ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

COMPUTER ORGANISATION
(TỔ CHỨC MÁY TÍNH)

MSI Components

Acknowledgement

- The contents of these slides have origin from School of Computing, National University of Singapore.
- We greatly appreciate support from Mr. Aaron Tan Tuck Choy for kindly sharing these materials.

Policies for students

- These contents are only used for students PERSONALLY.
- Students are NOT allowed to modify or deliver these contents to anywhere or anyone for any purpose.

WHERE ARE WE NOW?

- Number systems and codes
- Boolean algebra
- Logic gates and circuits
- Simplification
- Combinational circuits
- Sequential circuits
- Performance
- Assembly language
- The processor: Datapath and control
- Pipelining
- Memory hierarchy: Cache
- Input/output

Preparation: 2 weeks

Logic Design: 3 weeks

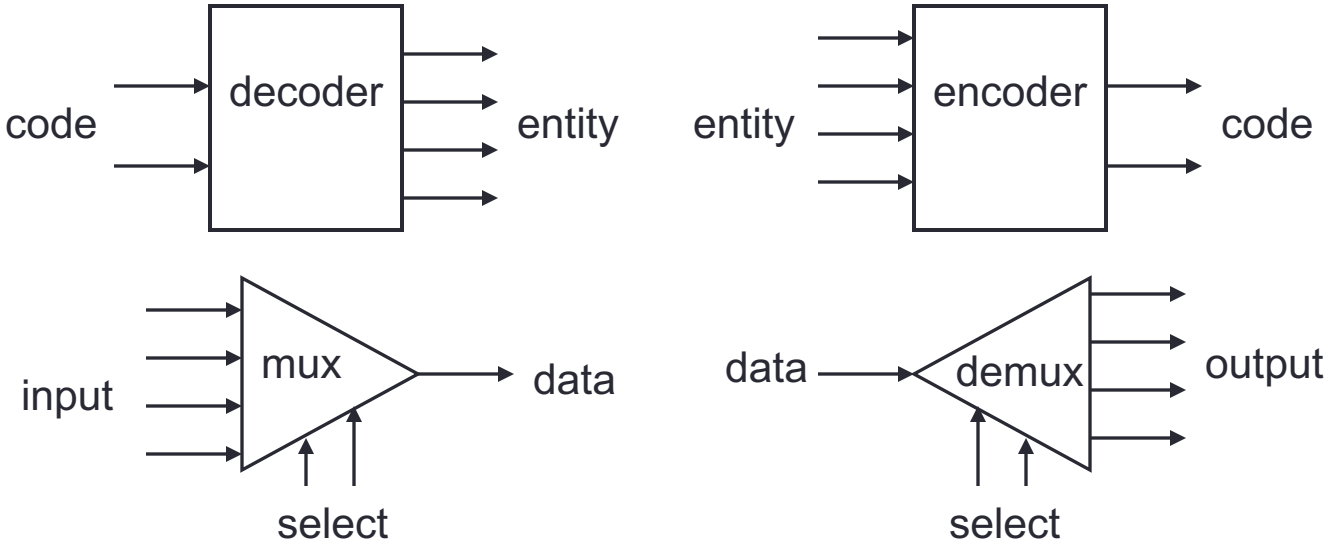
Computer organisation

MSI COMPONENTS

- Introduction
- Decoders
- Encoders
- Demultiplexers
- Multiplexers

INTRODUCTION

- Four common and useful MSI circuits:
  - Decoder
  - Demultiplexer
  - Encoder
  - Multiplexer
- Block-level outlines of MSI circuits:

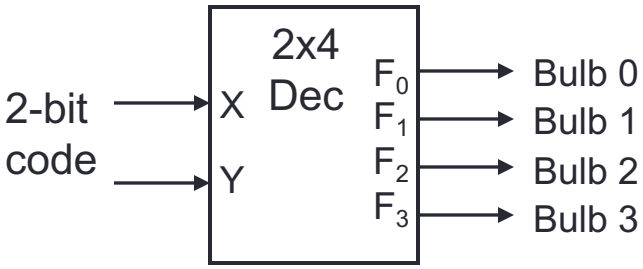


DECODERS (1/5)

- 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 (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.

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:

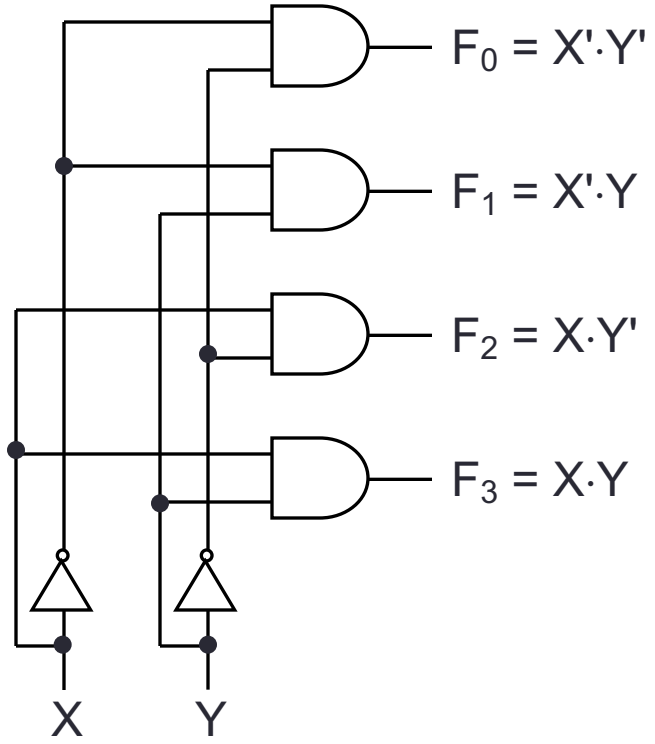
X	Y	F_0	F_1	F_2	F_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

DECODERS (3/5)

- From truth table, circuit for 2×4 decoder is:

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

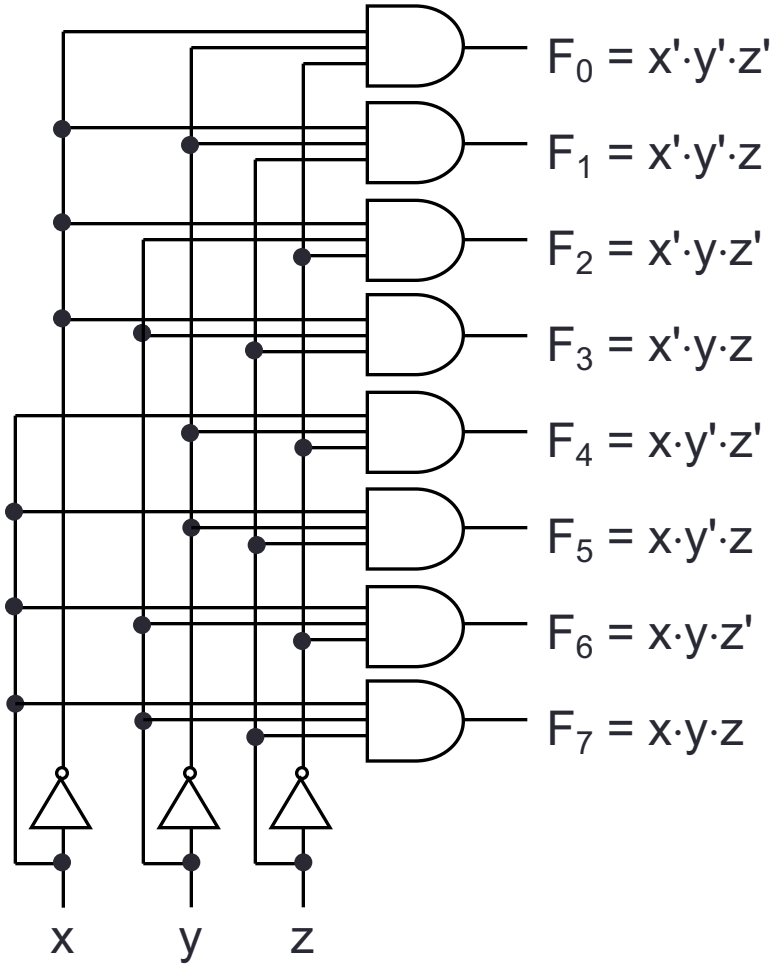
- Note: Each output is a 2-variable minterm (X'·Y', X'·Y, X·Y' or X·Y)



DECODERS (4/5)

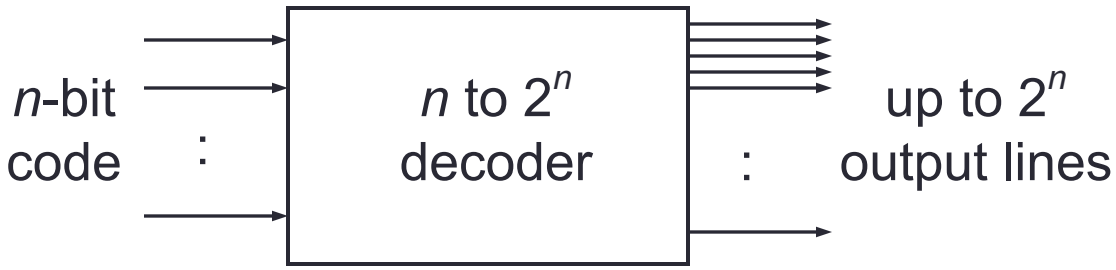
- Design a 3×8 decoder.

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



DECODERS (5/5)

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



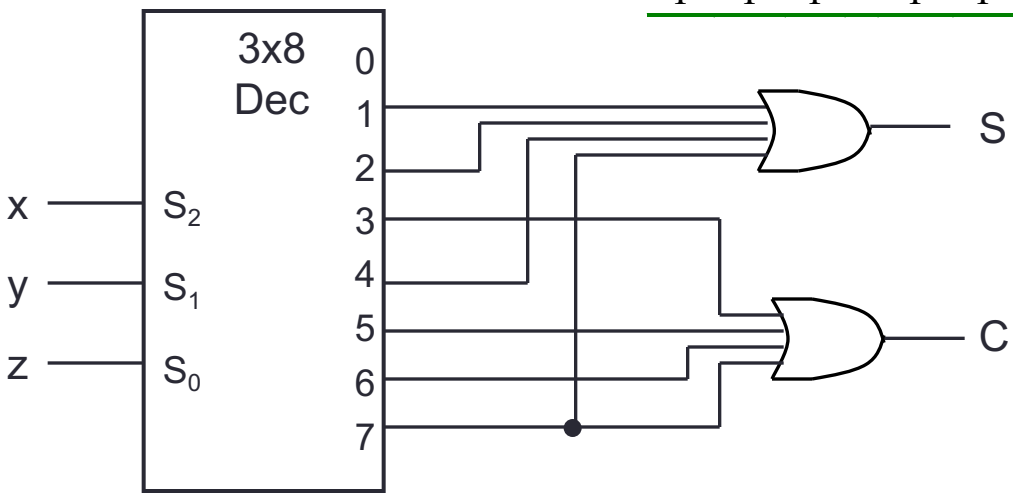
DECODERS: IMPLEMENTING FUNCTIONS (1/5)

- A Boolean function, in sum-of-minterms form ⇨ 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 few minterms.

