KTU NOTES

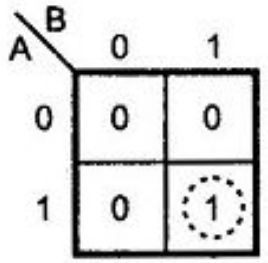
The learning companion.

KTU STUDY MATERIALS | SYLLABUS | LIVE NOTIFICATIONS | SOLVED QUESTION PAPERS

HALF ADDER

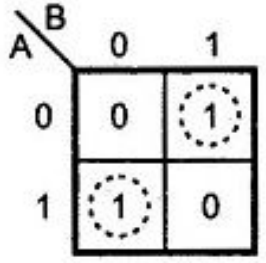
Inputs		Outputs	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

For Carry



Carry = AB

For Sum



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

FULL ADDER

Inputs			Outputs	
A	B	C_{in}	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

For Carry (C_{out})

A \ BC_{in}	00	01	11	10
0	0	0	1	0
1	0	1	1	1

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

For Sum

A \ BC_{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

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

FULL ADDER

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

$$= C_{in}\,(\bar{A}\,\bar{B} + AB) + \bar{C}_{in}\,(\bar{A}\,B + A\,\bar{B})$$

$$= C_{in}\,(A \odot B) + \bar{C}_{in}\,(A \oplus B)$$

$$= C_{in}\,\overline{(A \oplus B)} + \bar{C}_{in}\,(A \oplus B)$$

$$= C_{in} \oplus (A \oplus B)$$

$$C = AB\bar{C}_{in} + A\bar{B}C_{in} + \bar{A}BC_{in} + ABC_{in}$$

$$= AB + (A \oplus B)\,C_{in}$$

HALF SUBTRACTOR

Half Subtractor Truth Table:

Inputs		Outputs	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

HALF SUBTRACTOR

K-map simplification for half-subtractor:

For Difference

A \ B	0	1
0	0	(1)
1	(1)	0

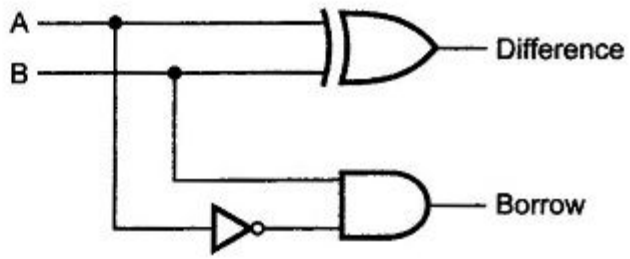
Difference = $A\overline{B} + \overline{A}B$
= $A \oplus B$

For Borrow

A \ B	0	1
0	0	(1)
1	0	0

Borrow = $\overline{A}B$

Half Subtractor Logic Diagram:



FULL SUBTRACTOR

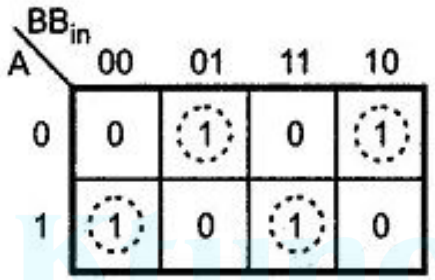
Full Subtractor Truth Table:

Inputs			Outputs	
A	B	B_{in}	D	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

FULL SUBTRACTOR

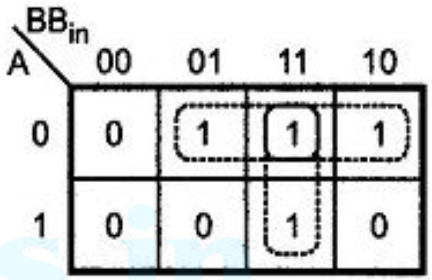
K-map simplification of D and B_{out}:

For D



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

For B_{out}

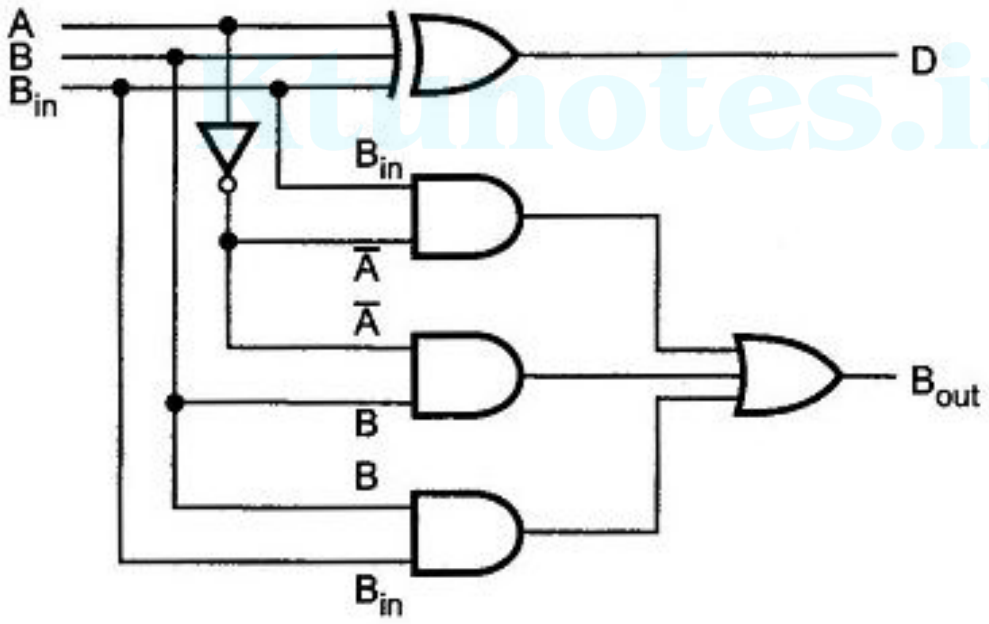


$B_{out} = \overline{A}B_{in} + \overline{A}B + BB_{in}$

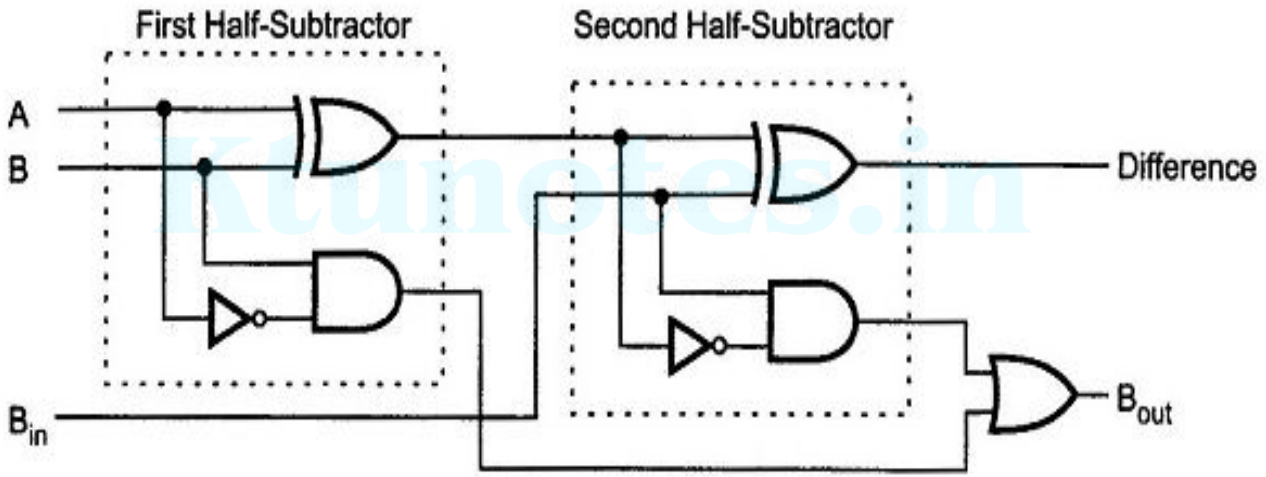
$$
\begin{aligned}
D &= \overline{A}\,\overline{B}\,B_{in} + \overline{A}\,B\,\overline{B}_{in} + A\,\overline{B}\,\overline{B}_{in} + A\,B\,B_{in} \\
&= B_{in}\,(\overline{A}\,\overline{B} + AB) + \overline{B}_{in}\,(\overline{A}\,B + A\,\overline{B}) \\
&= B_{in}\,(A \odot B) + \overline{B}_{in}\,(A \oplus B) \\
&= B_{in}\,(\overline{A \oplus B}) + \overline{B}_{in}\,(A \oplus B) \\
&= B_{in} \oplus (A \oplus B)
\end{aligned}
$$

FULL SUBTRACTOR

Full Subtractor Logic Diagram:



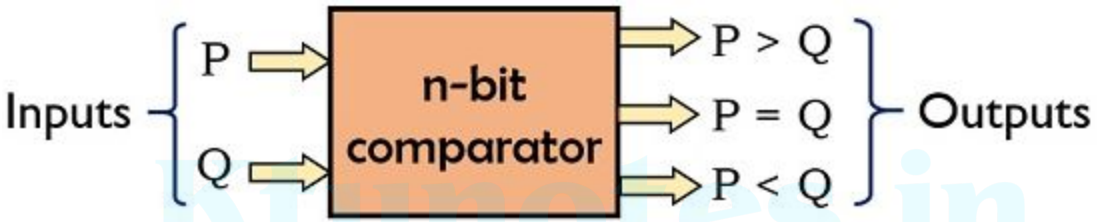
Implementation of Full Subtractor with Two Half Subtractors



MAGNITUDE COMPARATOR

- A magnitude comparator makes the comparison by considering all the factors
- It shows results for either greater, equal or lesser than value by comparing the magnitude of two inputs
- Hence contains 3 output pins and accordingly, any one of the 3 output pins of a magnitude comparator becomes high.
- Suppose P and Q are the two inputs of magnitude comparator, 3 outputs will be $P > Q$, $P = Q$ and $P < Q$
- Depending upon the comparison performed, any one of the given outputs will be high.

