

به نام خدا

# طراحی سیستم های دیجیتال ۱

## فصل چهارم

### مدارهای منطقی ترکیبی

## Combinational Logic Circuits

## ✓ 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 ✓

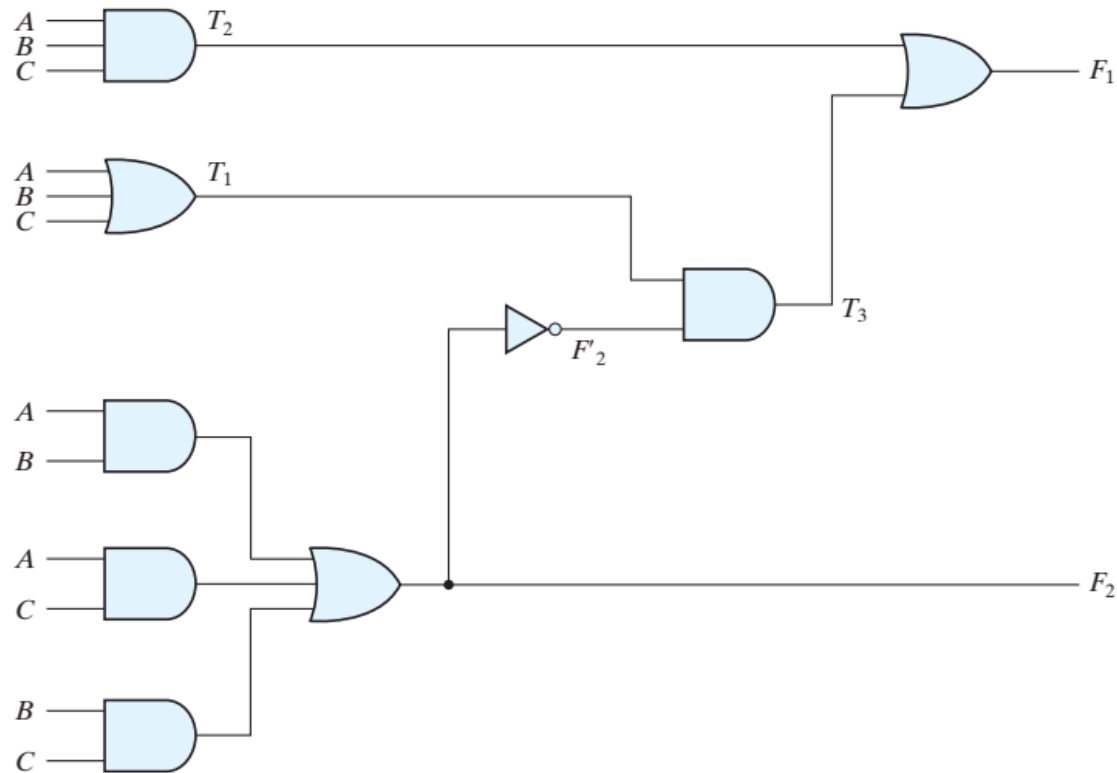
Multiplexer ✓

## ✓ 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:



$$F_2 = AB + AC + BC$$

$$T_1 = A + B + C$$

$$T_2 = ABC$$

*I*

<i>A</i>	<i>B</i>	<i>C</i>	<i>F</i> <sub>2</sub>	<i>F</i> ' <sub>2</sub>	<i>T</i> <sub>1</sub>	<i>T</i> <sub>2</sub>	<i>T</i> <sub>3</sub>	<i>F</i> <sub>1</sub>
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

*ABC*

## ✓ 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

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

❖ مثالی از کد گردانی (Code Conversion):

طراحی یک مدار برای تبدیل کد BCD به

Excess-3

## ✓ مدارهای منطقی ترکیبی

❖ مثالی از کد گردانی (Code Conversion):

طراحی یک مدار برای تبدیل کد BCD به

Excess-3

AB \ CD		C			
		00	01	11	10
A	00	$m_0$ 1	$m_1$	$m_3$	$m_2$ 1
	01	$m_4$ 1	$m_5$	$m_7$	$m_6$ 1
	11	$m_{12}$ X	$m_{13}$ X	$m_{15}$ X	$m_{14}$ X
	10	$m_8$ 1	$m_9$	$m_{11}$ X	$m_{10}$ X
		D			

$z = D'$

AB \ CD		C			
		00	01	11	10
A	00	$m_0$ 1	$m_1$	$m_3$ 1	$m_2$
	01	$m_4$ 1	$m_5$	$m_7$ 1	$m_6$
	11	$m_{12}$ X	$m_{13}$ X	$m_{15}$ X	$m_{14}$ X
	10	$m_8$ 1	$m_9$	$m_{11}$ X	$m_{10}$ X
		D			

$y = CD + C'D'$

AB \ CD		C			
		00	01	11	10
A	00	$m_0$	$m_1$ 1	$m_3$ 1	$m_2$ 1
	01	$m_4$ 1	$m_5$	$m_7$	$m_6$ 1
	11	$m_{12}$ X	$m_{13}$ X	$m_{15}$ X	$m_{14}$ X
	10	$m_8$	$m_9$ 1	$m_{11}$ X	$m_{10}$ X
		D			

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

AB \ CD		C			
		00	01	11	10
A	00	$m_0$	$m_1$	$m_3$	$m_2$
	01	$m_4$	$m_5$ 1	$m_7$ 1	$m_6$ 1
	11	$m_{12}$ X	$m_{13}$ X	$m_{15}$ X	$m_{14}$ X
	10	$m_8$ 1	$m_9$ 1	$m_{11}$ X	$m_{10}$ X
		D			

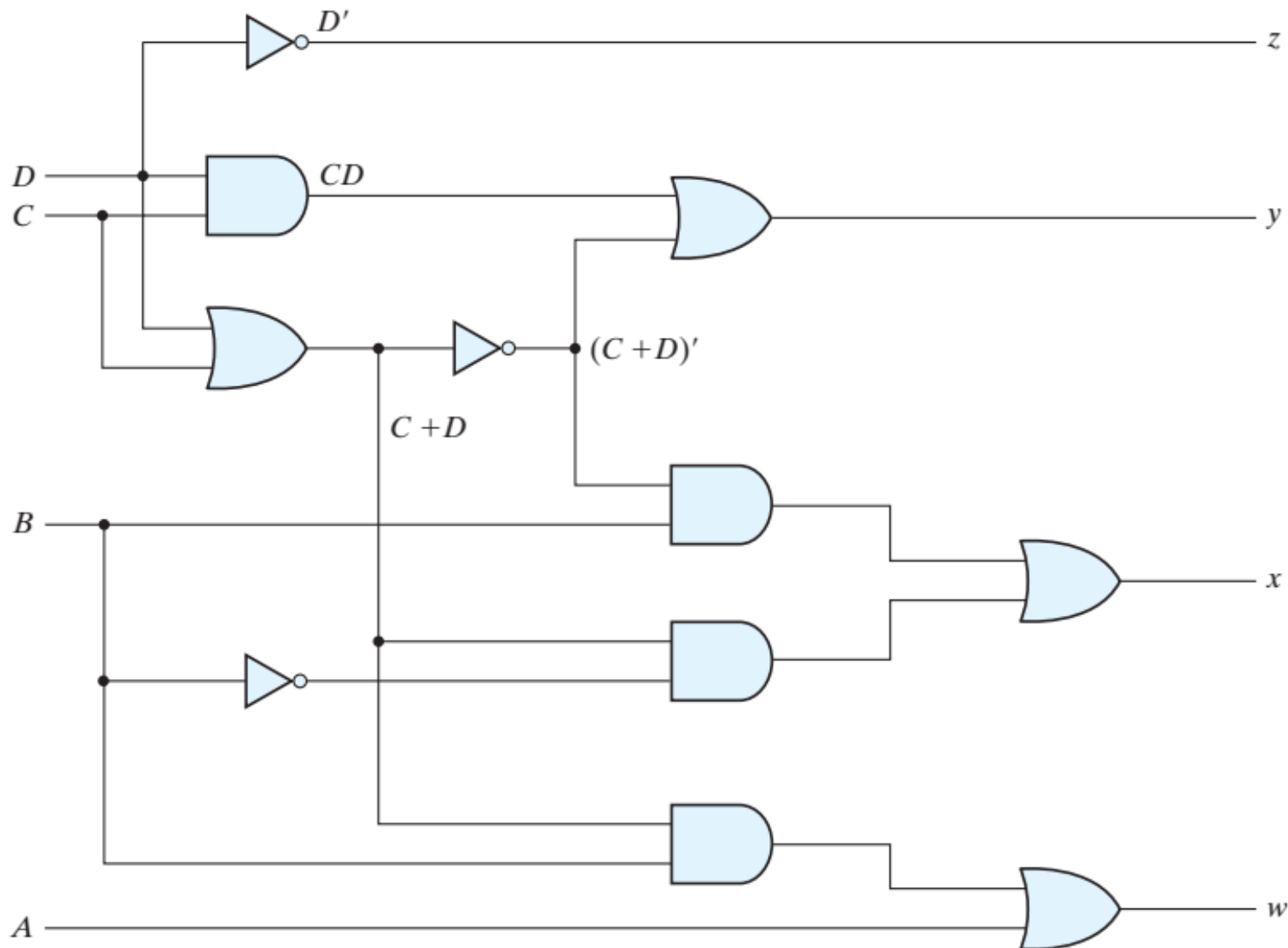
$w = A + BC + BD$

## ✓ مدارهای منطقی ترکیبی

❖ مثالی از کد گردانی :

طراحی یک مدار برای تبدیل

کد BCD به Excess-3

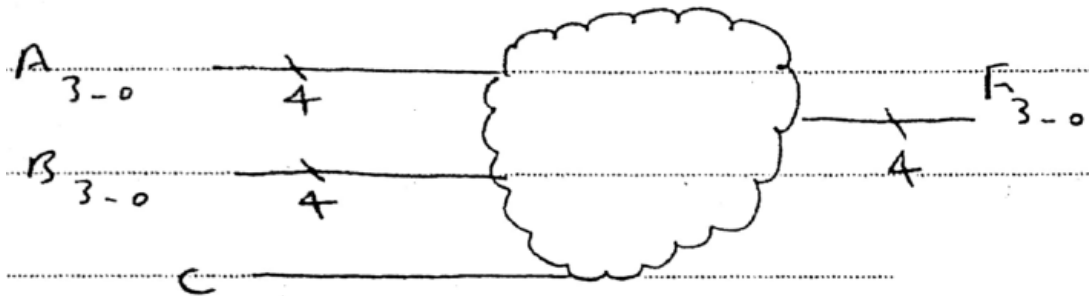


## ✓ مدارهای منطقی ترکیبی

❖ مثال: مداری طراحی کنید که اگر در آن  $C=0$  شد، چهار ورودی A نظیر به نظیر به چهار خروجی F منتقل

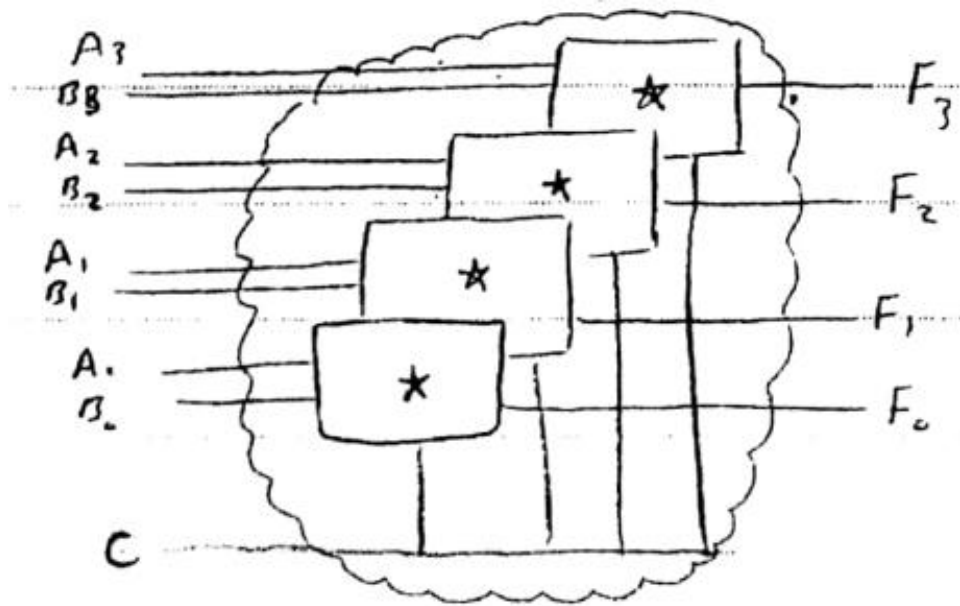
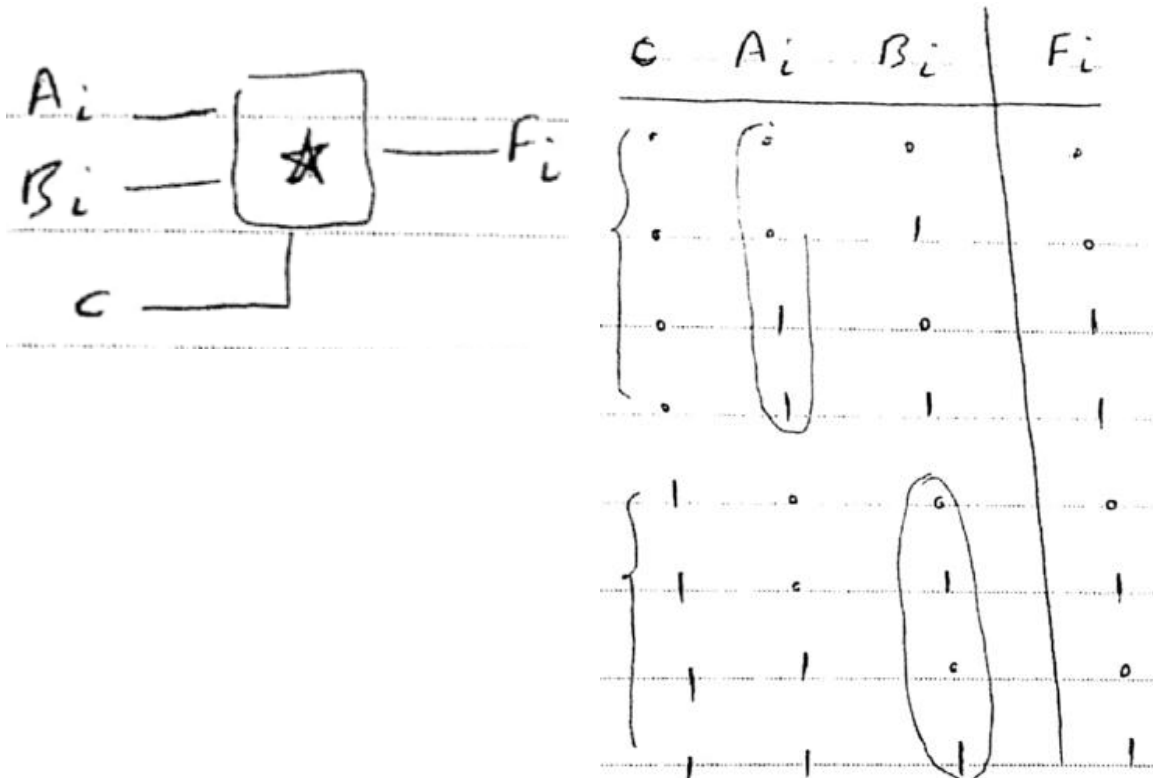
شود. در غیراینصورت چهار ورودی B به چهار خروجی F

منتقل شود.



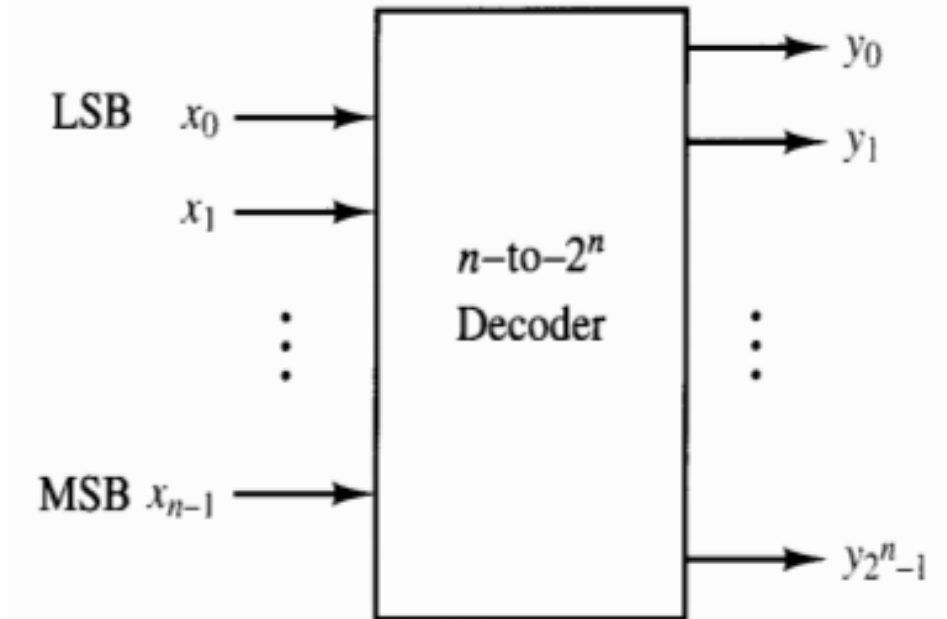
$$F_i = \sum m(2, 3, 5, 7)$$

$$F_i = C'A_i + C B_i$$



## ✓ 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

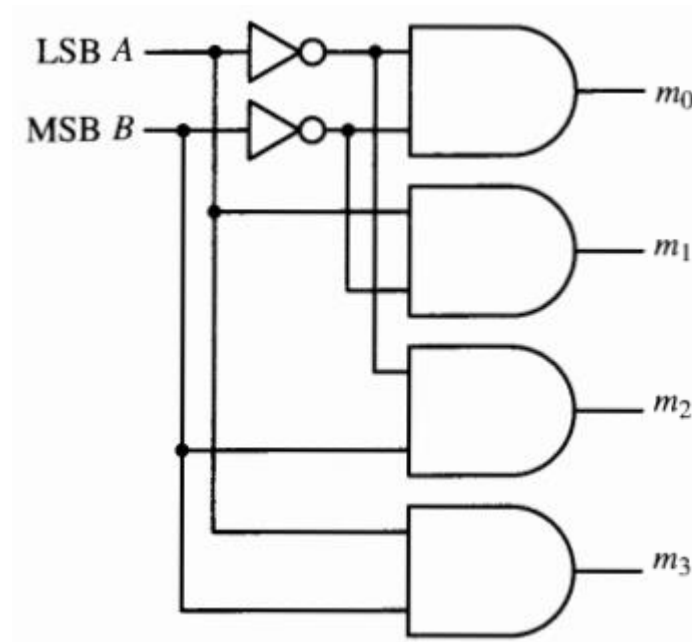




## ✓ Decoders

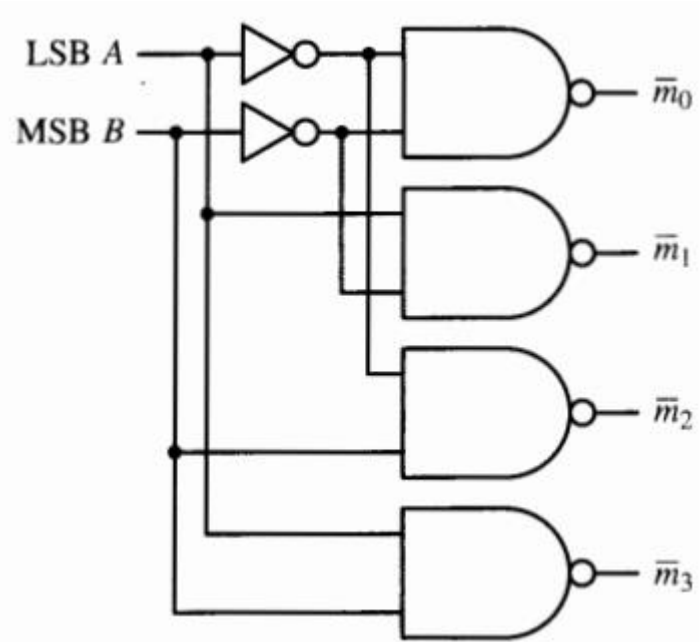
### ➤ 2-to-4 decoder

- ❖ We can consider the  $n$ -to- $2^n$  decoder as simply a **minterm generator**
- ❖ An input combination of  $BA=00$  selects the  $m_0$  output line,  $BA=01$  selects the  $m_1$  and so on.

