به نام خدا

طراحی سیستم های دیجیتال ۱

فصل چهارم

مدارهای منطقی ترکیبی

Combinational Logic Circuits

Dr. M. Beigizadeh

✓ مدارهای منطقی ترکیبی

❖ مدارهای منطقی:

✓ **ترکیبی:** خروجی در هر لحظه به ورودی در همان لحظه وابسته است.

✓ **ترتیبی:** خروجی در هر لحظه به ورودی در همان لحظه و به مقادیر قبلی خروجی وابسته است.

❖ مهمترین مدارهای منطقی ترکیبی:

✓ Adders

✓ Subtractors

✓ Decoders

✓ Encoder

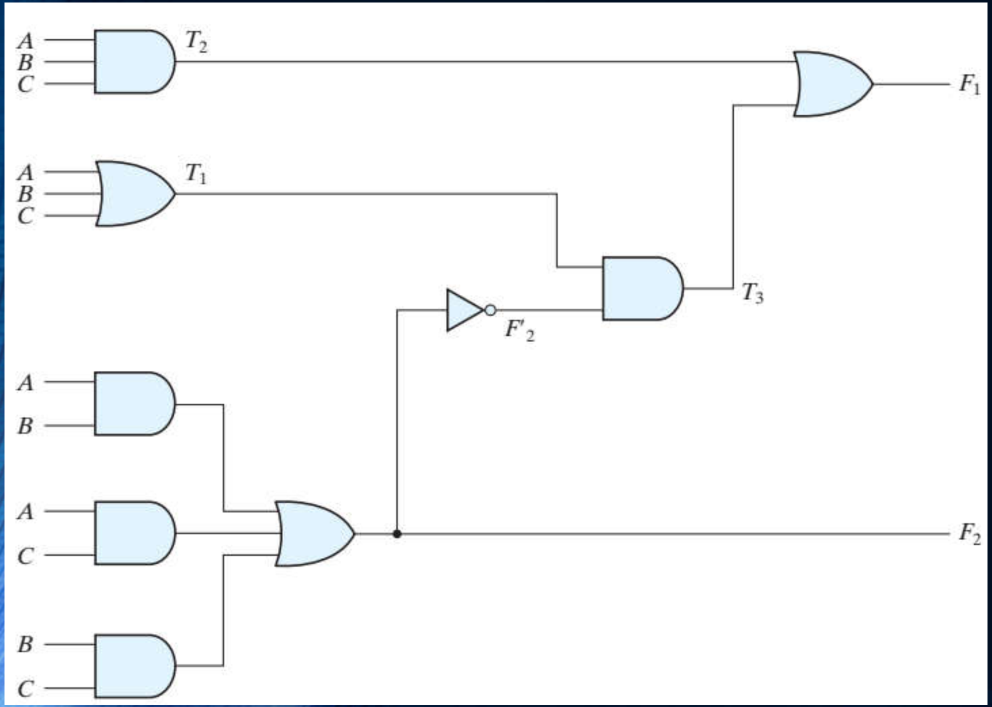
✓ Multiplexer

✓ مدارهای منطقی ترکیبی

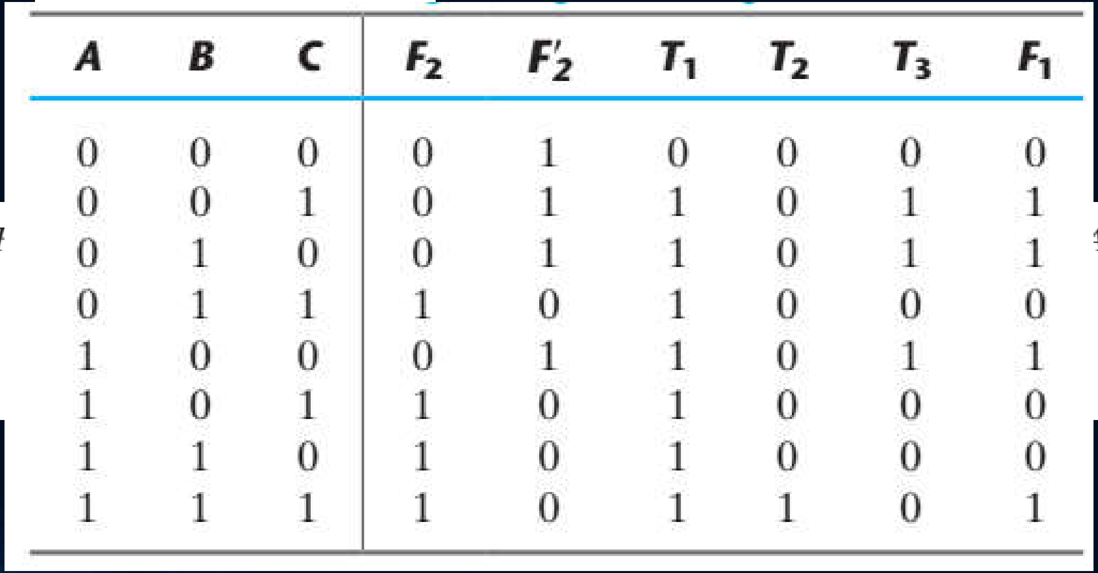
❖ روند آنالیز (Analysis Procedure):

✓ در آنالیز مدار ترکیبی پیاده سازی شده را داریم و باید از روی آن ضابطه تابع، جدول صحت و تابع ساده شده را بدست بیاوریم.

❖مثال:



$$F_2 = AB + AC + BC$$
$$T_1 = A + B + C$$
$$T_2 = ABC$$



A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

✓ مدارهای منطقی ترکیبی

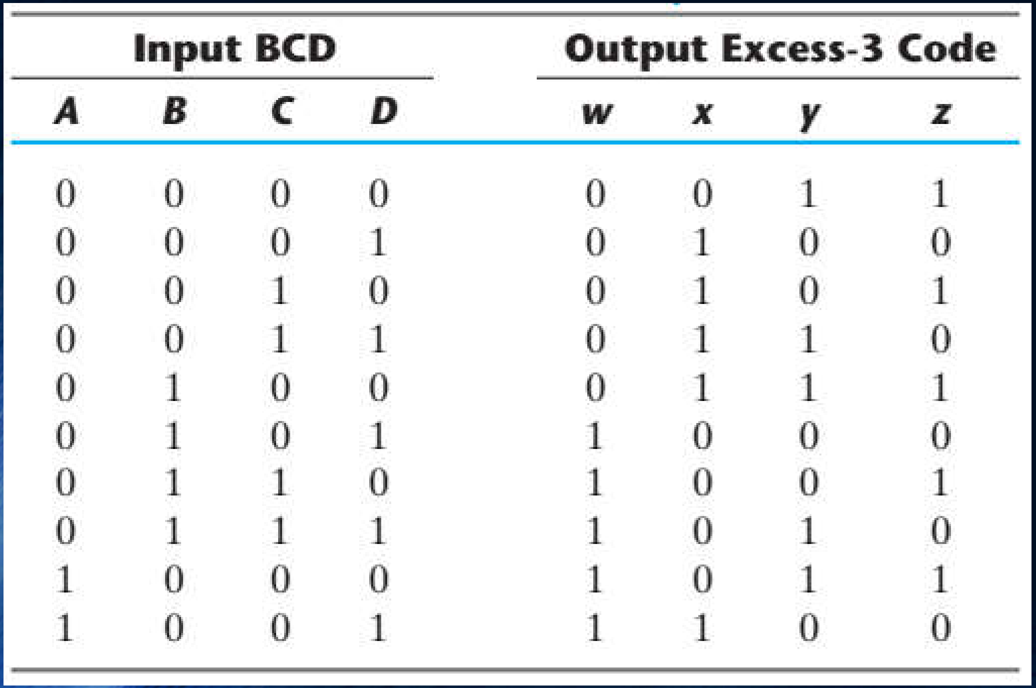
❖ روند طراحی (Design Procedure):

✓ در طراحی صورت مسئله را داریم و باید از روی آن باید ضابطه تابع و درنهایت پیاده سازی را انجام داد.

✓ صورت مسئله ← جدول صحت ← استخراج تابع از روی جدول صحت ← ساده سازی تابع ← پیاده سازی

❖مثالی از کد گردانی (Code Conversion):

طراحی یک مدار برای تبدیل کد BCD به Excess-3

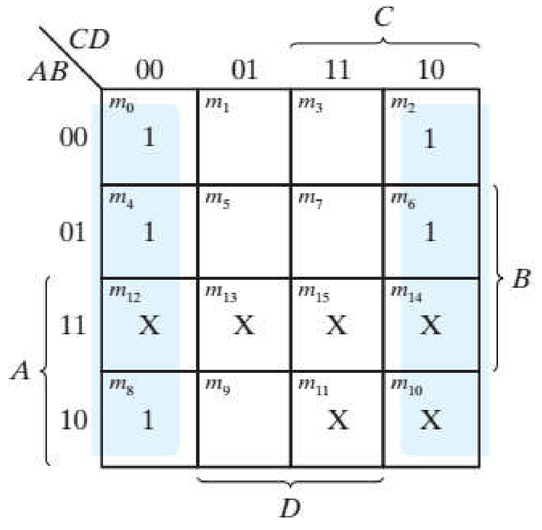


Input BCD				Output Excess-3 Code			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

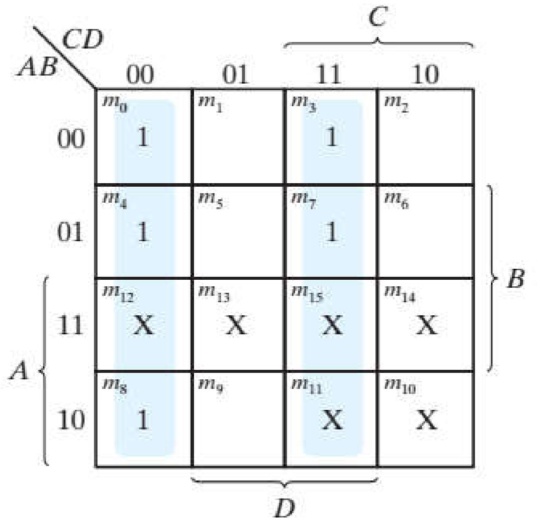
✓ مدارهای منطقی ترکیبی

❖مثالی از کد گردانی (Code Conversion):

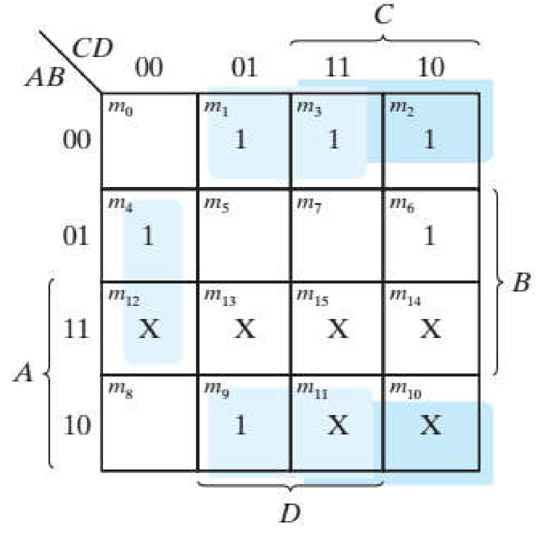
طراحی یک مدار برای تبدیل کد BCD به Excess-3