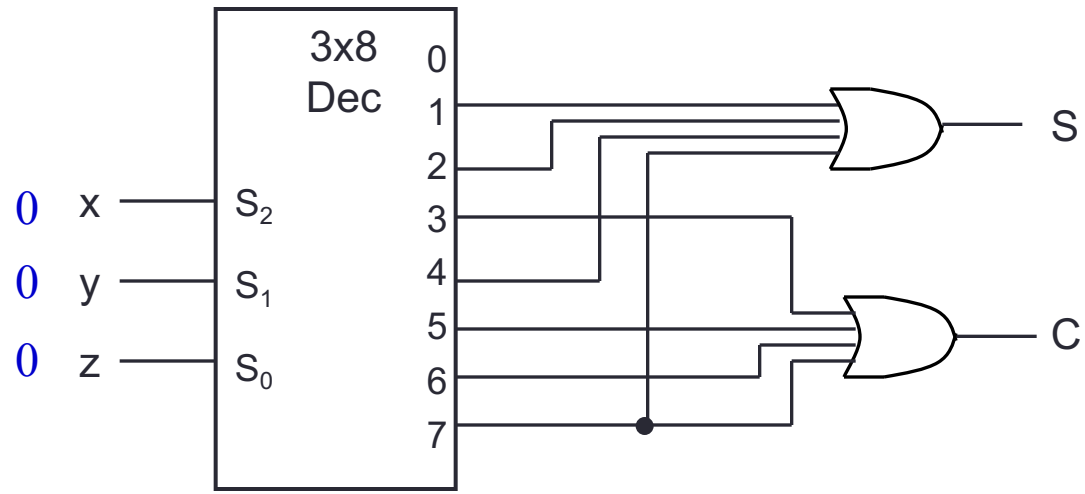
DECODERS: IMPLEMENTING FUNCTIONS (2/5)

- 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



DECODERS: IMPLEMENTING FUNCTIONS (3/5)



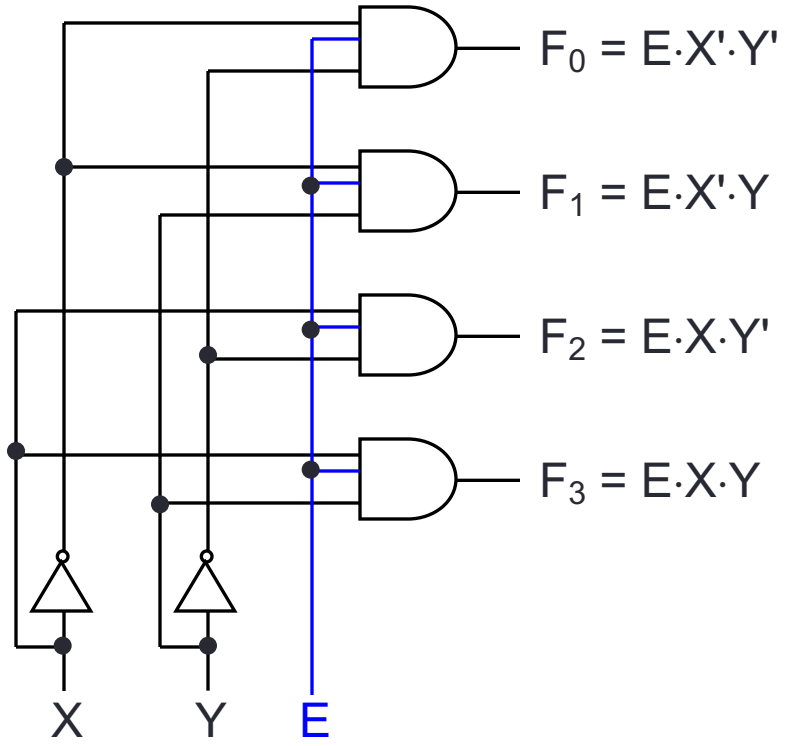
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

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	X	X	0	0	0	0

- Circuit of a 2×4 decoder with enable:



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	X	X	0	0	0	0

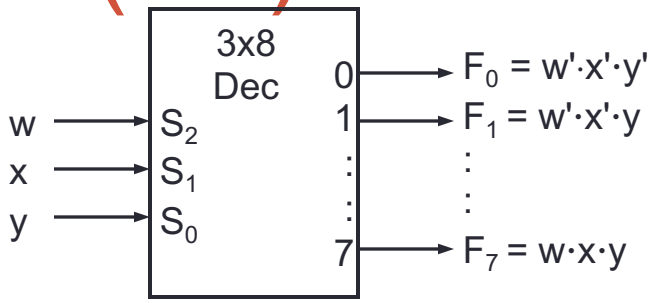
Decoder with 1-enable

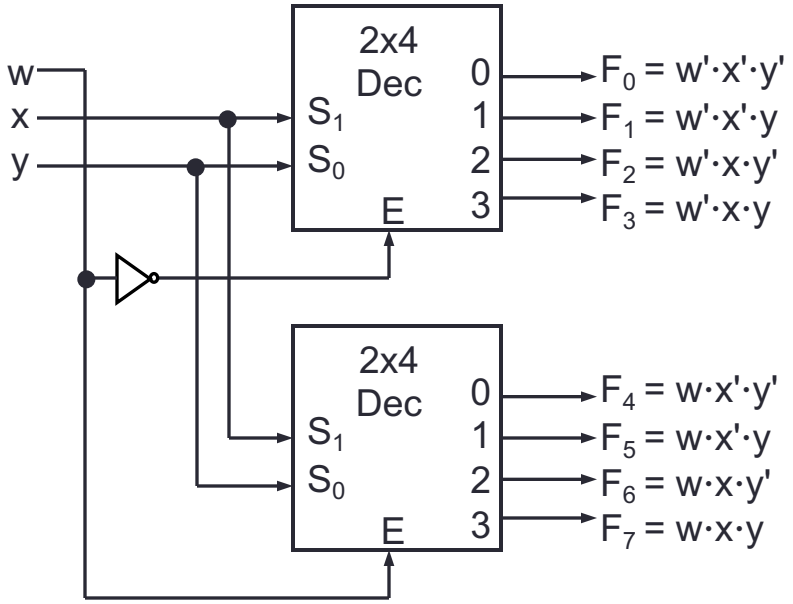
E'	X	Y	F_0	F_1	F_2	F_3
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1
1	X	X	0	0	0	0

Decoder with 0-enable

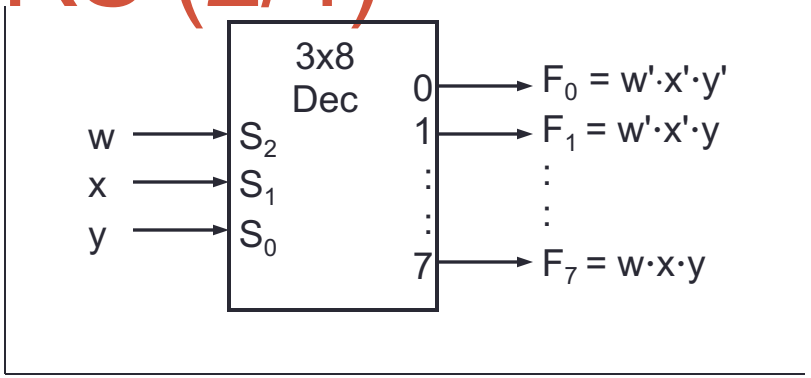
LARGER DECODERS (1/4)

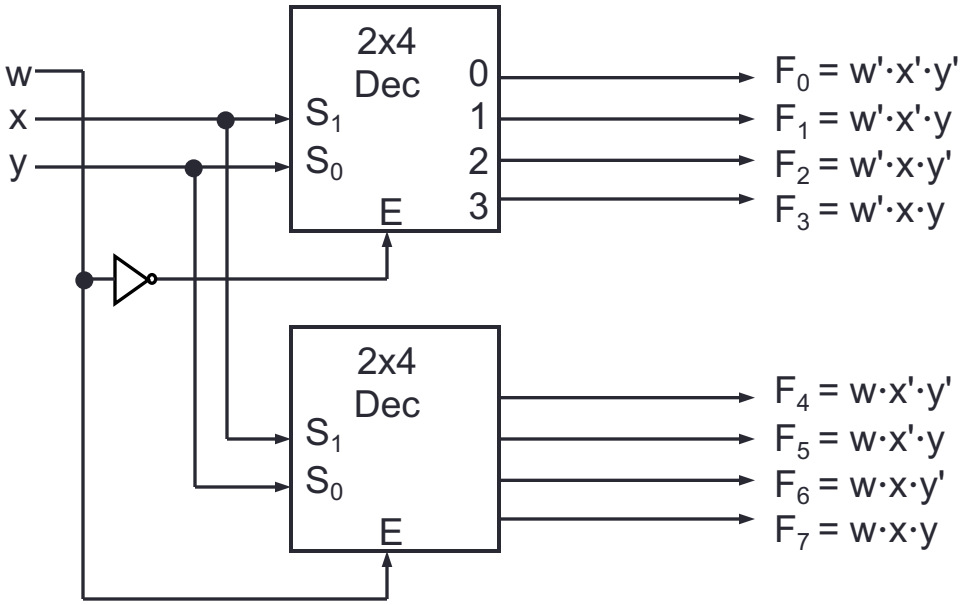
- Larger decoders can be constructed from smaller ones.
- Example: A 3×8 decoder can be built from two 2×4 decoders (with one-enable) and an inverter.