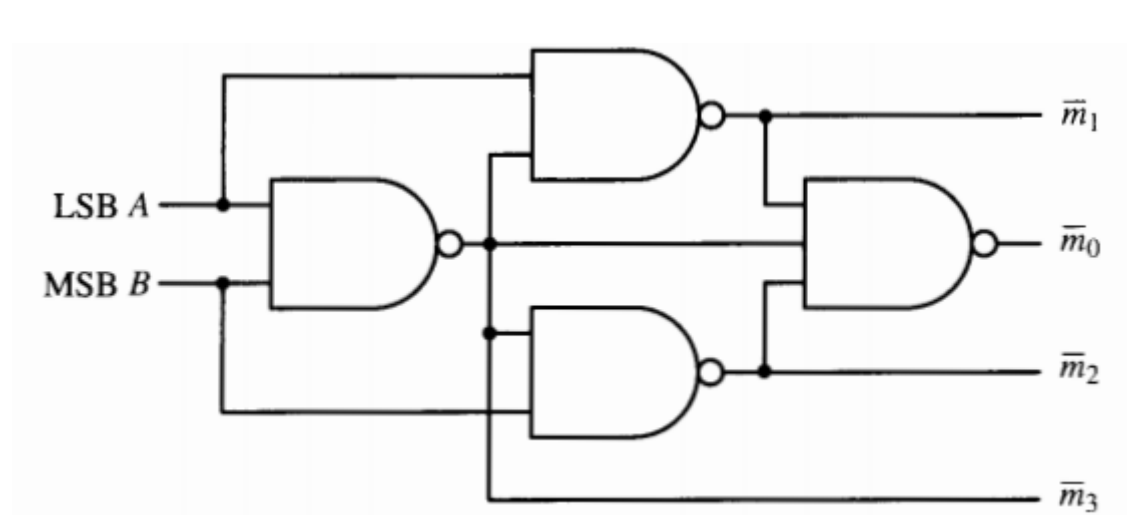


$$\begin{aligned}m_0 &= \bar{B}\bar{A} \\m_1 &= \bar{B}A \\m_2 &= B\bar{A} \\m_3 &= BA\end{aligned}$$

## ✓ Decoders



Using only NAND gates

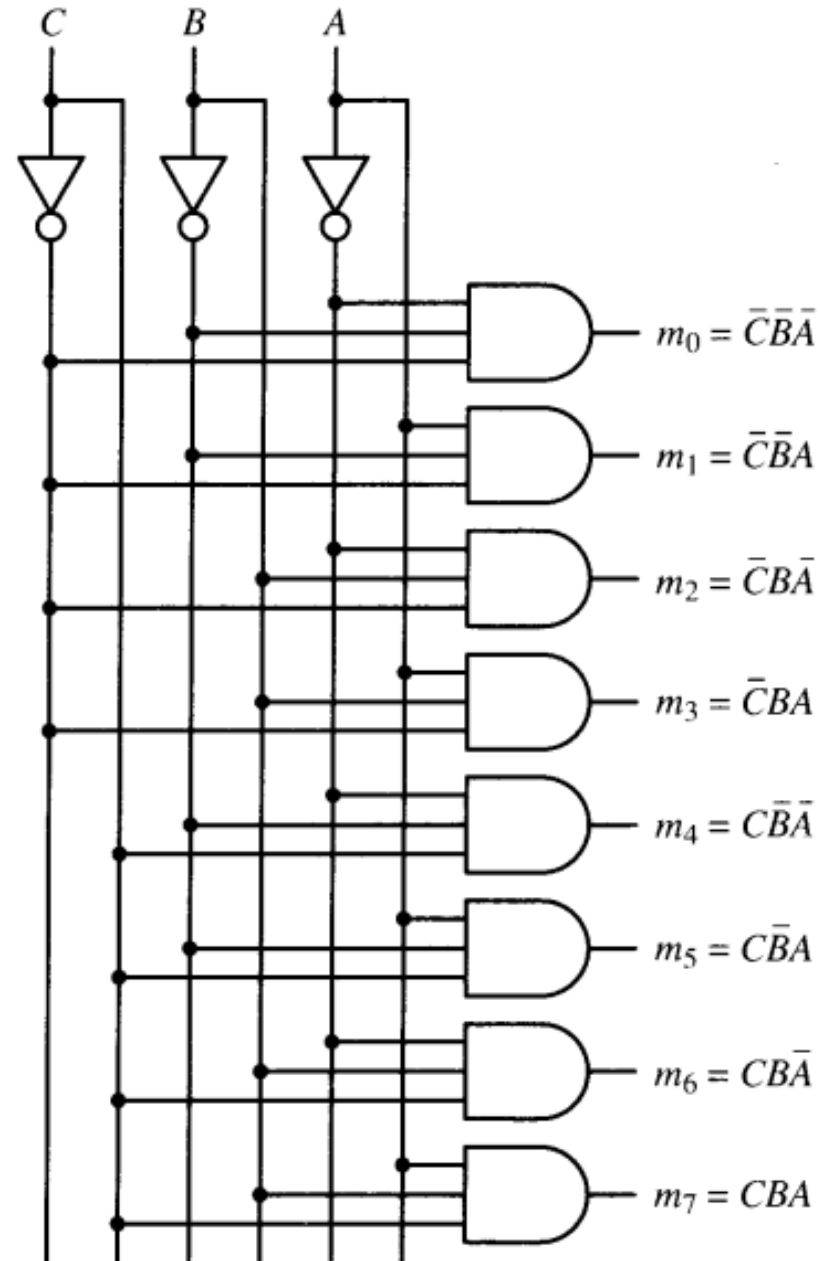


Using only NAND gates  
with no inverters

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

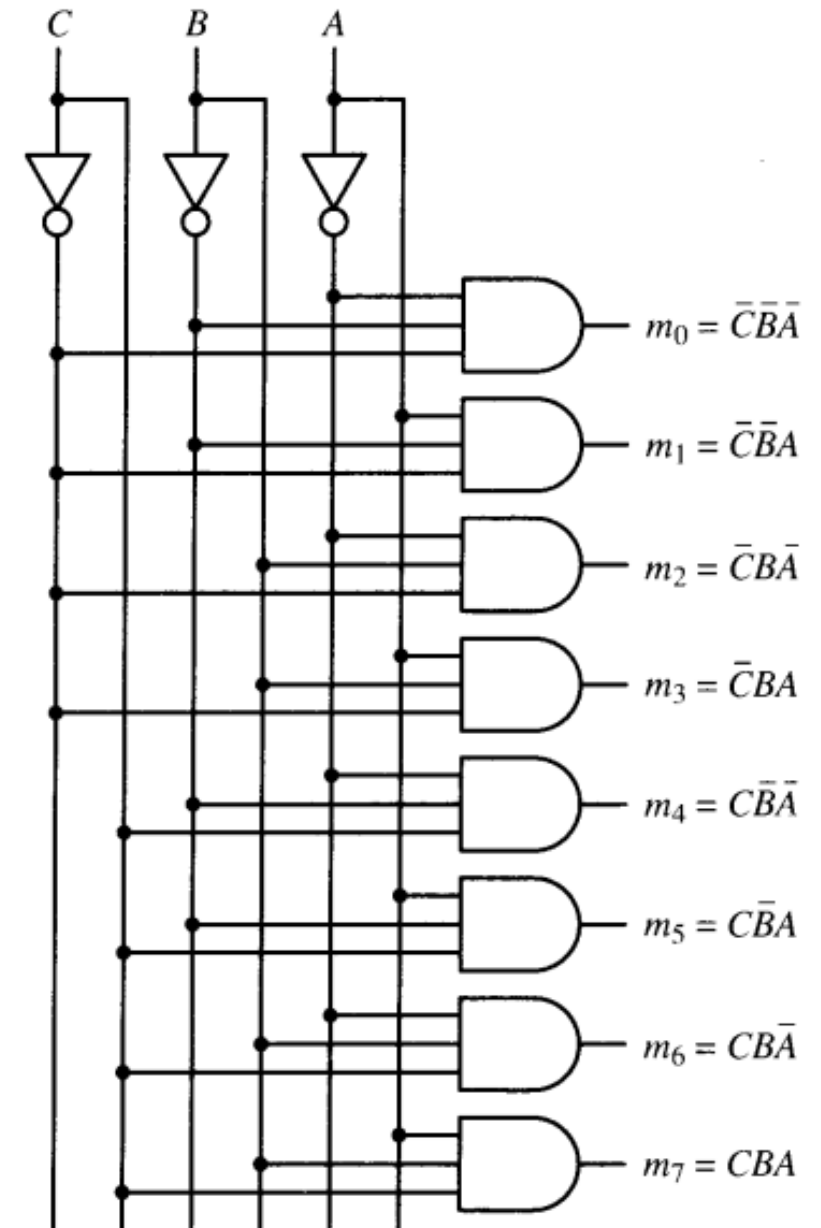
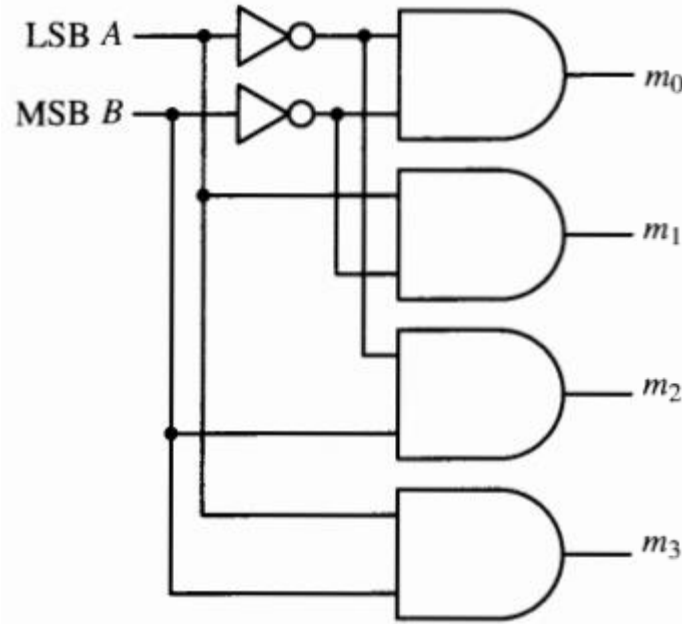
## ✓ Decoders

### 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



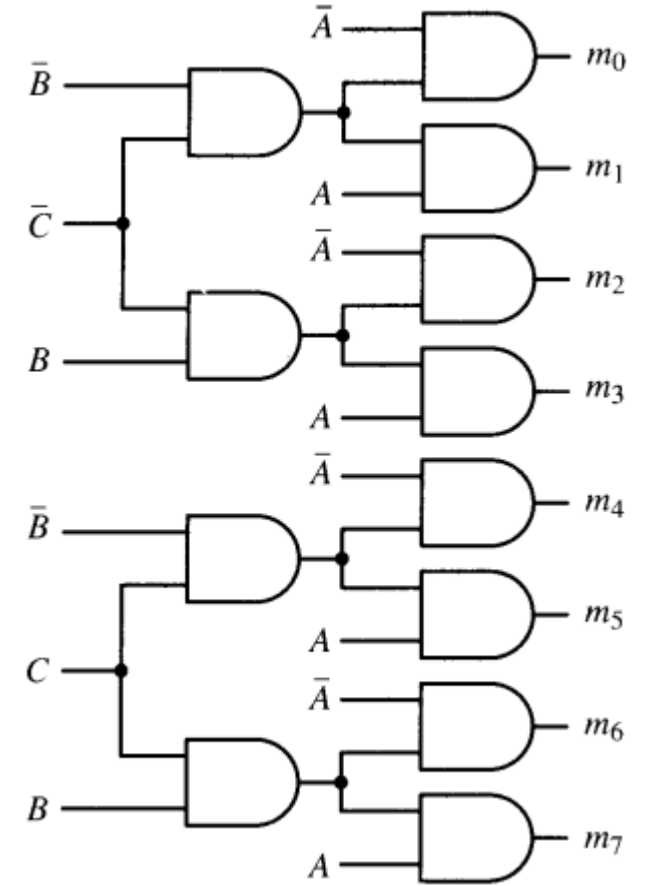
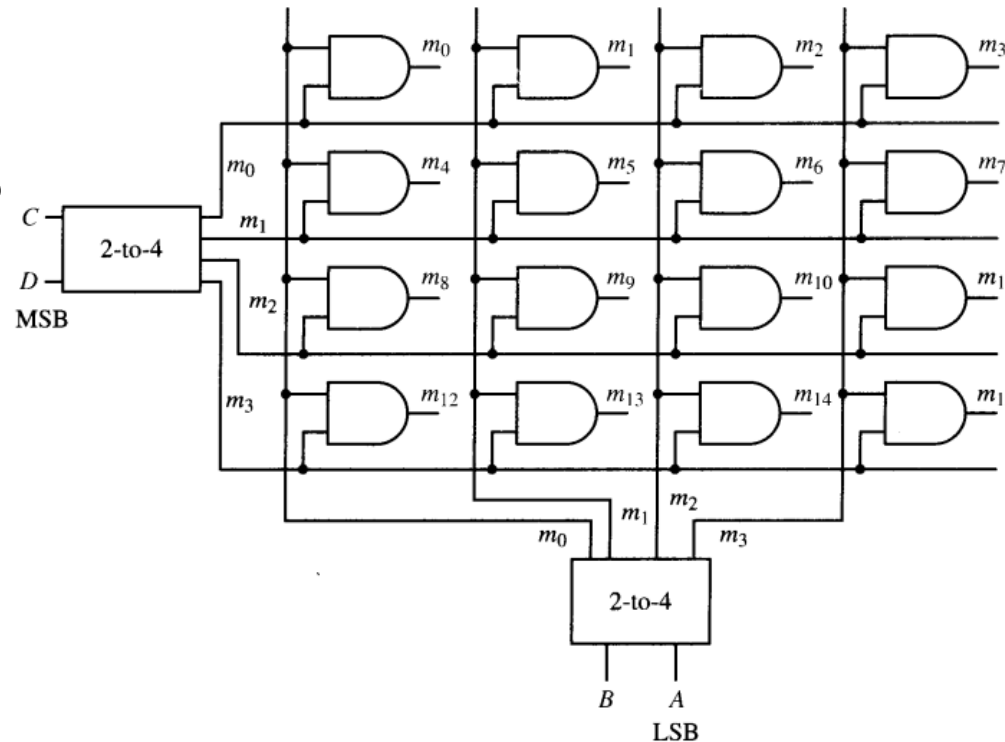
## ✓ Decoders

❖ **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

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

- ✓ 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  $M_i = \bar{m}_i$

$$f(A, B, \dots, Z) = M_i \cdot M_j \cdot \dots \cdot M_k \longrightarrow \text{Using a decoder with active low outputs and an AND gate}$$

## ✓ Decoders

- **Example:** Implement the following function

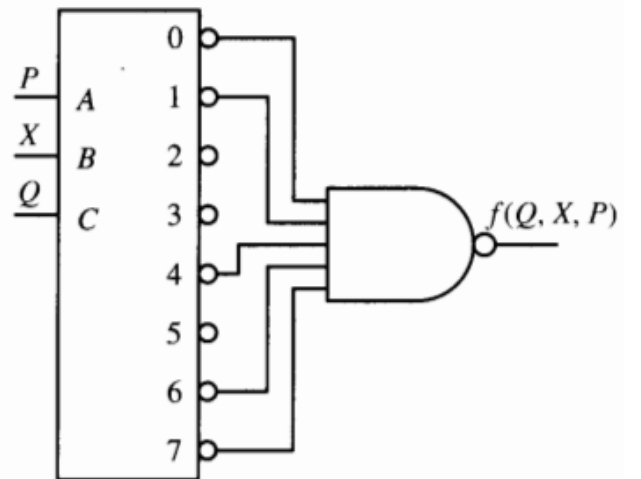
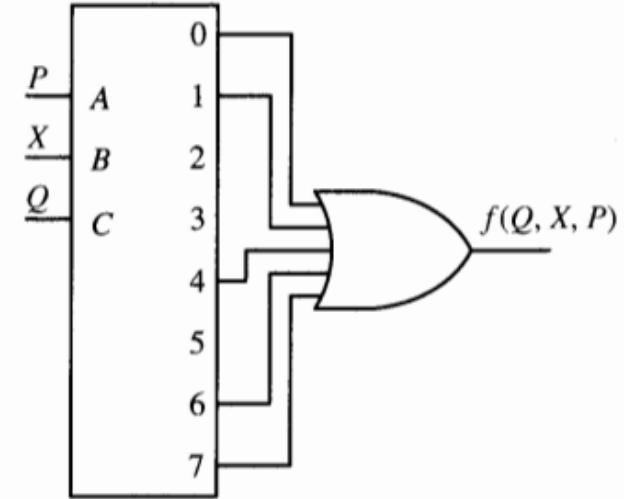
$$\begin{aligned} f(Q, X, P) &= \sum m(0, 1, 4, 6, 7) \\ &= \prod M(2, 3, 5) \end{aligned}$$

- ❖ 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:

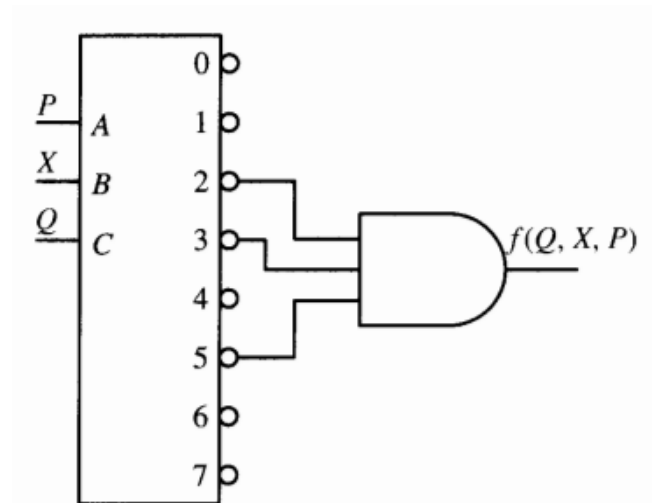
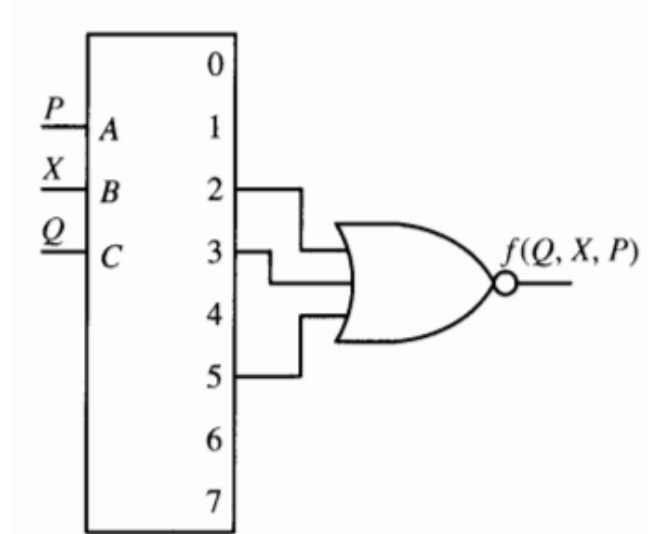
$$\begin{aligned} f(Q, X, P) &= \sum m(0, 1, 4, 6, 7) \\ &= \prod M(2, 3, 5) \end{aligned}$$

