ELECTRICAL
Engineering KMITNB

Combinational Circuit Building Blocks

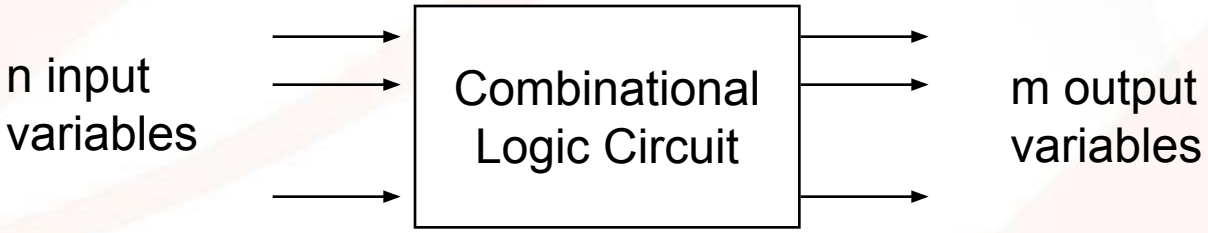
Chapter 6

Combinational Circuits

- Logic circuits for digital system
  - **Combinational circuits**
    - the outputs are a function of the current inputs
  - **Sequential circuits**
    - contain memory elements (Flip-Flops)
    - the outputs are a function of the current inputs and the state of the memory elements
    - the outputs also depend on past inputs

Combinational Logic Circuit

- A combinational circuits
  - 2^n possible combinations of input values

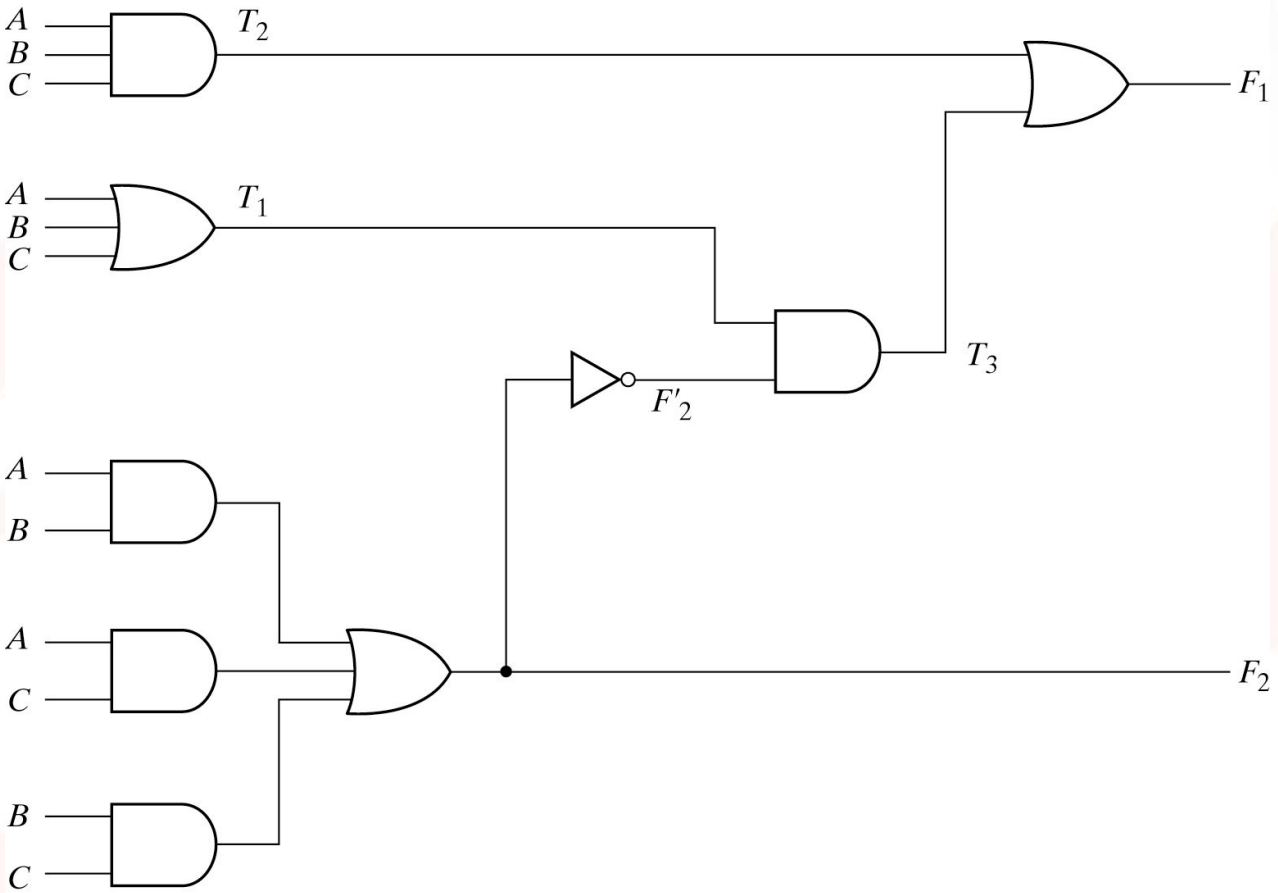


  - Specific functions
    - Adders, subtractors, comparators, decoders, encoders, and multiplexers
    - Medium-scale integration (MSI) circuits or standard cells

Analysis Procedure

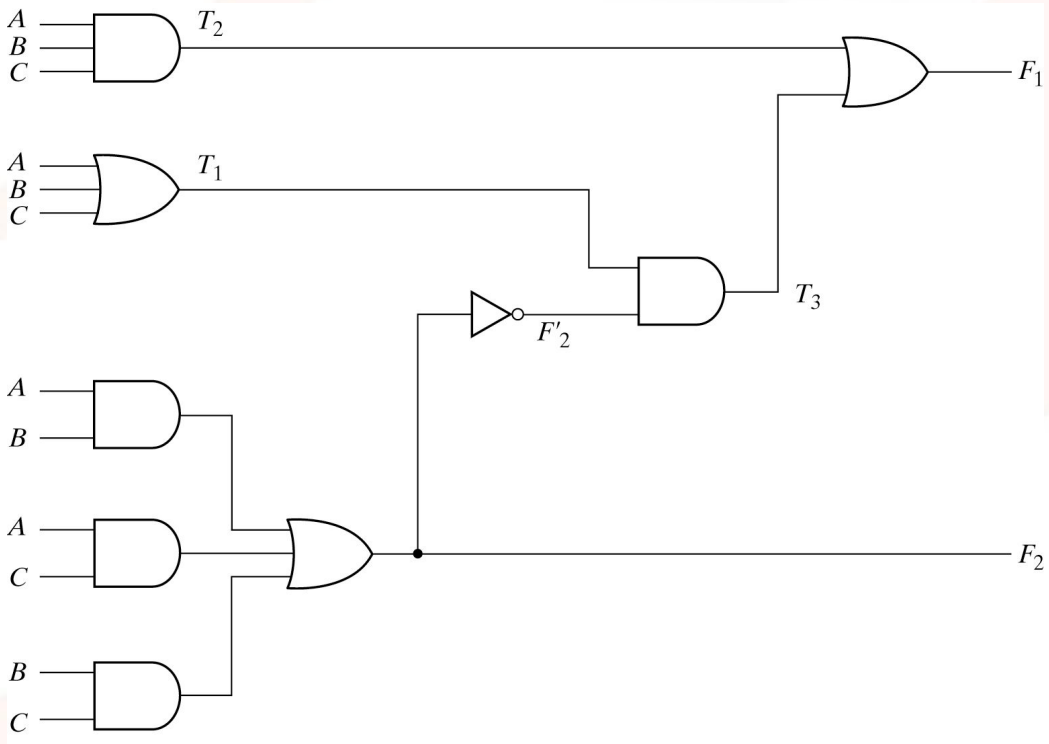
- A combinational circuit
  - make sure that it is combinational not sequential
    - No feedback path
  - derive its Boolean functions (truth table)
  - design verification
  - a verbal explanation of its function

Analysis Procedure (cont.)



Analysis Procedure (cont.)

- A straightforward procedure
  - $F_2 = AB+AC+BC$
  - $T_1 = A+B+C$
  - $T_2 = ABC$
  - $T_3 = F_2'T_1$
  - $F_1 = T_3+T_2$



Analysis Procedure (cont.)

