SEMESTER EXAM SERIES

DIGITAL ELECTRONICS

IN 6 HOURS + NOTES

Video chapters

Chapter-1 (Boolean Algebra & Logic Gates): Introduction to Digital Electronics, Advantage of Digital System, Boolean Algebra, Laws, Not, OR, AND, NOR, NAND, EX-OR, EX-NOR, AND-OR, OR-AND, Universal Gate Functionally Complete Function.

- **Chapter-2 (**Boolean Expressions**):** Boolean Expressions, SOP(Sum of Product), SOP Canonical Form, POS(Product of Sum), POS Canonical Form, No of Functions Possible, Complementation, Duality, Simplification of Boolean Expression, K-map, Quine Mc-CluskyMethod.

- **Chapter-3 (Combinational Circuits):**Basics, Design Procedure, Half Adder, Half subtractor, Full Adder, Full Subtractor, Four-bit parallel binary adder / Ripple adder, Look ahead carry adder, Four-bit ripple adder/subtractor, Multiplexer, Demultiplexer, Decoder, Encoder, Priority Encoder

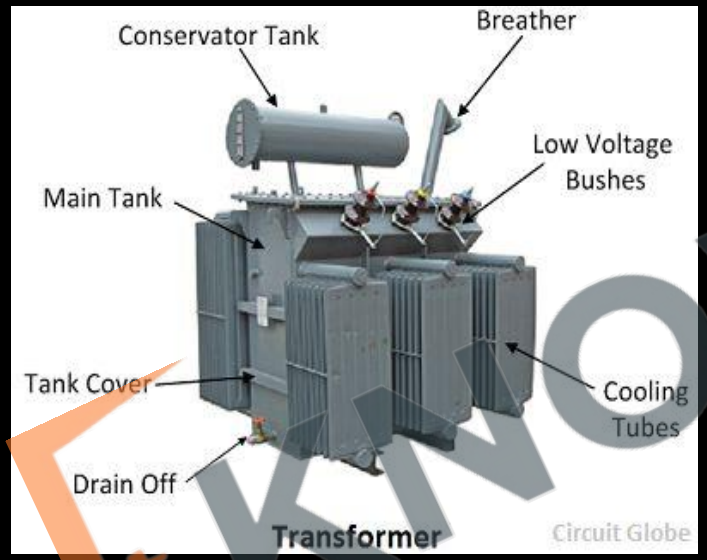
- **Chapter-4 (Sequential Circuits):**Basics,NOR Latch, NAND Latch, SR flip flop, JK flip flop, T(Toggle) flip flop, D flip flop, Flip Flops Conversion, Basics of counters, Finding Counting Sequence Synchronous Counters, Designing Synchronous Counters, Asynchronous/Ripple Counter, Registers, Serial In-Serial Out (SISO), Serial-In Parallel-Out shift Register (SIPO), Parallel-In Serial-Out Shift Register (PISO), Parallel-In Parallel-Out Shift Register (PIPO), Ring Counter, Johnson Counter

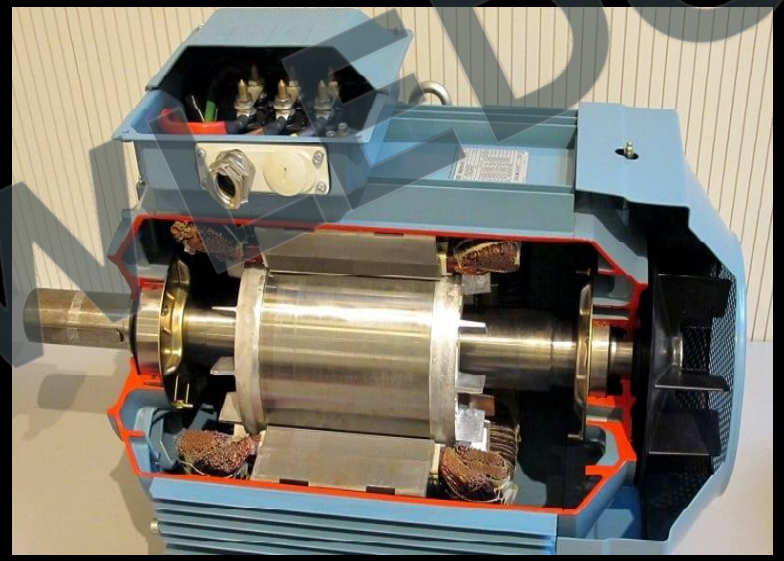
- **Chapter-5 (Number Sysem& Representations):** Basics, Conversion, Signed number Representation, Signed Magnitude, 1's Complement, 2's Complement, Gray Code, Binary-Coded Decimal Code (BCD), Excess-3 Code.

Basics

- **Electrical engineering** is a professional engineering discipline that generally deals with the study and application of electricity, electronics and electromagnetism and heavy voltage devices like transformers, motors etc.

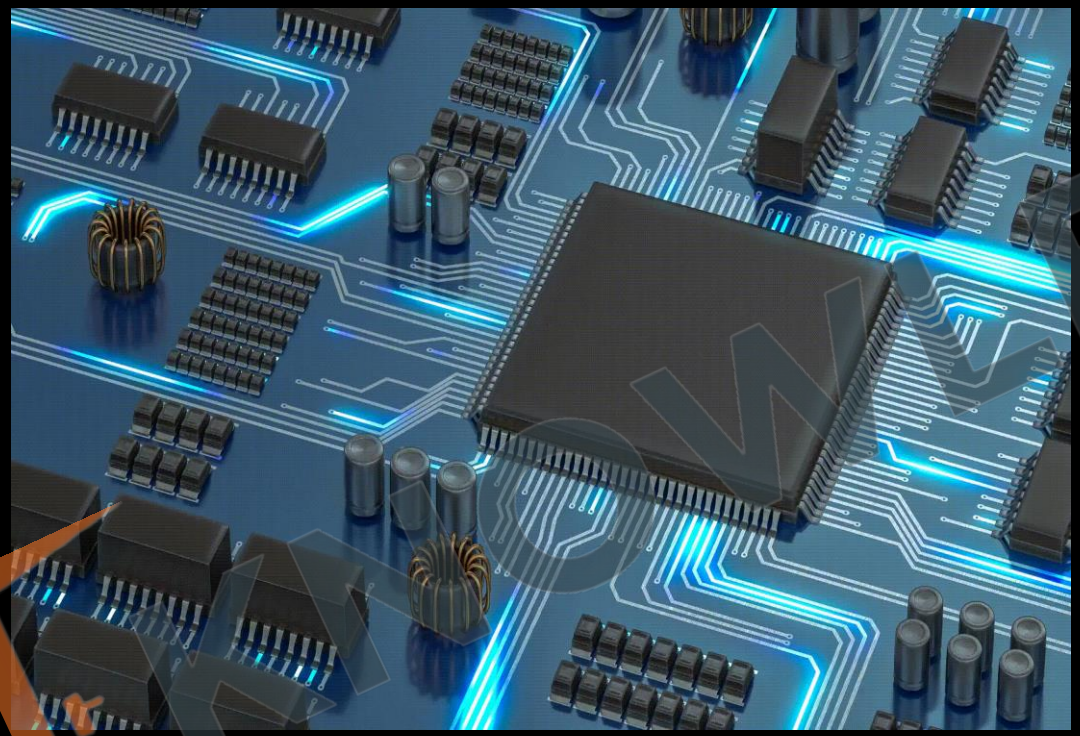
High Voltage



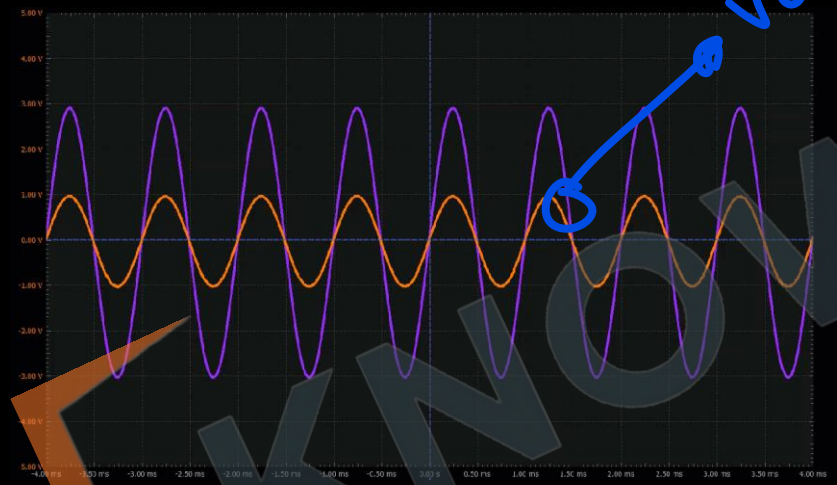


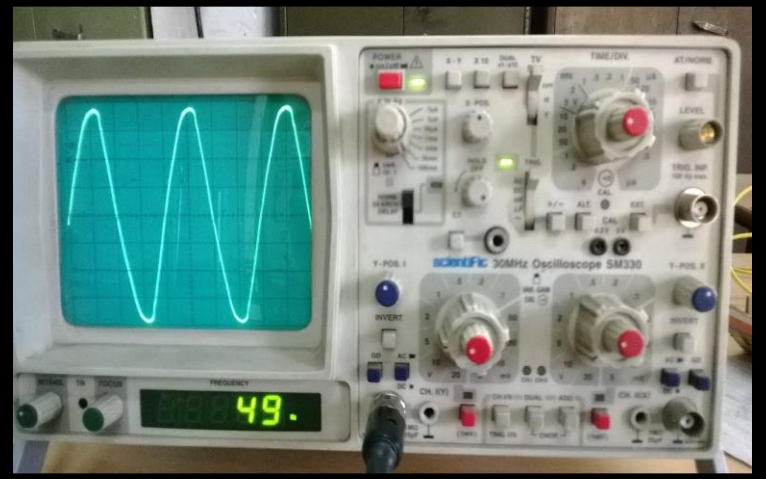
- **Electronic engineering** is an electrical engineering discipline where we work on low voltage devices (such as semiconductor devices, especially transistor, diodes and integrated circuits) to design electronic circuits, VLSI devices and systems.

→ low Voltage

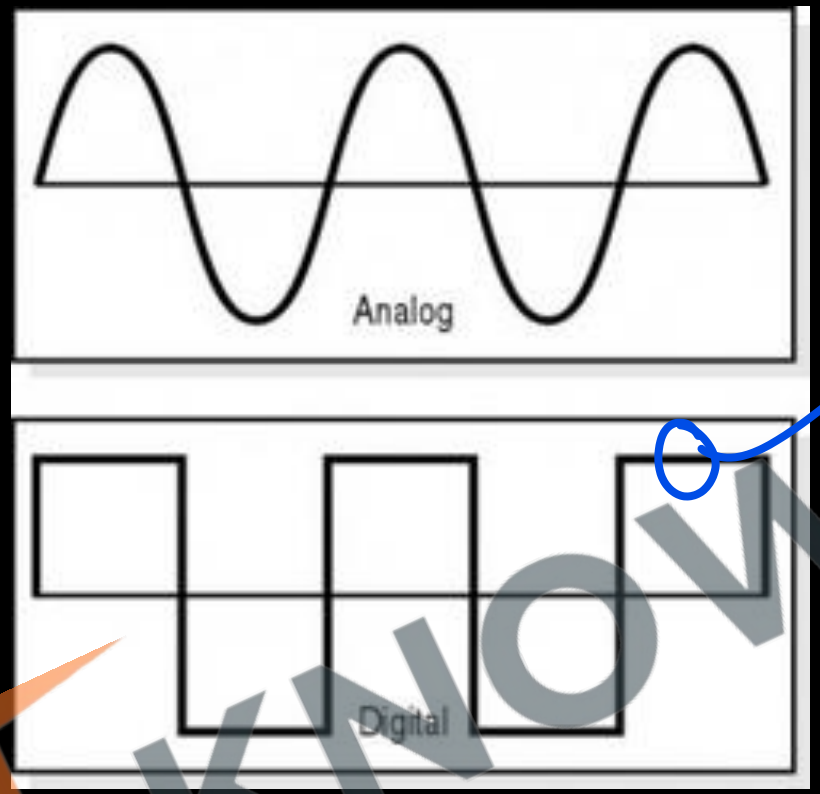


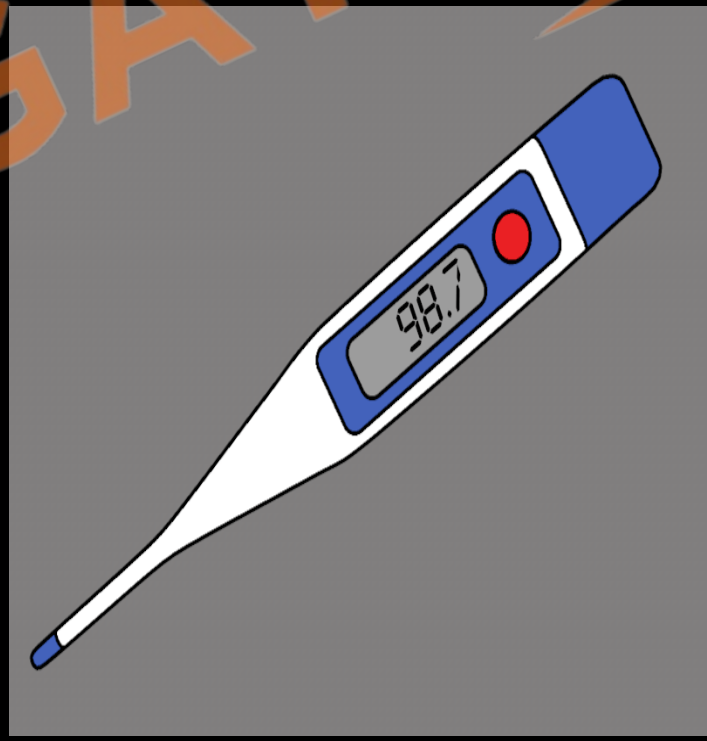
- Electronic systems are generally of two types –
  - **Analog system** in an analog representation, a continuous value is used to denote the information. Analog signal is defined as any physical quantity which varies continuously with respect to time. e.g. amplifier, cro, ecg. Watch, radio.





- **Digital system** – the information is denoted by a finite sequence of discrete value or digits. Digital signal is defined as any physical quantity having discrete values. E.g. digital watch, calculator, bp machine, thermometer etc.

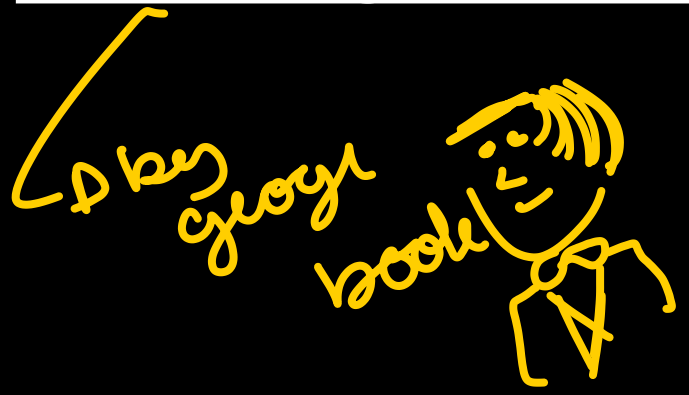




Advantage of Digital system

- **Easy to Design -** Digital systems are simpler to design because you mostly need knowledge of switching circuits. You don't need to be an expert in engineering or math.
- **Long-Term Storage and Easy Processing -** You can store information for a very long time, and it's easier to process data in digital systems.
- **Higher Accuracy and Precision -** Digital devices give more accurate and precise results. They are less affected by things like noise or electric fields.
- **Better Modularity and Fabrication -** Digital systems are modular, meaning they are built in parts that can be easily replaced or upgraded. They are also smaller in size, thanks to things like integrated circuits.
- **Cost-Effective -** Generally, digital systems are less expensive to build and maintain.

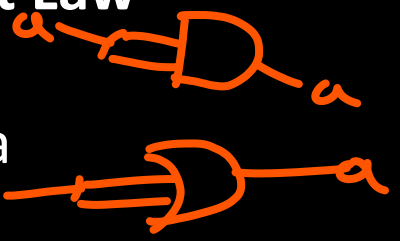
Boolean Algebra Laws



Idempotent Law

$a \cdot a = a$

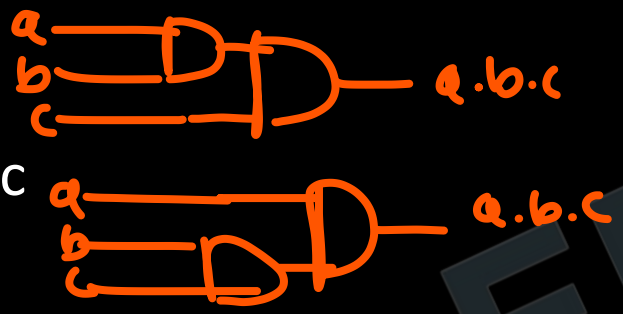
$a + a = a$



Associative law

$a.(b.c) = (a.b).c$

$a + (b + c) = (a + b) + c$



Commutative law

$a.b = b.a$

$a + b = b + a$

Distributive law

$a.(b + c) = a.b + a.c$

$a + (b.c) = (a + b).(a + c)$

De-Morgan law

(a + b)' = a' . b'

(a. b)' = a' + b'

most imp. law

$\overline{a+b} = \overline{a} \cdot \overline{b}$

$\overline{a \cdot b} = \overline{a} + \overline{b}$

content @ Acharya Pingala Sreyobhatta (geekbode)

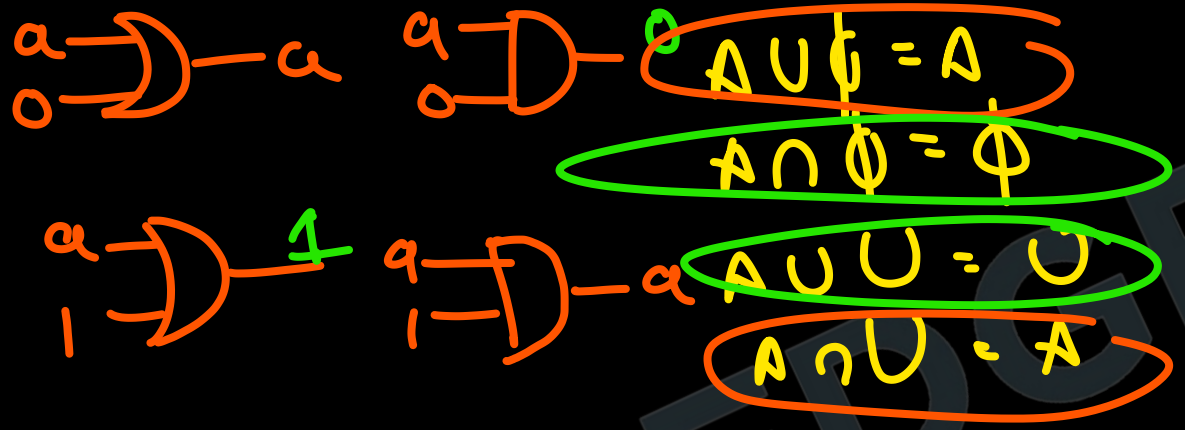
Identity law

a + 0 = a

a. 0 = 0

a + 1 = 1

a. 1 = a



$A \cup \phi = A$

$A \cap \phi = \phi$

$A \cup U = U$

$A \cap U = A$

Complementation law

0' = 1

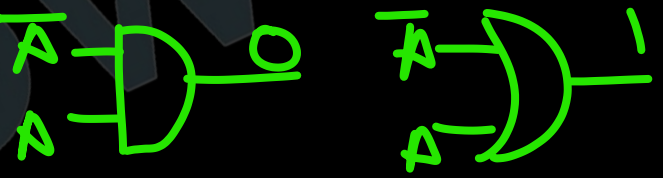
1' = 0

a. a'= 0

a+ a' = 1

$\overline{\phi} = U$

$\overline{U} = \phi$



* bool set th.

$+ \rightarrow \cup$

$\cdot \rightarrow \cap$

$0 \rightarrow \phi$

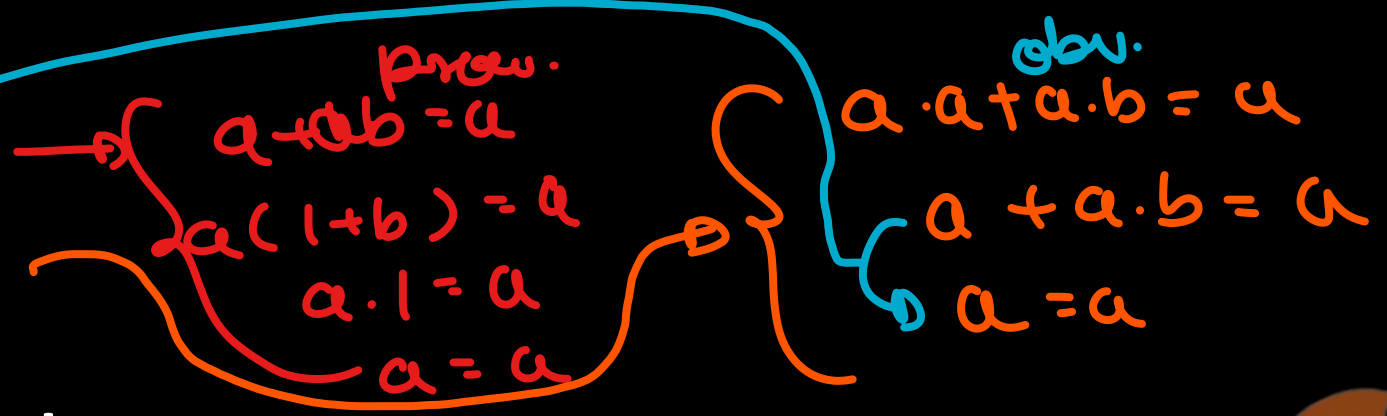
$1 \rightarrow U$

Involution law

(a')' = a

Absorption law

- a + ab = a
- a.(a+b) =a



Compensation theorem

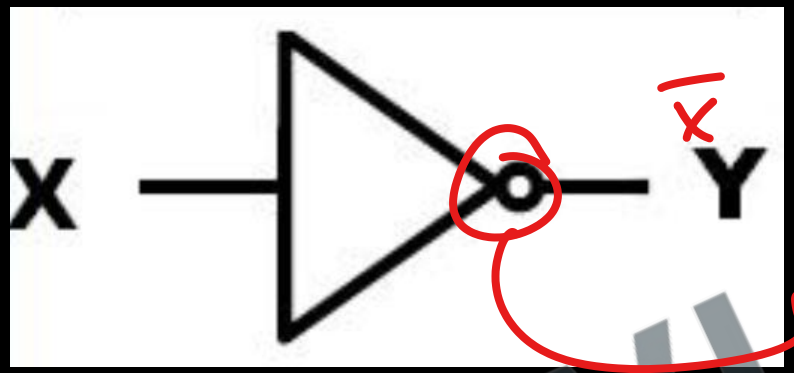
- ab + a'c + bc = ab + a'c
- (a+b) (a' + c) (b + c) = (a+b) (a' + c)

- a + a'b = a + b
- a.(a'+b) = a. b

Not Gate(Inverter)

- It represents not logical operator is also known as inverter, it is a unary operator, which simply complement the input.

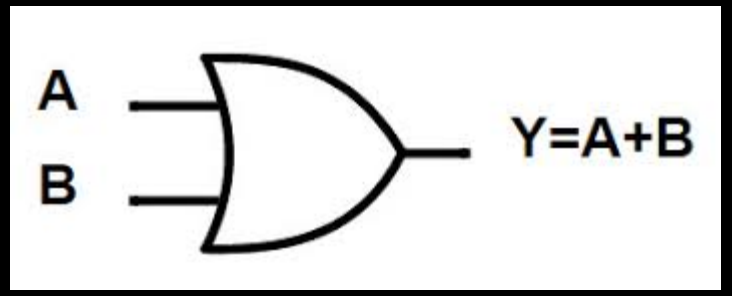
$Y = \overline{X}$

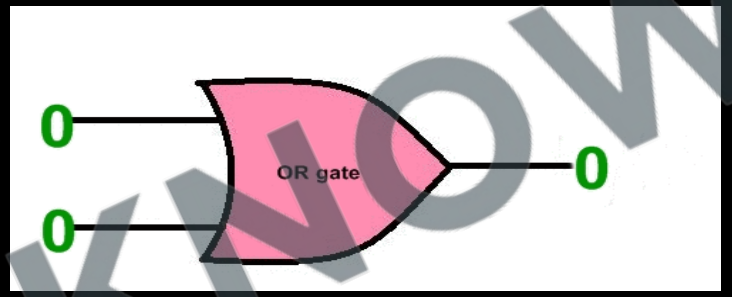


Truth Table	
Input	Output
X	Y = X'
0	1
1	0

OR Gate

- It is a digital logic gate, that implements logical disjunction. The output will be high if at least one of the input lines is high.

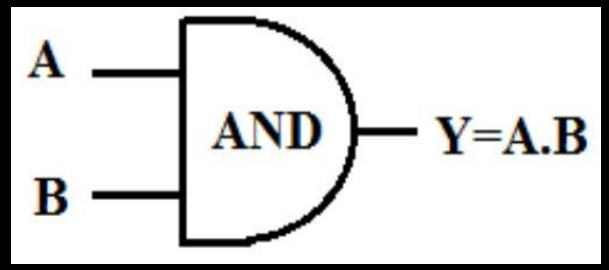




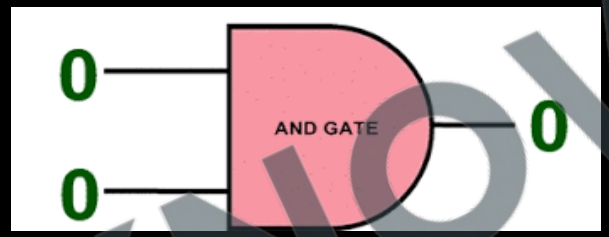
Truth Table		
Input		Output
A	B	Y = A + B
0	0	0
0	1	1
1	0	1
1	1	1

And Gate

- It is a digital logic gate, that implements logical conjunction. Output will be high if and only if all input are high otherwise low.



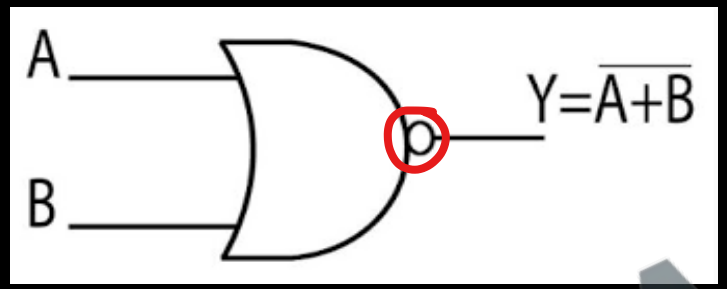
both



Truth Table		
Input		Output
A	B	Y = A . B
0	0	0
0	1	0
1	0	0
1	1	1

Nor gate

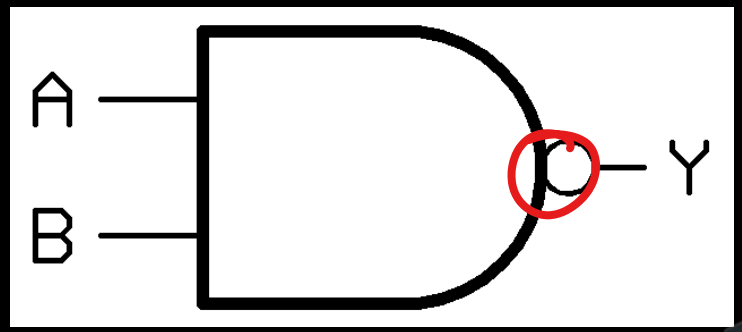
- The output will be high if and only if all inputs are low. Or simply a OR gate followed by an invertor.
- NOR gate is also called universal gate because it can be used to implement any other logic gate.



Truth Table		
Input		Output
A	B	Y = (A + B)'
0	0	1
0	1	0
1	0	0
1	1	0

NAND Gate

- The output will be low if and only if all inputs are high. Or simply an and gate followed by an invertor
- NAND gate is also called universal gate.

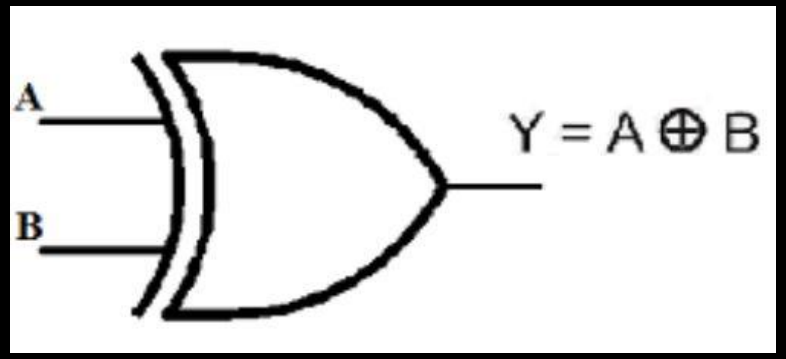


AND

Truth Table		
Input		Output
A	B	Y = (A . B)'
0	0	1
0	1	1
1	0	1
1	1	0

EX-OR

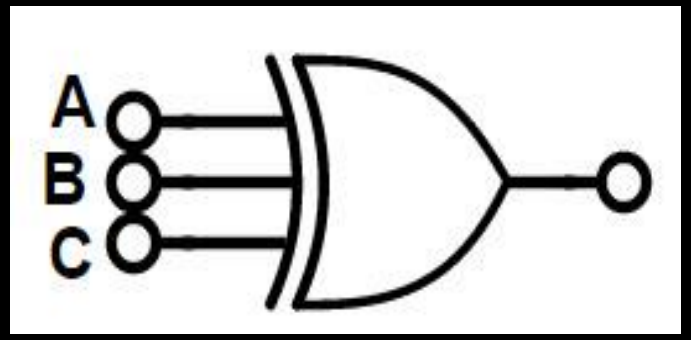
- For two inputs, output will be high if and only if both the input values are different.
- $a \oplus b = a'.b + a.b'$



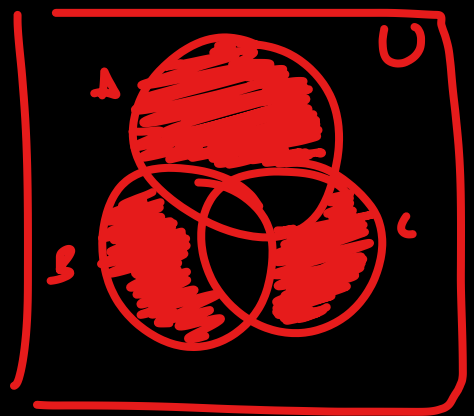
Truth Table		
Input		Output
A	B	Y = A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

EX-OR

- The XOR gate is a digital logic gate that gives High as output when the number of inputs High are odd.

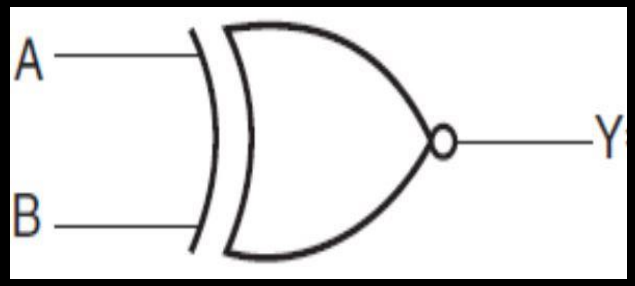


A	B	C	a ⊕ b ⊕ c
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



EX-NOR

- For two input, output will be high if and only if both the input values are same
- $a \odot b = a'.b' + a.b$



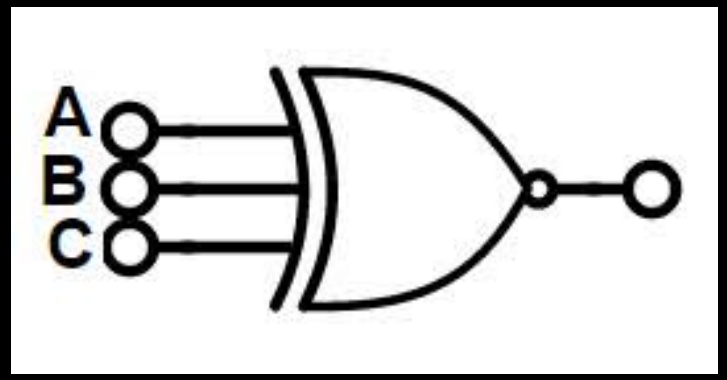
Truth Table		
Input		Output
A	B	Y = A ⊙ B
0	0	1
0	1	0
1	0	0
1	1	1

$a \cdot b = \bar{a} \cdot \bar{b} + a \cdot b$

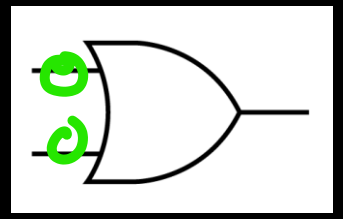
$\overline{EX\,OR} = EX\,NOR$

EX-NOR

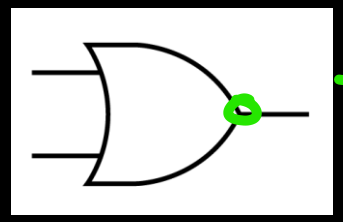
- The EX-NOR gate is a digital logic gate that gives output High when the number inputs low are even.



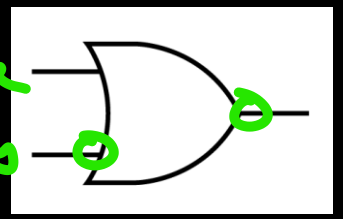
A	B	C	a ⊙ b ⊙ c
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



$\overline{a+b}$

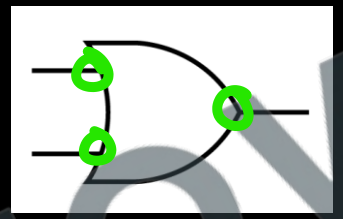


$\overline{a+b} = \bar{a} \cdot \bar{b}$



$\overline{a+b} = \bar{a} \cdot \bar{b}$

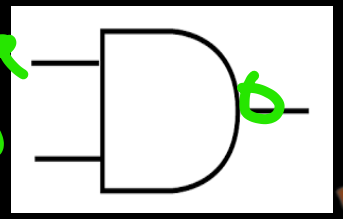
$= a \cdot b$



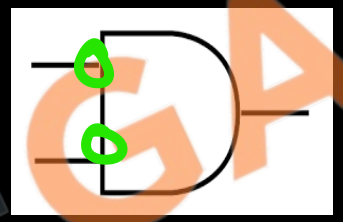
$\overline{a+b} = \bar{a} \cdot \bar{b}$

$= a \cdot b$

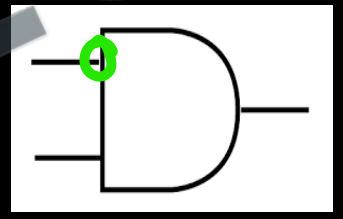
=



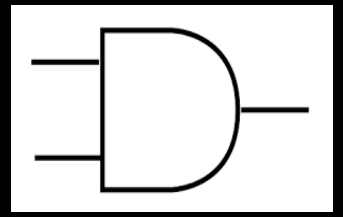
$\overline{a \cdot b} = \overline{a+b}$



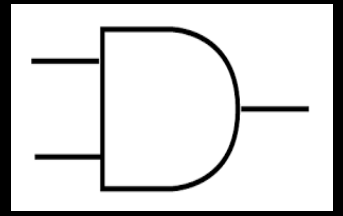
$\bar{a} \cdot \bar{b}$



$\bar{a} \cdot b$

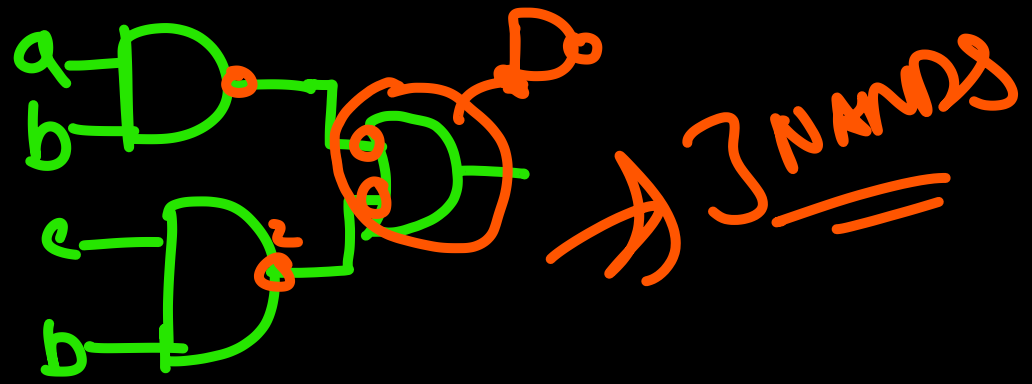


$a \cdot b$



SOP AND-OR(NAND-NAND) Implementation

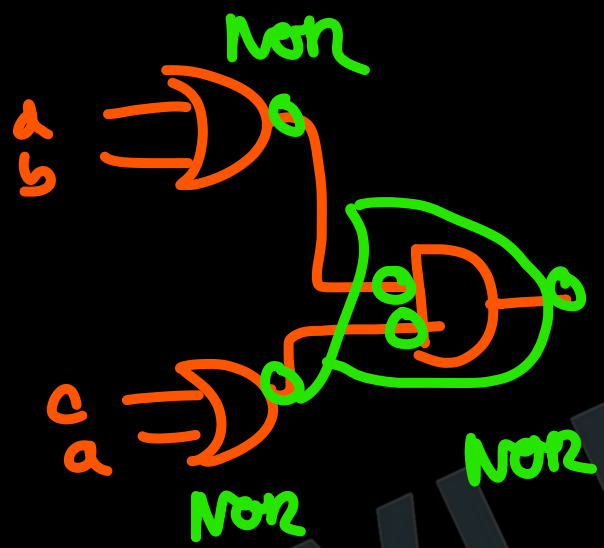
ab + cd



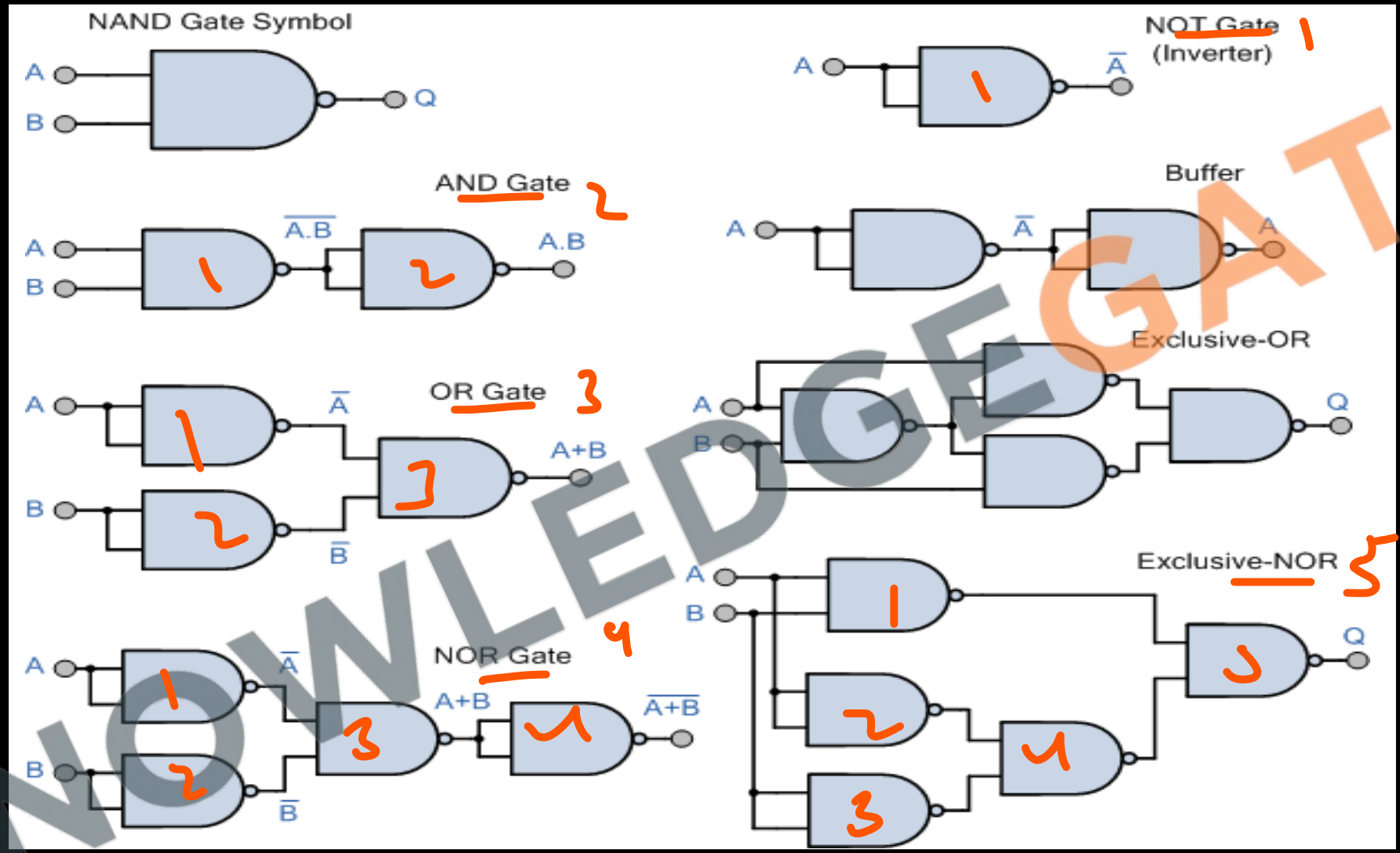
same as OR imp.

POS **OR-AND(NOR-NOR) Implementation**

$(a+b)\cdot(c+d)$

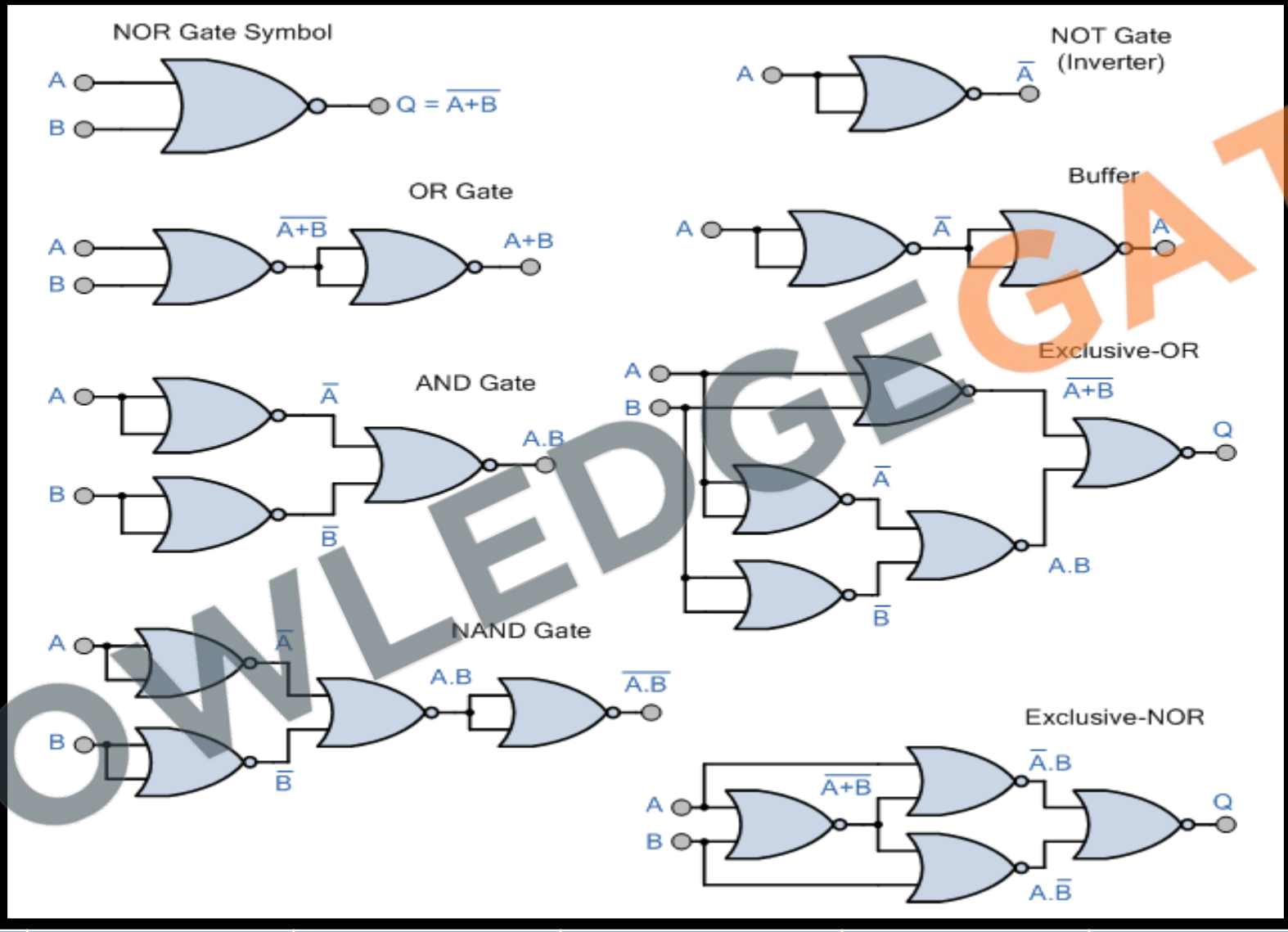


Implementing Every Gate with NAND Gate



	NOT	AND	OR	NAND	NOR	EX-OR	EX-NOR
NAND	1	2	3	1	4	4	5

Implementing Every Gate with NOR Gate

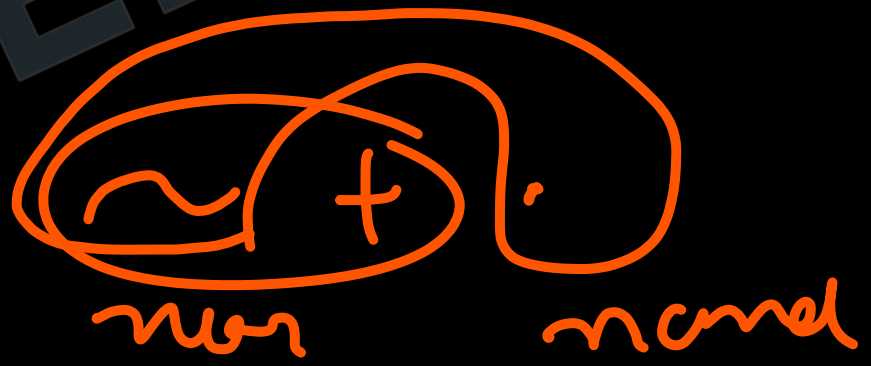


	NOT	AND	OR	NAND	NOR	EX-OR	EX-NOR
NOR	1	3	2	4	1	5	4

Functionally Complete Function

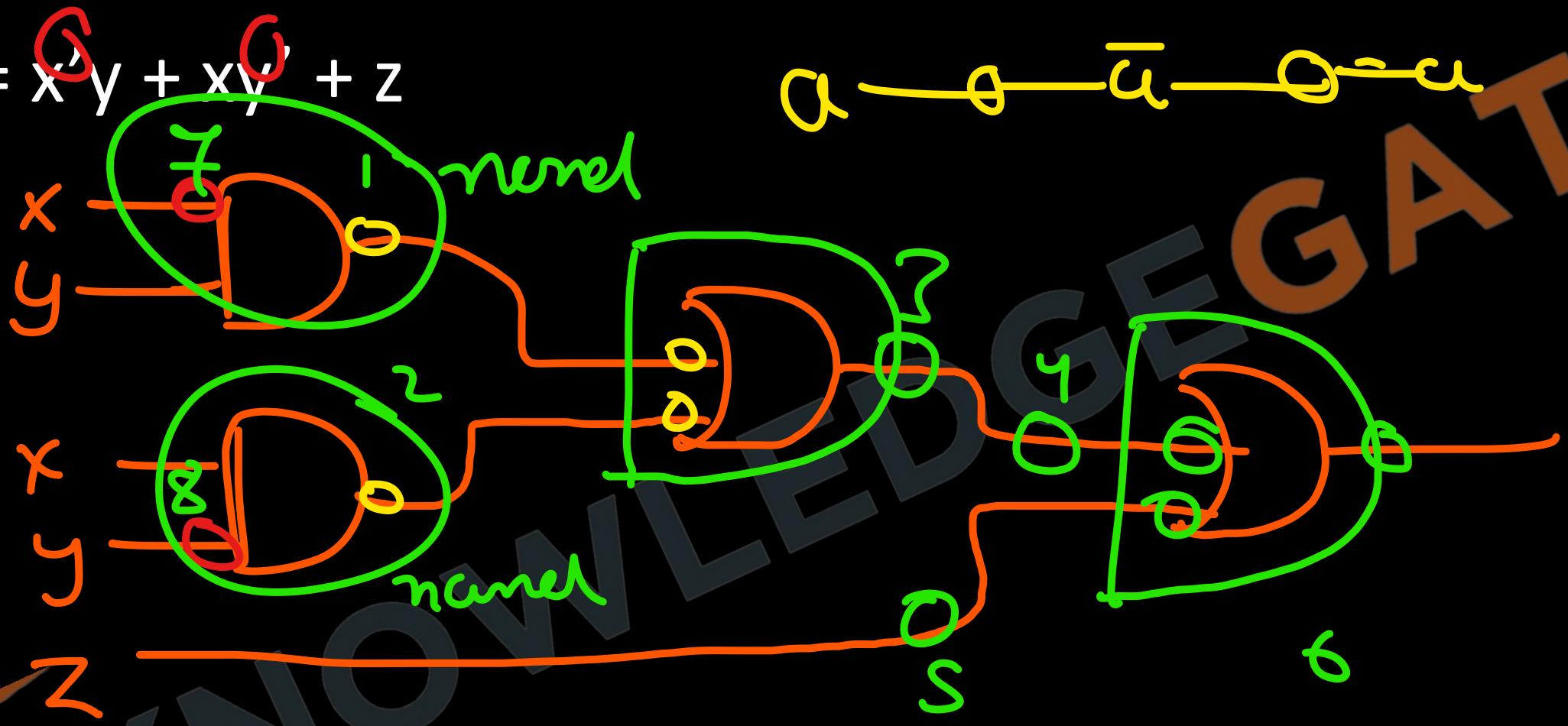
- As we know there are only three fundamental Boolean operator NOT, AND and OR. All the other operators are derived from these operators
- We understand NOT along with OR(NOR) and NOT along with AND(NAND) are used as universal gates.
- Any Boolean operation or function can be implemented with NAND or NOR operation.

* you can make anything with And, Or, not



Implementing any function with universal gates

$F = x'y + xy' + z$



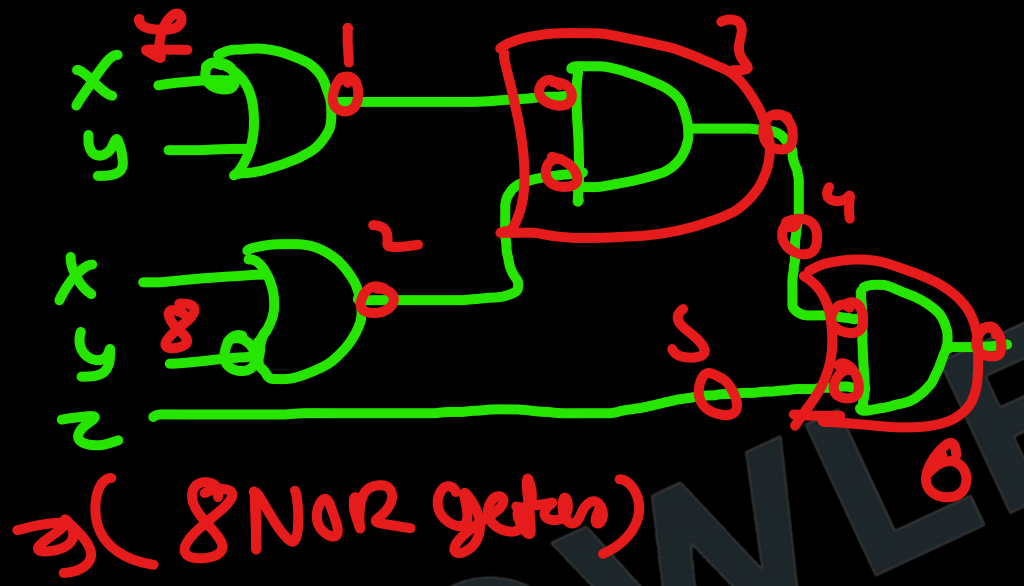
* and output comes or input comes

Implementing any function with universal gates

F = bd + bc + acd’

Implementing any function with universal gates

$F = (x' + y)(x + y')(z)$ POS



Boolean Expressions

- Boolean expressions are the method using which we save the information about the Boolean function, that when we get value 1 and when we get value 0 as output.
- So, we convert the truth table of the function into an expression, reading which we can understand when the output is 1 and when it is zero.

