Combinational Logic Design

Dept. of Electrical Engineering and Computer Science
North South University

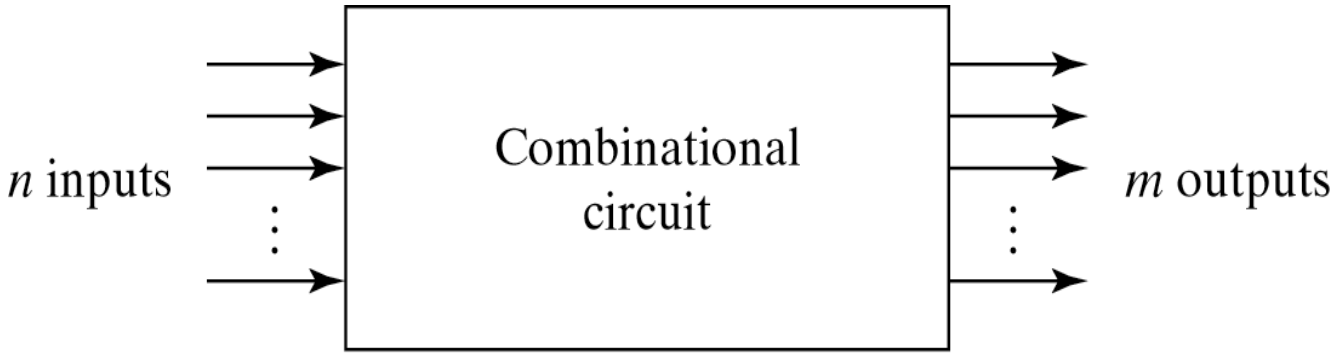
Combinational Logic

- Logic circuits for digital systems may be

  combinational

  sequential

- A combinational circuit consists of input variables, logic gates, and output variables.



Analysis procedure

- To obtain the output Boolean functions from a logic diagram, proceed as follows:

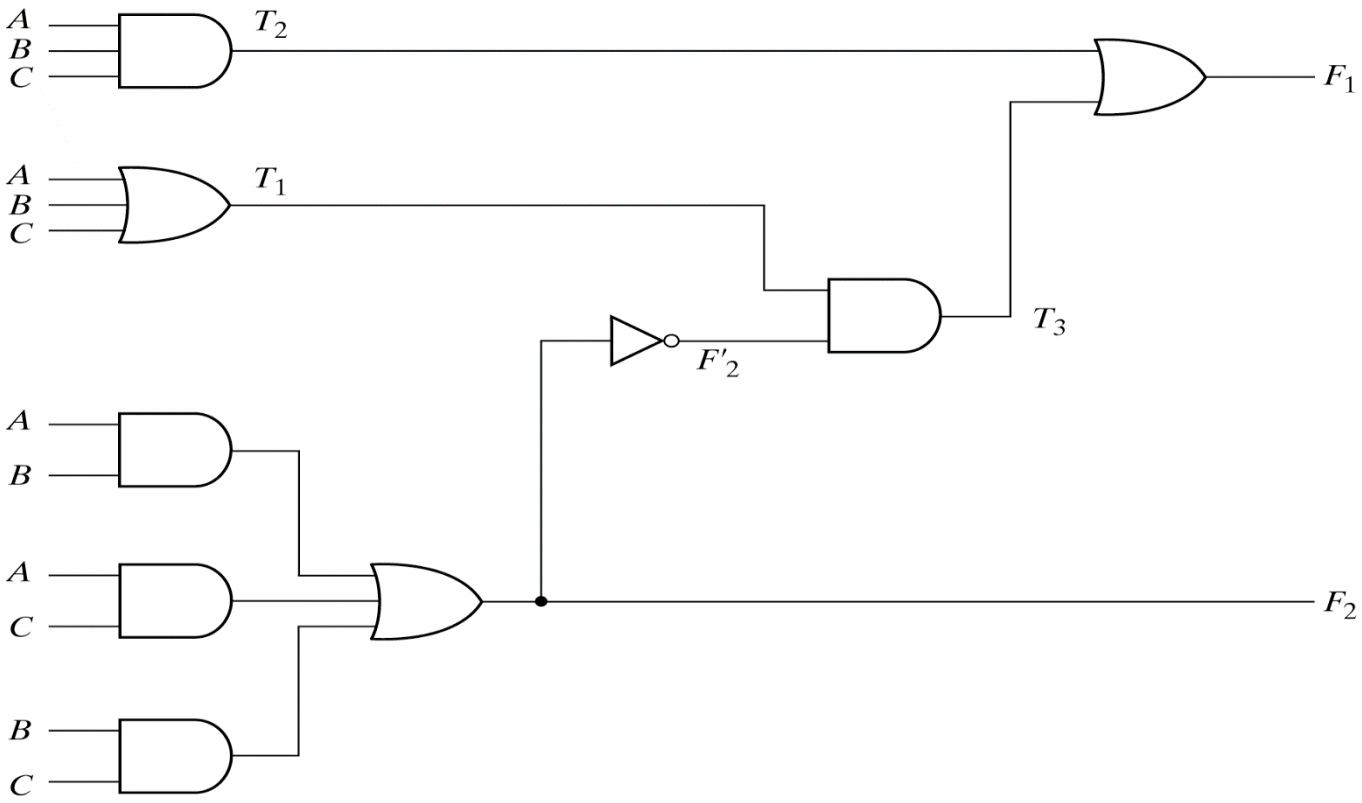
1. Label all gate outputs that are a function of input variables with arbitrary symbols. Determine the Boolean functions for each gate output.
2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates.
3. Repeat the process outlined in step 2 until the outputs of the circuit are obtained.
4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

Example

$F_2 = AB + AC + BC$; $T_1 = A + B + C$; $T_2 = ABC$; $T_3 = F_2'T_1$;

$F_1 = T_3 + T_2$

$F_1 = T_3 + T_2 = F_2'T_1 + ABC = A'BC' + A'B'C + AB'C' + ABC$



Combinational Logic Design

- A process with 5 steps
  - Specification
  - Formulation
  - Optimization
  - Technology mapping
  - Verification
- 1st three steps and last best illustrated by example

Specifications

- Write a specification for the circuits
- Specification includes
  - What are the inputs: how many, how many bits in a given output, how are they grouped,, are they control, are they active high?
  - What are the outputs: how many and how many bits in a each, active high, active low, tristate output?
  - The functional operation that takes place in the chip, i.e., for given inputs what will appear on the outputs.

Formulation step

- Convert the specifications into a variety forms for optimal implementation.
  - Possible forms
    - Truth Tables
    - Expressions
    - K-maps
    - Binary Decision Diagrams
- IF THE SPECIFCATION IS ERRONOUS OR INCOMPLETE (open for various interpretation) then the circuit will perform as specified but will not perform as desired.

Last 3 steps

- Best illustrated by example
  - A BCD to Excess-3 code converter
  - BCD-to-7-segment decoder

BCD-to-Excess-3 Code converter

- BCD is a code for the decimal digits 0-9
- Excess-3 is also a code for the decimal digits

Decimal Digit	Input BCD	Output Excess-3
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0

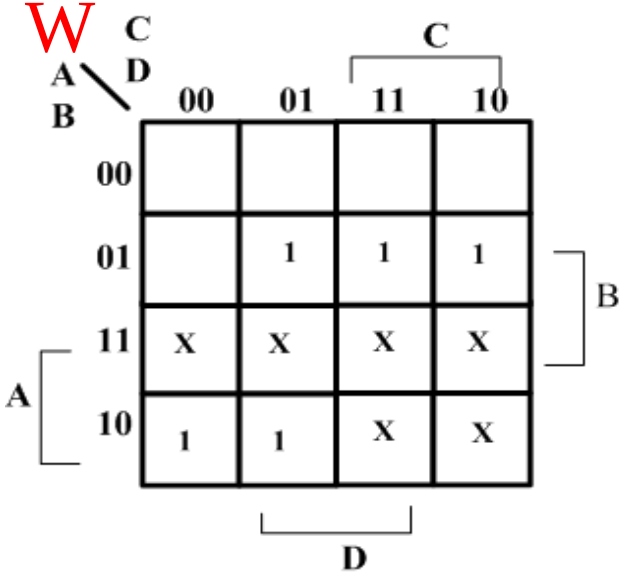
Design Procedure for BCD-to-Excess3

- **Specification:**
  - Inputs: a BCD input, A,B,C,D with A as the most significant bit and D as the least significant bit.
  - Outputs: an Excess-3 output W,X,Y,Z that corresponds to the BCD input.
  - Internal operation – circuit to do the conversion in combinational logic.
- **Formulation**
  - Excess-3 code is easily formed by adding a binary 3 to the binary or BCD for the digit.
  - There are 16 possible inputs for both BCD and Excess-3.
  - It can be assumed that only valid BCD inputs will appear so the six combinations not used can be treated as don't cares.

