Chap 4
Combinational Logic (2)

Hanwool Jeong

hwjeong@kw.ac.kr

Revisit Binary Coded Decimal (BCD)

- How many bits do you need for representing one digit of decimal number ***X***?

Decimal Number X	***BCD***			
	A	***B***	***C***	***D***
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

How About Multiple Digit Case?

- Can you represent 25 using BCD system?

Decimal Number X	***BCD***			
	A	***B***	***C***	***D***
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Now, Decimal Adder

- Consider the arithmetic addition of two one-digit decimal numbers in BCD – A and B – with a carry from a previous stage.
- Note that the output should be expressed in BCD.
- It should be able to perform A + B + C_{in} = 7 + 9 + **1** = **1**7, i.e.,

$$0111 + 1001 + \mathbf{1} = \mathbf{1}0111$$

- It is different form the binary addition,

$$0111 + 1001 + 1 = 10001$$

Table 4.5
Derivation of 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		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

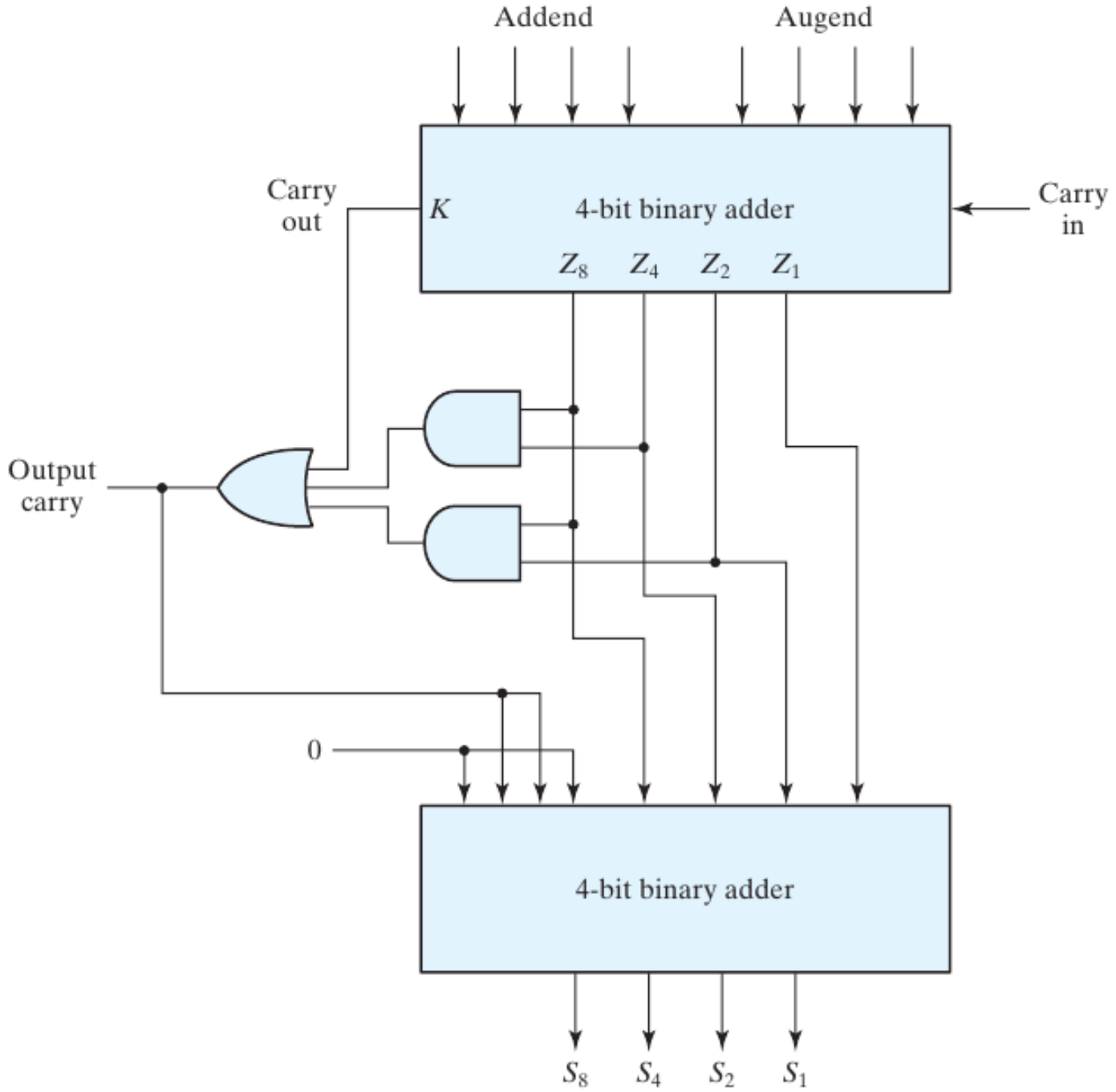


For What Cases Needed Conversion?

- We first perform binary addition first then convert in a certain manner when it is needed
- **When?** We generate the signal “C” for those cases. That is,

C =

- **How?** when C =1, the data is converted with adding 0110.
- Combining above two, we can add 0CC0 to the binary results.