Light	Day	Engine	Warning
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

table → expression

but not Canonical Form

$\bar{a}bc + a\bar{b}\bar{c} + ab(\bar{c}+c)$

this is SOP

this Raw Form is Canonical form

SOP = $\bar{a}b\bar{c} + ab\bar{c} + abc$

- There are two popular approaches for writing these expressions.
  - Sum of Product (SOP) (which remember when we get 1)
  - Product of Sum (POS) (which remember when we get 0)
- Make a note of this as we are studying Boolean function remembering both 0 and 1 is not required, so we can either concentrate on 0 or 1.

Light	Day	Engine	Warning
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

- $W(L, D, E) = \Sigma_m(1, 4, 6, 7)$
- $W(L, D, E) = \Pi_M(0, 2, 3, 5)$

- Power of SOP and POS bother are same, i.e. any Boolean functions can be represented using both SOP and POS.

- Mathematically speaking we should choose (0 or 1), whatever is less in number because then it will be easy to represent

A	B	$f_1 = a+b$	$f_2 = a \cdot b$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

SOP

3 zeros

POS

$\bar{a}b + a\bar{b} + ab$

SOP (Sum of Product)

- A sum of product form expression contains product terms (AND terms) which are sum (OR) together, that's why called sum of product.
- Each product terms (AND terms) consists of one or more literals (variables) appearing either in complements or uncomplemented form. E.g. a'b + b'c' + abc
- A product term which contains all the literals (variables) either in complemented or uncomplemented form is called minterm.
- In a n variable function, there will be 2^n minterms.

Binary Representation	Sequence	Minterm	Designation
000	0	a'b'c'	m_0
001	1	a'b'c	m_1
010	2	a'bc'	m_2
011	3	a'bc	m_3
100	4	ab'c'	m_4
101	5	ab'c	m_5
110	6	abc'	m_6
111	7	abc	m_7

- The result of a product term must always be 1. If a literal is having value 1 then it is ok, but if not, then we complement those which is 0, it to make it 1.
- There is only 1 input sequence of variables for any minterm on which the output is 1, so it represents information.
- Then the sum of all product term(minterm) to from a function, and functions will have value 1, if at least one of the product term(minterm) is 1.

Binary Representation	Sequence	Minterm	Designation
000	0	a'b'c'	m_0
001	1	a'b'c	m_1
010	2	a'bc'	m_2
011	3	a'bc	m_3
100	4	ab'c'	m_4
101	5	ab'c	m_5
110	6	abc'	m_6
111	7	Abc	m_7

Canonical logic forms

- Either in POS or SOP form it is not essential that all product or sum terms contains all the literals.
- **Canonical SOP form**: - In a sum of product form expression, if each AND term (product term) consists all the literals(variables) appearing either in complements or uncomplemented form. E.g. a'bc + ab'c' + abc. Then the form is said to be canonical SOP.

- F (a, b, c) = a + a'b + abc + bc'

* independent so which is work with complement like $a(b+\bar{b})(c+\bar{c})$

$\Rightarrow a(b+\bar{b})(c+\bar{c}) + a'b(c+\bar{c}) + abc + (\bar{a}+a)bc'$

$\Rightarrow a\bar{b}\bar{c} + ab\bar{c} + abc + abc + \bar{a}b\bar{c} + \bar{a}bc + \bar{a}bc + a\bar{b}\bar{c}$

eliminate duplicate

Solve it and get Canonical SOP

POS (Product of Sum)

- A product of sum (POS) form of expression contains OR (sum) terms which are AND (product) together, that's why called product of sum expression.
- Each OR term (sum term) consists of one or more literals(variables) appearing either in complemented or uncomplemented form. (a' + b). (b' + c'). (a + c)
- A OR (sum) term which contains all the literals(variables) either in complemented or uncomplemented form is called maxterm. In a n variable function, there will be 2^n maxterms.

Binary Representation	Sequence	Maxterm	Designation
000	0	a + b + c	M_0
001	1	a + b + c'	M_1
010	2	a + b' + c	M_2
011	3	a + b' + c'	M_3
100	4	a' + b + c	M_4
101	5	a' + b + c'	M_5
110	6	a' + b' + c	M_6
111	7	a' + b' + c'	M_7

- The result of the OR (sum) term must be 0, so If a variable is having value 0 then it is ok, but if not then we complement the variable it to make it 0.
- There is only 1 input sequence for which the output of a Maxterm is 0. Because, it requires all values as zero.
- Then the product of all sum term (maxterm), from a function, and functions will have a value 0, if any of the sum term(maxterm) is 0.

Binary Representation	Sequence	Maxterm	Designation
000	0	a + b + c	M_0
001	1	a + b + c’	M_1
010	2	a + b’ + c	M_2
011	3	a + b’ + c’	M_3
100	4	a’ + b + c	M_4
101	5	a’ + b + c’	M_5
110	6	a’ + b’ + c	M_6
111	7	a’ + b’ + c’	M_7

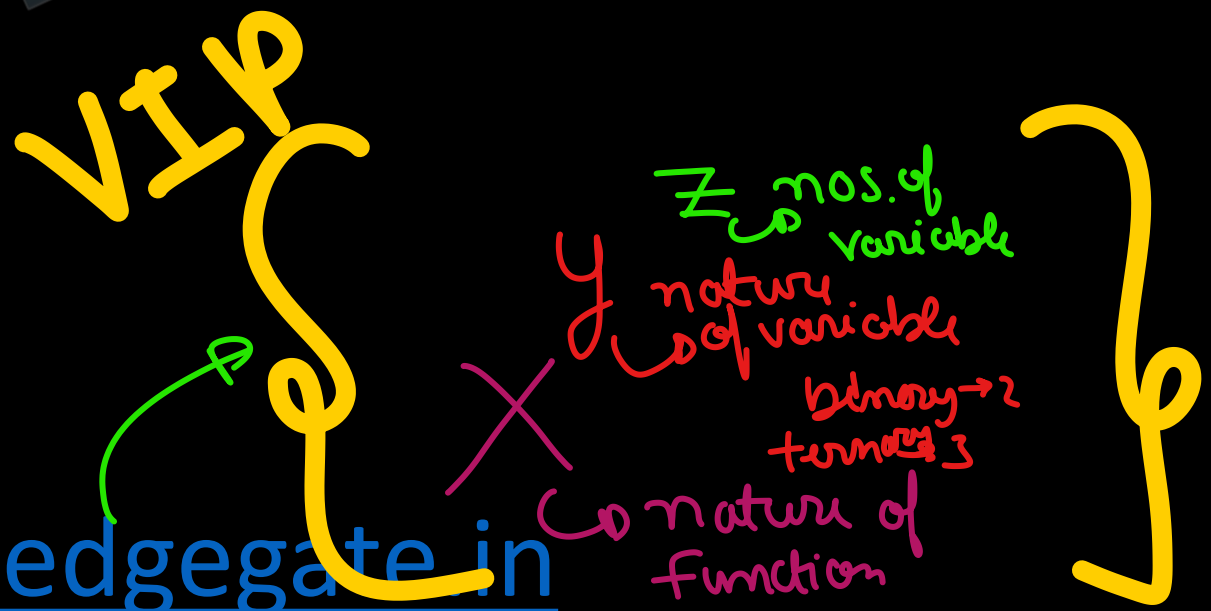
Canonical logic forms

- **Canonical POS form**: - In a product of sum form expression, if each OR term (sum term) consists all the literals(variables) appearing either in complements or uncomplemented form. E.g. (a' + b + c). (a + b' + c'). (a + b + c). Then the form is said to be Canonical POS form.

$(a+b)\ (a+\bar{c})$

$(a+b+\bar{c}\cdot c)\ (a+b\cdot\bar{b}+\bar{c})$

solve it :)



No of functions possible

Q With n-Boolean variables how many different Boolean functions are possible?

With n binary variable we can generate 2^n combinations. And as we know that a Boolean function can also have only 2 possible values either 0 or 1, so total number of different functions possible will be $2^{(2^n)}$.

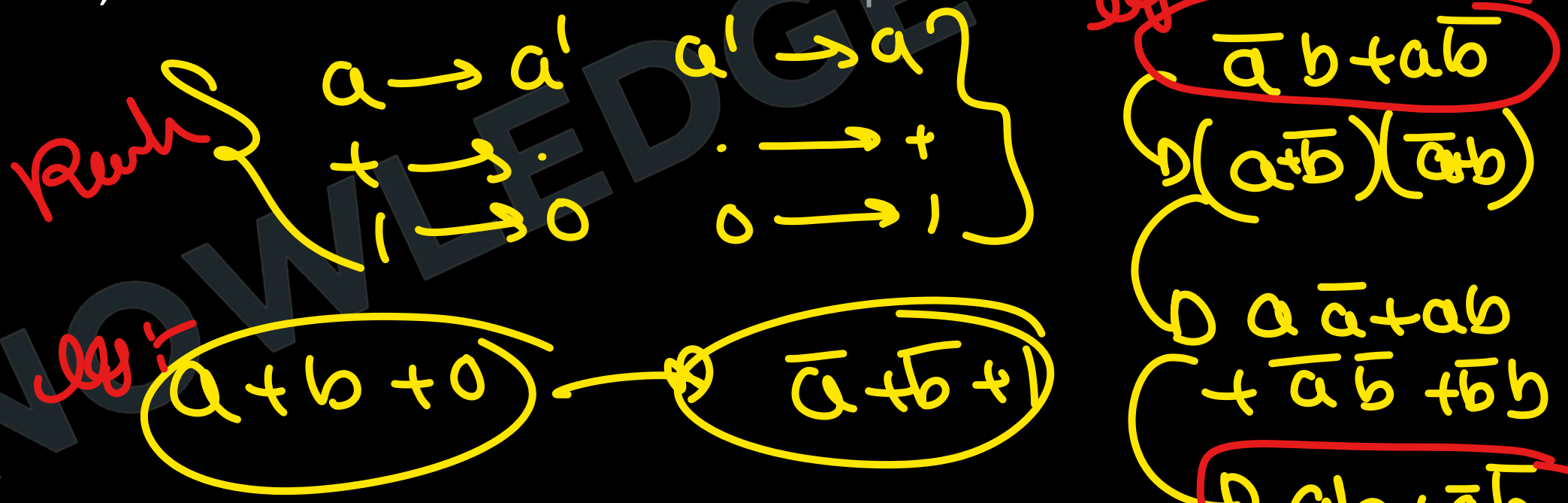
Similarly, we can generalize this idea as $x^{(y^z)}$ are the total number of functions possible, x is the nature of the function, y is the nature of the variable and z is the number of variables.

A	B	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Complementation

- Let us consider a function f(a, b, c, d,……, 0, 1, +, ·), then the complement of the function is defined as f'(a', b', c',……, z', 1, 0, ., +).

- i.e. When all the variables are replaced by their compliments, 0→1,1→0, or →and, and → or, then we will be called them compliment of a function.



Duality

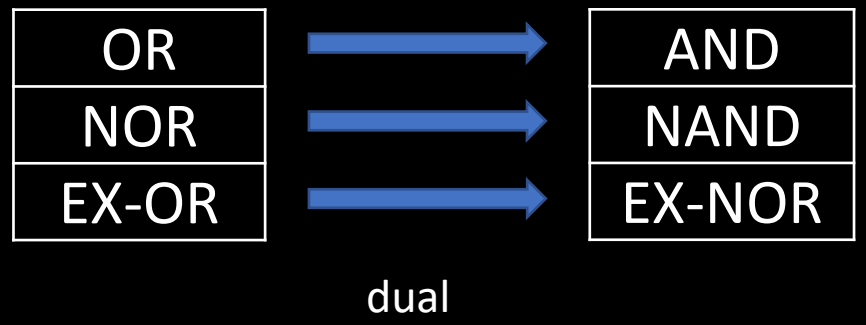
- Let us consider a function f(a, b, c, d,……,z,0,1,+,·), then the dual of the function is defined as f^d(a, b, c,……,z,1,0,·.,+).

- When the nature of variable remains same but 0 → 1, 1 → 0, or → and, and → or, then they are called dual functions.

a	b		
L 0 H	L 0 H	L 0 H	L 0 H
L 0 H	H 1 L	H 1 L	L 0 H
H 1 L	L 0 H	H 1 L	L 0 H
H 1 L	H 1 L	H 1 L	H 1 L

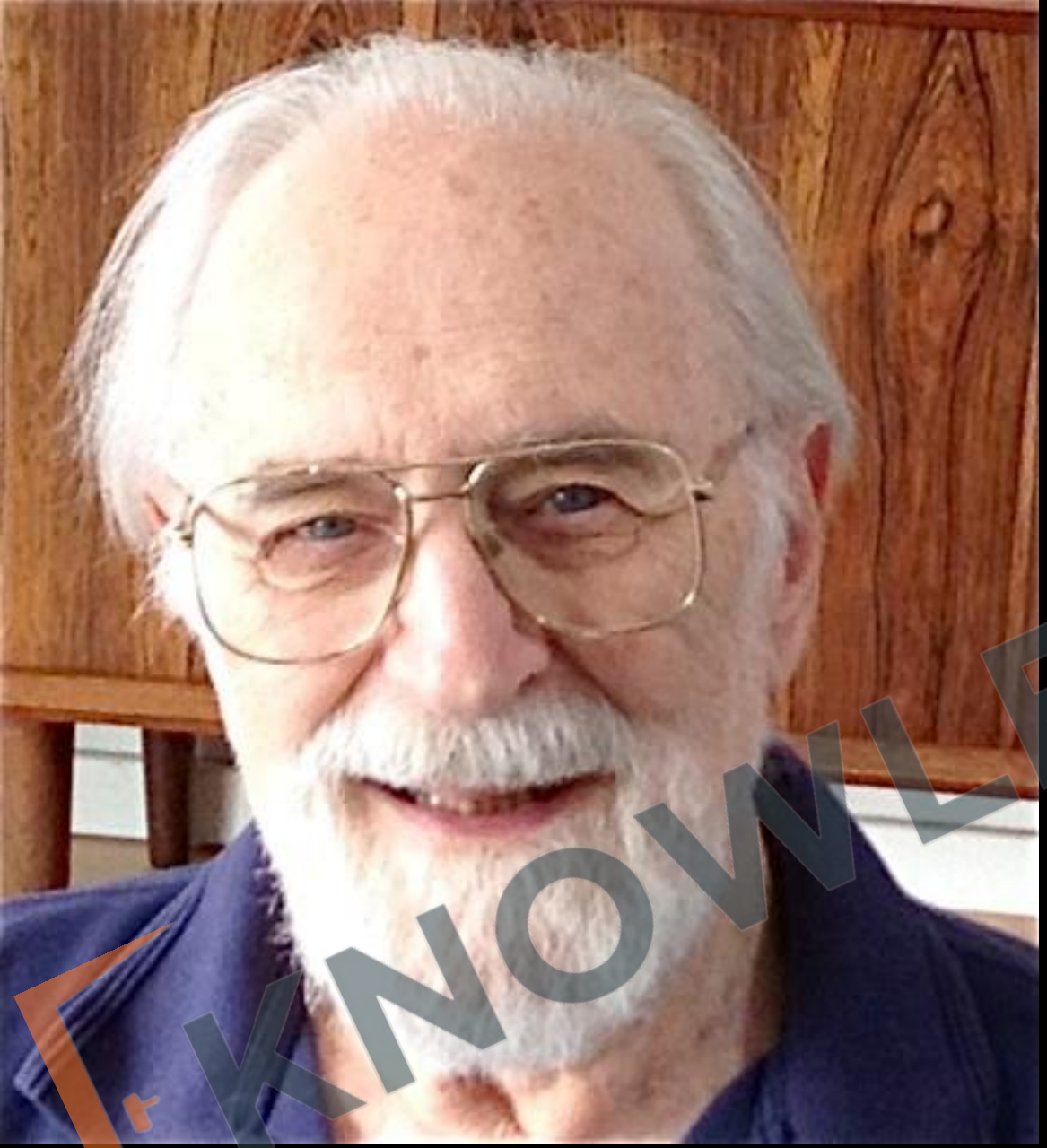
- When we take dual of a function, then the functionality of a function remains the same but a positive logic system is transformed to negative logic system.

- If a function works corrects in positive logic system, then it must also work correct in negative logic system as it do not depends on magnitude.



Simplification of Boolean expression

- After deriving a Boolean expression from the truth table next important step is to minimise it, so that the cost of implementation into hardware and delay because of additional gates can be reduced. There are following methods for simplifying a Boolean expression
- Using Karnaugh-map
- Using Quine Mc-Clusky Method(Tabular Method)
- Using Algebraic method (Boolean Laws)



- Maurice Karnaugh (4 October 1924 – 8 Nov 2022) is an American physicist, mathematician and inventor known for the Karnaugh map used in Boolean algebra.

- Karnaugh map is one of the most extensively used tool, it is a graphical representation, represents truth table by pictorial form, provides a systematic method for simplifying or minimizing a Boolean expression. For a n-variable k-map, there will be 2^n cells addressed by a gray code. Each cell corresponds to one minterm or maxterm.

	a	b	c	d	F
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	
10	1	0	1	0	
11	1	0	1	1	
12	1	1	0	0	
13	1	1	0	1	
14	1	1	1	0	
15	1	1	1	1	

	ab				
cd					

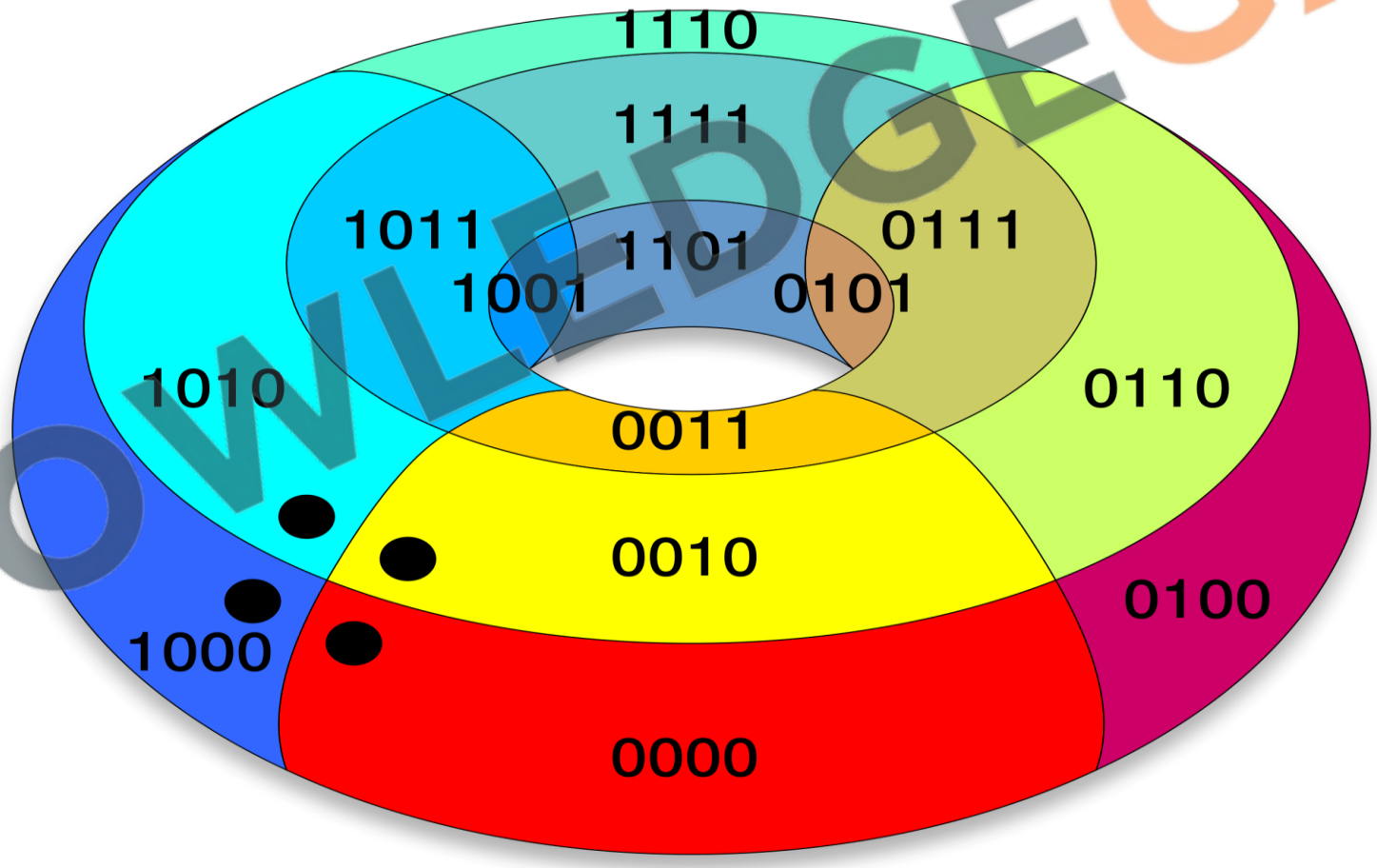
	ab	a’b’	a’b	ab	ab’
cd		00	01	11	10
c’d’	00	0	4	12	8
c’d	01	1	5	13	9
cd	11	3	7	15	11
cd’	10	2	6	14	10

	cd	c’d’	c’d	cd	cd’
ab		00	01	11	10
a’b’	00	0	1	3	2
a’b	01	4	5	7	6
ab	11	12	13	15	14
ab’	10	8	9	11	10

	ad	a’d’	a’d	ad	ad’
bc		00	01	11	10
b’c’	00	0	1	9	8
b’c	01	2	3	11	10
bc	11	6	7	15	14
bc’	10	4	5	13	12

	db	d’b’	d’b	db	db’
ca		00	01	11	10
c’a’	00	0	4	5	1
c’a	01	8	12	13	9
ca	11	10	14	15	11
ca’	10	2	6	7	3

0000	0100	1100	1000
0001	0101	1101	1001
0011	0111	1111	1011
0010	0110	1110	1010



	ab	a’b’	a’b	ab	ab’
c		00	01	11	10
c’	0	0	2	6	4
c	1	1	3	7	5

	bc	b’c’	b’c	bc	bc’
a		00	01	11	10
a’	0	0	1	3	2
a	1	4	5	7	6

	a	a’	a
bc		0	1
b’c’	00	0	4
b’c	01	1	5
bc	11	3	7
bc’	10	2	6

	a	a’	a
b		0	1
b’	0	0	2
b	1	1	3

	b	b’	b
a		0	1
a’	0	0	1
a	1	2	3

5-Variable K-Map

a'

	bc	b'c'	b'c	bc	bc'
de		00	01	11	10
d'e'	00	0	4	12	8
d'e	01	1	5	13	9
de	11	3	7	15	11
de'	10	2	6	14	10

a

	bc	b'c'	b'c	bc	bc'
de		00	01	11	10
d'e'	00	16	20	28	24
d'e	01	17	21	29	25
de	11	19	23	31	27
de'	10	18	22	30	26

6-Variable K-Map

a'b'

	cd	c'd'	c'd	cd	cd'
ef		00	01	11	10
e'f'	00	0	4	12	8
e'f	01	1	5	13	9
ef	11	3	7	15	11
ef'	10	2	6	14	10

a'b

	cd	c'd'	c'd	cd	cd'
ef		00	01	11	10
e'f'	00	16	20	28	24
e'f	01	17	21	29	25
ef	11	19	23	31	27
ef'	10	18	22	30	26

ab'

	cd	c'd'	c'd	cd	cd'
ef		00	01	11	10
e'f'	00	32	36	44	40
e'f	01	33	37	45	41
ef	11	35	39	47	43
ef'	10	34	38	46	42

ab

	cd	c'd'	c'd	cd	cd'
ef		00	01	11	10
e'f'	00	48	52	60	56
e'f	01	49	53	61	57
ef	11	51	55	63	59
ef'	10	50	54	62	58

7-Variable K-Map

a'b'c'

	cd	c'd'	c'd	cd	cd'
ef		00	01	11	10
e'f'	00				
e'f	01		0-15		
ef	11				
ef'	10				

a'b'c

	cd	c'd'	c'd	cd	cd'
ef		00	01	11	10
e'f'	00				
e'f	01		16-31		
ef	11				
ef'	10				

a'bc'

	cd	c'd'	c'd	cd	cd'
ef		00	01	11	10
e'f'	00				
e'f	01		32-47		
ef	11				
ef'	10				

a'bc

	cd	c'd'	c'd	cd	cd'
ef		00	01	11	10
e'f'	00				
e'f	01		48-63		
ef	11				
ef'	10				

ab'c'

	cd	c'd'	c'd	cd	cd'
ef		00	01	11	10
e'f'	00				
e'f	01		64-79		
ef	11				
ef'	10				

ab'c

	cd	c'd'	c'd	cd	cd'
ef		00	01	11	10
e'f'	00				
e'f	01		80-95		
ef	11				
ef'	10				

abc'

	cd	c'd'	c'd	cd	cd'
ef		00	01	11	10
e'f'	00				
e'f	01		96-111		
ef	11				
ef'	10				

abc

	cd	c'd'	c'd	cd	cd'
ef		00	01	11	10
e'f'	00				
e'f	01		112-127		
ef	11				
ef'	10				

a'b’c’d’

	cd	c’d’	c’d	cd	cd’
ef		00	01	11	10
e’f’	00				
e’f	01		0-15		
ef	11				
ef’	10				

a’b’c’d

	cd	c’d’	c’d	cd	cd’
ef		00	01	11	10
e’f’	00				
e’f	01		16-31		
ef	11				
ef’	10				

a’b’cd’

	cd	c’d’	c’d	cd	cd’
ef		00	01	11	10
e’f’	00				
e’f	01		32-47		
ef	11				
ef’	10				

a’b’cd

	cd	c’d’	c’d	cd	cd’
ef		00	01	11	10
e’f’	00				
e’f	01		48-63		
ef	11				
ef’	10				

a’bc’d’

	cd	c’d’	c’d	cd	cd’
ef		00	01	11	10
e’f’	00				
e’f	01		64-79		
ef	11				
ef’	10				

a’bc’d

	cd	c’d’	c’d	cd	cd’
ef		00	01	11	10
e’f’	00				
e’f	01		80-95		
ef	11				
ef’	10				

a’bcd’

	cd	c’d’	c’d	cd	cd’
ef		00	01	11	10
e’f’	00				
e’f	01		96-111		
ef	11				
ef’	10				

a’bcd

	cd	c’d’	c’d	cd	cd’
ef		00	01	11	10
e’f’	00				
e’f	01		112-127		
ef	11				
ef’	10				

ab’c’d’

	cd	c’d’	c’d	cd	cd’
ef		00	01	11	10
e’f’	00				
e’f	01		128-143		
ef	11				
ef’	10				

ab’c’d

	cd	c’d’	c’d	cd	cd’
ef		00	01	11	10
e’f’	00				
e’f	01		144-159		
ef	11				
ef’	10				

ab’cd’

	cd	c’d’	c’d	cd	cd’
ef		00	01	11	10
e’f’	00				
e’f	01		160-175		
ef	11				
ef’	10				

ab’cd

	cd	c’d’	c’d	cd	cd’
ef		00	01	11	10
e’f’	00				
e’f	01		176-191		
ef	11				
ef’	10				

abc’d’

	cd	c’d’	c’d	cd	cd’
ef		00	01	11	10
e’f’	00				
e’f	01		192-207		
ef	11				
ef’	10				

abc’d

	cd	c’d’	c’d	cd	cd’
ef		00	01	11	10
e’f’	00				
e’f	01		208-223		
ef	11				
ef’	10				

abcd’

	cd	c’d’	c’d	cd	cd’
ef		00	01	11	10
e’f’	00				
e’f	01		224-239		
ef	11				
ef’	10				

abcd

	cd	c’d’	c’d	cd	cd’
ef		00	01	11	10
e’f’	00				
e’f	01		240-255		
ef	11				
ef’	10				

K-Map for POS

	a+b	a+b	a+b'	a'+b'	a'+b
c+d		0+0	0+1	1+1	1+0
c+d	0+0	0	4	12	8
c+d'	0+1	1	5	13	9
c'+d'	1+1	3	7	15	11
c'+d	1+0	2	6	14	10

- **Minimal function**: - A function is said to be minimal if it is representing the function expression, if it is using minimal no of literals to represent the function.

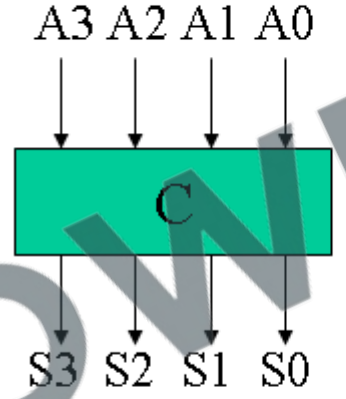
- Rules of grouping: -

1. Every minterm must be covered
2. Group must have contiguous Cells(circular)
3. Group must be in horizontal or vertical fashion(circular)
4. Number of cells in a group must be in power of 2
5. Will Try to make largest group possible, so that number of literals in the expression can be reduced
6. Can also take don't care if it helps in creating the larger groups, other wise don't care.
7. Will consider new implicant if it is covering some new minterm.

	ab	a'b'	a'b	ab	ab'
cd		00	01	11	10
c'd'	00	0	4	12	8
c'd	01	1	5	13	9
cd	11	3	7	15	11
cd'	10	2	6	14	10

Don’t care condition

- Don’t care cases are those cases which can never occur logically in that function.
- It is not essential to cover don’t care conditions, but if don’t care are helping to generate bigger prime implicants, then we can use them.



A3	A2	A1	A0	S3	S2	S1	S0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

	ab	a’b’	a’b	ab	ab’
cd		00	01	11	10
c’d’	00			1	
c’d	01	D	D	1	D
cd	11		1	1	
cd’	10		1	1	

First try to cover those minterms which do not have an option.

	ab	a'b'	a'b	ab	ab'
cd		00	01	11	10
c'd'	00		1		
c'd	01		1	1	1
cd	11	1	1	1	
cd'	10			1	

For some functions more than one minimal Boolean expression are possible

	ab	a'b'	a'b	ab	ab'
cd		00	01	11	10
c'd'	00		D		D
c'd	01	1	1	D	
cd	11		D	1	1
cd'	10		1	D	

Q $f(a, b, c) = \sum_m \{1,2,3,4,5\}$

	ab	a'b'	a'b	ab	ab'
c		00	01	11	10
c'	0	0	2	6	4
c	1	1	3	7	5

	ab	a’b’	a’b	ab	ab’
cd		00	01	11	10
c’d’	00	1			1
c’d	01	1	1		
cd	11		1	1	
cd’	10			1	1

	ab	a’b’	a’b	ab	ab’
cd		00	01	11	10
c’d’	00	1	D	D	1
c’d	01	D			
cd	11				
cd’	10	1			D

	ab	a’b’	a’b	ab	ab’
cd		00	01	11	10
c’d’	00		1	1	
c’d	01	D			1
cd	11	D			1
cd’	10		1	1	D

	ab	a’b’	a’b	ab	ab’
cd		00	01	11	10
c’d’	00	D	1		1
c’d	01		1	D	
cd	11	1	D	D	
cd’	10	D			D

K-Map for POS

	a+b	a+b	a+b’	a’+b’	a’+b
c+d		0+0	0+1	1+1	1+0
c+d	0+0			D	1
c+d’	0+1		1	D	1
c’+d’	1+1		1	D	D
c’+d	1+0		1	D	D

Using Quine Mc-Clusky Method(Tabular Method)

Y(a, b, c, d) = $\sum_m$ (0, 1, 3, 7, 8, 9, 11, 15)

Group	Minterm	Binary Representation
0	0	0 0 0 0
1	1 8	0 0 0 1 1 0 0 0
2	3 9	0 0 1 1 1 0 0 1
3	7 11	0 1 1 1 1 0 1 1
4	15	1 1 1 1

Group	Minterm	Binary Representation
0	0,1 0,8	0 0 0 _ _ 0 0 0
1	1,3 1,9 8,9	0 0 _ 1 _ 0 0 1 1 0 0 _
2	3,7 3,11 9,11	0 _ 1 1 _ 0 1 1 1 0 _1
3	7,15 11,15	_ 1 1 1 1 _ 1 1

Group	Minterm	Binary Representation
0	0,1 0,8	0 0 0 _ _ 0 0 0
1	1,3 1,9 8,9	0 0 _ 1 _ 0 0 1 1 0 0 _
2	3,7 3,11 9,11	0 _ 1 1 _ 0 1 1 1 0 _ 1
3	7,15 11,15	_ 1 1 1 1 _ 1 1

Group	Minterm	Binary Representation
0	0,1,8,9 0,8,1,9	_ 0 0 _ _ 0 0 _
1	1,3,9,11 1,9,3,11	_ 0 _ 1 _ 0 _ 1
2	3,7,11,15 3,11,7,15	_ _ 1 1 _ _ 1 1

Group	Minterm	Binary Representation	
0	0,1,8,9 0,8,1,9	_ 0 0 _ _ 0 0 _	b'c'
1	1,3,9,11 1,9,3,11	_ 0 _ 1 _ 0 _ 1	b'd
2	3,7,11,15 3,11,7,15	_ _ 1 1 _ _ 1 1	cd

Y(a, b, c, d) = b'c' + cd

PI	Minterm	0	1	3	7	8	9	11	15
b'c'	0,1,8,9	X	X			X	X		
b'd	1,3,9,11		X	X			X	X	
cd	3,7,11,15			X	X			X	X

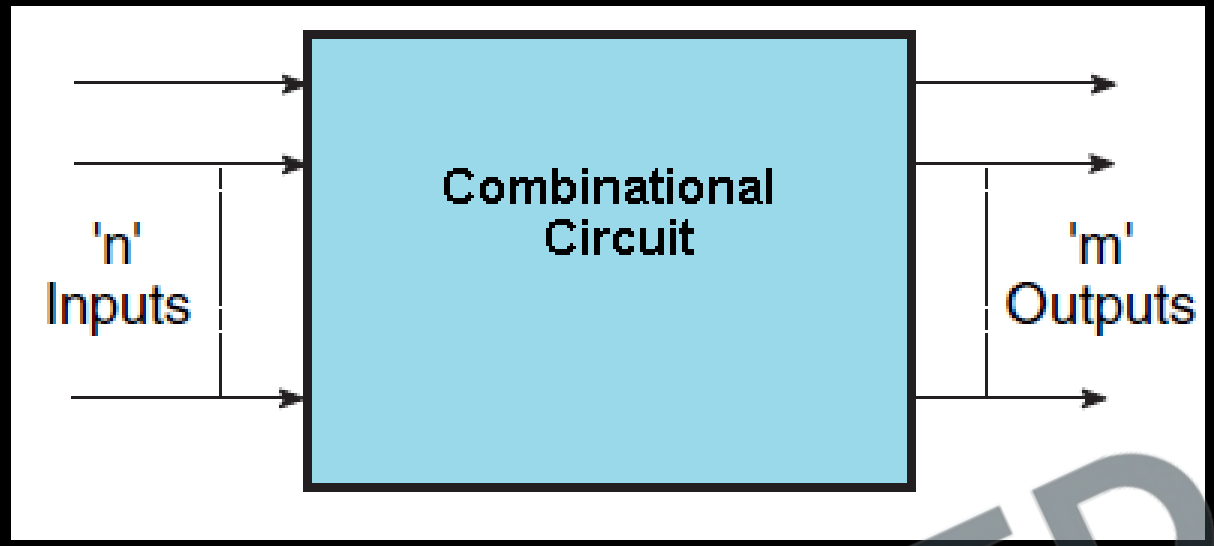
PI	Minterm	0	1	3	7	8	9	11	15
b'c'	0,1,8,9	X	X			X	X		
b'd	1,3,9,11		X	X			X	X	
cd	3,7,11,15			X	X			X	X

Y(a, b, c, d) = $\sum_m$ (0, 1, 3, 7, 8, 9, 11, 15)

Y(a, b, c, d) = b'c' + cd

	ab	a'b'	a'b	ab	ab'
cd		00	01	11	10
c'd'	00	1			1
c'd	01	1			1
cd	11	1	1	1	1
cd'	10				

Combinational Circuits



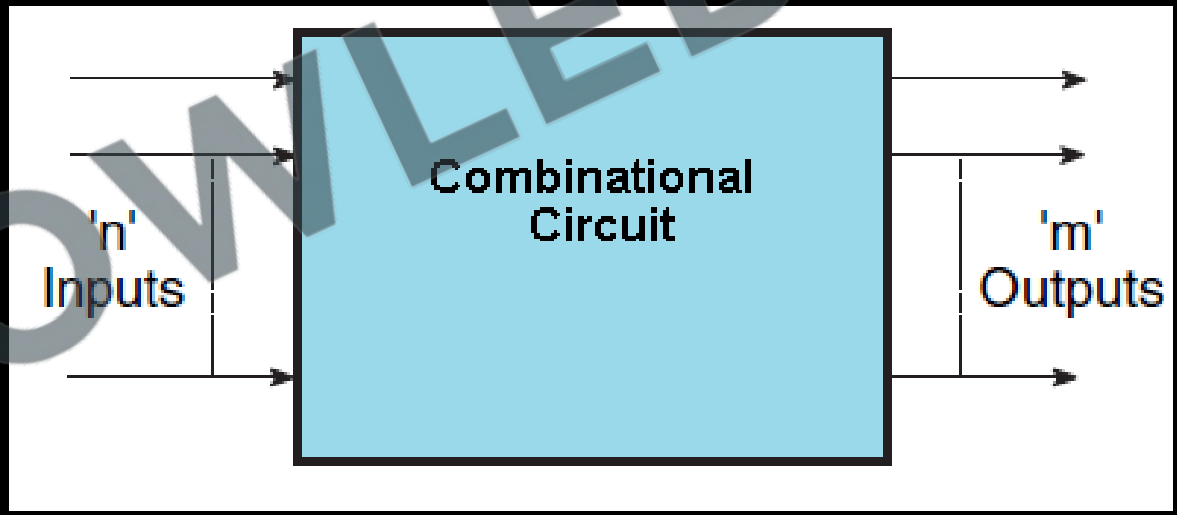
O/p = f(i/p)

Combinational Circuits

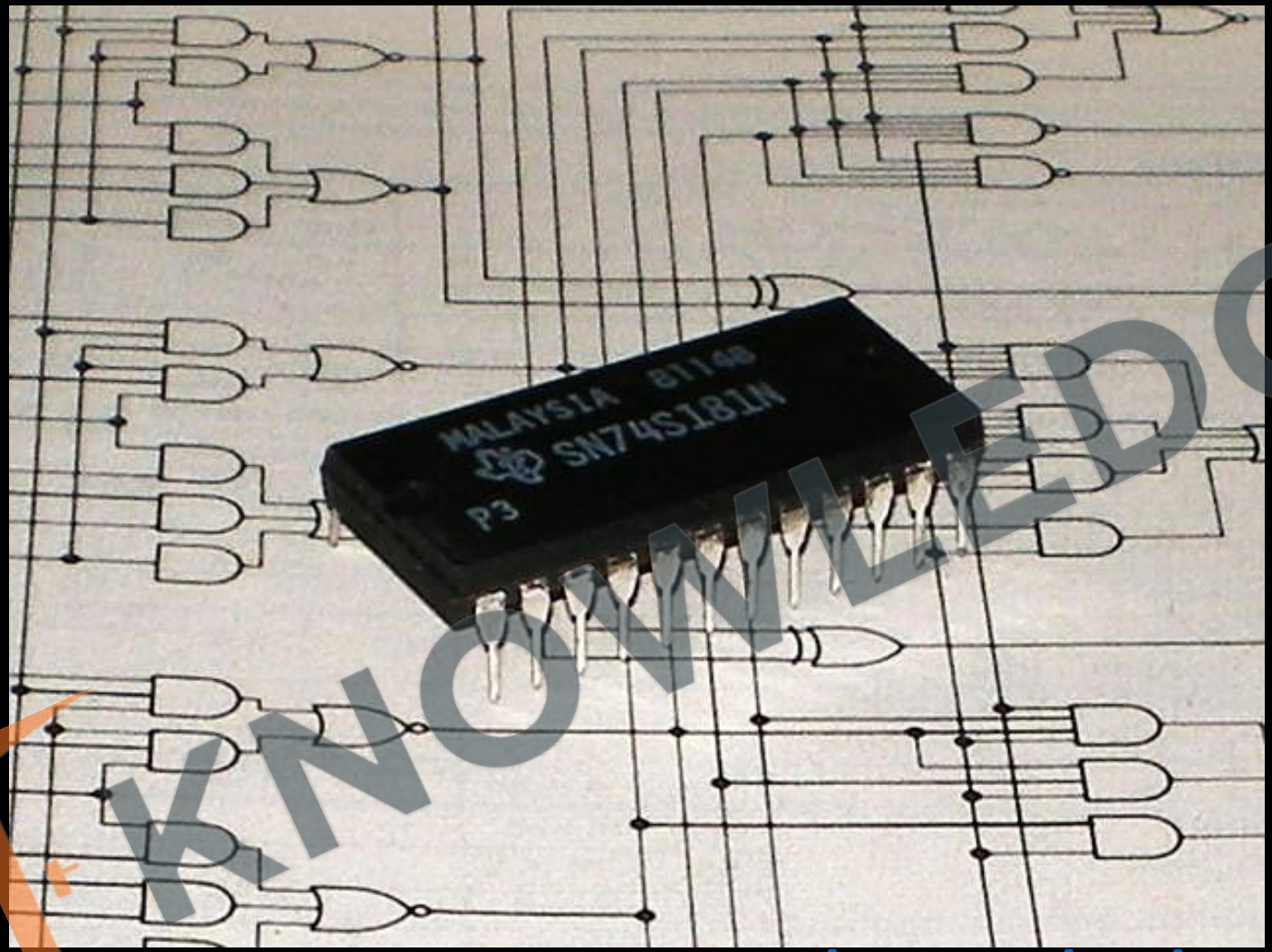
- When logic gates are connected together to produce a specified out put on certain specified combinations of input variables, with no memory involved, then the resulting circuit is called a combinational circuit.

- Output depends only on present input. O/p = f(i/p), combinational circuit performs an operation that can be specified logically by a set of Boolean function.

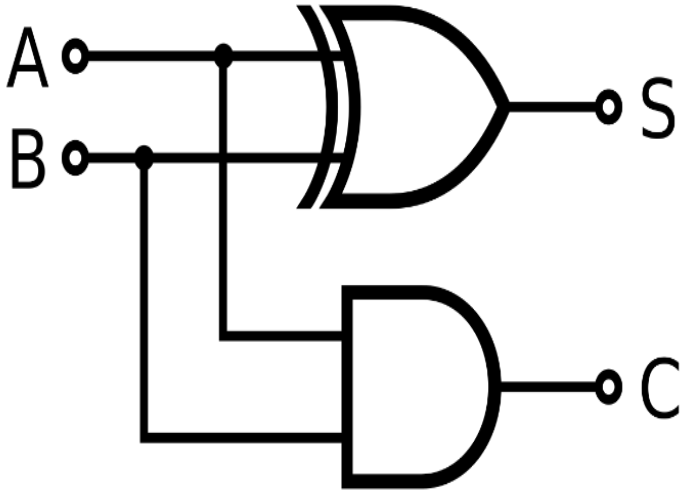
- A combinational circuit may have n-binary inputs and m-binary outputs.

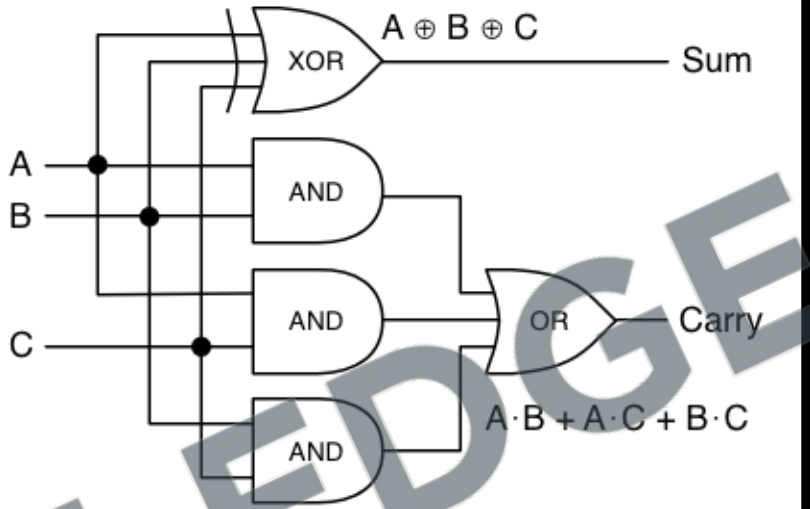


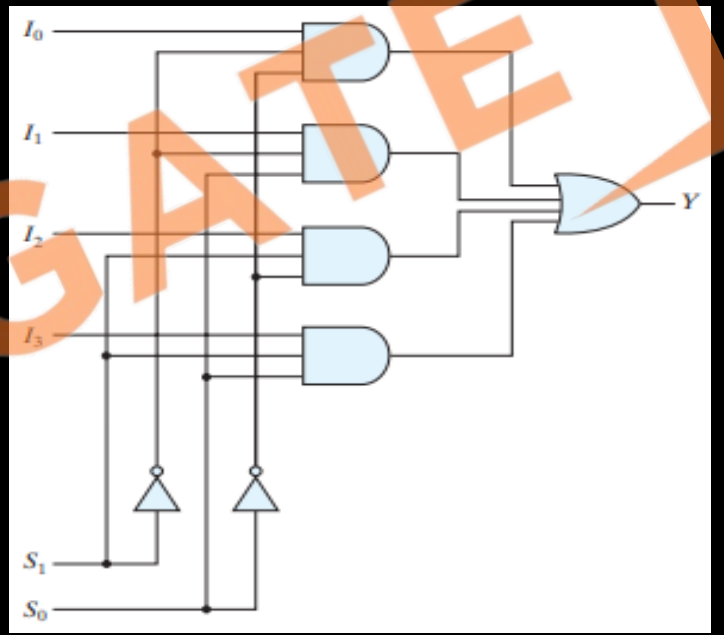
- Application, Practical computer circuits normally contain combinational and sequential logic. For e.g. the part of ALU, that does mathematical calculations is constructed using combinational logic.



- Other circuits such as half adders, full adders, half subtractors, full subtractors, multiplexers, demultiplexers, encoders and decoders are also made by using combinational logic.







www.knowledgegate.in

Design procedure: -

1. Analyse the given problem and identify the number of i/p and o/p variables.

2. Write the truth table based on the specification of the problem.

3. Convert the truth table in minimized Boolean expression using k-map.

4. Draw the logic circuit for the above obtained output expression.

Q Design a digital system for a car manufacturing company, where we want to design a warning signal for a car, there are three inputs, lights of the car(L), day or night(D), ignition (on/off).?

Solution

1) Understand the problem- here we will Understand the definition of the problem

Design the truth table –

Light	Day	Engine	Warning
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

2) Write the Boolean expression

- $W(L, D, E) = \sum_m (1, 4, 6, 7)$

- $W(L, D, E) = \prod_M (0, 2, 3, 5)$

Light	Day	Engine	Warning
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

3) Minimize Boolean expression

W = a’b’c + ab’c’ + abc’ + abc

	ab	a’b’	a’b	ab	ab’
c		00	01	11	10
c’	0	0	2	6	4
c	1	1	3	7	5

W =

Light	Day	Engine	Warning
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

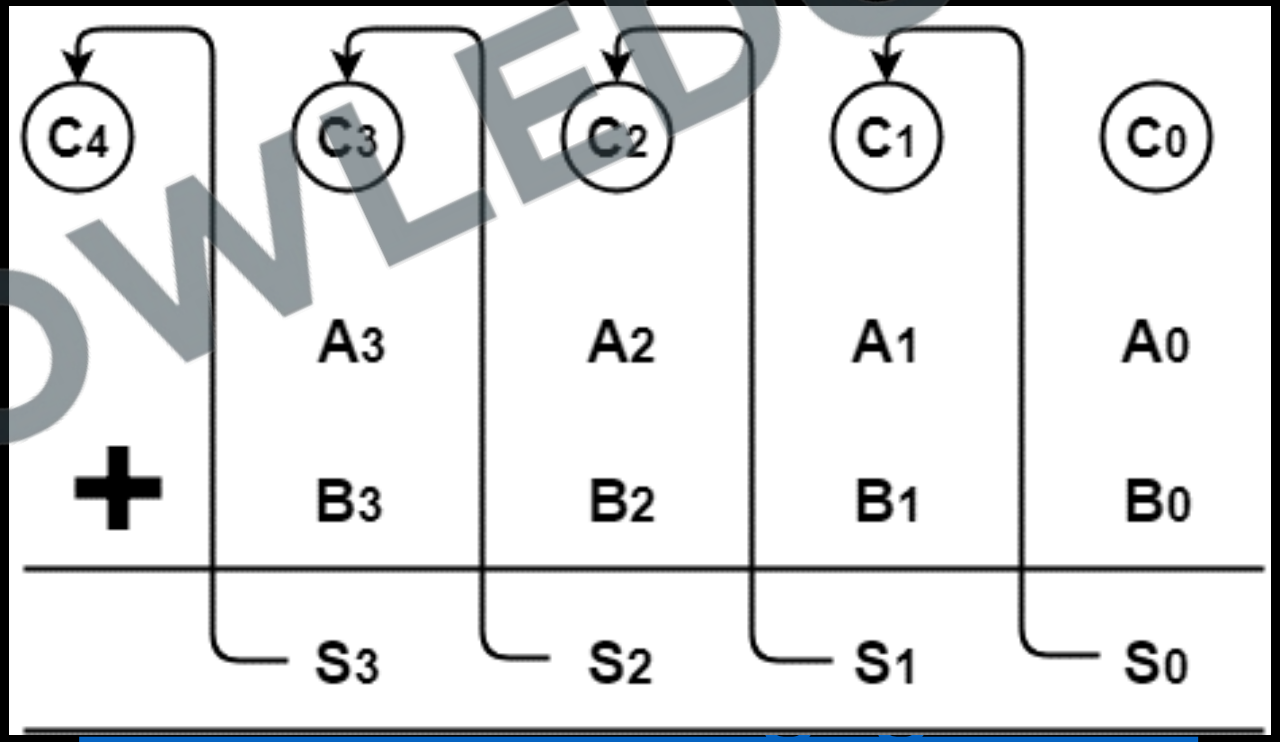
4) Implement the expression using logic gates, draw the implementation using logic gates

$W = ac' + ab + a'b'c$

Light	Day	Engine	Warning
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Adder

- An **adder** is a digital combinational circuit that performs addition of numbers. Are used in the arithmetic logic units or ALU.
- They are also utilized in other parts of the processor, where they are used to calculate addresses, table indices, increment and decrement operators, and similar operations.
- Although adders can be constructed for many number representations, such as binary-coded decimal or excess-3, the most common adders operate on binary numbers.
- Can also be used to perform substruction In cases of 2's representation.