- $F_1 = T_3 + T_2 = F_2'T_1 + ABC$
  $= (AB+AC+BC)'(A+B+C)+ABC$
  $= (A'+B')(A'+C')(B'+C')(A+B+C)+ABC$
  $= (A'+B'C')(AB'+AC'+BC'+B'C)+ABC$
  $= A'BC'+A'B'C+AB'C'+ABC$

Analysis Procedure (end)

- The truth table
  - F_1 = A'BC'+A'B'C+AB'C'+ABC
  - F_2 = AB+AC+BC

A	B	C	F_2	F_2‘	T_1	T_2	T_3	F_1
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

Design Procedure

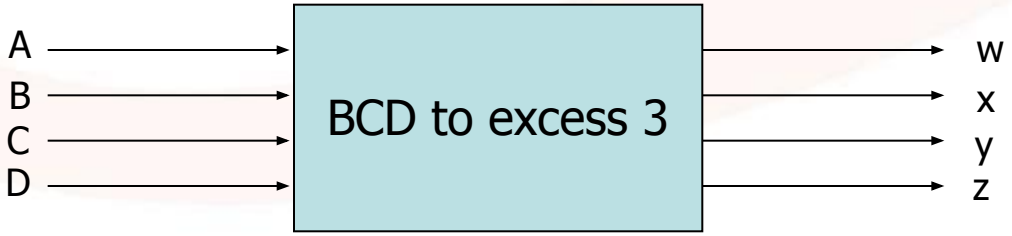
- The design procedure of combinational circuits
  - State the problem (system spec.)
  - determine the inputs and outputs
  - the input and output variables are assigned symbols
  - derive the truth table
  - derive the simplified Boolean functions
  - draw the logic diagram and verify the correctness

Combinational Circuit Design

- Functional description
  - Boolean function
  - HDL (Hardware description language)
    - Verilog HDL
    - VHDL
  - Schematic entry
- Logic minimization
  - number of gates
  - number of inputs to a gate (Fan-in)
  - propagation delay
  - number of interconnection
  - limitations of the driving capabilities (Fan-out)

Design Example: Code conversion

- BCD to excess-3 code
  - System Specification
    - excess-3 code is BCD + 3
  - Inputs and Outputs
    - Inputs = BCD (4 digits)
    - Outputs = excess-3 code (4 digits)
  - Symbols for input and output variables
    - Input variables = ABCD
    - Output variables =wxyz



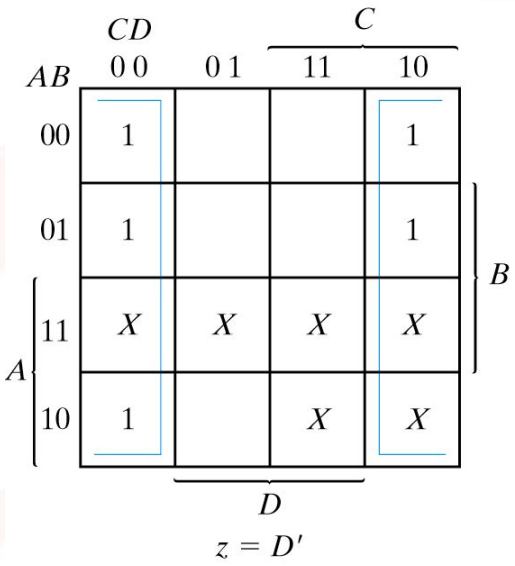
Code conversion example

- The truth table

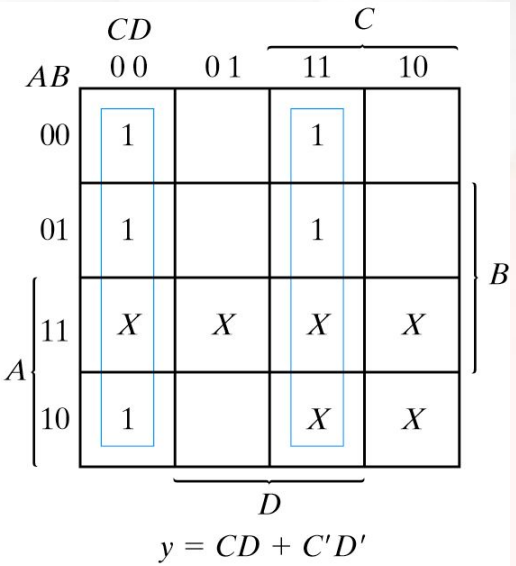
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

ELECTRICAL
Engineering KMITNB

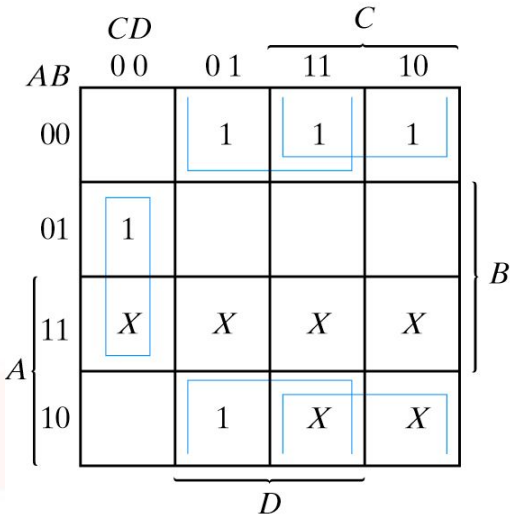
- The maps



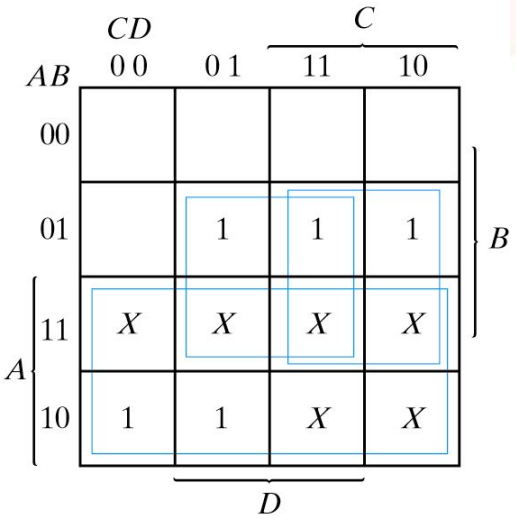
$z = D'$



$y = CD + C'D'$



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

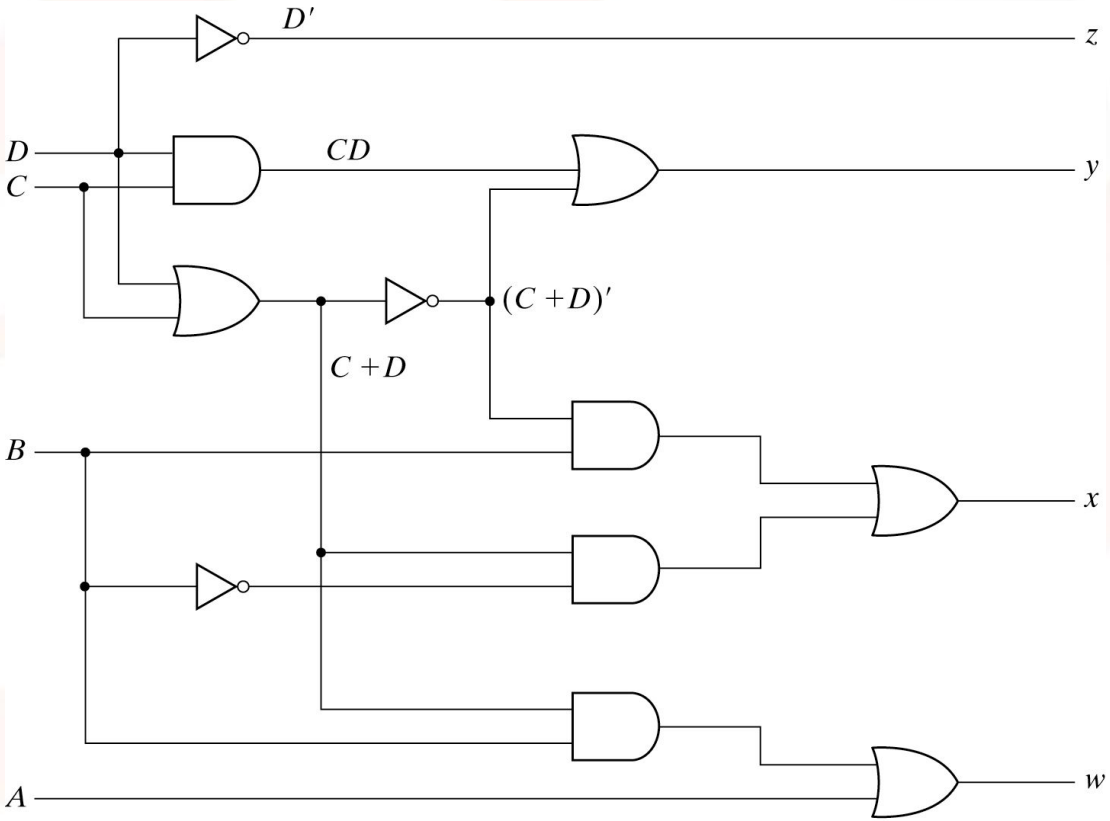


$w = A + BC + BD$

- The simplified functions
  - z = D'
  - y = CD +C'D'
  - x = B'C + B'D+BC'D'
  - w = A+BC+BD
- Another implementation
  - z = D'
  - y = CD +C'D' = CD + (C+D)'
  - x = B'C + B'D+BC'D' = B'(C+D) +B(C+D)'
  - w = A+BC+BD