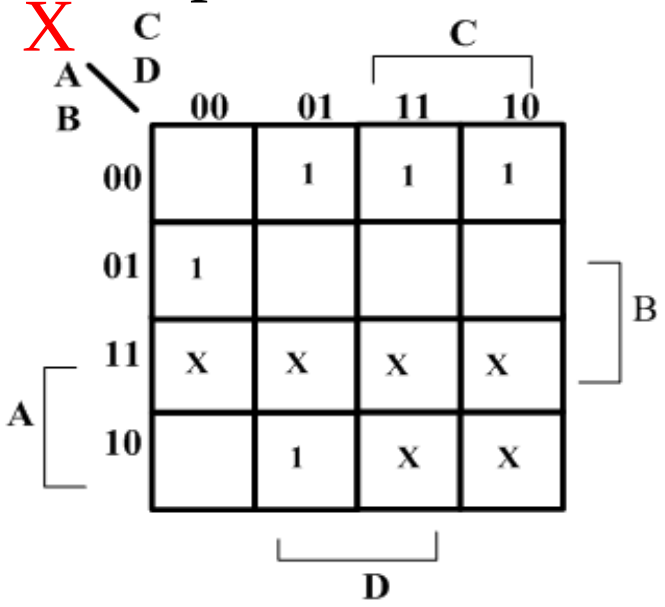
Optimization – BCD-to-Excess-3

Lay out K-maps for each output, W X Y Z

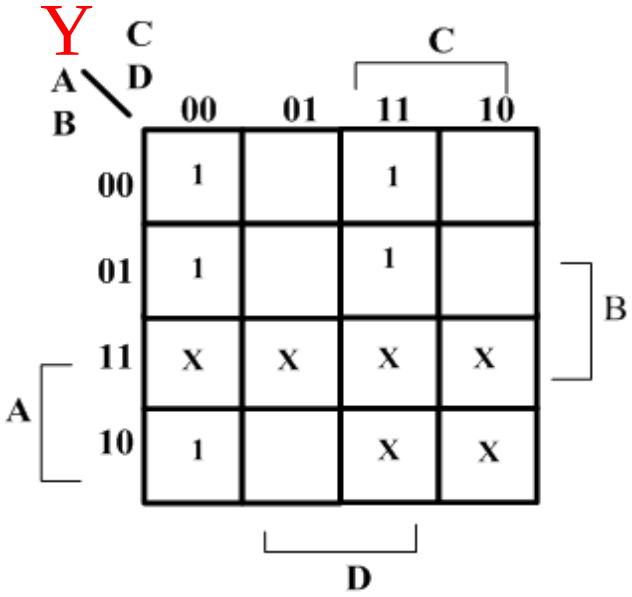
W



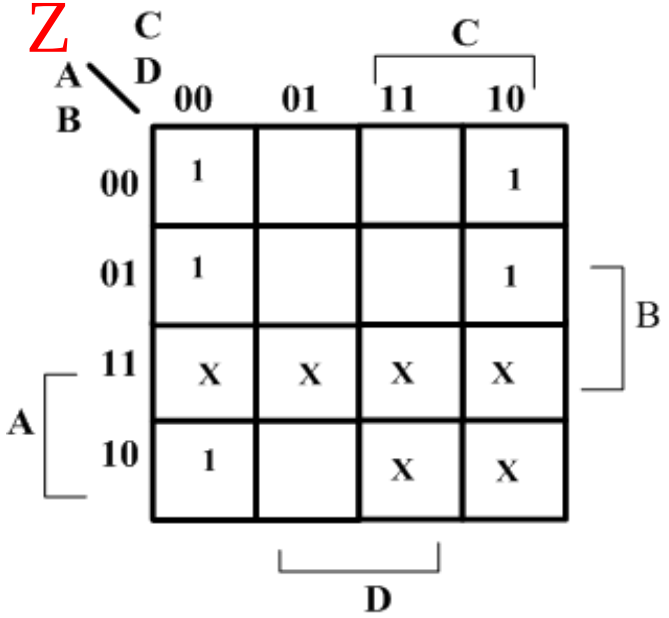
X



Y



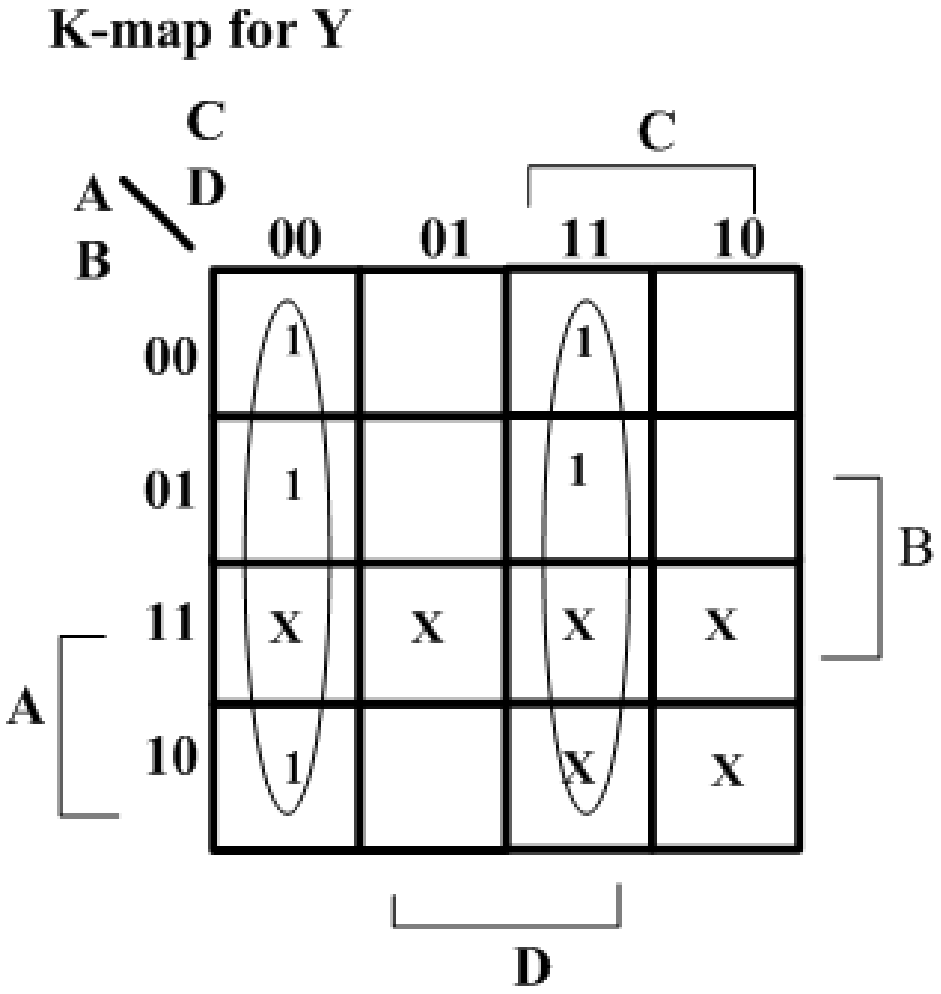
Z



Expressions for W X Y Z

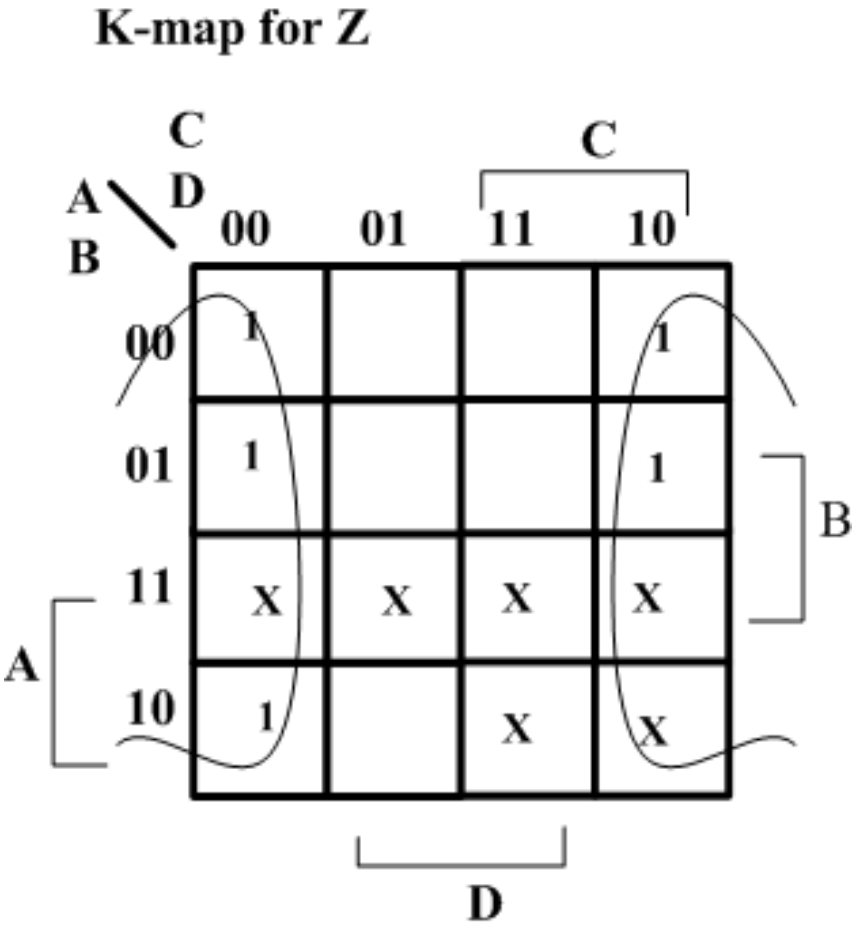
- $W(A,B,C,D) = \Sigma m(5,6,7,8,9) + d(10,11,12,13,14,15)$
- $X(A,B,C,D) = \Sigma m(1,2,3,4,9) + d(10,11,12,13,14,15)$
- $Y(A,B,C,D) = \Sigma m(0,3,4,7,8) + d(10,11,12,13,14,15)$
- $Z(A,B,C,D) = \Sigma m(0,2,4,6,8) + d(10,11,12,13,14,15)$

Minimize K-Maps



- Y minimization

  Y = CD + C'D'

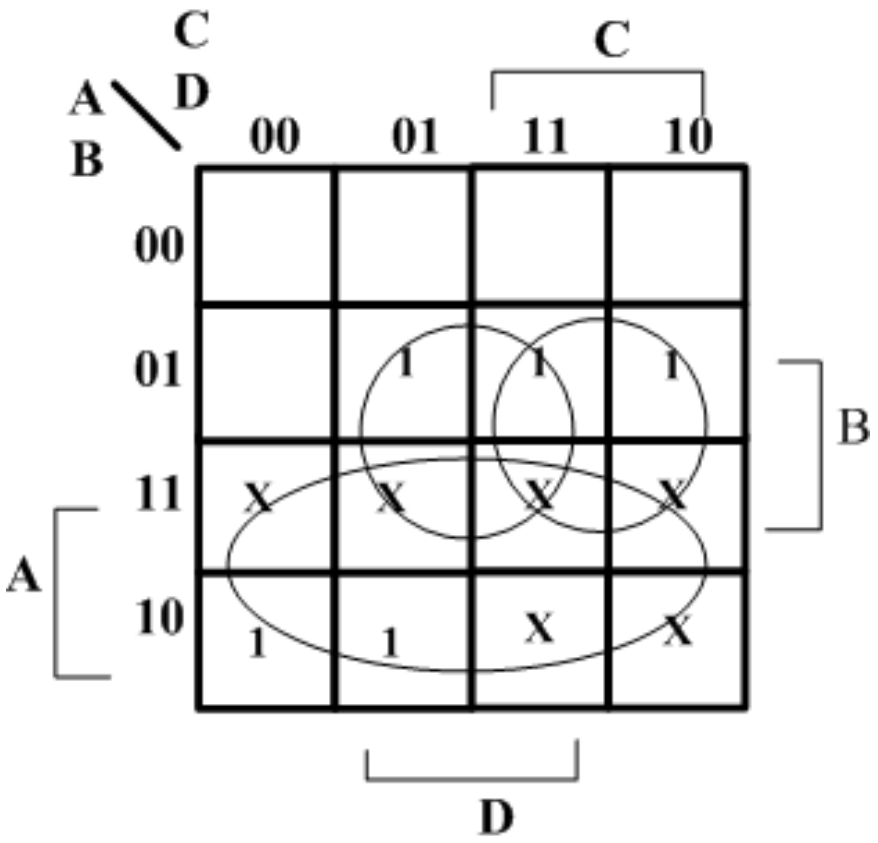


- Z minimization

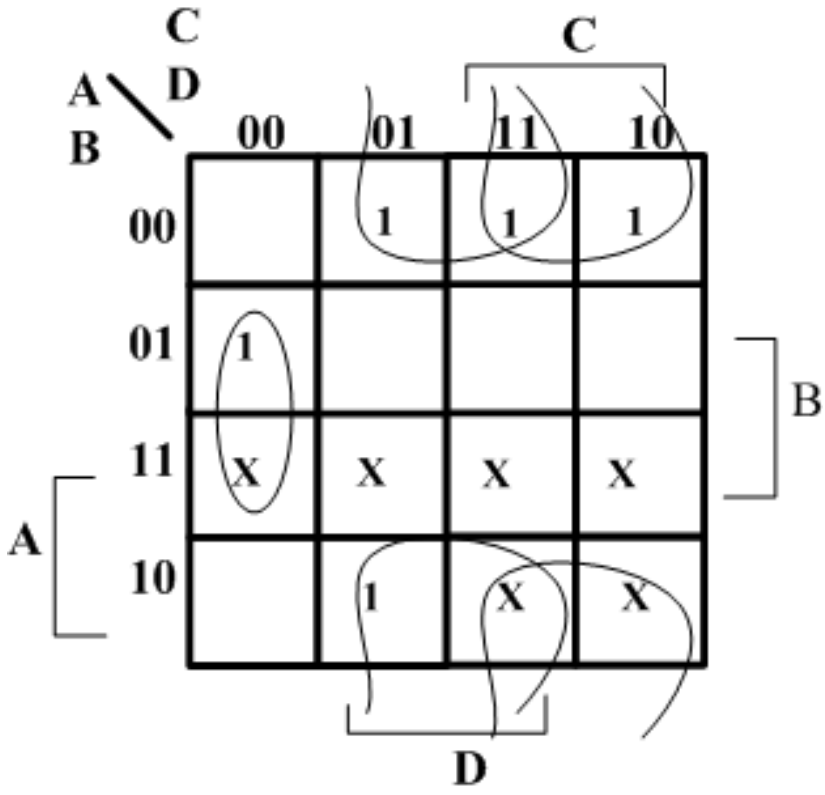
  Z = D'

Minimize K-Maps

K-map for W



K-map for X



- W minimization

  W = A + BC + BD

- X minimization

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

Two level circuit implementation

- Have equations

  W = A + BC + BD = A + B(C+D)

  X = B'C + B'D + BC'D' = B'(C+D) + BC'D'

  Y = CD + C'D'

  Z = D'

- Factoring out (C+D) and call it T
- Then T' = (C+D)' = C'D'

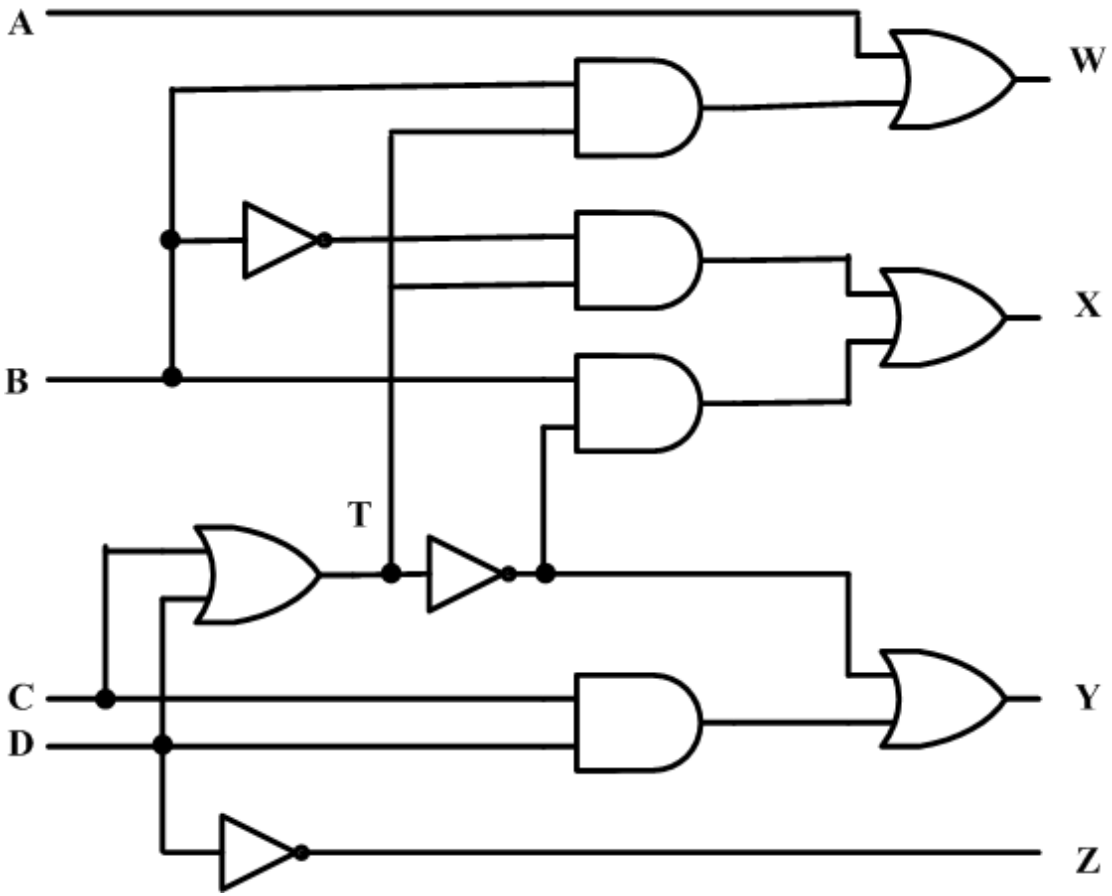
  W = A + BT

  X = B'T + BT'

  Y = CD + T'

  Z = D'

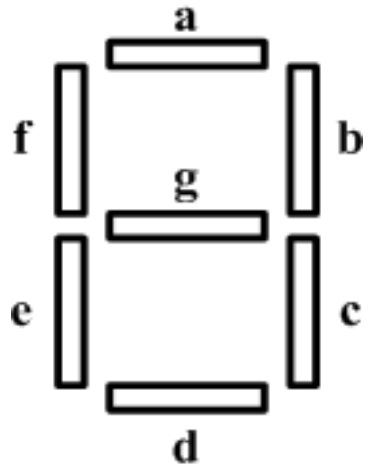
Create the digital circuit



- Implementing the second set of equations where T=C+D results in a lower gate count.
- This gate has a fanout of 3

BCD-to-Seven-Segment Decoder

- **Specification**
  - Digital readouts on many digital products often use LED seven-segment displays.
  - Each digit is created by lighting the appropriate segments. The segments are labeled a,b,c,d,e,f,g
  - The decoder takes a BCD input and outputs the correct code for the seven-segment display.



- **Formulation**
  - Input: A 4-bit binary value that is a BCD coded input.
  - Outputs: 7 bits, a through g for each of the segments of the display.
  - Operation: Decode the input to activate the correct segments.

