

CS2100

COMPUTER ORGANISATION

<http://www.comp.nus.edu.sg/~cs2100/>

Lecture #18

MSI Components



NUS
National University
of Singapore

School of
Computing

Lecture #18: MSI Components

1. Introduction
2. Decoders
3. Encoders
4. Demultiplexers
5. Multiplexers

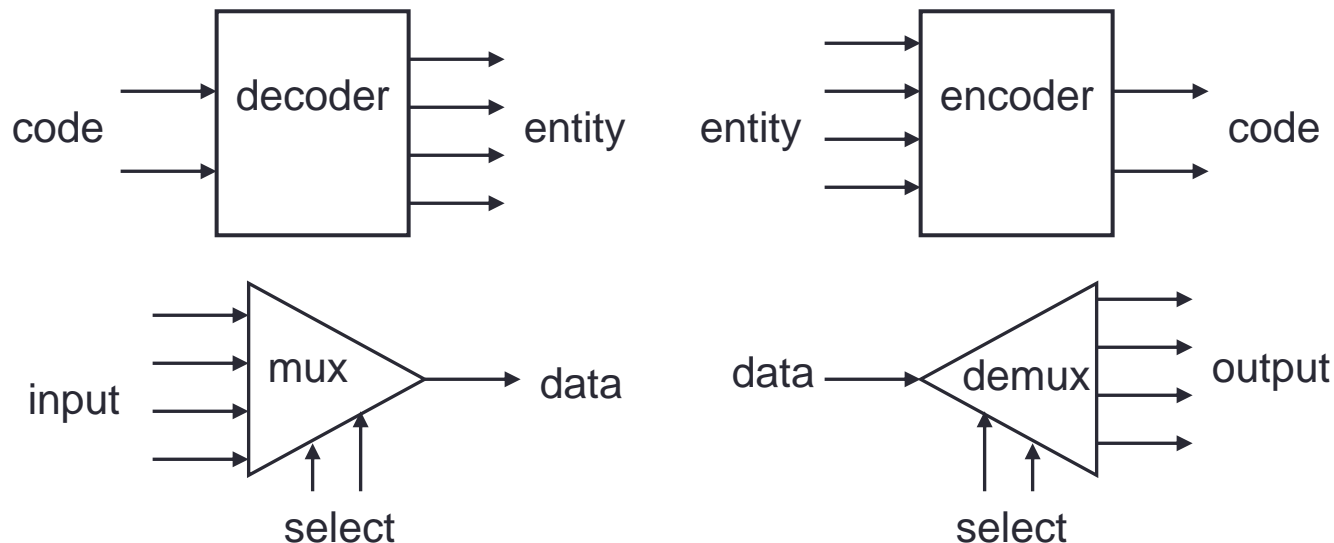
1. Introduction (1/2)

- An **integrated circuit** (referred to as an **IC**, a **chip** or a **microchip**) is a set of electronic circuits on one small flat piece (or 'chip') of semiconductor material.
- **Scale of integration**: the number of components fitted into a standard size IC

Name	Signification	Year	#transistors	#logic gates
SSI	Small-scale integration	1964	1 to 10	1 to 12
MSI	Medium-scale integration	1968	10 to 500	13 to 99
LSI	Large-scale integration	1971	500 to 20000	100 to 9999
VLSI	Very large-scale integration	1980	20k to 1m	10k to 99999
ULSI	Ultra-large-scale integration	1984	1m and more	100k and more

1. Introduction (2/2)

- Four common and useful MSI circuits:
 - Decoder
 - Demultiplexer
 - Encoder
 - Multiplexer
- Block diagrams of the above MSI circuits:

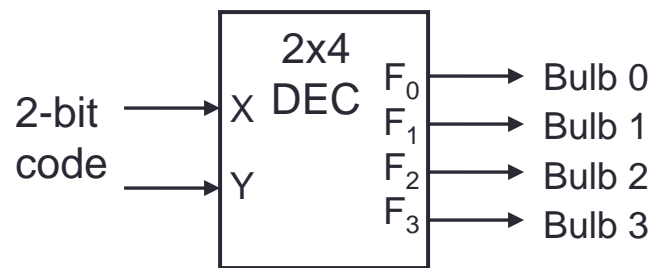


2. Decoders (1/5)

- Codes are frequently used to represent entities, eg: your name is a code to denote yourself (an entity!)
- These codes can be identified (or decoded) using a decoder. Given a code, identify the entity.
- Convert binary information from n input lines to (a maximum 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 minterms of n input variables.

2. Decoders (2/5)

- 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.

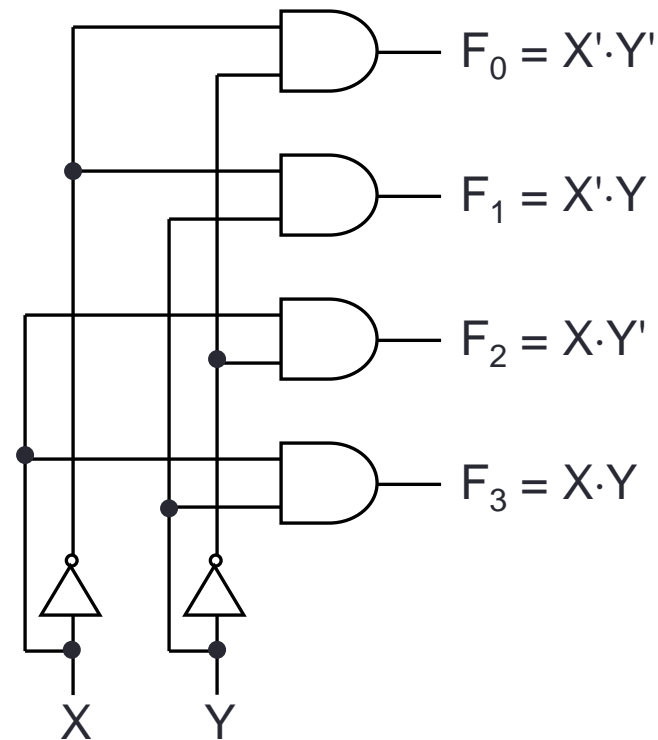
- Truth table:

X	Y	F 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

2. Decoders (3/5)

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

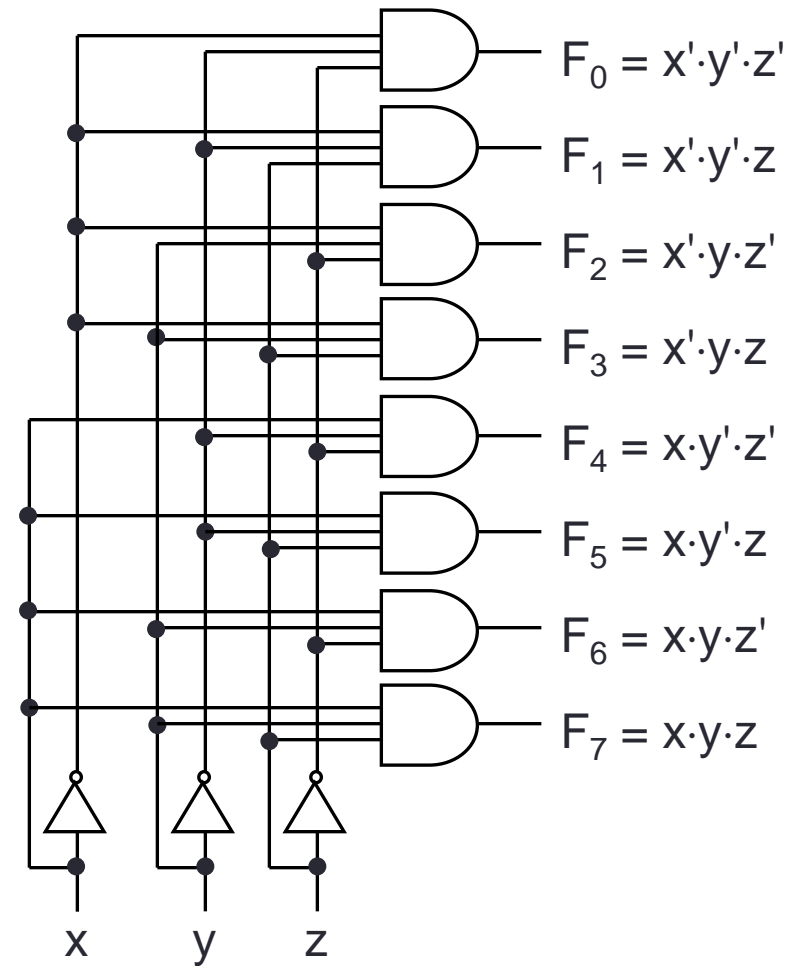
X	Y	F ₀	F ₁	F ₂	F ₃
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



2. Decoders (4/5)

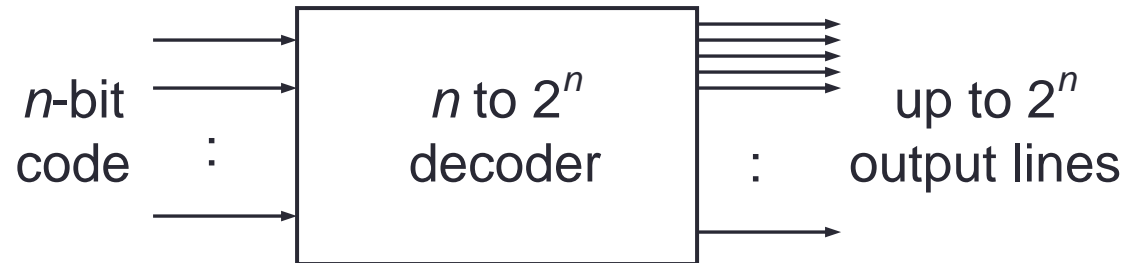
- Design a **3×8 decoder**.

x	y	z	F ₀	F ₁	F ₂	F ₃	F ₄	F ₅	F ₆	F 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



2. Decoders (5/5)

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



2. Decoders: Implementing Functions (1/3)

- A Boolean function, in **sum-of-minterms** form \Rightarrow
 - decoder to generate the minterms, and
 - an OR gate to form the sum.
- Any combinational circuit with n inputs and m outputs can be implemented with an $n:2^n$ decoder with m OR gates.
- Good when circuit has many outputs, and each function is expressed with a few minterms.