MAGNITUDE COMPARATOR

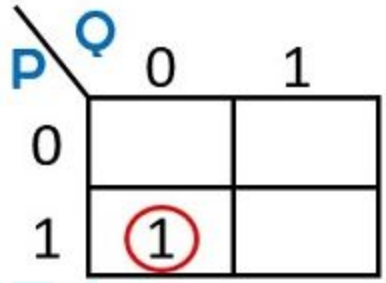


1-bit Magnitude Comparator

P	Q	P > Q	P = Q	P < Q
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

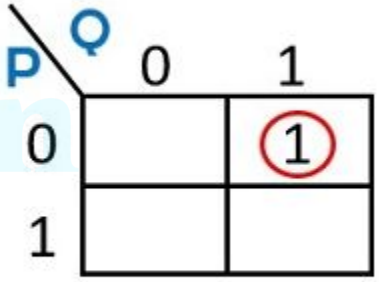
K-map representation of 1-bit Magnitude Comparator

For P > Q



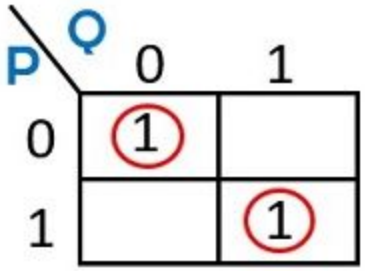
$P > Q \rightarrow P\overline{Q}$

For P < Q



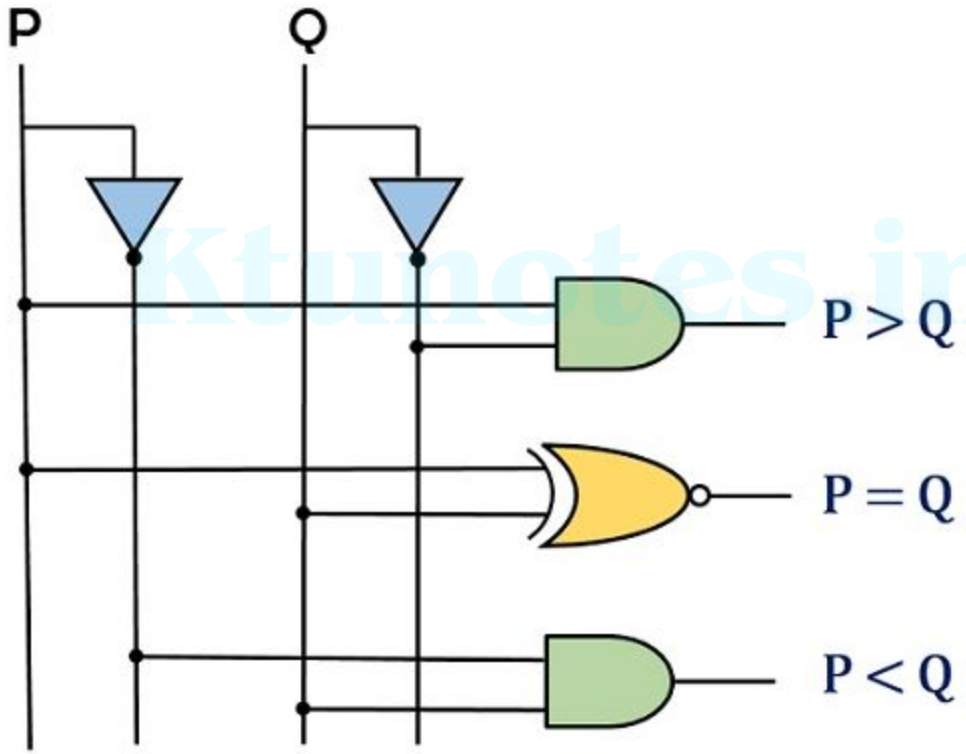
$P < Q \rightarrow \overline{P}Q$

For P = Q



$P = Q \rightarrow \overline{P}\,\overline{Q} + PQ$

1-bit Magnitude Comparator Circuit

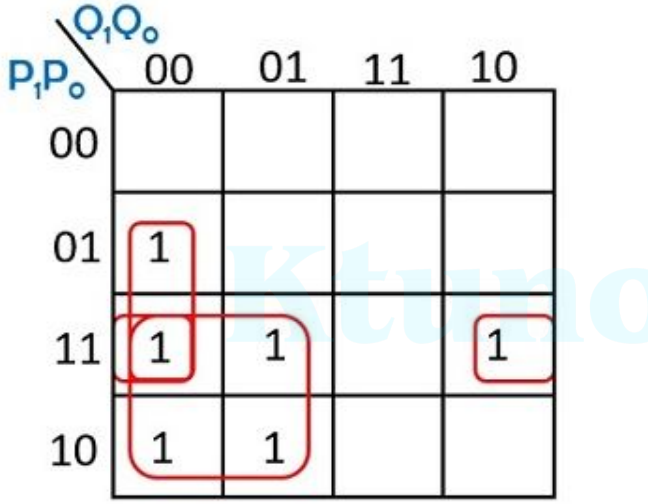


2-bit Magnitude Comparator

P_1	P_0	Dec equi. of P	Q_1	Q_0	Dec equi. of Q	P > Q	P = Q	P < Q
0	0	0	0	0	0	0	1	0
0	0	0	0	1	1	0	0	1
0	0	0	1	0	2	0	0	1
0	0	0	1	1	3	0	0	1
0	1	1	0	0	0	1	0	0
0	1	1	0	1	1	0	1	0
0	1	1	1	0	2	0	0	1
0	1	1	1	1	3	0	0	1
1	0	2	0	0	0	1	0	0
1	0	2	0	1	1	1	0	0
1	0	2	1	0	2	0	1	0
1	0	2	1	1	3	0	0	1
1	1	3	0	0	0	1	0	0
1	1	3	0	1	1	1	0	0
1	1	3	1	0	2	1	0	0
1	1	3	1	1	3	0	1	0

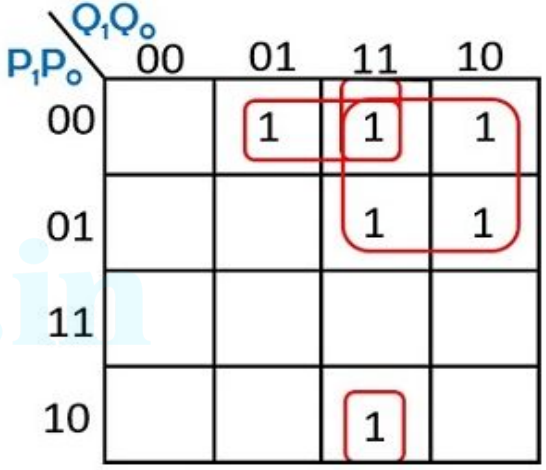
K-map representation of 2-bit Magnitude Comparator

For P > Q



$$P > Q \rightarrow P_0 \overline{Q_1}\,\overline{Q_0} + P_1 \overline{Q_1} + P_1 P_0 \overline{Q_0}$$

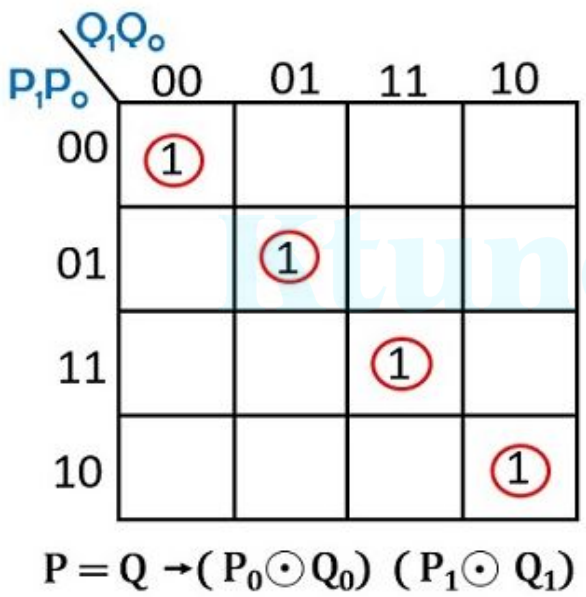
For P < Q



$$P < Q \rightarrow \overline{P_1}\,\overline{P_0}\,Q_0 + \overline{P_0}\,Q_1\,Q_0 + \overline{P_1}\,Q_1$$

K-map representation of 2-bit Magnitude Comparator

For P = Q



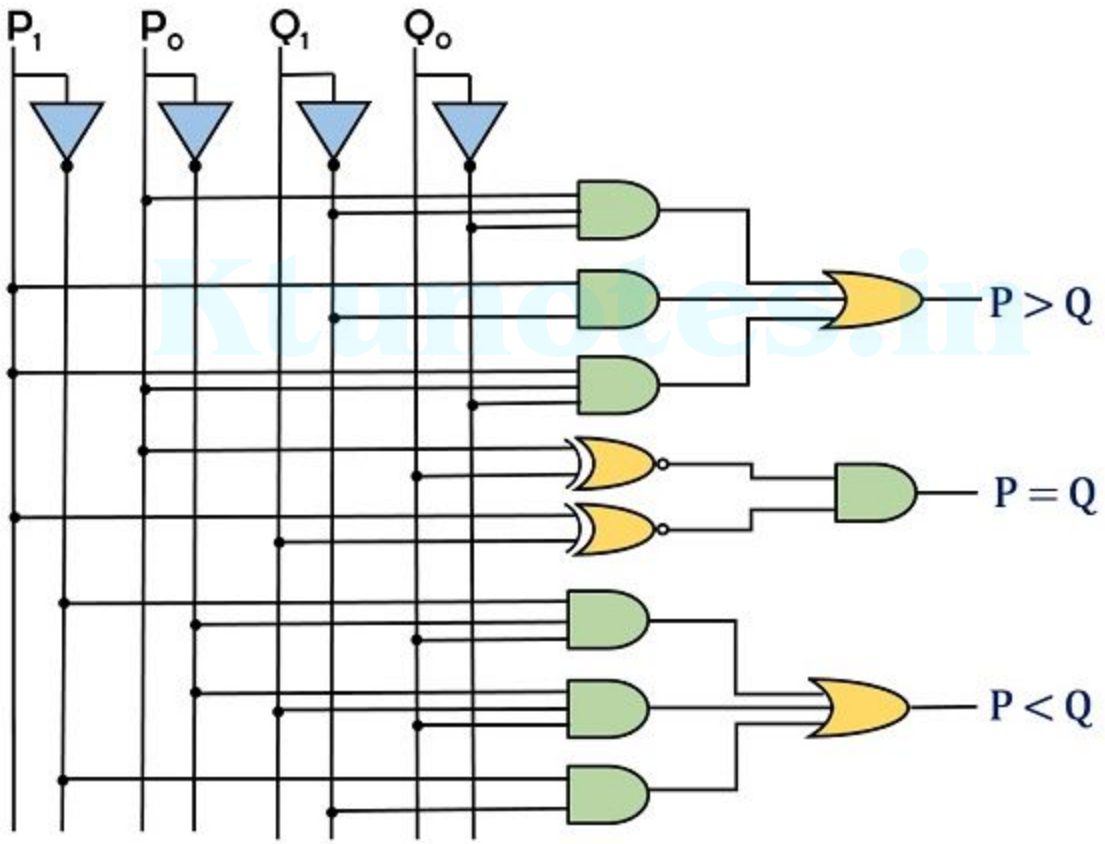
=P1'P0'Q1'Q0' + P1'P0Q1'Q0 + P1P0Q1Q0 + P1P0'Q1Q0'

=P1'Q1' (P0'Q0' + P0Q0) + P1Q1 (P0Q0 + P0'Q0')

=(P0Q0 + P0'Q0') (P1Q1 + P1'Q1')

=(P0 Ex-Nor Q0) (P1 Ex-Nor Q1)

2-bit Magnitude Comparator Circuit



4-Bit Magnitude Comparator

- **A>B** can be possible in following four cases:
  1. If A3 = 1 and B3 = 0
  2. If A3 = B3 and A2 = 1 and B2 = 0
  3. If A3 = B3, A2 = B2 and A1 = 1 and B1 = 0
  4. If A3 = B3, A2 = B2, A1 = B1 and A0 = 1 and B0 = 0

- **A<B** can be possible in following four cases:
  1. If A3 = 0 and B3 = 1
  2. If A3 = B3 and A2 = 0 and B2 = 1
  3. If A3 = B3, A2 = B2 and A1 = 0 and B1 = 1
  4. If A3 = B3, A2 = B2, A1 = B1 and A0 = 0 and B0 = 1

- **A=B** is true only when **A3=B3** and **A2=B2** and **A1=B1** and **A0=B0**.

