FUNDAMENTALS OF DIGITAL ELECTRONICS

FIRST SEMESTER

CORE PAPER I - FUNDAMENTALS OF DIGITAL ELECTRONICS

UNIT I: Digital Computers and Digital Systems. **Number Systems & Codes:** Number System - Base Conversion - Binary Codes - Code Conversion. **Digital Logic:** Logic Gates - Truth Tables - Universal Gates.

UNIT II: Boolean algebra: Laws & Theorems - SOP, POS Methods - Simplification of Boolean Functions using theorems – Simplification of Boolean Functions using K-Map (Two, Three and Four variables).

UNIT III: Binary Arithmetic: Binary Addition –Binary Subtraction - Arithmetic Building Blocks. **Adders:** Half Adder and Full Adder. **Subtractors:** Half Subtractor and Full Subtractor. **Combinational Logic:** Multiplexers - Demultiplexers - Decoders – Encoders.

UNIT IV: Sequential Logic: RS, JK, D and T Flip-Flops. **Registers:** Shift Registers - Types of Shift Registers – Implementation of Serial-In Serial-Out Shift Register and Serial-In Parallel-Out Shift Register.

UNIT V: Counters: Asynchronous Counters Ripple, Mod, Up-Down Counters- Synchronous Counters - Types of ROM and RAM.

TEXT BOOK:

1. **V.Rajaraman and T.Radhakrishnan,** *"Digital Computer Design"*, Fifth Edition, 2012, Prentice Hall of India.

REFERENCE BOOKS:

1. **D.P.Leach and A.P.Malvino,** *"Digital Principles and Applications"*, Seventh Edition, 2011,TMH.
2. **T.C.Bartee,** *"Digital Computer Fundamentals"*, Sixth Edition, Tata McGraw Hill.
3. **Floyd and Jain,** *"Digital Fundamentals"*, Ninth Edition, Pearson Education.

E-REFERENCES:

1. http://nptel.iitm.ac.in/video.php?subjectId=117106086
2. http://nptel.iitm.ac.in/Onlinecourses/Srinivasan/

DIGITAL ELECTRONICS LAB :-

DIGITAL ELECTRONICS LAB

1. Verification of Truth Table for AND, OR, NOT, NAND, NOR and EX-OR gates
2. Realization of NOT, AND, OR, EX-OR gates using NAND gate.
3. Realization of NOT, AND, OR, EX-OR gates using NOR gate.
4. Karnaugh Map Reduction and Logic Circuit Implementation.
5. Verification of DeMorgan's Law
6. Verification of Associative Law
7. Verification of Distributive Law.
8. Implementation of Half-Adder and Full-Adder.
9. Implementation of Half-Subtractor and Full-Subtractor.
10. Four Bit Binary Adder
11. Four Bit Binary Subtractor
12. Decimal adder (BCD)
13. Verification of Characteristic Table of [redacted]ous flipflops
14. Design of Shift registers
15. Design of Counters

UNIT - I

DIGITAL COMPUTER AND DIGITAL SYSTEM -

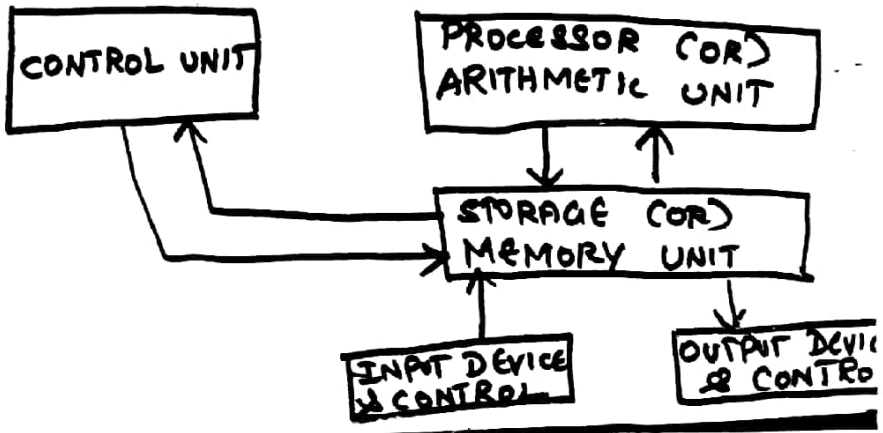
* computers are used for scientific calculations, commercial and business data processing, air-traffic control, space guide, educational field and many other areas.
* Digital computers made possible in many commercial and scientific advancements.
* Digital computers are in a flow of sequencing the instructions called program, that operates on a given data.
* The user can change program/data according to specific need. so that it can perform information-processing tasks.
* The digital computer includes telephone switching ex-changes, digital voltmeters, calculating machines.
* Discrete elements may be electric impulses, decimal digits, the letters of an alphabet, arithmetic operations, punctuation marks and other symbols

  Ex: The letters G, O, D form GOD

  2, 3, 7 for a number.

* Sequence of Discrete elements forms a language (ie) it conveys an information.
* Early digital computers used with Numerical computations and used only with numbers so it has been emerged with digital computers
* Discrete elements of information are represented in a digital system by physical quantities called signals.
* voltages and current are most common electrical signals.
* It has two discrete values called as binary. (0, 1)
* Analog computers perform a direct simulation of a physical system. (Mathematically by differential equation)
* It operates with continuous variables.

BLOCK DIAGRAM OF DIGITAL COMPUTER :-



(3)

The control unit supervises the flow of Information between various units. The control unit retrieves the instructions one by one from the program which is stored in a memory. The control unit supervises the program instructions and processor manipulates the data specified by the program.

The memory unit stores programs as well as Input / output and the data.

The processor unit performs arithmetic and data processing task specified by the program.

The Input and output devices are specified as special in digital systems and controlled by electronic digital circuits. The I/P devices are punch card reader and O/P devices are printers.

A digital computer is a interconnection of digital modules which is necessary to have a basic knowledge of digital system and general behaviour.

NUMBER SYSTEM:-

Decimal Number System:-

The decimal number system makes use of ten digits namely 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. The decimal number system have a base or radix of 10. All the digits in decimal number system are expressed in powers of 10 like 10^0, 10^1, 10^2.... The quantities 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5} are called weights. The Integer part and fractional part are separated by a decimal point.

Binary Number system:

A binary number system uses only two symbols 0 and 1. It has a base or a radix of 2. A binary digit 0 or 1 is called as bit. All the bits have the power of 2 like 2^0, 2^1, 2^2 etc.. Bit is an abbreviation of binary digit.

→ 4 bit = 1 nibble
→ 8 bit = 1 byte
→ 16 bit = word
→ 32 bit = Double word.

Decimal to Binary conversion:-

* Popular method used is 'double dabble'.
* In this, divide the decimal number by 2, writing down the remainder after each division.
* The remainders take in reverse order (down to top) to form binary numbers.

Ex: $(19)_{10} \rightarrow (?)_2$

MSB → Most significant bit
LSB → Least " "

2	19	
2	9	1
2	4	1
2	2	0
	1	0

LSB ↑ MSB

$(10011)_2$

(5) 2) $(107)_{10} \rightarrow (1101011)_2$

3) $(52)_{10} \rightarrow (110100)_2$

4) $(19.625)_{10} \rightarrow (?)_2$

2 | 19	
2 | 9 - 1	$0.625 \times 2 = 1.250 \rightarrow$ carry is 1
2 | 4 - 1	$0.250 \times 2 = 0.500 \rightarrow$ carry is 0
2 | 2 - 0	$0.500 \times 2 = 1.000 \rightarrow$ carry is 1
1 - 0	

(MSB) (LSB)

$(10011)_2$

$\therefore$ $(10011.101)_2$.

6) $(125)_{10} \rightarrow (1111101)_2$

7) $(34.5)_{10} \rightarrow (100010.11)_2$

8) $(48.125)_{10} \rightarrow (110000.001)_2$

* The fractional part has to be multiplied by 2 by collecting the carries from top to bottom.
* The multiplication can be repeated till the fractional part becomes zero.
* If the fractional part is not zero after 4 or 5 steps, it can be stopped at 4th step and take nearest value.

5) $(13.7)_{10} \rightarrow (?)_2$

2 | 13	
2 | 6 - 1	$0.7 \times 2 = 1.4 \rightarrow 1$
2 | 3 - 0	$0.4 \times 2 = 0.8 \rightarrow 0$
1 - 1	$0.8 \times 2 = 1.6 \rightarrow 1$
	$0.6 \times 2 = 1.2 \rightarrow 1$

$(1101.1011)_2$

Binary to Decimal Conversion :-

Ex $(11011)_2 \rightarrow (?)_{10}$

1)

1 1 0 1 1	
1	$1 \times 2^0 = 1$
1	$1 \times 2^1 = 2$
0	$0 \times 2^2 = 0$
1	$1 \times 2^3 = 8$
1	$1 \times 2^4 = 16$
	27

(OR)

$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$16 + 8 + 0 + 2 + 1 = 27$

$(27)_{10}$

2) $(1011101.101)_2 \rightarrow (?)_{10}$

1 0 1 1 1 0 1	
1	$1 \times 2^0 = 1$
0	$0 \times 2^1 = 0$
1	$1 \times 2^2 = 4$
1	$1 \times 2^3 = 8$
1	$1 \times 2^4 = 16$
0	$0 \times 2^5 = 0$
1	$1 \times 2^6 = 32$
	45

$2^{-1} = \frac{1}{2}$

$2^{-2} = \frac{1}{2^2} = \frac{1}{4}$

$2^{-3} = \frac{1}{2^3} = \frac{1}{8}$

.101	
1	$1 \times 2^{-3} = 0.125$
0	$0 \times 2^{-2} = 0$
1	$1 \times 2^{-1} = 0.5$

(ie) $0.5 + 0 + 0.125$

$= 0.625$

$(45.625)_{10}$

3) $(1011010)_2 \rightarrow (90)_{10}$

4) $(111.11)_2 \rightarrow (7.75)_{10}$

5) $(1101.1011)_2 \rightarrow (13.6875)_{10}$

6) $(11101)_2 \rightarrow (29)_{10}$.

Steps

1. write Binary number
2. Directly write $2^0, 2^1, 2^2 \ldots$ from right to left for integers
3. write $2^{-1}, 2^{-2}, 2^{-3}$ from left to right for fractions
4. Add the weights.

OCTAL NUMBER SYSTEM :-

The number system with base 8 is known as octal. In this eight symbols 0, 1, 2, 3, 4, 5, 6, 7. As in decimal & binary, it also has Integer and fractional Part. Since $2^3 = 8$. It is a group of 3-bit binary sequence.

Decimal	octal	Binary $2^2+2^1+2^0$
0	0	000
1	1	001
2	2	010
3	3	011
4	4	100
5	5	101
6	6	110
7	7	111

Decimal to octal conversion: — (Octal dabble method) To convert a decimal number to octal, divide the number by 8, and collect the remainders from bottom to top.

Ex: 1) $(68)_{10} \rightarrow (?)_8$

8 | 68
8 | 8 - 4 ↑
1 - 0

$(104)_8$

2) $(109)_{10} \rightarrow (?)_8$

8 | 109
8 | 13 - 5 ↑
1 - 5

$(155)_8$

3) $(98.625)_{10} \rightarrow (?)_8$

8 | 98
8 | 12 - 2 ↑
1 - 4

$(142)_8$

$0.625 \times 8 = 5.000 \rightarrow$ carry is 5

$(142.5)_8$

4) $(18.58)_{10} \rightarrow (?)_8$

8 | 18
2 - 2 ↑
$(22)_8$

$0.58 \times 8 = 4.64 \rightarrow 4$
$0.64 \times 8 = 5.12 \rightarrow 5$
$0.12 \times 8 = 0.96 \rightarrow 0$

$(22.450)_8$

5) $(0.68)_{10} \rightarrow (?)_8$

$0.68 \times 8 = 5.44 \rightarrow 5$
$0.44 \times 8 = 3.52 \rightarrow 3$
$0.52 \times 8 = 4.16 \rightarrow 4$
$0.16 \times 8 = 1.28 \rightarrow 1$

$(0.5341)_8$

6) $(186)_{10} \rightarrow (272)_8$

8 | 186
8 | 23 - 2 ↑
2 - 7

(272)

7) $(19.5)_{10} \rightarrow (23.4)_8$

8) $(49.625)_{10} \rightarrow (19.79)_8$

$(61.5)_8$

(6) Octal to Decimal Conversions :-

In the octal number system, each digit position corresponds to a power of 8.

$$8^3\ 8^2\ 8^1\ 8^0 . 8^{-1}\ 8^{-2}\ 8^{-3}$$

Ex: $(362)_8 \rightarrow (?)_{10}$

1)

3 6 2

- $2 \times 8^0 = 2$
- $6 \times 8^1 = 48$
- $3 \times 8^2 = 192$

242

$64 \times 3 = 192$

$(242)_{10}$

2) $(0.763)_8 \rightarrow (?)_{10}$

0 . 7 6 3

- $3 \times 8^{-3} = 3 \times \frac{1}{8} = 0.375$
- $6 \times 8^{-2} = 0.093$
- $7 \times 8^{-1} = 0.0058$

$(0.9738)_{10}$

0.125
8 | 10
8
20
16
40
40

0.125×3
0.375

3) $(6327.4051)_8 \rightarrow (?)_{10}$

6327

- $7 \times 8^0 = 7$
- $2 \times 8^1 = 16$
- $3 \times 8^2 = 192$
- $6 \times 8^3 = 3072$

3286

. 4 0 5 1

- $1 \times 8^{-4} = \frac{1}{4096}$
- $5 \times 8^{-3} = \frac{5}{512}$
- $0 \times 8^{-2} = 0$
- $4 \times 8^{-1} = \frac{4}{8}$

$= 5099$

$(3287.5099)_{10}$

4) $(23.625)_8 \rightarrow (?)_{10}$

2 3

- $3 \times 8^0 = 3$
- $2 \times 8^1 = 16$

19

6 2 5

- $5 \times 8^{-3} = 0.00097$
- $2 \times 8^{-2} = 0.03$
- $6 \times 8^{-1} = 0.75$

0.78097

$5 \times \frac{1}{8^3}$

$\frac{5}{512} =$

0.00097

512 | 5000
4608
3920
3584

0.00097
0.03
0.75
0.78097

$(19.79)_{10}$

5) $(36.77)_8 \rightarrow (?)_{10}$

3 6

- $6 \times 8^0 = 6$
- $3 \times 8^1 = 24$

30

7 7

- $7 \times 8^{-2} = \frac{7}{64}$
- $7 \times 8^{-1} = 0.109$

0.875

0.984

$(30.984)_{10}$

6) $(12.6)_8 \rightarrow (?)_{10}$

1 2

- $2 \times 8^0 = 2$
- $1 \times 8^1 = 8$

10

$.6 \times 8^{-1} = 0.75$

$6 \times \frac{1}{8} = \frac{6}{8} =$

$(10.75)_{10}$

HEXADECIMAL NUMBER SYSTEM:-

The computers requires a large number of bits to represent biggest numbers. It has a base of 16 and requires 16 distinct symbols to represent the numbers. These are numerals 0 to 9 and alphabets A to F. As numbers and alphabets are used, this system is also known as 'Alphanumeric number system'. Since $2^4 = 16$, it follows four bit binary sequence.

Hexa Decimal	Hexadecimal	Binary $2^3+2^2+2^1+2^0$ $8+4+2+1$
0	0000	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Decimal to hexadecimal conversion:

The method used is "Hex dabble". This method is to divide successively by 16 and writing the remainders from down to top.

Ex

1) $(256)_{10} \rightarrow (?)_{16}$

16 | 256
16 | 16 - 0
1 - 0

$(100)_{16}$

2) $(1397)_{10} \rightarrow (?)_{16}$

16 | 1397
16 | 87 - 5
5 - 7

$(575)_{16}$

3) $(1020)_{10} \rightarrow (?)_{16}$

16 | 1020
16 | 63 - 12
3 - 15

12 = C, 15 = F

$\therefore (3FC)_{16}$

4) $(213)_{10} \rightarrow (?)_{16}$

16 | 213
13 - 5

13 = D

$(D5)_{16}$

5) $(98.625)_{10} \rightarrow (?)_{16}$

16 | 98
6 - 2

$(62)_{16}$

$0.625 \times 16 = 10.000$
A

$(62.A)_{16}$

E) $(39.5)_{10} \rightarrow (?)_{16}$

16 | 39
2 – 7

$(27)_{16}$

$.5 \times 16 = 8.0$

0.5 × 16 = 30, 05 / 080

$(27.8)_{16}$

F) $(45.65)_{10} \rightarrow (?)_{16}$

16 | 45
2 – 13 (13 = D)

$(2D)_{16}$

$.65 \times 16 = 10.4 \rightarrow$ carry 10 (A)
$0.40 \times 16 = 6.4 \rightarrow$ carry 6
$0.40 \times 16 = 6.4 \rightarrow$ carry 6

$(2D.A66)_{16}$

8) $(88.525)_{10} \rightarrow (58.866)_{16}$.

Heradecimal to decimal conversion :-

In this case the weights used are

$16^3\ 16^2\ 16^1\ 16^0 . 16^{-1}\ 16^{-2}\ 16^{-3}$.

Ex 1) $(D5)_{16} \rightarrow (?)_{10}$

D 5
- $5 \times 16^0 = 5$
- $13 \times 16^1 = 208$

213

$16^1 = 16$
$16^2 = 256$
$16^3 = 4096$
$16^4 = 65,536$.

$(213)_{10}$

2) $(E9)_{16} \rightarrow (?)_{10}$

E 9
- $9 \times 16^0 = 9$
- $14 \times 16^1 = 224$

233

$(233)_{10}$

3) $(FFFF)_{16} \rightarrow (?)_{10}$

15 15 15 15
- $15 \times 16^0 = 15$
- $15 \times 16^1 = 240$
- $15 \times 16^2 = 3840$
- $15 \times 16^3 = 61440$

65,535

$(65,535)_{10}$

4) $(3FC.8)_H \rightarrow (?)_{10}$

3 15 12
- $12 \times 16^0 = 12$
- $15 \times 16^1 = 240$
- $3 \times 16^2 = 768$

1020

$0.8 \times 16^{-1} = 0.5$

8 / 16 = 0.5

$\therefore (1020.5)_{10}$

5) $(9A.A)_{16} \rightarrow (?)_{10}$

9 10
- $10 \times 16^0 = 10$
- $9 \times 16^1 = 144$

154

$0.10 \times 16^{-1} = 0.625$

$(154.625)_{10}$.

6) $(56.48)_{16} \rightarrow (?)_{10}$

5 6
- $6 \times 16^0 = 6$
- $5 \times 16^1 = 80$

86

48
- $8 \times 16^{-2} = 0.0312$
- $4 \times 16^{-1} = 0.25$

0.2812

$(86.2812)_{10}$.

Octal to Binary conversion :-

TO convert an octal to binary, we have to do & replace each octal digit with its equivalent 3-bit binary. The weights required are $2^2, 2^1, 2^0$ (4, 2, 1)

(3 5)$_8$ = (001 011 . 101)$_2$

(543)$_8$ = (101 100 011)$_2$

octal	Binary	Decimal
$(27)_8$	$(010\ \ 111)_2$	23
$(135)_8$	$(001\ \ 011\ 101)_2$	93
$(45.5)_8$	$(100\ \ 101 \cdot 101)_2$	37.625

$27 \rightarrow 7 \times 8^0 = 7$, $2 \times 8^1 = 16$, $(23)_{10}$

8 | 23, 2 — 7, $(27)_8$

Binary to octal conversion :-

TO convert a binary number to octal, we have to arrange the bits into group of 3 bits starting from LSB. If the final group has less than 3 bits, just include zeros to make it group of 3 bits.

Ex: $(10101)_2 \rightarrow (?)_8$

$(10101)_2 = (010101)_2 = (25)_8$

* For integer part, 3 bits groups are formed starting from point and moving towards left.
* For fractional part, from point and moving towards right.

2) $(110.101)_2 \rightarrow (?)_8$

$\left(\frac{110}{6} \cdot \frac{101}{5}\right) = (6.5)_8$

3) $(010\ 111 \cdot 100)_2 = (27.4)_8$

4) $(11.1)_2 \rightarrow (?)_8$

$(011 \cdot 100)_2 = (3.4)_8$

5) (1010110.01)

001 010 110 . 010

$(126.2)_8$

Hexadecimal to Binary Conversion :-

TO convert a hexadecimal to binary, we have to do & replace each hex digit with equivalent 4-bit binary.

Ex $(25)_{16} \rightarrow (?)_2$ $(0010\ 0101)_2$

2) $(3A)_{16} \rightarrow (?)_2$ $(0011\ 1010)_2$

3) $(3A.7)_{16} \rightarrow (?)_2$ $(0011 \cdot 1010 \cdot 0111)_2$

4) $(CD.E8)_{16} \rightarrow (?)$ C=12, D=13, E=14.

$(1100\ 1101 \cdot 1110\ 1000)_2$

5) $(EF.E6)_{16} \rightarrow (?)_2$ $(1111\ 1111 \cdot 1110\ 0110)_2$

Binary to Hexadecimal Conversion :-

TO convert a binary number to hexadecimal, arrange the bits into group of 4 bits from LSB. If it is less than 4 bits just include zeros.

Ex 1) $(100101)_2 = (?)_{16}$ $(0010\ 0101)_2 = (25)_{16}$

2) $(1010 \cdot 1101)_2 = (AD)_{16}$

3) $(110 \cdot 101)_2 = (?)_{16}$ $0110\ 1010 = (6A)_{16}$

4) $(1110 \cdot 11)_2$ $(1110 \cdot 1100)_2 = (E.C)_{16}$

5) $(1101111 \cdot 1)_2$ $0111\ 0111 \cdot 1000$ $(77 \cdot 8)_{16}$

BINARY ARITHMETIC

We know how to perform arithmetic operations like addition, subtraction.. in decimal system. Now we have to know in binary number system which help us to understand arithmetic digital circuits like adders, subtractors etc..

Binary addition :-

The addition of two binary numbers is very much similar to addition of two decimal numbers. The following rules are followed while adding two binary numbers.

0+0	= 0
0+1	= 1
1+0	= 1
1+1	= 10 (Read as 0 with a carry 1)
1+1+1	= 11 (Read as 1 with carry 1)

(1+1 is 2 which is 10)

Ex 1) Add 3 and 4

Solution : Apply the rules of binary addition.

Augend 3 — 011
Addend 4 — 100
Σ = 111

2) Add 5 and 6

Augend 5 — 101
Addend 6 — 110
carry 1 | 011

(1011)$_{BCD}$ = 576
8 2 1 = 11

3) Add 13 and 7.

1 1 (carry)
1101
111
1|0100

2 | 20
2 | 10 - 0
2 | 5 - 0
2 | 2 - 1
1 - 0

4) Add 10 and 11

10 — 2
11 — 3
$(101)_2$ $(5)_{10}$

5)

Decimal	Binary
7	111
+ 4	100
11	1011

6)

Binary	Decimal
1111	15
10100	20
100011	35

when the decimal numbers are.. large, convert it to binary convert decimal to hexa and hexa to binary.

7)

Decimal	Binary
4.25	0.25×2 = 0.50 → 0
7.75	0.50×2 = 1.00 → 1
12.00	

4 → 0100
7 → 0111

0.25×2 = 0.50
0.50×2 = 1.00

(01 0.75×2 = 1.50
0.50×2 = 1.00
$)_2$

$(4.25)_{10} = (0100.01)_2$

$(7.75)_{10} = (0111.11)_2$

(+) $(1100.00)_2 = (12)_{10}$

1 1 1 1
0100 . 01
0111 . 11
1100 . 00

Hexadecimal addition :-

1) Add 94 and 125

16 | 94
5 - E ↑ $(5E)_{16}$

16 | 125
7 - 13 ↑
$(7D)_{16}$.

16 | 219
13 - 11 ↑
$(DB)_{16}$

$(94)_{10} = 5E = 0101\ 1110$ → augend
$(125)_{10} = 7D = 0111\ 1101$ → addend

$(219)_{10} = (DB)_{16} = 1101\ 1011 \Rightarrow (DB)_{16}$

2) Add 5.125 and 7.75

$(5.125)_{10} = (0101.0001\ 0010\ 0101)$ → augend
$(7.75)_{10} = (0111.0111\ 0101)$ → Addend
12.875
$1100.1000\ 0111\ 0101$
$(12 \cdot 8\ 7\ 5)$

Binary subtraction :-

The following rules are used in subtracting two binary numbers.

0 - 0 = 0	
0 - 1 = 11	(Read as difference 1 with a borrow 1)
1 - 0 = 1	
1 - 1 = 0	

Example:

1) Subtract 5 from 7

7 - Minuend
5 - Subtrahend
2

111
101
010 = 2.

2) subtract 3 from 13

13 - Minuend
3 - Subtrahend
10

0 10
1101
0011
1010 = 10.

3) Subtract 14 from 17.

17
14
3

$^{1}1\ ^{1}0\ ^{1}0\ ^{1}0\ ^{1}1$
1110
00011 = 3.

Binary Multiplication:

The following rules are used to multiply two binary numbers.

0 × 0 = 0
0 × 1 = 0
1 × 0 = 0
1 × 1 = 1

Multiplication of binary is same as that of decimal.

Ex 1) Multiply 7 and 3

7 x → multiplicand
3 → Multiplier
21

```
      1 1 1   x
        0 1 1
    ---------
      1 1 1
    1 1 1
  0 0 0
---------------
1  0  1  0  1
2⁴ 2³ 2² 2¹ 2⁰
16  4  1
```

$= (21)_{10}$.

2) Multiply 13 and 11

13 x
11
143

```
    1 1 0 1
      1 0 1 1
    ---------
    1 1 0 1
  1 1 0 1
0 0 0 0
1 1 0 1
-----------------
1 0 0 0 1  1 1 1
```

$= (8F)_H = (143)_{10}$

8 F

16 x 8 = 128

16 | 143
8 — 15

Binary Division:-

Binary division has only two rules

$0 \div 1 = 0$
$1 \div 1 = 1$

Example: Divide 6 by 3

```
      2
Divisor 3 | 6 → Dividend
          6
          0
```



```
      10 → (2)
0 11 | 110
       11
       --
       00
```

2) Divided 10 by 4

```
   2.5
4 | 10
    8
    --
    20
    20
    --
     0
```

```
    10.1
100 | 1010
      100
      ---
      100
      100
      ---
        0
```

$(2.5)_{10}$

1's and 2's complement:

1's complement:

The 1's complement of the bit is defined as 1 minus that bit.