Addend
Augend
Carry
out
K
4-bit binary adder
Carry
in
Z_8 Z_4 Z_2 Z_1
Output
carry
0
4-bit binary adder
S_8 S_4 S_2 S_1

Binary Multiplier

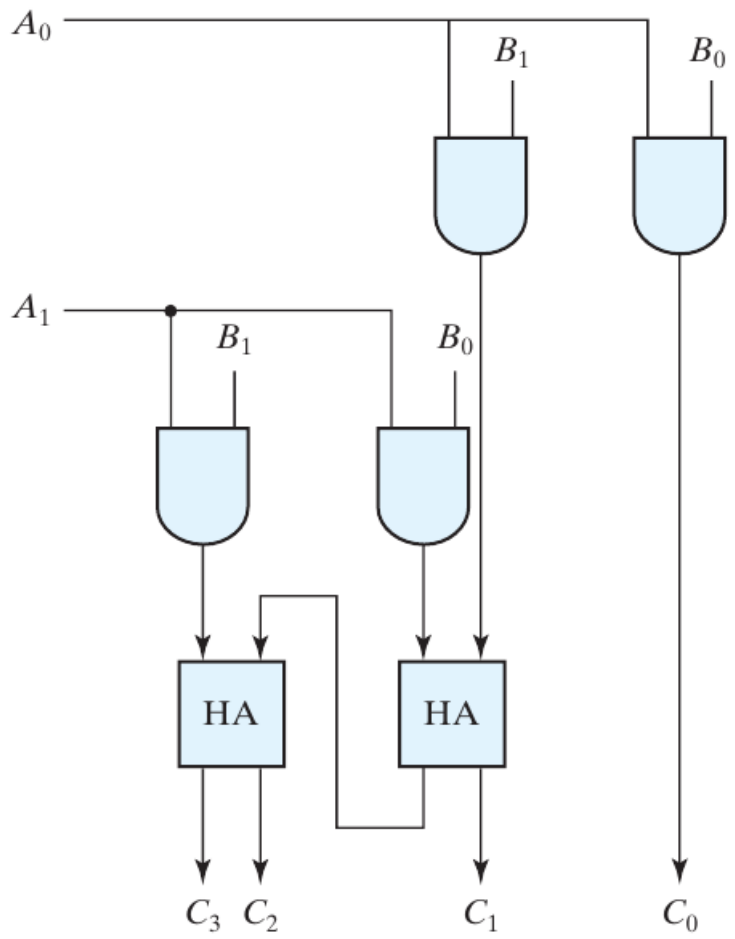
- How do you perform 101 × 110?

- In general, to implement the following 2bit x 2bit multiplication, we need four ANDs, and adders.

$$
\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\hline
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}
$$

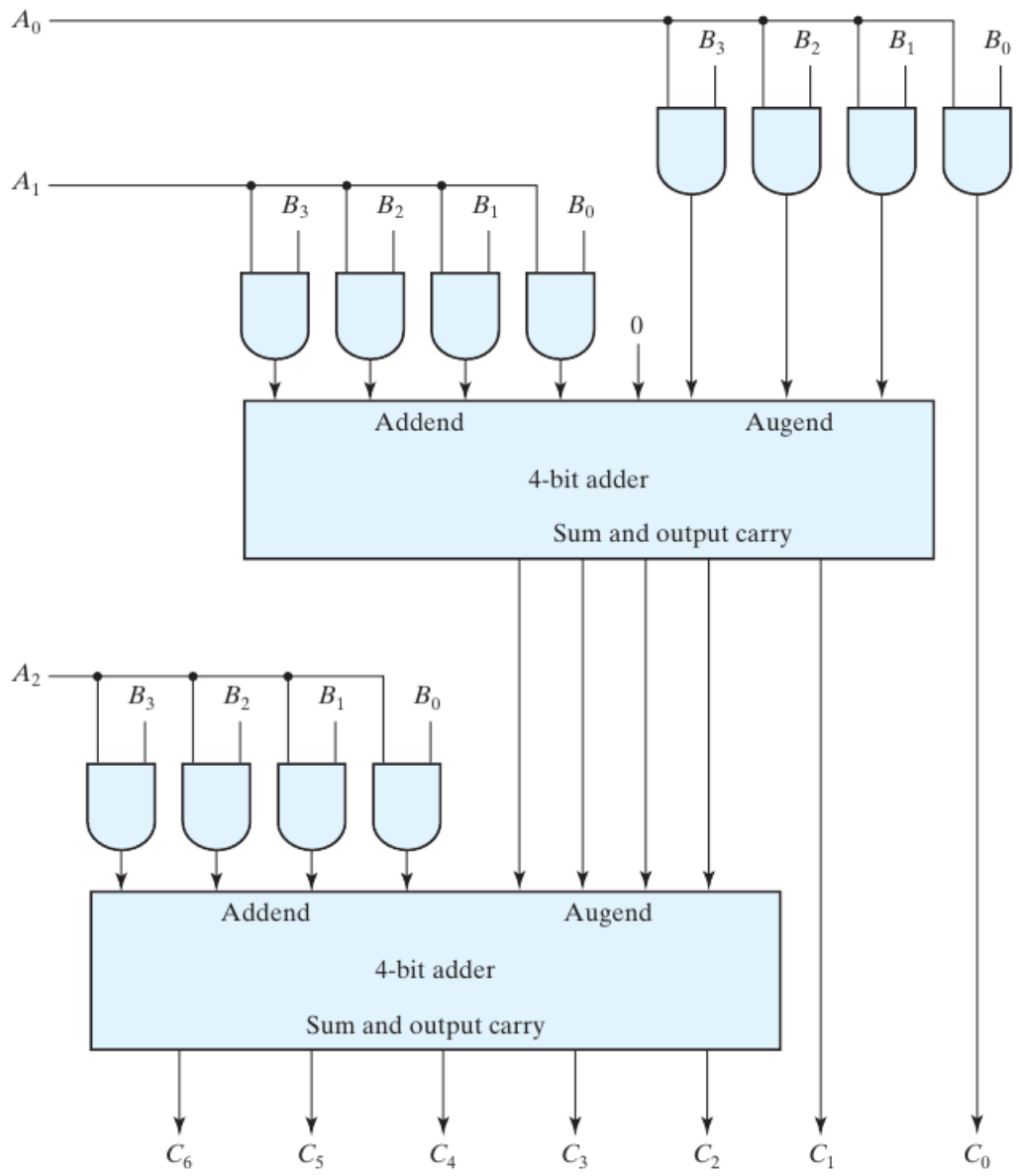
Gate Implmentation

$$
\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\hline
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}
$$



4-bit × 3-bit Multiplier

		B_3	B_2	B_1	B_0
			A_2	A_1	A_0
		A_0B_3	A_0B_2	A_0B_1	A_0B_0
	A_1B_3	A_1B_2	A_1B_1	A_1B_0	
A_2B_3	A_2B_2	A_2B_1	A_2B_0		



Magnitude Comparator (cont'd)

- Suppose a system compare the magnitude of two 4 bit inputs
- How should be output look like?