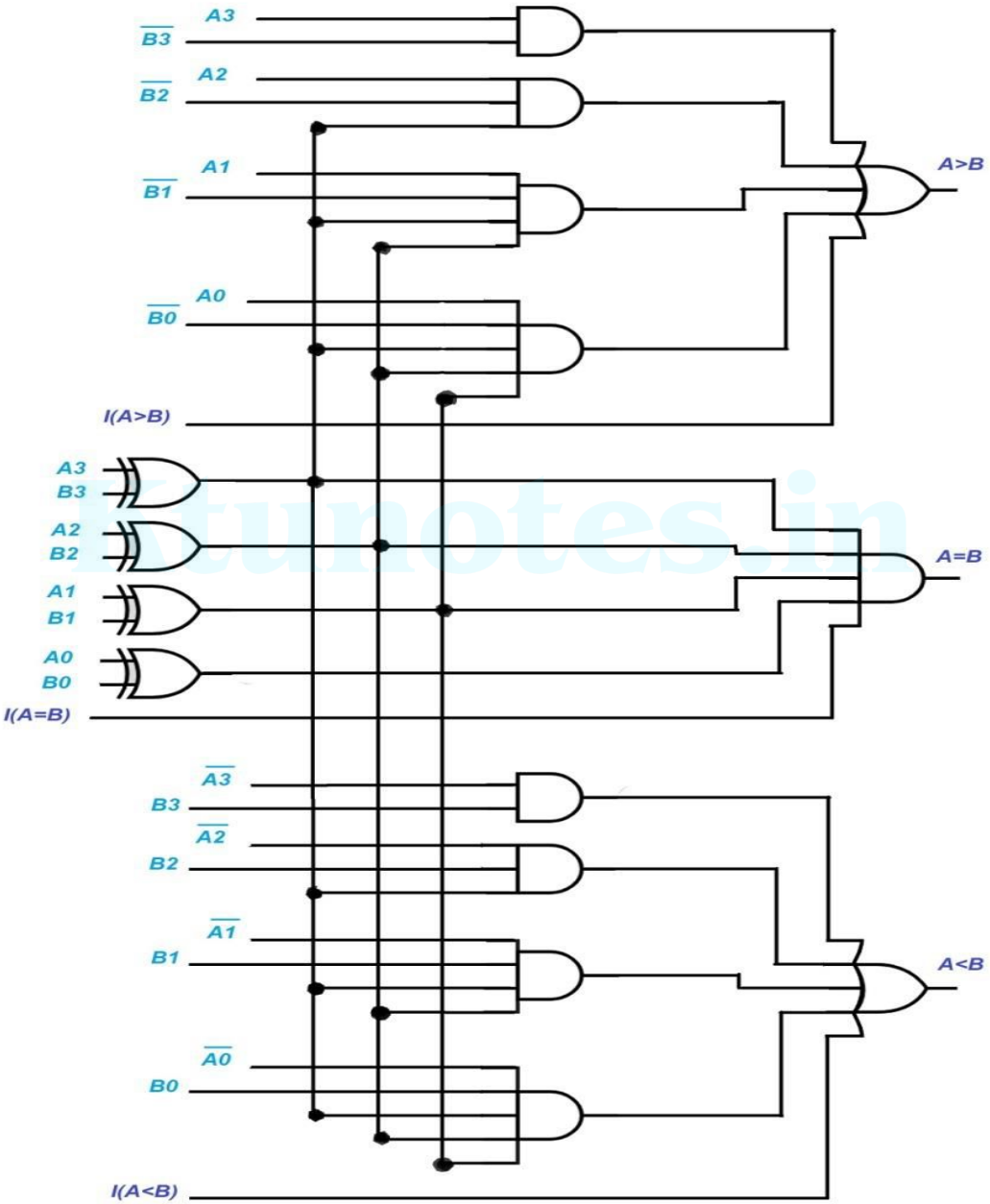
4-Bit Magnitude Comparator

A>B =A3 B3' + (A3 XNOR B3) A2 B2' +(A3 XNOR B3) (A2 XNOR B2) A1 B1' + (A3 XNOR B3) (A2 XNOR B2) (A1 XNOR B1) A0 B0'

A<B = A3' B3 + (A3 XNOR B3) A2' B2 + (A3 XNOR B3) (A2 XNOR B2) A1' B1 + (A3 XNOR B3) (A2 XNOR B2) (A1 XNOR B1) A0' B0

A=B = (A3 XNOR B3) (A2 XNOR B2) (A1 XNOR B1) (A0 XNOR B0)

4-bit Magnitude Comparator Circuit



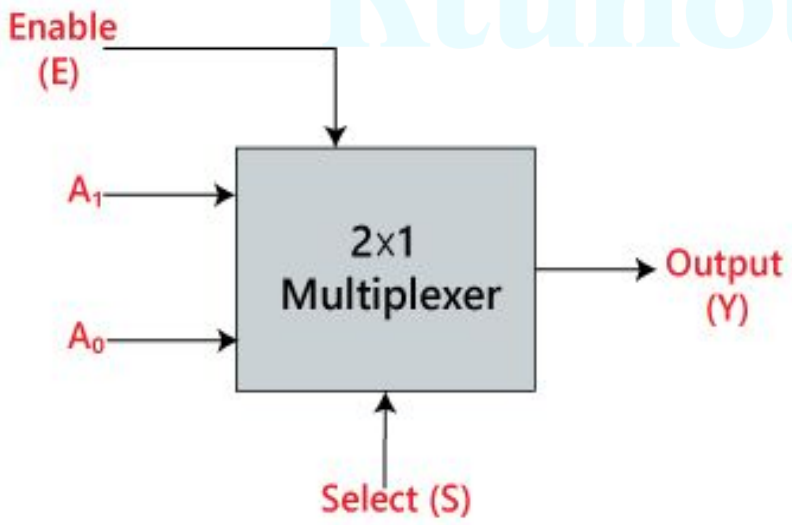
Multiplexer/Data Selector

- A multiplexer is a combinational circuit that has 2^n input lines and a single output line
- Simply, the multiplexer is a multi-input and single-output combinational circuit
- On the basis of values of selection lines, one of these data inputs will be connected to output
- There are n selection lines and 2^n input lines
- So, there is a total of 2^n possible combinations of inputs
- A multiplexer is also called as **MUX**.

2×1 Multiplexer

- In 2×1 multiplexer, there are only two inputs, A_0 and A_1, 1 selection line S_0 and single output Y

Block Diagram:



Truth Table:

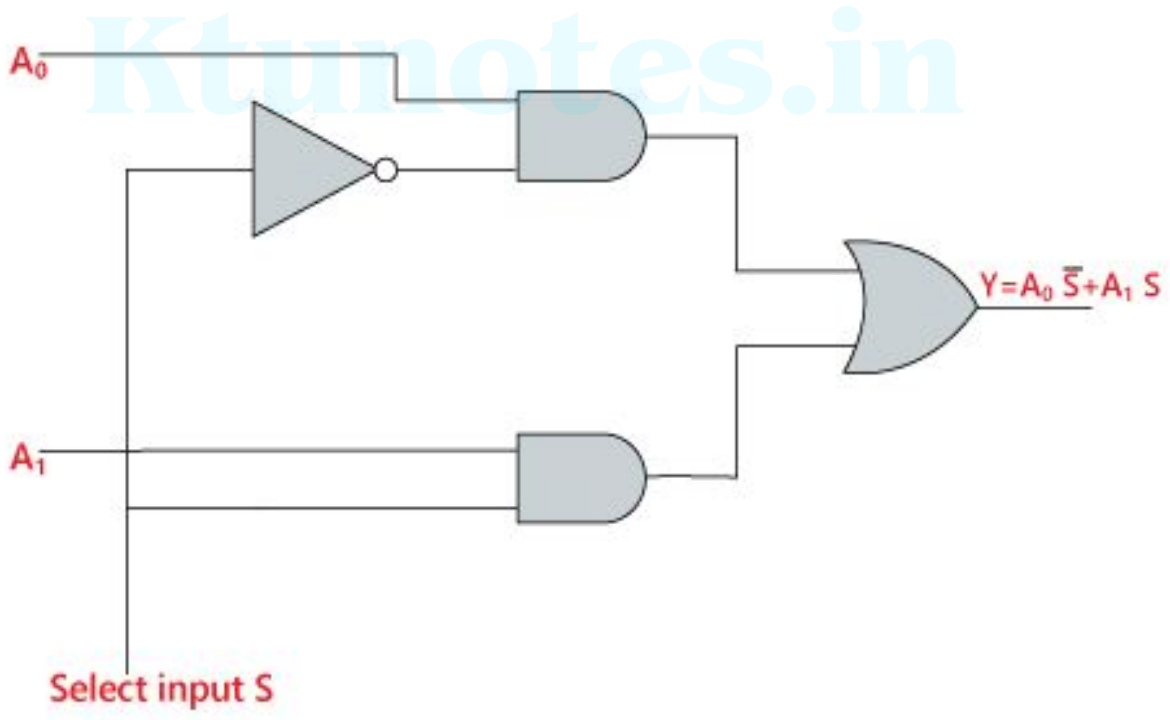
INPUTS	Output
S_0	Y
0	A_0
1	A_1

2×1 Multiplexer

- Logical expression of term Y is as follows:

  $Y=S_0'.A_0 + S_0.A_1$

- Logical circuit of above expression is

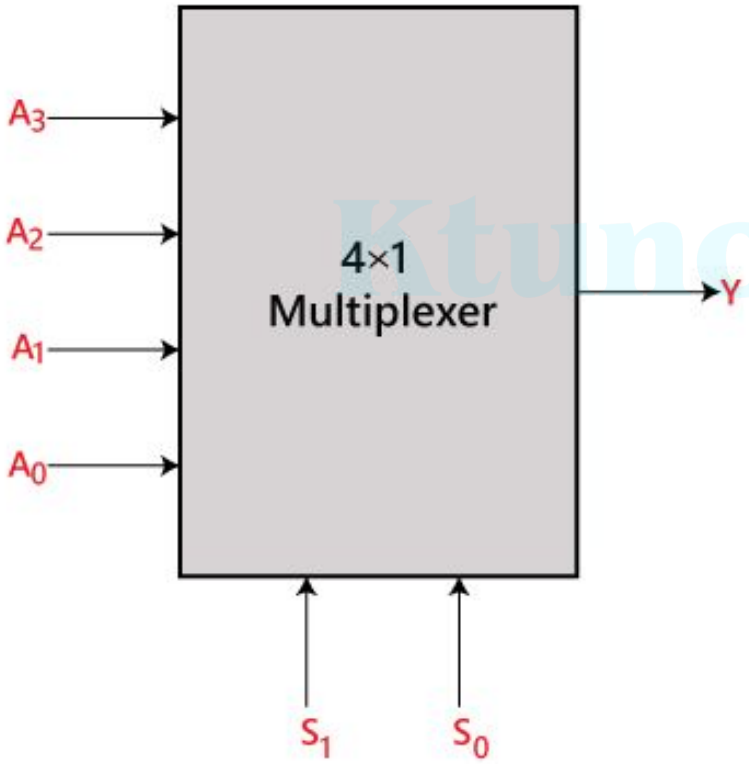


4×1 Multiplexer

- In the 4×1 multiplexer, there is a total of four inputs A_0, A_1, A_2, and A_3, 2 selection lines S_0 and S_1 and single output Y
- On basis of combination of inputs that are present at selection lines S_0 and S_1, one of these 4 inputs are connected to output