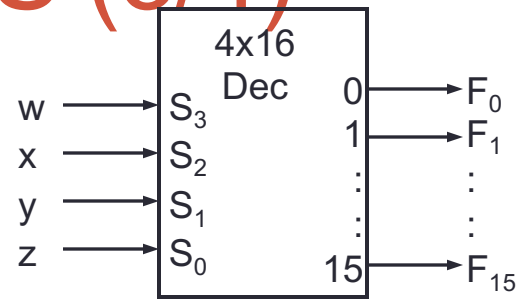
LARGER DECODERS (2/4)

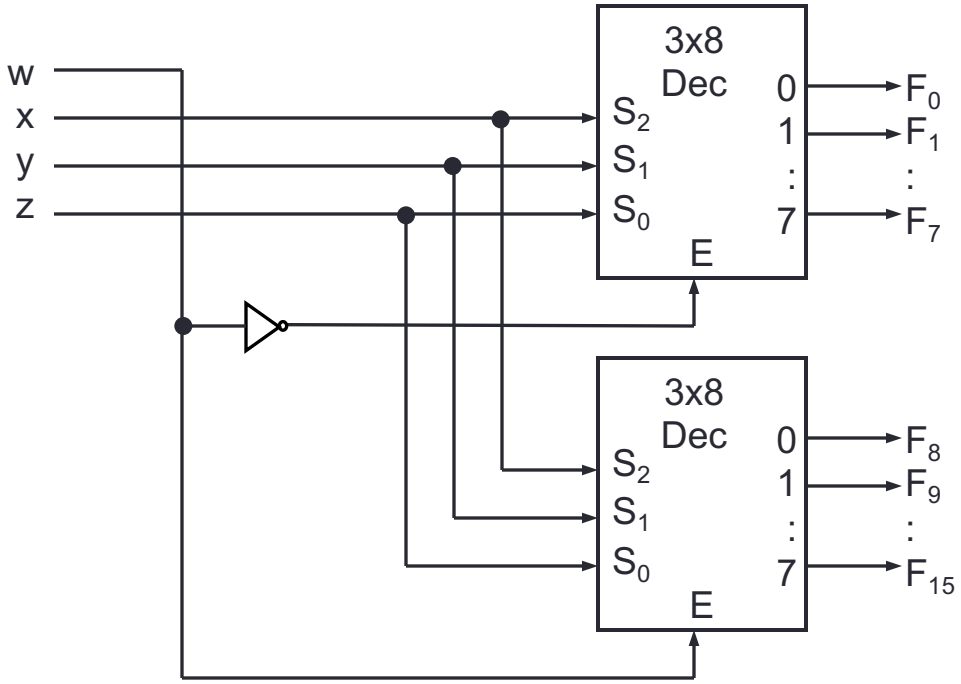




LARGER DECODERS (3/4)

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



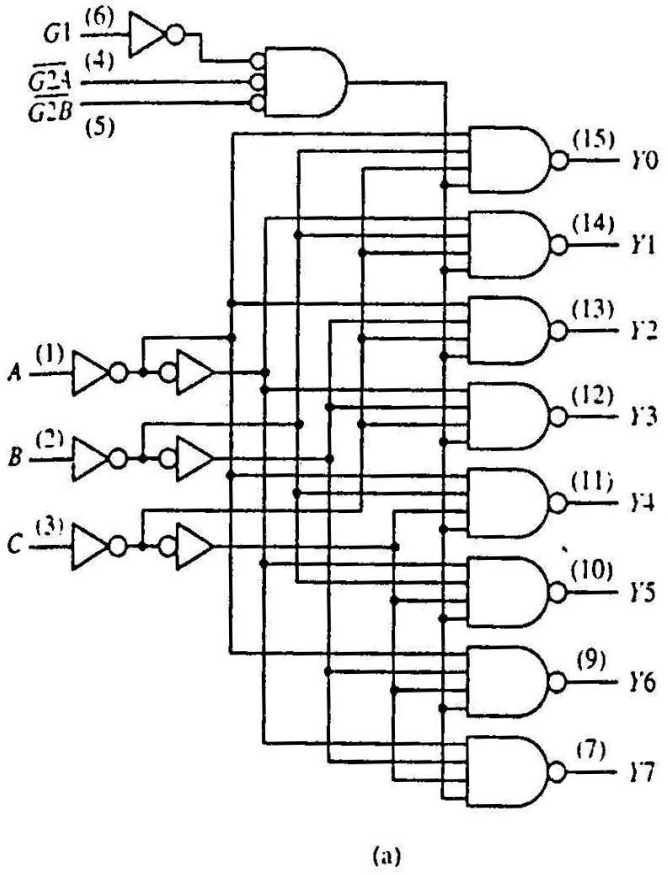


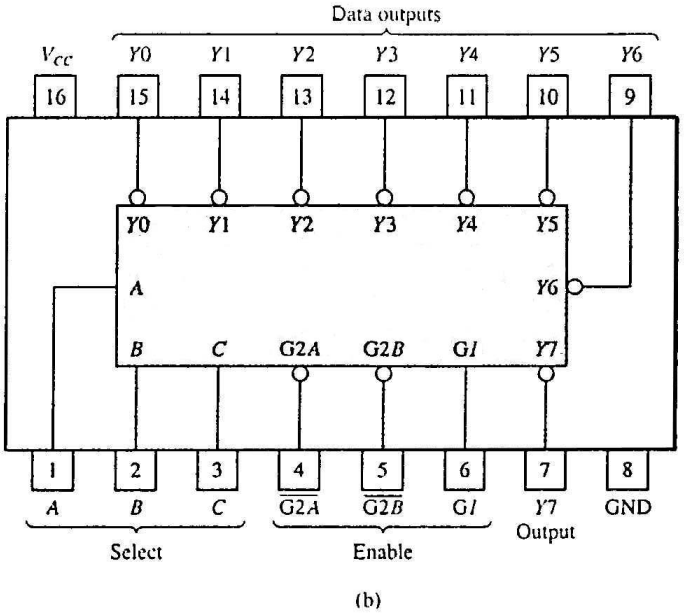
LARGER DECODERS (4/4)

- Note: The input, w and its complement, w', are used to select either one of the two smaller decoders.
- Decoders may also have zero-enable and/or negated outputs.
  - Normal outputs = active high
  - Negated outputs = active low
- 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 slides?
- Exercise: How to construct a 4×16 decoder using five 2×4 decoders with enable?

STANDARD MSI DECODER (1/2)

- 74138 (3-to-8 decoder)





74138 decoder module.
(a) Logic circuit.
(b) Package pin configuration.

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	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

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

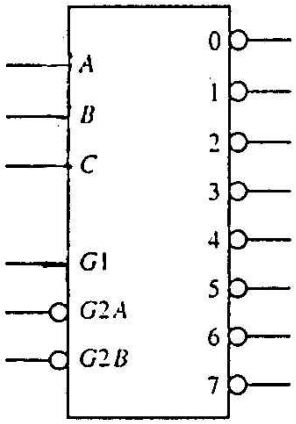
H = high level, L = low level, X = irrelevant

(c)

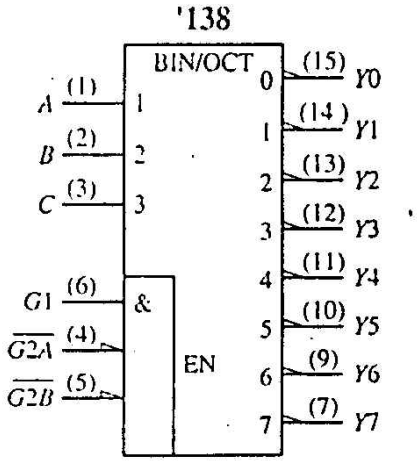
74138 decoder module.
(c) Function table.

74138 decoder module.
(d) Generic symbol.
(e) IEEE standard logic symbol.

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



(d)



(e)

DECODERS: IMPLEMENTING FUNCTIONS REVISIT (1/2)

- Example: Implement the following function using a 3×8 decoder and 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:

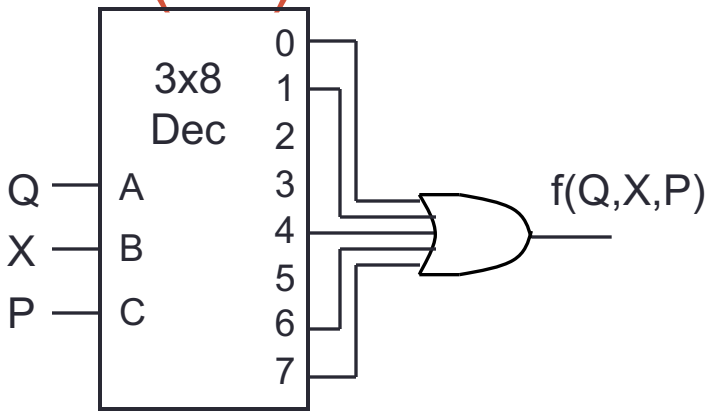
    $f(Q,X,P) = (m_0' \cdot m_1' \cdot m_4' \cdot m_6' \cdot m_7')'$
  - Using a decoder with active-high outputs with a NOR gate:

    $f(Q,X,P) = (m_2 + m_3 + m_5)' [= M_2 \cdot M_3 \cdot M_5]$
  - Using a decoder with active-low outputs with an AND gate:

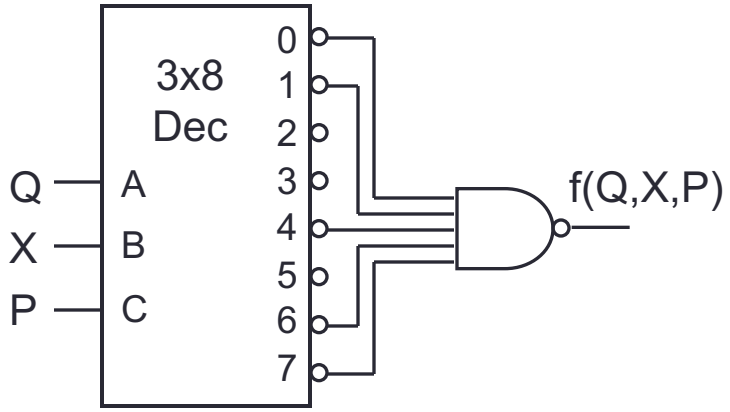
    $f(Q,X,P) = m_2' \cdot m_3' \cdot m_5'$

DECODERS: IMPLEMENTING FUNCTIONS REVISIT (2/2)