❖ 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

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

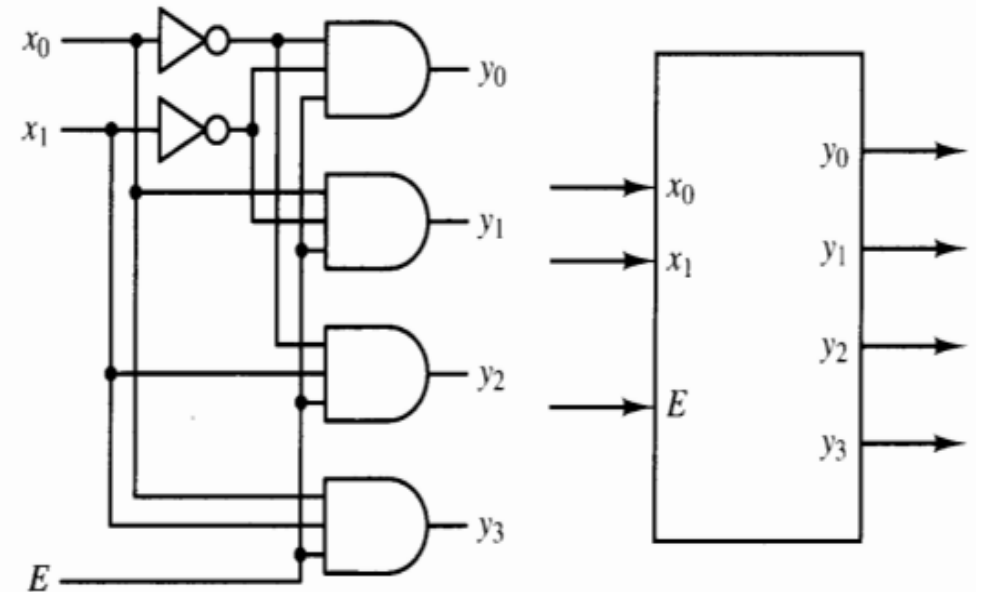




## ✓ Decoders

### ➤ Enable Control Input

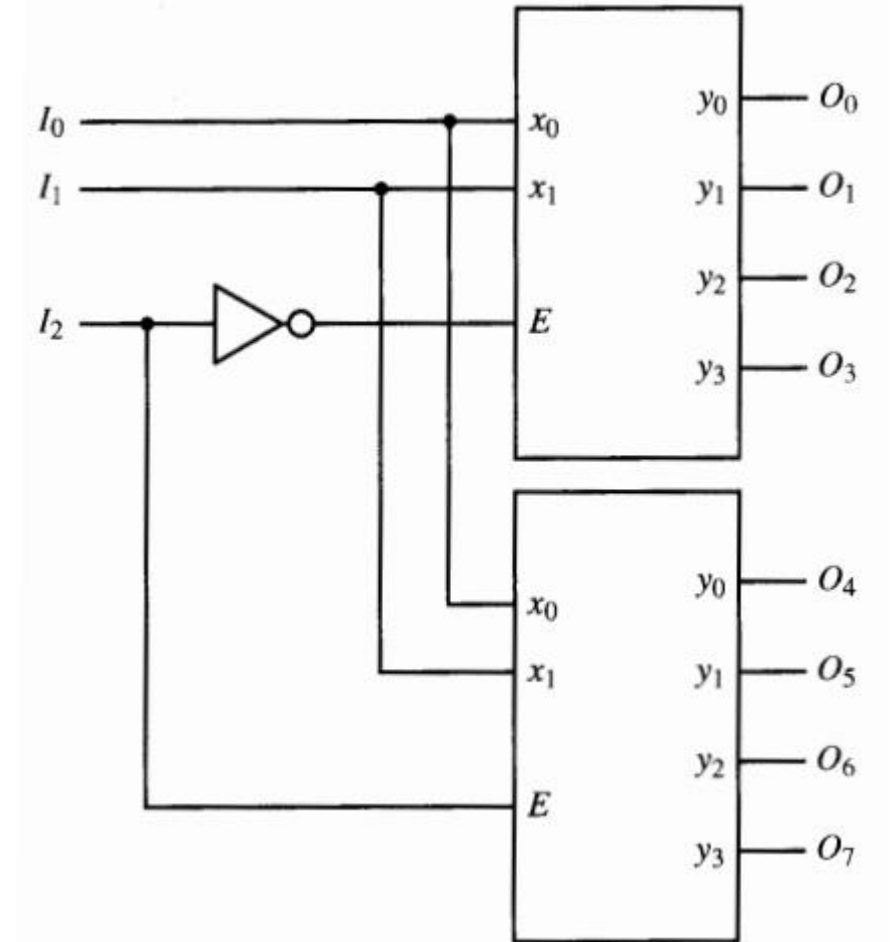
- ❖ Enable Inputs can be used to either enable or disable the designated functions to be performed
- ❖ The decoding function of a decoder is disabled by forcing all its outputs to the inactive state
- ❖ In general, we have  $y_k = m_k E$   
When  $E=0$ , all outputs are forced to 0, whereas for  $E=1$ ,  $y_k$  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

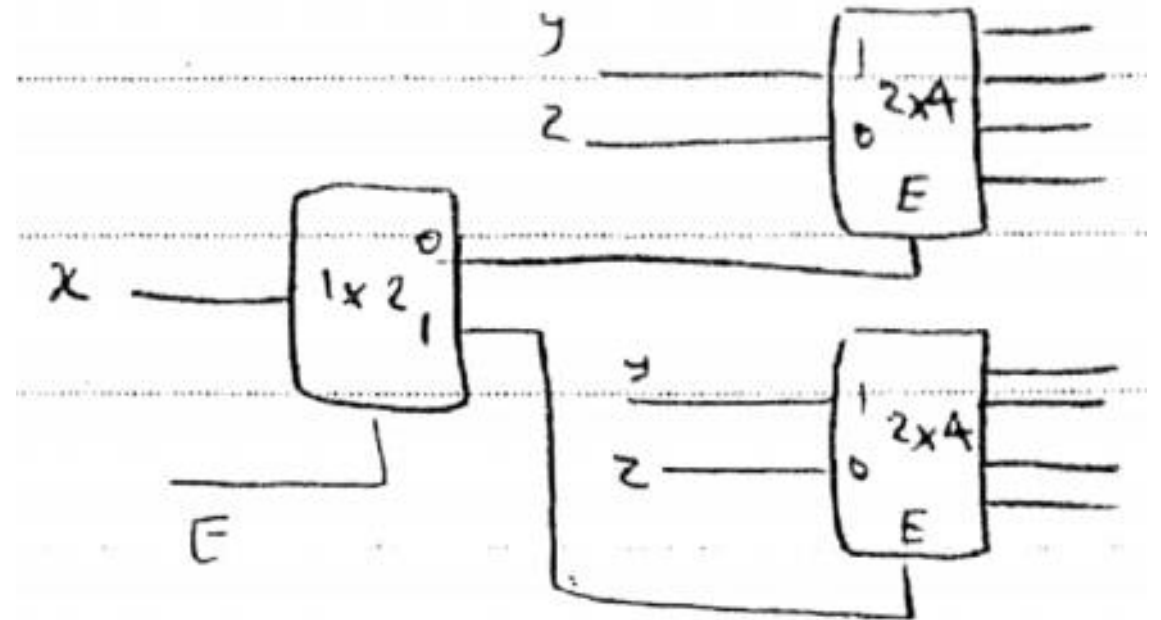
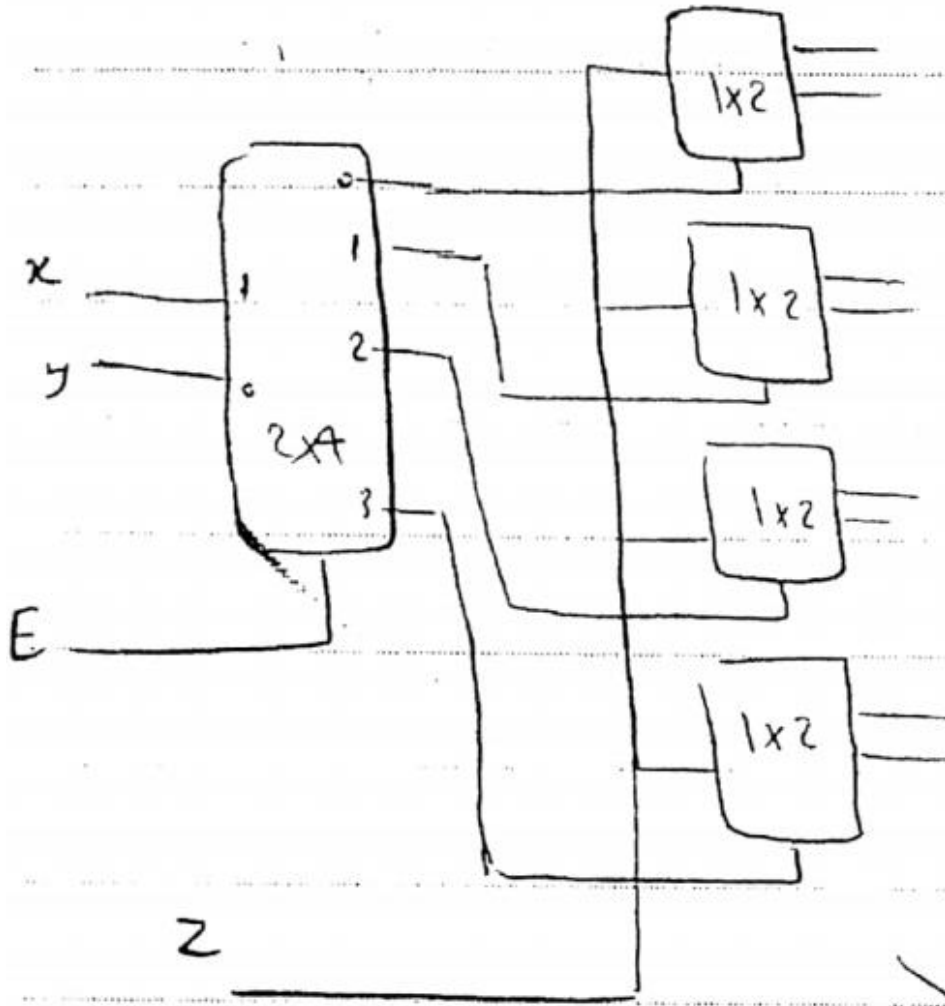
### ➤ Use of 2-to-4 decoder to realize 3-to-8 decoder

- ❖ 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)



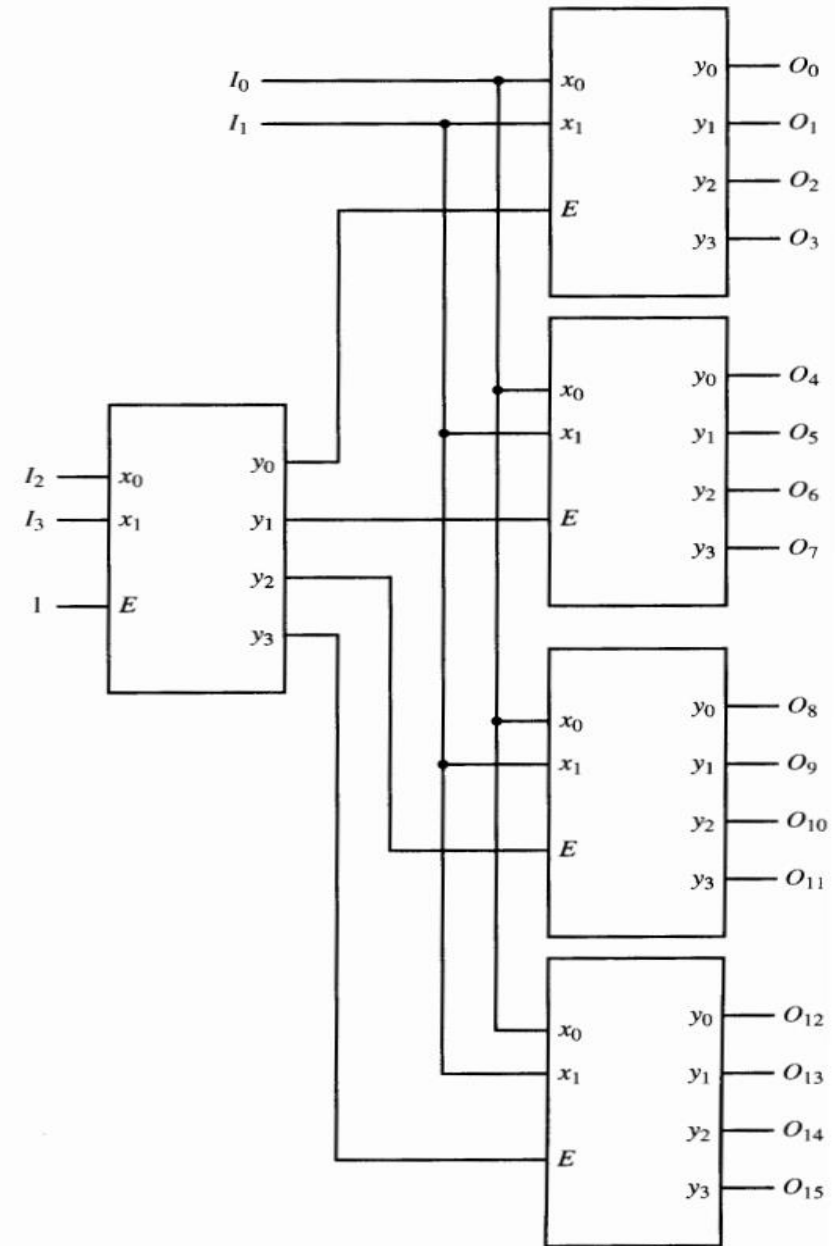
## ✓ Decoders

### ➤ 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)



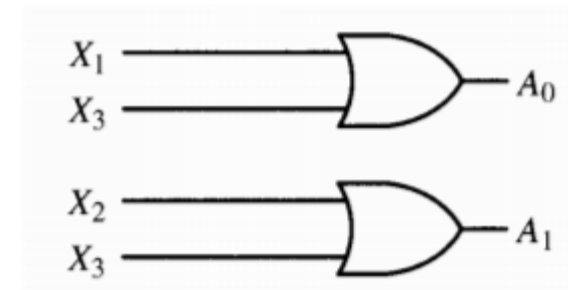
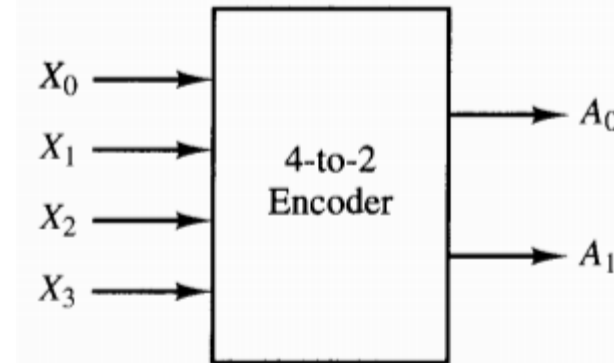
## ✓ Encoders

❖ 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*

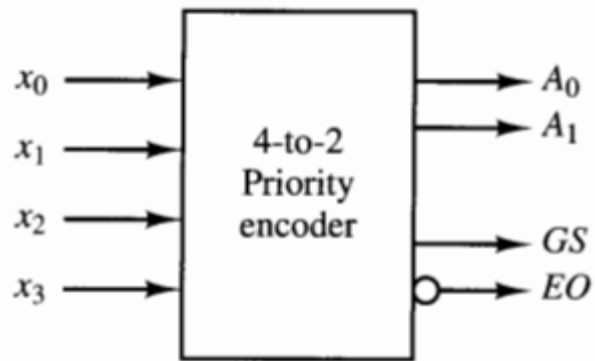
- ✓ Multiple input lines can be active
- ✓ Sends out the binary value of the subscript of the input line with highest priority
- ✓ The highest priority is assigned to the highest subscript
- ✓ If no input line is active, the priority encoder sends out (00)
- ✓ If a single line is active, the encoder 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

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

## ✓ Encoders

### ➤ Encoder Circuit Structure

#### ■ *Priority Encoders*



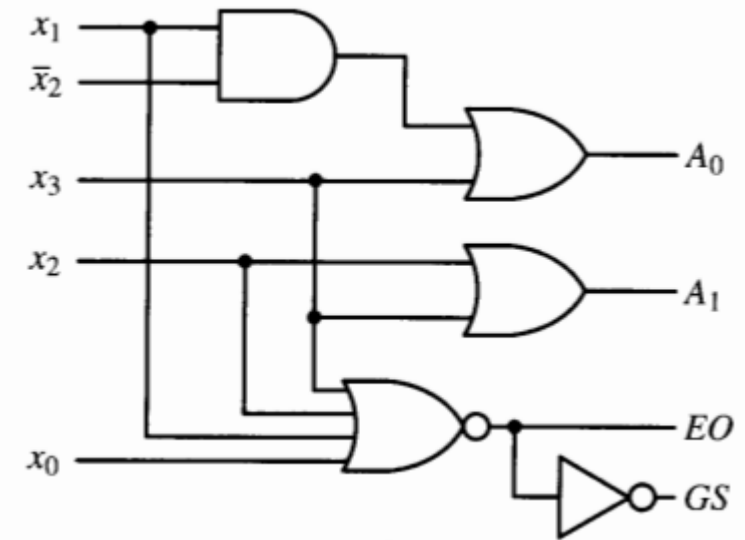
- ✓ Two additional output lines indicate that no input line is active ( $EO=1$ ) and one or more inputs are active ( $GS=1$ )

