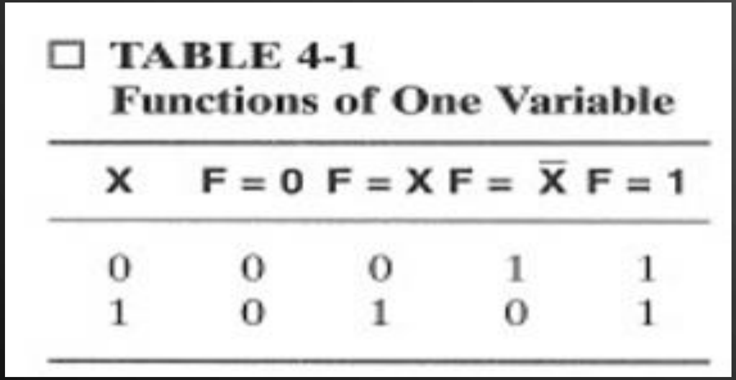
LEARNING TOPICS

- ❑Combinational Functional blocks: Specific combinational functions and corresponding combinational circuits are referred as functional blocks.
- ❑Rudimentary logic functions
- ❑Decoding using Decoders
    - Implementing Combinational Functions with Decoders
- ❑Encoding using Encoders
- ❑Selecting using Multiplexers
    - Implementing Combinational Functions with Multiplexers

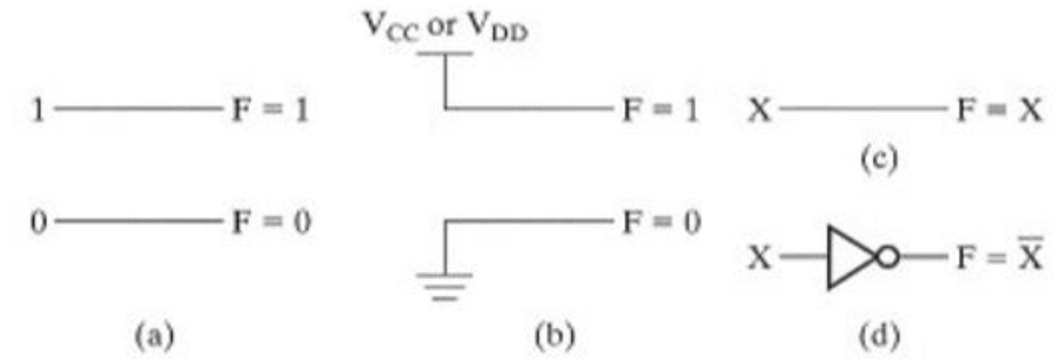
RUDIMENTARY FUNCTIONAL BLOCKS

- Value fixing, transferring, inverting and Enabling



☐ **TABLE 4-1**
Functions of One Variable

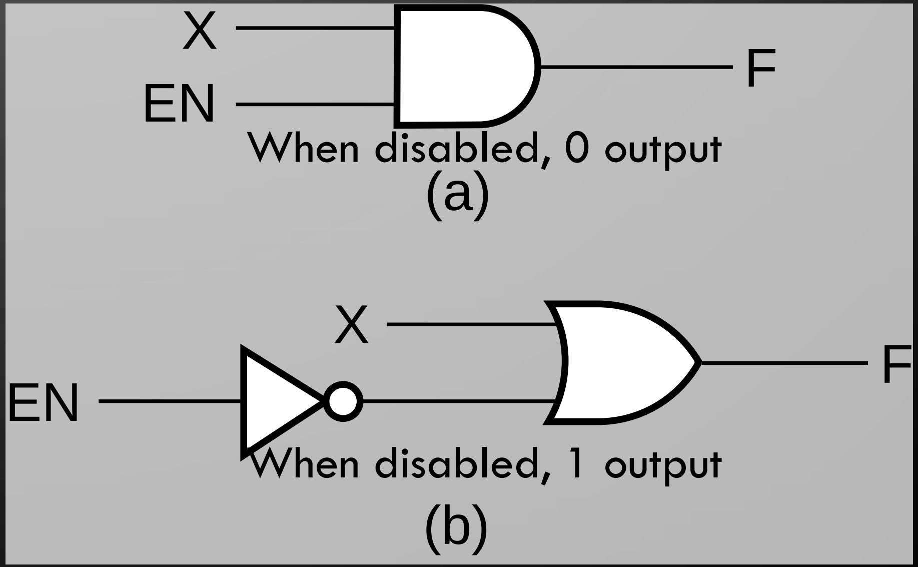
X	F = 0	F = X	F = $\overline{X}$	F = 1
0	0	0	1	1
1	0	1	0	1



☐ **FIGURE 4-2**
Implementation of Functions of a Single Variable *X*

ENABLING FUNCTION

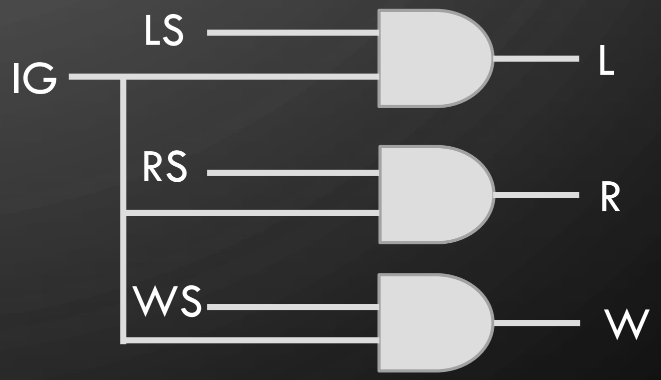
- Enabling permits an input signal to pass through to an output
- Disabling blocks an input signal from passing through to an output, replacing it with a fixed value
- The value on the output when it is disable can be Hi-Z (as for three-state buffers and transmission gates), 0 , or 1



EXAMPLE: CAR ELECTRICAL CONTROL USING ENABLING

In most automobiles, the lights, radio and power windows operate only if ignition switch is turned on. In that case, the ignition switch acts as an "enabling" signal. Suppose that we model this automotive system using the following variables and definitions:

- Ignition Switch: IG – 1 if on, 0 if off
- Light Switch: LS – 1 if on, 0 if off
- Radio Switch: RS – 1 if on, 0 if off
- Power Window Switch: WS – 1 if on, 0 if off
- Light: L – 1 if on, 0 if off
- Radio: R – 1 if on, 0 if off
- Power Window: W – 1 if on, 0 if off



DECODING

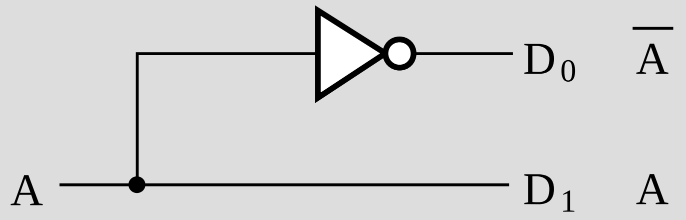
- Decoding - the conversion of an n-bit input code to an m-bit output code with $n \leq m \leq 2^n$, such that each valid code word produces a unique output code
- Circuits that perform decoding are called decoders
- Here, functional blocks for decoding are
  - called n-to-m line decoders, where $m \leq 2^n$, and
  - generate 2^n (or fewer) minterms for the n input variables

DECODER EXAMPLES

- 1-to-2-Line Decoder

A	**D_0**	**D_1**
0	1	0
1	0	1

(a)



(b)

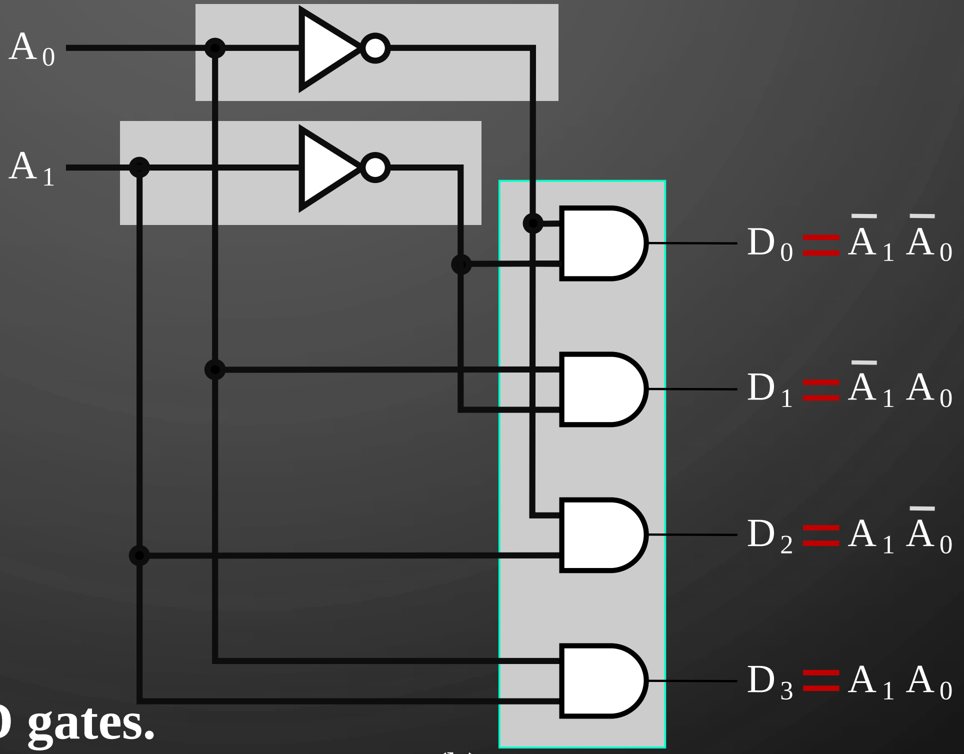
DECODER EXAMPLES

2 to 4 line decoder

A_1	A_0	D_0	D_1	D_2	D_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

(a)

- **Note that the 2-4-line made up of 2 1-to-2-line decoders and 4 AND gates.**



(b)