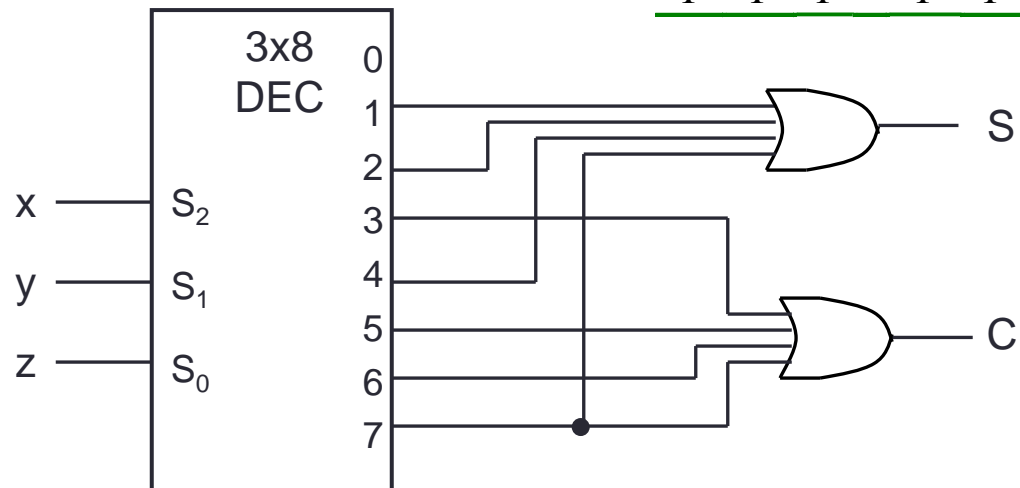
2. Decoders: Implementing Functions (2/3)

- Example: Full adder

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

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

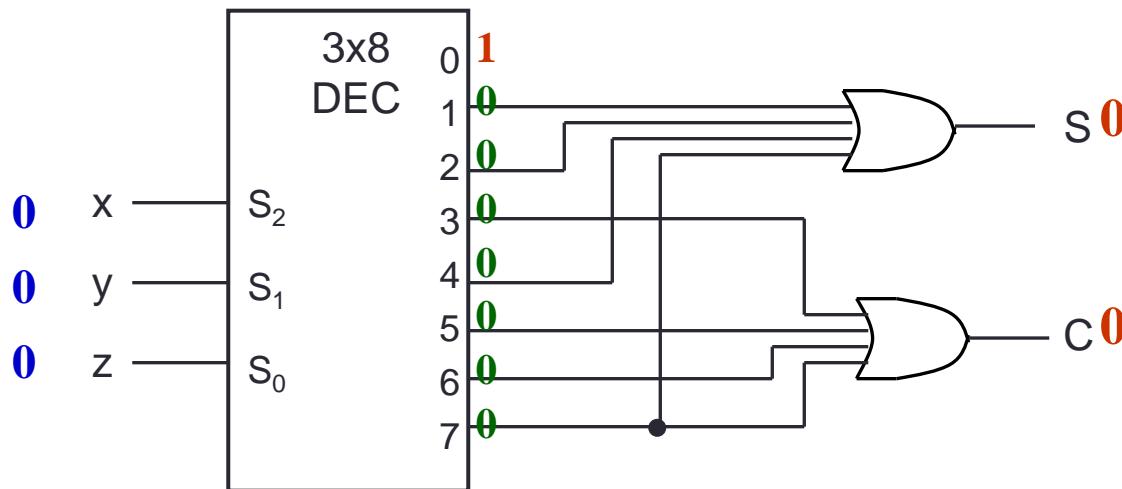
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



2. Decoders: Implementing Functions (3/3)

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

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



x	y	z	C	S
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

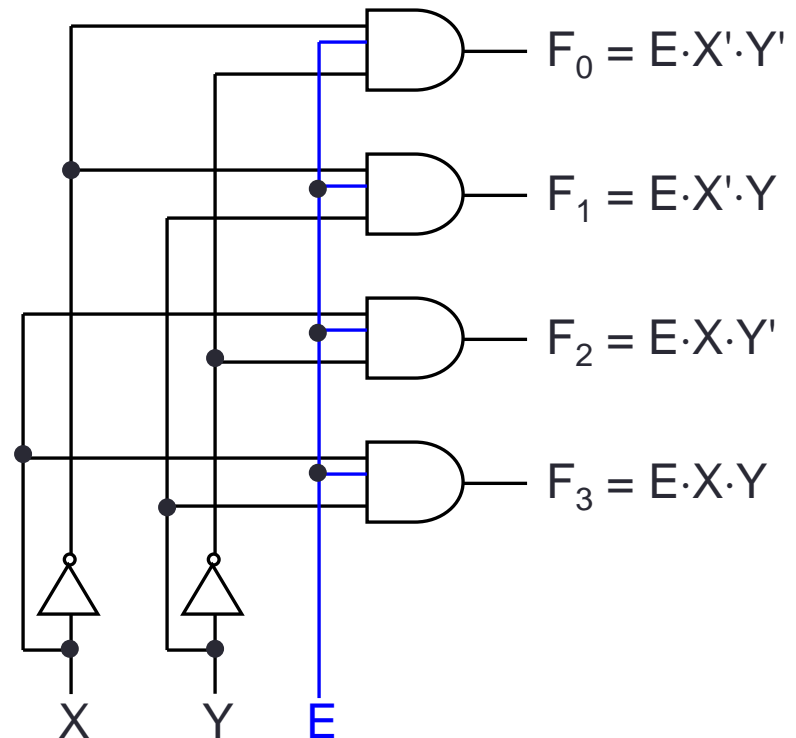
2. Decoders with Enable (1/2)

- Decoders often come with an **enable control** signal, so that the device is only activated when the enable, $E = 1$.

- Truth table:

E	X	Y	F_0	F_1	F_2	F_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	d	d	0	0	0	0

- Circuit of a **2×4 decoder with enable**:



2. Decoders with Enable (2/2)

- In the previous slide, the decoder has a **one-enable** control signal, i.e. the decoder is enabled with $E=1$.
- In most MSI decoders, enable signal is **zero-enable**, usually denoted by E' or \bar{E} . The decoder is enabled when the signal is zero (low).

E	X	Y	F ₀	F ₁	F ₂	F ₃
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	d	d	0	0	0	0

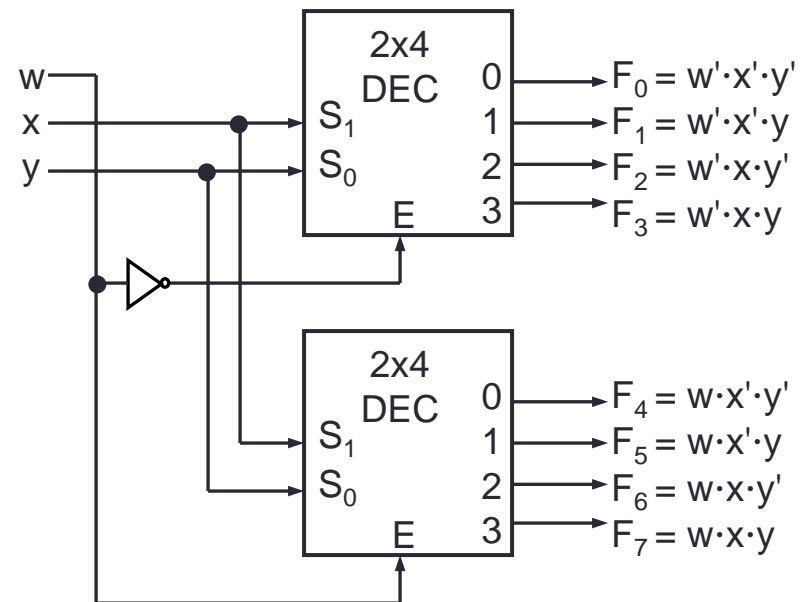
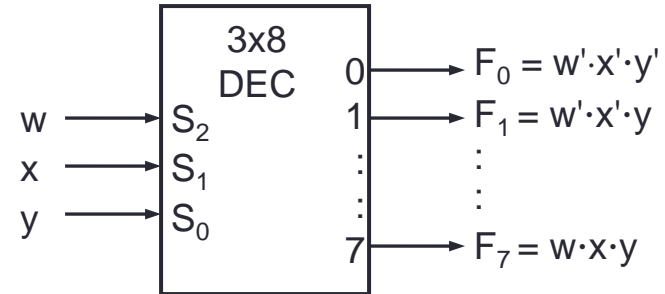
Decoder with 1-enable

E'	X	Y	F ₀	F ₁	F ₂	F ₃
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1
1	d	d	0	0	0	0

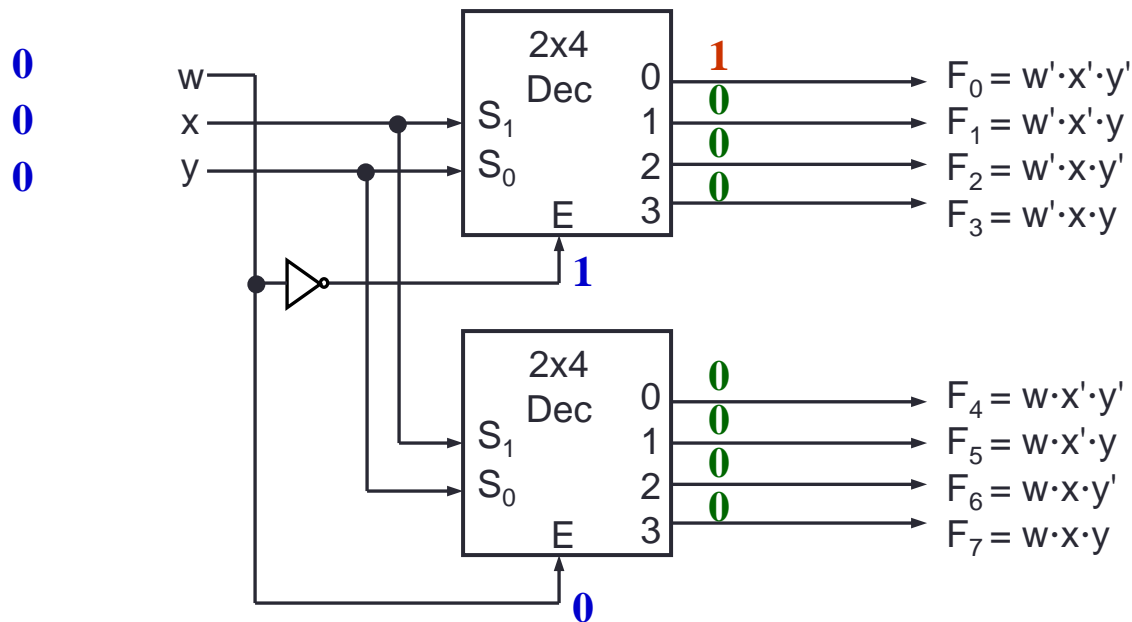
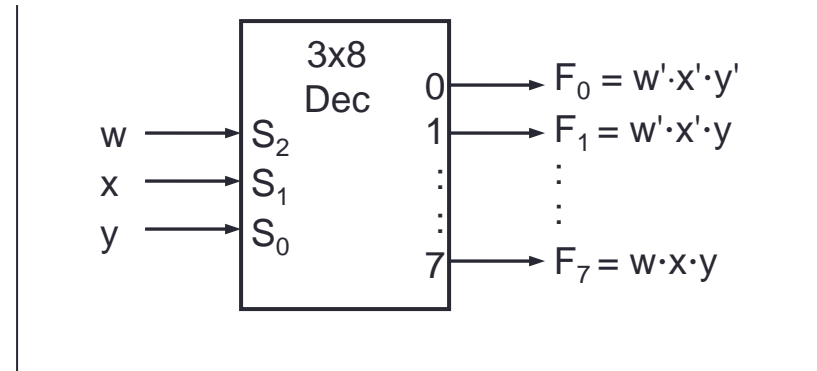
Decoder with 0-enable

2. Constructing Larger Decoders (1/4)

- Larger decoders can be constructed from smaller ones.
- Example: A 3×8 decoder can be built from two 2×4 decoders (with one-enable) and an inverter.