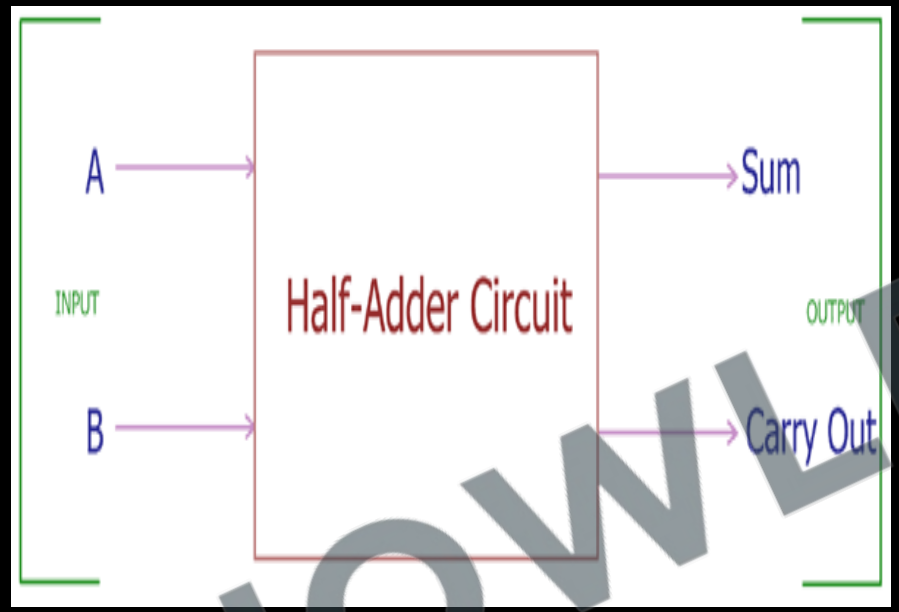
Half adder

- The simplest form of addition is addition of two binary digits, consists of four possible elementary operations

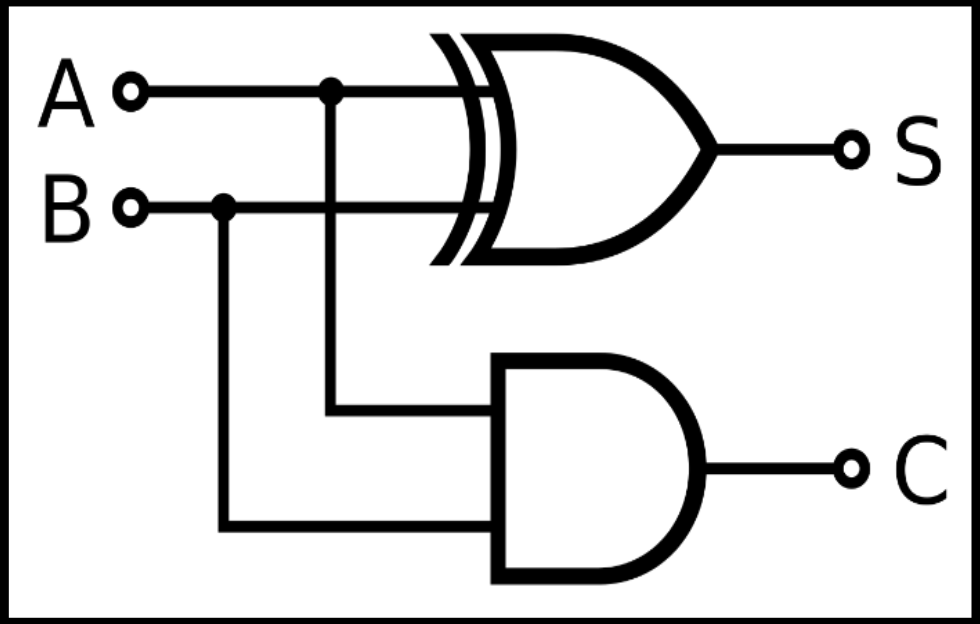
$$\begin{array}{r} 1 \\ +1 \\ \hline \\ \hline \end{array} \qquad \begin{array}{r} 1 \\ +0 \\ \hline \\ \hline \end{array} \qquad \begin{array}{r} 0 \\ +1 \\ \hline \\ \hline \end{array} \qquad \begin{array}{r} 0 \\ +0 \\ \hline \\ \hline \end{array}$$

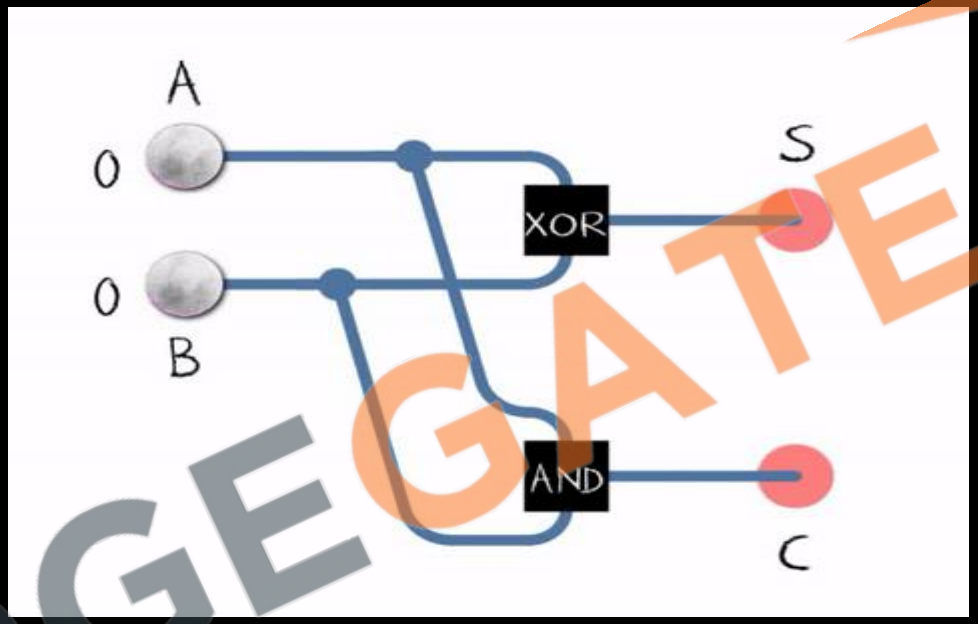
- The first three operations produce a sum of two digits, but when both augend and addend bits are equal to 1, the binary sum consists of two digits. The higher significant bit of this result is called a carry.

- It is a combinational circuit, which perform the arithmetic addition of two one-bit binary numbers is referred to as an half-adder.
- So, in half adder inputs are adds two single binary bits *A* and *B*, and two outputs, sum (*S*) and carry (C).



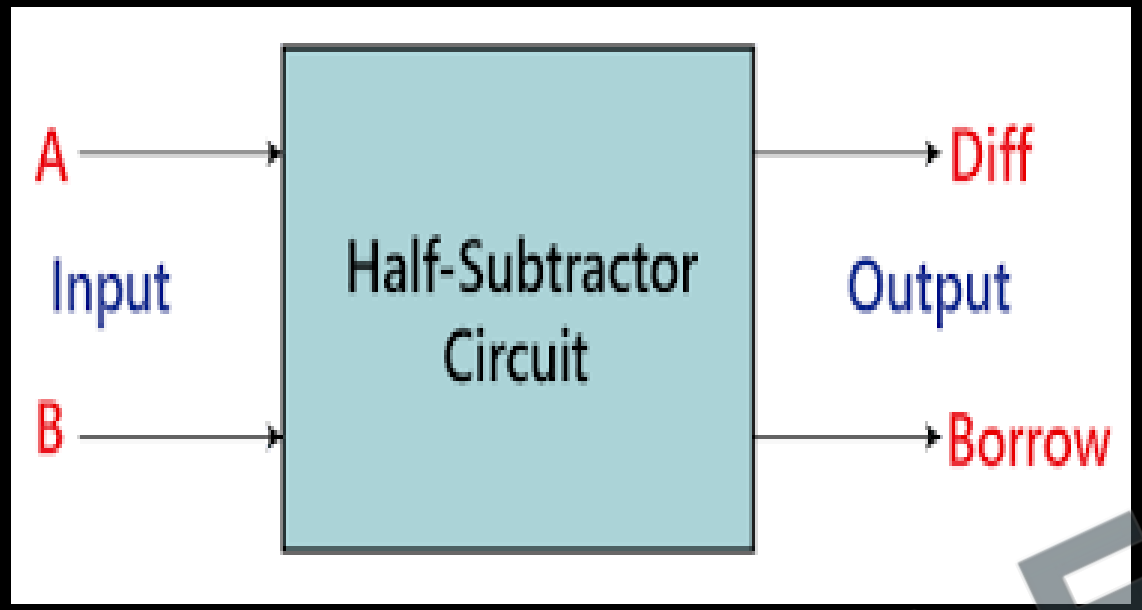
INPUTS		OUTPUTS	
A	B	Carry	Sum
0	0		
0	1		
1	0		
1	1		



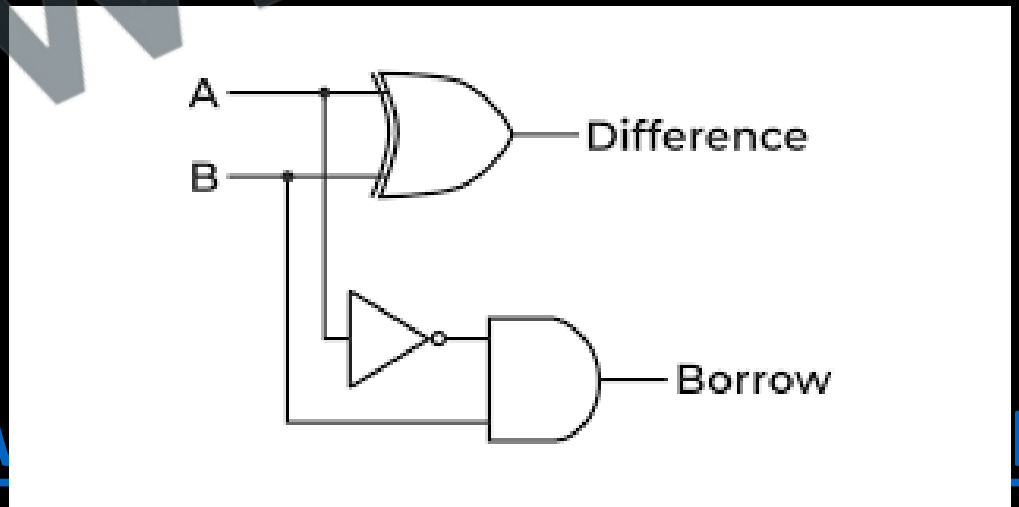


- Cost of implementation a half adder is one EX-OR gate and one AND gate.
- A half adder has only two inputs and there is no provision to add a carry coming from the lower order bits when multi bit number addition is performed. For this reason, we have designed a full adder.

Half subtractor

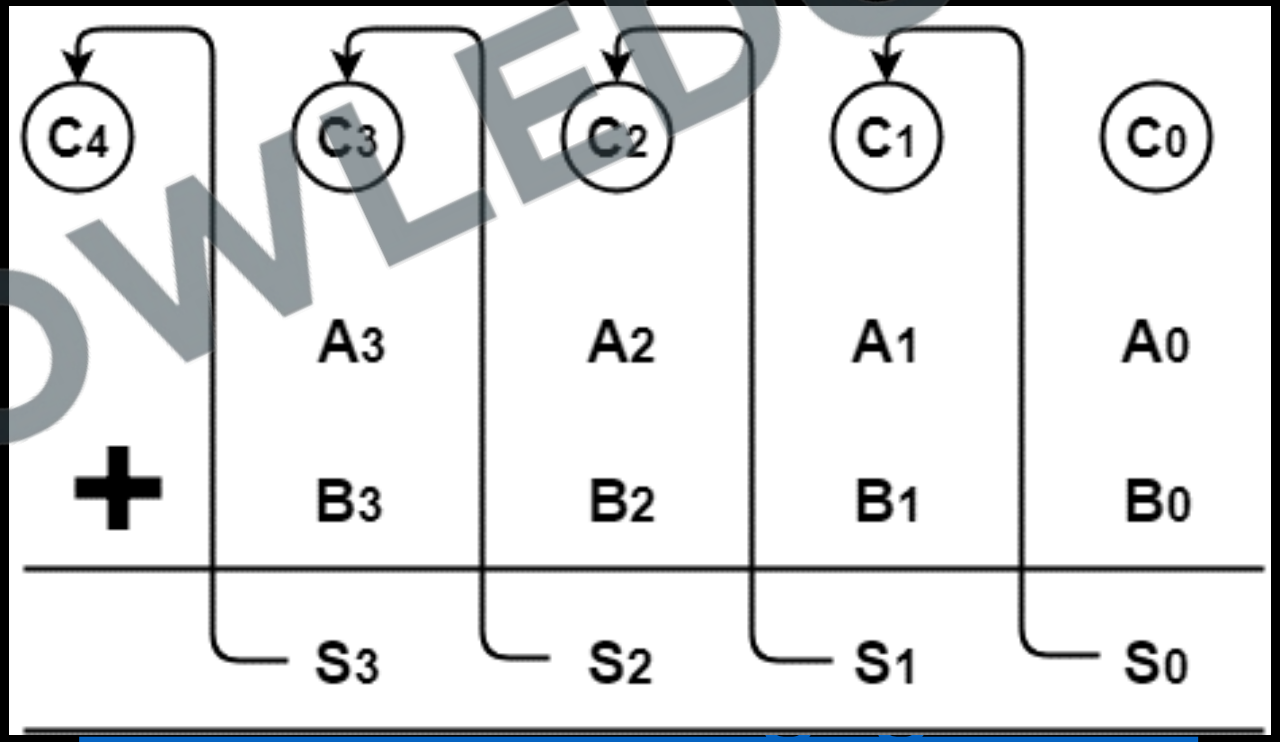


Inputs		Outputs	
A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

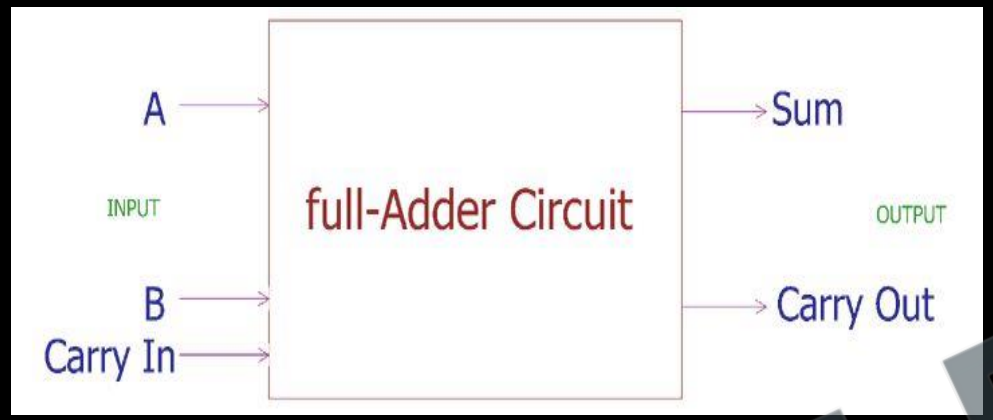


Full adder

1. A full adder is a combinational logic circuit that performs the arithmetic sum of three input bits.
2. Where A_n, B_n are the n^{th} order bits of the number A and B respectively and C_n is the carry generated from the addition of $(n-1)^{th}$ order bits.
3. It consists of three input bits, denoted by A (First operand), B (Second operand), C_{in} (Represents carry from the previous lower significant position).



- Two output bits are same as of half adder, which is Sum and $Carry_{out}$.
- When the augend and addend number contain more significant digits, the carry obtained from the addition of two bits is added to the next higher order pair of significant bits.

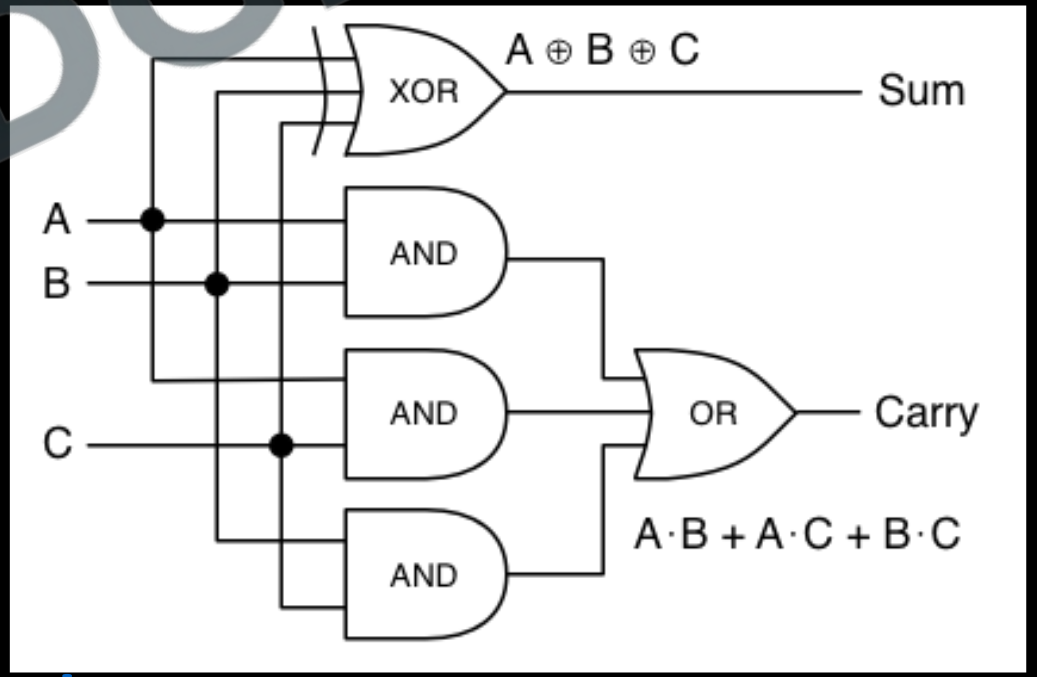


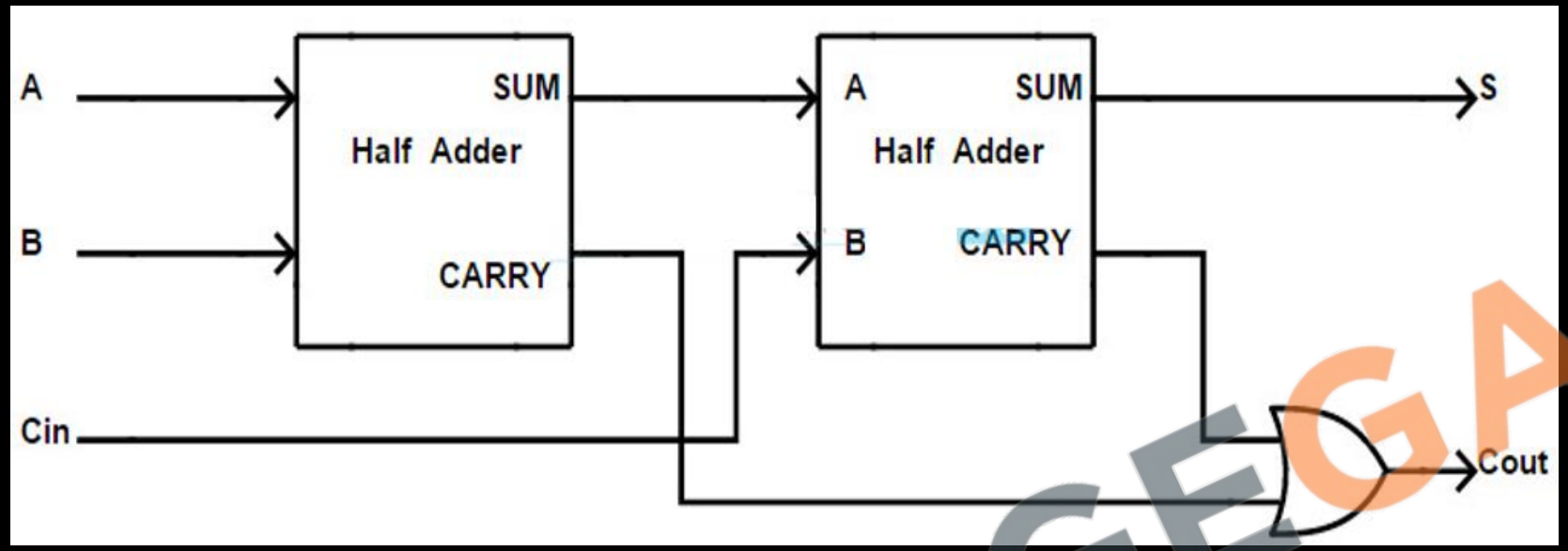
INPUTS			OUTPUTS	
A	B	C_{in}	C_{out}	Sum
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

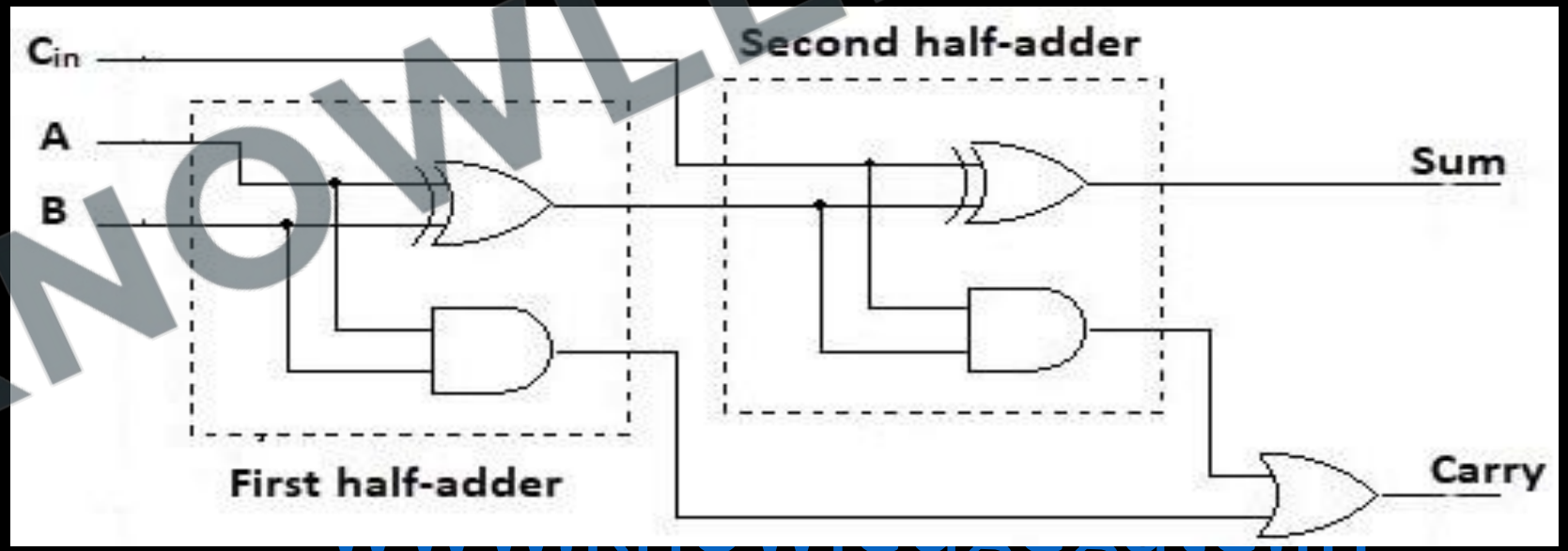
	ab	a'b'	a'b	ab	ab'
c_{in}		00	01	11	10
c_{in}'	0	0	2	6	4
c_{in}	1	1	3	7	5

	ab	a'b'	a'b	ab	ab'
c_{in}		00	01	11	10
c_{in}'	0	0	2	6	4
c_{in}	1	1	3	7	5

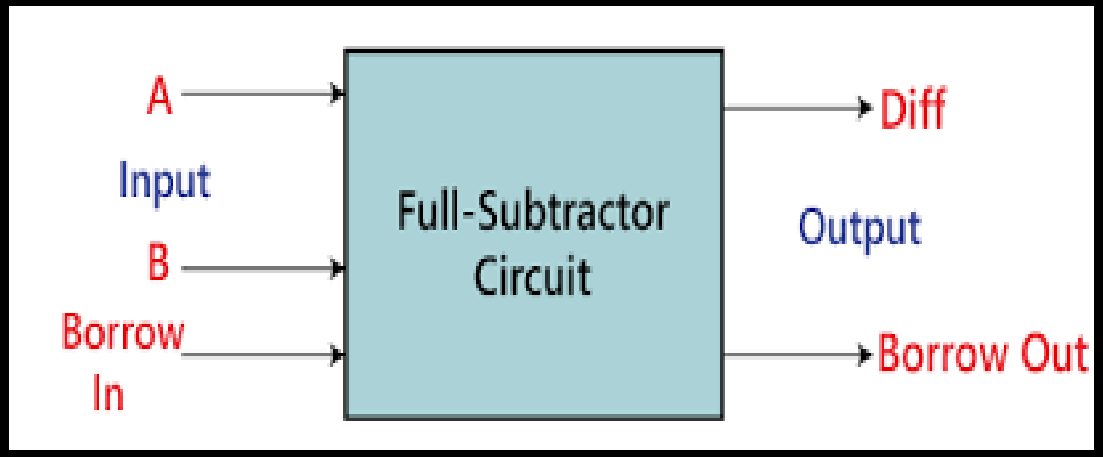
INPUTS			OUTPUTS	
A	B	C_{in}	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

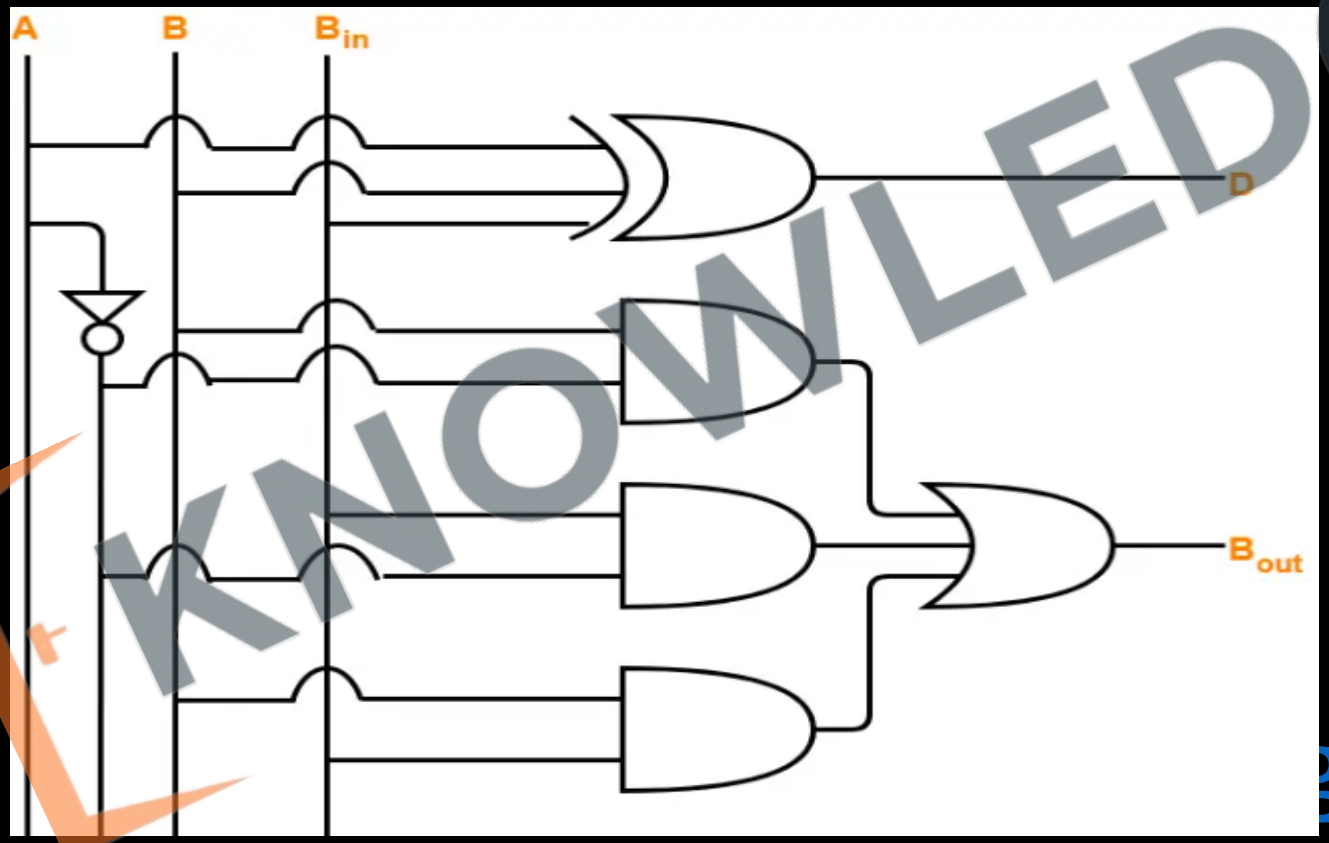




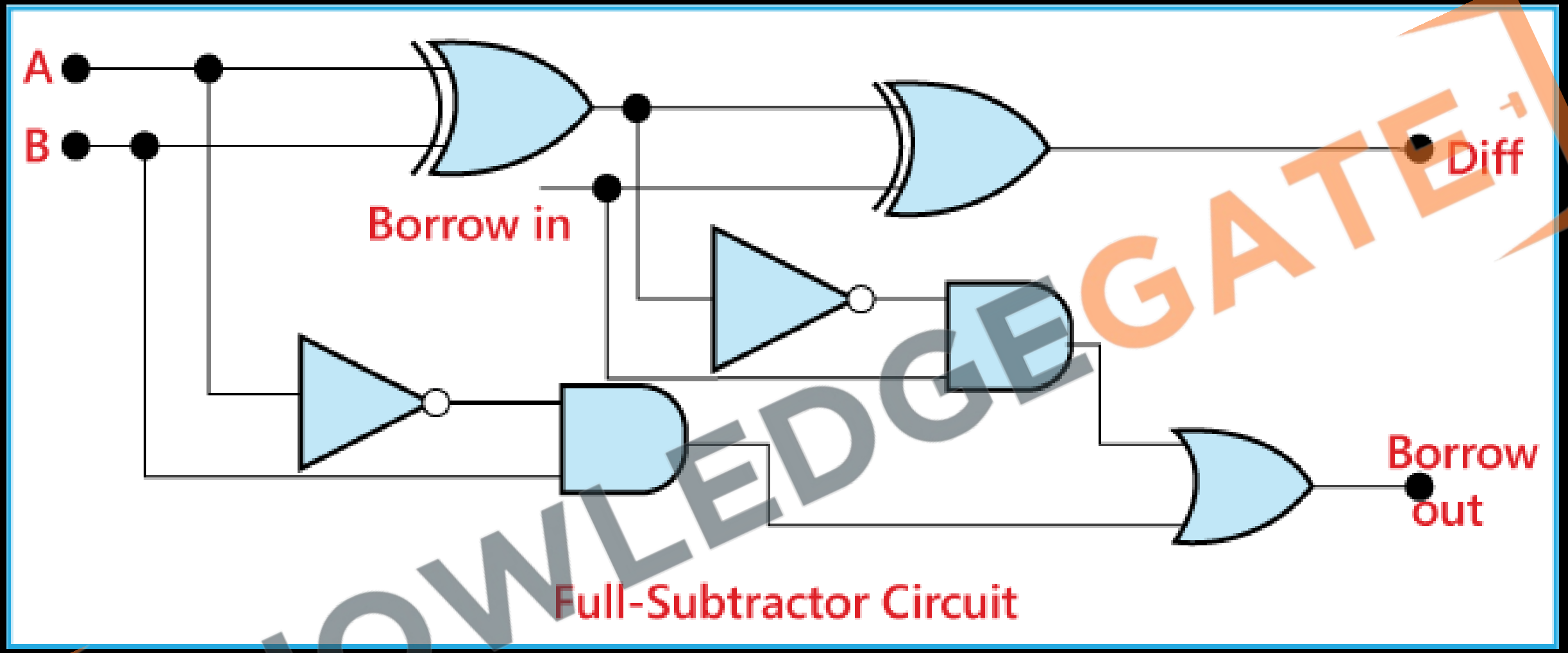


Full subtractor





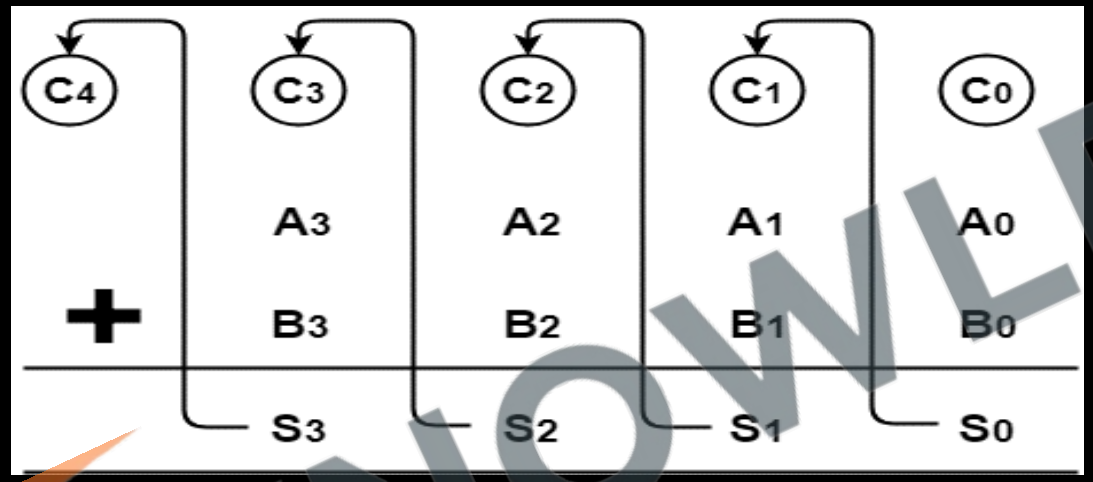
Inputs			Outputs	
A	B	$Borrow_{in}$	Diff	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

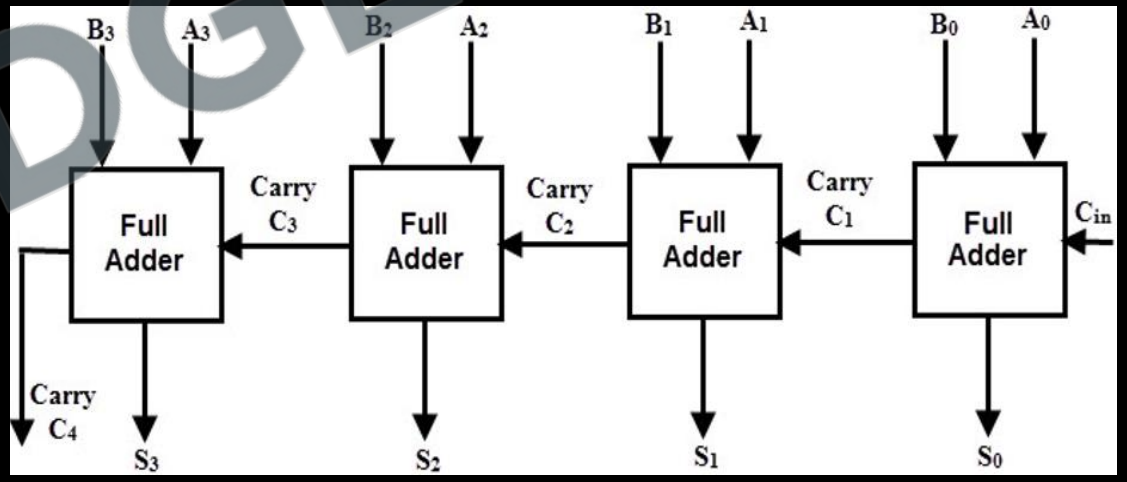


Full-Subtractor Circuit

Four-bit parallel binary adder / Ripple adder

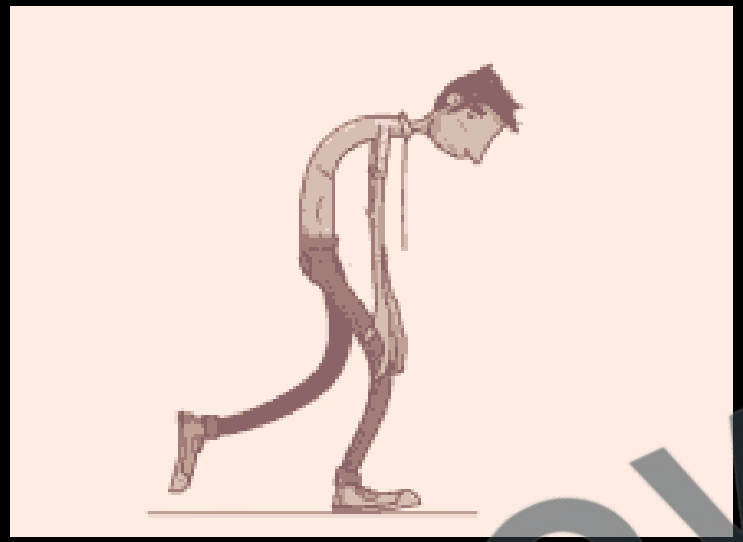
- As we know that full adder is capable of adding two 1 bit number and 1 previous carry, but in order to add binary numbers with more than one bits, additional full adders must be employed. For e.g. a four bit binary adder can be constructed using four full adders.
- Theses four full adders are connected in cascade, carry output of each adder is connected to the carry input of the next higher-order adder. So a n-bit parallel adder is constructed using ‘n’ number of full adders.





- There are some scope of improvement in parallel binary adder / Ripple adder is it is very slow

Carry propagation delay

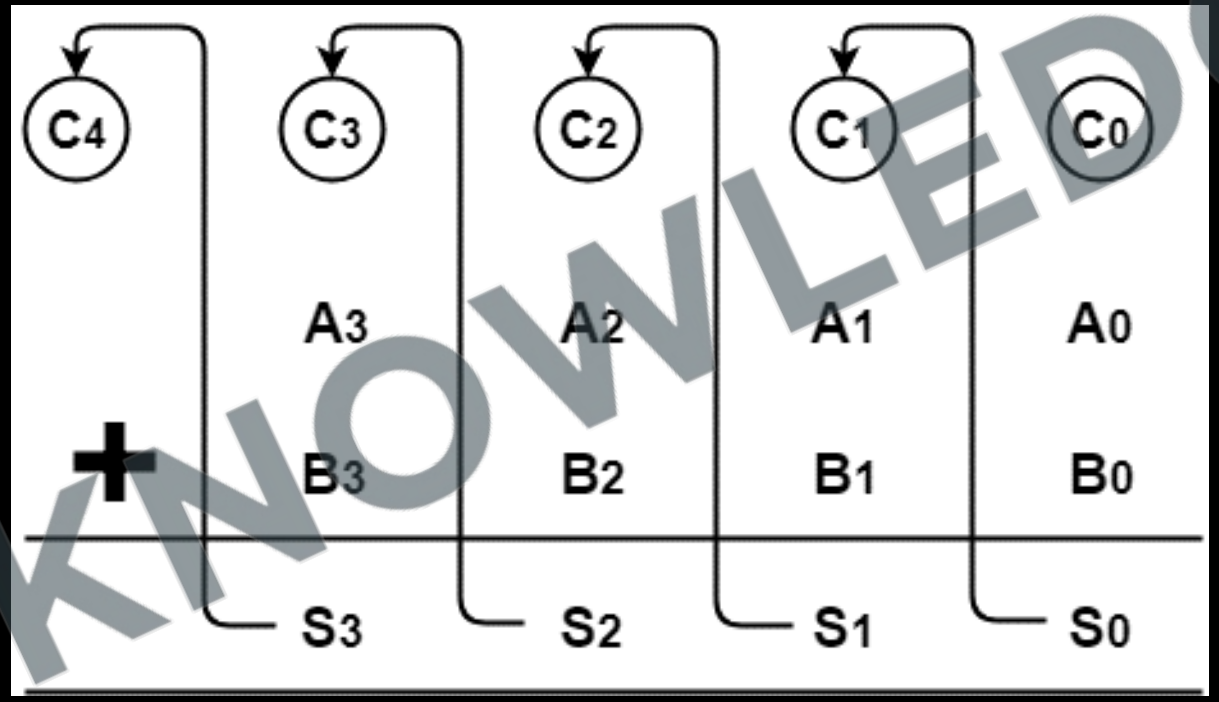


Look ahead Carry Generator

Look ahead carry adder

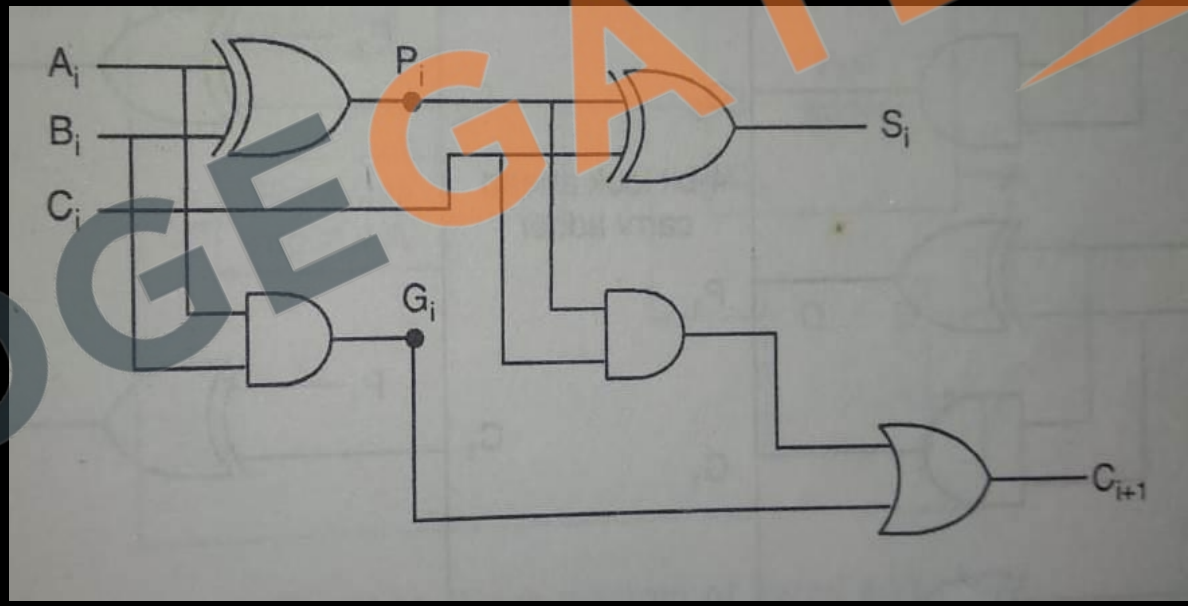
- The carry propagation time is an important attribute of the adder because it limits the speed with which two numbers are added. The solution to delay is to increase the complexity of the equipment in such a way that the carry delay time is reduced.

- To solve this problem most widely used technique employs the principle of 'look ahead carry'. This method utilizes logic gates to look at the lower order bits of the augend and addend to see if a higher order carry is to be generated. It uses two functions carry generate G_i and carry propagate P_i



$A(A_3\ A_2\ A_1\ A_0)$
$B(B_3\ B_2\ B_1\ B_0)$

- G_i is called a ***carry generate***, and it produces a carry of 1 when both A_i and B_i are 1, regardless of the input carry C_i.
- P_i is called a ***carry propagate***, because it determines whether a carry into stage i will propagate into stage i + 1.
- We now write the Boolean functions for the carry outputs of each stage and substitute the value of each C_i from the previous equations:

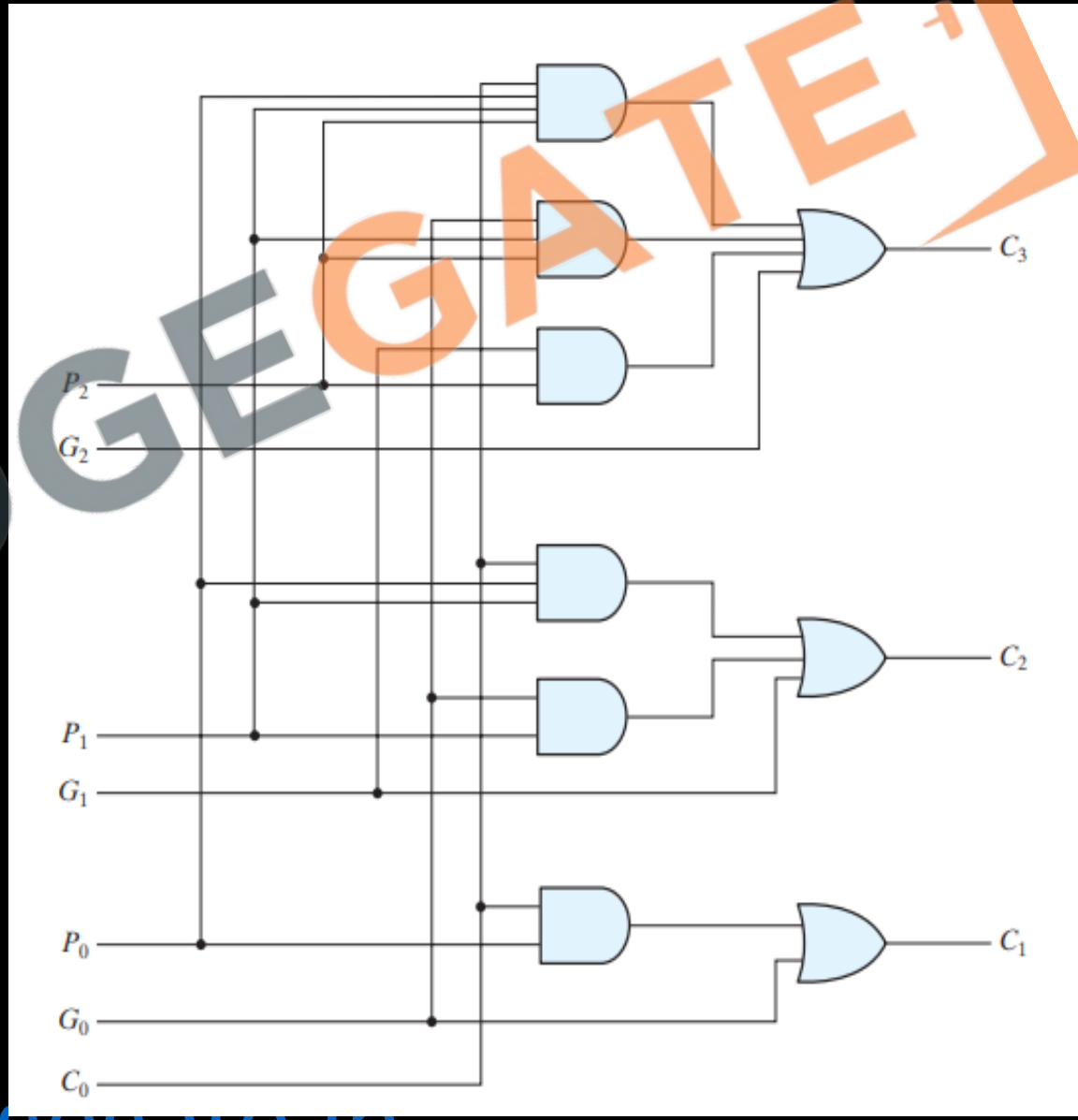


- $P_i = A_i \oplus B_i$
  $G_i = A_i \cdot B_i$
  $S_i = P_i \oplus C_i$
  $C_{i+1} = G_i + P_i C_i$

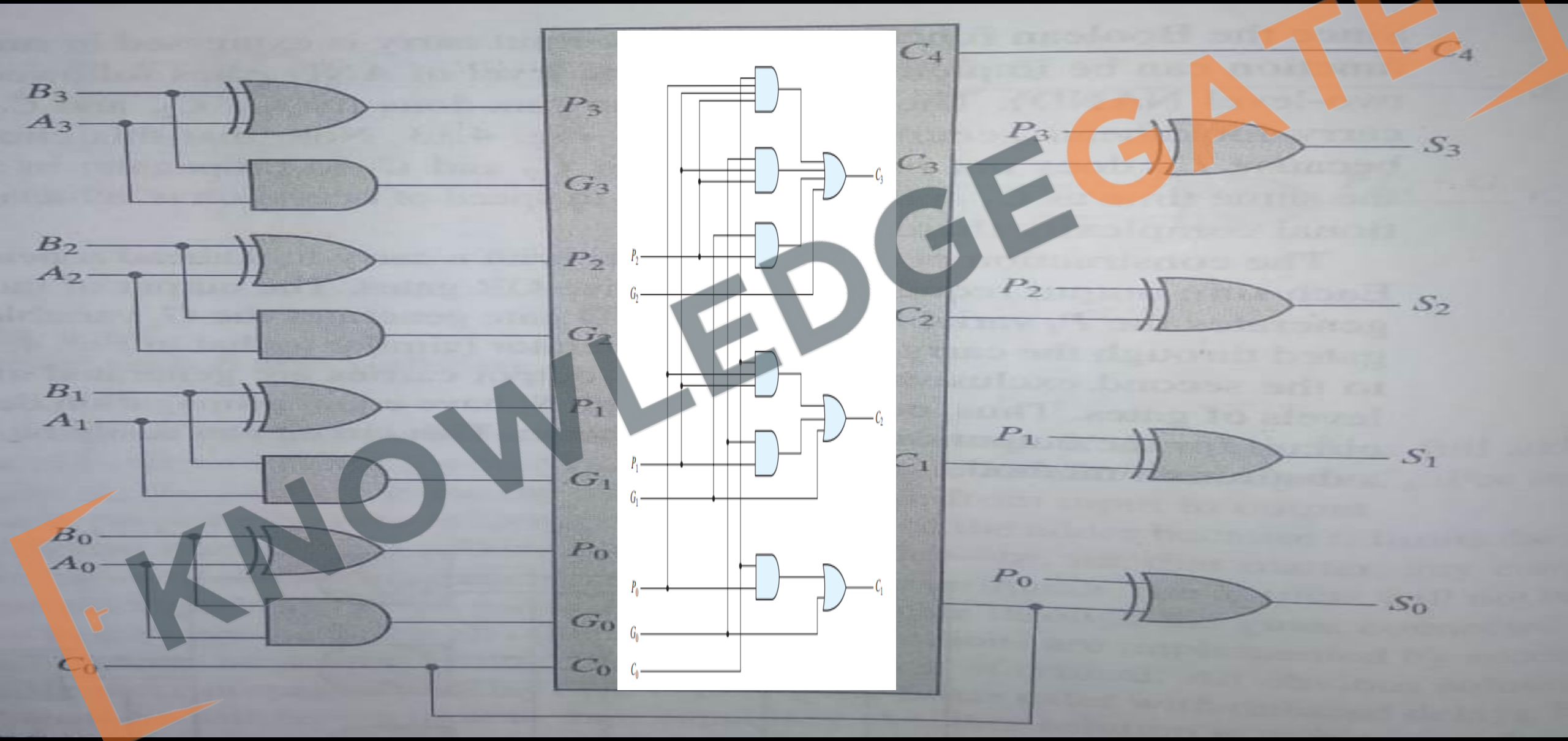
- $C_0 = 0$
- $C_1 = G_0 + P_0 C_0$
- $C_2 = G_1 + P_1G_0 + P_1P_0 C_0$
- $C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1 P_0 C_0$
- $C_4 = G_3 + G_2.P_3 + G_1.P_2.P_3 + G_0.P_1.P_2.P_3 + C_0.P_0.P_1.P_2.P_3$

- Since the Boolean function for each output carry is expressed in sum-of-products form. Each function can be implemented with one level of AND gates followed by an OR gate (or by a two-level NAND).

- $C_0 = 0$
- $C_1 = G_0 + P_0 C_0$
- $C_2 = G_1 + P_1G_0 + P_1P_0 C_0$
- $C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1 P_0 C_0$
- $C_4 = G_3 + G_2.P_3 + G_1.P_2.P_3 + G_0.P_1.P_2.P_3 + C_0.P_0.P_1.P_2.P_3$

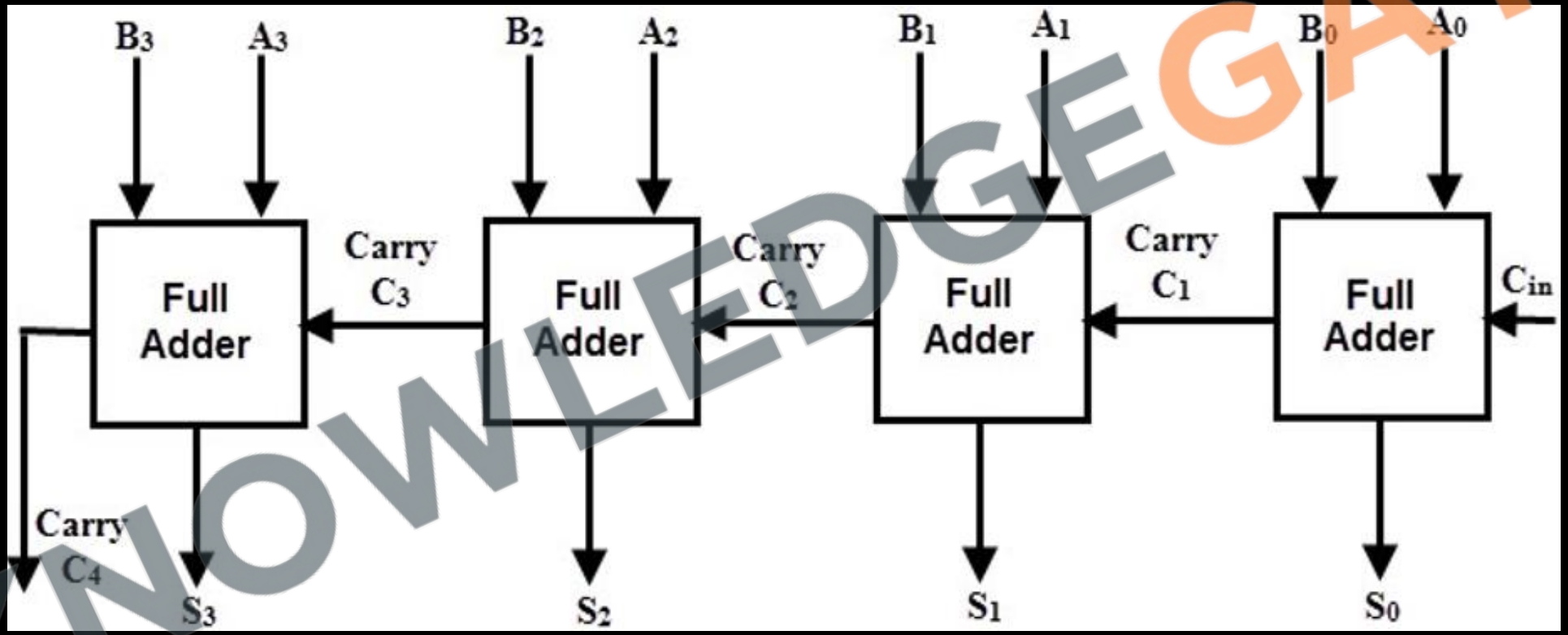


- All output carries are generated after a delay through two levels of gates. Thus, outputs S_1 through S_3 have equal propagation delay times.

