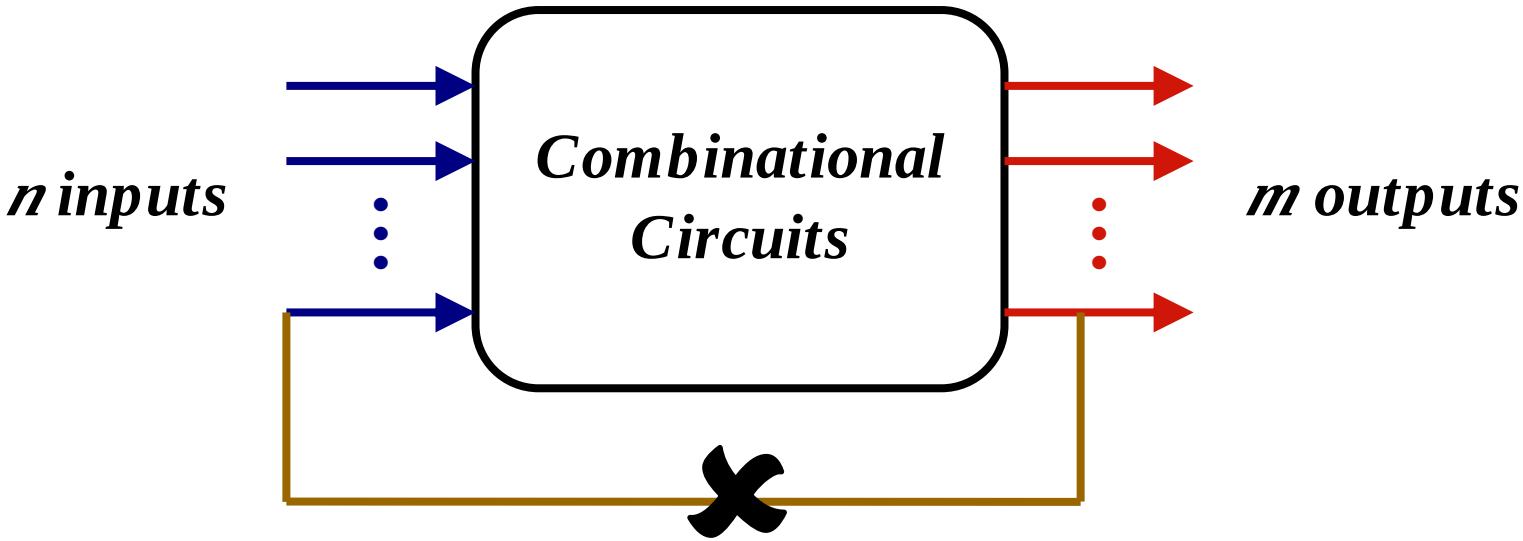
Digital Logic Design
Chapter 4

Combinational Logic

Combinational Circuits

- Output is function of input only (present combination of inputs).

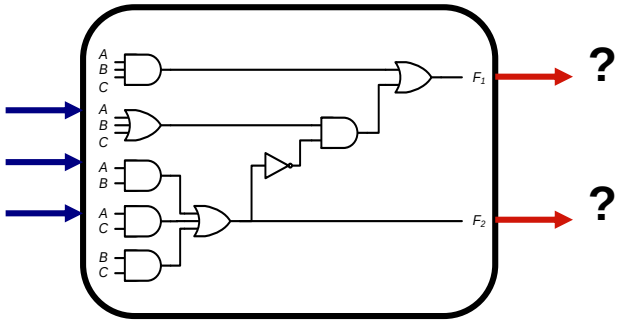
  i.e. no feedback



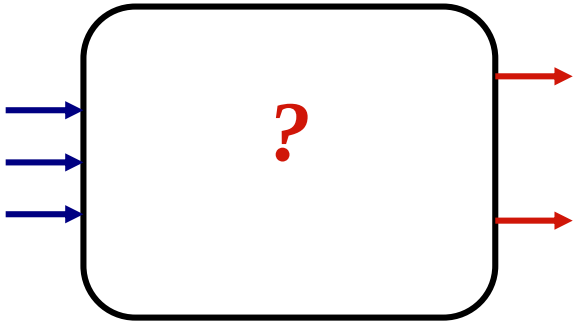
When input changes, output may change (after a delay)

Combinational Circuits

- Analysis
  - Given a circuit, find out its *function*
  - Function may be expressed as:
    - » Boolean function
    - » Truth table

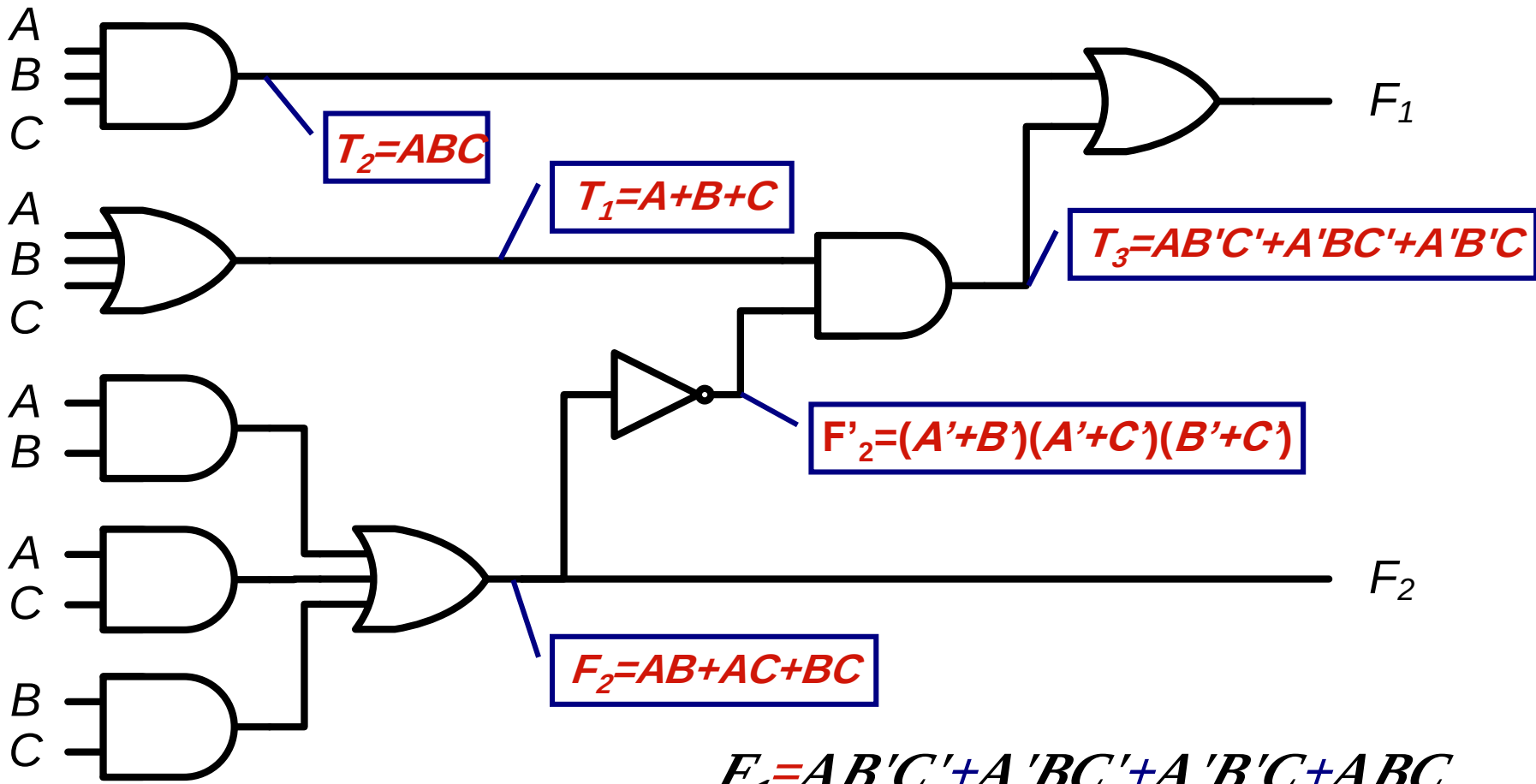


- Design
  - Given a desired function, determine its *circuit*
  - Function may be expressed as:
    - » Boolean function
    - » Truth table



Analysis Procedure

- Boolean Expression Approach

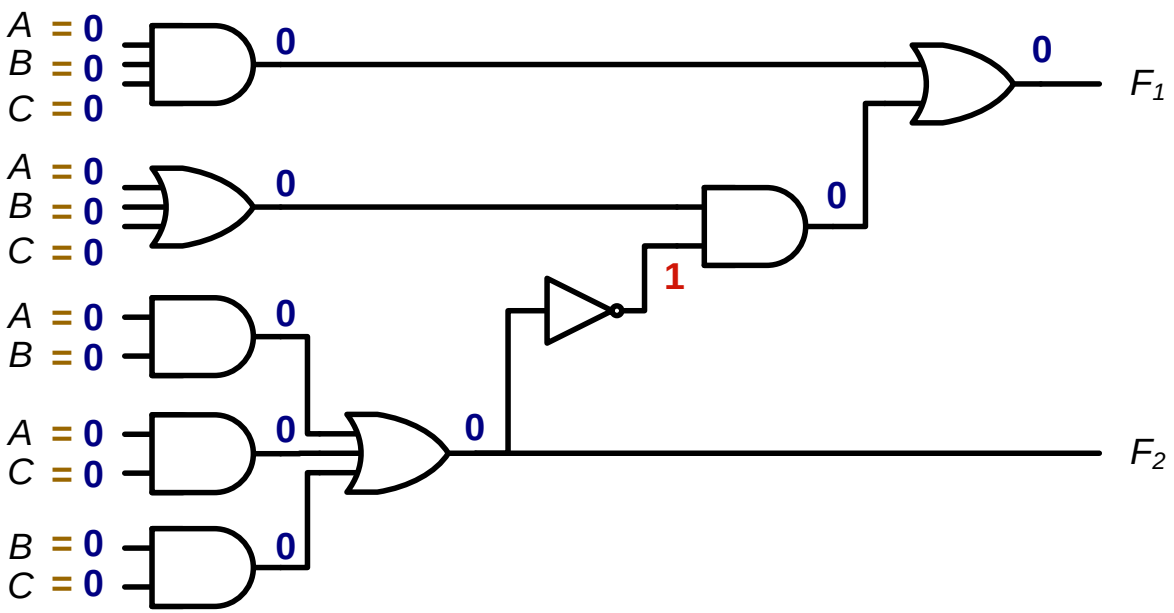


$$F_1 = AB'C' + A'BC' + A'B'C + ABC$$

$$F_2 = AB + AC + BC$$

Analysis Procedure

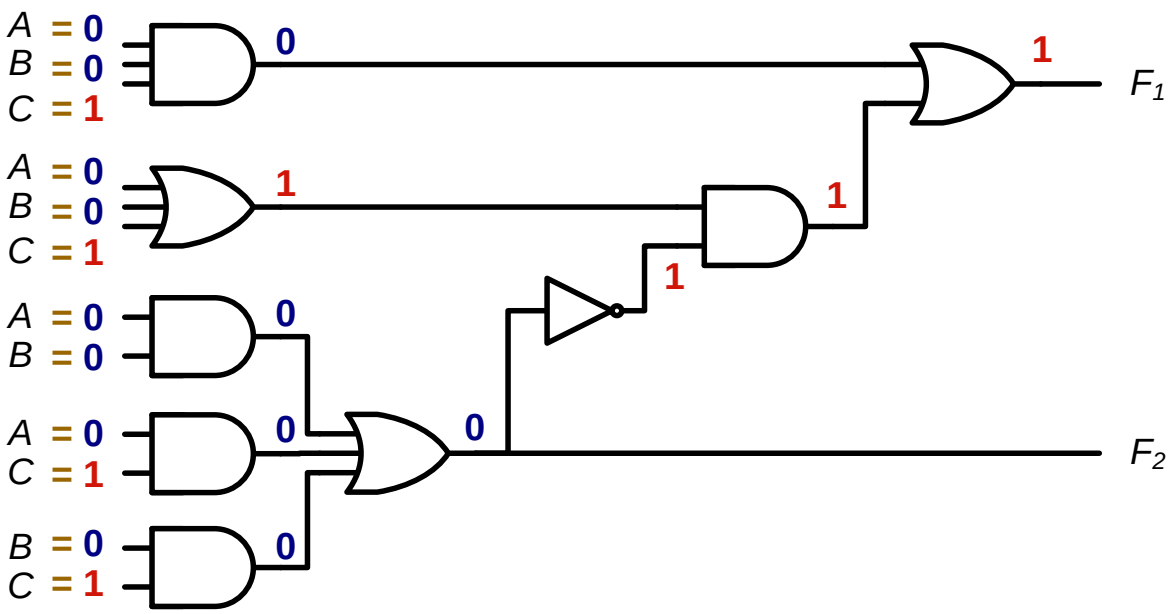
- Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0

Analysis Procedure

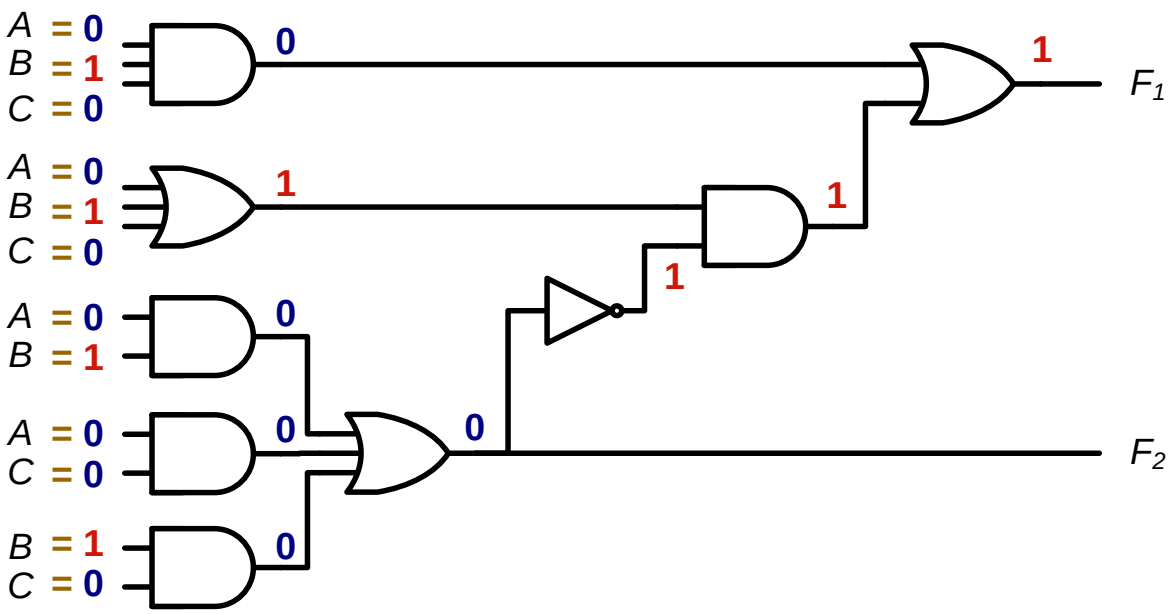
- Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0
0 0 1	1	0

Analysis Procedure

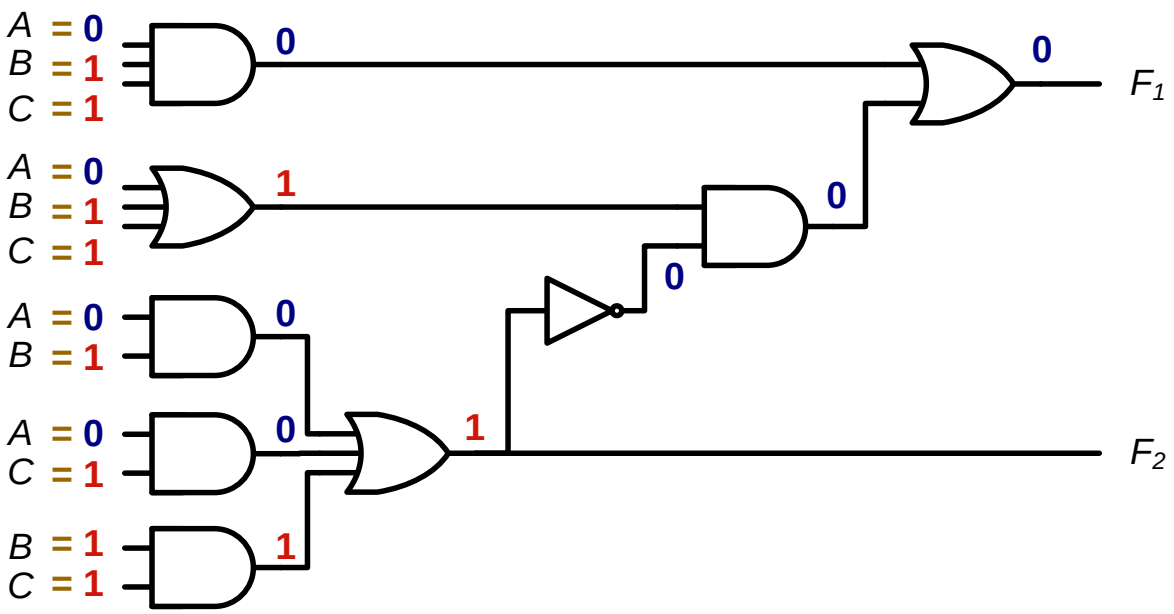
- Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0
0 0 1	1	0
0 1 0	1	0

Analysis Procedure

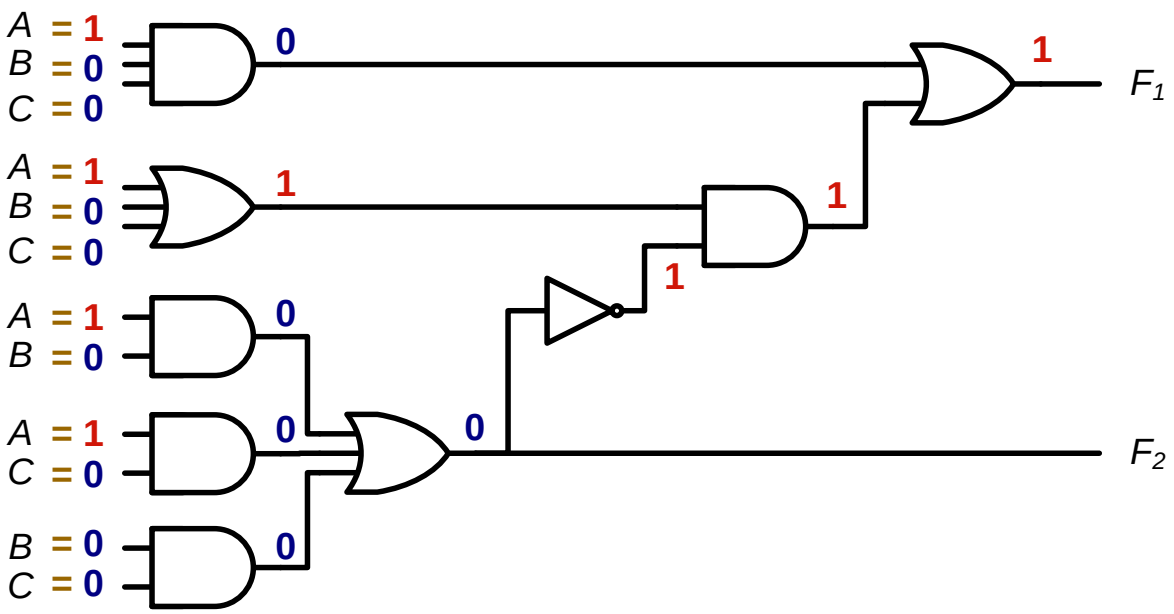
- Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0
0 0 1	1	0
0 1 0	1	0
0 1 1	0	1

Analysis Procedure

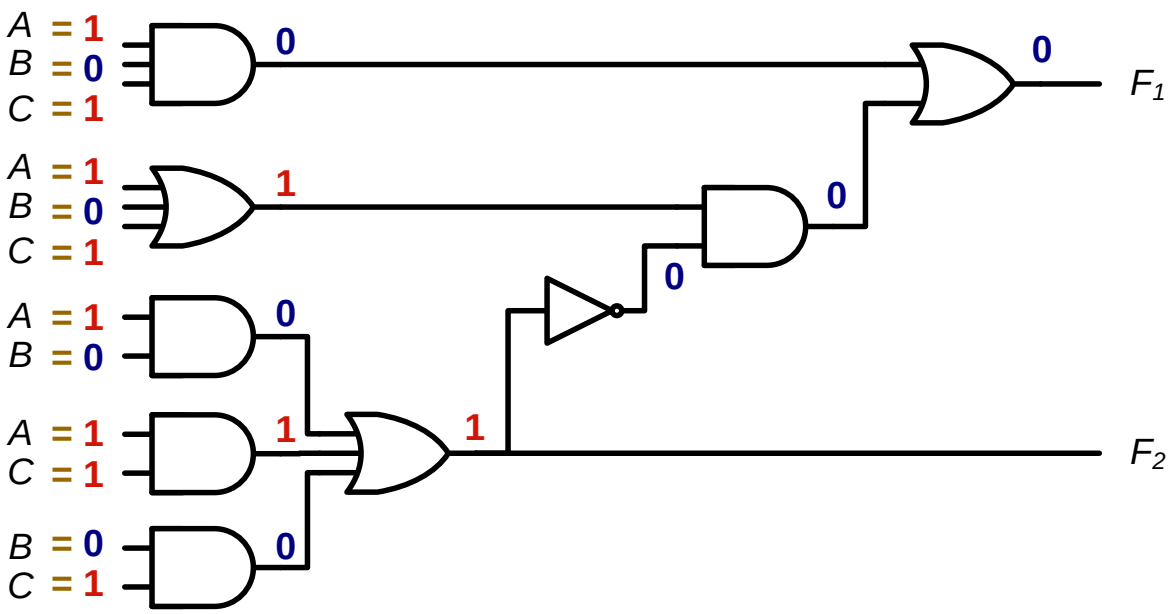
- Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0
0 0 1	1	0
0 1 0	1	0
0 1 1	0	1
1 0 0	1	0