4×1 Multiplexer

Block Diagram:



Truth Table:

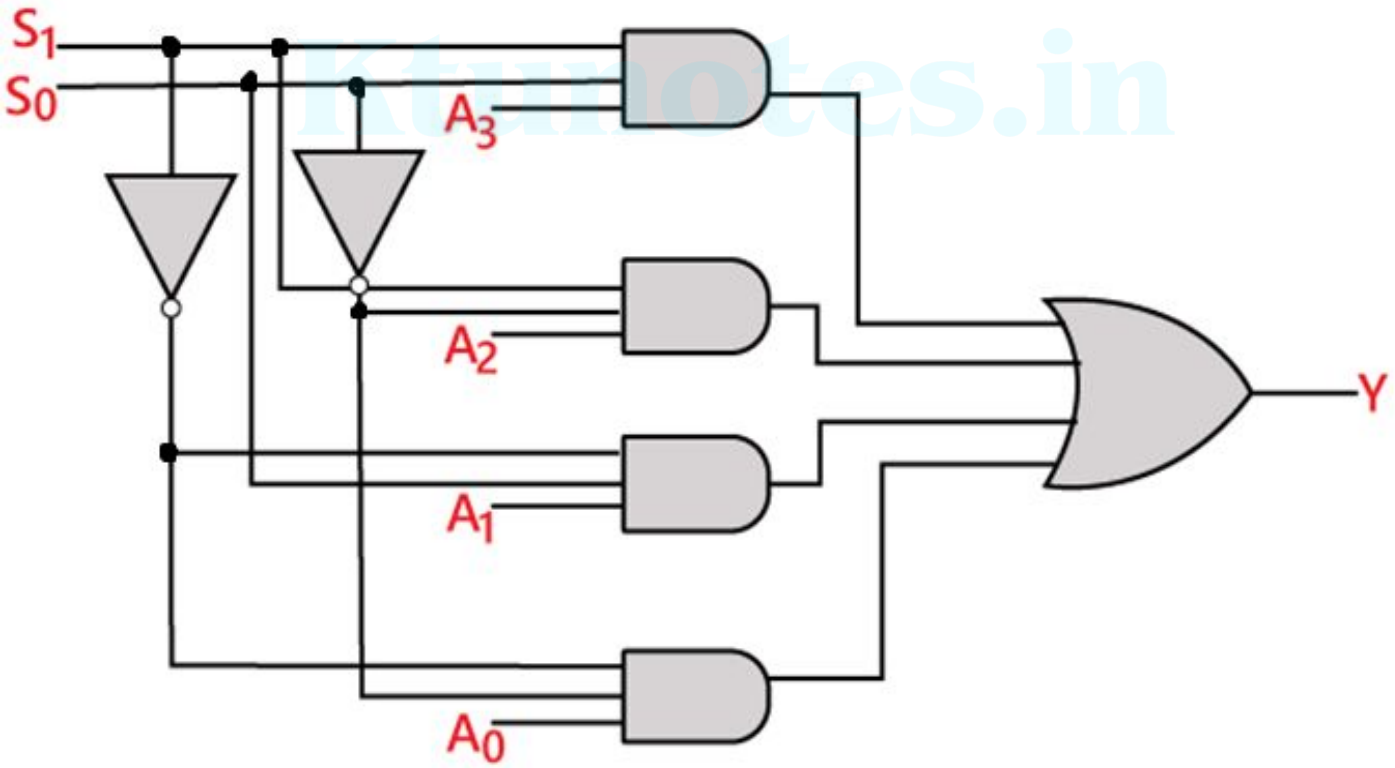
INPUTS		Output
S_1	S_0	Y
0	0	A_0
0	1	A_1
1	0	A_2
1	1	A_3

4×1 Multiplexer

- Logical expression of term Y is as follows

$$Y = S_1' S_0' A_0 + S_1' S_0 A_1 + S_1 S_0' A_2 + S_1 S_0 A_3$$

- Logical circuit of above expression is

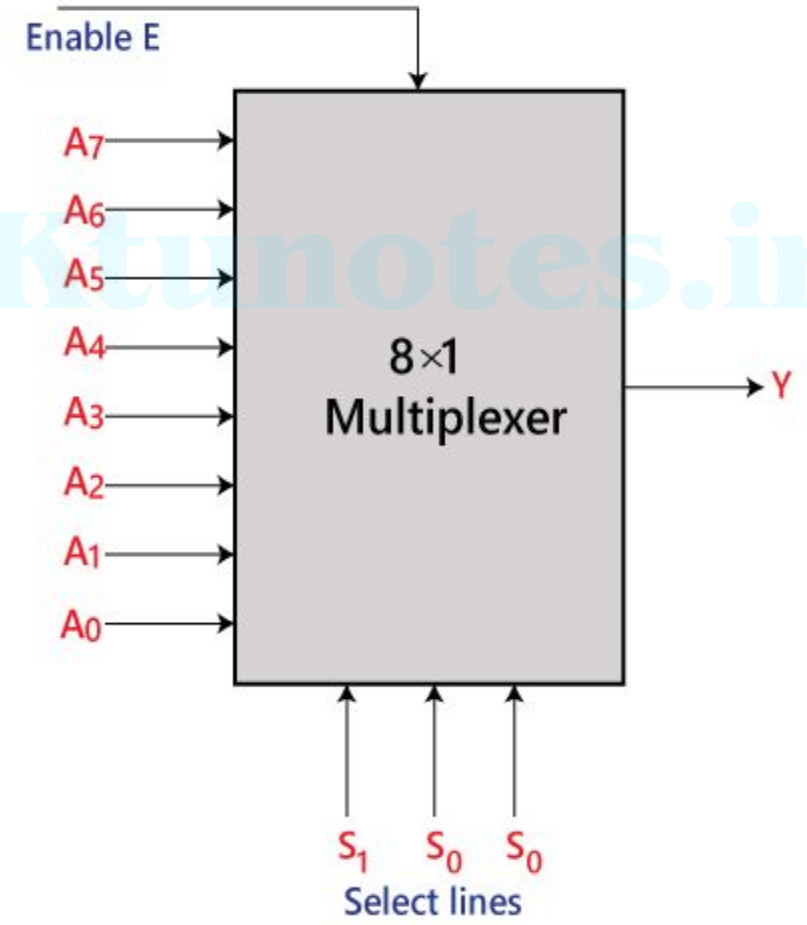


8 to 1 Multiplexer

- In 8 to 1 multiplexer, there are total eight inputs A_0, A_1, A_2, A_3, A_4, A_5, A_6, and A_7, 3 selection lines S_0, S_1 and S_2 and single output Y
- On basis of combination of inputs that are present at selection lines S_0, S_1 and S_2, one of these 8 inputs are connected to output

8 to 1 Multiplexer

Block Diagram:



8 to 1 Multiplexer

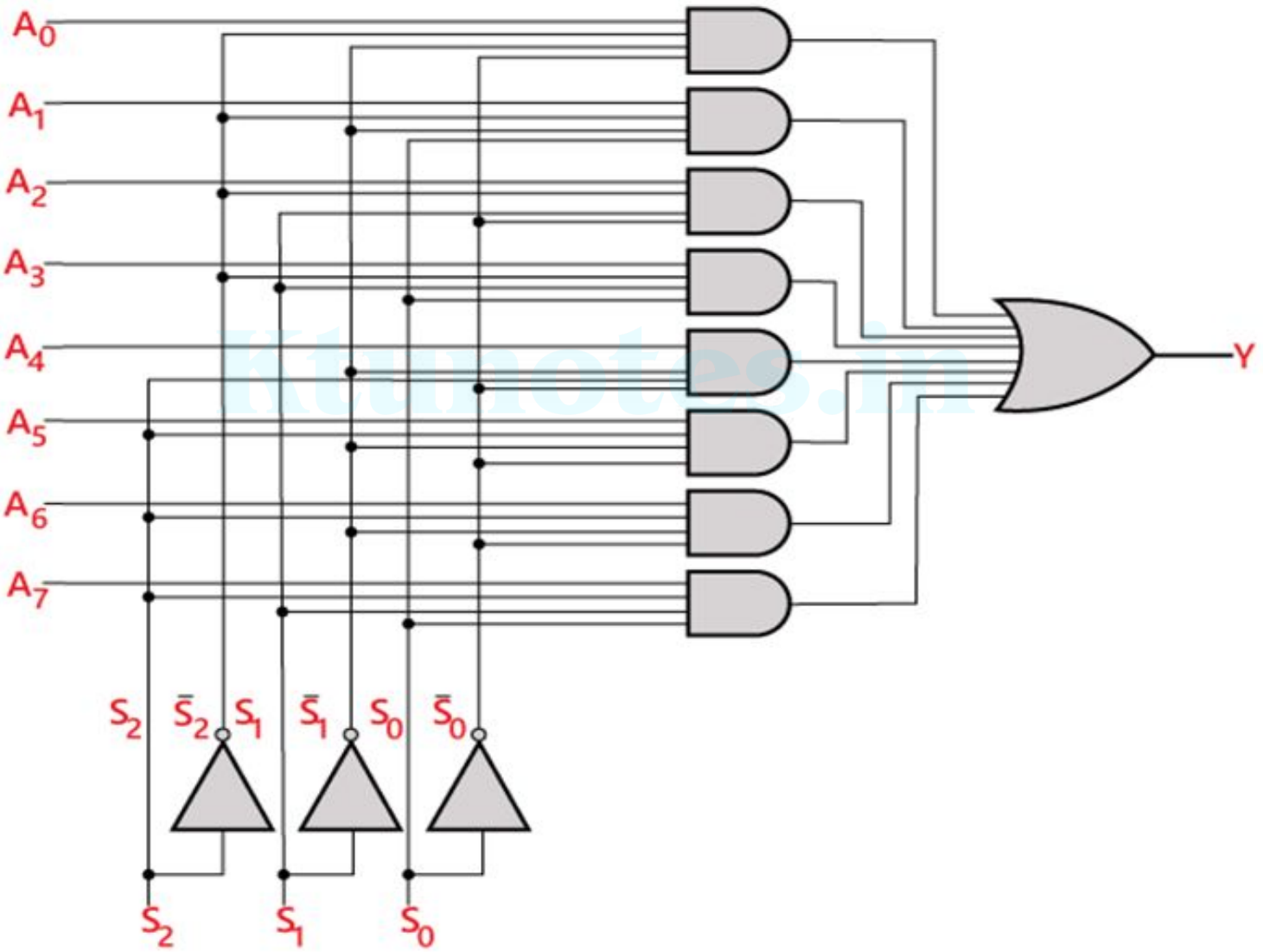
Truth Table:

INPUTS			Output
S_2	S_1	S_0	Y
0	0	0	A_0
0	0	1	A_1
0	1	0	A_2
0	1	1	A_3
1	0	0	A_4
1	0	1	A_5
1	1	0	A_6
1	1	1	A_7