(ie) 1's complement of 1 is $1 - 1 = 0$
1's complement of 0 is $1 - 0 = 1$

It is easy to say that 1's complement of 1 is 0 and 0 is 1.

Ex: 1011 → 0100
100100 → 011011
10101101 → 01010010

2's complement:-

2's complement is formed by adding 1 to the 1's complement.

Ex Find 2's complement of 1010

1's complement of 1010 = 0101

(+) 1 = 1

0110

2's complement of 1010 = 0110

9's complement and 10's complement :-

9's → It is obtained by subtracting the decimal number from 9. For more number subtract each digit from 9.

Ex 9's complement of 4 is 9-4 = 5

9's complement of 7 is 9-7 = 2

9's complement of 123 is 999-123 = 876

9's complement of 5647 is 9999-5647 = 4352

10's complement → It is obtained by adding 1 to the 9's complement.

Ex 10's complement of 4 is 9-4 = 5+1 = 6.

10's complement of 25 is 99-25 = 74+1 = 75

10's complement of 123 is 999-123 = 876+1 = 877

Subtraction using complements :-

1's complement subtraction :-

Steps:

1) Take the minuend in binary form as it is.

2) Take 1's complement of subtrahend and add it to the minuend.

3) If a carry is produced add the carry to the sum.

Ex : Subtract 4 from 13.

4 → 0100 → Subtrahend. 13 → 1101

13 → 1101 → Minuend 4 → 1011

1's comp

0100 = 1011

1 | 1000

1001 → 9

(→ 13, 4, 9)

2) Subtract 14 from 17

17 → 10001 → Minuend

14 → 01110 → 1's = 10001 (+1)

Subtrahend

1 | 00010

00011 = 3.

(→ 17, 14, 3)

2's complement subtraction :-

Steps :

1) Take the minuend in binary form as it is

2) Take 2's complement to subtrahend and add to the minuend.

3) If carry produces ignore the carry.

Ex: Subtract 4 from 13

13 → 1101

4 → 0100

1's = 1011

2's = 1100

1100 (+)

1 | 1001 = 9

(→ 13, 4, 9)

2) Subtract 14 from 17

17 → 10001

14 → 10010 (+)

01110

1's = 10001

2's = 10010

1 | 00011 → 3.

9's complement subtraction:-

1) Take the minuend as it is. (not in binary)
2) Take 9's complement of subtrahend and add to minuend.
3) If a carry is produced add carry to minuend.

Ex: subtract 5 from 8

8 → sub → 8
5 → min → 5 → 4
9's 9-5 = 4
4 + 1 = 5
1 2 + 1 = 3

10's complement Subtraction:

1) Take the minuend as it is
2) Take 10's complement of subtrahend and add to minuend
3) Ignore the carry.

Ex Subtract 5 from 8.

8 → sub → 8
5 → min → 5
9's 9-5=4
10's 4+1=5
1 3 = 3.

Signed Binary numbers:-

In a decimal number system we use '+' sign to denote positive number and '-' to denote -ve number. The digital circuits understand only 0 and 1. The bit '0' is used to represent positive number and '1' is used to represent negative number. The bit used to represent the sign is called sign bit It is always MSB of binary number.

Ex

Decimal	sign	Magnitude (8 bits)
+5	0	000 0101
-5	1	000 0101

Sign -1's complement.

To represent a negative number in 1's complement form the following two steps are used.

1) write a positive binary form of the number including the sign bit
2) Invert each bit including the sign bit (ie) take 1's complement.

Ex: The sign -1's complement representation for -12.

+12 = 0000 1100
-12 = 1111 0011

Sign -2's complement:

1) write a positive binary form of the number including sign bit.
2) Invert each bit including the sign bit (ie) take 1's complement.
3) Add 1 to the result.

Ex The sign -2's complement for 12.

+12 = 0000 1100
-12 = 1111 0011 +
1's 1
2's 1111 0100

for Unit 1

UNIT - 1

BINARY CODES

All digital circuits operate with only two states 1 and 0. Some binary codes are required to represent alphabets and special characters. Digital calculators, frequency meters prefer 4-bit code to represent decimal numbers 0 to 9.

BCD codes - 8421 codes :-

A group of bits (usually four) which are used to represent decimal numbers 0 to 9 are called Binary coded Decimal codes or BCD codes.

The most popular BCD code is 8421 code. The 8421 indicates the binary weights of four bits ($2^3, 2^2, 2^1, 2^0$). The four bits with weights 8, 4, 2, 1 can be easily represent the decimal numbers 0-9.

The four bit binary combination can be represented for first ten combination. The six remaining combinations 1010, 1011, 1100, 1101, 1110, 1111 are invalid 8421 BCD codes.

Decimal	8 4 2 1	Decimal	8421
0	0000	6	0110
1	0001	7	0111
2	0010	8	1000
3	0011	9	1001
4	0100		
5	0101		

Ex:

Decimal	8421 BCD code
15	0001 0101
29	0010 1001
468	0100 0110 1000
97.5	1001 0111 . 0101

Note: Each digit must be represented by 4-bit BCD code.

BCD (8421) Addition :-

* The general rule for Binary Addition 0+0=0, 0+1=1, 1+0=1, 1+1=10. (0 with carry 1)

The rules for BCD addition

1) Add the two numbers using rules of binary addition.
2) If 4-bit sum is equal to or less than 9, it is valid BCD number.
3) If the 4-bit sum is greater than 9, or if a carry out of a BCD group is generated, the result is invalid BCD. To get correct BCD result add (06) to the 1st group and take carry to next group.

Ex: 1) Add 5 and 2.

5 - 0101
2 - 0010
7 = 0111 → valid BCD

2) Add 7+6.

7 - 0111
6 - 0110
13 1101 → invalid, it exceeds 9
(7) 6 → 0110 → Add (06)
1 0011
0001 0011
(13)

3) 9 and 8

1001
1000
1 0001
0 0110
1 0111

0001 0111 (7)

2421 codes and 4221 codes:-

The 8421 code is very useful code. we have seen how to add two BCD numbers. But there is a difficulty in this code. we know that subtraction operation is done in computers, by adding the complement of the subtrahend. This method is not useful in 8421 BCD code.

Ex Subtrahend is 0010(2)
1's complement is 1101
and 1101 is not a valid BCD number because it is greater than 9.

To solve this problem, the codes are designed in such a way that when the binary code representing a decimal number is complemented, the binary code representing the 9's complement of the decimal number is obtained. Such codes are called self-complementing codes. Two such codes are 2421 and 4221 codes.

Decimal	2421	4221	Decimal	2421	4221
0	0000	0000	7	1101	1101
1	0001	0001	8	1110	1110
2	0010	0010	9	1111	1111
3	0011	0011			
4	0100	0110			
5	1011	1001			
6	1100	1100			

2421 code.

Ex 1) Binary code for 2 = 0010
Taking 1's complement 1101 and this represents 7. which is 9's complement of 2.

9's complement : 9 - 2 = 7

2) Binary code for 4 = 0100
Taking 1's complement 1011 and this represent 5 which is 9's complement 4.

9's comp 9 - 4 = 5

4221 code:-

Ex 1) Binary code for 1 = 0001
Taking 1's complement 1110 and this code represe-nts 8, which is 9's complement of 1.

9's comp 9 - 1 = 8

2) Binary code for 4 = 0110
1's complement = 1001 → 5 of 4221

9's comp of 4 = 9 - 4 = 5

Excess-3 code:-
The Excess-3 code is another BCD code. The Excess-3 code for decimal digit is obtained by adding 0011 (3) in

Decimal	8421	Excess-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

The Excess-3 code has 10 valid codes and 6 invalid codes are 0000, 0001, 0010, 1101, 1110 and 1111. Like in 8421 code, In Excess-3 code also if the number is greater than 9, the code is given separately for each digit.

Decimal	Excess-3 code
26	0101 1001
97	1100 1010
853	1011 1000 0110

Excess-3 addition:-

The addition in Excess-3 code is also not straight forward and requires inclusion of correction methods. Two types of corrections are needed

Case 1: Sum less than 9

Whenever adding two decimal numbers represented in Excess-3 code, if the sum is less than 9 an Excess-6 number results. To return to Excess-3 form we have to subtract 3.

Ex 2 + 3 = 5

0101 Excess-3 of 2
0110 Excess-3 of 3
1011 Excess-6 of 5
(-) 0011 Subtract 3 of 8421
1000 = Excess-3 of 5 which is correct result

0101 → 3+2
0110 → (6) over 6.
1011 (5)

2) 32 + 47 = 79

(1)
0110 0101 Ex-3 of 32
0111 1010 Ex-3 of 42
1101 1111 Excess-6 of 79
(-) 0011 0011 Subtract 3 from each group
1010 1100 = Ex-3 of 79 which is correct result

Case 2: A carry is produced.

When the sum of two decimal digit represented in Excess-3 exceeds 9, there will be a carry from one group into next. When it happens, the group that produced the carry will be 8421 and the next group will be Ex-3

To get the result in proper Excess-3 form, we have to add 3 to the group that has produced the carry and subtract from next group.

Ex 29 + 49 = 78

0101	1100	Ex-3 for 29
0111	1100	Ex-3 for 49
1101	1000	first group in Excess-6 and second group in 8421.
0011	0011	Sub 3 from 1st group and add 3 to 2nd
1010	1011	→ correct result in ex-3 (78).

Gray codes :-

This code can be used in sequence counting. When the count advances by one, to reduce error, the number of changes in the bits has to be kept minimum.

Decimal	Binary	Gray	Decimal	Binary	Gray
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1001
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

Binary to Gray conversion :-

To convert a binary number to its equivalent Gray code, the following rules are applied.

The MSB of the Gray code is the same as the MSB of the binary.

Coding from left to right, add each adjacent pair of bits to get the next bit of the Gray code. Omit the carries if occurs.

Ex: Let us convert the binary number 1011 to Gray code. (Binary number 1011 = 1110 in Gray code)

Step 1 :-

The left most bit (MSB) in Gray code is the same as MSB of the binary.

1 0 1 1 Binary

↓

1 Gray

Step 2 :

1 + 0 = 1

1 + 1 = 0

1 0 1 1 : 10

1010 → 1 carry (leave)

1110 → leave the carry.

Ex 2) Represent $(45)_{10}$ in binary and Gray code.

2 | 45
2 | 22 - 1
2 | 11 - 0
2 | 5 - 0
2 | 2 - 1
1 - 0

$(101101)_2 = (111011)_{Gray}$.

ASCII code :-

ASCII stands for American standard code for information Interchange. This is 7-bit code used to represent decimal digits 0 to 9, alphabets A to Z (both lower case and upper case) and some special characters. That is ASCII is an alpha numeric code.

Since ASCII is a 7-bit code there are $2^7 = 128$ possible binary combinations.

ASCII symbol	Decimal	Hex	7-binary
0	48	30	011 0000
1	49	31	011 0001
2	50	32	011 0010
9	57	89	011 1001
:	58	3A	011 1010
?	63	3F	011 1111
A	65	41	100 0001
B	66	42	100 0010
Z	90	5A	101 1010
a	97	61	110 0001
b	98	62	110 0010
z	122	7A	111 1010
DEL	127	7F	111 1111

Note:

Another alphanumeric code commonly used in IBM machine is called as Extended Binary coded Decimal Interchange code - EBCDIC

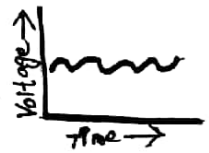
DIGITAL LOGIC

Logic Gates:-

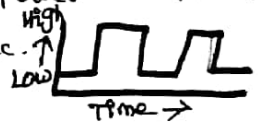
The binary number system uses only two bits '0' and '1'. The digital circuits use only two states ON and OFF, HIGH and LOW, TRUE or FALSE. The boolean algebra is developed by Boole the digital circuits, which use binary number system are also called as logic circuits. The electronic switching circuits are called as logic gates. Logic gates is the fundamental building blocks for all the digital circuits. A low voltage is represented by '0' and high voltage is represented by '1'.

Analog and Digital signals:-

Analog signal is continuous in nature and takes all possible values within a limit. A sample of analog signal that values with time.



A digital signal or takes only two values. Low (0) and High (1). It requires a power supply of 5 volts is marked as Vcc.



Basic logic Gates

NOT, OR, AND:

Gate is an electronic circuit with one or more inputs but only output. All the gates are digital because the input and output signals are either low or High.

NOT GATE OR INVERTER :-

A NOT gate has only one input and one output. For the NOT gate, when the input is LOW (logic 0) and the output is HIGH (logic 1), and when the input is HIGH the output is LOW. (ie) the output is complement or inverse of the input. the input is marked as A and the output is marked as $\overline{A}$.

$$Y = \overline{A}.$$

The output $\overline{A}$ can be read as complement of A or inverse of A or not A or simply A bar.

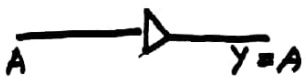
Truth Table :-

A truth table gives the output state for each possible input state combinations. for the NOT gate, there is only one input and can have two possible combinations namely 0 and 1. The truth table for NOT gate is as follows :

A	$Y = \overline{A}$	A	$Y = \overline{A}$
0	1	LOW	HIGH
1	0	HIGH	LOW

A is the input binary variable and Y is the output binary variable. The binary variable is also called as logical variable.

In the symbol of the NOT gate, the small circle usually called as the bubble at the end of the triangle is used to indicate inversion. If the bubble is removed, the gate is called as the buffer. For the buffer the output is equal to the input.



A	Y=A
0	0
1	1

OR Gate :-

An OR gate performs logical addition. An OR gate has two or more inputs and the output is one. If any input signal is HIGH, the output signal goes HIGH. The output is LOW only when all the inputs are LOW. For two inputs there are $2^2 = 4$ combinations.

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

Here A and B are input logical variables and y is the output. The output y is given by the expression $y = A+B$.

For 3 inputs $2^3 = 8$ combinations. $Y = A+B+C$.

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

AND GATE :-

An AND gate performs logical multiplication. An AND gate has two or more inputs and one output. If any input signal is low, the output signal goes low. The output is HIGH only when all the inputs are HIGH.

A	B	Y = A·B
0	0	0
0	1	0
1	0	0
1	1	1

The logic symbol and truth table for 3-inputs are

A	B	C	Y = ABC
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Logic circuits and Logic expressions :-

The logic expression are also called as Boolean expression. We must learn to construct digital circuits for a given problem, using

* A set of statements (or)
* Boolean expression (or)
* Truth table.

Ex for the given logic circuit, obtain the logic expression for the output and form the truth table

Solution :-

The Input to the AND gate is A and B and C and therefore, the output is A·B. This output is given as input to the OR gate.

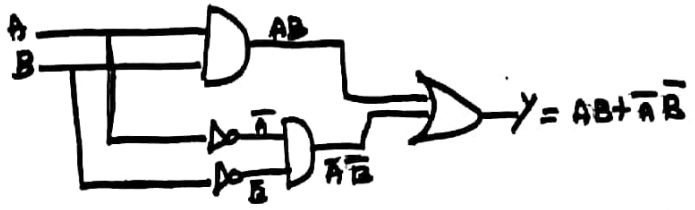
Therefore, the final output is

$$Y = AB + c$$

There are three variables A, B and C. There will be 8 binary combinations.

A	B	C	AB	Y = AB + C
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

Ex 2) For the given logic circuit obtain, the logic expression for the output and form truth table.



Solution:-

The Input to the top AND gate is A and B therefore the output is A.B. The input to the bottom AND gate is $\bar{A}$ and $\bar{B}$ and therefore its output is $\bar{A}.\bar{B}$. These two outputs act as input to an OR gate and the output of OR gate is

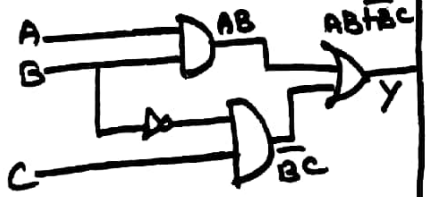
$$Y = AB + \bar{A}\bar{B}$$

A	B	$\bar{A}$	$\bar{B}$	AB	$\bar{A}\bar{B}$	$Y = AB + \bar{A}\bar{B}$
0	0	1	1	0	1	1
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	1	0	0	1	0	1

Ex 3) For the logic expression $Y = AB + \bar{B}C$, draw the logic circuit using only AND, OR and NOT gate.

Solution:-

$Y = AB + \bar{B}C$

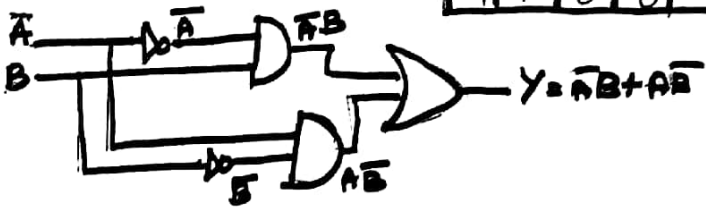


A	B	C	$\bar{B}$	AB	$\bar{B}C$	$AB + \bar{B}C$
0	0	0	1	0	0	0
0	0	1	1	0	1	1
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	0	1	1	0	1	1
1	1	0	0	1	0	1
1	1	1	0	1	0	1

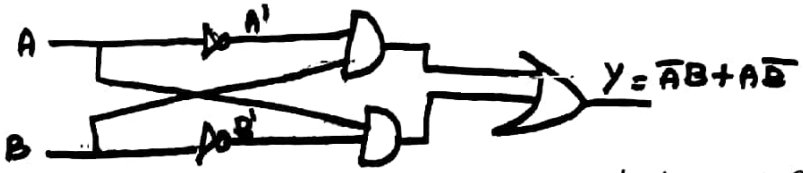
4) For the logic expression $Y = \bar{A}B + A\bar{B}$ draw the logic gates using AND, OR and NOT.

Solution:-

A	B	$\bar{A}$	$\bar{B}$	$\bar{A}B$	$A\bar{B}$	$\bar{A}B + A\bar{B}$
0	0	1	1	0	0	0
0	1	1	0	1	0	1
1	0	0	1	0	1	1
1	1	0	0	0	0	0



(or)



There are two methods widely used to convert the truth table to a logical expression. They are

1) Sum of product (SOP)
2) Product of sum (POS)

Sum of Products (SOP):-

In SOP, the expression consists of number of terms (depending on the problem). Each term is a logical product of input variables. All the terms are logically summed up to give the output and hence the name sum of product.

Ex: $Y = \bar{A}\bar{B}C + \bar{A}B\bar{C} + ABC$

This expression is said to be Sum of Product form. The first term is a logical product of $\bar{A}\bar{B}$ and C. The second term is a logical product of $\bar{A}$ B and C. The third term is a logical Product of A B and C. The products $\bar{A}\bar{B}C$ and $\bar{A}B\bar{C}$ and ABC are logically summed up to give the Sum of Product expression. This expression is also called as standard sum of Product or standard SOP. The products $\bar{A}\bar{B}C$, $\bar{A}B\bar{C}$, ABC are called Fundamentals or standard Products. These fundamental products are also called as minterms.

For two variables A and B there are four possible combinations. $\bar{A}\bar{B}$ (00), $\bar{A}B$ (01), $A\bar{B}$ (10), AB (11) and therefore there are 4 standard products or 4 minterms.

Truth table:

A	B	Y
0	0	1
0	1	0
1	0	1
1	1	0

To write the standard SOP expression for the truth table, select the rows whose output is 1. In 1st row Y=1, when A=0 and B=0. Both are complemented $\bar{A}=1$, $\bar{B}=1$ so that logical Product $\bar{A}\bar{B}$ gives 1. In 3rd row Y=1, A=1, B=0. In this case only B needs to be complemented and $A\bar{B}$ gives a 1. Therefore the two standard products are $\bar{A}\bar{B}$ & $A\bar{B}$. These two standard products are logically summed to give the standard SOP.

$$Y = \bar{A}\bar{B} + A\bar{B}$$

(using Boolean algebra this expression can be simplified to give $Y = \bar{B}$) (ie $A + \bar{A} = 1$)

Ex:

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Y=1, when A=0, B=0, C=1 standard product SOP $\bar{A}\bar{B}\bar{C}$

A=1, B=0, C=1 SOP $A\bar{B}C$

A=1, B=1, C=0 SOP $AB\bar{C}$

A=1, B=1, C=1, SOP ABC

∴ Standard SOP expression is

$Y = \bar{A}\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC$

PRODUCT OF SUM:-

In this method each term is a logical sum (OR) of input variables. All the terms are logically multiplied to give the output and hence the name Product of sums (POS)

In SOP each term is logical product and gives a logical 1. In POS each term is a logical sum and produces a logical 0. Therefore, the product of the sums also gives a zero.

Ex: $F(A,B,C) = (A+\bar{B}+\bar{C})(A+\bar{B}+C)(\bar{A}+B+C)$

(or) $F(A,B,C) = (\bar{A}+\bar{B}+C)(\bar{A}+B+\bar{C})(A+B+C)$

is said to be in product of sum form.

The sums of $(A+B+C)$, $(A+B+\bar{C})$ are called fundamental or standard sums.

* The term $(\bar{A}+\bar{B}+C)$ corresponds to $A=1$, $B=1$ and $C=0$ so that $(\bar{A}+\bar{B}+C)=0$.
* The term $(\bar{A}+B+\bar{C})$ corresponds to $A=1$, $B=0$, $C=1$ $(\bar{A}+B+\bar{C})=0$
* The term $(A+B+C)$ corresponds to $A=0$, $B=0$, $C=0$ so that $(A+B+C)=0$

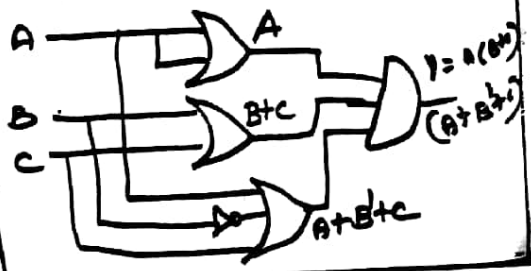
∴ The terms put together gives $Y = 0\cdot0\cdot0 = 0$

These fundamental/standard sums are also called as Maxterms.

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Ex Gate for POS:-

$F = A(B+C)(A+B'+C)$



$Y=0$, when $A=0$, $B=0$, and $C=0$, the standard sum is $(A+B+C)$

when $A=0$, $B=1$ and $C=0$, the standard sum is $(A+\bar{B}+C)$

$A=0$, $B=1$, $C=1$, the standard sum is $(A+\bar{B}+\bar{C})$

$A=1$, $B=0$, $C=0$, the standard sum is $(\bar{A}+B+C)$

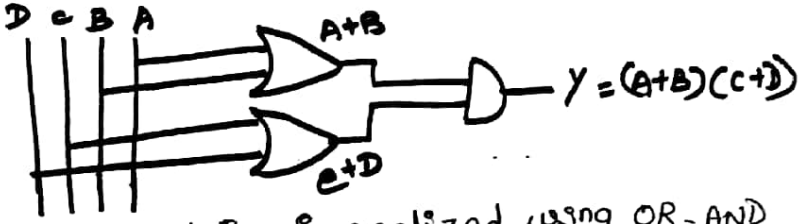
Therefore, the standard POS expression is

$Y = (A+B+C)(A+\bar{B}+C)(A+\bar{B}+\bar{C})(\bar{A}+B+C)$

Ex: Draw the circuit for,

$F(A,B,C,D) = (A+B)(C+D)$

The expression is in POS form.



The POS representation is realized using OR-AND configuration. This is a two-level realization. The first level consists of OR gates and the second level consists of AND gate.

Boolean functions expressed as sum of minterms or product of maxterms are said to be in canonical - form.

NAND and NOR GATES:-

Ex Sum of product:-

$F = A + AB' + A'Bc'$

When a logical circuit is designed, we take two sets of known values into consideration. They are

1) The different states that the inputs to the logical network can assume.
2) The desired output for each input values.

Derivation of a Boolean Expression:- (SOP)

A binary variable x can exist either in its normal form (x) or in its complement form ($\bar{x}$).

Minterms for three variables

x	y	z	Product term	Designation
0	0	0	$x'y'z'$	m0
0	0	1	$x'y'z$	m1
0	1	0	$x'yz'$	m2
0	1	1	$x'yz$	m3
1	0	0	$xy'z'$	m4
1	0	1	$xy'z$	m5
1	1	0	xyz'	m6
1	1	1	xyz	m7

These are called as standard product or minterms

If we OR these variables, we will obtain standard sums or maxterms.

Maxterms for three variables

x	y	z	Product term	Designation
0	0	0	$x+y+z$	M0
0	0	1	$x+y+z'$	M1
0	1	0	$x+y'+z$	M2
0	1	1	$x+y'+z'$	M3
1	0	0	$x'+y+z$	M4
1	0	1	$x'+y+z'$	M5
1	1	0	$x'+y'+z$	M6
1	1	1	$x'+y'+z'$	M7

A Boolean function f may be expressed algebraically from a truth table by forming a minterm for each combination of the variables that produces 1 in the function and then taking the OR of all those terms.

Any Boolean function can be expressed as a product of maxterms. Boolean function expressed as a sum of minterms or product of maxterms are said to be canonical form

There are two types of canonical forms

* Sum of minterms
* Product of maxterms.

Sum of Minterms :-

For a function with n binary variable, a maximum of 2^n distinct minterms can be obtained. If the given Boolean function is not in the sum of minterms form, it should be converted by expanding the expression into a sum of AND terms. If any of these terms misses any one of the variable, then that particular term in the expression should be ANDed with $(x+\bar{x})$ where x is missing variable.

Ex Express the Boolean function $F = A'+BC$ in a sum of minterms.

Solution: The given function $F = A'+BC$ contains three variables A, B and C. The 1st term contains only one variable and 2nd term contains 2 variables. Like should include the missing variables.

1st term : $A' = A'(B+B')$

B $= \boxed{A'B + A'B'}$

2nd term : $A' = A'B(C+C') + A'B'(C+C')$

C $= \boxed{A'BC + A'BC' + A'B'C + A'B'C'}$

Now we shall perform the same for 2nd variable.