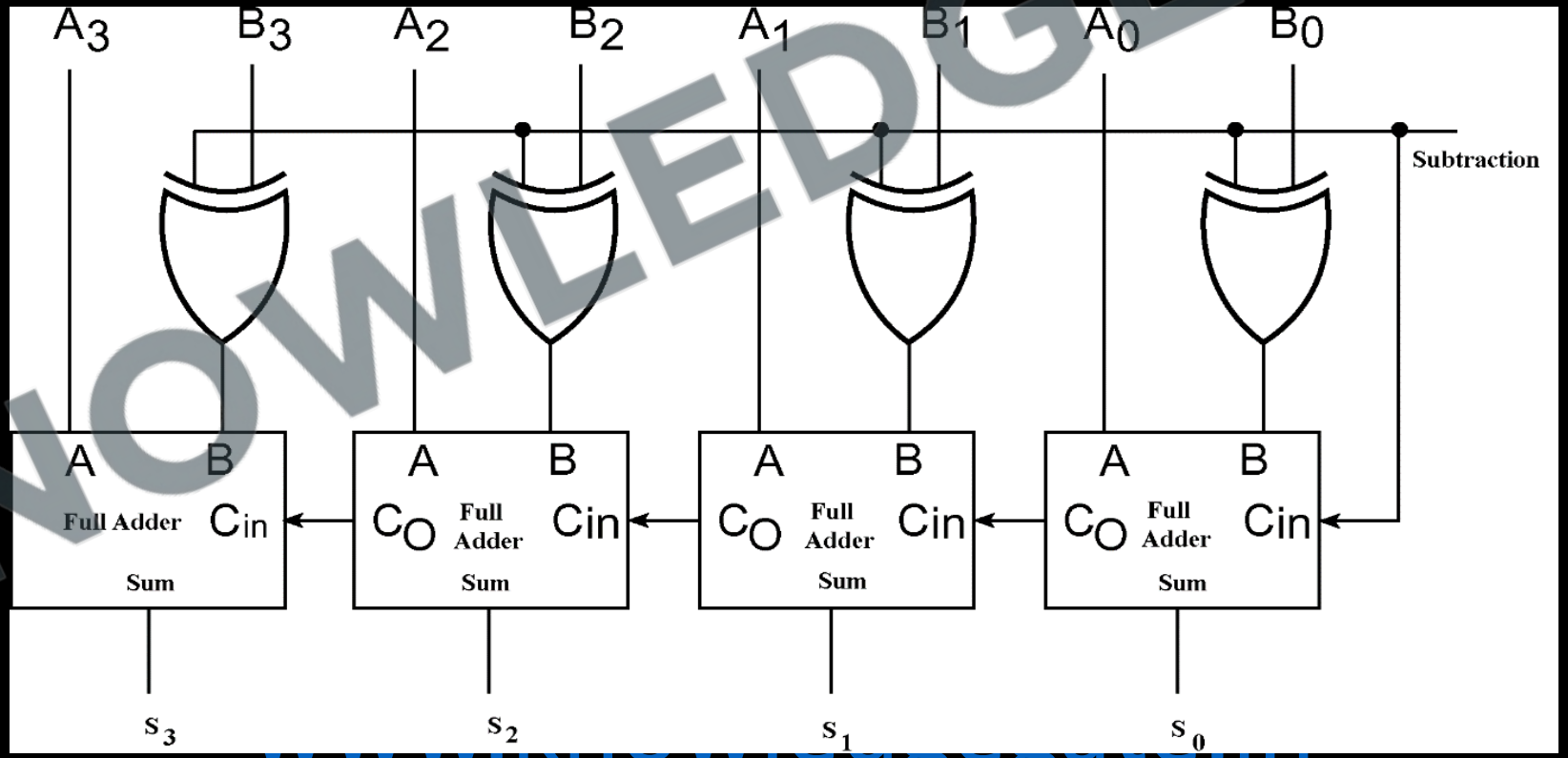


Four-bit ripple adder/subtractor

- The subtraction A - B can be done by taking the 2's complement of B and adding it to A.
- A + (-B)



- The circuit for subtracting A - B consists of an adder with inverters placed between each data input B and the corresponding input of the full adder. The mode input M controls the operation. When M = 0, the circuit is an adder, and when M = 1, the circuit becomes a subtractor.
- When M = 0, we have B $\oplus$ 0 = B. The full adders receive the value of B, the input carry is 0, and the circuit performs A plus B.
- When M = 1, we have B $\oplus$ 1 = B' and C_0 = 1. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation A plus the 2's complement of B.



A_1	A_0	B_1	B_0	A>B	A<B	A=B
0	0	0	0	0	0	1
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	1
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	0	1
1	0	1	1	0	1	0
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	0	1

A>B	a_1a_0	$a'_1a'_0$	a'_1a_0	a_1a_0	$a_1a'_0$
b_1b_0		00	01	11	10
$b'_1b'_0$	00		1	1	1
b'_1b_0	01			1	1
B_1b_0	11				
$B_1b'_0$	10			1	

A=B	a_1a_0	$a'_1a'_0$	a'_1a_0	a_1a_0	$a_1a'_0$
b_1b_0		00	01	11	10
$b'_1b'_0$	00	1			
b'_1b_0	01		1		
B_1b_0	11			1	
$B_1b'_0$	10				1

A<B	a_1a_0	$a'_1a'_0$	a'_1a1_0	a_1a_0	$a_1a'_0$
b_1b_0		00	01	11	10
$b'_1b'_0$	00				
b'_1b_0	01	1			
B_1b_0	11	1	1		1
$B_1b'_0$	10	1	1		

A>B	a_1a_0	$a'_1a'_0$	a'_1a_0	a_1a_0	$a_1a'_0$
b_1b_0		00	01	11	10
$b'_1b'_0$	00		1	1	1
b'_1b_0	01			1	1
B_1b_0	11				
$B_1b'_0$	10			1	

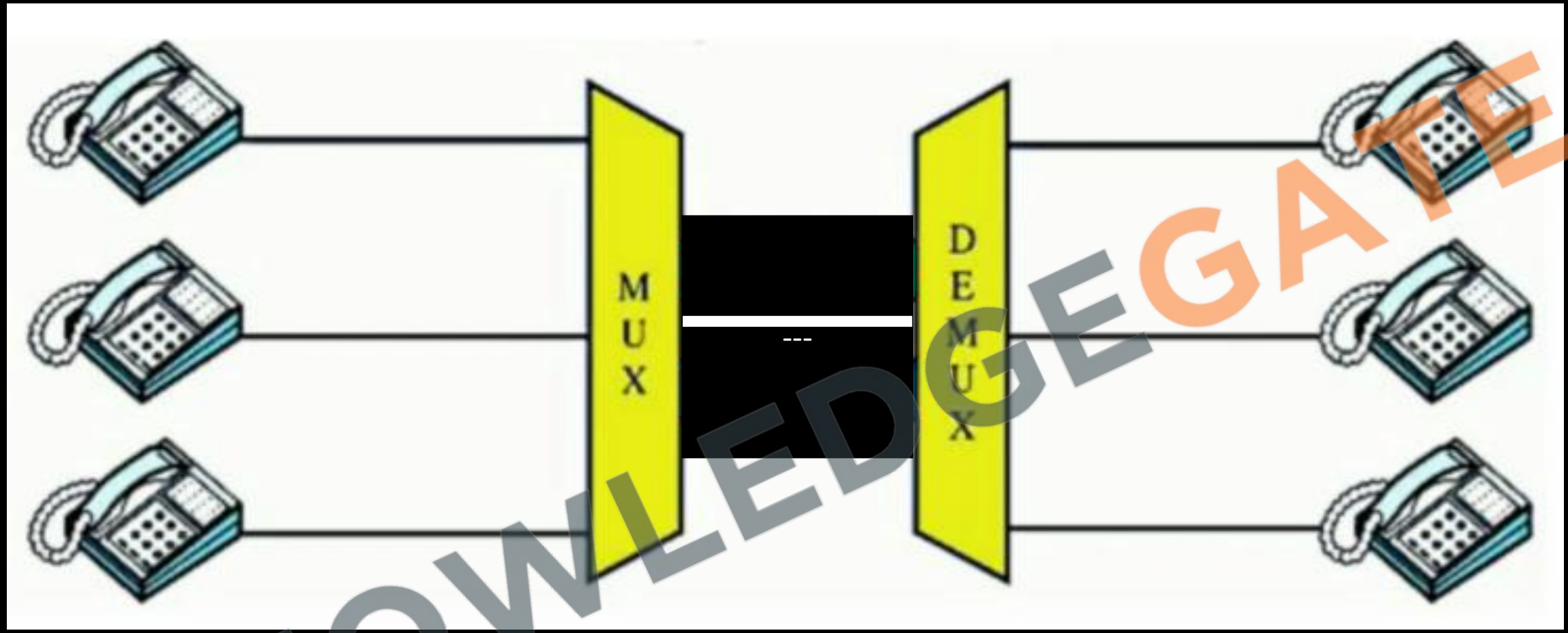
$$A>B = a_0b_1b_0 + a_1a_1b_1 + a_1b_1$$

A=B	a_1a_0	$a'_1a'_0$	a'_1a_0	a_1a_0	$a_1a'_0$
b_1b_0		00	01	11	10
$b'_1b'_0$	00	1			
b'_1b_0	01		1		
B_1b_0	11			1	
$B_1b'_0$	10				1

$$A=B = (a_1 \odot b_1)\,(a_0 \odot b_0)$$

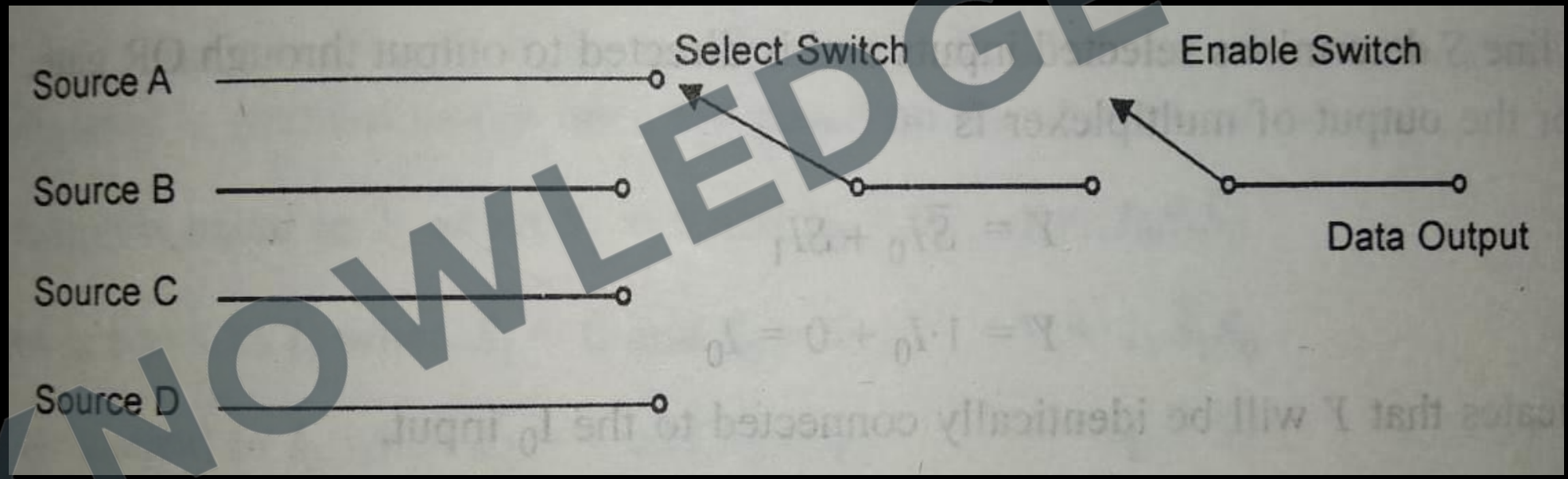
A<B	a_1a_0	$a'_1a'_0$	a'_1a1_0	a_1a_0	$a_1a'_0$
b_1b_0		00	01	11	10
$b'_1b'_0$	00				
b'_1b_0	01	1			
B_1b_0	11	1	1		1
$B_1b'_0$	10	1	1		

$$A<B = a_1a_0b_0 + a_1b_1 + a_0b_1b_0$$

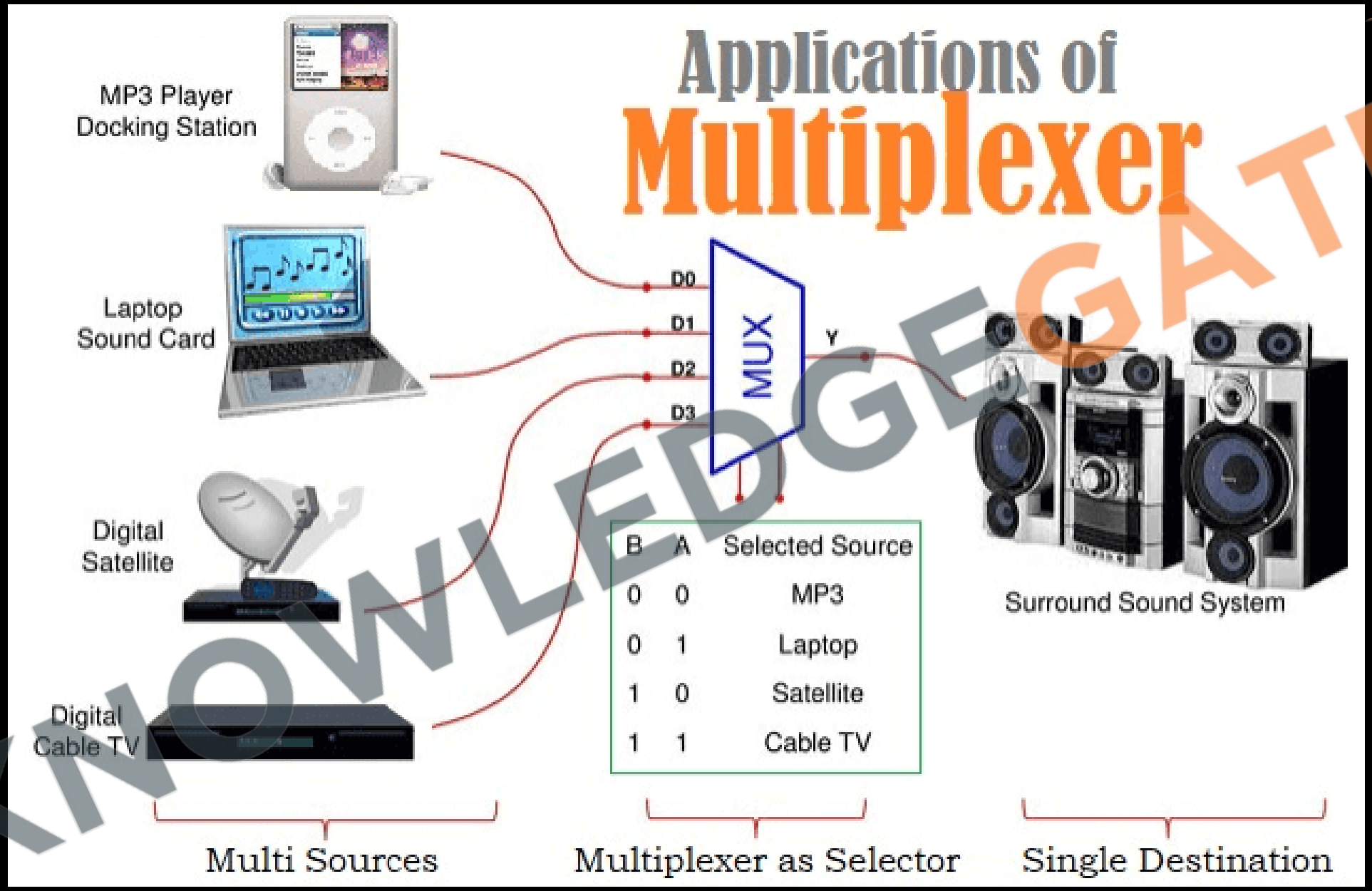


Multiplexer (Selector)

- Multiplexer are special and one of the most widely used combinational circuits.
- Main requirement is out of many inputs we have to select one for e.g. telephone or train leaving the station. So multiplexer do not perform any logical operation or comparison, it just acts as a switch or relay.



Applications of Multiplexer



B	A	Selected Source
0	0	MP3
0	1	Laptop
1	0	Satellite
1	1	Cable TV

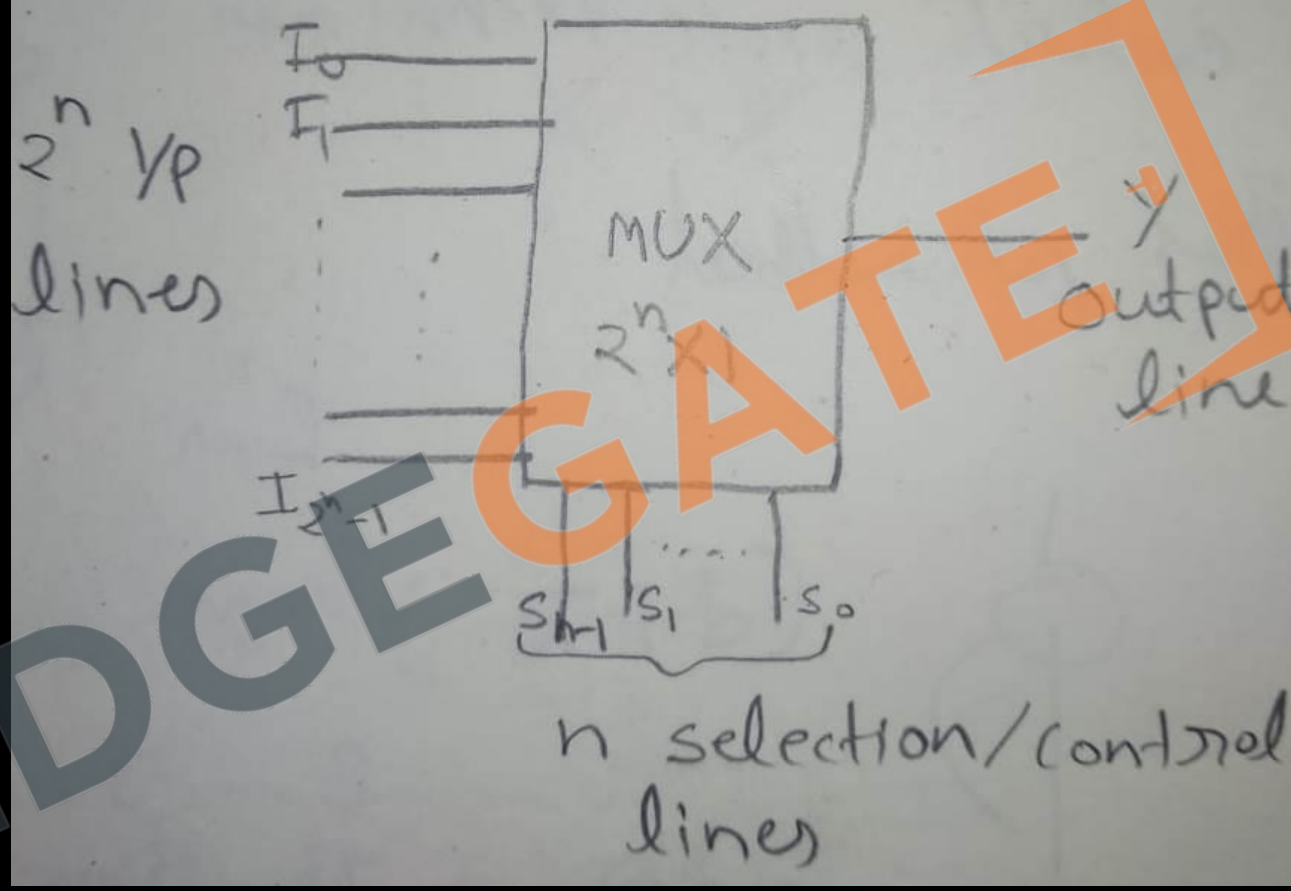
1. A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.

2. A multiplexer is also called a data selector, since it selects one of many inputs and steers the binary information to the output line.

3. The selection of a particular input line is controlled by a set of selection lines.

4. There are 2^n input lines and n selection lines whose bit combinations determine which input is to be selected.

5. No of Input line =< 2^n



Note:

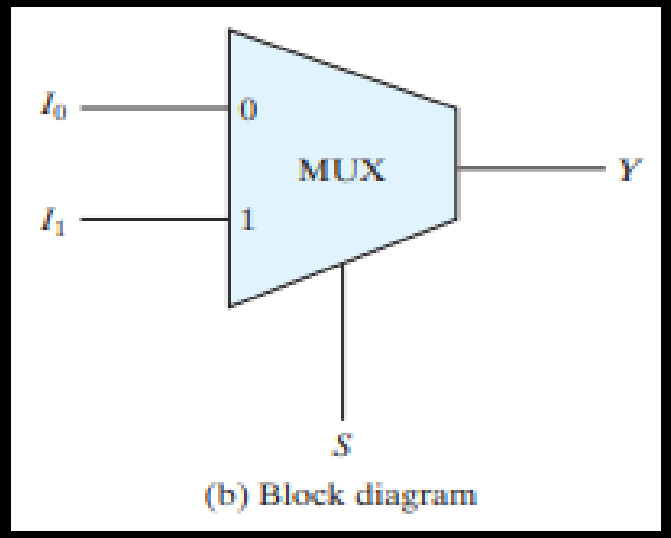
- Can never have two i/p connected to out at any time

Application

- Parallel data to serial data conversion
- Used as data selector, as the output of a multiplexer is directed from one of various inputs
- Used in implementation of Boolean functions
- Used in communication systems, **Computer Memory, Telephone Network, Transmission from the Computer System of a Satellite**

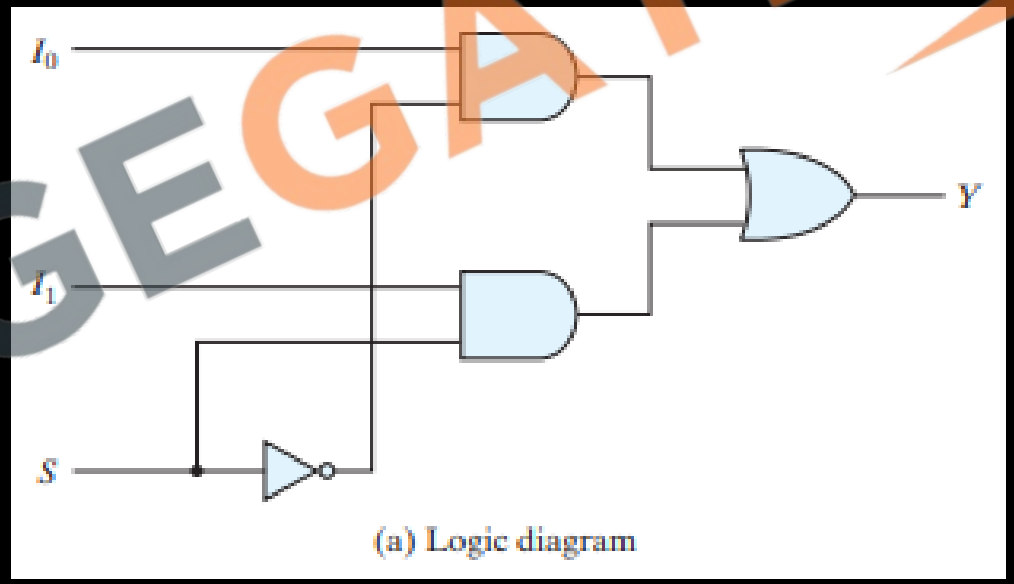
A two-to-one-line multiplexer

- When S = 0, the upper AND gate is enabled and I_0 has a path to the output.
- When S = 1, the lower AND gate is enabled and I_1 has a path to the output.



(b) Block diagram

Input	Output
S	Y
0	I_0
1	I_1

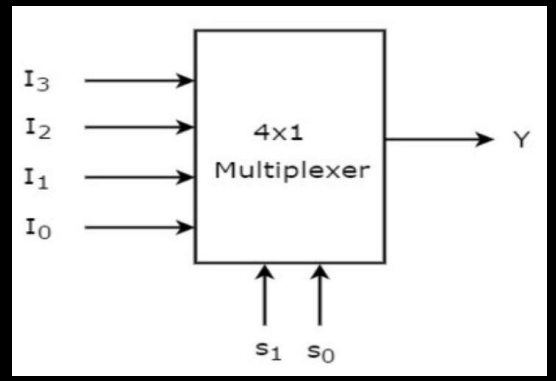


(a) Logic diagram

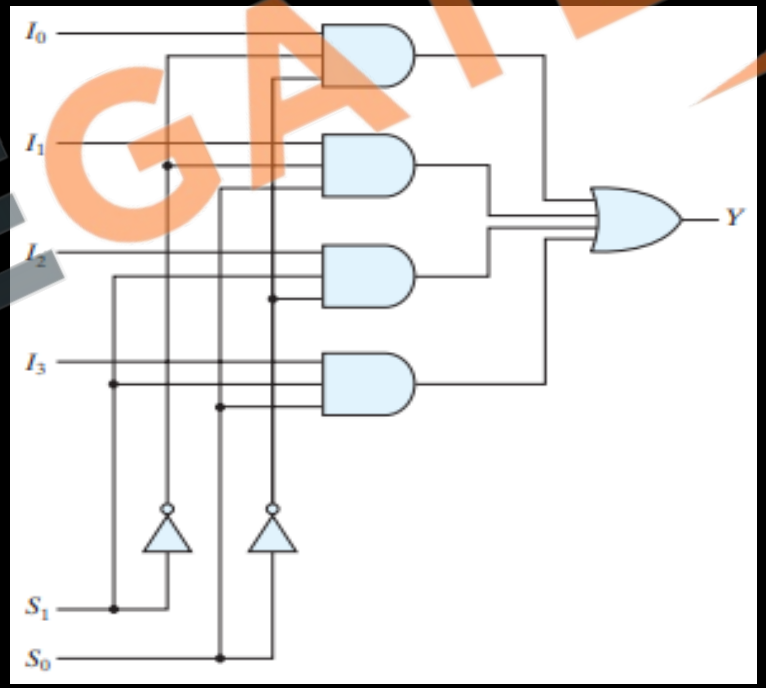
Characteristic equation: $Y = S_0' I_0 + S_0 I_1$

Case study of 4 to 1

- Each of the four inputs, I_0 through I_3, is applied to one input of an AND gate.
- Selection lines S_1 and S_0 are decoded to select a particular AND gate.
- The outputs of the AND gates are applied to a single OR gate that provides the one-line output.



Input		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

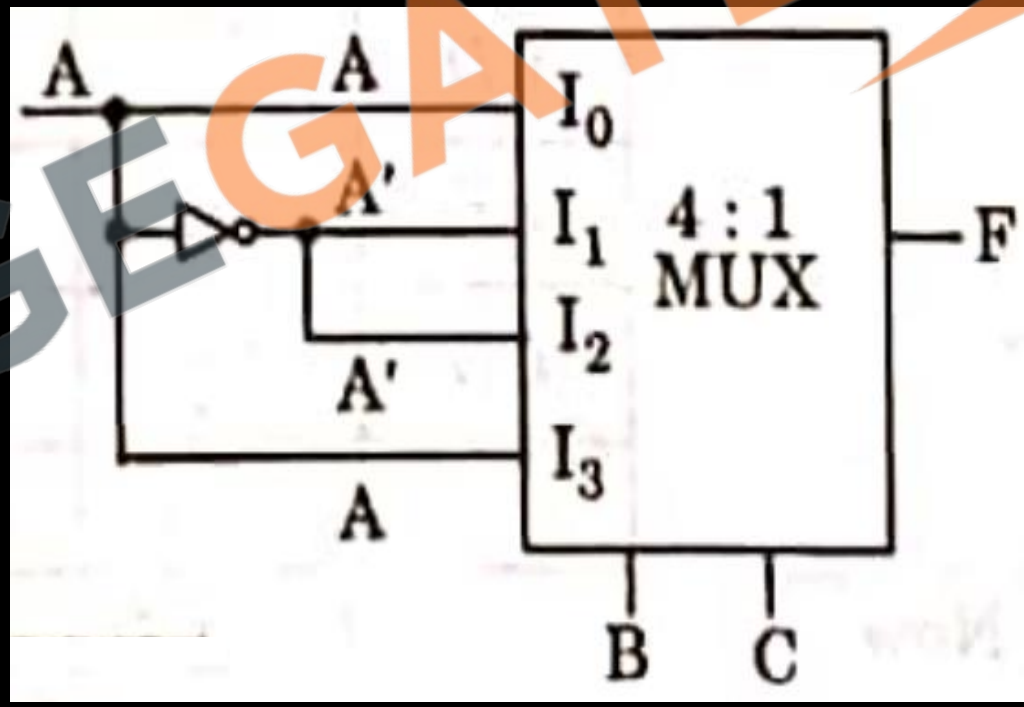


Characteristic equation

$$Y = E\,[(S'_1\,S'_0\,I_0) + (S'_1\,S_0\,I_1) + (S_1\,S'_0\,I_2) + (S_1\,S_0\,I_3)]$$

Q Implement the function F(a,b,c) = a'b'c + a'bc' + ab'c' + abc. With b and c variable on select line?

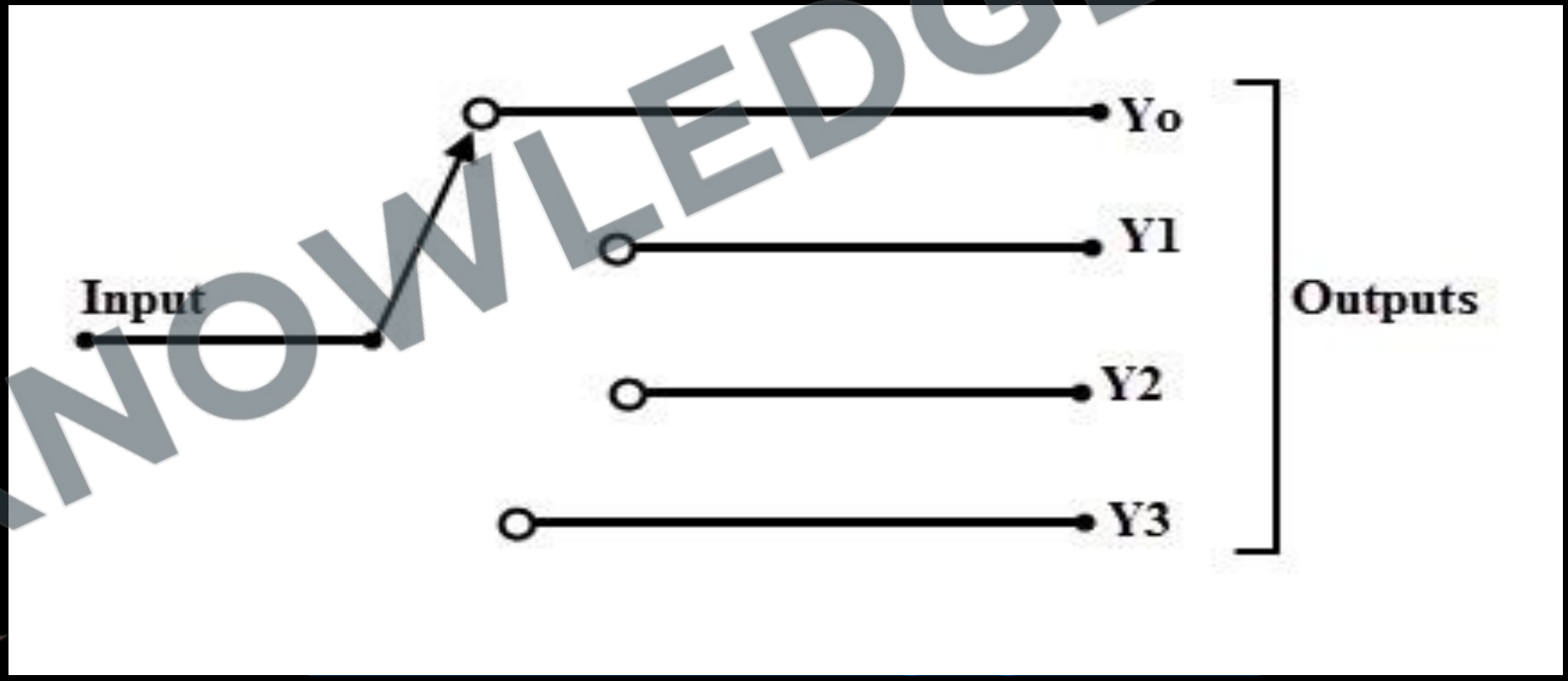
	I_0	I_1	I_2	I_3
A'	0	1	2	3
A	4	5	5	7



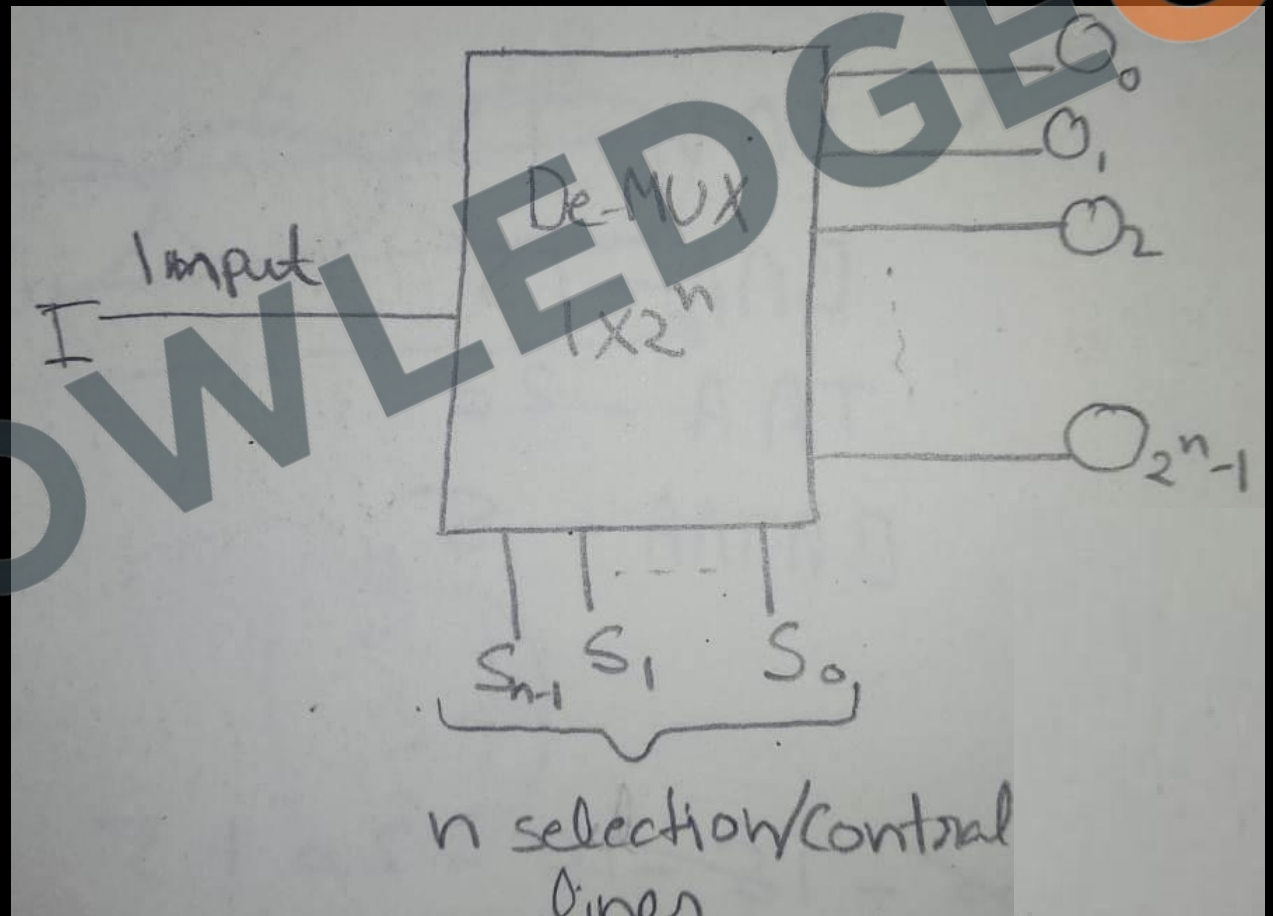
Q Implement the function F(a,b,c) = a’b’c + a’bc’ + ab’c’ + abc. With one, two and three variable on the select line?

Demultiplexer

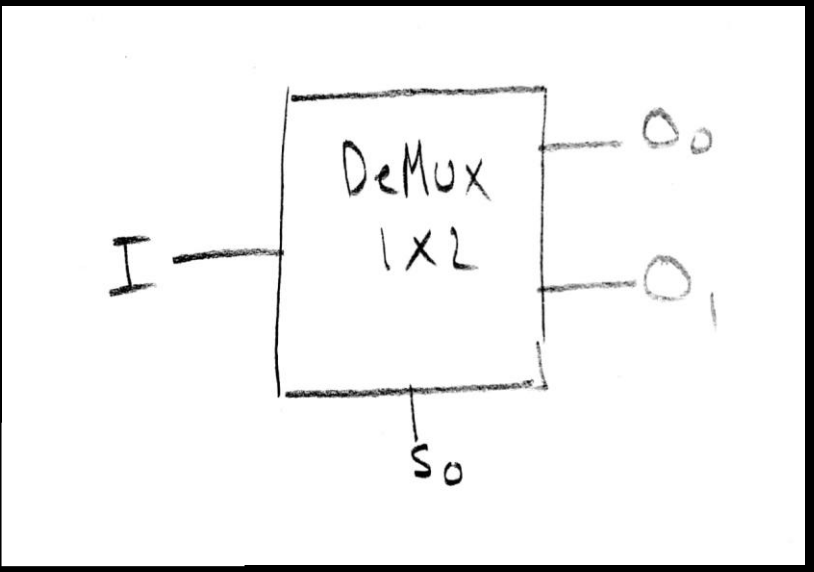
1. A demultiplexer (or DeMux) is a device that takes a single input line and routes it to one of several digital output lines.
2. A demultiplexer is also called a data distributor.
3. It is conceptually same as Mux just with reverse logic.



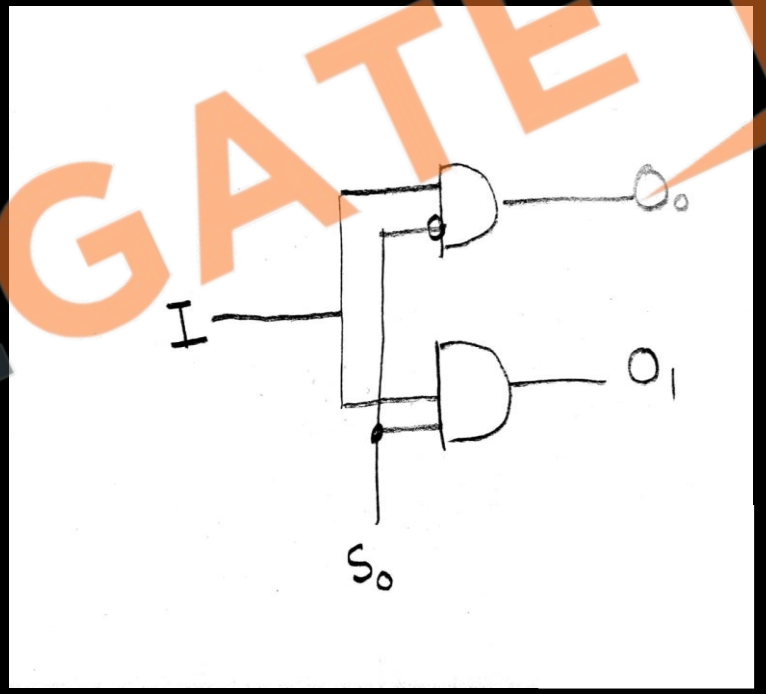
- A demultiplexer of 2^n outputs has n select lines, which are used to select which output line to send the input.
- A DeMux is a combinational circuit, which is used in data communication
- Serial to parallel conversation
- Demultiplexers are mainly used in Boolean function generators and decoder circuits.



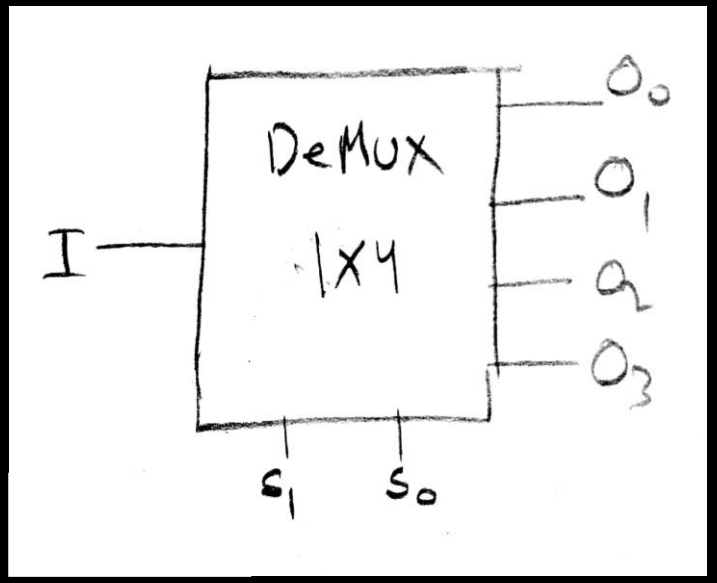
1 to 2 Demultiplexer



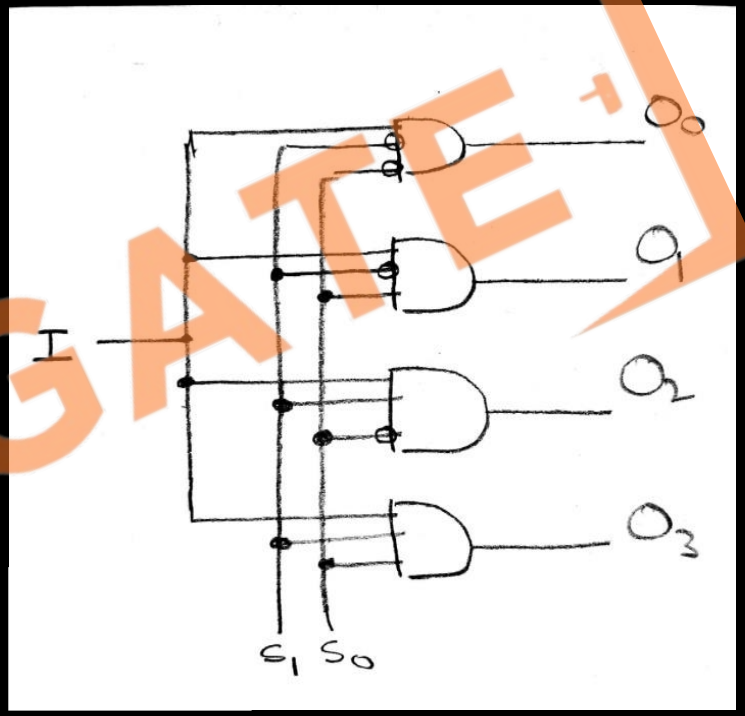
S_0	O_1	O_0
0	0	I
1	I	0



1 to 4 Demultiplexer



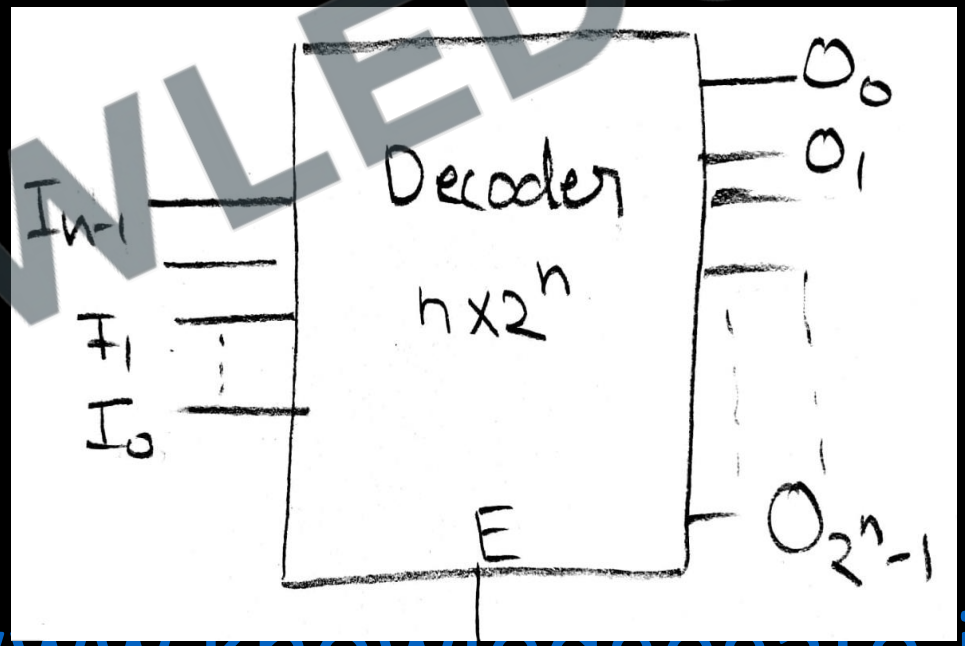
S_1	S_0	O_3	O_2	O_1	O_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0



www.knowledgegate.in

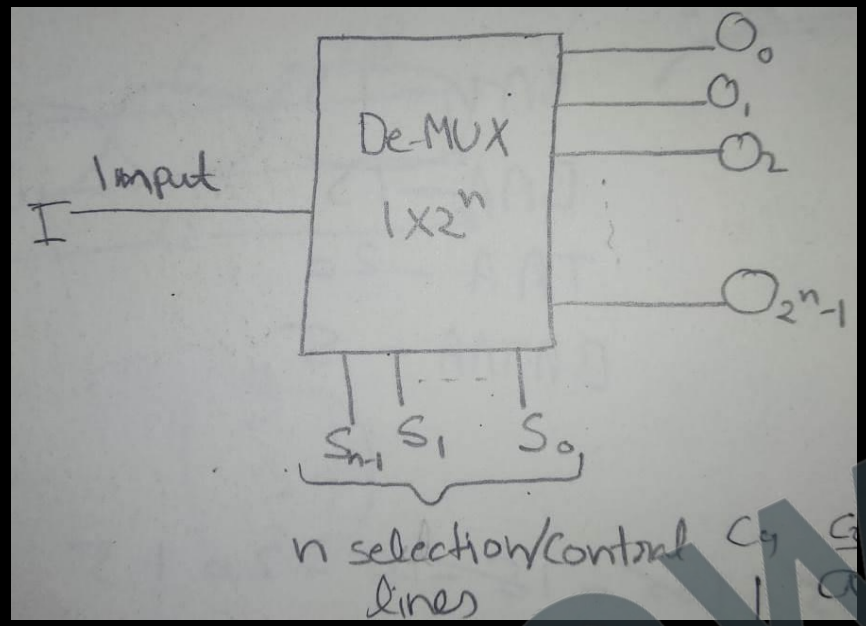
Decoder

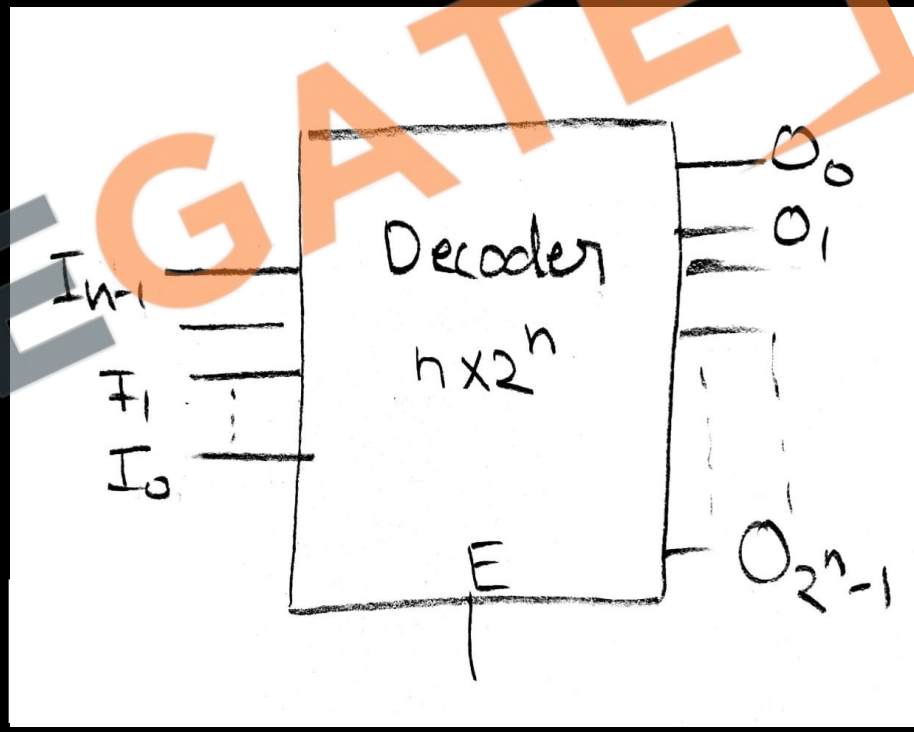
- A decoder is a combinational circuit that decodes binary information from n input lines to a maximum of 2^n unique output lines.
- The decoders are called n -to- m -line decoders, where $m \leq 2^n$.
- Their purpose is to generate the 2^n (or fewer) minterms of n input variables. Each combination of inputs will assert a unique output.
- If the n -bit coded information has unused combinations, the decoder may have fewer than 2^n outputs.