Formulation

- Construct a truth table

Decimal Digit	Input BCD	Seven-Segment Decoder Outputs a b c d e f g
0	0 0 0 0	1 1 1 1 1 1 0
1	0 0 0 1	0 1 1 0 0 0 0
2	0 0 1 0	1 1 0 1 1 0 1
3	0 0 1 1	1 1 1 1 0 0 1
4	0 1 0 0	1 0 1 1 0 1 1
5	0 1 0 1	1 0 1 1 0 1 1
6	0 1 1 0	1 0 1 1 1 1 1
7	0 1 1 1	1 1 1 0 0 0 0
8	1 0 0 0	1 1 1 1 1 1 1
9	1 0 0 1	1 1 1 1 0 1 1
All other inputs		0 0 0 0 0 0 0

Optimization

- Create a K-map for each output and get

  a = A’C+A’BD+B’C’D’+AB’C’

  b = A’B’+A’C’D’+A’CD+AB’C’

  c = A’B+A’D+B’C’D’+AB’C’

  d = A’CD’+A’B’C+B’C’D’+AB’C’+A’BC’D

  e = A’CD’+B’C’D’

  f = A’BC’+A’C’D’+A’BD’+AB’C’

  g = A’CD’+A’B’C+A’BC’+AB’C’

Note on implementation

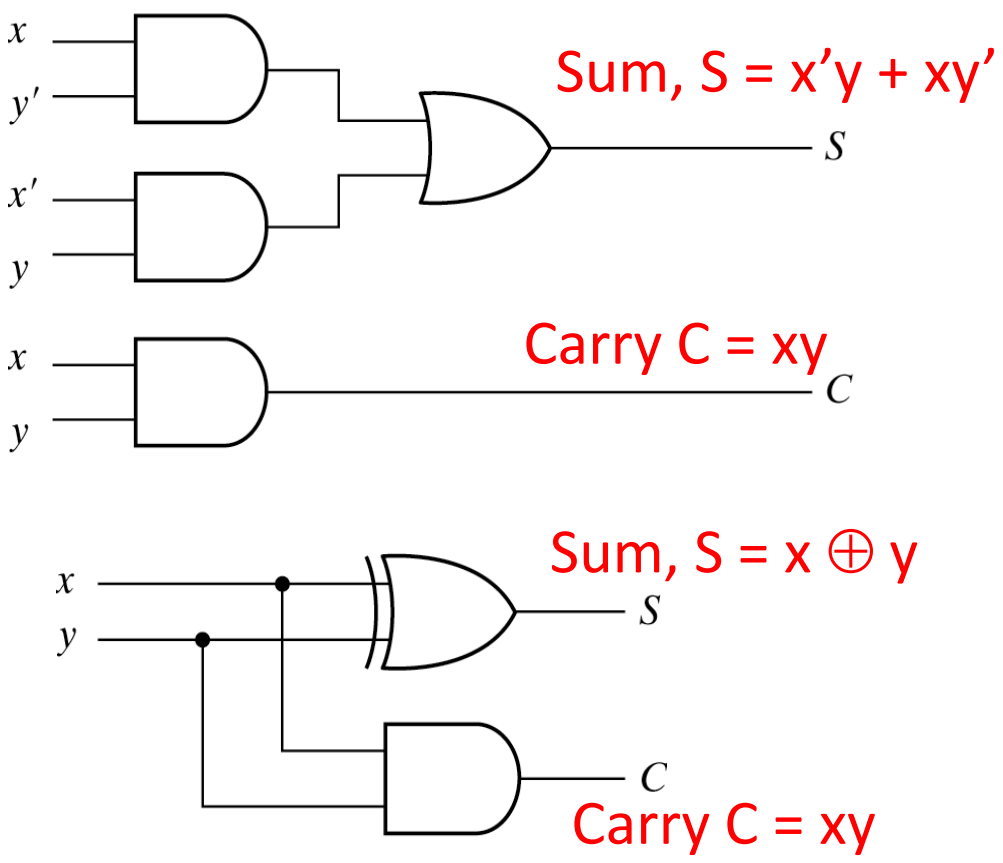
- Direct implementation would require 27 AND gates and 7 OR gates
- By sharing terms, can actualize and implementation with 14 less gates.
- Normally decoder in a device name indicates that the number of outputs is less than the number of inputs.

Binary Adder-Subtractor

- A combinational circuit that performs the addition of two bits is called a half adder.
- The truth table for the half adder is listed below:

Truth Table – Half Adder

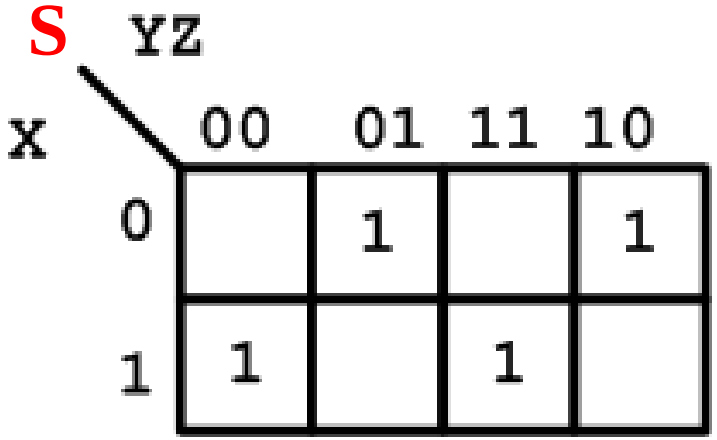
X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



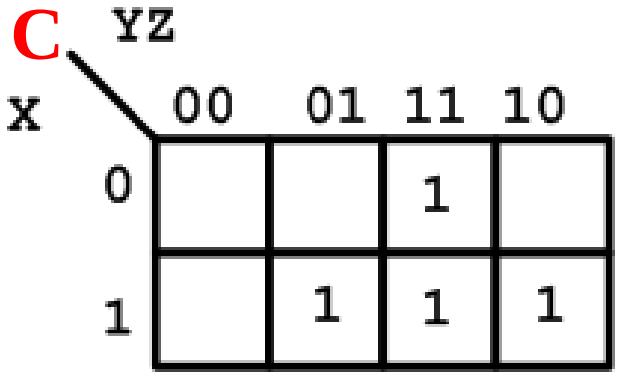
Full-Adder

- Performs the addition of three bits (two significant bits and a previous carry)

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

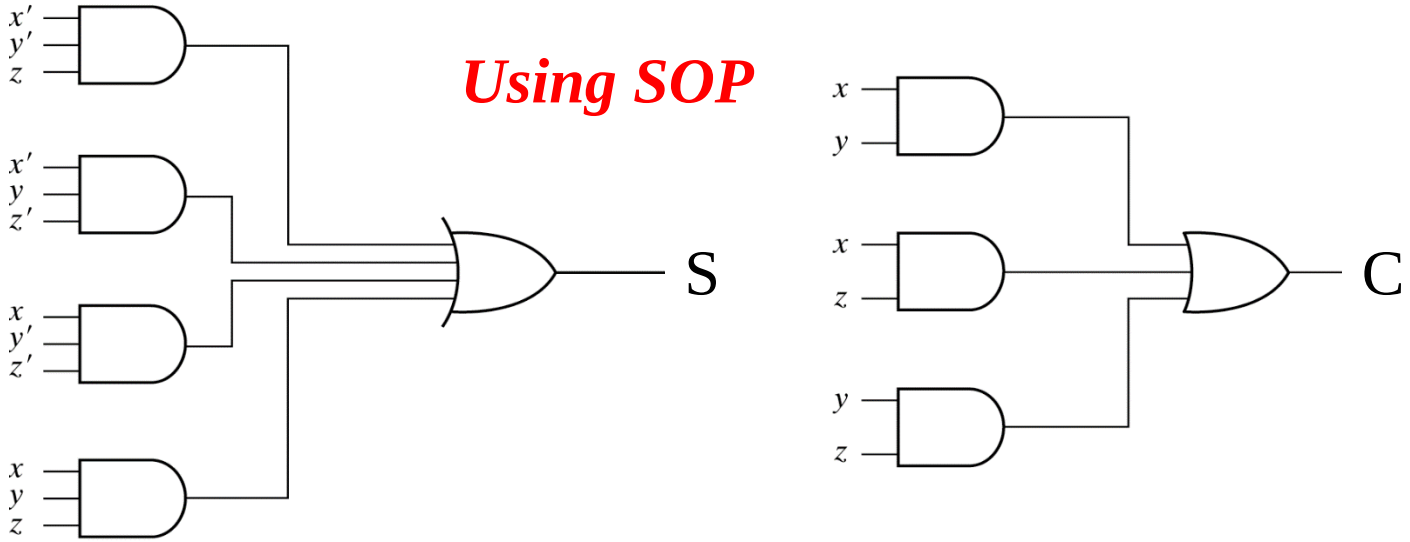


$$S = X'Y'Z + X'YZ' + XY'Z' + XYZ$$
$$= X \oplus Y \oplus Z$$
$$= (X \oplus Y) \oplus Z$$

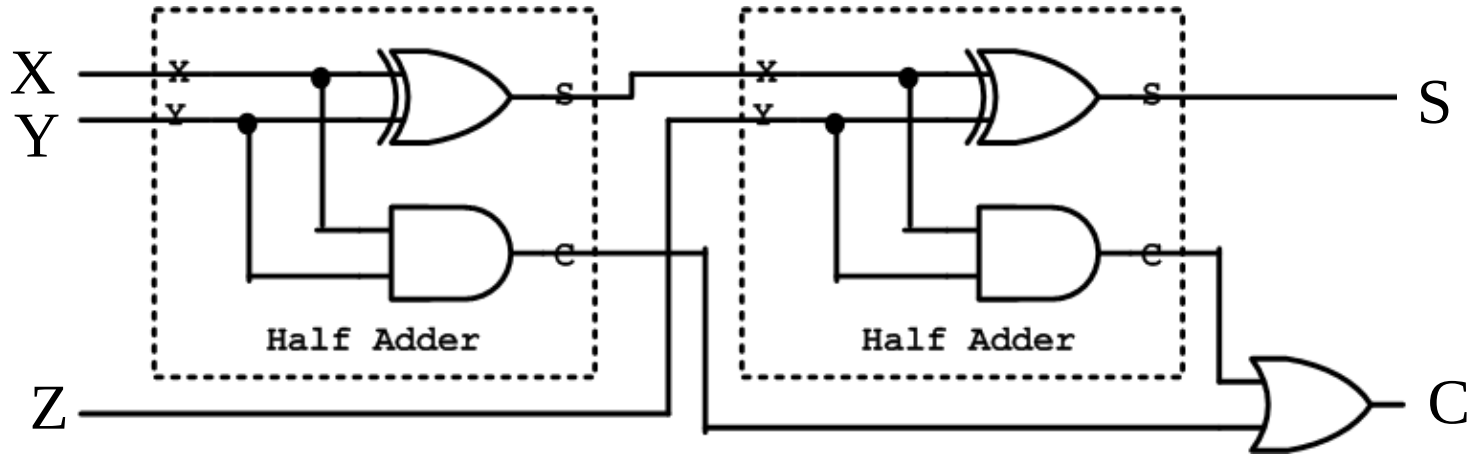


$$C = XY + XY'Z + X'YZ$$
$$= XY + Z(XY' + X'Y)$$
$$= XY + Z(X \oplus Y)$$

Full adder Implementation

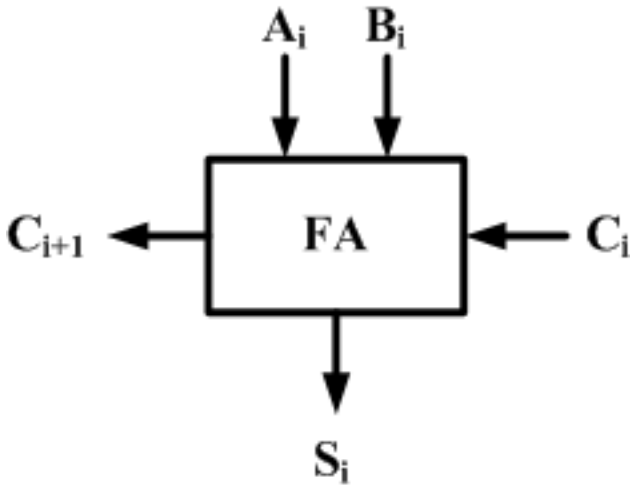


Using two half adders and one OR gate (Carry Look-Ahead adder)



Full Adder Symbol

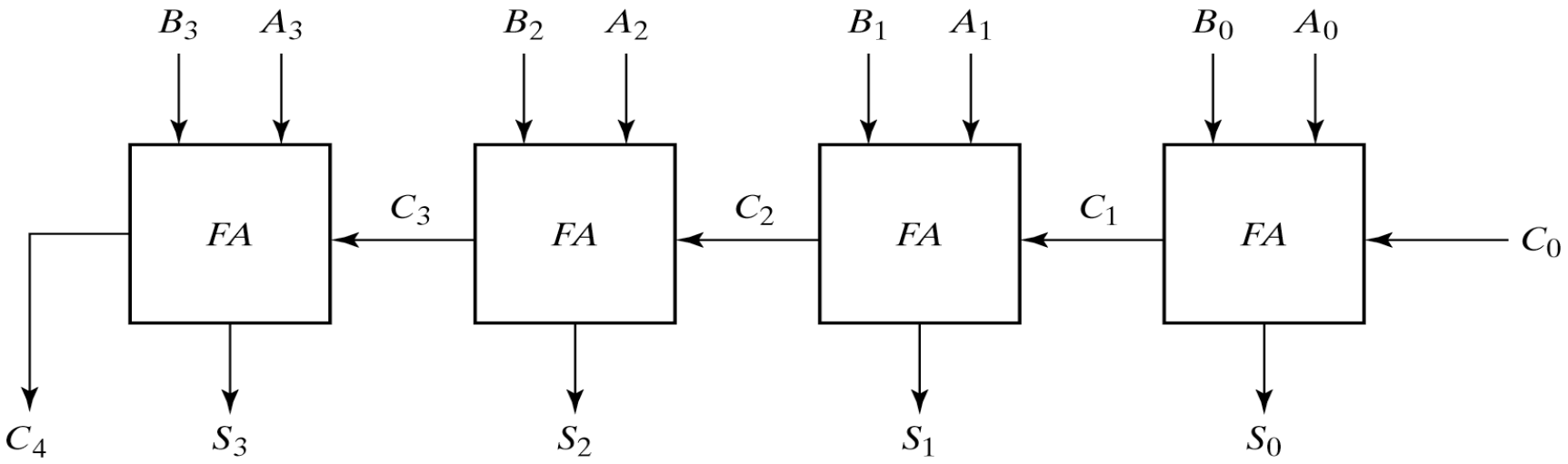
- For a multibit implementation need a symbol for the unit. And then can use that symbol in multi-bit or hierarchical representations.



Binary adder

- Ripple Carry Adder (RCA): full adders are connected in cascade.
- All inputs, As, Bs, and C_0 arrive –> C_1 becomes valid –> C_2 becomes valid –> C_3 becomes valid –> C_4 becomes valid

Subscript i:	3	2	1	0	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

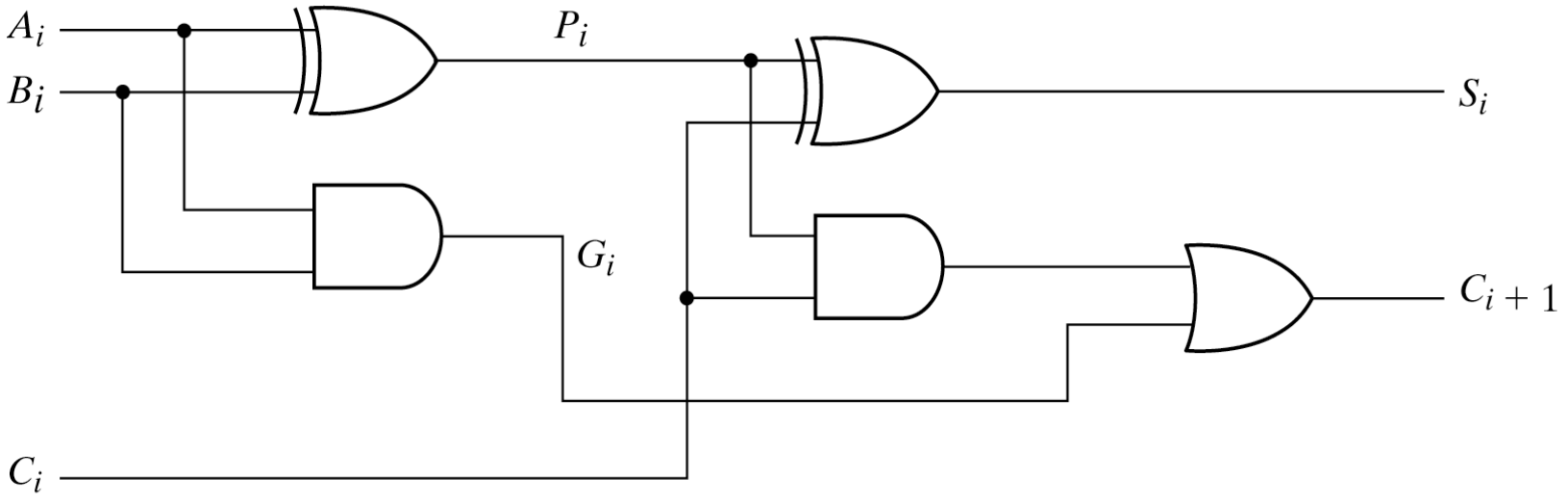


Carry Propagation

- Causes a unstable factor on carry bit, and produces a longest propagation delay.
- The signal from C_i to the output carry C_{i+1}, propagates through an AND and OR gates, so, for an n-bit RCA, there are 2n gate levels for the carry to propagate from input to output.