Analysis Procedure

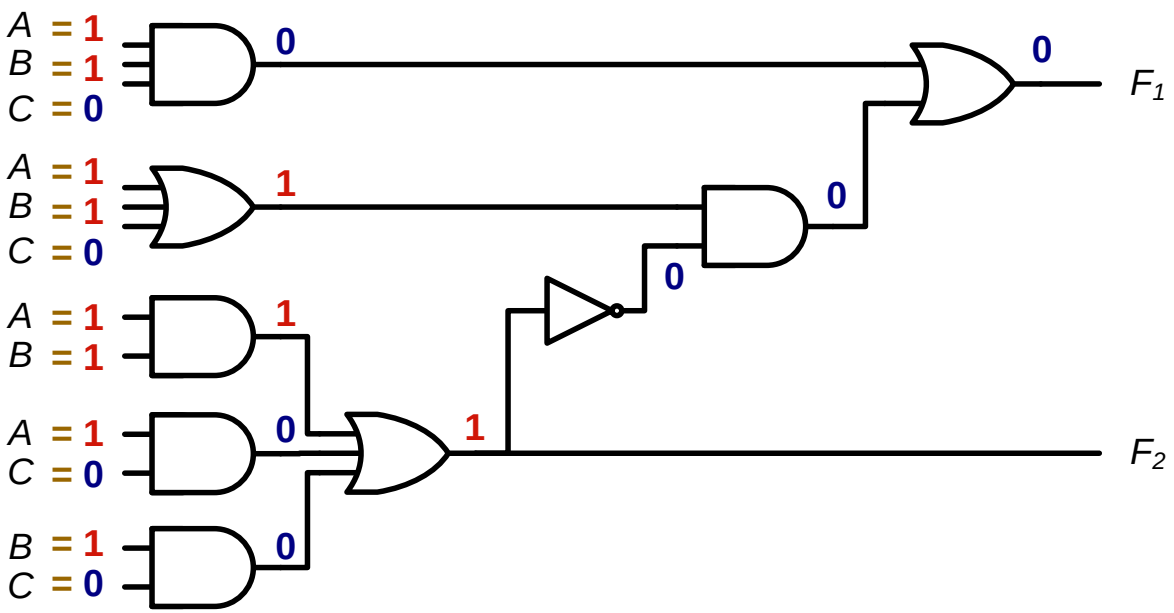
- Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0
0 0 1	1	0
0 1 0	1	0
0 1 1	0	1
1 0 0	1	0
1 0 1	0	1

Analysis Procedure

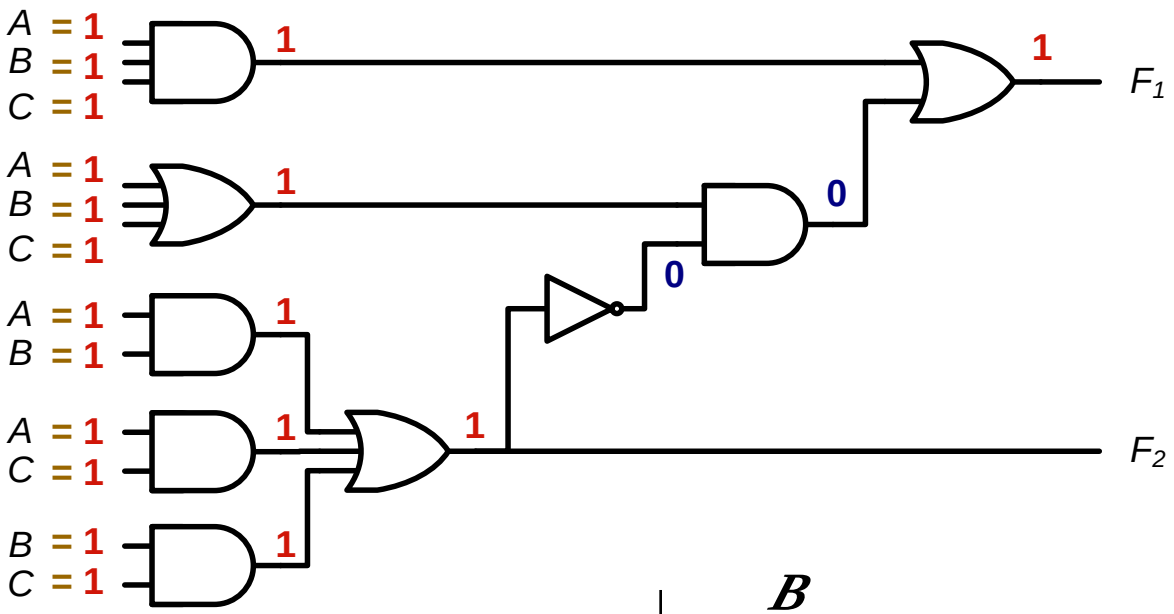
- Truth Table Approach



A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1

Analysis Procedure

- Truth Table Approach



A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A \ BC	00	01	11	10
0	0	1	0	1
1	1	0	1	0

A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$F_1 = AB'C' + A'BC' + A'B'C + ABC$

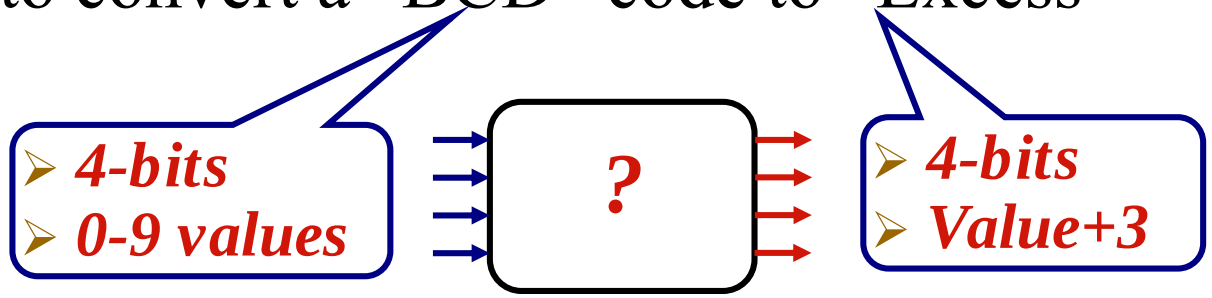
$F_2 = AB + AC + BC$

Design Procedure

- Given a problem statement:
  - Determine the number of *inputs* and *outputs*
  - Derive the truth table
  - Simplify the Boolean expression for each output
  - Produce the required circuit

Example:

Design a circuit to convert a “BCD” code to “Excess 3” code

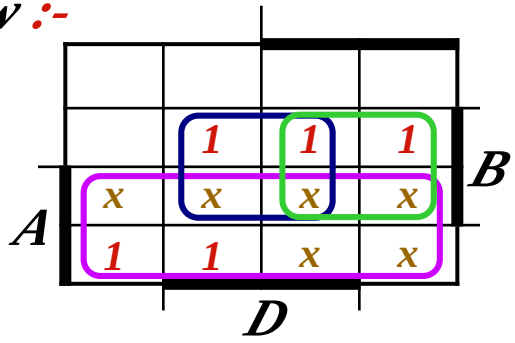


Design Procedure

- BCD-to-Excess 3 Converter

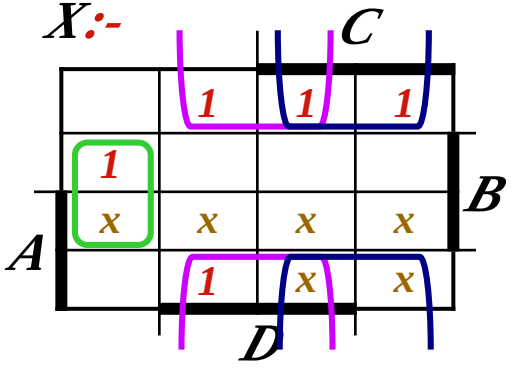
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

w :-



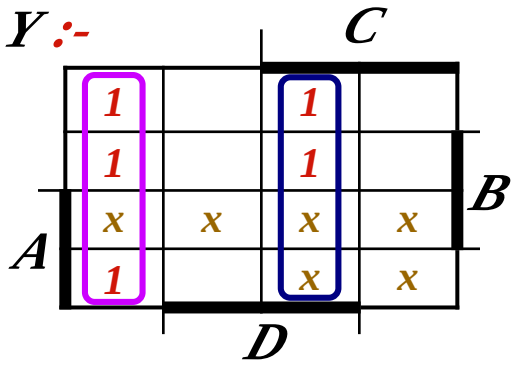
$w = A + BC + BD$

X:-



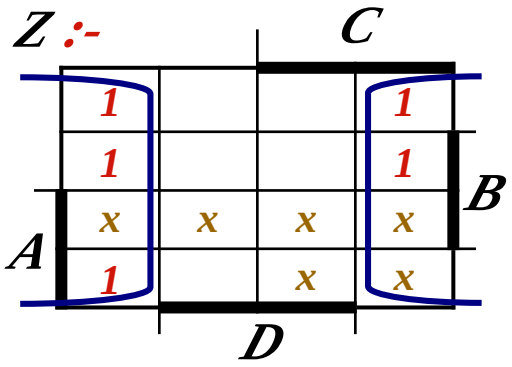
$x = B'C + B'D + BC'D'$

Y :-



$y = C'D' + CD$

Z :-

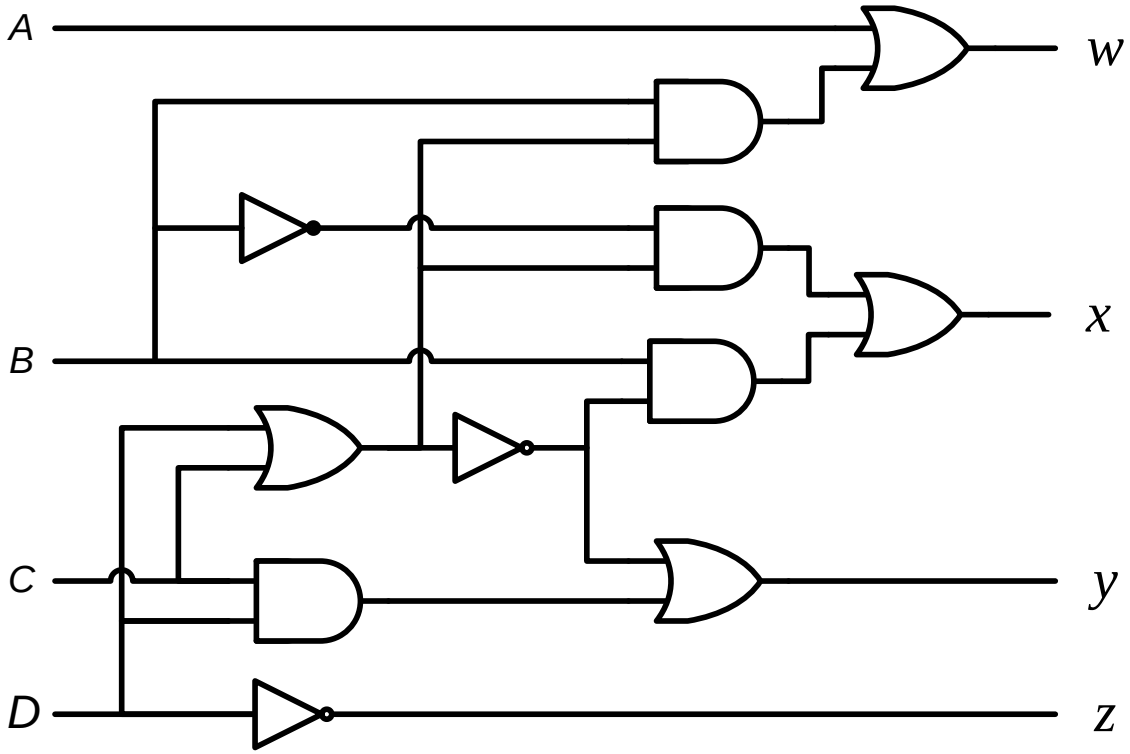


$z = D'$

Design Procedure

- BCD-to-Excess 3 Converter

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

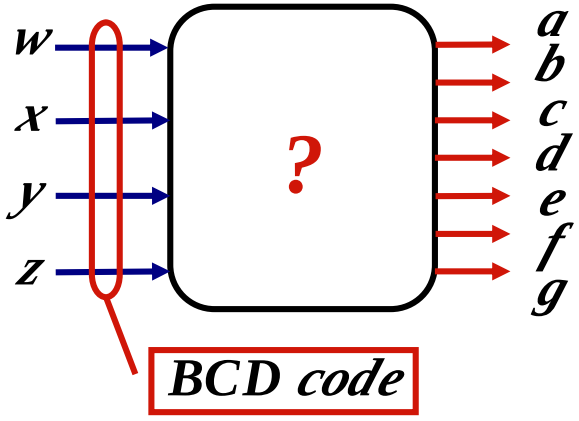


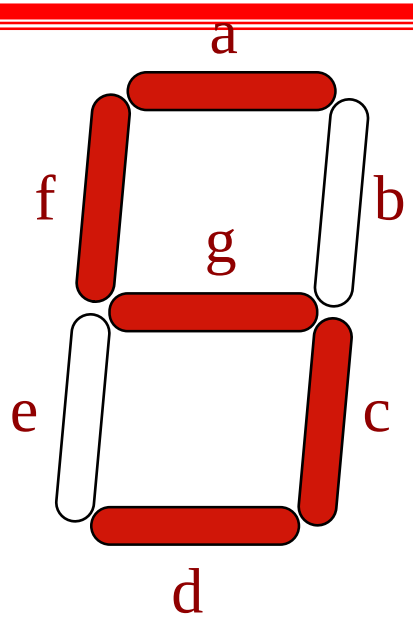
$w = A + B(C+D)$ $\quad y = (C+D)' + CD$

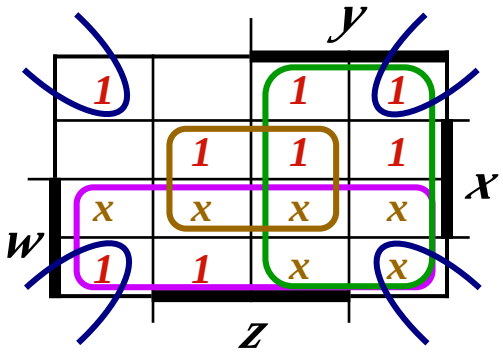
$x = B'(C+D) + B(C+D)'$ $\quad z = D'$

Seven-Segment Decoder

w x y z	a b c d e f g
0 0 0 0	1 1 1 1 1 1 0
0 0 0 1	0 1 1 0 0 0 0
0 0 1 0	1 1 0 1 1 0 1
0 0 1 1	1 1 1 1 0 0 1
0 1 0 0	0 1 1 0 0 1 1
0 1 0 1	1 0 1 1 0 1 1
0 1 1 0	1 0 1 1 1 1 1
0 1 1 1	1 1 1 0 0 0 0
1 0 0 0	1 1 1 1 1 1 1
1 0 0 1	1 1 1 1 0 1 1
1 0 1 0	x x x x x x x
1 0 1 1	x x x x x x x
1 1 0 0	x x x x x x x
1 1 0 1	x x x x x x x
1 1 1 0	x x x x x x x
1 1 1 1	x x x x x x x







$a = w + y + xz + x'z'$

$b = \ldots$

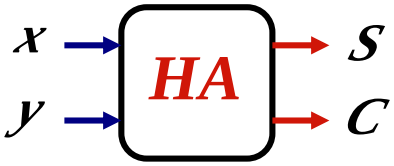
$c = \ldots$

$d = \ldots$

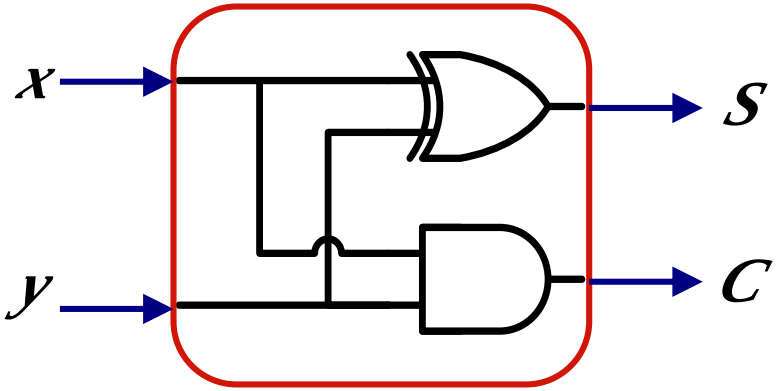
Binary Adder

- Half Adder
  - Adds 1-bit plus 1-bit
  - Produces Sum and Carry

$$\begin{array}{rr} & x \\ + & y \\ \hline C & S \end{array}$$



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

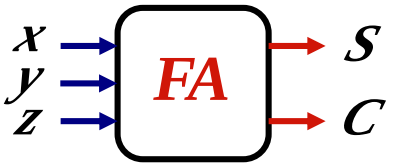


Binary Adder

- Full Adder
  - Adds 1-bit plus 1-bit plus 1-bit (carry from the previous position)
  - Produces Sum and Carry

$$\begin{array}{r} x \\ + \quad y \\ + \quad z \\ \hline C \quad S \end{array}$$

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



x \ yz	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$S = xy'z' + x'yz' + x'y'z + xyz = x \oplus y \oplus z$$

