Section C

1. Digital Electronics — 20 Q
2. Computer Arch. — 15 Q
3. Microprocessor. — 15 Q

= 40 Q / Marks.

=

# Digital Electronics

*Trainer: Sohail Inamdar*

# 1's and 2's Complement

- **1's Complement**
  - The 1's complement of a number is found by changing all 1's to 0's and all 0's to 1's. This is called as1's complement.

1 → 0
0 → 1

101011100
⇩
010100011

- **2's Complement**
  - The 2's complement of binary number is obtained by adding 1 to the Least Significant Bit (LSB) of 1's complement of the number.
  - 2's complement = 1's complement + 1

101011100
010100100

# WEIGHTED AND NON-WEIGHTED CODES

- **Weighted codes**
  - Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the number represents a specific weight.
  - Examples of weighted code is BCD. In these codes each decimal digit is represented by a group of four bits.

- **Non-Weighted codes**
  - In this type of binary codes, the positional weights are not assigned. The examples of non-weighted codes are Excess-3 code and Gray code.

# BCD Number

10 → 1010 , 6 = 0110

- BCD – Binary Coded Decimal.
- In this code each decimal digit is represented by a 4-bit binary number.
- BCD is a weighted code its weight are 8421.BCD code are used only till 9(0000 to 1001).
- BCD to Decimal.

  (0101 1000) BCD = ( ? )$_{10}$ (5 8)
  (1001 0110) BCD = ( ? )$_{10}$ $(9\ 6)_{10}$
  (0110 1000) BCD = ( ? )$_{10}$ $(6\ 8)_{10}$
  (1000 0010) BCD = ( ? )$_{10}$ $(8\ 2)_{10}$

  $\boxed{0101}\boxed{1000}$ =

- Decimal to BCD

  ( 6 ) = ( ? )BCD → (0110) BCD
  ( 254 )$_{10}$ = ( ? )BCD → (0010 0101 0100) BCD (2 5 4)
  ( 168 )$_{10}$ = ( ? )BCD (0001 0110 1000) BCD
  ( 85 ) = ( ? )BCD

- Binary to BCD and BCD to Binary Conversion.
  - Step 1- Convert the binary/BCD number to decimal.
  - Step 2- Convert decimal number to binary/BCD.

BCD
⇓ ← step1
Decimal
⇓ ← step2
Binary

# Excess-3 code

- The Excess-3 code is also called as XS-3 code.
- It is non-weighted code used to express decimal numbers.
- The Excess-3 code words are derived from the 8421 BCD code words by adding (0011) (3) to each code word in 8421.
- **Decimal to Excess-3**
  - Step 1- Convert decimal to BCD.
  - Step 2- Add 3 (0011) to this BCD number.

$(5)_{10} \Rightarrow 0101 \Rightarrow 0101 + 0011.$

Decimal BCD $= \boxed{1000}$ Excess-3

- **Excess-3 to Decimal**
  - Step 1- Subtract $(0011)_2$ from each 4 bit of excess-3 digit to obtain the corresponding BCD code.
  - Step 2- Convert BCD to Decimal.

1000 → Excess3

step 1 → − 0011

0101 → BCD

5 → Decimal

- **Binary to Excess-3**
  - Step 1- Convert Binary to decimal.
  - Step 2- Convert decimal to BCD.
  - Step 3 - Add 3 (0011) to this BCD number.

# Gray Code

4 3 2 1 0
2 2 2 2 2
1 0 0 0 1

- It is the non-weighted code and it is not arithmetic codes.
- There are no specific weights assigned to the bit position.
- It has a very special feature that, only one bit will change each time the decimal number is incremented(only one bit changes at a time).
- Gray code is popularly used in the shaft position encoders.

| Decimal Number | Gray Code |
|---|---|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0011 |
| 3 | 0010 |
| 4 | 0110 |
| 5 | 0111 |
| 6 | 0101 |

① It changes by 1 bit at a time
② On each itteration pattern will not be repeated.

0000 → 0
0001 → 1
0011 → 2
0010 → 3
0110 → 4
0111 → 5
0101 → 6
0100 → 7
1100 → 8

# Boolean algebra

- Algebra that deals with binary number system
- George Boole developed it for simplification and manipulation of logic
- Boolean algebra uses
  + Binary digits - 0 and 1
  + Logical addition '+' also known as 'OR' which follows law of binary addition
  + Logical Multiplication '.' also known as 'AND' which follows law of binary multiplication
  + Complementation '-' also known as 'NOT' which follows law of binary complement
- Boolean algebra is used to simplify Boolean expressions which represent combinational logic circuits.
- **Operator precedence (scanned from left to right)**
  - ()
  - NOT
  - AND
  - OR

# Boolean Function

- Boolean function is expression formed with
  - + Binary variables
  - + Operators (AND, OR, NOT)
  - + Parentheses and equal to sign
- Value of Boolean function can be either 0 or 1
- Boolean function can be represented as an algebraic expression or a truth table

  e.g. W = f(x,y,z)

  Where, W is a function

  x,y,z are variables or literals

- **Minimization of Boolean functions**

  Minimization deals with

  + Reduction in number of variables

  + Reduction in number of terms

- There are 2 methods to minimize any Boolean function
  - + Using Boolean laws
  - + Using K-map

# Laws of Boolean Algebra

- Idempotent Law
  - A * A = A  $1 * 1 = 1$
  - A + A = A  $1 + 1 = 1$
- Associative Law
  - (A * B) * C = A * (B * C)
  - (A + B) + C = A + (B + C)

$(1+0)+1 = 1+1 = 1$ | $1+(0+1) = 1+1 = 1$

- Commutative Law
  - A * B = B * A
  - A + B = B + A
- Distributive Law
  - A * (B + C) = A * B + A * C
  - A + (B * C) = (A + B) * (A + C)
- Identity Law
  - A * 0 = 0  A * 1 = A
  - A + 1 = 1  A + 0 = A
- Complement Law
  - A * ~A = 0  $1 * (\sim 1) = 1 * 0 = 0$
  - A + ~A = 1  $1 + (\sim 1) = 1 + 0 = 1$
- Involution Law
  - ~(~A) = A  $\sim(\sim 1) = \sim(0) = 1$
- DeMorgan's Law
  - ~(A * B) = ~A + ~B
  - ~(A + B) = ~A * ~B

