

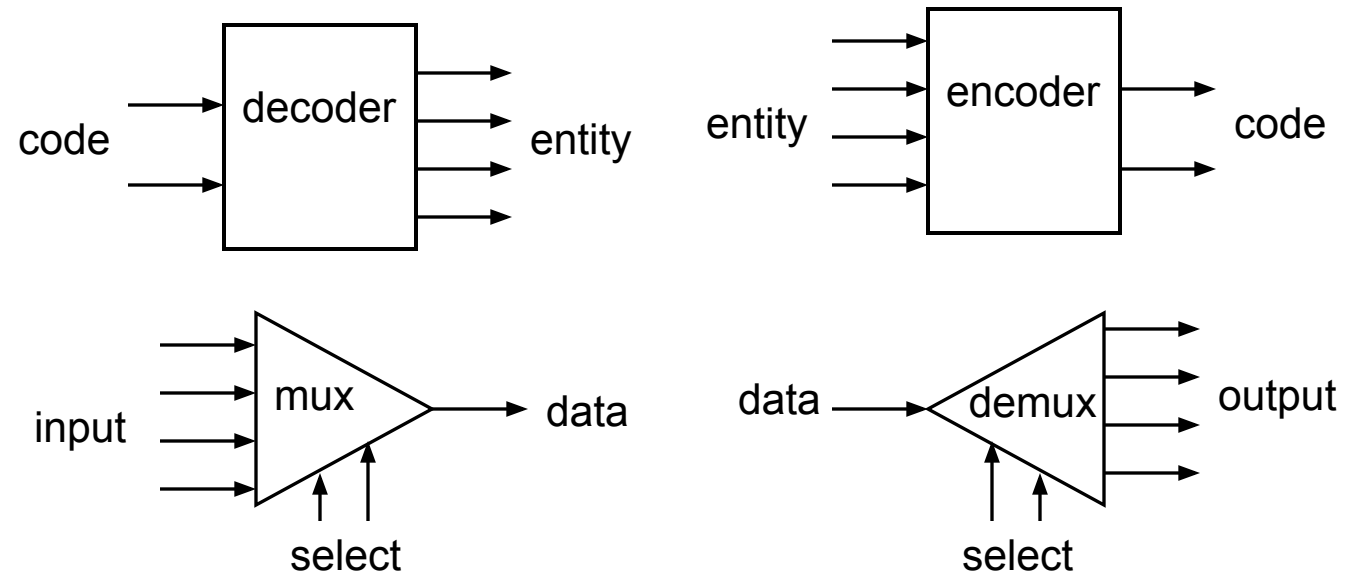
# MSI Circuits

# Useful MSI circuits

- Four common and useful MSI circuits are:

- ❖ Decoder
- ❖ Demultiplexer
- ❖ Encoder
- ❖ Multiplexer

- Block-level outlines of MSI circuits:

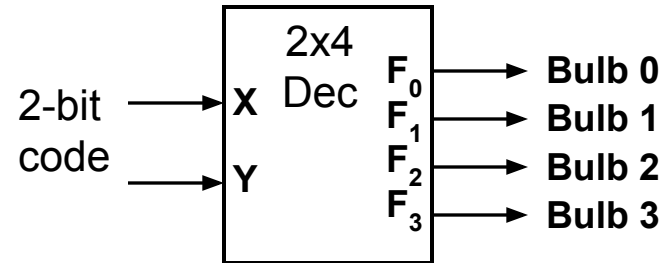


# Decoders

- Convert binary information from  $n$  input lines to (max. of)  $2^n$  output lines.
- Known as  $n$ -to- $m$ -line decoder, or simply  $n:m$  or  $n \times m$  decoder ( $m \leq 2^n$ ).
- May be used to generate  $2^n$  (or fewer) minterms of  $n$  input variables.

# Decoders

- Example: if codes 00, 01, 10, 11 are used to identify four light bulbs, we may use a 2-bit decoder:



- This is a 2×4 decoder which selects an output line based on the 2-bit code supplied.

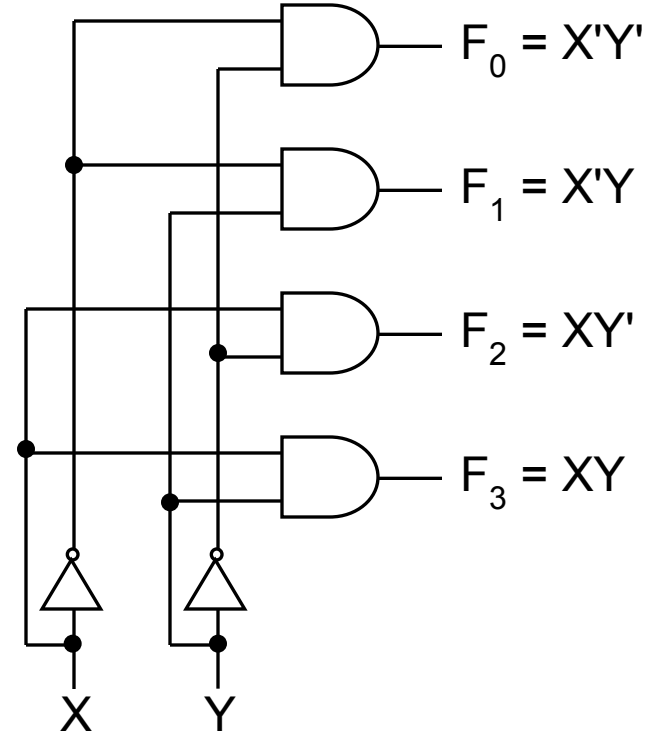
- Truth table:

X	Y	F <sub>0</sub>	F <sub>1</sub>	F <sub>2</sub>	F <sub>3</sub>
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

# Decoders

- From truth table, circuit for 2×4 decoder is:
- Note: Each output is a 2-variable minterm ( $X'Y'$ ,  $X'Y$ ,  $XY'$  or  $XY$ )

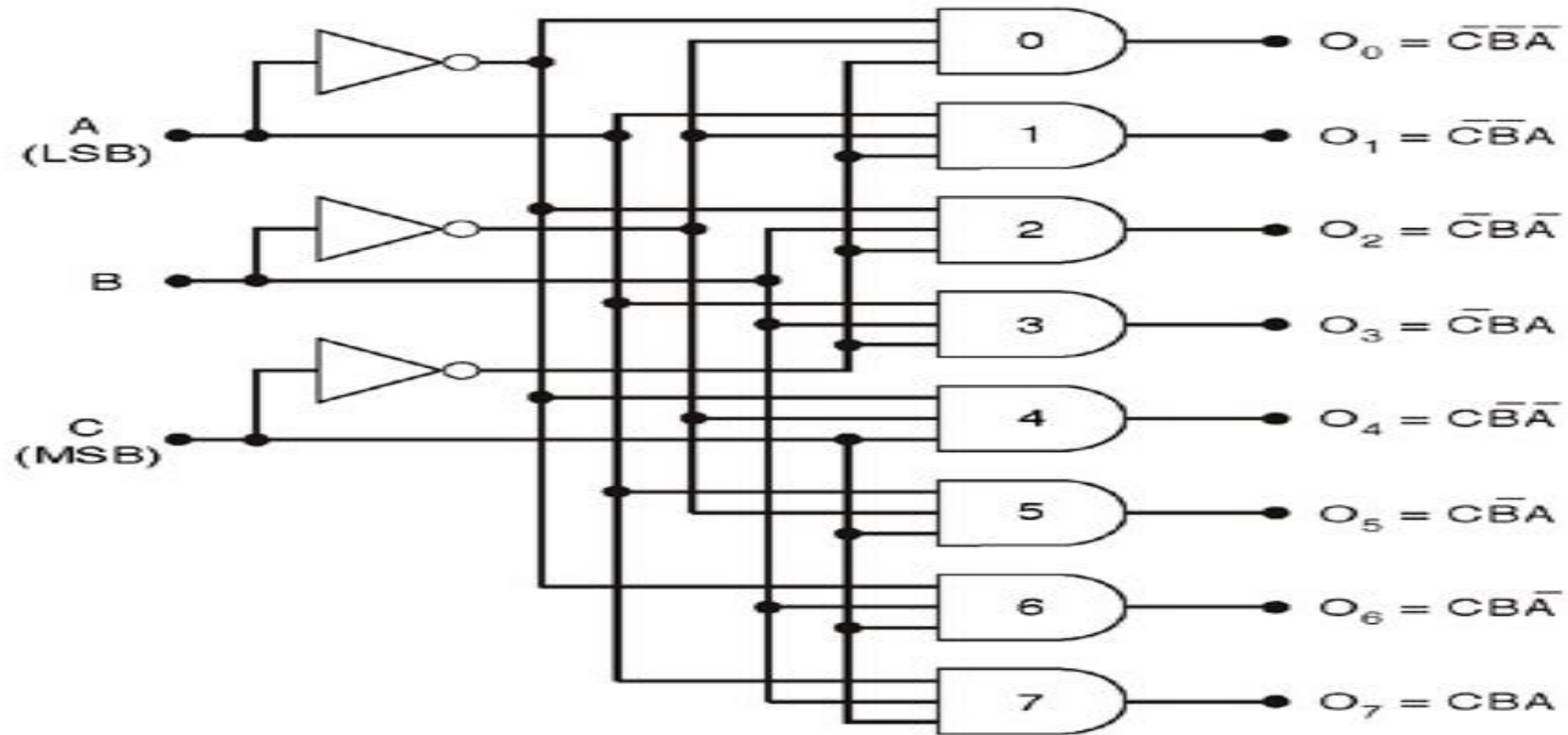
X	Y	$F_0$	$F_1$	$F_2$	$F_3$
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



# Decoders

- Design a 3×8 decoder **by yourself**.

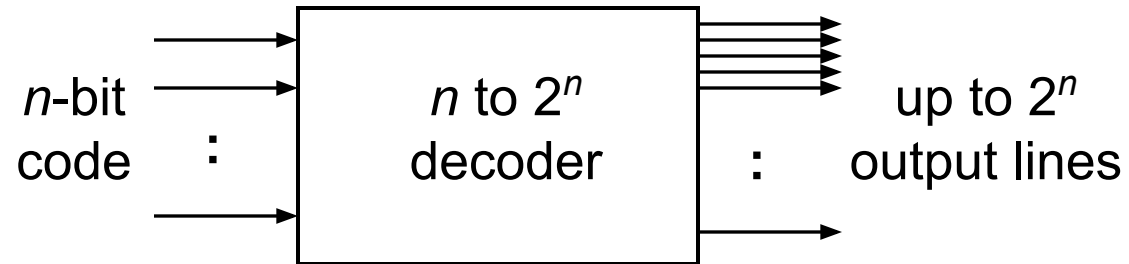
# Solution



C	B	A	O <sub>7</sub>	O <sub>6</sub>	O <sub>5</sub>	O <sub>4</sub>	O <sub>3</sub>	O <sub>2</sub>	O <sub>1</sub>	O <sub>0</sub>
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

# Decoders

- In general, for an  $n$ -bit code, a decoder could select up to  $2^n$  lines:

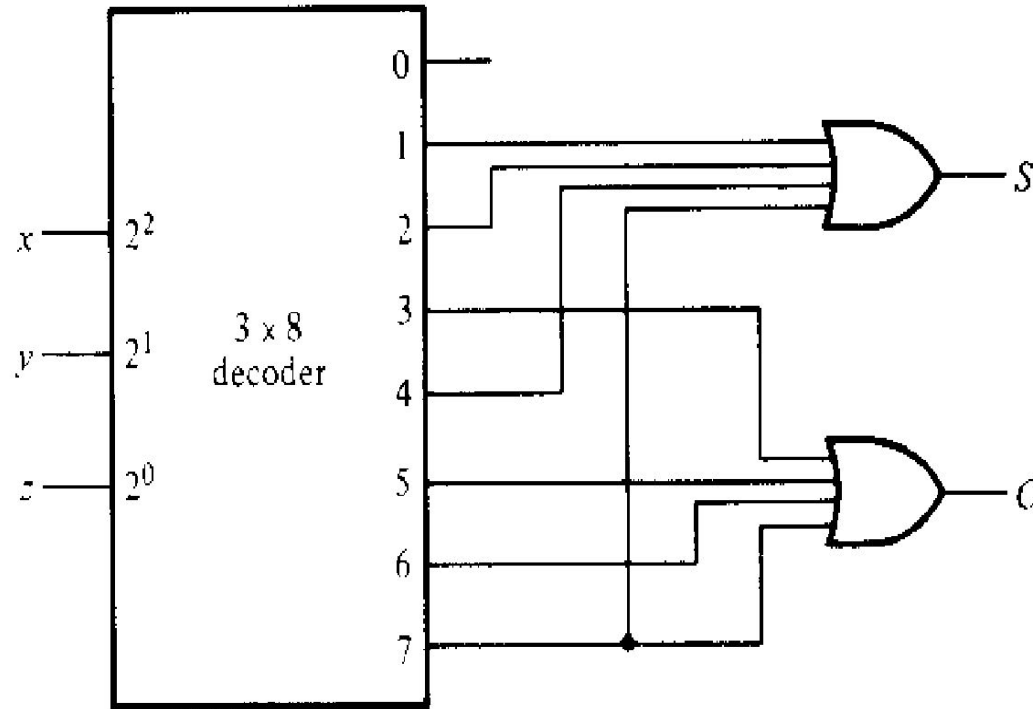


As  $n$  input generates  $2^n$  output, which reminds us of canonical SOP, thus a decoder can be used to generate any function