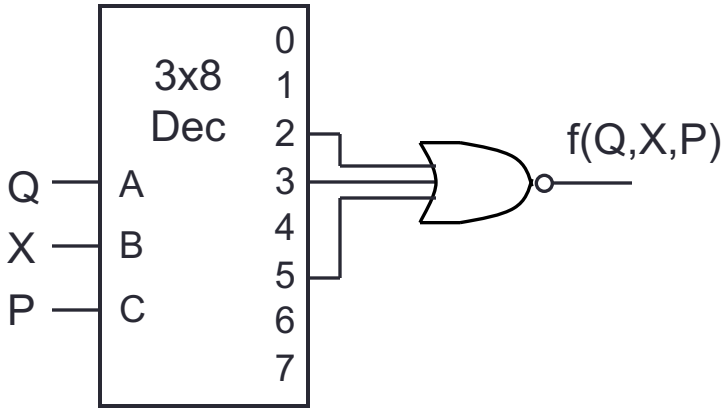
$$f(Q,X,P) = \Sigma m(0,1,4,6,7) = \Pi M(2,3,5)$$



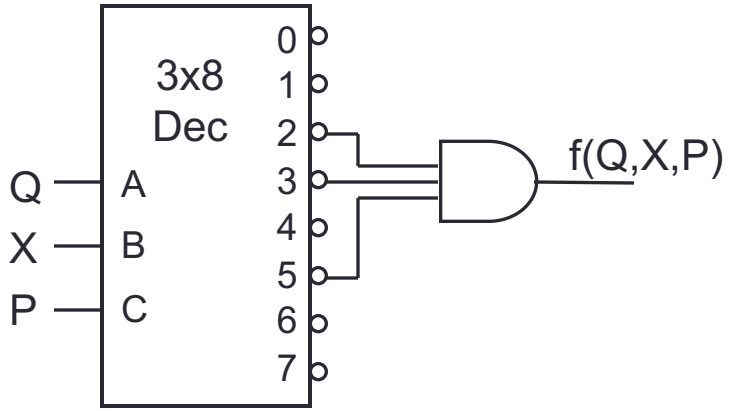
(a) Active-high decoder with OR gate.



(b) Active-low decoder with NAND gate.



(c) Active-high decoder with NOR gate.



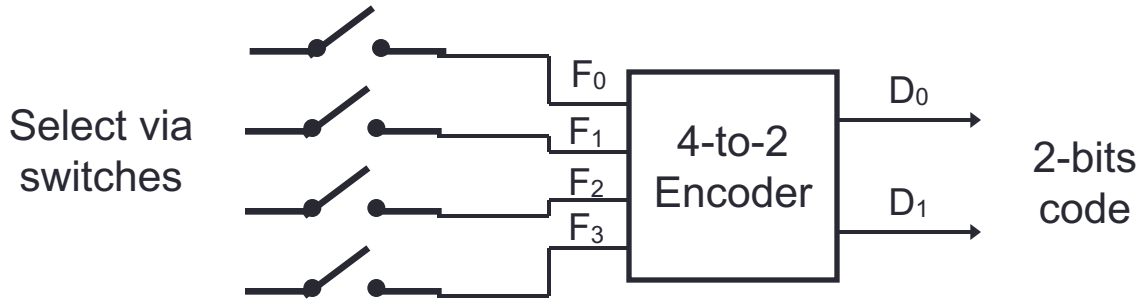
(d) Active-low decoder with AND gate.

READING ASSIGNMENT

- Reducing Decoders
  - Read up DLD pg 136 – 140.

ENCODERS (1/4)

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



ENCODERS (2/4)

- Truth table:

- With K-map, we obtain:

  $D_0 = F1 + F3$
  $D_1 = F2 + F3$

- Circuit:

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

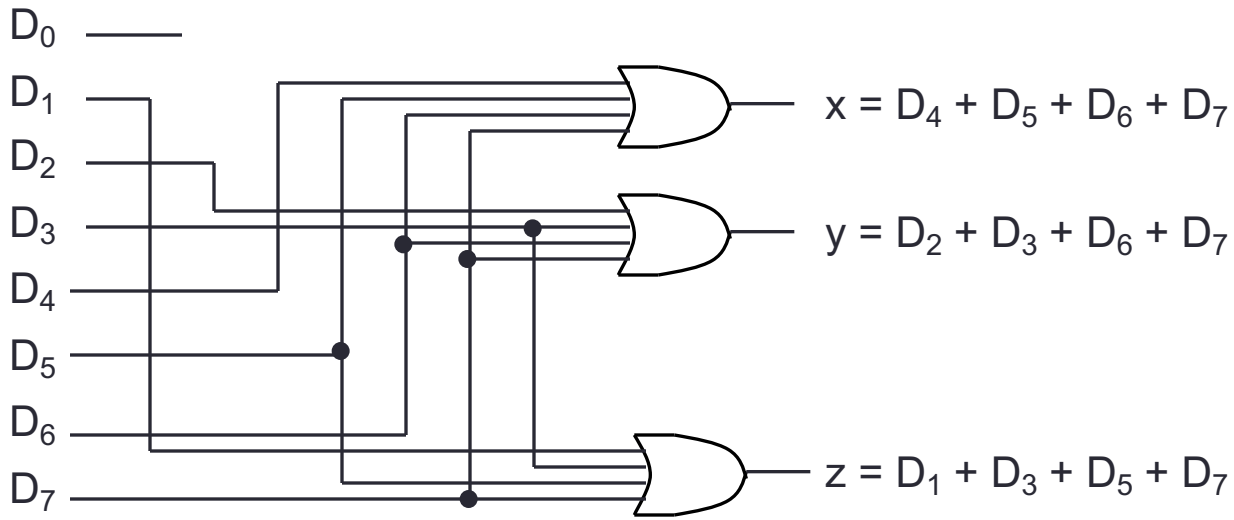
ENCODERS (3/4)

- Example: Octal-to-binary encoder.
  - At any one time, only one input line has a value of 1.
  - Otherwise, 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

ENCODERS (4/4)

- Example: Octal-to-binary encoder.



An 8-to-3 encoder

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

PRIORITY ENCODERS (1/2)

- A priority encoder is one with priority
  - If two or more inputs or 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	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

PRIORITY ENCODERS (2/2)

- Understanding "compact" function table

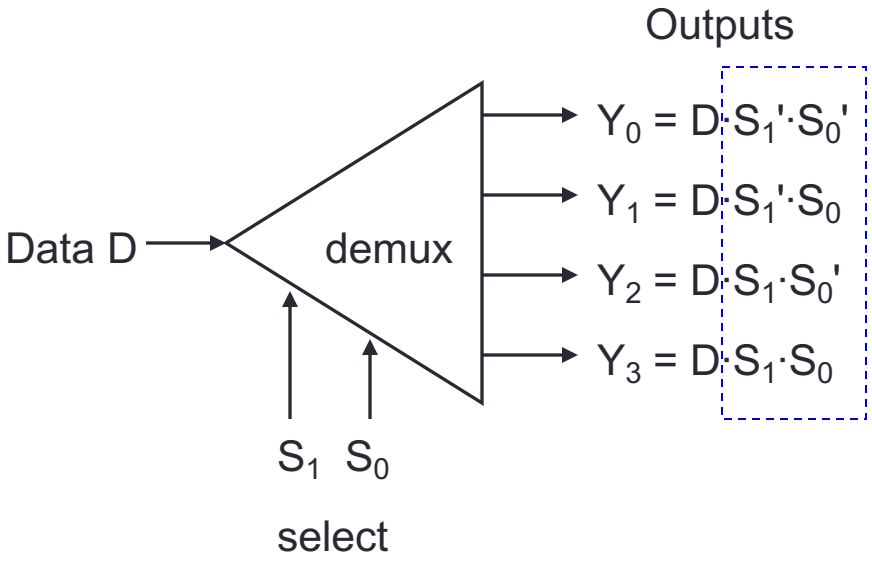
Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

- Exercise: Obtain the simplified expressions for x, y and V.

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
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

DEMULTIPLEXERS (1/2)

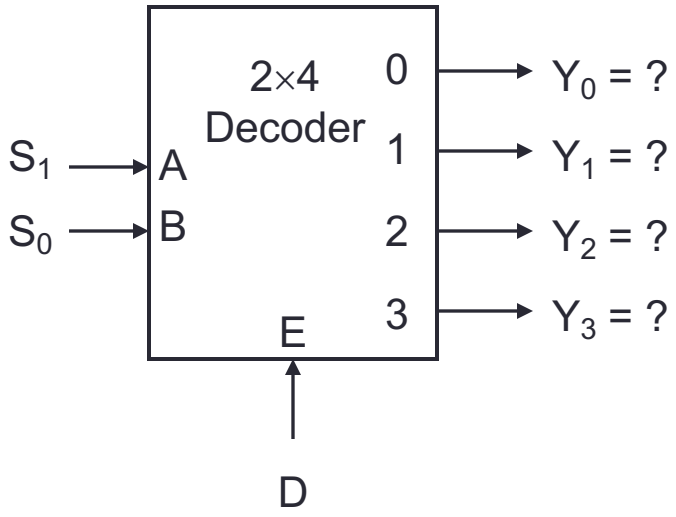
- Given an input line and a set of selection lines, a demultiplexer directs data from the input to one selected output line.
- Example: 1-to-4 demultiplexer.



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

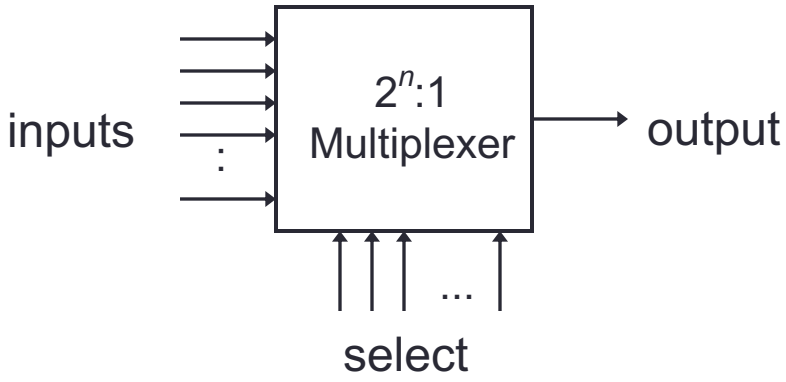
DEMULTIPLEXERS (2/2)

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



MULTIPLEXERS (1/5)

- A multiplexer is a device which 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*.