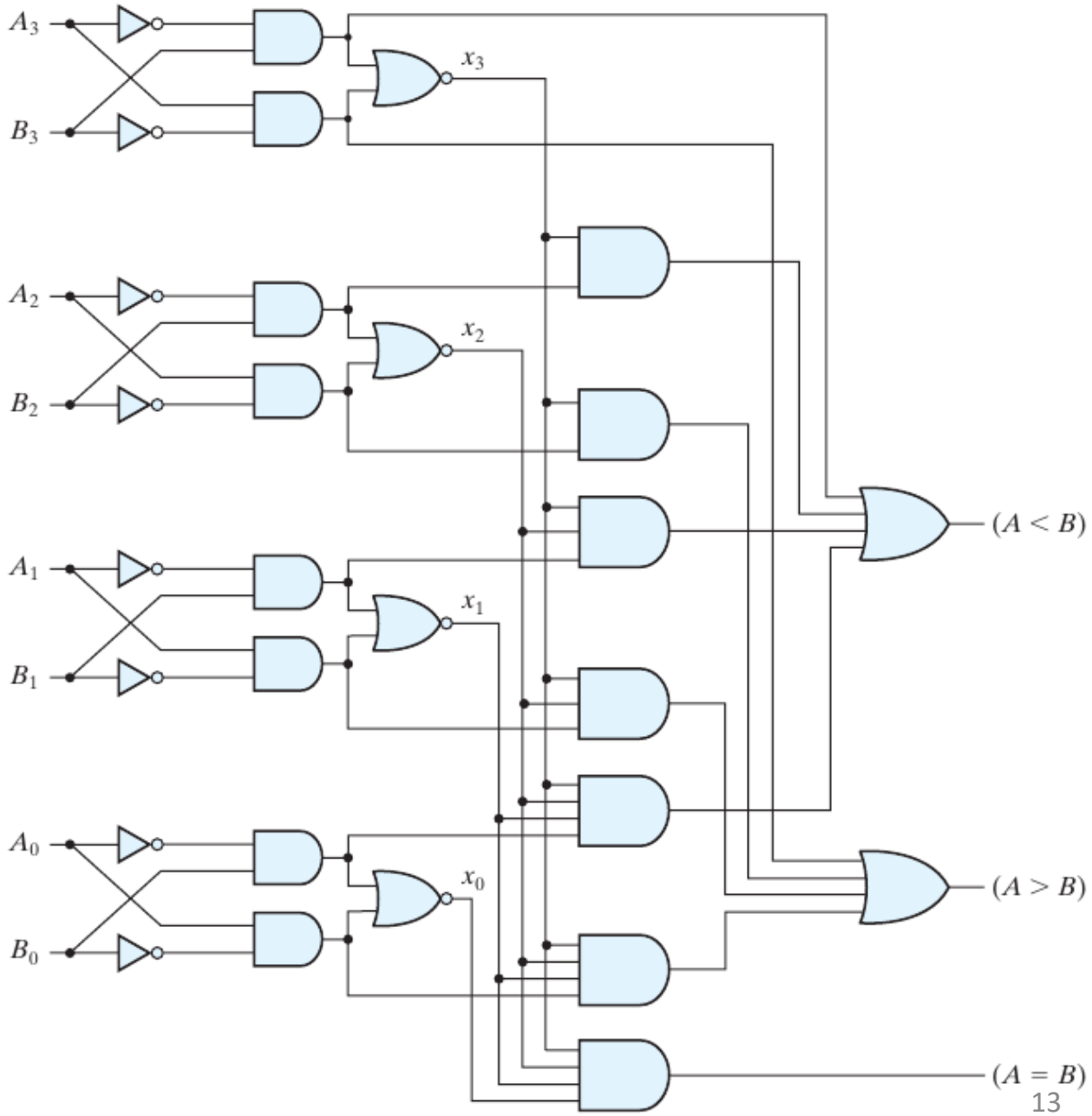
- You remember XNOR2 = 1 only when two inputs are same!
- For (A=B), bit by bit XNOR, x_i, should be one for all bit (all i's)

Magnitude Comparator (cont'd)

- How about (A>B) or (A<B)? Consider how YOU will do if two binary numbers are given.

- It can be obtained through bit by bit comparison from the higher significant bits.
- That is,

$$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$
$$(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1' + x_3x_2x_1A'n_0B_0'$$