Carry Propagation

- Because the propagation delay will affect the output signals on different time, so the signals are given enough time to get the precise and stable outputs.
- The most widely used technique employs the principle of carry look-ahead to improve the speed of the algorithm.



Boolean functions of CLA-Adder

$P_i = A_i \oplus B_i$ steady state value

$G_i = A_i B_i$ steady state value

Output sum and carry

$S_i = P_i \oplus C_i$

$C_{i+1} = G_i + P_i C_i$

G_i : carry generate P_i : carry propagate

C_0 = input carry

$C_1 = G_0 + P_0 C_0$

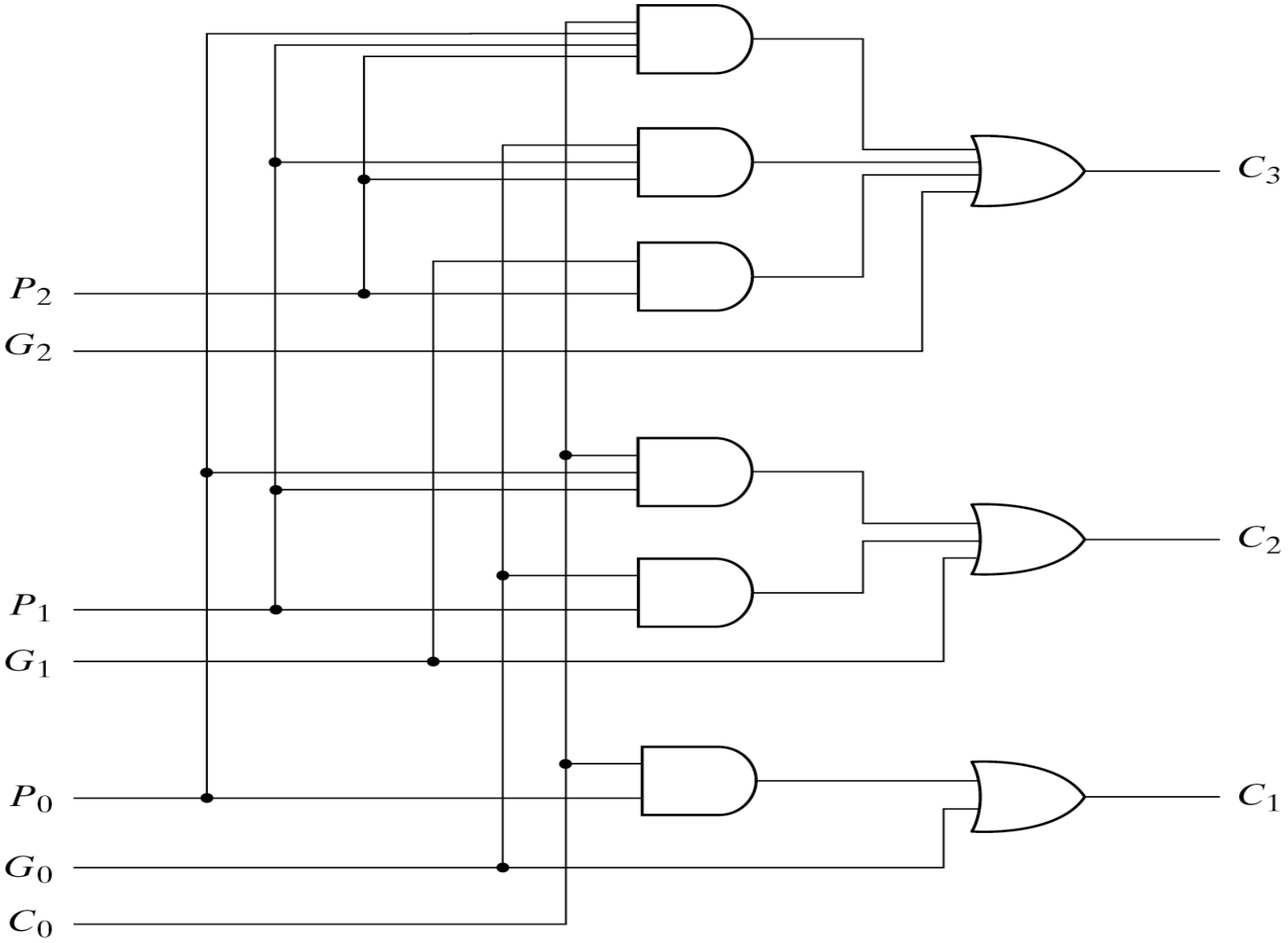
$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_0$

$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

- C_3 does not have to wait for C_2 and C_1 to propagate.

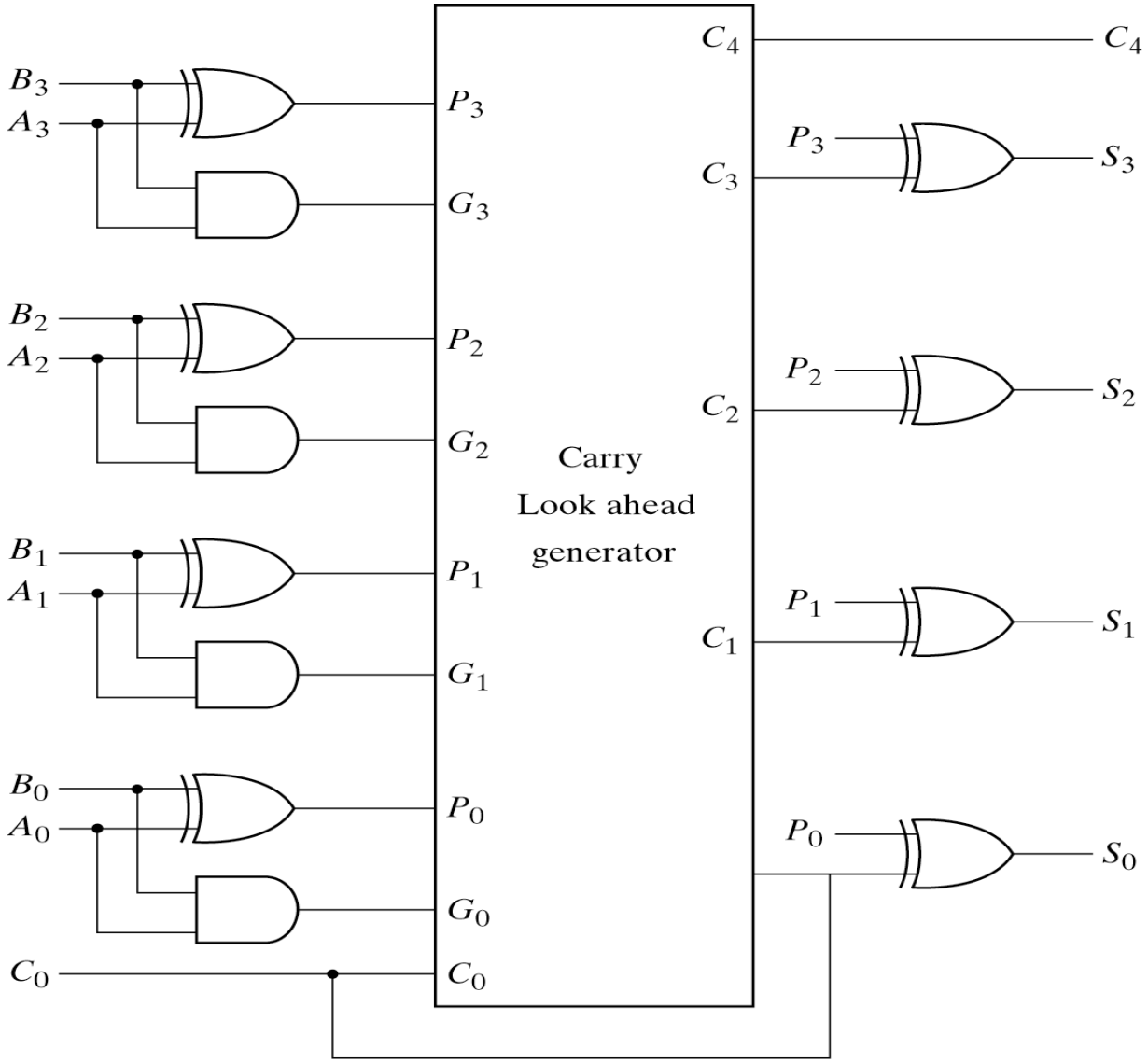
Logic Diagram of CLA Generator

- C_3 is propagated at the same time as C_2 and C_1.



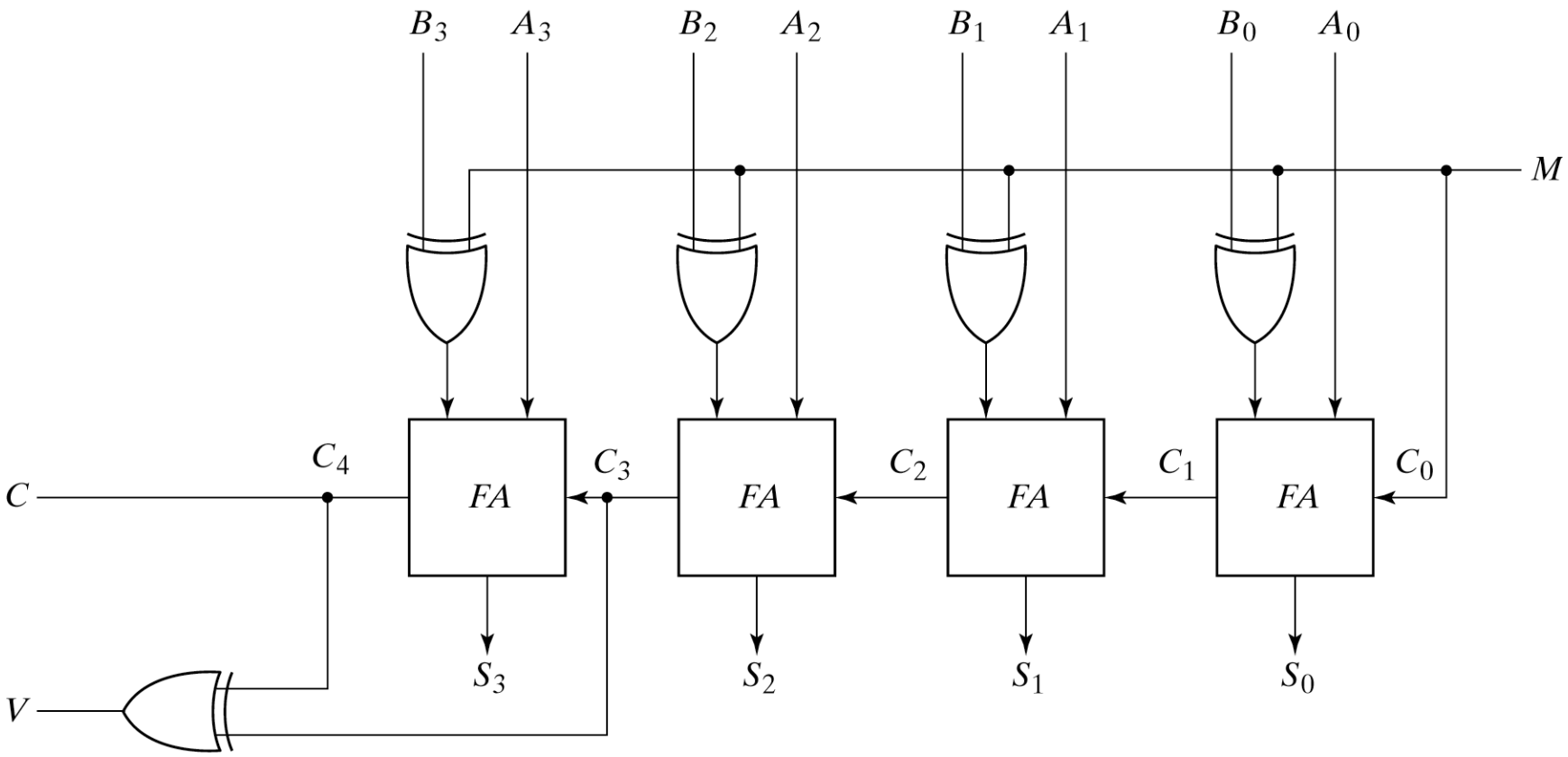
4-bit CLA-Adder

- Delay time of n-bit CLAA = XOR + (AND + OR) + XOR



Binary Adder-Subtractor

M = 1→subtractor ; M = 0→adder



Overflow on signed and unsigned

- Binary numbers in the signed-complement system are added and subtracted by the same basic addition and subtraction rules as unsigned numbers.
- Overflow is a problem in digital computers because the number of bits that hold the number is finite and a result that contains n+1 bits cannot be accommodated.
- When two unsigned numbers are added, an overflow is detected from the end carry out of the MSB position.
- When two signed numbers are added, the sign bit is treated as part of the number and the end carry does not indicate an overflow.
- An overflow cann't occur after an addition if one number is positive and the other is negative.
- An overflow may occur if the two numbers added are both positive or both negative.

Overflow indication.

- In 8-bit 2's complement notation the range that can be represented is -127 to +127.
- Then the operation to add +70 to +80 is

```
Carries   0 1
  +70       0 100 0110
  +80       0 101 0000
 +150       1 001 0110
```

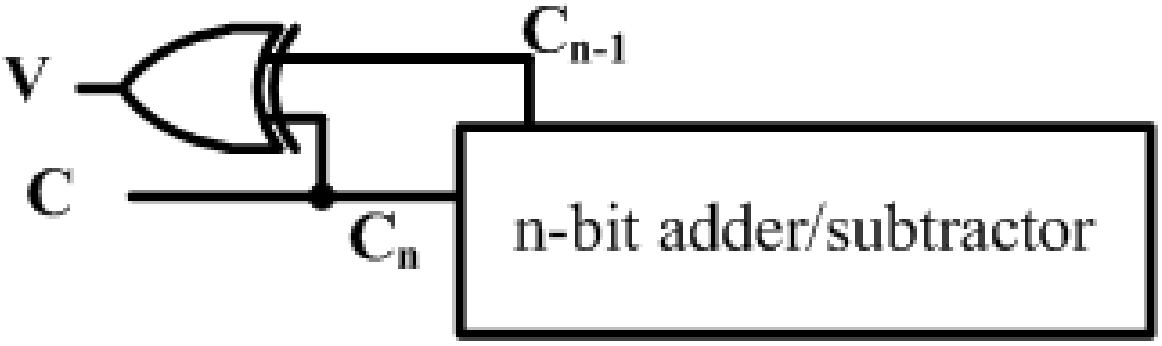
- Also look at the addition of -70 and -80

The other addition

- The addition of -70 and -80

```
 Carries   1 0
   -70      1 011 1010
   -80      1 011 0000
  -150      0 110 1010
```

- The rule – if the carry into the msb position differs from the carry out from the msb position then an overflow has occurred.