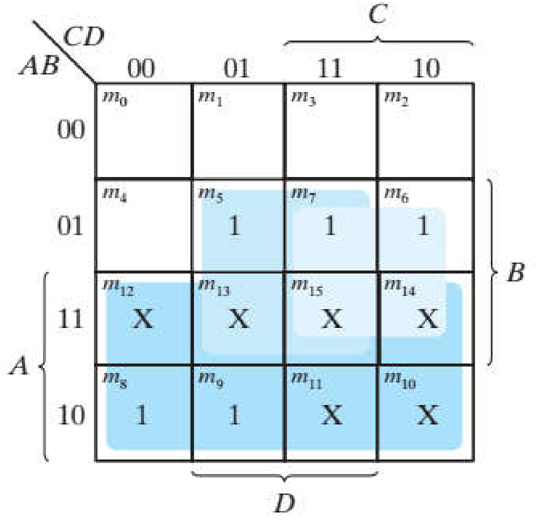
$z = D'$



$y = CD + C'D'$

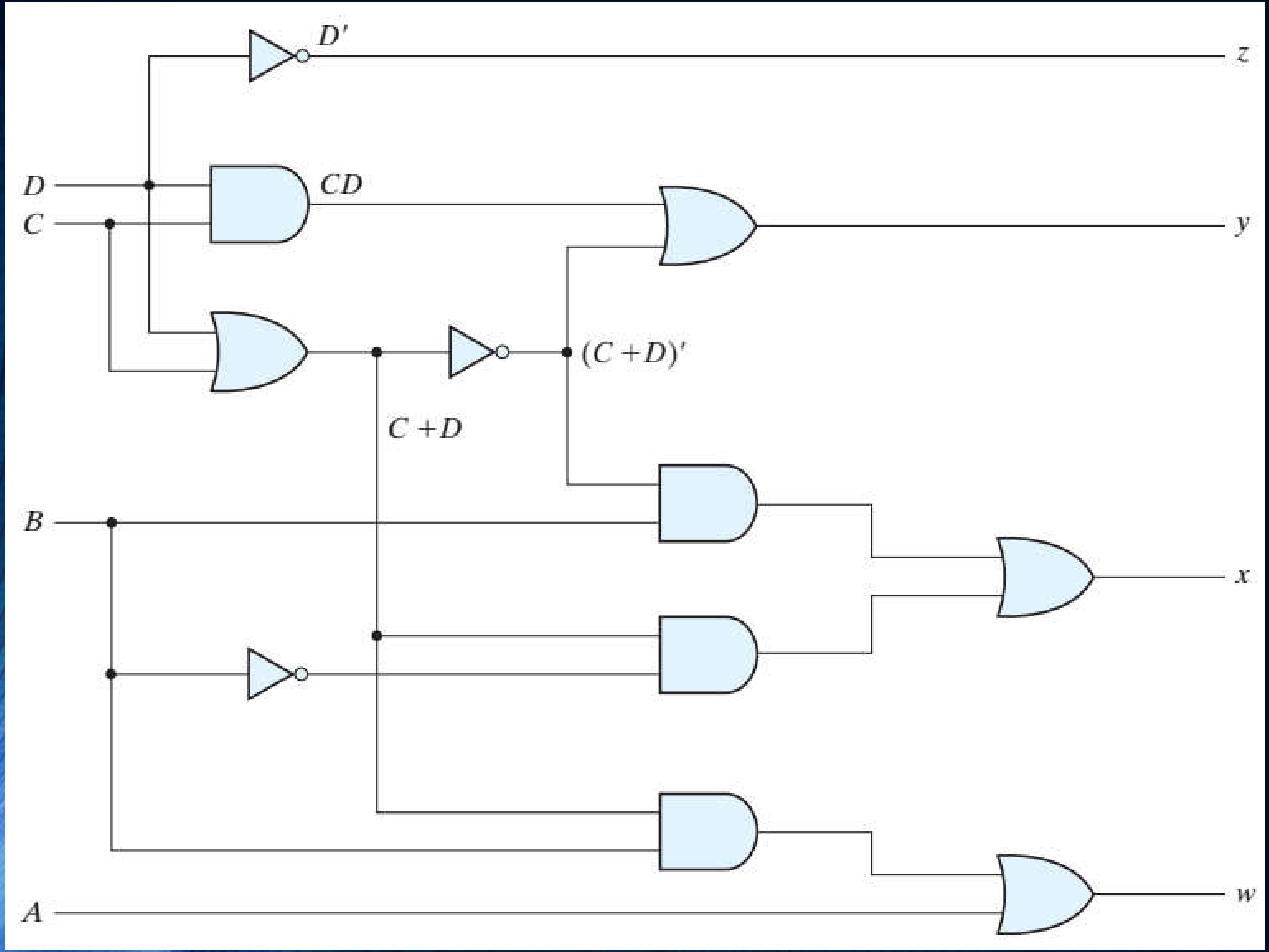


$x = B'C + B'D + BC'D'$



$w = A + BC + BD$

✓ مدارهای منطقی ترکیبی

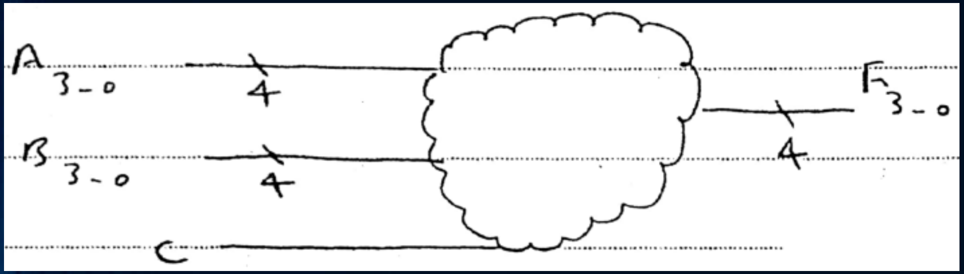


❖مثالی از کد گردانی :

طراحی یک مدار برای تبدیل کد BCD به Excess-3

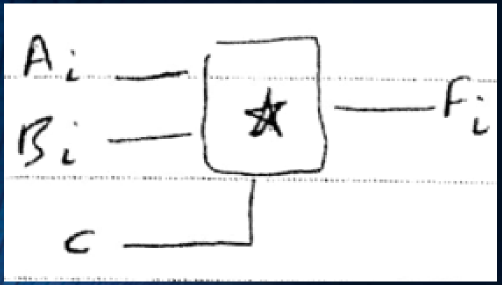
✓ مدارهای منطقی ترکیبی

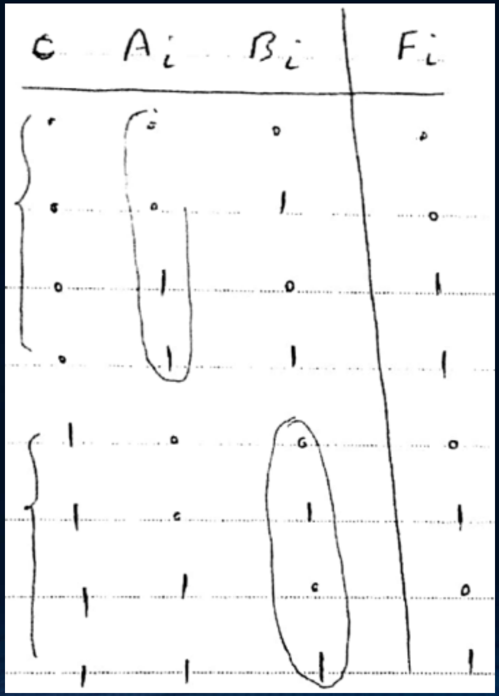
❖**مثال:** مداری طراحی کنید که اگر در آن C=0 شد، چهار ورودی A نظیر به نظیر به چهار خروجی F منتقل شود. در غیراینصورت چهار ورودی B به چهار خروجی F منتقل شود.



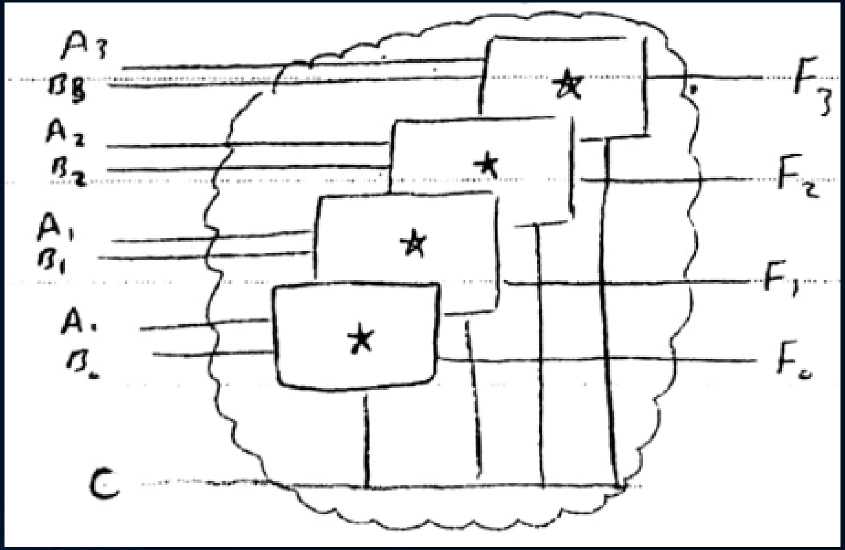
$$F_i = \Sigma m(2,3,5,7)$$

$$F_i = C'A_i + CB_i$$





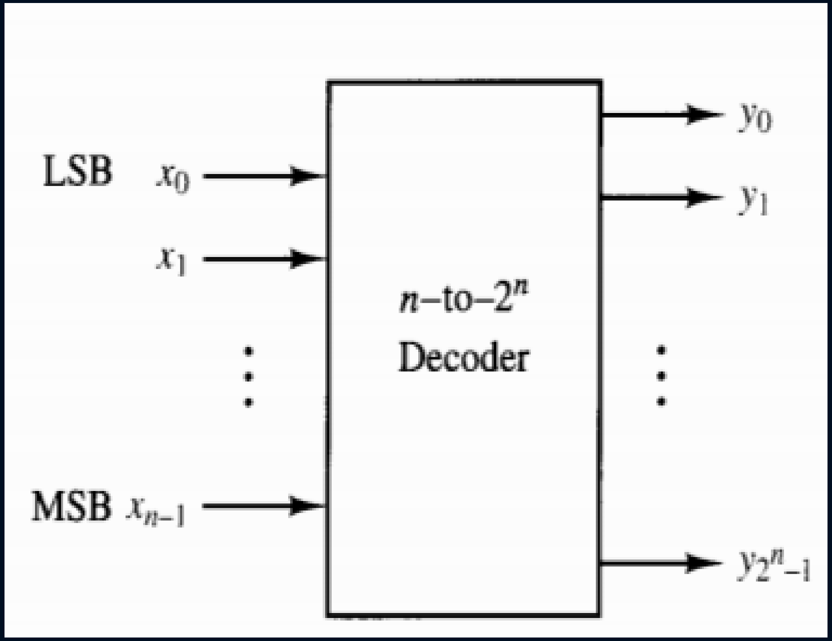
C	A_i	B_i	F_i
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



✓Decoders

❖ An n-to-2^n decoder is a multiple-output combinational logic with n input and 2^n output

❖ For each possible input condition, one and only one output will be at logic 1

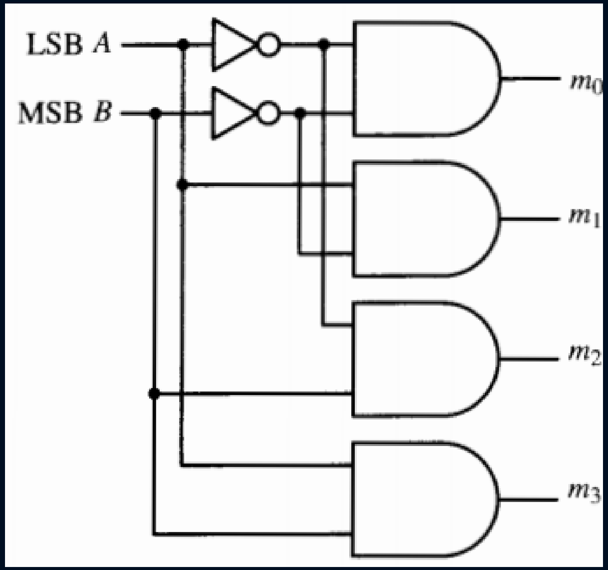


✓Decoders

➢ 2-to-4 decoder

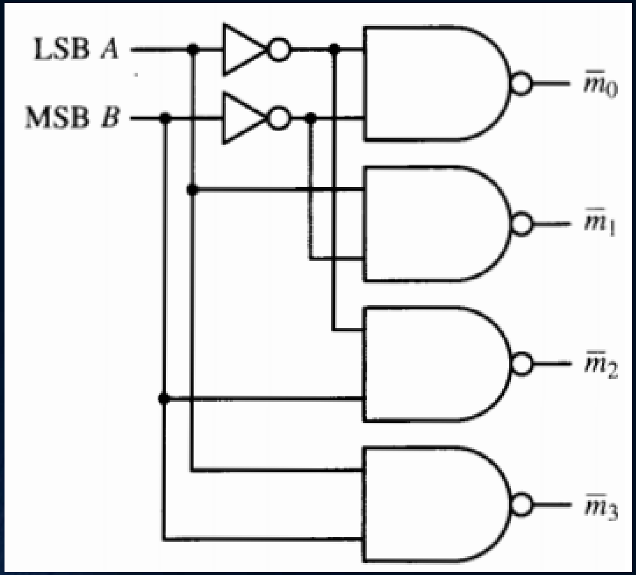
❖ We can consider the n-to-2^n decoder as simply a **minterm generator**

❖ An input combination of BA=00 selects the m0 output line, BA=01 selects the m1 and so on.