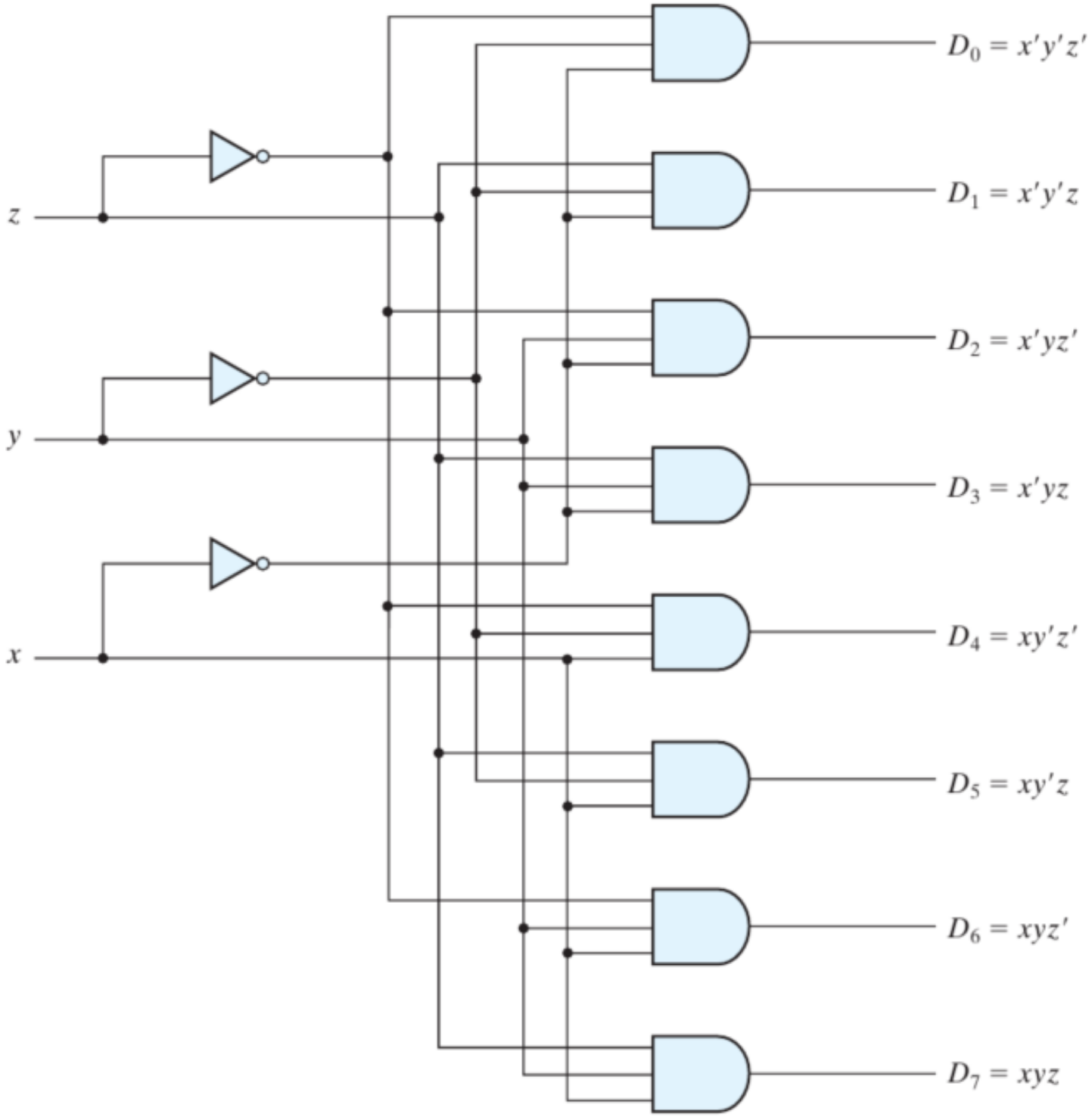


Decoders

- With n bits, 2^n distinct elements can be represented
- 3-to-8 Decoder example

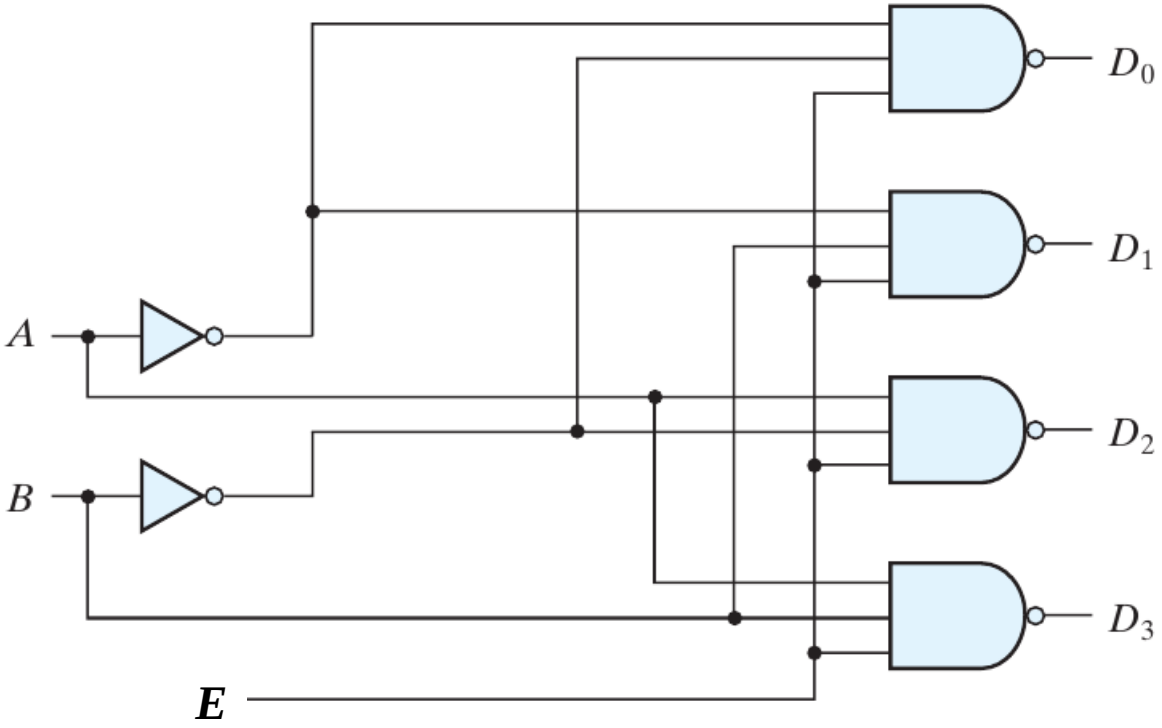
Table 4.6
Truth Table of a Three-to-Eight-Line Decoder

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


z
y
x
$D_0 = x'y'z'$
$D_1 = x'y'z$
$D_2 = x'yz'$
$D_3 = x'yz$
$D_4 = xy'z'$
$D_5 = xy'z$
$D_6 = xyz'$
$D_7 = xyz$

Decoder w/NAND or Enable Inputs

- Can be implemented with NAND3
- Now assume that the output is low enable.