1st term $BC = BC(A+A')$

A $= BCA + BCA'$

$= \boxed{ABC + A'BC}$

Hence the function F can be rewritten as

$F = A' + BC$

$= A'BC + A'BC' + A'B'C + A'B'C' + ABC + A'BC$

(omit the repeated terms once)

$\therefore F = A'BC + A'BC' + A'B'C + A'B'C' + ABC.$

From the truth table, we can obtain the minterms as $F = A'B'C + A'B'C + A'BC' + A'BC + ABC$

$F = m0 + m1 + m2 + m3 + m7$

$\therefore F(A,B,C) = \boxed{\Sigma(0,1,2,3,7)}$

Ex 2) $F(A,B,C) = AB + \bar{B}C =$

Product of Maxterms :-

To express a Boolean function as a product of maxterms, it should appear in the form of OR terms. To do so, we can make use of the distributive law $\boxed{a+bc = (a+b)(a+c)}$. If any of the term misses any variable, should be include in the missing term with xx', where x is the missing variable for that term.

Ex: Express the Boolean function $F = AC + B'C'$ in a product of maxterm.

Solution: First we should convert the function $F = AC + B'C'$ into OR terms using distributive law.

$F = Ac + B'c'$

$= (Ac + B')(Ac + c')$

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

$= (A + B')(c + B')(A + c')$ $(c + c' = 1)$

we have converted the given function into OR terms. Next we include the missing variable.

$(A + B') = A + B' + (C \cdot C') = (A + B' + C)(A + B' + C')$

$C + B' = C + B' + (A \cdot A') = (A + B' + C)(A' + B' + C)$

$A + C' = A + C' + (B \cdot B') = (A + C' + B)(A + C' + B')$

we can rewrite the above function F as follows

$F = (A + B' + C)(A + B' + C')(A + B' + C)(A' + B' + C)(A + B + C')(A + B' + C')$

(remove redundant terms)

$F = (A + B' + C)(A + B + C')(A' + B' + C)(A + B' + C')$

$= M_6 \cdot M_1 \cdot M_2 \cdot M_3$

$F(A, B, C) = \Pi(6, 1, 2, 3)$.

conversion of sum of Minterms and product of Maxterms:

To convert from one canonical form to another, interchange the symbols Σ and Π and include the numbers missing from the original form.

Ex $F(A, B, C) = \Sigma(0, 3, 5, 6)$

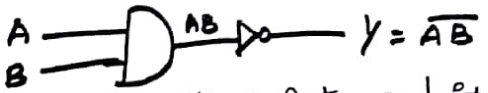
The function expressed in sum of minterms can be converted to the product of maxterms as shown below

$F(A, B, C) = \Pi(1, 2, 4, 7)$

NAND and NOR Gates:-

NAND gate:

NAND can be imagined to be an AND gate followed by a NOT gate as shown below.



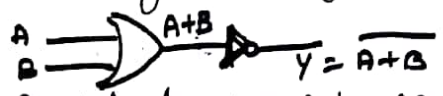
The Symbol for NAND gate and its truth table

The output is 1 when any input goes to 0. The output is 0 only when all the inputs are 1.

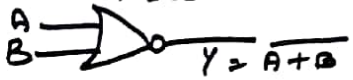
A	B	$Y = \overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

NOR gate:-

NOR can be imagined to be an OR gate followed by a NOT gate.



The Symbol for NOR gate and its truth table.



A	B	$Y = \overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Ex-OR and Ex-NOR gates :-

Ex-OR gate :-

In a two input Exclusive OR (Ex-OR or XOR) the truth table is as below

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

for OR gate A=1 B=1 the O/P Y=1

For Ex-OR gate A=1, B=1 the O/P Y=0

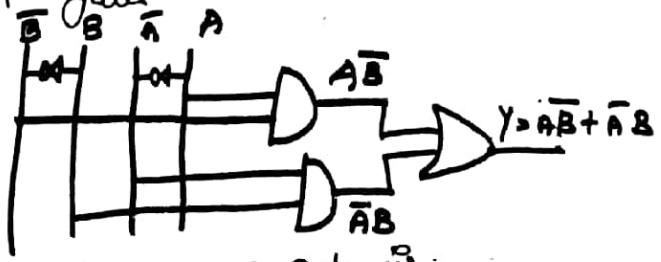
From the table,

* Y=1 for A=0, B=1 ∴ the standard product is $\boxed{\bar{A}B}$.
* Y=1 for A=1, B=0, the standard product is $\boxed{A\bar{B}}$.

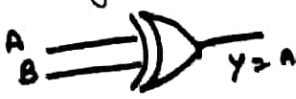
∴ The SOP expression is

$$\boxed{Y = \bar{A}B + A\bar{B}}$$

This can be implemented using two AND gates and two NOT gates



The symbol for Ex-OR gate is



The output is given by the expression

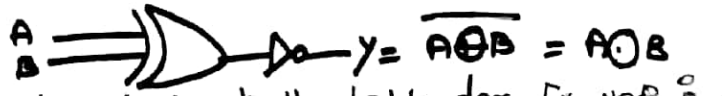
$$Y = \bar{A}B + A\bar{B}$$

Also $Y = A \oplus B$

and read as A Ex-OR B.

Ex-NOR Gate :-

Exclusive – NOR gate (XNOR) is equivalent to Ex-OR followed by an inverter (NOT) gate



The symbol and the truth table for Ex-NOR is as below.

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

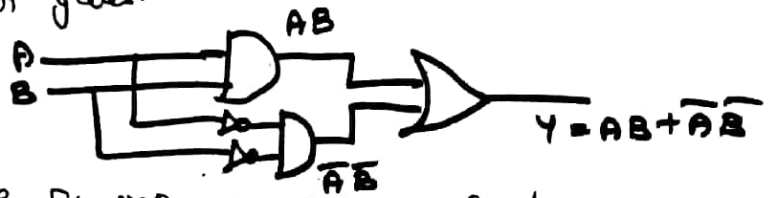
From the truth table we can write, SOP

Y=1 for A=0, B=0 the standard Product is $\bar{A}\bar{B}$.

Y=1 for A=1, B=1 the standard product is AB

Therefore, $Y = \bar{A}\bar{B} + AB$.

The Ex-NOR can be implemented using AND, OR and NOT gates.



As the Ex-NOR operation is derived by complementing the Ex-OR, we can prove with the help of Boolean laws that,

$$\overline{(\bar{A}B + A\bar{B})} = \bar{A}\bar{B} + AB.$$

Since the output is 1 only when both inputs are equal, the Ex-NOR gate is called as equivalence gate.

UNIT II

BOOLEAN ALGEBRA

Hierarchy of Logic Circuits :-

While evaluating the logic circuits, the following rules with hierarchy should be followed.

* Parentheses have highest priority (ie) all terms within parantheses are evaluated first.
* Inversions have next highest priority over other operations (ie) if single term with 0 is inverted as 1 and 1 as 0.
* AND operations are evaluated
* OR operations are evaluated after AND.
* If an expression has a bar over it, the operations should be performed first and the result should be inverted next.

Laws of Boolean Algebra :-

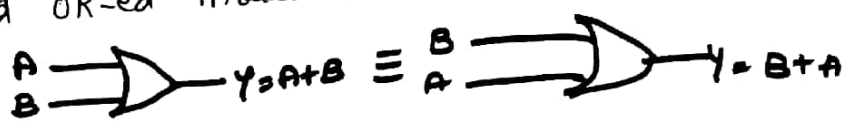
Boolean Algebra uses only two bits 0 and 1 and has only two operations + (OR) and . (AND). A number of rules in Boolean algebra are similar to ordinary algebra but a number of other rules are different.

commutative law :

The commutative law for addition of two variables is written as

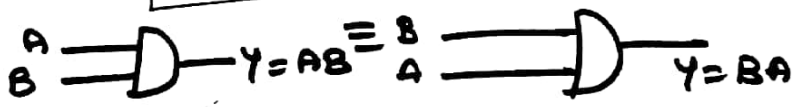
$$A+B = B+A$$

This means that the order in which the variables are OR-ed makes no difference.



In a similar way, the commutative law for multiplication of two variables is

$$AB = BA$$

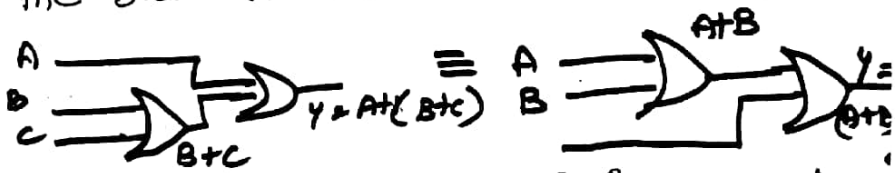


Associative law :-

The associative law for addition of three variable is

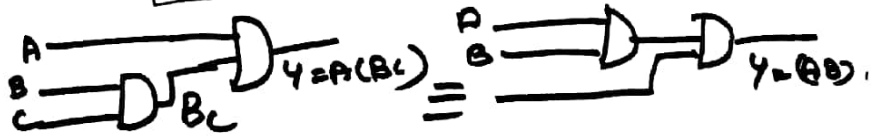
$$A+(B+C) = (A+B)+C$$

This means that the result is the same regardless of the order in which the variables are grouped



In a similar way, the associative law for multiplication of three variables is

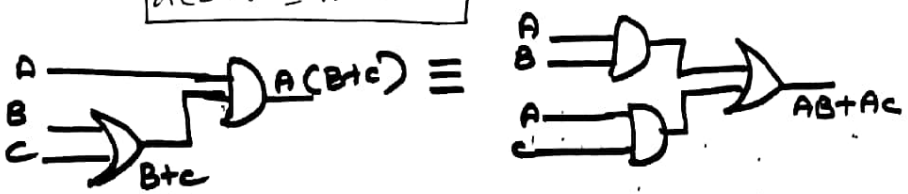
$$A(BC) = (AB)C$$



Distributive law :-

The distributive law for three variables is written as

$$A(B+C) = AB+AC$$



Rule 1a:

$$A+0 = A$$

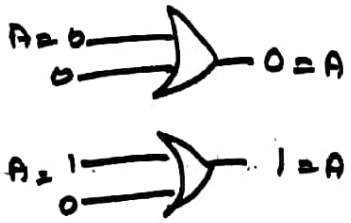
when $A = 0$

$A+0 = 0+0 = 0 = A$

when $A = 1$

$A+0 = 1+0 = 1 = A$

Therefore $A+0 = A$.



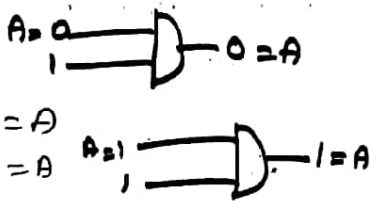
Rule 1b:

$$A \cdot 1 = A$$

when $A = 0$, $A \cdot 1 = 0 \cdot 1 = 0 = A$

when $A = 1$, $A \cdot 1 = 1 \cdot 1 = 1 = A$

$\therefore A \cdot 1 = A$



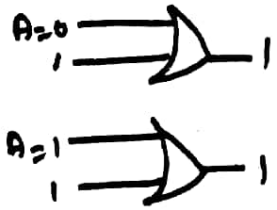
Rule 2a:

$$A+1 = 1$$

when $A = 0$, $A+1 = 0+1 = 1$

when $A = 1$, $A+1 = 1+1 = 1$

$\therefore A+1 = 1$



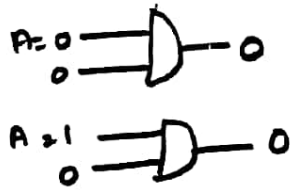
Rule 2b:

$$A \cdot 0 = 0$$

when $A = 0$, $A \cdot 0 = 0 \cdot 0 = 0$

when $A = 1$, $A \cdot 0 = 1 \cdot 0 = 0$

$\therefore A \cdot 0 = 0$



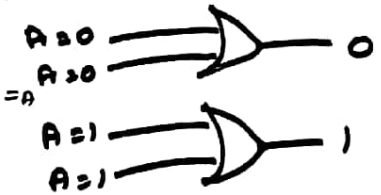
Rule 3a:

$$A+A = A$$

when $A = 0$, $A+A = 0+0 = 0 = A$

when $A = 1$, $A+A = 1+1 = 1 = A$

$\therefore A+A = A$.



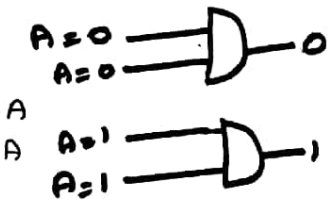
Rule 3b:

$$A \cdot A = A$$

when $A = 0$, $A \cdot A = 0 \cdot 0 = 0 = A$

when $A = 1$, $A \cdot A = 1 \cdot 1 = 1 = A$

$\therefore A \cdot A = A$



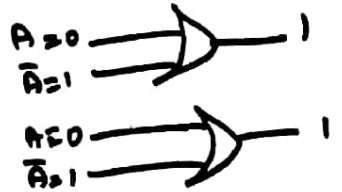
Rule 4a:

$$A+\bar{A} = 1$$

when $A = 0$, $A+\bar{A} = 0+1 = 1$

when $A = 1$, $A+\bar{A} = 1+0 = 1$

$\therefore A+\bar{A} = 1$



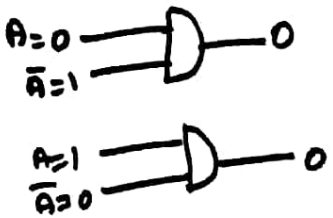
Rule 4b:

$$A \cdot \bar{A} = 0$$

when $A = 0$, $A \cdot \bar{A} = 0 \cdot 1 = 0$

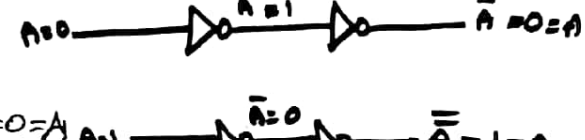
when $A = 1$, $A \cdot \bar{A} = 1 \cdot 0 = 0$

$\therefore A \cdot \bar{A} = 0$.



Rule 5:

$\overline{\overline{A}} = A$



when $A = 0$, $\overline{A} = 1$, $\overline{\overline{A}} = 0 = A$

when $A = 1$, $\overline{A} = 0$, $\overline{\overline{A}} = 1 = A$

$\therefore \overline{\overline{A}} = A$

This double complement rule is also called as Involution.

Rule 6a:

$A + AB = A$

Method 1:

L.H.S $= A + AB$

$= A(1 + B)$

$= A \cdot 1$; for $1 + B = 1$

$= A$; for $A \cdot 1 = A$

$A + AB = A$

Method 2:

using truth table

A	B	AB	A + AB
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Comparing columns 3 & 4 it follows that

$A + AB = A$

This method of using truth table to prove an identity is called "proof by Perfect induction".

Rule 6b:

$A(A + B) = A$

Method 1:

L.H.S $= A(A + B)$

$= AA + AB$

$= A + AB$; for $AA = A$

$= A$; Rule 6a

$=$ RHS

Method 2:

using truth table

A	B	A+B	A(A+B)
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

Comparing column 2 & 4, it follows

$A(A + B) = A$

Rule 7a:

$A + \overline{A}B = A + B$

Proof 1:

LHS $= A + \overline{A}B$

$= A \cdot 1 + \overline{A}B$; for $A \cdot 1 = A$

$= A(1 + B) + \overline{A}B$; for $1 + B = 1$

$= A + AB + \overline{A}B$

$= A + B(A + \overline{A})$; for $A + \overline{A} = 1$

$= A + B$

$=$ RHS

Proof 2:

RHS $= A + B$

$= A + B \cdot 1$

$= A + B(A + \overline{A})$

$= A + BA + B\overline{A}$

$= A + AB + \overline{A}B$;

$= A + \overline{A}B$; A+A

$=$ LHS.

Rule 7b:

$A(\bar{A}+B) = A \cdot B$

Proof:

LHS $= A(\bar{A}+B)$

$= A\bar{A} + AB$

$= AB$; $A\bar{A} = 0$

$\therefore A(\bar{A}+B) = AB$

Rule 8a:

$A + BC = (A+B)(A+C)$ ✓

Proof: RHS $= (A+B)(A+C)$

$= AA + AB + AC + BC$

$= A + AB + AC + BC$; for $AA = A$

$= A(1 + B + C) + BC$

$= A + BC$; for $1 + B + C = 1$

$A + BC = (A+B)(A+C)$

Since

$A + BC = (A+B)(A+C)$

$A + \bar{B}C = (A+B)(A+\bar{C})$

$\bar{A} + \bar{B}\bar{C} = (\bar{A}+B)(\bar{A}+\bar{C})$

Rule 8b:

$A(B+C) = AB + AC$ This is distributive law.

Rule 9a:

$AB + \bar{A}C + BC = AB + \bar{A}C$

Proof:

LHS $= AB + \bar{A}C + BC$

$= AB + \bar{A}C + BC(A+\bar{A})$; $A + \bar{A} = 1$

$= AB + \bar{A}C + ABC + \bar{A}BC$

$= AB(1+C) + \bar{A}C(1+B)$

RHS $= AB + \bar{A}C$; for $1+C = 1$; $1+B = 1$

Since

$A\bar{B} + \bar{A}C + \bar{B}C = A\bar{B} + \bar{A}C$

$\bar{A}B + AC + BC = \bar{A}B + AC$

$\bar{A}\bar{B} + A\bar{C} + \bar{B}\bar{C} = \bar{A}\bar{B} + A\bar{C}$

$XY + \bar{X}Z + YZ = XY + \bar{X}Z$.

Rule 9b:

$(A+B)(\bar{A}+C)(B+C) = (A+B)(\bar{A}+C)$

Proof:

LHS $= (A+B)(\bar{A}+C)(B+C)$

$= (A\bar{A} + \bar{A}B + AC)(B+C)$

$= (\bar{A}B + AC + BC)(B+C)$; for $A\bar{A} = 0$

$= \bar{A}BB + ABC + BBC + \bar{A}BC + ACC + BCC$

$= \bar{A}B + ABC + BC + \bar{A}BC + AC + BC$; for $AA = A$, $BB = B$

$= \bar{A}B + \bar{A}BC + ABC + BC + AC + BC$

$= \bar{A}B(1+C) + BC(A+1) + AC + BC$

$= \bar{A}B + BC + AC + BC$; for $1+C = 1$, $1+A = 1$

$= \bar{A}B + AC + BC$; $BC + BC = BC$

RHS $= (A+B)(\bar{A}+C)$

$= A\bar{A} + \bar{A}B + AC + BC$

$= \bar{A}B + AC + BC$; for $A\bar{A} = 0$

LHS = RHS

Duality Principle :-

If Rule 1a is given 1b can be obtained. In all these cases, if one rule is given, the other one of the pair can be obtained by interchanging the binary operator OR and AND by replacing with 0's and 1's. This property is called DUALITY PRINCIPLE.

Ex: $A+0=A$ dual of $A\cdot 1=A$ Ily $A+1=1$ dual of $A\cdot 0=0$

Given a Boolean rule, the dual can be obtained by

* Changing each OR sign to AND sign
* Changing each AND sign to OR sign
* complementing any 0 or 1 present in expression.

Ex simplify $Y=\bar{A}\bar{B}+\bar{A}B$

solution:

$$Y=\bar{A}(\bar{B}+B)$$
$$=\bar{A}\cdot 1 \quad (\bar{B}+B=1)$$
$$=\bar{A} \quad (\bar{A}\cdot 1=\bar{A})$$

$\therefore \boxed{Y=\bar{A}}$

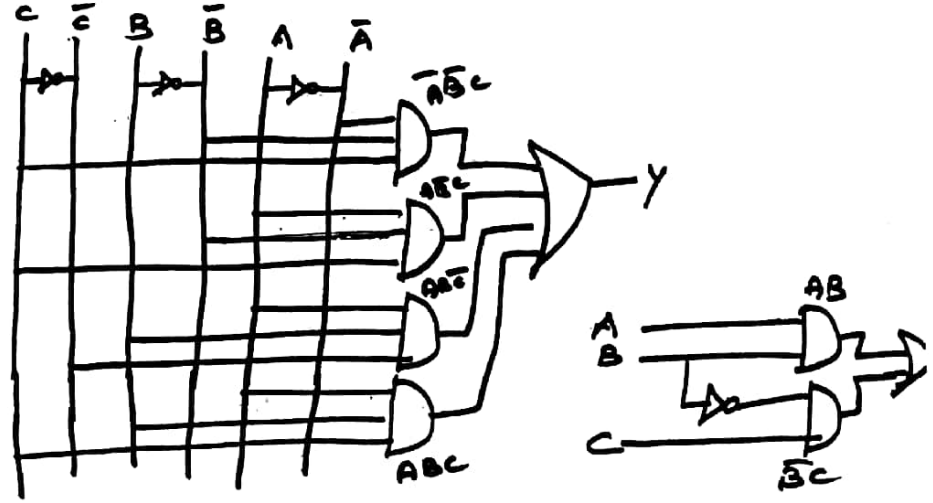
Ex 2 : show that $\bar{A}\bar{B}C+A\bar{B}C+AB\bar{C}+ABC = AB+\bar{B}C$

solution : $LHS=\bar{A}\bar{B}C+A\bar{B}C+AB\bar{C}+ABC$

$$=\bar{B}C(\bar{A}+A)+AB(C+\bar{C})$$
$$=\bar{B}C+AB$$

$RHS=\boxed{AB+\bar{B}C}$

simplification of logical expressions using Boolean algebra is important, because a reduced expression means less number of gates, hence reduced size and cost and more efficient.



Ex 3: $Y=[A\bar{B}C(C+BD)+\bar{A}\bar{B}]C$

Sln $Y=[A\bar{B}(C+BD)+\bar{A}\bar{B}]C$

$=[A\bar{B}C+A\bar{B}BD+\bar{A}\bar{B}]C$; $B\bar{B}=0$

$=[A\bar{B}C+\bar{A}\bar{B}]C$

$=[A\bar{B}CC+\bar{A}\bar{B}C]$; $CC=C$

$=A\bar{B}C+\bar{A}\bar{B}C$

$=\bar{B}C(A+\bar{A})$; $A+\bar{A}=1$

$=\boxed{\bar{B}C}$

Ex 4: Reduce the following Boolean expression of the literals.

$F(A,B,C) = ABC + \bar{A}\bar{B}C + \bar{A}B\bar{C} + AB\bar{C} + \bar{A}\bar{B}\bar{C}$

Sln $F(A,B,C) = ABC + \bar{A}\bar{B}C + \bar{A}B\bar{C} + AB\bar{C} + \bar{A}\bar{B}\bar{C}$

$= ABC + AB\bar{C} + \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C}$

$= AB(C+\bar{C}) + \bar{A}\bar{B}(C+\bar{C}) + \bar{A}B\bar{C}$

$= AB + \bar{A}\bar{B} + \bar{A}B\bar{C}$

$= \bar{A}\bar{B} + B(A+\bar{A}\bar{C})$

$= \boxed{\bar{A}\bar{B} + B(A+\bar{C})}$

(The five literals are $A, \bar{A}, B, \bar{B}$ & $\bar{C}$)

Ex 5: $F(A,B,C,D) = BC + A\bar{C} + AB + BCD$

Sln $= BC + BCD + A\bar{C} + AB$

$= BC(1+D) + A\bar{C} + AB$

$= BC + A\bar{C} + AB \quad ; 1+D = 1$

$= \boxed{BC + A\bar{C}}$ Rule 9a.

Ex 6: Simplify $AB + AB' + A'C + A'C'$

Sln $AB + AB' + A'C + A'C' = A(B+B') + A'(C+C')$

$= A(1) + A'(1)$

$= A + A'$

$= \boxed{1}$

Ex 7: Simplify $(A+B)\cdot(A+B')$

Sln $(A+B)(A+B') = AA + AB + AB + BB'$ $(AB+AB=AB,\ BB'=0)$

$= A + AB$

$= A(1+B)$

$= \boxed{A}$

Ex 8: Simplify $xy(x'yz' + xy'z' + x'y'z')$

Sln $= xy(x'yz' + y'z'(x+x'))$

$= xy(x'yz' + y'z')$

$= xx'yz + xyy'z'$

$= \boxed{0}$

Ex 9: Complement the Boolean expression $(AB' + C'D')$

Sln $F = AB' + C'D'$

$\boxed{F' = (A+B')\cdot(C'+D')}$

Ex 10: complement $(A + BC + AB)$

$F = A + BC + AB$

$= A(1+B) + BC$

$= A + BC$

Dual of $F = A\cdot(B+C)$

$\boxed{F' = A\cdot(B+C)}$

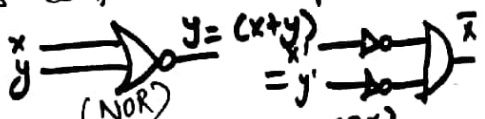
DEMORGANS THEOREM:-

The following two rules are known as Demorgan theorem.

* The + symbols (OR) are replaced with · (AND)
* Each of the term is complemented.

$\boxed{(x+y)' = x'y'}$

$\boxed{(xy)' = x'+y'}$



NOR = Bubbled AND

Bubbled A

Ex. Using demorgans theorem simplify the expression.

$((A'+C) * (B+D'))'$

Sol $= (A'+C')' + (B+D')'$ (By theorem $xy' = x+y'$)

$= ((A')' \cdot c') + B' \cdot (D')')$

$= \boxed{Ac' + B'D}$.

Three variables Demorgan's theorem:

$\boxed{(x+y+z)' = x'y'z'}$

$\boxed{(xyz)' = x'+y'+z'}$

Summary of Boolean Theorems:-

S.NO	classification & Definition
	I. Single variable Theorems
	a. Laws of intersection
1.	$a*0 = 0$
2.	$a*1 = a$
3.	$a*a = a$ (Idempotent law or Tautology)
4.	$a*a' = 0$ (Complement law)
	b. Laws of union
5.	$a+0 = a$
6.	$a+1 = 1$
7.	$a+a = a$ (Tautology)
8.	$a+a' = 1$ (Complement law)

S.NO	classification & Definition
	II Multivariable theorems
	c. commutative laws
9.	$a+b = b+a$
10.	$a.b = b.a$
	d. Associative laws
11.	$a+(b+c) = (a+b)+c = a+b+c$
12.	$a(bc) = (ab)c = abc$
	e. Distributive laws
13.	$a(b+c) = ab+ac$
14.	$(w+x)\cdot(y+z) = wy+wz+xy+xz$
15.	$x+yz = (x+y)\cdot(x+z)$
16.	$xyz + wxy = xy(w+z)$
17.	$xy+yz+wy = y(w+x+z)$
	g. Transposition laws
18.	$xy + x'z = (x+y)\cdot(x+z)$
19.	$(x+y)(x'+z) = x'y+xz$
	h. Absorption laws
20	$x(x+y) = x$
21	$x + x.y = x$
22	$x(x'+y) = x.y$
23	$x.y+y' = x+y'$
24	$xy'+y = x+y$
	i. Involution law
25	$(x')' = x$
26.	$xy + x'z + yz = xy + x'z$
27.	$(x+y)(x'+z)(y+z) = (x+y)(x'+z)$
	K. Demorgans Theorem Two variables
28.	$(x+y)' = x'y'$
29.	$(xy)' = x'+y'$
	Three variable Demorgan
30.	$(x+y+z)' = x'y'z'$
31	$(xyz)' = x'+y'+z'$.