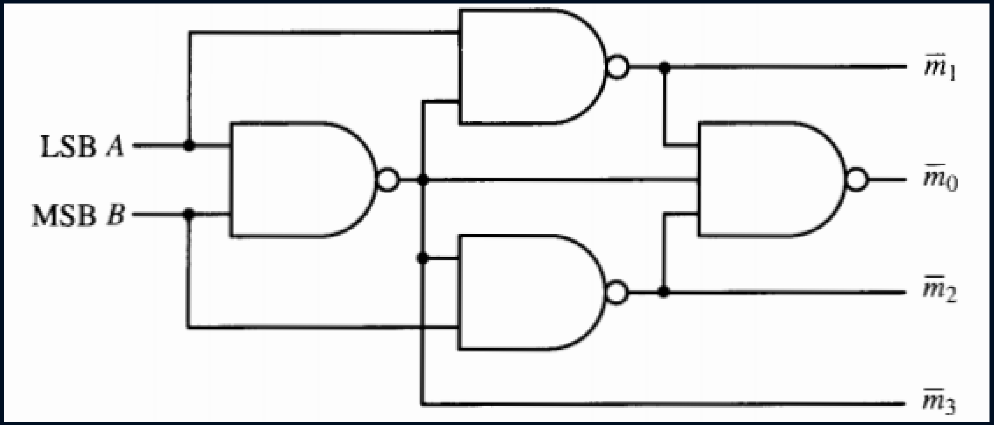


$$m_0 = \bar{B}\bar{A}$$
$$m_1 = \bar{B}A$$
$$m_2 = B\bar{A}$$
$$m_3 = BA$$

✓Decoders



Using only NAND gates

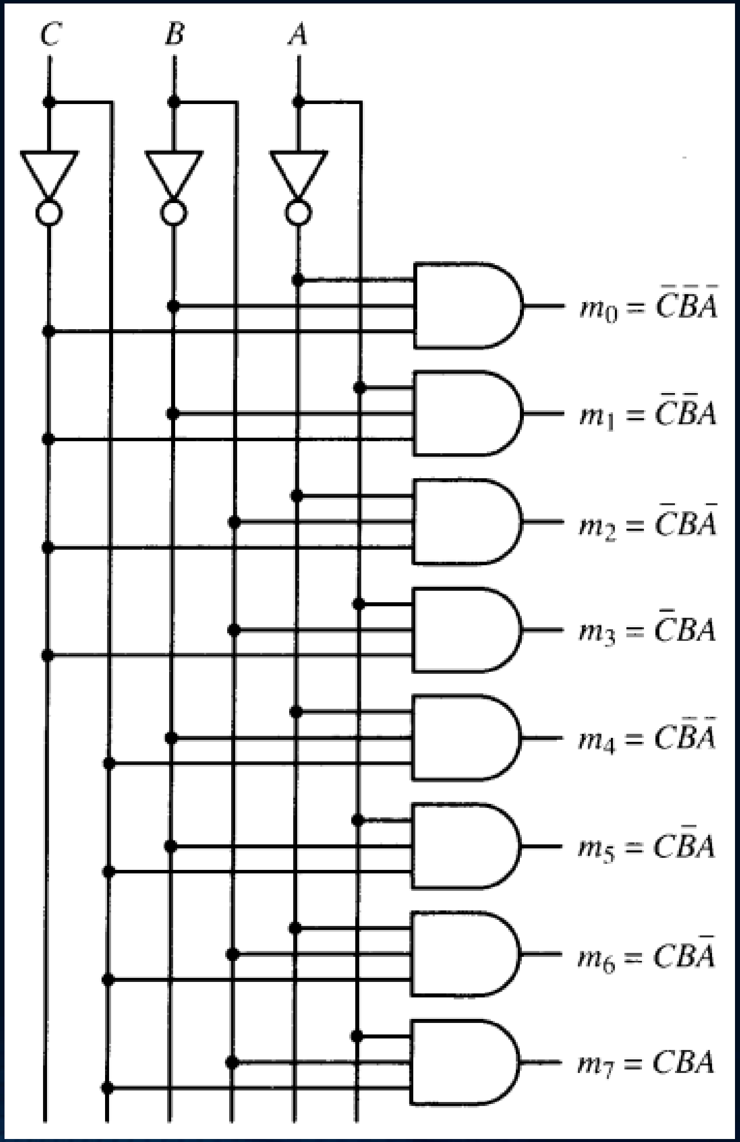


Using only NAND gates
with no inverters

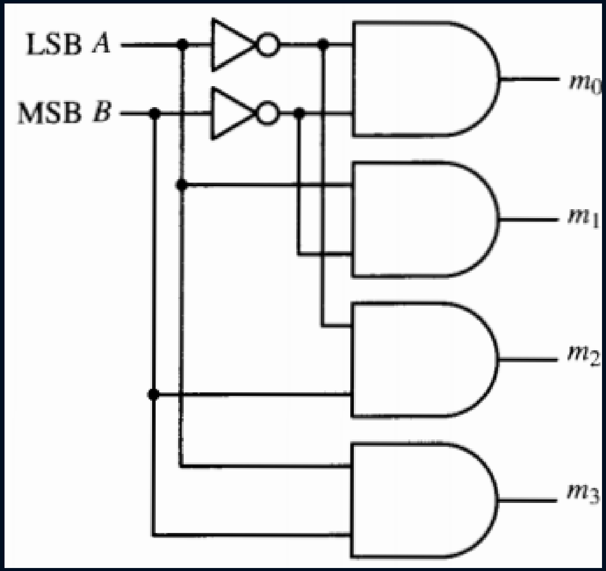
- An output of 0 indicates the presence of the corresponding minterms
- In this case, the outputs are said to be "active low"

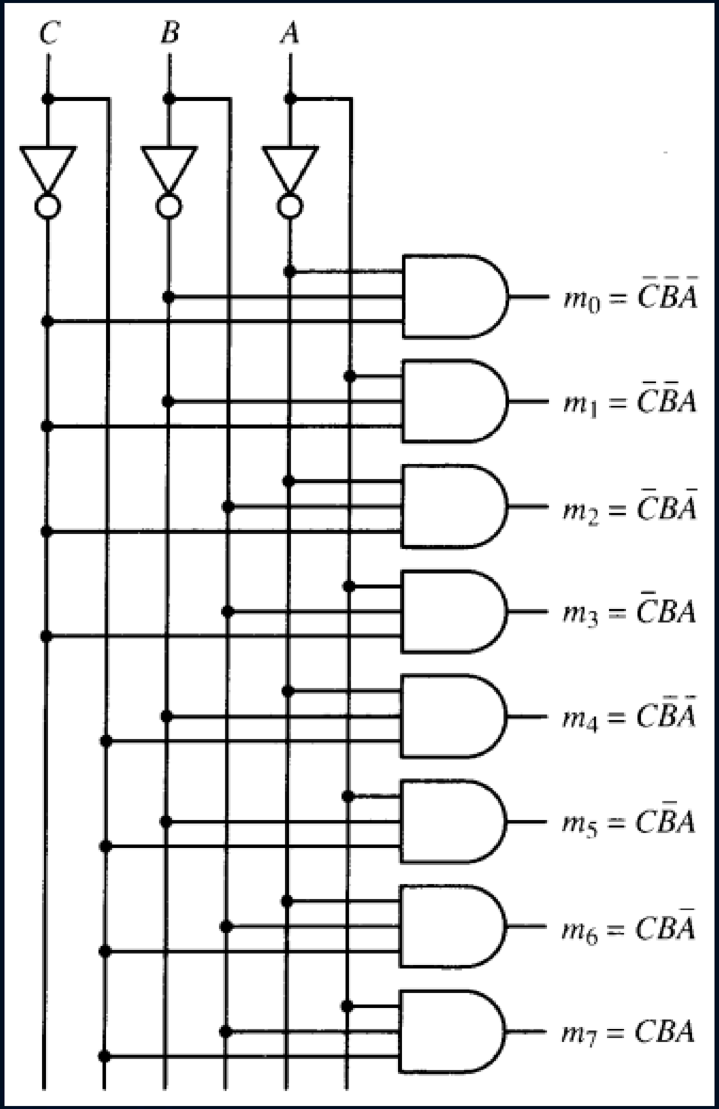
✓Decoders

3-to-8 Decoder
(parallel-type)



✓Decoders

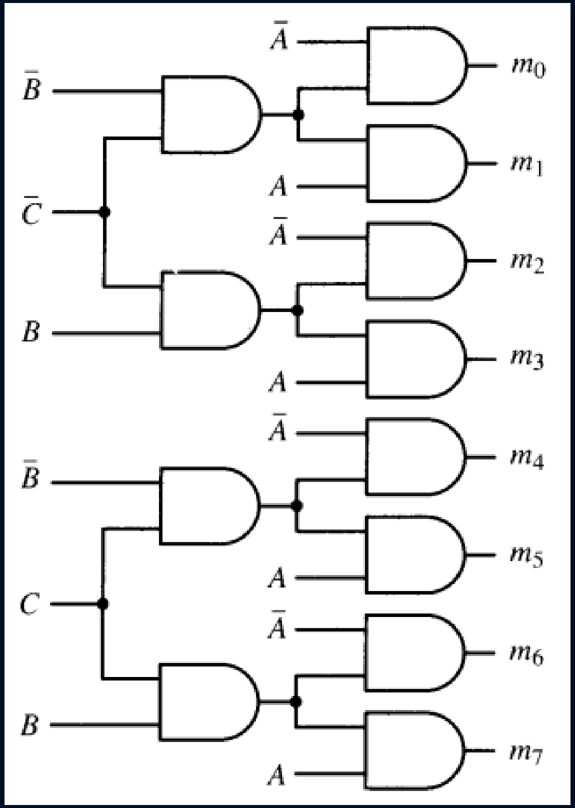




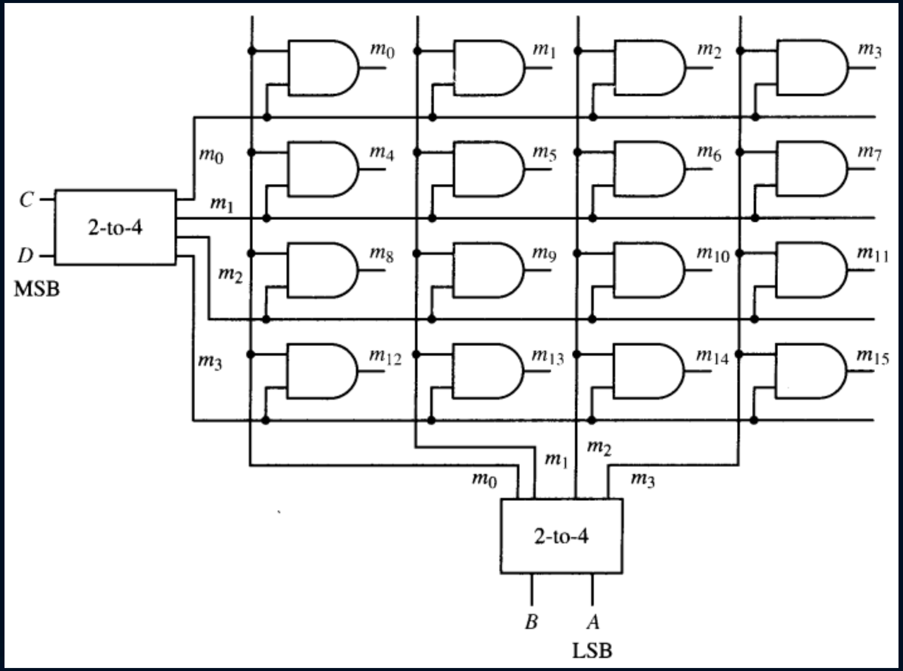
- ❖ There is only a single level of logic
- ❖ One n input AND gate is required for each of the 2^n output lines
- ❖ A problem is occurred as n becomes large because of limitation of the fan-in
- ❖ So, **tree decoder** has been proposed

✓Decoders

❖ **Tree Decoder:** employs multilevel logic with only two-input AND gates, independent of the number of input lines



❖ **Dual Tree Decoder:**



✓ n input are divided into j and k groups

✓ Two smaller decoders j-to-2^j and k-to-2^k are used

✓ Two-input AND gates are used to combine these signals to form the 2^n output lines

✓Decoders

❖ Implementing logic function

✓ Decoders output signals in complement form are suitable for further processing using NAND logic

$$f(A, B, \ldots, Z) = m_i + m_j + \cdots + m_k \longrightarrow f(A, B, \ldots, Z) = \overline{\bar{m}_i \cdot \bar{m}_j \cdot \cdots \cdot \bar{m}_k}$$

✓ This function can be implemented using a single k-input NAND gate and a decoder with active-low outputs