  \+  *

# Standard Form

$A = 1 \to A$

$A = 0 = \bar{A}$

- **Minterm**
  - + Minterm is a product or AND term
  - + contains n variables with 2^n possible combinations
  - + variables either in normal or in complemented form
  - e.g. ABC, A'BC, AB'C'

$A + B + C$

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

- **Maxterm**
  - + Maxterm is a sum or OR term
  - + contains n variables with 2^n possible combinations
  - + variables either in normal or in complemented form
  - e.g. A+B+C, A+B'+C

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

- **Sum of product (SOP) expression**

  SOP expression is minterm or minterms logically added(ORed) together.

  e.g. A'B + AB

$A \times B + B \times C$

- **Product of sum (POS) expression**

  POS expression is maxterm or maxterms logically multiplied(ANDed) together.

  e.g, (A+B).(A'+B)

$A+B * A+C$

# Standard Form

$A = 1$
$\bar{A} = 0$

sum of products.

- Solve the X(A,B,C) = A + BC convert in Standard SOP form

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

$BC = BC(A+\bar{A})$
$= BCA + BC\bar{A}$
$= ABC + \bar{A}BC$

$A+BC = ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + ABC + \bar{A}BC$
$= ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + \bar{A}BC$
$\quad 111 \quad 110 \quad 101 \quad 100 \quad 011$
$= 7 \quad 6 \quad 5 \quad 4 \quad 3$

$\Sigma(7,6,5,4,3)$

$x(A,B,C) = \Sigma(3,4,5,6,7)$

# K-Map

- Karnaugh introduced a simplification of Boolean functions in an easy way.
- This method is known as Karnaugh map method or K-map method.
- It is a pictorial representation of graphical method, which consists of 2# cells for ‘n’ variables. The adjacent cells are differed only in single bit position.
- K-map uses Gray code.

- 2- bits

| A\B | 0 | 1 |
|---|---|---|
| 0 | 0th | 1st |
| 1 | 2nd | 3rd |

- 3-bits

| A\BC | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0th | 1st | 3rd | 2nd |
| 1 | 4th | 5th | 7th | 6th |

# K-Map

- Karnaugh Map Simplification Rules-
- To minimize the given Boolean function, we draw a K-Map according to the number of variables it contains.
- We fill the K-Map with 0's and 1's according to its function.Then, we minimize the function in accordance with the following rules.
- **Rule-1**:
  - We can either group 0's with 0's or 1's with 1's but we can not group 0's and 1's together.
  - X representing don't care can be grouped with 0's as well as 1's.
- **Rule-02:**
  - Groups may overlap each other.
- **Rule-03**:
  - We can only create a group whose number of cells can be represented in the power of 2.
  - In other words, a group can only contain 2n i.e. 1, 2, 4, 8, 16 and so on number of cells.

# K-Map

- **Rule-4**:
  - Groups can be only either horizontal or vertical.
  - We can not create groups of diagonal or any other shape.
- **Rule-5**:
  - Each group should be as large as possible.
- **Rule-6**:
  - Opposite grouping and corner grouping are allowed.
- **Rule-7**:
  - There should be as few groups as possible.

# K-Map

- Simplify the Boolean function F(A,B,C) =Σ(1,5,6,7)
- Simplify the Boolean function F(A,B,C) =Σ(0,1,3,4,5)
- Simplify the Boolean function F(A,B,C,D) =Σ(0,2,4,6,8,9,10)

H.W.

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

$2^n = 2^3 = 8$

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

$\Rightarrow$

| A \ BC | 01 |
|---|---|
| 0 | 1 |
| 1 | 1 |

$\bar{B}C$

| A \ BC | 11 | 10 |
|---|---|---|
| 1 | 1 | 1 |

$A.B$

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

# Logic Gates

**What are logic gates :-**

- It is physical device which performs logical operation on one or more logical i/p(s) and produces a single logical o/p.

- Logical operations :- inversion, logical multiplication, logical sum etc.

**Categories:-**

| | | |
|---|---|---|
| I. | Basic Gates : | AND, OR, NOT. |
| II. | Universal Gates : | NAND, NOR |
| III. | Arithmetic Gates : | X-OR, X-NOR |

# Logic Gates

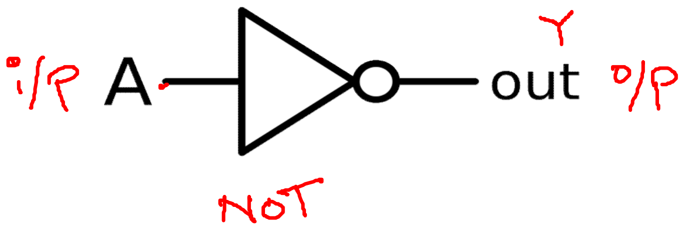
## Truth Table

- A truth table show how a logic circuit responds to various combinations of i/p, using logic 1 for true and 0 for false.
- **Formula for truth table :-** $2^n = m$

  $2^1 = 2$

  $2^2 = 4$
- Where,
  - n → number of inputs
  - M → combination of inputs

➢ NOT Gate :-

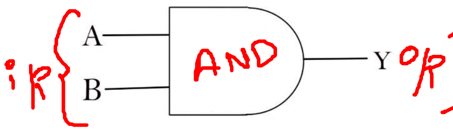



| A | Y |
|---|---|
| 0 | 1 |
| 1 | 0 |

# Logic Gates

## AND Gate:

multiplication

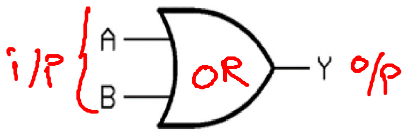



$2^2 = 4$

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

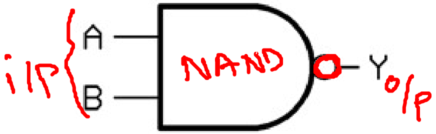
## OR Gate:

OR = sum




| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

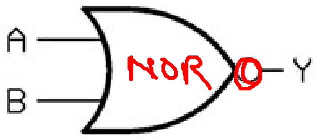
## NAND Gate:




NOT + AND

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

## NOR Gate:

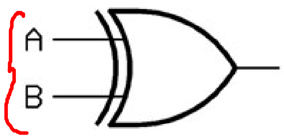



| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

# Logic Gates

- XOR Gate :-

i/p




o/p

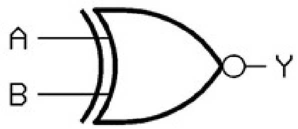
$A \cdot B = Y$

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

$= A \oplus B$

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

XNOR Gate :-




$Y = \overline{A \oplus B}$

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

- Ques.