ELECTRICAL
Engineering KMITNB

- The logic diagram

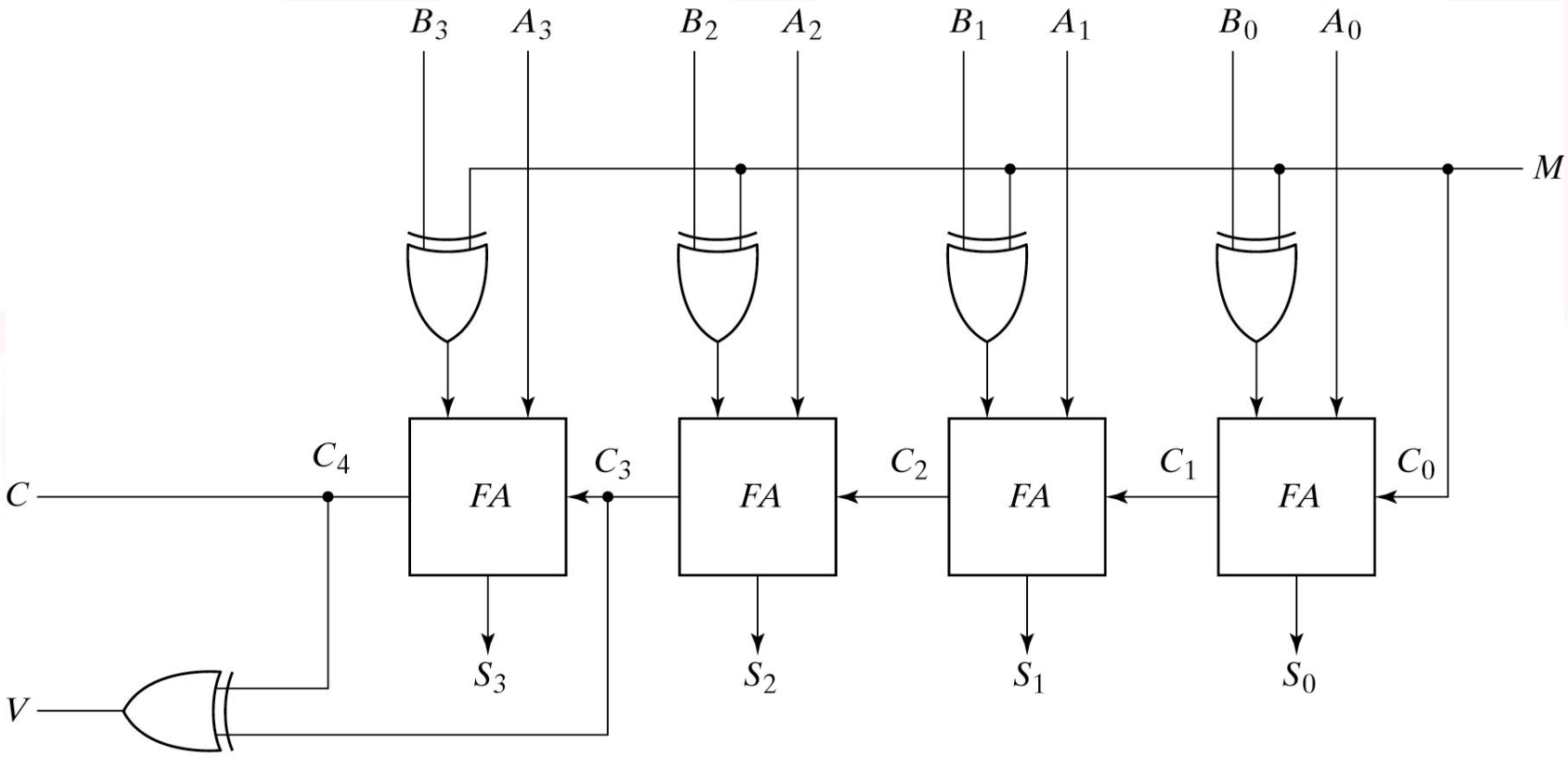


Arithmetic combinational circuits

- Adder/Subtractor
  - Full Adder
  - Half Adder
  - Carry Look Ahead (CLA) Adder
- Multiplier
- Divider

Binary Subtractor (Recall)

- A-B = A+(2's complement of B)
- 4-bit Adder-subtractor



Conditions of Overflow (Recall)

- Overflow
  - The storage is limited
  - Add two positive numbers and obtain a negative number
  - Add two negative numbers and obtain a positive number
  - V=0, no overflow; V=1, overflow

Decimal Adder (BCD Adder)

- Add two BCD's
  - **9 inputs**: two BCD's and one carry-in
  - **5 outputs**: one BCD and one carry-out
- Design approaches
  - A truth table with 2^9 entries
  - use binary full Adders
    - the sum <= 9+9+1 = 19
    - binary to BCD

Decimal Adder (BCD Adder)

- The truth table

Derivation of a BCD Adder

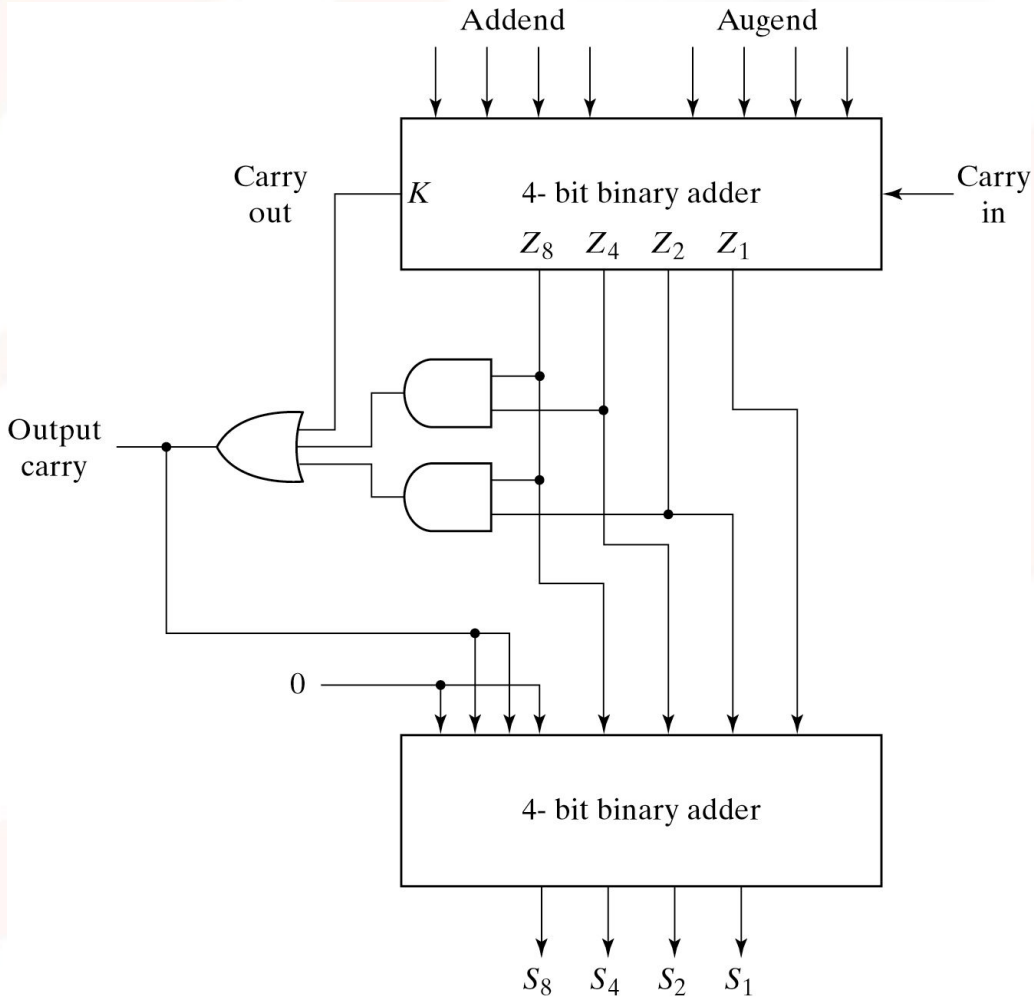
Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	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

Decimal Adder (BCD Adder)

- Modifications are needed if the sum > 9
  - C = 1
    - K = 1
    - $Z_8Z_4 = 1$
    - $Z_8Z_2 = 1$
  - modification: $-(10)_d$ or +6

Decimal Adder (BCD Adder)