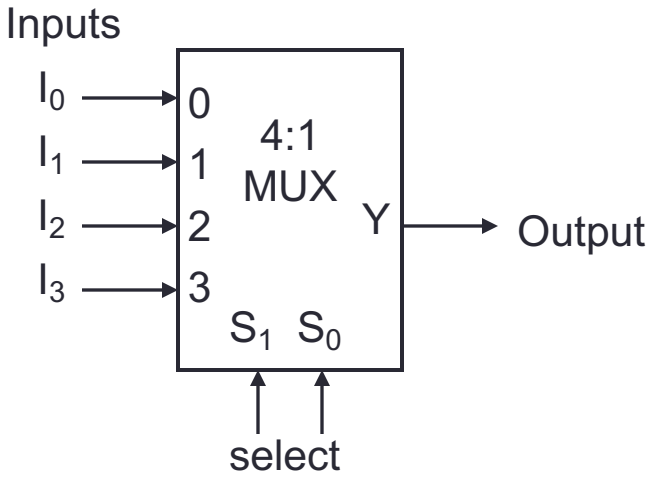


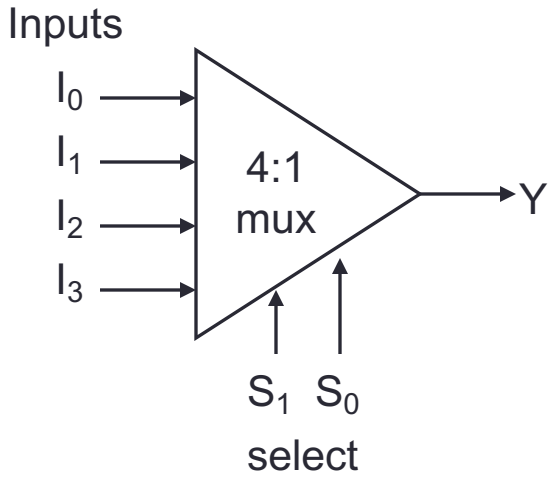
MULTIPLEXERS (2/5)

- 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





MULTIPLEXERS (3/5)

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*)"

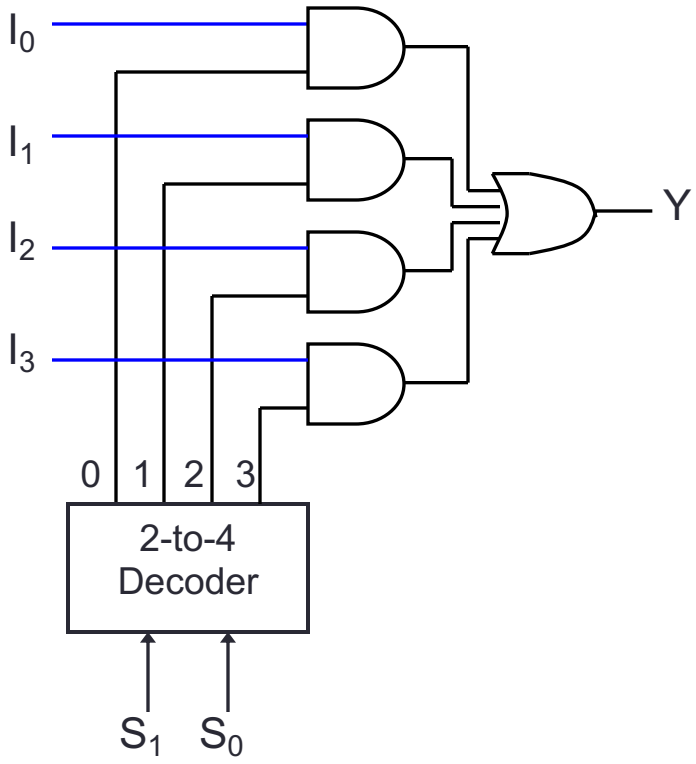
- Example: Output of a 4-to-1 multiplexer is:

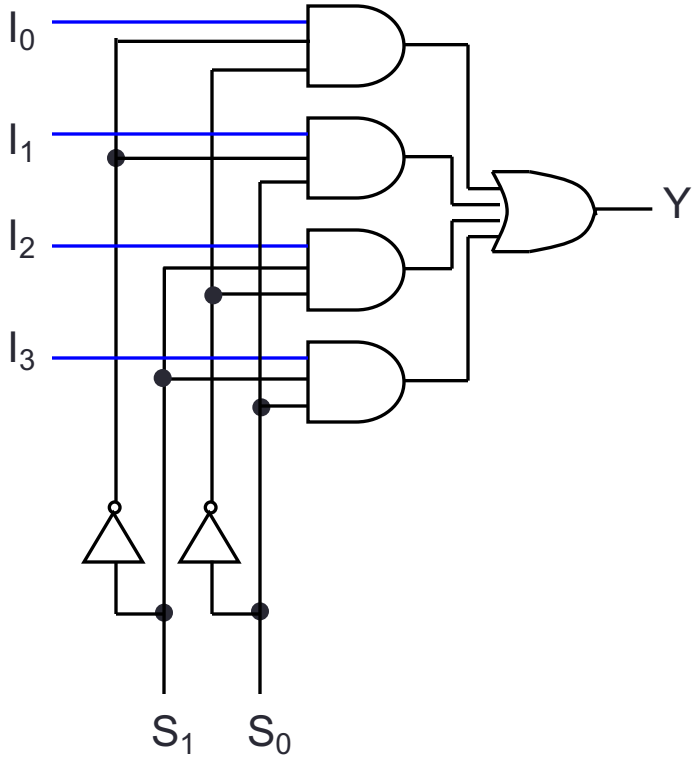
  Y = ?

- 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.

MULTIPLEXERS (4/5)

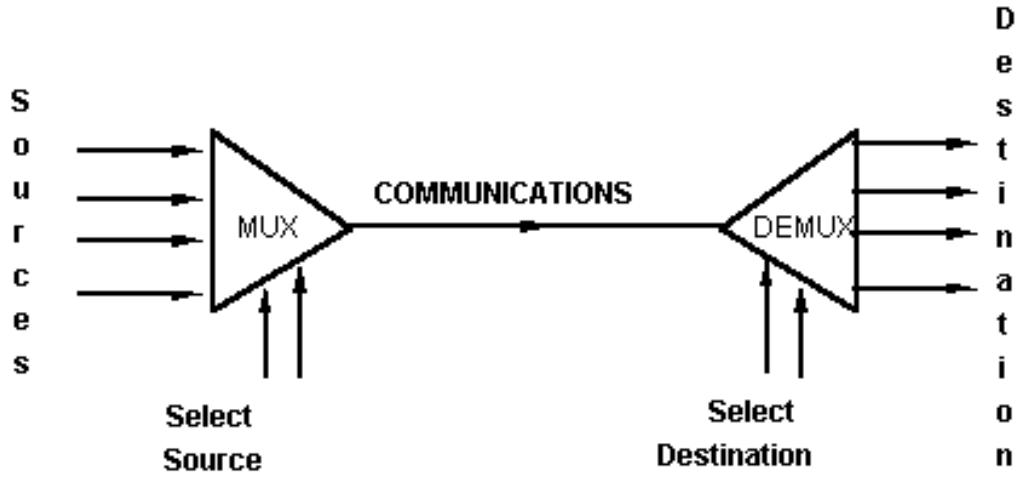
- A 4:1 multiplexer circuit:





MULTIPLEXERS (5/5)

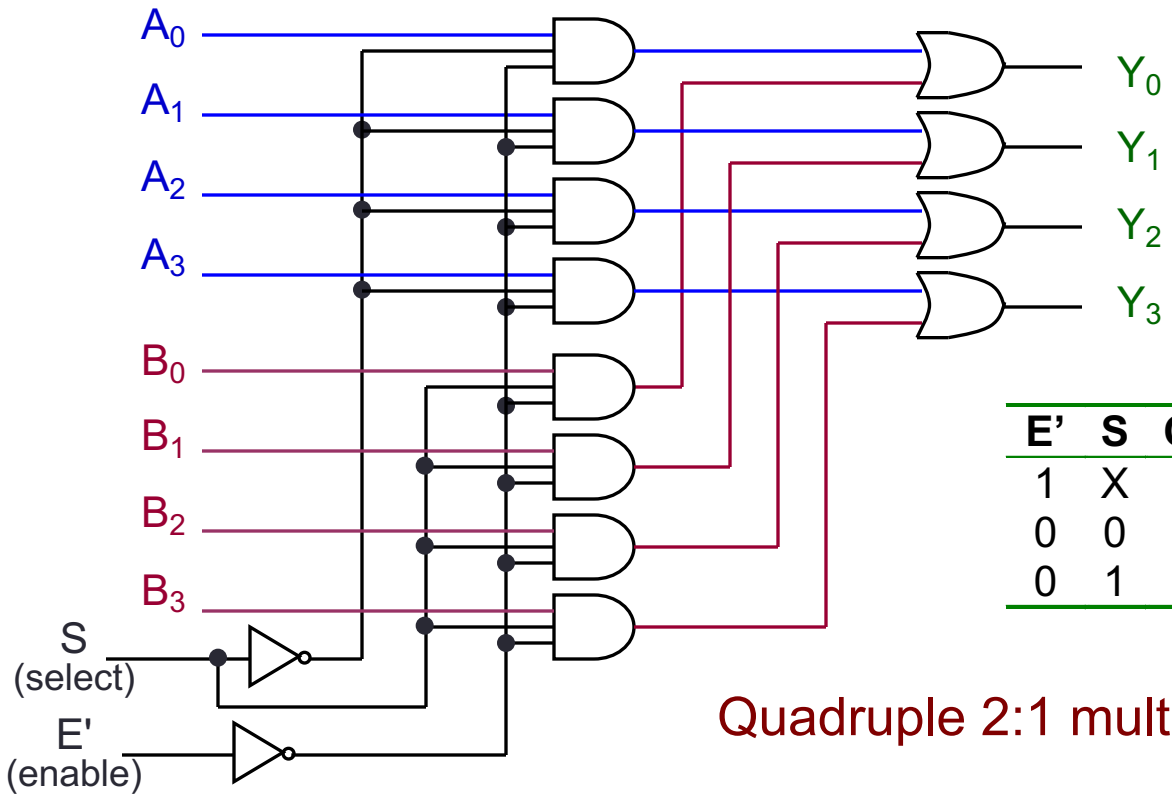
- An application:



- 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.

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.