Decimal adder

BCD adder can't exceed 9 on each input digit. K is the carry.

Table 4-5
Derivation of BCD Adder

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

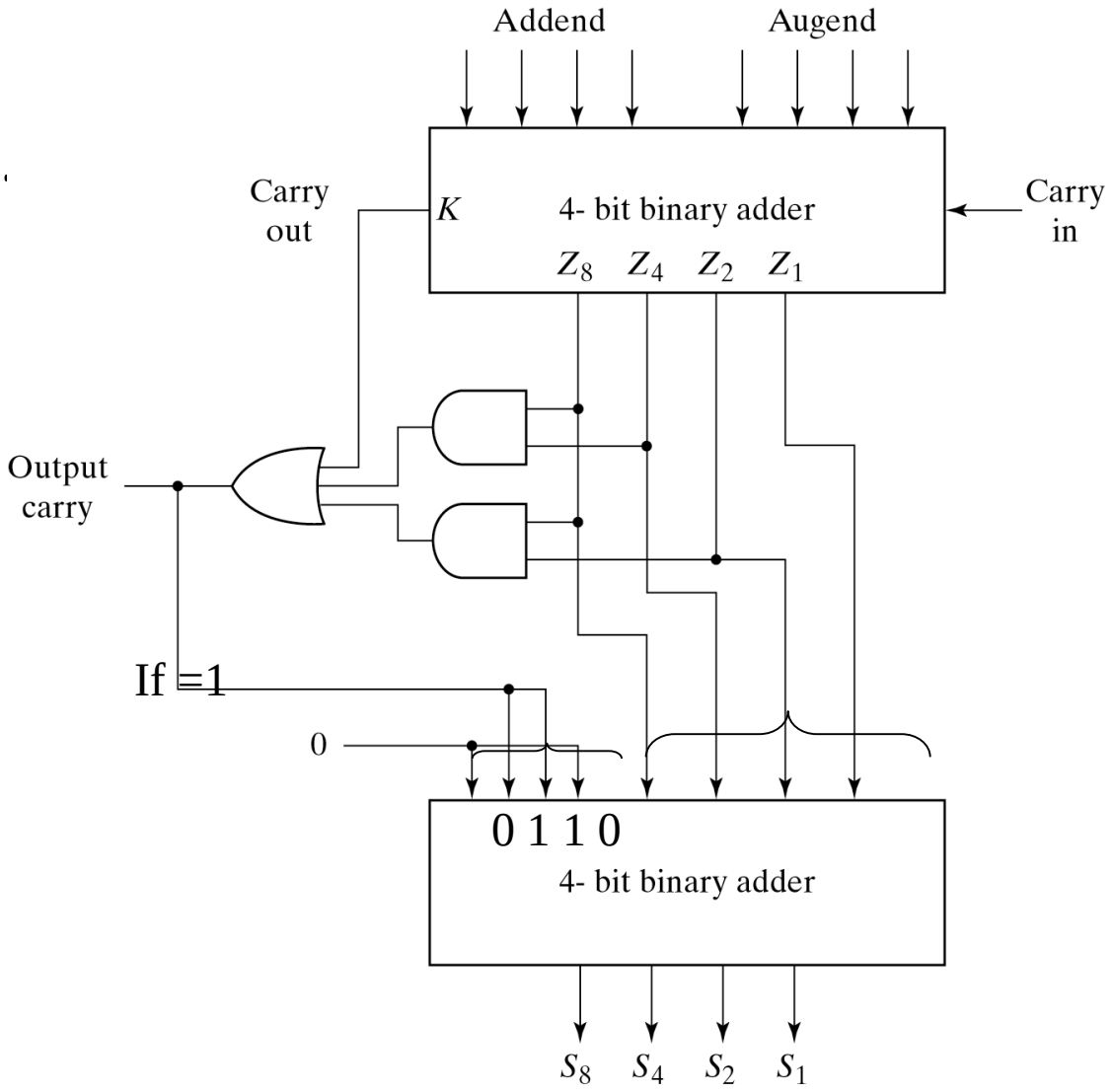
Rules of BCD adder

- When the binary sum is greater than 1001, we obtain a non-valid BCD representation.
- The addition of binary 6(0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.
- To distinguish them from binary 1000 and 1001, which also have a 1 in position Z_8, we specify further that either Z_4 or Z_2 must have a 1.

$$C = K + Z_8Z_4 + Z_8Z_2$$

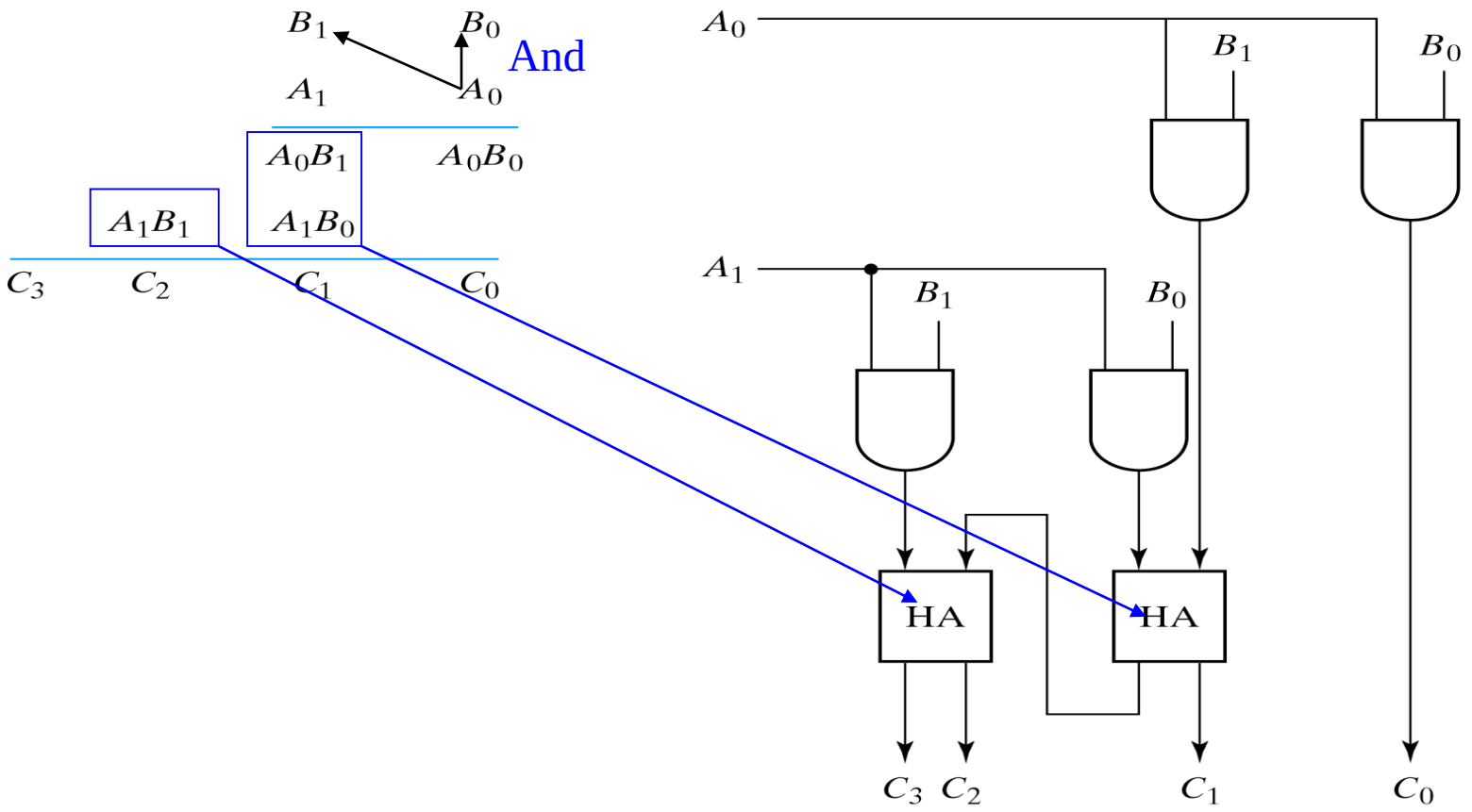
Implementation of BCD adder

- A decimal parallel adder that adds n decimal digits needs n BCD adder stages.
- The output carry from one stage must be connected to the input carry of the next higher-order stage.



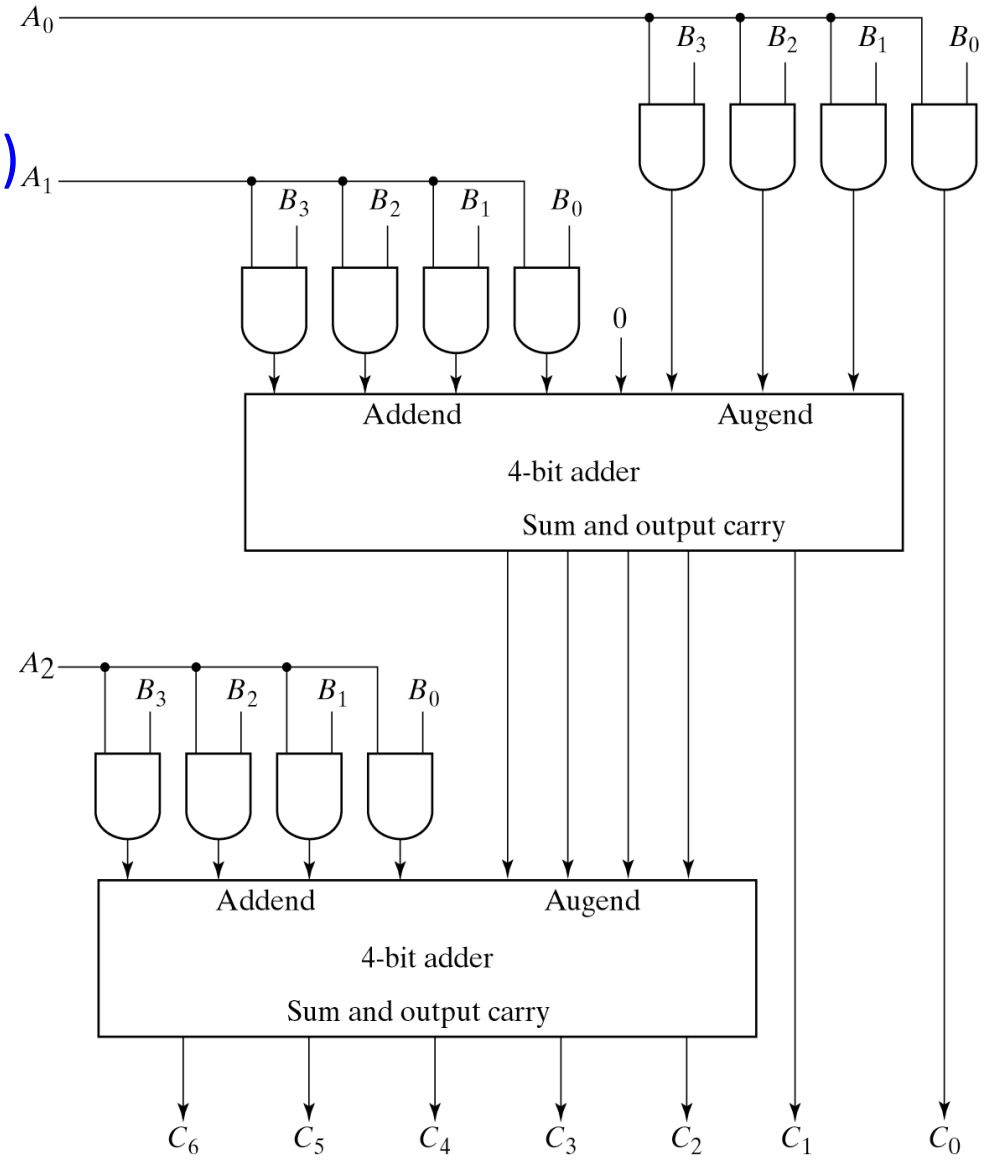
Binary multiplier

- Usually there are more bits in the partial products and it is necessary to use full adders to produce the sum of the partial products.



4-bit by 3-bit binary multiplier

- For J multiplier bits and K multiplicand bits we need (J X K) AND gates and (J – 1) K-bit adders to produce a product of J+K bits.
- K=4 and J=3, we need 12 AND gates and two 4-bit adders.