✓ This is equal to implement a function with its maxterm $M_i = \bar{m}_i$

$$f(A, B, \ldots, Z) = M_i \cdot M_j \cdot \cdots \cdot M_k \longrightarrow$$

Using a decoder with active low outputs and an AND gate

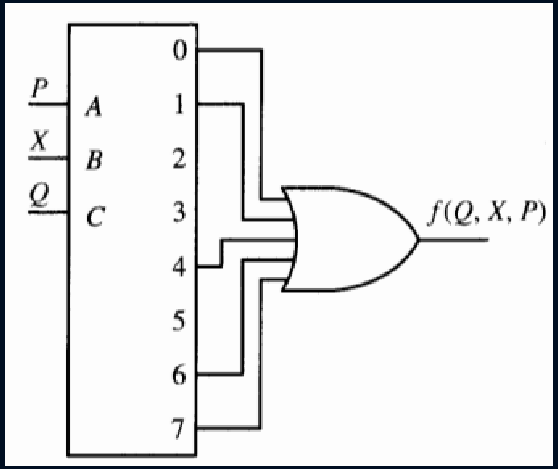
✓Decoders

➢ **Example:** Implement the following function

$$f(Q,X,P) = \sum m(0,1,4,6,7) = \prod M(2,3,5)$$

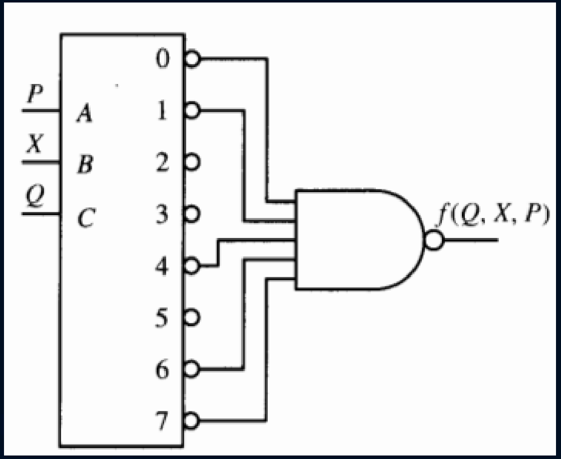
❖ Use a decoder (with active high outputs) with an OR gate

$$f(Q,X,P) = m_0 + m_1 + m_4 + m_6 + m_7$$



❖ Use a decoder (with active low outputs) with an NAND gate

$$f(Q,X,P) = \overline{\bar{m}_0 \cdot \bar{m}_1 \cdot \bar{m}_4 \cdot \bar{m}_6 \cdot \bar{m}_7}$$



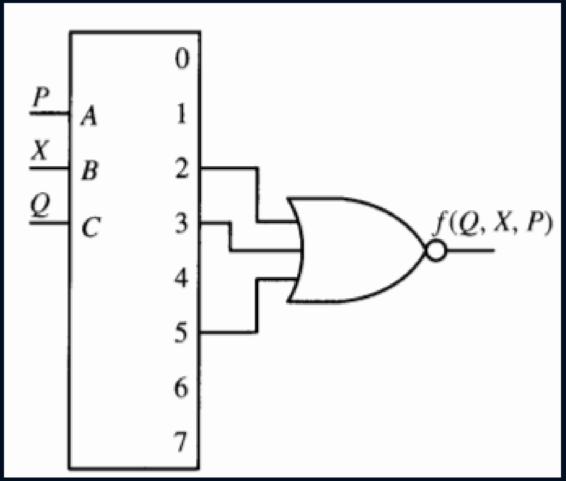
✓Decoders

❖ **Example:**

$$f(Q,X,P) = \sum m(0,1,4,6,7) = \prod M(2,3,5)$$

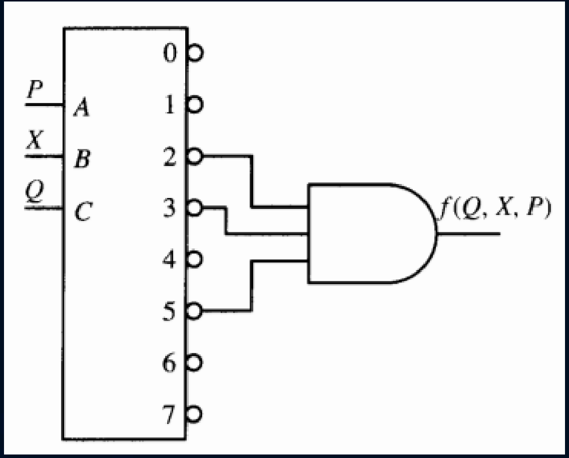
❖ Use a decoder (with active high outputs) with an NOR gate

$$f(Q,X,P) = \overline{m_2 + m_3 + m_5}$$



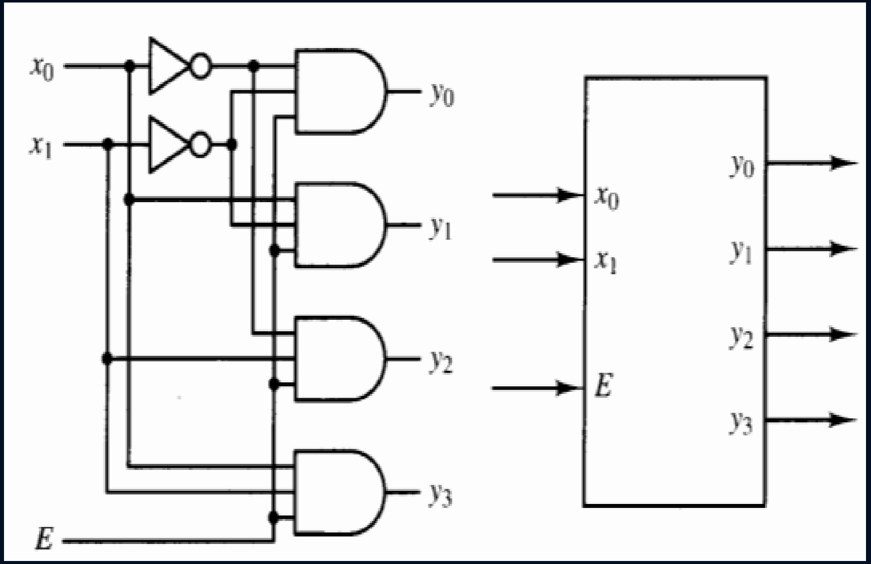
❖ Use a decoder (with active low outputs) with an AND gate

$$f(Q,X,P) = \bar{m}_2 \cdot \bar{m}_3 \cdot \bar{m}_5$$



✓Decoders

➢ **Enable Control Input**



❖Enable Inputs can be used to either enable or disable the designated functions to be performed

❖The decoding function of a decoder is disabled by forcing all its outputs to the inactive state

❖In general, we have $y_k = m_k E$

When E=0, all outputs are forced to 0, whereas for E=1, y_k is equal to m_k

❖A common use of the enable function of a decoder is to extend the decoding capability to implement larger decoder by cascading smaller decoders

✓Decoders

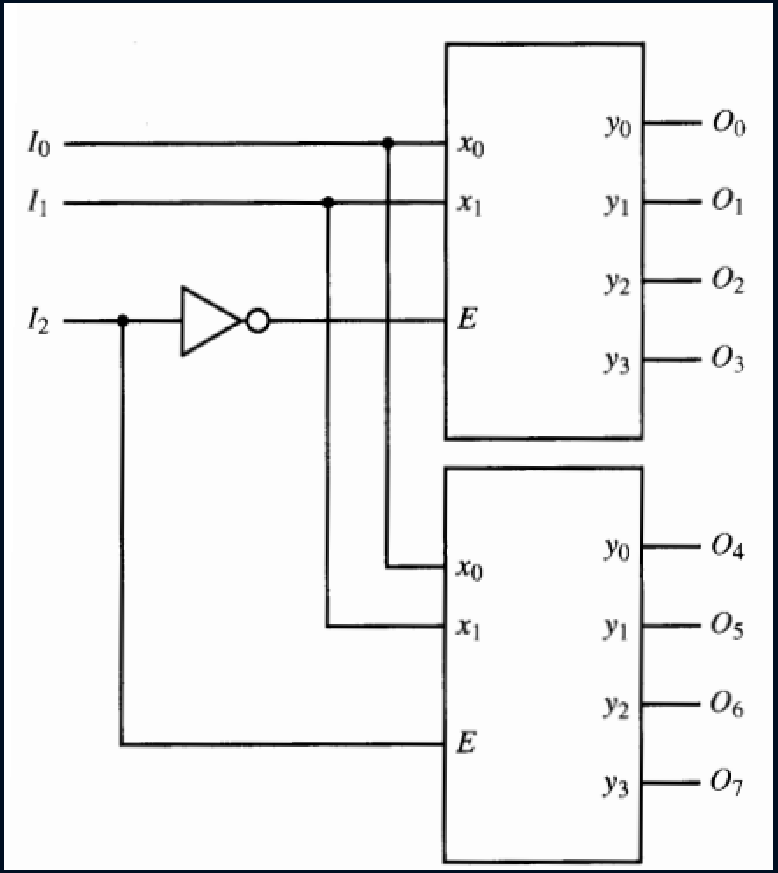
➢ **Use of 2-to-4 decoder to realize 3-to-8 decoder**

❖Input $I_2 = 0$ enables the top decoder

❖Thus, generates input codes $I_2I_1I_0$ equal to 000, 001, 010, 011 (codes 0 through 3)

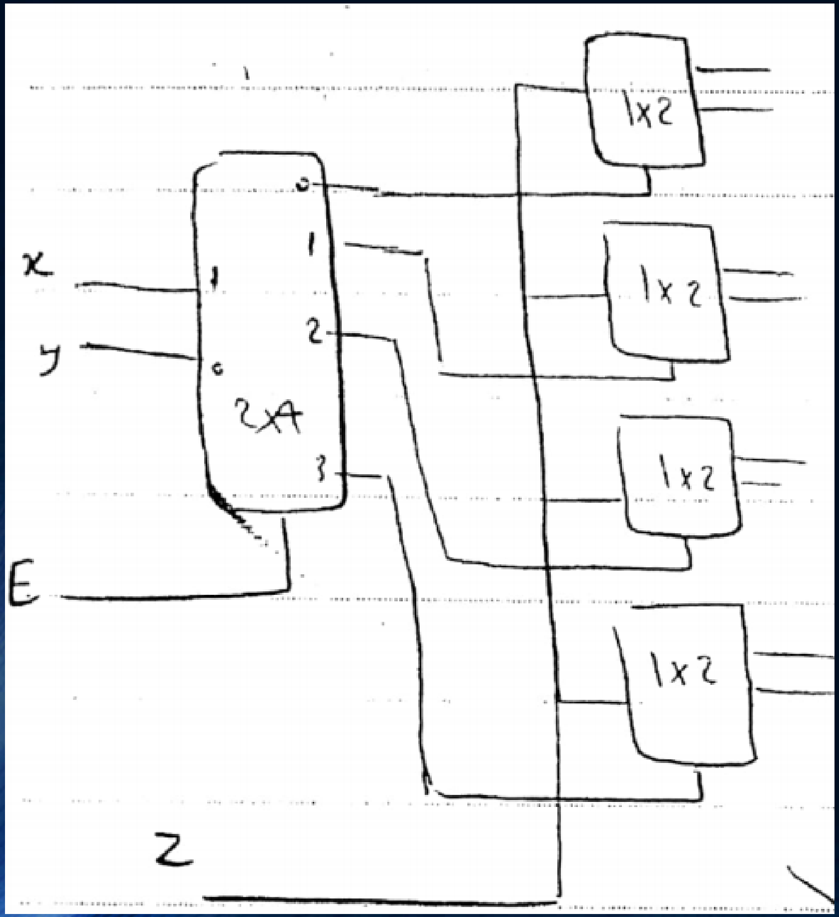
❖Input $I_2 = 1$ enables the bottom decoder

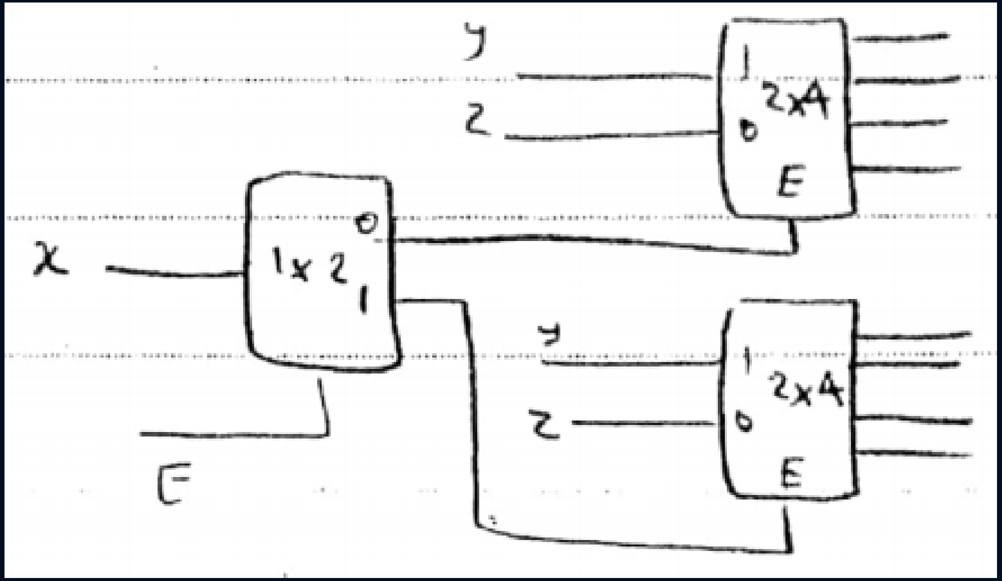
❖Thus, generates input codes $I_2I_1I_0$ equal to 100, 101, 110, 111 (codes 4 through 7)