# Application of Decoder

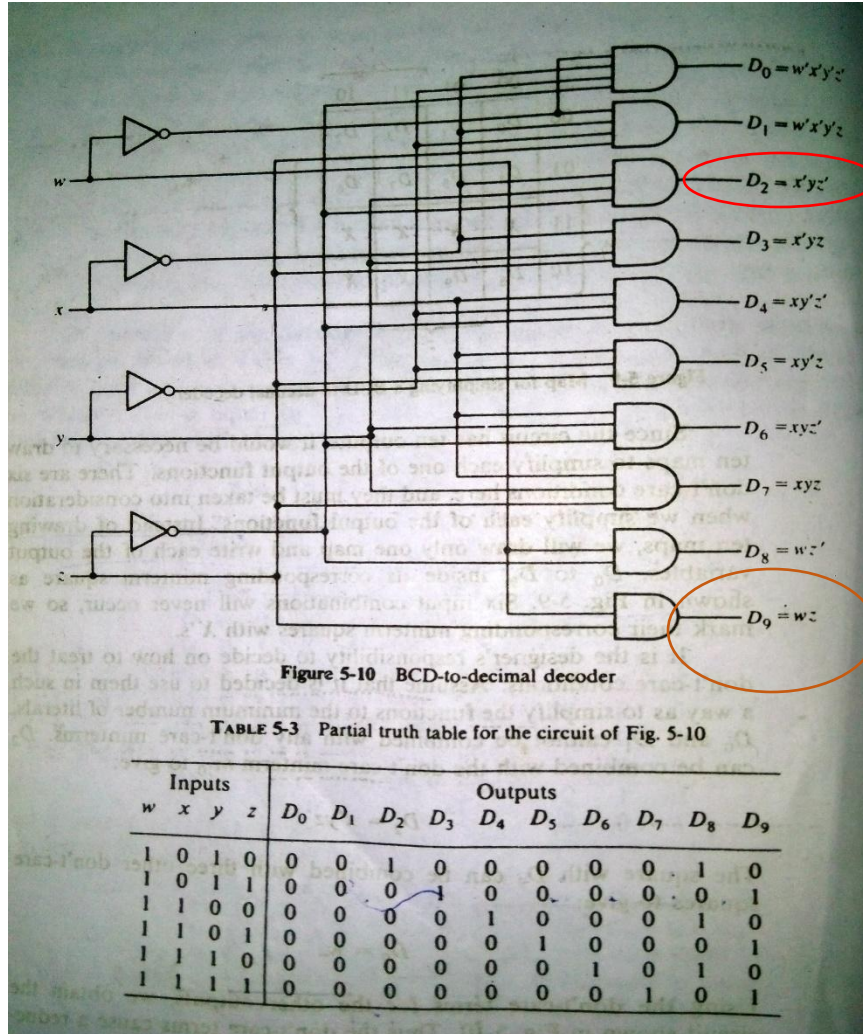
- Example 1: Full adder circuit with decoder (3 x 8 decoder)



$$S(x, y, z) = \Sigma(1, 2, 4, 7)$$

$$C(x, y, z) = \Sigma(3, 5, 6, 7)$$

# Application of Decoder

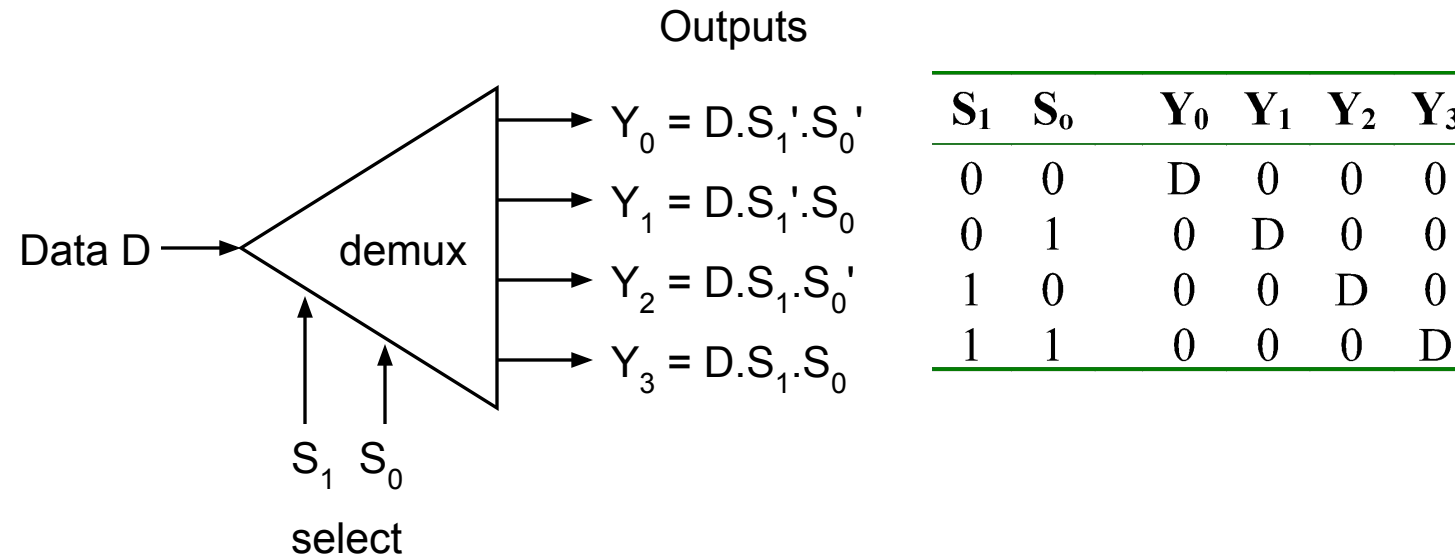


- Example 2: BCD to Decimal decoder: 4 input-10 output . We can use don't cares for simplification.

		yz			
		00	01	11	10
wx	00	D <sub>0</sub>	D <sub>1</sub>	D <sub>3</sub>	D <sub>2</sub>
	01	D <sub>4</sub>	D <sub>5</sub>	D <sub>7</sub>	D <sub>6</sub>
	11	X	X	X	X
	10	D <sub>8</sub>	D <sub>9</sub>	X	X

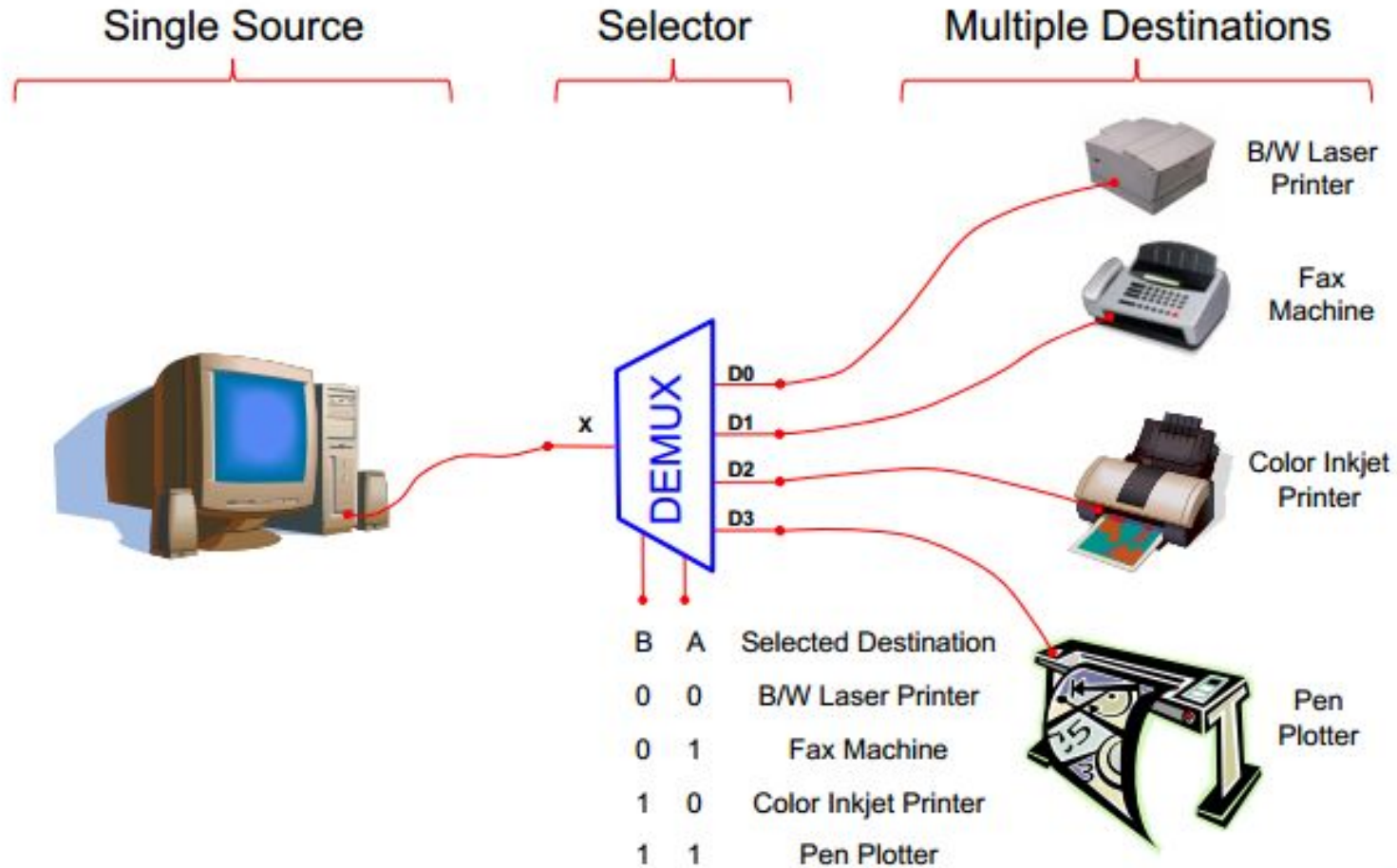
# Demultiplexer

- Given an input line and a set of selection lines, the demultiplexer will direct data from input to a **selected output line**.
- An example of a 1-to-4 demultiplexer:



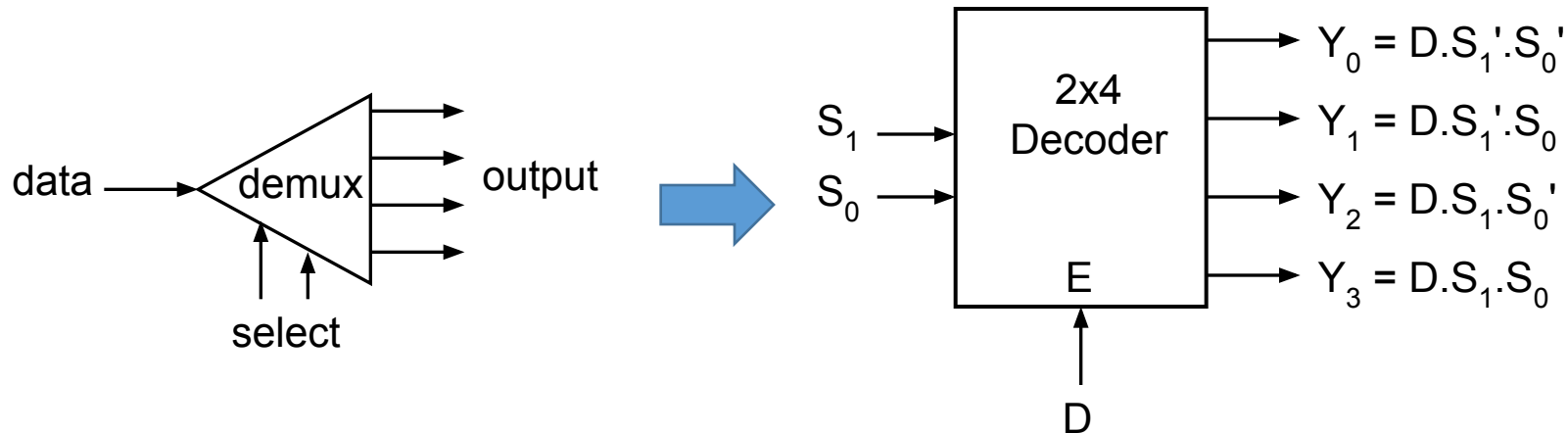


# Typical Application of a DEMUX



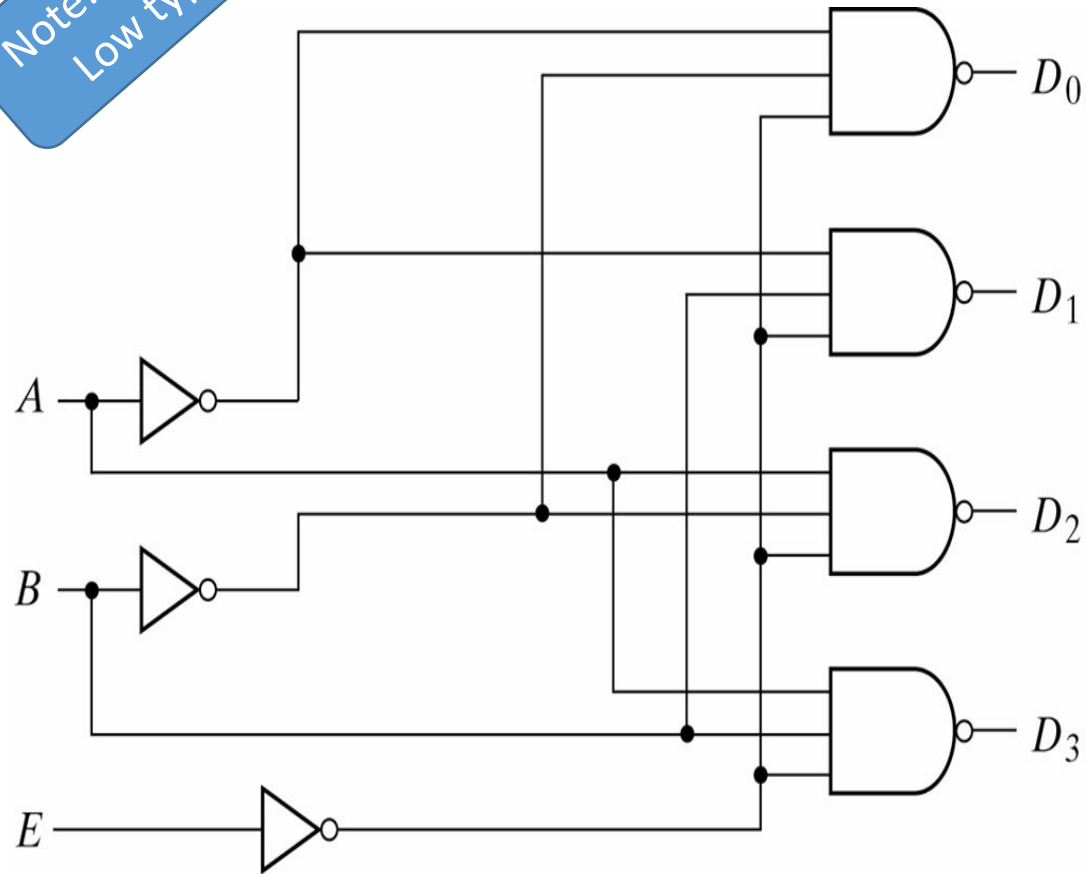
# Demultiplexer

- The demultiplexer is actually identical to a decoder with enable. A **decoder with an enable** input can function as a demultiplexer (or demux)  
Decoder + enable = demultiplexer
- The selection of a specific output line is controlled by the bit values of 'n'-selection lines.



**2-to-4 decoder with Enable input**  
*constructed with NAND and Not gate*

Note: Known as Active Low type decoder



(a) Logic diagram

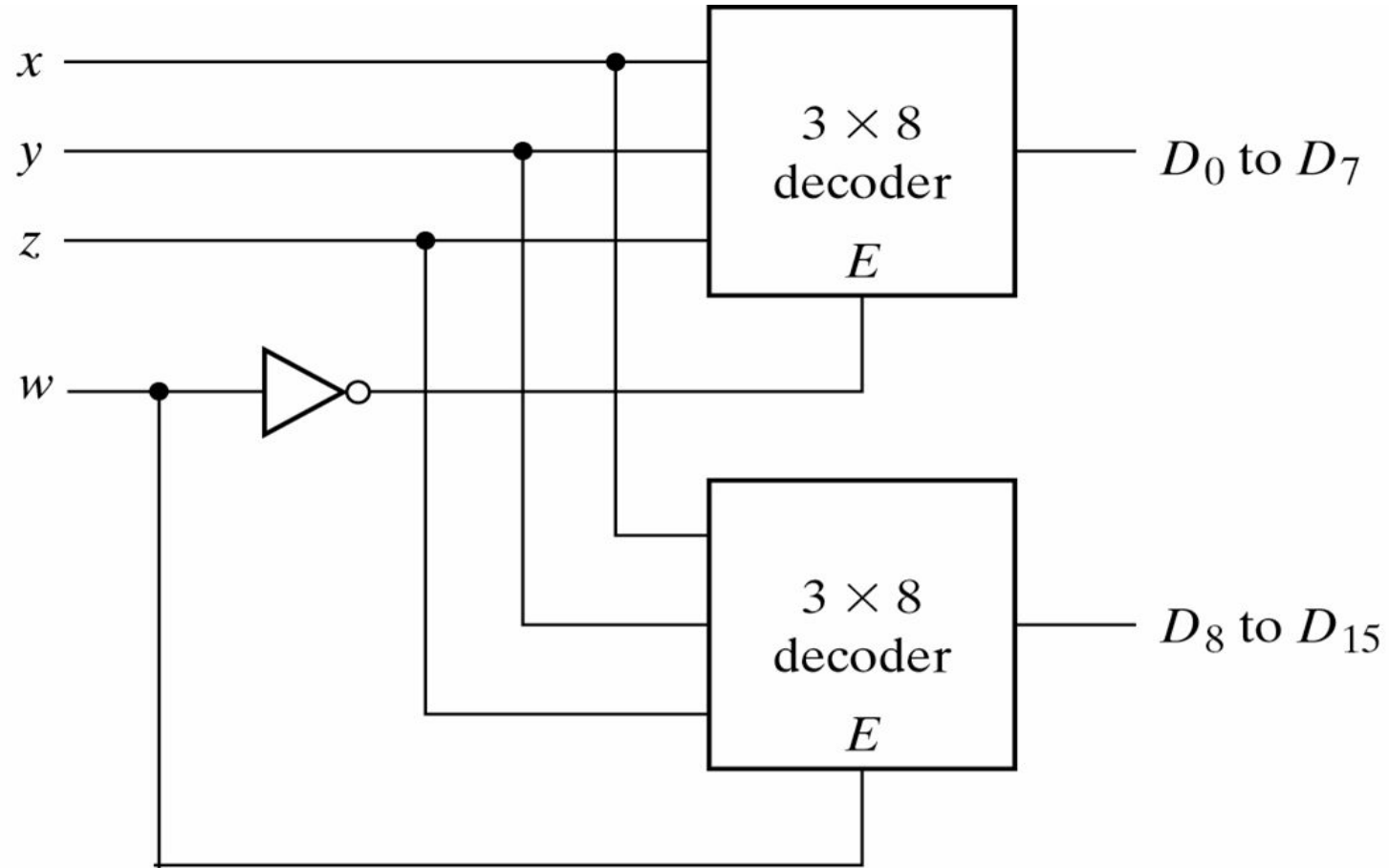
$E$	$A$	$B$	$D_0$	$D_1$	$D_2$	$D_3$
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

At a time only 1 output is low

If enable ( E ) is 0, circuit works as decoder.

(b) Truth table

## 4-line-to-16 line Decoder constructed with two 3-line-to-8 line decoders with enables





## 4-line-to-16 line Decoder constructed with two 3-line-to-8 line decoders with enables

w	x	y	z
0	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	0	0	0
1	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

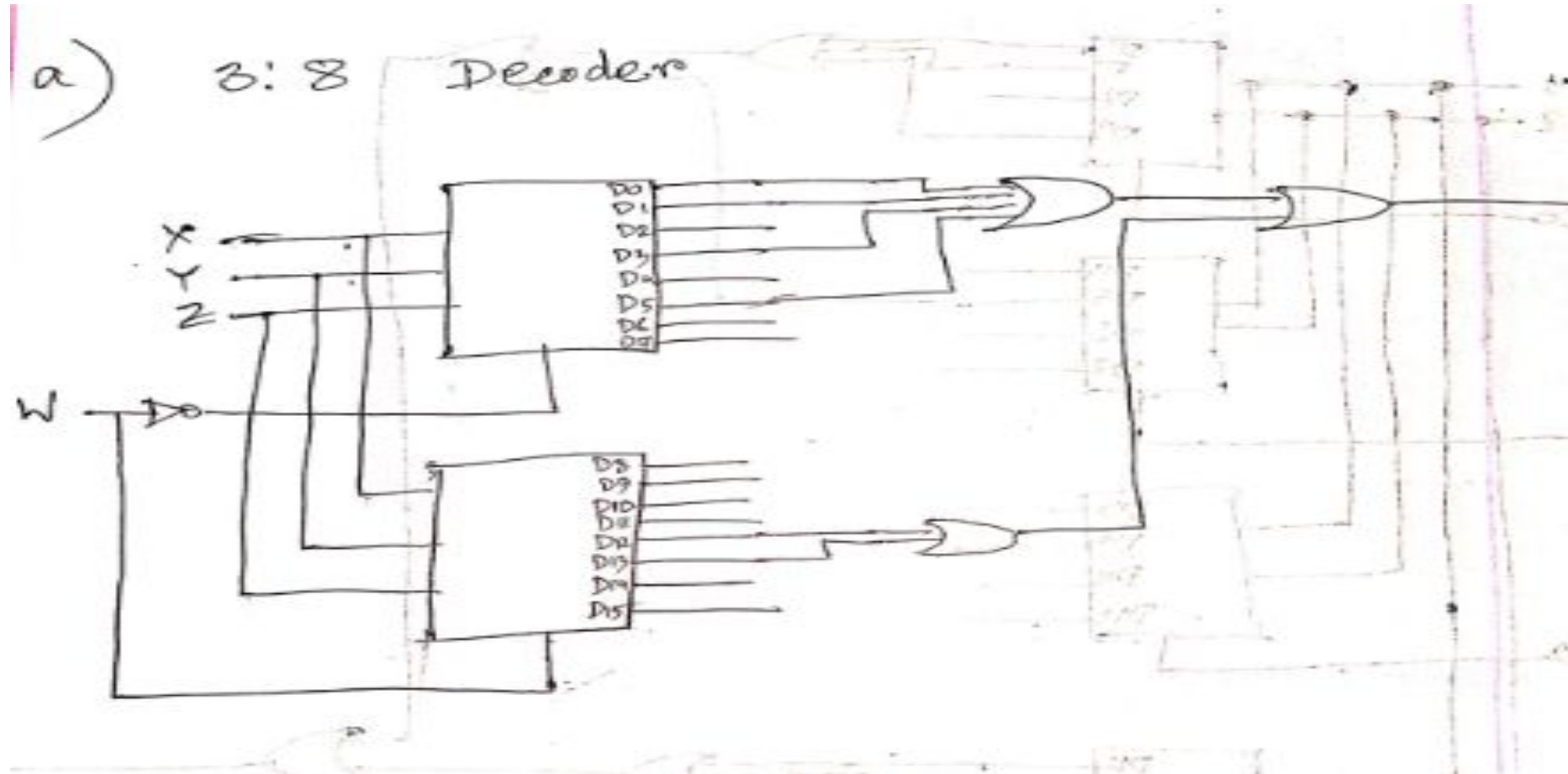
- When  $w=0$ , the top decoder is enabled and the other is disabled. The bottom decoder outputs are all 0's, and the top eight outputs generate min-terms 0000 to 0111.
- When  $w=1$ , the enable conditions are reversed. The bottom decoder outputs generate min-terms 1000 to 1111, while the outputs of the top decoder are all 0's.