x \ yz	00	01	11	10
0	0	0	1	0
1	0	1	1	1

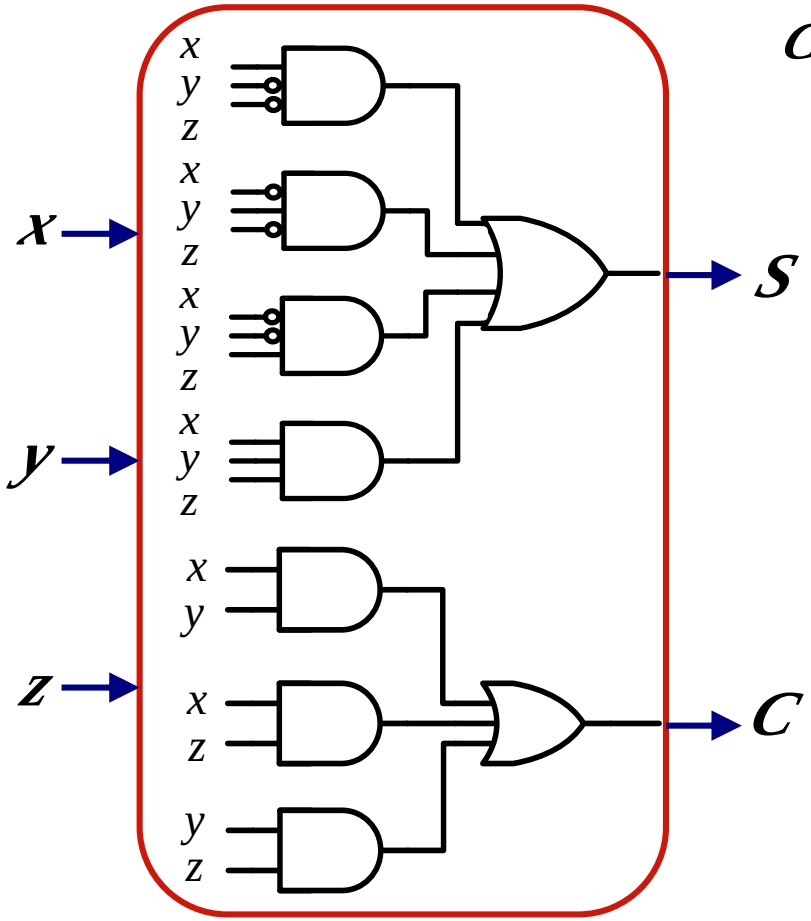
$$C = xy + xz + yz$$

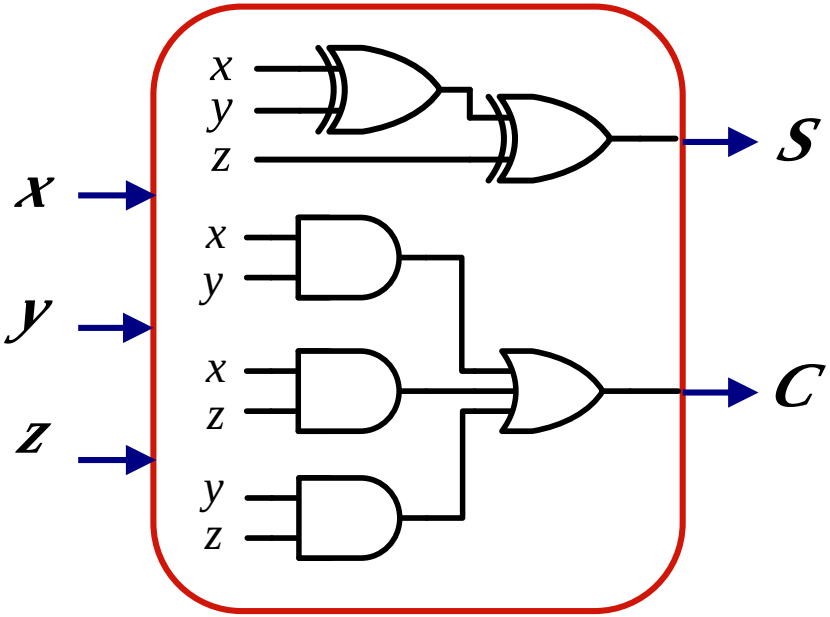
Binary Adder

- Full Adder

$S = xy'z' + x'yz' + x'y'z + xyz = x \oplus y \oplus z$

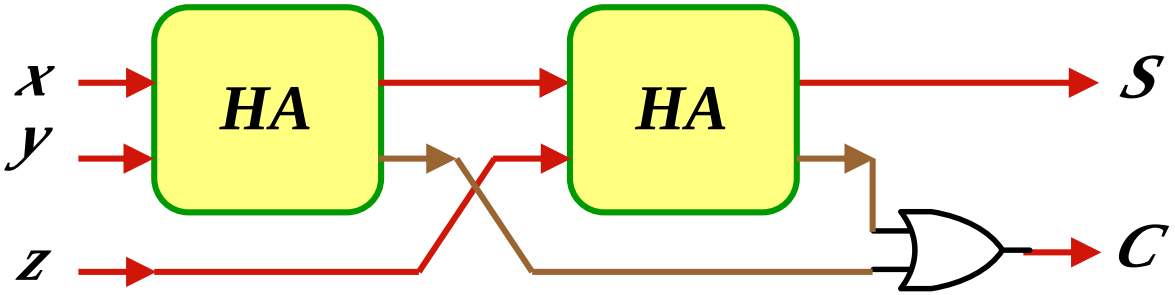
$C = xy + xz + yz$

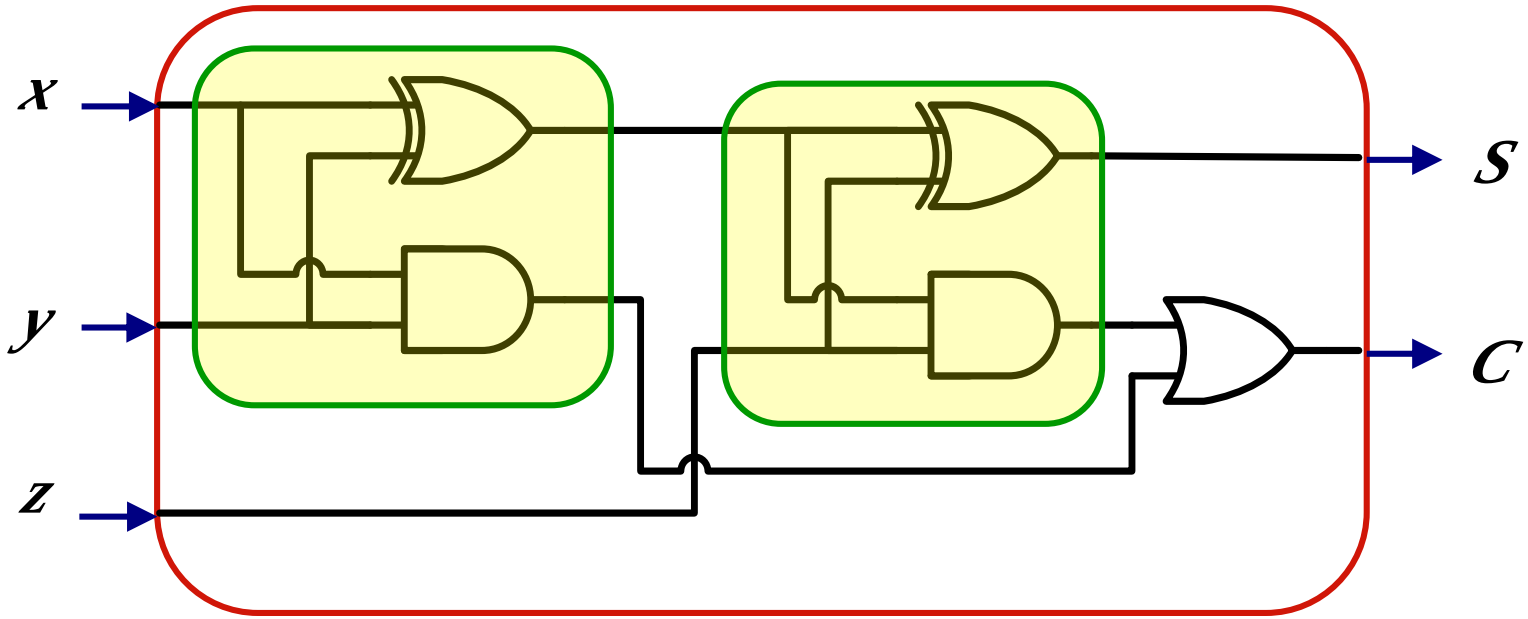




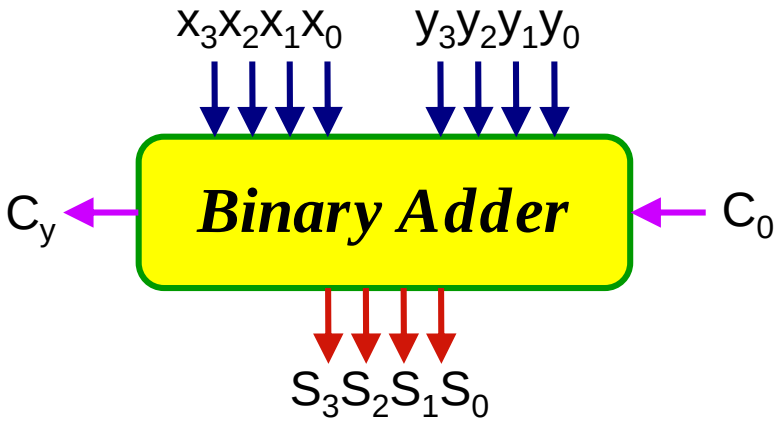
Binary Adder

- Full Adder = Two half adder + OR gate



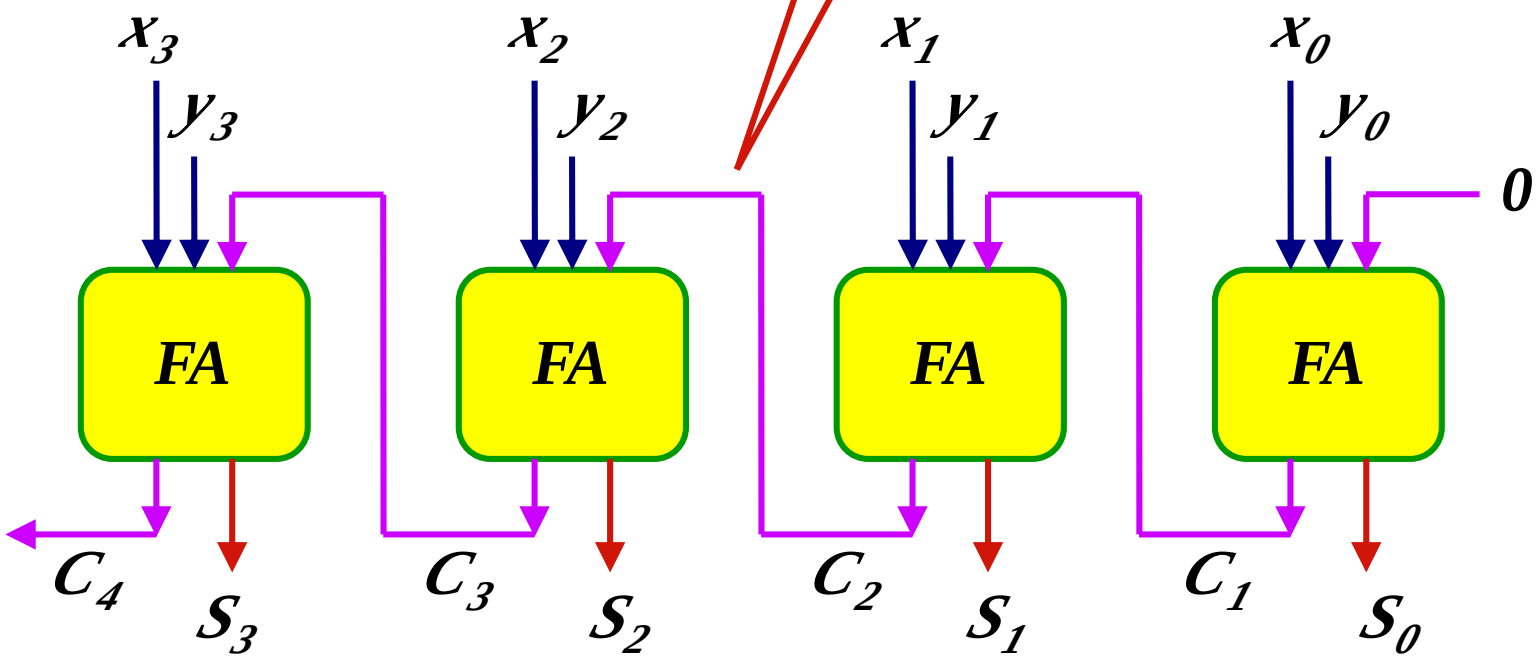


Binary Adder



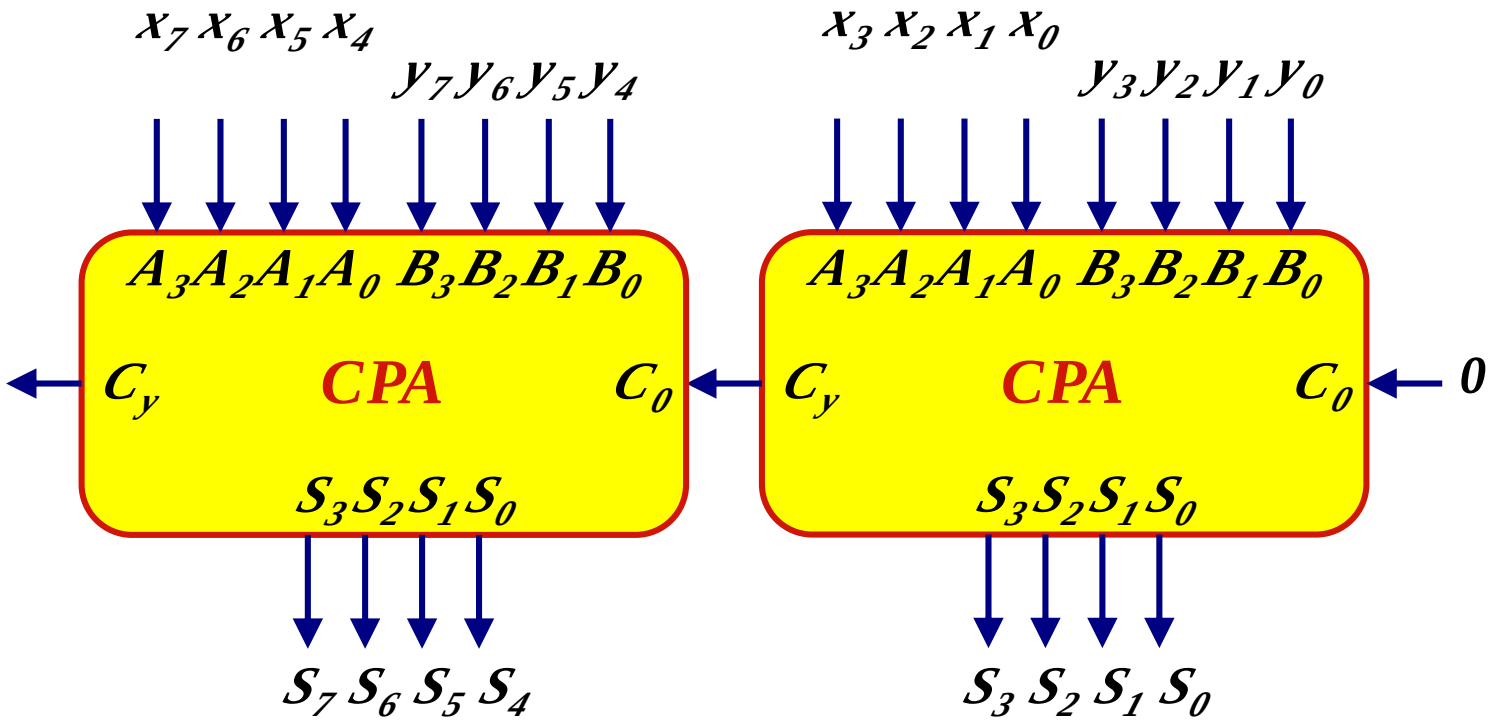
Carry Propagate Addition

$$\begin{array}{llllll} & c_3 & c_2 & c_1 & & \\ + & x_3 & x_2 & x_1 & x_0 & \\ + & y_3 & y_2 & y_1 & y_0 & \\ \hline Cy & S_3 & S_2 & S_1 & S_0 & \end{array}$$