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

$$A_1 = X_2 + X_3$$

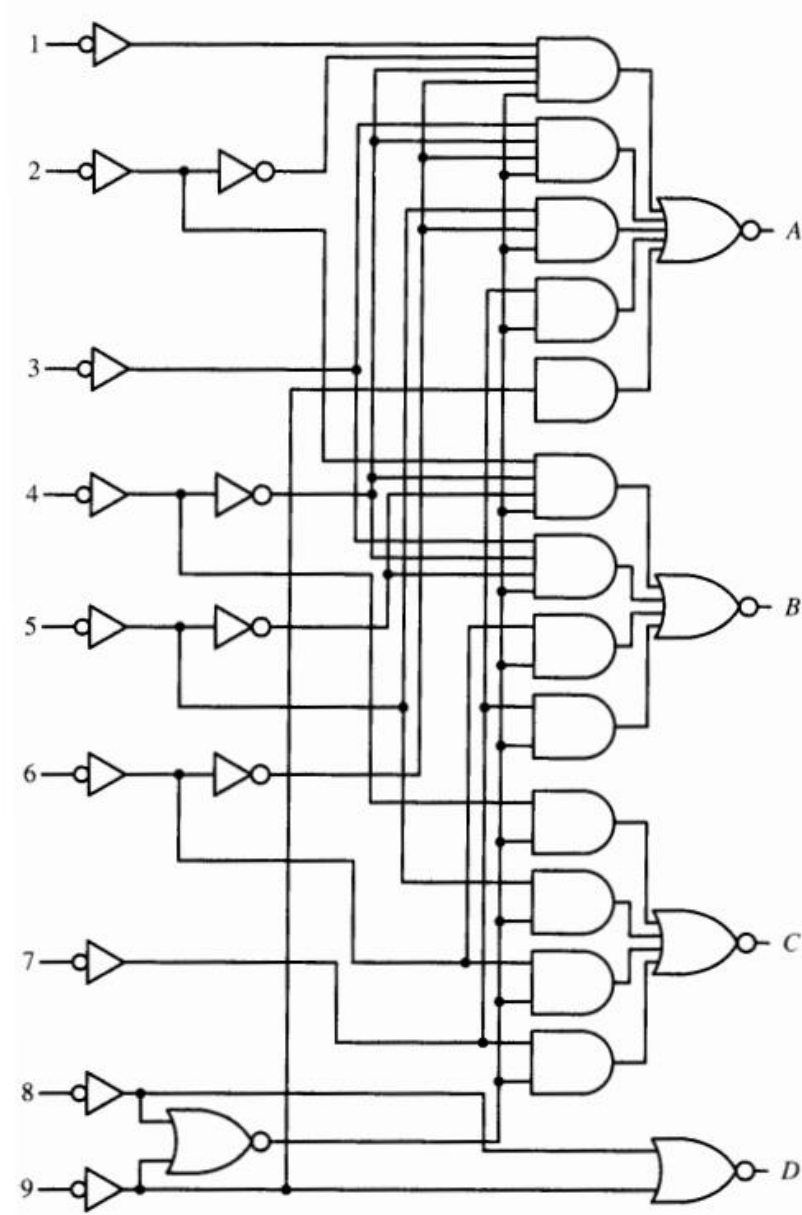
$$A_0 = X_3 + X_1 \bar{X}_2$$

$$EO = \overline{GS} = \overline{X_3 + X_2 + X_1 + X_0}$$

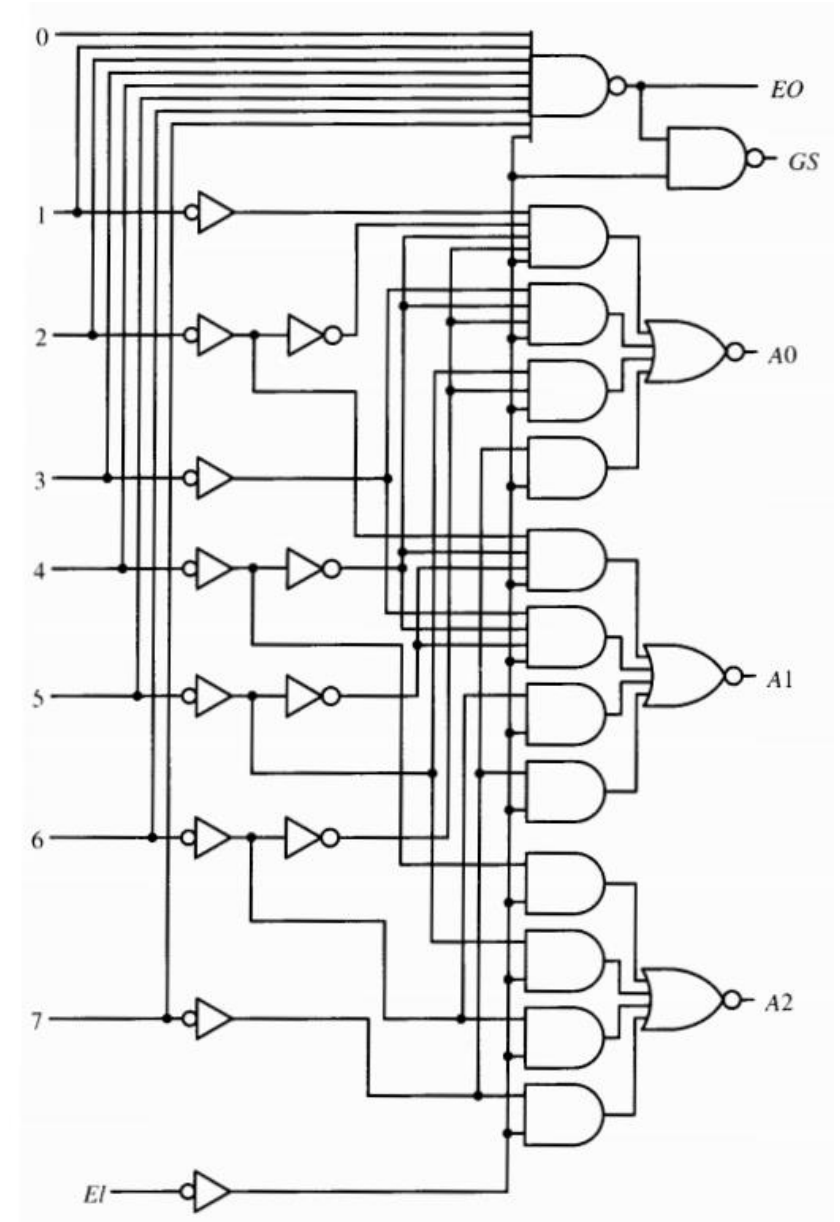


# ✓ Encoders

## ➤ Standard MSI Encoders



74147

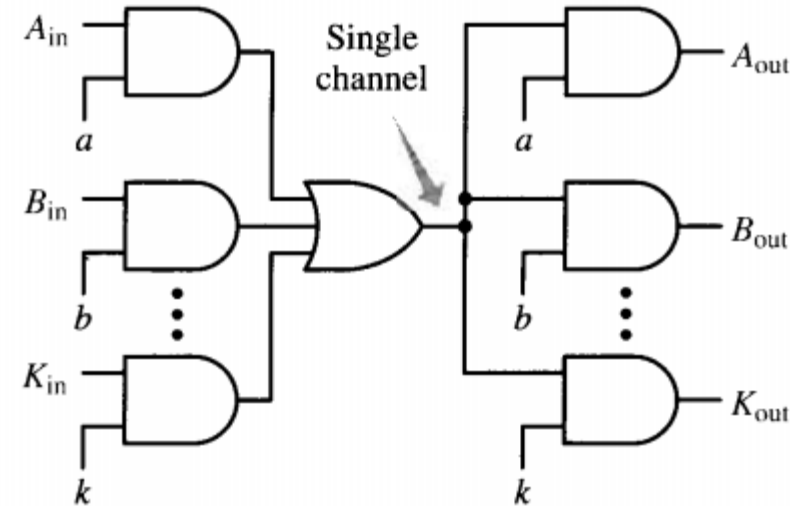
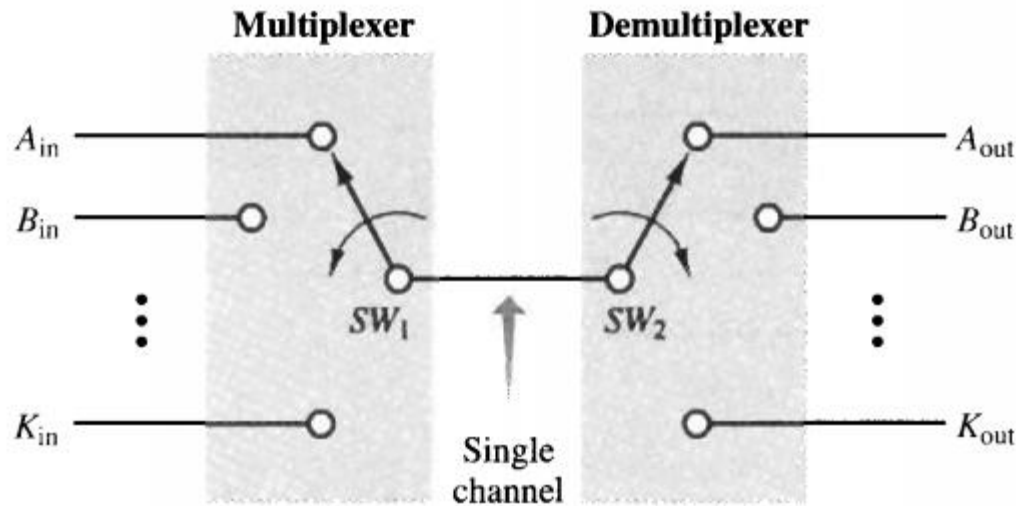


74148



## ✓ Multiplexers

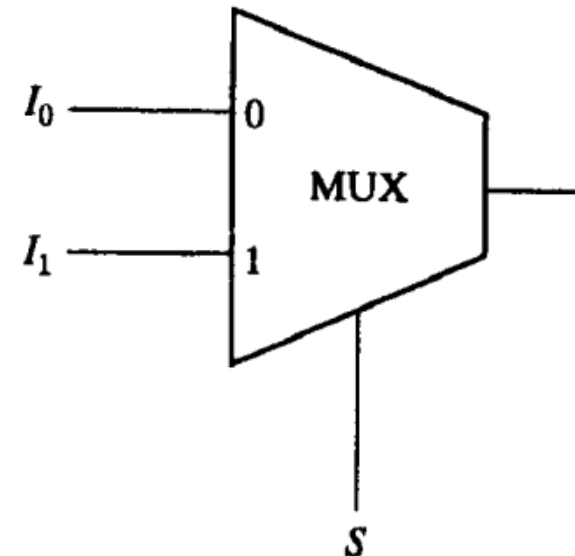
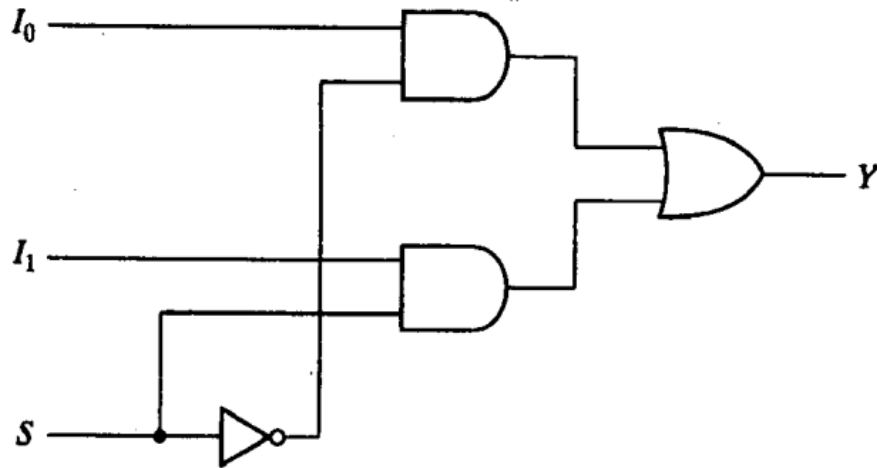
- ❖ *Multiplexers (Data Selectors)* is a modular device that selects one of many input lines to appear on a single output line
- ❖ A *demultiplexer* performs the inverse operation which takes a single input line and routes it to one of several output lines



# ✓ Multiplexers

## ➤ Multiplexer Circuit Structure

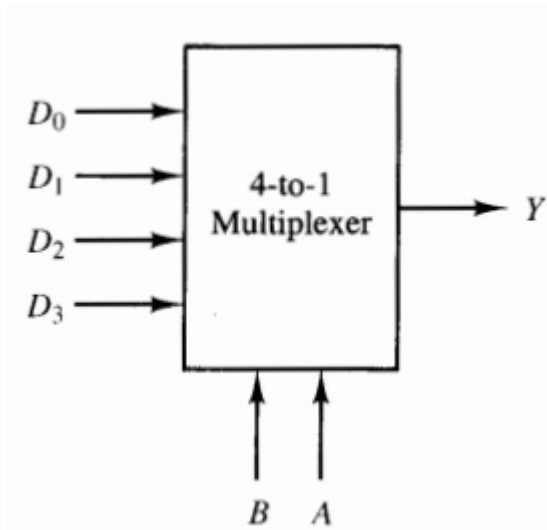
- ✓ The selection of a particular input line is controlled by a set of selection lines
- ✓ There are  $2^n$  input lines and  $n$  selection lines
- ✓ A 2-to-1 multiplexer has two data input lines, one output line and one selection line



# ✓ Multiplexers

## ➤ Multiplexer Circuit Structure

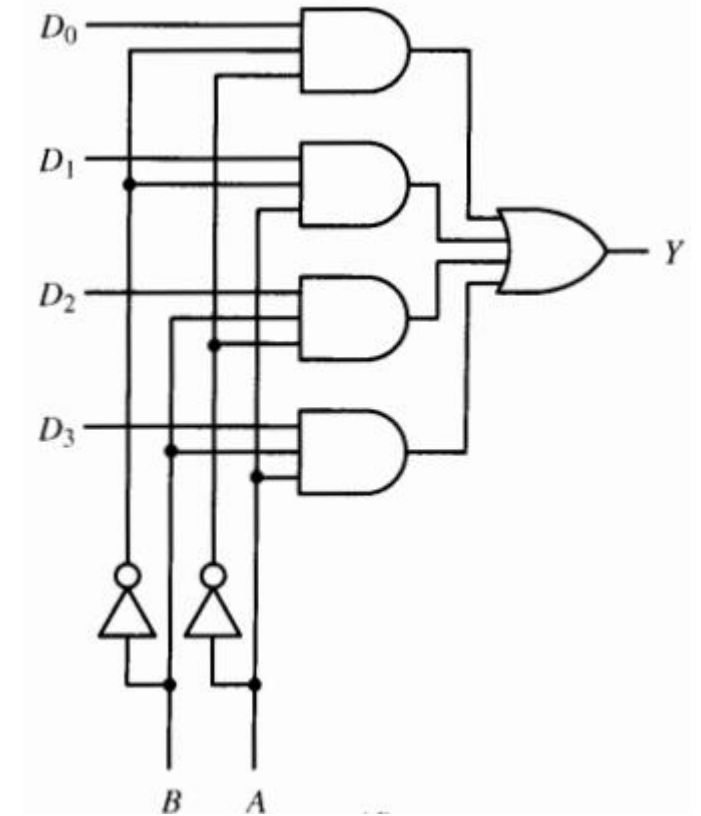
✓ A 4-to-1 multiplexer



<i>B</i>	<i>A</i>	<i>Y</i>
0	0	$D_0$
0	1	$D_1$
1	0	$D_2$
1	1	$D_3$

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

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



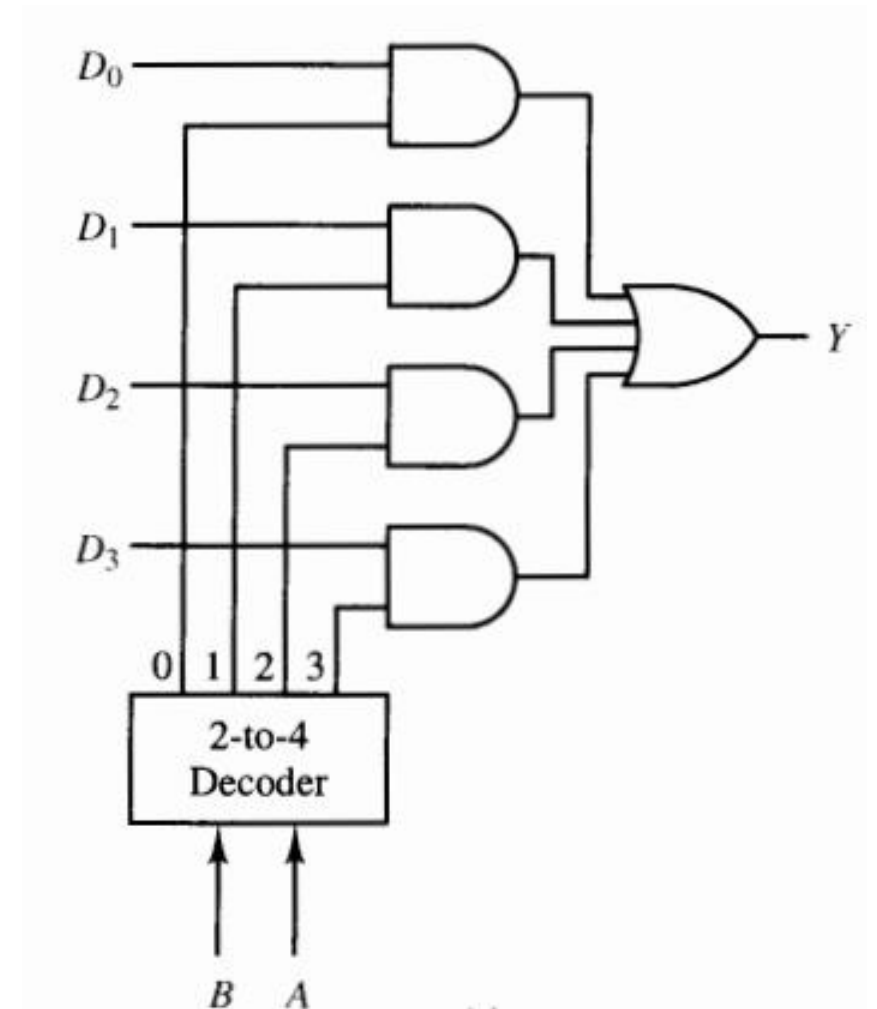
✓  $m_i$  are the minterms of the selection code

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

# ✓ Multiplexers

## ➤ Multiplexer Circuit Structure

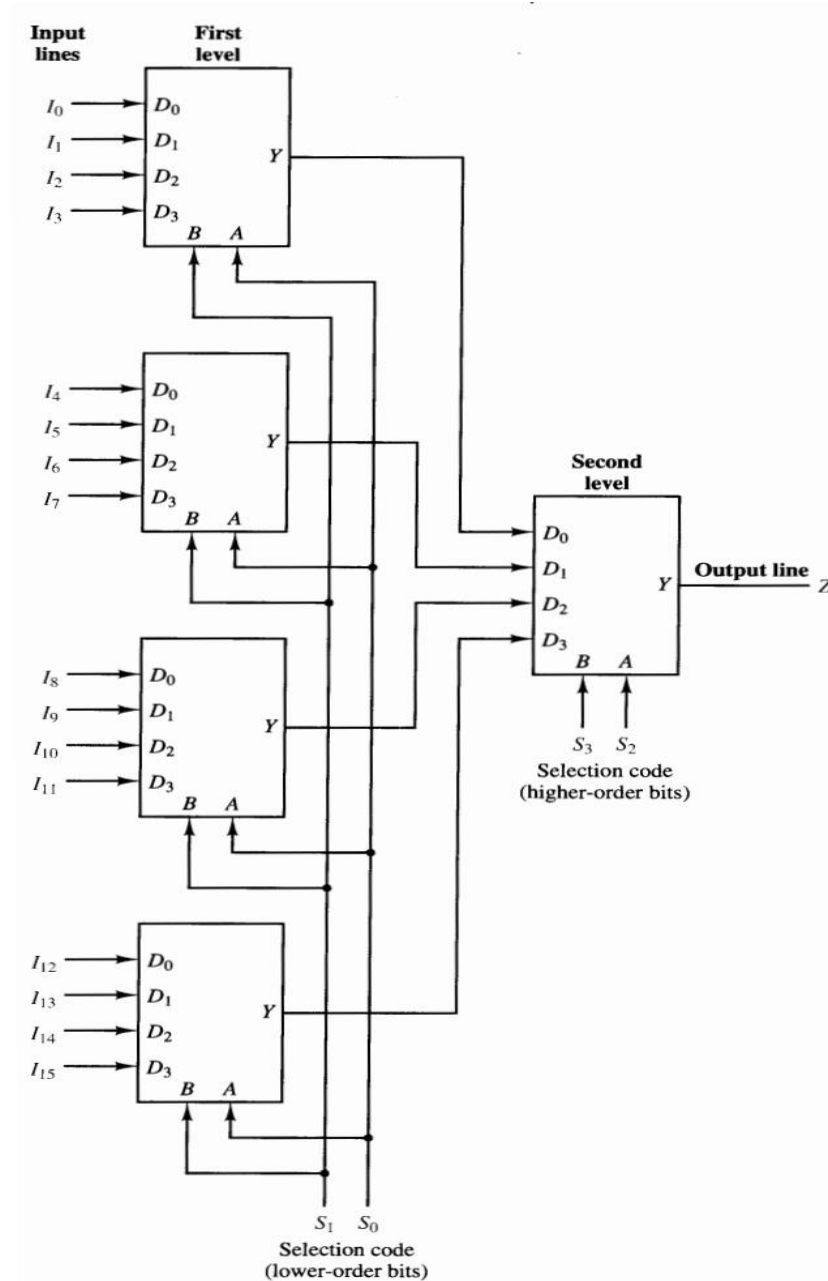
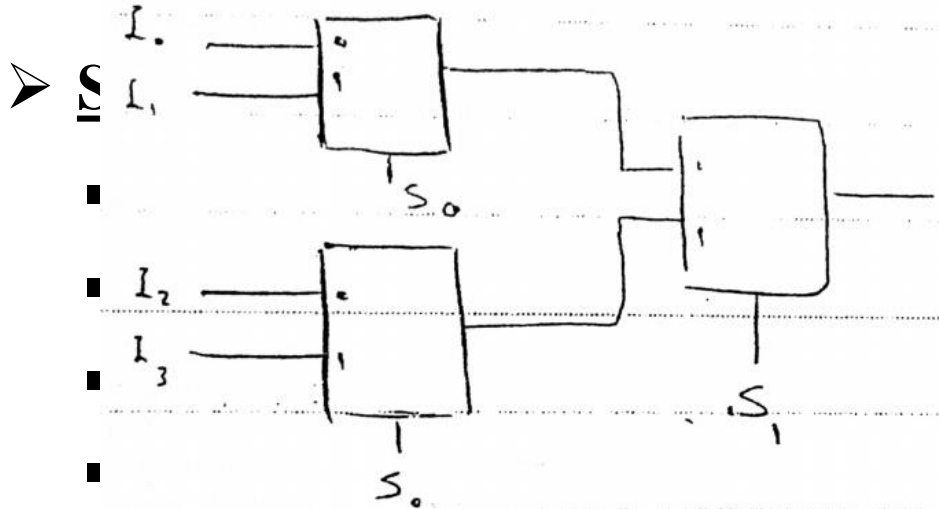
- ✓ In general, a  $2^n$ -to-1 line multiplexer is constructed from an  $n$ -to- $2^n$  decoder by adding  $2^n$  input lines to it, one to each AND gate



# ✓ Multiplexers

## ➤ Multiplexer Circuit Structure

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



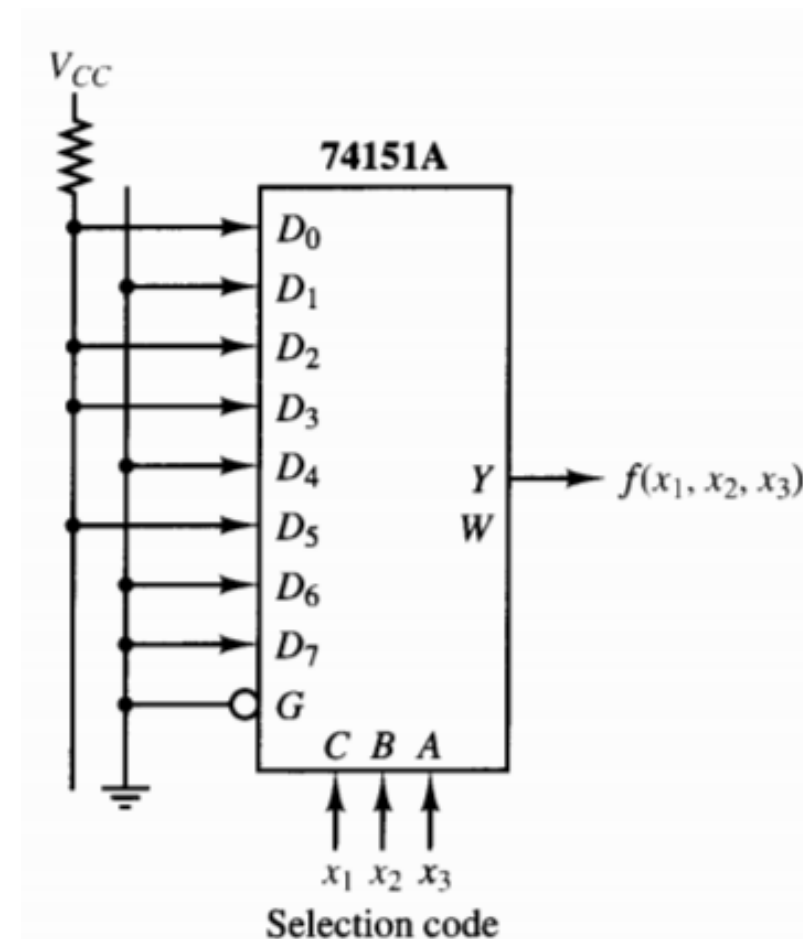
## ✓ Multiplexers

### ➤ 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

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

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

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

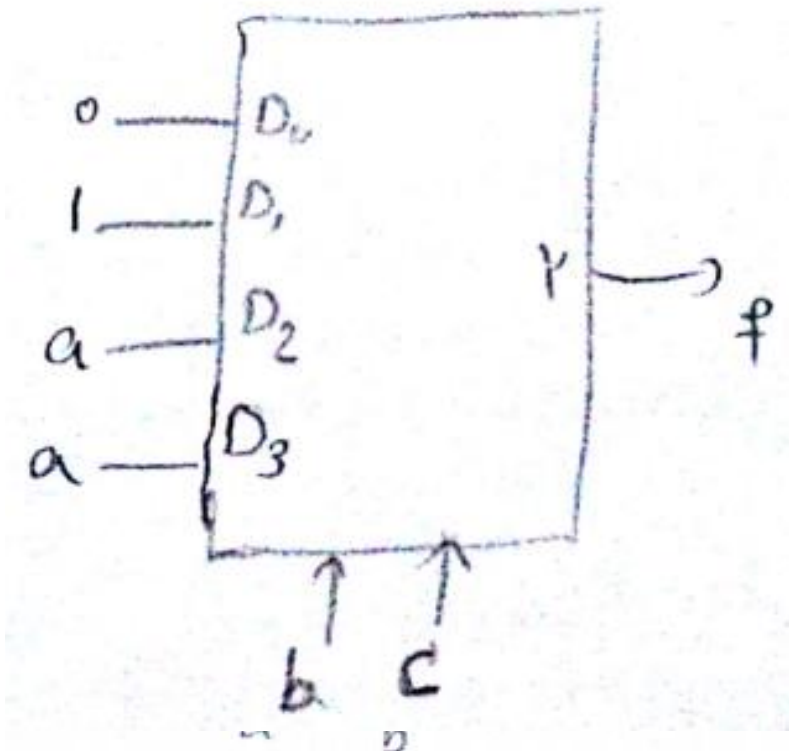
a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$(f=c) D_0$$