$$F(a,b,c,d) = \sum(0,1,3,5,12,13)$$

Implement the above boolean function using

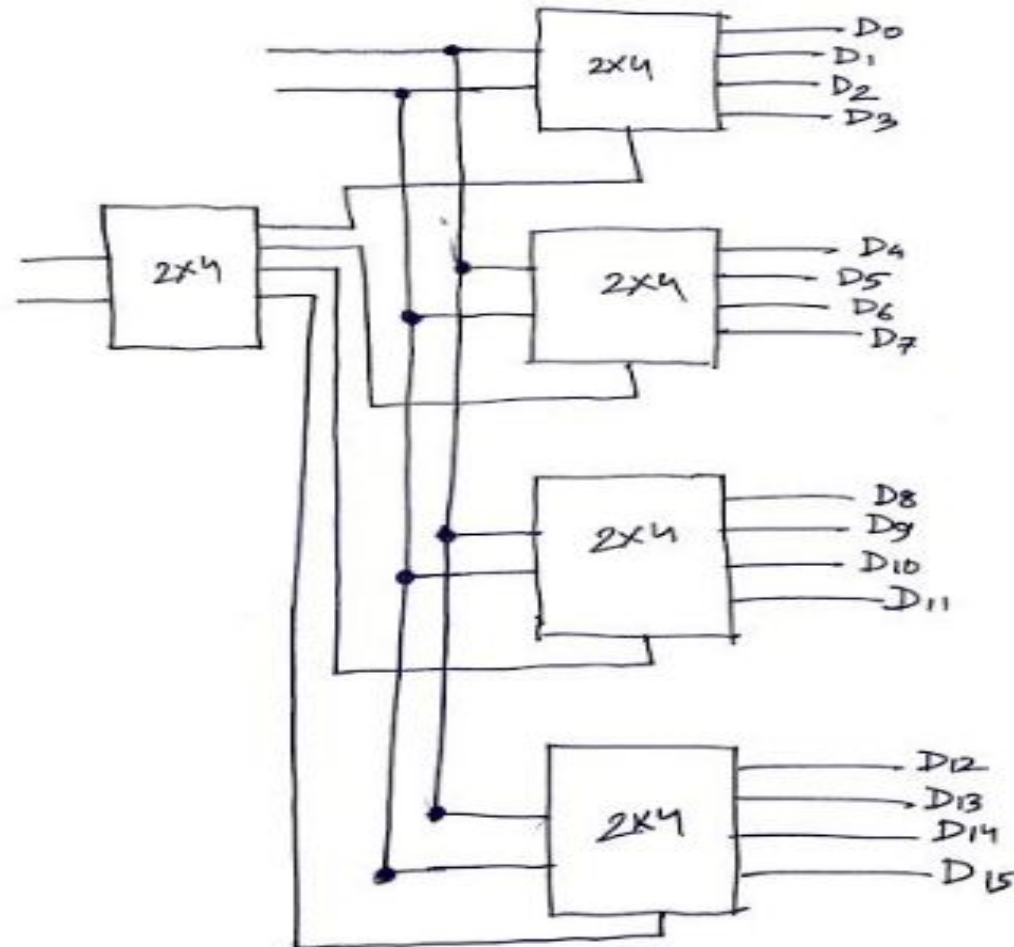
- 3:8 Decoder(s).
- 2:4 Decoder(s).



$$F(a,b,c,d)=\sum(0,1,3,5,12,13)$$

Implement the above boolean function using

- 3:8 Decoder(s).
- 2:4 Decoder(s).



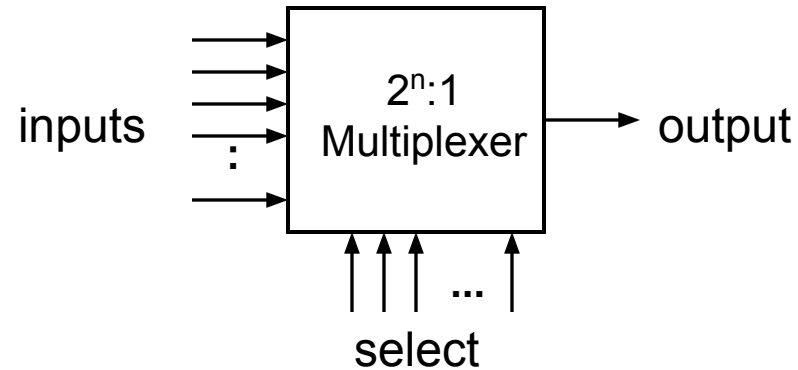
You have to connect D0,D1,D3,D5,D12 & D13 with OR Gate

# HomeWork

- Design a BCD to 7 segment display using a 4x16 line decoder

# Multiplexer

- A multiplexer is a device which has
  - (i) a number of *input* lines
  - (ii) a number of *selection* lines
  - (iii) one *output* line
- A Multiplexer steers one of  $2^n$  inputs to a single output line, using  $n$  selection lines. Also known as a *data selector*.



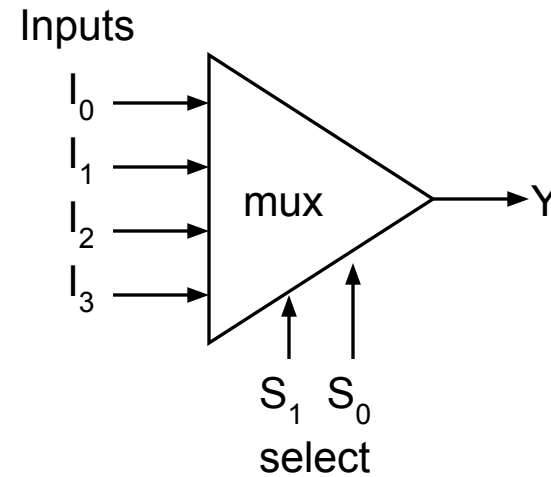
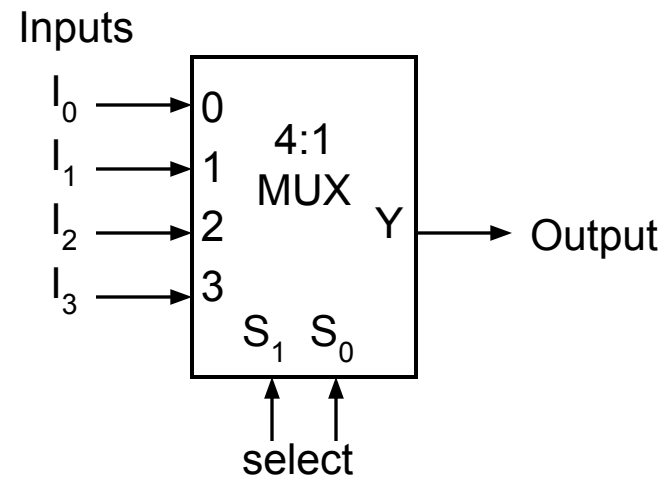
# Multiplexer

- Truth table for a 4-to-1 multiplexer:

$I_0$	$I_1$	$I_2$	$I_3$	$S_1$	$S_0$	$Y$
$d_0$	$d_1$	$d_2$	$d_3$	0	0	$d_0$
$d_0$	$d_1$	$d_2$	$d_3$	0	1	$d_1$
$d_0$	$d_1$	$d_2$	$d_3$	1	0	$d_2$
$d_0$	$d_1$	$d_2$	$d_3$	1	1	$d_3$

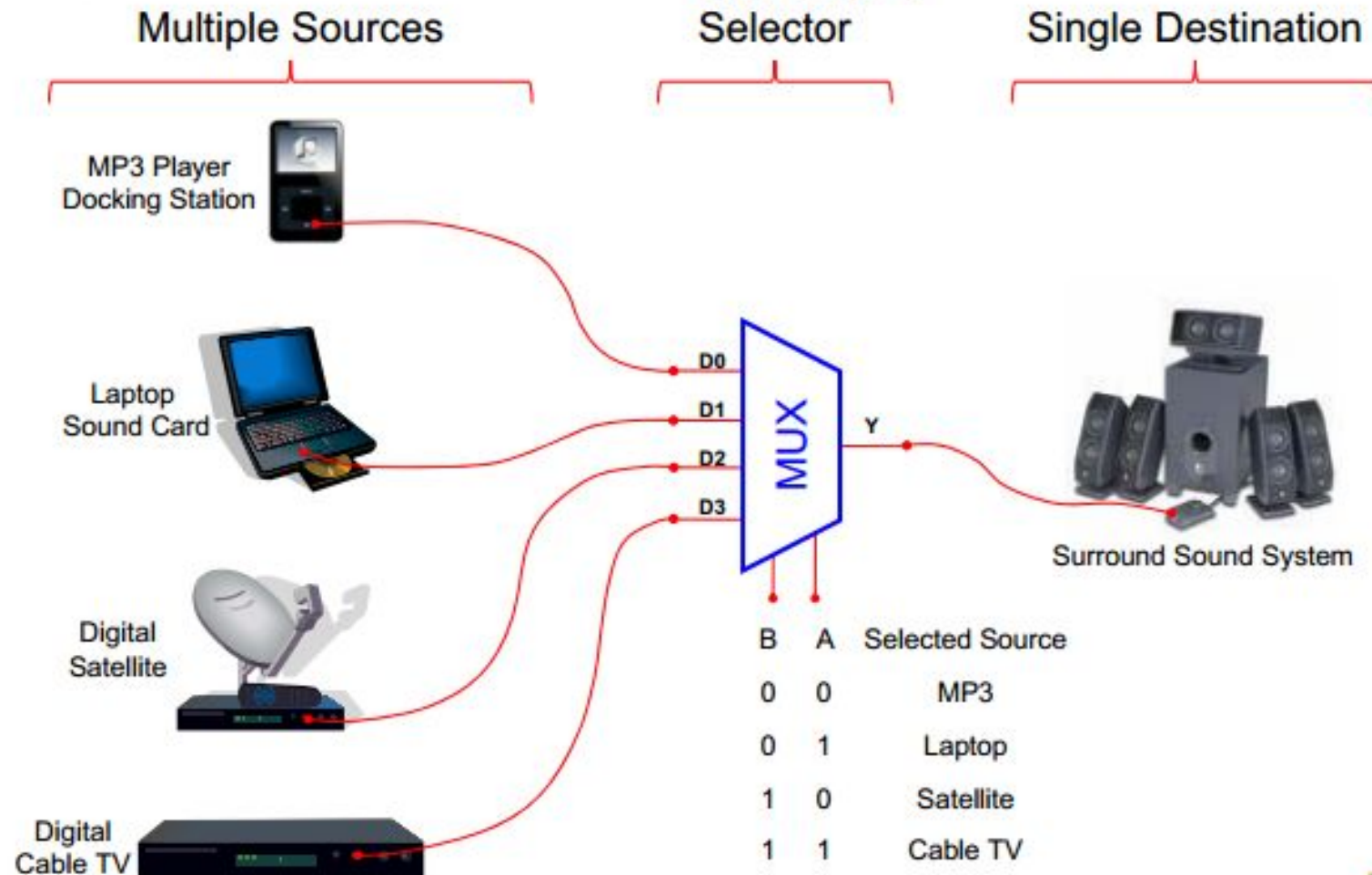


$S_1$	$S_0$	$Y$
0	0	$I_0$
0	1	$I_1$
1	0	$I_2$
1	1	$I_3$





# Typical Application of a MUX



# Multiplexer

- Output of multiplexer is  
“sum of the (product of *data lines* and *selection lines*)”
- Often known as Data selector as it selects one of the many inputs and steers the binary information to the output line.
- Example: the output of a 4-to-1 multiplexer is:

$$Y = I_0.(S_1'.S_0') + I_1.(S_1'.S_0) + I_2.(S_1.S_0') + I_3.(S_1.S_0)$$

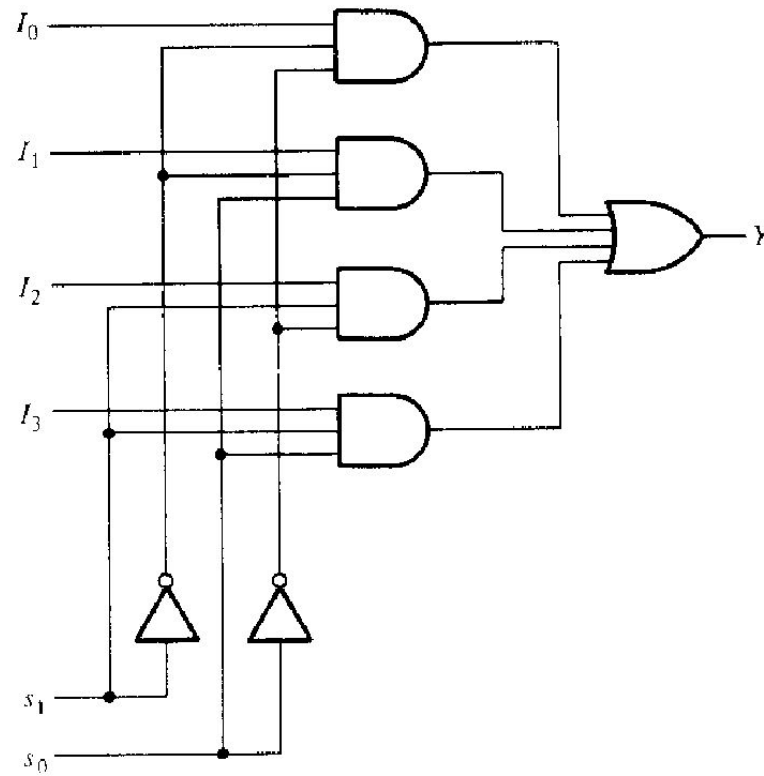
# Try it yourself

- Draw the internal circuit diagram (logic diagram) of a 4-to-1 multiplexer.



# Solution

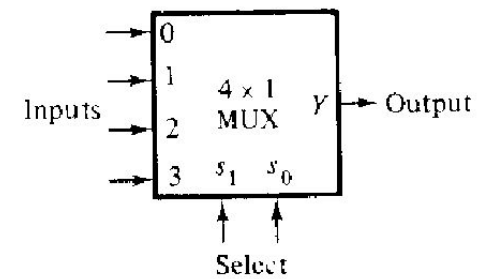
$$Y = I_0.(S_1'.S_0') + I_1.(S_1'.S_0) + I_2.(S_1.S_0') + I_3.(S_1.S_0)$$



(a) Logic diagram

$s_1$	$s_0$	$Y$
0	0	$I_0$
0	1	$I_1$
1	0	$I_2$
1	1	$I_3$

(b) Function table



(c) Block diagram

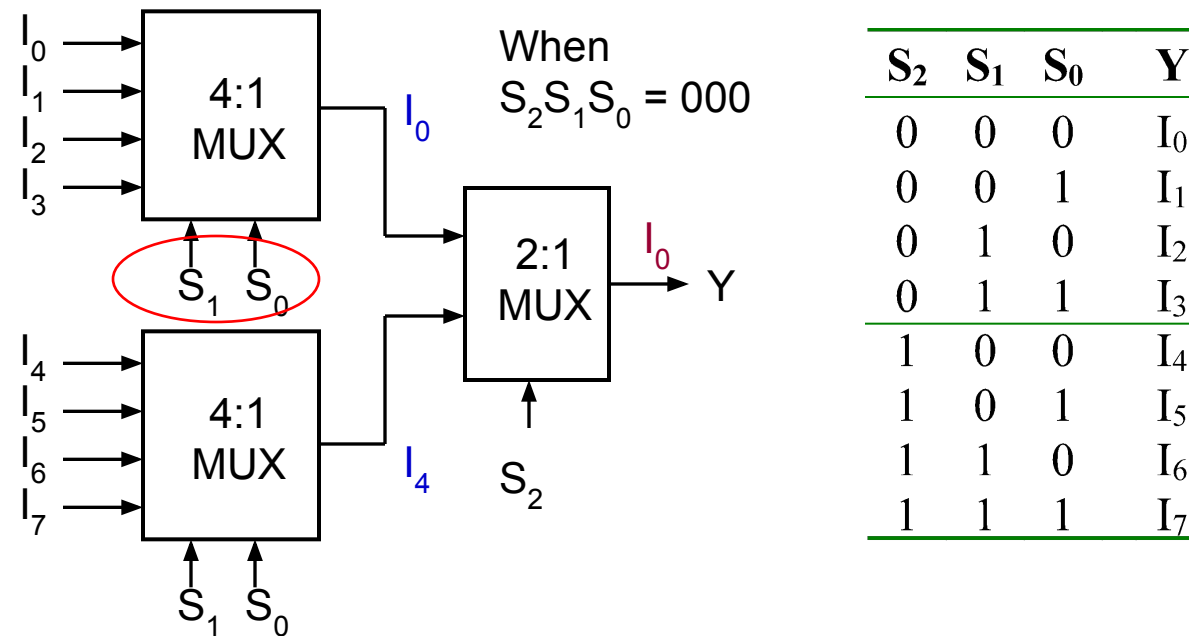
# Larger Multiplexers

- Larger multiplexers can be constructed from smaller ones.
- An 8-to-1 multiplexer can be constructed from smaller multiplexers like this (from two 4x1 and one 2x1):

S <sub>2</sub>	S <sub>1</sub>	S <sub>0</sub>	Y
0	0	0	I <sub>0</sub>
0	0	1	I <sub>1</sub>
0	1	0	I <sub>2</sub>
0	1	1	I <sub>3</sub>
1	0	0	I <sub>4</sub>
1	0	1	I <sub>5</sub>
1	1	0	I <sub>6</sub>
1	1	1	I <sub>7</sub>

# Larger Multiplexers

- Larger multiplexers can be constructed from smaller ones.
- An 8-to-1 multiplexer can be constructed from smaller multiplexers like this (from two 4x1 and one 2x1):



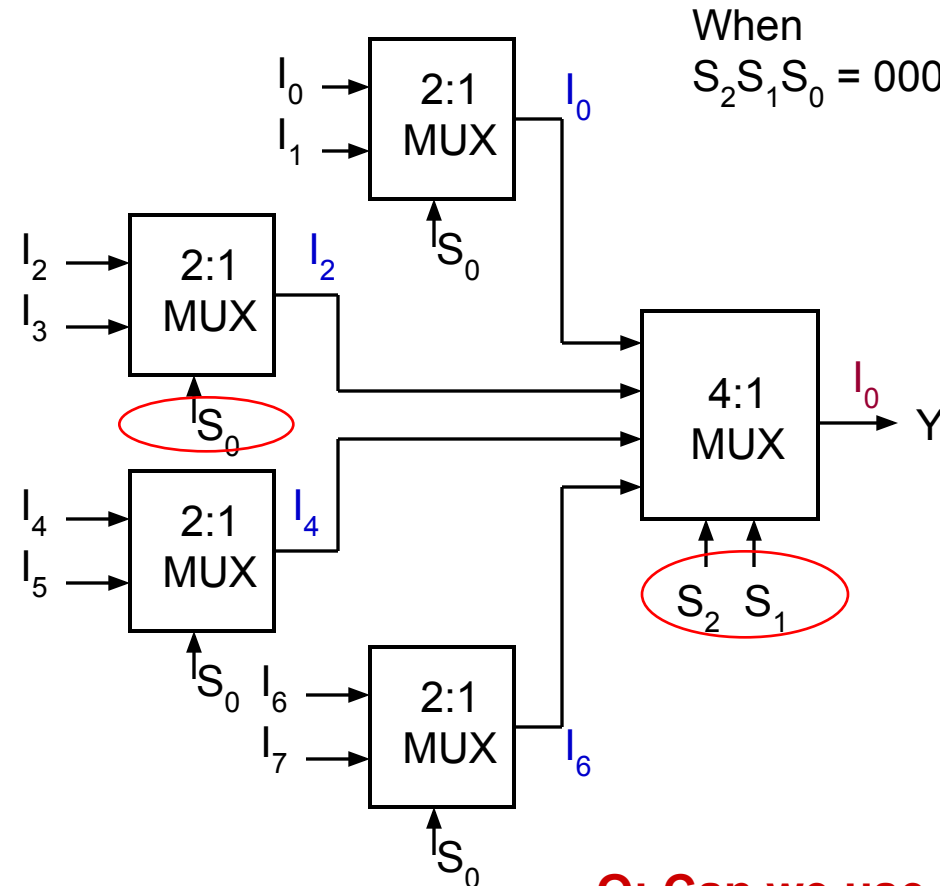
# Larger Multiplexers

- Another implementation of an 8-to-1 multiplexer using smaller multiplexers (four 2x1 and one 4x1):

<b>S<sub>2</sub></b>	<b>S<sub>1</sub></b>	<b>S<sub>0</sub></b>	<b>Y</b>
0	0	0	I <sub>0</sub>
0	0	1	I <sub>1</sub>
0	1	0	I <sub>2</sub>
0	1	1	I <sub>3</sub>
1	0	0	I <sub>4</sub>
1	0	1	I <sub>5</sub>
1	1	0	I <sub>6</sub>
1	1	1	I <sub>7</sub>

# Larger Multiplexers

- Another implementation of an 8-to-1 multiplexer using smaller multiplexers (four 2x1 and one 4x1):

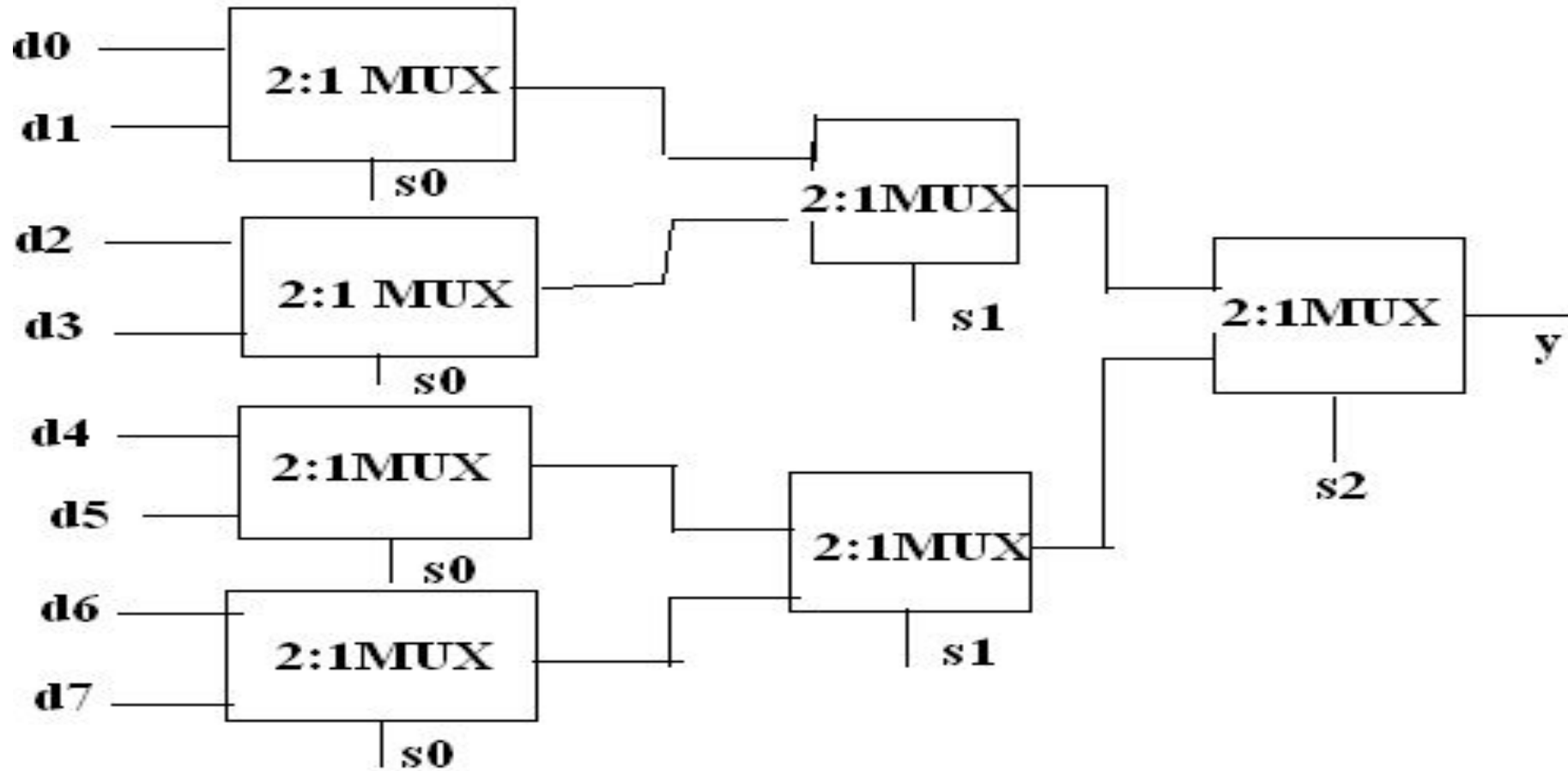


$S_2$	$S_1$	$S_0$	$Y$
0	0	0	$I_0$
0	0	1	$I_1$
0	1	0	$I_2$
0	1	1	$I_3$
1	0	0	$I_4$
1	0	1	$I_5$
1	1	0	$I_6$
1	1	1	$I_7$

**Q: Can we use only 2:1 multiplexers?**

# Larger Multiplexers

Q: Can we use only 2:1 multiplexers?