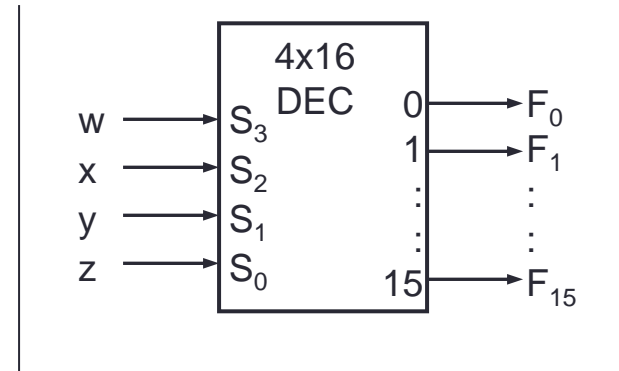
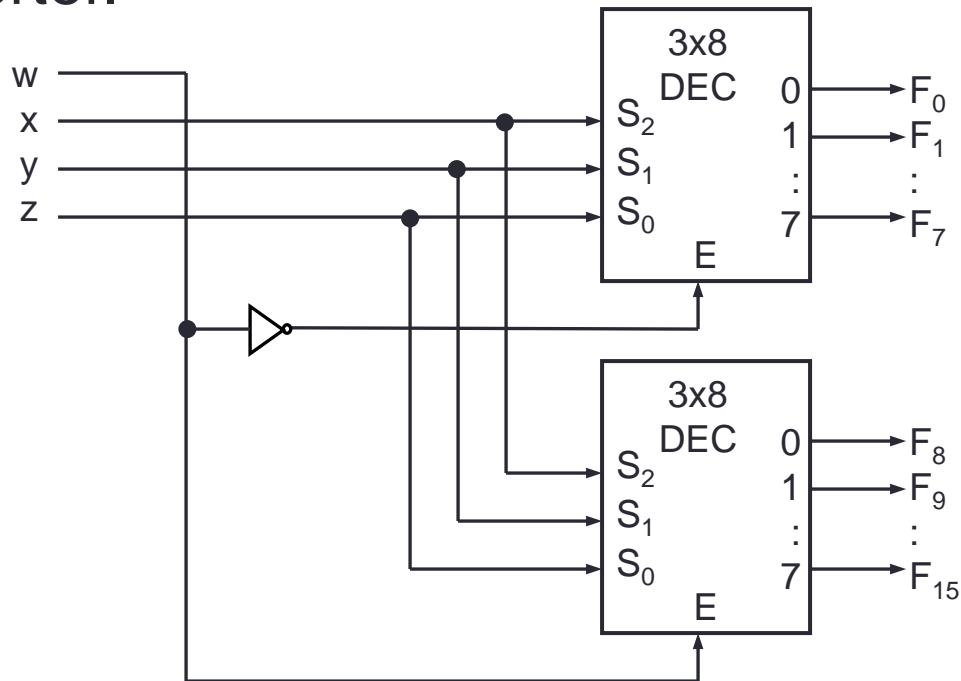


2. Constructing Larger Decoders (2/4)



2. Constructing Larger Decoders (3/4)

- Construct a 4×16 decoder from two 3×8 decoders with one-enable and an inverter.



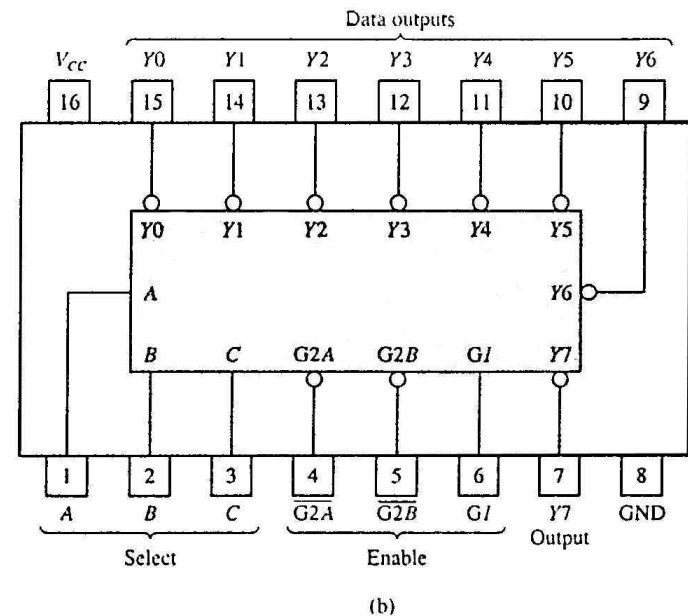
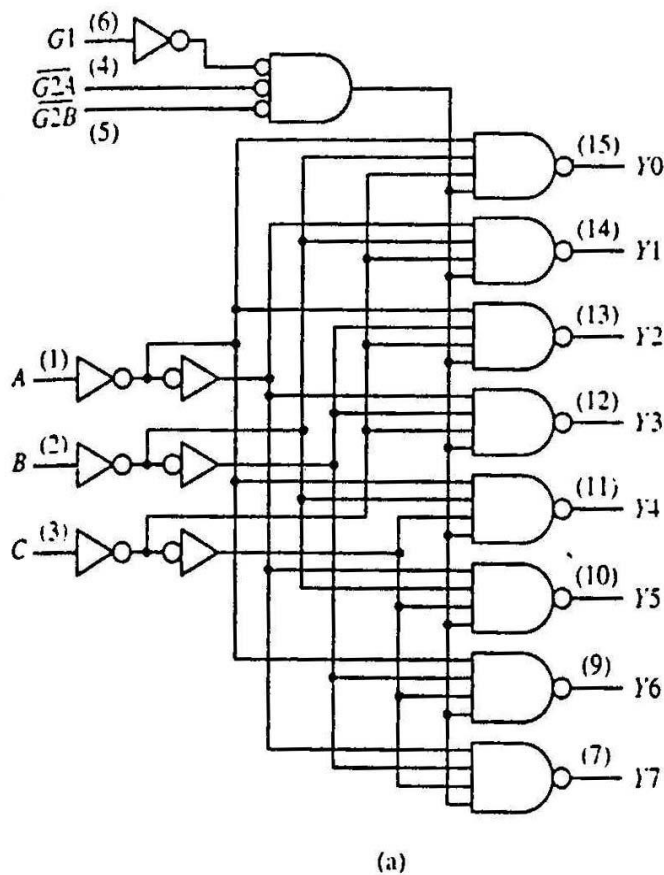
- Note: The input w and its complement w' are used to select either one of the two smaller decoders.

2. Constructing Larger Decoders (4/4)

- **Exercise:** What modifications should be made to provide an ENABLE input for the 3×8 decoder and the 4×16 decoder created in the previous two examples?
- **Exercise:** How to construct a 4×16 decoder using five 2×4 decoders with enable?
- Decoders may also have **zero-enable** and/or **negated outputs**. (See next two slides.)
 - Normal outputs = active high outputs
 - Negated outputs = active low outputs

2. Standard MSI Decoder (1/2)

■ 74138 (3-to-8 decoder)



74138 decoder module.

(a) Logic circuit.

(b) Package pin configuration.

2. Standard MSI Decoder (2/2)

INPUTS					OUTPUTS							
ENABLE		SELECT										
G1	$\overline{G2}^*$	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	L	H	H	H	H	H
H	L	H	L	L	H	H	H	L	H	H	H	H
H	L	H	L	H	H	H	H	L	H	H	H	H
H	L	H	H	L	H	H	H	H	L	H	H	H
H	L	H	H	H	H	H	H	H	H	L	H	H

74138 decoder module.
(c) Function table.

$$*\overline{G2} = \overline{G2A} + \overline{G2B}$$

H = high level, L = low level, X = irrelevant

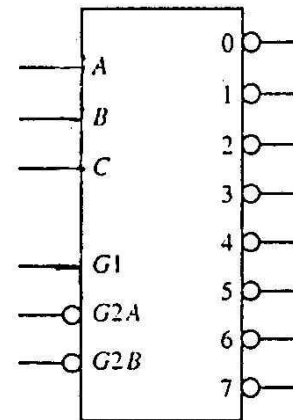
(c)

74138 decoder module.

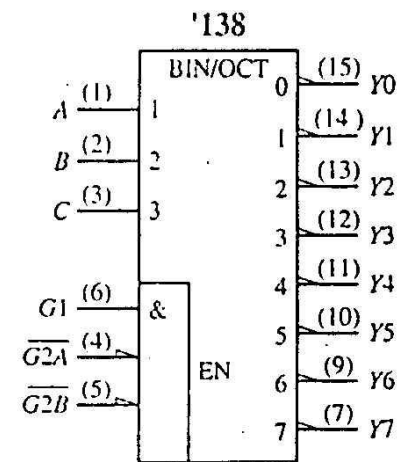
(d) Generic symbol.

(e) IEEE standard logic symbol.

Source: *The Data Book Volume 2, Texas Instruments Inc., 1985*



(d)



(e)

2. Decoders: Implementing Functions Revisit (1/2)

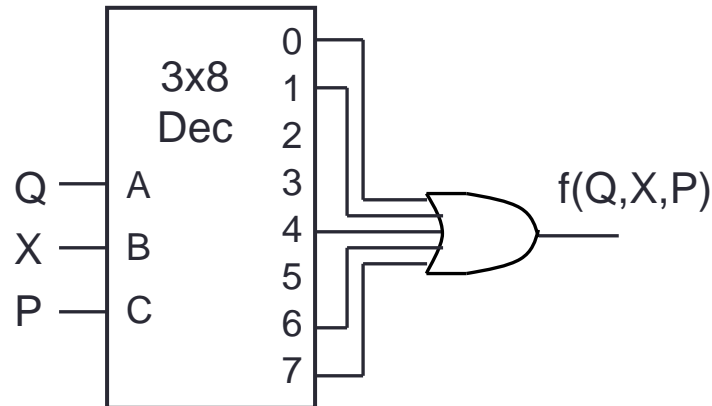
- Example: Implement the following function using a 3×8 decoder and an appropriate logic gate

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

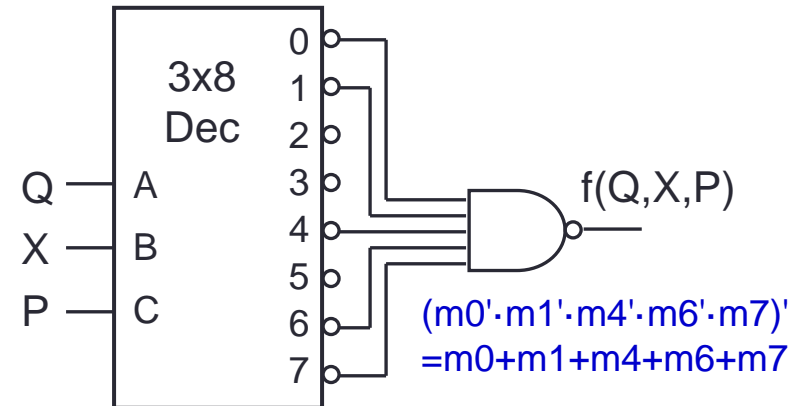
- We may implement the function in several ways:
 - Using a decoder with active-high outputs with an OR gate:
$$f(Q,X,P) = m_0 + m_1 + m_4 + m_6 + m_7$$
 - Using a decoder with active-low outputs with a NAND gate:
$$f(Q,X,P) = (m_0' \cdot m_1' \cdot m_4' \cdot m_6' \cdot m_7')'$$
 - Using a decoder with active-high outputs with a NOR gate:
$$f(Q,X,P) = (m_2 + m_3 + m_5)' [= M_2 \cdot M_3 \cdot M_5]$$
 - Using a decoder with active-low outputs with an AND gate:
$$f(Q,X,P) = m_2' \cdot m_3' \cdot m_5'$$

2. Decoders: Implementing Functions Revisit (2/2)

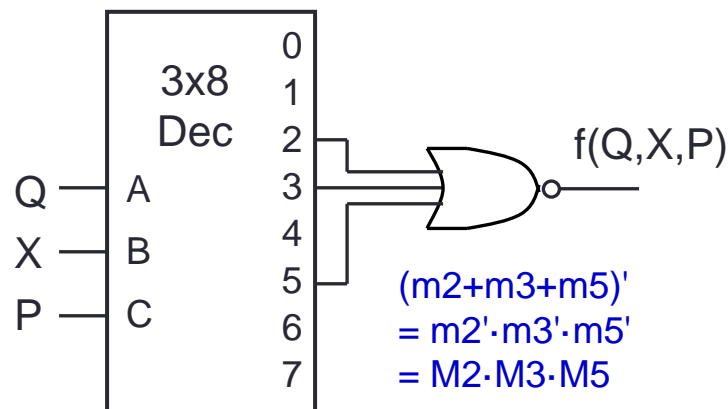
$$f(Q,X,P) = \sum m(0,1,4,6,7) = \prod M(2,3,5)$$



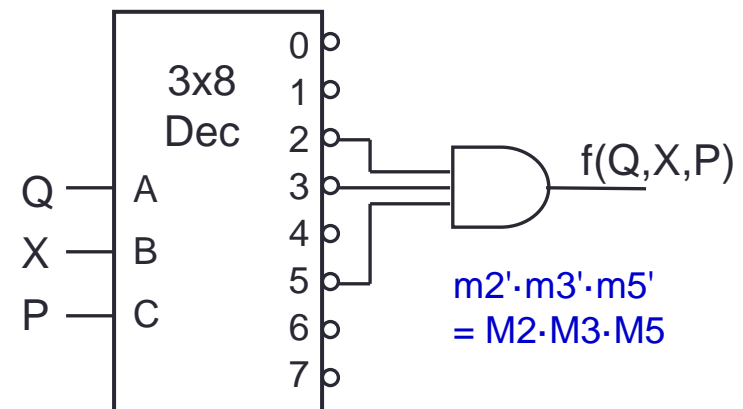
(a) Active-high decoder with OR gate.