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

Compare
3 to 8 Decoder vs. 2 to 4 Decoder

- You should know how to utilize what you have!

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

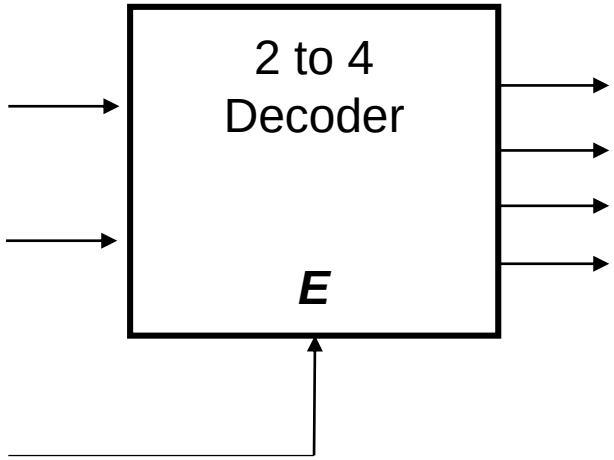
We have this! Can you utilize this?

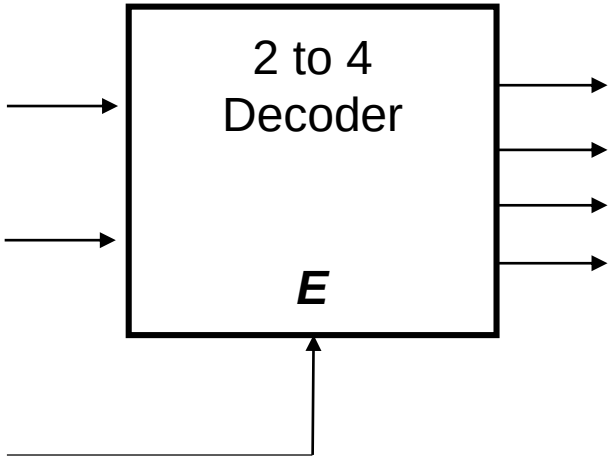
E	A	B	D_0	D_1	D_2	D_3
0	X	X	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	0	1	1
1	1	0	1	1	0	1
1	1	1	1	1	1	0

vs.

A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

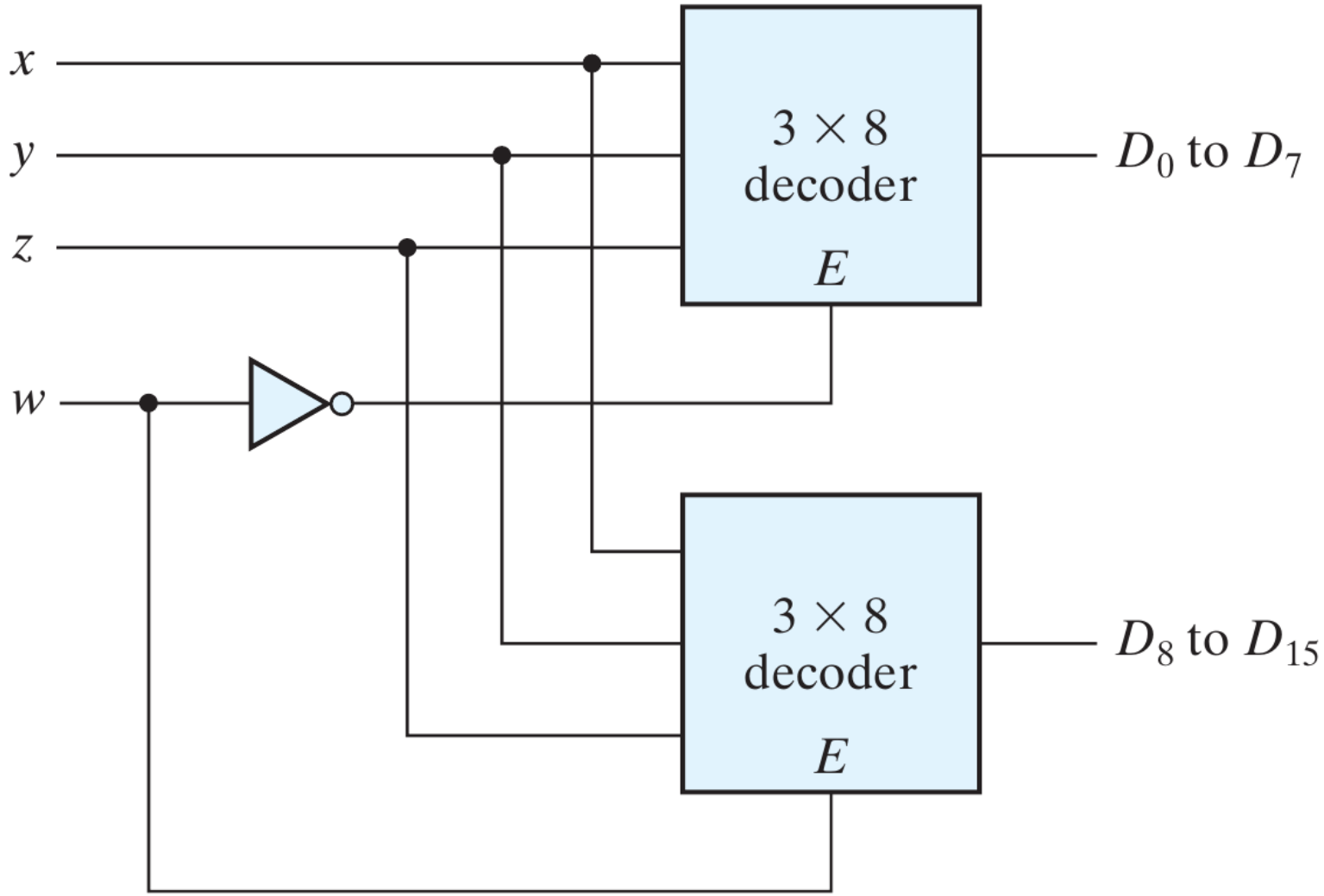
Can you Make 3 to 8 Dec Utilizing 2 to 4 Dec w/ Enable Signal?