XNOR Gate, with 5 i/p.

i.e. 101010 → 0 O/p ?

# Universal Gate

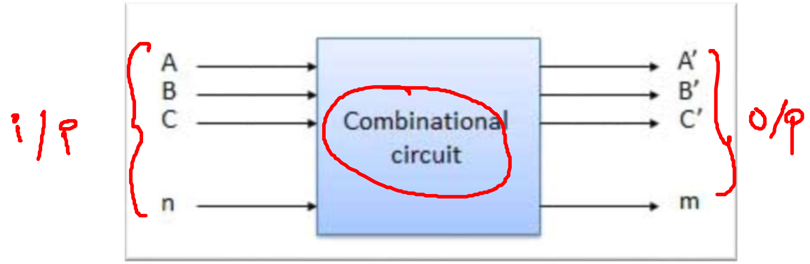
- A Universal gate is a logic gate which can implement any Boolean function without the need to use any other type of logic gate.
- NAND gate and NOR gate are universal gate.
- Any logic circuit can be built using NAND gate or NOR gate.

# Combinational Circuits & Sequential Circuit

Digital Electronics

## Combinational Circuits

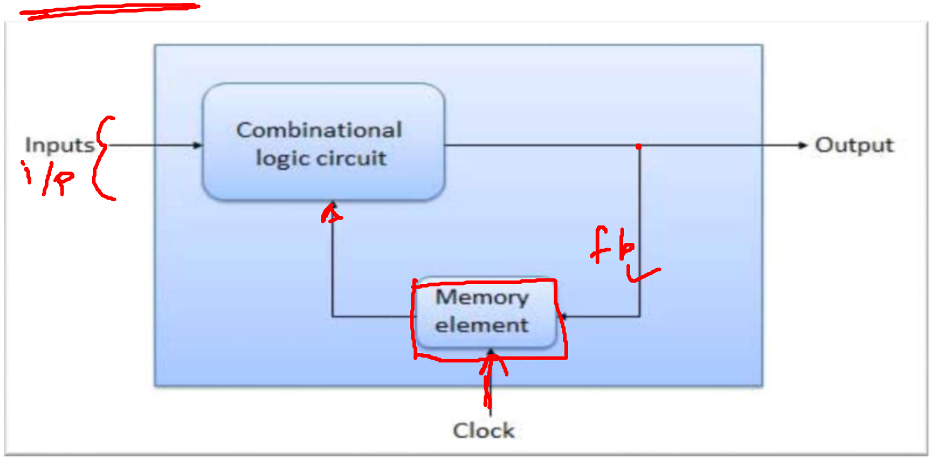
- O/p is only depending on the present I/p.




- No feedback. ✗
- No memory. ✗

## Sequential Circuit

- O/p is depending on the present I/p as well as past outputs.



# Combinational Circuits & Sequential Circuit

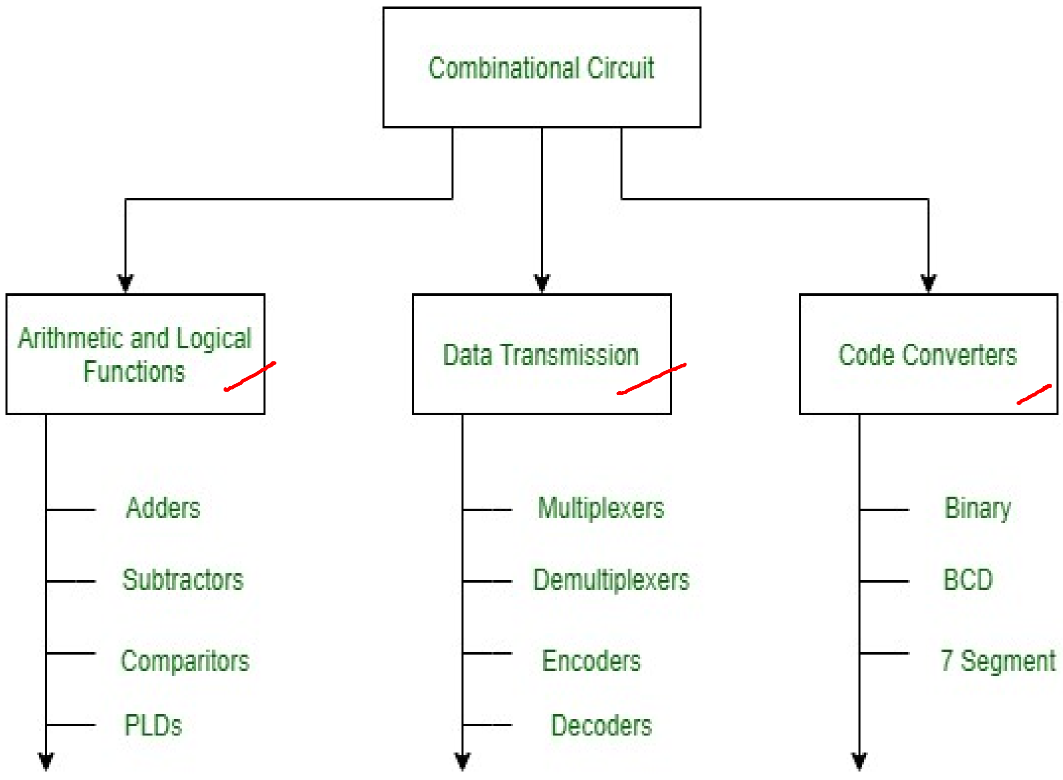
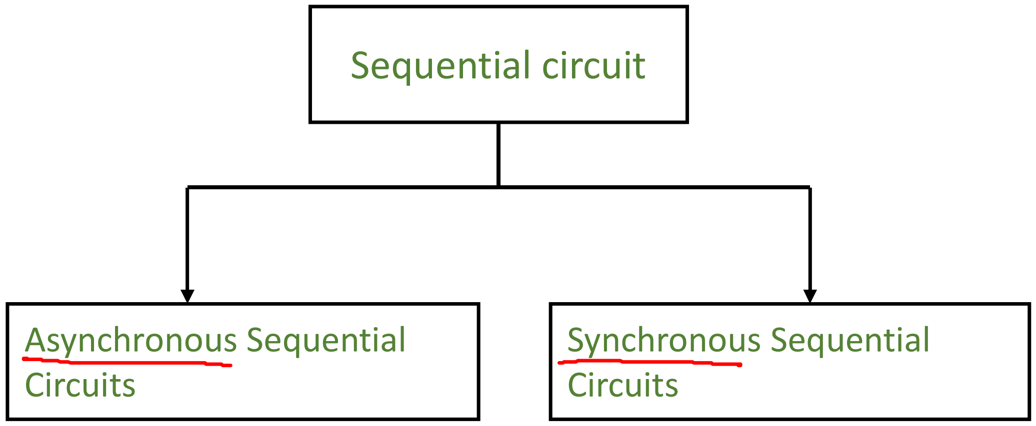