(b) Active-low decoder with NAND gate.



(c) Active-high decoder with NOR gate.



(d) Active-low decoder with AND gate.

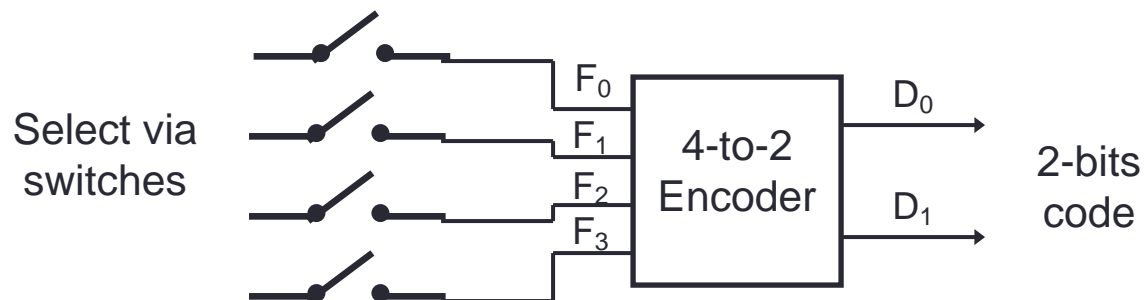
Reading

- **Reducing Decoders**
 - Read up DLD pages 136 – 140.



3. Encoders (1/4)

- **Encoding** is the converse of decoding.
- Given a set of input lines, of which exactly one is high and the rest are low, the **encoder** provides a code that corresponds to that high input line.
- Contains 2^n (or fewer) input lines and n output lines.
- Implemented with OR gates.
- Example:



3. Encoders (2/4)

- Truth table:
- With K-map, we obtain:
 - $D_0 = F1 + F3$
 - $D_1 = F2 + F3$
- Circuit:

?

Simple 4-to-2 encoder

F_0	F_1	F_2	F_3	D_1	D_0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1
0	0	0	0	X	X
0	0	1	1	X	X
0	1	0	1	X	X
0	1	1	0	X	X
0	1	1	1	X	X
1	0	0	1	X	X
1	0	1	0	X	X
1	0	1	1	X	X
1	1	0	0	X	X
1	1	0	1	X	X
1	1	1	0	X	X
1	1	1	1	X	X

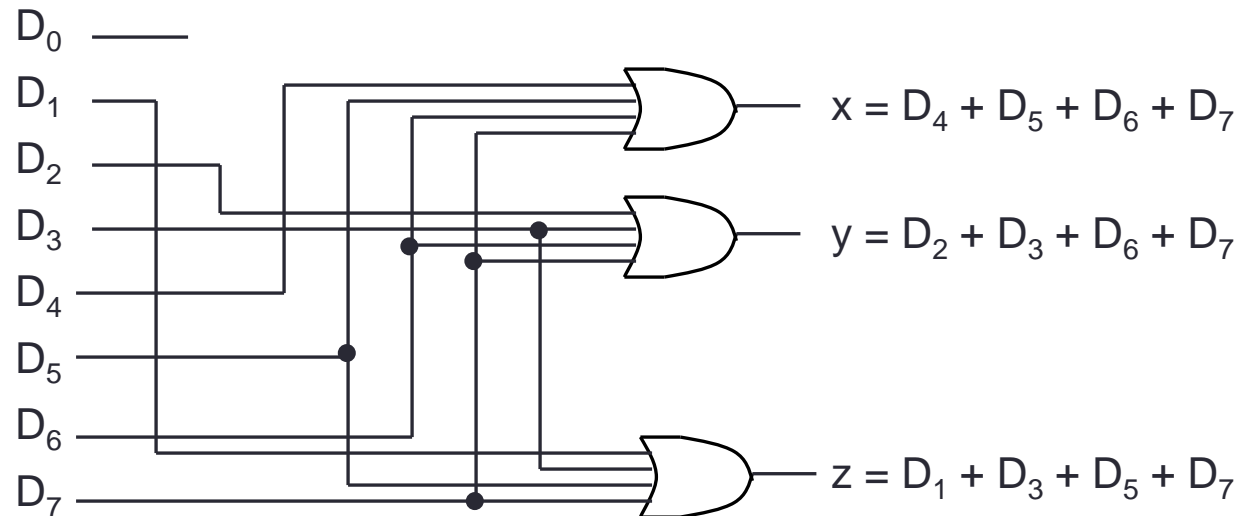
3. Encoders (3/4)

- Example: 8-to-3 encoder.
 - At any one time, only one input line of an encoder has a value of 1 (high), the rest are zeroes (low).
 - To allow for more than one input line to carry a 1, we need **priority 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

3. Encoders (4/4)

- Example: 8-to-3 encoder.



An 8-to-3 encoder

- Exercise:** Can you design a 2^n -to- n encoder without using K-map?

3. Priority Encoders (1/2)

- A **priority encoder** is one with priority
 - If two or more inputs are equal to 1, the input with the highest priority takes precedence.
 - If all inputs are 0, this input combination is considered invalid.
- Example of a **4-to-2 priority encoder**:

Inputs				Outputs		
D_0	D_1	D_2	D_3	f	g	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

3. Priority Encoders (2/2)

- Understanding “compact” function table

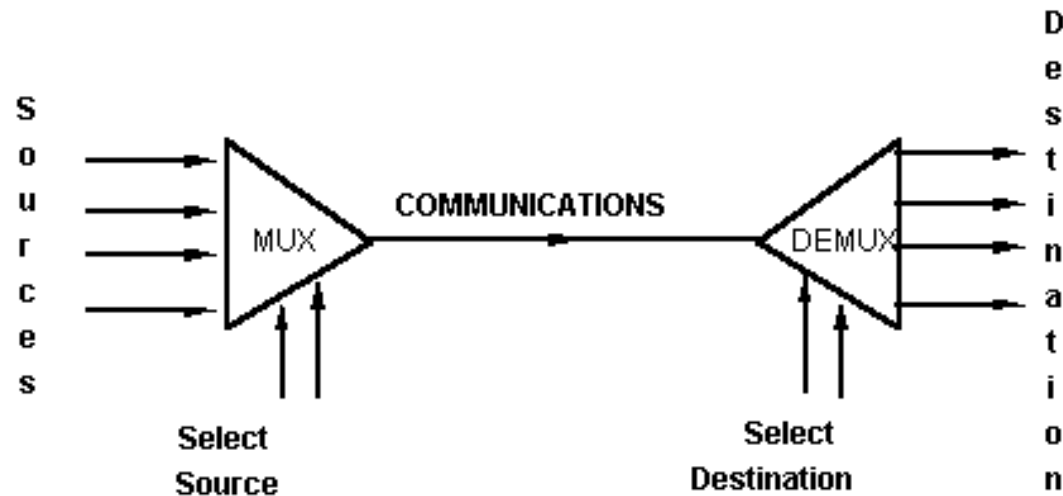
Inputs				Outputs		
D ₀	D ₁	D ₂	D ₃	f	g	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Inputs				Outputs		
D ₀	D ₁	D ₂	D ₃	f	g	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
0	1	0	0	0	1	1
1	1	0	0	0	1	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	0	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	1

- Exercise:** Obtain the simplified expressions for f , g and V .

Multiplexers and Demultiplexers

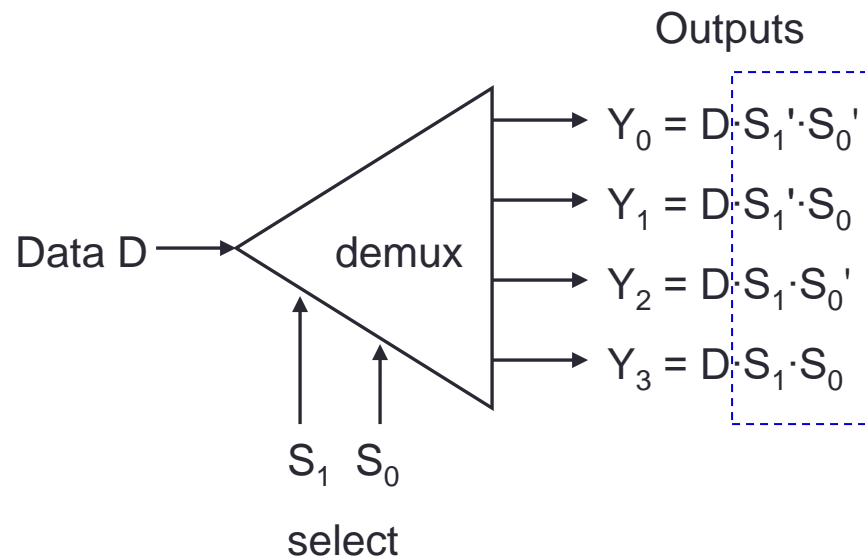
- An application:



- Helps share a *single communication line* among a number of devices.
- At any time, only one source and one destination can use the communication line.

4. Demultiplexers (1/2)

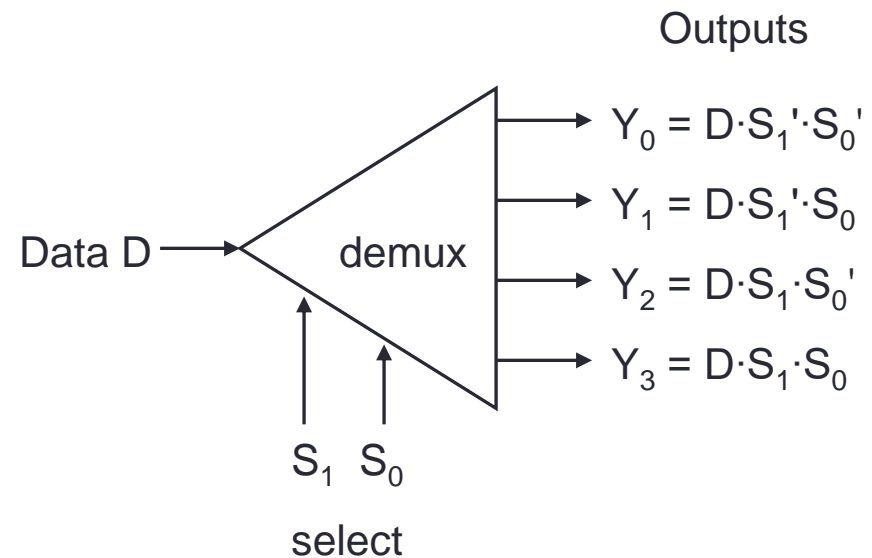
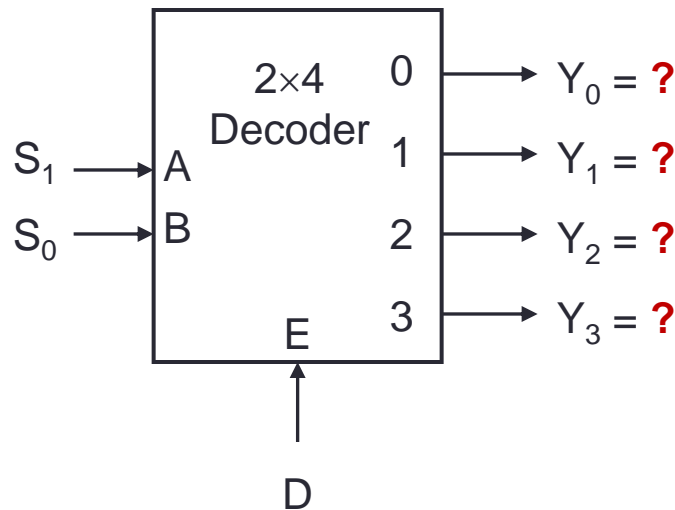
- Given an input line and a set of selection lines, a **demultiplexer** directs data from the input to one selected output line.
- Example: **1-to-4 demultiplexer**.