$$(f=0) D_1$$

$$(f=c) D_2$$

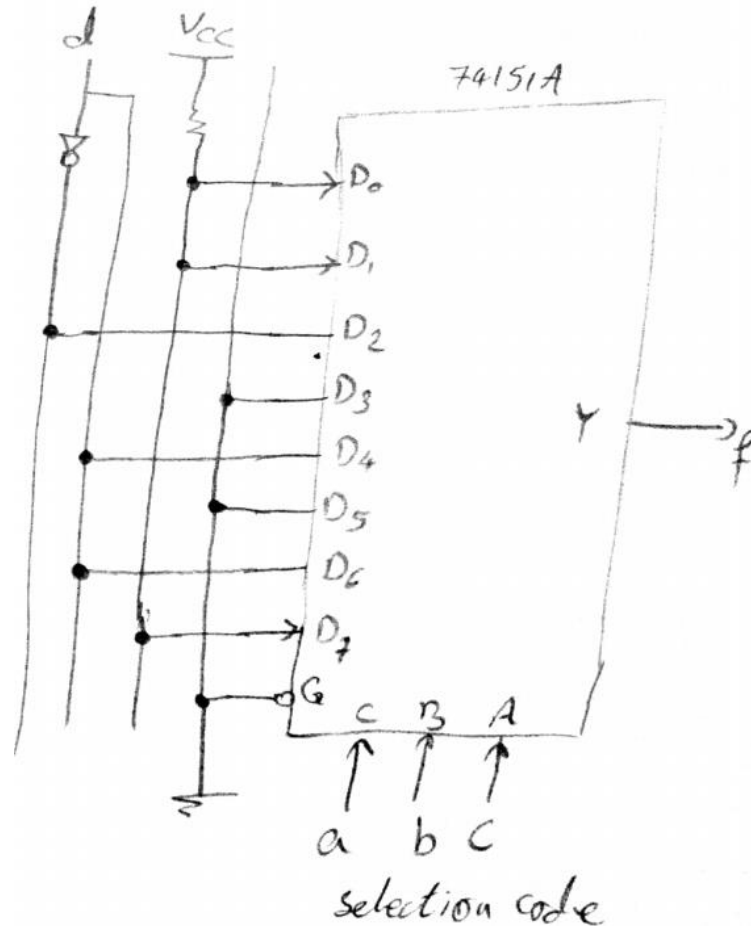
$$(f=1) D_3$$



## ✓ Multiplexers

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

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



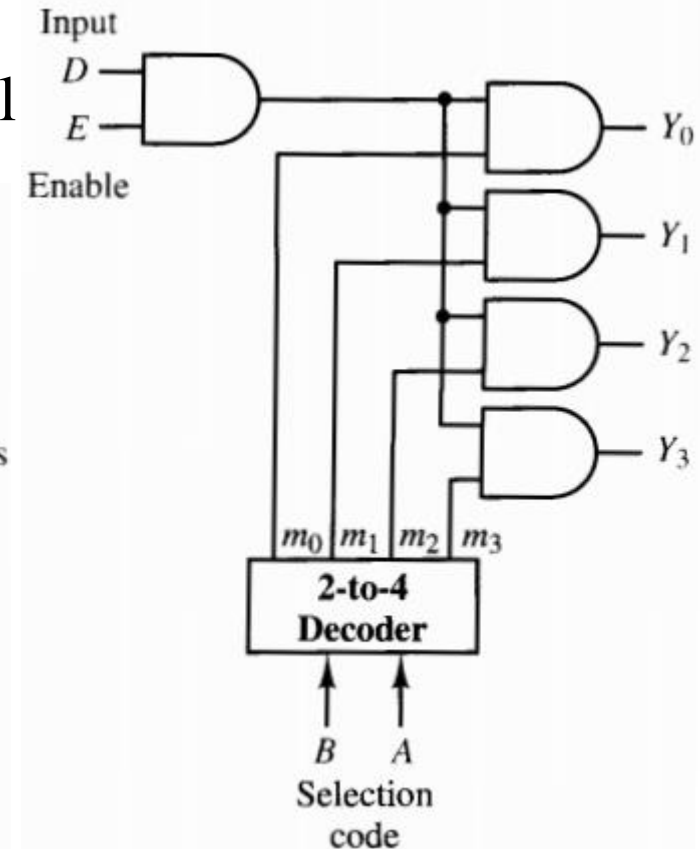
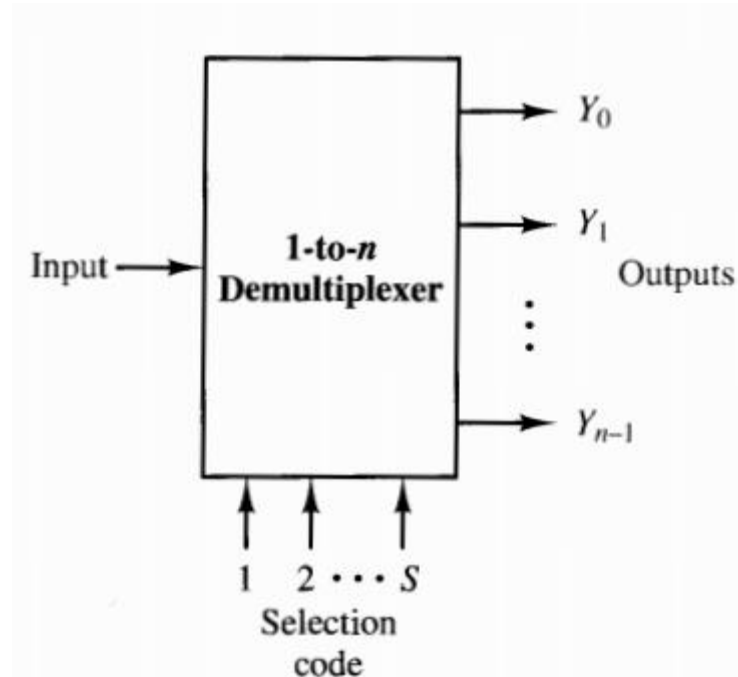
a	b	c	d	f
0	0	0	0	$f=1$ $D_0$
0	0	0	1	
0	0	1	0	$f=1$ $D_1$
0	0	1	1	
0	1	0	0	$f=\bar{d}$ $D_2$
0	1	0	1	
0	1	1	0	$f=0$ $D_3$
0	1	1	1	
1	0	0	0	$f=d$ $D_4$
1	0	0	1	
1	0	1	0	$f=0$ $D_5$
1	0	1	1	
1	1	0	0	$f=d$ $D_6$
1	1	0	1	
1	1	1	0	$f=1$ $D_7$
1	1	1	1	



## ✓ 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

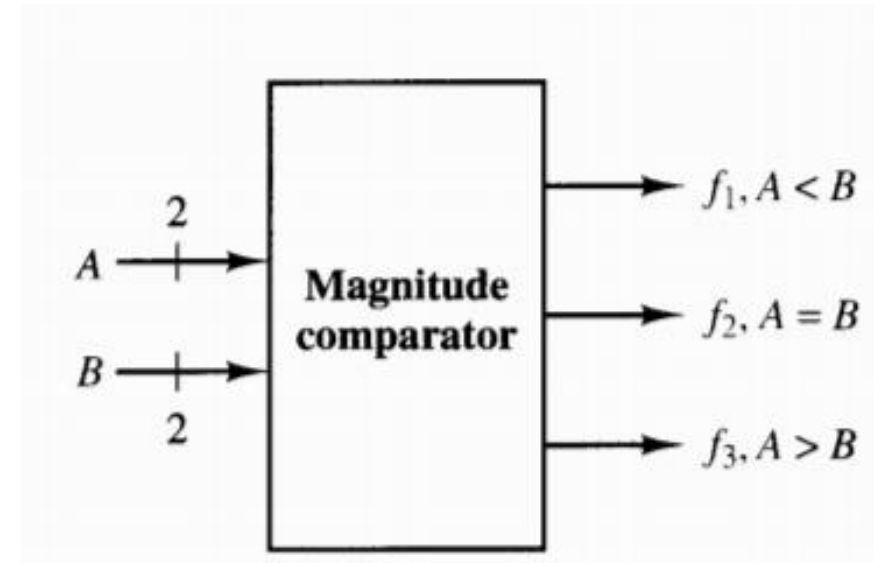
- ❖ Perform a magnitude comparison of two binary numbers A and B
- ❖ The outcome of a comparison is specified by three binary variables that indicate whether  $A > B$ ,  $A < B$  or  $A = B$
- ❖ So, the comparator will generate three output signal

as:

$$f_1 = 1, \quad \text{if } A < B$$

$$f_2 = 1, \quad \text{if } A = B$$

$$f_3 = 1, \quad \text{if } A > B$$



## ✓ Magnitude Comparator

### ❖ Design:

$$A = A_3 A_2 A_1 A_0$$

$$B = B_3 B_2 B_1 B_0$$

➤  $f_{A=B}$ :

$$x_i = A_i \odot B_i = A_i B_i + \overline{A_i} \overline{B_i}, i = 0, 1, 2, 3, \dots$$

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

➤  $f_{A>B}$ :

$$A_3 > B_3 \text{ or}$$

$$A_3 = B_3 \text{ and } A_2 > B_2 \text{ or}$$

$$A_3 = B_3 \text{ and } A_2 = B_2 \text{ and } A_1 > B_1 \text{ or}$$

$$A_3 = B_3 \text{ and } A_2 = B_2 \text{ and } A_1 = B_1 \text{ and } A_0 > B_0$$

$$f_{A>B} = A_3 \overline{B_3} + x_3 A_2 \overline{B_2} + x_3 x_2 A_1 \overline{B_1} + x_3 x_2 x_1 A_0 \overline{B_0}$$

3-level

➤  $f_{A<B}$ :

$$f_{A<B} = \overline{A_3} B_3 + x_3 \overline{A_2} B_2 + x_3 x_2 \overline{A_1} B_1 + x_3 x_2 x_1 \overline{A_0} B_0$$

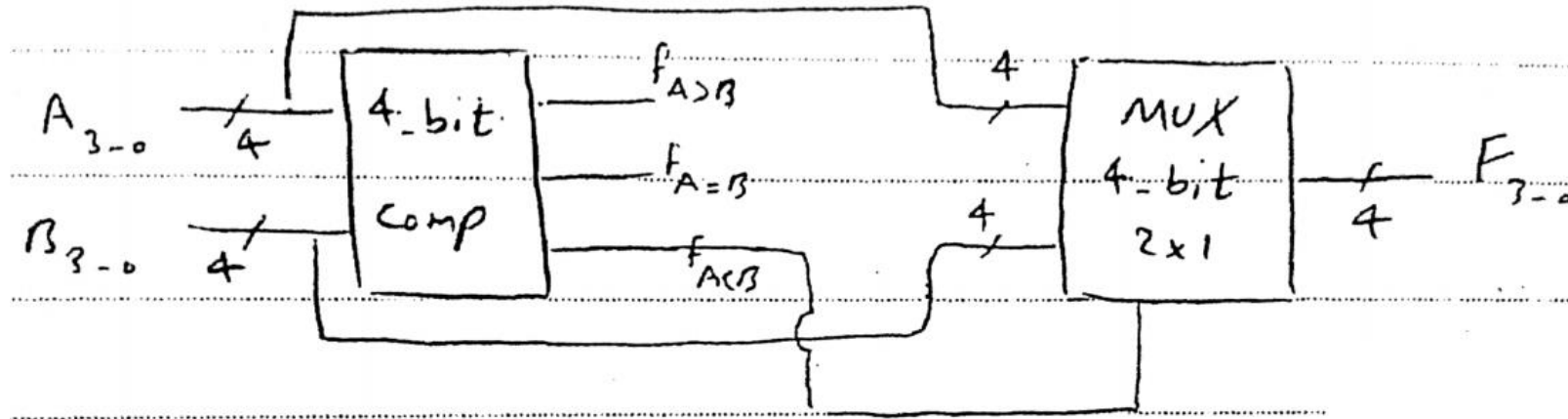
3-level

$$f_{A<B} = f_{A>B} \odot f_{A=B}$$

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.

## ✓ جمع کننده (Adder)

### ❖ نیم جمع کننده (Half Adder):

تنها دو بیت را باهم جمع می کند.

$$s_i = x_i \oplus y_i$$

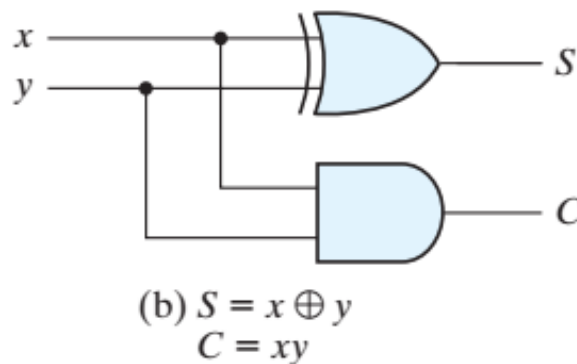
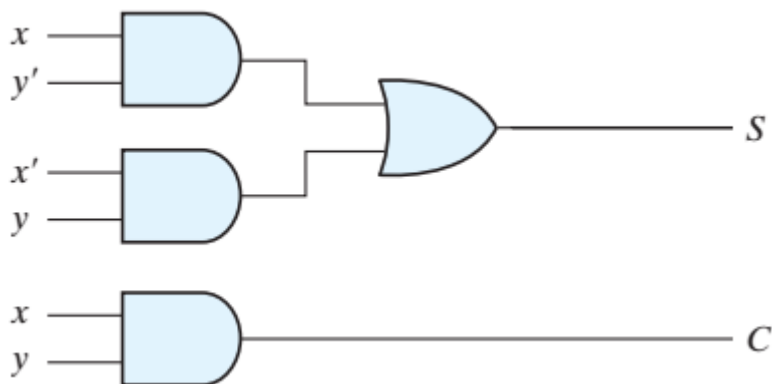
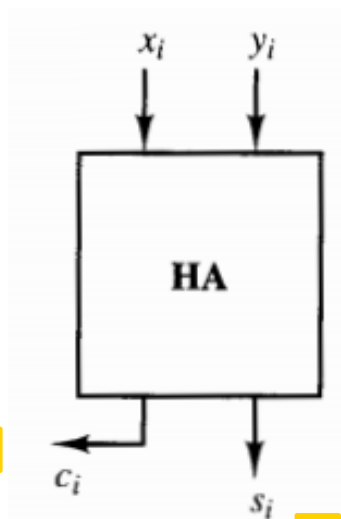
$$c_i = x_i y_i$$

S: Sum-bit

C: Carry-bit

Half Adder

<b>x</b>	<b>y</b>	<b>C</b>	<b>S</b>
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



❖ تاخیر این بلوک را یک مرحله در نظر می گیریم.

## ✓ جمع کننده (Adder)

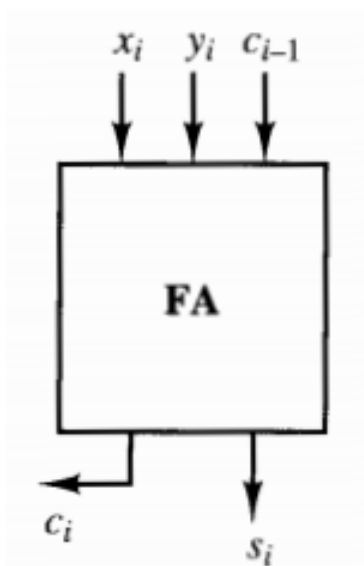
### ❖ تمام جمع کننده (Full Adder):

می تواند سه بیت را باهم جمع می کند.

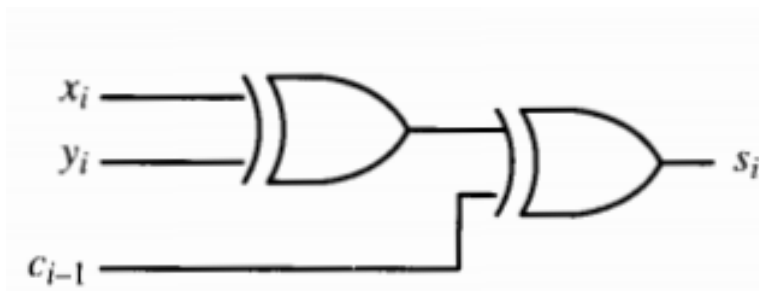
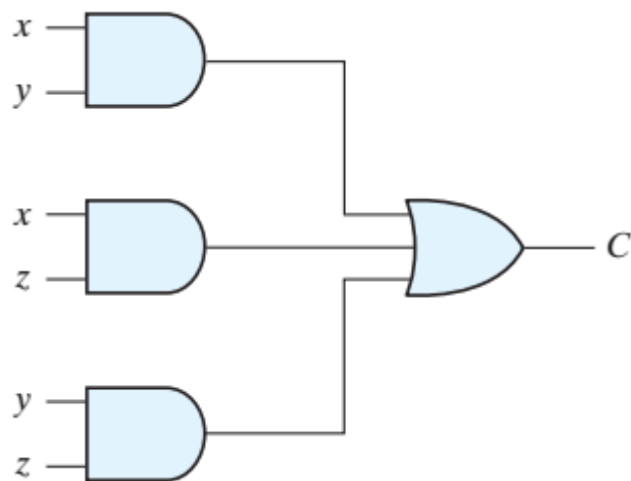
❖  $C_{i-1}$  بیت carry مرحله قبل است.

$$s_i = x_i \oplus y_i \oplus c_{i-1}$$

$$c_i = x_i y_i + x_i c_{i-1} + y_i c_{i-1}$$



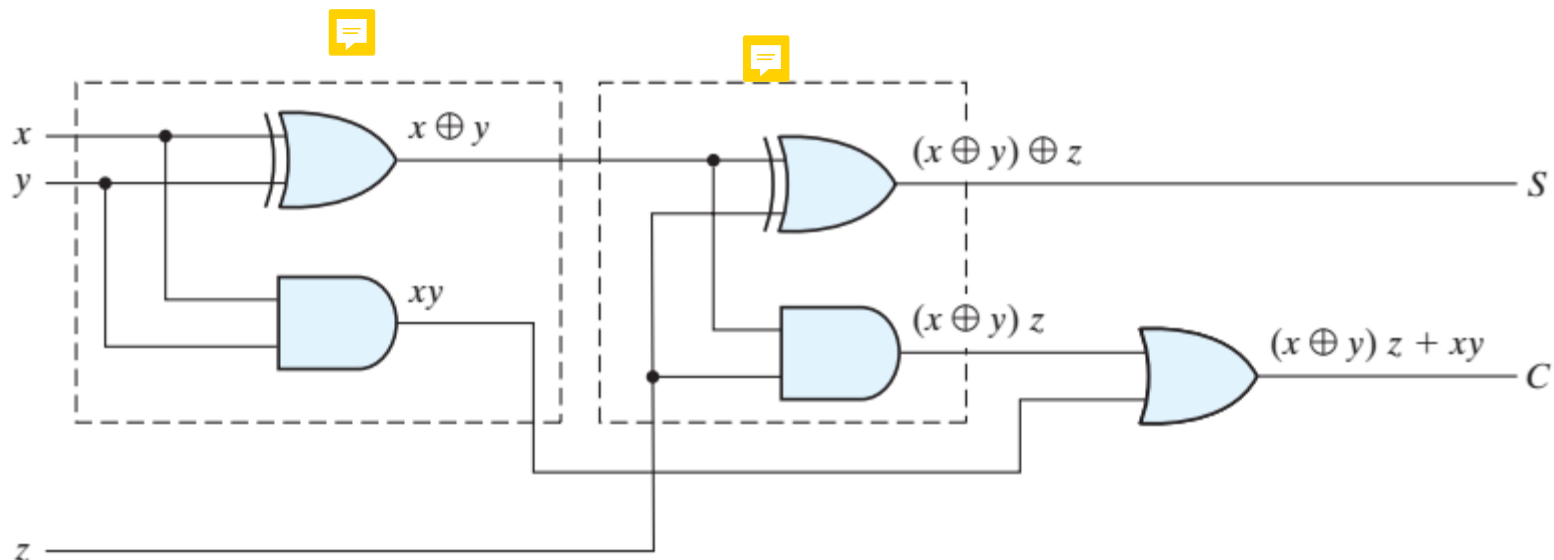
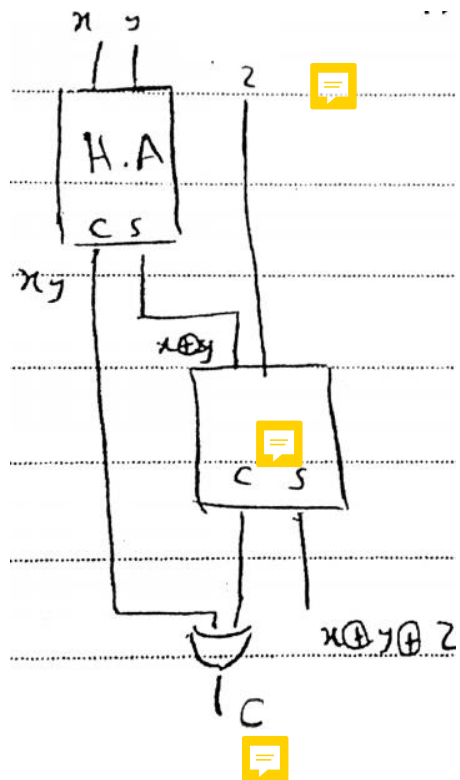
$x_i$	$y_i$	$c_{i-1}$	$c_i$	$s_i$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



❖ تاخیر این بلوک را دو مرحله در نظر می گیریم.