✓Decoders

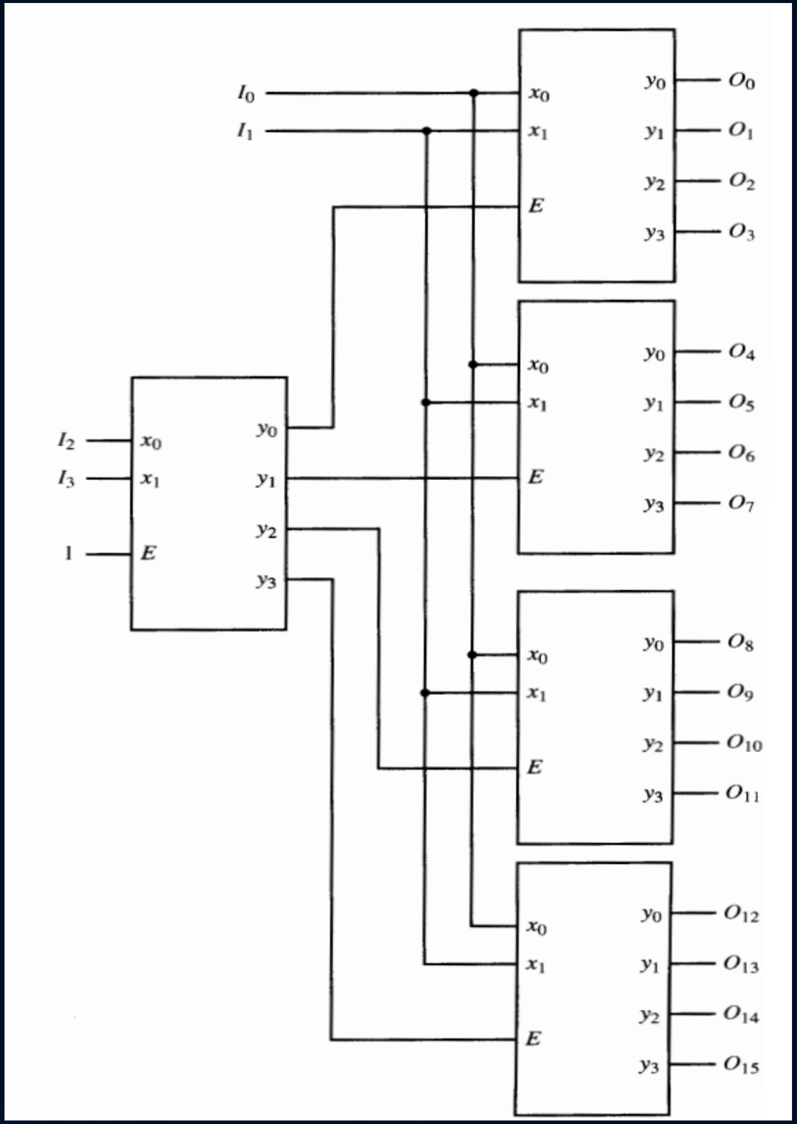
➢ **Use of 2-to-4 and 1-to-2 decoder to realize 3-to-8 decoder**





✓Decoders

- ❖ **Use of 2-to-4 decoder to realize 4-to-16 decoder**
- ❖ Two common standard MSI decoders are 74138 (3-to-8 decoder) and 74154 (4-to-16 decoder)
- ❖ ***Decoder Applications:***
  - ✓Address Decoding (Computer Memories)
  - ✓Minterm Generation
  - ✓Code Converter (BCD to Decimal, …)
  - ✓Display Decoders (7-Segment)

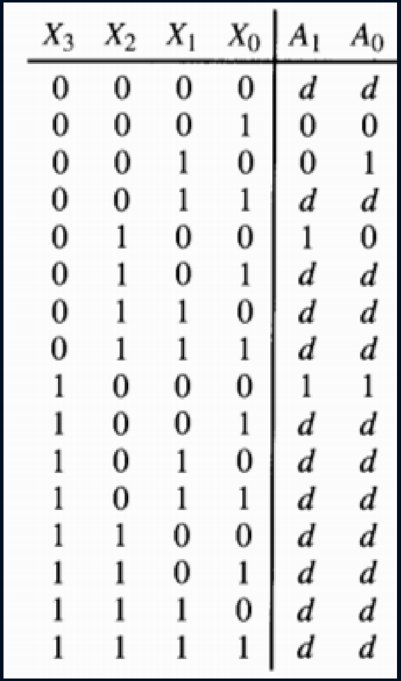


✓Encoders

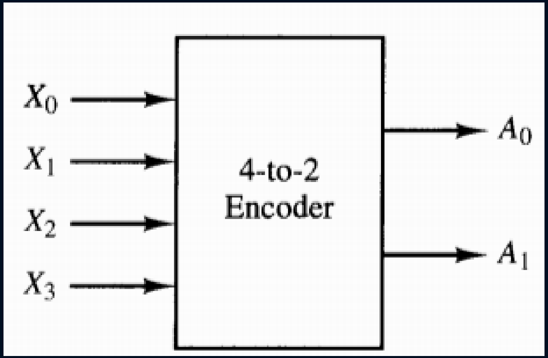
❖ An Encoder is the opposite of a decoder which has 2^n input lines and n output lines

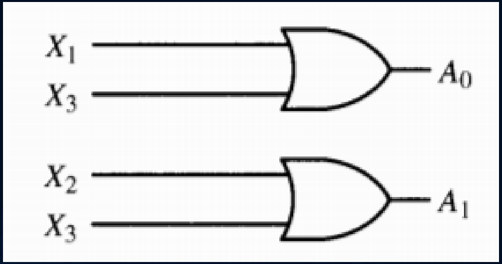
➢ **Encoder Circuit Structure**

- *Encoders with Mutually Exclusive Inputs*
  - ✓One and only one of the input lines is active
  - ✓Two or more input lines are never simultaneously active
  - ✓The input combinations that never occur may be used as don’t-care
  - ✓The output functions yield the binary value of the input variable’s subscript



X_3	X_2	X_1	X_0	A_1	A_0
0	0	0	0	d	d
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	d	d
0	1	0	0	1	0
0	1	0	1	d	d
0	1	1	0	d	d
0	1	1	1	d	d
1	0	0	0	1	1
1	0	0	1	d	d
1	0	1	0	d	d
1	0	1	1	d	d
1	1	0	0	d	d
1	1	0	1	d	d
1	1	1	0	d	d
1	1	1	1	d	d



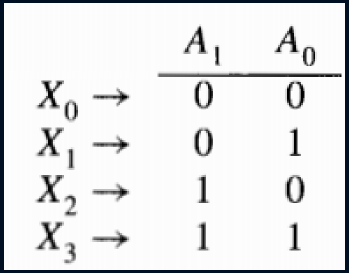


$$A_1 = X_3 + X_2$$
$$A_0 = X_3 + X_1$$

✓Encoders

➢ **Encoder Circuit Structure**

- *Priority Encoders*



	A_1	A_0
$X_0 \rightarrow$	0	0
$X_1 \rightarrow$	0	1
$X_2 \rightarrow$	1	0
$X_3 \rightarrow$	1	1

✓Multiple input lines can be active

✓Sends out the binary value of the subscript of the input line with highest priority

✓The highest priority is assigned to the highest subscript

✓If no input line is active, the priority encoder sends out (00)

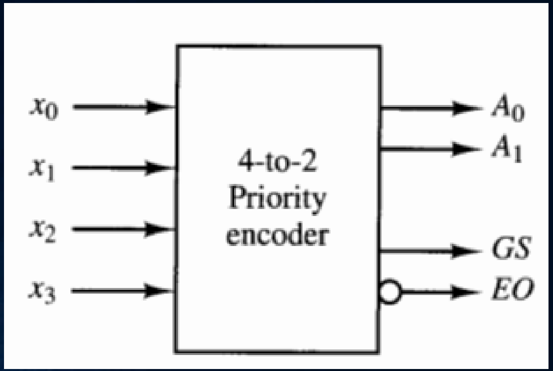
✓If a single line is active, the encoder sends out the binary value of the subscript of the active line

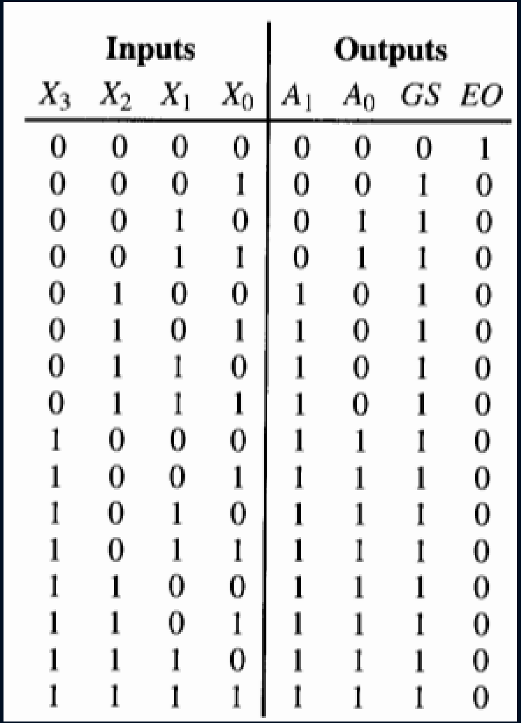
✓If more than one input is active, the encoder sends out the binary value of the largest subscript of the active lines

✓Encoders

➢ **Encoder Circuit Structure**

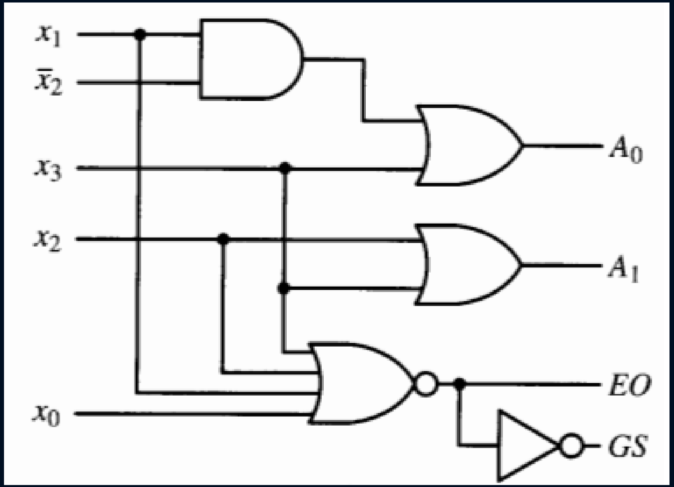
- *Priority Encoders*





Inputs				Outputs			
X_3	X_2	X_1	X_0	A_1	A_0	GS	EO
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	1	1	0
0	0	1	1	0	1	1	0
0	1	0	0	1	0	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	0
1	0	0	0	1	1	1	0
1	0	0	1	1	1	1	0
1	0	1	0	1	1	1	0
1	0	1	1	1	1	1	0
1	1	0	0	1	1	1	0
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	0
1	1	1	1	1	1	1	0

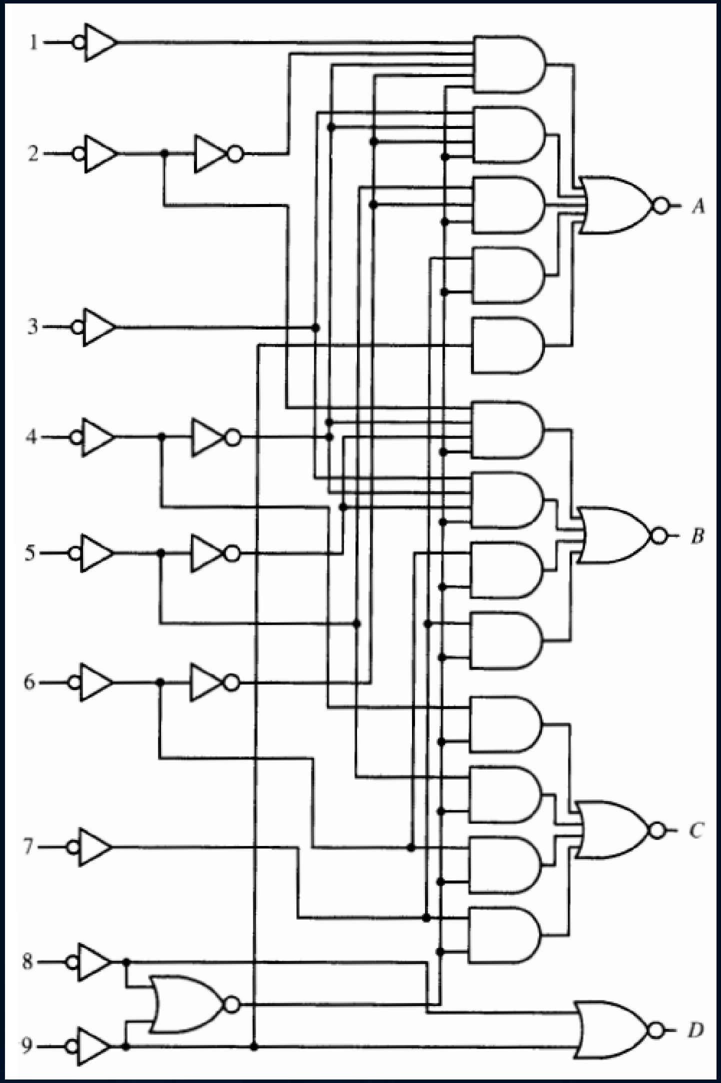
$$A_1 = X_2 + X_3$$
$$A_0 = X_3 + X_1\bar{X}_2$$
$$EO = \overline{GS} = \overline{X_3 + X_2 + X_1 + X_0}$$