S ₁	S ₀	Y ₀	Y ₁	Y ₂	Y ₃
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

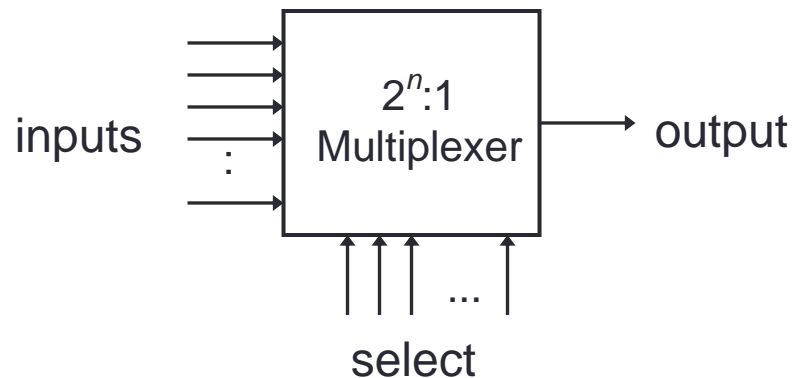
4. Demultiplexers (2/2)

- It turns out that the demultiplexer circuit is actually identical to a decoder with enable.



5. Multiplexers (1/4)

- A **multiplexer** is a device that has
 - A number of input lines
 - A number of selection lines
 - One output line
- It steers one of 2^n inputs to a single output line, using n selection lines. Also known as a ***data selector***.

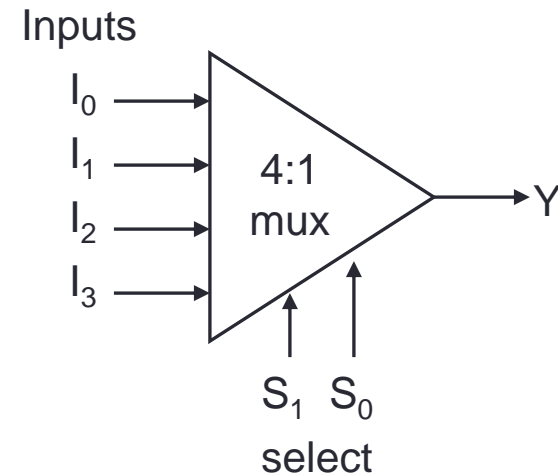
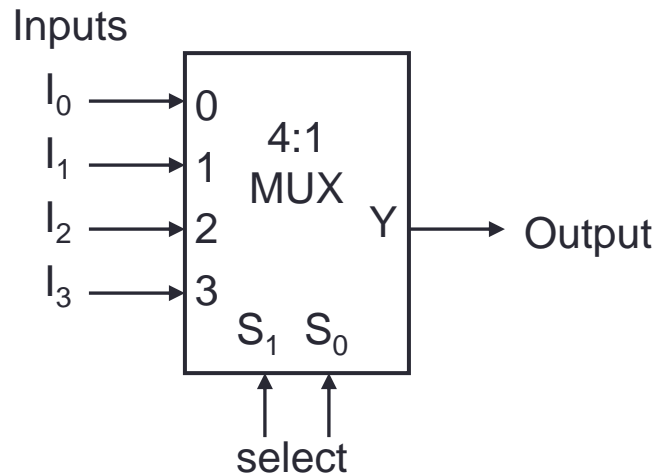


5. Multiplexers (2/4)

- 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



5. Multiplexers (3/4)

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

- Output of multiplexer is
“sum of the (product of *data lines* and *selection lines*)”
- Example: Output of a 4-to-1 multiplexer is:

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

Note:

Expressing

$$I_0 \cdot (S_1' \cdot S_0') + I_1 \cdot (S_1' \cdot S_0) + I_2 \cdot (S_1 \cdot S_0') + I_3 \cdot (S_1 \cdot S_0)$$

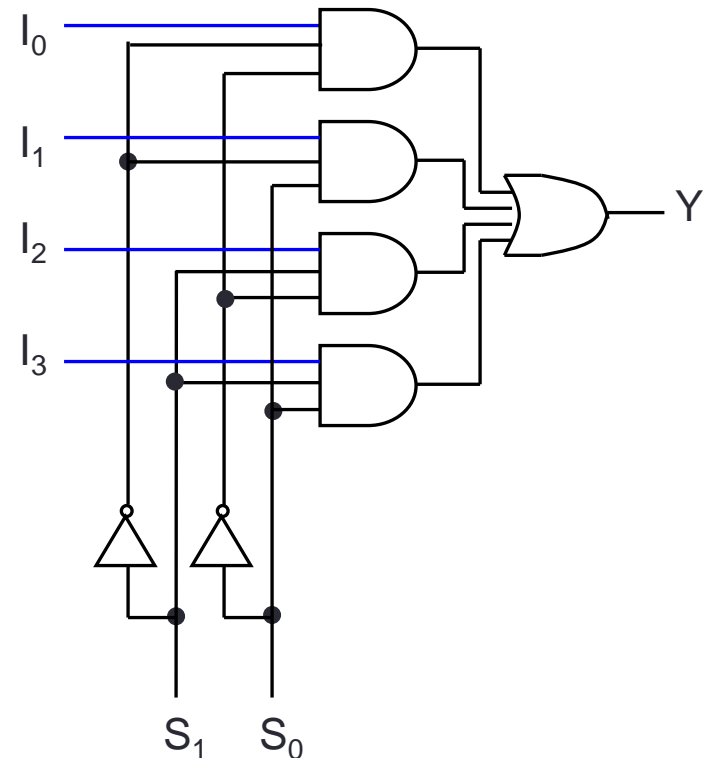
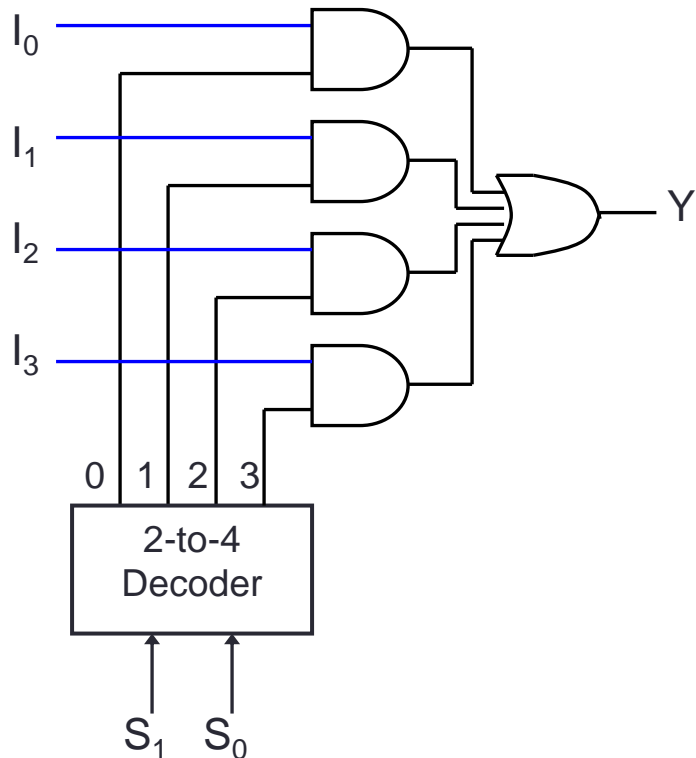
in minterms notation, it is equal to

$$I_0 \cdot m_0 + I_1 \cdot m_1 + I_2 \cdot m_2 + I_3 \cdot m_3$$

This is useful later (eg: slide 45).

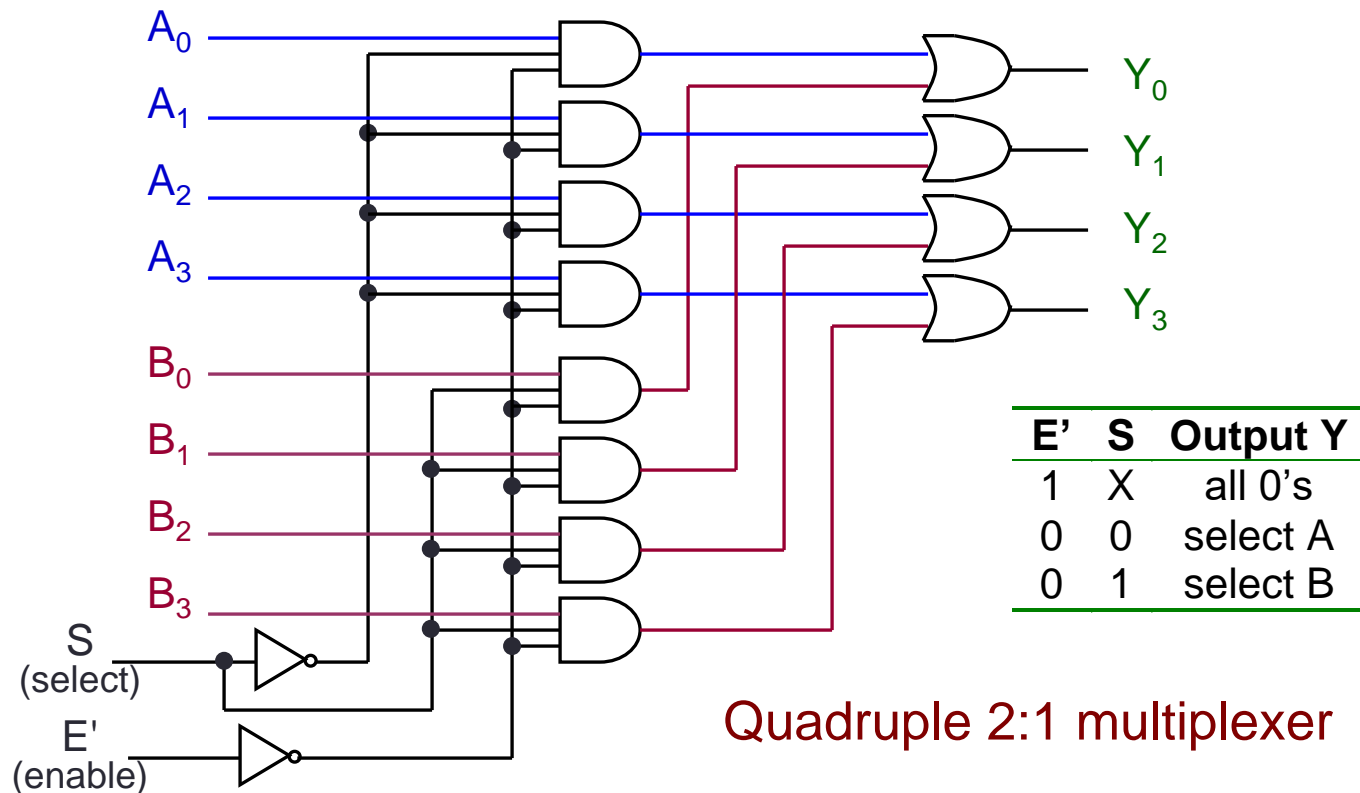
5. Multiplexers (4/4)