Figure - Classifications of Combinations Circuits

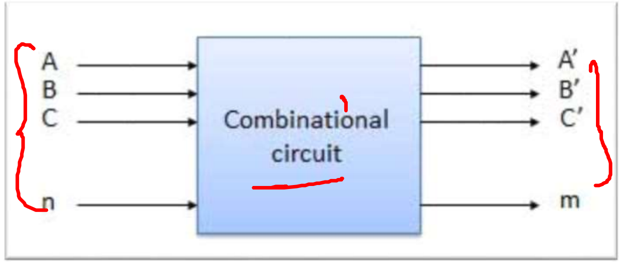


# Combinational Circuits

- **Combinational Logic Circuits** are made up from basic logic AND, OR, NAND, NOR, or NOT gates that are "combined" or connected together to produce more complicated switching circuits.

- Combinational circuits consist
  - Input variables
  - Logic Gates
  - Output variables

i/p




o/p

$Y = F(X)$

Where,

$X = \{x_0, x_1, x_2, \ldots\ldots, x_{n-1}\}$

$Y = \{y_0, y_1, y_2, \ldots\ldots, y_{n-1}\}$

- Design Procedure
  1. Define the problem statement
  2. Determine number of input and output variables
  3. Assign letter symbols
  4. Construct a truth table
  5. Obtain simplified Boolean Expression
  6. Draw a logic diagram

# Designing of Combinational Circuit

Step 1: Problem statement

Design a circuit with 3 inputs which produce a logic 1 output when more than one inputs are logic 1.

Step 2: Input – 3 Output – 1

Step 3: Input variables – A, B, C Output variable – Y

Step 4: Truth Table

$2^3 = 8$

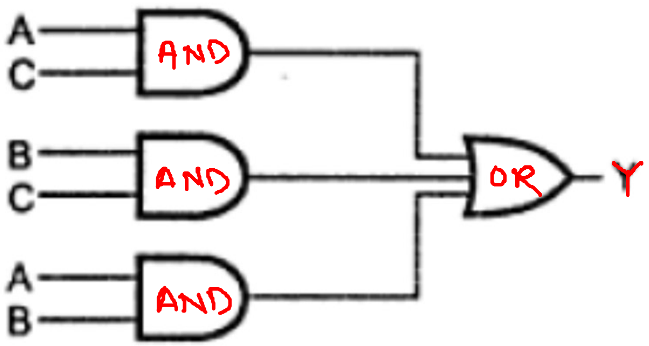
| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Step 5 : Simplified Boolean Expression