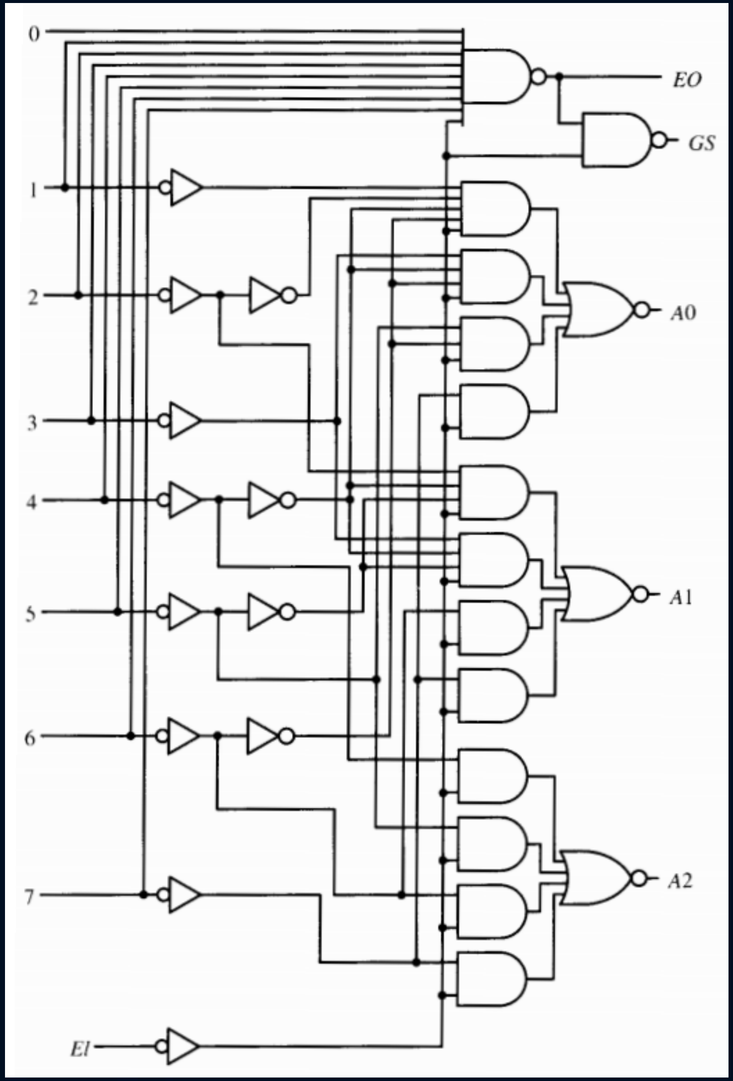
✓Two additional output lines indicates that no input line is active (EO=1) and one or more inputs are active (GS=1)

✓Encoders

➢ **Standard MSI Encoders**



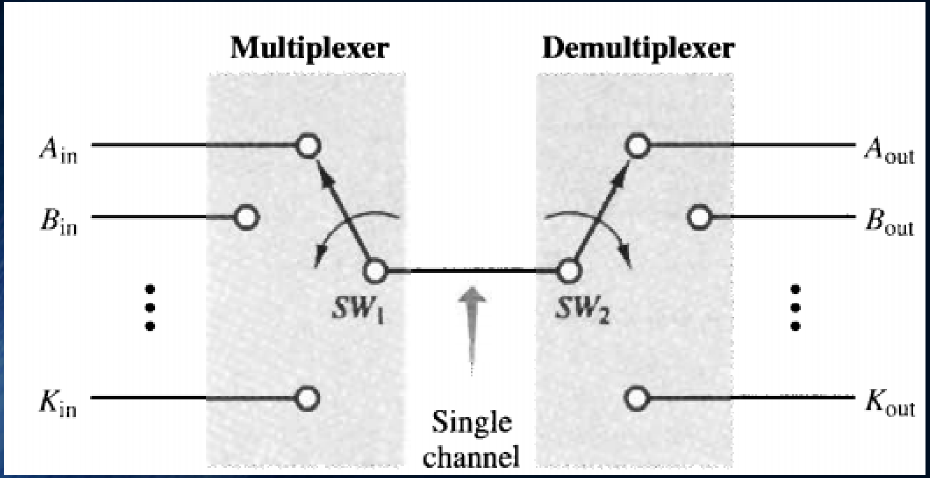
74147

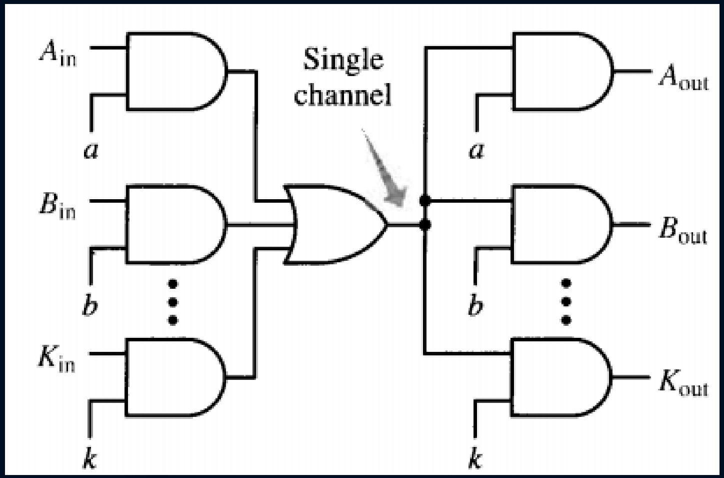


74148

✓Multiplexers

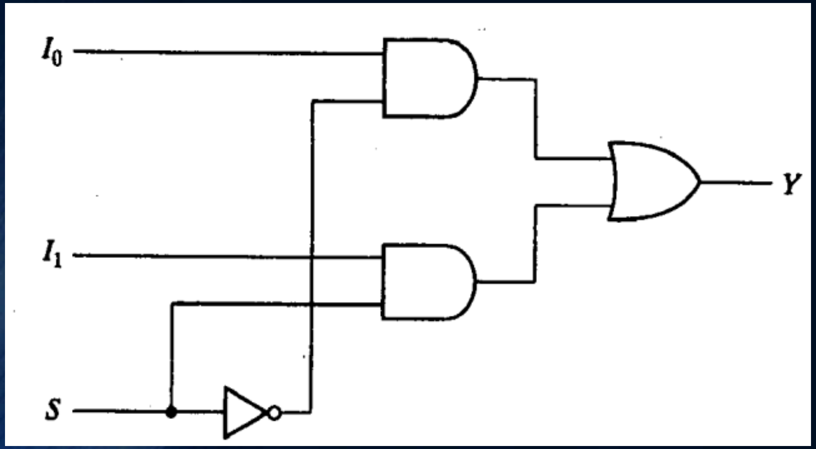
- *Multiplexers (Data Selectors)* is a modular device that selects one of many input lines to appear on a single output line
- A *demultiplexer* performs the inverse operation which takes a single input line and routes it to one of several output lines

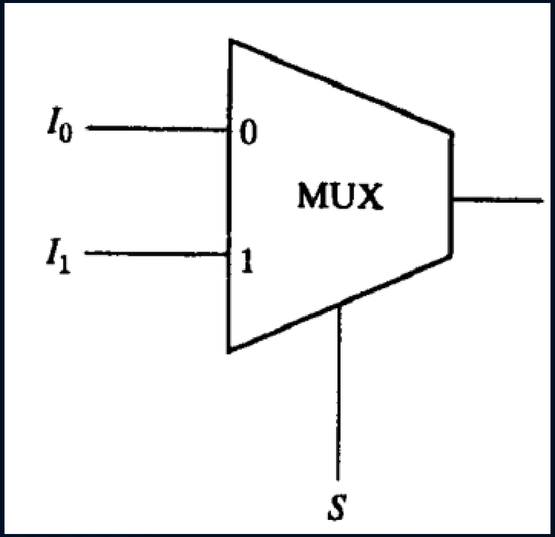




✓ Multiplexers

- **Multiplexer Circuit Structure**
  - ✓ The selection of a particular input line is controlled by a set of selection lines
  - ✓ There are 2^n input lines and n selection lines
  - ✓ A 2-to-1 multiplexer has two data input lines, one output line and one selection line

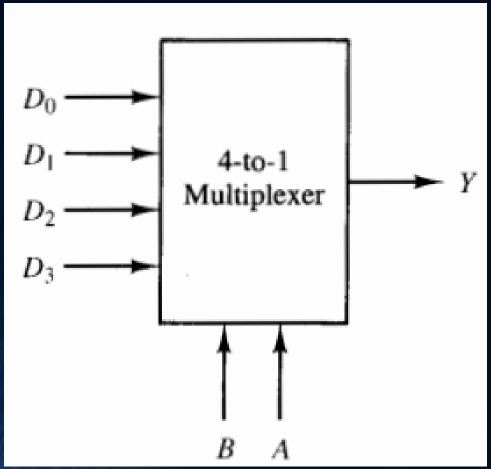




✓Multiplexers

➢ Multiplexer Circuit Structure

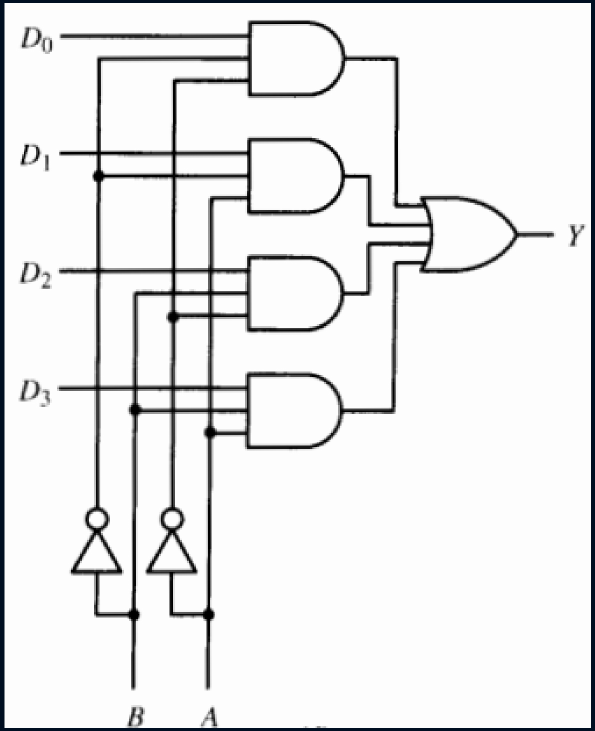
✓A 4-to-1 multiplexer



B	A	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

$$Y = (\bar{B}\bar{A})D_0 + (\bar{B}A)D_1 + (B\bar{A})D_2 + (BA)D_3$$

$$Y = \sum_{i=0}^{3} m_i D_i$$



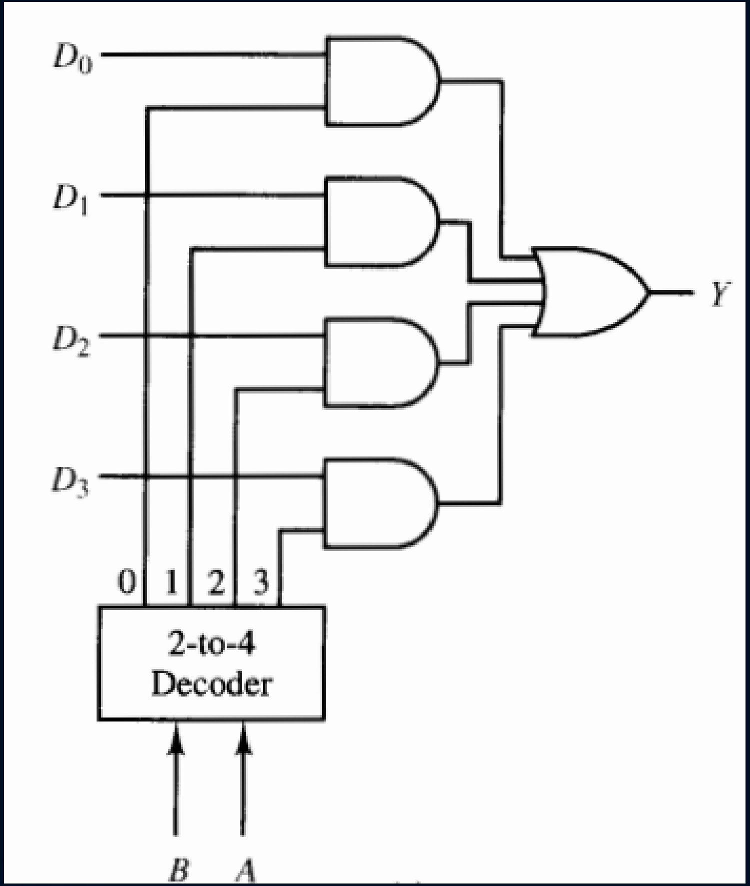
✓mi are the minterms of the selection code