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

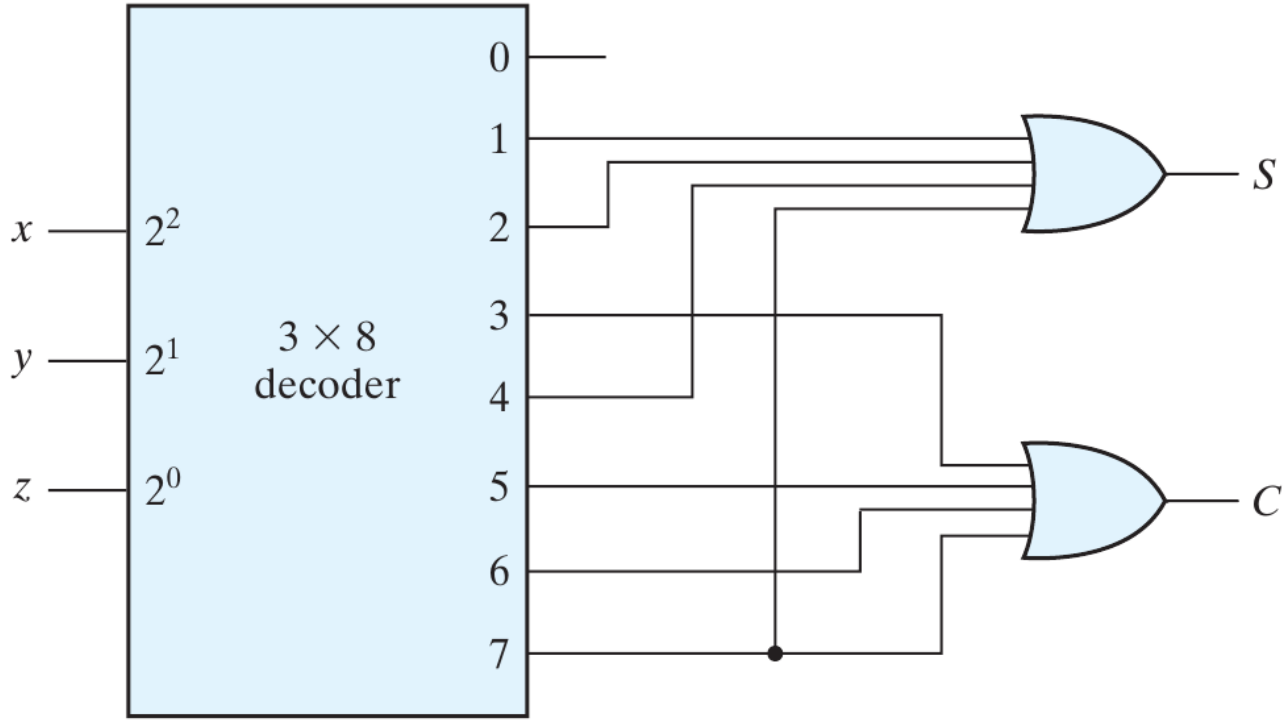
4 to 16 Decoder w/ Two 3 to 8 Decoders



Combinational Logic Implementation

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

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



Encoders

- Inverse operation of Decoder as (but not limited to) follows

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

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

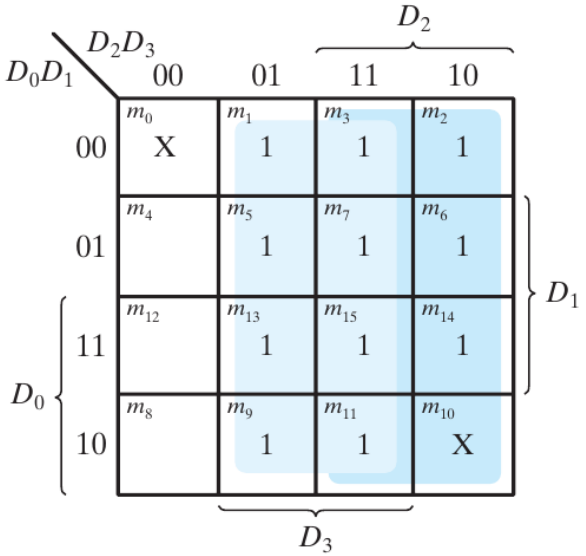
- However, it has a ambiguity as
  - D_3 and D_6 are 1 simultaneously, the output will be 111
  - When all inputs are 0, the output will be 000