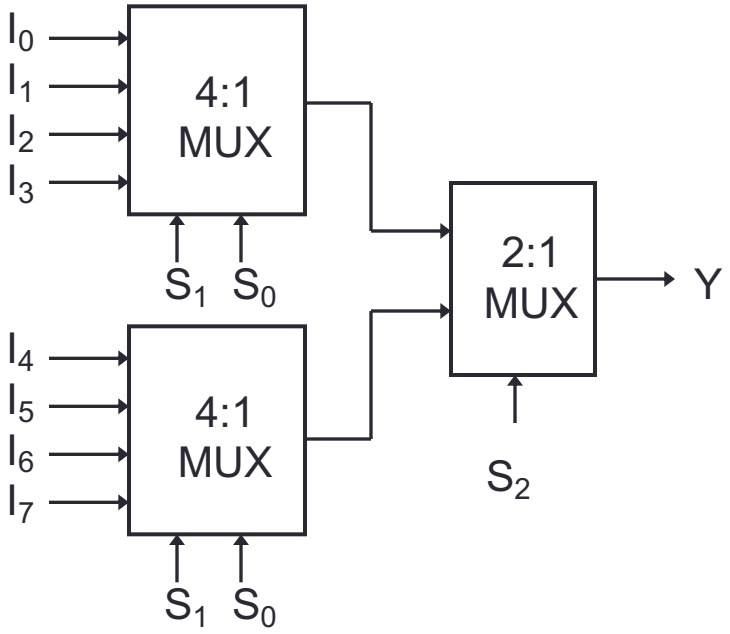


E'	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

Quadruple 2:1 multiplexer

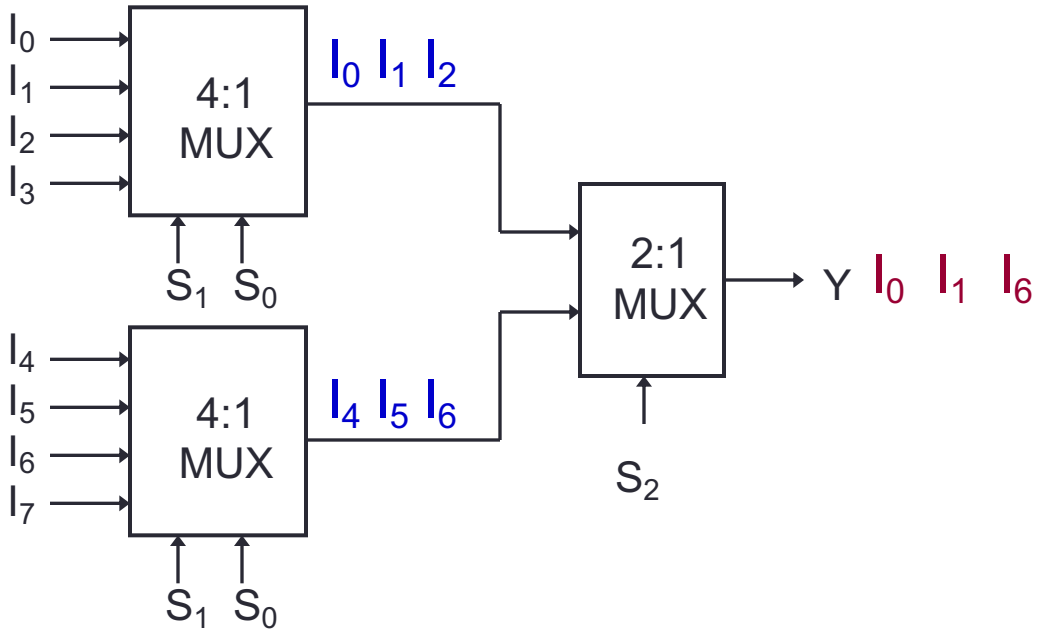
LARGER MULTIPLEXERS (1/4)

- 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

LARGER MULTIPLEXERS (2/4)

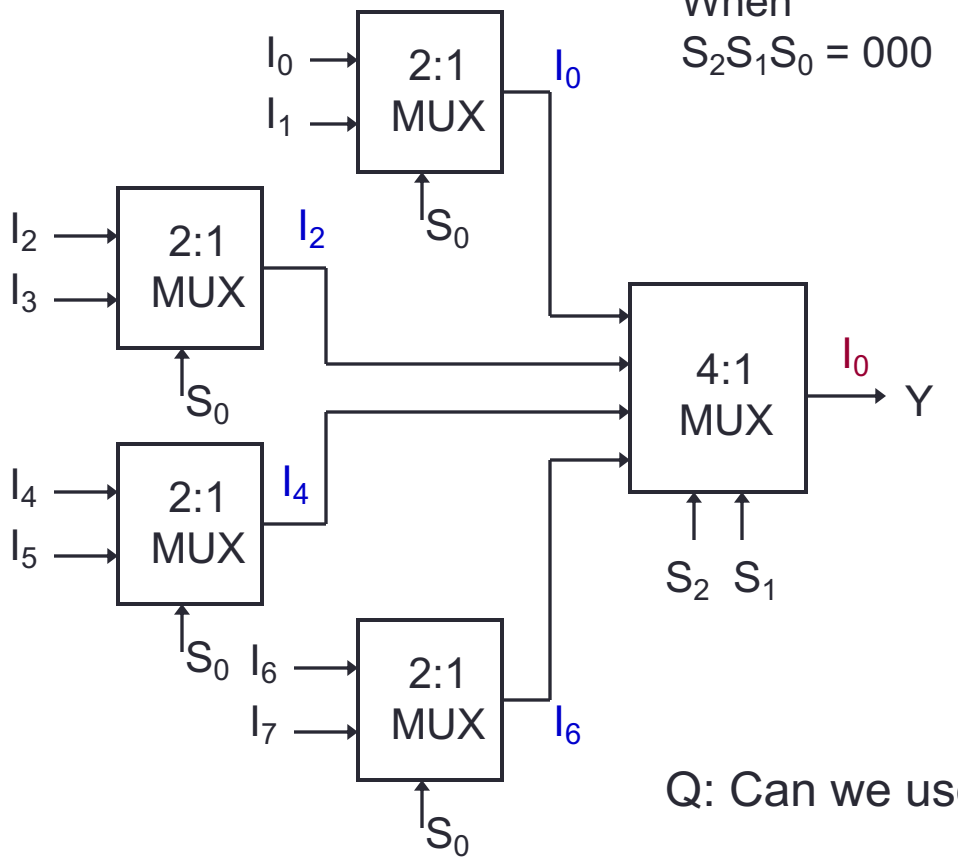


S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

- When $S_2S_1S_0 = 000$
  - When $S_2S_1S_0 = 001$
  - When $S_2S_1S_0 = 110$

LARGER MULTIPLEXERS (3/4)

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

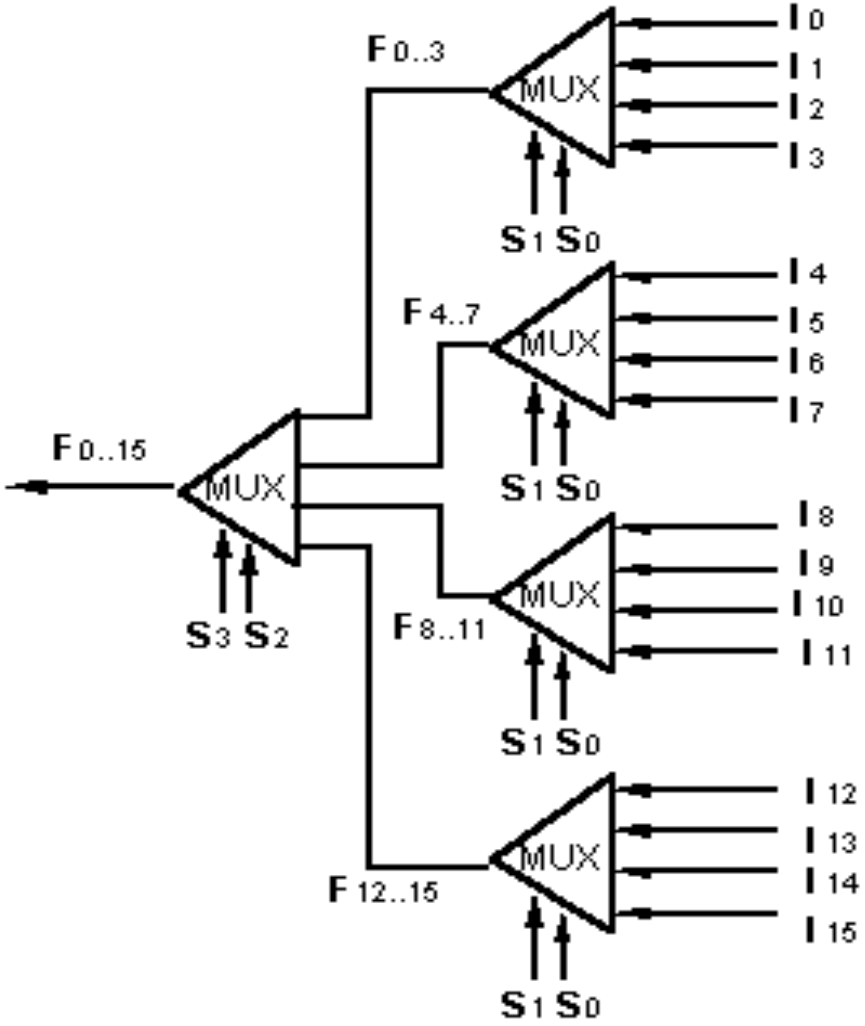


S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

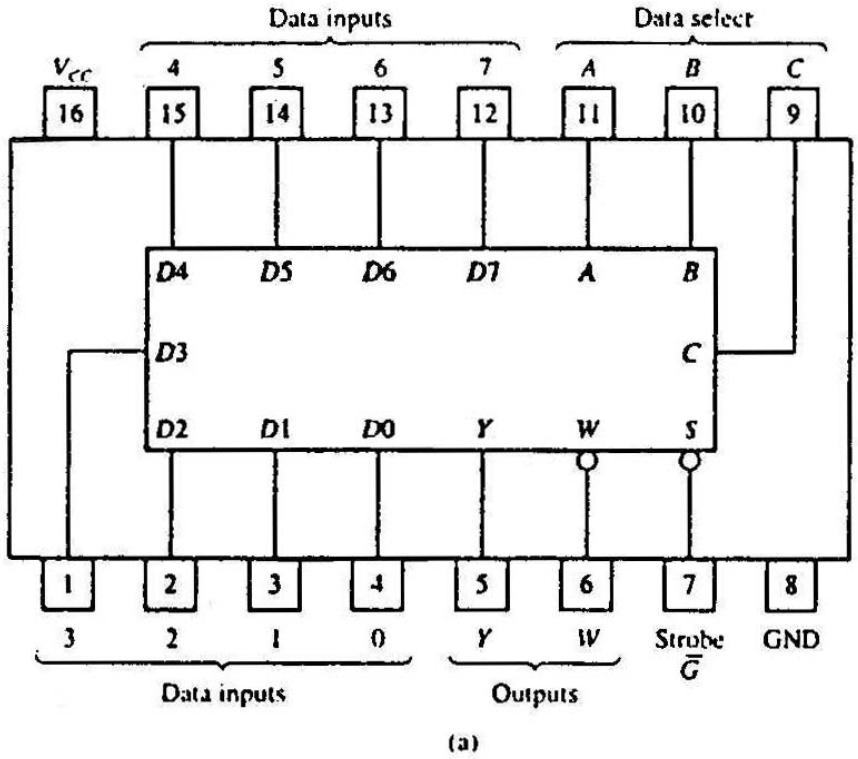
Q: Can we use only 2:1 multiplexers?