✓The delay of the all 2^n-to-1 multiplexers is 2, because of their SOP forms

✓Multiplexers

➢ **Multiplexer Circuit Structure**

- ✓In general, a 2^n-to-1 line multiplexer is constructed from an n-to-2^n decoder by adding 2^n input lines to it, one to each AND gate



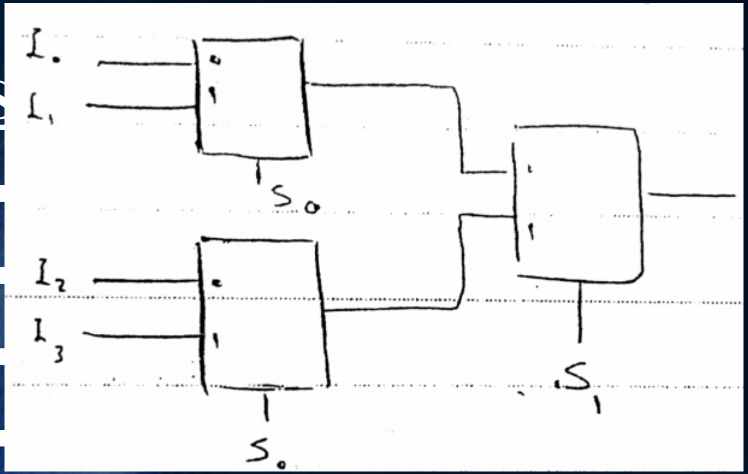
✓Multiplexers

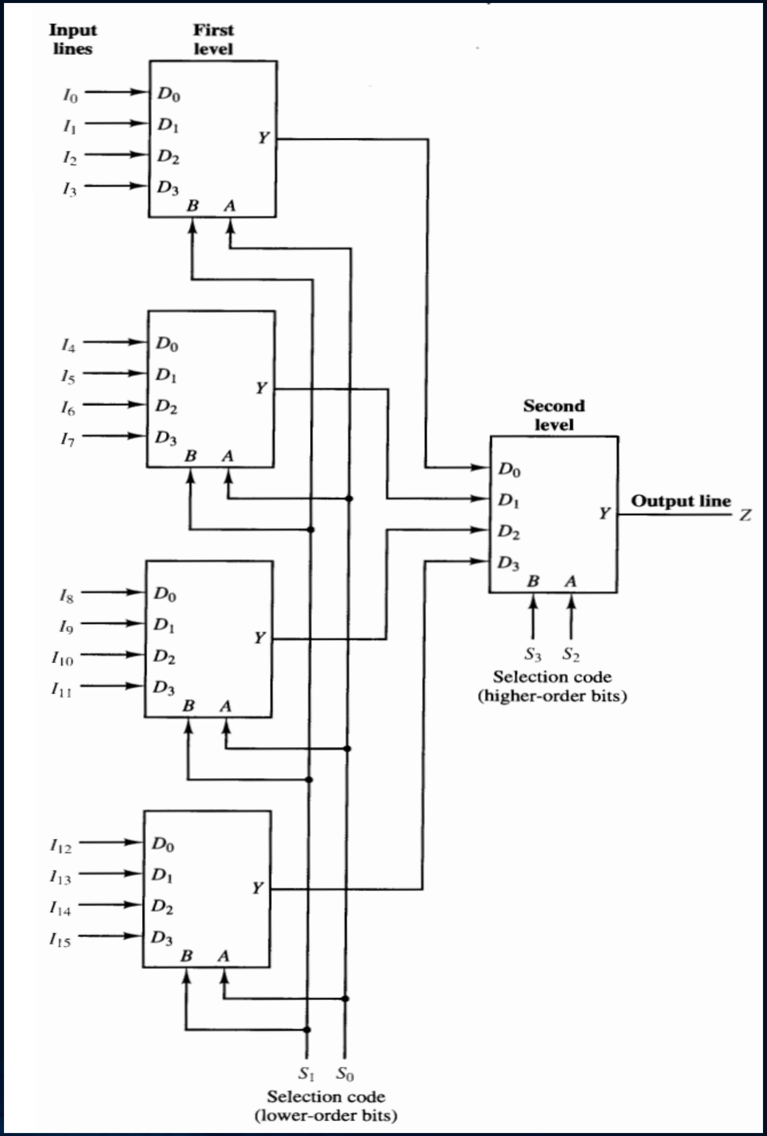
➢ **Multiplexer Circuit Structure**

✓16-to-1 multiplexer using 4-to-1 multiplexers

✓ 4-to-1 multiplexer using 2-to-1 multiplexers

➢ S



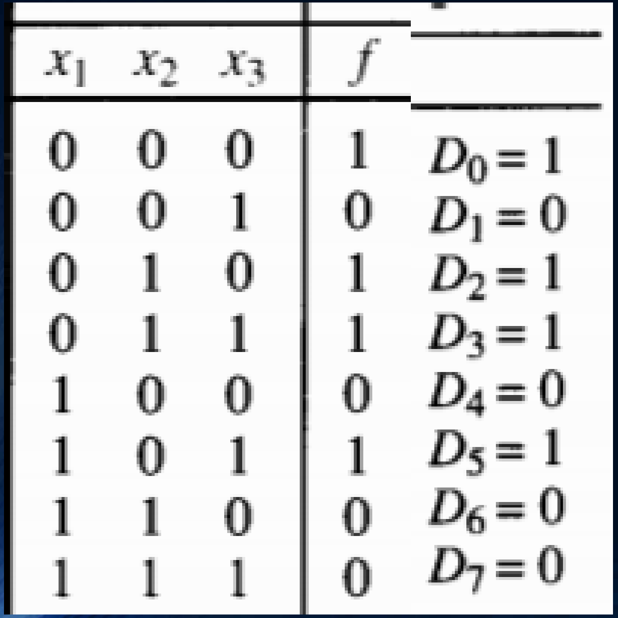


✓Multiplexers

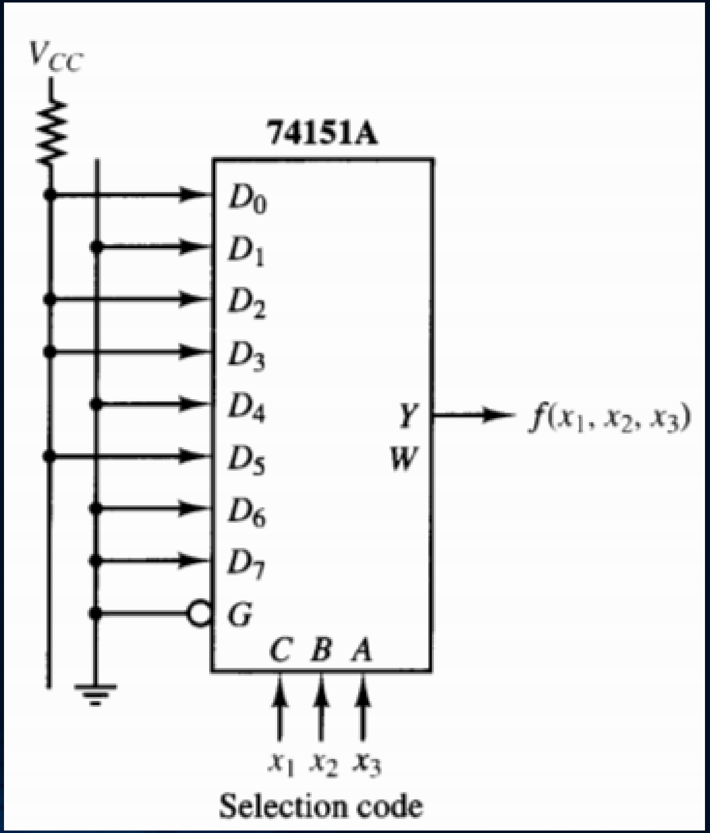
➢ **Implementing logic functions**

✓ Ex. Implement the function using 74151A (8×1 MUX)

$$f(x_1,x_2,x_3)=\sum m(0,2,3,5)$$



x_1	x_2	x_3	f	
0	0	0	1	$D_0=1$
0	0	1	0	$D_1=0$
0	1	0	1	$D_2=1$
0	1	1	1	$D_3=1$
1	0	0	0	$D_4=0$
1	0	1	1	$D_5=1$
1	1	0	0	$D_6=0$
1	1	1	0	$D_7=0$



✓Multiplexers

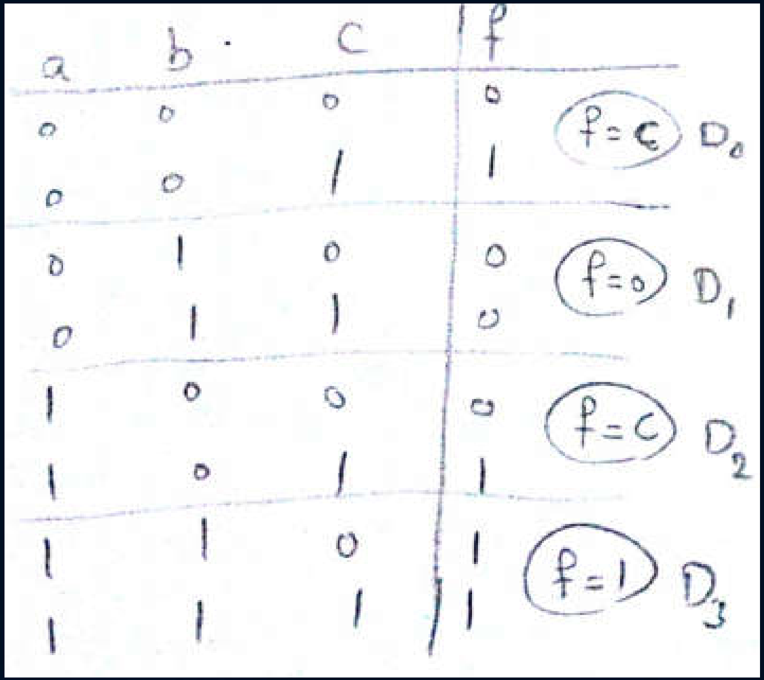
➢ **Implementing logic functions**

✓ Ex. Implement the function using a 4×1 MUX

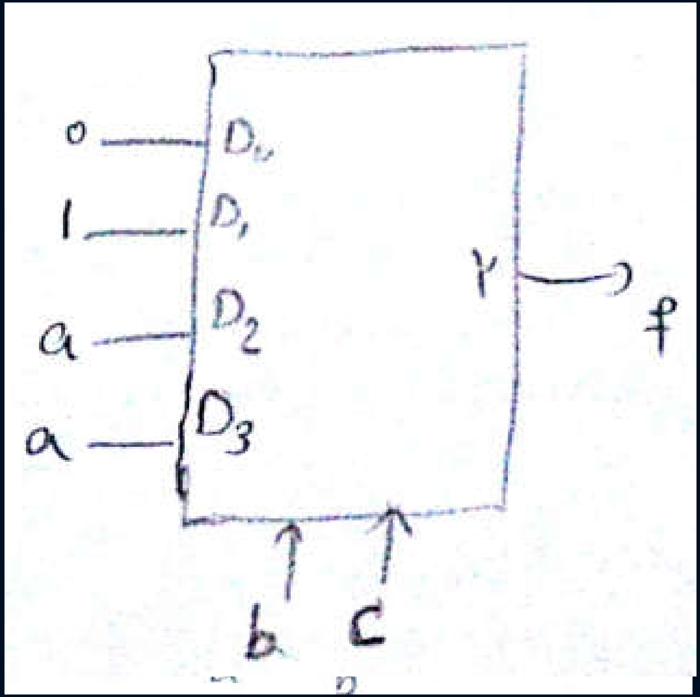
$$f(a,b,c) = ab + \bar{b}c$$

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

$$= \sum m(1,5,6,7)$$



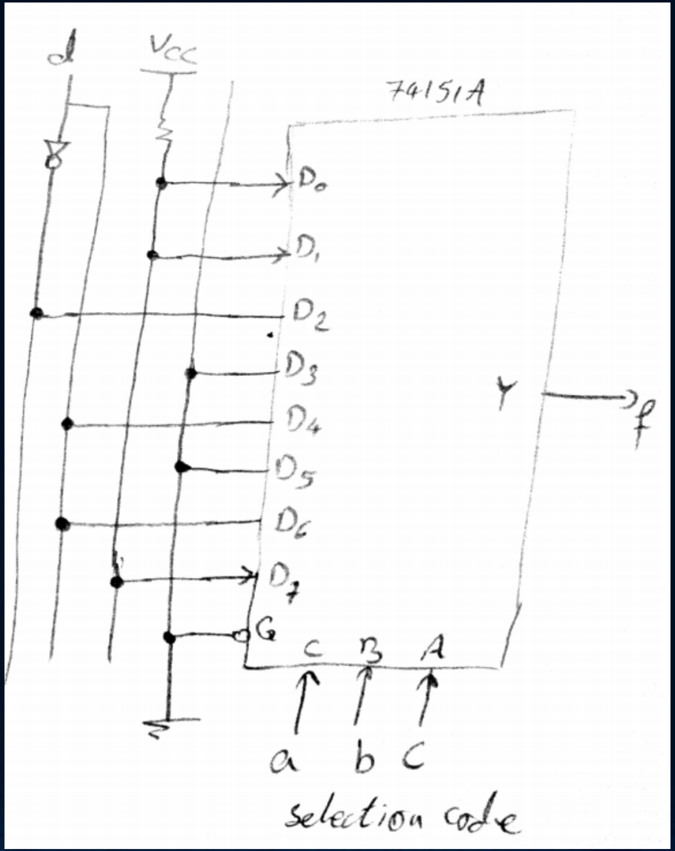
a	b	c	f	
0	0	0	0	$f=c$ D_0
0	0	1	1	
0	1	0	0	$f=0$ D_1
0	1	1	0	
1	0	0	0	$f=c$ D_2
1	0	1	1	
1	1	0	1	$f=1$ D_3
1	1	1	1	