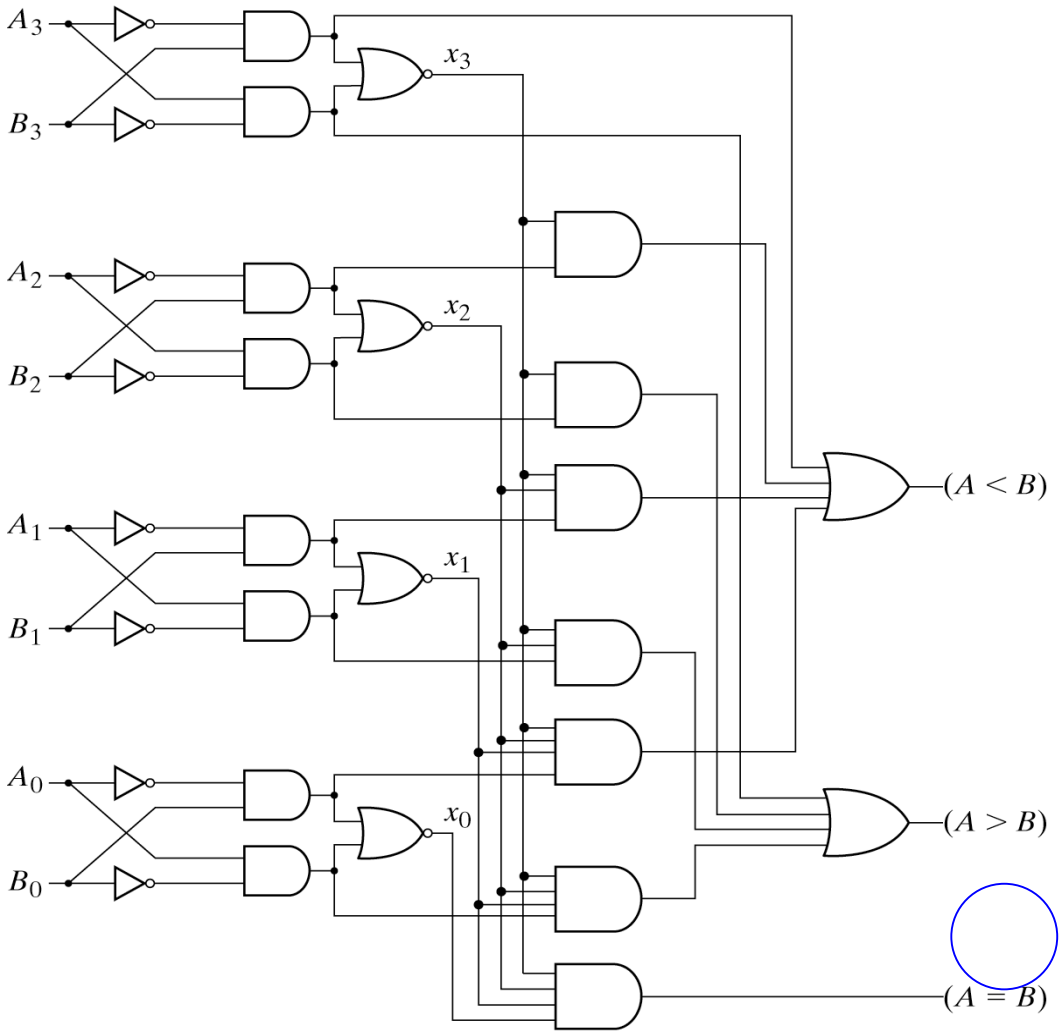
Magnitude comparator

- The equality relation of each pair of bits can be expressed logically with an exclusive-NOR function as:

$A = A_3A_2A_1A_0$; $B = B_3B_2B_1B_0$

$x_i = A_iB_i + A_i'B_i'$ for i = 0, 1, 2, 3

$(A = B) = x_3x_2x_1x_0$



Magnitude comparator

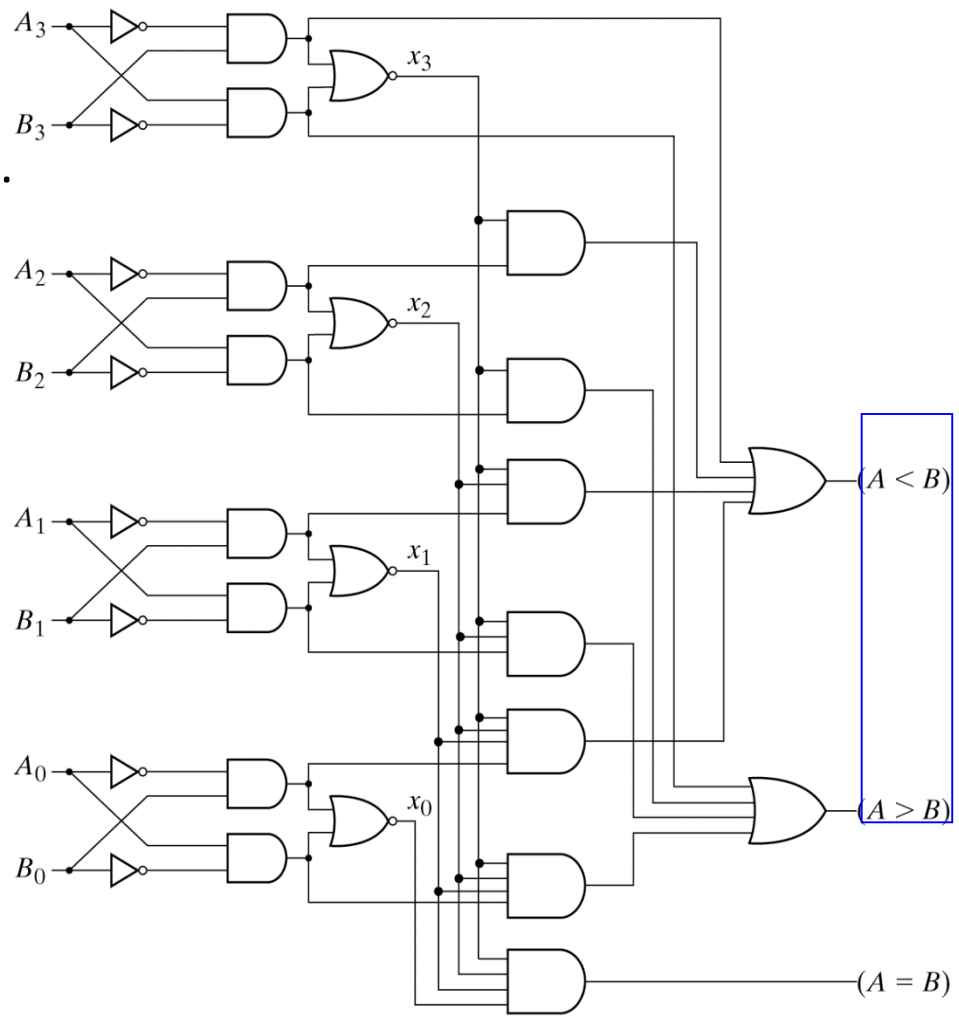
- We inspect the relative magnitudes of pairs of MSB. If equal, we compare the next lower significant pair of digits until a pair of unequal digits is reached.
- If the corresponding digit of A is 1 and that of B is 0, we conclude that A>B.

(A>B)=

$A_3B'_3+x_3A_2B'_2+x_3x_2A_1B'_1+x_3x_2x_1A_0B'_0$

(A<B)=

$A'_3B_3+x_3A'_2B_2+x_3x_2A'_1B_1+x_3x_2x_1A'_0B_0$

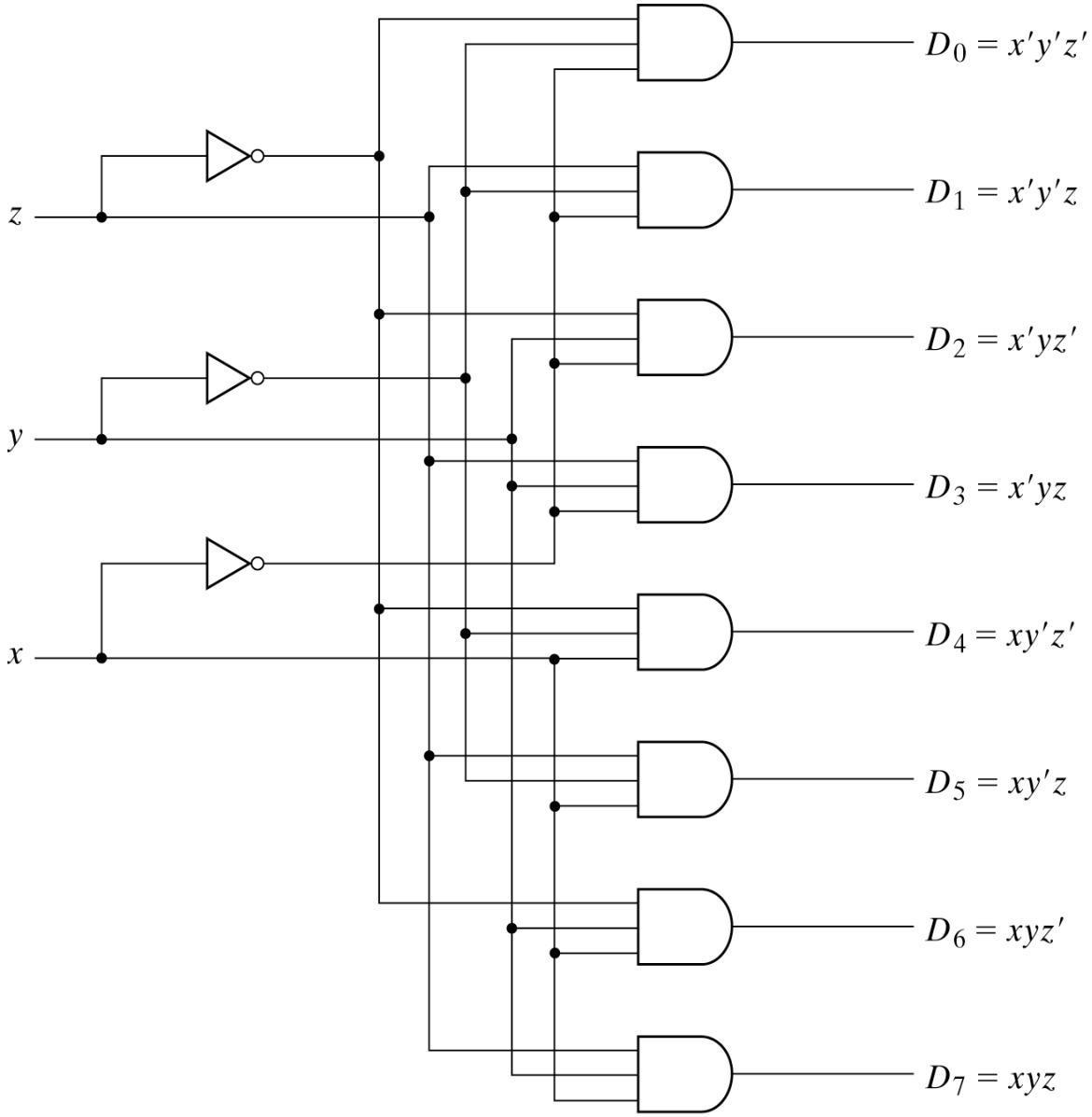


Decoders

- The decoder is called n-to-m-line decoder, where $m \leq 2^n$.
- the decoder is also used in conjunction with other code converters such as a BCD-to-seven_segment decoder.
- 3-to-8 line decoder: For each possible input combination, there are seven outputs that are equal to 0 and only one that is equal to 1.

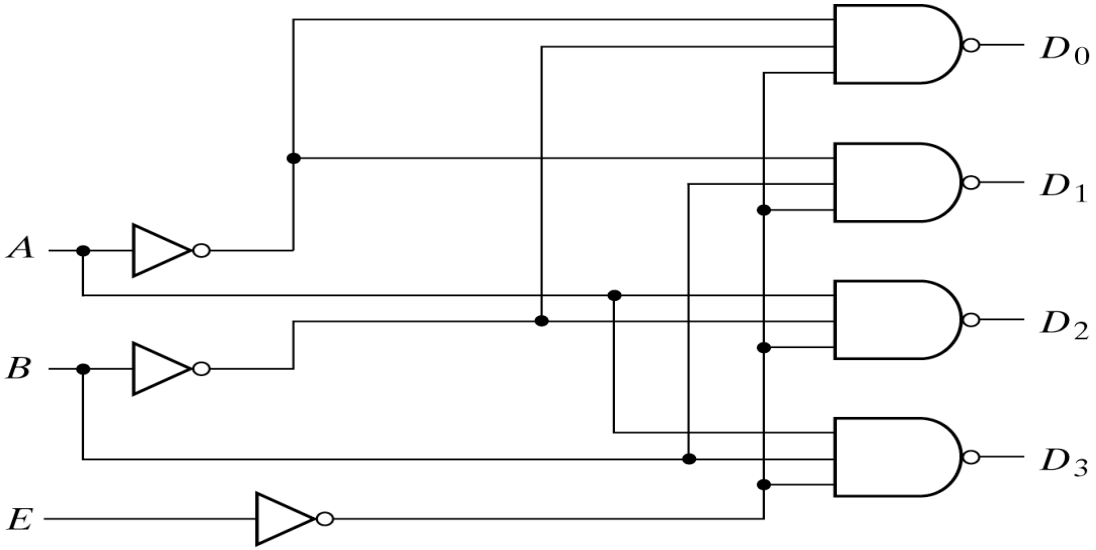
Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Implementation of 3-to-8 Decoder



Decoder with enable input

- Some decoders are constructed with NAND gates, it becomes more economical to generate the decoder minterms in their complemented form.
- As indicated by the truth table , only one output can be equal to 0 at any given time, all other outputs are equal to 1.