LARGER MULTIPLEXERS (4/4)

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



STANDARD MSI MULTIPLEXER (1/2)



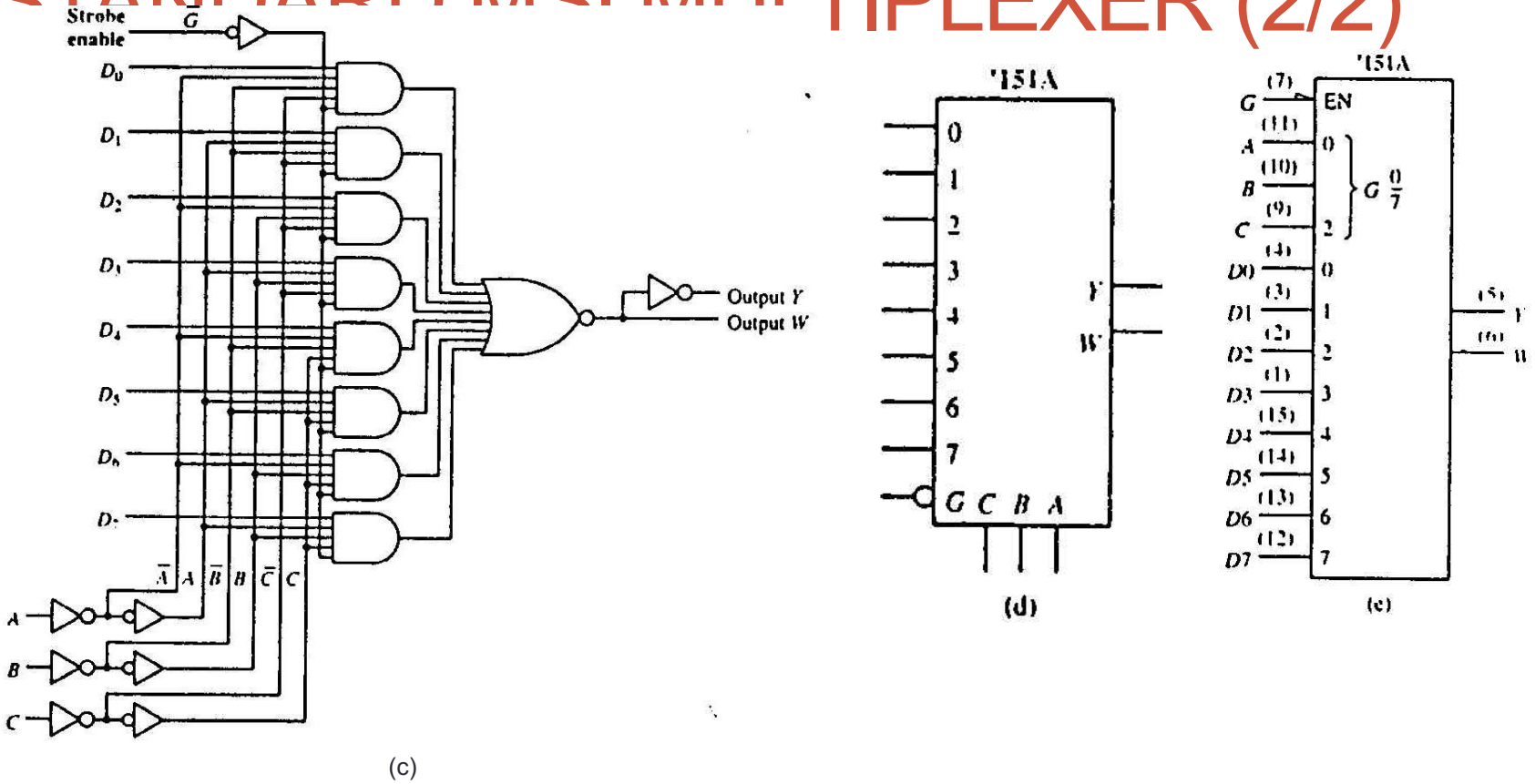
(a)

INPUTS				OUTPUTS	
SELECT			STROBE	Y	W
C	B	A	$\overline{G}$		
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.

STANDARD MSI MULTIPLEXER (2/2)



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

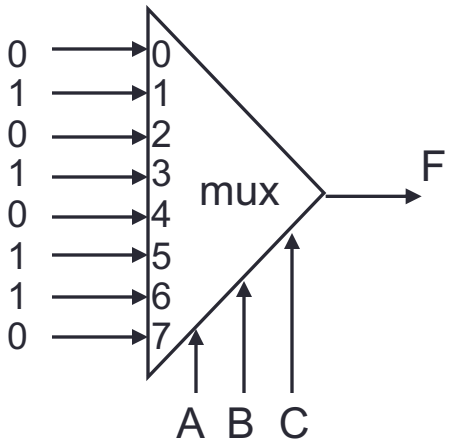
Source: The TTL Data Book Volume 2. Texas Instruments Inc.,1985.

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:
     $F(A,B,C) = A'\cdot B'\cdot C + A'\cdot B\cdot C + A\cdot B'\cdot C + A\cdot B\cdot C'$
     $= \Sigma\, m(1,3,5,6)$
  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.

MULTIPLEXERS: IMPLEMENTING FUNCTIONS (2/3)

- F(A,B,C) = Σ m(1,3,5,6)



This method works because:

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

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

Output = $m_1 + m_3 + m_5 + m_6$

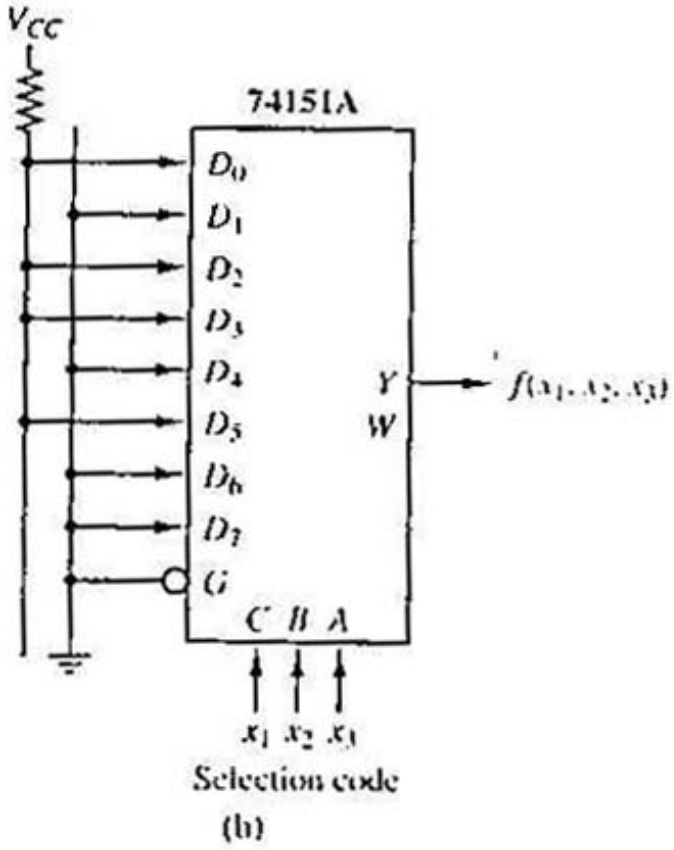
MULTIPLEXERS: IMPLEMENTING FUNCTIONS (3/3)

- Example: Use a 74151A to implement

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

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)



(b)

Realization of $f(x_1,x_2,x_3) = \sum m(0,2,3,5)$.
(a) Truth table.
(b) Implementation with 74151A.

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

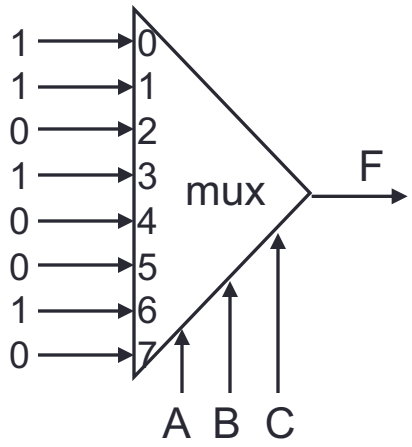
  $$F(A,B,C) = \Sigma\, m(1,3,5,6)$$

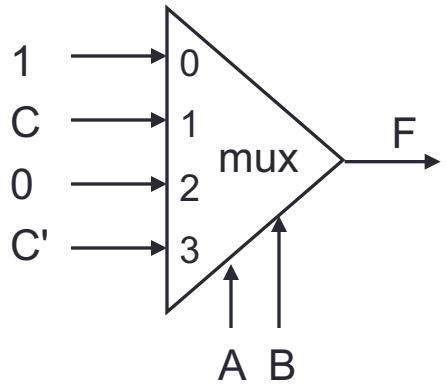
  can be implemented using a 4-to-1 multiplexer (rather than an 8-to-1 multiplexer).

USING SMALLER MULTIPLEXERS (2/6)

- Let's look at this example:

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





- 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'.

USING SMALLER MULTIPLEXERS (3/6)

- Procedure

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

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

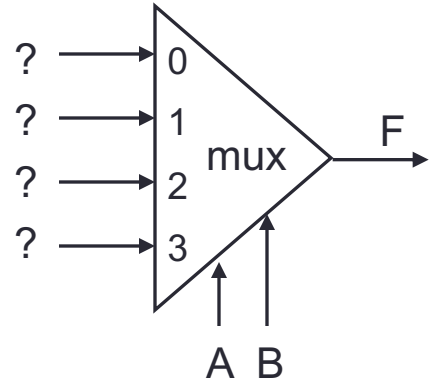
2. Reserve one variable (in our example, we take the least significant one) for input lines of multiplexer, and use the rest for selection lines.