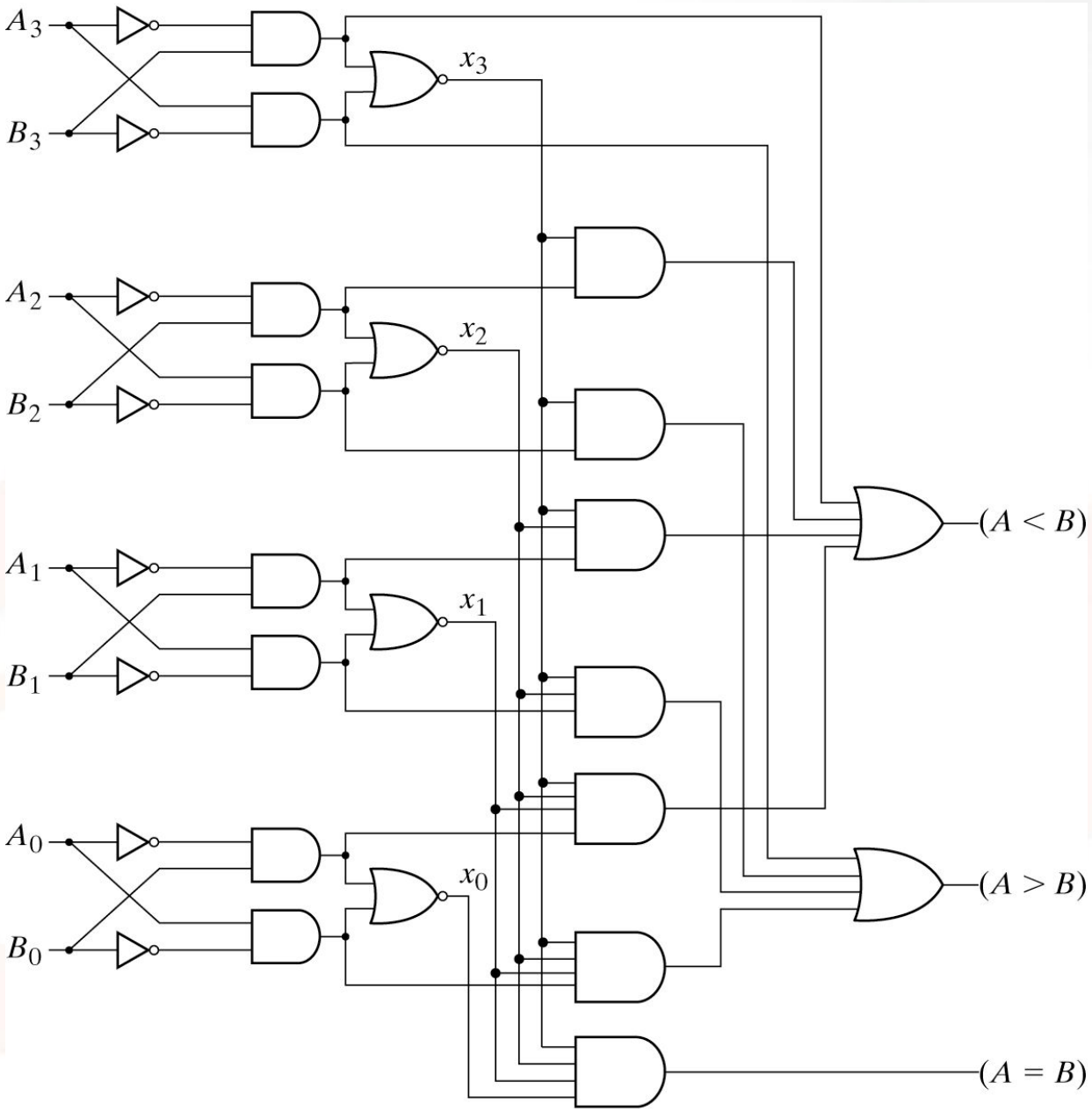
- Logic Circuit Diagram



Magnitude Comparator

- The comparison of two numbers
  - outputs: A>B, A=B, A<B
- Design Approaches
  - the truth table
    - 2^{2n} entries - too cumbersome for large n
  - use inherent regularity of the problem
    - reduce design efforts
    - reduce human errors

- Algorithm -> logic
  - $A = A_3A_2A_1A_0$; $B = B_3B_2B_1B_0$
  - A=B if $A_3=B_3$, $A_2=B_2$, $A_1=B_1$and $A_0=B_0$
    - equality: $x_i= A_iB_i+A_i'B_i'$
    - $(A=B) = x_3x_2x_1x_0$
  - $(A>B) = A_3B_3'+x_3A_2B_2'+x_3x_2A_1B_1'+x_3x_2x_1 A_0B_0'$
  - $(A<B) = A_3'B_3+x_3A_2'B_2+x_3x_2A_1'B_1+x_3x_2x_1 A_0'B_0$
- Implementation
  - $x_i = (A_iB_i'+A_i'B_i)'$


A_3
B_3
x_3
A_2
B_2
x_2
A_1
B_1
x_1
A_0
B_0
x_0
$(A < B)$
$(A > B)$
$(A = B)$

Decoder

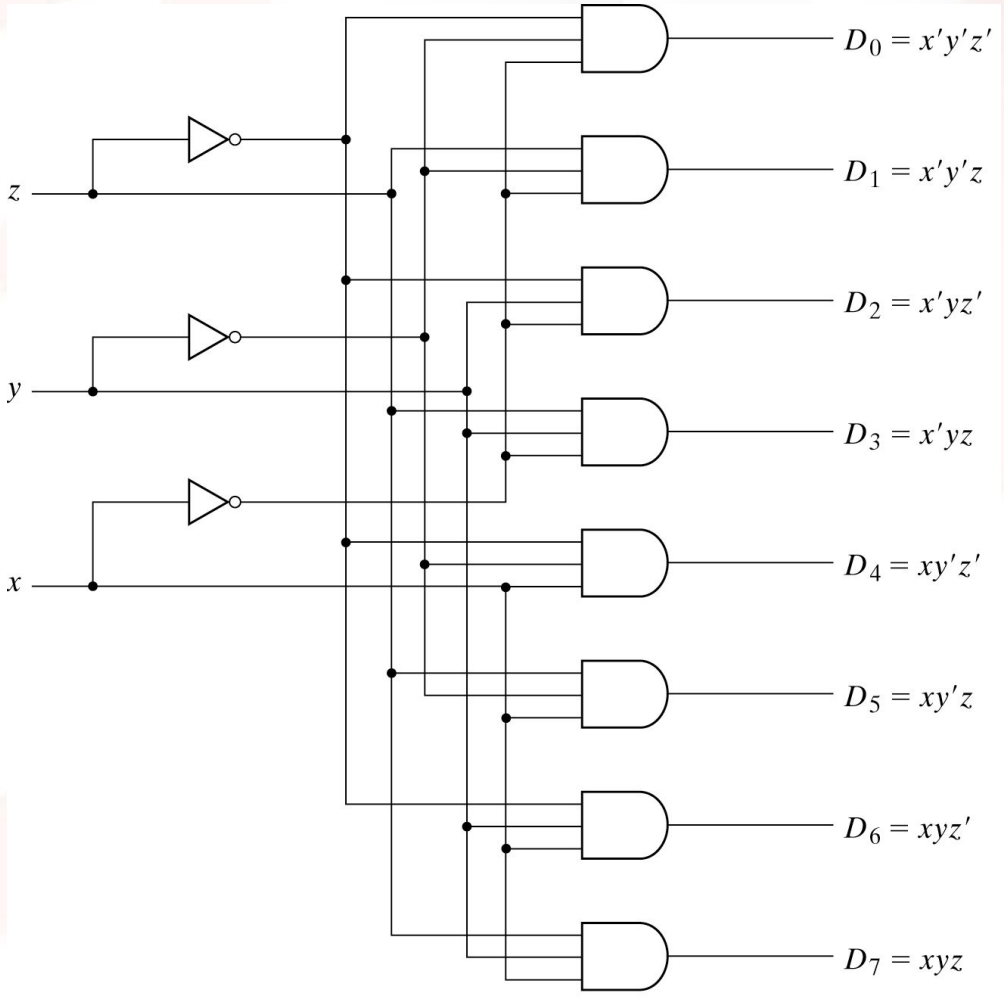
- A n-to-m decoder
  - a binary code of n bits = 2^n distinct information
  - n input variables; up to 2^n output lines
  - only one output is active at time

Truth Table of a 3-to-8-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

Decoder (cont.)