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

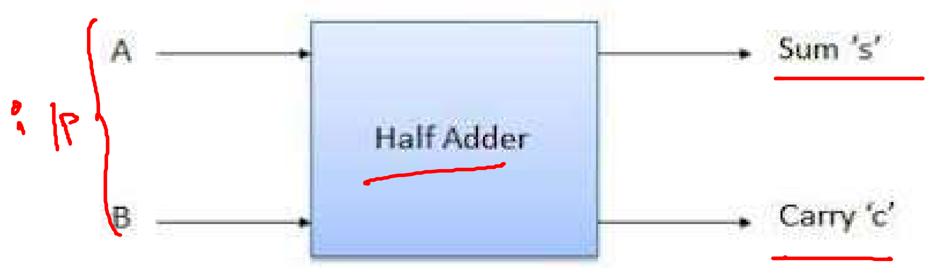
Y = AC + BC + AB

Step 6: Logic Diagram

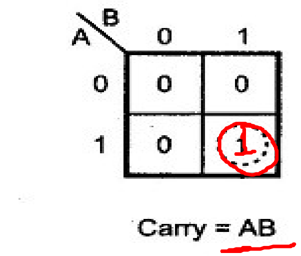


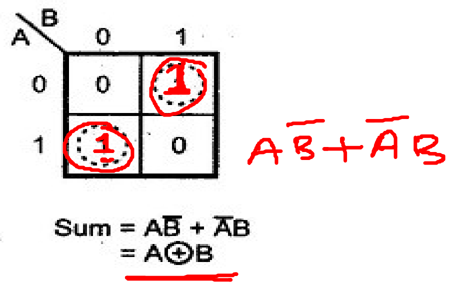
# Half Adder

- Used to add two single bit numbers




## Boolean Expressions



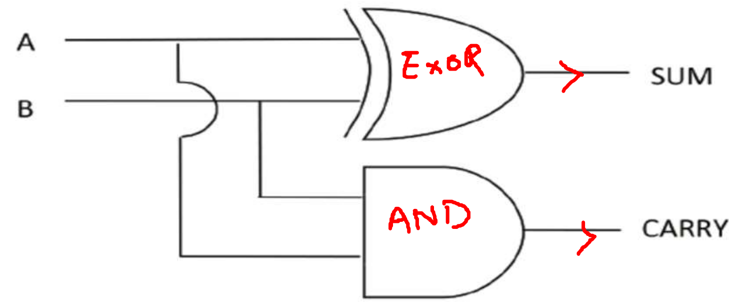




$C = A.B$

$S = A \oplus B$

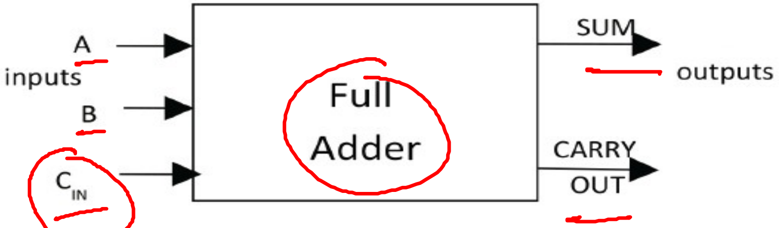
| Inputs | | Outputs | |
|---|---|---|---|
| A | B | Sum | Carry |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

## Logic Diagram



# Full Adder

- Used to add 1 bit numbers with carry

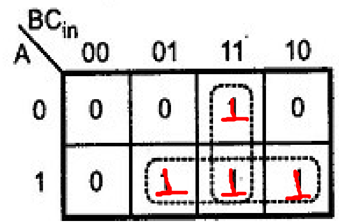



Truth Table

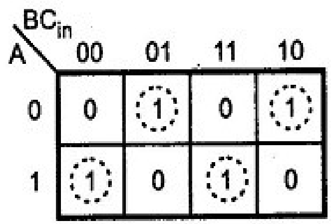
| A | B | $C_{an}$ | S | $C_{oot}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Boolean Expressions

For Carry ($C_{out}$)




For Sum

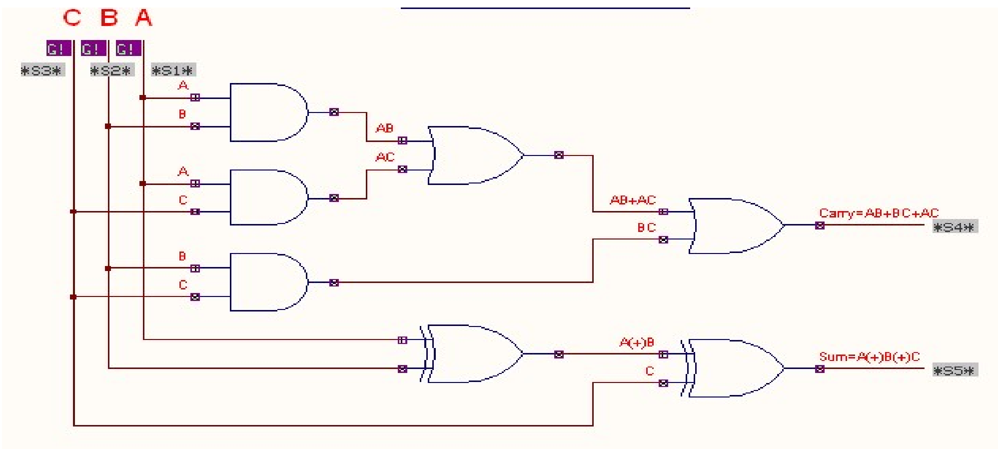



$C_{out} = AB + A\,C_{in} + B\,C_{in}$

$Sum = \bar{A}\,\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\,\bar{C}_{in} + ABC_{in}$

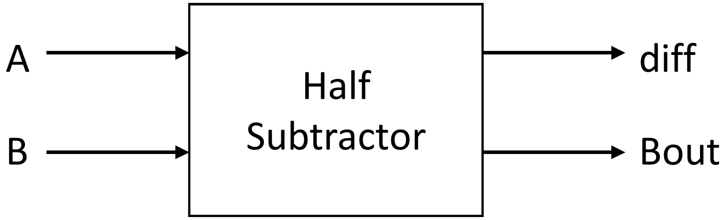
$S = A \oplus B \oplus Ci$

Logic Diagram



# Half Subtractor

- Used to subtract two single bit numbers




Boolean Expressions

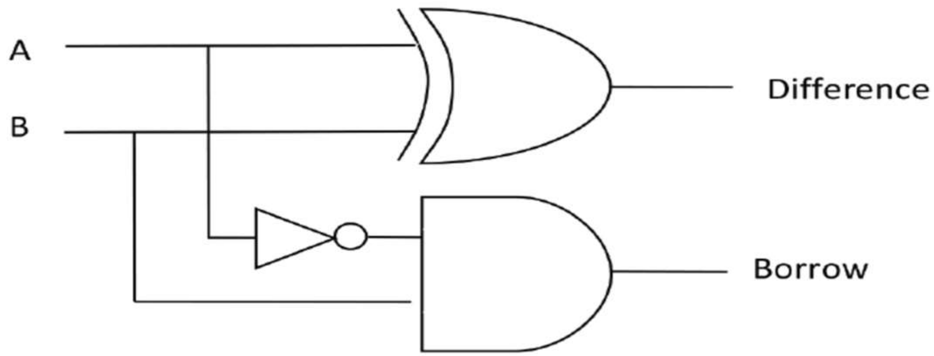

Difference $= A\overline{B} + \overline{A}B$
$= A \oplus B$

Borrow $= \overline{A}B$

Truth Table

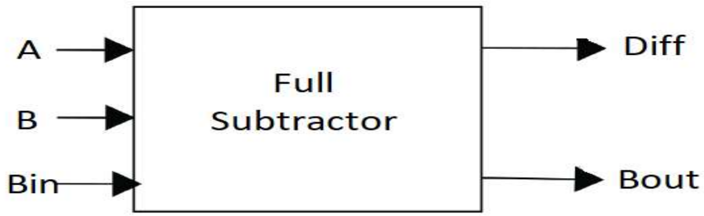
| A | B | Difference | Borrow |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