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

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	



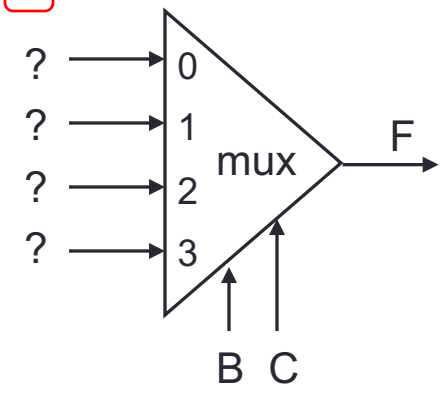
USING SMALLER MULTIPLEXERS (5/6)

- 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)



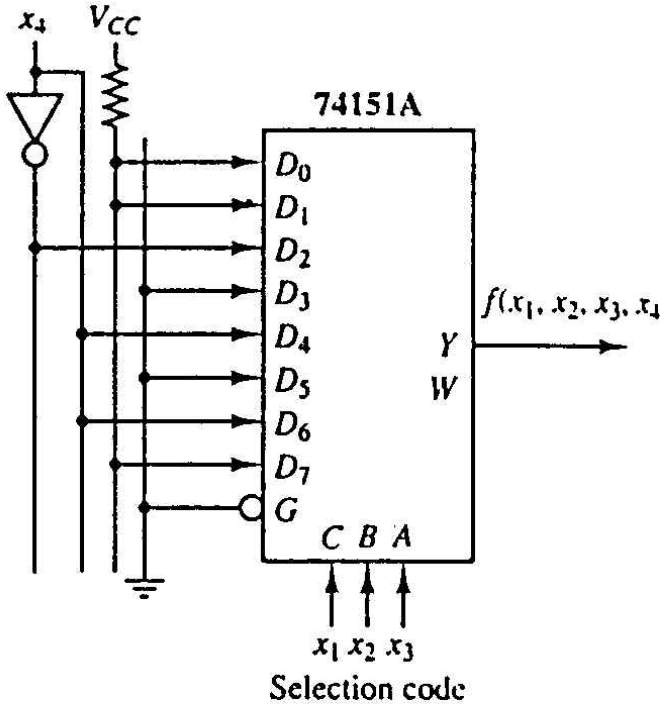
USING SMALLER MULTIPLEXERS (6/6)

- Example: Implement the function below with 74151A:

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

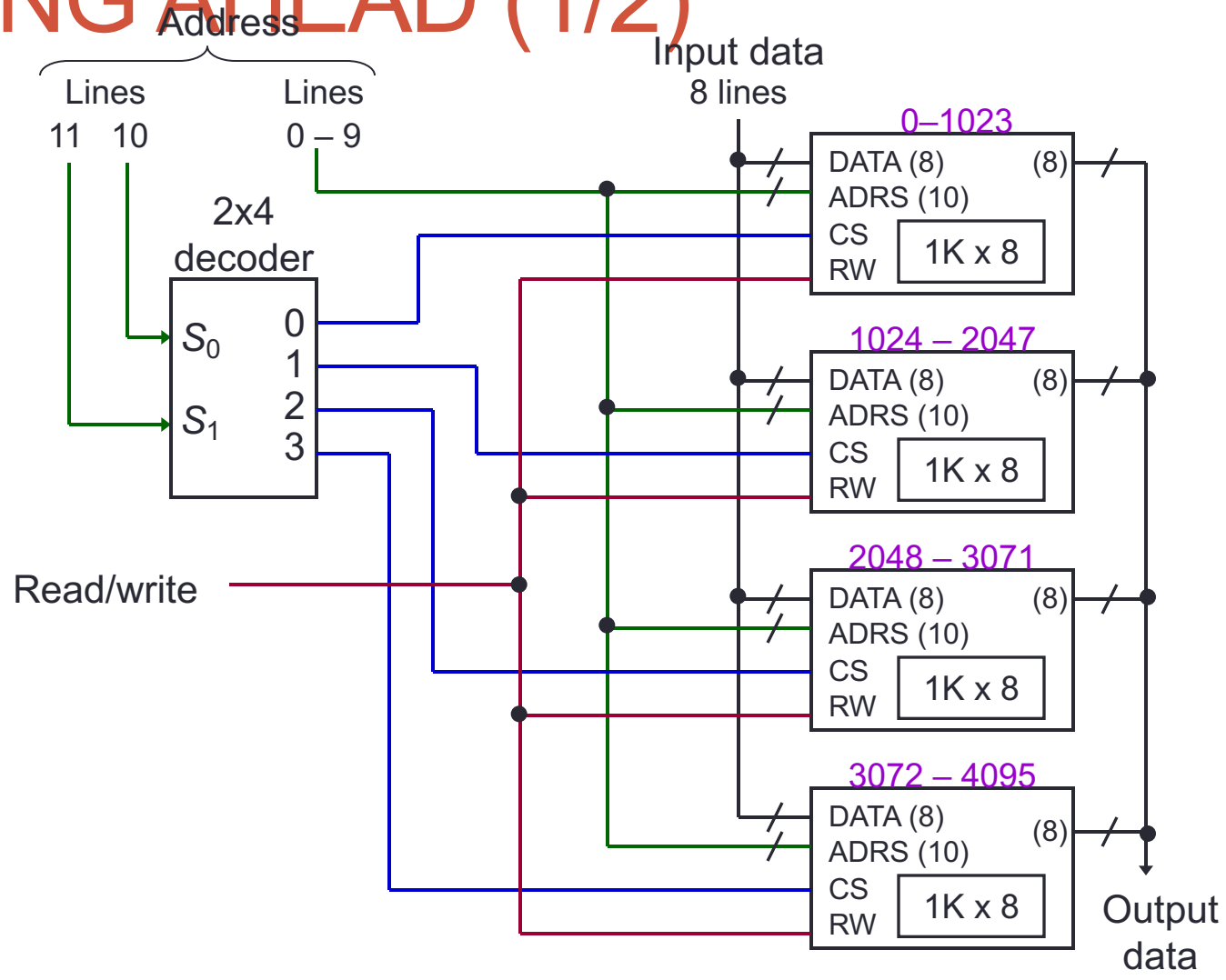
	C	B	A				Y
i	X_1	X_2	X_3	X_4	f	f	
0	0	0	0	0	1		
	0	0	0	1	1	1	$D_0 = 1$
1	0	0	1	0	1		
	0	0	1	1	1	1	$D_1 = 1$
2	0	1	0	0	1		
	0	1	0	1	0	$\bar{X}_4$	$D_2 = \bar{X}_4$
3	0	1	1	0	0		
	0	1	1	1	0	0	$D_3 = 0$
4	1	0	0	0	0		
	1	0	0	1	1	X_4	$D_4 = X_4$
5	1	0	1	0	0		
	1	0	1	1	0	0	$D_5 = 0$
6	1	1	0	0	0		
	1	1	0	1	1	X_4	$D_6 = X_4$
7	1	1	1	0	1		
	1	1	1	1	1	1	$D_7 = 1$

(a)

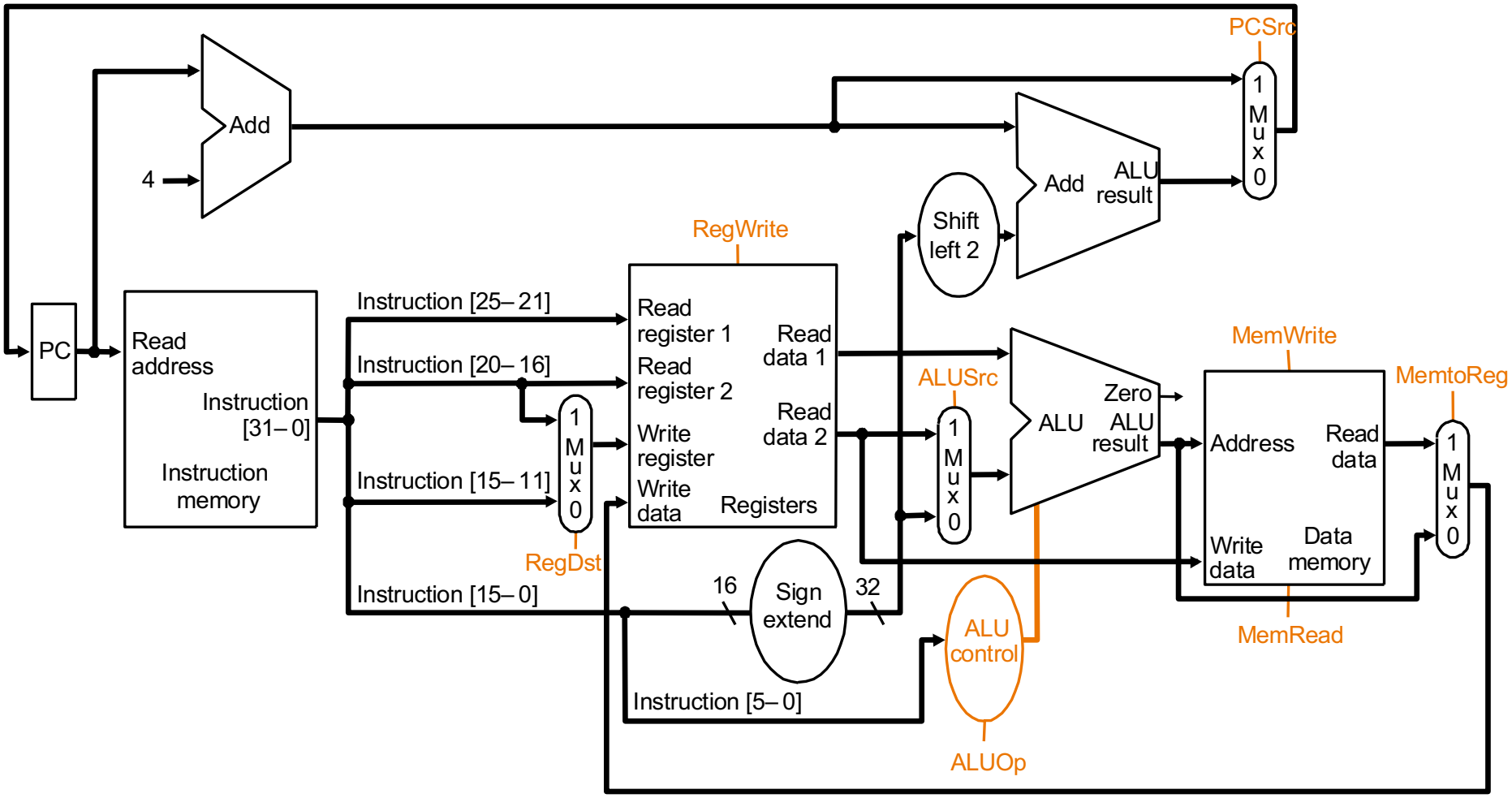


(b)

PEEKING AHEAD (1/2)



PEEKING AHEAD (2/2)



Q&A