- A 2^n -to-1-line multiplexer, or simply $2^n:1$ MUX, is made from an $n:2^n$ decoder by adding to it 2^n input lines, one to each AND gate.
- A **4:1 multiplexer circuit:**



5. Multiplexer IC Package

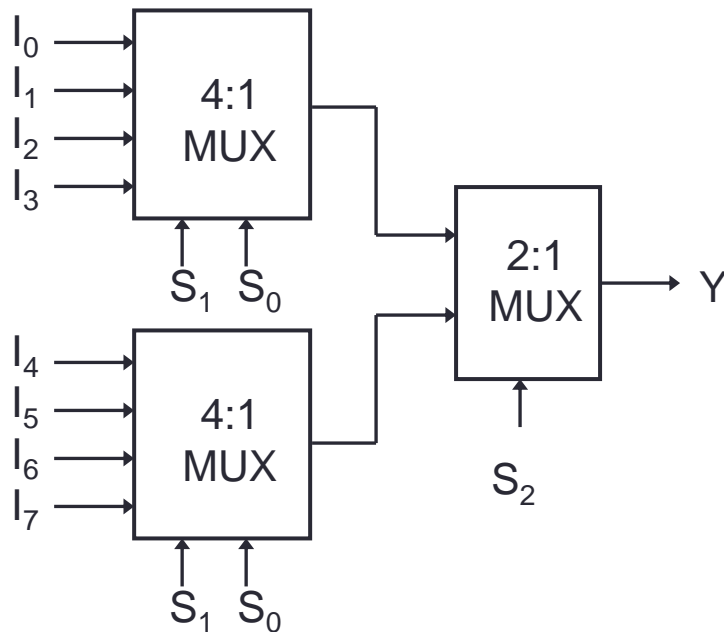
- Some IC packages have a few multiplexers in each package (chip). The selection and enable inputs are common to all multiplexers within the package.



Quadrate 2:1 multiplexer

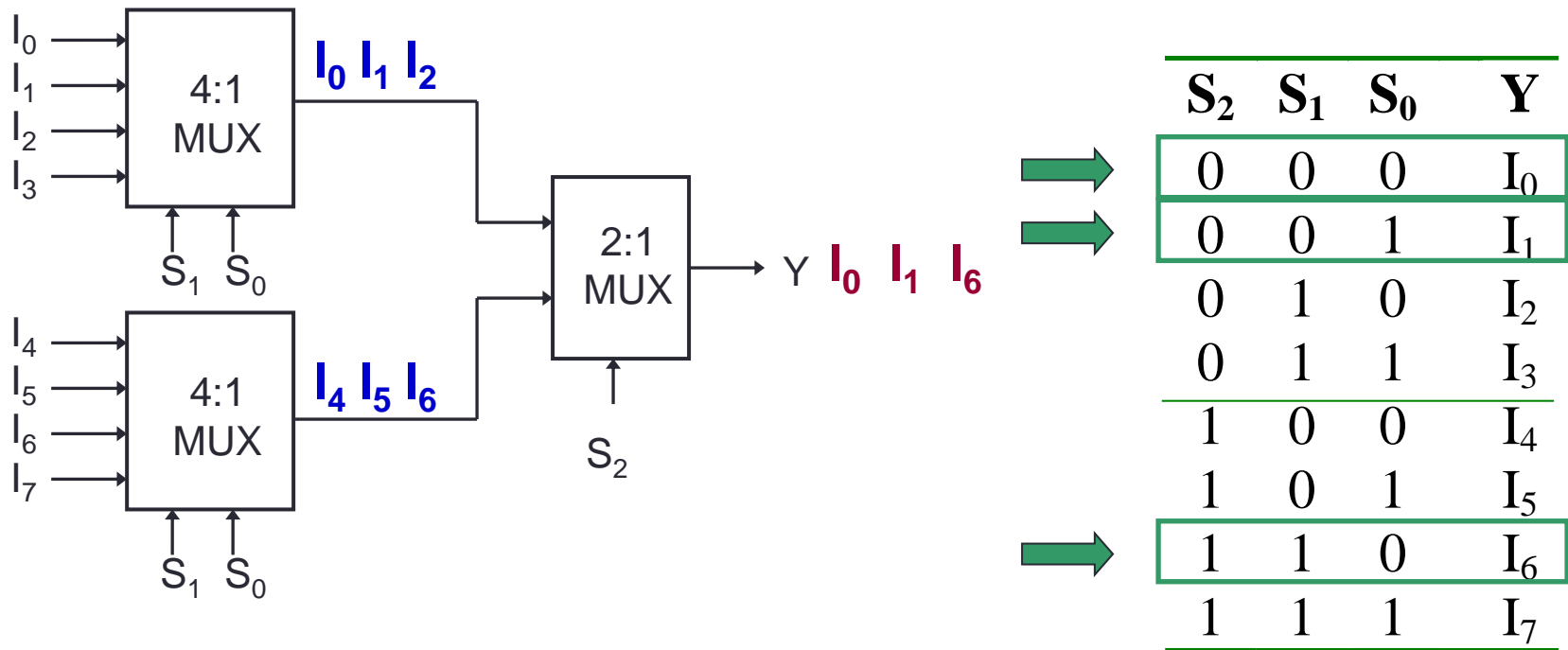
5. Constructing Larger Multiplexers (1/4)

- Larger multiplexers can be constructed from smaller ones.
- An 8-to-1 multiplexer can be constructed from smaller multiplexers like this (note placement of selector lines):



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

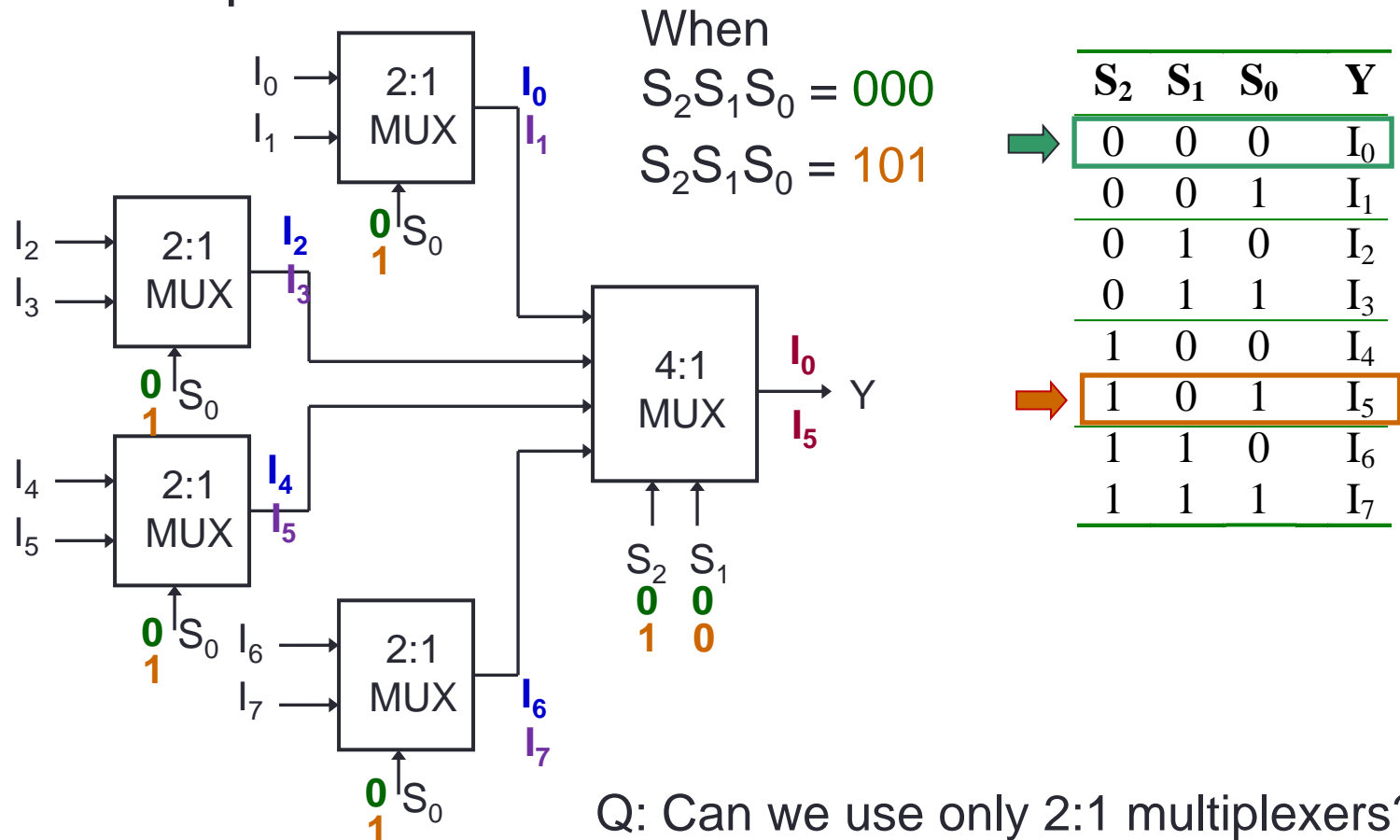
5. Constructing Larger Multiplexers (2/4)



- When $S_2 S_1 S_0 = 000$
- When $S_2 S_1 S_0 = 001$
- When $S_2 S_1 S_0 = 110$

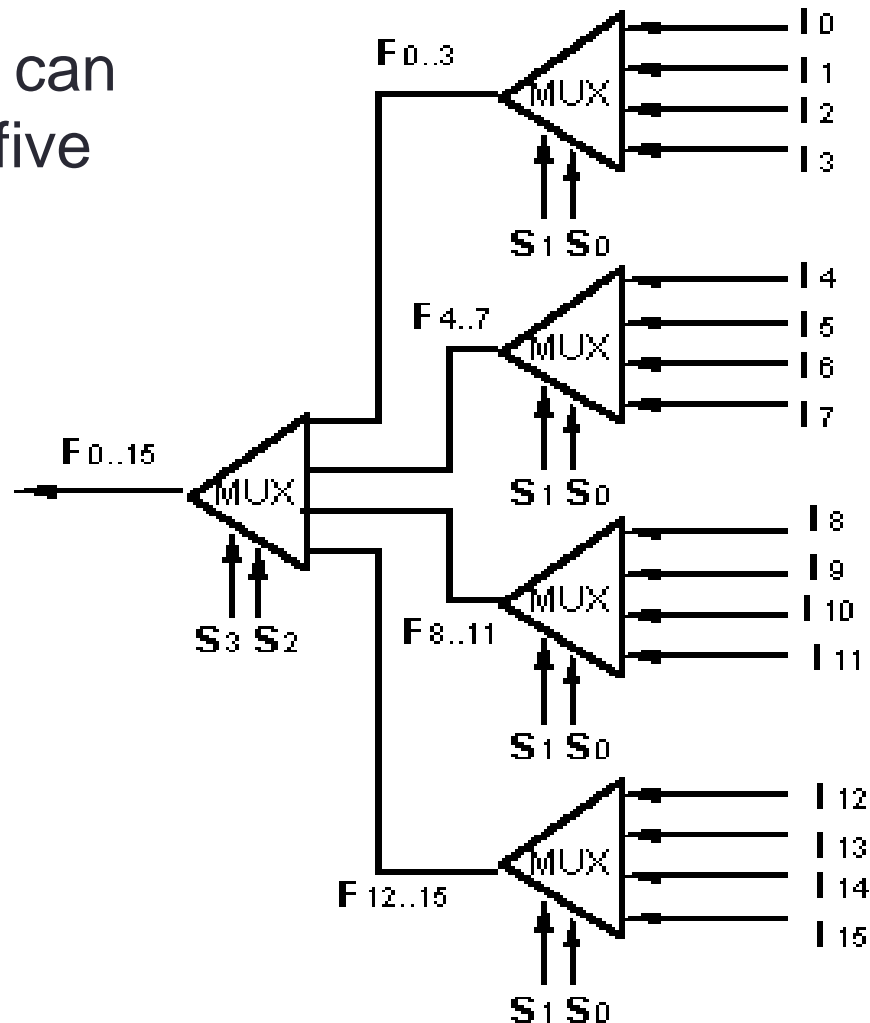
5. Constructing Larger Multiplexers (3/4)