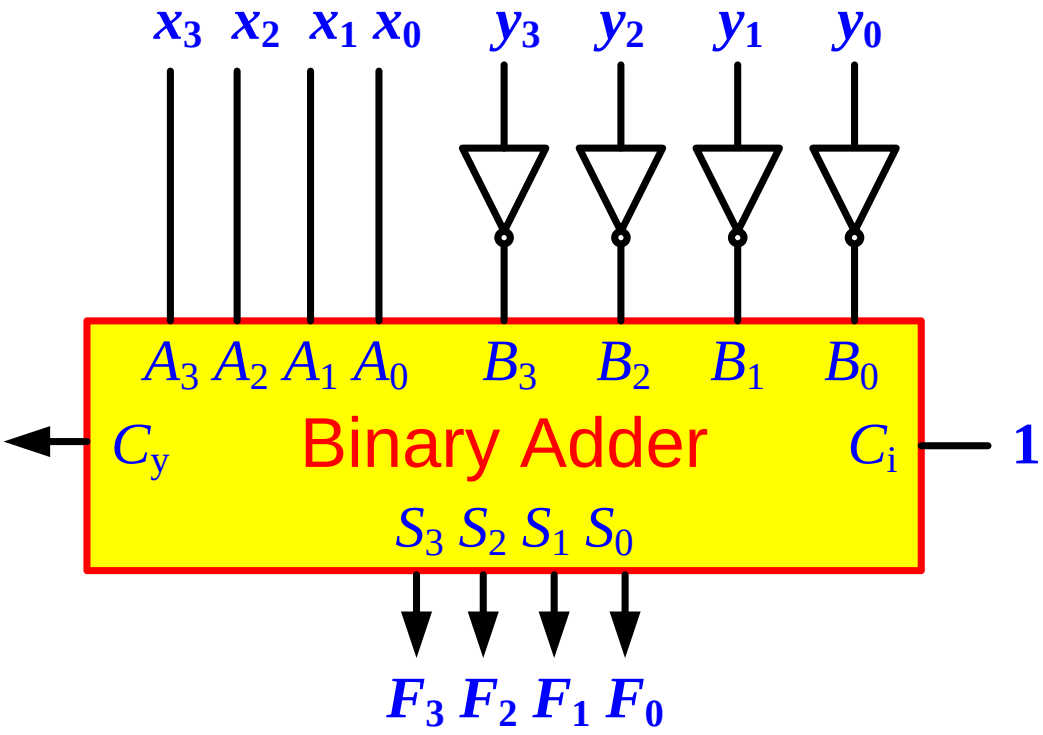
Binary Adder

- Carry Propagate Adders



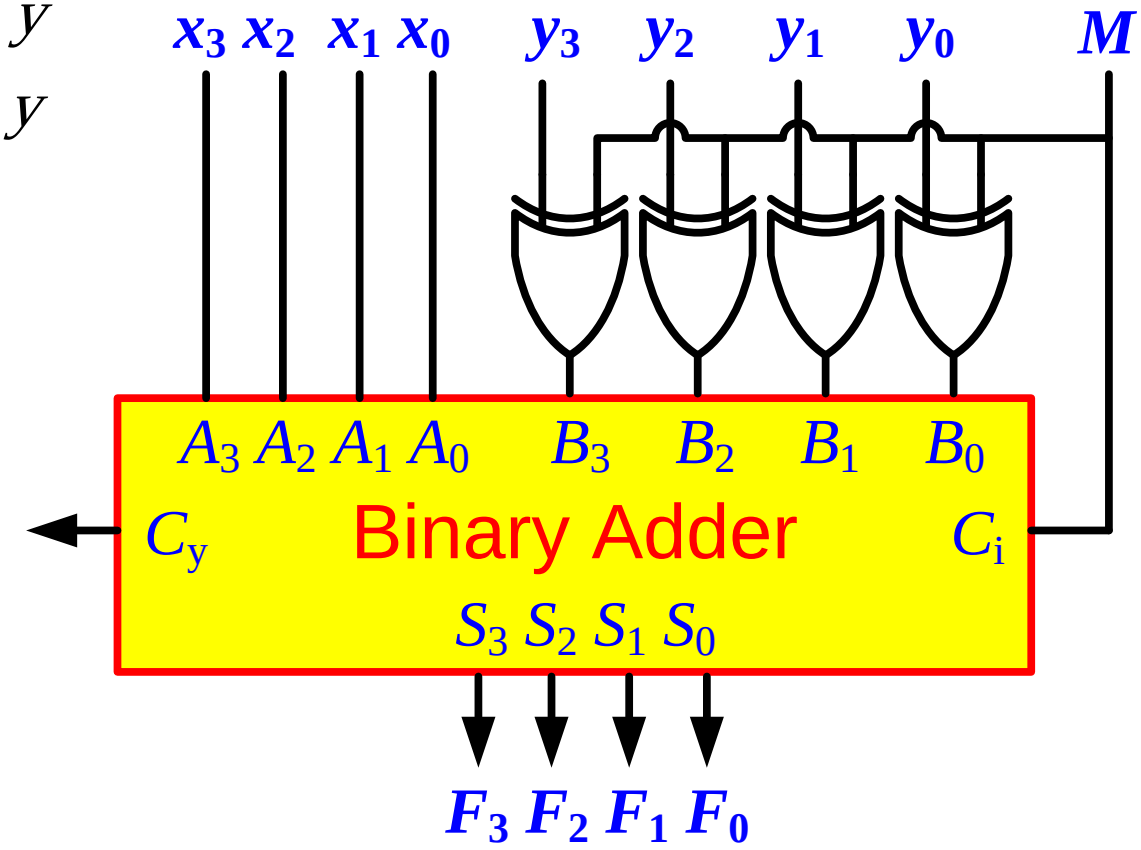
Binary Subtractor

- Use 2's complement with binary adder
  - $x - y = x + (-y) = x + y' + 1$



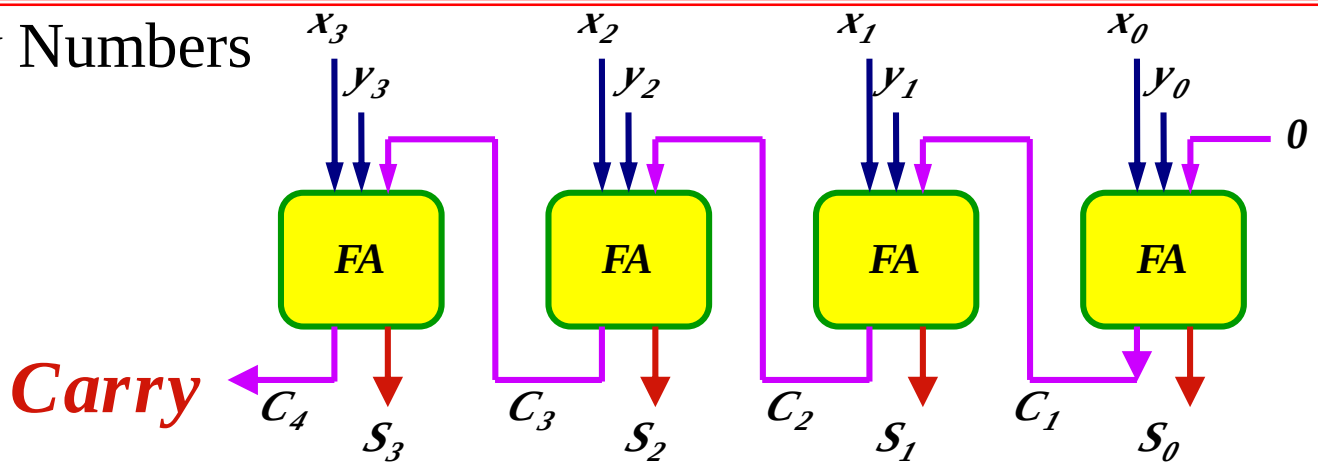
Binary Adder/Subtractor

- *M*: Control Signal (Mode)
  - *M*=0 ➔ $F = x + y$
  - *M*=1 ➔ $F = x - y$

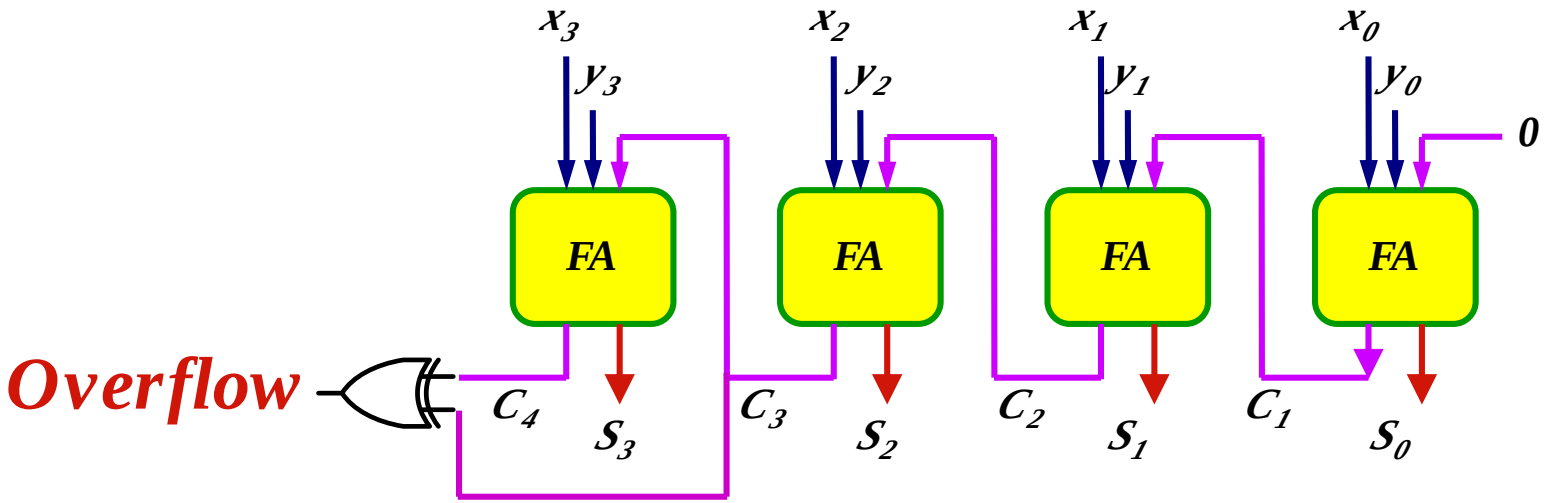


Carry and Overflow

- Unsigned Binary Numbers



- Signed Numbers: 2's Complement Numbers (overflow may occur if the two numbers added are ***both*** positive or negative.)



BCD Adder

- 4-bits plus 4-bits
- Operands and Result: 0 to 9

$$\begin{array}{r} + \; x_3\; x_2\; x_1\; x_0 \\ + \; y_3\; y_2\; y_1\; y_0 \\ \hline Cy\;\; S_3\; S_2\; S_1\; S_0 \end{array}$$

X +Y	$x_3 x_2 x_1 x_0$	$y_3 y_2 y_1 y_0$	Sum	Cy	$S_3 S_2 S_1 S_0$
0 + 0	0 0 0 0	0 0 0 0	= 0	0	0 0 0 0
0 + 1	0 0 0 0	0 0 0 1	= 1	0	0 0 0 1
0 + 2	0 0 0 0	0 0 1 0	= 2	0	0 0 1 0
0 + 9	0 0 0 0	1 0 0 1	= 9	0	1 0 0 1
1 + 0	0 0 0 1	0 0 0 0	= 1	0	0 0 0 1
1 + 1	0 0 0 1	0 0 0 1	= 2	0	0 0 1 0
1 + 8	0 0 0 1	1 0 0 0	= 9	0	1 0 0 1
1 + 9	0 0 0 1	1 0 0 1	= A	0	1 0 1 0
2 + 0	0 0 1 0	0 0 0 0	= 2	0	0 0 1 0
9 + 9	1 0 0 1	1 0 0 1	= 18	1	0 0 1 0

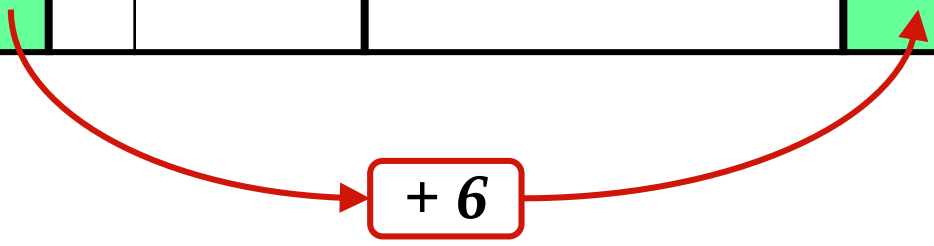
Invalid Code

Wrong BCD Value

0001 1000

BCD Adder

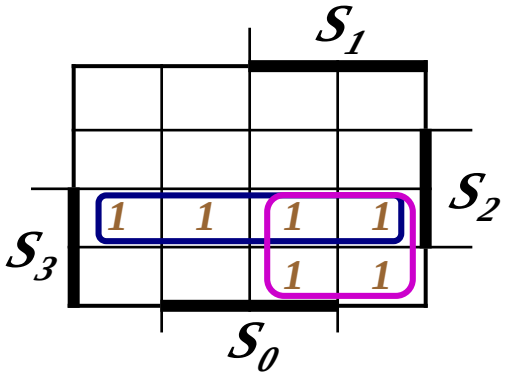
X +Y	$x_3 x_2 x_1 x_0$	$y_3 y_2 y_1 y_0$	Sum	Cy	$S_3 S_2 S_1 S_0$	Required BCD Output	Value
9 + 0	1 0 0 1	0 0 0 0	= 9	0	1 0 0 1	0 0 0 0 1 0 0 1	= 9
9 + 1	1 0 0 1	0 0 0 1	= 10	0	1 0 1 0	0 0 0 1 0 0 0 0	= 16
9 + 2	1 0 0 1	0 0 1 0	= 11	0	1 0 1 1	0 0 0 1 0 0 0 1	= 17
9 + 3	1 0 0 1	0 0 1 1	= 12	0	1 1 0 0	0 0 0 1 0 0 1 0	= 18
9 + 4	1 0 0 1	0 1 0 0	= 13	0	1 1 0 1	0 0 0 1 0 0 1 1	= 19
9 + 5	1 0 0 1	0 1 0 1	= 14	0	1 1 1 0	0 0 0 1 0 1 0 0	= 20
9 + 6	1 0 0 1	0 1 1 0	= 15	0	1 1 1 1	0 0 0 1 0 1 0 1	= 21
9 + 7	1 0 0 1	0 1 1 1	= 16	1	0 0 0 0	0 0 0 1 0 1 1 0	= 22
9 + 8	1 0 0 1	1 0 0 0	= 17	1	0 0 0 1	0 0 0 1 0 1 1 1	= 23
9 + 9	1 0 0 1	1 0 0 1	= 18	1	0 0 1 0	0 0 0 1 1 0 0 0	= 24



BCD Adder

- Correct Binary Adder's Output (+6)
  - If the result is between 'A' and 'F'
  - If Cy = 1 (result grater than 'F')

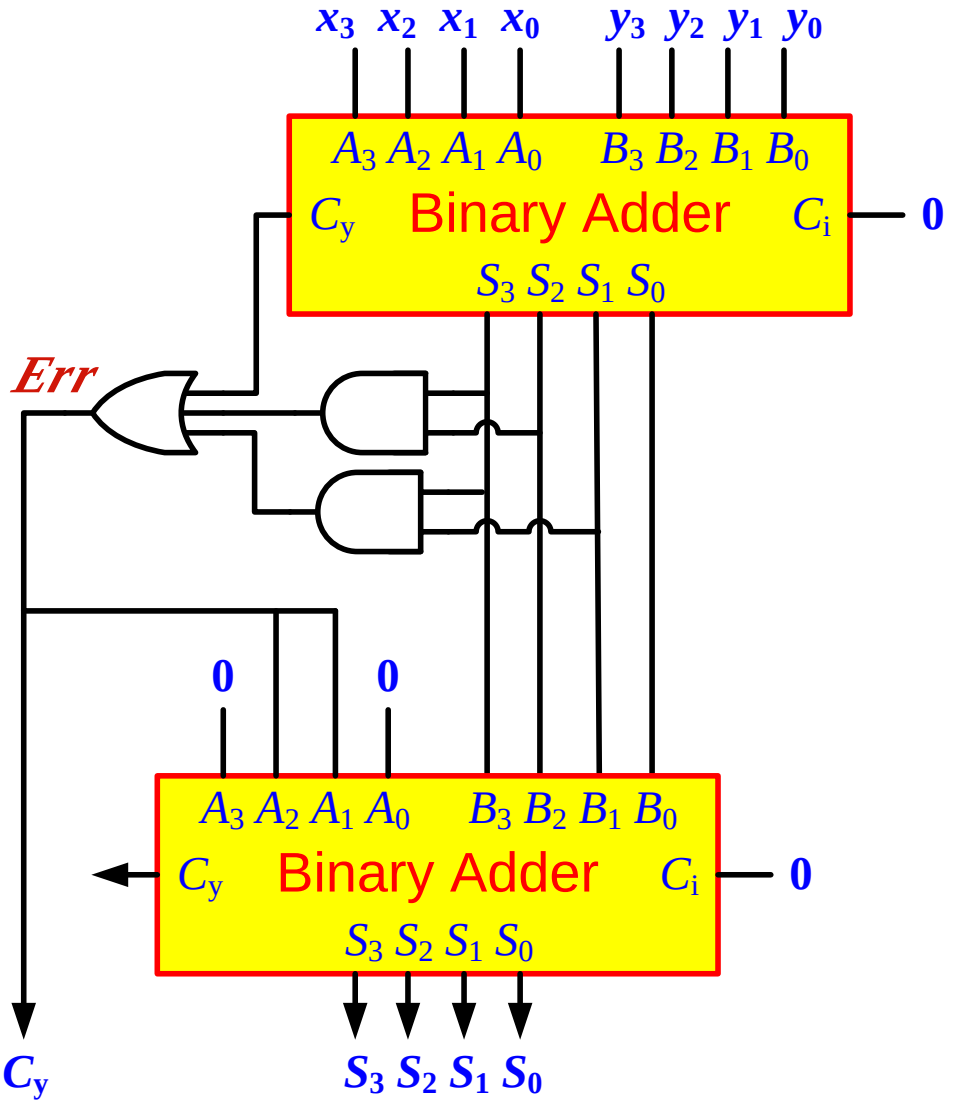
$S_3\ S_2\ S_1\ S_0$	Err
0 0 0 0	0
1 0 0 0	0
1 0 0 1	0
1 0 1 0	1
1 0 1 1	1
1 1 0 0	1
1 1 0 1	1
1 1 1 0	1
1 1 1 1	1



$$Err = S_3 S_2 + S_3 S_1 + C_y$$

BCD Adder

$$Err = S_3 S_2 + S_3 S_1 + C_y$$



Home Work (7)

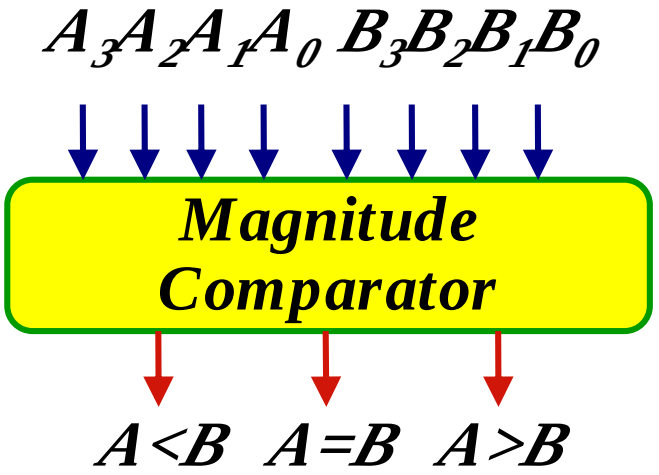
Digital Design (**4th**)- Morris Mano-Page 175

-Problems:

- 4.4
- 4.5
- 4.6 (a)
- 4.7 (a)
- 4. 10
- 4.12
- 4.22

Magnitude Comparator

- Compare 4-bit number to 4-bit number
  - 3 Outputs: < , = , >



$$x_3 = \overline{A}_3\,\overline{B}_3 + A_3\,B_3$$

$$x_2 = \overline{A}_2\,\overline{B}_2 + A_2\,B_2$$

$$x_1 = \overline{A}_1\,\overline{B}_1 + A_1\,B_1$$

$$x_0 = \overline{A}_0\,\overline{B}_0 + A_0\,B_0$$

$$(A = B) = x_3\,x_2\,x_1\,x_0$$

$$(A > B) = A_3\,\overline{B}_3 + x_3\,A_2\,\overline{B}_2 + x_3\,x_2\,A_1\,\overline{B}_1 + x_3\,x_2\,x_1\,A_0\,\overline{B}_0$$

$$(A < B) = \overline{A}_3\,B_3 + x_3\,\overline{A}_2\,B_2 + x_3\,x_2\,\overline{A}_1\,B_1 + x_3\,x_2\,x_1\,\overline{A}_0\,B_0$$

Magnitude Comparator

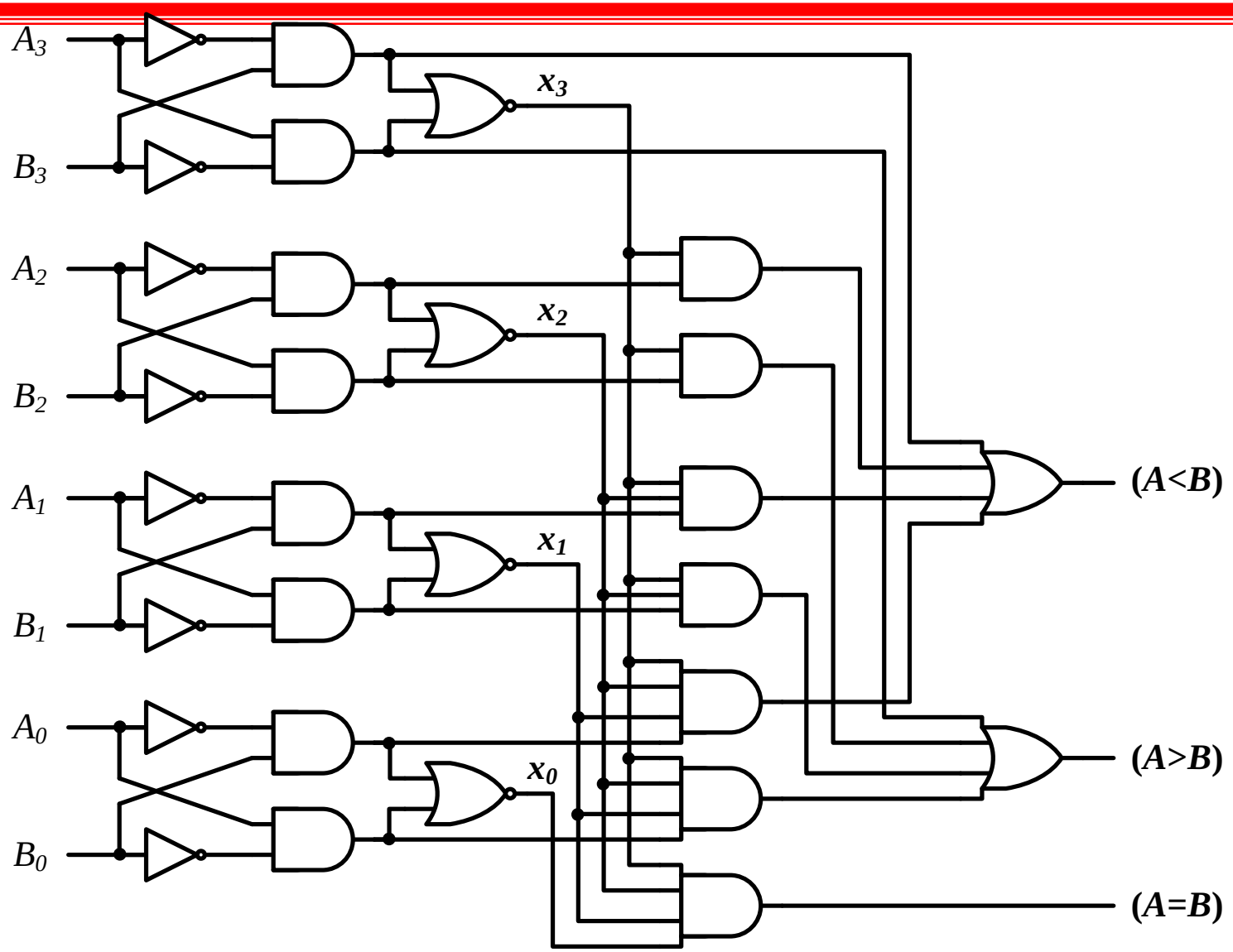
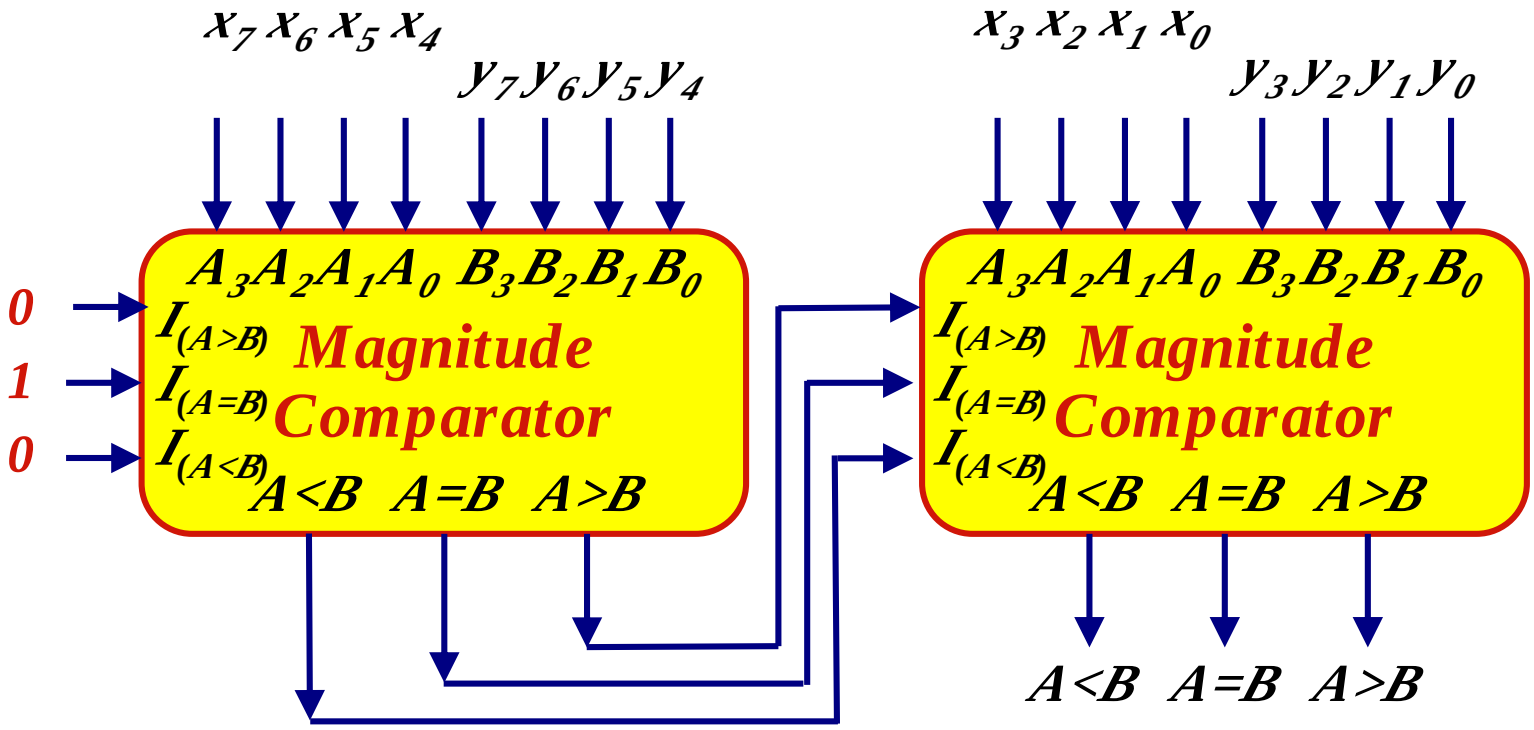