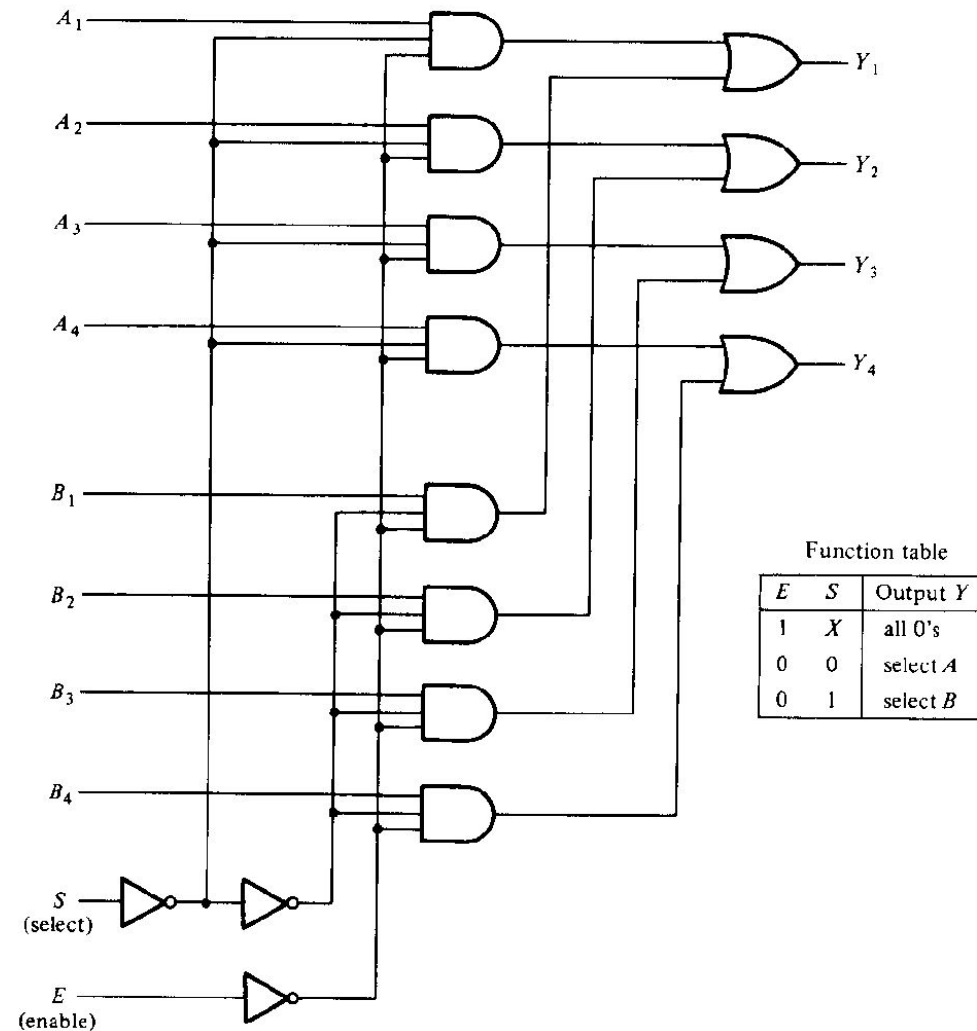


# Try it yourself: Larger Multiplexers

- A 16-to-1 multiplexer can be constructed from only 4-to-1 multiplexers:

# Multiplexer with enable input

We can construct it using four 2x1 line multiplexer for inserting the input and then three 2x1 line multiplexer for combining the result





# Encoder

- Encoder is a digital function that produces a reverse operation of a decoder!
- It has  $2^n$  input lines and n output lines

# Example: Octal-binary encoder

**TABLE 5-3**  
**Truth Table of Octal-to-Binary Encoder**

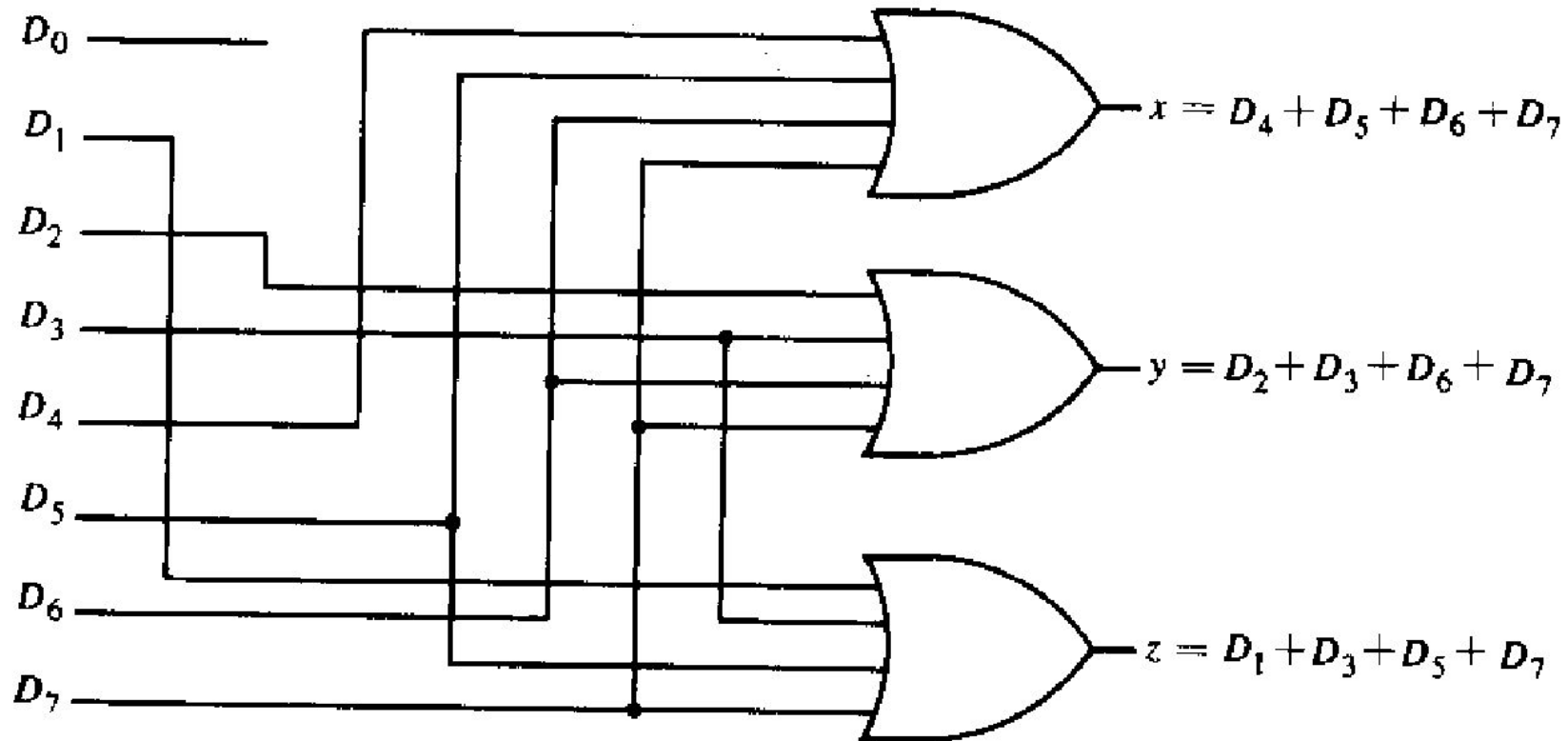
Inputs								Outputs		
$D_0$	$D_1$	$D_2$	$D_3$	$D_4$	$D_5$	$D_6$	$D_7$	$x$	$y$	$z$
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$$z = D_1 + D_3 + D_5 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$x = D_4 + D_5 + D_6 + D_7$$

# Example: Octal-binary encoder



**FIGURE 5-13**

Octal-to-binary encoder

There are 8 input variables so there will  $2^8$  input combination, amongst which in Octal-binary encoder only 8 are useful

# Example: Priority Encoder

- Design a priority encoder, which will allow more than one input to exist and encodes only the highest priority input line. For example, 8x3 encoder in a if user give D2 and D7 together, it will allow data of D7 to pass.

# Priority Encoder

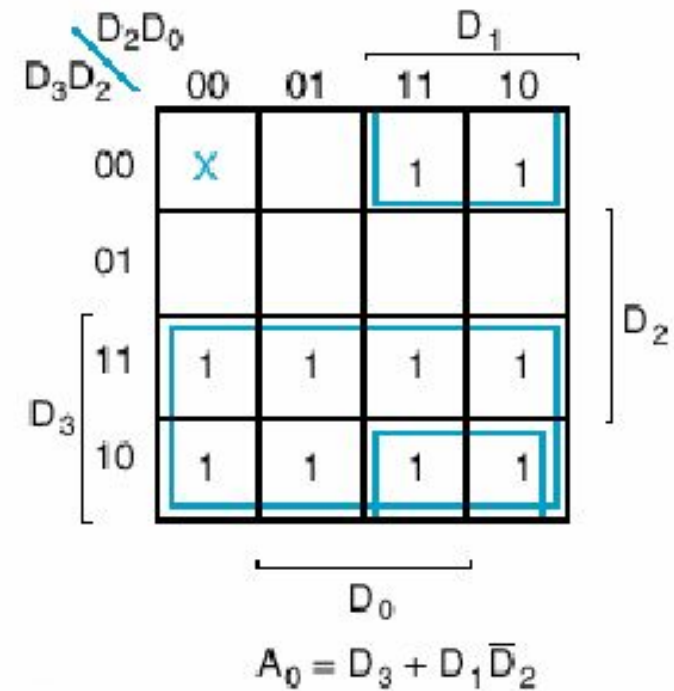
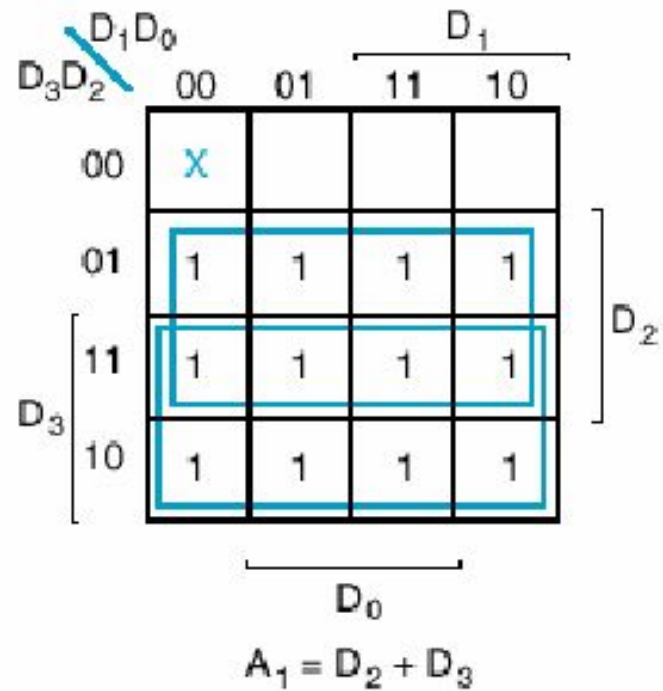
- Accepts multiple values and encodes them
  - Works when more than one input is active
- Consists of:
  - Inputs ( $2^n$ )
  - Outputs
    - when more than one output is active, sets output to correspond to highest input
  - V (indicates whether any of the inputs are active). This helps to show the output when all inputs are 0s
- Selectors / Enable