NAND as UNIVERSAL Gate:

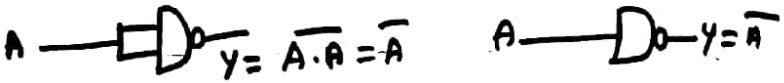
By connecting NAND gates in different ways, it is possible to get the function of any other gate. That is, the function of NOT, AND, OR etc. can be implemented using NAND gates.

∴ The NAND gate is called as universal gate or universal building block.

i) NAND as NOT.

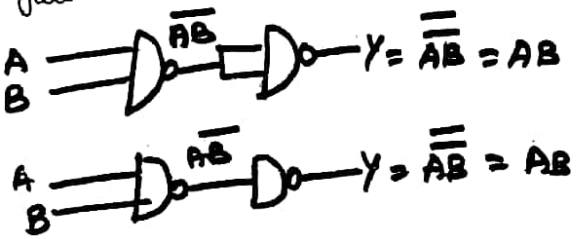
A NOT gate has only one input and one output. In a too input NAND gate,

if both the inputs are marked as A, then the output is given by $A\bar{A}$ which is equal to $\bar{A}$. since the two inputs are same, the two inputs of the NAND gate can be connected together and used as a gate with single input.

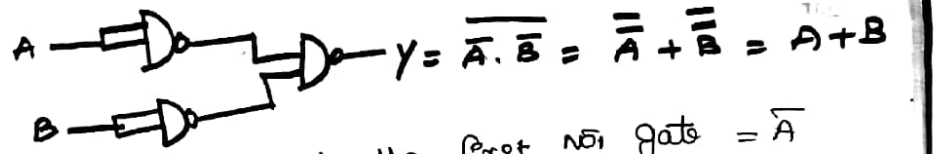


Note:
A 3 input, 4 I/P, & I/P NAND gate can also be converted into a NOT gate by connecting all the inputs together.

ii) NAND as AND:
The output of NAND gate is $\overline{AB}$. To make it as AB, the output of NAND gate is inverted once. But the inverter is also implemented using a second NAND gate.



iii) NAND as OR:-
To get the OR expression A+B, the inputs are first inverted and then pass through a NAND gate. Three NAND gates are needed to get the OR function.

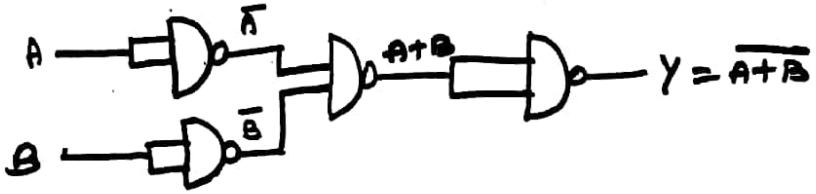


→ The out-put of the first NOT gate = $\bar{A}$
→ The output of second NOT gate = $\bar{B}$
→ These two outputs are fed as input to a two input NAND gate.
→ The output of NAND gate = $\overline{\bar{A}\cdot\bar{B}}$

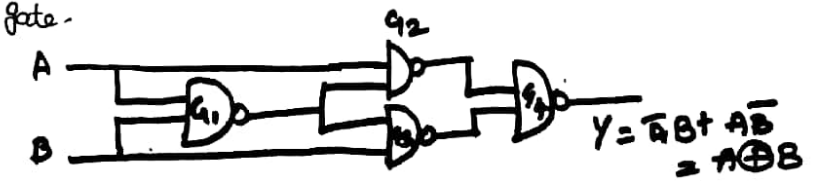
By DeMorgans theorem,

$\overline{\bar{A}\cdot\bar{B}} = \bar{\bar{A}}+\bar{\bar{B}}$
$= A+B$.

iv) NAND as NOR:
To get the NOR expression, simply inverting the output of OR gate. The implementation of NOR gate using NAND gate is given below



v) NAND as EX-OR:
The implementation of EX-OR gate requires four NAND gate.



using DeMorgans theorems wherever required, it can be shown the output of the circuit represents an EX-OR

→ The o/p of $G_1 = \overline{A\bar{B}}$

→ The o/p of $G_2 = \overline{A\bar{A}\bar{B}}$

→ The o/p of $G_3 = \overline{B\bar{A}\bar{B}}$

→ The o/p of $G_4 = \overline{\overline{A\bar{A}\bar{B}}\ \overline{B\bar{A}\bar{B}}}$

$= \overline{\overline{A\bar{A}\bar{B}}} + \overline{\overline{B\bar{A}\bar{B}}}$

$= A\bar{A}\bar{B} + B\bar{A}\bar{B}$

$= A(\bar{A}+\bar{B}) + B(\bar{A}+\bar{B})$

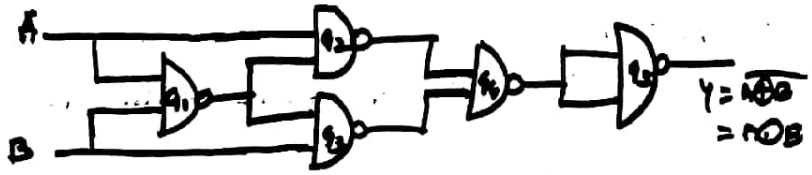
$= A\bar{A} + A\bar{B} + \bar{A}B + B\bar{B}$; $A\bar{A} = B\bar{B} = 0$.

$= A\bar{B} + \bar{A}B$

$= \boxed{A \oplus B}$.

vi) NAND as EX-NOR:

By adding an inverter to the Ex-OR gate, we get the Ex-NOR gate. The implementation of EX-NOR requires five NAND gates.



The output of $G_4 = \bar{A}B + A\bar{B}$

The output of $G_5 = \bar{A}\bar{B} + AB$

$G_4 = \bar{A}B + A\bar{B}$

$= \overline{\bar{A}B} \cdot \overline{A\bar{B}}$

$= A\bar{A}(\bar{A}+\bar{B})(\bar{A}+\bar{B})$

$= (A+\bar{B})(\bar{A}+B)$

$= A\bar{A} + \bar{A}\bar{B} + AB + B\bar{B}$, $A\bar{A} = B\bar{B} = 0$.

$G_5 = \bar{A}\bar{B} + AB$

$= \boxed{A \odot B}$

NAND - NAND Network:

Since NAND is a universal gate, any logic circuit can be constructed using only NAND gates. The AND, OR and NOT gates are directly replaced by the equivalent NAND arrangement. The following steps can be followed to make the procedure unique

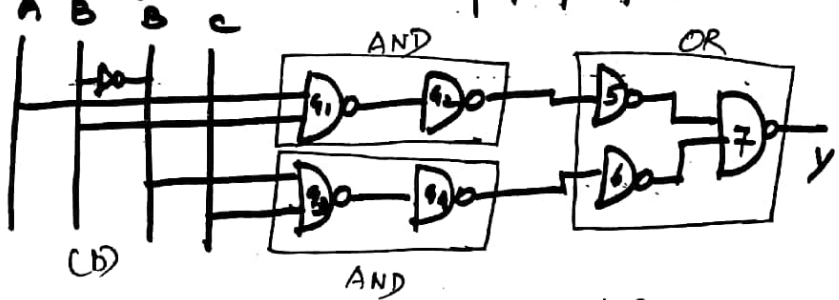
1. The given logic function is simplified and expressed in SOP form.
2. A logic circuit is drawn using basic gates AND, OR, NOT.
3. Each basic gate is replaced by equivalent NAND circuit.
4. When two inverters are cascaded, they can be removed as double inversion does not perform a logic function.
5. If a single inverter occurs connected to an external input, then inverter can be removed and I/P is complem

Ex $f(A,B,C) = \bar{A}\bar{B}C + A\bar{B}C + AB\bar{C} + ABC$

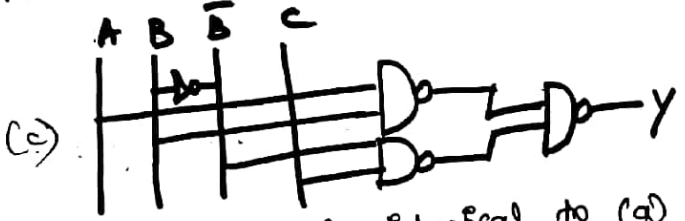
$= \bar{B}C(A+\bar{A}) + AB(C+\bar{C})$ Two-level AND-OR circuit

$= \bar{B}C + AB$

$= AB + \bar{B}C$ (a)



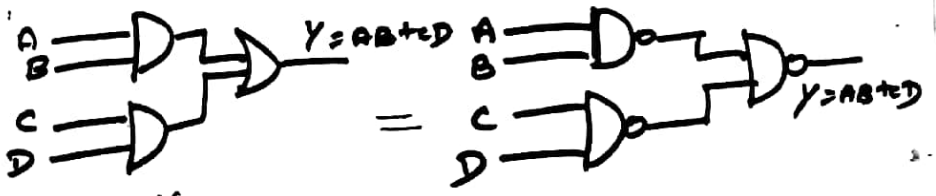
We can see that gates 3 and 4 are cascaded and can be removed. Similarly 4 and 6 can be removed. This gives the final circuit as



In this the gate is identical to (a) but the OR and AND gate is replaced by NAND. This is called two-level NAND-NAND circuit.

Therefore we can conclude that an AND-OR combination circuit can be directly replaced by NAND-NAND combination circuit.

Ex: $Y = AB + CD$ draw the logic circuit using two level AND-OR, two level NAND-NAND combination.



$Y = AB + CD$

$= \overline{AB + CD}$ (Demorgans $\overline{x+y} = \bar{x}\bar{y}$)

$= \overline{AB} \cdot \overline{CD}$

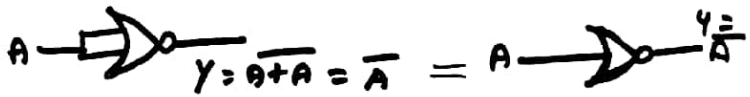
Taking its complement $= \overline{\overline{AB}} \cdot \overline{\overline{CD}}$

$= AB + CD$.

NOR AS UNIVERSAL GATE:

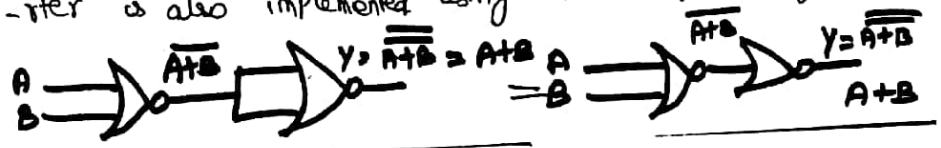
(i) NOR as NOT:-

In a two input NOR gate, if both the inputs are masked as A, then the output is given by $\overline{A+A}$, which is equal to $\bar{A}$.



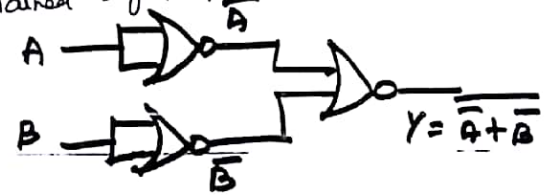
ii) NOR as OR:-

The output of NOR gate is $\overline{A+B}$. To make it A+B the output of NOR gate is inverted once. But inverter is also implemented using a second NOR gate.



iii) NOR as AND:

To get AND expression $A \cdot B$, the inputs are first inverted and then passed through NOR gate. The not gates are obtained using NOR:



The o/p of first NOT gate $= \overline{A}$

The o/p of 2nd NOT gate $= \overline{B}$

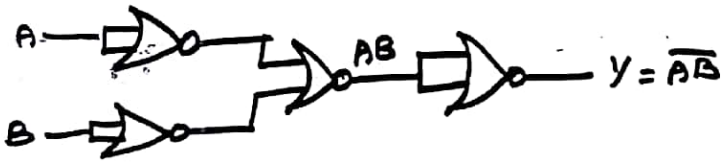
The o/p of NOR gate $= \overline{\overline{A}+\overline{B}}$

By De morgan's theorem $\overline{\overline{A}+\overline{B}} = \overline{\overline{A}} \cdot \overline{\overline{B}}$

$= A \cdot B$

iv) NOR as NAND:-

The NAND gate is obtained by simply inverting the output of the AND gate. The implementation of NAND gate using NOR gate is below.



v) NOR as EX-NOR:

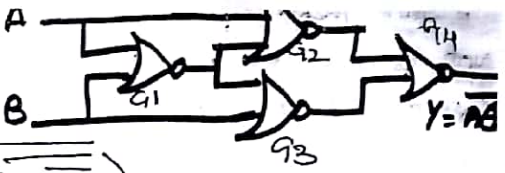
The implementation of EX-NOR gate requires 4 NOR gates.

using Demorgans theorem, we can be shown the output of the circuit.

The output of $G_1 = \overline{A+B}$

The output of $G_2 = \overline{A+\overline{A+B}}$

The output of $G_3 = \overline{B+\overline{A+B}}$



The output of $G_4 = \overline{(A+\overline{A+B})+(B+\overline{A+B})}$

$= \overline{(A+\overline{A+B})} \cdot \overline{(B+\overline{A+B})}$

$= (A+\overline{A+B}) \cdot (B+\overline{A+B})$

$= (A+\overline{A}\,\overline{B}) \cdot (B+\overline{A}\,\overline{B})$

$(A\overline{A}=B\overline{B}=0) = AB+\overline{A}\,\overline{B}B+A\overline{A}\,\overline{B}+\overline{A}\,\overline{B}\,\overline{A}\,\overline{B}$

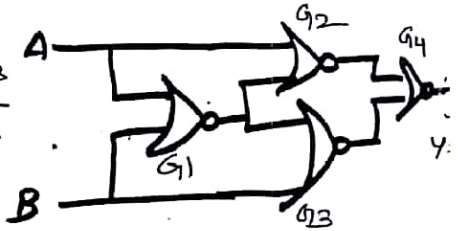
$(\overline{A}\overline{A}=\overline{A}, B\overline{B}=B) = \boxed{AB+\overline{A}\,\overline{B}}$

(which is the expression of EX-NOR)

vi) NOR as EX-OR:

The output expression of EX-OR gate is $\overline{A}B+A\overline{B}$.

By adding an inverter to the EX-NOR gate we get EX-OR gate. It requires five NOR-gate as G1, G2, G3, G4 and G5.



The output of $G_4 = \overline{A}\overline{B}+AB$

The output of $G_5 = \overline{A}B+A\overline{B}$

$\overline{(\overline{A}\overline{B}+AB)} = \overline{A}B+A\overline{B}$

$= (\overline{\overline{A}+\overline{B}})(\overline{A}+\overline{B})$

$= (A+B)(\overline{A}+\overline{B})$

$= A\overline{A}+\overline{A}B+A\overline{B}+B\overline{B}$

$= \boxed{A\overline{B}+\overline{A}B}$

NOR - NOR network :-

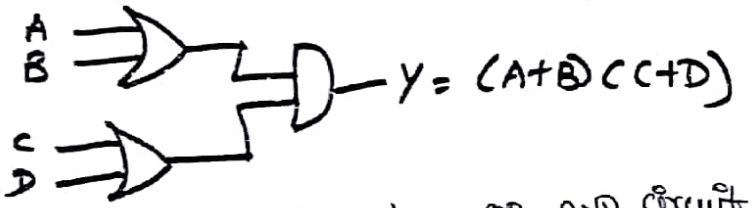
NOR is a universal gate, any logic expression can be implemented using NOR gates only. The following steps are to be followed.

1. The given logic function is simplified and expressed in POS form.
2. A logic circuit is drawn using basic gates AND, OR, NOT gates. The I/P variables are assumed to be available in the normal as well as in complemented form.
3. Each basic gate is replaced by equivalent NOR circuit.
4. When two inverters are cascaded, they can be removed, as double inversion does not perform a logic function.
5. If a single inverter occurs connected to an external input, then the inverter can be removed and the corresponding input is complemented.

Let us take POS expression,

$$Y = (A+B)(C+D)$$

The logic circuit can be drawn as two level OR - AND combination circuit.



We can show that the above OR - AND circuit can be directly replaced by NOR - NOR circuit.

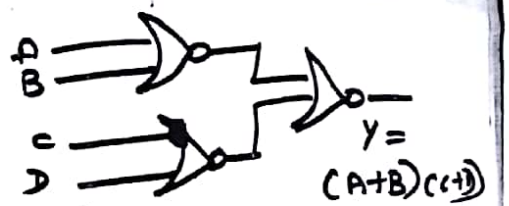
$Y = (A+B)(C+D)$

Taking 1's complement

$$\overline{Y} = \overline{(A+B)(C+D)}$$
$$= \overline{(A+B)} + \overline{(C+D)}$$

Taking 1's complement once again

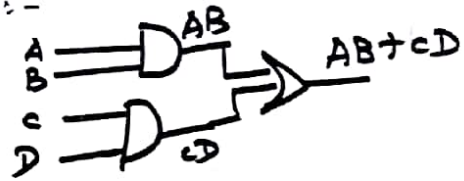
$$\overline{\overline{Y}} = \overline{\overline{(A+B)} + \overline{(C+D)}}$$
$$Y = \overline{\overline{(A+B)} + \overline{(C+D)}}$$
$$= \overline{\overline{(A+B)}} \cdot \overline{\overline{(C+D)}}$$
$$= (A+B)(C+D)$$



That is an OR - AND combination circuit can be directly replaced by a NOR - NOR combination circuit.

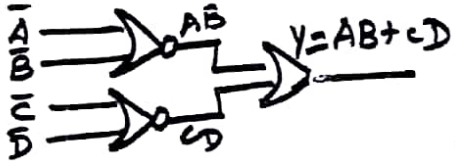
NOR to OR Gate Network :-

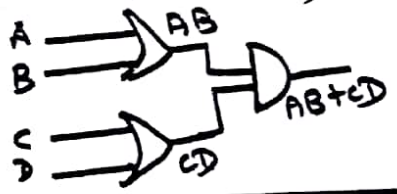
$$Y = \overline{(\overline{A+B})} + \overline{(\overline{C+D})}$$
$$= \overline{A} \cdot \overline{B} + \overline{C} \cdot \overline{D}$$
$$= AB + CD$$

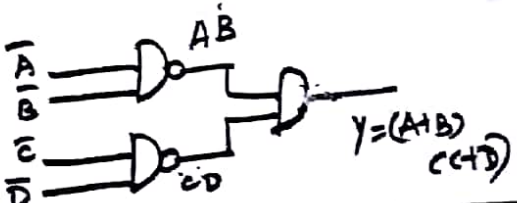


NAND to AND gate Network

$$Y = \overline{(\overline{AB})(\overline{CD})}$$
$$= (\overline{\overline{A}+\overline{B}})(\overline{\overline{C}+\overline{D}})$$
$$= (A+B)(C+D)$$







KARNAUGH MAP

Minterms and Maxterms:

Minterms are associated with SOP and Maxterms are associated with POS

Minterm:-

$f(A,B,C) = \bar{A}\bar{B} + AB\bar{C}$

The first term is a product. The second term contains all the three variables A, B and $\bar{C}$ in its complement form. This type is called standard product. The standard product is also called as minterm and is represented by 'm'. The standard product should produces a logical 1.

A	B	Term	Designation
0	0	$\bar{A}\bar{B}$	m_0
0	1	$\bar{A}B$	m_1
1	0	$A\bar{B}$	m_2
1	1	AB	m_3

A	B	C	Term	Designation
0	0	0	$\bar{A}\bar{B}\bar{C}$	m_0
0	0	1	$\bar{A}\bar{B}C$	m_1
0	1	0	$\bar{A}B\bar{C}$	m_2
0	1	1	$\bar{A}BC$	m_3
1	0	0	$A\bar{B}\bar{C}$	m_4
1	0	1	$A\bar{B}C$	m_5
1	1	0	$AB\bar{C}$	m_6
1	1	1	ABC	m_7

$f(A,B,C) = \bar{A}\bar{B} + AB\bar{C}$

$= \bar{A}\bar{B}(C+\bar{C}) + AB\bar{C}$

$= \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C} + AB\bar{C}$ ✓

$= (m_1 + m_0 + m_6)$ ∴ $m_0, m_1, m_6 = \Sigma(0,1,6)$

Maxterms:

$f(A,B,C) = (\bar{A}+\bar{B})(A+B+\bar{C})$

The second term contains all the three variables A and B and $\bar{C}$ in its complement form. This type is called standard sum or fundamental sum. It is designated with 'M'. It should produce a logical 0.

A	B	C	Term	Designation
0	0	0	$A+B+C$	M_0
0	0	1	$A+B+\bar{C}$	M_1
0	1	0	$A+\bar{B}+C$	M_2
0	1	1	$A+\bar{B}+\bar{C}$	M_3
1	0	0	$\bar{A}+B+C$	M_4
1	0	1	$\bar{A}+B+\bar{C}$	M_5
1	1	0	$\bar{A}+\bar{B}+C$	M_6
1	1	1	$\bar{A}+\bar{B}+\bar{C}$	M_7

$f(A,B,C) = (\bar{A}+\bar{B})(A+B)$

$= (\bar{A}+\bar{B}+C\bar{C})(A+B+\bar{C})$

$= (\bar{A}+\bar{B}+C)(\bar{A}+\bar{B}+\bar{C})(A+B+$

$= M_6, M_7, M_1$

$f(A,B,C) = \Pi(1,6,7)$

From truth table to canonical form:-

$F = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + AB\bar{C}$

$F = \Sigma(0,2,3,5,6)$

collecting the minterms for which F=0

$\bar{F} = \bar{A}\bar{B}C + A\bar{B}\bar{C} + ABC$

Taking complement on both sides and using DeMorgan's theorem:

$\bar{\bar{F}} = \overline{\bar{A}\bar{B}C + A\bar{B}\bar{C} + ABC}$

$F = \overline{\bar{A}\bar{B}C} \cdot \overline{A\bar{B}\bar{C}} \cdot \overline{ABC}$

$= (A+B+\bar{C})(\bar{A}+B+C)(\bar{A}+\bar{B}+\bar{C})$

$= (A+B+\bar{C})(\bar{A}+B+C)(\bar{A}+\bar{B}+\bar{C})$

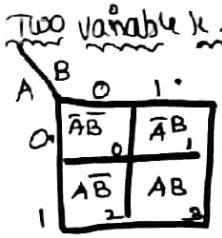
$= \Pi(1,4,7)$

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

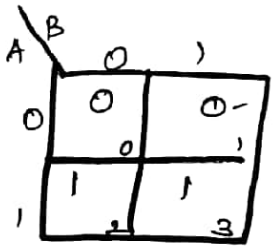
Relationship between K-map and truth table

The K-map can be rectangular or square shape and divided into number of square cells. The number of cells in a map depends of the number of variables. (ie) $2^2 = 4$ ce

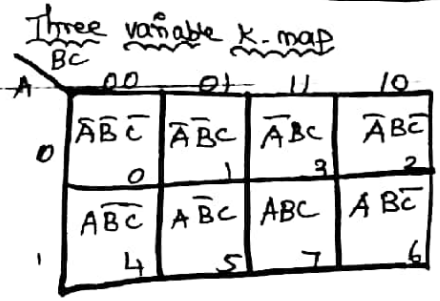
Two variable K-map

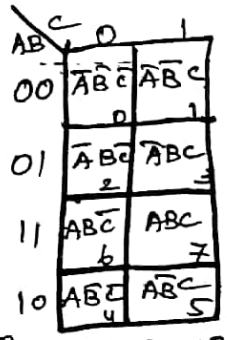


A	B	Y
0	0	0
0	1	0
1	0	1
1	1	1



Three variable K-map





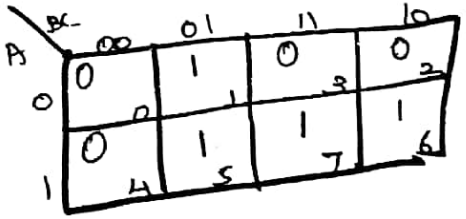
Here Instead of using normal binary progression, Gray code format is used. That is Bc is taken as 00, 01, 11, 10. This is done so that when we go from one cell to the next cell (Horizontal or vertical) there is only one change in binary combination.

Ex: From 01 to 11 in Gray code - one change
From 01 to 10 in binary - two changes

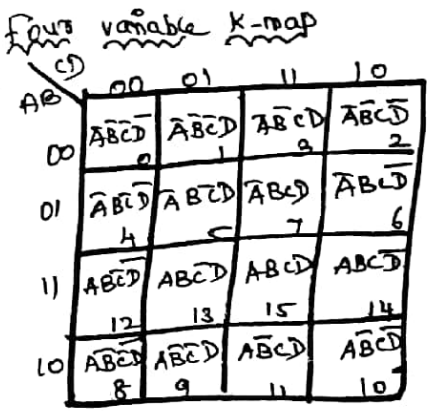
Ex:

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$Y = F(A,B,C) = \Sigma(1,5,6,7)$



Four variable K-map

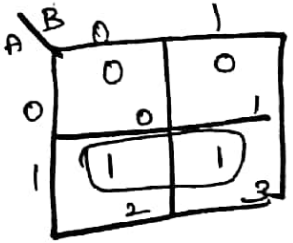


Five variable K-map

Ex:-

1)

A	B	Y
0	0	0
0	1	0
1	0	1
1	1	1



From the truth table, we can write SOP expression as

$$F(A,B) = A\bar{B} + AB$$
$$= A(\bar{B}+B)$$
$$= A(1) = \boxed{A}$$

* When a K-map contains two 1's that are horizontally adjacent or vertically adjacent then these two 1's form a Pair. The pair is encircled to indicate that they can give a reduced expression.

* In the above example, the pair of 1's cover the minterm m_2 and m_3 or $A\bar{B}$ and AB. This gets reduced to A.