Fig 4.17 : Four-bit magnitude Comparator

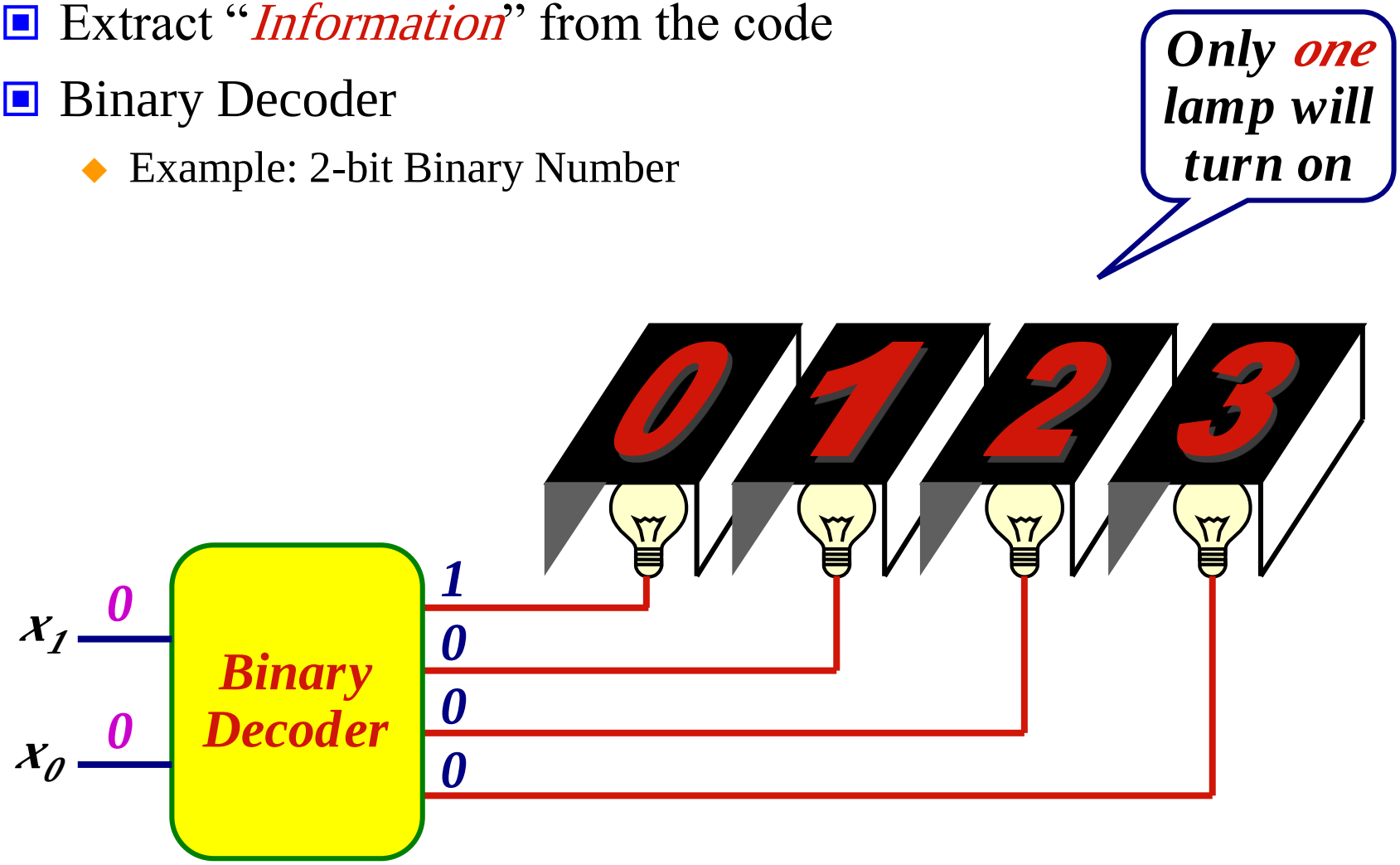
Magnitude Comparator

- Expandable to more number of bits



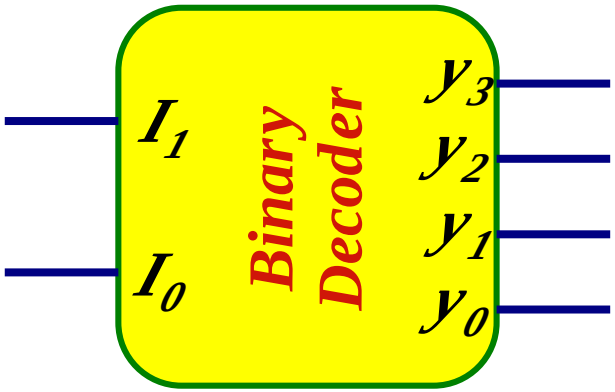
Decoders

- Extract "*Information*" from the code
- Binary Decoder
  - Example: 2-bit Binary Number

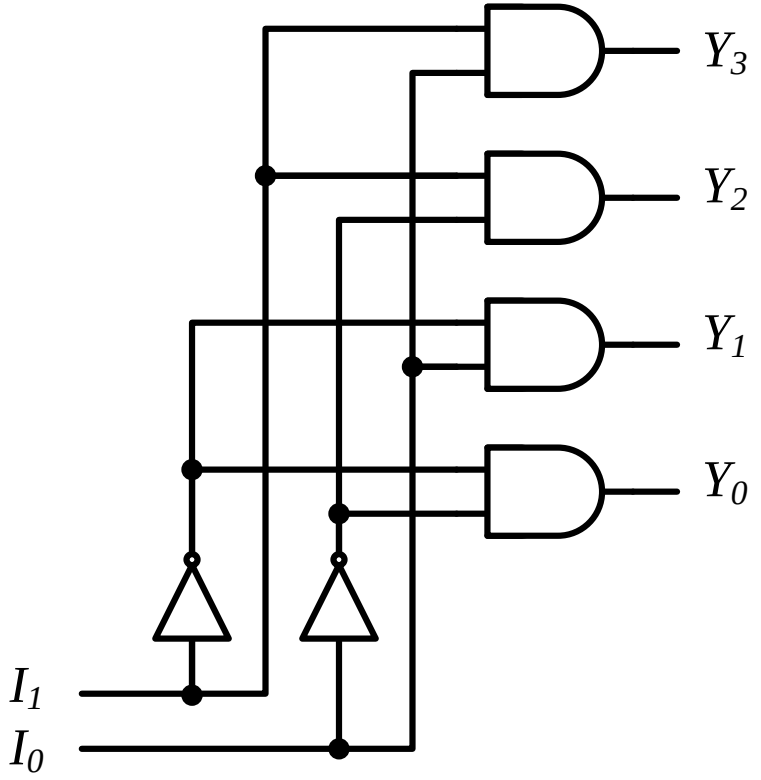


Decoders

- 2-to-4 Line Decoder



I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

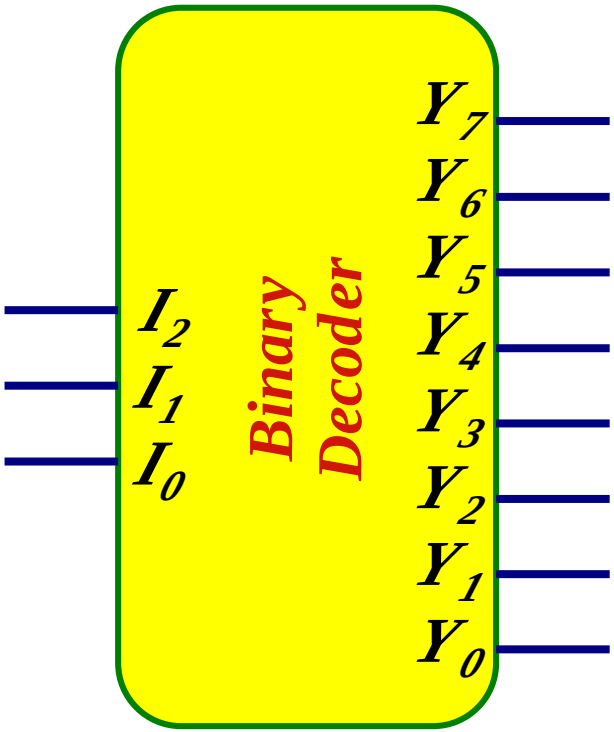


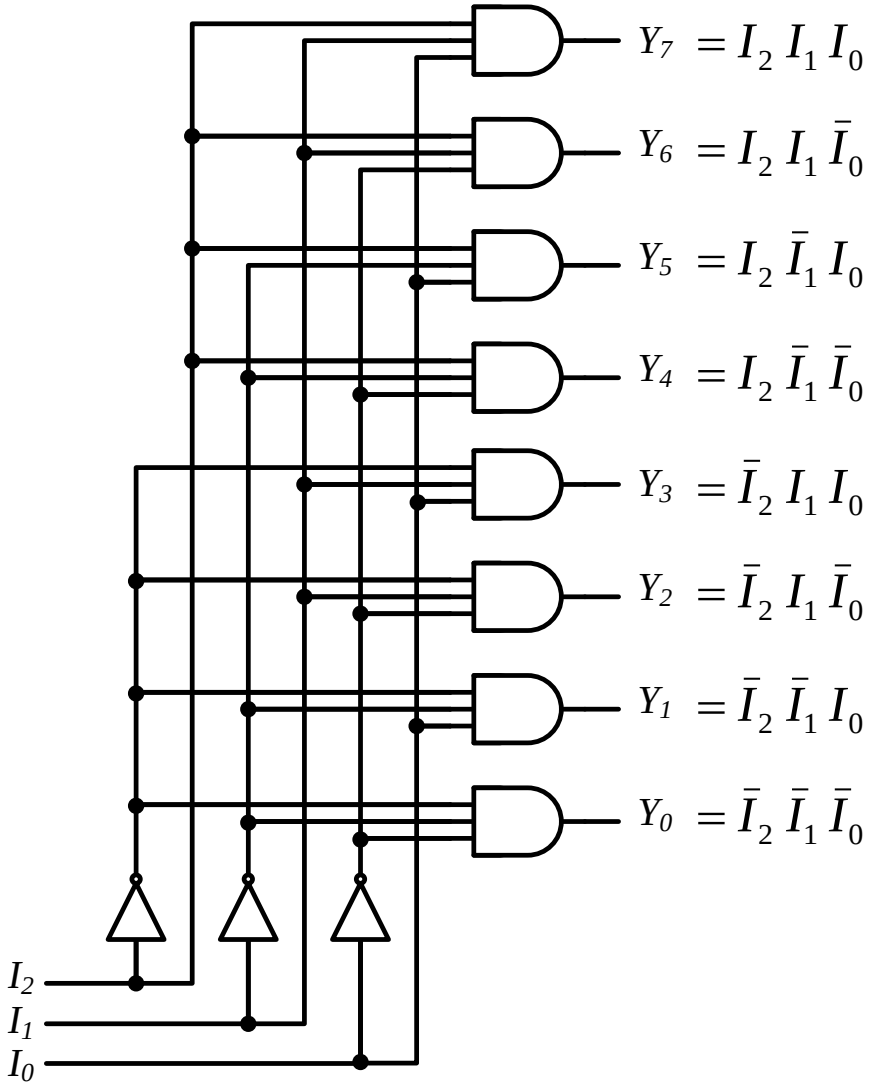
$$Y_3 = I_1 I_0 \qquad Y_2 = I_1 \bar{I}_0$$

$$Y_1 = \bar{I}_1 I_0 \qquad Y_0 = \bar{I}_1 \bar{I}_0$$

Decoders

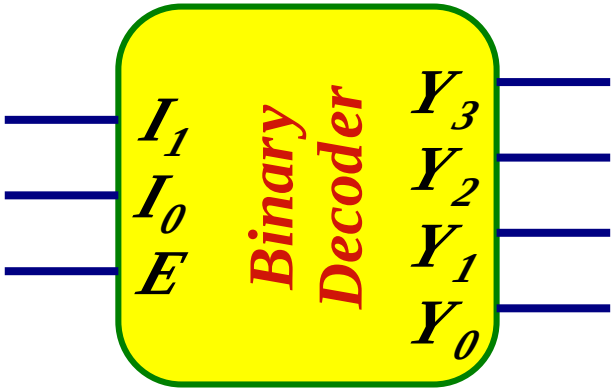
- 3-to-8 Line Decoder



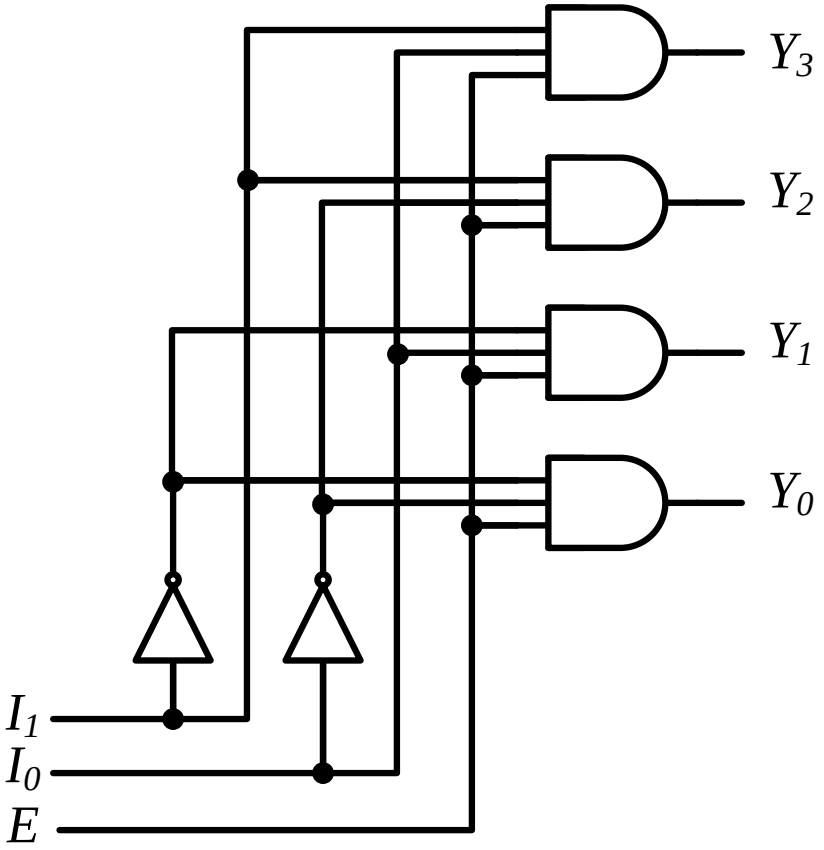


Decoders

- "*Enable*" input used to control the circuit operation



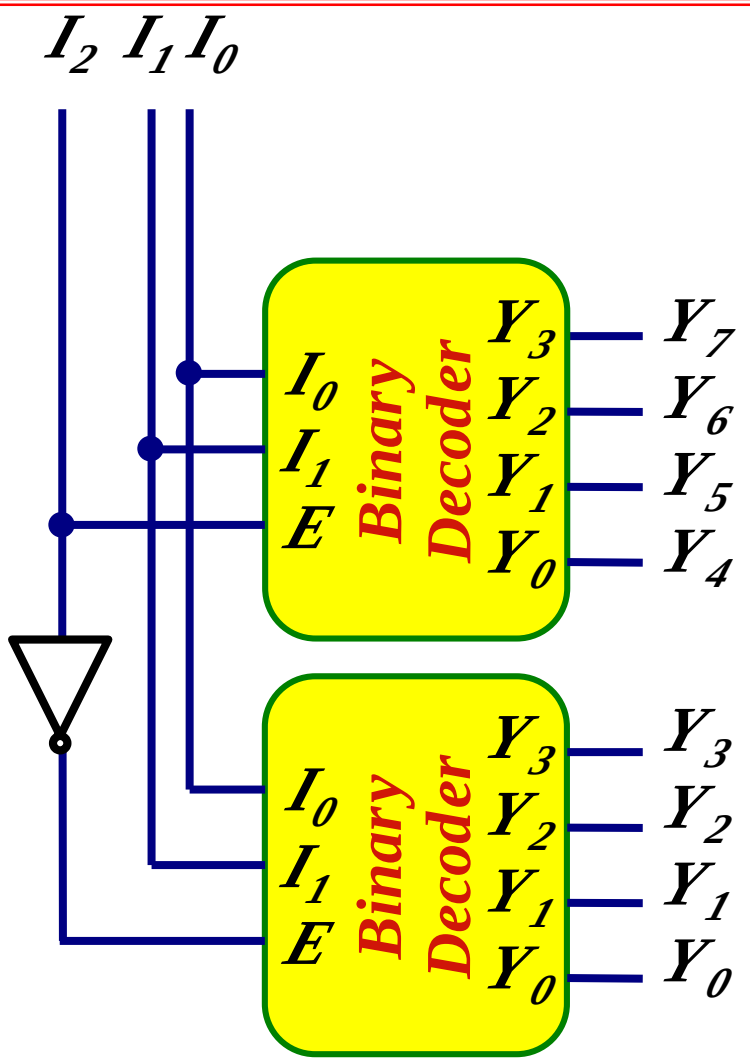
E	I_1 I_0	Y_3 Y_2 Y_1 Y_0
0	x x	0 0 0 0
1	0 0	0 0 0 1
1	0 1	0 0 1 0
1	1 0	0 1 0 0
1	1 1	1 0 0 0