DECODER EXPANSION

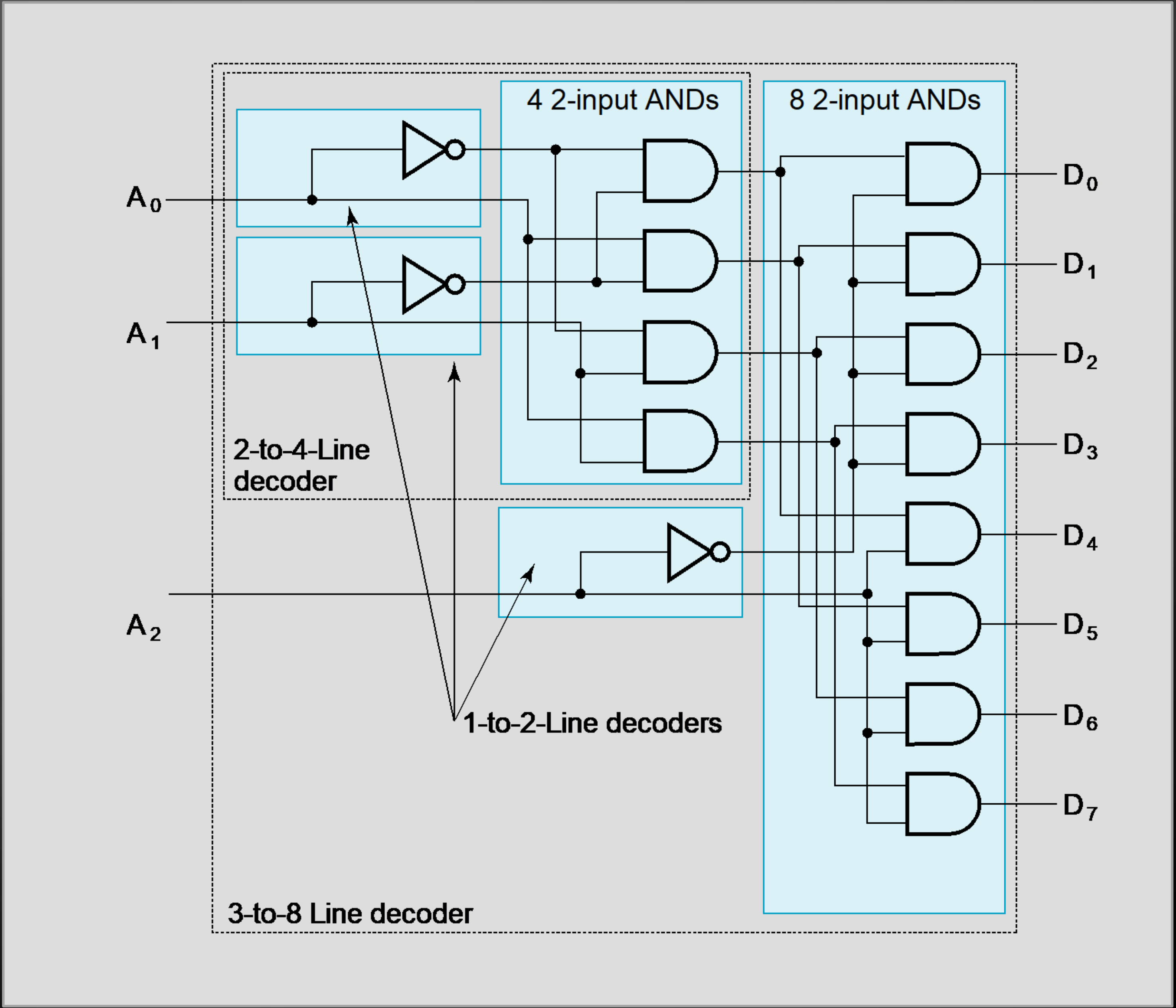
- General procedure given in book for any decoder with n inputs and 2^n outputs.
- This procedure builds a decoder backward from the outputs.
- These decoders are then designed using the same procedure until 1-to-2-line decoders are reached.
- Book page 123 has the complete rule

DECODER EXPANSION - EXAMPLE 1

3-to-8-line decoder

- Number of output ANDs = 8
- Number of inputs to decoders driving output ANDs = 3
- Closest possible split to equal
  - 2-to-4-line decoder
  - 1-to-2-line decoder
- 2-to-4-line decoder
  - Number of output ANDs = 4
  - Number of inputs to decoders driving output ANDs = 2
- Closest possible split to equal
  - Two 1-to-2-line decoders

RESULT

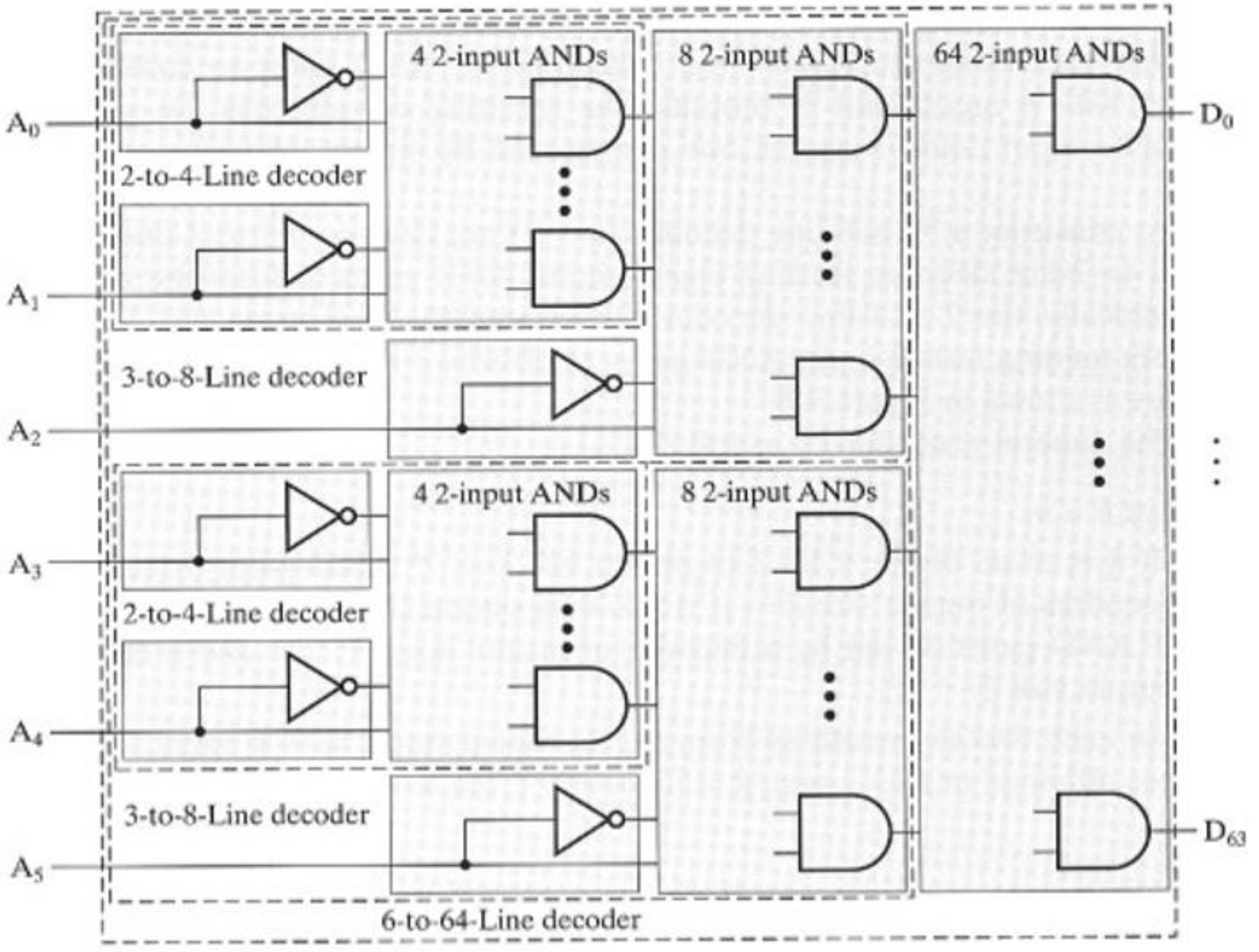


4 2-input ANDs
8 2-input ANDs
A_0
A_1
A_2
D_0
D_1
D_2
D_3
D_4
D_5
D_6
D_7
2-to-4-Line decoder
1-to-2-Line decoders
3-to-8 Line decoder

DECODER EXPANSION - EXAMPLE 2

- 6-to-64-line decoder
  - Number of output ANDs = 64
  - Number of inputs to decoders driving output ANDs = 6
  - Closest possible split to equal
    - two 3-to-8-line decoder
  - 3-to-8-line decoder
    - Number of output ANDs = 8
    - Number of inputs to decoders driving output ANDs = 3
    - Closest possible split to equal
      - 2-to-4-line decoders
  - Complete using known 3-8 and 2-to-4 line decoders

RESULT



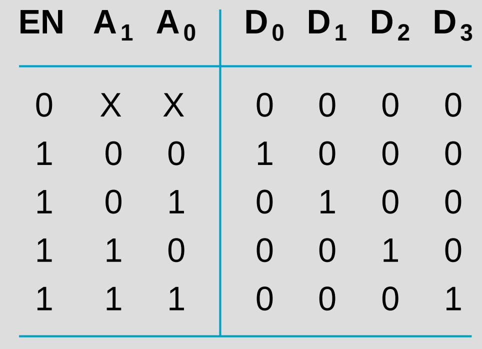
☐ **FIGURE 4-9**
A 6-to-64-Line Decoder

DECODER EXPANSION - EXAMPLE 3

- 7-to-128-line decoder
  - Number of output ANDs = 128
  - Number of inputs to decoders driving output ANDs = 7
  - Closest possible split to equal
    - 4-to-16-line decoder
    - 3-to-8-line decoder
  - 4-to-16-line decoder
    - Number of output ANDs = 16
    - Number of inputs to decoders driving output ANDs = 2
    - Closest possible split to equal
      - two 2-to-4-line decoders
  - Complete using known 3-8 and 2-to-4 line decoders