Implementing Priority Encoding

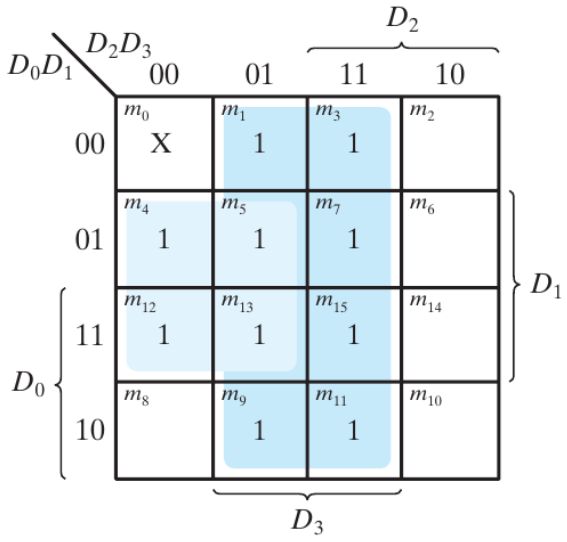
- In 4-to-2 encoding Example,

Inputs				Outputs	
D_0	D_1	D_2	D_3	x	y

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

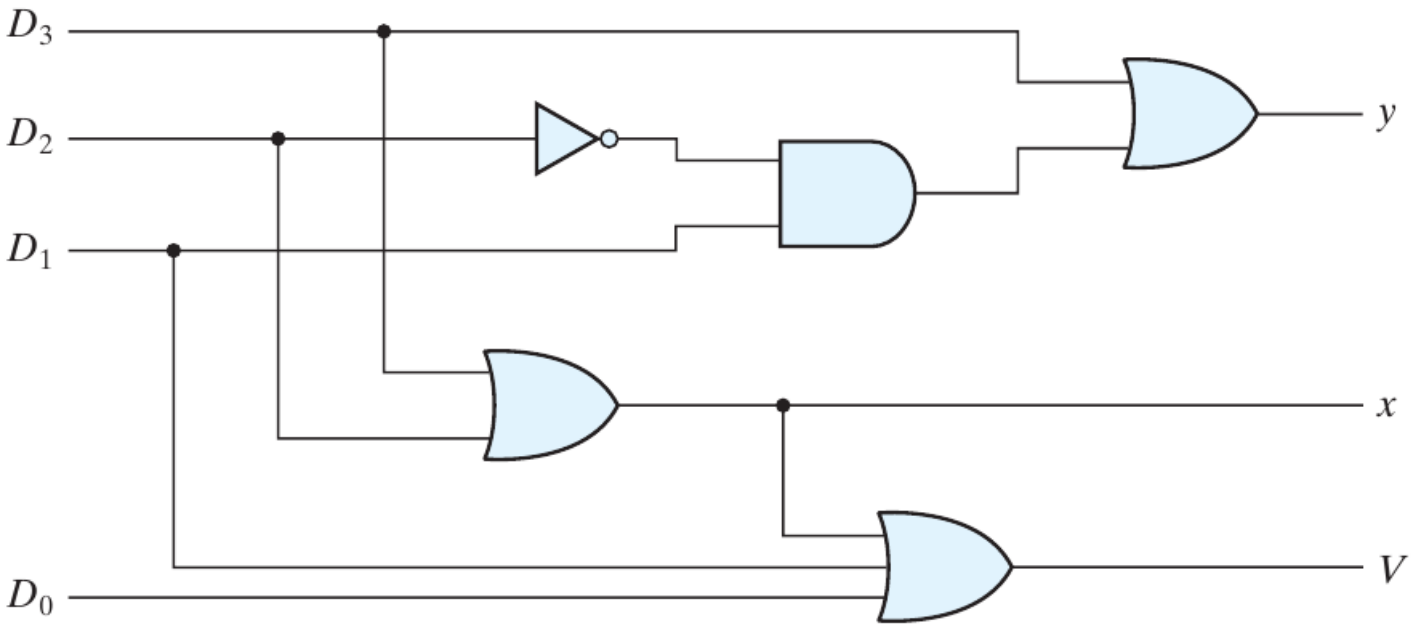


$x = D_2 + D_3$



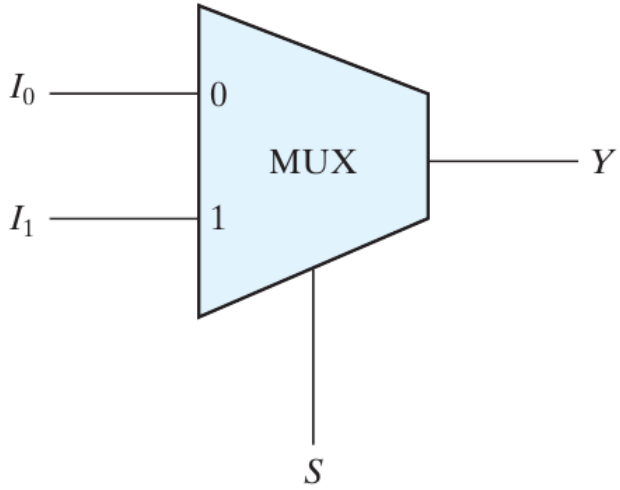
$y = D_3 + D_1D'_2$

Gate Implementation



Multiplexer

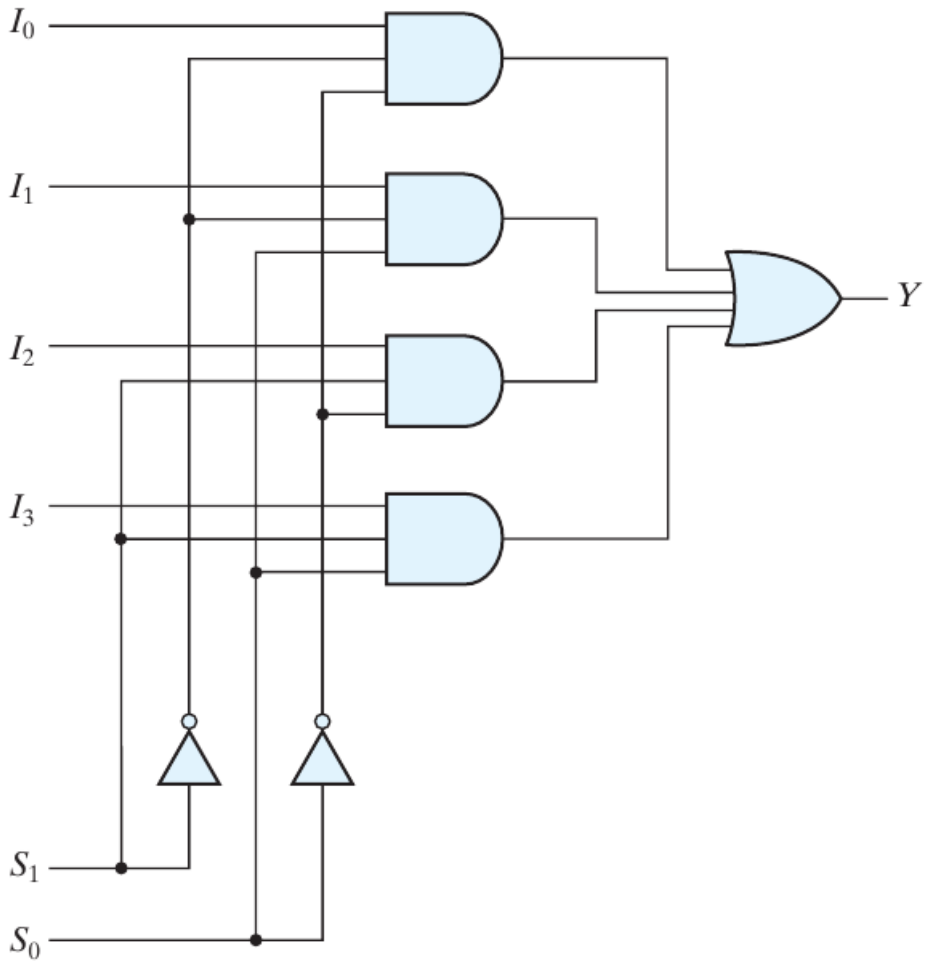
- 2-tio-1 MUX Selects between I_0 and I_1 according to S