- An implementation of 3-to-8-line decoder

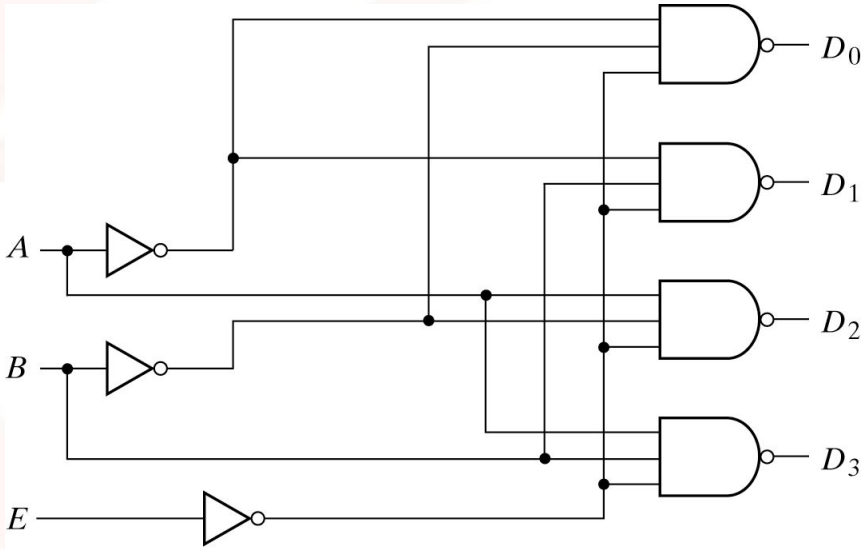


Decoder

- Combinational logic implementation
  - each output = a minterm
  - use a decoder and an external OR gate to implement any Boolean function of n input variables

Demultiplexers (DEMUX)

- Demultiplexers
  - a decoder with an enable input
  - receive information on a single line and transmits it on one of 2^n possible output lines



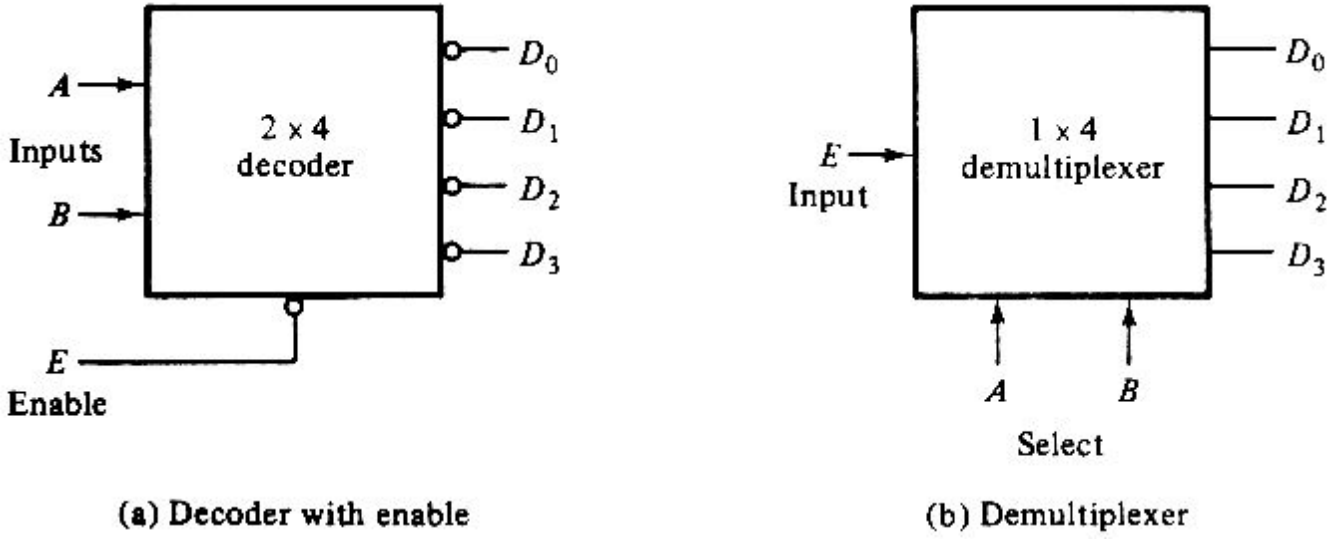
(a) Logic diagram

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

(b) Truth table

Demultiplexer (cont.)

- Decoder/demultiplexers

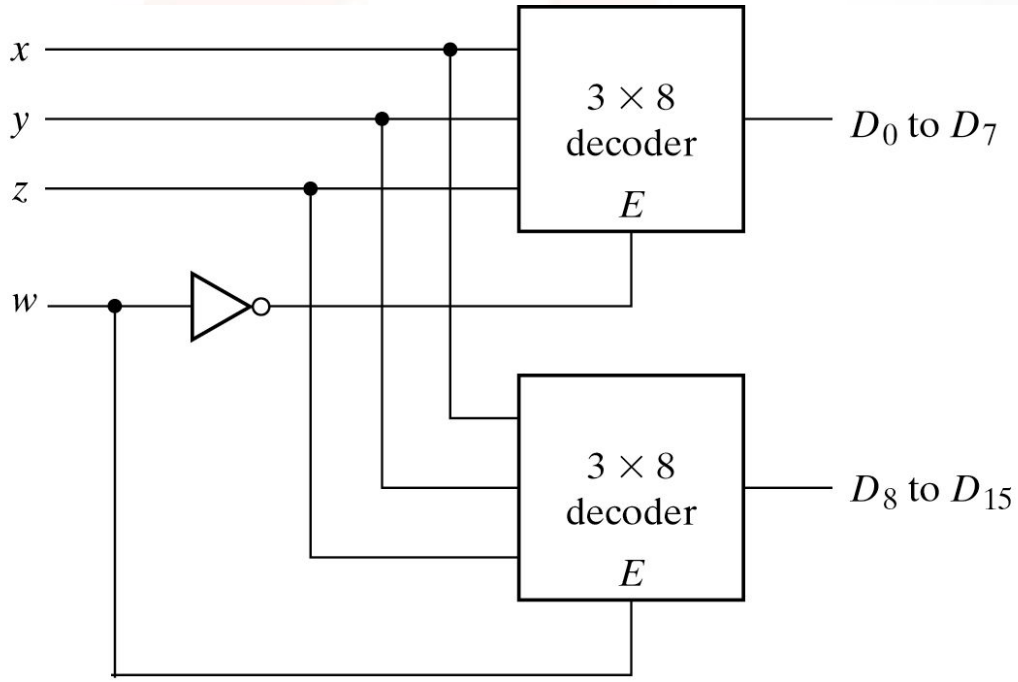


(a) Decoder with enable

(b) Demultiplexer

Demultiplexer (cont.)

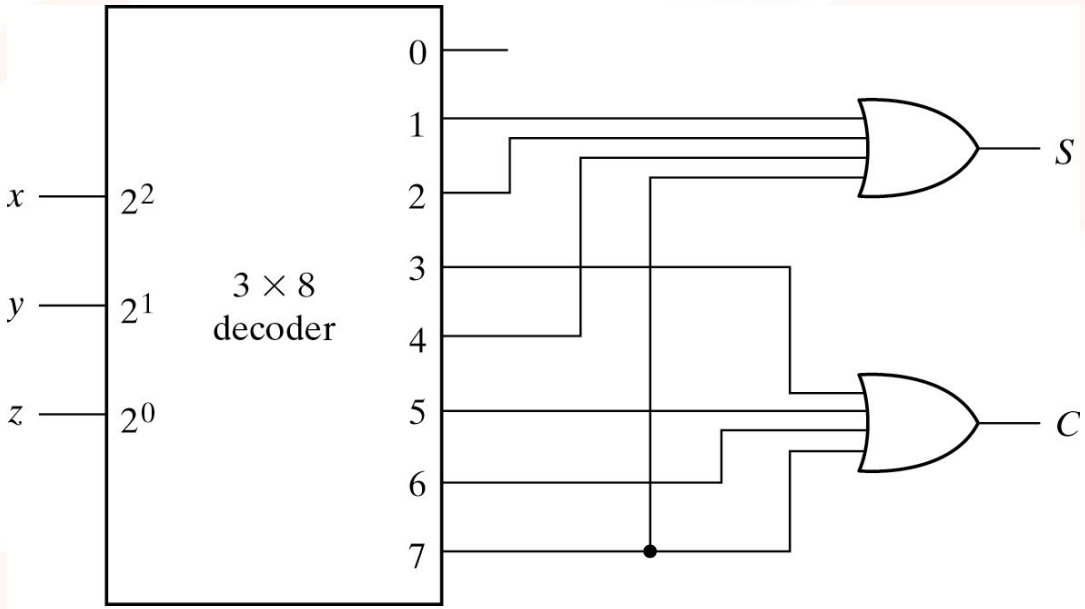
- Expansion
  - two 3-to-8 decoder: a 4-to-16 deocder



  - a 5-to-32 decoder?