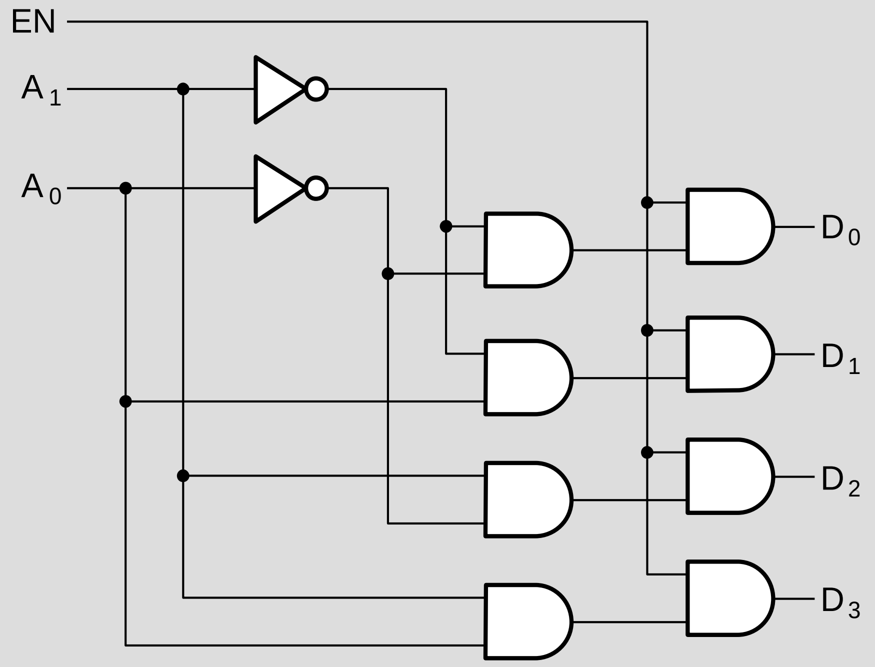
DECODER WITH ENABLE

- In general, attach *m*-enabling circuits to the outputs
- See truth table below for function
  - Note use of X’s to denote both 0 and 1
  - Combination containing two X’s represent four binary combinations



EN	A_1	A_0	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
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

(a)



(b)

COMBINATIONAL LOGIC IMPLEMENTATION - DECODER AND OR GATES

Implement m functions of n variables with:

- Sum-of-minterms expressions
- One n-to-2n-line decoder
- m OR gates, one for each output

Approach 1:

- Find the truth table for the functions
- Make a connection to the corresponding OR from the corresponding decoder output wherever a 1 appears in the truth table

Approach 2

- Find the minterms for each output function
- OR the minterms together

COMBINATIONAL LOGIC IMPLEMENTATION - DECODER AND OR GATES

Binary Adder Circuit

$$S(X,Y,Z) = \sum m\,(1,2,4,7)$$

$$C(X,Y,Z) = \sum m\,(3,5,6,7)$$

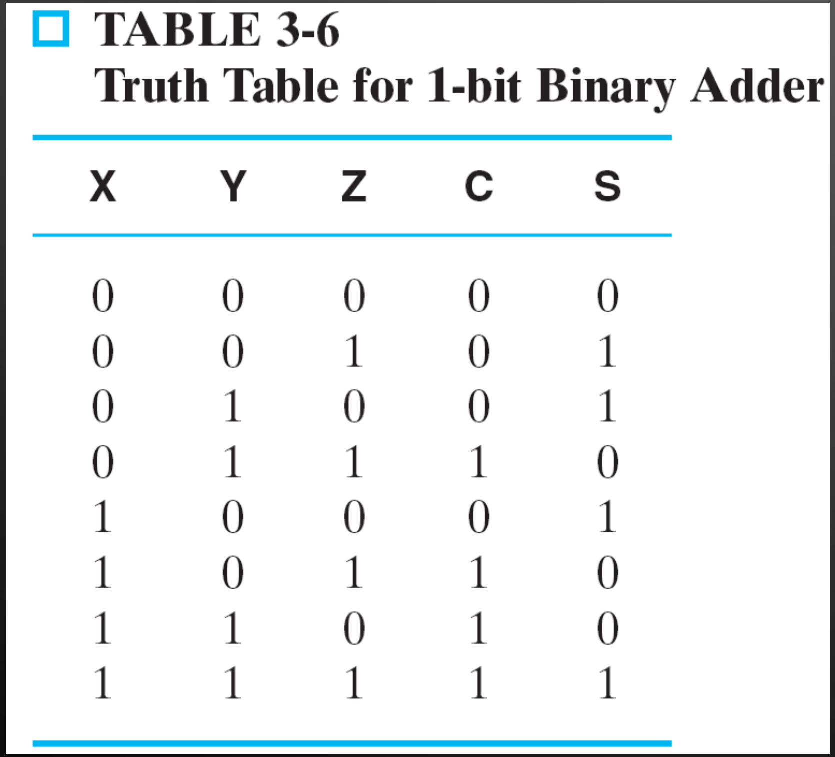
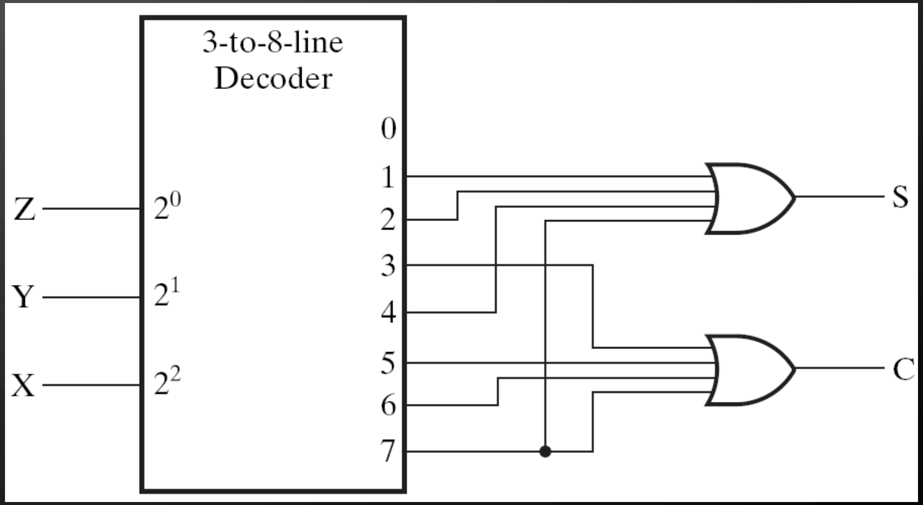


TABLE 3-6
Truth Table for 1-bit Binary Adder

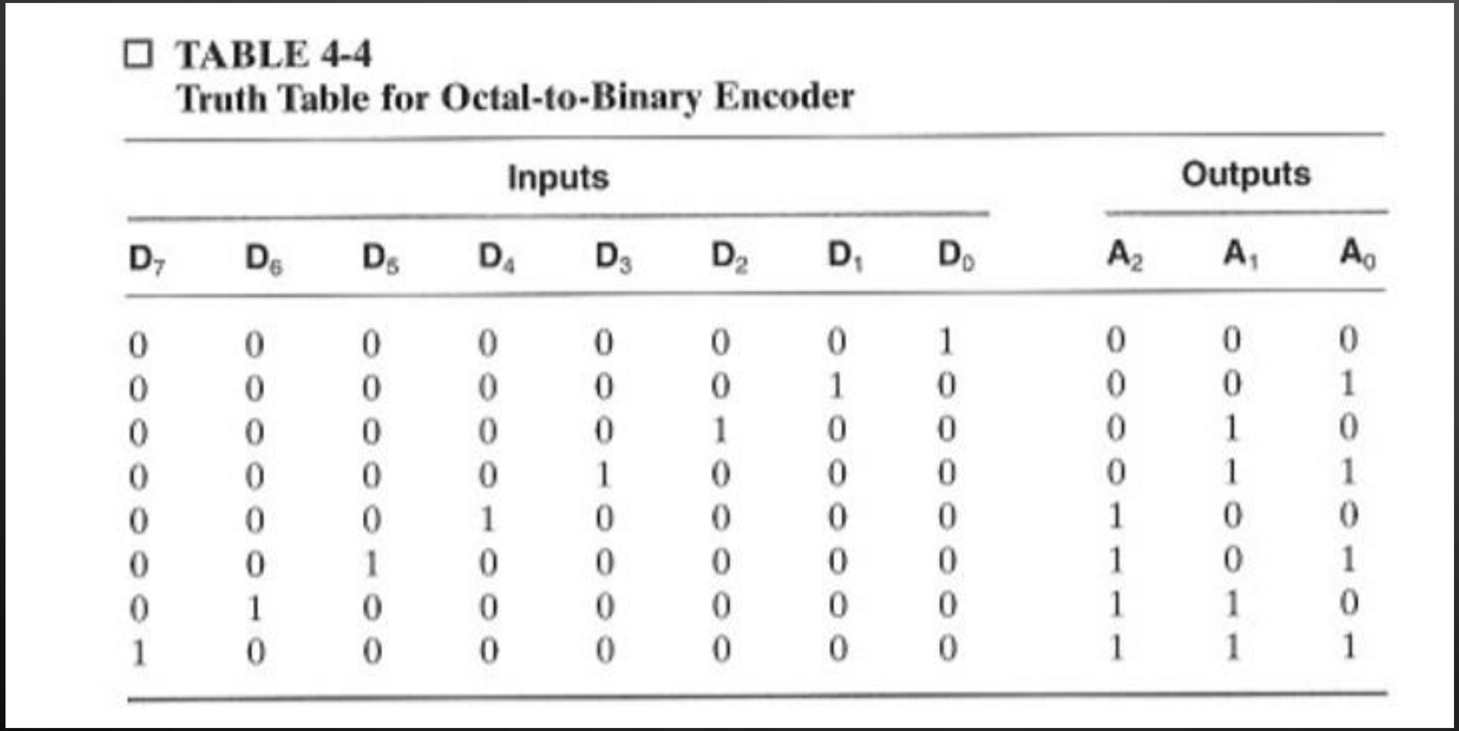
X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



ENCODING

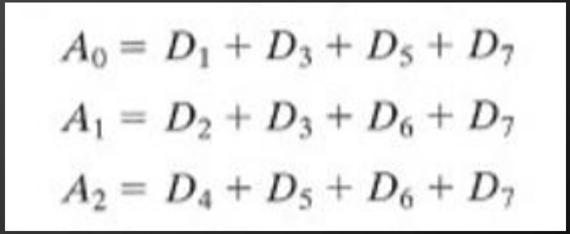
- Encoding - the opposite of decoding - the conversion of an m-bit input code to a n-bit output code with $n \leq m \leq 2^{\wedge} n$ such that each valid code word produces a unique output code
- Circuits that perform encoding are called encoders
- An encoder has 2^n (or fewer) input lines and n output lines which generate the binary code corresponding to the input values
- Typically, an encoder converts a code containing exactly one bit that is 1 to a binary code corresponding to the position in which the 1 appears.

