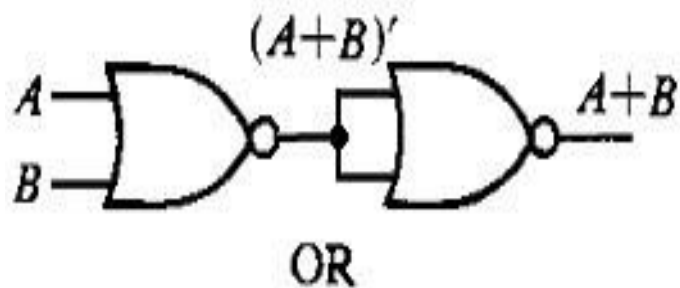
$$\begin{aligned} Y &= (A+A)' \\ \Rightarrow Y &= (A)' \end{aligned}$$



2. NOR gates as OR gate

- ✓ A NOR produces complement of OR gate.
- ✓ So, if the output of a NOR gate is inverted.
- ✓ Overall output will be that of an OR gate.

$$\begin{aligned} Y &= ((A+B)')' \\ \Rightarrow Y &= (A+B) \end{aligned}$$

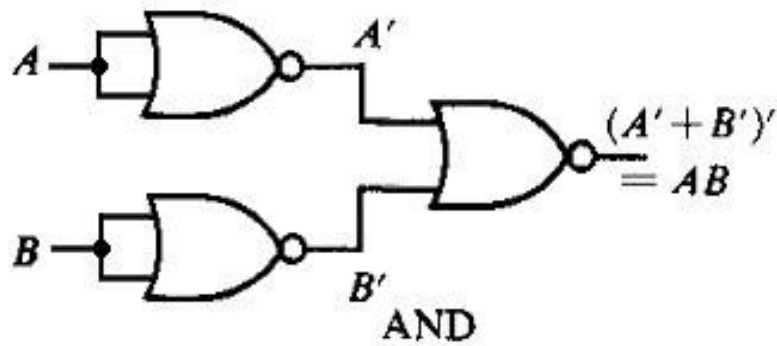


3. NOR gates as AND gate

From DeMorgan's theorems: $(A+B)' = A'B'$

$$\Rightarrow (A'+B')' = A''B'' = AB$$

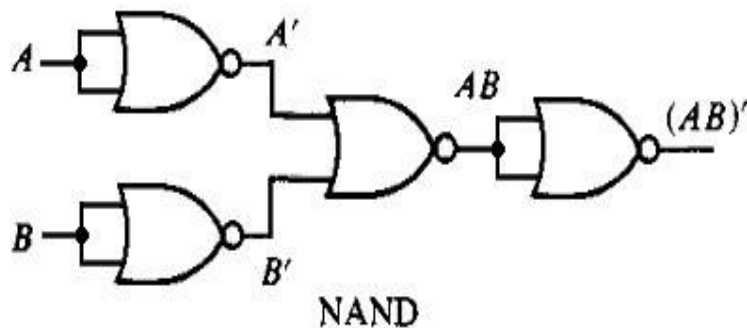
So, give the inverted inputs to a NOR gate, obtain AND operation at output.



4. NOR gates as NAND gate

- ✓ A NAND gate is an AND gate followed by NOT gate.
- ✓ So connect the output of AND gate to a NOT gate.
- ✓ Overall output is that of a NAND gate.

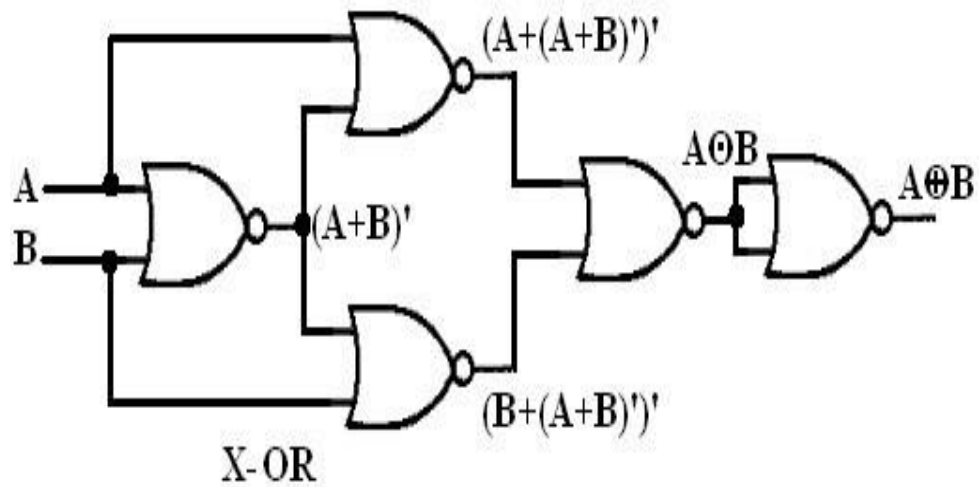
$$Y = (AB)'$$



5.NOR gates as X-OR gate

- ✓ X-OR gate is actually X-NOR gate followed by NOT gate.
- ✓ So give the output of X-NOR gate to a NOT gate.
- ✓ Overall output is that of an X-OR gate.

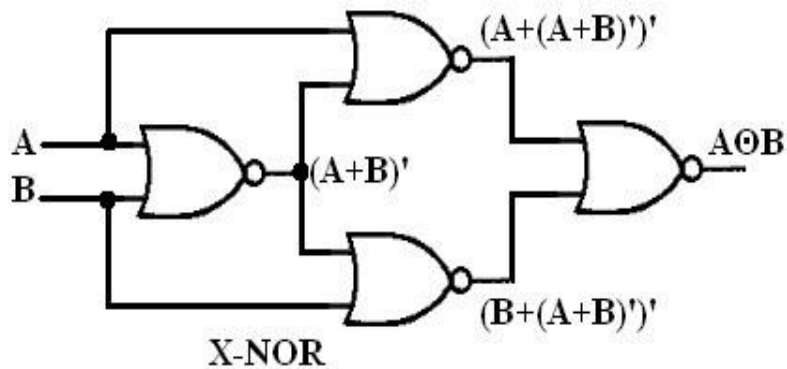
$$Y = A'B + AB'$$



6. NOR gates as X-NOR gate

The output of a two input X-NOR gate is shown by: $Y = AB + A'B'$.

This can be achieved with the logic diagram shown in the left side.



Gate No.	Inputs	Output
1	A, B	$(A + B)'$
2	A, $(A + B)'$	$(A + (A+B))'$
3	$(A + B)'$, B	$(B + (A+B))'$
4	$(A + (A + B))'$, $(B + (A+B))'$	$AB + A'B'$

Now the output from gate no. 4 is the overall output of the configuration.

$$\begin{aligned}
 Y &= ((A + (A+B))' (B + (A+B))')' \\
 &= (A+(A+B))'' \cdot (B+(A+B))'' \\
 &= (A+(A+B)) \cdot (B+(A+B)) \\
 &= (A+A'B') \cdot (B+A'B') \\
 &= (A + A') \cdot (A + B') \cdot (B+A') \cdot (B+B') \\
 &= 1 \cdot (A+B') \cdot (B+A') \cdot 1 \\
 &= (A+B') \cdot (B+A') \\
 &= A \cdot (B + A') + B' \cdot (B+A') \\
 &= AB + AA' + B'B + B'A' \\
 &= AB + 0 + 0 + B'A' \\
 &= AB + B'A'
 \end{aligned}$$

=> $Y = AB + A'B'$