* In K-map we get the reduced expression

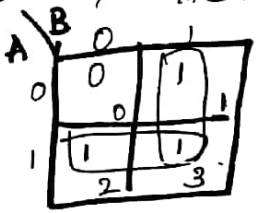
by simply examining the map. To do that we can follow simple procedure. Move from cell 2 to cell 3. For both the cells the variable A is common with a value A = 1.

* In this example, we find that when we move from cell 2 to cell 3 the variable B is changing from $\bar{B}$ to B. The variable that changes is eliminated. Therefore the simplified result for k-map is

$$F(A,B) = A$$

Overlapping of groups:

2) $Y = \bar{A}B + A\bar{B} + AB$



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

In this map the '1' in cell 1 is paired with cell 3. The 1's in cell 2 and 3 are paired. It means that the 1 in cell 3 helps form a pair with cell 1 and 2. This type of grouping is known as overlapping of groups.

For pair of cell 2 and 3, A is constant with 1 where B changes from 0 to 1 and B is eliminated.

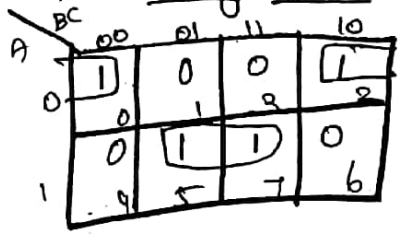
For pair 1 and 3, B is constant and A changes from 0 to 1.

$\therefore Y = A + B.$

Ex: 3) Draw a map for $F(A,B,C) = \Sigma(0,2,5,7)$

Solution:-

Here folding method can be used.



The 1's in cell 5 and 7 form a pair. The variable A and C=1 remains unchanged and B changes from 0 to 1

$\therefore$ The product obtained for this pair is AC.

The cell 0 and cell 2 are paired only the Variable B changes from 0 to 1. These two cells are not adjacent, they form a pair by process called folding or rolling the map. when the map is folded, the cells 0 and 2 become adjacent.

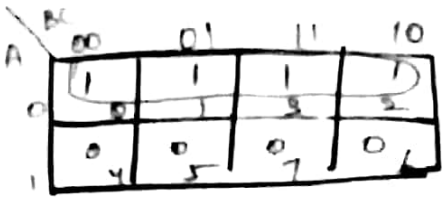
$P_1(5,7)$ gives AC

$P_2(0,2)$ gives $\bar{A}\bar{C}$

$$F(A,B,C) = AC + \bar{A}\bar{C}$$

Ex: 4) $f(A,B,C) = \Sigma(0,1,2,3)$

In a three variable k-map, while forming we are able to group four 1's together, then arrangement is called a quad. A quad eliminates two variables and their complement.



For all the four cells only the variable A remains unchanged with A = 0, cell or

$\therefore F(A, B, C) = \boxed{\overline{A}}$

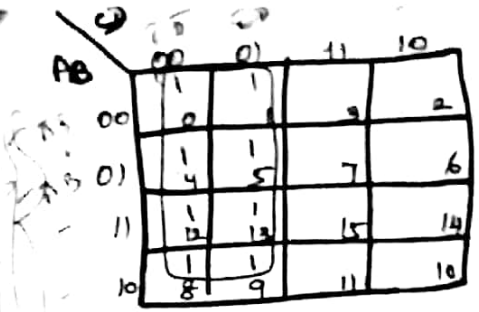
Steps for K-map :-

1) Draw the K-map and number the minterm cells.
2) Fill the map with 1's and 0's.
3) Group the 1's as quad and pairs, starting from bigger group.
4) Allow the groups to overlap, if necessary
5) Avoid redundant groups
6) Fold or roll the map if necessary to get bigger group.

Octet :-

In 4 variable K-map besides the pairs and quads a bigger group called octet. An octet is group with eight 1's. An octet eliminates three variables and their complements.

Ex: $F(A, B, C, D) = \Sigma (m0, m1, m4, m5, m8, m9, m12, m13)$



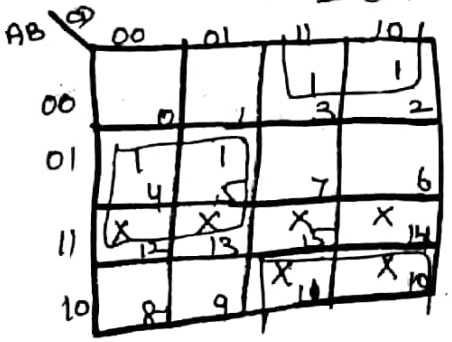
Octet $0 = (0, \ldots)$

$\boxed{\overline{F} \quad \overline{C}}$

DON'T care condition :-

The don't care condition can be used in K-map to our convenience to provide bigger group and hence more simplified expressions. The don't care condition is denoted with 'X' to differentiate from 0 and 1. the X's may be assumed as 0 or 1. In K-map minterms X can be taken as 1 and in max-terms X can be taken as 0. And also X can altogether be omitted.

$Y = F(A, B, C, D) = \Sigma (2, 3, 4, 5) + \Sigma d (10, 11, 12, 13, 14, 15)$



$Y = F(A, B, C, D) = \boxed{B\overline{C} + \overline{B}C}$

$Y = X$

TABULATION METHOD

When the number of variables is more than five, it is difficult to simplify because there will be a large number of terms in the expression, or too many cells. To solve this the other method called Tabulation method or Quine-Mccluskey method.

Procedure for Tabulation method:

The procedure involves four steps

* By careful search, find out terms those are necessary and required for simplification. These terms are called prime-implicants.
* From among the prime implicants, choose the ones which give the least number of literals.

* From given function, list all the minterms and get the binary equivalent. The minterms are arranged in different groups, the first group does not contain any 1's, the second group contains only one 1, third group contains two 1's etc.. The minterms in one group are compared with minterms in immediate next group.
* If two terms differ in only one variable, that variable is removed and term with one less literal is found. The remaining terms and all the terms that did not match during the process form the prime implicants.

Ex: Simplify by tabulation method

$$f(A,B,C,D) = \Sigma(0,1,4,5,10,11,14,15)$$

Solution:

1) Represent each minterm in binary.

Decimal	Binary
0	0000
1	0001
4	0100
5	0101
10	1010
11	1011
14	1110
15	1111

2) Step 2: Arrange the minterms in different groups, so that first group does not contain any 1, 2nd group only one 1, 3rd group two 1's etc..

0 - 0000 - 1st group

1 - 0001 }
4 - 0100 } - 2nd

5 - 0101 }
10 - 1010 } - 3rd

11 - 1011 }
14 - 1110 } - 4th

15 - 1111 - 5th.

3) The first column contains the minterms arranged as per step 2. The minterms (as) in one group are compared

with the minterms in the immediate next group. Any two minterms, which differ each other by only one variable can be combined and unmatched variable removed. Two minterms fits into this category if they both have the same bit value in all positions except one.

Column (a)		Column (b)	
	A B C D		A B C D
0	0000 ✓	(0, 1)	000 –
1	0001 ✓	(0, 4)	0 – 00
4	0100 ✓	(1, 5)	0 – 01
5	0101 ✓	(4, 5)	010 –
10	1010 ✓	(10, 11)	101 –
11	1011 ✓	(10, 14)	1 – 10
14	1110 ✓	(11, 15)	1 – 11
15	1111 ✓	(14, 15)	111 –

The minterm m0 and m1 is compared The minterm m0 (0000) and m1 (0001) are compared and the result is (0, 1) 000 –. A check mark is placed when the comparison completes.

4) Terms in one group are compared only with terms in the immediate next group. Also terms in each group need to be compared only if they have dashes in the same position. As before a check mark (✓) is placed for the terms those are used for further reduction of a variable.

Column (a)		Column (b)		Column (c)	
	A B C D		A B C D		A B C D
0	0000 ✓	(0, 1)	000 – ✓		
		(0, 4)	0 – 00 ✓	(0, 1, 4, 5)	0 – 0 –
1	0001 ✓			(0, 4, 1, 5)	0 – 0 –
4	0100 ✓	(1, 5)	0 – 01 ✓		
		(4, 5)	010 – ✓		
5	0101 ✓				
10	1010 ✓	(10, 11)	101 – ✓	(10, 11, 14, 15)	1 – 1 –
		(10, 14)	1 – 10 ✓	(10, 14, 11, 15)	1 – 1 –
11	1011 ✓				
14	1110 ✓	(11, 15)	1 – 11 ✓		
15	1111 ✓	(14, 15)	111 – ✓		

5) The unchecked terms in the table form the prime implicants.

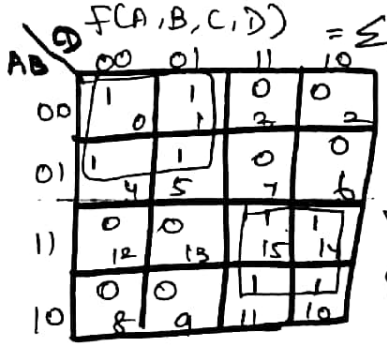
In this example all the terms in column (a) and (b) are checked. There are four unchecked terms in column (c). But two terms get repeated and as long as the term forms a prime implicants it is unnecessary to use the same term twice.

∴ The prime implicants are $\bar{A}\bar{C}$ corresponding to 0 0 – 0 – and Ac corresponding to 1 – 1 –

The sum of prime implicants gives the minimised function in SOP (ie) $F(A,B,C,D) = \bar{A}\bar{C} + AC$.

The same problem can be solved by K-map and verify

$F(A,B,C,D) = \Sigma(0,1,4,5,10,11,14,15)$



The 1st quad $Q_1(0,1,4,5)$ gives the reduced expression $\bar{A}\bar{C}$

The 2nd quad $Q_2(10,11,14,15)$ gives reduced expression AC.

$F = \bar{A}\bar{C} + AC$.

Ex (2) $F(A,B,C,D) = \Sigma(0,5,7,8,13,15)$

Solution:

1) First write the binary combination

0 → 0000, 5 → 0101, 7 → 0111, 8 → 1000, 13 → 1101, 15 → 1111.

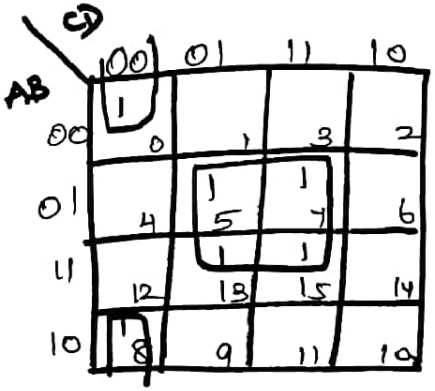
Column (a) ABCD	Column (b) ABCD	Column (c) ABCD
0 0000 ✓		
8 1000 ✓	(0,8) −000 ✓	
5 0101 ✓	(5,7) 01−1 ✓	
7 0111 ✓	(5,13) −101 ✓	(5,7,13,15) −1−1
13 1101 ✓	(7,15) −111 ✓	(5,13,7,15) −1−1
15 1111 ✓	(13,15) 11−1 ✓	

The first term in column (b) is unchecked. Similarly the two terms in column (c) are also unchecked. But from column (c), it is enough if we take any one of them since one of them is duplicate of the other.

The unchecked terms are −000 which corresponds to $\bar{B}\bar{C}\bar{D}$ and −1−1 which corresponds to BD.

∴ The final solution is $F = BD + \bar{B}\bar{C}\bar{D}$

$F(A,B,C,D) = \Sigma(0,5,7,8,13,15)$ by K-map



The map contains a quad and a pair

The quad Q(5,7,13,15) gives the reduced expression BD.

The pair P(0,8) gives the reduced expn $\bar{B}\bar{C}\bar{D}$.

∴ The final expression is

$F = BD + \bar{B}\bar{C}\bar{D}$

UNIT - III

Arithmetic building blocks:

Introduction:-

circuit consist of input variables, logic gates and output variables. The output of the combinational circuits depend only on the current value of the inputs.

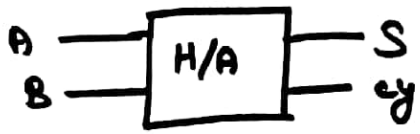
The block diagram of combinational circuit:

Arithmetic circuits:

* Binary addition
* Half adder and full adder
* Four bit binary adder
* BCD adder
* Half subtractor and full subtractor
* Four bit adder / subtractor circuit.

Half-adder:

Half adder is a logic circuit that adds 2 bits and gives the result on 2 output lines as Sum (S) and carry (cy). Let A and B be 2 inputs and said cy be the 2 outputs.



Block diagram of Half adder.

Since A and B can take values 0 and 1. The total number of possible input combination is 4.

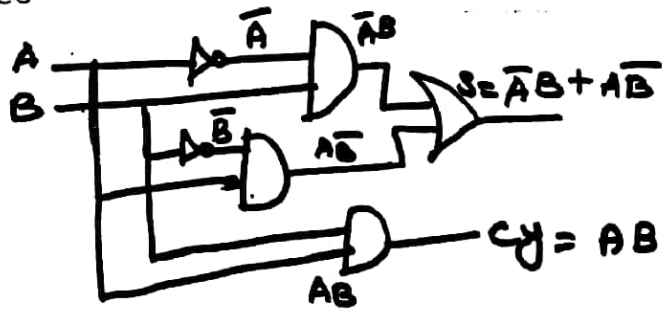
Truth table:

A	B	Sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

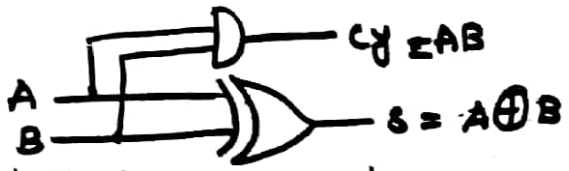
From the truth table the sum of product expressions for sum and carry can be written as

$$\text{Sum} = \bar{A}B + A\bar{B} \qquad \text{carry} = AB.$$

These expression can be implemented using the circuit

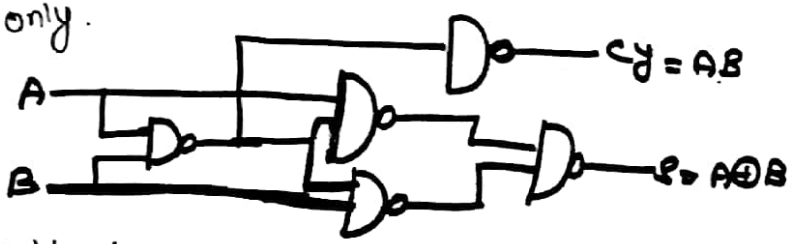


The sum expression is directly an Ex-OR expression, the Half adder circuit can also be drawn as



Half adder using NAND gates:

NAND is a universal gate, it is possible to implement the half adder circuit using NAND gates only.



Half Subtractor :-

A Half subtractor is a combinational circuit that subtracts 2 bits and generates difference D and Borrow B.

Truth table :

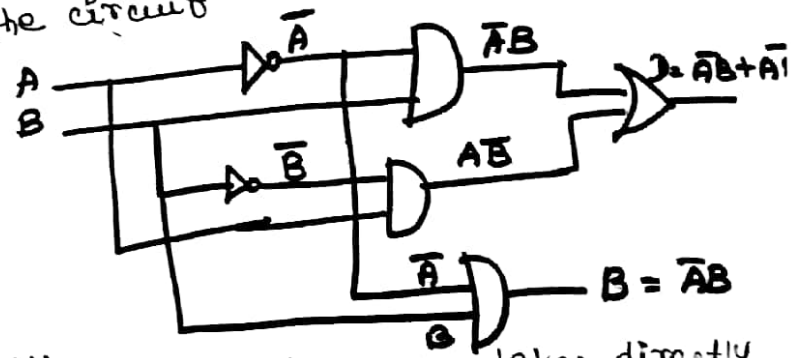
X	Y	Diff	Br
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

The block diagram of half subtractor

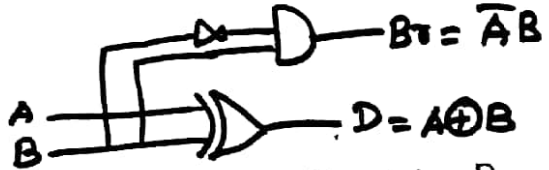
From the truth table the Boolean expression for diff and Br are found to be

$$Diff = \bar{A}B + A\bar{B} \qquad Br = \bar{A}B$$

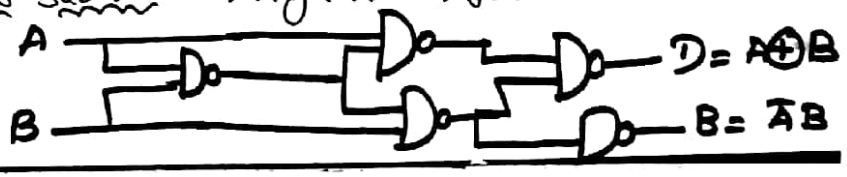
These 2 expressions can be implemented using the circuit



The difference output can be taken directly from an Ex-OR gate. The half subtractor can also be drawn as



Half subtractor using NAND gate is



Full Adder :-

A full adder is a combinational circuit. It is very useful when a carry input must be added to the 2 binary digits to obtain the correct sum. It consists of 3 I/P and 2 outputs. 2 Input variables are denoted by A and B and the 3rd Input 'C' represents the carry from the previous lower significant position. The two outputs are sum (S) and carry (Cy).

Block diagram :

Truth table of Full adder

A	B	C	S	Cy
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

From the Truth table the sum of product expression for sum and carry is

$$S = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$

$$Cy = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

$$S = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$

$$= (\bar{A}B + A\bar{B})\bar{C} + (\bar{A}\bar{B} + AB)C$$

Let $M = \bar{A}B + A\bar{B}$

$$\bar{M} = \overline{\bar{A}B + A\bar{B}}$$

$$= \overline{\bar{A}B} \cdot \overline{A\bar{B}}$$

$$= (A + \bar{B}) \cdot (\bar{A} + B)$$

$$= A\bar{A} + \bar{A}\bar{B} + AB + B\bar{B}$$

$$= 0 + \bar{A}\bar{B} + AB + 0$$

$$\bar{M} = \bar{A}\bar{B} + AB$$

$$\therefore S = M\bar{C} + \bar{M}C$$

$$= \boxed{M \oplus C}$$

But $M = \bar{A}B + A\bar{B} \Rightarrow A \oplus B$

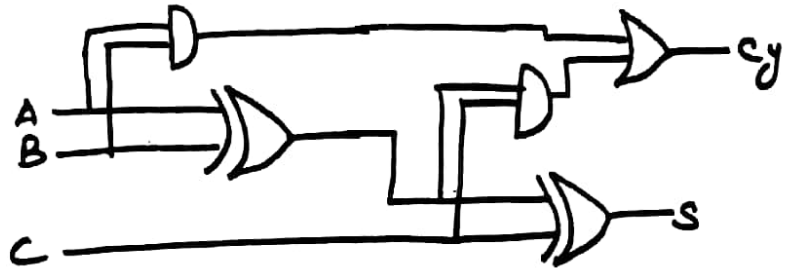
$$S = \boxed{A \oplus B \oplus C}$$

Similarly $Cy = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

$$= (\bar{A}B + A\bar{B})C + AB(C + \bar{C})$$

$$= \boxed{(A \oplus B) \cdot C + AB}$$

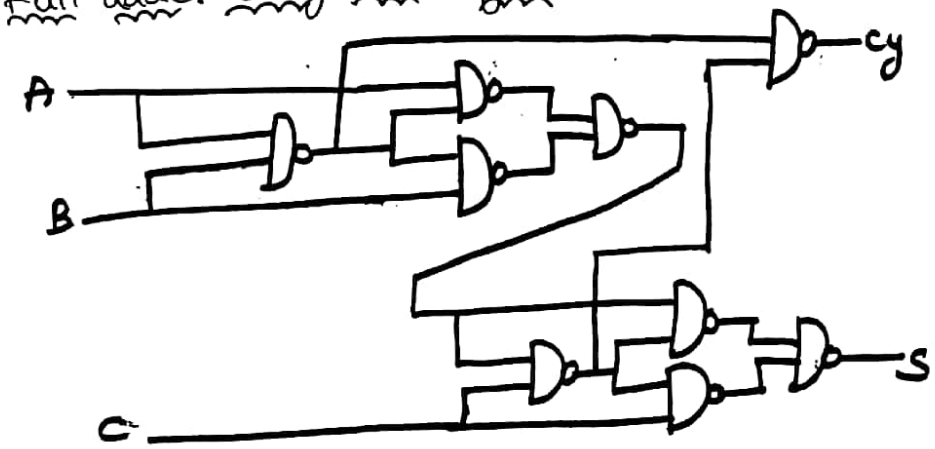
From the above expression the full adder circuit is as follows.



The full adder circuit can be made up of 2 half adders and an OR gate.

First half adder sum is added with the 3rd input C using 2nd half adder. The carry of of 1st and 2nd half adder are combined using an OR gates to produce a single final carry.

Full adder using NAND Gate:



Full Subtractor :-

A full subtractor is a combinational circuit that perform arithmetic subtraction, of 3 inputs. It consist of 3 inputs and 2 outputs. A and B represents the 2 significant bits to be subtracted. 3rd input C represents the borrow taken from the next significant position. The outputs are Different and borrow.

Block diagram:

Truth table:

A	B	C	Diff	Br
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

The SOP expression for the Diff and Borrow is

$Diff = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$

$Br = \bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC + ABC$

The difference is same as the one for the sum expression for the full adder.

So $Diff = A \oplus B \oplus C$

$Diff = \Sigma(1,2,4,7)$ $Br = \Sigma(1,2,3,7)$

The K-map for Br is

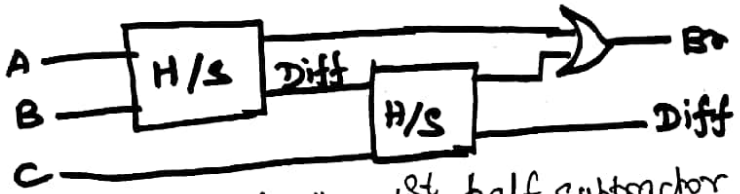
A \ BC	00	01	11	10
0	0	1	1	1
1	0	0	1	0

The 1's in the K-map for Br combine to give 3 pairs and gives a reduced expression

$$Br = \bar{A}B + Bc + c\bar{A}$$

A full adder can be converted into a full subtractor with the addition of only one NOT gate.

A full subtractor can also be implemented using 2 half subtractor and one OR gate.

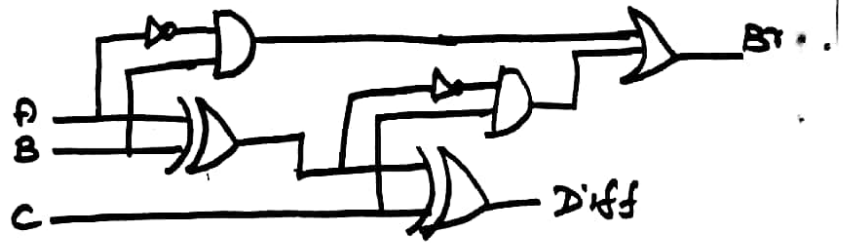


The difference of the 1st half subtractor is Subtractor with the 3rd Input using 2nd Half Subtractor. The 1st and 2nd half subtractor's borrow combined using an OR gate to produce a single final Br output.

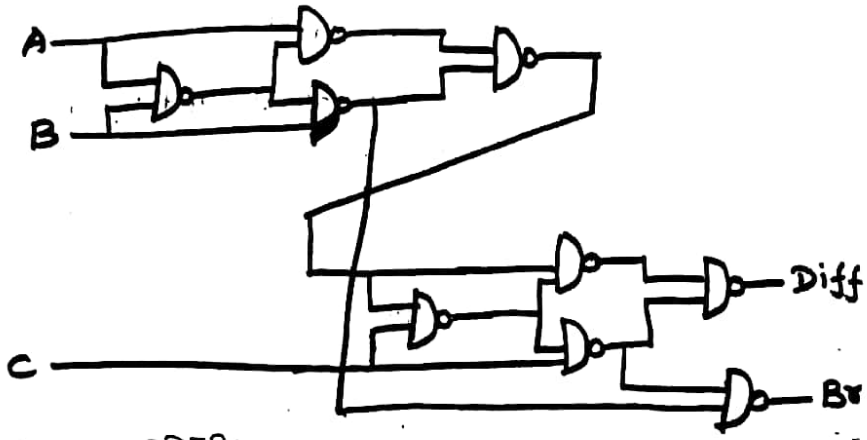
From the truth table

$$Br = \bar{A}\bar{B}c + \bar{A}B\bar{c} + \bar{A}Bc + ABc$$
$$= (\bar{A}\bar{B} + AB)c + \bar{A}B(c + \bar{c})$$
$$= (\overline{A \oplus B})c + \bar{A}B$$

using the above expression we can construct the full subtractor circuits



NAND using full subtractor:

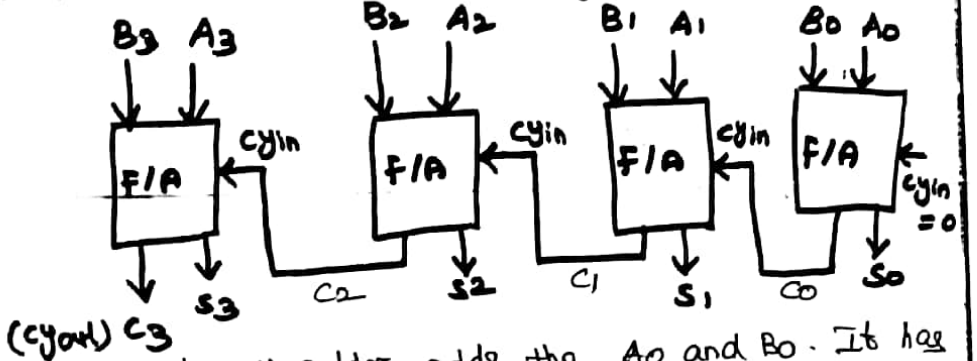


FOUR-BIT ADDER:-

A 4-bit adder adds two 4-bits numbers. This is represented as

$$\begin{array}{llllll} & A_3 & A_2 & A_1 & A_0 & \rightarrow \text{Augend} \\ (+) & B_3 & B_2 & B_1 & B_0 & \rightarrow \text{Addend} \\ \hline & S_3 & S_2 & S_1 & S_0 & \rightarrow \text{Sum} \end{array}$$

The 4-bit adder requires 4- full adders. Each full adder adds 3 bits A, B, c and Produces 2 Outputs as S (sum) and cy (carry).