Combinational Logic Implementation using a decoder

- each output = a minterm
- use a decoder and an external OR gate to implement any Boolean function of n input variables
- A full-adder
  - S(x,y,x)=Σ(1,2,4,7)
  - C(x,y,z)= Σ(3,5,6,7)



Combinational Logic Implementation using a decoder

- two possible approaches using decoder
  - OR(minterms of F): k inputs
  - NOR(minterms of F'): 2^n - k inputs
- In general, it is not a practical implementation

Encoder

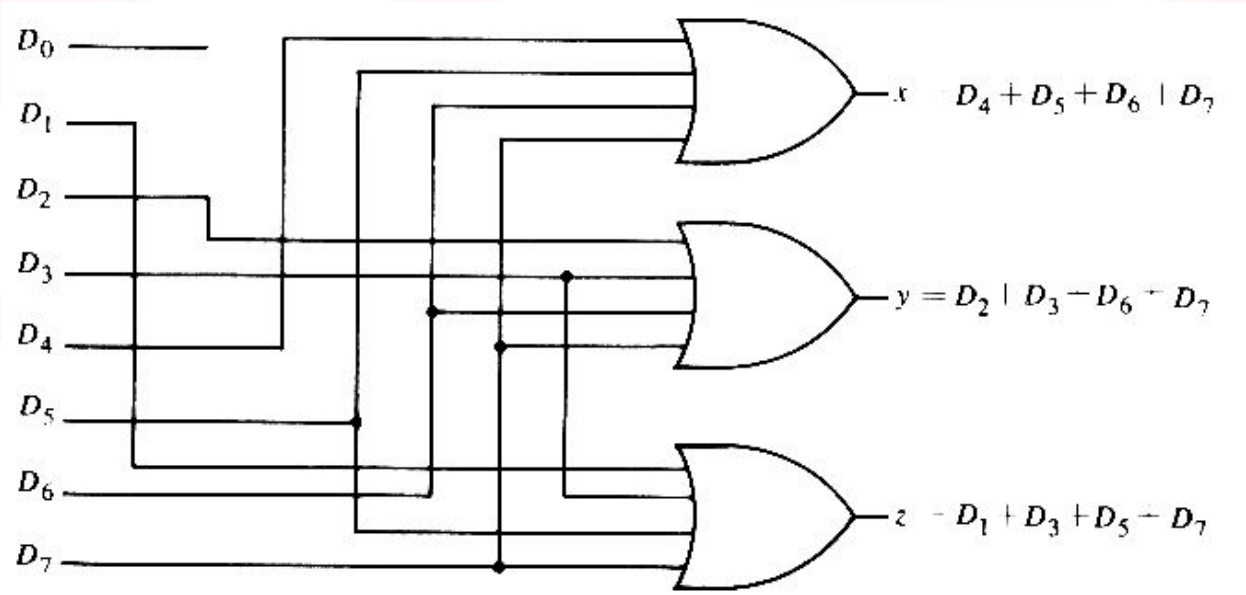
- The inverse function of a decoder

Truth Table of Octal-to-Binary 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

Encoder

- An implementation



  - limitations
    - illegal input: e.g. D_3=D_6=1
    - the output = 111 (^{1}3 and ^{1}6)

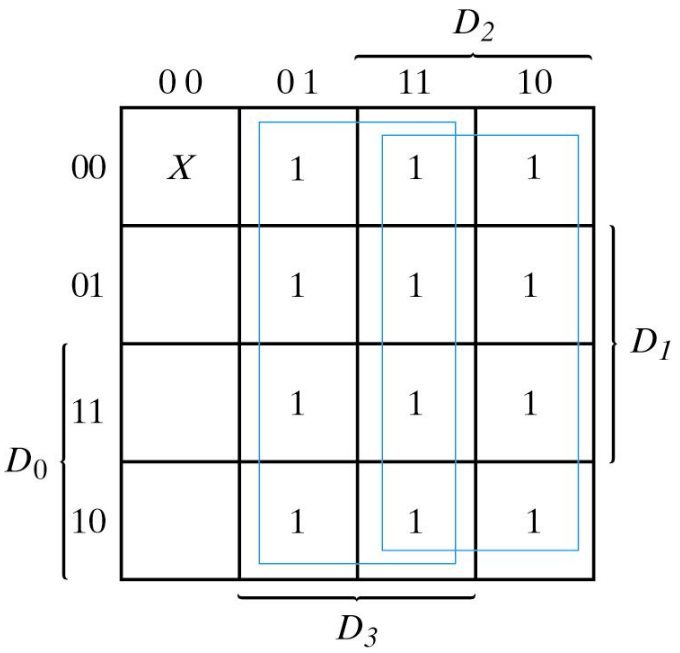
Priority Encoder

- resolve the ambiguity of illegal inputs
- only one of the input is encoded

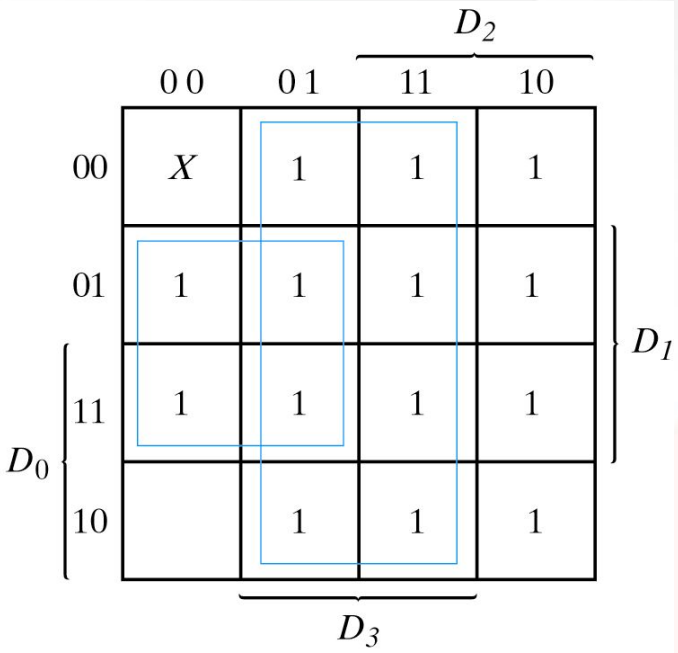
Truth Table of a Priority Encoder

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

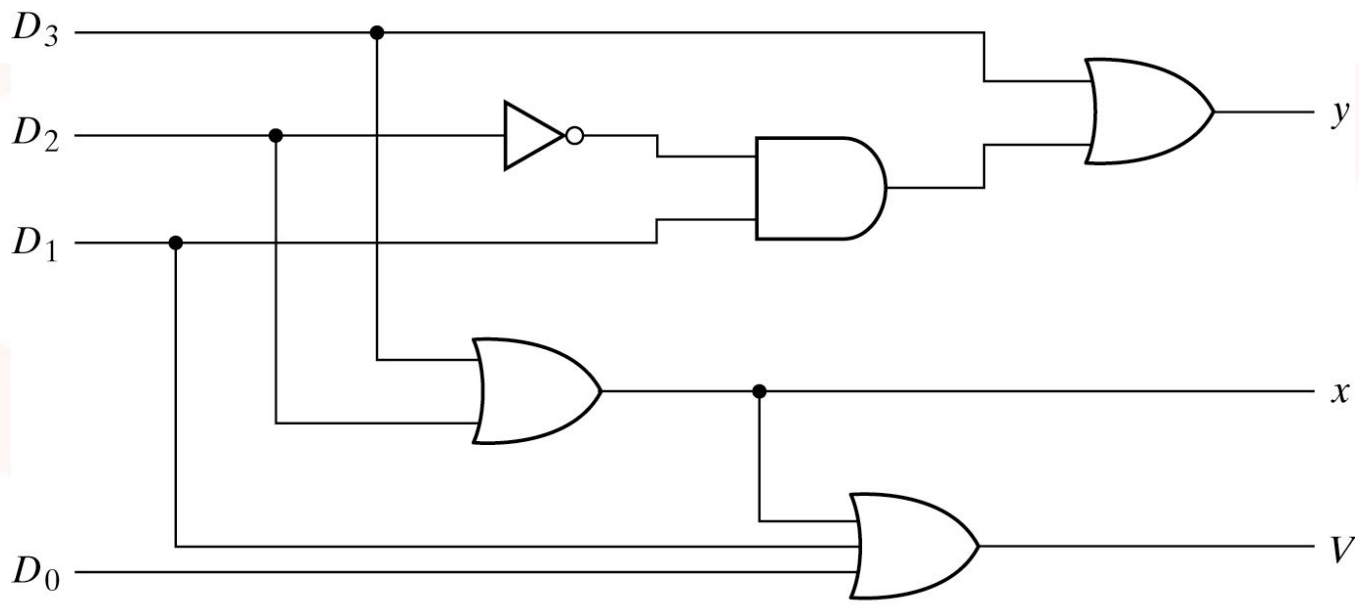
- D_3 has the highest priority
- D_0 has the lowest priority
- X: don't-care conditions
- V: valid output indicator