D3	D2	D1	D0	A1	A0	V
0	0	0	0	x	X	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	0	1	1	0	1	1
0	1	0	0	1	0	1
0	1	0	1	1	0	1
0	1	1	0	1	0	1
0	1	1	1	1	0	1
1	0	0	0	1	1	1
1	0	0	1	1	1	1
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

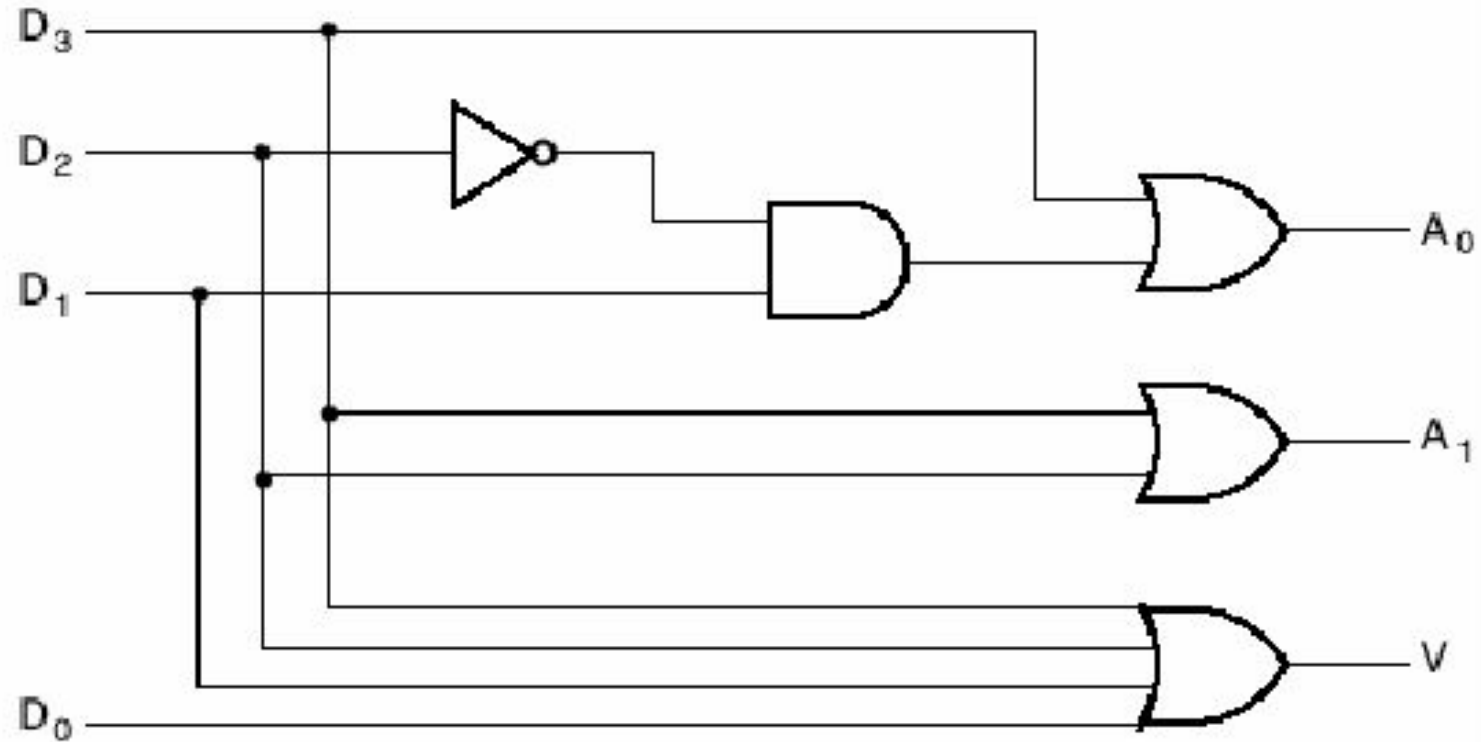
Note: Amongst all 3 input, D2 is highest so output reflects the binary value of 2

Note: V is not really an output, just shows whether there is an active input or not.

# Priority Encoder



# Priority Encoder





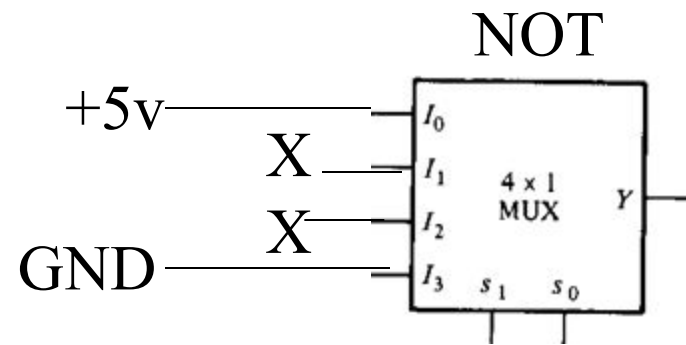
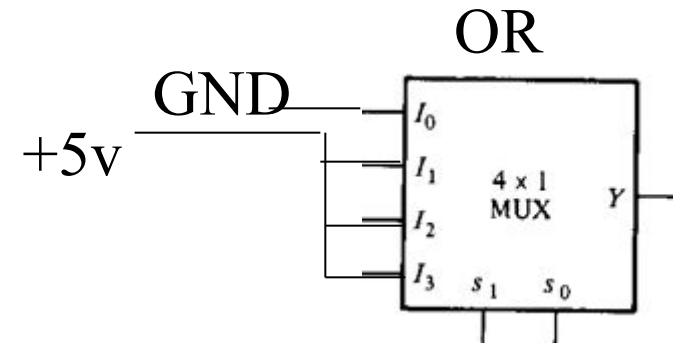
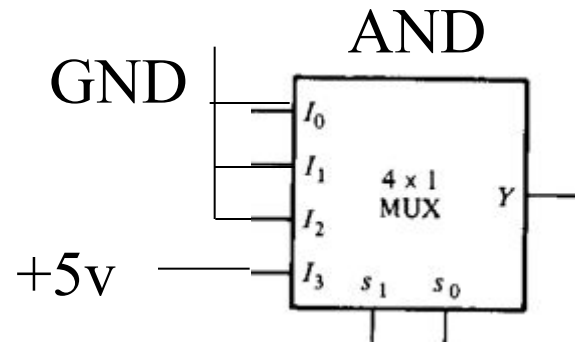
# Boolean Function implementation using MSI

# Try it yourself

Using 4:1 MUX design

- AND gate
- OR gate
- NOT gate

# Solution



# Try it yourself

a) Built the following function using 8x1 Mux.

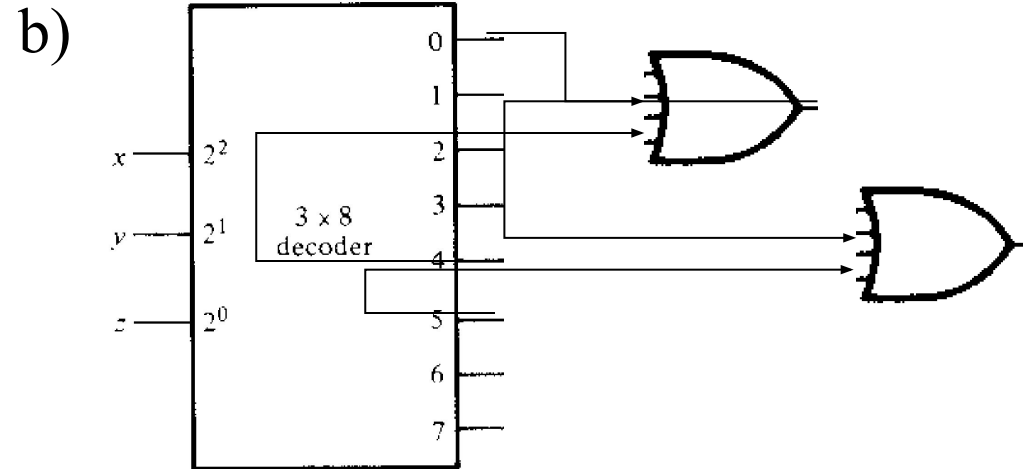
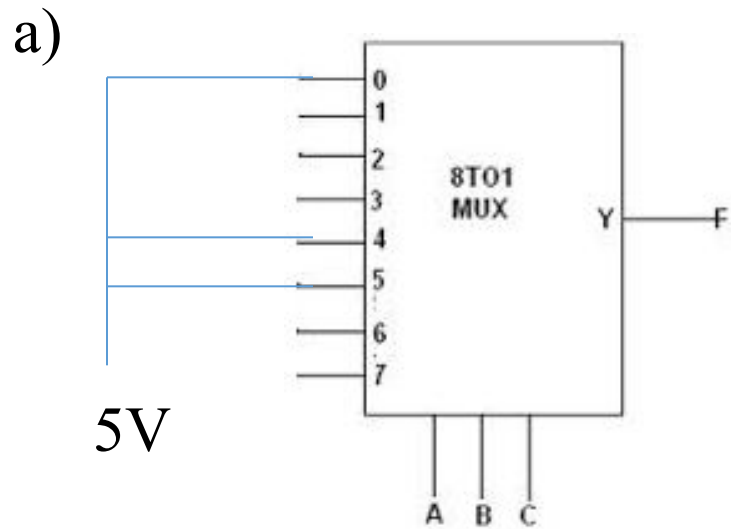
$$F = \sum(0, 4, 5)$$

b) Design same thing with 3x8 decoder

c) Try designing it with single 4x1 Mux

# Solution

$$F = \sum(0, 4, 5)$$



# Rules: for using smaller mux to build larger equation

If the two minterms in a column are not circled, apply 0 to the corresponding multiplexer input.

If the two minterms are circled, apply 1 to the corresponding multiplexer input.

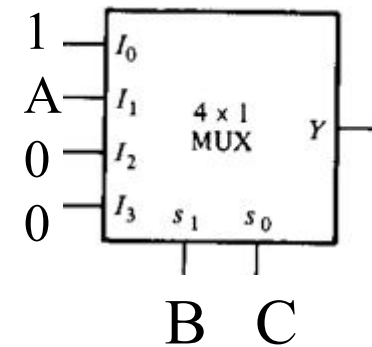
If the bottom minterm is circled and the top is not circled, apply  $A$  to the corresponding multiplexer input.

If the top minterm is circled and the bottom is not circled, apply  $A'$  to the corresponding multiplexer input.