Decoders

- Expansion

I_2	I_1	I_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

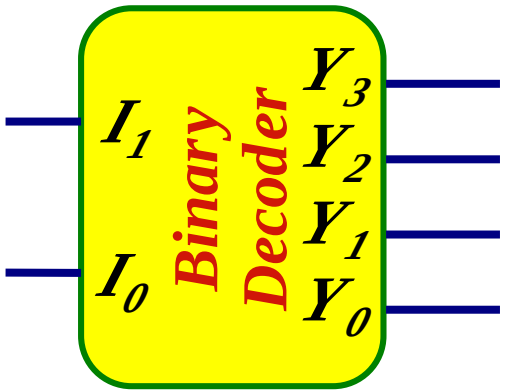


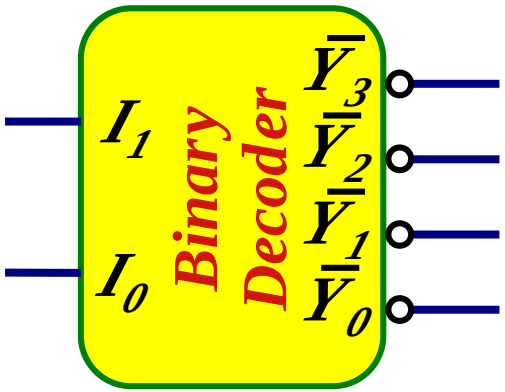
Decoders

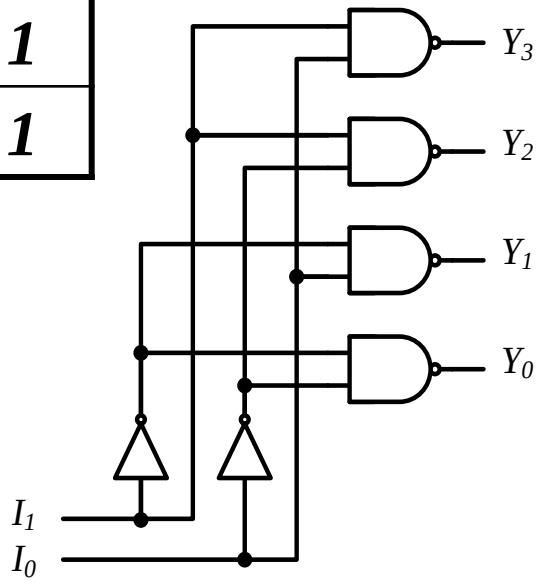
- Active-High / Active-Low

I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1



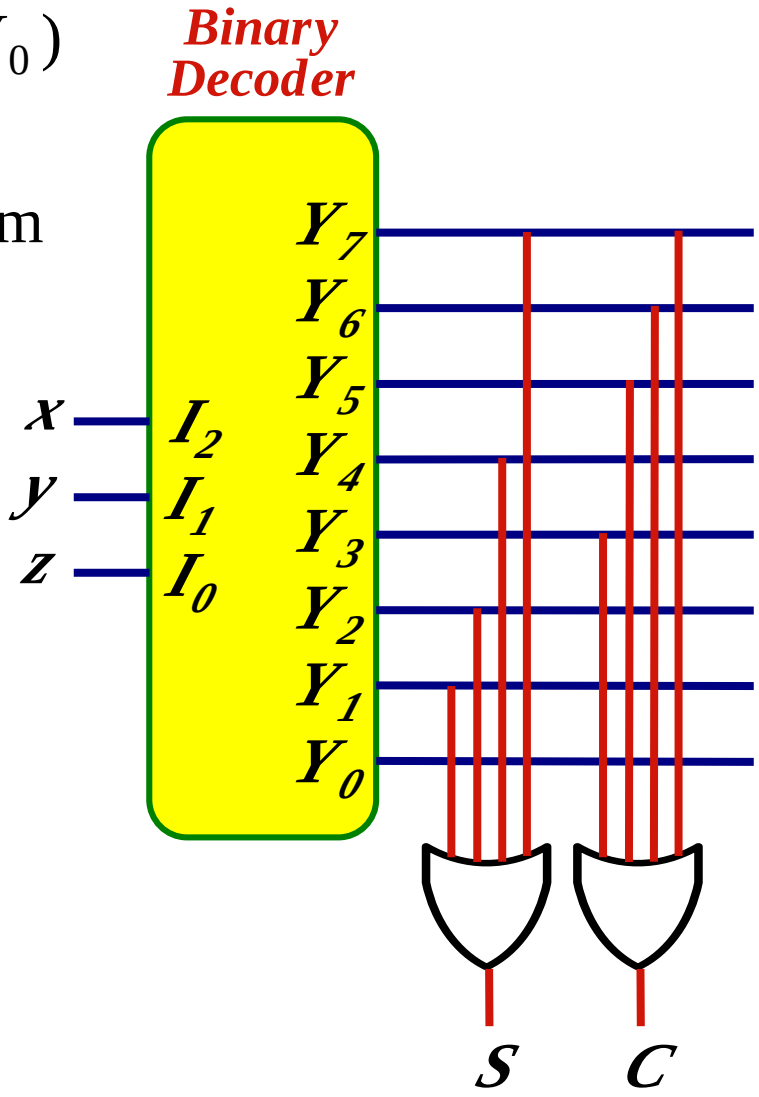




Uses NAND instead of AND

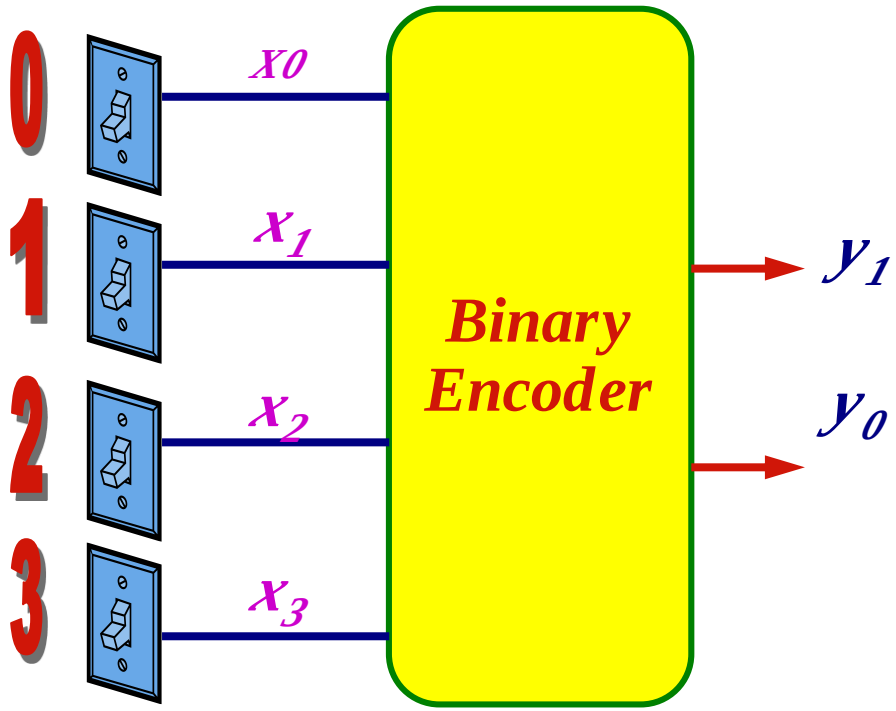
Implementation Using Decoders

- Each output is a minterm ($x`y`z` = Y_0$)
- All minterms are produced
- To implement a Boolean function sum the required minterms.
  - Example: Full Adder
    - $S(x, y, z) = \sum(1, 2, 4, 7)$
    - $C(x, y, z) = \sum(3, 5, 6, 7)$



Encoders

- Put "*Information*" into code
- Binary Encoder
  - Example: 4-to-2 Binary Encoder



Only one switch should be activated at a time

x_3 x_2 x_1 x_0	y_1 y_0
0 0 0 1	0 0
0 0 1 0	0 1
0 1 0 0	1 0
1 0 0 0	1 1

If **two** inputs are active simultaneously, the output produces an **undefined** combination

Encoders

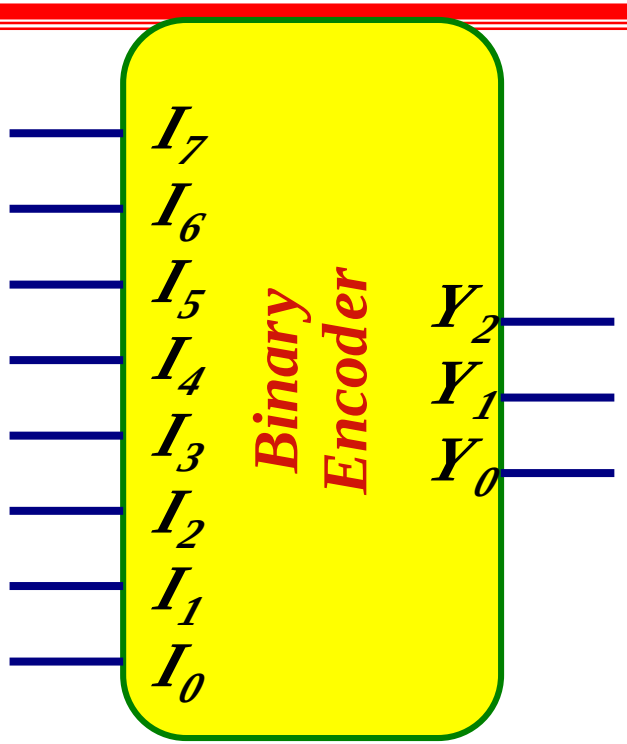
- Eight inputs Encoder (8-to-3)

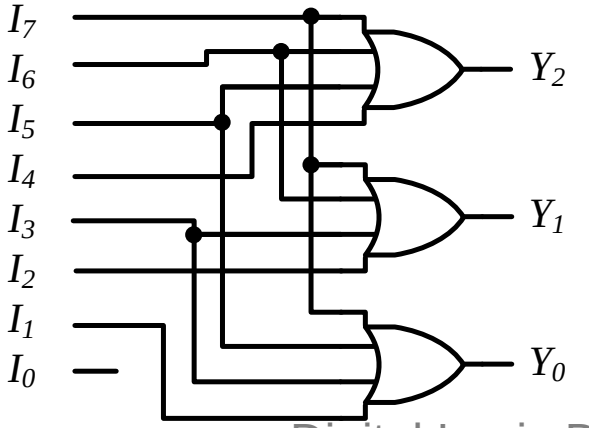
I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

$$Y_2 = I_7 + I_6 + I_5 + I_4$$

$$Y_1 = I_7 + I_6 + I_3 + I_2$$

$$Y_0 = I_7 + I_5 + I_3 + I_1$$

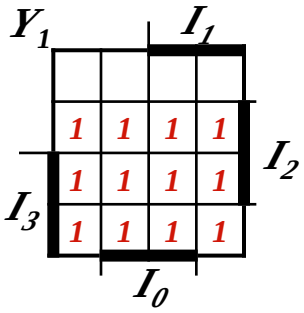




Priority Encoders

- The operation of the priority encoder is such that ***if two or more*** inputs are equa1 t o (1) at the same time, the input having the ***highest*** priority will take precedence.
- V is a ***valid*** bit indicator, that is set to 1 when one or more inputs are equal to 1

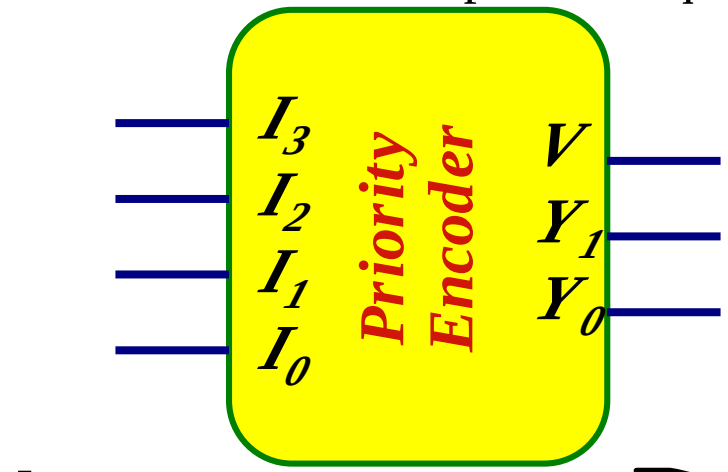
I_3	I_2	I_1	I_0	Y_1	Y_0	V
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

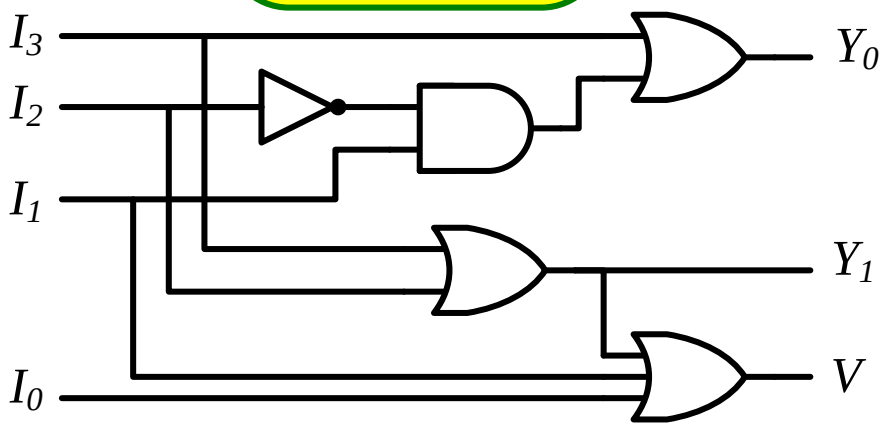


$$Y_1 = I_3 + I_2$$

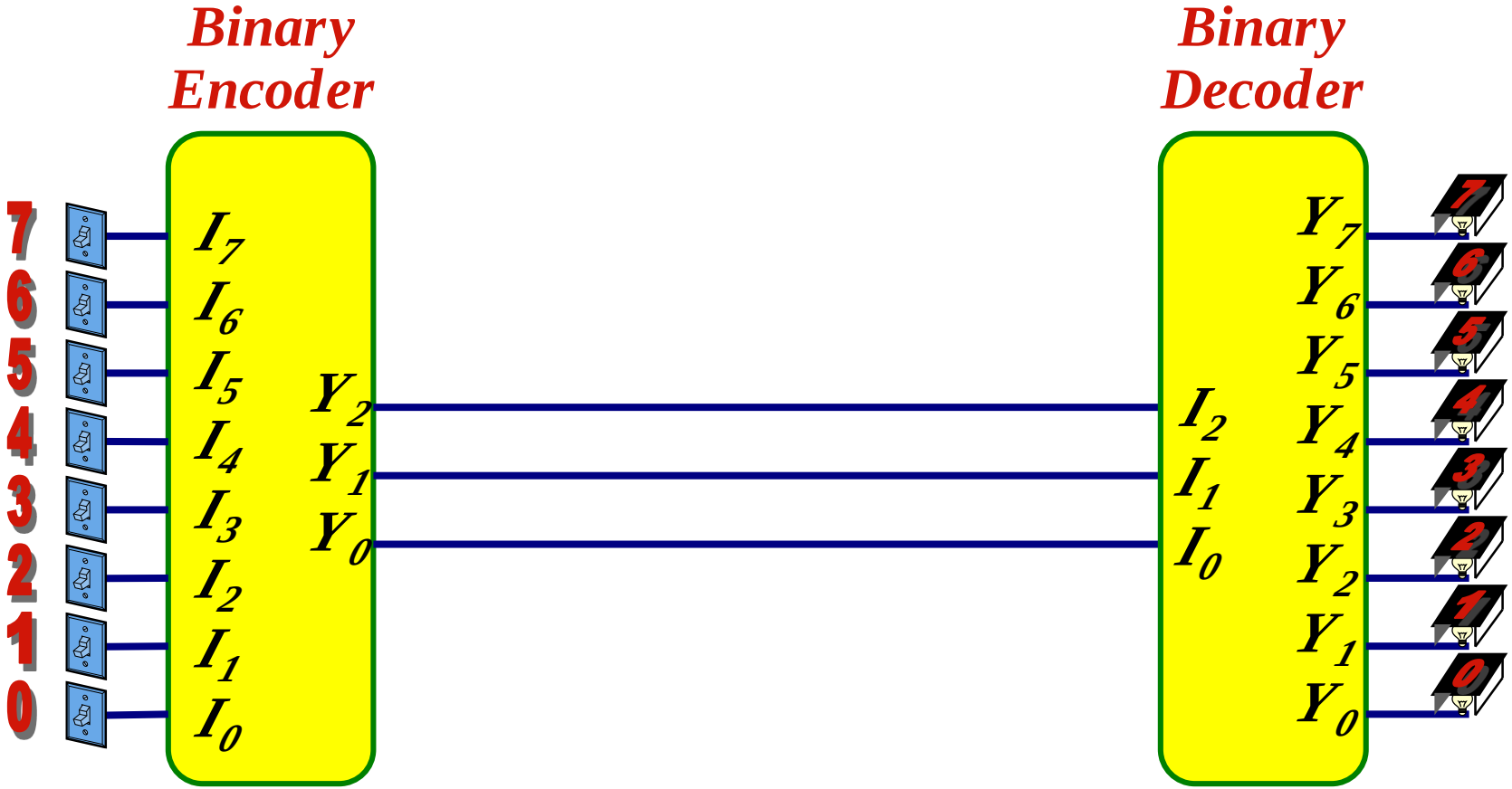
$$Y_0 = I_3 + \bar{I}_2 I_1$$

$$V = I_3 + I_2 + I_1 + I_0$$

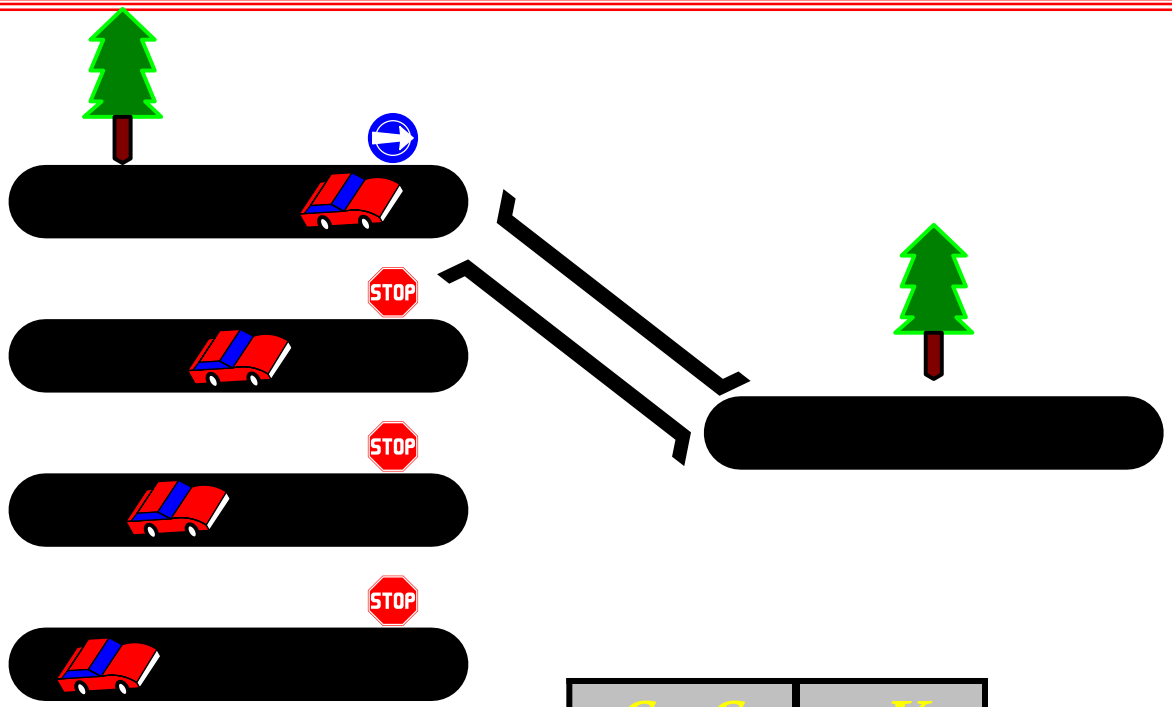




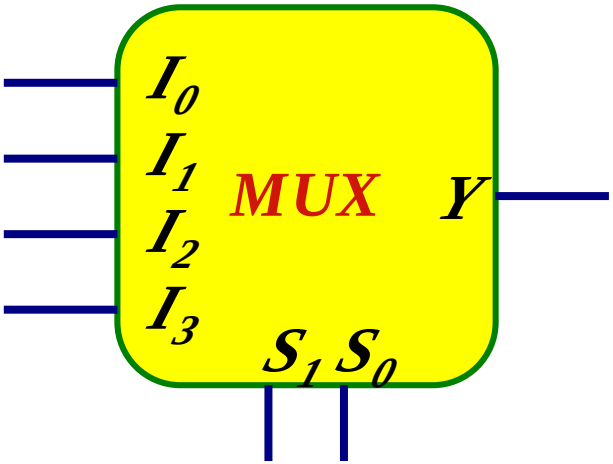
Encoder / Decoder Pairs



Multiplexers

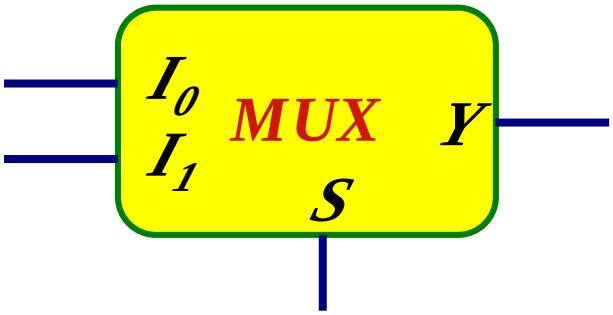


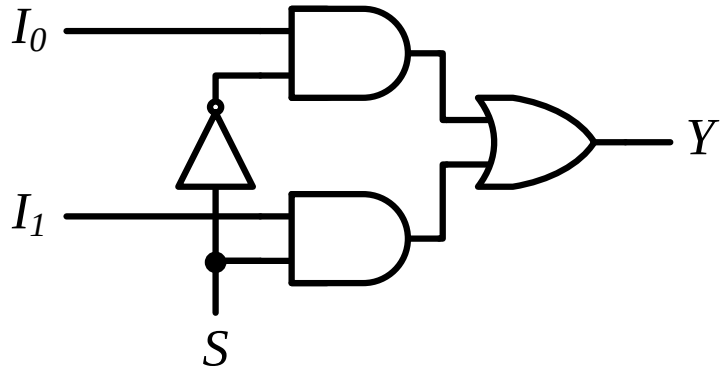
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