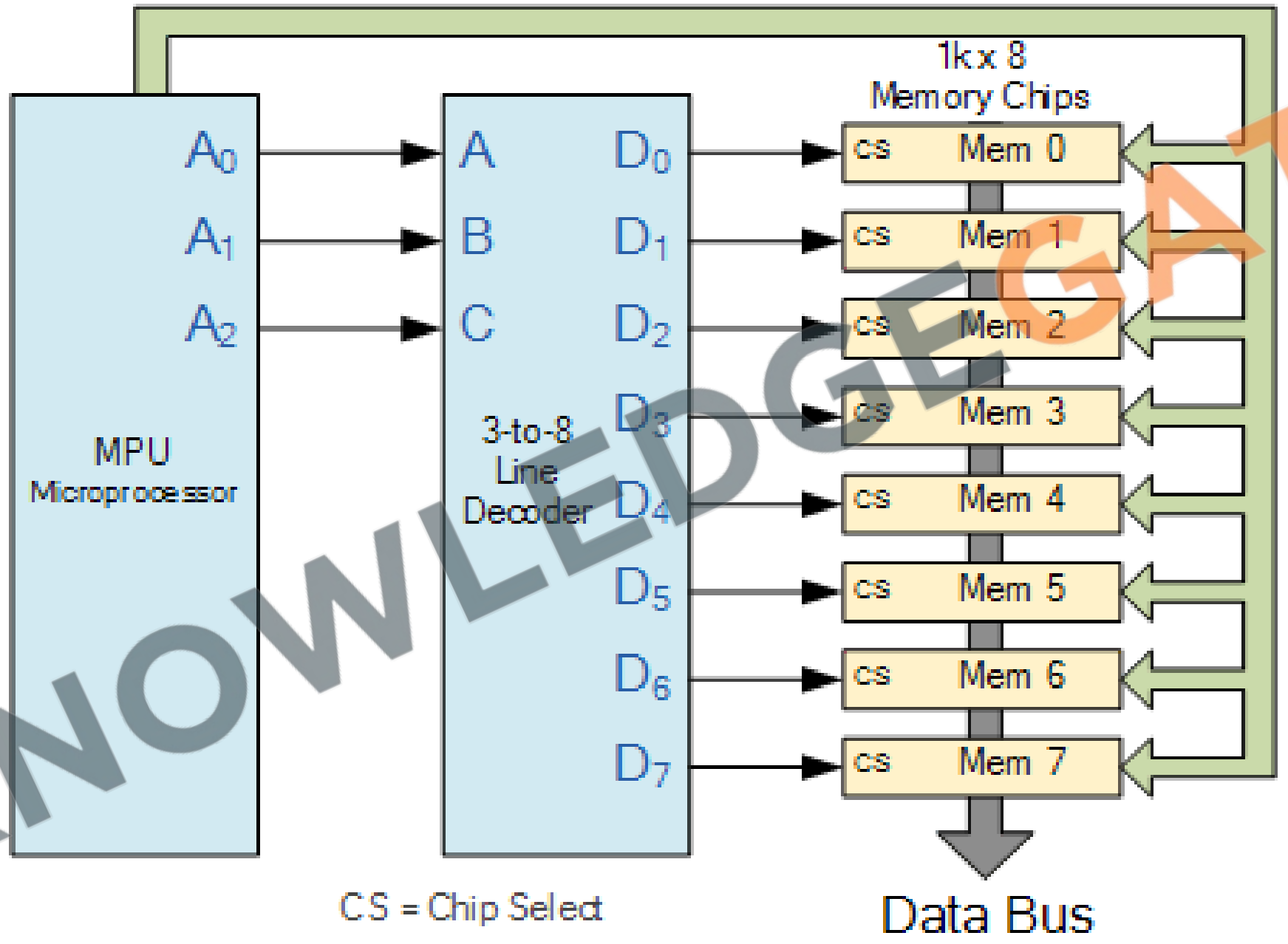


Decoder

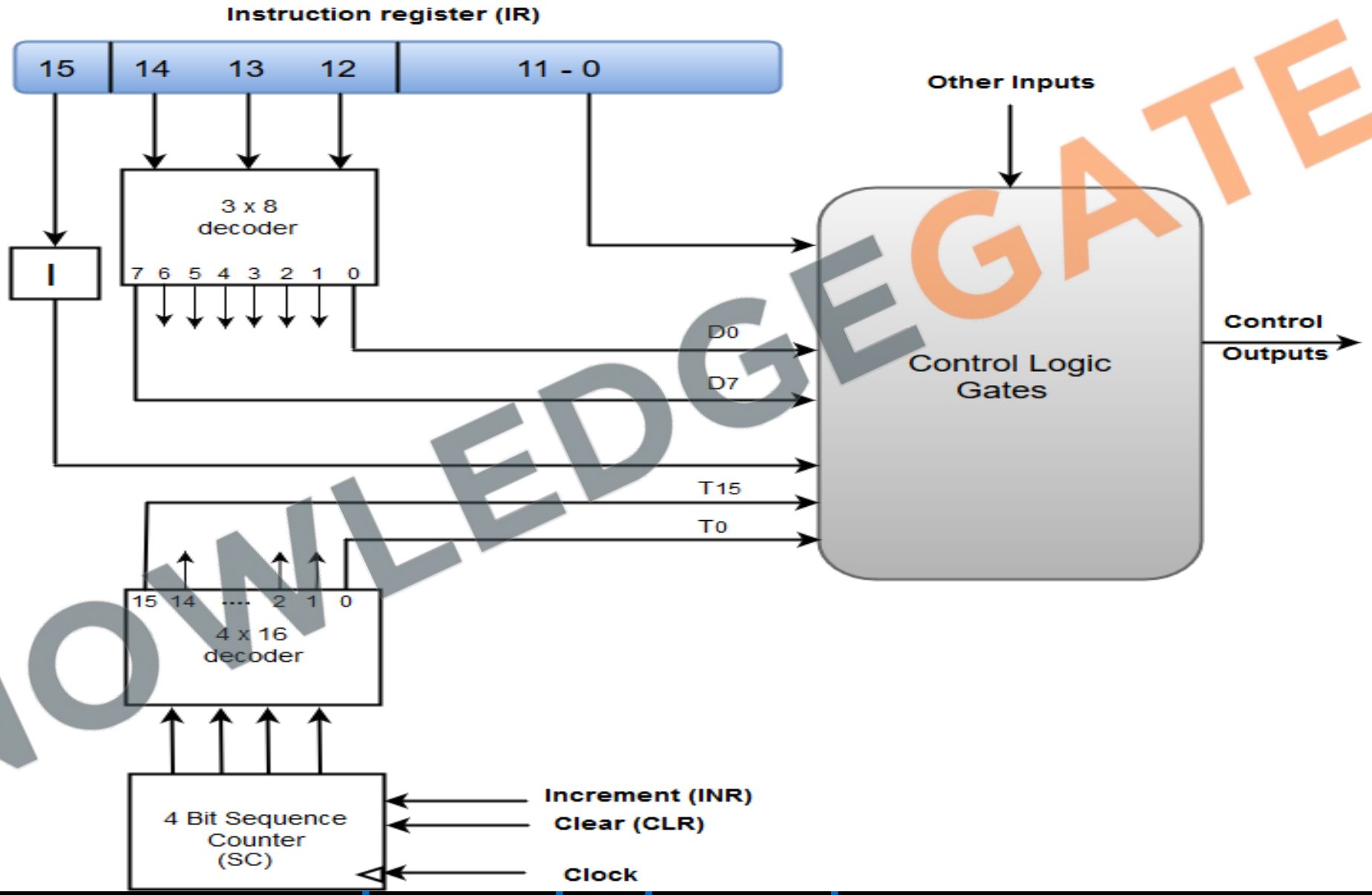
- Decoders are also combinational circuits logically we can say a DeMux can be converted into a decoder by setting input line as enable line and selection line as input lines.

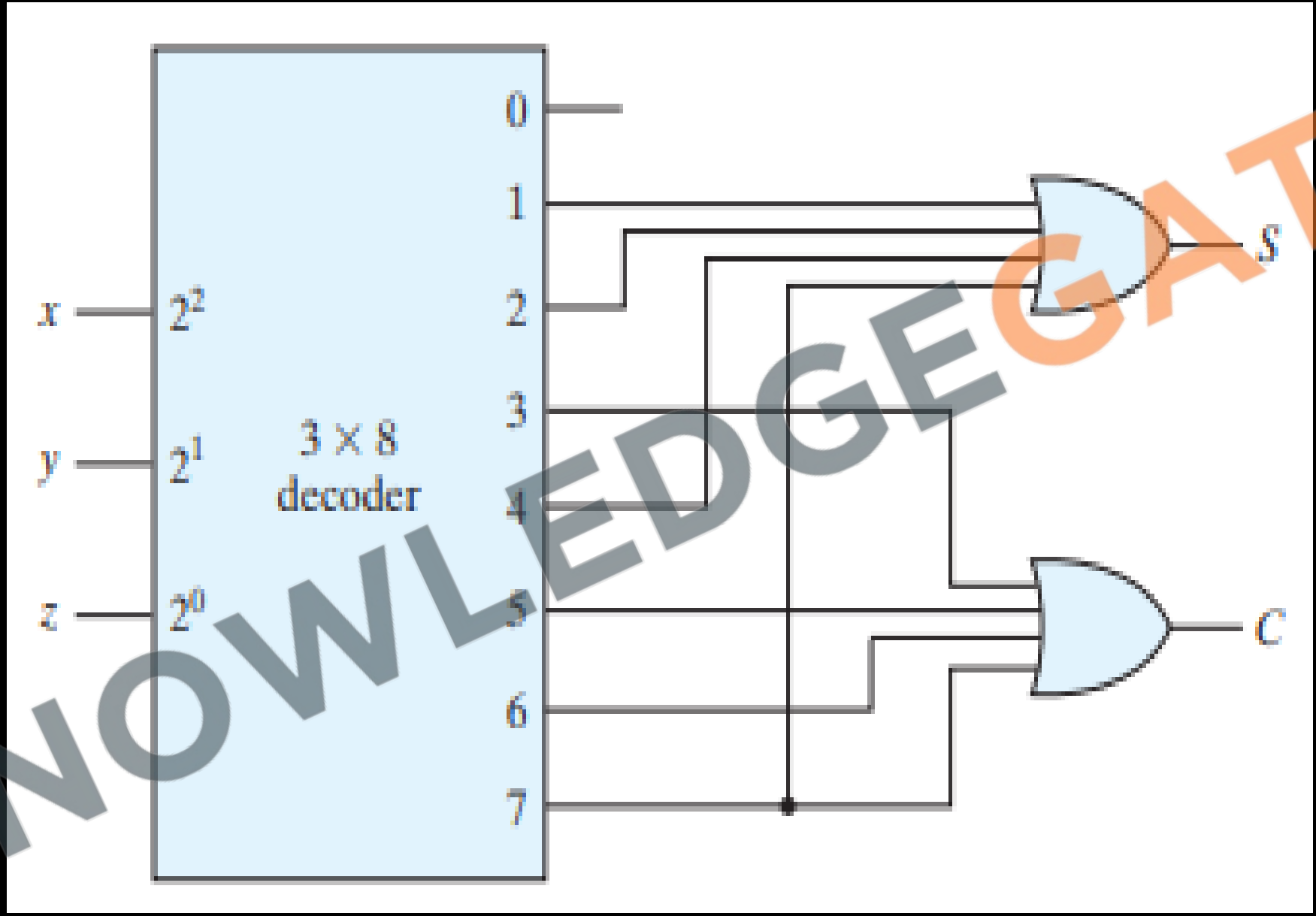




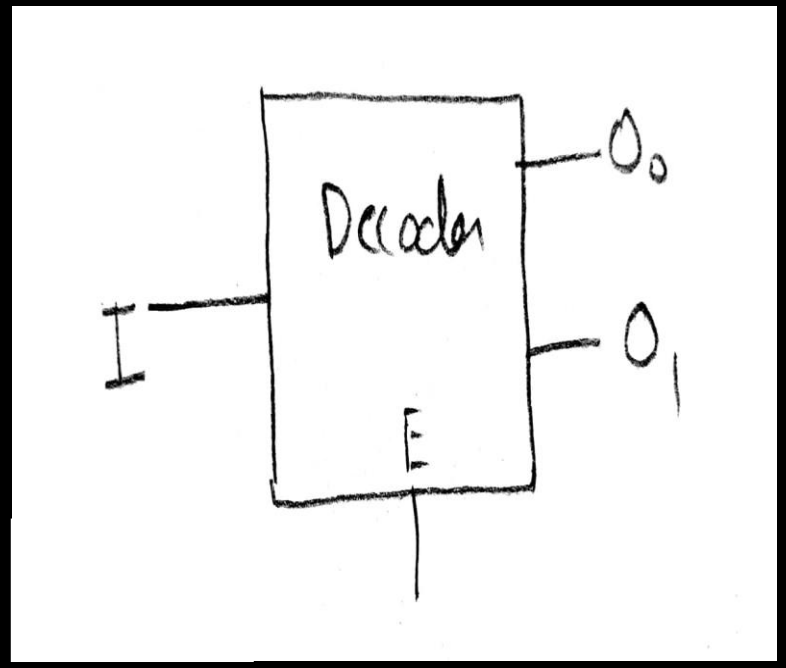


Control Unit of a Basic Computer:

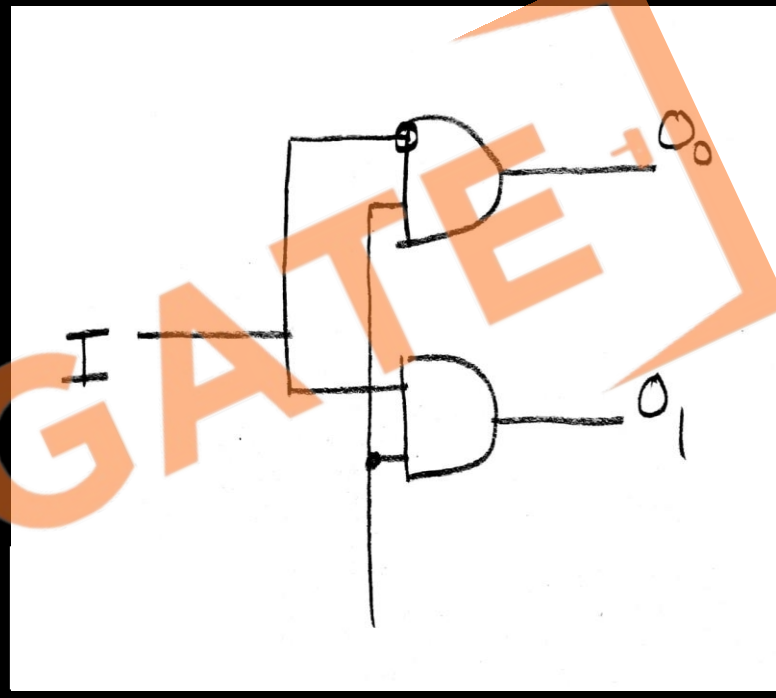




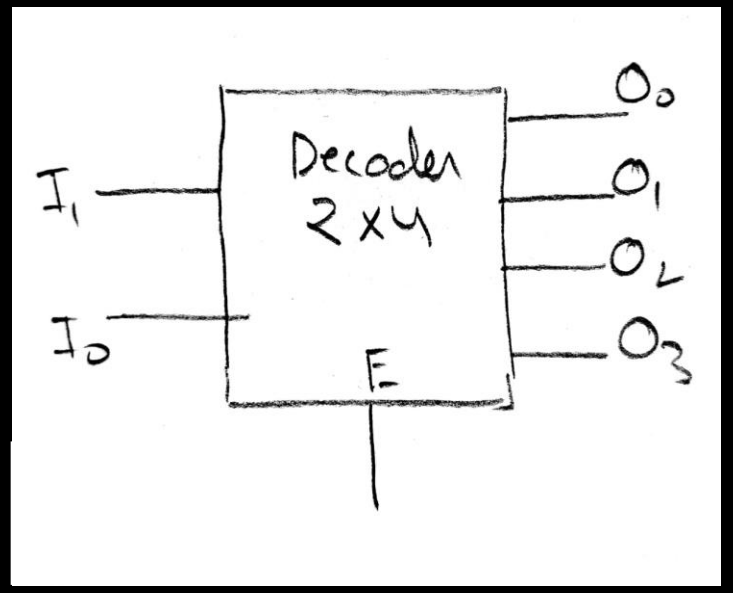
1-to-2 Decoder



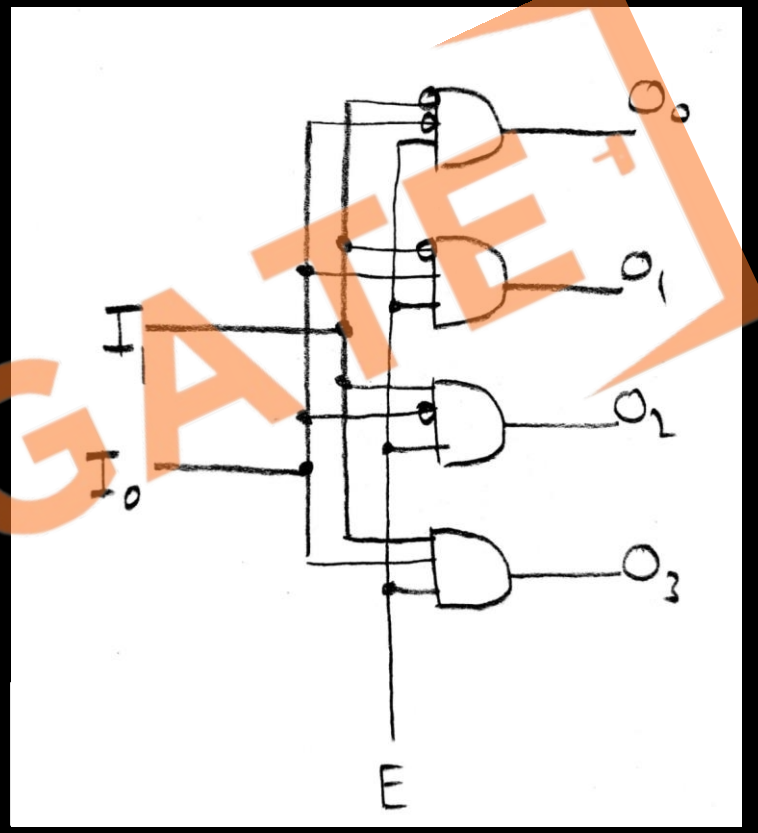
I	O_1	O_0
0	0	1
1	1	0



2-to-4 Decoder



Input		Output			
I_1	I_0	O_3	O_2	O_1	O_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

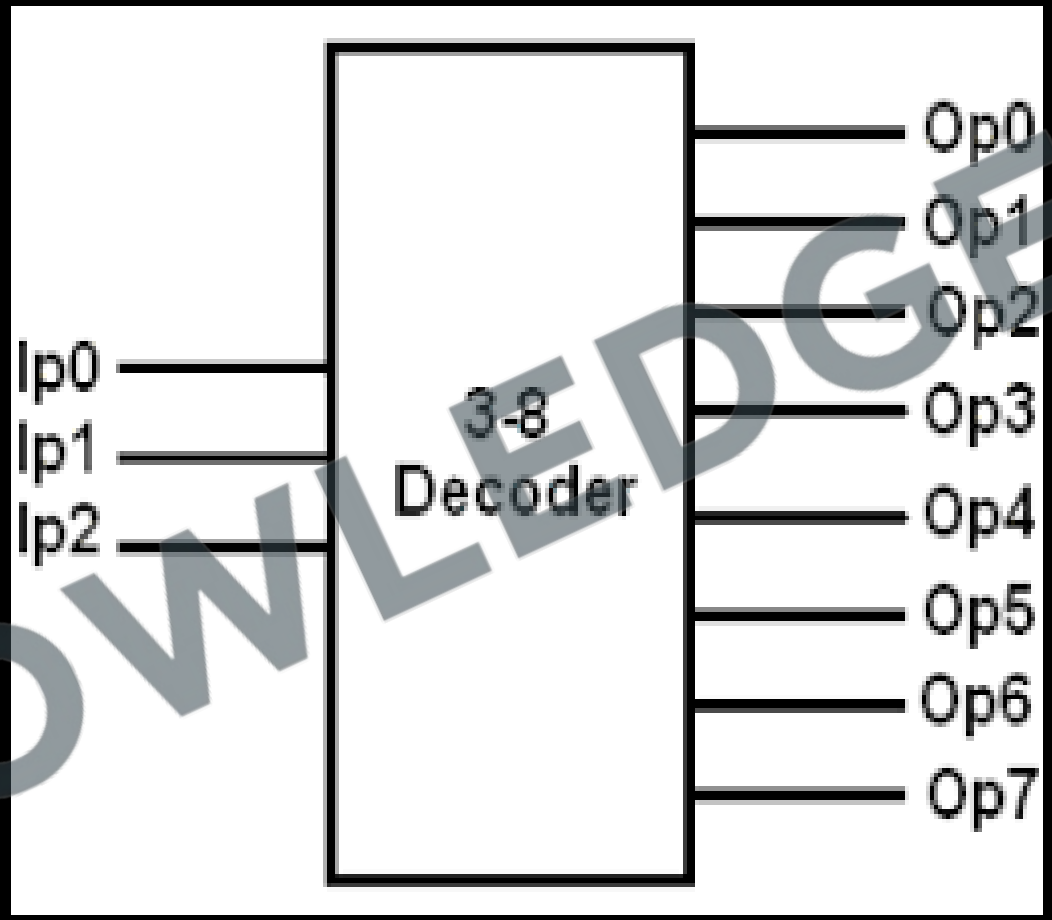


Q Implementation of a full adder with a decoder

From the truth table of the full adder, we obtain the functions for the combinational circuit in sum-of-minterms form:

S(x, y, z) = ∑ (1, 2, 4, 7)

C(x, y, z) = ∑ (3, 5, 6, 7)

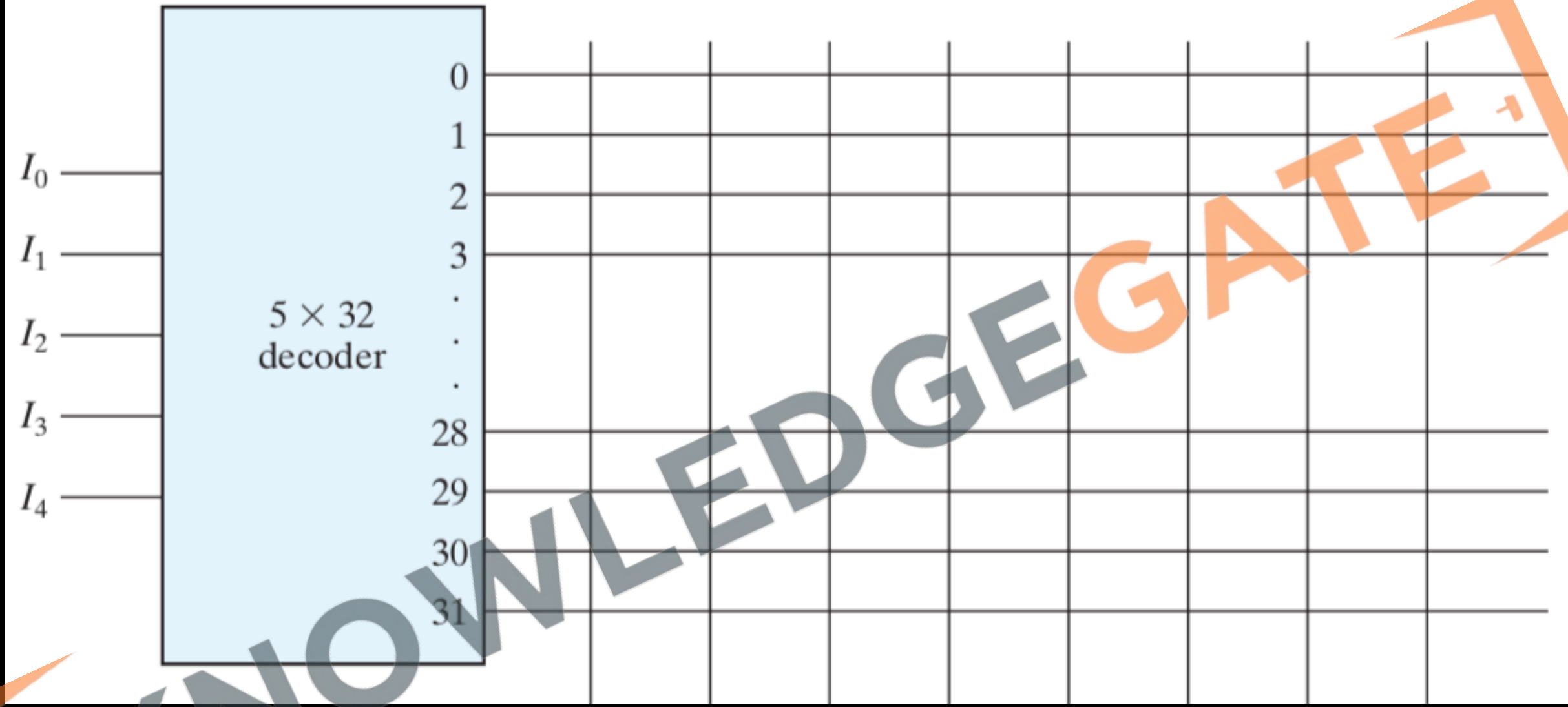


Combinational Logic Implementation

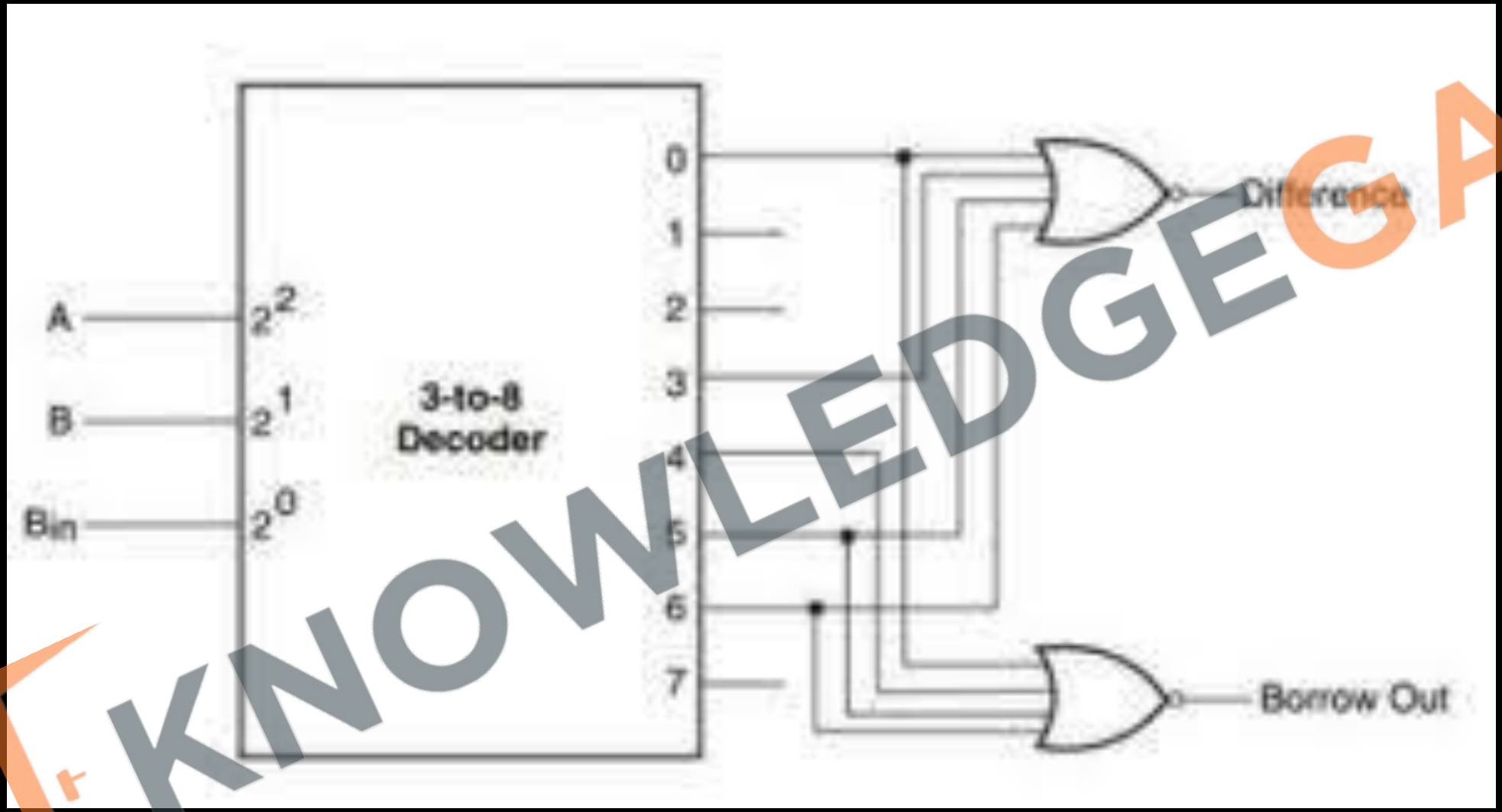
- Any combinational circuit with n inputs and m outputs can be implemented with an n -to-2^n -line decoder and OR gates.

- The procedure for implementing a combinational circuit by means of a decoder and OR gates requires that the Boolean function for the circuit be expressed as a sum of minterms.

- A decoder is then chosen that generates all the minterms of the input variables. The inputs to each OR gate are selected from the decoder outputs according to the list of minterms of each function.

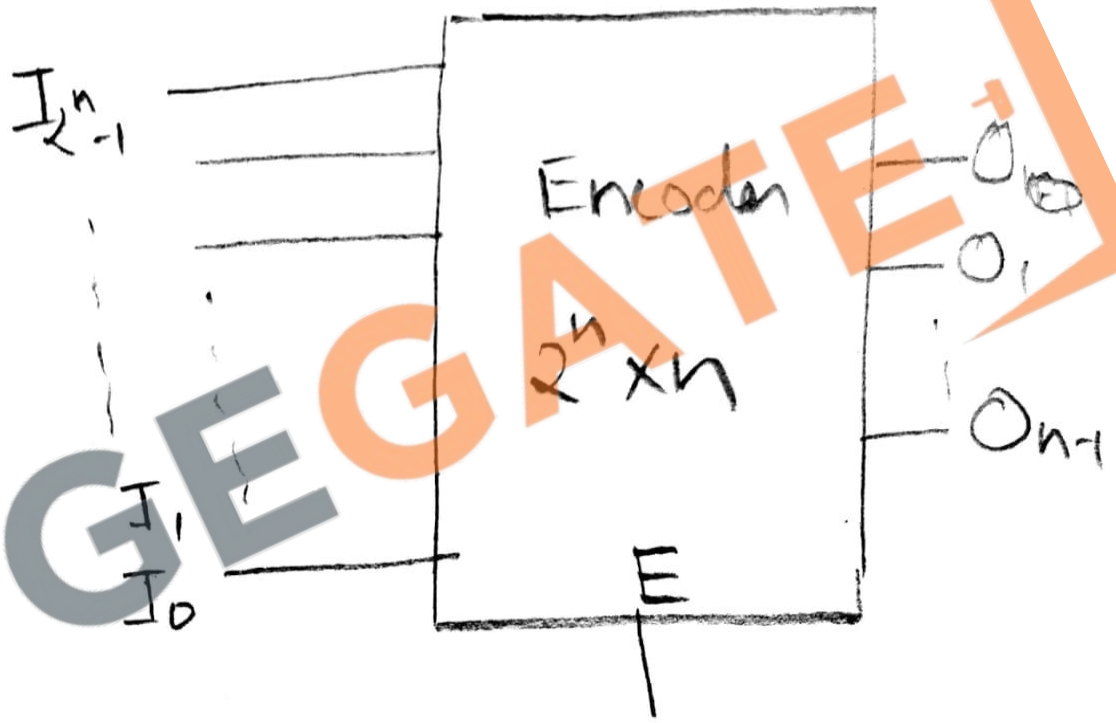


Q A full Subtractor using Decoder?

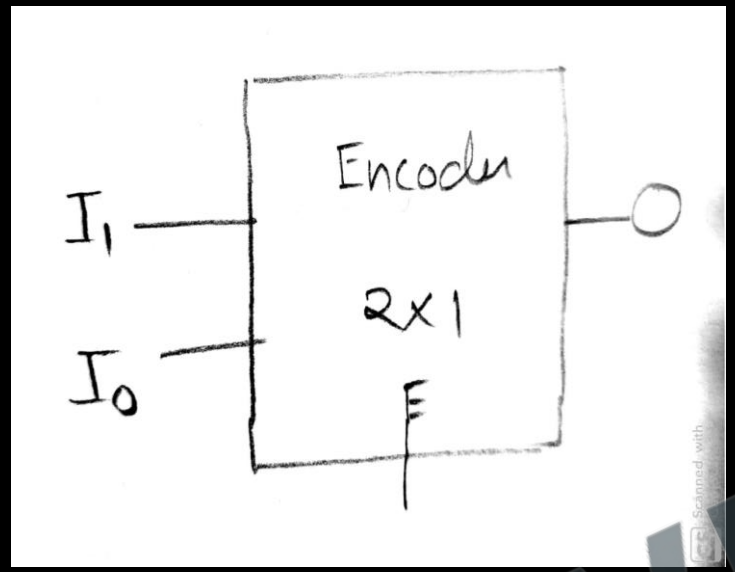


Encoder

- An encoder is a combinational circuit that encode binary information form one of a 2^N input lines and encode it into N output lines, which represent N bit code for the input.
- For simple encoders, it is assumed that only one input line is active at a time.
- Encoder performs the inverse operation of a decoder.

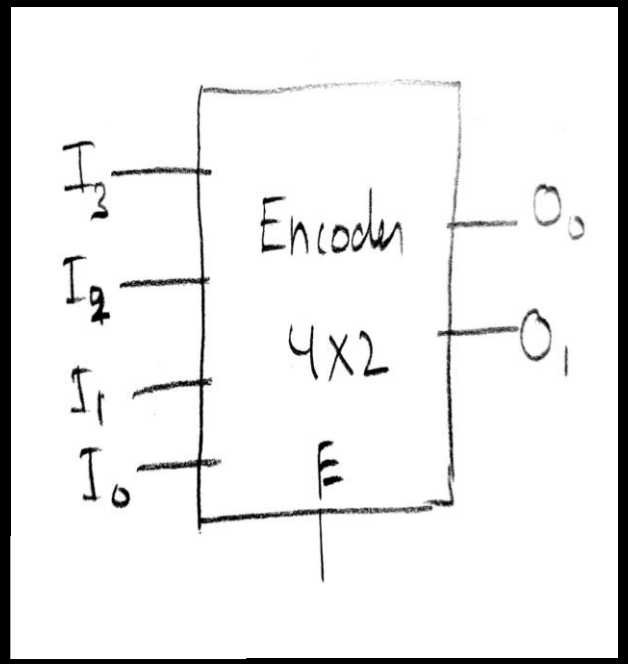


2-to-1 Encoder



I_1	I_0	O_0
0	1	0
1	0	1

4-to-2 Encoder



I_3	I_2	I_1	I_0	O_1	O_0
0	0	0	1	**0**	**0**
0	0	1	0	**0**	**1**
0	1	0	0	**1**	**0**
1	0	0	0	**1**	**1**

- $O_0 = I_0' I_1' (I_2 \oplus I_3)$
- $O_1 = I_0' I_2' (I_1 \oplus I_3)$

Priority Encoder

- In some practical cases more than one input can be high at a time, there we can not use simple encoder. In a priority encoder more than one input can be high at a time. A priority encoder is an encoder circuit that includes the priority function.
- The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

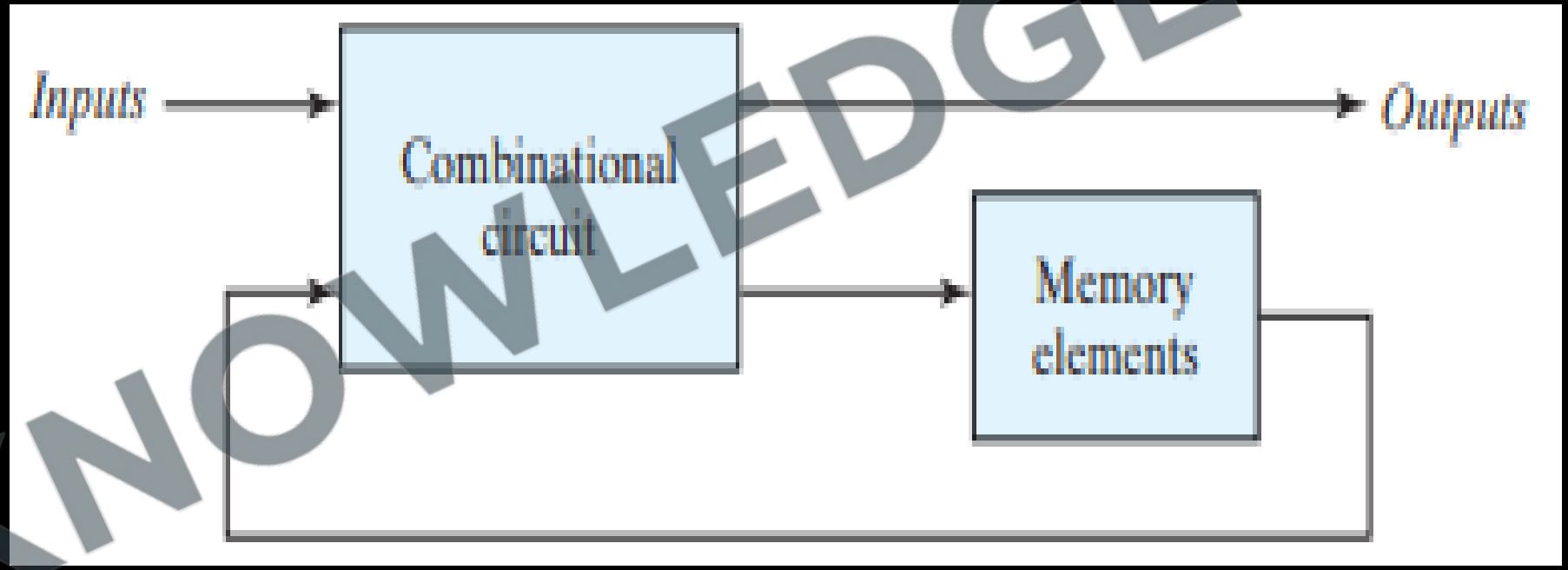
- They are often used to control interrupt requests by acting on the highest priority interrupt input. The *priority encoders* output corresponds to the currently active input which has the highest priority. So when an input with a higher priority is present, all other inputs with a lower priority will be ignored.

- $I_3 > I_2 > I_1 > I_0$
- $O_0 = I_3 + I_3' I_2' I_1$
- $O_1 = I_2 + I_3$

I_3	I_2	I_1	I_0	O_1	O_0
0	0	0	1	**0**	**0**
0	0	1	d	**0**	**1**
0	1	d	d	**1**	**0**
1	d	d	d	**1**	**1**

SEQUENTIAL CIRCUITS

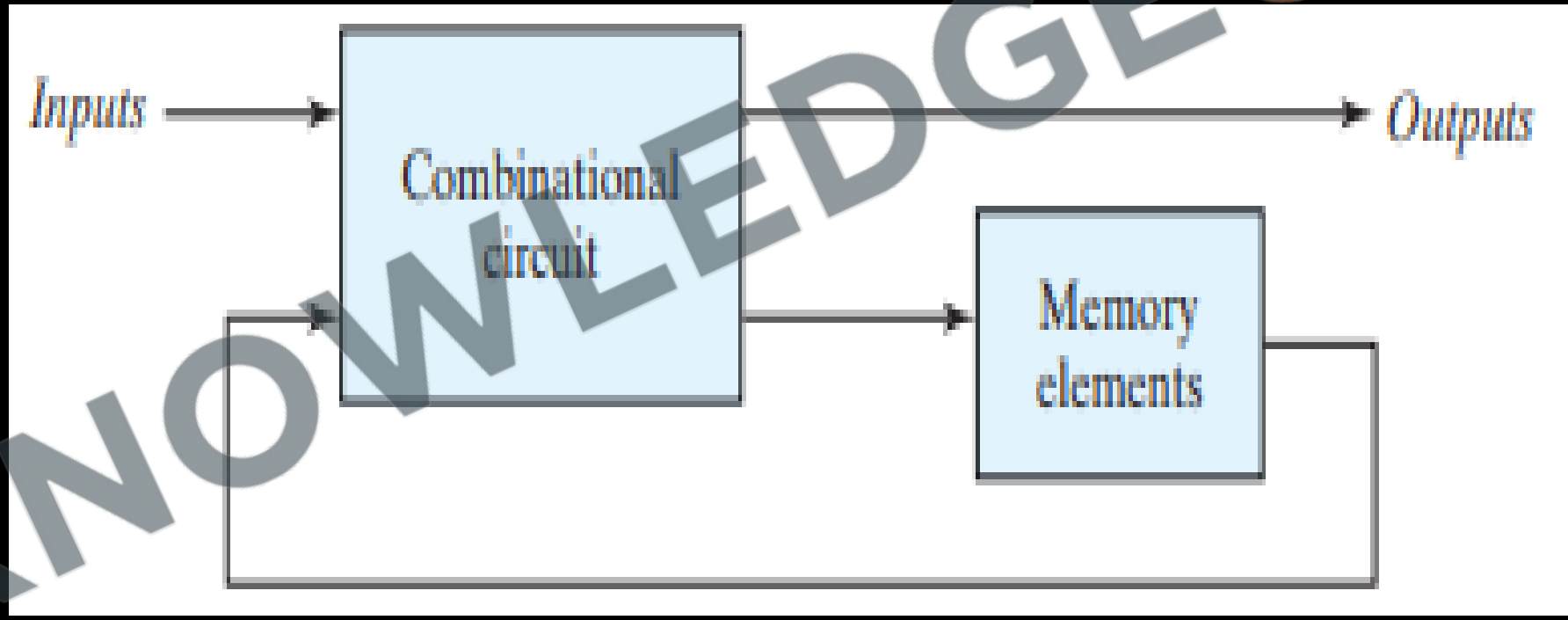
- Sequential Circuits consists of a combinational circuit to which memory elements are connected to form a feedback path. The memory elements are devices capable of storing binary information.
- The binary information stored in these elements at any given time defines the state of the sequential circuit at that time.



- The sequential circuit receives binary information from external inputs (x_n) that, together with the present state (y_{n-1}) of the memory elements, determine the binary value of the outputs(y_n).

- A sequential circuit is specified by a time sequence of inputs, outputs, and internal states.

$$f(x_n, y_{n-1}) = y_n$$



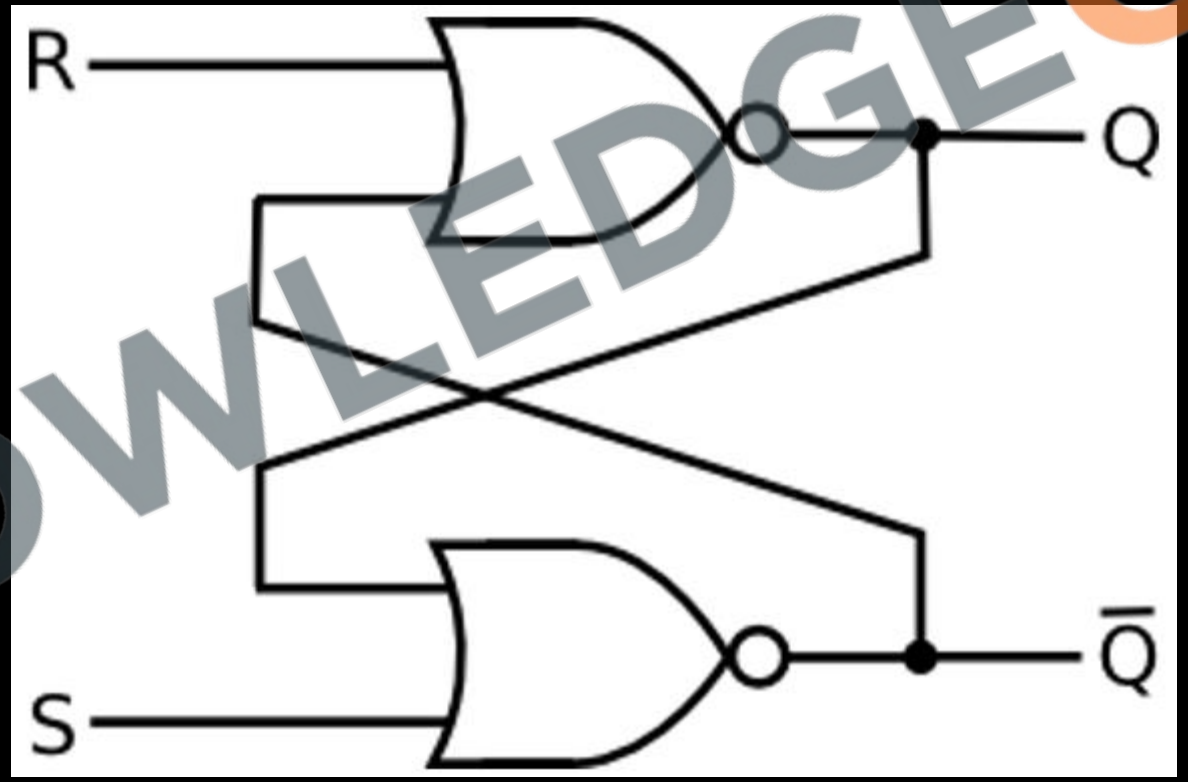
Latches

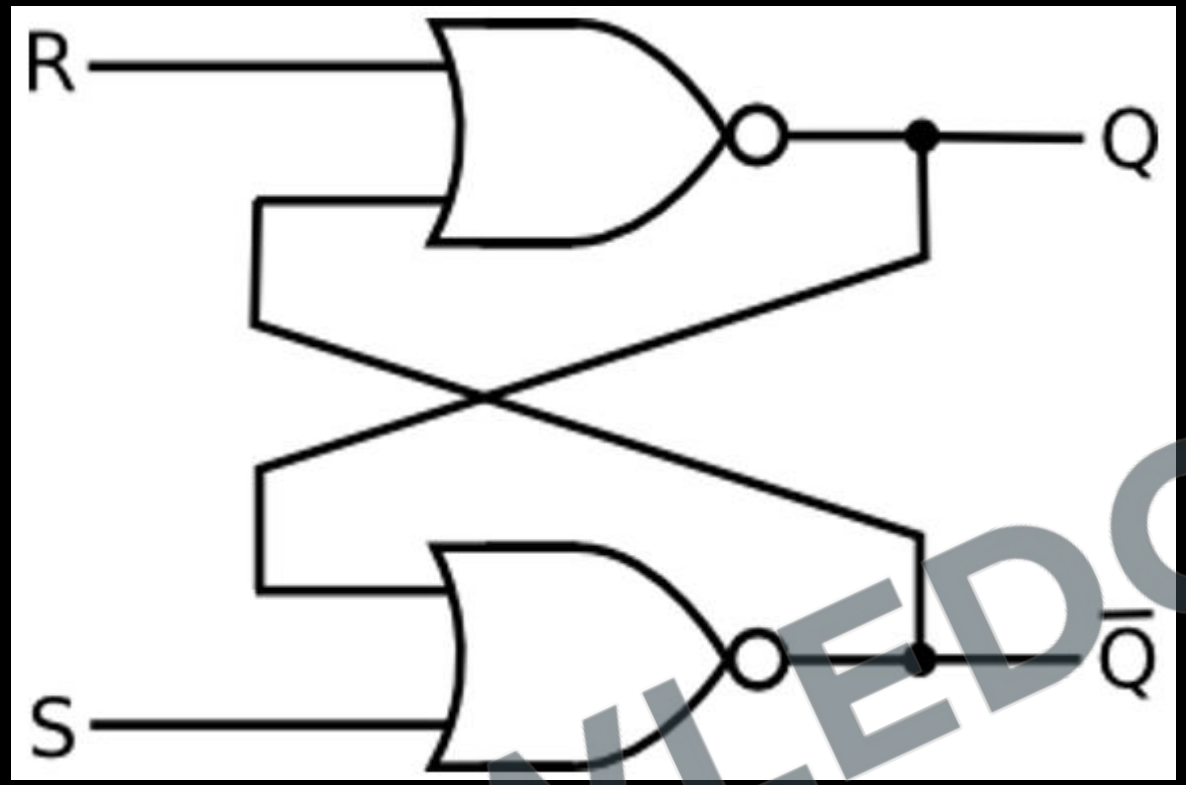
- Latch means to hold something or something which do not change.
- latches are the basic building blocks of any flip flop and they are capable of holding 1 bit until necessary.
- Storage elements that operate with signal levels are referred to as latches.
- Latches are level sensitive devices.

NOR Latch

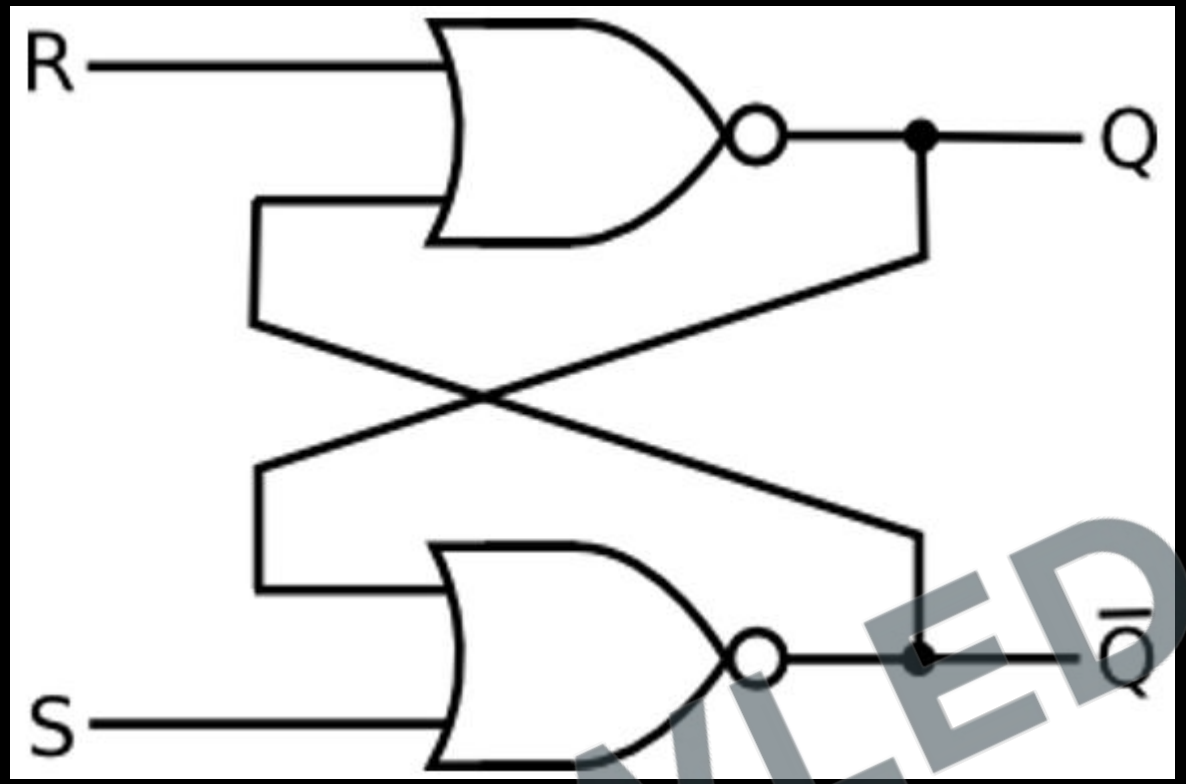
- The SR latch is a circuit with two cross-coupled NOR gates or two cross-coupled NAND gates, and two inputs labeled S for set and R for reset.

- Outputs Q_n and Q_n' are the complement of each other, in valid scenario.