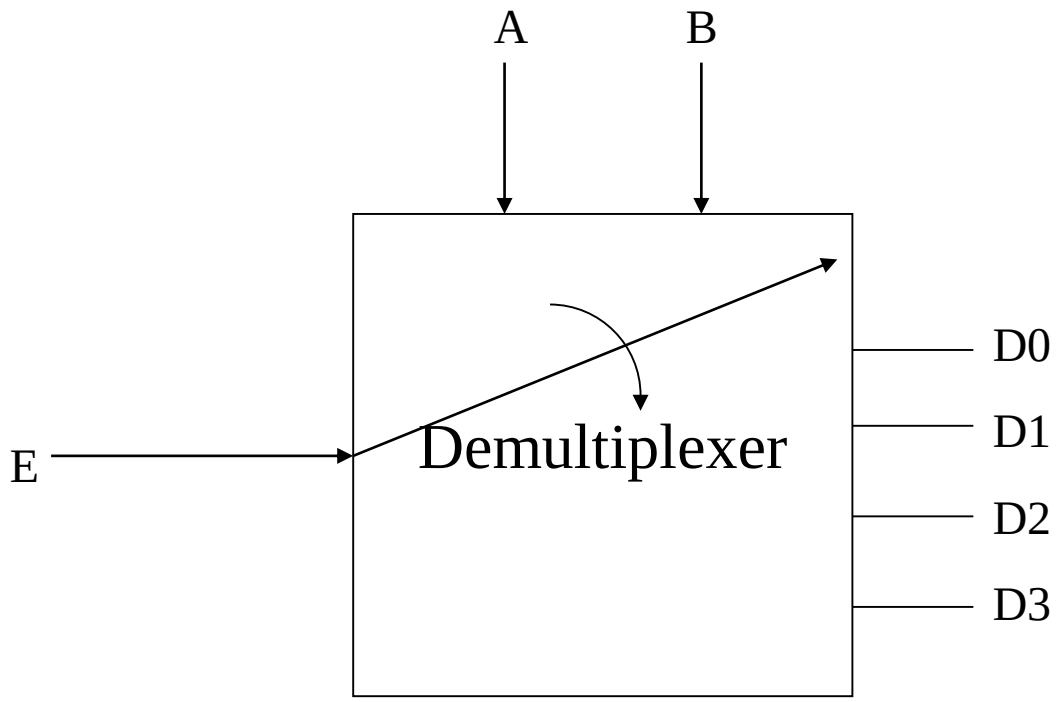
(a) Logic diagram

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

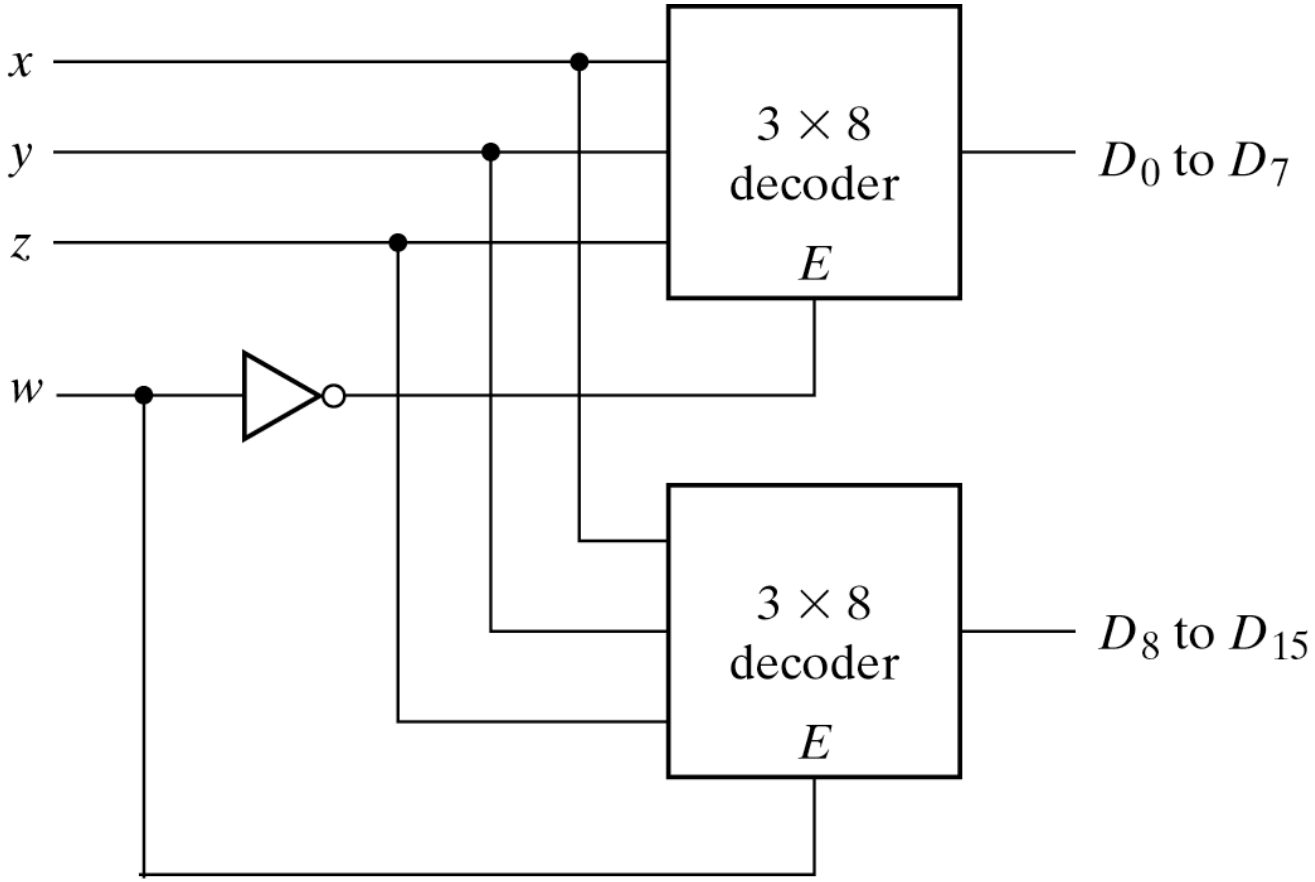
(b) Truth table

Demultiplexer

- A decoder with an enable input is referred to as a decoder/demultiplexer.
- The truth table of demultiplexer is the same with decoder.



3-to-8 decoder with enable implement the 4-to-16 decoder

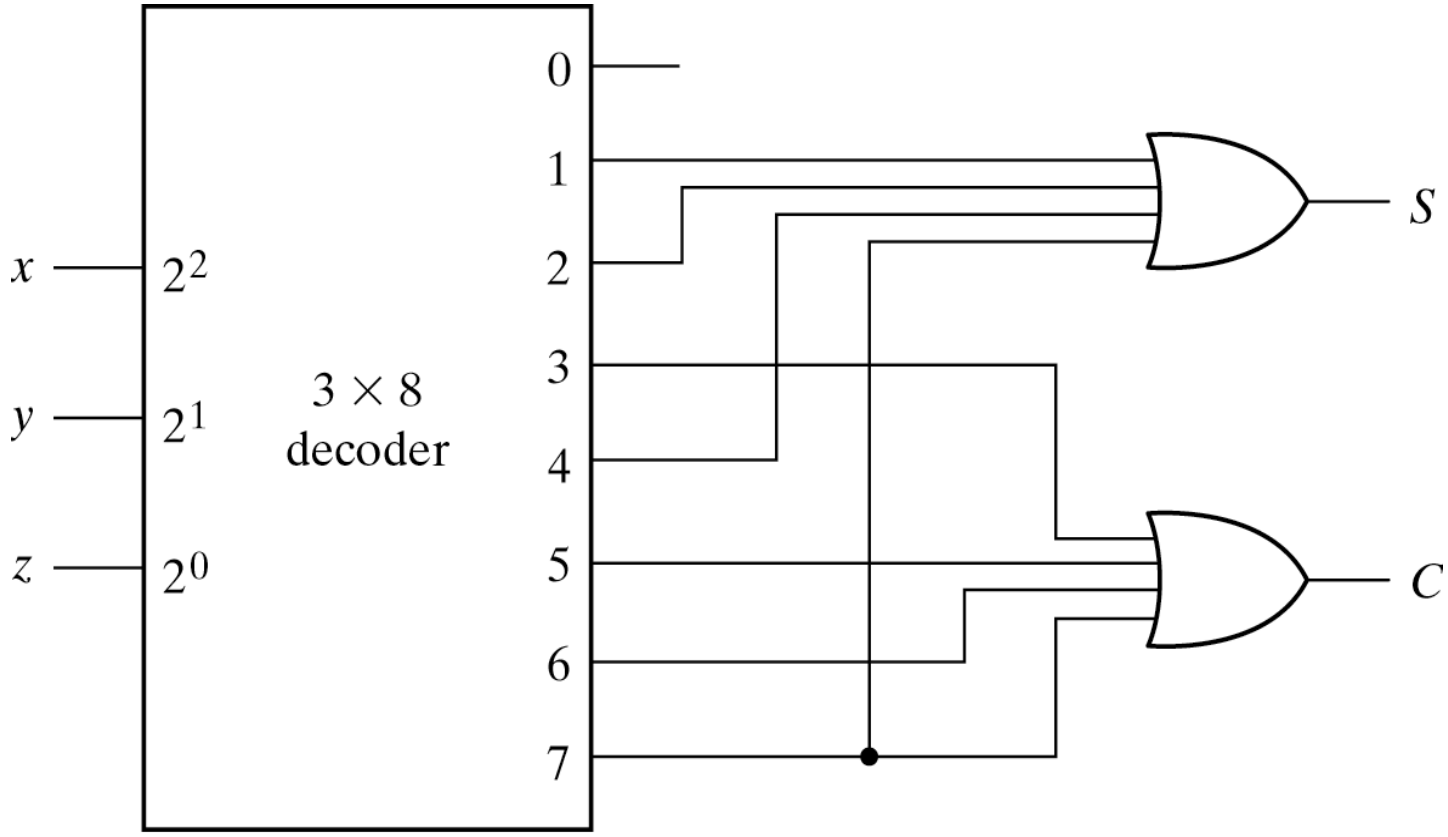


Full Adder using Decoder

- From the truth table, we obtain the functions for the combinational circuit in sum of minterms:

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

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



Encoders

- An encoder is the inverse operation of a decoder.
- We can derive the Boolean functions by table 4-7

$$z = D_1 + D_3 + D_5 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$x = D_4 + D_5 + D_6 + D_7$$

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

Priority encoder

- If two inputs are active simultaneously, the output produces an undefined combination. We can establish an input priority to ensure that only one input is encoded.
- Another ambiguity in the octal-to-binary encoder is that an output with all 0's is generated when all the inputs are 0; the output is the same as when D_0 is equal to 1.
- The discrepancy tables can resolve aforesaid condition by providing one more output to indicate that at least one input is equal to 1.

Priority encoder

V=0→no valid inputs

V=1→valid inputs

X's in output columns represent don't-care conditions

X's in the input columns are useful for representing a truth table in condensed form.

Instead of listing all 16 minterms of four variables.

Table 4-8
Truth Table of a Priority Encoder

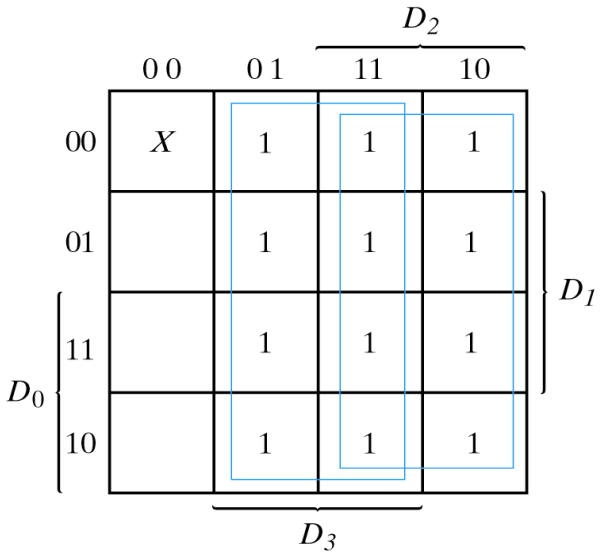
Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

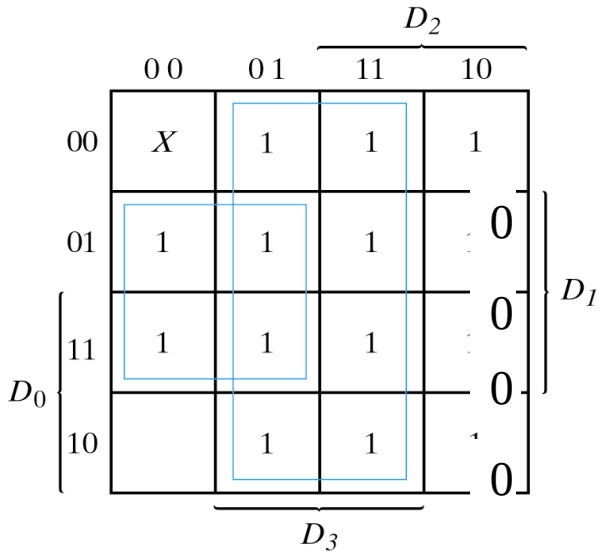
4-input priority encoder

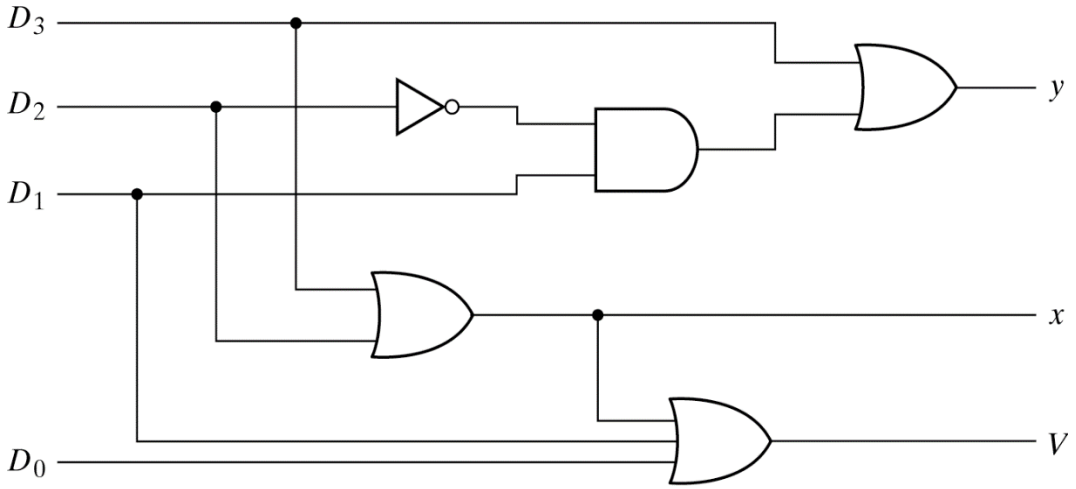
$x = D_2 + D_3$

$y = D_3 + D_1D'_2$

$V = D_0 + D_1 + D_2 + D_3$







Multiplexers

- Select from one of many Inputs and directs it to a single Output
- N selectors → 2^N Input lines