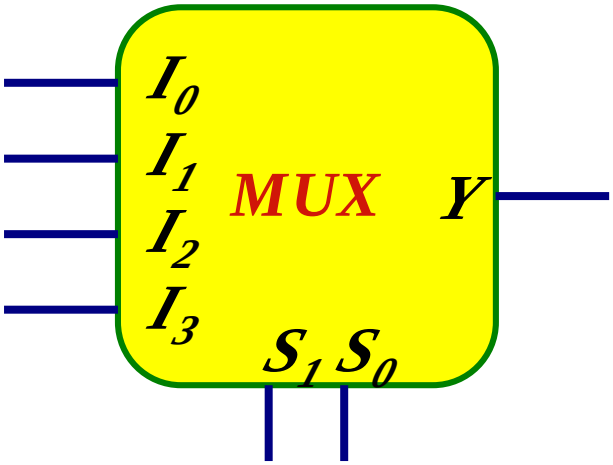
Multiplexers

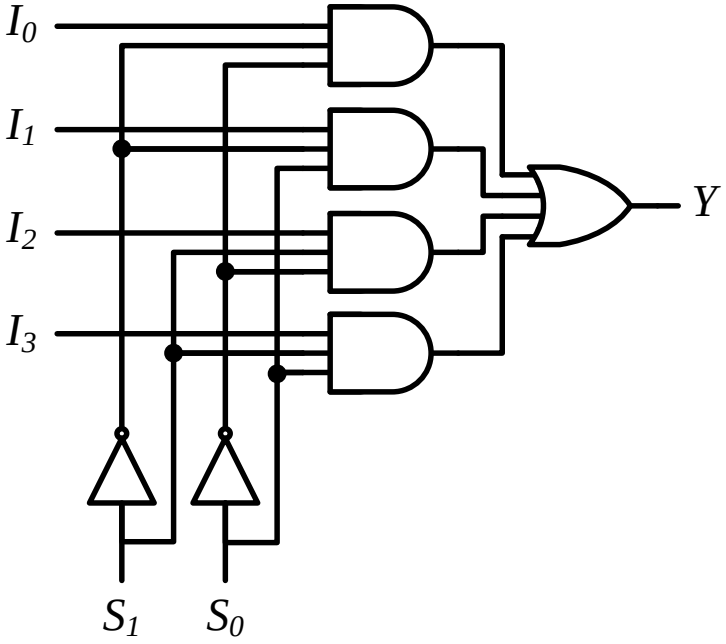
- 2-to-1 MUX





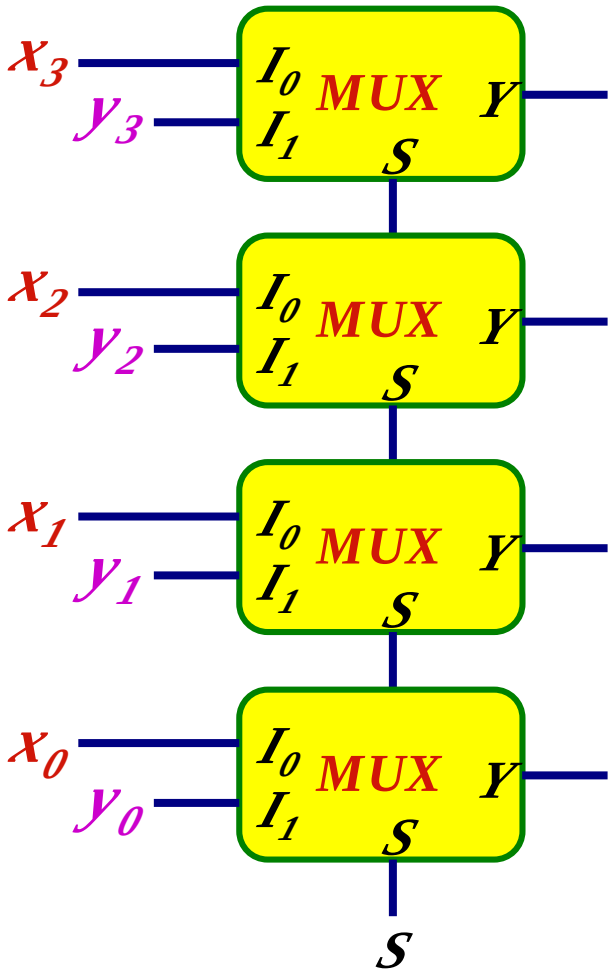
- 4-to-1 MUX

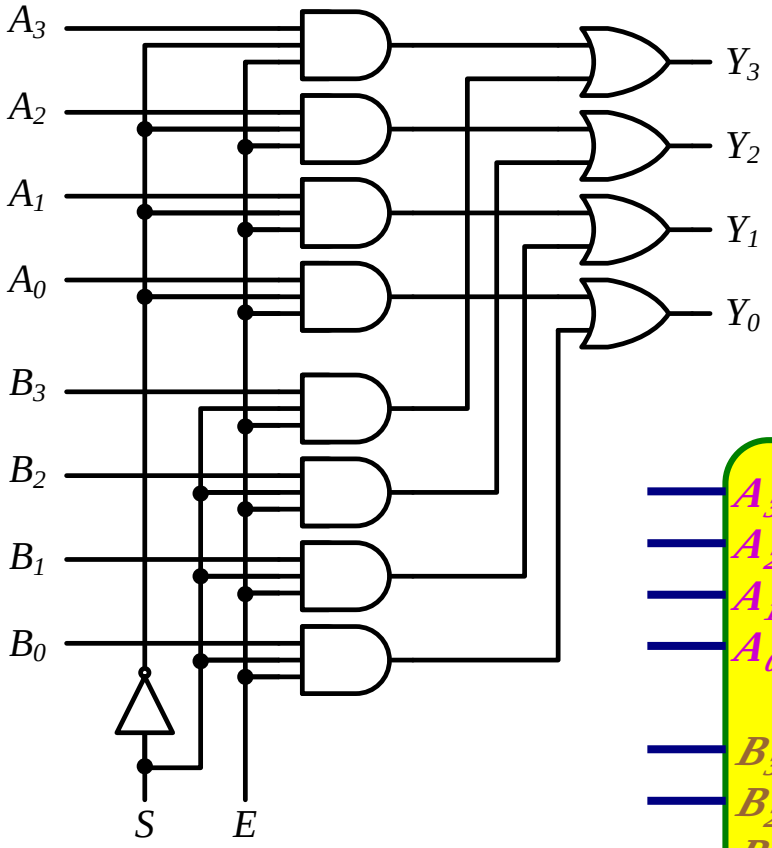


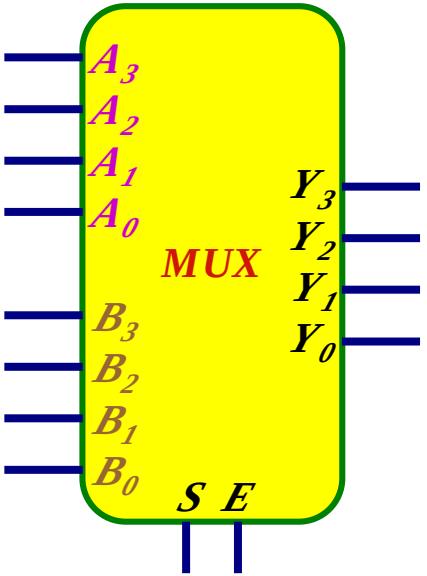


Multiplexers

- Quad 2-to 1 Mux

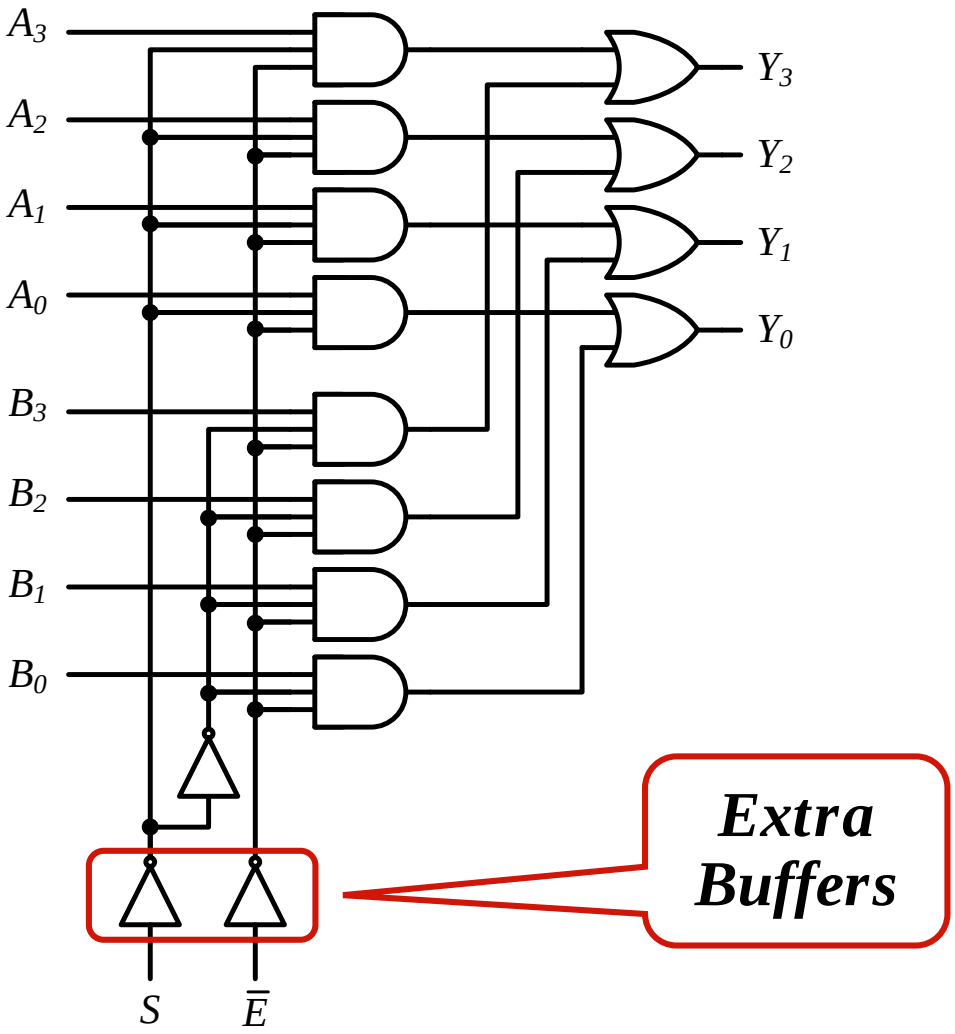


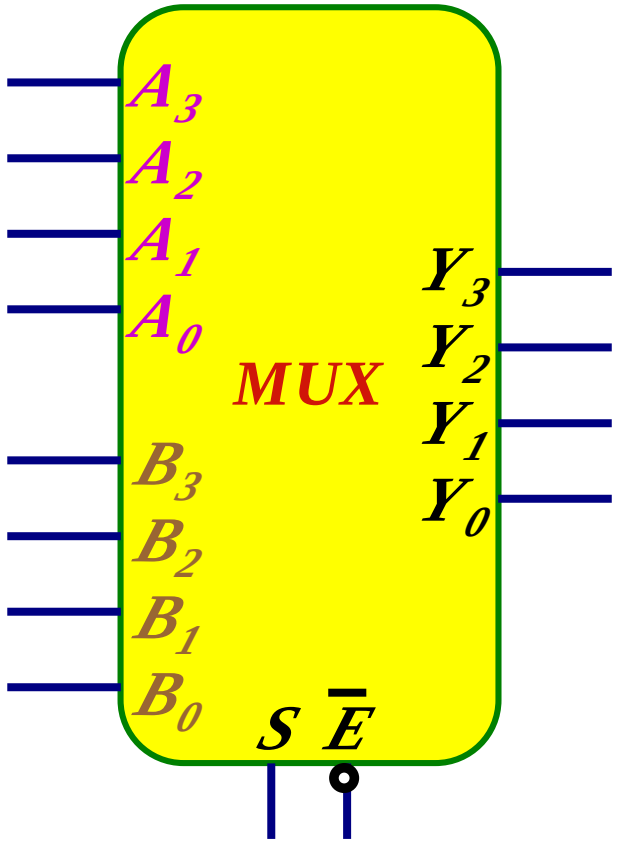




Multiplexers

- Quadruple 2-to-1 MUX

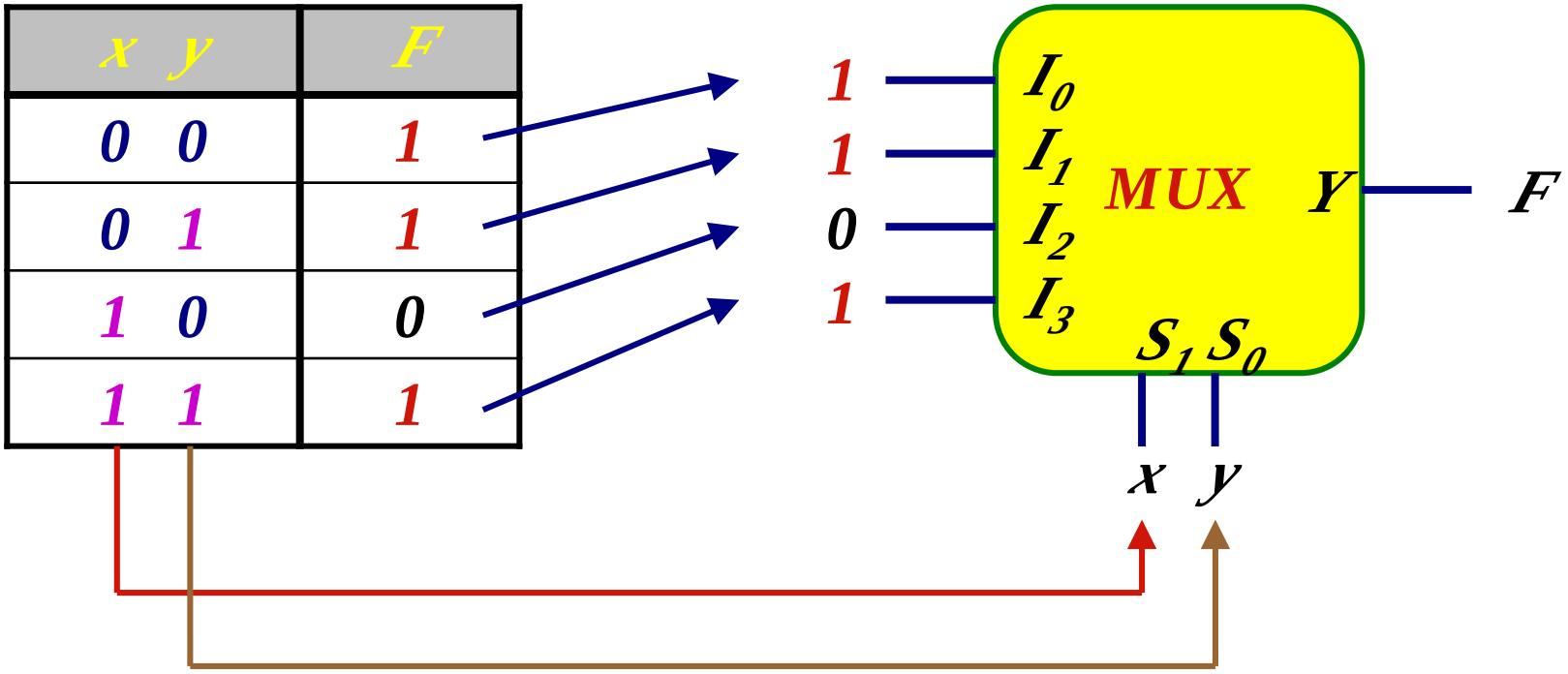




Implementation Using Multiplexers

- Example

  $F(x, y) = \sum(0, 1, 3)$

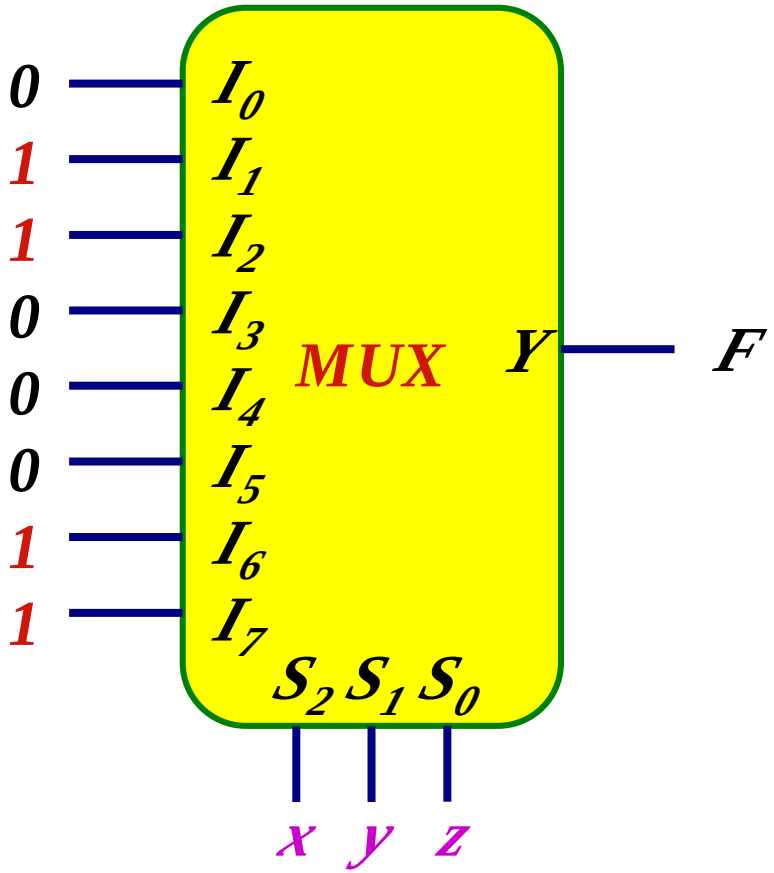


Implementation Using Multiplexers

- Example

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

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

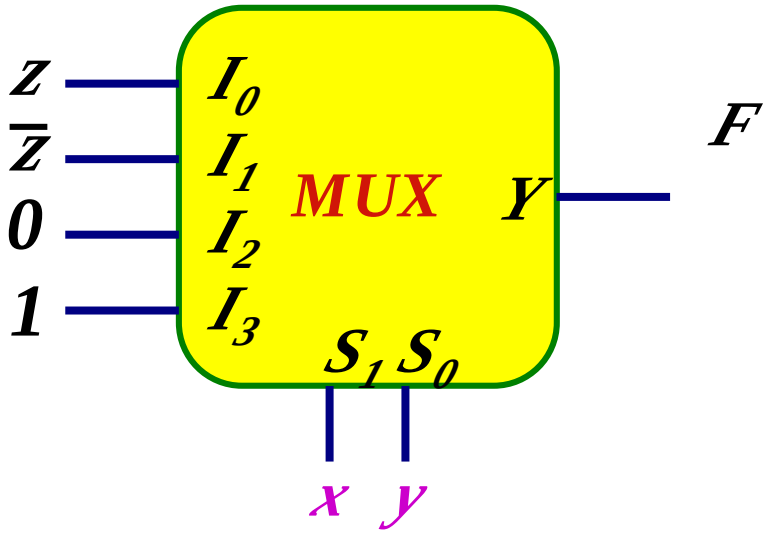


Implementation Using Multiplexers

- Example

  Implement $F(x, y, z) = \sum(1, 2, 6, 7)$ using four-to-one Mux

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = \bar{z}$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

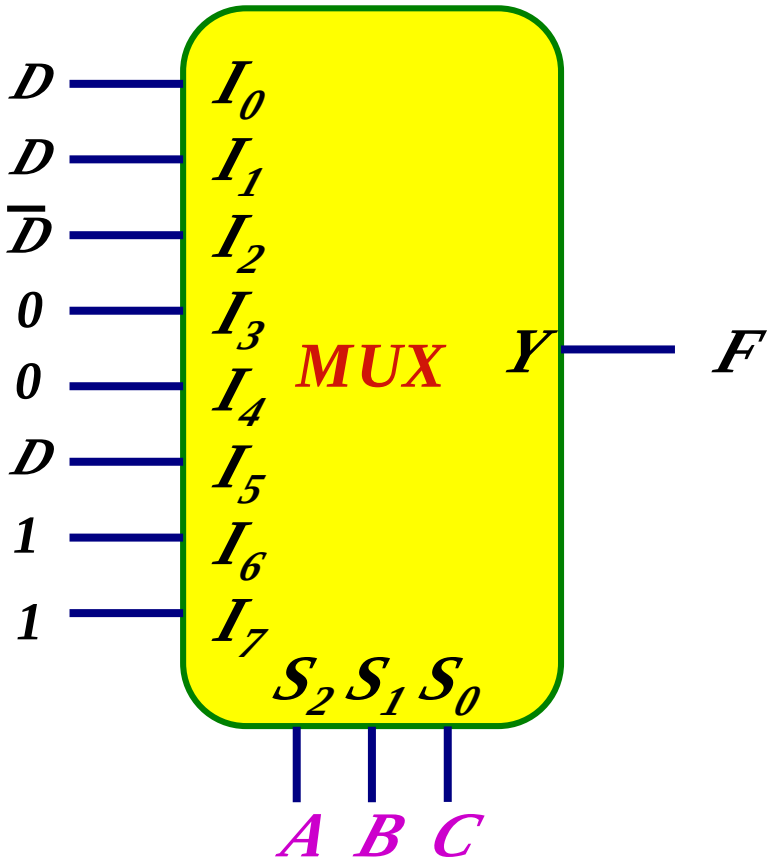


Implementation Using Multiplexers

- Example

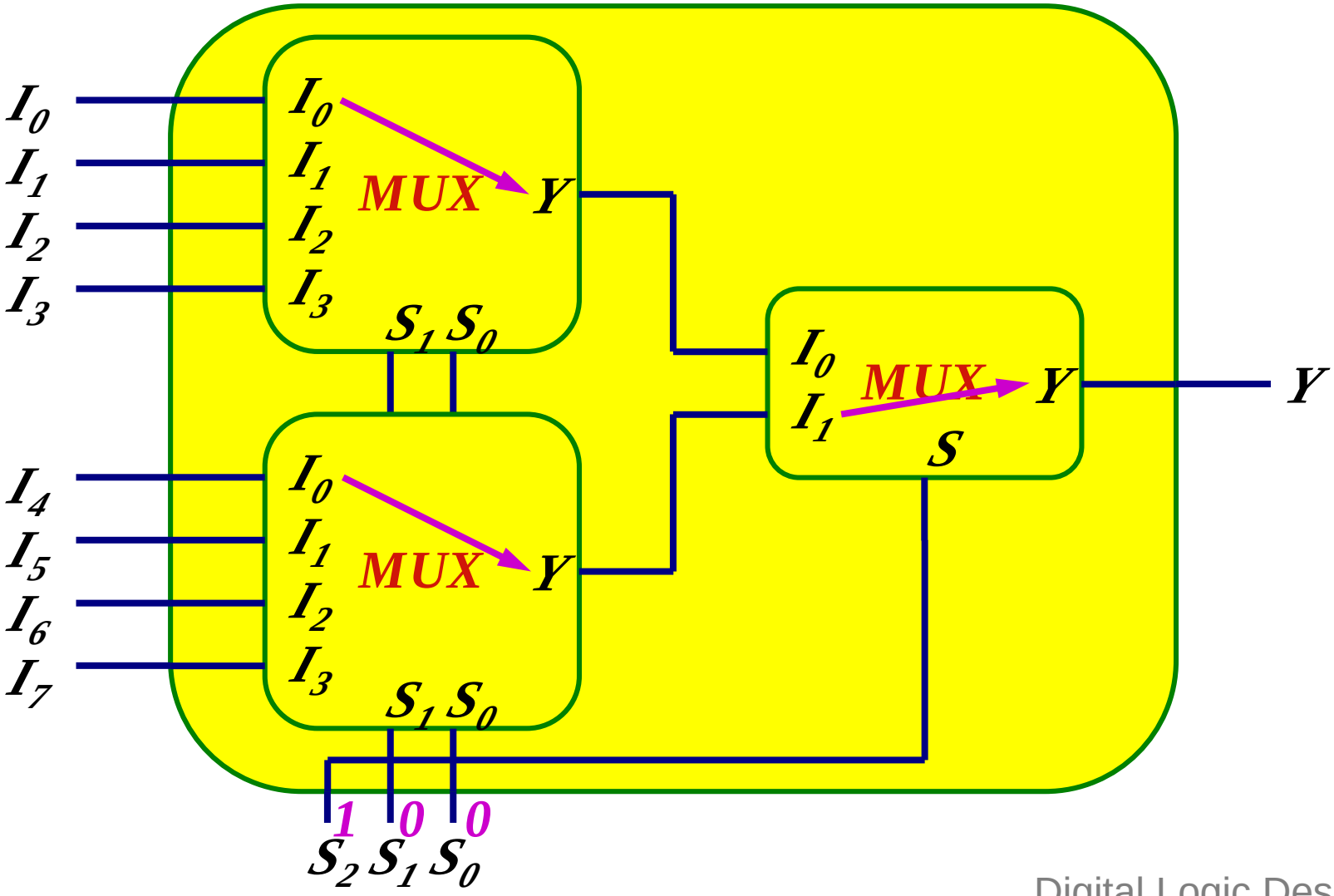
Implement $F(A, B, C, D)=\sum(1, 3, 4, 11, 12, 13, 14, 15)$ using 8-to-1 Mux.