EXAMPLE : OCTAL TO BINARY ENCODER



☐ **TABLE 4-4**
Truth Table for Octal-to-Binary Encoder

Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1



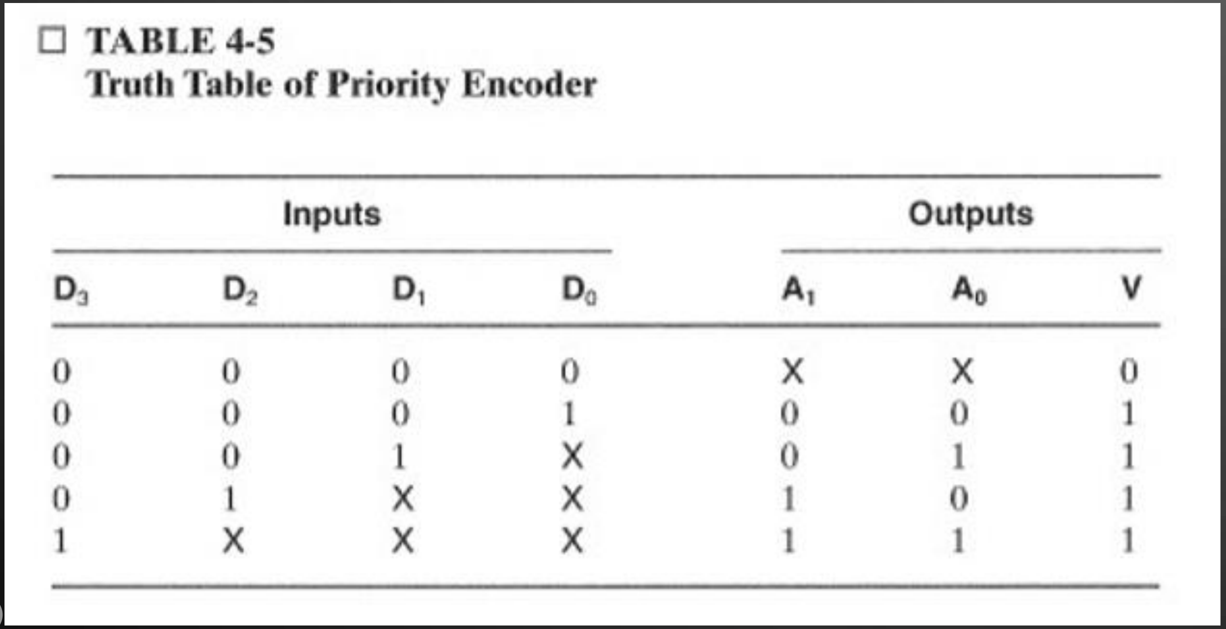
$$A_0 = D_1 + D_3 + D_5 + D_7$$
$$A_1 = D_2 + D_3 + D_6 + D_7$$
$$A_2 = D_4 + D_5 + D_6 + D_7$$

PRIORITY ENCODER

- If more than one input bit is 1, then the encoder just designed does not work.
- One encoder that can accept all possible combinations of input values and produce a meaningful result is a priority encoder.
- Among the 1s that appear, it selects the most significant input position (or the least significant input position) containing a 1 and responds with the corresponding binary code for that position.

PRIORITY ENCODER EXAMPLE

Priority encoder with 4 inputs (D3, D2, D1, D0) - highest priority to most significant 1 present - Code outputs A1, A0 and V ,where V indicates at least one 1 present.



☐ **TABLE 4-5**
Truth Table of Priority Encoder

Inputs				Outputs		
D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

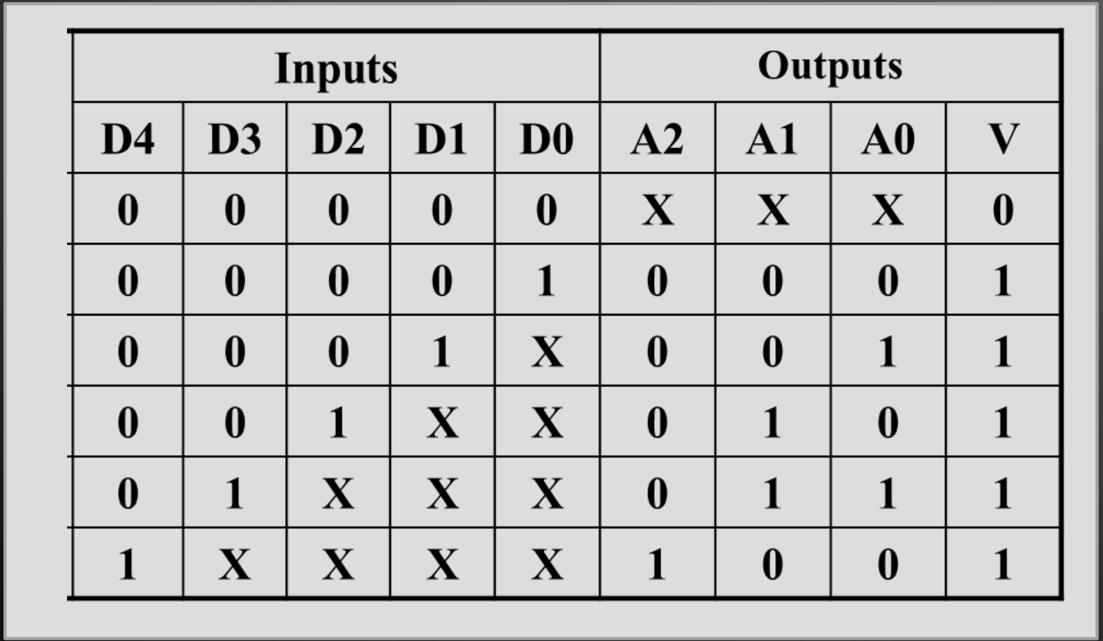
$A_1 = \overline{D}_3 D_2 + D_3 = D_2 + D_3$ [Boolean identity]

$A_0 = D_3 + \overline{D}_3 \overline{D}_2 D_1 = D_3 + \overline{D}_2 D_1$

$V = D_0 + D_1 + D_2 + D_3$

PRIORITY ENCODER EXAMPLE

- Priority encoder with 5 inputs (D4, D3, D2, D1, D0) - highest priority to most significant 1 present - Code outputs A2, A1, A0 and V where V indicates at least one 1 present.



Inputs					Outputs			
D4	D3	D2	D1	D0	A2	A1	A0	V
0	0	0	0	0	X	X	X	0
0	0	0	0	1	0	0	0	1
0	0	0	1	X	0	0	1	1
0	0	1	X	X	0	1	0	1
0	1	X	X	X	0	1	1	1
1	X	X	X	X	1	0	0	1

- Could use a K-map to get equations, but can be read directly from table and manually optimized if careful:
- $A_2 = D_4$
- $A_1 = \overline{D}_4 D_3 + \overline{D}_4 \overline{D}_3 D_2$
- $A_0 = \overline{D}_4 D_3 + \overline{D}_4 \overline{D}_3 \overline{D}_2 D_1$

SELECTING

- Selecting of data or information is a critical function in digital systems and computers
- Circuits that perform selecting have:
  - A set of information inputs from which the selection is made
  - A single output
  - A set of control lines for making the selection
- Logic circuits that perform selecting are called multiplexers

MULTIPLEXERS (MUX)

- A multiplexer selects information from an input line and directs the information to an output line. It is often called data selector.