The 1st full adder adds the A0 and B0. It has a carry input Cyin it is kept in '0' state. the 1st circuit outputs sum S0 and carry C0.

The 2nd full adder adds A1 and B1 and the carry C0 from the previous addition. similarly the 3rd adder and 4th full adder adds their input bits and carry from the previous addition. output is a 4-bit sum S_3, S_2, S_1, S_0 and a carry Cyout.

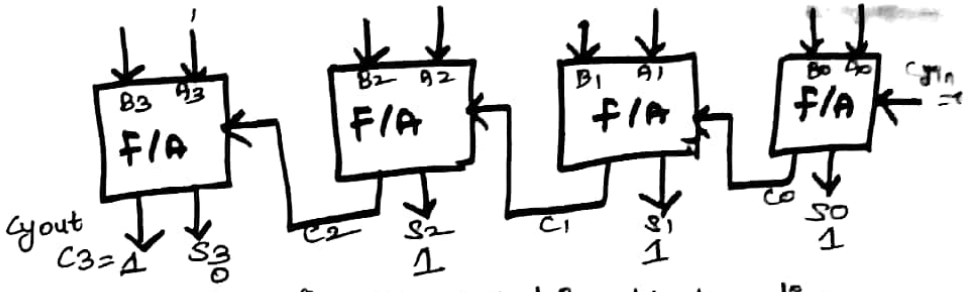
Example : Add 9 TO 14.

$A_3\ A_2\ A_1\ A_0 = 1\ 0\ 0\ 1 \rightarrow (9)_{10}$

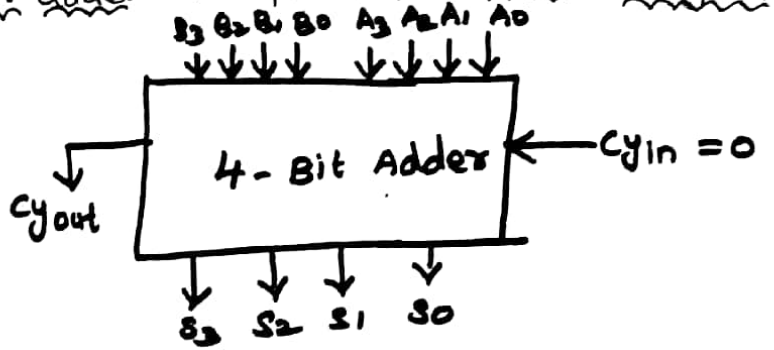
$B_3\ B_2\ B_1\ B_0 = 1\ 1\ 1\ 0 \quad (14)_{10}$

$1\ 0\ 1\ 1\ 1 \rightarrow (23)_{10}$

16 8 4 2 1

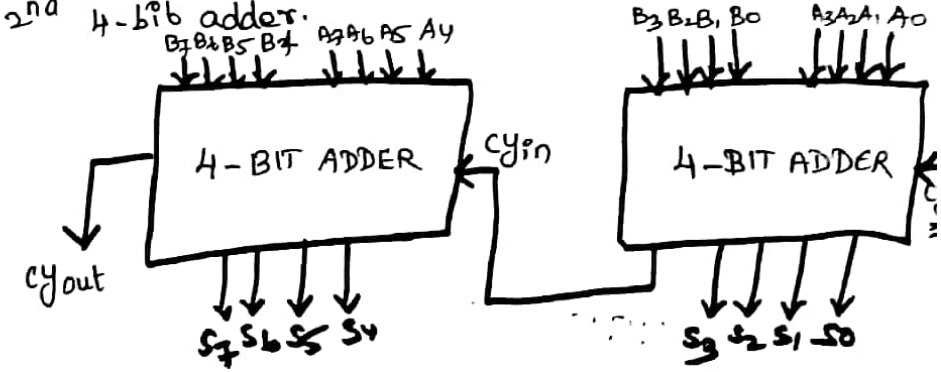


4- bit adder is represented in block - diagram



8- bit adder :

combining two 4-bit adder circuits, an 8- bit adder circuit can be constructed. Then the Cyout of the 1st 4-bit adder will act as Cy of Input to the 2nd 4-bit adder.



BCD ADDER:-

A BCD adder adds two BCD numbers and gives the sum in BCD with a possible carry. We know that in BCD we have only ten combinations from 0000 - 1001 i.e (0-9).

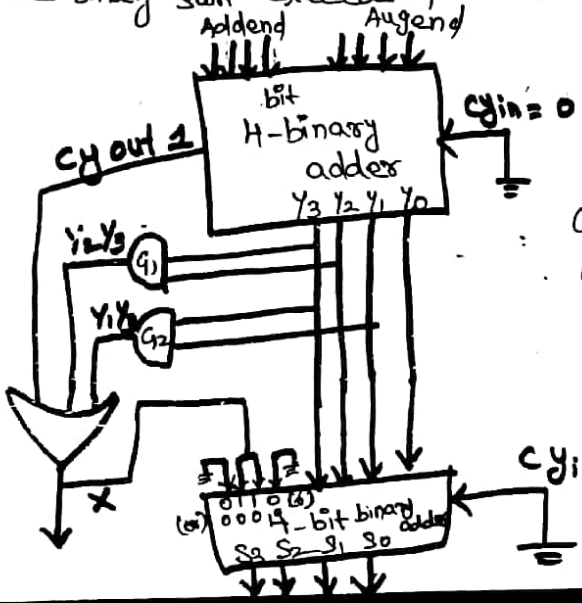
Example: ADD $(9)_{10}$ with $(8)_{10}$

9+	1001
8	1000
17	10001 ; valid BCD with a carry
	0110 ; Add 6
	10111 ; corrected BCD for 17 - 0001 0111.

A correction factor of 6 must be added to the result

(i) If the result in lower 4 bits is greater than 9 (or)
(ii) if the initial addition produces a carry.

A BCD adder adds two BCD digits in parallel and produces a sum digit also in BCD. The BCD adder must include the correction logic (ie) add 0110 (6) to binary sum whenever the binary sum exceeds 9 or if there is a carry.



when adding BCD, Invalid combinations

	Y_3	Y_2	Y_1	Y_0
(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

when any one of invalid combination occurs at the 4 bit it must be detected so that correction factor of 0110 is added. The correction factor must be added also when a carry is produced. $X = Y_3Y_2 + Y_3Y_1 + C_{yout}$

where X is the output of the combination circuit. It requires 2-I/P gate and one 3-I/P OR gate.

FOUR-BIT SUBTRACTOR:-

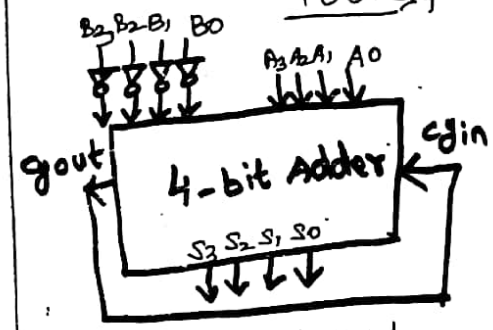
Subtraction by 1's complement using adder:

i) Take the minuend as it is.
ii) Take 1's complement of subtrahend
iii) Add them and also add the end carry.
iv) If there is no carry, it is -ve result and it is in 1's complement form.

$A = A_3A_2A_1A_0$, $B = B_3B_2B_1B_0$

Ex: subtract 4 from 13

4 → 0100
13 → 1101
4 → 0100 → 1's comp 1011 (+)
1 ← 1000
1001 → 9



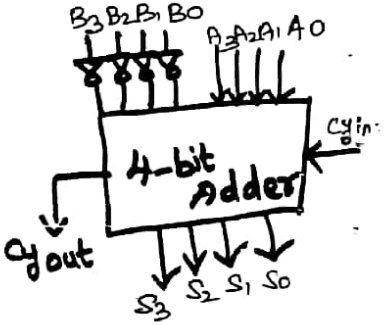
1's complement

Subtraction by 2's complement using adder:

i) Take the minuend as it is
ii) Take 2's complement of subtrahend.
iii) Add them and omit the carry.

Ex: subtract 4 from 13

13 → 1101 → 1101
4 → 0100 → 1100 (+)
1011 - 1's comp 1001 → 9.
1 ← 2's Leave carry
1100



2's complement

Combinational circuit Applications

* Multiplexer
* Demultiplexer
* Encoder
* Decoder

Multiplexer :-

* A multiplexer (data selector) has 2^n inputs and one output. Any one of the input can be brought out to the output.

For example, a 2-input multiplexer has 2 inputs, one o/p and one select line ($2^1 = 2$).

* A 4-I/p multiplexer has 4 input one o/p and 2 select lines ($2^2 = 4$).
* Similarly A 8-I/p multiplexer has 8 input one o/p and 3 select lines ($2^3 = 8$)

2 I/p multiplexer

Function table

S_0	Y
0	I_0
1	I_1

→ Where I_0, I_1 are Inputs, Y is an output line. S_0 is a selection line.

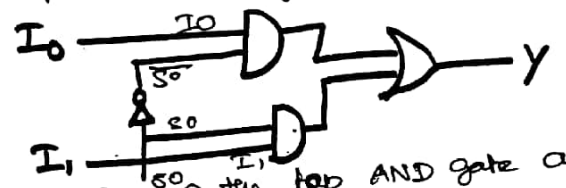
→ When $S_0 = 0$ the Input I_0 is selected & $Y = I_0$

→ When $S_0 = 1$ the input I_1 is selected and $Y = I_1$.

The o/p Y can be expressed as

$$Y = \bar{S}_0 I_0 + S_0 I_1$$

The 2 I/p multiplexer logic circuit Implemented using AND, OR and NOT gate

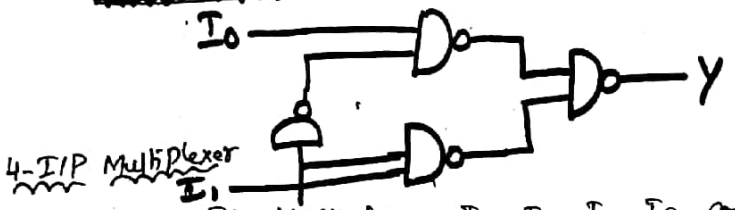


When $S_0 = 0$ the top AND gate allow I/p I_0 to pass on while the bottom AND gate gives the o/p and '0' as

$\therefore Y = I_0$. Similarly when $S_0 = 1$ the bottom AND gate allow I/p I_1 to pass on while the bottom AND gate gives the o/p as '0'. $\therefore Y = I_1$.

The circuit is also called as a 2-line to 1 line data selector.

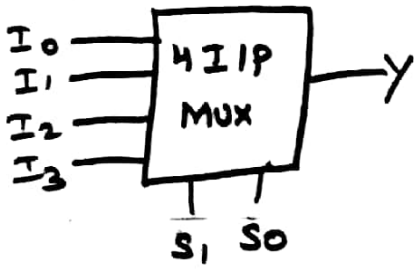
NAND CIRCUIT



4-I/P Multiplexer

* In 4 I/p Multiplexer I_0, I_1, I_2, I_3 are the 4 I/ps Y is the o/p. S_1 & S_0 are 2 select lines.

Block diagram

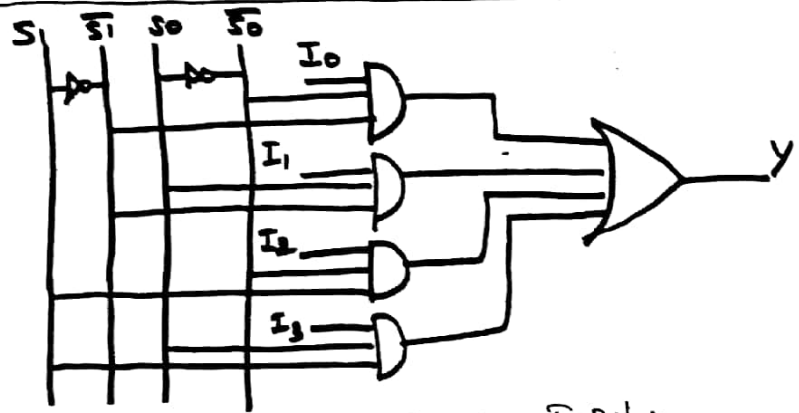


Function table:

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

From the function table the boolean expression is

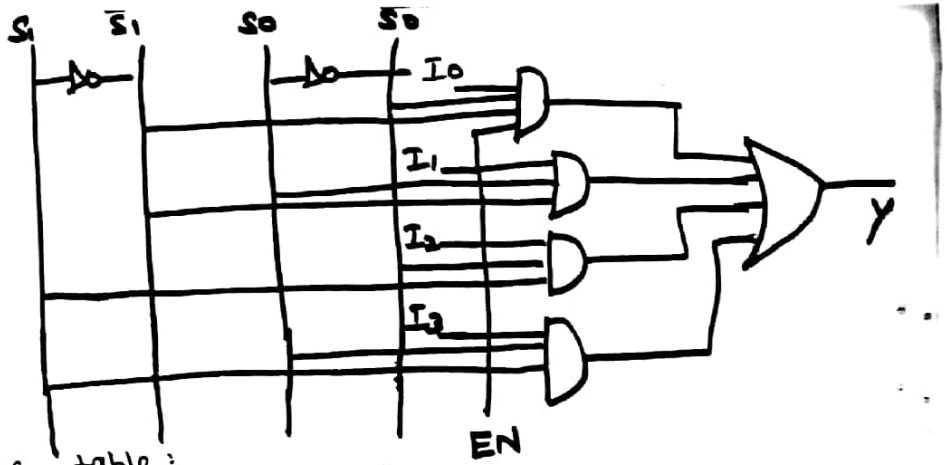
$$Y = \bar{S_1}\bar{S_0}I_0 + \bar{S_1}S_0I_1 + S_1\bar{S_0}I_2 + S_1S_0I_3$$



Multiplexers with Strobe / Enable Input :-

In Multiplexer one of the Input is transferred to the output. The Inputs is selected by the select lines. when we combine a number of multiplexers and enable only one of the multiplexer. Therefore a strobe/enable terminal (EN) is included in some multiplexer chips.

Logic diagram for Strobe / Enable Input and function table for 4-Input multiplexer is given below.



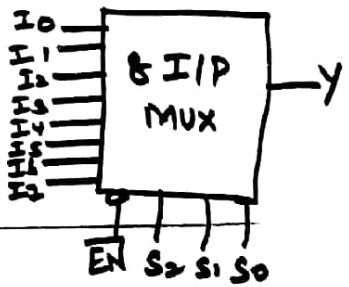
Function table:

EN	S_1	S_0	Y
0	X	X	0
1	0	0	I_0
1	0	1	I_1
1	1	0	I_2
1	1	1	I_3

When the Strobe terminal is in '0' state, O/p of all the AND gates becomes '0' & the O/p Y is also 0 & this is independent of the select inputs S_1 & S_0 In this condition multiplexer is disabled. When Strobe 4 I/p is in '1' state all the 4 AND gates are enabled and the circuit functions normally as a multiplexer.

The Strobe input active high or low the multi-plexer is enabled when Strobe is high and disabled when Strobe is low. on the other hand, if the Strobe is active low, then the multiplexer is enabled when Strobe is low and disabled when Strobe is high.

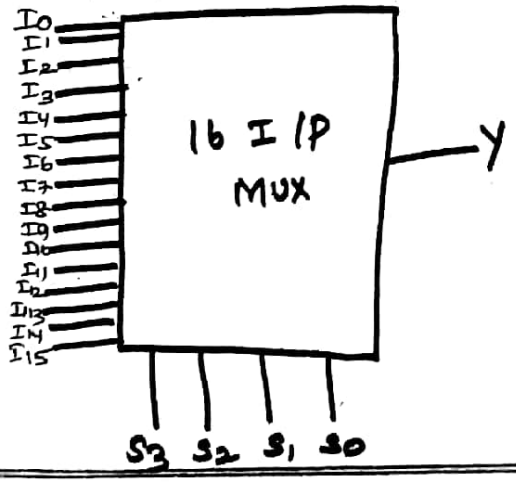
8- Input Multiplexer



$\overline{EN}$	S_2	S_1	S_0	Y
1	X	X	X	0
0	0	0	0	I_0
0	0	0	1	I_1
0	0	1	0	I_2
0	0	1	1	I_3
0	1	0	0	I_4
0	1	0	1	I_5
0	1	1	0	I_6
0	1	1	1	I_7

The multiplexer has 8 I/ps $I_0, I_1, \ldots\ldots I_7$, one o/p Y and 3 select lines S_0, S_1, S_2. It has active low strobe input $\overline{EN}$.

16- I/P Multiplexer.



DEMULTIPLEXER

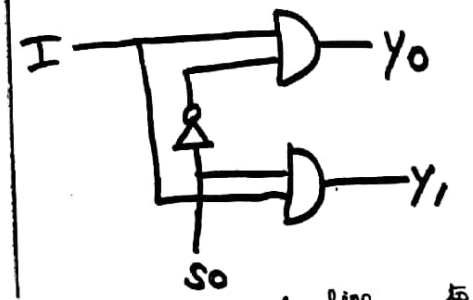
It is reverse of multiplexers. It has one I/P 2^n O/P and n select line or it is one to many circuit.

1- line to 2 - line demultiplexers.

S_0	Y_0	Y_1
0	I	0
1	0	I

When I is I/P, Y_0, Y_1 are 2 O/P lines & S_0 is a select line.

Logic diagram



When $S_0 = 0$ Input I is passed on to the O/P Y_0. When $S_0 = 1$ Input I is passed on to the O/P

1- line to 4 - line demultiplexer

This type of multiplexer will have one input 4 O/Ps and 2 select lines.

S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	I	0	0	0
0	1	0	I	0	0
1	0	0	0	I	0
1	1	0	0	0	I

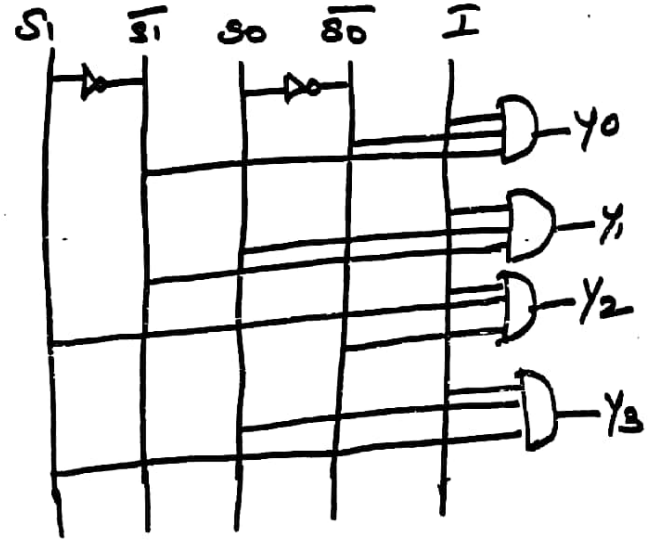
When $S_1 = 0$ and $S_0 = 0$ then $Y_0 = I$

$S_1 = 0$ and $S_0 = 1$ then $Y_1 = I$

$S_1 = 1$ " $S_0 = 0$ then $Y_2 = I$

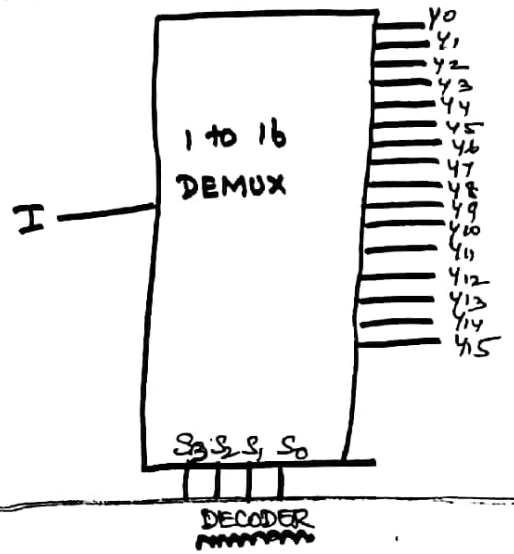
$S_1 = 1$ " $S_0 = 1$ then $Y_3 = I$

LOGIC DIAGRAM



1 line to 16 Demultiplexer

Here the working principle is same as previous circuits. It has one input, sixteen outputs & 4 select lines



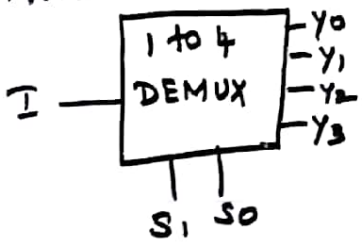
It has n address lines and 2^n o/ps for a given address only one of the output goes high and remaining o/ps remain in low state. In this case the decoder is said to have active high outputs. In this case only one output goes low while the remaining o/ps are in the high state.

2 to 4 Decoder

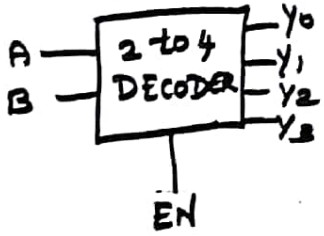
A one line to 4-line multiplexer acts as a 2 to 4 Decoder, with the I/p line of the demultiplexer acting as an enable input. The I/p is now taken as enable EN and the circuit is enabled for a logic 1. The select lines

S_1, S_0 are taken as address input A B.

1 to 4 DeMultiplexer



2 to 4 Decoder

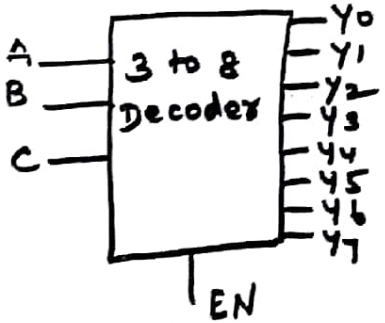


EN	A	B	Y_0	Y_1	Y_2	Y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	X	X	0	0	0	0

When EN = 1 only one of the output goes to '1' state (active high), while the remaining O/ps are in '0' state.

3 to 8 Decoder

It has 3 address I/ps ABC and Eight active low (or) active high outputs. $Y_0, Y_1 \ldots\ldots Y_7$ For the given address only one of the eight O/ps goes low while the remaining Seven outputs are in high state.



$\overline{Y}$ is the output

n inputs = 2^n outputs

EN	A	B	C	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
1	0	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	0	1	1	1	1	1	1
1	0	1	0	1	1	0	1	1	1	1	1
1	0	1	1	1	1	1	0	1	1	1	1
1	1	0	0	1	1	1	1	0	1	1	1
1	1	0	1	1	1	1	1	1	0	1	1
1	1	1	0	1	1	1	1	1	1	0	1
0	X	X	X	1	1	1	1	1	1	1	0

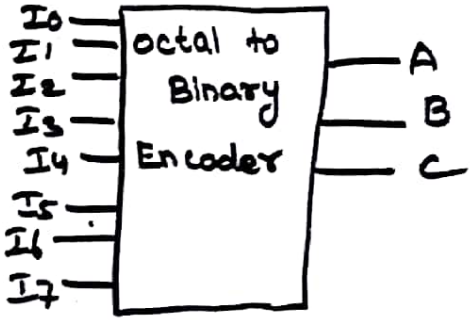
ENCODER

An encoder is a logic circuit that produces a reverse operations of a decoder. For example a 3 to 8 decoder has 3 inputs and 8 outputs. The Encoder has 8 Inputs and 3 outputs. The I/p level can be active high or active low. Depending on the I/p that is active the O/p produces an equivalent 3 Bit binary equivalent.

$8 = 2^{(3)}$

n inputs = m outputs where $n = 2^m$

CIRCUIT



Functiontable

I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

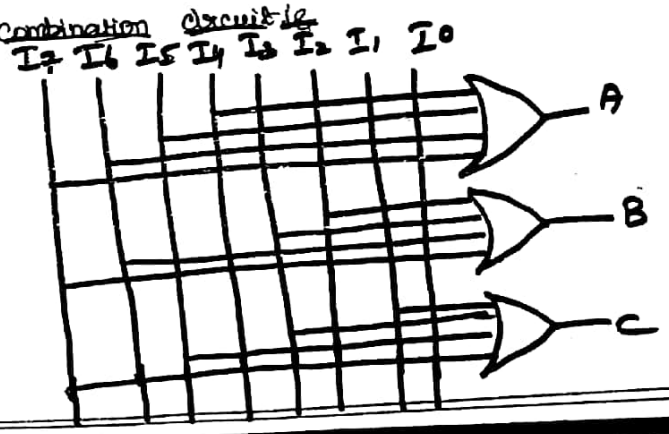
From the function table output line A goes to '1' state when I_4, or I_5 or I_6 or I_7 is active. That is

$A = I_4 + I_5 + I_6 + I_7$

Similarly $B = I_2 + I_3 + I_6 + I_7$

$C = I_1 + I_3 + I_5 + I_7$

The combination circuit is



UNIT-IV FLIP-FLOPS

Combinational circuits consist of finite number of Input and one (or) more outputs. One circuit outputs becomes the input of another circuit is called Sequential circuit. The basic building block of sequential circuit is called a flip-flop in digital electronics.

Types of FLIP-FLOP

* NAND Latch
* SR Flip Flop (SET-RESET)
* D Flip Flop. (DELAY)
* JK Flip Flop. (J-SET / K-RESET)
* T Flip Flop (TOGGLE)

INTRODUCTION :-

A Flip-Flop is a bi-state device ie, a circuit with only 2 stable states, namely, the '0' state and the '1' state. It can store a bit of Information. A basic Flip-Flop circuit is a basic memory circuit or a one bit memory call. A Flip Flop o/ps always complement each other.

A circuit based on Flip-Flops usually paused or clocked. When the output goes from 0 to 1 state and comes back to 0 state is known as pulse. The continuous pulses are called a pulse train. The Pulses are called clock pulse or clock (CLK). It is denoted by $f = 1/T$ { f = frequency, T = Regular period.

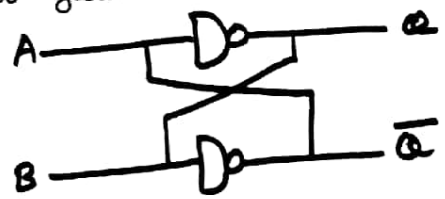
single pulse

Pulse Train.