$x = D_2 + D_3$

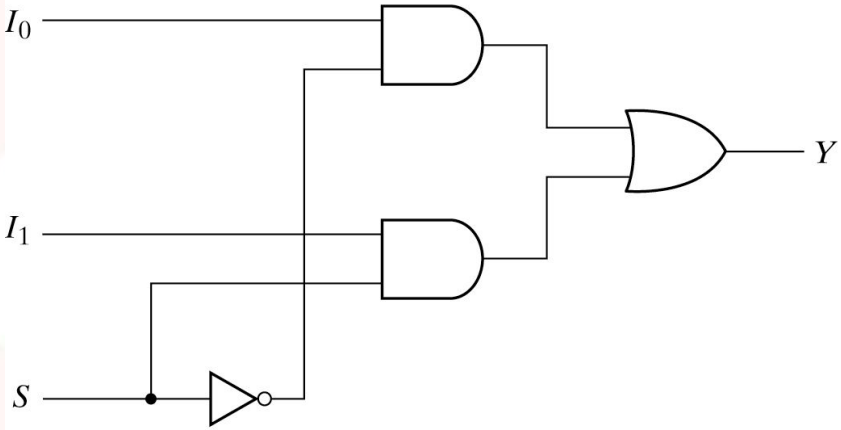


$y = D_3 + D_1D'_2$

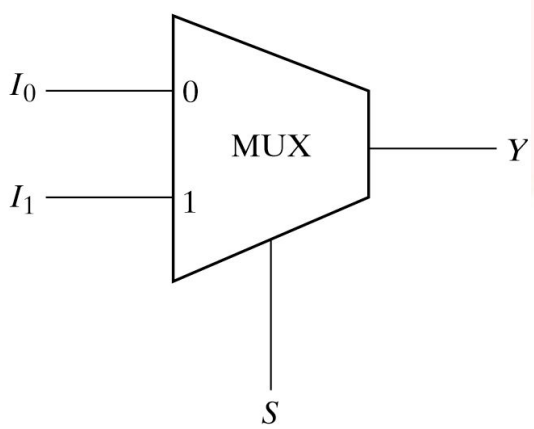


Multiplexer

- select binary information from one of many input lines and direct it to a single output line
- 2^n input lines, n selection lines and one output line
- e.g.: 2-to-1-line multiplexer



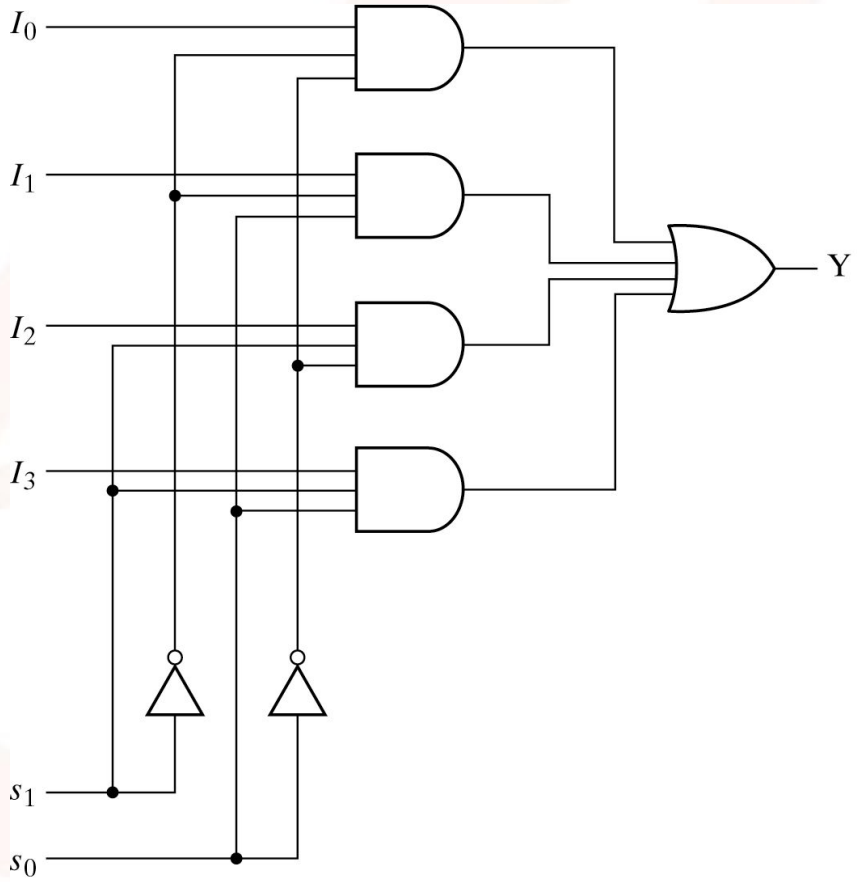
(a) Logic diagram



(b) Block diagram

Multiplexer

- 4-to-1-line multiplexer



(a) Logic diagram

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

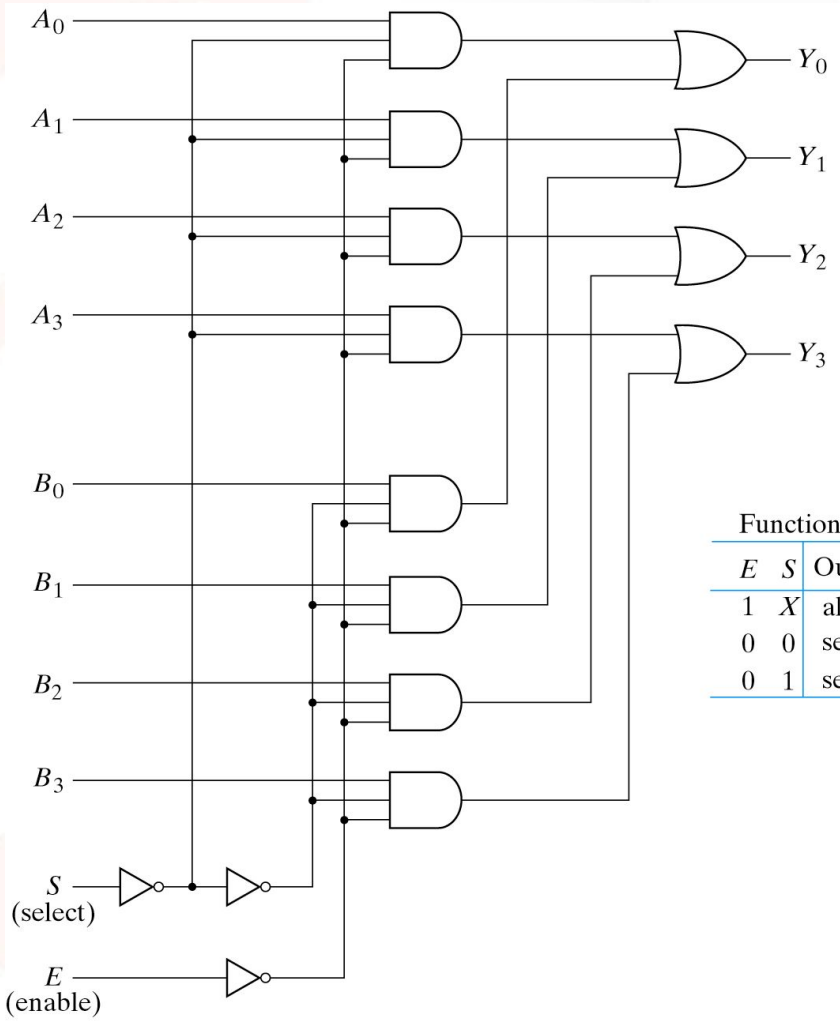
(b) Function table

Multiplexer

- Implementation note
  - n-to- 2^n decoder
  - add the 2^n input lines to each AND gate
  - OR(all AND gates)
  - an enable input (an option)

Multiplexer

Quadruple 2-to-1 MUX



Function table

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

Boolean function implementation using a multiplexer

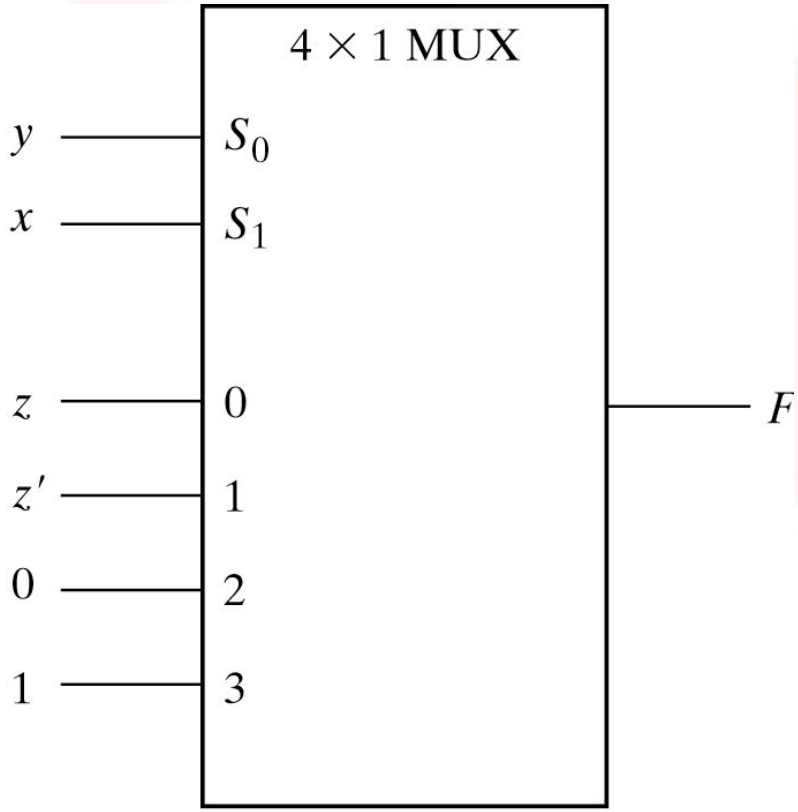
- MUX: a decoder + an OR gate
- 2^n-to-1 MUX can implement any Boolean function of n input variable
- a better solution: implement any Boolean function of n+1 input variable
  - n of these variables: the selection lines
  - the remaining variable: the inputs