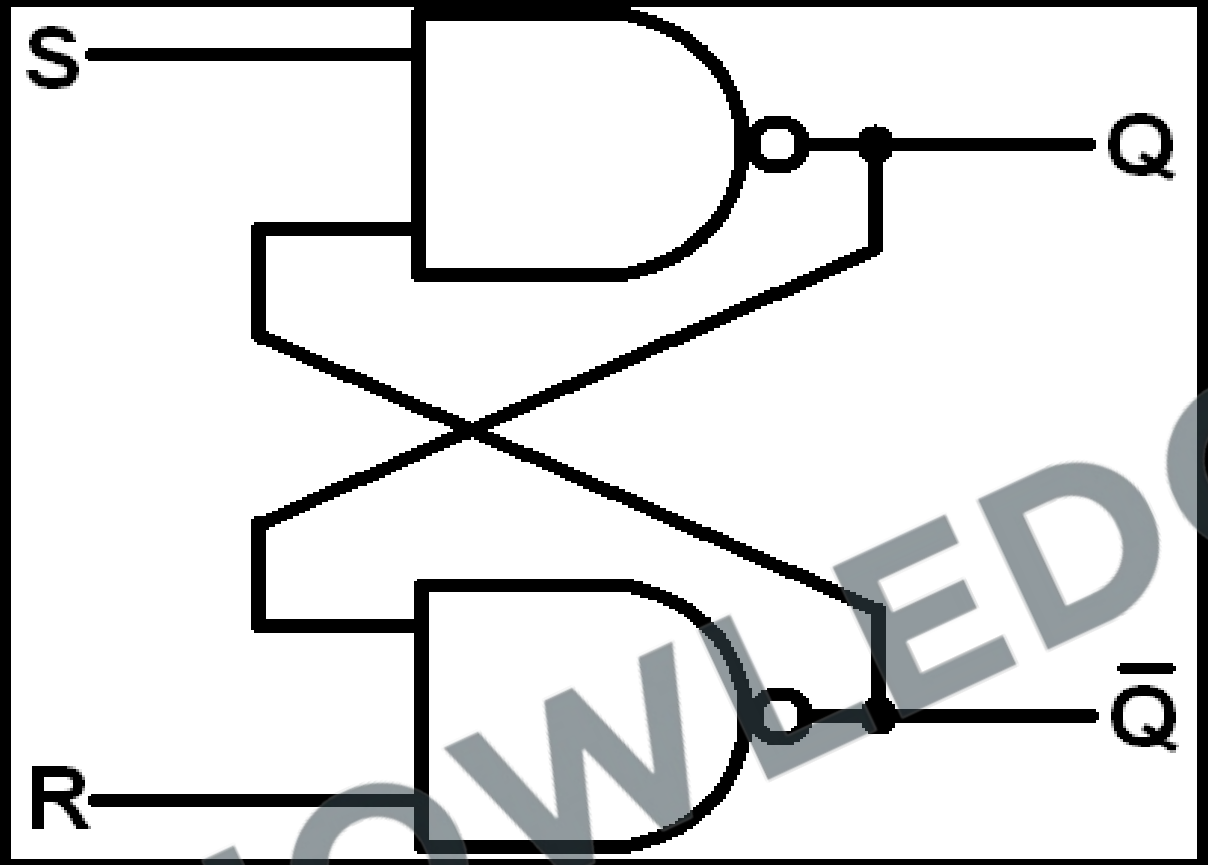


S	R	Q_n	Q_{n+1}
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

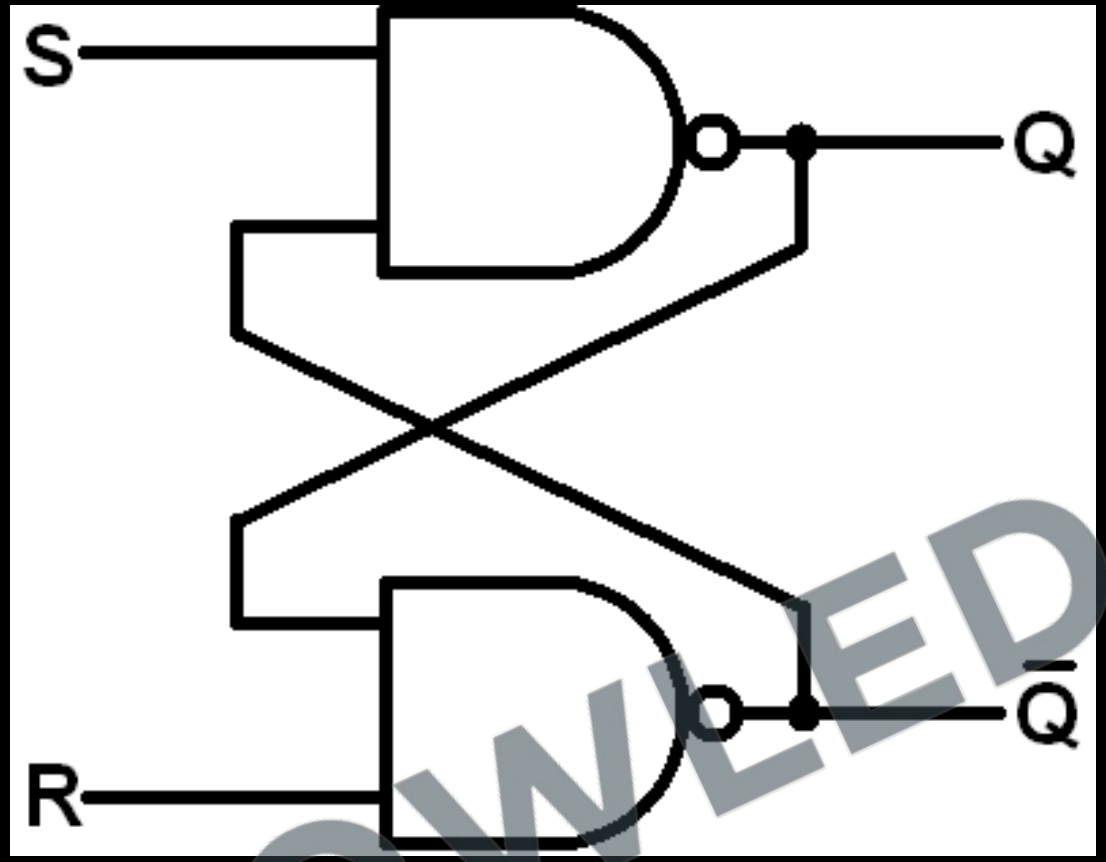


S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

NAND Latch



S	R	Q_n	Q_{n+1}
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



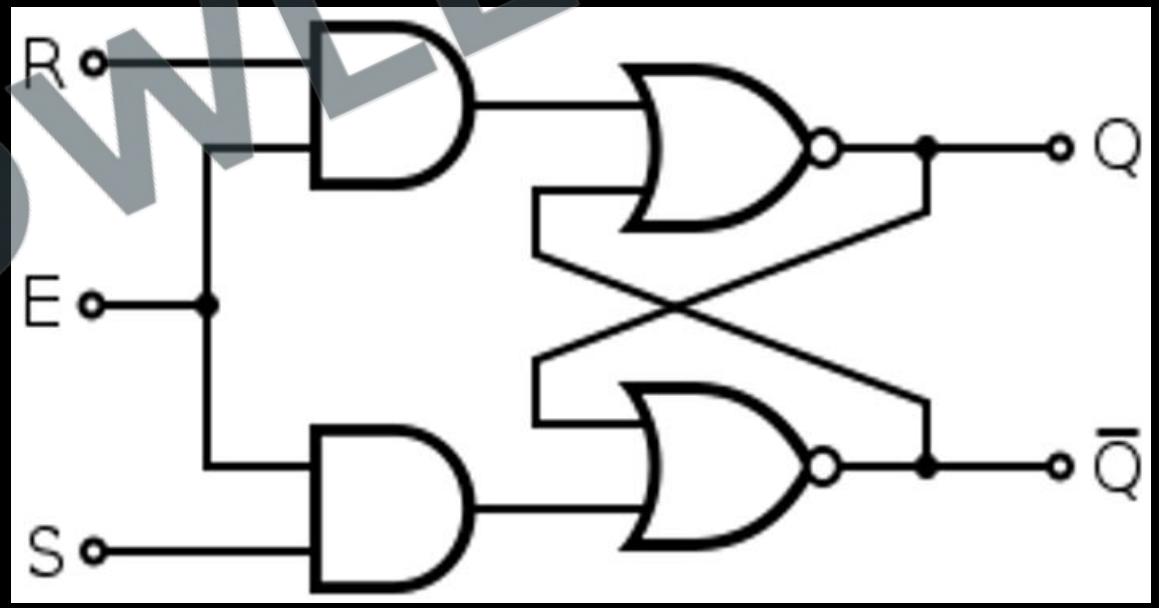
S	R	Q_n	Q_{n+1}
0	0	0	X
0	0	1	X
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Notes

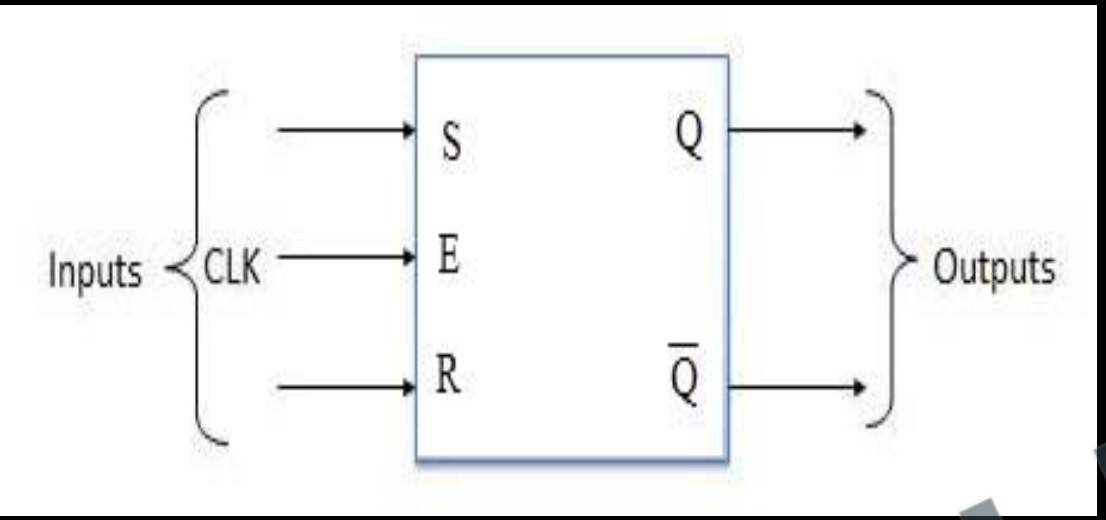
- Latches are the basic circuits from which all flip-flops are constructed.
- The storage elements (memory) used in clocked sequential circuits are called flipflops.
- A flip-flop is a binary storage device capable of storing one bit of information. In a stable state, the output of a flip-flop is either 0 or 1 (it is called bi-stable multi-vibrator).
- A flip-flop is said to be stable if it has complementary behavior.
- Latches are level sensitive devices; flip-flops are edge-sensitive devices.
- Storage elements that operate with signal levels are referred to as latches; those controlled by a clock transition are flip-flops.

SR flip flop

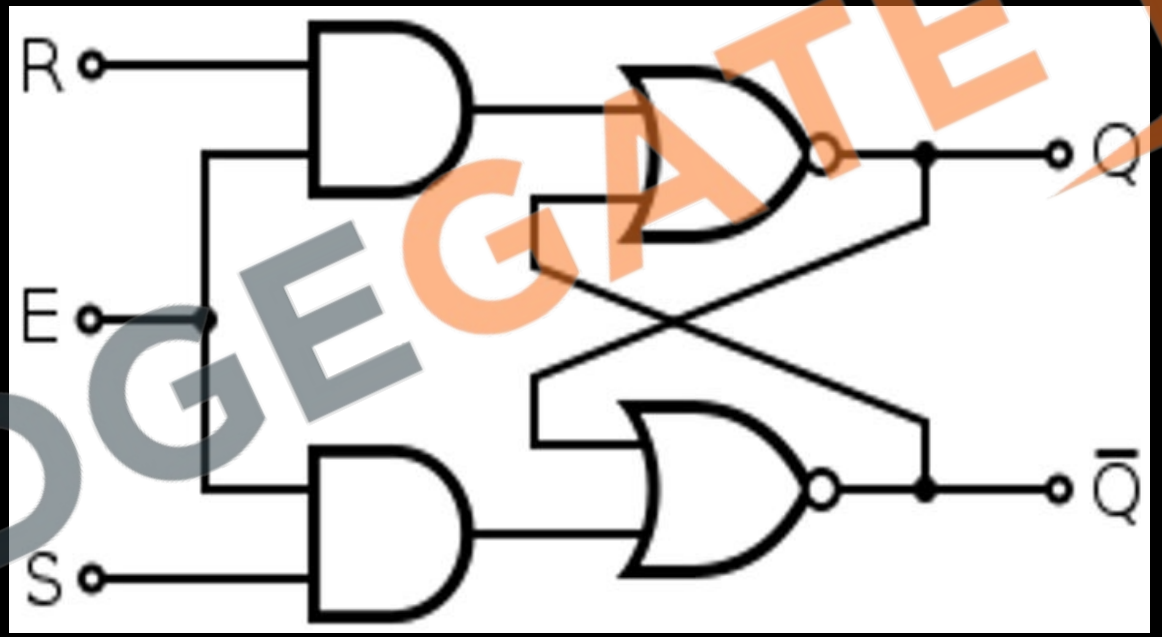
1. The operation of the basic SR latch can be modified by providing an additional input signal that determines (controls), when the state of the latch can be changed by determining whether S and R (or S and R) can affect the circuit.
2. It consists of the basic SR latch and two additional AND gates. The control input CP / E_n acts as an enable signal for the other two inputs.
3. The outputs of the AND gates stay at the 0 as long as the enable signal remains at 0 as one input of AND gate gets 0 resulting in 0 as output. When the enable input goes to 1, information from the S or R input is allowed to affect the latch.



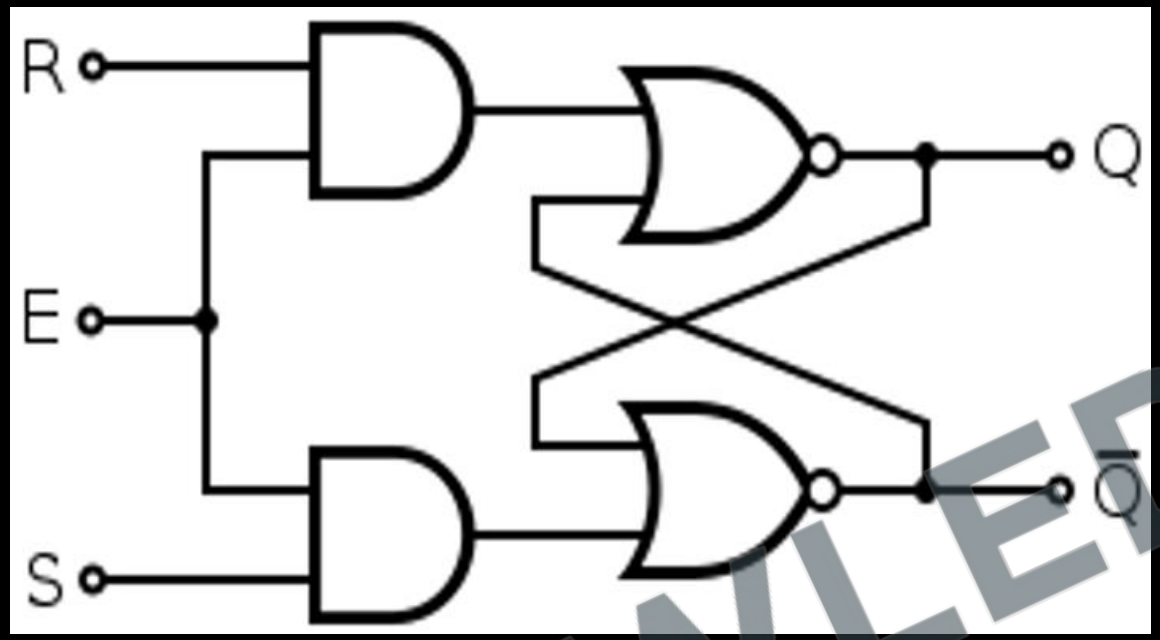
Block Diagram



Implementation



Implementation



Truth Table

S	R	Q_n	Q_{n+1}
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Truth Table

S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

K-Map

	sr	s'r'	s'r	sr	sr'
q		00	01	11	10
q'	00				
q	01				

Characteristics Equation

Q_{n+1} =

Truth Table

S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

Characteristics Equation

$$Q_{n+1} = S + R' Q_n$$

Function Table

S	R	Q_{n+1}
0	0	
0	1	
1	0	
1	1	

Truth Table

S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

Excitation Table

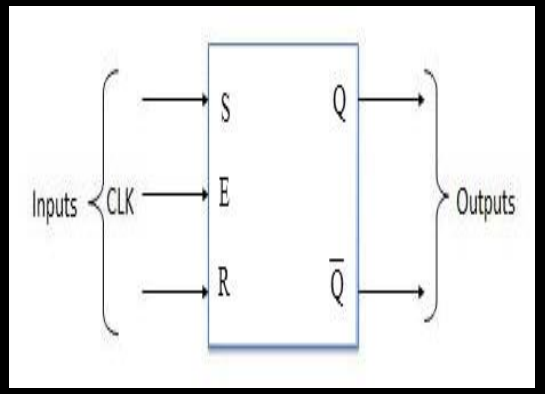
Q_n	Q_{n+1}	S	R
0	0		
0	1		
1	0		
1	1		

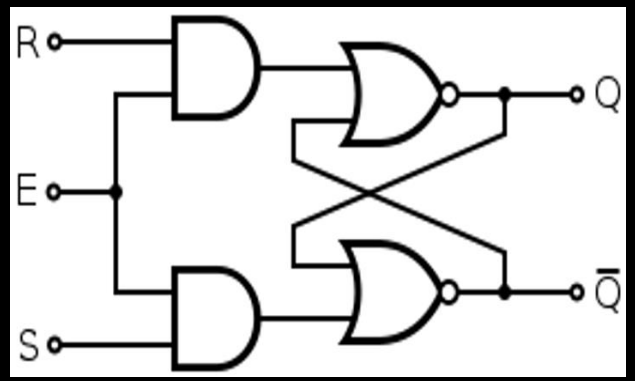
Truth Table

S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

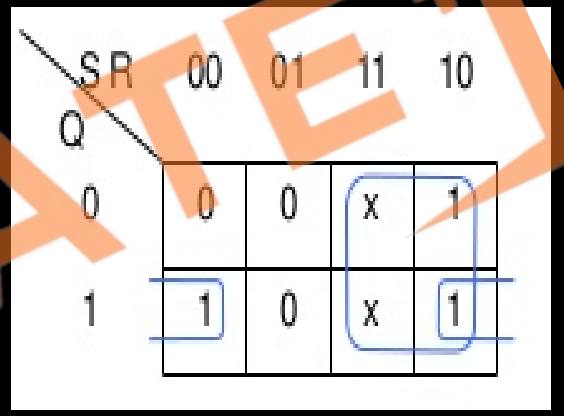
State Diagram

Block Diagram	Implementation	Truth Table	K-Map





S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

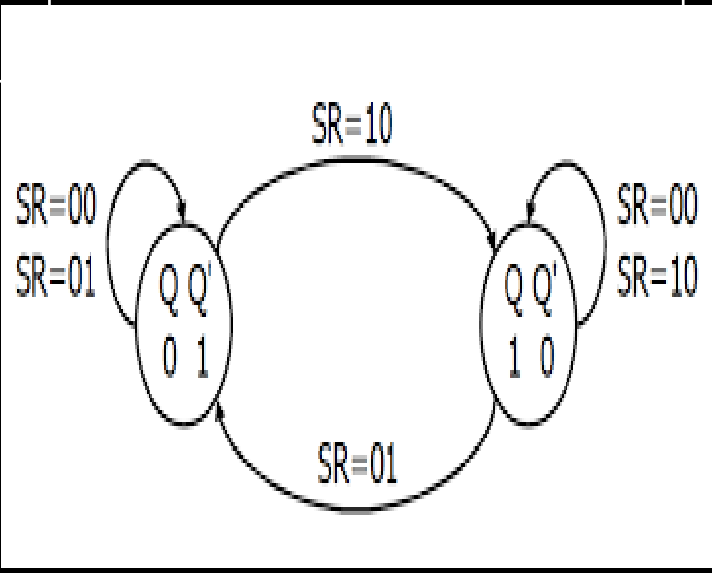


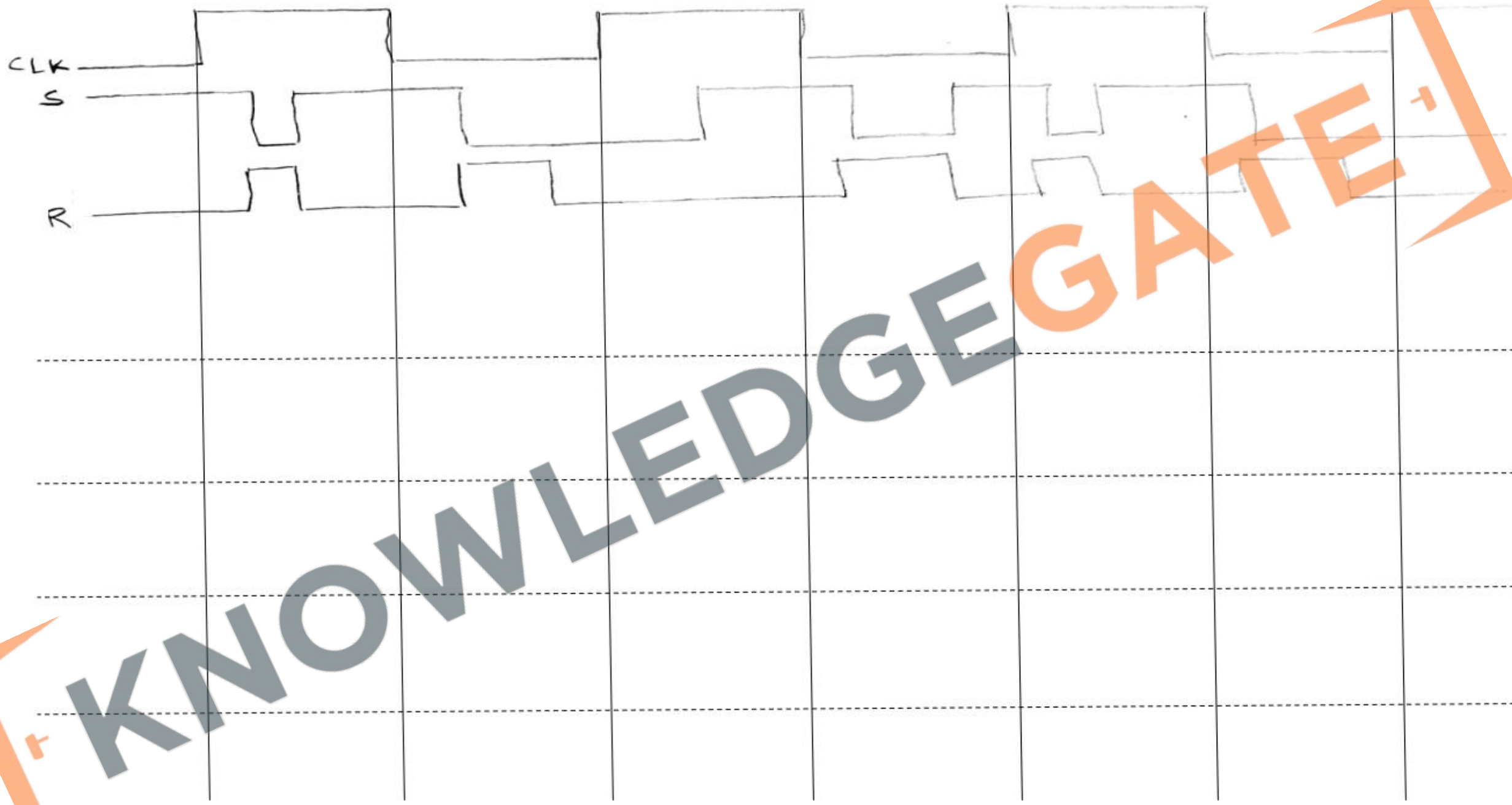
Characteristics Equation	Function Table	Excitation Table

$$Q_{n+1} = S + R' Q_n$$

S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	X

Q_n	Q_{n+1}	S	R
0	0	0	d
0	1	1	0
1	0	0	1
1	1	d	0





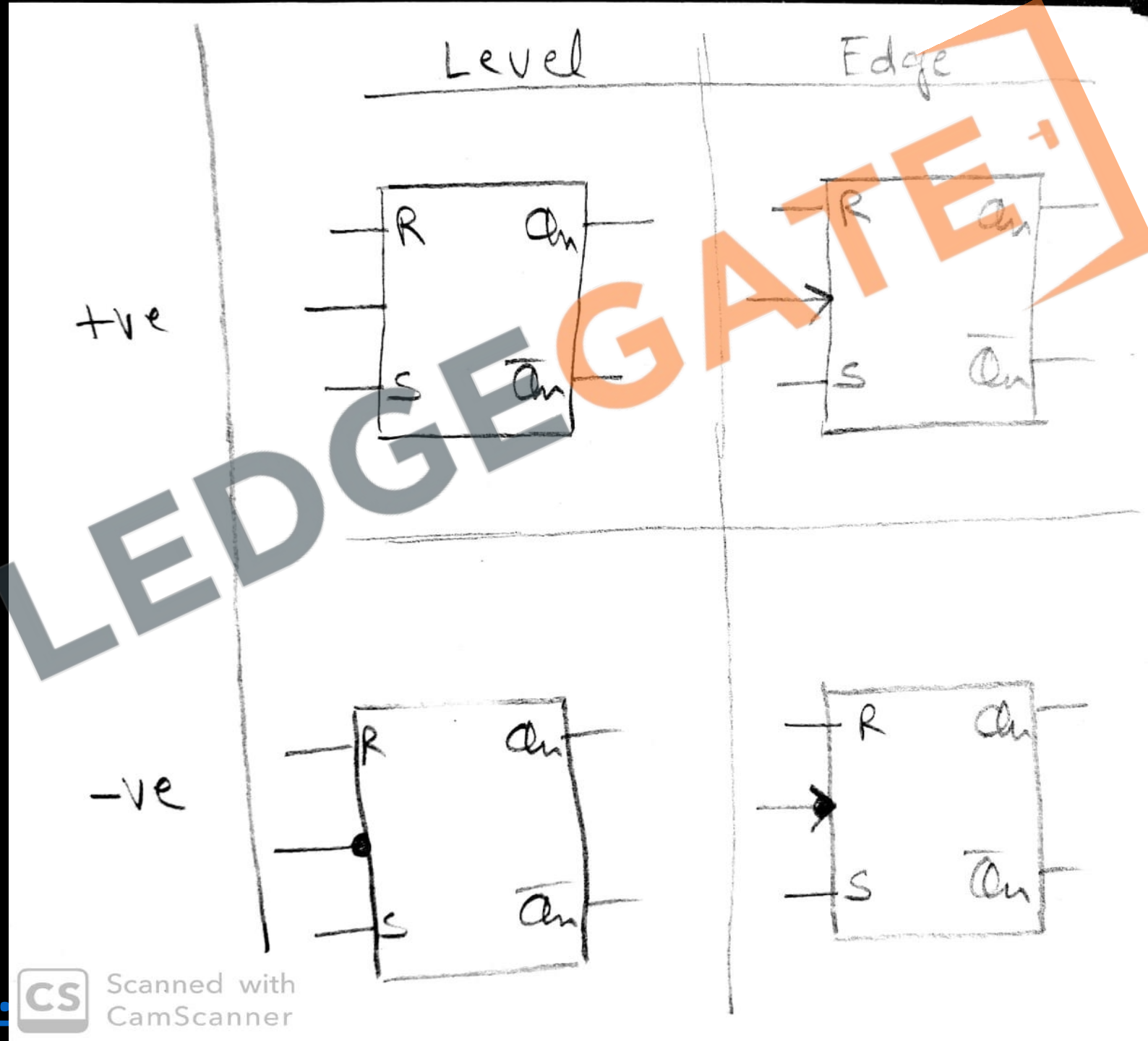
Order of Priority

-ve edge

+ve edge

-ve level

+ve level

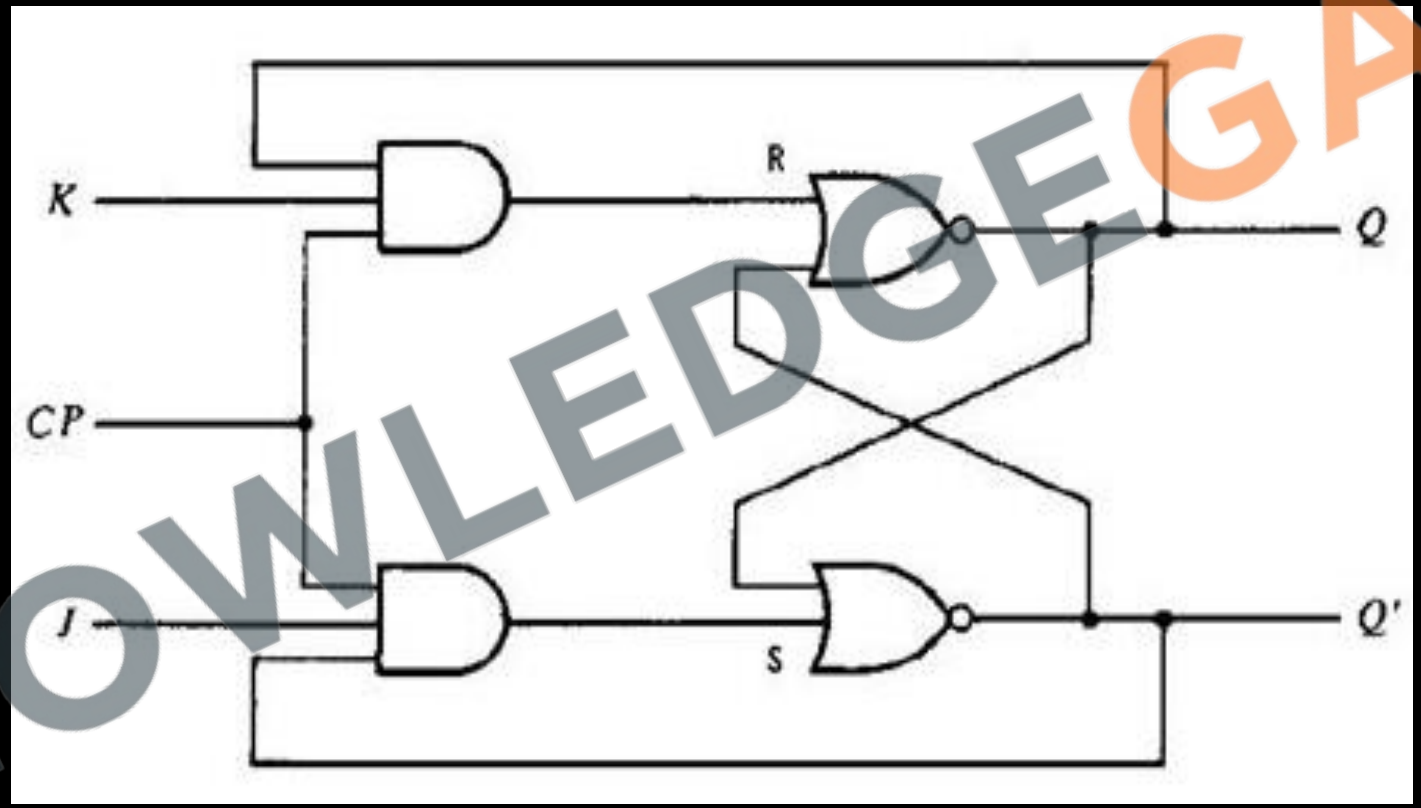


Notes

- Flip-flops can be either level-triggered (asynchronous, transparent or opaque) or edge-triggered (synchronous, or clocked).
- The term flip-flop has historically referred generically to both level-triggered and edge-triggered circuits that store a single bit of data using gates.
- Recently, some authors reserve the term *flip-flop* exclusively for discussing clocked circuits;
  - the simple ones are commonly called *transparent latches*. Using this terminology, a level-sensitive flip-flop is called a transparent latch
  - Whereas an edge-triggered flip-flop is simply called a flip-flop.
- Using either terminology, the term "flip-flop" refers to a device that stores a single bit of data. The terms "edge-triggered", and "level-triggered" may be used to avoid ambiguity.

JK flip flop

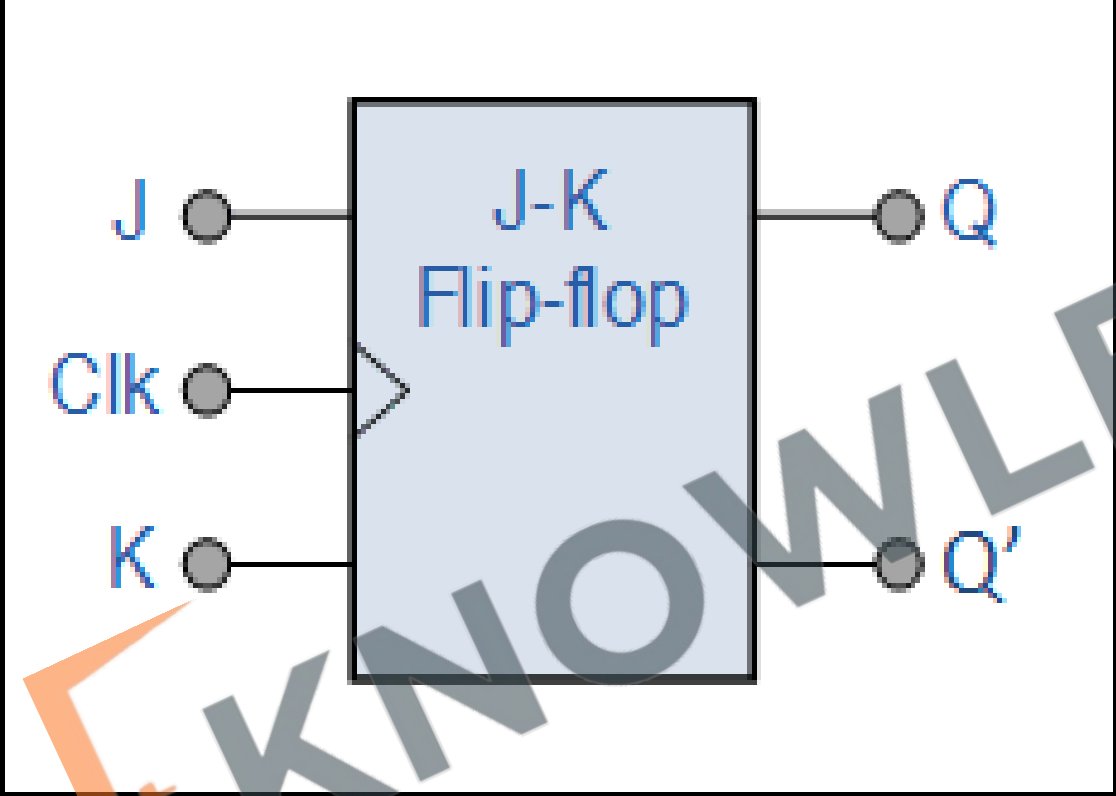
- SR flip flop when both S and R = 1, results in invalid output. To resolve the problem we use JK Flip Flop. We take a feedback from the outputs.



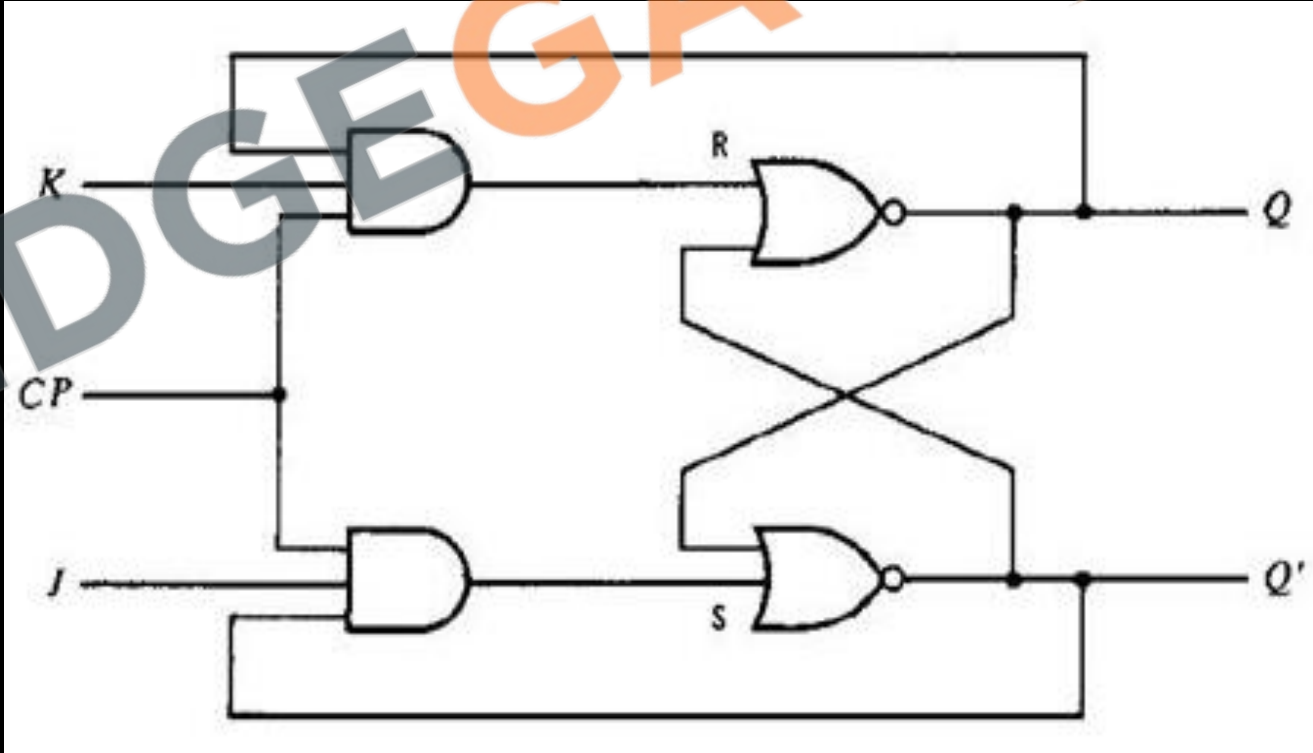
JK flip flop

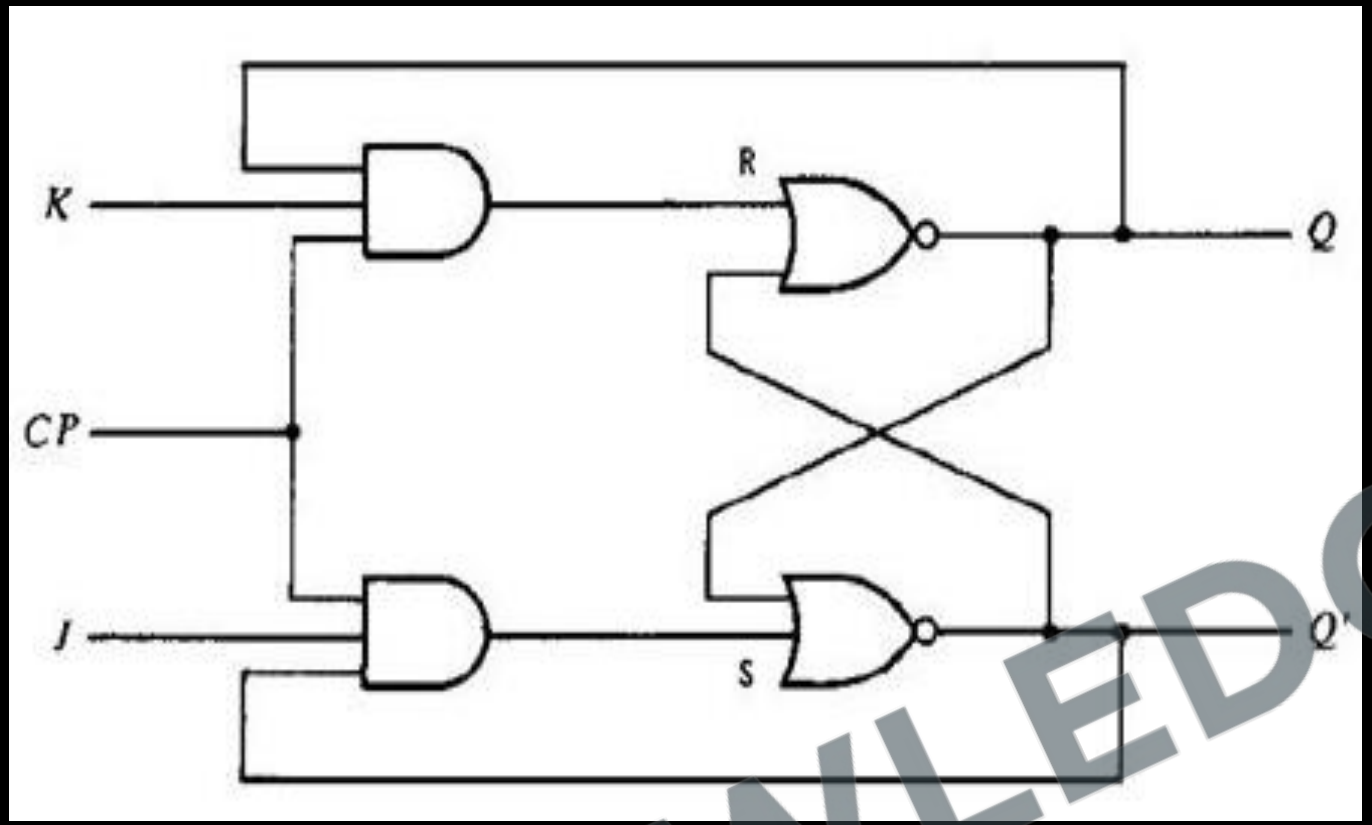
- SR flip flop when both S and R = 1, results in invalid output. To resolve the problem we use JK Flip Flop. We take a feedback from the outputs.

Block Diagram



Implementation





Truth Table

J	K	Q_n	Q_{n+1}
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Truth Table

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

K-Map

	JK	J'K'	J'K	JK	JK'
q		00	01	11	10
q'	00				
q	01				

Characteristics Equation

Q_{n+1} =

Truth Table

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Characteristics Equation

$$Q_{n+1} = J\,Q'_n + K'Q_n$$

Function Table

J	K	Q_{n+1}
0	0	
0	1	
1	0	
1	1	

Truth Table

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Excitation Table

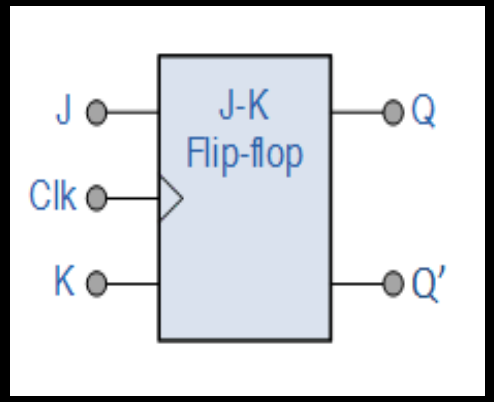
Q_n	Q_{n+1}	J	K
0	0		
0	1		
1	0		
1	1		

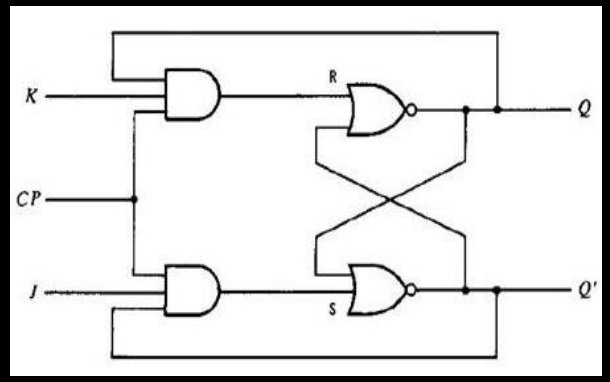
Truth Table

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

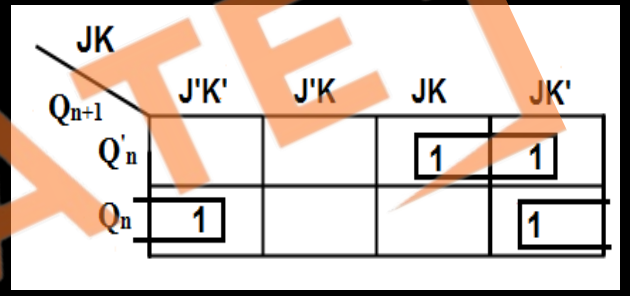
State Diagram

Block Diagram	Implementation	Truth Table	K-Map





J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

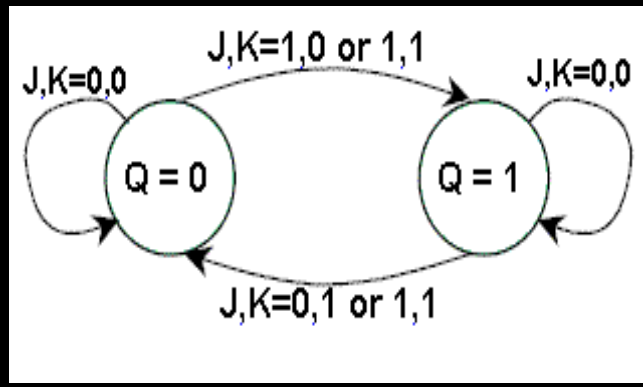


Characteristics Equation	Function Table	Excitation Table	State Diagram

$$Q_{n+1} = J\,Q'_n + K'Q_n$$

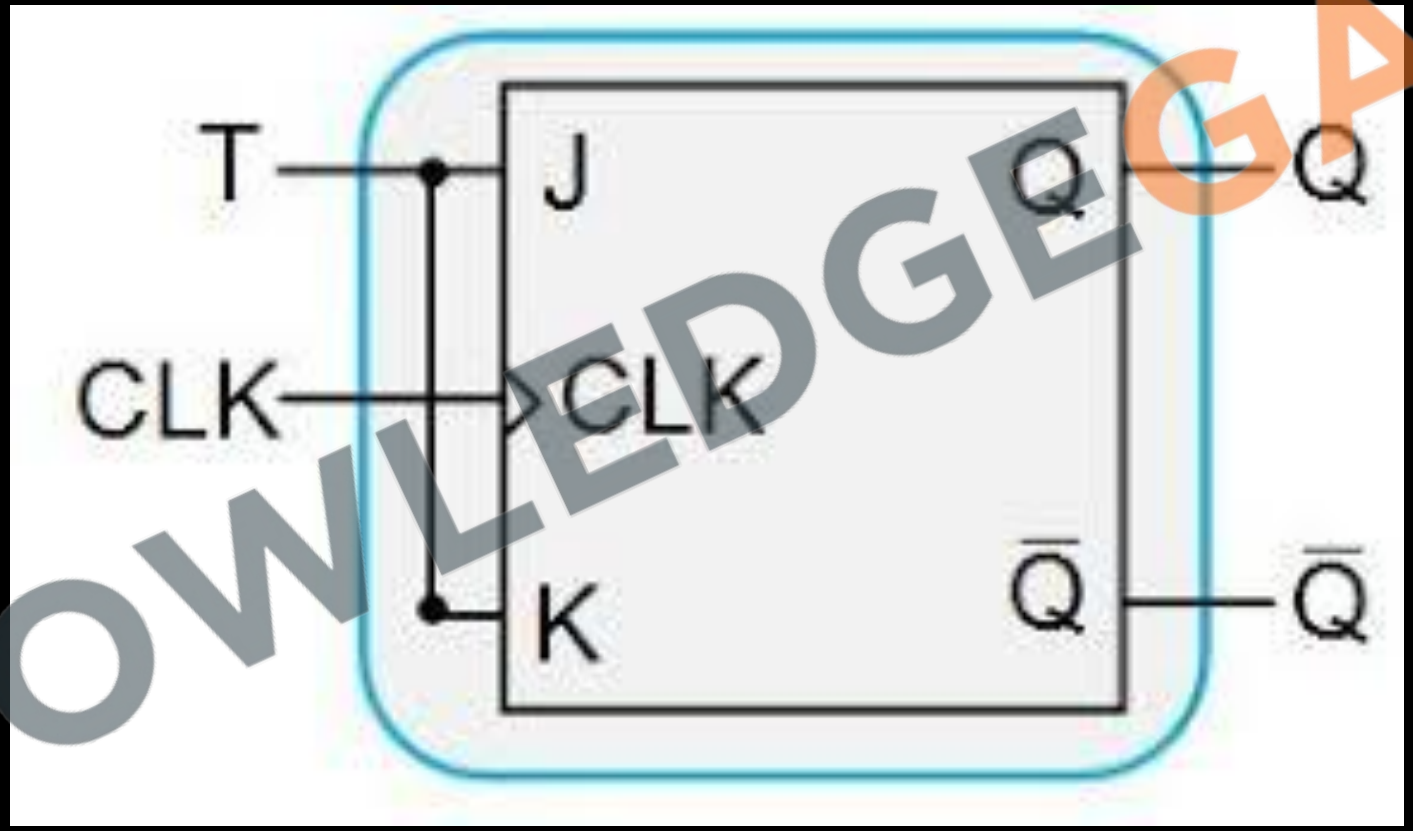
J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	Q'_n

Q_n	Q_{n+1}	J	K
0	0	0	d
0	1	1	d
1	0	d	1
1	1	d	0



T(Toggle) flip flop

- The T (toggle) flip-flop is a complementing flip-flop and can be obtained from a JK flip-flop when inputs J and K are tied together.



J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Truth Table

T	Q_n	Q_{n+1}
0	0	
0	1	
1	0	
1	1	

Truth Table

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

K-Map

	T	T’
q		0
q’	0	
q	1	

Characteristics Equation

Q_{n+1} =

Truth Table

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Characteristics Equation

$$Q_{n+1} = T \oplus Q_n$$

Function Table

T	Q_{n+1}
0	
1	

Truth Table

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Excitation Table

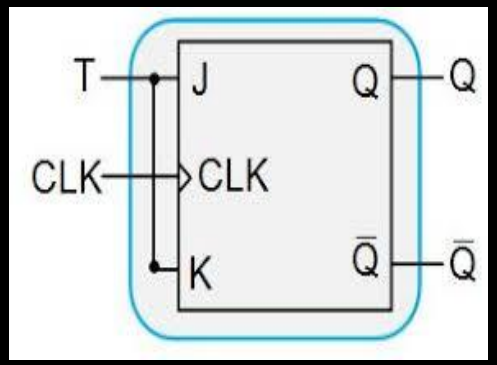
Q_n	Q_{n+1}	T
0	0	
0	1	
1	0	
1	1	

Truth Table

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

State Diagram

Block Diagram	Truth Table	K-Map



T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

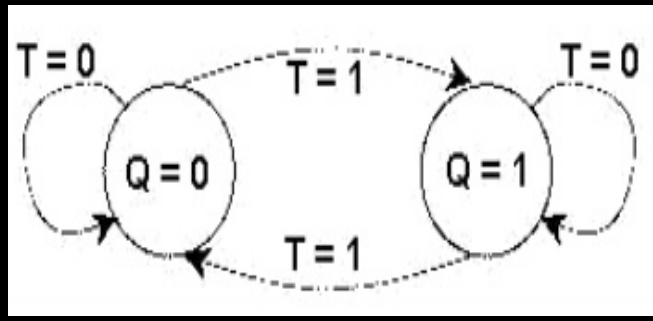
	T	T'	T
q		0	1
q'	0		1
q	1	1	

Characteristics Equation	Function Table	Excitation Table	State Diagram

$$Q_{n+1} = T \oplus Q_n$$

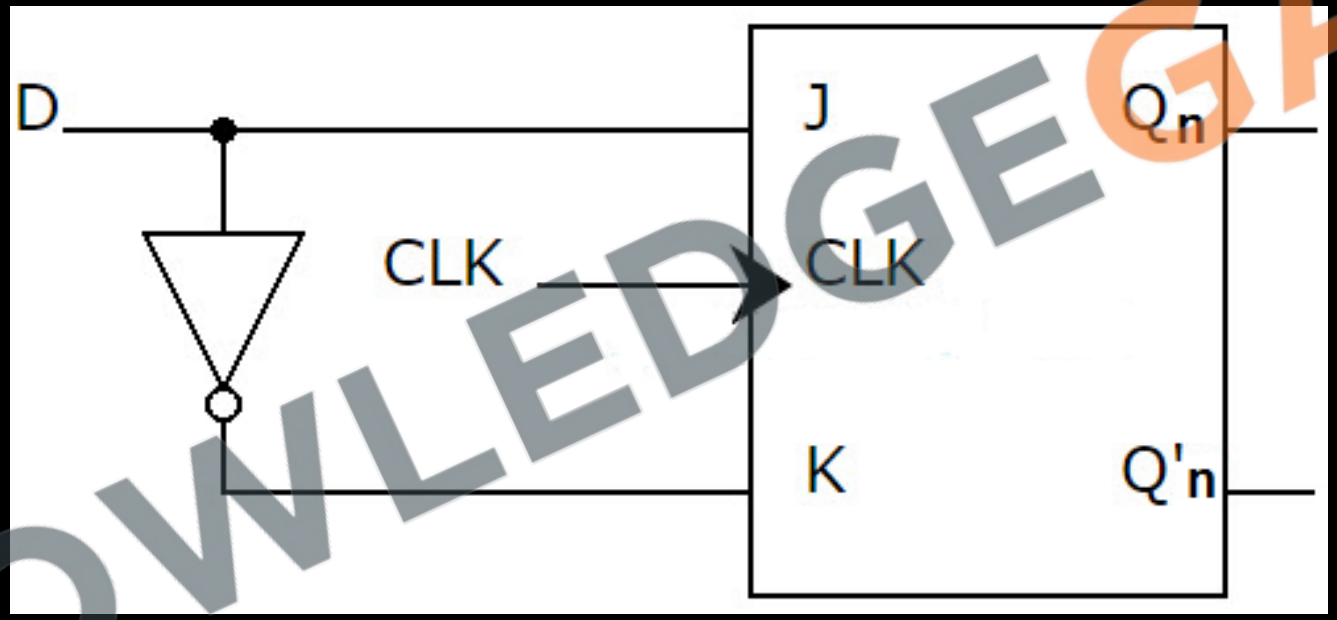
T	Q_{n+1}
0	Q_n
1	Q_n'

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0



D flip flop

- The D (Data/Delay) flip-flop, tracks the input at D and produces the same value as output.



J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Truth Table

D	Q_n	Q_{n+1}
0	0	
0	1	
1	0	
1	1	

Truth Table

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

K-Map

	D	D’	D
q		0	1
q’	0		
q	1		

Characteristics Equation

Q_{n+1} =

Truth Table

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

Characteristics Equation

$$Q_{n+1} = D$$

Function Table

D	Q_{n+1}
0	
1	

Truth Table

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

Excitation Table

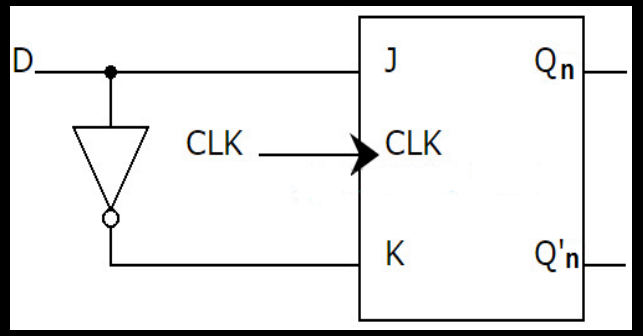
Q_n	Q_{n+1}	D
0	0	
0	1	
1	0	
1	1	

Truth Table

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

State Diagram

Block Diagram	Truth Table	K-Map



D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

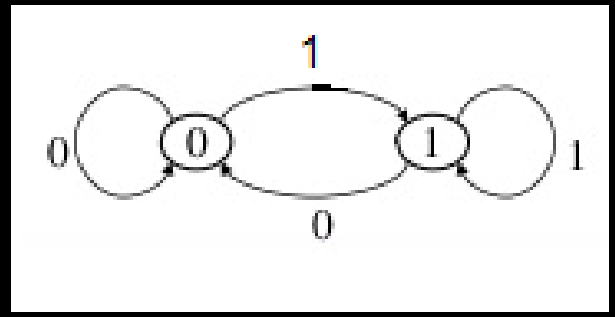
	D	D'	D
q		0	1
q'	0		1
q	1		1

Characteristics Equation	Function Table	Excitation Table	State Diagram

$$Q_{n+1} = D$$

D	Q_{n+1}
0	0
1	1

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1



Flip Flops Conversion

1. We will require the **Characteristics Table** of **target flip flop.**
2. We will require the **Excitation Table** of **given flip flop.**
3. Determine the excitation values for characteristics table.
4. Obtain the expressions for input of given flip flop in terms of target.