- A typical multiplexer has n control inputs ($S_{n-1}, \dots S_0$) called selection inputs, 2^n information inputs ($I_{2^n-1}, \dots I_0$), and one output Y
- A multiplexer can be designed to have m information inputs with $m < 2^n$ as well as n selection inputs

2-TO-1-LINE MULTIPLEXER

Since $2 = 2^1$, $n = 1$

The single selection variable S has two values:

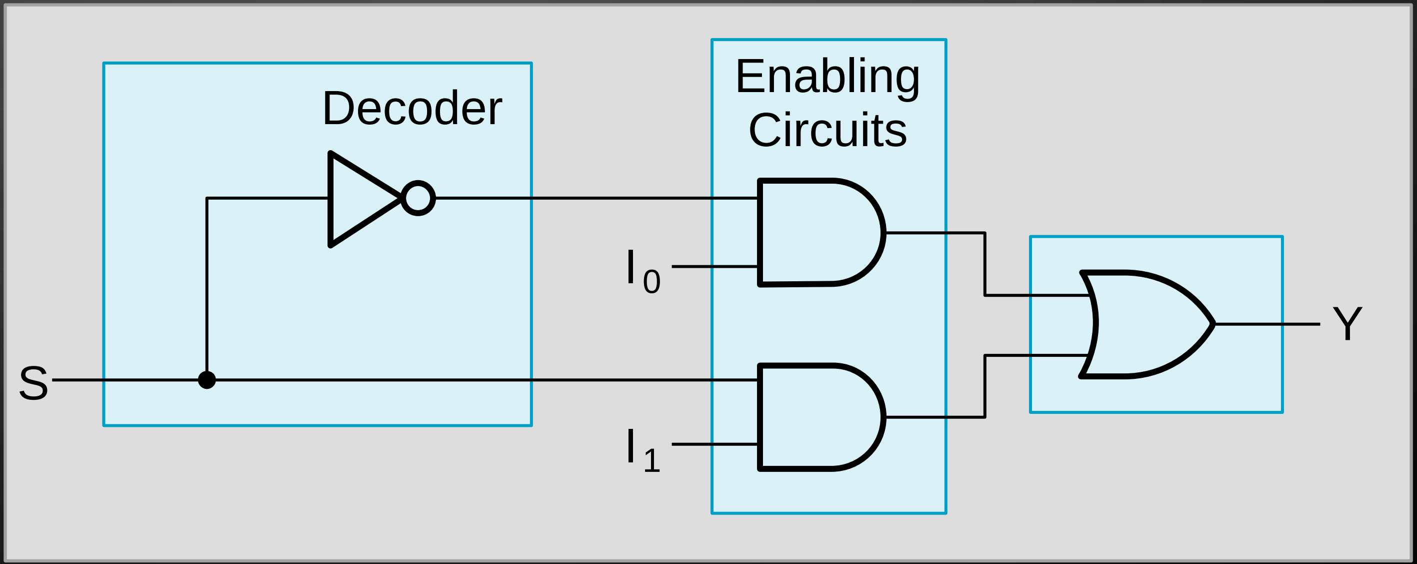
$S = 0$ selects input I_0

$S = 1$ selects input I_1

The equation:

$$Y = \bar{S} I_0 + S I_1$$

The circuit:

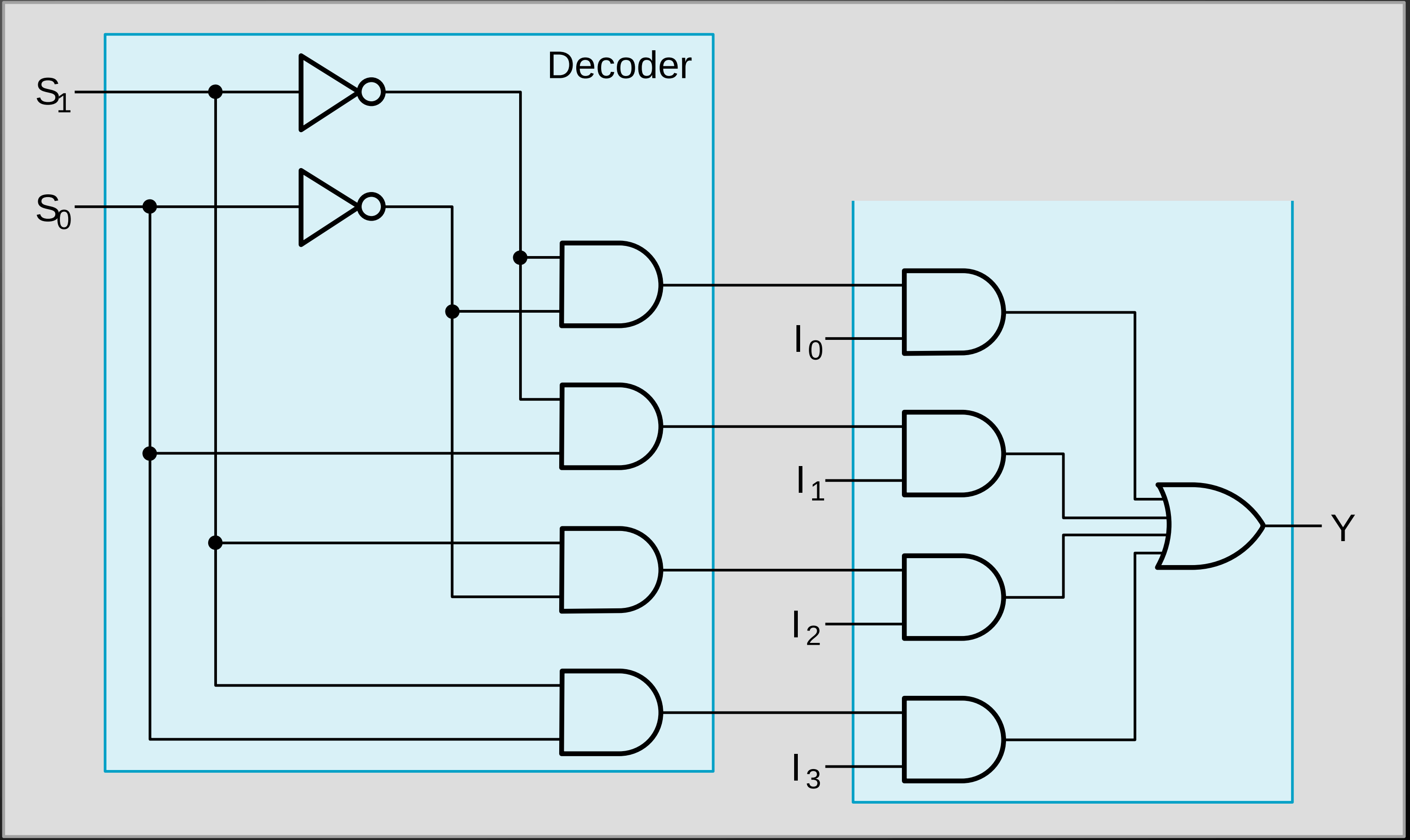


EXAMPLE: 4-TO-1-LINE MULTIPLEXER

- 2-to-2^2-line decoder

S_1	S_0	D_0	D_1	D_2	D_3	Y = ?
0	0	1	0	0	0	I0
0	1	0	1	0	0	I1
1	0	0	0	1	0	I2
1	1	0	0	0	1	I3

Total gate cost =22

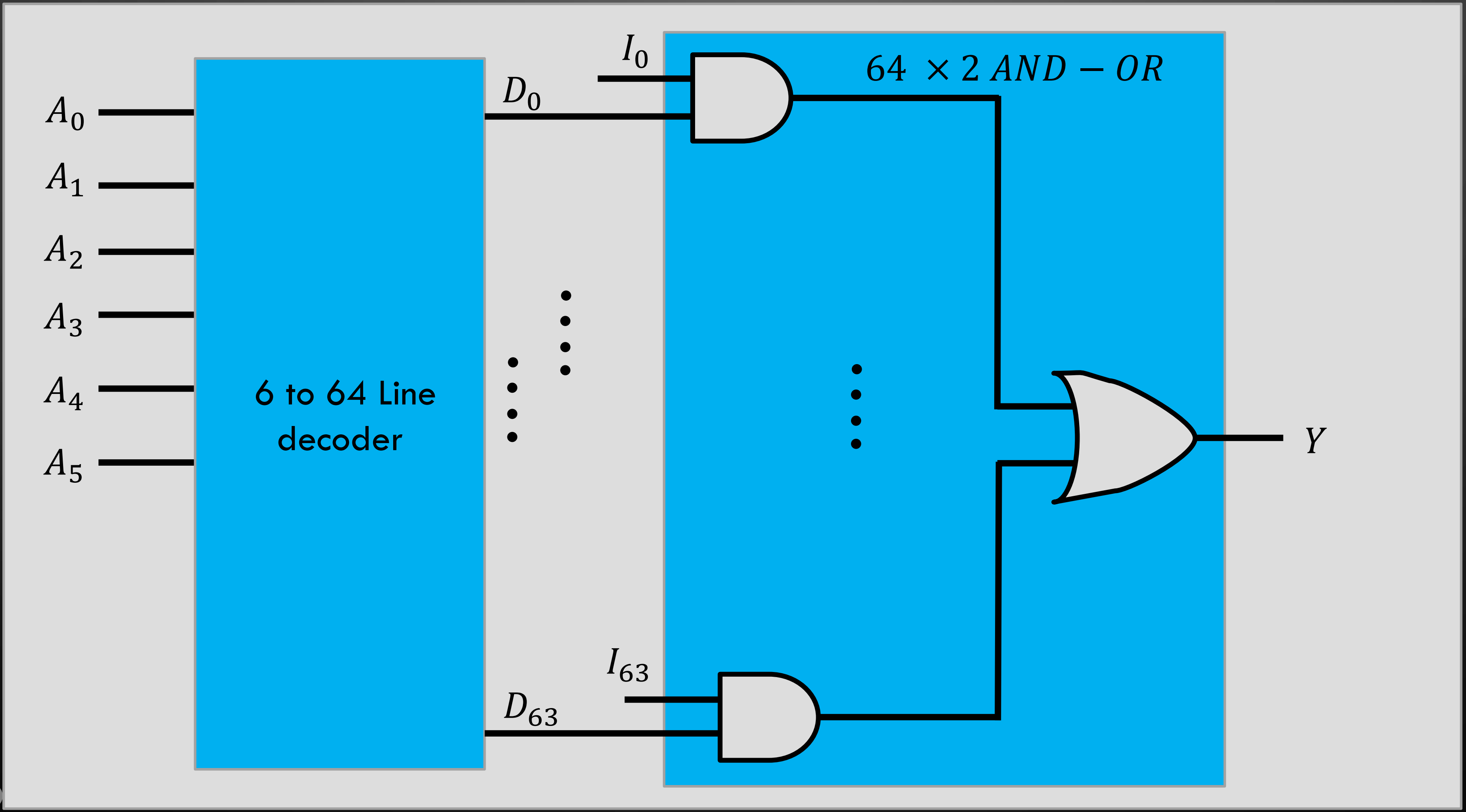


$$Y = \bar{S}_1 \bar{S}_0 I_0 + \bar{S}_1 S_0 I_1 + S_1 \bar{S}_0 I_2 + S_1 S_0 I_3$$