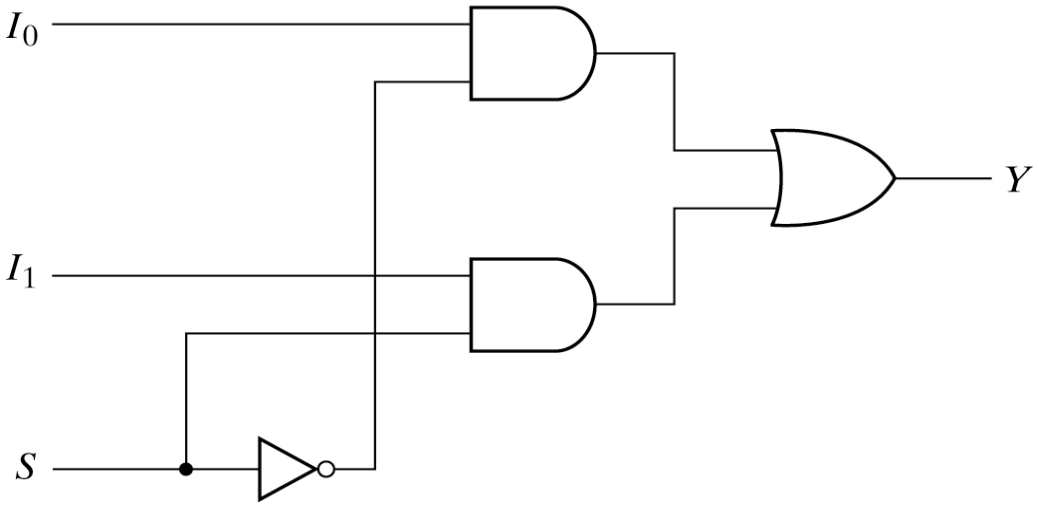
$S = 0, Y = I_0$

$S = 1, Y = I_1$

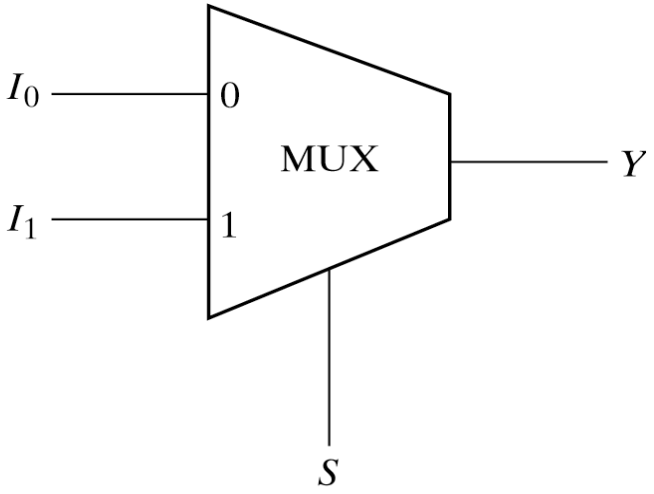
Truth Table →

S	Y
0	I_0
1	I_1

$Y = S'I_0 + SI_1$

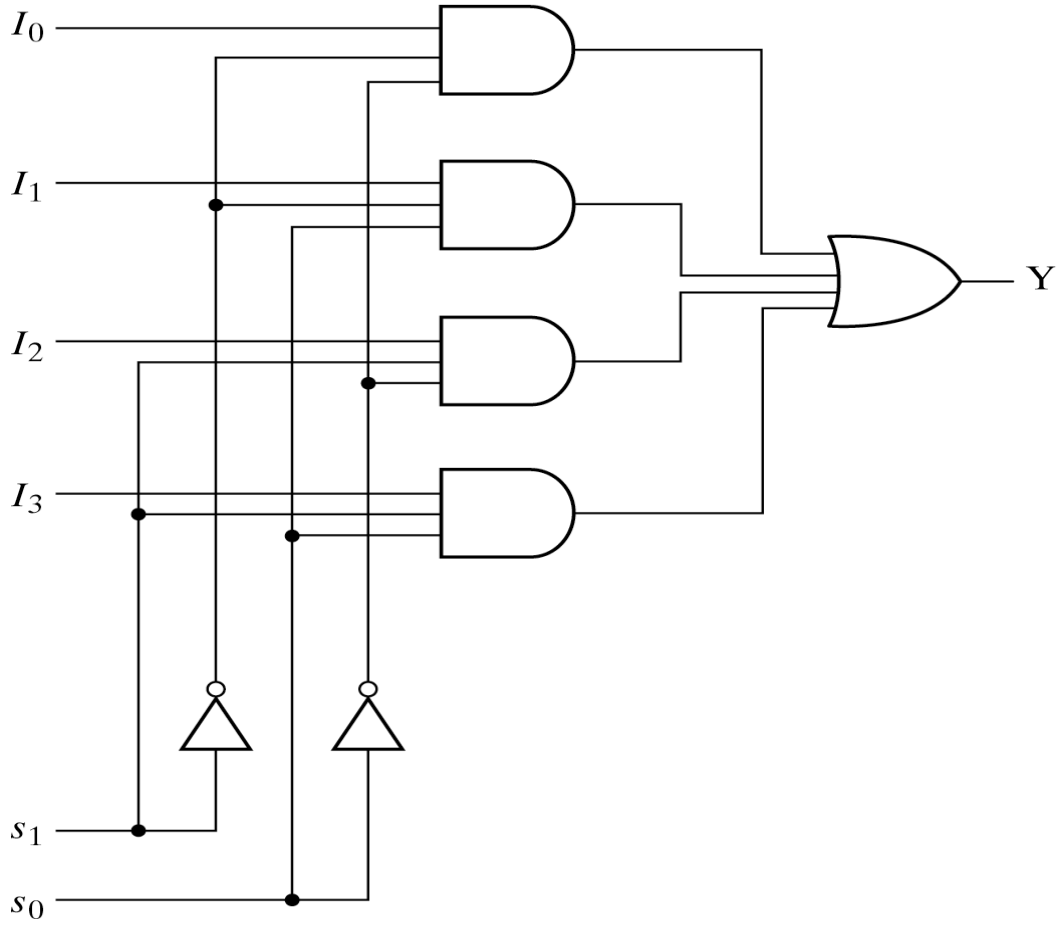


(a) Logic diagram



(b) Block diagram

4-to-1 Line Multiplexer



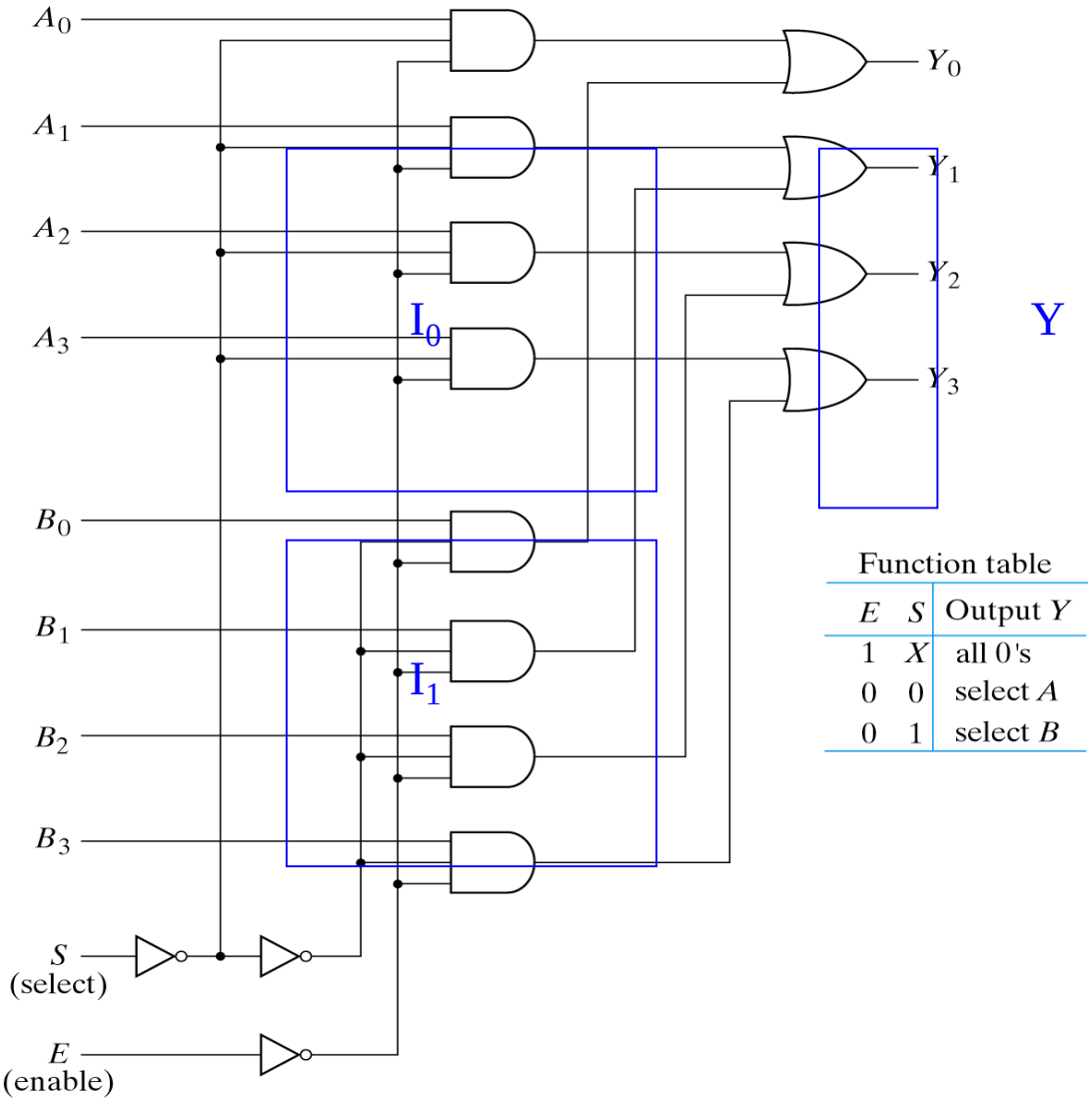
(a) Logic diagram

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

(b) Function table

Quadruple 2-to-1 Line Multiplexer

- Multiplexer circuits can be combined with common selection inputs to provide multiple-bit selection logic.



Function table

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

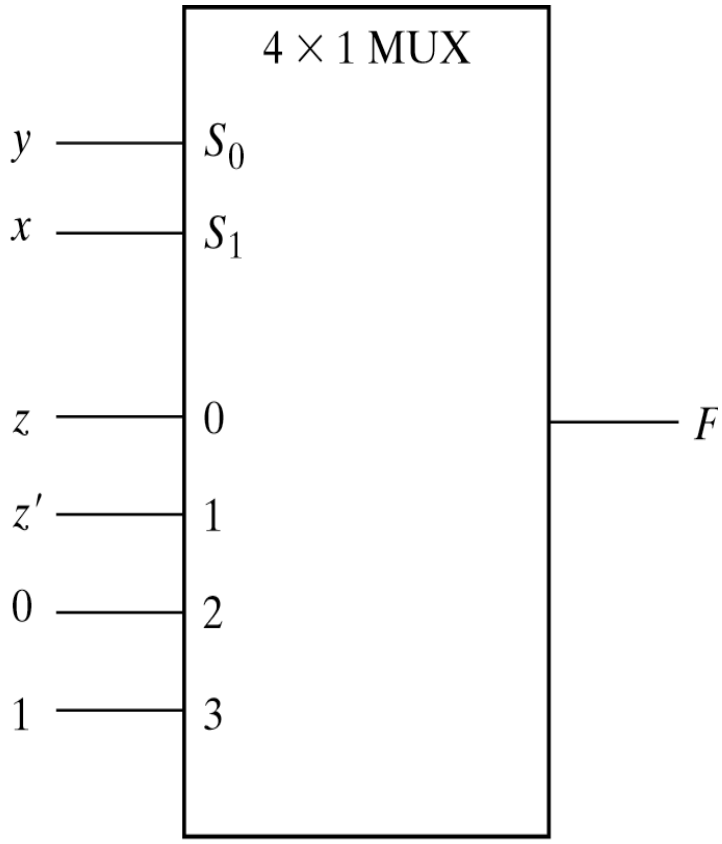
Boolean function implementation

- A more efficient method for implementing a Boolean function of n variables with a multiplexer that has n-1 selection inputs.

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

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

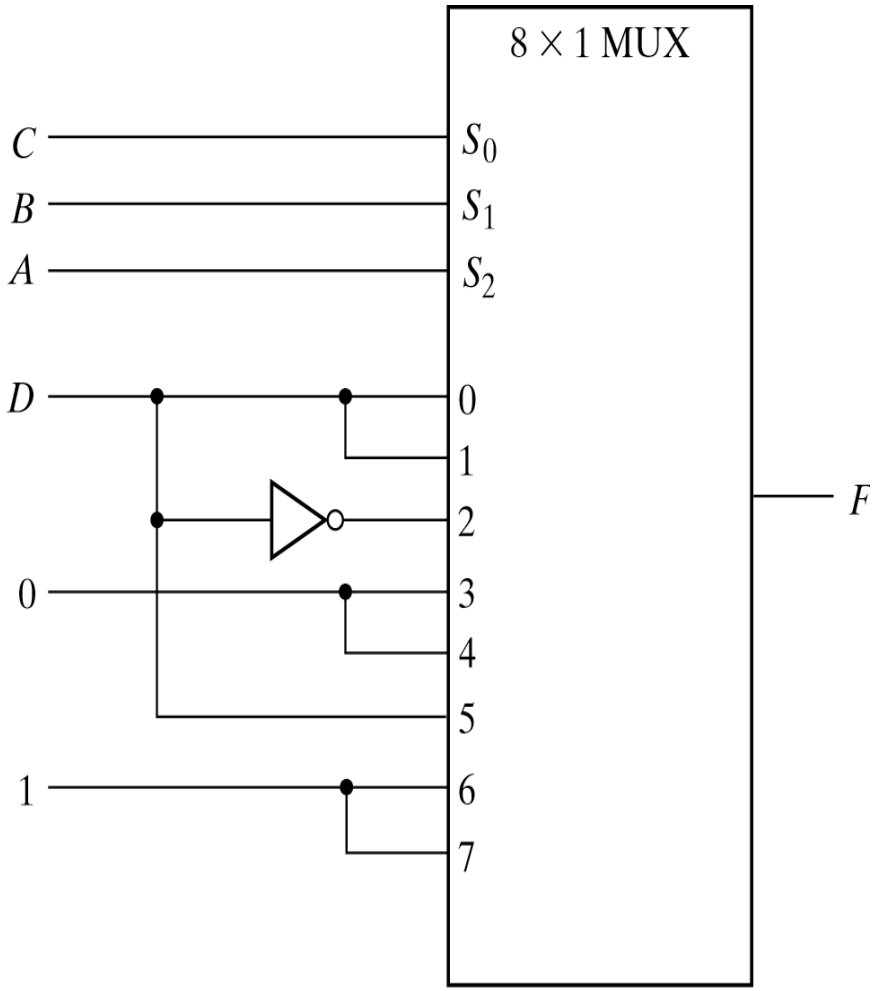


(b) Multiplexer implementation

4-input function with a multiplexer

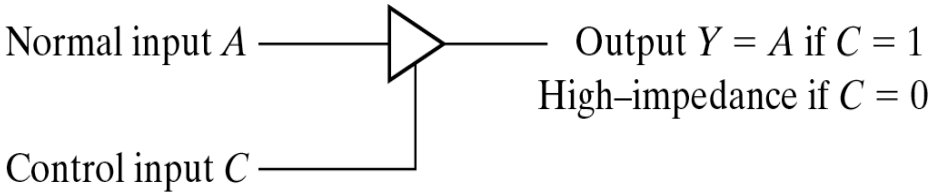
$F(A, B, C, D) = \Sigma(1, 3, 4, 11, 12, 13, 14, 15)$

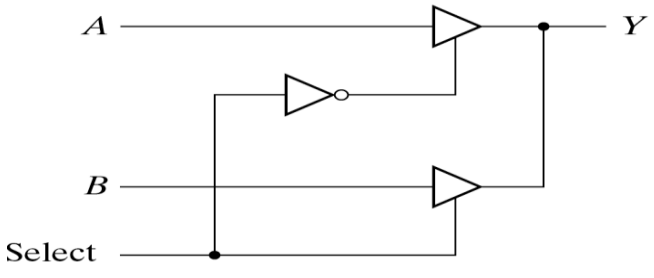
A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	



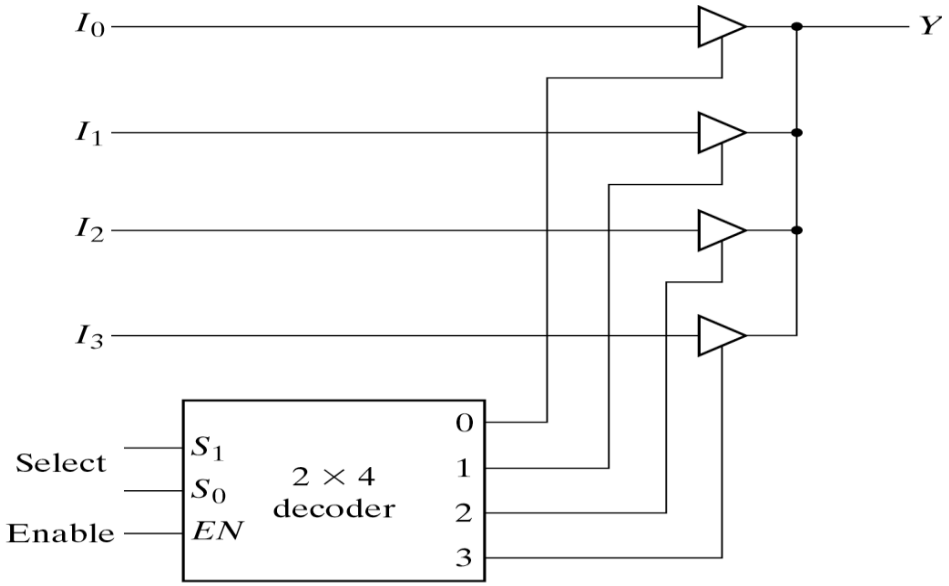
Multiplexer with Three-State Gates

- A multiplexer can be constructed with three-state gates.





(a) 2-to-1- line mux



(b) 4 - to - 1 line mux