Convert D Flip Flop to T Flip Flop

Characteristics table of T- Flip Flop

T	Q_n	Q_{n+1}
0	0	
0	1	
1	0	
1	1	

Excitation Table of D flip flop

Q_n	Q_{n+1}	D
0	0	
0	1	
1	0	
1	1	

Characteristics table of T- Flip Flop

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

T	Q_n	Q_{n+1}	D
0	0	0	
0	1	1	
1	0	1	
1	1	0	

Excitation Table of D flip flop

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Convert D Flip Flop to T Flip Flop

Excitation values for characteristics table

T	Q_n	Q_{n+1}	D
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Obtaining simplified expression

T	Q_n	D
0	0	
0	1	
1	0	
1	1	

Obtaining simplified expression

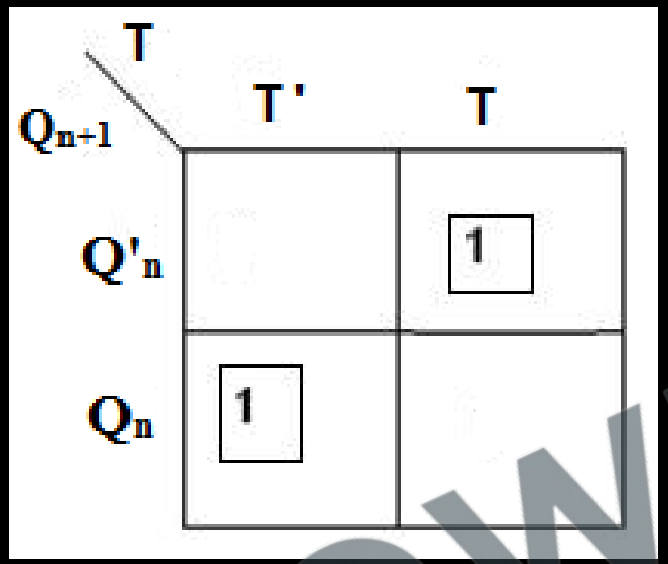
T	Q_n	D
0	0	0
0	1	1
1	0	1
1	1	0

Using K-Map to Simplify

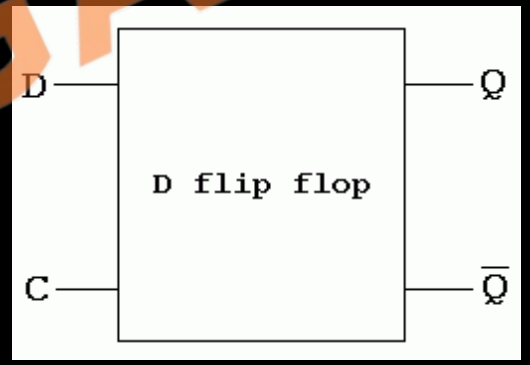
	T	T'	T
q	0	0	1
q'	1		
q			

Convert D Flip Flop to T Flip Flop

Using K-Map to Simplify



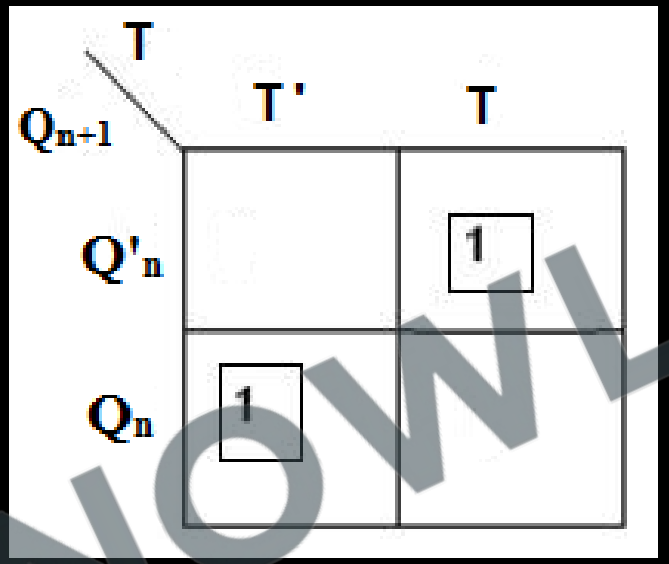
Block Diagram



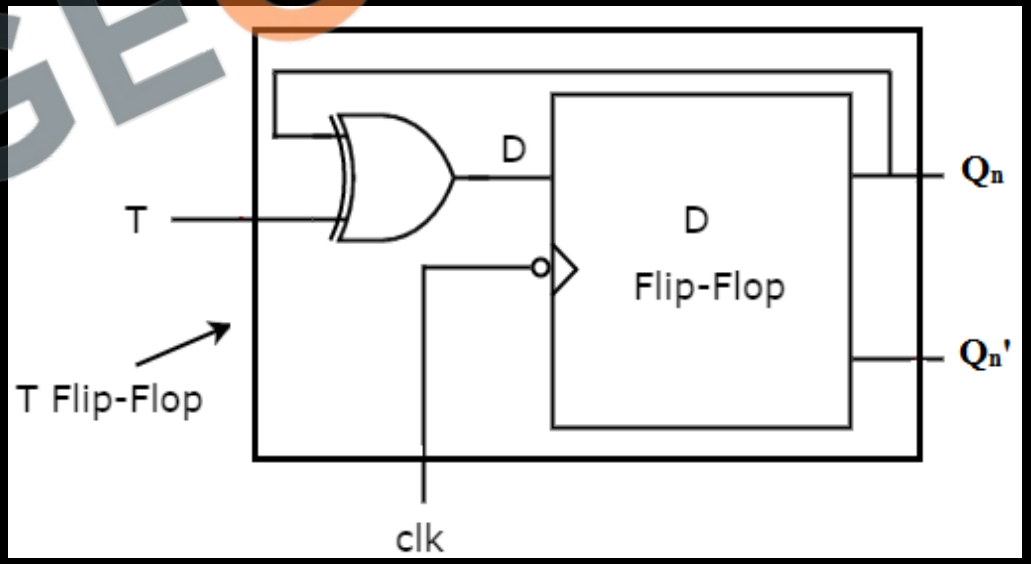
$$D = Q_n \oplus T$$

Convert D Flip Flop to T Flip Flop

Using K-Map to Simplify



Block Diagram



$$D = Q_n \oplus T$$

Flip Flops Conversion

1. We will require the **Characteristics Table** of **target flip flop.**
2. We will require the **Excitation Table** of **given flip flop.**
3. Determine the excitation values for characteristics table.
4. Obtain the expressions for input of given flip flop in terms of target.

Convert T flip flop to JK flip flop

Characteristics table of JK- Flip Flop

J	K	Q_n	Q_{n+1}
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Excitation Table of T flip flop

Q_n	Q_{n+1}	T
0	0	
0	1	
1	0	
1	1	

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

J	K	Q_n	Q_{n+1}	T
0	0	0	0	
0	0	1	1	
0	1	0	0	
0	1	1	0	
1	0	0	1	
1	0	1	1	
1	1	0	1	
1	1	1	0	

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Convert T flip flop to JK flip flop

Excitation values for characteristics table

J	K	Q_n	Q_{n+1}	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	1

Obtaining simplified expression

J	K	Q_n	T
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Convert T flip flop to JK flip flop

Using K-Map to Simplify

J	K	Q_n	T
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

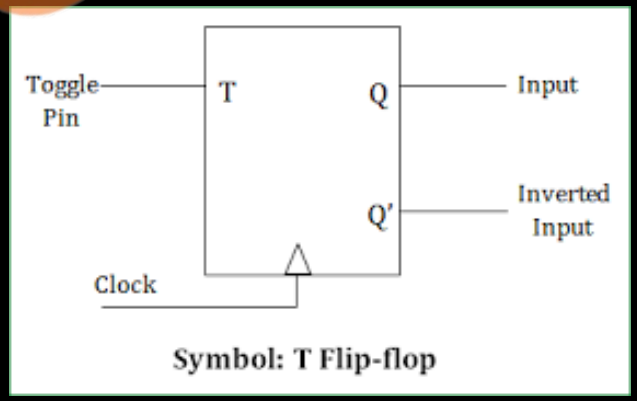
	JK	J'K'	J'K	JK	JK'
q		00	01	11	10
q'	00				
q	01				

T =

Convert T flip flop to JK flip flop

Block Diagram

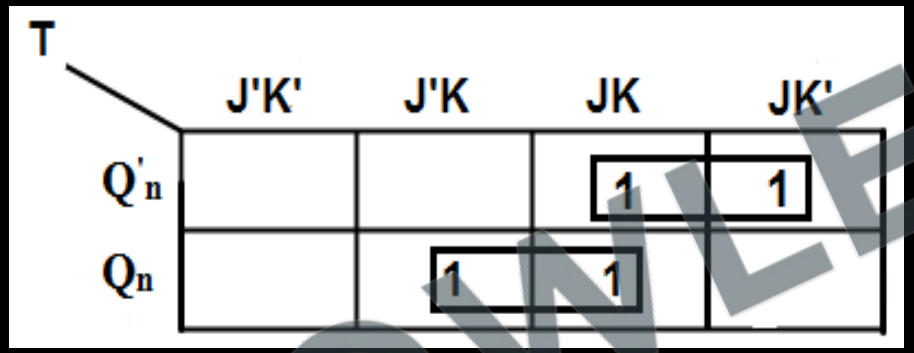
T =



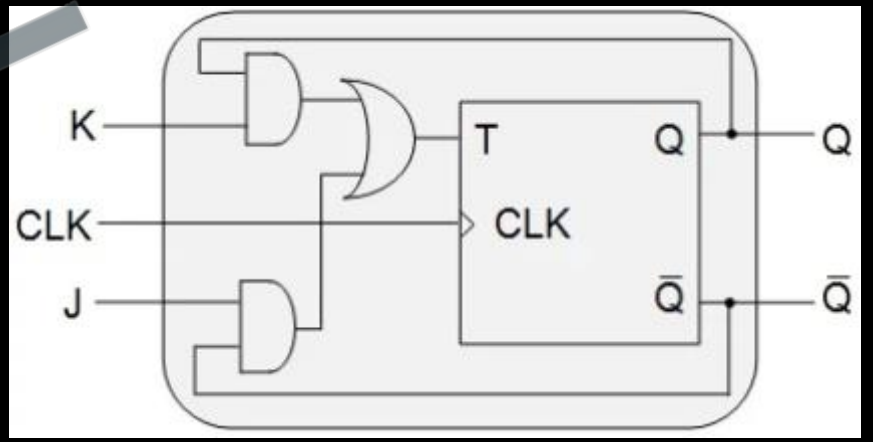
Symbol: T Flip-flop

Convert T flip flop to JK flip flop

Using K-Map to Simplify



Block Diagram



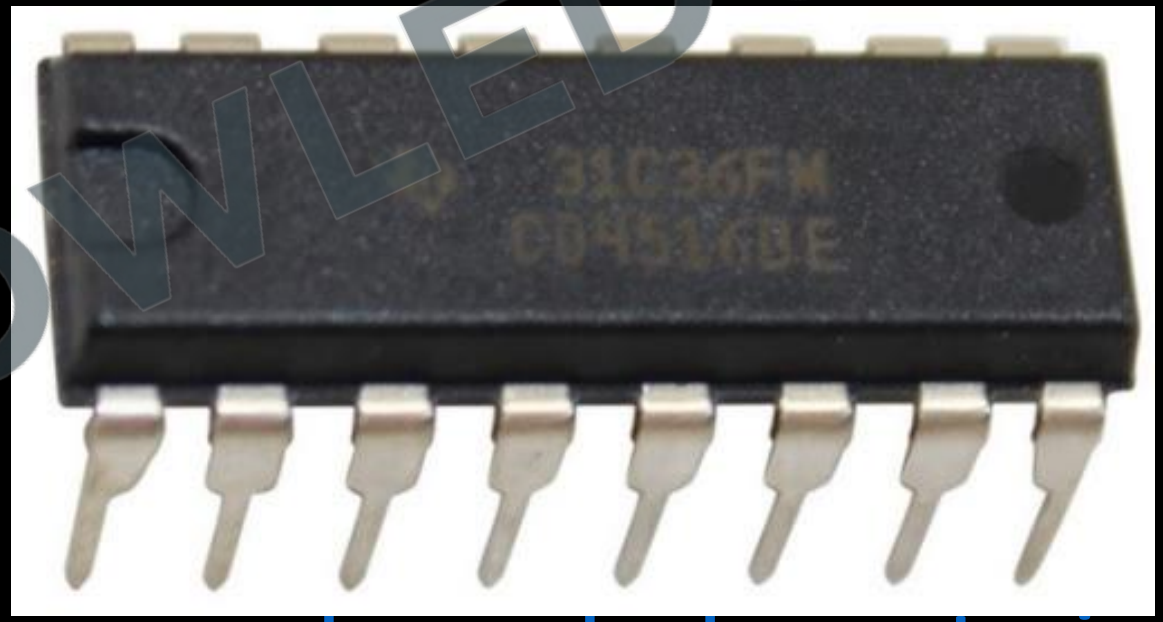
$$T = J Q_n' + K Q_n$$

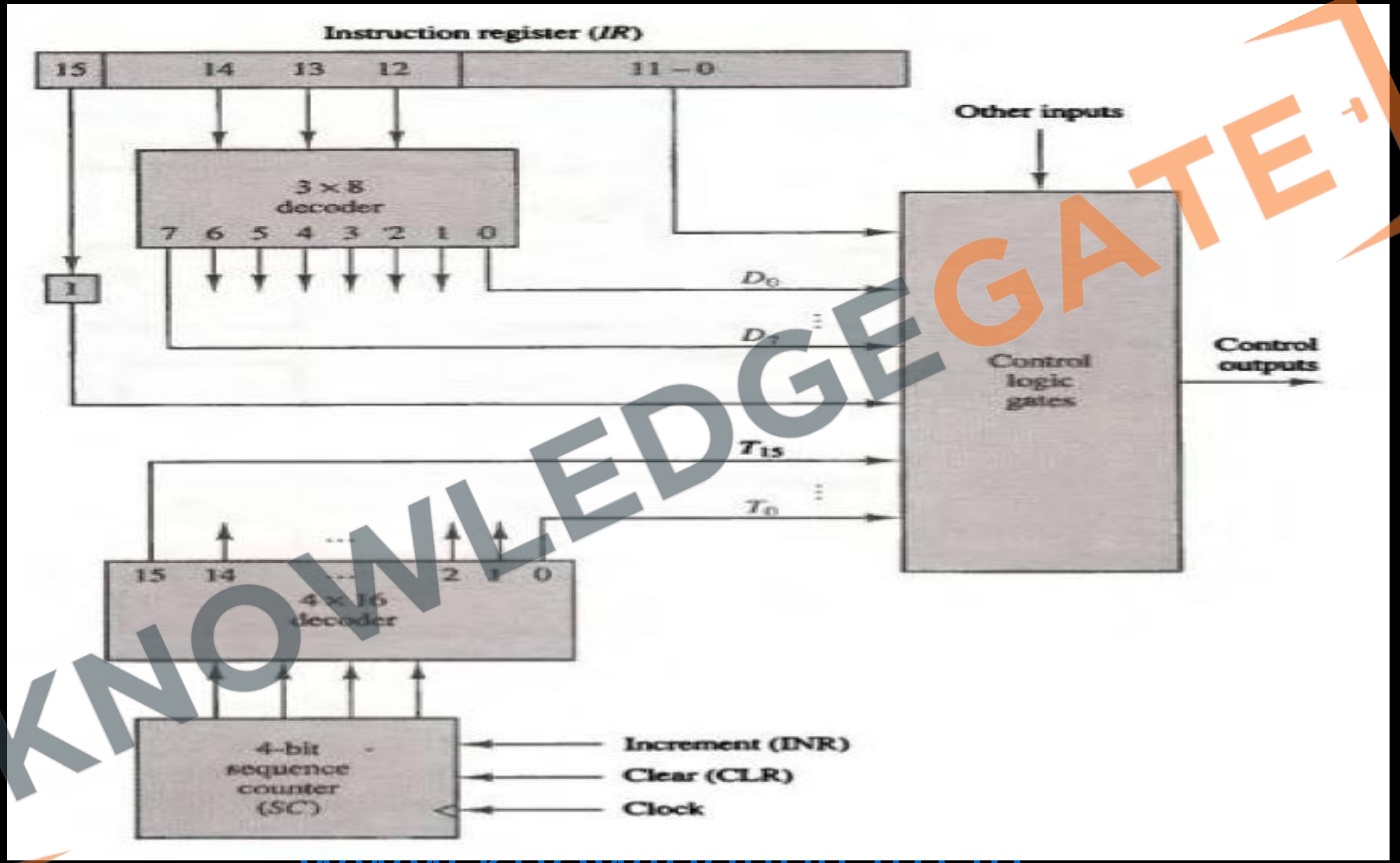
Flip Flops Conversion

1. We will require the **Characteristics Table** of **target flip flop.**
2. We will require the **Excitation Table** of **given flip flop.**
3. Determine the excitation values for characteristics table.
4. Obtain the expressions for input of given flip flop in terms of target.

Basics of counters

- A counter is a sequential circuit that goes through a predetermined sequence of binary states upon the application of input pulses. A counter that follows the binary number sequence is called a binary counter.
- An n - bit binary counter consists of n flip-flops and can count in binary from 0 through $2^n - 1$. Counters are available in two categories: **synchronous counters** and **Ripple counters (asynchronous)**.





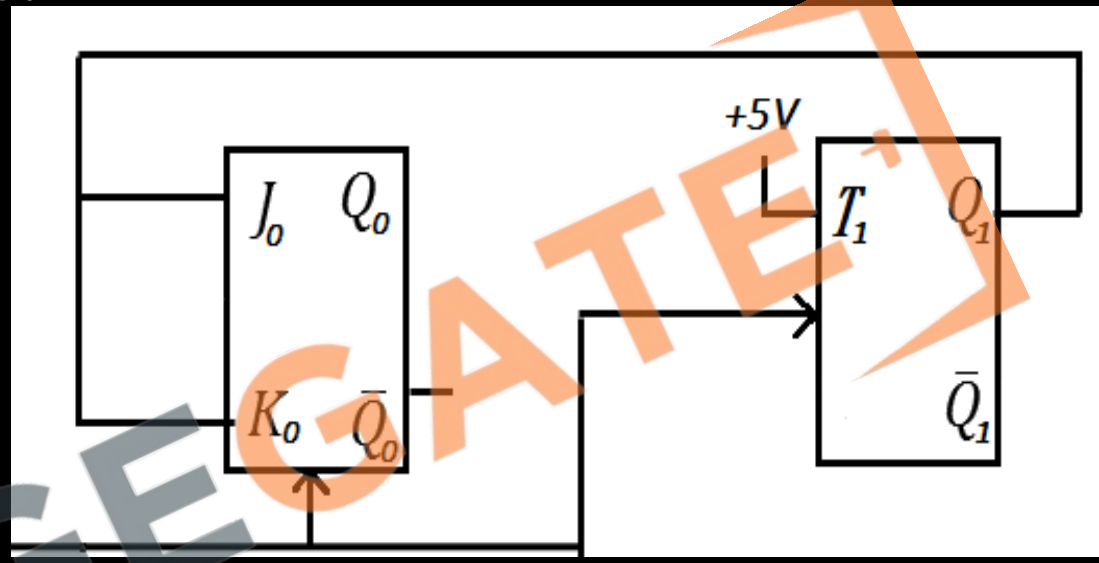
Q Find out the counting sequence for the counter below?

The characteristics equation for JK Flip Flop is Q_{n+1} =

Q_{0N} =

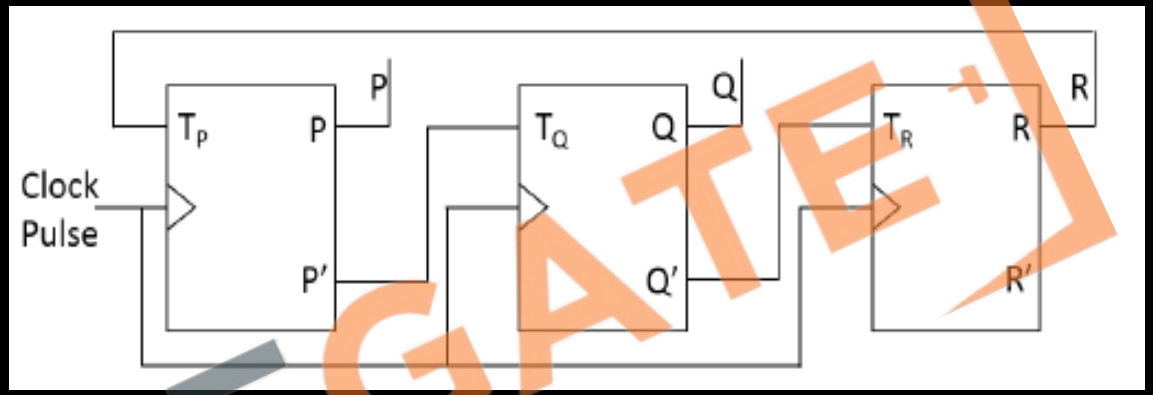
The characteristics equation for T Flip Flop is Q_{n+1} =

Q_{1N} =



Present State		Next State	
Q_{1p}	Q_{0p}	Q_{1N}	Q_{0N}
0	0		
0	1		
1	0		
1	1		

Q Consider a 3-bit counter, designed using T flip-flops, as shown below:



Present State			Next State		
Q_{2p}	Q_{1p}	Q_{0p}	Q_{2N}	Q_{1N}	Q_{0N}
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Q Design a synchronous counter for sequence: 0 → 1 → 2 → 3 → 0, using T flip flop.

Present State		Next State	
Q_{1p}	Q_{0p}	Q_{1N}	Q_{0N}
0	0		
0	1		
1	0		
1	1		

Q Example: Design a synchronous counter for sequence: 0 → 1 → 2 → 3 → 0, using D flip flop.

Present State		Next State	
Q_{1p}	Q_{0p}	Q_{1N}	Q_{0N}
0	0		
0	1		
1	0		
1	1		

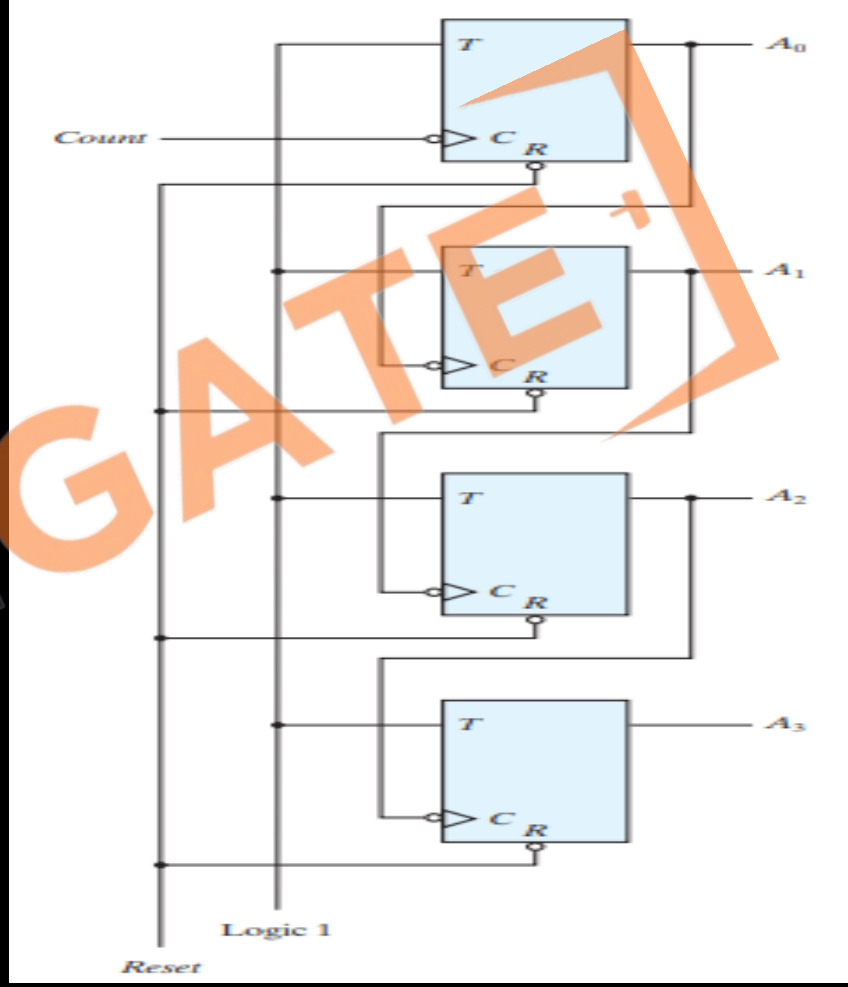
Q Design synchronous counter for sequence: 0 → 1 → 3 → 4 → 5 → 7 → 0, using T flip-flop.

Present State			Next State		
Q_{2p}	Q_{1p}	Q_{0p}	Q_{2N}	Q_{1N}	Q_{0N}
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

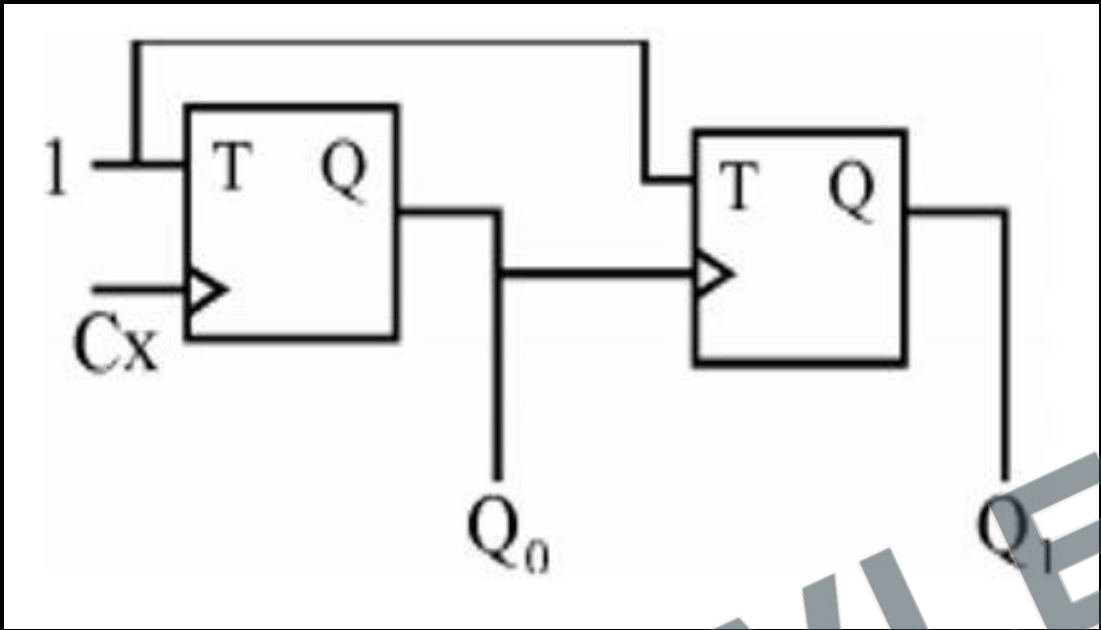
RIPPLE COUNTERS

1. A binary ripple counter consists of a series connection of complementing flip-flops, with the output of each flip-flop connected to the input of the next higher order flip-flop.

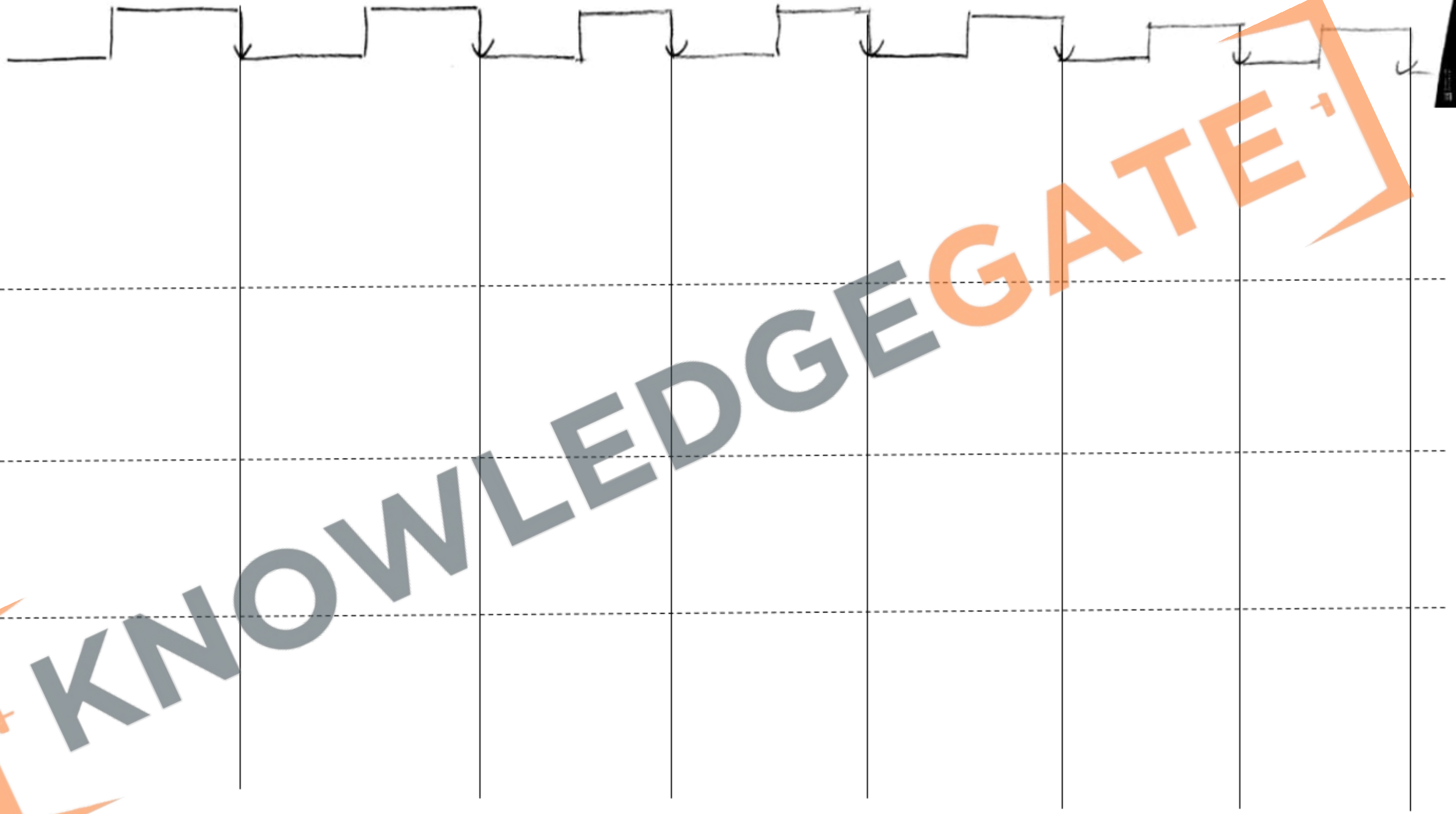
2. The flip-flop holding the least significant bit receives the incoming count pulses.



Q In the sequential circuit shown below, if the initial value of the output Q_1Q_0 is 00, what are the next four values of Q_1Q_0?



Present State		Next State	
Q_{1p}	Q_{0p}	Q_{1N}	Q_{0N}
0	0		
0	1		
1	0		
1	1		



Nature of Clock	Nature of Feedback	Nature of counting
+ve		UP Counting
+ve		Down Counting
-ve		UP Counting
-ve		Down Counting

Asynchronous counters

SYNCHRONOUS COUNTER	ASYNCHRONOUS COUNTER
In synchronous counter, all flip flops are triggered with same clock simultaneously.	In asynchronous counter, different flip flops are triggered with different clock, not simultaneously.
Synchronous Counter is faster than asynchronous counter in operation. $T_{delay} = T_{FF} + T_{CC}$	Asynchronous Counter is slower than synchronous counter in operation. $T_{delay} = n \times T_{FF} + T_{CC}$
Synchronous Counter is also called Parallel Counter.	Asynchronous Counter is also called Serial Counter.
Synchronous Counter designing as well implementation are complex due to increasing the number of states.	Asynchronous Counter designing as well as implementation is very easy.
Synchronous Counter will operate in any desired count sequence.	Asynchronous Counter will operate only in fixed count sequence (UP/DOWN).
Synchronous Counter examples are: Ring counter, Johnson counter.	Asynchronous Counter examples are: Ripple UP counter, Ripple DOWN counter.
A synchronous sequential circuit is a system whose behavior can be defined from the knowledge of its signals at discrete instants of time.	The behavior of an asynchronous sequential circuit depends upon the input signals at any instant of time and the order in which the inputs change.

Registers

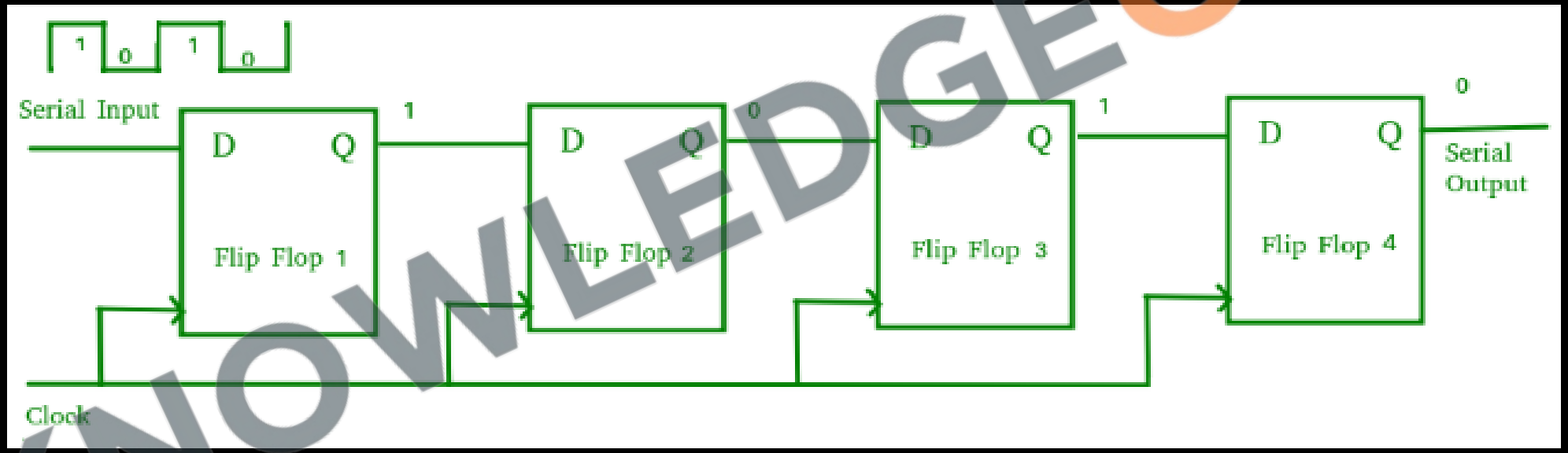
- Registers are basically storing devices which are also designed using basic element called flip-flop.
- D-flip-flop are most popular choice for register because they don’t perform any functionality and output is simply based on current input so, they act as a buffer.
- Apart from storing registers sometimes also be used in performing basic mathematical operation like multiply by 2 by left shit and divide by 2 by right shift.

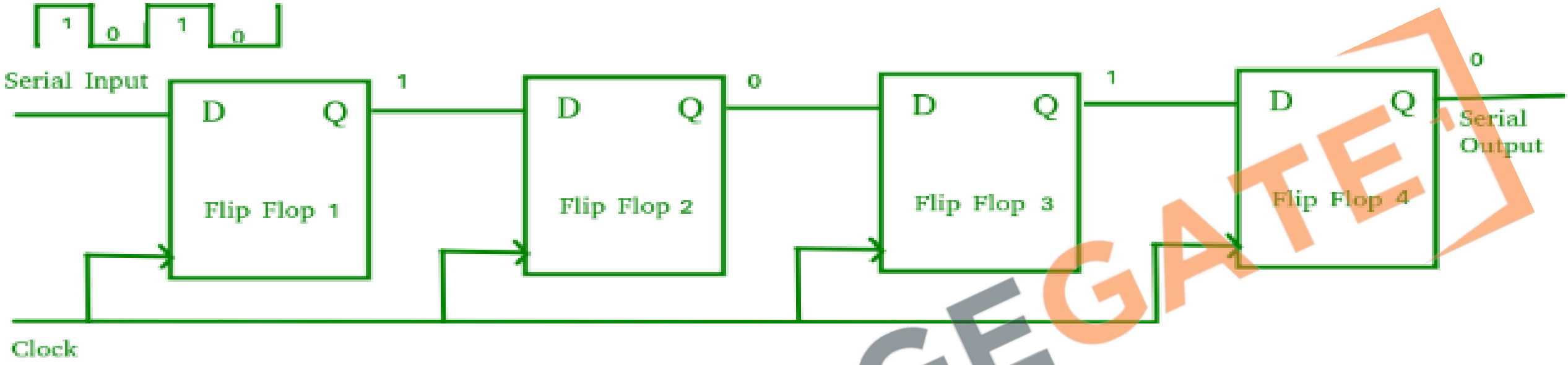
Shift Register Types & Operations

- There are four different modes in which registers operate.
  1. Serial In-Serial Out (SISO)
  2. Serial In-Parallel Out (SIPO)
  3. Parallel In-Serial Out (PISO)
  4. Parallel In- Parallel Out (PIPO)

Serial In-Serial Out (SISO)

- The shift register, which allows serial input (one bit after the other through a single data line) and produces a serial output is known as Serial-In Serial-Out shift register.
- Since there is only one output, the data leaves the shift register one bit at a time in a serial pattern, thus the name Serial-In Serial-Out Shift Register.





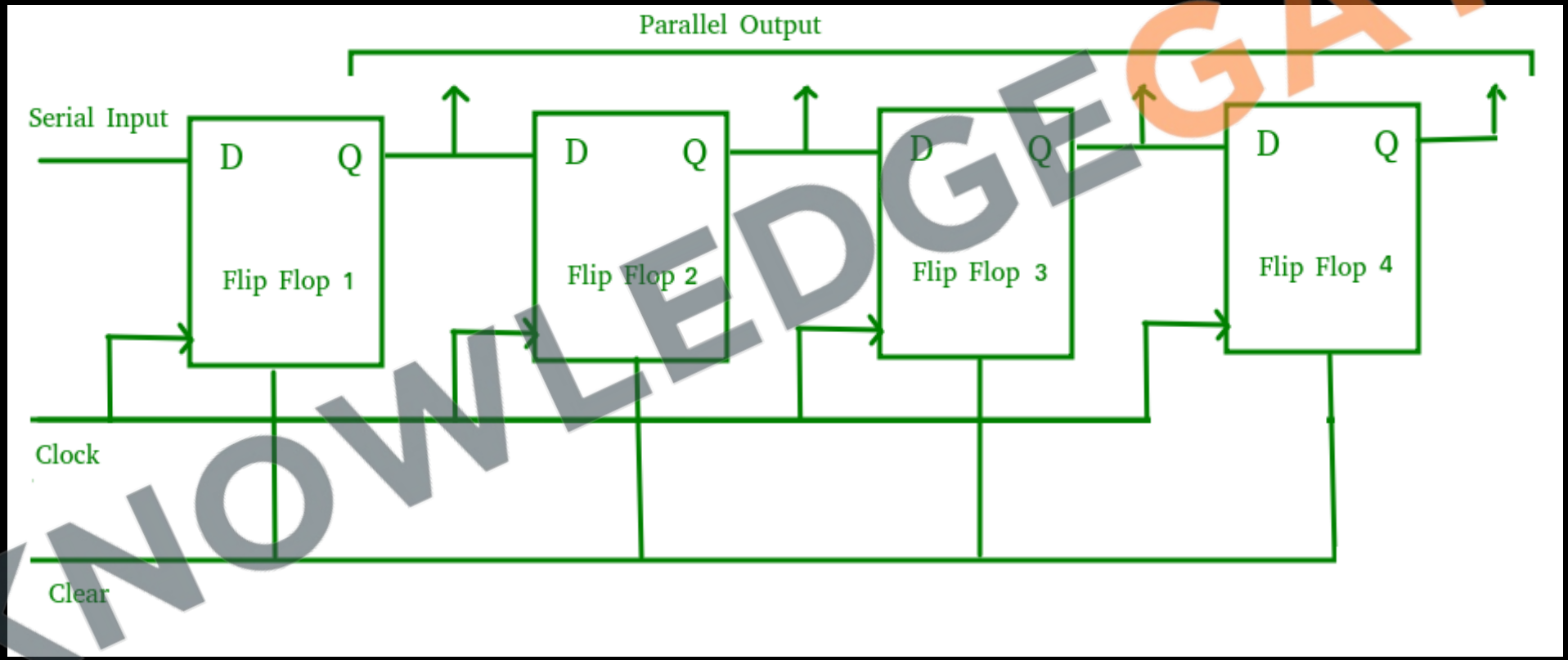
Sequence	Input	Q_2	Q_1	Q_0	Output

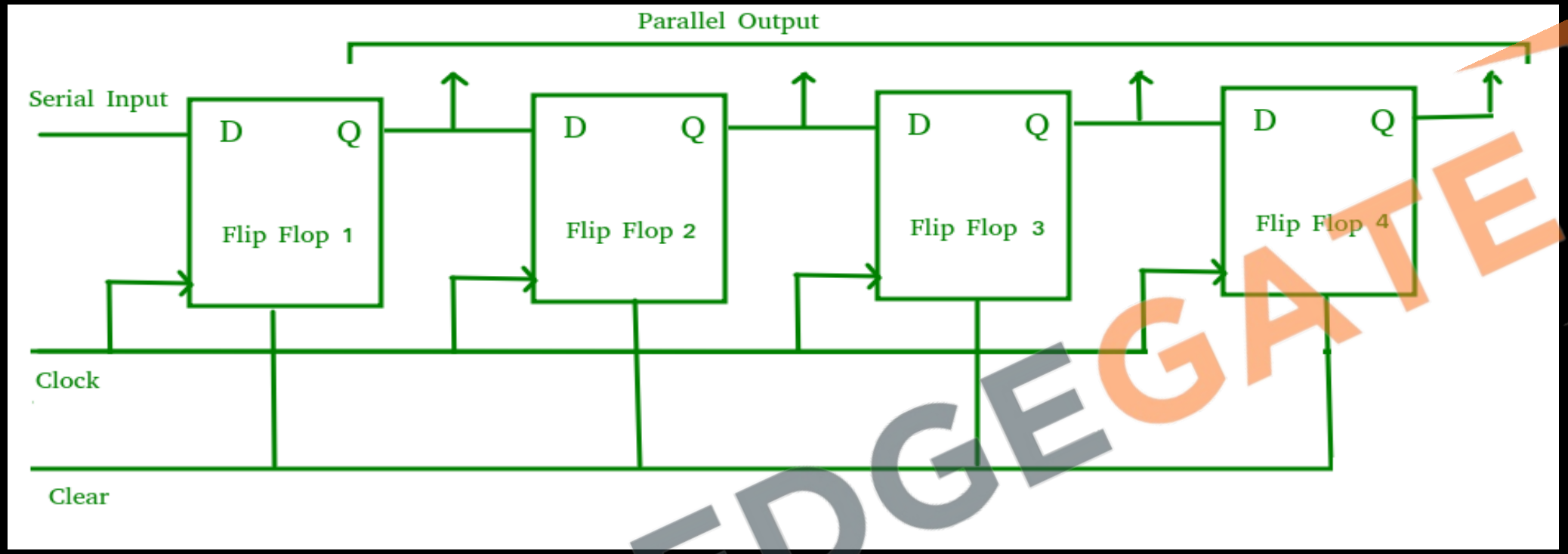
- The main use of a SISO is to act as a delay element.
- In SISO registers to provide n bit data serially in it requires n clock pulse and to provide serial output it requires n - 1 clock pulses.

	No of clock (write)	No of clock (Read)	total
SISO			
SIPO			
PISO			
PIPO			

Serial-In Parallel-Out shift Register (SIPO)

- The shift register, which allows serial input (one bit after the other through a single data line) and produces a parallel output is known as Serial-In Parallel-Out shift register.



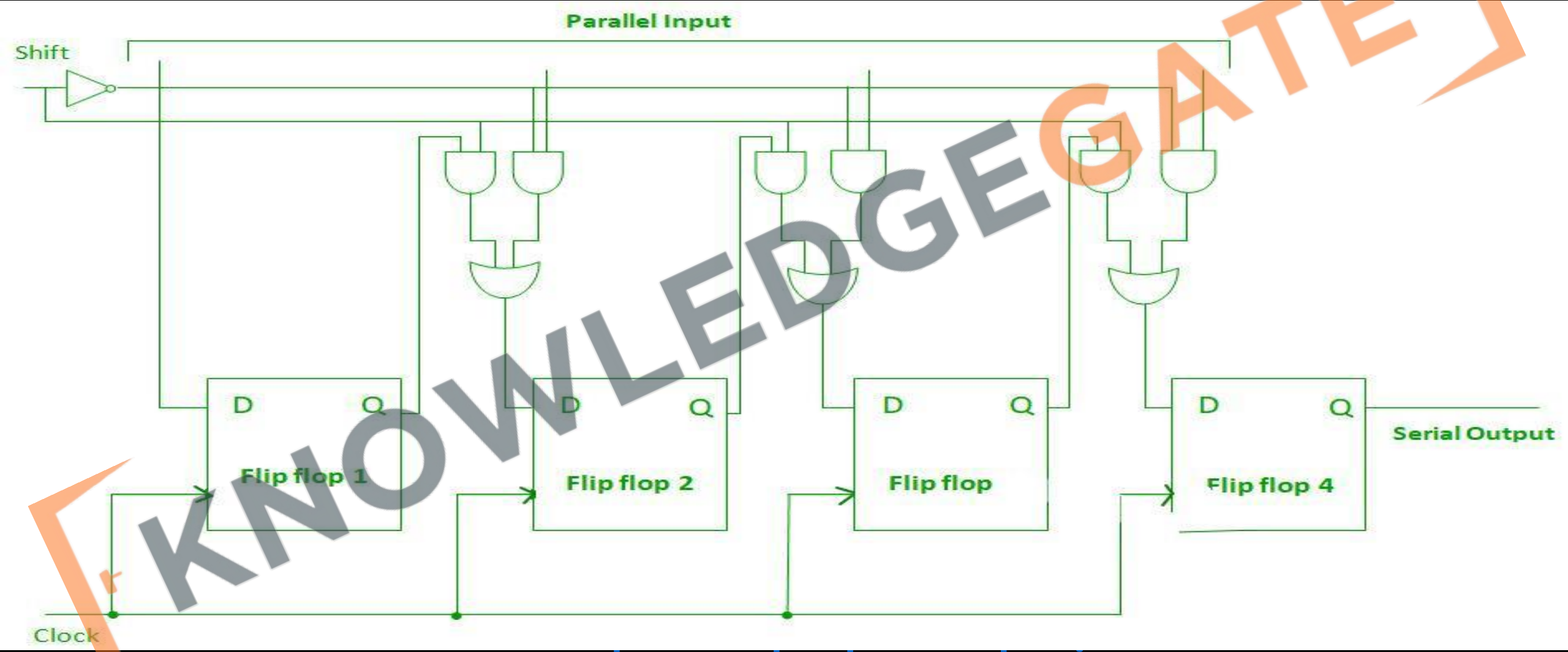


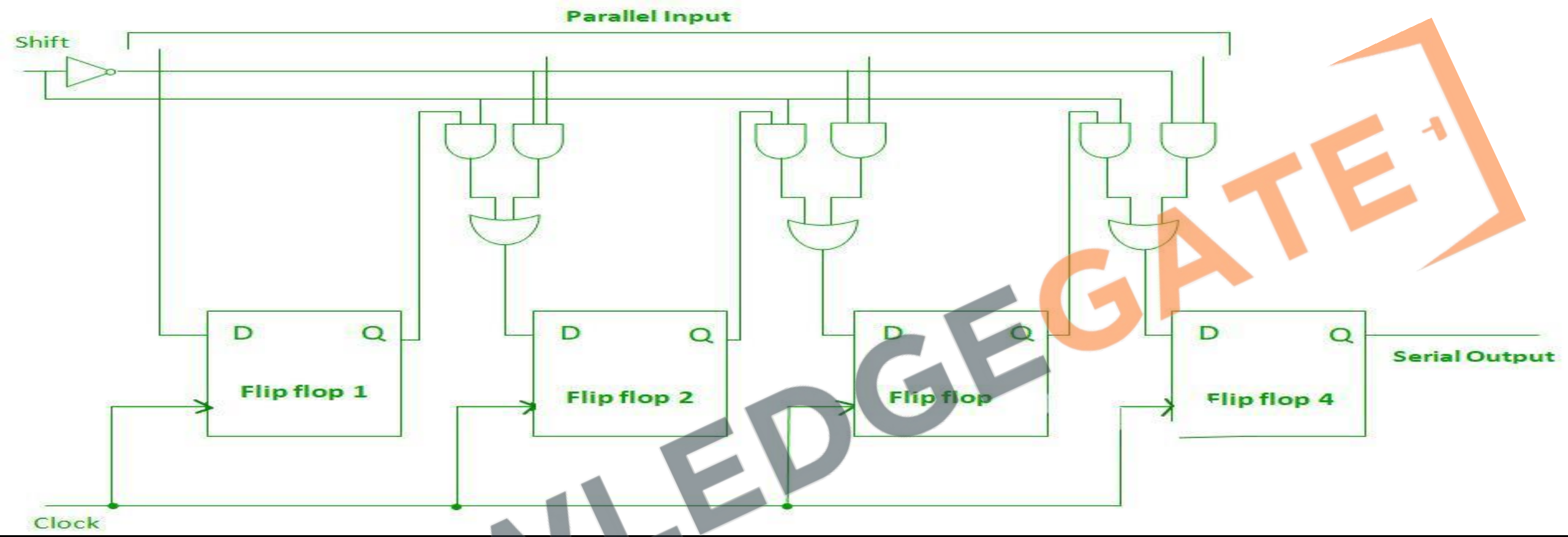
- They are used in communication lines where demultiplexing of a data line into several parallel lines is required because the main use of the SIPO register is to convert serial data into parallel data.

	No of clock (write)	No of clock (Read)	total
SISO	n	n-1	2n-1
SIPO			
PISO			
PIPO			

Parallel-In Serial-Out Shift Register (PISO)

- The shift register, which allows parallel input (data is given separately to each flip flop and in a simultaneous manner) and produces a serial output is known as Parallel-In Serial-Out shift register.



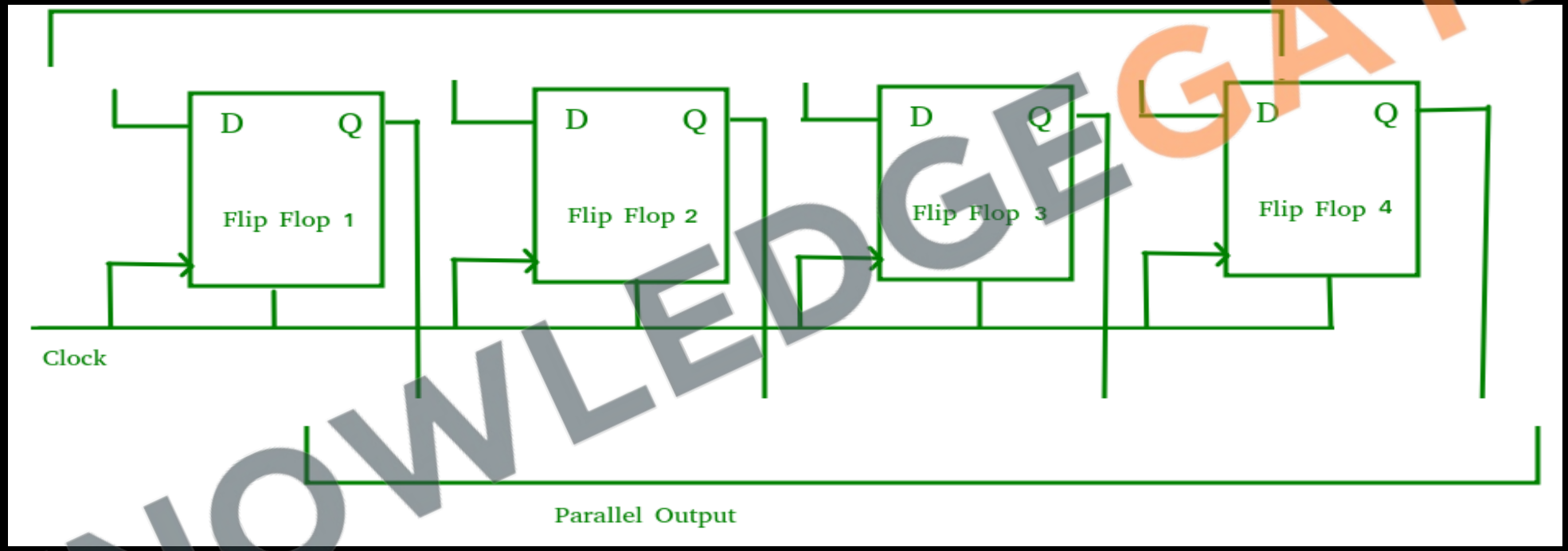


- The circuit consists of four D flip-flops which are connected.
- The clock input is directly connected to all the flip flops but the input data is connected individually to each flip flop through a multiplexer at the input of every flip flop.
- The output of the previous flip flop and parallel data input are connected to the input of the MUX and the output of MUX is connected to the next flip flop.
- A Parallel in Serial out (PISO) shift register us used to convert parallel data to serial data.