64 TO 1 LINE MULTIPLEXER (MUX)

- Information input m=64
- Selection input n ,
- Relationship between selection input n, and information input m; is $2^n \geq m$
- For this example $2^n = 64, \quad 2^n = 2^6, \; n = 6$
- Using Decoder circuit
  - The circuit requires 6 to 64 bit decoder.
  - 64 × 2 AND –OR gate.
  - Gate input cost (182+128+64=374), Two 3 to 8 decoder (27*2 +128 =182)
- Using 7 input AND gate ,
  - Gate input cost (6+448+64=518)

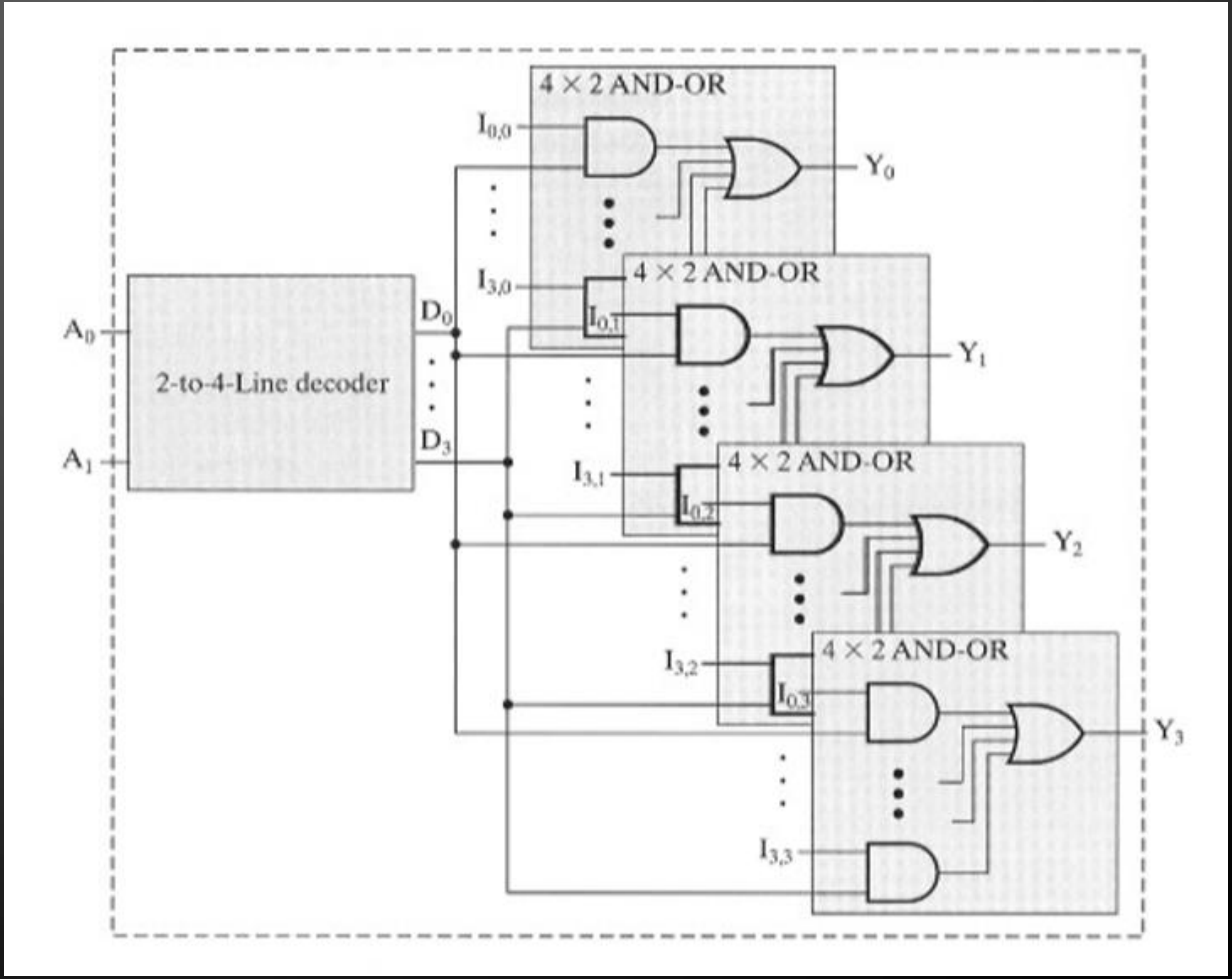
64 TO 1 LINE MUX



4 TO 1 LINE QUAD MUX

Specification: It has 2 selection input and each information input is replaced by a vector of four inputs. Since the information inputs are a vector, the output Y also becomes a four element vector.

Total gate cost (10+32+16=58)



MULTIPLEXER IMPLEMENTATION OF A BINARY ADDER BIT

The truth table can be generated by using value fixing on the information inputs of the multiplexer.

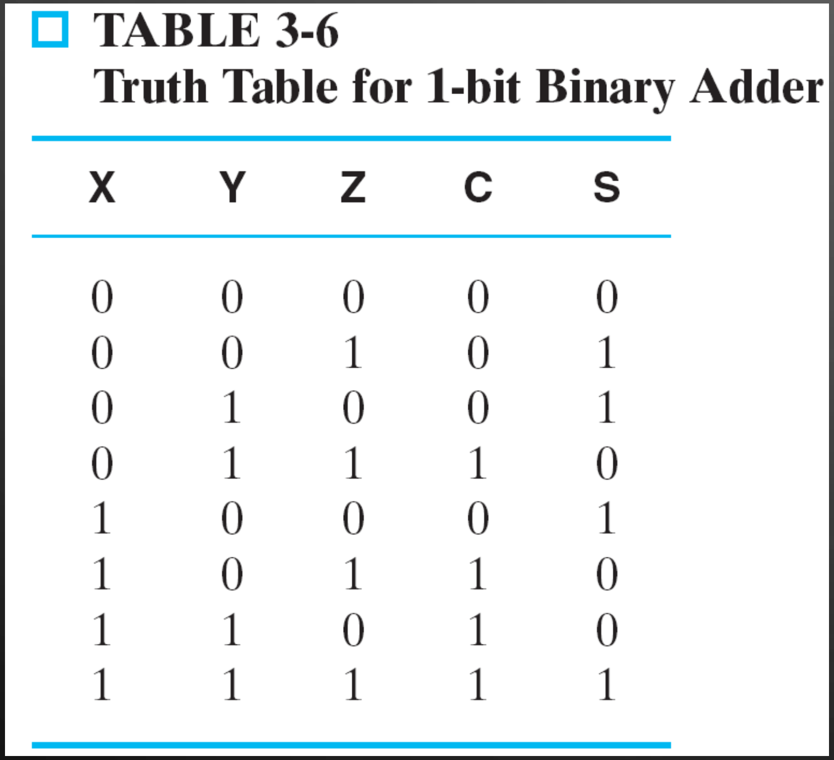
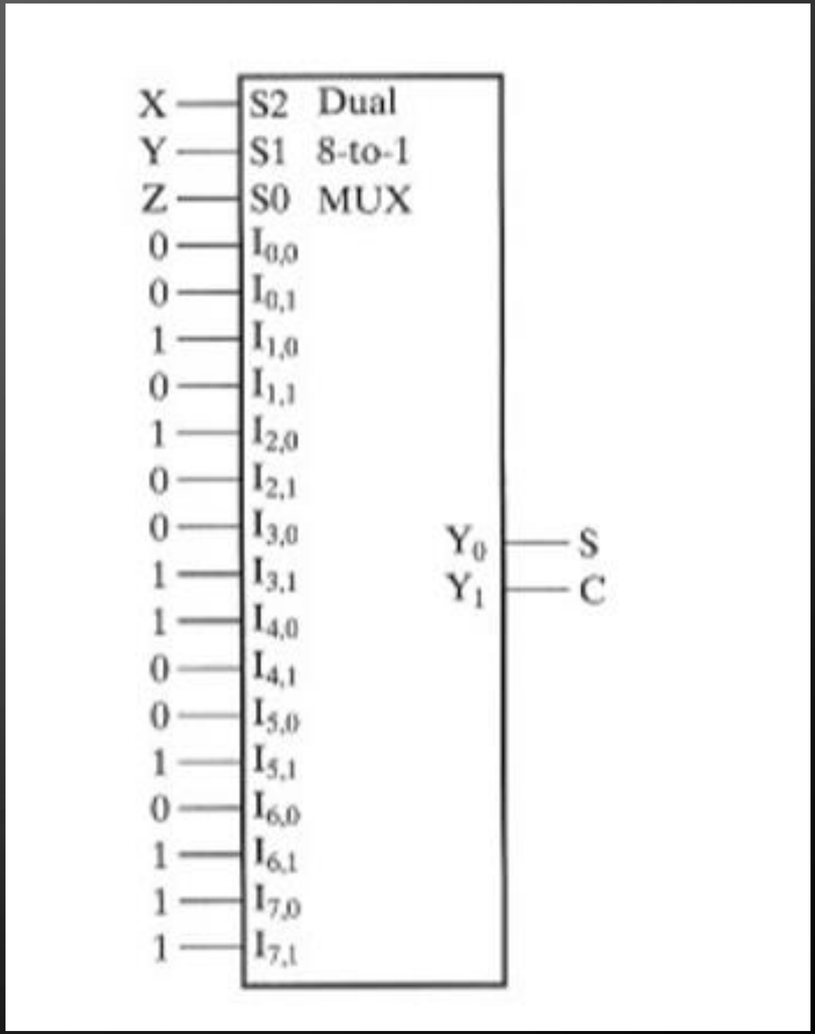
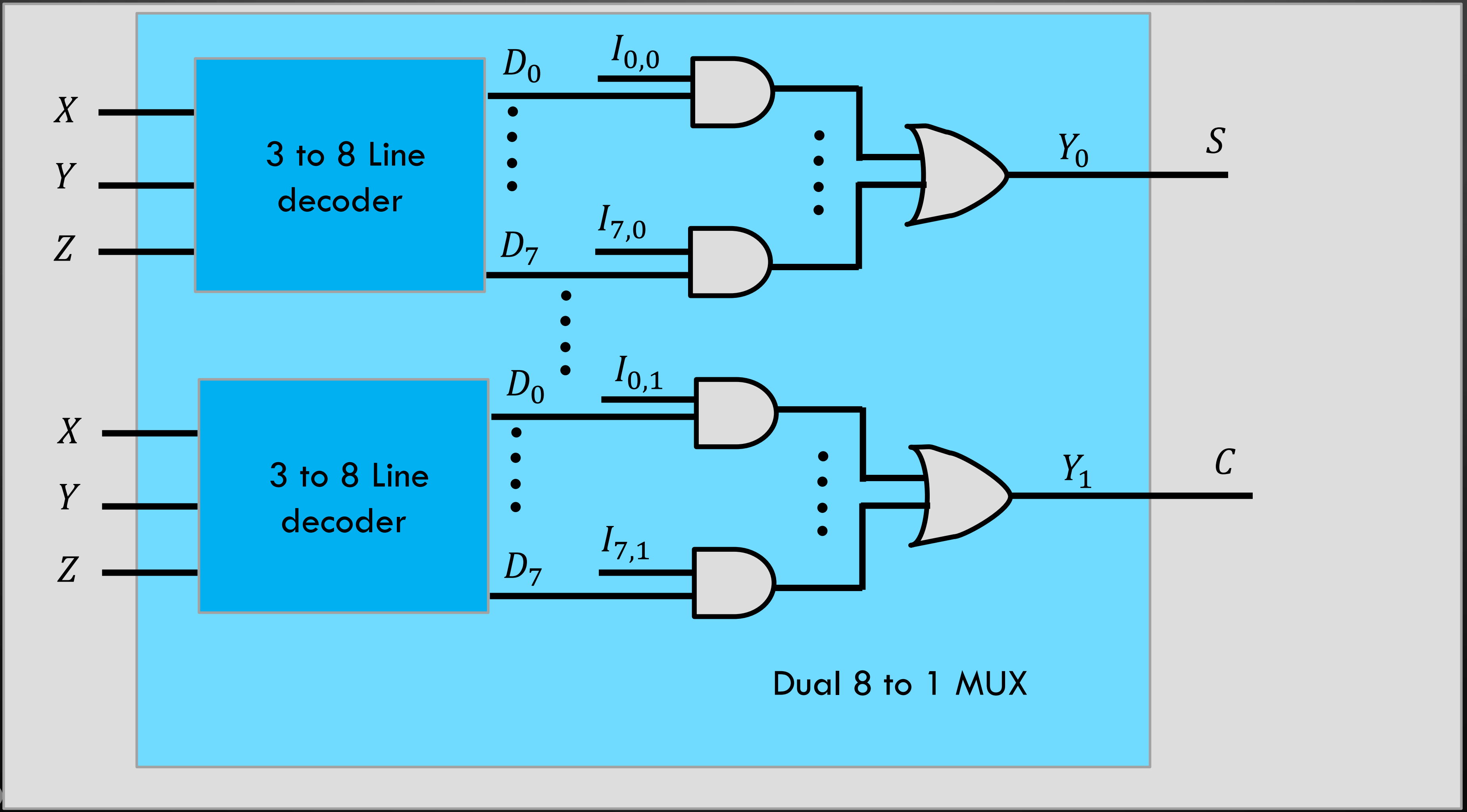