- Logical expression of term Y is as follows:

$$Y = S_2'.S_1'.S_0'.A_0 + S_2'.S_1'.S_0.A_1 + S_2'.S_1.S_0'.A_2 + S_2'.S_1.S_0.A_3 + S_2.S_1'.S_0'.A_4 + S_2.S_1'.S_0\ A_5 + S_2.S_1.S_0'.A_6 + S_2.S_1.S_0.A_7$$

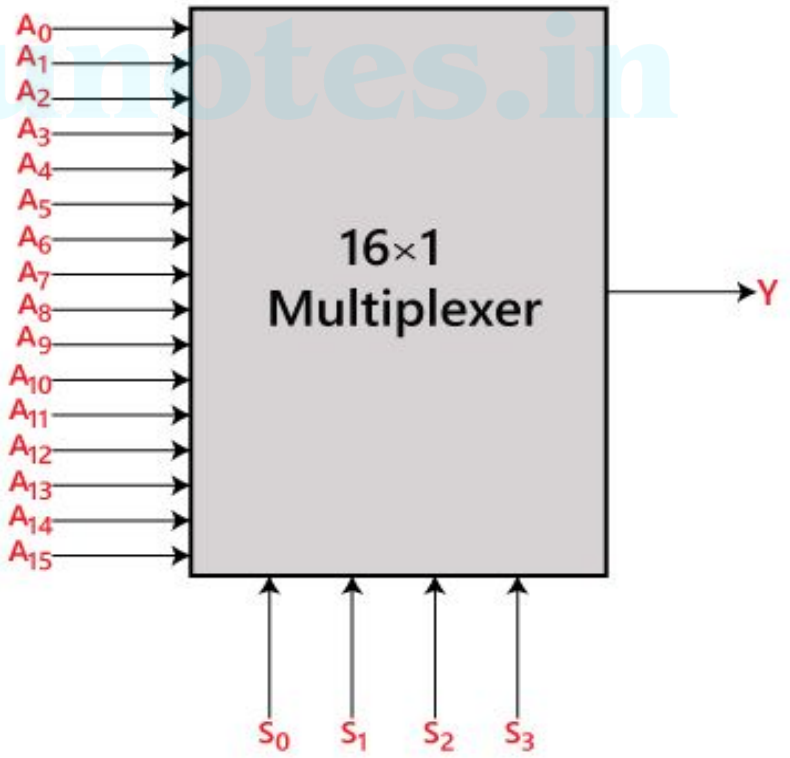
8 to 1 Multiplexer



16 to 1 Multiplexer

- In 16 to 1 multiplexer, there are total of 16 inputs A_0, A_1, …, A_{15}, 4 selection lines S_0, S_1, S_2, and S_3 and single output Y

Block Diagram:



16 to 1 Multiplexer

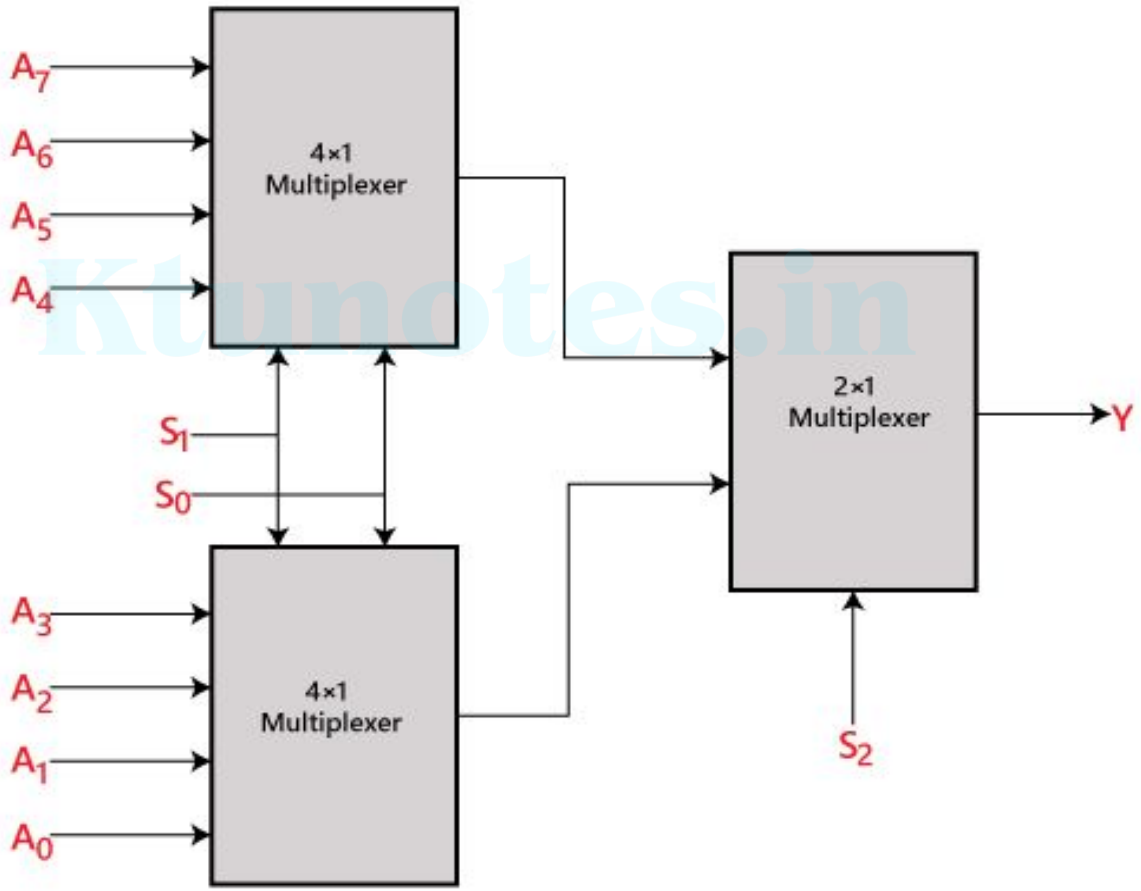
Truth Table:

INPUTS				Output
S_0	S_1	S_2	S_3	Y
0	0	0	0	A_0
0	0	0	1	A_1
0	0	1	0	A_2
0	0	1	1	A_3
0	1	0	0	A_4
0	1	0	1	A_5
0	1	1	0	A_6
0	1	1	1	A_7
1	0	0	0	A_8
1	0	0	1	A_9
1	0	1	0	A_{10}
1	0	1	1	A_{11}
1	1	0	0	A_{12}
1	1	0	1	A_{13}
1	1	1	0	A_{14}
1	1	1	1	A_{15}

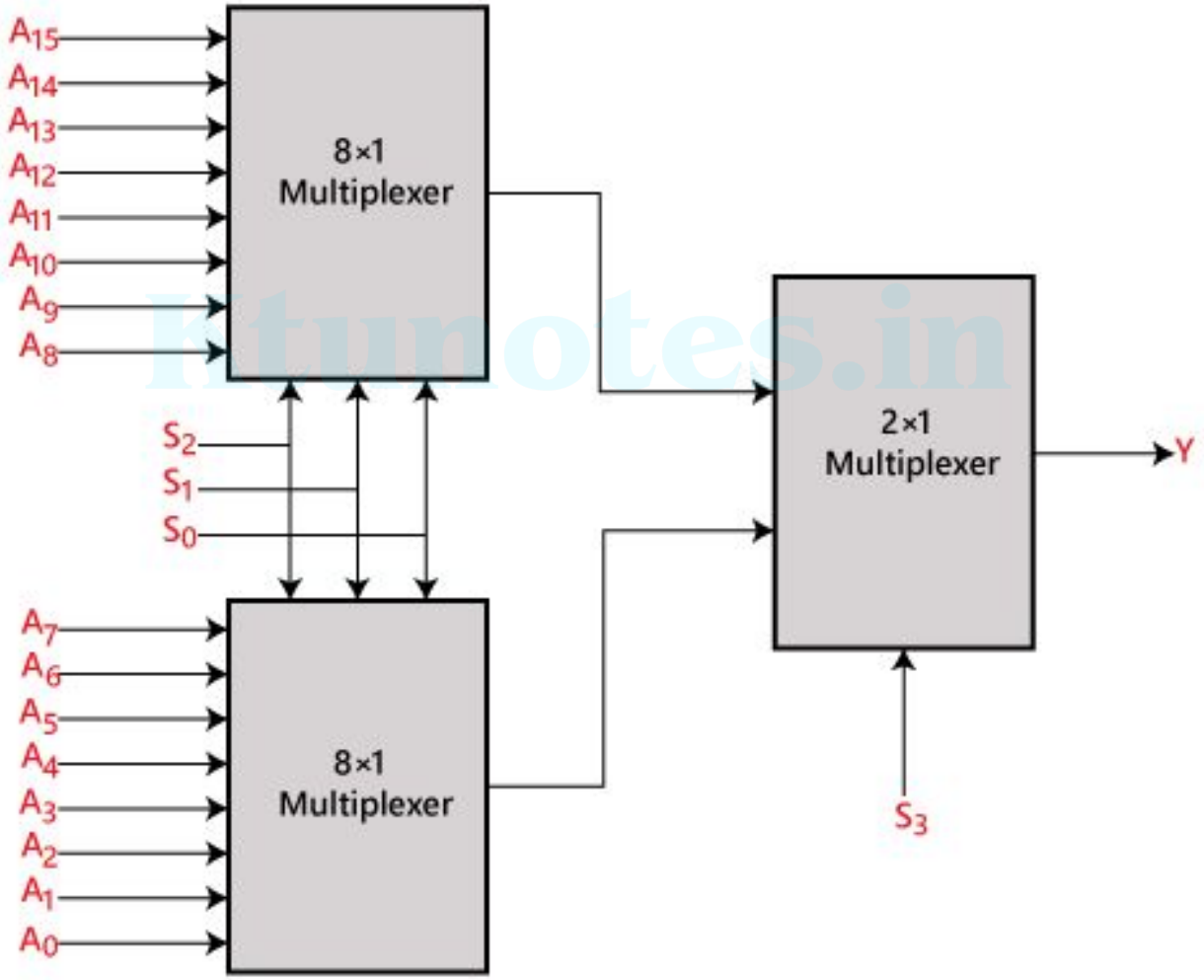
8 ×1 multiplexer using 4×1 and 2×1 multiplexer

- To implement 8×1 multiplexer, we need two 4×1 multiplexers and one 2×1 multiplexer
- 4×1 multiplexer has 2 selection lines, 4 inputs, and 1 output
- 2×1 multiplexer has only 1 selection line
- For getting 8 data inputs, we need two 4×1 multiplexers
- 4×1 multiplexer produces one output
- So, in order to get final output, we need a 2×1 multiplexer