- Another implementation of an **8-to-1 multiplexer** using smaller multiplexers:

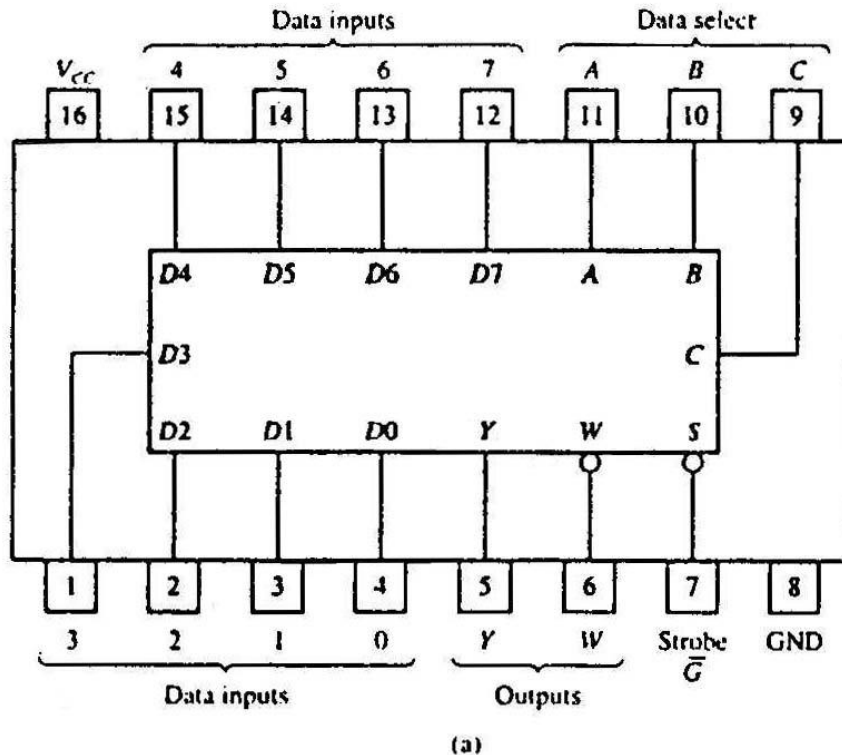


5. Constructing Larger Multiplexers (4/4)

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



5. Standard MSI Multiplexer (1/2)

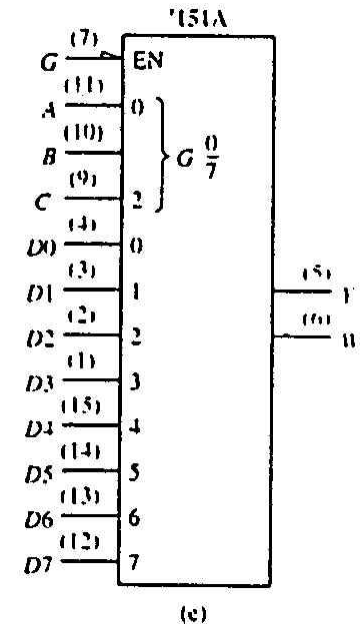
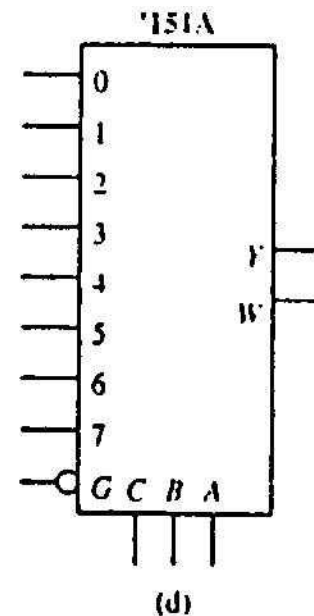
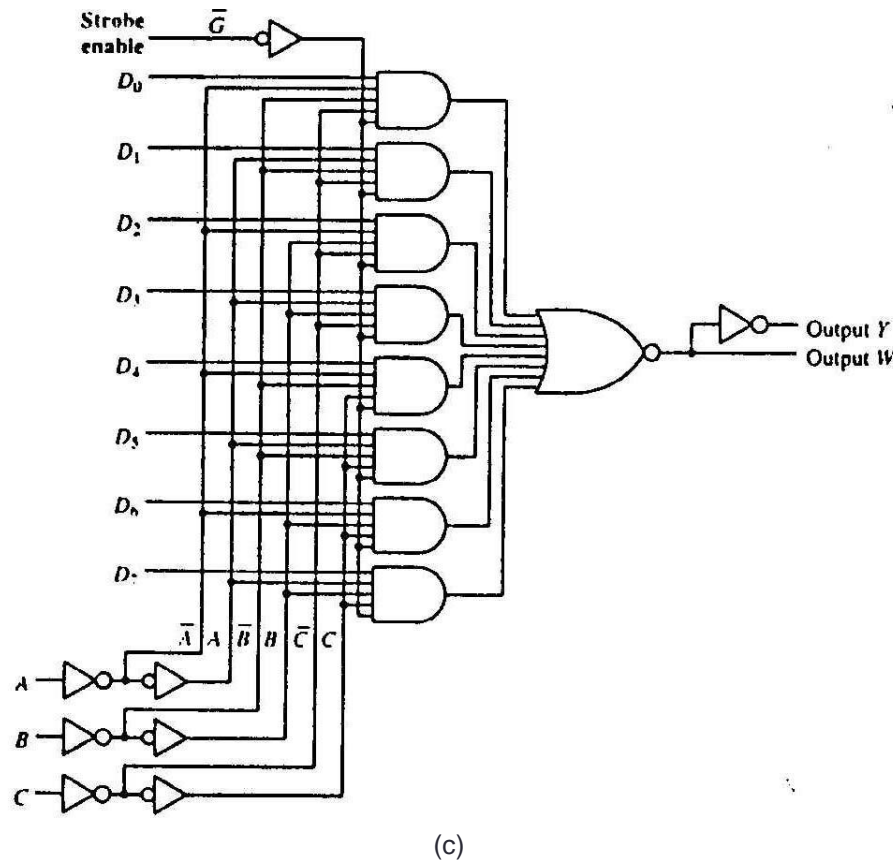


INPUTS				OUTPUTS	
SELECT			STROBE \overline{G}	Y	W
C	B	A			
X	X	X	H	L	H
L	L	L	L	D0	$\overline{D0}$
L	L	H	L	D1	$\overline{D1}$
L	H	L	L	D2	$\overline{D2}$
L	H	H	L	D3	$\overline{D3}$
H	L	L	L	D4	$\overline{D4}$
H	L	H	L	D5	$\overline{D5}$
H	H	L	L	D6	$\overline{D6}$
H	H	H	L	D7	$\overline{D7}$

(b)

74151A 8-to-1 multiplexer. (a) Package configuration. (b) Function table.

5. Standard MSI Multiplexer (2/2)



74151A 8-to-1 multiplexer. (c) Logic diagram. (d) Generic logic symbol.
(e) IEEE standard logic symbol.

Source: *The TTL Data Book Volume 2. Texas Instruments Inc., 1985.*

5. Multiplexers: Implementing Functions (1/3)

- Boolean functions can be implemented using multiplexers.
- A 2^n -to-1 multiplexer can implement a Boolean function of n input variables, as follows:

1. Express in sum-of-minterms form.

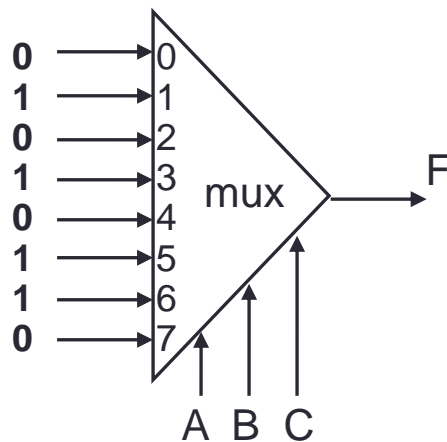
Example:

$$\begin{aligned} F(A,B,C) &= A' \cdot B' \cdot C + A' \cdot B \cdot C + A \cdot B' \cdot C + A \cdot B \cdot C' \\ &= \Sigma m(1,3,5,6) \end{aligned}$$

2. Connect n variables to the n selection lines.
3. Put a '1' on a data line if it is a minterm of the function, or '0' otherwise.

5. Multiplexers: Implementing Functions (2/3)

■ $F(A,B,C) = \Sigma m(1,3,5,6)$



This method works because:

$$\text{Output} = I_0 \cdot m_0 + I_1 \cdot m_1 + I_2 \cdot m_2 + I_3 \cdot m_3 \\ + I_4 \cdot m_4 + I_5 \cdot m_5 + I_6 \cdot m_6 + I_7 \cdot m_7$$

Supplying '1' to I_1, I_3, I_5, I_6 , and '0' to the rest:

$$\text{Output} = m_1 + m_3 + m_5 + m_6$$

From slide 34 (4:1 mux)

Expressing

$$I_0 \cdot (S_1' \cdot S_0') + I_1 \cdot (S_1' \cdot S_0) + I_2 \cdot (S_1 \cdot S_0') + I_3 \cdot (S_1 \cdot S_0)$$

in minterms notation, it is equal to

$$I_0 \cdot m_0 + I_1 \cdot m_1 + I_2 \cdot m_2 + I_3 \cdot m_3$$

5. Multiplexers: Implementing Functions (3/3)

- Example: Use a 74151A to implement $f(x_1, x_2, x_3) = \sum m(0, 2, 3, 5)$

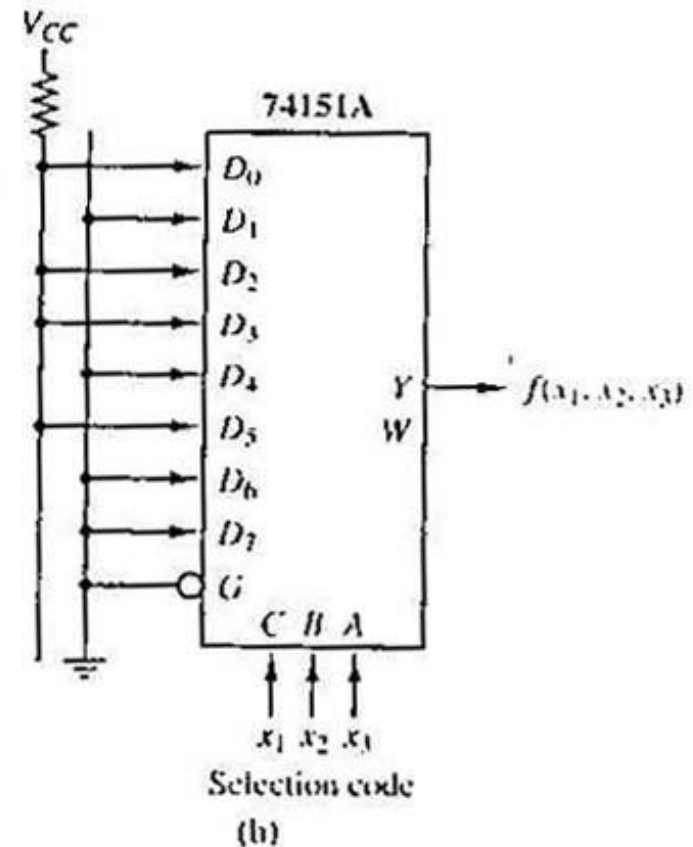
i	C	B	A	Y
	x_1	x_2	x_3	f
0	0	0	0	1 $D_0 = 1$
1	0	0	1	0 $D_1 = 0$
2	0	1	0	1 $D_2 = 1$
3	0	1	1	1 $D_3 = 1$
4	1	0	0	0 $D_4 = 0$
5	1	0	1	1 $D_5 = 1$
6	1	1	0	0 $D_6 = 0$
7	1	1	1	0 $D_7 = 0$

(a)

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

(a) Truth table.