TABLE 3-6
Truth Table for 1-bit Binary Adder

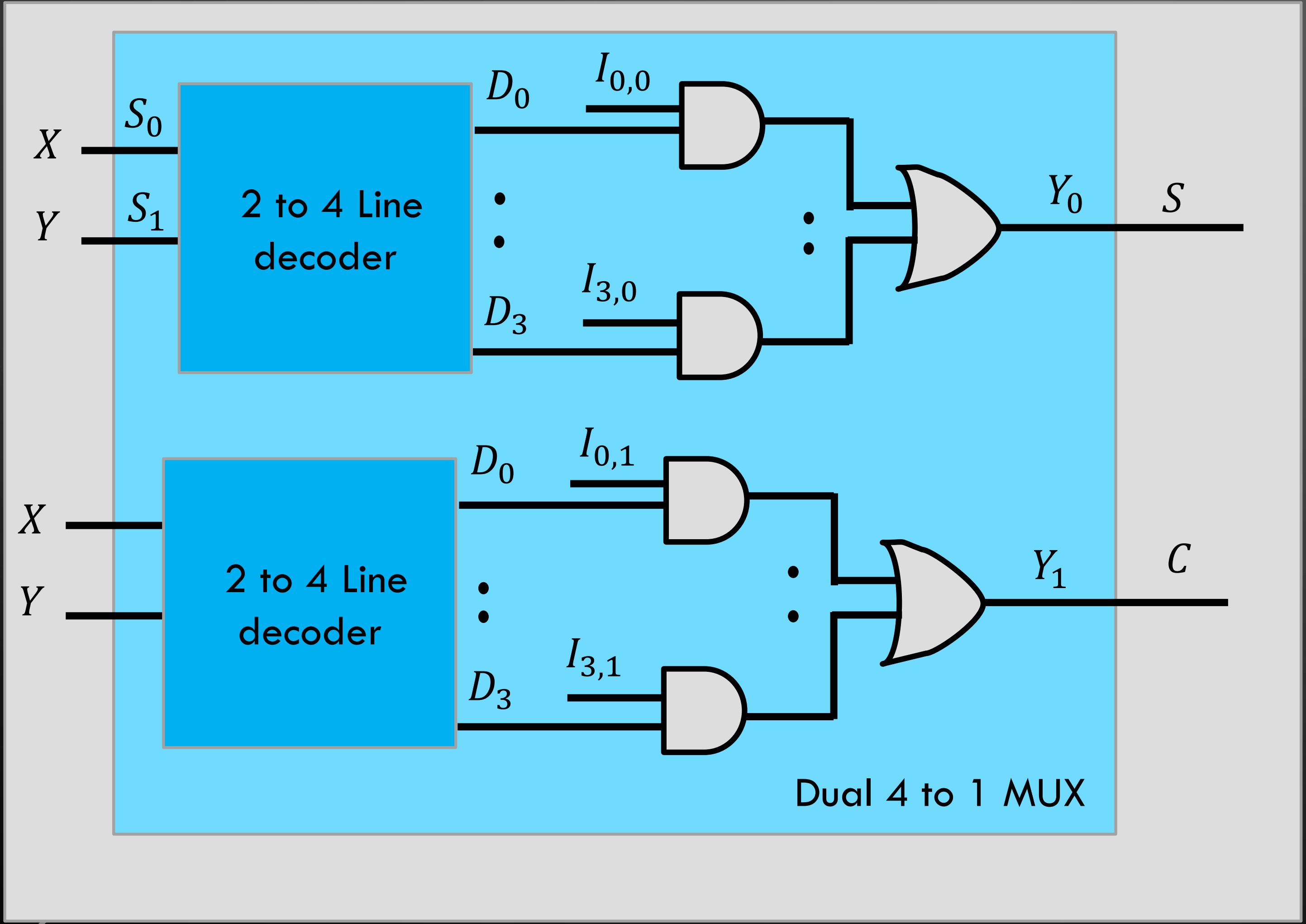
X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

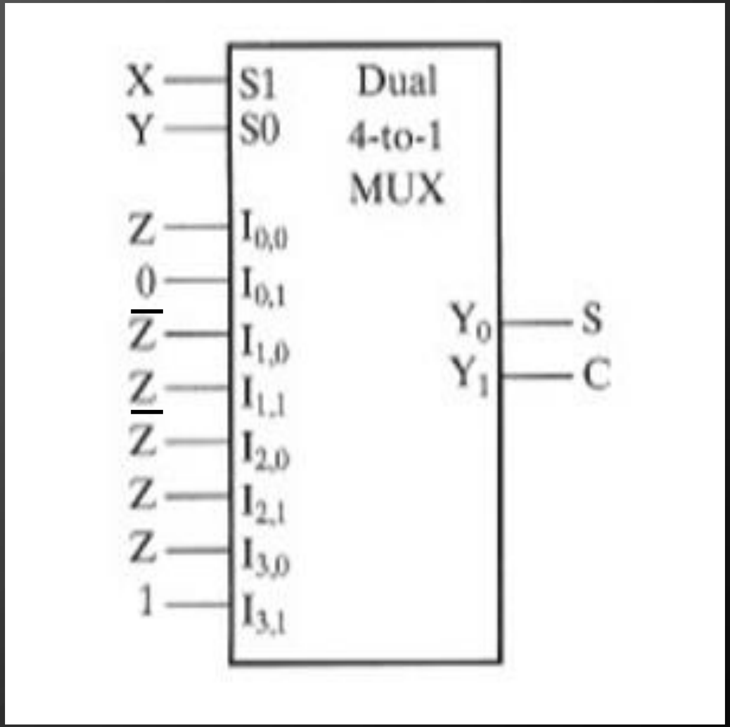


MULTIPLEXER IMPLEMENTATION OF A BINARY ADDER BIT



ALTERNATIVE MULTIPLEXER IMPLEMENTATION





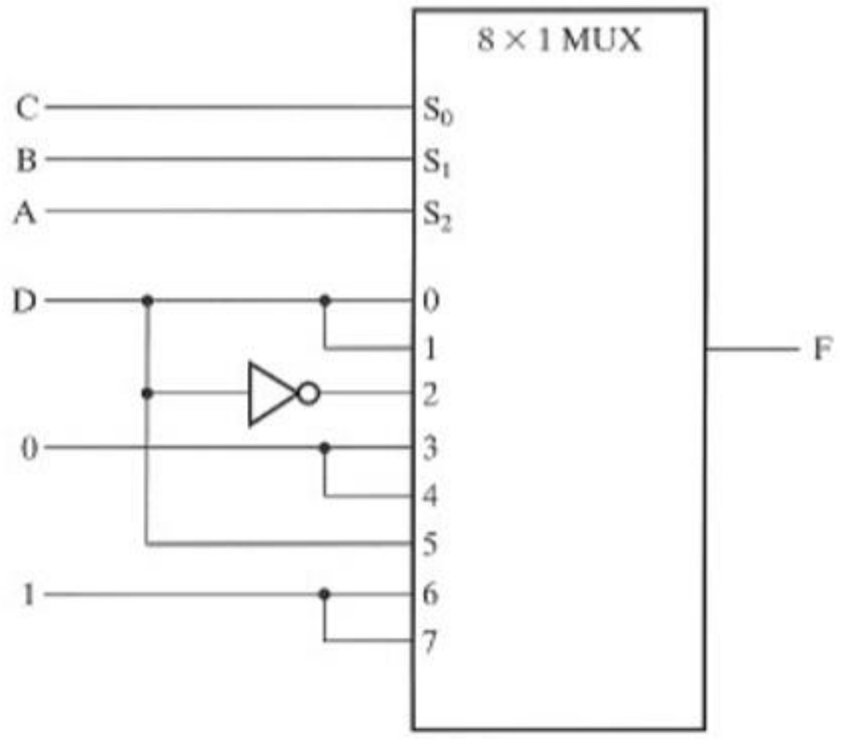
IMPLEMENTING BOOLEAN FUNCTION WITH A MUX