## ✓ جمع کننده (Adder)

❖ ساخت FA با استفاده از HA:



❖ تاخیر این نوع پیاده سازی ۳ مرحله است.

❖ در این درس فرض می کنیم که تمام FA ها از این طریق پیاده سازی شده اند.

## ✓ جمع کننده (Adder)

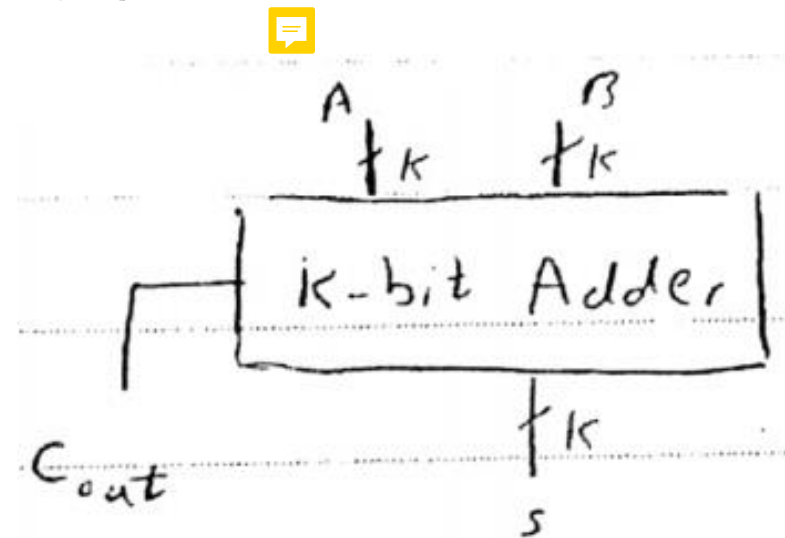
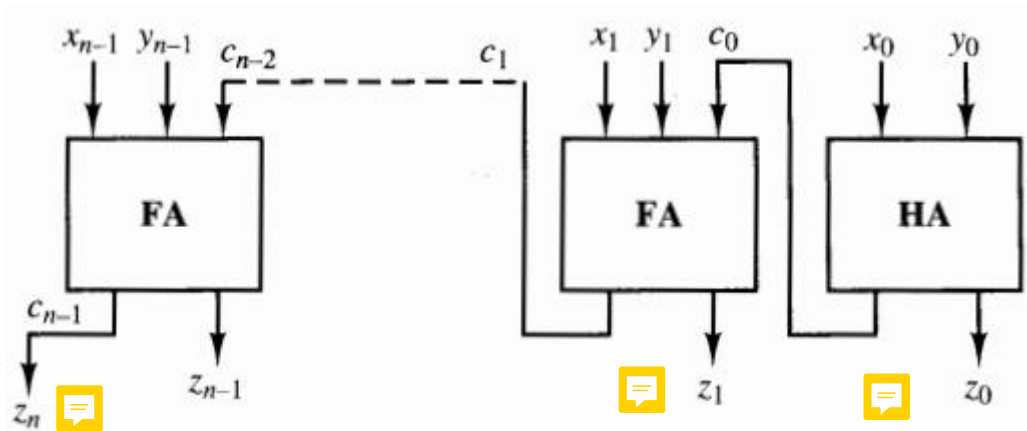
❖ جمع دو عدد  $n$  بیتی:

نیاز به 1 عدد HA و  $n-1$  عدد FA داریم.

❖ به این نوع جمع کننده، Pseudoparallel Adder گویند.

❖ از آنجا که رقم نقلی در طول جمع کننده انتشار می یابد، به آن Carry Propagation Adder (CPA) نیز گویند.

Input carry	0	1	1	0	$C_i$
Augend	1	0	1	1	$A_i$
Addend	0	0	1	1	$B_i$
Sum	1	1	1	0	$S_i$
Output carry	0	0	1	1	$C_{i+1}$





## ✓ جمع کننده (Adder)

### ❖ مشکل جمع کننده های Pseudoparallel:

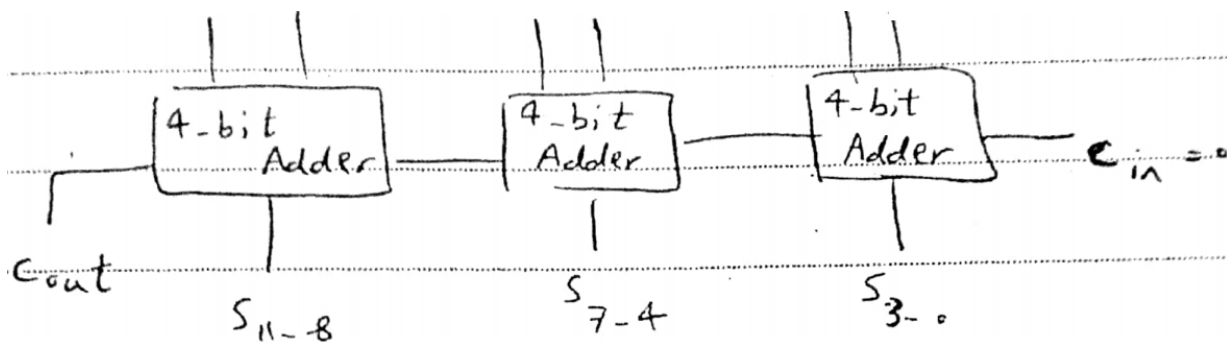
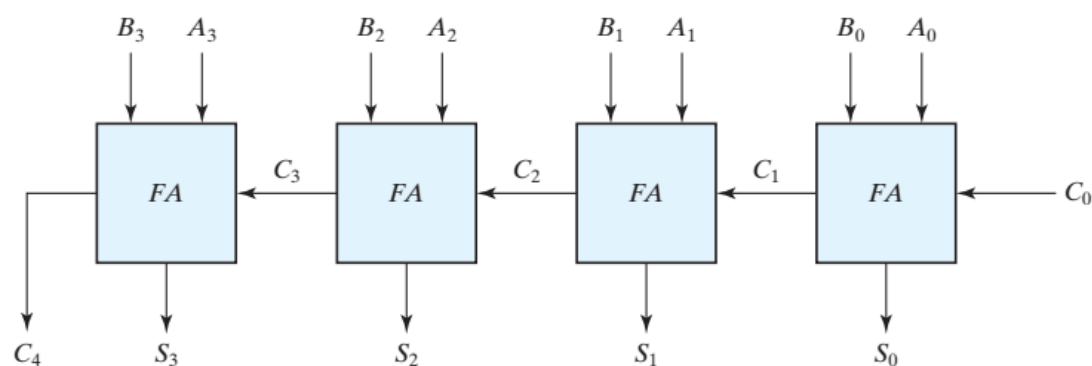
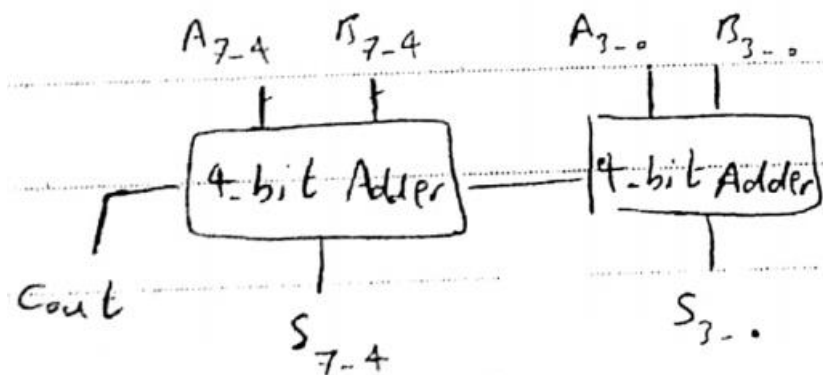
نمی توان دو یا چندتا از آن ها را کنار هم گذاشت.

❖ برای رفع این مشکل، در ابتدای CPA بجای اینکه از HA استفاده شود، از FA استفاده می شود با یک carry ورودی.

### ❖ (Fully Parallel Adder)

❖ جمع کننده ۴ بیتی که بدین صورت پیاده سازی شود، یک component استاندارد است که زیاد از آن استفاده می شود.

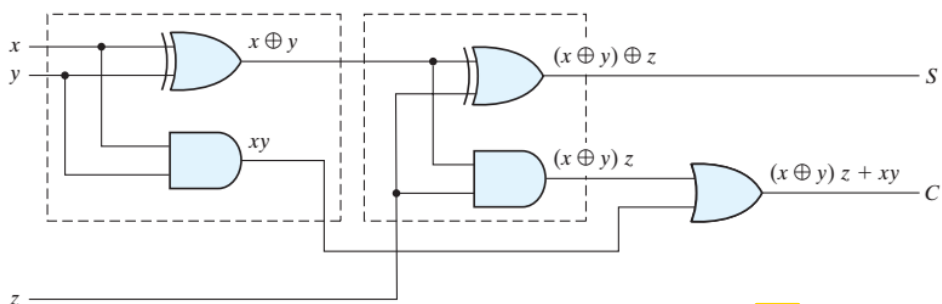
❖ به عنوان نمونه برای جمع دو عدد ۱۲ بیتی، می توان ۳ جمع کننده ۴ بیتی را کنار هم گذاشت.



## ✓ جمع کننده (Adder)

### ❖ تاخیر جمع کننده CPA:

✓ در این نوع جمع کننده بیت carry در طول جمع کننده انتشار می یابد.



✓ ملاحظه کردیم که تاخیر یک FA که با دو HA ساخته می شود، سه مرحله تاخیر است.

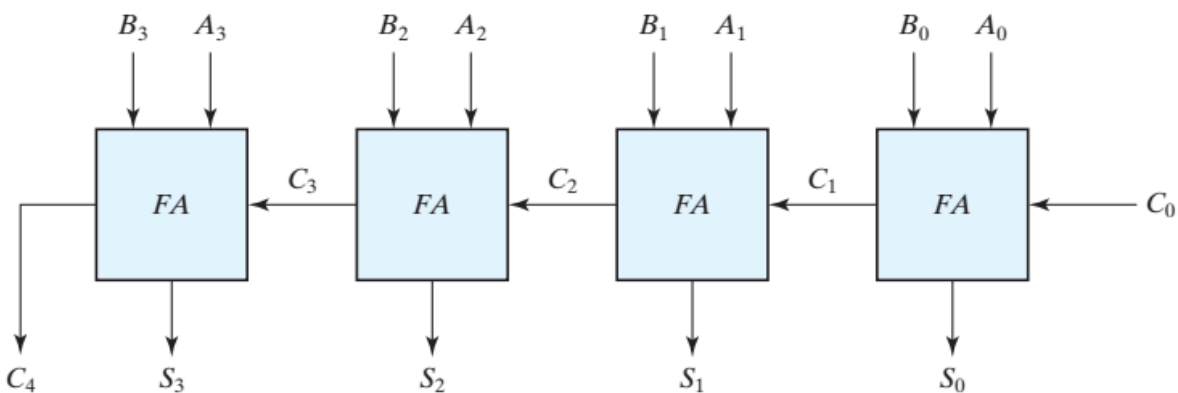
✓ تاخیر یک جمع کننده ۴ بیتی چقدر است؟



■ تاخیر بیت carry، ۹ مرحله است. **(1+2n)**

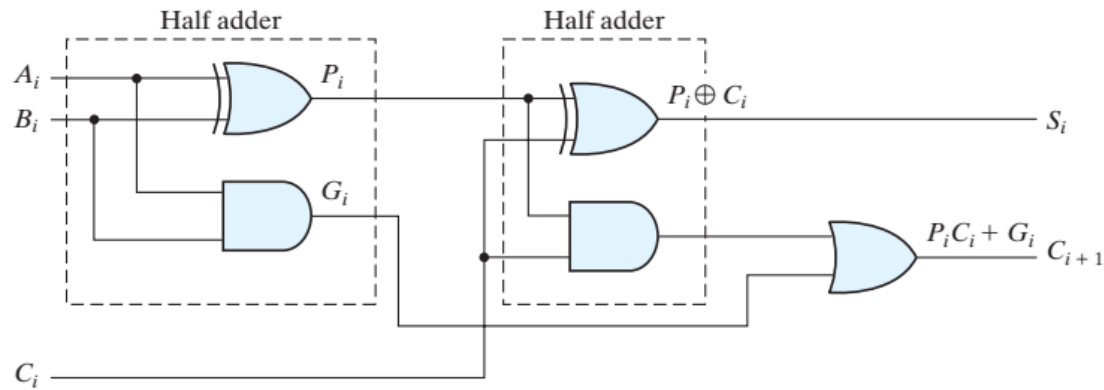
■ تاخیر Sها، ۸ مرحله است.

■ در کل این جمع کننده، ۹ مرحله تاخیر دارد.



✓ نقطه ضعف این نوع جمع کننده این است که تاخیر به تعداد بیت ورودی وابسته است.

## ✓ جمع کننده (Adder)



❖ جمع کننده با سرعت بالا (High-Speed Adders)

✓ Carry Look-ahead Adders (CLA)

$$P_i = A_i \oplus B_i$$

$$S_i = P_i \oplus C_i$$

$$G_i = A_i B_i$$

$$C_{i+1} = G_i + P_i C_i$$

✓  $G_i$ : Carry Generate

✓  $P_i$ : Carry Propagate

✓  $P_i$  و  $G_i$  بطور موازی برای همه بیت ها با یک سطح تاخیر ایجاد می شود.

$C_0$  = input carry

$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 = P_2 P_1 P_0 C_0$$

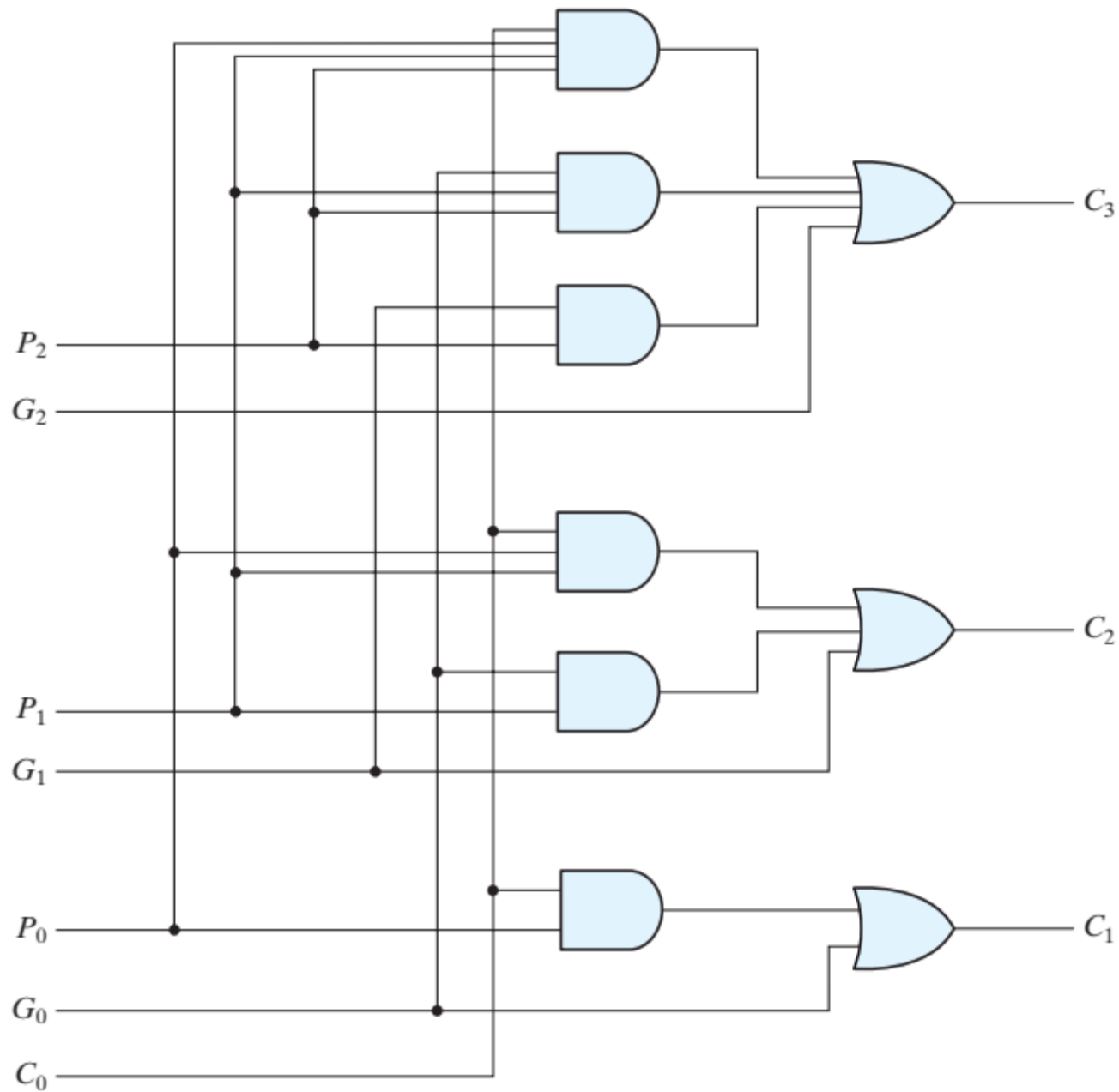
✓ سپس تمام Carry ها بطور موازی با

دو سطح تاخیر تولید می شوند (SOP)

✓ کل تاخیر این نوع جمع کننده (CLA) برای تولید Carry ها، ۳ مرحله است. (مستقل از تعداد بیت ورودی)

✓ جمع کننده (Adder)

❖ Carry Look-ahead Adders (CLA)



## ✓ جمع کننده (Adder)

### ❖ Carry Look-ahead Adders (CLA)

✓ یک مرحله تاخیر دیگر نیز برای تولید  $S_i$  ها خواهیم داشت.