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	$F = \overline{D}$
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	

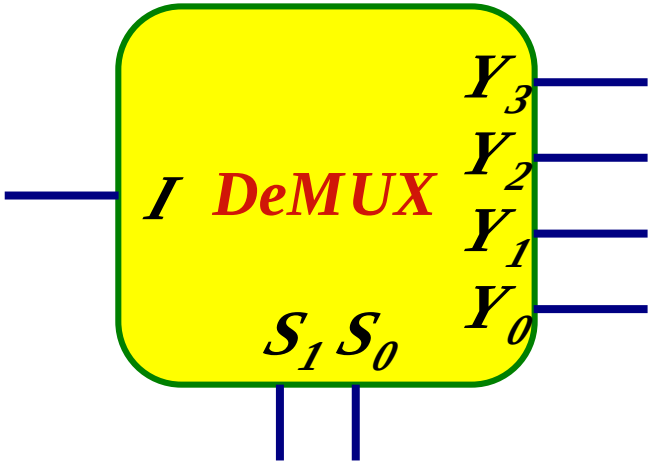


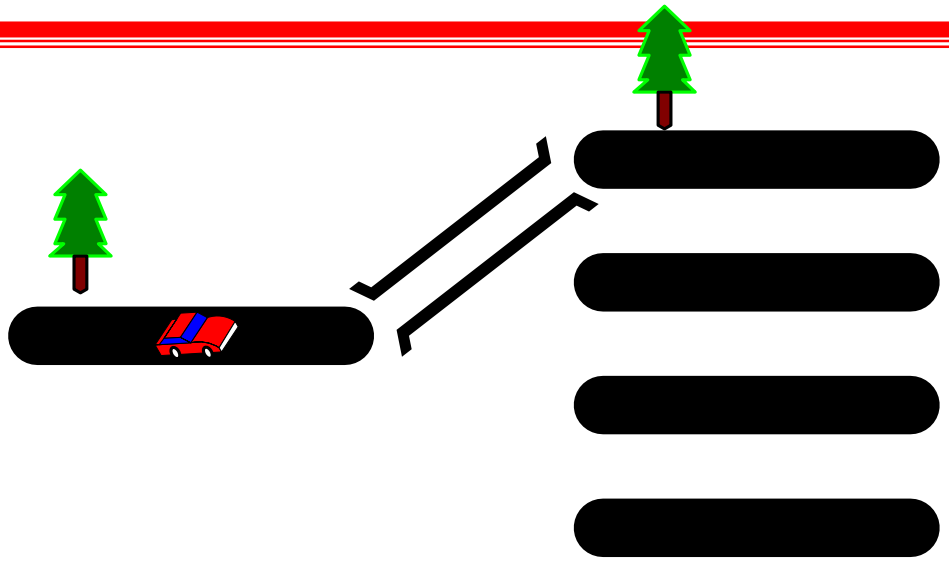
Multiplexer Expansion

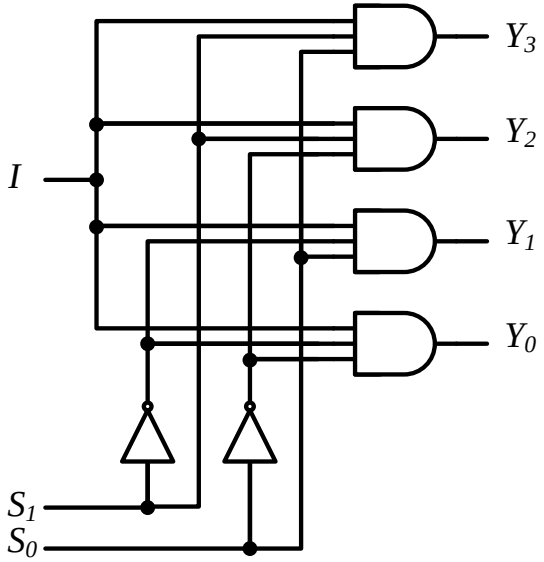
- 8-to-1 MUX using Dual 4-to-1 MUX



DeMultiplexers

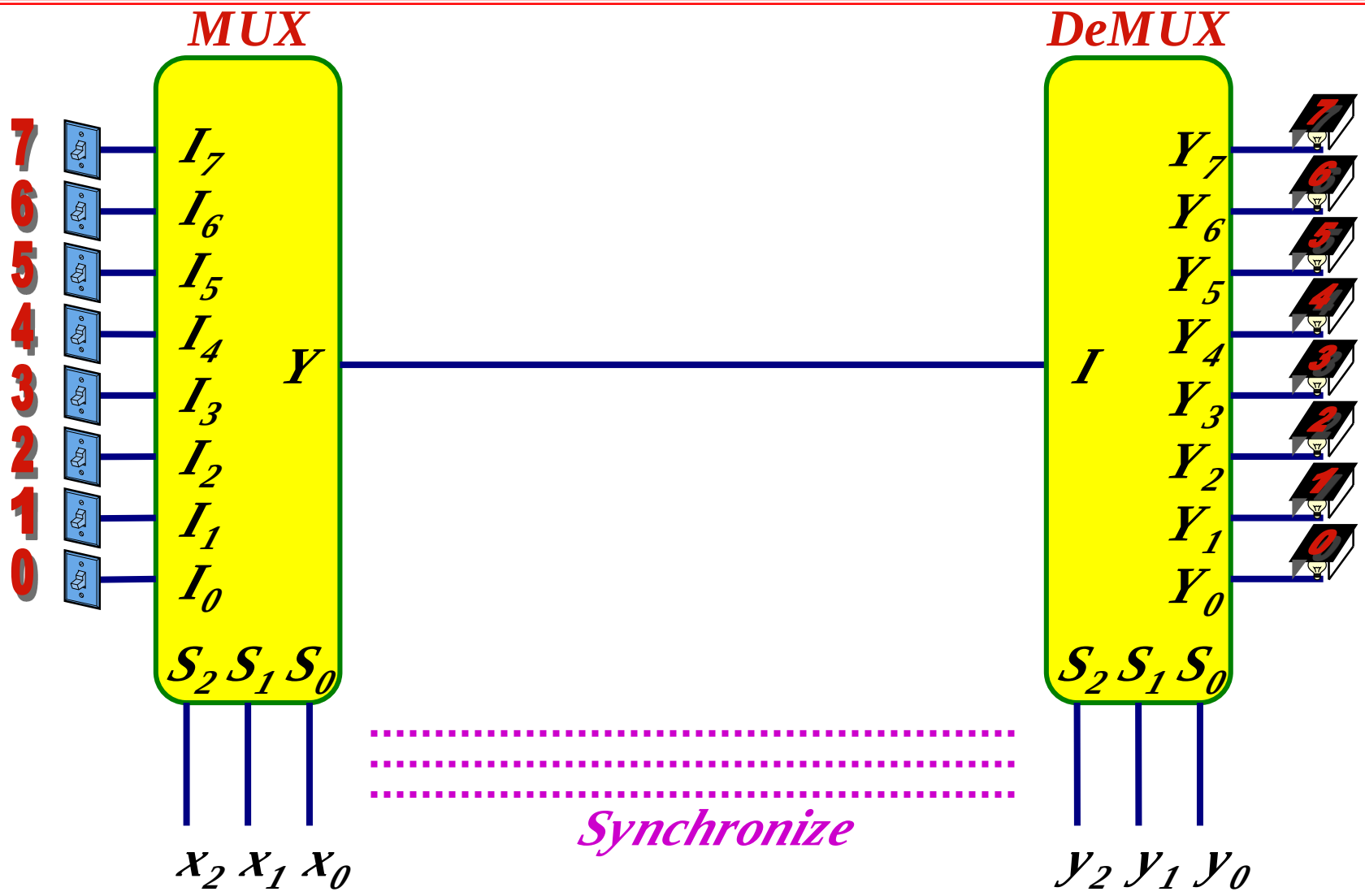




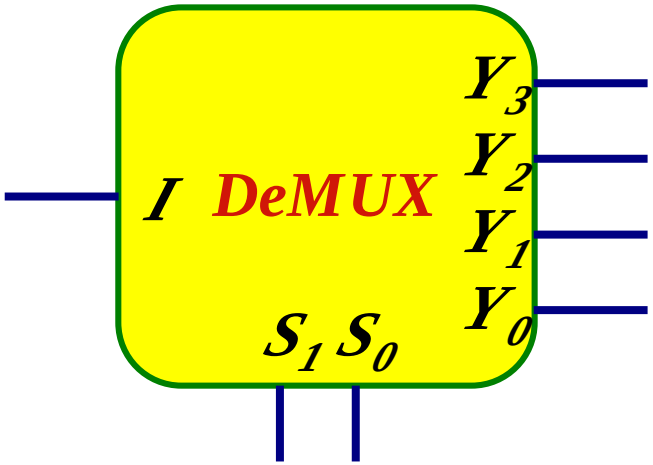


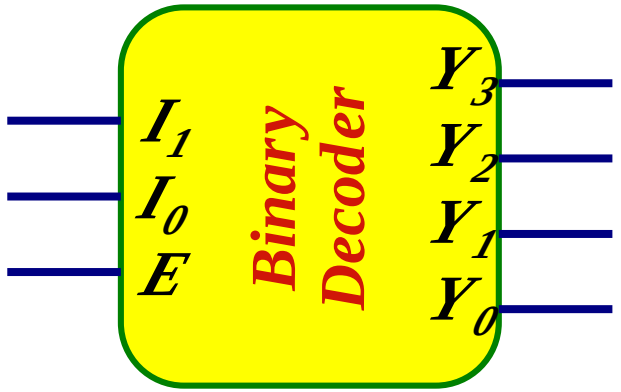
S_1 S_0	Y_3	Y_2	Y_1	Y_0
0 0	0	0	0	I
0 1	0	0	I	0
1 0	0	I	0	0
1 1	I	0	0	0

Multiplexer / DeMultiplexer Pairs



DeMultiplexers vs Decoders



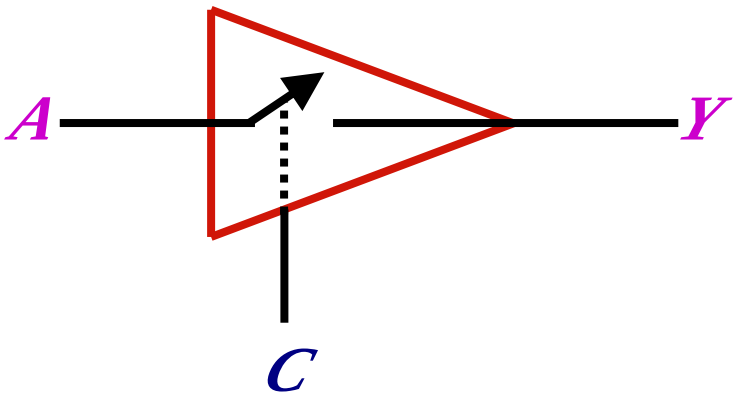


S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

E	I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

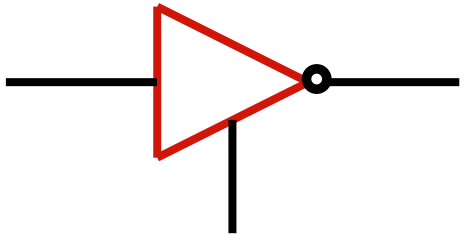
Three-State Gates

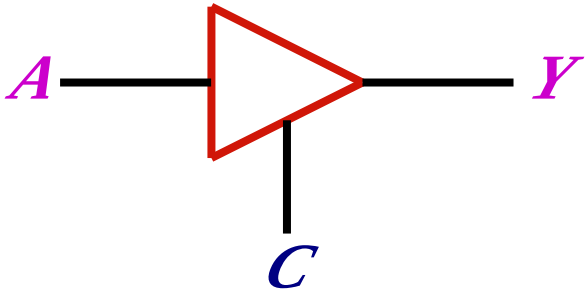
- Tri-State Buffer



C	A	Y
0	x	Hi-Z
1	0	0
1	1	1

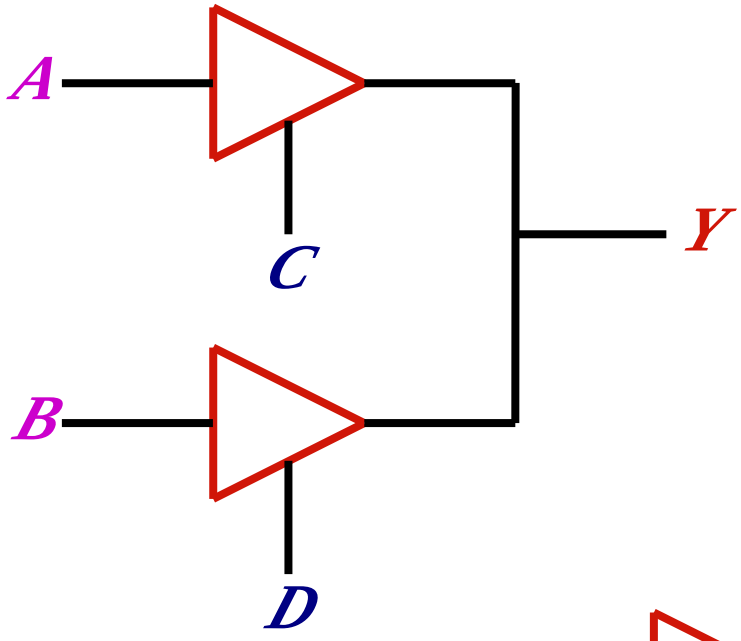
- Tri-State Inverter





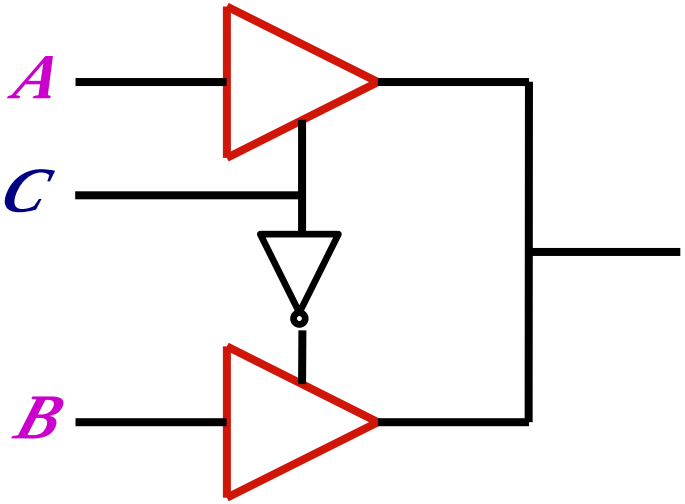
Graphic symbol for a three-state buffer

Three-State Gates



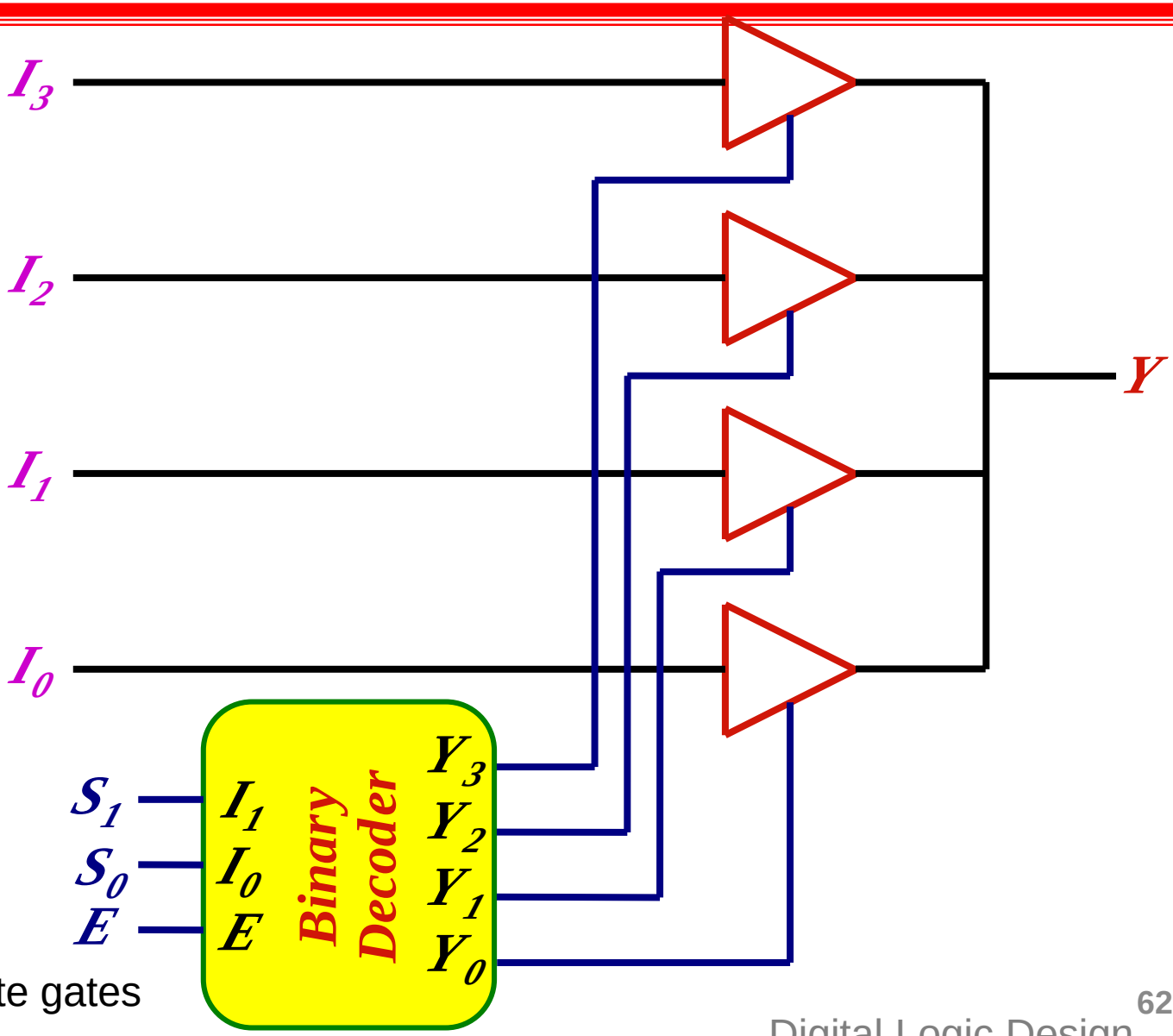
C D	Y
0 0	Hi-Z
0 1	B
1 0	A
1 1	?

Not Allowed



$$Y = \begin{cases} A & \text{if } C = 1 \\ B & \text{if } C = 0 \end{cases}$$

Three-State Gates



Decoder with three-state gates

Home Work (8)

Digital Design (**4th**)- Morris Mano-Page 175

-Problems:

4.21

4.26

4.27

4.32

4.33

تم بحمد الله