- For n variable Boolean function
  - Multiplexer selection input is n-1 , data input 2^{n-1}
- Boolean function is first listed in a truth table.
- The first n-1 variables in the table are applied to the selection input of the multiplexer.
- For each combination of the selection variables ,evaluate the output as a function of the last variable.
- This function can be 0,1 the variable or the complement of the variable.
- This values are then applied to the appropriate data inputs.

EXAMPLE OF IMPLEMENTING MUX

$$F(A,B,C,D) = \Sigma m(1,3,4,11,12,13,14,15)$$

	A	B	C	D	F	
0	0	0	0	0	0	F = D
	0	0	0	1	1	
1	0	0	1	0	0	F = D
	0	0	1	1	1	
2	0	1	0	0	1	$F = \overline{D}$
	0	1	0	1	0	
3	0	1	1	0	0	F = 0
	0	1	1	1	0	
4	1	0	0	0	0	F = 0
	1	0	0	1	0	
5	1	0	1	0	0	F = D
	1	0	1	1	1	
6	1	1	0	0	1	F = 1
	1	1	0	1	1	
7	1	1	1	0	1	F = 1
	1	1	1	1	1	



EXAMPLE: GRAY CODE TO BINARY

- Design a circuit to convert a 3-bit Gray code to a binary code
- The formulation gives the truth table on the right
- It is obvious from this table that X = C and the Y and Z are more complex

Gray	**Binary**
A B C	**x y z**
0 0 0	0 0 0
1 0 0	0 0 1
1 1 0	0 1 0
0 1 0	0 1 1
0 1 1	1 0 0
1 1 1	1 0 1
1 0 1	1 1 0
0 0 1	1 1 1

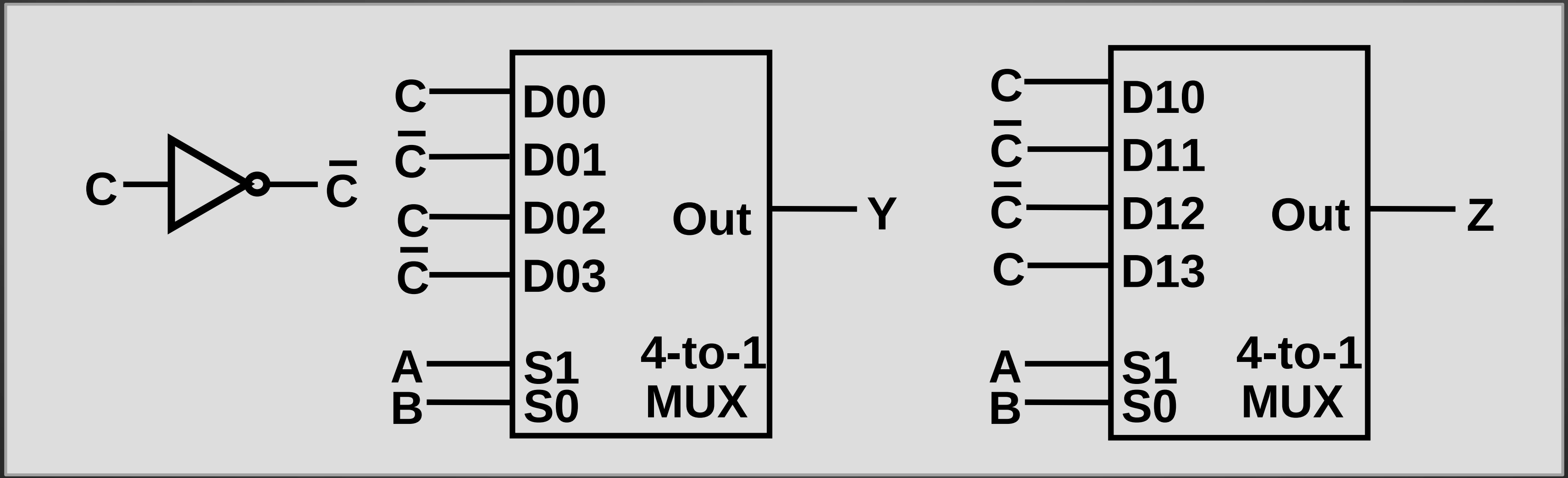
GRAY CODE TO BINARY (CONTINUED)

Rearrange the table so that the input combinations are in counting order, pair rows, and find rudimentary functions

Gray A B C	Binary x y z	Rudimentary Functions of C for y	Rudimentary Functions of C for z
0 0 0	0 0 0	$F = C$	$F = C$
0 0 1	1 1 1		
0 1 0	0 1 1	$F = \overline{C}$	$F = \overline{C}$
0 1 1	1 0 0		
1 0 0	0 0 1	$F = C$	$F = \overline{C}$
1 0 1	1 1 0		
1 1 0	0 1 0	$F = \overline{C}$	$F = C$
1 1 1	1 0 1		

GRAY CODE TO BINARY (CONTINUED)

- Assign the variables and functions to the multiplexer inputs:



- Note that this approach (Approach 2) reduces the cost by almost half compared to Approach 1.
- This result is no longer ROM-like
- Extending, a function of more than *n* variables is decomposed into several sub-functions defined on a subset of the variables. The multiplexer then selects among these sub-functions.

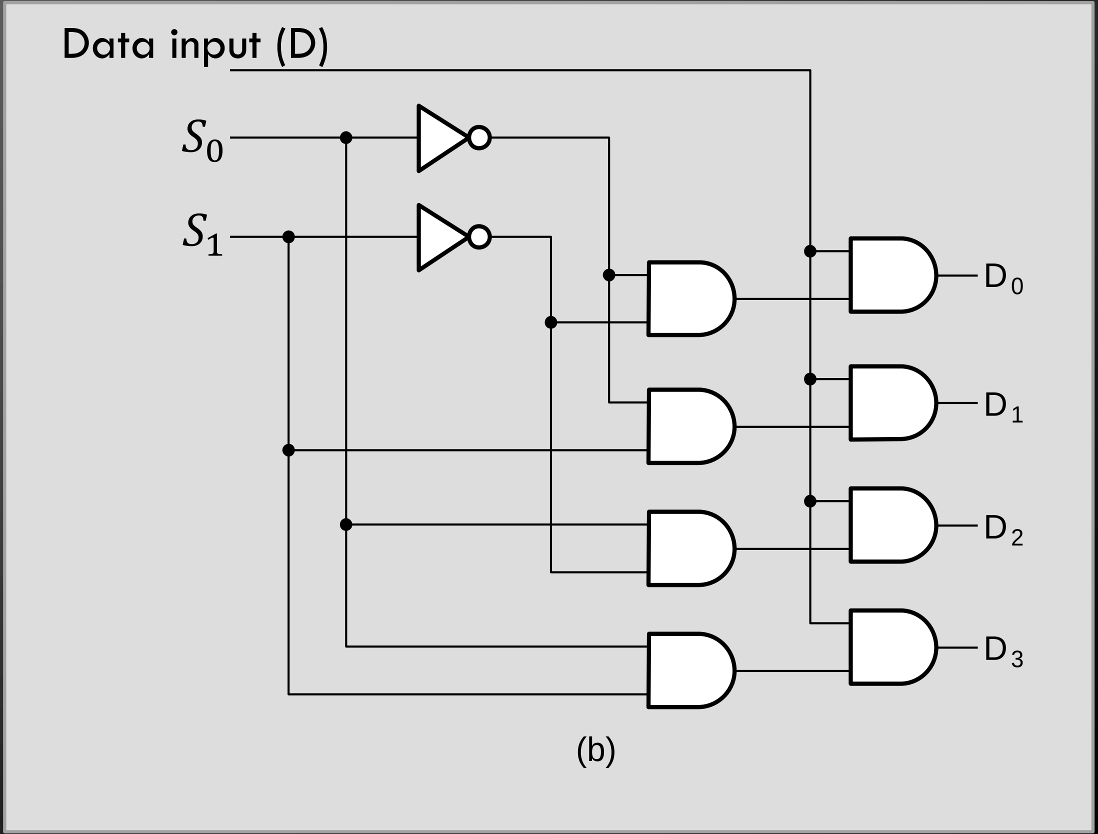
DEMULTIPLEXER (DEMUX)

- The inverse of selection is distribution, in which information received from a single line is transmitted to one of 2^n possible output lines.
- The circuit which implements such distribution is called a demultiplexer (DeMUX)

1:4 DEMULTIPLEXER

Truth Table

S_0	S_1	D_0	D_1	D_2	D_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


Data input (D)
S_0
S_1
D_0
D_1
D_2
D_3
(b)