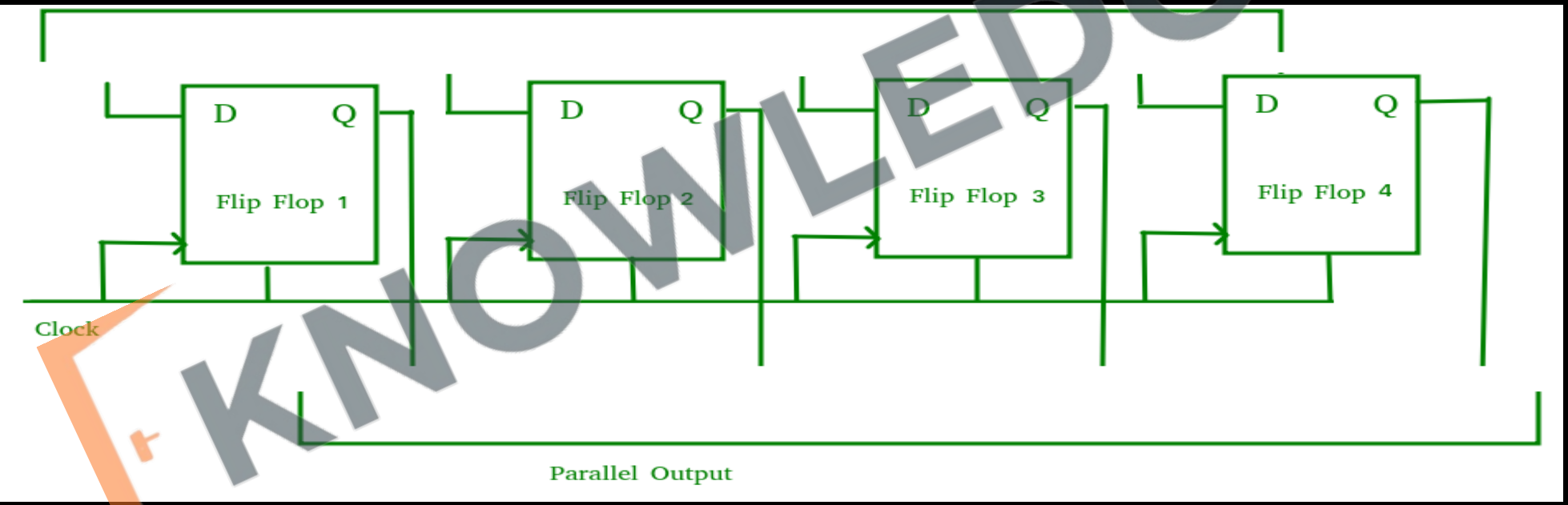
	No of clock (write)	No of clock (Read)	total
SISO	n	n-1	2n-1
SIPO	n	0	n
PISO			
PIPO			

Parallel-In Parallel-Out Shift Register (PIPO)

- The shift register, which allows parallel input (data is given separately to each flip flop and in a simultaneous manner) and also produces a parallel output is known as Parallel-In parallel-Out shift register.



- The circuit consists of four D flip-flops which are connected.
- In this type of register, there are no interconnections between the individual flip-flops since no serial shifting of the data is required.
- Data is given as input separately for each flip flop and in the same way, output also collected individually from each flip flop.
- A Parallel in Parallel out (PIPO) shift register is used as a temporary storage device and like SISO Shift register it acts as a delay element.
- For parallel input it requires 1 clock pulse and for parallel output it requires 0 clock.

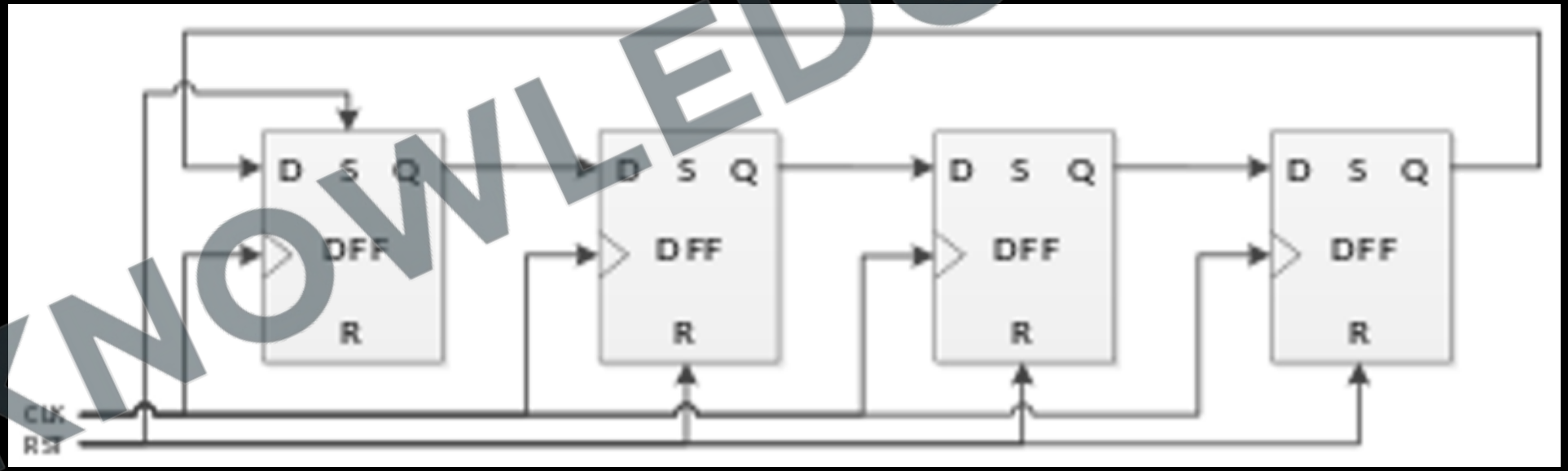


	No of clock (write)	No of clock (Read)	total
SISO	n	n-1	2n-1
SIPO	n	0	n
PISO	1	n-1	n
PIPO			

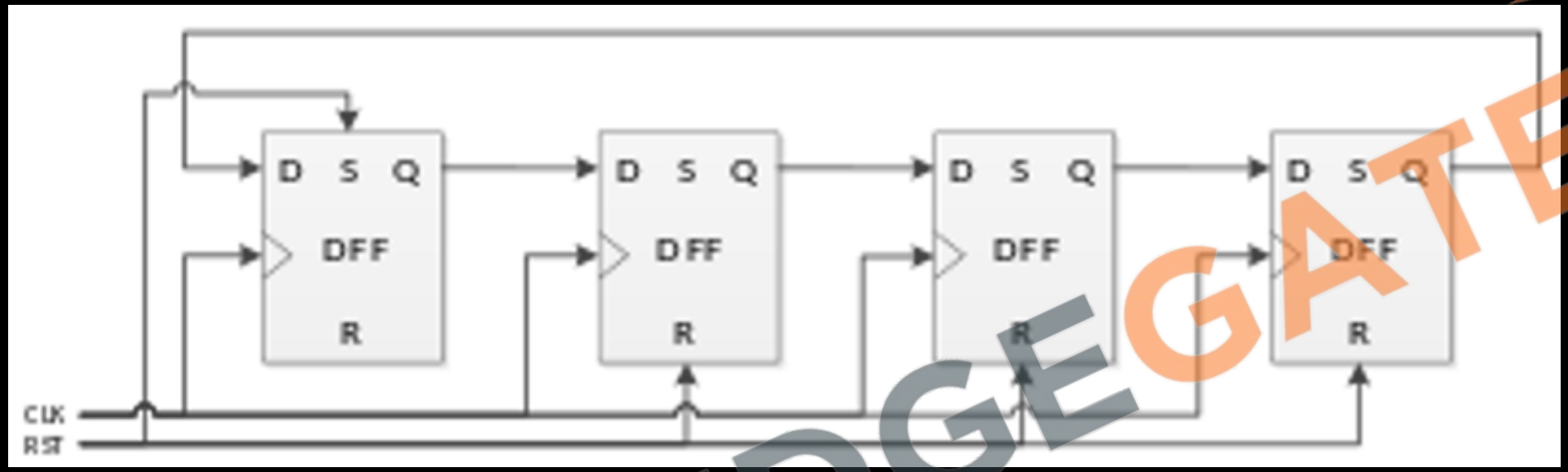
Ring Counter/ Straight Ring Counter

4-Bit Ring Counter

- A ring counter is a circular shift register with only one flip-flop being set at any particular time; all others are cleared.
- The single bit is shifted from one flip-flop to the next to produce the sequence of timing signals.
- Output of the last flip-flop is connected to the input of the first flip-flop in case of ring counter.
- ***No. of states in Ring counter = No. of flip-flop used***



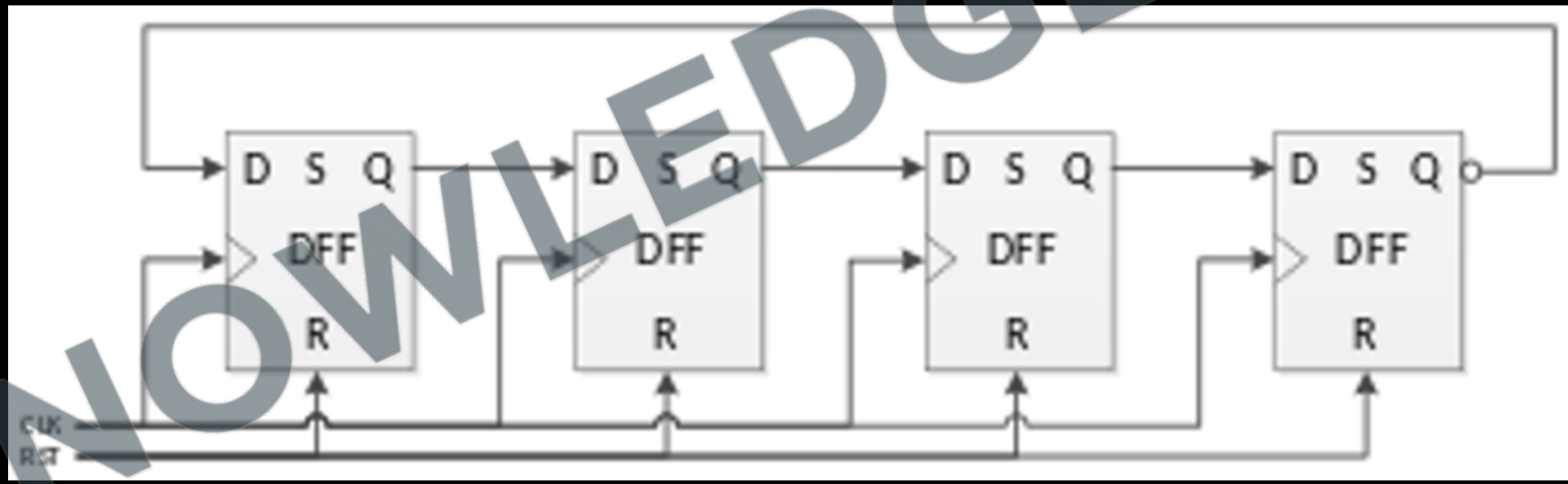
- Number of unused states in Ring Counter is 2^n-n

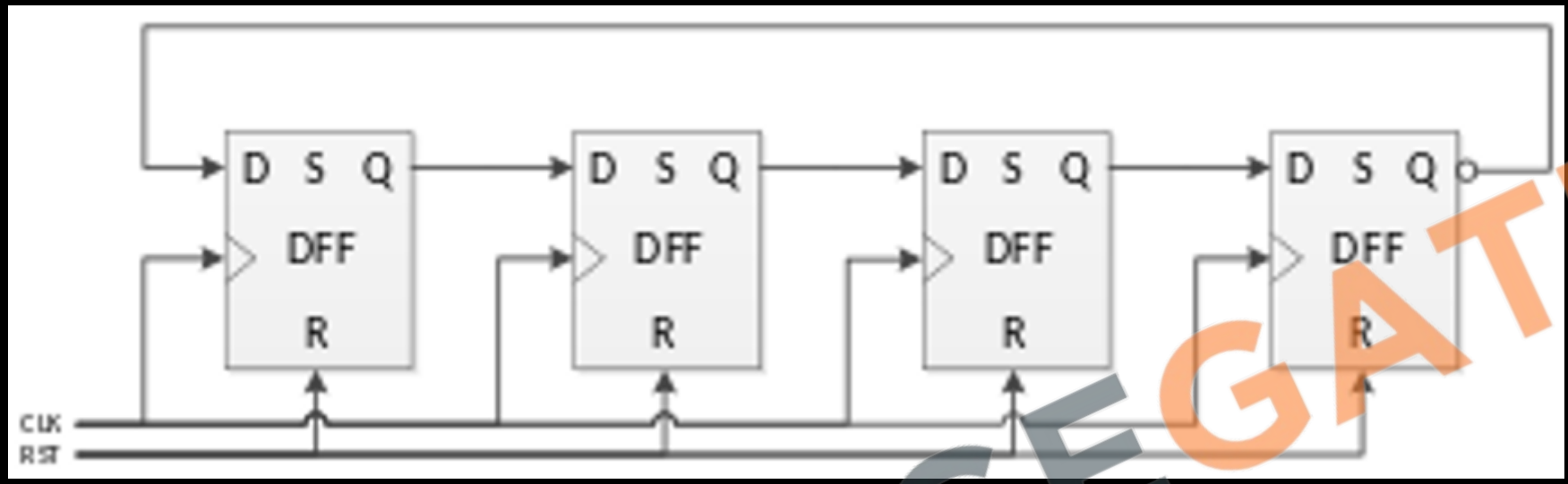


CLK	Q_3	Q_2	Q_1	Q_0
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
0	1	0	0	0

Johnson Counter/ Twisted ring counter/ Switch-tail ring counter/ Walking ring counter

- A k -bit ring counter circulates a single bit among the flip-flops to provide k distinguishable states.
- The number of states can be doubled if the shift register is connected as a switch-tail ring counter. A switch-tail ring counter is a circular shift register with the complemented output of the last flip-flop connected to the input of the first flip-flop.





CLK	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
0	0	0	0	0

Basics

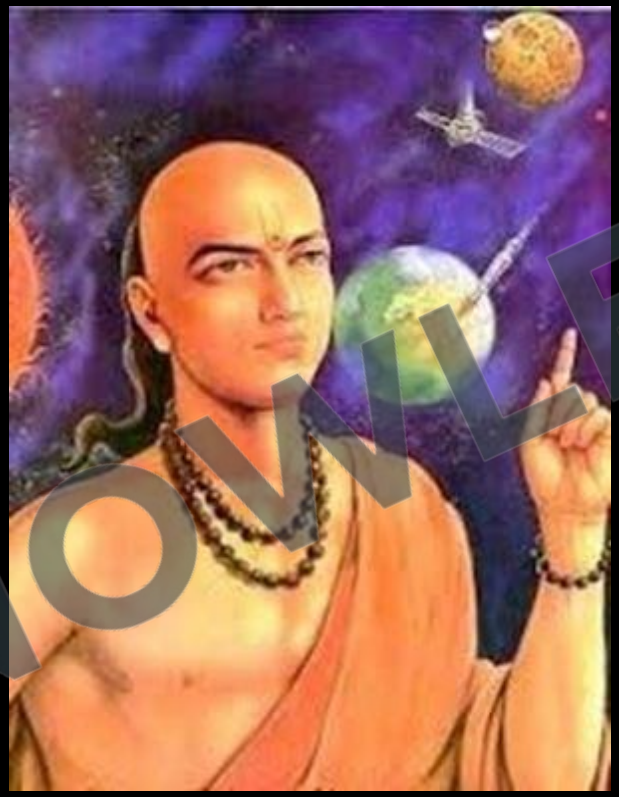
- Main idea in number system is counting and representation of quantity
- In stone age or even today the basic idea of counting in Unary counting
  - Remember how we started counting on fingers and using abacus
  - How can we use it for counting sheep or anything?

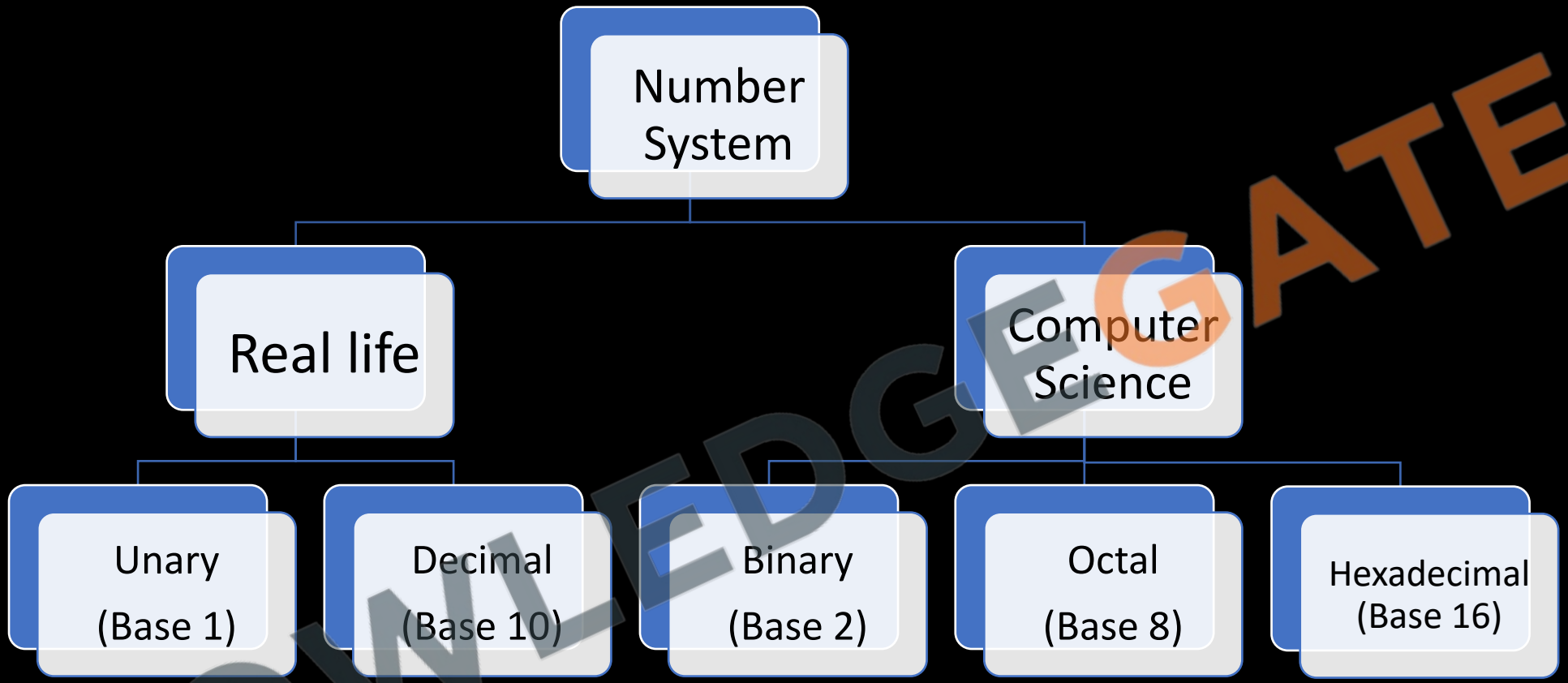
- But when we want to count larger quantity in anything, day to day life, business, maths or research, we cannot work with unary system.

Samsung pays Apple $1 Billion sending 30 trucks full of 5 cent coins

- So next popular system we use in real life is decimal system, may be because we have 10 fingers on over hands, in history different cultures have been using base 10 for general purpose counting, Indian, British, roman, Arabic etc.

- Thought **Aryabhata (3500 years back)** was the first one known to extensively worked on the idea of zero, pie, number system, trigonometry, quadratic equations, astronomy etc.
- In Indian culture we were working on very complex math from much early time, Aryabhata was may be first one, who have properly written and published his understanding, and some of his work even reached today.





Number System	Base or Radix	Digits or symbols
Binary	2	0,1
Octal	8	0,1,2,3,4,5,6,7
Hexadecimal	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Number System	Base or Radix	Digits or symbols
Unary	1	0/1
Decimal	10	0,1,2,3,4,5,6,7,8,9

Decimal Number System

- The Decimal Number System uses 10 different digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. It's called a "base-10" system because it relies on these 10 digits and uses powers of 10 to form numbers. This is the number system we commonly use in everyday activities.
- Example: $(739.2)_{10}$, where 739.2 is a shorthand notation for what should be written as

$$7 * 10^2 + 3 * 10^1 + 9 * 10^0 + 2 * 10^{-1}$$

Binary Number System

- In the Binary Number System, you only use two digits: 0 and 1. Each digit in a binary number is multiplied by a power of 2 (like 2^0, 2^1, 2^2, and so on). Then you add these products together to get the number in decimal form.

Example: $(11010.11)_2$ value in Decimal?

$$1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 + 1 * 2^{-1} + 1 * 2^{-2} = 26.75$$

- Similarly, there can be many bases and they can be easily converted to decimal number system.
- An example of a base-5 number is

$$(4021.2)_5$$

$$4 * 5^3 + 0 * 5^2 + 2 * 5^1 + 1 * 5^0 + 2 * 5^{-1} = (511.4)_{10}$$

Convert from Decimal to Any Base

$(4213.21)_5$ → $(558.44)_{10}$ → $(1425.3036)_7$

$(83)_{10} \longrightarrow (1010011)_2$

$(109.125)_{10} \longrightarrow (1101101.001)_2$

Octal Number System

$$(127.4)_8$$

$$1 * 8^2 + 2 * 8^1 + 7 * 8^0 + 4 * 8^{-1} = (87.5)_{10}$$

Hexadecimal Number System

- When the base of a number system is higher than 10, we use letters from the alphabet to represent the extra digits. For instance, in the Hexadecimal System, which is base-16, the first 10 digits are the same as in the Decimal System: 0 to 9. After that, the letters A, B, C, D, E, and F stand for the numbers 10, 11, 12, 13, 14, and 15 respectively. The hexadecimal system is most popular in digital systems.

$$(B65F)_{16}$$

$$11 * 16^3 + 6 * 16^2 + 5 * 16^1 + 15 * 16^0 = (46687)_{10}$$

- In case a binary number usually becomes so large so better idea is to club digits in a group of three or four, leading to octal, hexadecimal

Hexadecimal System / Base 16 System	
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10-A
1011	11-B
1100	12-C
1101	13-D
1110	14-E
1111	15-F

Octal System / Base 8 System	
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

base 4 System	
00	0
01	1
10	2
11	3

$(\ 110010.111 \ )_2$

$(\_\_\_\_\_.\_\_\_)_4$

$(\ 110010.111 \ )_2$

$(\_\_\_\_.\_)_8$

$(\ 110010.111 \ )_2$

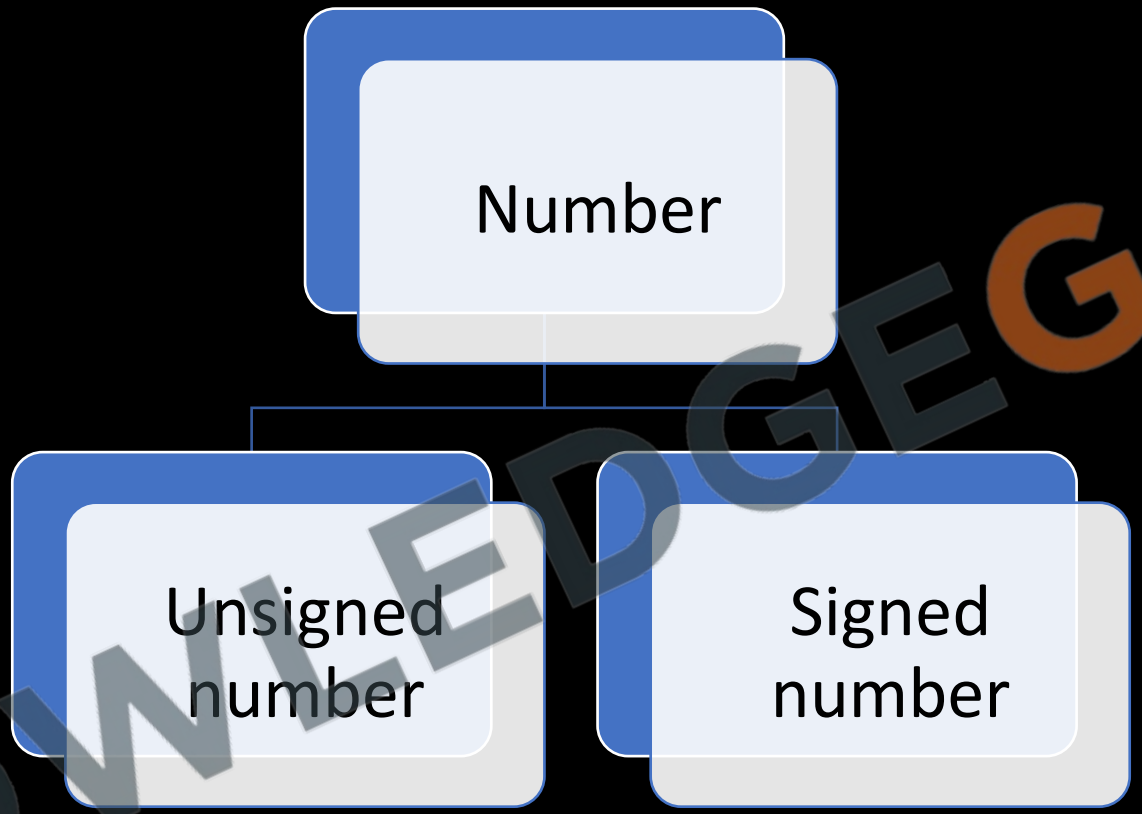
$(\_\_\_\_.\_\_\_)_{16}$

$(62.7)_8$

$(\_\_\_\_\_\cdot\_\_\_\_)_2$

$(\_\_\_\_\cdot\_\_\_)_{16}$

Representation of a number

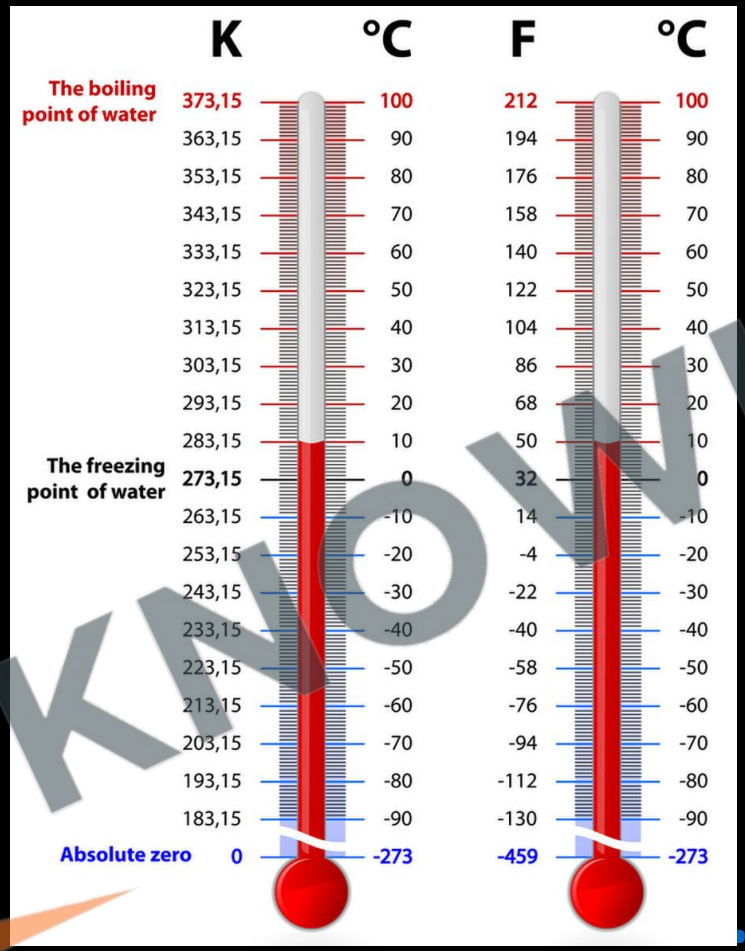


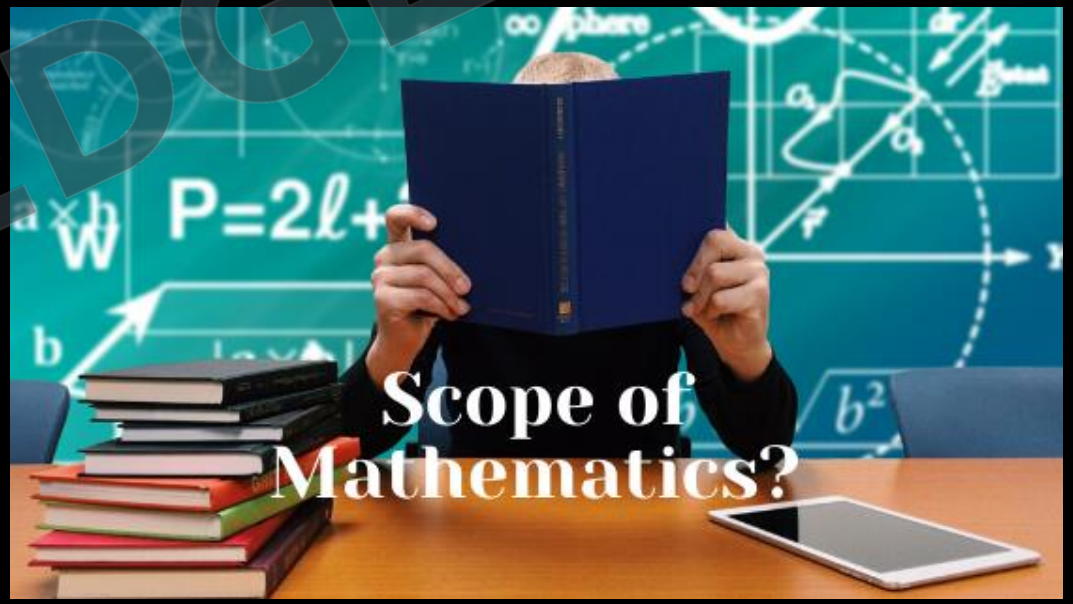
Unsigned number

- Until now, we have considered only unsigned numbers. Unsigned numbers without any positive or negative sign, these numbers represent only magnitude. In real life, we have only unsigned numbers (considered as positive) (natural numbers) for e.g. number of stars, number of trees etc.
- If n bits are used to store the number, then all the n bits are used to store number / magnitude / absolute value. If we have a n bit Unsigned number than range is from 0 to 2^n-1

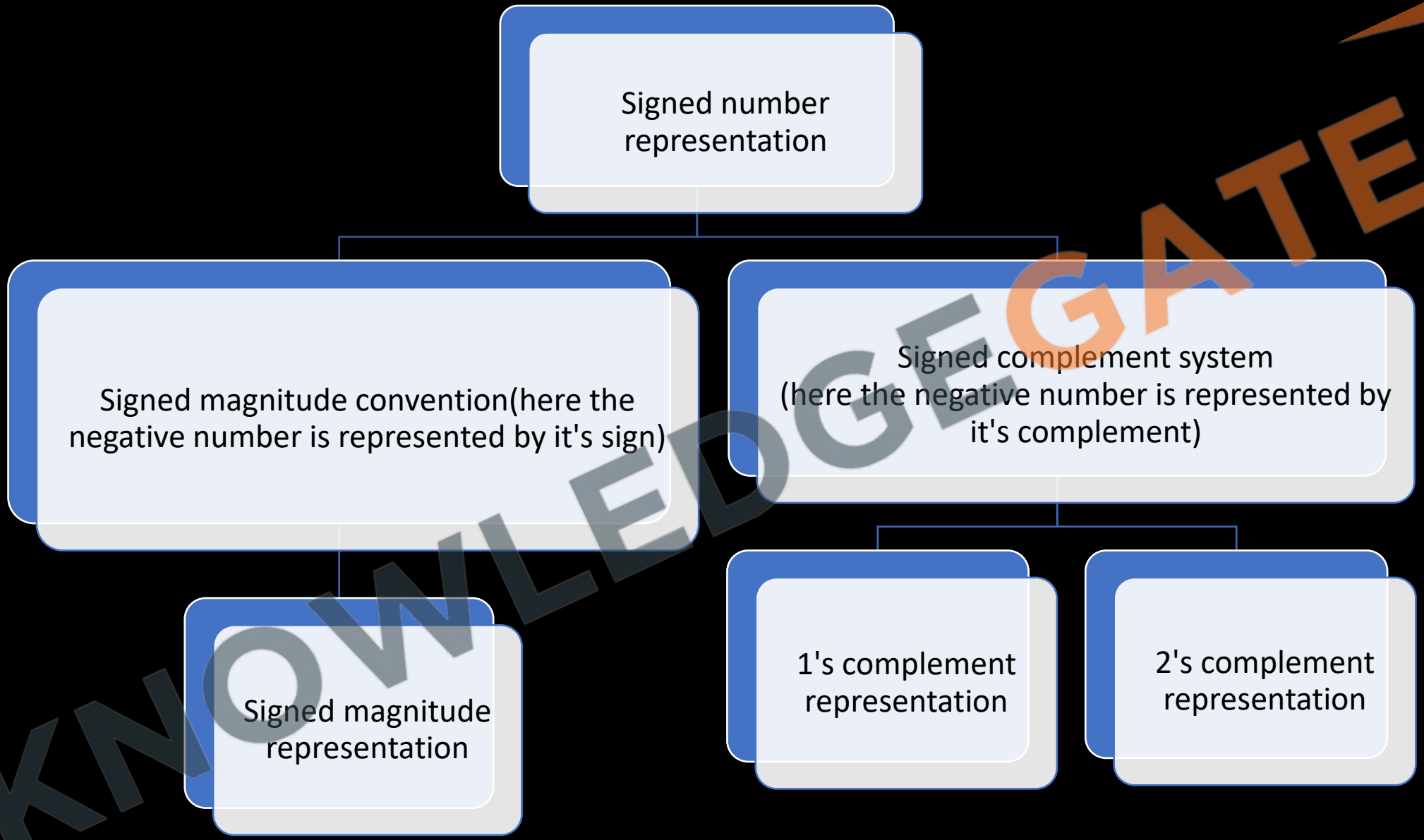
Signed number

- For modern-day math tasks like calculating profit and loss or measuring temperature, we often need to use negative numbers. In daily life, we write these numbers with both their sign and magnitude. For example, +9 shows a positive number and -543 indicates a negative one. We use the symbols "+" and "-" to show if the number is positive or negative, respectively.





- In a computer, an n-bit word can store a signed binary number. One usually the leftmost bit is set aside to show if the number is positive or negative, and the other n-1 bits represent the size or magnitude of the number.
- Because of the limitation of the computer hardware as circuit only understand 0 and 1, every piece of information in computer must be represented in terms of numbers i.e. 0 and 1, weather it is sign or magnitude. where first left most bit 0 means +Ve number and 1 means -Ve number.



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

- Represent decimal number -13 in all three methods?
  - Signed Magnitude
  - 1’s Complement
  - 2’s Complement

- Read number 10101 according to all three representation?
  - Signed Magnitude
  - 1’s Complement
  - 2’s Complement

Sign magnitude representation

- **Introduction: -** It is the simplest form of representation, where in an n-bit word, the right most n-1 bits (from LSB) hold the magnitude of the number and n^{th} bit (Left most, MSB) is assigned for sign, where 0→+ve and 1→-ve.
- **Range: -** from $-(2^{n-1} - 1)$ to $+(2^{n-1} - 1)$ total $2^n - 1$
- **Advantage: -** Easy to read and write, because it is a weighted code system, directly reading and writing is possible.
- **Disadvantage: -** One drawback in sign-magnitude representation is addition and subtraction require separate handling of the sign and the magnitude. One more problem is there are two representation for 0, i.e. -0 & +0, because of which there is a loss of one presentation, and create confusion.
- **Number extension : -** In computer the size of a location is fixed, so it is often required to write a smaller number in a larger space, this can be done by sign extension, where, the magnitude is written as far right as possible, sign bit is written left most, and all the empty cells must be filled/padded with zero.

1’s Complement Representation

- **Introduction: -** here the negative number is represented by taking complement.
- **Range: -** from $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$, total 2^n -1 representation
- **Advantage: -** Easy to calculate, used as logical operations.
- **Disadvantage: -** One drawback in 1’s complement representation is addition and subtraction in difficult, Not a weighted code system. there are two representation for 0, i.e. -0 & +0, because of which there is a loss of one presentation. Neither we can write a negative number directly nor we can read a negative number directly.
- **Number extension:** In computer the size of a location is fixed, so it is often required to write a smaller number in a larger space, this can be done by sign extension, where, if the number is positive extension is done same as that of sign magnitude extension. If the number is -ve then we must write the number as far right as possible and then all the empty cells must be filled with 1(in negative logic 0 is represented by 1).

2's Complement Representation

- **Introduction: -** if 1 is added to 1's complement of a binary number, the resulting number known as the 2's complement of the binary number.
- **Range: -** from $-(2^{n-1})$ to $+(2^{n-1}-1)$, total 2^n representation

Advantage: -

- Easy to do arithmetic operations with 2's complement representation. As end round carry can be discarded. Has only one representation for zero which is always positive, therefore provides better clarity, efficiency and more range with no wastage. It is a waited code system, direct reading and writing is possible.

- **Disadvantage: -** One drawback in 2's complement representation is relatively difficult to understand. But in application like floating point representation it is not much required, and it takes more space.

- **Number extension:** In computer the size of a location is fixed, so it is often required to write a smaller number in a larger space, this can be done by sign extension, where, if the number is positive extension is done same as that of sign magnitude extension. If the number is -ve then we must write the number as far right as possible and then all the empty cells must be filled with 1(in negative logic 0 is represented by 1).

- Subtraction using 2’s Complement arithmetic

- 9-4 =

- 7-11 =

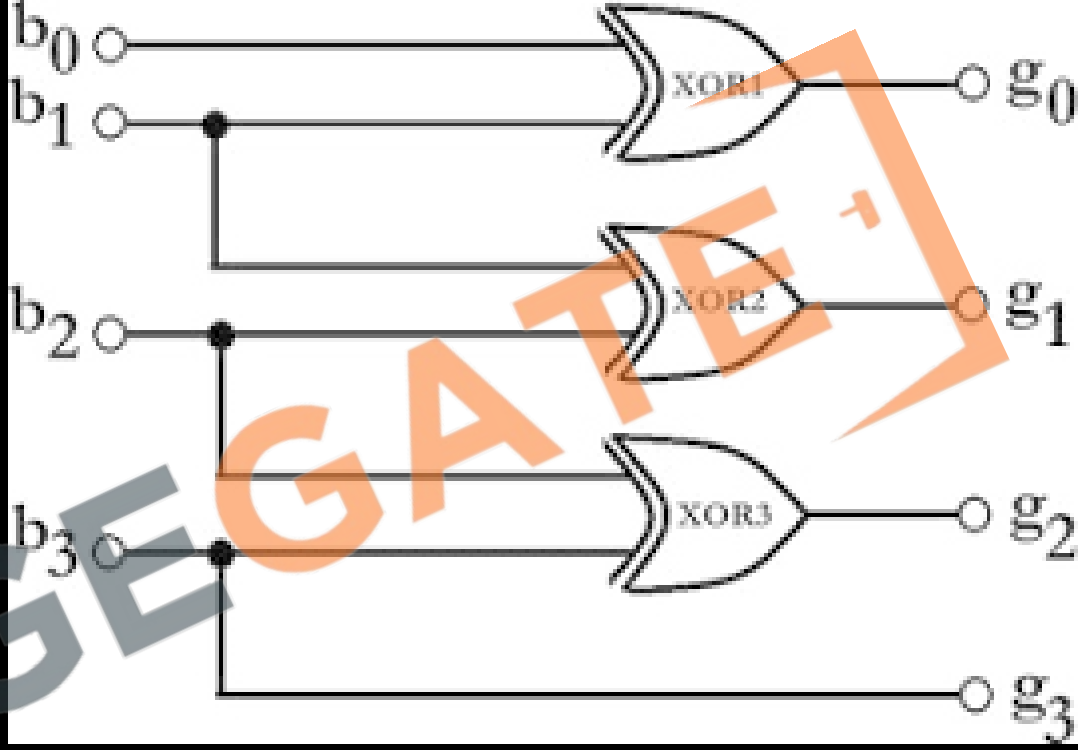
Gray Code

- Gray Code is a binary numbering system where two successive numbers differ in only one bit. It's essential because it reduces errors in digital systems, particularly in analog-to-digital conversions.

- Applications include rotary encoders in mechanical systems and minimizing errors in multi-bit data transfer. Gray Code contributes to system reliability by ensuring that only one bit changes at a time, reducing the risk of transitional errors.

- It's particularly useful in situations where an error in multiple bits changing at once could be catastrophic, like in control systems.

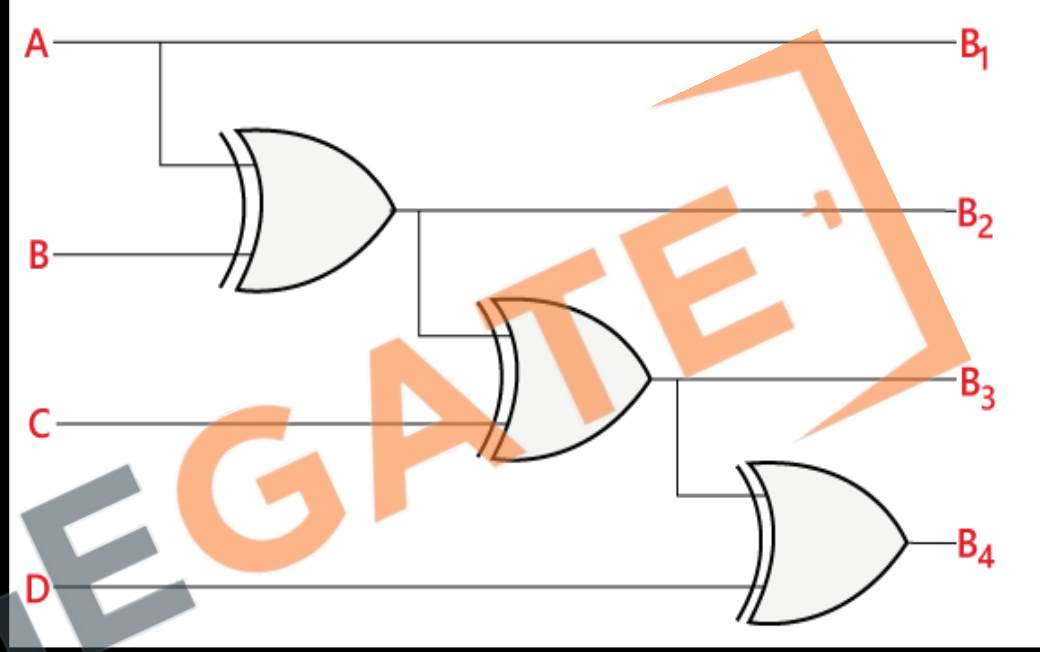
1 1 0 0 1 1 1 0



- $G_0 = B_1 \oplus B_0$
- $G_1 = B_2 \oplus B_1$
- $G_2 = B_3 \oplus B_2$
- $G_3 = B_3$

4-bit binary				4-bit gray			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

1 1 0 0 1 1 1 0



- $B_0 = B_1 \oplus G_0$
- $B_1 = B_2 \oplus G_1$
- $B_2 = B_3 \oplus G_2$
- $B_3 = G_3$

G3	G2	G1	G0	B3	B2	B1	B0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

Binary-Coded Decimal Code (BCD)

- BCD (Binary-Coded Decimal) is a class of binary encodings where each decimal digit is represented by a fixed number of binary digits, usually four. It's needed because it simplifies the process of converting between binary and decimal numbers, making calculations more accurate and easier to check.
- Applications of BCD are widespread, including in digital clocks, calculators, and financial systems where decimal representation is critical for accuracy. It is also useful in systems that require easy conversion to human-readable decimal numbers.
- BCD contributes to computational efficiency and accuracy. It makes it simpler to convert between binary and decimal, reducing the chance of errors that could occur in calculations or data transfers.

Decimal	B_3	B_2	B_1	B_0	D_3	D_2	D_1	D_0	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	1
2	0	0	1	0	0	0	0	0	0	0	1	0
3	0	0	1	1	0	0	0	0	0	0	1	1
4	0	1	0	0	0	0	0	0	0	1	0	0
5	0	1	0	1	0	0	0	0	0	1	0	1
6	0	1	1	0	0	0	0	0	0	1	1	0
7	0	1	1	1	0	0	0	0	0	1	1	1
8	1	0	0	0	0	0	0	0	1	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0	1
10	1	0	1	0	0	0	0	1	0	0	0	0
11	1	0	1	1	0	0	0	1	0	0	0	1
12	1	1	0	0	0	0	0	1	0	0	1	0
13	1	1	0	1	0	0	0	1	0	0	1	1
14	1	1	1	0	0	0	0	1	0	1	0	0
15	1	1	1	1	0	0	0	1	0	1	0	1

- Represent the unsigned decimal number 965 and 672 in BCD and then show the steps necessary to find their sum?

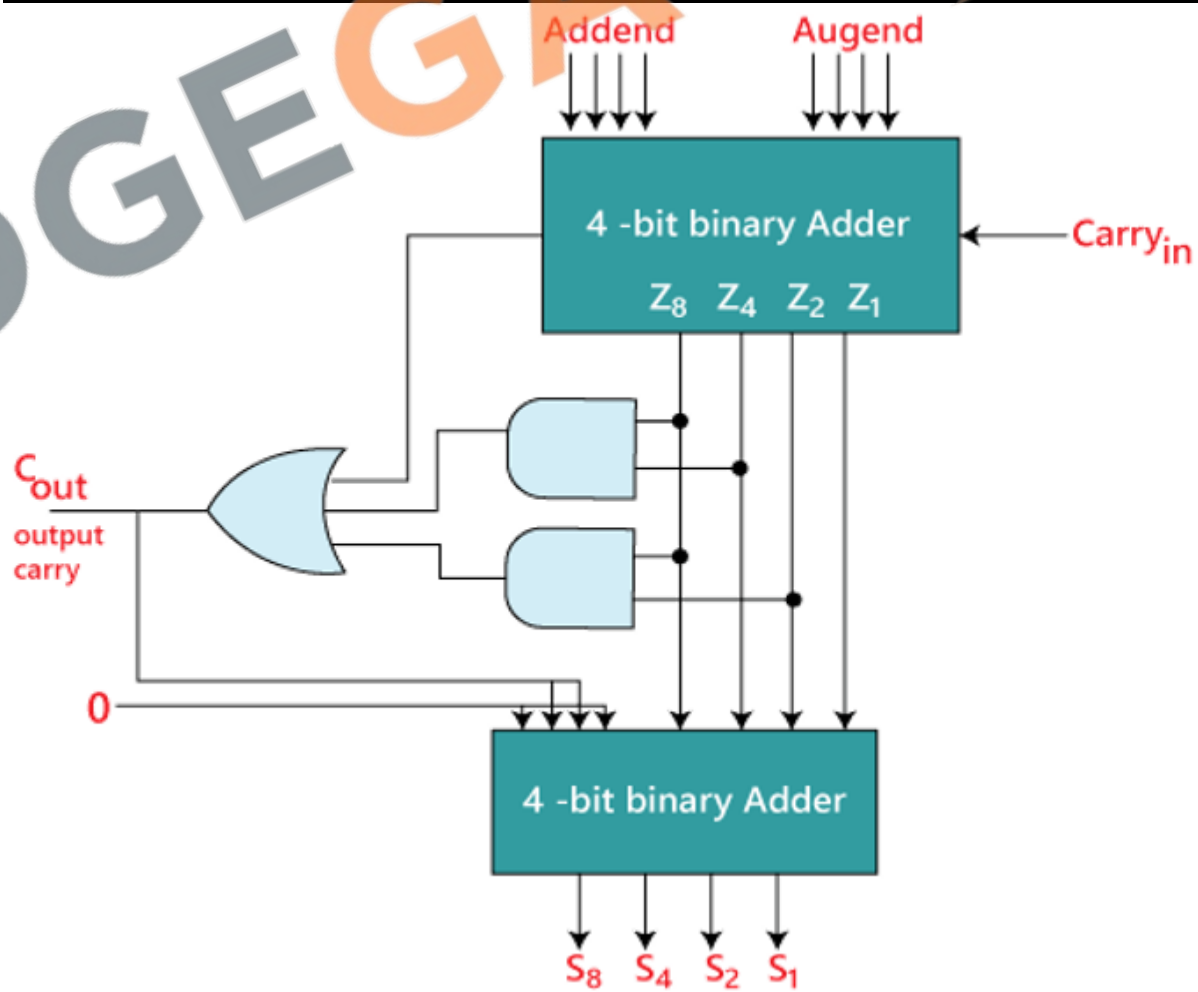
965	1001	0110	0101
672	0110	0111	0010
FD7	1111	1101	0111

FD7	1111	1101	0111
+	0110	0110	0000
0001	0110	0011	0111
1	6	3	7

If sum > 9 then add 6 for correction

Decimal	Binary Sum					BCD Sum				
	C'	S3'	S2'	S1'	S0'	C	S3	S2	S1	S0
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	1
2	0	0	0	1	0	0	0	0	1	0
3	0	0	0	1	1	0	0	0	1	1
4	0	0	1	0	0	0	0	1	0	0
5	0	0	1	0	1	0	0	1	0	1
6	0	0	1	1	0	0	0	1	1	0
7	0	0	1	1	1	0	0	1	1	1
8	0	1	0	0	0	0	1	0	0	0
9	0	1	0	0	1	0	1	0	0	1
10	0	1	0	1	0	1	0	0	0	0
11	0	1	0	1	1	1	0	0	0	1
12	0	1	1	0	0	1	0	0	1	0
13	0	1	1	0	1	1	0	0	1	1
14	0	1	1	1	0	1	0	1	0	0
15	0	1	1	1	1	1	0	1	0	1
16	1	0	0	0	0	1	0	1	1	0
17	1	0	0	0	1	1	0	1	1	1
18	1	0	0	1	0	1	1	0	0	0
19	1	0	0	1	1	1	1	0	0	1

$$C = C' + S'_3S'_2 + S'_3S'_1$$



Excess-3 Code

- Excess-3 Code is a binary-coded decimal (BCD) system that represents each digit of a decimal number by its corresponding 4-bit binary representation, but adds an offset of 3 to each digit. This makes arithmetic operations like addition and subtraction easier to implement.
- Excess-3 Code is primarily used in older computer systems and some types of calculators. It was important for minimizing hardware in early digital systems, where computational resources were limited.
- The main contribution of Excess-3 Code is that it simplifies the hardware required for arithmetic operations. Its self-complementing feature helps in carrying out subtraction by addition, thereby reducing computational complexity.

Decimal	Excess-3
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

- Example: 395 in decimal will be represented in excess-3 code as:

  0110 1100 1000.

  3 9 5

- 3 in excess-3 representation is: 3 + 3 = 6 (0110)
- 9 in excess-3 representation is: 9 + 3 = 12 (1100)
- 5 in excess-3 representation is: 5 + 3 = 8 (1000)

- First, convert the decimal digits to their 4-bit Excess-3 codes:
  - 5 in Excess-3: 5 + 3 = 8 in decimal, which is 1000 in binary.
  - 7 in Excess-3: 7 + 3 = 10 in decimal, which is 1010 in binary.
- Add the Excess-3 coded numbers:
  - 1000 + 1010 = 10010 in binary, which is 18 in decimal.
- Now, subtract 3 from the sum to remove the excess and get the real sum:
  - 18 − 3 = 15
  - 15 in Excess-3 code, real sum is 12