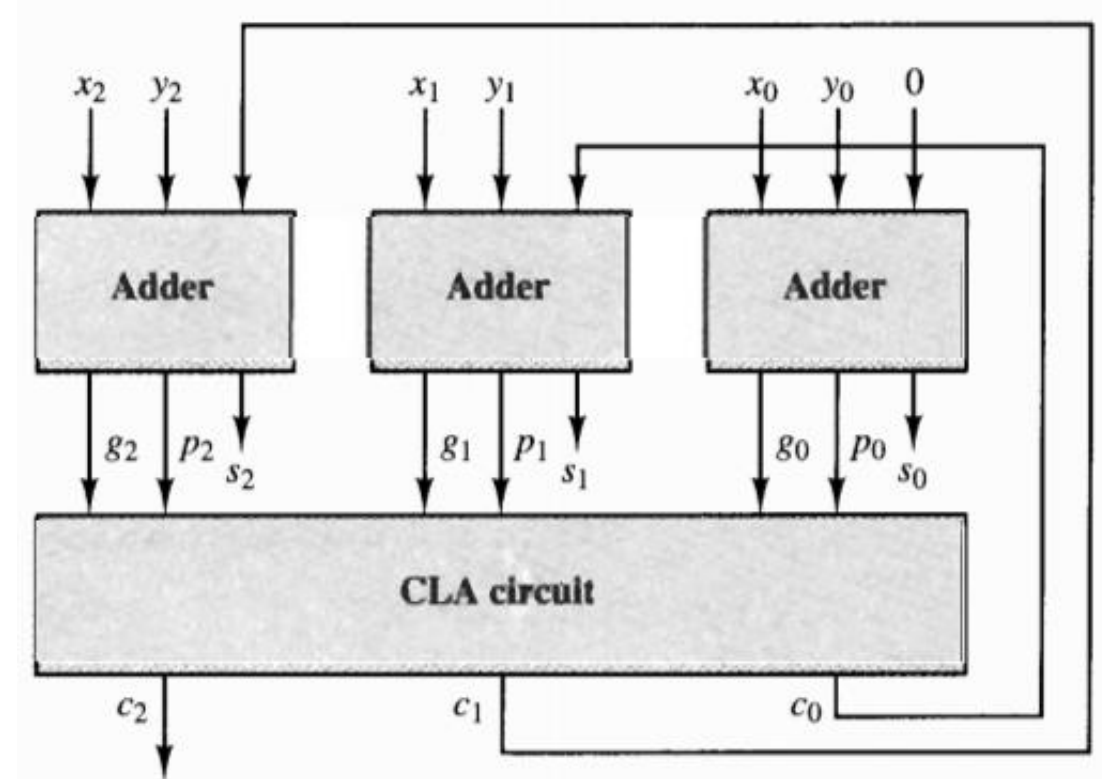
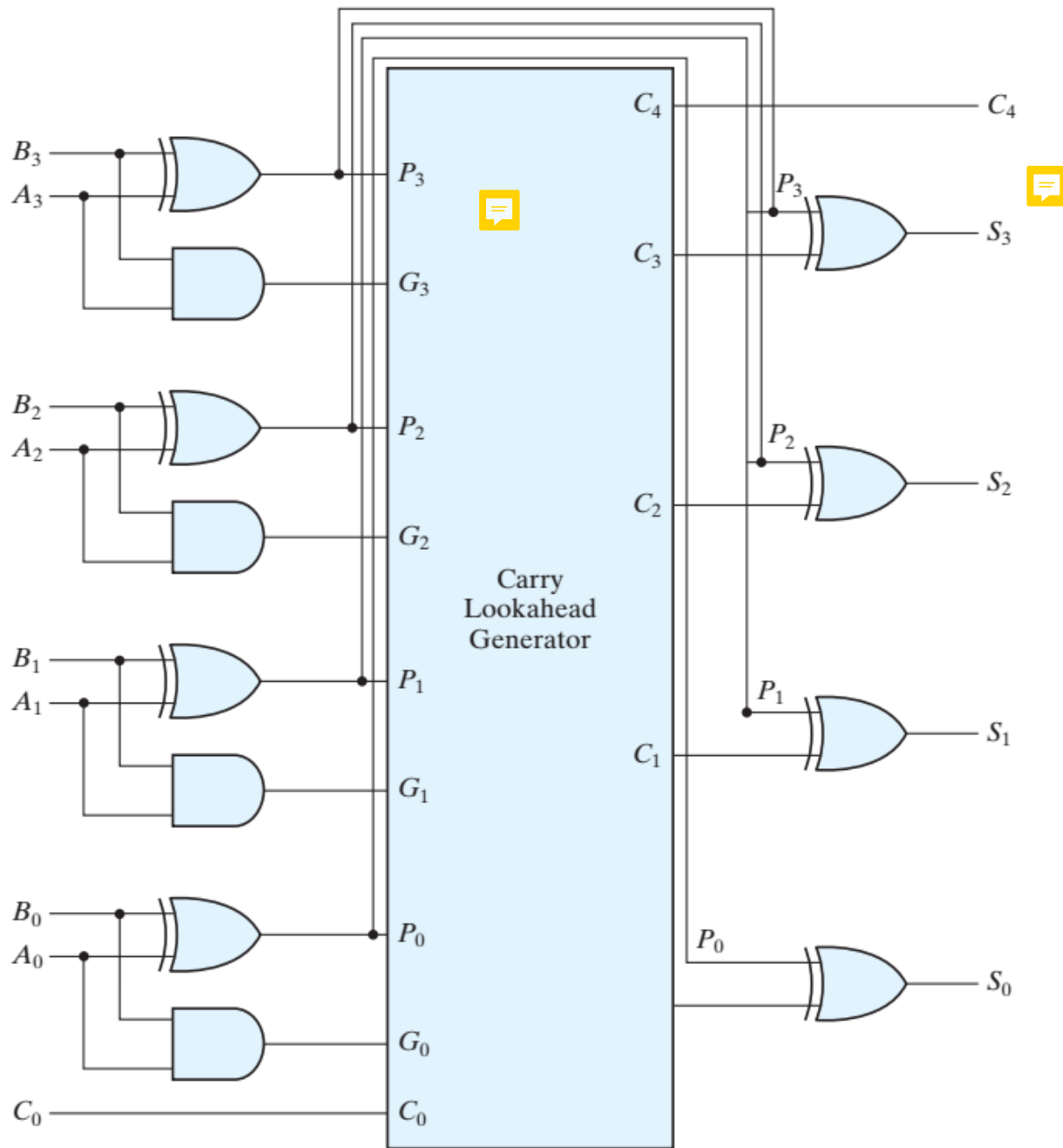
$$S_i = P_i \oplus C_i$$

✓ کل تاخیر این نوع جمع کننده (CLA) برای تولید S ها، ۴ مرحله است. (مستقل از تعداد بیت ورودی)

$$t_{\text{cla}} = 4t_{\text{gate}}$$

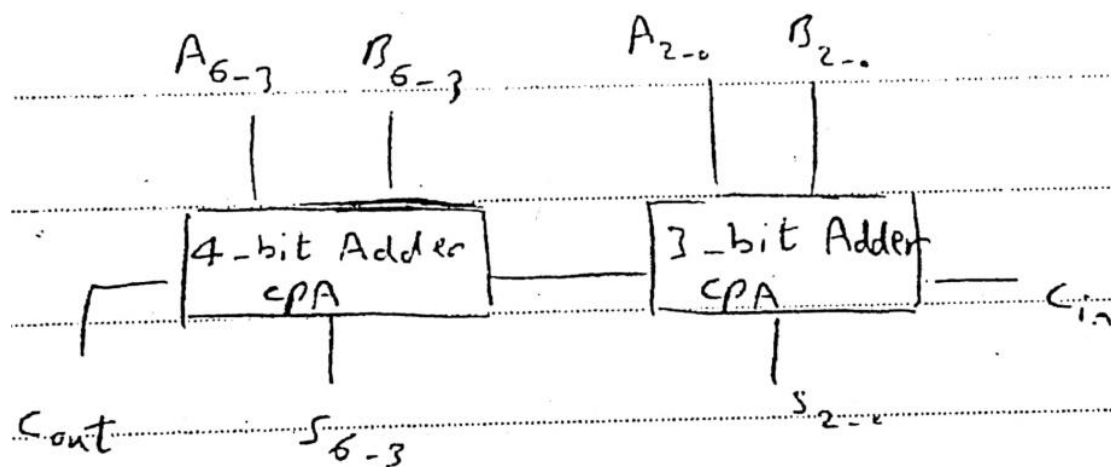
✓ جمع کننده (Adder)

❖ Carry Look-ahead Adders (CLA)



## ✓ جمع کننده (Adder)

❖ مثال: در شکل زیر تاخیر تولید بیت carry را برای حالت های مختلف جمع کننده بیابید.



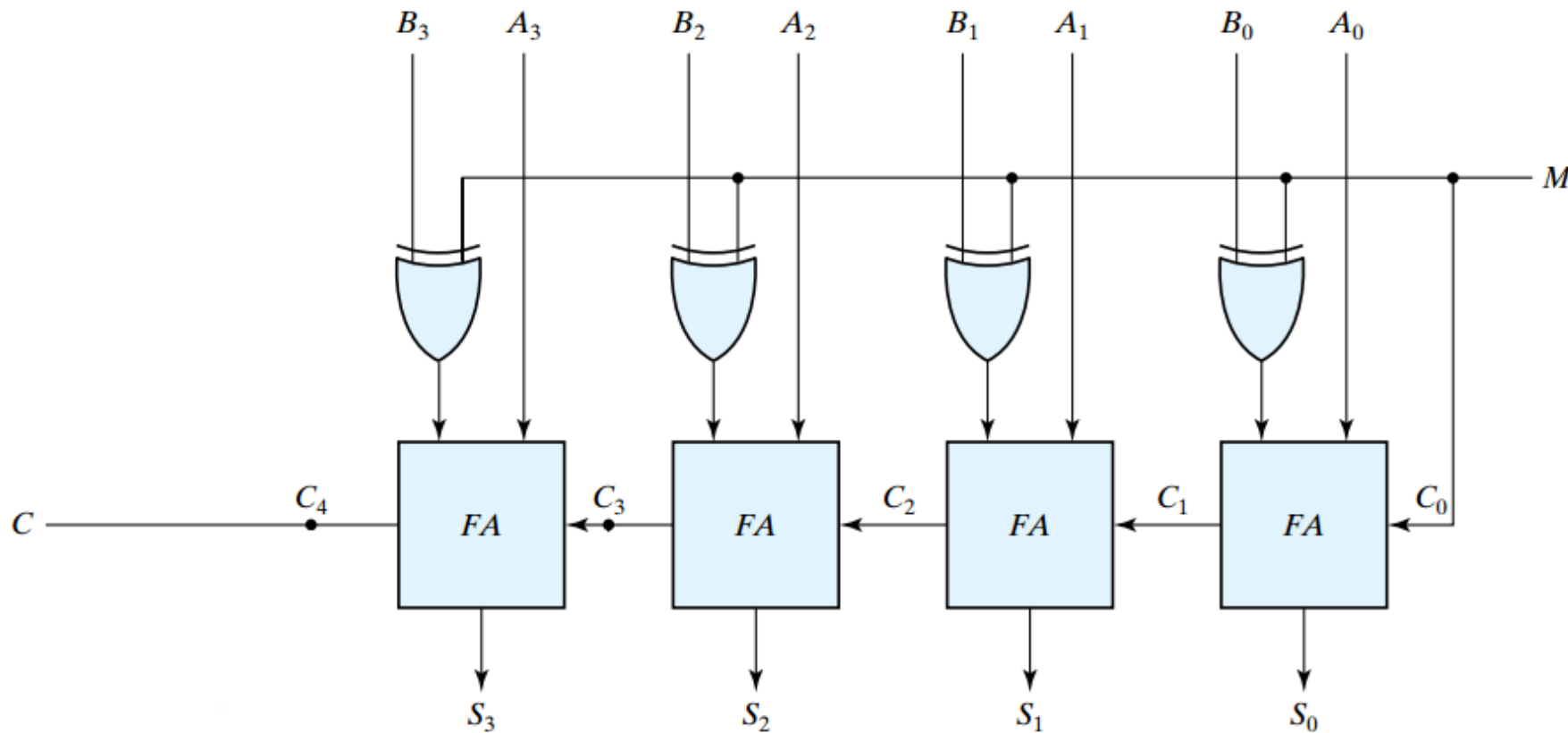
✓ ترکیب CPA, CPA: ۱۵ مرحله تاخیر

✓ ترکیب CPA, CLA: ۹ مرحله تاخیر

✓ ترکیب CPA, CLA: ۱۱ مرحله تاخیر

✓ ترکیب CLA, CLA: ۵ مرحله تاخیر

## ✓ تفريق کننده (Subtractor)



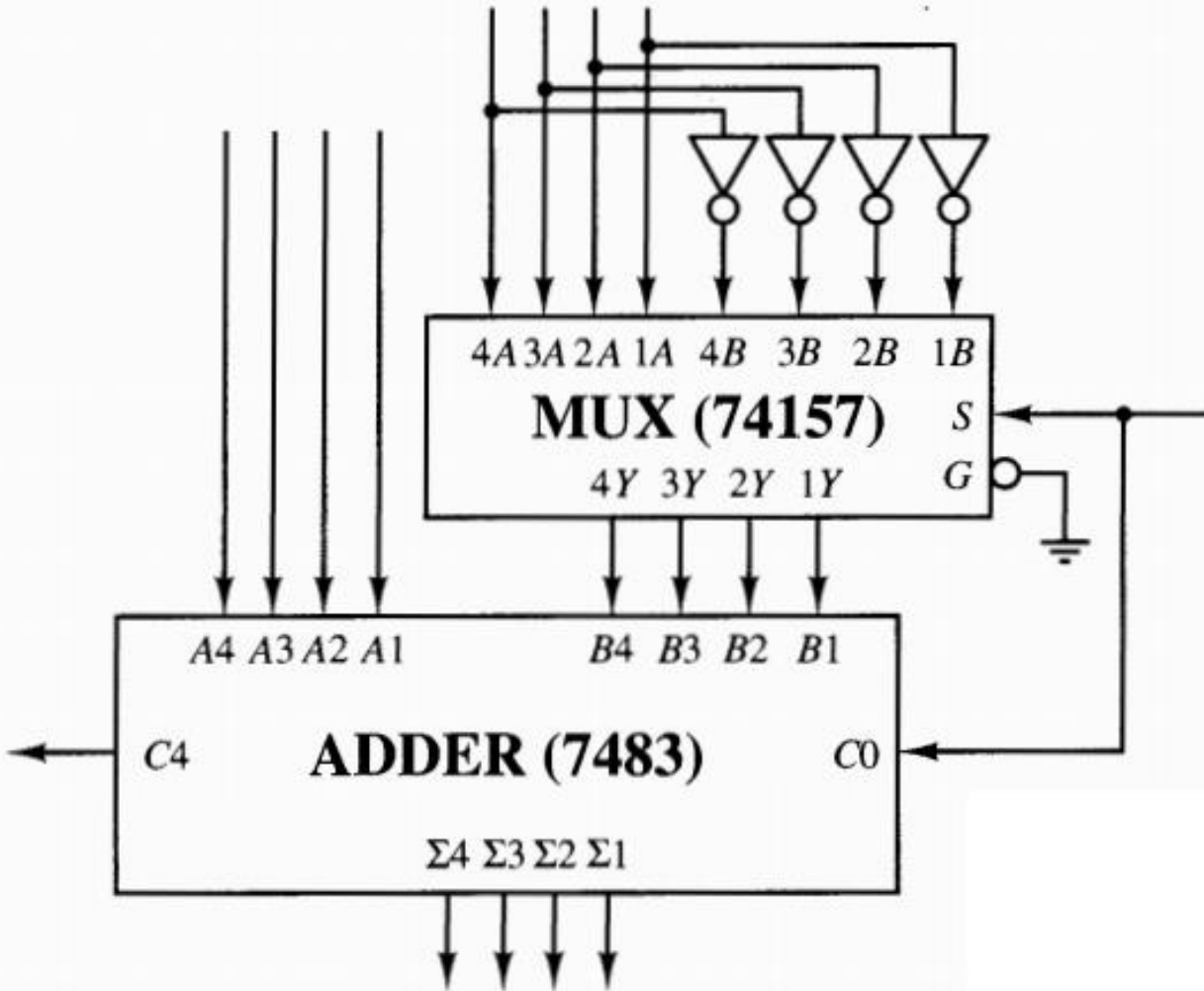
$if\ M = 0 \Rightarrow A + B + 0 = A + B$  (Adder)

$if\ M = 1 \Rightarrow A + \bar{B} + 1 = A + C_2(B) = A - B$  (Subtractor)

❖ دو جعبه K بیتی از این ها را نمی توان کنار هم گذاشت.



## ✓ تفريق کننده (Subtractor)



$if S = 0 \Rightarrow A + B$  (Adder)

$if S = 1 \Rightarrow A - B$  (Subtractor)

## ✓ تشخیص سرریز (Overflow Detection)

❖ سرریز در دو حالت رخ می دهد:

✓ در اعداد بی علامت: زمانی که جمع دو عدد  $n$  بیتی، یک عدد  $n+1$  بیتی شود.

✓ در اعداد علامت دار: زمانی که جمع دو عدد مثبت، منفی شود یا جمع دو عدد منفی، مثبت شود.

✓ پس در اعداد علامت دار با استفاده از بیت های آخر که علامت عدد را نشان می دهند، می توان سرریز را تشخیص داد.

carries:	0 1	carries:	1 0
+70	0 1000110	-70	1 0111010
+80	0 1010000	-80	1 0110000
<hr/>	<hr/>	<hr/>	<hr/>
+150	1 0010110	-150	0 1101010

# تشخیص سرریز (Overflow Detection) ✓

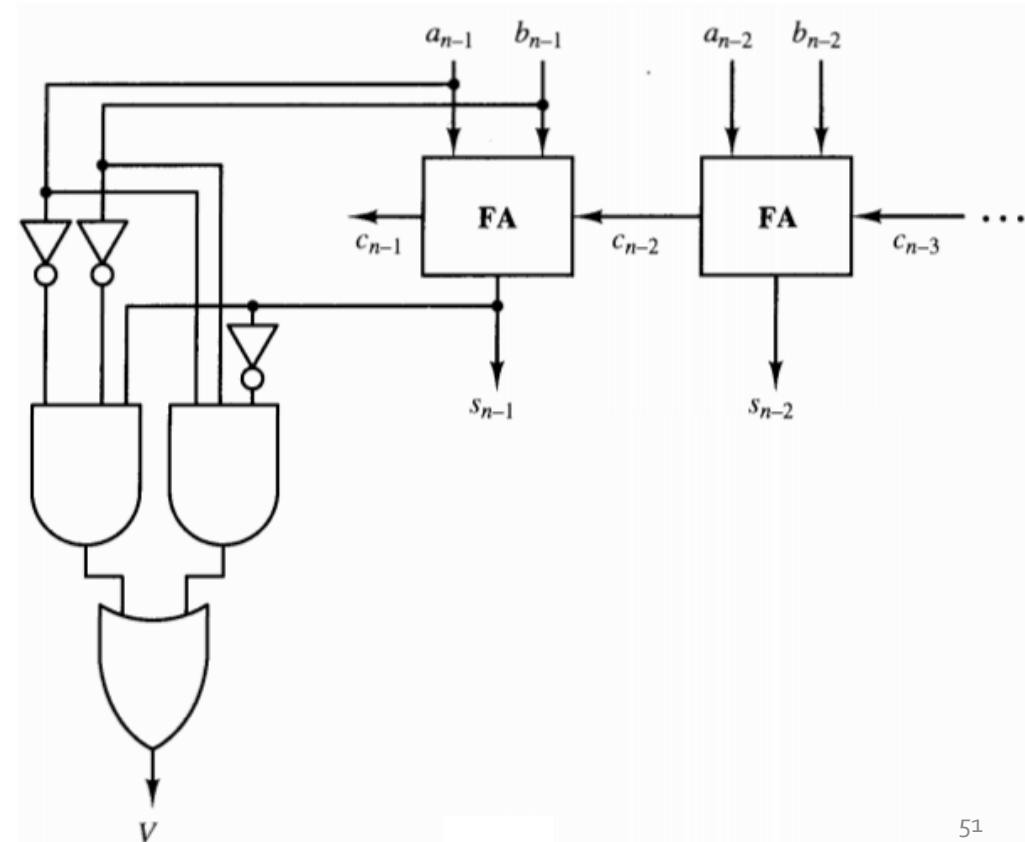
❖ سرریز زمانی رخ می دهد که:

✓ جمع دو عدد مثبت، نتیجه اش  $S_{n-1} = 1$

✓ جمع دو عدد منفی، نتیجه اش  $S_{n-1} = 0$

Adder Inputs			Adder Outputs		Overflow
$a_{n-1}$	$b_{n-1}$	$c_{n-2}$	$c_{n-1}$	$s_{n-1}$	$V$
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

$$V = \bar{a}_{n-1}\bar{b}_{n-1}s_{n-1} + a_{n-1}b_{n-1}\bar{s}_{n-1}$$



## ✓ تشخیص سرریز (Overflow Detection)

❖ روش دیگر استفاده از بیت carry دو مرحله آخر است.

✓ اگر بیت carry ورودی و خروجی، معادل باشند، سرریز نداریم و در غیر اینصورت داریم.

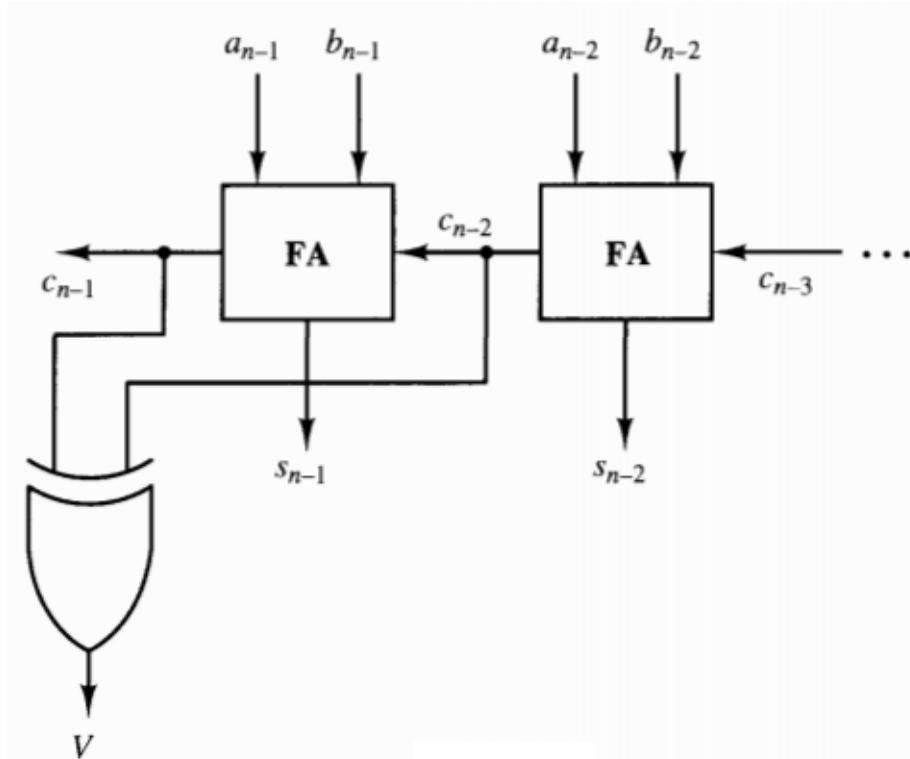
$$V = c_{n-2} \oplus c_{n-1}$$

✓ از این روش زمانی می توان استفاده کرد که بیت  $c_{n-2}$  در دسترس باشد.