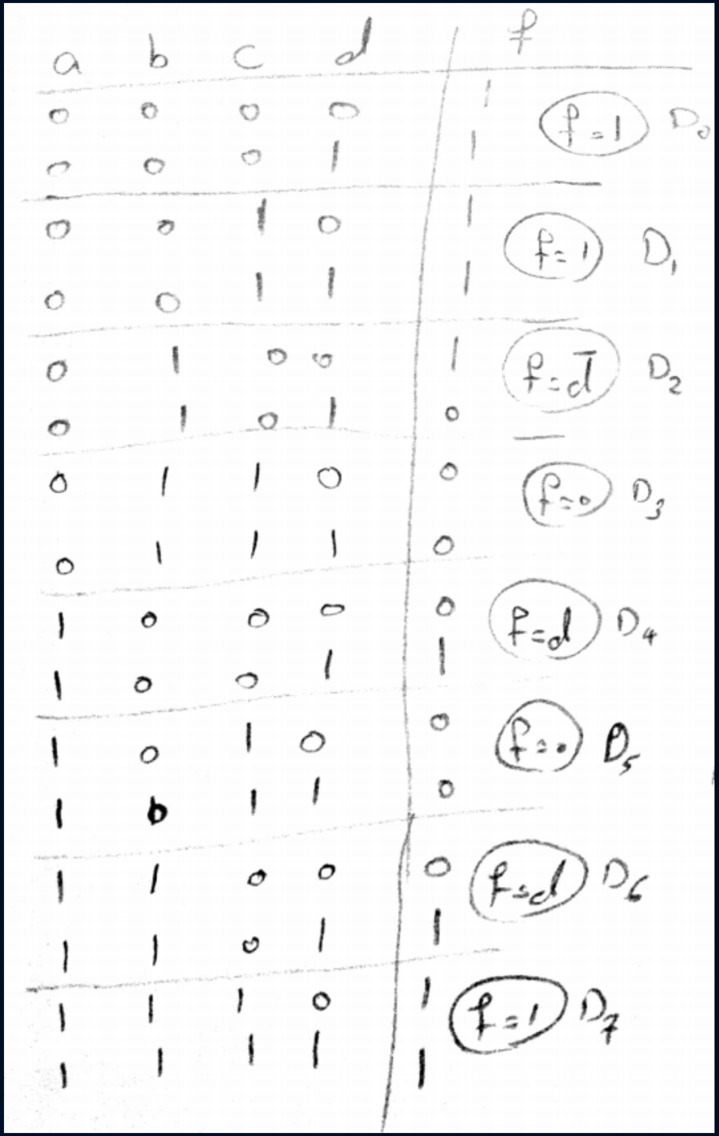


✓Multiplexers

➢ Ex. Implement the function using a 8×1 MUX

$$f(X_1, X_2, X_3, X_4) = \sum m(0,1,2,3,4,9,13,14,15)$$



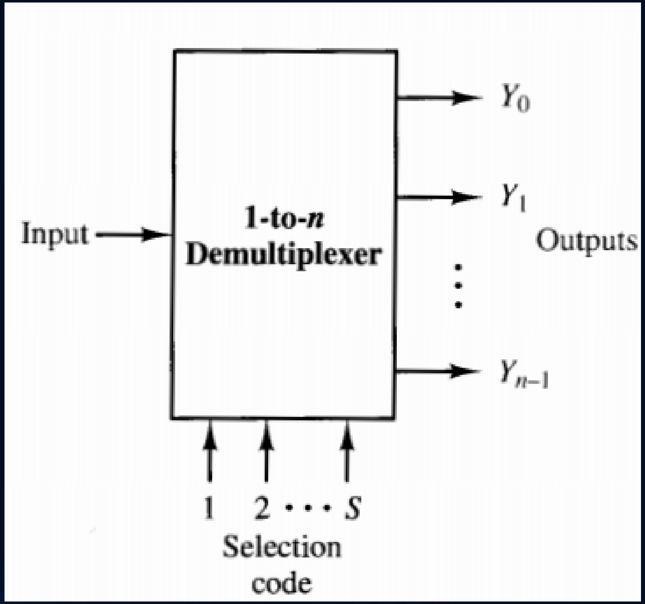


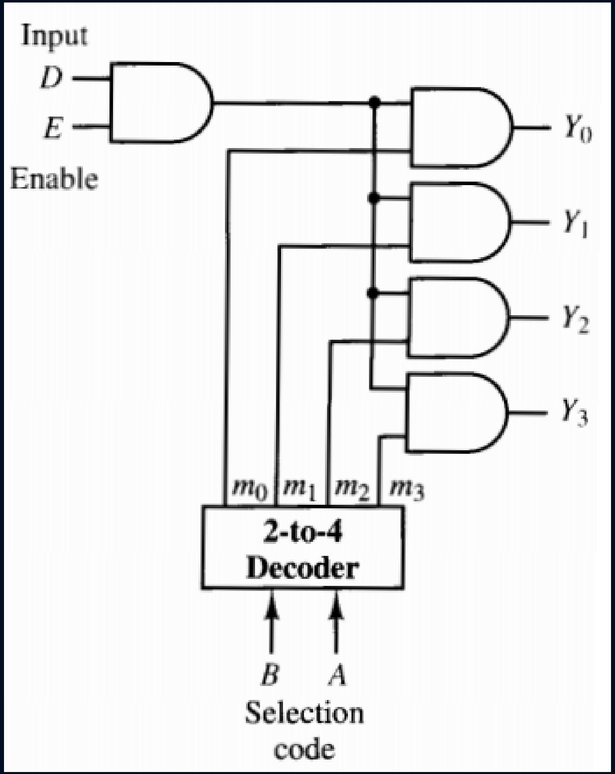
a	b	c	d	f		
0	0	0	0	1	$f=1$	D_0
0	0	0	1	1		
0	0	1	0	1	$f=1$	D_1
0	0	1	1	1		
0	1	0	0	1	$f=\bar{d}$	D_2
0	1	0	1	0		
0	1	1	0	0	$f=0$	D_3
0	1	1	1	0		
1	0	0	0	0	$f=d$	D_4
1	0	0	1	1		
1	0	1	0	0	$f=0$	D_5
1	b	1	1	0		
1	1	0	0	0	$f=d$	D_6
1	1	0	1	1		
1	1	1	0	1	$f=1$	D_7
1	1	1	1	1		

✓Demultiplexers

- *Demultiplexer (Data Distributor)* connects a single input line to one of n output lines
- The selection code is used to generate a minterm of s variables
- That minterm gates the input data to the proper output terminal
- The enable signal (E) controls the operation of the circuit

$$Y_i = (m_i D) E$$

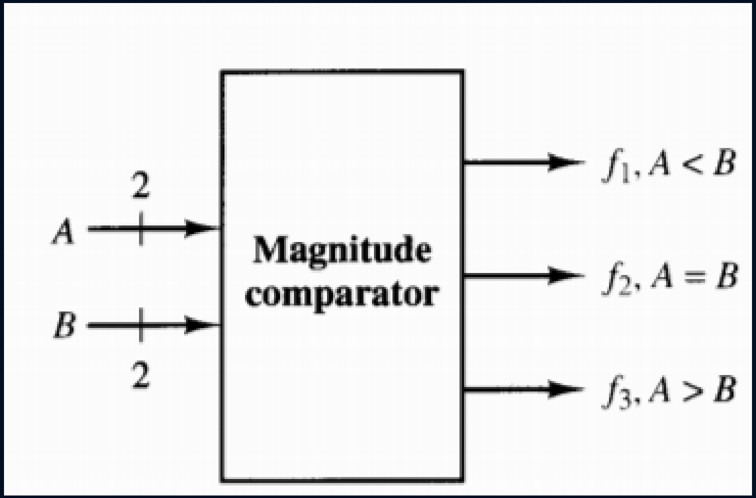




✓Magnitude Comparator

- ❖ Perform a magnitude comparison of two binary numbers A and B
- ❖ The outcome of a comparison is specified by three binary variables that indicate whether A>B, A<B or A=B
- ❖ So, the comparator will generate three output signal as:

$$
\begin{aligned}
f_1 &= 1, \quad \text{if } A < B \\
f_2 &= 1, \quad \text{if } A = B \\
f_3 &= 1, \quad \text{if } A > B
\end{aligned}
$$



✓Magnitude Comparator

❖ **Design:**

$$A = A_3\, A_2\, A_1\, A_0$$
$$B = B_3 B_2 B_1 B_0$$

➢ $f_{A=B}$:

$$x_i = A_i \quad B_i = A_i B_i + \overline{A_i}\,\overline{B_i} \;\;, i = 0,1,2,3, \ldots$$

$$\boldsymbol{f_{A=B} = x_0 . x_1 . x_2 . x_3}$$

➢ $f_{A>B}$:

$A_3 > B_3 \quad or$
$A_3 = B_3 \quad and \quad A_2 > B_2 \quad or$
$A_3 = B_3 \quad and \quad A_2 = B_2 \quad and \quad A_1 > B_1 \quad or$
$A_3 = B_3 \quad and \quad A_2 = B_2 \quad and \quad A_1 = B_1 \quad and \quad A_0 > B_0$

$$\boldsymbol{f_{A>B} = A_3\overline{B_3} + x_3A_2\overline{B_2} + x_3x_2A_1\overline{B_1} + x_3x_2x_1A_0\overline{B_0}}$$

3-level

➢ $f_{A<B}$:

$$\boldsymbol{f_{A<B} = \overline{A_3}B_3 + x_3\overline{A_2}B_2 + x_3x_2\overline{A_1}B_1 + x_3x_2x_1\overline{A_0}B_0}$$

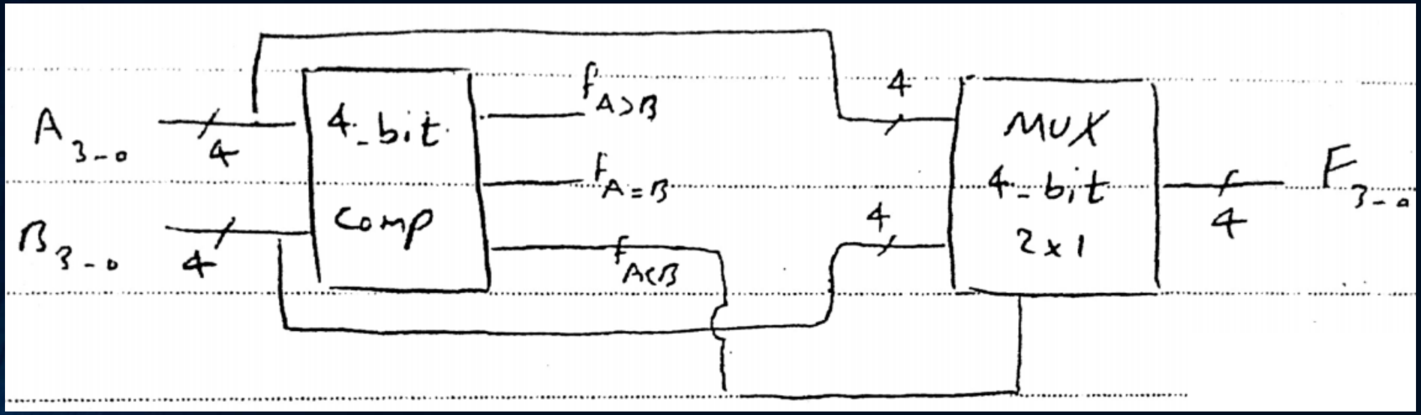
3-level

$$\boldsymbol{f_{A<B} = f_{A>B} \quad f_{A=B}}$$

4-level

✓ مثال

❖ مداری طراحی کنید که دارای دو ورودی BCD بوده و عدد بزرگتر را در خروجی نشان دهد.



❖ ۳ مرحله تاخیر برای مقایسه کننده و ۲ مرحله تاخیر برای مالتی پلکسر داریم که در مجموع ۵ مرحله تاخیر خواهیم داشت.