Logic Diagram



# Full Subtractor

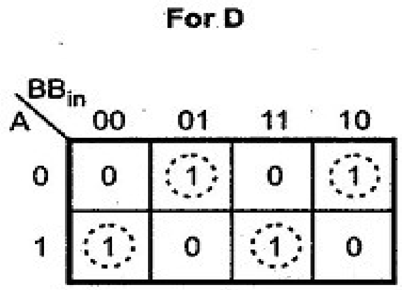
- Used to subtract 1 bit numbers with borrow

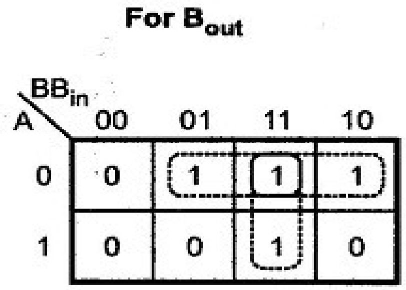



## Truth Table

| A | B | Bin | D | Bout |
|---|---|---|---|---|
| 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 |

## Boolean Expressions

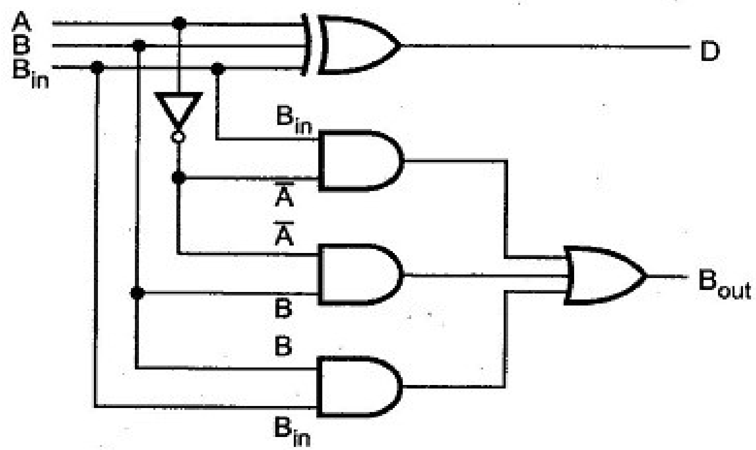



$D = \overline{A}\,\overline{B}B_{in} + \overline{A}B\overline{B}_{in} + A\overline{B}\,\overline{B}_{in} + ABB_{in}$

$B_{out} = \overline{A}B_{in} + \overline{A}B + BB_{in}$

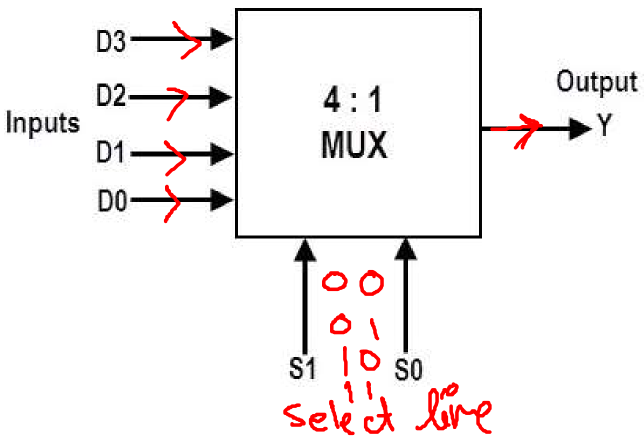
## Logic Diagram



# Multiplexers

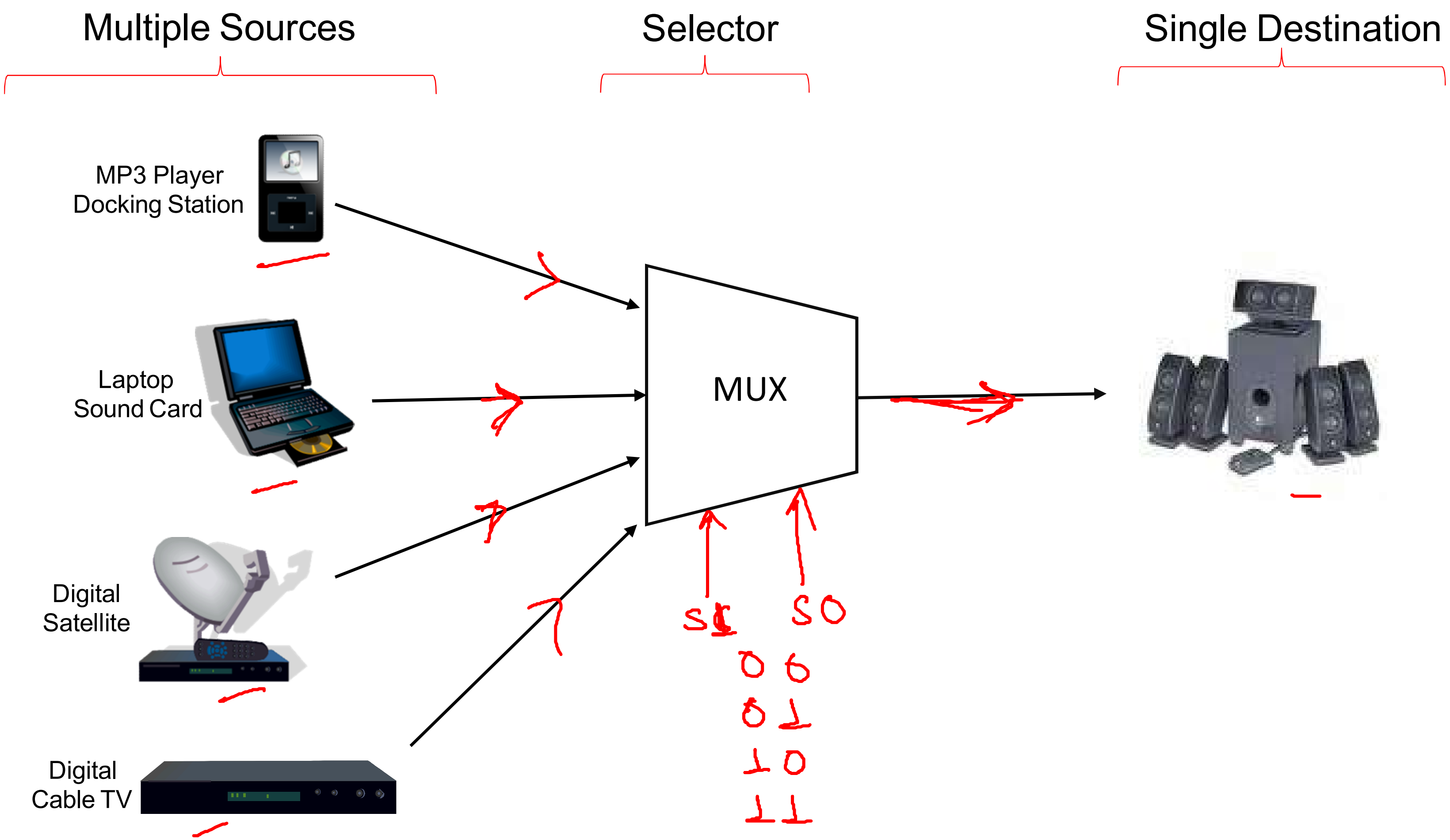
- Multiplexer is a Combinational circuit that selects binary information from one of many input lines and directs it to o/p line.
- Selection of input line is controlled by group of select lines.
- Select lines = n
- No. of inputs = $2^n$
- Types of MUX:-