NAND LATCH

A basic filp-flop or one bit memory cell is formed by cross coupling 2 NOT gates G_1 and G_2. The o/p of each gate is connected to the input of the other gate.

Circuit:



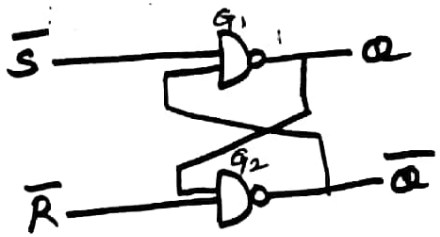
Truth table for NAND gate

A	B	$Y=\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

The o/p of gate 1 is Q is connected to the input of gate$_2$ (G_2). The o/p of the Gate G_2 is $\bar{Q}$. If $Q=0$ then $\bar{Q}=1$ and when $Q=1$ then $\bar{Q}=0$.

It store one bit of information it is called as 1-bit memory cell. This circuit is called as a LATCH. This circuit is modified using two 2-inputs NAND gates. and input of each NAND gate is used for the feed-back connection. This circuit is called as a NAND LATCH

Let $\bar{S}$ and $\bar{R}$ be I/p of the NAND LATCH and let Q, $\bar{Q}$ be the o/p of the latch.



$\bar{S}$	$\bar{R}$	Q	
0	0	?	Forbidden / Not allow
0	1	1	Set
1	0	0	Reset
1	1	Q	No Chang

Case 1:

$\bar{S}=0$ $\bar{R}=0$

The Input of both the gates are '0' So out of the G_1 and G_2 is 1, which is ∴ forbidden because $\bar{Q}$ must be always complement of Q. This state is forbidden or prohibited or indeterminat state. This state is not allowed.

Case 2:-

$\bar{S}=0$ $\bar{R}=1$

The output Q of G_1 is 1. This o/p Q is taken as input to the second NAND gate G_2. Now the IInd NAND Gate inputs are in 1 State. ∴ the output is 0. when its o/p goes to 1 the latch is said to be Set.

Case 3:-

$\bar{S}=1$ $\bar{R}=0$

The O/P of G_2 is 1 for G_1 both the i/p are 1 an the o/p of G_1 is 0 ∴ $Q=0$ and $\bar{Q}=1$ when the o/p goes '0' state, the latch is said to be reset

Case 4:

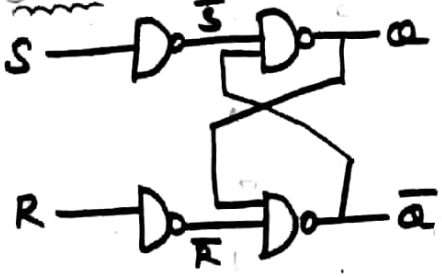
$\bar{S} = 1 \quad \bar{R} = 1$

When both the Inputs are 1 state the output is also 1 so there is no change.

SR FLIP FLOP

Add 2 NOT gates (G_3 and G_4) in front of the NAND LATCH is known as SR flip-flop.

Circuit diagram:



Block diagram

S	R	Q	
0	0	Q	No change
0	1	0	Reset
1	0	1	set
1	1	?	Not allowed

Case 1:

When S=0 R=0 we get $\bar{S}=1$ $\bar{R}=1$ from the NAND Latch truth table the output is - No change.

Case 2:

When S=0, R=1 we get $\bar{S}=1$, $\bar{R}=0$ from the Nand Latch truth table output Q=0.

Case 3:

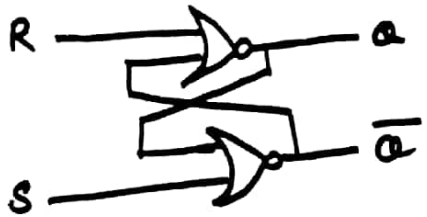
When S=1, R=0 we get $\bar{S}=0$, $\bar{R}=1$ from the NAND Latch truth table output Q=1.

Case 4:

When S=1, R=1 we get $\bar{S}=0$, $\bar{R}=0$ so that the output Q is Not allowed.

SR flip-flop using NOR Gate

An SR flip-flop can also be constructed using NOR gates with cross-coupled connections from the output of one gate to the input of the other gate.



R	S	Q	
0	0	Q	No change
0	1	1	Set
1	0	0	Reset
1	1	?	Not allowed

Case 1:

R=0 S=0

The output cannot be determined directly from the truth table NOR Gate. If Q=0 It continues to remain as 0. If Q=1 initially the Q continues to remain as 1

∴ The output is No change.

Case 2:

R=0 S=1

If one of the Input is in '1' state, its output goes '0' state. Since S=1 The output $\bar{Q}=0$ it goes to input of the

∴ The output of G_1 is Q=1 When R=0, S=1, Q=1. It is S

Case 4:

$\bar{S} = 1$ $\bar{R} = 1$

When both the inputs are 1 the output from the NOR Gate constructed using G1 output is 0. Now is also 1 so this goes to the 2. The output of

reset condition

A

N

P

... the gates go to 0 which ... that Q and $\bar{Q}$ must complement each other. Hence R=1 and S=1 is a forbidden condition.

CLOCKED S-R FLIP FLOP

All sequential circuits operate in synchronization with some form of clock pulses. In SR flip flop and 2 AND gates at the S and R input they are used to enable or disable the flip-flop.

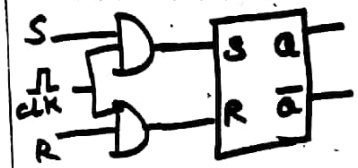
One of I/p of both the AND gates are connected together and taken as an enable or clock input (CLK). The 2nd inputs of the AND gates are again named as S and R when the CLK input is low, the output of both the AND gate is '0'. This '0' is passed to S and R.

When S=0, R=0, the flip flop will be in 'no change' condition which makes the outputs Q and $\bar{Q}$ to retain the previous

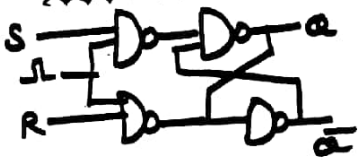
S R flip flop is disabled when the CLK input is low.

When the CLK inputs goes high, S and R values are pass on to the SR flip flops and depending on the SR values the O/p is modified. In other words the SR flip flop is enabled when the CLK is high.

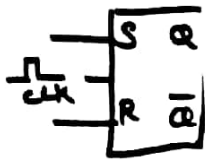
Block diagram



Logic circuit



Block diagram of clocked

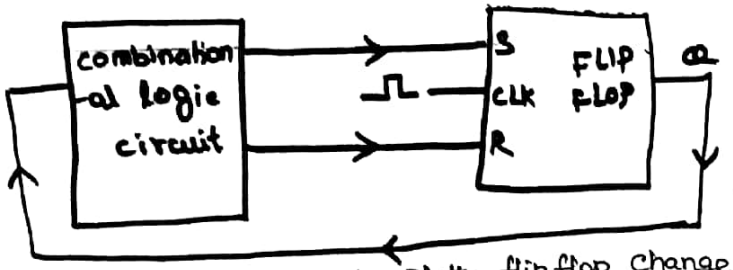


FUNCTION TABLE

CLK	S	R	Q	
0	X	X	Q	No Change
1	0	0	Q	No Change
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	?	Not allowed.

EDGE TRIGGERED FLIP-FLOPS

In the design of sequential circuits, combinational circuits are combined with flip-flops, the o/p of a combinational circuit is given as inputs to a flip-flop like the SR flip flop.



In this circuit the o/p of the flip flop change state depending on the S and R values only when a clock pulse is applied. The output of the flip-flop taken as input of the combinational circuit. Now this circuit gives a new output which will alter the values of S and R. The SR flip flop should respond only to the next pulse and give a new output.

The flip flops respond only during the rising edge or falling edge of the clock pulse and not during the period when the pulse remains high. The flip flops are then called as EDGE TRIGGERED FLIP FLOP

Positive Edge Trigger

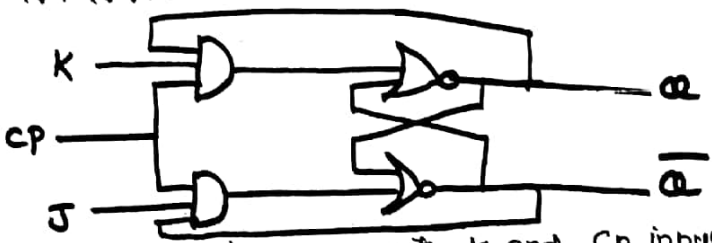
Negative Edge Trigger

The +ive edge trigger responds during the rising edge of the clock pulse. A -ive edge triggered flip flop responds ~~to the clock pulse~~ during the falling edge of the clock pulse.

JK FLIP FLOP

A JK Flip flop is a refinement of the RS flip-flop Inputs J and K behave like inputs S and R to set and clear the flip flop. When inputs are applied to both J and K simultaneously, the flip flop switches to its complement state, that is If Q=1. It switches to Q=0. and vice versa

A clocked JK flip flop:



output Q is AND with K and Cp inputs. So that the flip-flop is cleared during a clock pulse only if Q was previously 1. similarly output $\bar{Q}$ is ANDed with J and Cp inputs. so that the flip-flop is set with a clock pulse only if $\bar{Q}$ was previously 1.

Graphic symbol: (or)

Block diagram:

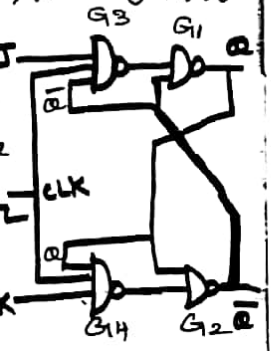
Characteristic table

Q	J	K	$Q_{(t+1)}$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Truth table

CLK	J	K	Q_n	Q_{n+1}	
0	X	X	X	Q	No change
1	0	0	0	0	No change
1	0	0	1	1	
1	0	1	0	0	Reset
1	0	1	1	0	
1	1	0	0	1	set
1	1	0	1	1	
1	1	1	0	?	Racing
1	1	1	1	?	

JK using NAND



Racing: $T_P << T_W$ (T_P - Propagation delay, T_W - Pulse width). The Clock pulse changes thousands of time so the output of JK can not predicted. This is called Racing or Race-around.

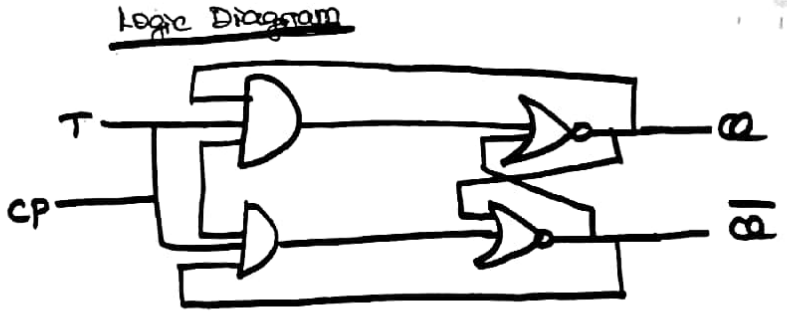
From the function table the JK flip-flop behave like an RS Flip-Flop, except when both J and K are equal to 1. the clock pulse is transmitted through one AND gate only - the one whose input is connected to the flip-flop output which is presently equal to 1.

If Q = 1 the output of the upper AND gate becomes 1 upon application of a clock pulse, and the flip flop is cleared. If Q = 1 the O/p of the lower AND gate becomes 1 and the flip-flop is set. In either case the output state of the flip-flop is complemented.

T - FLIP - FLOP

The T-Flip-Flop is a single input version of the JK Flip Flop. T stands for a Toggle flip-flop. A JK Flip-Flop can be used to construct a T flip flop. A T flip flop has one I/p which is connected to both J and K inputs of JK Flip Flop.

Logic Diagram



Function Table

CLK	T	J	K	Q	
0	X	X	X	Q	No change
1	0	0	0	0	No change
1	1	1	1	$\bar{Q}$	Toggle

When T = 0 the input J and K will also be equal to The output of the T flip-flop will be in 'no-change' cond when a clock pulse is applied.

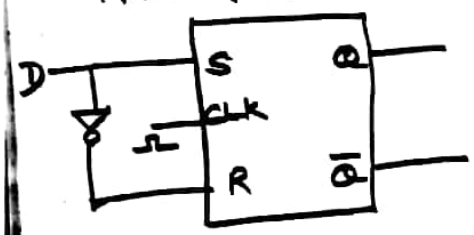
When T = 1 J and K will also be equal to 1. The output of the toggle from its previous state.

D - FLIP - FLOP

A Flip Flop which ~~only~~ has only one input and which transfers the input data to the output on application of a clock pulse is called a D-Flip-Flop.

The D Flip Flop can be constructed with a slight modification in clocked SR Flip-Flop. A NOT gate connect between the S and R inputs, so that $R = \bar{S}$. S Input taken as D input.

Block diagram



Function table:

CLK	D	Q	
0	X	Q	No change
1	0	0	
1	1	1	

When CLK input is '0', the output of SR flip flop is no-change. When CLK input is '1', the output of the flip flop depends on input D.

If D = 0 then S = 0 and R = 1 which is a reset condition that makes Q = 0.

If D = 1 then S = 1 R = 0 which is a set condition that makes Q = 1.

MASTER-SLAVE FLIP FLOP:

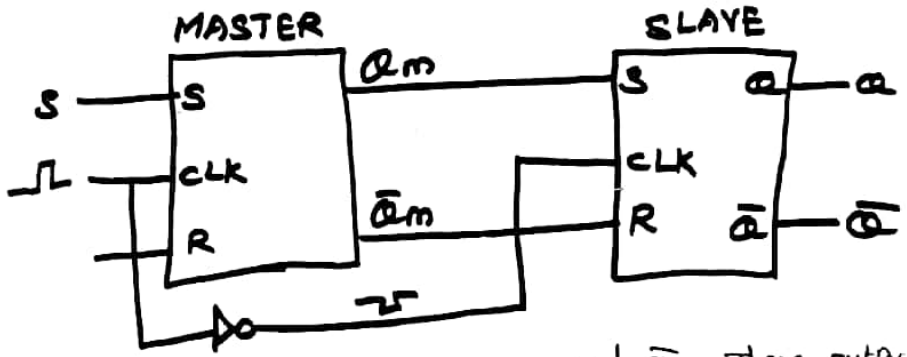
A master-slave flip-flop is constructed from 2 separate flip flops. One circuit serves as a master and the other as a slave, the overall circuit is referred to as a master-slave flip flop.

Master-slave SR flip-flop:-

SR flip-flop can be converted to master-slave SR flip flop by connecting two SR flip-flops one another so that one of them acts as a master and the other acts as a slave.

They follow SR truth table. Both SR respond when the clock applied to them goes from 0 to 1. When connected in master-slave mode, the clock is applied directly to master and the clock is inverted and applied to the slave.

By this arrangement, the master-SR responds when CLK pulse rises (0 to 1) and slave responds to (1 to 0). The final output is obtained from output of the slave. The final output is obtained at end of the pulse or during the negative edge this circuit is called as negative edge triggered SR flip flop.



The output of the master Q_m and $\bar{Q}_m$. These outputs are directly connected to S and R inputs of slave. ∴ The slave, the S and R inputs always complement each other. (ie) when S=0, R=1 and when S=1, R=0. There are no other combinations for the slave.

→ Let S=1 and R=0 for the master, the pulse is applied during rising edge the master respond

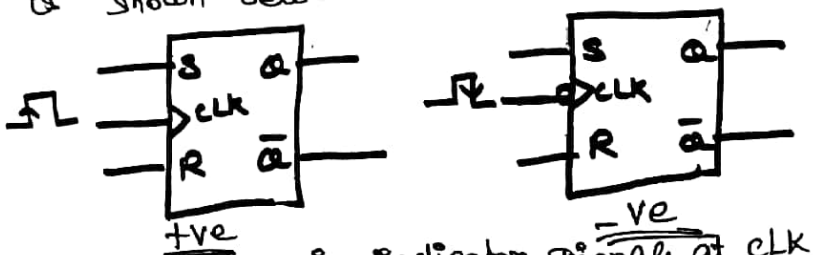
and its output $Q_m = 1$ and $\bar{Q}_m = 0$. It is directly transferred to SR inputs of slave. The slave responds during the falling edge and final output $Q = 1$, $\bar{Q} = 0$. This is set condition.

→ when $S = 0$ $R = 1$ for master, A pulse is applied during rising edge the master responds and its output $Q_m = 0$ and $\bar{Q}_m = 1$. It is directly transferred to SR inputs of slave. The slave responds during falling edge and final output $Q = 0$ and $\bar{Q} = 1$. This reset condition.

→ when $S = 0$ $R = 0$, Q_m and $\bar{Q}_m$ retains the previous values and so final output from the slave also does not change.

The second SR flip-flop always follows the first SR flip-flop and hence the first one is called the master and the second is slave.

→ Block diagram for positive edge triggered SR flip-flop, and negative edge triggered SR flip flop is shown below.

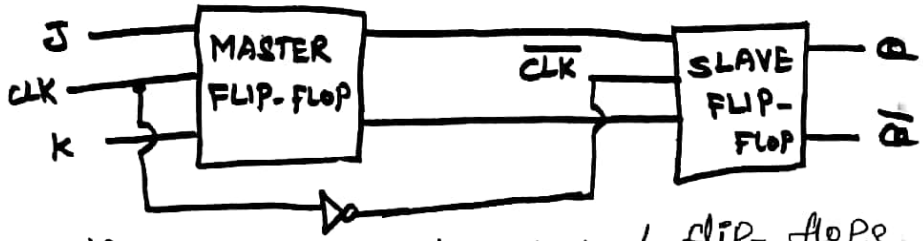


→ Indicates dynamic indicator. Triangle at CLK I/P indicates +ve edge, Triangle with bubble at front indicates -ve.

JK Master-slave flip flop :-

In the operation of JK flip-flop, it is very difficult to satisfy the requirements, which should be fulfilled to avoid 'racing condition'. when both the data I/Ps are high. the practical approach to overcome this problem is to use 'Master-slave JK flip-flop'.

Logic diagram :-



This diagram contains two clocked flip-flops. one flip-flop serves as master and other as slave.

→ whenever the clock is high, the master is active. According to data inputs, the output of master is set or reset. At this stage, slave is inactive and its output remains in previous stage.

→ whenever the clock I/P is low, the slave is active. The final output of master-slave flip flop is the O/P of slave flip-flop.

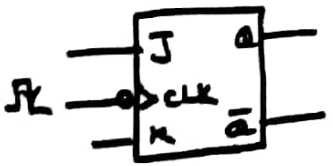
Operation :-

1) The information present at the J and K I/P is transmitted to master flip flop on the edge of a CLK is held until -ve edge of CLK pulse occ after which is allowed to pass through slave.

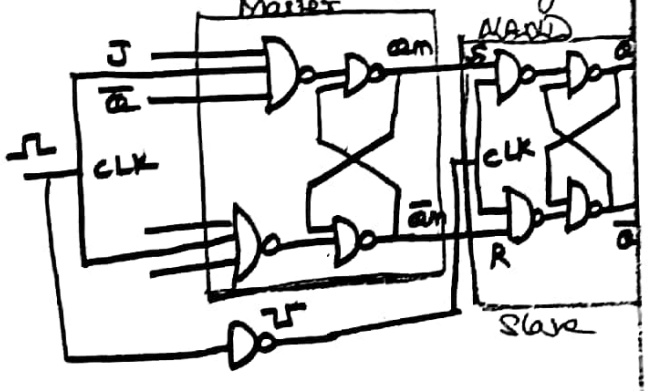
2) when CLK I/P is Low, it keeps the o/p of gates 1 and 2 at high level. this prevents the J and K I/P from affecting master flip-flop. when CLK is Low, the output gate is High, so that Q is equal to Y and $\overline{Q}$ is equal to Y.

3) when +ve edge of a clock Pulse occurs, the master flip-flop is affected and slave is isolated as long as clock pulse is at High. when CLK I/P is returned to Low, the master is isolated from J and K inputs and slave flip-flop goes to same state as the master flip-flop.

Block diagram



Logic Diagram using Master and Slave



Truth table :-

Inputs			outputs		Mode of operation
J	K	CLK	Q	$\overline{Q}$	
0	0	⊓	Q_0	$\overline{Q}_0$	NO change
0	1	⊓	0	1	RESET
1	0	⊓	1	0	SET
1	1	⊓	$\overline{Q}_0$	Q_0	Toggle.

REGISTERS

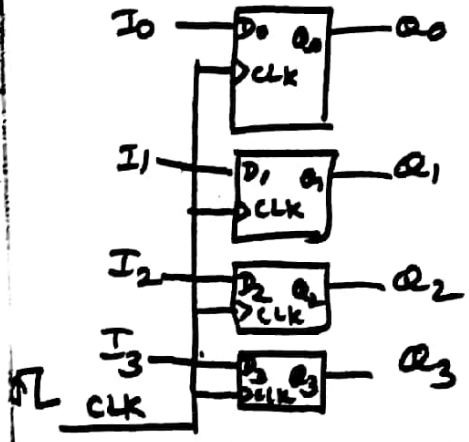
Introduction:

An array of flip-flops required to store the binary information is known as Registers. An n bit register has a group of n-flip flops and is capable of storing any binary information of n bits. The number of flip-flops are required as equal to the number of bits in the binary word. (ie), by taking one flip-flop for each bit.

Need for registers:

(i) Temporary storage
(ii) shifting capability.

A single D flip-flop or D latch can be used to store a single bit. It has one I/P, one o/p and a CLK I/P or an enable I/P. The register consists of four D flip-flops. The four binary I/P are I_0, I_1, I_2, I_3 and o/P Q_0, Q_1, Q_2 and Q_3. The clock pulse is also applied.

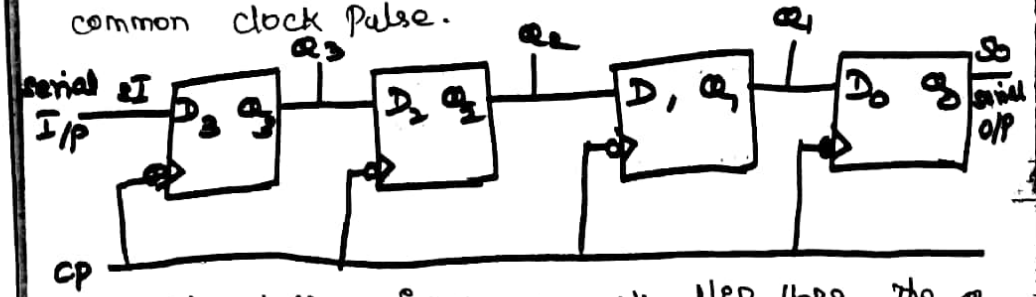


The flip-flops are edge triggered so that data available at that instant. The transfer of data goes on till the enable I/p gate I/p reaches low state. The last data received is retained as previous value.

Shift Registers :-

A register capable of shifting its binary information either to the right or to the left is called a shift register.

The logical configuration of a shift register consists of a chain of flip-flops connected in cascade, with the output of one flip-flop connected to the I/p of the next flip-flop. All flip-flop receives a common clock pulse.



The shift registers uses only flip-flops. The output of a given flip-flop is connected to the D input of the flip-flop at its right. Each clock pulse shifts the content of the registers one bit position to the right.

The serial I/p determines what goes into the left-most flip flop during the shift. The serial o/p is taken from the output of the right most flip-flops.

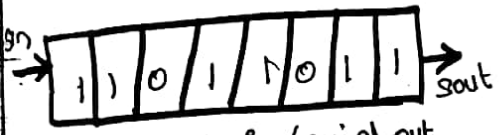
Block diagram of 4-bit shift register :-

Data movements in registers :-

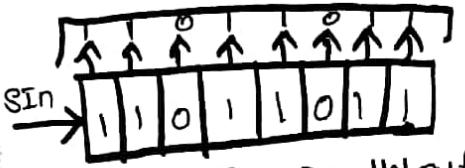
The data can be entered either in serial form (one bit at a time) or parallel form (all the bits simultaneously).

The four possible modes are

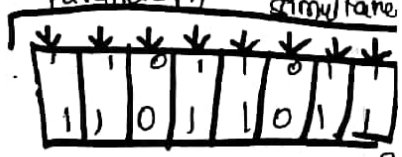
* Serial in - Serial out → one bit at a time on single li
* Serial in - Parallel out → similar I/P as serial in but O/P taken simultane
* Parallel in - Serial out → I/P simultaneous O/P ar on one bi
* Parallel in - Parallel out → Both I/P and O/P ar



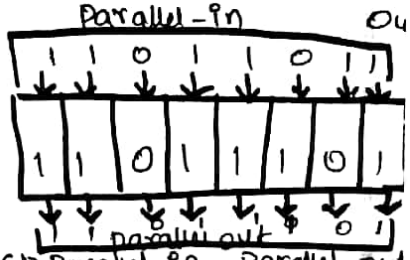
(a) Serial-in / serial-out



(b) Serial in - Parallel out



(c) Parallel in - Serial



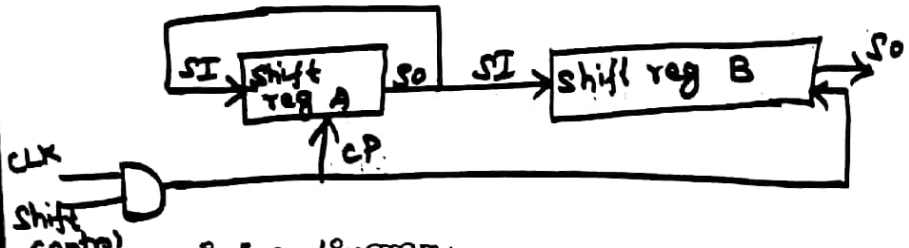
(d) Parallel in - Parallel out

Serial transfer :-

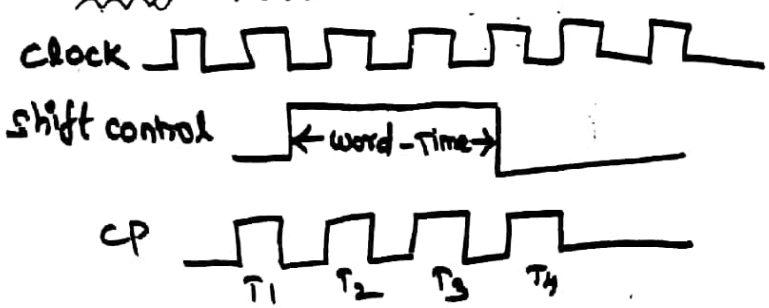
A digital system is said to operate in a serial mode when information is transferred and manipulated one bit at a time. The content of one register is transferred to another by shifting the bits from one register to the other. The information is transferred one bit at a time by shifting the bit out of the source register into the destination register.