❖ در اعداد بی علامت، با استفاده از بیت  $c_{n-1}$  می توان سرریز را تشخیص داد.

❖ در اعداد علامت دار، با استفاده از بیت  $V$  می توان سرریز را تشخیص داد.

Adder Inputs			Adder Outputs		Overflow
$a_{n-1}$	$b_{n-1}$	$c_{n-2}$	$c_{n-1}$	$s_{n-1}$	$V$
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0



## ✓ جمع کننده دهدهی (Decimal Adder)

- ❖ ماشین حساب ها و کامپیوترها برای جمع دو عدد دهدهی، معمولاً از کد BCD استفاده می کنند.
- ❖ پس می بایست ورودی و خروجی بصورت BCD باشند.
- ❖ مدار مورد نظر ۹ ورودی و ۵ خروجی خواهد داشت.

### ❖ جمع کننده BCD (BCD Adder):

✓ جمع دو رقم دهدهی، در بازه 0 تا 19 خواهد بود.


### ✓ روند طراحی جمع کننده BCD:

- دو کد BCD را بصورت باینری با هم جمع می کنیم.
- اگر حاصل کوچکتر یا مساوی ۹ بود، معادل BCD نیز همان می شود.
- اگر حاصل بزرگتر از ۹ بود، عدد بدست آمده را با 0110 (6) جمع می کنیم که ۴ بیت حاصل معادل BCD بوده و بیت سمت چپ، به عنوان carry در نظر گرفته می شود.

## ✓ جمع کننده دهدهی (Decimal Adder)



### ❖ جمع کننده BCD (BCD Adder):

$$\begin{array}{r}
 4 \quad 0100 \\
 +5 \quad +0101 \\
 \hline
 9 \quad 1001
 \end{array}
 \qquad
 \begin{array}{r}
 8 \quad 1000 \\
 +9 \quad 1001 \\
 \hline
 17 \quad 10001 \\
 +0110 \\
 \hline
 10111
 \end{array}$$

BCD	1	1 		
	0001	1000	0100	184
	+0101	0111	0110	+576
Binary sum	0111	10000	1010	
Add 6		0110	0110	
BCD sum	0111	0110	0000	760

✓ در نهایت تمامی قوانین را بصورت جدول نوشته و پیاده سازی می کنیم.

# (BCD Adder) ✓

Binary Sum					BCD Sum					Decimal
 K	Z <sub>8</sub>	Z <sub>4</sub>	Z <sub>2</sub>	Z <sub>1</sub>	 C	S <sub>8</sub>	S <sub>4</sub>	S <sub>2</sub>	S <sub>1</sub>	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
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

$$C = K + Z_8Z_4 + Z_8Z_2$$

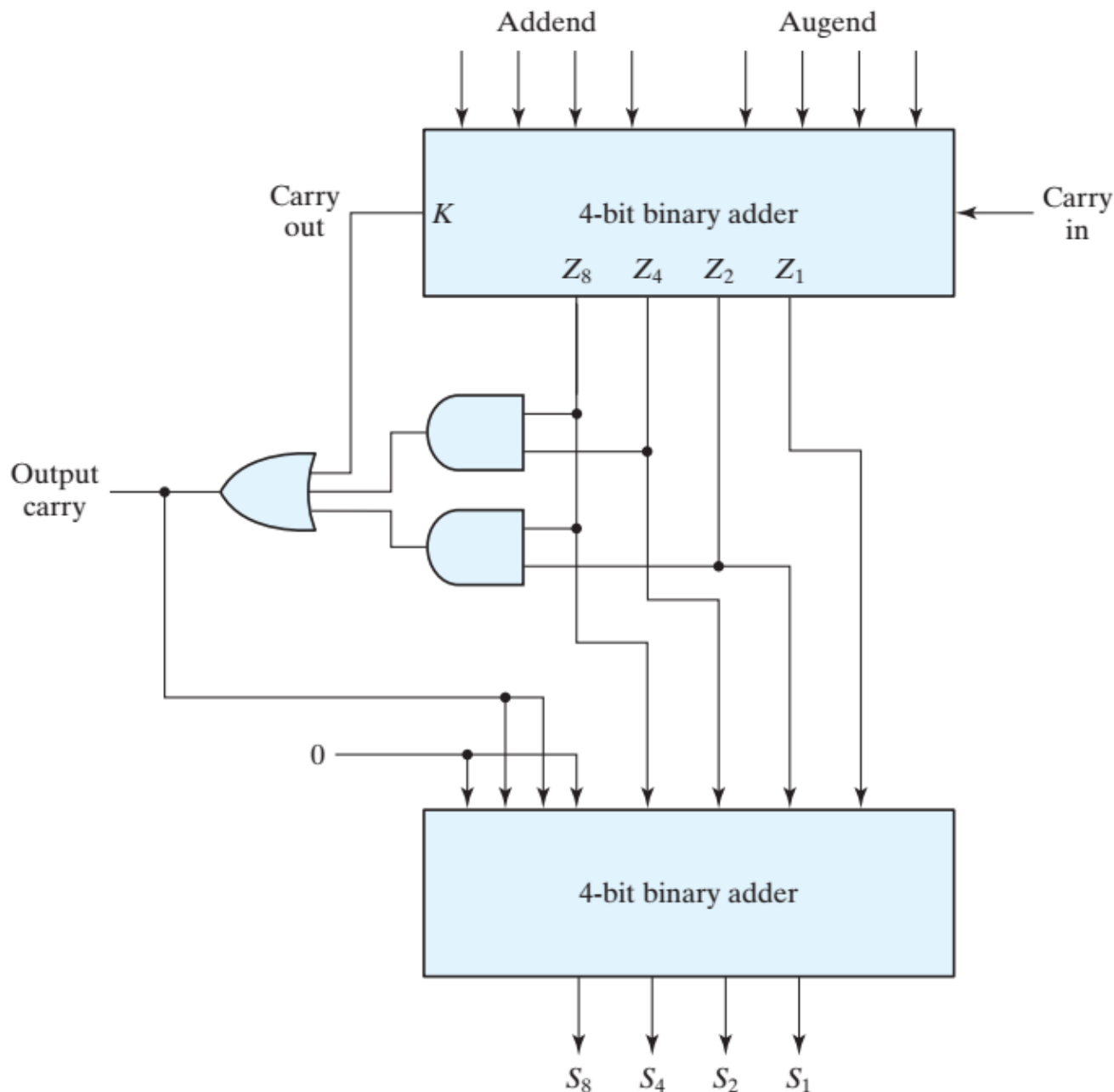


## (BCD Adder) ✓

❖ برای جمع کردن  $n$  رقم دهمی، به  $n$  طبقه از این نوع جمع کننده نیاز داریم.

❖ بیت carry خروجی هر طبقه باید به carry ورودی مرحله بعدی متصل شود.

❖ تمرین: تاخیر کل را بیابید.

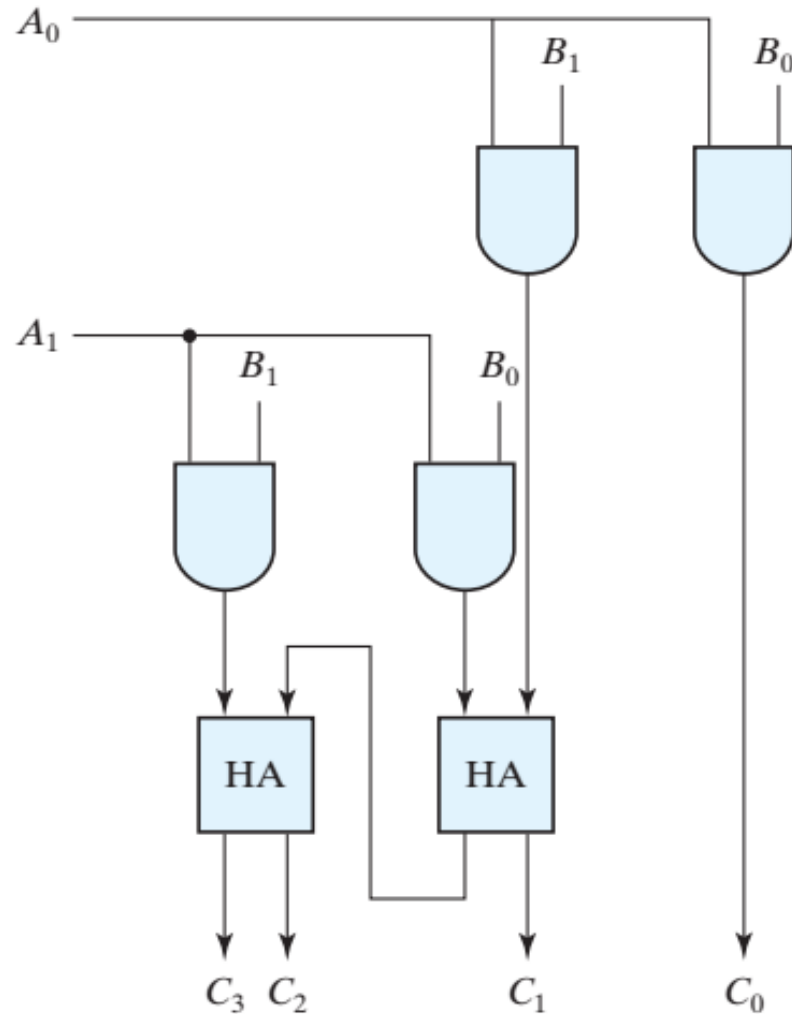




## ✓ ضرب کننده باینری (Binary Multiplier)

❖ ضرب باینری دقیقاً مثل ضرب دو عدد دهدهی محاسبه می شود.

$$\begin{array}{r} B_1 \quad B_0 \\ A_1 \quad A_0 \\ \hline A_0 B_1 \quad A_0 B_0 \\ \\ A_1 B_1 \quad A_1 B_0 \\ \hline C_3 \quad C_2 \quad C_1 \quad C_0 \end{array}$$



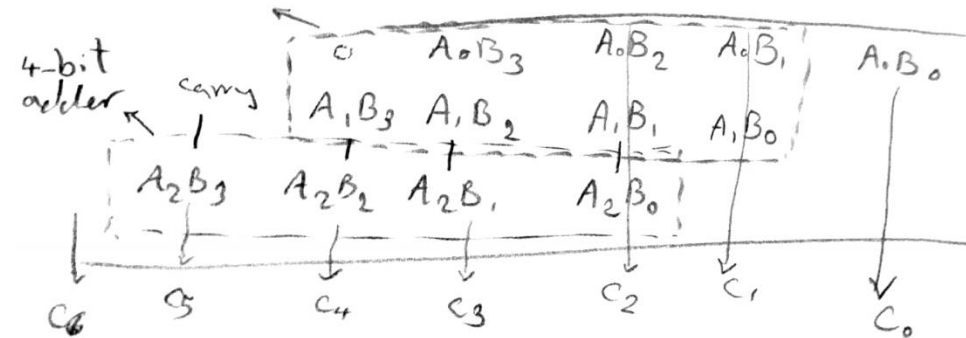
## ✓ ضرب کننده باینری

❖ برای تعداد بیت های بیشتر باید از FA استفاده کرد.

$B_3 \quad B_2 \quad B_1 \quad B_0$

4-bit adder

$A_2 \quad A_1 \quad A_0$



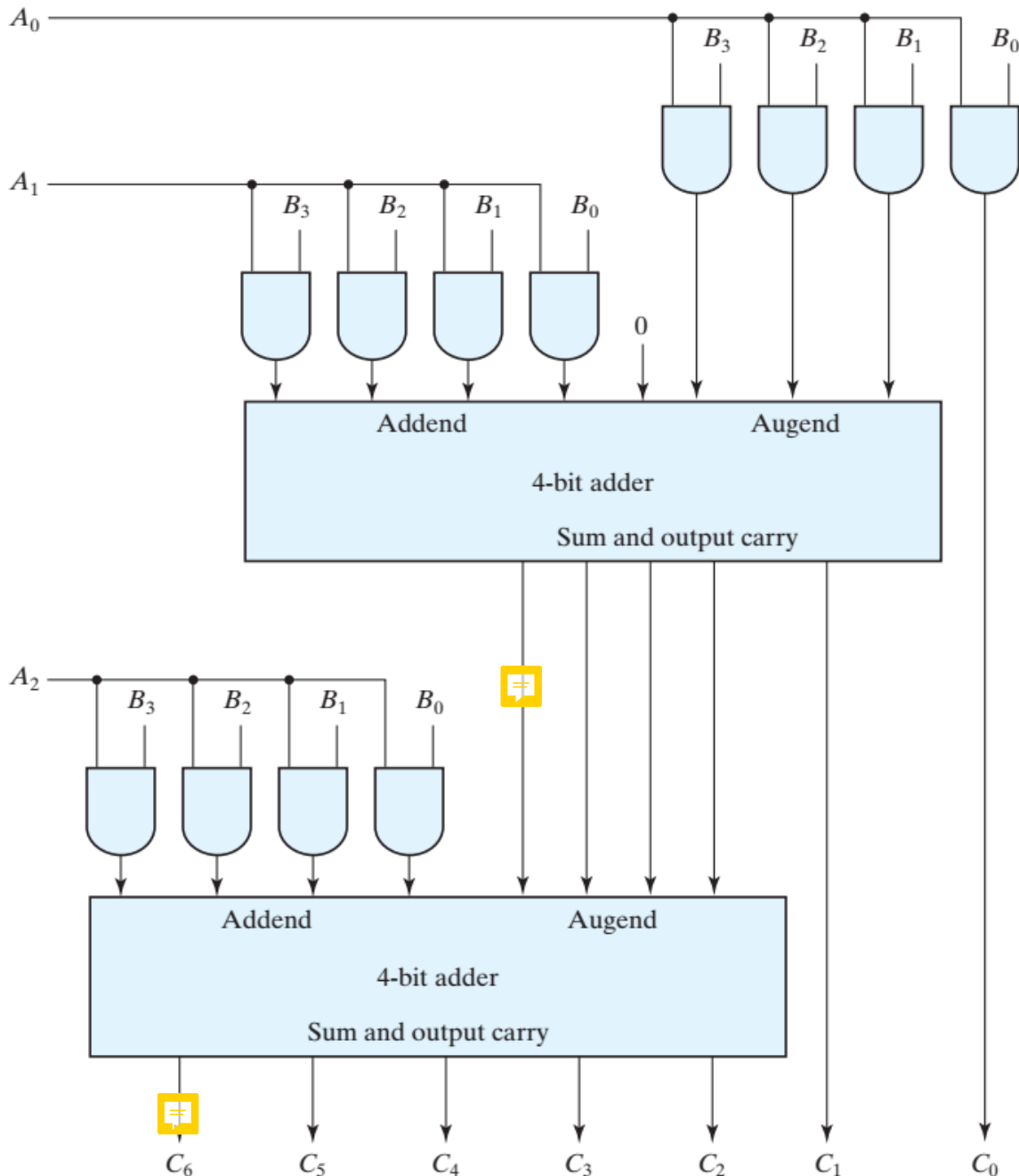
❖ برای ضرب یک عدد  $M$  بیتی در  $N$  بیتی ( $M > N$ ),

نیاز به  $M \times N$  تا گیت AND و  $N-1$  تا جمع کننده

$M$  بیتی داریم.

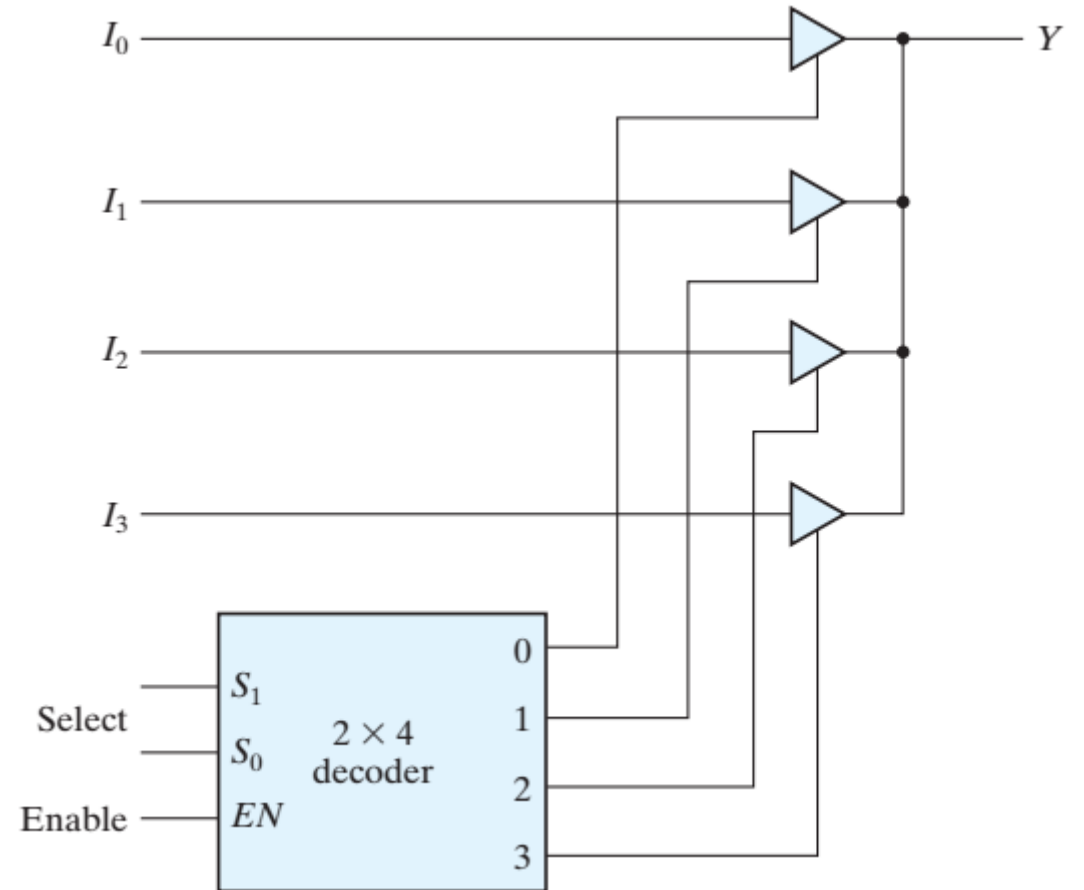
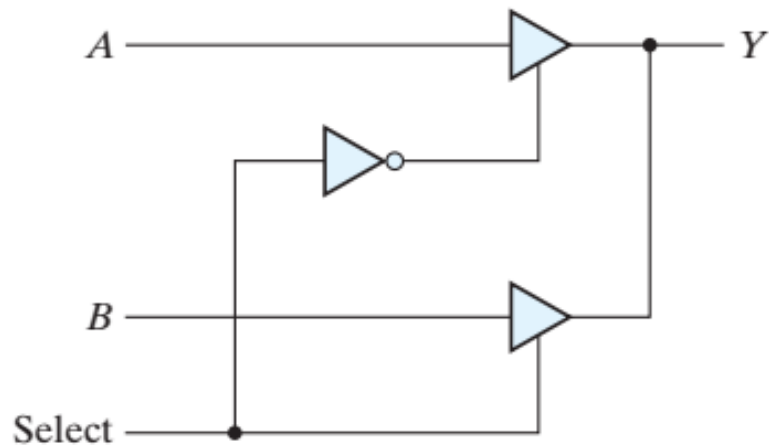
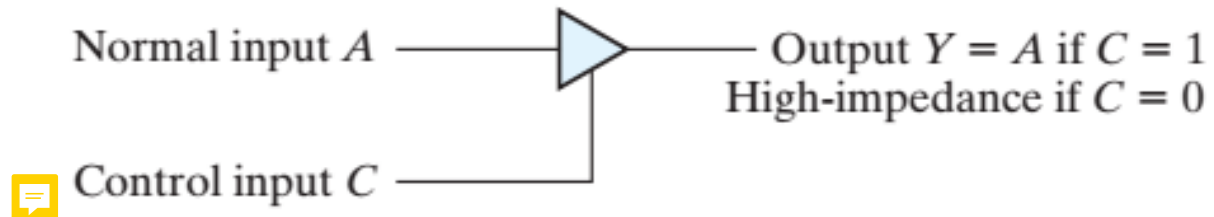
❖ تمرین: تاخیر را برای حالت های مختلف

جمع کننده ها محاسبه کنید.



## ✓ بافرهای سه حالت (Three-State Buffers)

❖ دارای سه حالت می باشند و برای ساخت مالتی پلکسر استفاده می شوند.



## ✓ خلاصه

❖ component های معرفی شده در این فصل:

- ✓ Decoder
- ✓ Encoder
- ✓ MUX / DeMUX
- ✓ Adder / Subtractor
- ✓ Comparator
- ✓ BCD-to-7Segment
- ✓ Bin-to-BCD
- ✓ Multiplier