Answer: Part ( c )

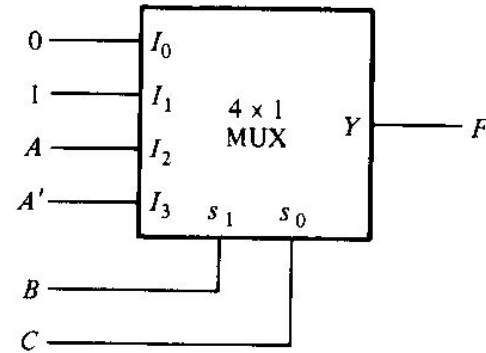
Minterm	$A$	$B$	$C$	$F$
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

	$I_0$	$I_1$	$I_2$	$I_3$
$A'$	0	1	2	3
$A$	4	5	6	7
	1	$A$	0	0



# Built the following function using 4x1 Mux

$$F(A, B, C) = \Sigma(1, 3, 5, 6)$$



(a) Multiplexer implementation

Minterm	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

(b) Truth table

	$I_0$	$I_1$	$I_2$	$I_3$
$A'$	0	①	2	③
$A$	4	⑤	⑥	7
	0	1	$A$	$A'$

(c) Implementation table

# Try it yourself

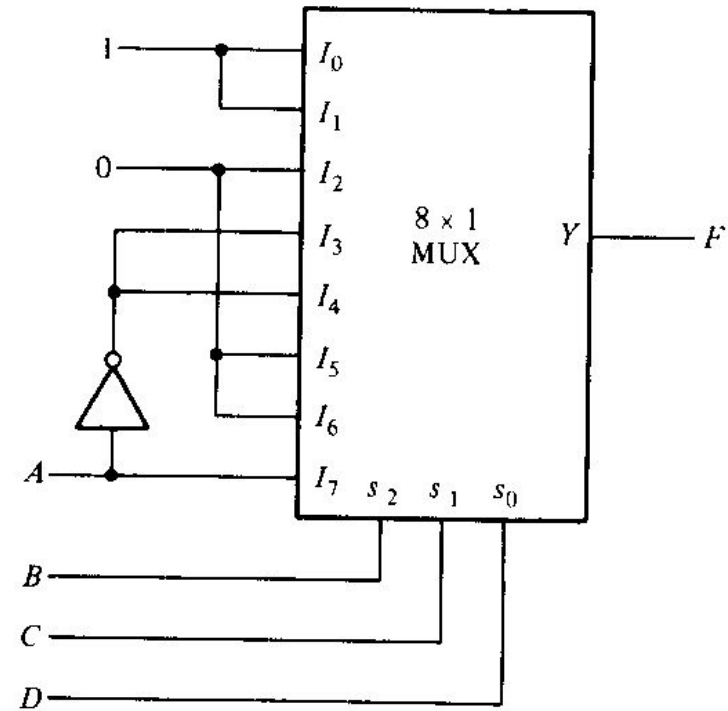
- a) Implement the below function using a 8x1 Mux.
- b) Implement the below function using a 4x16 decoder and OR gates

$$F(A, B, C, D) = \Sigma(0, 1, 3, 4, 8, 9, 15)$$



# Solution

	$I_0$	$I_1$	$I_2$	$I_3$	$I_4$	$I_5$	$I_6$	$I_7$
$A'$	①	①	2	③	④	5	6	7
$A$	⑧	⑨	10	11	12	13	14	⑮
	1	1	0	$A'$	$A'$	0	0	$A$



# Home-task: Try it yourself

Using 2:1 MUX design

- AND gate
- OR gate
- NOT gate

# Note:

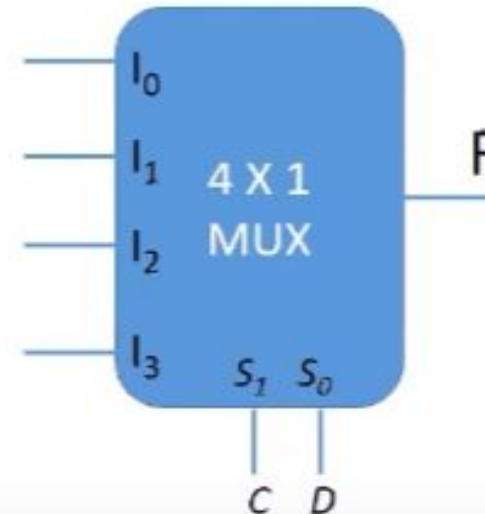
- Both mux and decoder can be used to design combinational circuit.
- Decoder are mostly used to decoding binary information and mux are mostly used to select path between multiple sources and a single destination.

# 16 variables with 4x1 Mux

## Implementation using Multiplexer

$F(A, B, C, D) = \sum(1, 3, 4, 11, 12, 13, 14, 15)$  using 4 x 1 Multiplexer with A, B as input variables and C, D as selection variables

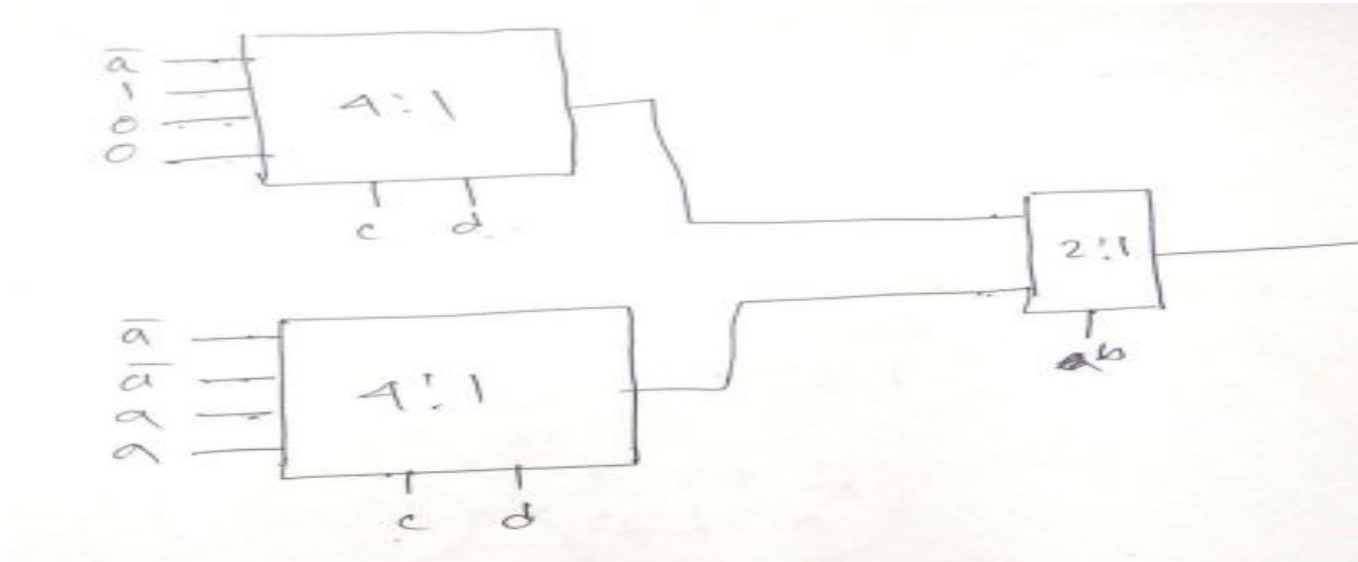
	$I_0$	$I_1$	$I_2$	$I_3$
$A'B'$	0	1	2	3
$A'B$	4	5	6	7
$AB'$	8	9	10	11
$AB$	12	13	14	15
	B	$A \odot B$	$AB$	$A+B'$



A	B	C	D	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

$$F(a,b,c,d) = \sum(0,1,4,5,9,14,15)$$

Implement the above boolean function using 4:1 MUX(s) and 2:1 MUX(s).



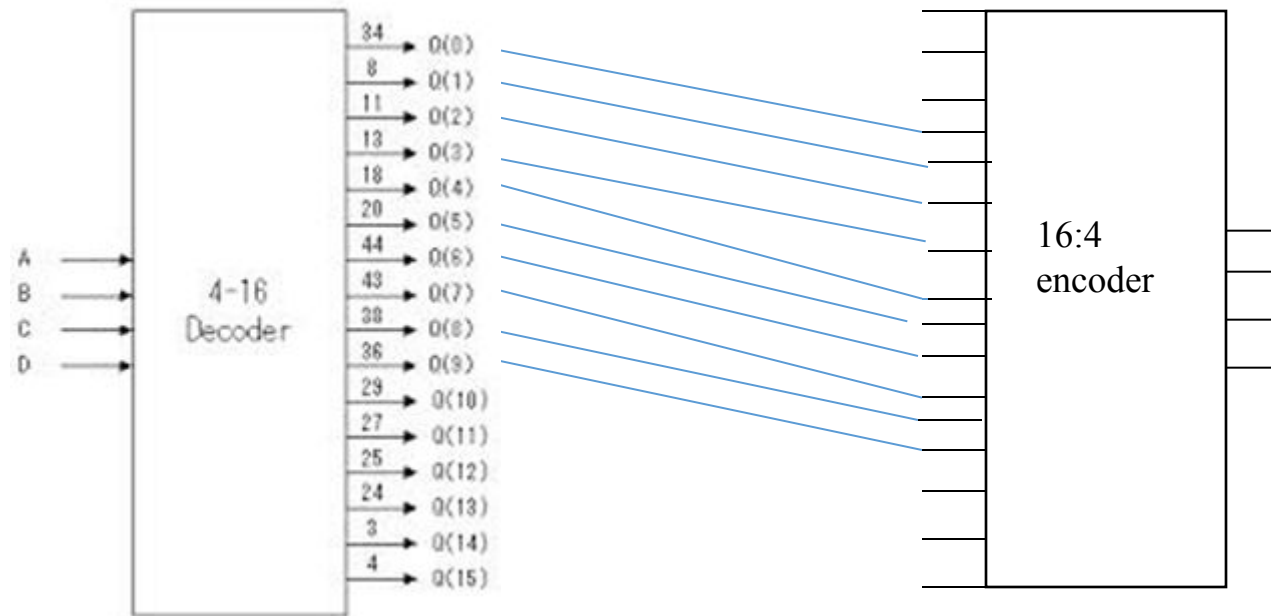
	$I_0$	$I_1$	$I_2$	$I_3$	$I_4$	$I_5$	$I_6$	$I_7$
$a$	0	1	2	3	4	5	6	7
$\bar{a}$	8	9	10	11	12	13	14	15
	$\bar{a}$	1	0	0	$\bar{a}$	$\bar{a}$	$a$	$a$

Combining MSI to build  
Combinational Design Circuit

# Exercise time!

- Design a BCD to Excess 3 code converter using '4x16' decoder and '16x4' encoder

# Solution

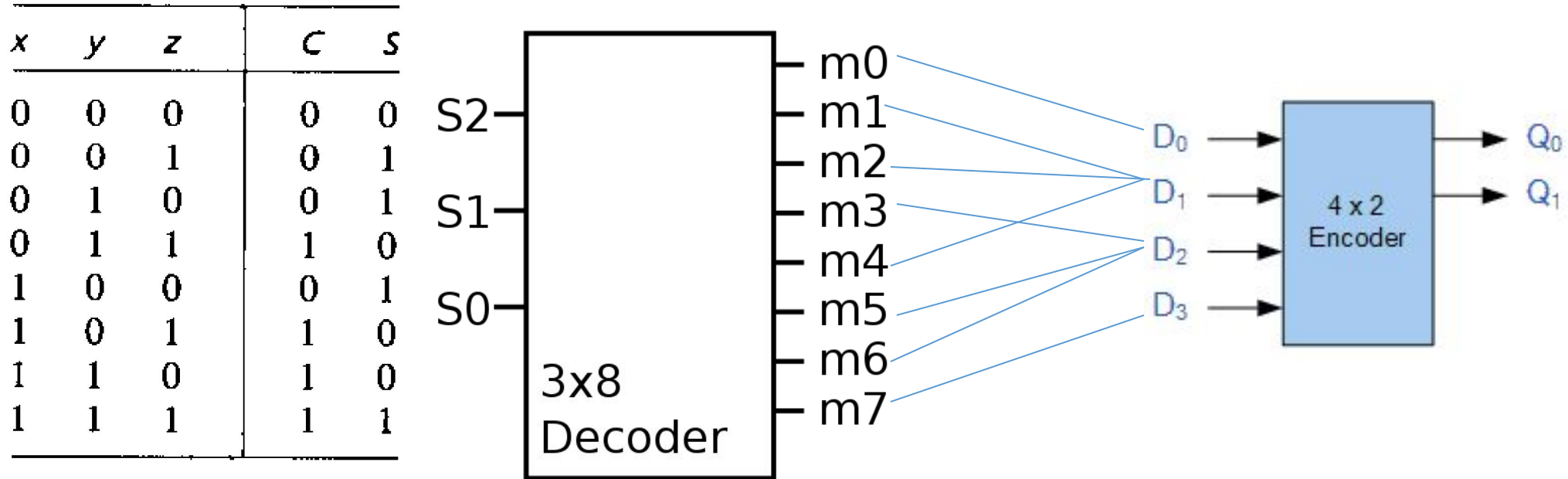




# Try it yourself

- Design a full adder using '3x8' decoder and '4x2' encoder

# Solution:



# Try it yourself

- Design '4x1' mux using '2x4' decoder
- Design '4x1' demux using '2x4' decoder