The serial transfer of information from register A to register B is done with shift register.

Serial transfer block diagram:



Timing diagram:



The time interval between the clock pulse is called bit time and the time required to shift the entire content of a shift register is called word-time.

The serial output (SO) of a register A goes to the serial Input (SI) of the register B. To prevent the loss of Information stored in the source register by circulated its serial output to its serial inputs. The Initial content of register B is shifted out through its serial output and is lost unless it is transferred to a 3rd shift register.

The shift control input determines when and by how many times the registers are shifted. This is done by the AND gate that allows clock pulse to pass into the CP terminals only when the shift control is 1.

Serial transfer Example:

Timing Pulse	Shift register A	Shift reg B	Serial o/p of
Initial value	1 0 1 1	0 0 1 0	0
After T1	1 1 0 1	1 0 0 1	1
After T2	1 1 1 0	1 1 0 0	0
After T3	0 1 1 1	0 1 1 0	0
After T4	1 0 1 1	1 0 1 1	1

Assume A is 1011 and the register B is 0010. The serial transfer from A to B will occur in 4 steps. After the first Pulse T, A is shifted into the leftmost bit of B and at the same time, this is circulated into the leftmost position of A. The other bits of A and B are shifted once to the right. The previous serial output from B is lost and its value changes from 0 to 1. After the 4th shift, the shift control goes to '0' and both register to B and have the value 1011.

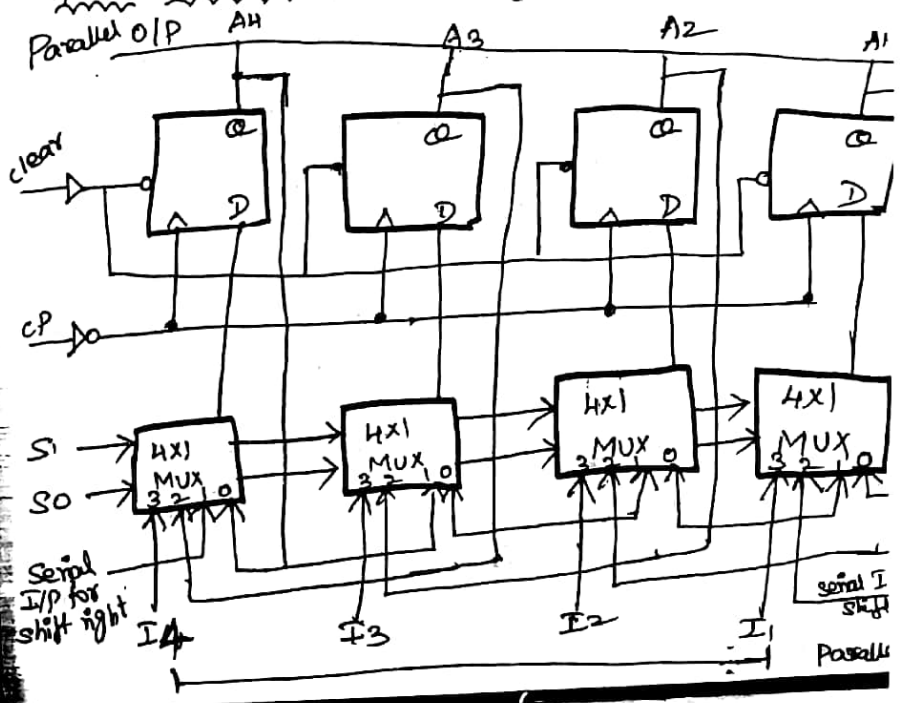
Serial Modes	Parallel Modes
* In serial mode, the register have a single serial I/P and a single serial O/P. one bit transferred at a time.	* In a parallel mode Inform-ation is available from all bits of a register and all bits can be transferred simultaneously Survey on clock Pulse
* Serial operations are slower. It requires less hardware to perform operation.	* Most computer operated in a parallel mode because this is faster mode of operation.
* Timing sequence are generated by the control section of the system.	* control signals are enabled during one clock pulse interval.

Bidirectional shift register with [illegible]

Shift registers can be used for converting serial data to parallel data and vice versa. If we have access to all the flip-flop output of a shift register, then information entered serially by shifting can be taken out in parallel from the outputs of the flip-flop.

If a parallel load capability is added to a shift register, then data entered in Parallel can taken out in serial fashion by shifting the data stored in the register.

4-bit bidirectional shift register with Parallel [illegible]



1. A clear control to clear the register to '0'.
2. A CP Input for clock pulse to synchronize all operations.
3. A shift right control to enable the shift-right operation and the serial input and output lines associated with the shift right.
4. Shift left control to enable the shift-left operation and serial Input and serial output lines associated with the shift-left.
5. A Parallel-load control to enable a parallel transfer and the n input lines associated with the parallel transfer.
6. A Parallel output lines.
7. A control state that leaves the information in the register unchanged even though clock pulses are continuously applied.

A register capable of shifting both right and left is called a bidirectional shift register. One that can shift in only one direction is called a unidirectional shift register. If the register has both shift and Parallel load capabilities. It is called a shift register with Parallel load.

Function table for the register:

Mode control		Register operation
S_1	S_0	
0	0	No change
0	1	shift right
1	0	shift left
1	1	Parallel load.

A bidirectional shift register with parallel load is a general-purpose register capable of performing 3 operations.

* shift left
* shift right
* Parallel load.

Types of shift register:-

→ shift right shift register
→ shift Left shift register

Shift right shift register:-

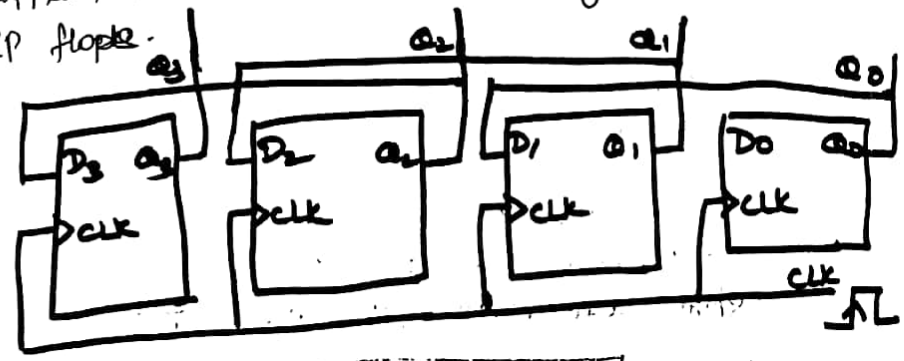
In application of a clock pulse, during the rising edge, the Input of each flip-flop(FF) is transferred to the output, providing a shift right operation.

Ex: Sin data as (1011) that is given one by one from LSB

CLK	Sin	Q_3	Q_2	Q_1	Q_0
—	1	0	0	0	0
↑	1	1	0	0	0
↑	0	1	1	0	0
↑	1	0	1	1	0
↑	—	1	0	1	1

Shift Left shift Registers:-

It shifts each bit to the Left once when a clk Pulse is applied. It can be constructed using 4 D (or) JK flip flops.



Ex:

CLK	Q_3	Q_2	Q_1	Q_0	Sin
—	0	0	0	0	
↓	0	0	0	0	1
↓	0	0	0	1	0
↓	0	0	1	0	1
↓	0	1	0	1	1
↓	1	0	1	1	

UNIT - V

Types of ROM and RAM:-

Introduction:-

Memory is a unit of computer, which stores instruction and data and supplies the same to the computer as and when required. It also stores the results that are computed by the computer.

The memory is divided as Internal and External m

External memory → used for long-term storage data
Ex: Floppy disk, hard disk.

Internal memory → It is smaller segment of memory temporary storage of program and data. Interna memory uses electronic semi-conductor device calle ROM and RAM.

ROM (Read - only Memory)

* It is a read only memory. program or data stored in the ROM can only be read. They c not be altered.
* It is non-volatile which means the contents ar not changed when the power is switched off the contents of ROM cannot be erased. The user cannot change the when he wants. To overcome this problem EPROM was introduce

EPROM: It is an Erasable programmable Read only memory. The contents can be erased by shining light-intensity ultra-violet (UV) light through a quartz window. The entire contents are erased when exposed to UV light.

Some popular EPROMs are

2716 – 2K x 8
2732 – 4K x 8
2764 – 8K x 8

EPROM
- EEPROM
- UVEPROM

MASK ROM: It is the memory in which the data is permenantly stored in the memory during the manufacturing process.

PROM: It is programmable ROM in which the data is electrically stored by the user with the aid of specialised equipment.

EEPROM: certain ROM devices can be both erased and programmed with electrical signals instead of ultraviolet light. These devices are known as electrically erasable PROMS (EEPROM). It is also called as electrically alterable (EAROM). It can be both electrically written into and electrically erased, the EEPROM can be programmed and erased in the circuit for reprogramming.

UVPROM: The contents can be erased using ultraviolet light.

RAM (Random-Access memory)

→ Data can be readily written into and read from a RAM at any selected address in any sequence. It has the facility to read the contents of any memory location as well as write any fresh data in any memory location.

→ It is volatile which means that the contents are changed once the power supply is switched off.

Two types of RAM:

1. static RAM (SRAM) 2. Dynamic RAM (DRAM)

Static RAM (SRAM): (volatile memory)

*In SRAM binary information is stored in flip-flops, one-bit for each flip-flop. A large number of flip-flops are arranged in the form of a matrix. A '0' or a '1' can be stored temporarily in a flip-flop. It is composed of storage elements such as flip-flop that are capable of retaining the information indefinetly as long as the power is ON, so that is known as static RAM.

Dynamic RAM (DRAM)

*A memory having cells that tend to lose the storage information over a period of time and therefore must be 'refreshed'. This memory is known as dynamic memory.

Static Memory	Dynamic Memory
* Stored data is retained as long as power is ON.	* Stored data gets lost and refreshing is required.
* the stored data do not change with time.	* Store data changes with time.
* Consumes more Power.	* consumes less power.
* Expensive	* Economical
* These memories have less Packing density.	* High packing density
* These memories are not easy to construct.	* Simpler in construction
* No refreshing required and easy operation.	* Refreshing required with additional memory circuit and complicates memory.
* No maintenance.	* Maintenance required.

Book:
NOTES FOR FUNDAMENTALS OF DIGITAL COMPUTER ELECTRONICS

NO. OF PAGES: 75, PAGE BACK DOUBLE COLUMNED + 6 PAGE SINGLE COLUMNED (TO BE PRINTED)

PAPER PRINTED UNDER: ROHITH KUMAR.

NOTE: BY MRS SHIVARANJANI

MAINLY FOR: 1 BCA 'A' STUDENTS

DEVELOPED BY: CAM SCANNER

~~HAPPY LEARNING~~

- HAPPY DIGITAL LEARNING -

NOTE: ANY MISTYPEN OR INPROPER OF LINEAR NOTES IN ANY PAGE IS NOT OUR RESPONCIBLE. ASK THE PRINTER INCHARGE FOR ANY COMPLAINTS.

UNIT - V

Counters:

A register is used solely for storing and shifting data which is in the form of 1s and / or 0s, entered from an external source. It has no specific sequence of states except in certain very specialized applications. In digital logic and computing, *a counter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal.* Count represents the number of clock pulses arrived. A specified sequence of states appears as the counter output. Specified set of states are different for different types of counters. In a digital circuit, counters are used to do 3 main functions.

1. Timing: High frequency pulse train can be used to produce low frequency pulse train (10 MHz oscillator to produce 1 MHz pulse train)
2. Sequencing: Applying power to large machine components in particular sequence
3. Counting: Total number of vehicles passing a particular road.

Asynchronous Counters

In Asynchronous counters only the first flip-flop is clocked by an external clock. All subsequent flip-flops are clocked by the output of the preceding flip-flop. Asynchronous counters are slower than synchronous counters because of the delay in the transmission of the pulses from flip-flop to flip-flop. Asynchronous counters are also called ripple counters because of the way the clock pulses, or ripples, its way through the flip-flops.

Synchronous Counters:

All flip-flops are clocked simultaneously by an external clock. Synchronous counters are faster than asynchronous counters because of the simultaneous clocking. Synchronous counters are an example of state machine design because they have a set of states and a set of transition rules for moving between those states after each clocked event.

Comparison of synchronous and asynchronous counters:

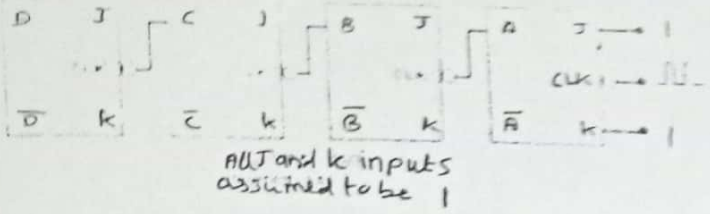
Asynchronous Counter	Synchronous Counter
Flip-flops are connected in such a way that output of first flip-flop drives the clock for the next flip-flop.	In this type there is no connection between output of first flip-flop and clock input of the next flip-flop.
All the flip-flops are not clocked simultaneously.	All the flip-flops are clocked simultaneously.
Logic circuit is very simple even for more number of states.	Design involves complex logic circuit as number of states increases.
Main drawback of these counters is their low speed as the clock is propagated through number of flip-flops before it reaches last flip-flop.	As clock is simultaneously given to all flip-flops there is no problem of propagation delay. Hence they are preferred when number of flip-flops increases in the given design.

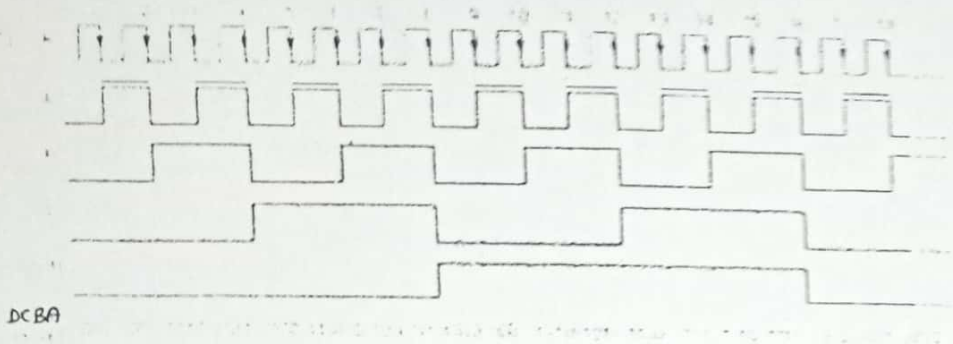
Ripple Counter:

A ripple counter is an asynchronous counter where only the first flip-flop is clocked by an external clock. All subsequent flip-flops are clocked by the output of the preceding flip-flop. There are many ways to implement the ripple counter depending on the characteristics of the flip flops used and the requirements of the count sequence.

- Clock Trigger: Positive edged or Negative edged
- JK or D flip-flops
- Count Direction: Up, Down, or Up/Down

Circuit Diagram:





Clock is applied only to FF A. J and K are high in all FFs to toggle on every clock pulse. Output of FF A is CLK of FF B and so forth. FF outputs D, C, B, and A are a 4 bit binary number with D as the MSB. After the negative transition of the 15th clock pulse the counter recycles to 0000. This is an asynchronous counter because state is not changed in exact synchronism with the clock.

Frequency division: The output frequency of each FF = the clock frequency of input / 2.

Truth Table:

Clock	Counter Output				States	Dec. Output
	D	C	B	A		
Initial	0	0	0	0	-	0
1	0	0	0	1	1	1
2	0	0	1	0	2	2
3	0	0	1	1	3	3
4	0	1	0	0	4	4
5	0	1	0	1	5	5
6	0	1	1	0	6	6
7	0	1	1	1	7	7
8	1	0	0	0	8	8
9	1	0	0	1	9	9
10	1	0	1	0	10	10
11	1	0	1	1	11	11
12	1	1	0	0	12	12
13	1	1	0	1	13	13
14	1	1	1	0	14	14
15	1	1	1	1	15	15

UP/DOWN sequence for a 3-bit binary counter

Clock Pulse	UP	Q_2	Q_1	Q_0	DOWN
0		0	0	0	
1		0	0	1	
2		0	1	0	
3		0	1	1	
4		1	0	0	
5		1	0	1	
6		1	1	0	
7		1	1	1	

An examination of Q0 for both the up and down sequences shows that FFO toggles on each clock pulse. So the J0 and K0 inputs of FFO are

$$J0 = K0 = 1$$

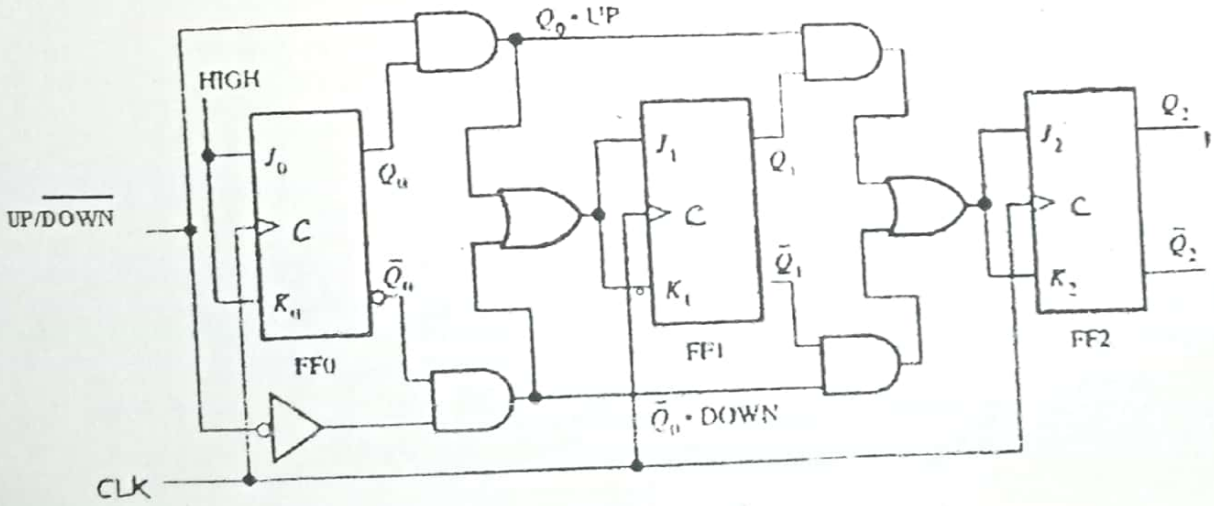
For the up sequence, Q1 changes state on the next clock pulse when Q0 = 1. For the down sequence, Q1 changes on the next clock pulse when Q0 = 0. Thus, the J1 and Kl inputs or FF1 must equal 1 under the conditions expressed by the following equation:

$$J_1 = K_1 = (Q_0 .UP) + (\overline{Q}_0 .DOWN)$$

For the up sequence, Q2 changes state on the next clock pulse when Q0 = Q1 = 1. For the down sequence, Q2 changes on the next clock pulse when Q0 = Q0 = 0. Thus, the J2 and K2 inputs of FF2 must equal 1 under the conditions expressed by the following equation:

$$J_2 = K_2 = (Q_0 .Q_1 .Up) + (\overline{Q}_0 .\overline{Q}_1 .DOWN)$$

Circuit Diagram:



Mod Counter:

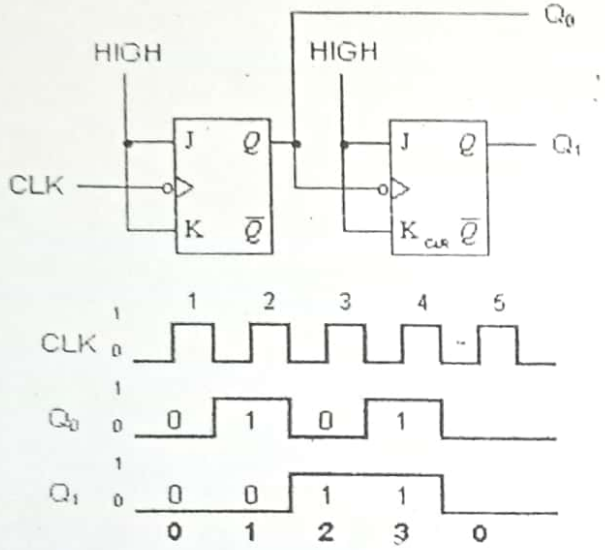
A mod n counter can also be termed as divide by n counter that divides the clock frequency by n. The divided frequency output is obtained from the MSB of the counter. The counter may be synchronous or asynchronous. In synchronous counter design, the clock is given simultaneously to all Flip Flops whereas in asynchronous design, the clock is given as an input to the LSB flip flop.

The 2-bit ripple counter is called as MOD-4 counter and 3-bit ripple counter is called as MOD-8 counter. So in general, an n-bit ripple counter is called as modulo-N counter. Where, MOD number = 2^n

Type of modulus:

- 2-bit up or down (MOD-4)
- 3-bit up or down (MOD-8)
- 4-bit up or down (MOD-16)
- ---

- Modulo 2 counter:



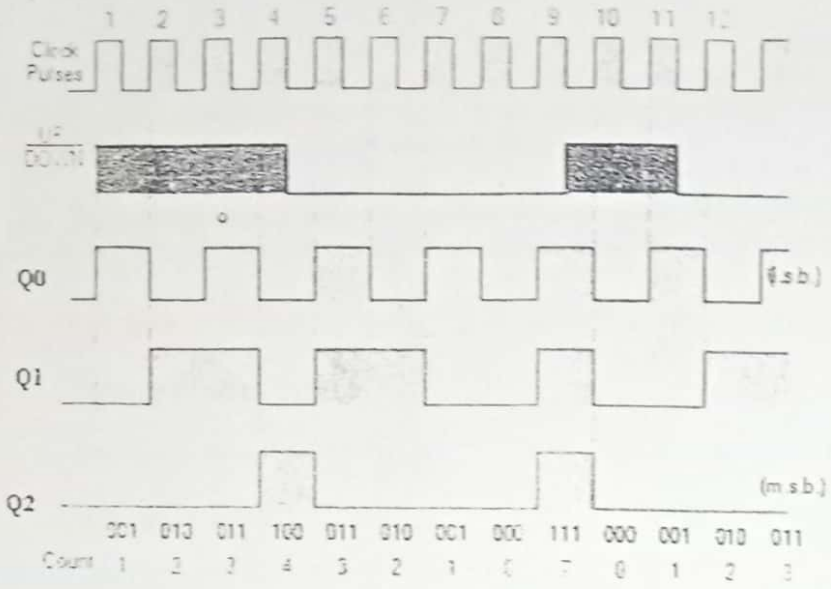
Up-Down Counters:

A synchronous binary up counter counts from 0 to 2^N-1, where N is the number of bits/flip-flops in the counter. Each flip-flop is used to represent one bit. The flip-flop in the lowest-order position is complemented/toggled with every clock pulse and a flip-flop in any other position is complemented on the next clock pulse provided all the bits in the lower-order positions are equal to 1. A synchronous binary down counter counts down from 2^N-1 to 0, where N is the number of bits/flip-flops in the counter.

An up/down counter is one that is capable of progressing in either direction through a certain sequence. An up/down counter is sometimes called bi-directional counter, can have any specified sequence of states. A 3-bit counter that up-ward through its sequence (0,1,2,3,4,5,6,7) and then can be reversed so that it goes through the sequence (7,6,5,4,3,2,1) in reverse direction is an illustration of up/down sequential operation.

Below table shows the complete up/down sequence for a 3-bit binary counter. The arrows indicate the state-to-state movement of the counter for both its UP and its DOWN modes of operation.

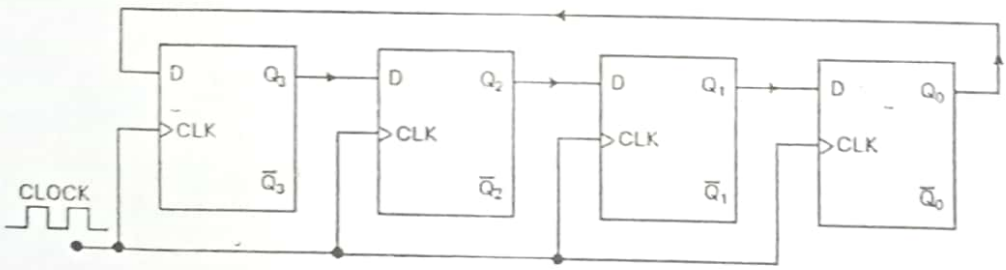
Timing Diagram:



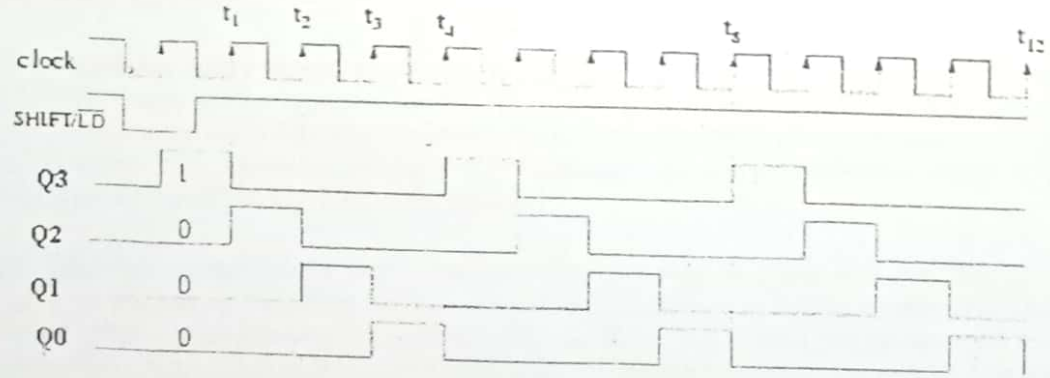
Ring Counter:

Ring counters are implemented using shift registers. It is essentially a circulating shift register connected so that the last flip-flop shifts its value into the first flip-flop. There is usually only a single 1 circulating in the register, as long as clock pulses are applied. '*Johnson Counter* is a special ring counter where the inverted output of the last flip flop is given as input to the first flip flop'.

Circuit Diagram:



Timing Diagram:



Load 1000 into 4-stage ring counter and shift