- Can you derive truth table or boolean function?

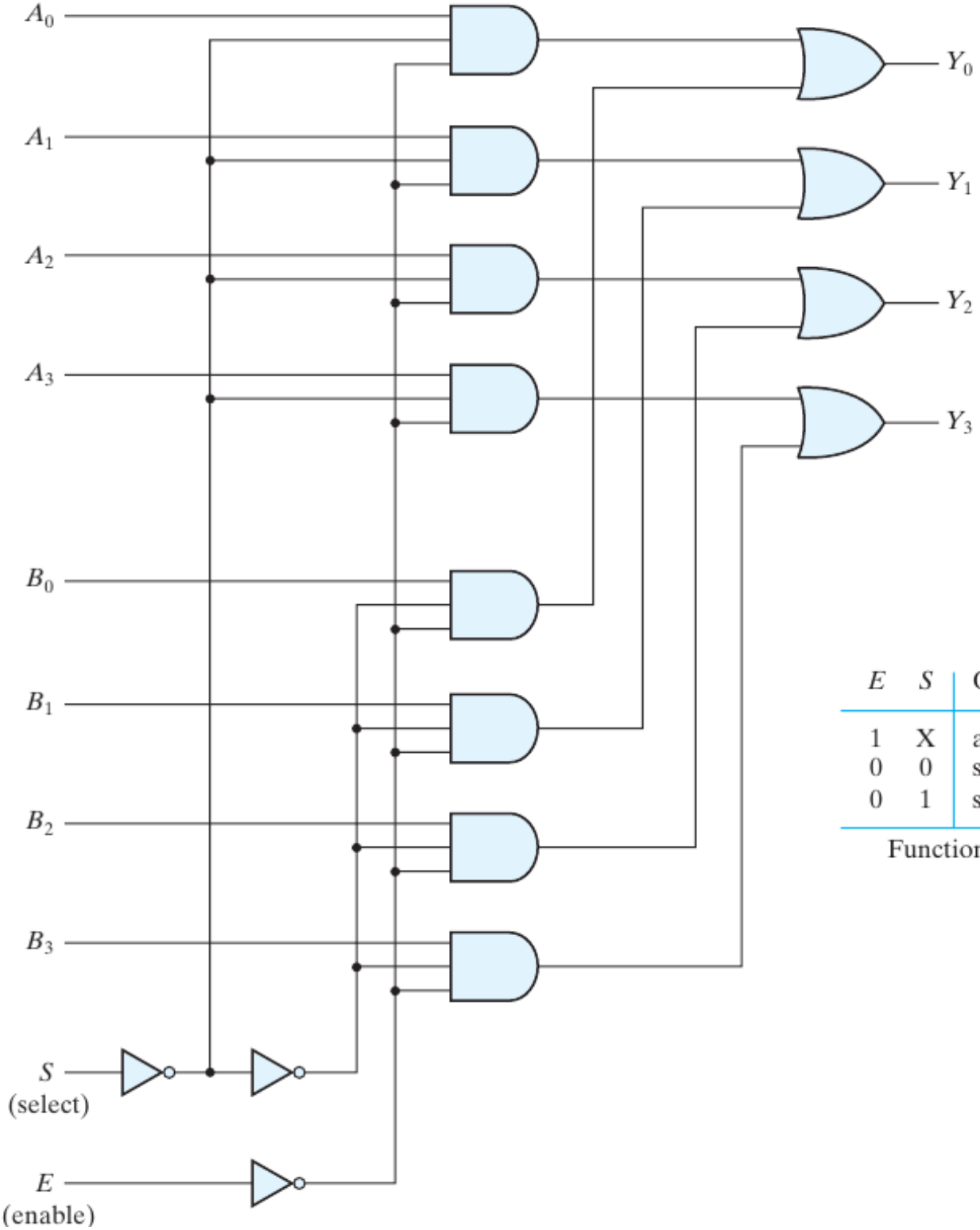
- How about 4-to-1 MUX?

Logic Diagram of 4-to-1 MUX



Quadruple 2-to-1 MUX

- There are two inputs A and B with 4 bits. 2-to-1 MUX is used to select between A and B according to S
- How does the circuit look like?



E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

Function table

Boolean Function Implementation with MUXes

- Can you realize the following function with MUX?

$$F(x, y, z) = \Sigma(1, 2, 6, 7)$$

- What type of MUX we need?
- See the truth table. If we see the x, y as select bits of 4-to-1 MUX, we can implement the above function.

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Example

- F(A,B,C,D) = Σ(1, 3, 4, 11, 12, 13, 14, 15)

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