(b) Implementation with 74151A.



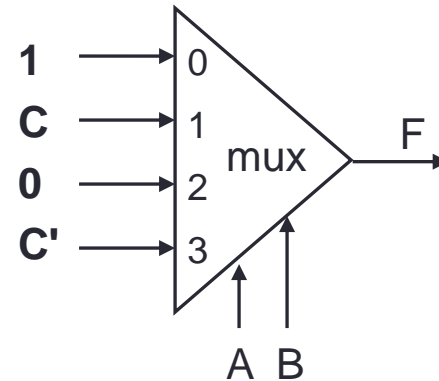
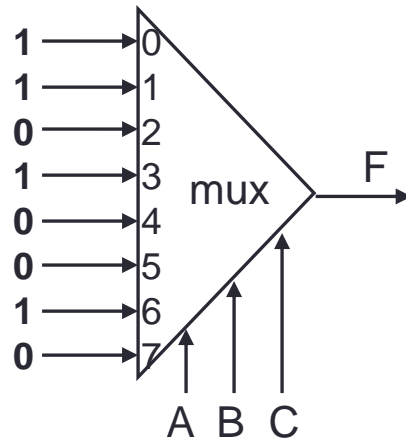
5. Using Smaller Multiplexers (1/6)

- Earlier, we saw how a 2^n -to-1 multiplexer can be used to implement a Boolean function of n (input) variables.
- However, we can use a single smaller $2^{(n-1)}$ -to-1 multiplexer to implement a Boolean function of n (input) variables.
- Example: The function
$$F(A,B,C) = \Sigma m(1,3,5,6)$$
can be implemented using a 4-to-1 multiplexer (rather than an 8-to-1 multiplexer).

5. Using Smaller Multiplexers (2/6)

- Let's look at this example:

$$F(A,B,C) = \sum m(0,1,3,6) = A' \cdot B' \cdot C' + A' \cdot B' \cdot C + A' \cdot B \cdot C + A \cdot B \cdot C'$$



- Note: Two of the variables, **A** and **B**, are applied as selection lines of the multiplexer, while the inputs of the multiplexer contain **1**, **C**, **0** and **C'**.

5. Using Smaller Multiplexers (3/6)

■ Procedure

1. Express Boolean function in sum-of-minterms form.

Example: $F(A,B,C) = \sum m(0,1,3,6)$

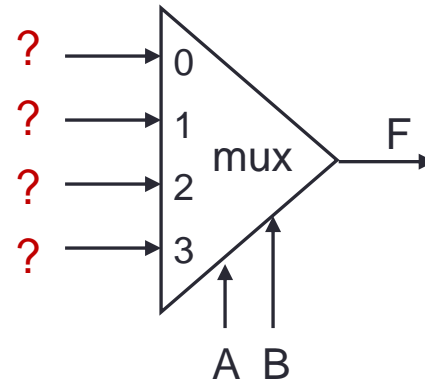
2. Reserve one variable (in our example, we take the least significant one) for input lines of multiplexer, and use the rest for selection lines.

Example: C is for input lines; A and B for selection lines.

5. Using Smaller Multiplexers (4/6)

3. Draw the truth table for function, by grouping inputs by selection line values, then determine multiplexer inputs by comparing input line (**C**) and function (**F**) for corresponding selection line values.

A	B	C	F	MUX input
0	0	0	1	
0	0	1	1	
0	1	0	0	
0	1	1	1	
1	0	0	0	
1	0	1	0	
1	1	0	1	
1	1	1	0	



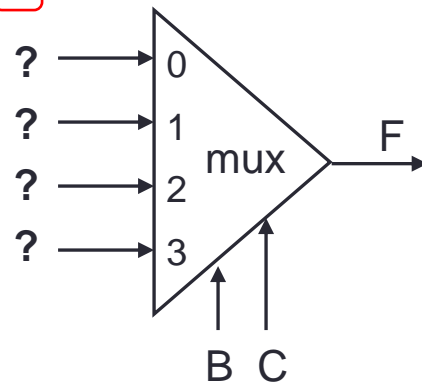
5. Using Smaller Multiplexers (5/6)

- Alternative: What if we use **A** for input lines, and **B**, **C** for selector lines?

A	B	C	F	Mux Input
0	0	0	1	1
0	0	1	1	
0	1	0	0	C
0	1	1	1	
1	0	0	0	0
1	0	1	0	
1	1	0	1	C'
1	1	1	0	

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

A' (when $BC = 00$)



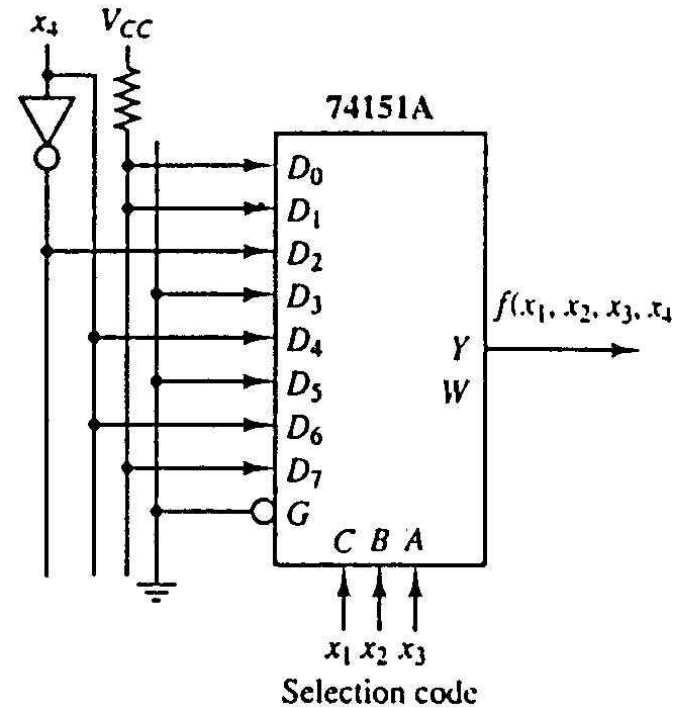
5. Using Smaller Multiplexers (6/6)

- Example: Implement the function below with 74151A:

$$f(x_1, x_2, x_3, x_4) = \Sigma m(0, 1, 2, 3, 4, 9, 13, 14, 15)$$

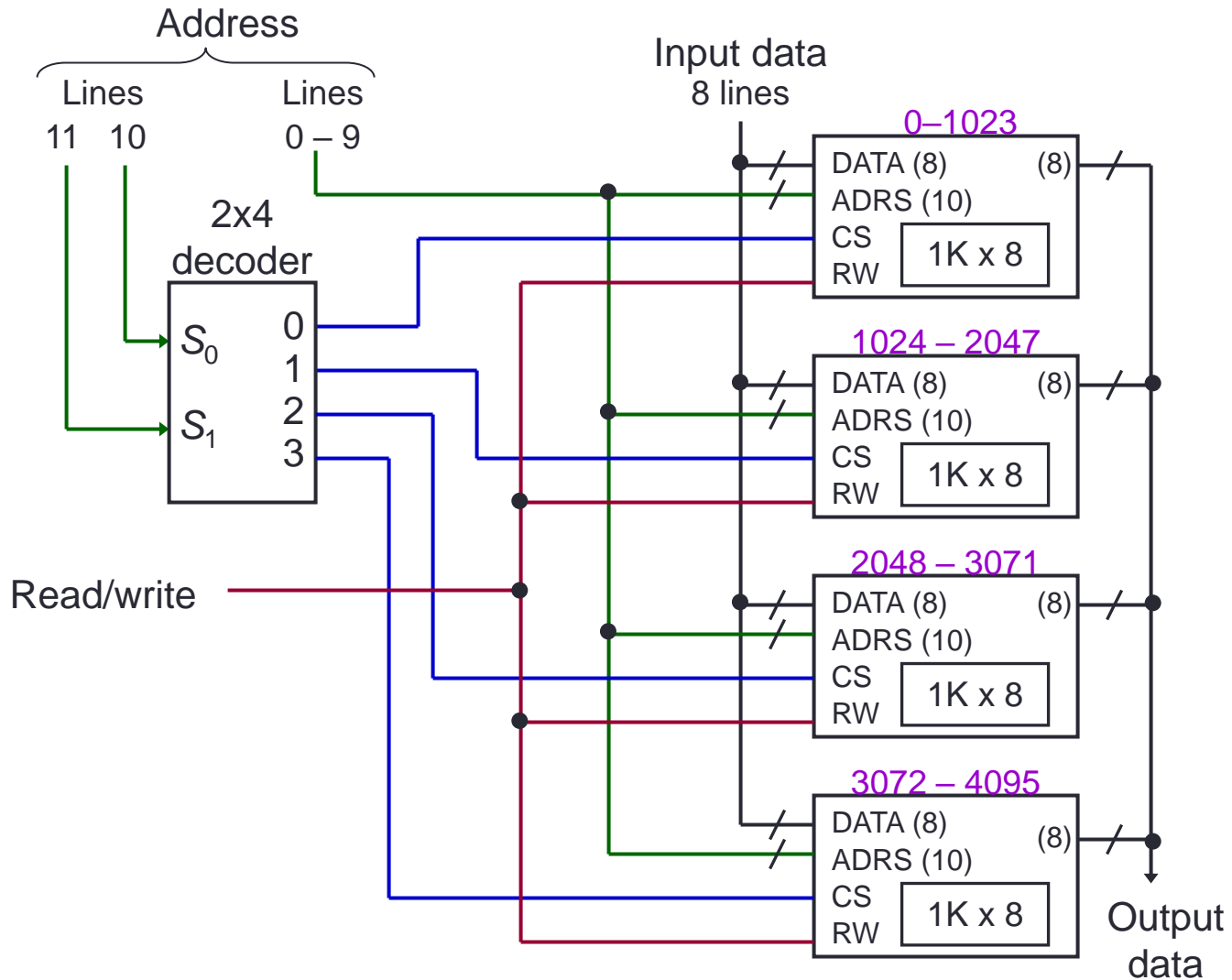
	C	B	A				Y
i	X ₁	X ₂	X ₃	X ₄	f	f	
0	0	0	0	0	1		
	0	0	0	1	1	1	D ₀ = 1
1	0	0	1	0	1		
	0	0	1	1	1	1	D ₁ = 1
2	0	1	0	0	1		
	0	1	0	1	0	\bar{X}_4	D ₂ = \bar{X}_4
3	0	1	1	0	0		
	0	1	1	1	0	0	D ₃ = 0
4	1	0	0	0	0		
	1	0	0	1	1	X ₄	D ₄ = X ₄
5	1	0	1	0	0		
	1	0	1	1	0	0	D ₅ = 0
6	1	1	0	0	0		
	1	1	0	1	1	X ₄	D ₆ = X ₄
7	1	1	1	0	1		
	1	1	1	1	1	1	D ₇ = 1

(a)



(b)

Peeking Ahead



End of File