  - 2-to-1 (1 select line)
  - 4-to-1 (2 select lines)
  - 8-to-1 (3 select lines)
  - 16-to-1 (4 select lines)



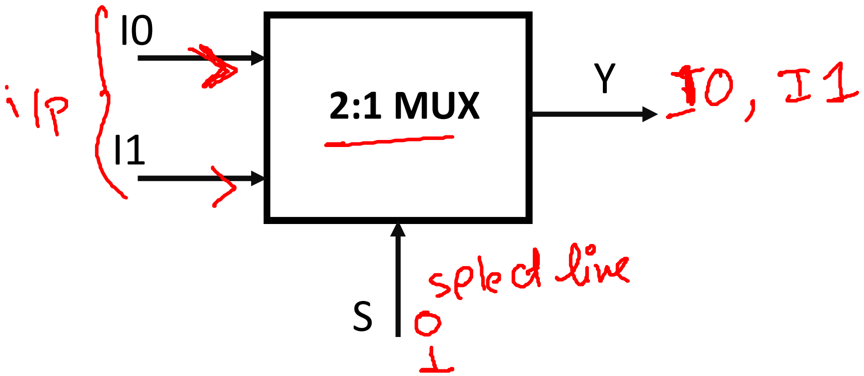
# Multiplexers

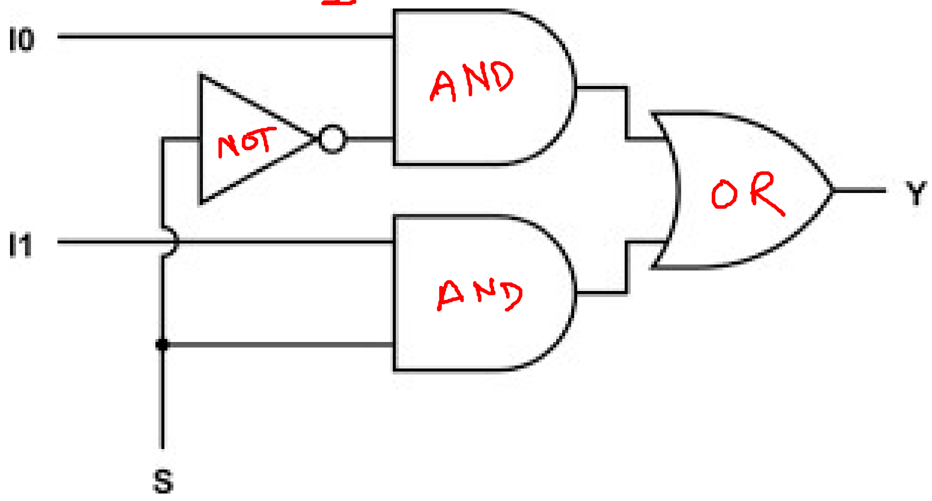
.