Boolean function implementation using a multiplexer

- an example: F(A,B,C)=Σ(1,2,6,7)

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table



(b) Multiplexer implementation

Boolean function implementation using a multiplexer

- Procedure:
  - assign an ordering sequence of the input variable
  - the rightmost variable (D) will be used for the input lines
  - assign the remaining n-1 variables to the selection lines w.r.t. their corresponding sequence
  - construct the truth table
  - consider a pair of consecutive minterms starting from m_0
  - determine the input lines

Boolean function implementation using a multiplexer

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	

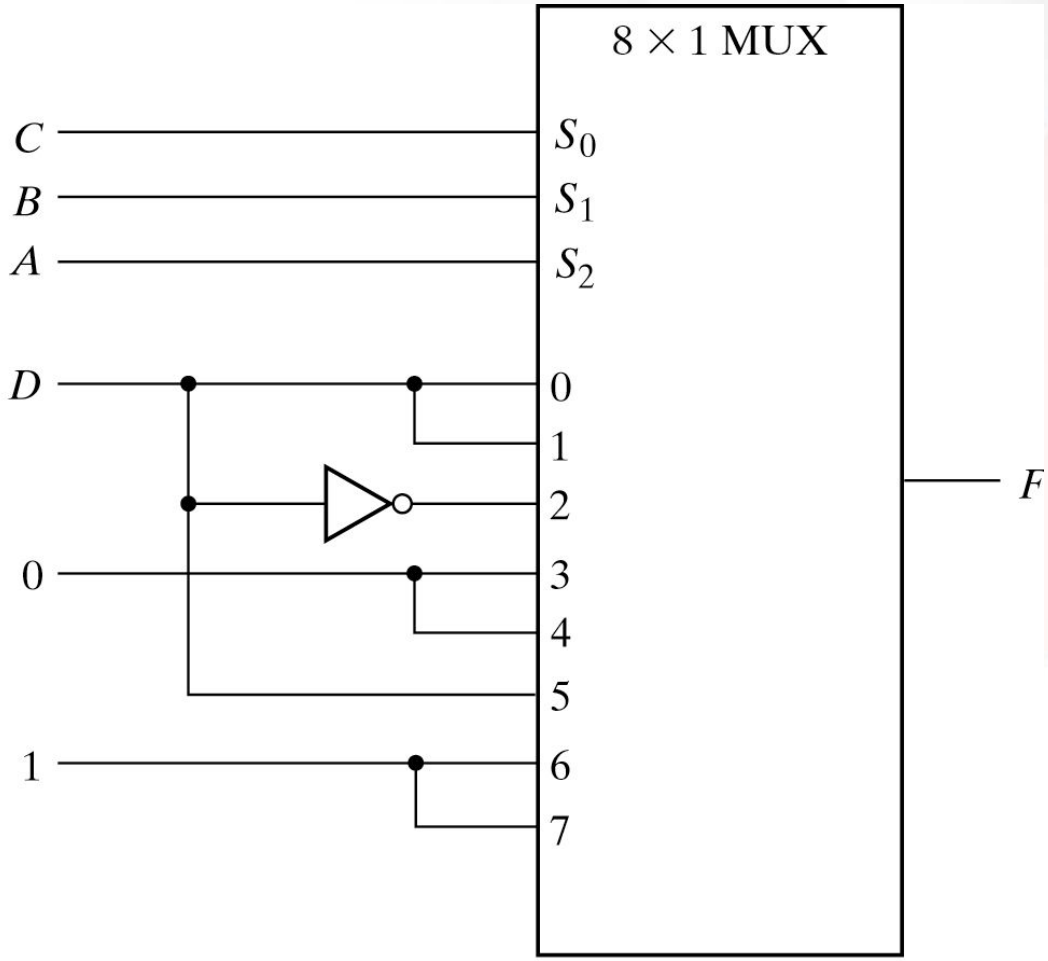
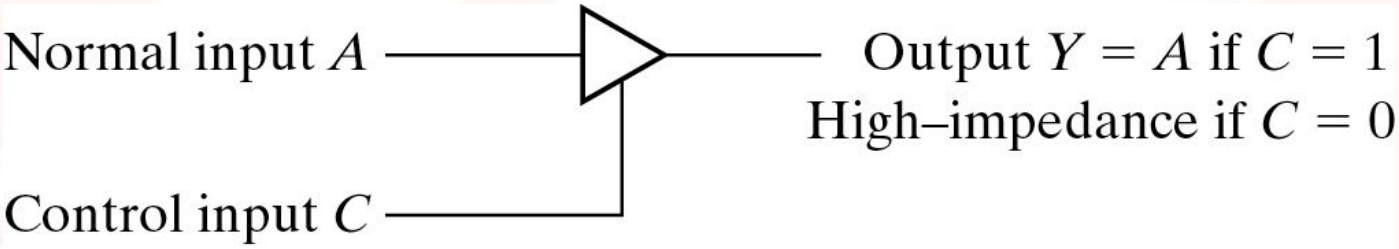


Fig. 4-28 Implementing a 4-Input Function with a Multiplexer

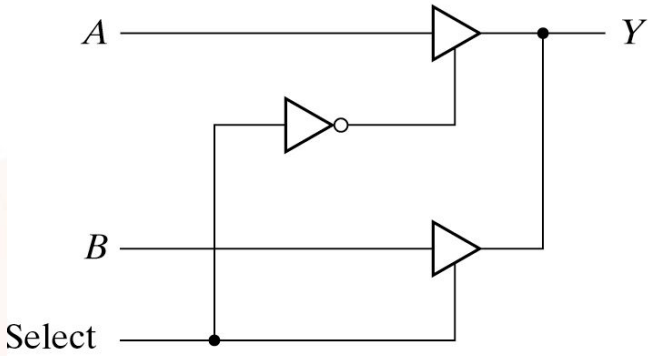
Three-state gates

- A multiplexer can be constructed with three-state gates
- Output state: 0, 1, and high-impedance (open circuit)

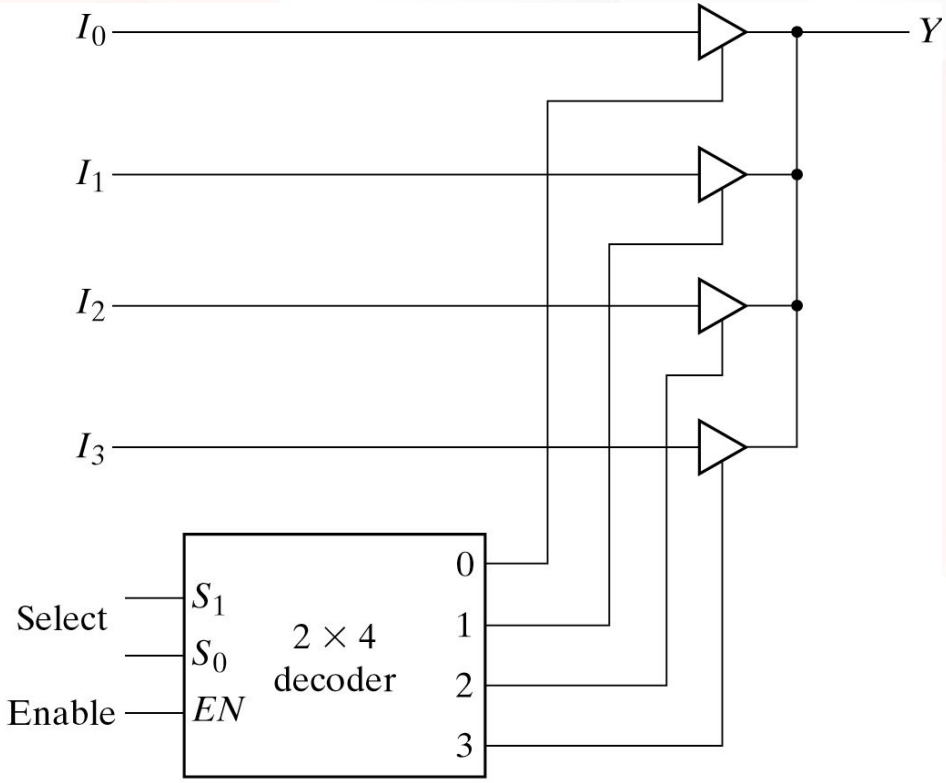


Tri state buffer

Multiplexer using three state gates



(a) 2-to-1- line mux



(b) 4 - to - 1 line mux