8 ×1 multiplexer using 4×1 and 2×1 multiplexer



16×1 multiplexer using 8×1 and 2×1 multiplexer



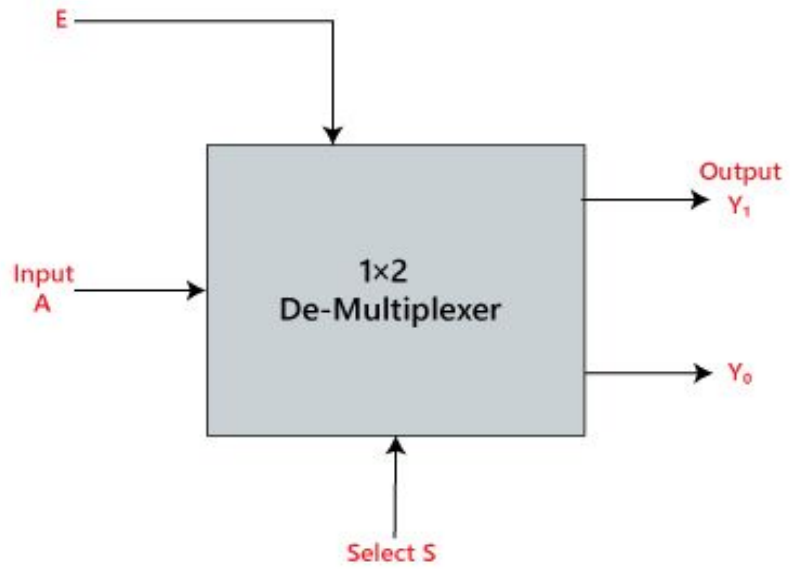
Demultiplexer/Data Distributor

- A De-multiplexer is a combinational circuit that has only 1 input line and 2^n output lines
- De-multiplexer is a single-input and multi-output combinational circuit
- Information is received from single input line and directed to output lines
- On the basis of the values of the selection lines, input will be connected to one of these outputs
- De-multiplexer is opposite to multiplexer
- There are n selection lines and 2^n output lines
- So, there is a total of 2^n possible combinations of outputs
- De-multiplexer is also known as **DE-MUX**

1×2 De-multiplexer

- In 1 to 2 De-multiplexer, there are only two outputs Y_0, and Y_1, 1 selection line S_0, and single input A
- On the basis of selection value, input will be connected to one of outputs

Block Diagram:



Truth Table:

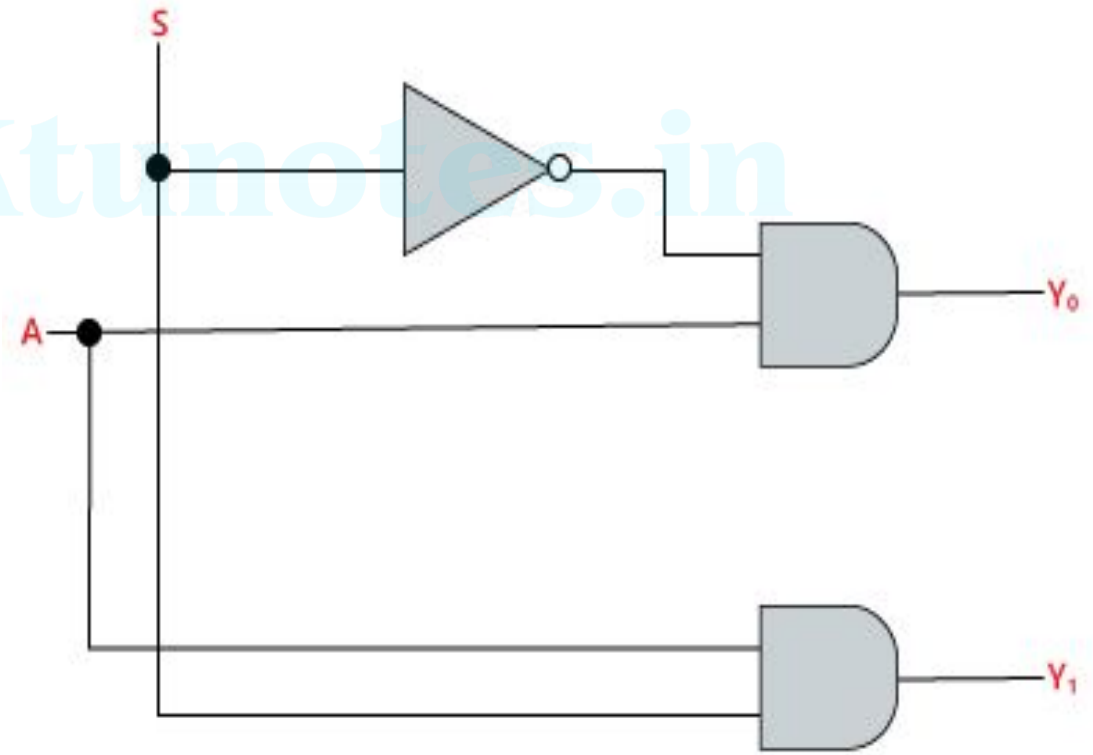
INPUTS	Output	
S_0	Y_1	Y_0
0	0	A
1	A	0

1×2 De-multiplexer

- The logical expression of the term Y is as follows:

  $Y_0 = S_0'.A$

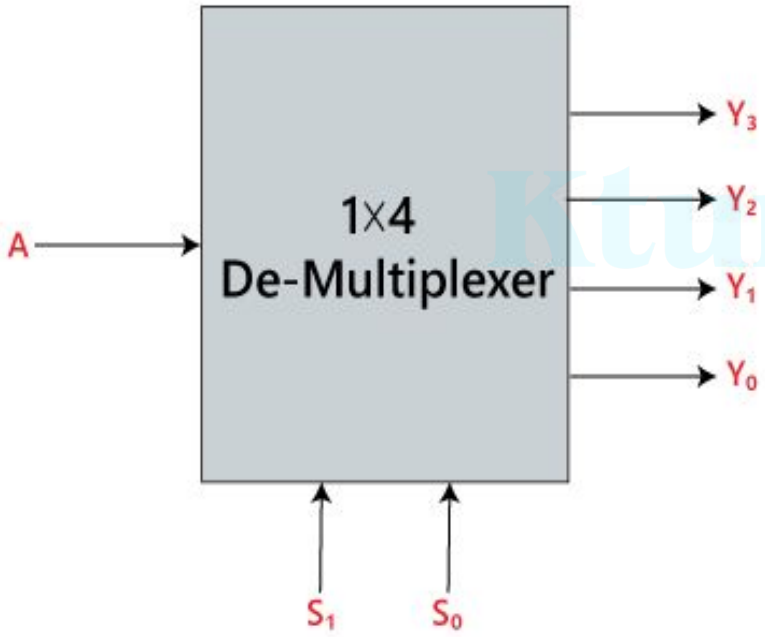
  $Y_1 = S_0.A$



1×4 De-multiplexer

- In 1 to 4 De-multiplexer, there are total of four outputs Y_0, Y_1, Y_2, and Y_3, 2 selection lines S_0 and S_1 and single input A
- On basis of combination of inputs which are present at selection lines S_0 and S_1, input be connected to one of outputs

1×4 De-multiplexer



Truth Table:

INPUTS		Output			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	A
0	1	0	0	A	0
1	0	0	A	0	0
1	1	A	0	0	0

1×4 De-multiplexer

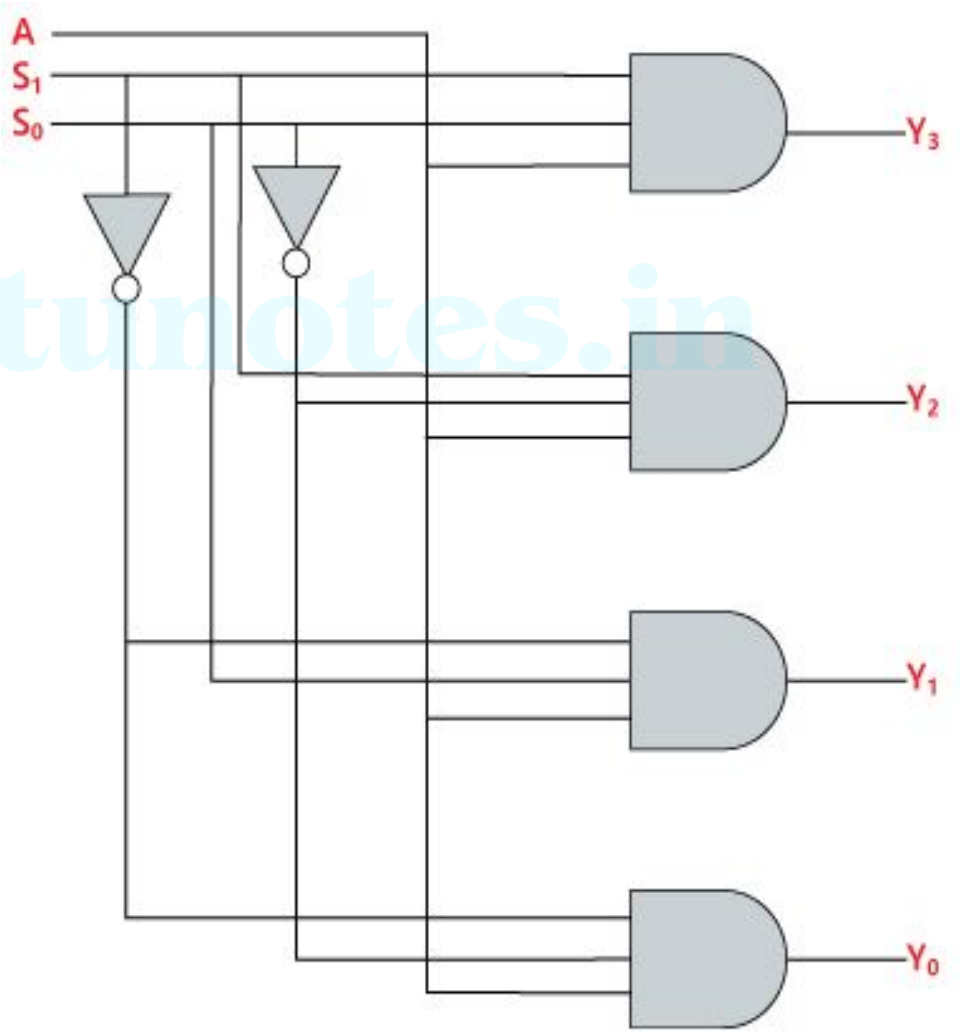
- Logical expression of term Y is as follows:

$Y_0 = S_1' S_0' A$

$y_1 = S_1' S_0 A$

$y_2 = S_1 S_0' A$

$y_3 = S_1 S_0 A$

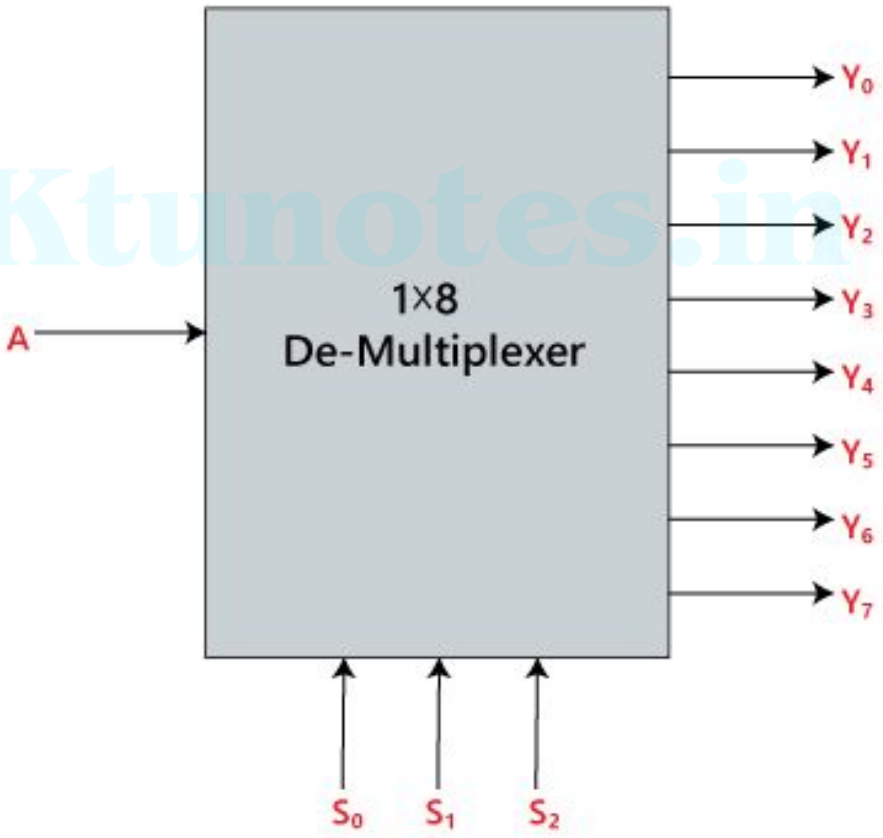


1×8 De-multiplexer

- In 1 to 8 De-multiplexer, there are total of eight outputs Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, and Y_7, 3 selection lines S_0, S_1 and S_2 and single input A
- On basis of combination of inputs which are present at selection lines S_0, S_1 and S_2, input will be connected to one of these outputs

1×8 De-multiplexer

Block Diagram:



1×8 De-multiplexer

Truth Table:

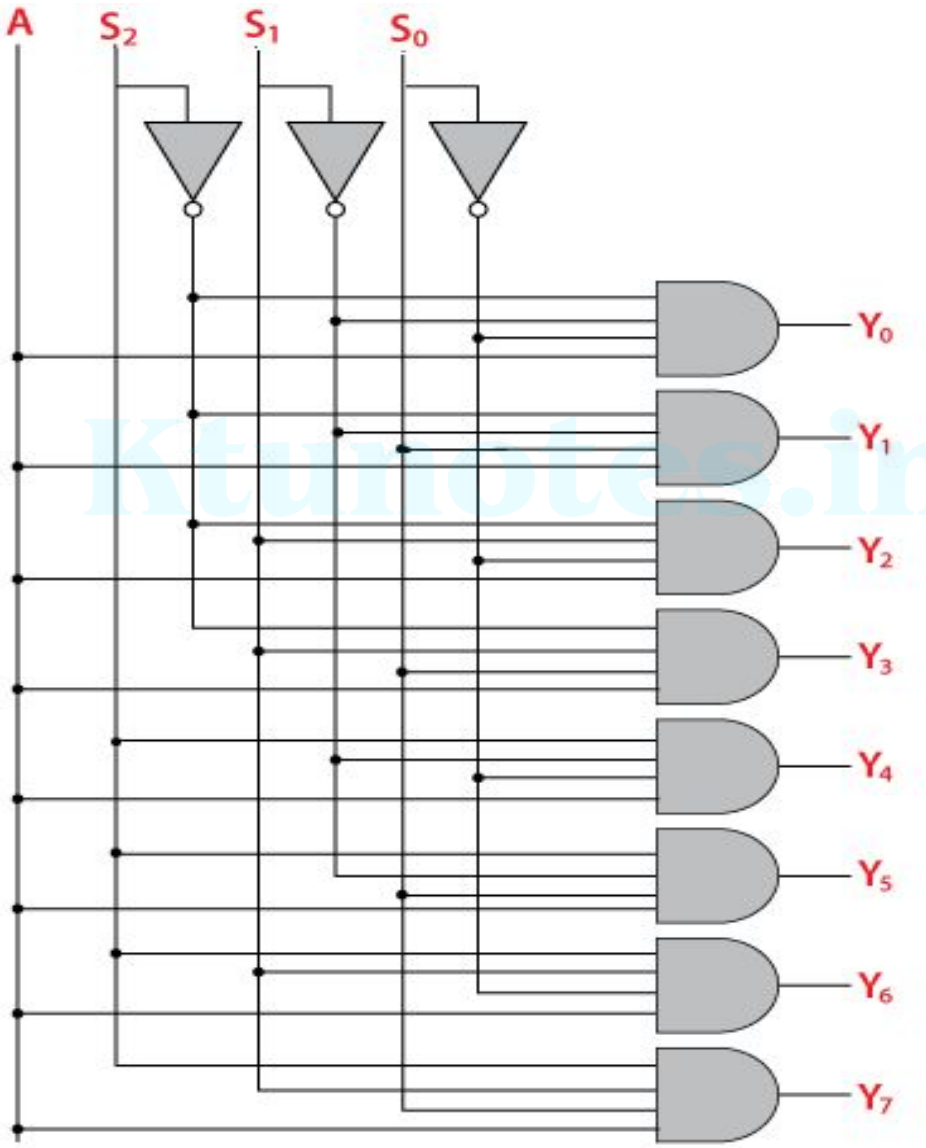
INPUTS			Output							
S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	A
0	0	1	0	0	0	0	0	0	A	0
0	1	0	0	0	0	0	0	A	0	0
0	1	1	0	0	0	0	A	0	0	0
1	0	0	0	0	0	A	0	0	0	0
1	0	1	0	0	A	0	0	0	0	0
1	1	0	0	A	0	0	0	0	0	0
1	1	1	A	0	0	0	0	0	0	0

1×8 De-multiplexer

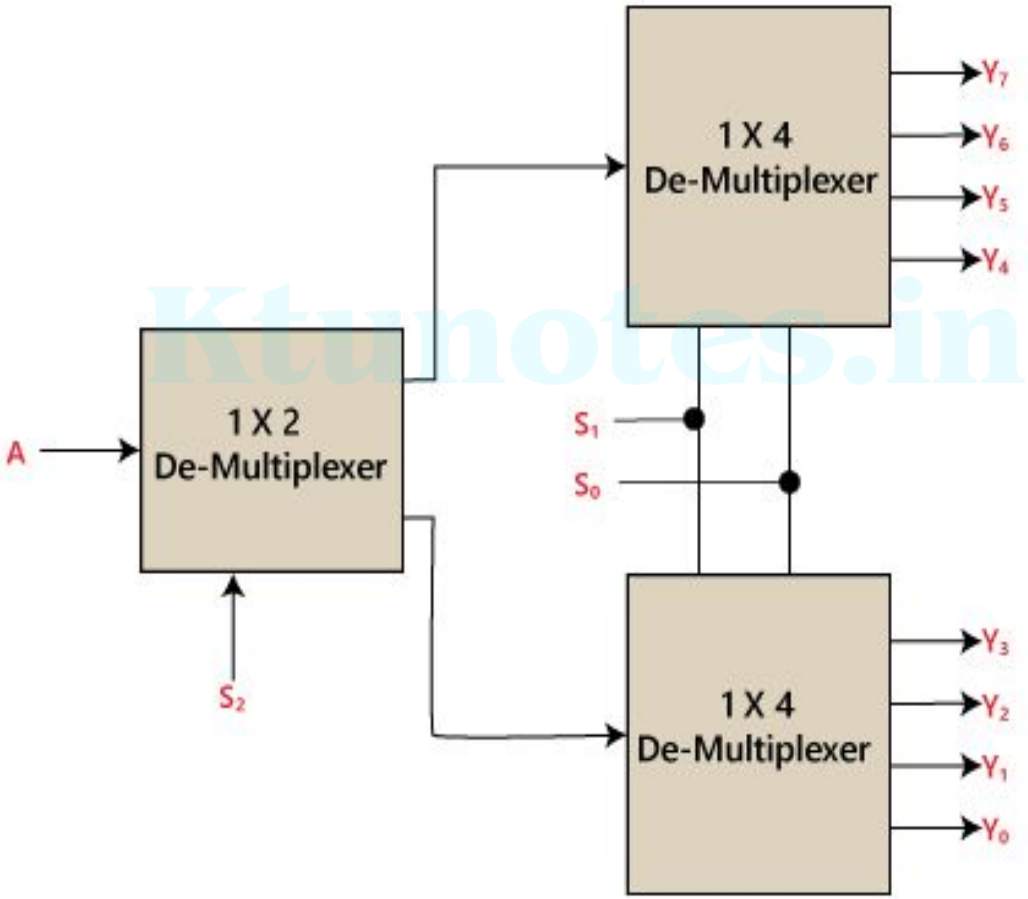
- Logical expression of term Y is as follows:

$Y_0 = S_0'.S_1'.S_2'.A$
$Y_1 = S_0.S_1'.S_2'.A$
$Y_2 = S_0'.S_1.S_2'.A$
$Y_3 = S_0.S_1.S_2'.A$
$Y_4 = S_0'.S_1'.S_2\ A$
$Y_5 = S_0.S_1'.S_2\ A$
$Y_6 = S_0'.S_1.S_2\ A$
$Y_7 = S_0.S_1.S_3.A$

1×8 De-multiplexer



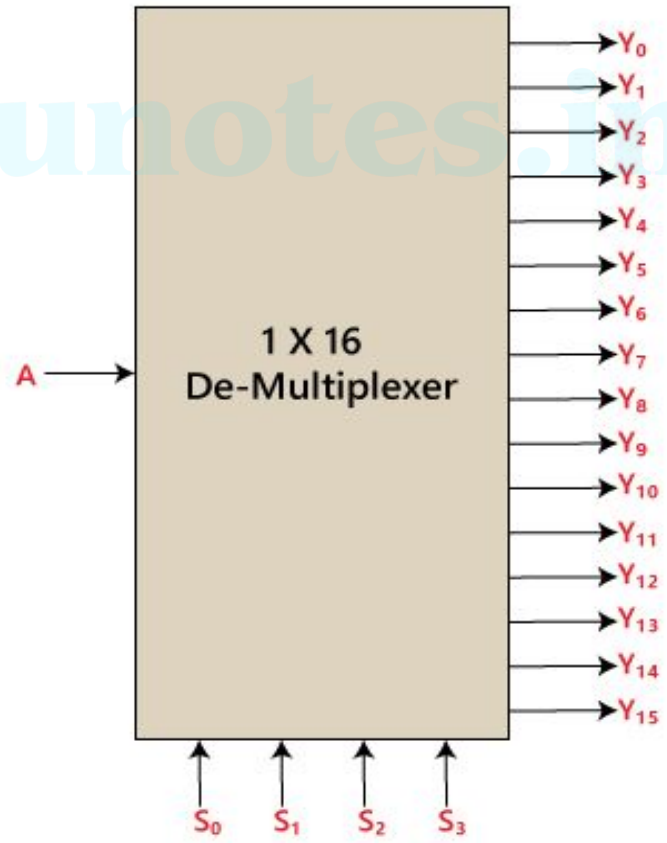
1×8 De-multiplexer using 1×4 and 1×2 de-multiplexer



1 x 16 De-multiplexer

- In 1×16 de-multiplexer, there are total of 16 outputs Y_0, Y_1, …, Y_{15}, 4 selection lines S_0, S_1, S_2, and S_3 and single input A

Block Diagram:

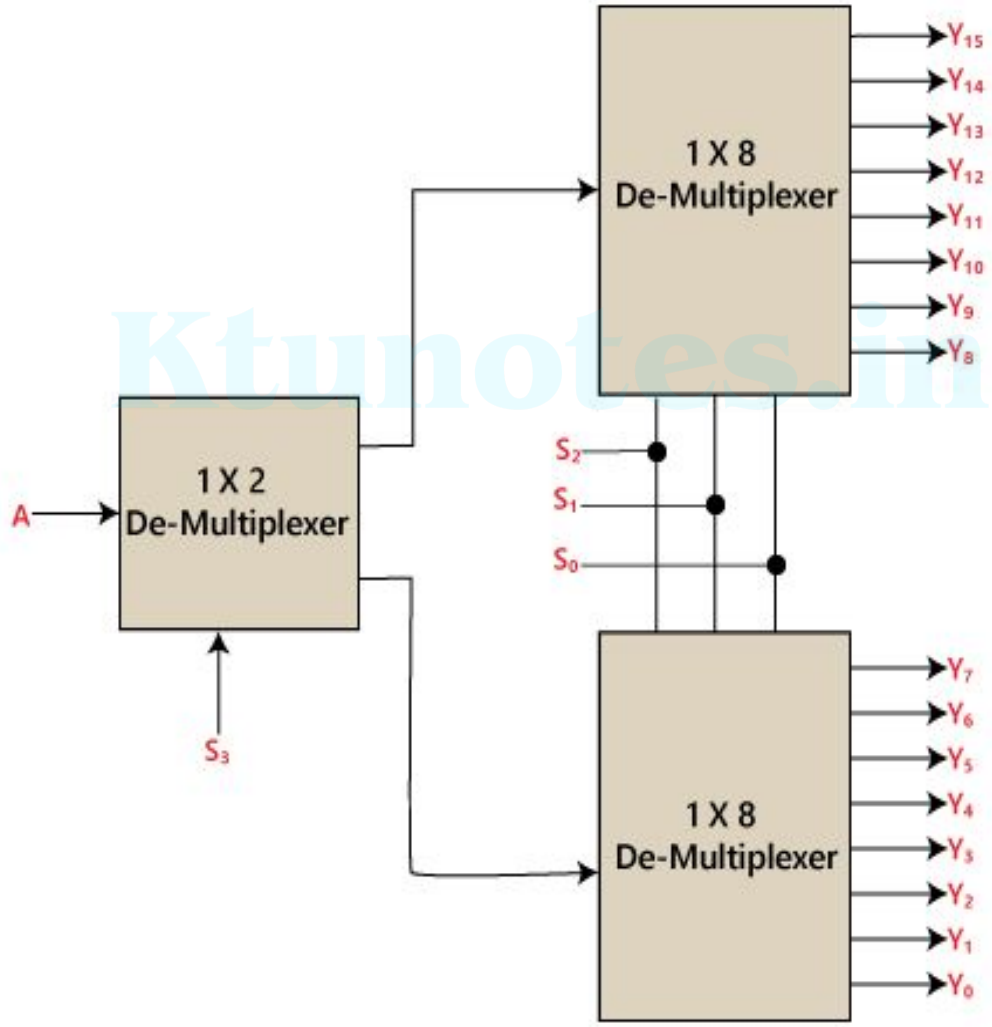


1 x 16 De-multiplexer

Truth Table:

INPUTS				OUTPUTS															
S_3	S_2	S_1	S_0	Y_{15}	Y_{14}	Y_{13}	Y_{12}	Y_{11}	Y_{10}	Y_9	Y_8	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	A	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	A	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	A	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	A	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	A	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	A	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	A	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	A	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	A	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	A	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	A	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	A	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1×16 de-multiplexer using 1×8 and 1×2 de-multiplexer



BCD Adder

- BCD-Adder is used in computers and calculators that perform arithmetic operation directly in decimal number system
- BCD-Adder accepts binary-coded form of decimal numbers
- If produced sum is between 0 to 9, Binary and BCD code is same
- But for 10 to 19 decimal numbers, both codes are different
- Binary sum combinations from 10 to 19 give invalid BCD

BCD Adder

Binary Sum						BCD Sum						Decimal
K	Z_8	Z_4	Z_2	Z_1		C	S_8	S_4	S_2	S_1		
0	0	0	0	0	S	0	0	0	0	0		0
0	0	0	0	1	A	0	0	0	0	1		1
0	0	0	1	0	M	0	0	0	1	0		2
.	.	.	.	.	E	.	.	.	.	.		.
.	.	.	.	.		.	.	.	.	.		.
.	.	.	.	.	C	.	.	.	.	.		.
.	.	.	.	.	O	.	.	.	.	.		.
0	1	0	0	0	D	0	1	0	0	0		8
0	1	0	0	1	E	0	1	0	0	1		9
10 to 19 Binary and BCD codes are not the same												
0	1	0	1	0		1	0	0	0	0		10
0	1	0	1	1		1	0	0	0	1		11
0	1	1	0	0		1	0	0	1	0		12
0	1	1	0	1		1	0	0	1	1		13
0	1	1	1	0		1	0	1	0	0		14
0	1	1	1	1		1	0	1	0	1		15
1	0	0	0	0		1	0	1	1	0		16
1	0	0	0	1		1	0	1	1	1		17
1	0	0	1	0		1	1	0	0	0		18
1	0	0	1	1		1	1	0	0	1		19

BCD Adder

1. A correction is needed when **'Binary Sum'** has an output carry K=1
2. Other six combinations from 10 to 15 need correction in which bit on Z_8 position is 1
3. In Binary sum of 8 and 9, bit on Z_8 position is also 1. So, second step fails, and we need to modify it
4. To distinguish these two numbers, we specify that bit on Z_4 or Z_2 position also needs to be 1 with bit of Z_8

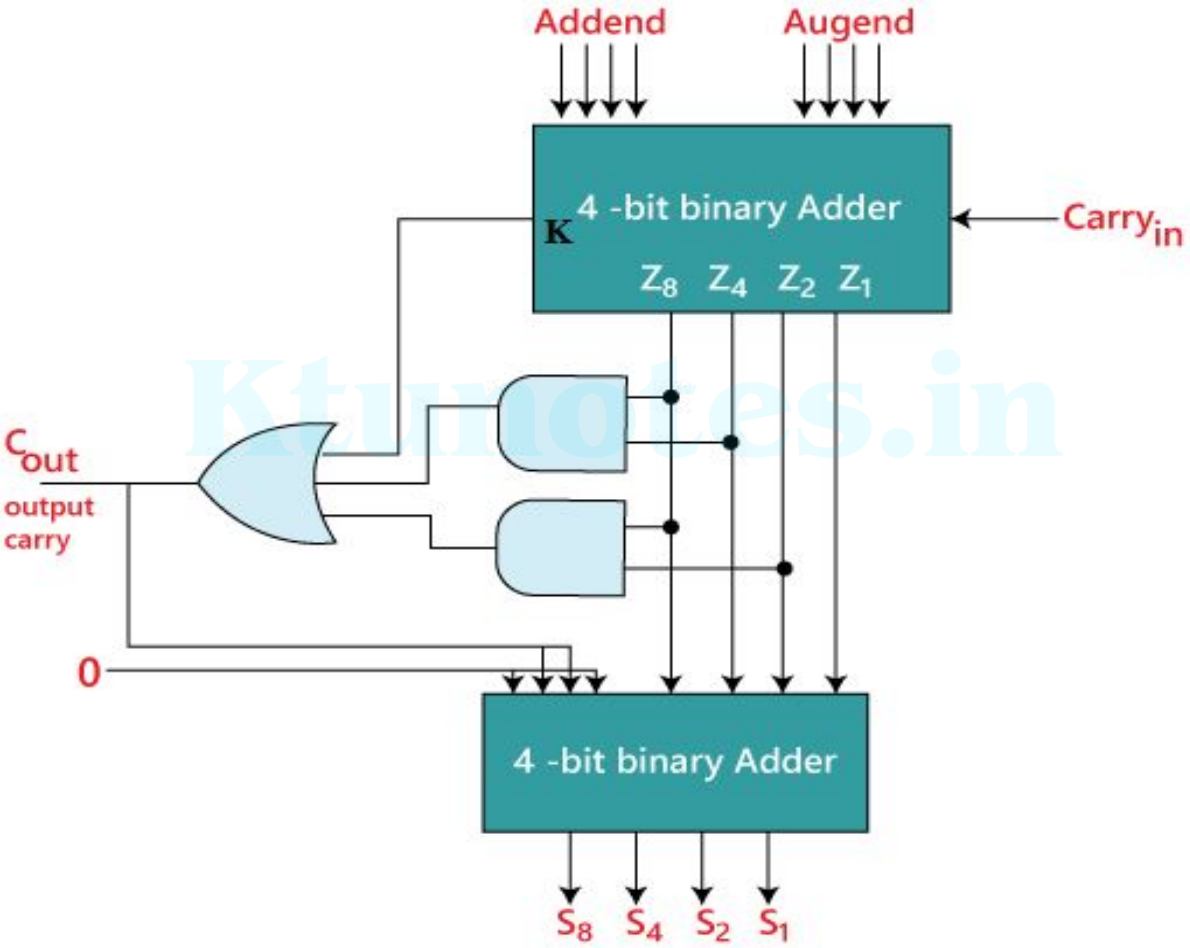
BCD Adder

- Condition for a correction and an output carry can be expressed by Boolean function:

  $\mathbf{C=K+Z_8.Z_4+Z_8.Z_2}$

- Once circuit found invalid BCD, circuit adds binary number of 6 into invalid BCD code to make it valid

BCD Adder



Black diagram of a BCD adder

BCD Adder

- We take a 4-bit Binary-Adder, which takes addend and augend bits as an input with an input carry **'Carry in'**
- Binary-Adder produces five outputs, i.e., Z8, Z4, Z2, Z1, and an output carry K
- With help of output carry K and Z8, Z4, Z2, Z1 outputs, logical circuit is designed to identify C_{out}
- Z8, Z4, Z2, and Z1 outputs of binary adder are passed into 2^{nd} 4-bit binary adder as an Augend.

BCD Adder

- Addend bit of 2^{nd} 4-bit binary adder is designed in such a way that 1^{st} and the 4^{th} bit of addend number are 0 and 2^{nd} and the 3^{rd} bit are same as C_{out}
- When value of C_{out} is 0, addend number will be 0000, which produce the same result as 1^{st} 4-bit binary number
- But when value of C_{out} is 1, addend bit will be 0110, i.e., 6, which adds with augent to get valid BCD number

BCD Adder

Example: 1001+1000

- First, add both numbers using a 4-bit binary adder and pass input carry to 0
- Binary adder produced result 0001 and carried output 'K' 1
- Then, find C_{out} value to identify that the produced BCD is invalid or valid using expression $C_{out}=K+Z_8.Z_4+Z_8.Z_2$.

  $K = 1 \quad Z_8 = 0 \quad Z_4 = 0 \quad Z_2 = 0$

  $C_{out} = 1+0*0+0*0$

  $C_{out} = 1+0+0$

  $C_{out} = 1$

BCD Adder

- Value of C_{out} is 1, which expresses that produced BCD code is invalid. Then, add output of 1^{st} 4-bit binary adder with 0110.
  = 0001+0110
  = 0111
- BCD is represented by carry output as:
  BCD = $C_{out}\ S_8\ S_4\ S_2\ S_1$=1 0 1 1 1