# Multiplexers

- **2:1 MUX**



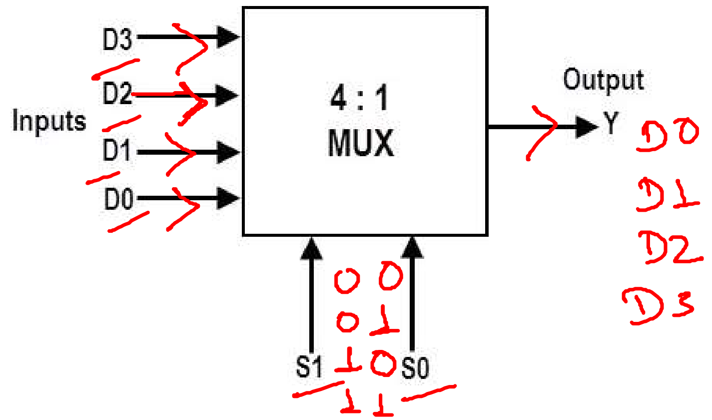




| S | I0 | I1 | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | ~~0~~ 1 |

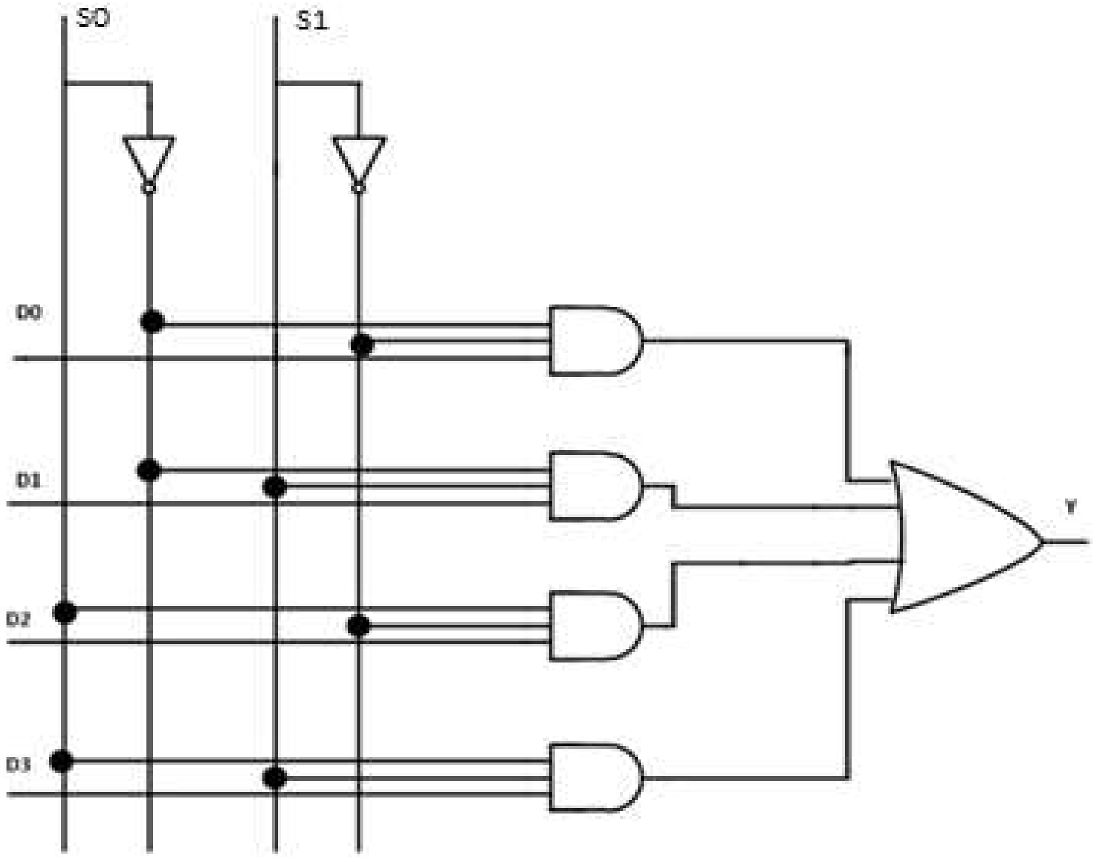
| S | Y |
|---|---|
| 0 | I0 |
| 1 | I1 |

# Multiplexers

- **4-to-1 (2 select lines)**




| Select data inputs | | Output |
|---|---|---|
| **S1** | **S0** | **Y** |
| 0 | 0 | D0 |
| 0 | 1 | D1 |
| 1 | 0 | D2 |
| 1 | 1 | D3 |

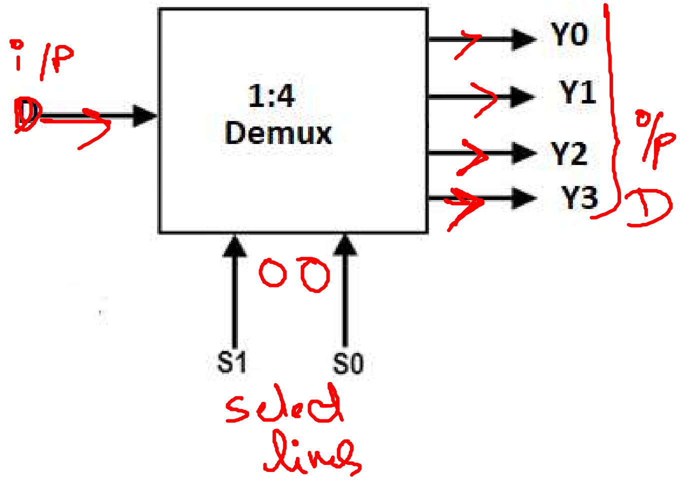


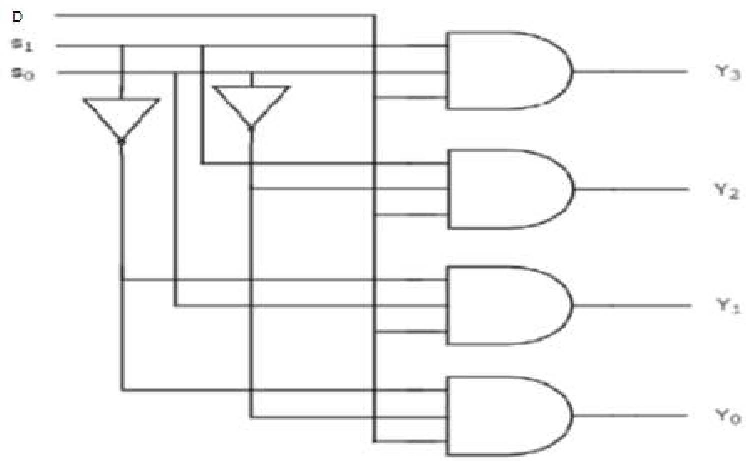
# Demultiplexers

- A DEMUX is a digital switch with a single input (source) and a multiple outputs (destinations).
- Selection of output line is controlled by select lines.
- Select lines = n,
- no. of outputs = $2^n$
- Types of DEMUX:-

| S1 | S0 | Y3 | Y2 | Y1 | Y0 |
|---|---|---|---|---|---|
| 0 | 0 | D | 0 | 0 | 0 |
| 0 | 1 | 0 | D | 0 | 0 |
| 1 | 0 | 0 | 0 | D | 0 |
| 1 | 1 | 0 | 0 | 0 | D |

  - 1-to-2 (1 select line)
  - 1-to-4 (2 select lines)
  - 1-to-8 (3 select lines)
  - 1-to-16 (4 select lines)



# Demultiplexers